Arrays 2.0:

Extending The Scope Of The Array Abstraction

Saman Amarasinghe

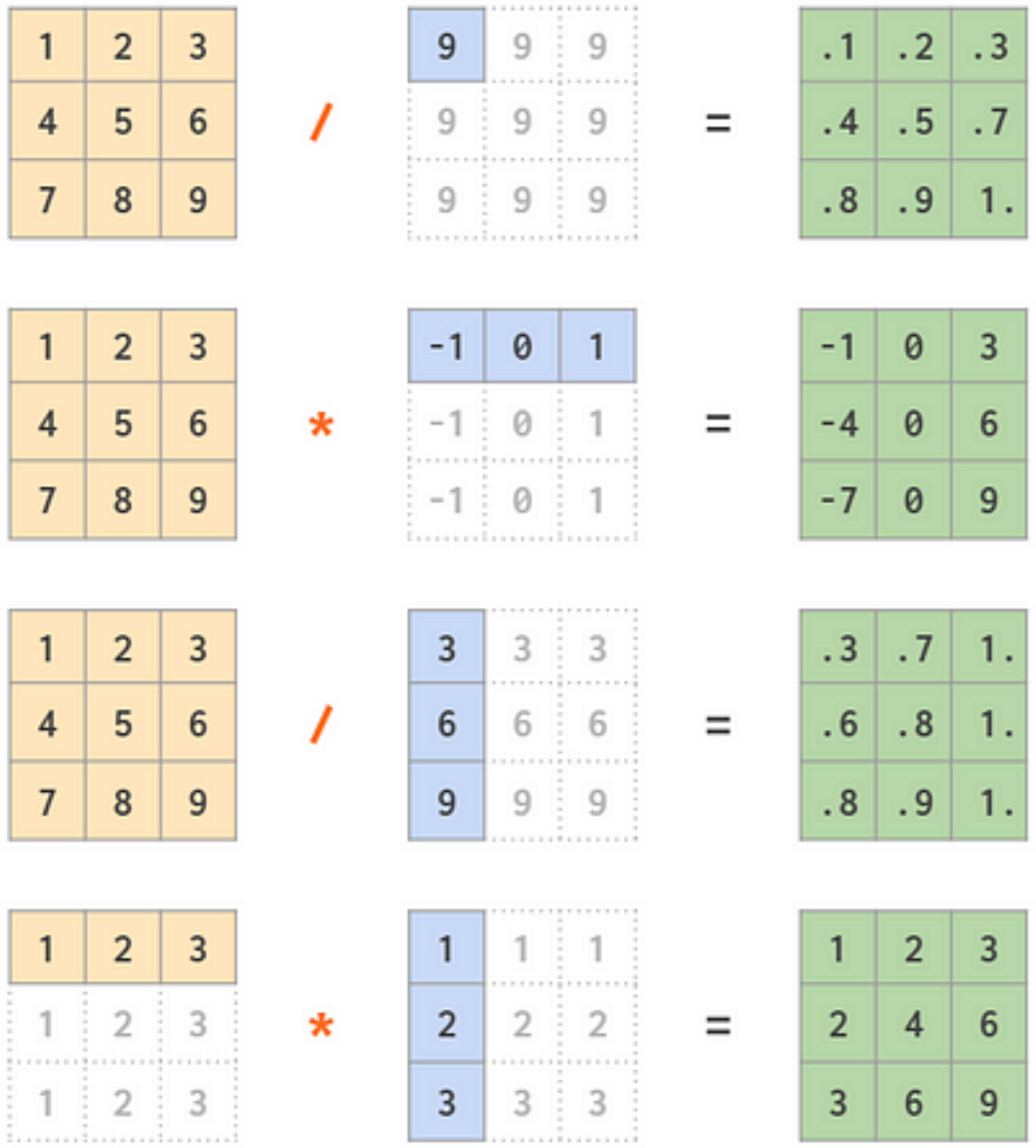
Fredrik Kjolstad (Stanford)
Jaeyeon Won (MIT)
Stephen Chou (MIT)
Rawn Henry (MIT)
Olivia Hsu (Stanford)
Rohan Yadav (Stanford)
Changwan Hong (MIT)

Willow Ahrens (MIT)
Daniel Donenfeld (MIT)
Ryan Senanayake (MIT)
Shoaib Kamil (Adobe)
David Lugato (CEA)
Charith Mendis (UIUC)
Joel Emer (MIT)

Massachusetts Institute of Technology

MIT CSAIL
compute. collaborate. create

Array Programming Is Productive



$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} / \begin{pmatrix} 9 & 9 & 9 \\ 9 & 9 & 9 \\ 9 & 9 & 9 \end{pmatrix} = \begin{pmatrix} .1 & .2 & .3 \\ .4 & .5 & .7 \\ .8 & .9 & 1. \end{pmatrix}$$

normalization

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} * \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & 3 \\ -4 & 0 & 6 \\ -7 & 0 & 9 \end{pmatrix}$$

multiplying several columns at once

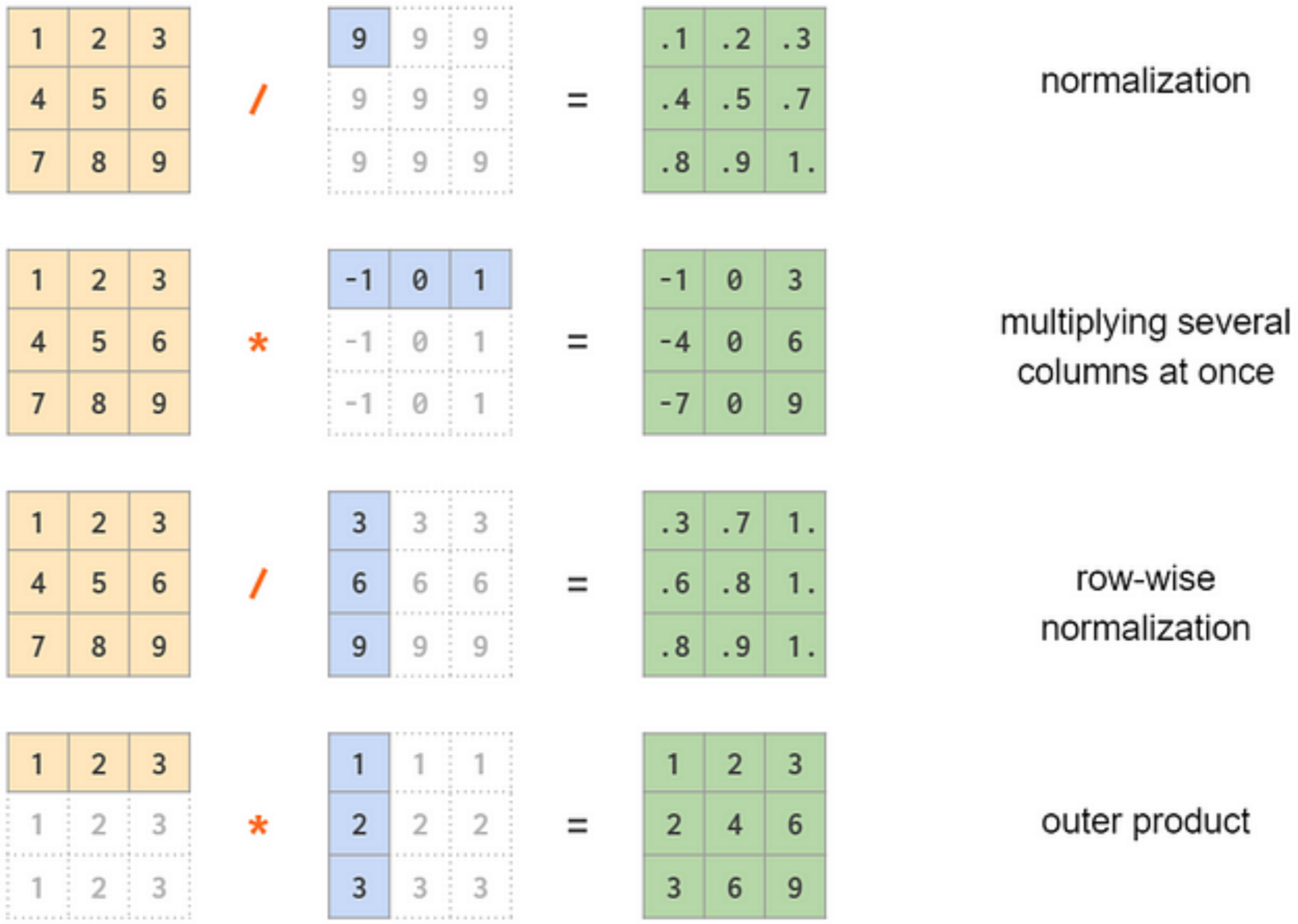
$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} / \begin{pmatrix} 3 & 3 & 3 \\ 6 & 6 & 6 \\ 9 & 9 & 9 \end{pmatrix} = \begin{pmatrix} .3 & .7 & 1. \\ .6 & .8 & 1. \\ .8 & .9 & 1. \end{pmatrix}$$

row-wise normalization

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} * \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{pmatrix}$$

outer product

Array Programming Is Productive



$$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} / \begin{bmatrix}9&9&9\\9&9&9\\9&9&9\end{bmatrix} = \begin{bmatrix}.1&.2&.3\\.4&.5&.7\\.8&.9&1.\end{bmatrix}$$

normalization

$$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} * \begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix} = \begin{bmatrix}-1&0&3\\-4&0&6\\-7&0&9\end{bmatrix}$$

multiplying several columns at once

$$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} / \begin{bmatrix}3&3&3\\6&6&6\\9&9&9\end{bmatrix} = \begin{bmatrix}.3&.7&1.\\.6&.8&1.\\.8&.9&1.\end{bmatrix}$$

row-wise normalization

$$\begin{bmatrix}1&2&3\\1&2&3\\1&2&3\end{bmatrix} * \begin{bmatrix}1&1&1\\2&2&2\\3&3&3\end{bmatrix} = \begin{bmatrix}1&2&3\\2&4&6\\3&6&9\end{bmatrix}$$

outer product

Matrix multiplication

$$c_{ik} = \sum_j a_{ij} b_{jk}$$

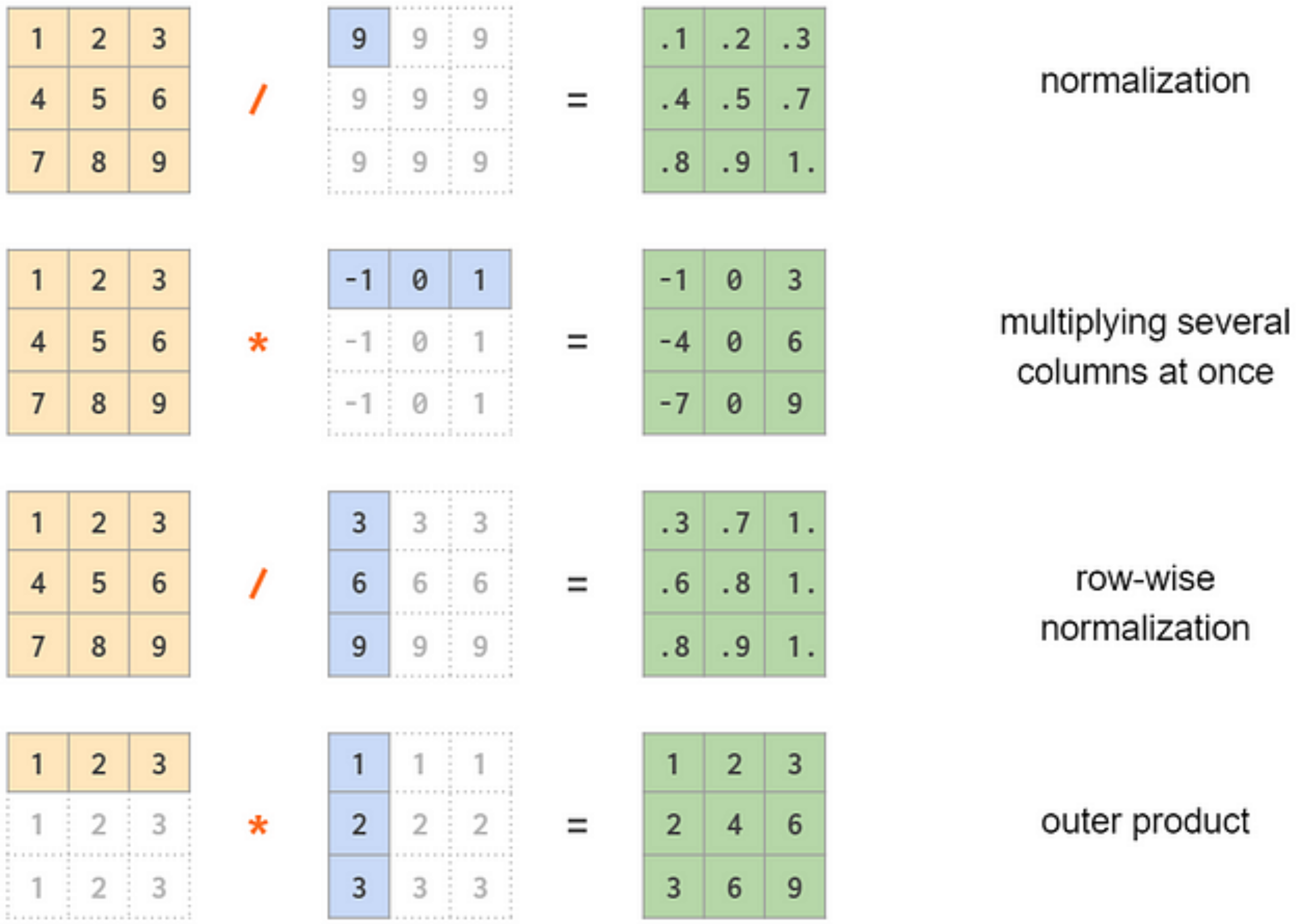
```
c = np.einsum('ij,jk->ik', a, b)
```

Tensor multiplication

$$c_{ijlm} = \sum_k a_{ijk} b_{klm}$$

```
c = np.einsum('ijk,klm->ijlm', a, b)
```

Array Programming Is Productive



$$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} / \begin{bmatrix}9&9&9\\9&9&9\\9&9&9\end{bmatrix} = \begin{bmatrix}.1&.2&.3\\.4&.5&.7\\.8&.9&1.\end{bmatrix}$$

normalization

$$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} * \begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix} = \begin{bmatrix}-1&0&3\\-4&0&6\\-7&0&9\end{bmatrix}$$

multiplying several columns at once

$$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} / \begin{bmatrix}3&3&3\\6&6&6\\9&9&9\end{bmatrix} = \begin{bmatrix}.3&.7&1.\\.6&.8&1.\\.8&.9&1.\end{bmatrix}$$

row-wise normalization

$$\begin{bmatrix}1&2&3\\1&2&3\\1&2&3\end{bmatrix} * \begin{bmatrix}1&1&1\\2&2&2\\3&3&3\end{bmatrix} = \begin{bmatrix}1&2&3\\2&4&6\\3&6&9\end{bmatrix}$$

outer product

TensorFlow

PyTorch

GRAPHBLAS

NumPy

`Matrix multiplication`

$$c_{ik} = \sum_j a_{ij} b_{jk}$$

```
c = np.einsum('ij,jk->ik', a, b)
```

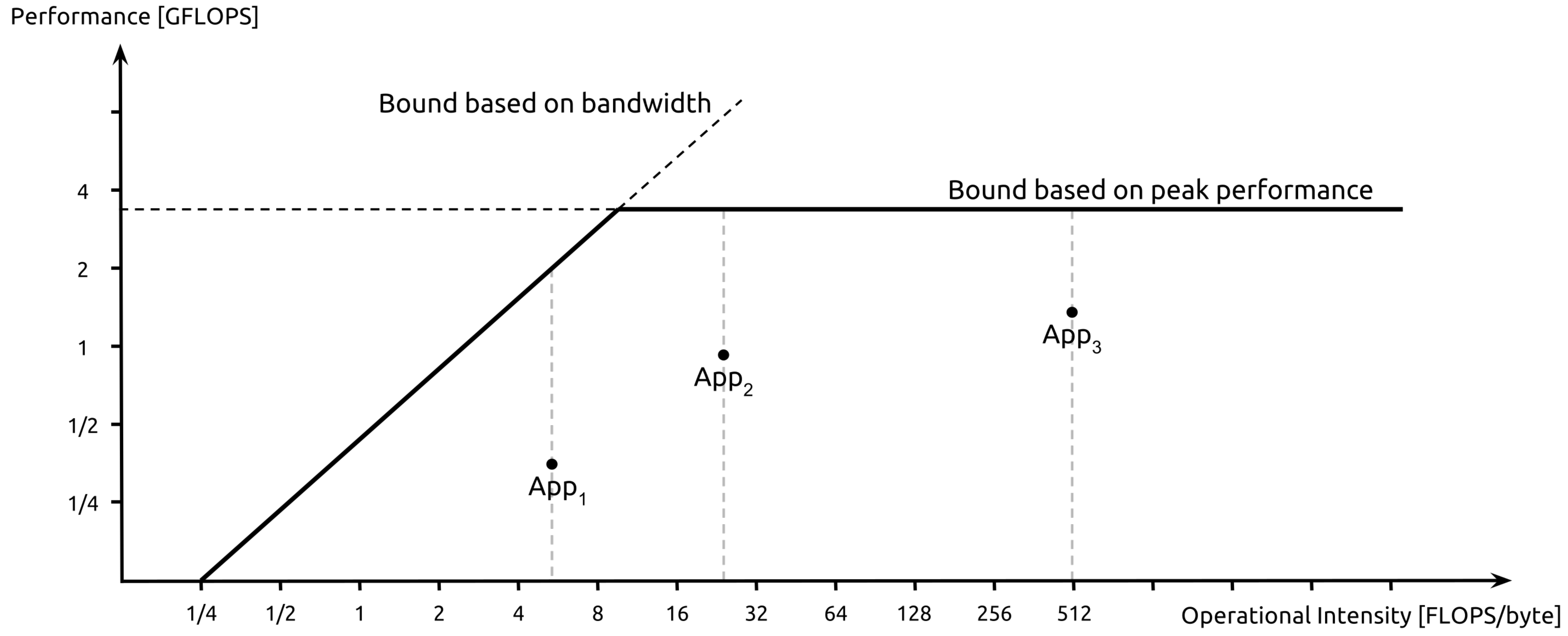
`Tensor multiplication`

$$c_{ijlm} = \sum_k a_{ijk} b_{klm}$$

```
c = np.einsum('ijk,klm->ijlm', a, b)
```

Arrays Are Fast

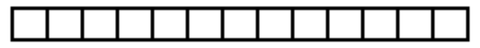
- Huge investments
  - Cache Blocking and Tiling
  - Loop unrolling
  - Vectorization
  - Multicore Parallelization
  - Communication-avoiding algorithms
- Often at 70-90 % of peak!



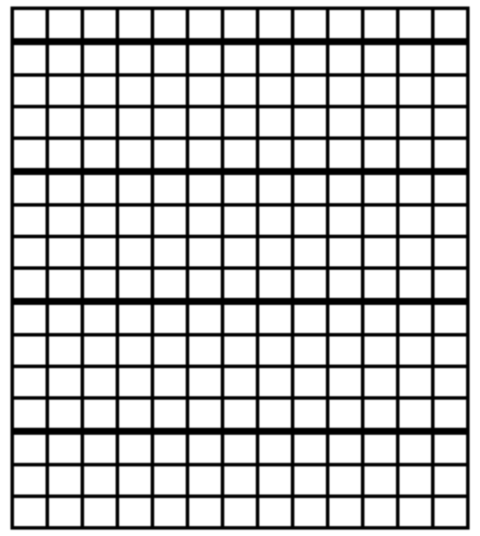
Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures.

Arrays Are The Oldest Abstraction…

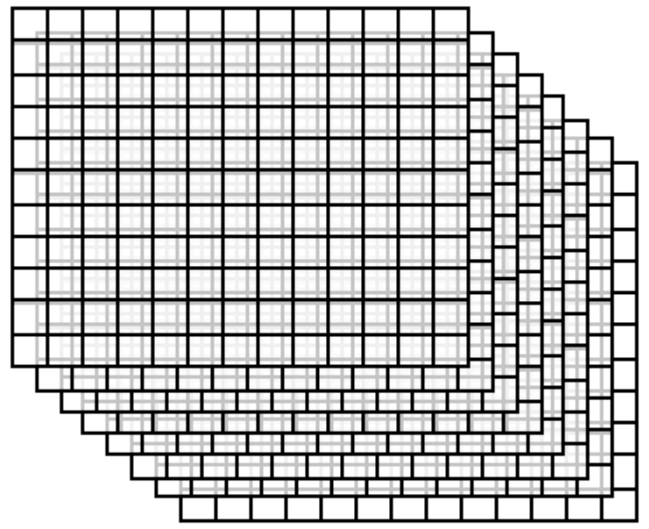
FORTRAN had Multidimensional arrays in 1957



Vector



Matrix



3-tensor

GEMM

```
*
*      Form  C := alpha*A*B + beta*C.
*
        DO 90 J = 1,N
          IF (BETA.EQ.ZERO) THEN
            DO 50 I = 1,M
              C(I,J) = ZERO
  50        CONTINUE
          ELSE IF (BETA.NE.ONE) THEN
            DO 60 I = 1,M
              C(I,J) = BETA*C(I,J)
  60        CONTINUE
          END IF
          DO 80 L = 1,K
            TEMP = ALPHA*B(L,J)
            DO 70 I = 1,M
              C(I,J) = C(I,J) + TEMP*A(I,L)
  70        CONTINUE
  80      CONTINUE
  90    CONTINUE
```

```
real :: x(14)
```

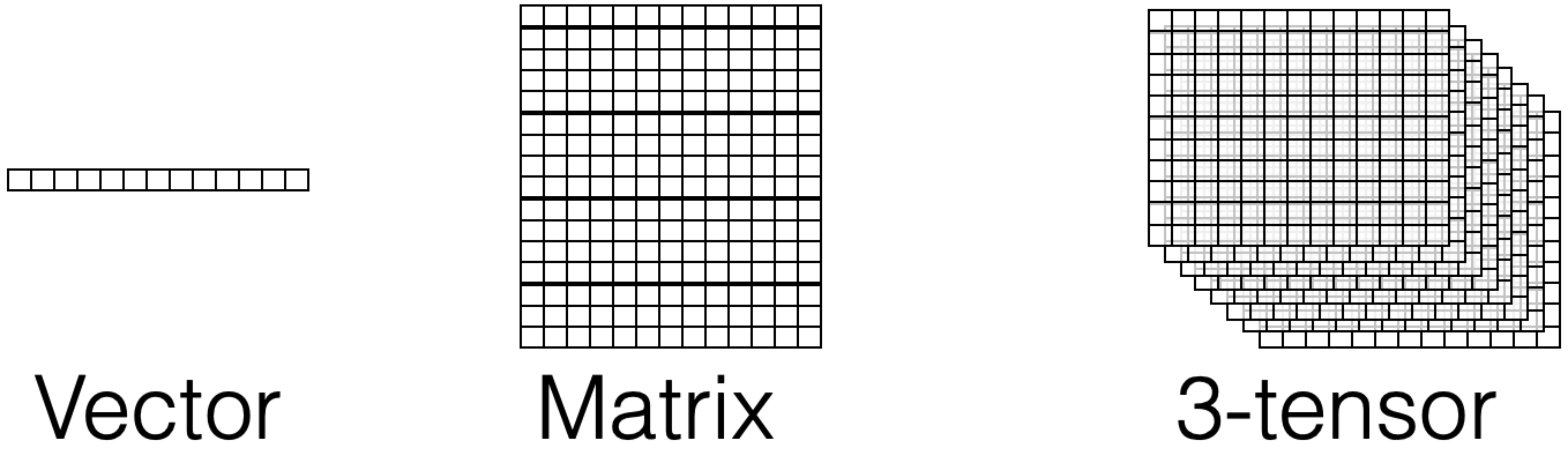
```
real :: T(8, 13, 11)
```

https://www.netlib.org/blas/#_reference_blas_version_3_11_0

```
integer, dimension(16, 14) :: A
```

… And Arrays Haven’t Changed Much Since

FORTRAN had Multidimensional arrays in 1957



```
real :: x(14)
```

```
real :: T(8, 13, 11)
```

```
integer, dimension(16, 14) :: A
```

GEMM

```
*
*          Form  C := alpha*A*B + beta*C.
*
              DO 90 J = 1,N
                  IF (BETA.EQ.ZERO) THEN
                      DO 50 I = 1,M
                          C(I,J) = ZERO
   50                 CONTINUE
                  ELSE IF (BETA.NE.ONE) THEN
                      DO 60 I = 1,M
                          C(I,J) = BETA*C(I,J)
   60                 CONTINUE
                  END IF
                  DO 80 L = 1,K
                      TEMP = ALPHA*B(L,J)
                      DO 70 I = 1,M
                          C(I,J) = C(I,J) + TEMP*A(I,L)
   70                 CONTINUE
   80             CONTINUE
   90         CONTINUE
```

https://www.netlib.org/blas/#_reference_blas_version_3_11_0

Arrays Are

- **Multi-dimensional**
- **Rectilinear**
- **Dense**
- **Integer grid**

Of points

GEMM

```
*
*     Form  C := alpha*A*B + beta*C.
*
       DO 90 J = 1,N
         IF (BETA.EQ.ZERO) THEN
           DO 50 I = 1,M
             C(I,J) = ZERO
  50       CONTINUE
         ELSE IF (BETA.NE.ONE) THEN
           DO 60 I = 1,M
             C(I,J) = BETA*C(I,J)
  60       CONTINUE
         END IF
         DO 80 L = 1,K
           TEMP = ALPHA*B(L,J)
           DO 70 I = 1,M
             C(I,J) = C(I,J) + TEMP*A(I,L)
  70       CONTINUE
  80     CONTINUE
  90   CONTINUE
```

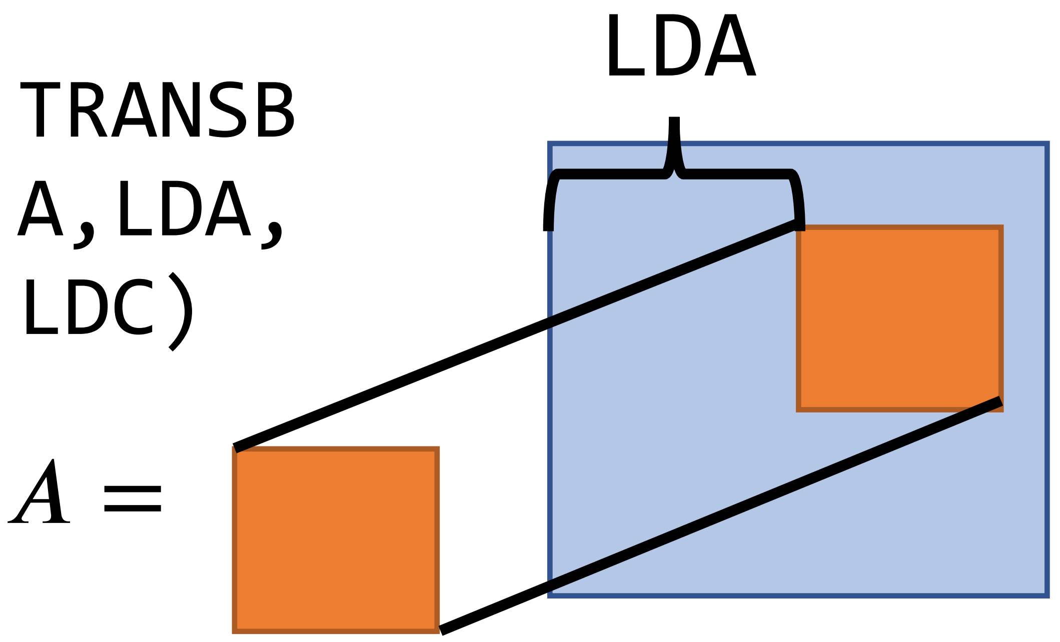
https://www.netlib.org/blas/#_reference_blas_version_3_11_0

The World Is Not Dense

The World Is Not Dense

Scientific Computing

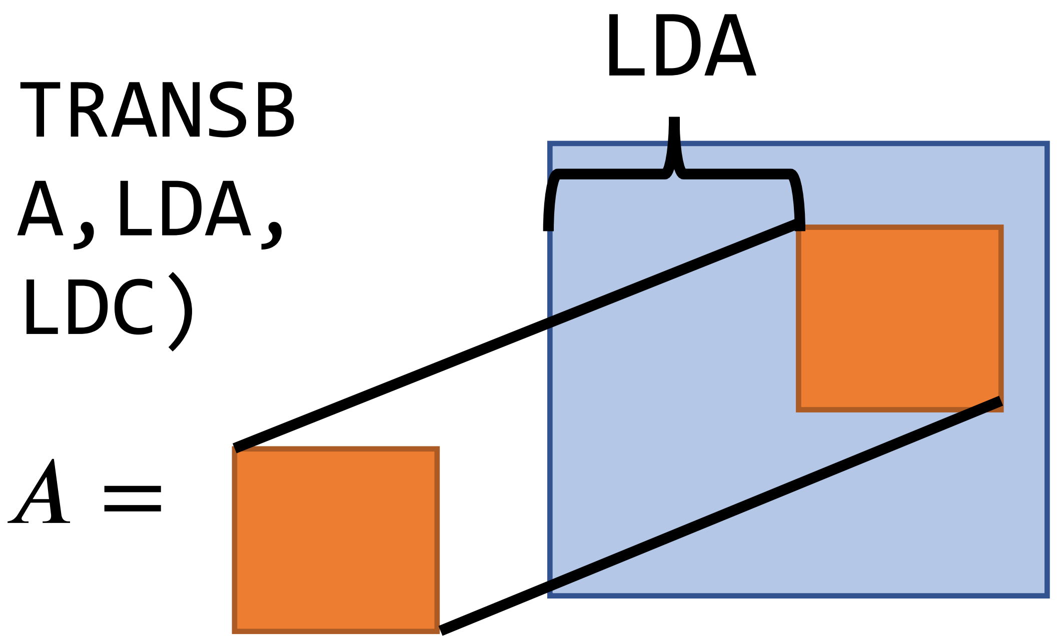
```
dgemm(TRANSA,TRANSB
,M,N,K,ALPHA,A,LDA,
B,LDB,BETA,C,LDC)
```



The World Is Not Dense

Scientific Computing

```
dgemm(TRANSA,TRANSB
,M,N,K,ALPHA,A,LDA,
B,LDB,BETA,C,LDC)
```



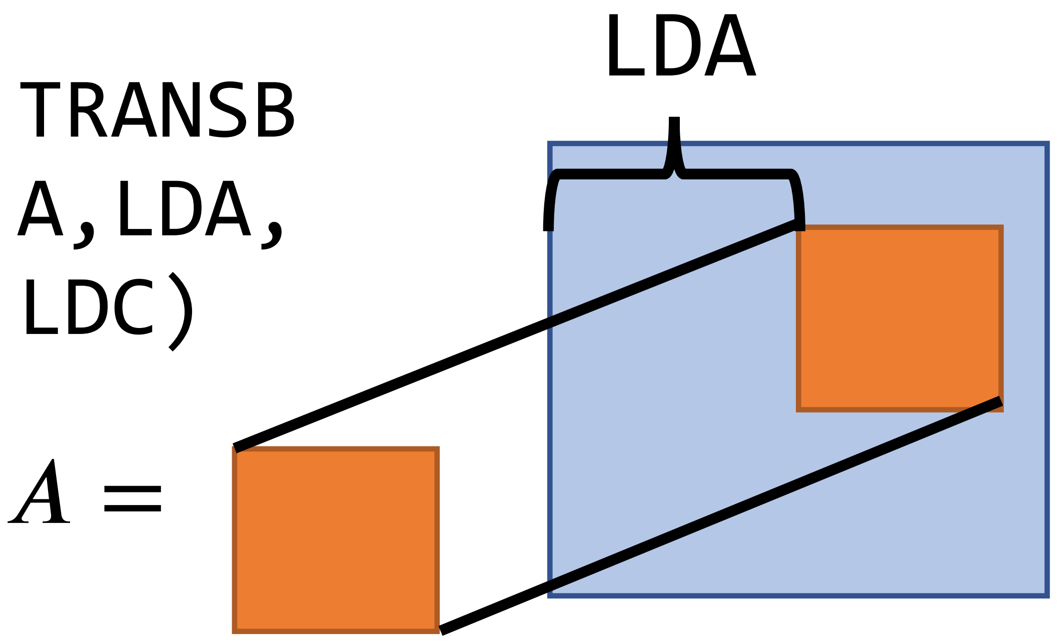
$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & u_{n-1,n} \\ 0 & & & & u_{n,n} \end{bmatrix}$$

```
dtrmm(SIDE,UPLO,
TRANSA,DIAG,M,N,
ALPHA,A,LDA,B,LDB)
```

The World Is Not Dense

Scientific Computing

`dgemm(TRANSA,TRANSB ,M,N,K,ALPHA,A,LDA, B,LDB,BETA,C,LDC)`



$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & u_{n-1,n} \\ 0 & & & & u_{n,n} \end{bmatrix}$$

`dtrmm(SIDE,UPLO, TRANSA,DIAG,M,N, ALPHA,A,LDA,B,LDB)`

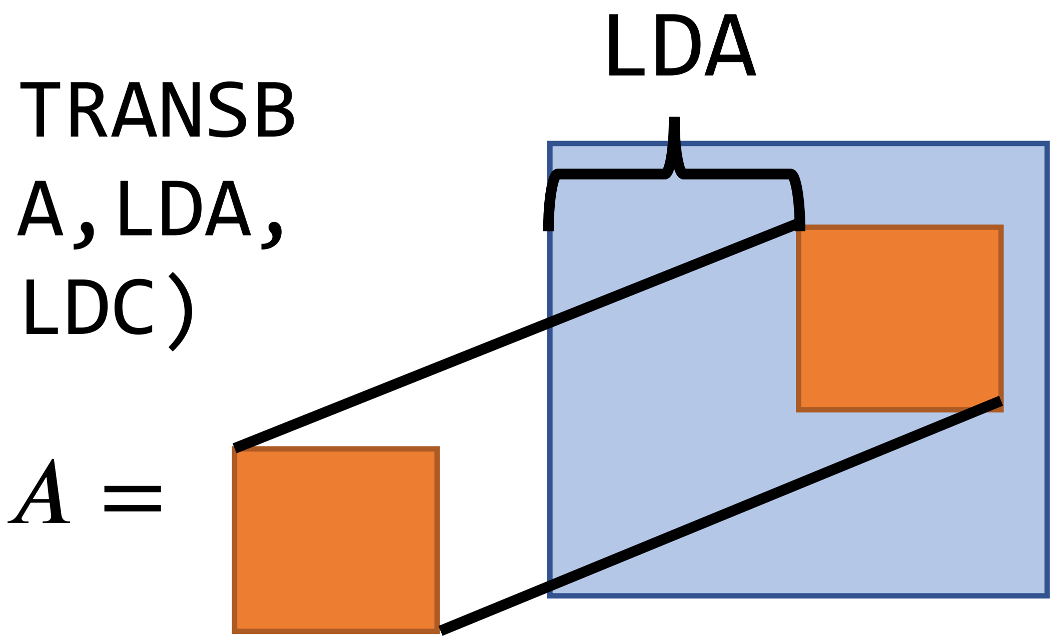
dgbmv(TRANS,M,N,KL ,KU,ALPHA,A,LDA,X,IN CX,BETA,Y,INCY)

$$\begin{bmatrix} B_{11} & B_{12} & 0 & \cdots & \cdots & 0 \\ B_{21} & B_{22} & B_{23} & \ddots & \ddots & \vdots \\ 0 & B_{32} & B_{33} & B_{34} & \ddots & \vdots \\ \vdots & \ddots & B_{43} & B_{44} & B_{45} & 0 \\ \vdots & \ddots & \ddots & B_{54} & B_{55} & B_{56} \\ 0 & \cdots & \cdots & 0 & B_{65} & B_{66} \end{bmatrix}$$

The World Is Not Dense

Scientific Computing

dgemm(TRANSA,TRANSB ,M,N,K,ALPHA,A,LDA, B,LDB,BETA,C,LDC)

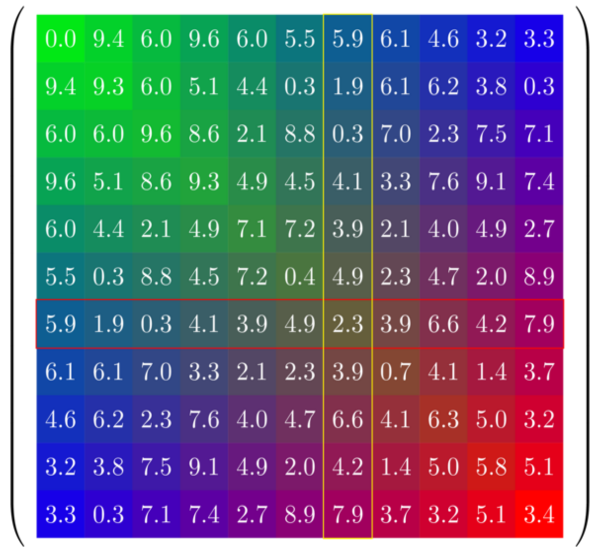


$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & u_{n-1,n} \\ 0 & & & & u_{n,n} \end{bmatrix}$$

dtrmm(SIDE,UPLO, TRANSA,DIAG,M,N, ALPHA,A,LDA,B,LDB)

dgbmv(TRANS,M,N,KL ,KU,ALPHA,A,LDA,X,IN CX,BETA,Y,INCY)

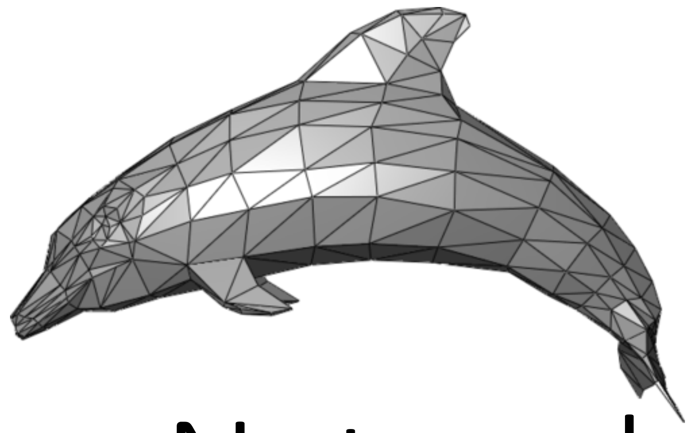
$$\begin{bmatrix} B_{11} & B_{12} & 0 & \cdots & \cdots & 0 \\ B_{21} & B_{22} & B_{23} & \ddots & \ddots & \vdots \\ 0 & B_{32} & B_{33} & B_{34} & \ddots & \vdots \\ \vdots & \ddots & B_{43} & B_{44} & B_{45} & 0 \\ \vdots & \ddots & \ddots & B_{54} & B_{55} & B_{56} \\ 0 & \cdots & \cdots & 0 & B_{65} & B_{66} \end{bmatrix}$$



$$\begin{pmatrix} 0.0 & 9.4 & 6.0 & 9.6 & 6.0 & 5.5 & 5.9 & 6.1 & 4.6 & 3.2 & 3.3 \\ 9.4 & 9.3 & 6.0 & 5.1 & 4.4 & 0.3 & 1.9 & 6.1 & 6.2 & 3.8 & 0.3 \\ 6.0 & 6.0 & 9.6 & 8.6 & 2.1 & 8.8 & 0.3 & 7.0 & 2.3 & 7.5 & 7.1 \\ 9.6 & 5.1 & 8.6 & 9.3 & 4.9 & 4.5 & 4.1 & 3.3 & 7.6 & 9.1 & 7.4 \\ 6.0 & 4.4 & 2.1 & 4.9 & 7.1 & 7.2 & 3.9 & 2.1 & 4.0 & 4.9 & 2.7 \\ 5.5 & 0.3 & 8.8 & 4.5 & 7.2 & 0.4 & 4.9 & 2.3 & 4.7 & 2.0 & 8.9 \\ 5.9 & 1.9 & 0.3 & 4.1 & 3.9 & 4.9 & 2.3 & 3.9 & 6.6 & 4.2 & 7.9 \\ 6.1 & 6.1 & 7.0 & 3.3 & 2.1 & 2.3 & 3.9 & 0.7 & 4.1 & 1.4 & 3.7 \\ 4.6 & 6.2 & 2.3 & 7.6 & 4.0 & 4.7 & 6.6 & 4.1 & 6.3 & 5.0 & 3.2 \\ 3.2 & 3.8 & 7.5 & 9.1 & 4.9 & 2.0 & 4.2 & 1.4 & 5.0 & 5.8 & 5.1 \\ 3.3 & 0.3 & 7.1 & 7.4 & 2.7 & 8.9 & 7.9 & 3.7 & 3.2 & 5.1 & 3.4 \end{pmatrix}$$

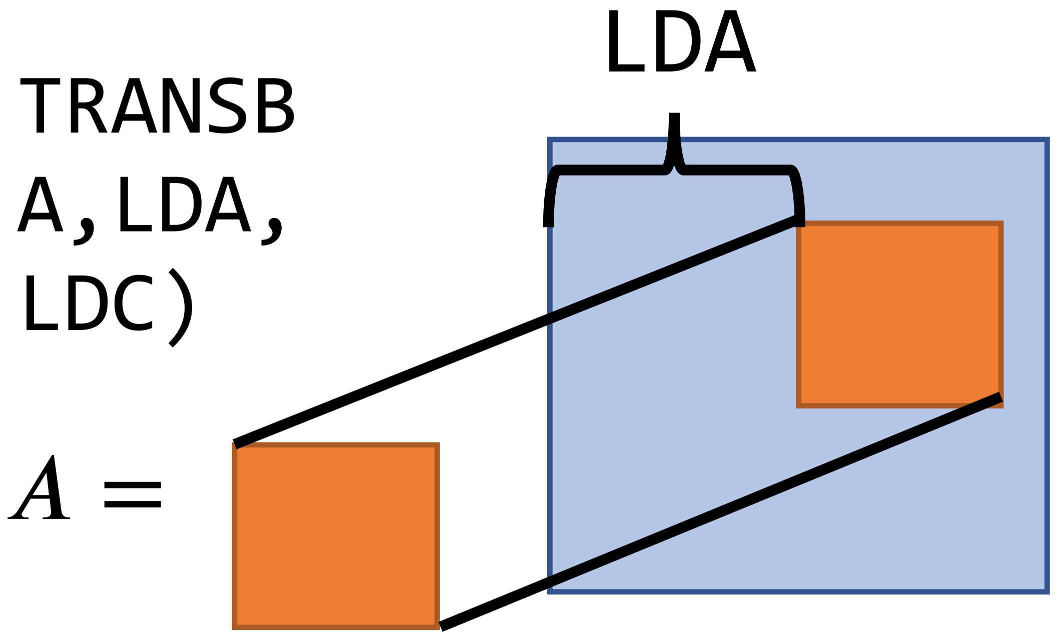
dsymm(SIDE,UPLO, M,N,ALPHA,A,LDA, B,LDB,BETA,C,LDC)

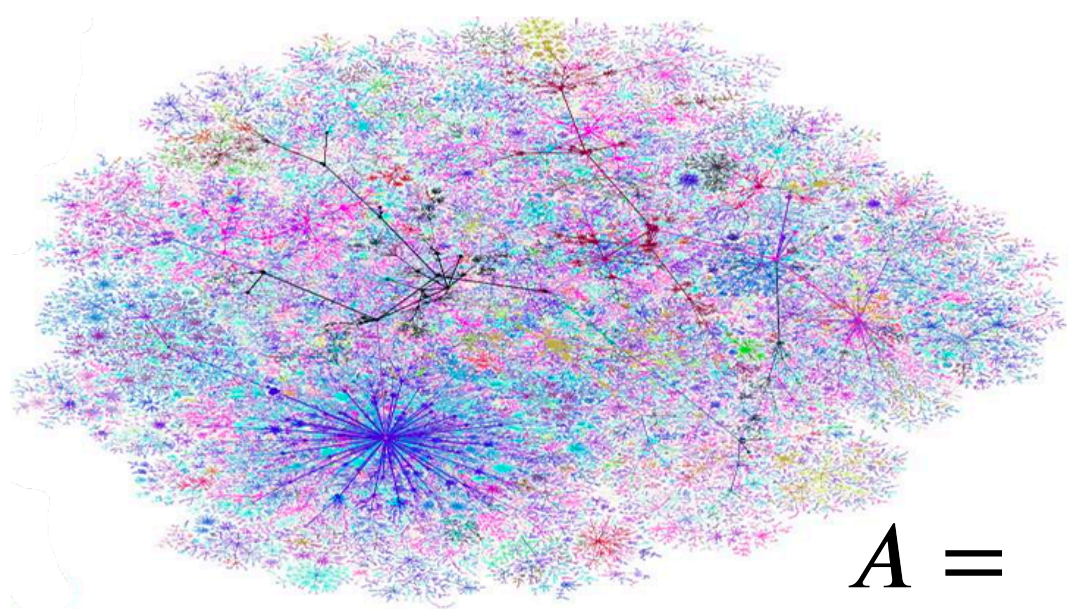
The World Is Not Dense



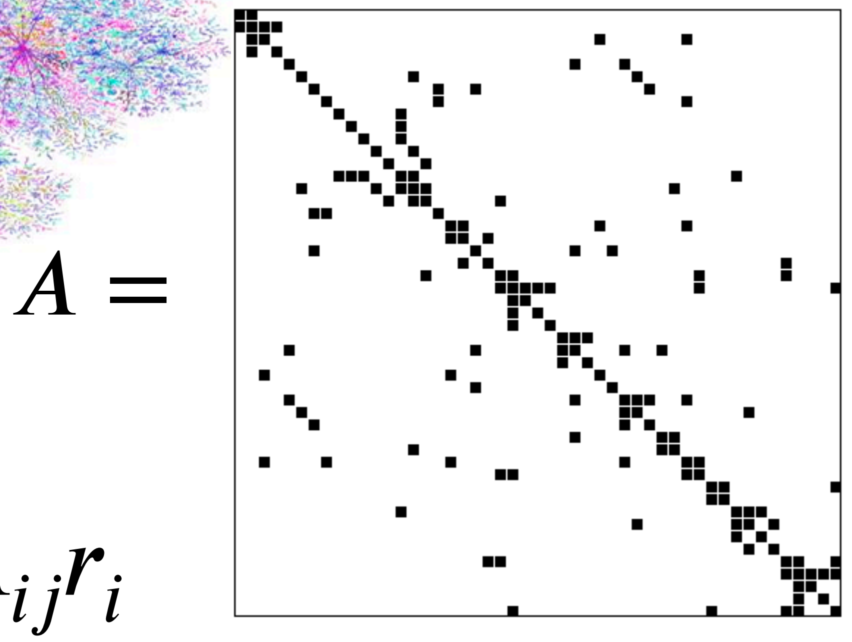
Scientific Computing

dgemm(TRANSA,TRANSB
,M,N,K,ALPHA,A,LDA,
B,LDB,BETA,C,LDC)





Networks



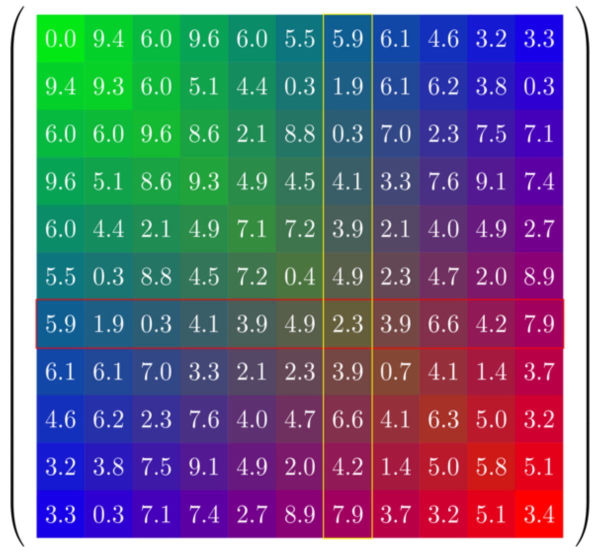
$$r_i = \frac{1-d}{N} + \sum_j d A_{ij} r_i$$

$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & u_{n-1,n} \\ 0 & & & & u_{n,n} \end{bmatrix}$$

dtrmm(SIDE,UPLO,
TRANSA,DIAG,M,N,
ALPHA,A,LDA,B,LDB)

dgbmv(TRANS,M,N,KL
,KU,ALPHA,A,LDA,X,IN
CX,BETA,Y,INCY)

$$\begin{bmatrix} B_{11} & B_{12} & 0 & \cdots & \cdots & 0 \\ B_{21} & B_{22} & B_{23} & \ddots & \ddots & \vdots \\ 0 & B_{32} & B_{33} & B_{34} & \ddots & \vdots \\ \vdots & \ddots & B_{43} & B_{44} & B_{45} & 0 \\ \vdots & \ddots & \ddots & B_{54} & B_{55} & B_{56} \\ 0 & \cdots & \cdots & 0 & B_{65} & B_{66} \end{bmatrix}$$



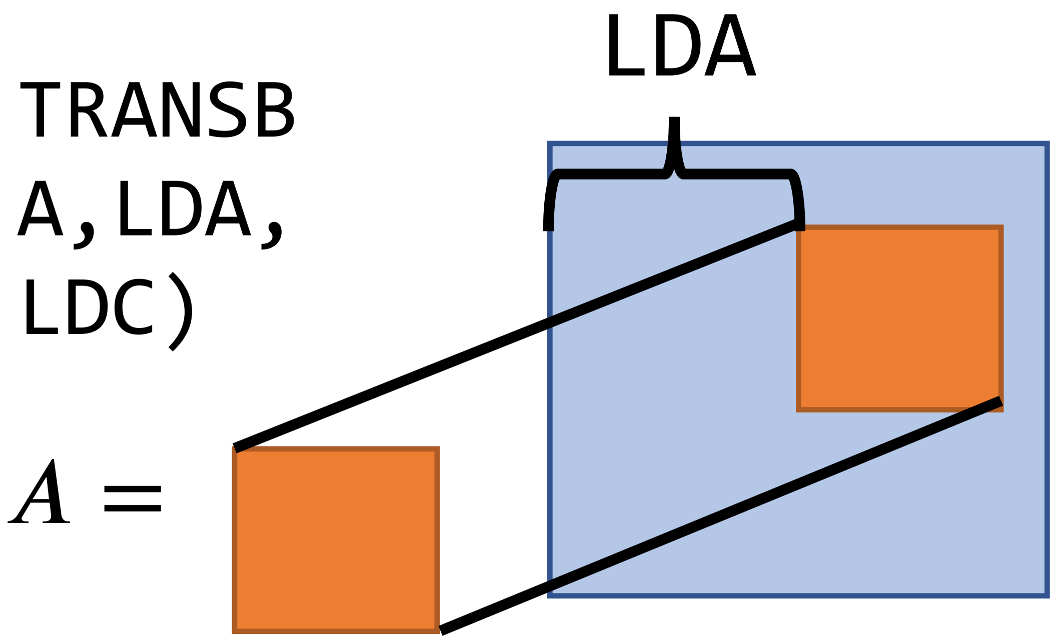
$$\begin{pmatrix} 0.0 & 9.4 & 6.0 & 9.6 & 6.0 & 5.5 & 5.9 & 6.1 & 4.6 & 3.2 & 3.3 \\ 9.4 & 9.3 & 6.0 & 5.1 & 4.4 & 0.3 & 1.9 & 6.1 & 6.2 & 3.8 & 0.3 \\ 6.0 & 6.0 & 9.6 & 8.6 & 2.1 & 8.8 & 0.3 & 7.0 & 2.3 & 7.5 & 7.1 \\ 9.6 & 5.1 & 8.6 & 9.3 & 4.9 & 4.5 & 4.1 & 3.3 & 7.6 & 9.1 & 7.4 \\ 6.0 & 4.4 & 2.1 & 4.9 & 7.1 & 7.2 & 3.9 & 2.1 & 4.0 & 4.9 & 2.7 \\ 5.5 & 0.3 & 8.8 & 4.5 & 7.2 & 0.4 & 4.9 & 2.3 & 4.7 & 2.0 & 8.9 \\ 5.9 & 1.9 & 0.3 & 4.1 & 3.9 & 4.9 & 2.3 & 3.9 & 6.6 & 4.2 & 7.9 \\ 6.1 & 6.1 & 7.0 & 3.3 & 2.1 & 2.3 & 3.9 & 0.7 & 4.1 & 1.4 & 3.7 \\ 4.6 & 6.2 & 2.3 & 7.6 & 4.0 & 4.7 & 6.6 & 4.1 & 6.3 & 5.0 & 3.2 \\ 3.2 & 3.8 & 7.5 & 9.1 & 4.9 & 2.0 & 4.2 & 1.4 & 5.0 & 5.8 & 5.1 \\ 3.3 & 0.3 & 7.1 & 7.4 & 2.7 & 8.9 & 7.9 & 3.7 & 3.2 & 5.1 & 3.4 \end{pmatrix}$$

dsymm(SIDE,UPLO,
M,N,ALPHA,A,LDA,
B,LDB,BETA,C,LDC)

The World Is Not Dense

Scientific Computing

```
dgemm(TRANSA,TRANSB
,M,N,K,ALPHA,A,LDA,
B,LDB,BETA,C,LDC)
```



$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & u_{n-1,n} \\ 0 & & & & u_{n,n} \end{bmatrix}$$

```
dtrmm(SIDE,UPLO,
TRANSA,DIAG,M,N,
ALPHA,A,LDA,B,LDB)
```

dgbmv(TRANS,M,N,KL
,KU,ALPHA,A,LDA,X,IN
CX,BETA,Y,INCY)

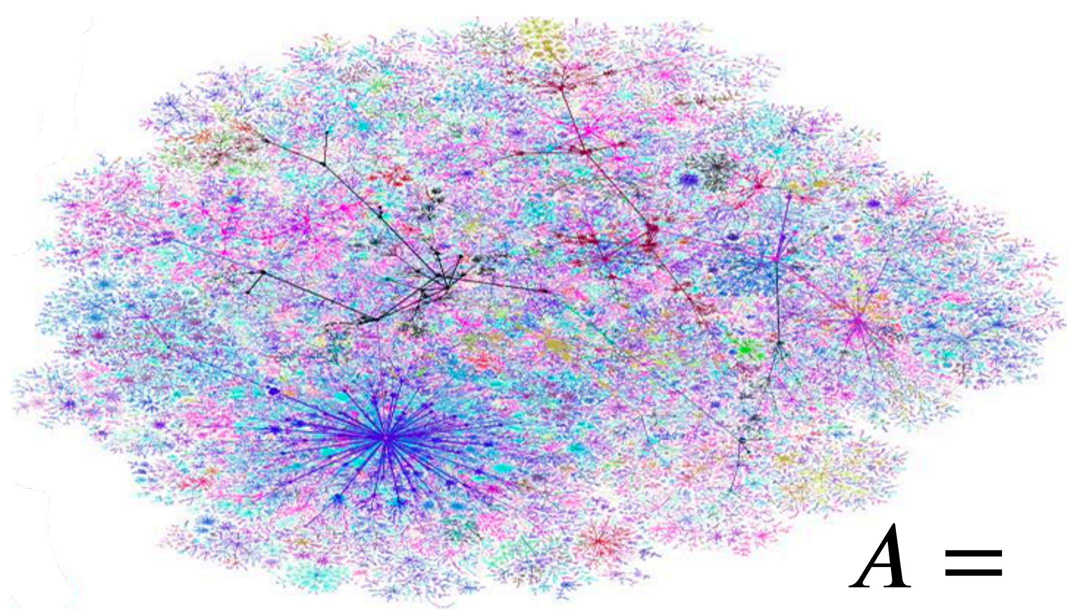
$$\begin{bmatrix} B_{11} & B_{12} & 0 & \cdots & \cdots & 0 \\ B_{21} & B_{22} & B_{23} & \ddots & \ddots & \vdots \\ 0 & B_{32} & B_{33} & B_{34} & \ddots & \vdots \\ \vdots & \ddots & B_{43} & B_{44} & B_{45} & 0 \\ \vdots & \ddots & \ddots & B_{54} & B_{55} & B_{56} \\ 0 & \cdots & \cdots & 0 & B_{65} & B_{66} \end{bmatrix}$$

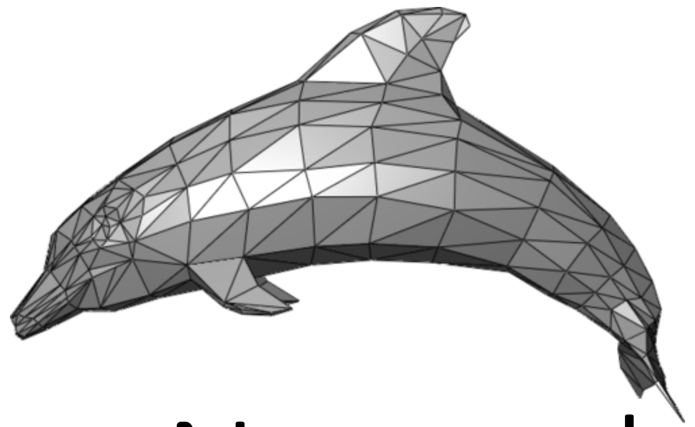
$$\begin{pmatrix} 0.0 & 9.4 & 6.0 & 9.6 & 6.0 & 5.5 & 5.9 & 6.1 & 4.6 & 3.2 & 3.3 \\ 9.4 & 9.3 & 6.0 & 5.1 & 4.4 & 0.3 & 1.9 & 6.1 & 6.2 & 3.8 & 0.3 \\ 6.0 & 6.0 & 9.6 & 8.6 & 2.1 & 8.8 & 0.3 & 7.0 & 2.3 & 7.5 & 7.1 \\ 9.6 & 5.1 & 8.6 & 9.3 & 4.9 & 4.5 & 4.1 & 3.3 & 7.6 & 9.1 & 7.4 \\ 6.0 & 4.4 & 2.1 & 4.9 & 7.1 & 7.2 & 3.9 & 2.1 & 4.0 & 4.9 & 2.7 \\ 5.5 & 0.3 & 8.8 & 4.5 & 7.2 & 0.4 & 4.9 & 2.3 & 4.7 & 2.0 & 8.9 \\ 5.9 & 1.9 & 0.3 & 4.1 & 3.9 & 4.9 & 2.3 & 3.9 & 6.6 & 4.2 & 7.9 \\ 6.1 & 6.1 & 7.0 & 3.3 & 2.1 & 2.3 & 3.9 & 0.7 & 4.1 & 1.4 & 3.7 \\ 4.6 & 6.2 & 2.3 & 7.6 & 4.0 & 4.7 & 6.6 & 4.1 & 6.3 & 5.0 & 3.2 \\ 3.2 & 3.8 & 7.5 & 9.1 & 4.9 & 2.0 & 4.2 & 1.4 & 5.0 & 5.8 & 5.1 \\ 3.3 & 0.3 & 7.1 & 7.4 & 2.7 & 8.9 & 7.9 & 3.7 & 3.2 & 5.1 & 3.4 \end{pmatrix}$$

dsymm(SIDE,UPLO,
M,N,ALPHA,A,LDA,
B,LDB,BETA,C,LDC)

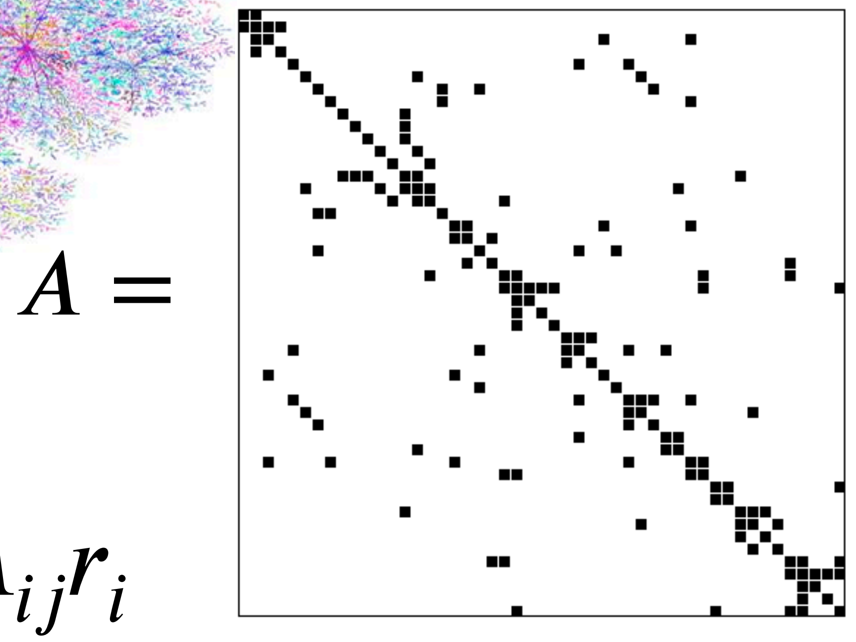
```
block
sparse:
```

$$\begin{pmatrix} 0 & 0 & 5.9 & 4.7 & 0.3 & 0 & 0 & 2.9 & 2.3 & 0 & 0 \\ 6.0 & 0 & 0 & 0 & 4.2 & 7.5 & 5.6 & 0 & 0 & 0 & 0 \\ 3.0 & 0 & 0 & 3.2 & 0 & 0 & 0 & 5.9 & 5.1 & 1.4 & 2.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6.5 & 5.9 & 6.3 & 0 & 0 \\ 0 & 0 & 2.7 & 5.0 & 0.9 & 0 & 0 & 1.4 & 2.3 & 0 & 0 \\ 0 & 0 & 0 & 5.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.9 & 3.9 & 8.2 & 0 & 0 & 0.1 & 4.4 & 4.1 & 0 \\ 8.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.1 \\ 7.9 & 0 & 2.3 & 0 & 0 & 9.0 & 9.1 & 2.5 & 1.1 & 0.8 & 8.6 \\ 0 & 0 & 0.7 & 6.7 & 5.2 & 3.2 & 0 & 0 & 0 & 0 & 0 \\ 1.8 & 0 & 0 & 3.6 & 0 & 4.1 & 0.7 & 7.7 & 3.1 & 0 & 3.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6.2 & 6.4 & 6.5 & 0 & 0 \end{pmatrix}$$





Networks

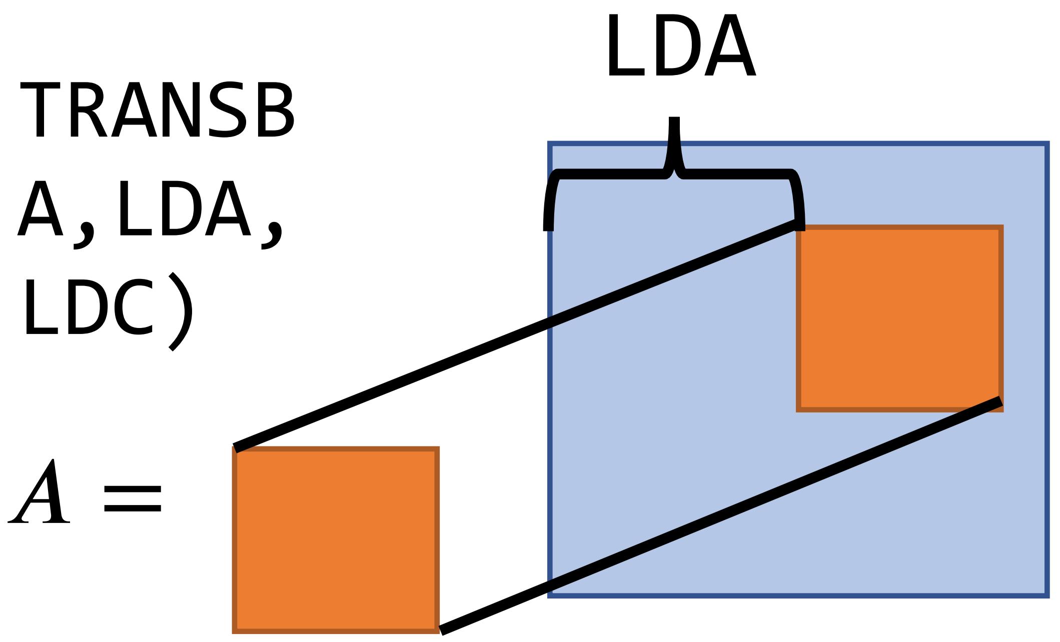


$$r_i = \frac{1-d}{N} + \sum_j d A_{ij} r_i$$

The World Is Not Dense

Scientific Computing

dgemm(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)

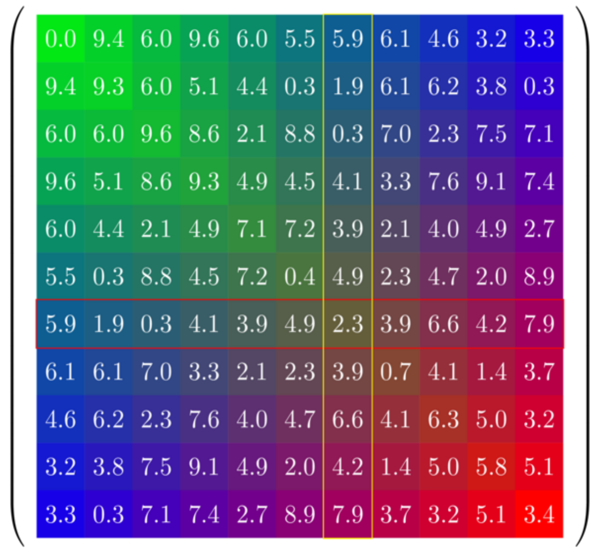


$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & u_{n-1,n} \\ 0 & & & & u_{n,n} \end{bmatrix}$$

dtrmm(SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,LDA,B,LDB)

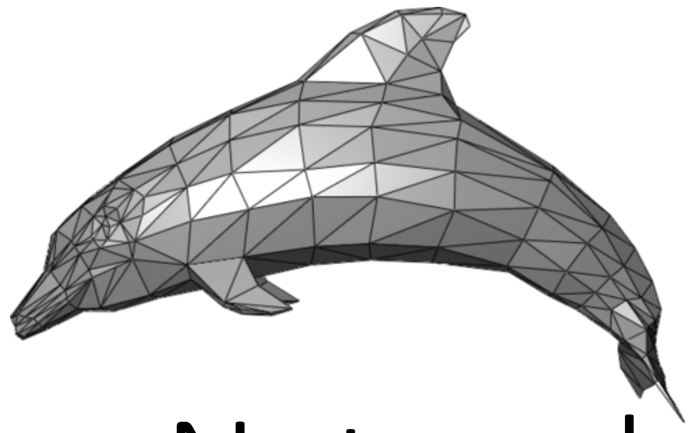
dgbmv(TRANS,M,N,KL,KU,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)

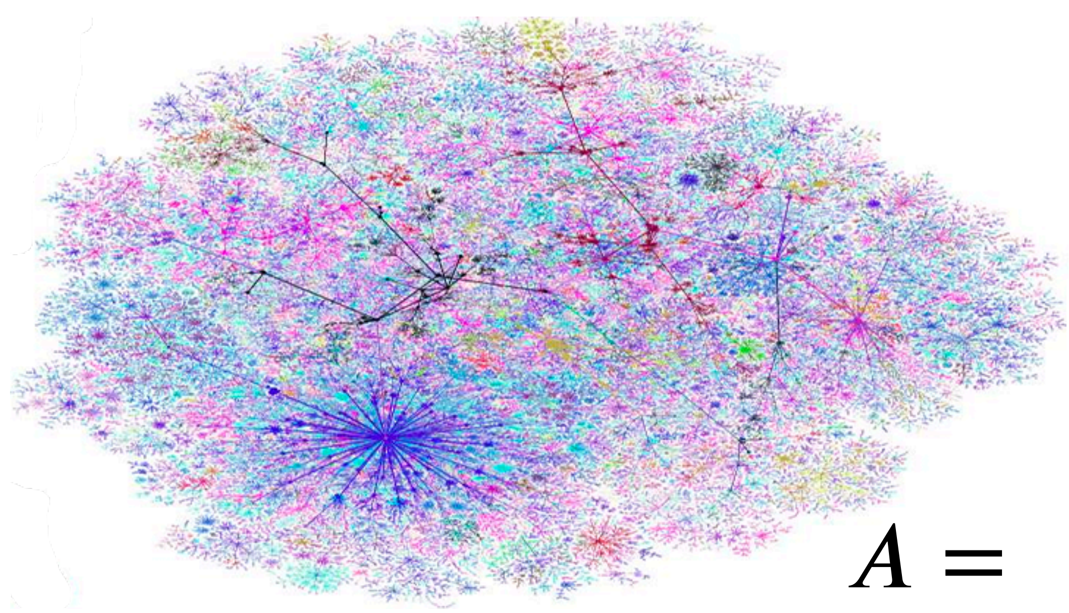
$$\begin{bmatrix} B_{11} & B_{12} & 0 & \cdots & \cdots & 0 \\ B_{21} & B_{22} & B_{23} & \ddots & \ddots & \vdots \\ 0 & B_{32} & B_{33} & B_{34} & \ddots & \vdots \\ \vdots & \ddots & B_{43} & B_{44} & B_{45} & 0 \\ \vdots & \ddots & \ddots & B_{54} & B_{55} & B_{56} \\ 0 & \cdots & \cdots & 0 & B_{65} & B_{66} \end{bmatrix}$$



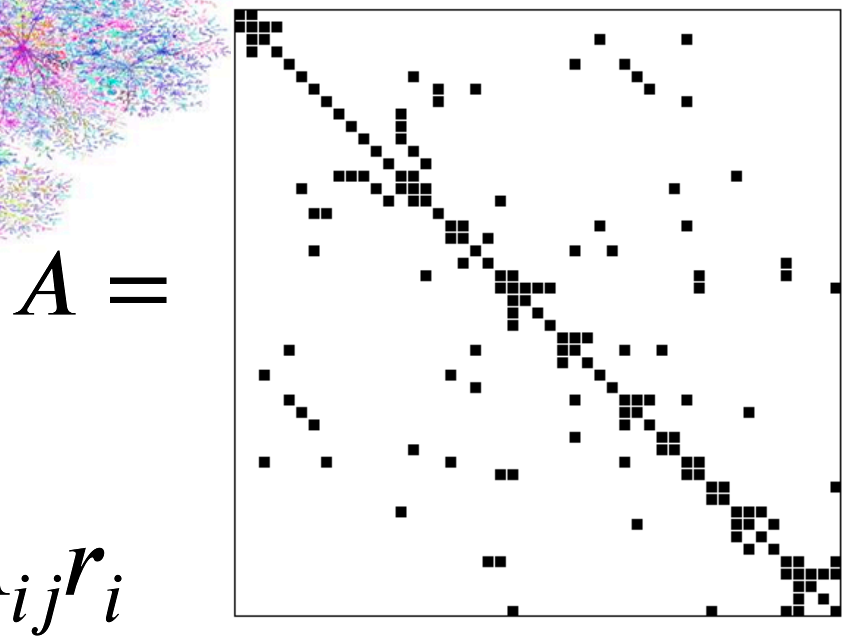
$$\begin{pmatrix} 0.0 & 9.4 & 6.0 & 9.6 & 6.0 & 5.5 & 5.9 & 6.1 & 4.6 & 3.2 & 3.3 \\ 9.4 & 9.3 & 6.0 & 5.1 & 4.4 & 0.3 & 1.9 & 6.1 & 6.2 & 3.8 & 0.3 \\ 6.0 & 6.0 & 9.6 & 8.6 & 2.1 & 8.8 & 0.3 & 7.0 & 2.3 & 7.5 & 7.1 \\ 9.6 & 5.1 & 8.6 & 9.3 & 4.9 & 4.5 & 4.1 & 3.3 & 7.6 & 9.1 & 7.4 \\ 6.0 & 4.4 & 2.1 & 4.9 & 7.1 & 7.2 & 3.9 & 2.1 & 4.0 & 4.9 & 2.7 \\ 5.5 & 0.3 & 8.8 & 4.5 & 7.2 & 0.4 & 4.9 & 2.3 & 4.7 & 2.0 & 8.9 \\ 5.9 & 1.9 & 0.3 & 4.1 & 3.9 & 4.9 & 2.3 & 3.9 & 6.6 & 4.2 & 7.9 \\ 6.1 & 6.1 & 7.0 & 3.3 & 2.1 & 2.3 & 3.9 & 0.7 & 4.1 & 1.4 & 3.7 \\ 4.6 & 6.2 & 2.3 & 7.6 & 4.0 & 4.7 & 6.6 & 4.1 & 6.3 & 5.0 & 3.2 \\ 3.2 & 3.8 & 7.5 & 9.1 & 4.9 & 2.0 & 4.2 & 1.4 & 5.0 & 5.8 & 5.1 \\ 3.3 & 0.3 & 7.1 & 7.4 & 2.7 & 8.9 & 7.9 & 3.7 & 3.2 & 5.1 & 3.4 \end{pmatrix}$$

dsymm(SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)



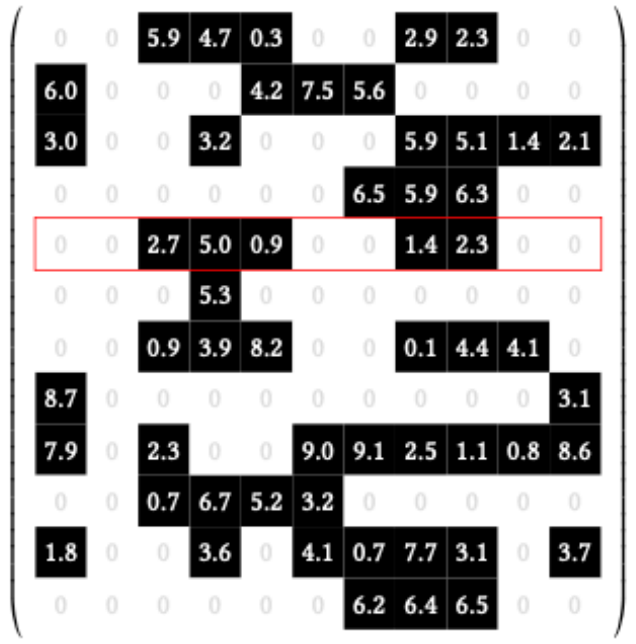


Networks



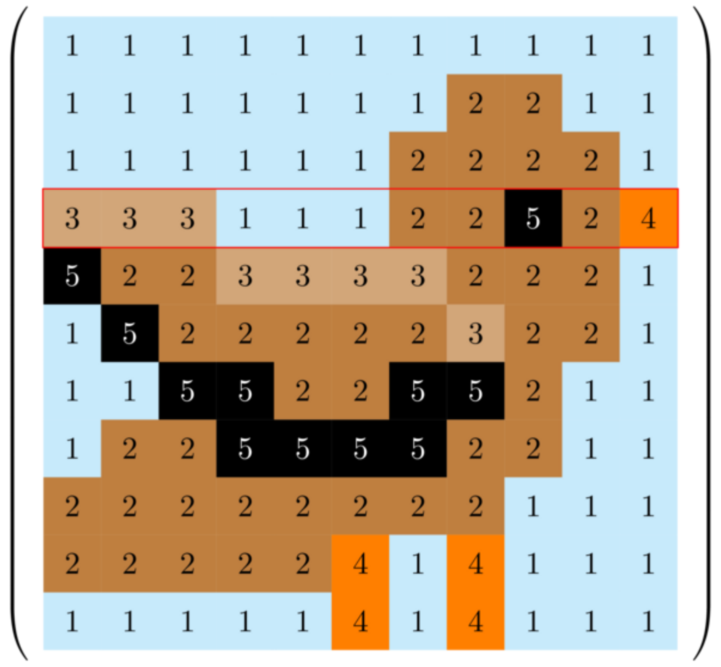
$$r_i = \frac{1-d}{N} + \sum_j d A_{ij} r_i$$

block sparse:



$$\begin{pmatrix} 0 & 0 & 5.9 & 4.7 & 0.3 & 0 & 0 & 2.9 & 2.3 & 0 & 0 \\ 6.0 & 0 & 0 & 0 & 4.2 & 7.5 & 5.6 & 0 & 0 & 0 & 0 \\ 3.0 & 0 & 0 & 3.2 & 0 & 0 & 0 & 5.9 & 5.1 & 1.4 & 2.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6.5 & 5.9 & 6.3 & 0 & 0 \\ 0 & 0 & 2.7 & 5.0 & 0.9 & 0 & 0 & 1.4 & 2.3 & 0 & 0 \\ 0 & 0 & 0 & 5.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.9 & 3.9 & 8.2 & 0 & 0 & 0.1 & 4.4 & 4.1 & 0 \\ 8.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.1 \\ 7.9 & 0 & 2.3 & 0 & 0 & 9.0 & 9.1 & 2.5 & 1.1 & 0.8 & 8.6 \\ 0 & 0 & 0.7 & 6.7 & 5.2 & 3.2 & 0 & 0 & 0 & 0 & 0 \\ 1.8 & 0 & 0 & 3.6 & 0 & 4.1 & 0.7 & 7.7 & 3.1 & 0 & 3.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6.2 & 6.4 & 6.5 & 0 & 0 \end{pmatrix}$$

Image Processing



$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 1 \\ 3 & 3 & 3 & 1 & 1 & 1 & 2 & 2 & 5 & 2 & 4 \\ 5 & 2 & 2 & 3 & 3 & 3 & 3 & 2 & 2 & 2 & 1 \\ 1 & 5 & 2 & 2 & 2 & 2 & 2 & 3 & 2 & 2 & 1 \\ 1 & 1 & 5 & 5 & 2 & 2 & 5 & 5 & 2 & 1 & 1 \\ 1 & 2 & 2 & 5 & 5 & 5 & 5 & 2 & 2 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 4 & 1 & 4 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 4 & 1 & 4 & 1 & 1 & 1 \end{pmatrix}$$

run-length encoding:

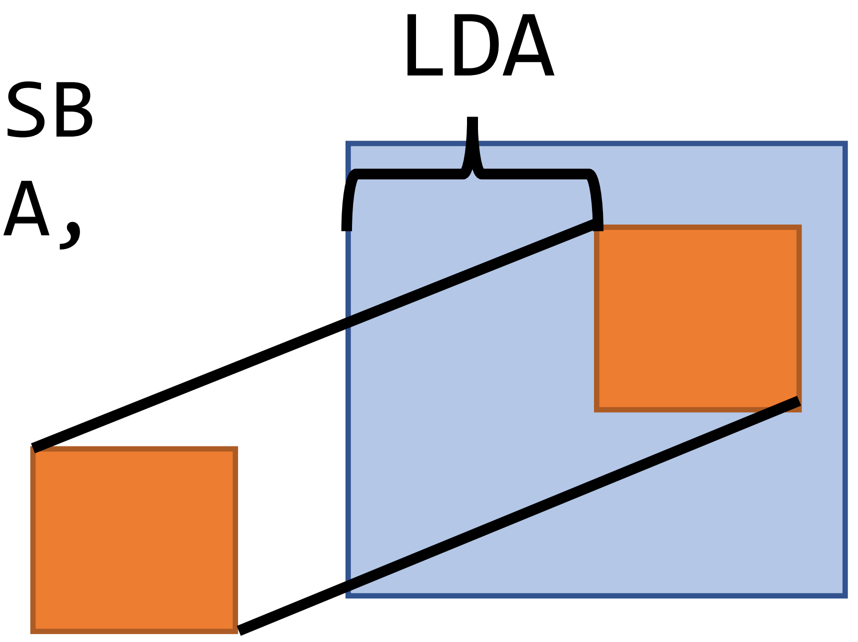
$$\Big(\underbrace{3 \quad 3 \quad 3}_{3} \quad \underbrace{1 \quad 1 \quad 1}_{1} \quad \underbrace{2 \quad 2}_{2} \quad \underbrace{5}_{5} \quad \underbrace{2}_{2} \quad \underbrace{4}_{4} \Big)$$

The World Is Not Dense

Scientific Computing

```
dgemm(TRANSA,TRANSB
,M,N,K,ALPHA,A,LDA,
B,LDB,BETA,C,LDC)
```

$A =$



$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & u_{n-1,n} \\ 0 & & & & u_{n,n} \end{bmatrix}$$

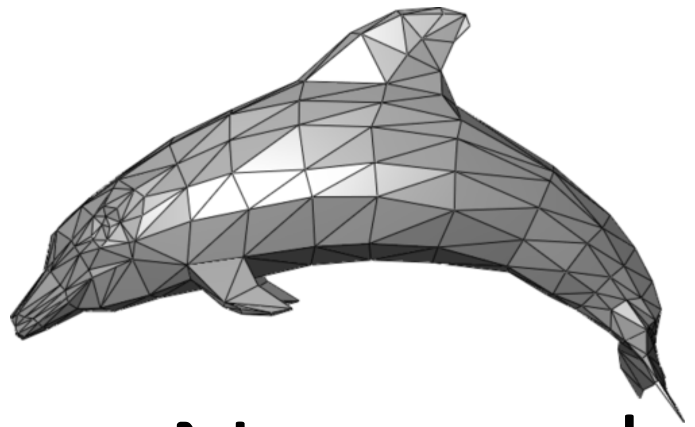
```
dtrmm(SIDE,UPLO,
TRANSA,DIAG,M,N,
ALPHA,A,LDA,B,LDB)
```

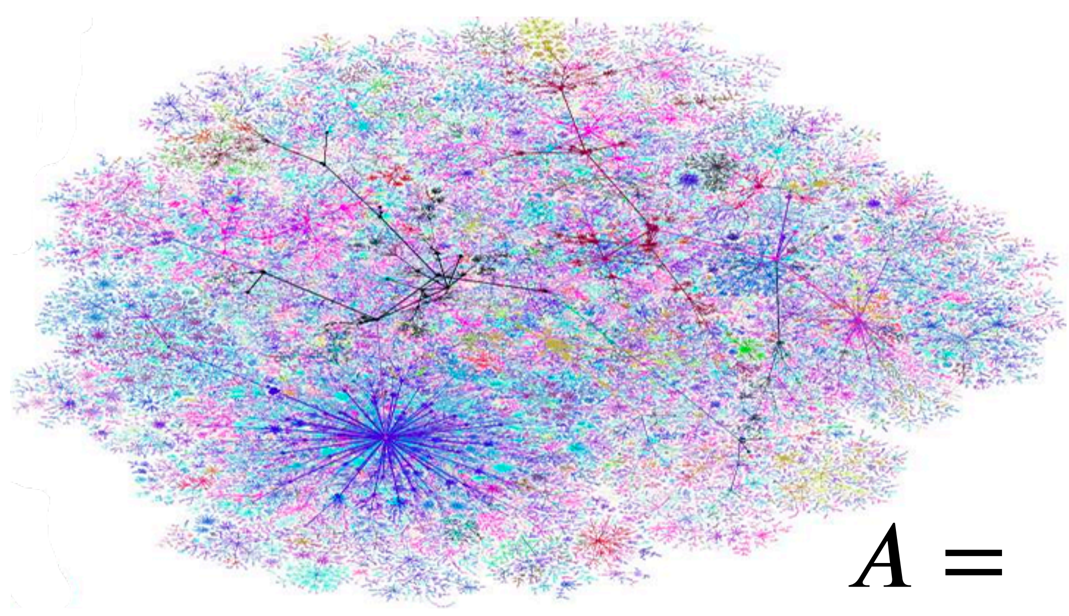
dgbmv(TRANS,M,N,KL
,KU,ALPHA,A,LDA,X,IN
CX,BETA,Y,INCY)

$$\begin{bmatrix} B_{11} & B_{12} & 0 & \cdots & \cdots & 0 \\ B_{21} & B_{22} & B_{23} & \ddots & \ddots & \vdots \\ 0 & B_{32} & B_{33} & B_{34} & \ddots & \vdots \\ \vdots & \ddots & B_{43} & B_{44} & B_{45} & 0 \\ \vdots & \ddots & \ddots & B_{54} & B_{55} & B_{56} \\ 0 & \cdots & \cdots & 0 & B_{65} & B_{66} \end{bmatrix}$$

$$\begin{pmatrix} 0.0 & 9.4 & 6.0 & 9.6 & 6.0 & 5.5 & 5.9 & 6.1 & 4.6 & 3.2 & 3.3 \\ 9.4 & 9.3 & 6.0 & 5.1 & 4.4 & 0.3 & 1.9 & 6.1 & 6.2 & 3.8 & 0.3 \\ 6.0 & 6.0 & 9.6 & 8.6 & 2.1 & 8.8 & 0.3 & 7.0 & 2.3 & 7.5 & 7.1 \\ 9.6 & 5.1 & 8.6 & 9.3 & 4.9 & 4.5 & 4.1 & 3.3 & 7.6 & 9.1 & 7.4 \\ 6.0 & 4.4 & 2.1 & 4.9 & 7.1 & 7.2 & 3.9 & 2.1 & 4.0 & 4.9 & 2.7 \\ 5.5 & 0.3 & 8.8 & 4.5 & 7.2 & 0.4 & 4.9 & 2.3 & 4.7 & 2.0 & 8.9 \\ 5.9 & 1.9 & 0.3 & 4.1 & 3.9 & 4.9 & 2.3 & 3.9 & 6.6 & 4.2 & 7.9 \\ 6.1 & 6.1 & 7.0 & 3.3 & 2.1 & 2.3 & 3.9 & 0.7 & 4.1 & 1.4 & 3.7 \\ 4.6 & 6.2 & 2.3 & 7.6 & 4.0 & 4.7 & 6.6 & 4.1 & 6.3 & 5.0 & 3.2 \\ 3.2 & 3.8 & 7.5 & 9.1 & 4.9 & 2.0 & 4.2 & 1.4 & 5.0 & 5.8 & 5.1 \\ 3.3 & 0.3 & 7.1 & 7.4 & 2.7 & 8.9 & 7.9 & 3.7 & 3.2 & 5.1 & 3.4 \end{pmatrix}$$

dsymm(SIDE,UPLO,
M,N,ALPHA,A,LDA,
B,LDB,BETA,C,LDC)





Networks

$$r_i = \frac{1-d}{N} + \sum_j d A_{ij} r_i$$

$A =$

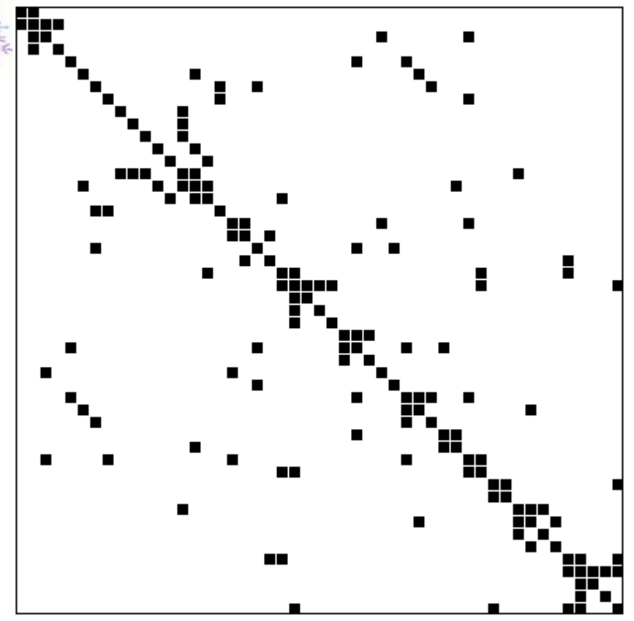


Image Processing

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 1 \\ 3 & 3 & 3 & 1 & 1 & 1 & 2 & 2 & 5 & 2 & 4 \\ 5 & 2 & 2 & 3 & 3 & 3 & 3 & 2 & 2 & 2 & 1 \\ 1 & 5 & 2 & 2 & 2 & 2 & 2 & 3 & 2 & 2 & 1 \\ 1 & 1 & 5 & 5 & 2 & 2 & 5 & 5 & 2 & 1 & 1 \\ 1 & 2 & 2 & 5 & 5 & 5 & 5 & 2 & 2 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 4 & 1 & 4 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 4 & 1 & 4 & 1 & 1 & 1 \end{pmatrix}$$

`run-length encoding:`

$$\left(\underbrace{3\ \ 3\ \ 3}_{3}\ \ \underbrace{1\ \ 1\ \ 1}_{1}\ \ \underbrace{2\ \ 2}_{2}\ \ \underbrace{5}_{5}\ \ \underbrace{2}_{2}\ \ \underbrace{4}_{4}\right)$$

Mathematical Optimization

`block sparse:`

$$\begin{pmatrix} 0 & 0 & 5.9 & 4.7 & 0.3 & 0 & 0 & 2.9 & 2.3 & 0 & 0 \\ 6.0 & 0 & 0 & 0 & 4.2 & 7.5 & 5.6 & 0 & 0 & 0 & 0 \\ 3.0 & 0 & 0 & 3.2 & 0 & 0 & 0 & 5.9 & 5.1 & 1.4 & 2.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6.5 & 5.9 & 6.3 & 0 & 0 \\ 0 & 0 & 2.7 & 5.0 & 0.9 & 0 & 0 & 1.4 & 2.3 & 0 & 0 \\ 0 & 0 & 0 & 5.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.9 & 3.9 & 8.2 & 0 & 0 & 0.1 & 4.4 & 4.1 & 0 \\ 8.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.1 \\ 7.9 & 0 & 2.3 & 0 & 0 & 9.0 & 9.1 & 2.5 & 1.1 & 0.8 & 8.6 \\ 0 & 0 & 0.7 & 6.7 & 5.2 & 3.2 & 0 & 0 & 0 & 0 & 0 \\ 1.8 & 0 & 0 & 3.6 & 0 & 4.1 & 0.7 & 7.7 & 3.1 & 0 & 3.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6.2 & 6.4 & 6.5 & 0 & 0 \end{pmatrix}$$

`convolution (Toeplitz):`

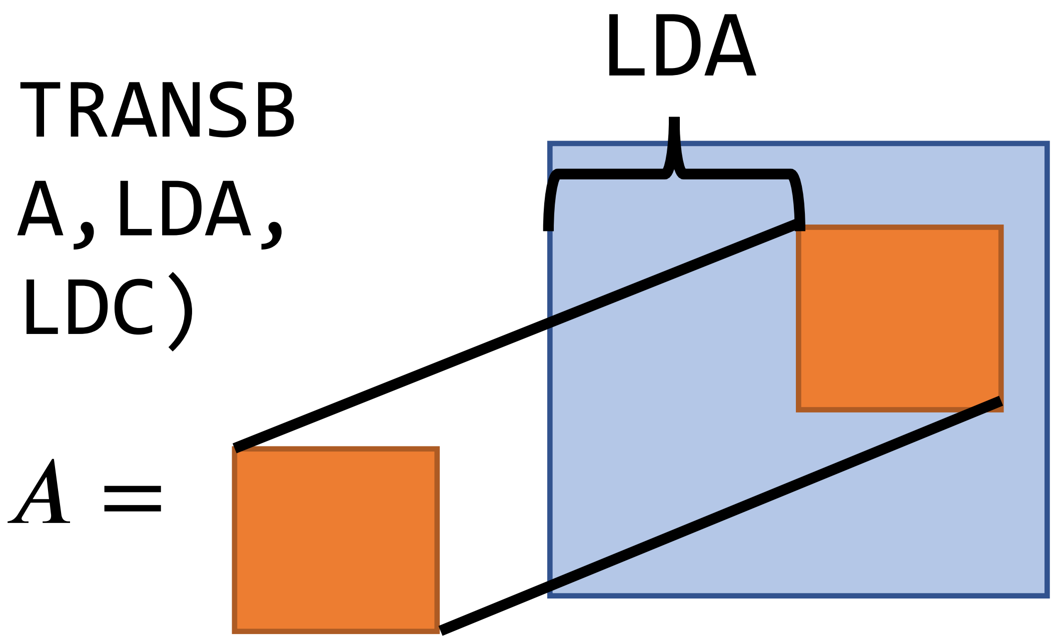
$$y_i = \sum_j x_{i-j} k_j \qquad A_{ij} = x_{i-j}$$

$$\begin{array}{c|c} & i \\ \hline j & \begin{bmatrix} 5 & 3 & 7 & 2 & 0 & 0 & 0 \\ 0 & 5 & 3 & 7 & 2 & 0 & 0 \\ 0 & 0 & 5 & 3 & 7 & 2 & 0 \\ 0 & 0 & 0 & 5 & 3 & 7 & 2 \end{bmatrix} \\ & A \end{array} \qquad \begin{array}{c} i-j \\ \begin{bmatrix} 5 & 3 & 7 & 2 \end{bmatrix} \\ x \end{array}$$

The World Is Not Dense

Scientific Computing

dgemm(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)



$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & u_{n-1,n} \\ 0 & & & & u_{n,n} \end{bmatrix}$$

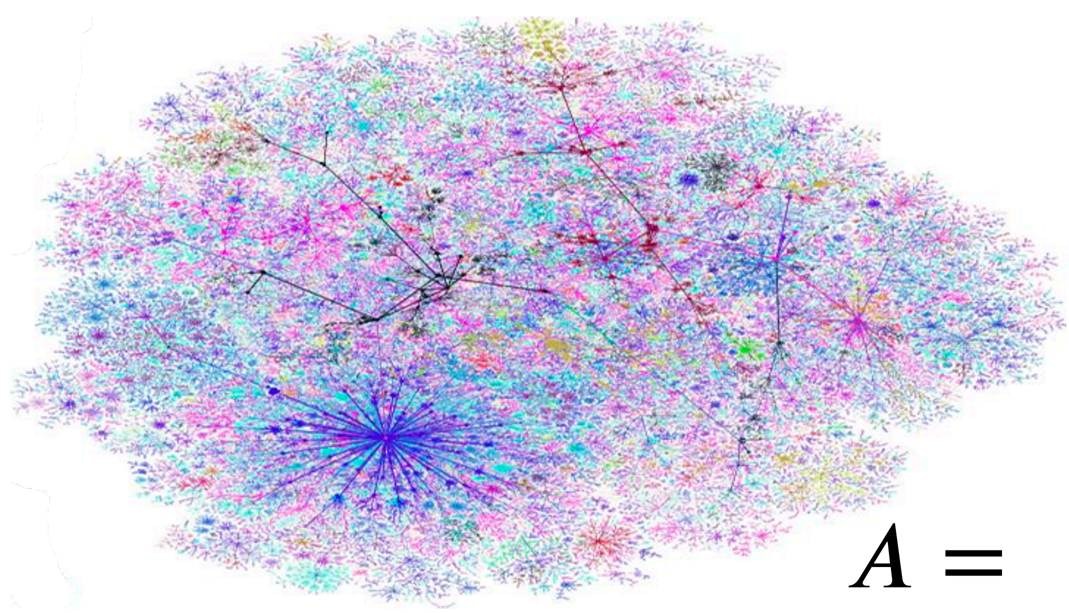
dtrmm(SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,LDA,B,LDB)

dgbmv(TRANS,M,N,KL,KU,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)

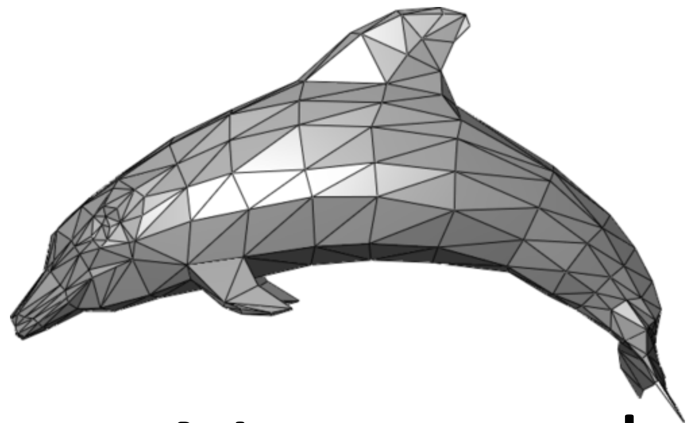
$$\begin{bmatrix} B_{11} & B_{12} & 0 & \cdots & \cdots & 0 \\ B_{21} & B_{22} & B_{23} & \ddots & \ddots & \vdots \\ 0 & B_{32} & B_{33} & B_{34} & \ddots & \vdots \\ \vdots & \ddots & B_{43} & B_{44} & B_{45} & 0 \\ \vdots & \ddots & \ddots & B_{54} & B_{55} & B_{56} \\ 0 & \cdots & \cdots & 0 & B_{65} & B_{66} \end{bmatrix}$$

$$\begin{pmatrix} 0.0 & 9.4 & 6.0 & 9.6 & 6.0 & 5.5 & 5.9 & 6.1 & 4.6 & 3.2 & 3.3 \\ 9.4 & 9.3 & 6.0 & 5.1 & 4.4 & 0.3 & 1.9 & 6.1 & 6.2 & 3.8 & 0.3 \\ 6.0 & 6.0 & 9.6 & 8.6 & 2.1 & 8.8 & 0.3 & 7.0 & 2.3 & 7.5 & 7.1 \\ 9.6 & 5.1 & 8.6 & 9.3 & 4.9 & 4.5 & 4.1 & 3.3 & 7.6 & 9.1 & 7.4 \\ 6.0 & 4.4 & 2.1 & 4.9 & 7.1 & 7.2 & 3.9 & 2.1 & 4.0 & 4.9 & 2.7 \\ 5.5 & 0.3 & 8.8 & 4.5 & 7.2 & 0.4 & 4.9 & 2.3 & 4.7 & 2.0 & 8.9 \\ 5.9 & 1.9 & 0.3 & 4.1 & 3.9 & 4.9 & 2.3 & 3.9 & 6.6 & 4.2 & 7.9 \\ 6.1 & 6.1 & 7.0 & 3.3 & 2.1 & 2.3 & 3.9 & 0.7 & 4.1 & 1.4 & 3.7 \\ 4.6 & 6.2 & 2.3 & 7.6 & 4.0 & 4.7 & 6.6 & 4.1 & 6.3 & 5.0 & 3.2 \\ 3.2 & 3.8 & 7.5 & 9.1 & 4.9 & 2.0 & 4.2 & 1.4 & 5.0 & 5.8 & 5.1 \\ 3.3 & 0.3 & 7.1 & 7.4 & 2.7 & 8.9 & 7.9 & 3.7 & 3.2 & 5.1 & 3.4 \end{pmatrix}$$

dsymm(SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)

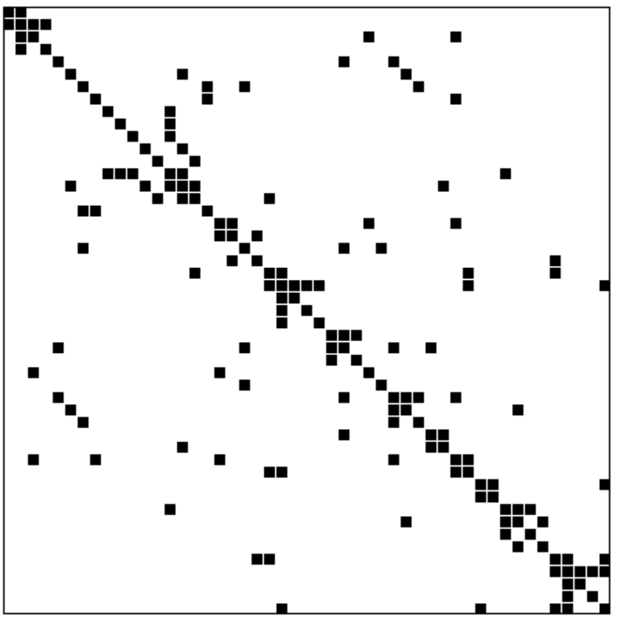


$$r_i = \frac{1-d}{N} + \sum_j d A_{ij} r_i$$



Networks

$A =$



Mathematical Optimization

block sparse:

$$\begin{pmatrix} 0 & 0 & 5.9 & 4.7 & 0.3 & 0 & 0 & 2.9 & 2.3 & 0 & 0 \\ 6.0 & 0 & 0 & 0 & 4.2 & 7.5 & 5.6 & 0 & 0 & 0 & 0 \\ 3.0 & 0 & 0 & 3.2 & 0 & 0 & 0 & 5.9 & 5.1 & 1.4 & 2.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6.5 & 5.9 & 6.3 & 0 & 0 \\ 0 & 0 & 2.7 & 5.0 & 0.9 & 0 & 0 & 1.4 & 2.3 & 0 & 0 \\ 0 & 0 & 0 & 5.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.9 & 3.9 & 8.2 & 0 & 0 & 0.1 & 4.4 & 4.1 & 0 \\ 8.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.1 \\ 7.9 & 0 & 2.3 & 0 & 0 & 9.0 & 9.1 & 2.5 & 1.1 & 0.8 & 8.6 \\ 0 & 0 & 0.7 & 6.7 & 5.2 & 3.2 & 0 & 0 & 0 & 0 & 0 \\ 1.8 & 0 & 0 & 3.6 & 0 & 4.1 & 0.7 & 7.7 & 3.1 & 0 & 3.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6.2 & 6.4 & 6.5 & 0 & 0 \end{pmatrix}$$

Image Processing

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 1 \\ 3 & 3 & 3 & 1 & 1 & 1 & 2 & 2 & 5 & 2 & 4 \\ 5 & 2 & 2 & 3 & 3 & 3 & 3 & 2 & 2 & 2 & 1 \\ 1 & 5 & 2 & 2 & 2 & 2 & 2 & 3 & 2 & 2 & 1 \\ 1 & 1 & 5 & 5 & 2 & 2 & 5 & 5 & 2 & 1 & 1 \\ 1 & 2 & 2 & 5 & 5 & 5 & 5 & 2 & 2 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 4 & 1 & 4 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 4 & 1 & 4 & 1 & 1 & 1 \end{pmatrix}$$

run-length encoding:

$$\big(\underbrace{3\ 3\ 3}_{3}\ \underbrace{1\ 1\ 1}_{1}\ \underbrace{2\ 2}_{2}\ \underbrace{5}_{5}\ \underbrace{2}_{2}\ \underbrace{4}_{4}\big)$$

convolution (Toeplitz):

$y_i = \sum_j x_{i-j} k_j \qquad A_{ij} = x_{i-j}$

$$\begin{array}{c|ccccccc} & & & & i & & & \\ \hline & 5 & 3 & 7 & 2 & 0 & 0 & 0 \\ j & 0 & 5 & 3 & 7 & 2 & 0 & 0 \\ & 0 & 0 & 5 & 3 & 7 & 2 & 0 \\ & 0 & 0 & 0 & 5 & 3 & 7 & 2 \end{array}$$

A

$i - j$

$$\begin{bmatrix} 5 & 3 & 7 & 2 \end{bmatrix}$$

x

padding:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 3 & 7 & 0 \\ 0 & 5 & 3 & 7 & 0 \\ 0 & 5 & 3 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Arrays Are

- Multi-dimensional
- Rectilinear
- Dense
- Integer grid

Of points

Arrays Are

- Multi-dimensional
- Rectilinear
- ~~Dense~~
- Integer grid

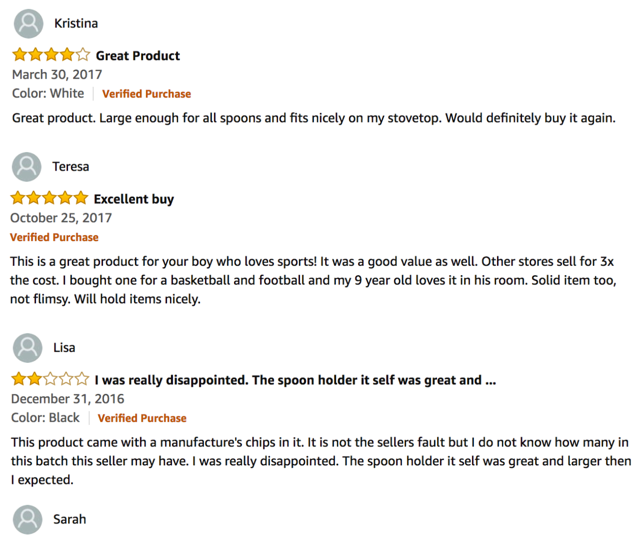
Of points

For Example, Sparse Tensors Are Everywhere

Data Analytics

New TV to watch instantly
See all ›
Avatar: The Last Airbender: Book 3
Weeds: Season 3
Arrested Development: Season 3
Sanctuary: Season 2
Bones: Season 3
Play
Not Interested

Movies



Kristina

★★★★☆ **Great Product**

March 30, 2017

Color: White | Verified Purchase

Great product. Large enough for all spoons and fits nicely on my stovetop. Would definitely buy it again.

Teresa

★★★★★ **Excellent buy**

October 25, 2017

Verified Purchase

This is a great product for your boy who loves sports! It was a good value as well. Other stores sell for 3x the cost. I bought one for a basketball and football and my 9 year old loves it in his room. Solid item too, not flimsy. Will hold items nicely.

Lisa

★★☆☆☆ **I was really disappointed. The spoon holder it self was great and ...**

December 31, 2016

Color: Black | Verified Purchase

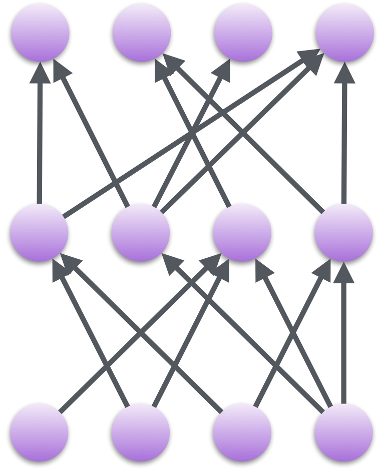
This product came with a manufacture's chips in it. It is not the sellers fault but I do not know how many in this batch this seller may have. I was really disappointed. The spoon holder it self was great and larger then I expected.

Sarah

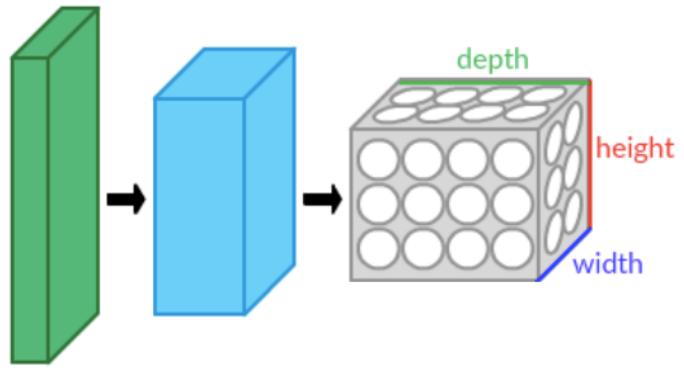
Social Networks

Product Reviews

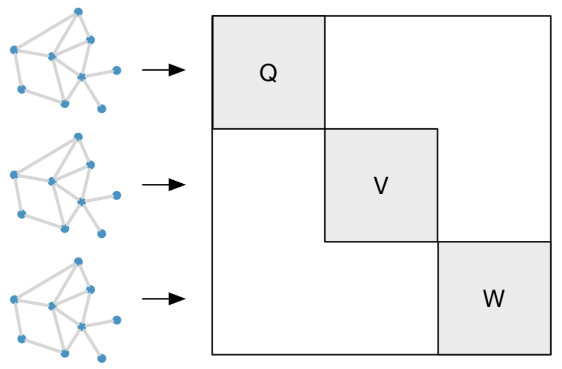
Machine Learning



Sparse Networks

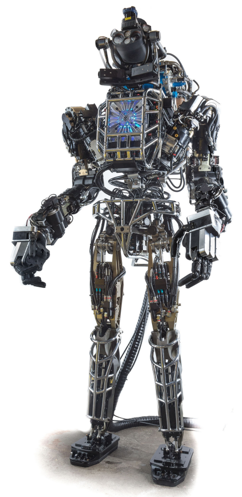


Sparse Convolutional Networks

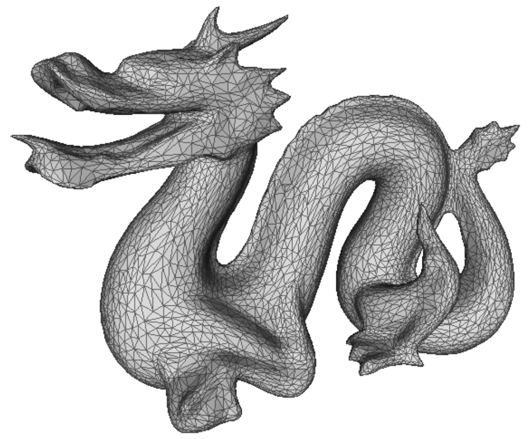


Graph Convolutional Network

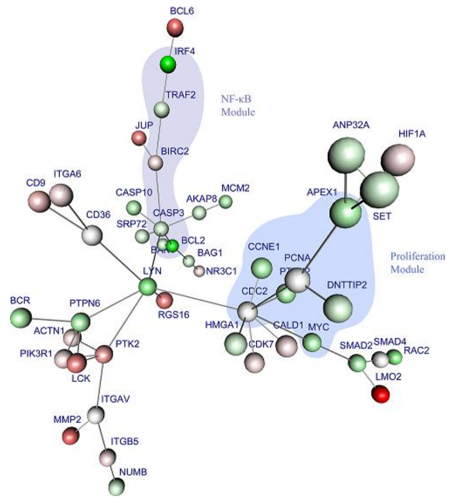
Science and Engineering



Robotics



Simulations



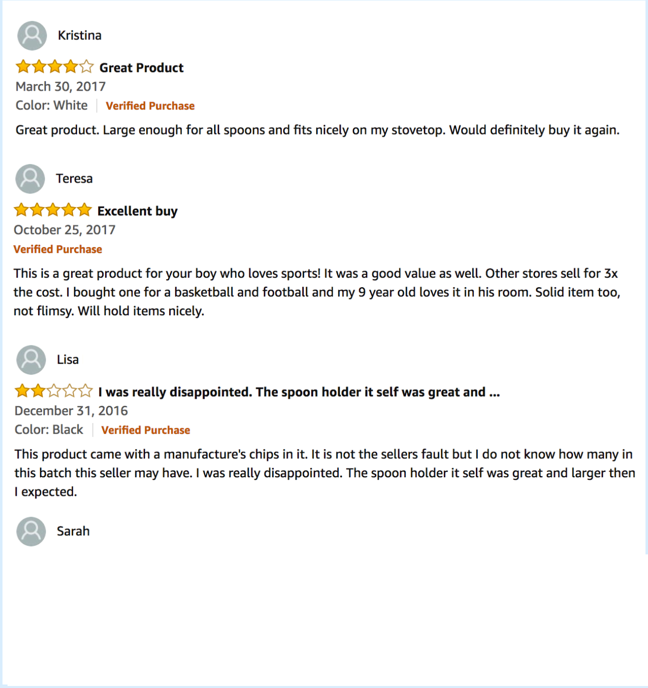
Computational Biology

For Example, Sparse Tensors Are Everywhere

Data Analytics

New TV to watch instantly
See all
Avatar: The Last Airbender: Book 3
Weeds: Season 3
Arrested Development: Season 3
Sanctuary: Season 2
Bones: Season 3
Play
Not Interested

Movies



Kristina

★★★★☆ **Great Product**

March 30, 2017

Color: White | Verified Purchase

Great product. Large enough for all spoons and fits nicely on my stovetop. Would definitely buy it again.

Teresa

★★★★★ **Excellent buy**

October 25, 2017

Verified Purchase

This is a great product for your boy who loves sports! It was a good value as well. Other stores sell for 3x the cost. I bought one for a basketball and football and my 9 year old loves it in his room. Solid item too, not flimsy. Will hold items nicely.

Lisa

★★☆☆☆ **I was really disappointed. The spoon holder it self was great and ...**

December 31, 2016

Color: Black | Verified Purchase

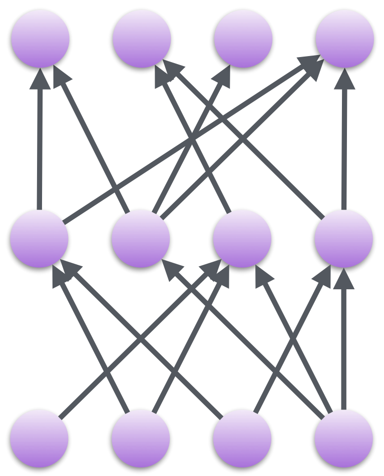
This product came with a manufacture's chips in it. It is not the sellers fault but I do not know how many in this batch this seller may have. I was really disappointed. The spoon holder it self was great and larger then I expected.

Sarah

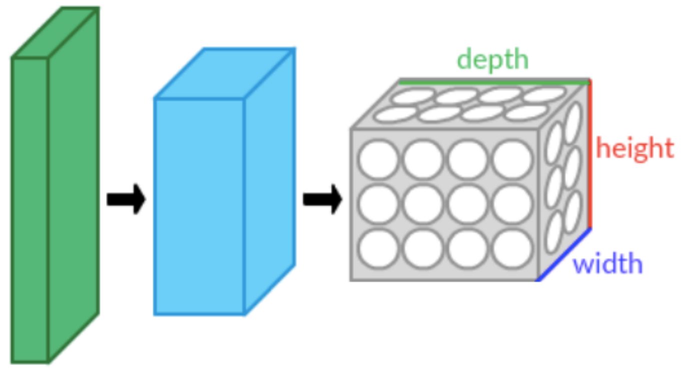
Product Reviews

Social Networks

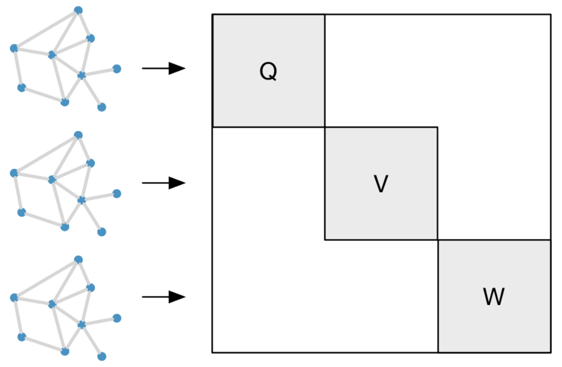
Machine Learning



Sparse Networks

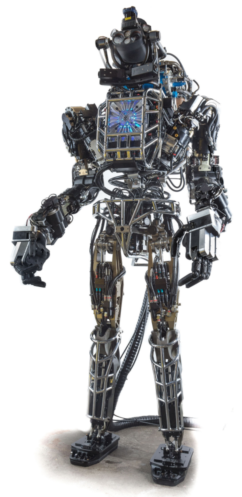


Sparse Convolutional Networks

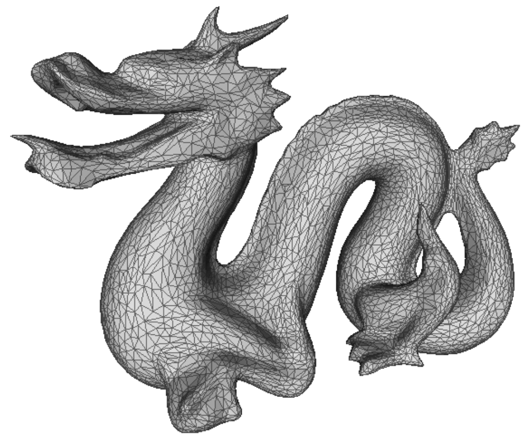


Graph Convolutional Network

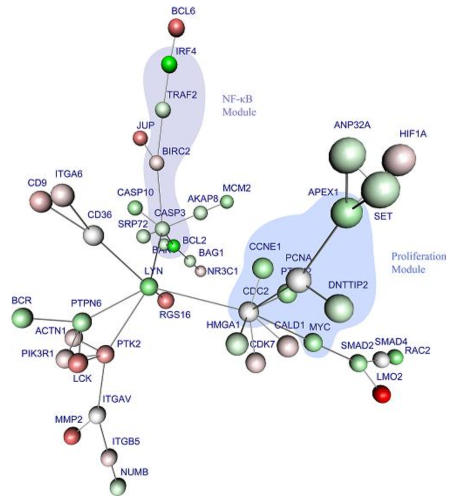
Science and Engineering



Robotics



Simulations



Computational Biology

For Example, Sparse Tensors Are Everywhere

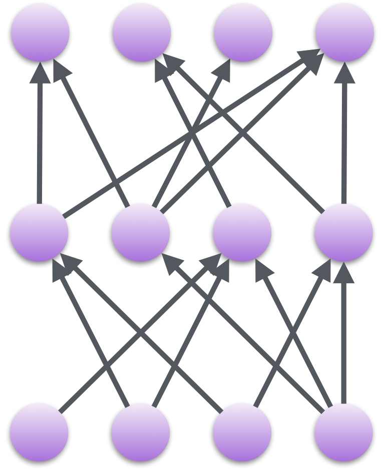
Data Analytics

New TV to watch instantly
See all
Avatar: The Last Airbender: Book 3
Weeds: Season 3
Arrested Development: Season 3
Sanctuary: Season 2
Bones: Season 3
Play
Not Interested

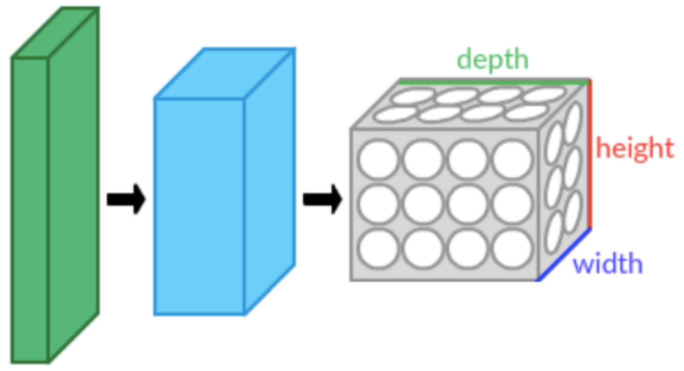
Movies

Social Networks

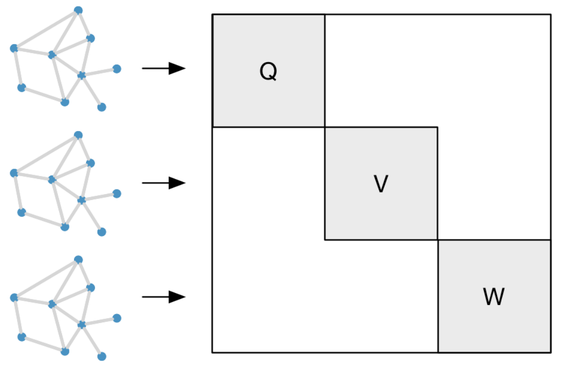
Machine Learning



Sparse Networks

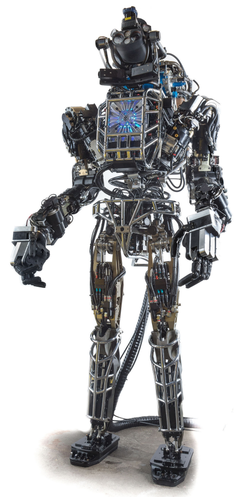


Sparse Convolutional Networks

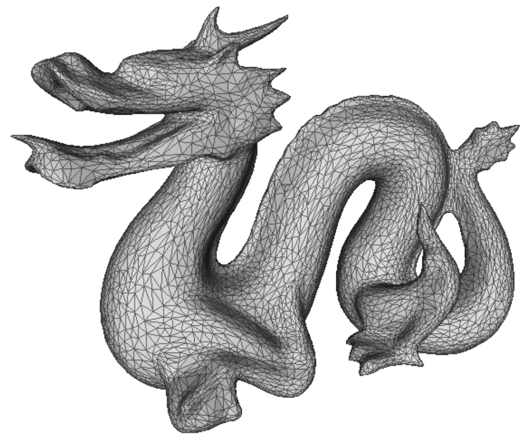


Graph Convolutional Network

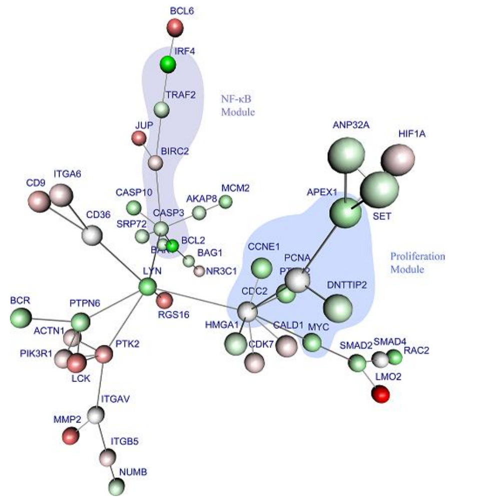
Science and Engineering



Robotics

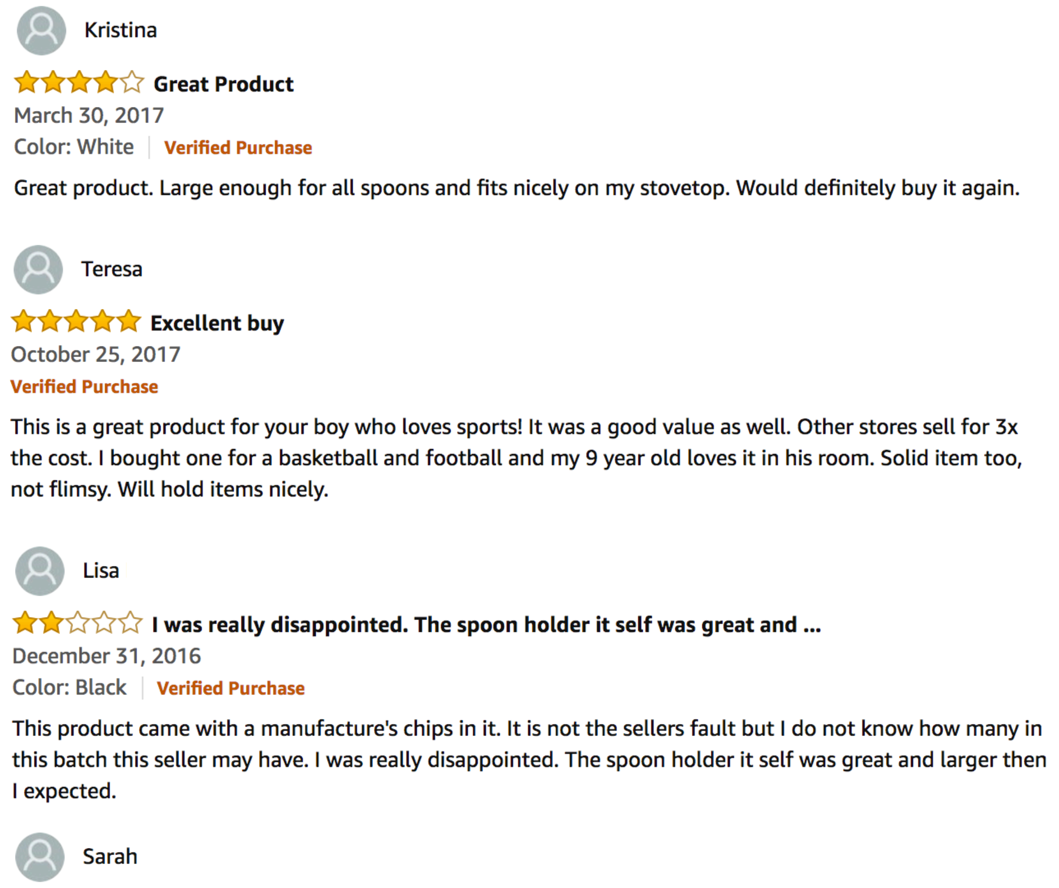


Simulations



Computational Biology

amazon



Kristina

★★★★☆ **Great Product**

March 30, 2017

Color: White | Verified Purchase

Great product. Large enough for all spoons and fits nicely on my stovetop. Would definitely buy it again.

Teresa

★★★★★ **Excellent buy**

October 25, 2017

Verified Purchase

This is a great product for your boy who loves sports! It was a good value as well. Other stores sell for 3x the cost. I bought one for a basketball and football and my 9 year old loves it in his room. Solid item too, not flimsy. Will hold items nicely.

Lisa

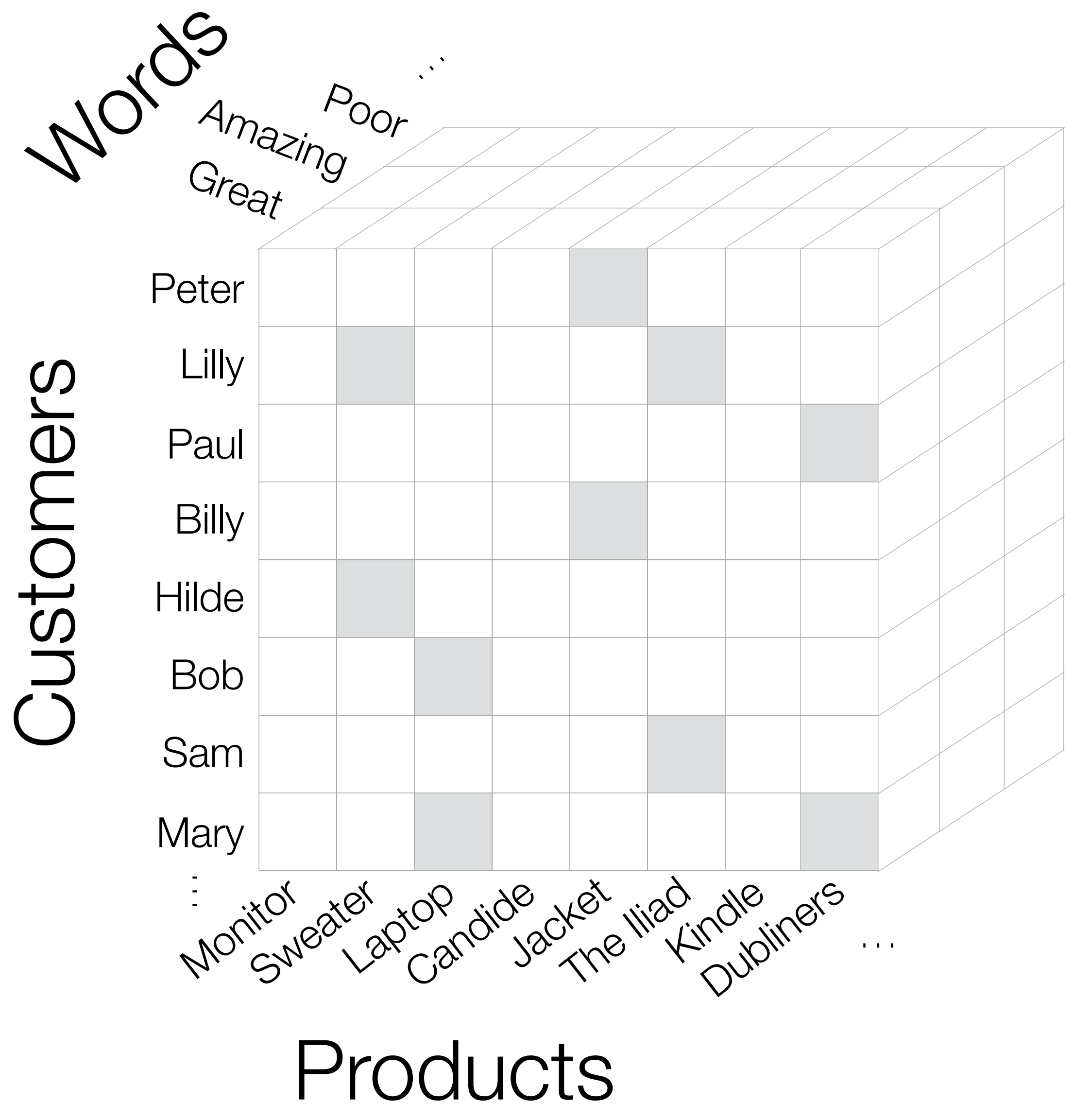
★★☆☆☆ **I was really disappointed. The spoon holder it self was great and ...**

December 31, 2016

Color: Black | Verified Purchase

This product came with a manufacture's chips in it. It is not the sellers fault but I do not know how many in this batch this seller may have. I was really disappointed. The spoon holder it self was great and larger then I expected.

Sarah



For Example, Sparse Tensors Are Everywhere

Data Analytics

New TV to watch instantly

See all >

Avatar: The Last Airbender: Book 3 — Play — ★★★★★ — Not Interested

Weeds: Season 3 — Play — ★★★★½ — Not Interested

Arrested Development: Season 3 — Play — ★★★★★ — Not Interested

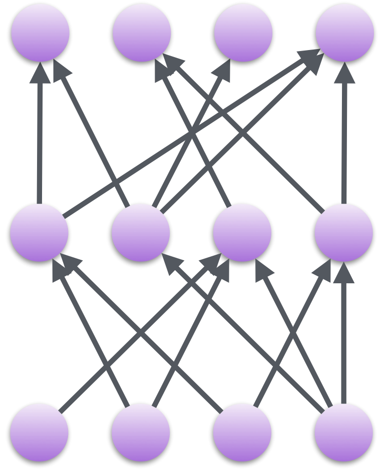
Sanctuary: Season 2 — Play — ★★★★½ — Not Interested

Bones: Season 3 — Play — ★★★★½ — Not Interested

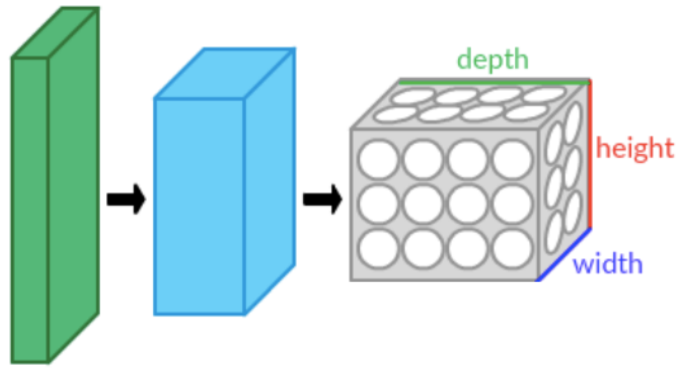
Movies

Social Networks

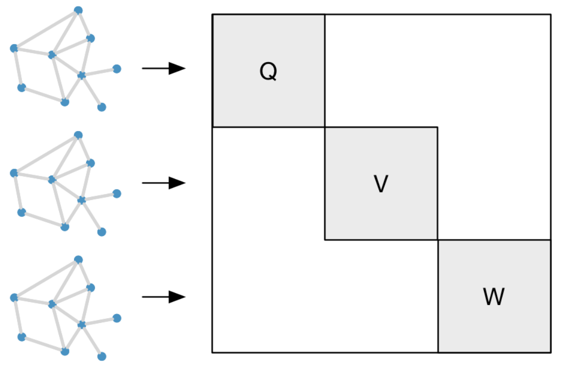
Machine Learning



Sparse Networks

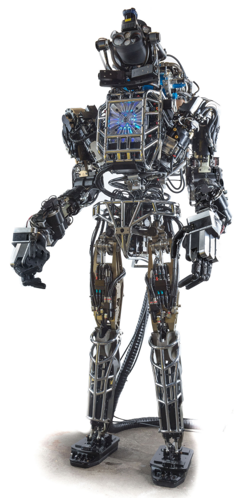


Sparse Convolutional Networks

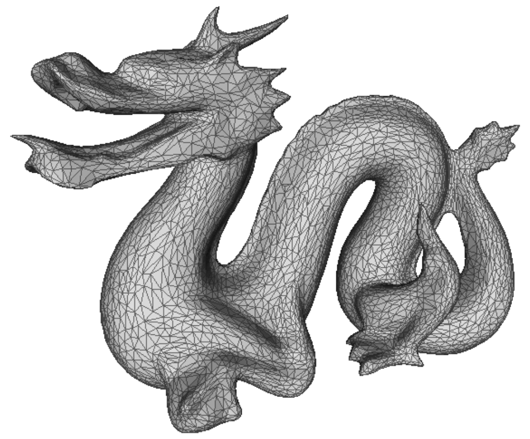


Graph Convolutional Network

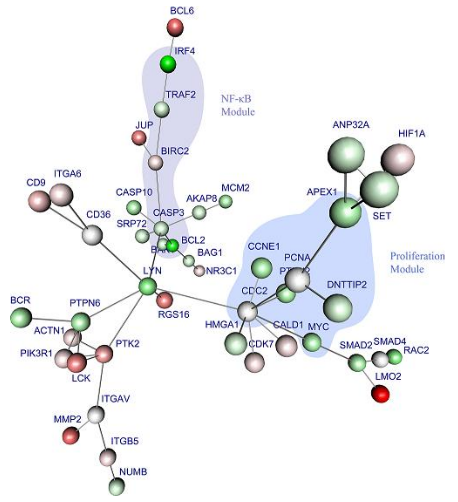
Science and Engineering



Robotics

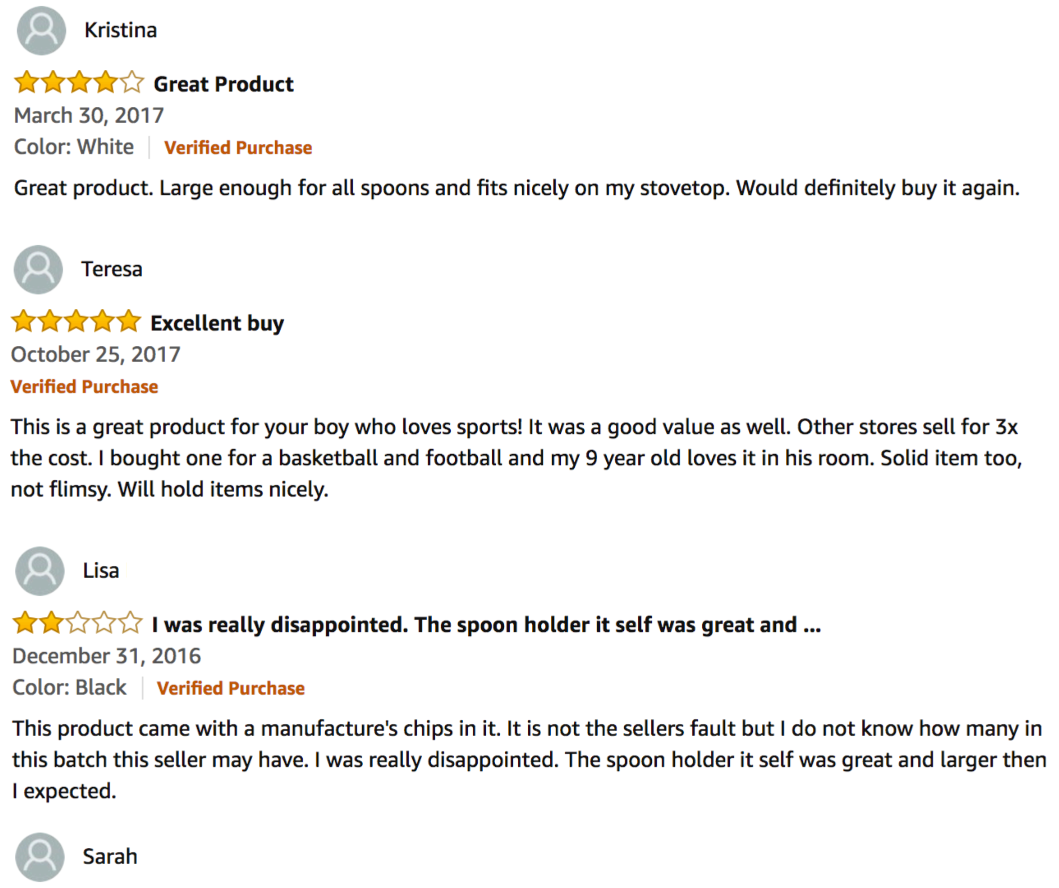


Simulations



Computational Biology

amazon



Kristina

★★★★☆ **Great Product**

March 30, 2017

Color: White | Verified Purchase

Great product. Large enough for all spoons and fits nicely on my stovetop. Would definitely buy it again.

Teresa

★★★★★ **Excellent buy**

October 25, 2017

Verified Purchase

This is a great product for your boy who loves sports! It was a good value as well. Other stores sell for 3x the cost. I bought one for a basketball and football and my 9 year old loves it in his room. Solid item too, not flimsy. Will hold items nicely.

Lisa

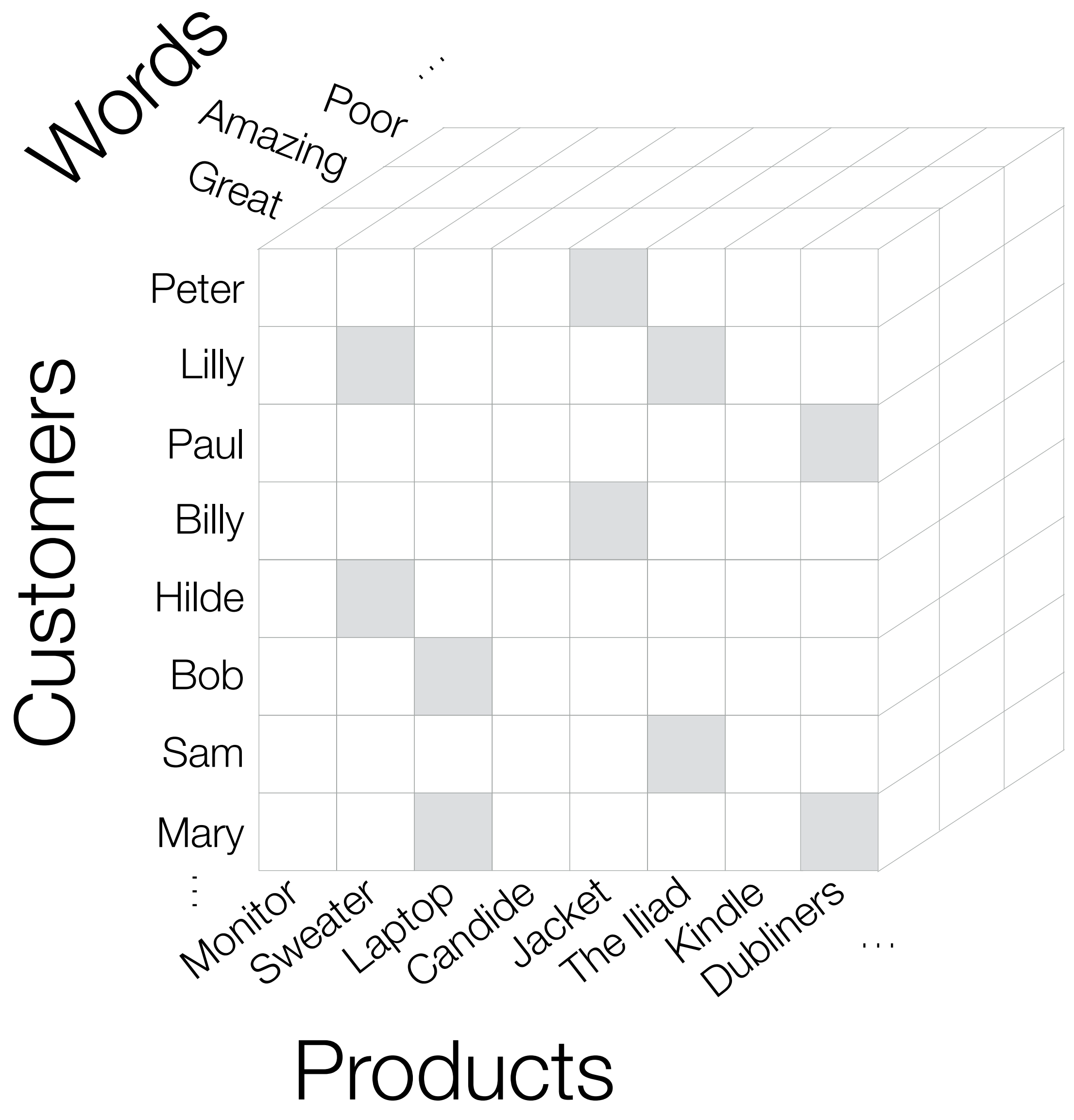
★★☆☆☆ **I was really disappointed. The spoon holder it self was great and ...**

December 31, 2016

Color: Black | Verified Purchase

This product came with a manufacture's chips in it. It is not the sellers fault but I do not know how many in this batch this seller may have. I was really disappointed. The spoon holder it self was great and larger then I expected.

Sarah



Extremely sparse

Dense storage: 107 Exabytes

Sparse storage: 13 Gigabytes

Dense Tensors Are Flexible But Can Waste Memory

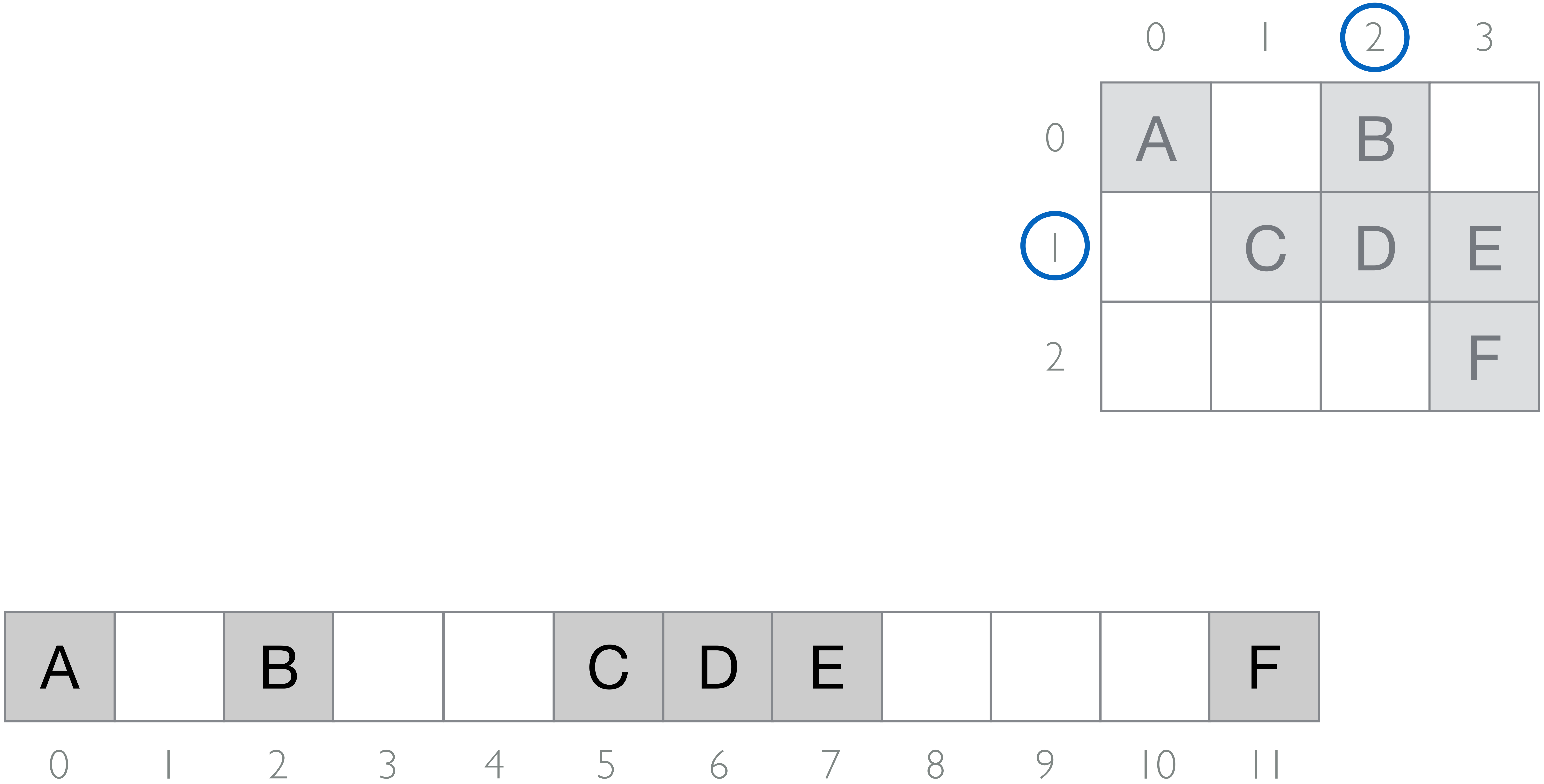
	0	1	2	3
0	A		B	
1		C	D	E
2				F

Dense Tensors Are Flexible But Can Waste Memory

	0	1	2	3
0	A		B	
1		C	D	E
2				F

A		B			C	D	E				F
0	1	2	3	4	5	6	7	8	9	10	11

Dense Tensors Are Flexible But Can Waste Memory

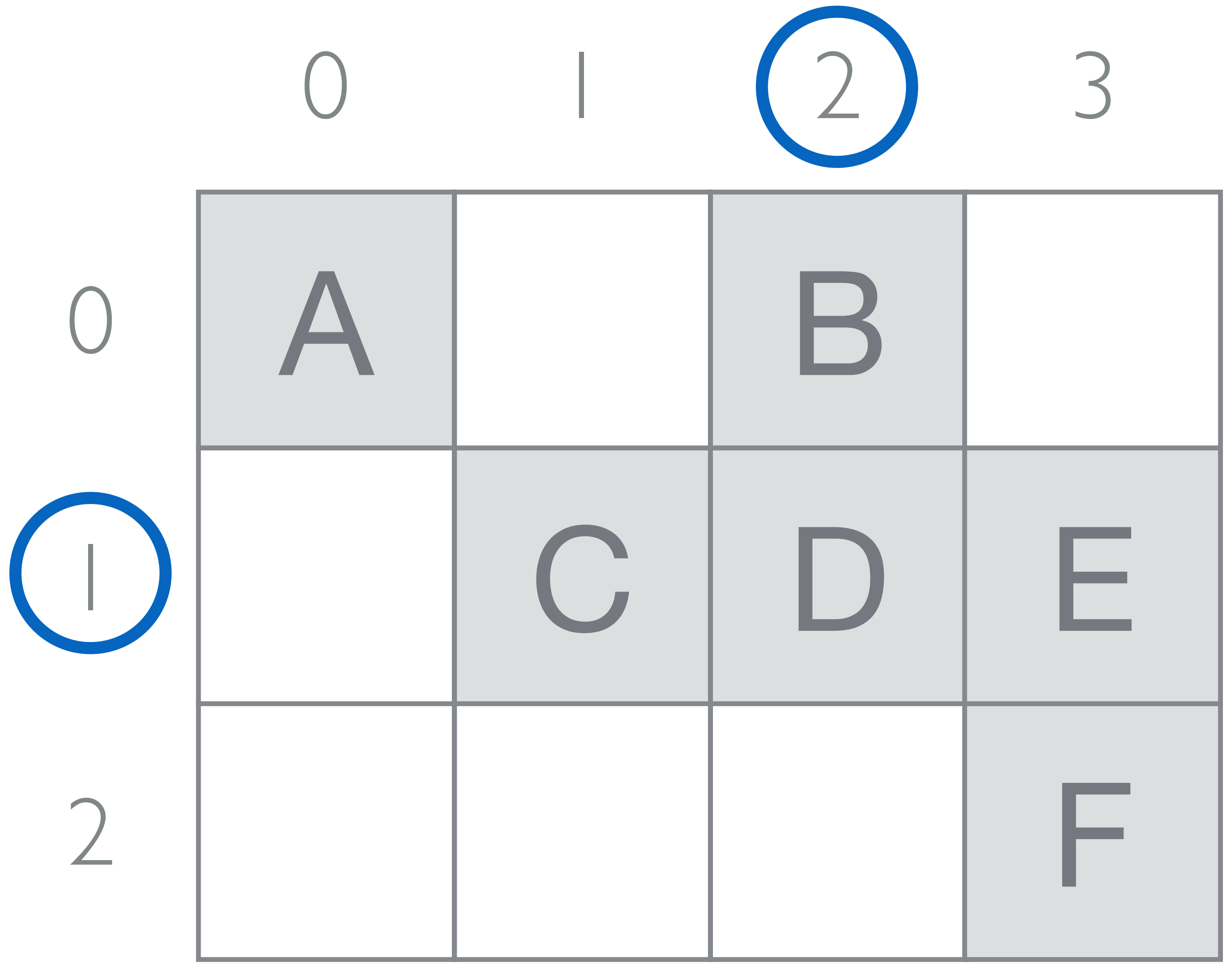


	0	1	2	3
0	A		B	
1		C	D	E
2				F

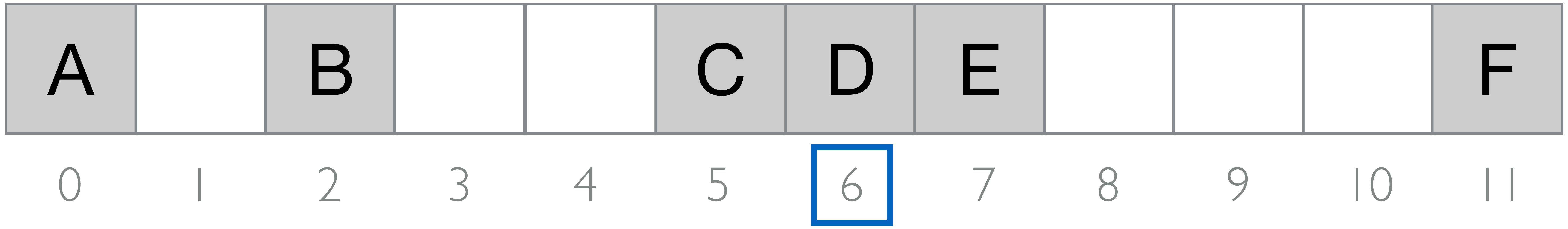
0	1	2	3	4	5	6	7	8	9	10	11
A		B			C	D	E				F

Dense Tensors Are Flexible But Can Waste Memory

```
locate(1,2) = 1*4 + 2
            = 6
```



	0	1	2	3
0	A		B	
1		C	D	E
2				F



A		B			C	D	E				F
0	1	2	3	4	5	6	7	8	9	10	11

Sparse Tensors Can Be Compressed By Adding Metadata

	0	1	2	3
0	A		B	
1		C	D	E
2				F

A		B			C	D	E				F
0	1	2	3	4	5	6	7	8	9	10	11

Sparse Tensors Can Be Compressed By Adding Metadata

	0	1	2	3
0	A		B	
1		C	D	E
2				F

A	B	C	D	E	F
0	1	2	3	4	5

Sparse Tensors Can Be Compressed By Adding Metadata

```
row(3) = ???
col(3) = ???
```

	0	1	2	3
0	A		B	
1		C	D	E
2				F

A	B	C	D	E	F
0	1	2	3	4	5

Sparse Tensors Can Be Compressed By Adding Metadata

Coordinate

rows	0	0	1	1	1	2
cols	0	2	1	2	3	3

A	B	C	D	E	F
0	1	2	3	4	5

	0	1	2	3
0	A		B	
1		C	D	E
2				F

Sparse Tensors Can Be Compressed By Adding Metadata

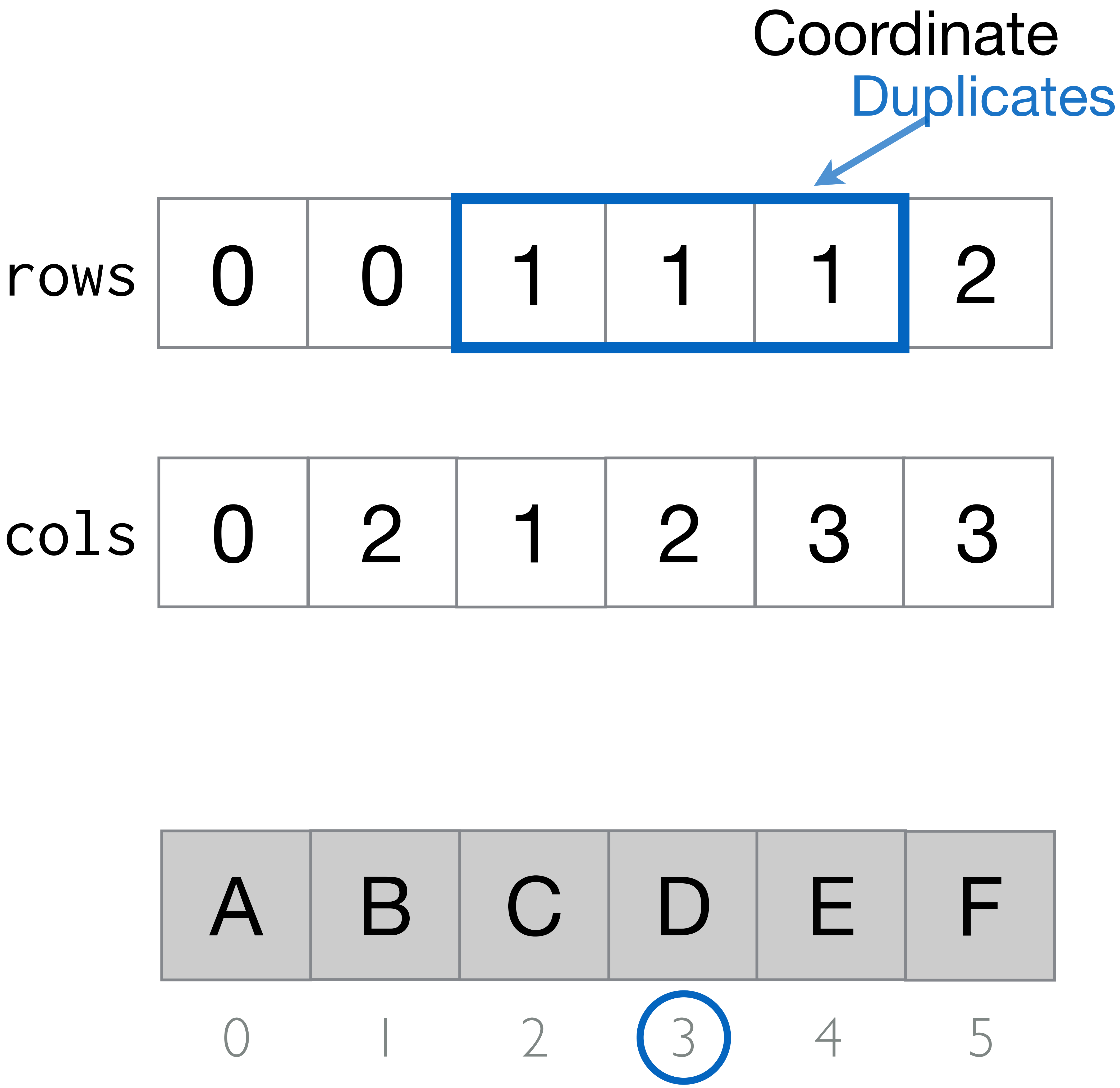
Coordinate

rows	0	0	1	1	1	2
cols	0	2	1	2	3	3

A	B	C	D	E	F
0	1	2	3	4	5

	0	1	2	3
0	A		B	
1		C	D	E
2				F

Sparse Tensors Can Be Compressed By Adding Metadata



Coordinate

Duplicates

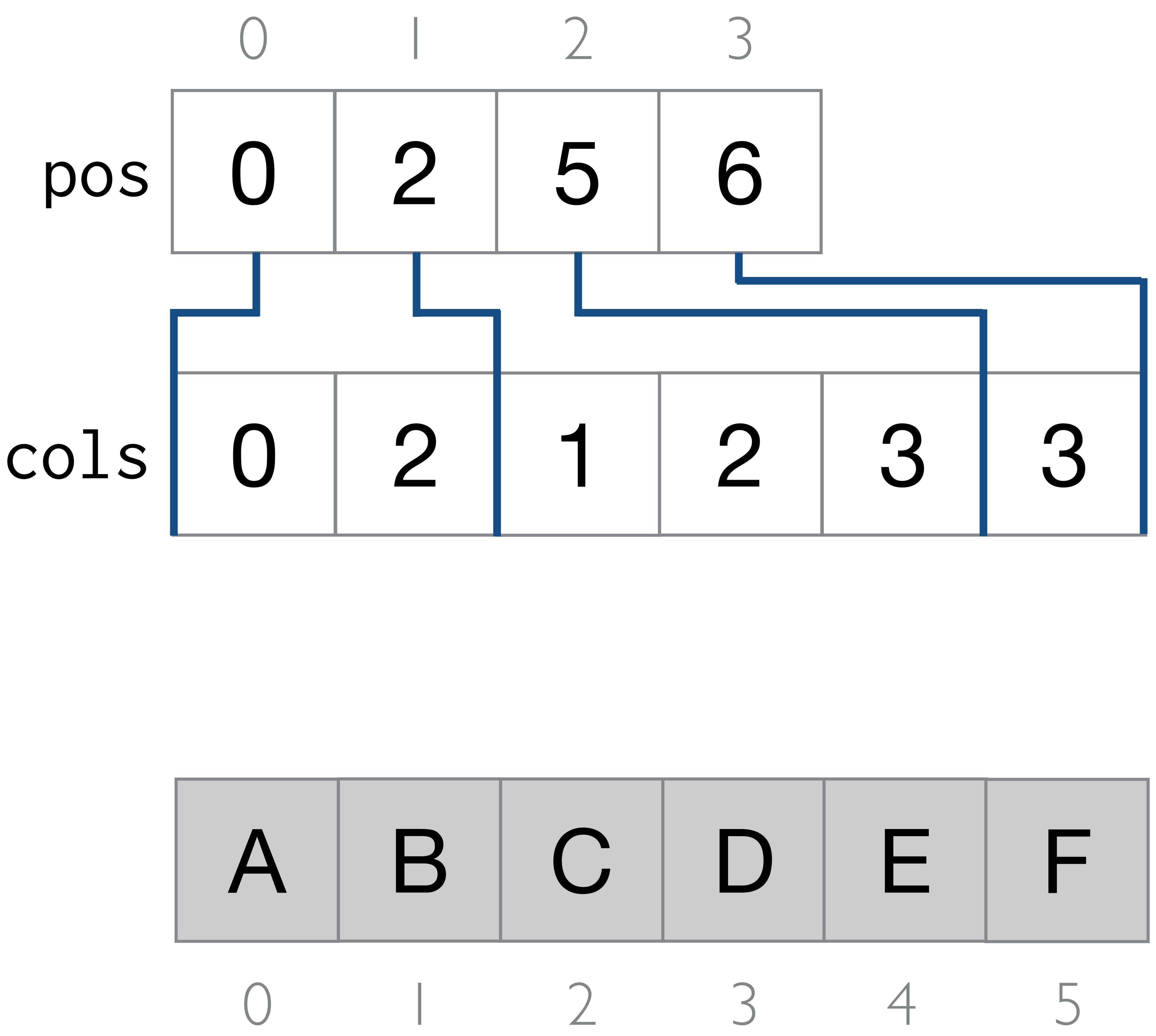
rows	0	0	1	1	1	2
cols	0	2	1	2	3	3

A	B	C	D	E	F
0	1	2	3	4	5

	0	1	2	3
0	A		B	
1		C	D	E
2				F

Sparse Tensors Can Be Compressed By Adding Metadata

Compressed Sparse Rows (CSR)



	0	1	2	3		
pos	0	2	5	6		
cols	0	2	1	2	3	3

A	B	C	D	E	F
0	1	2	3	4	5

	0	1	2	3
0	A		B	
1		C	D	E
2				F

Complexity Of Sparse Tensors & Code

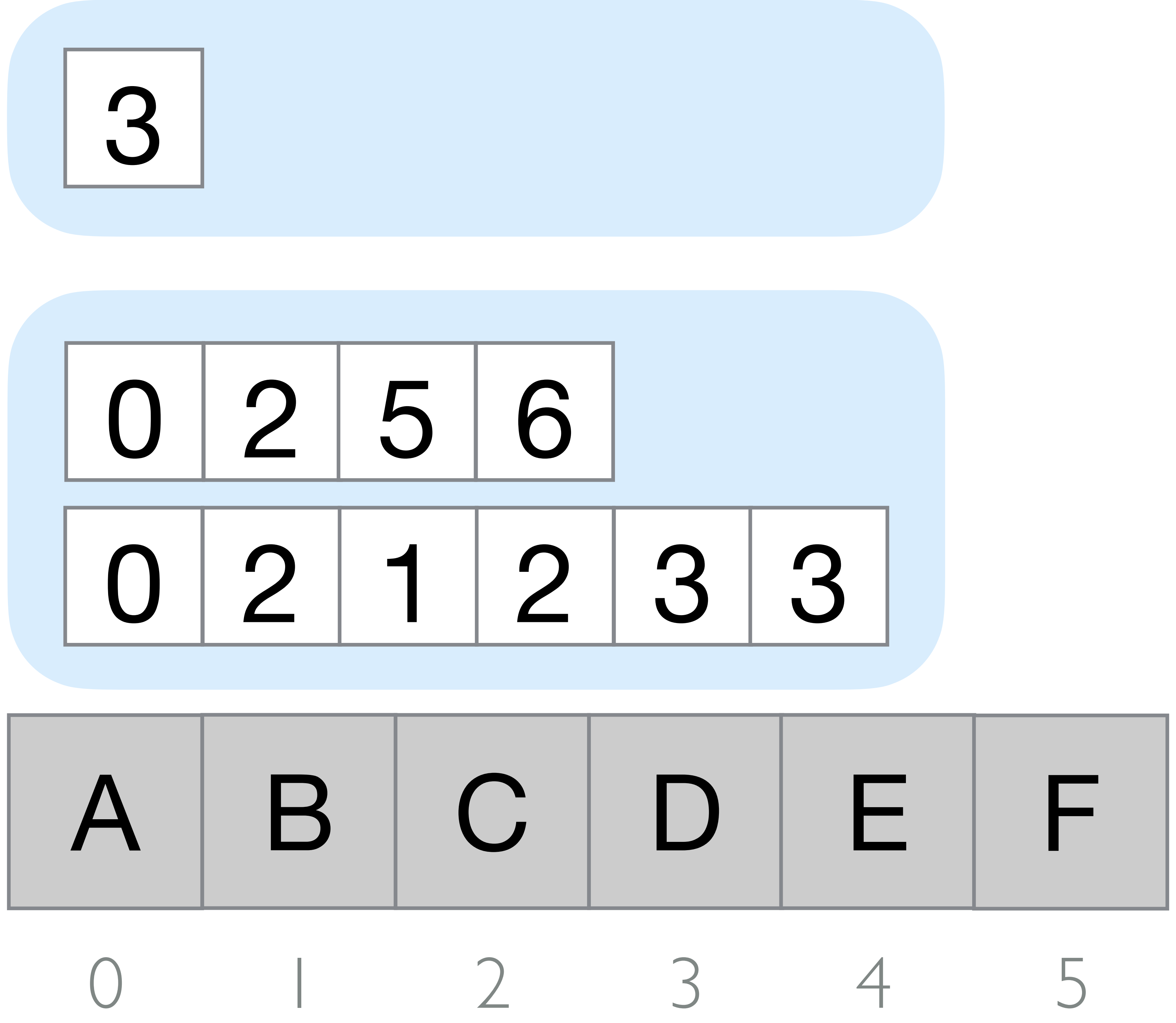
$$A_{ij} = \sum_k \underset{\overset{\uparrow}{\text{dense}}}{B_{ijk}} \underset{\overset{\uparrow}{\text{dense}}}{c_k}$$

```
for (int i = 0; i < m; i++) {

  for (int j = 0; j < n; j++) {
    int pB2 = i*n + j;
    int pA2 = i*n + j;
    double t = 0.0;
    for (int k = 0; k < o; k++) {
      int pB3 = pB2*o + k;
      t += B[pB3] * c[k];
    }
    A[pA2] = t;
  }
}
```

A		B			C	D	E				F
0	1	2	3	4	5	6	7	8	9	10	11

Complexity Of Sparse Tensors & Code

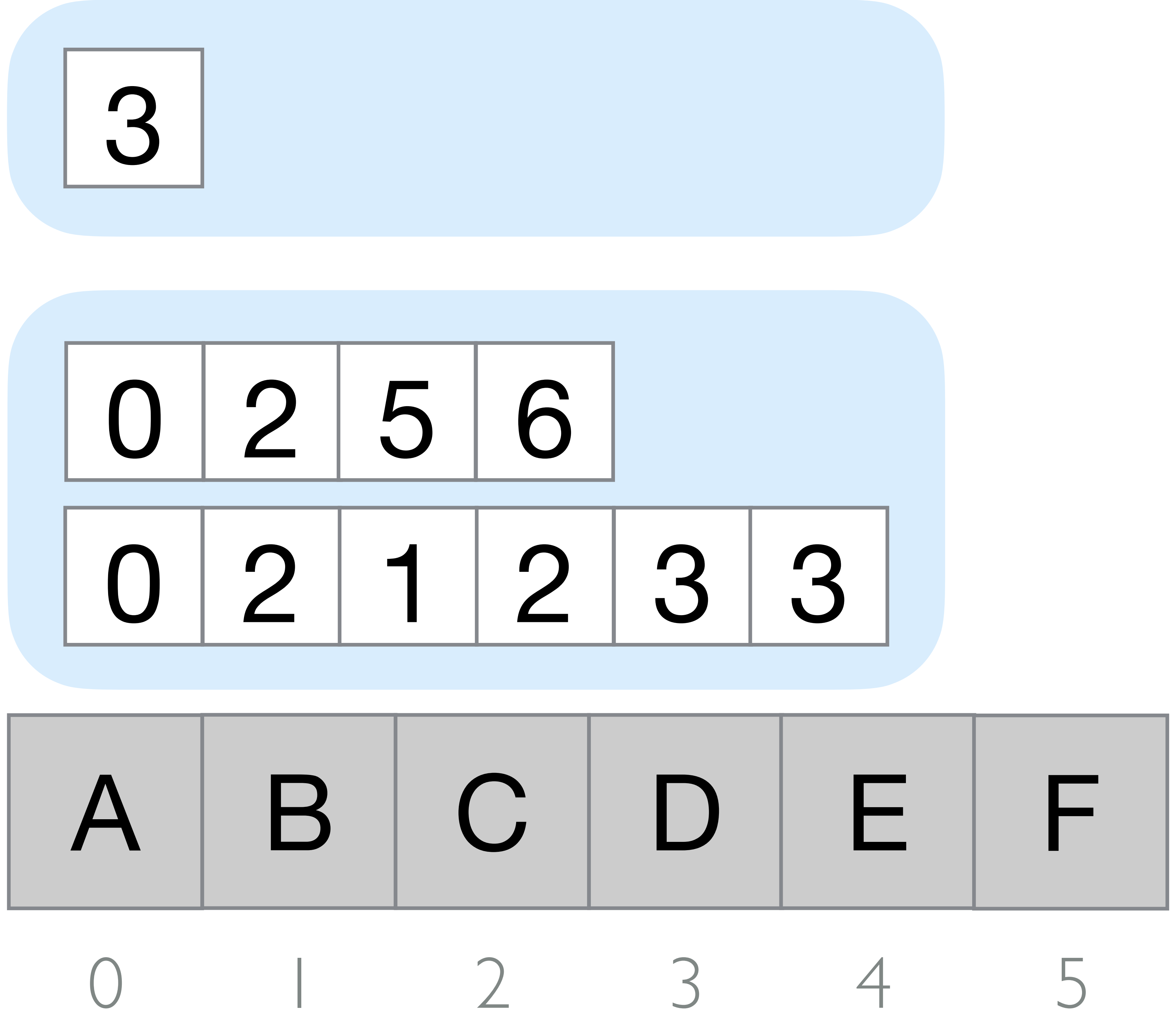
$$A_{ij} = \sum_k \underset{\text{CSF}}{\underset{\uparrow}{B_{ijk}}} \underset{\text{dense}}{\underset{\uparrow}{c_k}}$$



```
for (int pA = 0; pA < m*n; pA++) {
  A[pA] = 0.0;
}
for (int pB1 = B1_pos[0]; pB1 < B1_pos[1]; pB1++) {
  int i = B1_crd[pB1];
  for (int pB2 = B2_pos[pB1]; pB2 < B2_pos[pB1+1]; pB2++) {
    int j = B2_crd[pB2];
    int pA2 = i*n + j;
    double t = 0.0;
    for (int pB3 = B3_pos[pB2]; pB3 < B3_pos[pB2+1]; pB3++) {
      int k = B3_crd[pB3];
      t += B[pB3] * c[k];
    }
    A[pA2] = t;
  }
}
```

Complexity Of Sparse Tensors & Code

$$A_{ij} = \sum_k \underset{\text{CSF}}{\underset{\uparrow}{B_{ijk}}}\,\underset{\text{compressed}}{\underset{\uparrow}{c_k}}$$



3

0 2 5 6

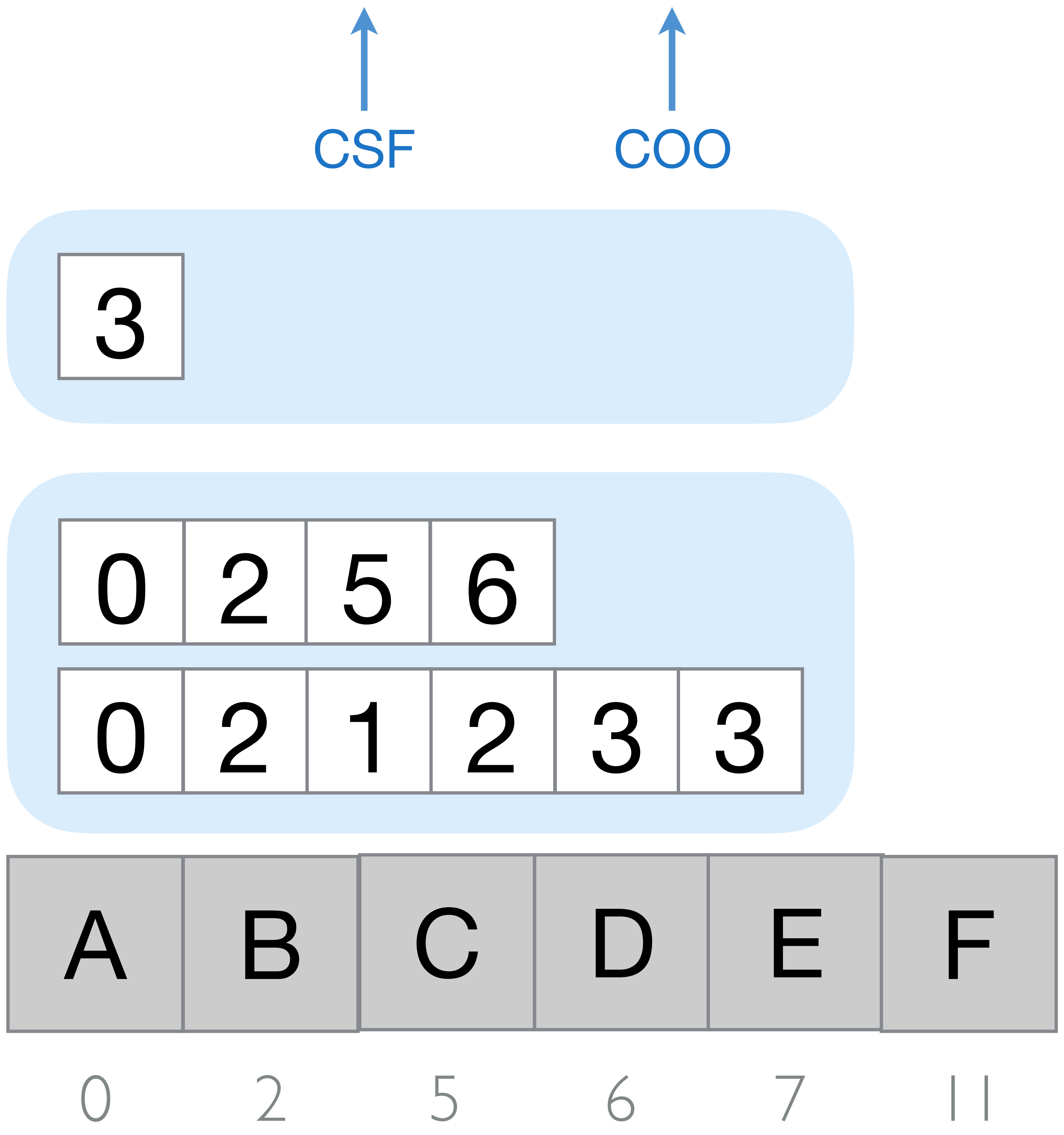
0 2 1 2 3 3

A	B	C	D	E	F
0	1	2	3	4	5

```
for (int pA = 0; pA < m*n; pA++) {
  A[pA] = 0.0;
}
for (int pB1 = B1_pos[0]; pB1 < B1_pos[1]; pB1++) {
  int i = B1_crd[pB1];
  for (int pB2 = B2_pos[pB1]; pB2 < B2_pos[pB1+1]; pB2++) {
    int j = B2_crd[pB2];
    int pA2 = i*n + j;
    double t = 0.0;
    int pB3 = B3_pos[pB2];
    int pc1 = c1_pos[0];
    while (pB3 < B3_pos[pB2+1] && pc1 < c1_pos[1]) {
      int kB = B3_crd[pB3];
      int kc = c1_crd[pc1];
      int k = min(kB, kc);
      if (kB == k && kc == k) {
        t += B[pB3] * c[pc1];
      }
      pB3 += (int)(kB == k);
      pc1 += (int)(kc == k);
    }
    A[pA2] = t;
  }
}
```

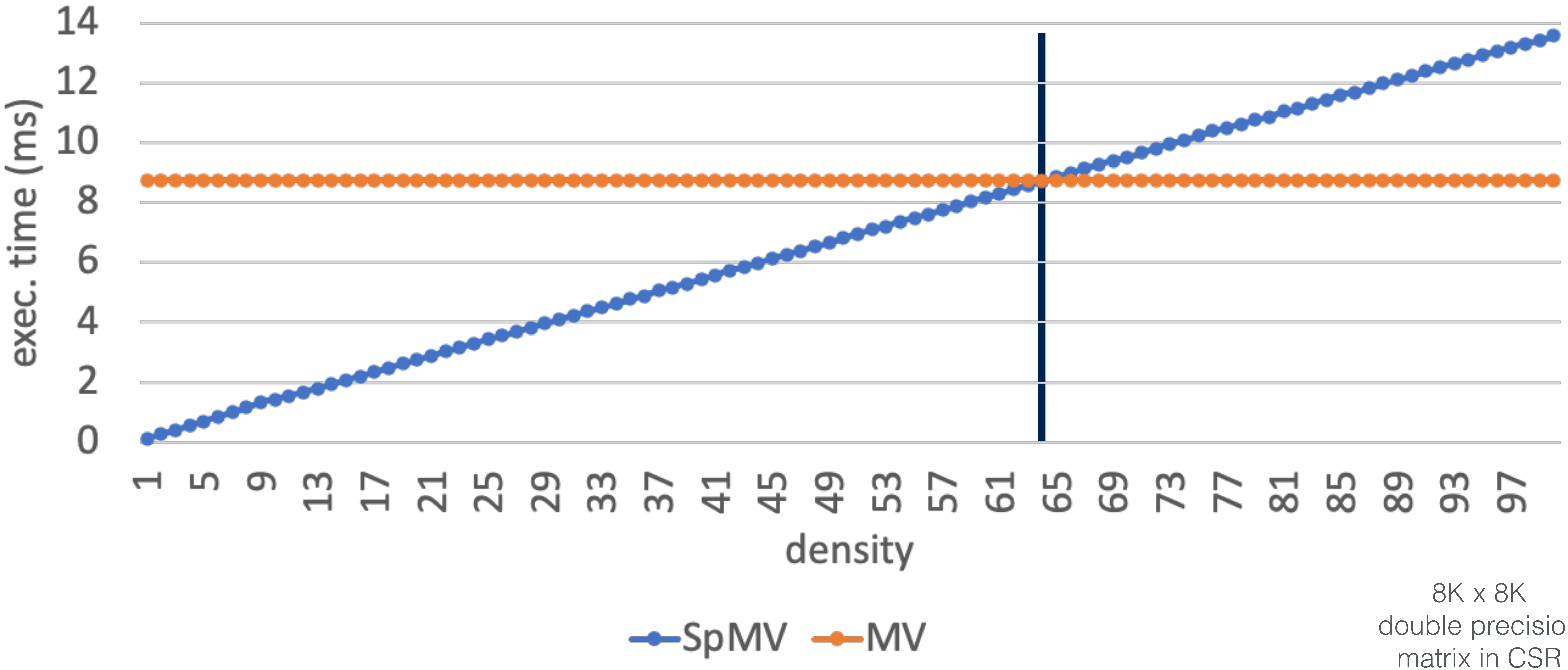
Complexity Of Sparse Tensors & Code

$$A_{ijk} = B_{ijk} + C_{ijk}$$



```c
int iB = 0;
int C0_pos = C0_pos[0];
while (C0_pos < C0_pos[1]) {
  int iC = C0_crd[C0_pos];
  int C0_end = C0_pos + 1;
  if (iC == iB)
    while ((C0_end < C0_pos[1]) && (C0_crd[C0_end] == iB)) {
      C0_end++;
    }
  if (iC == iB) {
    int B1_pos = B1_pos[iB];
    int C1_pos = C0_pos;
    while ((B1_pos < B1_pos[iB + 1]) && (C1_pos < C0_end)) {
      int jB = B1_crd[B1_pos];
      int jC = C1_crd[C1_pos];
      int j = min(jB, jC);
      int A1_pos = (iB * A1_size) + j;
      int C1_end = C1_pos + 1;
      if (jC == j)
        while ((C1_end < C0_end) && (C1_crd[C1_end] == j)) {
          C1_end++;
        }
      if ((jB == j) && (jC == j)) {
        int B2_pos = B2_pos[B1_pos];
        int C2_pos = C1_pos;
        while ((B2_pos < B2_pos[B1_pos + 1]) && (C2_pos < C1_end)) {
          int kB = B2_crd[B2_pos];
          int kC = C2_crd[C2_pos];
          int k = min(kB, kC);
          int A2_pos = (A1_pos * A2_size) + k;
          if ((kB == k) && (kC == k)) {
            A[A2_pos] = B[B2_pos] + C[C2_pos];
          } else if (kB == k) {
            A[A2_pos] = B[B2_pos];
          } else {
            A[A2_pos] = C[C2_pos];
          }
          if (kB == k) B2_pos++;
          if (kC == k) C2_pos++;
        }
        while (B2_pos < B2_pos[B1_pos + 1]) {
          int kB0 = B2_crd[B2_pos];
          int A2_pos0 = (A1_pos * A2_size) + kB0;
          A[A2_pos0] = B[B2_pos];
          B2_pos++;
        }
        while (C2_pos < C1_end) {
          int kC0 = C2_crd[C2_pos];
          int A2_pos1 = (A1_pos * A2_size) + kC0;
          A[A2_pos1] = C[C2_pos];
          C2_pos++;
        }
      } else if (jB == j) {
        for (int B2_pos0 = B2_pos[B1_pos];
                 B2_pos0 < B2_pos[B1_pos + 1]; B2_pos0++) {
          int kB1 = B2_crd[B2_pos0];
          int A2_pos2 = (A1_pos * A2_size) + kB1;
          A[A2_pos2] = B[B2_pos0];
        }
      } else {
        for (int C2_pos0 = C1_pos; C2_pos0 < C1_end; C2_pos0++) {
          int kC1 = C2_crd[C2_pos0];
          int A2_pos3 = (A1_pos * A2_size) + kC1;
          A[A2_pos3] = C[C2_pos0];
        }
      }
      if (jB == j) B1_pos++;
      if (jC == j) C1_pos = C1_end;
    }
    while (B1_pos < B1_pos[iB + 1]) {
      int jB0 = B1_crd[B1_pos];
      int A1_pos0 = (iB * A1_size) + jB0;
      for (int B2_pos1 = B2_pos[B1_pos];
               B2_pos1 < B2_pos[B1_pos + 1]; B2_pos1++) {
        int kB2 = B2_crd[B2_pos1];
        int A2_pos4 = (A1_pos0 * A2_size) + kB2;
        A[A2_pos4] = B[B2_pos1];
      }
      B1_pos++;
    }
    while (C1_pos < C0_end) {
      int jC0 = C1_crd[C1_pos];
      int A1_pos1 = (iB * A1_size) + jC0;
      int C1_end0 = C1_pos + 1;
      while ((C1_end0 < C0_end) && (C1_crd[C1_end0] == jC0)) {
        C1_end0++;
      }
      for (int C2_pos1 = C1_pos; C2_pos1 < C1_end0; C2_pos1++) {
        int kC2 = C2_crd[C2_pos1];
        int A2_pos5 = (A1_pos1 * A2_size) + kC2;
        A[A2_pos5] = C[C2_pos1];
      }
      C1_pos = C1_end0;
    }
  } else {
    for (int B1_pos0 = B1_pos[iB];
             B1_pos0 < B1_pos[iB + 1]; B1_pos0++) {
      int jB1 = B1_crd[B1_pos0];
      int A1_pos2 = (iB * A1_size) + jB1;
      for (int B2_pos2 = B2_pos[B1_pos0];
               B2_pos2 < B2_pos[B1_pos0 + 1]; B2_pos2++) {
        int kB3 = B2_crd[B2_pos2];
        int A2_pos6 = (A1_pos2 * A2_size) + kB3;
        A[A2_pos6] = B[B2_pos2];
      }
    }
  }
  if (iC == iB) C0_pos = C0_end;
  iB++;
}
while (iB < B0_size) {
  for (int B1_pos1 = B1_pos[iB];
           B1_pos1 < B1_pos[iB + 1]; B1_pos1++) {
    int jB2 = B1_crd[B1_pos1];
    int A1_pos3 = (iB * A1_size) + jB2;
    for (int B2_pos3 = B2_pos[B1_pos1];
             B2_pos3 < B2_pos[B1_pos1 + 1]; B2_pos3++) {
      int kB4 = B2_crd[B2_pos3];
      int A2_pos7 = (A1_pos3 * A2_size) + kB4;
      A[A2_pos7] = B[B2_pos3];
    }
  }
  iB++;
}
```

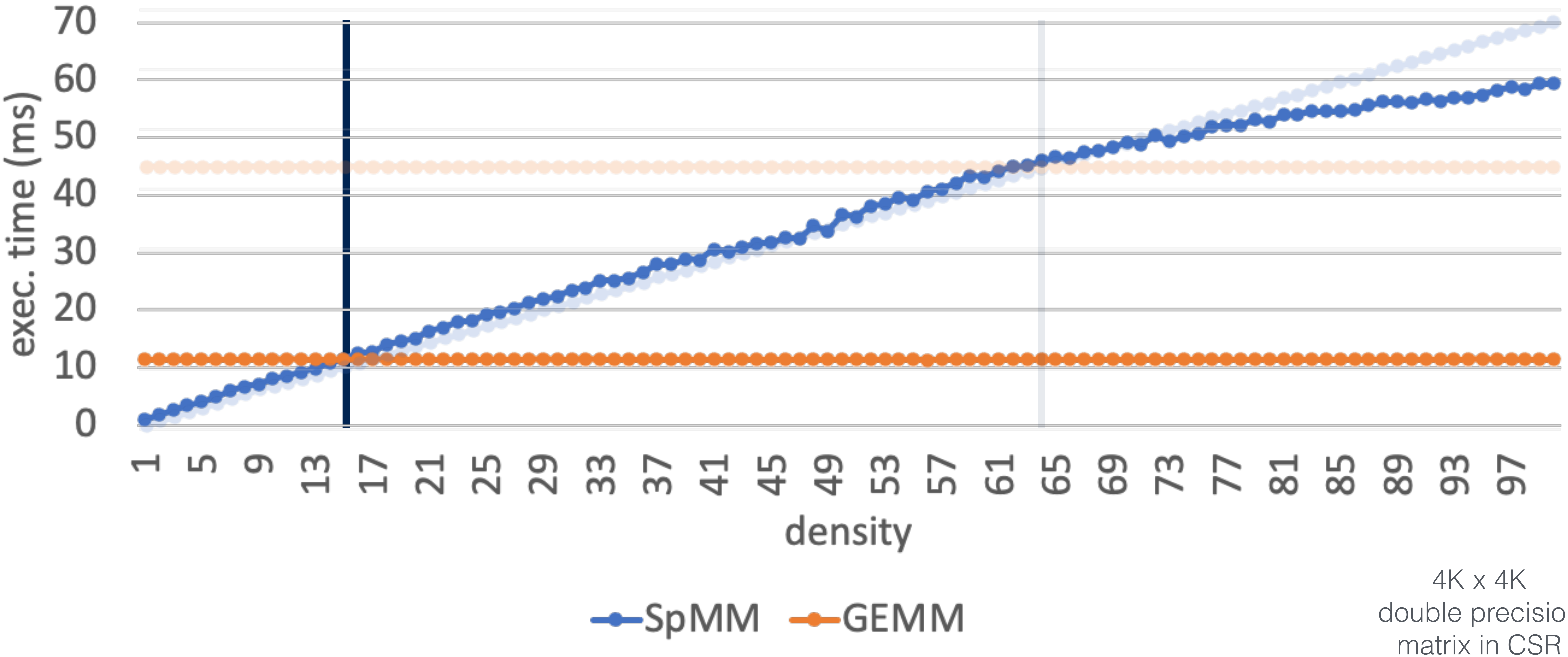
Ignoring Sparsity Is Throwing Away Performance



8K x 8K
double precision
matrix in CSR

Sparse Matrix Vector Multiplication (SpMV)

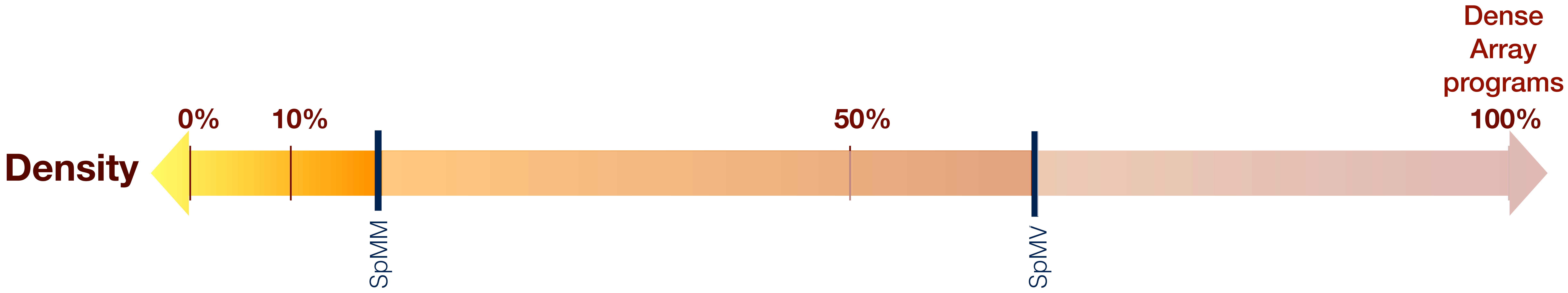
Ignoring Sparsity Is Throwing Away Performance



4K x 4K
double precision
matrix in CSR

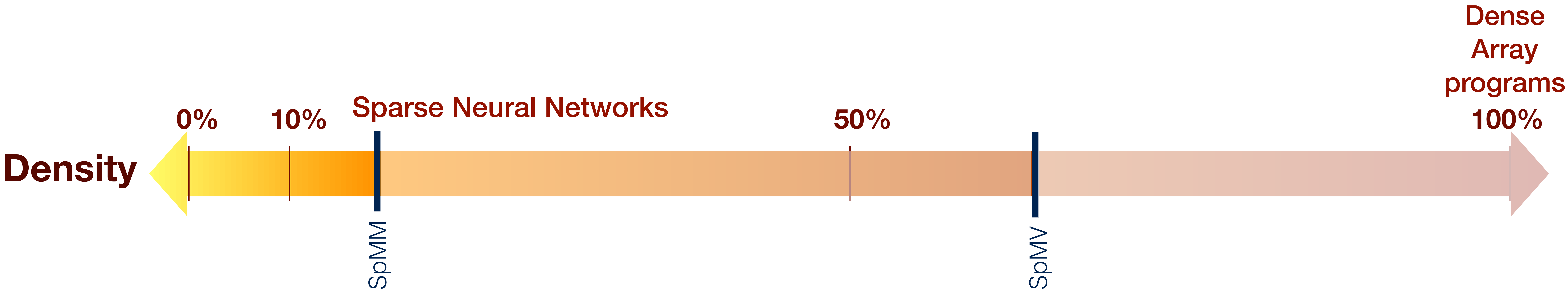
Sparse Matrix Matrix Multiplication (SpMV)

Sparse Problems Are Everywhere



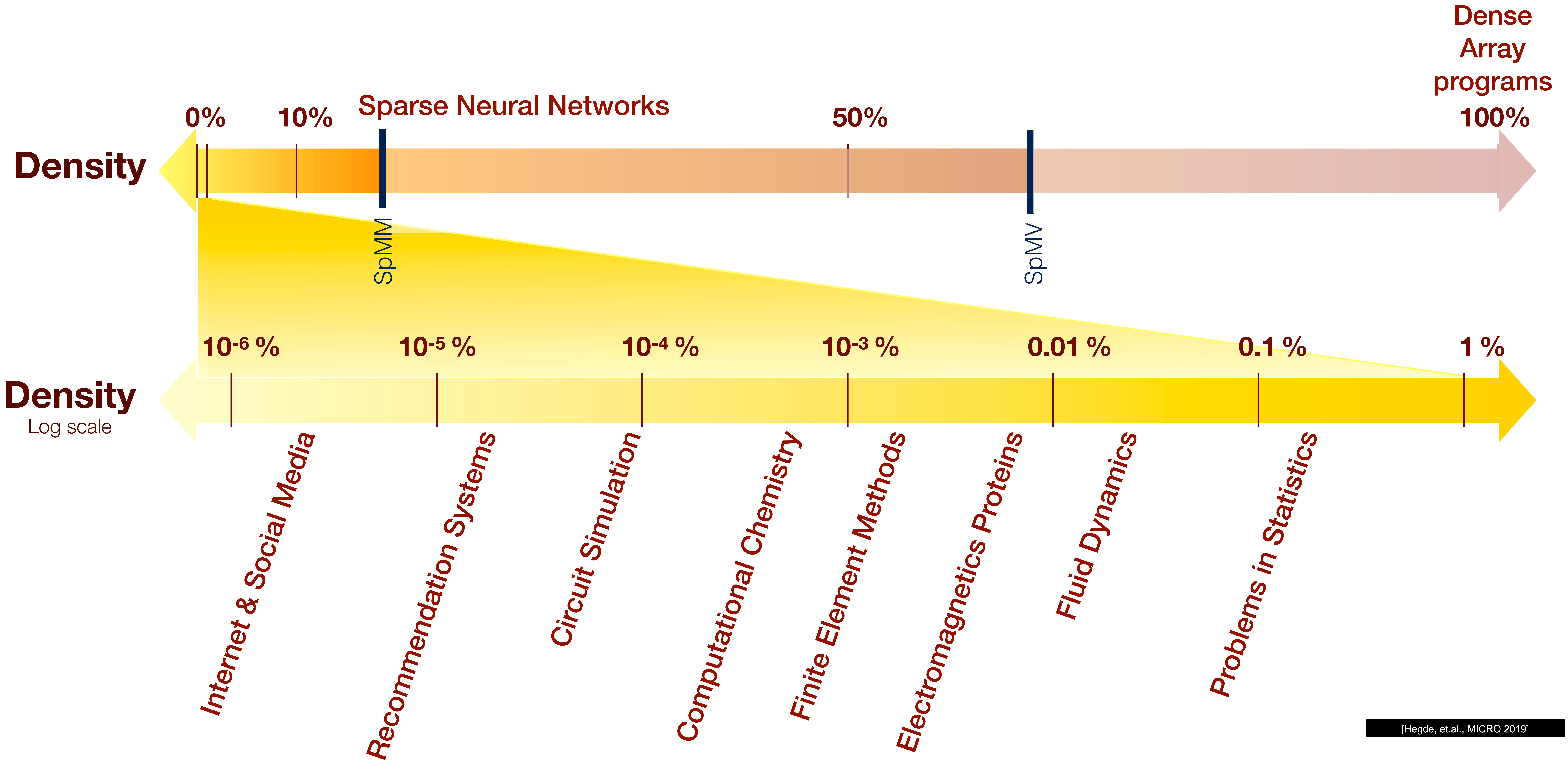
[Hegde, et.al., MICRO 2019]

Sparse Problems Are Everywhere



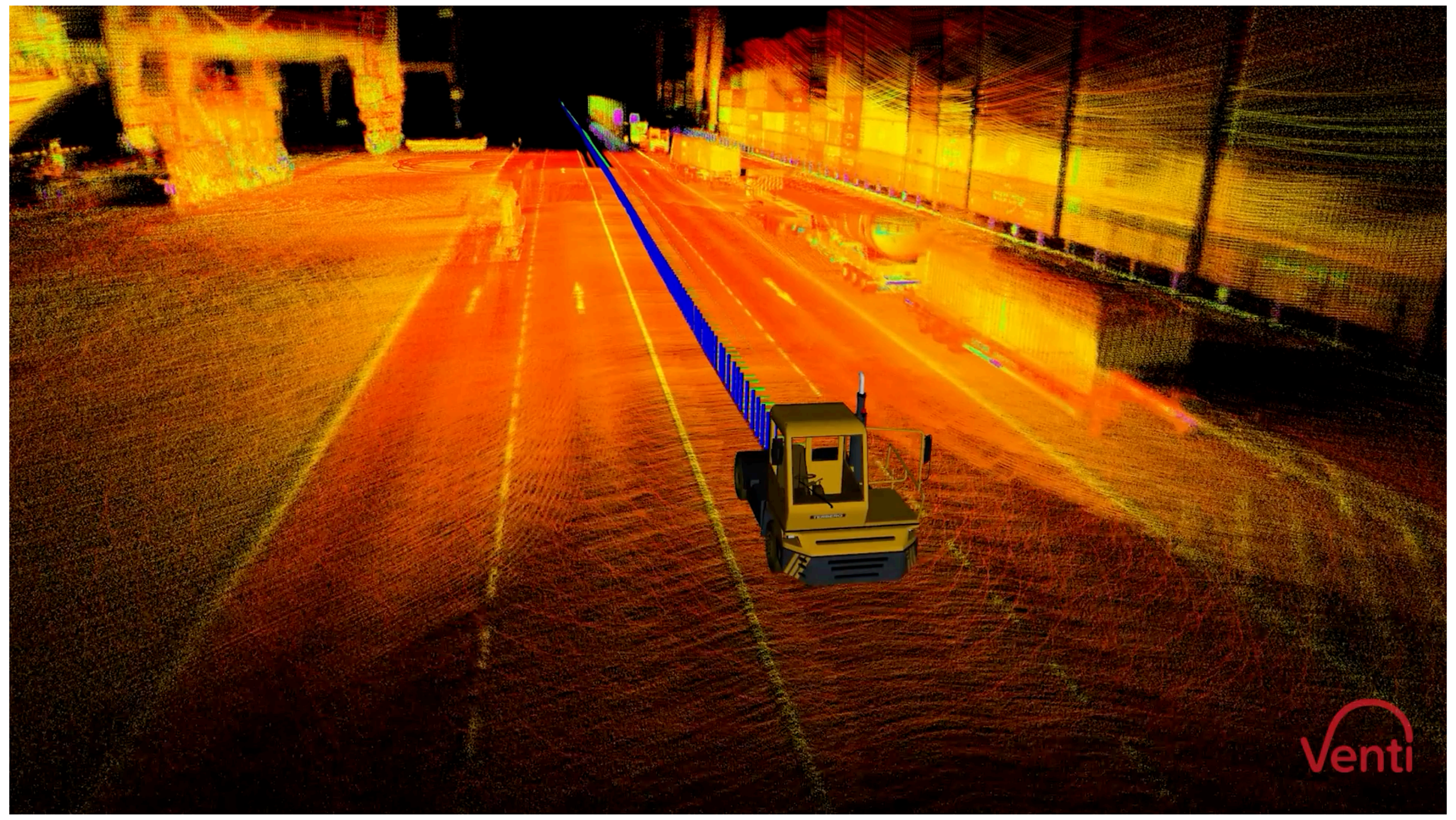
[Hegde, et.al., MICRO 2019]

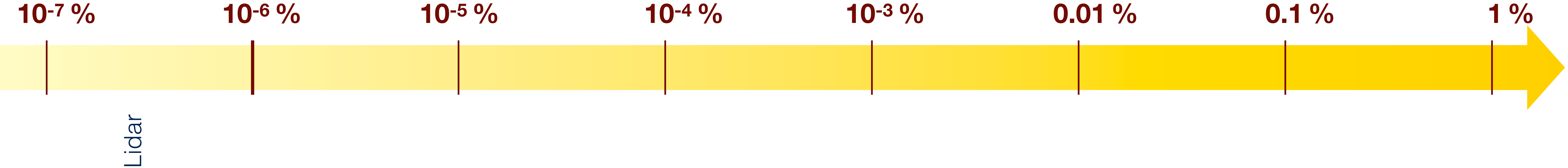
Sparse Problems Are Everywhere



[Hegde, et.al., MICRO 2019]

- X × Y × Z × (time)
- 8K × 8K × 8k
- But only 300,000 points
- Data density 0.00005859%





10^{-7} %

10^{-6} %

10^{-5} %

10^{-4} %

10^{-3} %

0.01 %

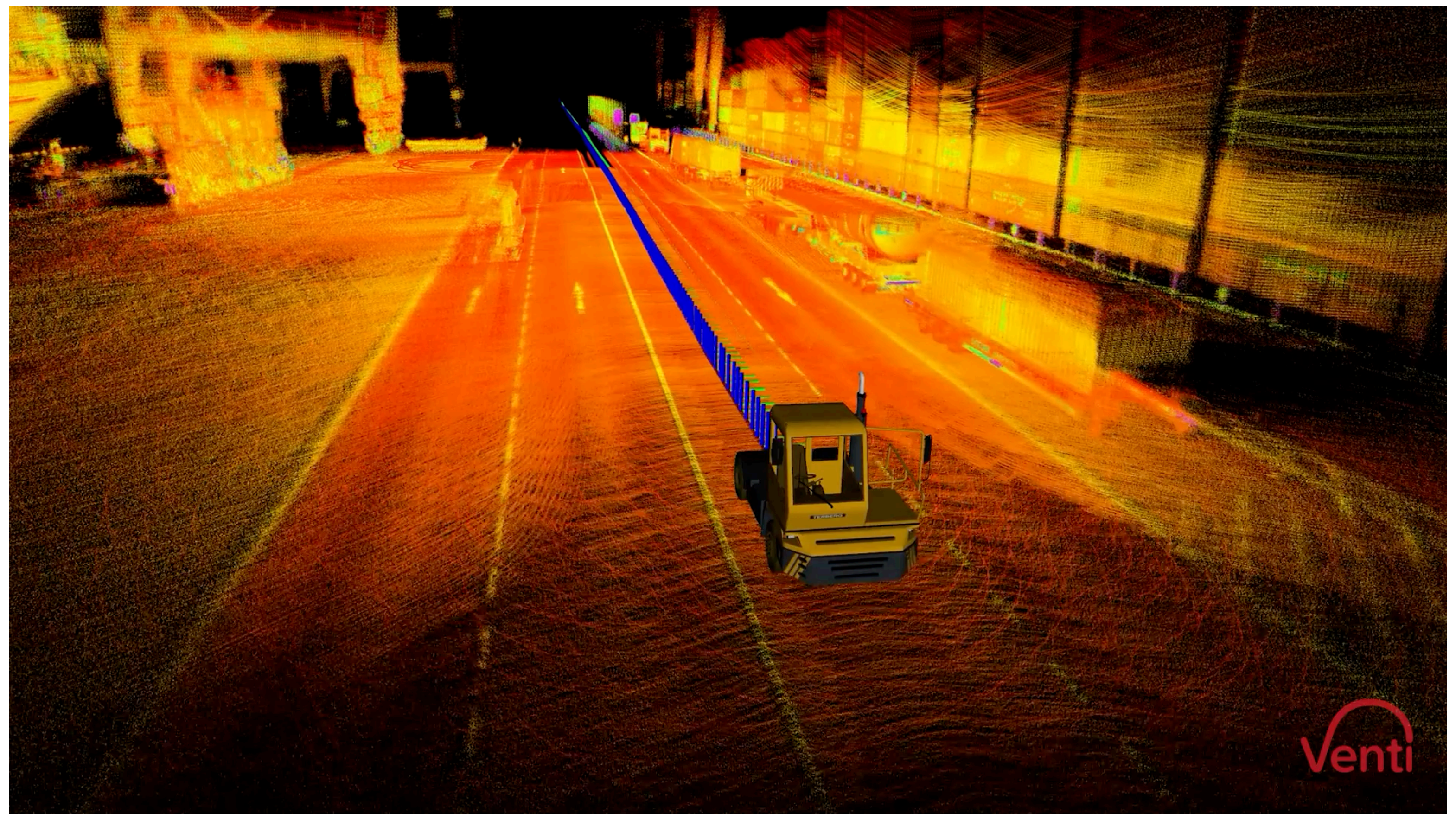
0.1 %

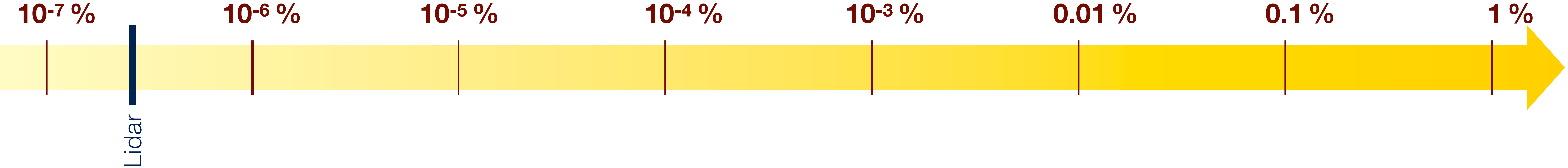
1 %

Lidar

Example: Sparsity In Lidar Data

- X × Y × Z × (time)
- 8K × 8K × 8k
- But only 300,000 points
- Data density 0.00005859%





10^{-7} %

10^{-6} %

10^{-5} %

10^{-4} %

10^{-3} %

0.01 %

0.1 %

1 %

Lidar

Sparsity Is Currently Addressed One-Problem-At-A-Time

$$a = Bc$$

$$a = Bc + a$$

$$a = Bc + b \quad A = B + C \quad a = \alpha Bc + \beta a$$

$$a = B^T c \quad A = \alpha B \quad a = B(c + d)$$

$$a = B^T c + d \quad A = B + C + D \quad A = BC$$

$$A = B \odot C \quad a = b \odot c \quad A = 0 \quad A = B \odot (CD)$$

$$A = BCd \quad A = B^T \quad a = B^T Bc$$

$$a = b + c \quad A = B \quad K = A^T CA$$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \quad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \quad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \quad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}} \, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}} \, \overline{P_{ip}}$$

Sparsity Is Currently Addressed One-Problem-At-A-Time

$a = Bc$

$a = Bc + a$

$a = Bc + b$ $\quad A = B + C$ $\quad a = \alpha Bc + \beta a$

$a = B^T c$ $\quad A = \alpha B$ $\quad a = B(c + d)$

$a = B^T c + d$ $\quad A = B + C + D$ $\quad A = BC$

$A = B \odot C$ $\quad a = b \odot c$ $\quad A = 0$ $\quad A = B \odot (CD)$

$A = BCd$ $\quad A = B^T$ $\quad a = B^T Bc$

$a = b + c$ $\quad A = B$ $\quad K = A^T CA$

Linear Algebra

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}} \, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}} \, \overline{P_{ip}}$$

Sparsity Is Currently Addressed One-Problem-At-A-Time

$a = Bc$

$a = Bc + a$

$a = Bc + b$ $\quad A = B + C$ $\quad a = \alpha Bc + \beta a$

$a = B^T c$ $\quad A = \alpha B$ $\quad a = B(c + d)$

$a = B^T c + d$ $\quad A = B + C + D$ $\quad A = BC$

$A = B \odot C$ $\quad a = b \odot c$ $\quad A = 0$ $\quad A = B \odot (CD)$

$A = BCd$ $\quad A = B^T$ $\quad a = B^T Bc$

$a = b + c$ $\quad A = B$ $\quad K = A^T CA$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

Data analytics
(tensor factorization)

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

Sparsity Is Currently Addressed One-Problem-At-A-Time

$a = Bc$

$a = Bc + a$

$a = Bc + b$ $A = B + C$ $a = \alpha Bc + \beta a$

$a = B^T c$ $A = \alpha B$ $a = B(c + d)$

$a = B^T c + d$ $A = B + C + D$ $A = BC$

$A = B \odot C$ $a = b \odot c$ $A = 0$ $A = B \odot (CD)$

$A = BCd$ $A = B^T$ $a = B^T Bc$

$a = b + c$ $A = B$ $K = A^T CA$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

Quantum Chromodynamics

Sparsity Is Currently Addressed One-Problem-At-A-Time

Eigen (SpMV)

$$a = Bc$$

$$a = Bc + a$$

$$a = Bc + b \qquad A = B + C \qquad a = \alpha Bc + \beta a$$

$$a = B^T c \qquad A = \alpha B \qquad a = B(c + d)$$

$$a = B^T c + d \qquad A = B + C + D \qquad A = BC$$

$$A = B \odot C \qquad a = b \odot c \qquad A = 0 \qquad A = B \odot (CD)$$

$$A = BCd \qquad A = B^T \qquad a = B^T Bc$$

$$a = b + c \qquad A = B \qquad K = A^T CA$$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

Sparsity Is Currently Addressed One-Problem-At-A-Time

Eigen (SpMV)

$a = Bc$

CSparse

$a = Bc + a$

OSKI

$a = Bc + b$ $\quad A = B + C \quad$ $a = \alpha Bc + \beta a$

PETSc $\quad a = B^T c \quad A = \alpha B \quad a = B(c + d)$

$a = B^T c + d \quad A = B + C + D \quad A = BC$

$A = B \odot C \quad a = b \odot c \quad A = 0 \quad A = B \odot (CD)$

$A = BCd \quad A = B^T \quad a = B^T Bc$

$a = b + c \quad A = B \quad K = A^T CA$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

Sparsity Is Currently Addressed One-Problem-At-A-Time

Eigen (SpMV)

$a = Bc$

CSparse

$a = Bc + a$

OSKI

OSKI has 282 specialized variants of this expression

$a = Bc + b$ $\quad A = B + C \quad a = \alpha B c + \beta a$

PETSc $\quad a = B^T c \quad A = \alpha B \quad a = B(c + d)$

$a = B^T c + d \quad A = B + C + D \quad A = BC$

$A = B \odot C \quad a = b \odot c \; A = 0 \quad A = B \odot (CD)$

$A = BCd \quad A = B^T \quad a = B^T B c$

$a = b + c \quad A = B \quad K = A^T C A$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

Sparsity Is Currently Addressed One-Problem-At-A-Time

$a = Bc$

$a = Bc + a$

$a = Bc + b$ $A = B + C$ $a = \alpha Bc + \beta a$

$a = B^T c$ $A = \alpha B$ $a = B(c + d)$

$a = B^T c + d$ $A = B + C + D$ $A = BC$

$A = B \odot C$ $a = b \odot c$ $A = 0$ $A = B \odot (CD)$

$A = BCd$ $A = B^T$ $a = B^T Bc$

$a = b + c$ $A = B$ $K = A^T CA$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

×

Dense Matrix

CSR DCSR BCSR

COO ELLPACK CSB

Blocked COO CSC

DIA Blocked DIA DCSC

Sparsity Is Currently Addressed One-Problem-At-A-Time

$a = Bc$

$a = Bc + a$

$a = Bc + b$ $\quad A = B + C$ $\quad a = \alpha Bc + \beta a$

$a = B^T c$ $\quad A = \alpha B$ $\quad a = B(c + d)$

$a = B^T c + d$ $\quad A = B + C + D$ $\quad A = BC$

$A = B \odot C$ $\quad a = b \odot c$ $A = 0$ $\quad A = B \odot (CD)$

$A = BCd$ $\quad A = B^T$ $\quad a = B^T Bc$

$a = b + c$ $\quad A = B$ $\quad K = A^T CA$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

×

Dense Matrix

CSR DCSR BCSR Thermal Simulation

COO ELLPACK CSB

Blocked COO CSC

DIA Blocked DIA DCSC

Sparsity Is Currently Addressed One-Problem-At-A-Time

$a = Bc$

$a = Bc + a$

$a = Bc + b$ $A = B + C$ $a = \alpha Bc + \beta a$

$a = B^T c$ $A = \alpha B$ $a = B(c + d)$

$a = B^T c + d$ $A = B + C + D$ $A = BC$

$A = B \odot C$ $a = b \odot c$ $A = 0$ $A = B \odot (CD)$

$A = BCd$ $A = B^T$ $a = B^T Bc$

$a = b + c$ $A = B$ $K = A^T CA$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$

$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj}$ $A_{ij} = \sum_{k} B_{ijk} c_k$

$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $A_{ik} = \sum_{j} B_{ijk} c_j$

$A_{jk} = \sum_{i} B_{ijk} c_i$ $A_{ijl} = \sum_{k} B_{ikl} C_{kj}$

$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$

$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$

$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$

×

Dense Matrix

CSR DCSR BCSR Web matrix [BG 2008]

COO ELLPACK CSB

Blocked COO CSC

DIA Blocked DIA DCSC

Sparsity Is Currently Addressed One-Problem-At-A-Time

$a = Bc$

$a = Bc + a$

$a = Bc + b$ $\quad A = B + C$ $\quad a = \alpha Bc + \beta a$

$a = B^T c$ $\quad A = \alpha B$ $\quad a = B(c + d)$

$a = B^T c + d$ $\quad A = B + C + D$ $\quad A = BC$

$A = B \odot C$ $\quad a = b \odot c$ $\quad A = 0$ $\quad A = B \odot (CD)$

$A = BCd$ $\quad A = B^T$ $\quad a = B^T Bc$

$a = b + c$ $\quad A = B$ $\quad K = A^T C A$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

×

Dense Matrix

CSR DCSR BCSR

COO ELLPACK CSB

Blocked COO CSC

DIA Blocked DIA DCSC

Finite Elements Method,
Block-Sparse NN Weights [GRK 2017]

Sparsity Is Currently Addressed One-Problem-At-A-Time

$a = Bc$

$a = Bc + a$

$a = Bc + b$ $\quad A = B + C$ $\quad a = \alpha Bc + \beta a$

$a = B^T c$ $\quad A = \alpha B$ $\quad a = B(c + d)$

$a = B^T c + d$ $\quad A = B + C + D$ $\quad A = BC$

$A = B \odot C$ $\quad a = b \odot c$ $\quad A = 0$ $\quad A = B \odot (CD)$

$A = BCd$ $\quad A = B^T$ $\quad a = B^T Bc$

$a = b + c$ $\quad A = B$ $\quad K = A^T C A$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

×

Dense Matrix

CSR DCSR BCSR

COO ELLPACK CSB Data Analytics

Blocked COO CSC

DIA Blocked DIA DCSC

Sparsity Is Currently Addressed One-Problem-At-A-Time

$$a = Bc$$

$$a = Bc + a$$

$$a = Bc + b \qquad A = B + C \qquad a = \alpha Bc + \beta a$$

$$a = B^T c \qquad A = \alpha B \qquad a = B(c + d)$$

$$a = B^T c + d \qquad A = B + C + D \qquad A = BC$$

$$A = B \odot C \qquad a = b \odot c \quad A = 0 \qquad A = B \odot (CD)$$

$$A = BCd \qquad A = B^T \qquad a = B^T Bc$$

$$a = b + c \qquad A = B \qquad K = A^T CA$$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

×

Dense Matrix

CSR DCSR BCSR

COO **ELLPACK** CSB Mesh Simulations on GPUs [BG 2009]

Blocked COO CSC

DIA Blocked DIA DCSC

Sparsity Is Currently Addressed One-Problem-At-A-Time

$a = Bc$

$a = Bc + a$

$a = Bc + b$ $A = B + C$ $a = \alpha Bc + \beta a$

$a = B^T c$ $A = \alpha B$ $a = B(c + d)$

$a = B^T c + d$ $A = B + C + D$ $A = BC$

$A = B \odot C$ $a = b \odot c$ $A = 0$ $A = B \odot (CD)$

$A = BCd$ $A = B^T$ $a = B^T Bc$

$a = b + c$ $A = B$ $K = A^T CA$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$

$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj}$ $A_{ij} = \sum_{k} B_{ijk} c_k$

$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $A_{ik} = \sum_{j} B_{ijk} c_j$

$A_{jk} = \sum_{i} B_{ijk} c_i$ $A_{ijl} = \sum_{k} B_{ikl} C_{kj}$

$$\tau = \sum_i z_i (\sum_j z_j \theta_{ij})(\sum_k z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

×

Dense Matrix

CSR DCSR BCSR

COO ELLPACK CSB

Blocked COO CSC

DIA Blocked DIA DCSC Convolutions, Image Processing

Sparsity Is Currently Addressed One-Problem-At-A-Time

$a = Bc$

$a = Bc + a$

$a = Bc + b$ $A = B + C$ $a = \alpha Bc + \beta a$

$a = B^T c$ $A = \alpha B$ $a = B(c + d)$

$a = B^T c + d$ $A = B + C + D$ $A = BC$

$A = B \odot C$ $a = b \odot c$ $A = 0$ $A = B \odot (CD)$

$A = BCd$ $A = B^T$ $a = B^T Bc$

$a = b + c$ $A = B$ $K = A^T CA$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$

$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj}$ $A_{ij} = \sum_{k} B_{ijk} c_k$

$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $A_{ik} = \sum_{j} B_{ijk} c_j$

$A_{jk} = \sum_{i} B_{ijk} c_i$ $A_{ijl} = \sum_{k} B_{ikl} C_{kj}$

$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$

$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$

$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$

×

Dense Matrix

CSR DCSR BCSR

COO ELLPACK CSB

Blocked COO CSC

DIA Blocked DIA DCSC Eulerian Simulations

Sparsity Is Currently Addressed One-Problem-At-A-Time

$a = Bc$

$a = Bc + a$

$a = Bc + b$ $\quad A = B + C$ $\quad a = \alpha Bc + \beta a$

$a = B^T c$ $\quad A = \alpha B$ $\quad a = B(c + d)$

$a = B^T c + d$ $\quad A = B + C + D$ $\quad A = BC$

$A = B \odot C$ $\quad a = b \odot c$ $\quad A = 0$ $\quad A = B \odot (CD)$

$A = BCd$ $\quad A = B^T$ $\quad a = B^T Bc$

$a = b + c$ $\quad A = B$ $\quad K = A^T CA$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

×

Dense Matrix

CSR DCSR BCSR

COO ELLPACK CSB

Blocked COO CSC

DIA Blocked DIA DCSC

Sparse vector Hash Maps

Sparsity Is Currently Addressed One-Problem-At-A-Time

$a = Bc$

$a = Bc + a$

$a = Bc + b$ $\quad A = B + C$ $\quad a = \alpha Bc + \beta a$

$a = B^T c$ $\quad A = \alpha B$ $\quad a = B(c + d)$

$a = B^T c + d$ $\quad A = B + C + D$ $\quad A = BC$

$A = B \odot C$ $\quad a = b \odot c$ $\quad A = 0$ $\quad A = B \odot (CD)$

$A = BCd$ $\quad A = B^T$ $\quad a = B^T Bc$

$a = b + c$ $\quad A = B$ $\quad K = A^T CA$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

×

Dense Matrix

CSR DCSR BCSR

COO ELLPACK CSB

Blocked COO CSC

DIA Blocked DIA DCSC

Sparse vector Hash Maps

Coordinates

Dense Tensors

CSF

Blocked Tensors

Sparsity Is Currently Addressed One-Problem-At-A-Time

$a = Bc$

$a = Bc + a$

$a = Bc + b$ $A = B + C$ $a = \alpha Bc + \beta a$

$a = B^T c$ $A = \alpha B$ $a = B(c + d)$

$a = B^T c + d$ $A = B + C + D$ $A = BC$

$A = B \odot C$ $a = b \odot c$ $A = 0$ $A = B \odot (CD)$

$A = BCd$ $A = B^T$ $a = B^T Bc$

$a = b + c$ $A = B$ $K = A^T CA$

$A_{ij} = \sum_{kl} B_{ikl}C_{lj}D_{kj}$ $A_{kj} = \sum_{il} B_{ikl}C_{lj}D_{ij}$

$A_{lj} = \sum_{ik} B_{ikl}C_{ij}D_{kj}$ $A_{ij} = \sum_{k} B_{ijk}c_k$

$A_{ijk} = \sum_{l} B_{ikl}C_{lj}$ $A_{ik} = \sum_{j} B_{ijk}c_j$

$A_{jk} = \sum_{i} B_{ijk}c_i$ $A_{ijl} = \sum_{k} B_{ikl}C_{kj}$

$\tau = \sum_{i} z_i(\sum_{j} z_j\theta_{ij})(\sum_{k} z_k\theta_{ik})$

$C = \sum_{ijkl} M_{ij}P_{jk}\overline{M_{lk}}\,\overline{P_{il}}$

$a = \sum_{ijklmnop} M_{ij}P_{jk}M_{kl}P_{lm}\overline{M_{nm}}P_{no}\overline{M_{po}}\,\overline{P_{ip}}$

×

Dense Matrix

CSR DCSR BCSR

COO ELLPACK CSB

Blocked COO CSC

DIA Blocked DIA DCSC

Sparse vector Hash Maps

Coordinates

Dense Tensors

CSF

Blocked Tensors

×

CPU

GPUs TPUs

FPGA

Sparse Tensor Hardware

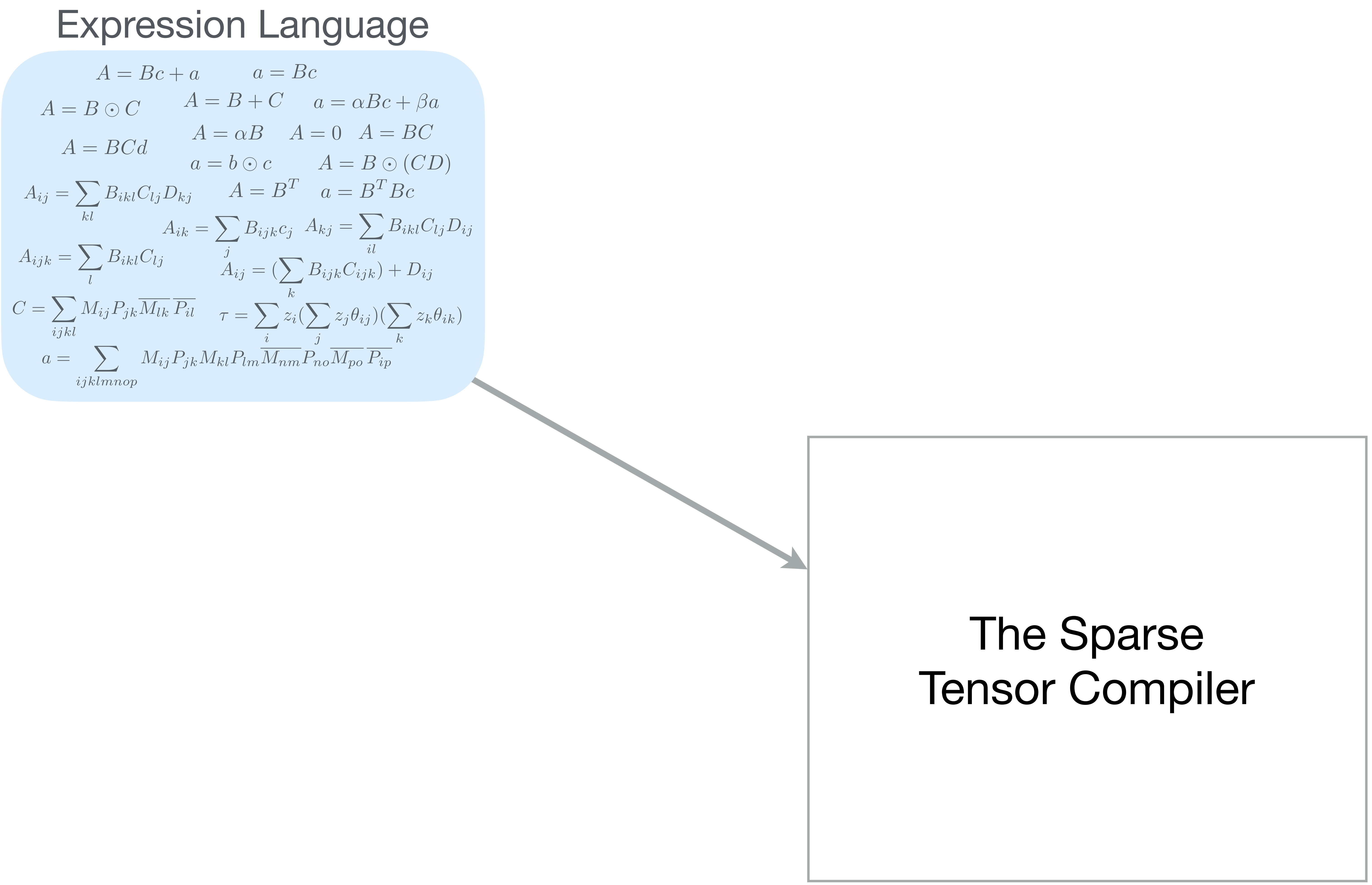
Cloud Computers

Supercomputers

Sparse Tensor Compiler

The Sparse
Tensor Compiler

Sparse Tensor Compiler



Expression Language

$A = Bc + a$ $\quad a = Bc$

$A = B \odot C$ $\quad A = B + C$ $\quad a = \alpha Bc + \beta a$

$A = BCd$ $\quad A = \alpha B$ $\quad A = 0$ $\quad A = BC$

$a = b \odot c$ $\quad A = B \odot (CD)$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $\quad A = B^T$ $\quad a = B^T Bc$

$A_{ik} = \sum_{j} B_{ijk} c_j$ $\quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$

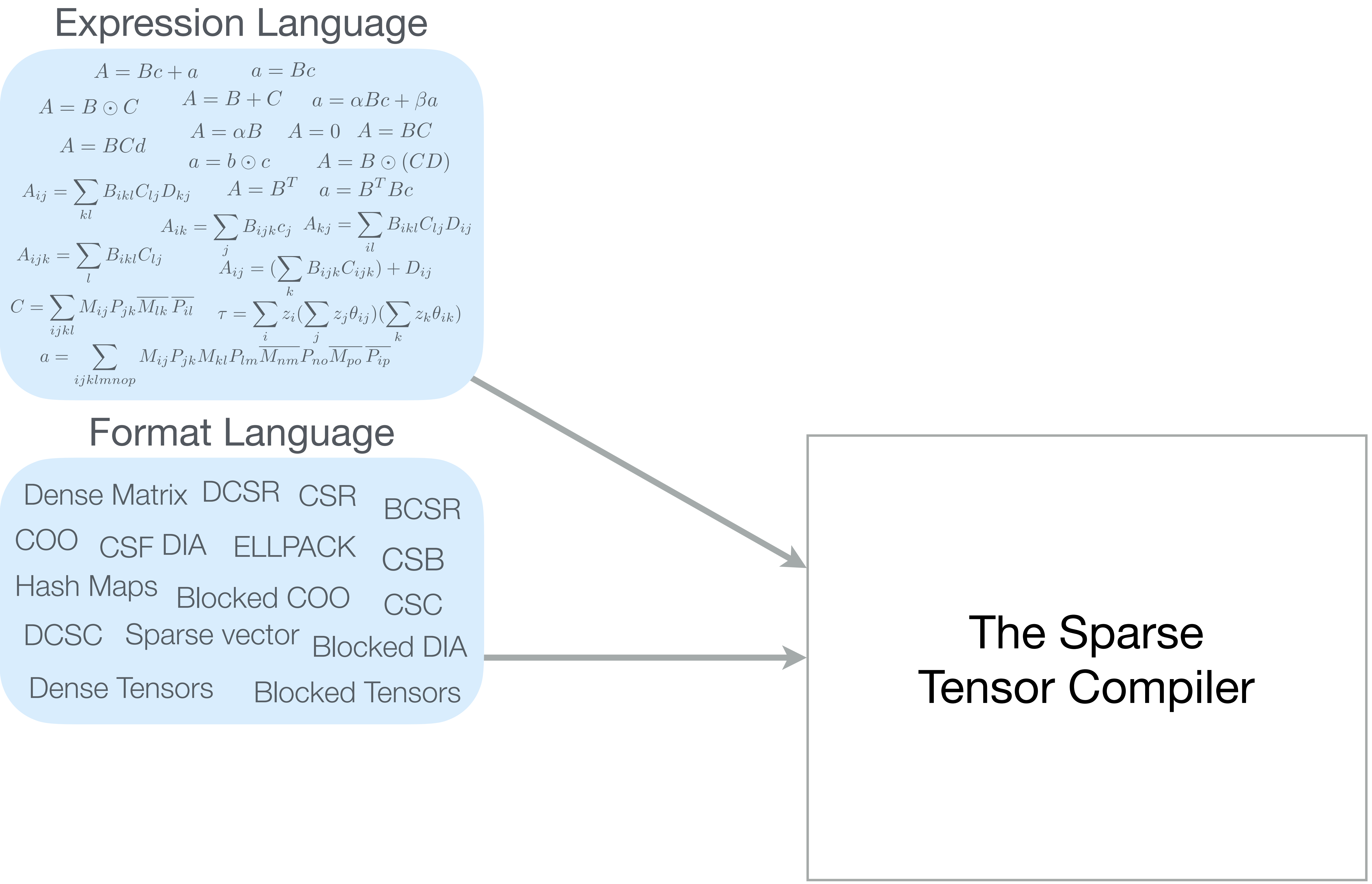
$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $\quad A_{ij} = (\sum_{k} B_{ijk} C_{ijk}) + D_{ij}$

$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$ $\quad \tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij}) (\sum_{k} z_k \theta_{ik})$

$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$

The Sparse
Tensor Compiler

Sparse Tensor Compiler



Expression Language

$A = Bc + a$ $a = Bc$

$A = B \odot C$ $A = B + C$ $a = \alpha Bc + \beta a$

$A = BCd$ $A = \alpha B$ $A = 0$ $A = BC$

$a = b \odot c$ $A = B \odot (CD)$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $A = B^T$ $a = B^T Bc$

$A_{ik} = \sum_{j} B_{ijk} c_j$ $A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$

$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $A_{ij} = (\sum_{k} B_{ijk} C_{ijk}) + D_{ij}$

$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$ $\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$

$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$

Format Language

Dense Matrix DCSR CSR BCSR

COO CSF DIA ELLPACK CSB

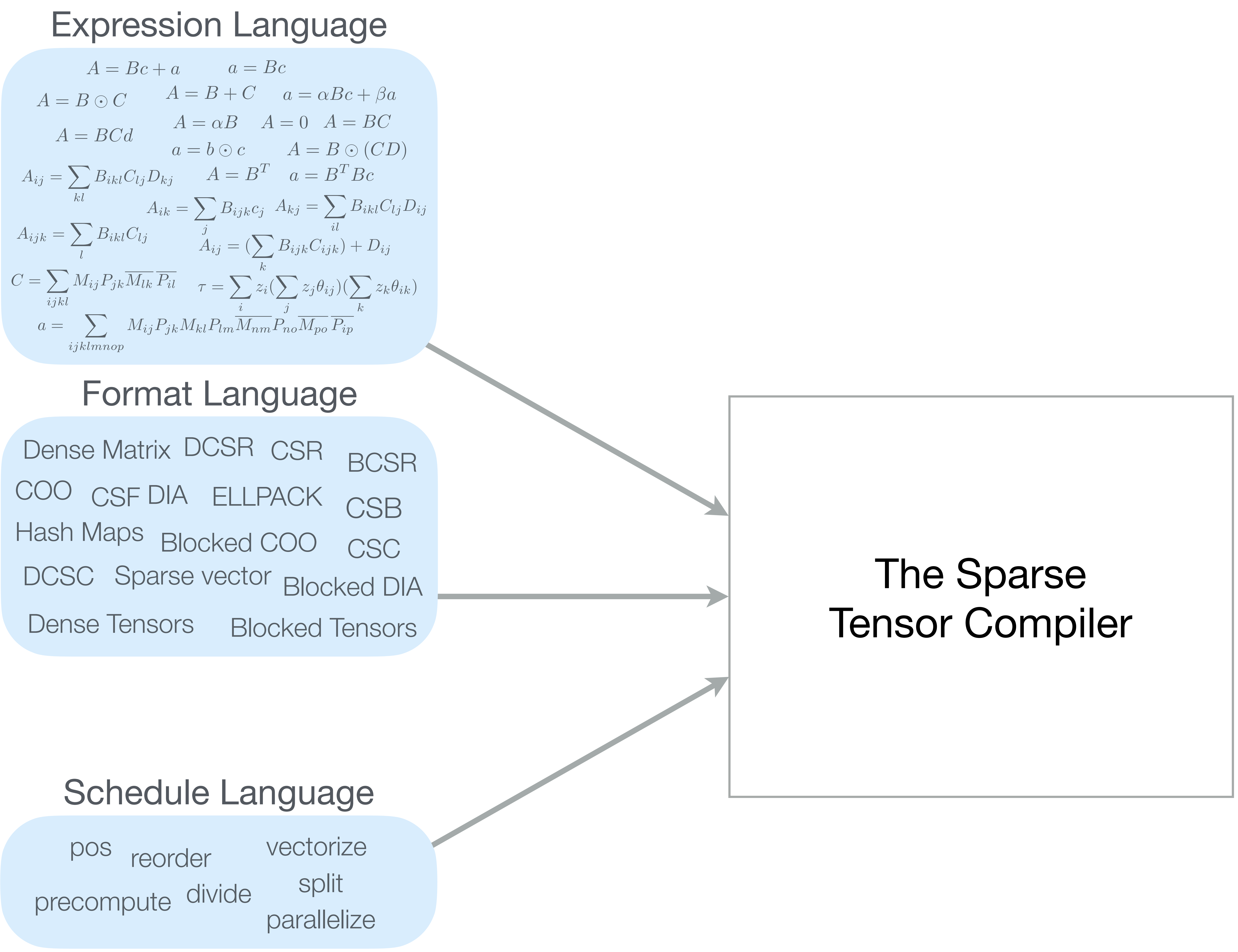
Hash Maps Blocked COO CSC

DCSC Sparse vector Blocked DIA

Dense Tensors Blocked Tensors

The Sparse Tensor Compiler

Sparse Tensor Compiler



Expression Language

$A = Bc + a$ $a = Bc$

$A = B \odot C$ $A = B + C$ $a = \alpha Bc + \beta a$

$A = BCd$ $A = \alpha B$ $A = 0$ $A = BC$

$a = b \odot c$ $A = B \odot (CD)$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $A = B^T$ $a = B^T Bc$

$A_{ik} = \sum_{j} B_{ijk} c_j$ $A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$

$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $A_{ij} = (\sum_{k} B_{ijk} C_{ijk}) + D_{ij}$

$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$ $\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$

$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$

Format Language

Dense Matrix DCSR CSR BCSR

COO CSF DIA ELLPACK CSB

Hash Maps Blocked COO CSC

DCSC Sparse vector Blocked DIA

Dense Tensors Blocked Tensors

Schedule Language

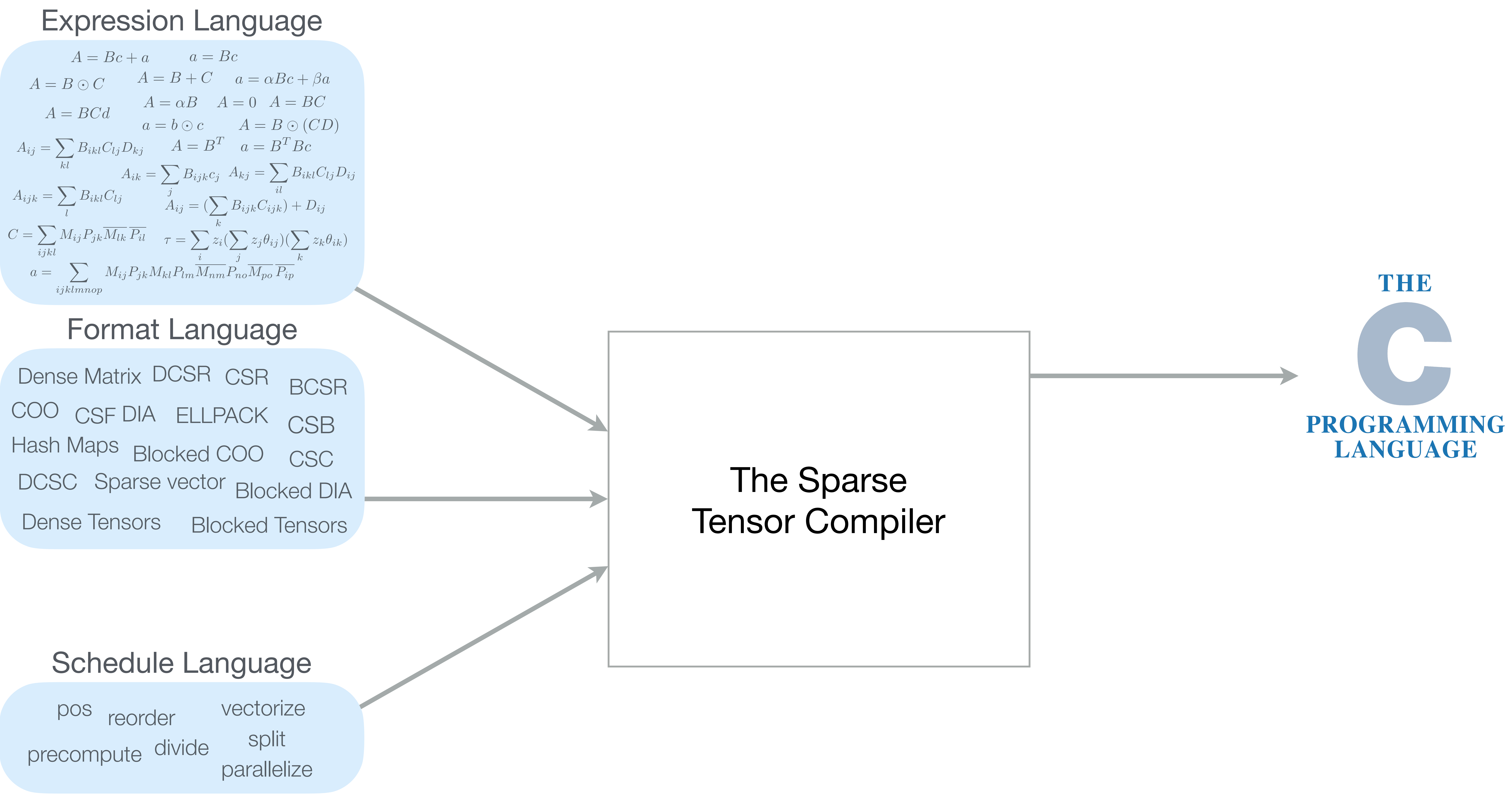
pos reorder vectorize

precompute divide split

parallelize

The Sparse Tensor Compiler

Sparse Tensor Compiler



Expression Language

$A = Bc + a$ $a = Bc$

$A = B \odot C$ $A = B + C$ $a = \alpha Bc + \beta a$

$A = BCd$ $A = \alpha B$ $A = 0$ $A = BC$

$a = b \odot c$ $A = B \odot (CD)$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $A = B^T$ $a = B^T Bc$

$A_{ik} = \sum_{j} B_{ijk} c_j$ $A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$

$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $A_{ij} = (\sum_{k} B_{ijk} C_{ijk}) + D_{ij}$

$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\,\overline{P_{il}}$ $\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$

$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\,\overline{P_{ip}}$

Format Language

Dense Matrix DCSR CSR BCSR
COO CSF DIA ELLPACK CSB
Hash Maps Blocked COO CSC
DCSC Sparse vector Blocked DIA
Dense Tensors Blocked Tensors

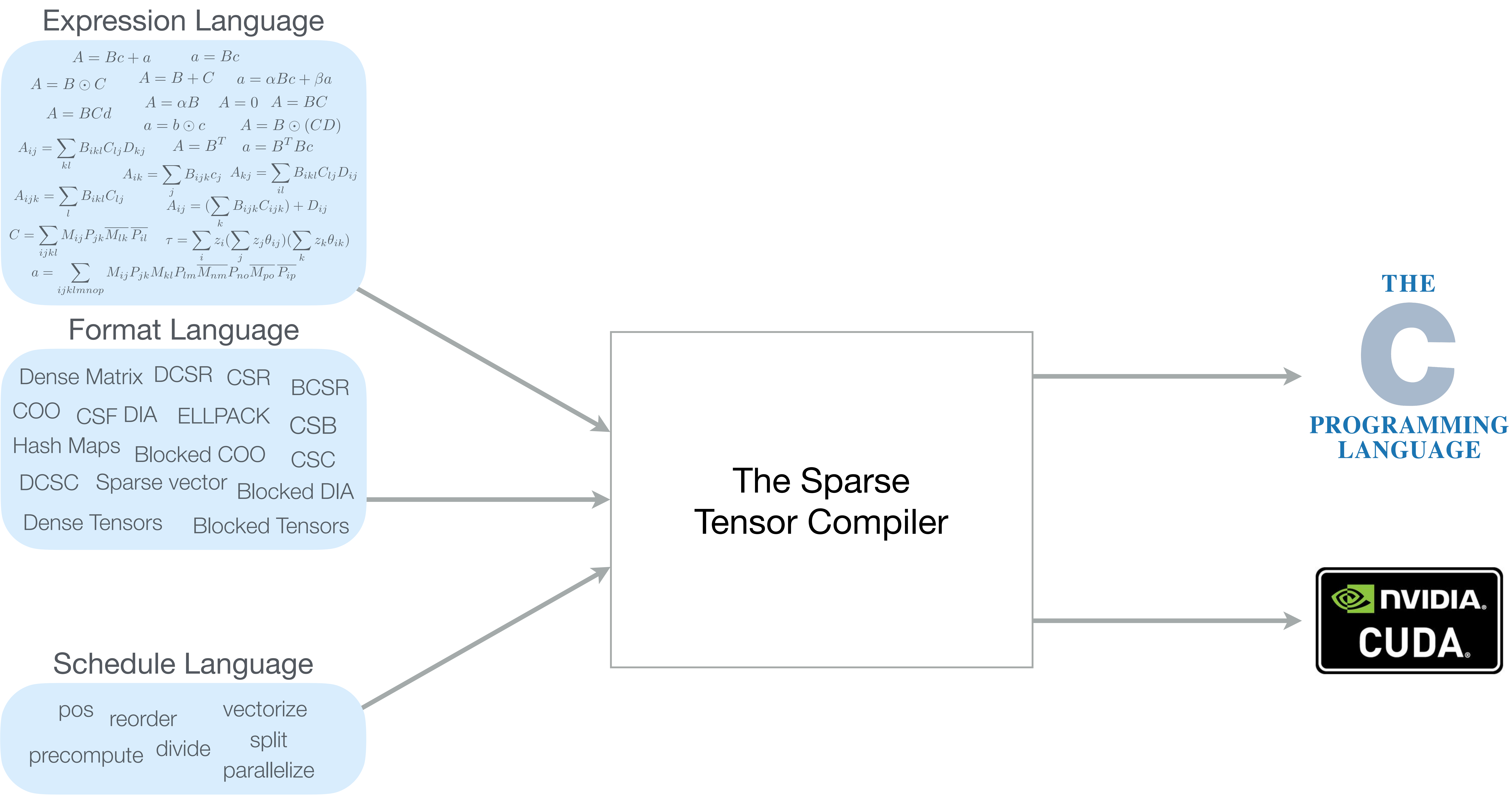
Schedule Language

pos reorder vectorize
precompute divide split
parallelize

The Sparse Tensor Compiler

THE C PROGRAMMING LANGUAGE

Sparse Tensor Compiler



Expression Language

$A = Bc + a$ $\quad a = Bc$

$A = B \odot C$ $\quad A = B + C$ $\quad a = \alpha Bc + \beta a$

$A = BCd$ $\quad A = \alpha B$ $\quad A = 0$ $\quad A = BC$

$a = b \odot c$ $\quad A = B \odot (CD)$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $\quad A = B^T$ $\quad a = B^T Bc$

$A_{ik} = \sum_{j} B_{ijk} c_j$ $\quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$

$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $\quad A_{ij} = (\sum_{k} B_{ijk} C_{ijk}) + D_{ij}$

$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\,\overline{P_{il}}$ $\quad \tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$

$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\,\overline{P_{ip}}$

Format Language

Dense Matrix DCSR CSR BCSR

COO CSF DIA ELLPACK CSB

Hash Maps Blocked COO CSC

DCSC Sparse vector Blocked DIA

Dense Tensors Blocked Tensors

Schedule Language

pos reorder vectorize

precompute divide split

parallelize

The Sparse Tensor Compiler

Generated Sparse Code Performance Matches Hand-Optimized Libraries

$a = Bc + a$ $\quad a = Bc$

$a = Bc + b$ $\quad A = B + C$ $\quad a = \alpha Bc + \beta a$

$a = B^T c$ $\quad A = \alpha B$ $\quad a = B(c + d)$

$a = B^T c + d$ $\quad A = B + C + D$ $\quad A = BC$

$A = B \odot C$ $\quad a = b \odot c$ $\quad A = 0$ $\quad A = B \odot (CD)$

$A = BCd$ $\quad A = B^T$ $\quad a = B^T Bc$

$a = b + c$ $\quad A = B$ $\quad K = A^T CA$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $\quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $\quad A_{ij} = \sum_{k} B_{ijk} c_k$

$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $\quad A_{ik} = \sum_{j} B_{ijk} c_j$

$A_{jk} = \sum_{i} B_{ijk} c_i$ $\quad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

Normalized time

Generated Sparse Code Performance Matches Hand-Optimized Libraries

$a = Bc + a$ $\quad a = Bc$

$a = Bc + b$ $\quad A = B + C$ $\quad a = \alpha Bc + \beta a$

$a = B^T c$ $\quad A = \alpha B$ $\quad a = B(c + d)$

$a = B^T c + d$ $\quad A = B + C + D$ $\quad A = BC$

$A = B \odot C$ $\quad a = b \odot c$ $\quad A = 0$ $\quad A = B \odot (CD)$

$A = BCd$ $\quad A = B^T$ $\quad a = B^T Bc$

$a = b + c$ $\quad A = B$ $\quad K = A^T CA$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

$$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

Normalized time

Generated Sparse Code Performance Matches Hand-Optimized Libraries

$a = Bc + a$

$a = Bc + b$ $A = B + C$ $a = \alpha Bc + \beta a$

$a = B^T c$ $A = \alpha B$ $a = B(c + d)$

$a = B^T c + d$ $A = B + C + D$ $A = BC$

$A = 0$

$A = B \odot C$ $a = b \odot c$ $A = B \odot (CD)$

$A = BCd$ $A = B^T$ $a = B^T Bc$

$a = b + c$ $A = B$ $K = A^T CA$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $A_{ij} = \sum_{k} B_{ijk} c_k$

$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $A_{ik} = \sum_{j} B_{ijk} c_j$

$A_{jk} = \sum_{i} B_{ijk} c_i$ $A_{ijl} = \sum_{k} B_{ikl} C_{kj}$

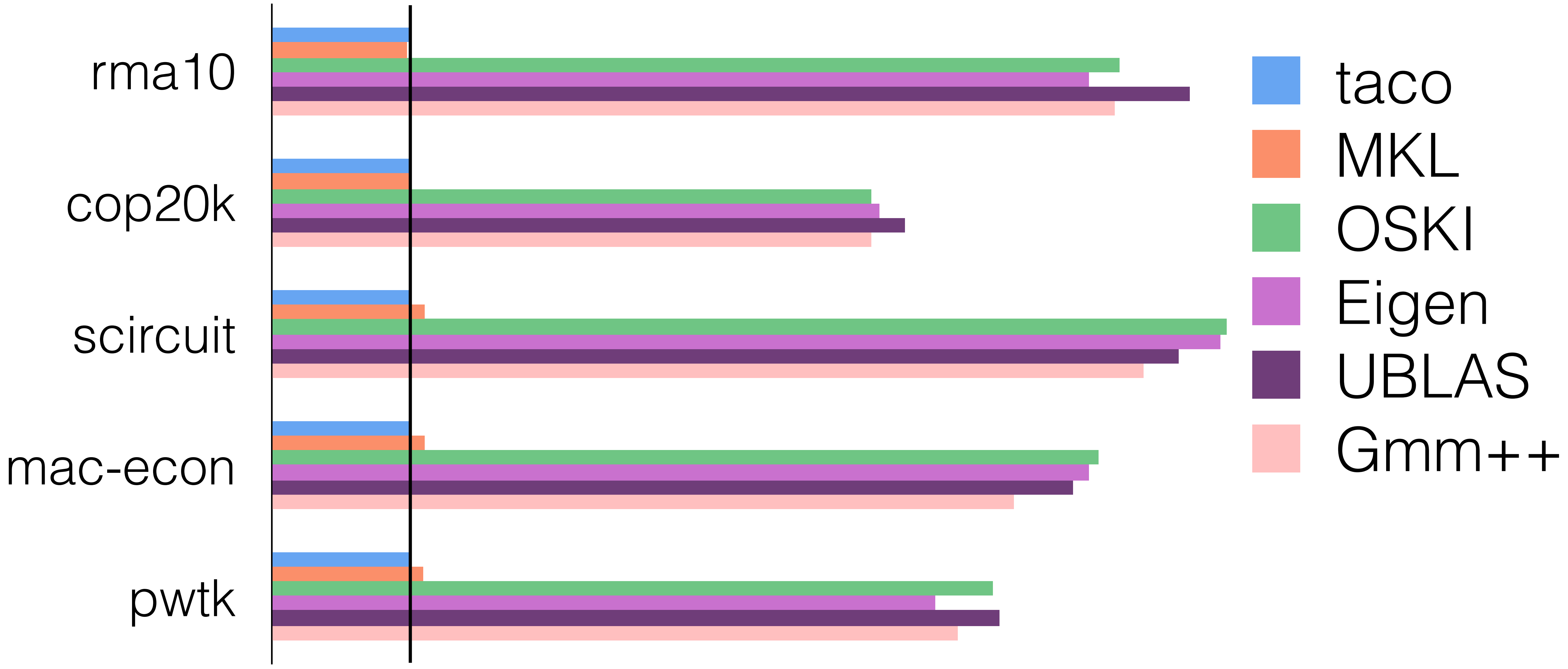
$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$

$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$

$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$

SpMV

$a = Bc$



Normalized time

Generated Sparse Code Performance Matches Hand-Optimized Libraries

$a = Bc + a$

$a = Bc + b$ $A = B + C$ $a = \alpha Bc + \beta a$

$a = B^T c$ $A = \alpha B$ $a = B(c + d)$

$a = B^T c + d$ $A = B + C + D$ $A = BC$

$A = 0$

$A = B \odot C$ $a = b \odot c$

$A = BCd$ $A = B^T$ $a = B^T Bc$

$a = b + c$ $A = B$ $K = A^T C A$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $A_{ij} = \sum_{k} B_{ijk} c_k$

$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $A_{ik} = \sum_{j} B_{ijk} c_j$

$A_{jk} = \sum_{i} B_{ijk} c_i$ $A_{ijl} = \sum_{k} B_{ikl} C_{kj}$

$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$

$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$

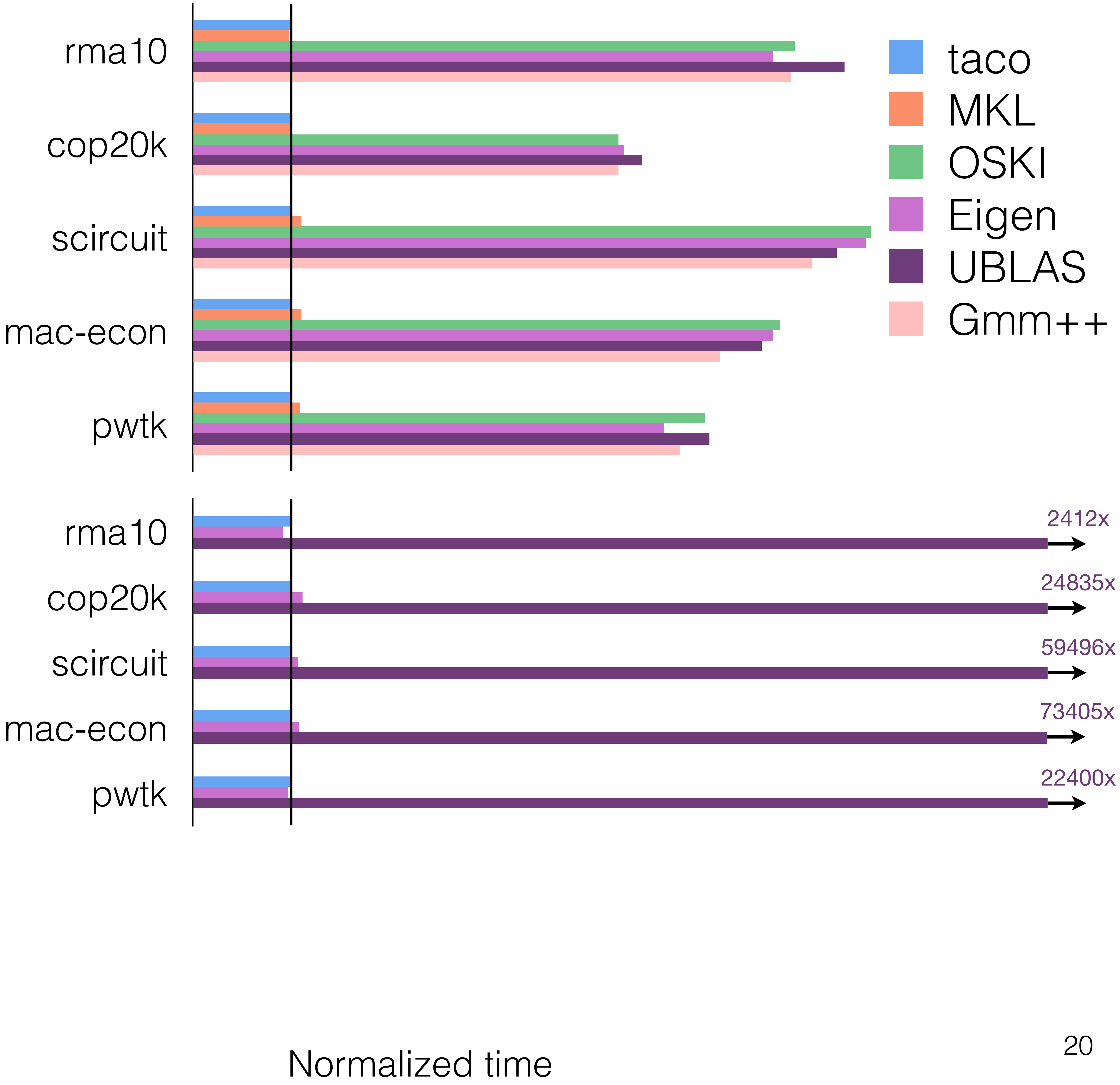
$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$

SpMV

$a = Bc$

SDDMM

$A = B \odot (CD)$



Generated Sparse Code Performance Matches Hand-Optimized Libraries

$a = Bc + a$

$a = Bc + b$ $A = B + C$ $a = \alpha Bc + \beta a$

$a = B^T c$ $A = \alpha B$ $a = B(c + d)$

$a = B^T c + d$ $A = B + C + D$ $A = BC$

$A = 0$

$A = B \odot C$ $a = b \odot c$

$A = BCd$ $A = B^T$ $a = B^T Bc$

$a = b + c$ $A = B$ $K = A^T CA$

$A_{ij} = \sum_{kl} B_{ikl}C_{lj}D_{kj}$ $A_{kj} = \sum_{il} B_{ikl}C_{lj}D_{ij}$

$A_{ij} = \sum_{k} B_{ijk}c_k$

$A_{ijk} = \sum_{l} B_{ikl}C_{lj}$ $A_{ik} = \sum_{j} B_{ijk}c_j$

$A_{jk} = \sum_{i} B_{ijk}c_i$ $A_{ijl} = \sum_{k} B_{ikl}C_{kj}$

$\tau = \sum_{i} z_i(\sum_{j} z_j\theta_{ij})(\sum_{k} z_k\theta_{ik})$

$C = \sum_{ijkl} M_{ij}P_{jk}\overline{M_{lk}}\,\overline{P_{il}}$

$a = \sum_{ijklmnop} M_{ij}P_{jk}M_{kl}P_{lm}\overline{M_{nm}}P_{no}\overline{M_{po}}\,\overline{P_{ip}}$

SpMV

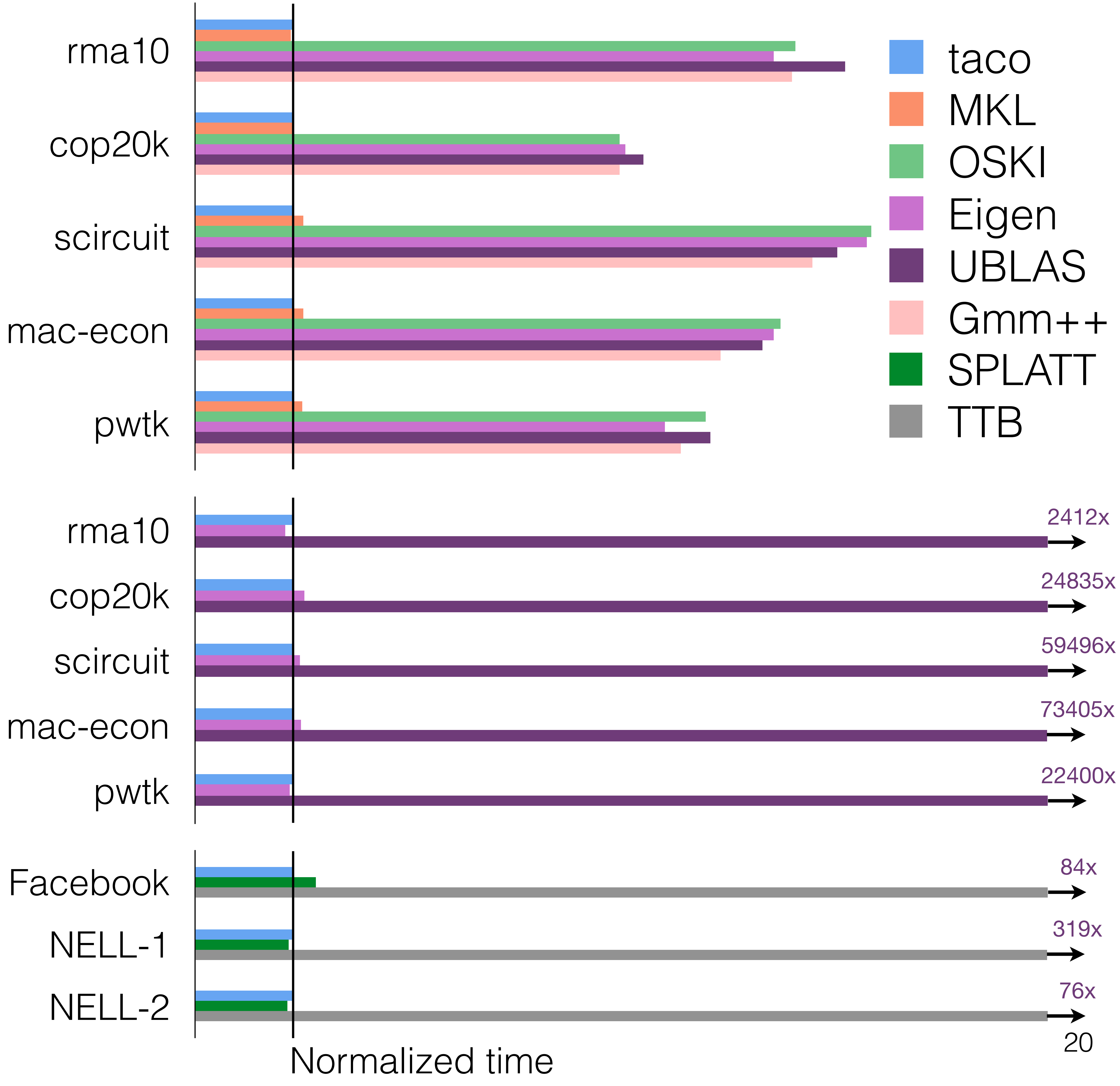
$a = Bc$

SDDMM

$A = B \odot (CD)$

MTTKRP

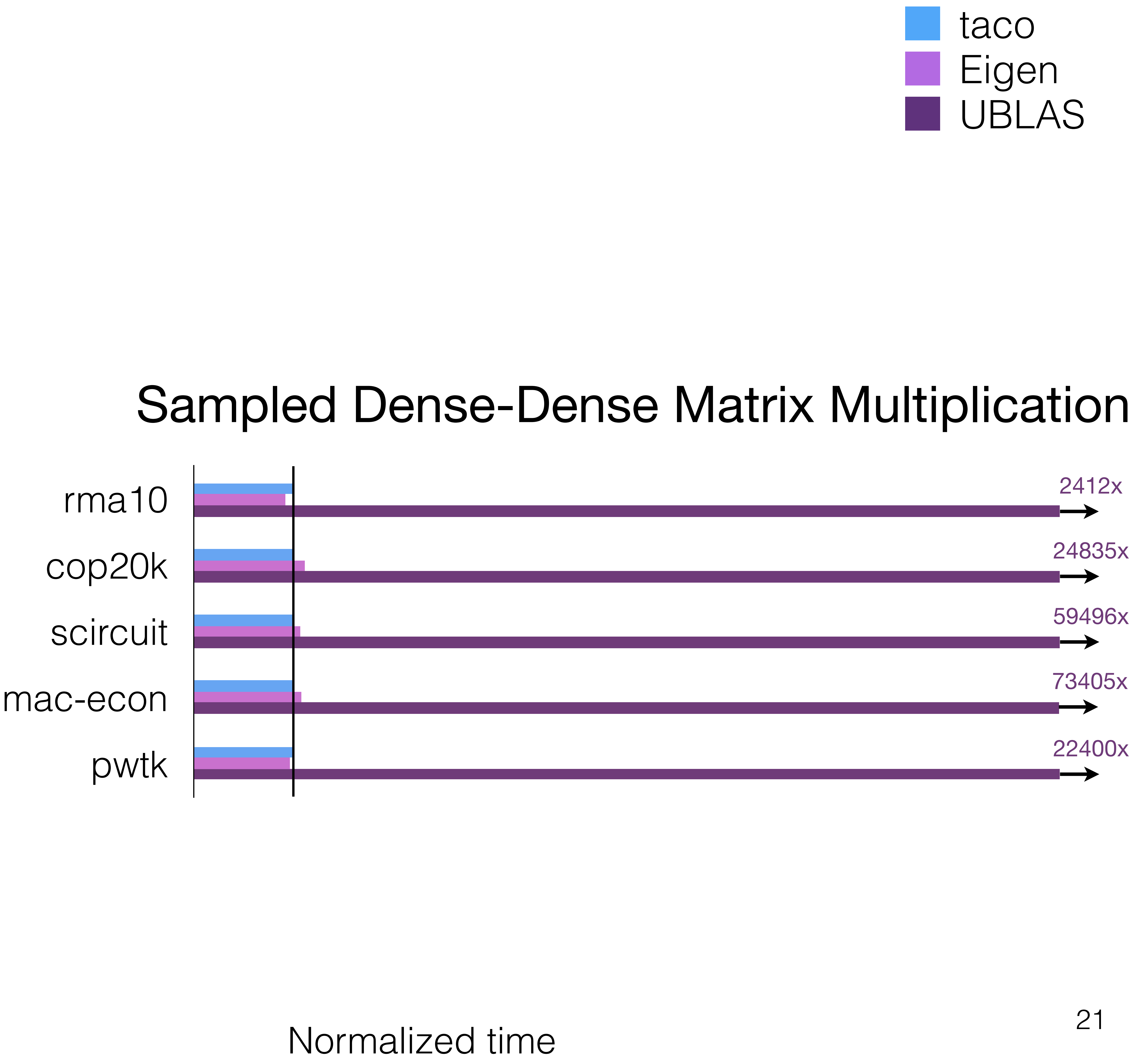
$A_{ij} = \sum_{kl} B_{ikl}C_{lj}D_{kj}$



Generated Sparse Code Performance Matches Hand-Optimized Libraries

SDDMM

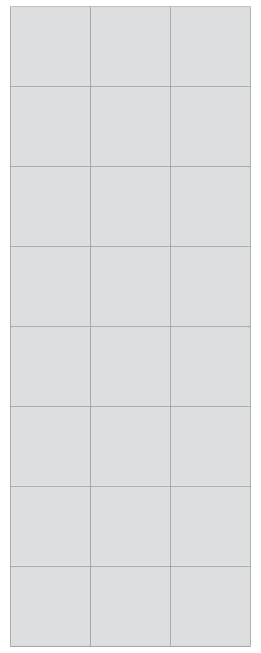
$$A = B \odot (CD)$$



Sampled Dense-Dense Matrix Multiplication

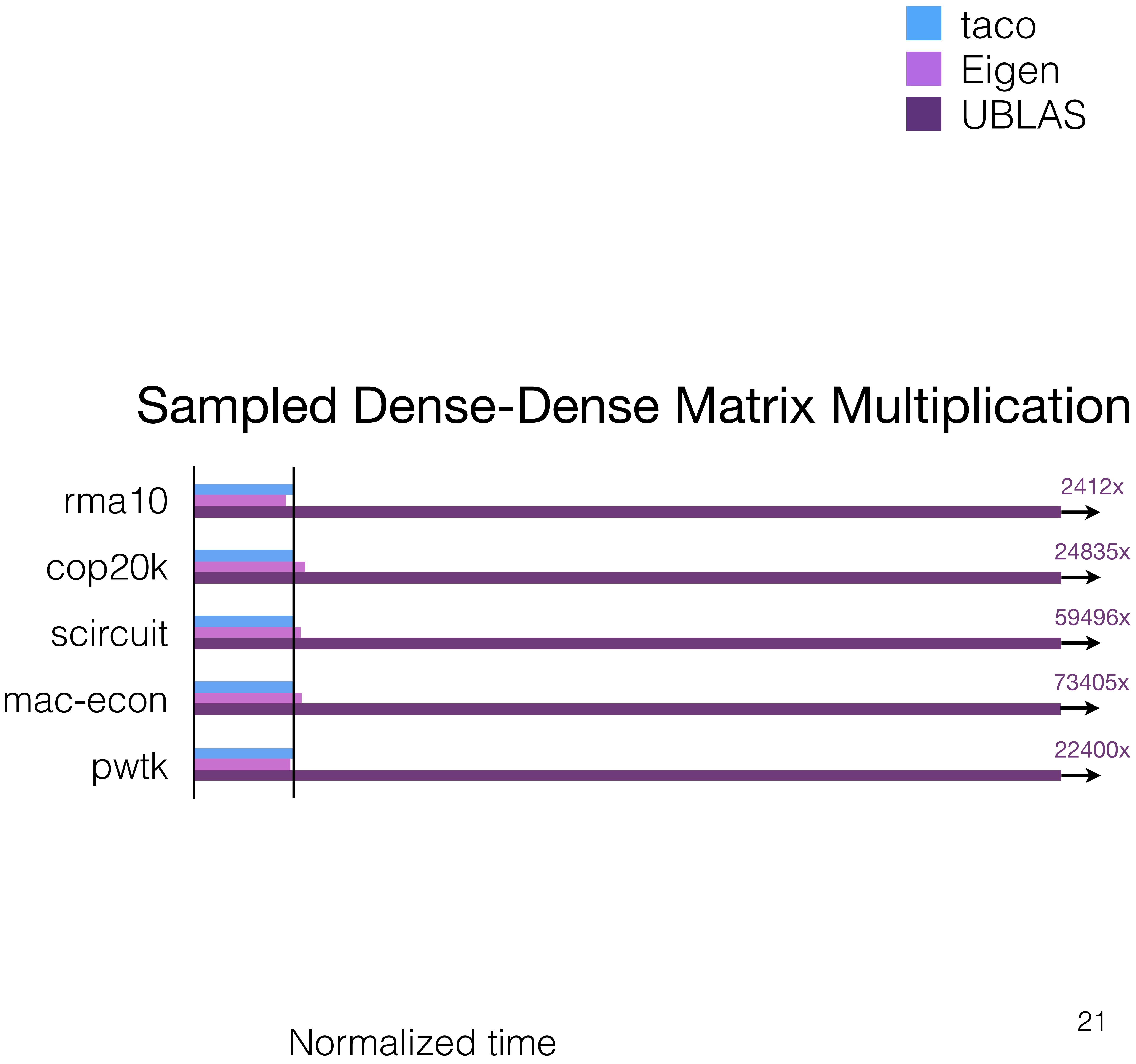
Generated Sparse Code Performance Matches Hand-Optimized Libraries

C



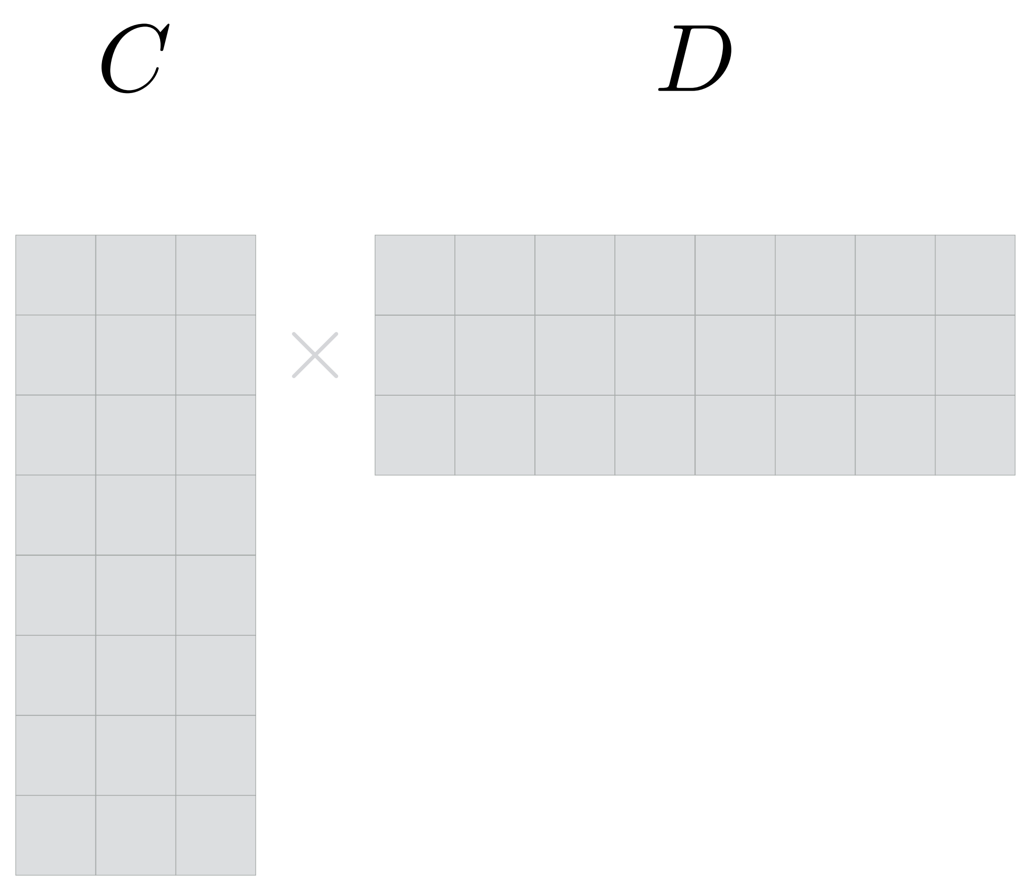
SDDMM

$$A = B \odot (CD)$$



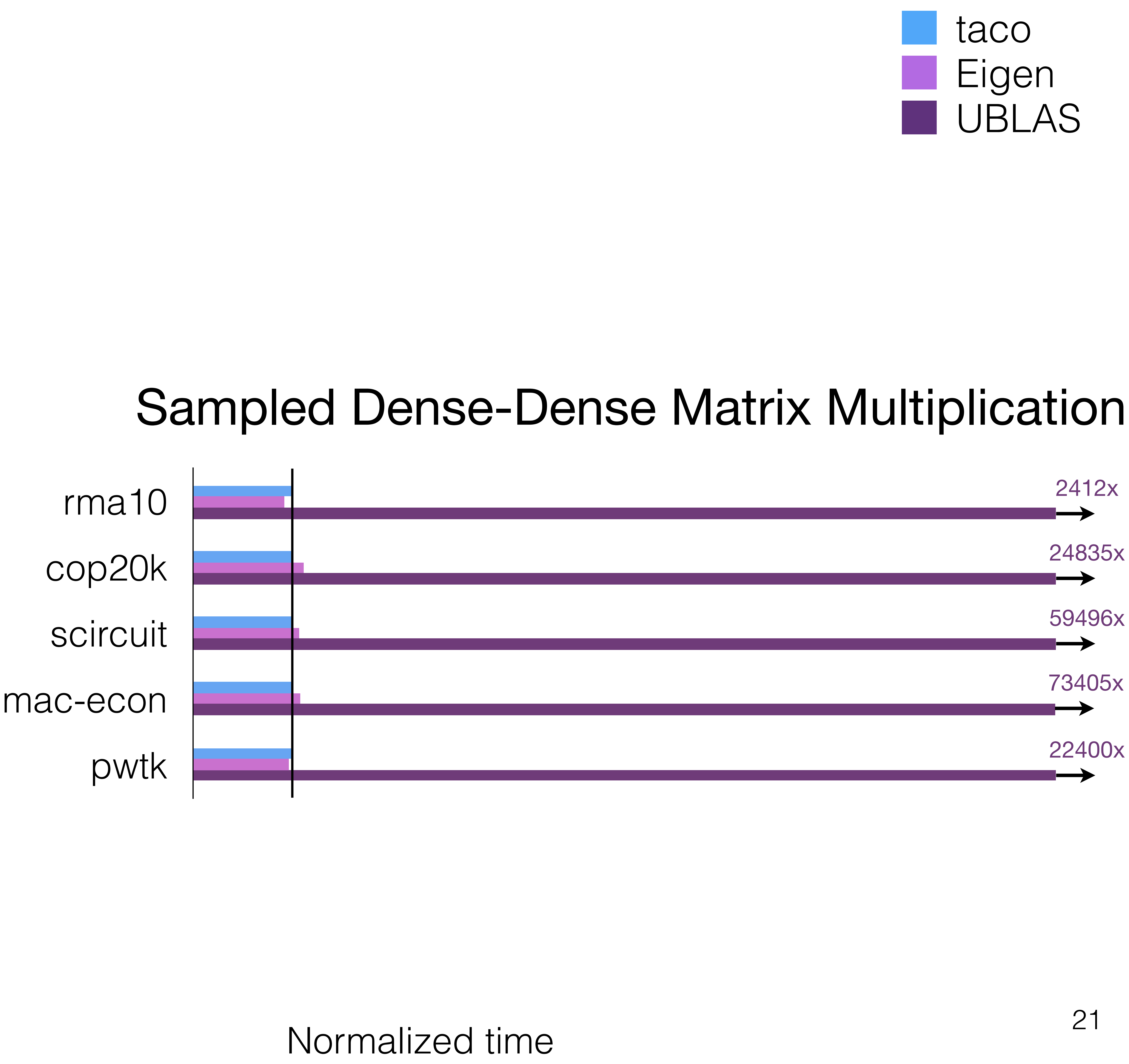
Sampled Dense-Dense Matrix Multiplication

Generated Sparse Code Performance Matches Hand-Optimized Libraries

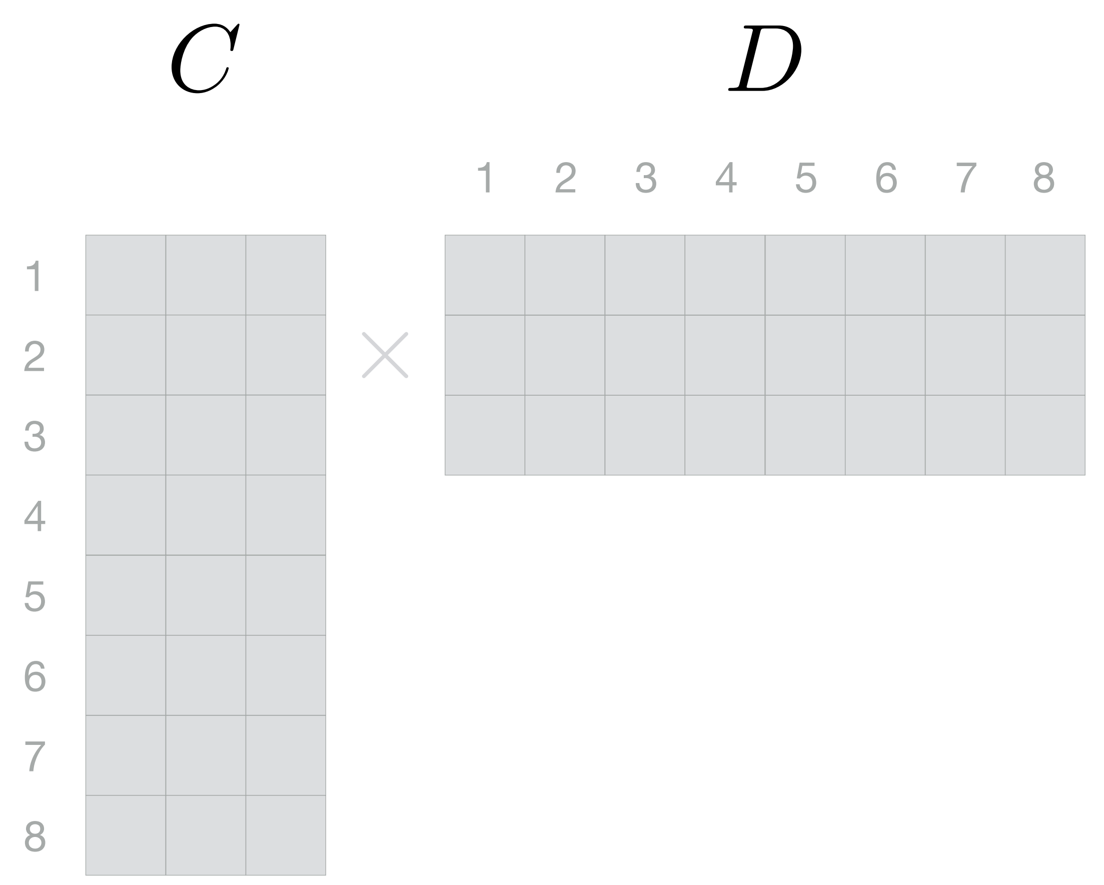


SDDMM

$$A = B \odot (CD)$$


Sampled Dense-Dense Matrix Multiplication

Generated Sparse Code Performance Matches Hand-Optimized Libraries

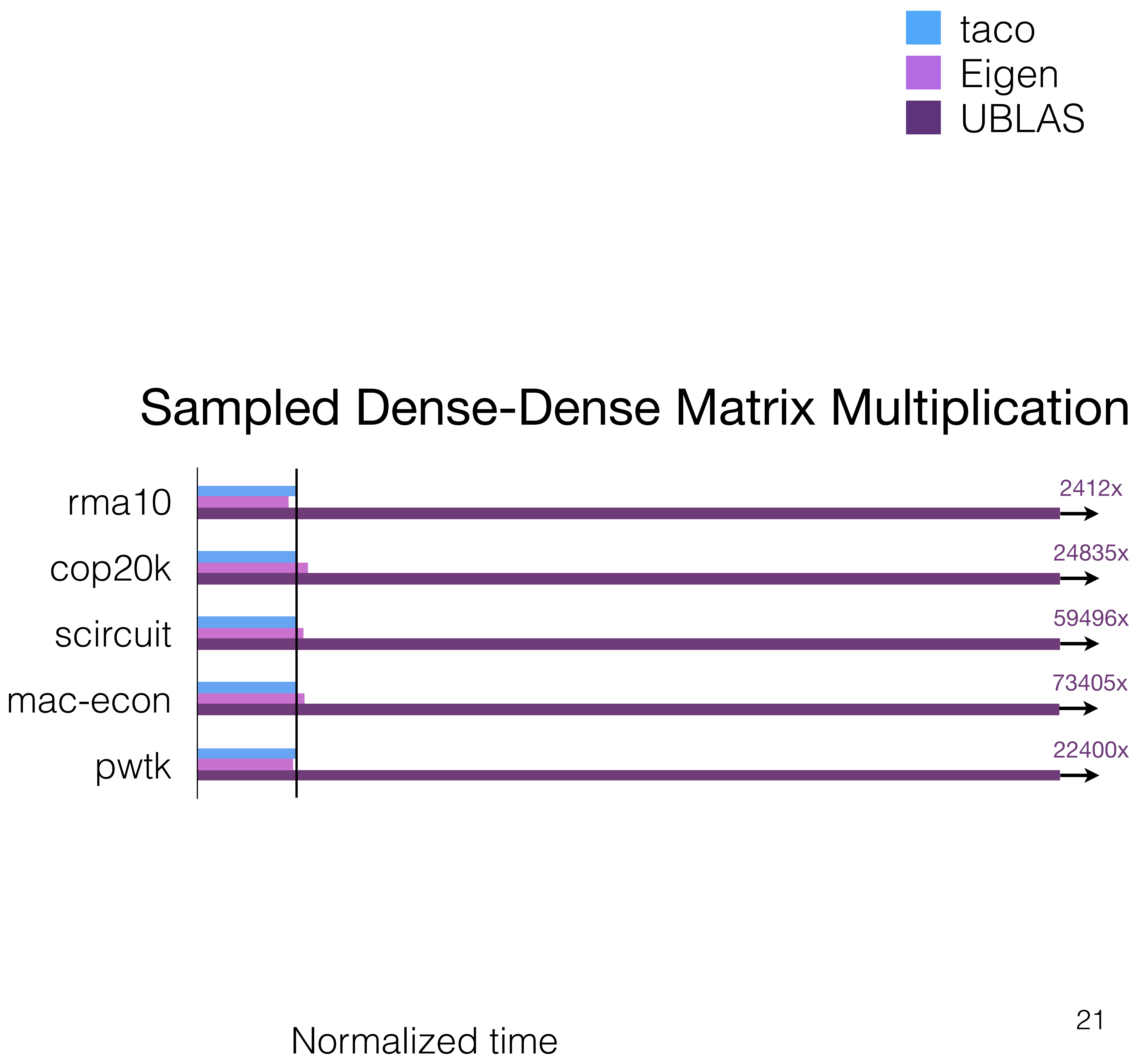


64 inner product

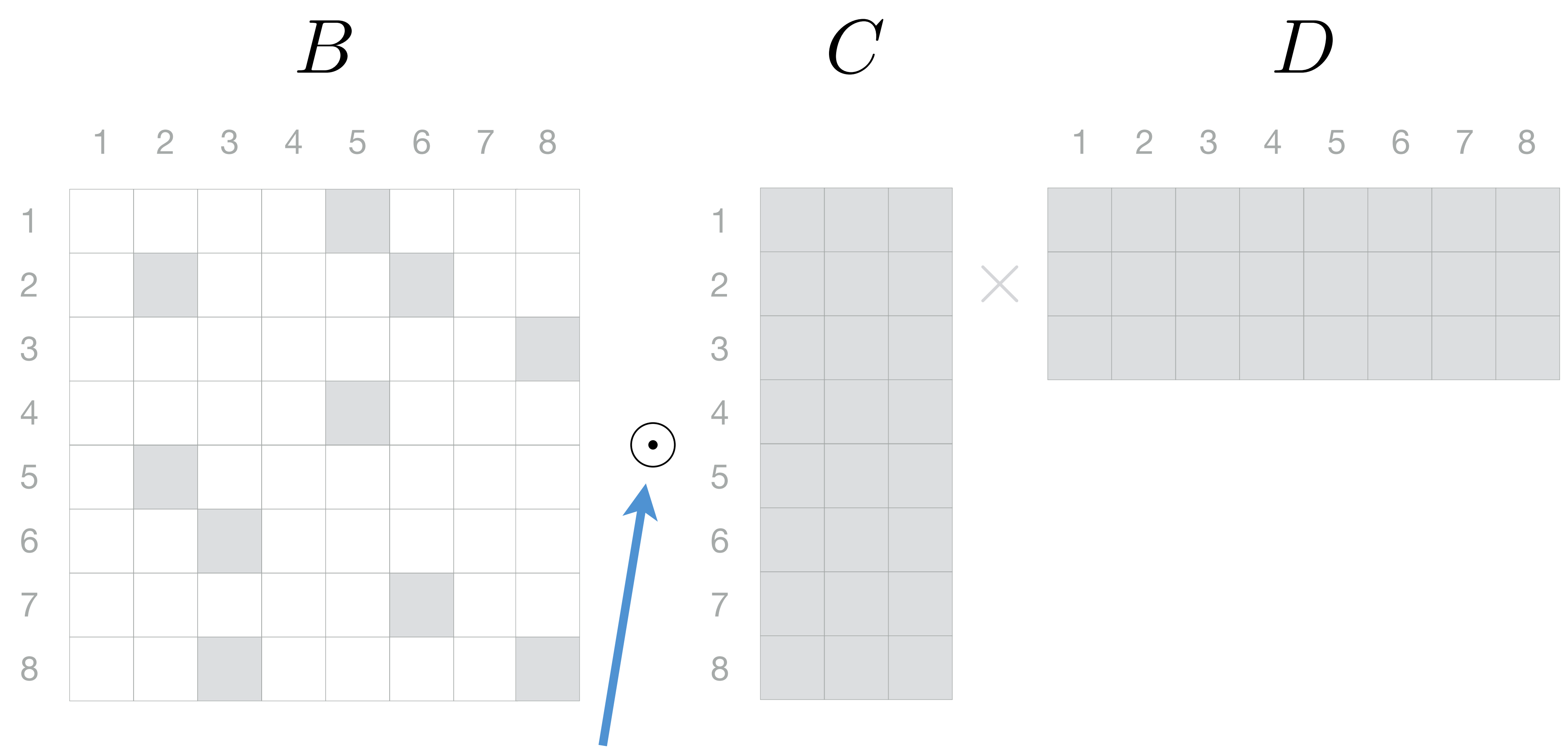
SDDMM

$$A = B \odot (CD)$$

Sampled Dense-Dense Matrix Multiplication



Generated Sparse Code Performance Matches Hand-Optimized Libraries

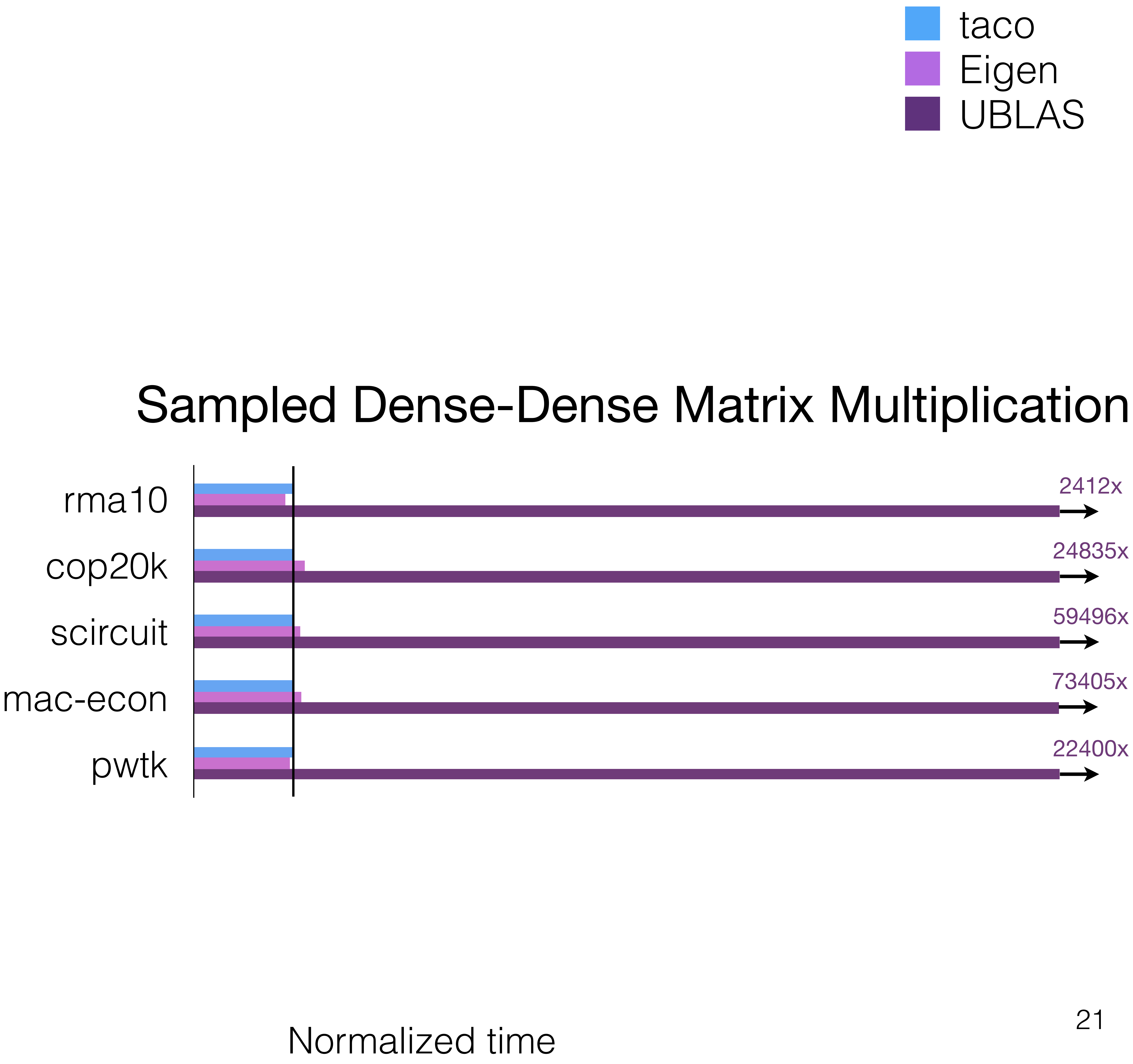


Element-wise multiplication

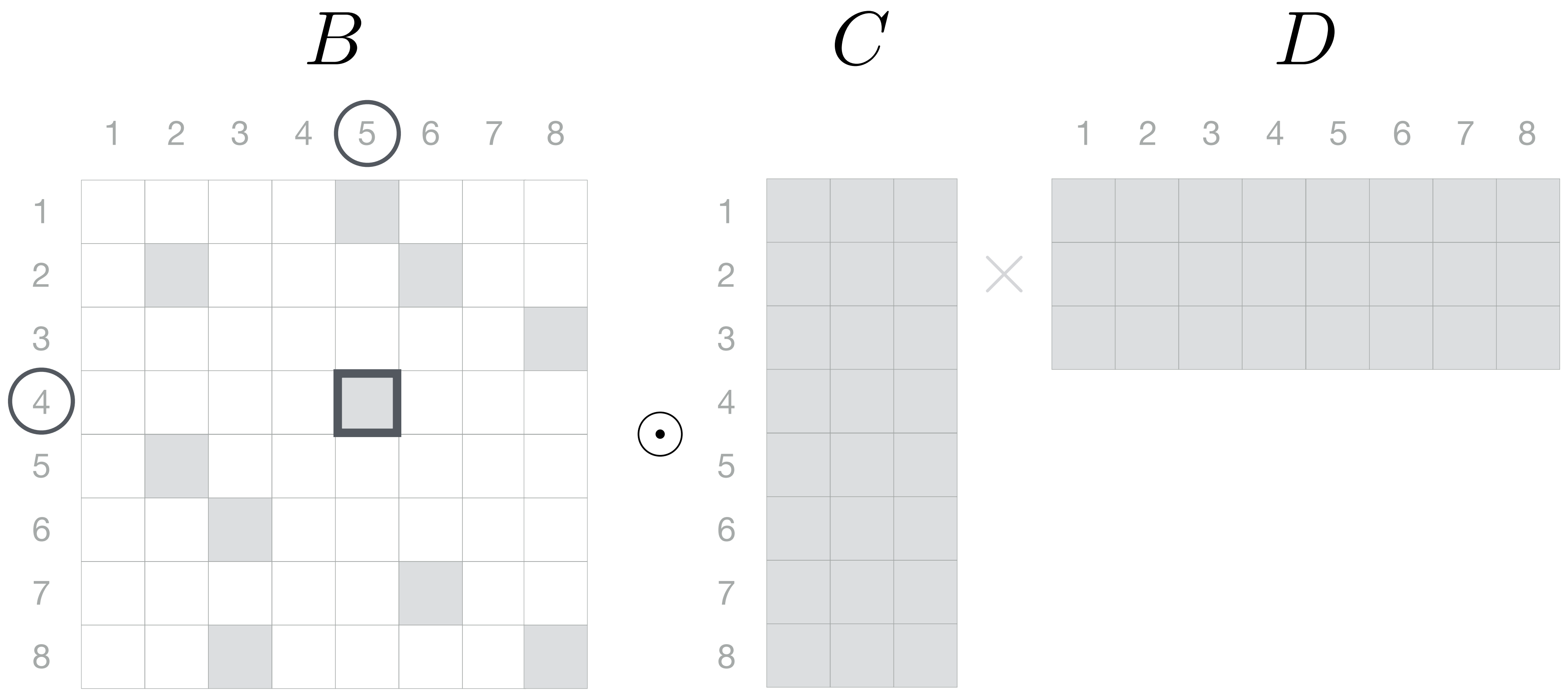
64 inner product

SDDMM

$$A = B \odot (CD)$$


Sampled Dense-Dense Matrix Multiplication

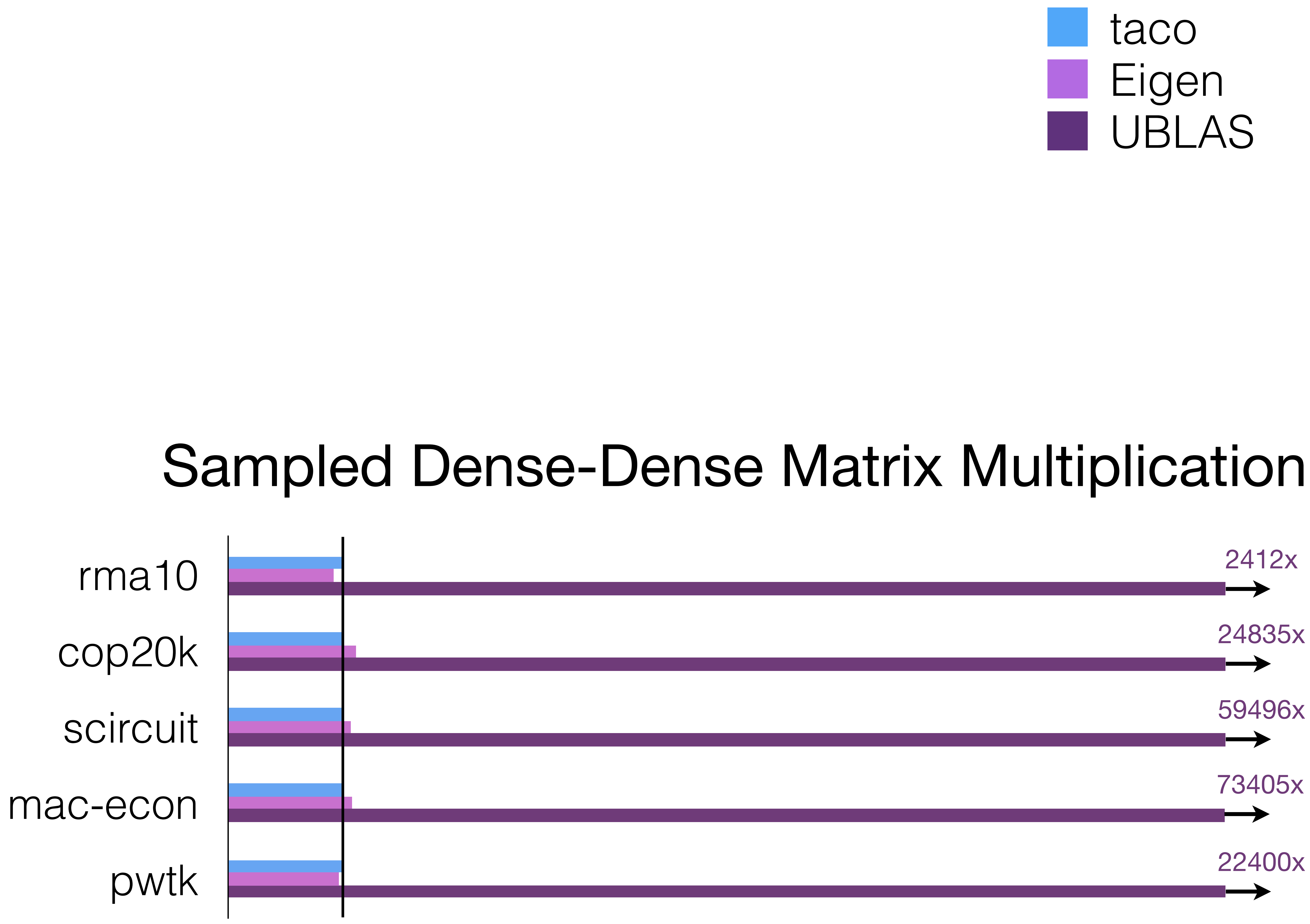
Generated Sparse Code Performance Matches Hand-Optimized Libraries



64 inner product

SDDMM

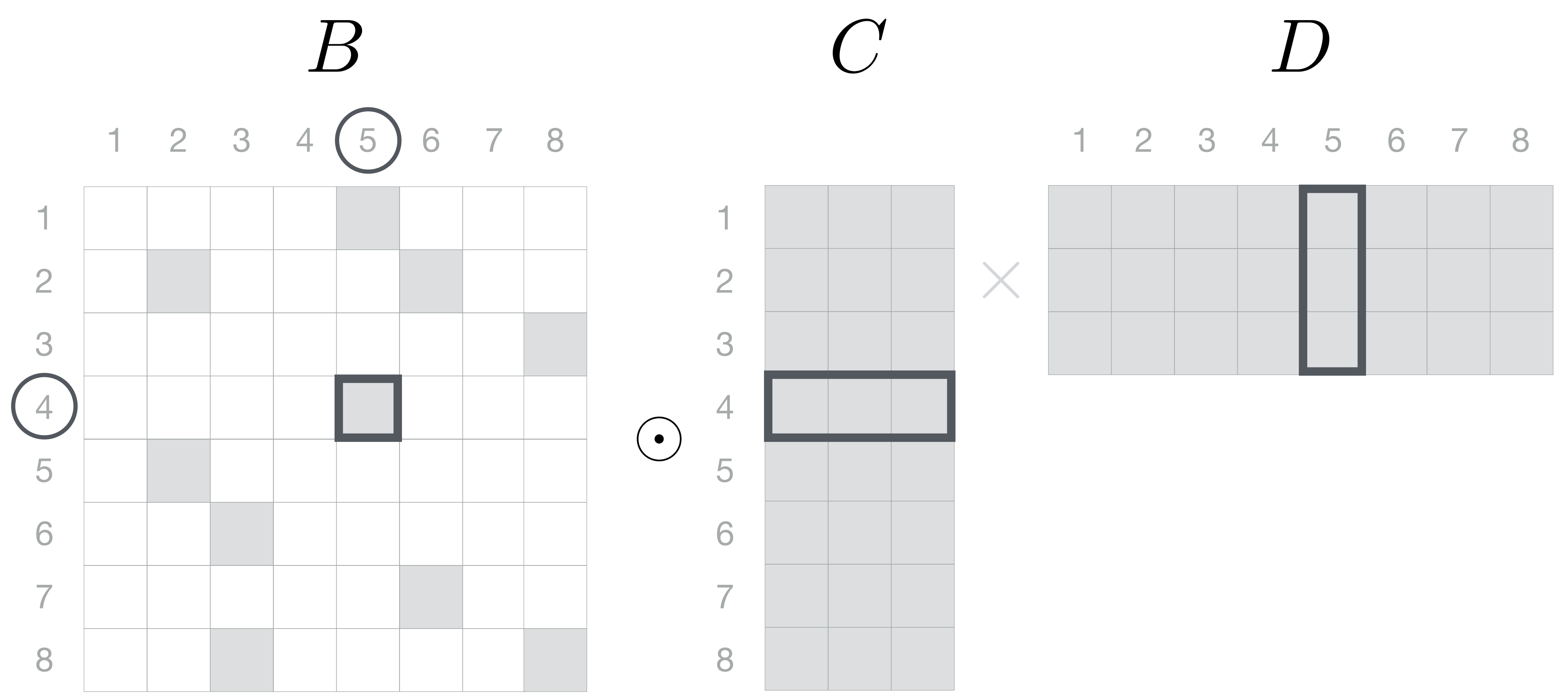
$$A = B \odot (CD)$$



Sampled Dense-Dense Matrix Multiplication

Normalized time

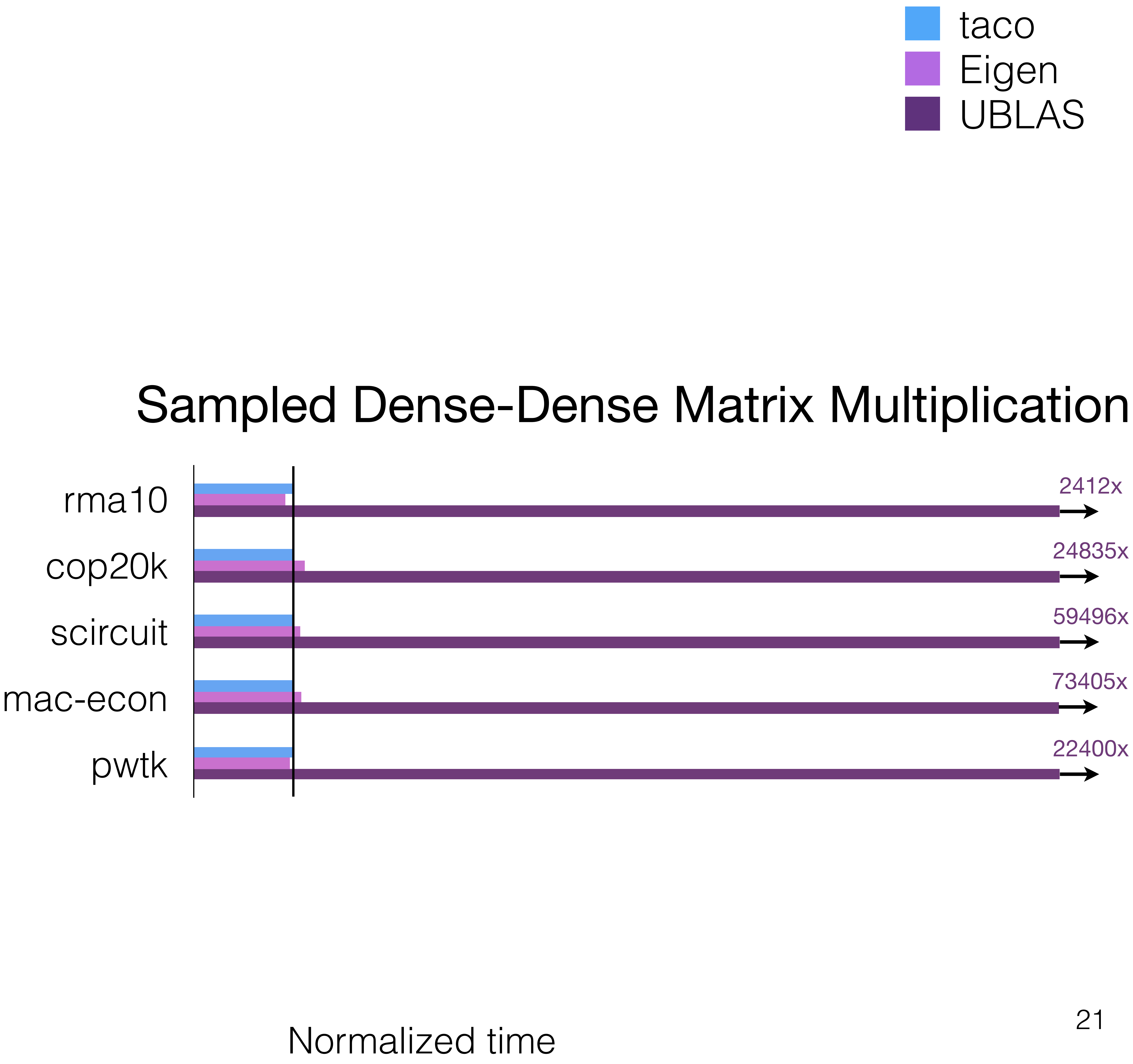
Generated Sparse Code Performance Matches Hand-Optimized Libraries



64 inner product

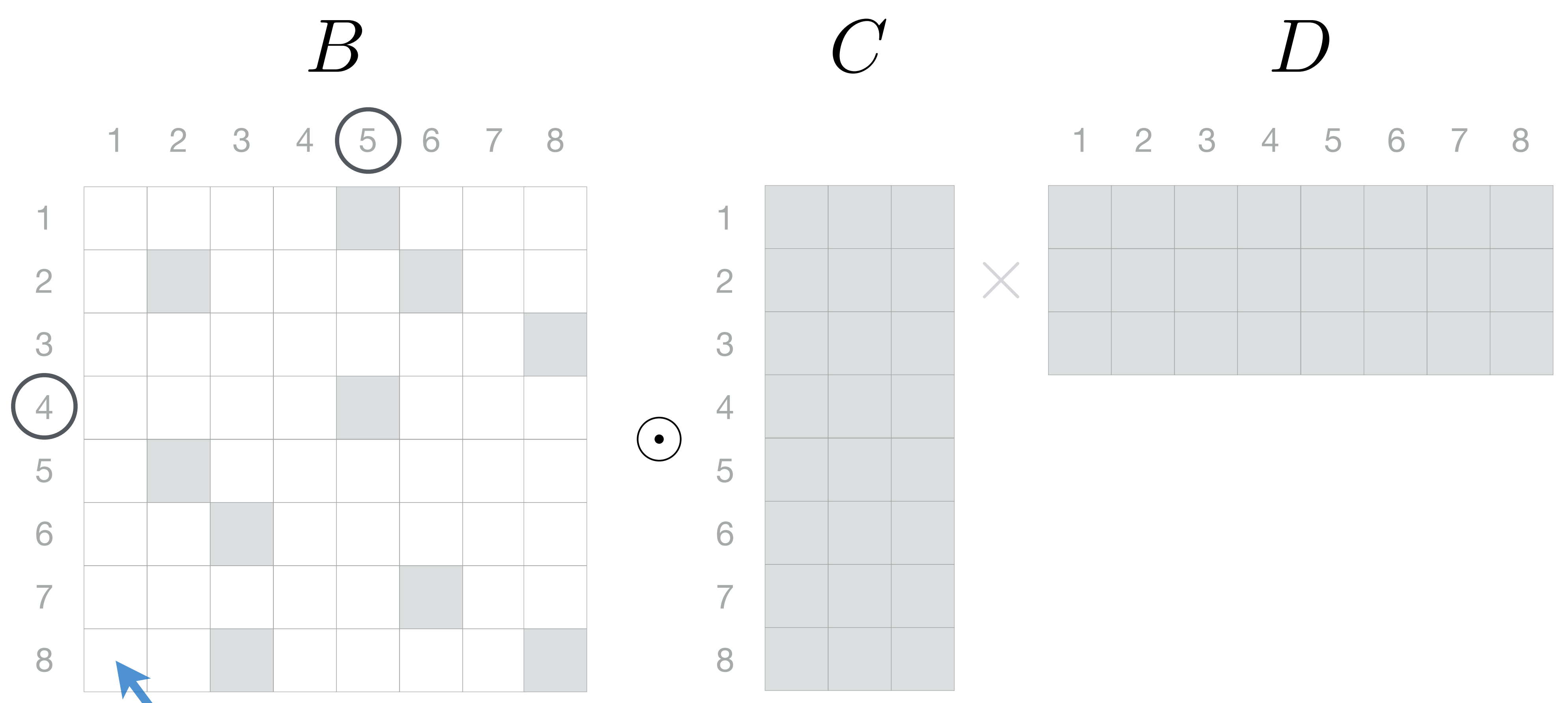
SDDMM

$$A = B \odot (CD)$$



Sampled Dense-Dense Matrix Multiplication

Generated Sparse Code Performance Matches Hand-Optimized Libraries

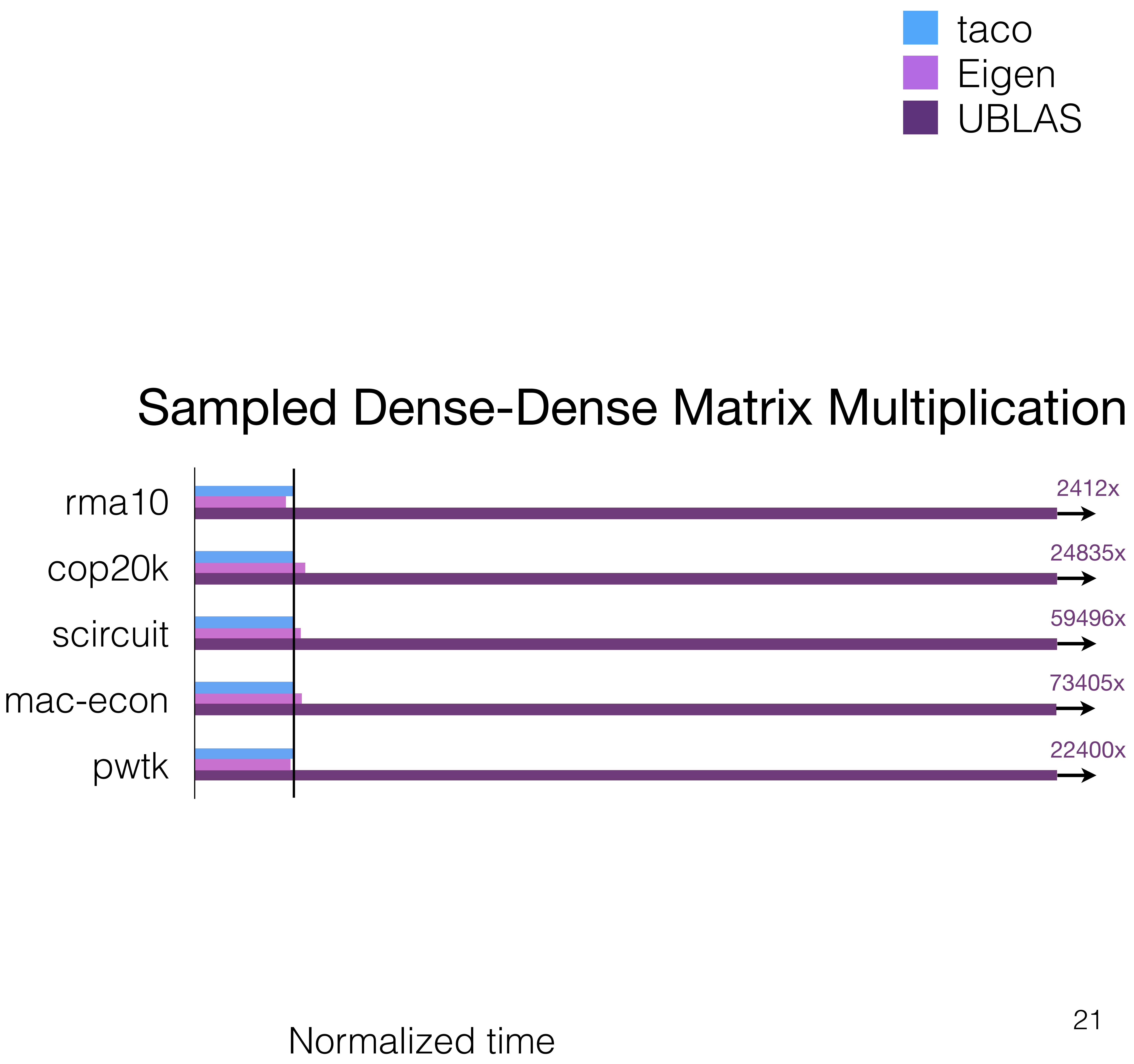


This dot product need not be computed

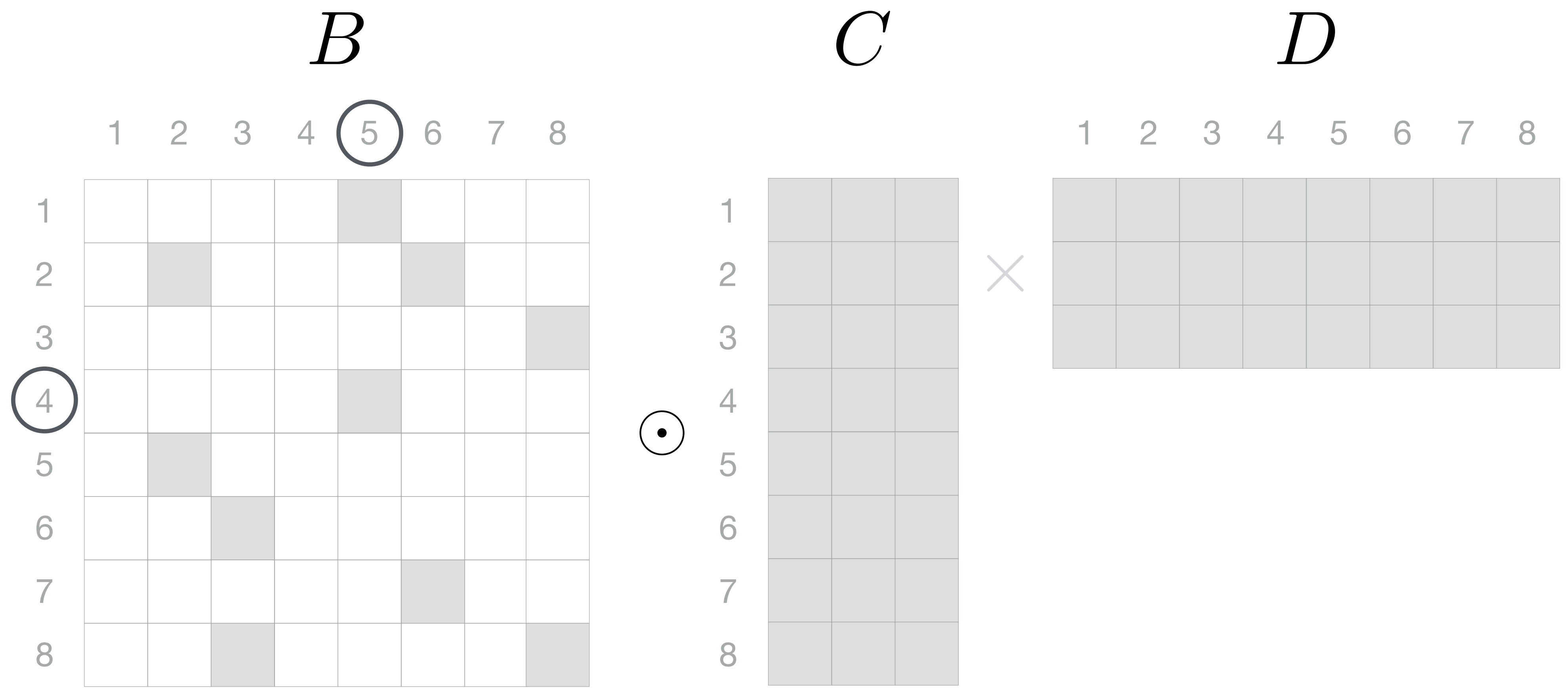
~~64 inner product~~
10 inner product

SDDMM

$$A = B \odot (CD)$$


Sampled Dense-Dense Matrix Multiplication

Generated Sparse Code Performance Matches Hand-Optimized Libraries

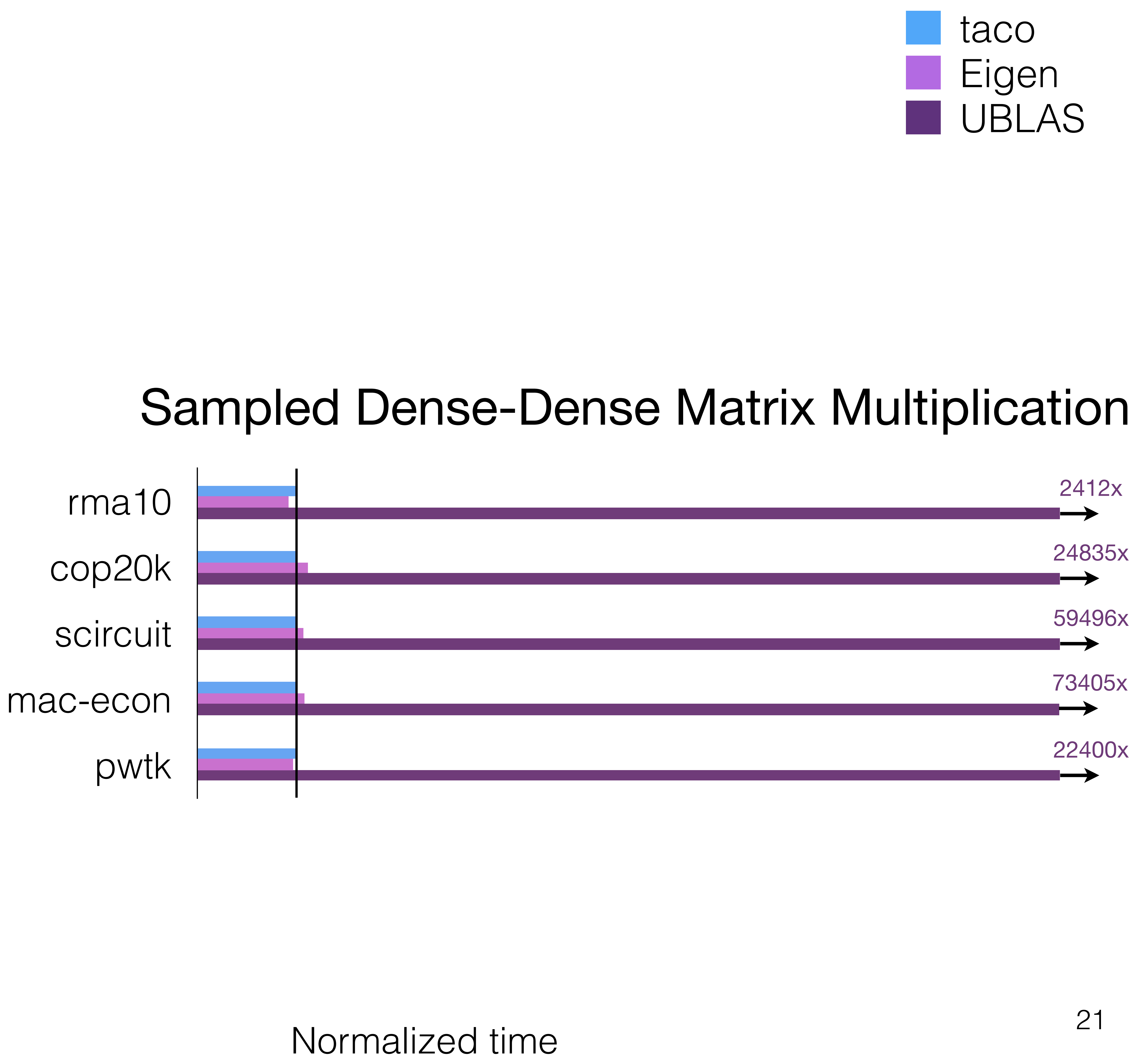


We will generate fused operations

SDDMM

$$A = B \odot (CD)$$

Sampled Dense-Dense Matrix Multiplication

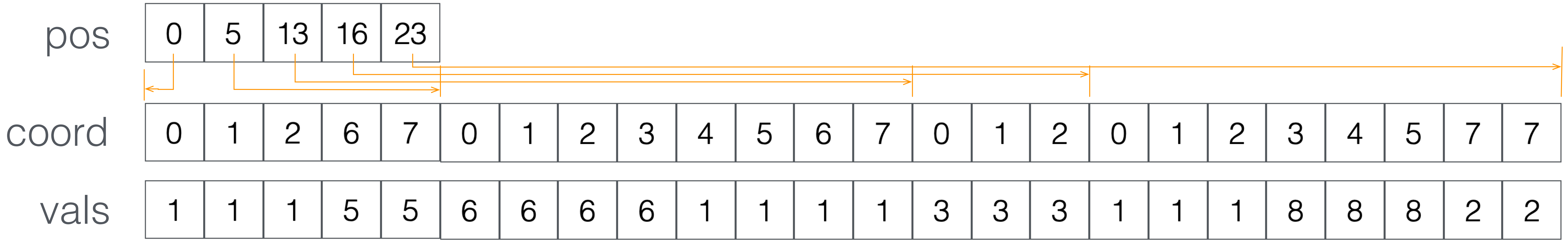


Sparsity Beyond Zero Fill Values

	0	1	2	3	4	5	6	7
0	1	1	1	0	0	0	5	5
1	6	6	6	6	1	1	1	1
2	3	3	3	0	0	0	0	0
3	1	1	1	8	8	8	2	2

Sparsity Beyond Zero Fill Values

Compressed Level Format



pos	0	5	13	16	23

coord	0	1	2	6	7	0	1	2	3	4	5	6	7	0	1	2	0	1	2	3	4	5	7	7
vals	1	1	1	5	5	6	6	6	6	1	1	1	1	3	3	3	1	1	1	8	8	8	2	2

	0	1	2	3	4	5	6	7
0	1	1	1	0	0	0	5	5
1	6	6	6	6	1	1	1	1
2	3	3	3	0	0	0	0	0
3	1	1	1	8	8	8	2	2

Sparsity Beyond Zero Fill Values

Compressed Level Format

pos	0	5	13	16	23																			
coord	0	1	2	6	7	0	1	2	3	4	5	6	7	0	1	2	0	1	2	3	4	5	7	7
vals	1	1	1	5	5	6	6	6	6	1	1	1	1	3	3	3	1	1	1	8	8	8	2	2

	0	1	2	3	4	5	6	7
0	1	1	1	0	0	0	5	5
1	6	6	6	6	1	1	1	1
2	3	3	3	0	0	0	0	0
3	1	1	1	8	8	8	2	2

Compressed Level Format with a Fill Value

pos	0	5	9	17	21																	
coord	3	4	5	6	7	0	1	2	3	0	1	2	3	4	5	6	7	3	4	5	6	7
vals	0	0	0	5	5	6	6	6	6	3	3	3	0	0	0	0	0	8	8	8	2	2
Fill	1																					

Sparsity Beyond Zero Fill Values

Compressed Level Format

pos	0	5	13	16	23																			
coord	0	1	2	6	7	0	1	2	3	4	5	6	7	0	1	2	0	1	2	3	4	5	7	7
vals	1	1	1	5	5	6	6	6	6	1	1	1	1	3	3	3	1	1	1	8	8	8	2	2

Compressed Level Format with a Fill Value

pos	0	5	9	17	21																	
coord	3	4	5	6	7	0	1	2	3	0	1	2	3	4	5	6	7	3	4	5	6	7
vals	0	0	0	5	5	6	6	6	6	3	3	3	0	0	0	0	0	8	8	8	2	2
Fill	1																					

	0	1	2	3	4	5	6	7
0	1	1	1	0	0	0	5	5
1	6	6	6	6	1	1	1	1
2	3	3	3	0	0	0	0	0
3	1	1	1	8	8	8	2	2

Run Length Encoding (RLE) Level Format

- Extension of the Compressed Format
- Last value is the Fill Value

Sparsity Beyond Zero Fill Values

	0	1	2	3	4	5	6	7
0	1	1	1	0	0	0	5	5
1	6	6	6	6	1	1	1	1
2	3	3	3	0	0	0	0	0
3	1	1	1	8	8	8	2	2

Compressed Level Format

pos	0	5	13	16	23

coord	0	1	2	6	7	0	1	2	3	4	5	6	7	0	1	2	0	1	2	3	4	5	7	7
vals	1	1	1	5	5	6	6	6	6	1	1	1	1	3	3	3	1	1	1	8	8	8	2	2

Compressed Level Format with a Fill Value

pos	0	5	9	17	21

coord	3	4	5	6	7	0	1	2	3	0	1	2	3	4	5	6	7	3	4	5	6	7
vals	0	0	0	5	5	6	6	6	6	3	3	3	0	0	0	0	0	8	8	8	2	2

Fill	1

Run Length Encoding (RLE) Level Format

pos	0	3	5	7	9

coord	0	3	6	0	4	0	3	0	3	6
vals	1	0	5	6	1	3	0	1	8	2

- Extension of the Compressed Format
- Last value is the Fill Value

Sparsity Beyond Zero Fill Values

Compressed Level Format

pos	0	5	13	16	23																		
coord	0	1	2	6	7	0	1	2	3	4	5	6	7	0	1	2	0	1	2	3	4	5	7	7
vals	1	1	1	5	5	6	6	6	6	1	1	1	1	3	3	3	1	1	1	8	8	8	2	2

Compressed Level Format with a Fill Value

pos	0	5	9	17	21																	
coord	3	4	5	6	7	0	1	2	3	0	1	2	3	4	5	6	7	3	4	5	6	7
vals	0	0	0	5	5	6	6	6	6	3	3	3	0	0	0	0	0	8	8	8	2	2
Fill	1																					

Run Length Encoding (RLE) Level Format

pos	0	3	5	7	9					
coord	0	3	6	0	4	0	3	0	3	6
vals	1	0	5	6	1	3	0	1	8	2

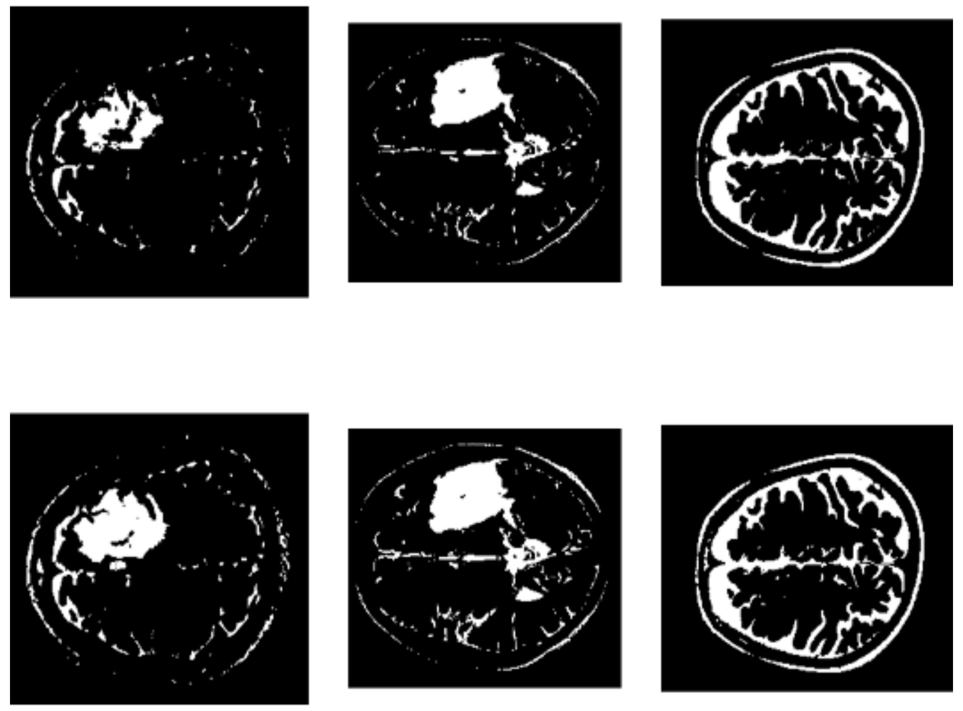
- Extension of the Compressed Format
- Last value is the Fill Value

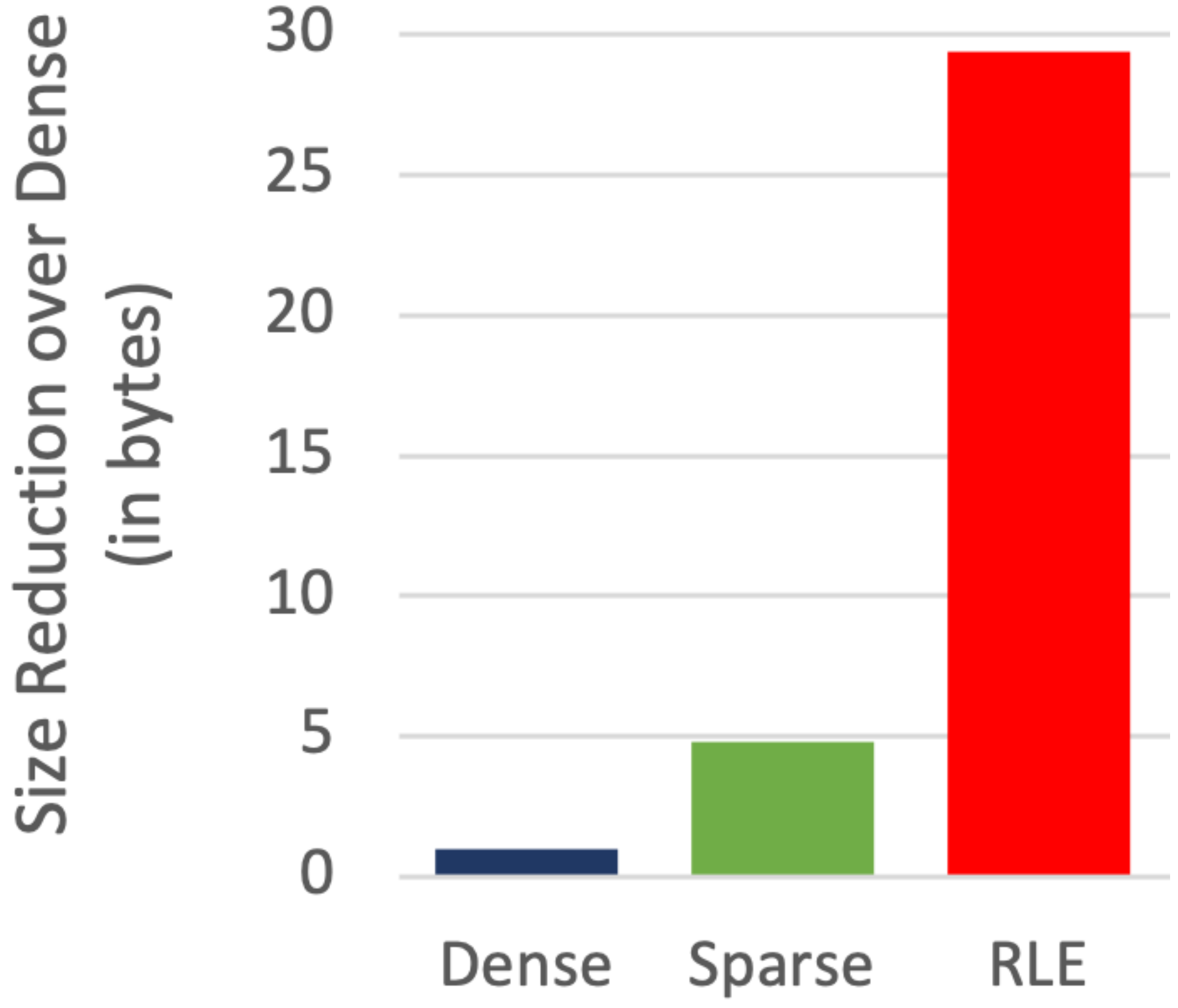
	0	1	2	3	4	5	6	7
0	1	1	1	0	0	0	5	5
1	6	6	6	6	1	1	1	1
2	3	3	3	0	0	0	0	0
3	1	1	1	8	8	8	2	2

Unifying Sparsity and Lossless Compression

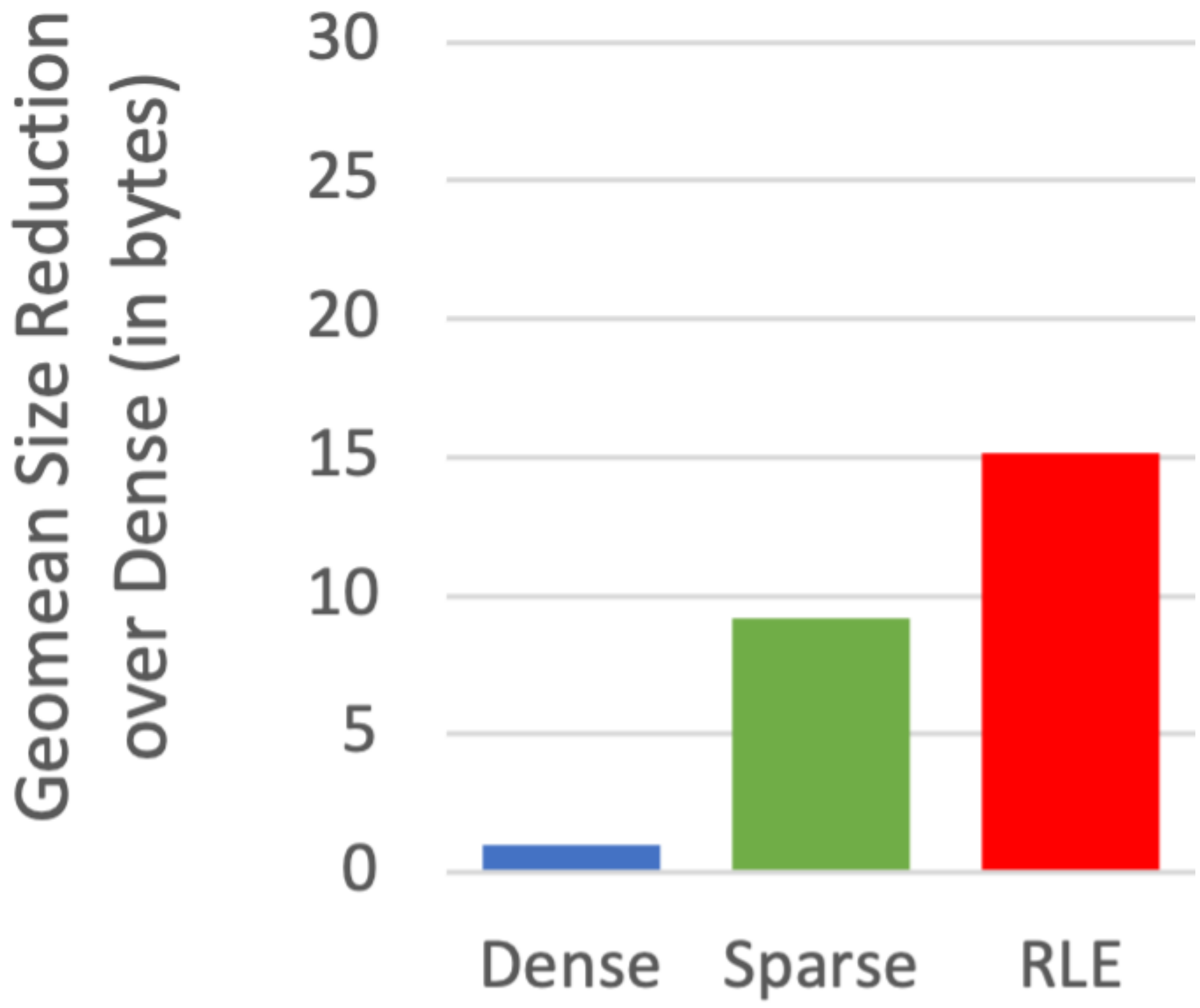
Performance Advantage In Lossless Compression

Edge Detection of MRI Image



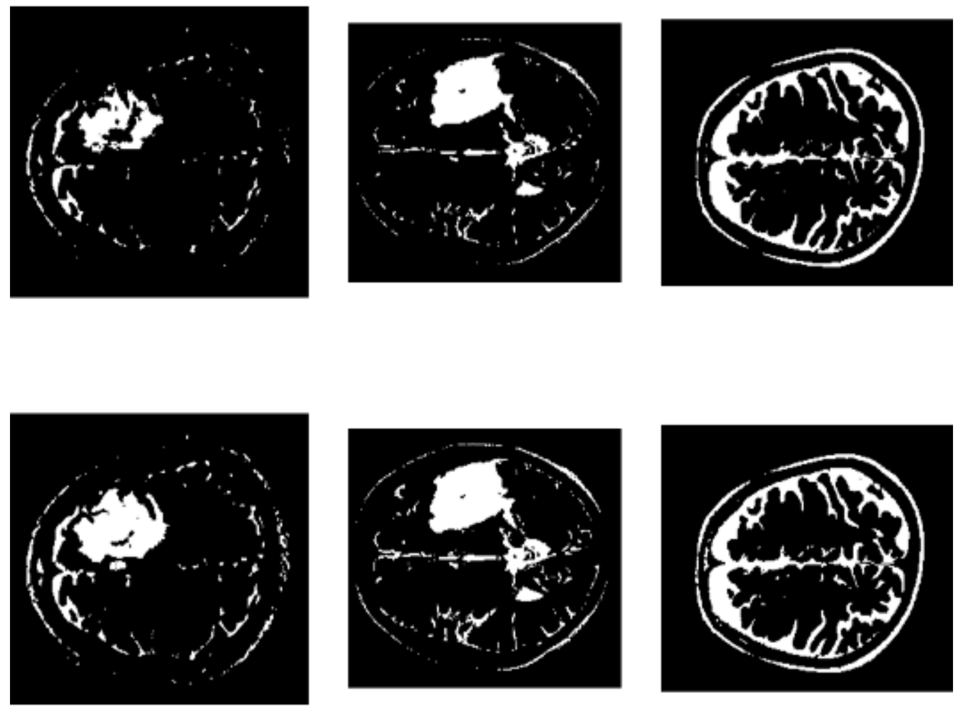


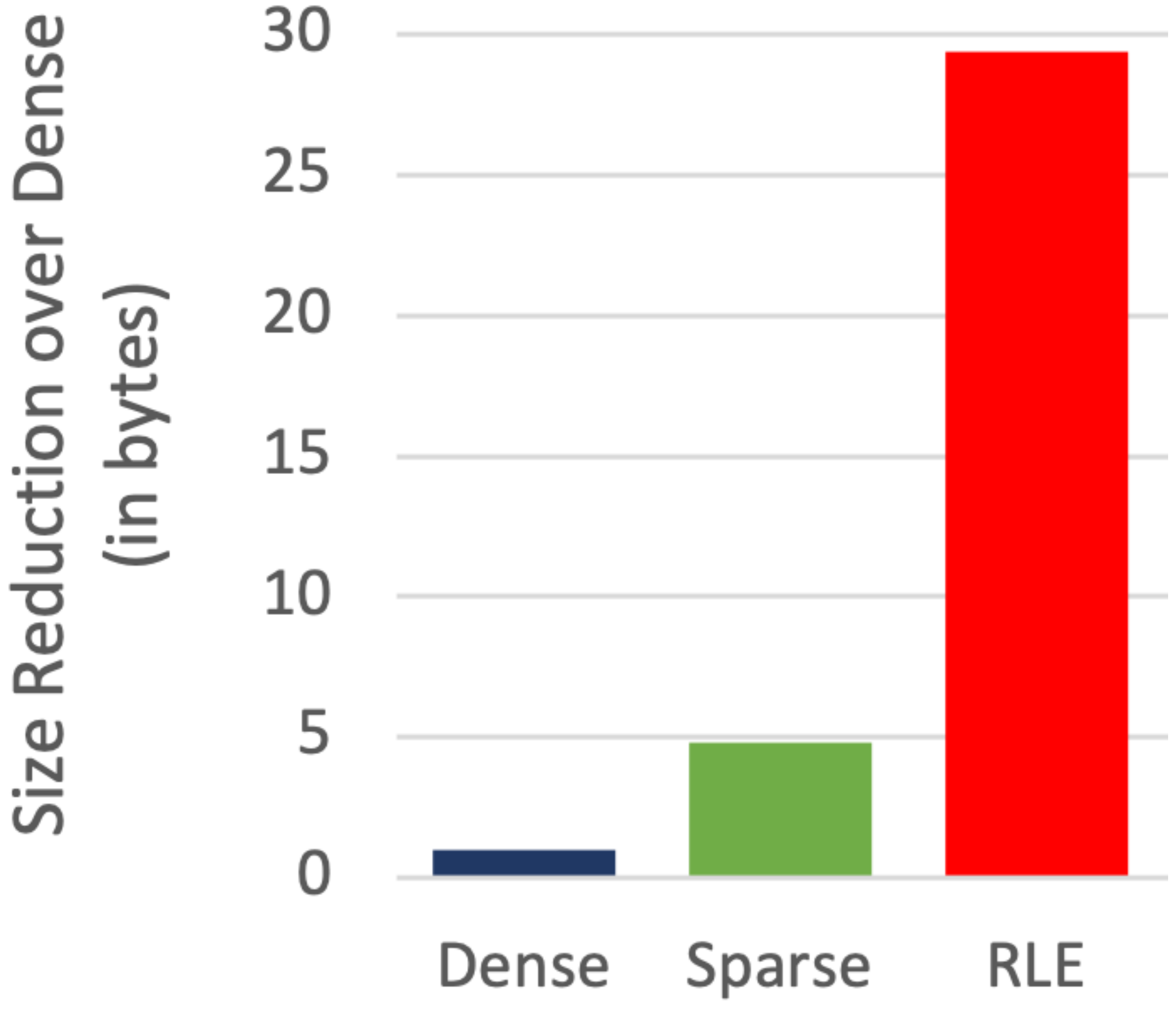
Alpha Blending of Two Videos



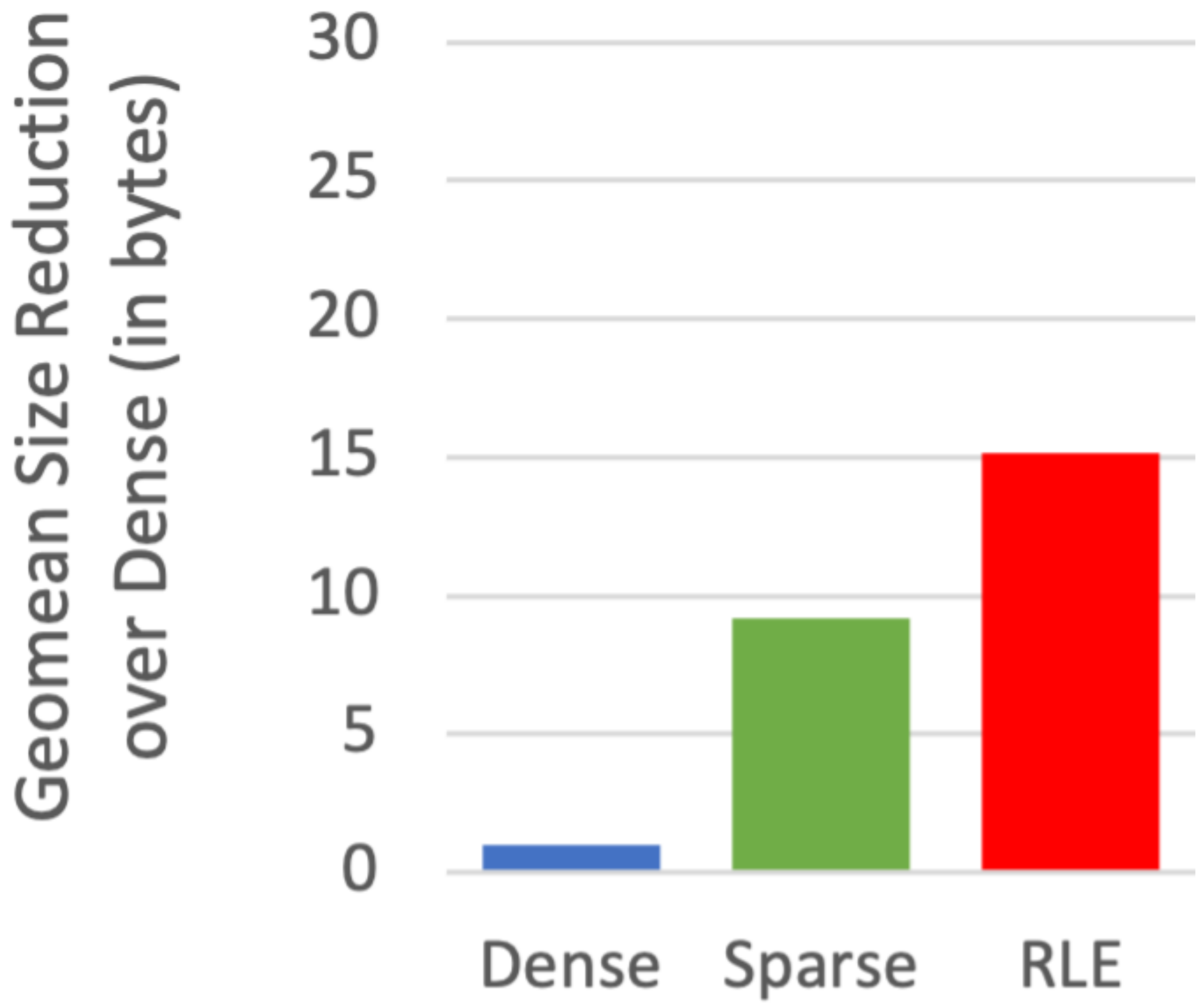
Performance Advantage In Lossless Compression

Edge Detection of MRI Image



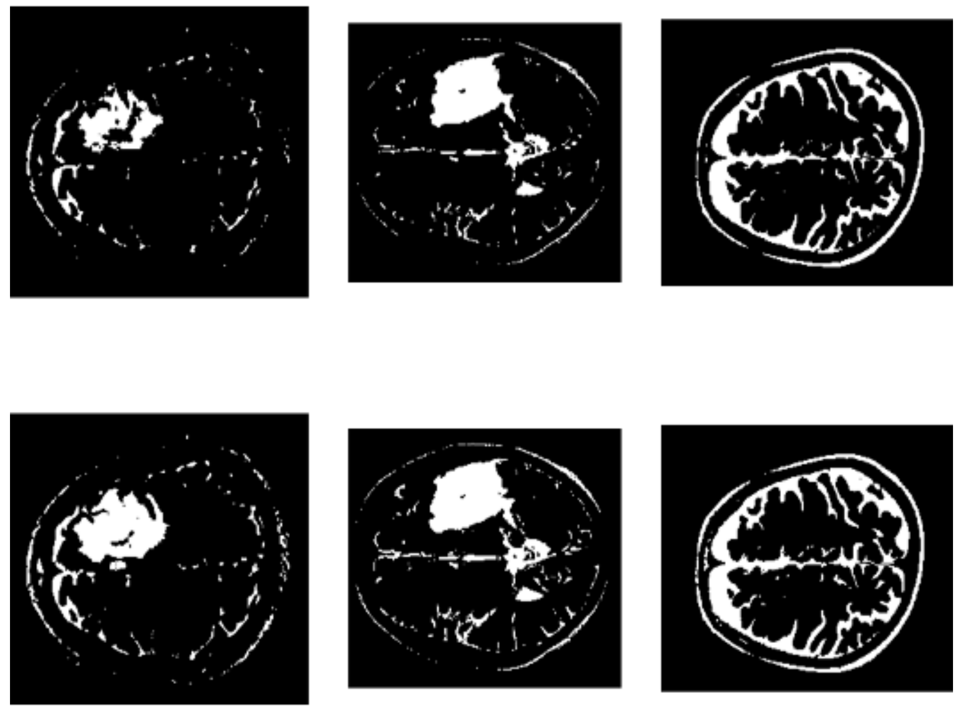


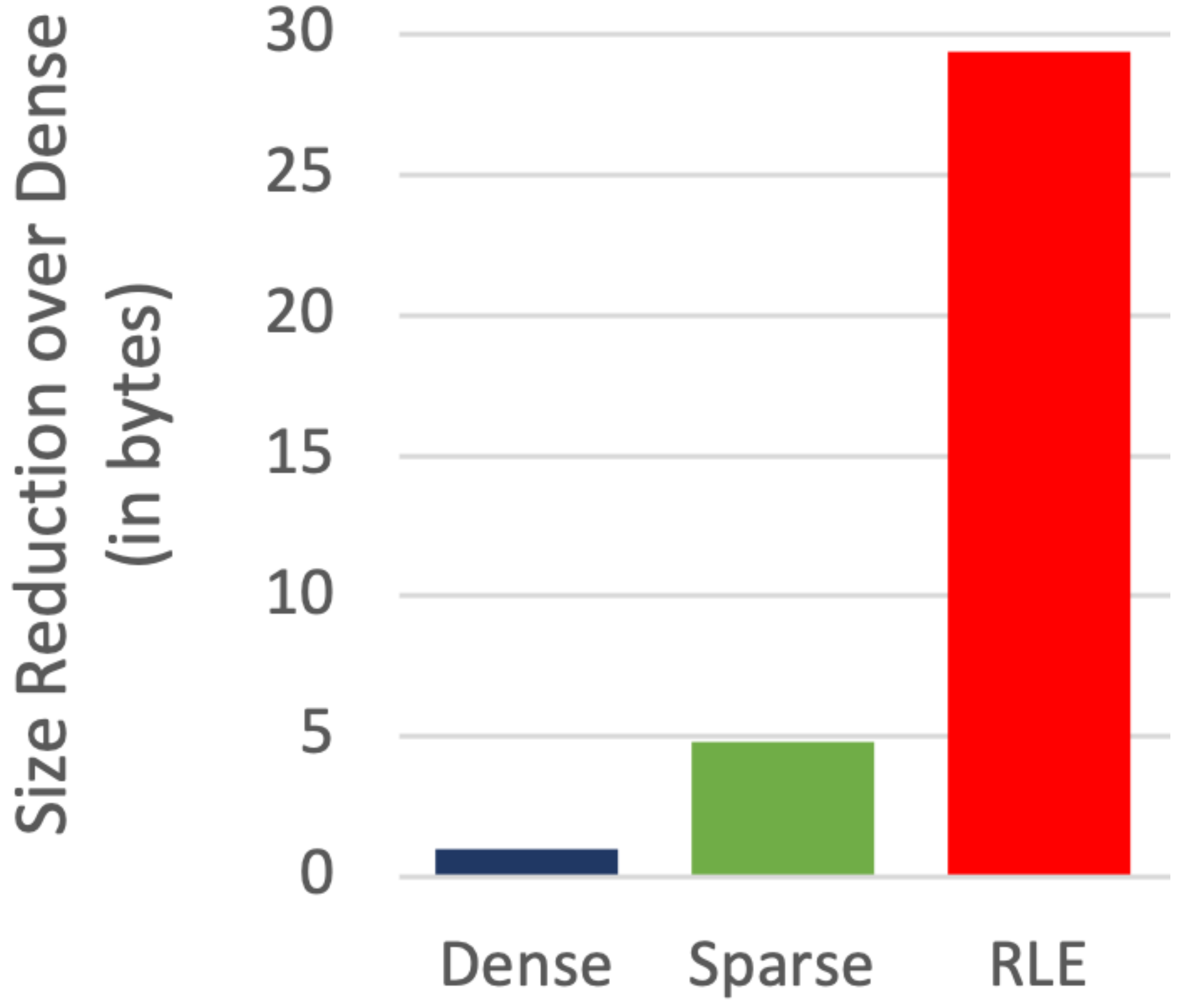
Alpha Blending of Two Videos

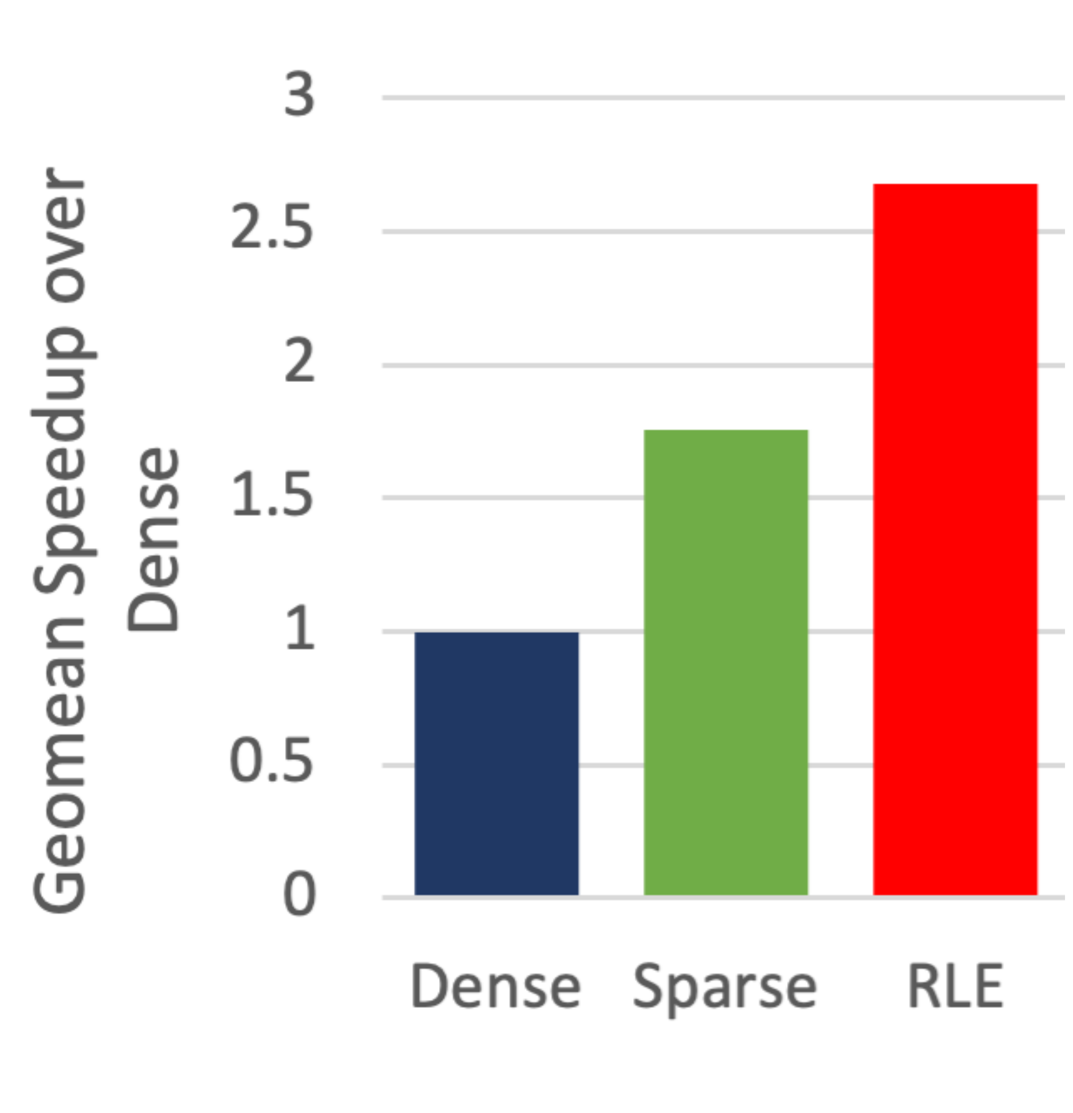


Performance Advantage In Lossless Compression

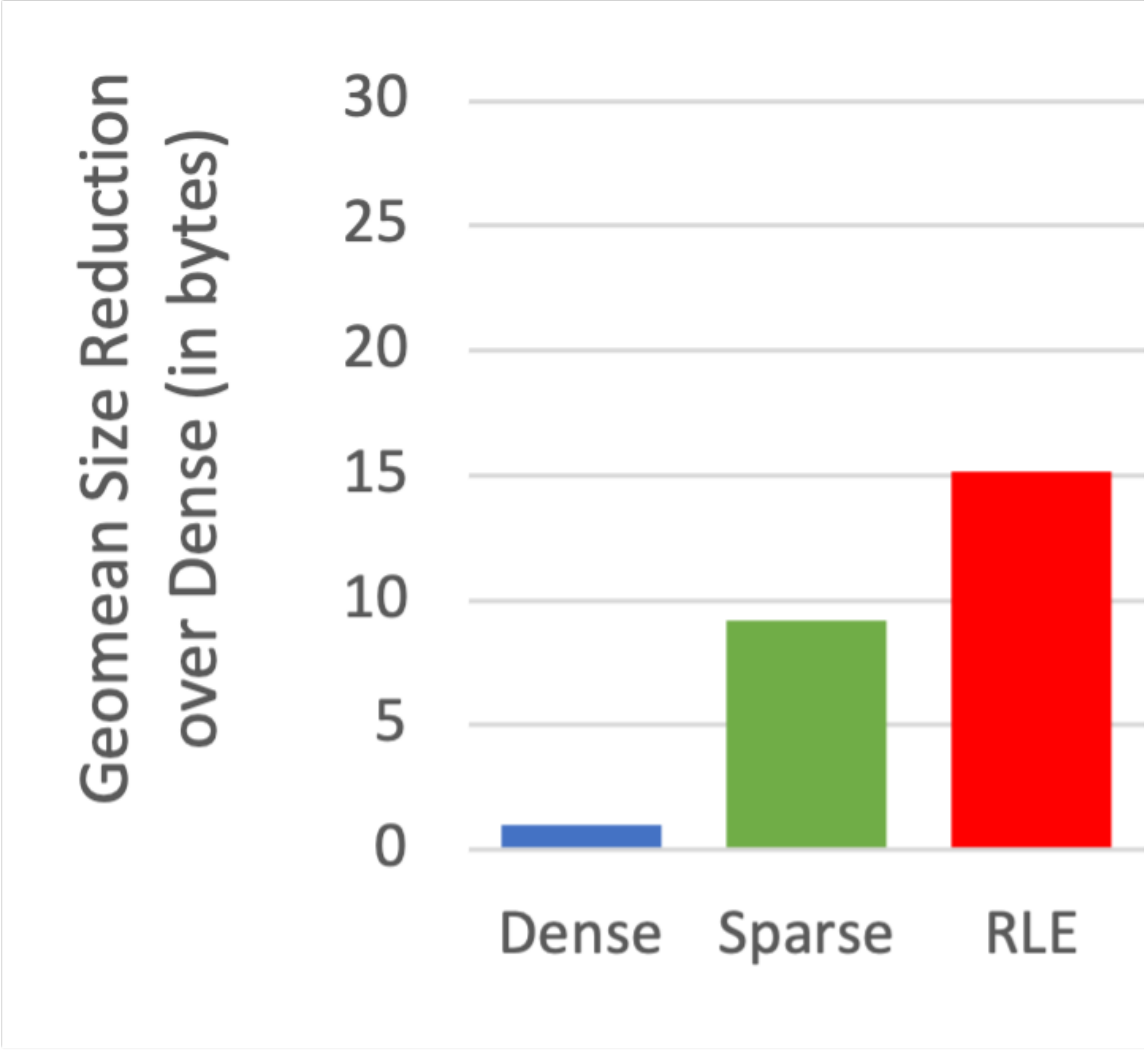
Edge Detection of MRI Image

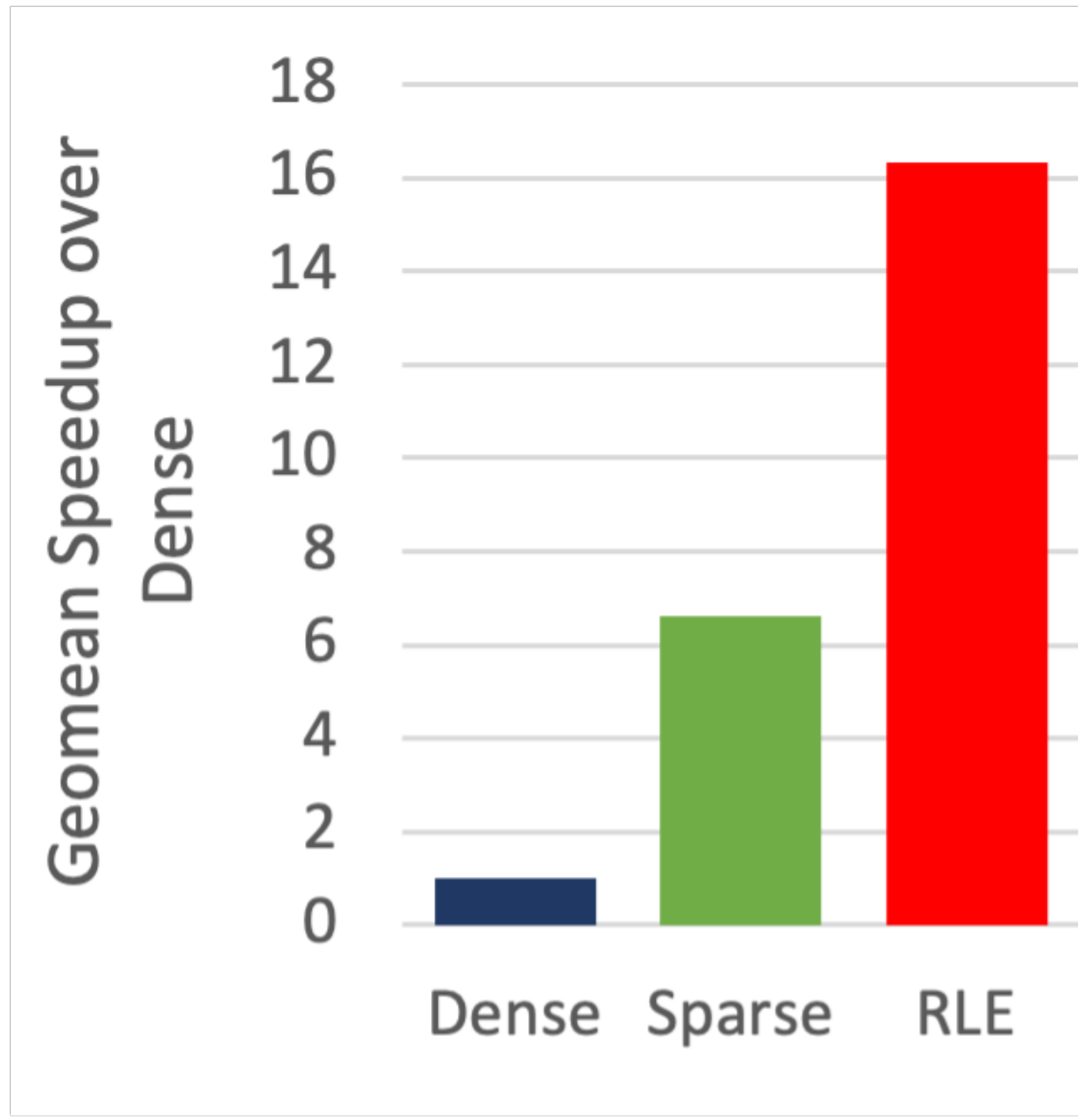






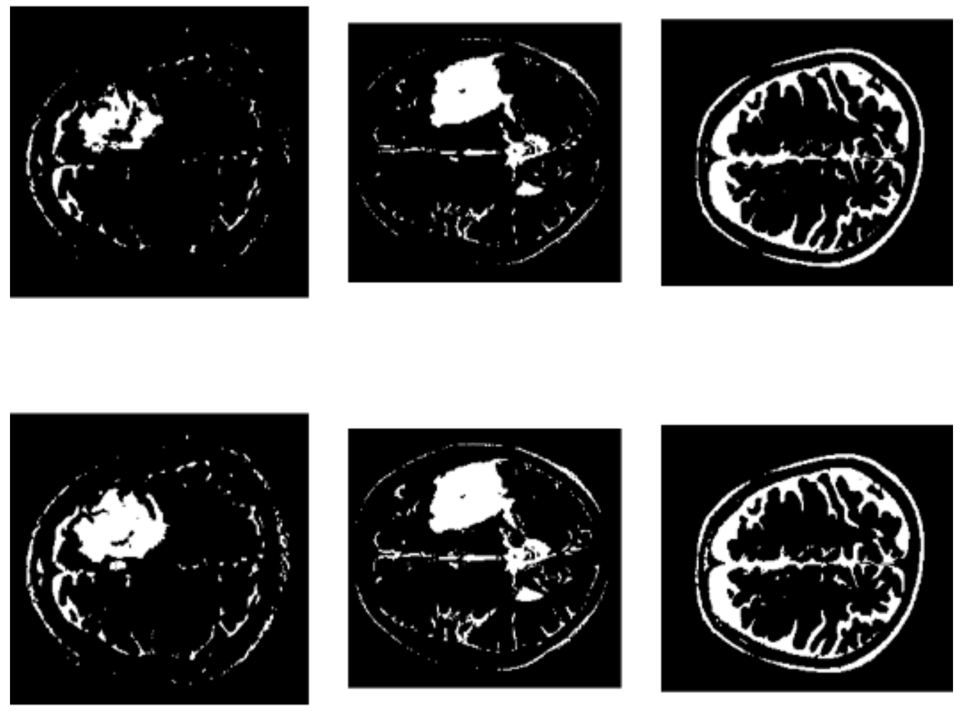
Alpha Blending of Two Videos

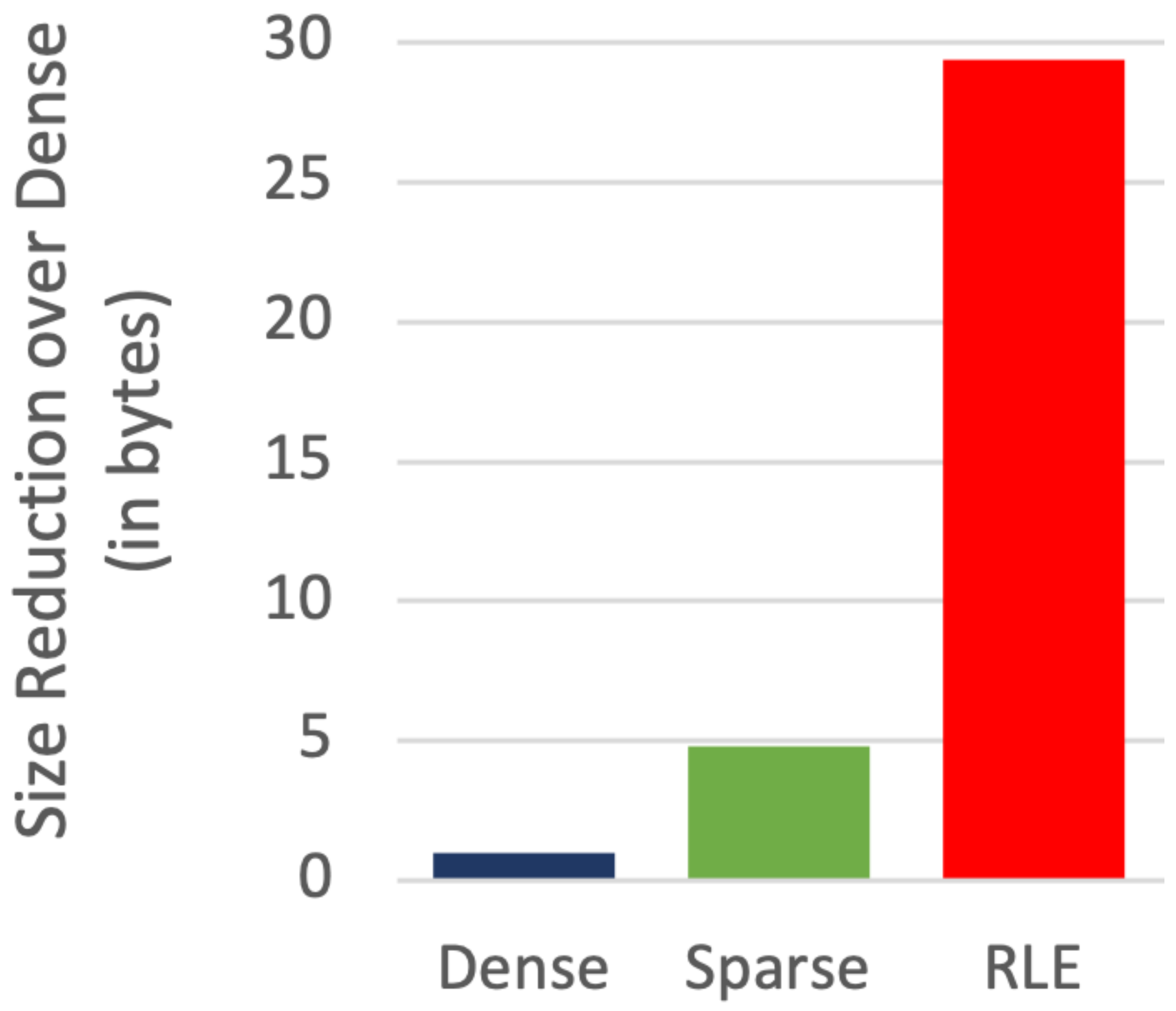


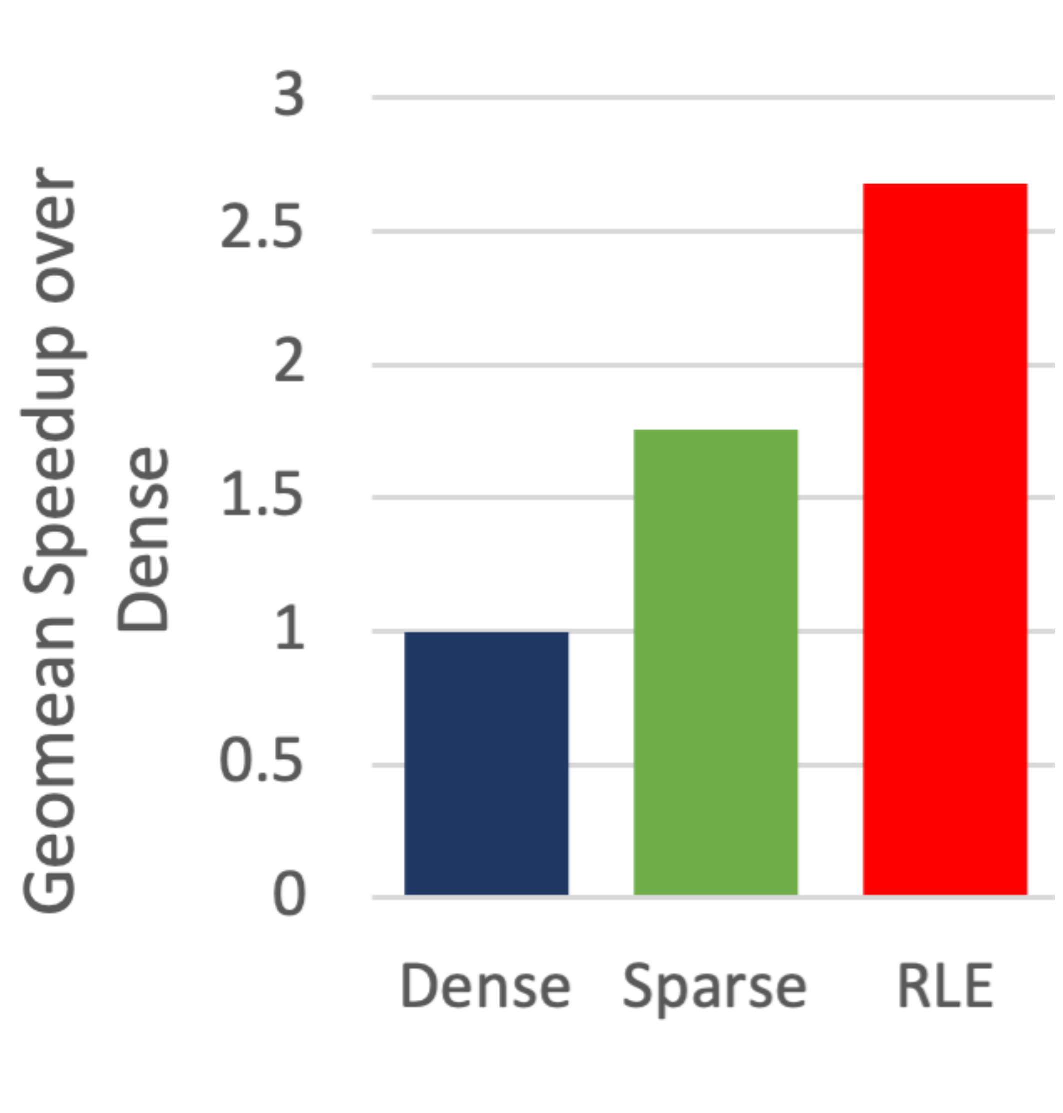


Performance Advantage In Lossless Compression

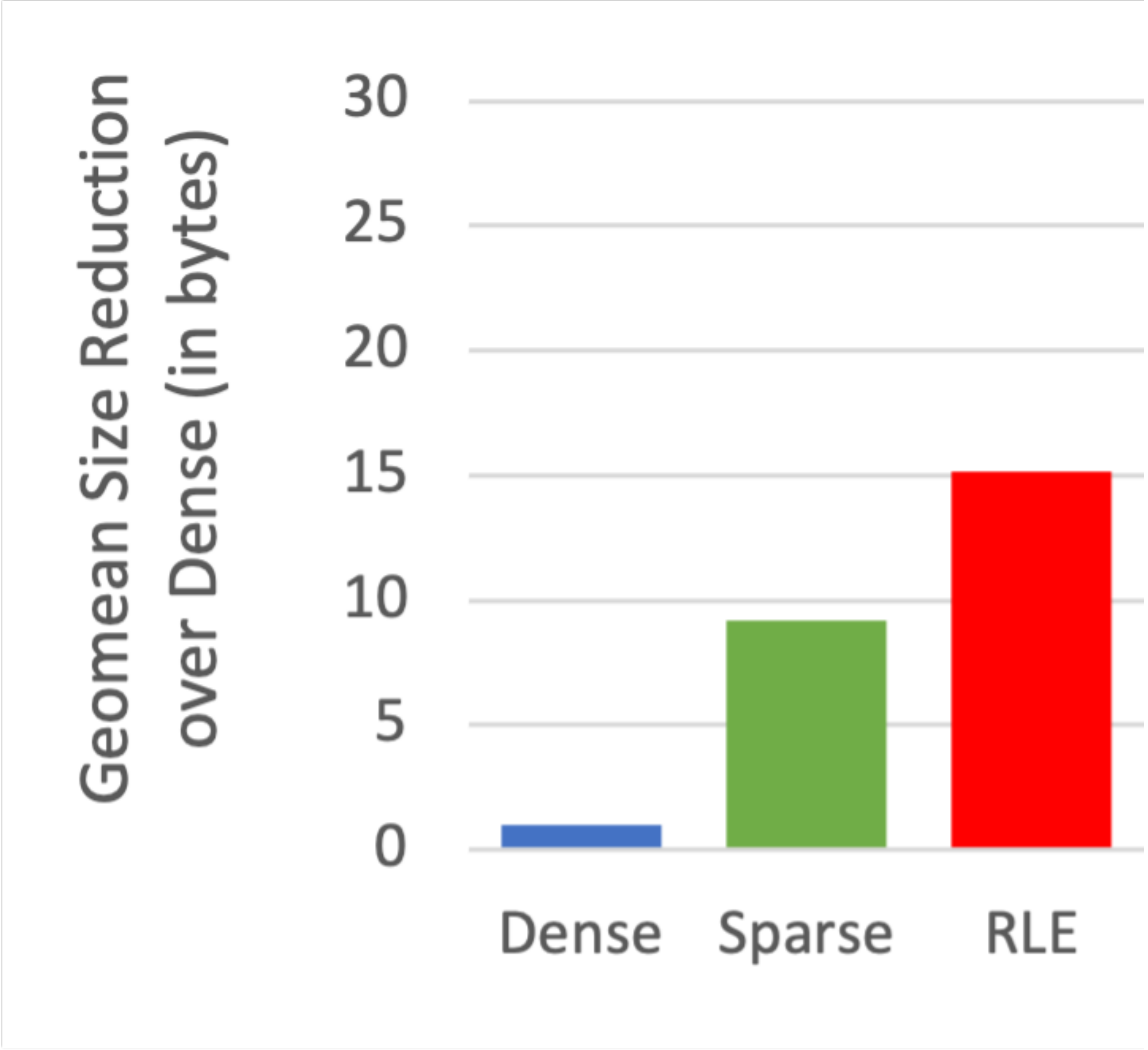
Edge Detection of MRI Image

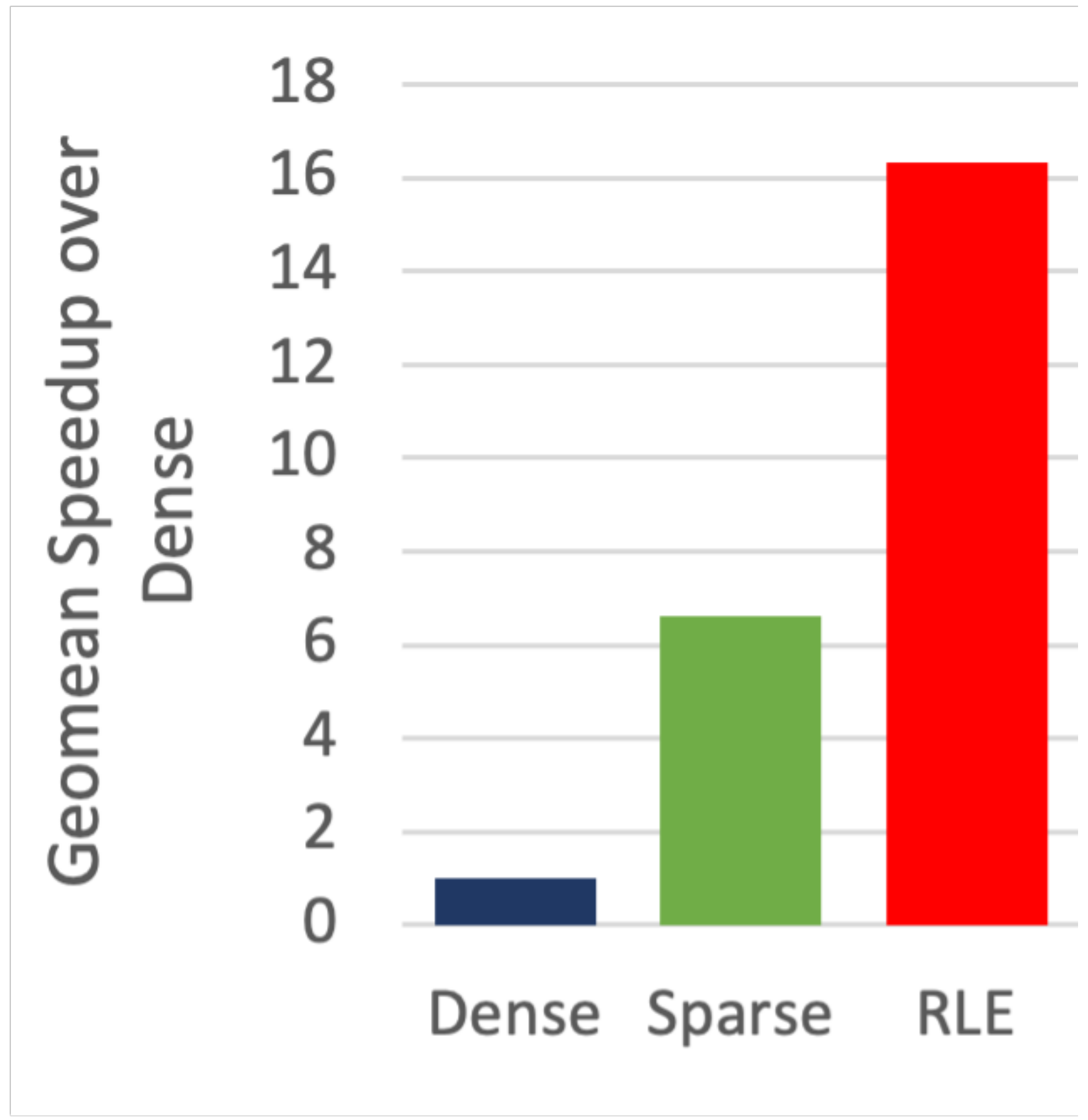


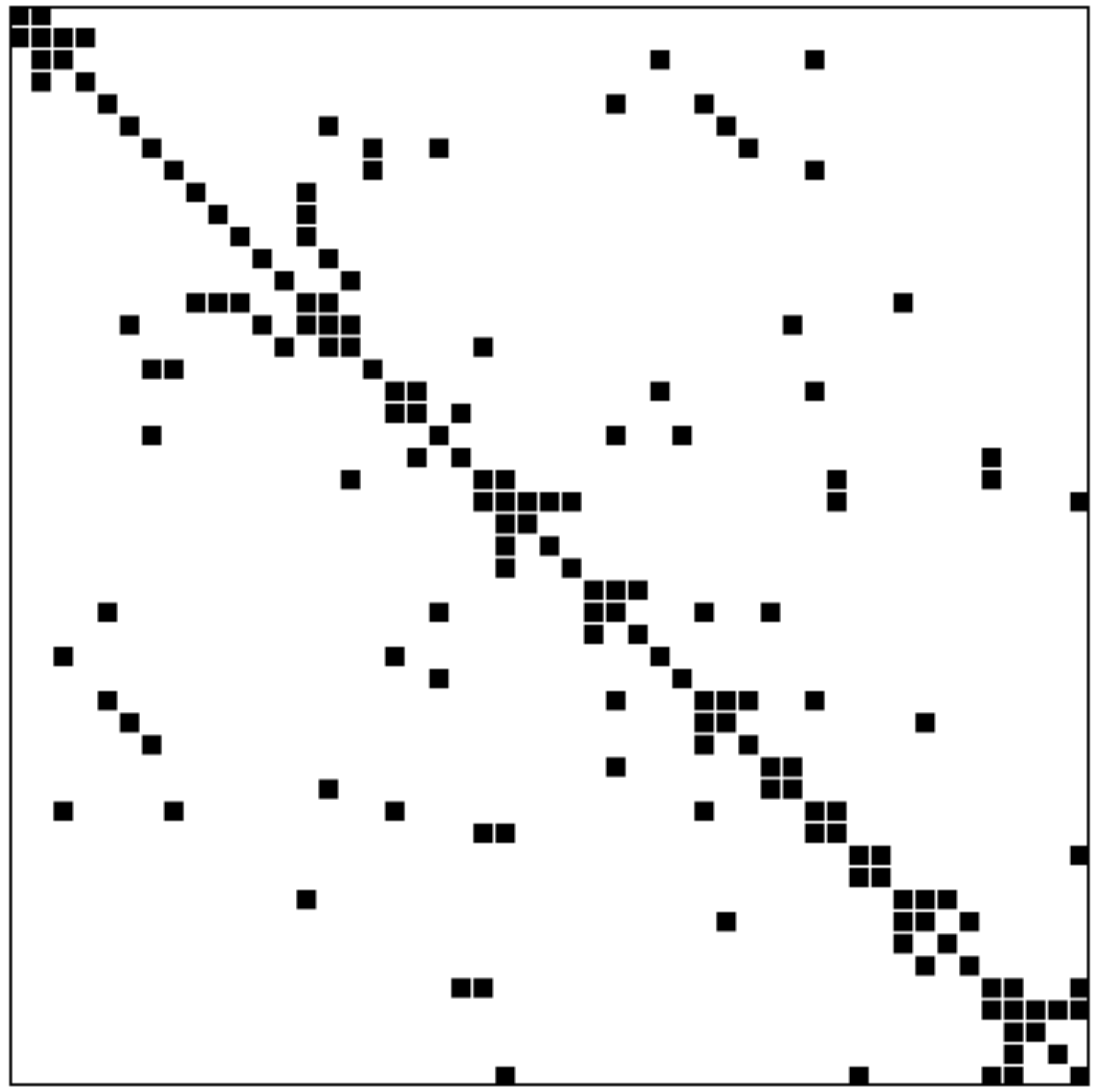




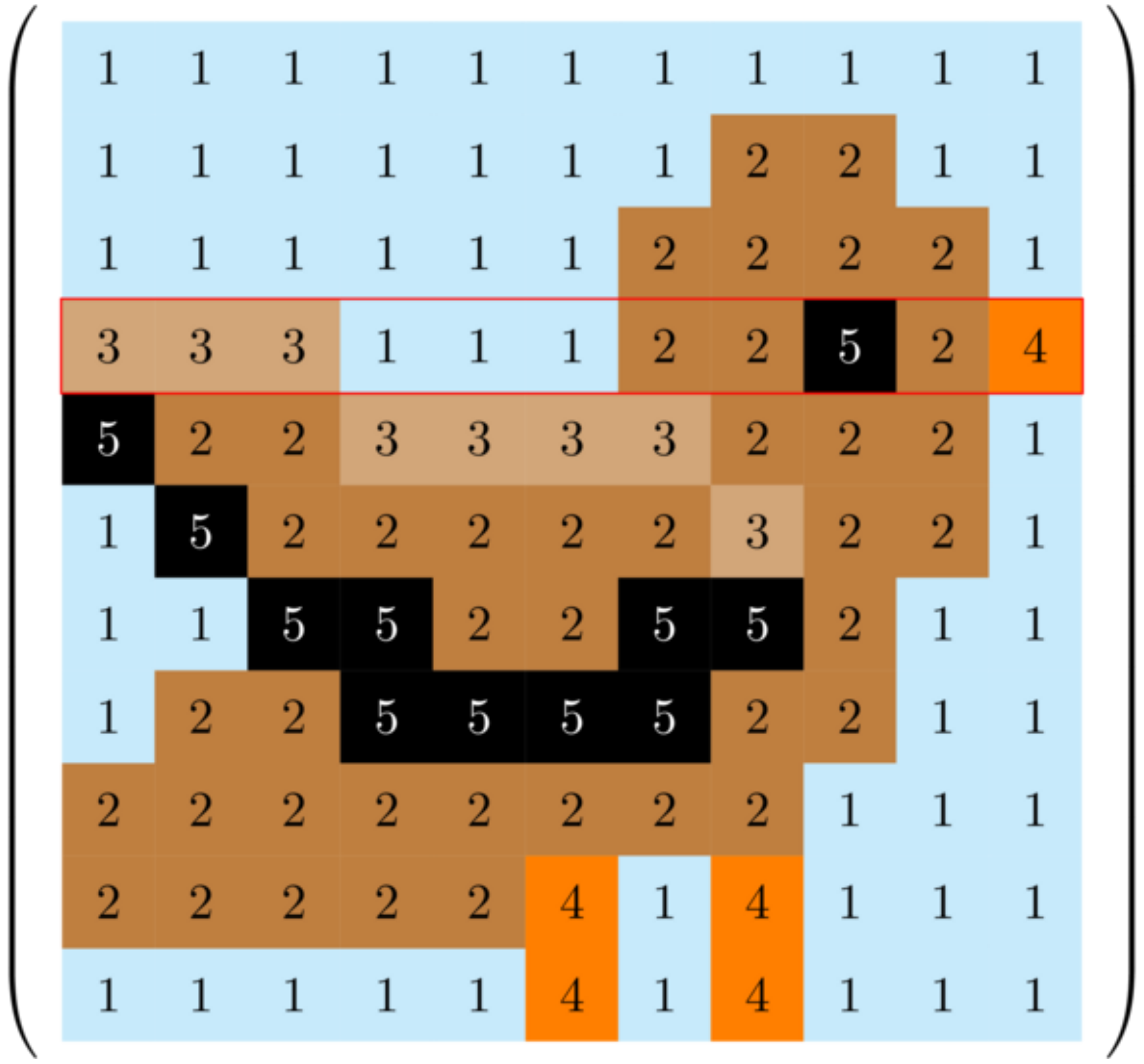
Alpha Blending of Two Videos







+



$$\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 1 \\
3 & 3 & 3 & 1 & 1 & 1 & 2 & 2 & 5 & 2 & 4 \\
5 & 2 & 2 & 3 & 3 & 3 & 3 & 2 & 2 & 2 & 1 \\
1 & 5 & 2 & 2 & 2 & 2 & 2 & 3 & 2 & 2 & 1 \\
1 & 1 & 5 & 5 & 2 & 2 & 5 & 5 & 2 & 1 & 1 \\
1 & 2 & 2 & 5 & 5 & 5 & 5 & 2 & 2 & 1 & 1 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 & 2 & 4 & 1 & 4 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 4 & 1 & 4 & 1 & 1 & 1
\end{pmatrix}$$

?

Example: Dot Product Of Two Vectors

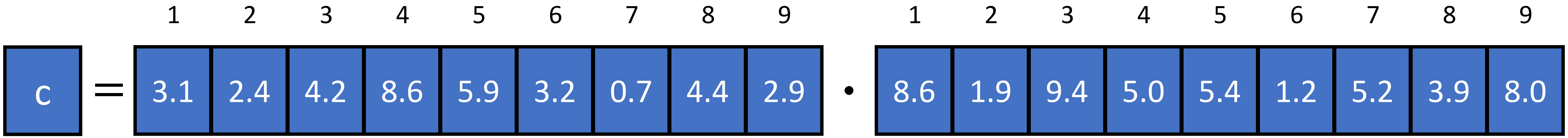
$$c = \sum_i a_i \cdot b_i$$

		1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7	8	9
c	=	3.1	2.4	4.2	8.6	5.9	3.2	0.7	4.4	2.9	•	8.6	1.9	9.4	5.0	5.4	1.2	5.2	3.9	8.0

a is a length n vector

b is a length n vector

Dense Arrays Store Every Value They Represent



c = ·

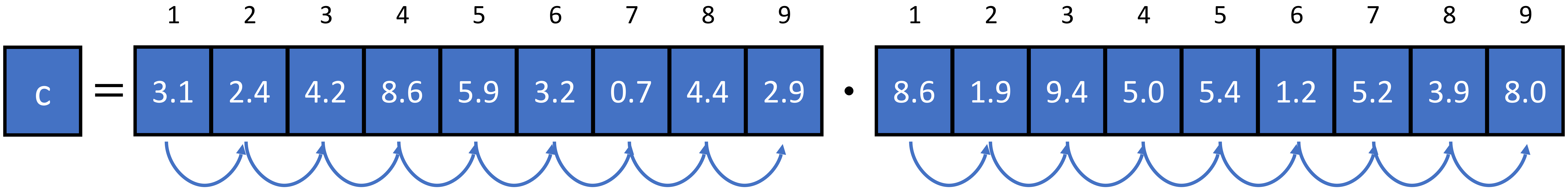
1	2	3	4	5	6	7	8	9
3.1	2.4	4.2	8.6	5.9	3.2	0.7	4.4	2.9

1	2	3	4	5	6	7	8	9
8.6	1.9	9.4	5.0	5.4	1.2	5.2	3.9	8.0

a is dense

b is dense

Dense Arrays Store Every Value They Represent



c =

1	2	3	4	5	6	7	8	9
3.1	2.4	4.2	8.6	5.9	3.2	0.7	4.4	2.9

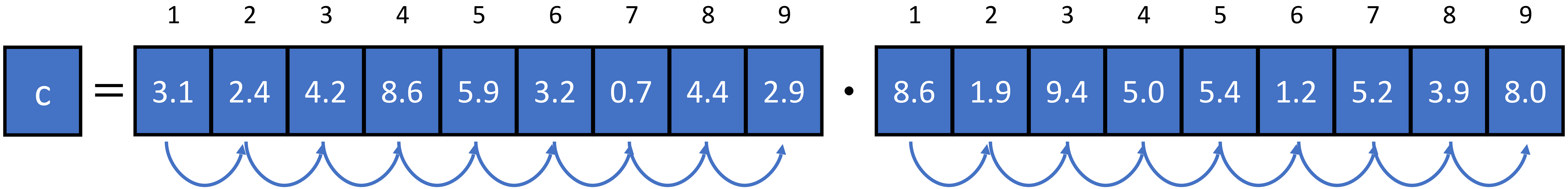
•

1	2	3	4	5	6	7	8	9
8.6	1.9	9.4	5.0	5.4	1.2	5.2	3.9	8.0

a is dense

b is dense

Dense Arrays Store Every Value They Represent



c = a · b

	1	2	3	4	5	6	7	8	9
a	3.1	2.4	4.2	8.6	5.9	3.2	0.7	4.4	2.9
b	8.6	1.9	9.4	5.0	5.4	1.2	5.2	3.9	8.0

```
for i = 1:n
    c += a[i] * b[i]
```

a is dense

b is dense

Sparse Arrays Interpret Stored Data Differently

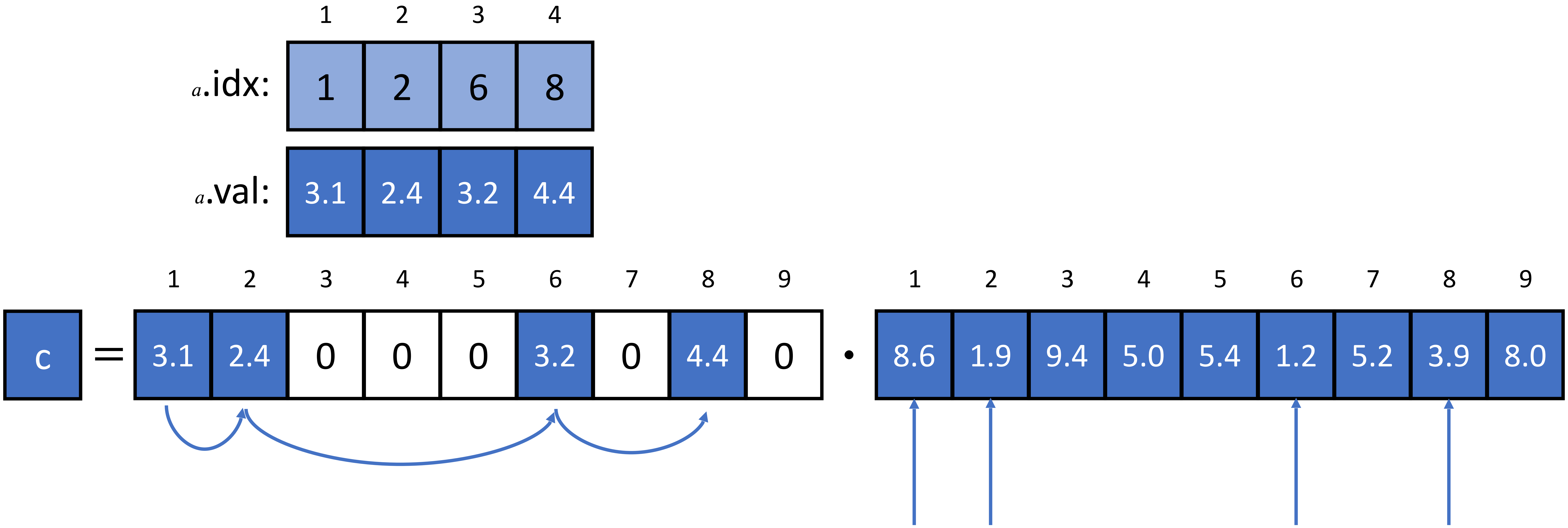
	1	2	3	4
a.idx:	1	2	6	8
a.val:	3.1	2.4	3.2	4.4

$$\boxed{c} = \begin{array}{|c|c|c|c|c|c|c|c|c|} \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} & \multicolumn{1}{c}{4} & \multicolumn{1}{c}{5} & \multicolumn{1}{c}{6} & \multicolumn{1}{c}{7} & \multicolumn{1}{c}{8} & \multicolumn{1}{c}{9} \\ \hline 3.1 & 2.4 & 0 & 0 & 0 & 3.2 & 0 & 4.4 & 0 \\ \hline \end{array} \cdot \begin{array}{|c|c|c|c|c|c|c|c|c|} \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} & \multicolumn{1}{c}{4} & \multicolumn{1}{c}{5} & \multicolumn{1}{c}{6} & \multicolumn{1}{c}{7} & \multicolumn{1}{c}{8} & \multicolumn{1}{c}{9} \\ \hline 8.6 & 1.9 & 9.4 & 5.0 & 5.4 & 1.2 & 5.2 & 3.9 & 8.0 \\ \hline \end{array}$$

a is sparse

b is dense

Sparse Arrays Interpret Stored Data Differently



	1	2	3	4
a.idx:	1	2	6	8
a.val:	3.1	2.4	3.2	4.4

c =

1	2	3	4	5	6	7	8	9
3.1	2.4	0	0	0	3.2	0	4.4	0

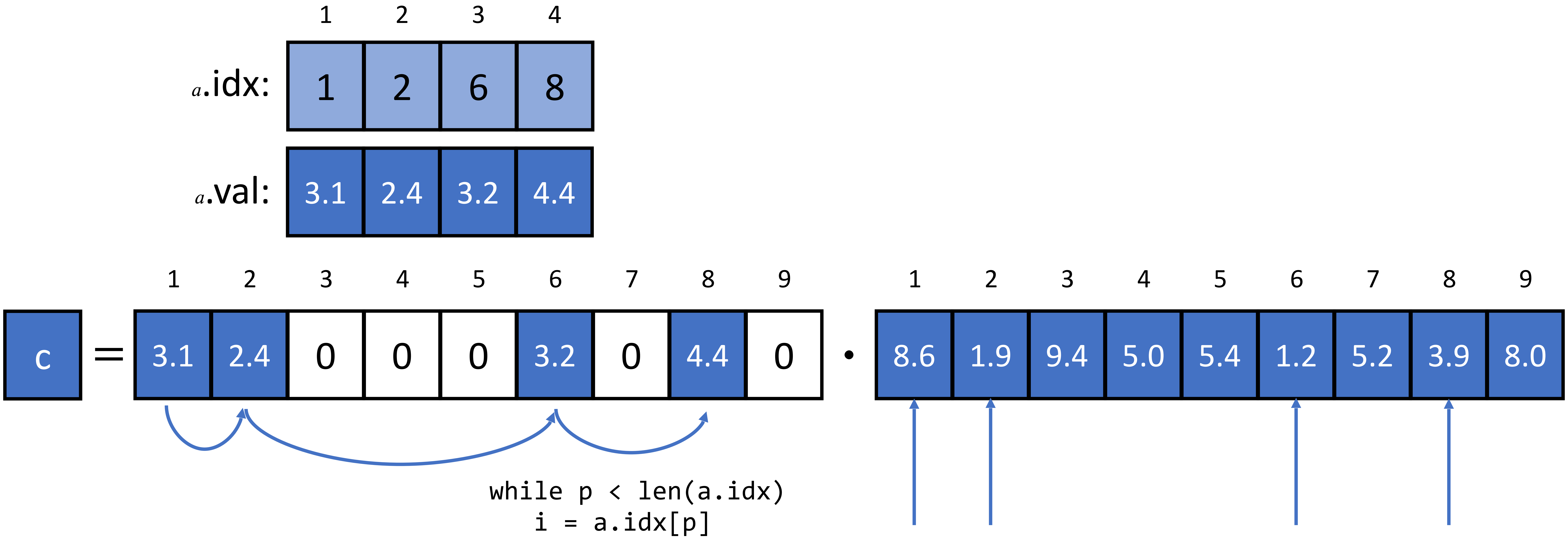
•

1	2	3	4	5	6	7	8	9
8.6	1.9	9.4	5.0	5.4	1.2	5.2	3.9	8.0

a is sparse

b is dense

Sparse Arrays Interpret Stored Data Differently



	1	2	3	4
a.idx:	1	2	6	8
a.val:	3.1	2.4	3.2	4.4

		1	2	3	4	5	6	7	8	9
c	=	3.1	2.4	0	0	0	3.2	0	4.4	0

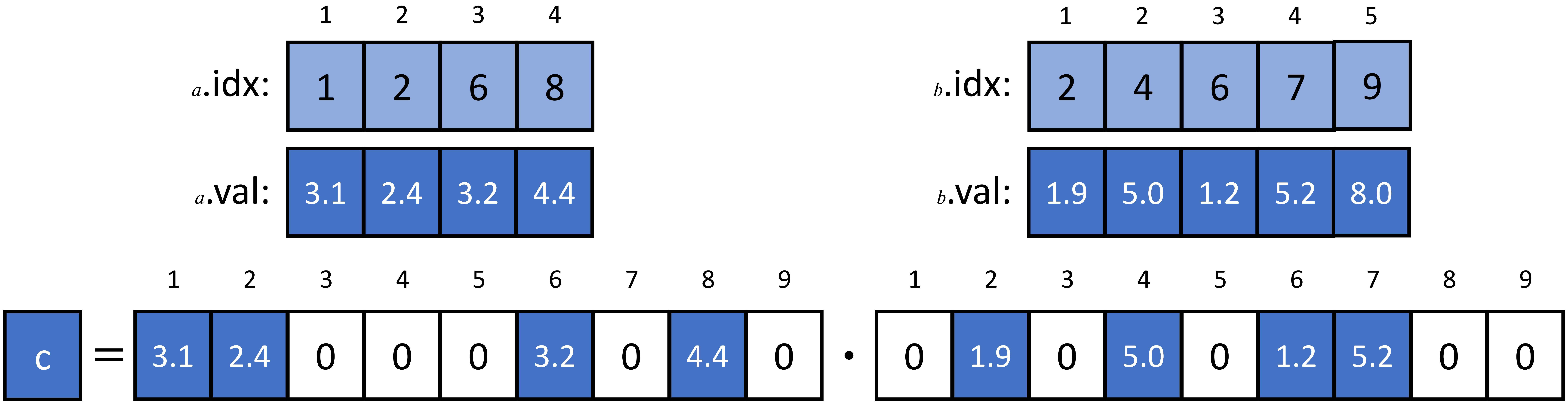
	1	2	3	4	5	6	7	8	9
·	8.6	1.9	9.4	5.0	5.4	1.2	5.2	3.9	8.0

```
while p < len(a.idx)
   i = a.idx[p]
   c += a.val[p] * b[i]
   p += 1
```

a is sparse

b is dense

Merging Multiple Sparse Requires Coordination



	1	2	3	4
a.idx:	1	2	6	8
a.val:	3.1	2.4	3.2	4.4

	1	2	3	4	5
b.idx:	2	4	6	7	9
b.val:	1.9	5.0	1.2	5.2	8.0

c =

1	2	3	4	5	6	7	8	9
3.1	2.4	0	0	0	3.2	0	4.4	0

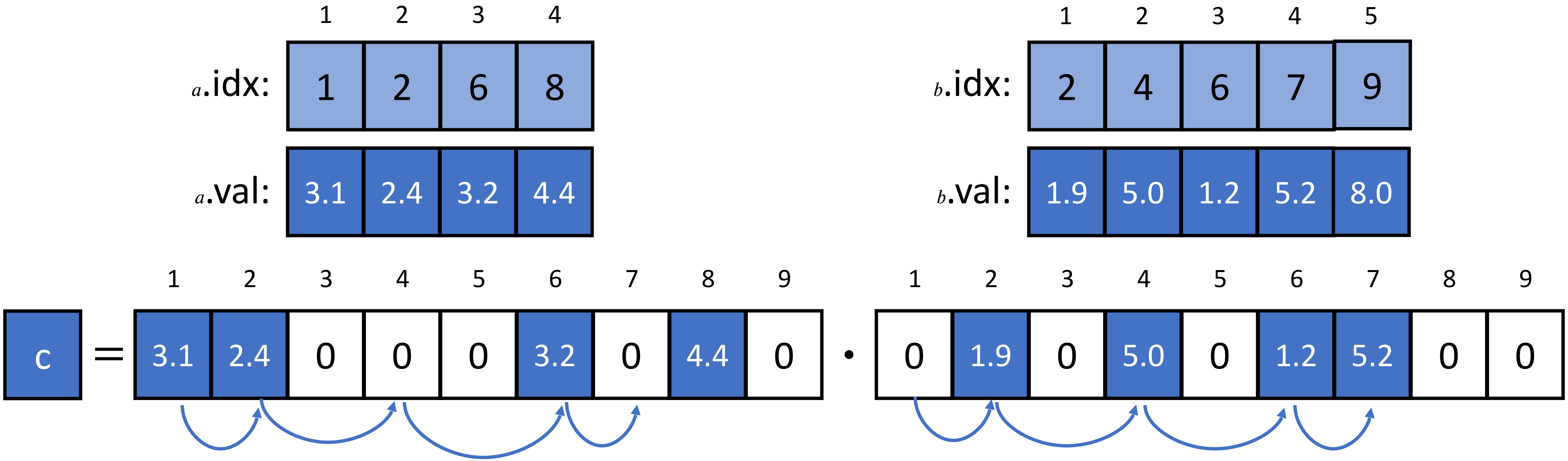
·

1	2	3	4	5	6	7	8	9
0	1.9	0	5.0	0	1.2	5.2	0	0

a is sparse

b is sparse

Merging Multiple Sparse Requires Coordination



	1	2	3	4
a.idx:	1	2	6	8
a.val:	3.1	2.4	3.2	4.4

	1	2	3	4	5
b.idx:	2	4	6	7	9
b.val:	1.9	5.0	1.2	5.2	8.0

c =

1	2	3	4	5	6	7	8	9
3.1	2.4	0	0	0	3.2	0	4.4	0

·

1	2	3	4	5	6	7	8	9
0	1.9	0	5.0	0	1.2	5.2	0	0

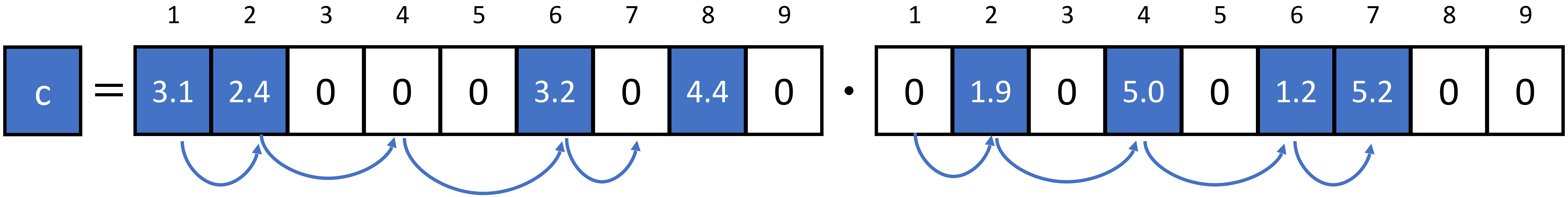
a is sparse

b is sparse

Merging Multiple Sparse Requires Coordination

	1	2	3	4
a.idx:	1	2	6	8
a.val:	3.1	2.4	3.2	4.4

	1	2	3	4	5
b.idx:	2	4	6	7	9
b.val:	1.9	5.0	1.2	5.2	8.0



	1	2	3	4	5	6	7	8	9
c =	3.1	2.4	0	0	0	3.2	0	4.4	0

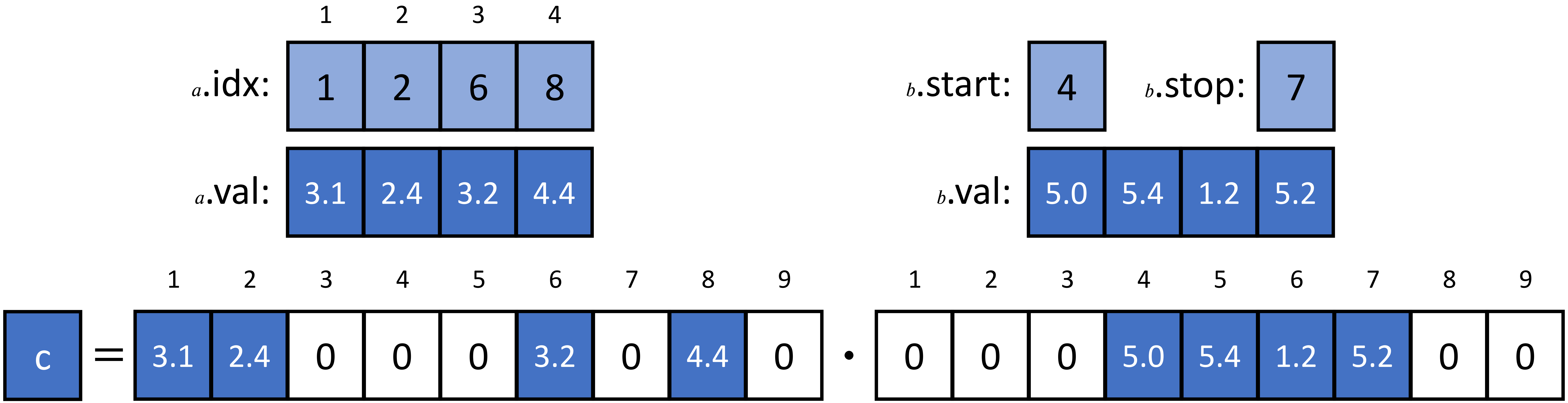
	1	2	3	4	5	6	7	8	9
·	0	1.9	0	5.0	0	1.2	5.2	0	0

```
while p < len(a.idx) && q < len(b.idx)
   i_a = a.idx[p]
   i_b = b.idx[q]
   i = min(i_a, i_b)
   if i == i_a && i == i_b
      c += a.val[p] * b.val[q]
   p += i_a == i
   q += i_b == i
```

a is sparse

b is sparse

Some Sparse Inputs Have Structure



	1	2	3	4
a.idx:	1	2	6	8
a.val:	3.1	2.4	3.2	4.4

b.start: 4 b.stop: 7

b.val:	5.0	5.4	1.2	5.2

c =

1	2	3	4	5	6	7	8	9
3.1	2.4	0	0	0	3.2	0	4.4	0

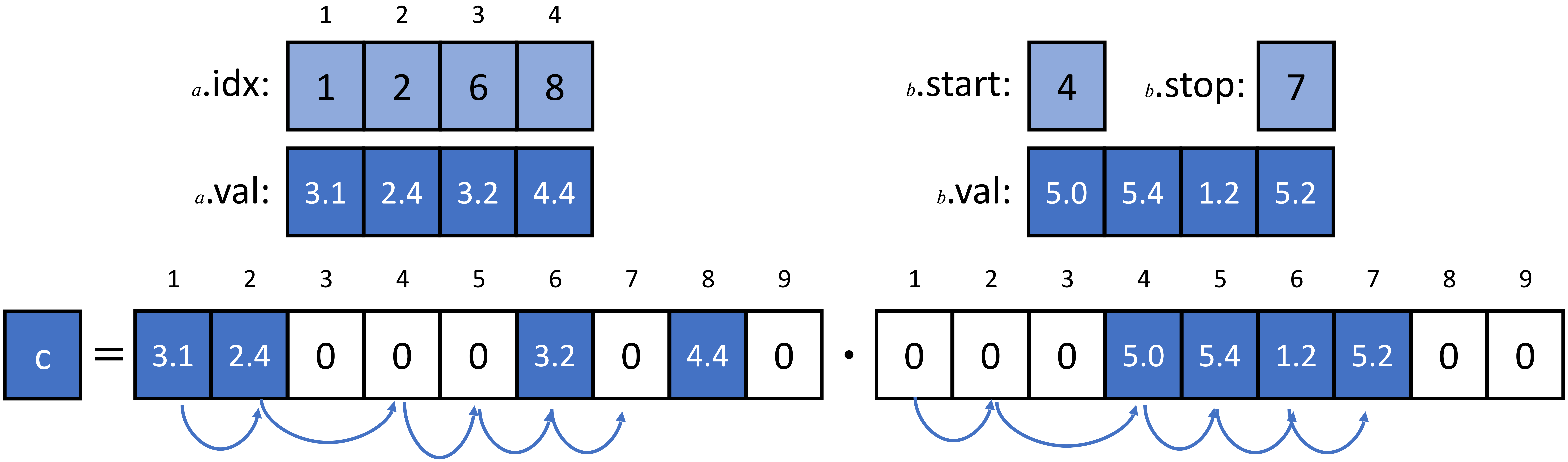
•

1	2	3	4	5	6	7	8	9
0	0	0	5.0	5.4	1.2	5.2	0	0

a is sparse

b is blocked

Some Sparse Inputs Have Structure



	1	2	3	4
a.idx:	1	2	6	8
a.val:	3.1	2.4	3.2	4.4

b.start:	b.stop:
4	7

b.val:	5.0	5.4	1.2	5.2

c =

1	2	3	4	5	6	7	8	9
3.1	2.4	0	0	0	3.2	0	4.4	0

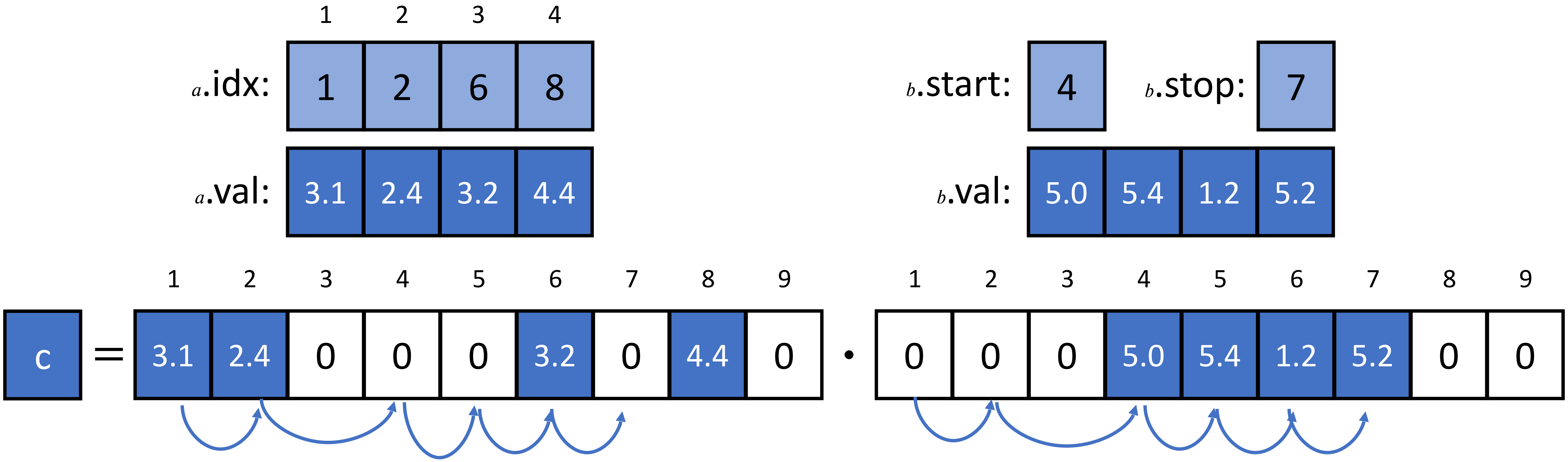
•

1	2	3	4	5	6	7	8	9
0	0	0	5.0	5.4	1.2	5.2	0	0

a is sparse

b is blocked

Some Sparse Inputs Have Structure



	1	2	3	4
a.idx:	1	2	6	8
a.val:	3.1	2.4	3.2	4.4

b.start: 4 *b*.stop: 7

b.val:	5.0	5.4	1.2	5.2

c =

1	2	3	4	5	6	7	8	9
3.1	2.4	0	0	0	3.2	0	4.4	0

·

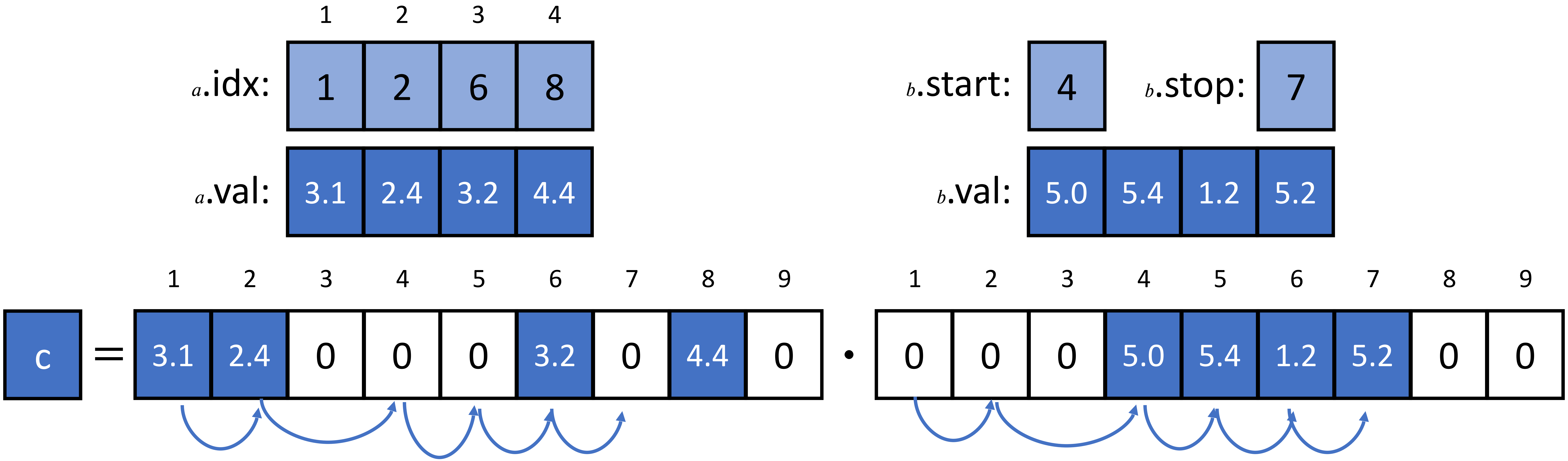
1	2	3	4	5	6	7	8	9
0	0	0	5.0	5.4	1.2	5.2	0	0

```
while p < len(a.idx) && q < len(b.idx)
    i_a = a.idx[p]
    i_b = b.idx[q]
    i = min(i_a, i_b)
    if i == i_a && i == i_b
        c += a.val[p] * b.val[q]
    p += i_a == i
    q += i_b == i
```

a is sparse

b is blocked

Some Sparse Inputs Have Structure



	1	2	3	4
a.idx:	1	2	6	8
a.val:	3.1	2.4	3.2	4.4

b.start: 4 b.stop: 7

b.val:	5.0	5.4	1.2	5.2

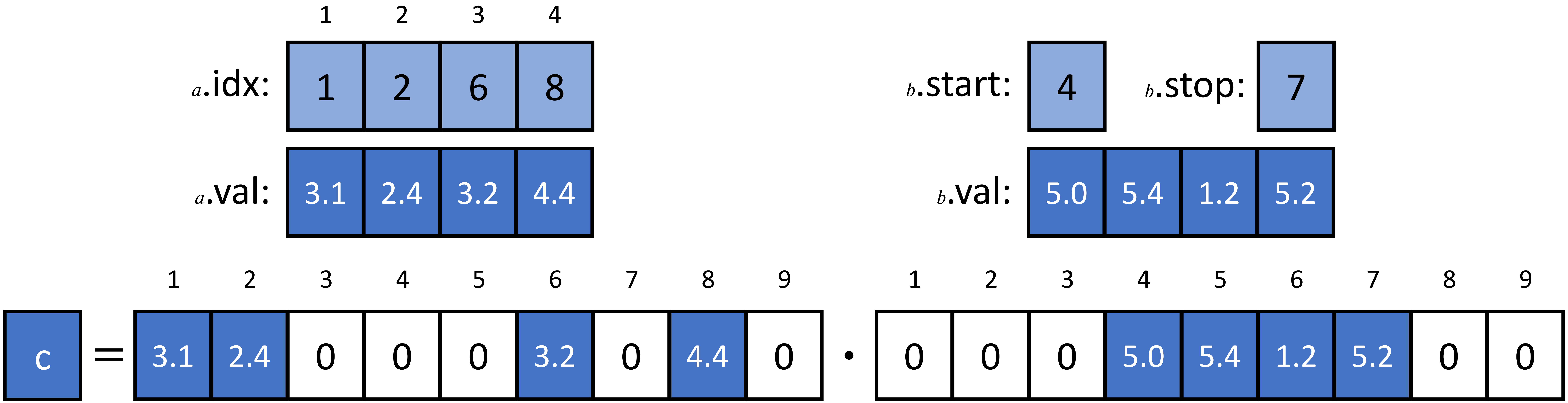
	1	2	3	4	5	6	7	8	9
c =	3.1	2.4	0	0	0	3.2	0	4.4	0
·	0	0	0	5.0	5.4	1.2	5.2	0	0

```
while p < len(a.idx) && q < b.stop – b.start
   i_a = a.idx[p]
   i_b = b.start + q
   i = min(i_a, i_b)
   if i == i_a && i == i_b
      c += a.val[p] * b.val[q]
   p += i_a == i
   q += i_b == i
```

a is sparse

b is blocked

We Can Use Structure



	1	2	3	4
a.idx:	1	2	6	8
a.val:	3.1	2.4	3.2	4.4

b.start: 4 b.stop: 7

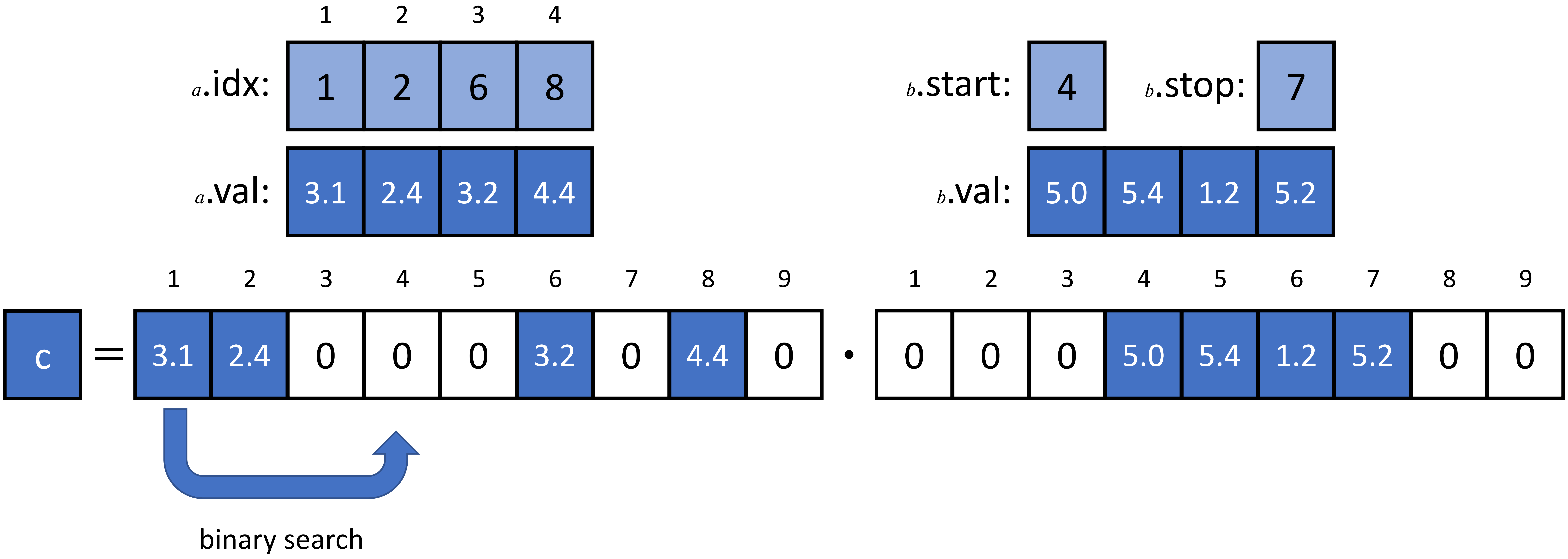
b.val:	5.0	5.4	1.2	5.2

		1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7	8	9
c	=	3.1	2.4	0	0	0	3.2	0	4.4	0	•	0	0	0	5.0	5.4	1.2	5.2	0	0

a is sparse

b is blocked

We Can Use Structure



	1	2	3	4
a.idx:	1	2	6	8
a.val:	3.1	2.4	3.2	4.4

b.start: 4 b.stop: 7

b.val:	5.0	5.4	1.2	5.2

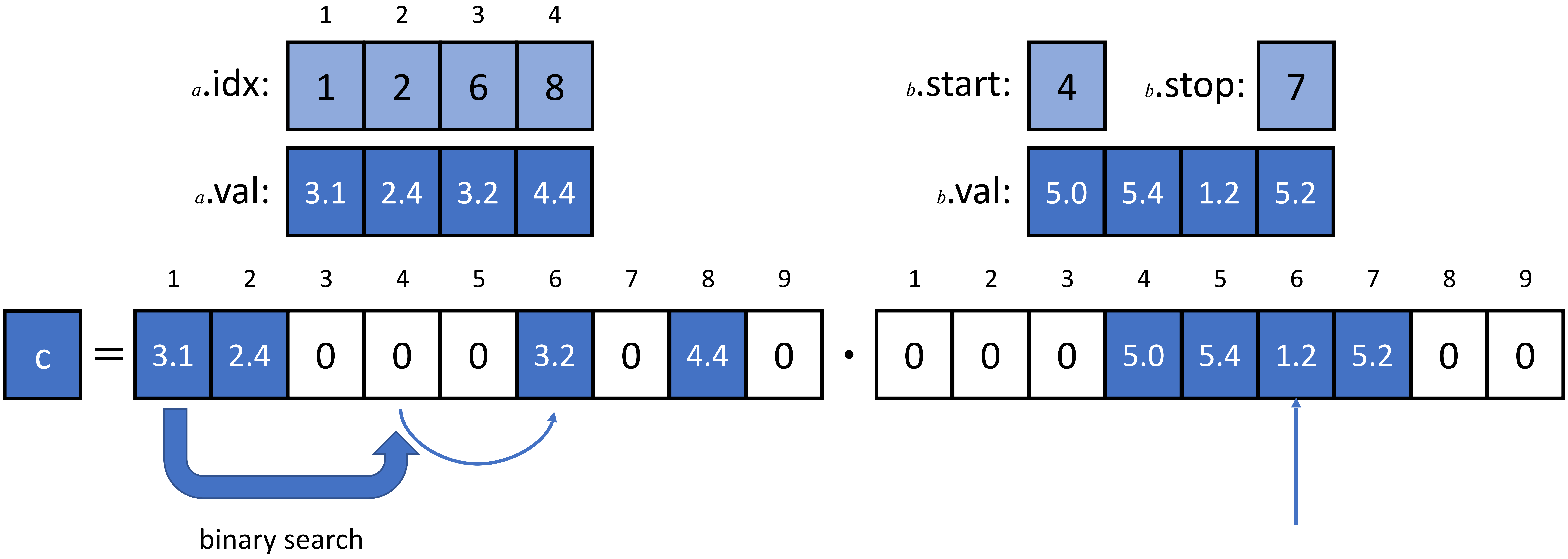
		1	2	3	4	5	6	7	8	9
c	=	3.1	2.4	0	0	0	3.2	0	4.4	0
	·	0	0	0	5.0	5.4	1.2	5.2	0	0

binary search

a is sparse

b is blocked

We Can Use Structure



	1	2	3	4
a.idx:	1	2	6	8
a.val:	3.1	2.4	3.2	4.4

b.start: 4 b.stop: 7

b.val:	5.0	5.4	1.2	5.2

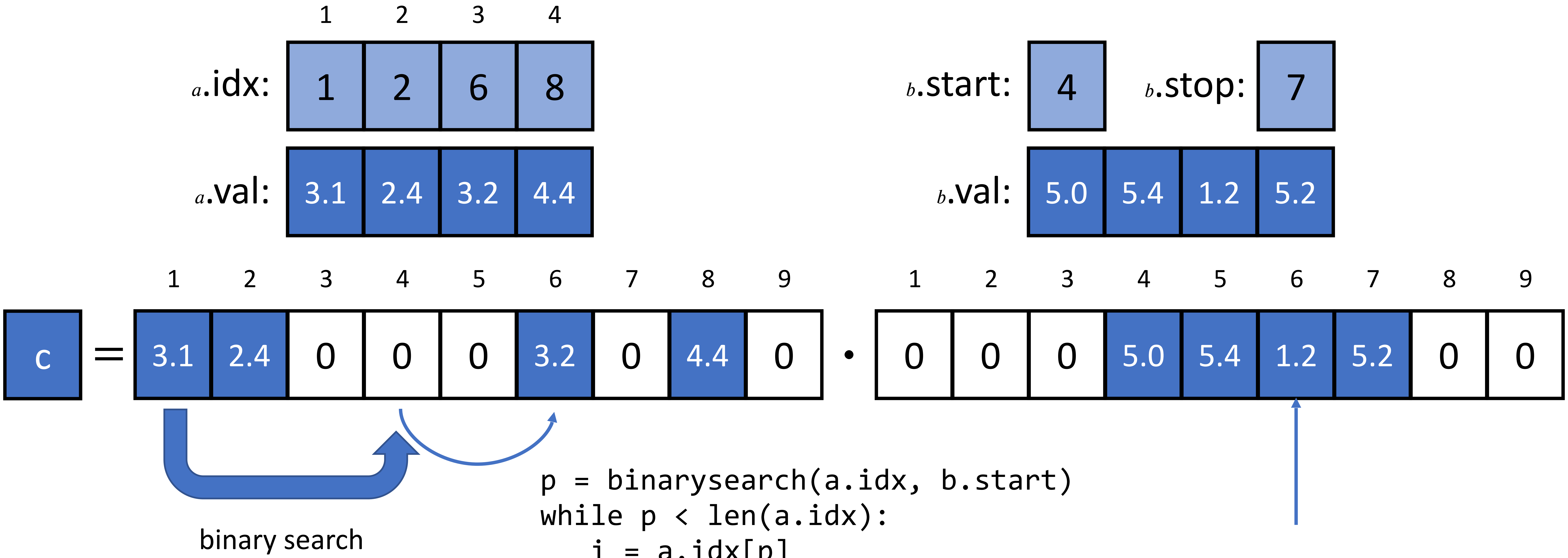
$$c = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3.1 & 2.4 & 0 & 0 & 0 & 3.2 & 0 & 4.4 & 0 \end{array} \cdot \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 0 & 0 & 0 & 5.0 & 5.4 & 1.2 & 5.2 & 0 & 0 \end{array}$$

binary search

a is sparse

b is blocked

We Can Use Structure



	1	2	3	4
a.idx:	1	2	6	8
a.val:	3.1	2.4	3.2	4.4

b.start:	4	*b*.stop:	7

b.val:	5.0	5.4	1.2	5.2

	1	2	3	4	5	6	7	8	9
c =	3.1	2.4	0	0	0	3.2	0	4.4	0
·	0	0	0	5.0	5.4	1.2	5.2	0	0

```
p = binarysearch(a.idx, b.start)
while p < len(a.idx):
    i = a.idx[p]
    if i > b.stop:
        break
    c += a.val[p] * b.val[i – b.start]
    p += 1
```

a is sparse

b is blocked

Algorithms For All Combinations?

	Dense	Sparse
Dense	For Loop	Gather
Sparse	Gather	Merge

Algorithms For All Combinations?

	Dense	Sparse	Blocked
Dense	For Loop	Gather	?
Sparse	Gather	Merge	?
Block	?	?	?

Algorithms For All Combinations?

	Dense	Sparse	Blocked	Ragged	Run Length	Symmetric
Dense	For Loop	Gather	?	?	?	?
Sparse	Gather	Merge	?	?	?	?
Block	?	?	?	?	?	?
Ragged	?	?	?	?	?	?
Run Length	?	?	?	?	?	?
Symmetric	?	?	?	?	?	?

Algorithms For All Combinations?

	Dense	Sparse	Blocked	Ragged	Run Length	Symmetric
Dense	For Loop	Gather	?	?	?	?
Sparse	Gather	Merge	?	?	?	?
Block	?	?	?	?	?	?
Ragged	?	?	?	?	?	?
Run Length	?	?	?	?	?	?
Symmetric	?	?	?	?	?	?

…

⋮

⋱

Algorithms For All Combinations?

TACO

	Dense	Sparse	Blocked	Ragged	Run Length	Symmetric
Dense	For Loop	Gather	?	?	?	?
Sparse	Gather	Merge	?	?	?	?
Block	?	?	?	?	?	?
Ragged	?	?	?	?	?	?
Run Length	?	?	?	?	?	?
Symmetric	?	?	?	?	?	?

…

⋮

⋱

F. Kjolstad, S. Kamil, S. Chou, D. Lugato, and S. Amarasinghe, “The Tensor Algebra Compiler,”.

Algorithms For All Combinations?

TACO

Halide

	Dense	Sparse	Blocked	Ragged	Run Length	Symmetric
Dense	For Loop	Gather	?	?	?	?
Sparse	Gather	Merge	?	?	?	?
Block	?	?	?	?	?	?
Ragged	?	?	?	?	?	?
Run Length	?	?	?	?	?	?
Symmetric	?	?	?	?	?	?

…

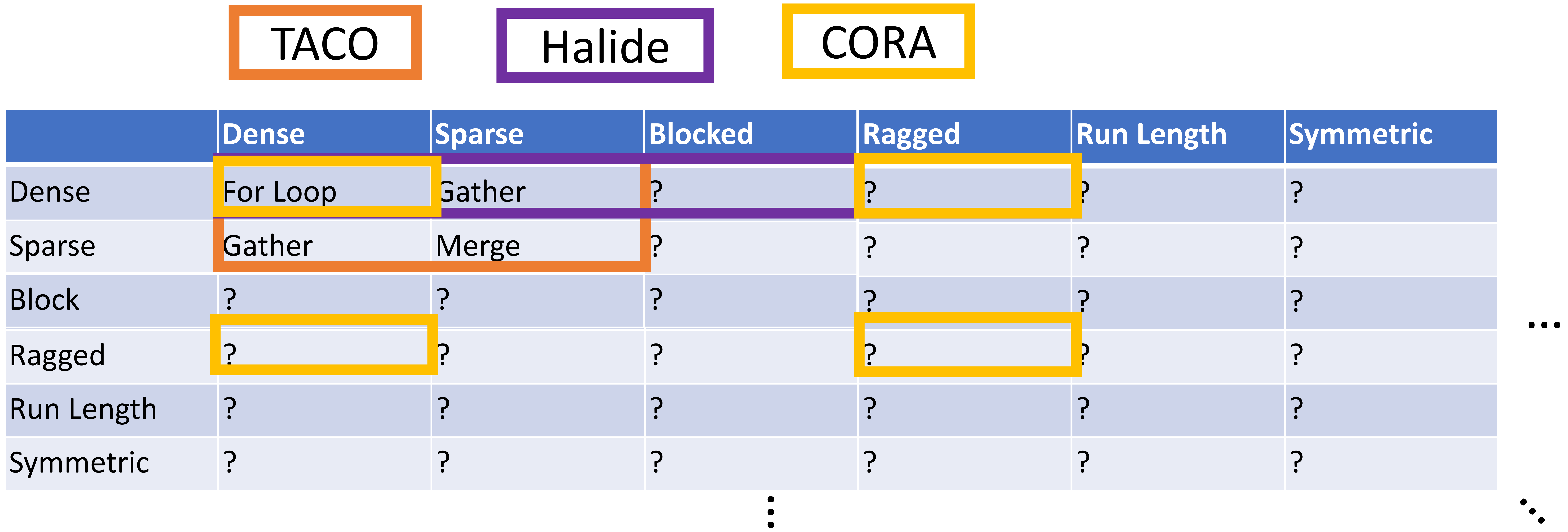
⋮

⋱

F. Kjolstad, S. Kamil, S. Chou, D. Lugato, and S. Amarasinghe, “The Tensor Algebra Compiler,”.

J. Ragan-Kelley, A. Adams, S. Paris, M. Levoy, S. Amarasinghe, and F. Durand, “Decoupling algorithms from schedules for easy optimization of image processing pipelines,”

Algorithms For All Combinations?



TACO

Halide

CORA

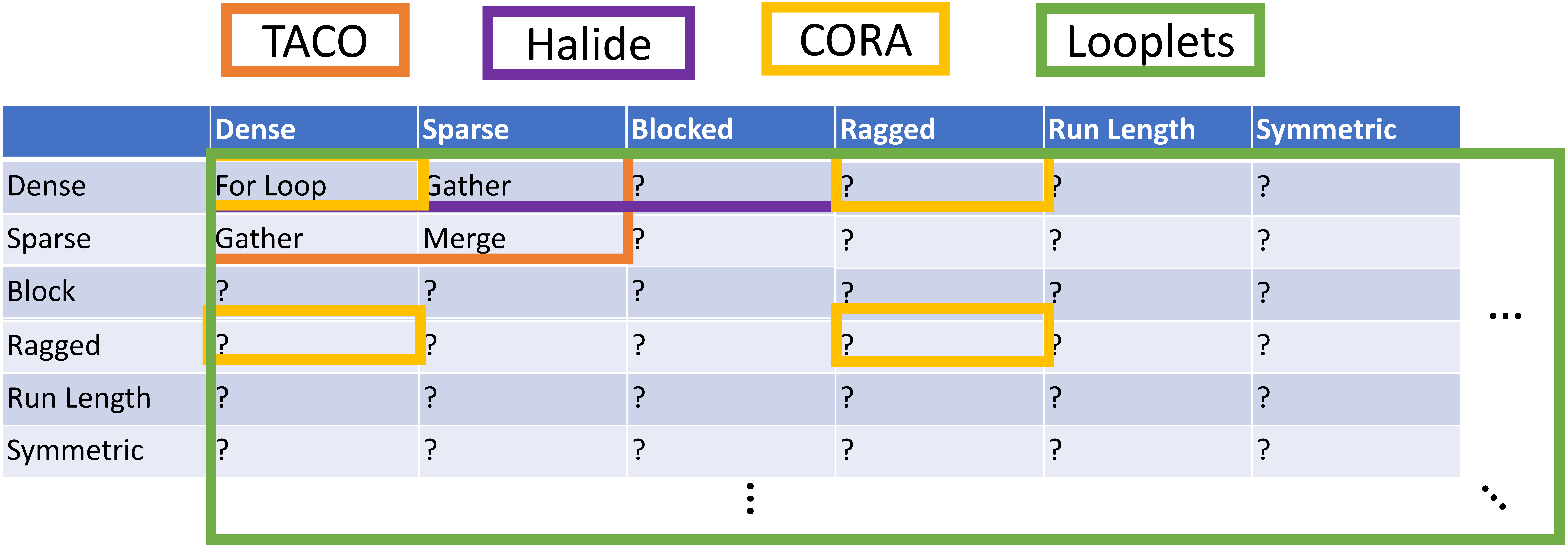
	Dense	Sparse	Blocked	Ragged	Run Length	Symmetric
Dense	For Loop	Gather	?	?	?	?
Sparse	Gather	Merge	?	?	?	?
Block	?	?	?	?	?	?
Ragged	?	?	?	?	?	?
Run Length	?	?	?	?	?	?
Symmetric	?	?	?	?	?	?

F. Kjolstad, S. Kamil, S. Chou, D. Lugato, and S. Amarasinghe, "The Tensor Algebra Compiler,".

J. Ragan-Kelley, A. Adams, S. Paris, M. Levoy, S. Amarasinghe, and F. Durand, "Decoupling algorithms from schedules for easy optimization of image processing pipelines,"

P. Fegade, T. Chen, P. B. Gibbons, and T. C. Mowry, "The CoRa Tensor Compiler: Compilation for Ragged Tensors with Minimal Padding"

Algorithms For All Combinations?



	Dense	Sparse	Blocked	Ragged	Run Length	Symmetric
Dense	For Loop	Gather	?	?	?	?
Sparse	Gather	Merge	?	?	?	?
Block	?	?	?	?	?	?
Ragged	?	?	?	?	?	?
Run Length	?	?	?	?	?	?
Symmetric	?	?	?	?	?	?

…

⋮

⋱

F. Kjolstad, S. Kamil, S. Chou, D. Lugato, and S. Amarasinghe, “The Tensor Algebra Compiler,”.

J. Ragan-Kelley, A. Adams, S. Paris, M. Levoy, S. Amarasinghe, and F. Durand, “Decoupling algorithms from schedules for easy optimization of image processing pipelines,”

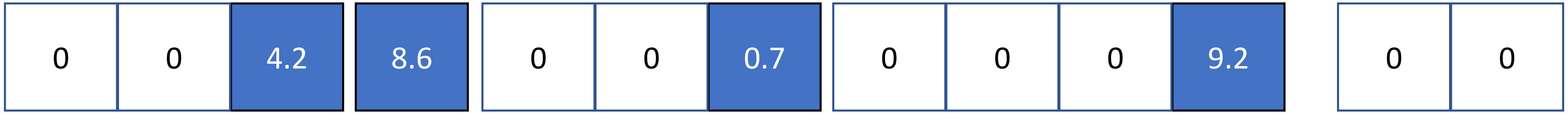
P. Fegade, T. Chen, P. B. Gibbons, and T. C. Mowry, “The CoRa Tensor Compiler: Compilation for Ragged Tensors with Minimal Padding”

Looplet Language

- A general language to iterate over structured data
  - Iterating over complex structured data expressed using a language of a few primitives
    - Lookup
    - Run
    - Spike
    - Pipeline
    - Stepper
    - Jumper
    - Shift
    - Switch

Looplet Language

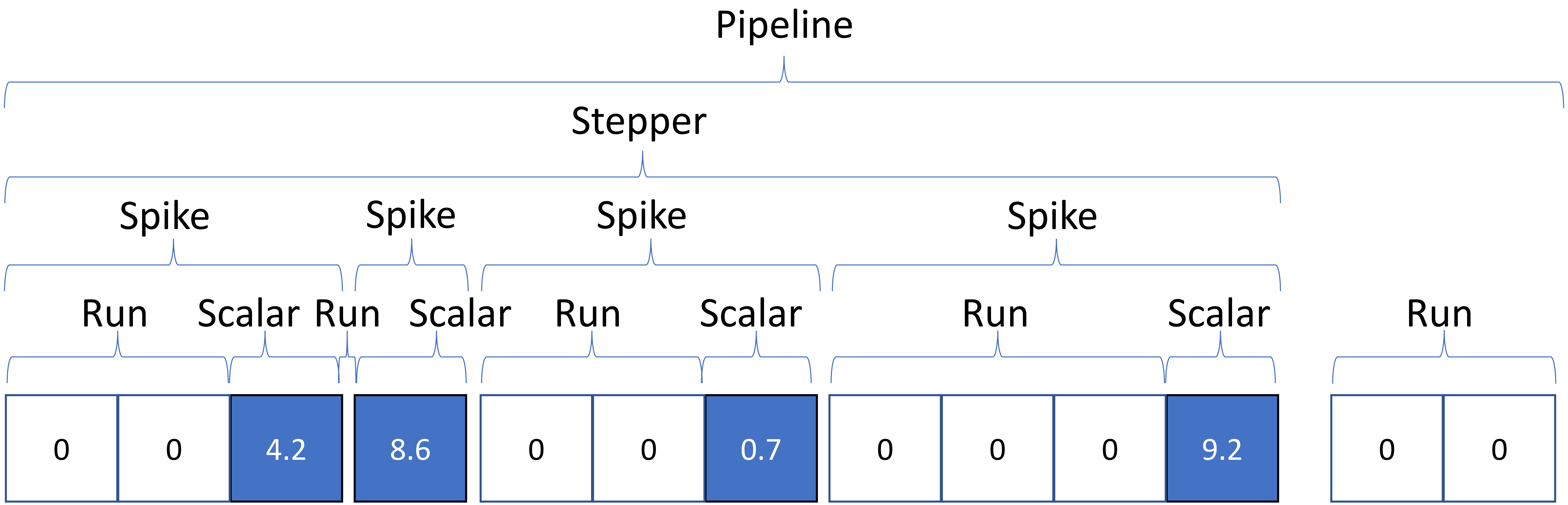
- A general language to iterate over structured data
  - Iterating over complex structured data expressed using a language of a few primitives
    - Lookup
    - Run
    - Spike
    - Pipeline
    - Stepper
    - Jumper
    - Shift
    - Switch



0	0	4.2	8.6	0	0	0.7	0	0	0	9.2	0	0

Looplet Language

- A general language to iterate over structured data
  - Iterating over complex structured data expressed using a language of a few primitives
    - Lookup
    - Run
    - Spike
    - Pipeline
    - Stepper
    - Jumper
    - Shift
    - Switch

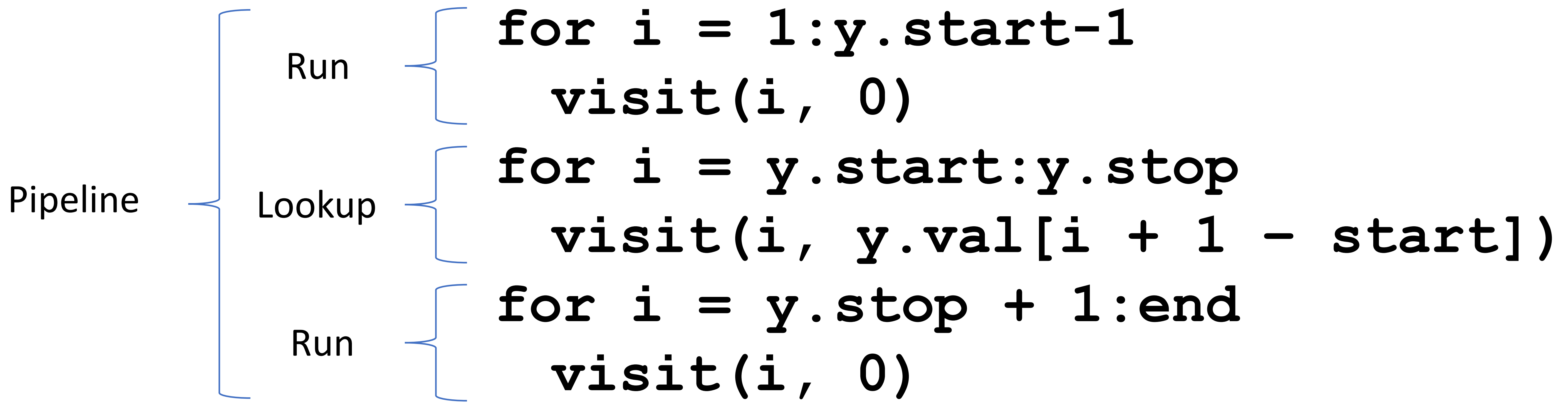


0	0	4.2	8.6	0	0	0.7	0	0	0	9.2	0	0		

Looplet Language

- A general language to iterate over structured data
  - Iterating over complex structured data expressed using a language of a few primitives
    - Lookup
    - Run
    - Spike
    - Pipeline
    - Stepper
    - Jumper
    - Shift
    - Switch

- Code generation from the iteration protocols is simple and mechanical



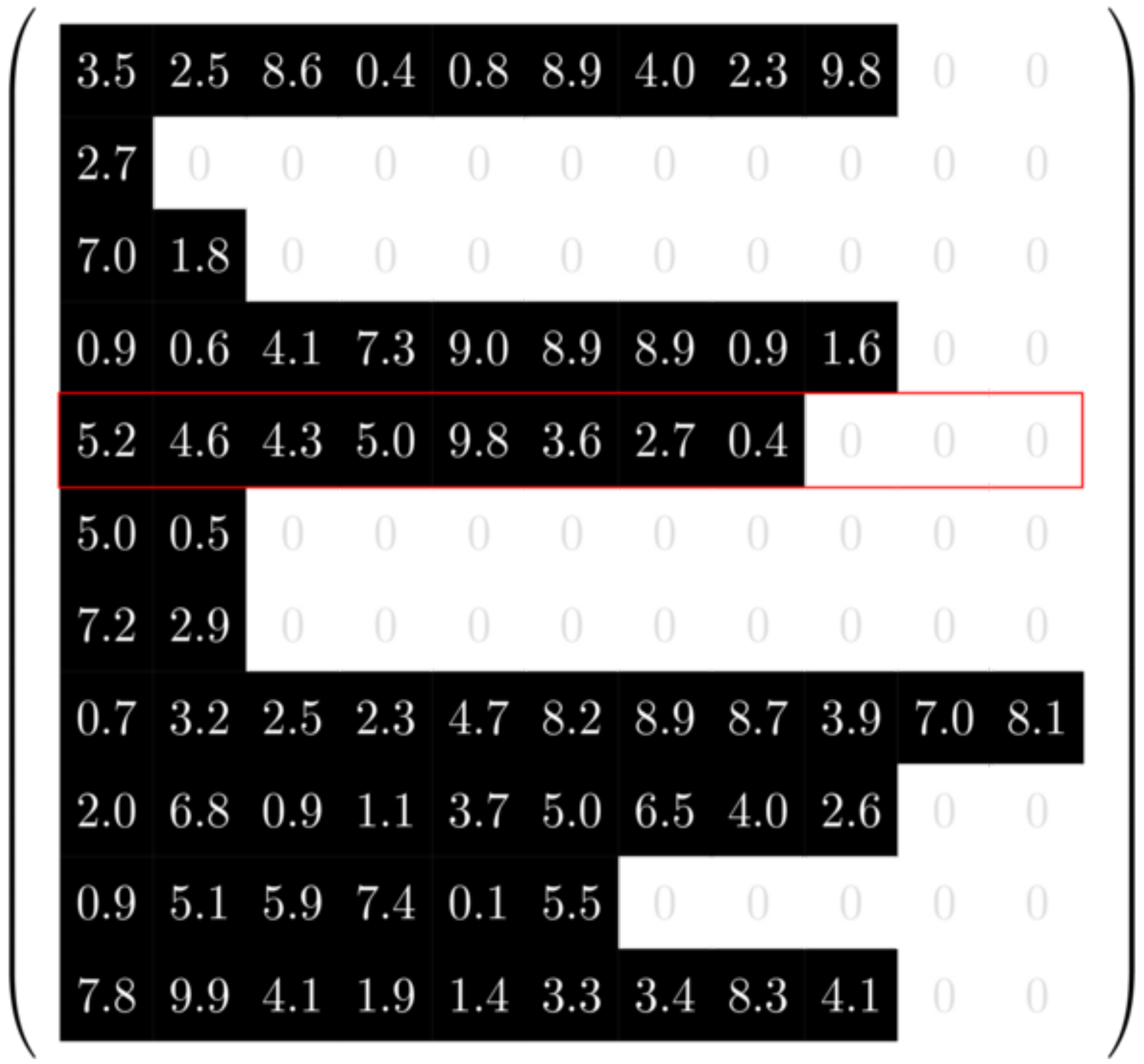
```
for i = 1:y.start-1
  visit(i, 0)
for i = y.start:y.stop
  visit(i, y.val[i + 1 – start])
for i = y.stop + 1:end
  visit(i, 0)
```

Looplet Language

- A general language to iterate over structured data
  - Iterating over complex structured data expressed using a language of a few primitives
    - Lookup
    - Run
    - Spike
    - Pipeline
    - Stepper
    - Jumper
    - Shift
    - Switch
- Code generation from the iteration protocols is simple and mechanical
- To coiterate, merge the individual iteration protocols
  - Use rewrite rules to simplify

Looplet Language Supports Many Types Of Structured Data

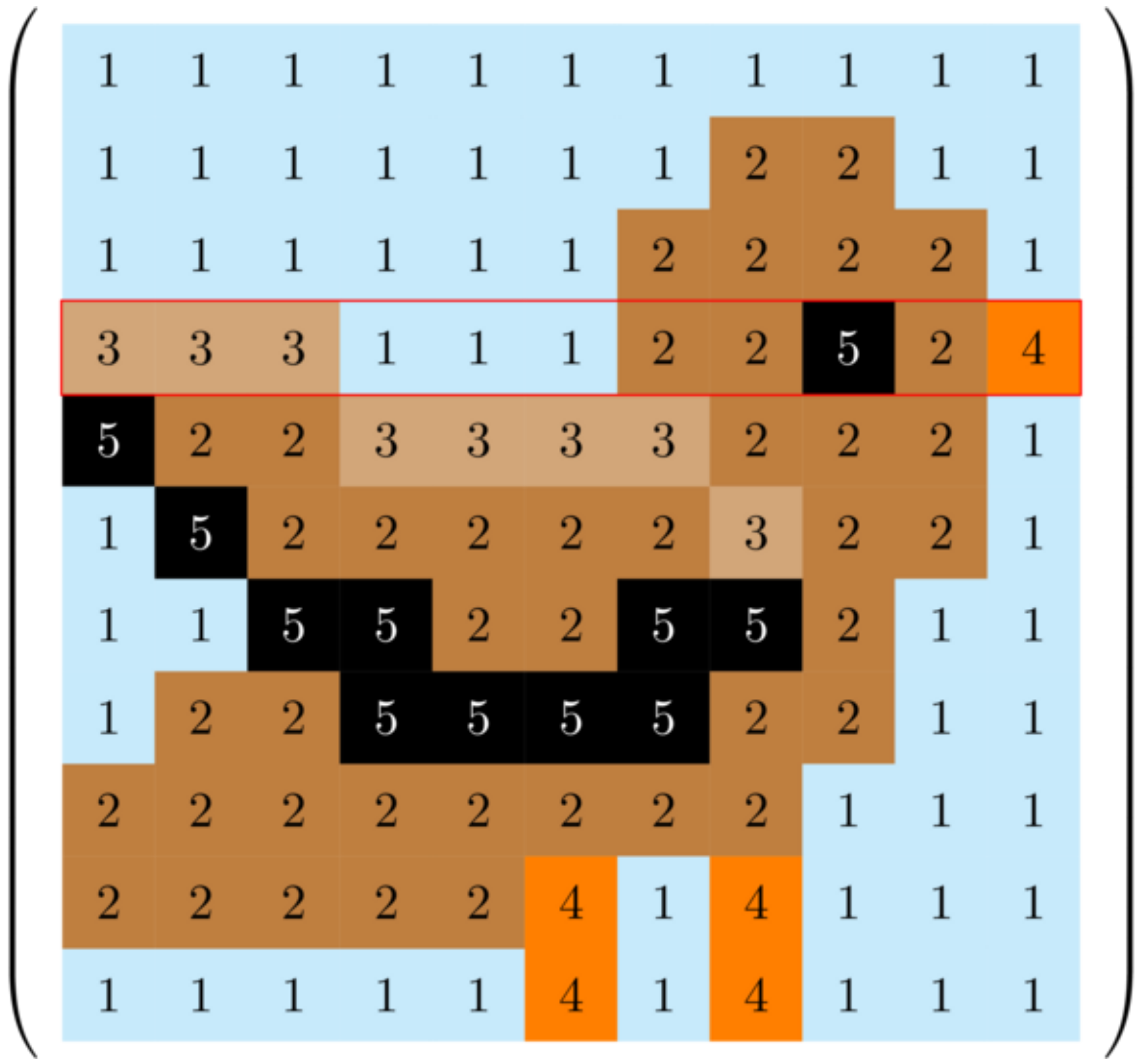
Ragged Matrix



3.5	2.5	8.6	0.4	0.8	8.9	4.0	2.3	9.8	0	0
2.7	0	0	0	0	0	0	0	0	0	0
7.0	1.8	0	0	0	0	0	0	0	0	0
0.9	0.6	4.1	7.3	9.0	8.9	8.9	0.9	1.6	0	0
5.2	4.6	4.3	5.0	9.8	3.6	2.7	0.4	0	0	0
5.0	0.5	0	0	0	0	0	0	0	0	0
7.2	2.9	0	0	0	0	0	0	0	0	0
0.7	3.2	2.5	2.3	4.7	8.2	8.9	8.7	3.9	7.0	8.1
2.0	6.8	0.9	1.1	3.7	5.0	6.5	4.0	2.6	0	0
0.9	5.1	5.9	7.4	0.1	5.5	0	0	0	0	0
7.8	9.9	4.1	1.9	1.4	3.3	3.4	8.3	4.1	0	0

P. Fegade, T. Chen, P. B. Gibbons, and T. C. Mowry, “The CoRa Tensor Compiler: Compilation for Ragged Tensors with Minimal Padding”

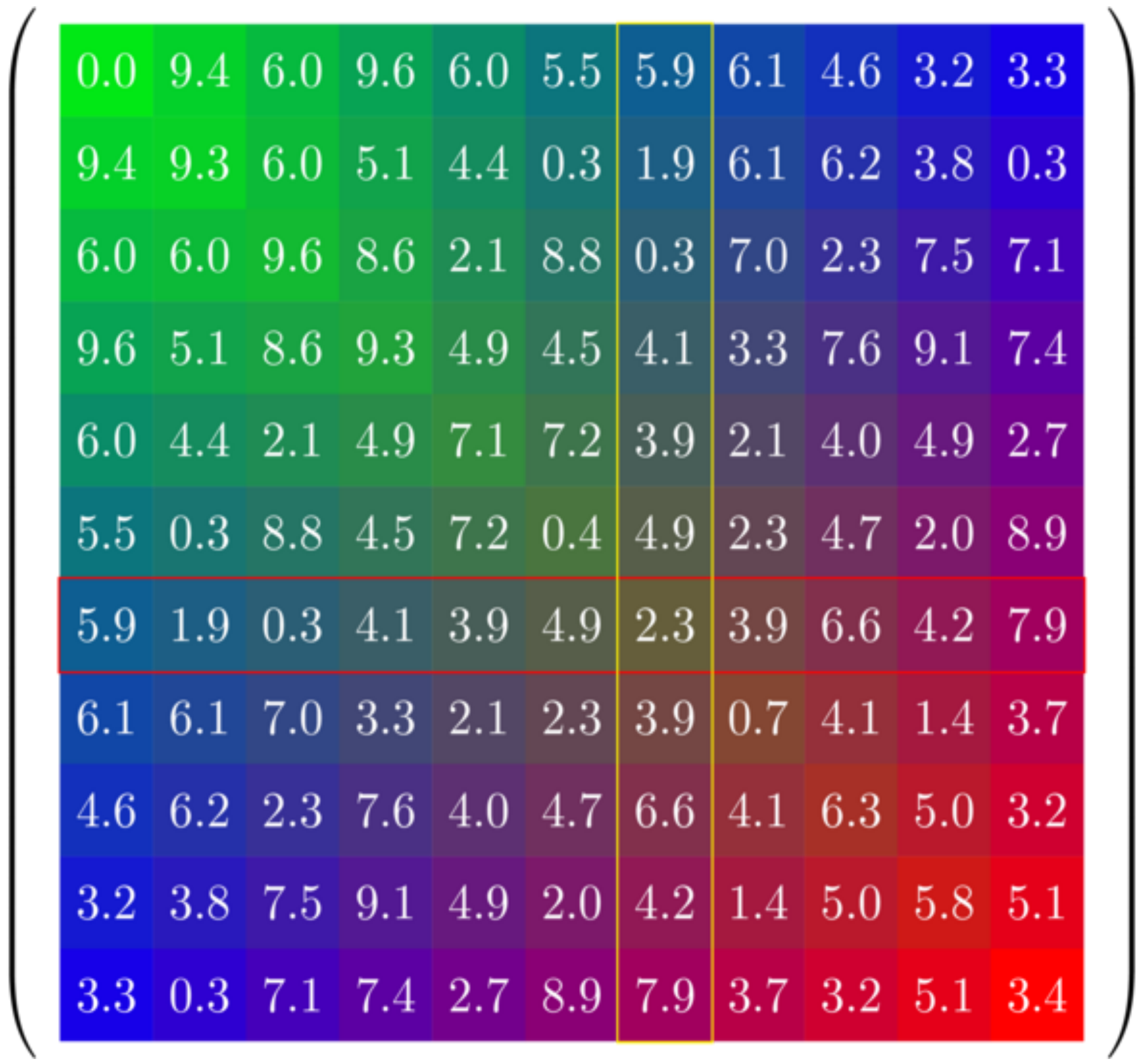
Run Length Matrix



1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	2	2	1	1
1	1	1	1	1	1	2	2	2	2	1
3	3	3	1	1	1	2	2	5	2	4
5	2	2	3	3	3	3	2	2	2	1
1	5	2	2	2	2	2	3	2	2	1
1	1	5	5	2	2	5	5	2	1	1
1	2	2	5	5	5	5	2	2	1	1
2	2	2	2	2	2	2	2	1	1	1
2	2	2	2	2	4	1	4	1	1	1
1	1	1	1	1	4	1	4	1	1	1

D. Donenfeld, S. Chou, and S. Amarasinghe, “Unified Compilation for Lossless Compression and Sparse Computing”

Symmetric Matrix



0.0	9.4	6.0	9.6	6.0	5.5	5.9	6.1	4.6	3.2	3.3
9.4	9.3	6.0	5.1	4.4	0.3	1.9	6.1	6.2	3.8	0.3
6.0	6.0	9.6	8.6	2.1	8.8	0.3	7.0	2.3	7.5	7.1
9.6	5.1	8.6	9.3	4.9	4.5	4.1	3.3	7.6	9.1	7.4
6.0	4.4	2.1	4.9	7.1	7.2	3.9	2.1	4.0	4.9	2.7
5.5	0.3	8.8	4.5	7.2	0.4	4.9	2.3	4.7	2.0	8.9
5.9	1.9	0.3	4.1	3.9	4.9	2.3	3.9	6.6	4.2	7.9
6.1	6.1	7.0	3.3	2.1	2.3	3.9	0.7	4.1	1.4	3.7
4.6	6.2	2.3	7.6	4.0	4.7	6.6	4.1	6.3	5.0	3.2
3.2	3.8	7.5	9.1	4.9	2.0	4.2	1.4	5.0	5.8	5.1
3.3	0.3	7.1	7.4	2.7	8.9	7.9	3.7	3.2	5.1	3.4

J. Shi, S. Chou, F. Kjolstad, and S. Amarasinghe, “An Attempt to Generate Code for Symmetric Tensor Computations”

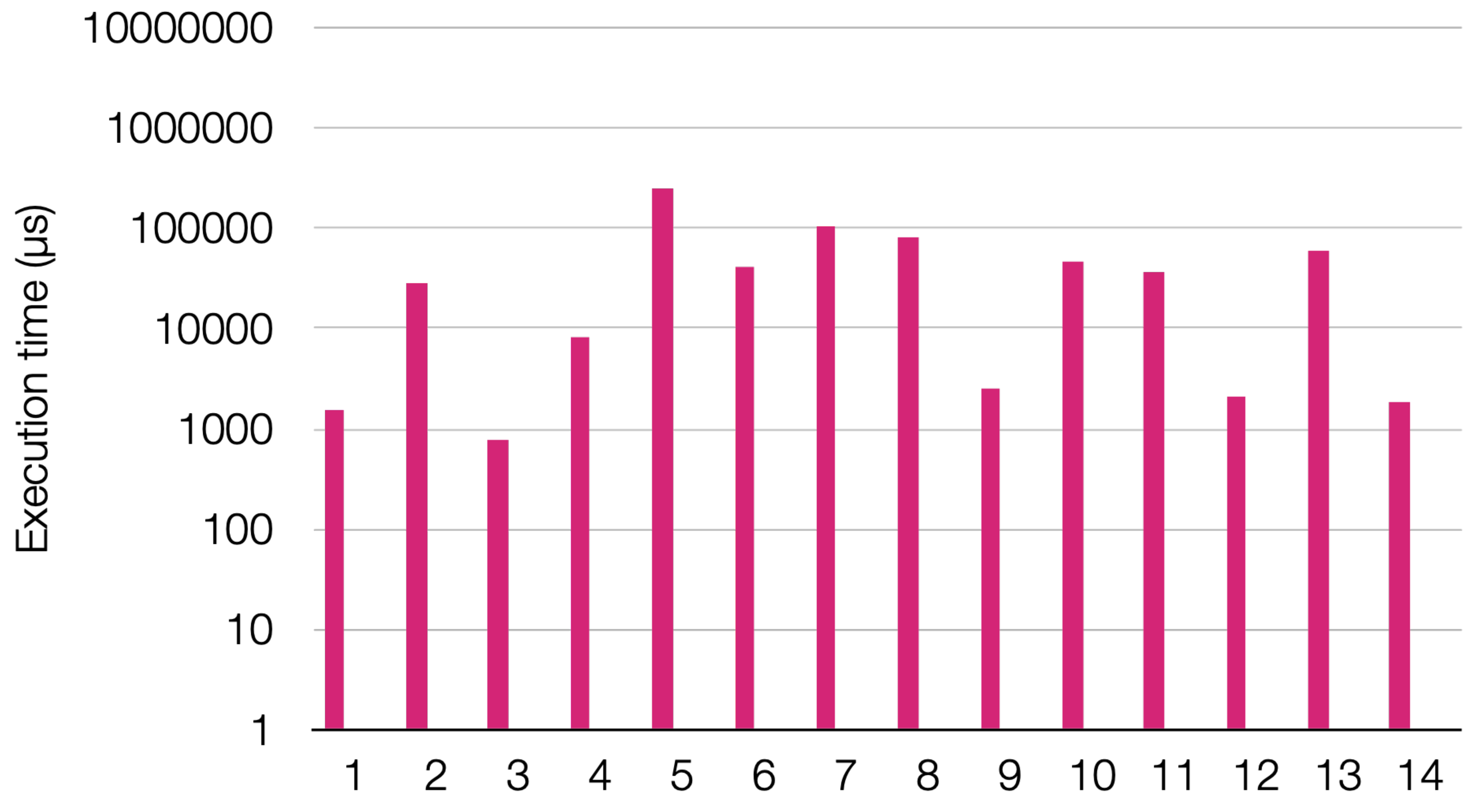
- Unifying what is currently done by multiple compilers
  - Hybrid “have-it-all” formats
  - Expanding into other types of structures

Dynamic Sparse Tensors

- All formats so far (CSR, COO, DIA, ELLPACK, RLE etc.) are static
  - Computing on them can be very fast
  - But…inserting or deleting an element can be (asymptotically) slow

Dynamic Sparse Tensors

- All formats so far (CSR, COO, DIA, ELLPACK, RLE etc.) are static
  - Computing on them can be very fast
  - But…inserting or deleting an element can be (asymptotically) slow

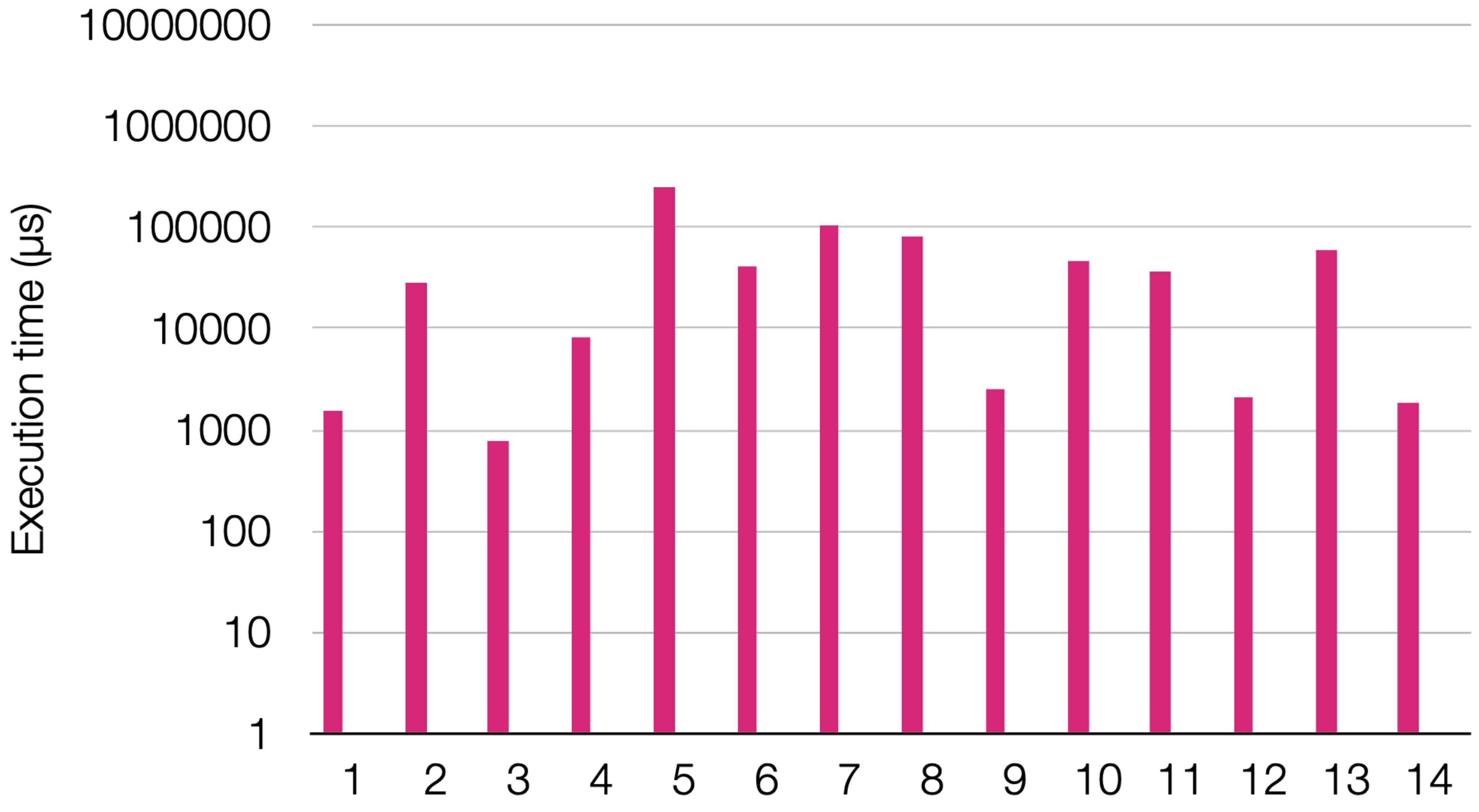


PageRank Computation

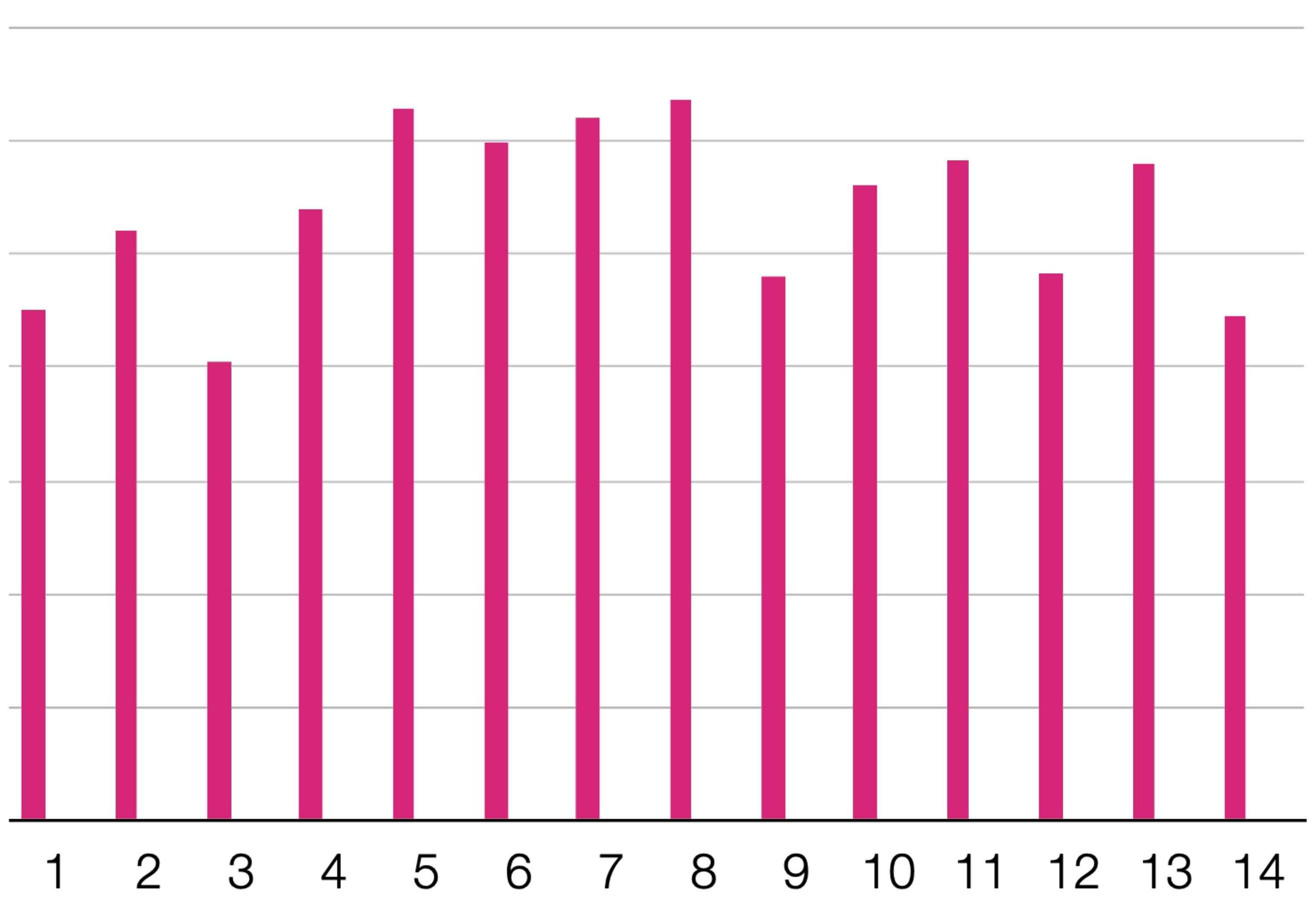
1.63M to 255M Number of Non-Zeros Stored in the CSR Format

Dynamic Sparse Tensors

- All formats so far (CSR, COO, DIA, ELLPACK, RLE etc.) are static
  - Computing on them can be very fast
  - But…inserting or deleting an element can be (asymptotically) slow



PageRank Computation



Insert a Single Element

1.63M to 255M Number of Non-Zeros Stored in the CSR Format

Dynamic Sparse Tensors

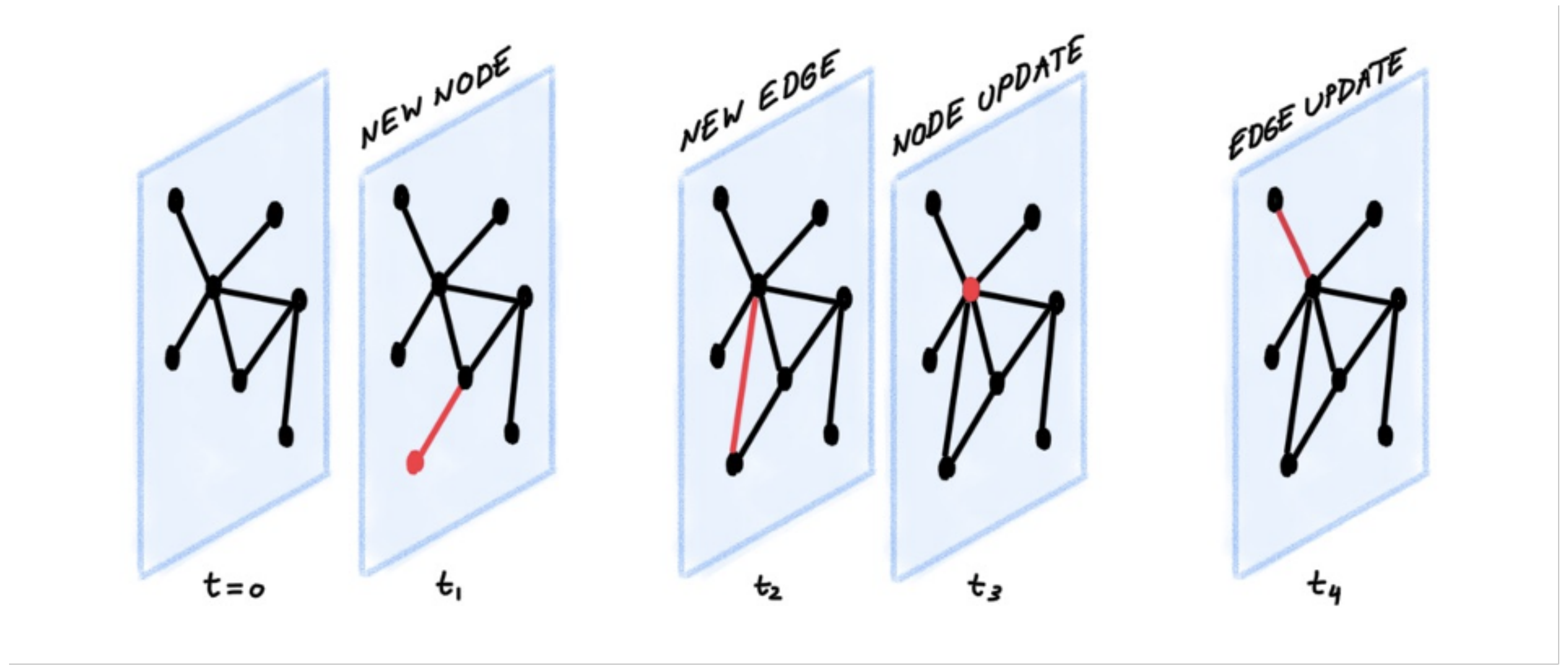
- All formats so far (CSR, COO, DIA, ELLPACK, RLE etc.) are static
  - Computing on them can be very fast
  - But…inserting or deleting an element can be (asymptotically) slow

Dynamic Sparse Tensors

- All formats so far (CSR, COO, DIA, ELLPACK, RLE etc.) are static
  - Computing on them can be very fast
  - But…inserting or deleting an element can be (asymptotically) slow

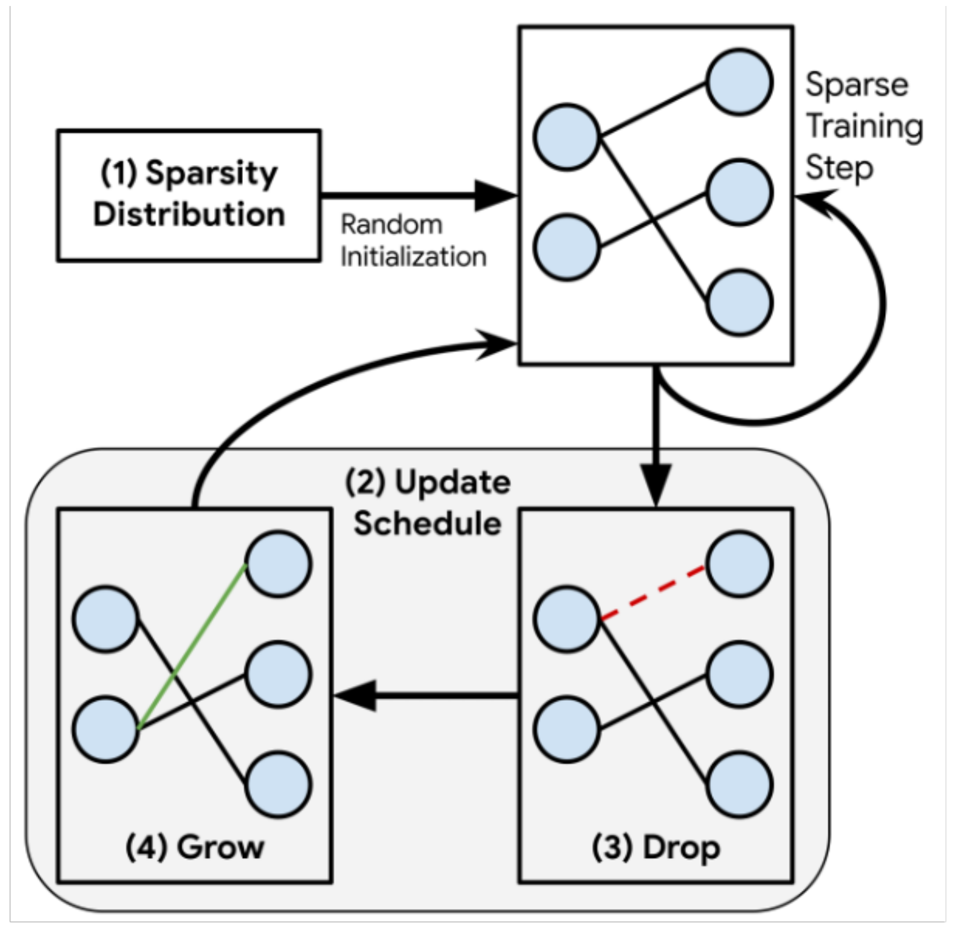
- Many real world Applications are dynamic

Dynamic Graph Processing

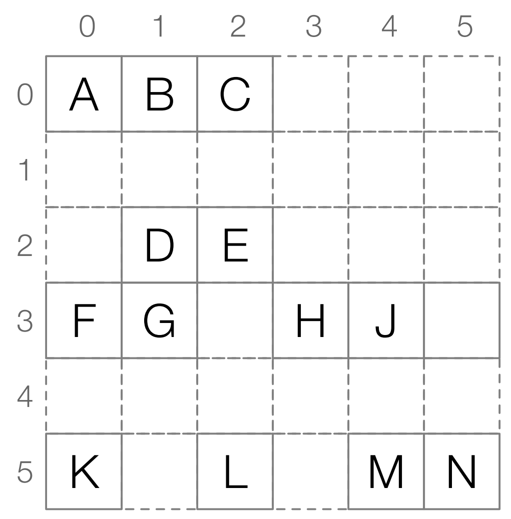


https://blog.twitter.com/engineering/en_us/topics/insights/2021/
temporal-graph-networks

Sparse Neural Network Training



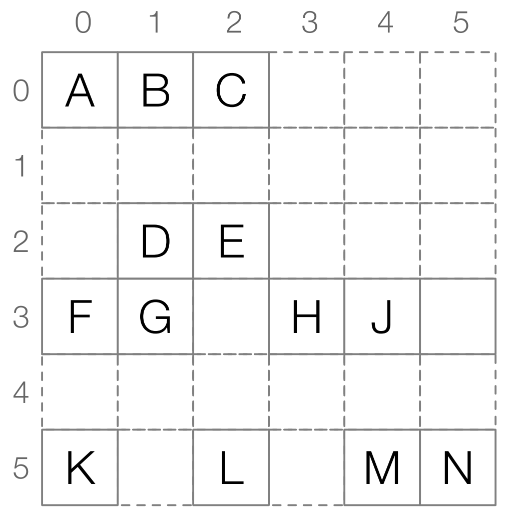
Dynamic Sparse Tensors



	0	1	2	3	4	5
0	A	B	C			
1						
2		D	E			
3	F	G		H	J	
4						
5	K		L		M	N

Dynamic Sparse Tensors

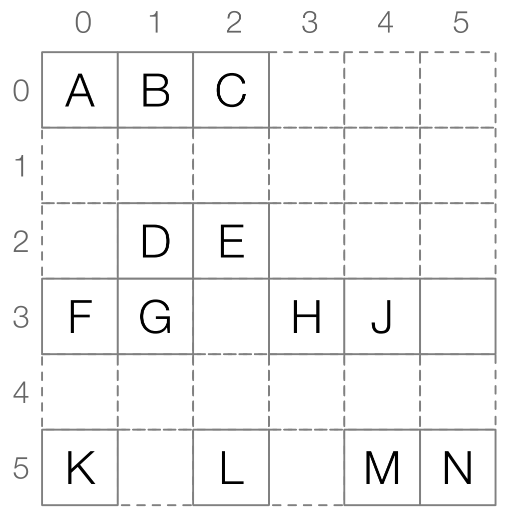
- Need pointer-based, recursive data structures



	0	1	2	3	4	5
0	A	B	C			
1						
2		D	E			
3	F	G		H	J	
4						
5	K		L		M	N

Dynamic Sparse Tensors

- Need pointer-based, recursive data structures
- Novel Node Schema Language
  - Automatically generate the data structures
  - Automatically Generate the code for iteration

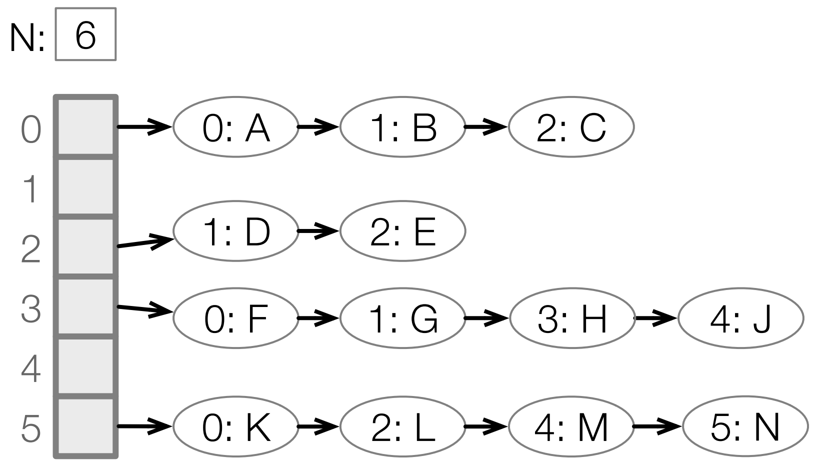


	0	1	2	3	4	5
0	A	B	C			
1						
2		D	E			
3	F	G		H	J	
4						
5	K		L		M	N

Dynamic Sparse Tensors

- Need pointer-based, recursive data structures
- Novel Node Schema Language
  - Automatically generate the data structures
  - Automatically Generate the code for iteration

	0	1	2	3	4	5
0	A	B	C			
1						
2		D	E			
3	F	G		H	J	
4						
5	K		L		M	N

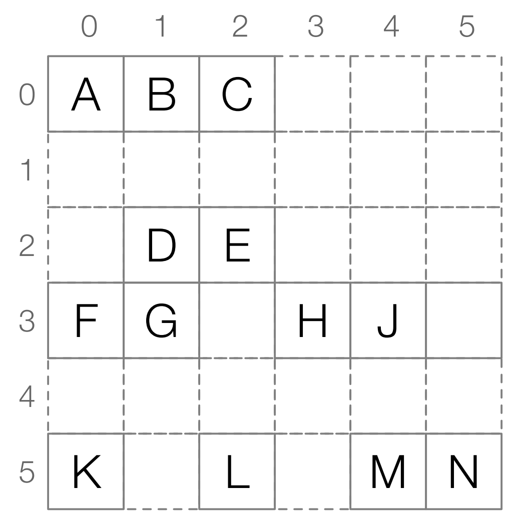


```
def list {
  e : elem nonempty
  n : list
  seq = {e}, n
}

def list_head {
  h : list
}
```

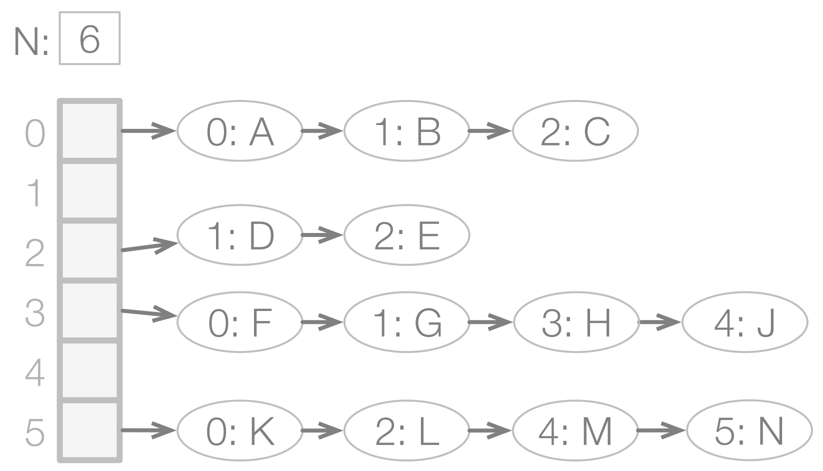
Linked List

Dynamic Sparse Tensors

- Need pointer-based, recursive data structures
- Novel Node Schema Language
  - Automatically generate the data structures
  - Automatically Generate the code for iteration

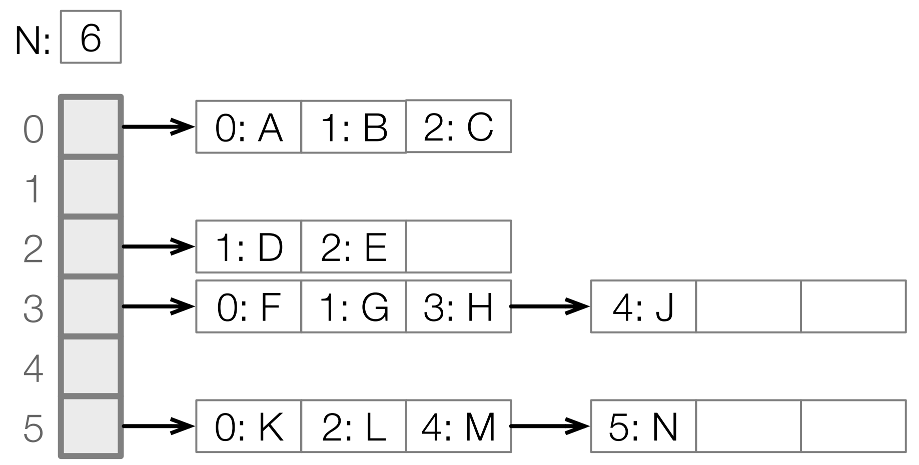


	0	1	2	3	4	5
0	A	B	C			
1						
2		D	E			
3	F	G		H	J	
4						
5	K		L		M	N



```
def list {
  e : elem nonempty
  n : list
  seq = {e}, n
}

def list_head {
  h : list
}
```

Linked List



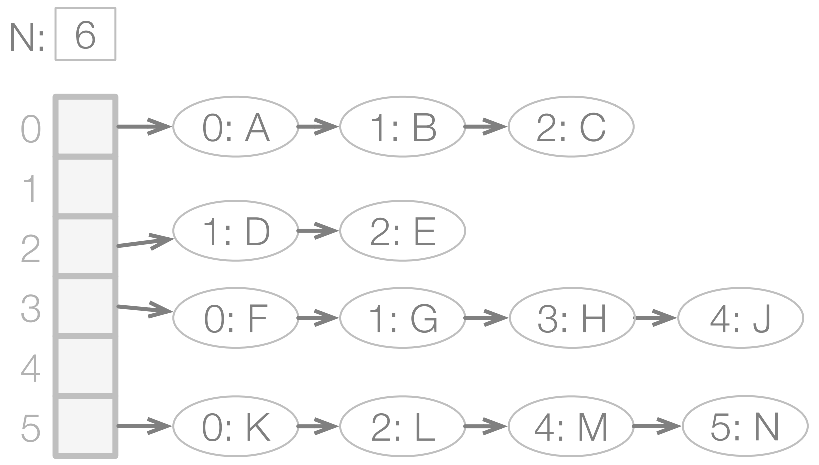
```
def blist {
  e : elem[B] nonempty
  n : blist
  B : size in [0, 3]
  seq = {e}, n
}

def blist_head {
  h : blist
}
```

Block Linked List

Dynamic Sparse Tensors

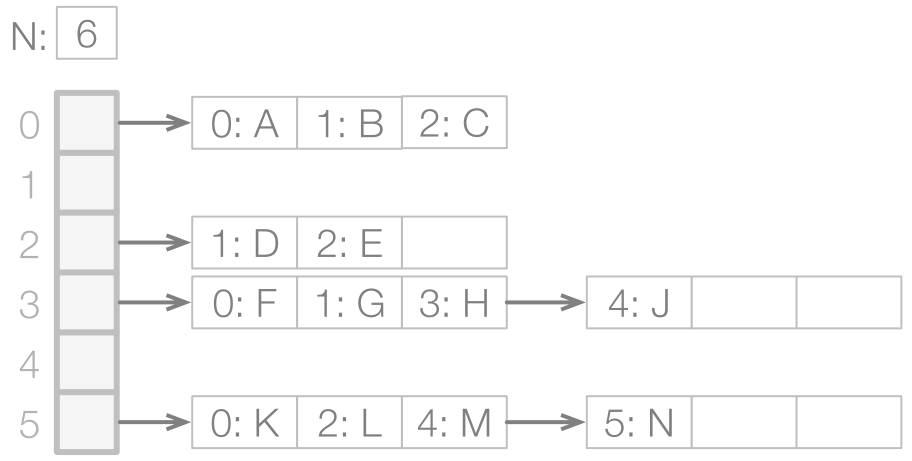
- Need pointer-based, recursive data structures
- Novel Node Schema Language
  - Automatically generate the data structures
  - Automatically Generate the code for iteration

	0	1	2	3	4	5
0	A	B	C			
1						
2		D	E			
3	F	G		H	J	
4						
5	K		L		M	N



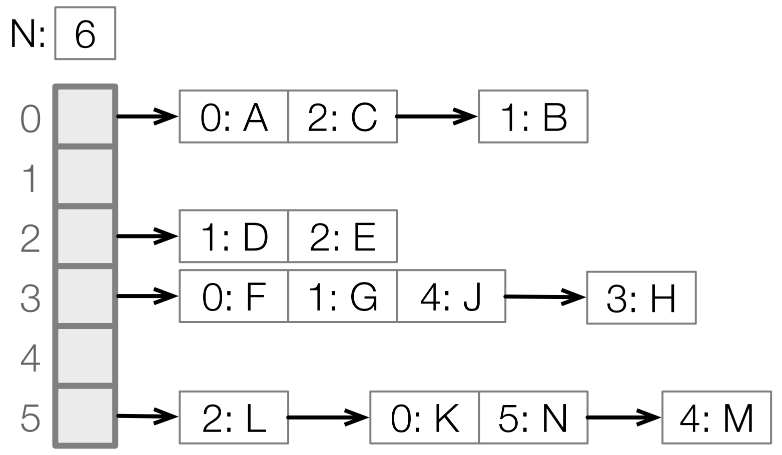
```
def list {
  e : elem nonempty
  n : list
  seq = {e}, n
}

def list_head {
  h : list
}
```

Linked List



```
def blist {
  e : elem[B] nonempty
  n : blist
  B : size in [0, 3]
  seq = {e}, n
}

def blist_head {
  h : blist
}
```

Block Linked List



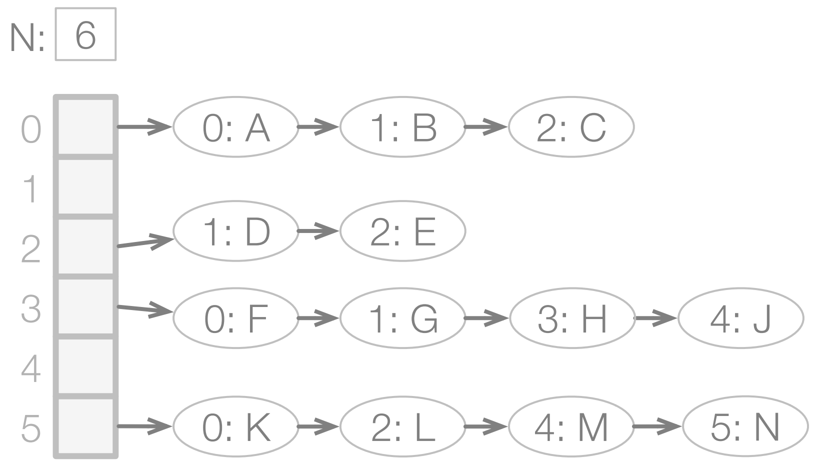
```
def vblist {
  e : elem[B] nonempty
  n : vblist
  B : size
  seq = {e}, n
}

def vblist_head {
  h : vblist
}
```

Variable Block Linked List

Dynamic Sparse Tensors

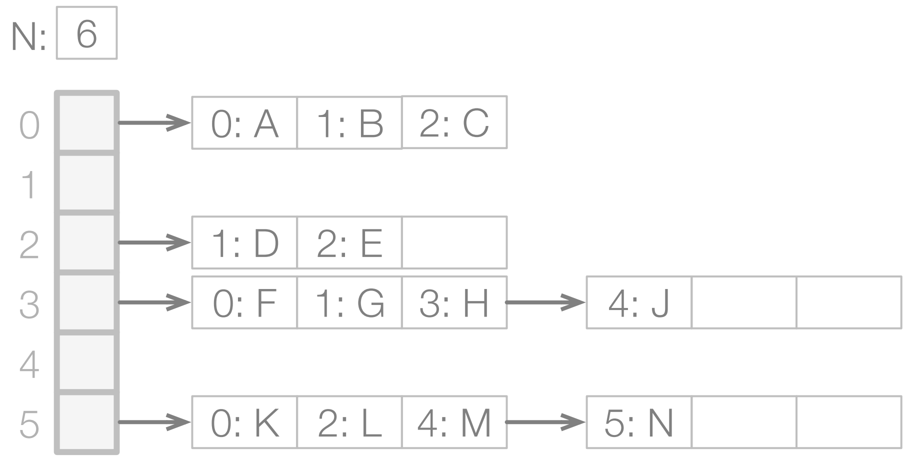
- Need pointer-based, recursive data structures
- Novel Node Schema Language
  - Automatically generate the data structures
  - Automatically Generate the code for iteration

	0	1	2	3	4	5
0	A	B	C			
1						
2		D	E			
3	F	G		H	J	
4						
5	K		L		M	N



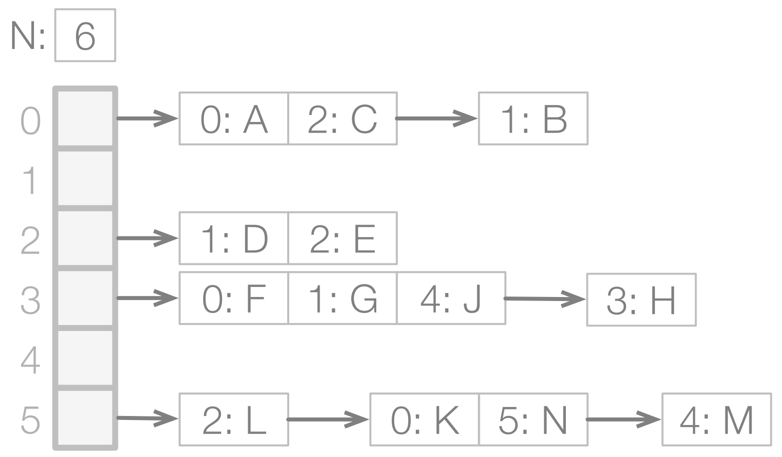
```
def list {
  e : elem nonempty
  n : list
  seq = {e}, n
}

def list_head {
  h : list
}
```

Linked List



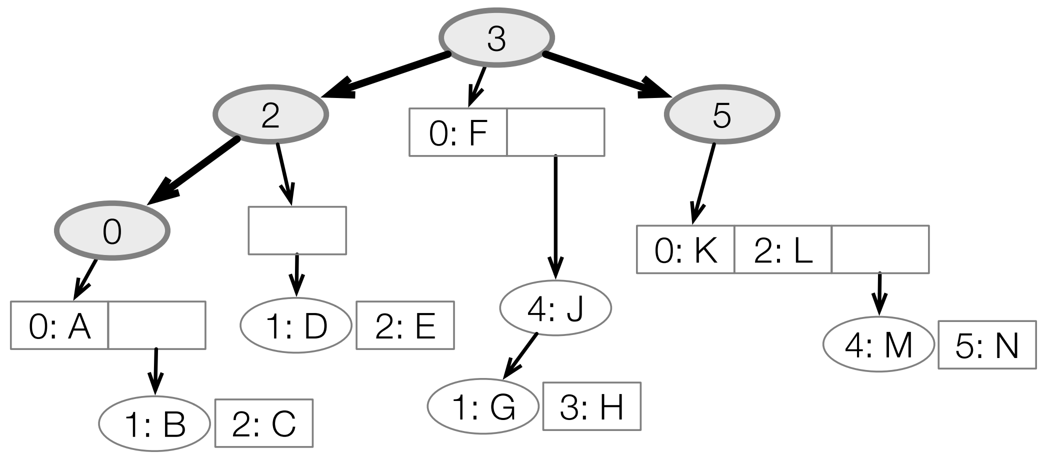
```
def blist {
  e : elem[B] nonempty
  n : blist
  B : size in [0, 3]
  seq = {e}, n
}

def blist_head {
  h : blist
}
```

Block Linked List



```
def vblist {
  e : elem[B] nonempty
  n : vblist
  B : size
  seq = {e}, n
}

def vblist_head {
  h : vblist
}
```

Variable Block Linked List



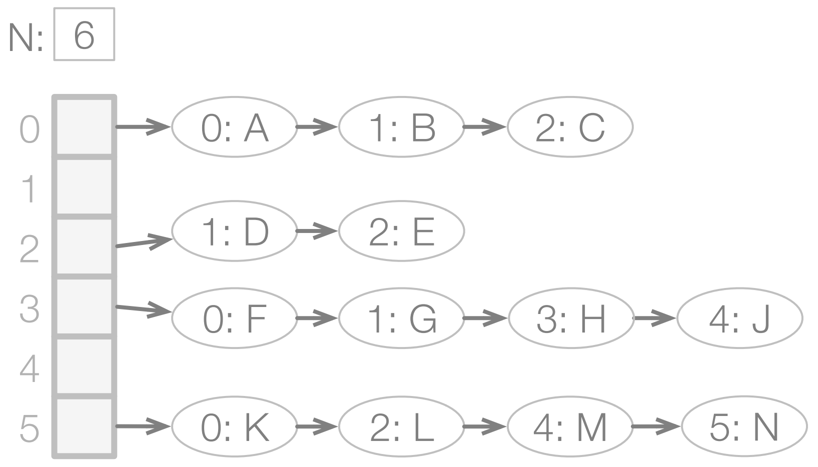
```
def ctree {
  h : elem nonempty
  t : elem[N] nonempty
  l : ctree
  r : ctree
  N : size
  seq = l, h, {t}, r
}

def prefix {
  e : elem[N] nonempty
  r : ctree
  N : size
  seq = {e}, r
}
```

C-Tree

Dynamic Sparse Tensors

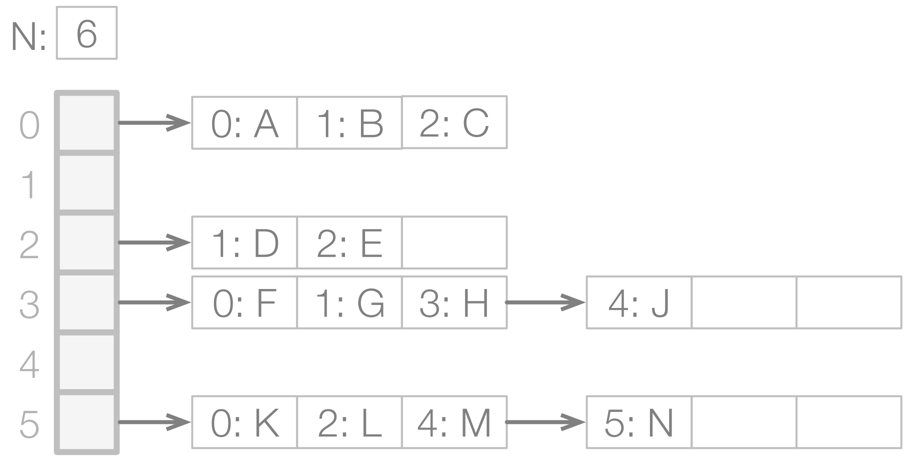
- Need pointer-based, recursive data structures
- Novel Node Schema Language
  - Automatically generate the data structures
  - Automatically Generate the code for iteration

	0	1	2	3	4	5
0	A	B	C			
1						
2		D	E			
3	F	G		H	J	
4						
5	K		L		M	N



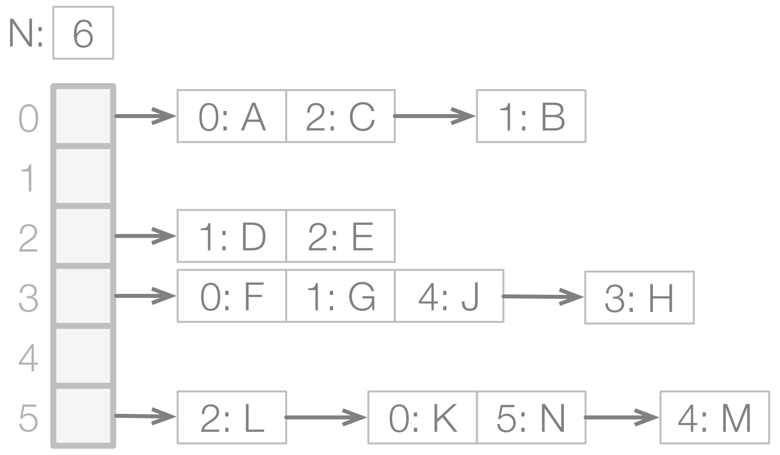
```
def list {
  e : elem nonempty
  n : list
  seq = {e}, n
}

def list_head {
  h : list
}
```

Linked List



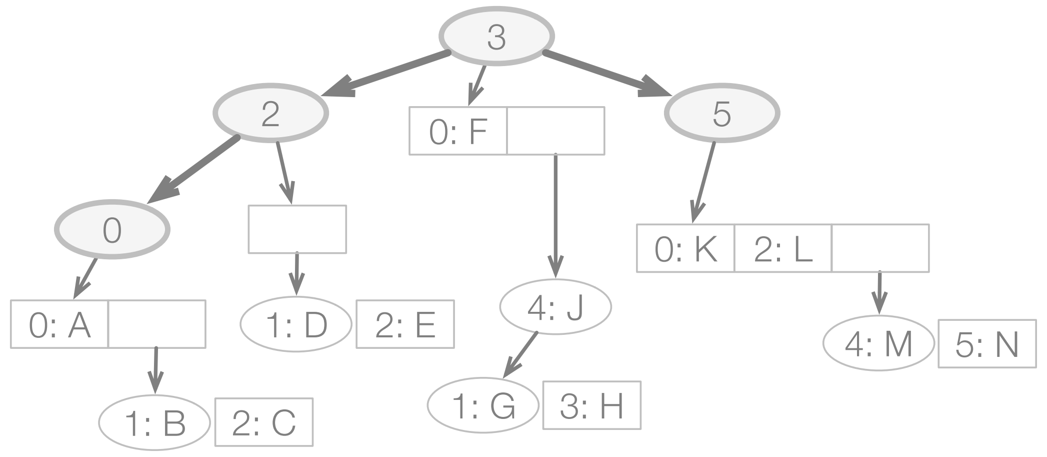
```
def blist {
  e : elem[B] nonempty
  n : blist
  B : size in [0, 3]
  seq = {e}, n
}

def blist_head {
  h : blist
}
```

Block Linked List



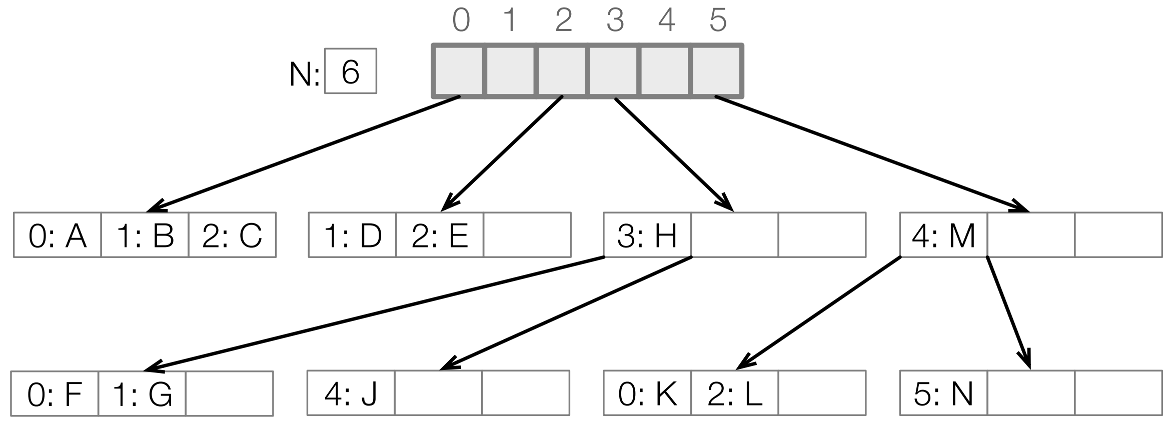
```
def vblist {
  e : elem[B] nonempty
  n : vblist
  B : size
  seq = {e}, n
}

def vblist_head {
  h : vblist
}
```

Variable Block Linked List



```
def ctree {
  h : elem nonempty
  t : elem[N] nonempty
  l : ctree
  r : ctree
  N : size
  seq = l, h, {t}, r
}

def prefix {
  e : elem[N] nonempty
  r : ctree
  N : size
  seq = {e}, r
}
```

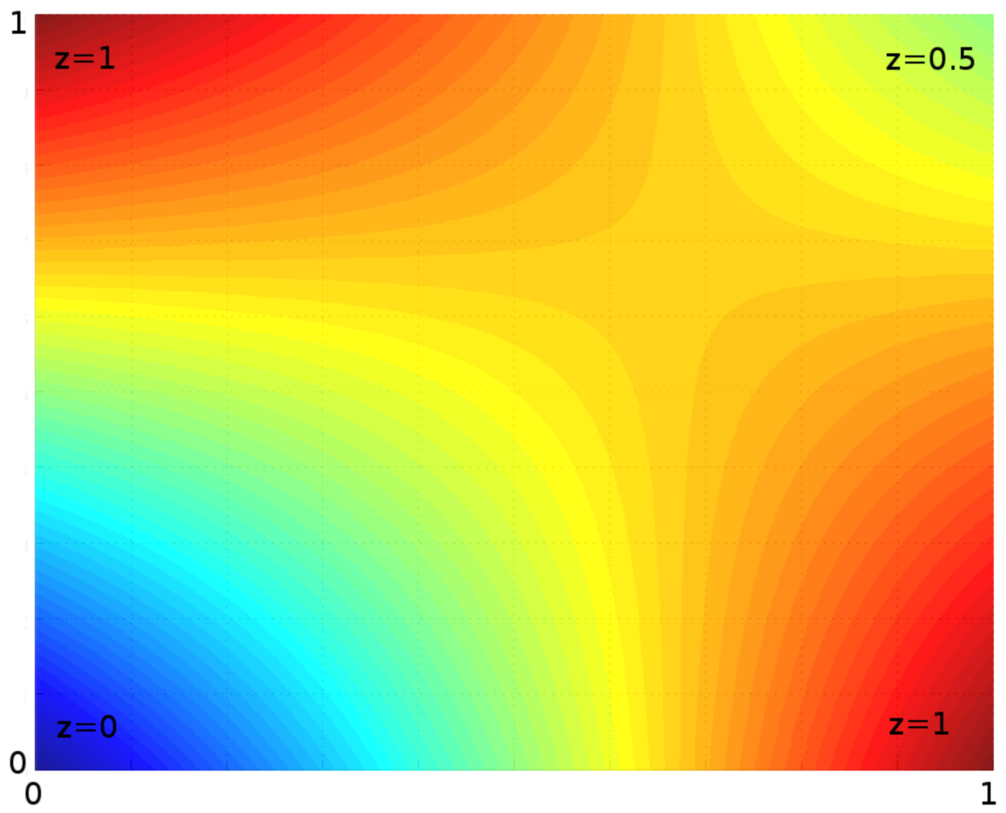
C-Tree



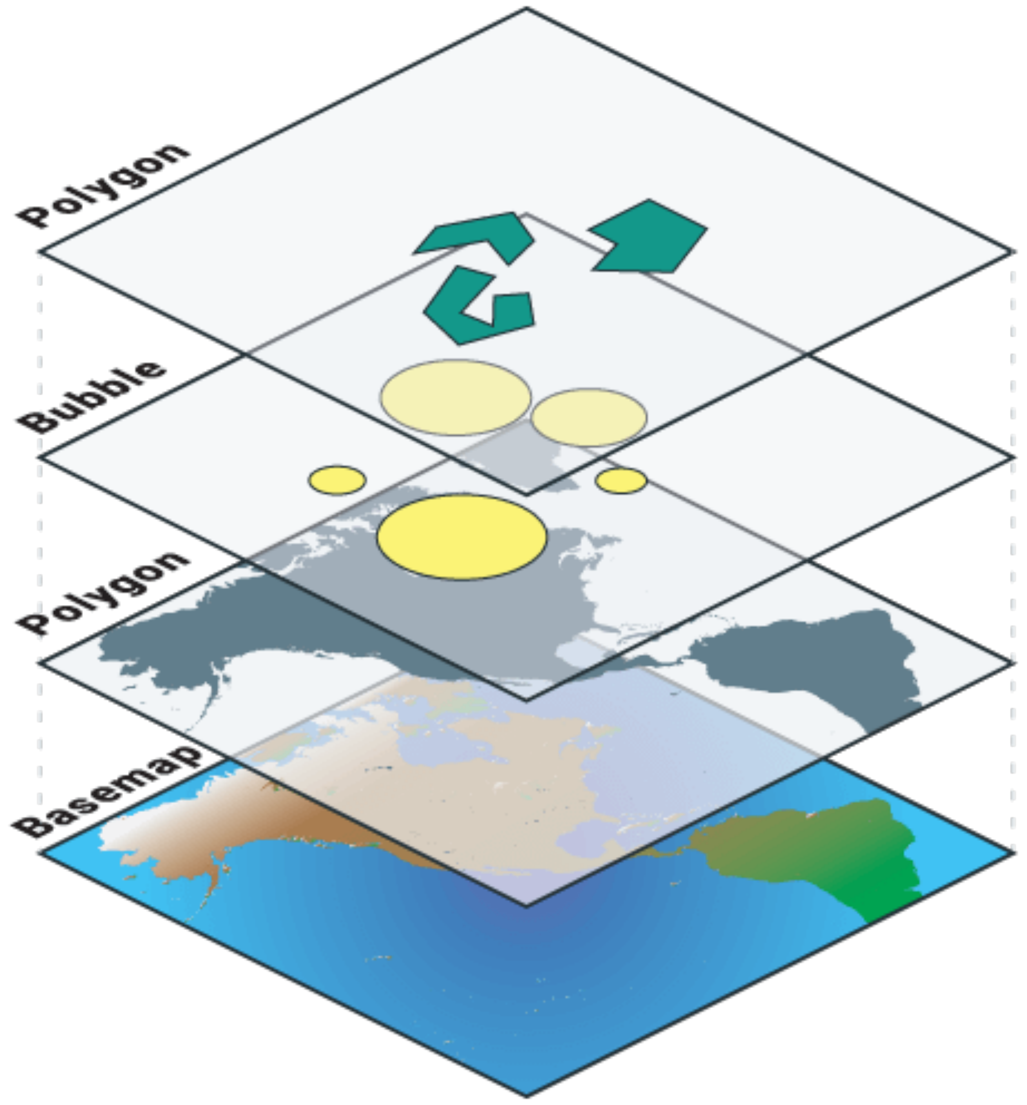
```
def supertype btree

def btree_internal : btree {
  e  : elem[B] nonempty
  c  : btree[B] nonempty
  cl : btree nonempty
  B  : size in [1, 3]
  seq = {c, e}, cl
}

def btree_leaf : btree {
  e : elem[B] nonempty
  B : size in [1, 3]
  seq = {e}
}

def btree_root {
  r : btree
}
```

B-Tree

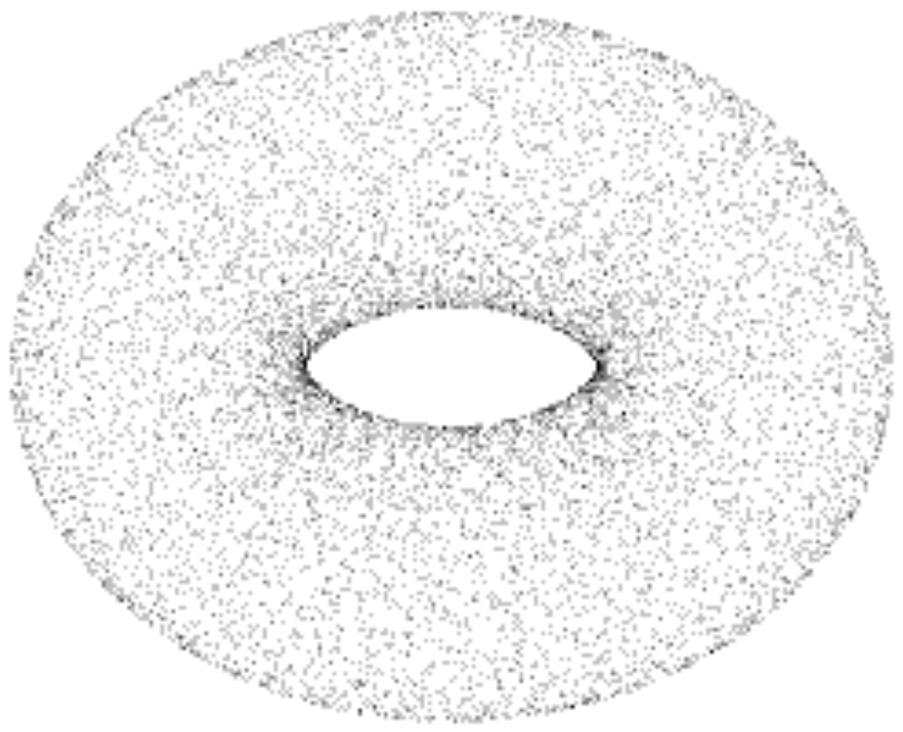
Programming On Continuous Data



Continuous Function



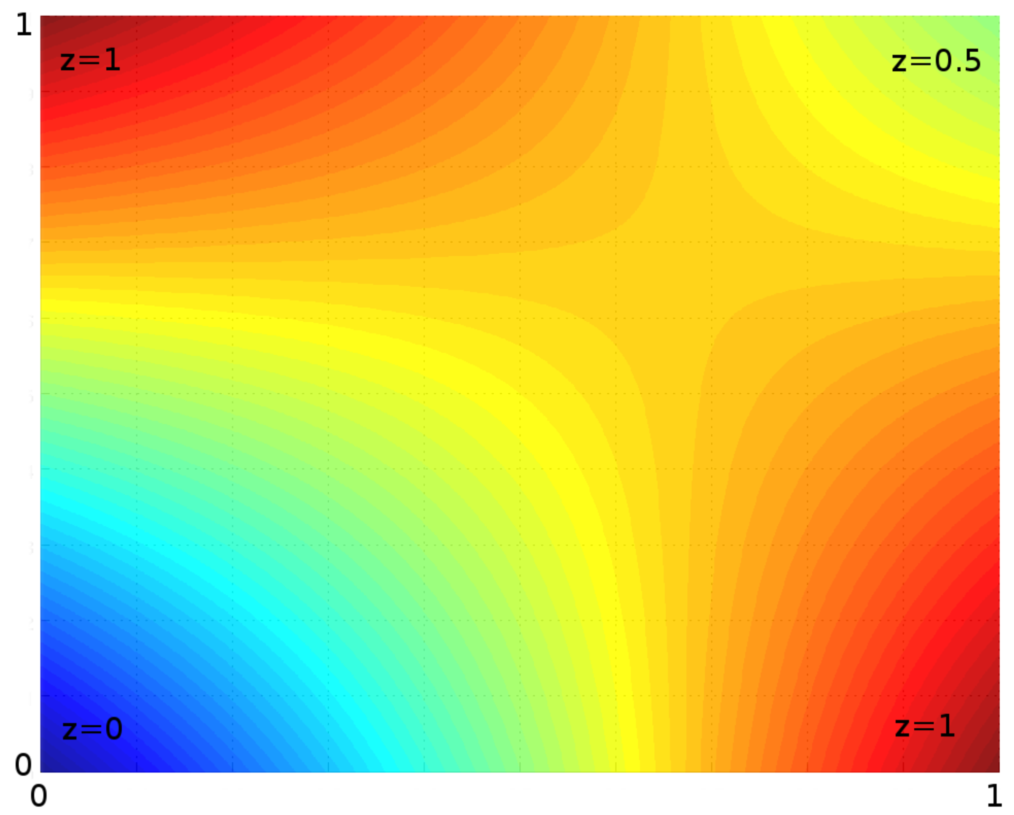
Spatial Database



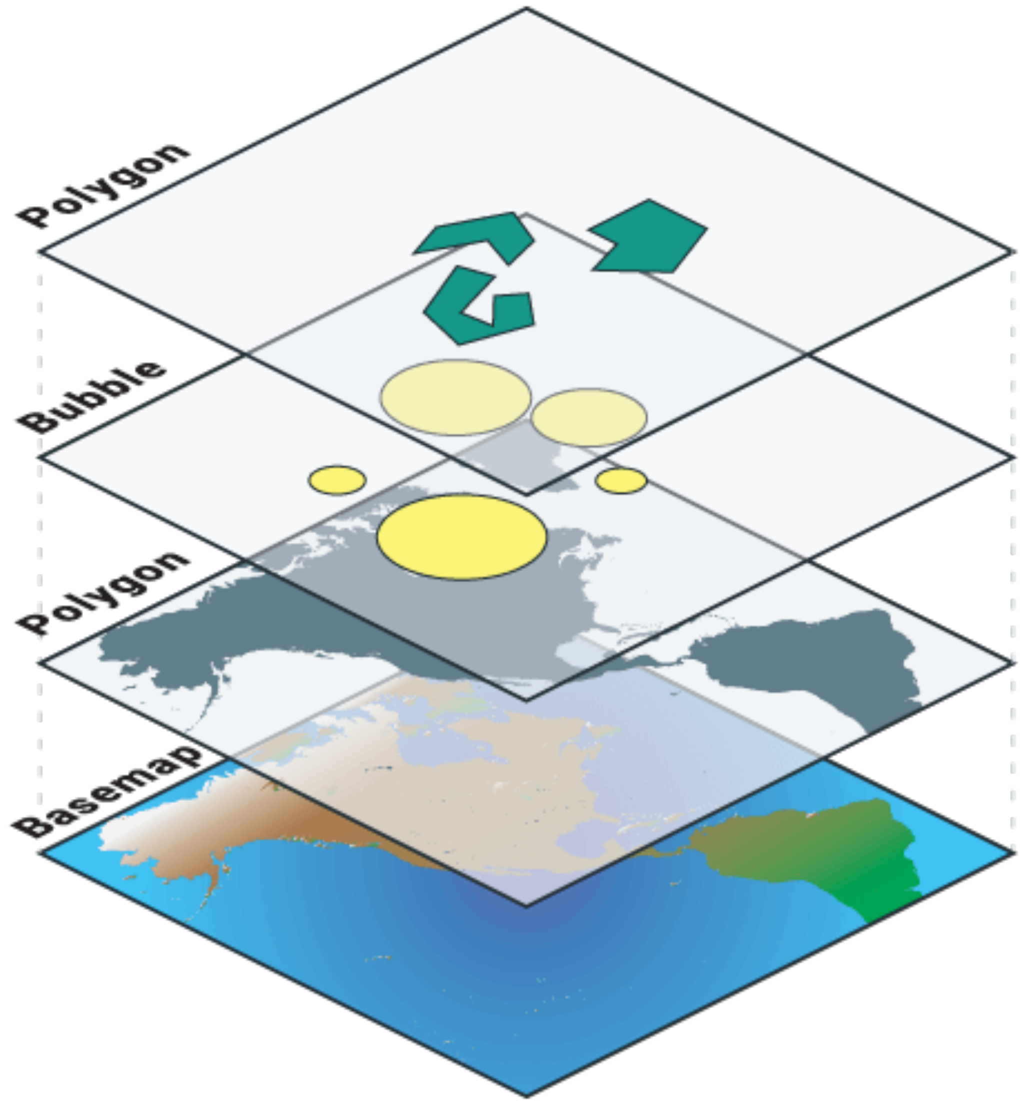
3D Point Cloud

Computer Graphics

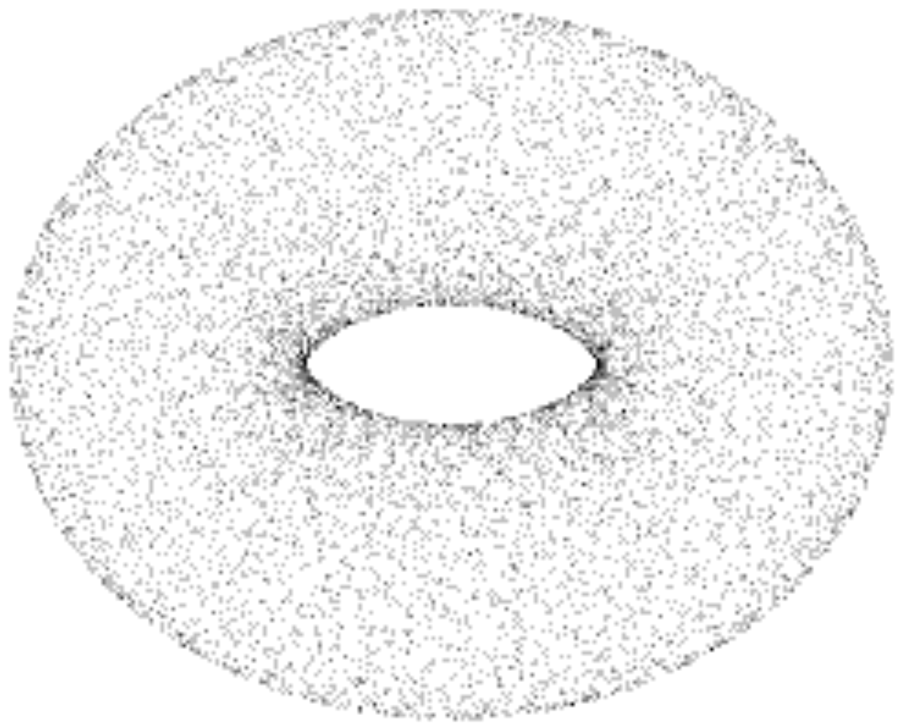
Programming On Continuous Data



Continuous Function



Spatial Database

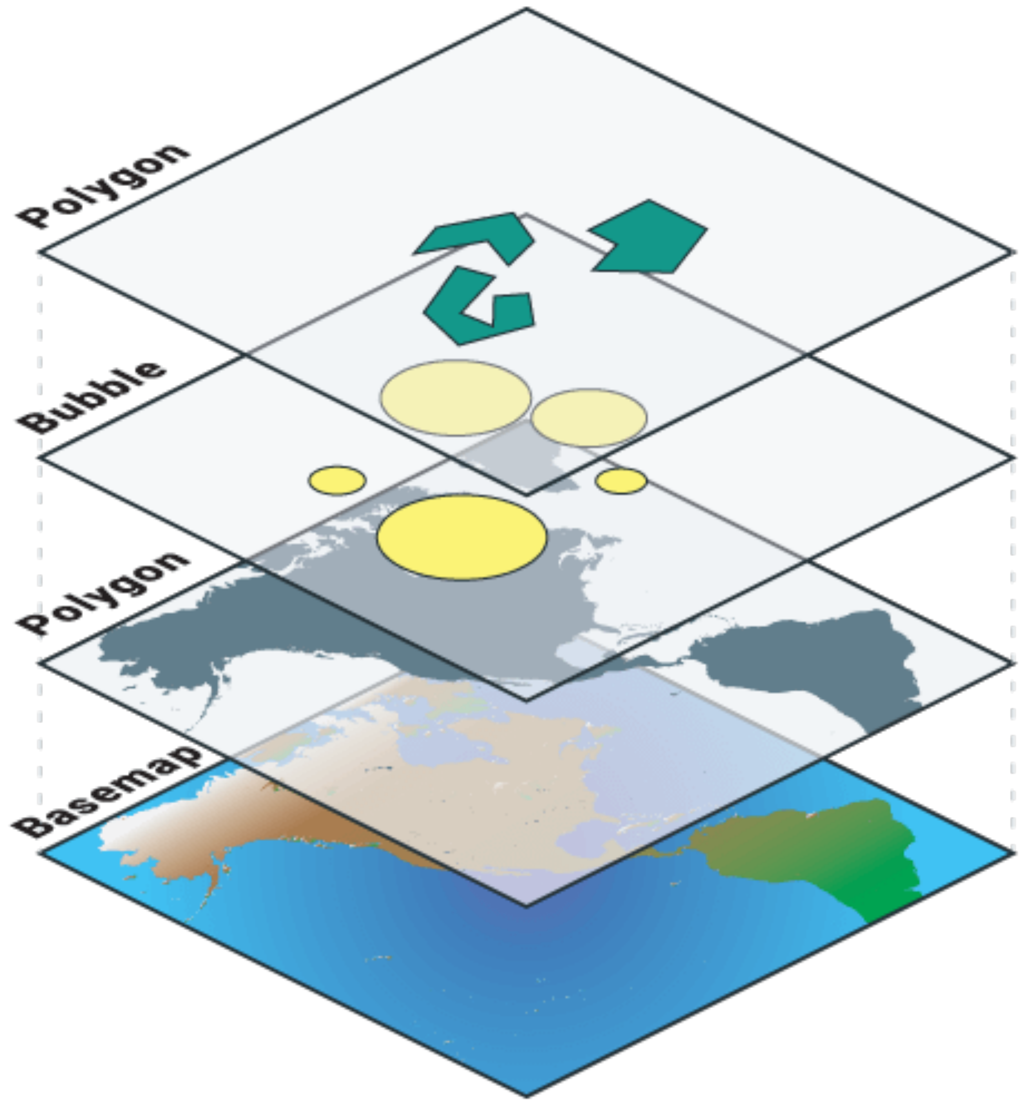


3D Point Cloud

Computer Graphics

Programming On Continuous Domain Is Difficult!

Continuous Function



Spatial Database

3D Point Cloud

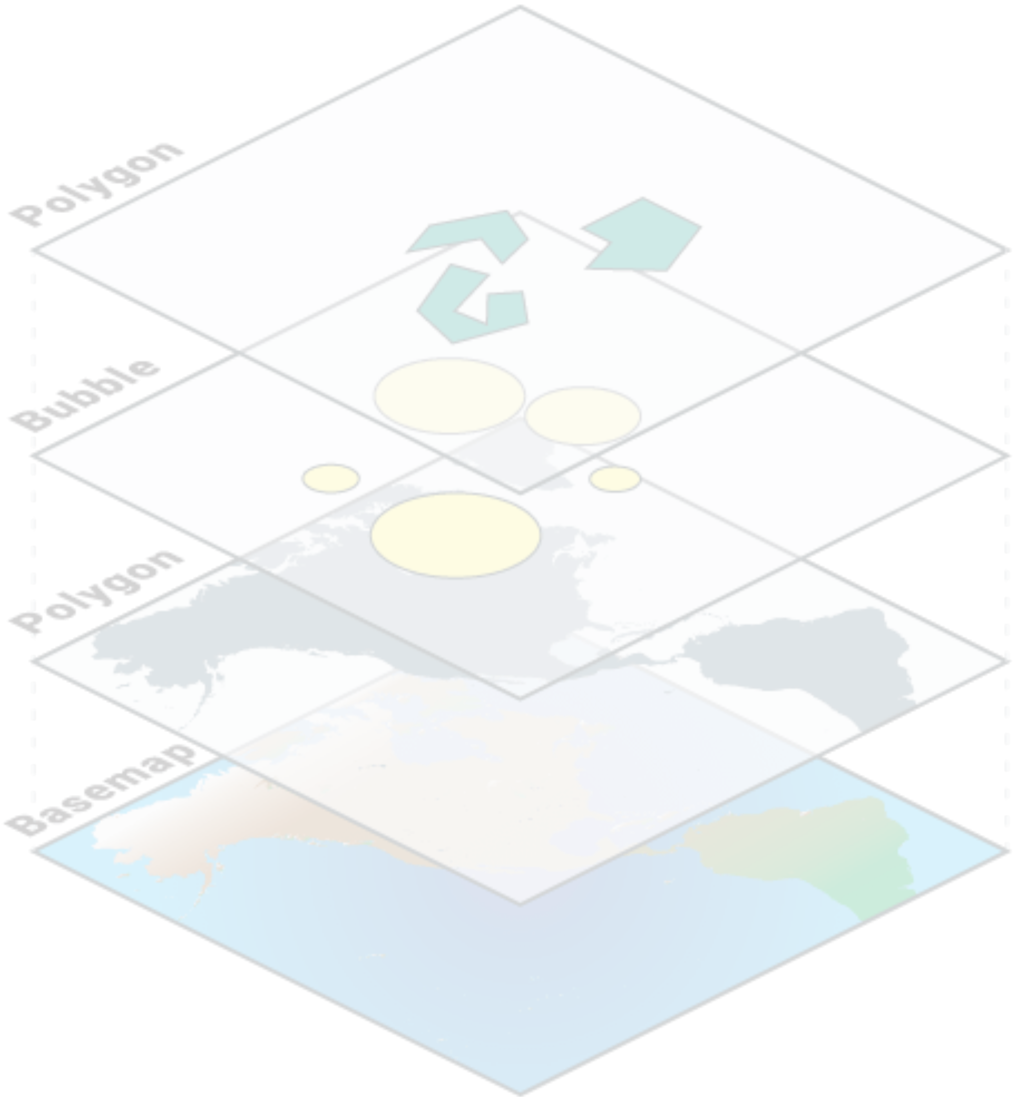
Computer Graphics

1. Storing or Iterating over geometries are non-trivial
⇒ (Quadtree/Octree, Bounding Volume Hierarchy..)

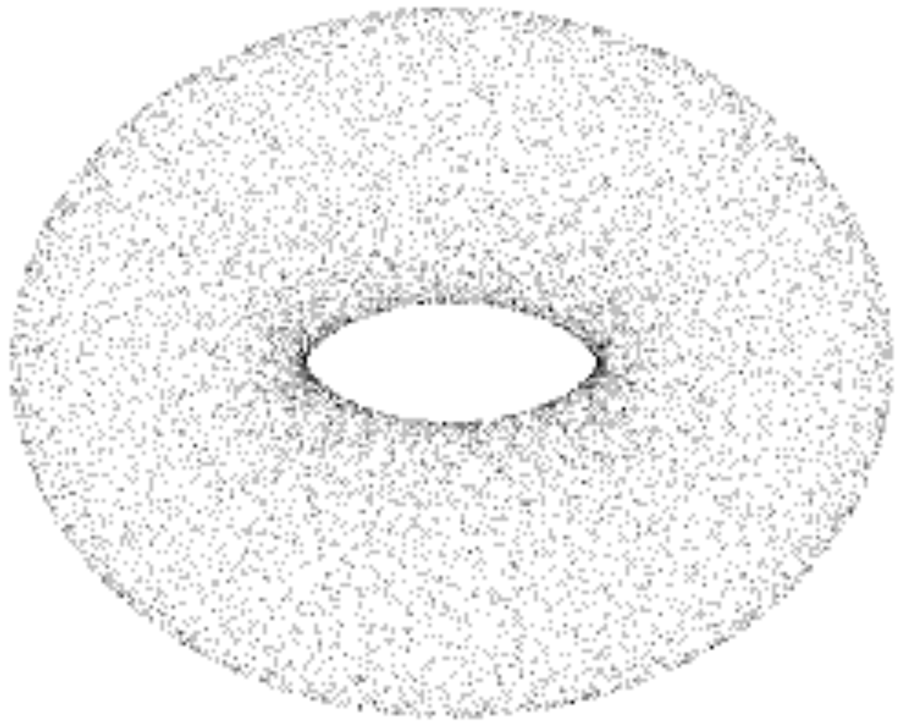
2. **501 Lines of Code** in hand-written library (Box search query, C++)

Programming On Continuous Domain Is Difficult!

Continuous Function



Spatial Database



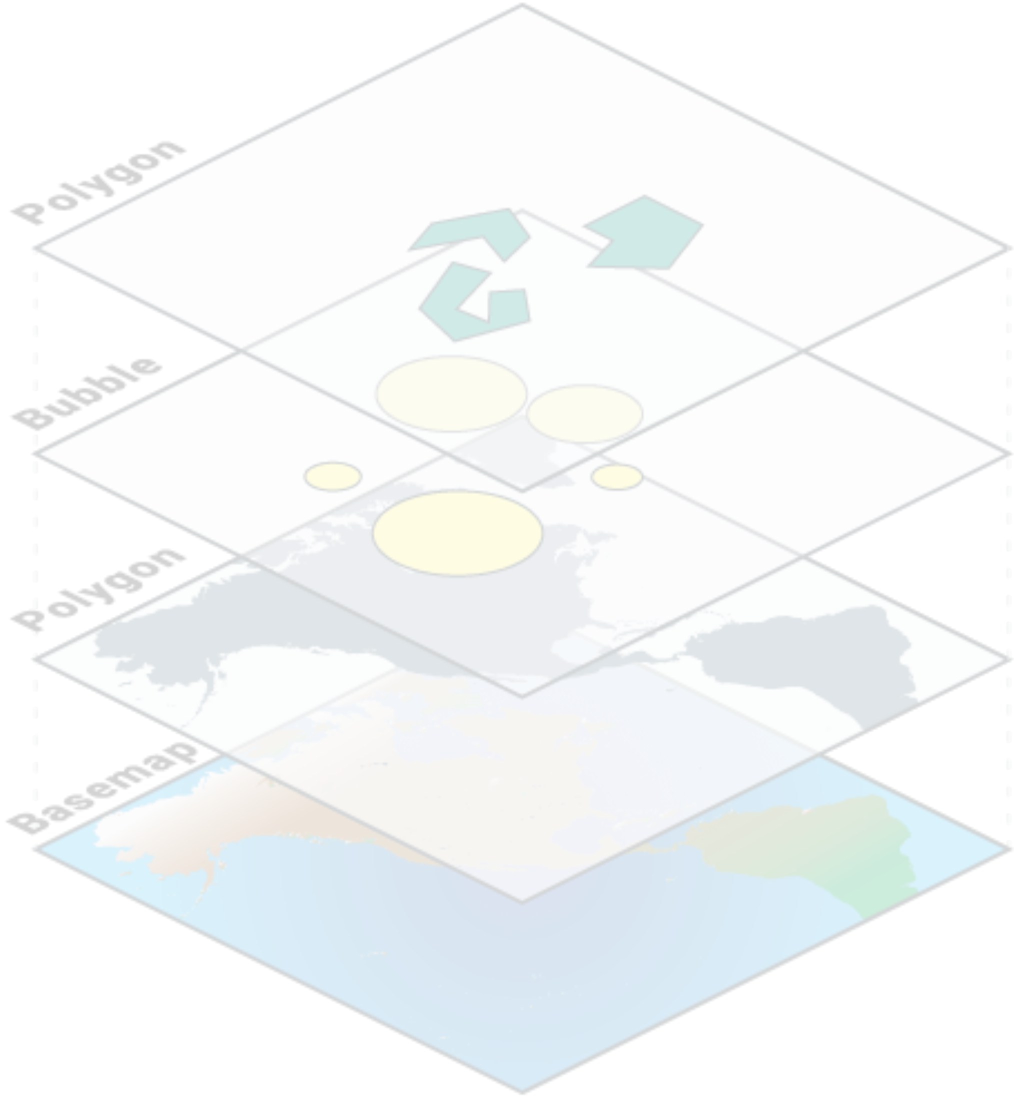
3D Point Cloud

Computer Graphics

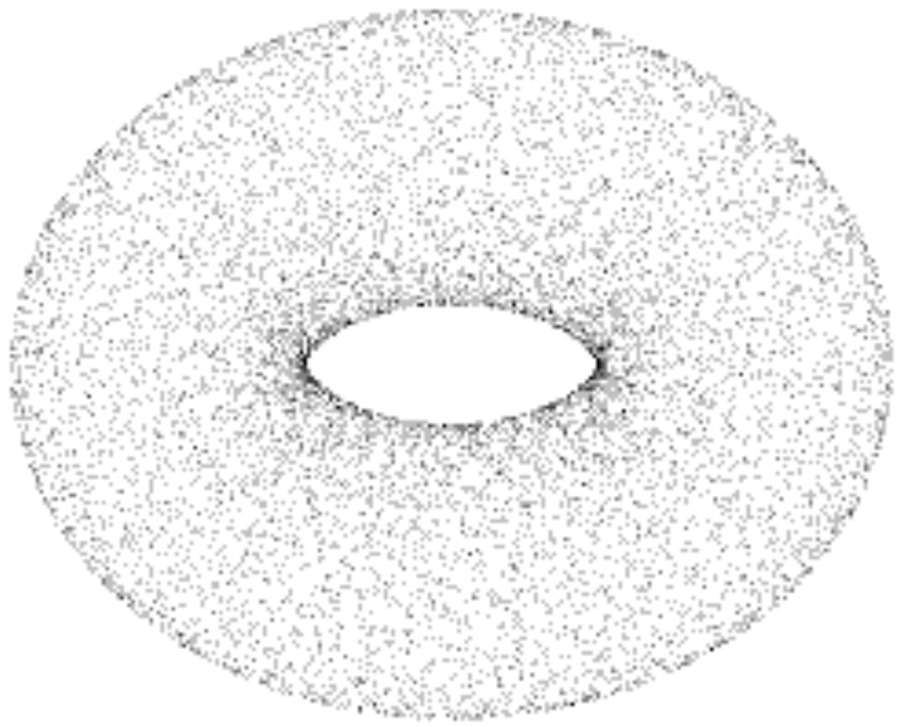
1. Core kernel(KPConv) can be expressed in **a single math equation**.
2. **2,330 Lines of Code** in PyTorch and C.

Programming On Continuous Domain Is Difficult!

Continuous Function



Spatial Database



3D Point Cloud

Computer Graphics

1. Core kernel(KPConv) can be expressed in **a single math equation**.

2. **2,330 Lines of Code** in PyTorch and C.

Arrays Are

- Multi-dimensional
- Rectilinear
- ~~Dense~~
- Integer grid

Of points

Arrays Are

- Multi-dimensional
- Rectilinear
- ~~Dense~~
- ~~Integer~~ grid

Of points

The Continuous Tensor Abstraction: Fresh Perspective On Tensor And Loops

```
A[2]
```

```
// i=[0,1]
for i = 0:1
```

Existing Tensor Abstraction

The Continuous Tensor Abstraction: Fresh Perspective On Tensor And Loops

```
A[2]
```

```
// i=[0,1]
for i = 0:1
```

Existing Tensor Abstraction

Continuous Tensor Abstraction

```
A[3.1415]
```

// i = $\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$

```
for i = 0.0:1.0
```

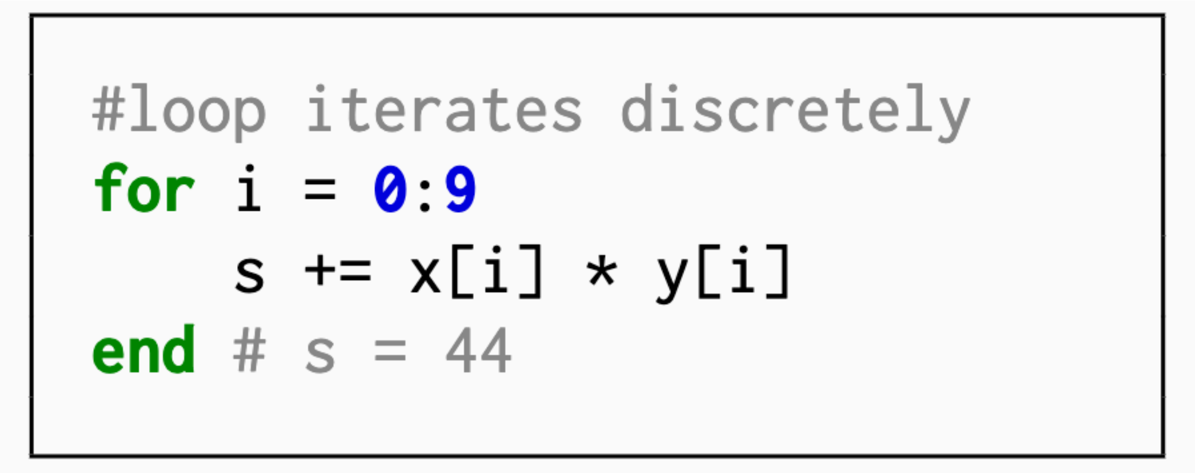
Real-Numbered Index Access

For loop on continuous domain

Comparing To Existing Array Programming Model

x[i]	0	1	0	2	0	0	3	0	0	4
y[i]	0	0	5	6	0	7	0	0	0	8
	0	1	2	3	4	5	6	7	8	9

Vector
(integer domain)



```
#loop iterates discretely
for i = 0:9
    s += x[i] * y[i]
end # s = 44
```

Existing tensor programming

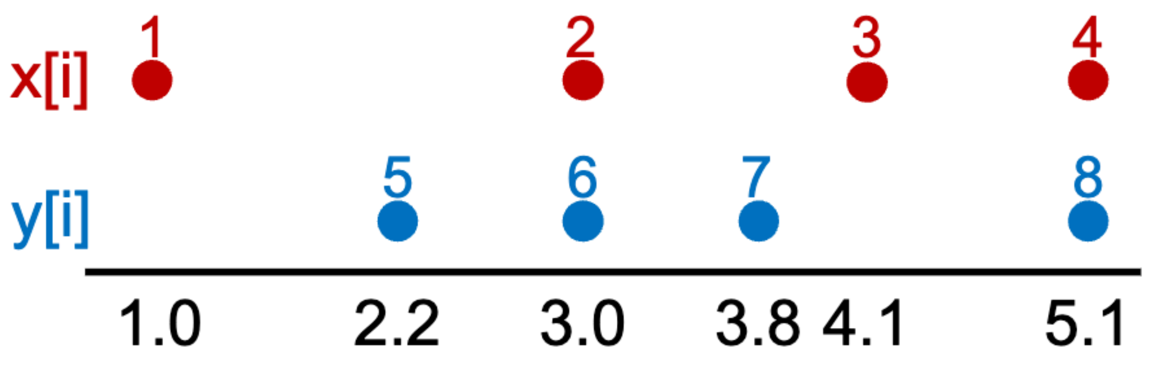
Comparing To Existing Array Programming Model

x[i]	0	1	0	2	0	0	3	0	0	4
y[i]	0	0	5	6	0	7	0	0	0	8
	0	1	2	3	4	5	6	7	8	9

```
#loop iterates discretely
for i = 0:9
    s += x[i] * y[i]
end # s = 44
```

Existing Tensor Abstraction

Continuous Tensor Abstraction



Pinpoint Coordinates
on Continuous Domain

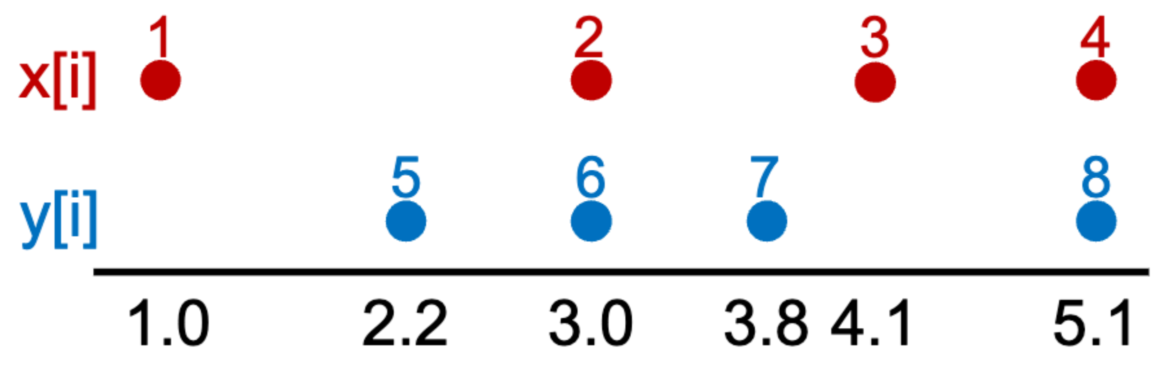
Comparing To Existing Array Programming Model

x[i]	0	1	0	2	0	0	3	0	0	4
y[i]	0	0	5	6	0	7	0	0	0	8
	0	1	2	3	4	5	6	7	8	9

```
#loop iterates discretely
for i = 0:9
    s += x[i] * y[i]
end # s = 44
```

Existing Tensor Abstraction

Continuous Tensor Abstraction

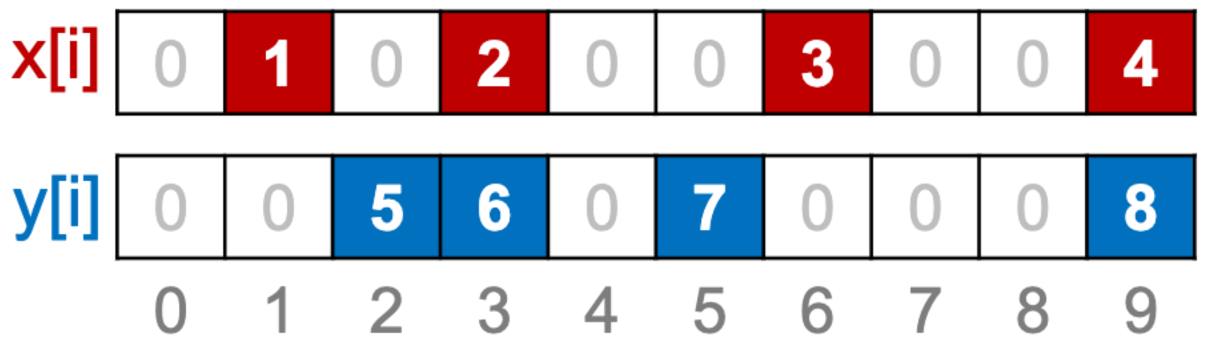


Pinpoint Coordinates
on Continuous Domain

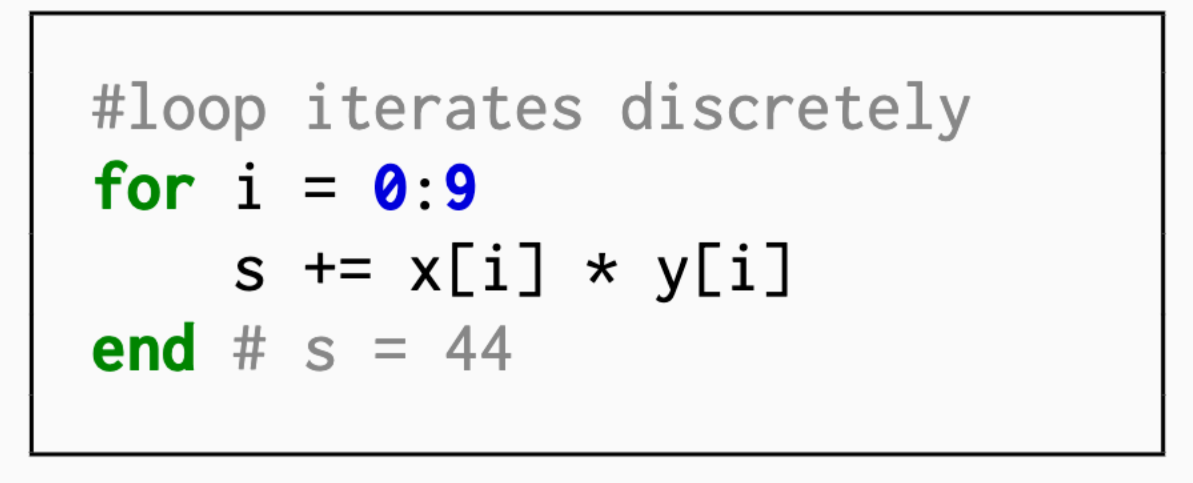
```
#loop iterates continuously
for i = 0.0:9.0
    s += x[i] * y[i]
end # s = 44
```

Continuous Dot-Product

Comparing To Existing Array Programming Model



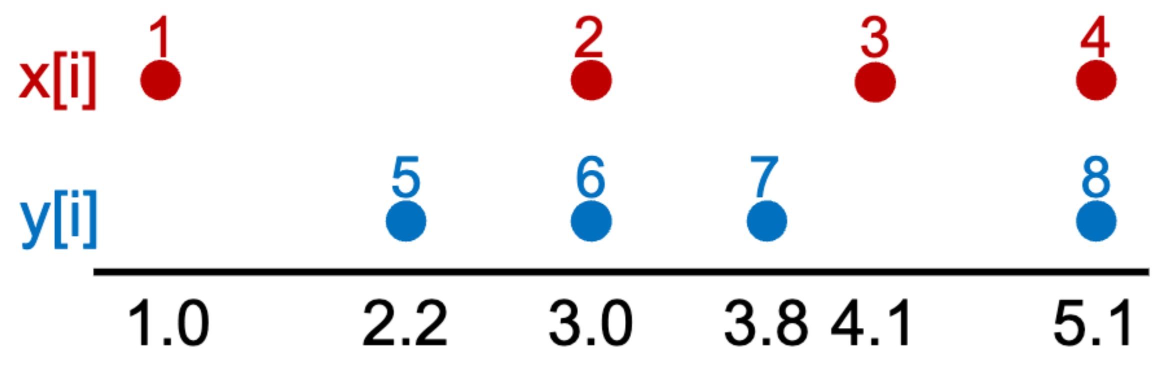
x[i]	0	1	0	2	0	0	3	0	0	4
y[i]	0	0	5	6	0	7	0	0	0	8
	0	1	2	3	4	5	6	7	8	9



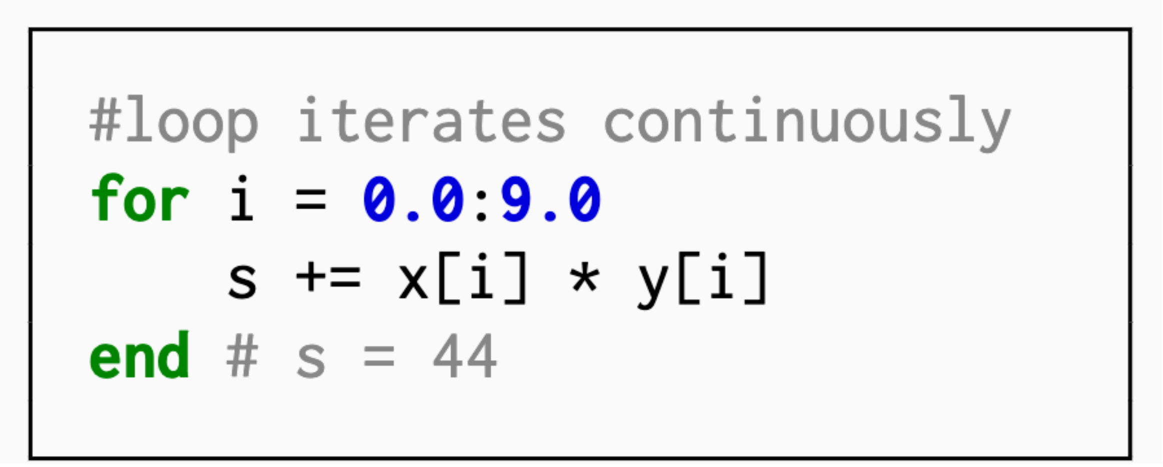
```
#loop iterates discretely
for i = 0:9
    s += x[i] * y[i]
end # s = 44
```

Existing Tensor Abstraction

Continuous Tensor Abstraction

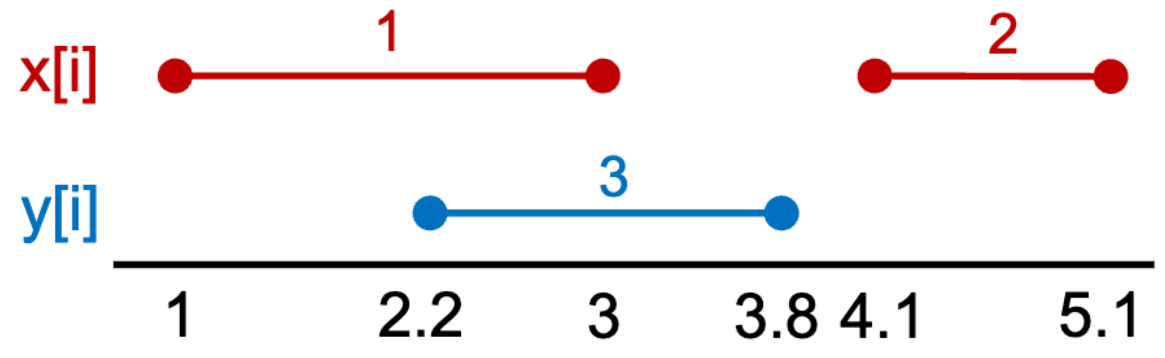


Pinpoint Coordinates
on Continuous Domain



```
#loop iterates continuously
for i = 0.0:9.0
    s += x[i] * y[i]
end # s = 44
```

Continuous Dot-Product



Interval Coordinates
on Continuous Domain

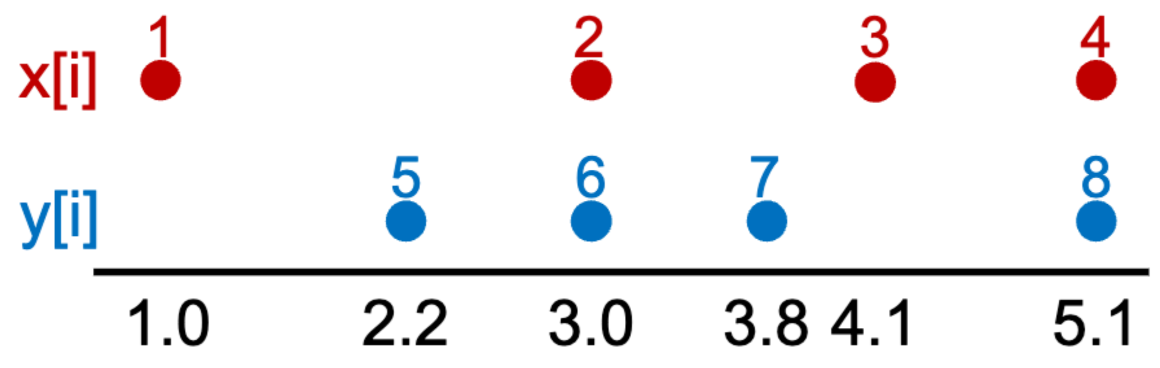
Comparing To Existing Array Programming Model

x[i]	0	1	0	2	0	0	3	0	0	4
y[i]	0	0	5	6	0	7	0	0	0	8
	0	1	2	3	4	5	6	7	8	9

```
#loop iterates discretely
for i = 0:9
    s += x[i] * y[i]
end # s = 44
```

Existing Tensor Abstraction

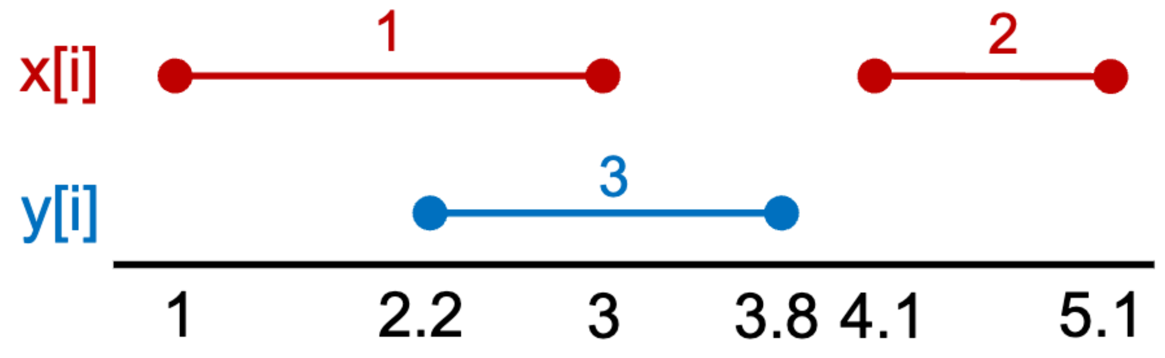
Continuous Tensor Abstraction



Pinpoint Coordinates
on Continuous Domain

```
#loop iterates continuously
for i = 0.0:9.0
    s += x[i] * y[i]
end # s = 44
```

Continuous Dot-Product

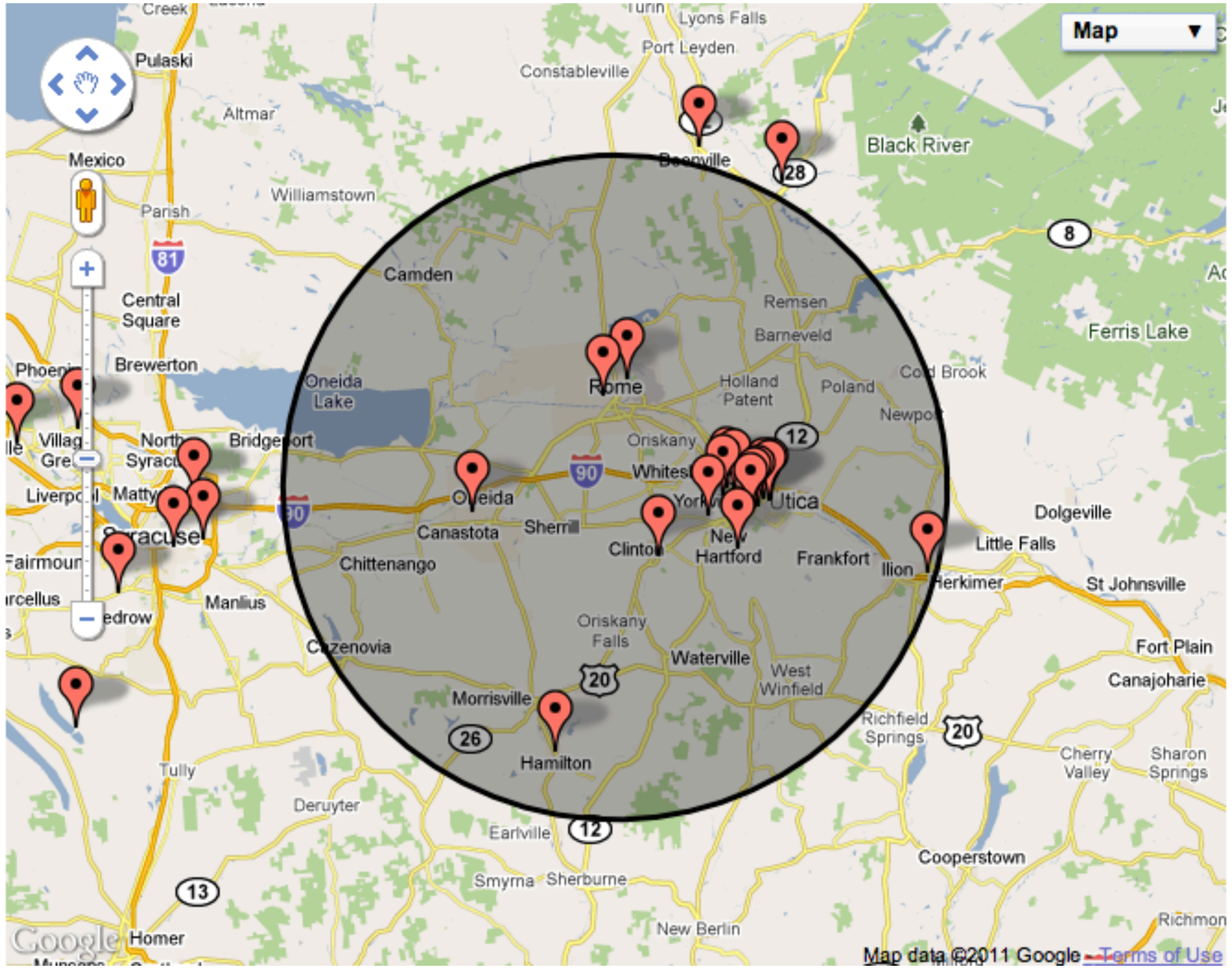


Interval Coordinates
on Continuous Domain

```
#loop iterates continuously
for i = 0.0:9.0
    s += x[i] * y[i] * d(i)
end # s = 2.4
```

$$s = s + \int_{0.0}^{9.0} x_i * y_i * di$$

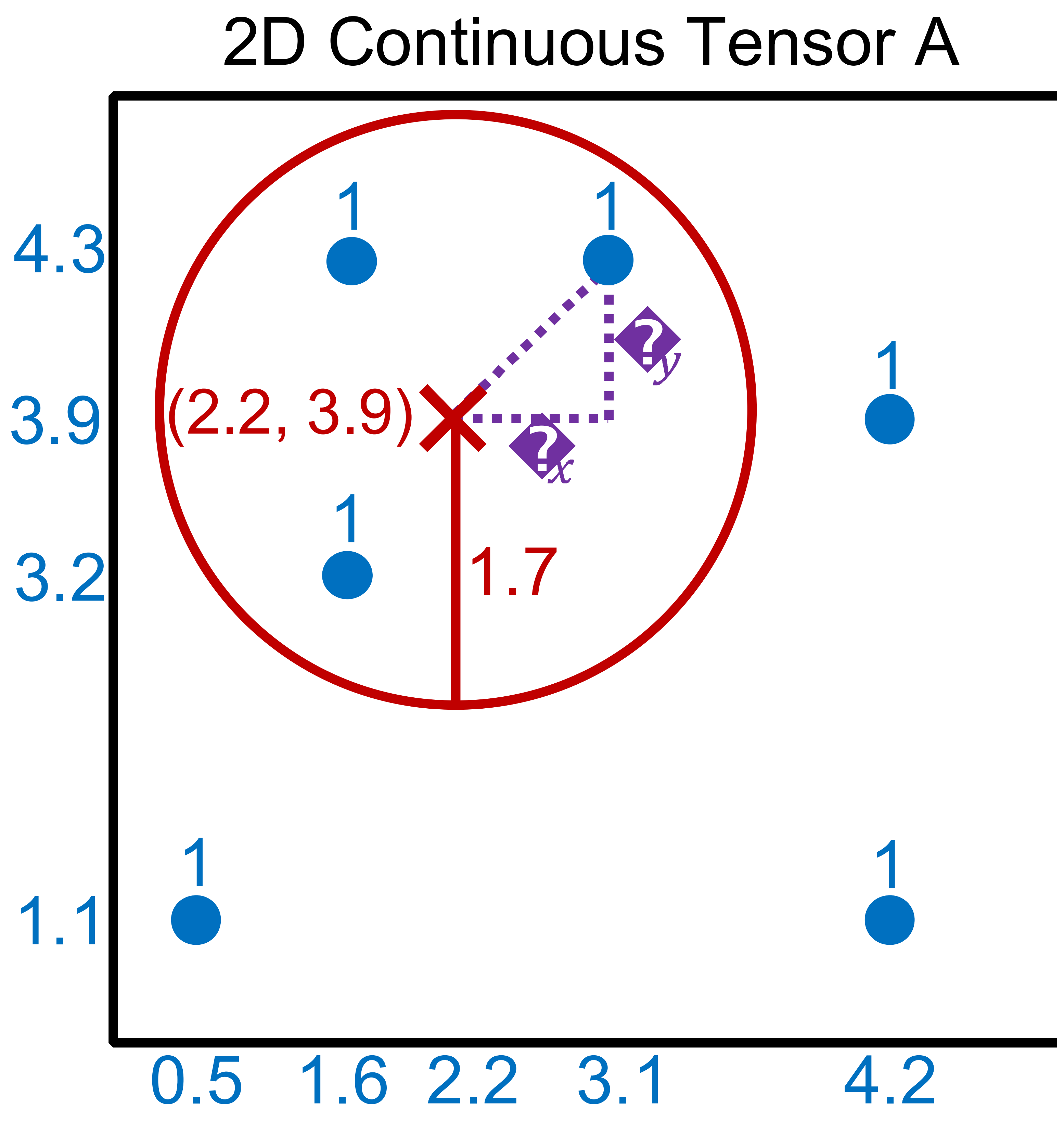
Motivational Example : Radius Search In Gis



Radius Search : Get the number of points within the distance R.

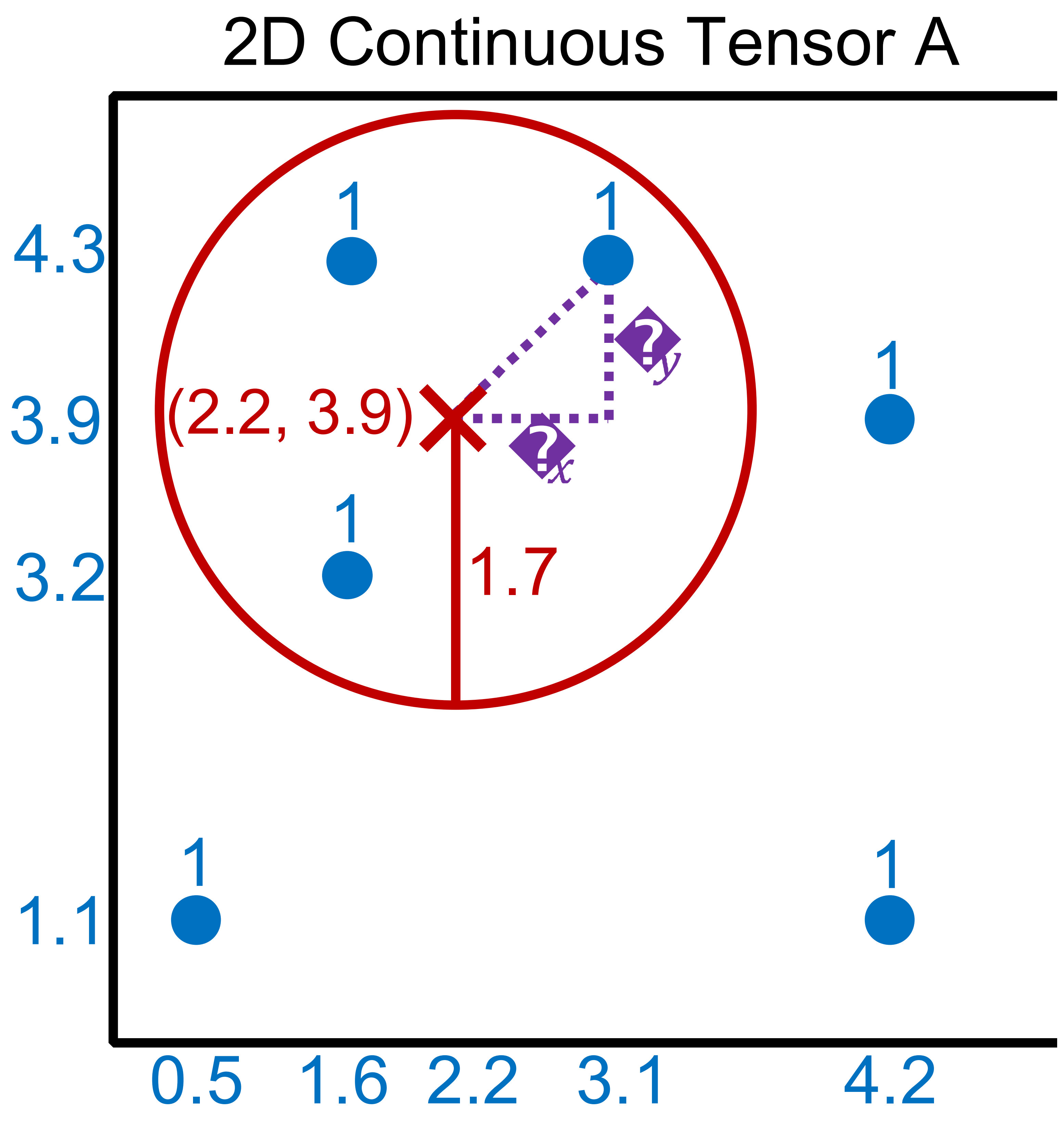
Motivational Example : Radius Search In Gis

```
count = 0
for dx=-1.7:1.7  # continuous
 for dy=-1.7:1.7 # continuous
  if dx*dx+dy*dy <= 1.7*1.7
   count += A[2.2+dx,3.9+dy]
# count = 3
```

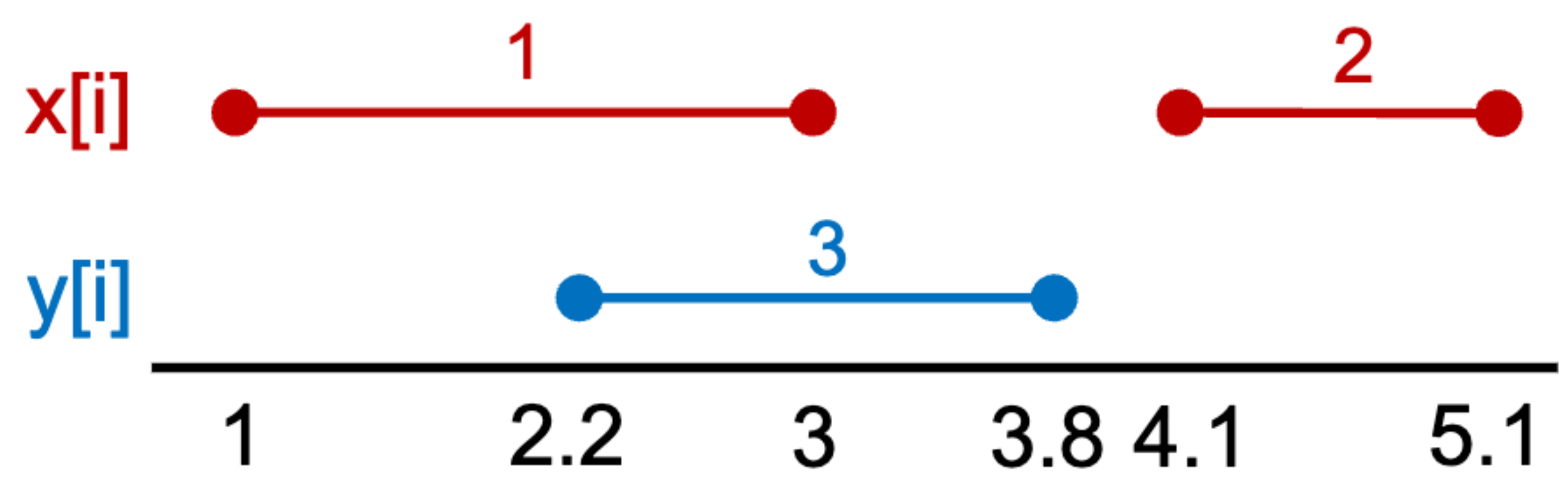


Motivational Example : Radius Search In Gis

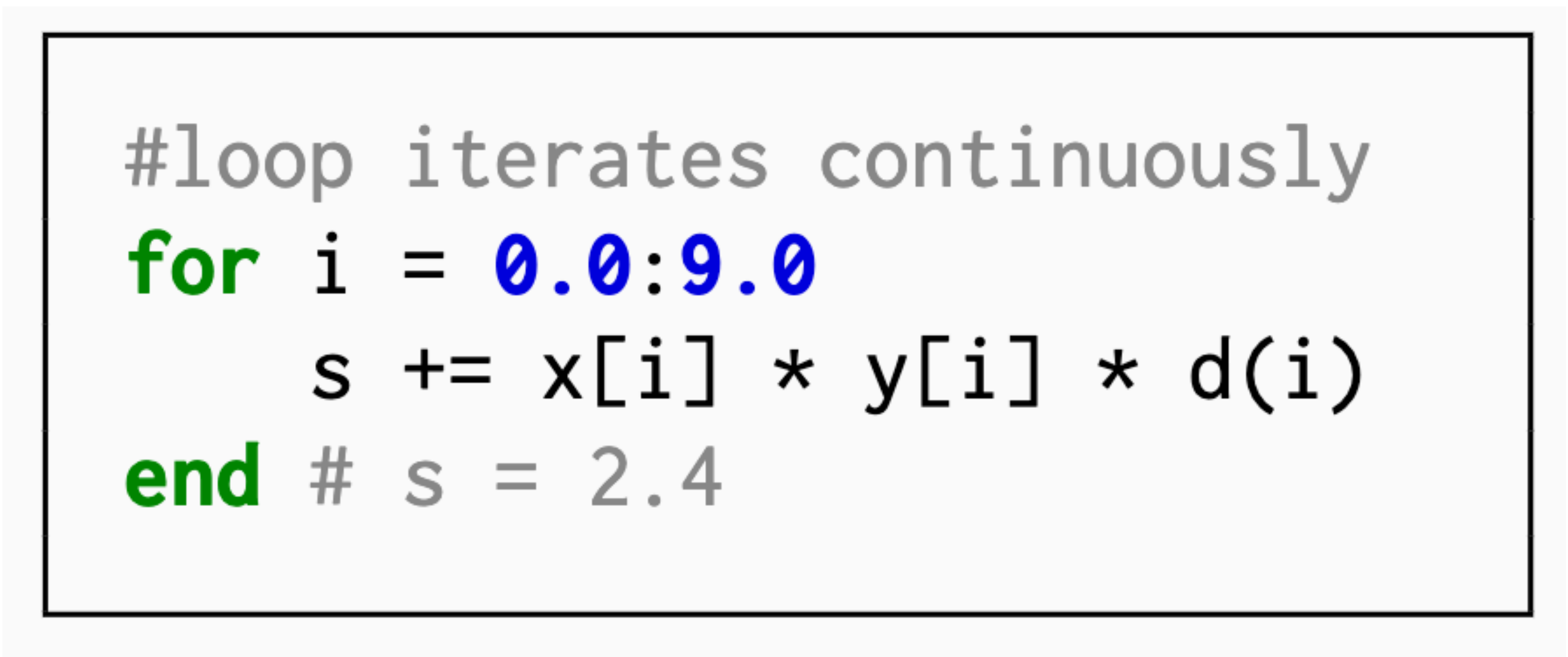
```
count = 0
for dx=-1.7:1.7  # continuous
 for dy=-1.7:1.7 # continuous
  if dx*dx+dy*dy <= 1.7*1.7
   count += A[2.2+dx,3.9+dy]
# count = 3
```



Research Questions



RQ1. How can we **store infinitely many** coordinates in continuous tensor?



```
#loop iterates continuously
for i = 0.0:9.0
    s += x[i] * y[i] * d(i)
end # s = 2.4
```

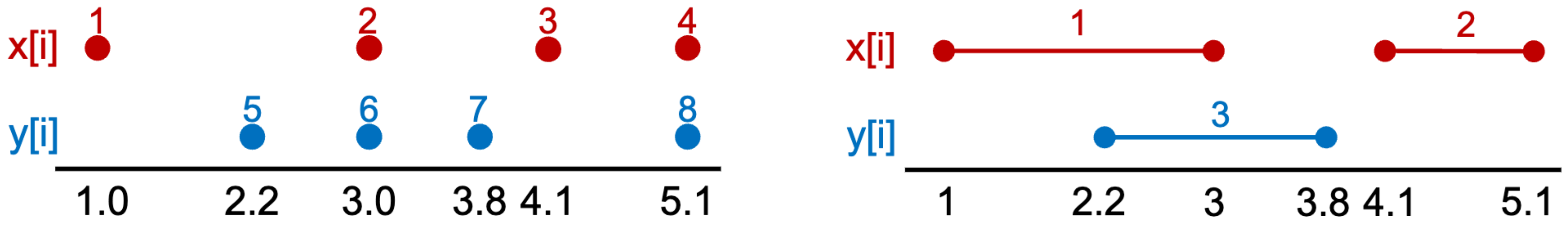
RQ2. How can we **iterate infinitely many** indices in continuous loop?

Piecewise-Constant Property

All Continuous Tensors must satisfy a piecewise-constant property

Piecewise-Constant Property

All Continuous Tensors must satisfy a piecewise-constant property



Piecewise-constant

$f(x) = cos(x)$

$g(x) = e^x$

Not Piecewise-constant

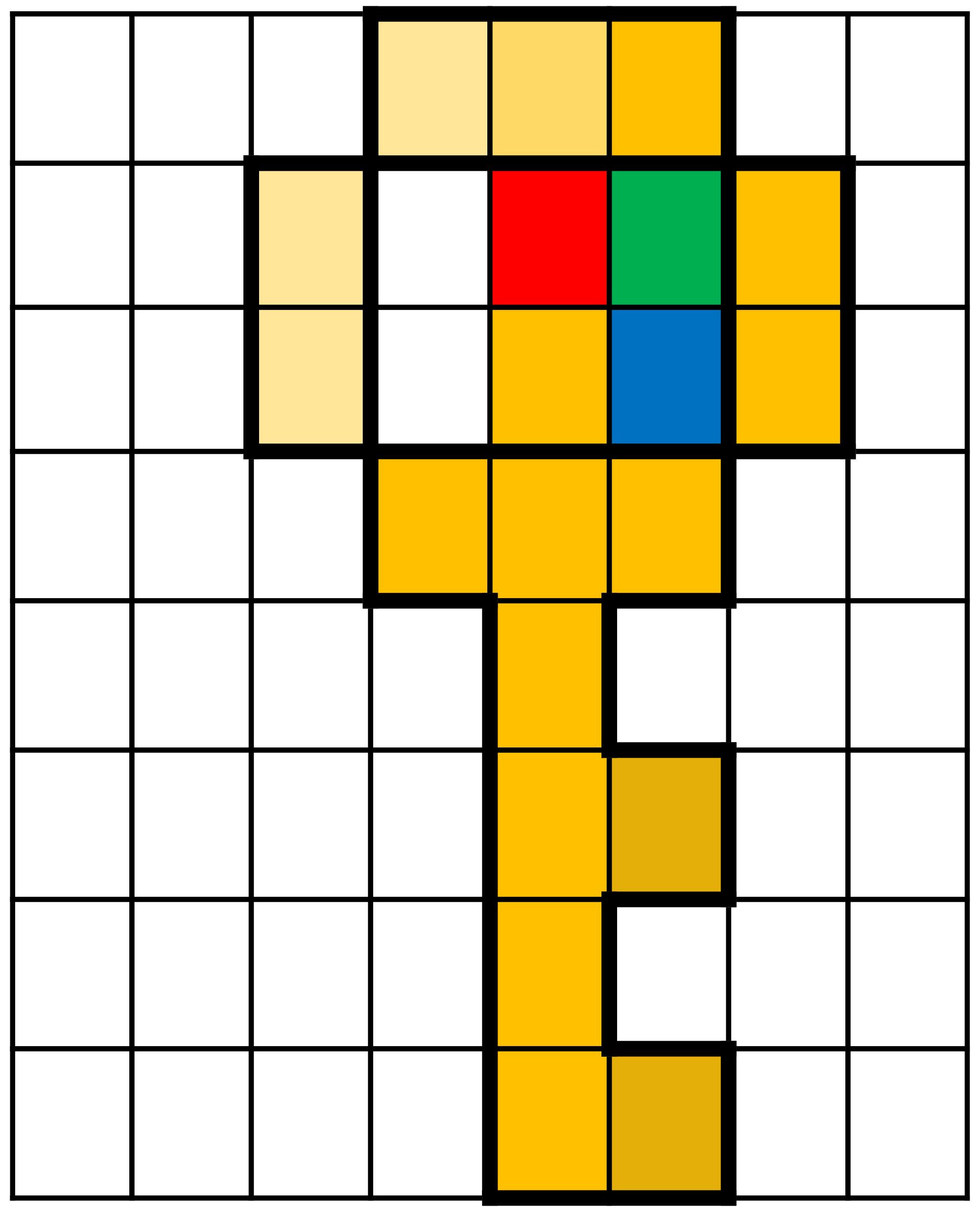
Case Studies

Applications	Baseline	Ours	LoC Saving	Perf Speedup
Radius Search Query	501 lines	5 lines	100×	9.20×
Point Cloud Convolution	2,330 lines	16 lines	145×	0.23×
Trilinear Interpolation in NeRF	82 lines	9 lines	9×	1.69×
Genomic Interval Overlapping Query	206 lines	8 lines	26×	1.22×

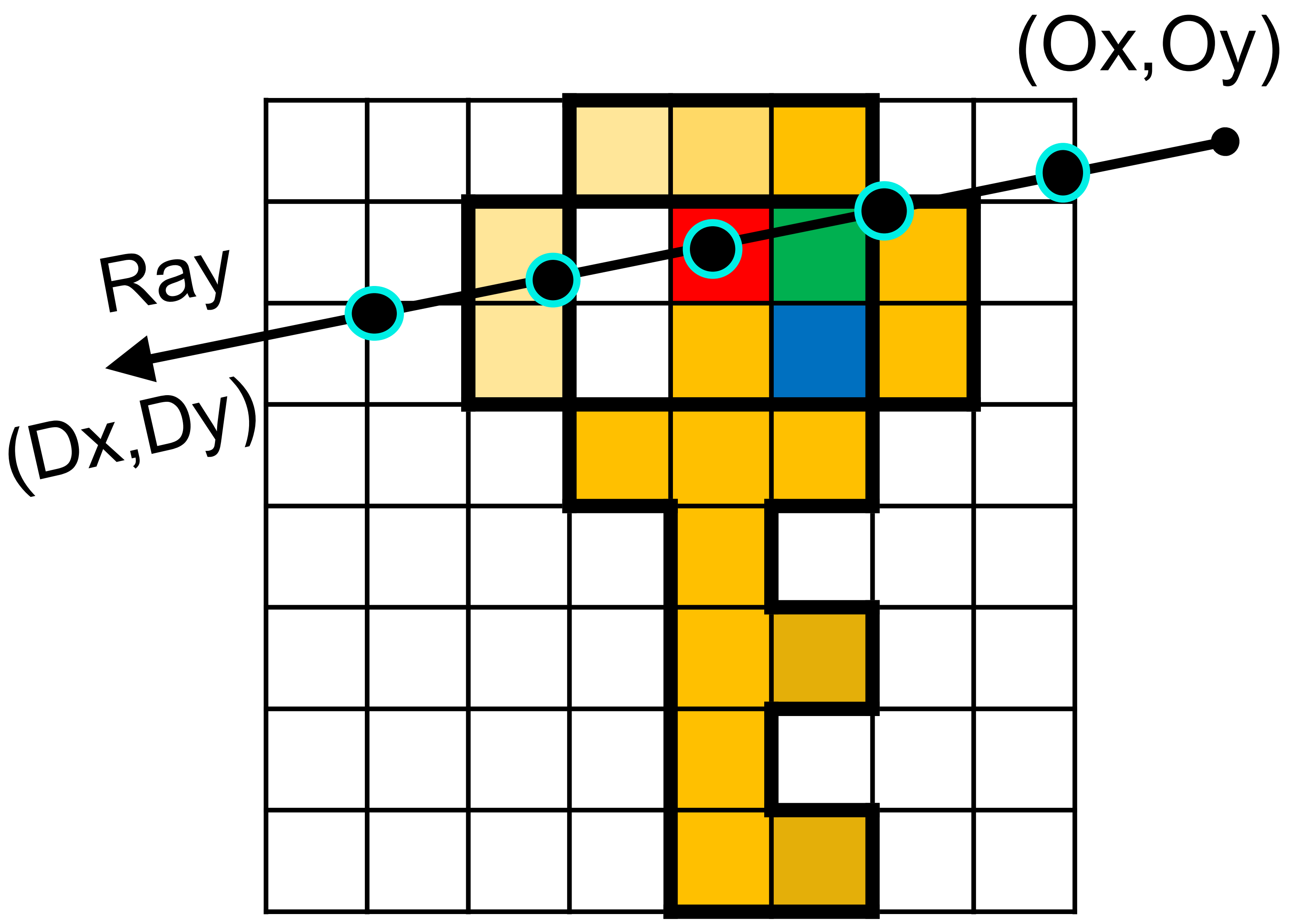
Code Fast

Run Fast

Case Study : Neural Radiance Field

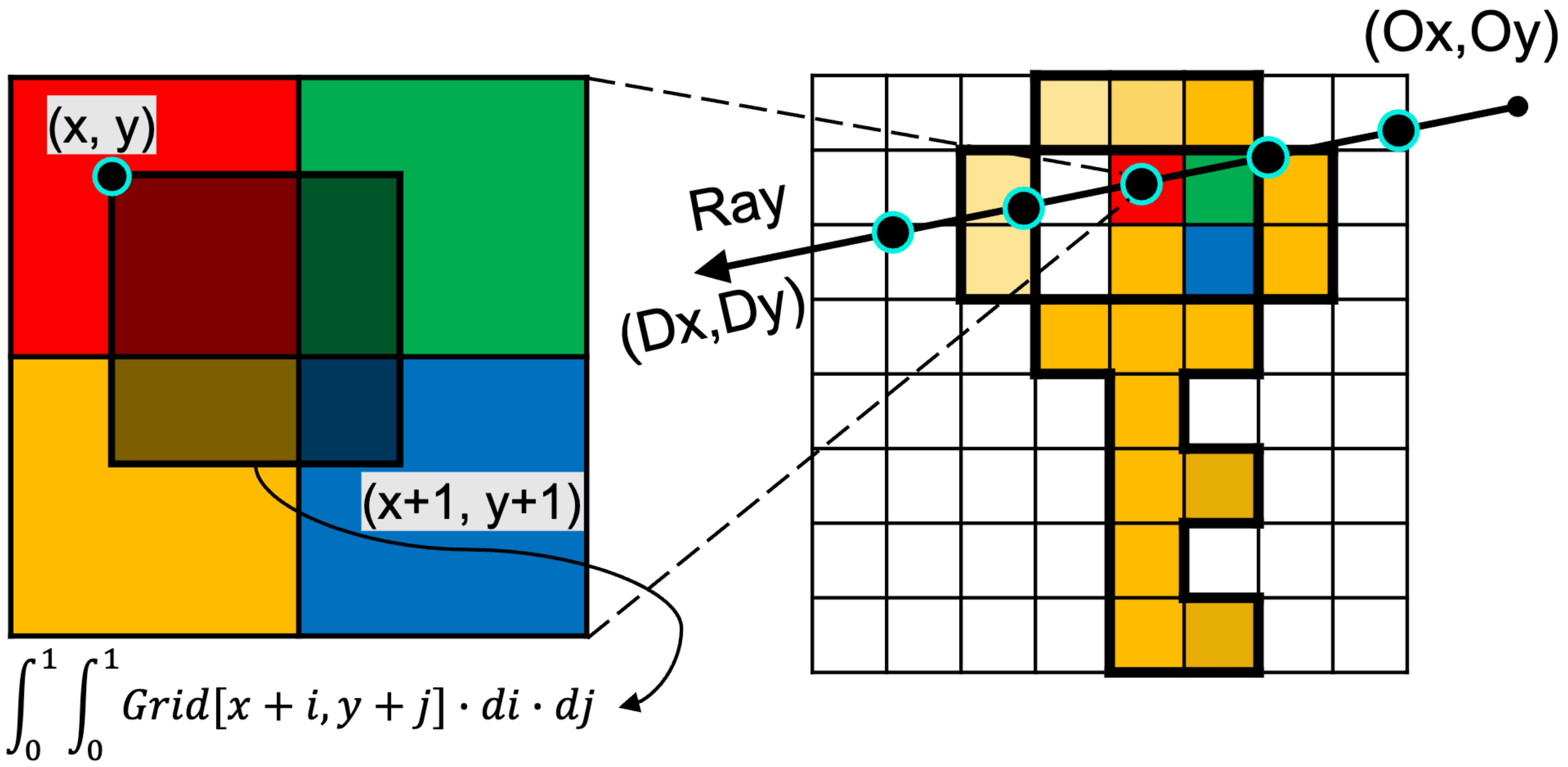


Case Study : Nerf

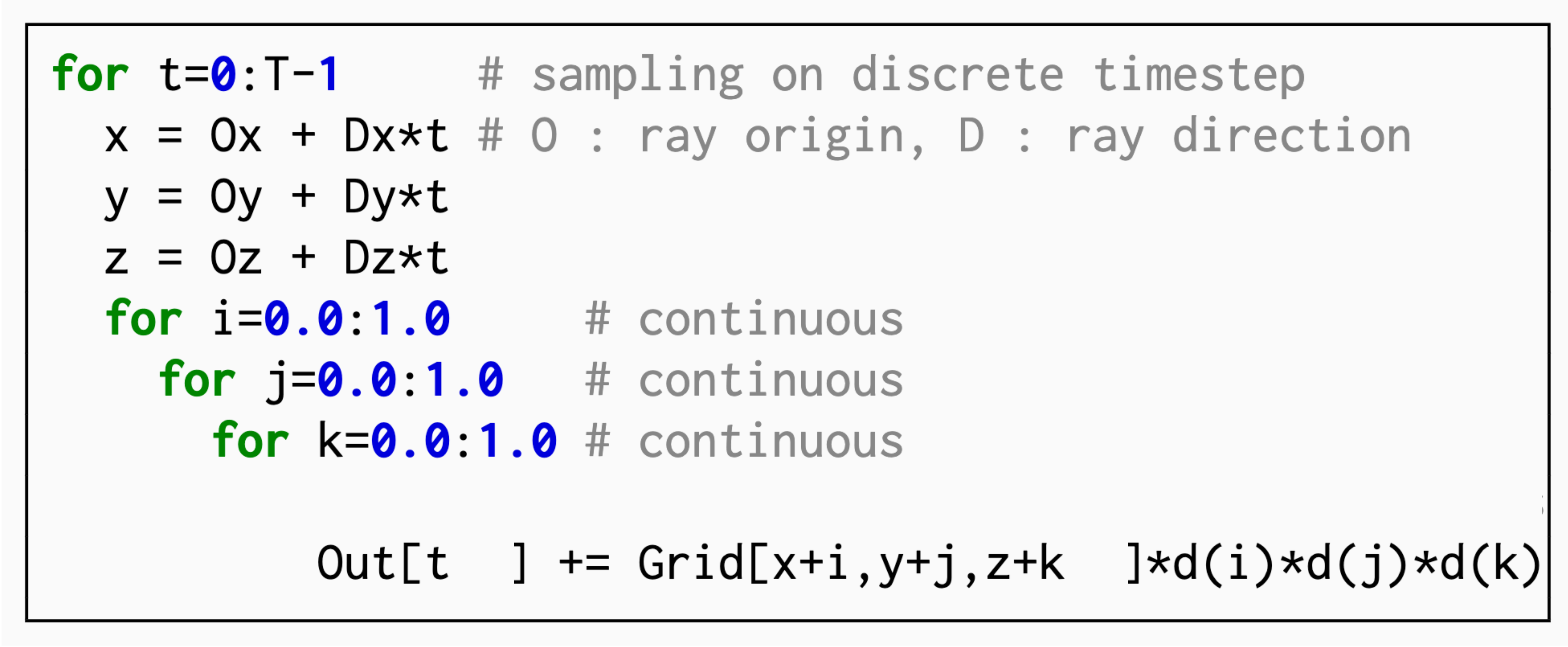


```
for t=0:T-1     # sampling on discrete timestep
  x = Ox + Dx*t # O : ray origin, D : ray direction
  y = Oy + Dy*t
  z = Oz + Dz*t
```

Case Study : Nerf



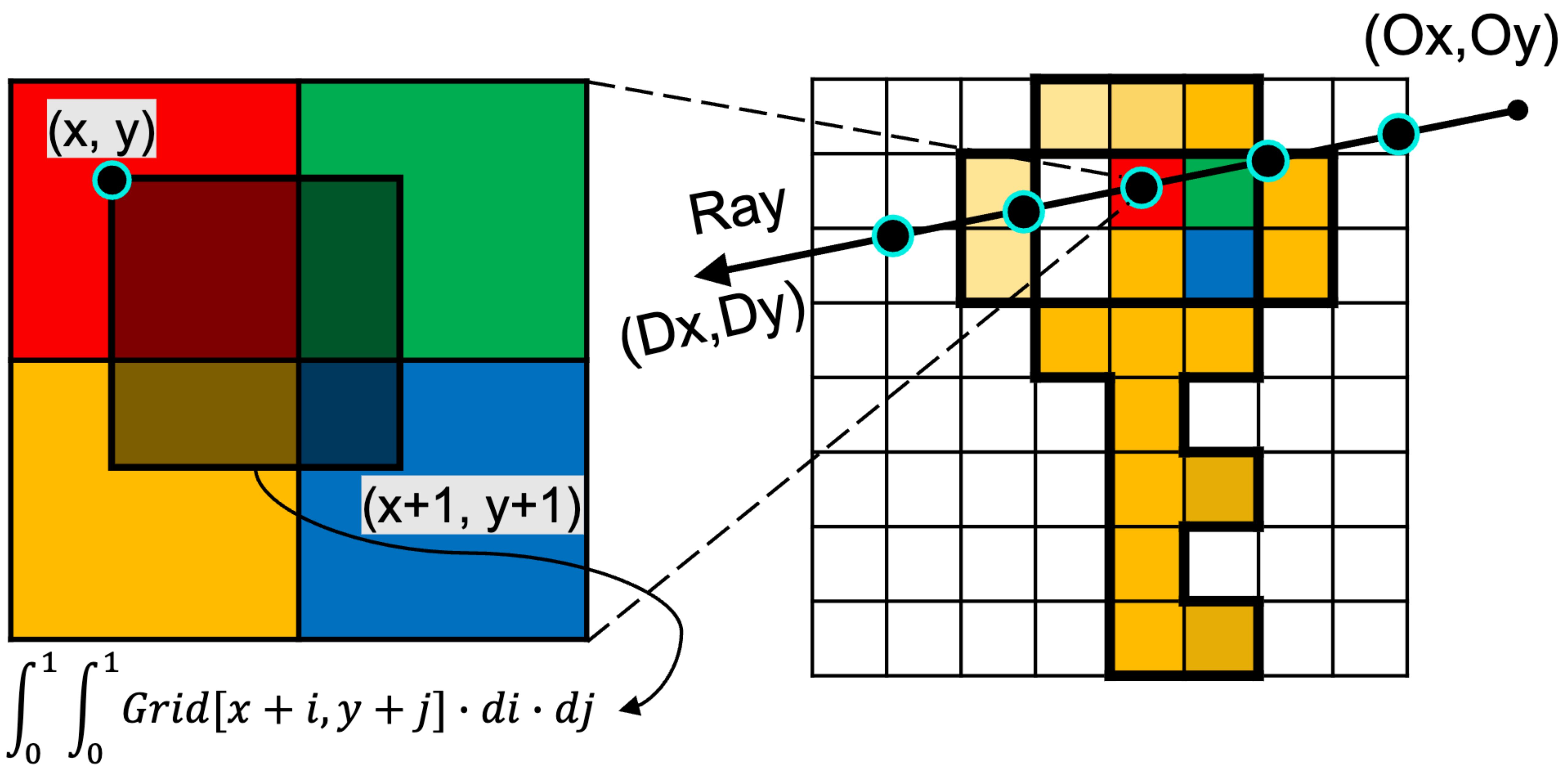
$$\int_0^1 \int_0^1 Grid[x+i, y+j] \cdot di \cdot dj$$



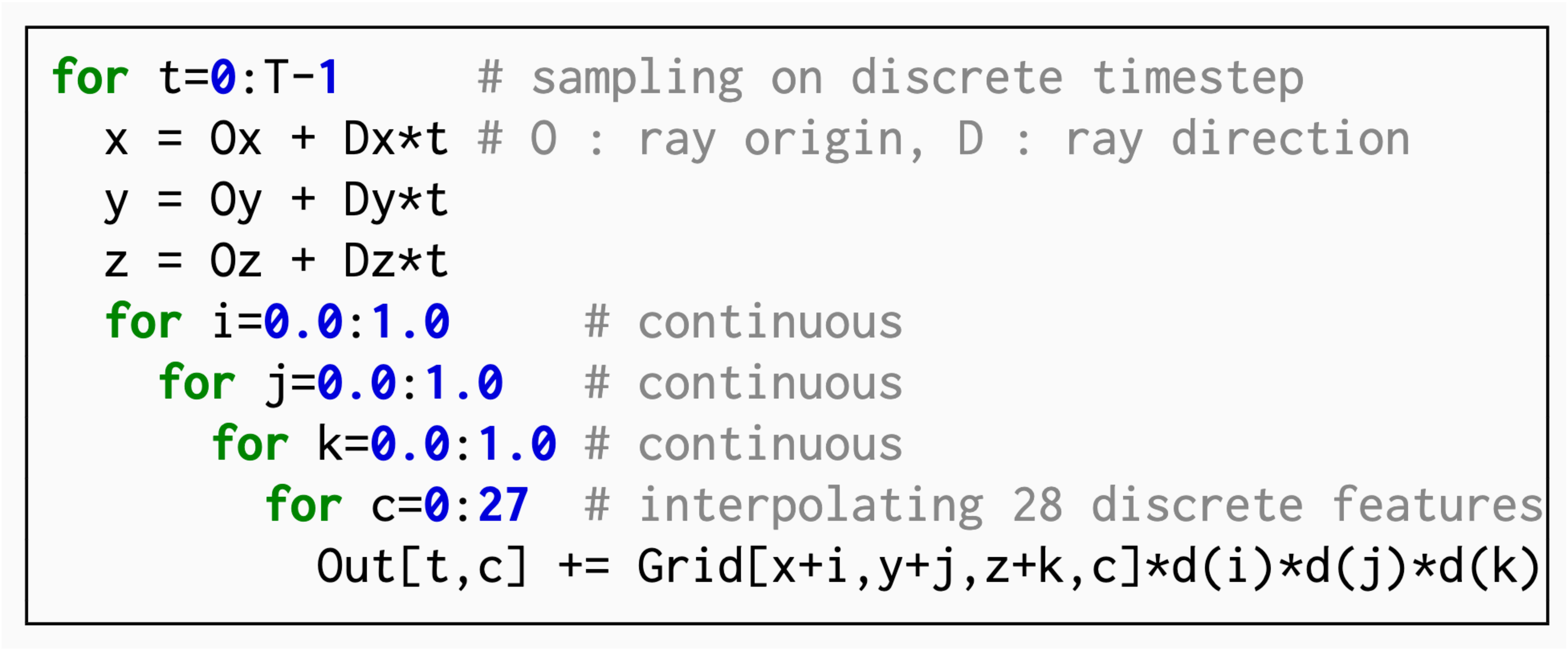
```
for t=0:T-1      # sampling on discrete timestep
  x = Ox + Dx*t # O : ray origin, D : ray direction
  y = Oy + Dy*t
  z = Oz + Dz*t
  for i=0.0:1.0     # continuous
    for j=0.0:1.0   # continuous
      for k=0.0:1.0 # continuous

          Out[t  ] += Grid[x+i,y+j,z+k  ]*d(i)*d(j)*d(k)
```

Compute interpolation on every sampled point in ray

Case Study : NeRF



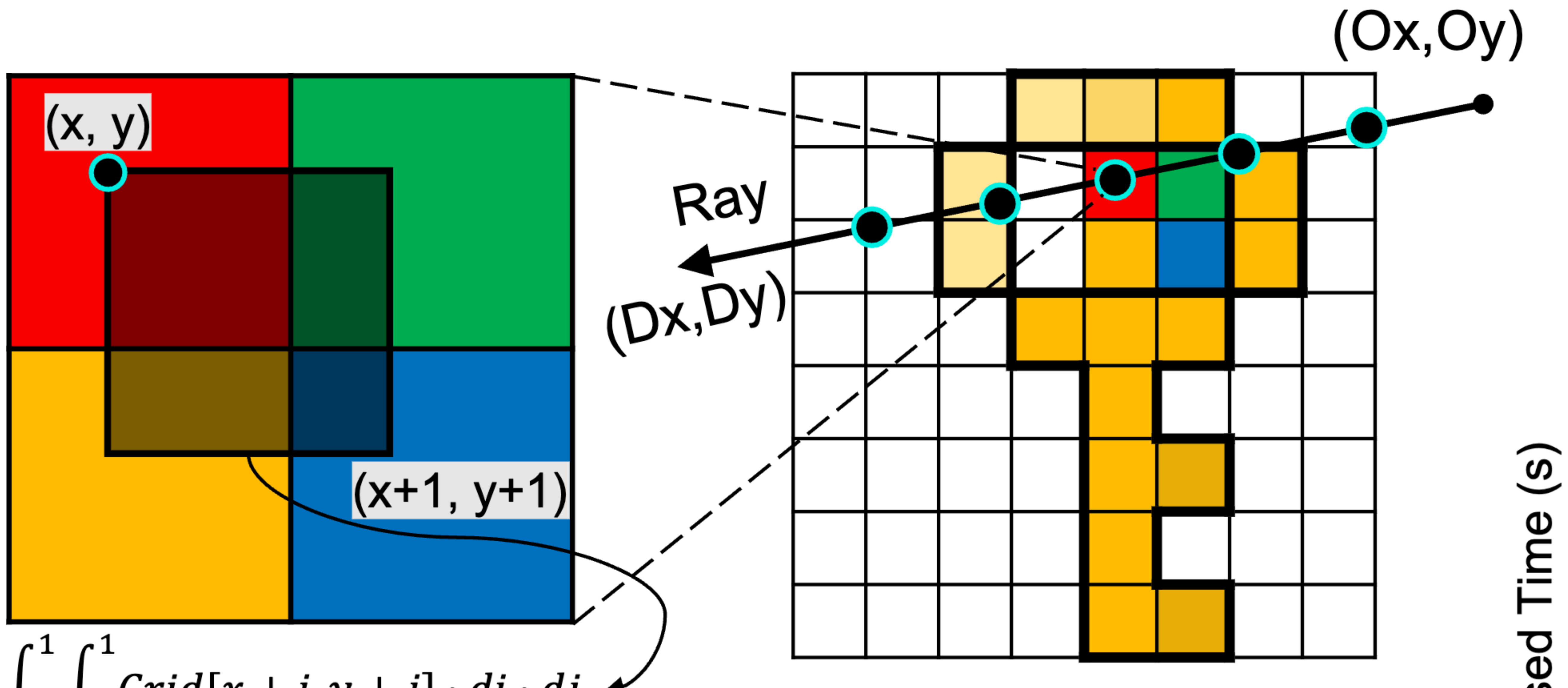
$$\int_0^1 \int_0^1 Grid[x+i, y+j] \cdot di \cdot dj$$



```
for t=0:T-1     # sampling on discrete timestep
  x = Ox + Dx*t # O : ray origin, D : ray direction
  y = Oy + Dy*t
  z = Oz + Dz*t
  for i=0.0:1.0     # continuous
    for j=0.0:1.0   # continuous
      for k=0.0:1.0 # continuous
        for c=0:27  # interpolating 28 discrete features
          Out[t,c] += Grid[x+i,y+j,z+k,c]*d(i)*d(j)*d(k)
```

9Lines vs. 82Lines (PyTorch)

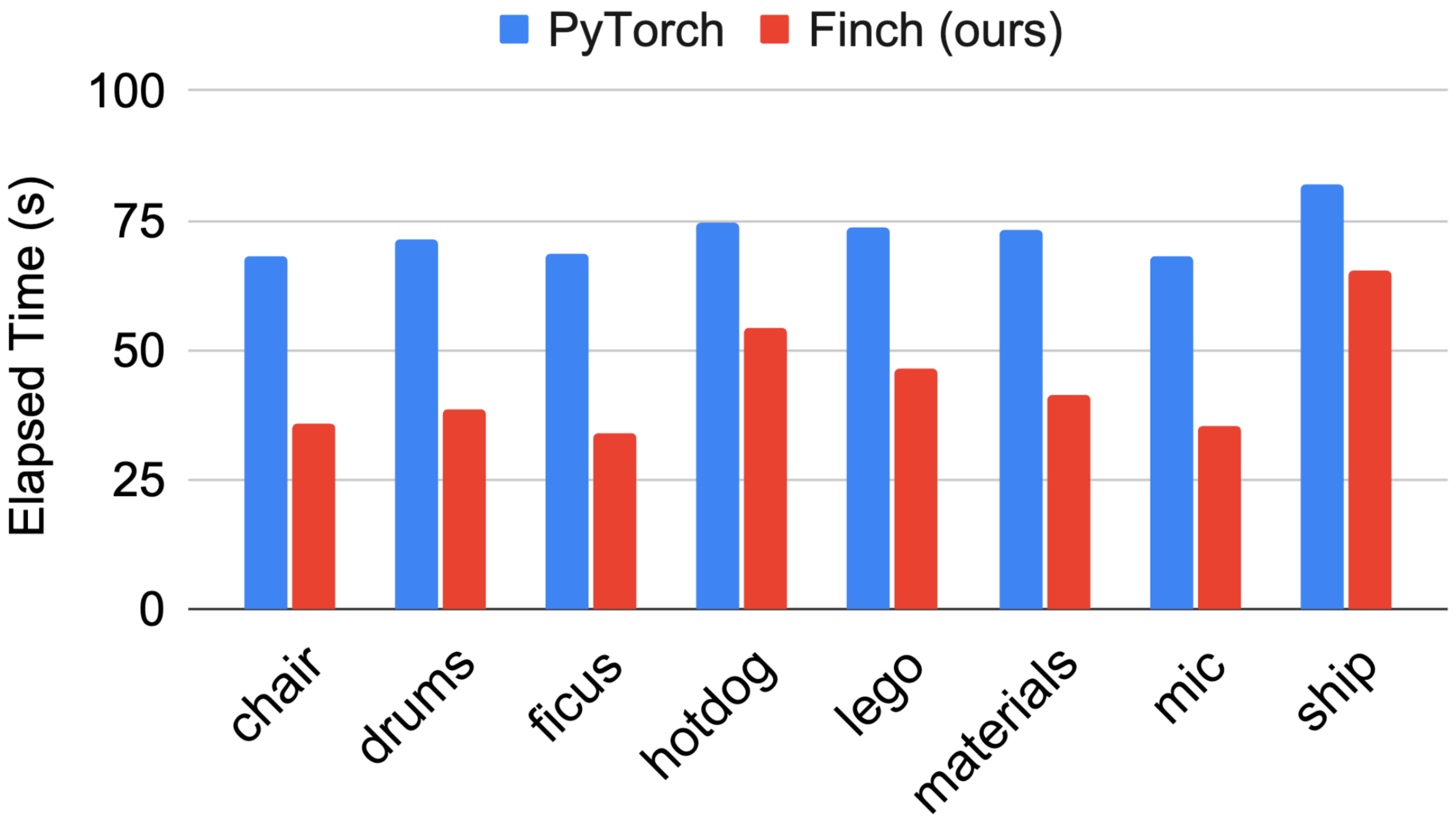
Case Study : Nerf



$$\int_0^1 \int_0^1 Grid[x+i, y+j] \cdot di \cdot dj$$

```
for t=0:T-1     # sampling on discrete timestep
  x = Ox + Dx*t # O : ray origin, D : ray direction
  y = Oy + Dy*t
  z = Oz + Dz*t
  for i=0.0:1.0     # continuous
    for j=0.0:1.0   # continuous
      for k=0.0:1.0 # continuous
        for c=0:27  # interpolating 28 discrete features
          Out[t,c] += Grid[x+i,y+j,z+k,c]*d(i)*d(j)*d(k)
```

9Lines vs. 82Lines (PyTorch)



Single threaded CPU

Hardware For Sparsity

Hardware For Sparsity

- Most sparse data never gets used in the computation (eg: SpMSpV)
  - But they all travel through most of the pipeline
  - Opportunities for near-memory filtering

Hardware For Sparsity

- Most sparse data never gets used in the computation (eg: SpMSpV)
  - But they all travel through most of the pipeline
  - Opportunities for near-memory filtering
- Structured formats have a lot of exploitable patterns
  - But machines don’t understand the formats, everything is just bits
    - Nvidia Ampere understands a single sparse format
  - Opportunities to compress storage and simple near-memory operations

Hardware For Sparsity

- Most sparse data never gets used in the computation (eg: SpMSpV)
  - But they all travel through most of the pipeline
  - Opportunities for near-memory filtering
- Structured formats have a lot of exploitable patterns
  - But machines don't understand the formats, everything is just bits
    - Nvidia Ampere understands a single sparse format
  - Opportunities to compress storage and simple near-memory operations
- Compound operations have huge benefits (eg: SDDMM)
  - Doing simple binary operations (mem → op → mem) can be asymptotically bad
  - Opportunities for doing custom compound operations

Hardware For Sparsity

- Most sparse data never gets used in the computation (eg: SpMSpV)
  - But they all travel through most of the pipeline
  - Opportunities for near-memory filtering
- Structured formats have a lot of exploitable patterns
  - But machines don’t understand the formats, everything is just bits
    - Nvidia Ampere understands a single sparse format
  - Opportunities to compress storage and simple near-memory operations
- Compound operations have huge benefits (eg: SDDMM)
  - Doing simple binary operations (mem → op → mem) can be asymptotically bad
  - Opportunities for doing custom compound operations
- Sparse-aware hardware can have a high impact

Hardware For Sparsity

Algorithm	***Name***	***Authors***	***Format***	***Dataflow***	***Platform***
SpMV	MergeSpMV	CMU	CSR	Tiled Rowmajor	FPGA / ASIC
	FPGASpMV	Univ.Florida / Microsoft	CSR Variant	Rowmajor	FPGA
SpMSpM	SIGMA	Georgia Tech, Intel	Bitmap	Inner product	ASIC
	OuterSPACE	Michigan, Arizona state	(CSC,CSR)	Outer product	ASIC
	GAMMA	MIT	(CSR,CSR)	Rowmajor (Gustavson)	ASIC
SpMM	NVIDIA Sparse Tensor Core	Nvidia	Structured CSR	?	GPU
Sparse Convolution	SCNN	Nvidia, MIT, Berkeley, Stanford	CSF	Input Stationary	ASIC
	FPGASpConv	Zhejiang University, USC	Tiled CSF	Tiled Output Stationary	FPGA
Sparse Transformer	Sanger	Peking University	Blocked CSR	Fused	ASIC
Intersection	AVX512 VP2INTERSECT	Intel	Bitmap	?	CPU
	SSE4.2	Intel	Compressed	?	CPU
	ExTensor	UIUC, Nvidia	Various Formats	Tiled Innerproduct	ASIC
Einsum (Compiled)	SAM	Stanford, MIT	Various Formats	Various Dataflows	ASIC

Sparse Array Programming in the Python Ecosystem

Sparse Array Programming in the Python Ecosystem

Sparse Array Programming in the Python Ecosystem

NumPy
python™

Sparse Array Programming in the Python Ecosystem

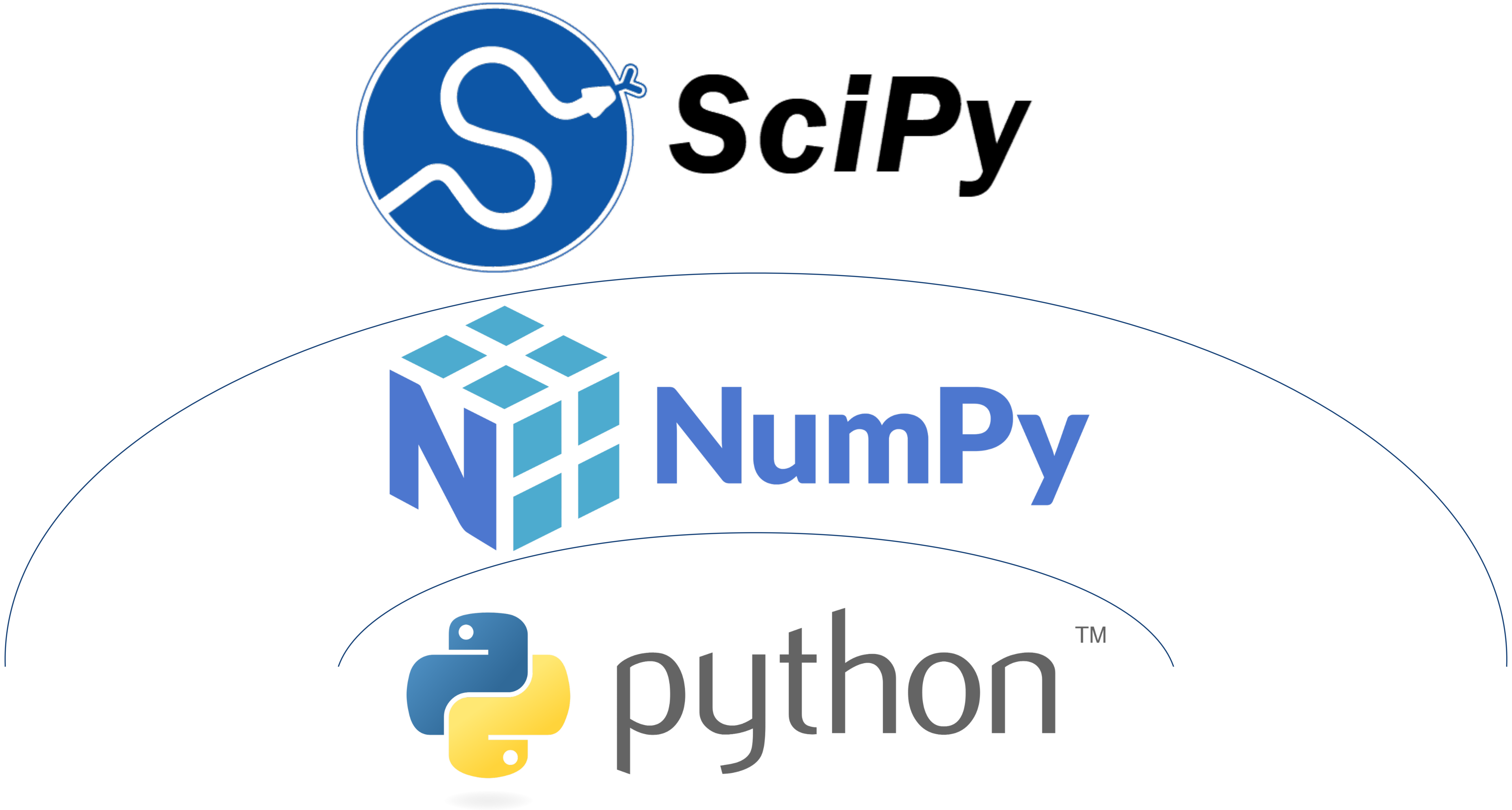
SciPy
NumPy
python™

Sparse Array Programming in the Python Ecosystem

Estimated User Base

6-15 million

10-25 million

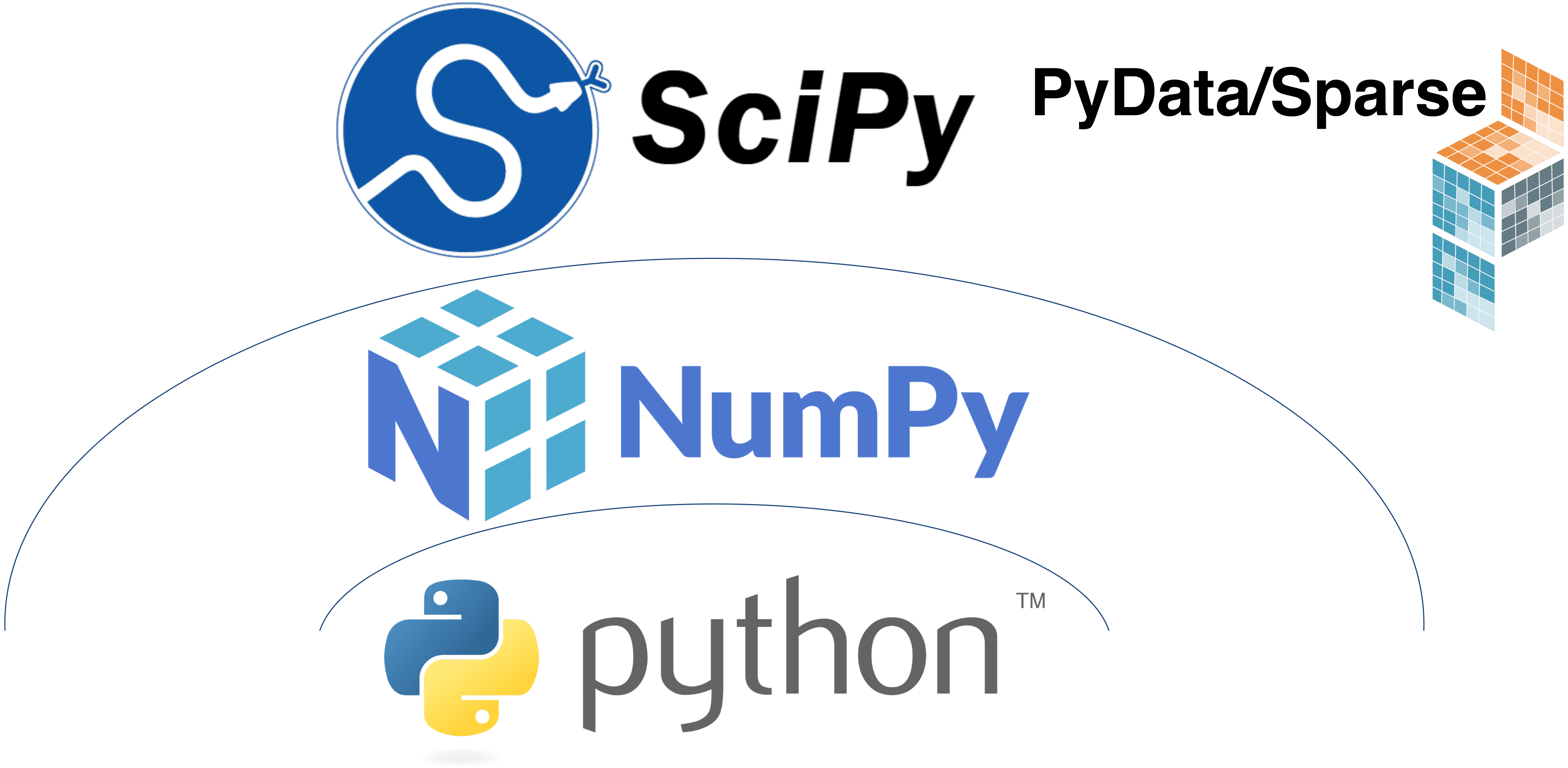


Sparse Array Programming in the Python Ecosystem

Estimated User Base

6-15 million

10-25 million



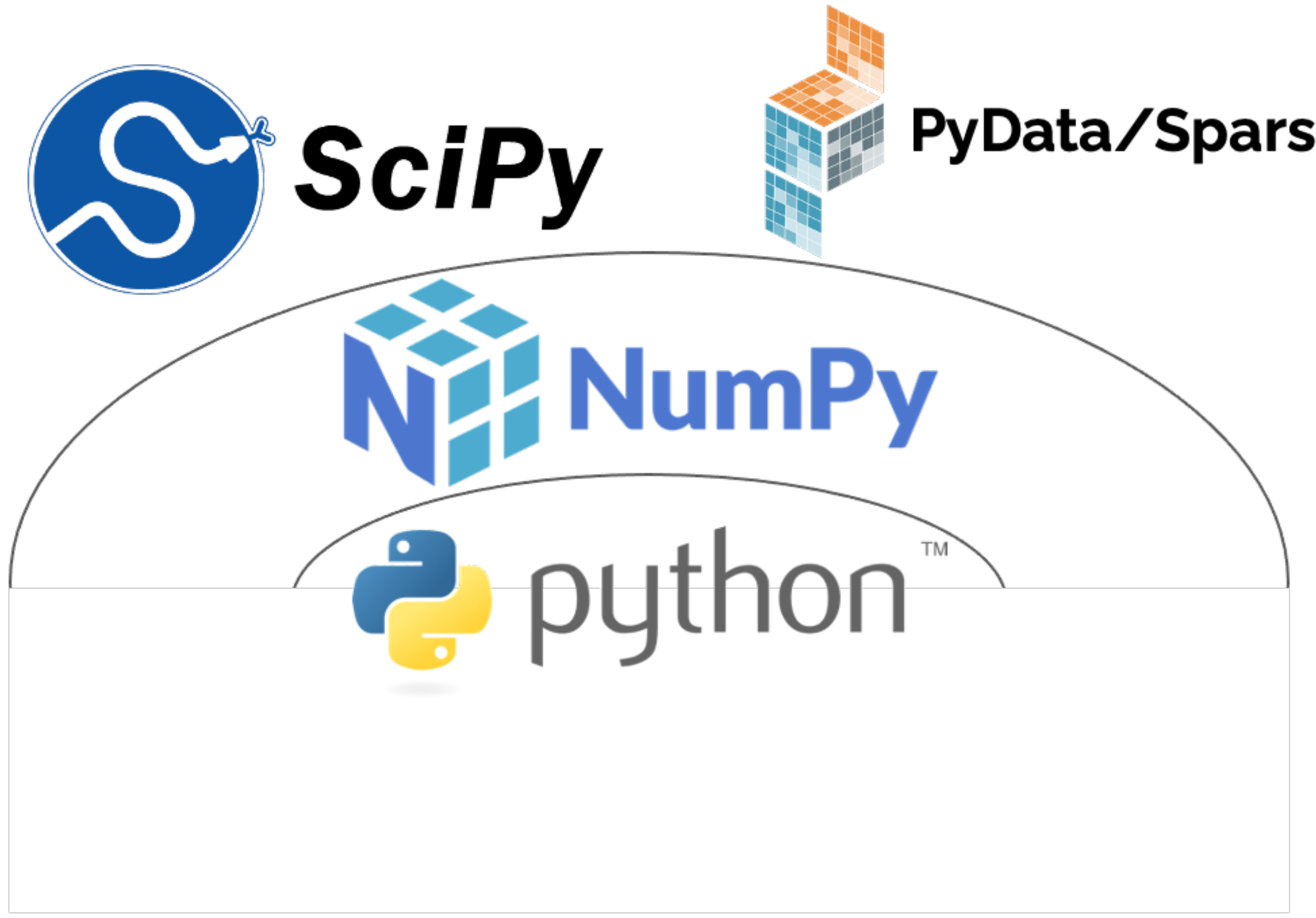
SciPy

PyData/Sparse

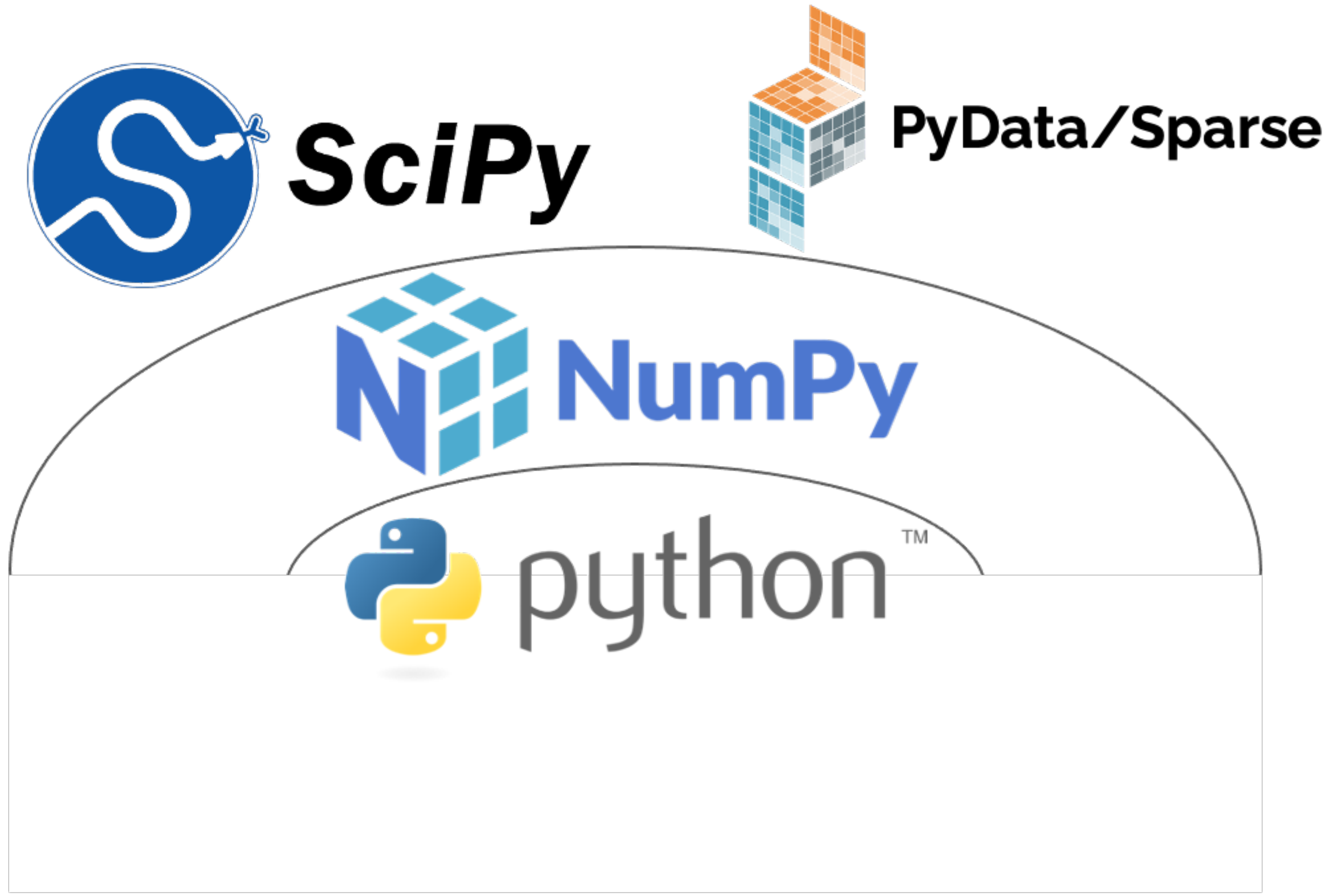
NumPy

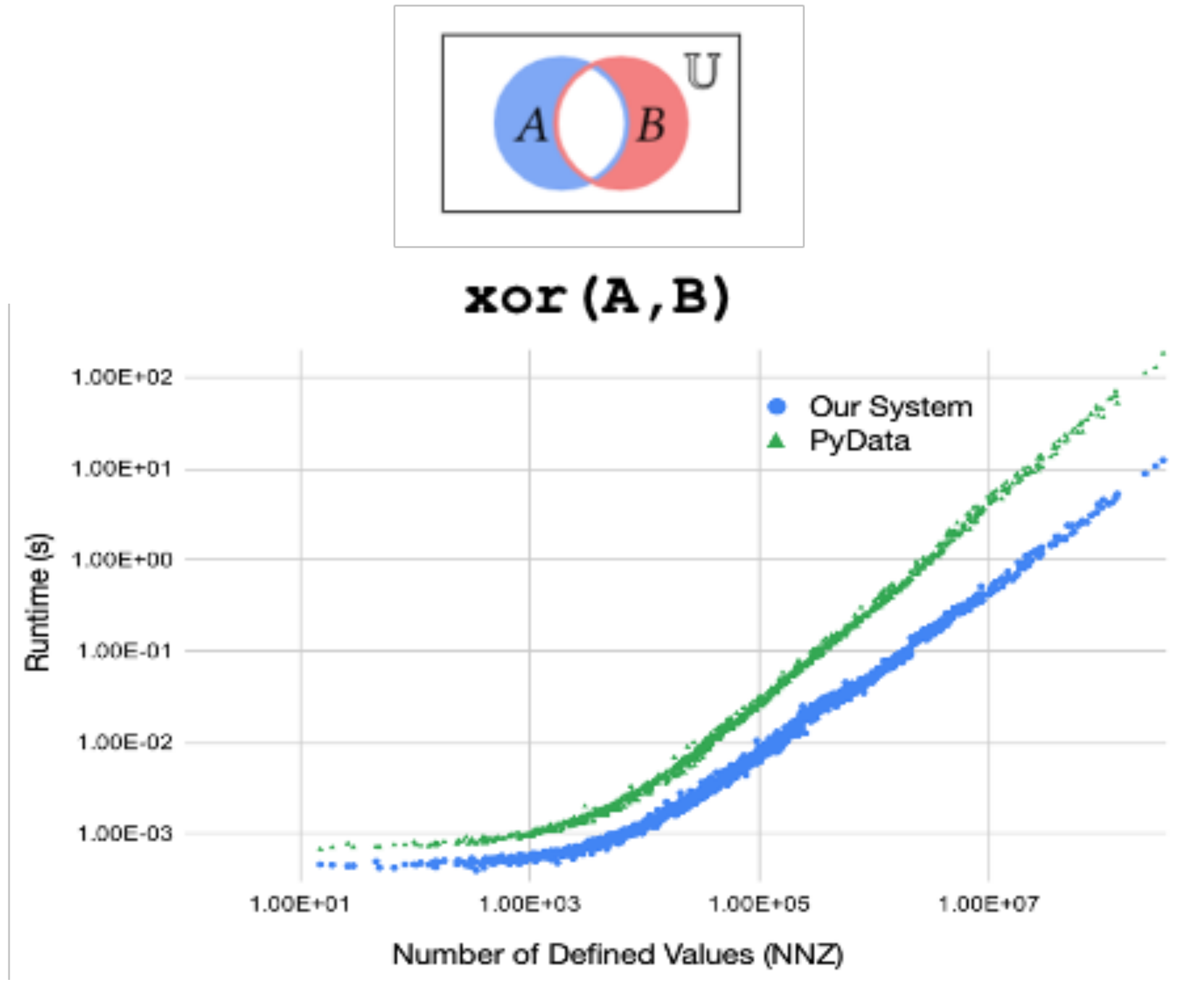
python™

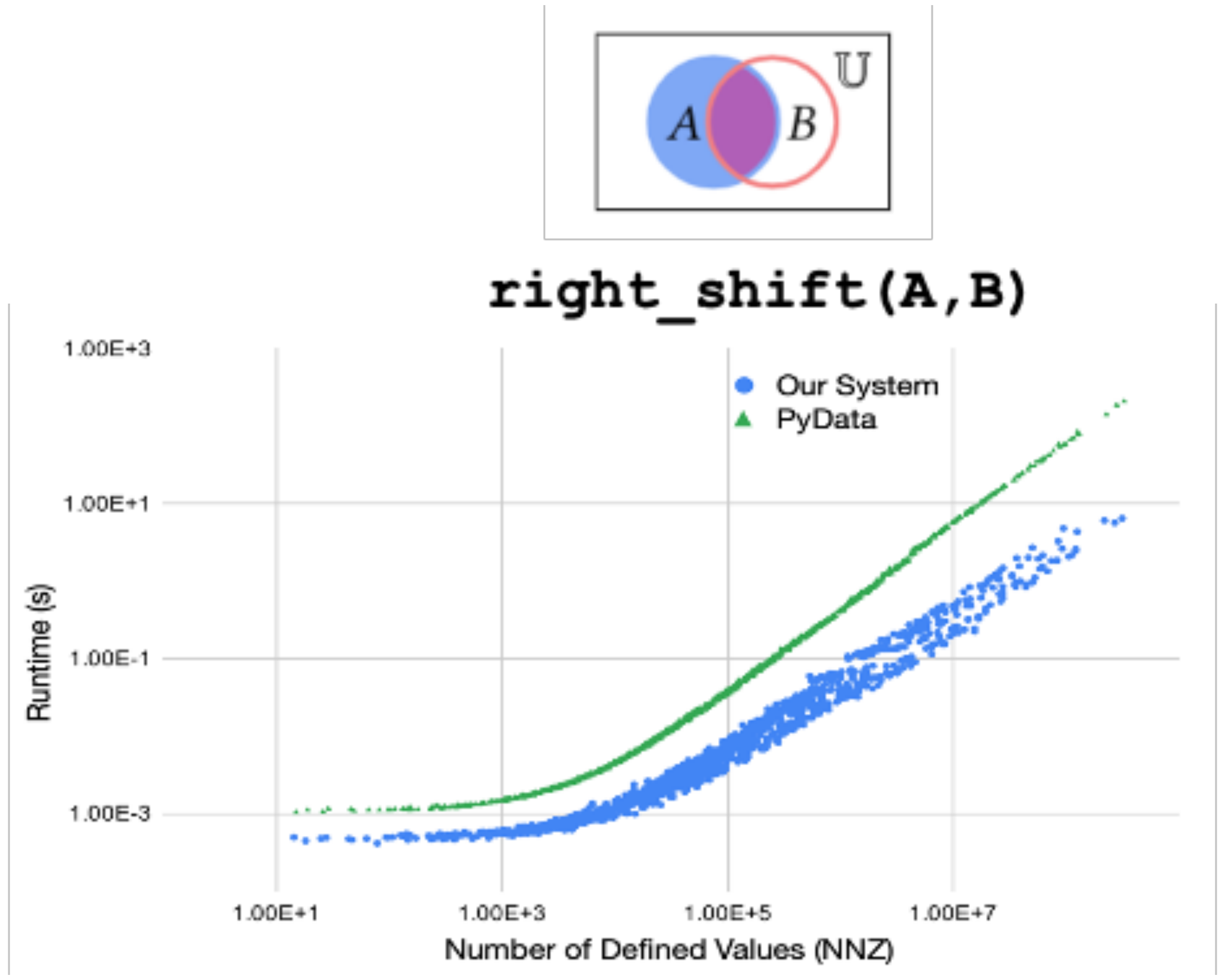
Speeding up Sparse Array Programming in the Python Ecosystem

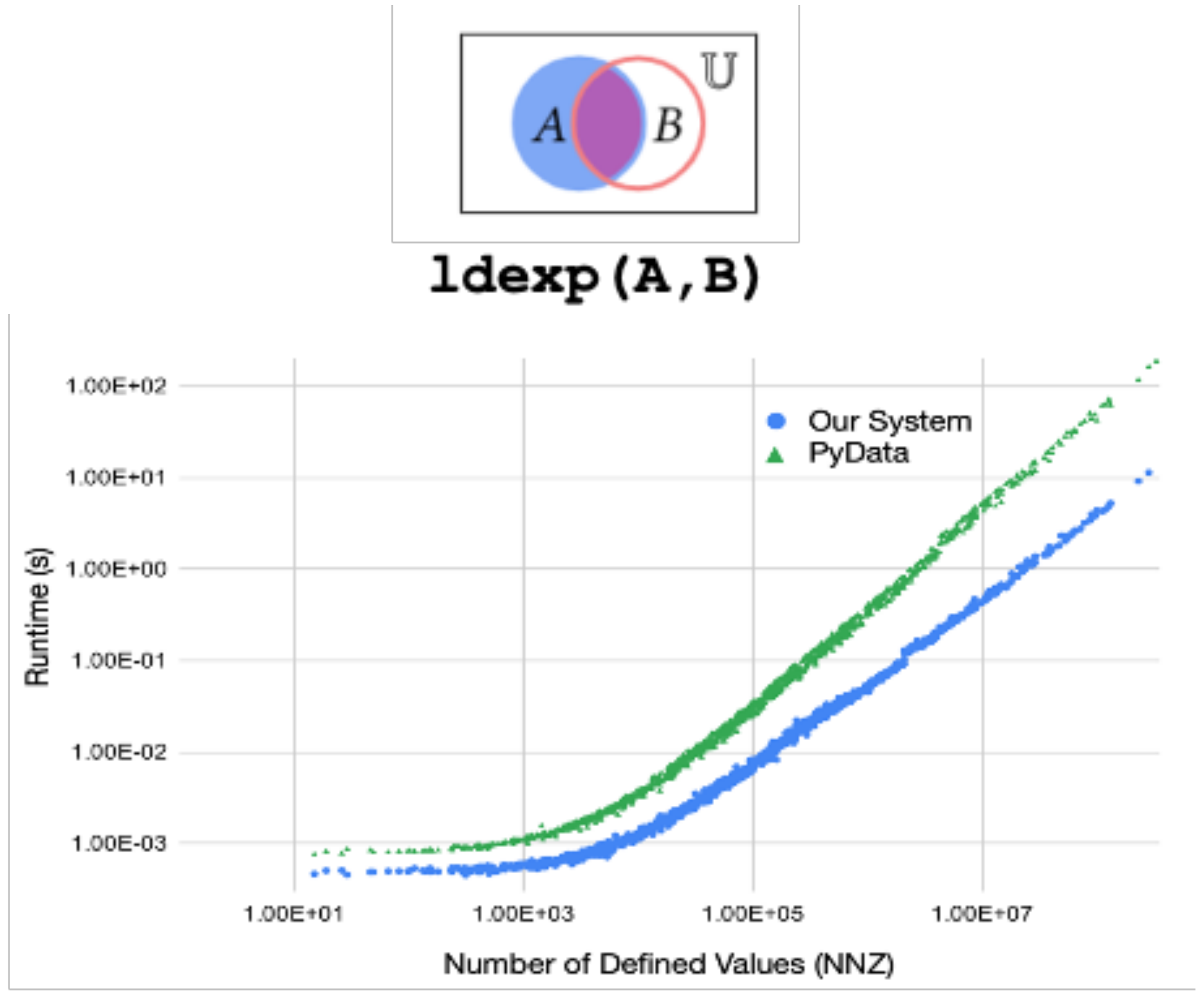


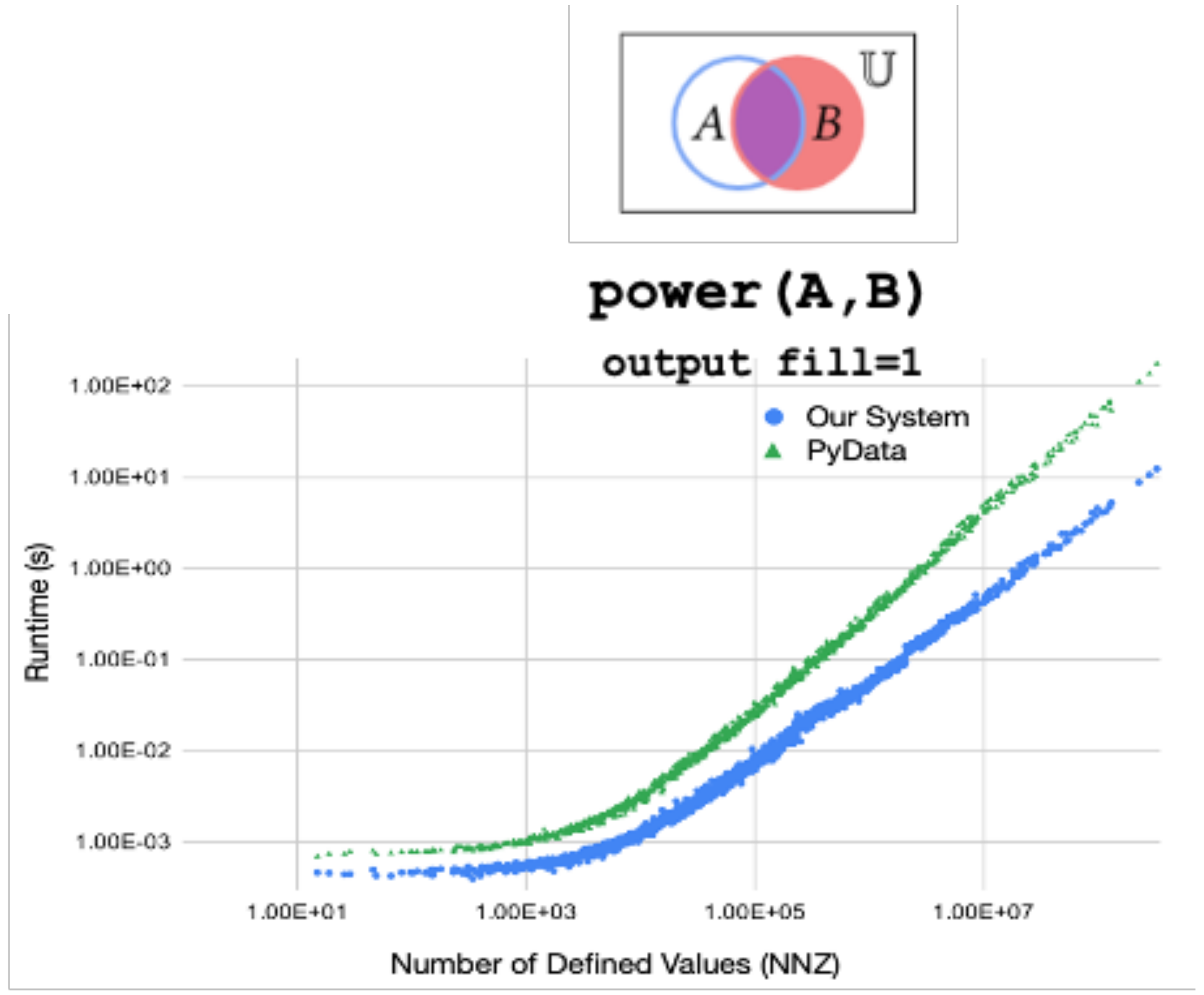
Speeding up Sparse Array Programming in the Python Ecosystem




xor(A,B)


right_shift(A,B)


ldexp(A,B)


power(A,B)

output fill=1

Conclusion

Conclusion

- Array Abstraction is:
  - Simple
  - In most imperative languages
  - Highest performance
  - Familiar to most programmers
  - Used a lot (and in almost all high-performance code)

Conclusion

- Array Abstraction is:
  - Simple
  - In most imperative languages
  - Highest performance
  - Familiar to most programmers
  - Used a lot (and in almost all high-performance code)
- Expanding from a

  Multi-dimensional, Rectilinear, Dense, Integer grid of points to

  Multi-dimensional, Rectilinear, ~~Dense, Integer~~ grid of points

  Increases the application domain for array programming

Conclusion

- Array Abstraction is:
  - Simple
  - In most imperative languages
  - Highest performance
  - Familiar to most programmers
  - Used a lot (and in almost all high-performance code)
- Expanding from a

  Multi-dimensional, Rectilinear, Dense, Integer grid of points to

  Multi-dimensional, Rectilinear, ~~Dense, Integer~~ grid of points

  Increases the application domain for array programming
- Need compiler support
  - To provide the simple array abstraction while maintaining high performance

Commit Group

- Current & recent projects
  - Finch: A DSL for structured data
  - TACO: A DSL for sparse tensor algebra
  - Netblocks: A DSL Custom Network Protocols
  - SEQ: A DSL for bio informatics
  - GraphIt: A DSL for graph analysts
  - BuildIt: A Multistage programming framework in C++
  - CoLa: A DSL for data compression
  - SimIt: A DSL for sparse systems
  - MILK: A DSL for Optimizing indirect memory references
  - Cimple: A DSL for Instruction and Memory Level Parallelism
  - Codon: A Pythonic DSL framework
  - Tiramisu: A polyhedral compiler for data parallel algorithms
  - Ithemal: Performance prediction using machine learning
  - VeGen: Generating Vectorizers for vector instructions beyond SIMD
  - Vemal: Vectorization using machine learning
  - goSLP & Revec: Modernizing vectorization technology
  - OpenTuner: An extensible framework for program autotuning

Thank You

http://tensor-compiler.org/

This Work Supported By:

DARPA NSF