Jet:
A Language and Heterogeneous Compiler for Fluid Simulations

dan bailey

double negative visual effects

double negative visual effects

Double Negative

- Largest Visual Effects studio in Europe
- Offices in London and Singapore
- Large and growing R & D team

Simulation

double negative visual effects

Hair / Fur

Particles / Fluids

Crowds

Muscles

Cloth

Rigid Bodies

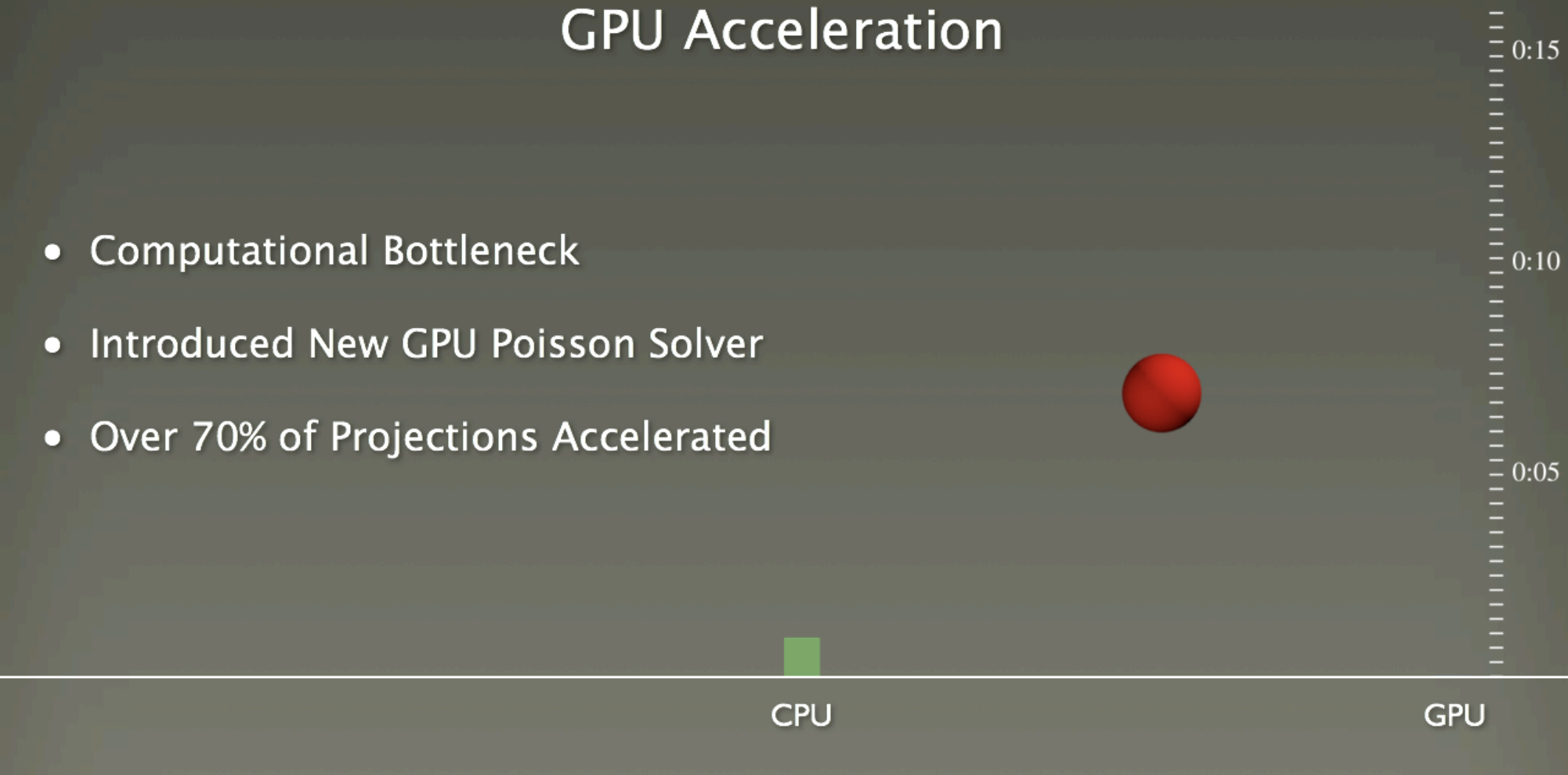
Squirt

- Proprietary Fluid Solver
- Plausibility over Accuracy
- Focus on Parallel Research

double negative visual effects

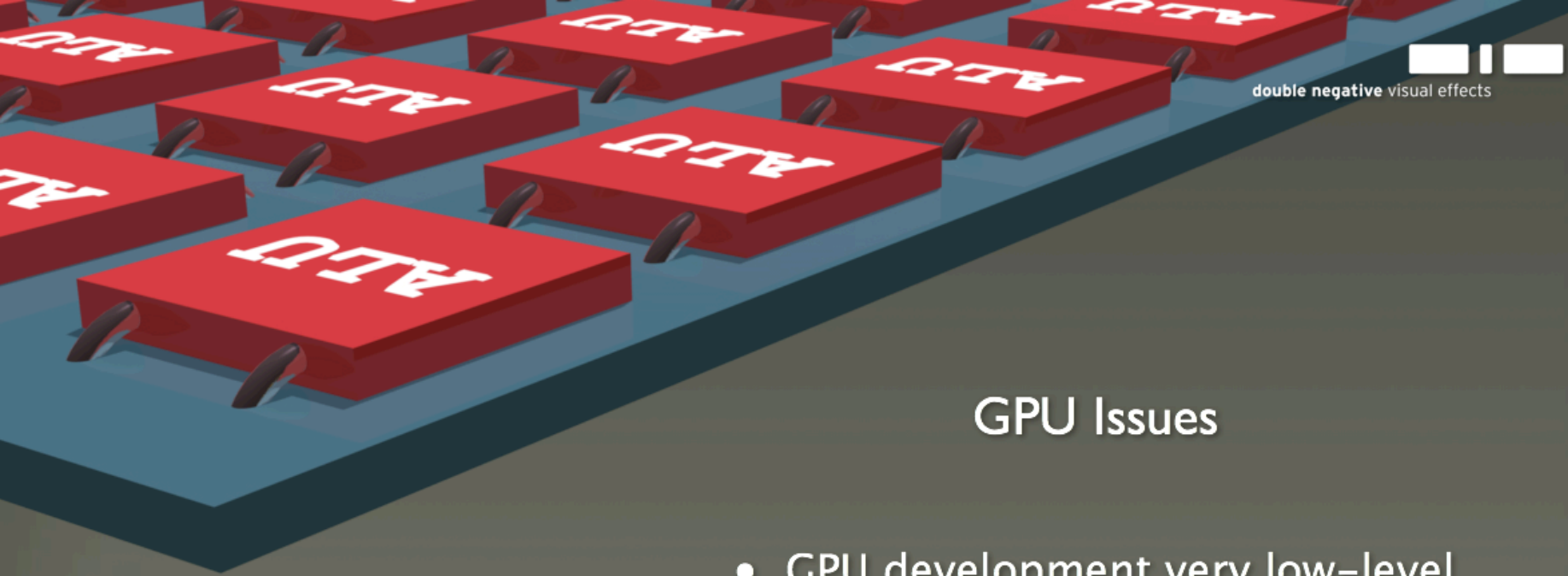
GPU Acceleration

- Computational Bottleneck
- Introduced New GPU Poisson Solver
- Over 70% of Projections Accelerated



Offloading Some Computation to GPU

- Succumb to Amdahl's Law
- Memory bandwidth limited
- GPU needs to do everything

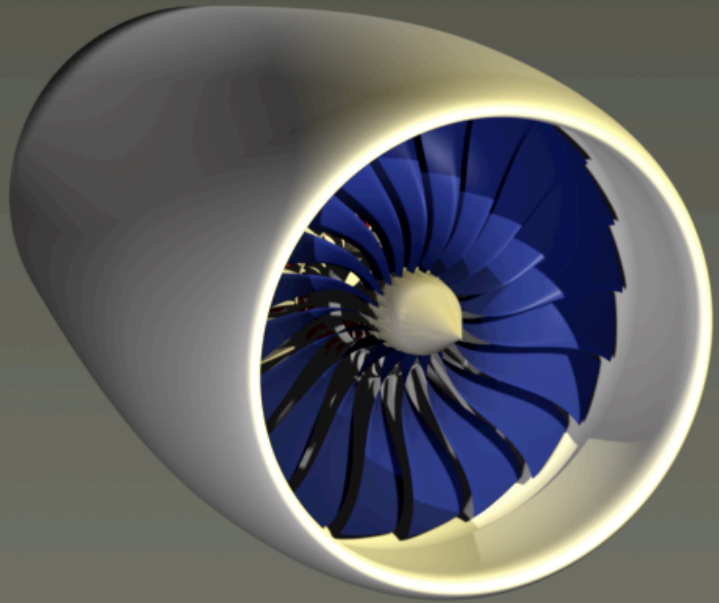


double negative visual effects

GPU Issues

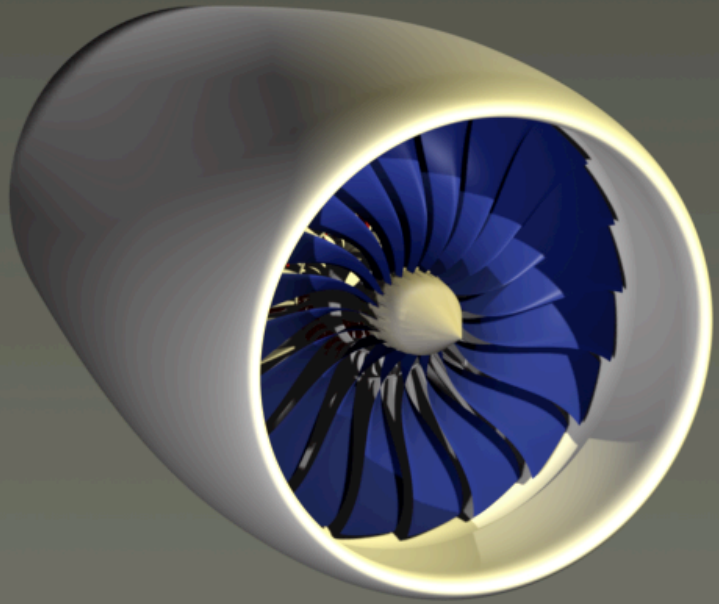
- GPU development very low-level
- Require dual GPU / CPU codebases
- Can only use NVidia hardware

Jet Language + Compiler



- Performance
- Productivity
- Portability

Jet Language

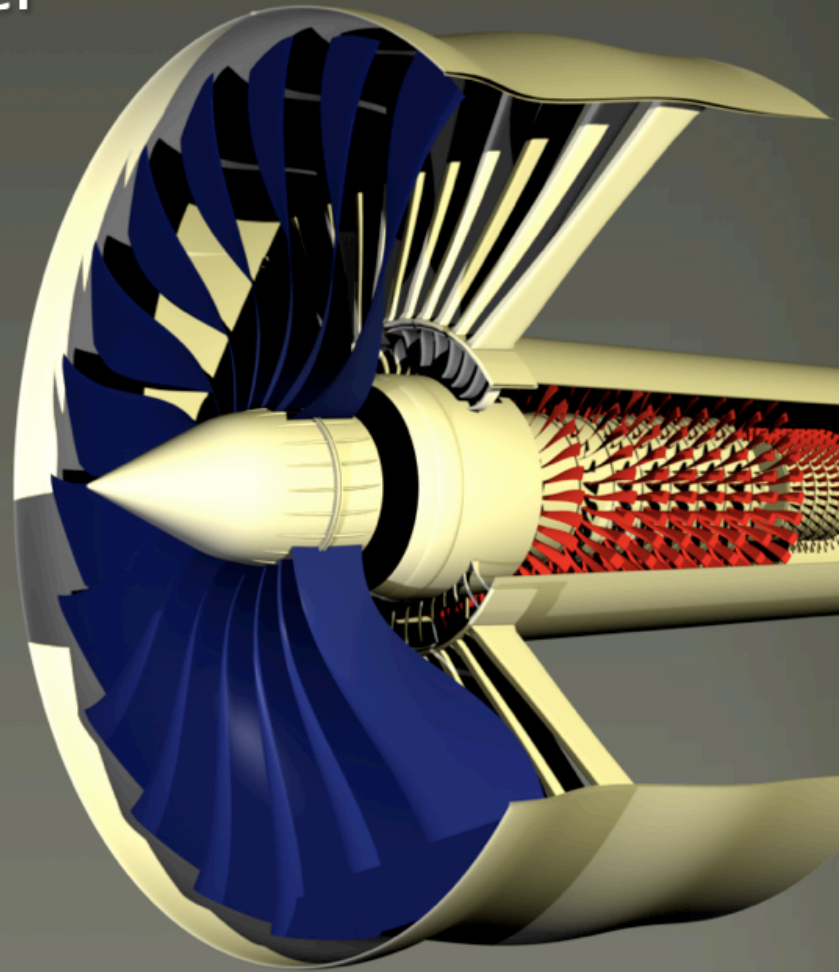


- DSL for simulation
- Expressive, high-level
- Restrictive, ‘atomic’ primitives:

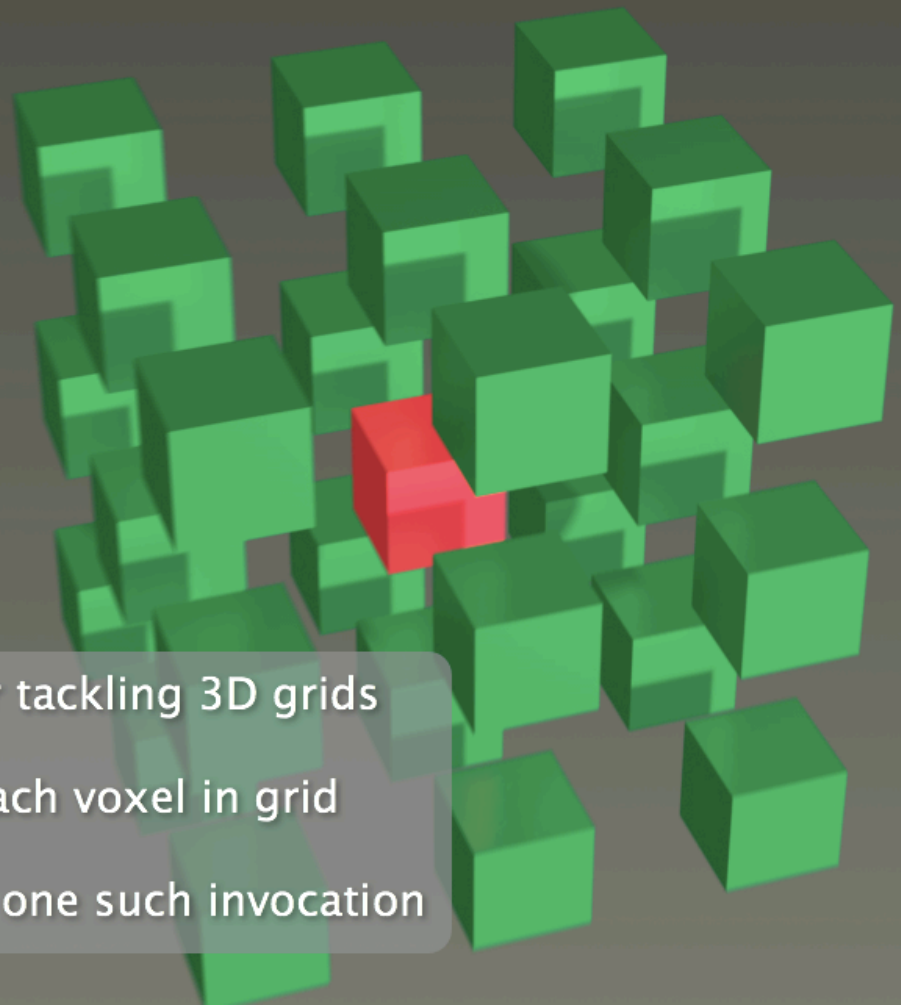
Jet Compiler

double negative visual effects

- Uses LLVM Infrastructure
- Re-usable Transformations
- Target Agnostic

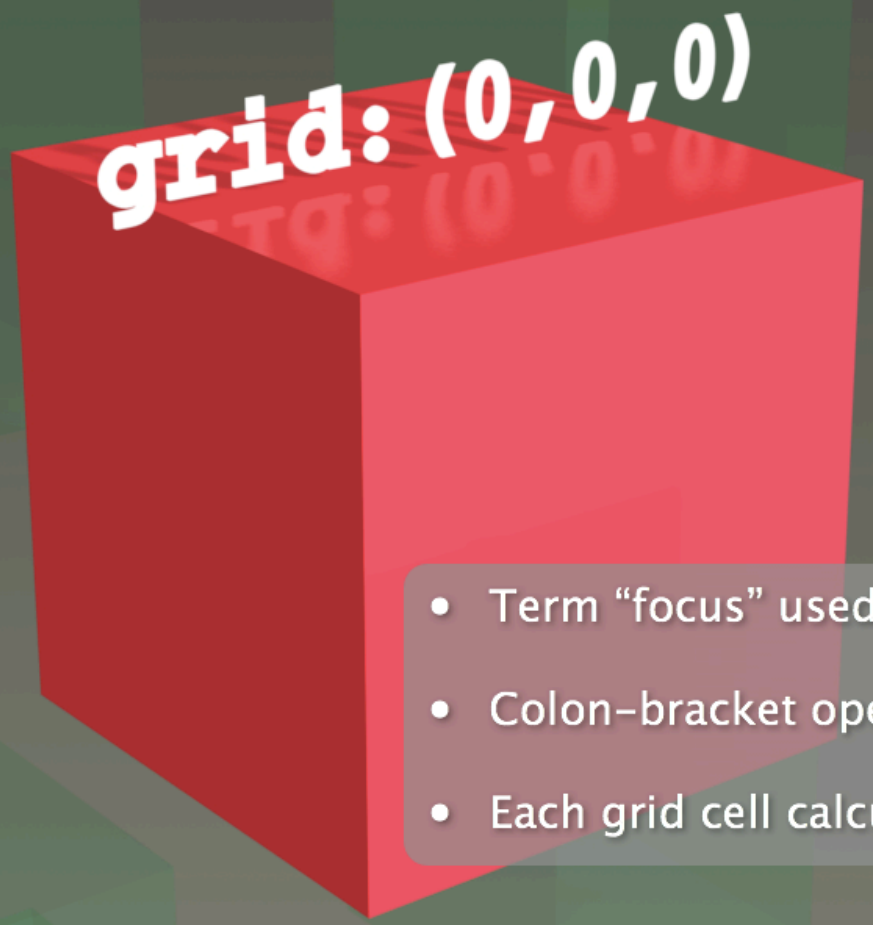


Voxel



- Voxel primitive is for tackling 3D grids
- Kernel invoked for each voxel in grid
- This example shows one such invocation

Voxel



- Term “focus” used for target voxel
- Colon-bracket operator introduced
- Each grid cell calculated independently

Voxel

double negative visual effects

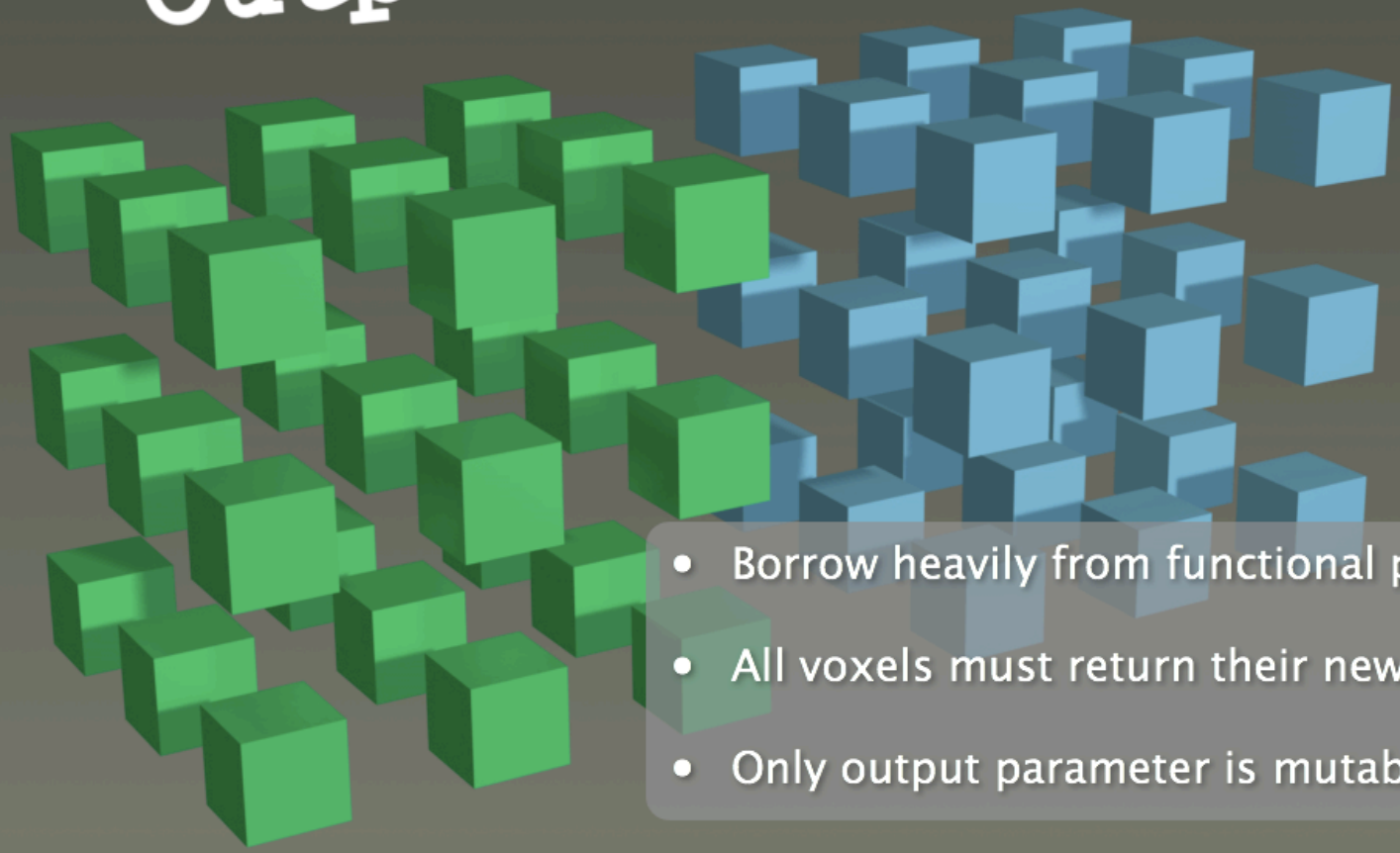
- Focus shown in red, neighbours in yellow
- All voxel offsets determined relatively
- Simplifies accessing cells and neighbours

Voxel

Output

Input

double negative visual effects

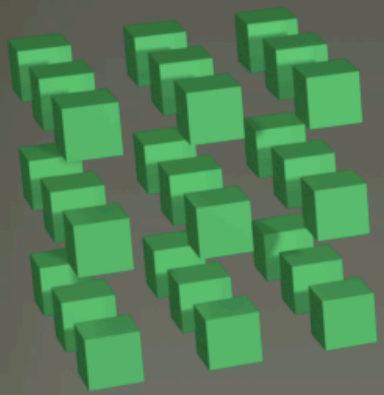


- Borrow heavily from functional programming
- All voxels must return their new value
- Only output parameter is mutable

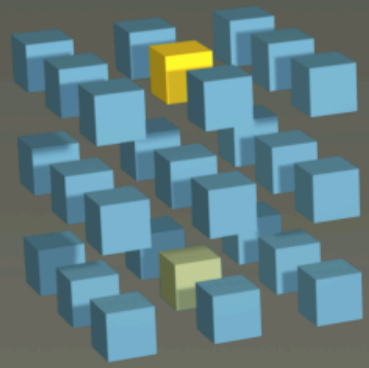
Voxel

double negative visual effects

Output

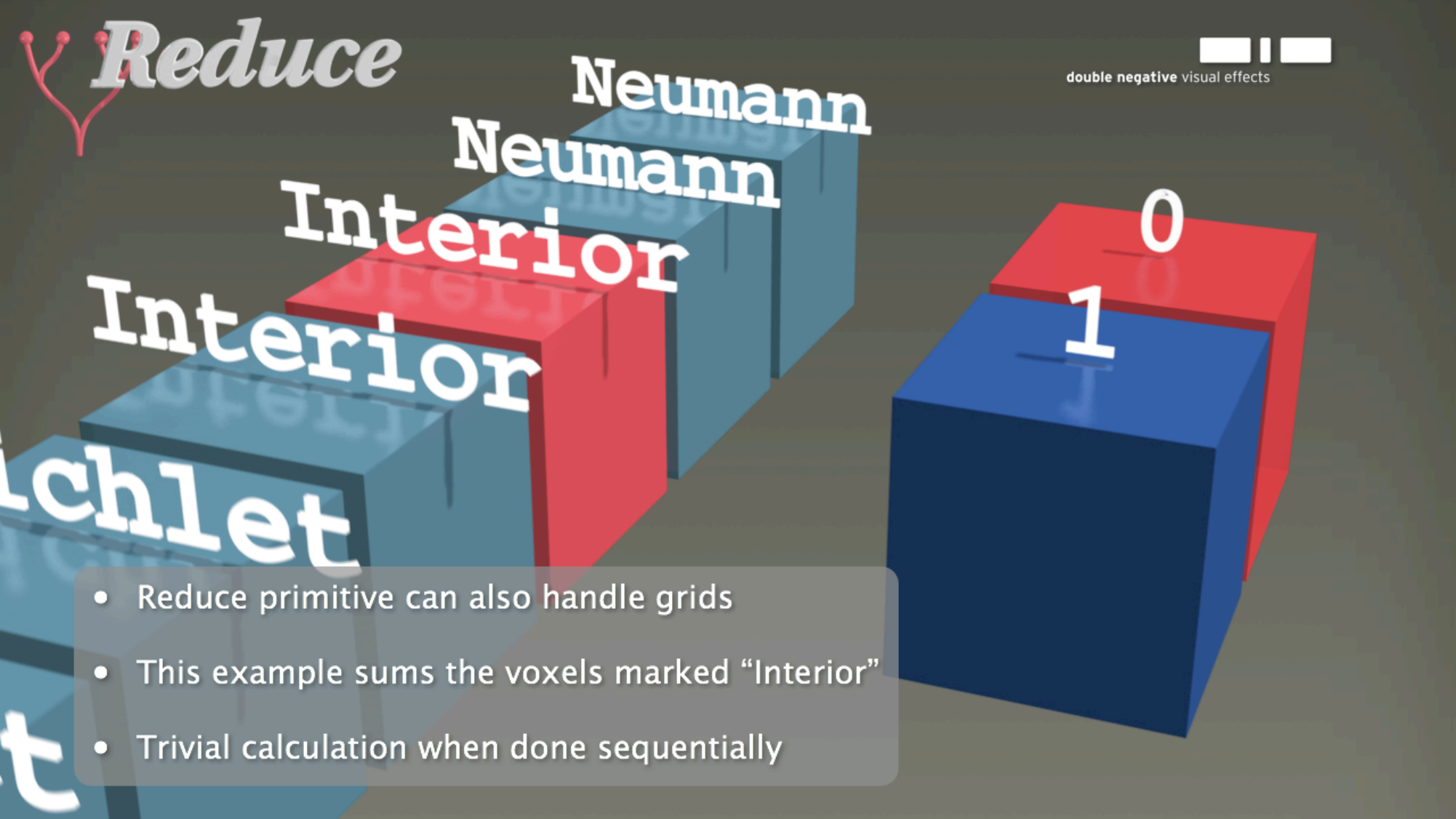


Input

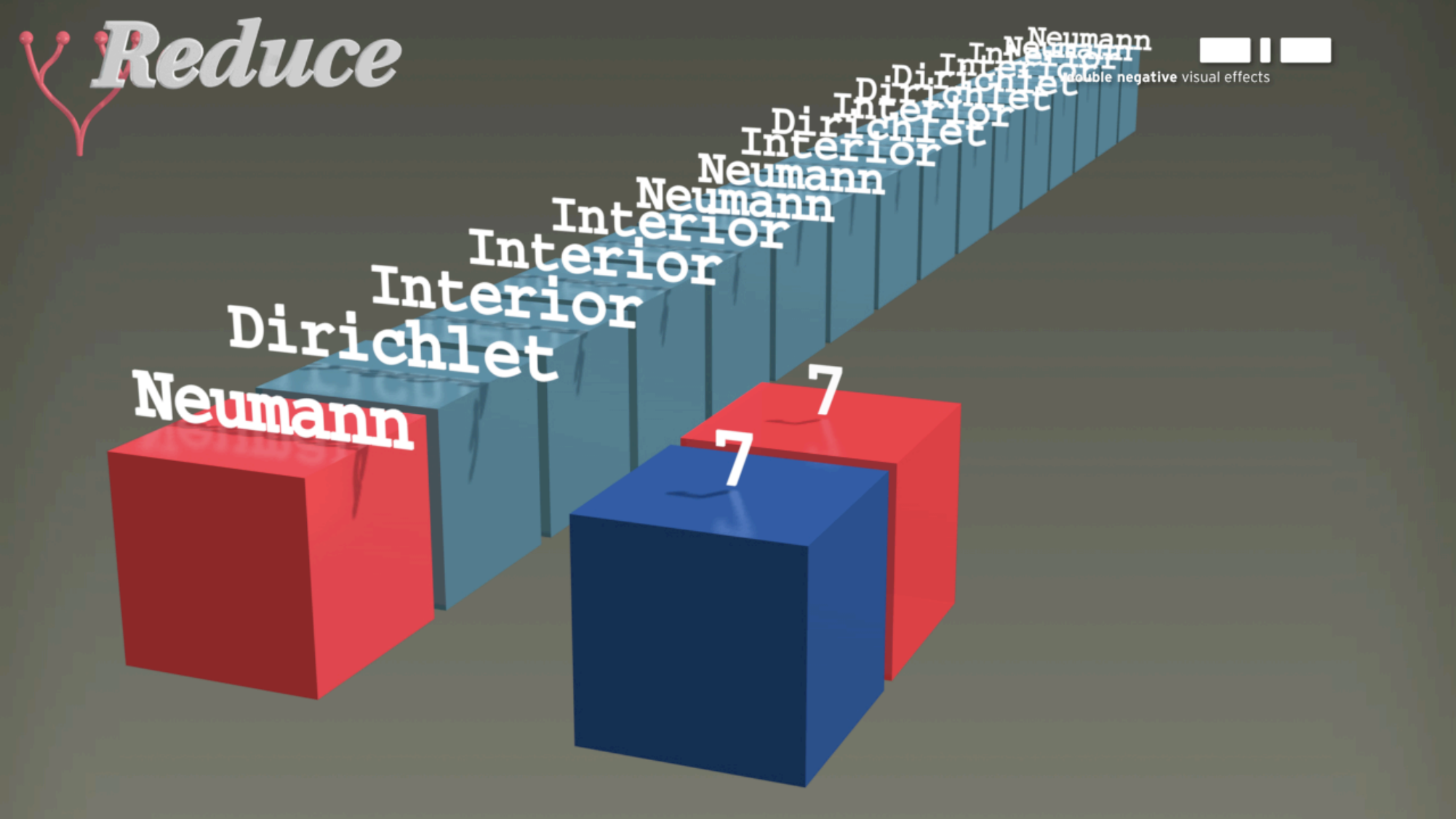


```
Weighted_Blur : Voxel<Box> (output, input)
{
    limit(1, 1, 1, 1, 1, 1);

    value = input:(0, 0, 0) * 4
          + input:(-1, 0, 0) + input:(1, 0, 0)
          + input:(0, -1, 0) + input:(0, 1, 0)
          + input:(0, 0, -1) + input:(0, 0, 1);

    return value / 10;
}
```

Reduce

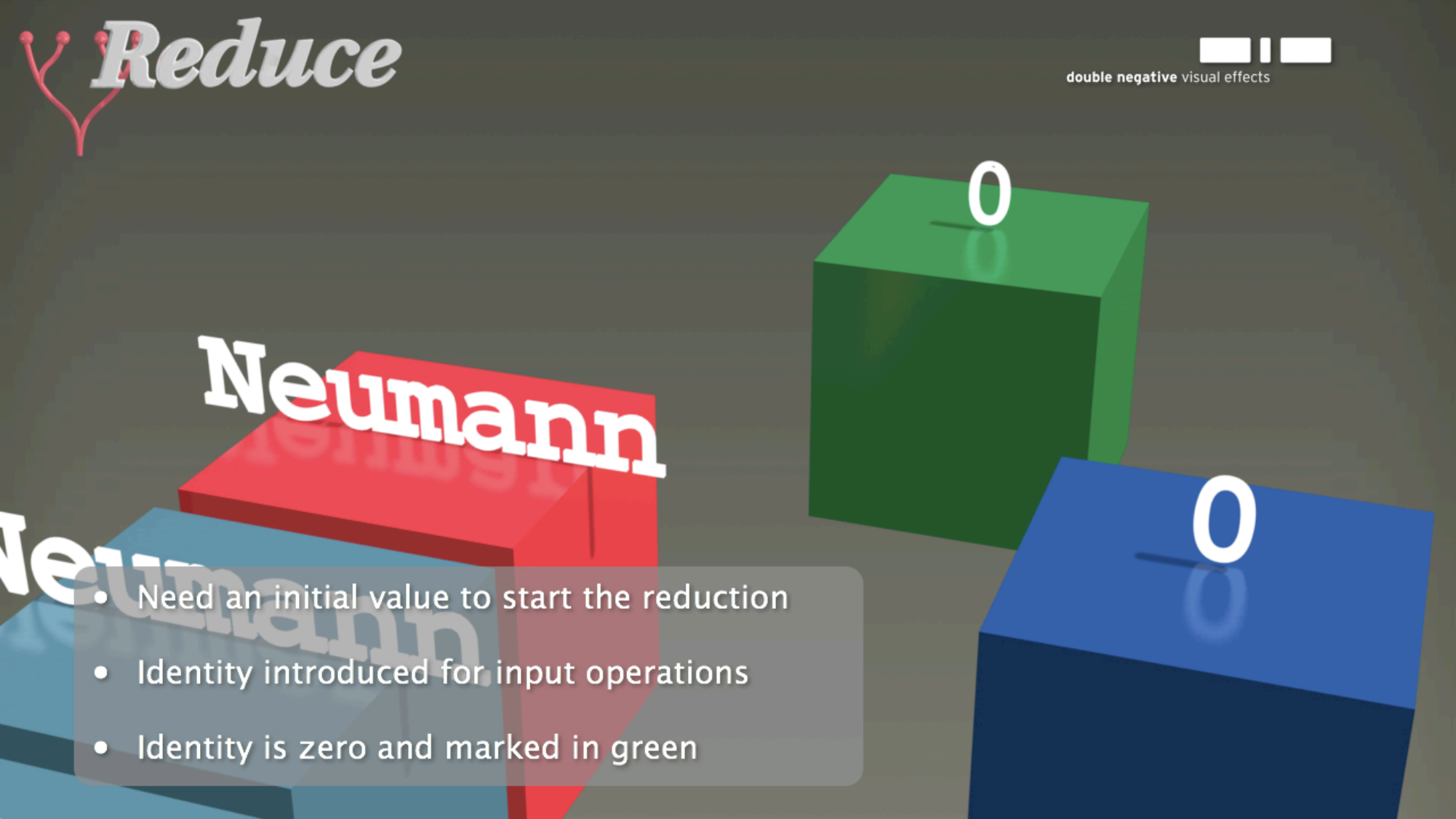


- Reduce primitive can also handle grids
- This example sums the voxels marked “Interior”
- Trivial calculation when done sequentially


Reduce
double negative visual effects
Neumann
Interior
Dirichlet
Interior
Interior
Dirichlet
Neumann
7
7

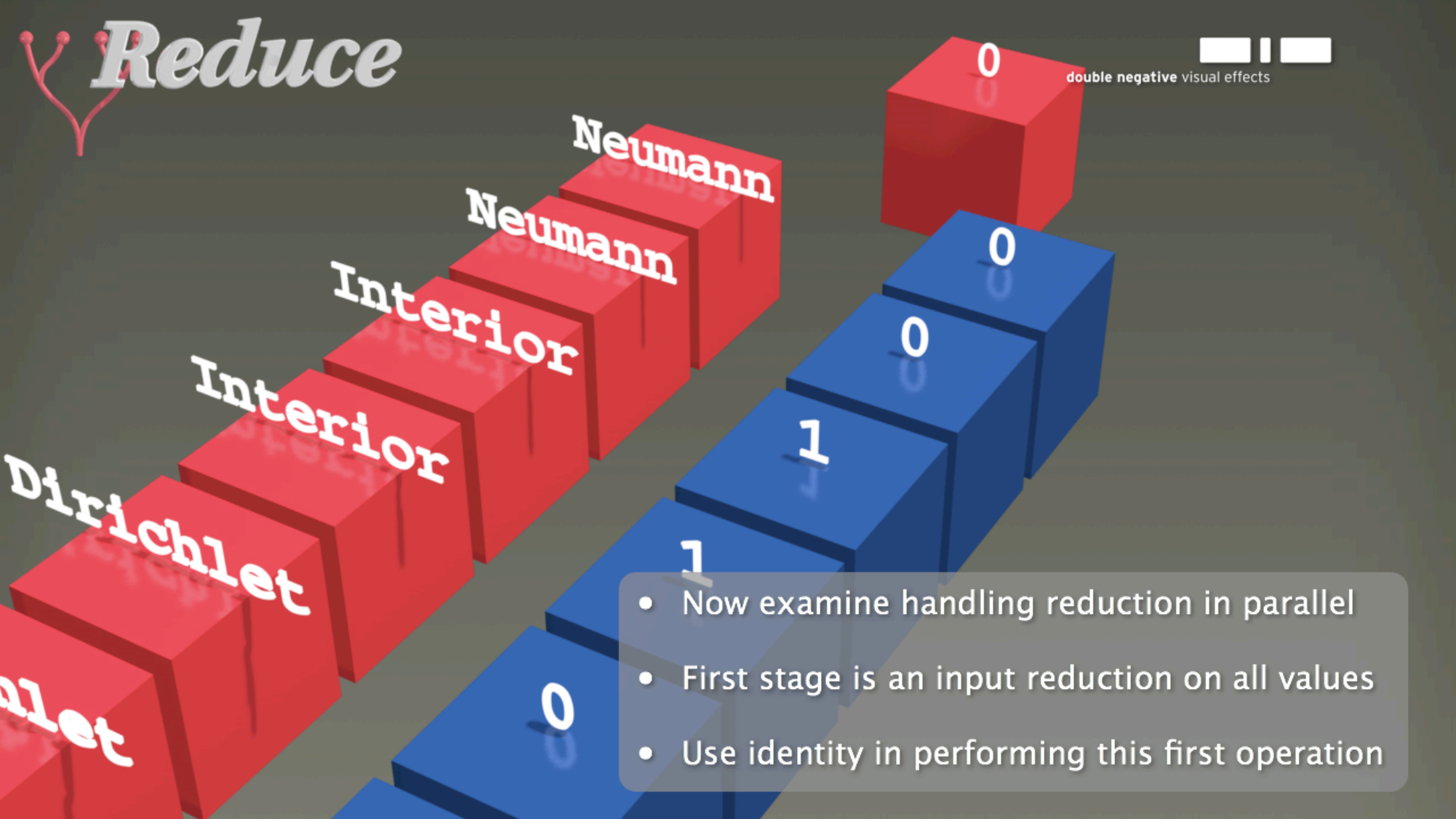
Reduce

double negative visual effects

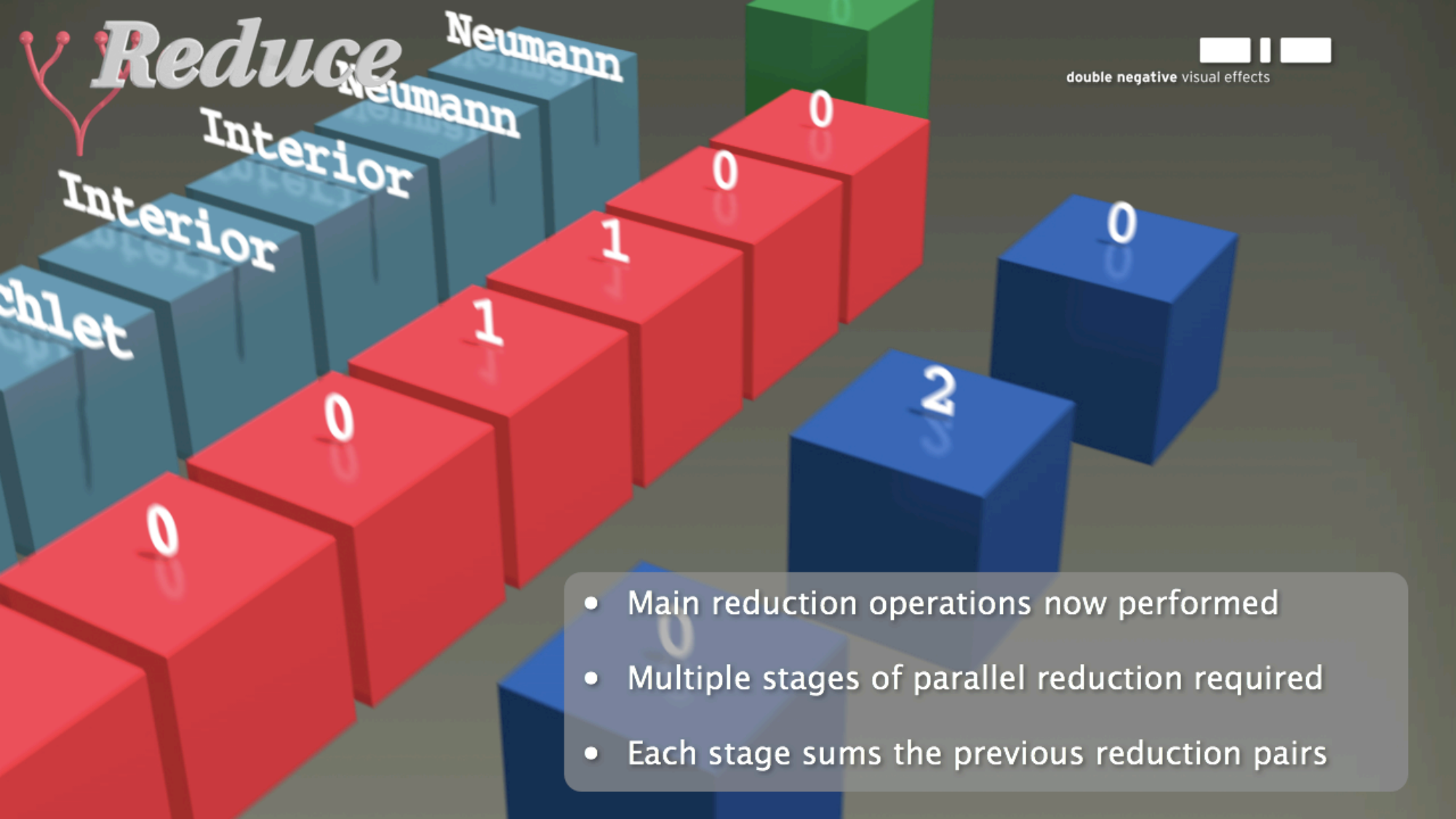


- Need an initial value to start the reduction
- Identity introduced for input operations
- Identity is zero and marked in green

Reduce

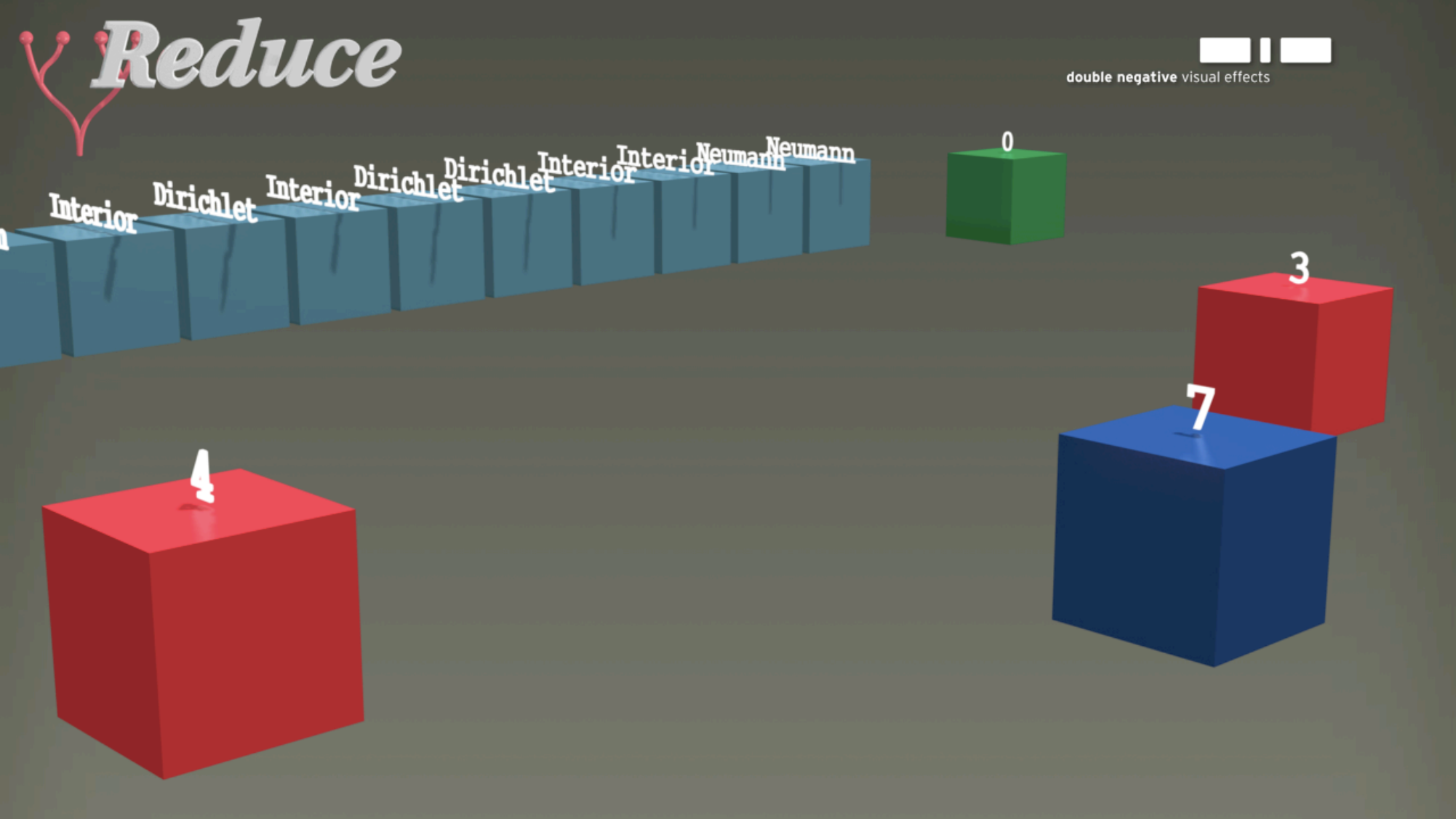


- Now examine handling reduction in parallel
- First stage is an input reduction on all values
- Use identity in performing this first operation



double negative visual effects

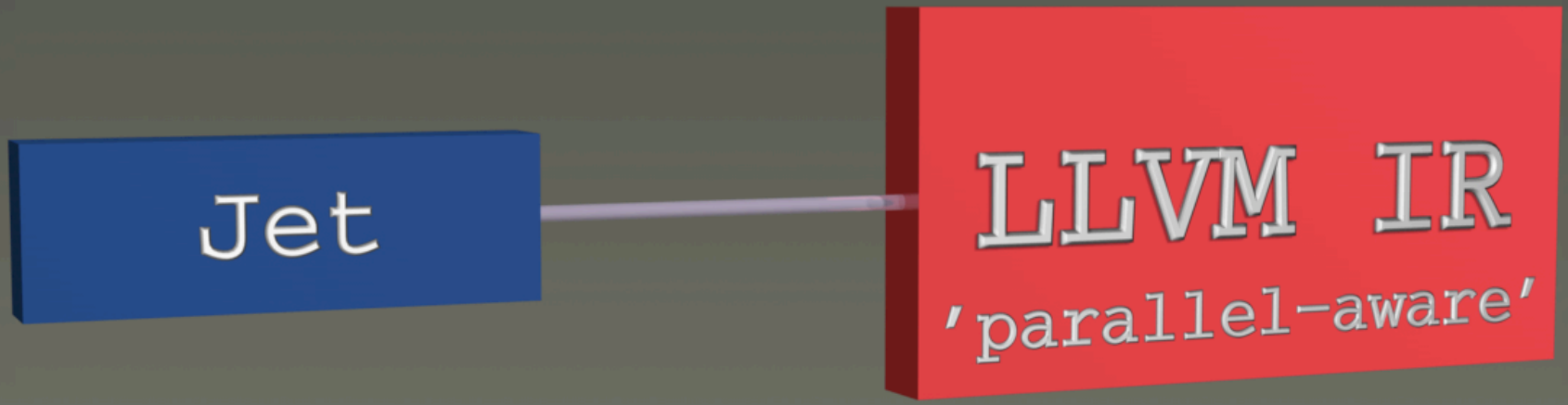
- Main reduction operations now performed
- Multiple stages of parallel reduction required
- Each stage sums the previous reduction pairs


Reduce
double negative visual effects
Interior
Dirichlet
Interior
Dirichlet
Dirichlet
Interior
Interior
Neumann
Neumann
0
3
7
4

Reduce

double negative visual effects

```
Interior_Count : Reduce (reduction, value)
{
    identity(0);

    if (is_input())
    {
        return reduction + (value == INTERIOR);
    }

    return reduction + value;
}
```



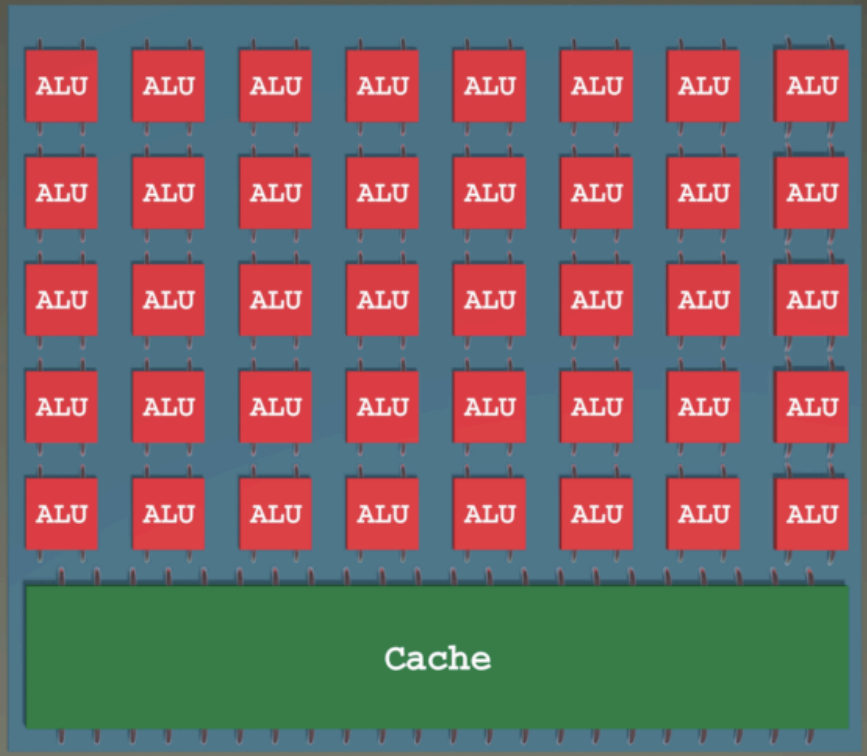
- Jet compiler has a frontend that produces LLVM IR
- IR produced is “parallel-aware”
- A valid form of the IR used in assisting transformation

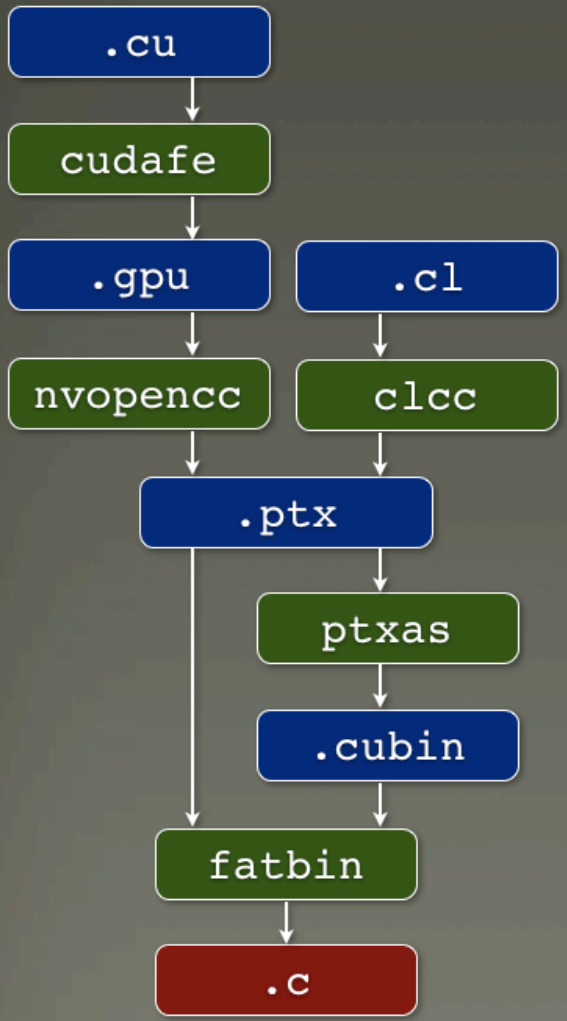
- Declared, but undefined Jet intrinsics used
- Custom passes handle translation of Jet intrinsics
- Some passes generic, others specific to target / data layout

- Standard optimisations still applied to IR
- X86 and PTX backends currently supported
- Backends used to produce machine code for CPU / GPU

GPU CG with LLVM

- NVidia's LLVM PTX backend (Grover)
- Open-source LLVM PTX Backend (Chiou, Holewinski)
- Experimental PTX Backend for AnySL (Rhodin)
- LLVM to AMD IL (Villmow)





NVidia PTX

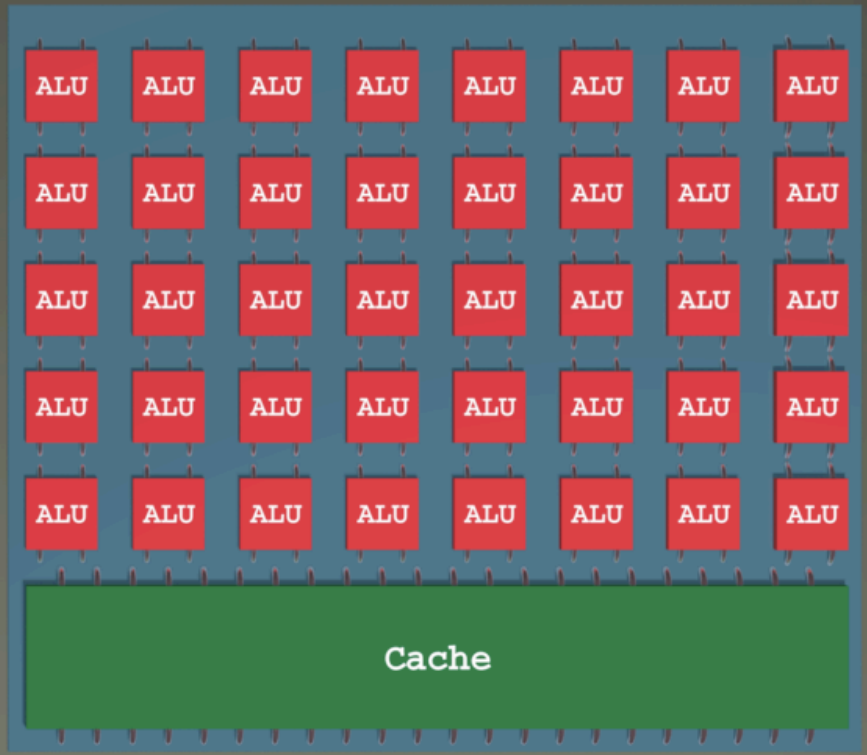
double negative visual effects

- NVidia GPU Compute ISA
- JIT Compiled to Native GPU ISA
- Supported by CUDA and OpenCL

	LLVM IR	PTX
SSA	Yes	No
Integer Types Signed	No	Yes
Register Set	Infinite	Finite

Open-source LLVM PTX Backend

- Target-independent codegen approach
- Instruction-selection in Tablegen
- Supports PTX 2.0+ and SM 1.0+
- Supports 32-bit and 64-bit targets



PTX Backend Usage

- Generating PTX code with llc:

```
llc -march=ptx32 < source.ll
llc -march=ptx64 < source.ll
```

- Generating PTX code with clang:

```
clang -ccc-host-triple ptx32 source.c -O1 -S
clang -ccc-host-triple ptx64 source.c -O1 -S
```

github.com/jholewinski/llvm-ptx-samples

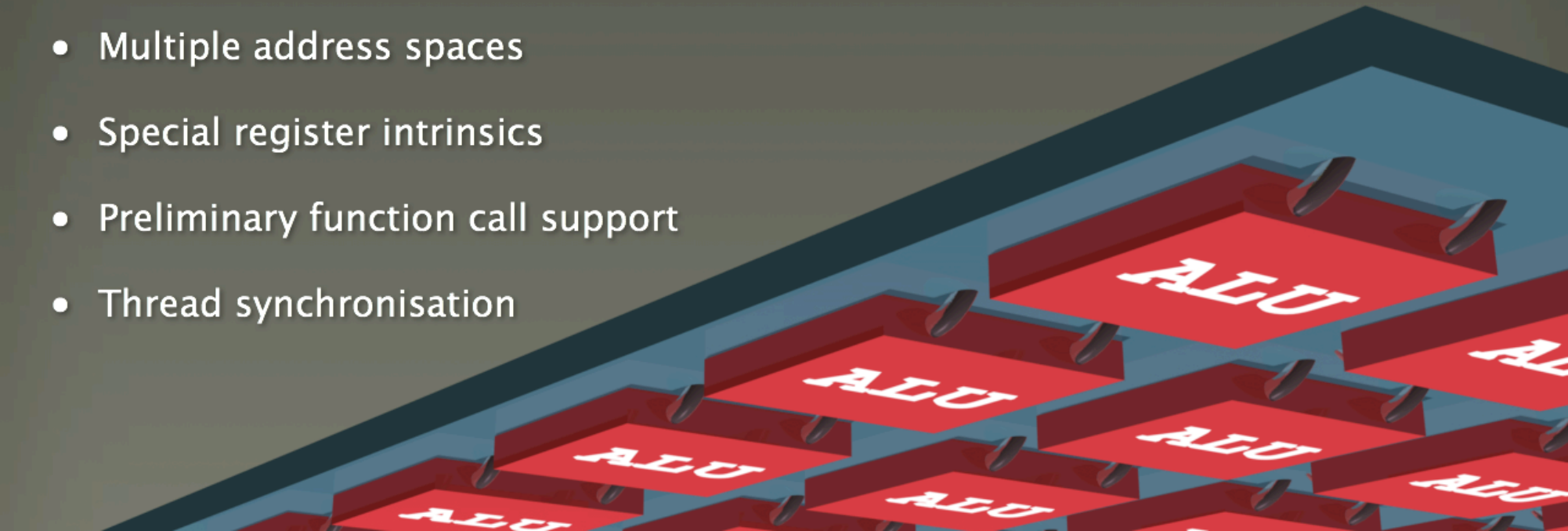
double negative visual effects

Current State

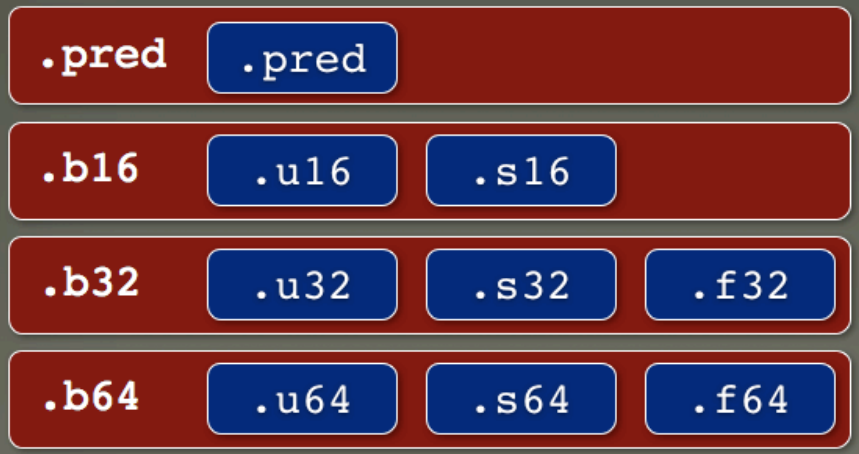
- Instruction-selection:

  - arithmetic, bitwise, control-flow

- Multiple address spaces
- Special register intrinsics
- Preliminary function call support
- Thread synchronisation

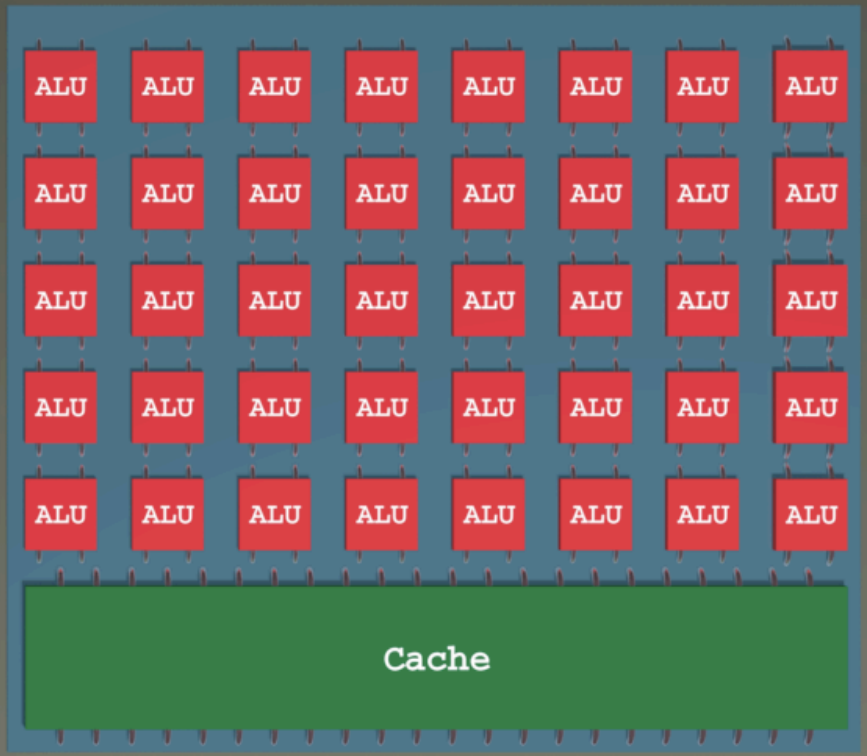


Register Usage



- Now use untyped registers
- Register overflow handled by spilling
- Register allocation done by ptxas

Performance Considerations



- Divergent branching
- Warp occupancy
- Computation vs register re-use

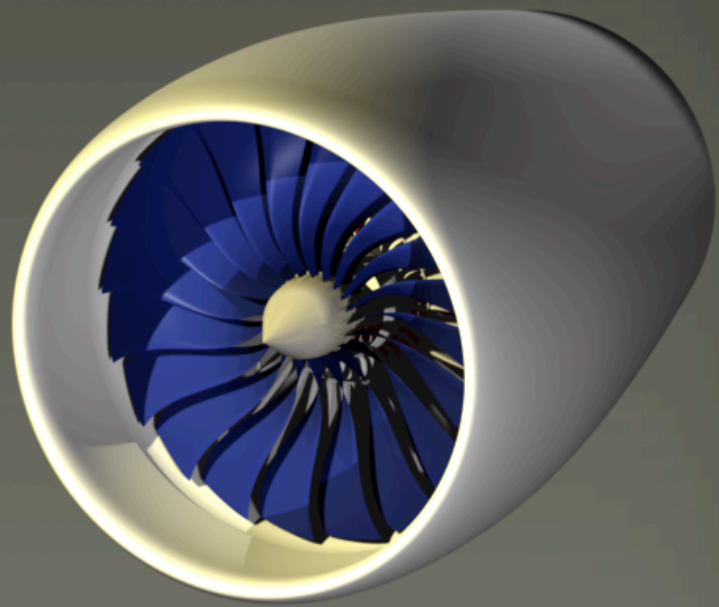
double negative visual effects

In Development

- Better function call support
- Stack frame allocation
- PTX-specific optimisations
- Predicated instructions
- Improved debugging

Jet Language and Compiler

- Separates logic from implementation
- LLVM X86 and PTX targets work well
- Future-proof methodology



Next Steps

- More Primitives
- More Targets
- More Optimisations

Squirt

- Grid Solver (80 kernels)
  - Multigrid Poisson Solver
  - RK2 Cubic Advection Scheme

	MGPCG	Jet CPU	Jet Q4000	Jet C2050	Speedup
64^3	1.76s	1.02s	0.25s	0.14s	7.29x
128^3	11.5s	9.10s	1.58s	0.75s	12.2x
256^3	93.1s	93.5s	13.2s	6.07s	15.4x

double negative visual effects

Squirt

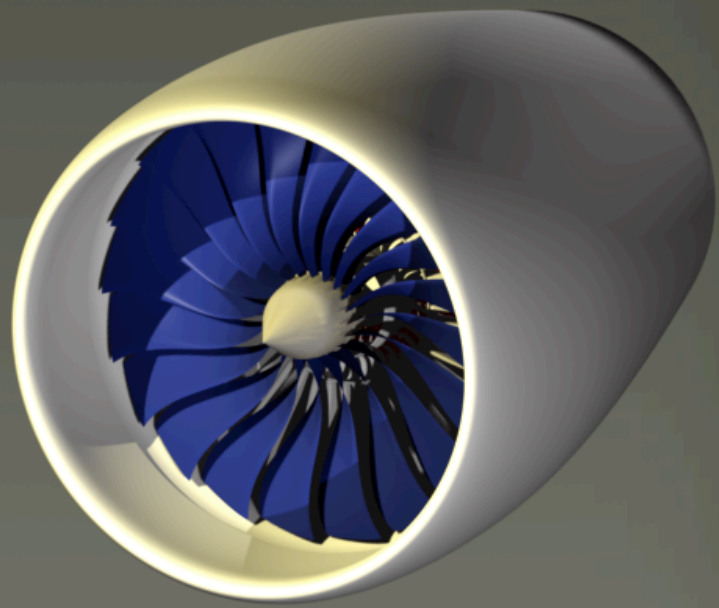
- Grid Solver (80 kernels)
  - Multigrid Poisson Solver
  - RK2 Cubic Advection Scheme

	Squirt	Jet CPU	Jet GPU	Speedup
50^3	1m58s	1m40s	12.5s	6.70x
100^3	17m40s	13m24s	1m19s	10.4x
200^3	3h06m	2h41m	11m45s	13.7x

double negative visual effects

Jet

- Flexible, Expressive Language
- Fast, Heterogeneous Compiler
- Productivity, Performance, Portability



Double Negative are recruiting compiler engineers!

Email: dan@dneg.com