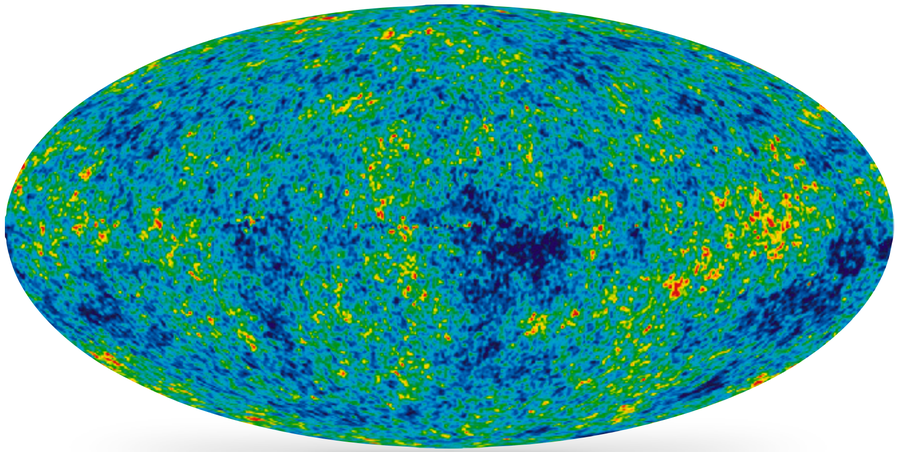
PUBLIC
Sony Interactive Entertainment

“Demo of a repository for statically compiled programs”
2016 US LLVM Developers’ Meeting

Paul Bowen-Huggett
paul.huggett@sony.com

Agenda

Background

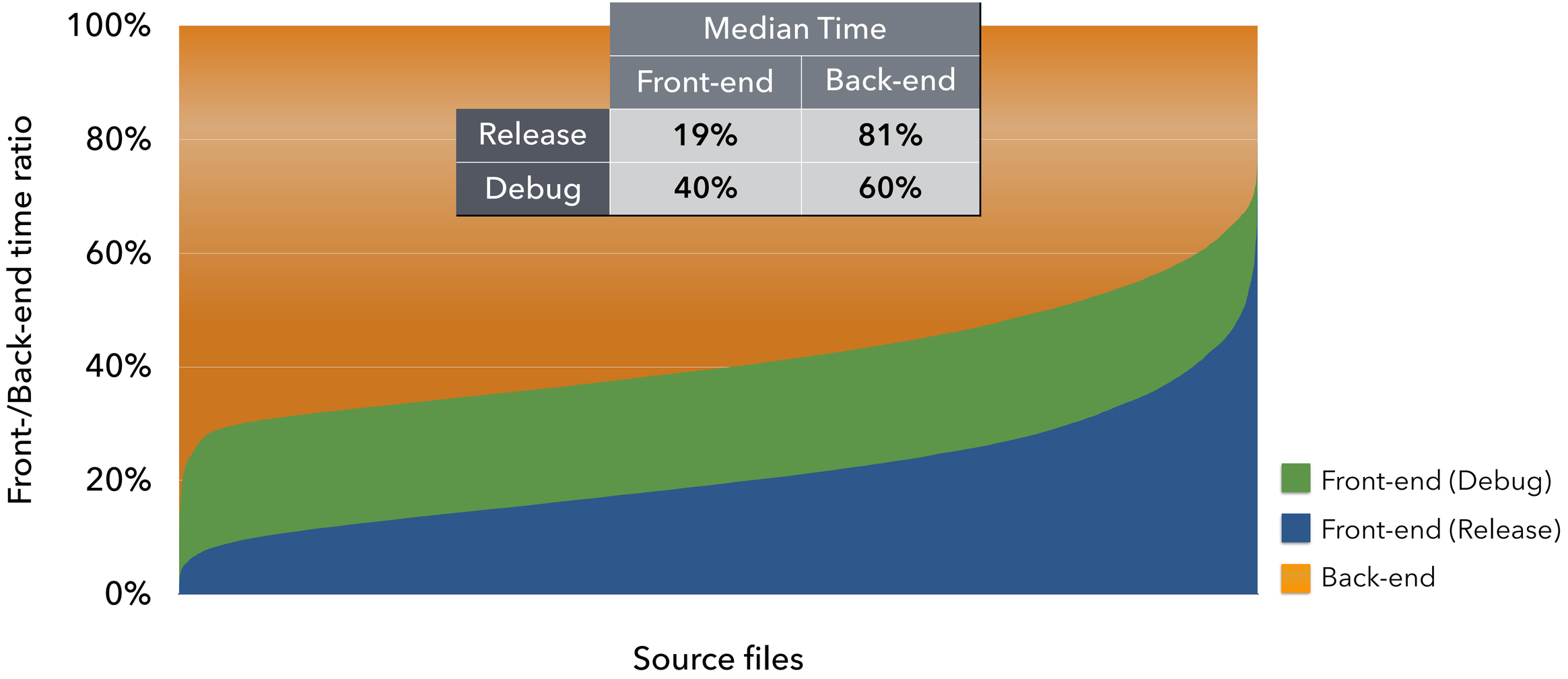


RFC

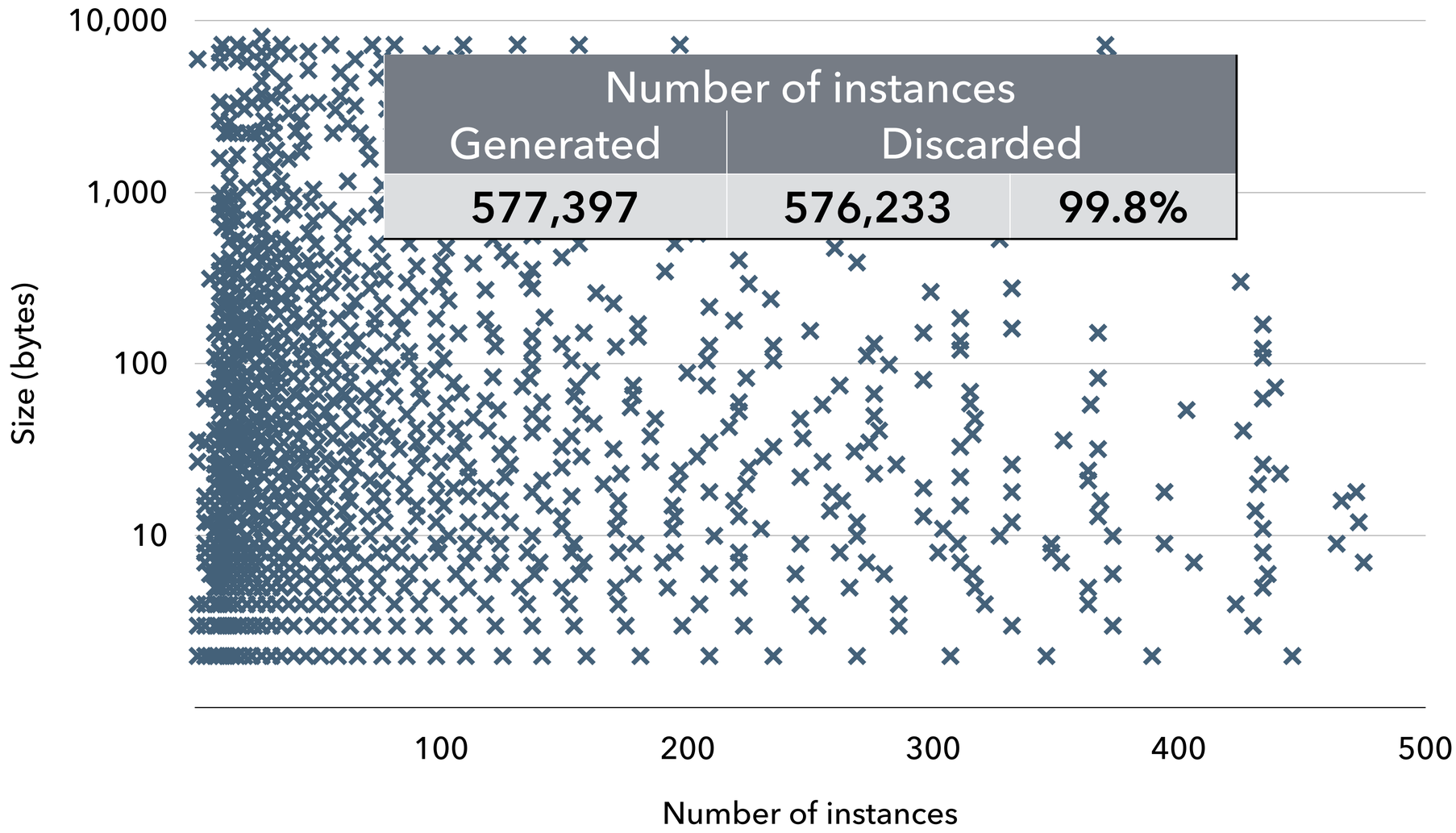
- Is the idea generally sound? Obvious improvements?
- Is it something we should think about for LLVM?
- There are several potentially related projects (C++ modules IFC, compilation database, ThinLTO, etc.) Views from respective owners?

Chromium Browser Build Ratios

	Median Time	
	Front-end	Back-end
Release	19%	81%
Debug	40%	60%



Chromium Browser COMDAT Groups

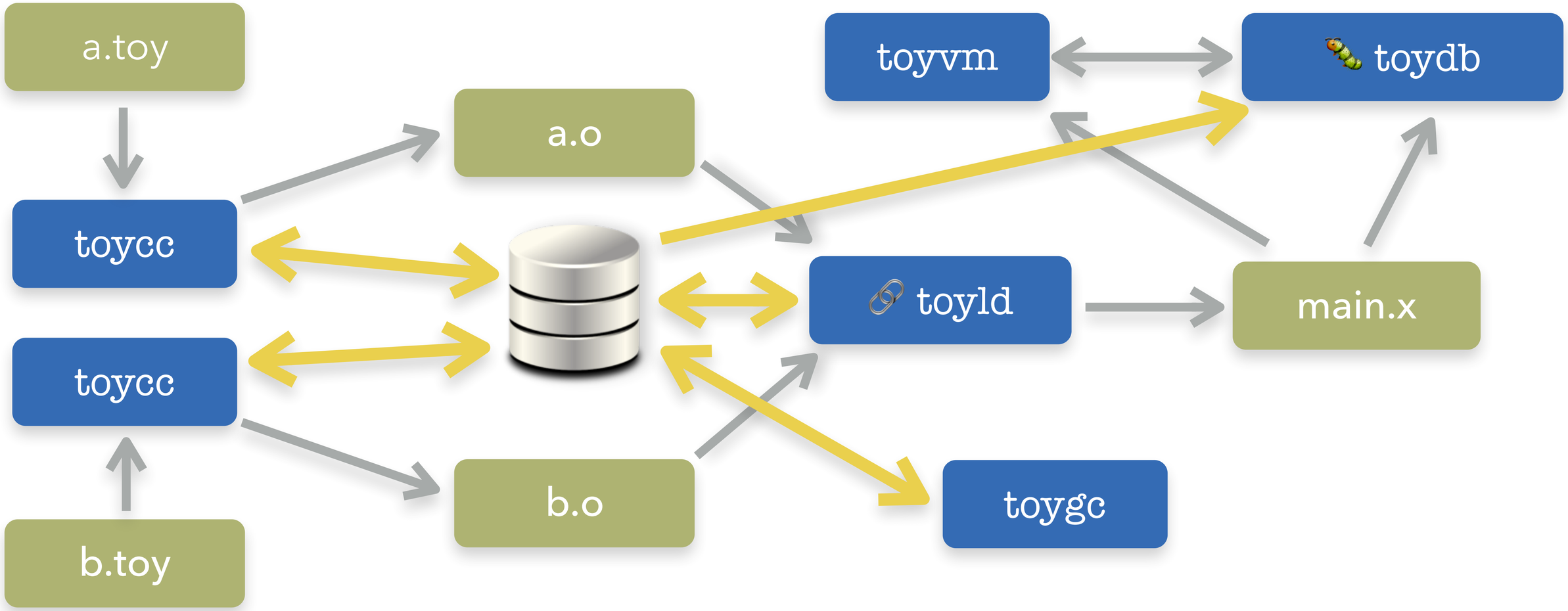


Toy Tools

- Toy programming language
- Available on github: https://github.com/SNSystems/Toy-tools
- Command line tools:

Role	Name
Compiler	toycc
Linker	toyld
Debugger	toydb
Runtime	toyvm

Role	Name
Garbage Collector	toygc
Strip	toystrip
Merge	toymerge

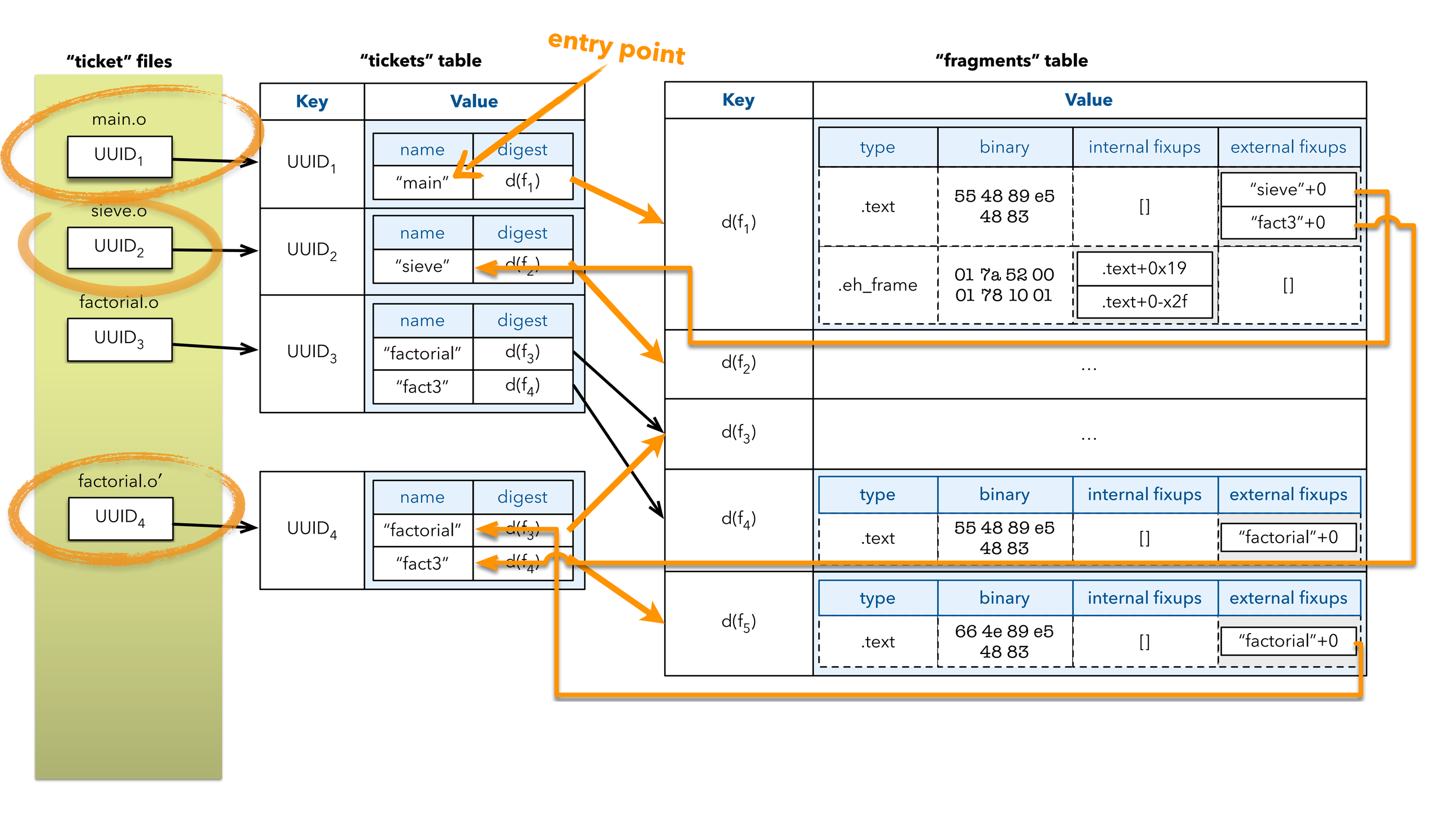

a.toy
toycc
toycc
b.toy
a.o
b.o
toyvm
toydb
toyld
main.x
toygc

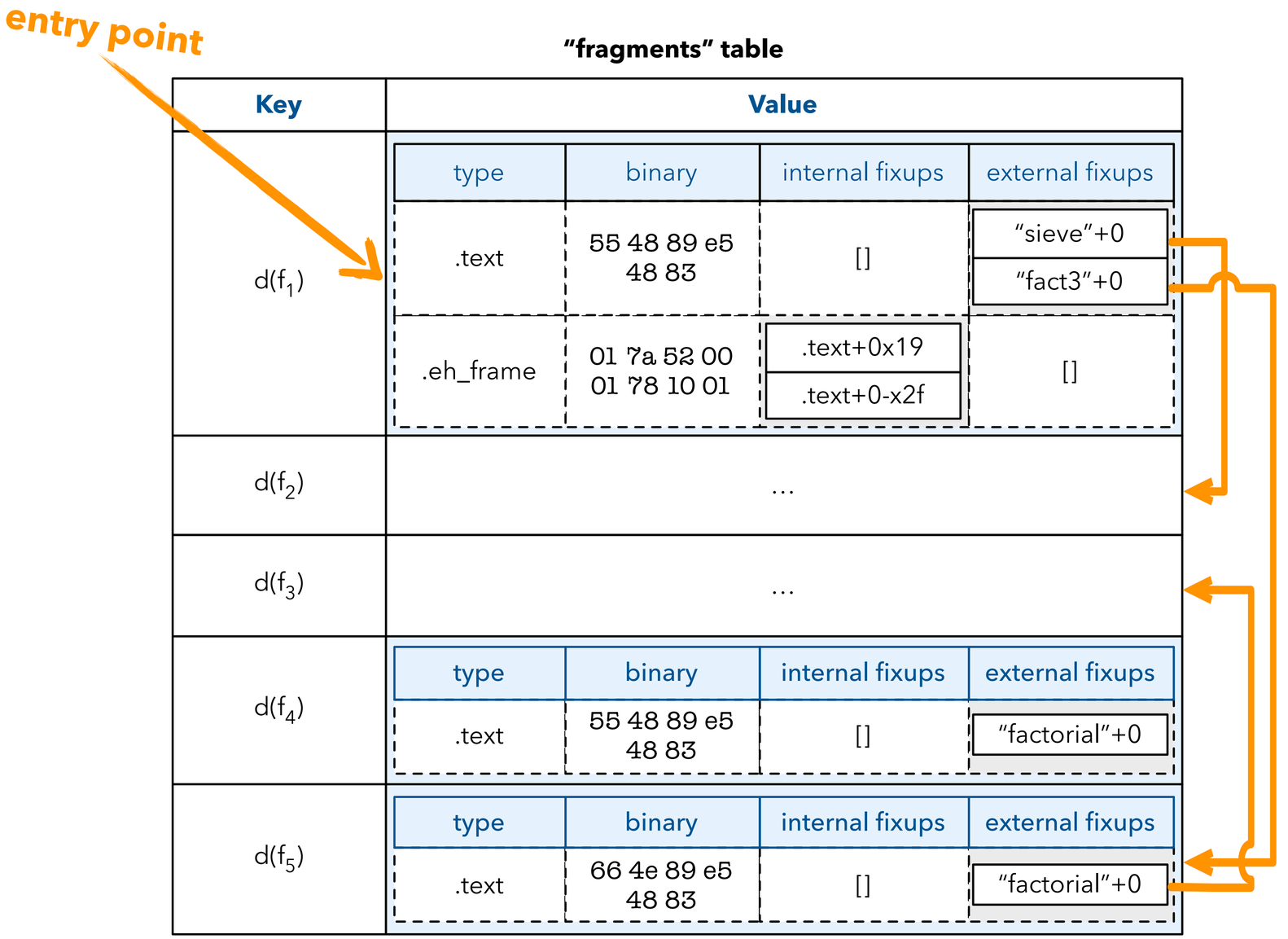
Limitations

1. It’s just a toy!
2. Written in Python (3.5)
3. Output files are YAML
4. No concurrency
5. No backward compatibility
6. Says nothing about performance
7. The Toy language is nothing like C++:
    - VM has no registers, 3 stacks
    - Dynamic language, no user-defined types, no vague linkage…

Demo

1. “Hello, World”
2. “Modules”
3. “Distributed”

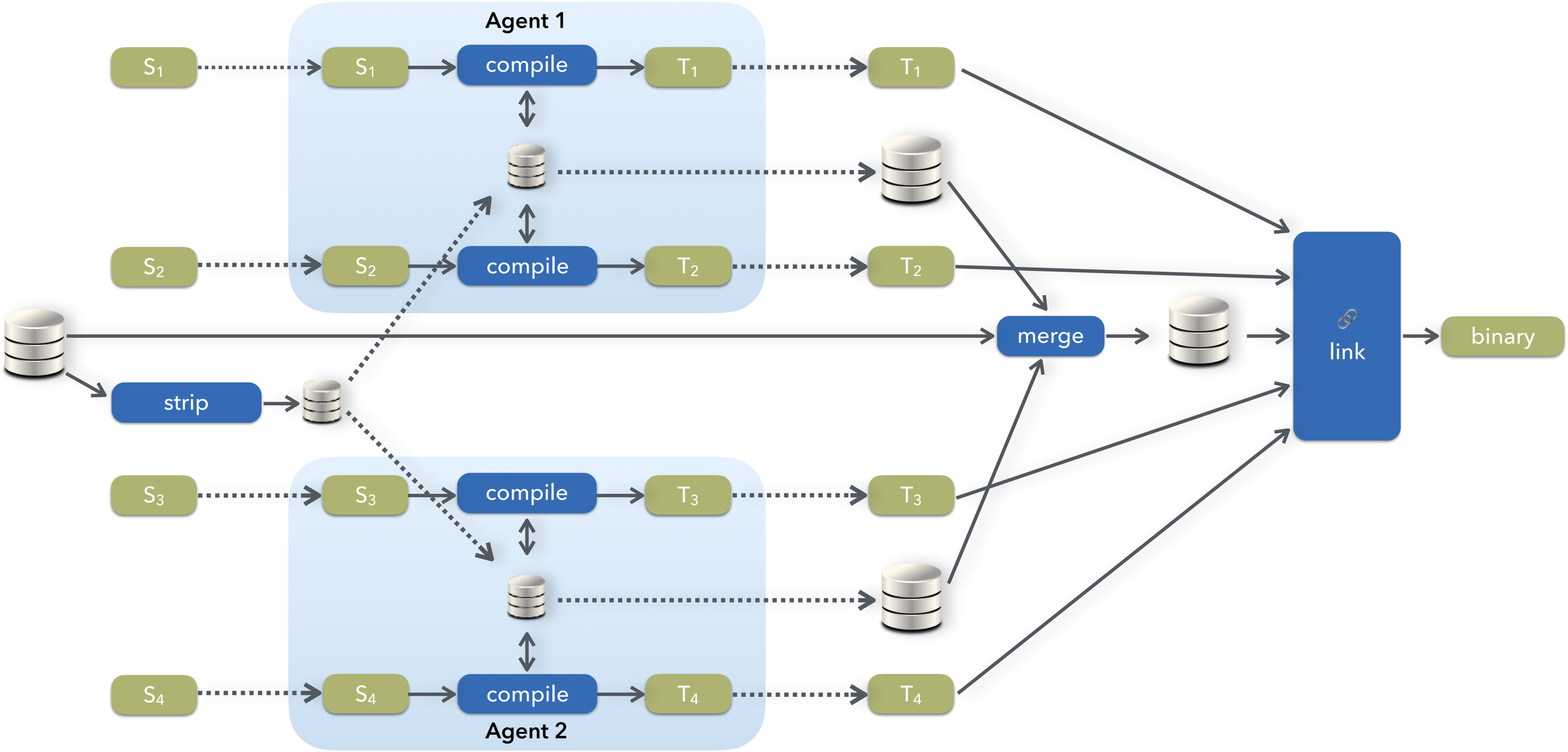

"ticket" files
"tickets" table
entry point
"fragments" table
main.o
UUID1
sieve.o
UUID2
factorial.o
UUID3
factorial.o'
UUID4
Key
Value
name
digest
"main"
d(f1)
"sieve"
"factorial"
d(f3)
"fact3"
d(f4)
type
binary
internal fixups
external fixups
.text
55 48 89 e5
48 83
[]
"sieve"+0
"fact3"+0
.eh_frame
01 7a 52 00
01 78 10 01
.text+0x19
.text+0-x2f
d(f2)
...
d(f5)
66 4e 89 e5
48 83
"factorial"+0


entry point
"fragments" table
Key
Value
d(f1)
type
binary
internal fixups
external fixups
.text
55 48 89 e5
48 83
[]
"sieve"+0
"fact3"+0
.eh_frame
01 7a 52 00
01 78 10 01
.text+0x19
.text+0-x2f
[]
d(f2)
...
d(f3)
...
d(f4)
type
binary
internal fixups
external fixups
.text
55 48 89 e5
48 83
[]
"factorial"+0
d(f5)
type
binary
internal fixups
external fixups
.text
66 4e 89 e5
48 83
[]
"factorial"+0

Distributed Builds

Should the repository be a network resource?

Distributed Build



Challenges?

- Remember, it’s just a toy… Need a production-ready C++ implementation
- Understand real-world growth rates and GC strategy
- Doesn’t show solutions to:
  - Fast storage with efficient indices
  - LLVM IR hashing
  - Efficient debug type references

Conclusion

- Potential to reduce re-compile times by ~60% (“speed-of-light” based on Chrome Debug)
- Small code changes benefit the most
- No source code changes
- Eradicate duplication and redundancy **at source**: minimize link-time processing and copying
- (Almost) No change to workflows
- Next steps:
  - Data store (in-process, memory-mapped hash tables)
  - Prototype:
    - IR hashing
    - MC back-end
    - Repository-based linker

Q&A