JetStream: Cluster-scale Parallelization of Information Flow Queries

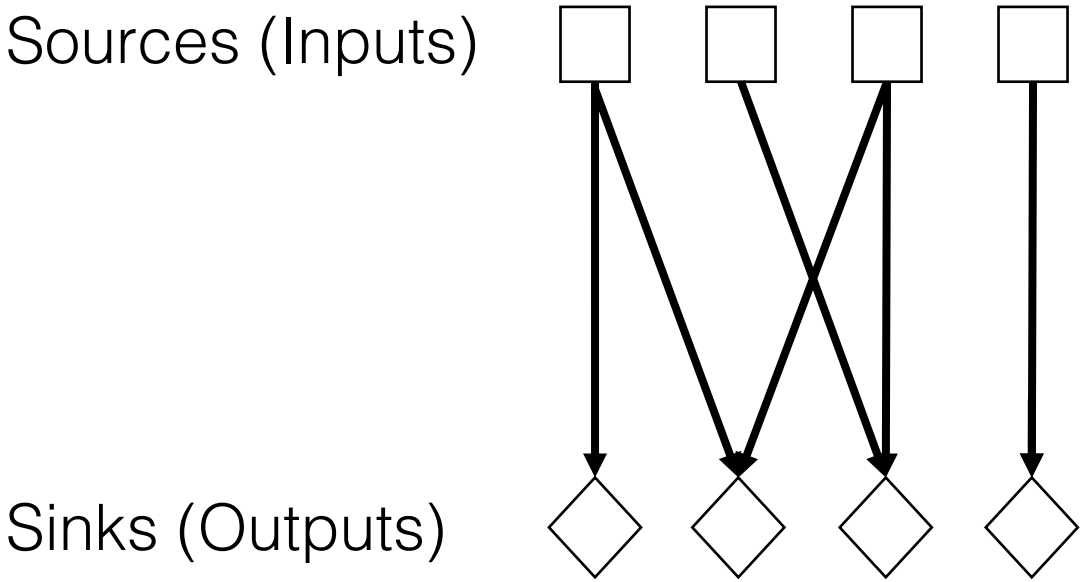
Andi Quinn, David Devecsery,

Peter Chen and Jason Flinn

University of Michigan

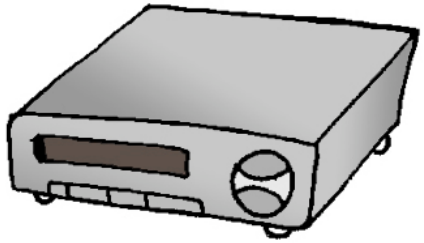
Dynamic Information Flow Tracking

- DIFT instruments execution to track causality
- Also known as “Taint-Tracking”



DIFT for Debugging

Server

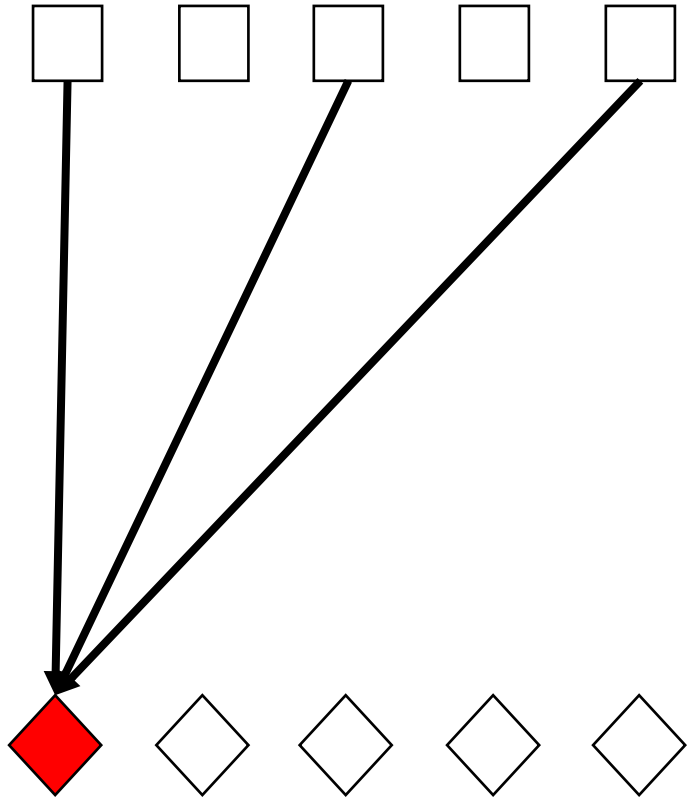


DIFT for Debugging

Inputs

Server

Outputs

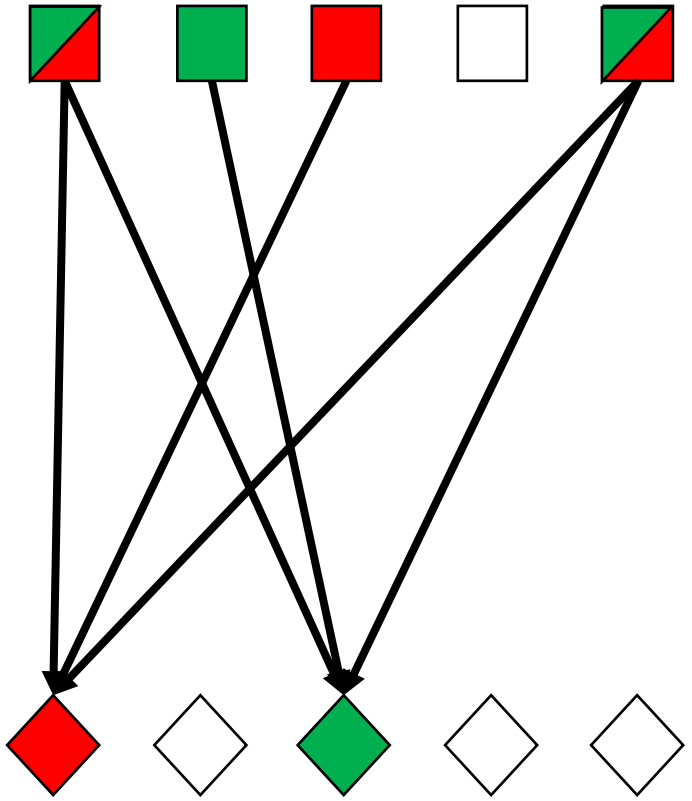


DIFT for Debugging

Inputs

Server

Outputs

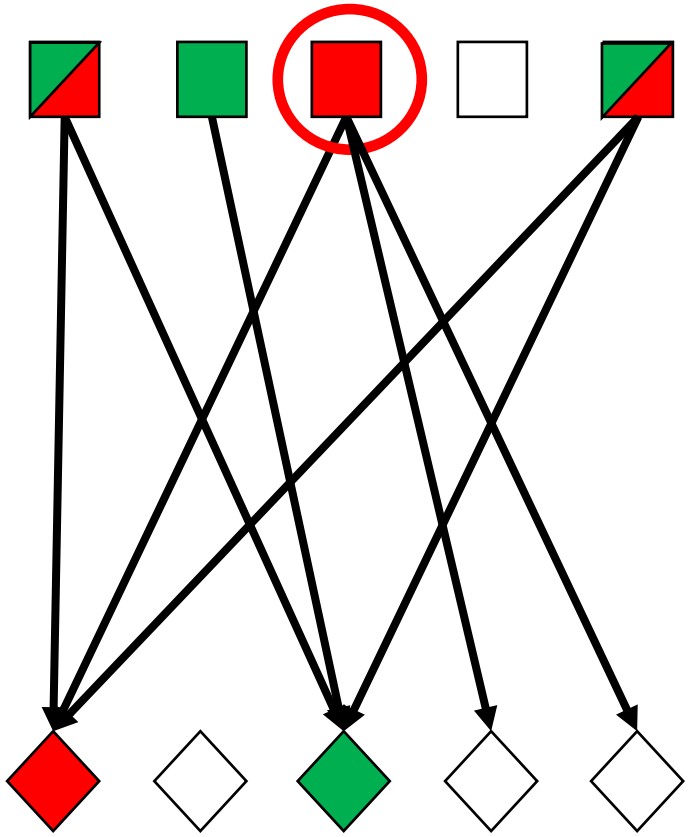


DIFT for Debugging

Inputs

Server

Outputs

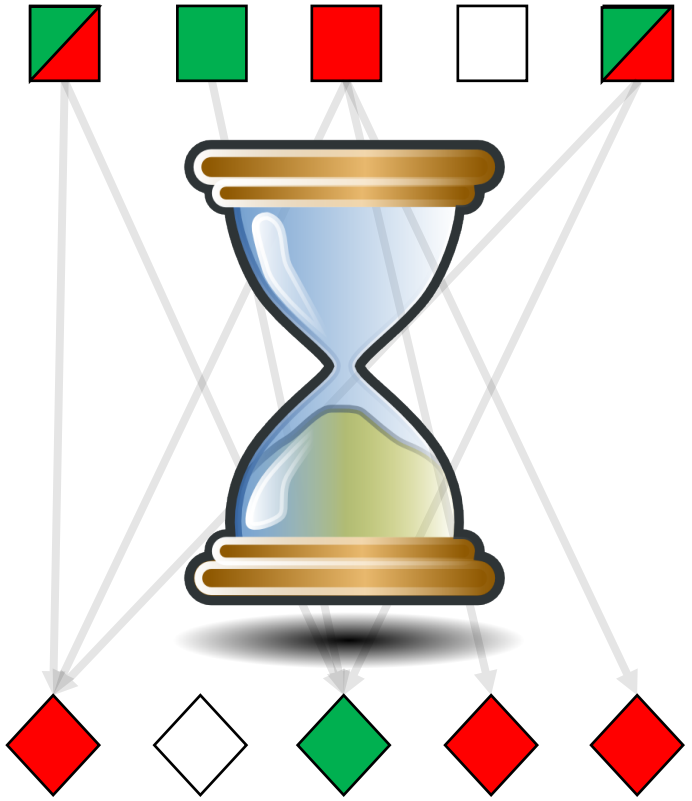


DIFT for Debugging

Inputs

Server

Outputs



DIFT – limitations

Arnold ‘14

Overheads ~100x

X-Ray ‘12

Long queries

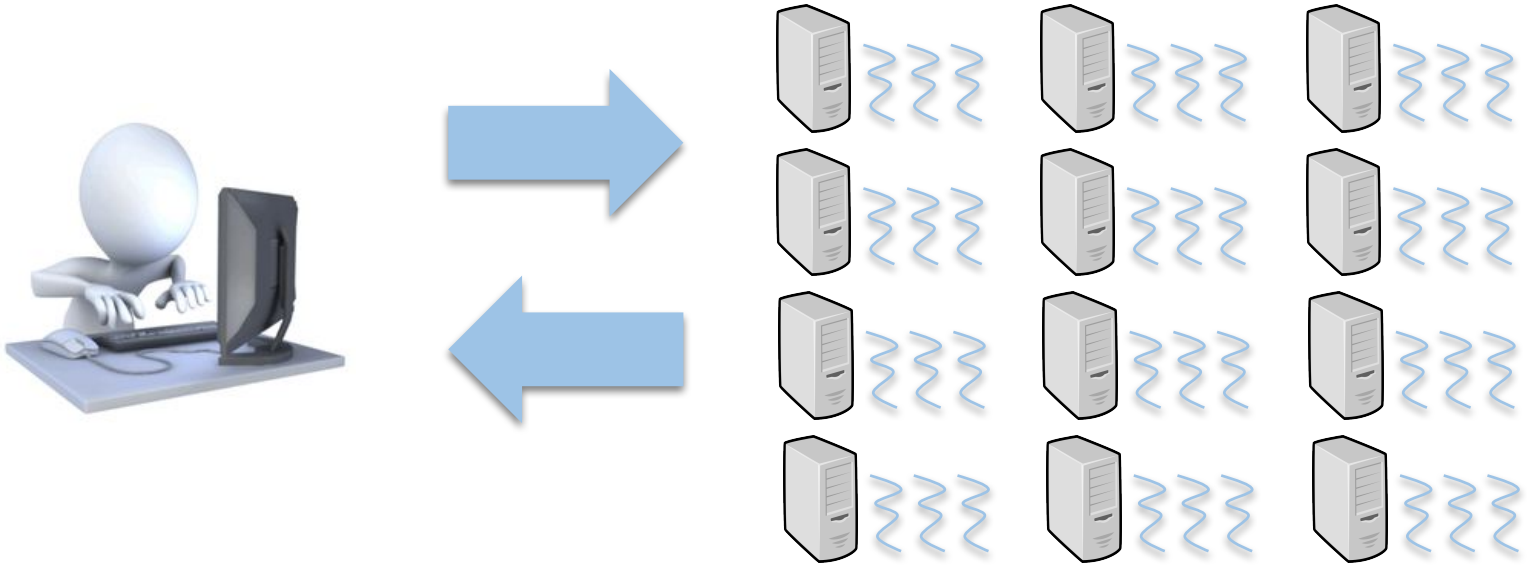
TaintDroid ‘10

No native code

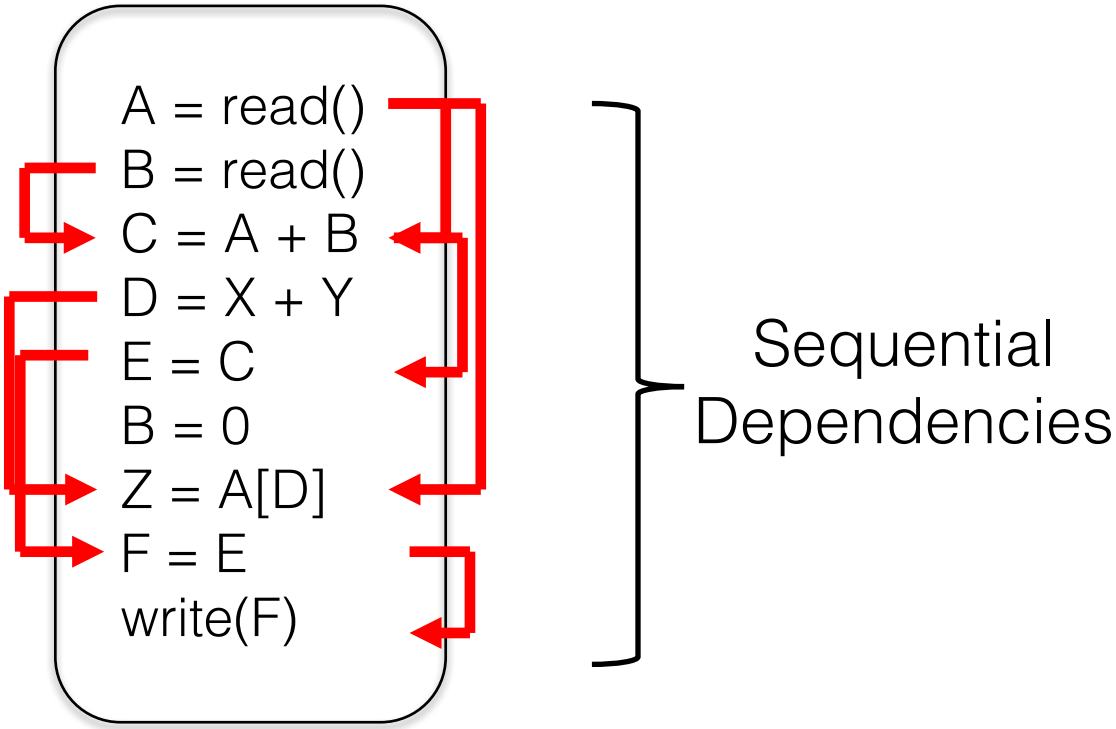
Backtracker ‘03

Coarse-grained causality

Parallelize DIFT



Parallelizing DIFT is HARD



Parallelizing DIFT is HARD

Speck (ASPLOS '08)

> the parallel taint tracker outperforms the sequential version. With 8 cores, the parallel version achieves a 2x speedup compared to its sequential counterpart.

Parallel Lifeguards (ASPLOS '08)

> As shown in Figure 10, we achieve 1.2X–3.4X speedup with 8 workers.

Parallelizing DIFT is HARD

Speck (ASPLOS '08)

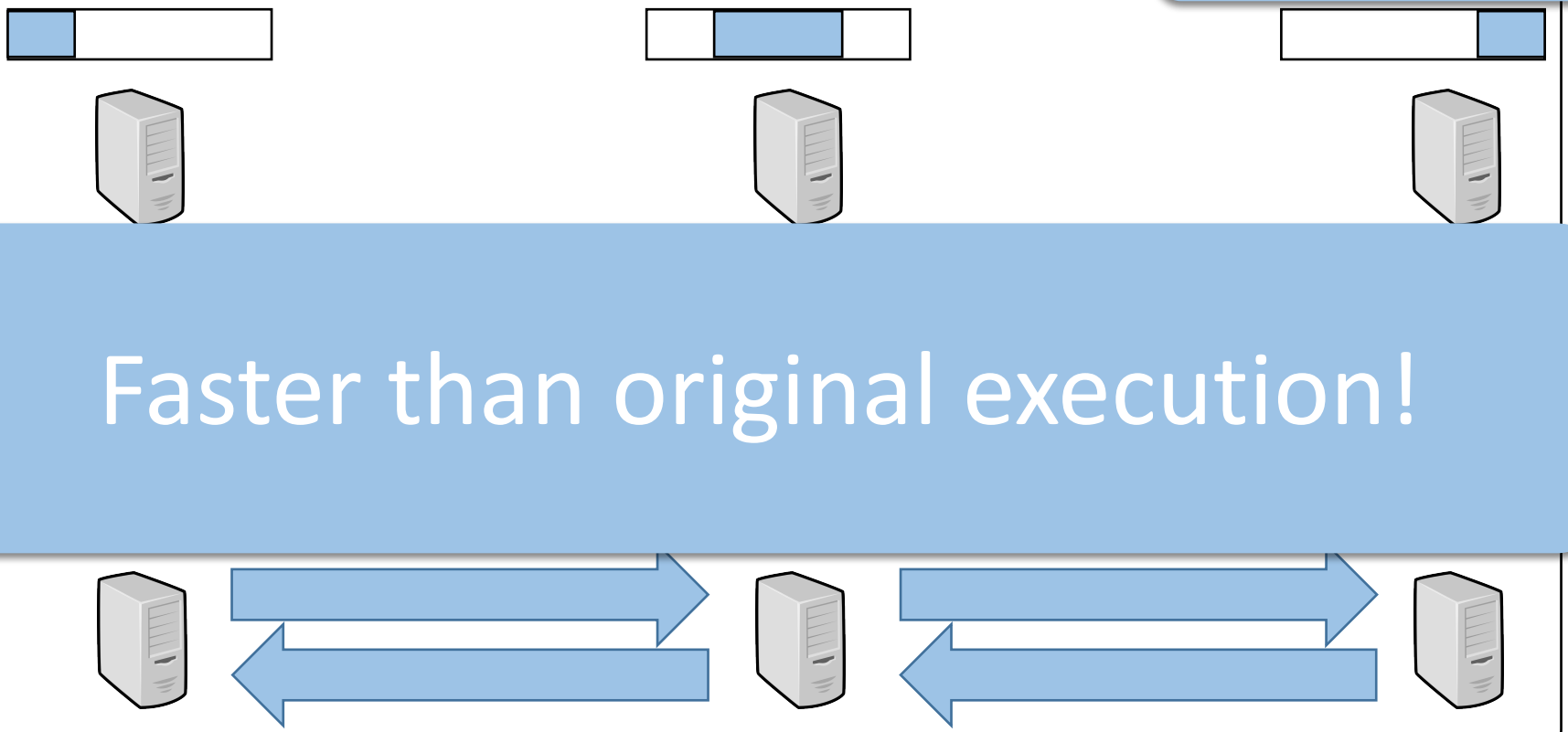
"Embarrassingly Sequential"
- Ruwase et al.

As shown in Figure 10, we achieve 1.2X–3.4X speedup with 8 workers.

JetStream

2x → 21x

Local DIFT – epoch parallelism

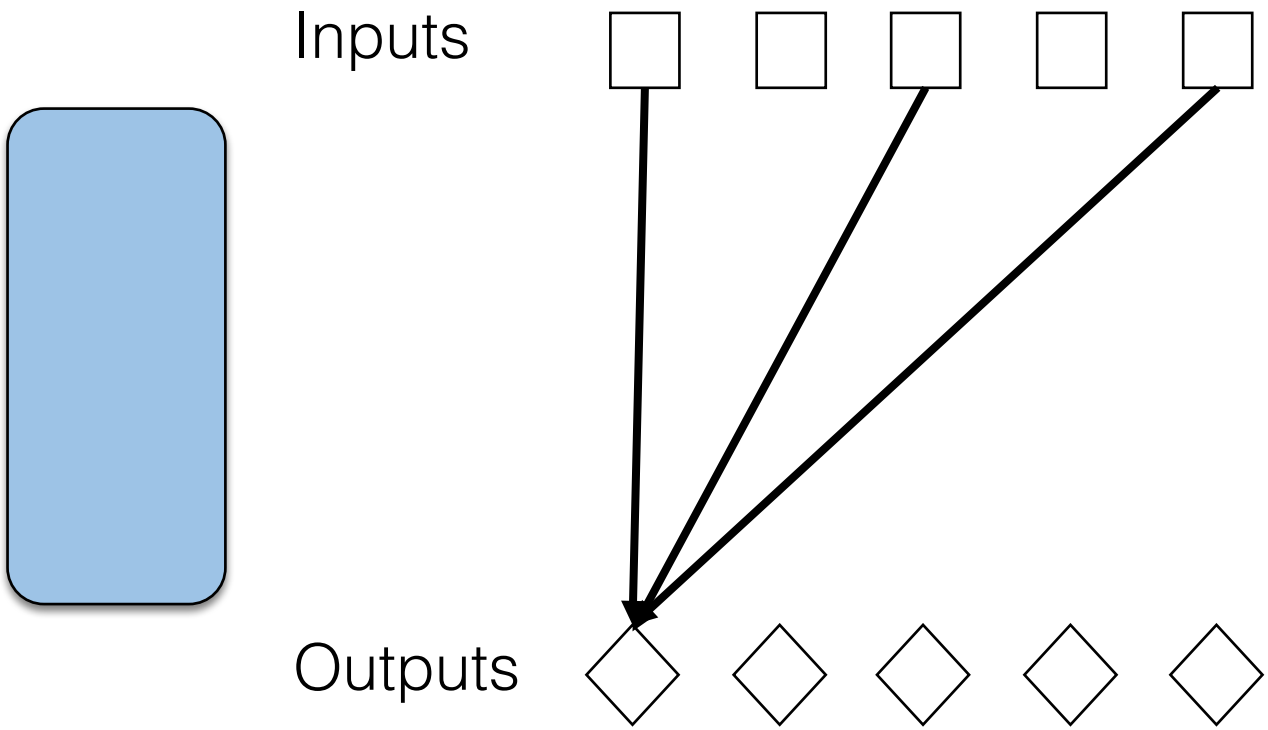


Faster than original execution!

Outline

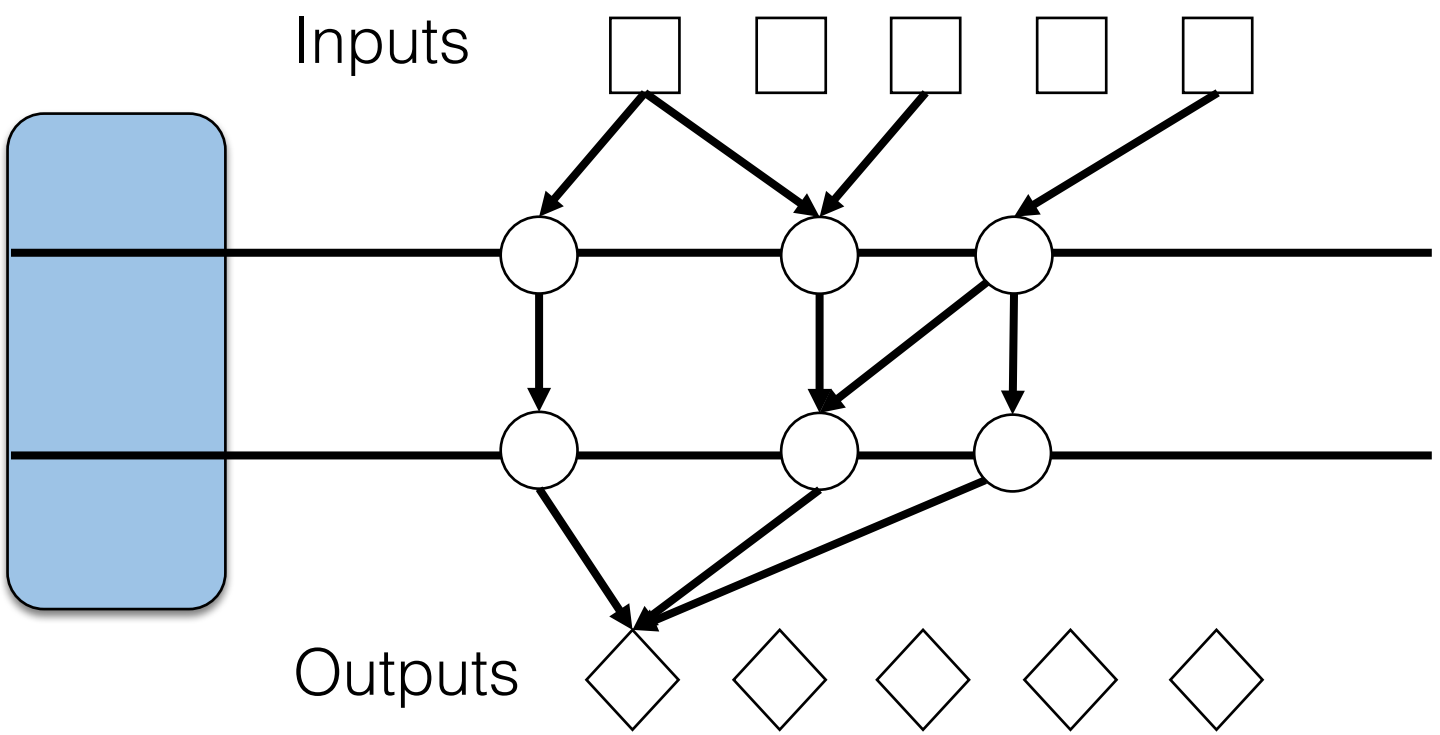
- Motivation and Introduction
- Design of JetStream
  - Local DIFT
  - Aggregation
- Evaluation

Debugging Query



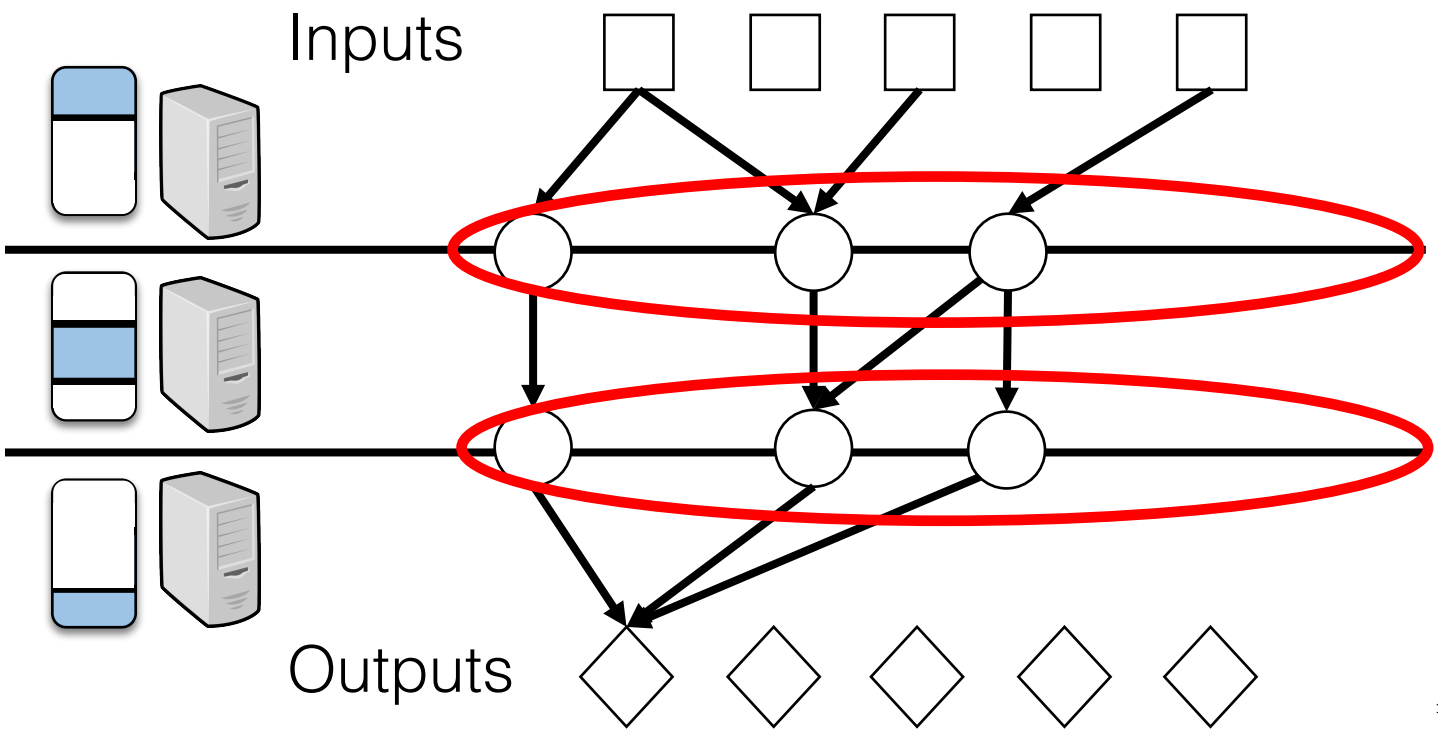
Local DIFT

Time slice execution into Epochs



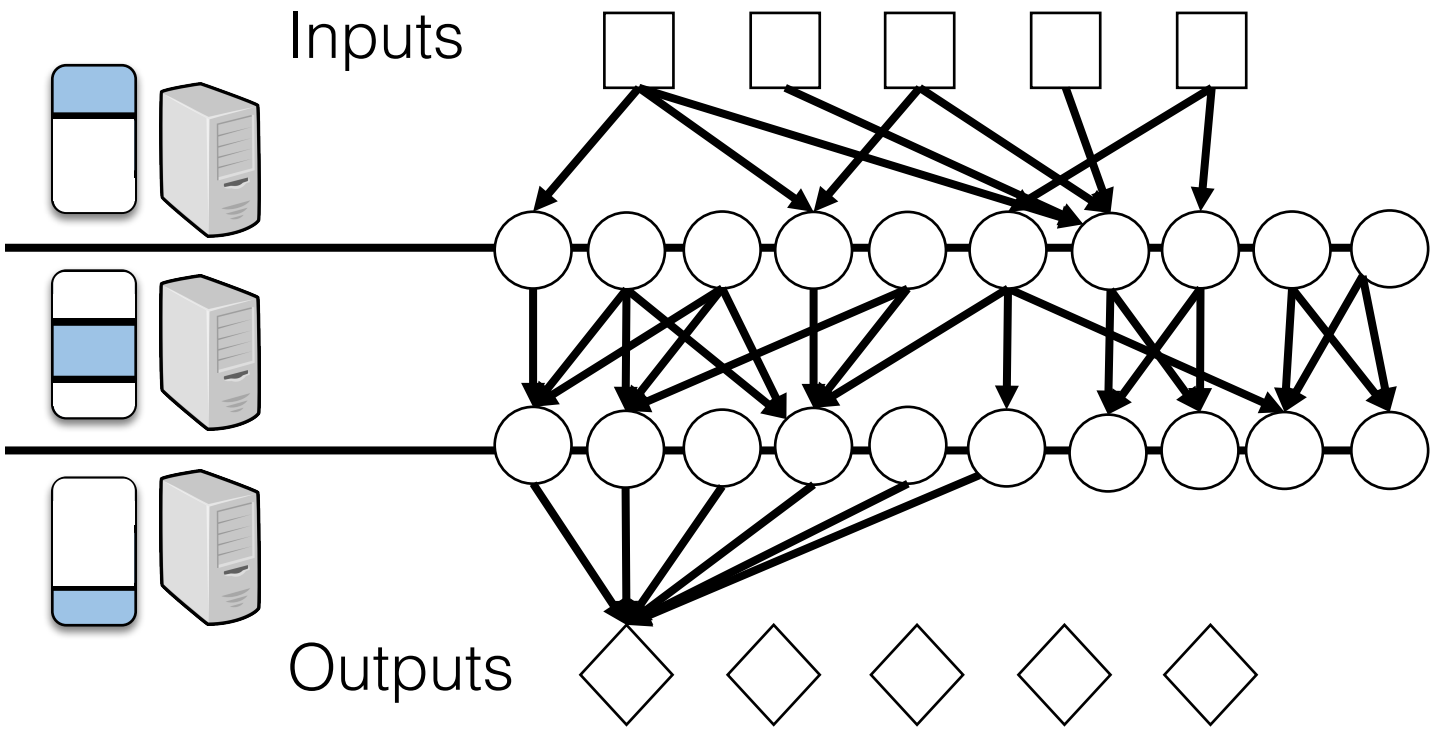
Local DIFT

Leverage Record and Replay to calculate DIFT in parallel



Local DIFT

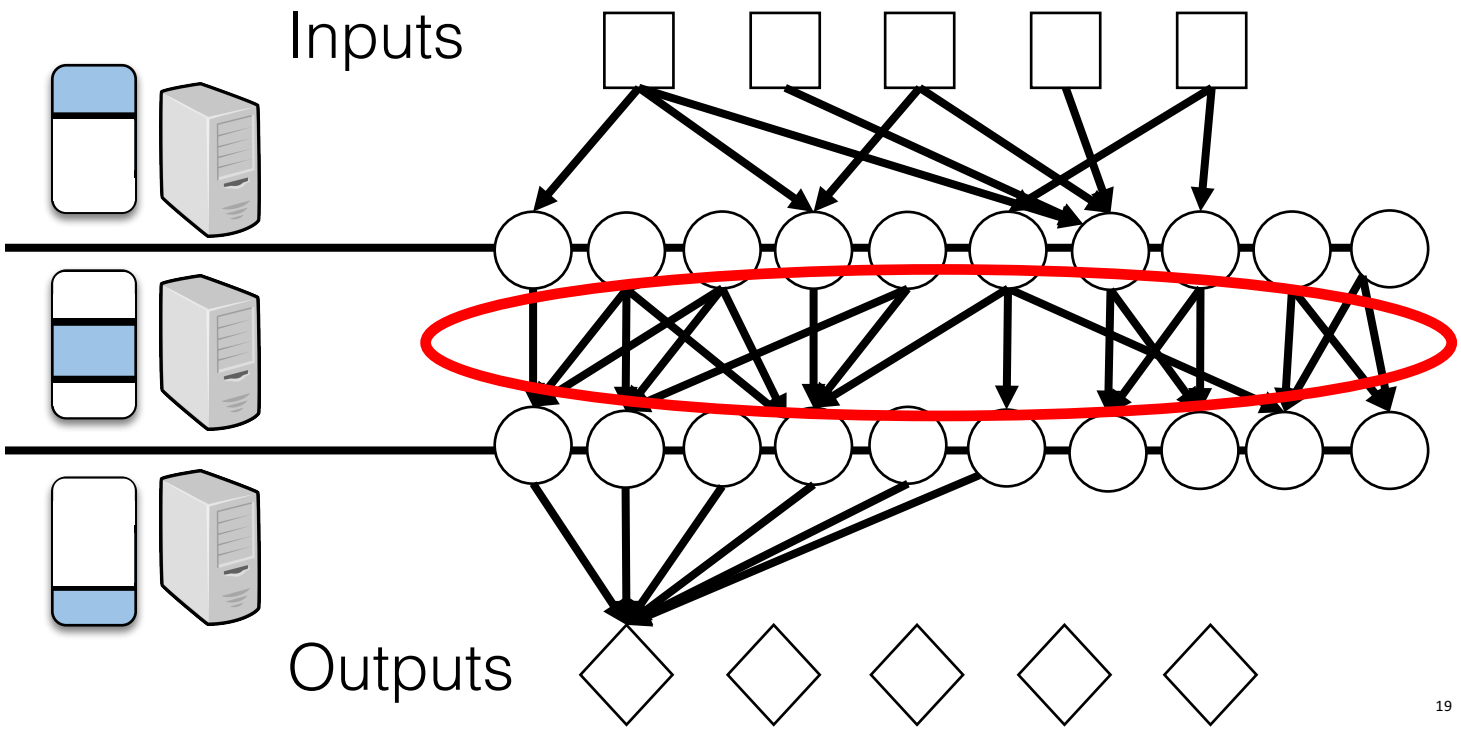
Track mapping between all intermediate locations



Local DIFT

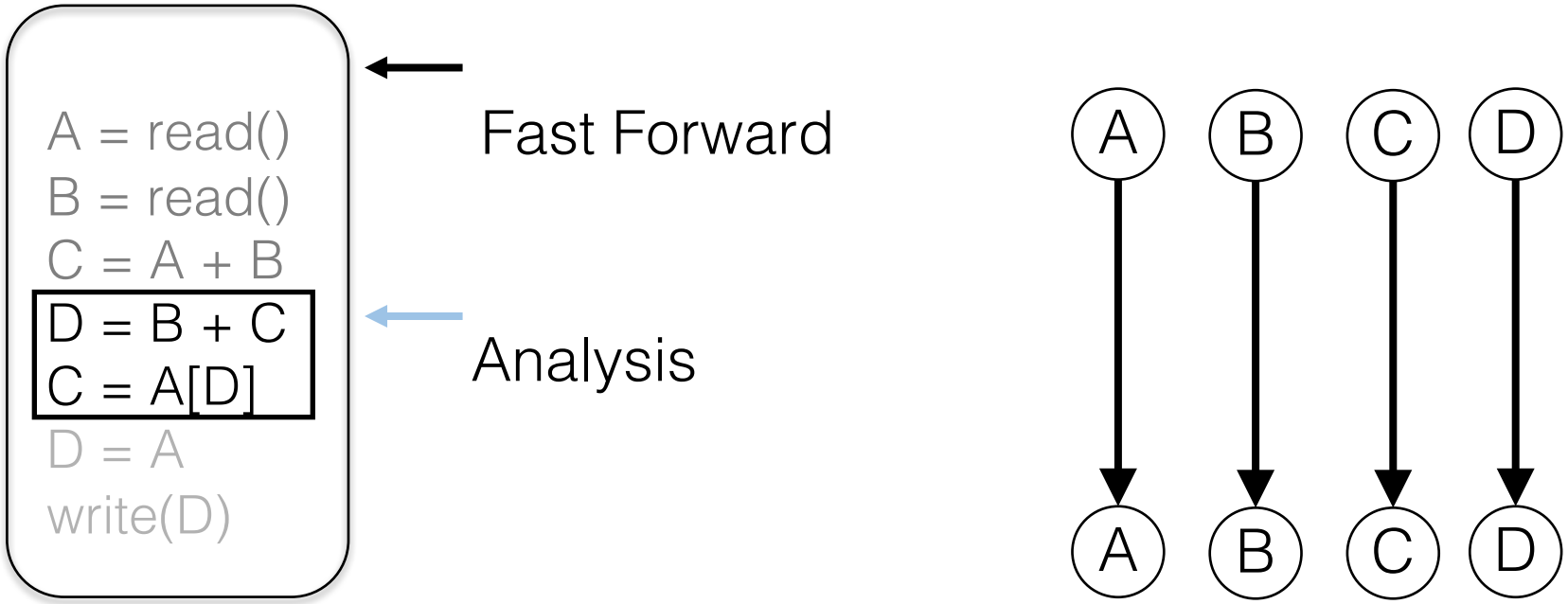
Mapping is too expensive to calculate:

- log operations
- defer calculating relationships until aggregation



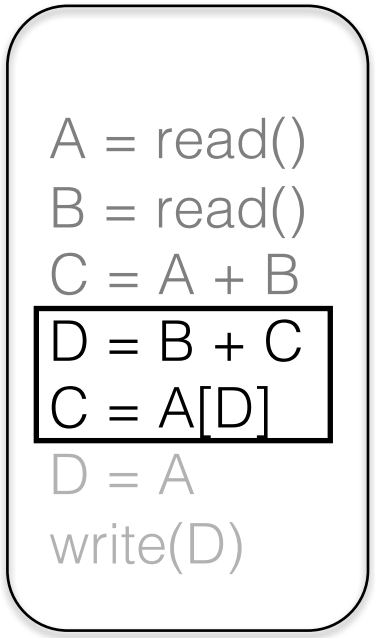
DIFT

- Fast Forward: replay execution until start of epoch
- Analysis: log operations using a graph



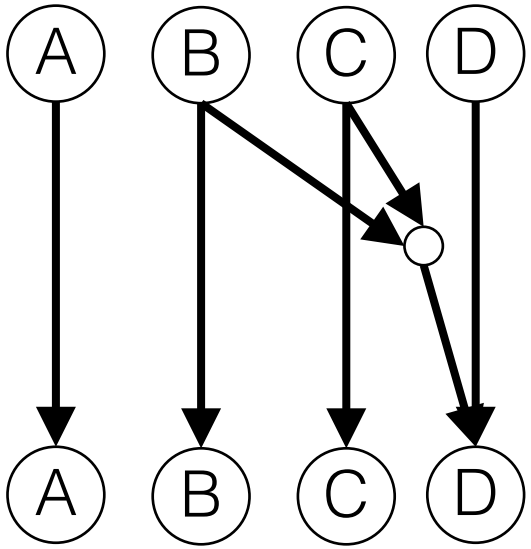
DIFT

- Fast Forward: replay execution until start of epoch
- Analysis: log operations using a graph



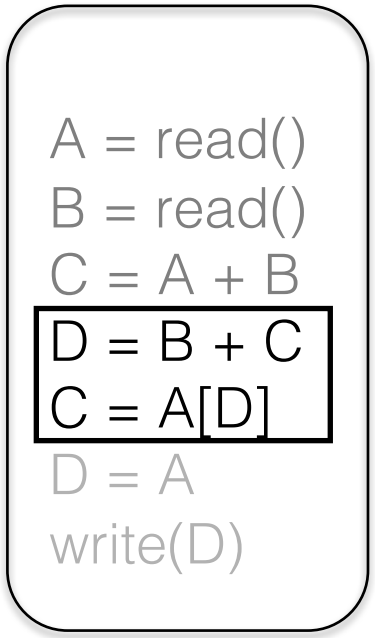
Fast Forward

Analysis



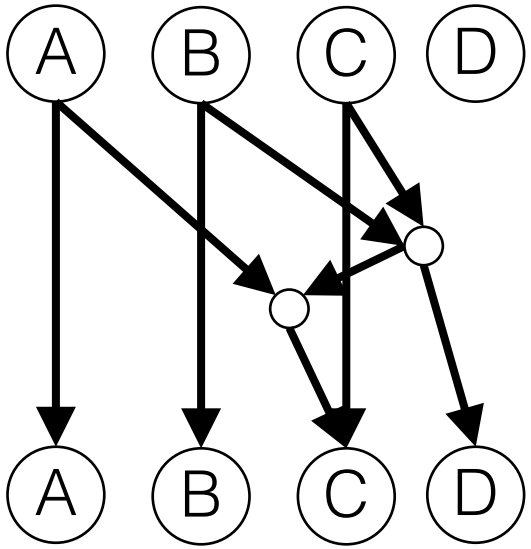
DIFT

- Fast Forward: replay execution until start of epoch
- Analysis: log operations using a graph

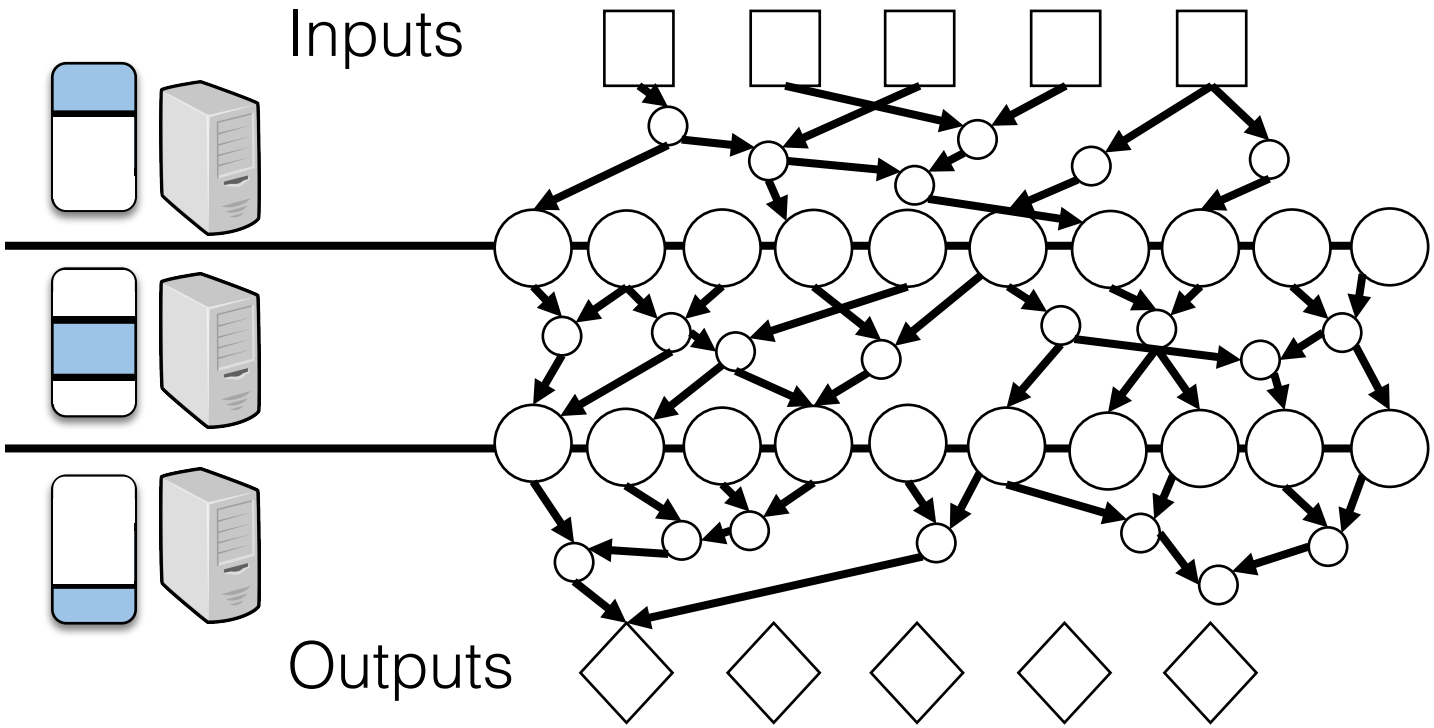


Fast Forward

Analysis

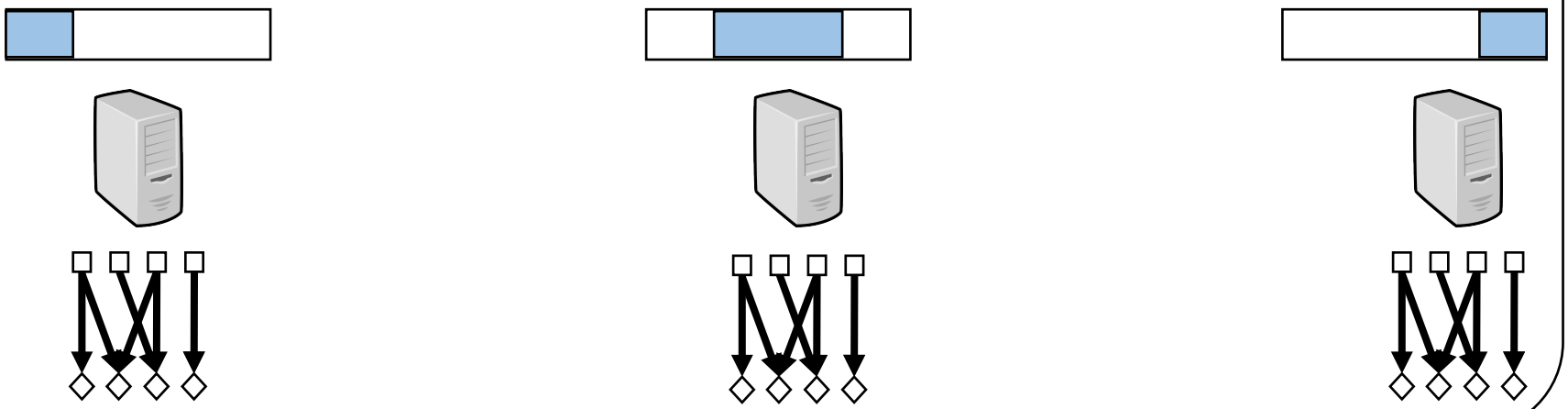


Local DIFT output

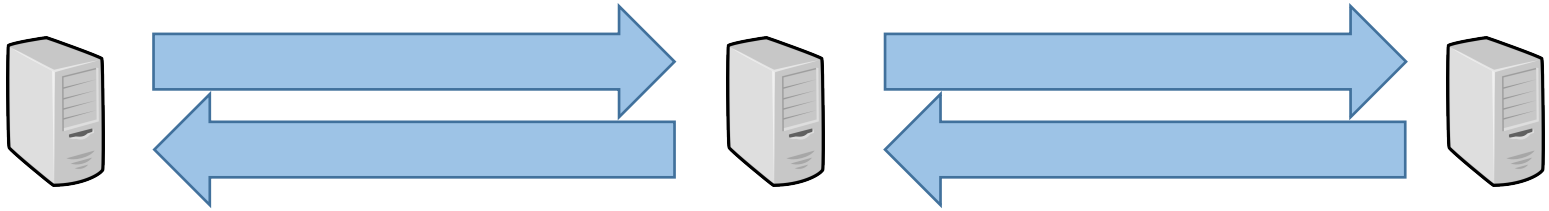


JetStream

Local DIFT – epoch parallelism



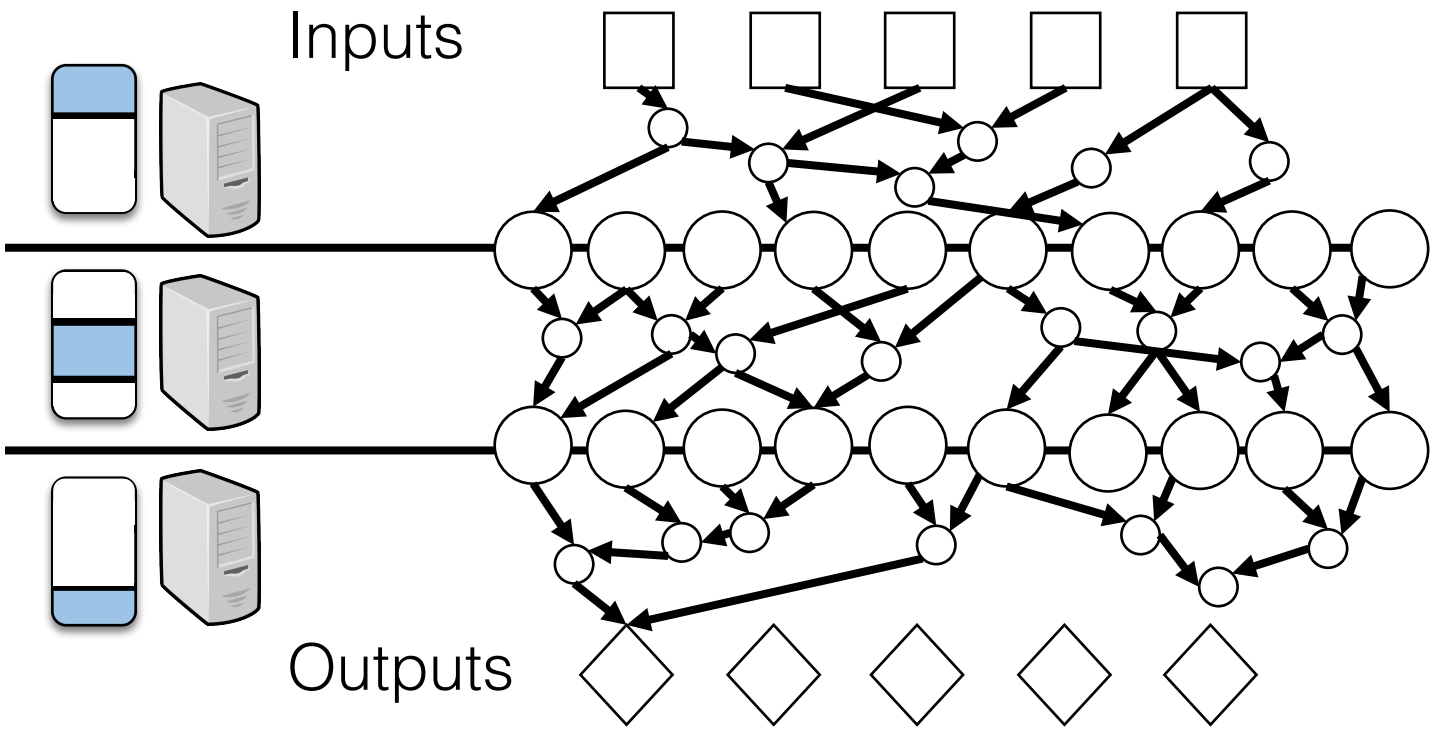
Aggregation – pipeline parallelism



Aggregation

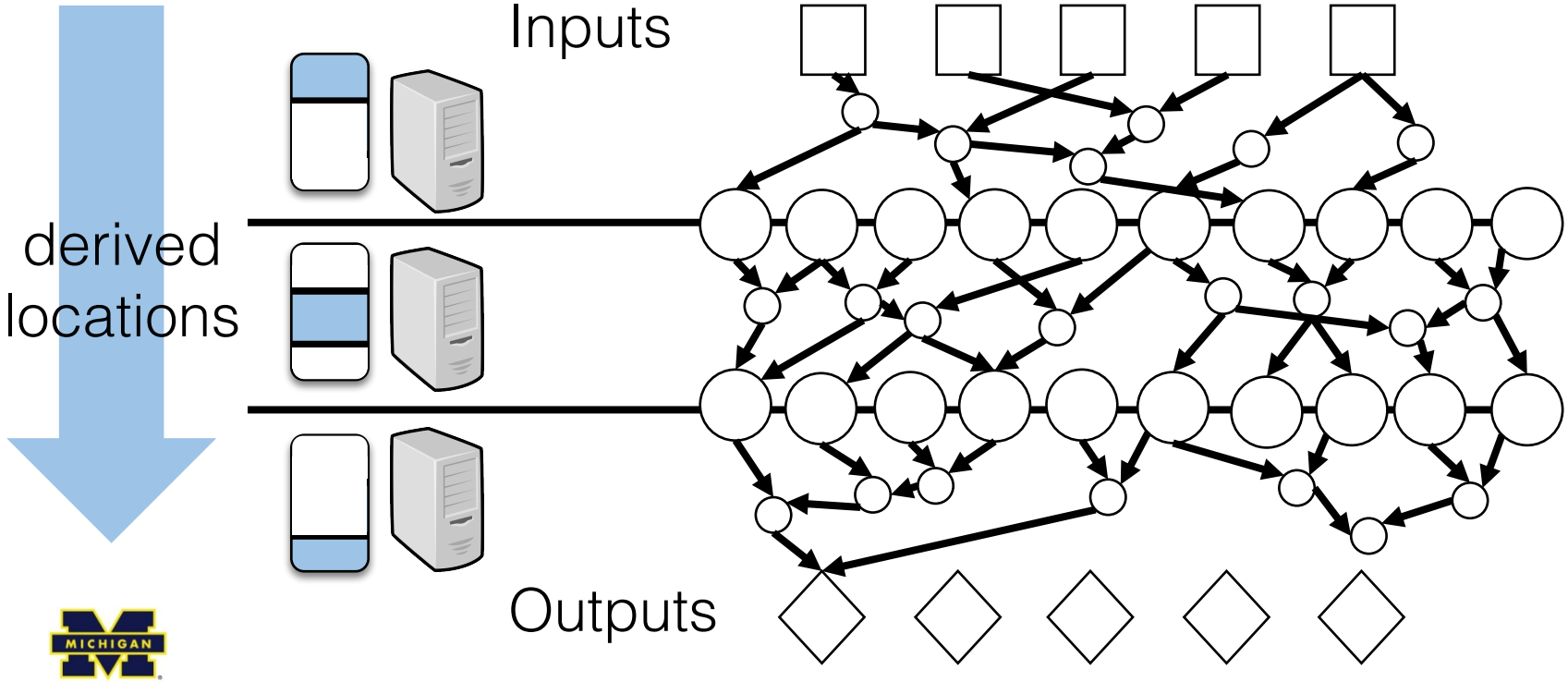
Calculate paths between source and sinks

- Many nodes are not on path between source and sink
- Use sequential information to prune work



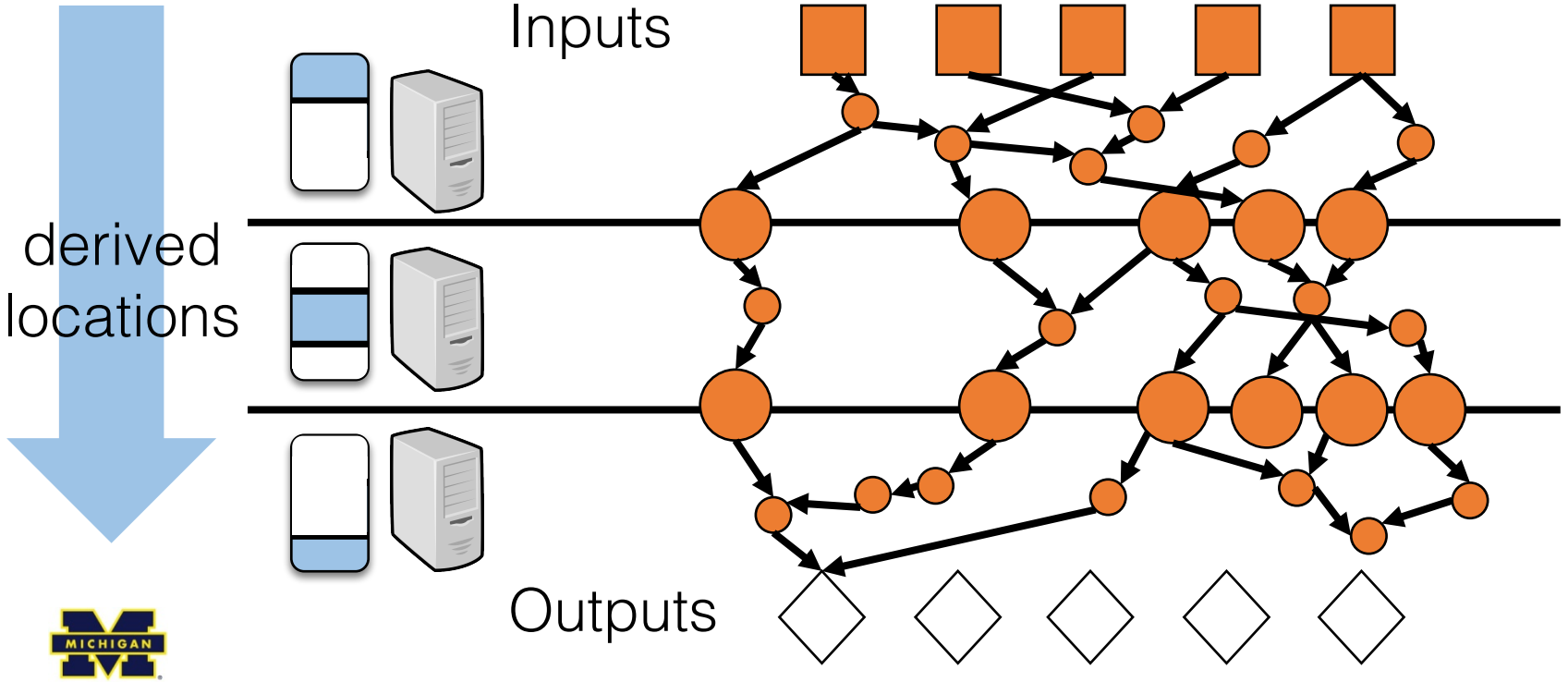
Forward Pass

Pass locations which are derived from a source



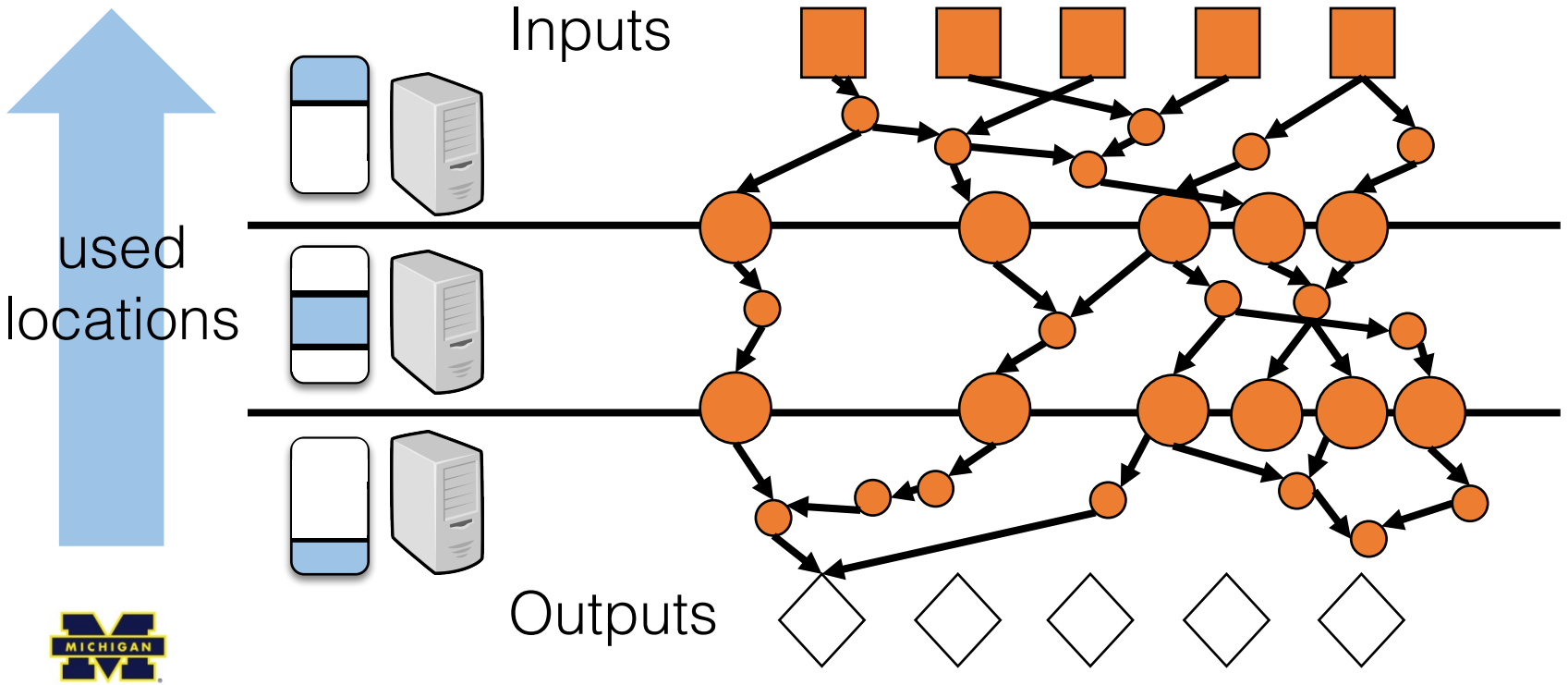
Forward Pass

Pass locations which are derived from a source



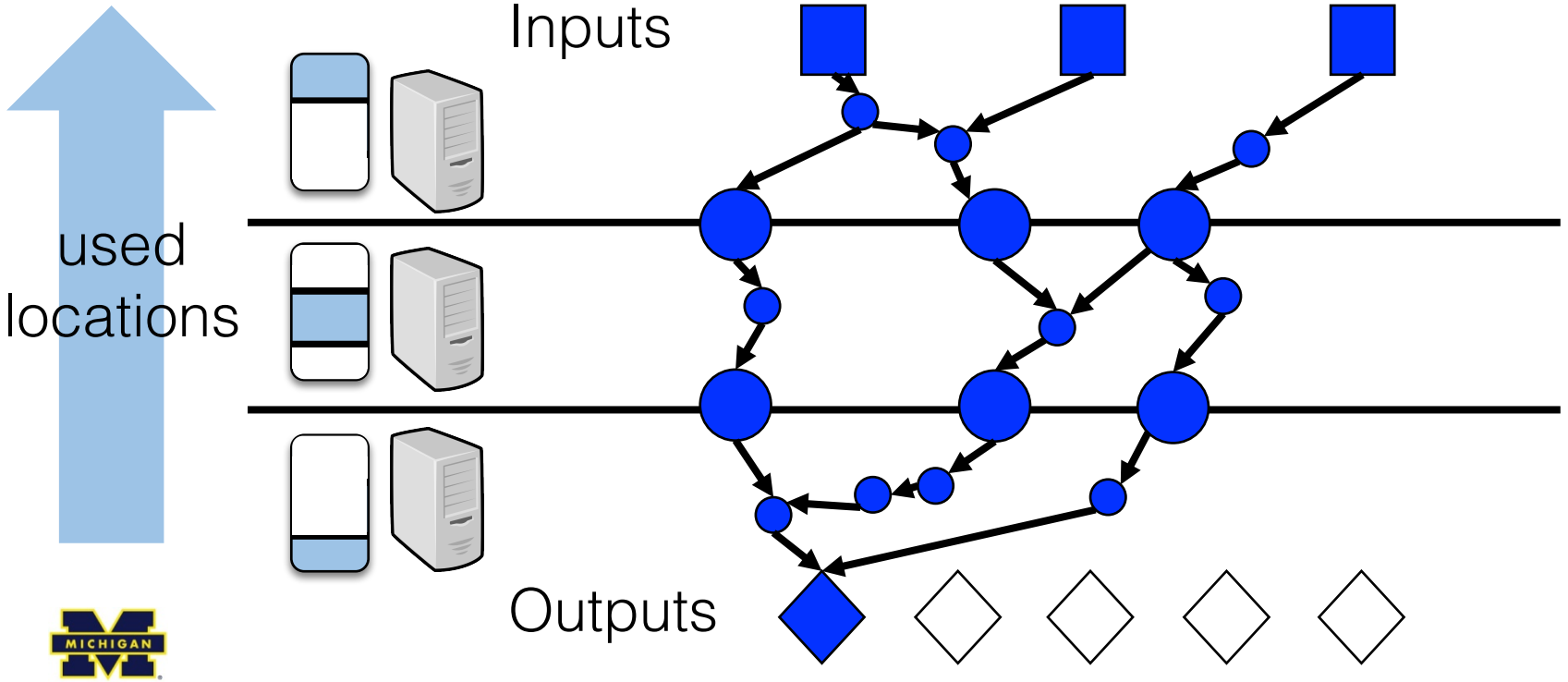
Backward Pass

Pass locations which are used by a sink



Backward Pass

Pass locations which are used by a sink



JetStream

In the paper:

- Insights about why naïve approaches fail
- Partitioning – challenging to predict the local DIFT time
- Pre-pruning – garbage collection of the graph

Outline

- Motivation and Introduction
- Design of JetStream
  - Local DIFT
  - Aggregation
- Evaluation

Experimental Setup

- CloudLab cluster of 32 machines, 1–128 cores

Benchmark	Sequential DIFT Time (Minutes)	Sources (millions)	Sinks (millions)
Ghostscript	1.3	3	0.2
Gzip	1.8	64	488
Evince	3.9	10	104
Nginx	3.3	10	35
Mongodb	5.2	9	117
OpenOffice	7.0	10	32
Firefox	30.6	0.9	2

Experimental Setup

- CloudLab cluster of 32 machines, 1–128 cores

Benchmark	Sequential DIFT Time (Minutes)	Sources (millions)	Sinks (millions)
Ghostscript	1.3	3	0.2
Gzip			
Evince			
Nginx			
Mongodb	5.2	9	117
OpenOffice			
Firefox			

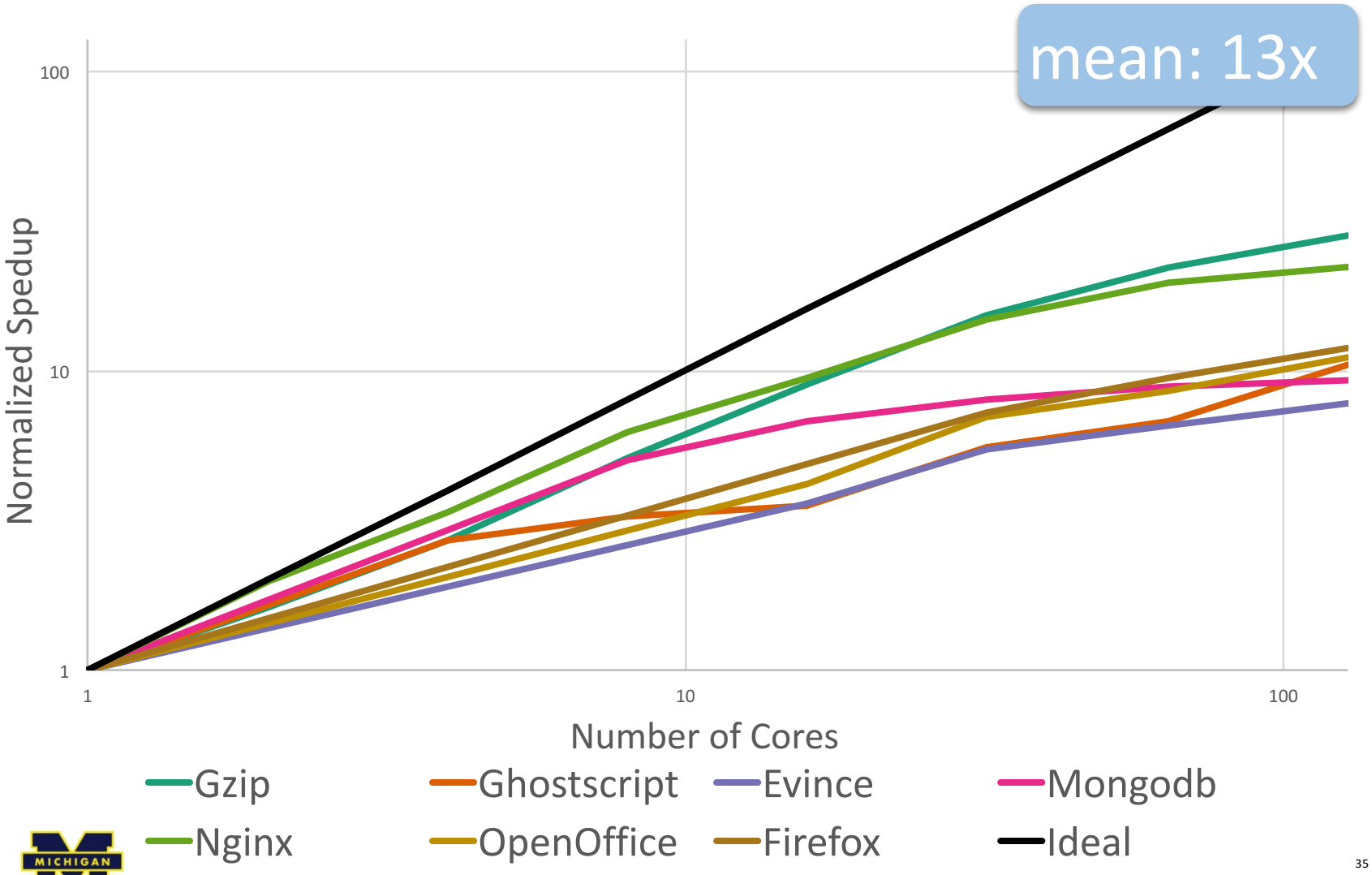
sources: home directory
sinks: all

sources: Cookies
sinks: suspicious connections

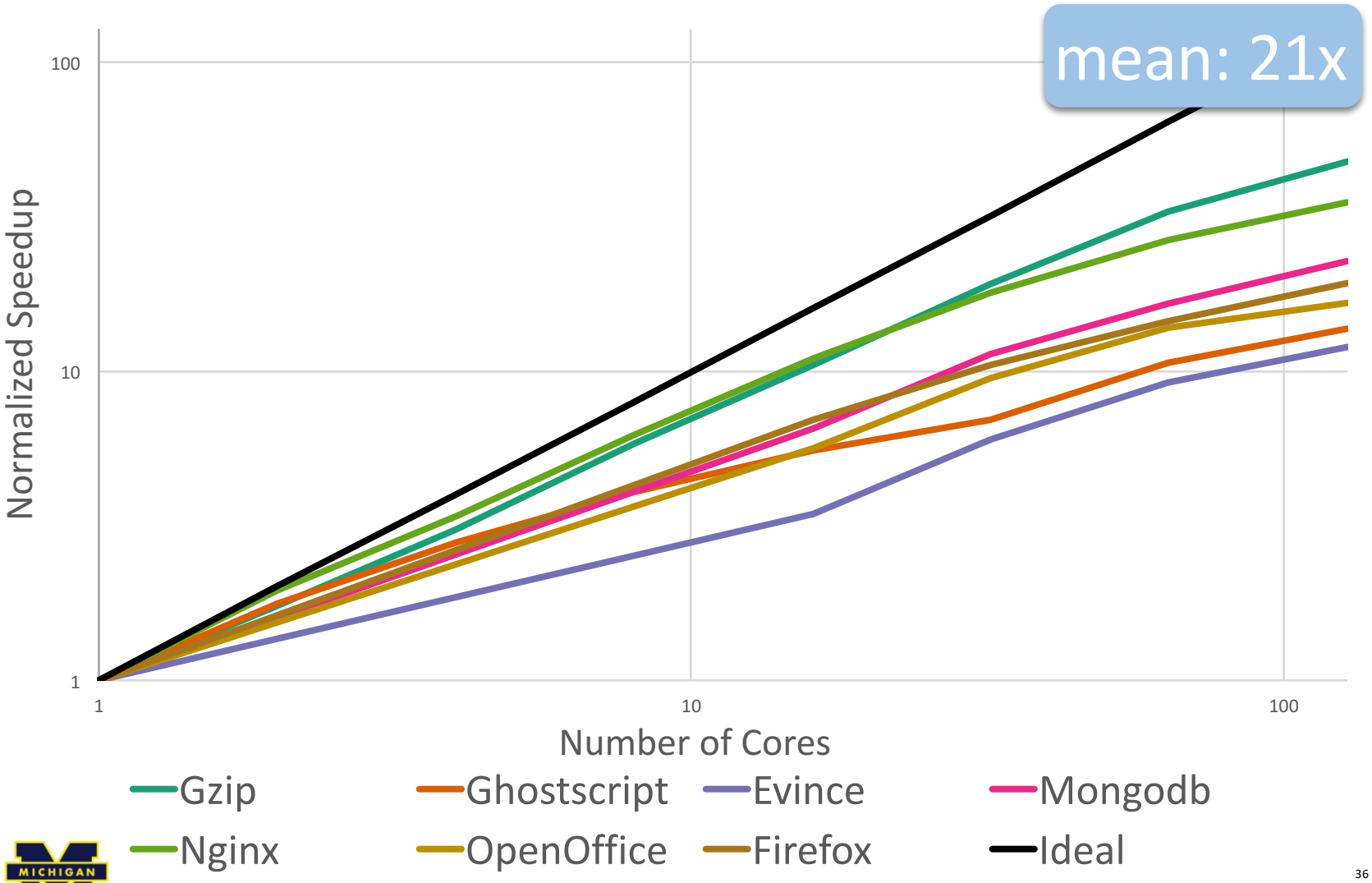
Two Different Scenarios

- Unexpected analysis:
  - prioritize low record overhead
- Expected analysis
  - periodic checkpoint
  - gather partitioning stats

Scalability of Unexpected Analysis



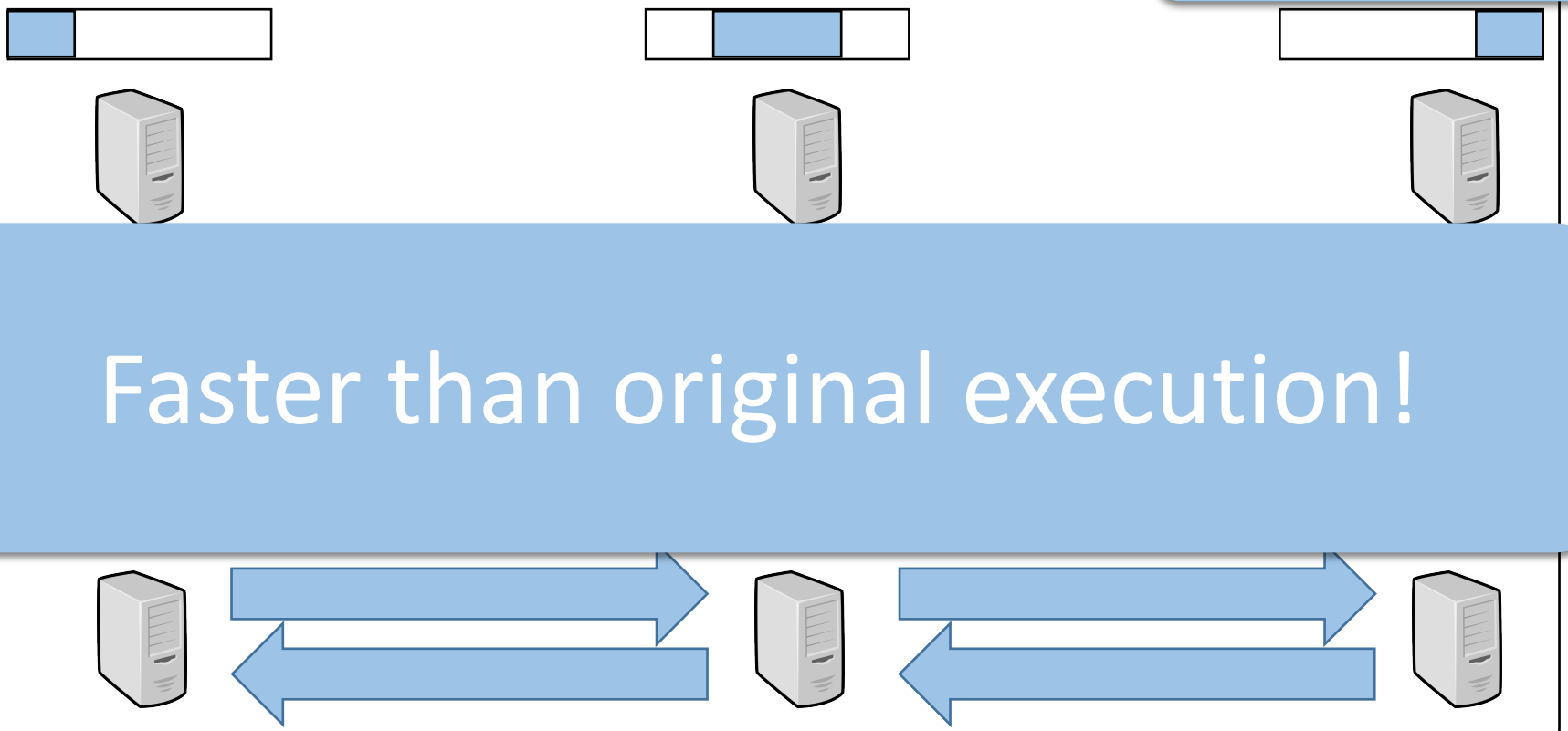
Scalability of Expected Analysis



JetStream

2x → 21x

Local DIFT – epoch parallelism



Faster than original execution!

Questions

Related Work

Epoch Parallelism

- Wallace and Hazelwood, “SuperPin: Parallelizing Dynamic Instrumentation for Real-Time Performance”

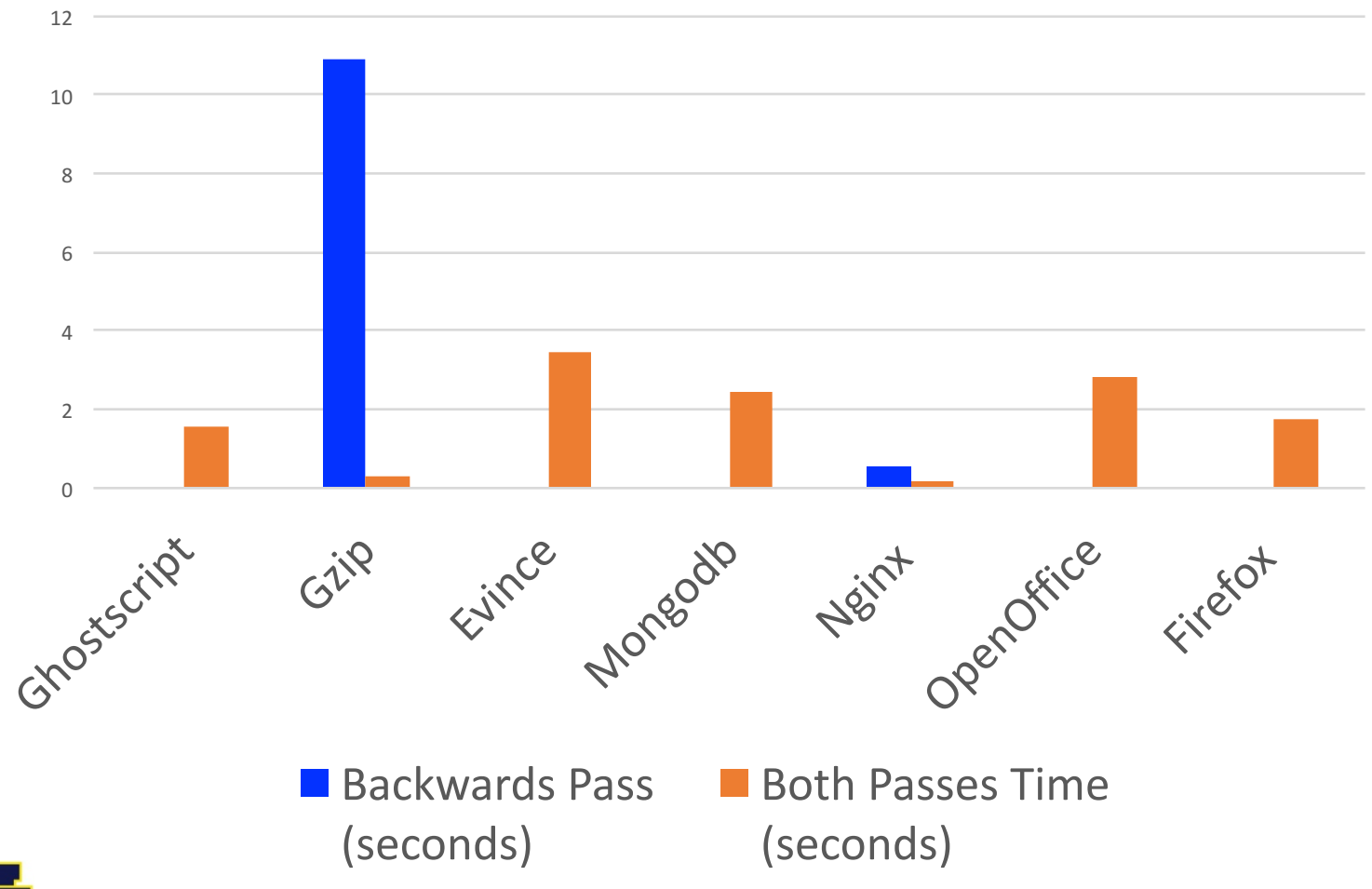
Local DIFT

- Ruwase et al. “Parallelizing Dynamic Information Flow Tracking”
- Nightengale et al. “Parallelizing security checks on commodity hardware”

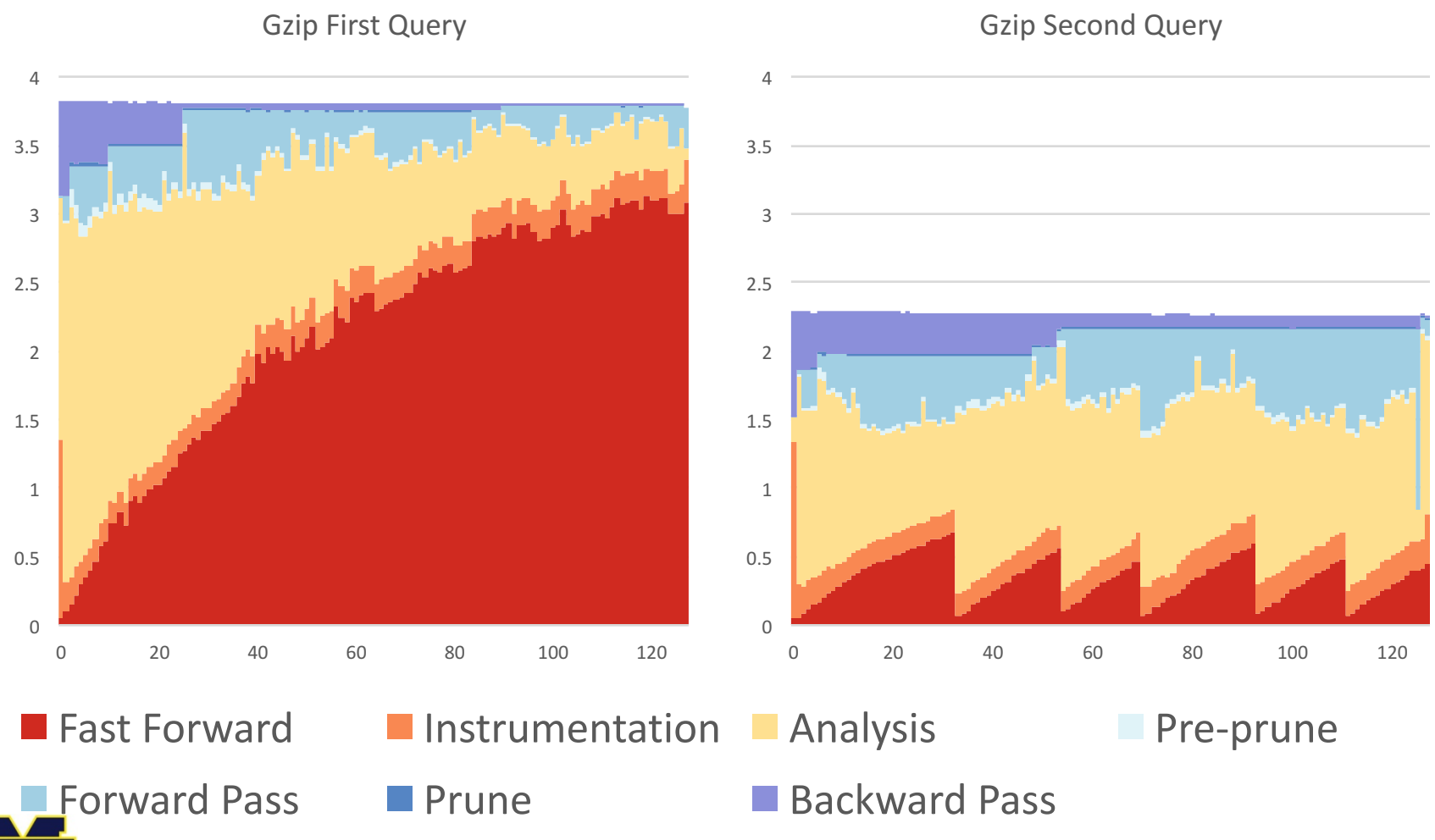
Benchmarks

Benchmark	Replay Time (seconds)	JetStream Time (seconds)
Ghostscript	1.0	5.6
Gzip	3.0	2.3
Evince	13.5	19.5
Nginx	4.8	5.6
Mongodb	22.8	13.8
OpenOffice	7.6	25.0
Firefox	67.4	94.4

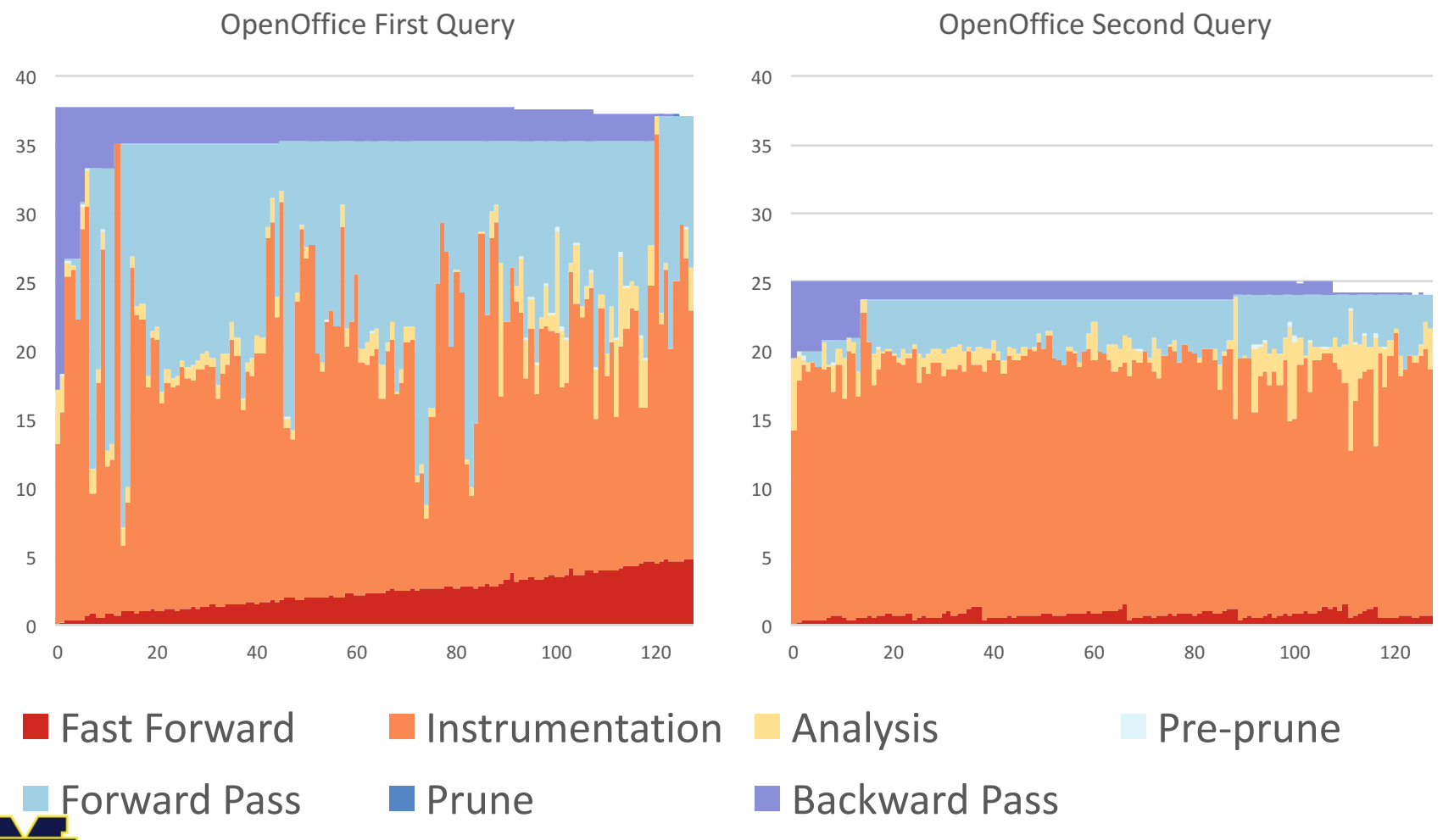
Aggregation Results



Gzip


Gzip First Query
Gzip Second Query
4
3.5
3
2.5
2
1.5
1
0.5
0
0
20
40
60
80
100
120
Fast Forward
Instrumentation
Analysis
Pre-prune
Forward Pass
Prune
Backward Pass

OpenOffice


OpenOffice First Query
OpenOffice Second Query
40
35
30
25
20
15
10
5
0
0
20
40
60
80
100
120
Fast Forward
Instrumentation
Analysis
Pre-prune
Forward Pass
Prune
Backward Pass