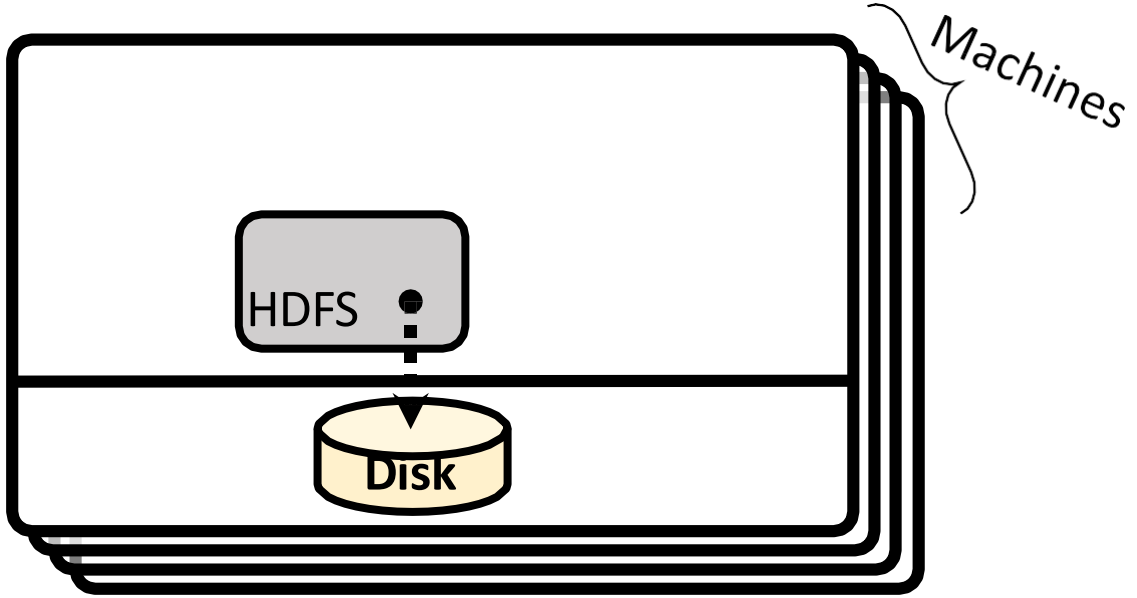
CE 528 Cloud Computing

Lecture 16: Performance Profiling
Spring 2026

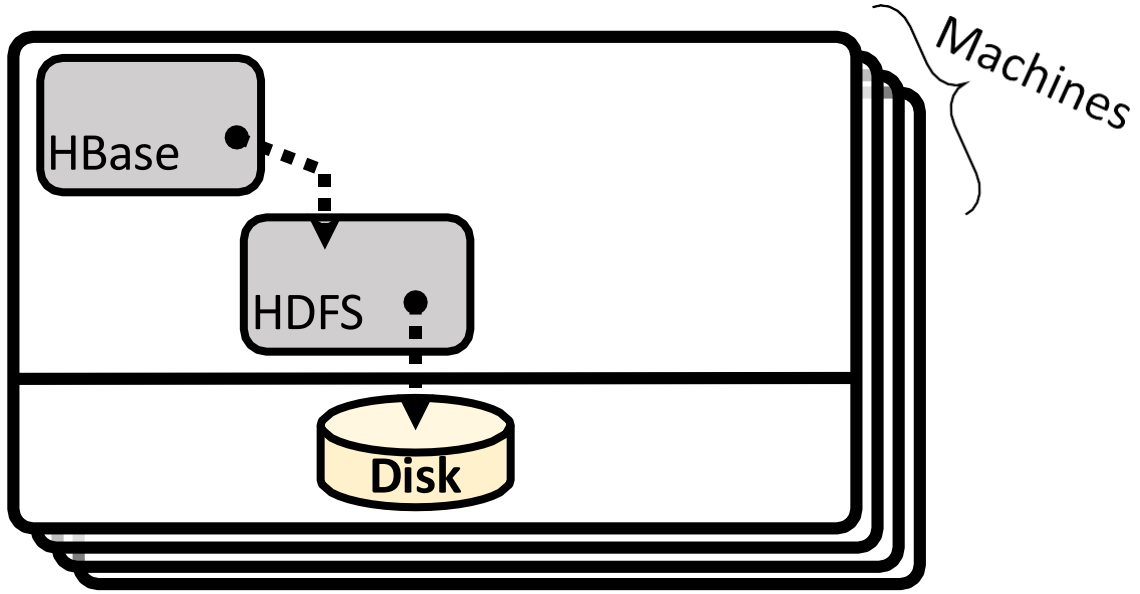
Prof. Yigong Hu

Slides courtesy of Michael Chow and Jonathan Mace

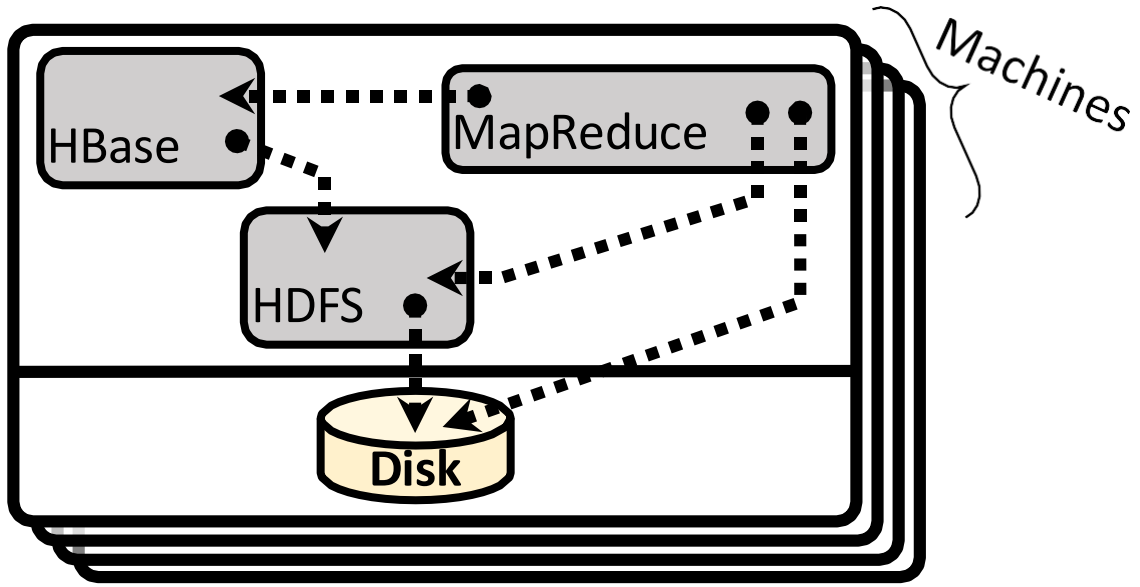
Case Study: Hadoop



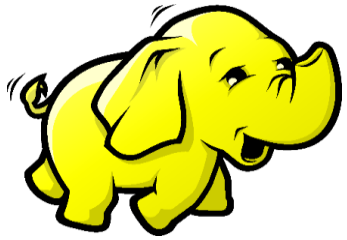
Case Study: Hadoop

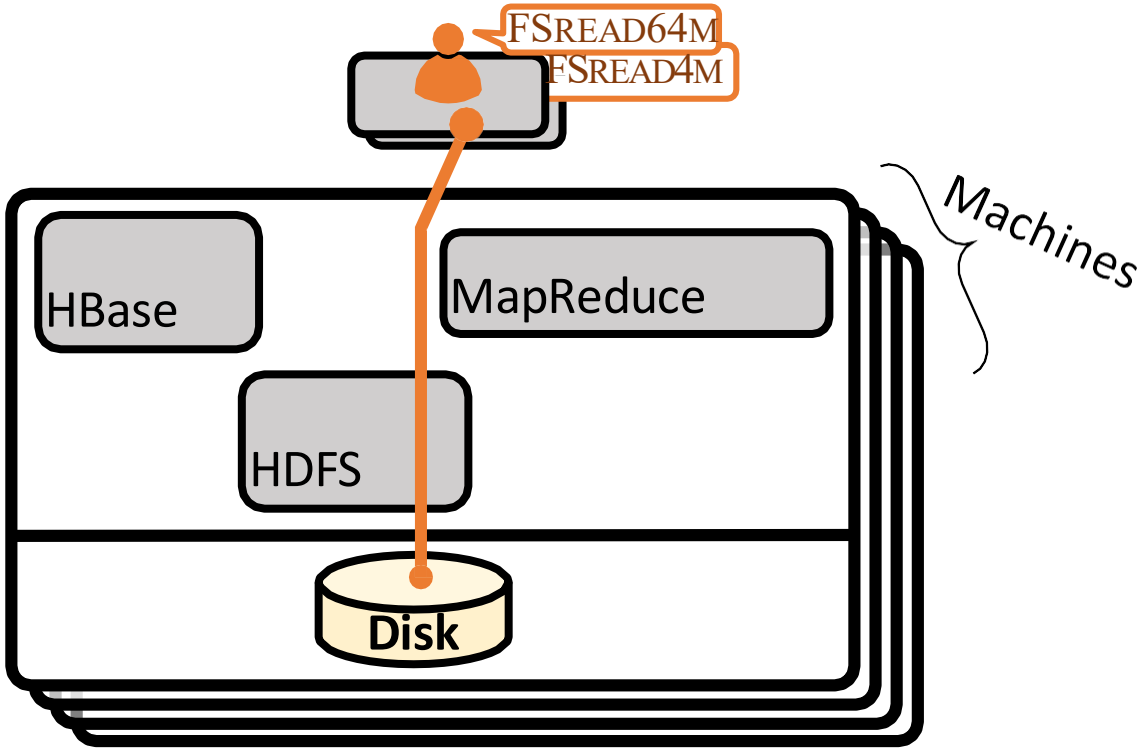


Case Study: Hadoop

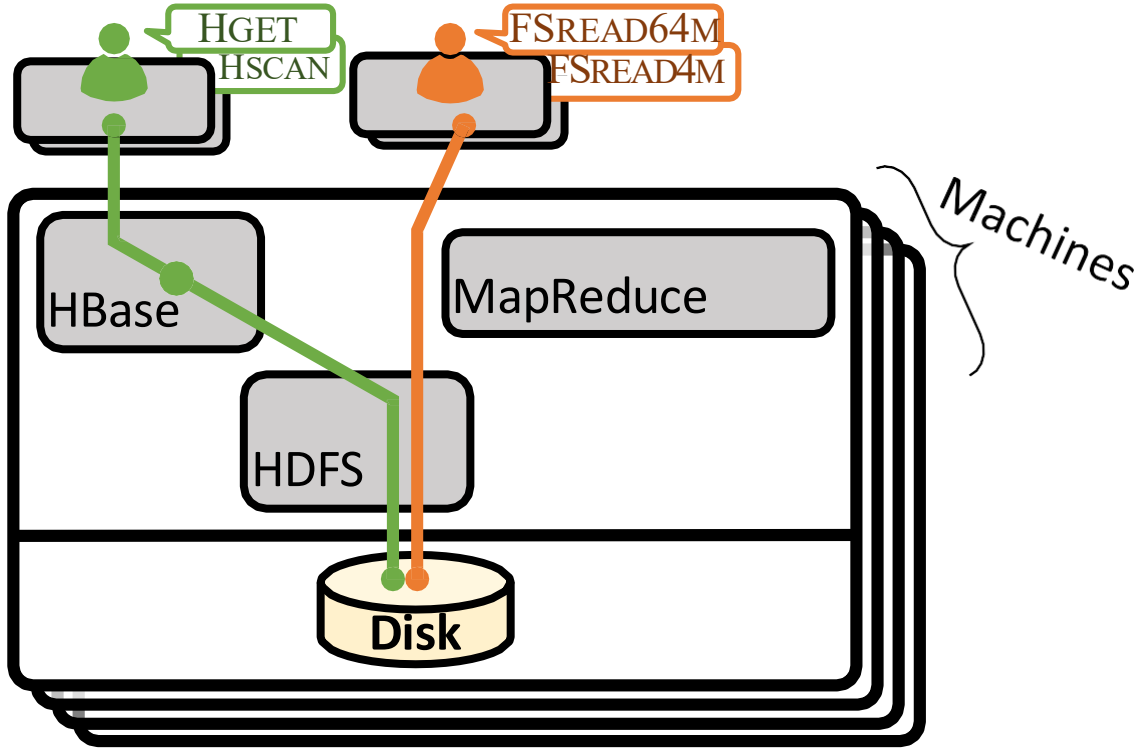


Case Study: Hadoop

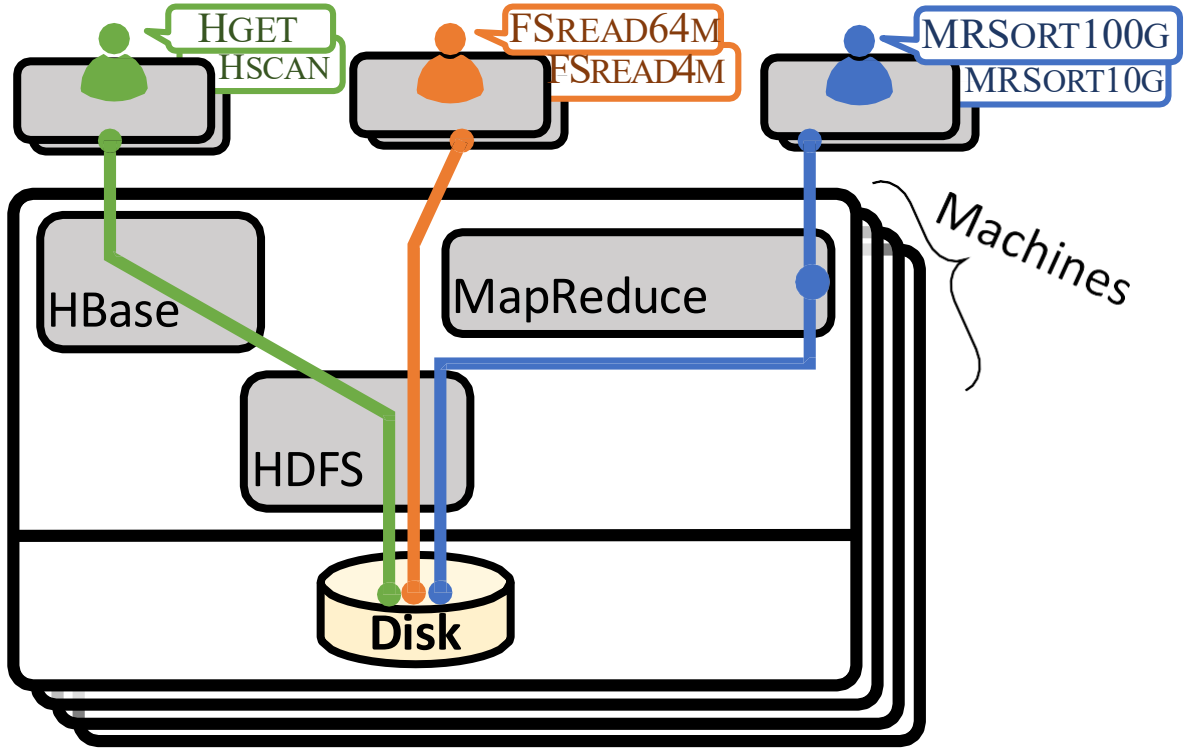




Case Study: Hadoop

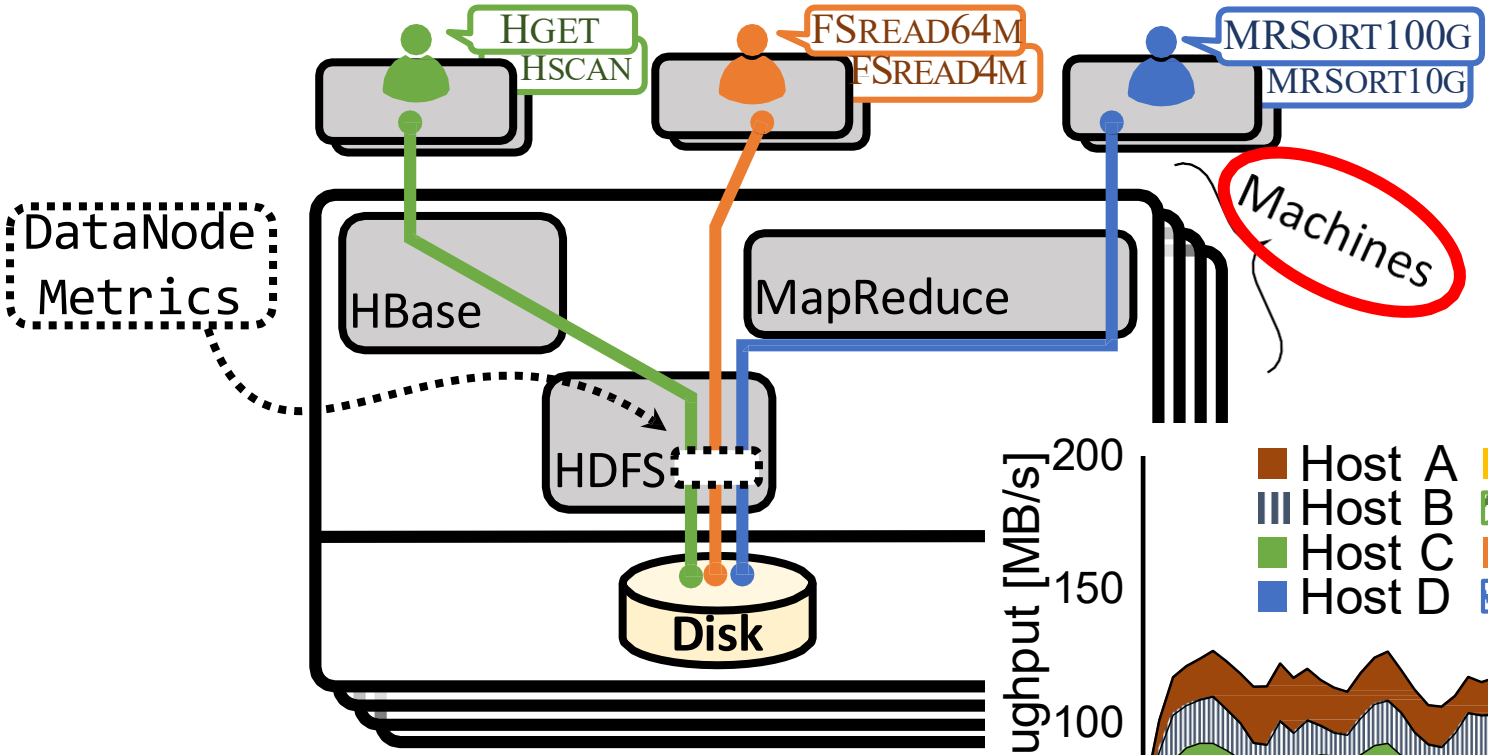


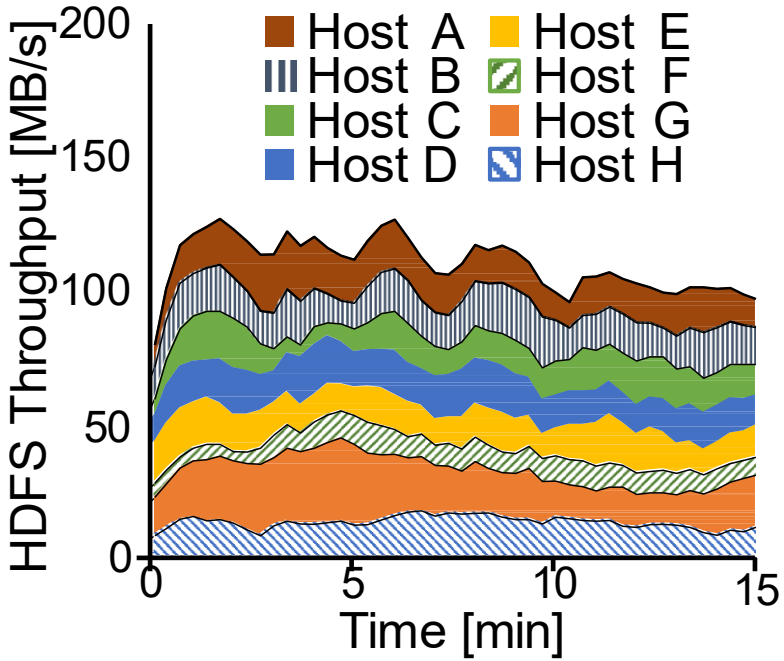
Case Study: Hadoop



How is disk bandwidth being used?

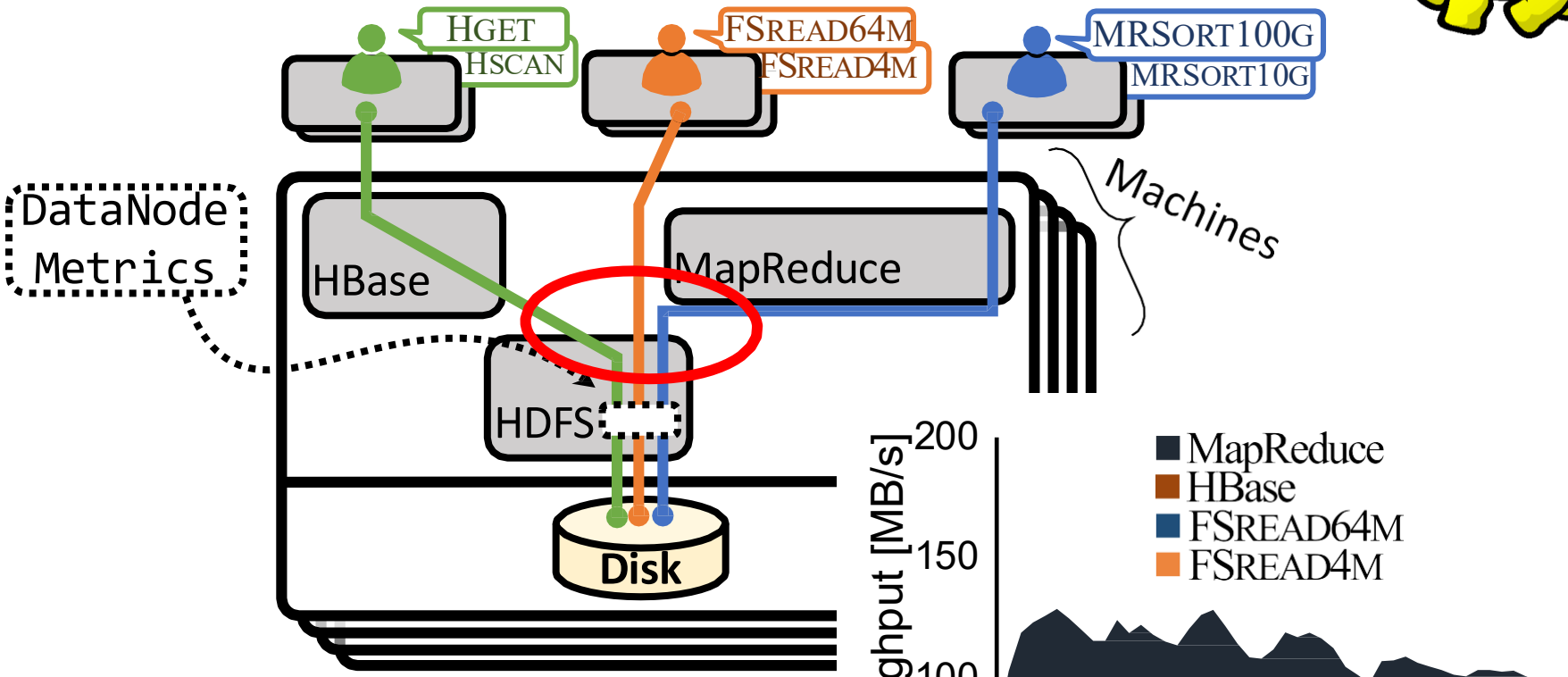
Case Study: Hadoop

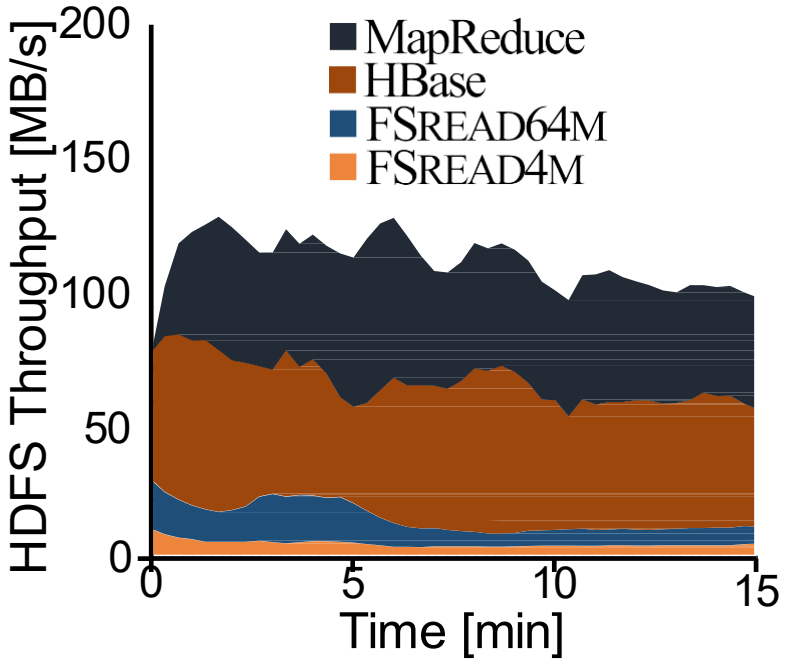




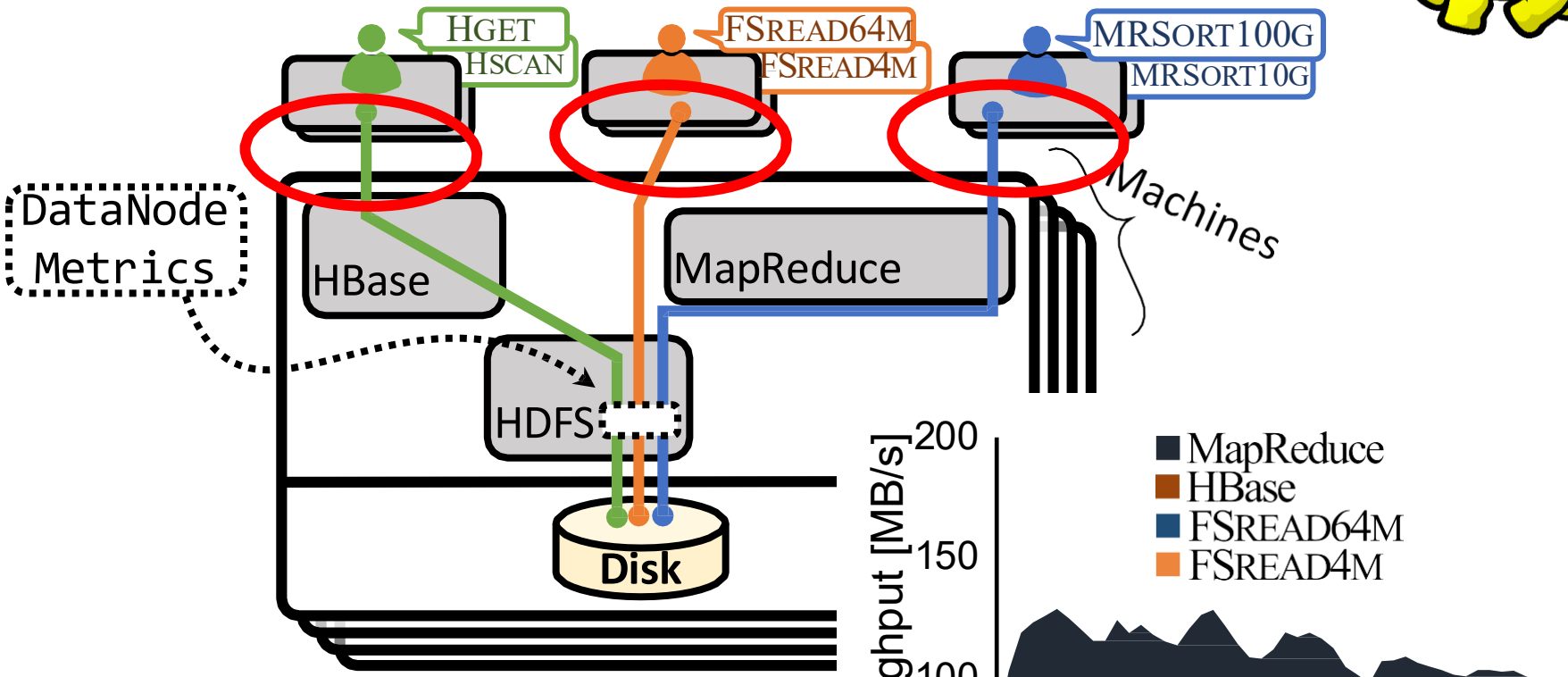
How is disk bandwidth being used?

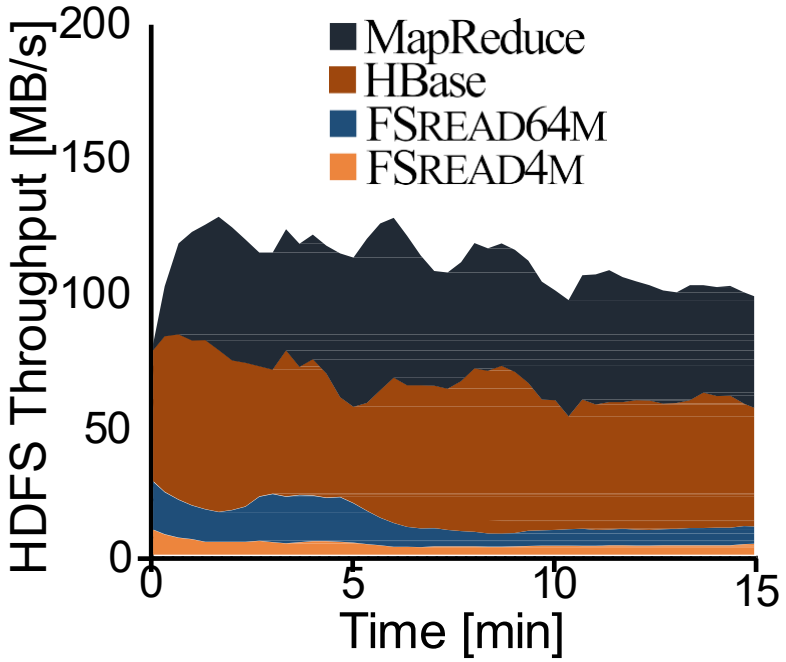
Case Study: Hadoop



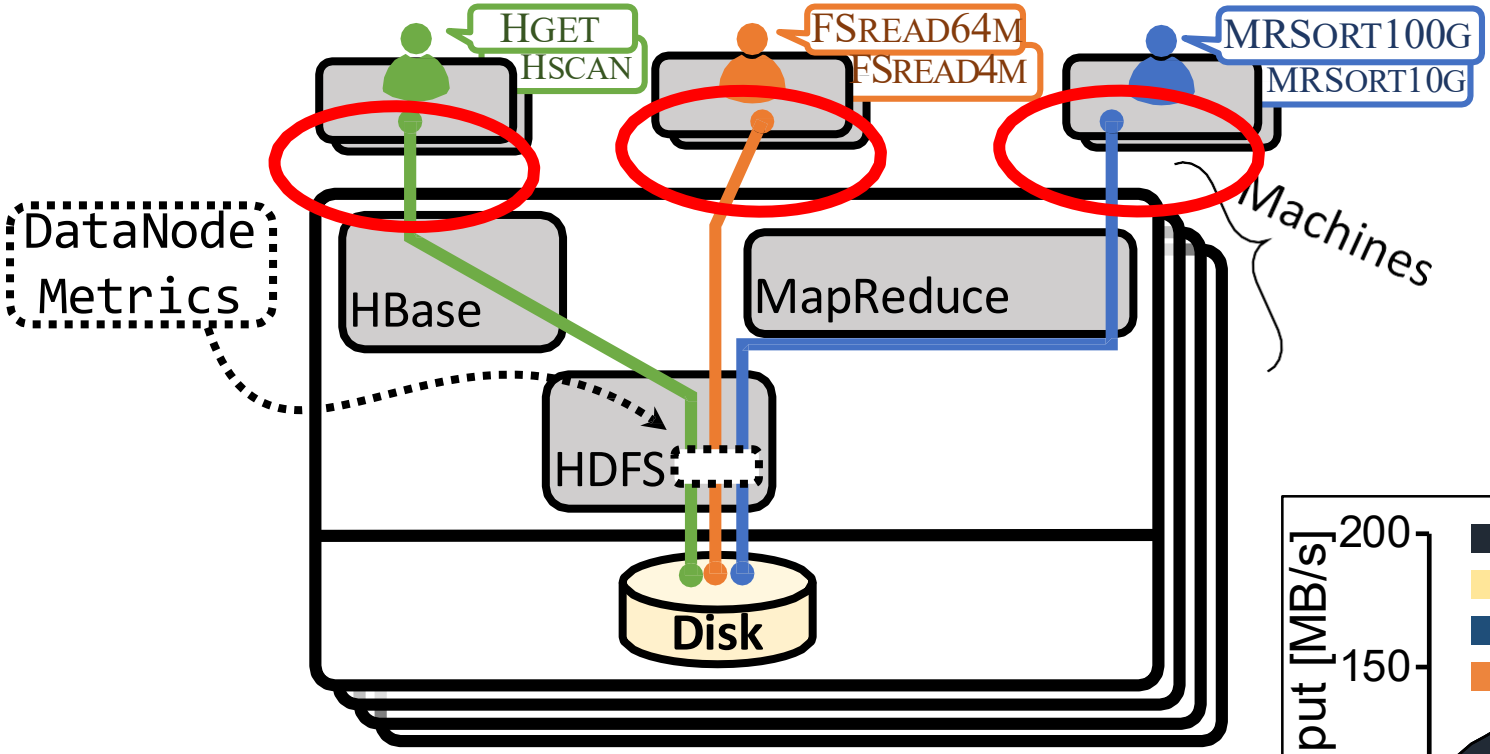


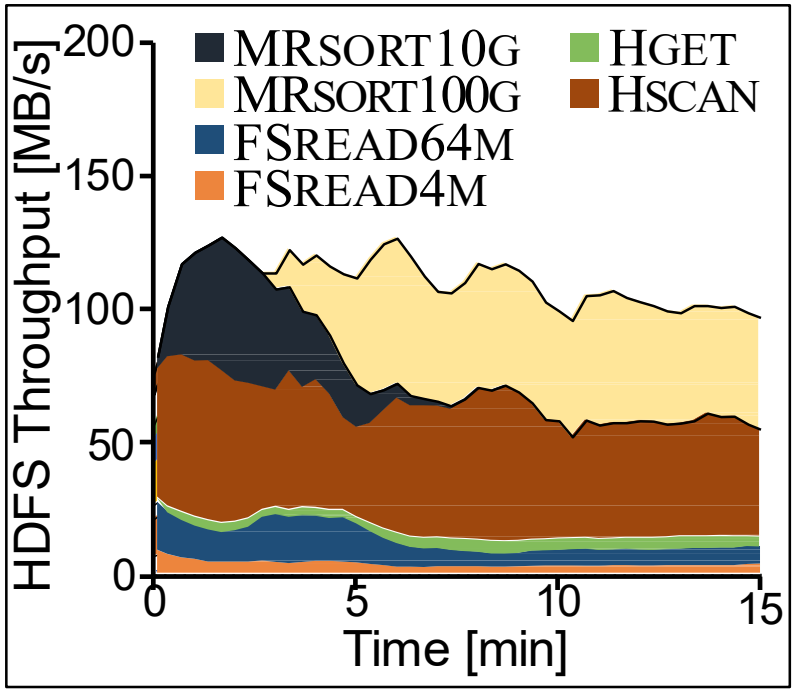
Case Study: Hadoop



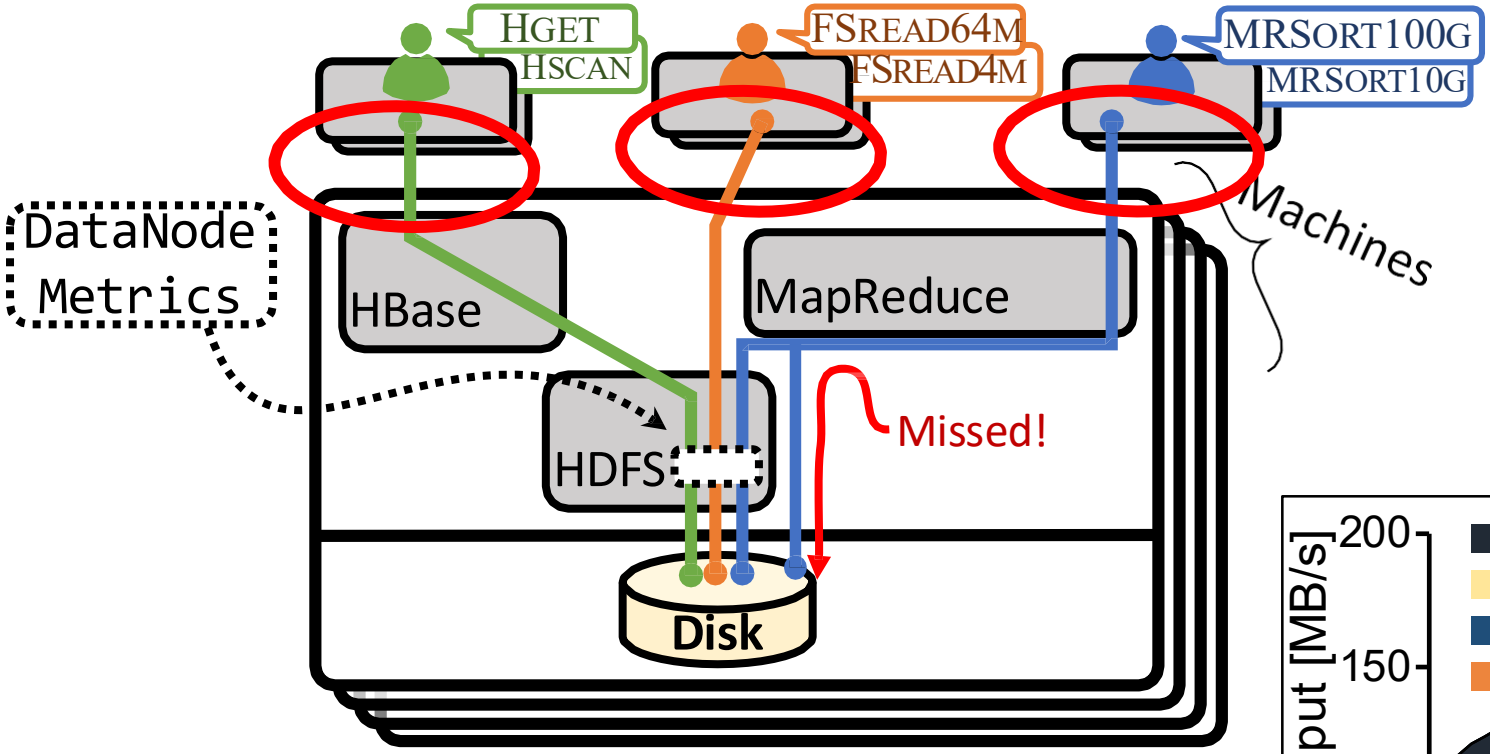


Case Study: Hadoop

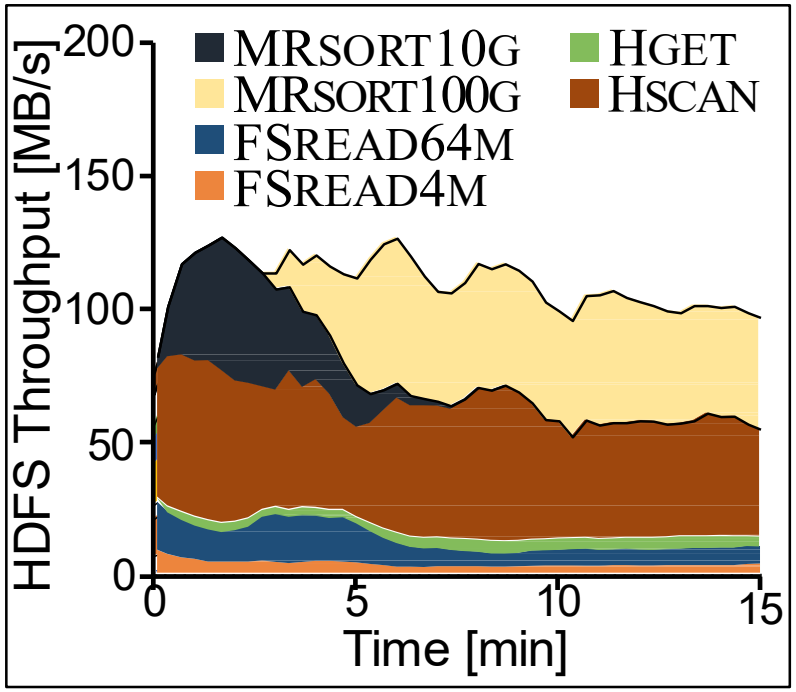




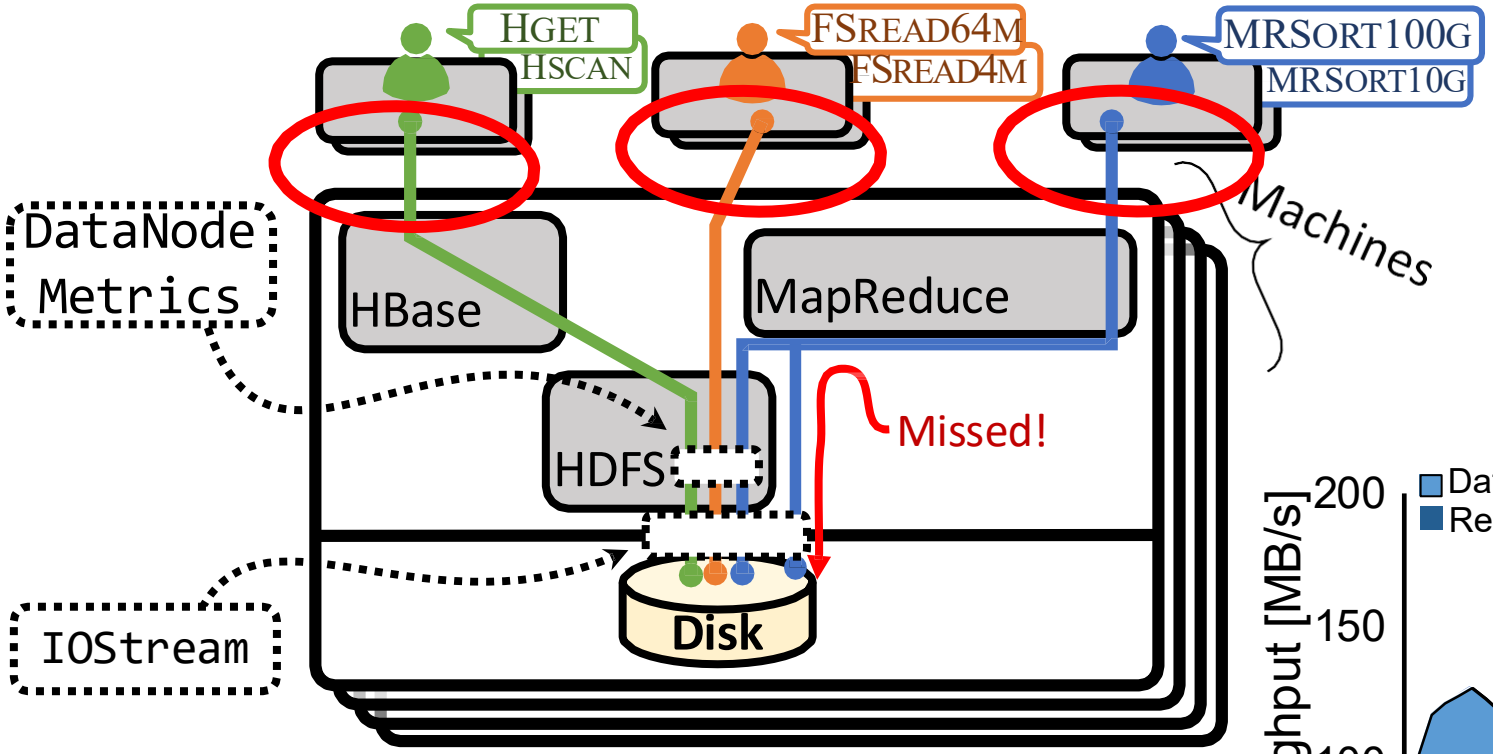
Case Study: Hadoop

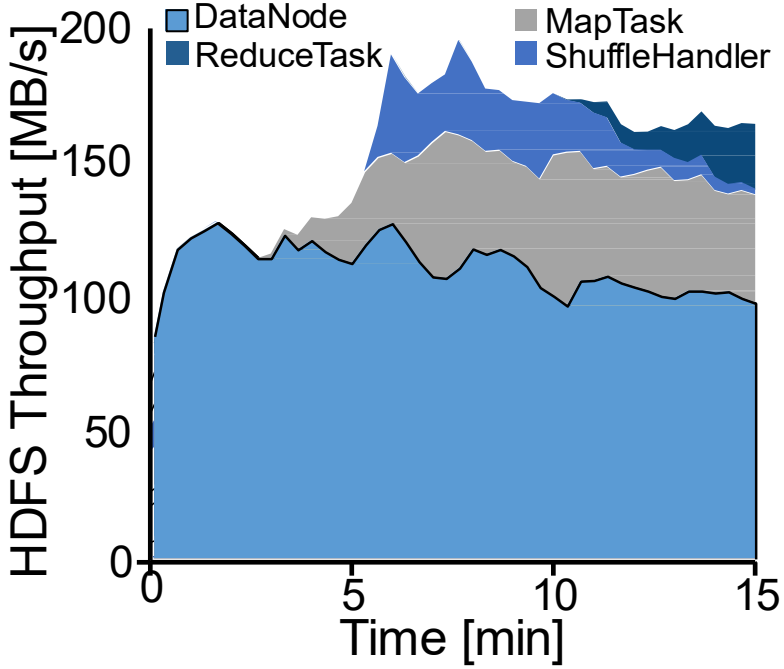


How is disk bandwidth being used?



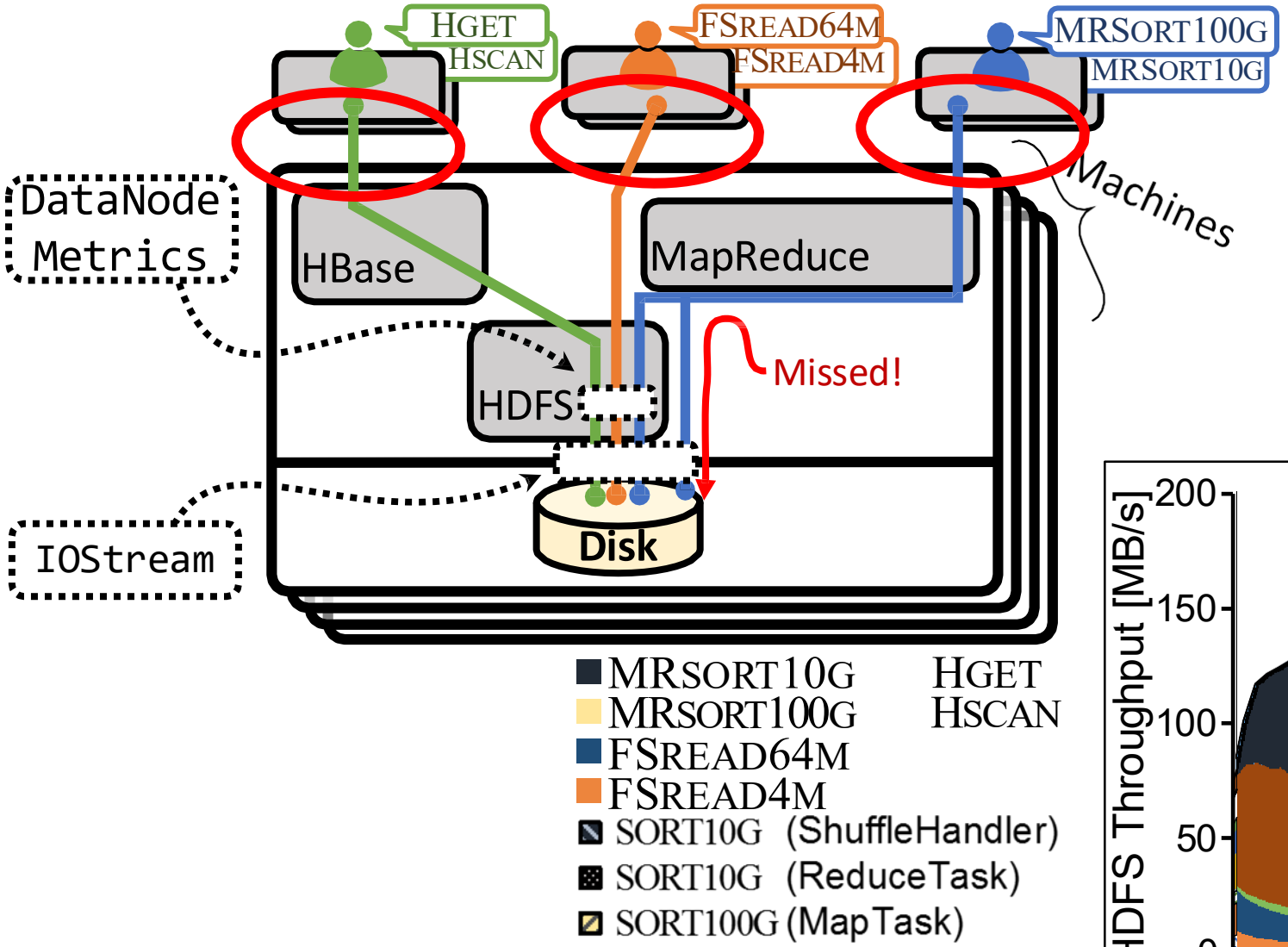
Case Study: Hadoop

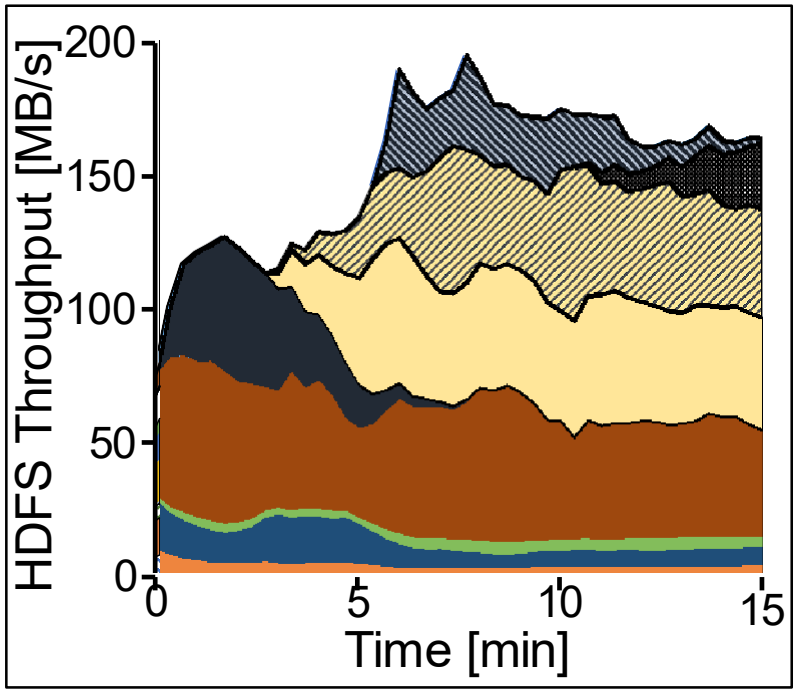




How is disk bandwidth being used?

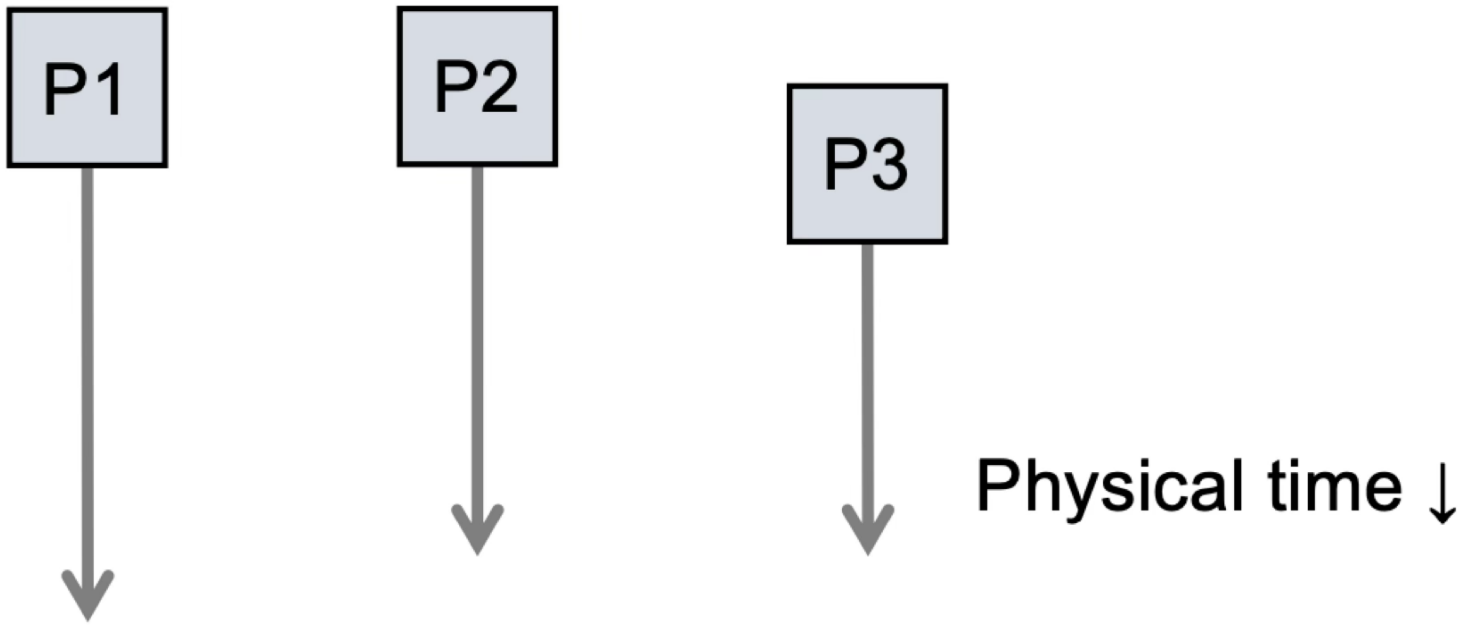
Case Study: Hadoop





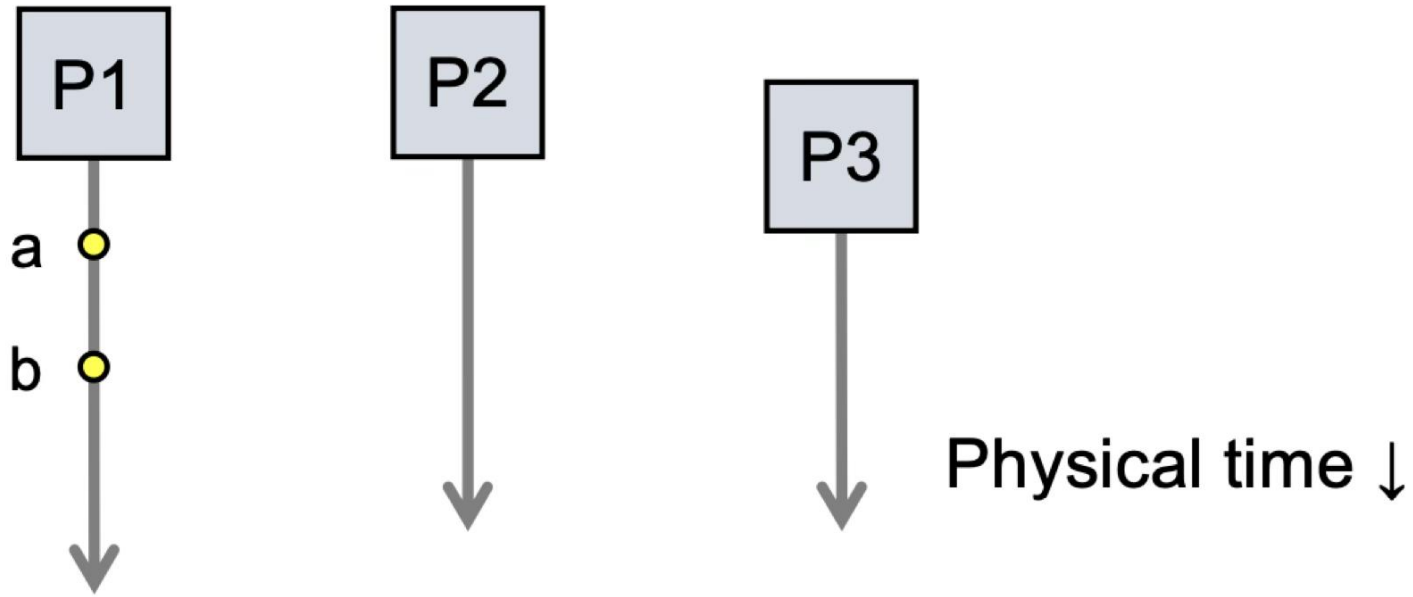
Happens-Before

- Consider three processes: P1, P2, and P3
- Notation: Event a happens before event b (a->b)



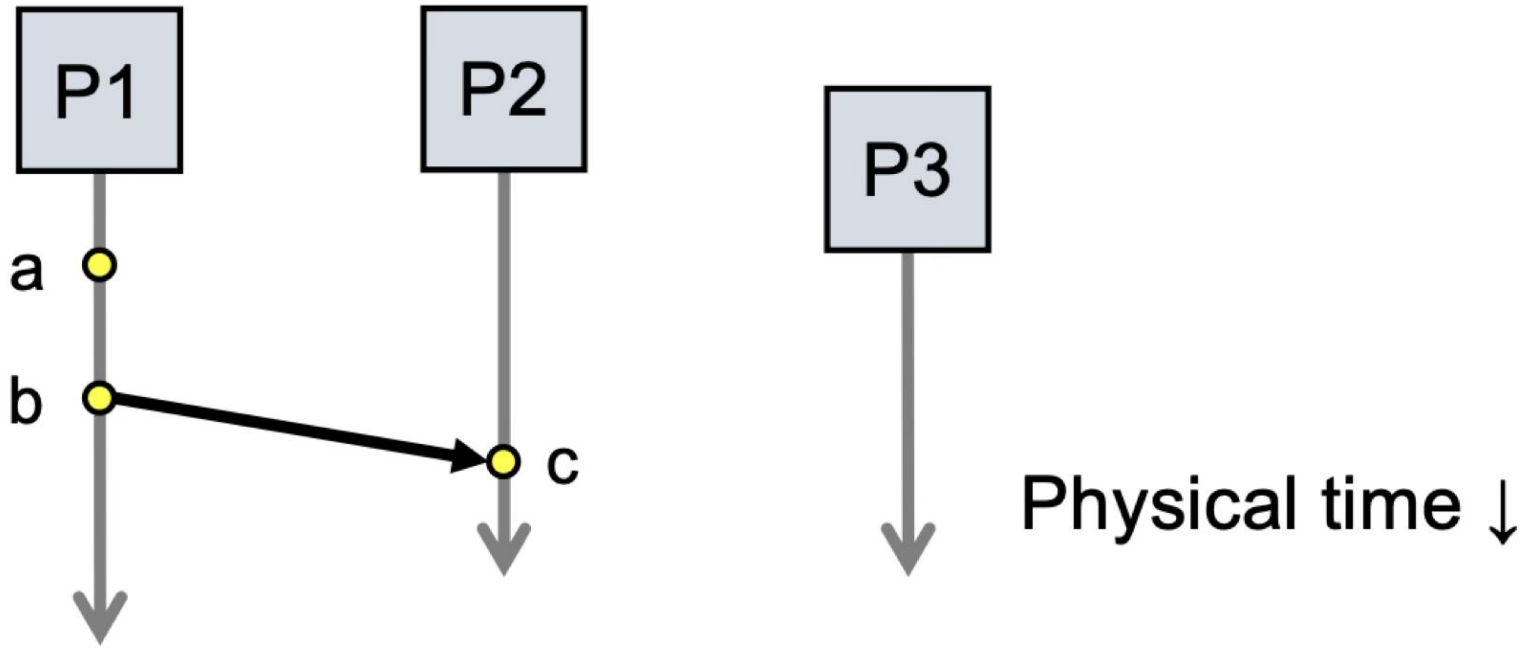
Happens-Before

1. If same process and a occurs before b, then a -> b



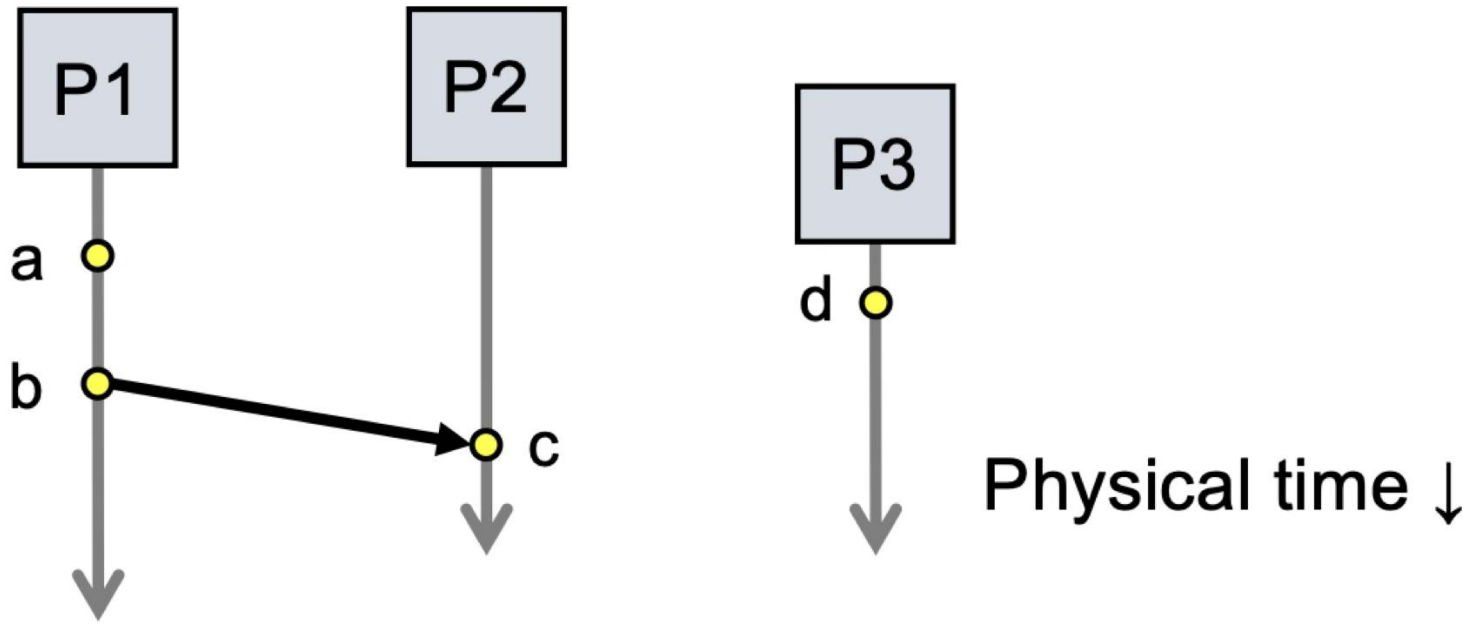
Happens-Before

1. If same process and a occurs before b, then a -> b
2. If c is a message receipt of b, then b -> c
3. If a->b and b->c, then a->c



Happens-Before

- Not all events are related by a
- a, d not related by a so concurrent, written as a || d



Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems

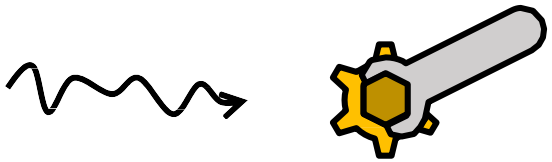
Jonathan Mace, Ryan Roelke, Rodrigo Fonseca

Brown University

Key insight of Pivot Tracing

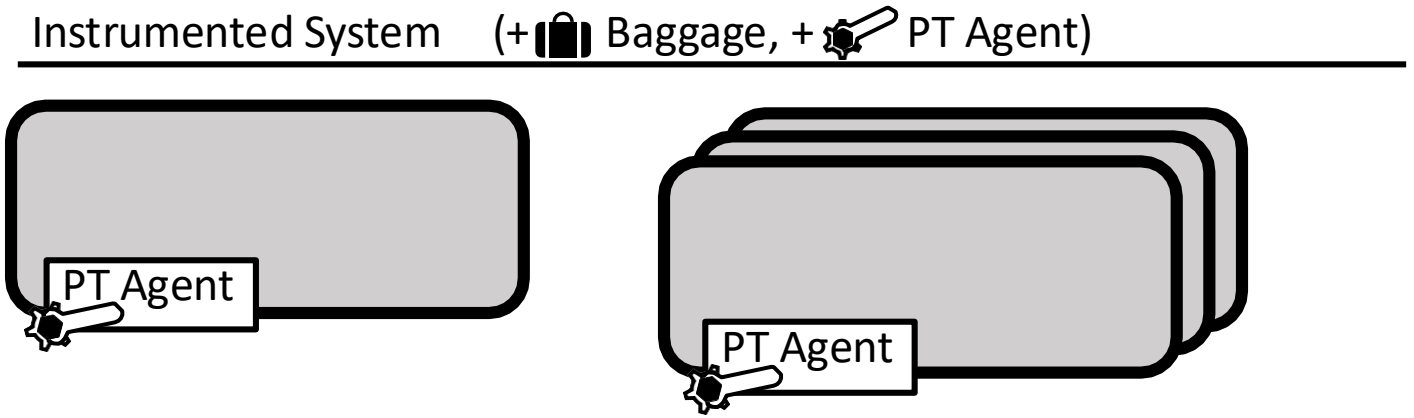
- **Model system events as tuples in a streaming, distributed dataset**
- **Dynamically evaluate relational queries over this dataset**
- **Happened-before Join**

Pivot Tracing Design

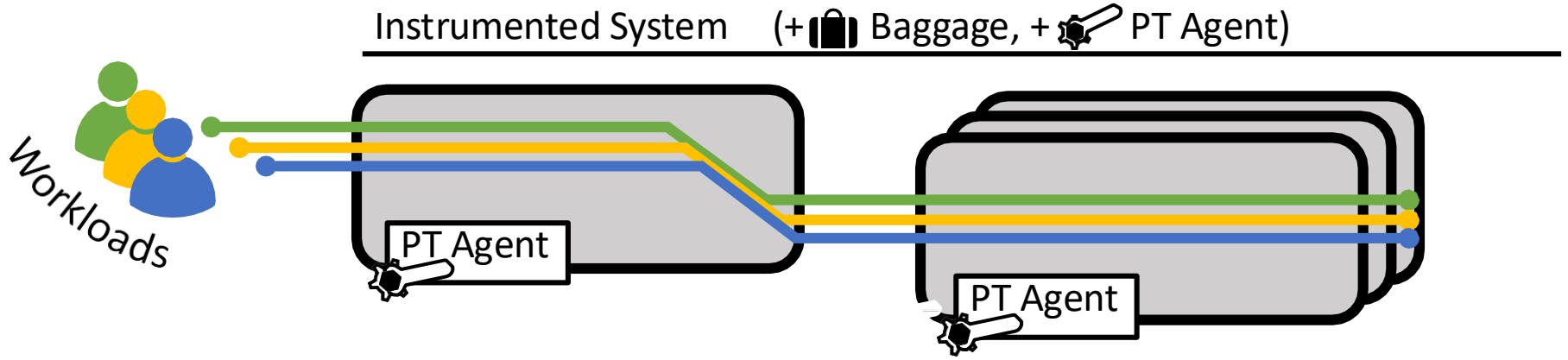
Dynamic instrumentation  PT Agent

Causal tracing  Baggage

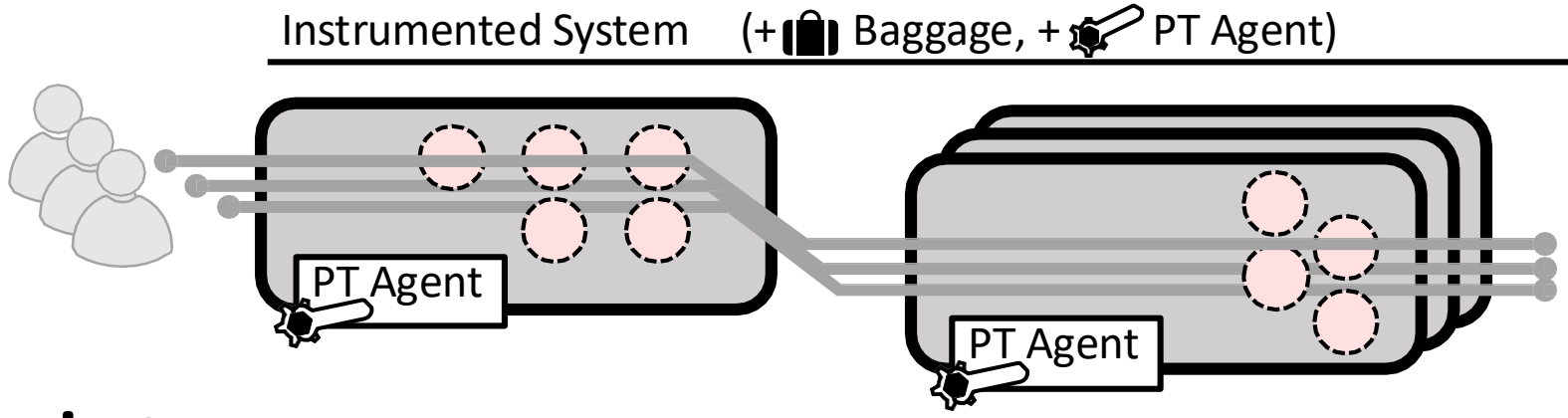
Pivot Tracing Workflow



Pivot Tracing Workflow



Pivot Tracing Workflow



Tracepoints

Places where PT can add instrumentation

Export identifiers accessible to queries

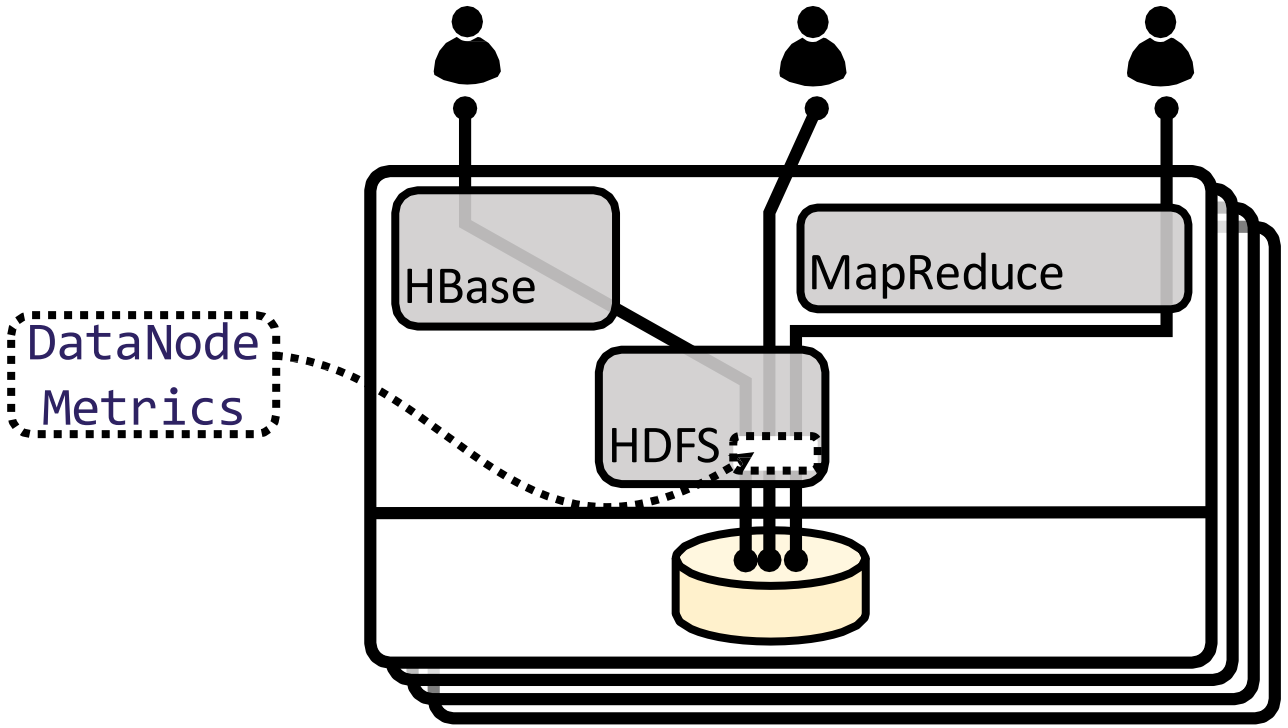
Defaults: host, timestamp, pid, proc name

Only references – not materialized until query is installed

Tracepoint A
Class: A
Method: `A1()`

Tracepoint B
Class: B
Method: `B1()`
Exports: "delta"=`delta`

Tracing Point in Pivot Tracing



Tracing Point in Pivot Tracing

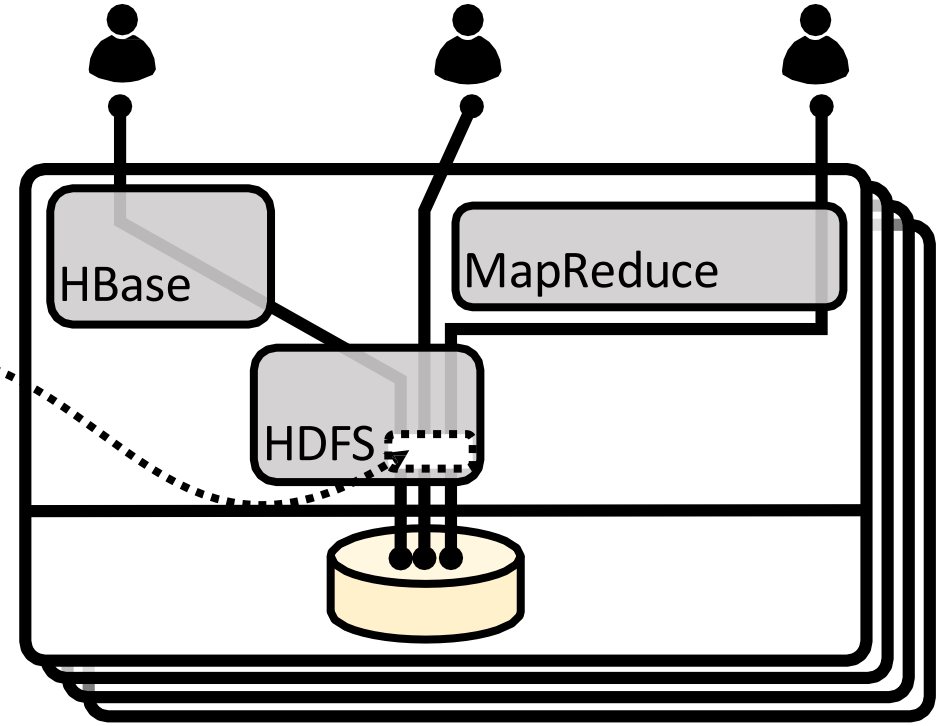
DataNodeMetrics.java

```
50  public class DataNodeMetrics {
      ...
266   public void incrBytesRead(int delta)
      {
267     ...
268   }
      ...
407 }
```

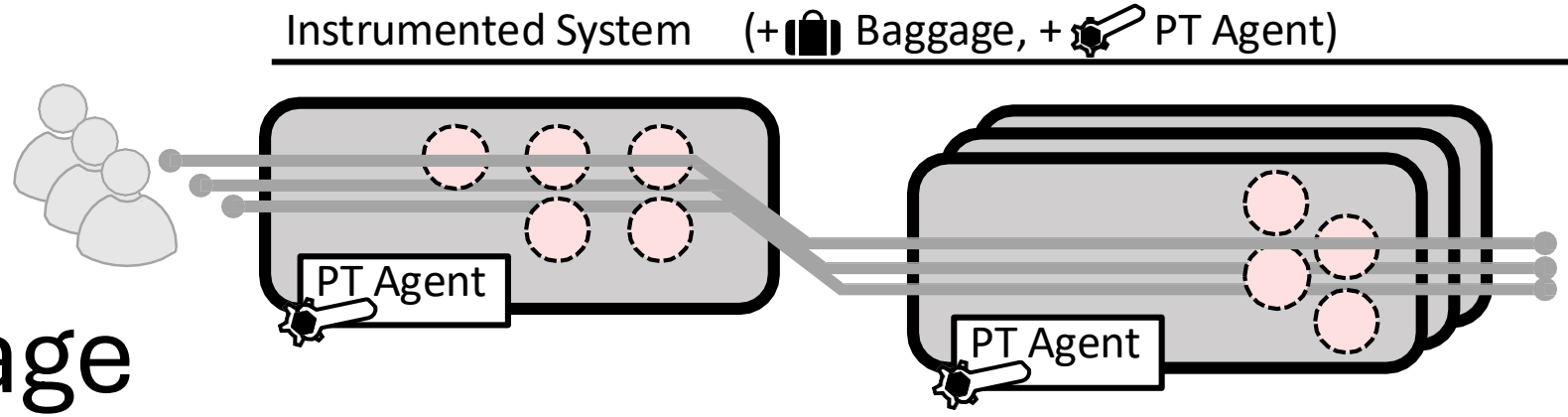
Tracepoint
Method: incrBytesRead
Class: DataNodeMetrics
Exports: "delta"=delta

("DataNodeMetrics", delta=10, host="hop01", …)

DataNode Metrics



Pivot Tracing Workflow



Query Language

Relational query language, similar to SQL, LINQ

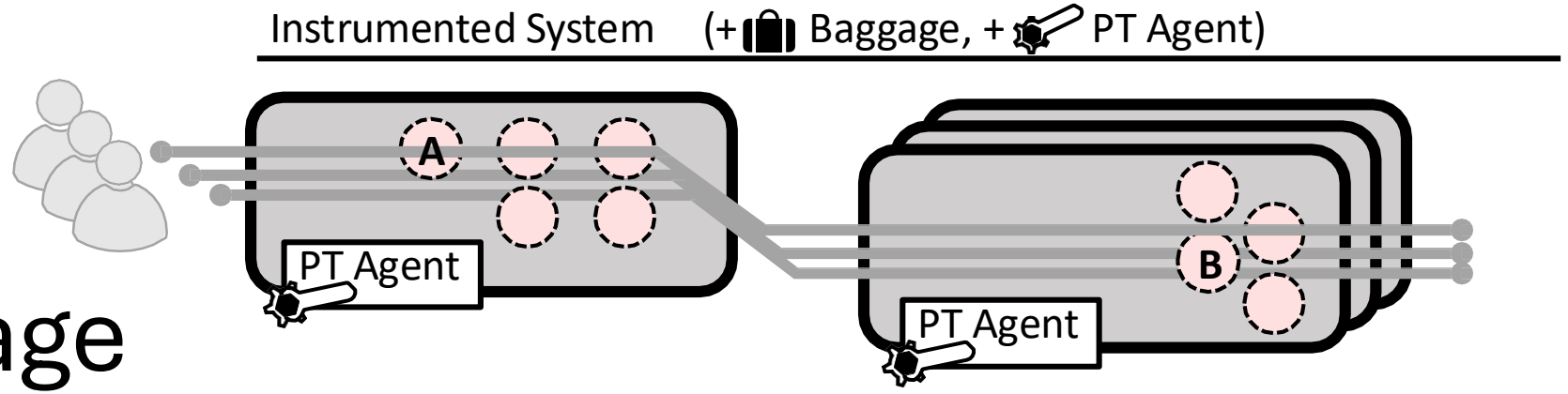
- Selection
- Projection
- Filter
- GroupBy
- Aggregation
- Happened-Before Join

Refers to trace point-exported identifiers

Tracepoint A
Class: A
Method: A1()

Tracepoint B
Class: B
Method: B1()
Exports: "delta"=delta

Pivot Tracing Workflow



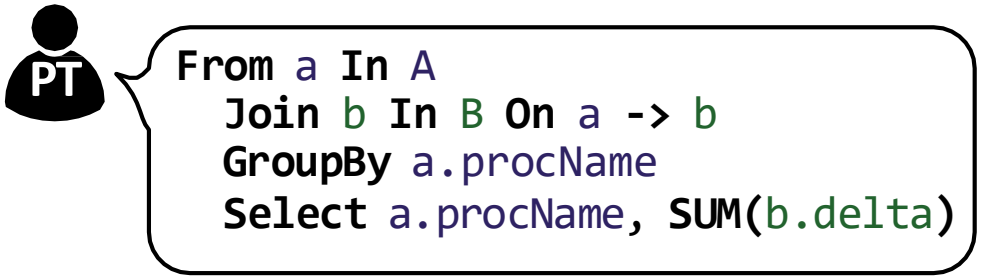
Query Language

Relational query language, similar to SQL, LINQ

- Selection
- Projection
- Filter
- GroupBy
- Aggregation
- Happened-Before Join

Refers to trace point-exported identifiers

Output: stream of tuples



Tracepoint A
Class: A
Method: A1()

Tracepoint B
Class: B
Method: B1()
Exports: "delta"=delta

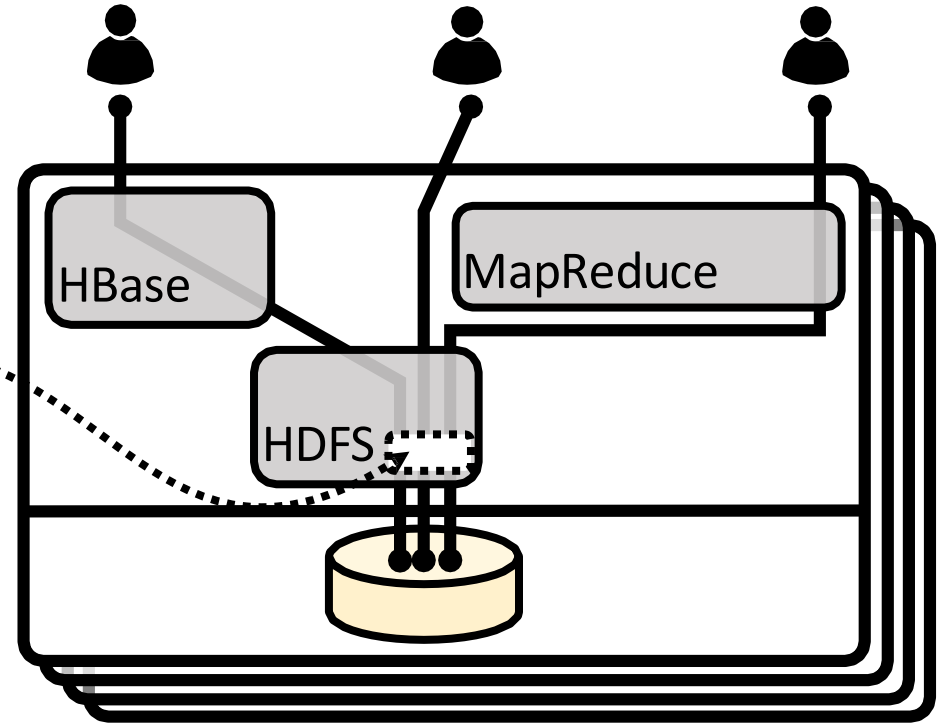
Query in Pivot Tracing

DataNodeMetrics.java

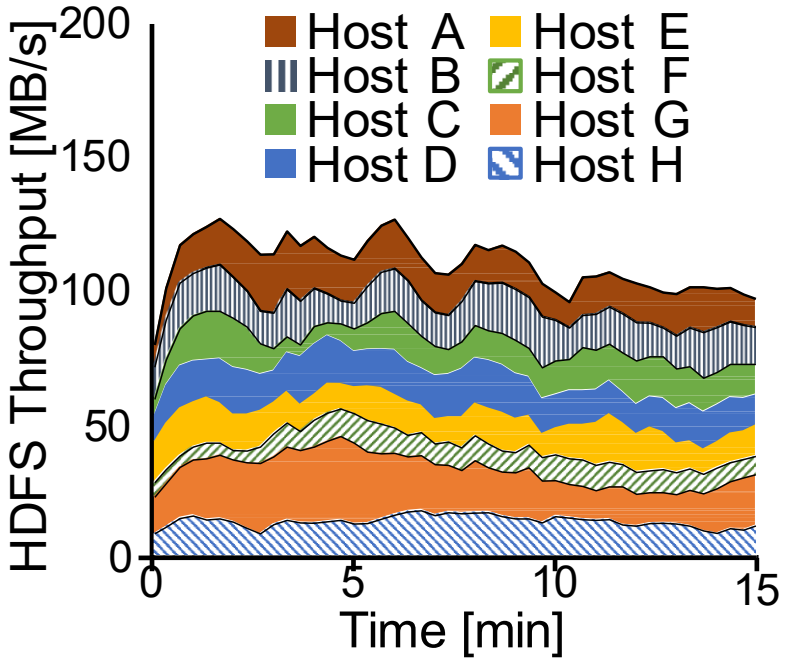
```
50  public class DataNodeMetrics {
      ...
266   public void incrBytesRead(int delta)
      {
267     ...
268   }
      ...
407 }
```

Tracepoint
Method: incrBytesRead
Class: DataNodeMetrics
Exports: "delta"=delta

DataNode Metrics

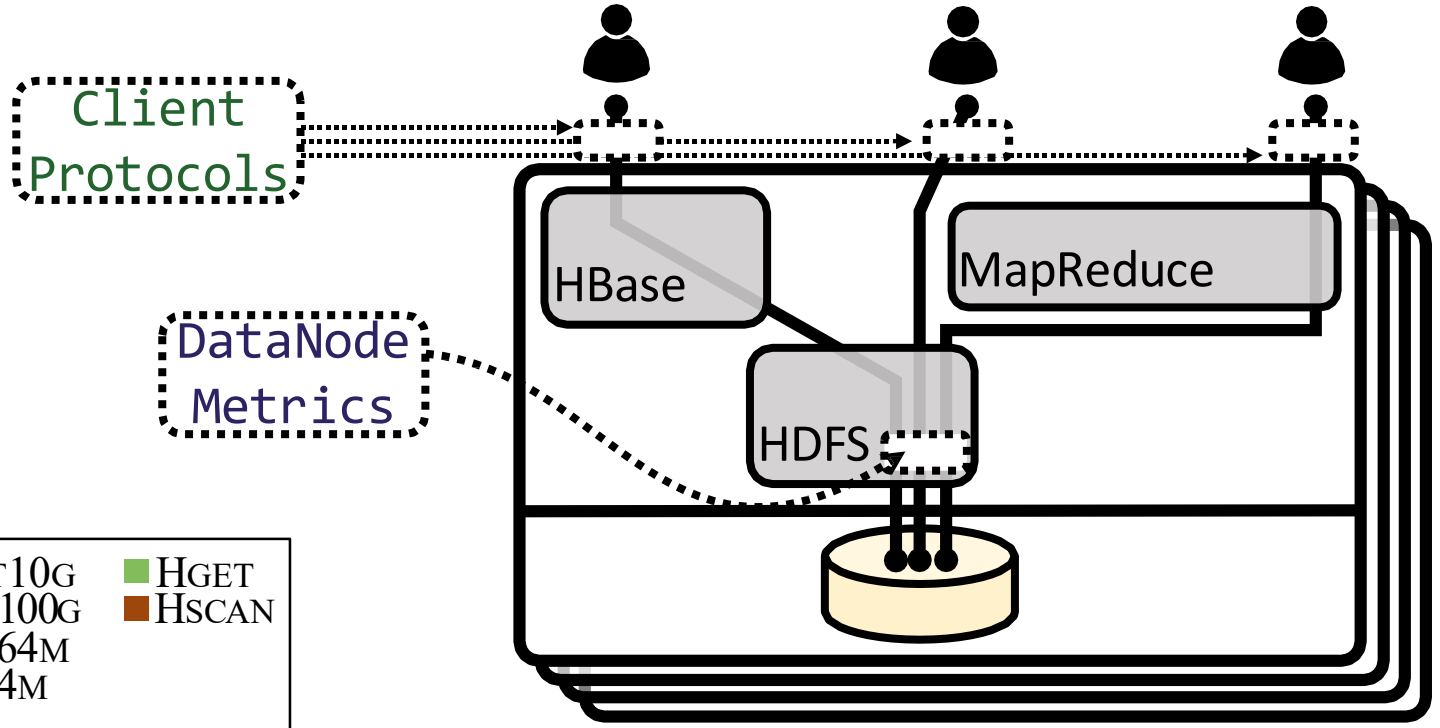


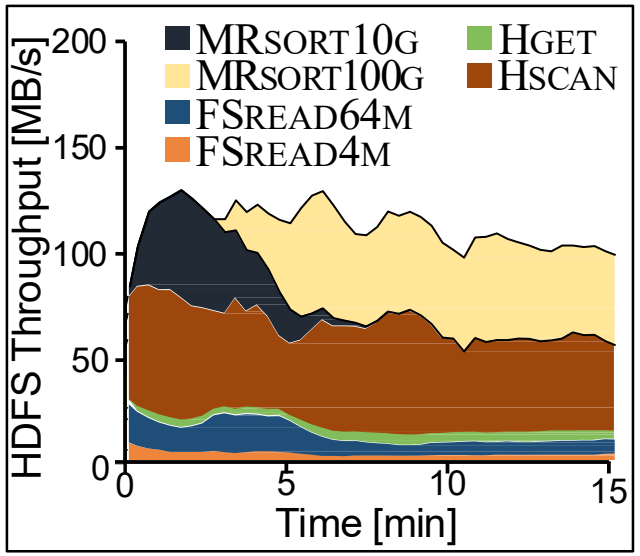
("
hc
),




```
From incr In
DataNodeMetrics.incrBytesRead
 GroupBy incr.host
 Select incr.host, SUM(incr.delta)
```

Join Query


Client Protocols
DataNode Metrics
HBase
MapReduce
HDFS

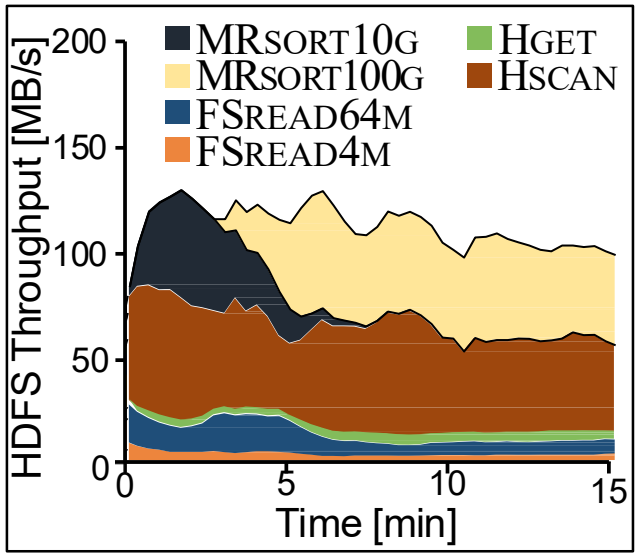

MRSORT10G
MRSORT100G
FSREAD64M
FSREAD4M
HGET
HSCAN
HDFS Throughput [MB/s]
200
150
100
50
0
0
5
10
15
Time [min]

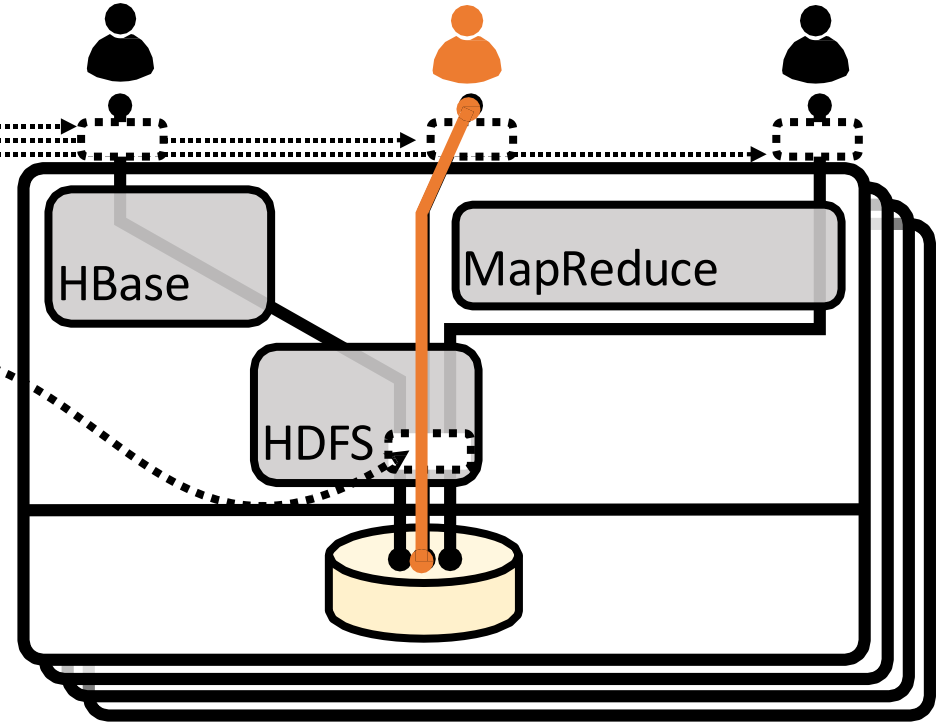
Join Query

Client Protocols

("ClientProtocols", procName="HGET", …)

DataNode Metrics



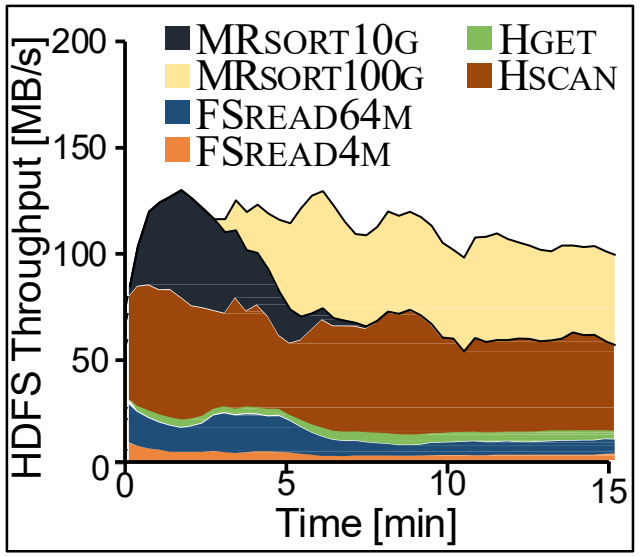


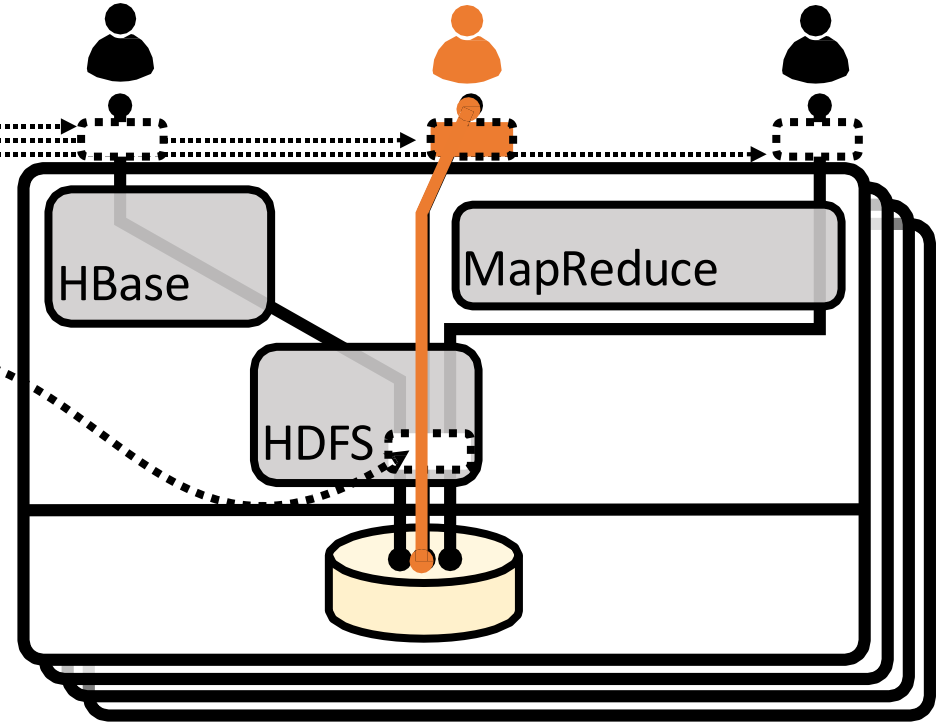
Join Query

Client Protocols

("ClientProtocols", procName="HGET", …)

DataNode Metrics



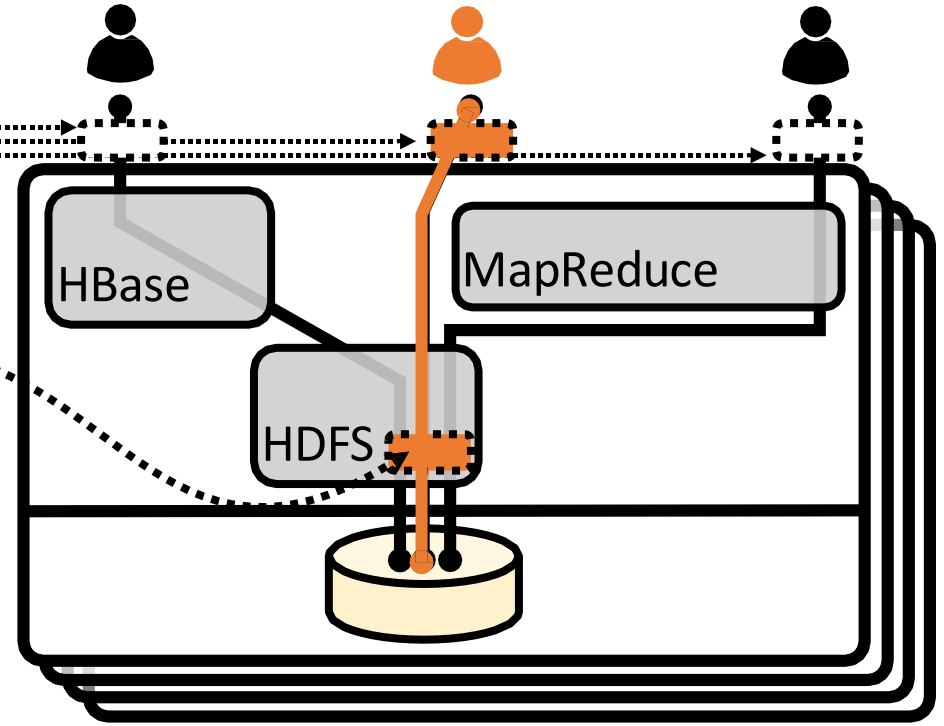


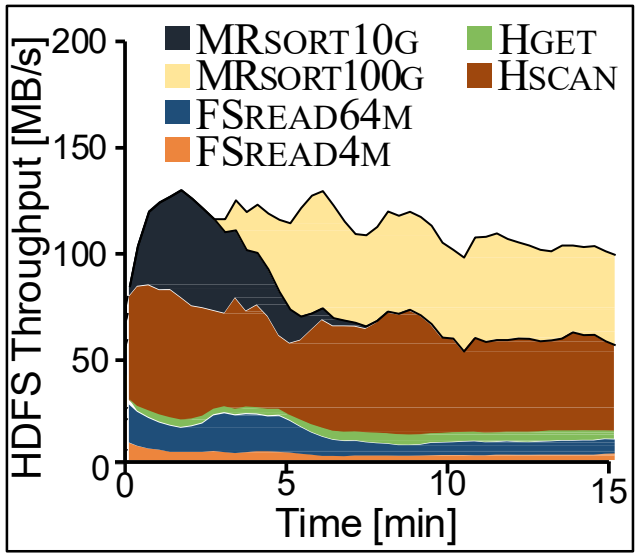
Join Query

Client Protocols

("ClientProtocols", procName="HGET", …)

DataNode Metrics





Client Protocols ⋈ DataNode Metrics

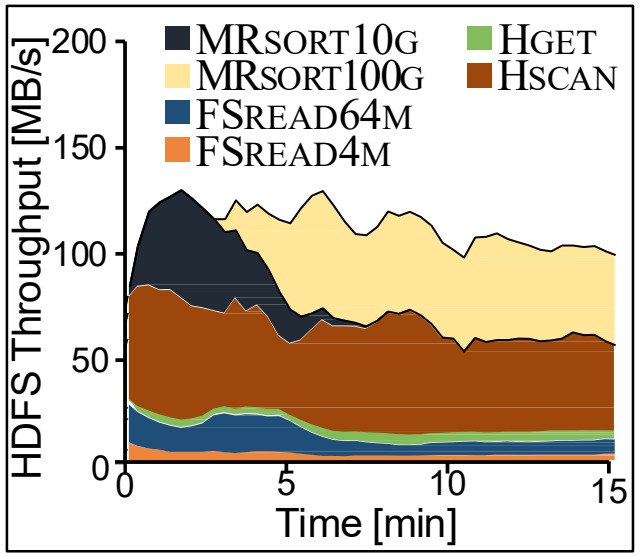
("ClientProtocols", procName="HGET", …
"DataNodeMetrics", delta=10, host="Hop01", …)

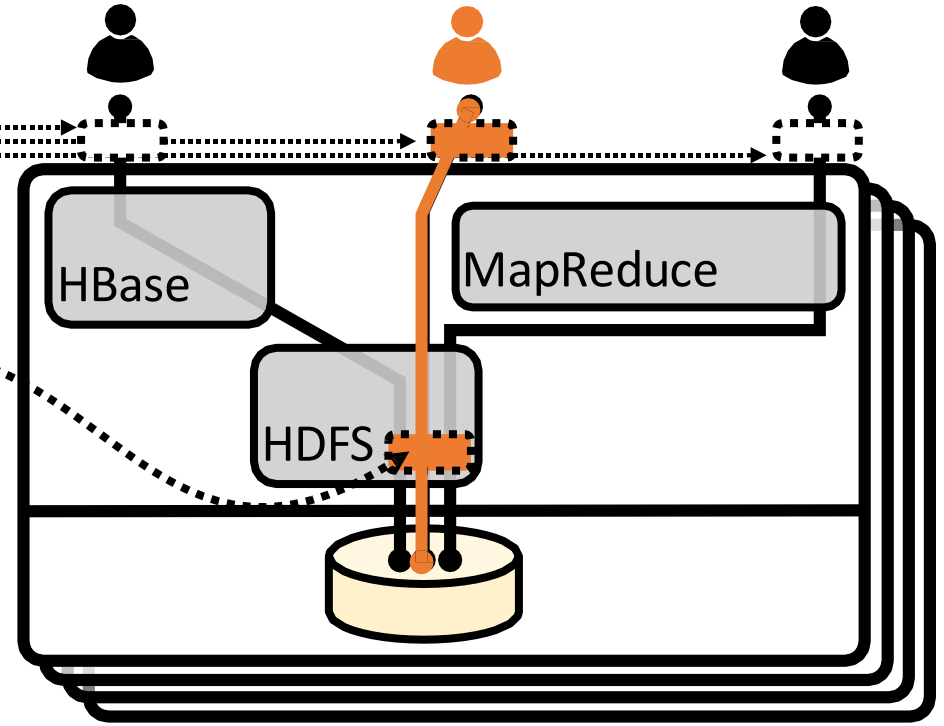
Join Query

Client Protocols

("ClientProtocols", procName="HGET", …)

DataNode Metrics

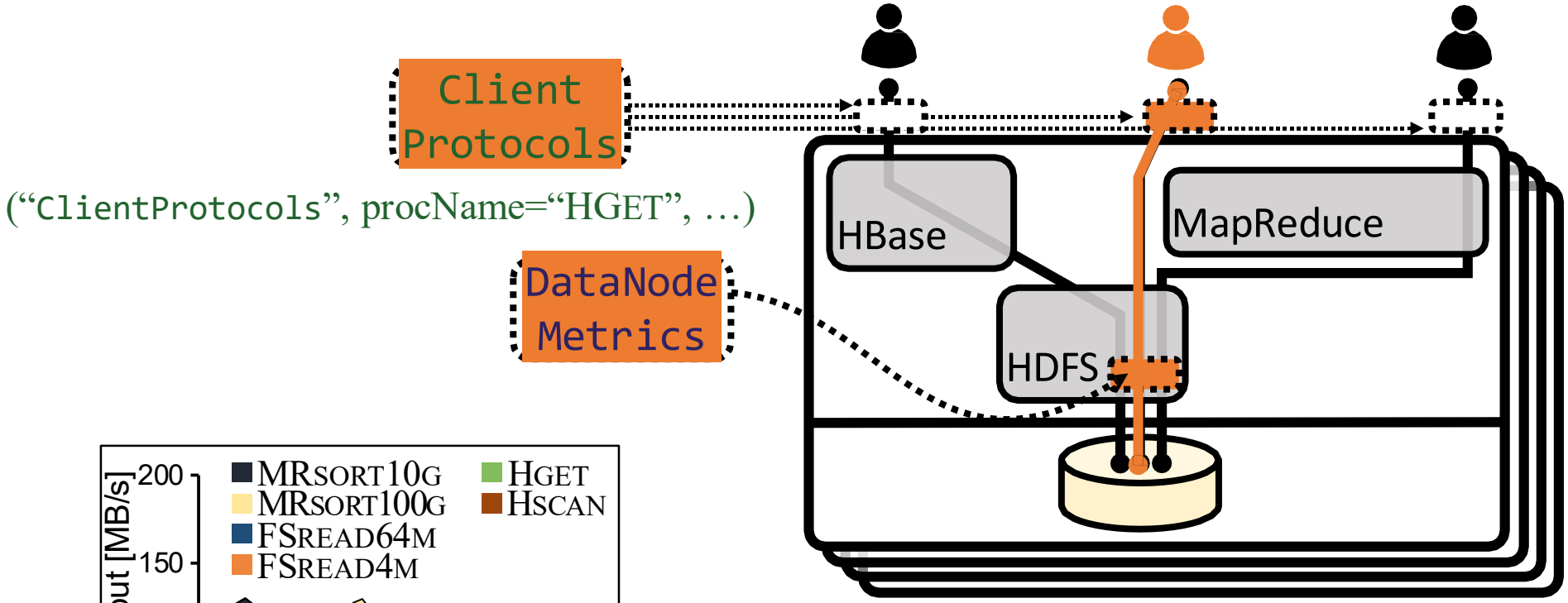


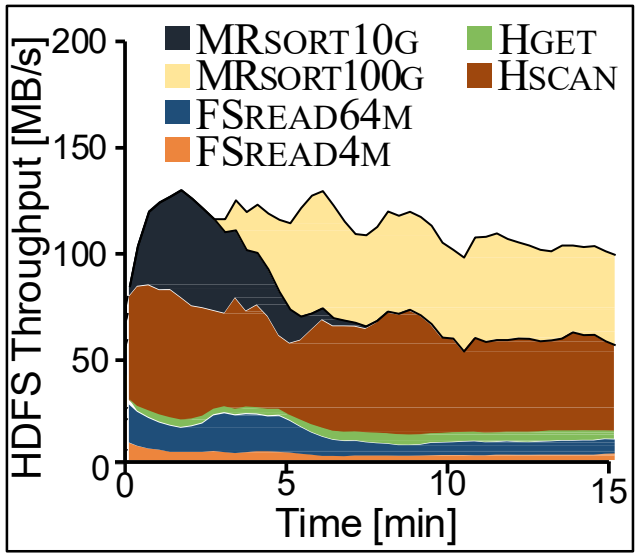


Client Protocols ⋈ DataNode Metrics

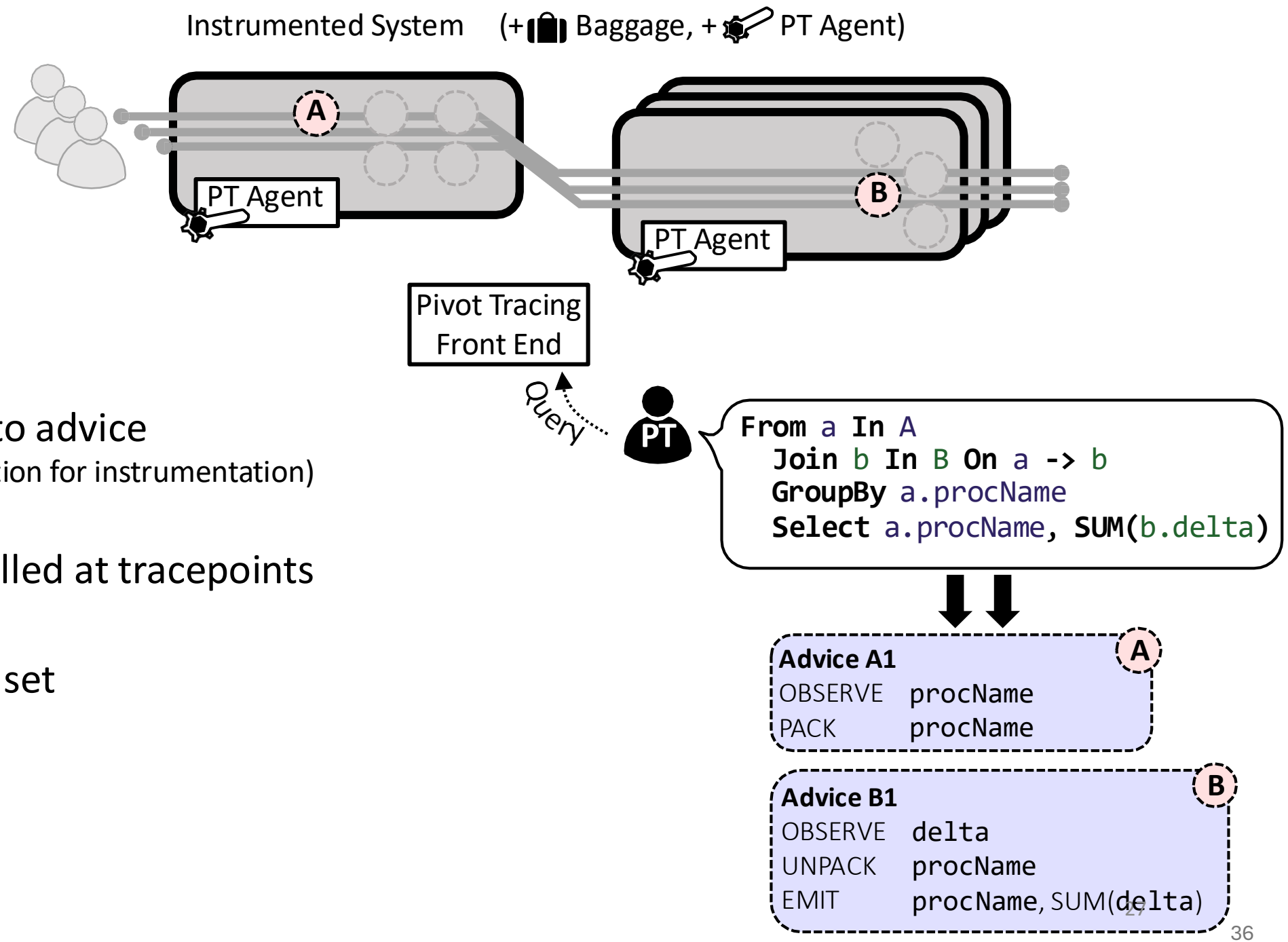
("ClientProtocols", procName="HGET", …
"DataNodeMetrics", delta=10, host="Hop01", …)

Join Query






```
From incr In
DataNodeMetrics.incrBytesRead
 Join client In First(ClientProtocols) On
 client -> incr
 GroupBy client.procName
 Select client.procName, SUM(incr.delta)
```



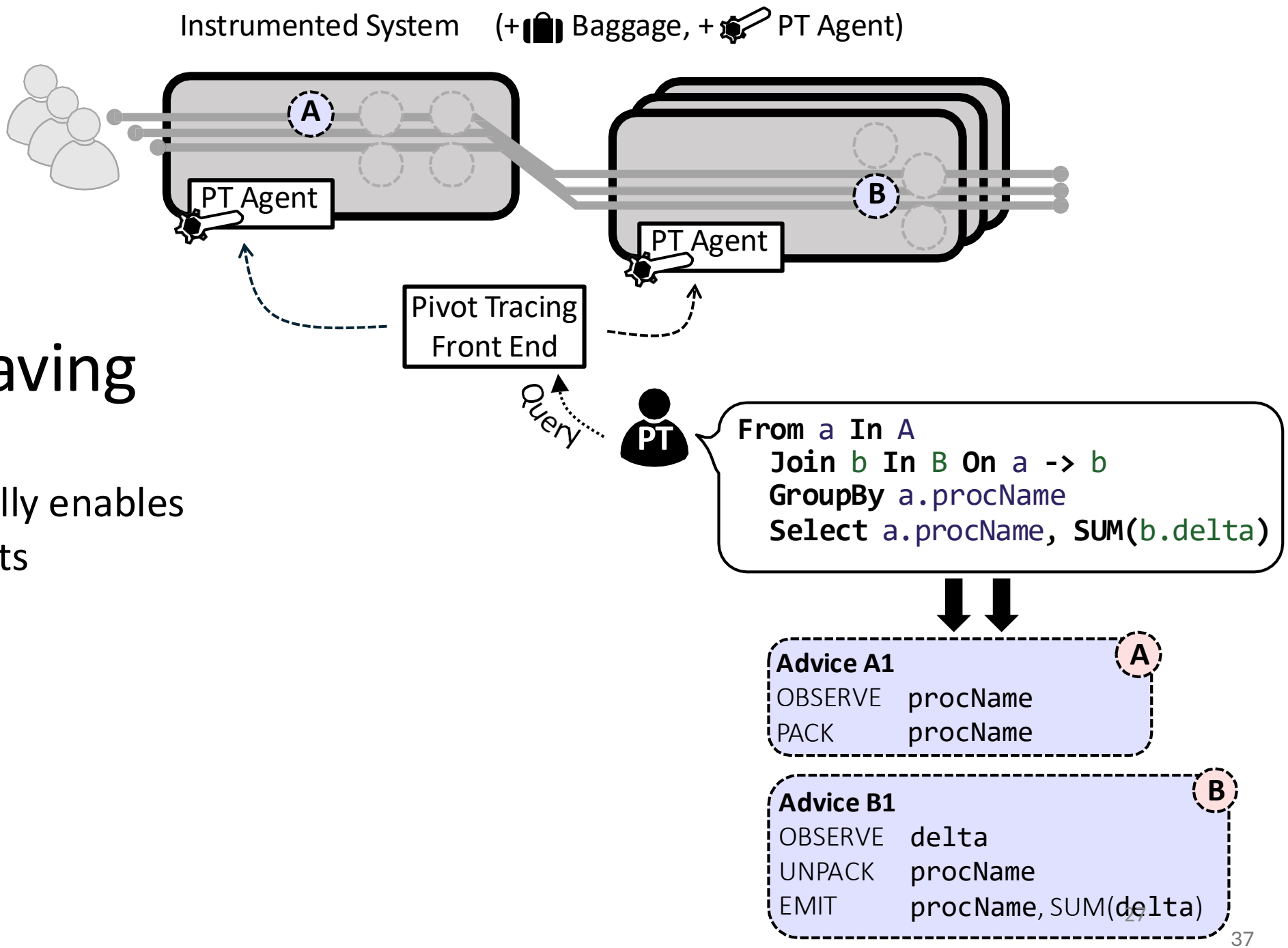
Advice

Query is compiled to advice
(intermediate representation for instrumentation)

Advice will be installed at tracepoints

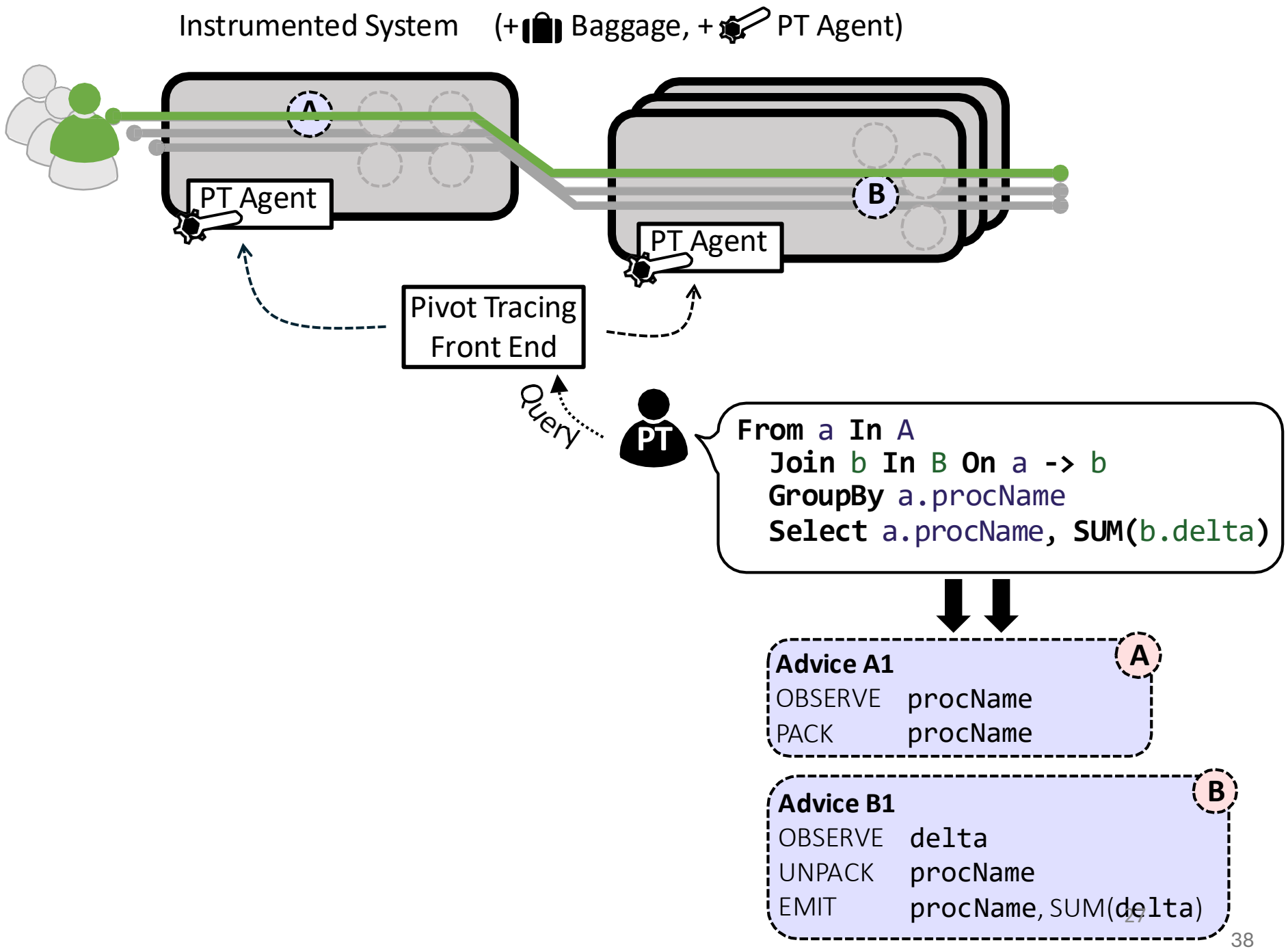
Limited instruction set

- OBSERVE
- PACK
- FILTER
- UNPACK
- EMIT

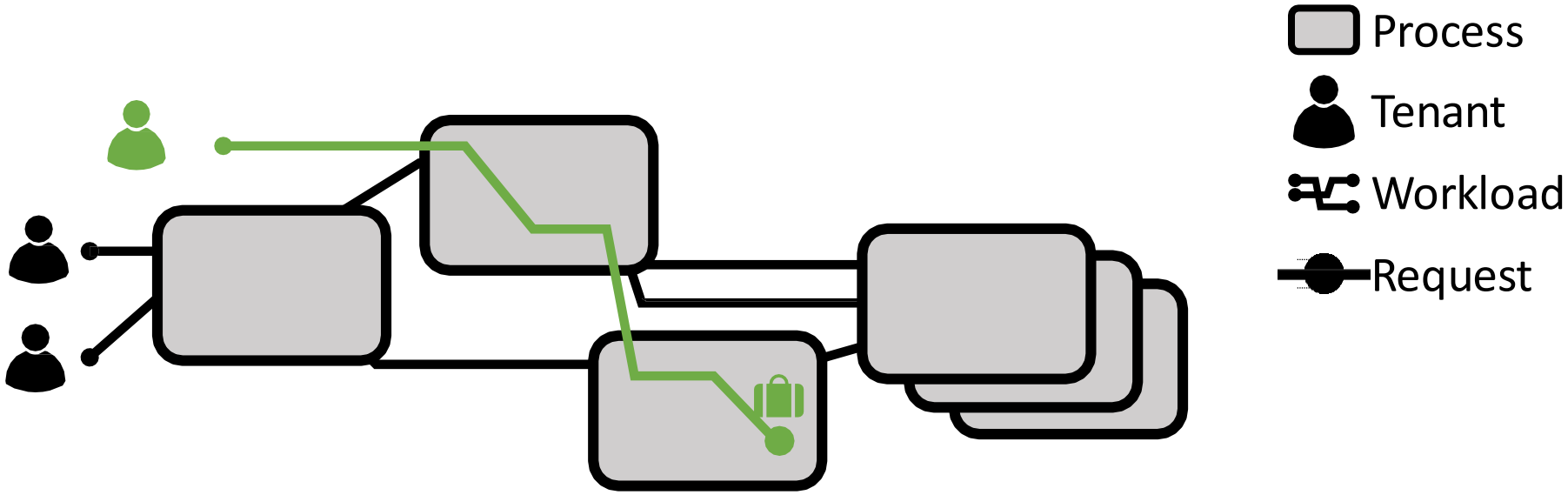


Weaving

PT Agent dynamically enables advice at tracepoints

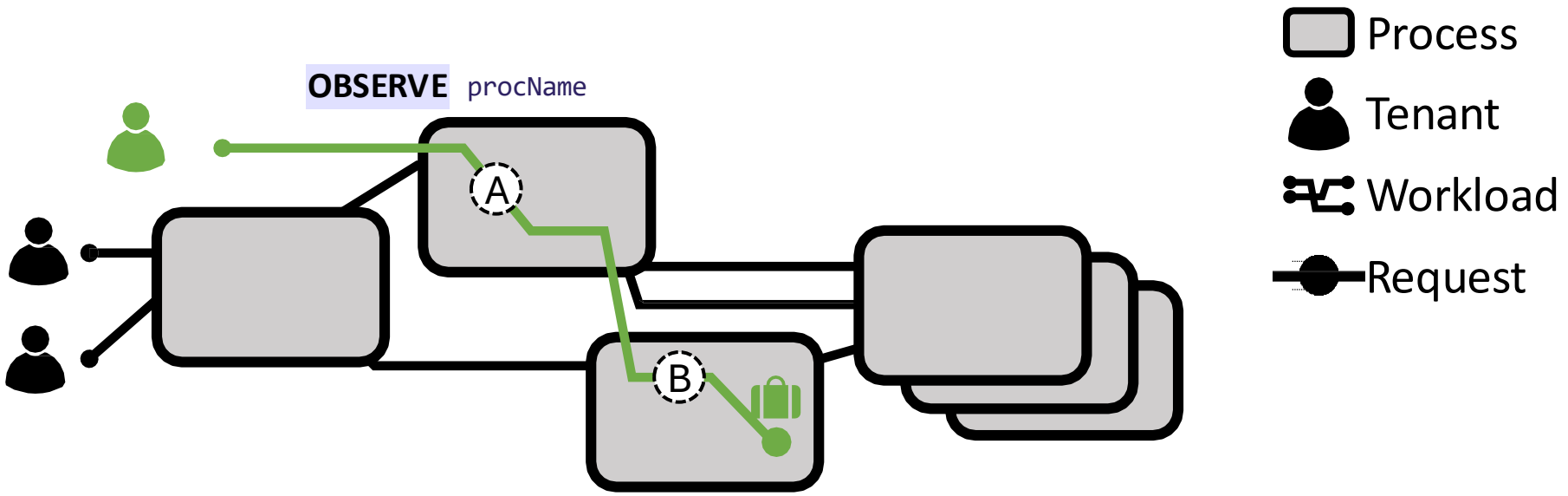

Instrumented System (+ Baggage, + PT Agent)
A
PT Agent
B
PT Agent
Pivot Tracing Front End
Query
PT
From a In A
Join b In B On a -> b
GroupBy a.procName
Select a.procName, SUM(b.delta)
Advice A1
OBSERVE procName
PACK procName
A
Advice B1
OBSERVE delta
UNPACK procName
EMIT procName, SUM(delta)
B

Baggage



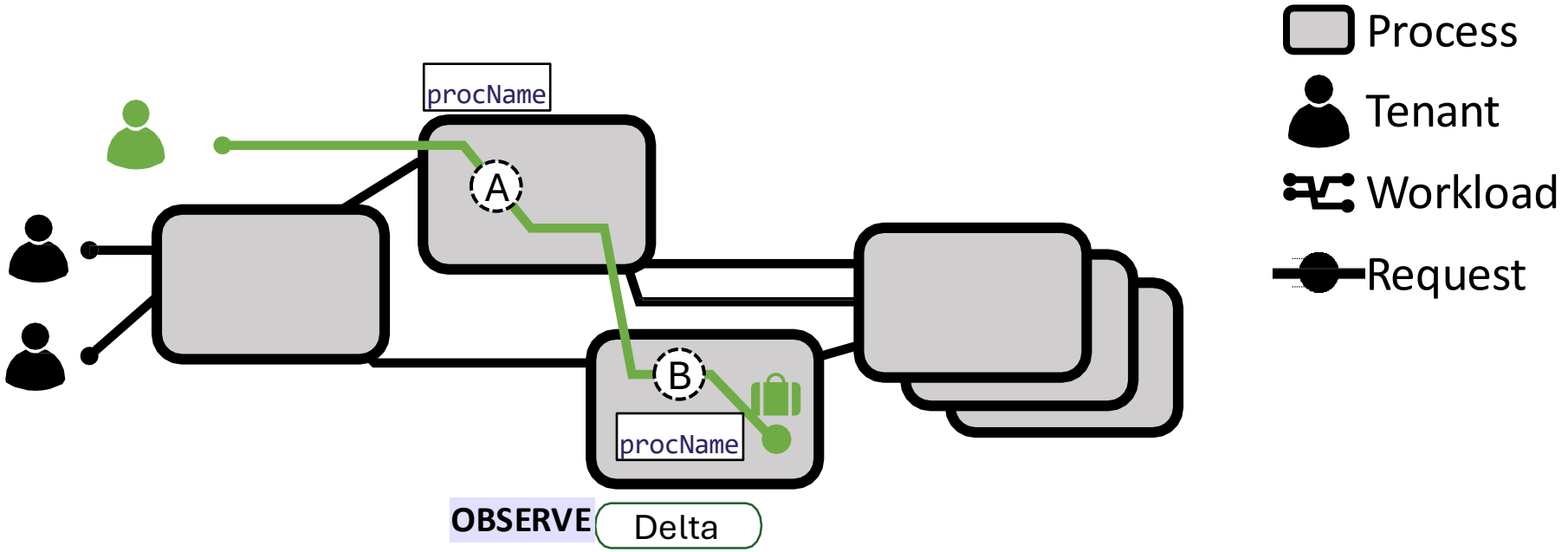
- Baggage is a Key: Value container propagated alongside a request
  - Generalization of metadata in end-to-end tracing
  - One instance per request

Baggage



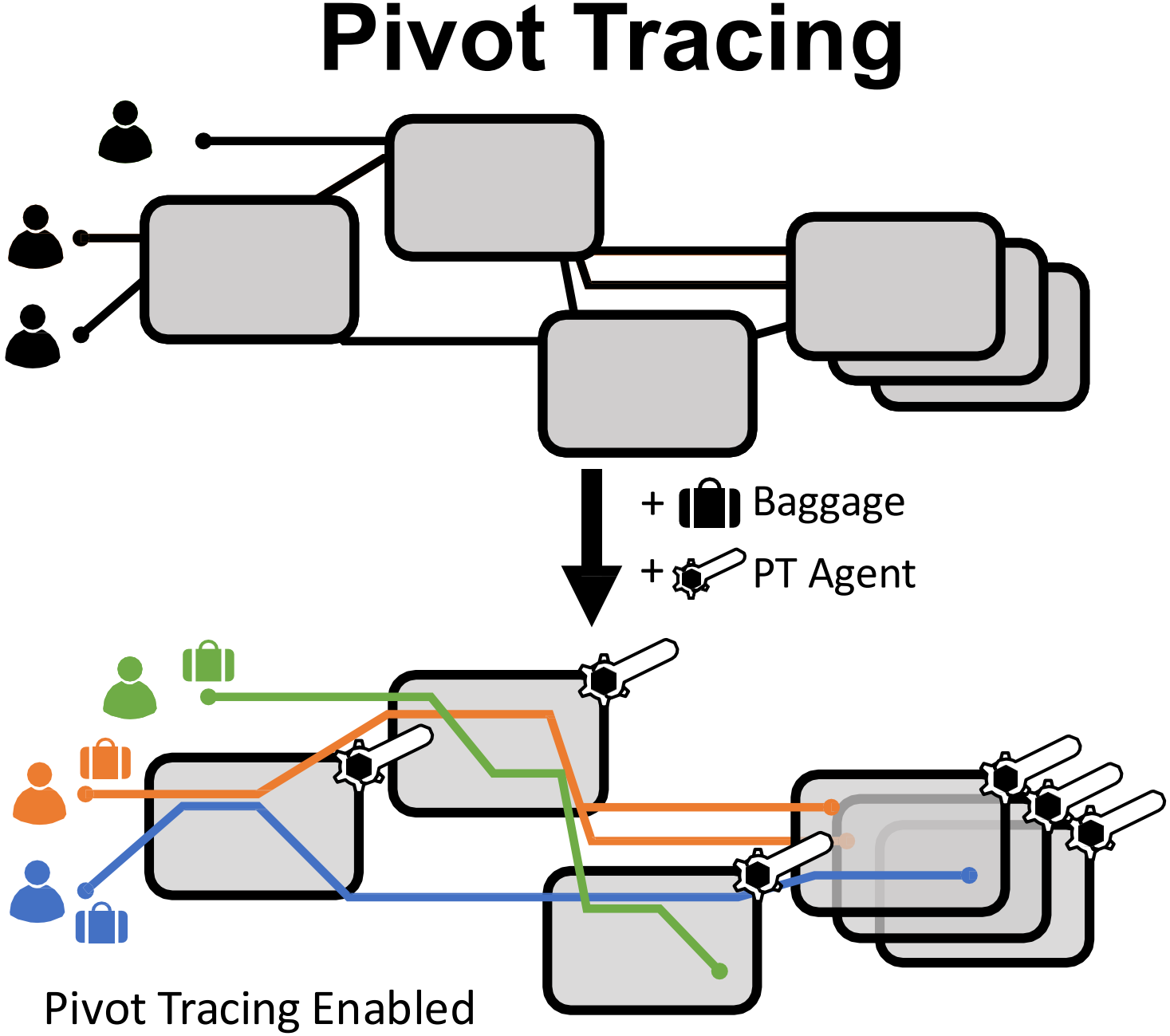
- Baggage is a Key: Value container propagated alongside a request
  - Generalization of metadata in end-to-end tracing
  - One instance per request

Baggage



- Baggage is a Key: Value container propagated alongside a request
  - Generalization of metadata in end-to-end tracing
  - One instance per request

Pivot Tracing

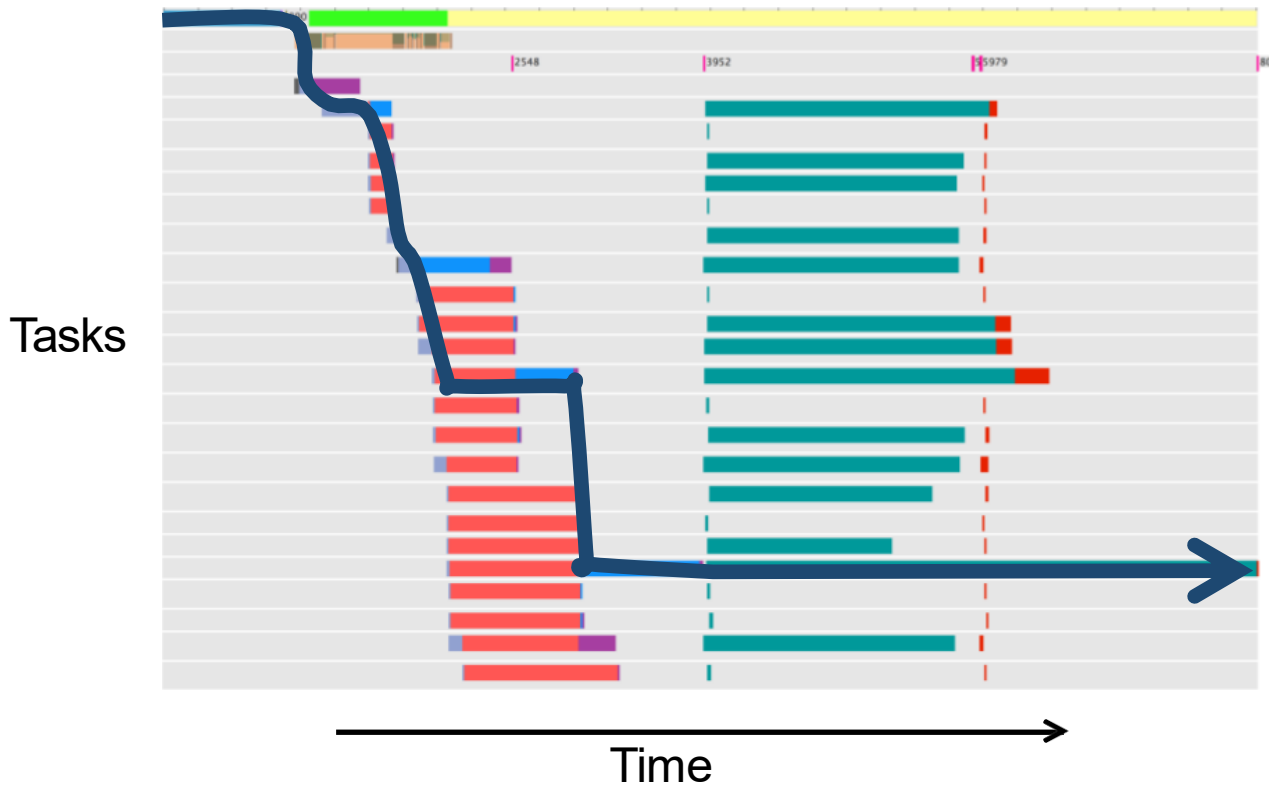


The Mystery Machine: End-to-end performance analysis of large-scale Internet services

Michael Chow
David Meisner, Jason Flinn, Daniel Peek,
Thomas F. Wenisch

OSDI'14

Internet Services Are Complex



Scale and heterogeneity make Internet services complex

Challenges

Previous methods derive a causal model

- Instrument scheduler and communication
- Build model through human knowledge

Need method that works at scale with heterogeneous components

Opportunities

Component-level logging is ubiquitous

- Tremendous detail about a request's execution

Handle a large number of requests

- Coverage of a large range of behaviors

The Mystery Machine

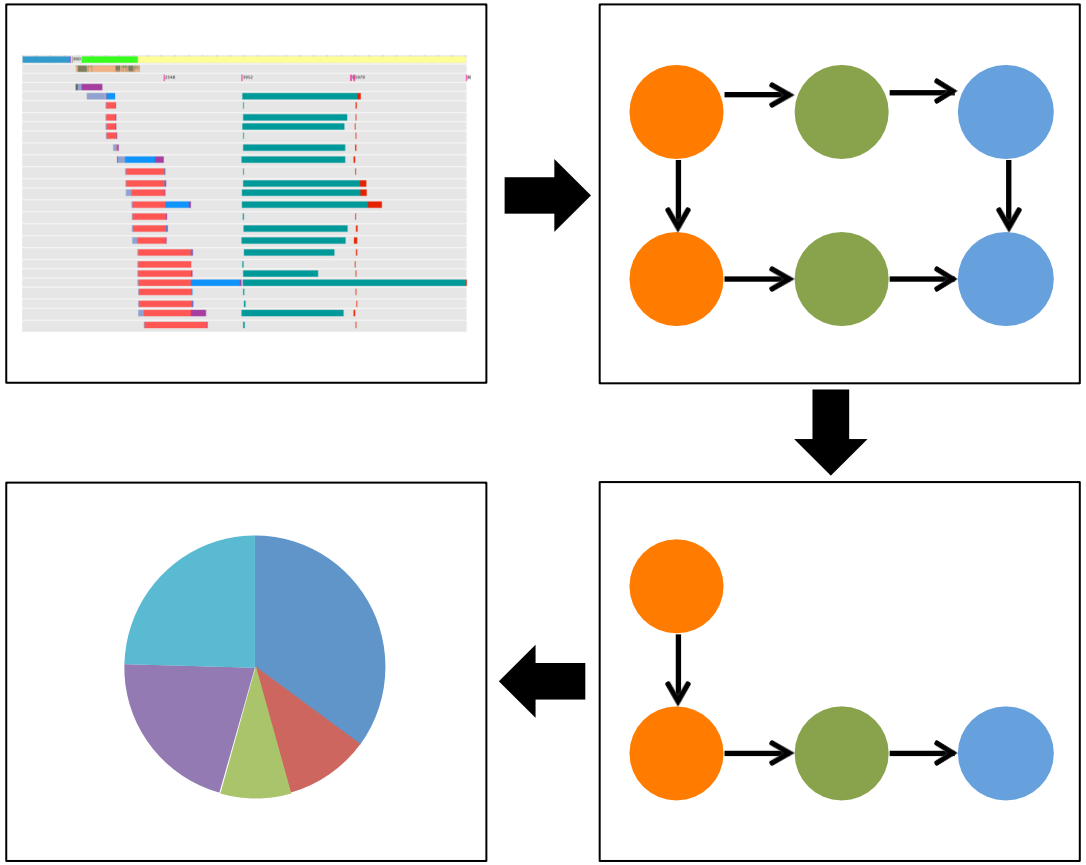
1) Infer causal model from large corpus of traces

- Identify segments
- Hypothesize all possible causal relationships
- Reject hypotheses with observed counterexamples

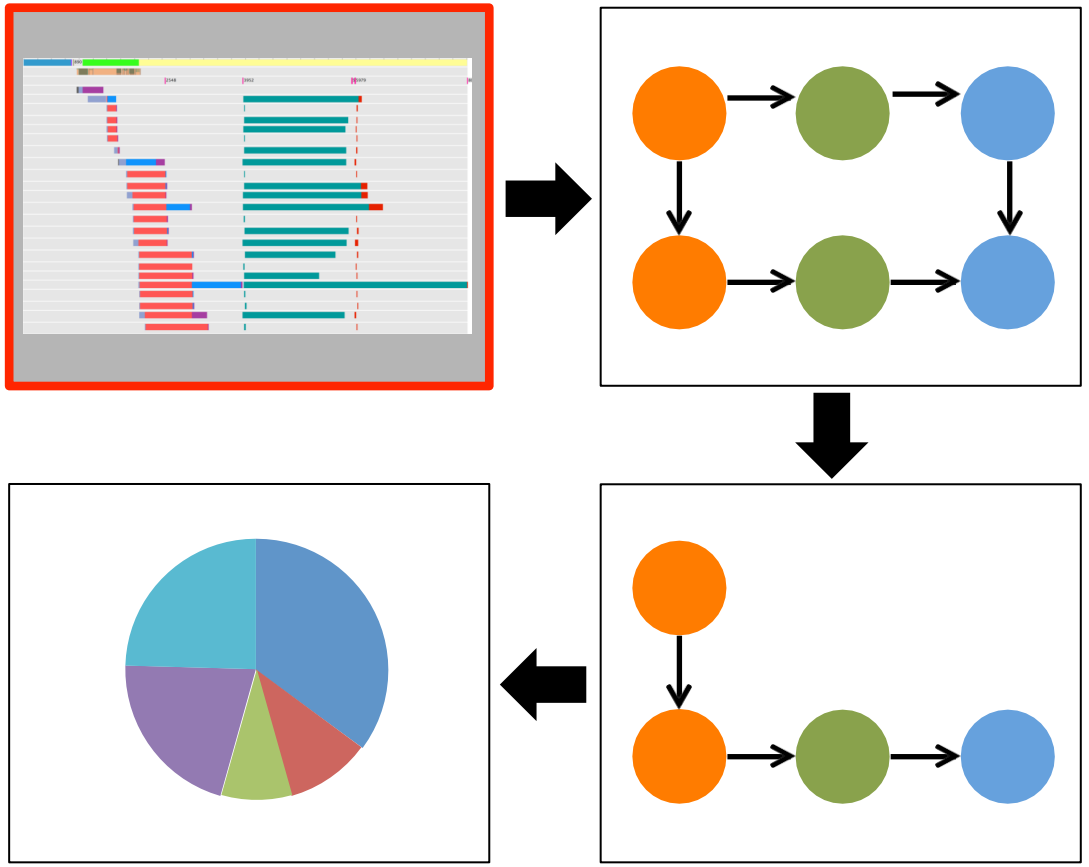
2) Analysis

- Critical path, slack, anomaly detection, what-if

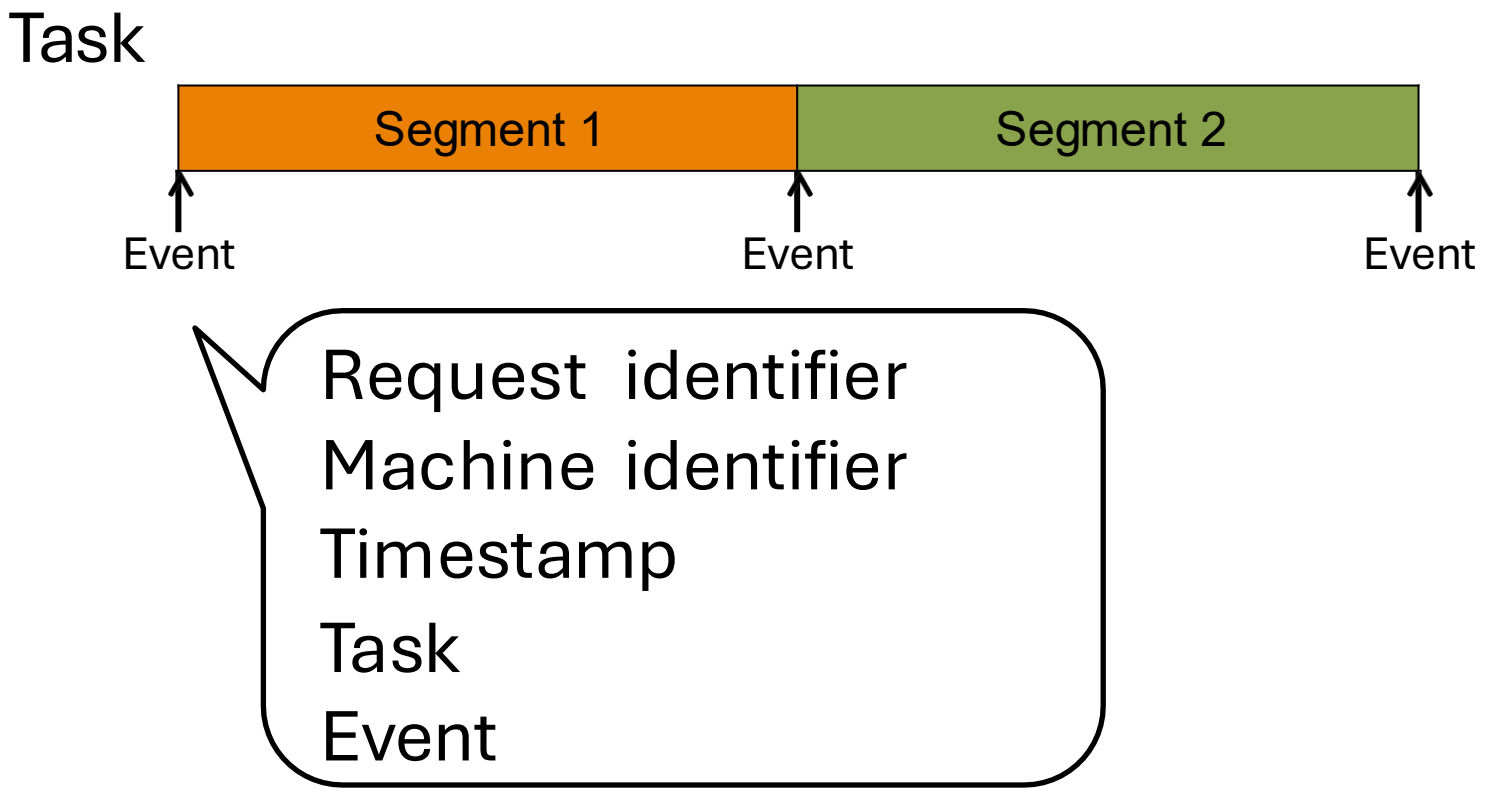
Analysis Pipeline



Step 1: Identify Segments

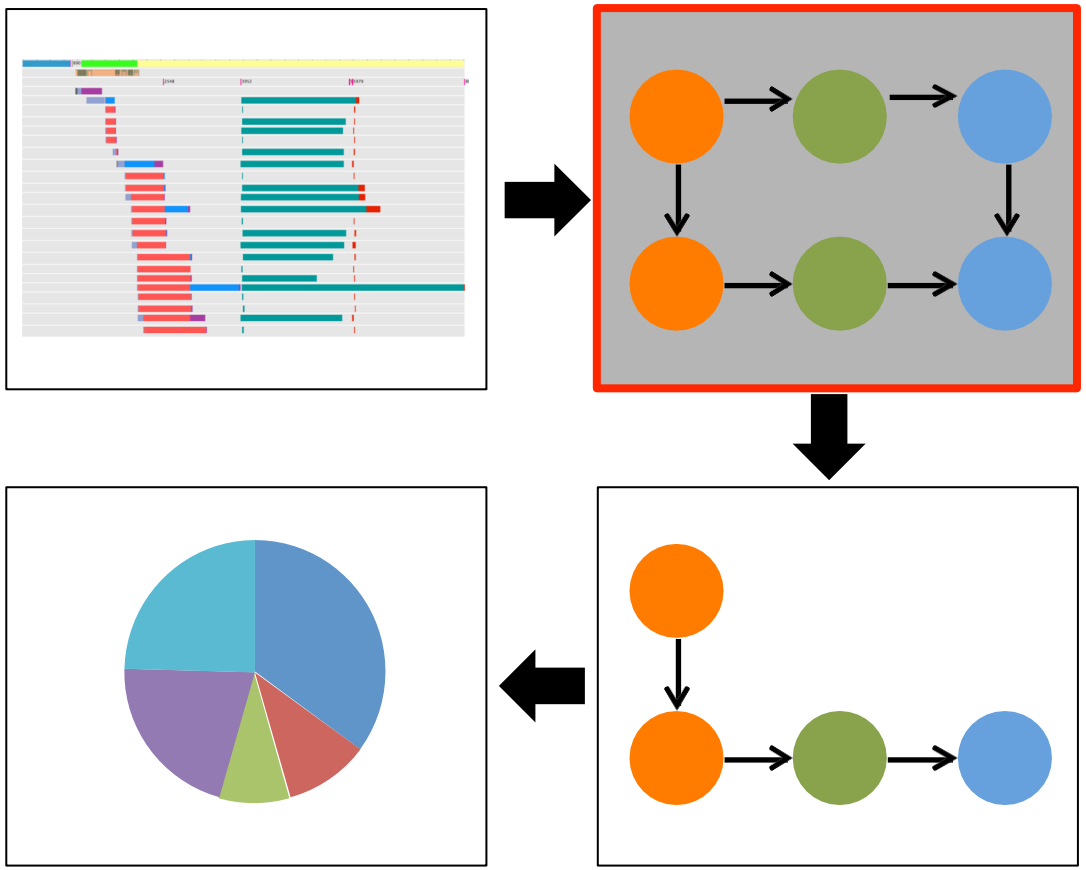


Define a Minimal Schema

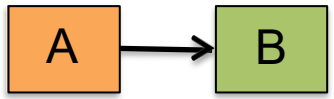
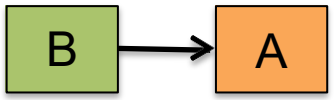
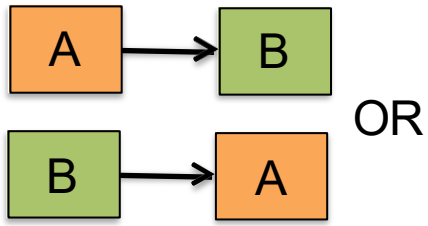
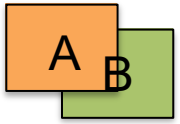
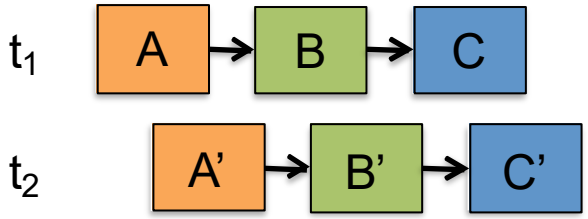
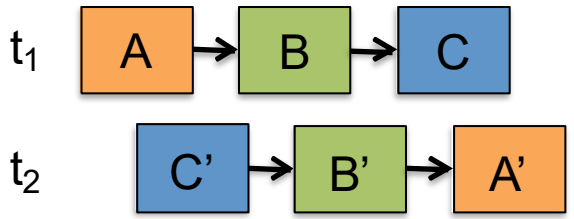


Aggregate existing logs using minimal schema

Step 2: Infer Causal Model

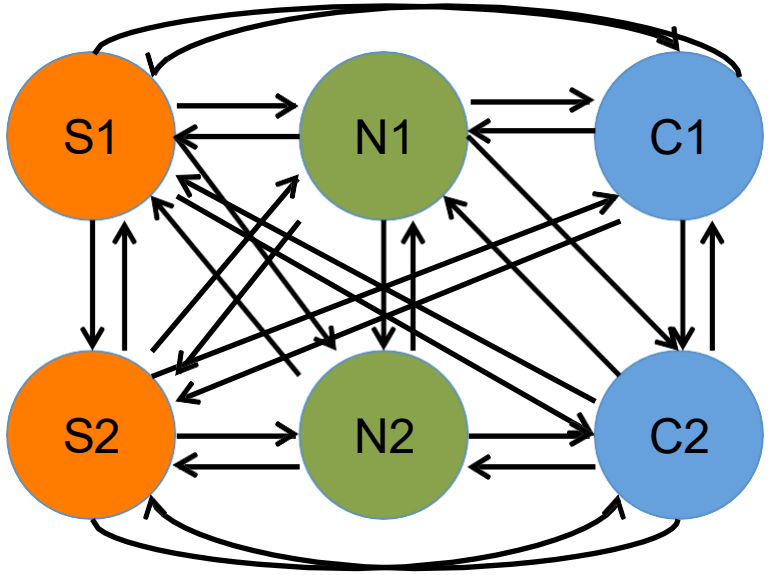


Types of Causal Relationships

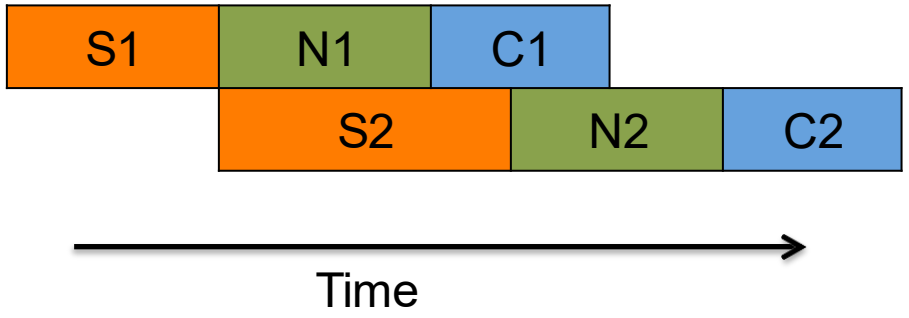
Relationship	Counterexample
Happens-Before 	
Mutual Exclusion 	
Pipeline 	

Producing Causal Model

Causal Model

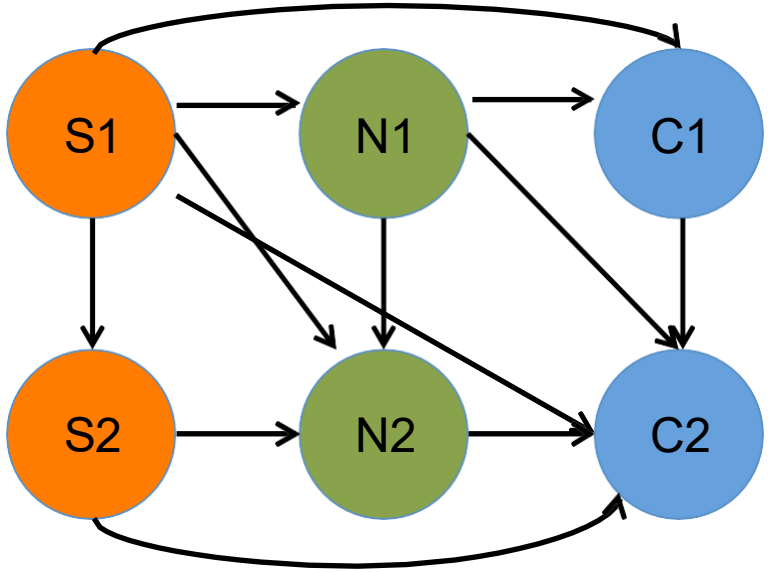


Trace 1

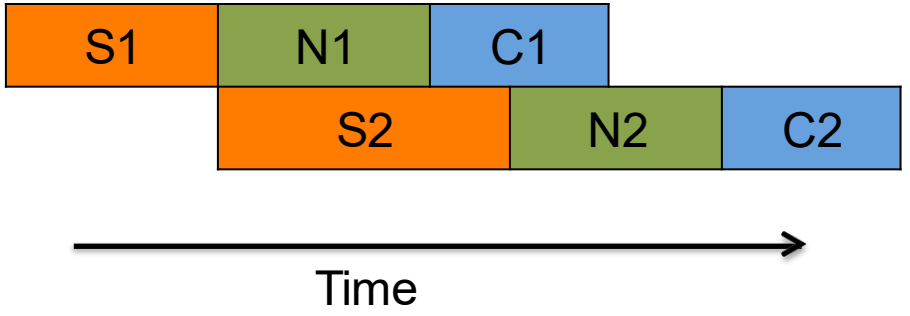


Producing Causal Model

Causal Model

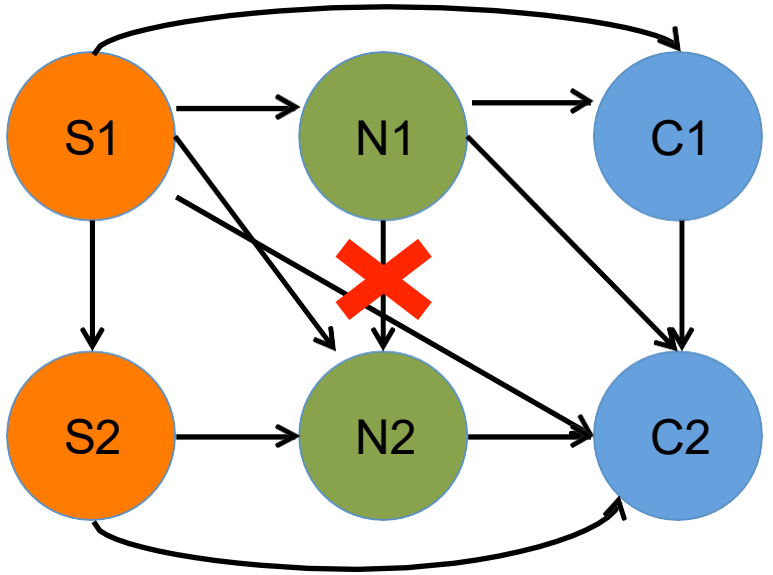


Trace 1

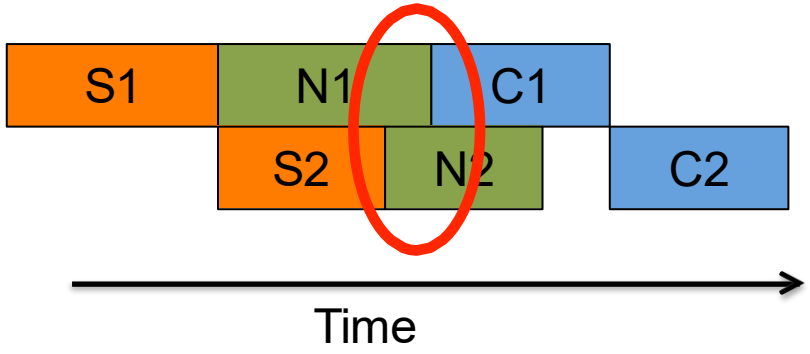


Producing Causal Model

Causal Model

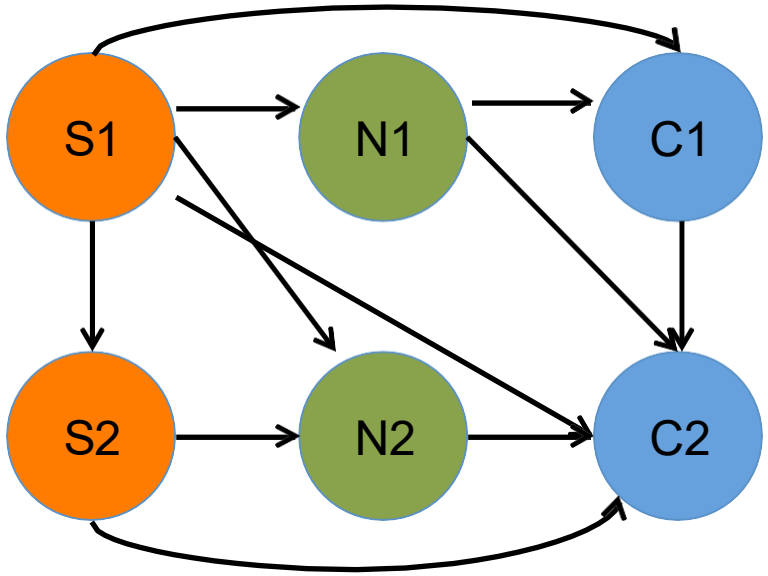


Trace 2

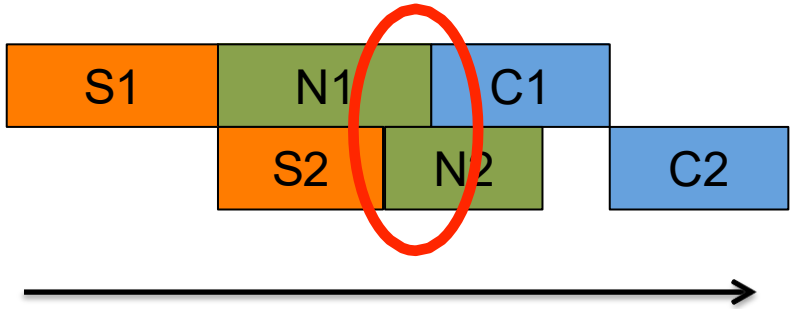


Producing Causal Model

Causal Model

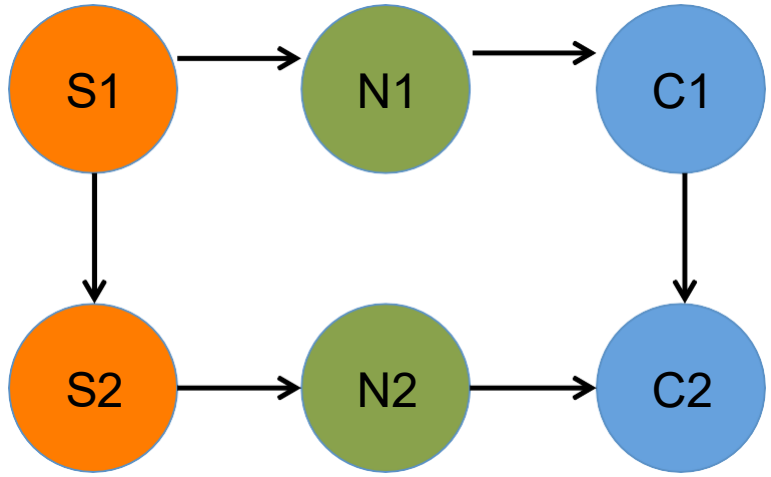


Trace 2

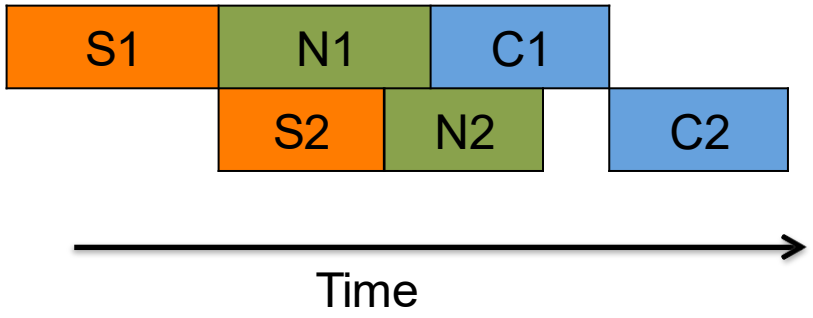


Producing Causal Model

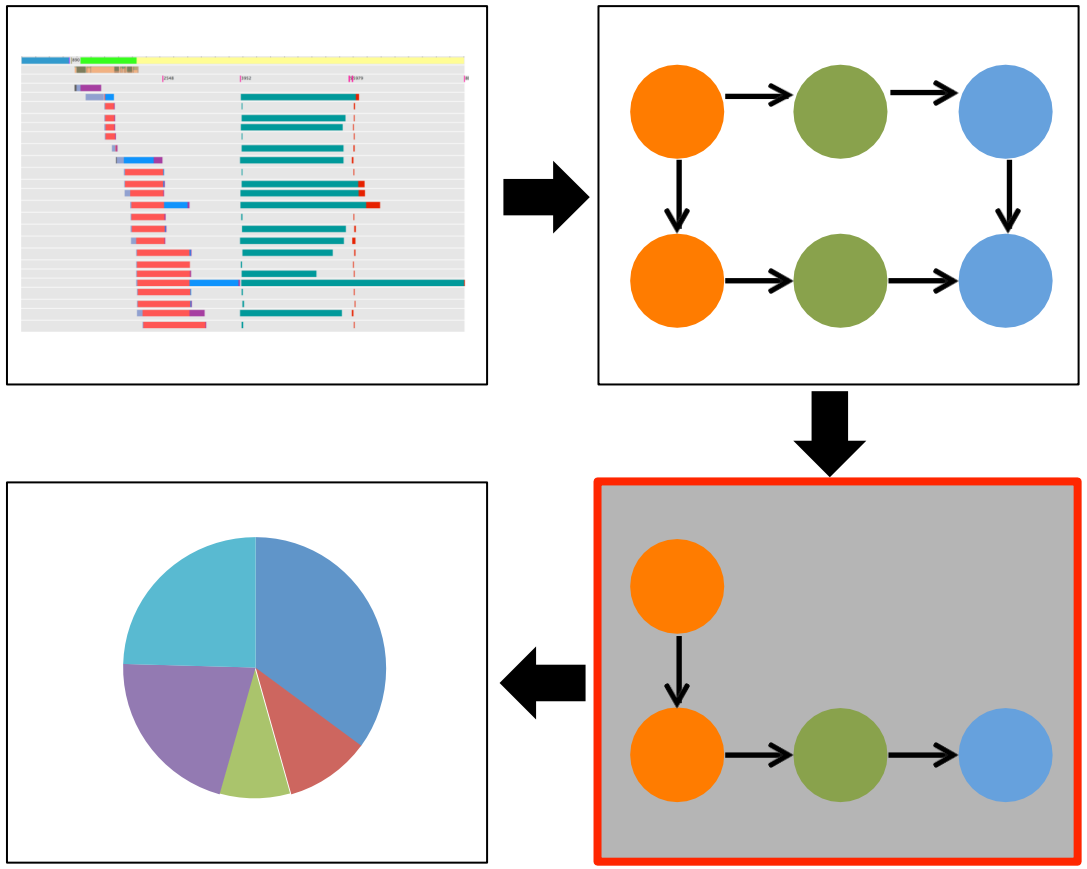
Causal Model



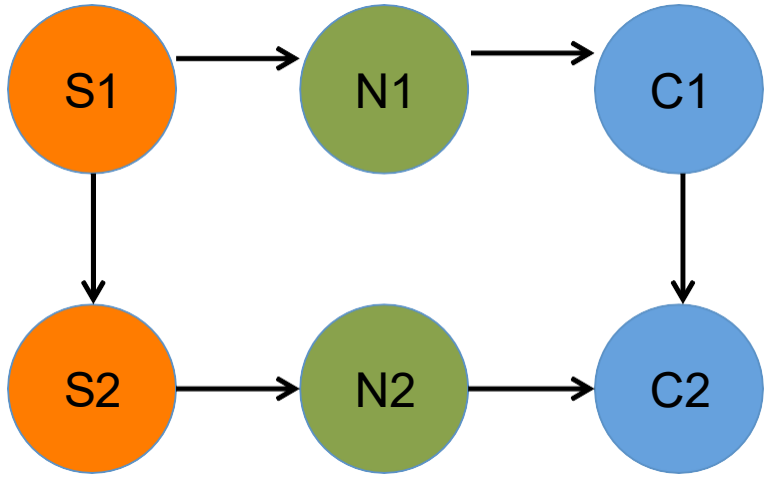
Trace 2

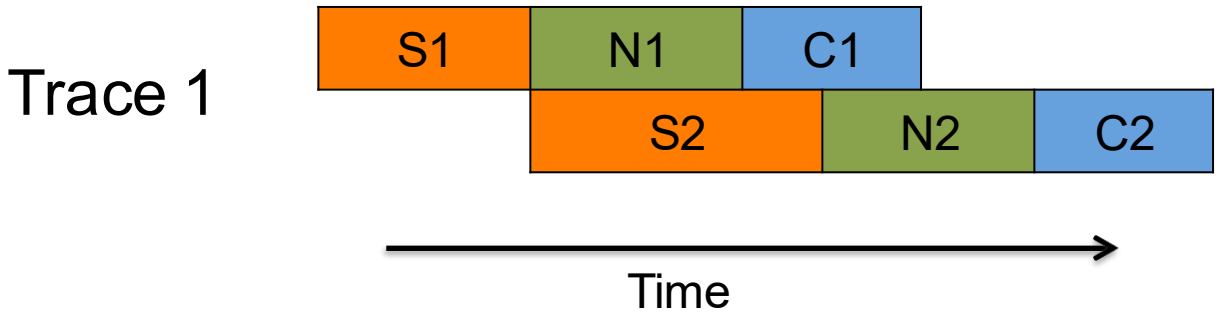


Step 3: Analyze Individual Requests

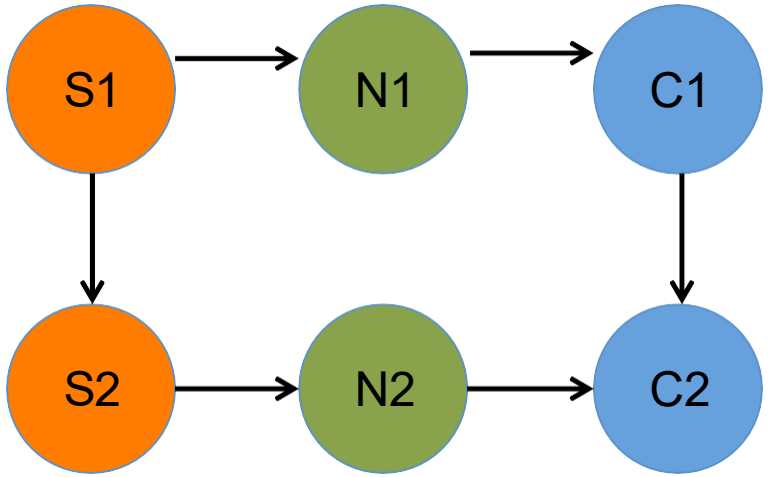


Critical Path Using Causal Model

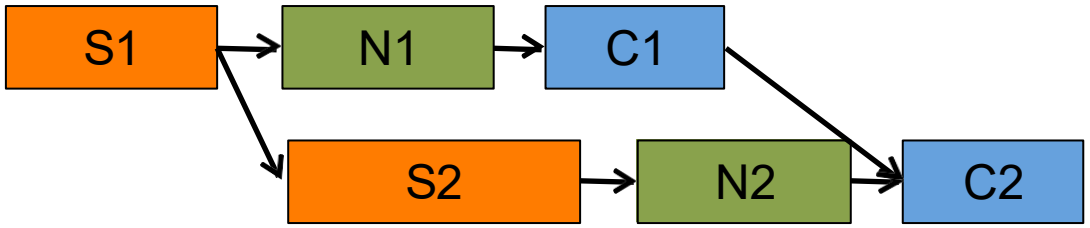




Critical Path Using Causal Model



Trace 1



Critical Path Using Causal Model