Pivot Tracing
Dynamic Causal Monitoring for Distributed Systems

Jonathan Mace, Ryan Roelke, Rodrigo Fonseca

Brown University

Pivot Tracing
Dynamic Causal Monitoring for Distributed Systems

Dynamically instrument live distributed systems

Pivot Tracing

Dynamic Causal Monitoring for Distributed Systems

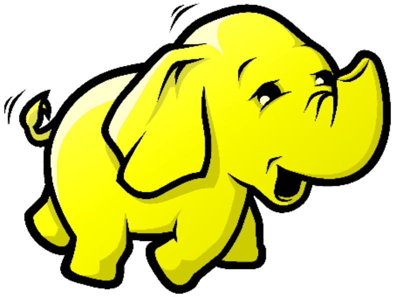
Dynamically instrument live distributed systems

Correlate and group events across components

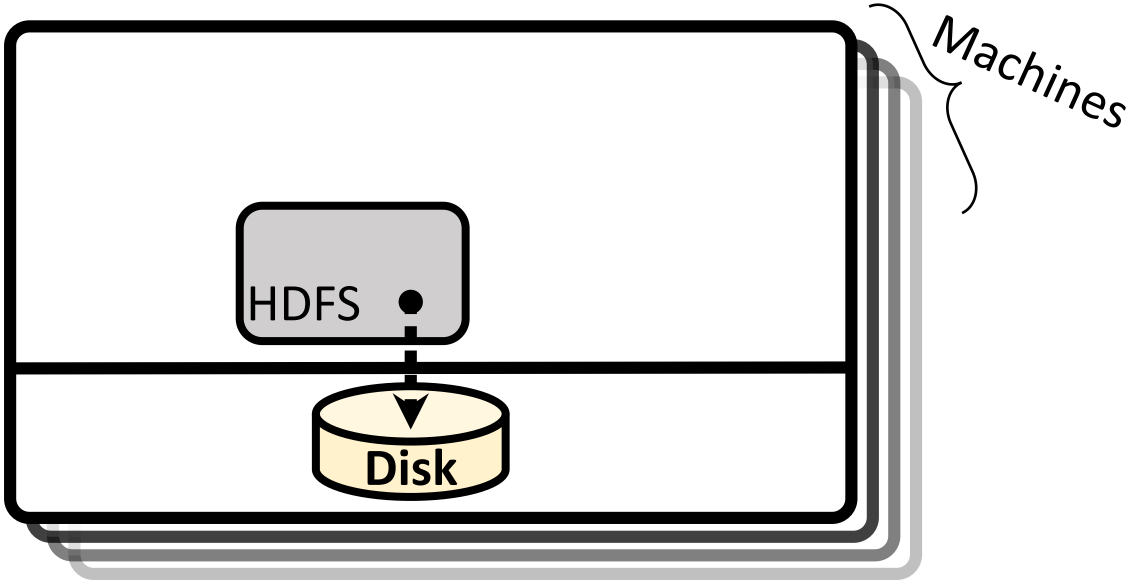
Pivot Tracing
Dynamic Causal Monitoring for Distributed Systems

Dynamically instrument live distributed systems

Correlate and group events across components



Hadoop Stack

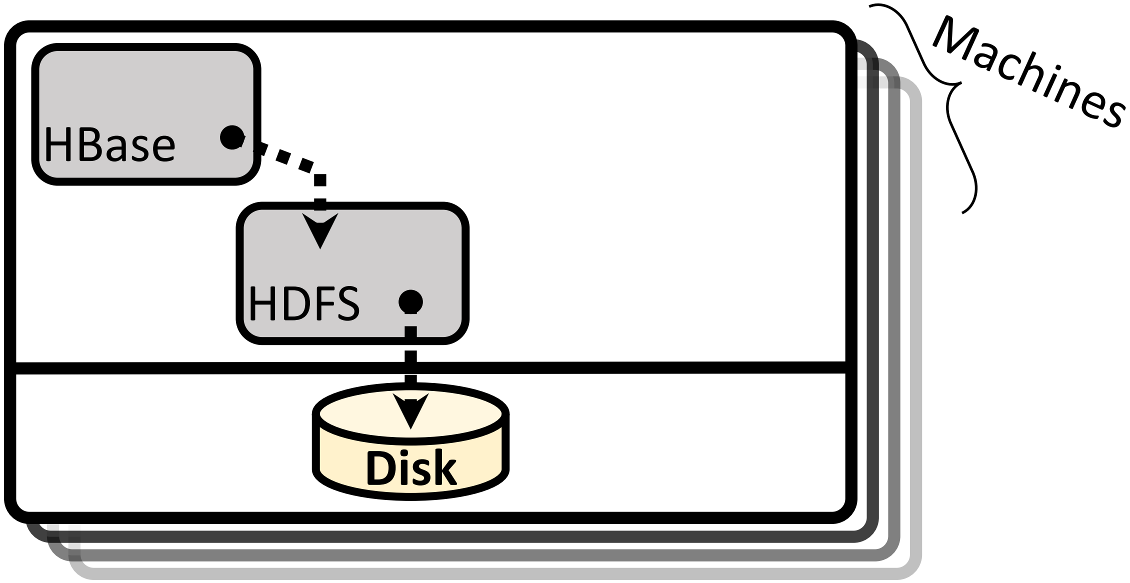


Machines

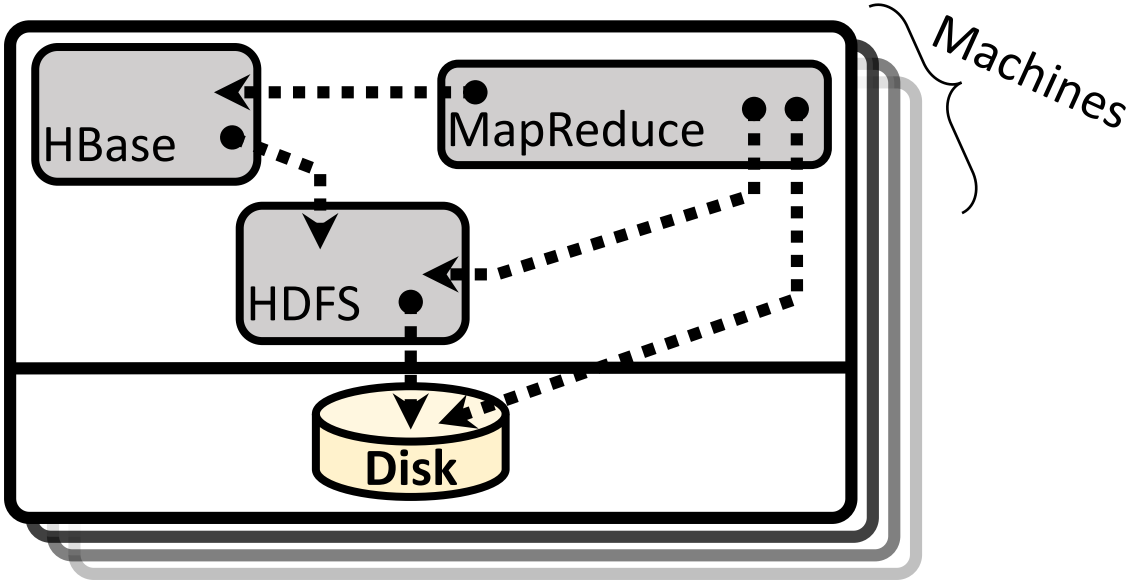
HDFS

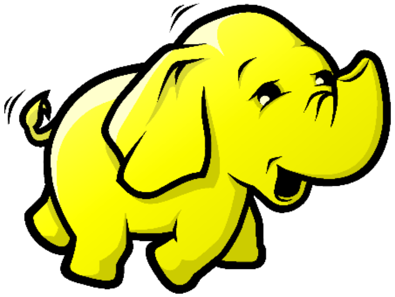
Disk

Hadoop Stack

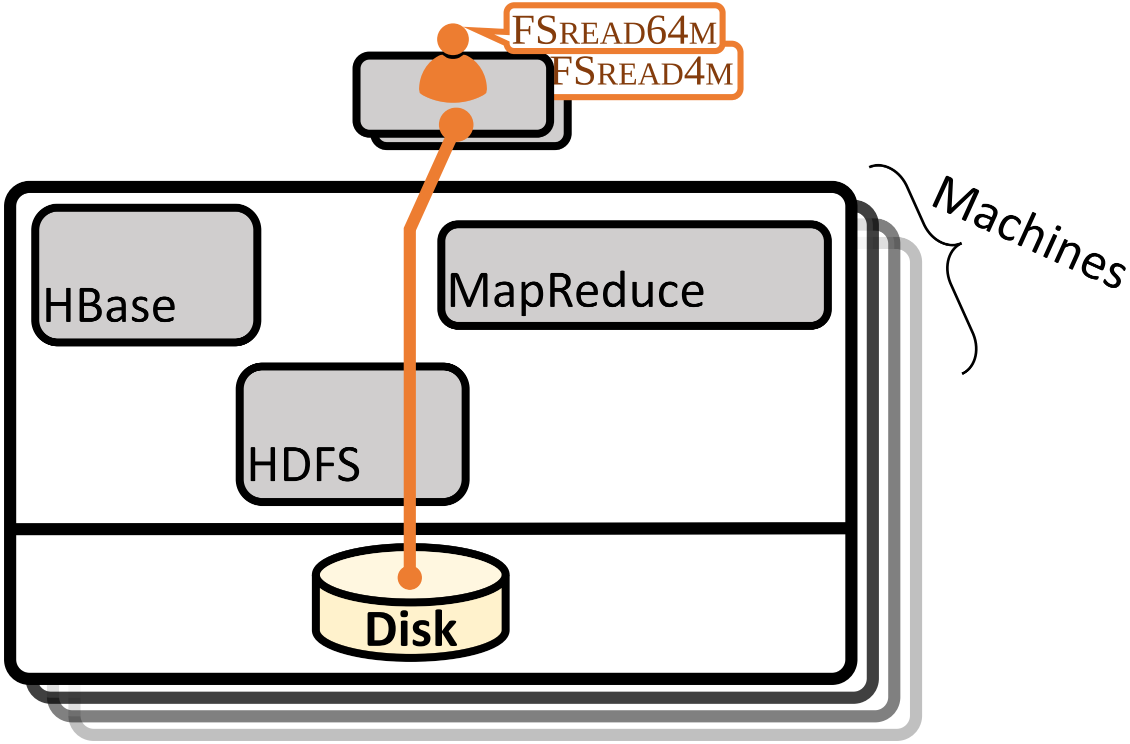


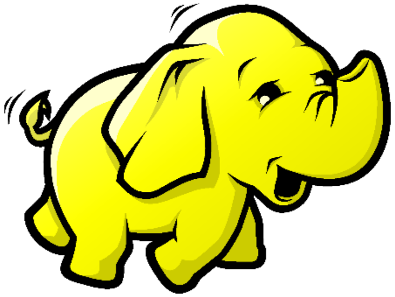
Hadoop Stack



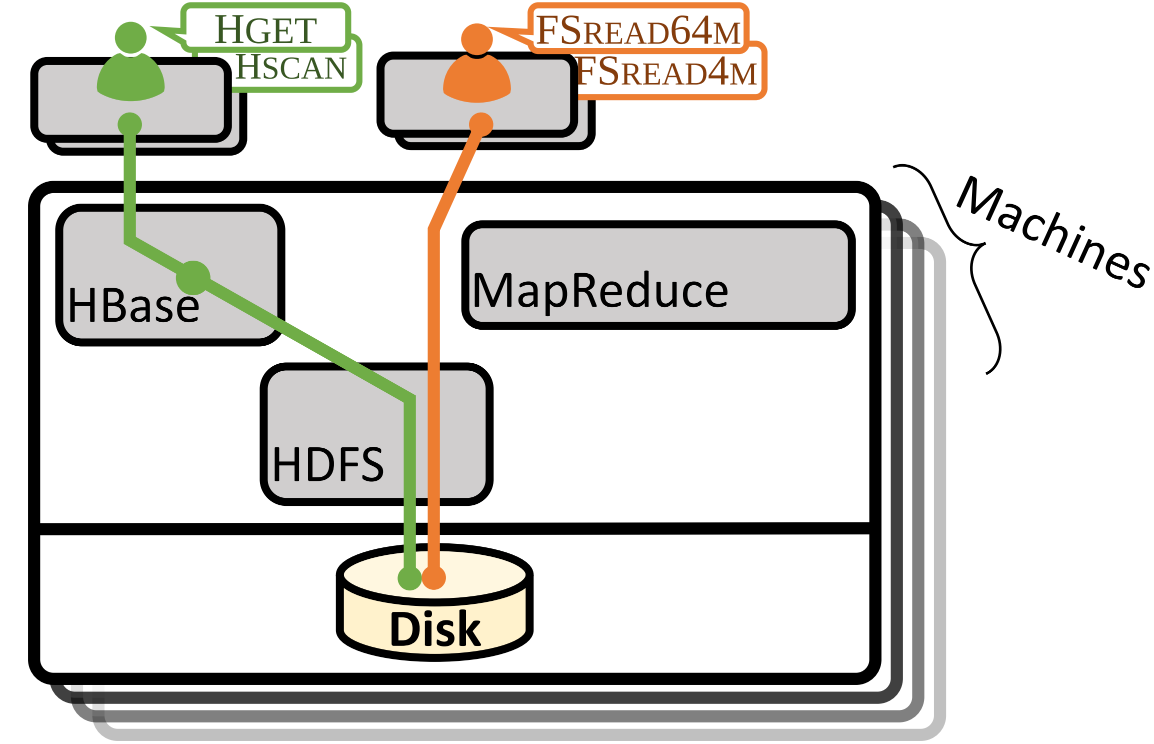


Hadoop Stack

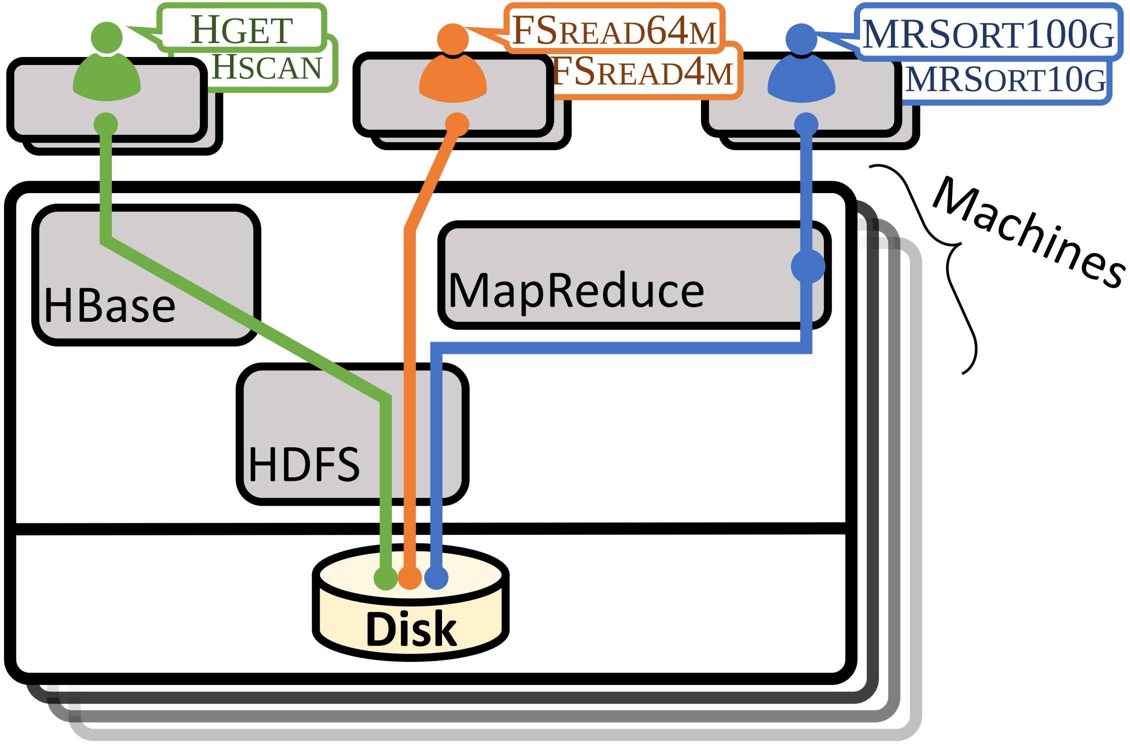




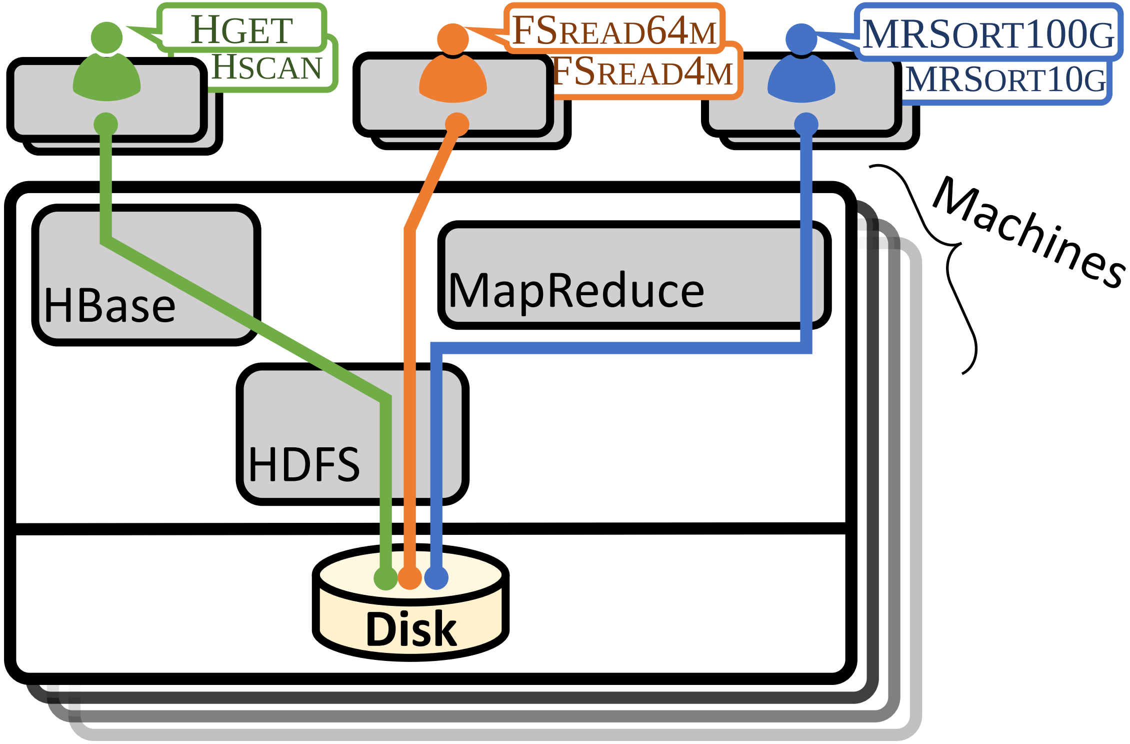
Hadoop Stack



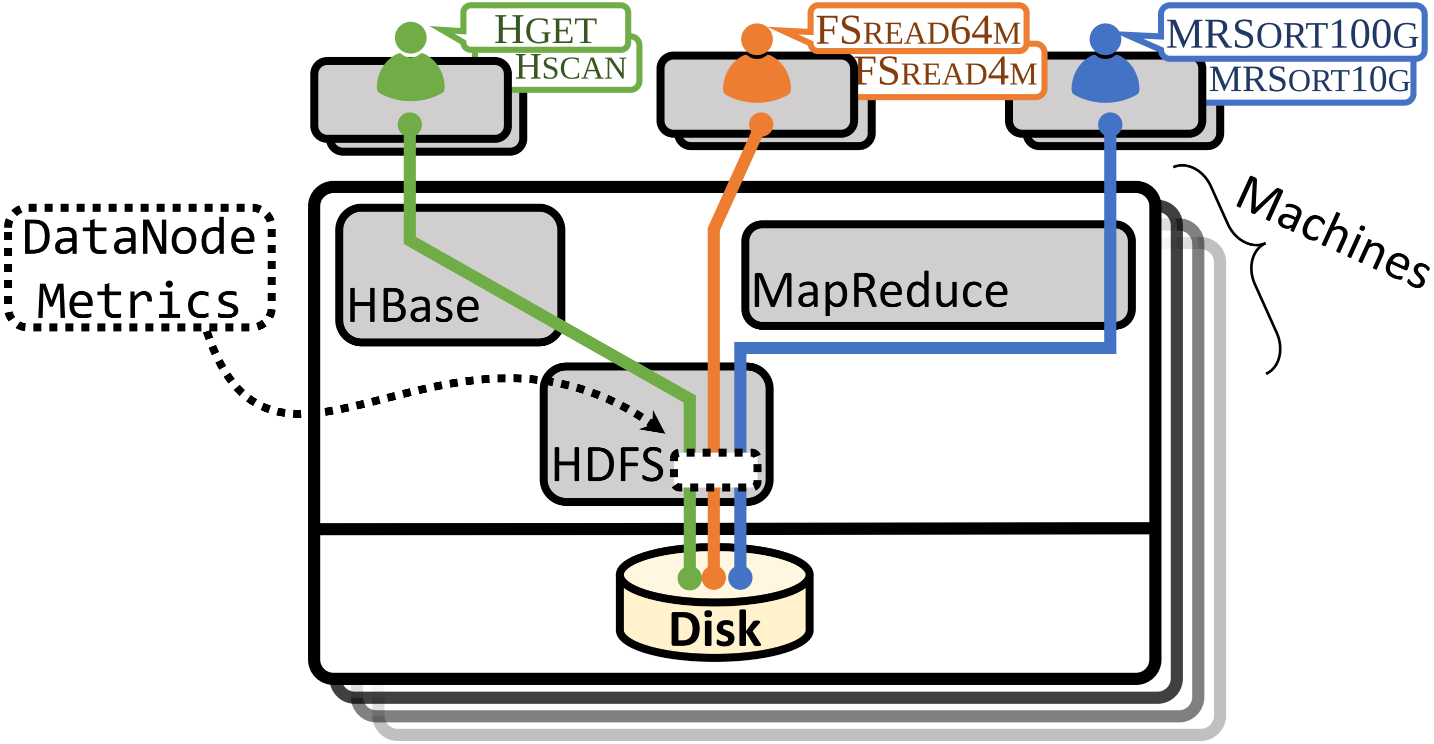
Hadoop Stack



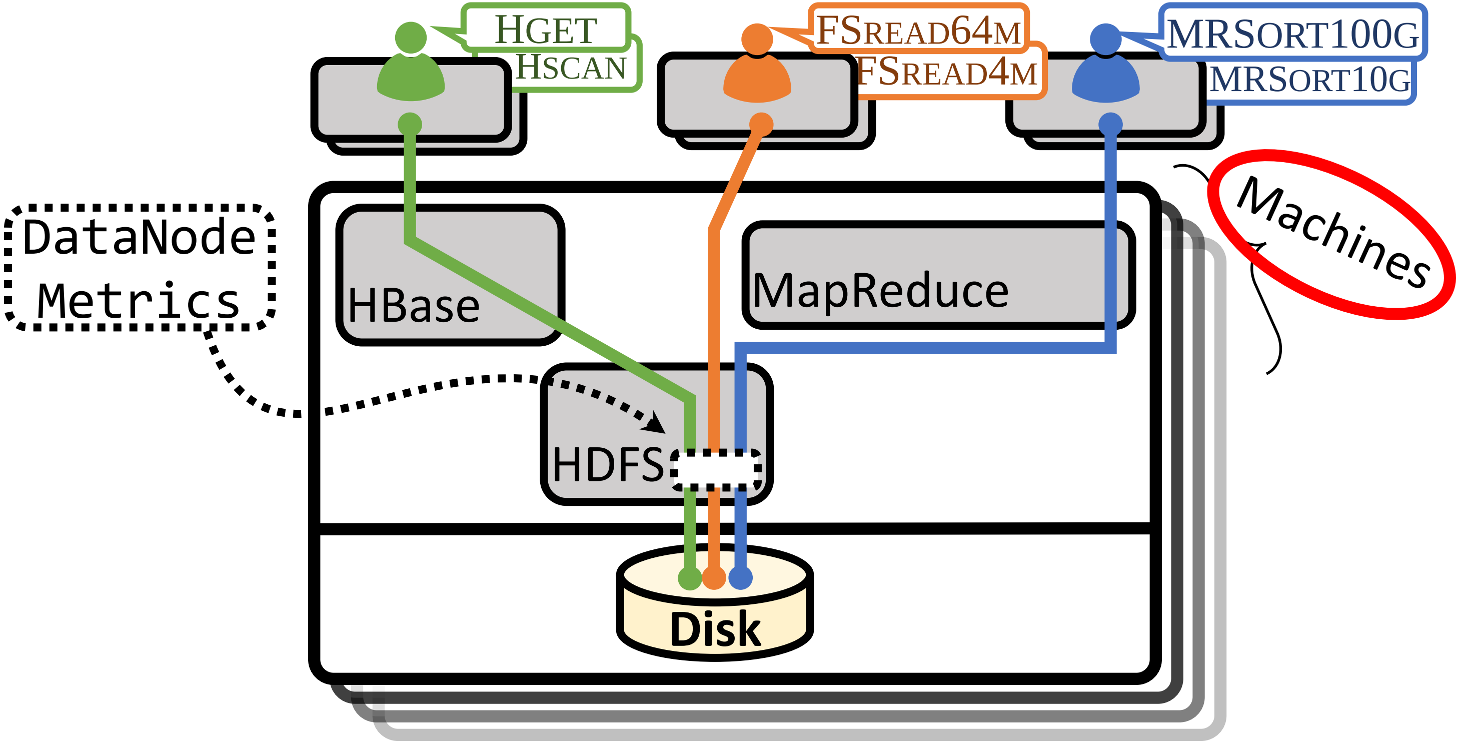
Hadoop Stack



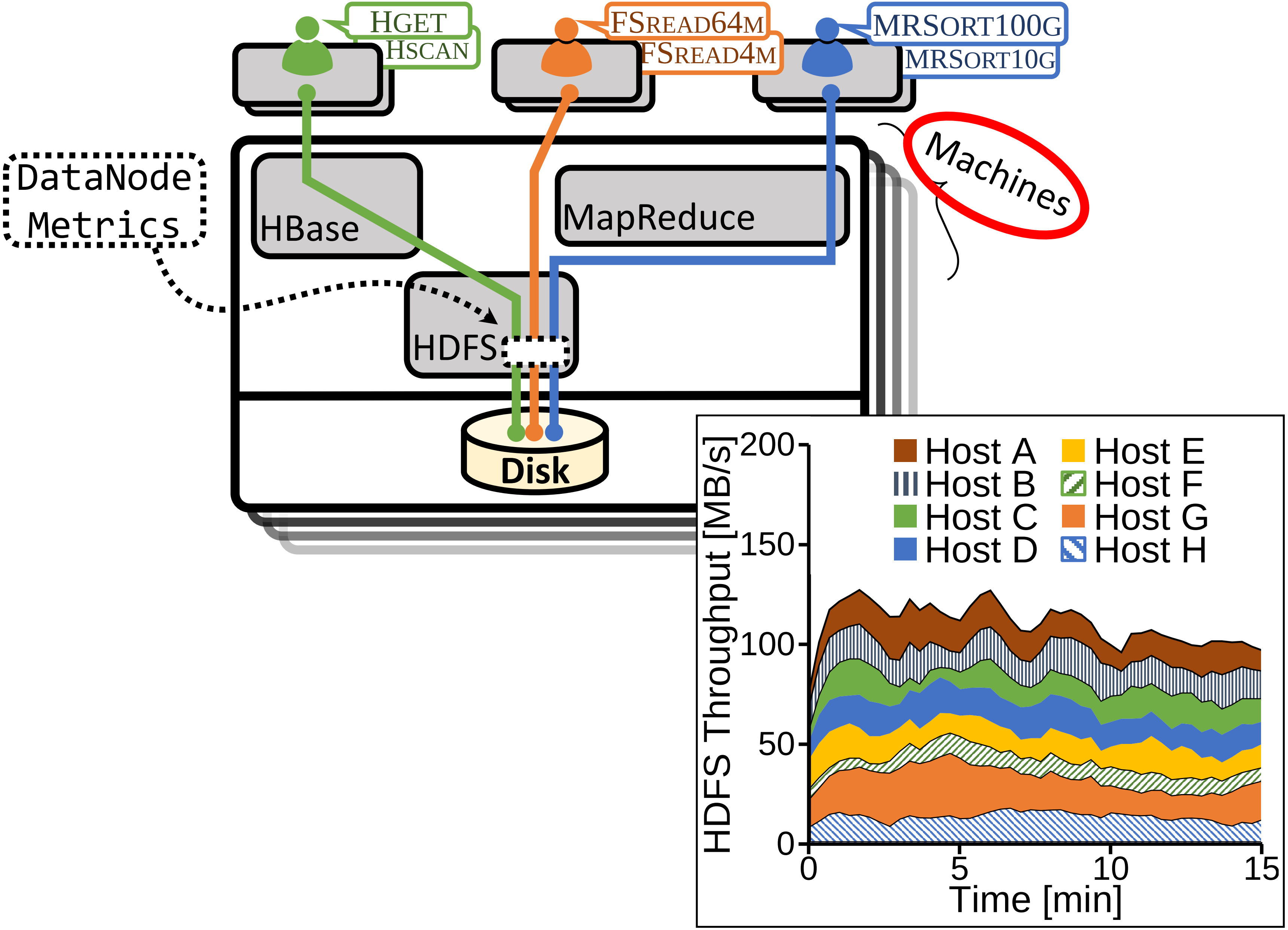
How is disk bandwidth being used?



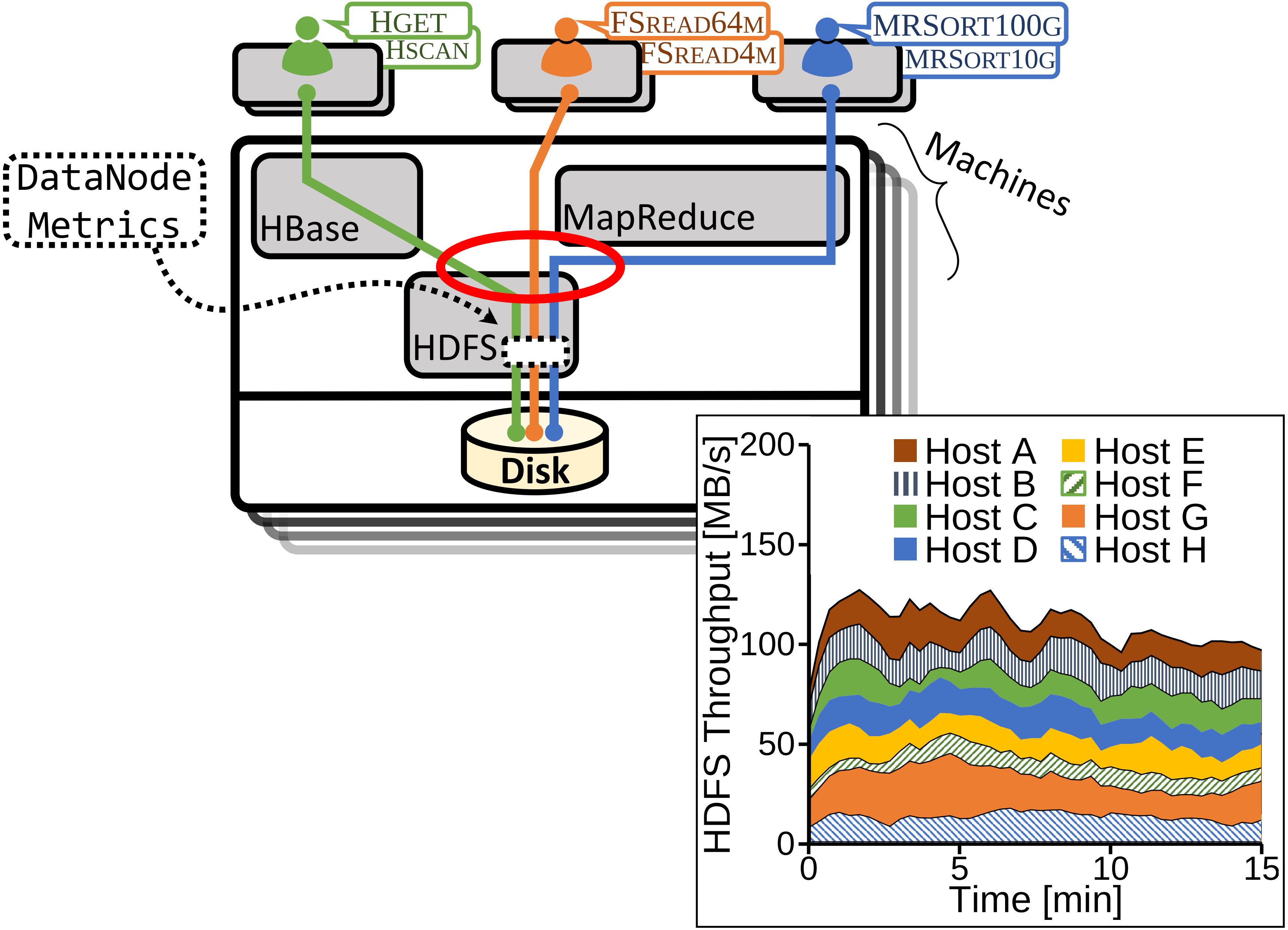
How is disk bandwidth being used?

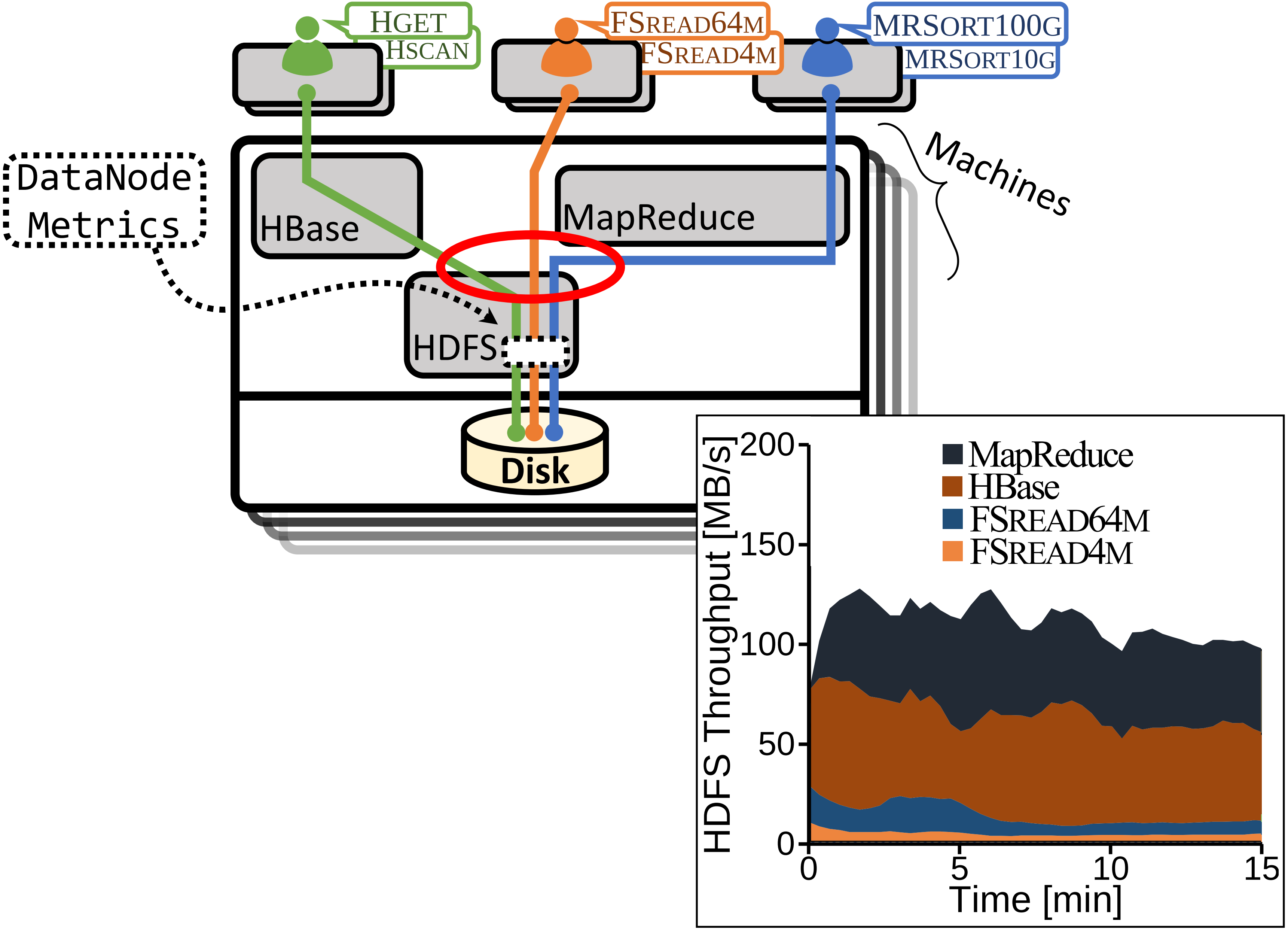


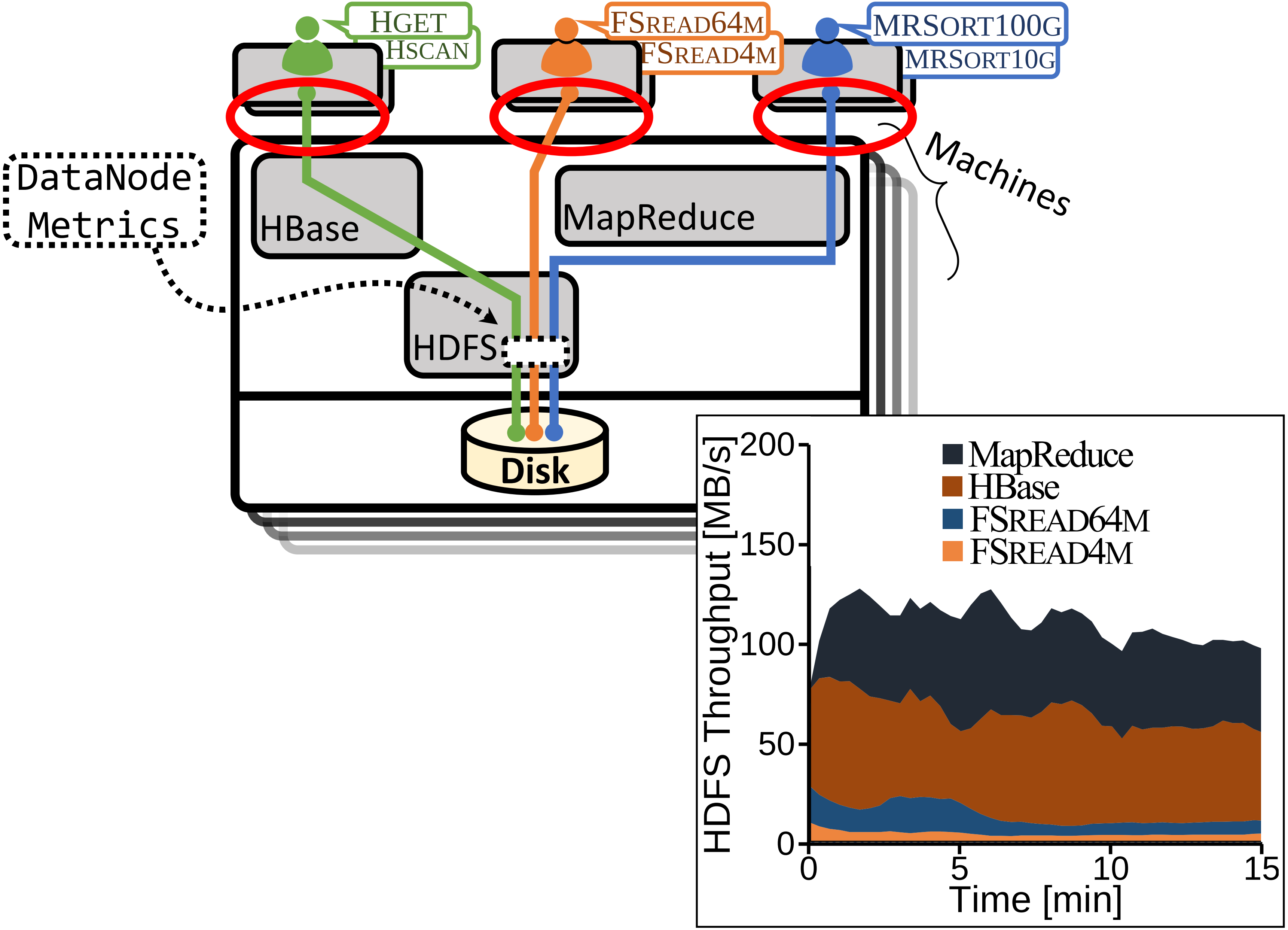
How is disk bandwidth being used?

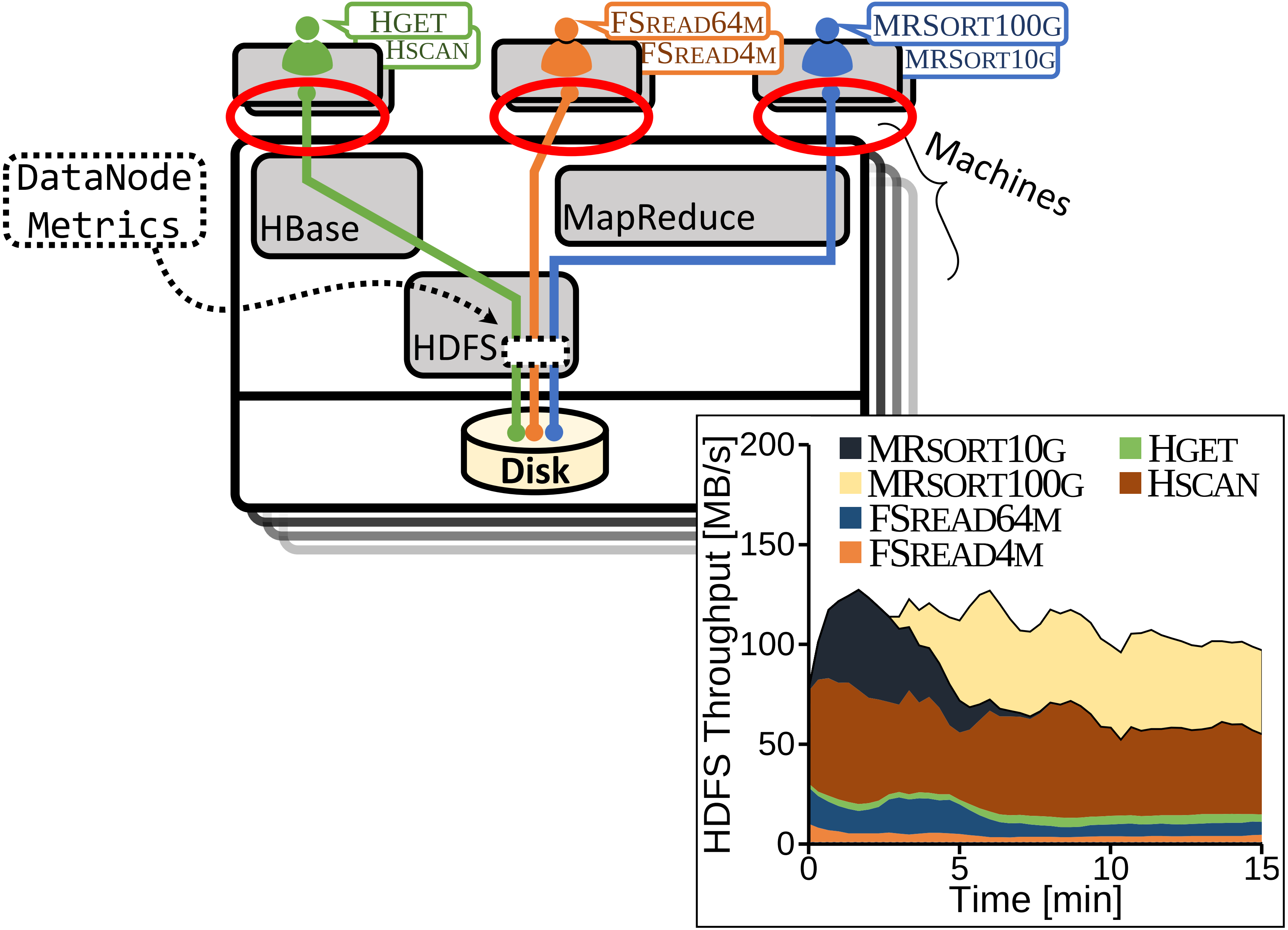


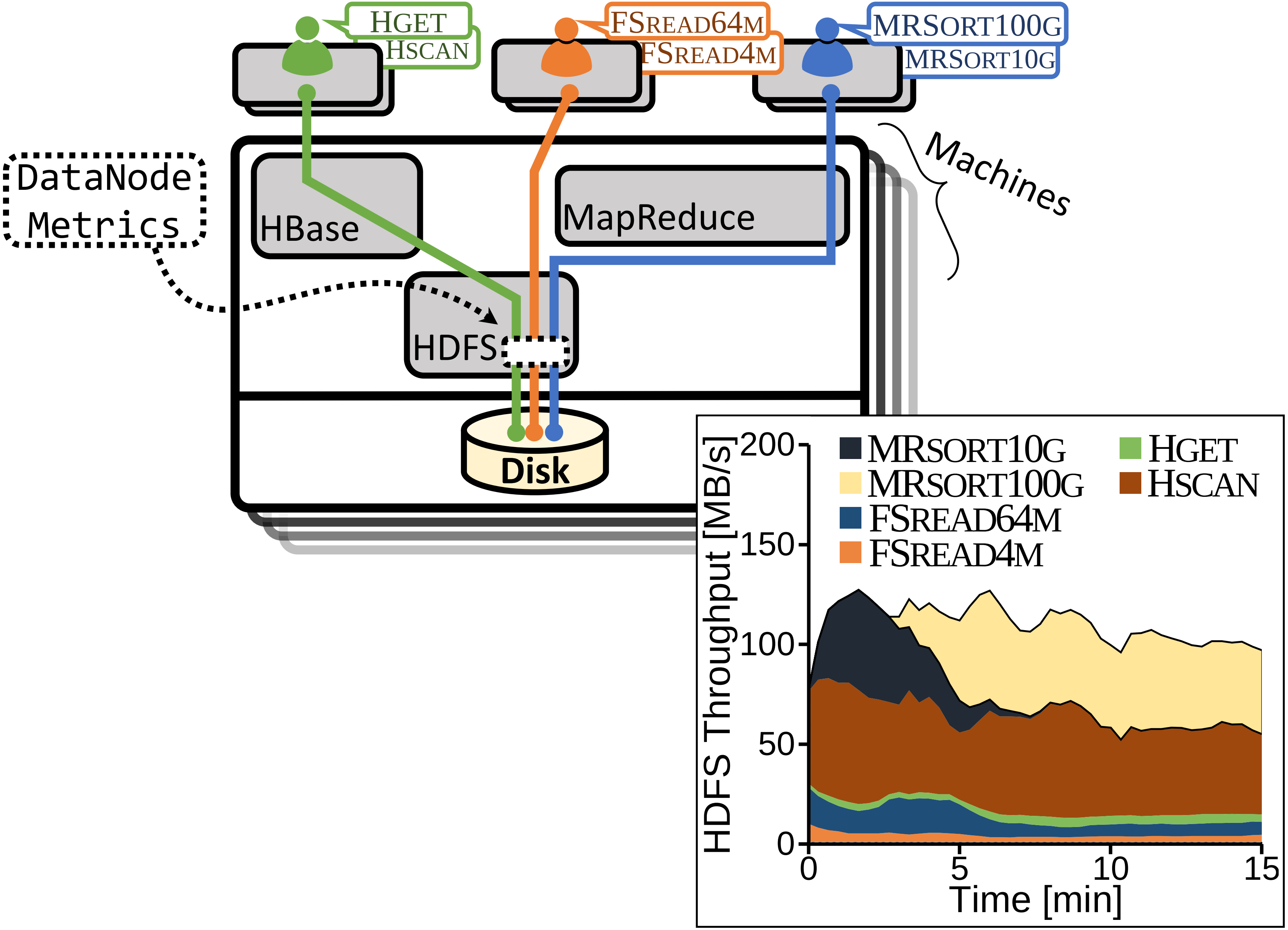
HGET
HSCAN
FSREAD64M
FSREAD4M
MRSORT100G
MRSORT10G
Machines
DataNode Metrics
HBase
MapReduce
HDFS
Disk
HDFS Throughput [MB/s]
200
150
100
50
0
0
5
10
15
Time [min]
Host A
Host B
Host C
Host D
Host E
Host F
Host G
Host H


HGET
HSCAN
FSREAD64M
FSREAD4M
MRSORT100G
MRSORT10G
Machines
DataNode Metrics
HBase
MapReduce
HDFS
Disk
HDFS Throughput [MB/s]
0
50
100
150
200
0
5
10
15
Time [min]
Host A
Host B
Host C
Host D
Host E
Host F
Host G
Host H

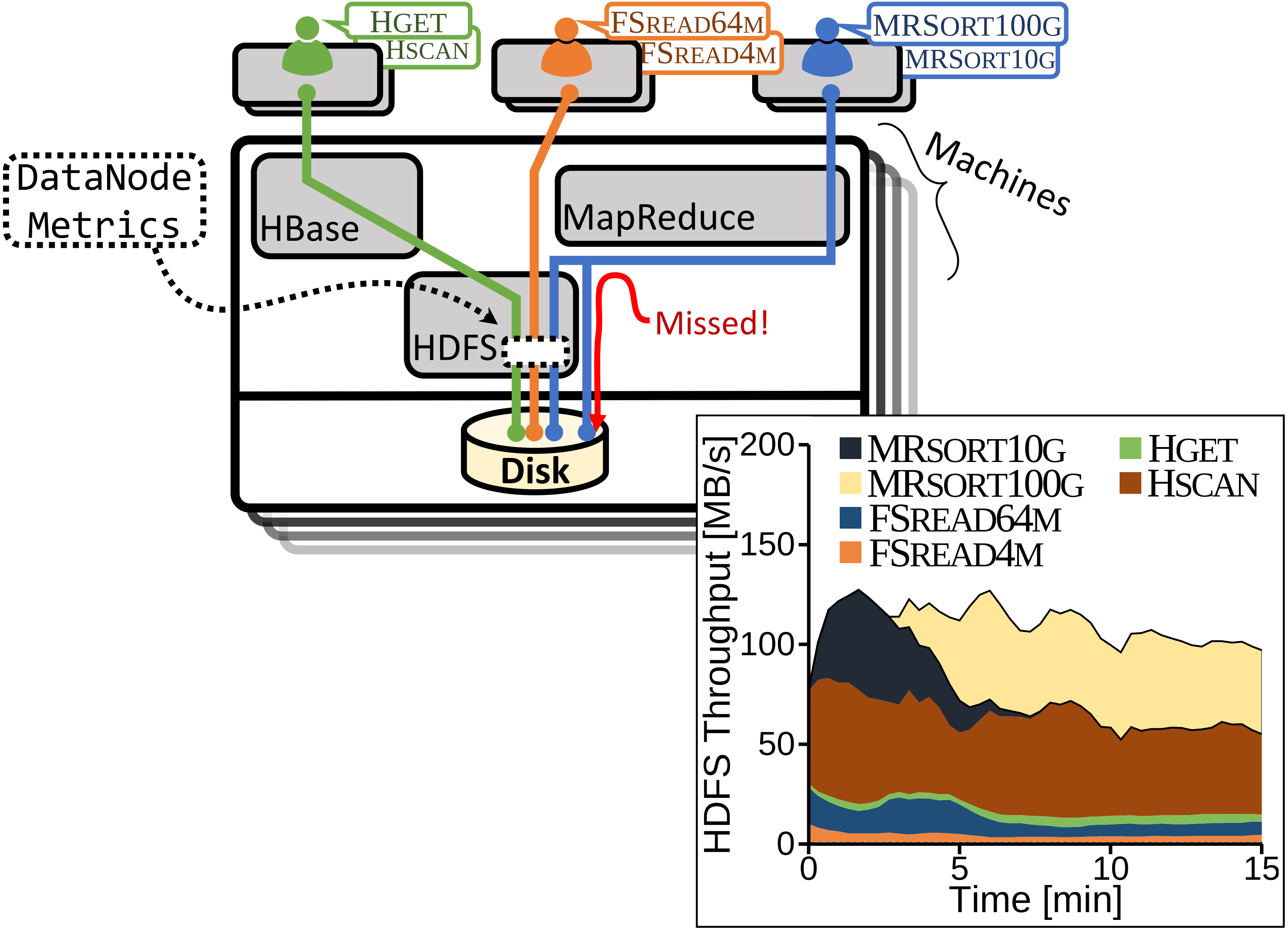

HGET
HSCAN
FSREAD64M
FSREAD4M
MRSORT100G
MRSORT10G
Machines
DataNode Metrics
HBase
MapReduce
HDFS
Disk
HDFS Throughput [MB/s]
200
150
100
50
0
0
5
10
15
Time [min]
MapReduce
HBase
FSREAD64M
FSREAD4M


HGET
HSCAN
FSREAD64M
FSREAD4M
MRSORT100G
MRSORT10G
DataNode Metrics
HBase
MapReduce
HDFS
Disk
Machines
MapReduce
HBase
FSREAD64M
FSREAD4M
HDFS Throughput [MB/s]
0
50
100
150
200
Time [min]
0
5
10
15

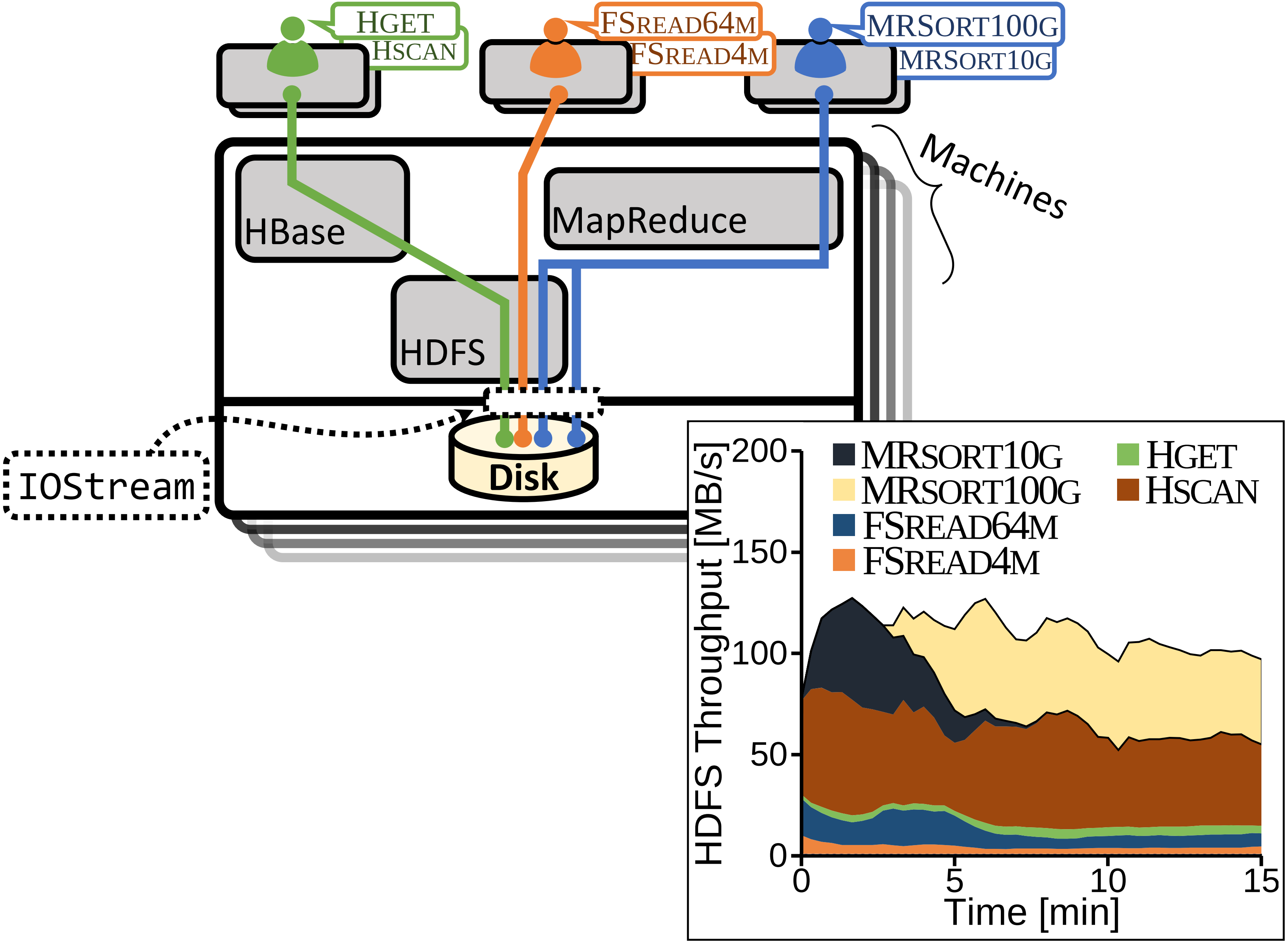

HGET
HSCAN
FSREAD64M
FSREAD4M
MRSORT100G
MRSORT10G
DataNode Metrics
HBase
MapReduce
HDFS
Disk
Machines
HDFS Throughput [MB/s]
200
150
100
50
0
Time [min]
0
5
10
15
MRSORT10G
MRSORT100G
FSREAD64M
FSREAD4M
HGET
HSCAN



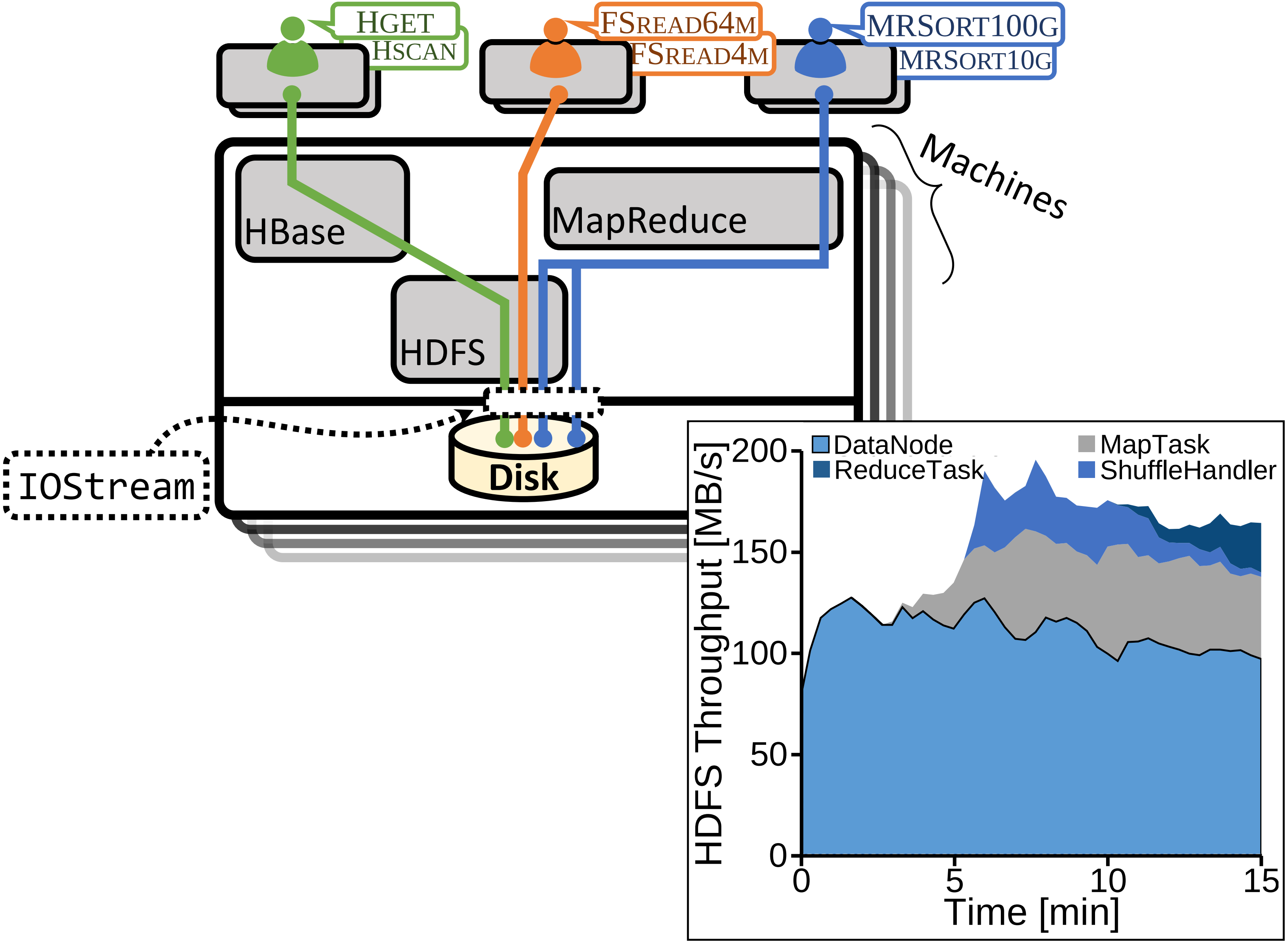
How is disk bandwidth being used?



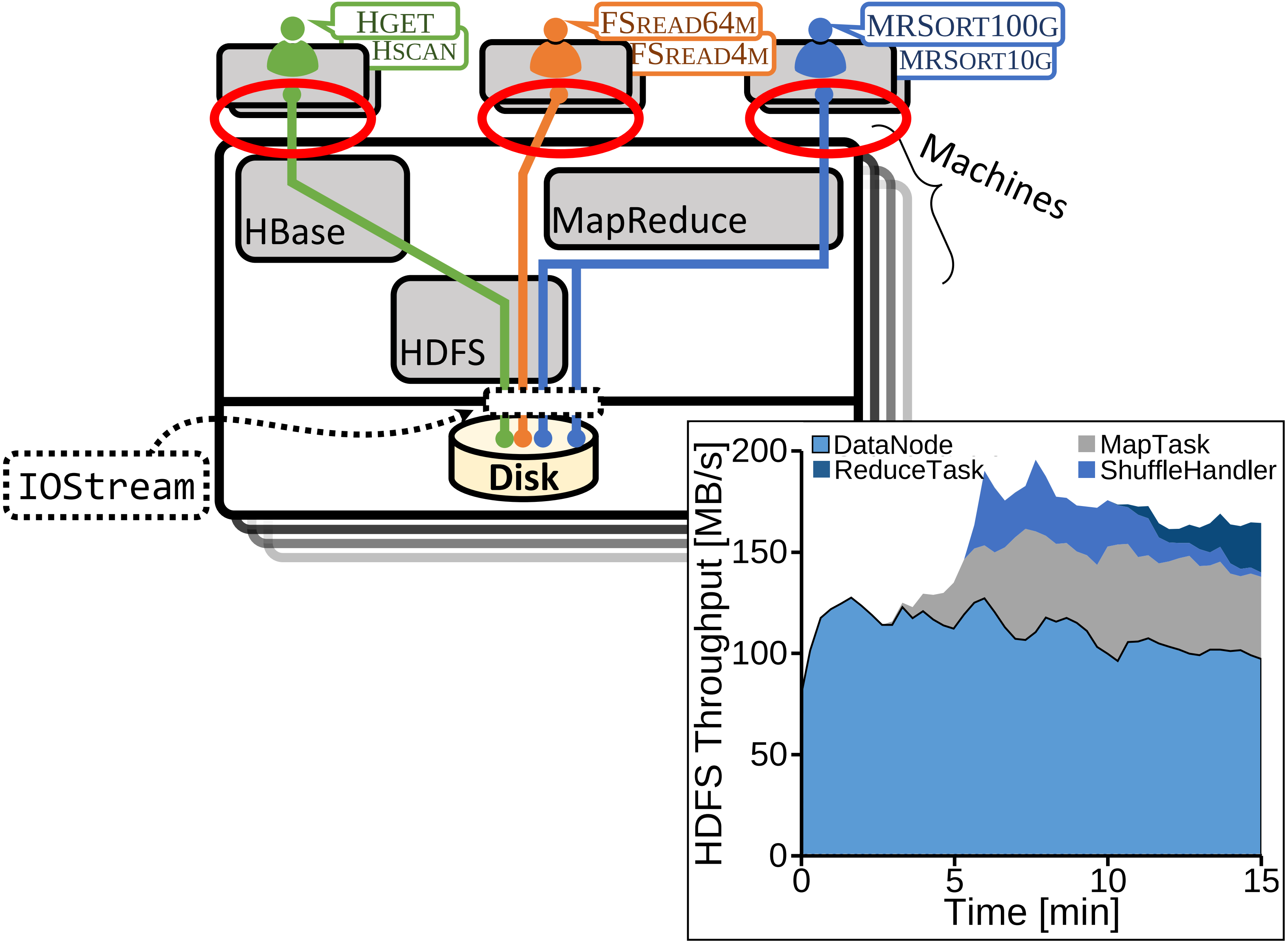
How is disk bandwidth being used?

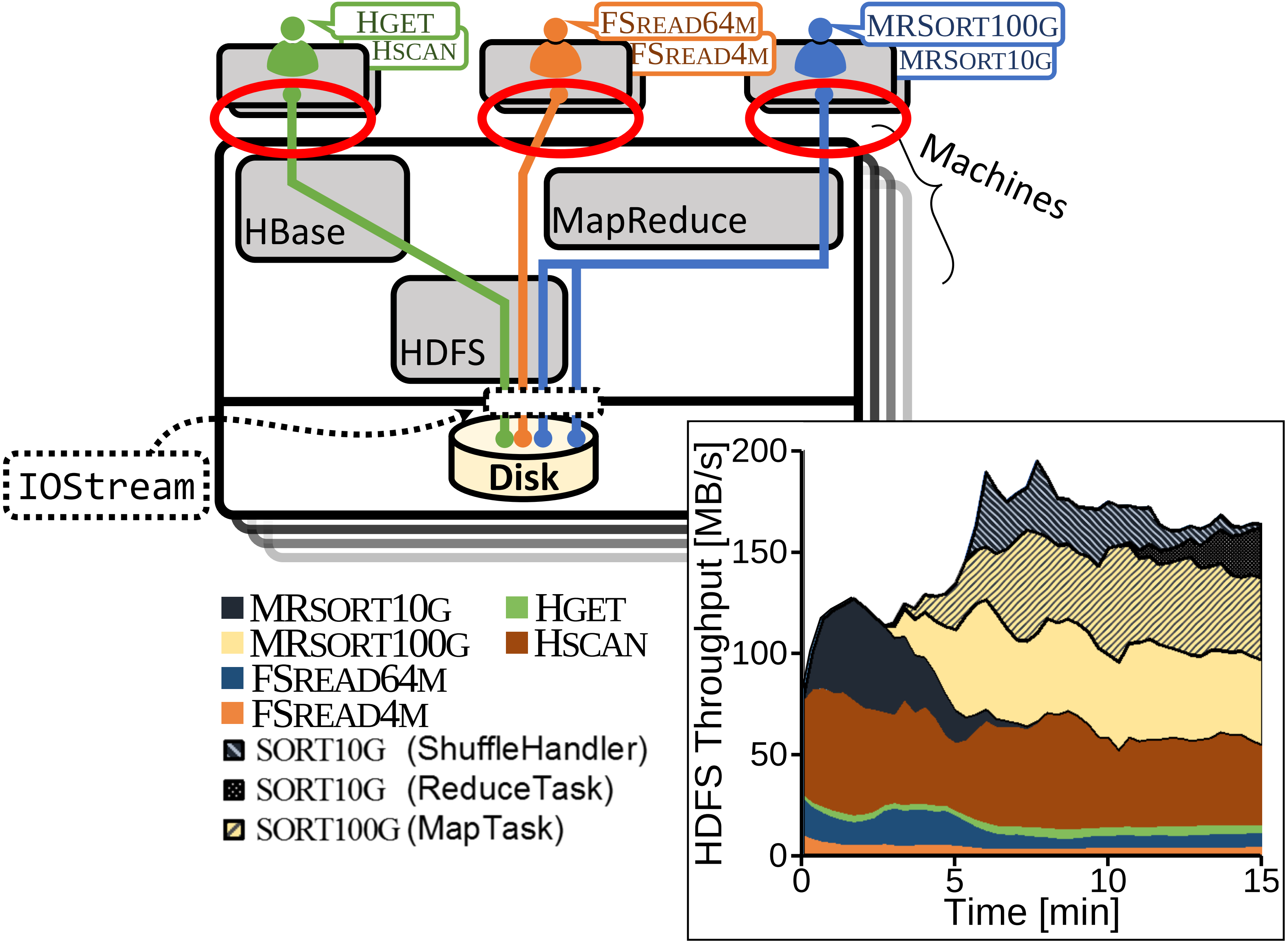


How is disk bandwidth being used?



How is disk bandwidth being used?


HGET
HSCAN
FSREAD64M
FSREAD4M
MRSORT100G
MRSORT10G
Machines
HBase
MapReduce
HDFS
IOStream
Disk
DataNode
ReduceTask
MapTask
ShuffleHandler
HDFS Throughput [MB/s]
200
150
100
50
0
0
5
10
15
Time [min]


HGET
HSCAN
FSREAD64M
FSREAD4M
MRSORT100G
MRSORT10G
Machines
HBase
MapReduce
HDFS
Disk
IOStream
HDFS Throughput [MB/s]
200
150
100
50
0
0
5
10
15
Time [min]
MRSORT10G
MRSORT100G
FSREAD64M
FSREAD4M
HGET
HSCAN
SORT10G (ShuffleHandler)
SORT10G (ReduceTask)
SORT100G (MapTask)

Instrumentation is decided at development time

Instrumentation is decided at development time

Probably not have enough info for your problem

Probably too much irrelevant info for your problem

Instrumentation is decided at development time

Probably not have enough info for your problem

Probably too much irrelevant info for your problem

Should every user bear the cost of a feature?

Instrumentation is decided at development time

Probably not have enough info for your problem

Probably too much irrelevant info for your problem

Should every user bear the cost of a feature?

HDFS-6292 Display HDFS per user and per group usage on webUI.

https://issues.apache.org/jira/browse/HDFS-6292

Instrumentation is decided at development time

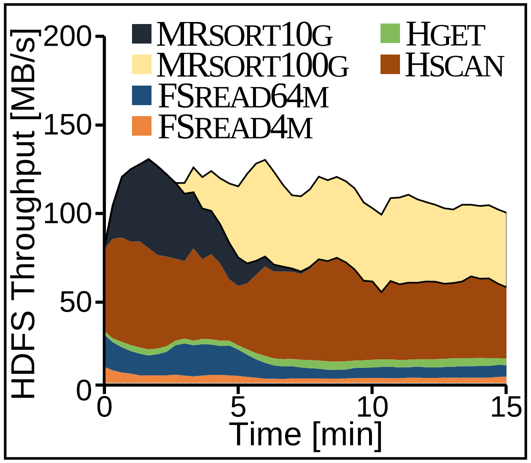
Probably not have enough info for your problem

Probably too much irrelevant info for your problem

Should every user bear the cost of a feature?

HDFS-6292 Display HDFS per user and per group usage on webUI.

https://issues.apache.org/jira/browse/HDFS-6292



Dynamic dependencies

You often need to correlate information from different points in the system

Dynamic dependencies

You often need to correlate information from different points in the system

Systems are designed to compose

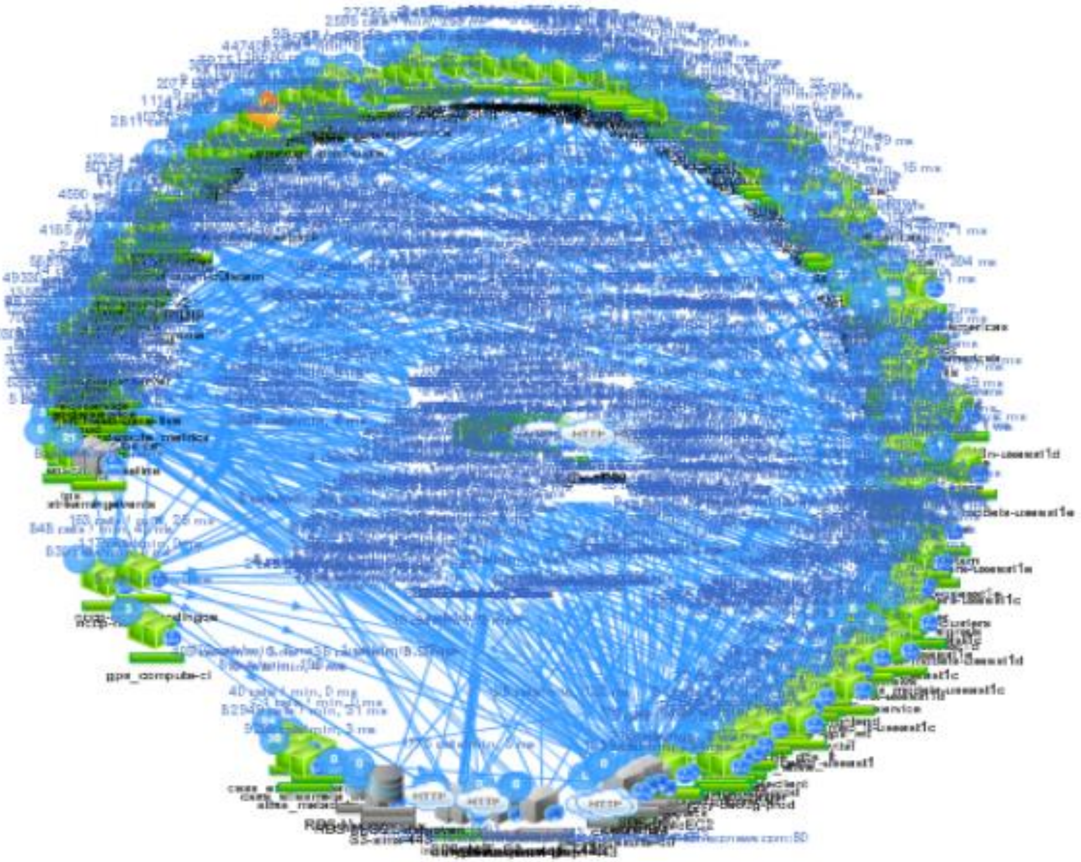
Systems don’t embed monitoring that relates to other services

Dynamic dependencies

You often need to correlate information from different points in the system

Systems are designed to compose

Systems don’t embed monitoring that relates to other services



Netflix “Death Star” Microservices Dependencies

@bruce_m_wong

Pivot Tracing

Pivot Tracing

You don’t know the questions in advance

Dynamic instrumentation

Fay (SOSP’11), Dtrace (ATC’04), …

Pivot Tracing

You don’t know the questions in advance

Dynamic instrumentation

Fay (SOSP’11), Dtrace (ATC’04), …

You often need to correlate information from different points in the system

Causal tracing

X-Trace (NSDI’07), Dapper (Google), Pip (NSDI’06), …

Pivot Tracing

Pivot Tracing

Model system events as tuples in a streaming, distributed dataset

Pivot Tracing

Model system events as tuples in a streaming, distributed dataset

Dynamically evaluate relational queries over this dataset

Pivot Tracing

Model system events as tuples in a streaming, distributed dataset

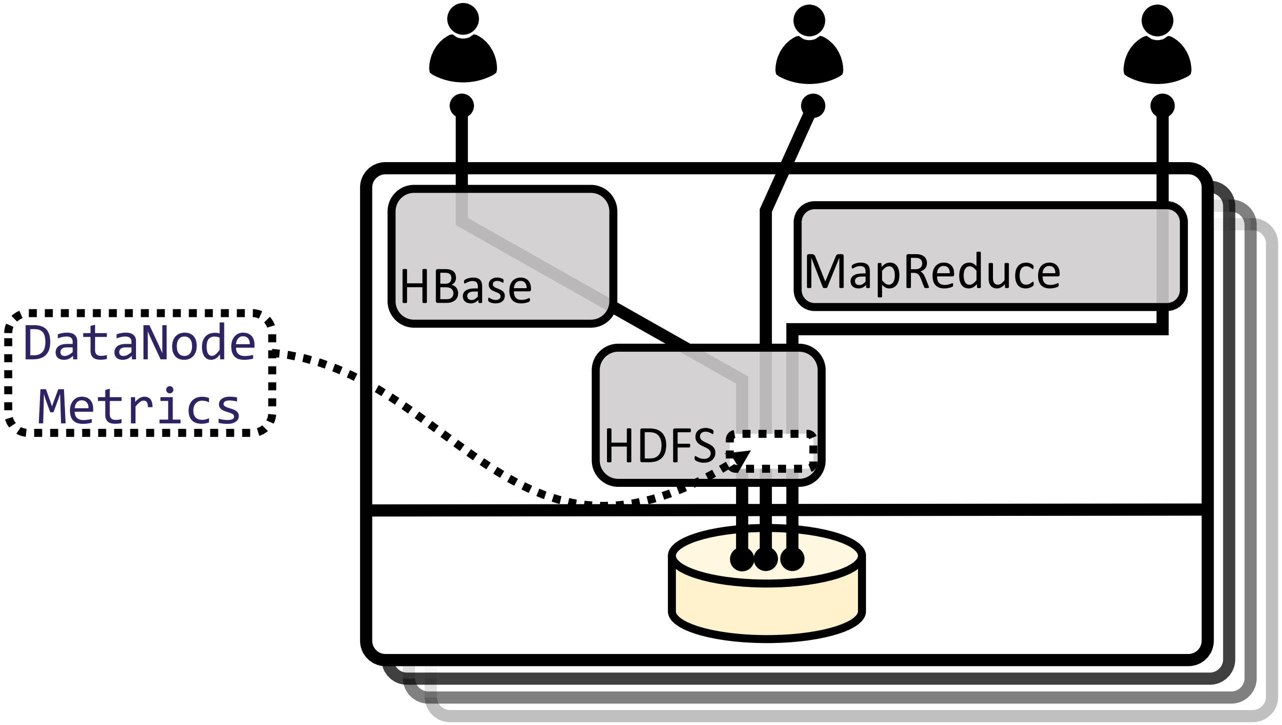
Dynamically evaluate relational queries over this dataset

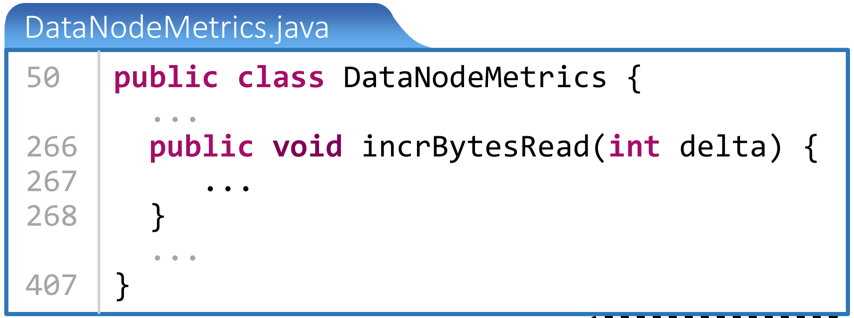
Happened-before Join ($\overrightarrow{\bowtie}$)

Join based on Lamport's *happened-before* relation

Pivot Tracing

Overview

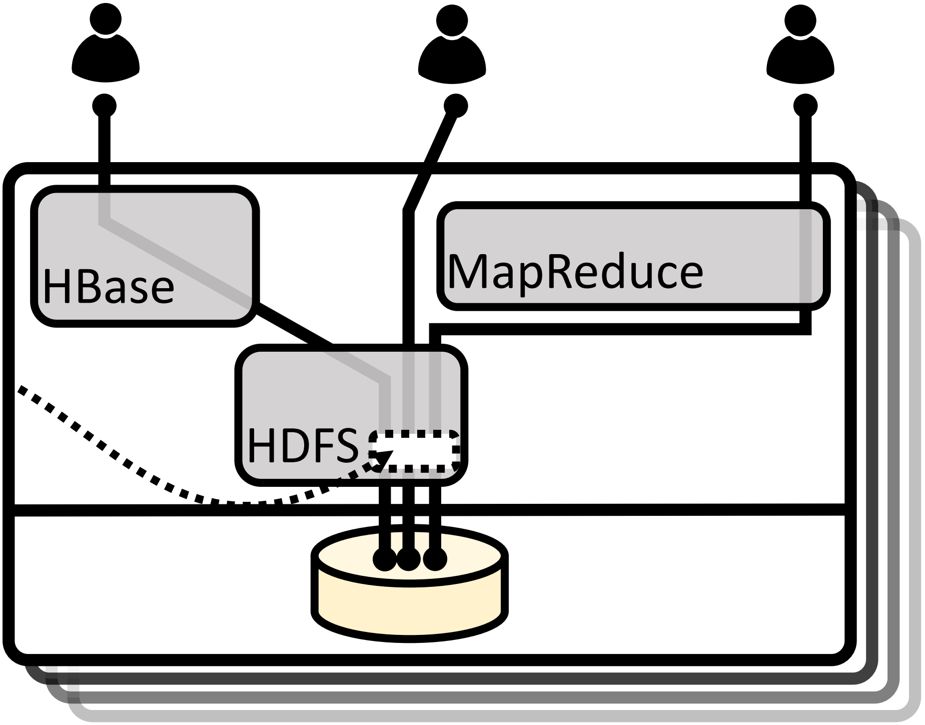

DataNode Metrics
HBase
MapReduce
HDFS

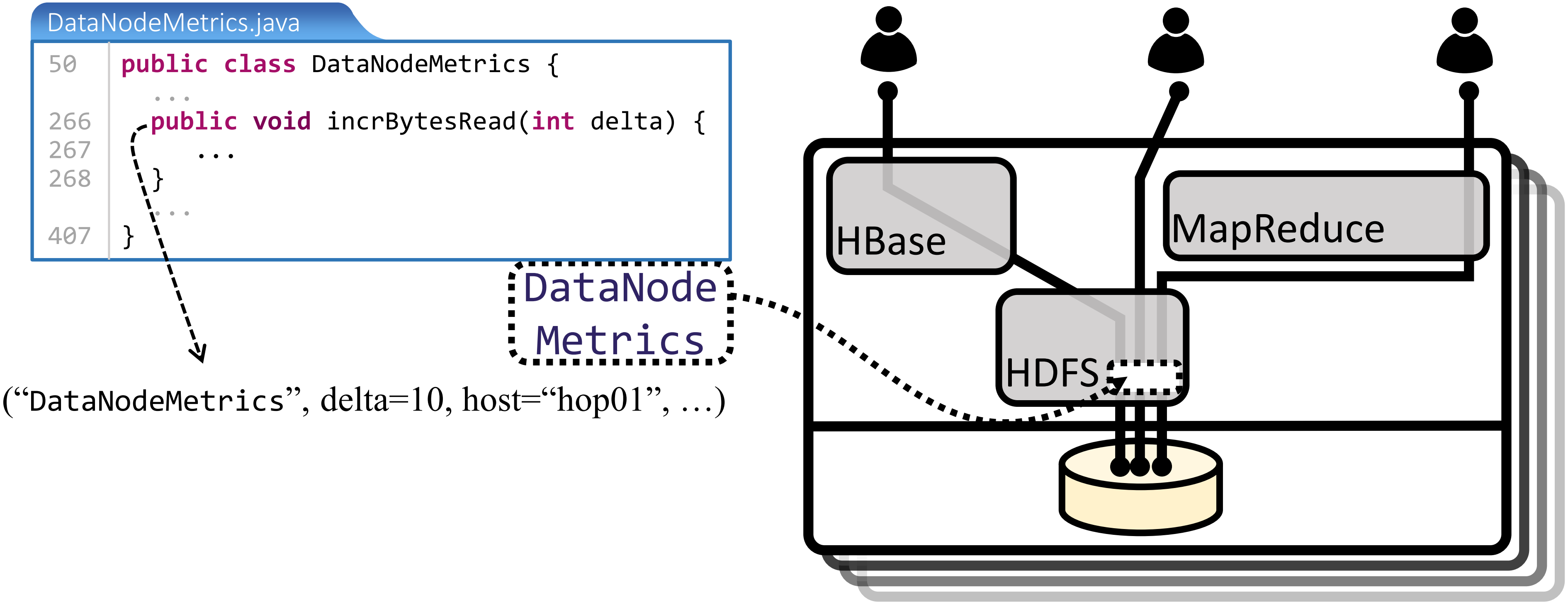


DataNodeMetrics.java

```
public class DataNodeMetrics {
  ...
  public void incrBytesRead(int delta) {
     ...
  }
  ...
}
```

DataNode Metrics

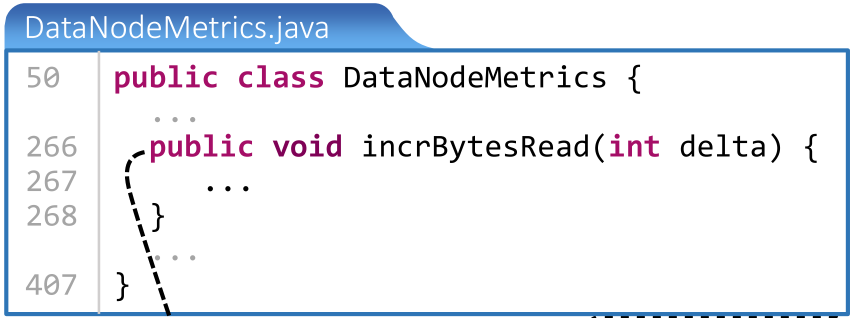




DataNodeMetrics.java

```
public class DataNodeMetrics {
  ...
  public void incrBytesRead(int delta) {
     ...
  }
  ...
}
```

(“DataNodeMetrics”, delta=10, host=“hop01”, …)

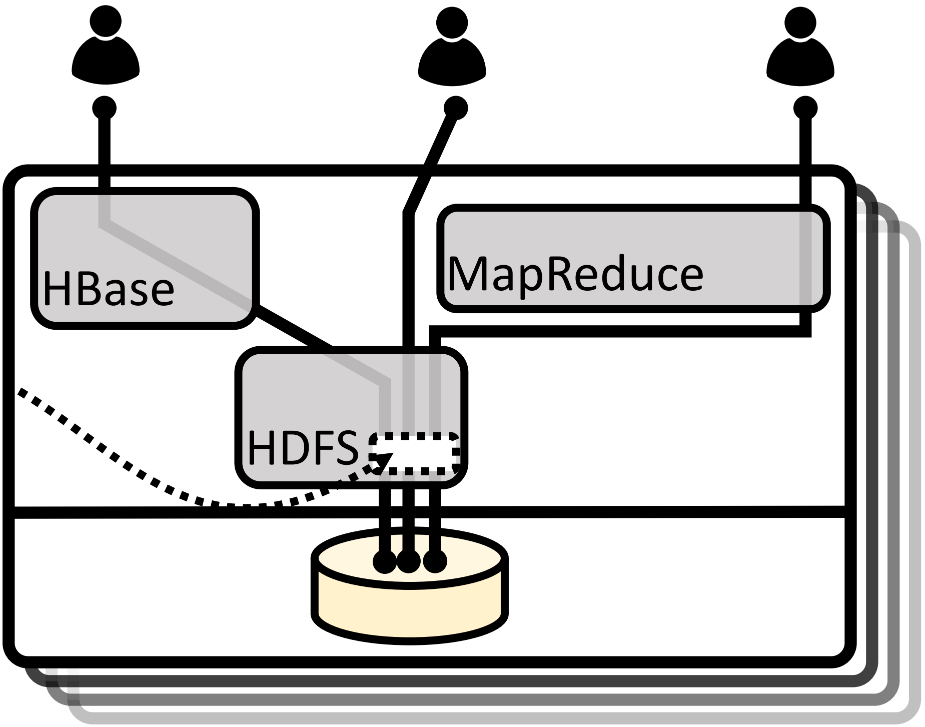


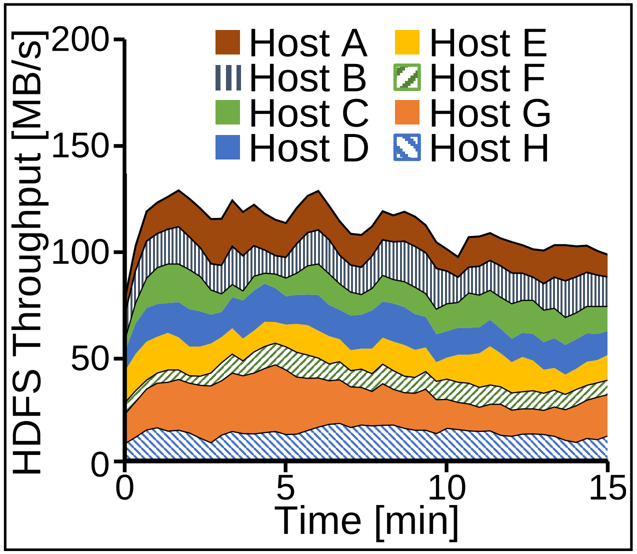
DataNodeMetrics.java

```
public class DataNodeMetrics {
  ...
  public void incrBytesRead(int delta) {
     ...
  }
  ...
}
```

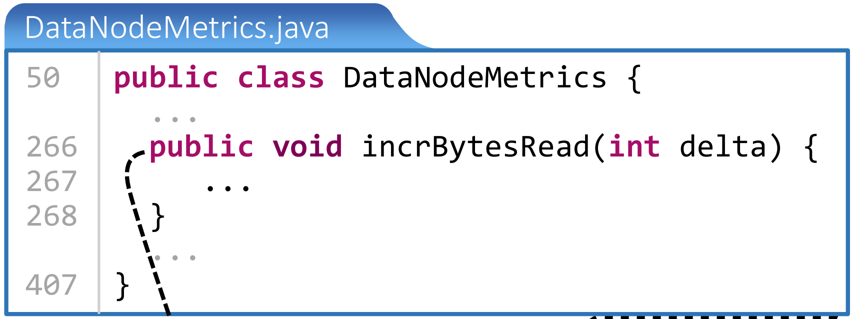
DataNode Metrics

(“DataNodeMetrics”, delta=10, host=“hop01”, …)





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

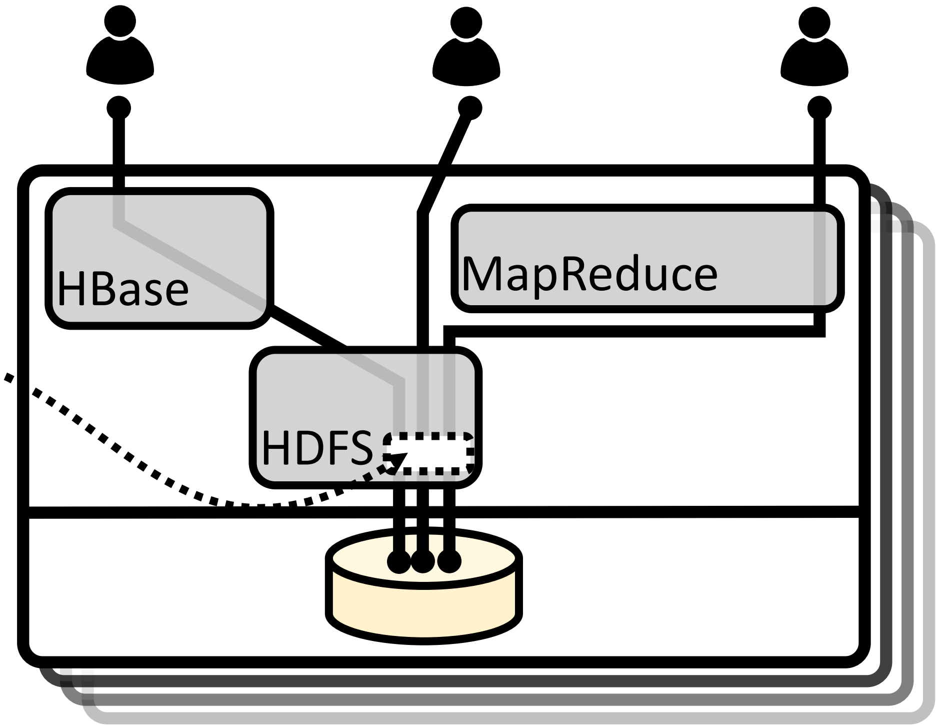


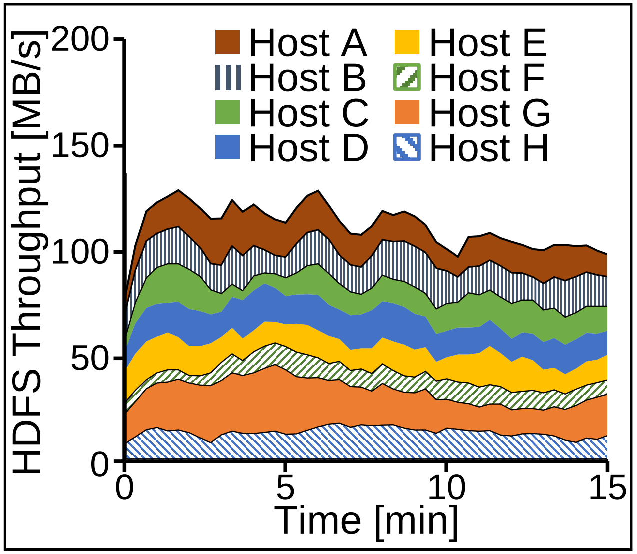
DataNodeMetrics.java

```java
public class DataNodeMetrics {
  ...
  public void incrBytesRead(int delta) {
      ...
  }
  ...
}
```

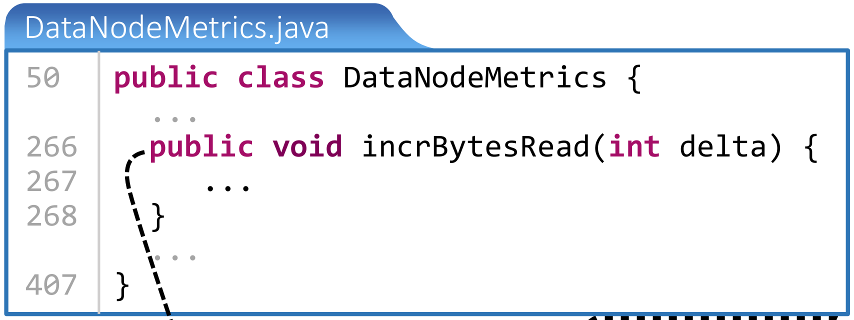
(“DataNodeMetrics”, delta=10, host=“hop01”, …)

DataNode Metrics





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

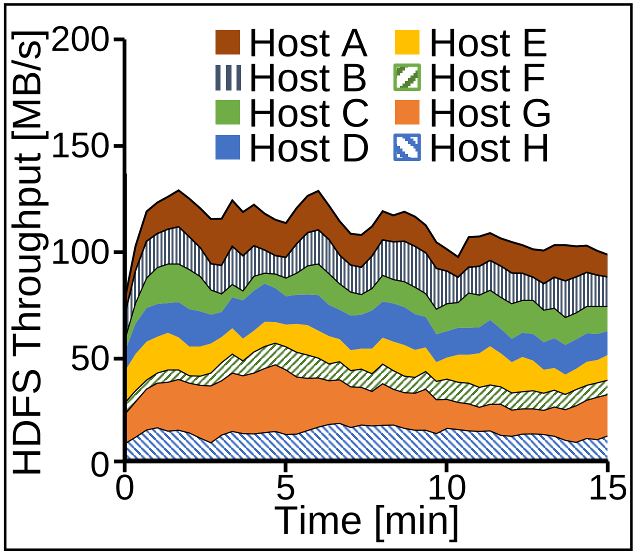


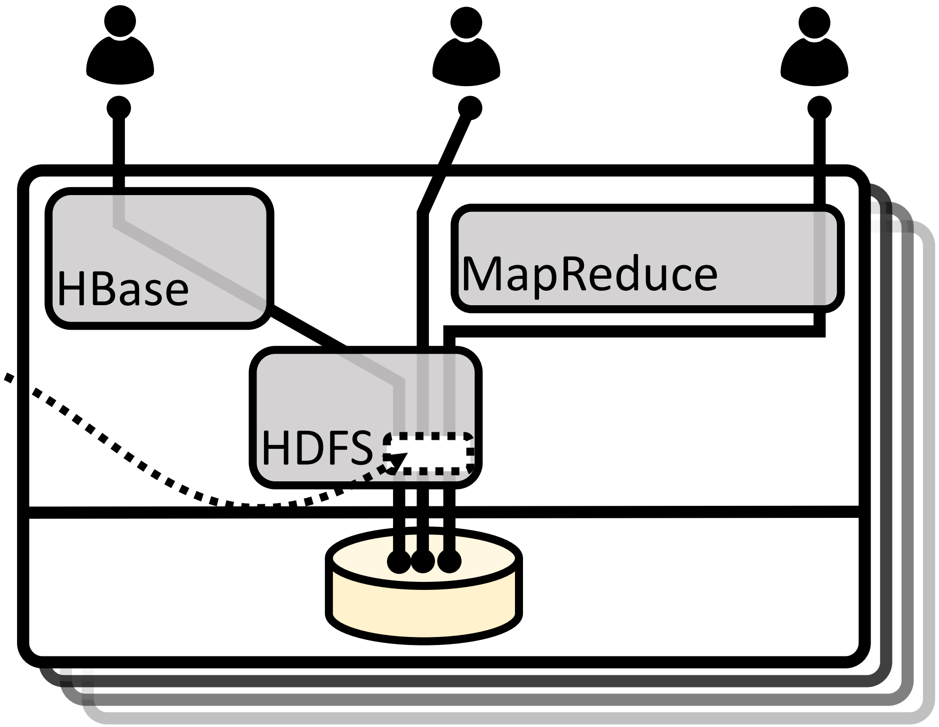
DataNodeMetrics.java

```
public class DataNodeMetrics {
  ...
  public void incrBytesRead(int delta) {
     ...
  }
  ...
}
```

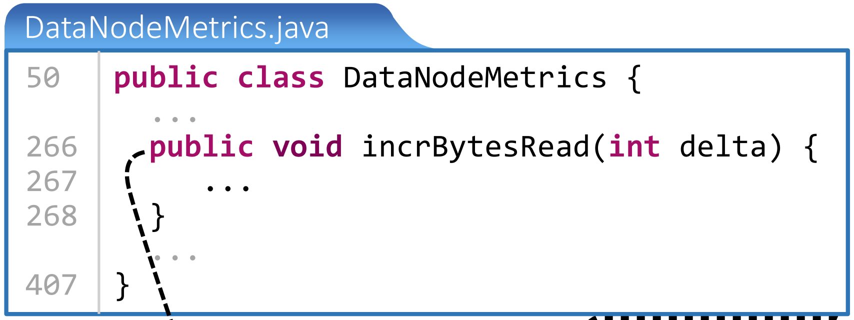
DataNode Metrics

(“DataNodeMetrics”, delta=10, host=“hop01”, …)



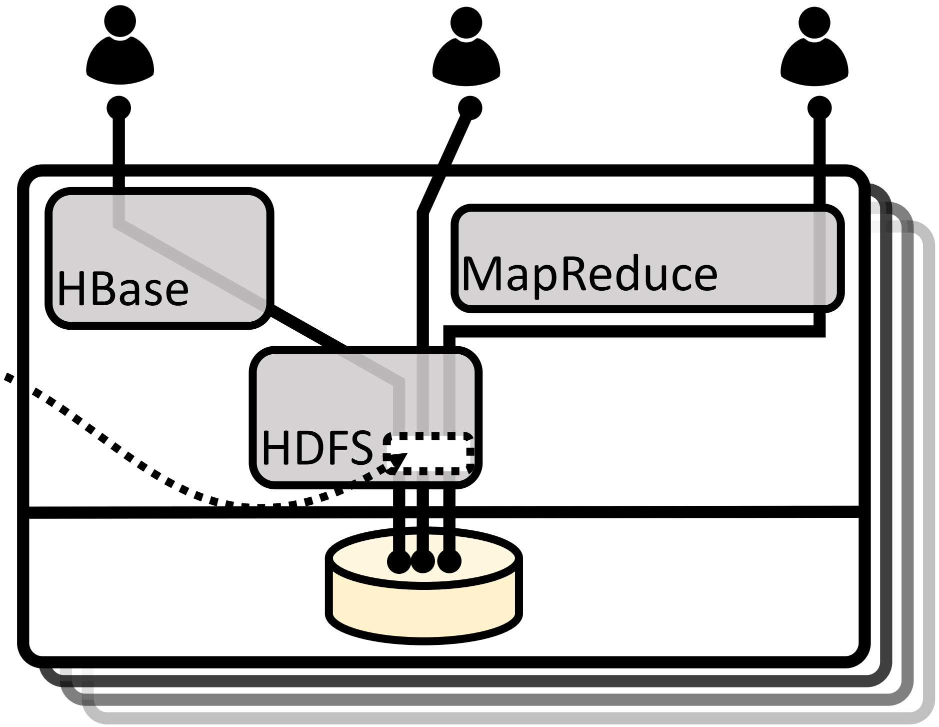


```
From incr In DataNodeMetrics.incrBytesRead
  GroupBy incr.host
  Select incr.host, SUM(incr.delta)
```

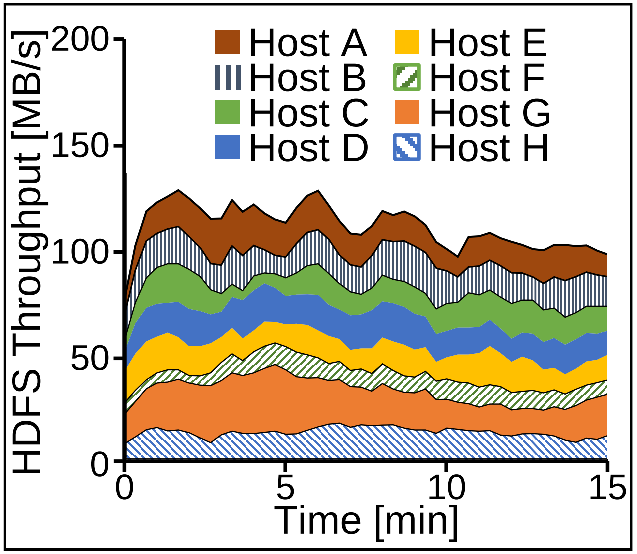


DataNodeMetrics.java

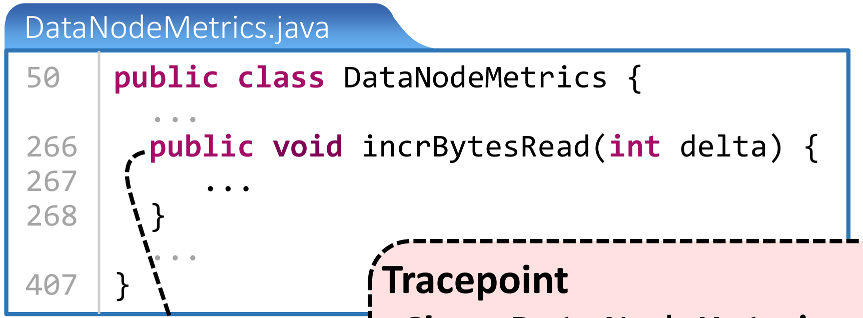
```
public class DataNodeMetrics {
  ...
  public void incrBytesRead(int delta) {
      ...
  }
  ...
}
```



(“DataNodeMetrics”, delta=10, host=“hop01”, …)



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

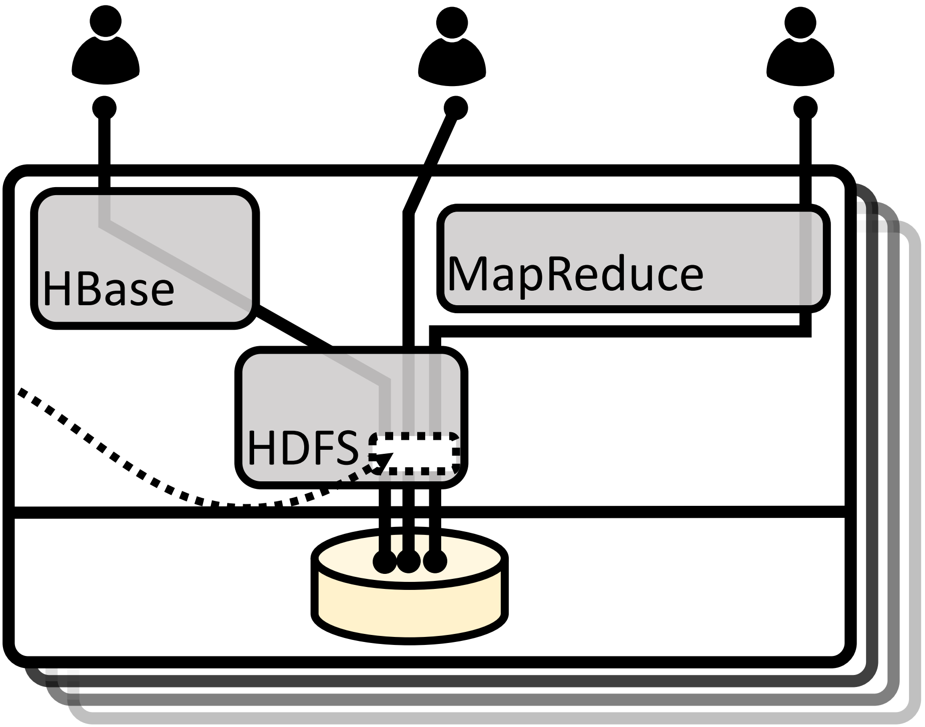


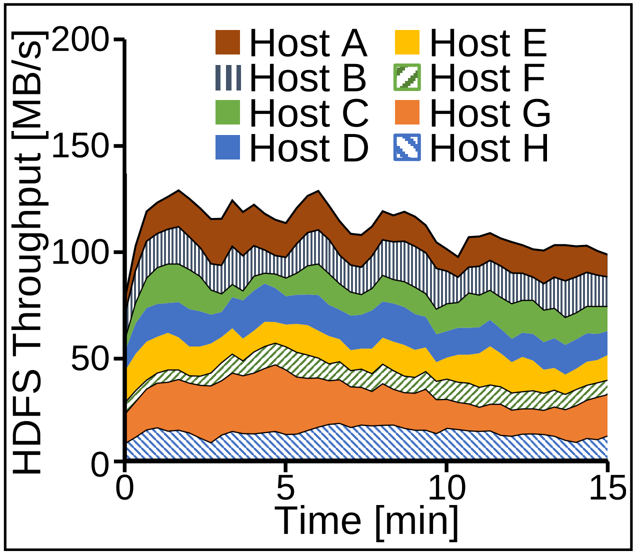
DataNodeMetrics.java

```
public class DataNodeMetrics {
  ...
  public void incrBytesRead(int delta) {
     ...
  }
  ...
}
```

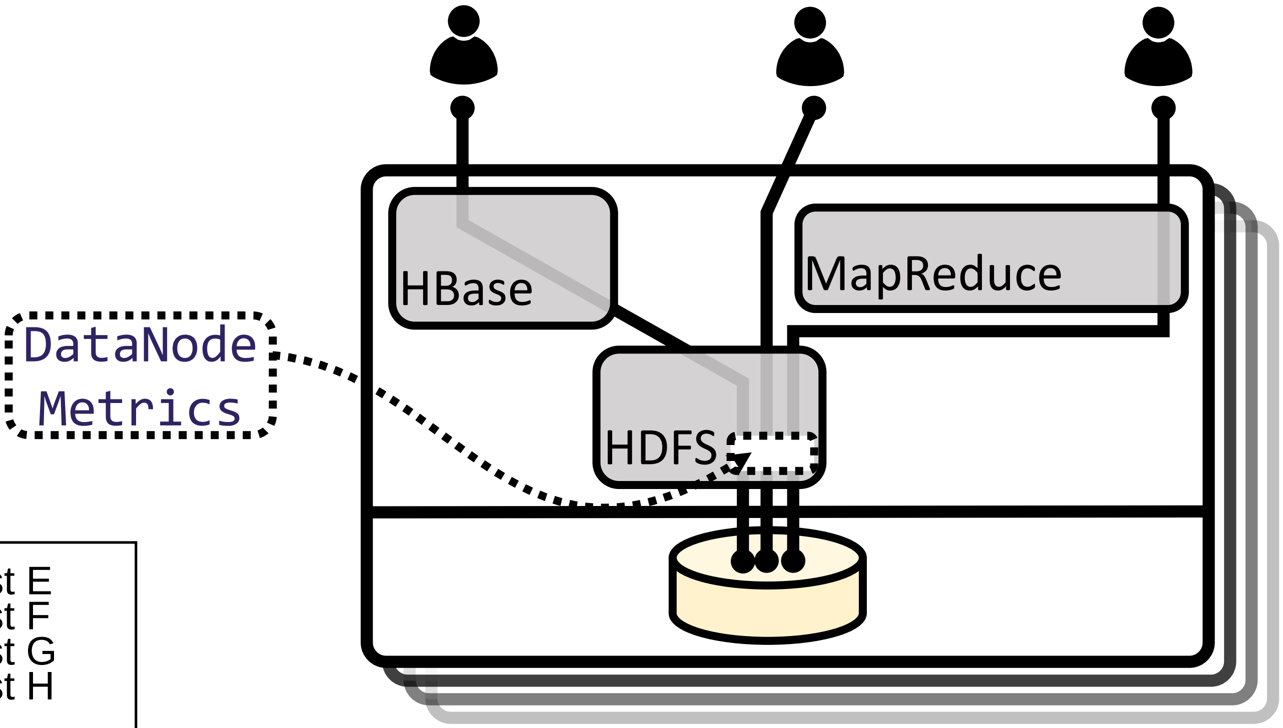
Tracepoint
Class: DataNodeMetrics
Method: incrBytesRead
Exports: "delta"=delta

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





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

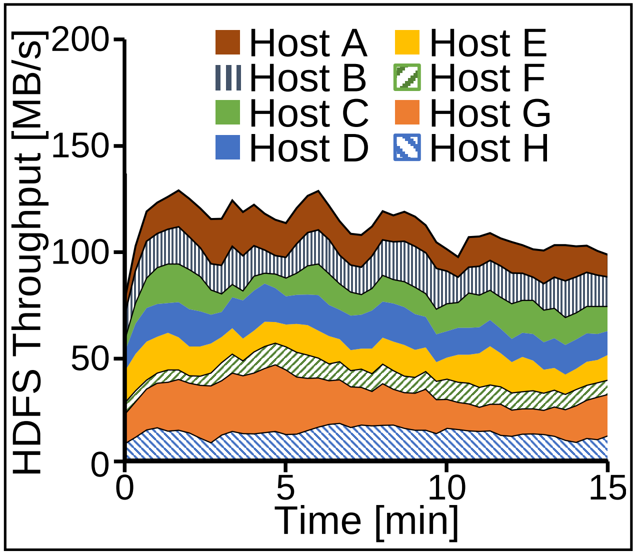


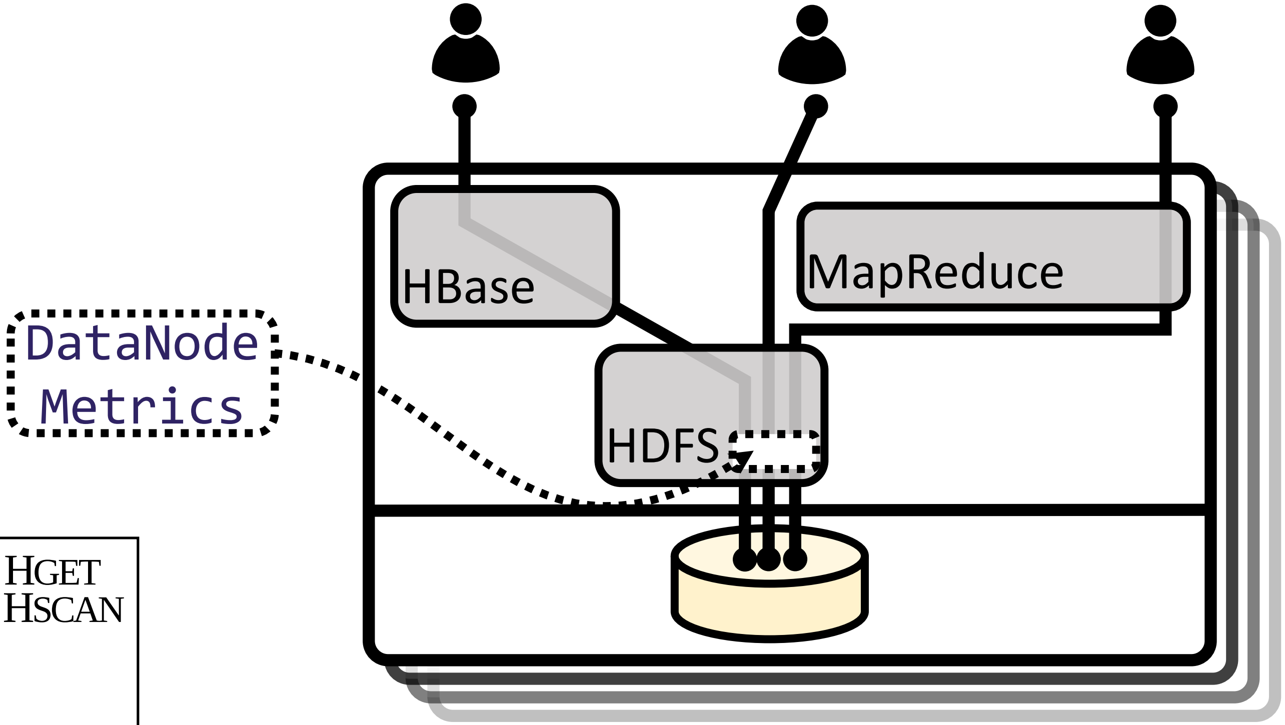
DataNode Metrics

HBase

MapReduce

HDFS



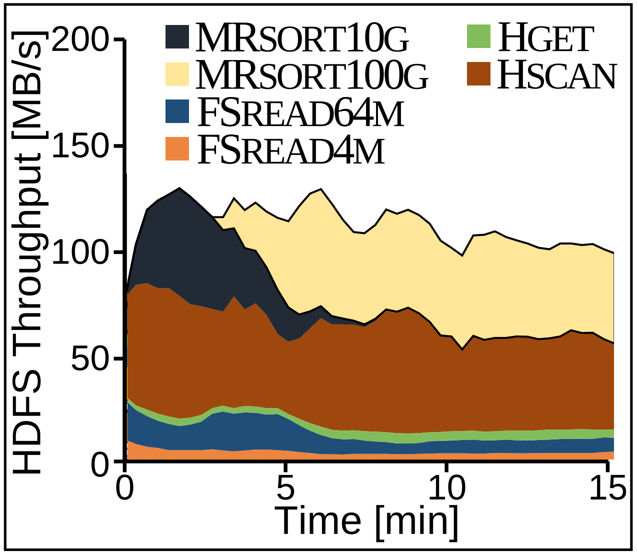


DataNode Metrics

HBase

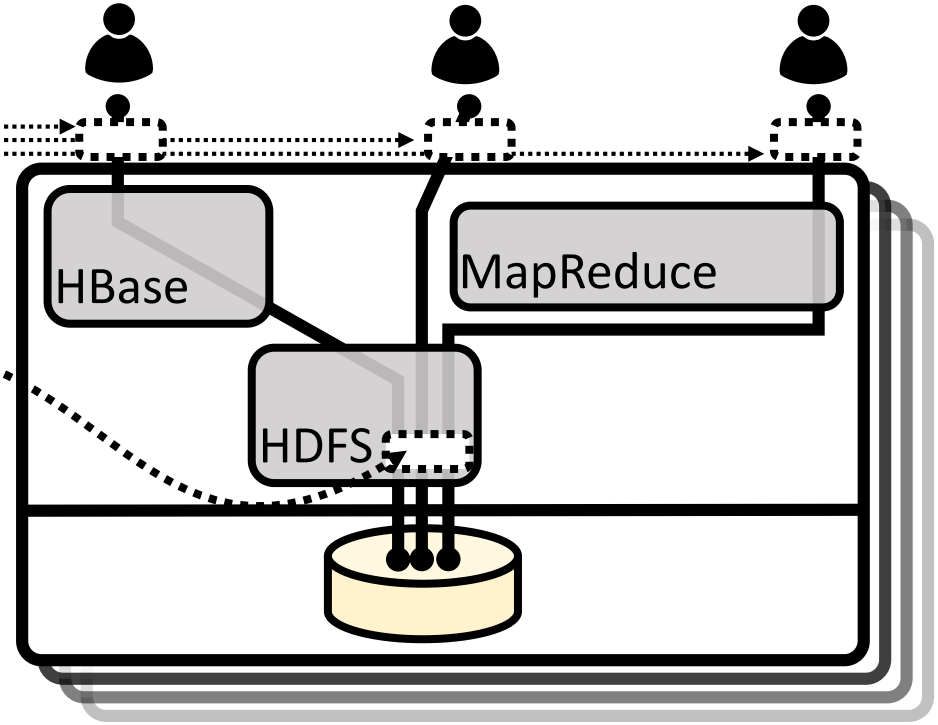
MapReduce

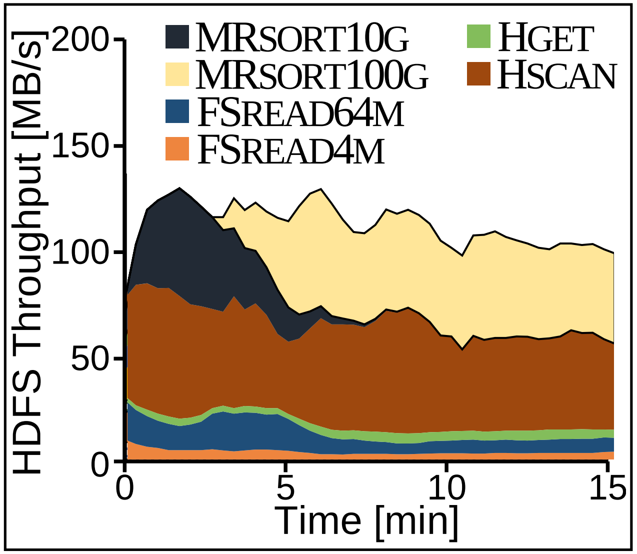
HDFS



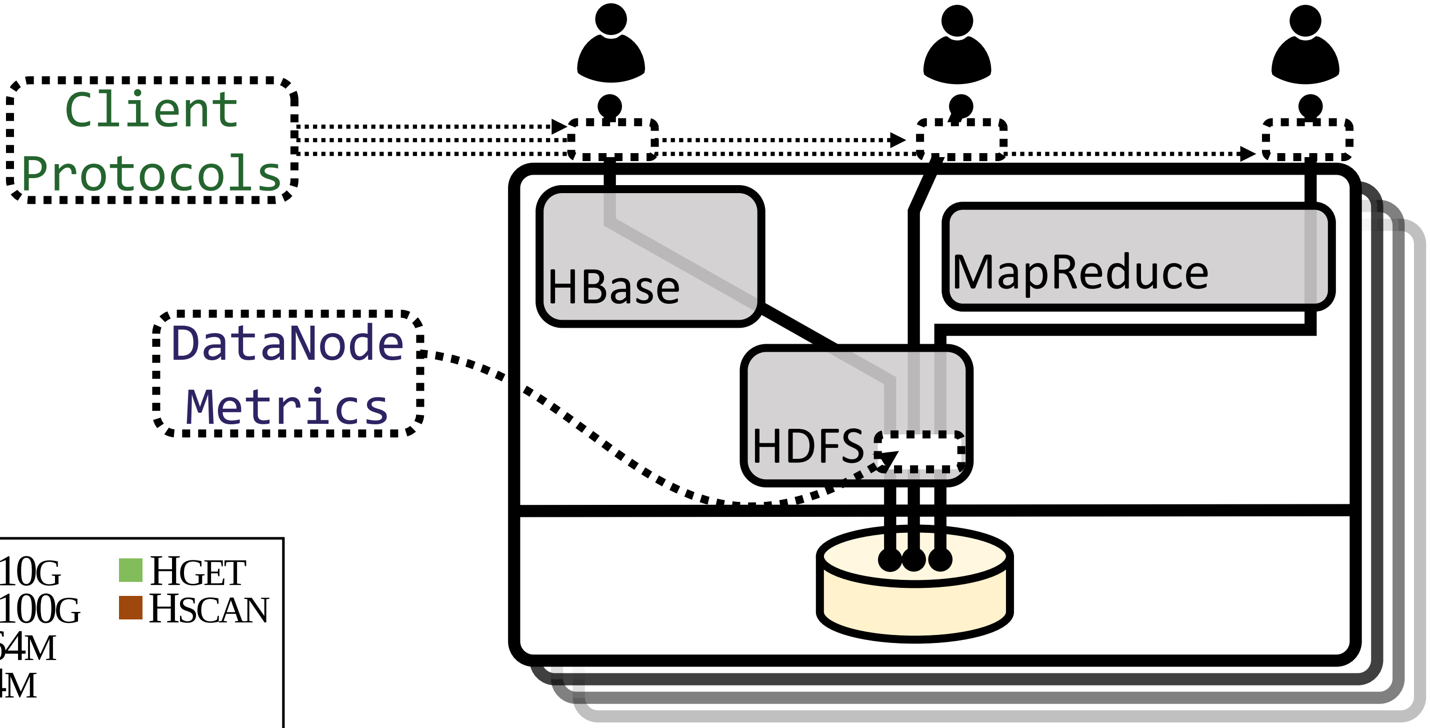
Client Protocols

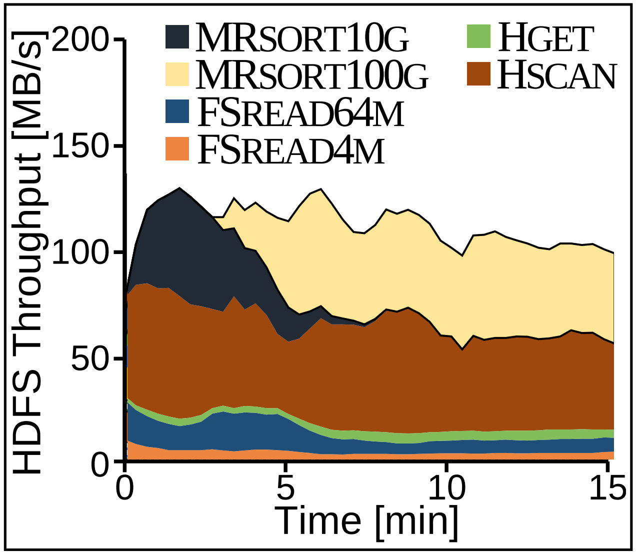
DataNode Metrics



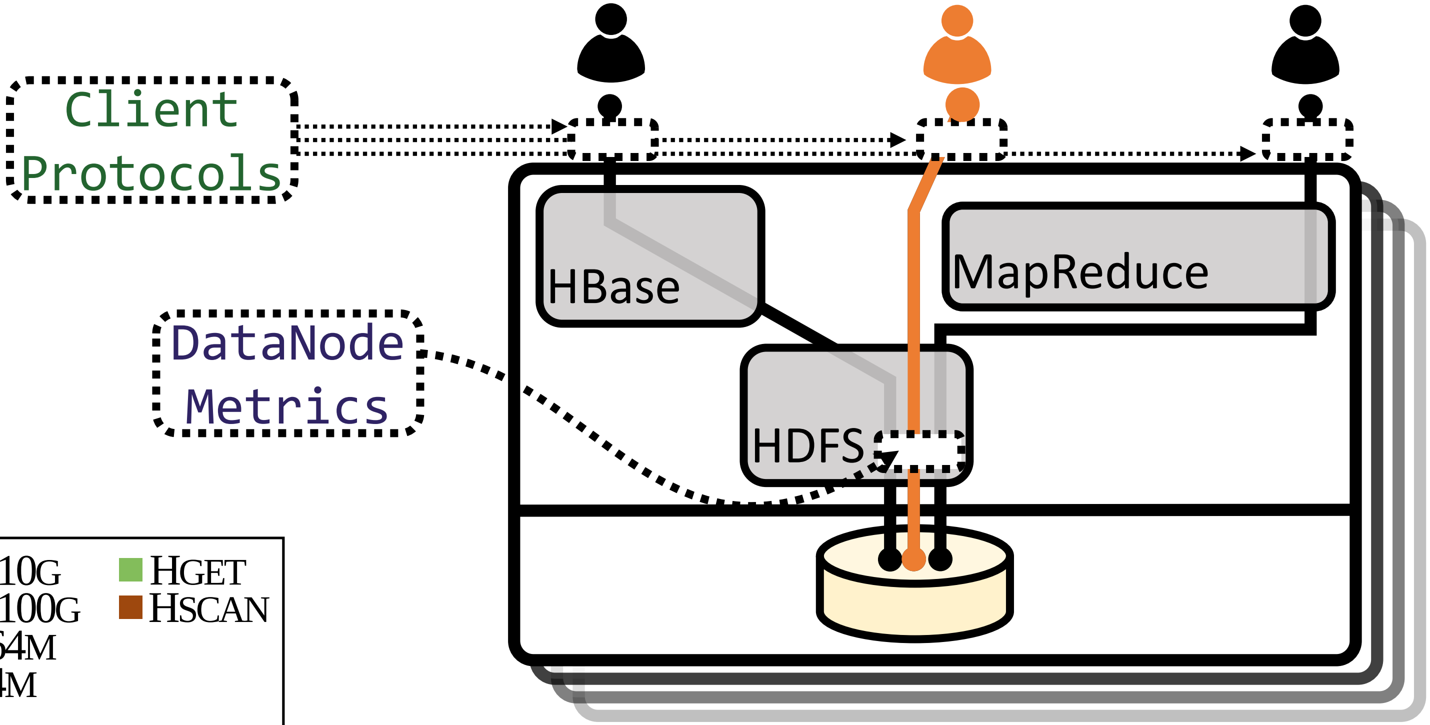


Happened-before Join ($\overrightarrow{\bowtie}$)



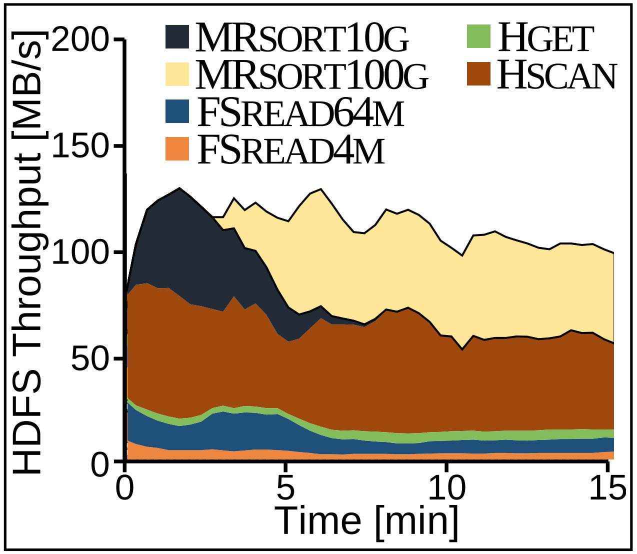


Happened-before Join ($\overrightarrow{\bowtie}$)

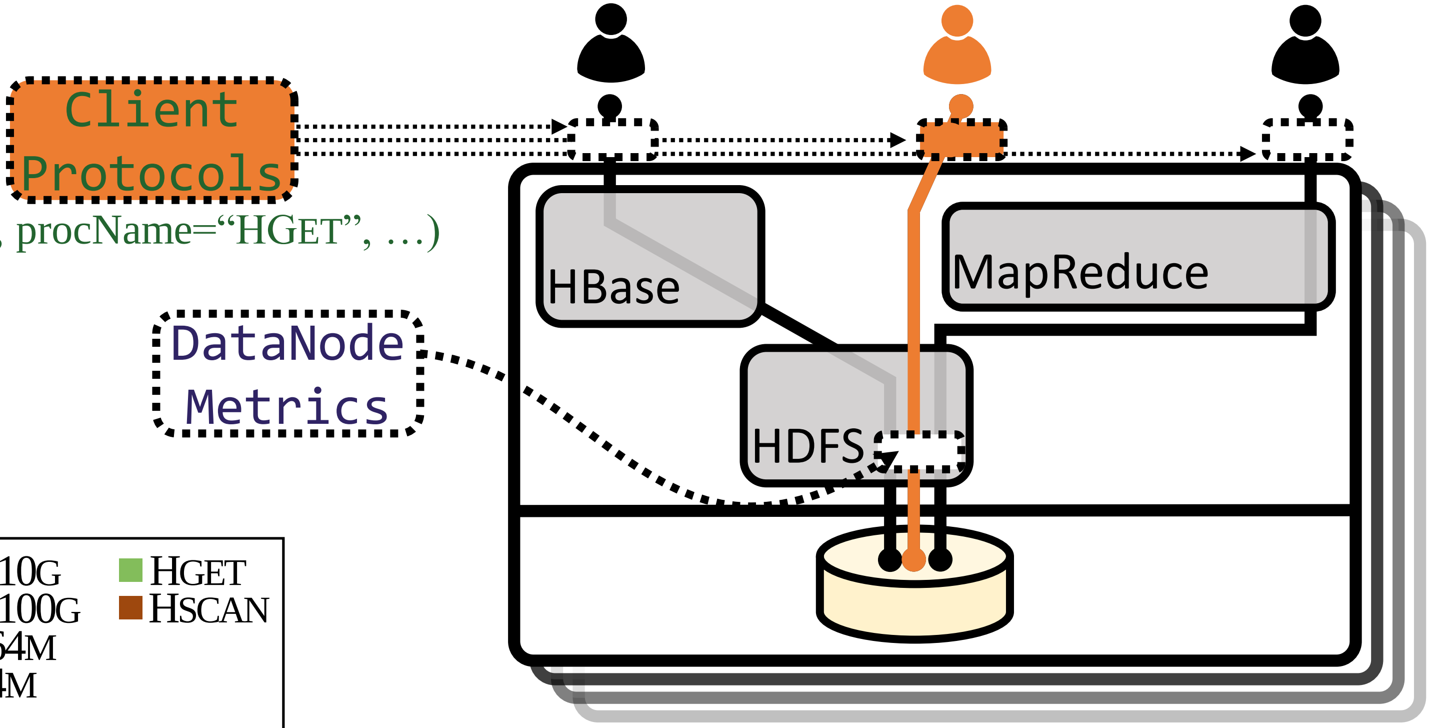


Client Protocols

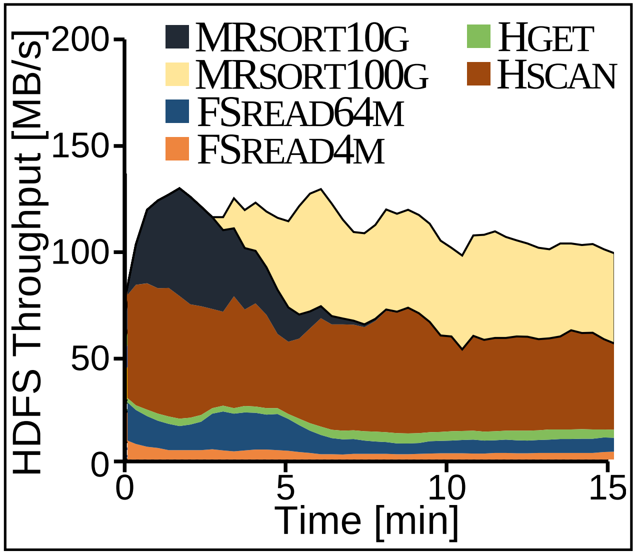
DataNode Metrics



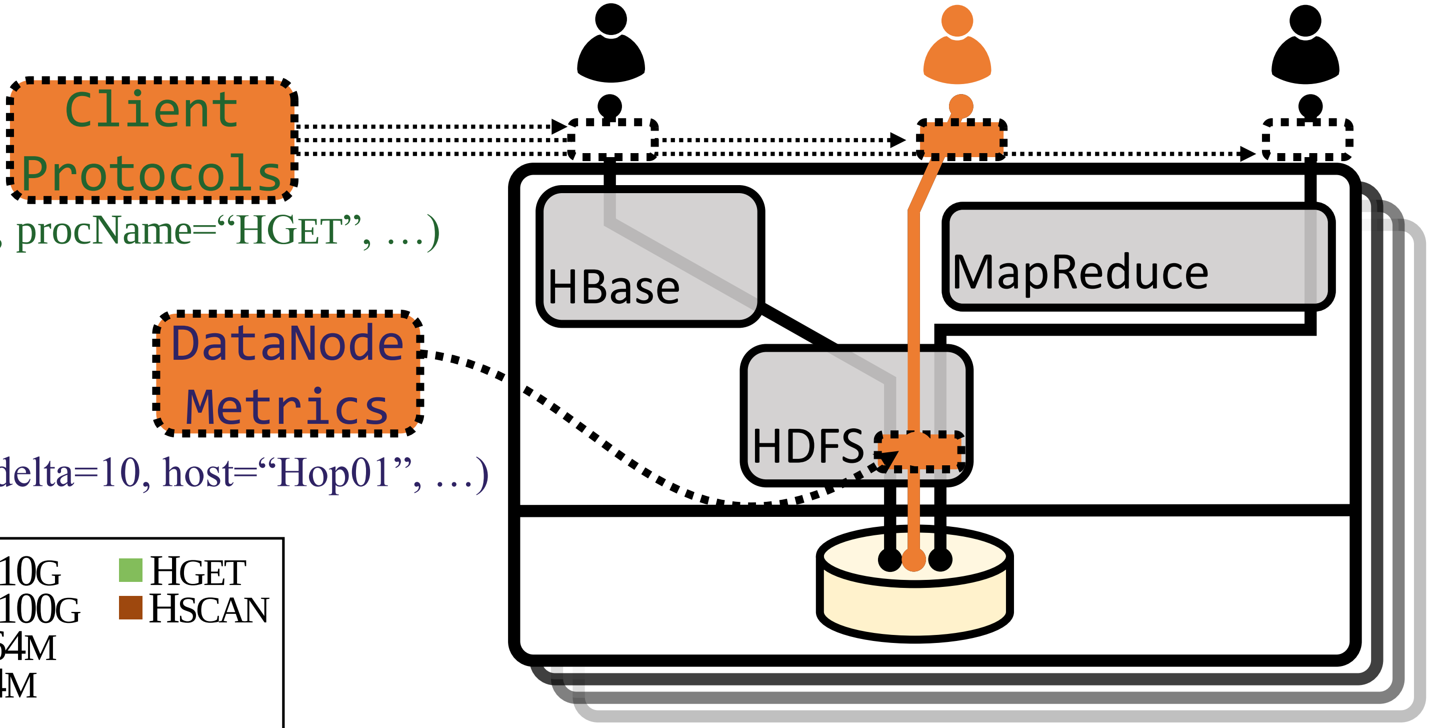
Happened-before Join ($\overrightarrow{\bowtie}$)



(“ClientProtocols”, procName=“HGET”, …)

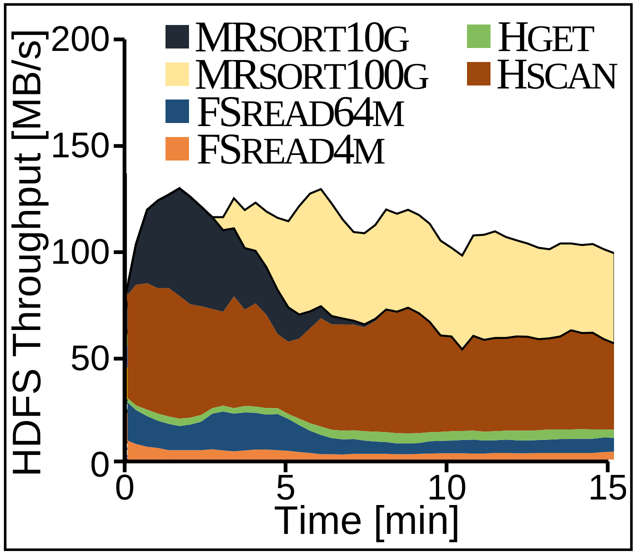


Happened-before Join ($\overrightarrow{\bowtie}$)



(“ClientProtocols”, procName=“HGET”, …)

(“DataNodeMetrics”, delta=10, host=“Hop01”, …)



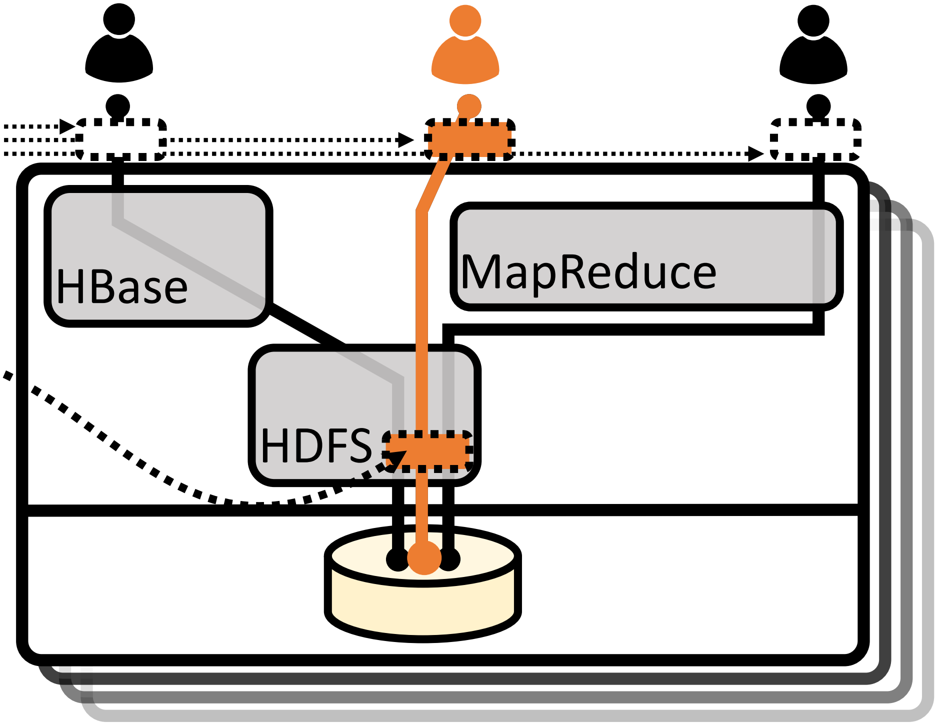
Happened-before Join ($\overrightarrow{\bowtie}$)

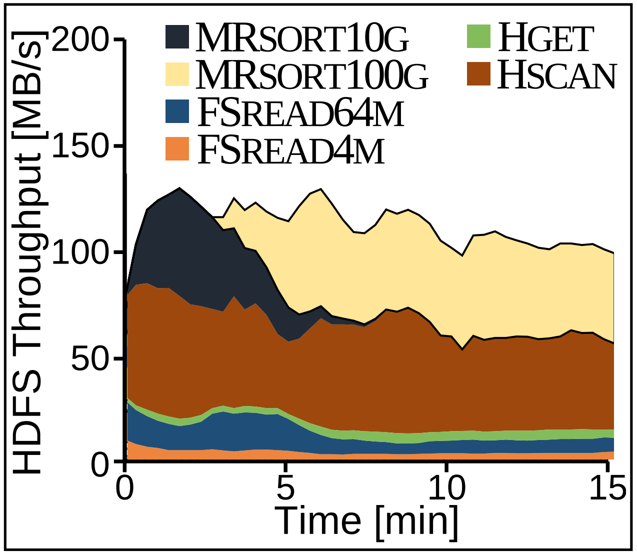
Client Protocols

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

DataNode Metrics

("DataNodeMetrics", delta=10, host="Hop01", …)





Client Protocols $\overrightarrow{\bowtie}$ DataNode Metrics

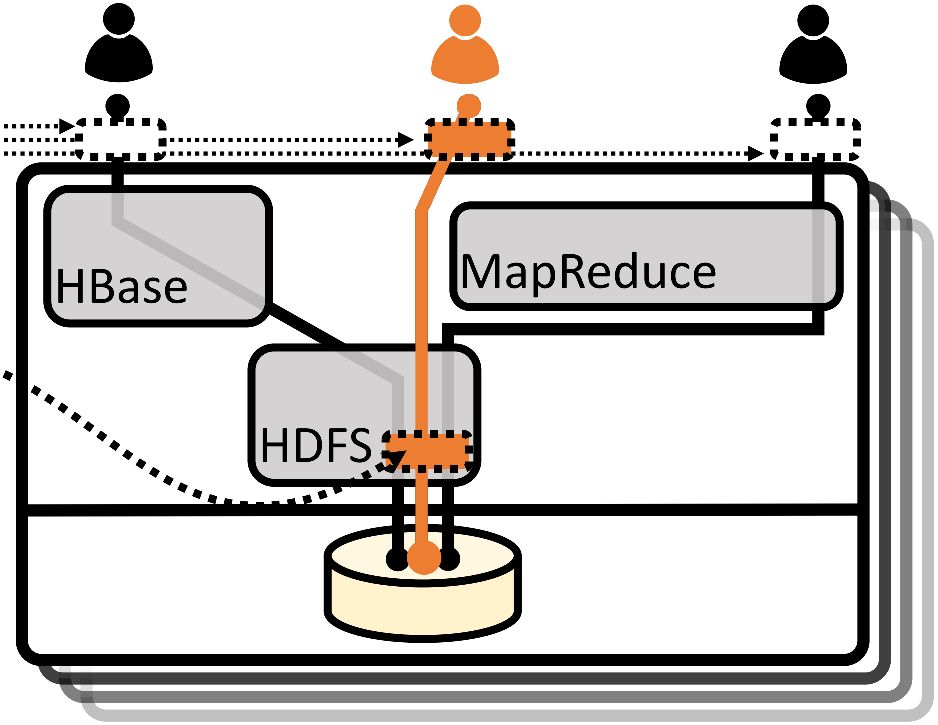
Happened-before Join ($\overrightarrow{\bowtie}$)

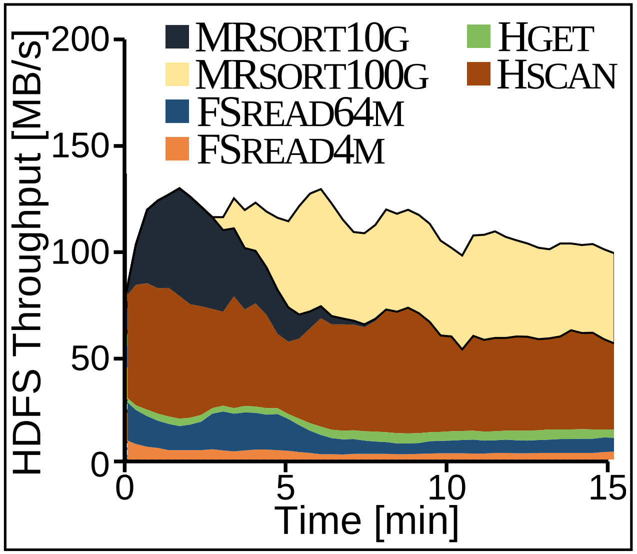
Client Protocols

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

DataNode Metrics

("DataNodeMetrics", delta=10, host="Hop01", …)





Client Protocols $\overrightarrow{\bowtie}$ DataNode Metrics

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

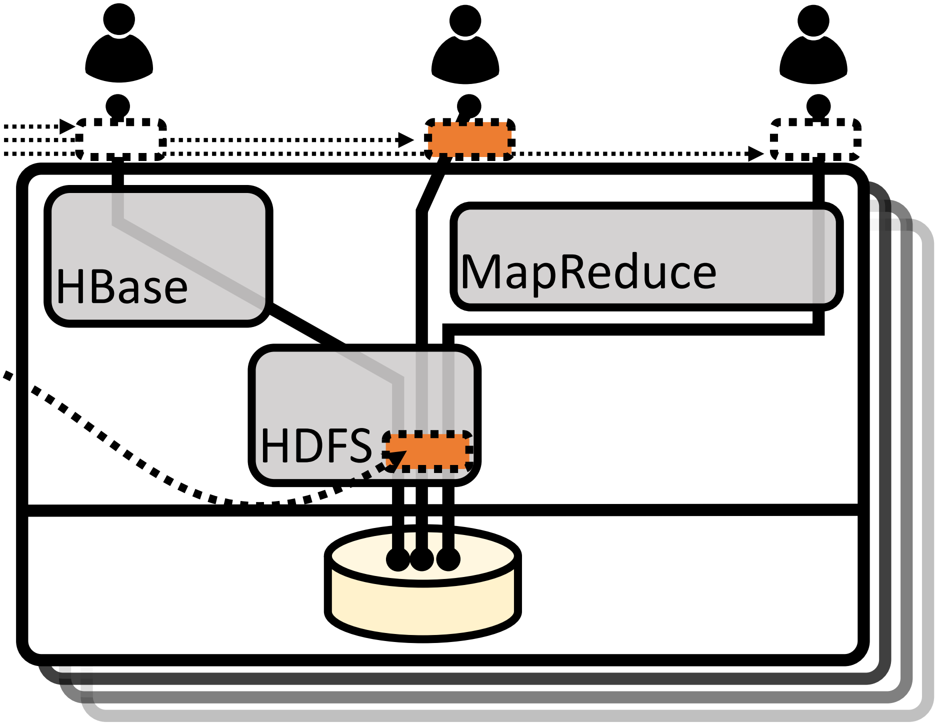
Happened-before Join ($\overrightarrow{\bowtie}$)

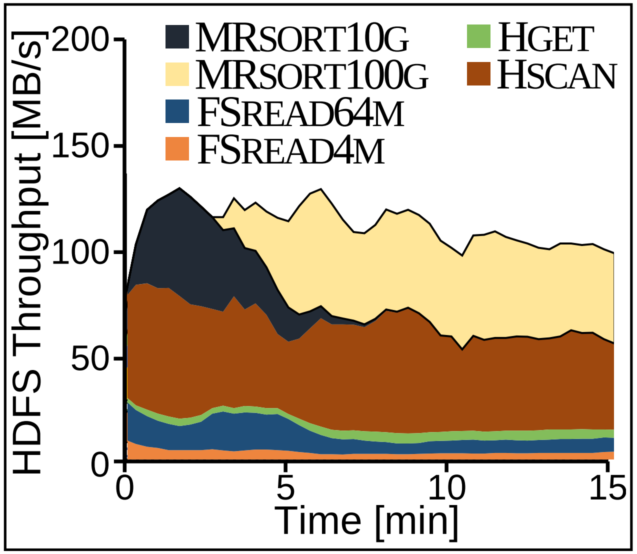
Client Protocols

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

DataNode Metrics

("DataNodeMetrics", delta=10, host="Hop01", …)





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

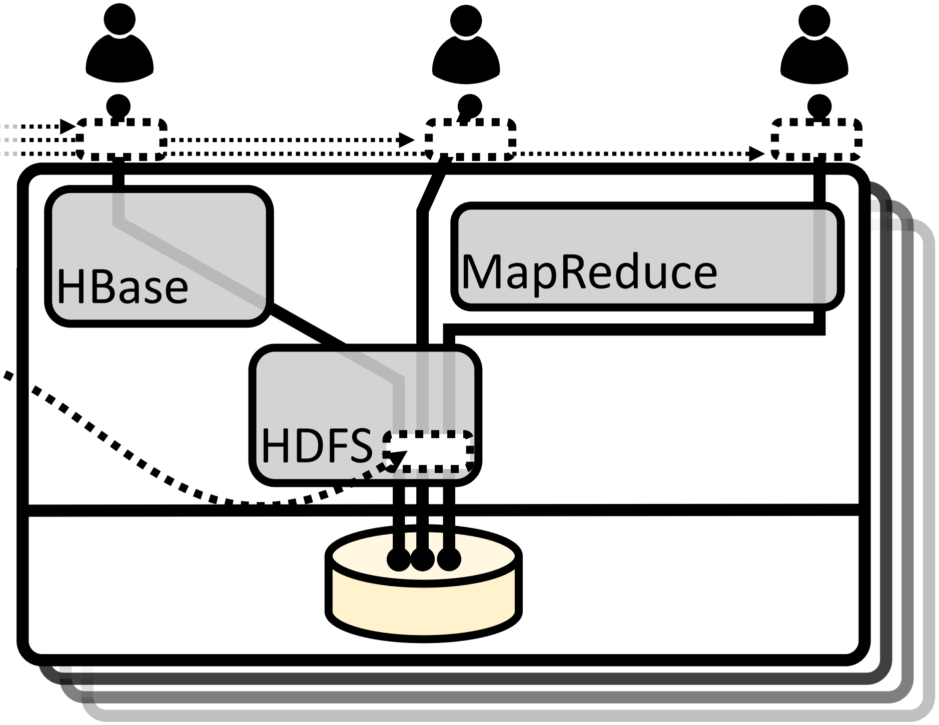
Happened-before Join ($\overrightarrow{\bowtie}$)

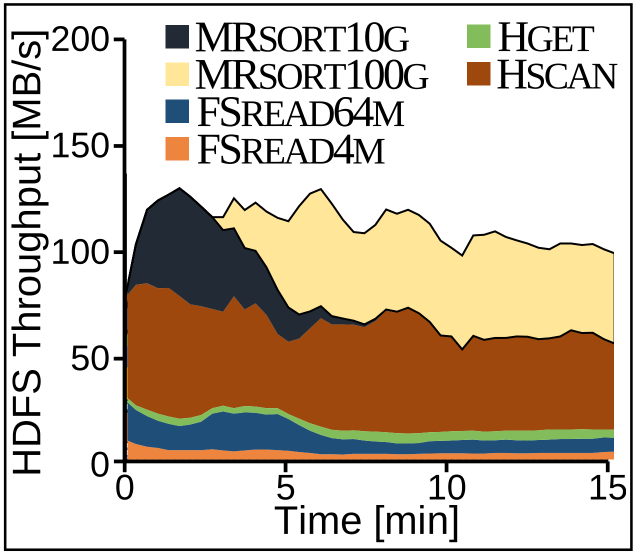
Client Protocols

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

DataNode Metrics

("DataNodeMetrics", delta=10, host="Hop01", …)





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

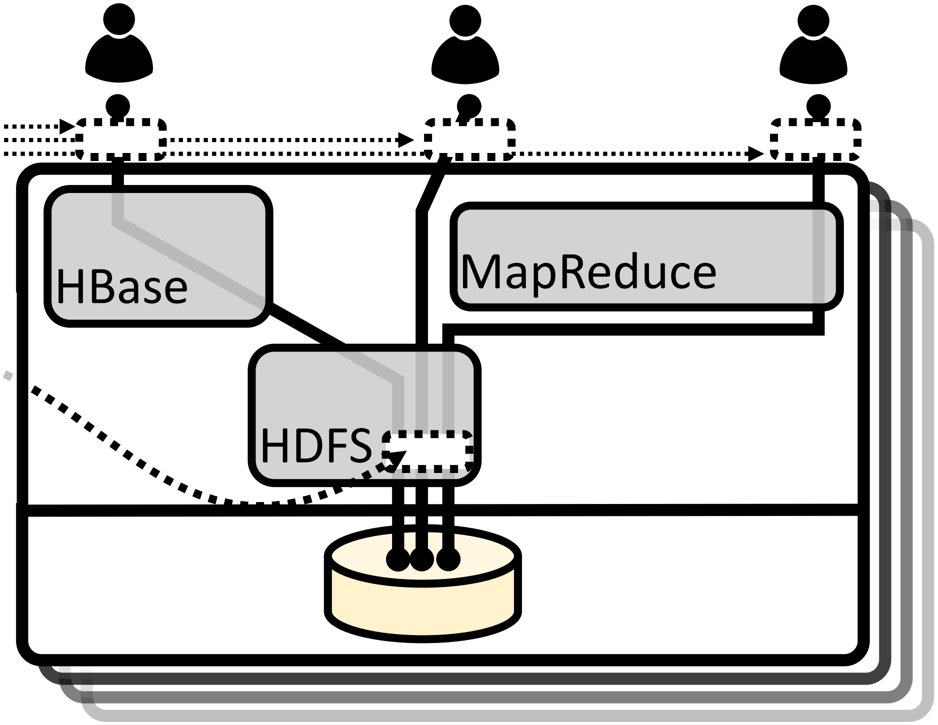
Happened-before Join ($\overrightarrow{\bowtie}$)

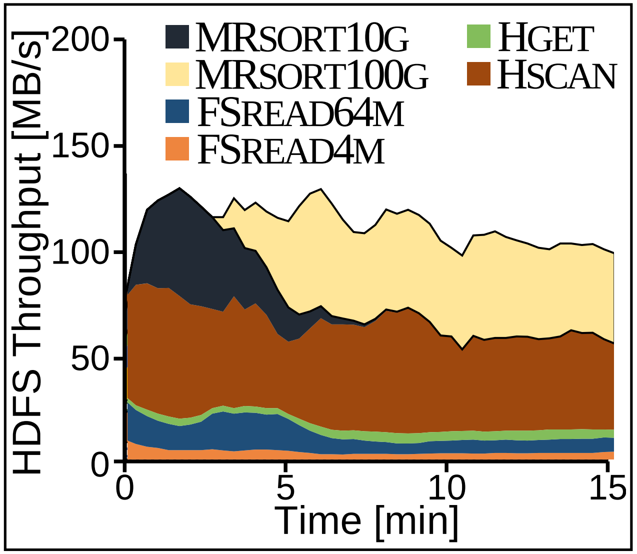
Client Protocols

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

DataNode Metrics

("DataNodeMetrics", delta=10, host="Hop01", …)





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

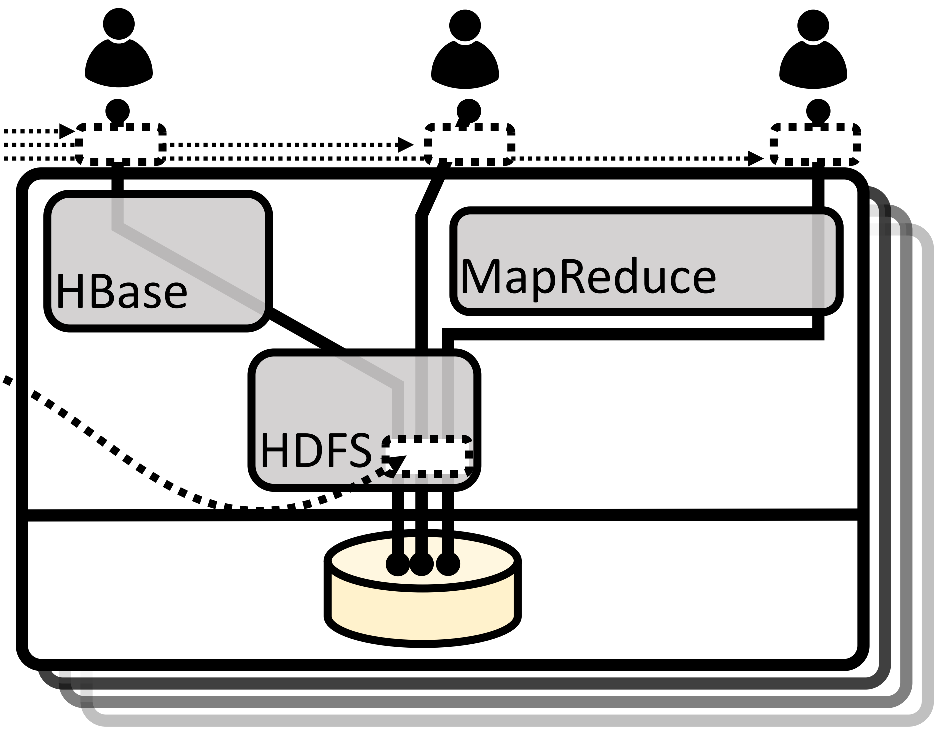
Happened-before Join ($\overrightarrow{\bowtie}$)

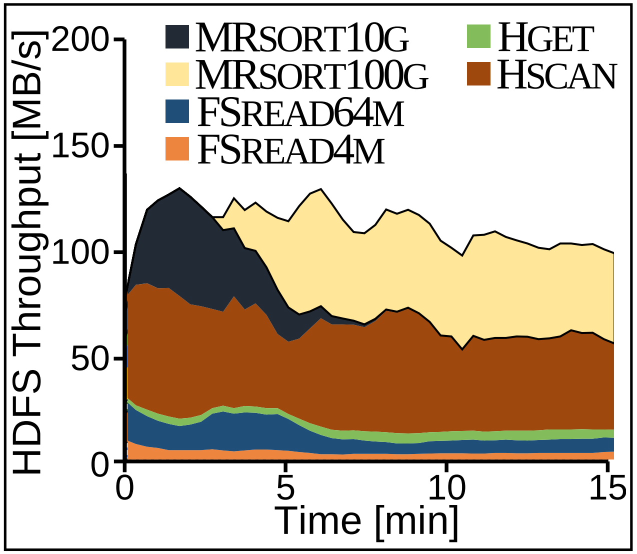
Client Protocols

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

DataNode Metrics

("DataNodeMetrics", delta=10, host="Hop01", …)





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

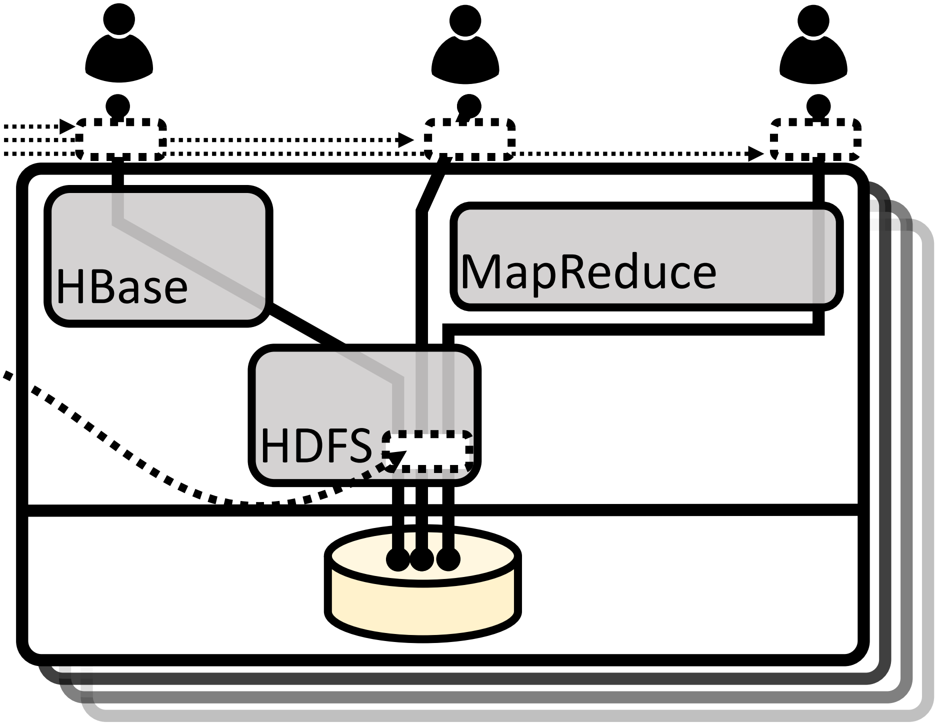
Happened-before Join ($\overrightarrow{\bowtie}$)

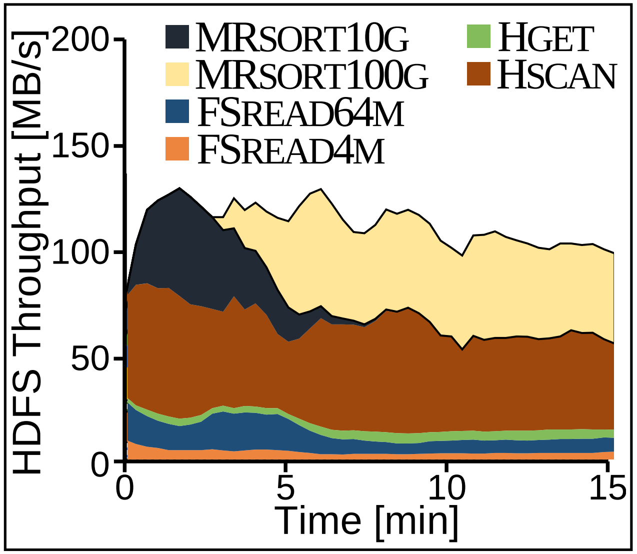
Client Protocols

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

DataNode Metrics

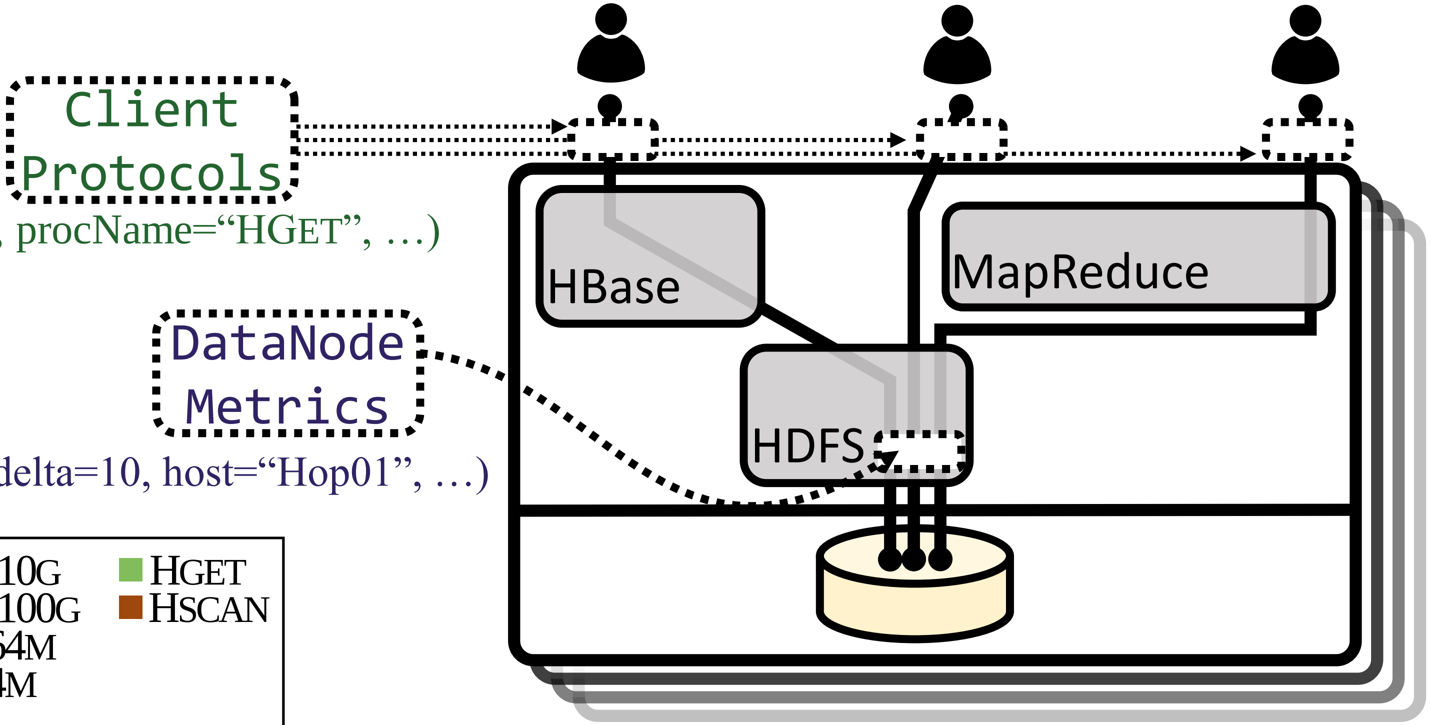
("DataNodeMetrics", delta=10, host="Hop01", …)





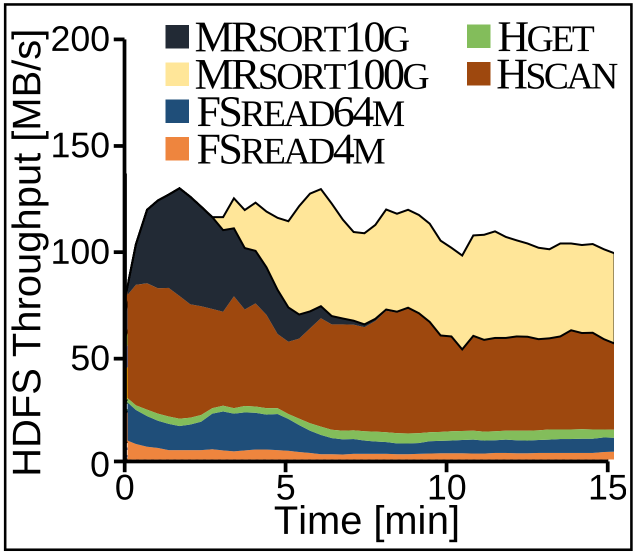
```
From incr In DataNodeMetrics.incrBytesRead
  Join client In First(ClientProtocols) On client -> incr
  GroupBy client.procName
  Select client.procName, SUM(incr.delta)
```

Happened-before Join ($\overrightarrow{\bowtie}$)



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

("DataNodeMetrics", delta=10, host="Hop01", …)



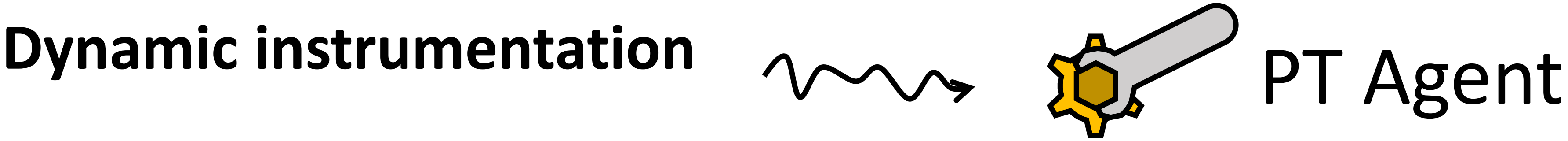
```
From incr In DataNodeMetrics.incrBytesRead
  Join client In First(ClientProtocols) On client -> incr
  GroupBy client.procName
  Select client.procName, SUM(incr.delta)
```

(procName="HGET", delta=10, …)

Design & Implementation
Pivot Tracing Pre-requisites

Design & Implementation

Pivot Tracing Pre-requisites



Dynamic instrumentation

PT Agent

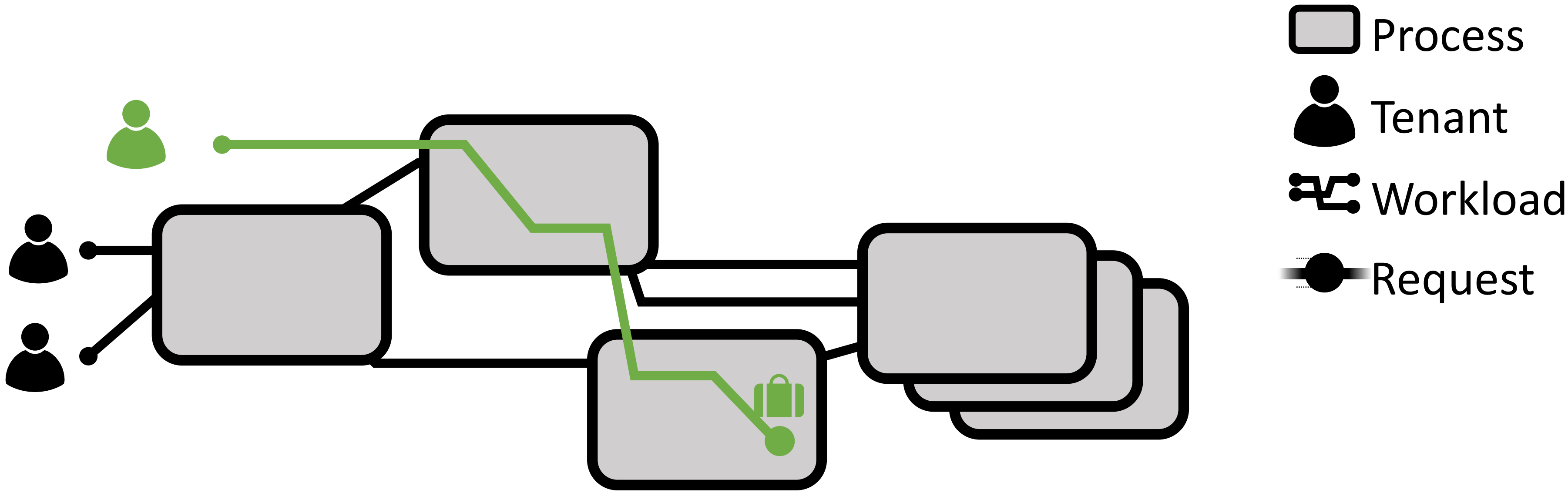
Design & Implementation

Pivot Tracing Pre-requisites

Dynamic instrumentation ⇝ PT Agent

Causal tracing ⇝ Baggage

Causal tracing ⇝ Baggage



Process

Tenant

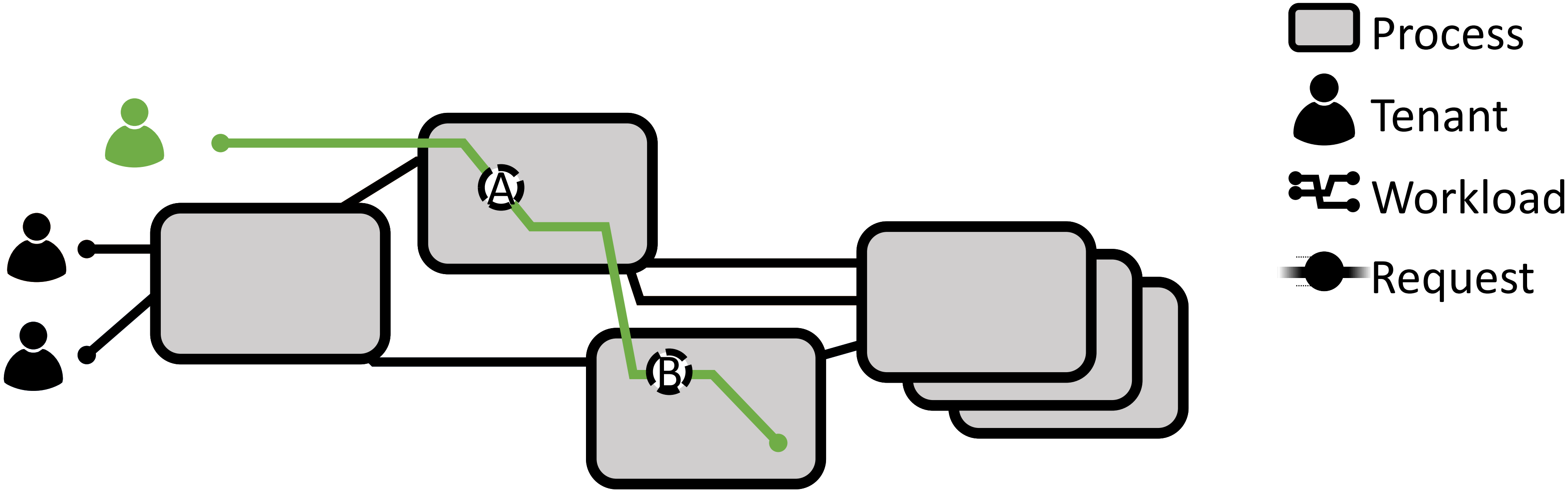
Workload

Request

Baggage is a Key:Value container propagated alongside a request

- Generalization of metadata in end-to-end tracing
- One instance per request

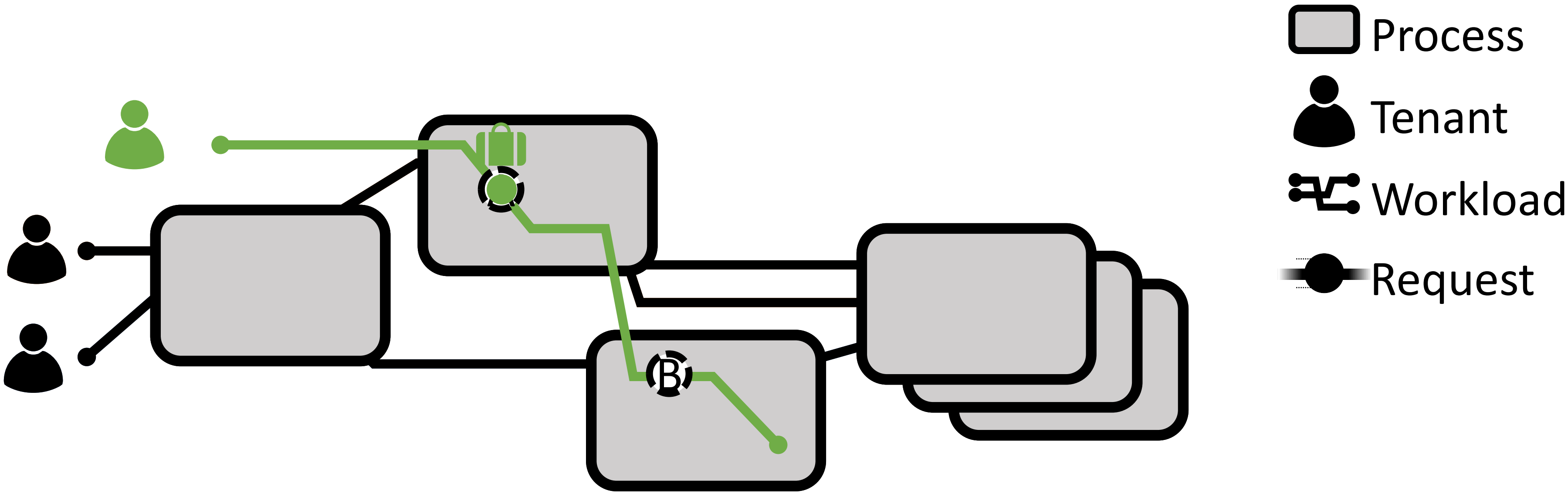
Causal tracing → Baggage



Baggage is a Key:Value container propagated alongside a request

- Generalization of metadata in end-to-end tracing
- One instance per request

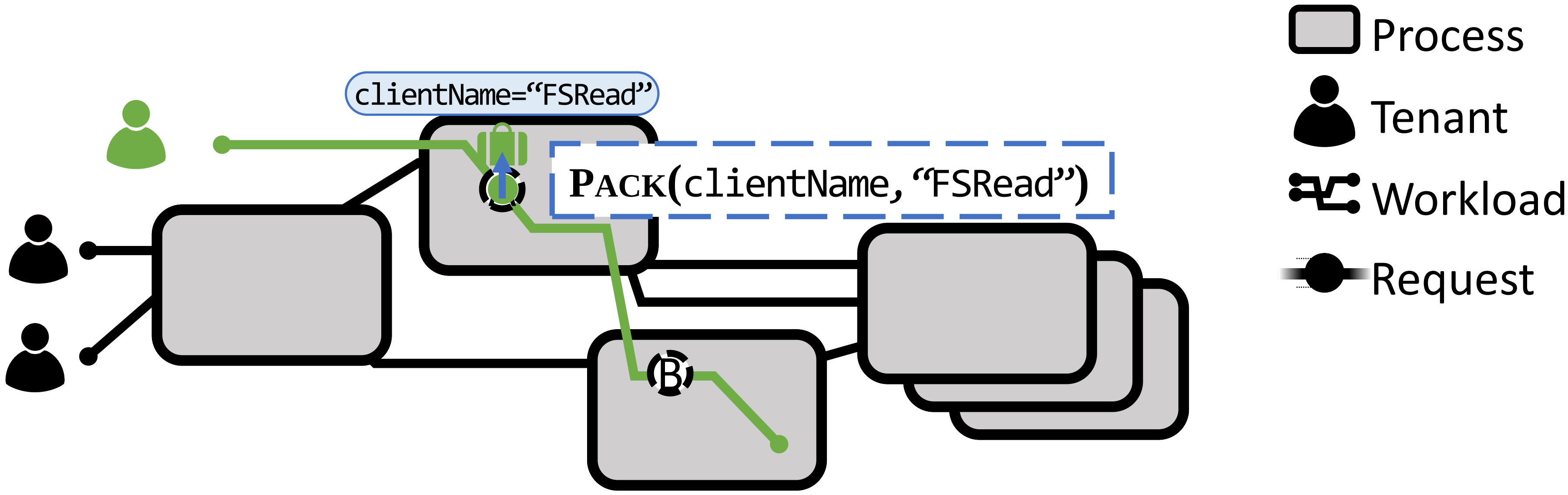
Causal tracing → Baggage



Baggage is a Key:Value container propagated alongside a request

- Generalization of metadata in end-to-end tracing
- One instance per request

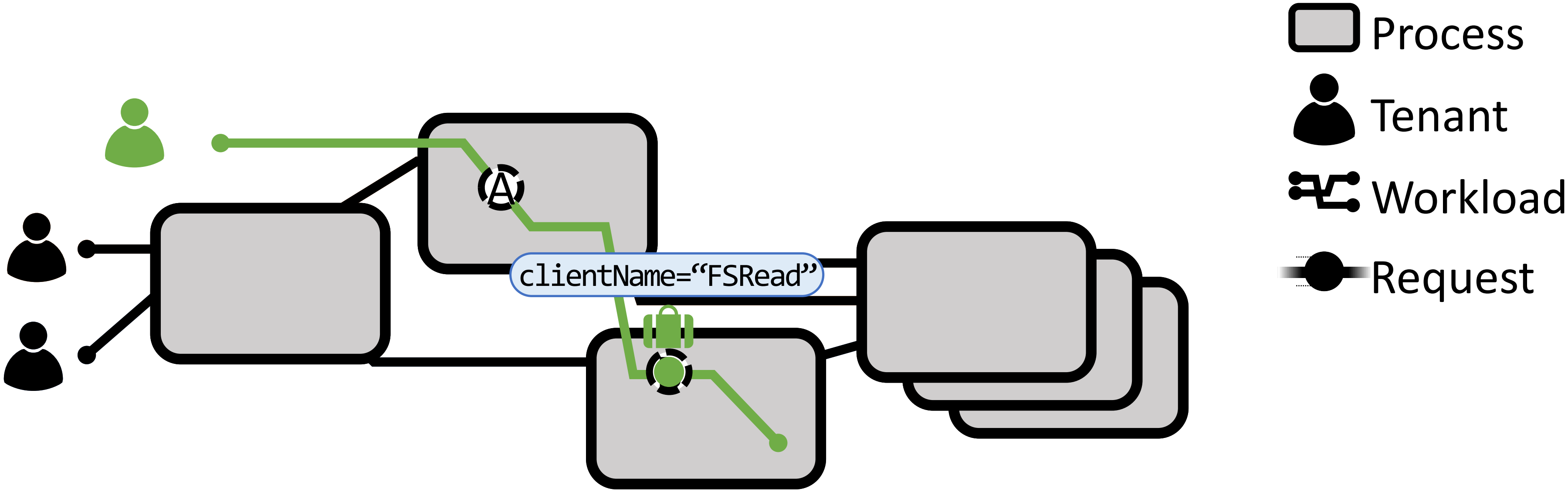
Causal tracing ⇝ Baggage



Baggage is a Key:Value container propagated alongside a request

- Generalization of metadata in end-to-end tracing
- One instance per request

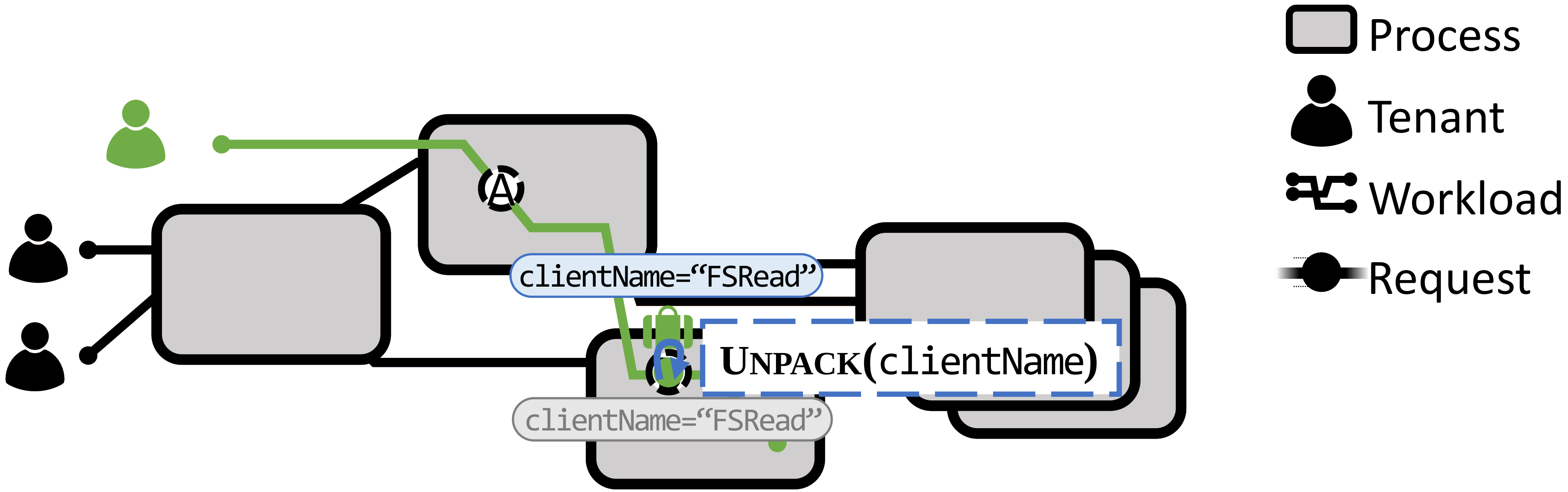
Causal tracing ⟿ Baggage



Baggage is a Key:Value container propagated alongside a request

- Generalization of metadata in end-to-end tracing
- One instance per request

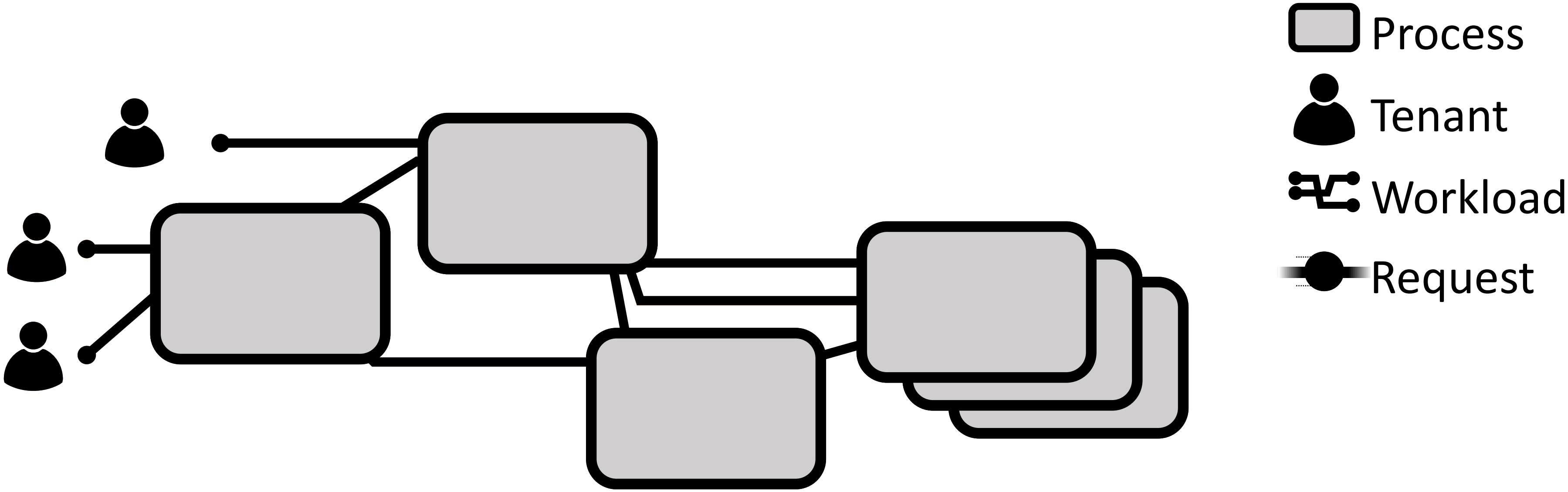
Causal tracing ⇝ Baggage



Baggage is a Key:Value container propagated alongside a request

- Generalization of metadata in end-to-end tracing
- One instance per request

Causal tracing ⟿ Baggage



Process

Tenant

Workload

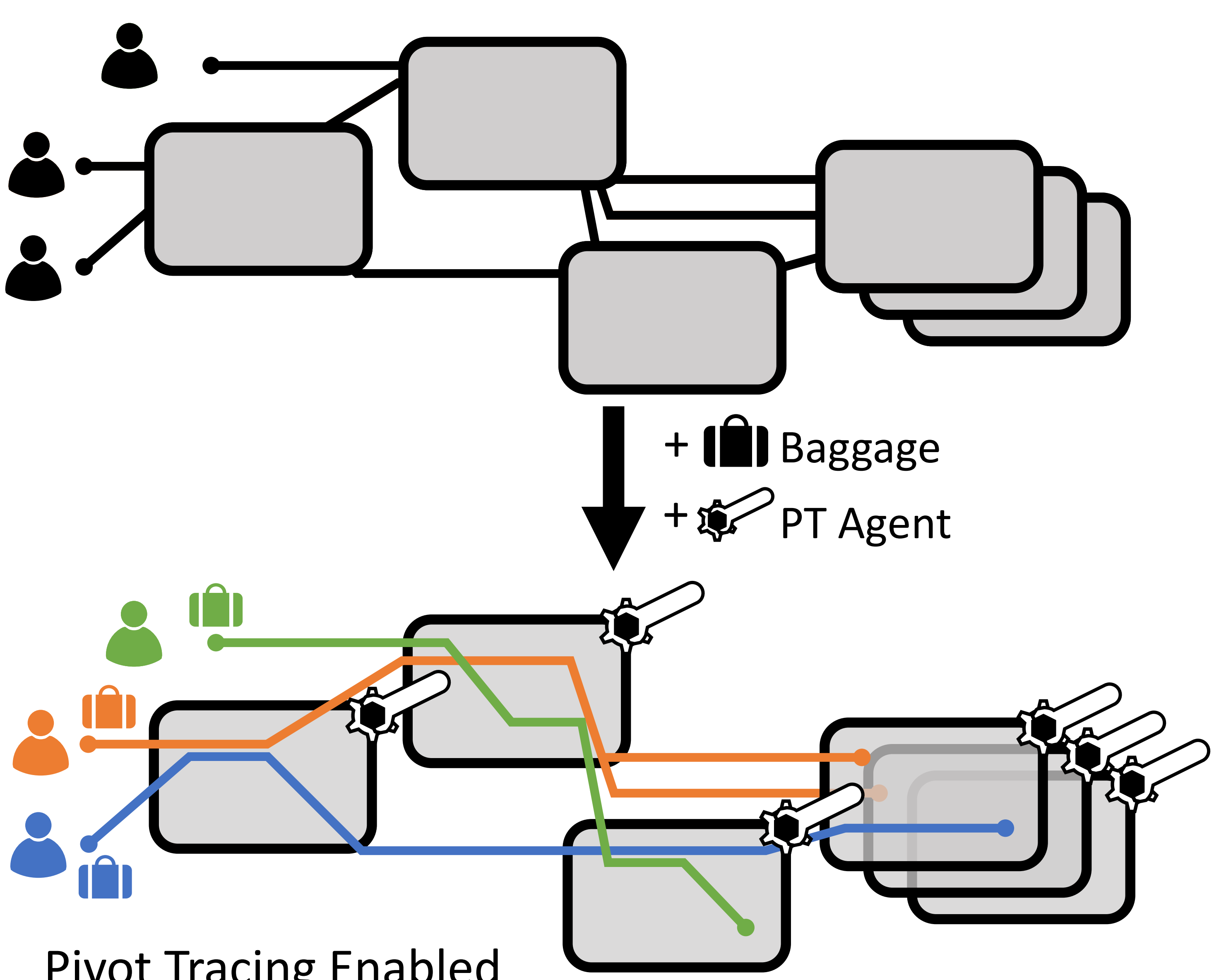
Request

Process

Tenant

Workload

Request



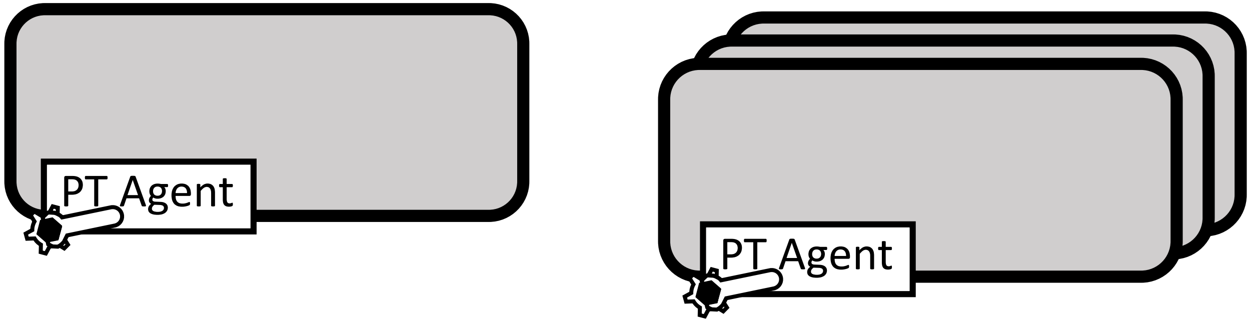
+ Baggage

+ PT Agent

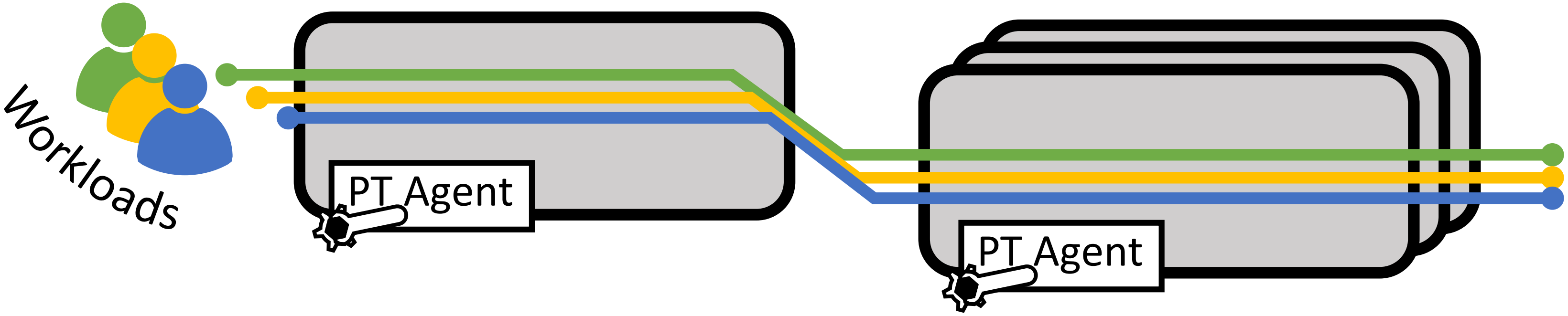
Pivot Tracing Enabled

Design & Implementation
Queries

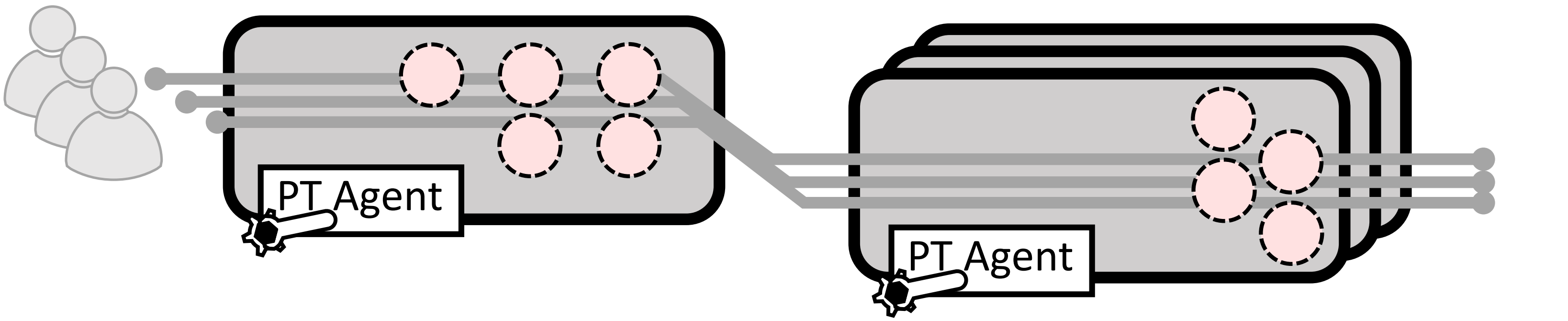
Instrumented System (+🧳 Baggage, +🔧 PT Agent)



Instrumented System (+ Baggage, + PT Agent)



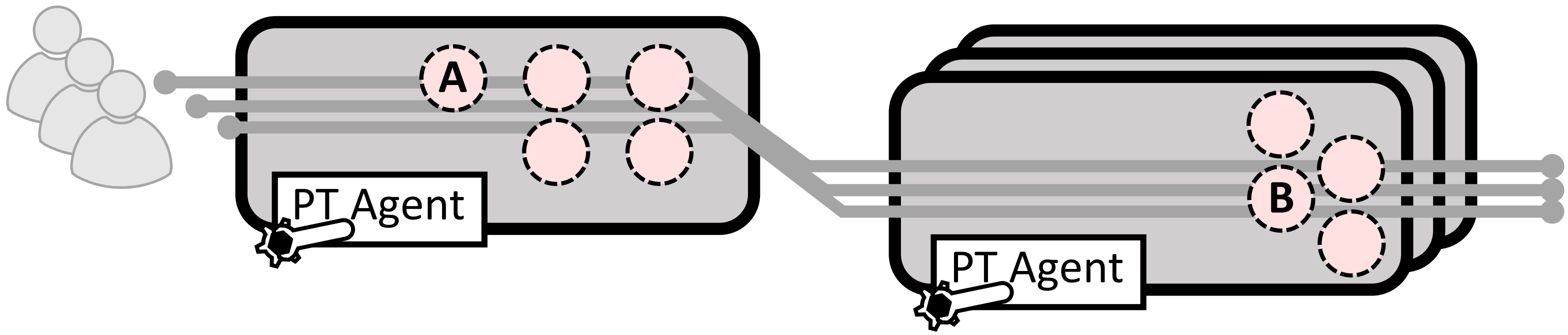
Instrumented System (+ Baggage, + PT Agent)



Tracepoints

Places where PT can add instrumentation

Instrumented System (+ Baggage, + PT Agent)

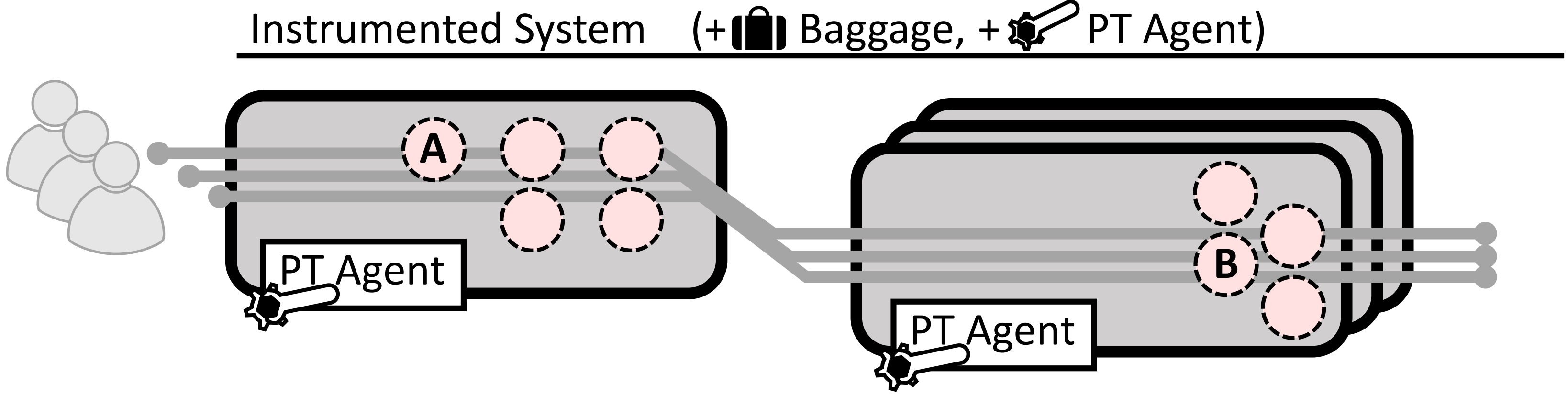


Tracepoints

Places where PT can add instrumentation

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

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



Tracepoints

Places where PT can add instrumentation

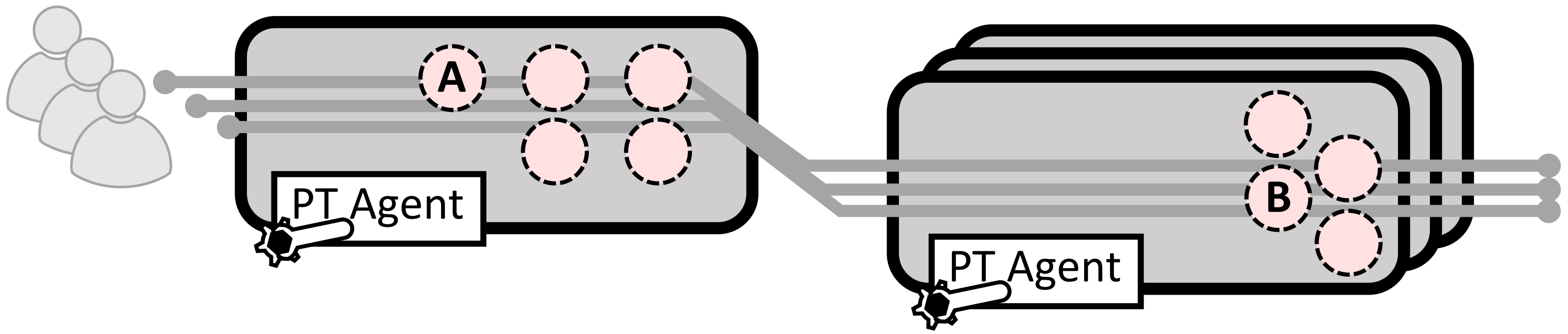
Export identifiers accessible to queries

Defaults: host, timestamp, pid, proc name

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

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

Instrumented System (+ Baggage, + PT Agent)



Tracepoints

Places where PT can add instrumentation

Export identifiers accessible to queries

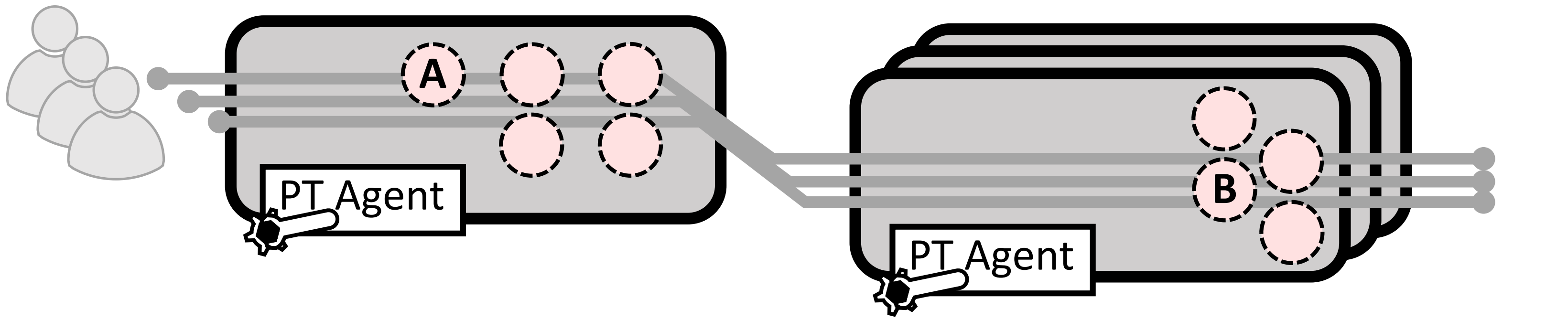
Defaults: host, timestamp, pid, proc name

Only references – not materialized until query is installed

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

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

Instrumented System (+ Baggage, + PT Agent)



Query Language

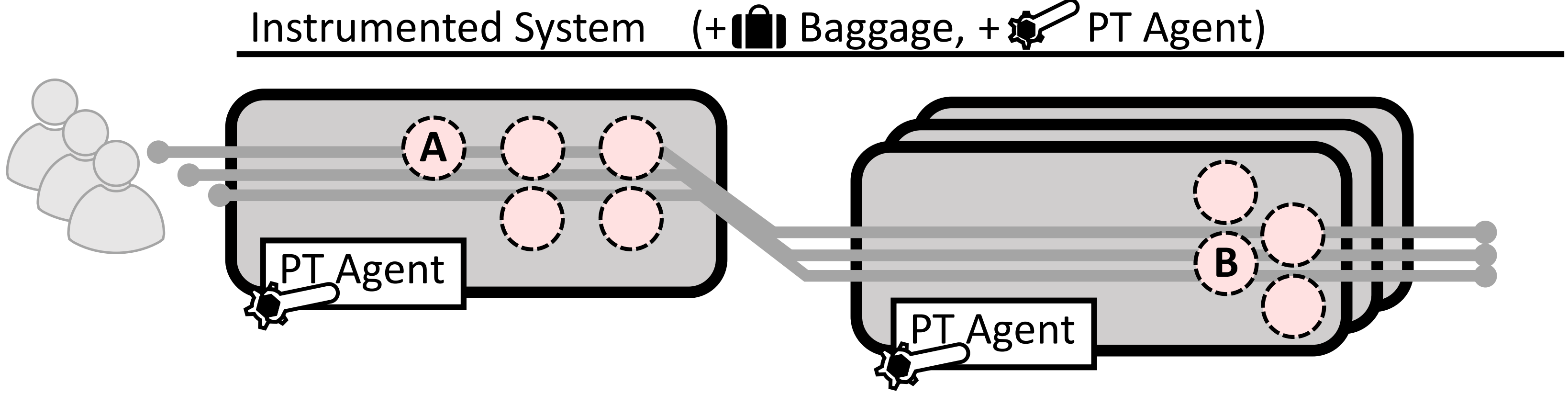
Relational query language, similar to SQL, LINQ

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

Refers to tracepoint-exported identifiers

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

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



Query Language

Relational query language, similar to SQL, LINQ

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

Refers to tracepoint-exported identifiers

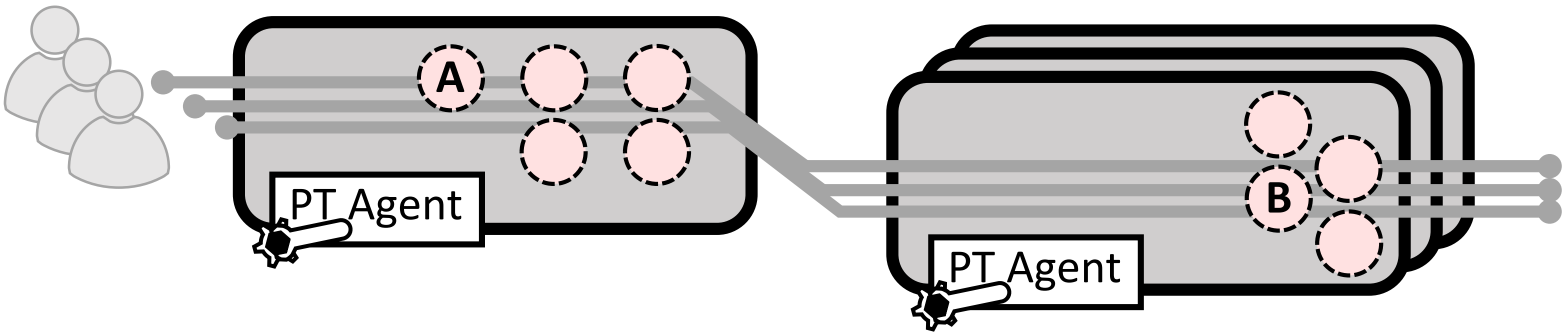
PT

```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

Tracepoint A
Class: A
Method: A1()

Tracepoint B
Class: B
Method: B1()
Exports: “delta”=delta

Instrumented System (+ Baggage, + PT Agent)



Query Language

Relational query language, similar to SQL, LINQ

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

Refers to tracepoint-exported identifiers

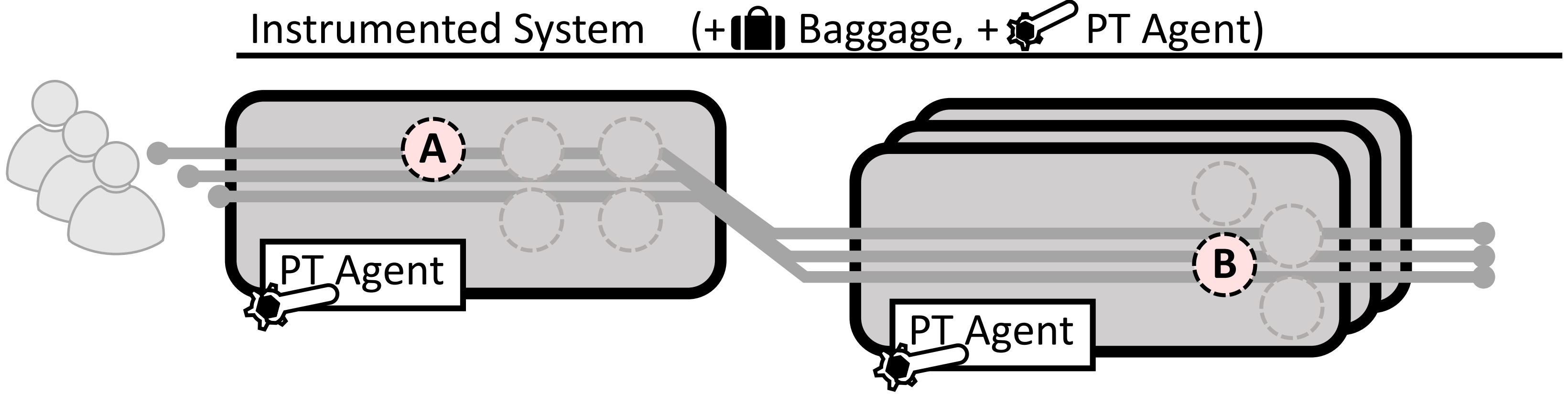
Output: stream of tuples

e.g., (procName, delta)

```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

Tracepoint A
 Class: A
 Method: A1()

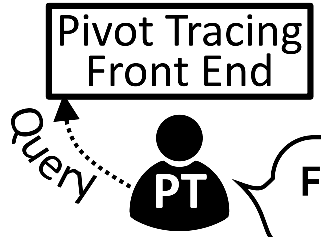
Tracepoint B
 Class: B
 Method: B1()
 Exports: “delta”=delta



Advice

Query is compiled to advice

(intermediate representation for instrumentation)



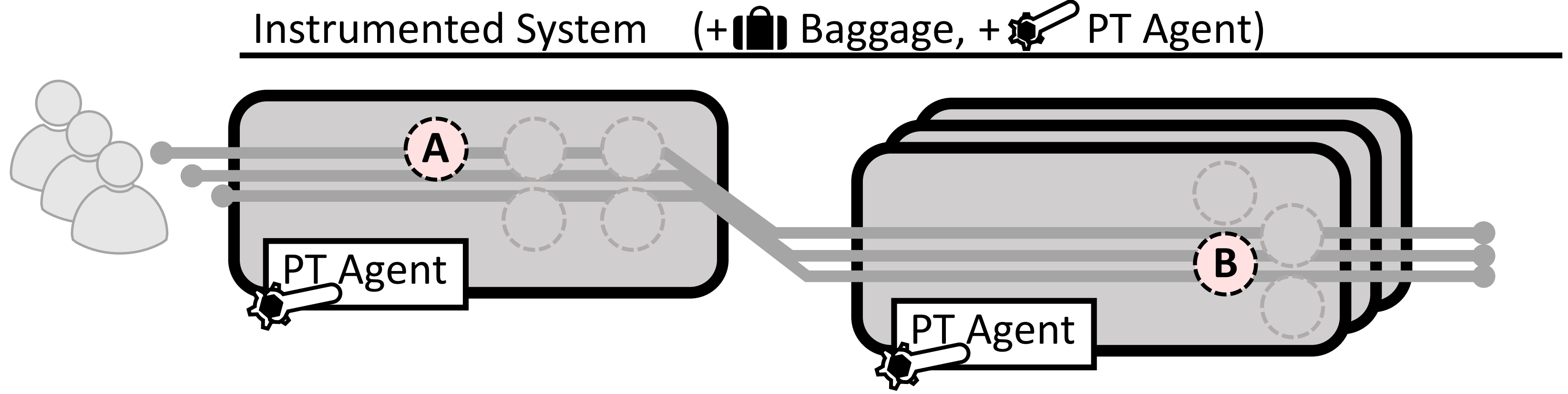
```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

Advice A1 (A)

```
OBSERVE  procName
PACK     procName
```

Advice B1 (B)

```
OBSERVE  delta
UNPACK   procName
EMIT     procName, SUM(delta)
```

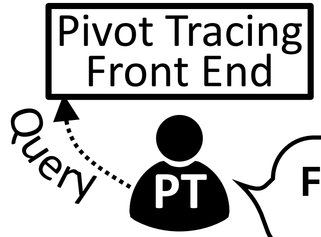


Advice

Query is compiled to advice

(intermediate representation for instrumentation)

Advice will be installed at tracepoints



```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

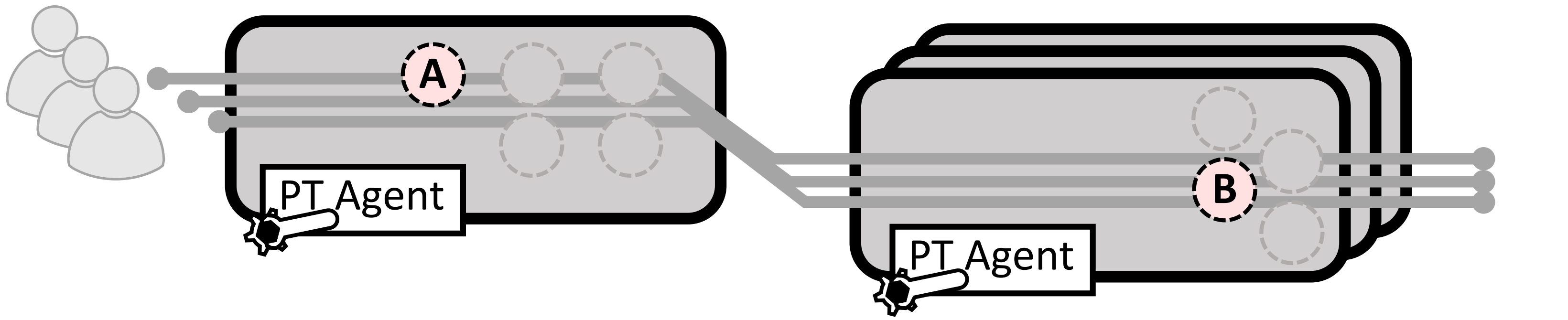
Advice A1

```
OBSERVE   procName
PACK      procName
```

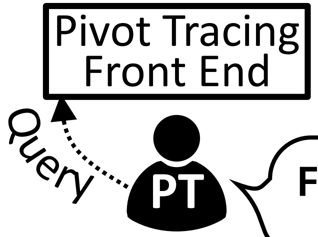
Advice B1

```
OBSERVE   delta
UNPACK    procName
EMIT      procName, SUM(delta)
```

Instrumented System (+ Baggage, + PT Agent)



Advice



```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

Query is compiled to advice

(intermediate representation for instrumentation)

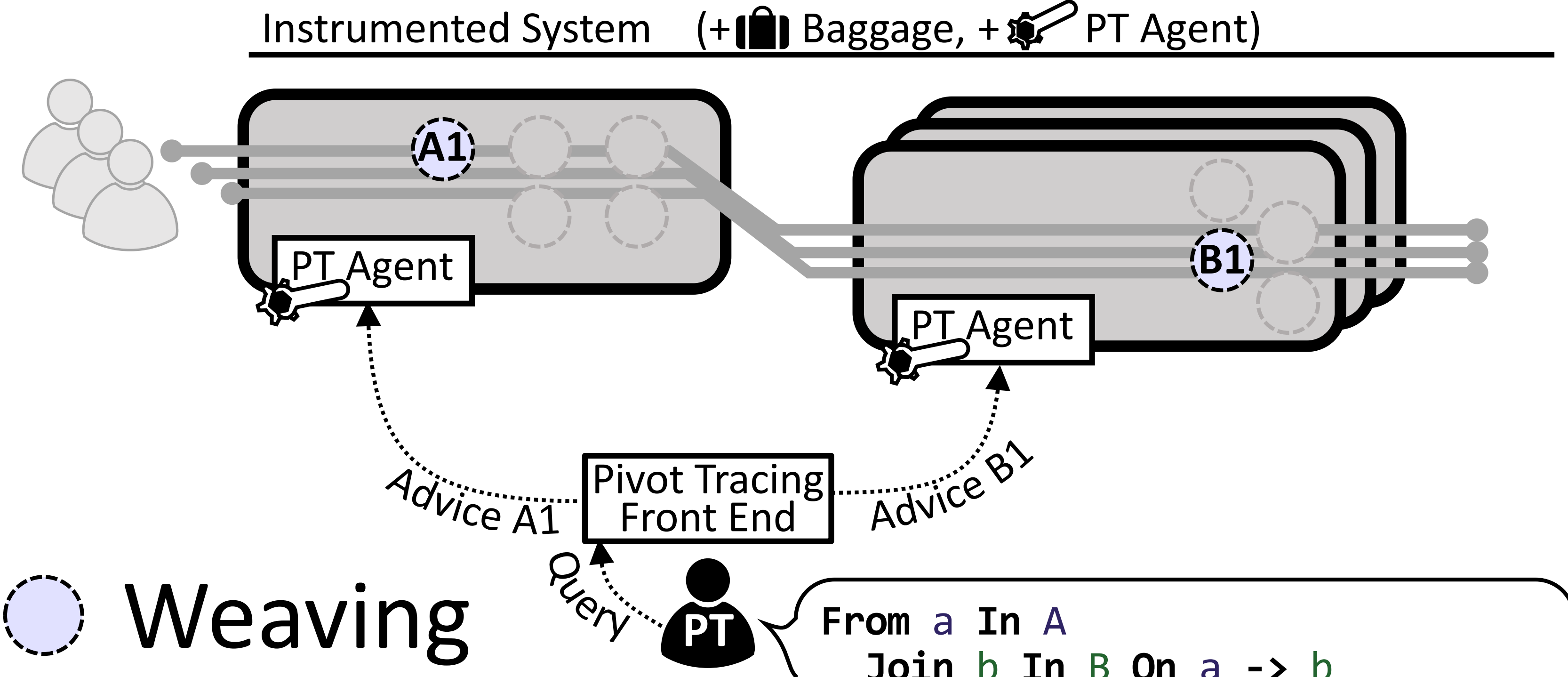
Advice will be installed at tracepoints

Limited instruction set

- OBSERVE
- PACK
- FILTER
- UNPACK
- EMIT

```
Advice A1                          A
OBSERVE   procName
PACK      procName
```

```
Advice B1                          B
OBSERVE   delta
UNPACK    procName
EMIT      procName, SUM(delta)
```



```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

Weaving

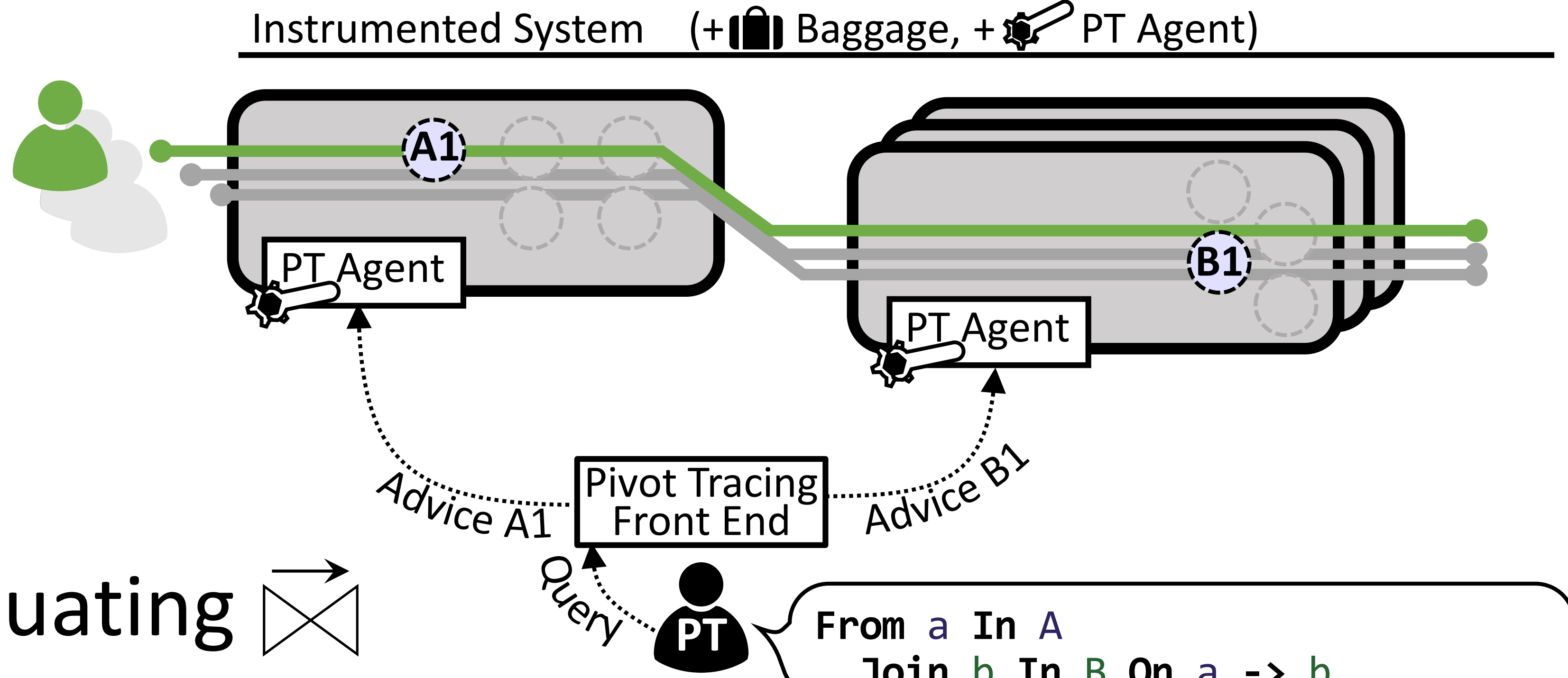
PT Agent dynamically enables advice at tracepoints

Advice A1 (A)

```
OBSERVE   procName
PACK      procName
```

Advice B1 (B)

```
OBSERVE   delta
UNPACK    procName
EMIT      procName, SUM(delta)
```



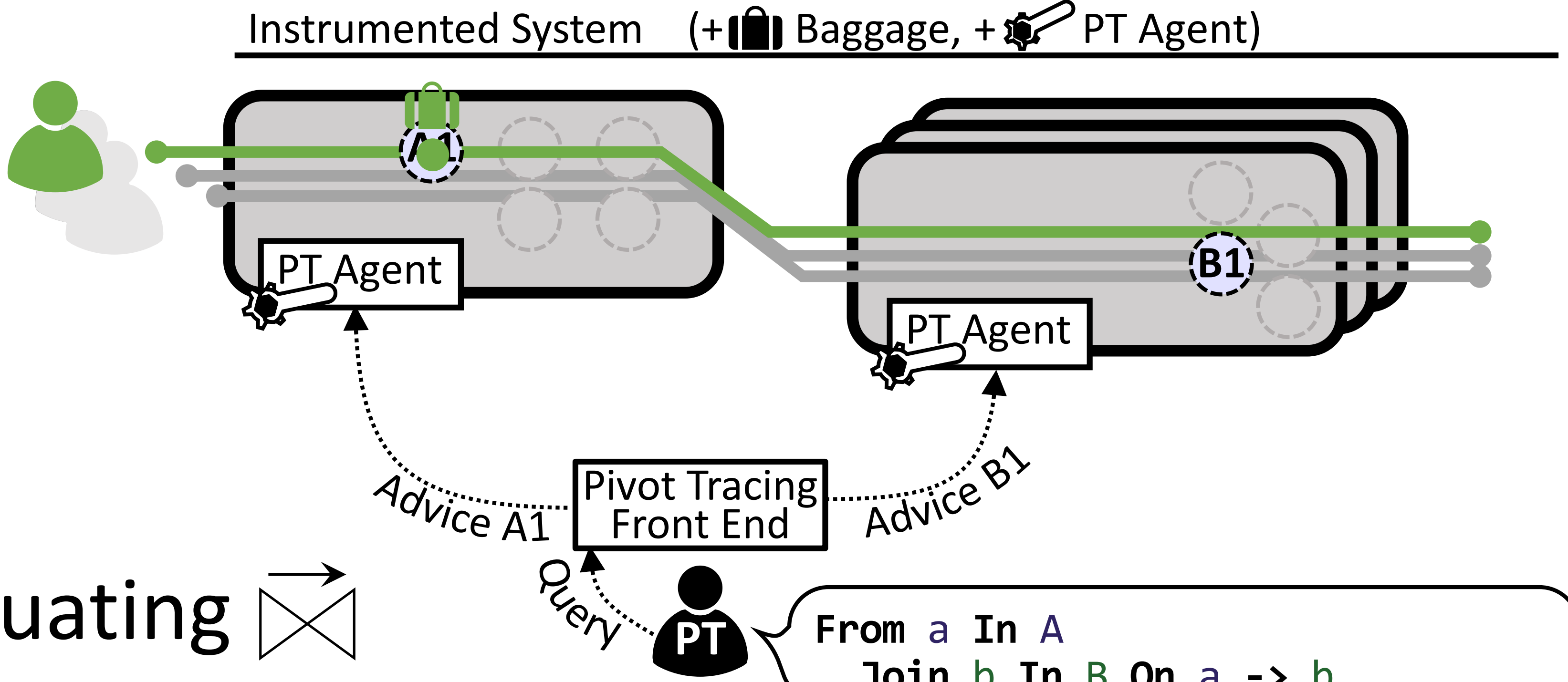
Evaluating $\overrightarrow{\bowtie}$

```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

```
Advice A1                              A
OBSERVE   procName
PACK      procName
```

```
Advice B1                                  B
OBSERVE   delta
UNPACK    procName
EMIT      procName, SUM(delta)
```

Evaluating ⋈



```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

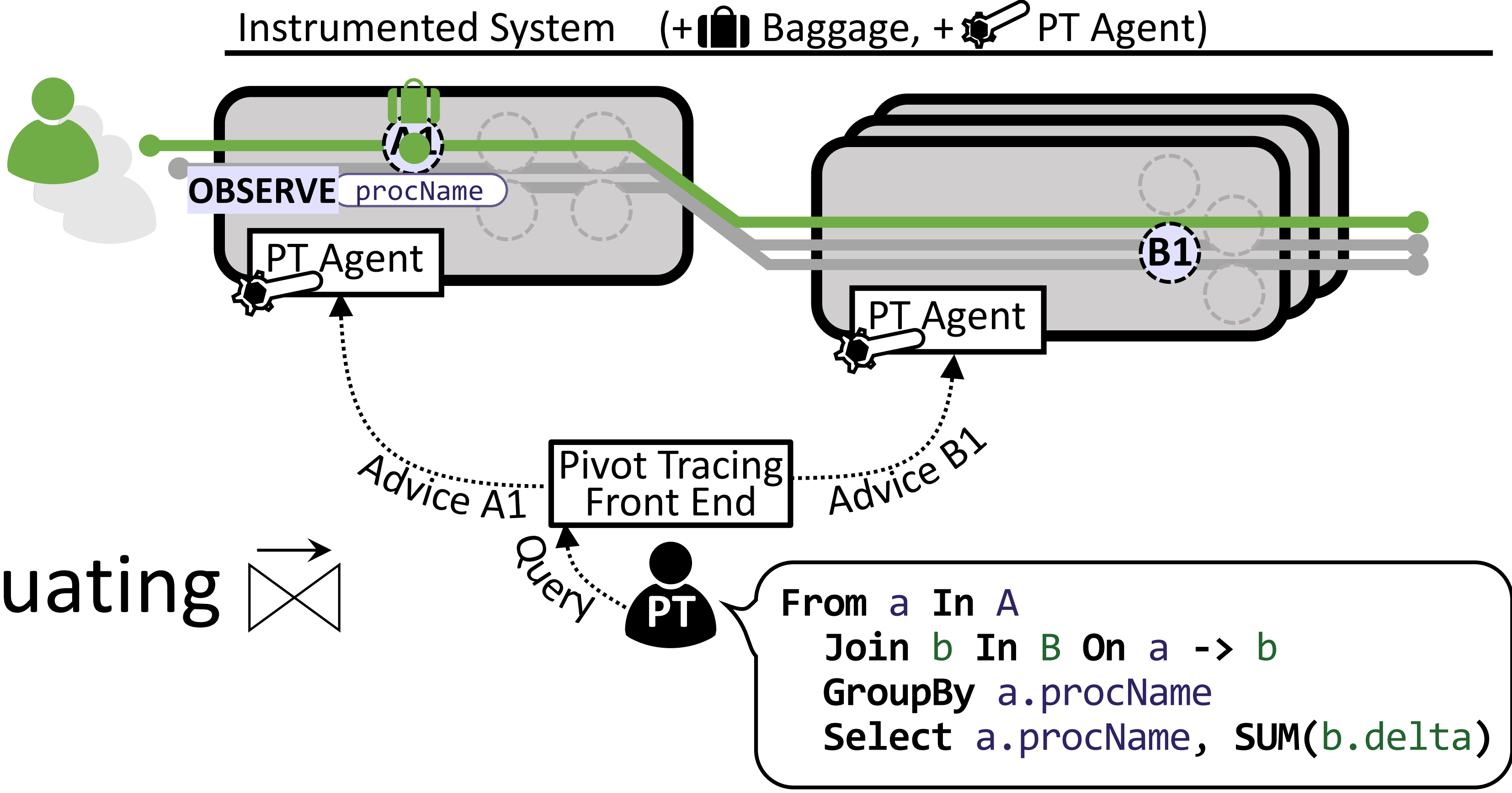
Advice A1 A

OBSERVE procName
PACK procName

Advice B1 B

OBSERVE delta
UNPACK procName
EMIT procName, SUM(delta)

Evaluating $\overrightarrow{\bowtie}$



```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

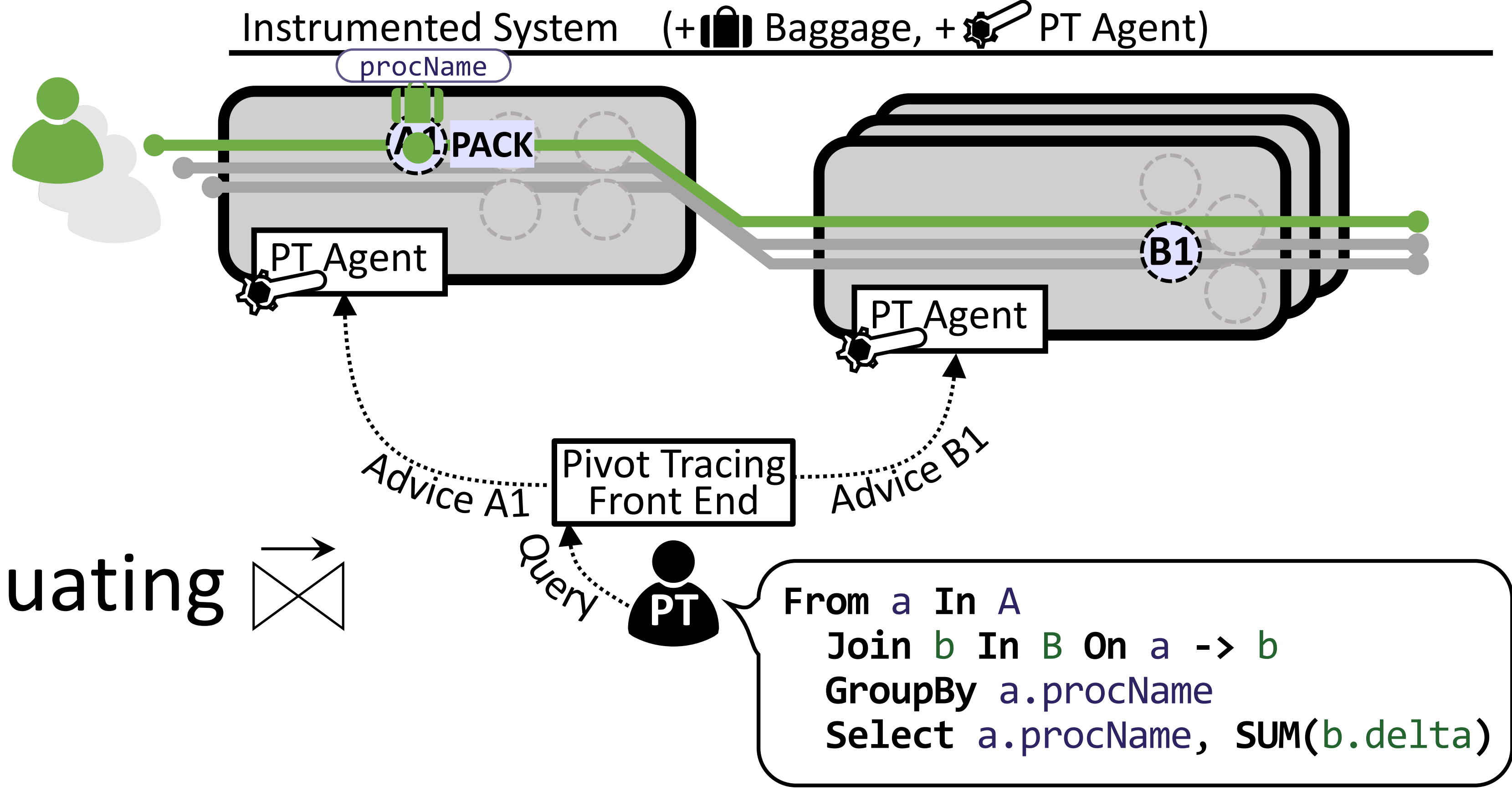
Advice A1 A

OBSERVE	procName
PACK	procName

Advice B1 B

OBSERVE	delta
UNPACK	procName
EMIT	procName, SUM(delta)

Evaluating $\overrightarrow{\bowtie}$



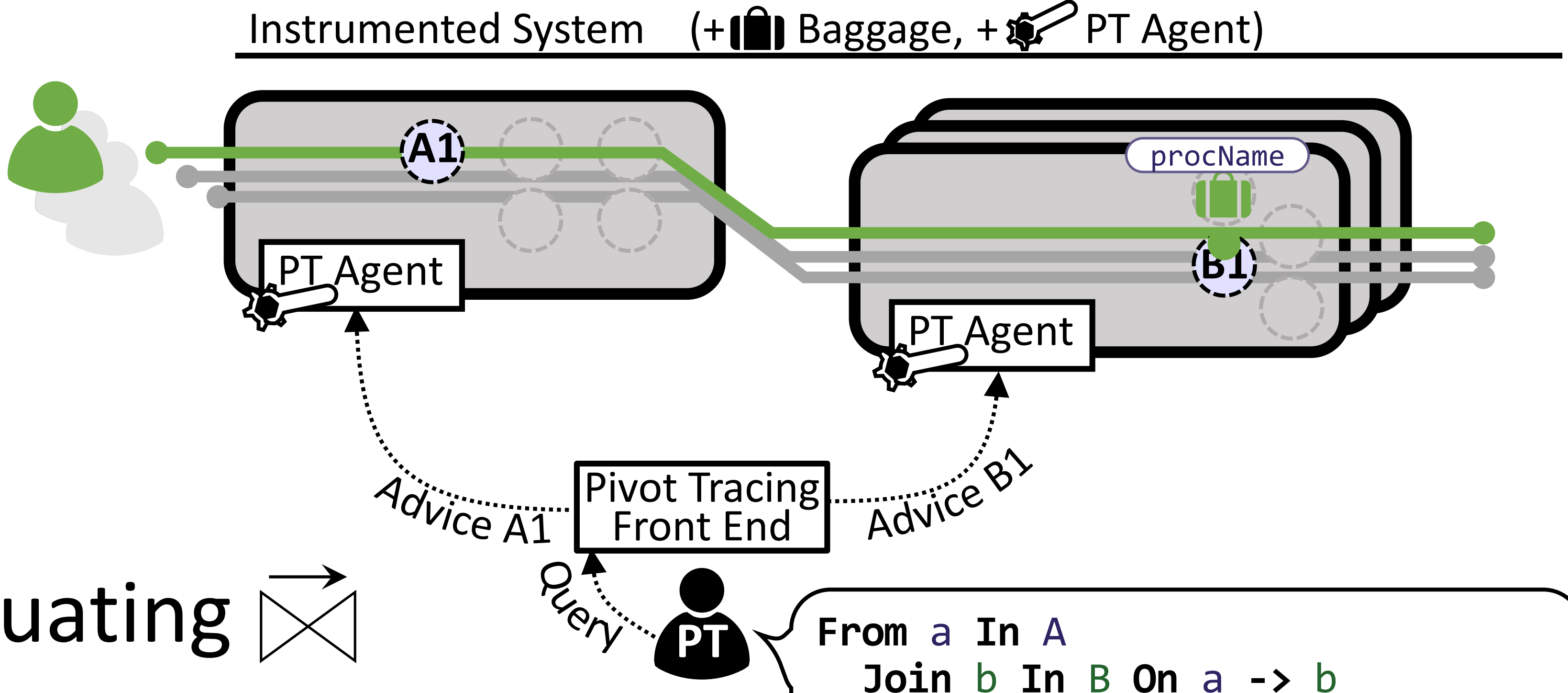
```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

Advice A1 A

OBSERVE	procName
PACK	procName

Advice B1 B

OBSERVE	delta
UNPACK	procName
EMIT	procName, SUM(delta)



Evaluating $\overrightarrow{\bowtie}$

```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

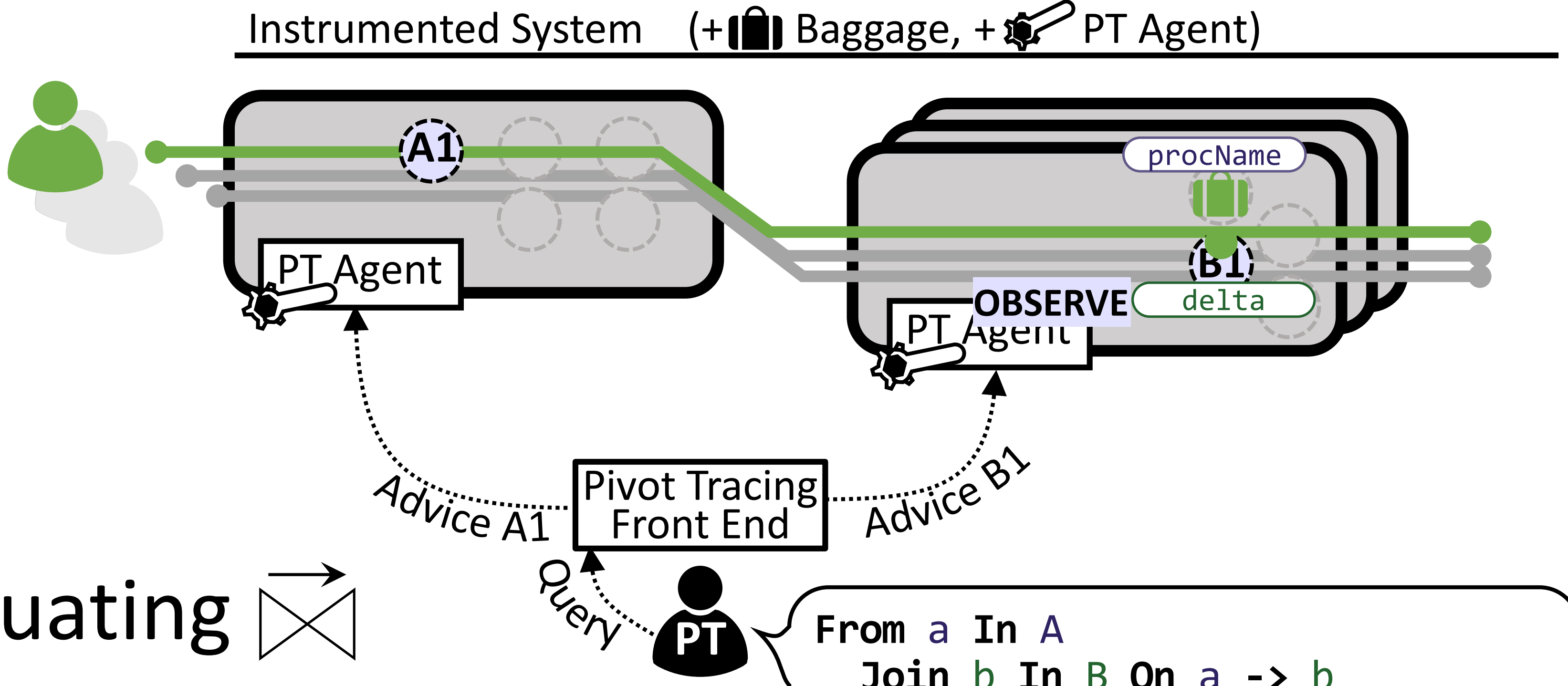
Advice A1 A

```
OBSERVE   procName
PACK      procName
```

Advice B1 B

```
OBSERVE   delta
UNPACK    procName
EMIT      procName, SUM(delta)
```

Evaluating $\overrightarrow{\bowtie}$



```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

Advice A1 A

OBSERVE procName
PACK procName

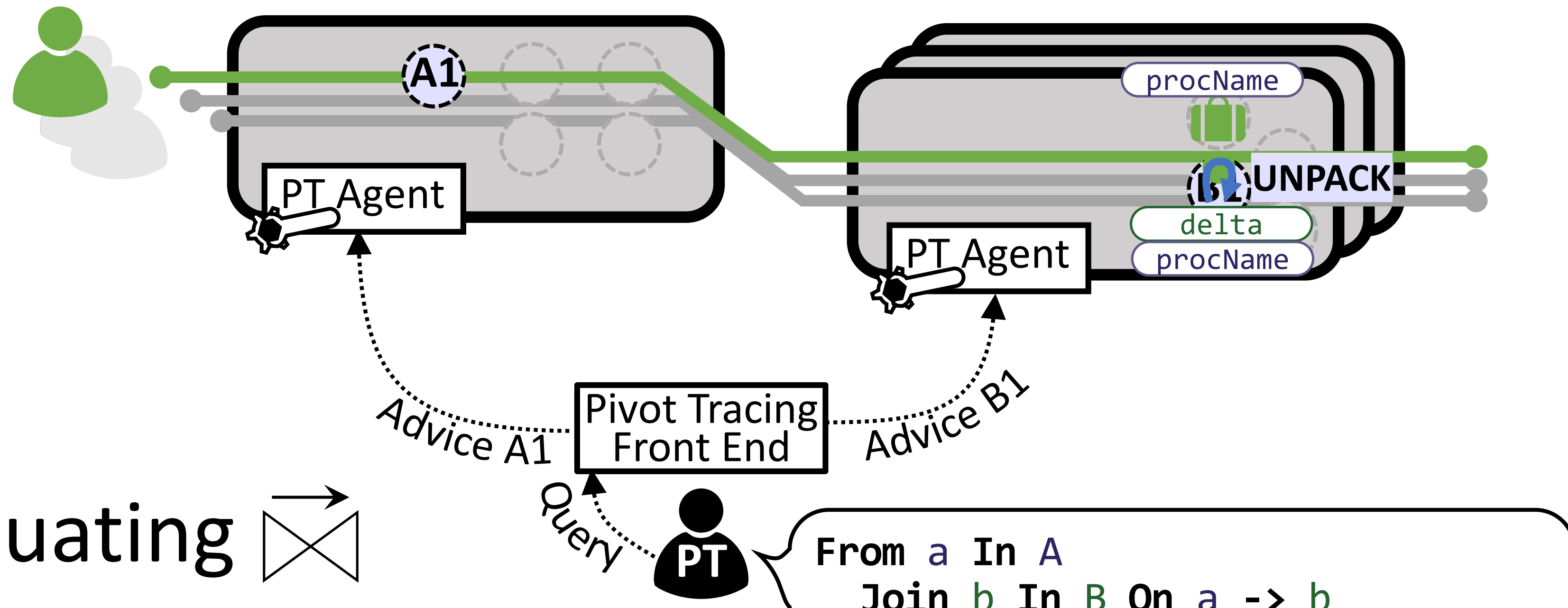
Advice B1 B

OBSERVE delta

UNPACK procName

EMIT procName, SUM(delta)

Instrumented System (+ Baggage, + PT Agent)



Evaluating $\overrightarrow{\bowtie}$

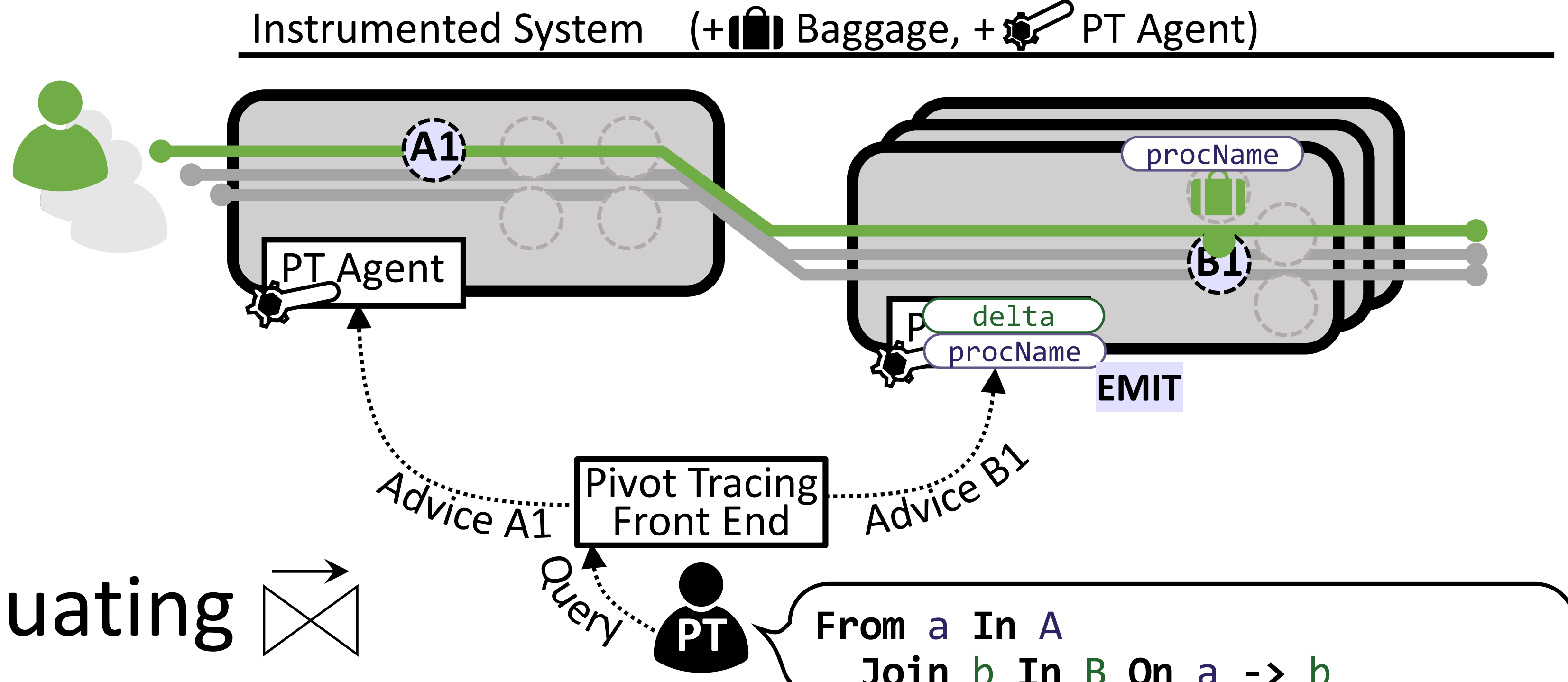
```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

Advice A1 A

OBSERVE procName
PACK procName

Advice B1 B

OBSERVE delta
UNPACK procName
EMIT procName, SUM(delta)



Evaluating $\overrightarrow{\bowtie}$

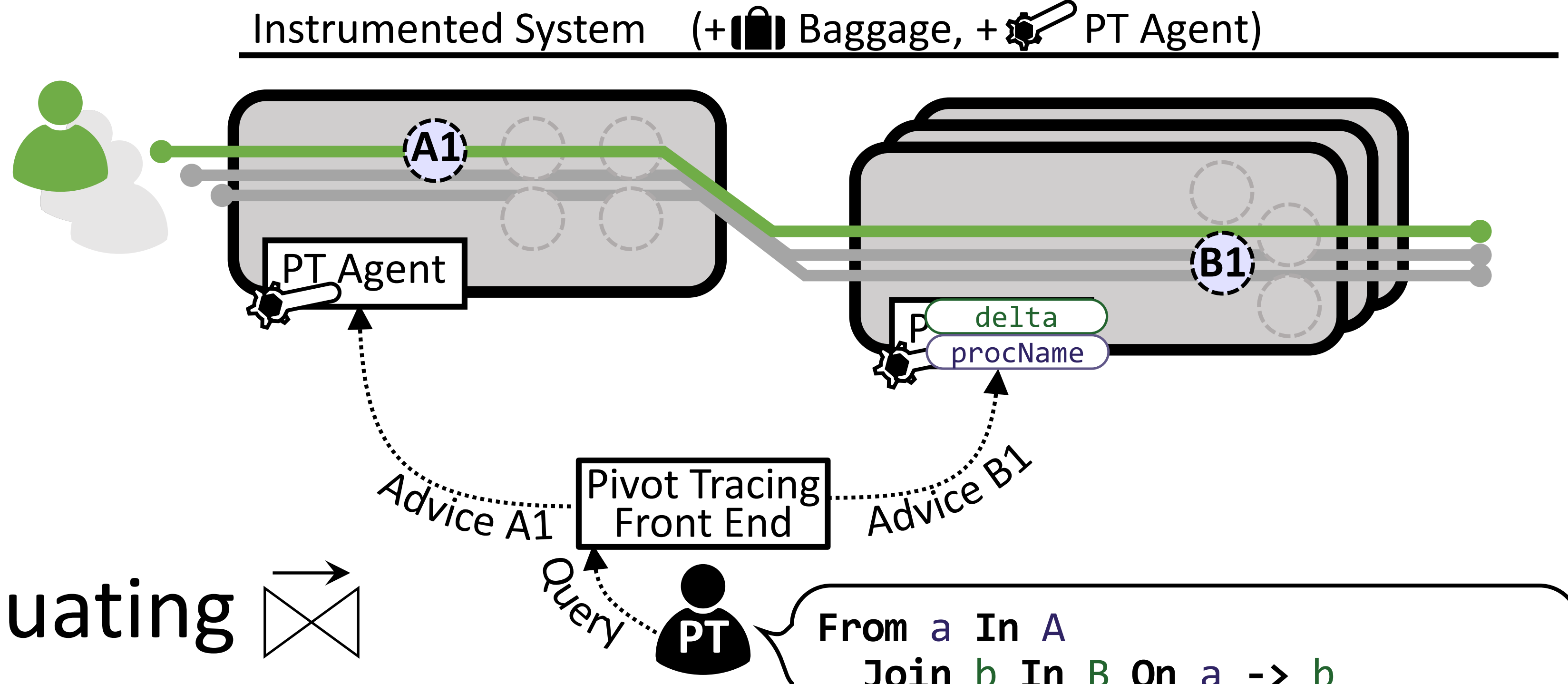
```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

A

```
Advice A1
OBSERVE   procName
PACK      procName
```

B

```
Advice B1
OBSERVE   delta
UNPACK    procName
EMIT      procName, SUM(delta)
```



```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

Evaluating $\overrightarrow{\bowtie}$

Baggage explicitly follows execution

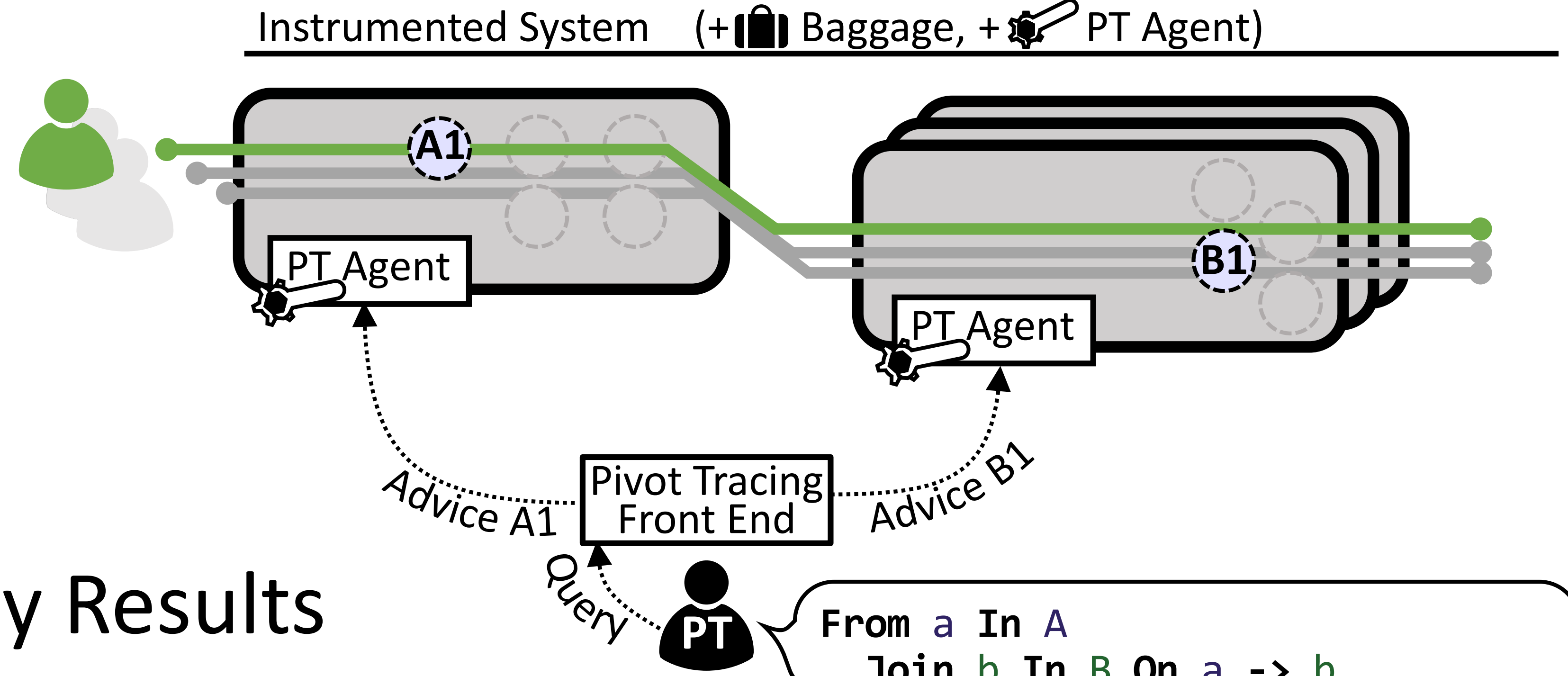
Evaluated inline during a request
 (no global aggregation needed)

Advice A1 A

```
OBSERVE   procName
PACK      procName
```

Advice B1 B

```
OBSERVE   delta
UNPACK    procName
EMIT      procName, SUM(delta)
```



Query Results

Tuples are accumulated locally in PT Agent

```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

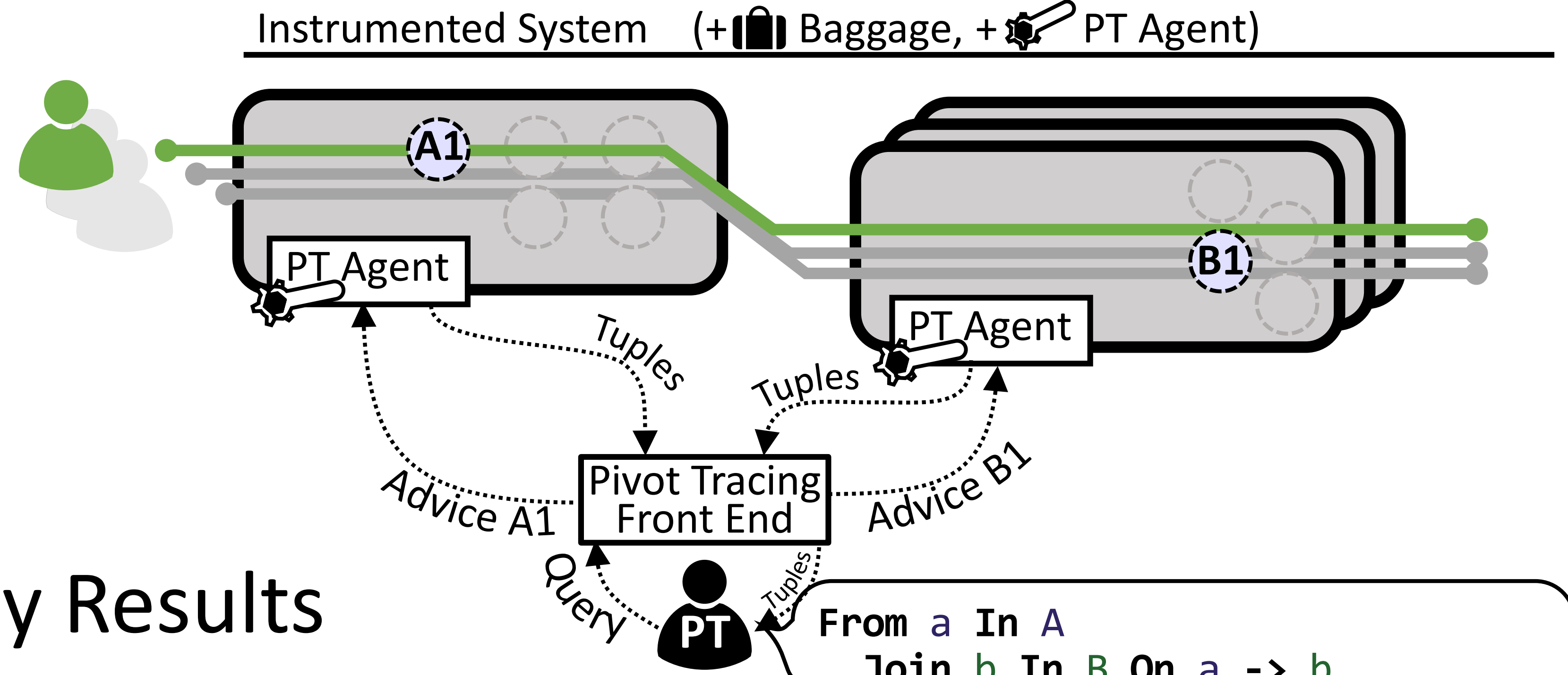
Advice A1 A

OBSERVE	procName
PACK	procName

Advice B1 B

OBSERVE	delta
UNPACK	procName
EMIT	procName, SUM(delta)

Query Results



```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

Tuples are accumulated locally in PT Agent

Periodically reported back to user

e.g., every second

Advice A1 A

OBSERVE	procName
PACK	procName

Advice B1 B

OBSERVE	delta
UNPACK	procName
EMIT	procName, SUM(delta)

Pivot Tracing

Evaluation

Java-Based Implementation

Java-Based Implementation

PT Agent

PT agent thread that runs inside each process

- Javassist for dynamic instrumentation
- PubSub to receive commands / send tuples

Java-Based Implementation

PT Agent

PT agent thread that runs inside each process

- Javassist for dynamic instrumentation
- PubSub to receive commands / send tuples

Baggage library for use by instrumented system

- Data format specified using Protocol Buffers

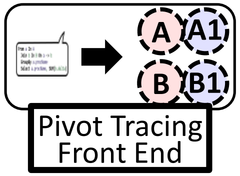
Java-Based Implementation

PT agent thread that runs inside each process

- Javassist for dynamic instrumentation
- PubSub to receive commands / send tuples

Baggage library for use by instrumented system

- Data format specified using Protocol Buffers

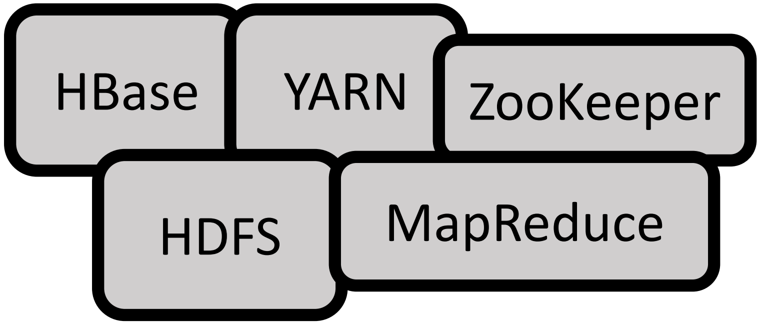


Front-end client library

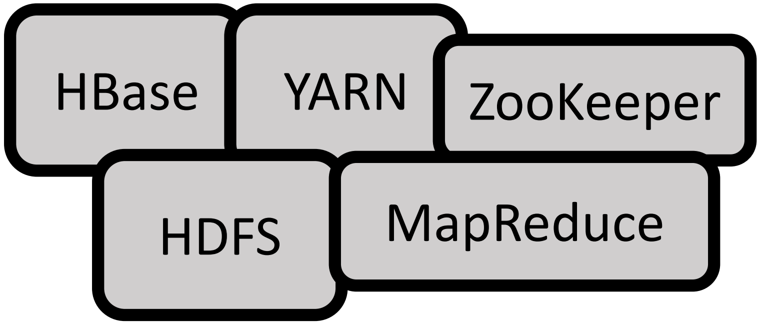
- Define tracepoints and write text queries
- Compile queries to advice
- Submit advice to PT agents

Pivot Tracing Enabled (+ Baggage, + PT Agent)

hadoop



Pivot Tracing Enabled (+ Baggage, + PT Agent)



HBase

YARN

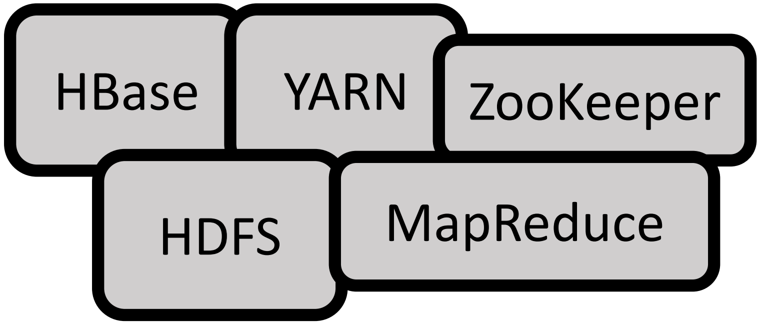
ZooKeeper

HDFS

MapReduce

Adding Baggage: ~50-200 lines of code per system

Pivot Tracing Enabled (+ Baggage, + PT Agent)

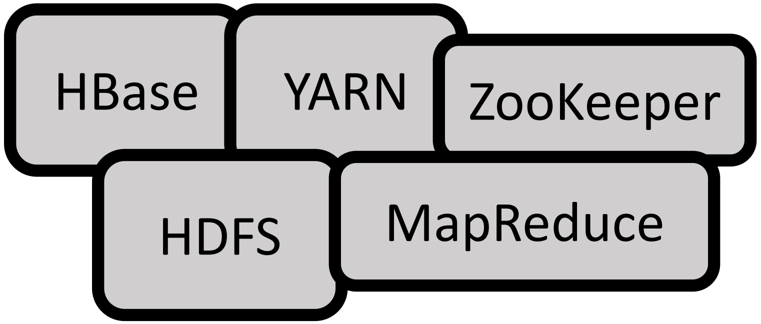


Adding Baggage: ~50-200 lines of code per system

Primarily modifying execution boundaries:

- Thread, Runnable, Callable, Queue
- RPC invocations

Pivot Tracing Enabled (+ Baggage, + PT Agent)



Adding Baggage: ~50-200 lines of code per system

Primarily modifying execution boundaries:

Thread, Runnable, Callable, Queue

RPC invocations

ONE TIME

Pivot Tracing Overheads

- Pivot Tracing Enabled (+🧳 Baggage, + 🔧 PT Agent)

  Application level benchmarks: **baseline** 0.3% overhead

Pivot Tracing Overheads

- Pivot Tracing Enabled (+🧳 Baggage, + 🔧 PT Agent)

  Application level benchmarks: **baseline** 0.3% overhead

- No overhead for queries / tracepoints until installed

Pivot Tracing Overheads

- Pivot Tracing Enabled (+💼 Baggage, + 🔧 PT Agent)

  Application level benchmarks: **baseline** 0.3% overhead

- No overhead for queries / tracepoints until installed

- With queries from paper installed

  Application level benchmarks: max 14.3% overhead

  (CPU-only lookups)

Pivot Tracing Overheads

- Pivot Tracing Enabled (+🧳 Baggage, + 🔧 PT Agent)

  Application level benchmarks: **baseline** 0.3% overhead

- No overhead for queries / tracepoints until installed

- With queries from paper installed

  Application level benchmarks: max 14.3% overhead

  (CPU-only lookups)

  Largest baggage size: ~137 bytes

Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems

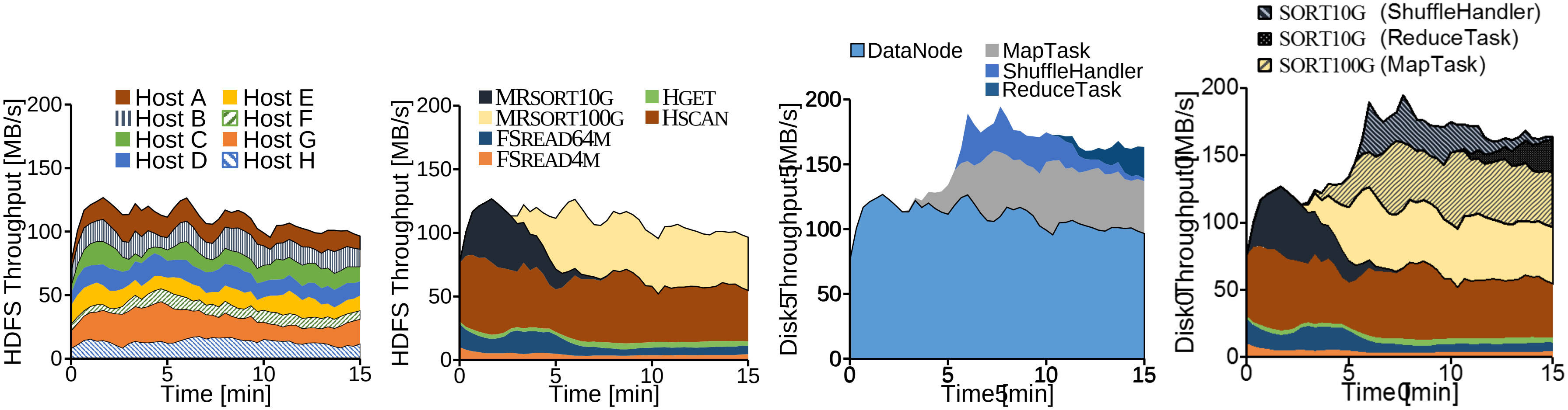
Jonathan Mace Ryan Roelke Rodrigo Fonseca
Brown University

Abstract

Experiments

Experiments

1. Monitoring queries with various groupings



Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems

Jonathan Mace Ryan Roelke Rodrigo Fonseca
Brown University

[illegible]

Experiments

1. Monitoring queries with various groupings
2. Decomposing request latencies

Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems

Jonathan Mace Ryan Roelke Rodrigo Fonseca
Brown University

Experiments

1. Monitoring queries with various groupings

2. Decomposing request latencies

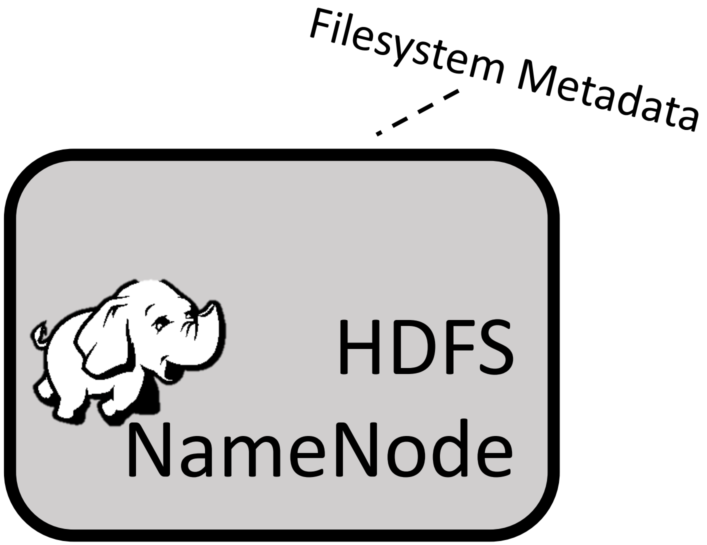
3. Debugging recurring problems

Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems

Jonathan Mace Ryan Roelke Rodrigo Fonseca
Brown University

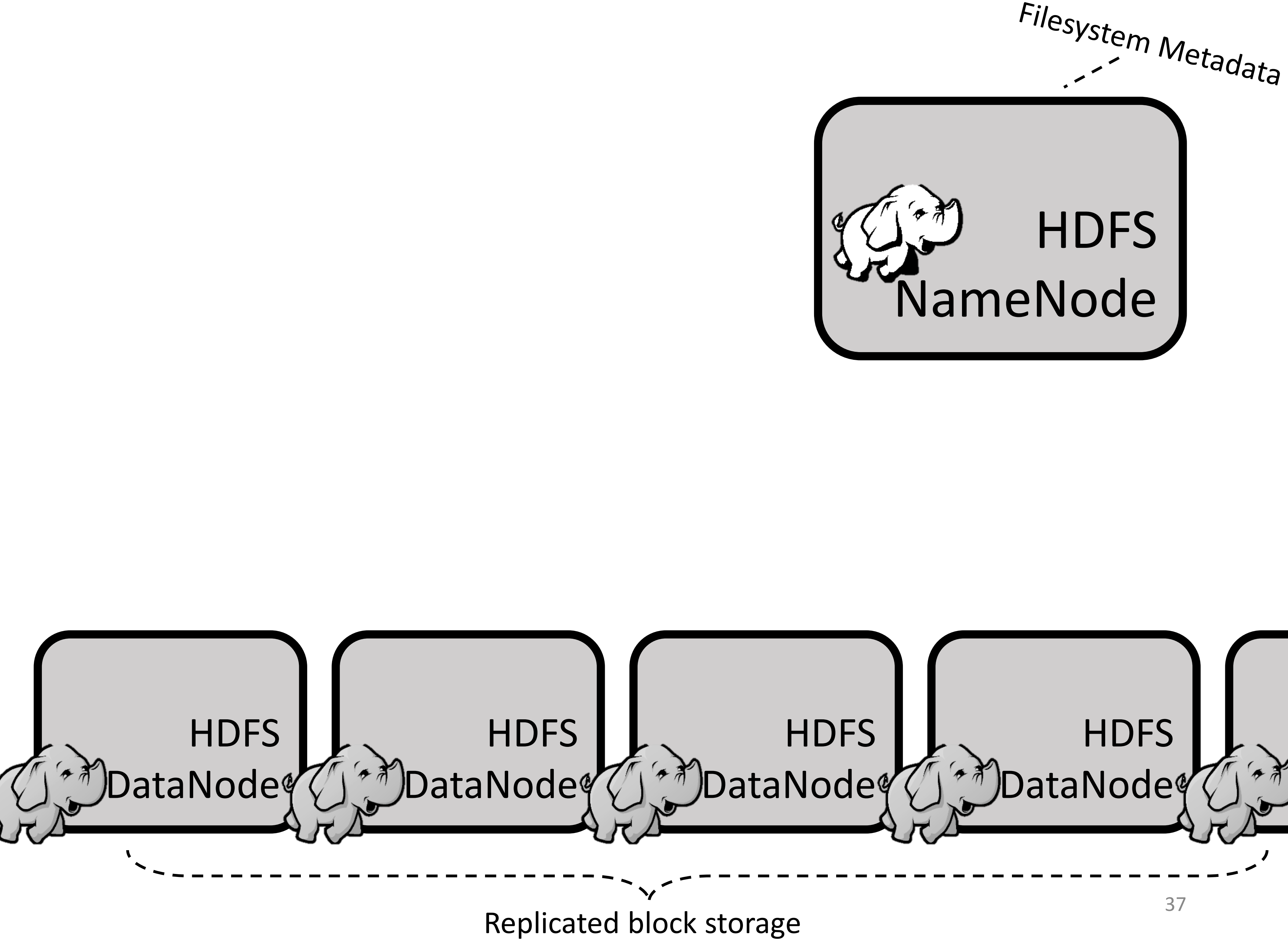
Experiments

1. Monitoring queries with various groupings
2. Decomposing request latencies
3. Debugging recurring problems



Filesystem Metadata

HDFS
NameNode



Filesystem Metadata

HDFS NameNode

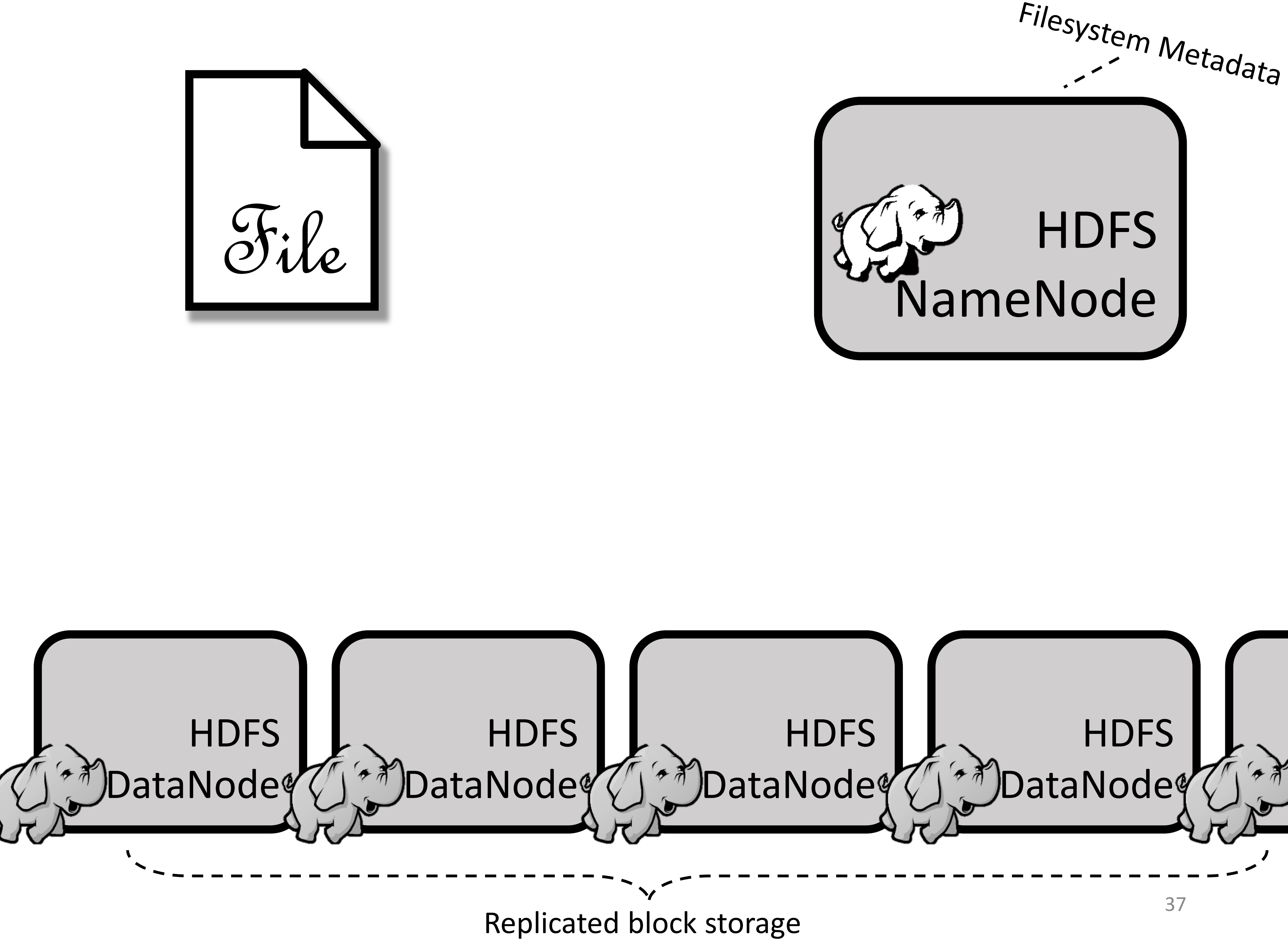
HDFS DataNode

HDFS DataNode

HDFS DataNode

HDFS DataNode

Replicated block storage



File

Filesystem Metadata

HDFS NameNode

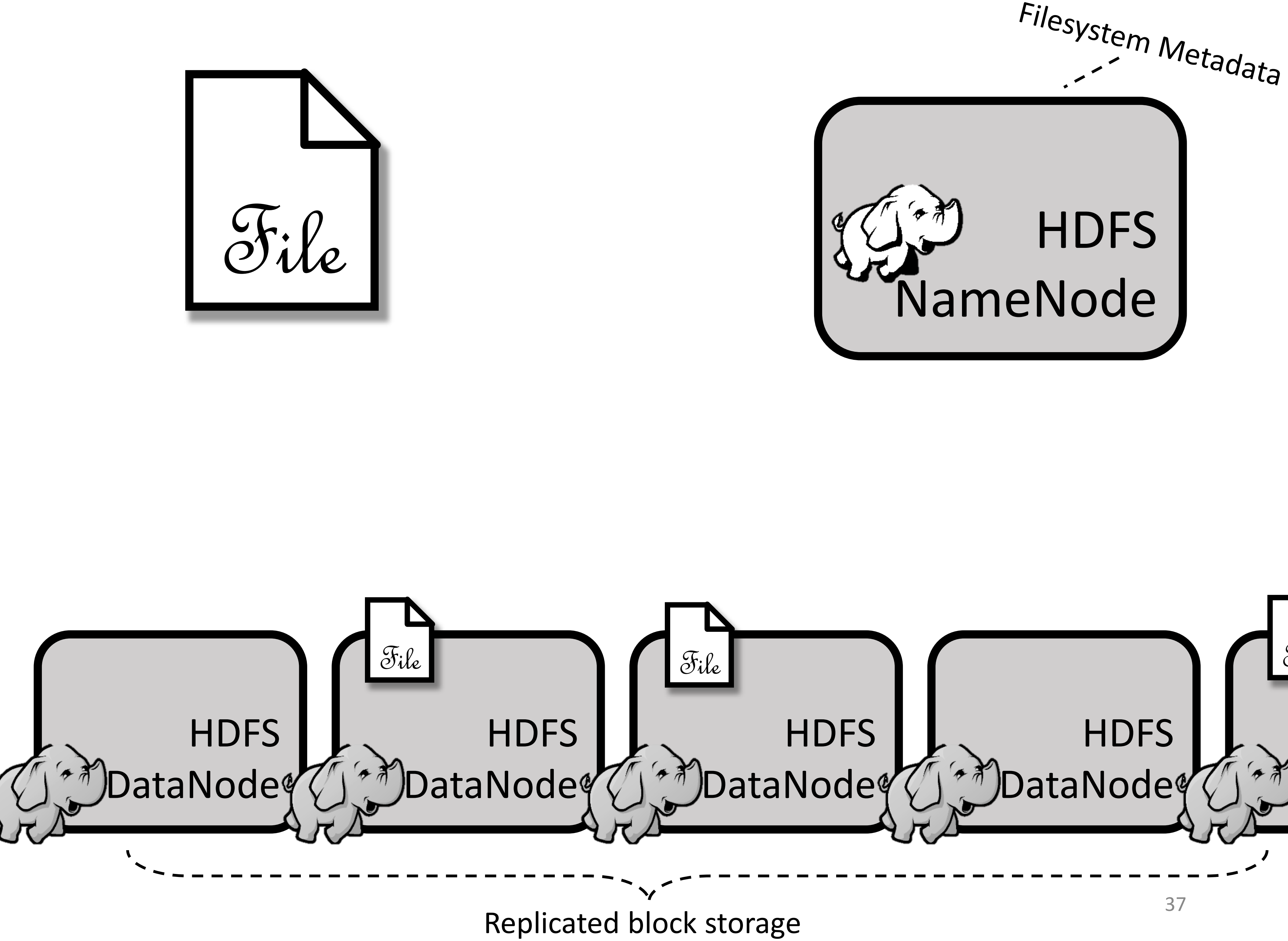
HDFS DataNode

HDFS DataNode

HDFS DataNode

HDFS DataNode

Replicated block storage



File

Filesystem Metadata

HDFS
NameNode

File

File

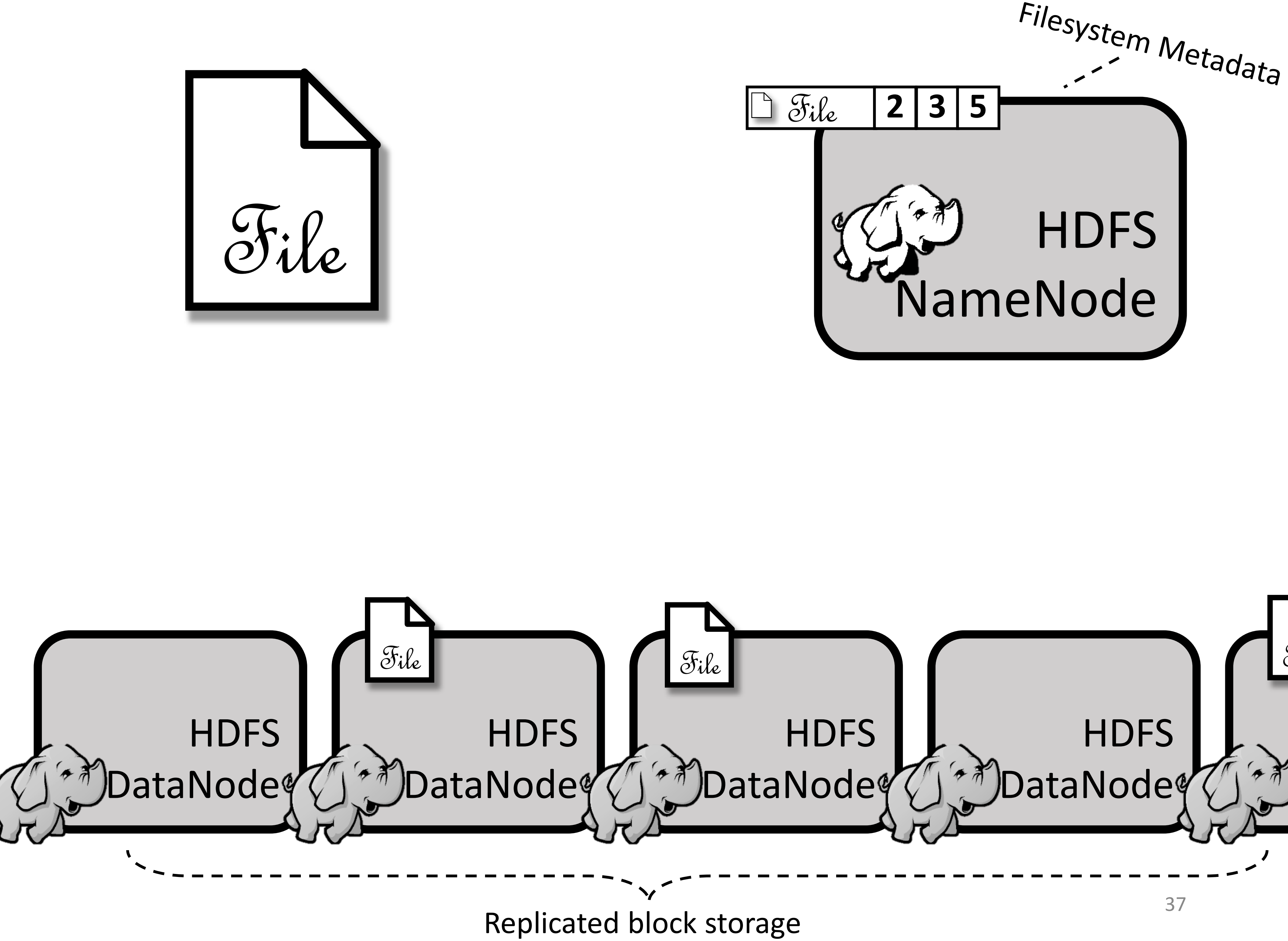
HDFS
DataNode

HDFS
DataNode

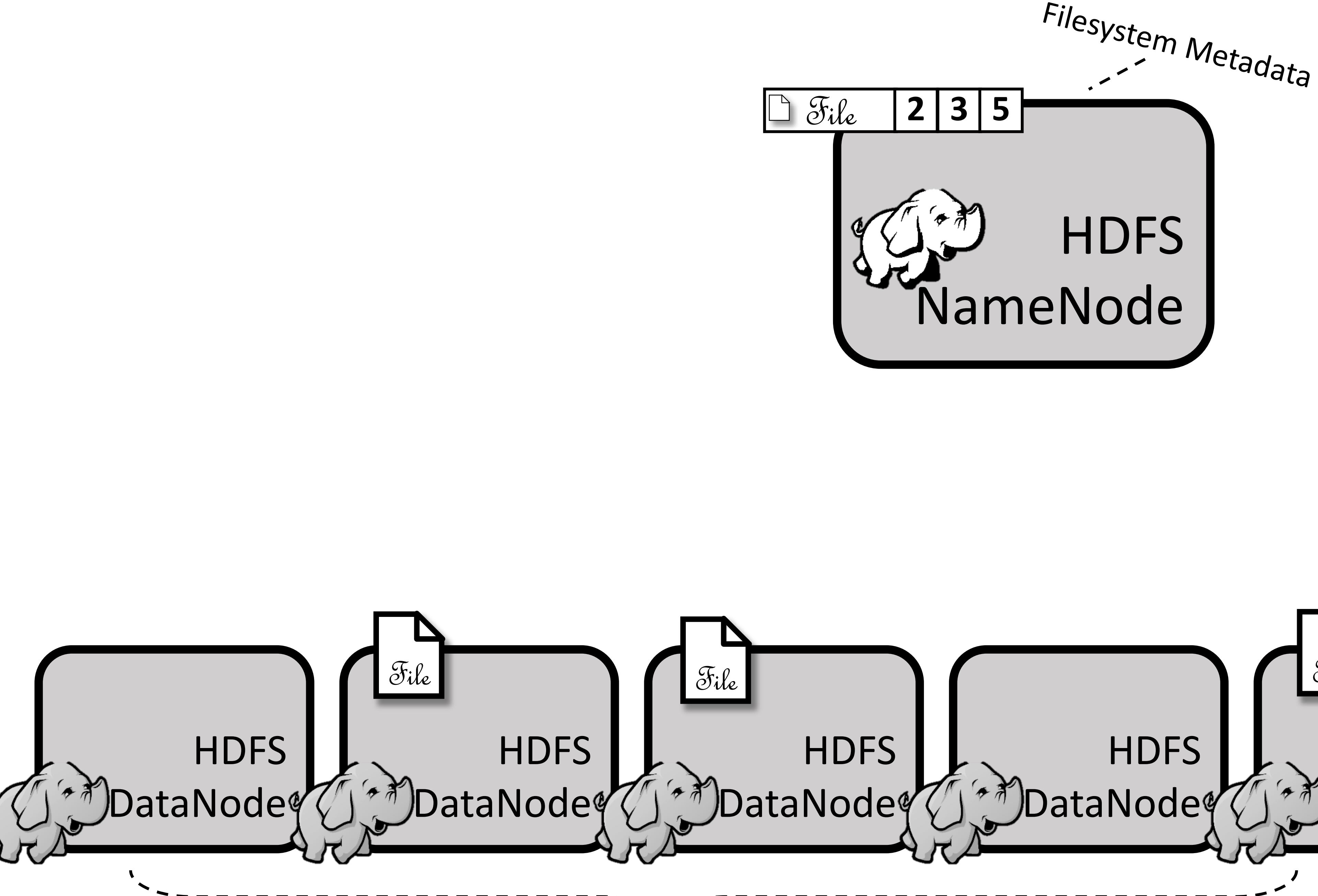
HDFS
DataNode

HDFS
DataNode

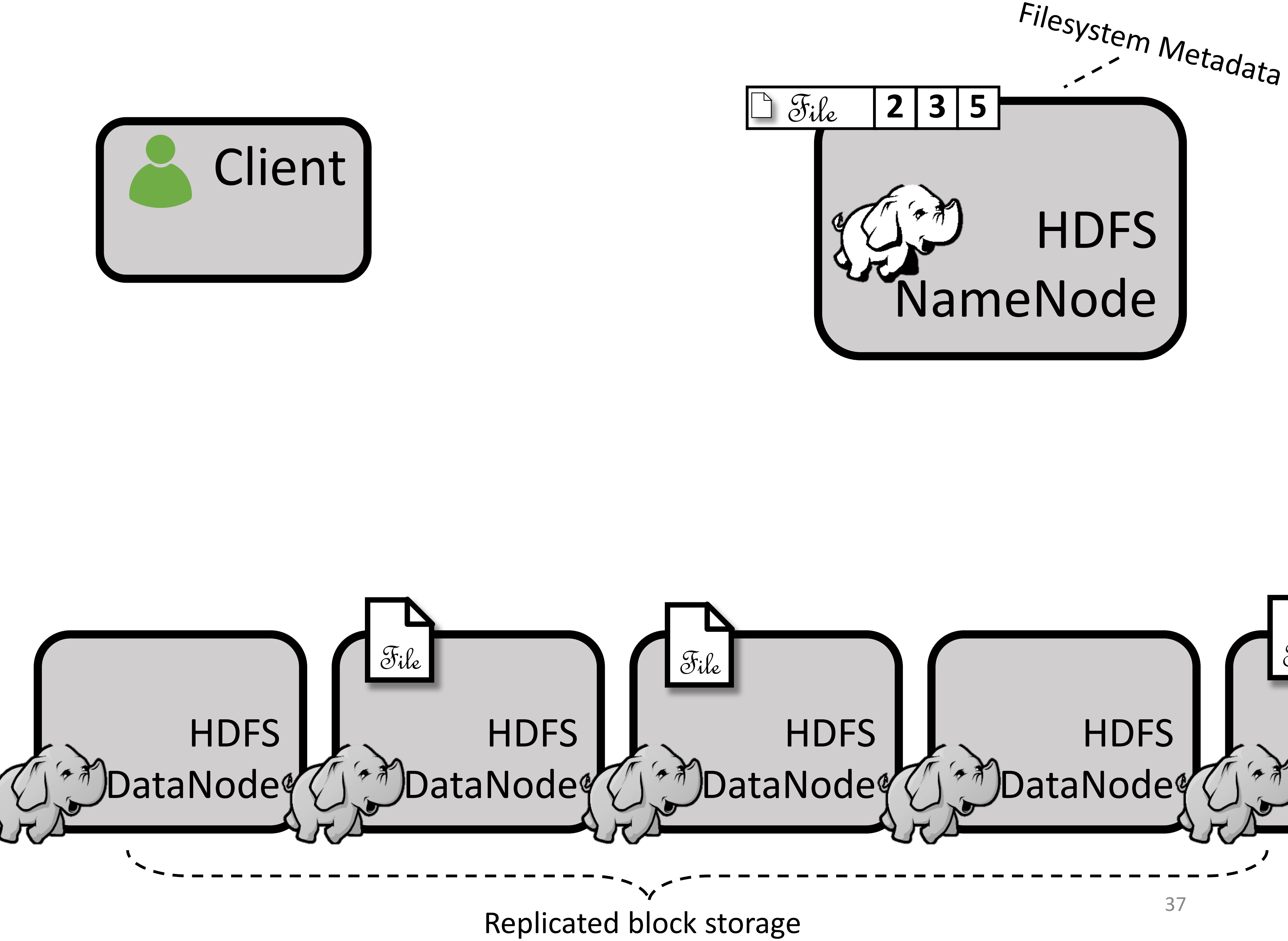
Replicated block storage

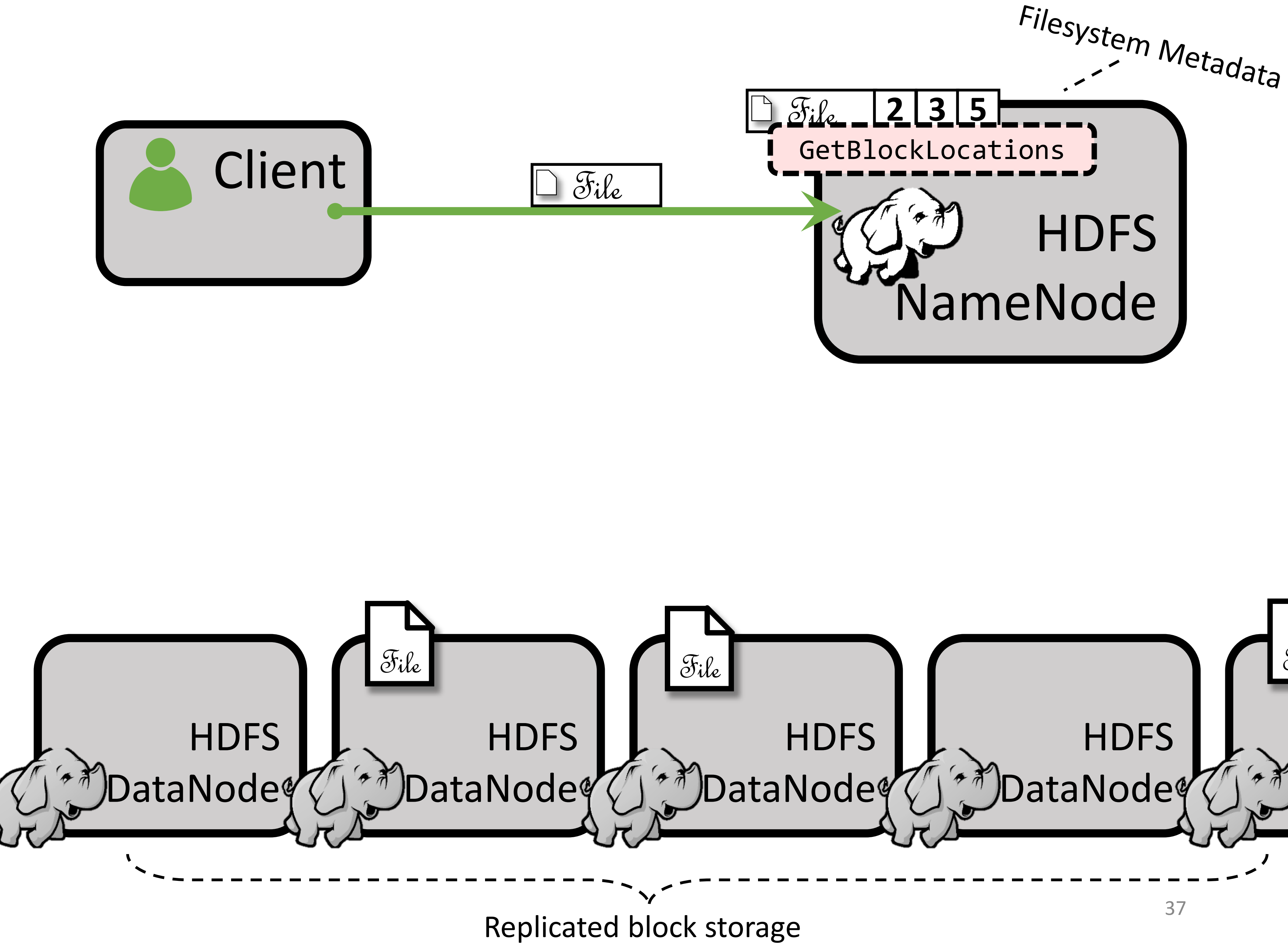


Filesystem Metadata
File 2 3 5
File
HDFS NameNode
File
File
HDFS DataNode
HDFS DataNode
HDFS DataNode
HDFS DataNode
Replicated block storage



Replicated block storage


Filesystem Metadata
File
2
3
5
Client
HDFS
NameNode
File
File
HDFS
DataNode
HDFS
DataNode
HDFS
DataNode
HDFS
DataNode
Replicated block storage



Filesystem Metadata

File 2 3 5

GetBlockLocations

Client

File

HDFS NameNode

File

File

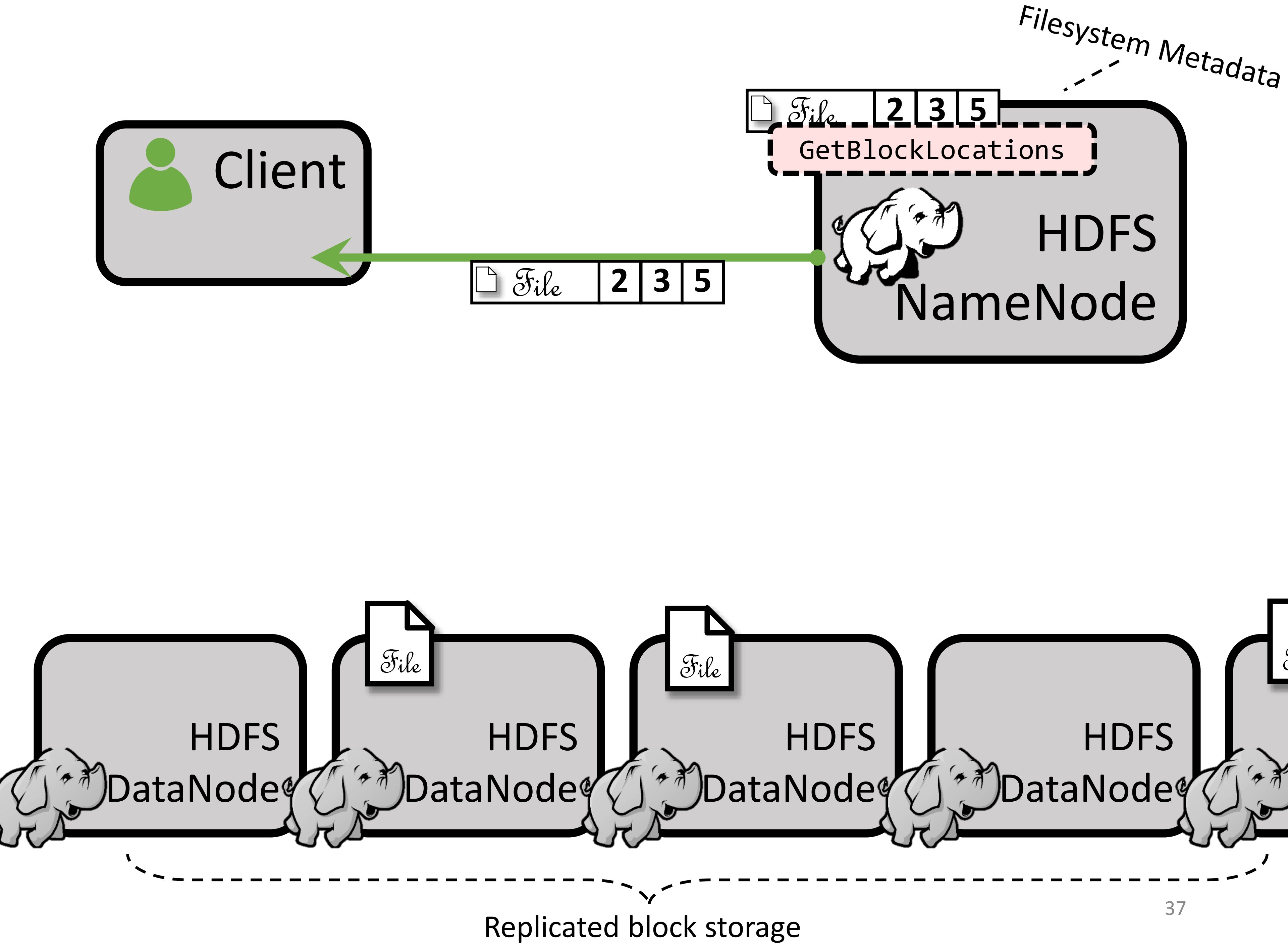
HDFS DataNode

HDFS DataNode

HDFS DataNode

HDFS DataNode

Replicated block storage



Filesystem Metadata

File 2 3 5

GetBlockLocations

Client

File 2 3 5

HDFS NameNode

File

File

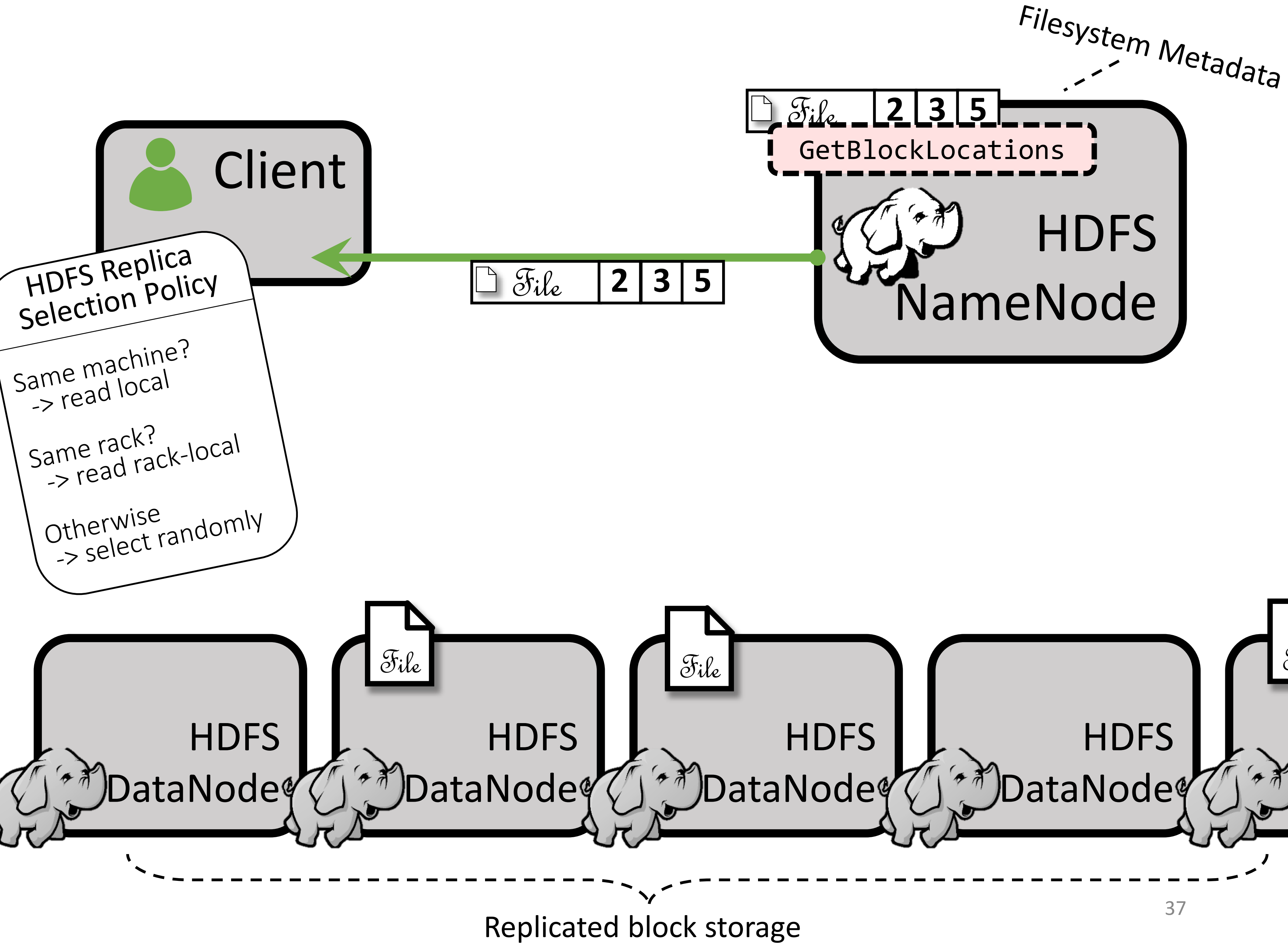
HDFS DataNode

HDFS DataNode

HDFS DataNode

HDFS DataNode

Replicated block storage

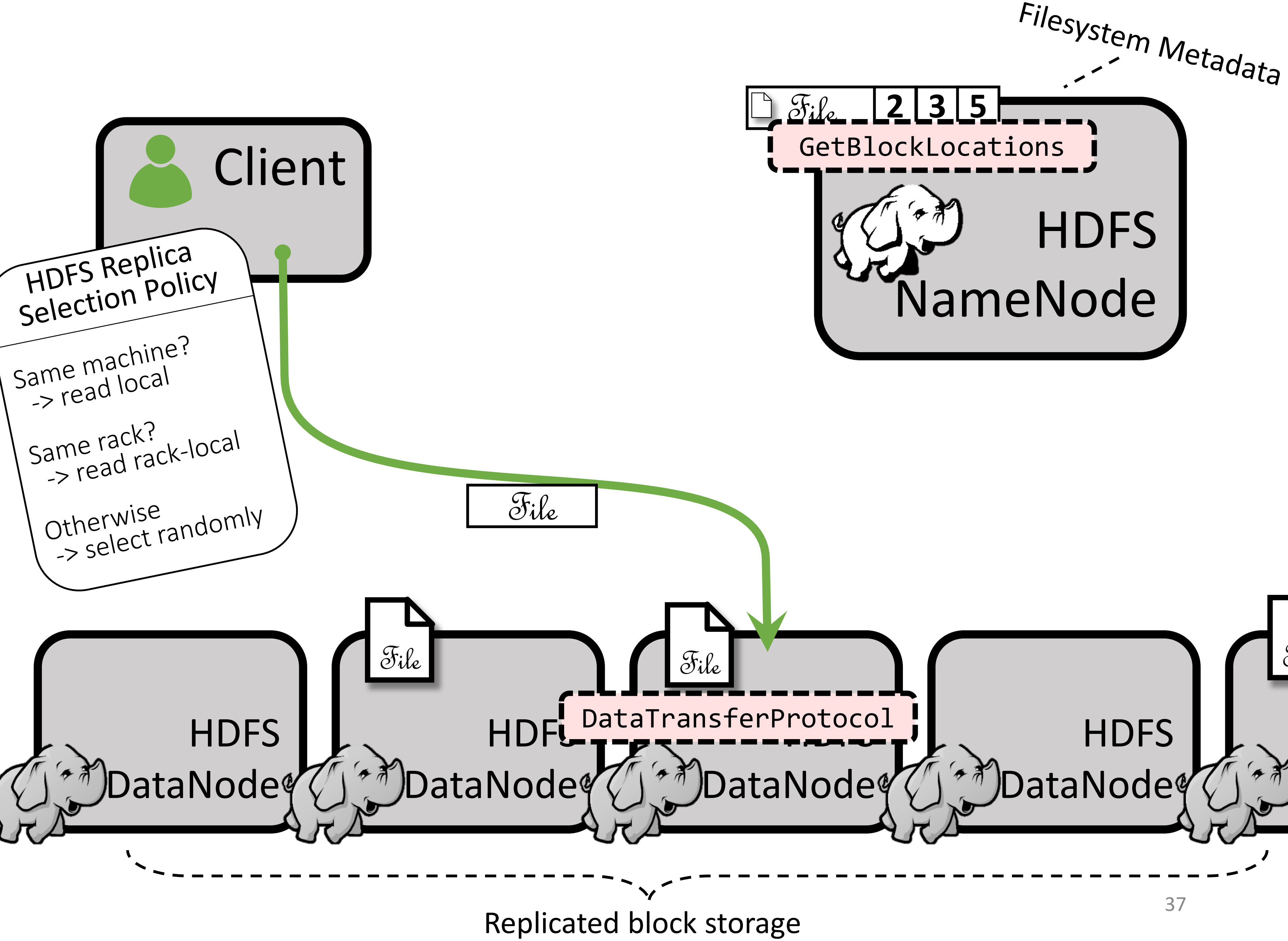


HDFS Replica Selection Policy

Same machine?
-> read local

Same rack?
-> read rack-local

Otherwise
-> select randomly



Client

HDFS Replica Selection Policy

Same machine?
-> read local

Same rack?
-> read rack-local

Otherwise
-> select randomly

Filesystem Metadata

File 2 3 5

GetBlockLocations

HDFS NameNode

File

HDFS DataNode

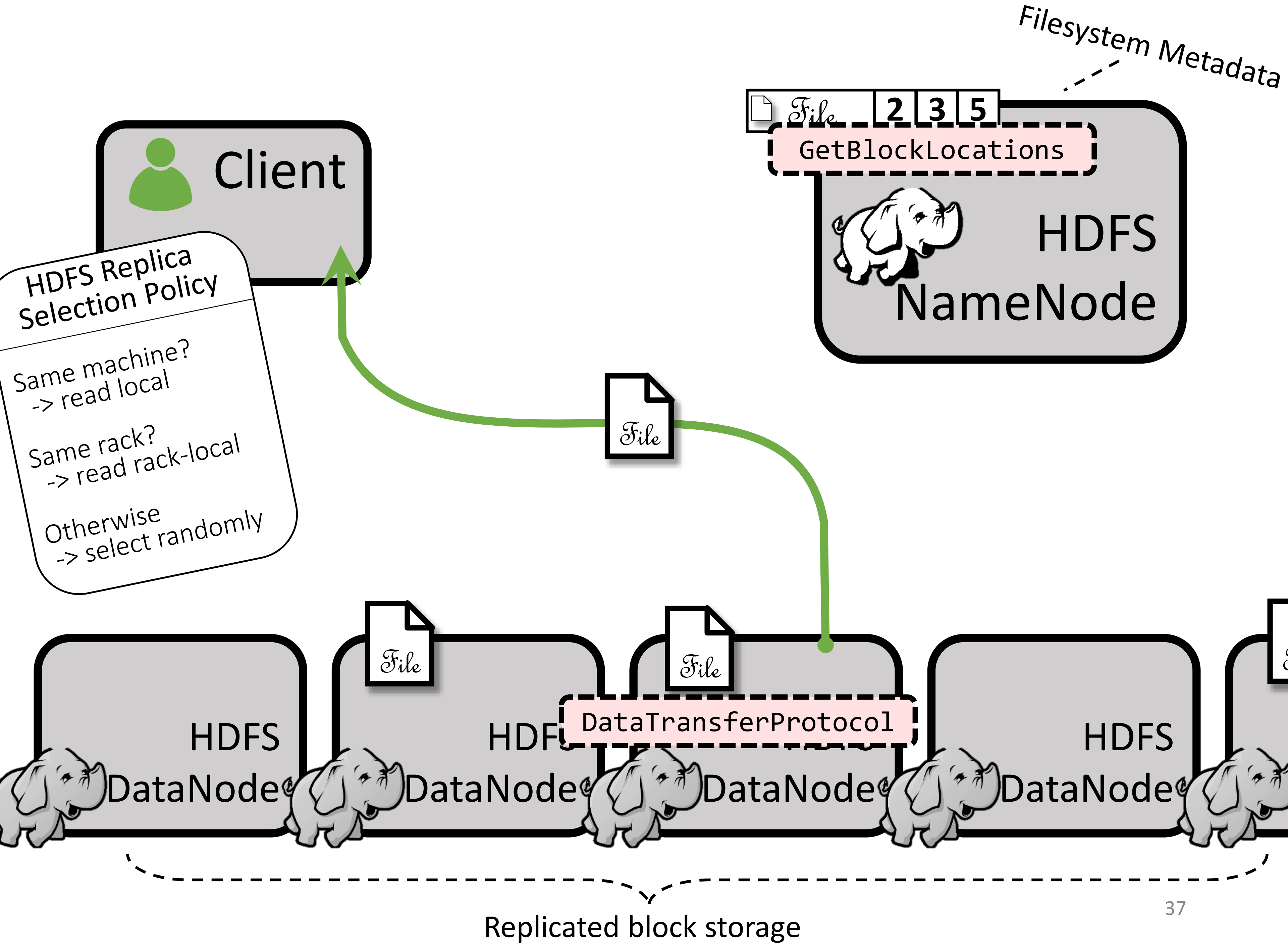
HDFS DataNode

HDFS DataNode

HDFS DataNode

DataTransferProtocol

Replicated block storage



Filesystem Metadata

File 2 3 5

GetBlockLocations

HDFS NameNode

Client

HDFS Replica Selection Policy

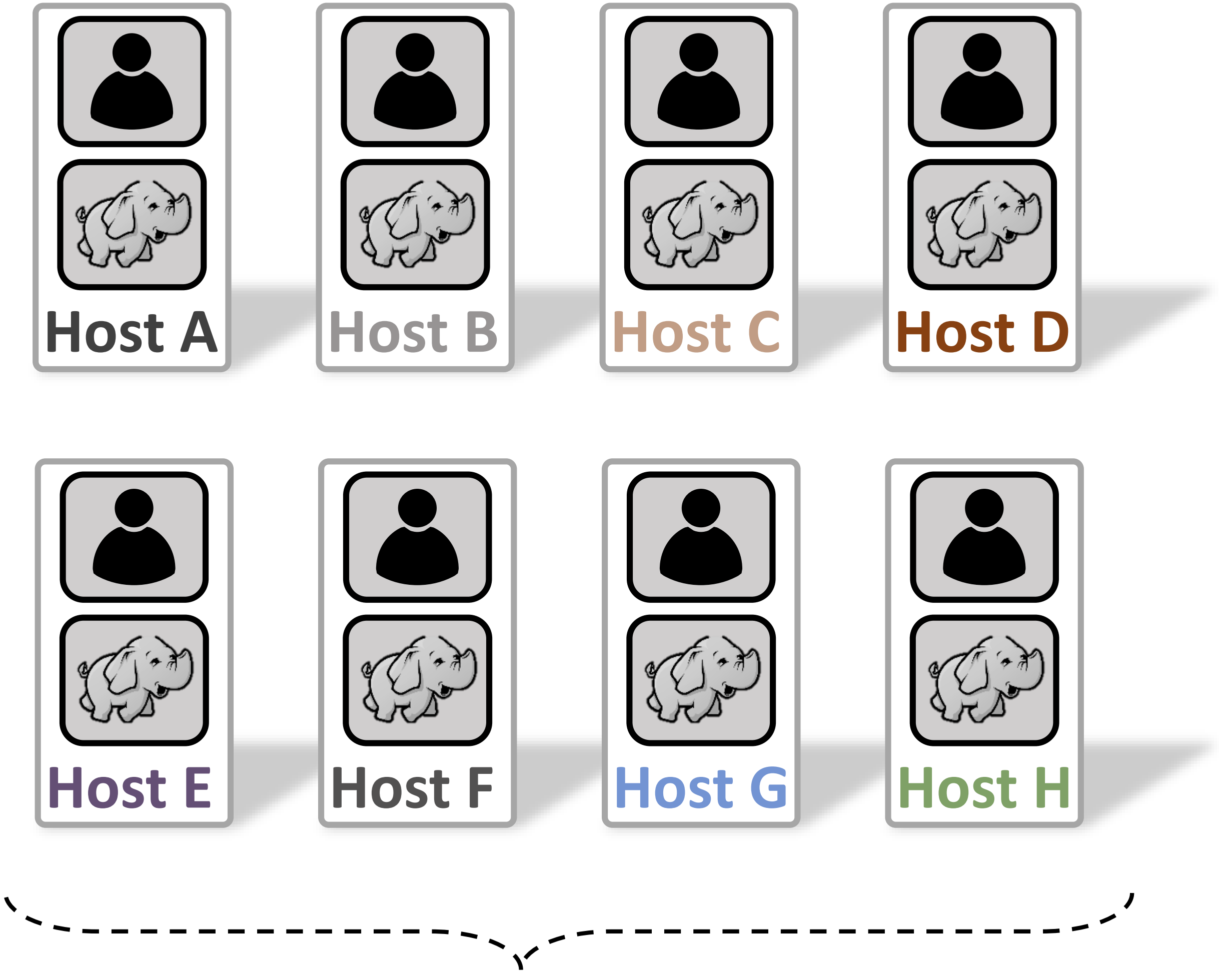
Same machine?
-> read local

Same rack?
-> read rack-local

Otherwise
-> select randomly

DataTransferProtocol

Replicated block storage

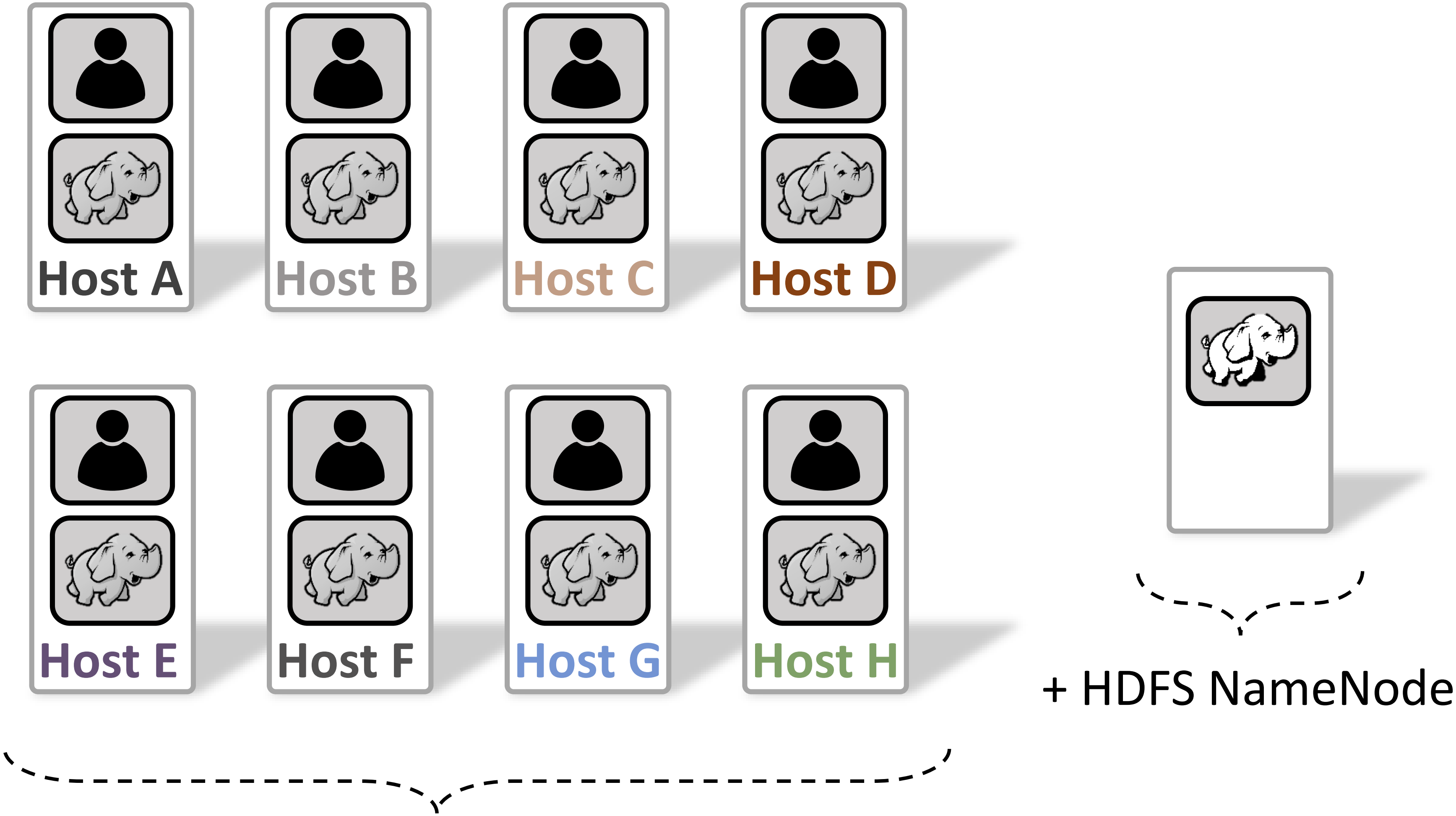


8 Worker Hosts

👤 Client Workload Generator

- Randomly read from large dataset

🐘 HDFS DataNode



+ HDFS NameNode

8 Worker Hosts

Client Workload Generator

- Randomly read from large dataset

HDFS DataNode

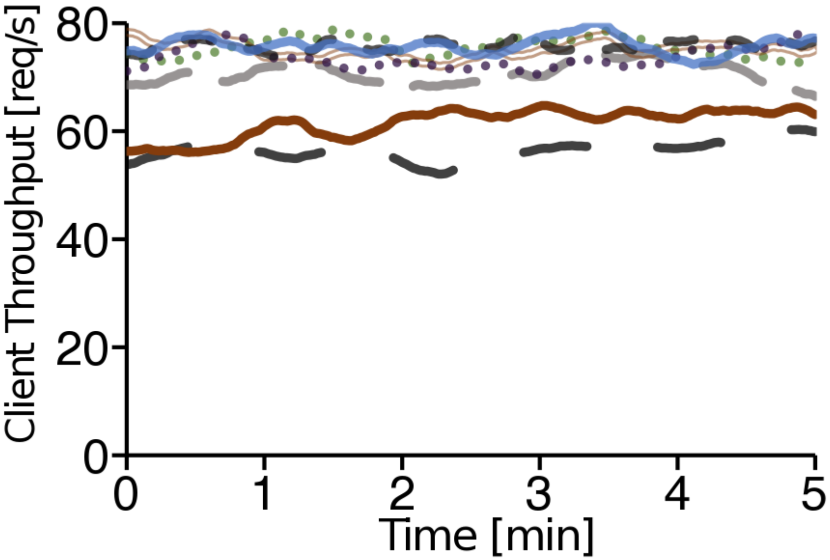
Same machines, same processes, same workloads

Same machines, same processes, same workloads

I expected uniform throughput from workload generators

Same machines, same processes, same workloads

I expected uniform throughput from workload generators

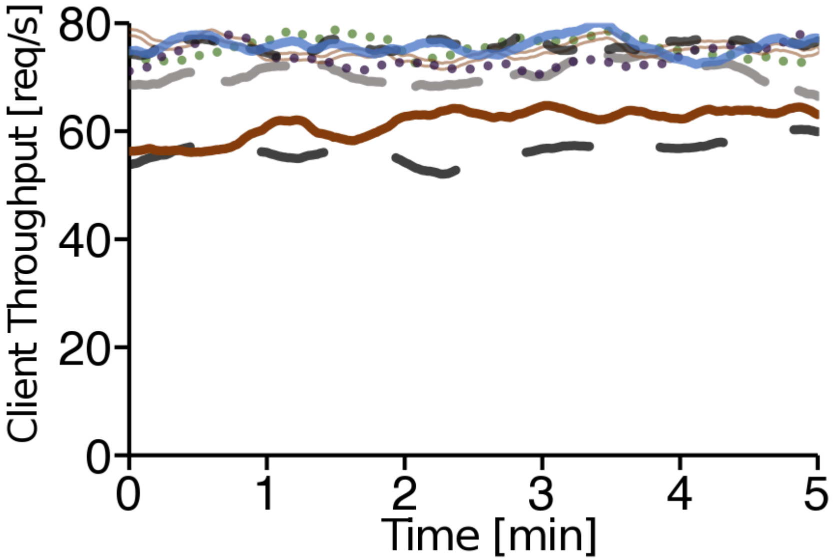


Consistently lower throughput

Same machines, same processes, same workloads

I expected uniform throughput from workload generators

I expected uniform throughput on DataNodes

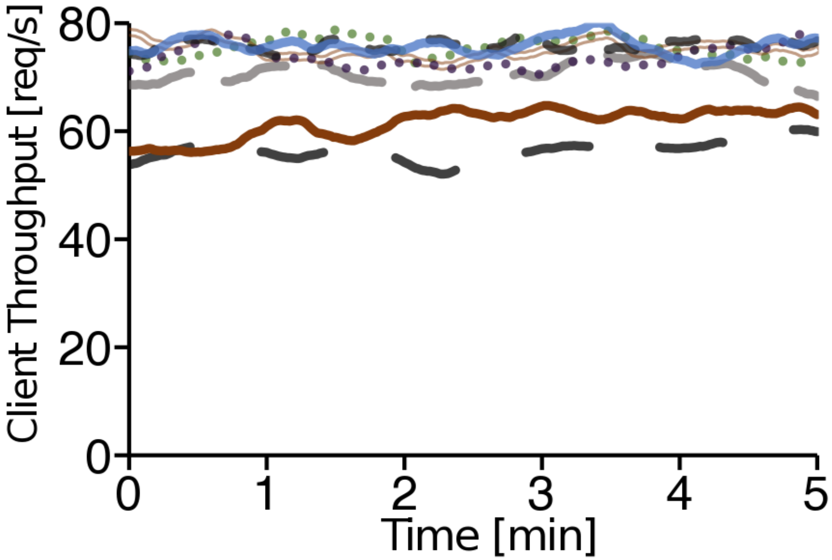


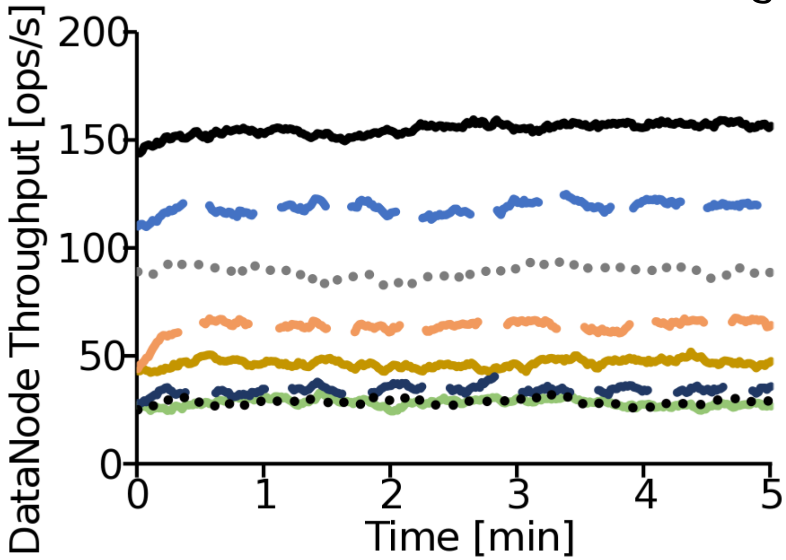
Consistently lower throughput

Same machines, same processes, same workloads

I expected uniform throughput from workload generators

I expected uniform throughput on DataNodes





Large variations in DataNode throughput

It’s probably a bug in the workload generator I wrote

My hypothesis:

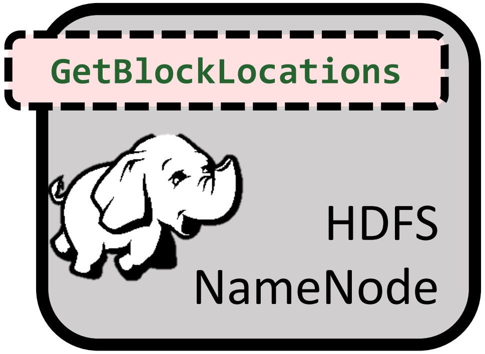
Workload generator is not randomly looking up files

It’s probably a bug in the workload generator I wrote

My hypothesis:

Workload generator is not randomly looking up files

```
From blockLocations In NameNode.GetBlockLocations
  GroupBy blockLocations.fileName
  Select blockLocations.fileName, COUNT
```

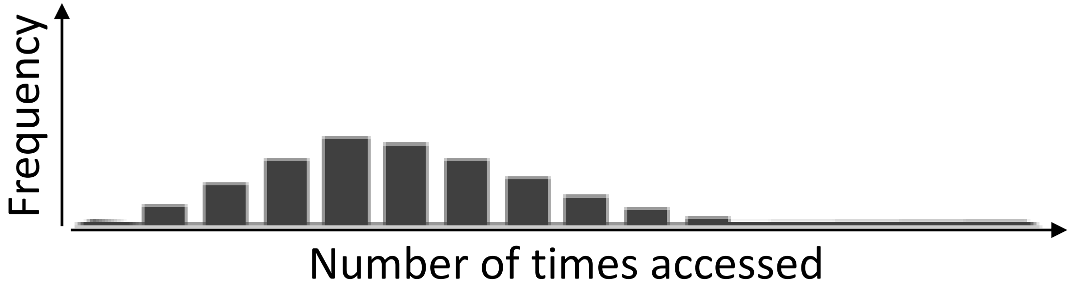


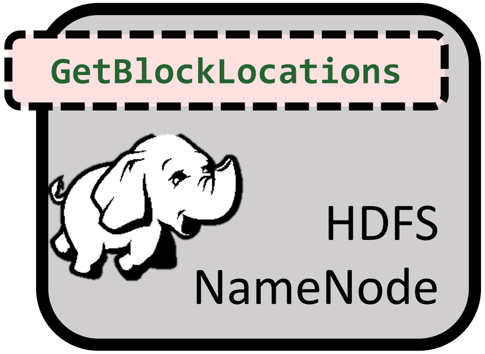
It’s probably a bug in the workload generator I wrote

My hypothesis:

Workload generator is not randomly looking up files

```
From blockLocations In NameNode.GetBlockLocations
  GroupBy blockLocations.fileName
  Select blockLocations.fileName, COUNT
```



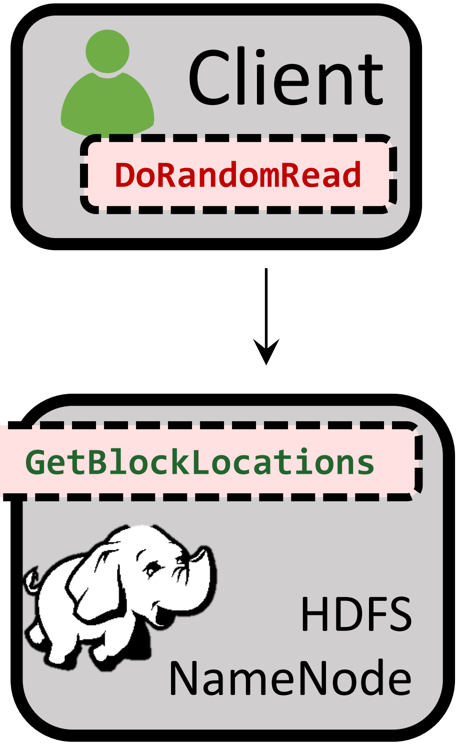


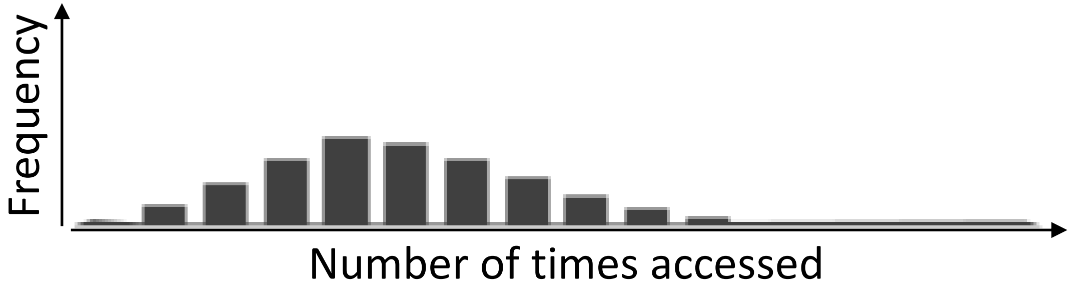
It’s probably a bug in the workload generator I wrote

My hypothesis:

Workload generator is not randomly looking up files

```
From blockLocations In NameNode.GetBlockLocations
  Join cl In Client.DoRandomRead On cl -> blockLocations
  GroupBy cl.host, blockLocations.fileName
  Select cl.host, blockLocations.fileName, COUNT
```



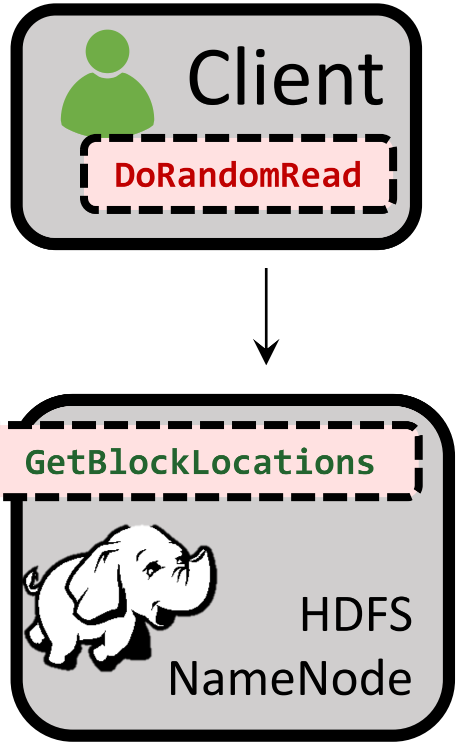


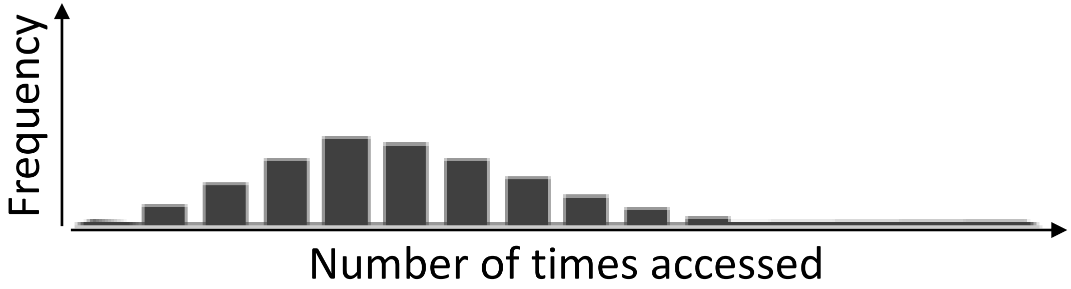
It’s probably a bug in the workload generator I wrote

My hypothesis:

Workload generator is not randomly looking up files

```
From blockLocations In NameNode.GetBlockLocations
  Join cl In Client.DoRandomRead On cl -> blockLocations
  GroupBy cl.host, blockLocations.fileName
  Select cl.host, blockLocations.fileName, COUNT
```



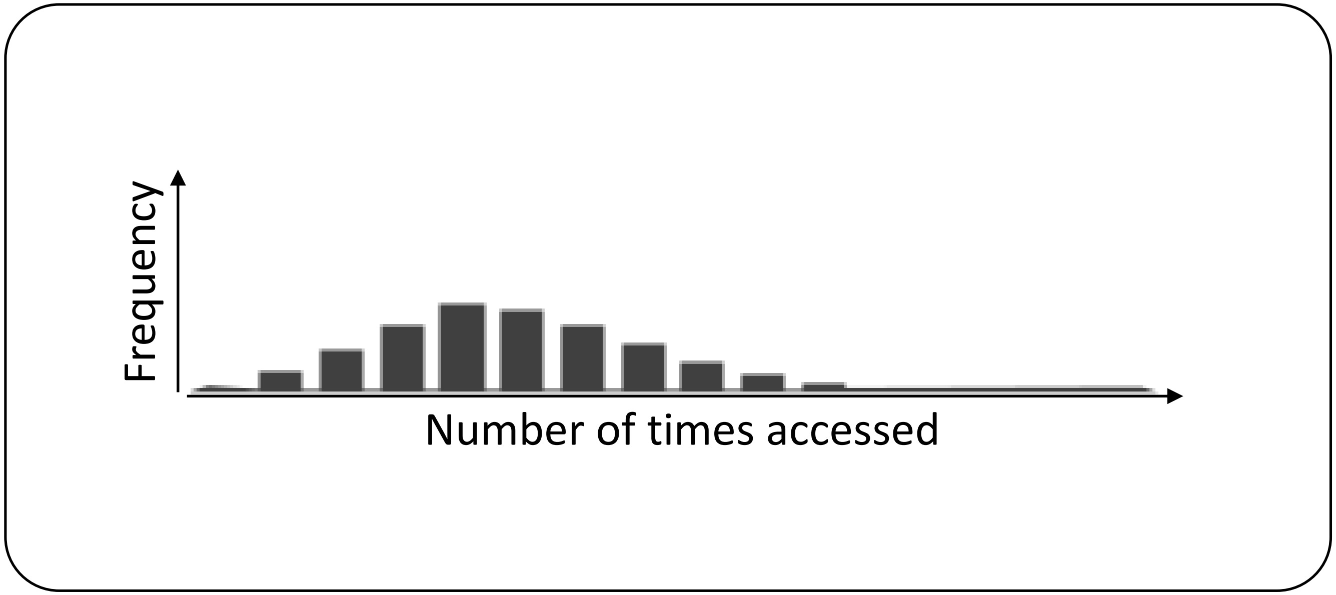


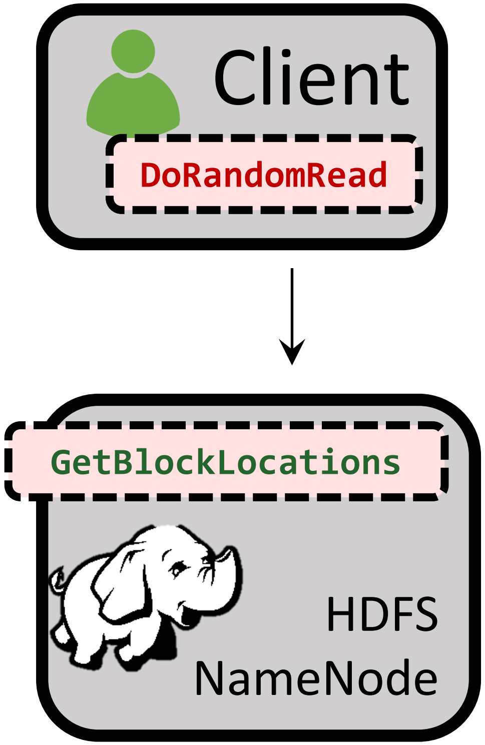
It's probably a bug in the workload generator I wrote

My hypothesis:

Workload generator is not randomly looking up files

```
From blockLocations In NameNode.GetBlockLocations
  Join cl In Client.DoRandomRead On cl -> blockLocations
  GroupBy cl.host, blockLocations.fileName
  Select cl.host, blockLocations.fileName, COUNT
```



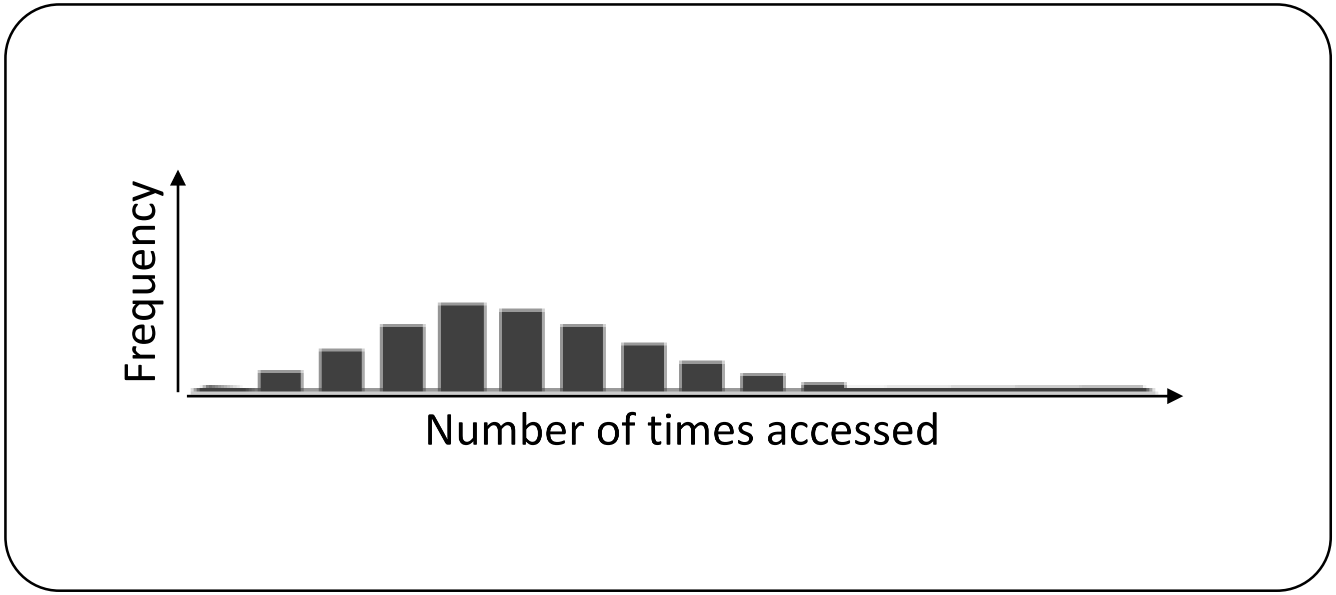


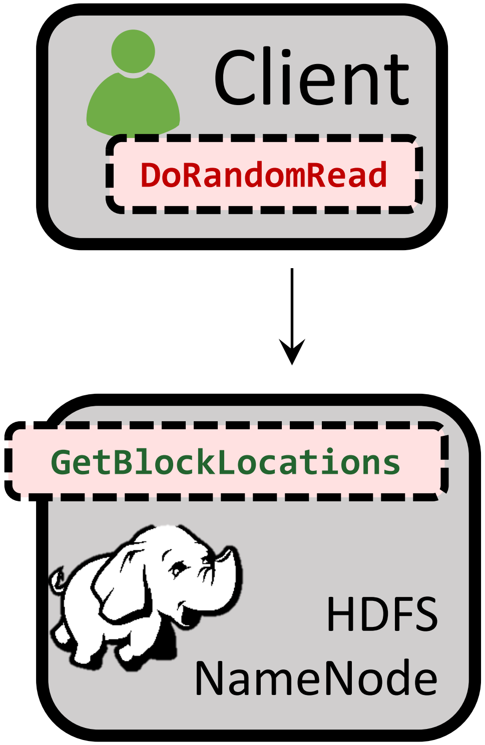
It’s probably a bug in the workload generator I wrote

My hypothesis:

Workload generator is not randomly looking up files

```
From blockLocations In NameNode.GetBlockLocations
  Join cl In Client.DoRandomRead On cl -> blockLocations
  GroupBy cl.host, blockLocations.fileName
  Select cl.host, blockLocations.fileName, COUNT
```



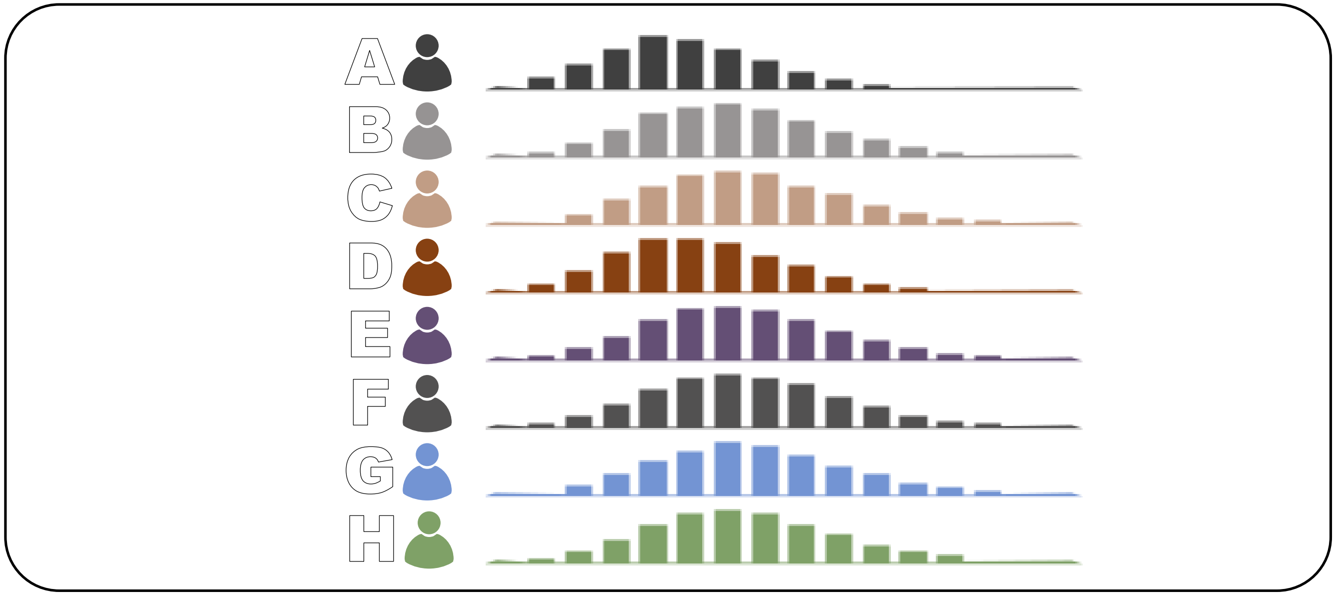


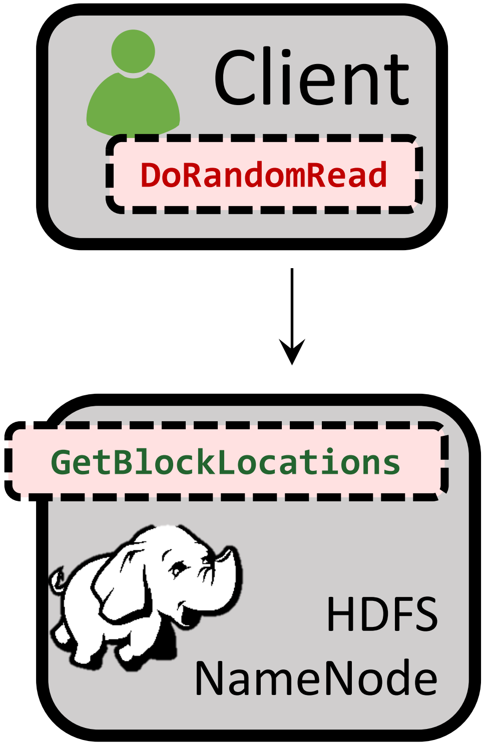
It’s probably a bug in the workload generator I wrote

My hypothesis:

Workload generator is not randomly looking up files

```
From blockLocations In NameNode.GetBlockLocations
  Join cl In Client.DoRandomRead On cl -> blockLocations
  GroupBy cl.host, blockLocations.fileName
  Select cl.host, blockLocations.fileName, COUNT
```





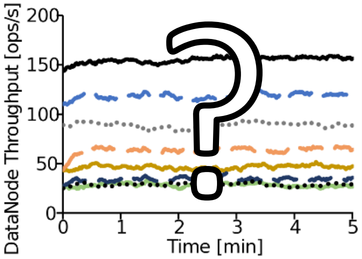
Client

DoRandomRead

GetBlockLocations

HDFS NameNode

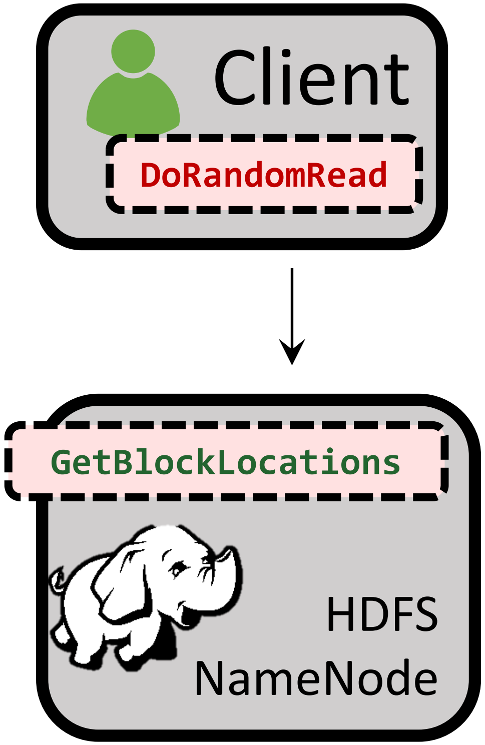
It’s probably a bug in the workload generator I wrote

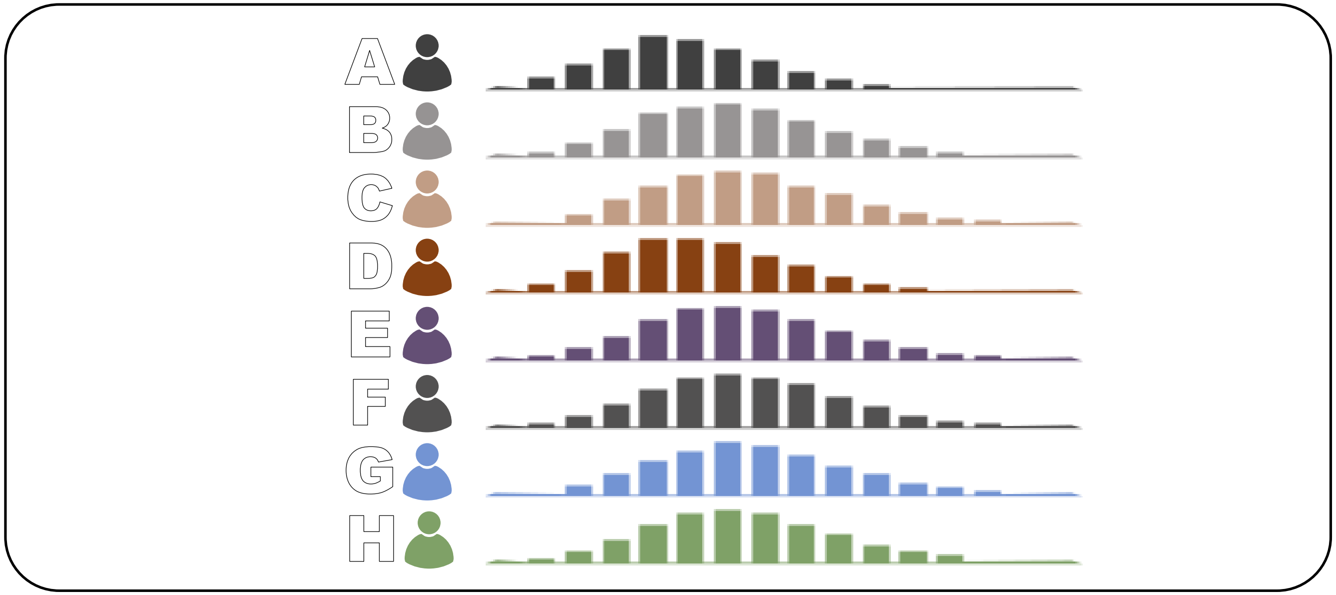


My hypothesis:

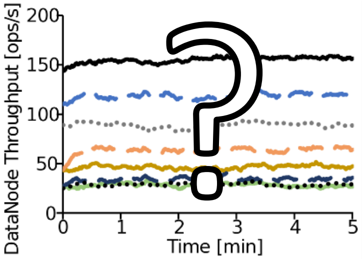
Workload generator is not randomly looking up files

```
From blockLocations In NameNode.GetBlockLocations
  Join cl In Client.DoRandomRead On cl -> blockLocations
  GroupBy cl.host, blockLocations.fileName
  Select cl.host, blockLocations.fileName, COUNT
```

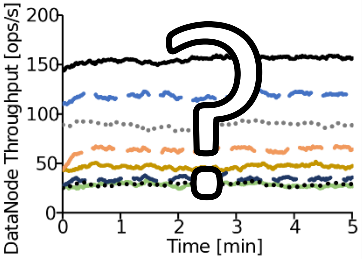




Maybe skewed DataNode throughput is because some DataNodes store more files than others

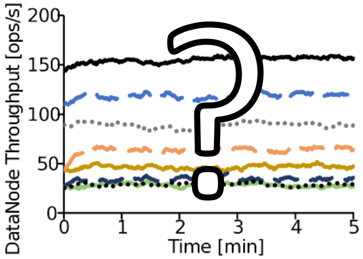


Maybe skewed DataNode throughput is because some DataNodes store more files than others



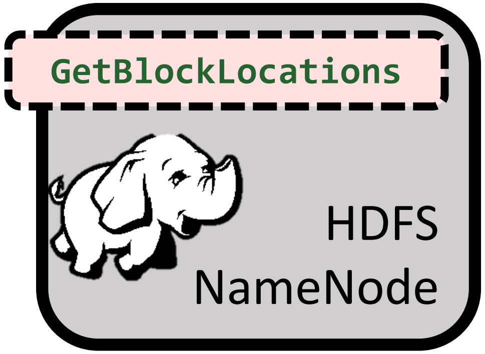
How often was each DataNode a replica host?

Maybe skewed DataNode throughput is because some DataNodes store more files than others



How often was each DataNode a replica host?

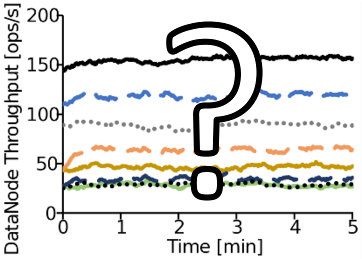
```
From blockLocations In NameNode.GetBlockLocations
  GroupBy blockLocations.replicas
  Select blockLocations.replicas, COUNT
```



GetBlockLocations

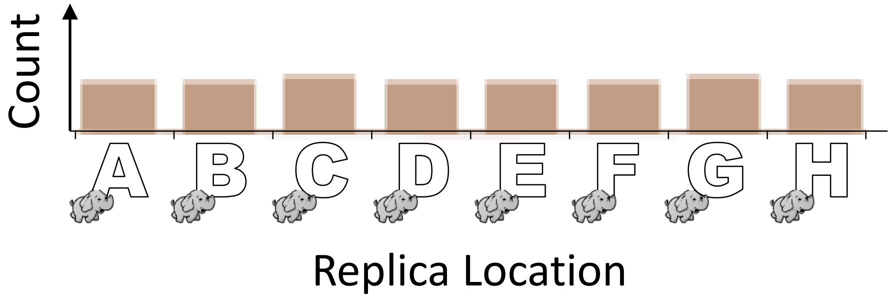
HDFS NameNode

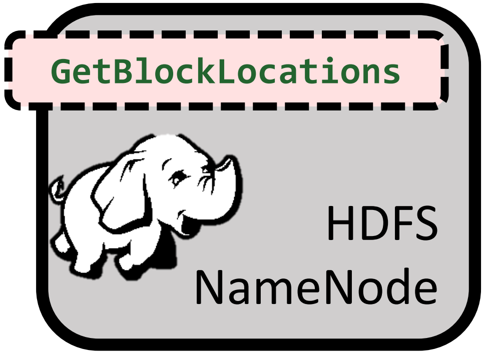
Maybe skewed DataNode throughput is because some DataNodes store more files than others



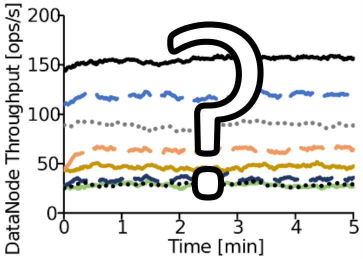
How often was each DataNode a replica host?

```
From blockLocations In NameNode.GetBlockLocations
  GroupBy blockLocations.replicas
  Select blockLocations.replicas, COUNT
```



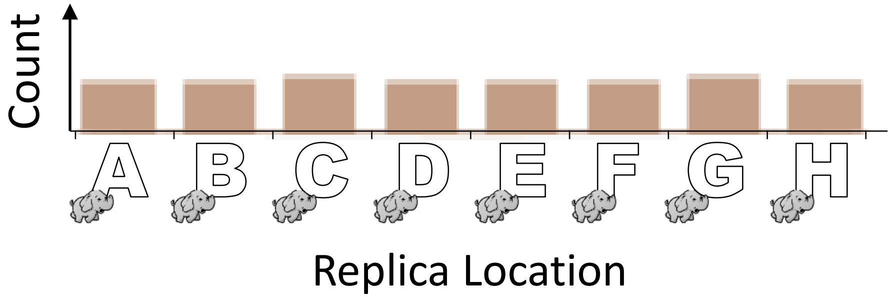


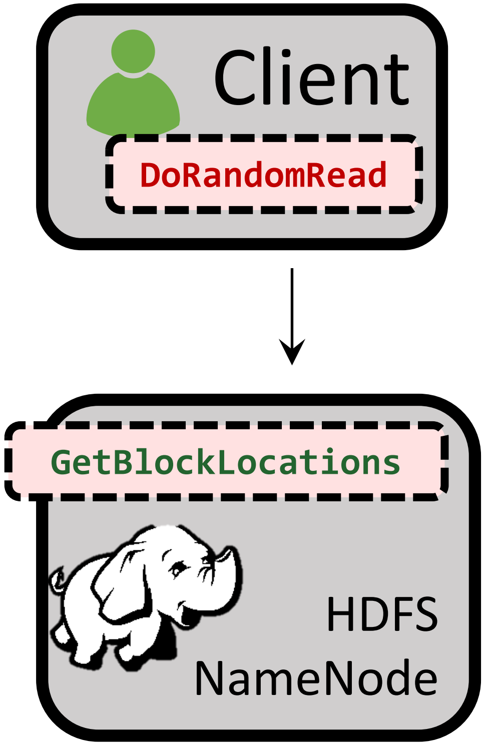
Maybe skewed DataNode throughput is because some DataNodes store more files than others



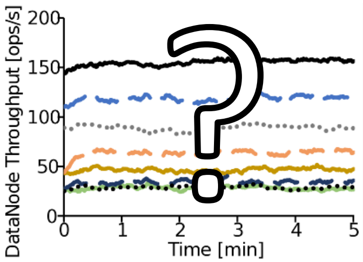
How often was each DataNode a replica host?

```
From blockLocations In NameNode.GetBlockLocations
  Join cl In Client.DoRandomRead On cl -> blockLocations
  GroupBy cl.host, blockLocations.replicas
  Select cl.host, blockLocations.replicas, COUNT
```



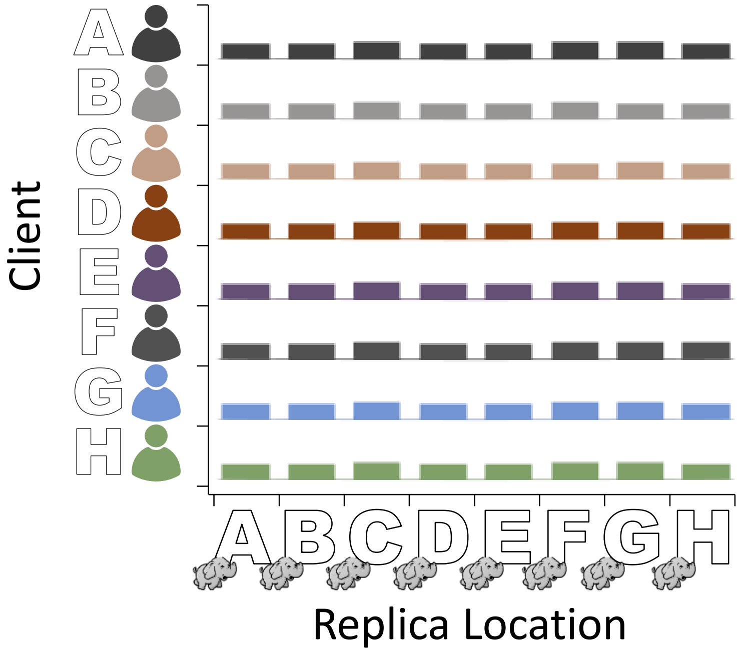


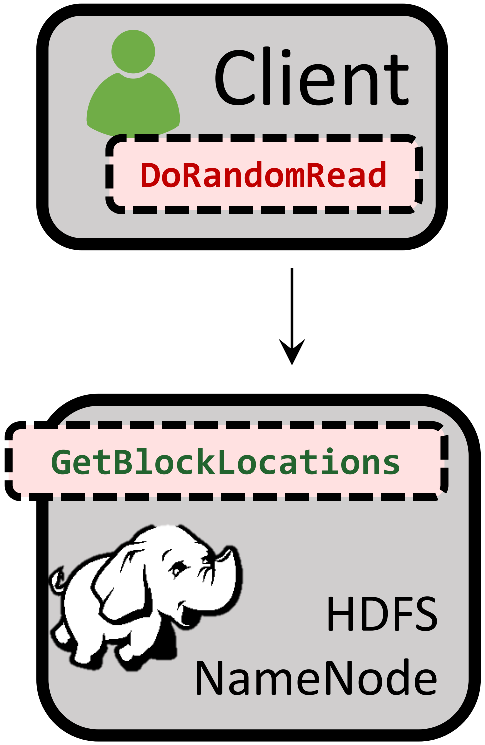
Maybe skewed DataNode throughput is because some DataNodes store more files than others



How often was each DataNode a replica host?

```
From blockLocations In NameNode.GetBlockLocations
  Join cl In Client.DoRandomRead On cl -> blockLocations
  GroupBy cl.host, blockLocations.replicas
  Select cl.host, blockLocations.replicas, COUNT
```





Client

DoRandomRead

GetBlockLocations

HDFS NameNode

Conclusions so far:

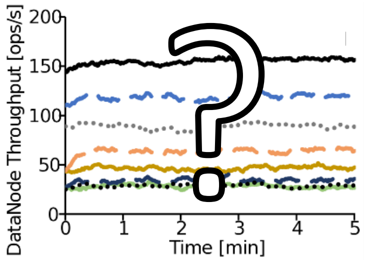
Clients are selecting files uniformly at random

Files are distributed across DNs uniformly at random

Conclusions so far:

Clients are selecting files uniformly at random

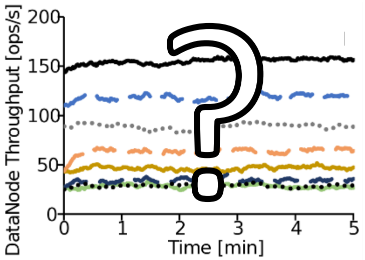
Files are distributed across DNs uniformly at random



Conclusions so far:

Clients are selecting files uniformly at random

Files are distributed across DNs uniformly at random

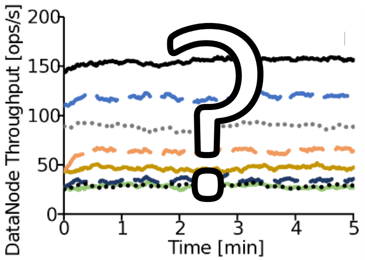


Hypothesis: choice of replica isn’t random?

Conclusions so far:

Clients are selecting files uniformly at random

Files are distributed across DNs uniformly at random



Hypothesis: choice of replica isn’t random?

HDFS Replica Selection Policy

Same machine?
-> read local

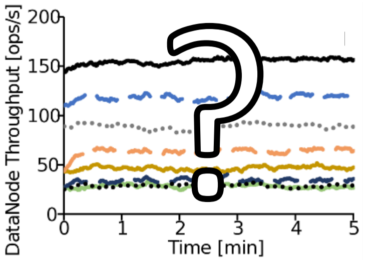
Same rack?
-> read rack-local

Otherwise
-> select randomly

Conclusions so far:

Clients are selecting files uniformly at random

Files are distributed across DNs uniformly at random



Hypothesis: choice of replica isn’t random?

When a file is read from a DataNode, where else *could* it have been read from?

HDFS Replica Selection Policy

Same machine?
-> read local

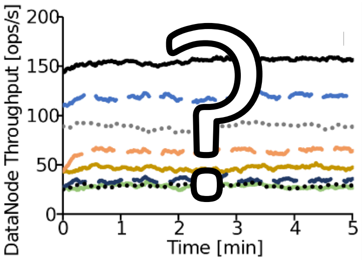
Same rack?
-> read rack-local

Otherwise
-> select randomly

Conclusions so far:

Clients are selecting files uniformly at random

Files are distributed across DNs uniformly at random



Hypothesis: choice of replica isn’t random?

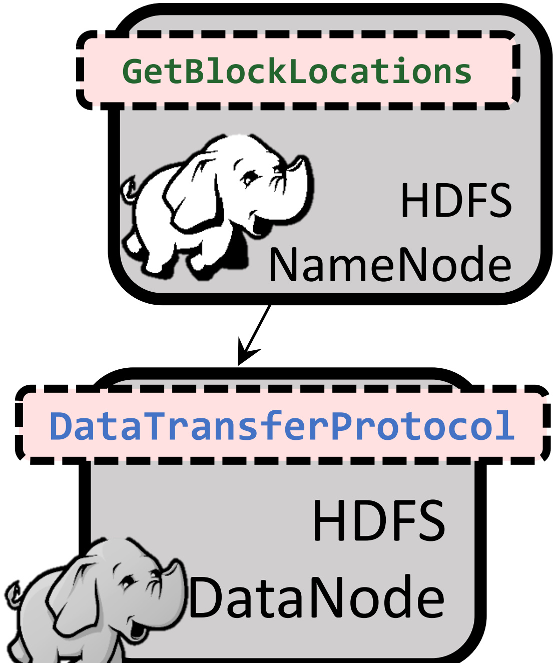
HDFS Replica Selection Policy

Same machine?
-> read local

Same rack?
-> read rack-local

Otherwise
-> select randomly

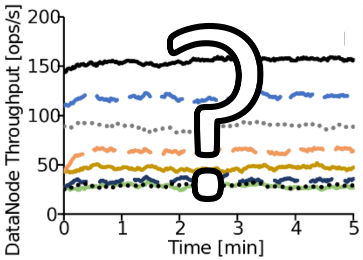
When a file is read from a DataNode,
where else *could* it have been read from?



Conclusions so far:

Clients are selecting files uniformly at random

Files are distributed across DNs uniformly at random



Hypothesis: choice of replica isn’t random?

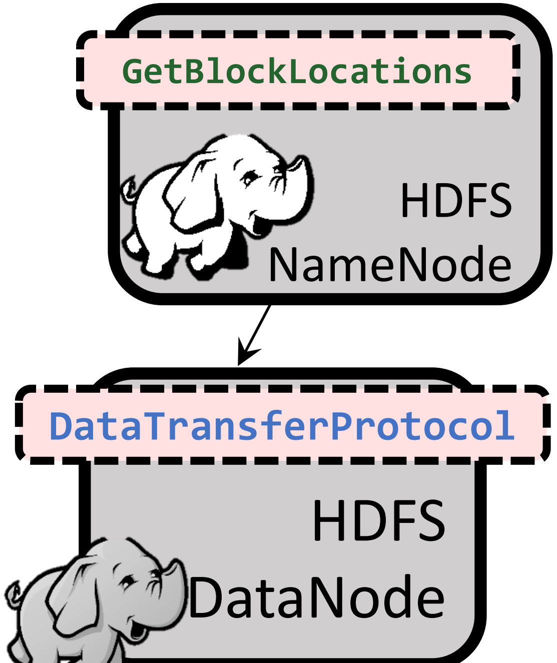
HDFS Replica Selection Policy

Same machine?
-> read local

Same rack?
-> read rack-local

Otherwise
-> select randomly

When a file is read from a DataNode,
where else *could* it have been read from?



GetBlockLocations

HDFS NameNode

DataTransferProtocol

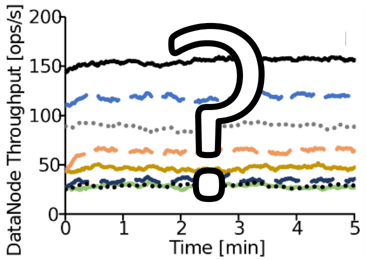
HDFS DataNode

```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
                   On blockLocations -> readBlock
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```

Conclusions so far:

Clients are selecting files uniformly at random

Files are distributed across DNs uniformly at random



Hypothesis: choice of replica isn't random?

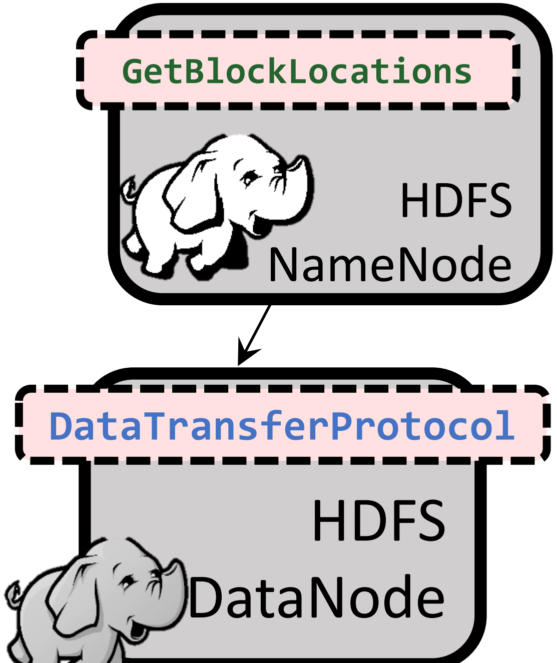
HDFS Replica Selection Policy

~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

Otherwise
-> select randomly

When a file is read from a DataNode,
where else *could* it have been read from?

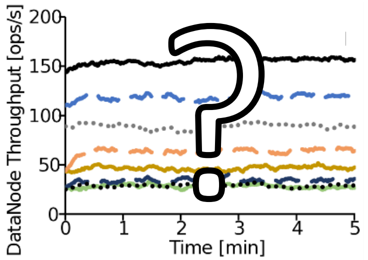


```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
                 On blockLocations -> readBlock
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```

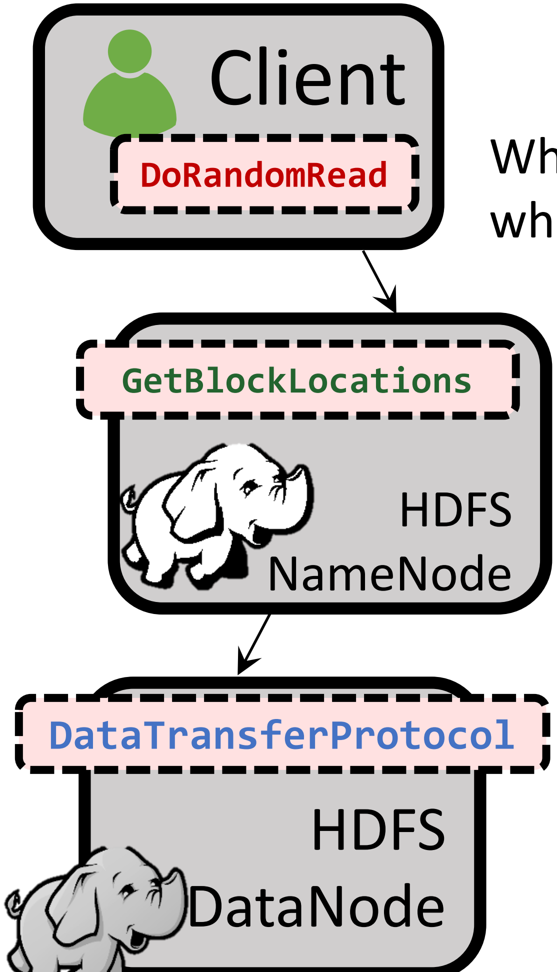
Conclusions so far:

Clients are selecting files uniformly at random

Files are distributed across DNs uniformly at random



Hypothesis: choice of replica isn’t random?


Client

DoRandomRead

GetBlockLocations

HDFS NameNode

DataTransferProtocol

HDFS DataNode

When a file is read from a DataNode, where else *could* it have been read from?

HDFS Replica Selection Policy

~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

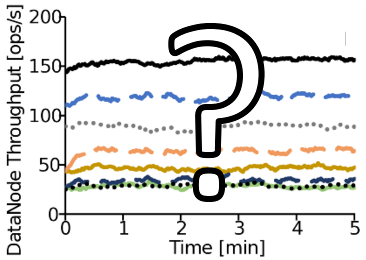
Otherwise
-> select randomly

```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
                      On blockLocations -> readBlock
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```

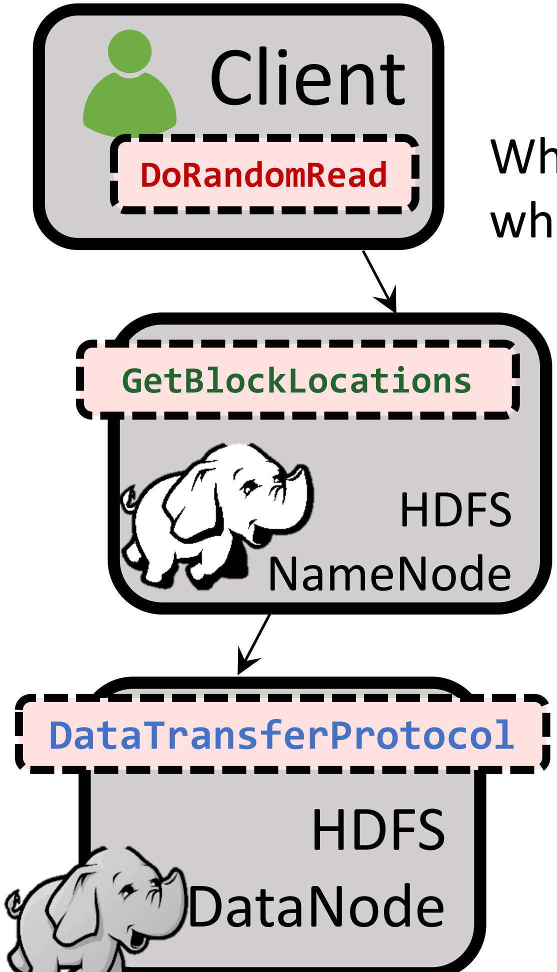
Conclusions so far:

Clients are selecting files uniformly at random

Files are distributed across DNs uniformly at random



Hypothesis: choice of replica isn't random?



When a file is read from a DataNode, where else *could* it have been read from?

HDFS Replica Selection Policy

~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

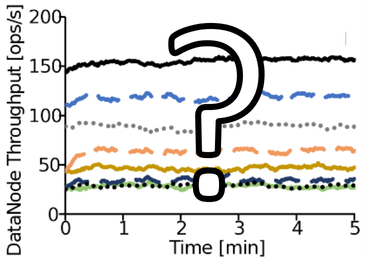
Otherwise
-> select randomly

```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
                     On blockLocations -> readBlock
  Join cl In Client.DoRandomRead
                     On cl -> blockLocations
  Where cl.host != readBlock.host
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```

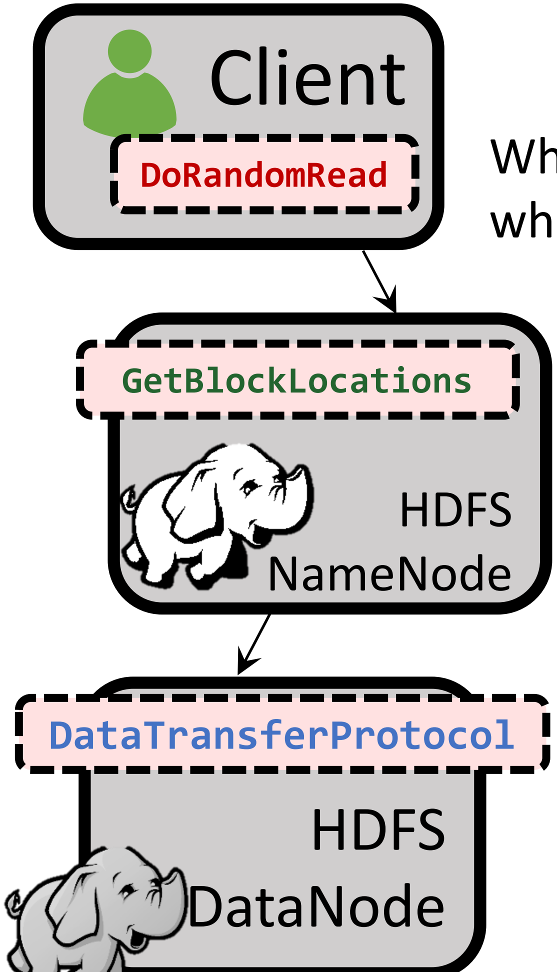
Conclusions so far:

Clients are selecting files uniformly at random

Files are distributed across DNs uniformly at random



Hypothesis: choice of replica isn't random?



When a file is read from a DataNode, where else *could* it have been read from?

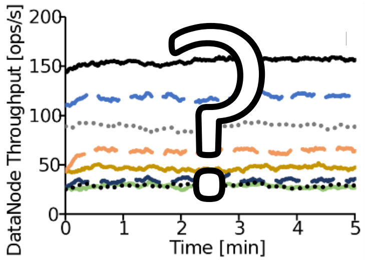
HDFS Replica Selection Policy

~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

Otherwise
-> select randomly

```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
                  On blockLocations -> readBlock
  Join cl In Client.DoRandomRead
                  On cl -> blockLocations
  Where cl.host != readBlock.host
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```

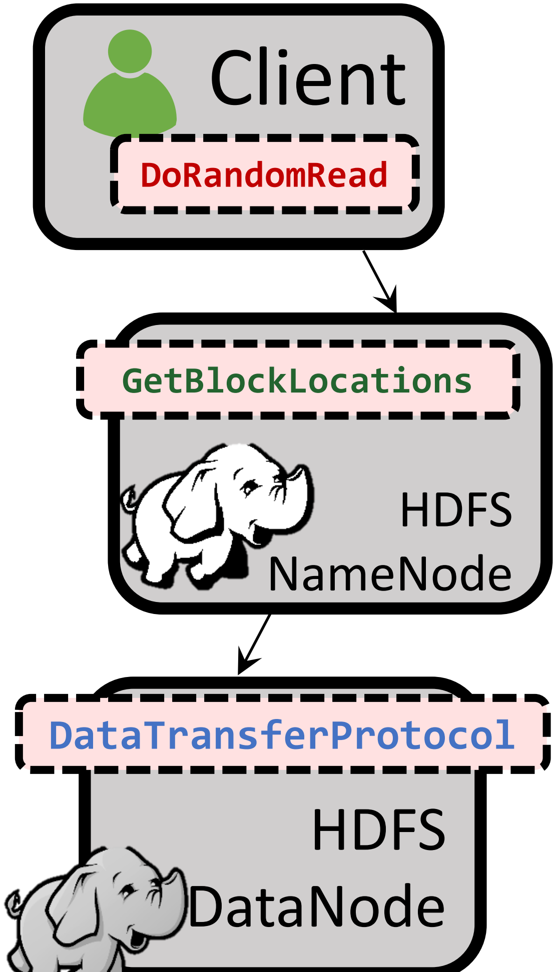


HDFS Replica Selection Policy

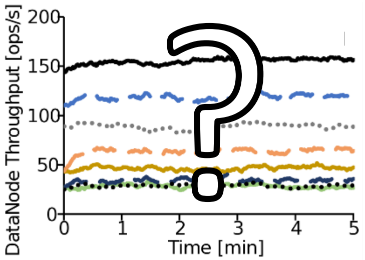
~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

Otherwise
-> select randomly



```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
                  On blockLocations -> readBlock
  Join cl In Client.DoRandomRead
                  On cl -> blockLocations
  Where cl.host != readBlock.host
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```



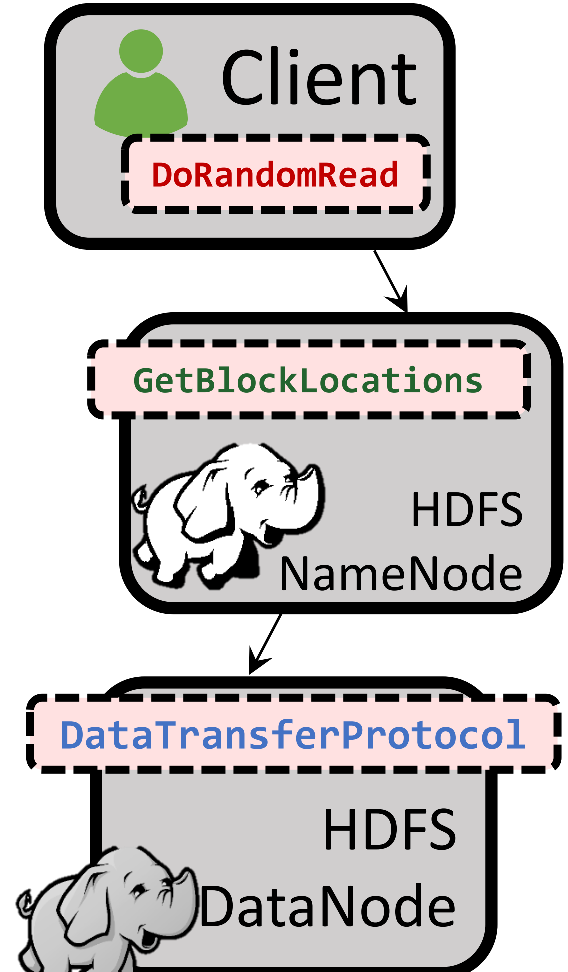
When both B and D host replicas,
Clients choose B this often: ▬ (~50%)
Clients choose D this often: ▬ (~50%)

HDFS Replica Selection Policy

~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

Otherwise
-> select randomly



Client

DoRandomRead

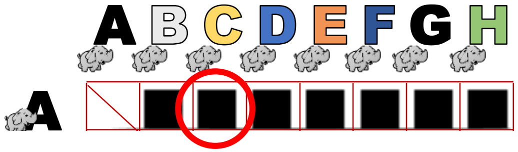
GetBlockLocations

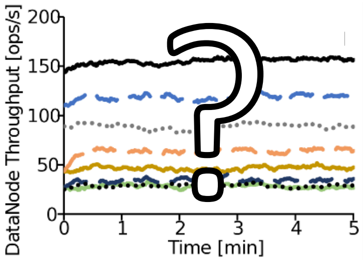
HDFS NameNode

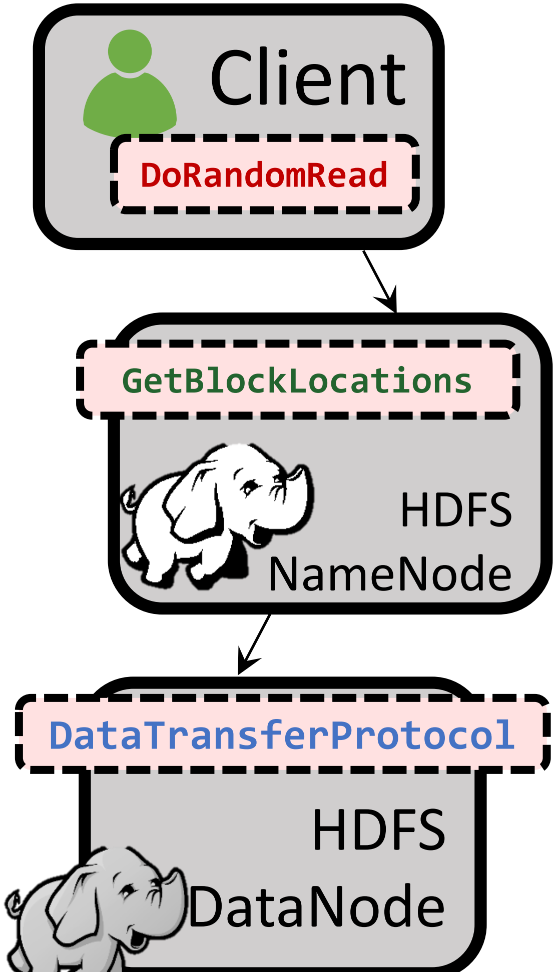
DataTransferProtocol

HDFS DataNode

```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
                On blockLocations -> readBlock
  Join cl In Client.DoRandomRead
                On cl -> blockLocations
  Where cl.host != readBlock.host
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```






Client

DoRandomRead

GetBlockLocations

HDFS NameNode

DataTransferProtocol

HDFS DataNode

When both A and C host replicas,
Clients choose A this often: ■ (100%)

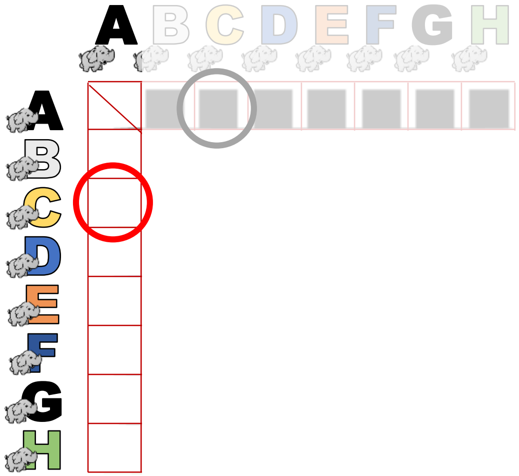
HDFS Replica Selection Policy

~~Same machine?~~
~~-> read local~~

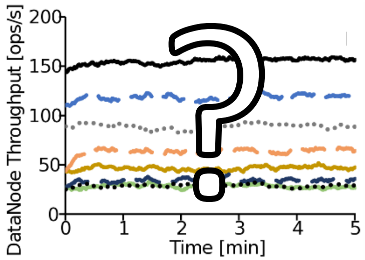
Same rack?
-> read rack-local

Otherwise
-> select randomly

```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
               On blockLocations -> readBlock
  Join cl In Client.DoRandomRead
               On cl -> blockLocations
  Where cl.host != readBlock.host
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```



When both A and C host replicas,
Clients choose A this often: ■ (100%)
Clients choose C this often: (0%)

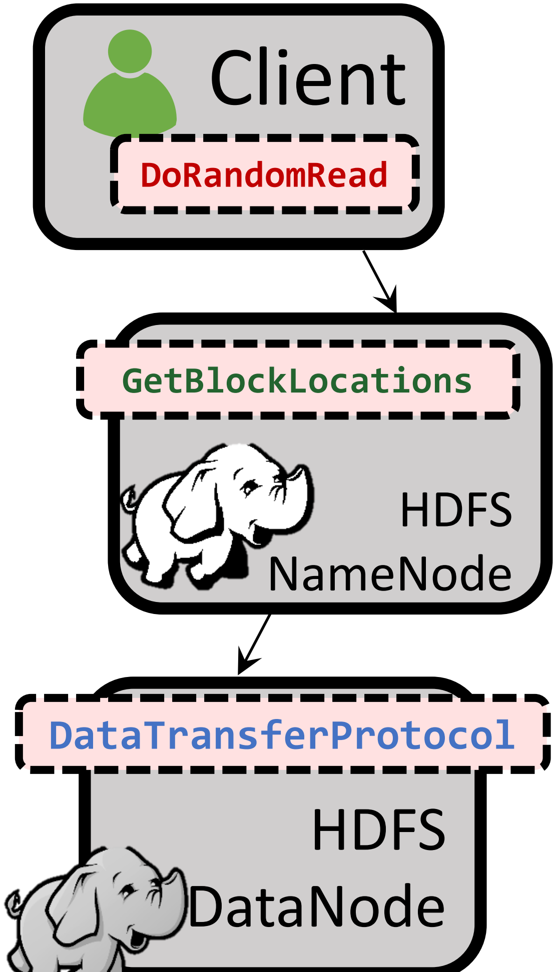


HDFS Replica Selection Policy

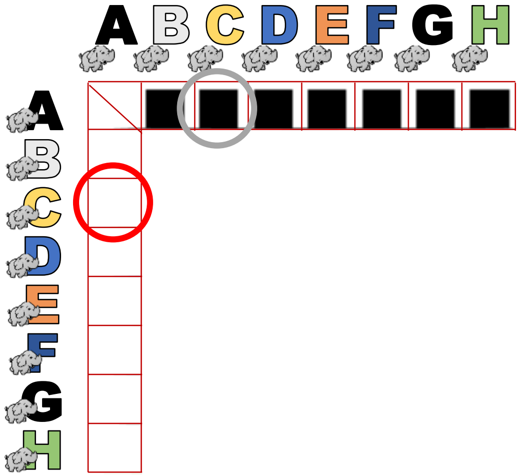
~~Same machine?~~
~~-> read local~~

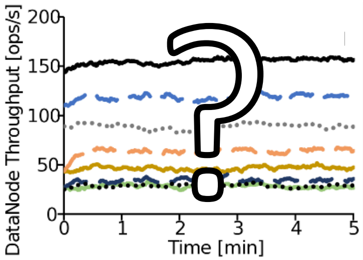
Same rack?
-> read rack-local

Otherwise
-> select randomly



```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
                 On blockLocations -> readBlock
  Join cl In Client.DoRandomRead
                 On cl -> blockLocations
  Where cl.host != readBlock.host
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```





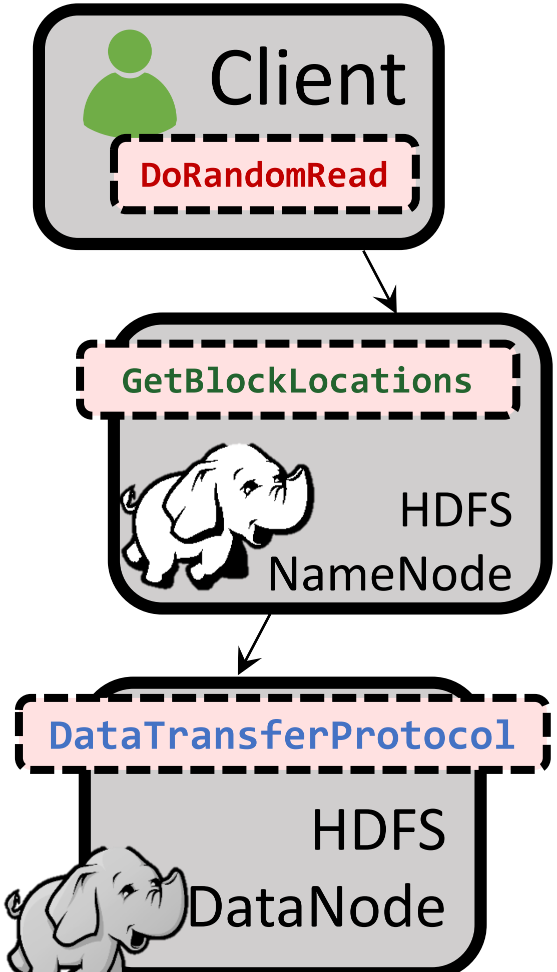
When both A and C host replicas,
Clients choose A this often: ■ (100%)
Clients choose C this often: (0%)

HDFS Replica Selection Policy

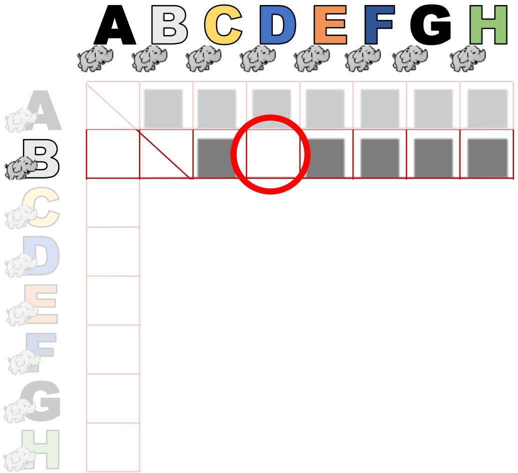
~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

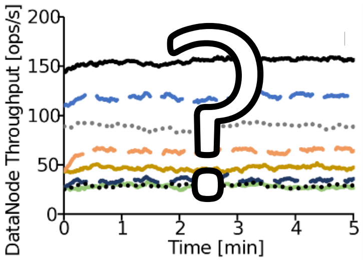
Otherwise
-> select randomly



```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
                 On blockLocations -> readBlock
  Join cl In Client.DoRandomRead
                 On cl -> blockLocations
  Where cl.host != readBlock.host
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```



When both B and D host replicas,
Clients choose B this often: ___ (0%)

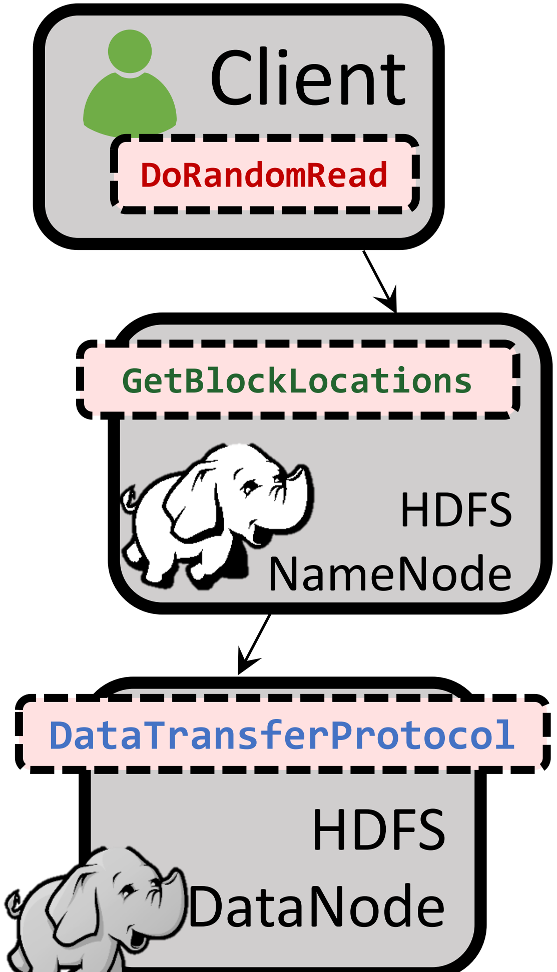


HDFS Replica Selection Policy

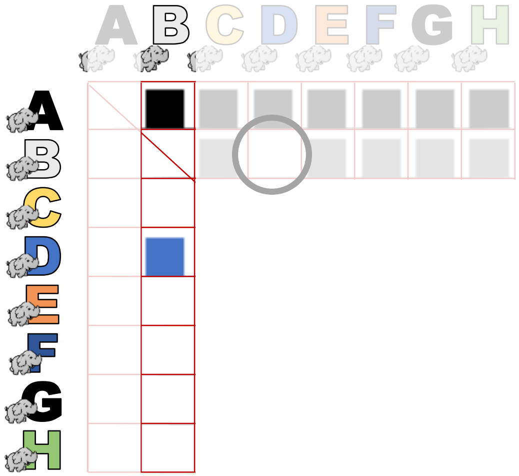
~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

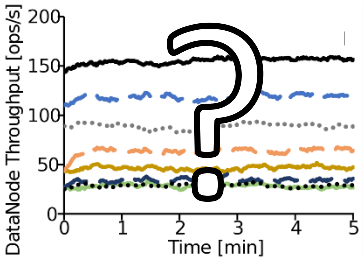
Otherwise
-> select randomly



```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
               On blockLocations -> readBlock
  Join cl In Client.DoRandomRead
               On cl -> blockLocations
  Where cl.host != readBlock.host
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```



When both B and D host replicas,
Clients choose B this often: ___ (0%)

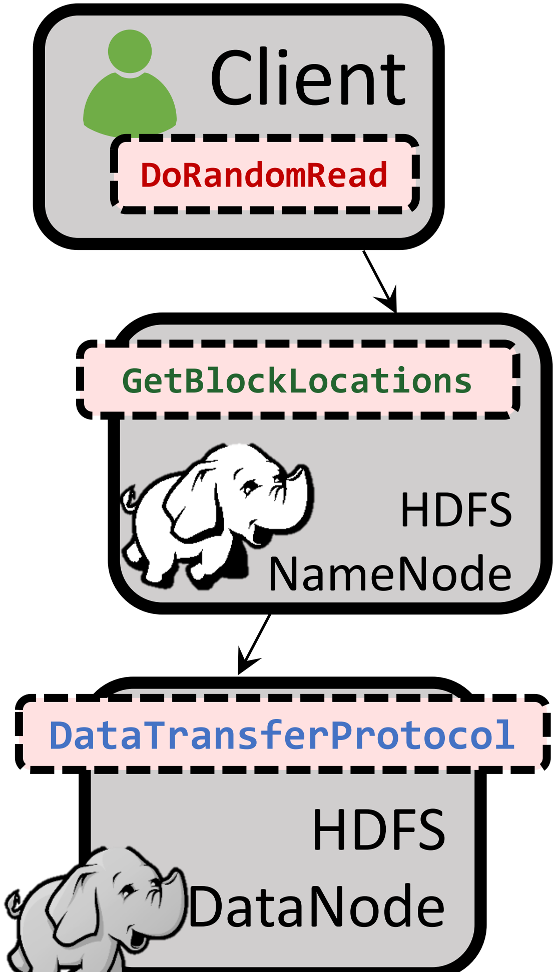


HDFS Replica Selection Policy

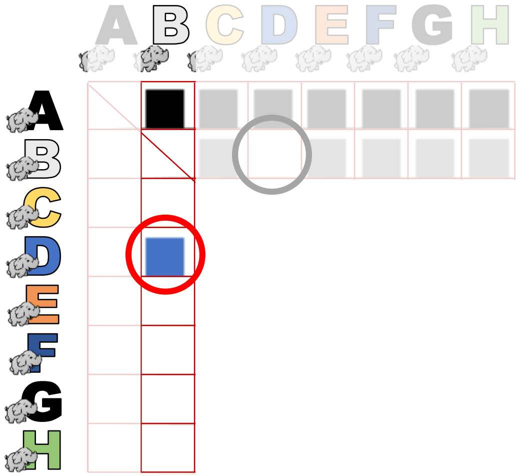
~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

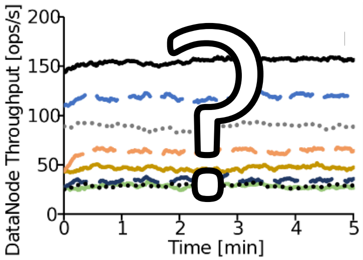
Otherwise
-> select randomly



```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
                      On blockLocations -> readBlock
  Join cl In Client.DoRandomRead
                      On cl -> blockLocations
  Where cl.host != readBlock.host
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```



When both B and D host replicas,
Clients choose B this often: (0%)
Clients choose D this often: 🟦 (100%)

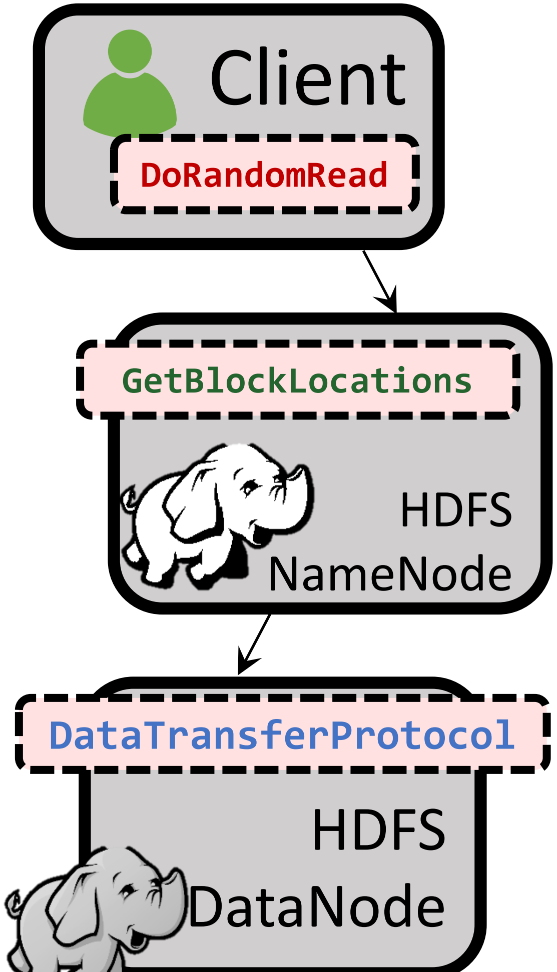


HDFS Replica Selection Policy

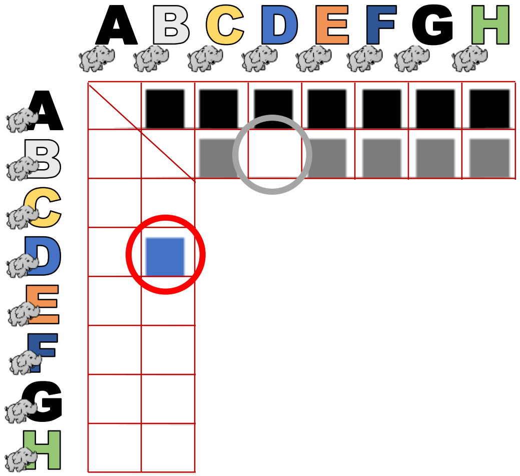
~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

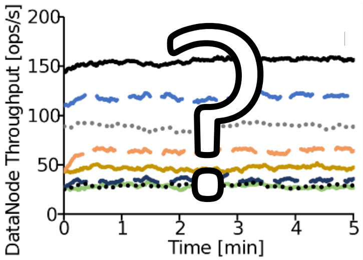
Otherwise
-> select randomly



```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
                 On blockLocations -> readBlock
  Join cl In Client.DoRandomRead
                 On cl -> blockLocations
  Where cl.host != readBlock.host
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```



When both B and D host replicas,
Clients choose B this often: (0%)
Clients choose D this often: ■ (100%)

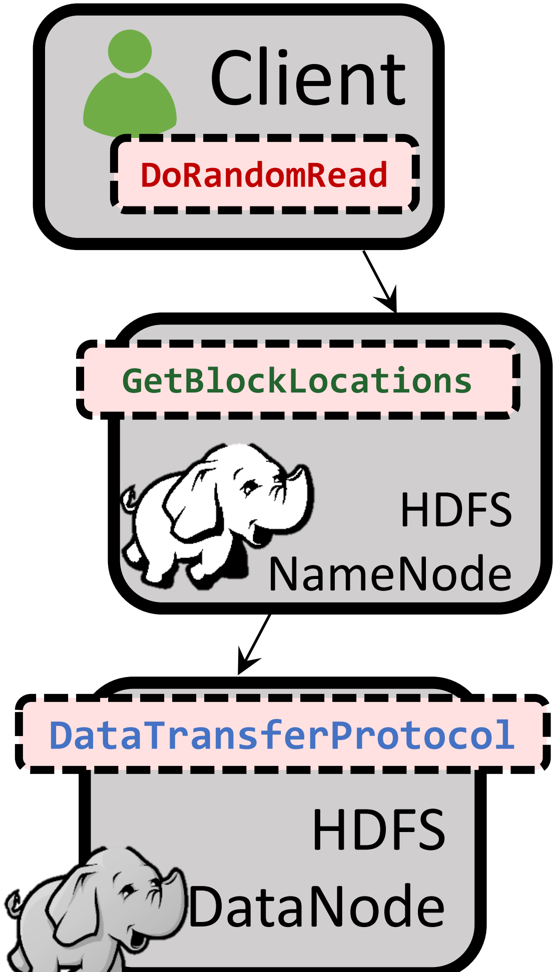


HDFS Replica Selection Policy

~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

Otherwise
-> select randomly



Client

DoRandomRead

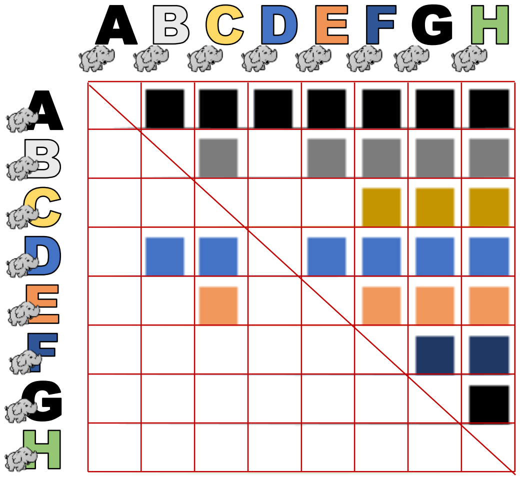
GetBlockLocations

HDFS NameNode

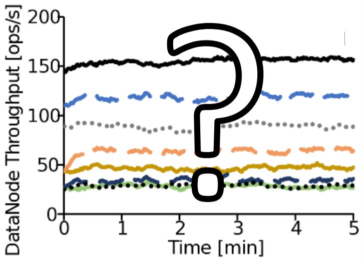
DataTransferProtocol

HDFS DataNode

```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
                 On blockLocations -> readBlock
  Join cl In Client.DoRandomRead
                 On cl -> blockLocations
  Where cl.host != readBlock.host
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```



	A	B	C	D	E	F	G	H
A		■	■	■	■	■	■	■
B			■		■	■	■	■
C						■	■	■
D		■	■		■	■	■	■
E			■			■	■	■
F							■	■
G								■
H								



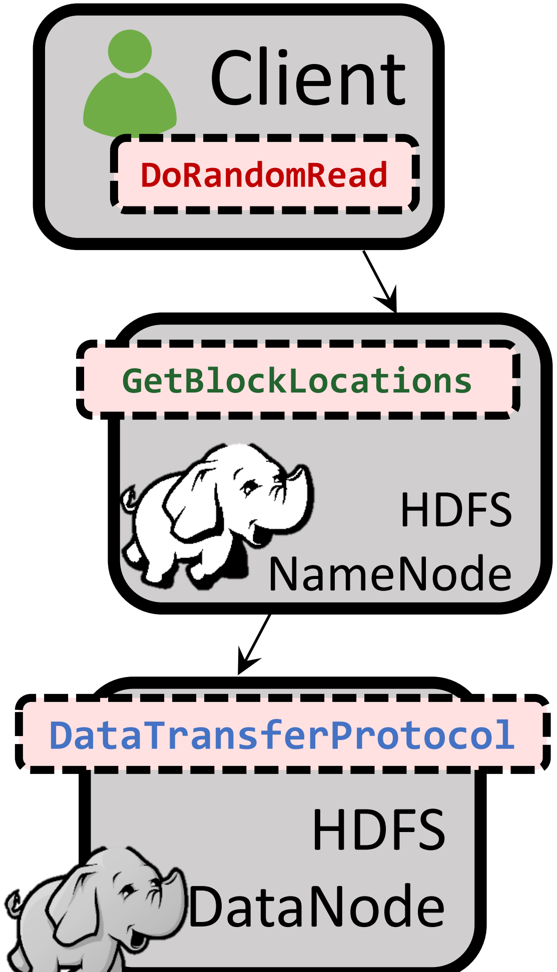
When both B and D host replicas,
Clients choose B this often: ___ (0%)
Clients choose D this often: ■ (100%)

HDFS Replica Selection Policy

~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

Otherwise
-> select randomly



Client

DoRandomRead

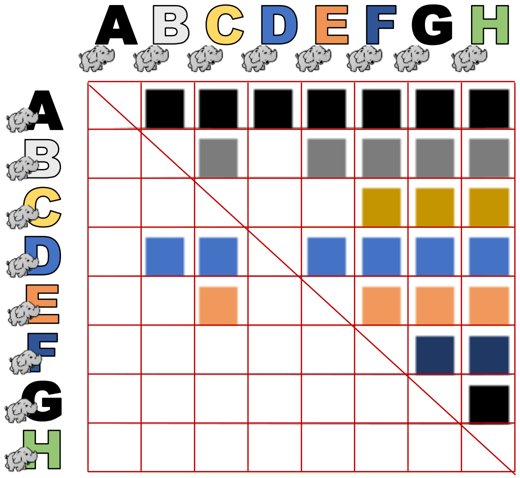
GetBlockLocations

HDFS NameNode

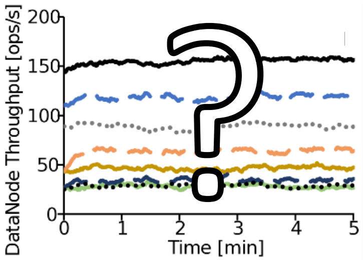
DataTransferProtocol

HDFS DataNode

```
From readBlock In DataNode.DataTransferProtocol
  Join blockLocations In NameNode.GetBlockLocations
                 On blockLocations -> readBlock
  Join cl In Client.DoRandomRead
                 On cl -> blockLocations
  Where cl.host != readBlock.host
  GroupBy blockLocations.replicas, readBlock.host
  Select blockLocations.replicas, readBlock.host, COUNT
```



	A	B	C	D	E	F	G	H
A		■	■	■	■	■	■	■
B			■		■	■	■	■
C						■	■	■
D		■	■		■	■	■	■
E			■			■	■	■
F							■	■
G								■
H								

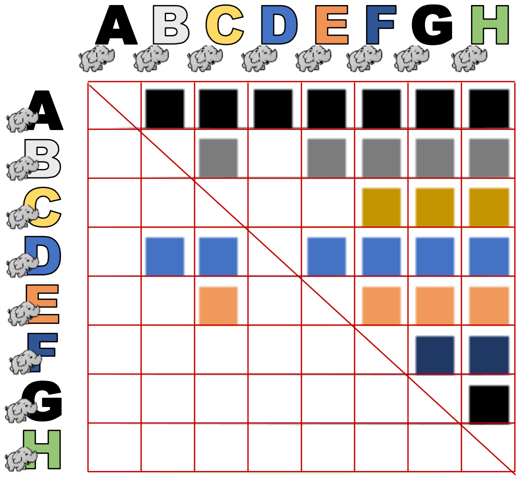


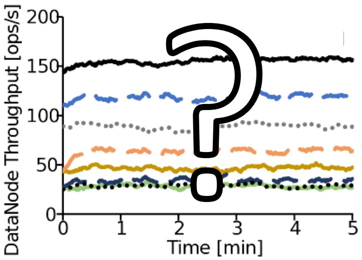
HDFS Replica Selection Policy

~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

Otherwise
-> select randomly





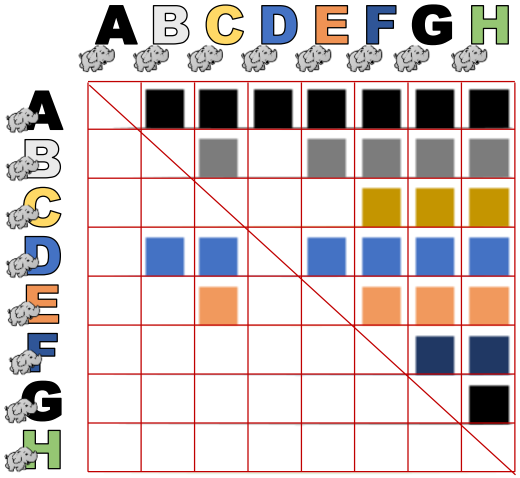
- Lack of randomization skewed workload toward certain DNs

HDFS Replica Selection Policy

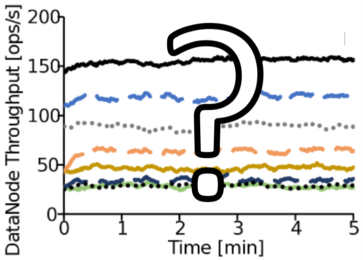
~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

Otherwise
-> select randomly



	A	B	C	D	E	F	G	H
A		■	■	■	■	■	■	■
B			■		■	■	■	■
C						■	■	■
D		■	■		■	■	■	■
E			■			■	■	■
F							■	■
G								■
H								



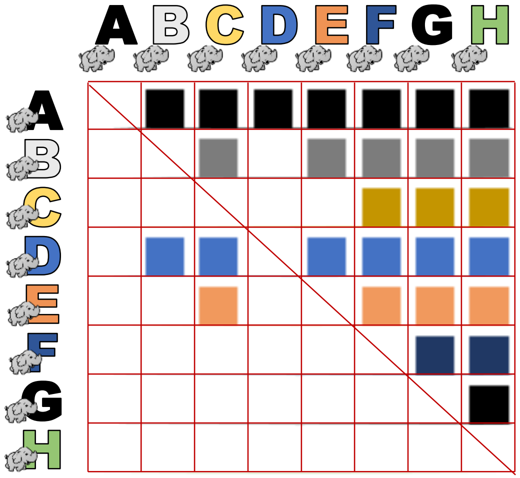
HDFS Replica Selection Policy

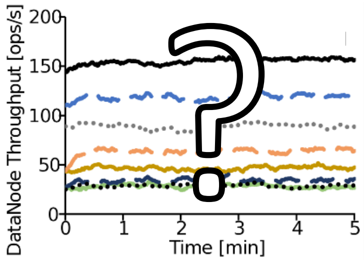
~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

Otherwise
-> select randomly

- Lack of randomization skewed workload toward certain DNs
- **HDFS-6268** Independently discovered. Fixed in HDFS 2.5





HDFS Replica Selection Policy

~~Same machine?~~
~~-> read local~~

Same rack?
-> read rack-local

Otherwise
-> select randomly

- Lack of randomization skewed workload toward certain DNs
- **HDFS-6268** Independently discovered. Fixed in HDFS 2.5

- Seamlessly add correlations between multiple components
- Very specific, one-off metrics
- This experiment: 1.5% application-level overhead

Pivot Tracing

Dynamic Causal Monitoring for Distributed Systems

Pivot Tracing

Dynamic Causal Monitoring for Distributed Systems

$$\overrightarrow{\bowtie}$$

Happened-Before Join

Pivot Tracing

Dynamic Causal Monitoring for Distributed Systems

$\overrightarrow{\bowtie}$

Happened-Before Join

Dynamic Instrumentation

Causal Tracing

Pivot Tracing

Dynamic Causal Monitoring for Distributed Systems

$\overrightarrow{\bowtie}$

Happened-Before Join

Dynamic Instrumentation

Causal Tracing

Acceptable overheads for production (we think)

Pivot Tracing

Dynamic Causal Monitoring for Distributed Systems

$$\overrightarrow{\bowtie}$$

Happened-Before Join

Dynamic Instrumentation

Causal Tracing

Acceptable overheads for production (we think)

Standing basic queries

Potential to dig deeper

Pivot Tracing

Dynamic Causal Monitoring for Distributed Systems

$\overrightarrow{\bowtie}$

Happened-Before Join

Dynamic Instrumentation

Causal Tracing

Acceptable overheads for production (we think)

Standing basic queries

Potential to dig deeper

Tracepoint A
Class: A
Method: A1()

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

```
From a In A
  Join b In B On a -> b
  GroupBy a.procName
  Select a.procName, SUM(b.delta)
```

Advice A1 A

```
OBSERVE     procName
PACK        procName
```

Advice B1 B

```
OBSERVE     delta
UNPACK      procName
EMIT        procName, SUM(delta)
```

Jonathan Mace

Ryan Roelke

Rodrigo Fonseca

BROWN