facebook

# Analysis of HDFS Under HBase

## A Facebook Messages Case Study

Tyler Harter, Dhruba Borthakur*, Siying Dong*, Amitanand Aiyer*, Liyin Tang*, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

University of Wisconsin-Madison

*Facebook Inc.

# Why Study Facebook Messages?

Represents an important type of application. Universal backend for:

- Cellphone texts
- Chats
- Emails

# Why Study Facebook Messages?

Represents an important type of application. Universal backend for:

- Cellphone texts
- Chats
- Emails

# Why Study Facebook Messages?

Represents an important type of application. Universal backend for:

- Cellphone texts
- Chats
- Emails

# Why Study Facebook Messages?

Represents an important type of application. Universal backend for:

- Cellphone texts
- Chats
- Emails

# Why Study Facebook Messages?

Represents an important type of application. Universal backend for:

- Cellphone texts
- Chats
- Emails

Represents HBase over HDFS

- Common backend at Facebook and other companies
- Similar stack used at Google (BigTable over GFS)

# Why Study Facebook Messages?

Represents an important type of application. Universal backend for:

- Cellphone texts
- Chats
- Emails

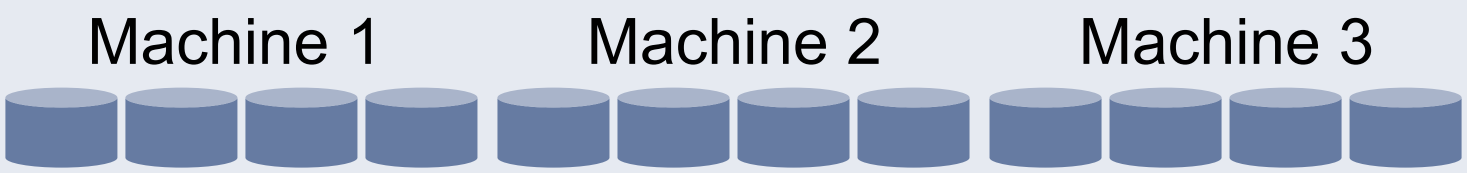
Represents HBase over HDFS

- Common backend at Facebook and other companies
- Similar stack used at Google (BigTable over GFS)

Represents layered storage

# Building a Distributed Application (Messages)

We have many machines with many disks.
*How should we use them to store messages?*



# Building a Distributed Application (Messages)

One option: use machines and disks directly.

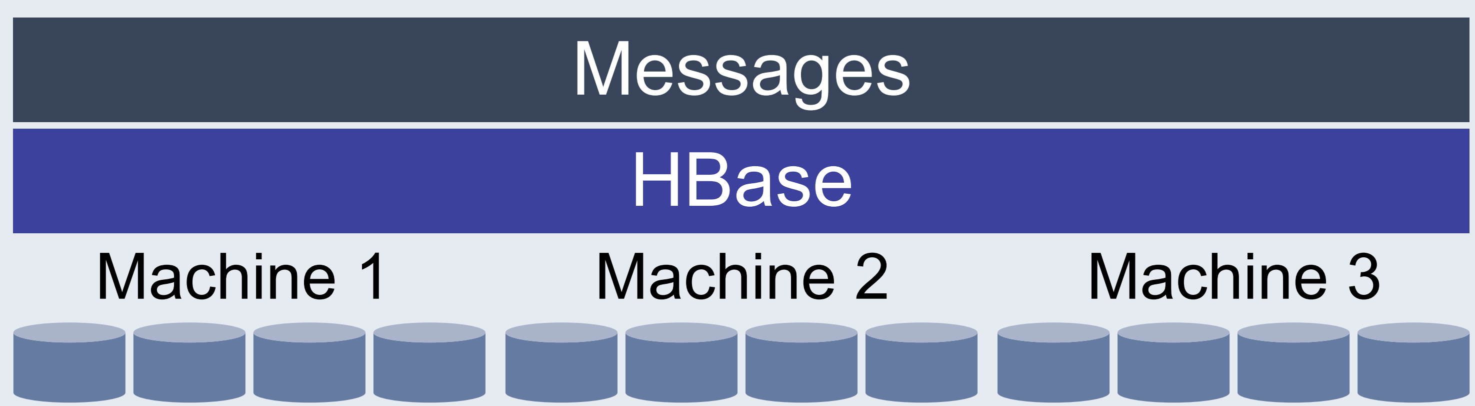
# Building a Distributed Application (Messages)

One option: use machines and disks directly.
Very specialized, but very high development cost.

# Building a Distributed Application (Messages)

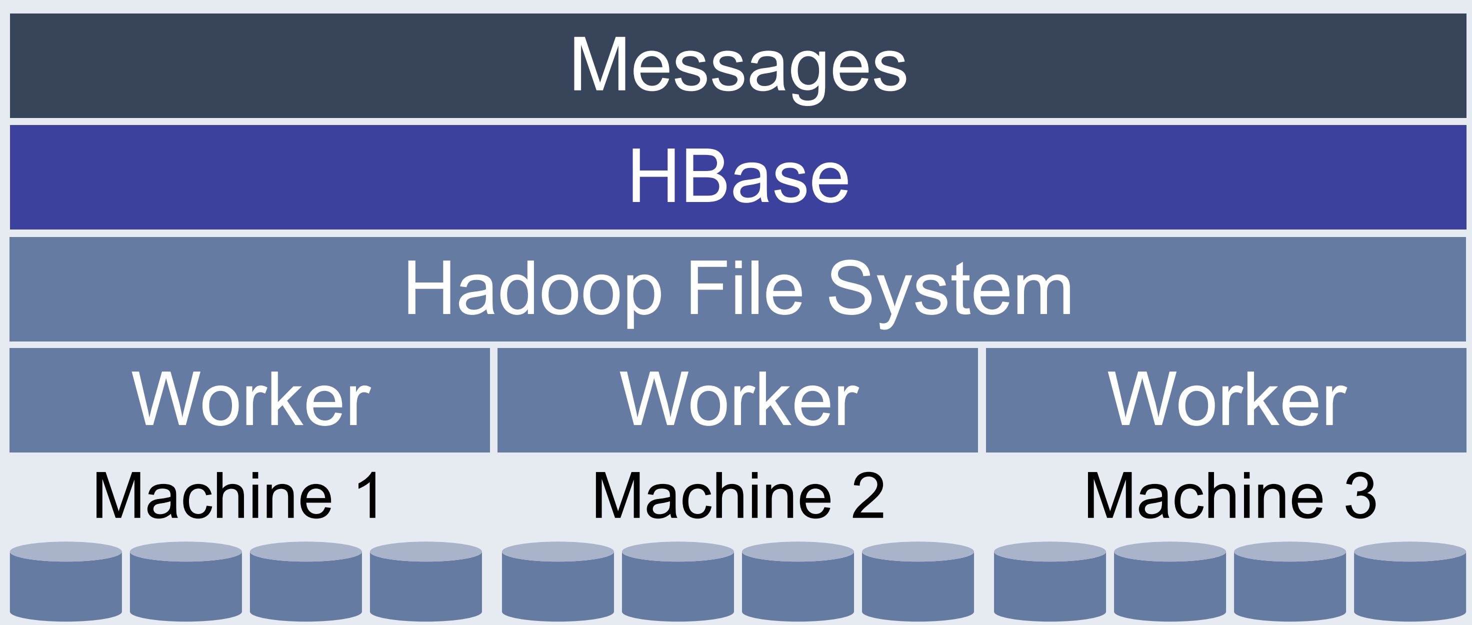
# Building a Distributed Application (Messages)

Use HBase for K/V logic



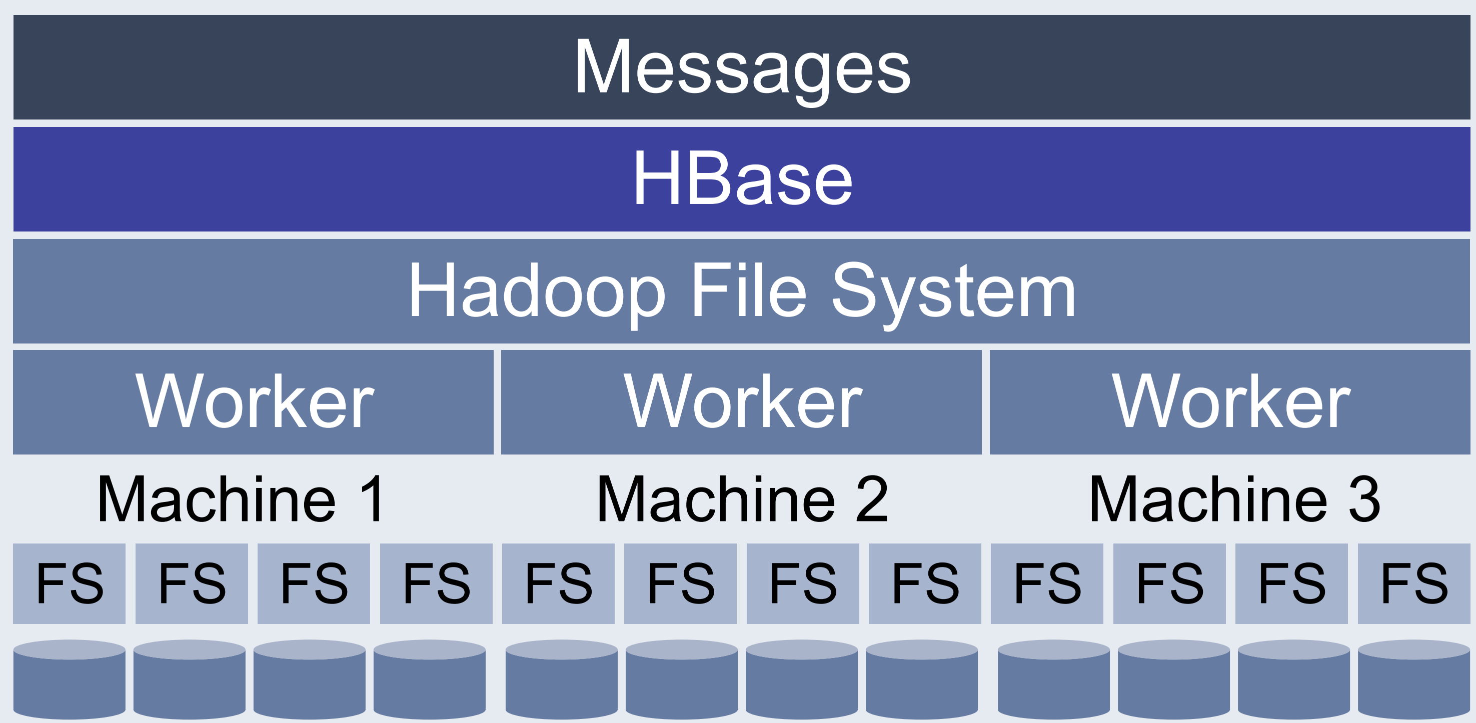
# Building a Distributed Application (Messages)

Use HBase for K/V logic
Use HDFS for replication



# Building a Distributed Application (Messages)

Use HBase for K/V logic
Use HDFS for replication
Use Local FS for allocation



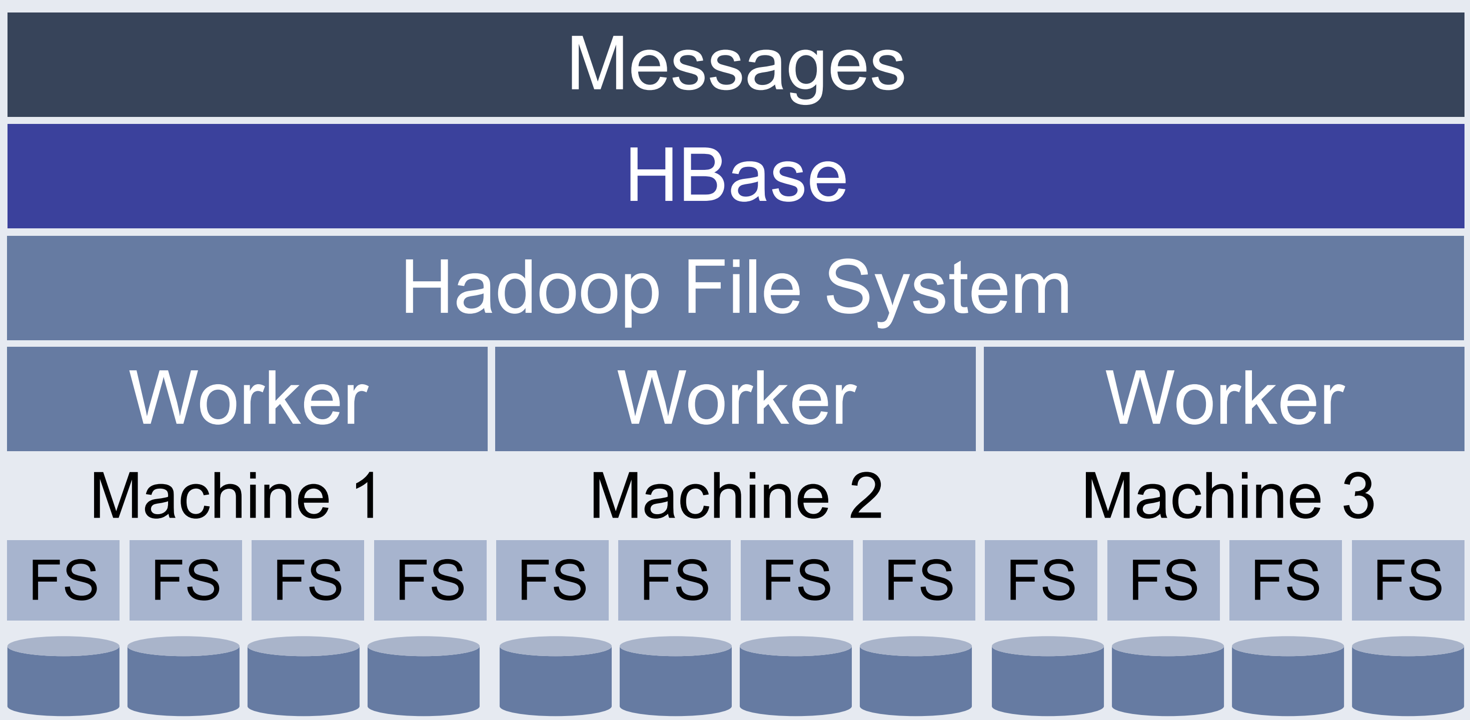
# Layered Storage Discussion

## Layering Advantages

- Simplicity (thus fewer software bugs)
- Lower development costs
- Code sharing between systems

## Layering Questions

- Is layering free performance-wise?
- Can layer integration be useful?
- Should there be multiple HW layers?



# Outline

Intro

- Messages stack overview
- Methodology: trace-driven analysis and simulation
- HBase background

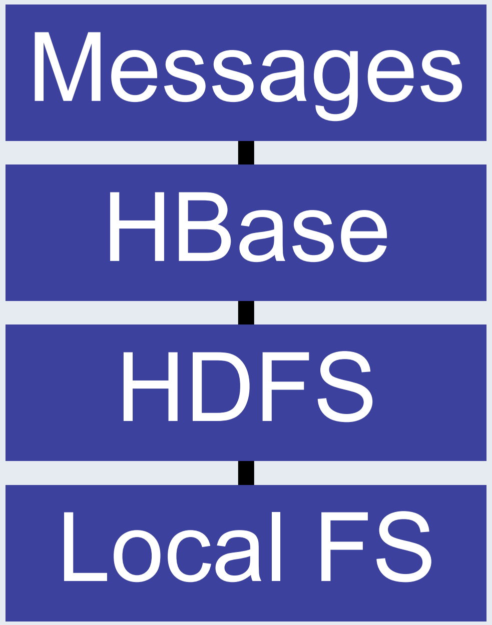
Results

- Workload analysis
- Hardware simulation: adding a flash layer
- Software simulation: integrating layers

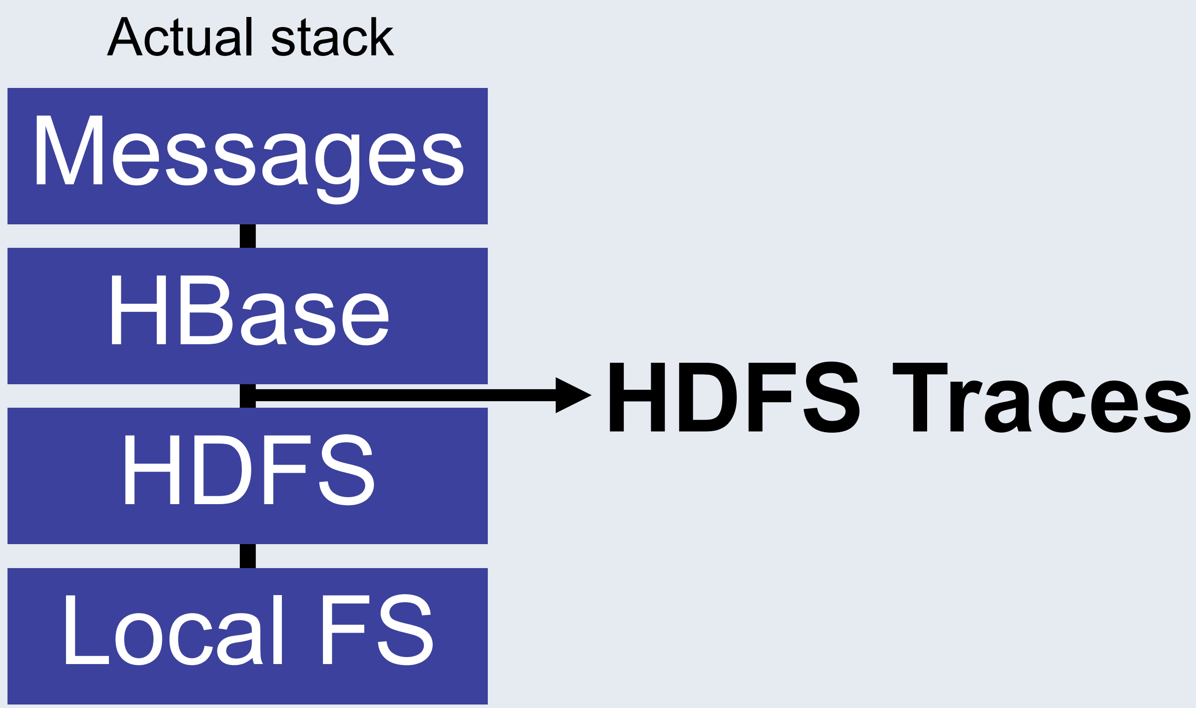
Conclusions

# Methodology

Actual stack



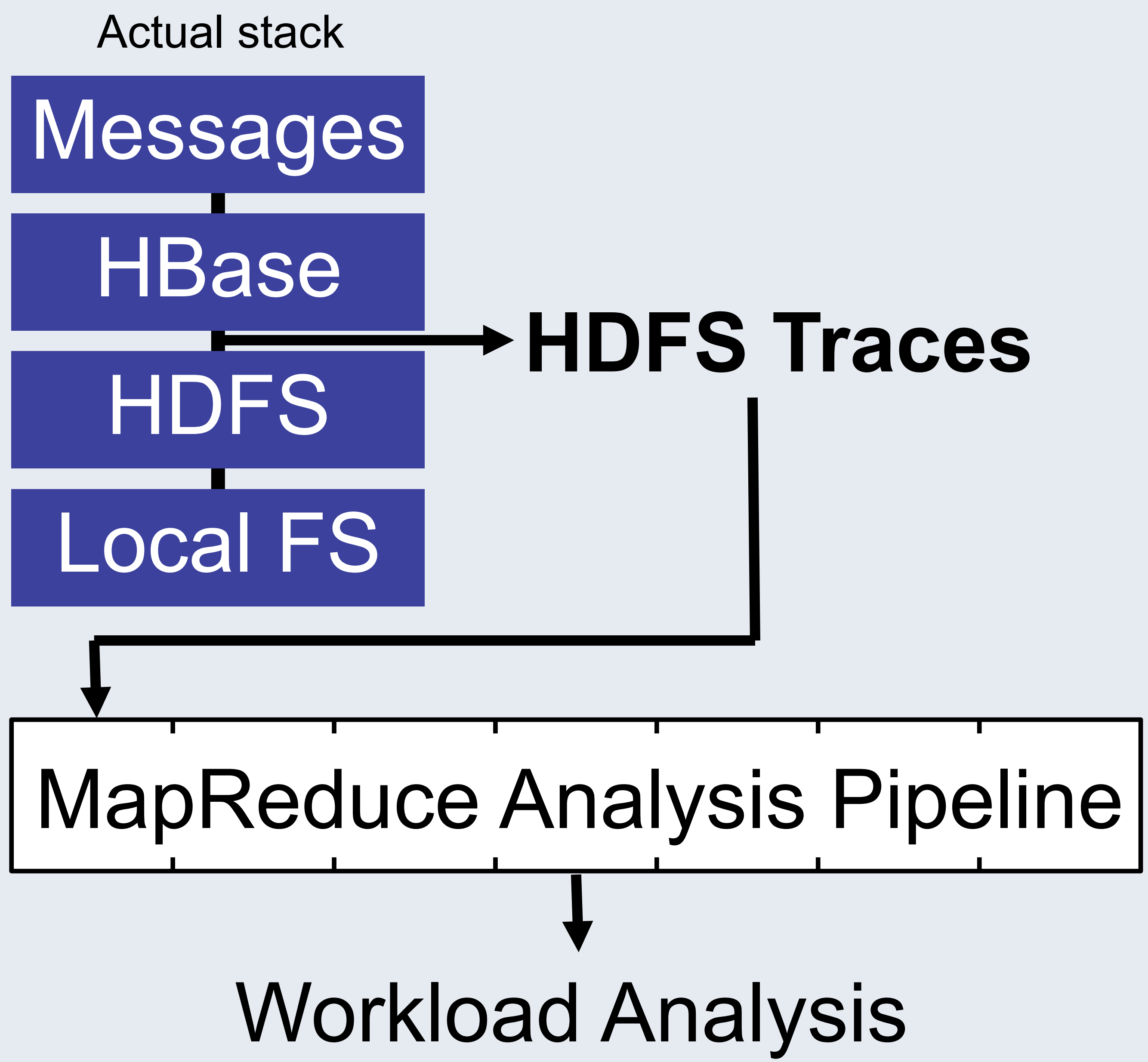
# Methodology



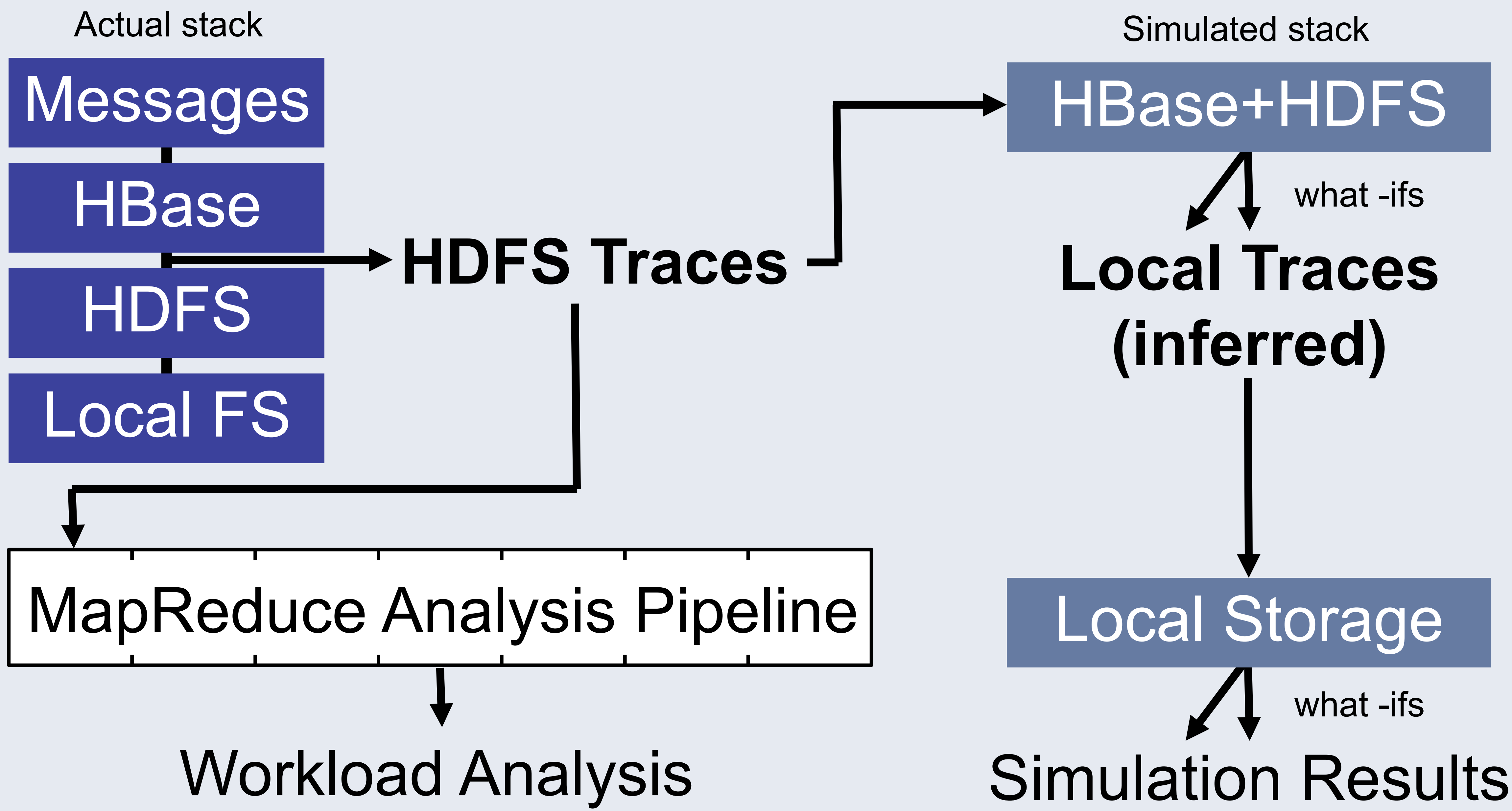

Hadoop Trace FS (HTFS)

- Collects request details
  - Reads/writes, offsets, lengths
- 9 shadow machines
- 8.3 days

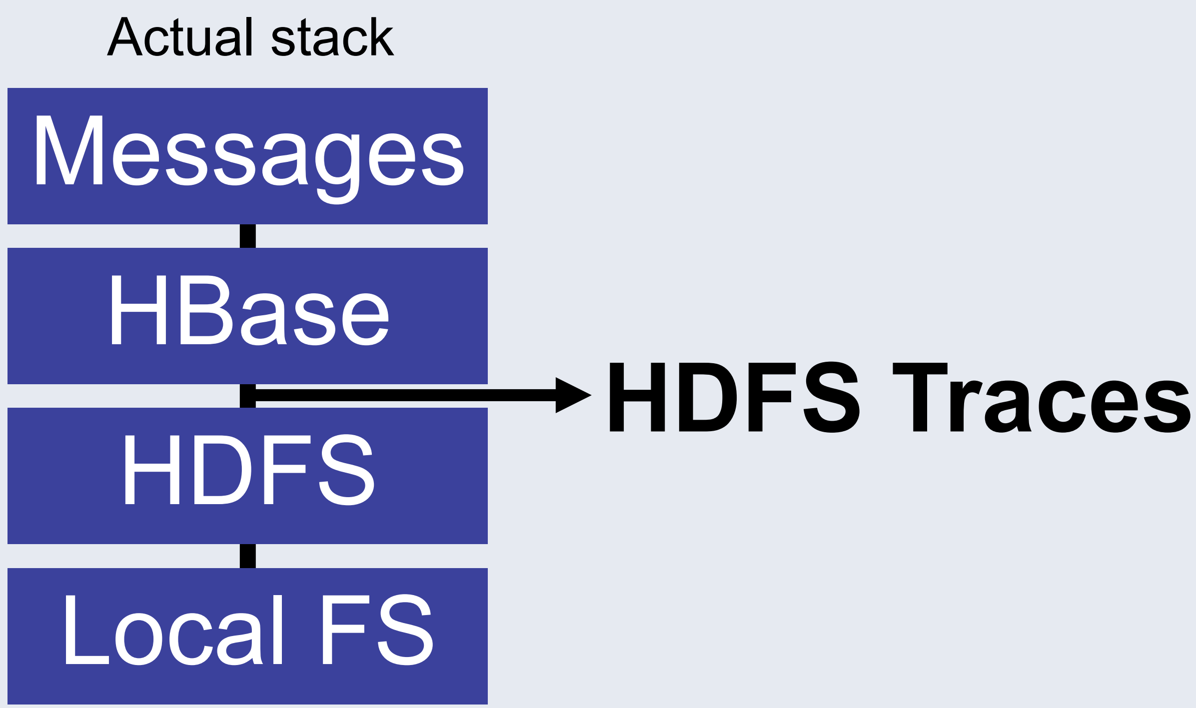
# Methodology



# Methodology

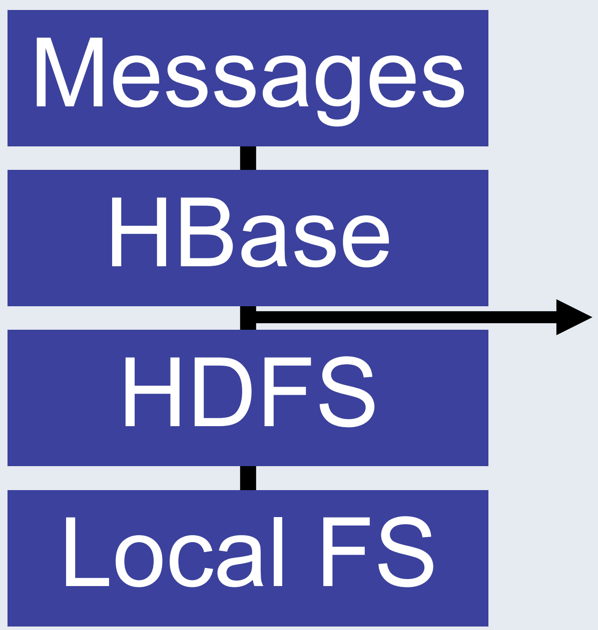


# Methodology



# Methodology

Actual stack




**HDFS Traces**

**Background: how does HBase use HDFS?**

# Outline

Intro

- Messages stack overview
- Methodology: trace-driven analysis and simulation
- HBase background

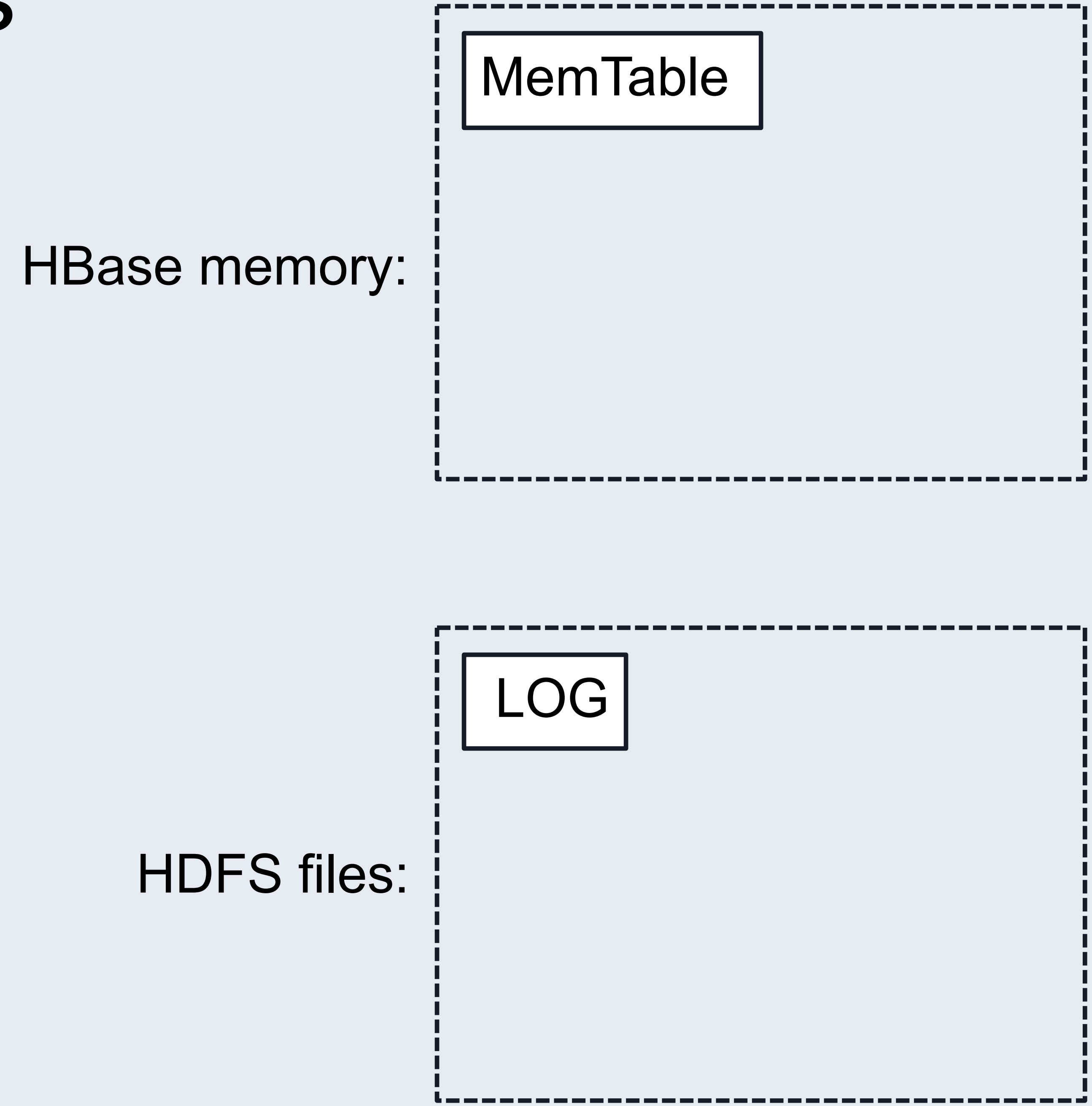
Results

- Workload analysis
- Hardware simulation: adding a flash layer
- Software simulation: integrating layers

Conclusions

# HBase’s HDFS Files

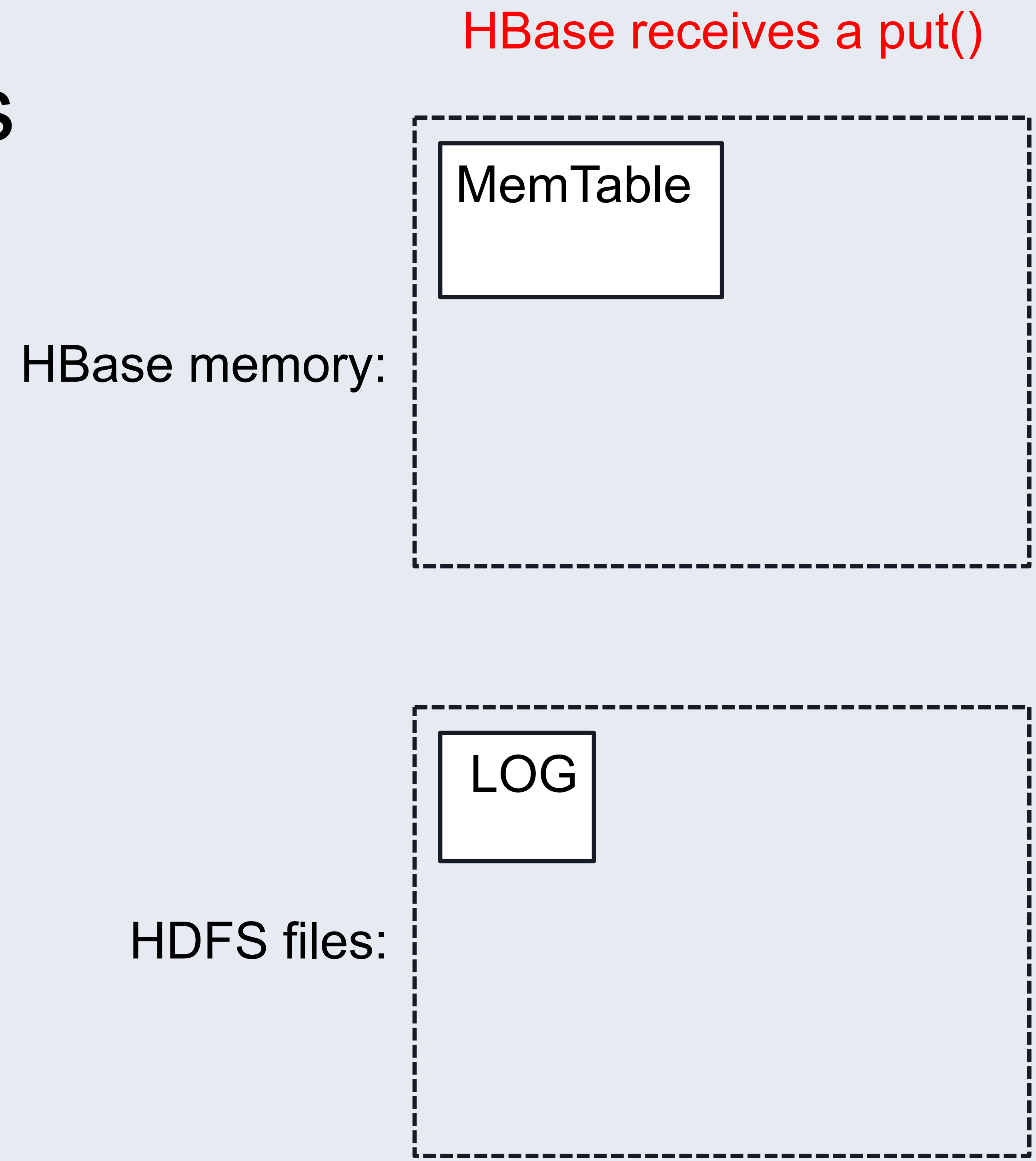
Four activities do HDFS I/O:



# HBase's HDFS Files

Four activities do HDFS I/O:

- Logging

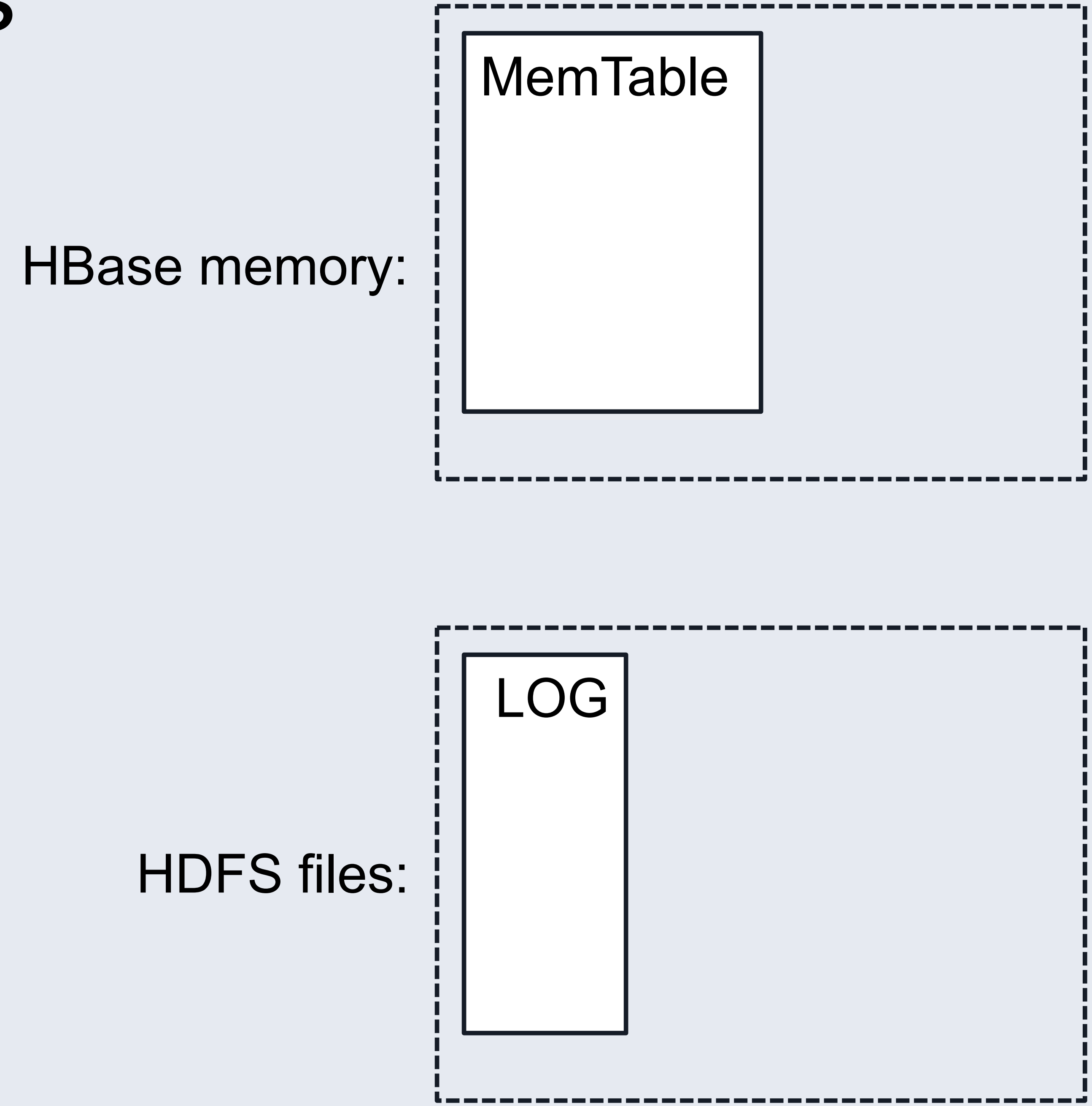


# HBase's HDFS Files

Four activities do HDFS I/O:

- Logging

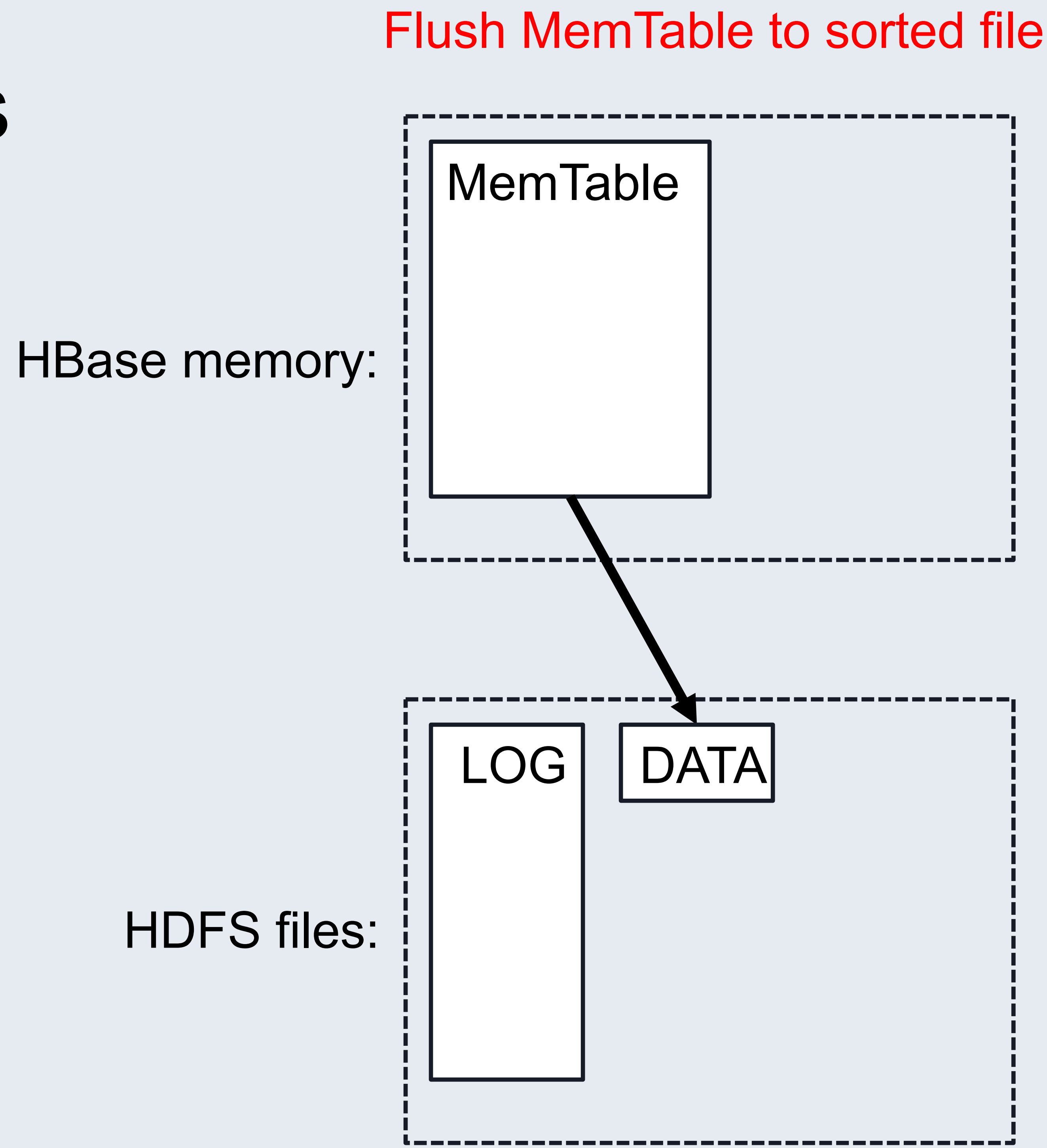
After many puts, MemTable is full



# HBase's HDFS Files

Four activities do HDFS I/O :

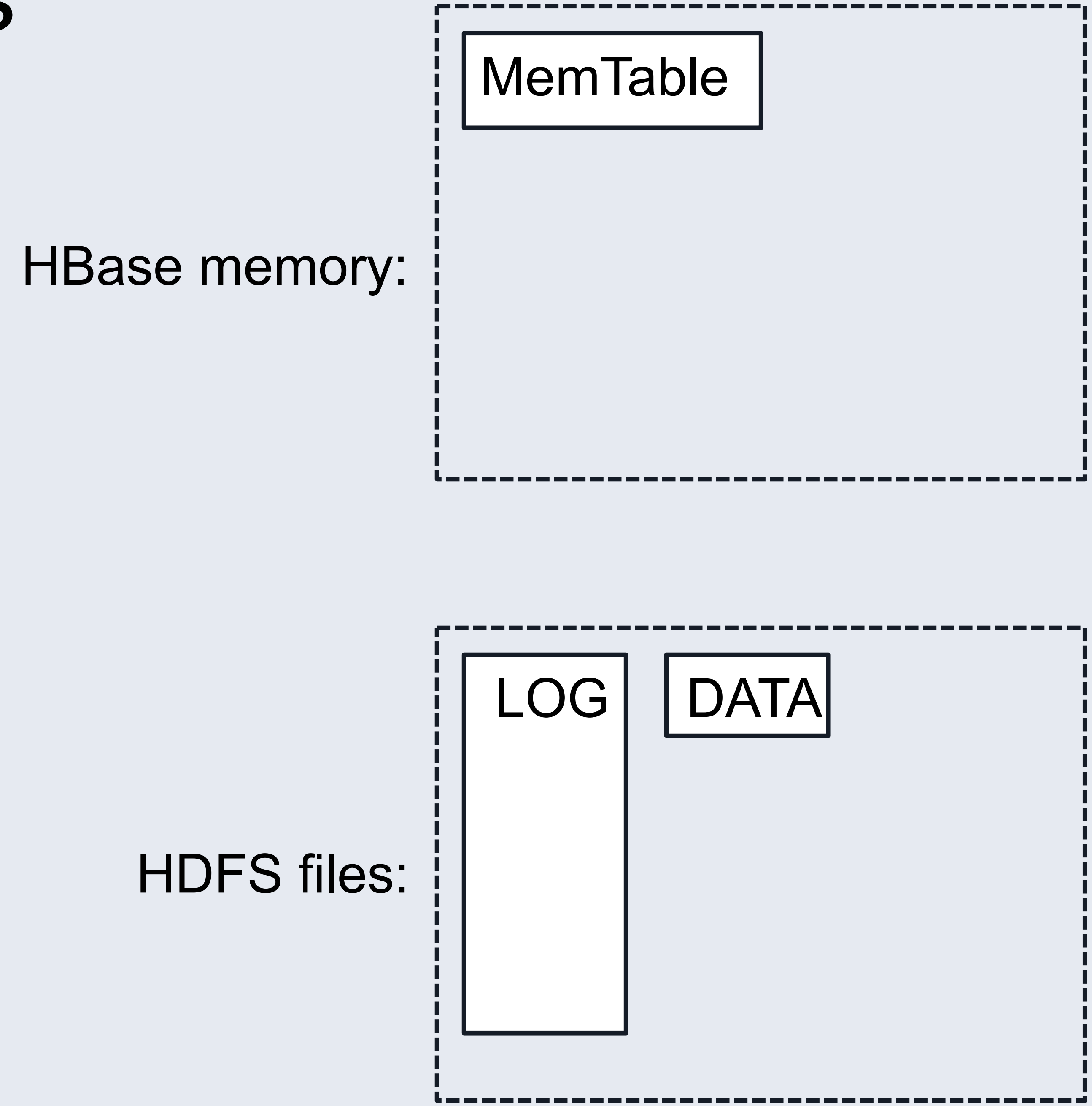
- Logging
- Flushing



# HBase's HDFS Files

Four activities do HDFS I/O :

- Logging
- Flushing

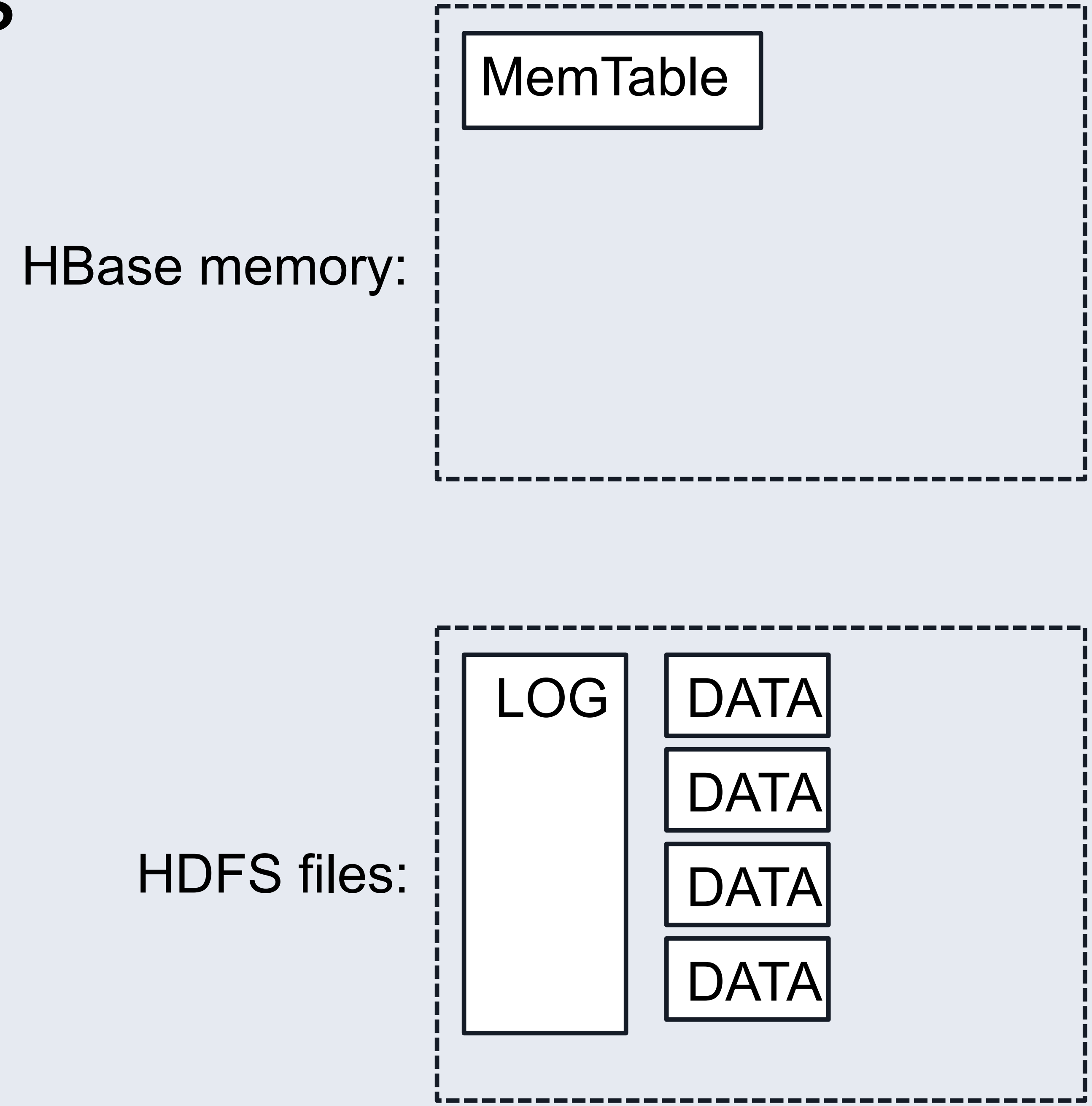


After many flushes, files accumulate

# HBase’s HDFS Files

Four activities do HDFS I/O :

- Logging
- Flushing

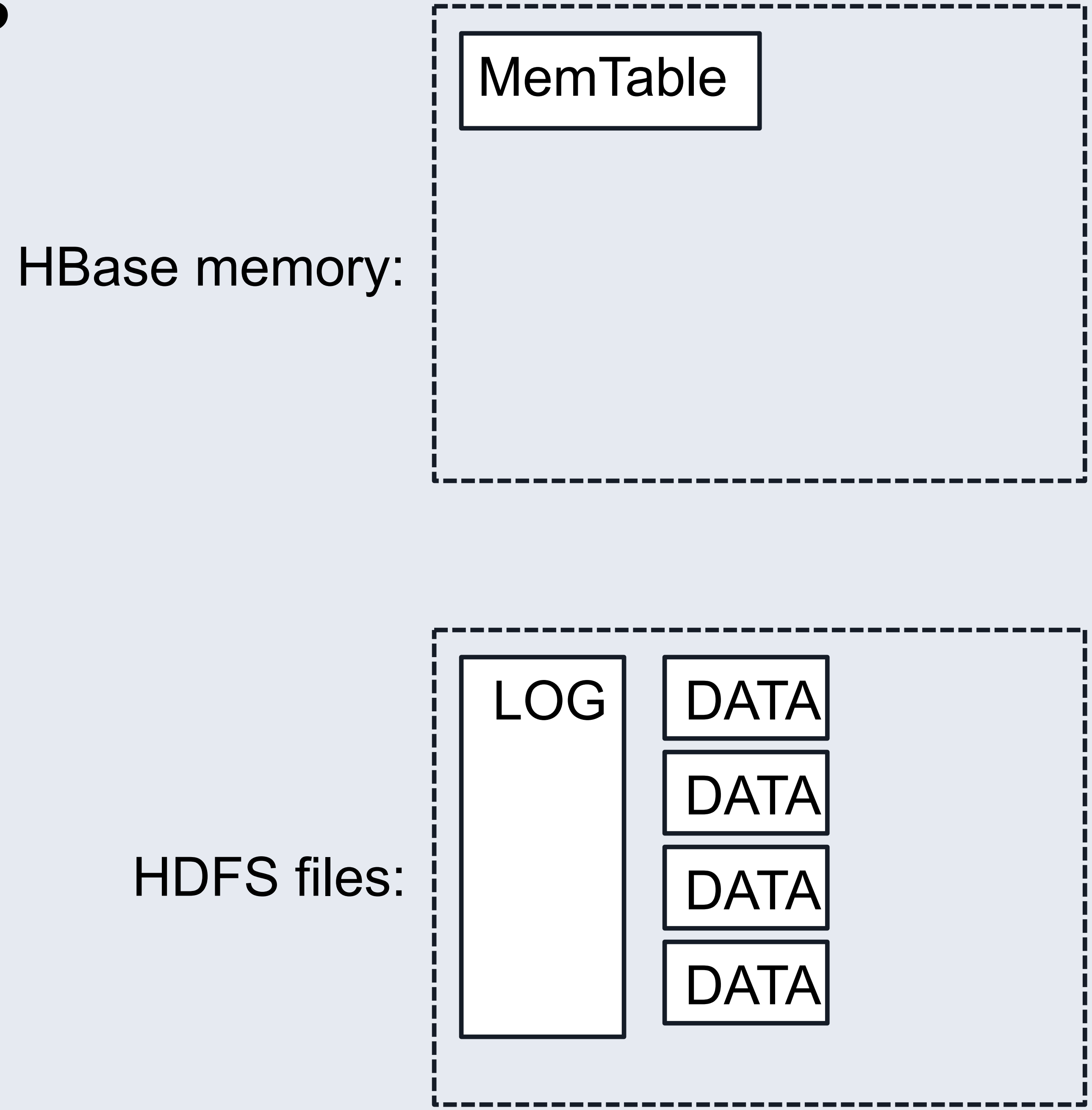


# HBase's HDFS Files

get() requests may check many of these

Four activities do HDFS I/O:

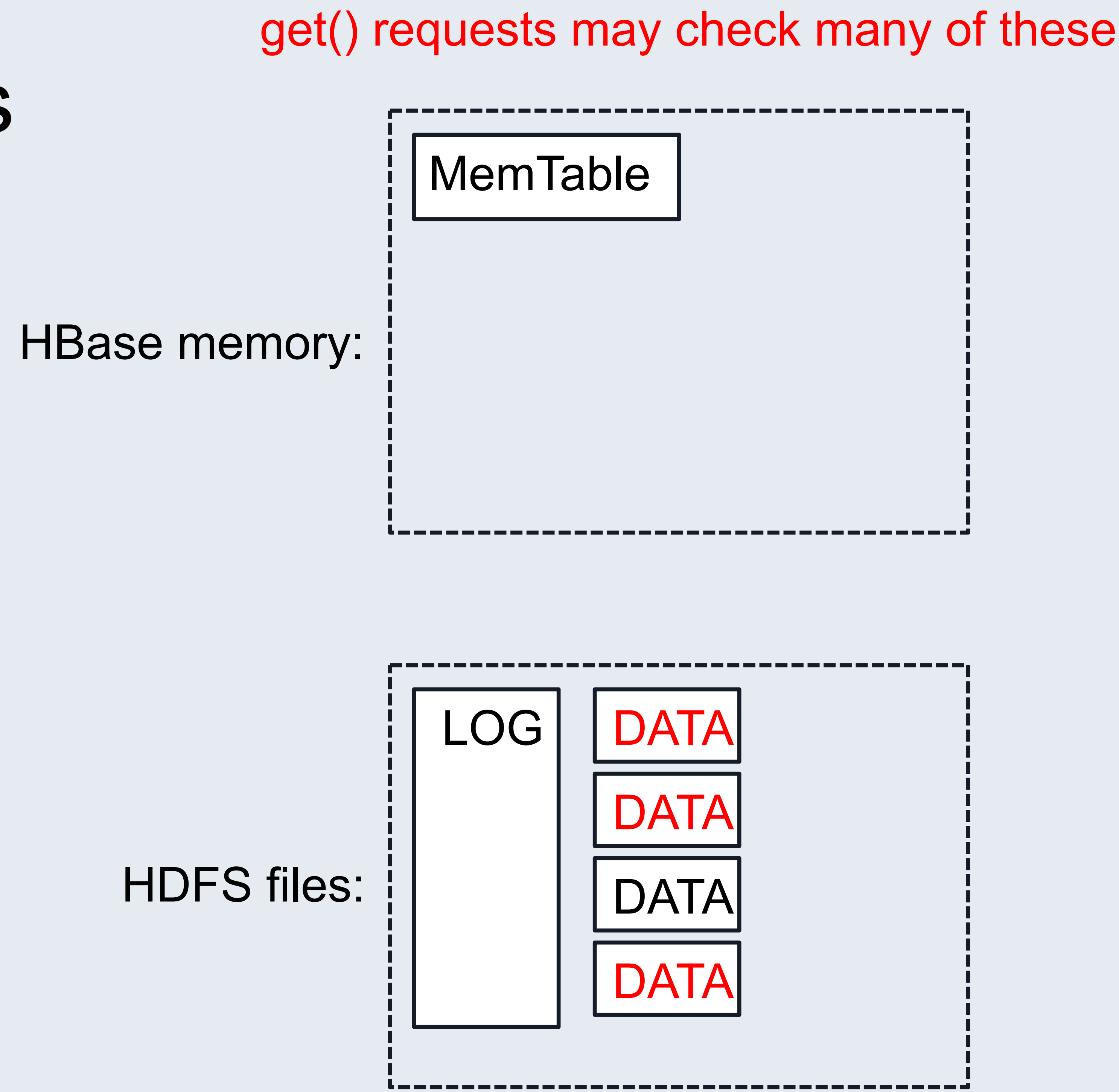
- Logging
- Flushing



# HBase's HDFS Files

Four activities do HDFS I/O:

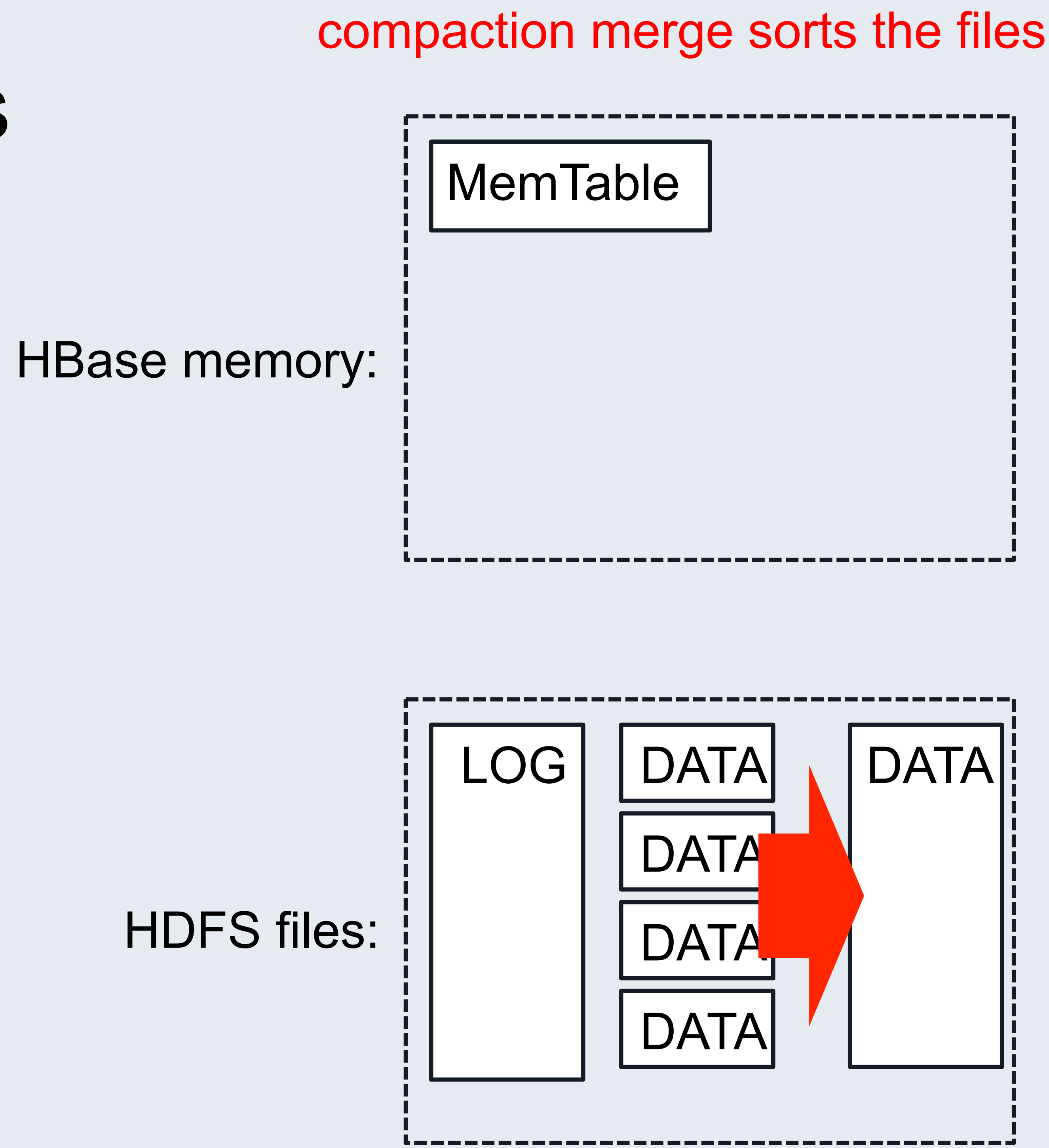
- Logging
- Flushing
- Foreground reads



# HBase's HDFS Files

Four activities do HDFS I/O:

- Logging
- Flushing
- Foreground reads
- Compaction

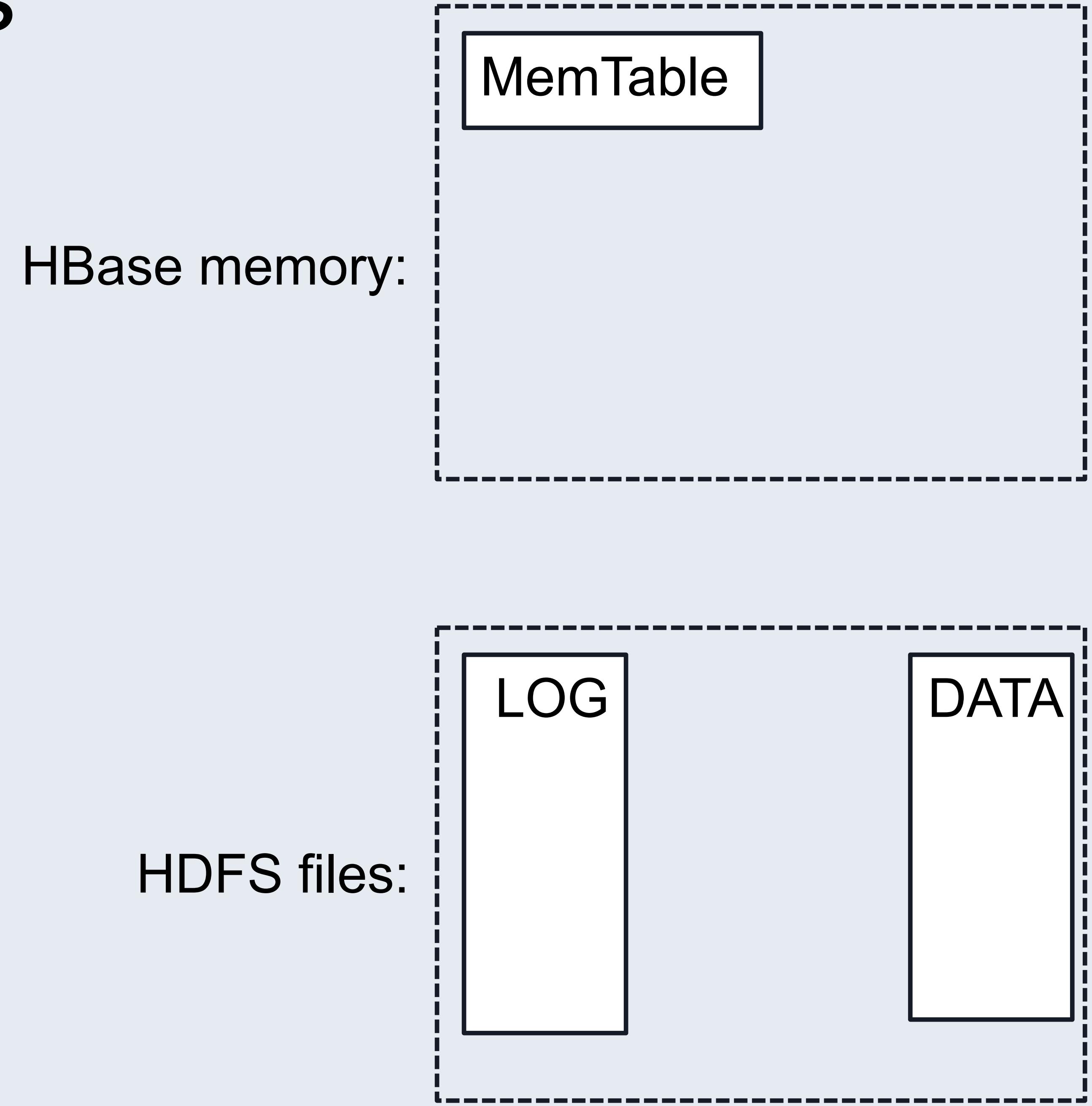


compaction merge sorts the files

# HBase's HDFS Files

Four activities do HDFS I/O:

- Logging
- Flushing
- Foreground reads
- Compaction



# HBase's HDFS Files

Four activities do HDFS I/O:

- Logging
- Flushing
- Foreground reads
- Compaction

Baseline I/O:

- Flushing and foreground reads are always required

# HBase's HDFS Files

Four activities do HDFS I/O:

- Logging
- Flushing
- Foreground reads
- Compaction

Baseline I/O:

- Flushing and foreground reads are always required

HBase overheads:

- Logging: useful for crash recovery (but not normal operation)
- Compaction: improves performance (but not required for correctness)

# Outline

Intro

- Messages stack overview
- Methodology: trace-driven analysis and simulation
- HBase background

Results

- Workload analysis
- Hardware simulation: adding a flash layer
- Software simulation: integrating layers

Conclusions

# Workload Analysis Questions

At each layer, what activities read or write?

How large is the dataset?

How large are created files?

How sequential is I/O?

# Workload Analysis Questions

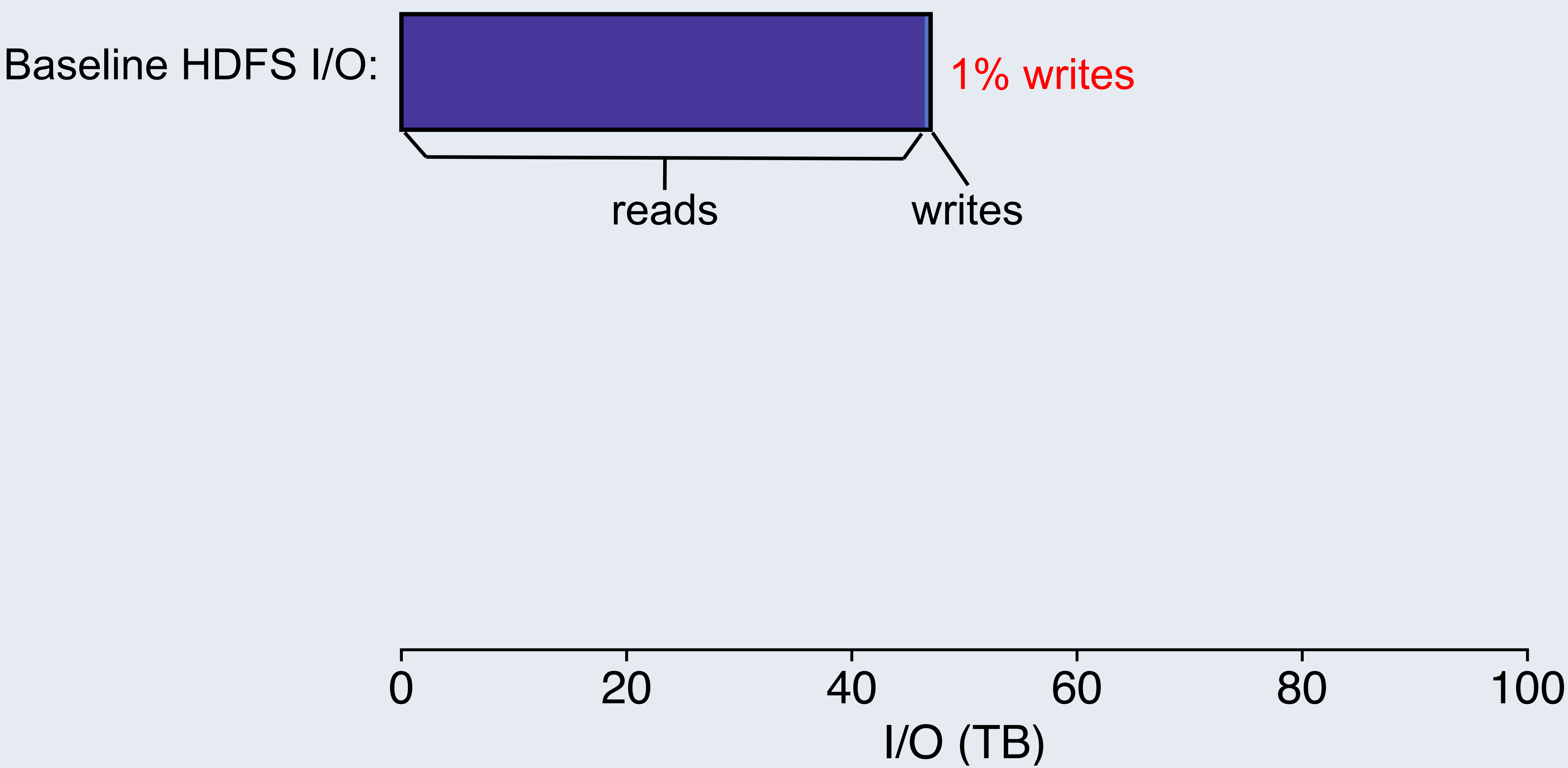
At each layer, what activities read or write?

How large is the dataset?

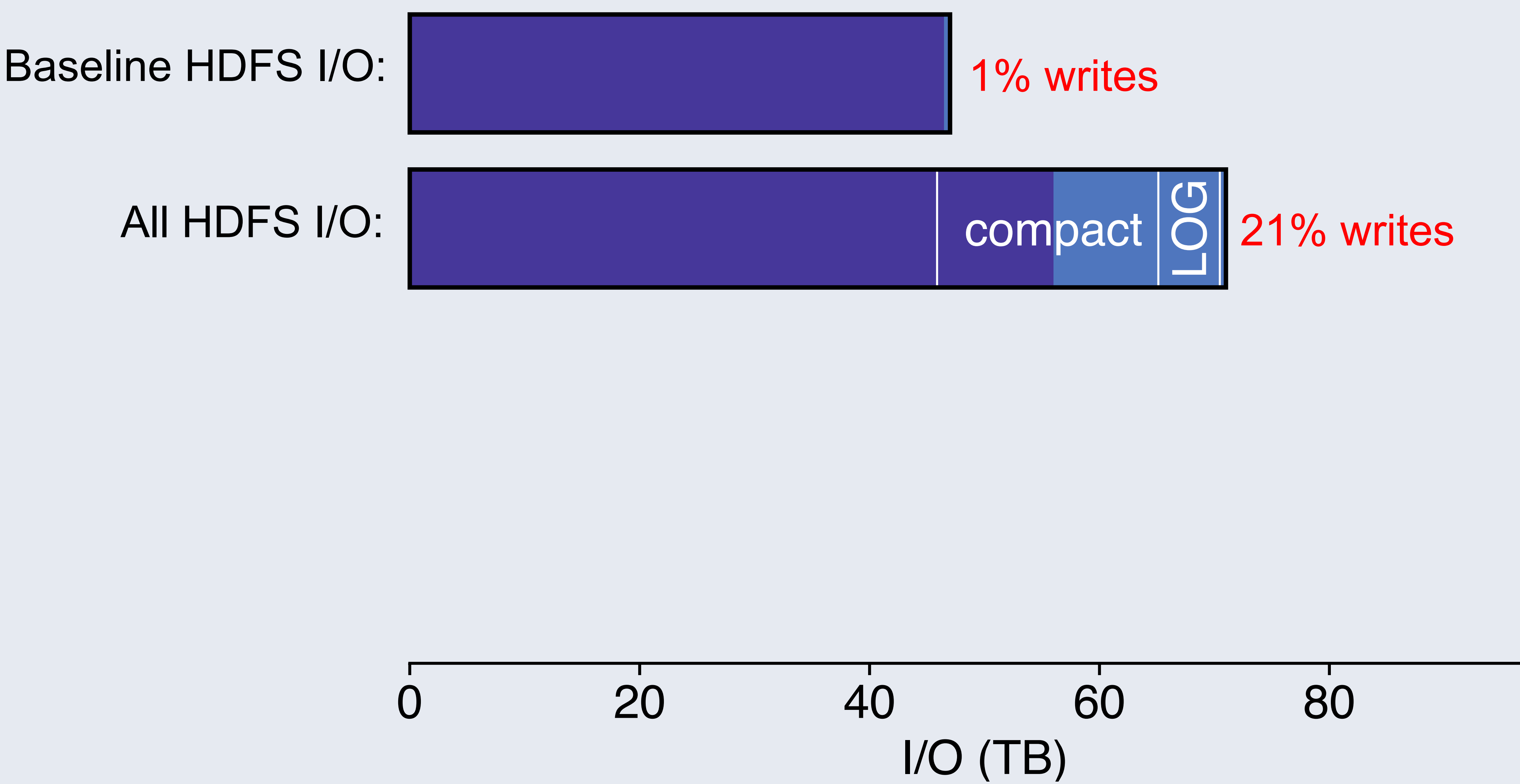
How large are created files?

How sequential is I/O?

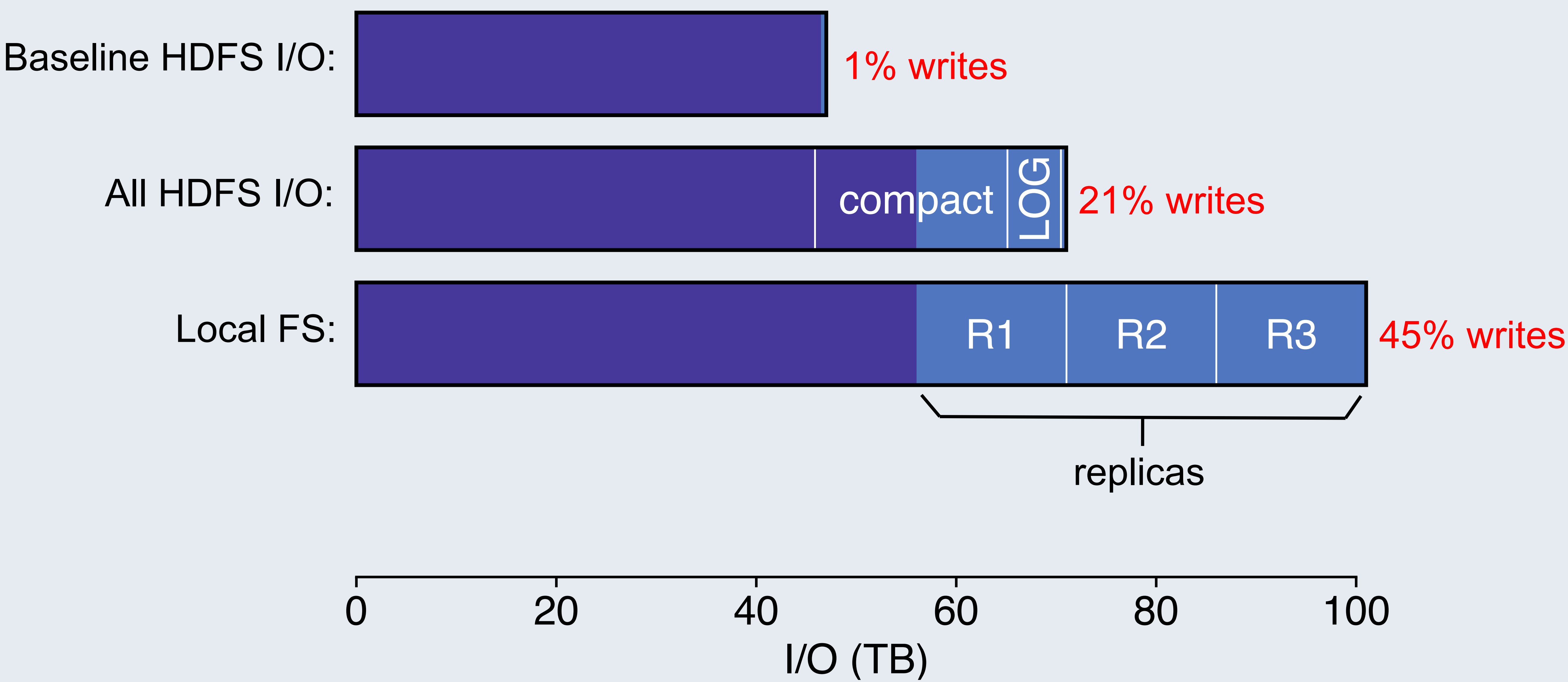
# Cross-layer R/W Ratios

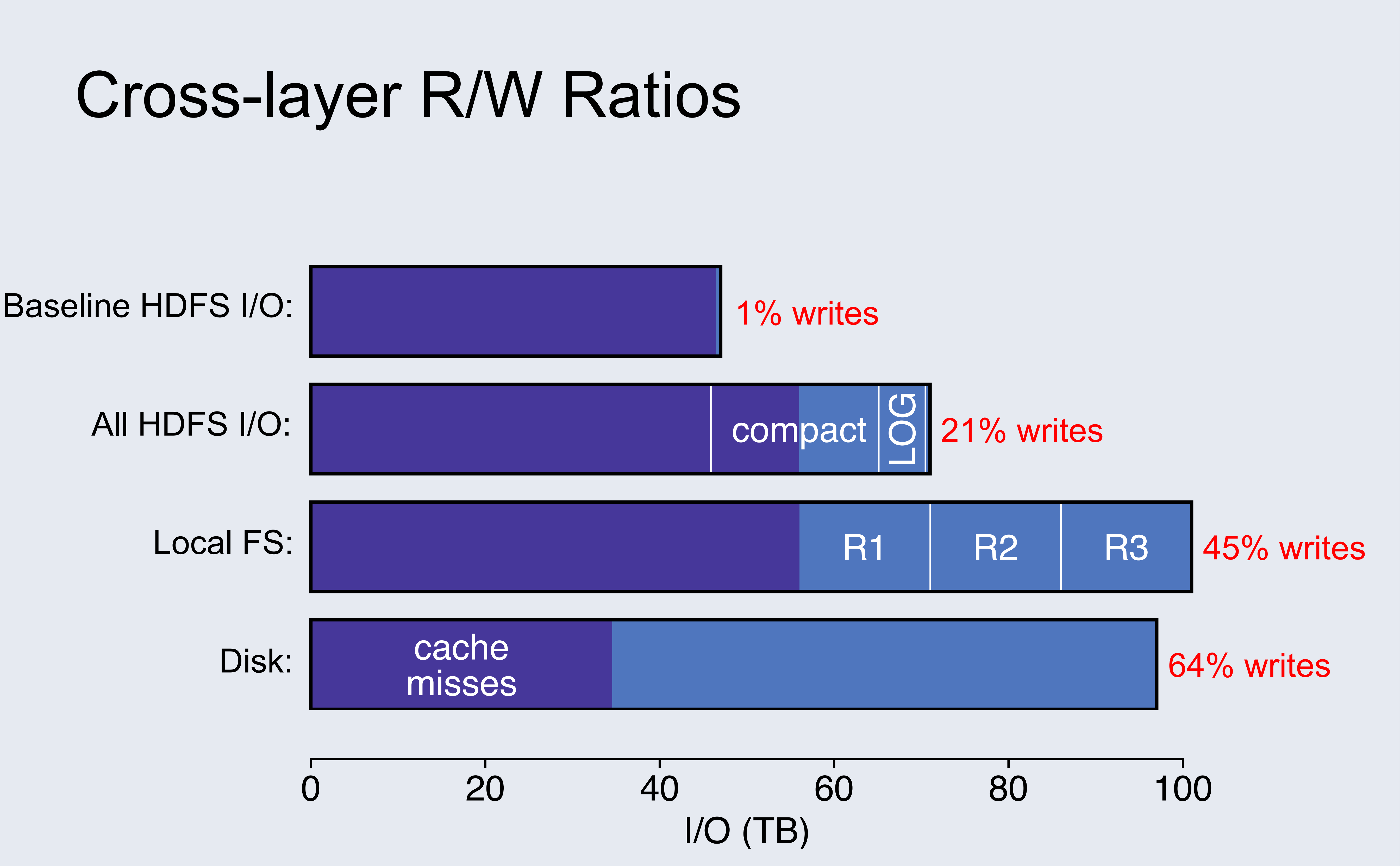


# Cross-layer R/W Ratios



# Cross-layer R/W Ratios




Cross-layer R/W Ratios
Baseline HDFS I/O:
1% writes
All HDFS I/O:
compact
LOG
21% writes
Local FS:
R1
R2
R3
45% writes
Disk:
cache
misses
64% writes
0
20
40
60
80
100
I/O (TB)

# Workload Analysis Conclusions

① Layers amplify writes: 1% => 64%

- Logging, compaction, and replication increase writes
- Caching decreases reads

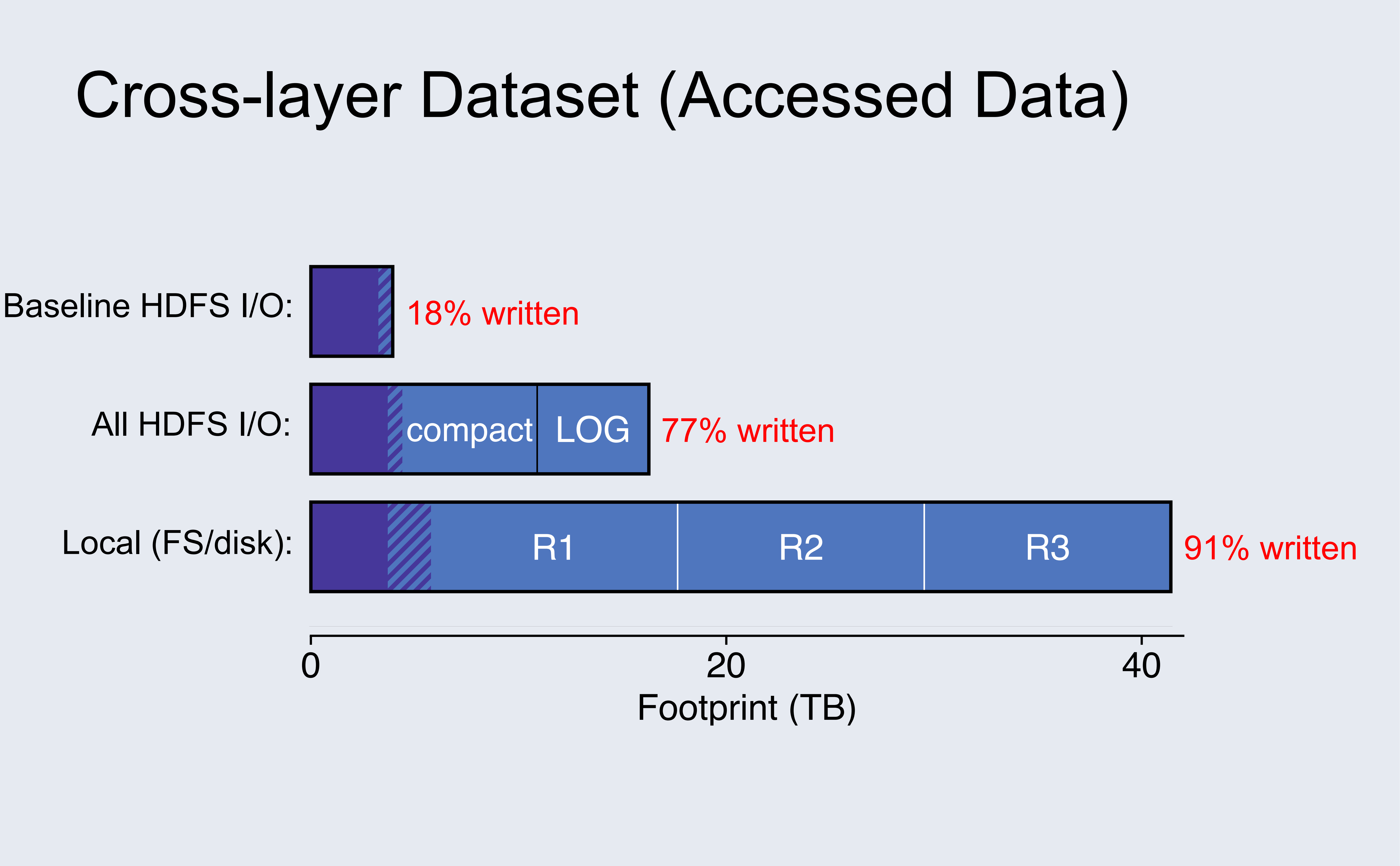
# Workload Analysis Questions

At each layer, what activities read or write?

How large is the dataset?

How large are created files?

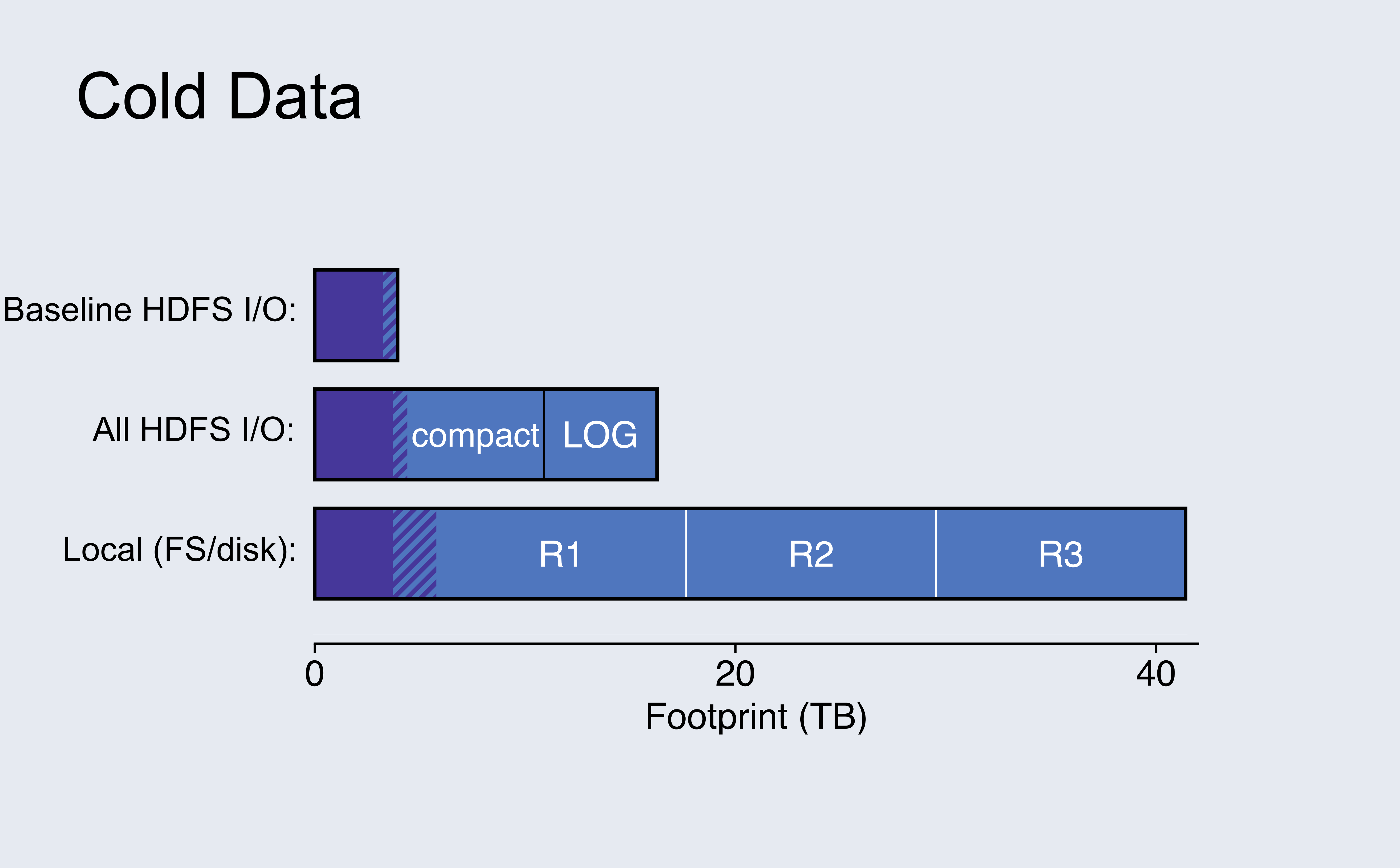
How sequential is I/O?


Cross-layer Dataset (Accessed Data)
Baseline HDFS I/O:
18% written
All HDFS I/O:
compact
LOG
77% written
Local (FS/disk):
R1
R2
R3
91% written
0
20
40
Footprint (TB)

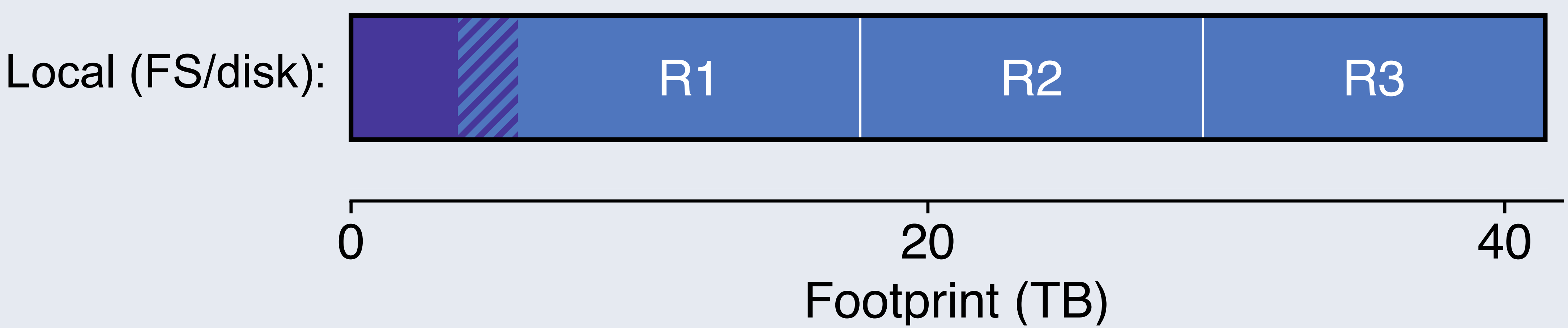
# Workload Analysis Conclusions

① Layers amplify writes: 1% => 64%

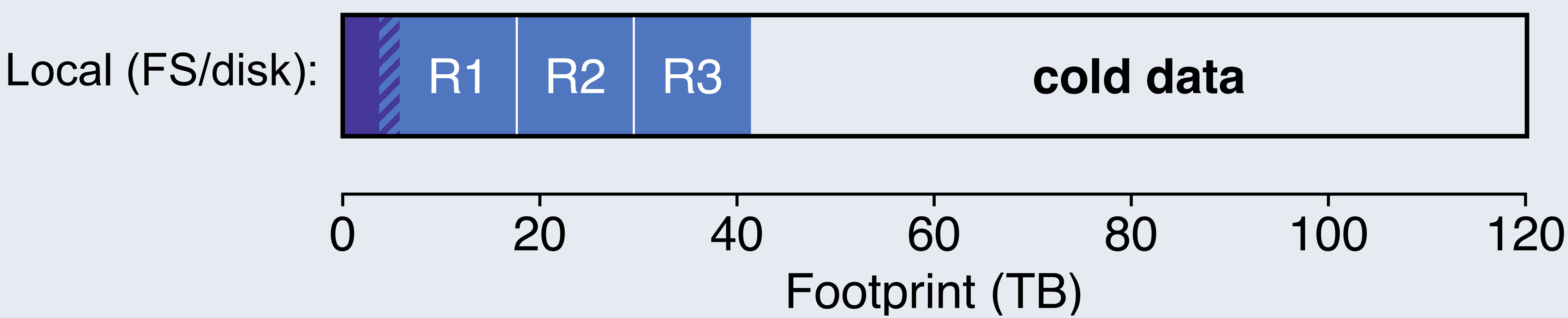
② Most touched data is only written


Cold Data
Baseline HDFS I/O:
All HDFS I/O:
compact
LOG
Local (FS/disk):
R1
R2
R3
0
20
40
Footprint (TB)

# Cold Data



# Cold Data



Local (FS/disk):
R1
R2
R3
cold data
0
20
40
60
80
100
120
Footprint (TB)

# Workload Analysis Conclusions

① Layers amplify writes: 1% => 64%

② Most touched data is only written

③ The dataset is large and cold: 2/3 of 120TB never touched

# Workload Analysis Questions

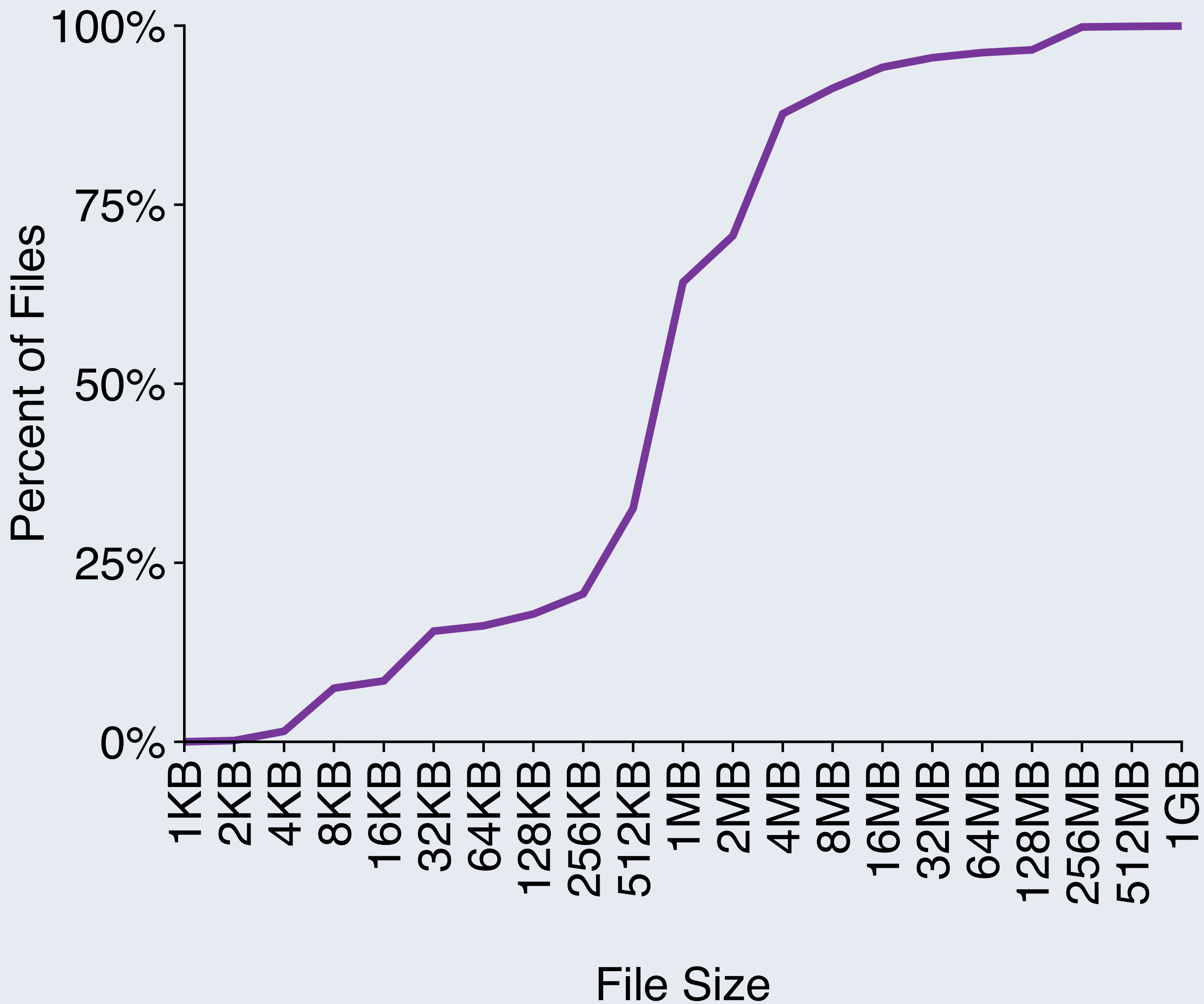
At each layer, what activities read or write?

How large is the dataset?

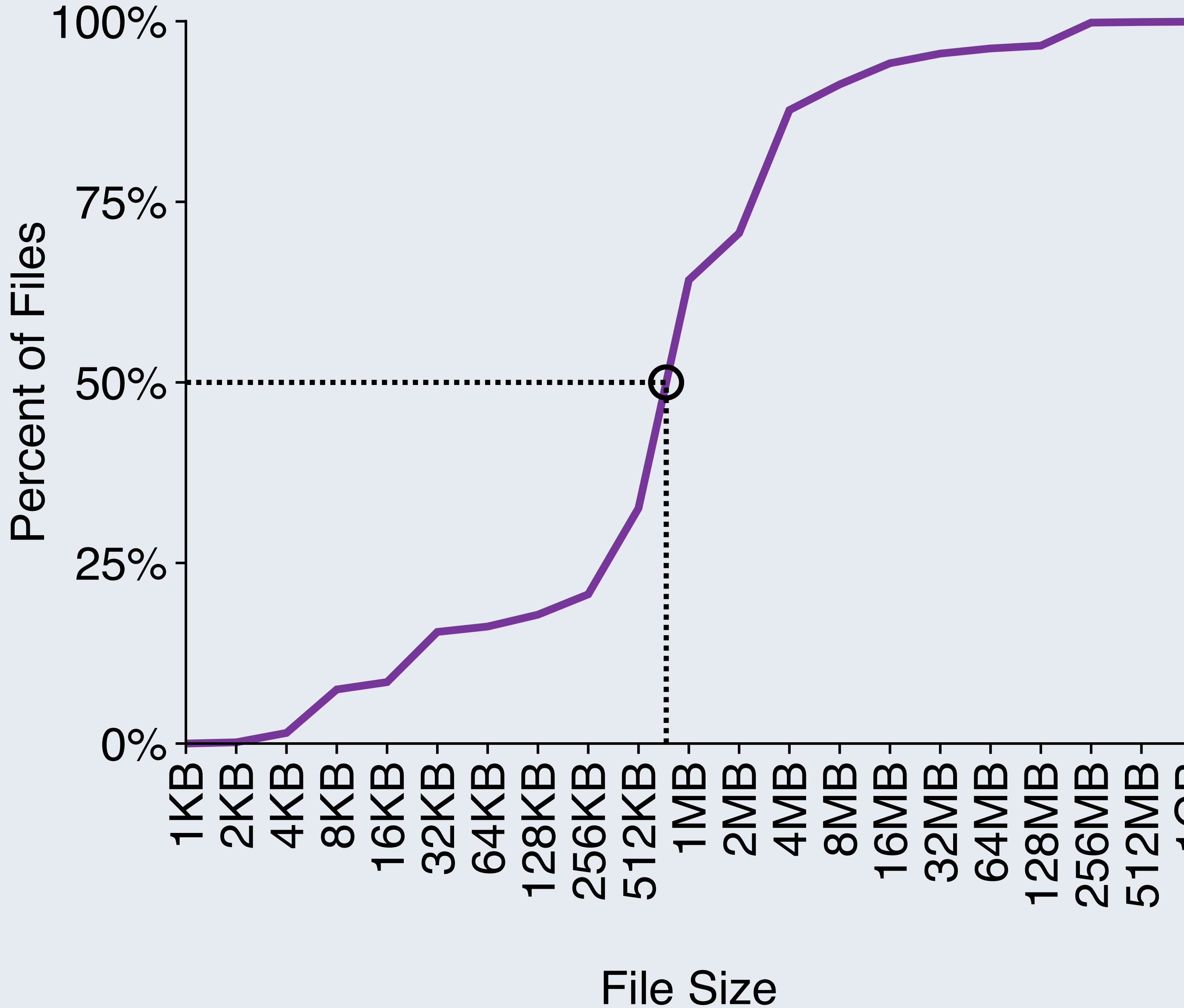
How large are created files?

How sequential is I/O?

# Created Files: Size Distribution

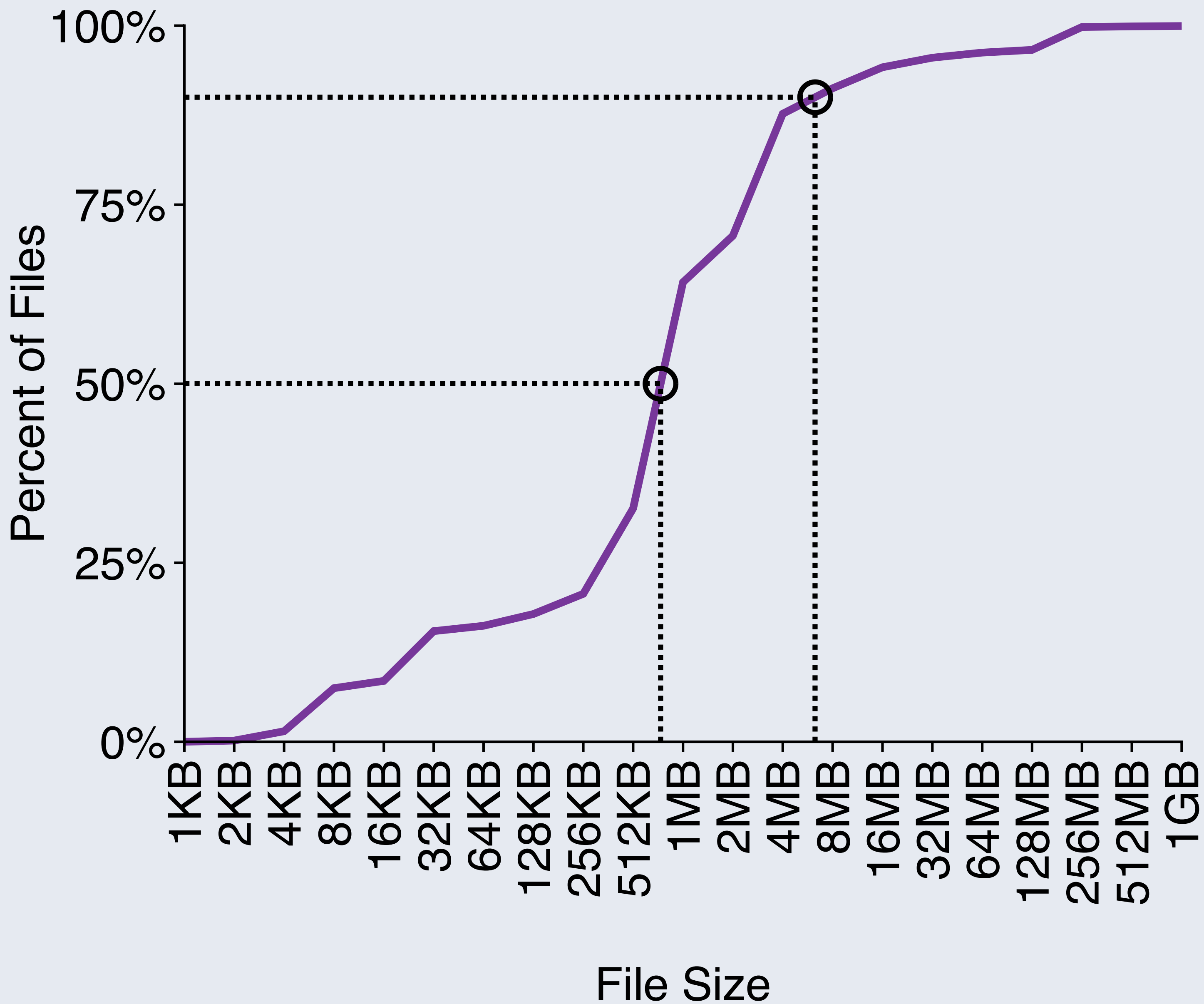


# Created Files: Size Distribution




50% of files are <750KB

# Created Files: Size Distribution




90% of files are <6.3MB

# Workload Analysis Conclusions

① Layers amplify writes: 1% => 64%

② Most touched data is only written

③ The dataset is large and cold: 2/3 of 120TB never touched

④ Files are very small: 90% smaller than 6.3MB

# Workload Analysis Questions

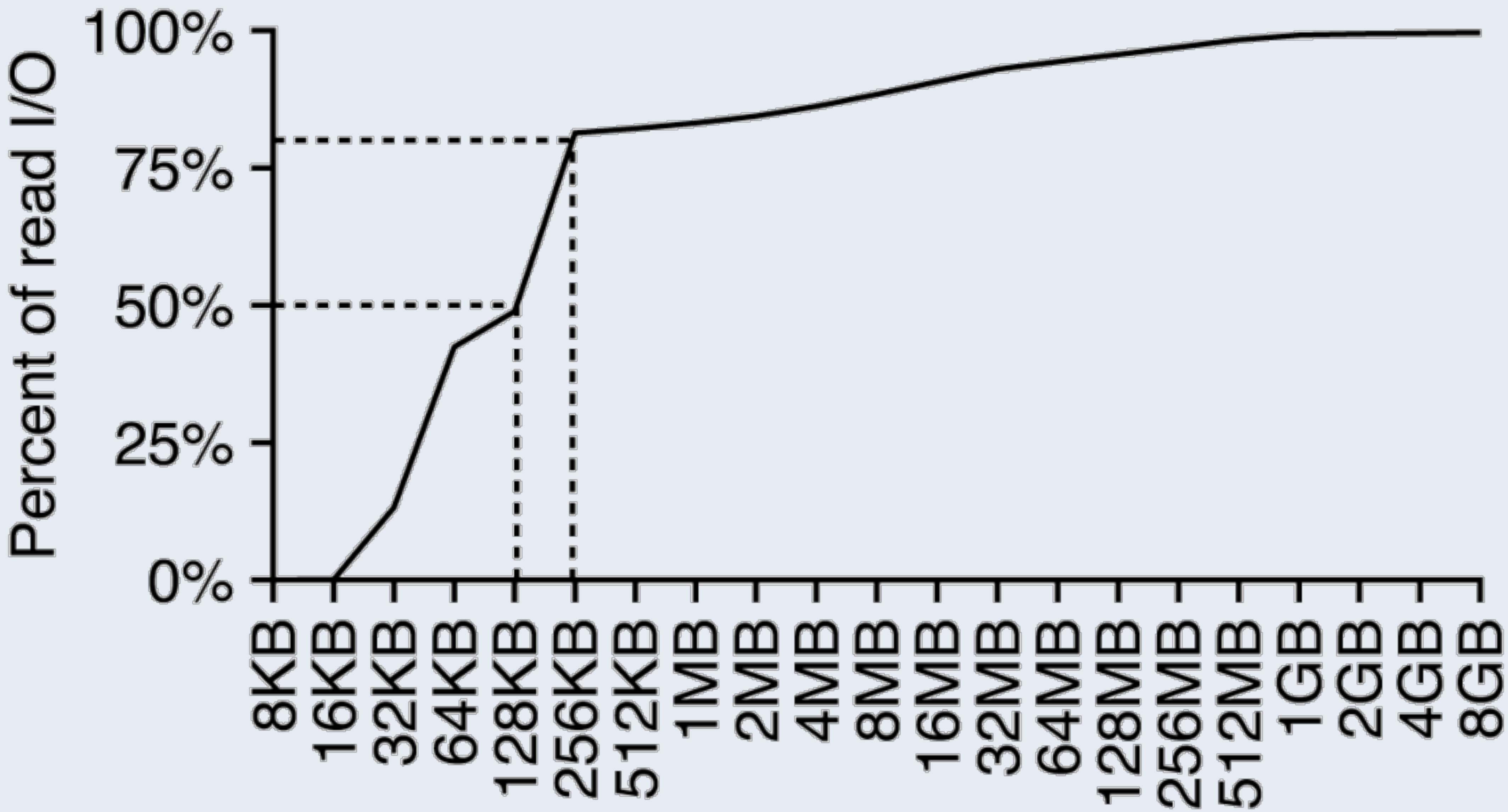
At each layer, what activities read or write?

How large is the dataset?

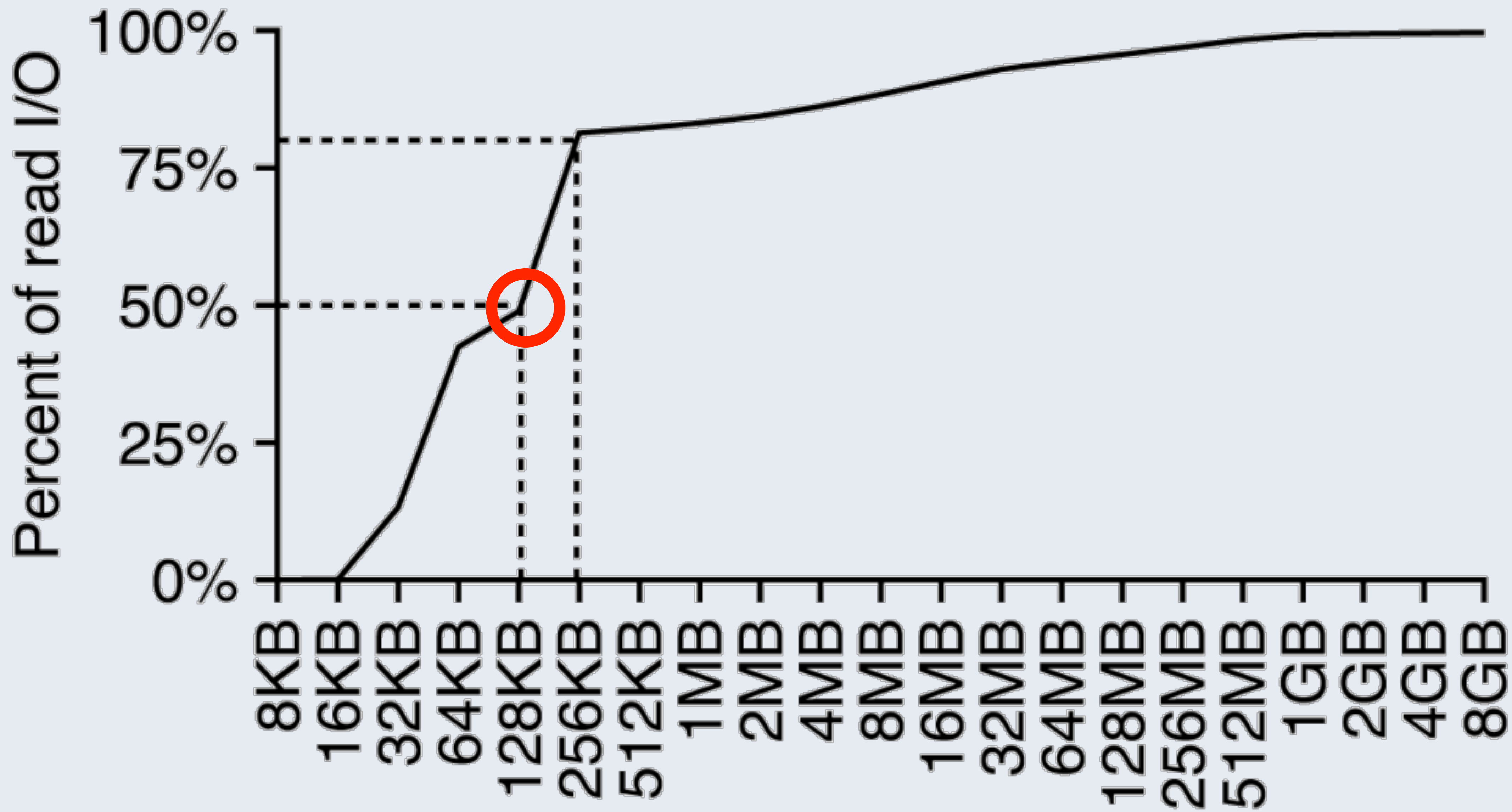
How large are created files?

How sequential is I/O?

# Reads: Run Size

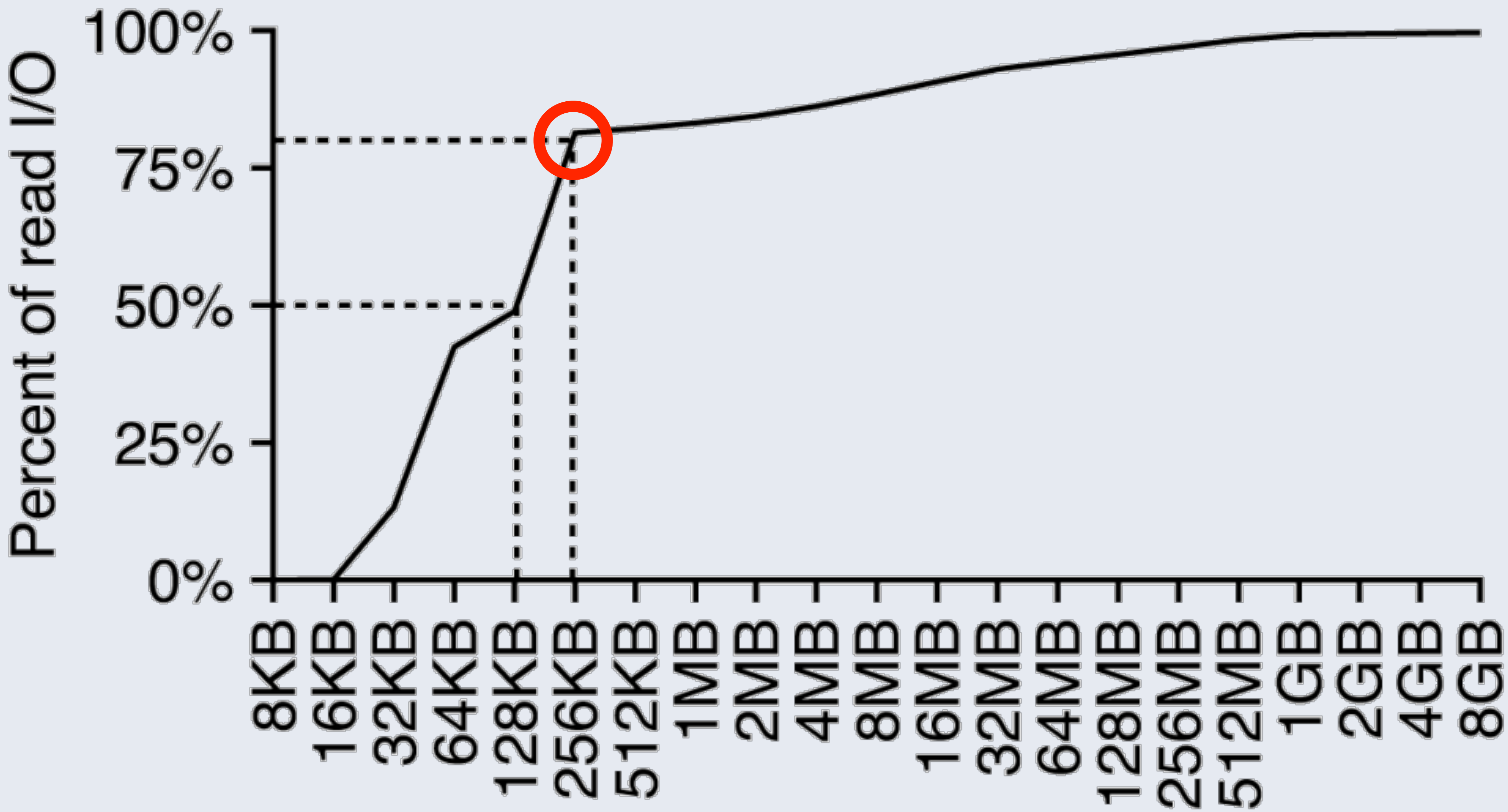


# Reads: Run Size




50% of runs (weighted by I/O) <130KB

# Reads: Run Size




80% of runs (weighted by I/O) <250KB

# Workload Analysis Conclusions

① Layers amplify writes: 1% => 64%

② Data is read or written, but rarely both

③ The dataset is large and cold: 2/3 of 120TB never touched

④ Files are very small: 90% smaller than 6.3MB

⑤ Fairly random I/O: 130KB median read run

# Outline

Intro

- Messages stack overview
- Methodology: trace-driven analysis and simulation
- HBase background

Results

- Workload analysis
- Hardware simulation: adding a flash layer
- Software simulation: integrating layers

Conclusions

# Hardware Architecture: Workload Implications

Option 1: pure disk

Option 2: pure flash

Option 3: hybrid

# Hardware Architecture: Workload Implications

Option 1: pure disk

- Very random reads
- Small files

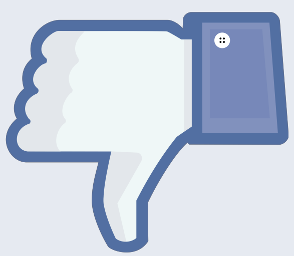
Option 2: pure flash

Option 3: hybrid

# Hardware Architecture: Workload Implications

Option 1: pure disk

- Very random reads
- Small files



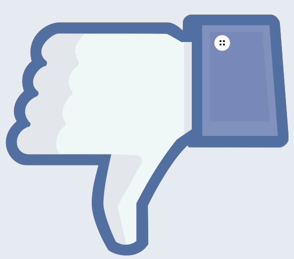
Option 2: pure flash

Option 3: hybrid

# Hardware Architecture: Workload Implications

Option 1: pure disk

- Very random reads
- Small files



Option 2: pure flash

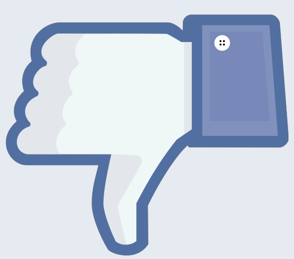
- Large dataset
- Mostly very cold
- >$10K / machine

Option 3: hybrid

# Hardware Architecture: Workload Implications

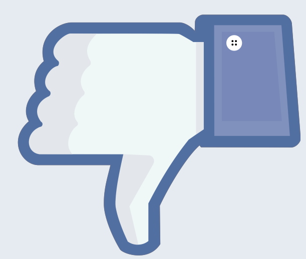
Option 1: pure disk

- Very random reads
- Small files



Option 2: pure flash

- Large dataset
- Mostly very cold
- >$10K / machine

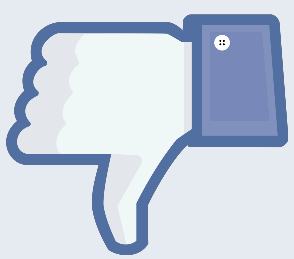


Option 3: hybrid

# Hardware Architecture: Workload Implications

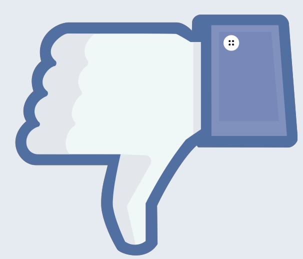
Option 1: pure disk

- Very random reads
- Small files



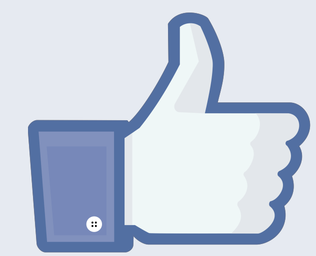
Option 2: pure flash

- Large dataset
- Mostly very cold
- >$10K / machine



Option 3: hybrid

- Process of elimination



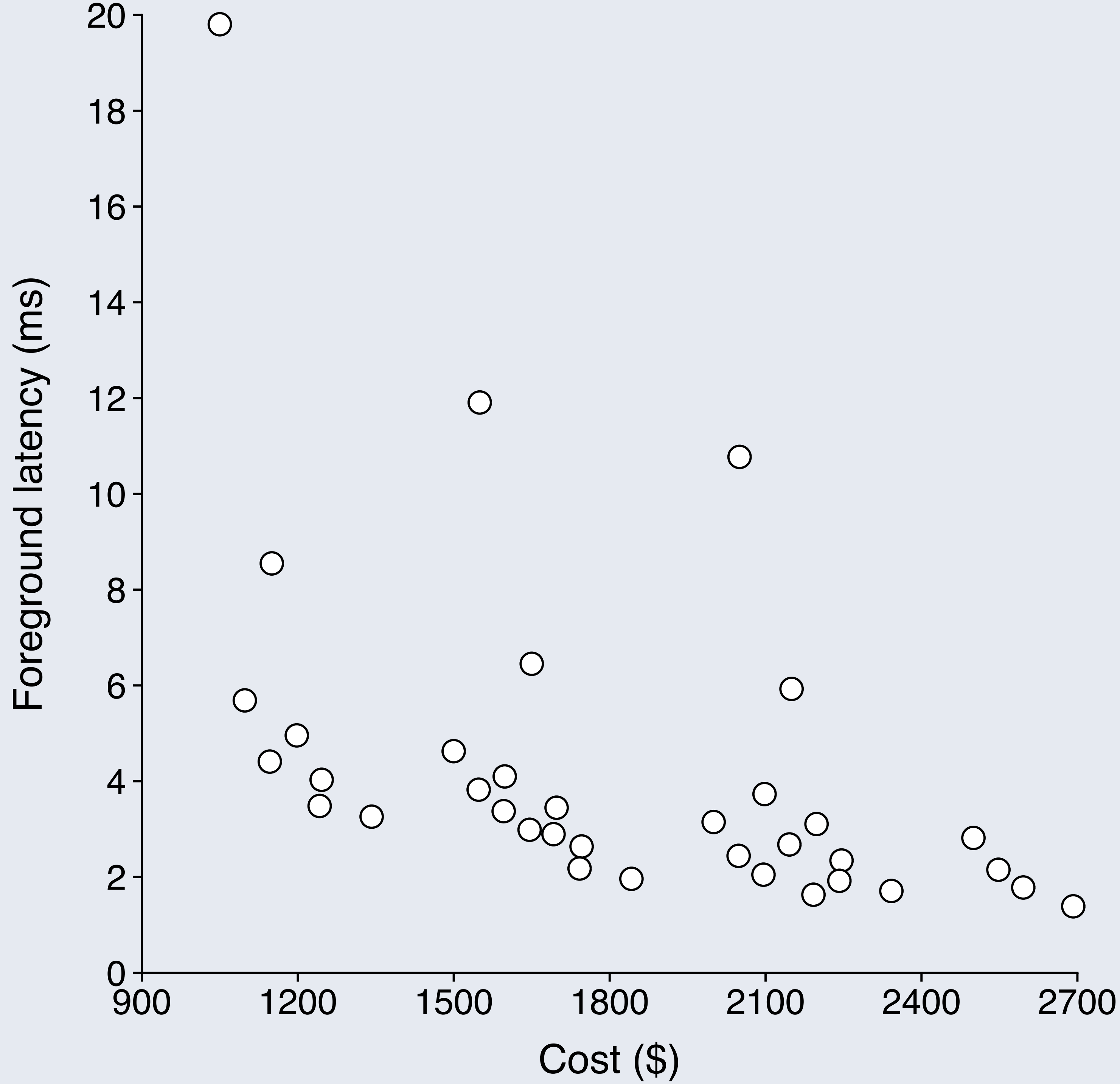
# Hardware Architecture: Simulation Results

Evaluate <u>cost</u> and <u>performance</u> of 36 hardware combinations (3x3x4)

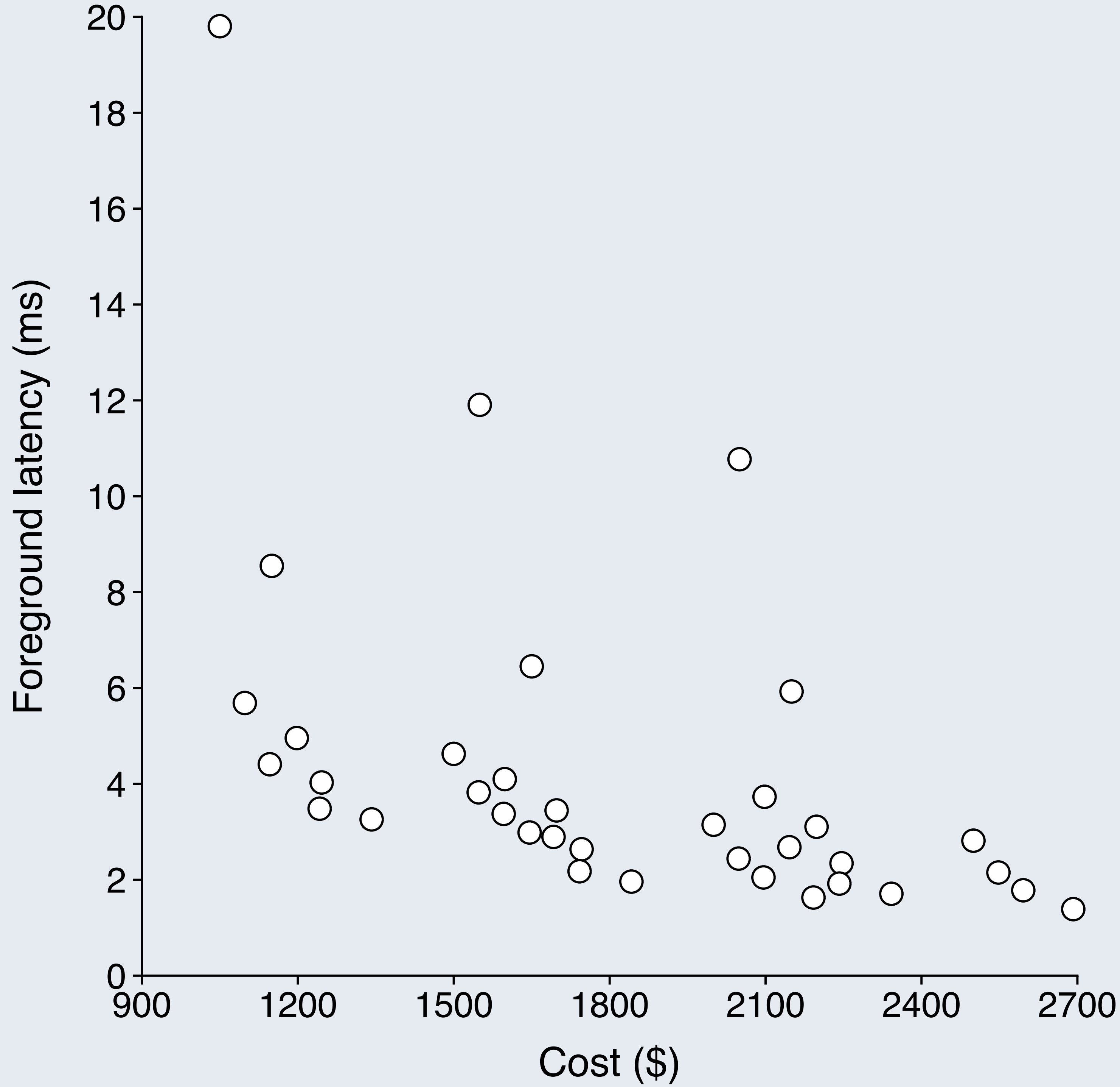
- Disks: 10, 15, or 20
- RAM (cache): 10, 30, or 100GB
- Flash (cache): 0, 60, 120, or 240GB

Assumptions:

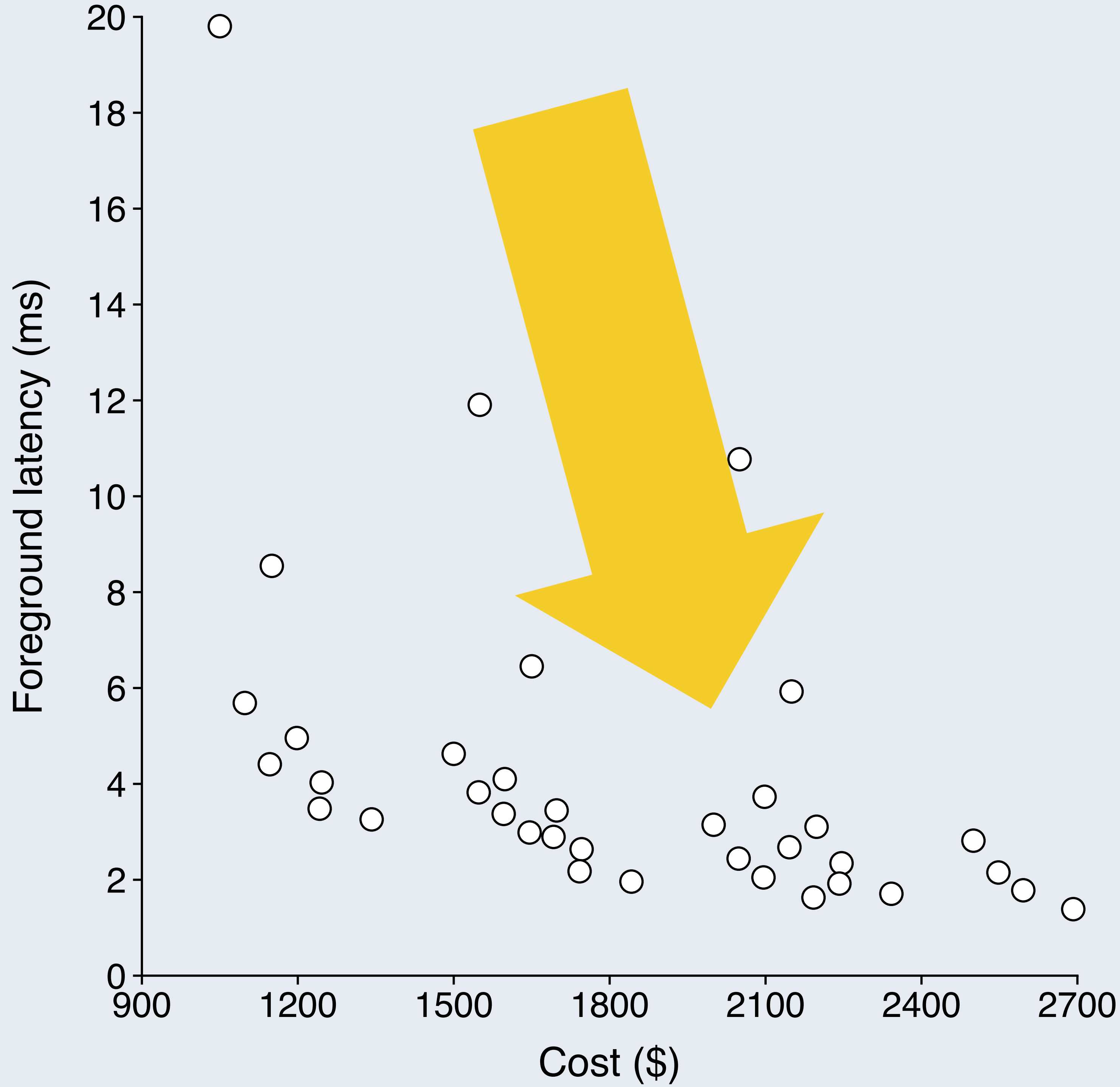
| Hardware | Cost | Performance |
|---|---|---|
| HDD | $100/disk | 10ms seek, 100MB/s |
| RAM | $5/GB | zero latency |
| Flash | $0.8/GB | 0.5ms |



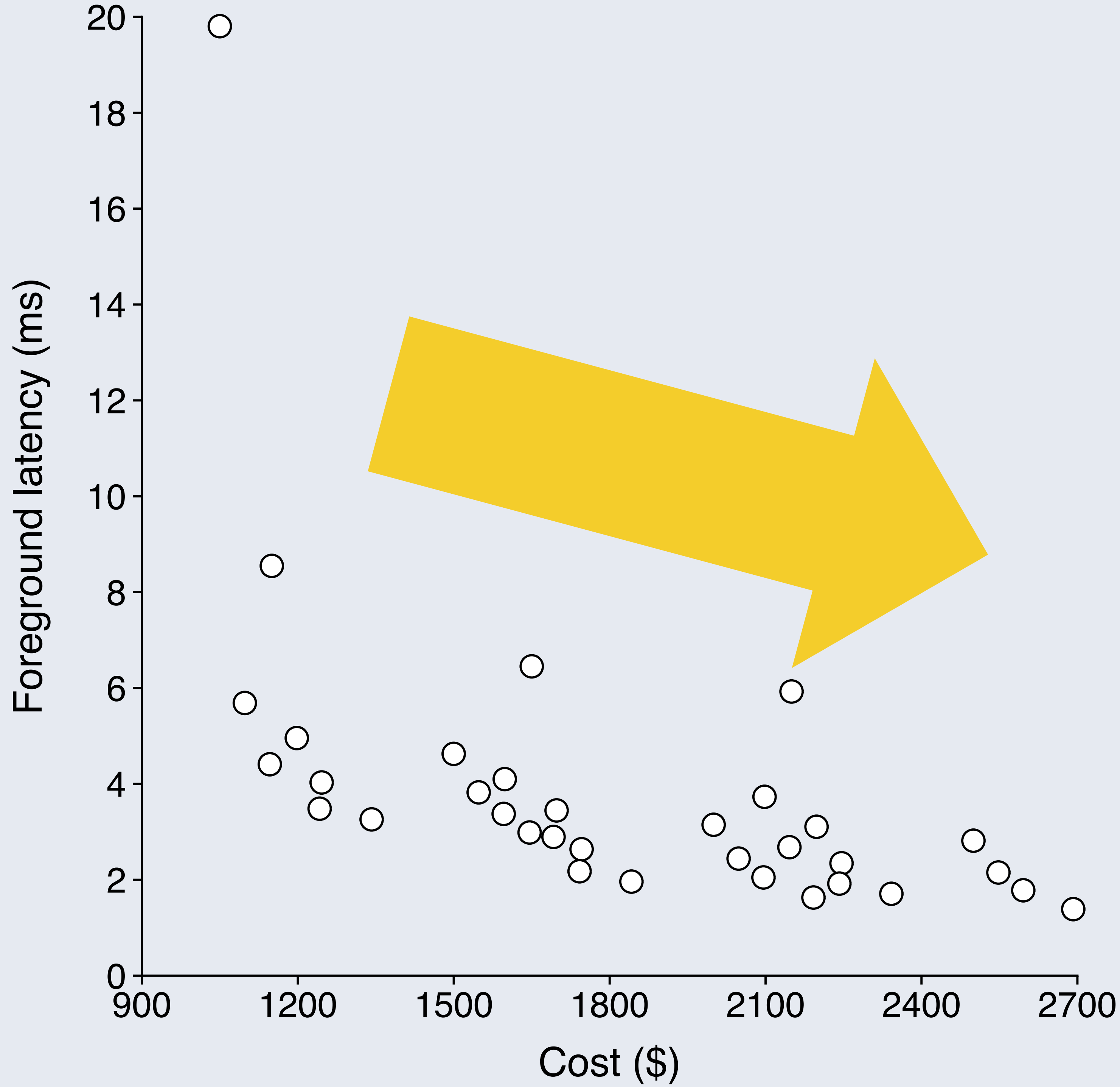

Cost/performance tradeoff for 36 hardware combinations




Upgrades decrease latency but increase cost

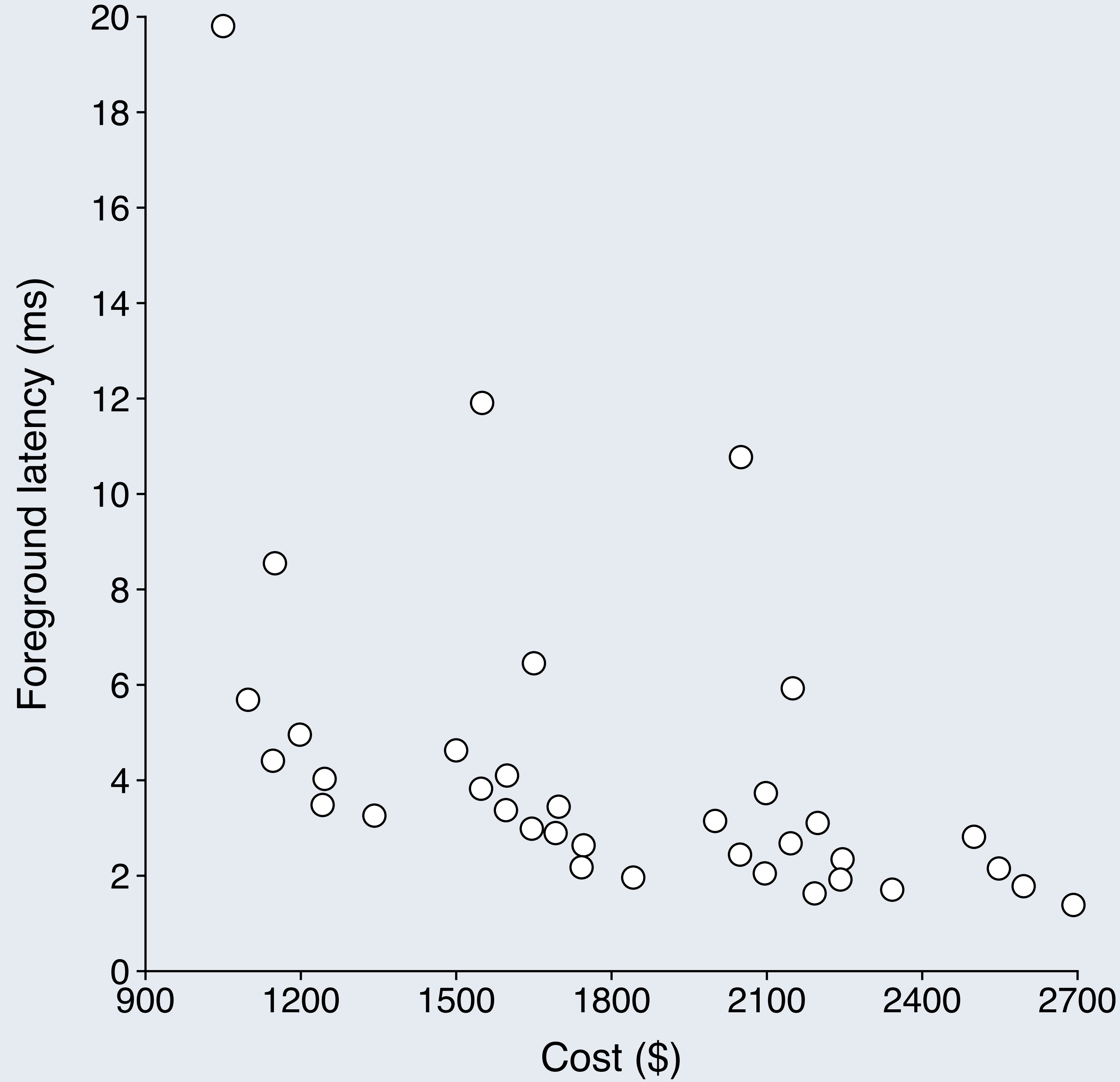



Good upgrade

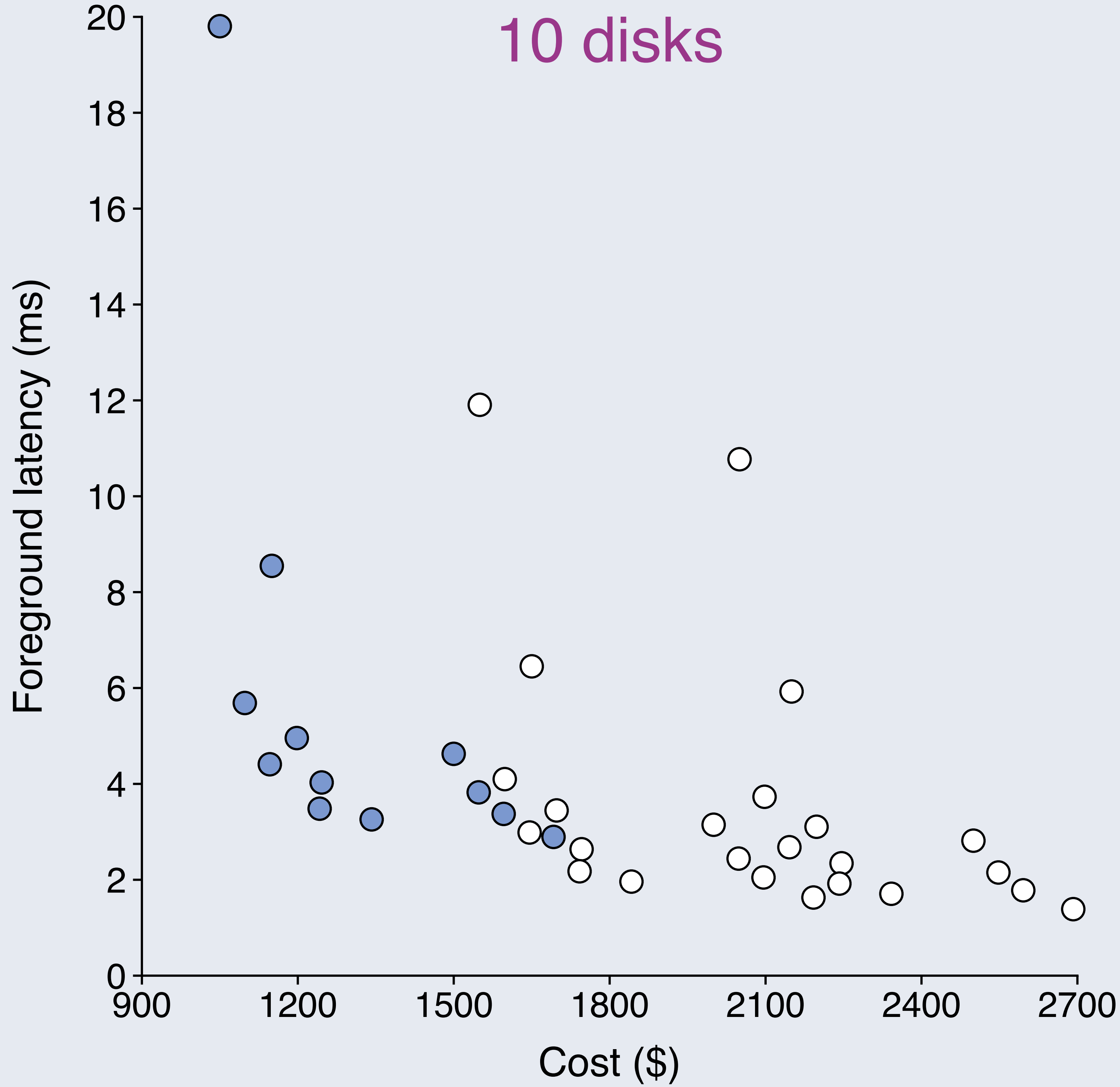


Foreground latency (ms)
20
18
16
14
12
10
8
6
4
2
0
900
1200
1500
1800
2100
2400
2700
Cost ($)

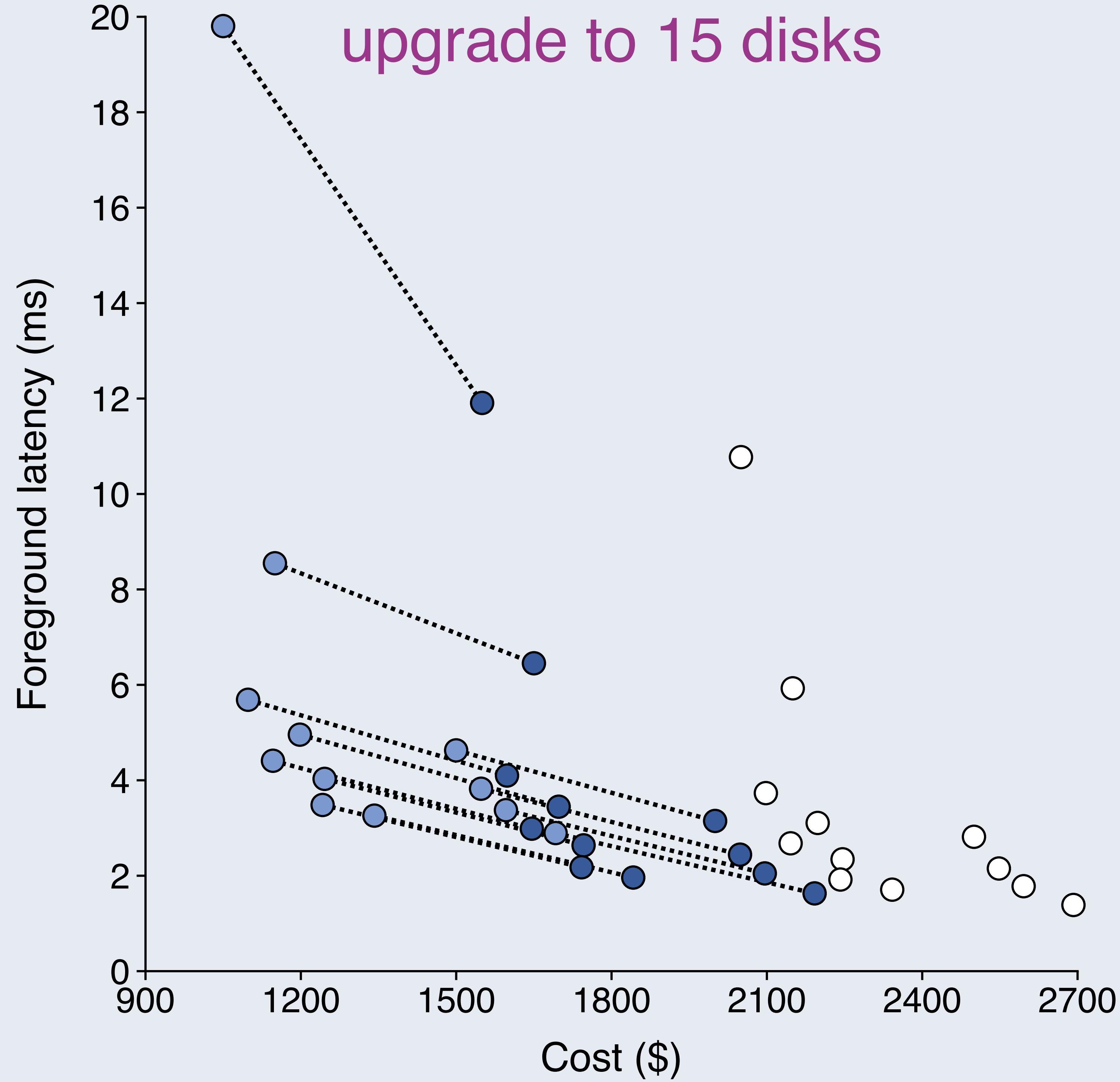

Bad upgrade



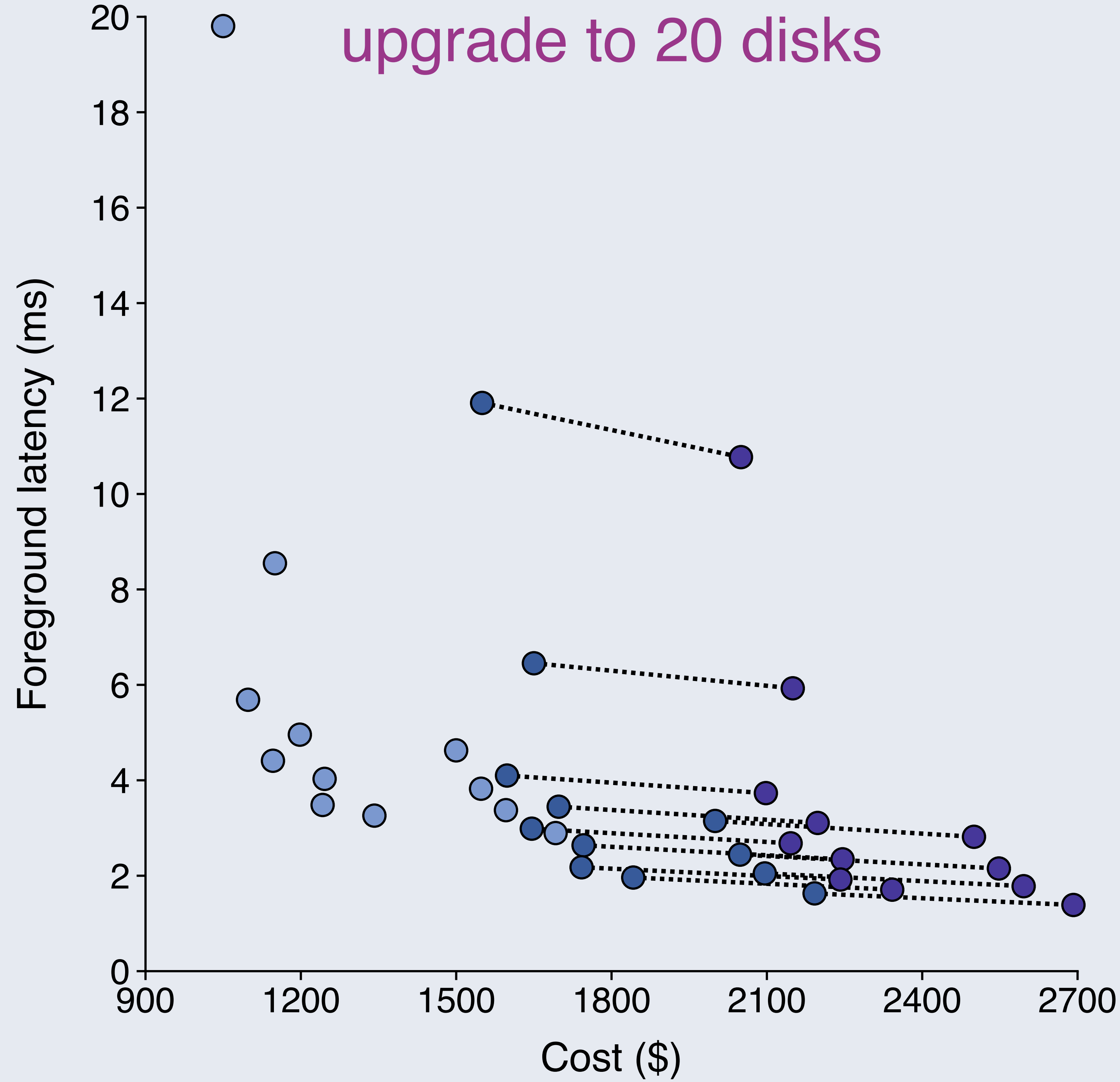
Foreground latency (ms)
20
18
16
14
12
10
8
6
4
2
0
900
1200
1500
1800
2100
2400
2700
Cost ($)

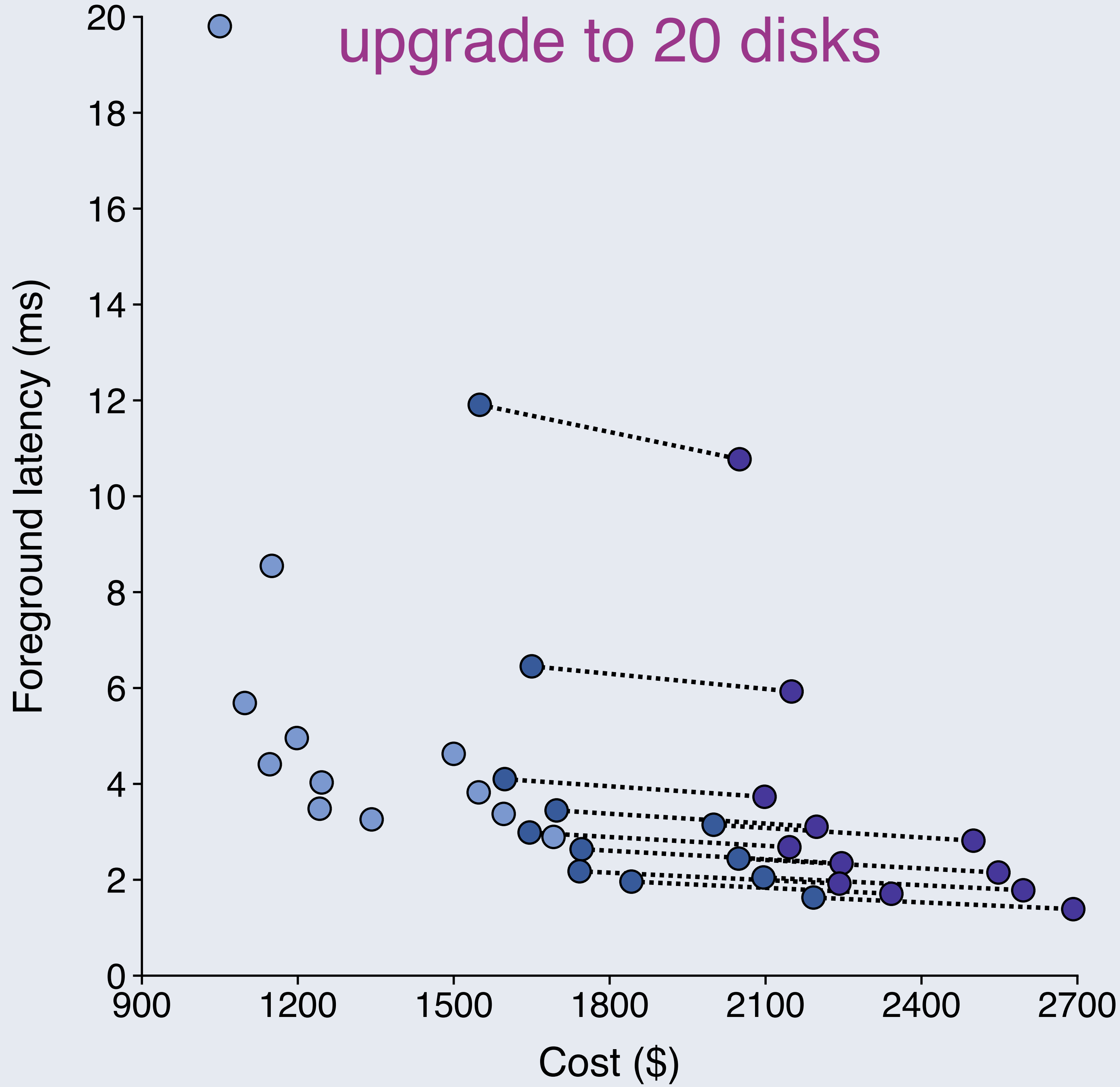


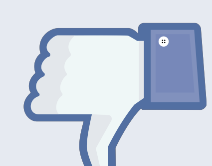
10 disks
Foreground latency (ms)
Cost ($)
0
2
4
6
8
10
12
14
16
18
20
900
1200
1500
1800
2100
2400
2700

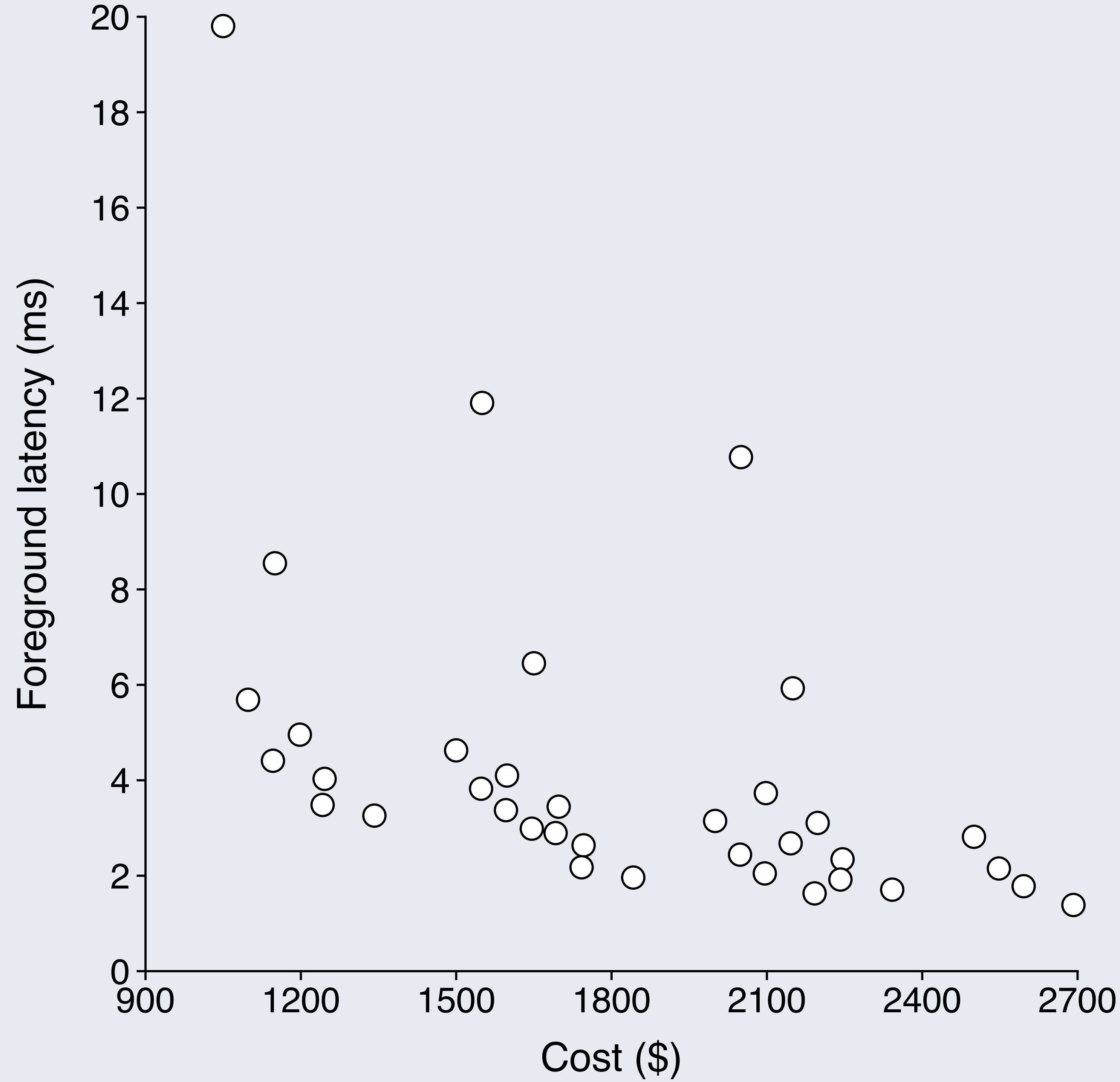


upgrade to 15 disks
Foreground latency (ms)
Cost ($)
0
2
4
6
8
10
12
14
16
18
20
900
1200
1500
1800
2100
2400
2700

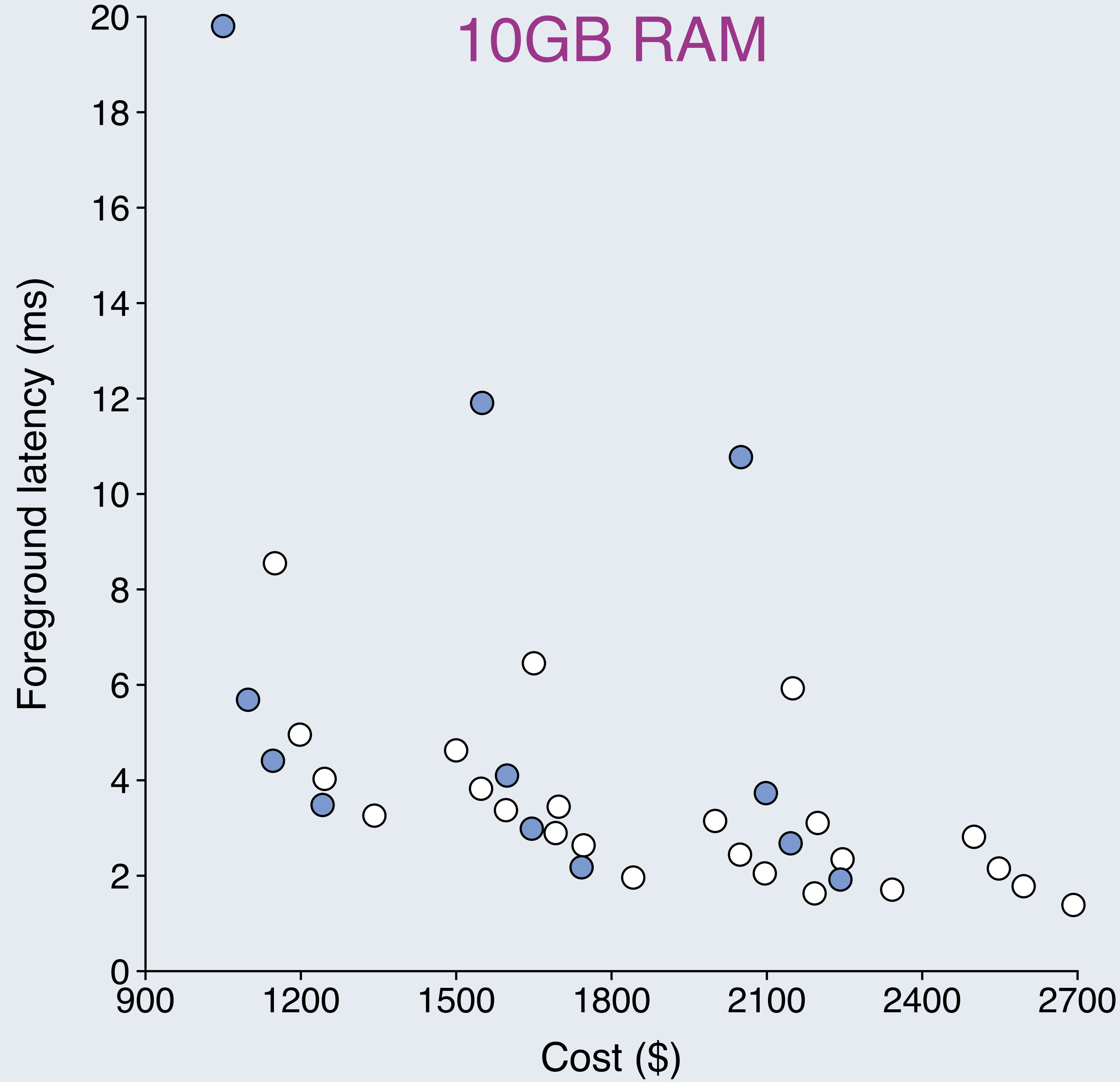

upgrade to 20 disks
Foreground latency (ms)
Cost ($)
0
2
4
6
8
10
12
14
16
18
20
900
1200
1500
1800
2100
2400
2700

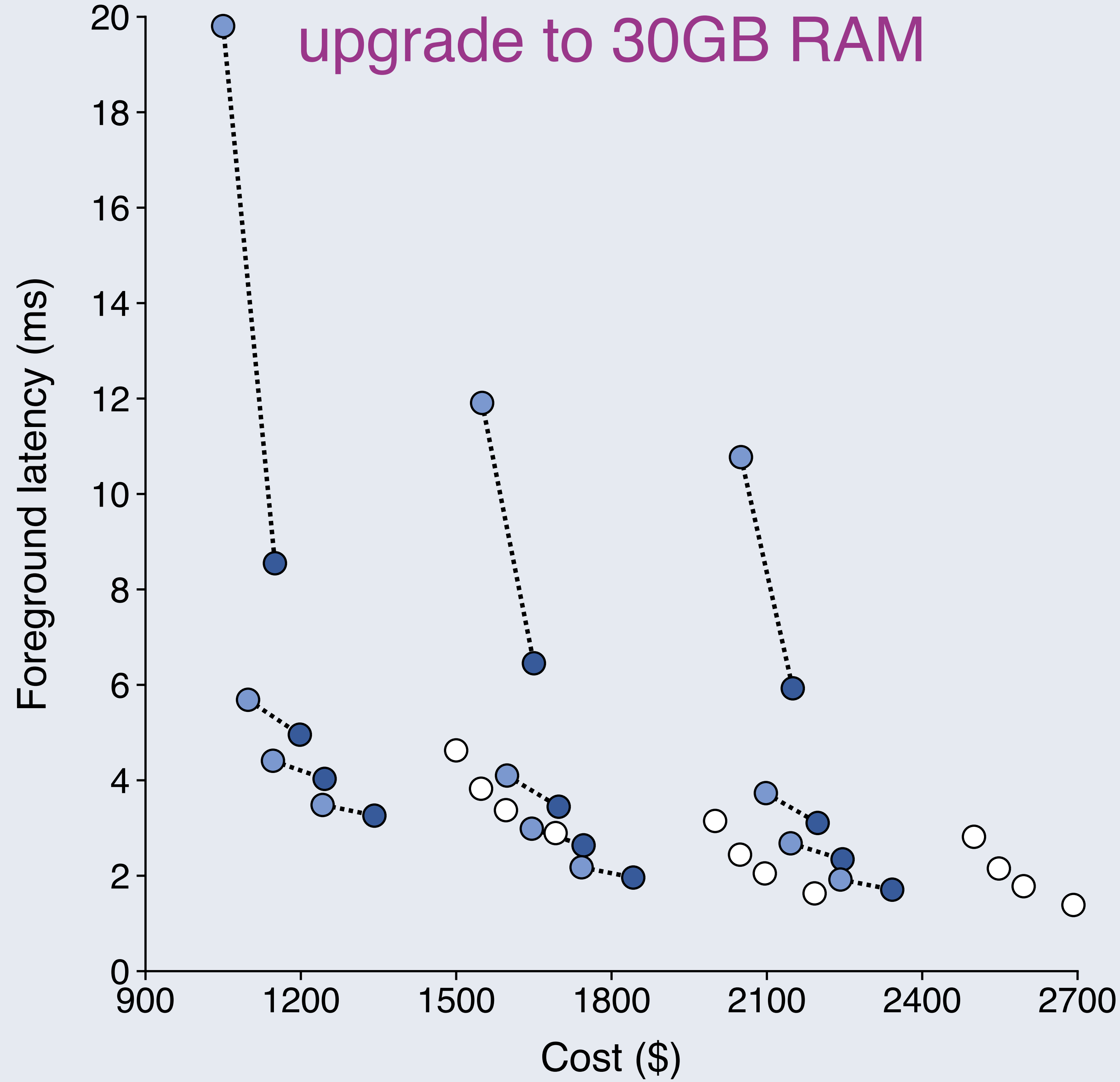



Upgrading disk: 



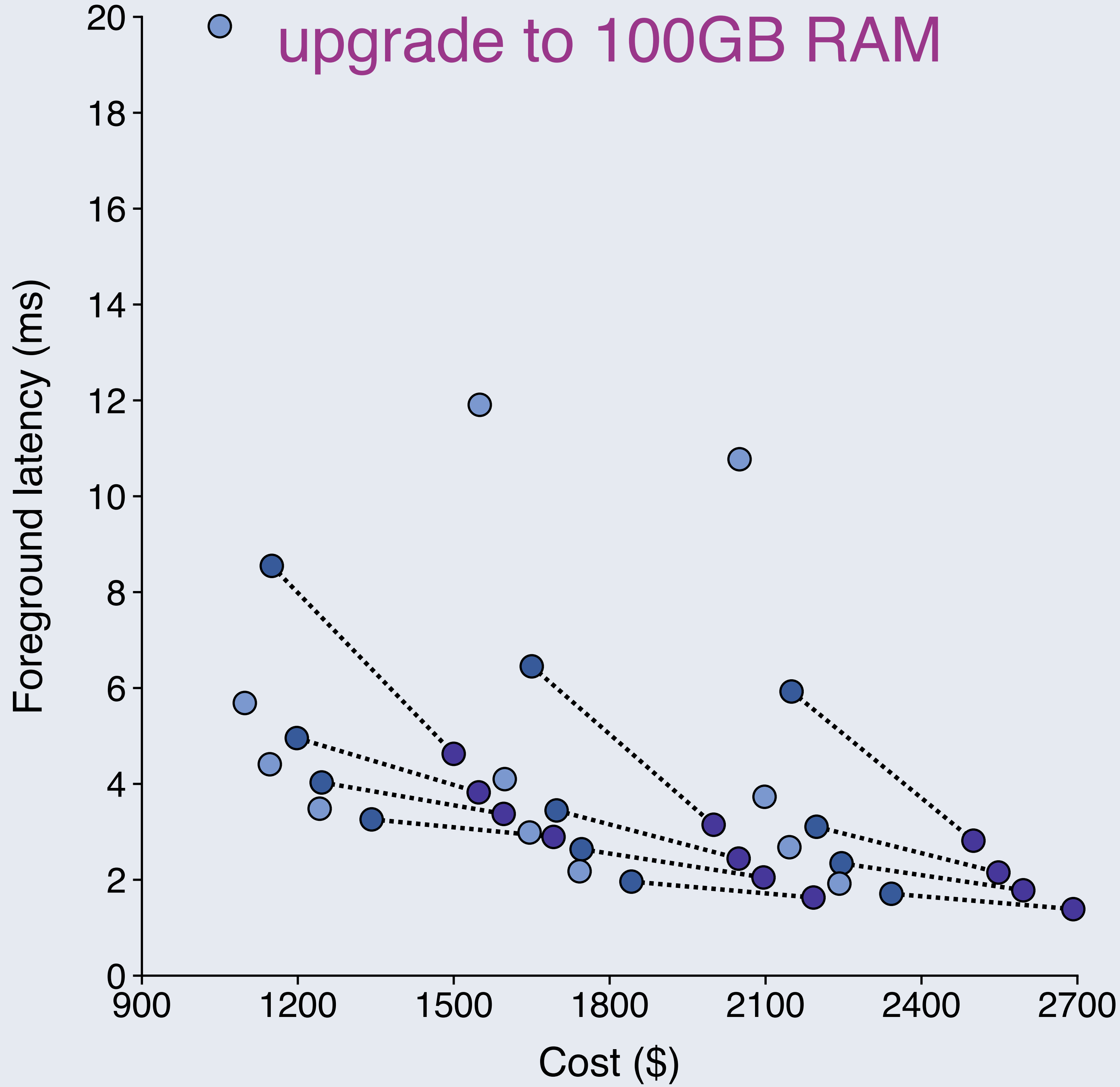
Foreground latency (ms)
20
18
16
14
12
10
8
6
4
2
0
900
1200
1500
1800
2100
2400
2700
Cost ($)


10GB RAM
Foreground latency (ms)
Cost ($)
0
2
4
6
8
10
12
14
16
18
20
900
1200
1500
1800
2100
2400
2700

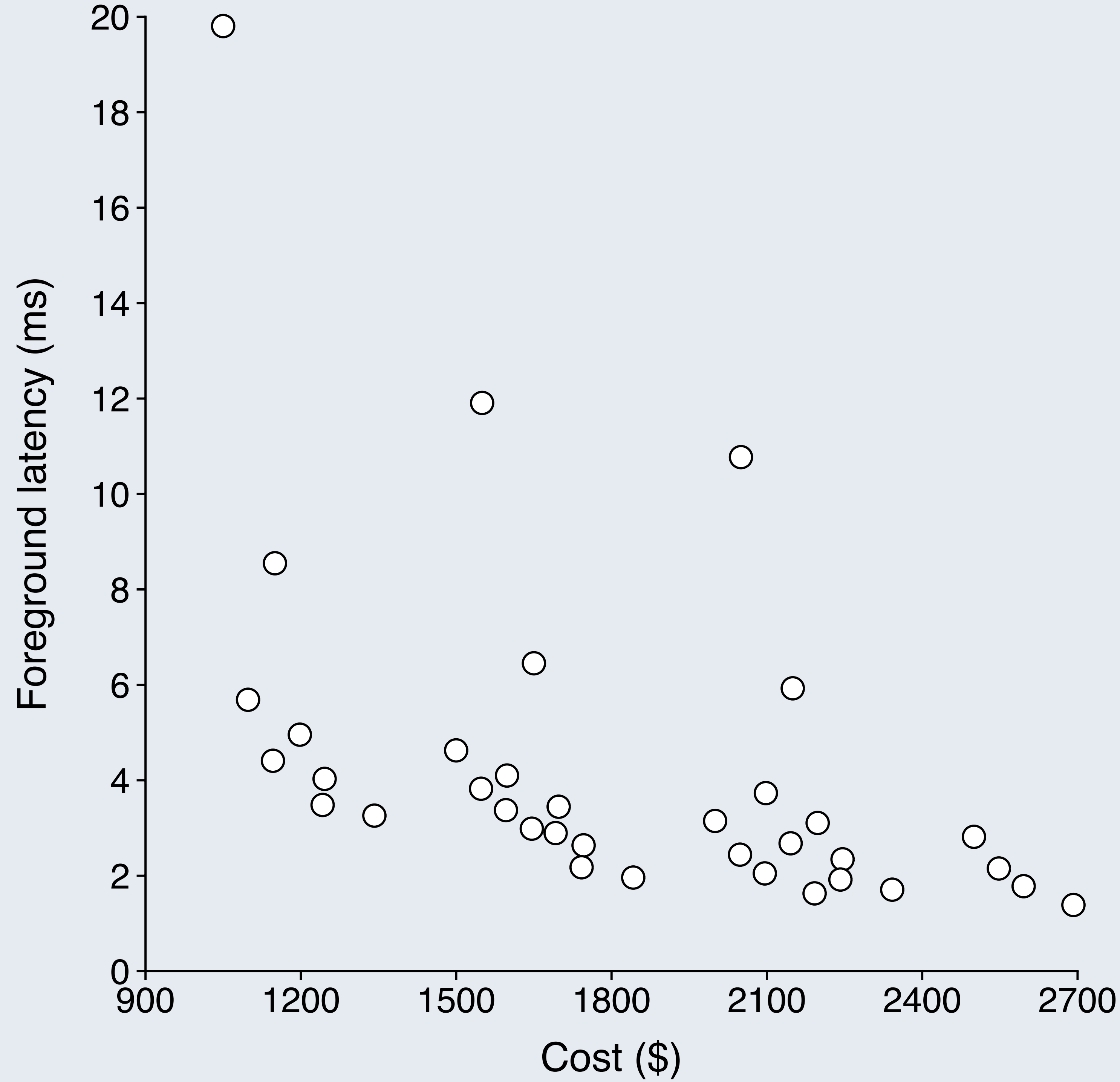


upgrade to 30GB RAM
Foreground latency (ms)
0
2
4
6
8
10
12
14
16
18
20
900
1200
1500
1800
2100
2400
2700
Cost ($)

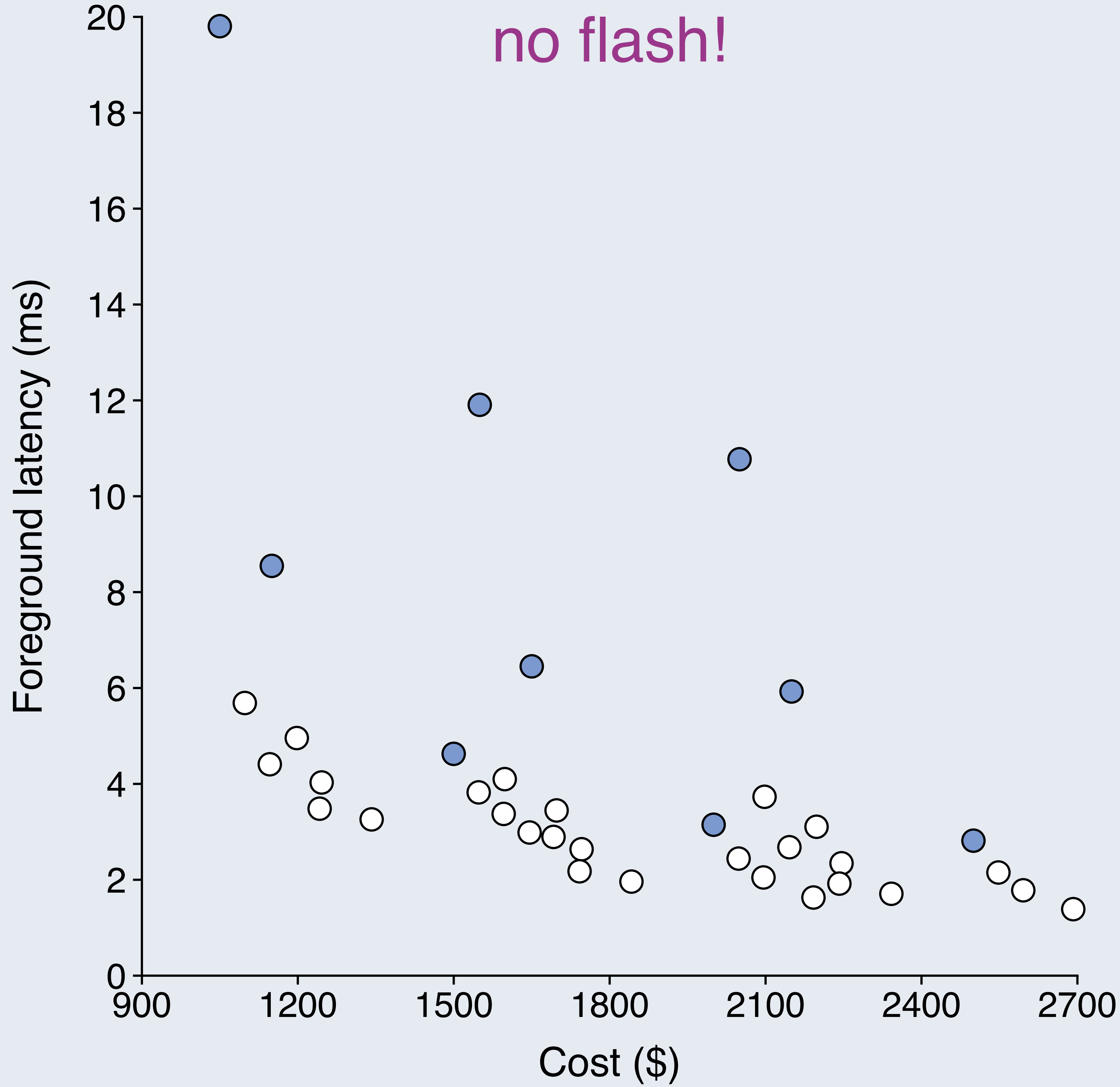

upgrade to 100GB RAM
Foreground latency (ms)
Cost ($)
0
2
4
6
8
10
12
14
16
18
20
900
1200
1500
1800
2100
2400
2700



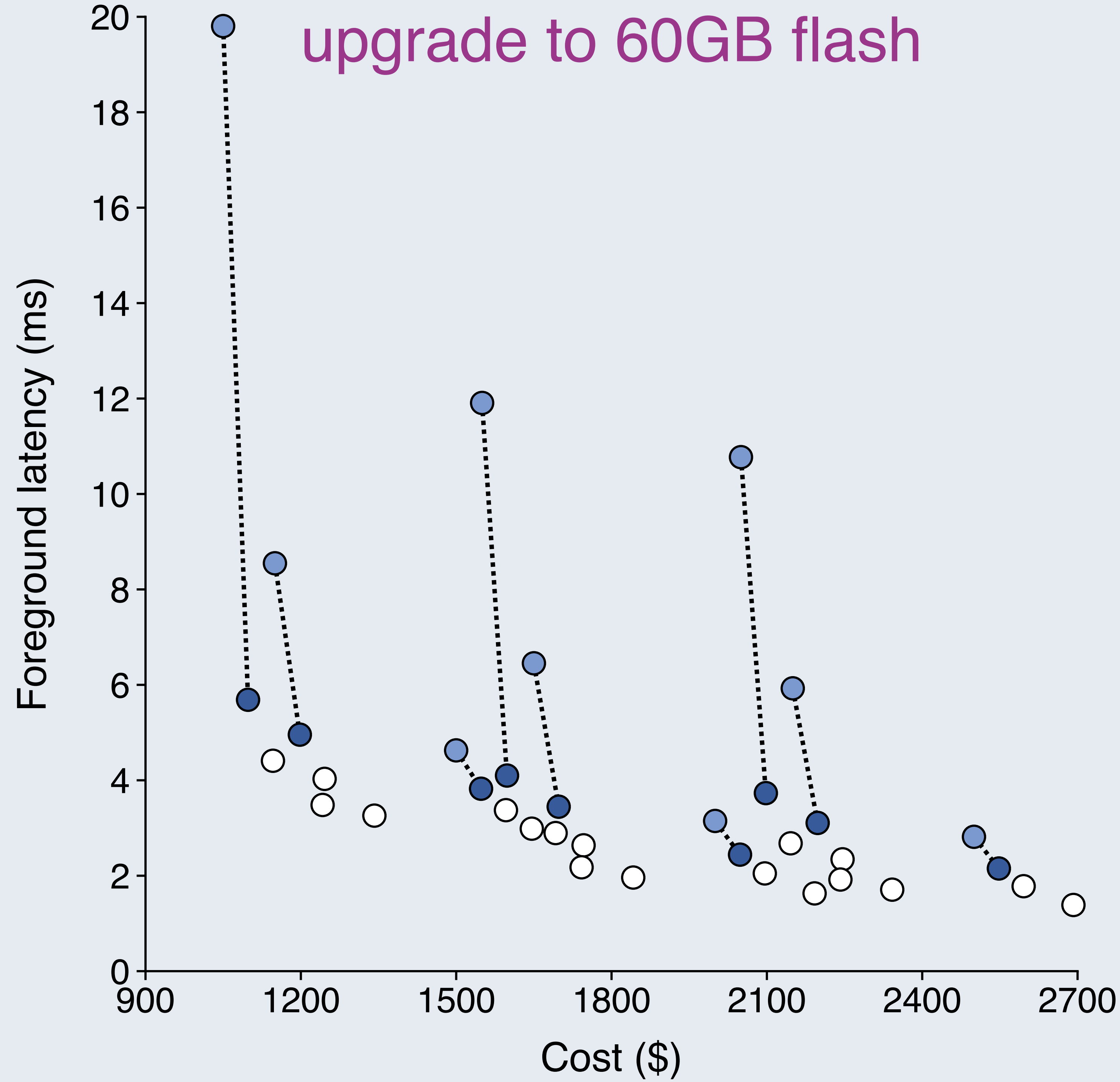

Upgrading RAM:



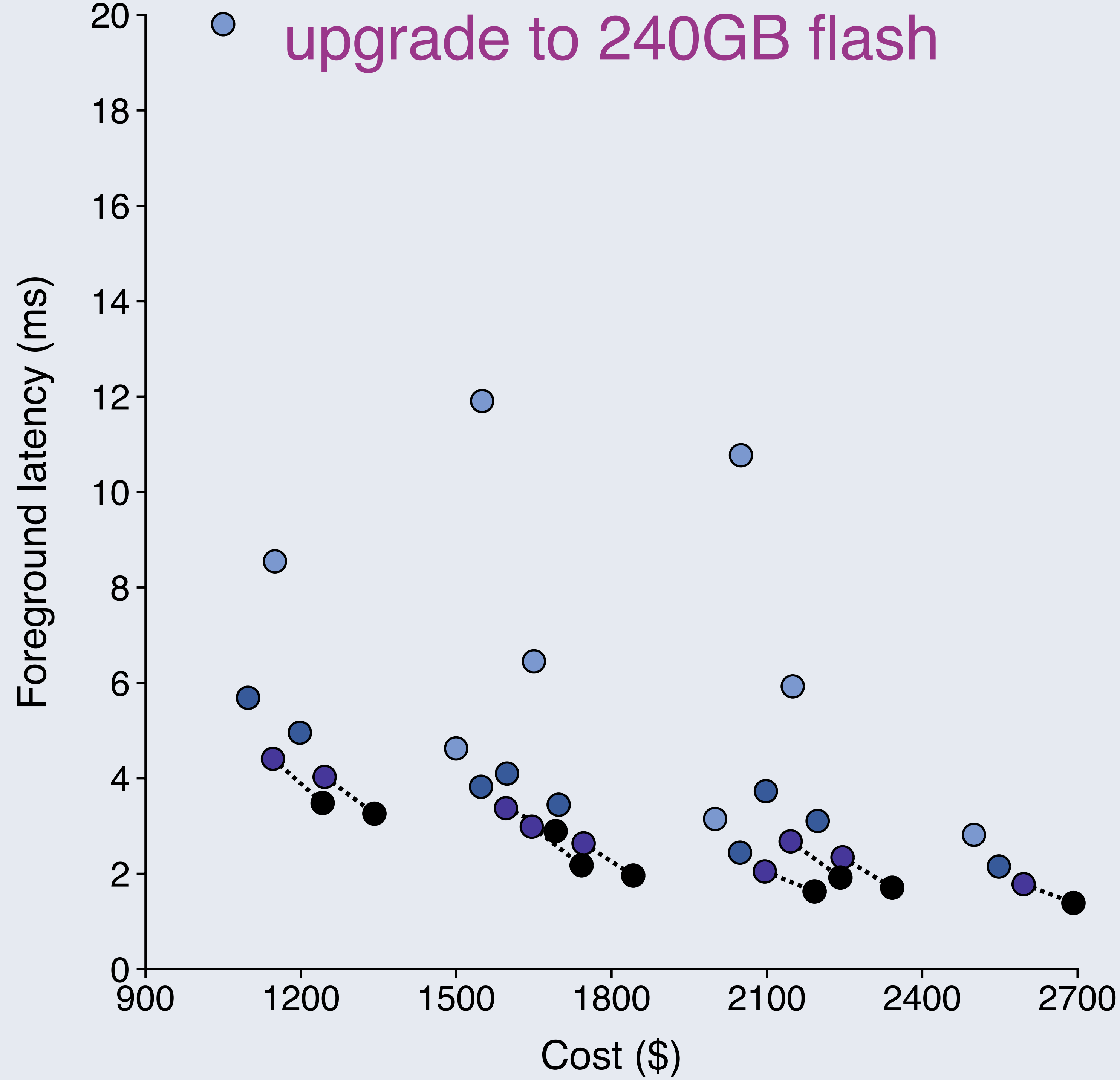
Foreground latency (ms)
20
18
16
14
12
10
8
6
4
2
0
900
1200
1500
1800
2100
2400
2700
Cost ($)

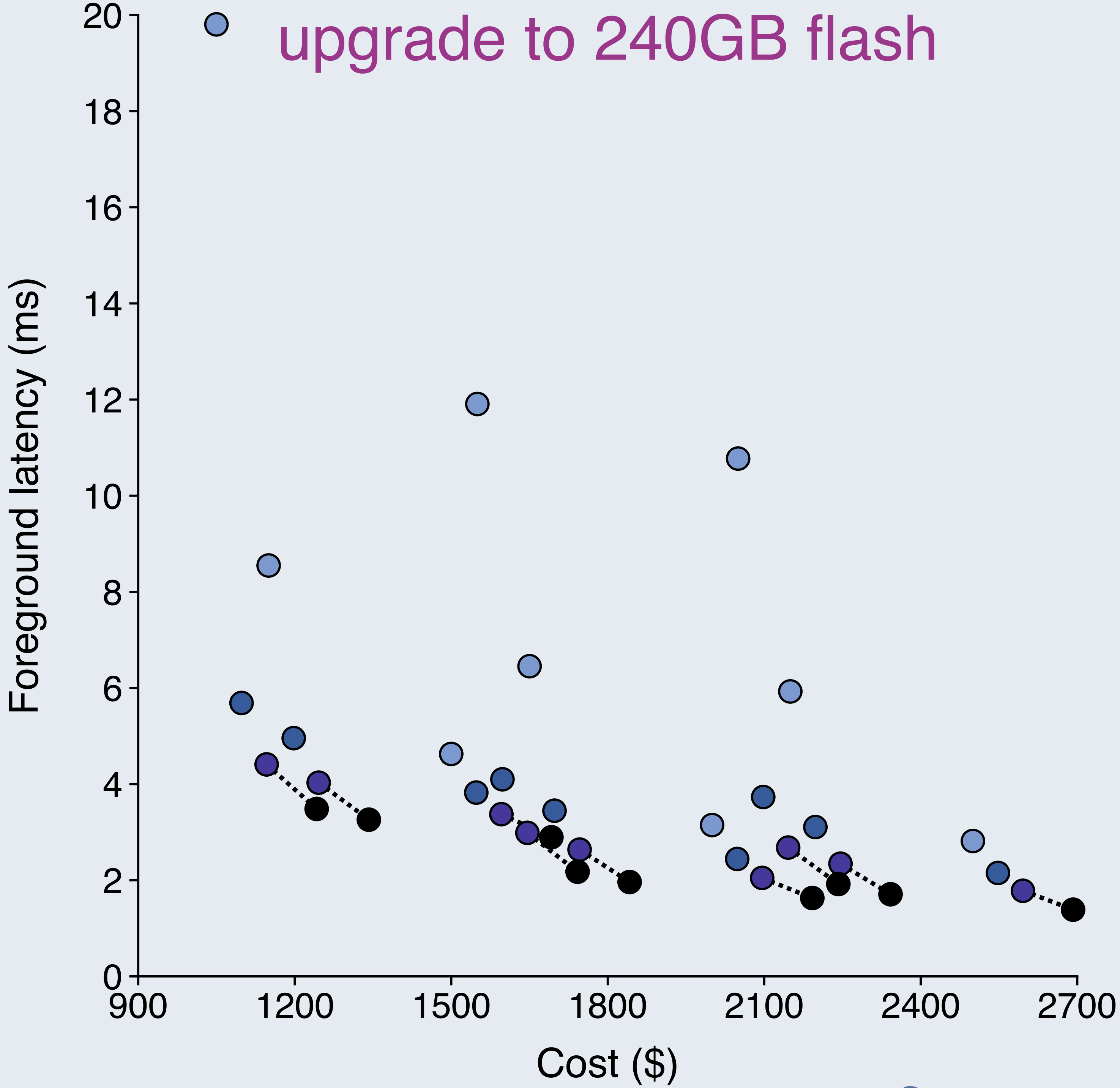


no flash!
Foreground latency (ms)
0
2
4
6
8
10
12
14
16
18
20
Cost ($)
900
1200
1500
1800
2100
2400
2700


upgrade to 60GB flash
Foreground latency (ms)
20
18
16
14
12
10
8
6
4
2
0
900
1200
1500
1800
2100
2400
2700
Cost ($)

upgrade to 120GB flash
Foreground latency (ms)
Cost ($)
0
2
4
6
8
10
12
14
16
18
20
900
1200
1500
1800
2100
2400
2700


upgrade to 240GB flash
Foreground latency (ms)
0
2
4
6
8
10
12
14
16
18
20
900
1200
1500
1800
2100
2400
2700
Cost ($)




Upgrading flash:

# Outline

Intro

- Messages stack overview
- Methodology: trace-driven analysis and simulation
- HBase background

Results

- Workload analysis
- Hardware simulation: adding a flash layer
- Software simulation: integrating layers

Conclusions

# Software Architecture: Workload Implications

Writes are amplified

- 1% at HDFS (excluding overheads) to 64% at disk (given 30GB RAM)
- We should optimize writes

# Software Architecture: Workload Implications

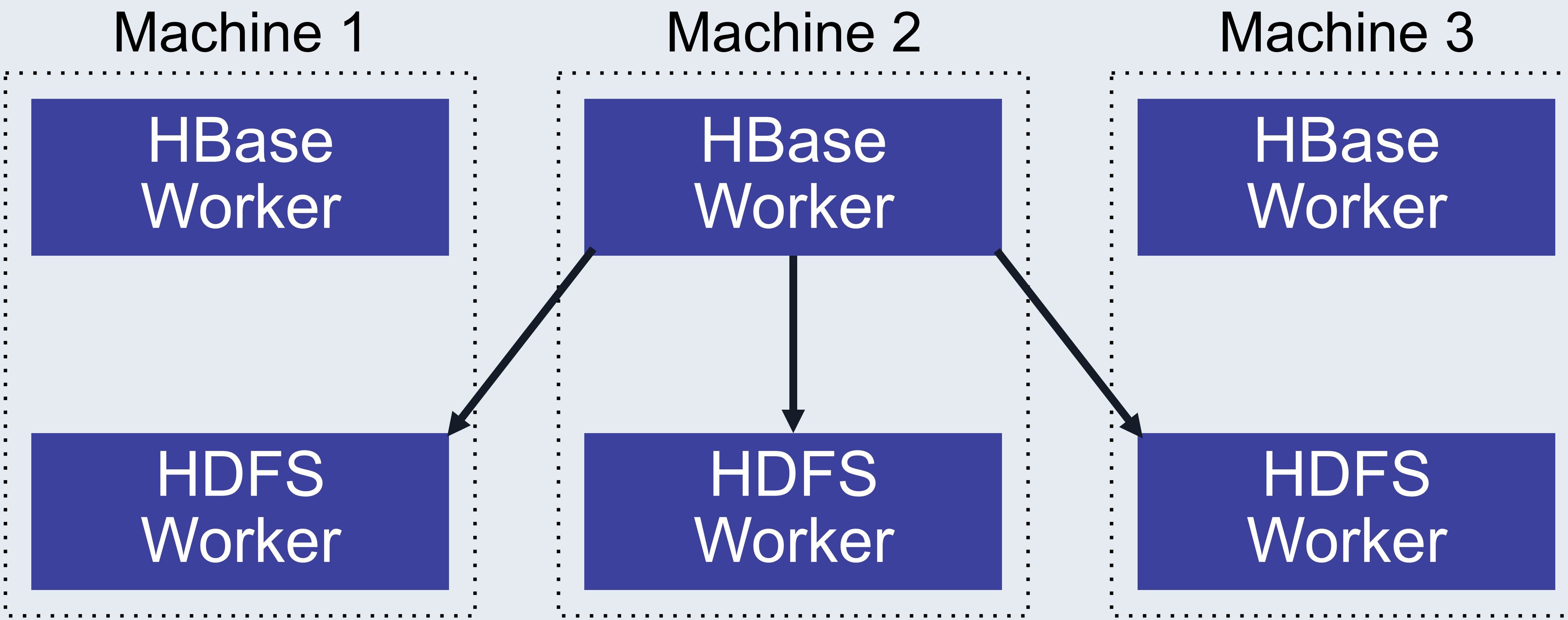
Writes are greatly amplified

- 1% at HDFS (excluding overheads) to 64% at disk
- We should optimize writes

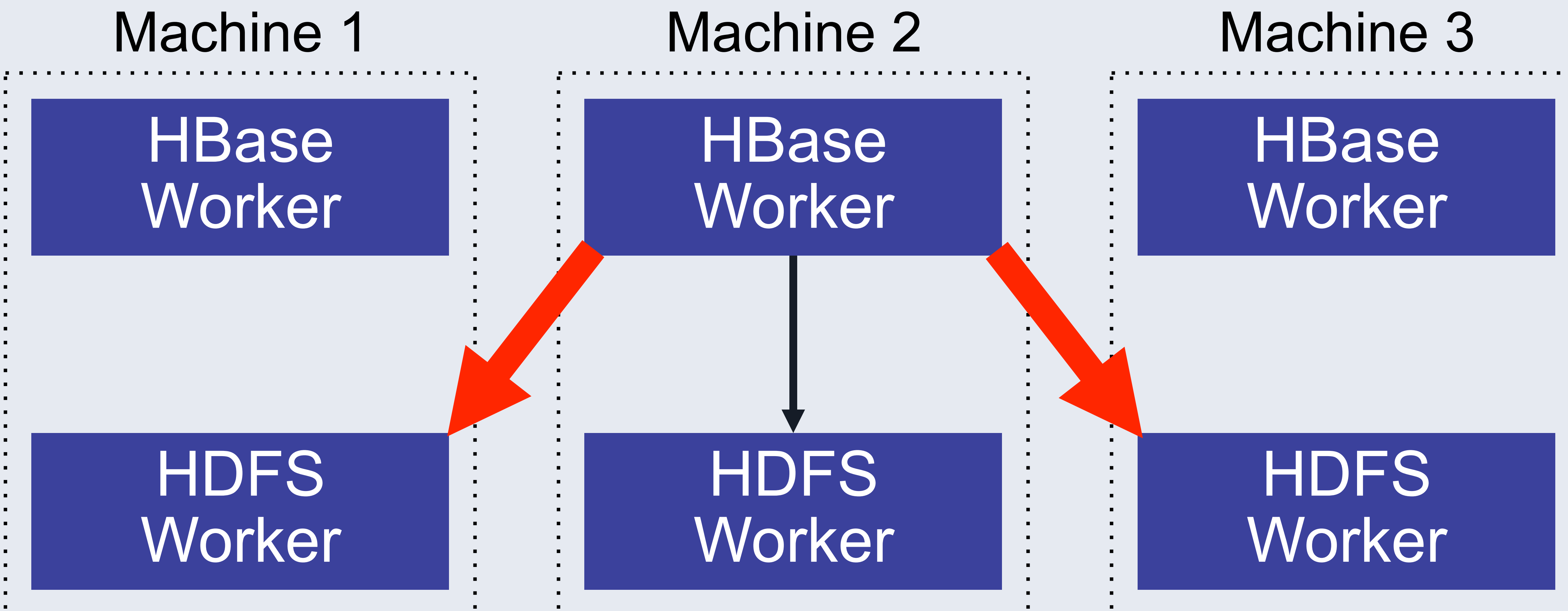
61% of writes are for compaction

- We should optimize compaction
- Compaction interacts with replication inefficiently

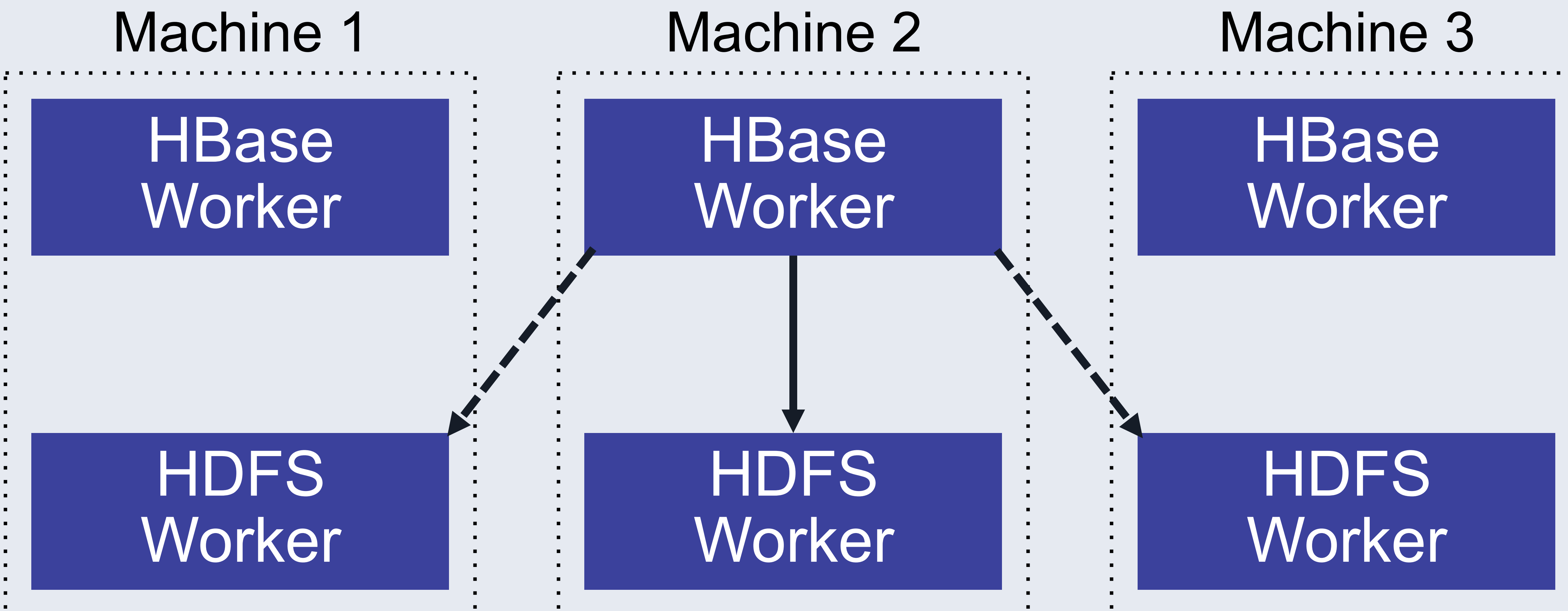
# Replication Overview



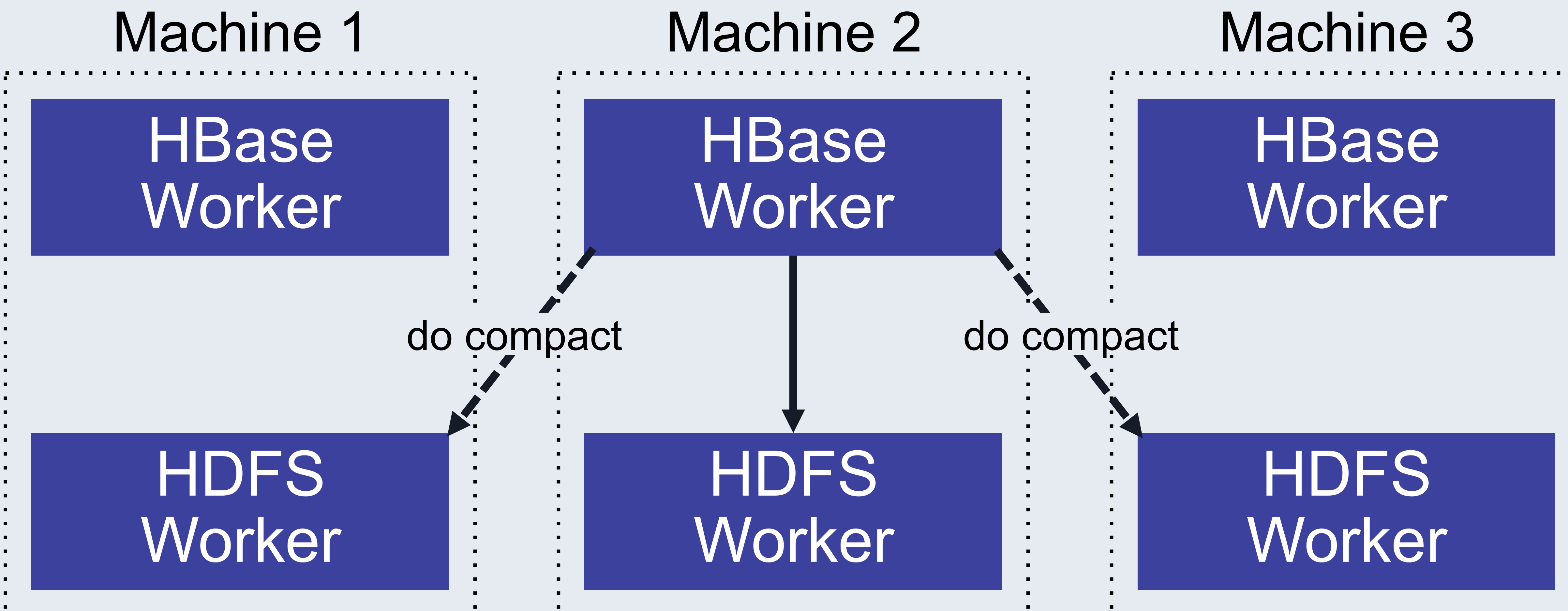
# Problem: Network I/O (red lines)



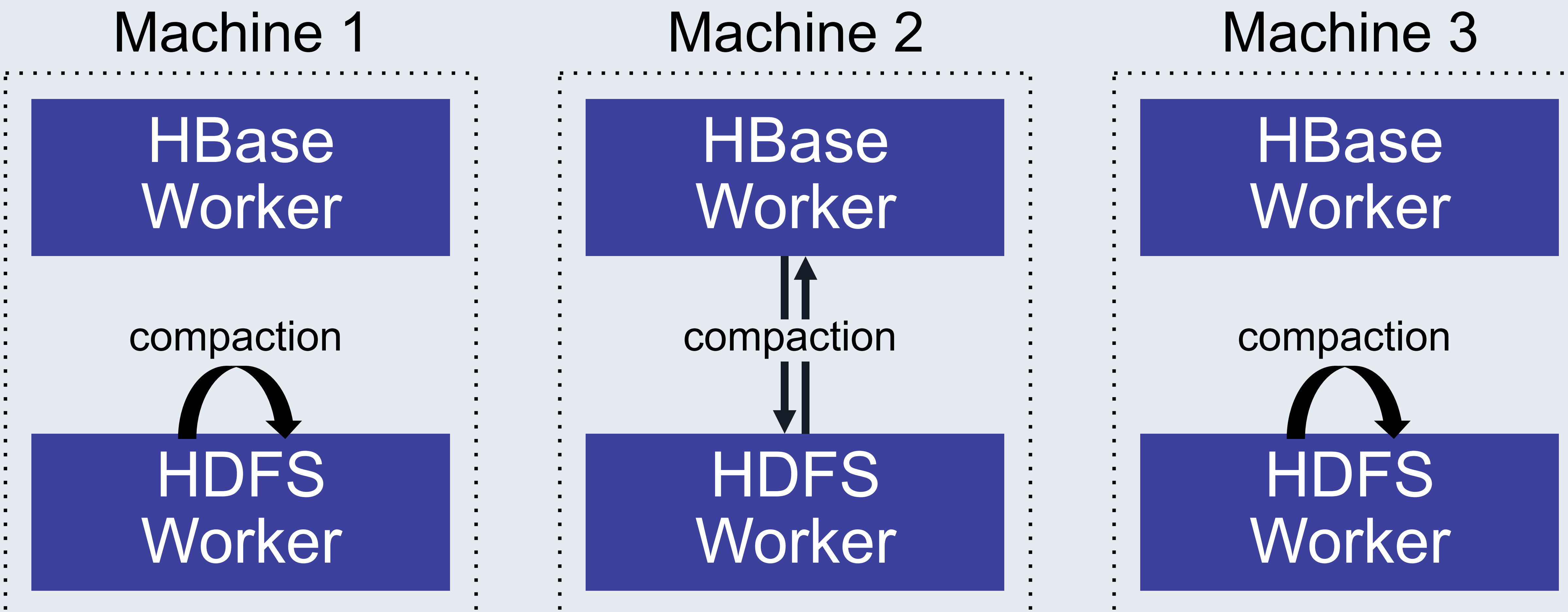
# Solution: Ship Computation to Data



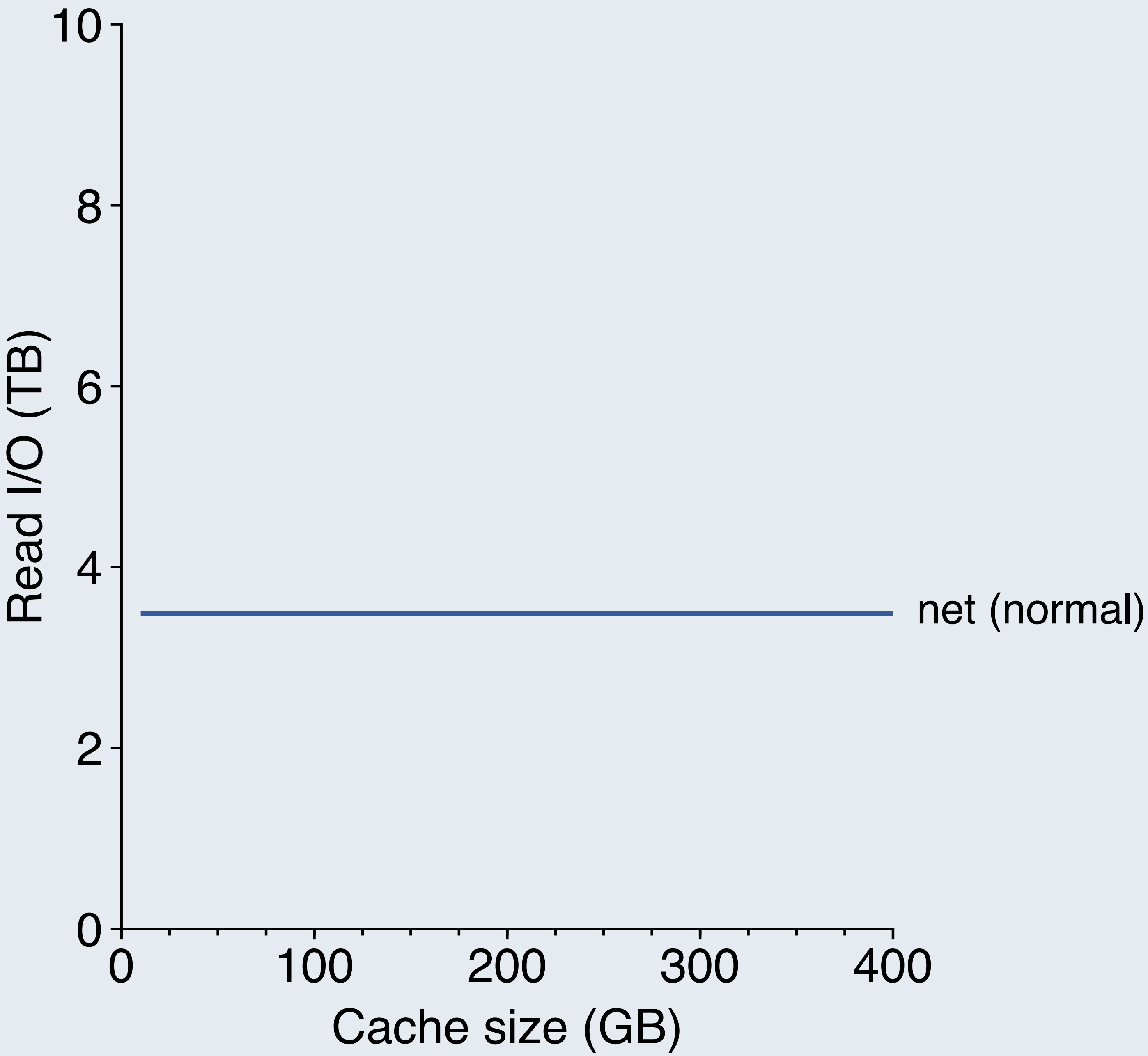
# Solution: do Local Compaction



# Solution: do <u>Local Compaction</u>

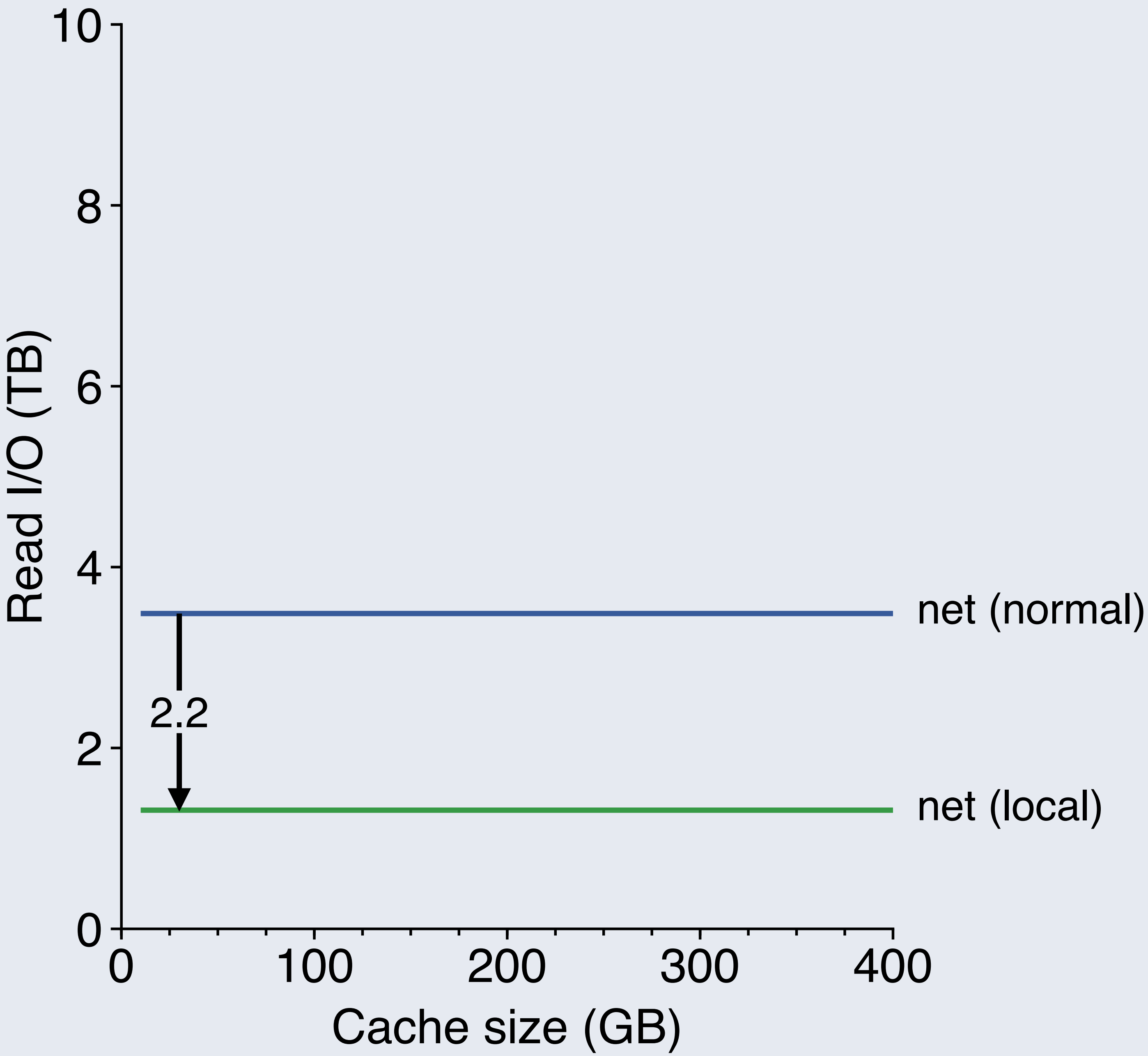


# Local Compaction




Normally 3.5TB of network I/O

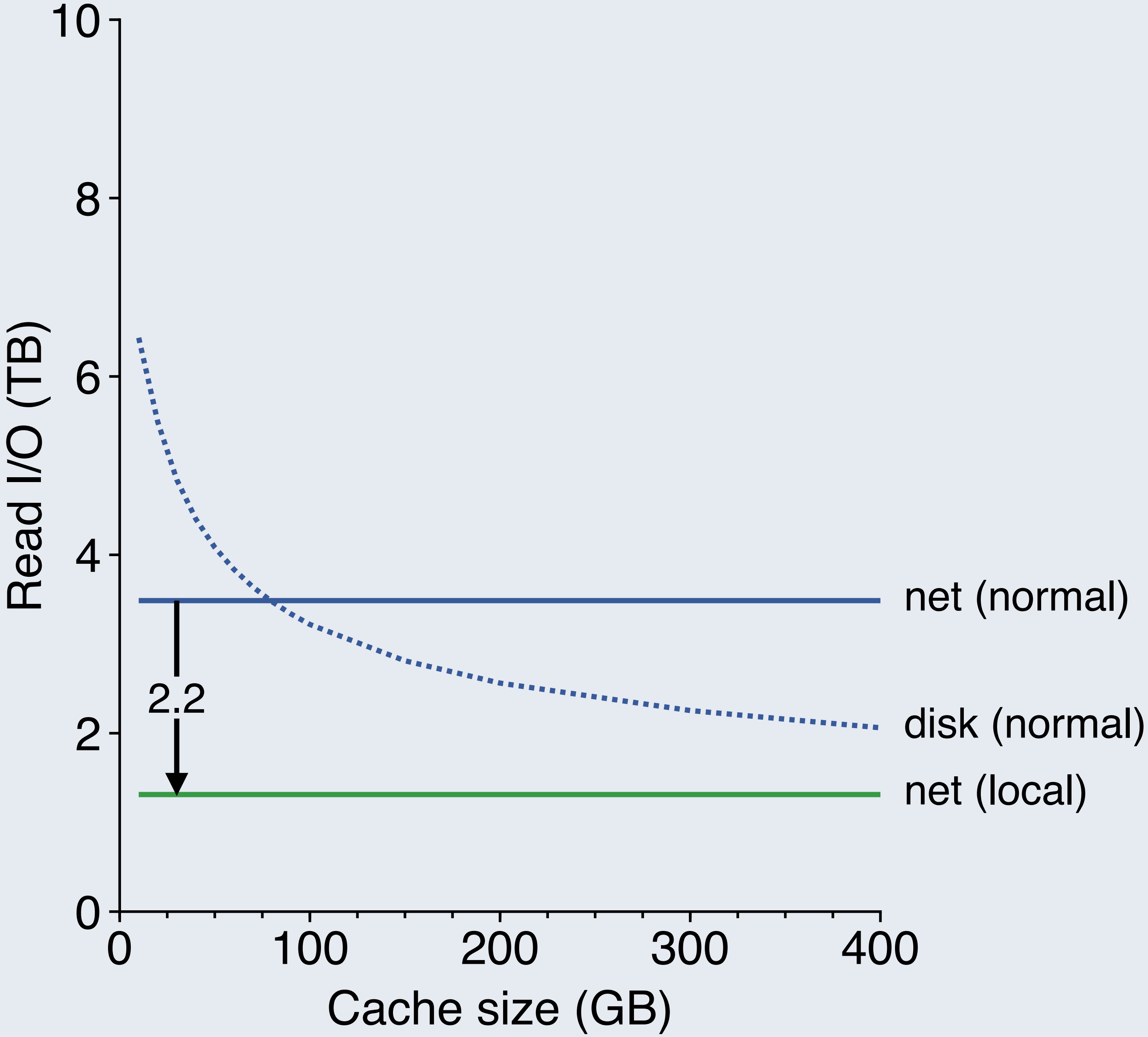
# Local Compaction




Normally 3.5TB of network I/O

Local comp: 62% reduction

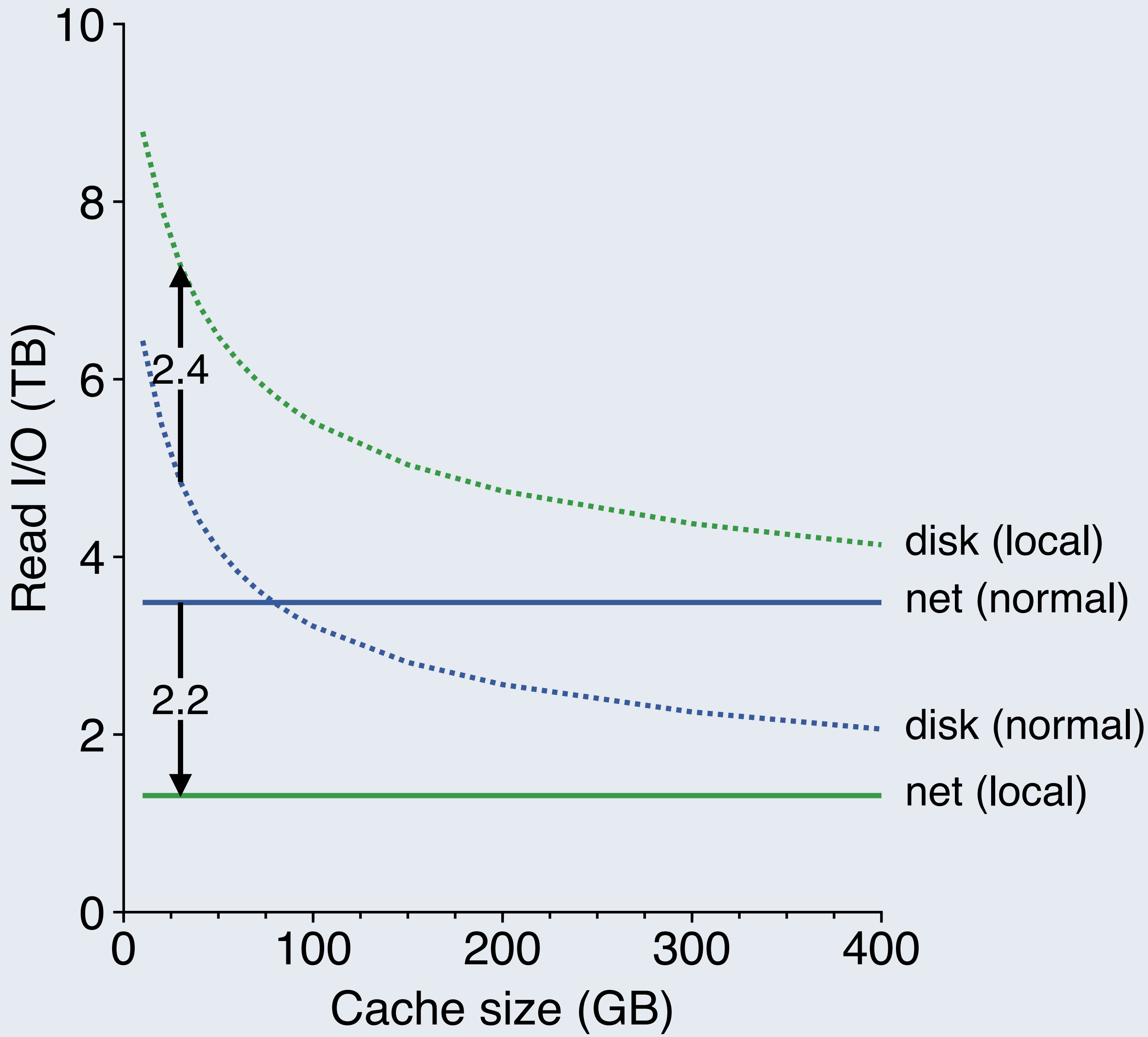
# Local Compaction




Normally 3.5TB of network I/O

Local comp: 62% reduction

# Local Compaction



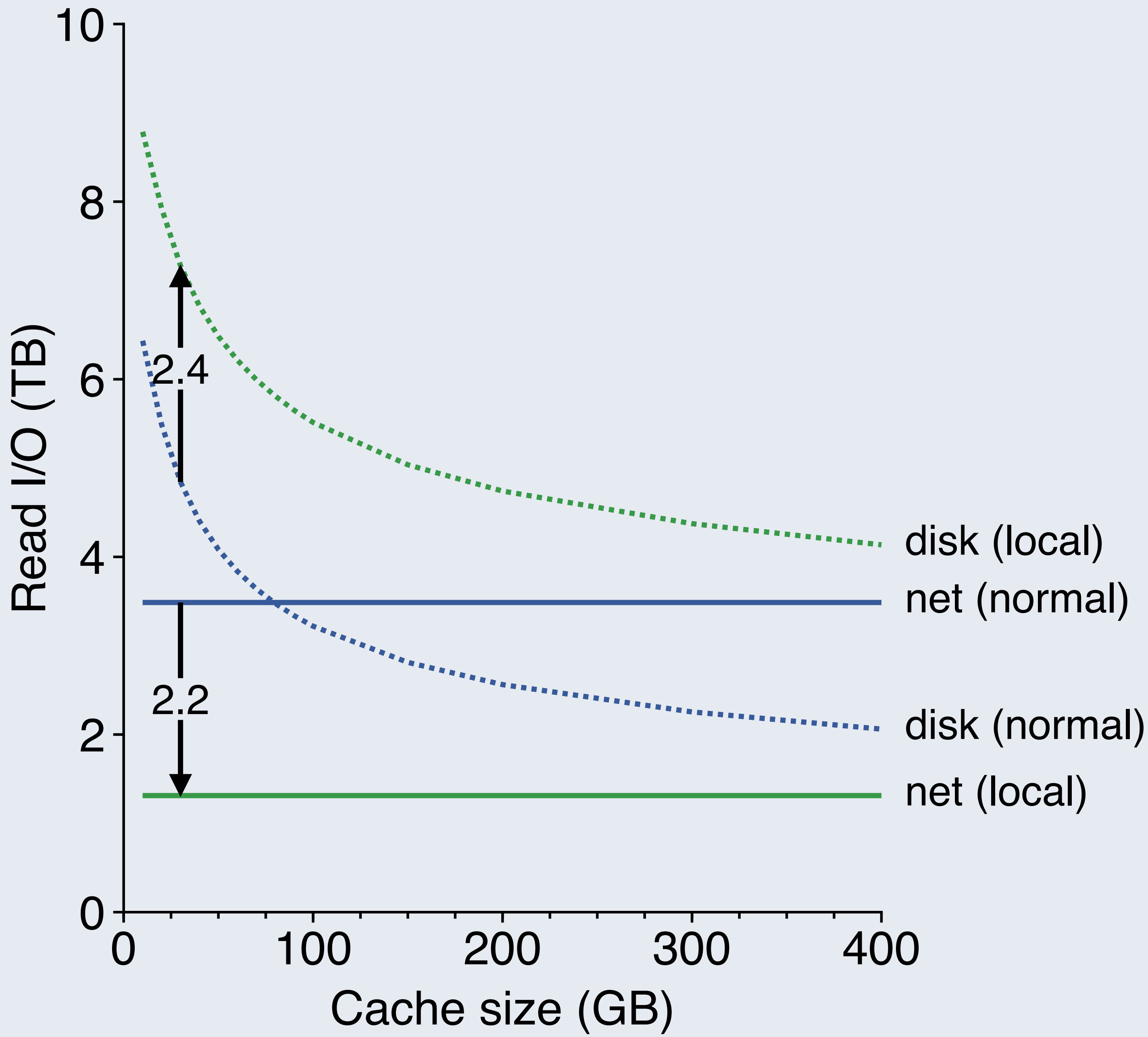

Normally 3.5TB of network I/O

Local comp: 62% reduction

Network I/O becomes disk I/O

- 9% overhead (30GB cache)
- Compaction reads: (a) usually misses, (b) pollute cache

# Local Compaction




Normally 3.5TB of network I/O

Local comp: 62% reduction

Network I/O becomes disk I/O

- 9% overhead (30GB cache)
- Compaction reads: (a) usually misses, (b) pollute cache

Still good!

- Disk I/O is cheaper than network

# Outline

Intro

- Messages stack overview
- Methodology: trace-driven analysis and simulation
- HBase background

Results

- Workload analysis
- Hardware simulation: adding a flash layer
- Software simulation: integrating layers

Conclusions

# Conclusion 1: Messages is a New HDFS Workload

Original GFS paper:

- "*high sustained bandwidth is more important than low latency*"
- "*multi-GB files are the common case*"

We find files are small and reads are random

- 50% of files <750KB
- >75% of reads are random

# Conclusion 2: Layering is Not Free

Layering "*proved to be vital for the verification and logical soundness*" of the THE operating system ~ Dijkstra

We find <u>layering is not free</u>

- Over half of network I/O for replication is unnecessary

Layers can amplify writes, multiplicatively

- E.g., logging overhead (10x) with replication (3x) => 30x write increase

Layer integration can help

- Local compaction reduces network I/O caused by layers

# Conclusion 3: Flash Should not Replace Disk

Jim Gray predicted (for ~2012) that "*tape is dead, disk is tape, flash is disk*"

We find flash is a poor disk replacement for Messages

- Data is very large and mostly cold
- Pure flash would cost >$10K/machine

However, small flash tier is useful

- A 60GB SSD cache can double performance for a 5% cost increase

# Thank you! Any questions?

University of Wisconsin-Madison

Facebook Inc.