Stage-Aware Anomaly Detection Through Execution Flow Tracking

Cristiana Amza

Department of Electrical and Computer Engineering
University of Toronto

Logging in Large Scale Systems

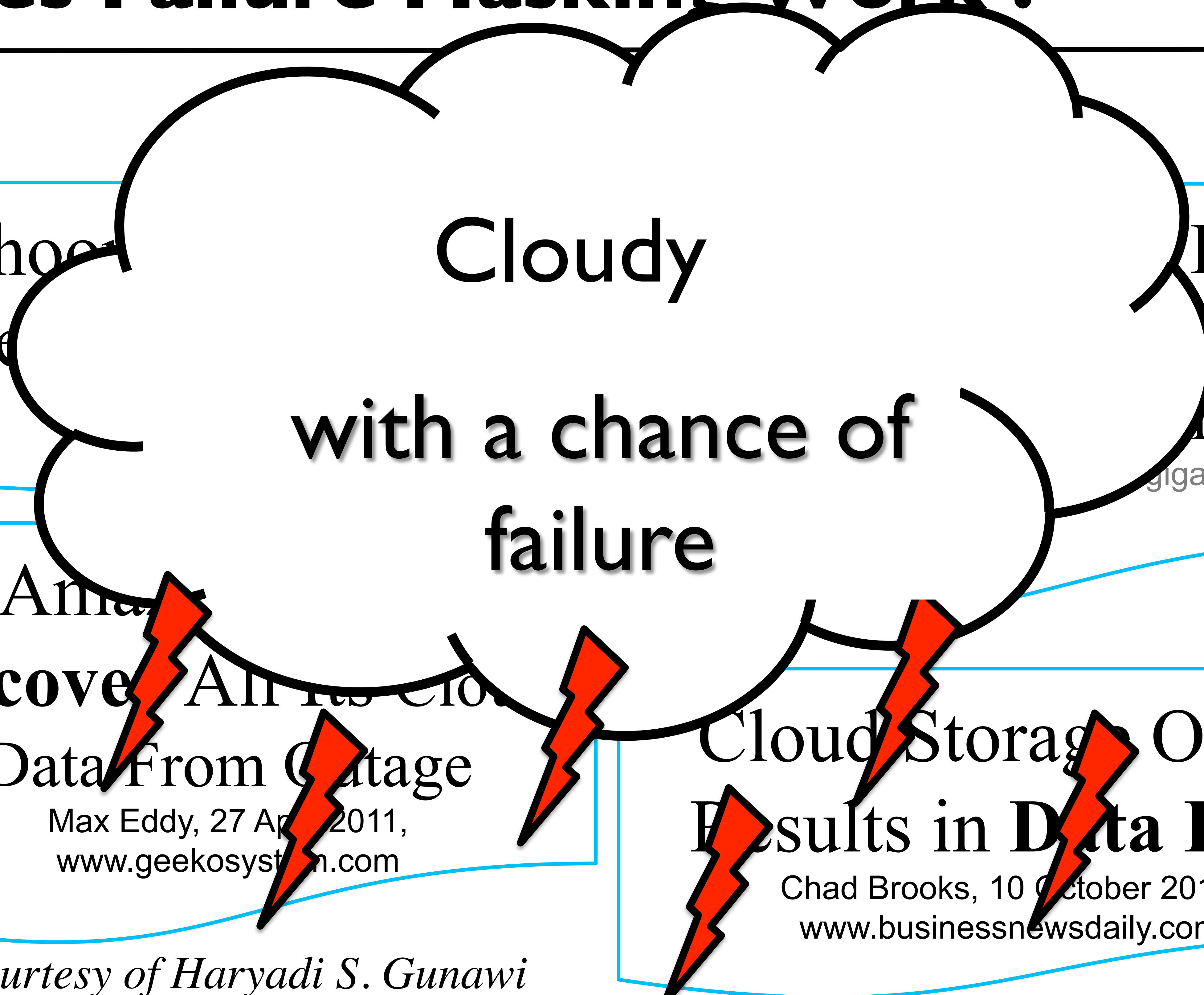
- Distributed storage systems of large-scale websites:
  - 1000s of servers
- Designed to be fault tolerant (automatic failure masking)
- Generate large volume of log data

hadoop Cassandra APACHE HBASE

Why Logging ?

- We log detailed information for anomaly diagnosis
- Verbose logging: Calls, network, control flow, errors
- Goal: Understand call graph, execution flow, root cause analysis

Does Failure Masking Work ?



Whoo...
lose...

...Fails:
...t
...r Data
...gigaom.com

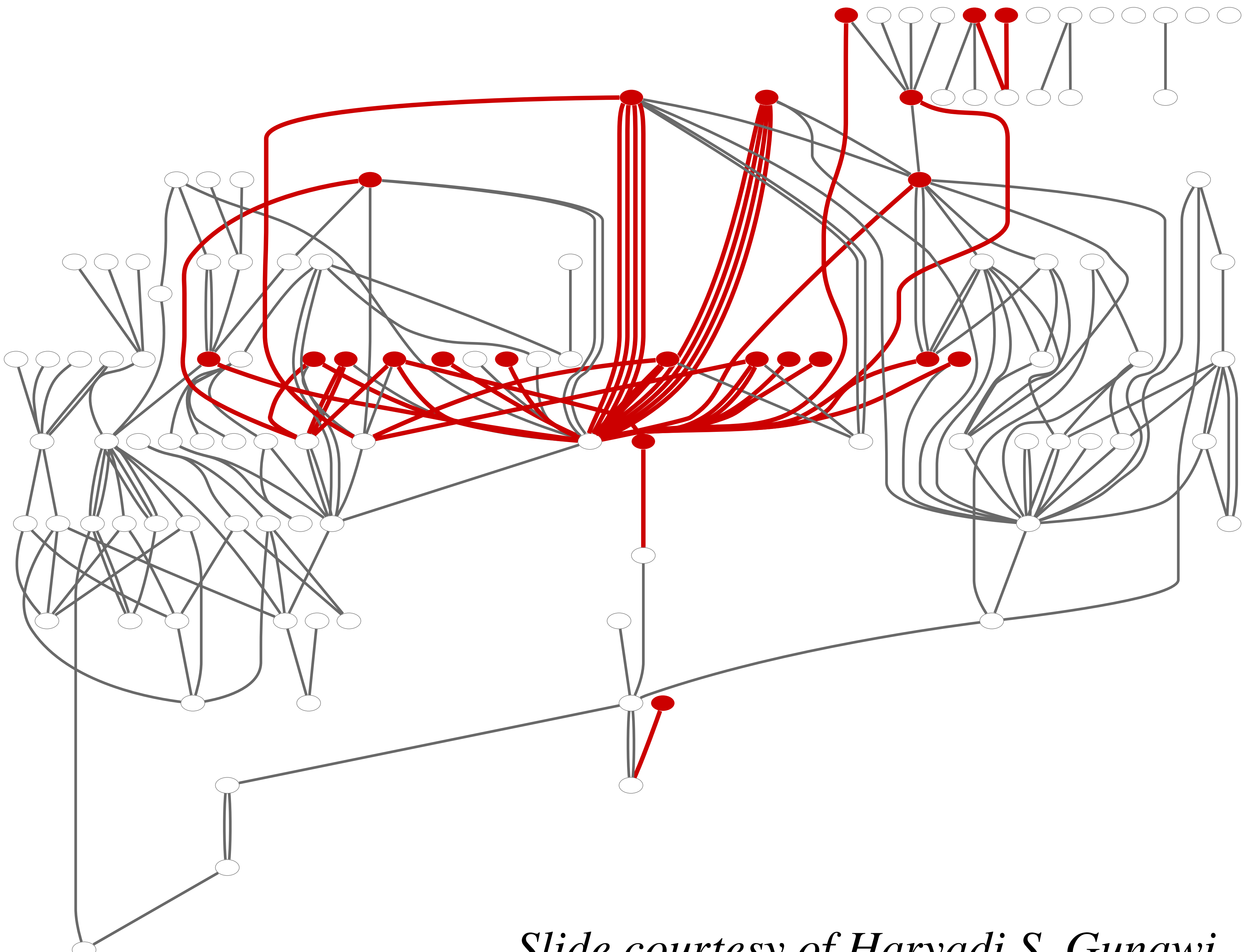
Ama...
Recove... All Its Clo...
Data From Outage
Max Eddy, 27 Ap... 2011,
www.geekosyst...m.com

Cloud Storage Often
Results in **Data Loss**
Chad Brooks, 10 October 2011,
www.businessnewsdaily.com

Slide courtesy of Haryadi S. Gunawi

Call Graph for FS

37 errors out of 188 calls

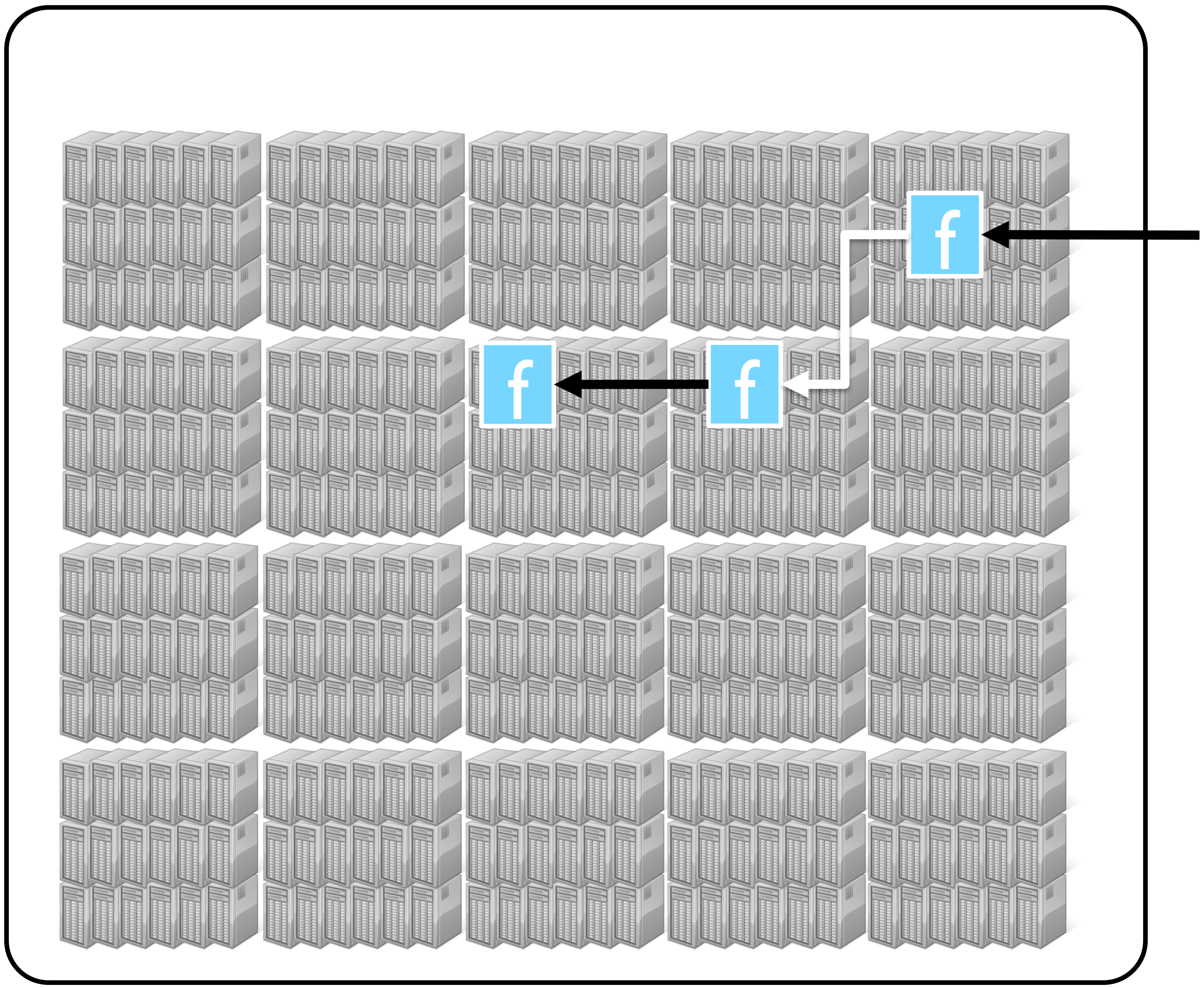


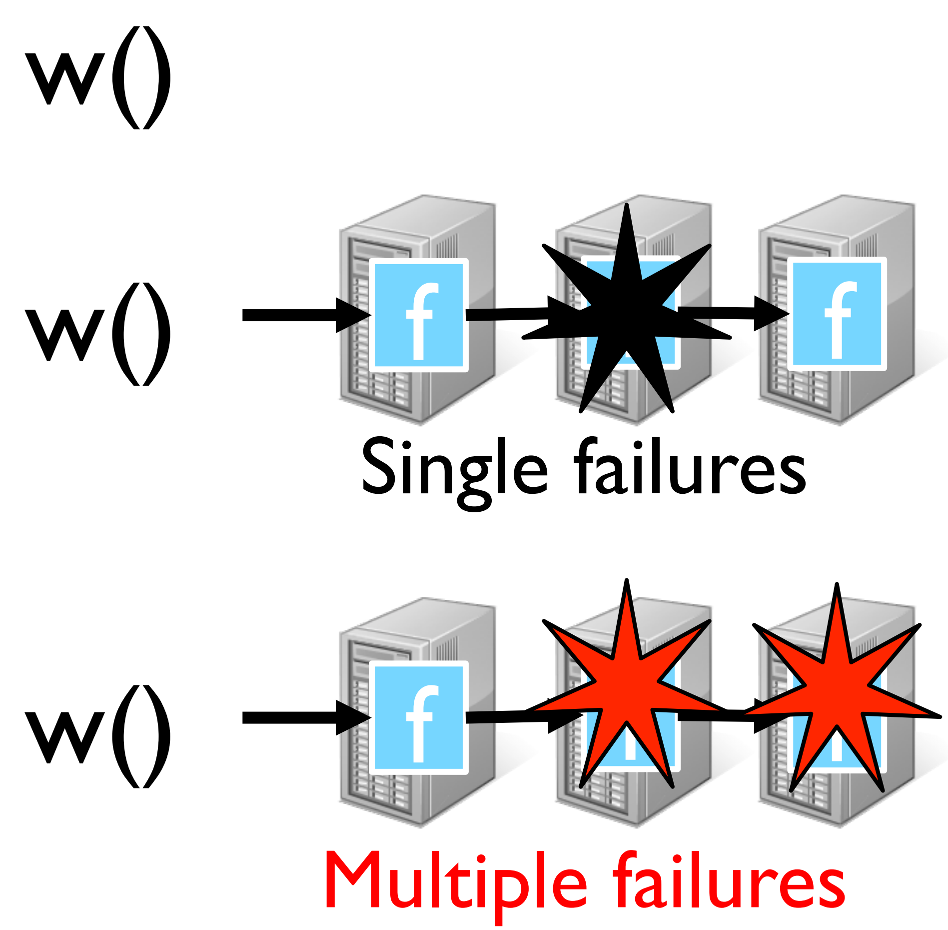
Slide courtesy of Haryadi S. Gunawi

HDFS: 3-way Replication (3 nodes)

- Storage for Hadoop MapReduce Jobs

- Yahoo!: 4000 nodes
- Facebook: 2000 nodes
- Ebay: 700 nodes



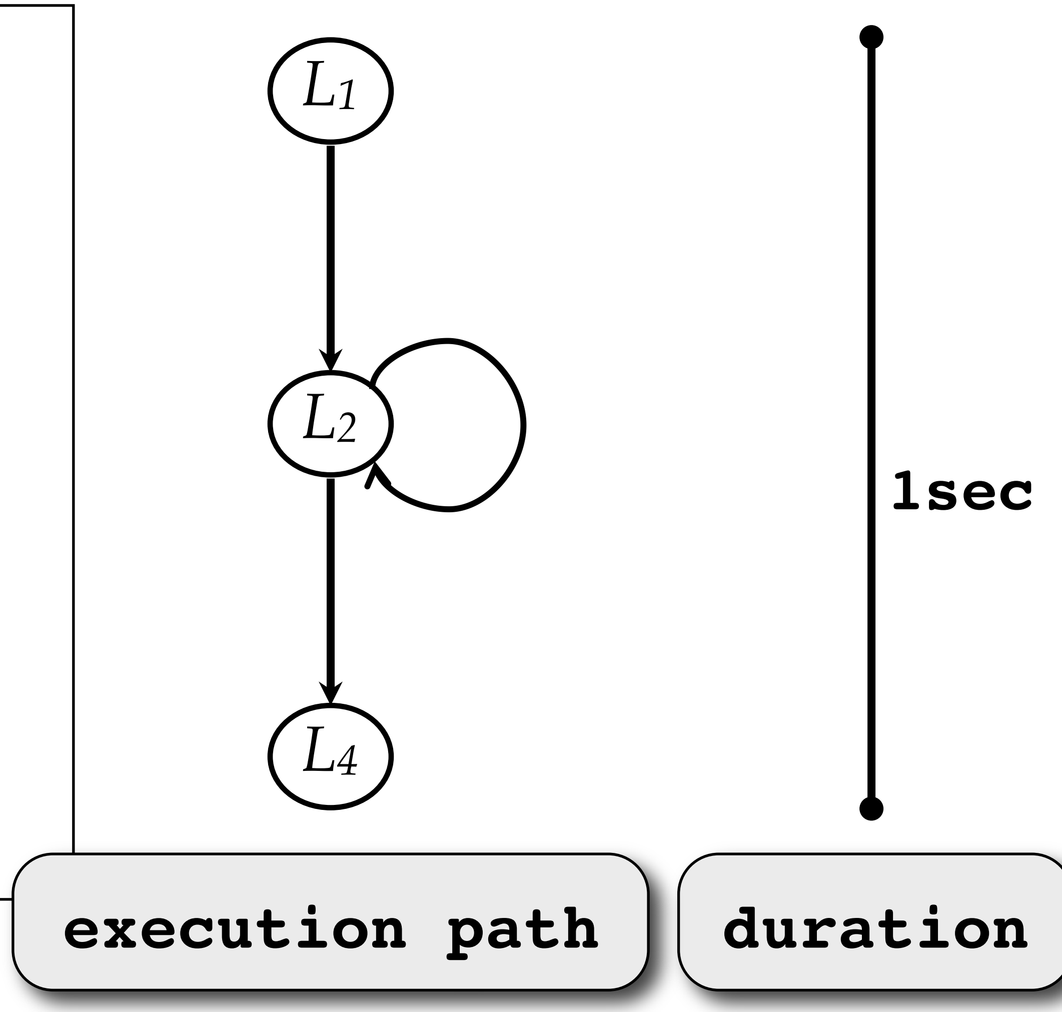


In An Ideal World ...

In the Real World of Logs, Logs, ...

FileTransfer

```
LOG("Sending file "+file.name);  //L1
int i=0;
while(i < file.size){
    int ack=send_recv(dest,file.block(i));
    if( ack == 0){
       i++;
       LOG("Sent block "+ i);  //L2
    }else{
       LOG("Failed to send file "+ file.name); //L3
       return -1;
   }
}
LOG("Sent file "+ file.name); //L4
return 0;
```



Logs

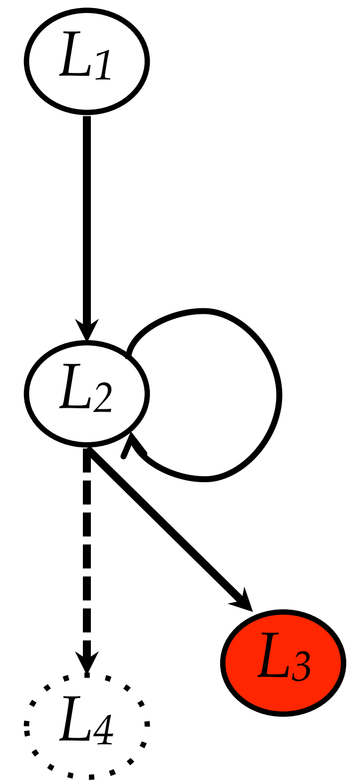
```
1:10.032 Sending file  A
1:10.04
1:10
....
1:11
1:11
```

Logs contain information about the execution flow and duration

Logs Contain Detailed Information

FileTransfer

```
LOG("Sending file "+file.name);  //L1
int i=0;
while(i < file.size){
    int ack=send_recv(dest,file.block(i));
    if( ack == 0){
       i++;
       LOG("Sent block "+ i);  //L2
    }else{
       LOG("Failed to send file "+ file.name); //L3
       return -1;
   }
}
LOG("Sent file "+ file.name); //L4
return 0;
```



execution path

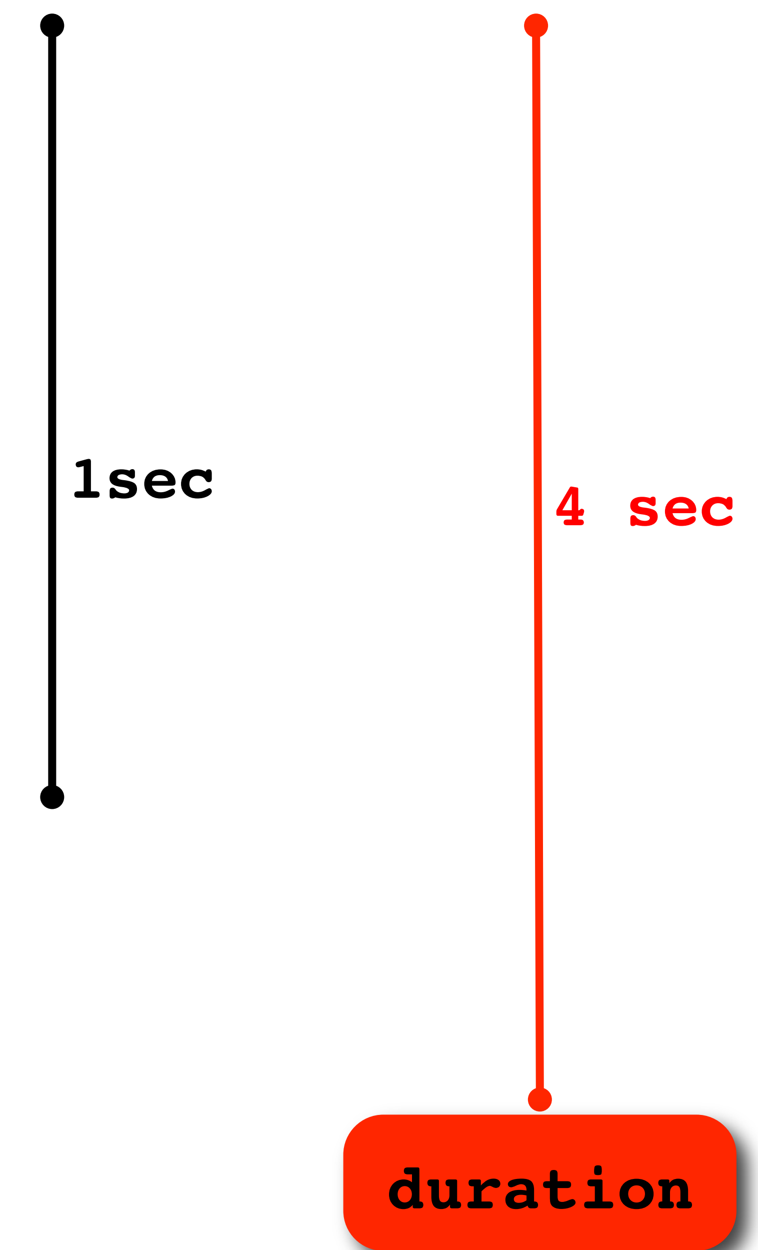
Logs

```
1:10.023 Sending file  F
1:10.042 Sent block 1
1:10.099 Sent block 2
....
1:11.130 Sent block 300
1:11.131 Failed to send file F
```

Logs Contain Detailed Information

FileTransfer

```
LOG("Sending file "+file.name);  //L1
int i=0;
while(i < file.size){
    int ack=send_recv(dest,file.block(i));
    if( ack == 0){
       i++;
       LOG("Sent block "+ i);  //L2
    }else{
       LOG("Failed to send file "+ file.name); //L3
       return -1;
    }
}
LOG("Sent file "+ file.name); //L4
return 0;
```

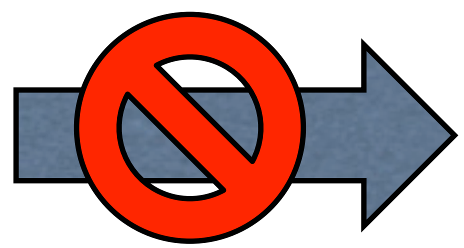


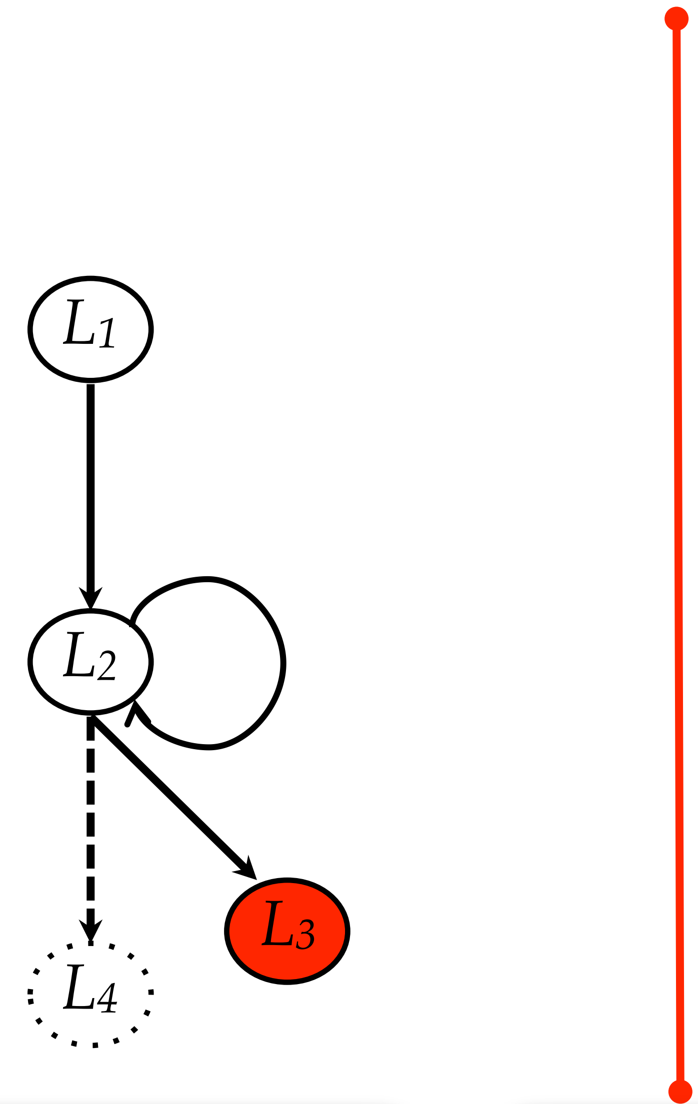
Logs

```
1:10.032 Sending file  B
1:10.042 Sent block 1
1:10.099 Sent block 2
....
1:14.030 Sent block 1000
1:14.032 Sent file B
```

Inferring Execution Flow from Logs

```
1:10.023 Sending file  F
1:10.042 Sent block 1
1:10.099 Sent block 2
....
1:11.130 Sent block 300
1:11.131 Failed to send file F
```





execution path

duration

Challenges

- Identify tasks from logs
- Distinguish normal vs. anomalous tasks
- Overheads
  - Storage and processing

Challenge 1: Identify tasks from logs

Normal Execution Flow

```
1:10.023 Start sending file  A
1:10.042 Sent block 1
1:10.099 Sent block 2
....
1:11.130 Sent block 1000
1:11.132 Sent file A
```

Anomalous Execution Flow

```
1:10.033 Start sending file  F
1:10.032 Sent block 1
1:11.049 Sent block 2
....
1:11.050 Sent block 300
1:11.131 Failed to send file F
```

```
1:10.023 Start sending file  A
1:10.033 Start sending file  F
1:10.032 Sent block 1
1:10.042 Sent block 1
1:11.049 Sent block 2
1:10.099 Sent block 2
…
1:11.050 Sent block 300
1:11.130 Sent block 1000
1:11.131 Failed to send file F
1:11.132 Sent file A
```

Server runs forever, how to delimit tasks ?

Challenge 1: Identify tasks from logs

Normal Execution Flow

```
1:10.023 Start sending file  A
1:10.042 Sent block 1
1:10.099 Sent block 2
....
1:11.130 Sent block 1000
1:11.132 Sent file A
```

Anomalous Execution Flow

```
1:10.033 Start sending file  F
1:10.032 Sent block 1
1:11.049 Sent block 2
....
1:11.050 Sent block 300
1:11.131 Failed to send file F
```

```
1:10.023 Start sending file  A
1:10.033 Start sending file  F
1:10.032 Sent block 1
1:10.042 Sent block 1
1:11.049 Sent block 2
1:10.099 Sent block 2
…
1:11.050 Sent block 300
1:11.130 Sent block 1000
1:11.131 Failed to send file F
1:11.132 Sent file A
```

Thread reuse, data flow tracking may obscure task execution flow

Challenge 2: Normal vs Anomaly

- No baseline
- Anomalies do not always generate an explicit error/warning log message

How to identify normal vs. rare execution flows?

Challenge 3: Overhead

- Log messages are text to be read by human
- Verbose logging generates huge log volume!
  - **2600** times more than INFO-level (default) logging
- Text processing is expensive and imprecise

2.4TB per day for a 100 node Cassandra cluster
Not many people can look into a 2TB log file

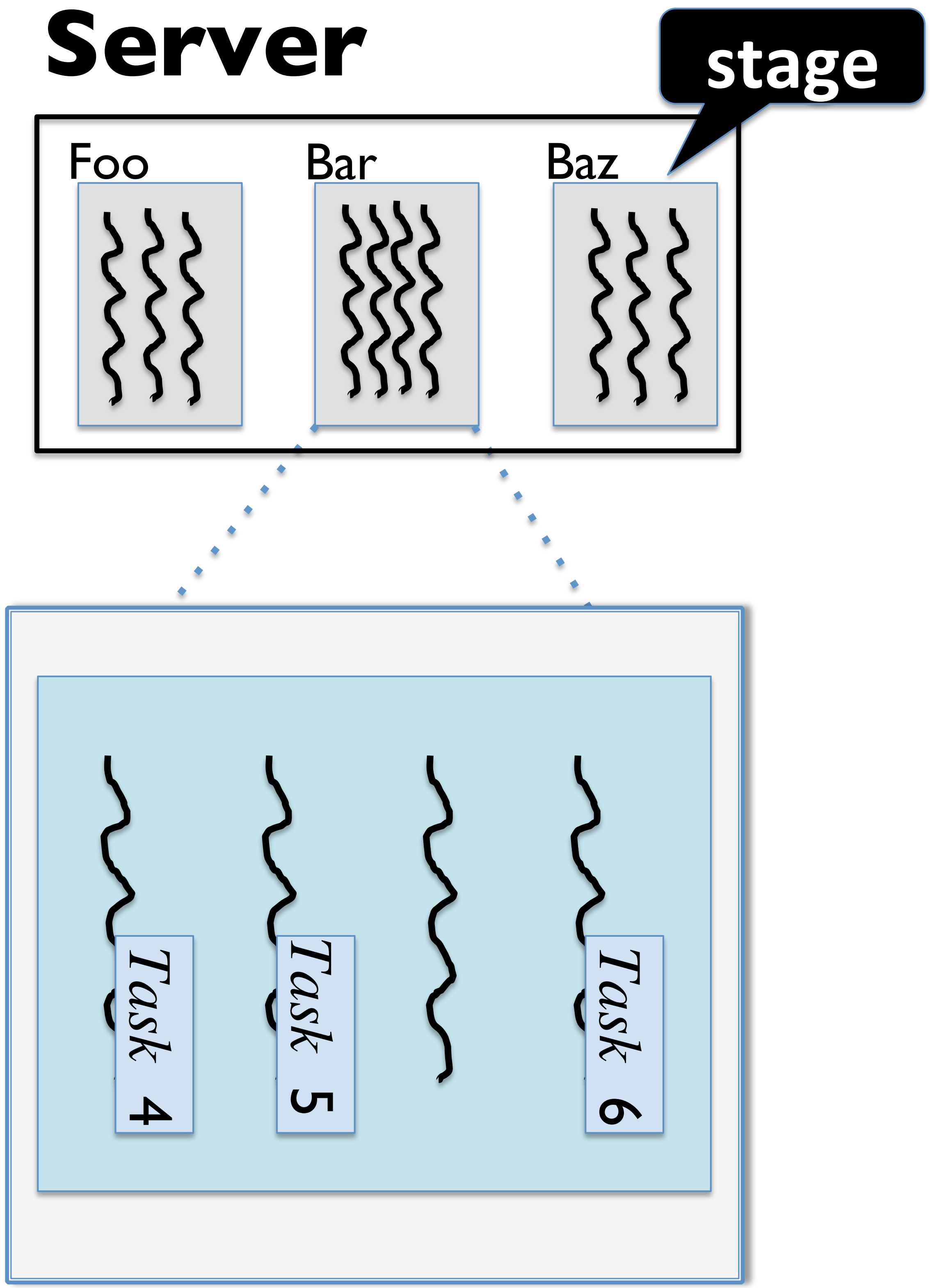
Key Observation

Many server codes have a modular or stage-based architecture

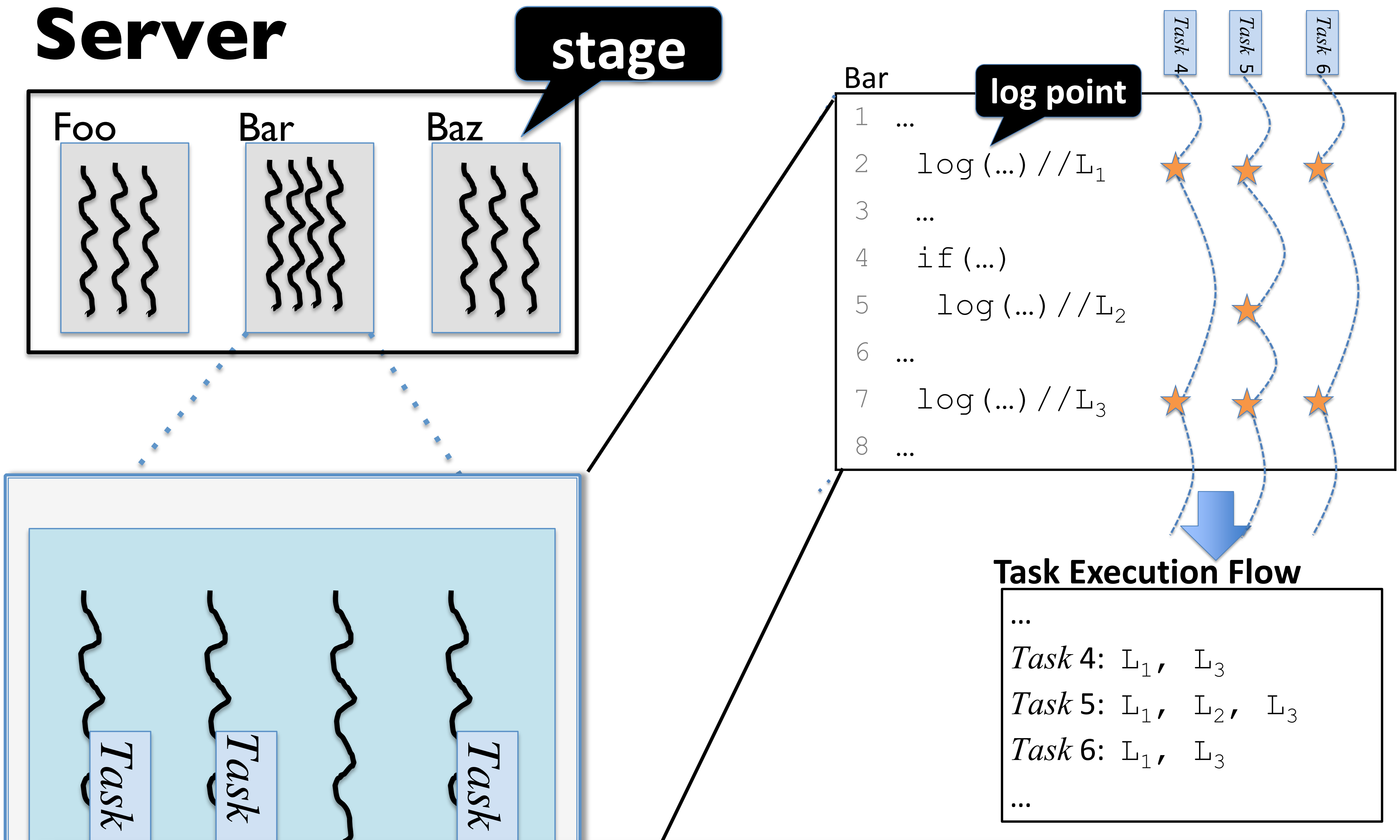
- Stage-aware anomaly detection
  - on-the-fly
  - with low overhead
  - almost completely automated

Staged-Architecture

Server

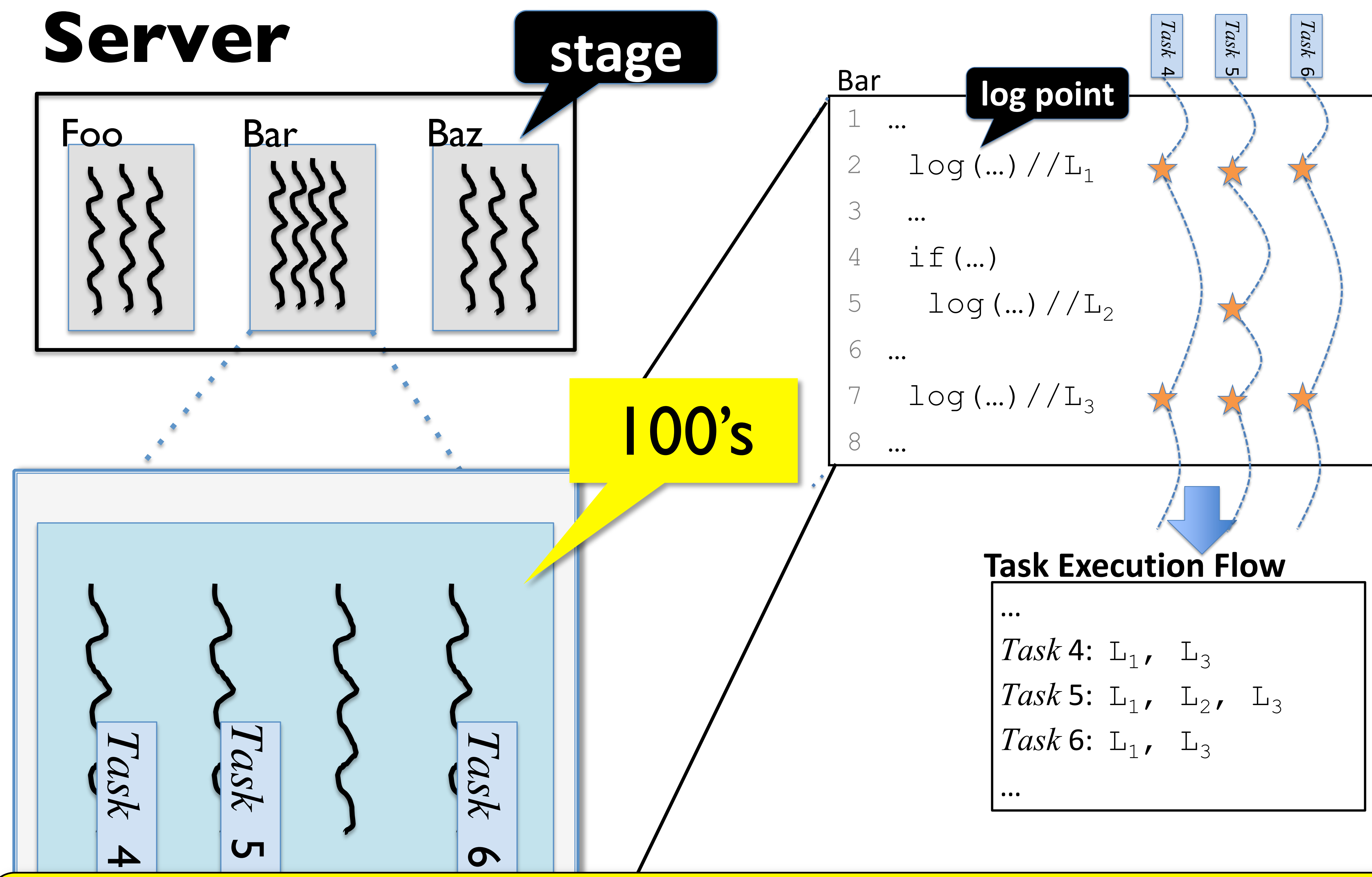


Staged-Architecture



We capture execution flow from log points on-the-fly without generating logs

Staged-Architecture

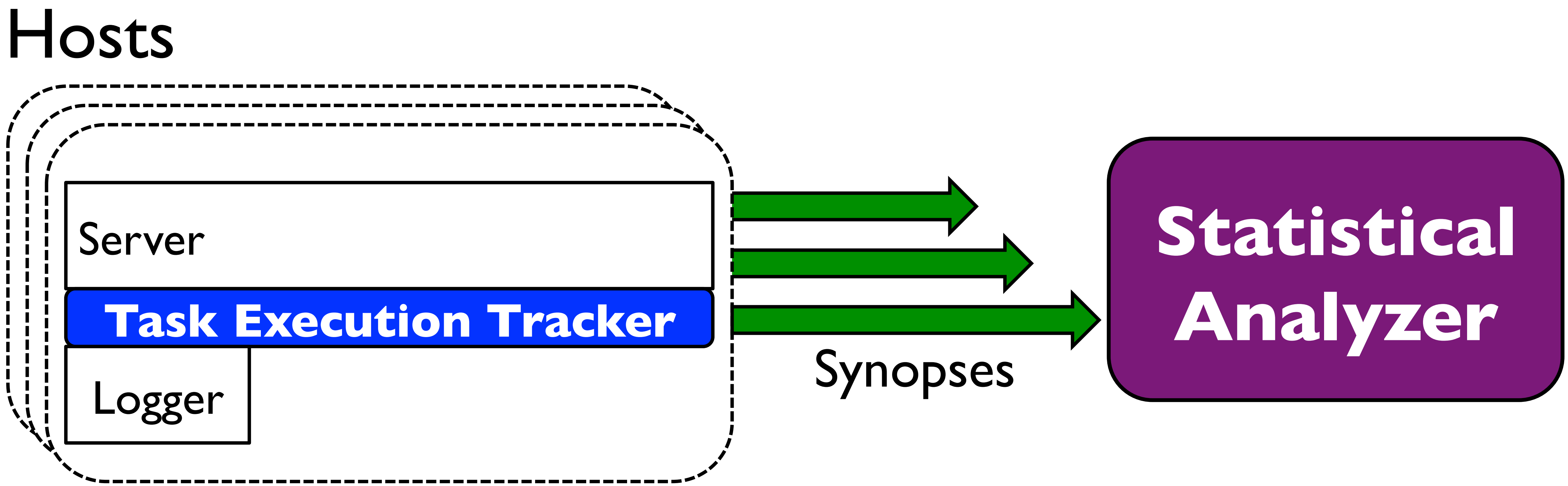


We exploit similarity between tasks for statistical anomaly detection

Stage-Aware Anomaly Detection

- Leverage the staged code structure
  - To track start and end of each task
- Log statements as trace points
  - To track execution flow of the tasks
- Exploit the statistical similarity of tasks to detect
  - **Flow anomalies:** rare execution paths
  - **Performance anomalies:** unusually high duration

Prototype



- **Task Execution Tracker** on each sever
  - tracks log points encountered by each task
  - generates **synopses** of task execution flows
  - streams to **Statistical Analyzer** for real-time anomaly detection

Instrumentation

FileTransfer

```
Tracker.setContext("FileTransfer", task_id);
LOG("Sending file "+file.name);  //L1
int i=0;
while(i < file.size){
    int ack=send_recv(dest,file.block(i));
    if( ack == 0){
       i++;
       LOG("Sent block "+ i);  //L2
    }else{
       LOG("Failed to send file "+ file.name); //L3
       return -1;
   }
}
LOG("Sent file "+ file.name); //L4
return 0;
```

FileTransfer:134

Tracking Log Points

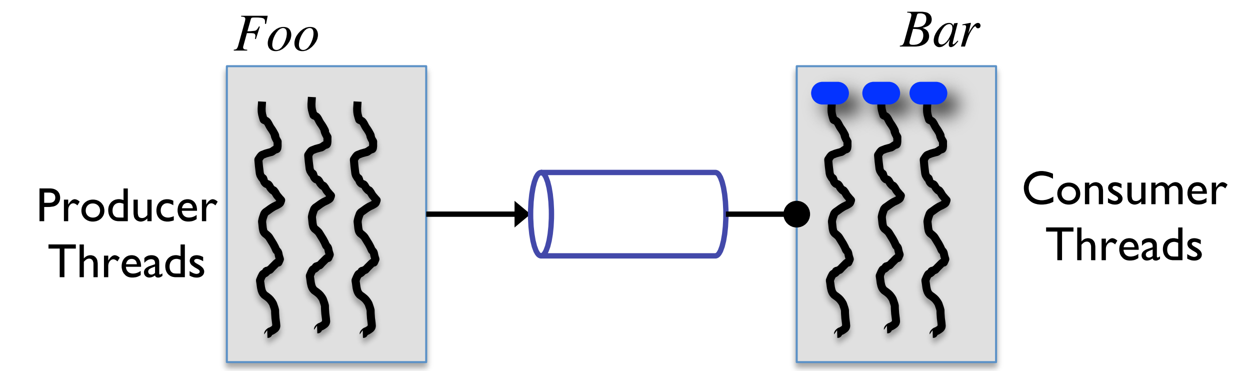
FileTransfer

```
Tracker.setContext("FileTransfer", task_id)
LOG(1,"Sending file "+file.name);  //L1
int i=0;
while(i < file.size){
    int ack=send_recv(dest,file.block(i));
    if( ack == 0){
       i++;
       LOG(2,"Sent block "+ i);  //L2
    }else{
       LOG(3,"Failed to send file "+ file.name); //L3
       return -1;
   }
}
LOG(4,"Sent file "+ file.name); //L4
return 0;
```

FileTransfer:134

Synopsis: <134, "FileTransfer", 3.5 ms, [L_1, L_2, L_4]>

Automatic Instrumentation

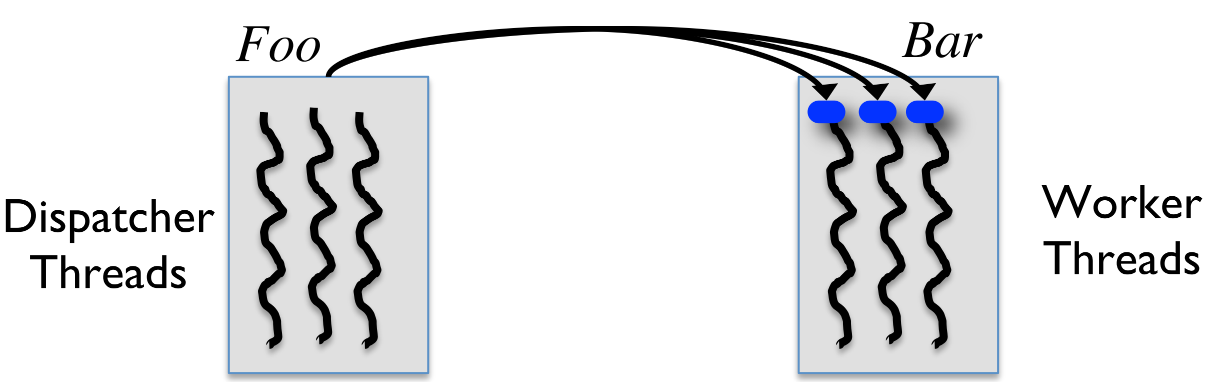


producer-consumer model

consumer thread

```
while (1){
 req=deqeue()
 Tracker.setConte                                        )
 do(req)
}
```

server	#stages
HDFS	**10**
HBase	**38**
Cassandra	**78**



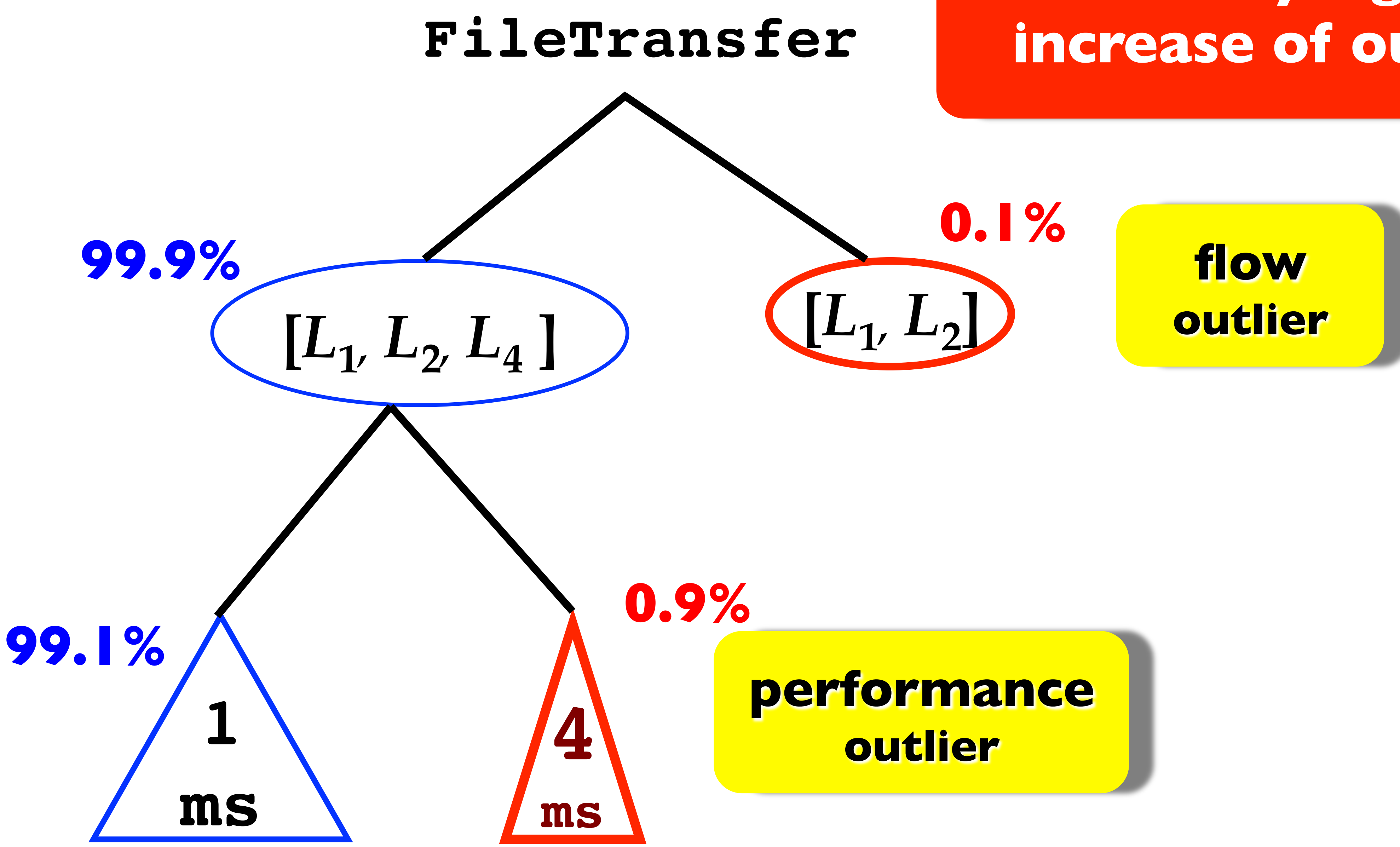
dispatcher-worker model

worker thread

```
void Thread.run(){
 Tracker.setContext("FileTransfer",task_id)
 do()
}
```

Model Building (per Stage)

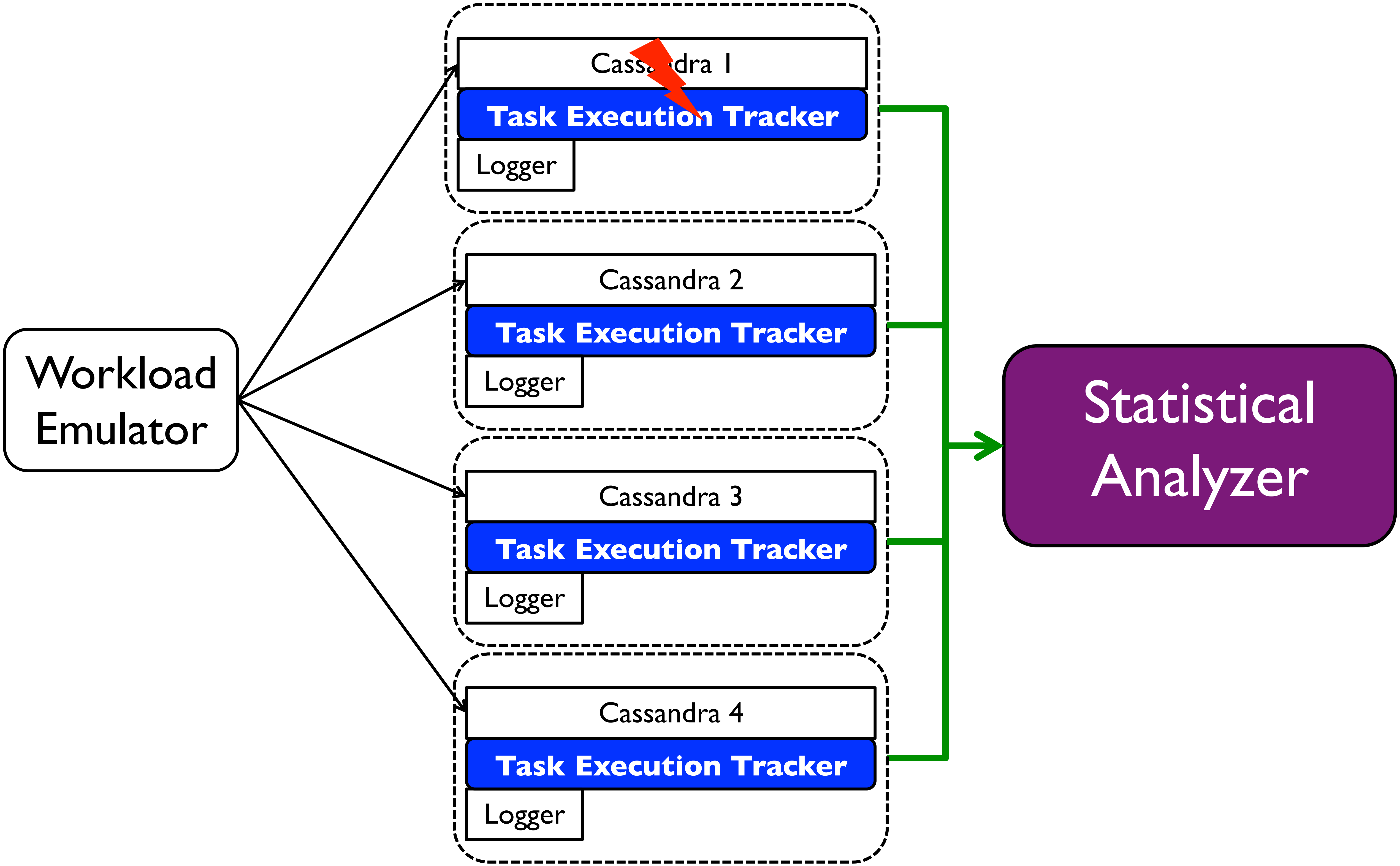
Anomaly: statistically significant increase of outliers



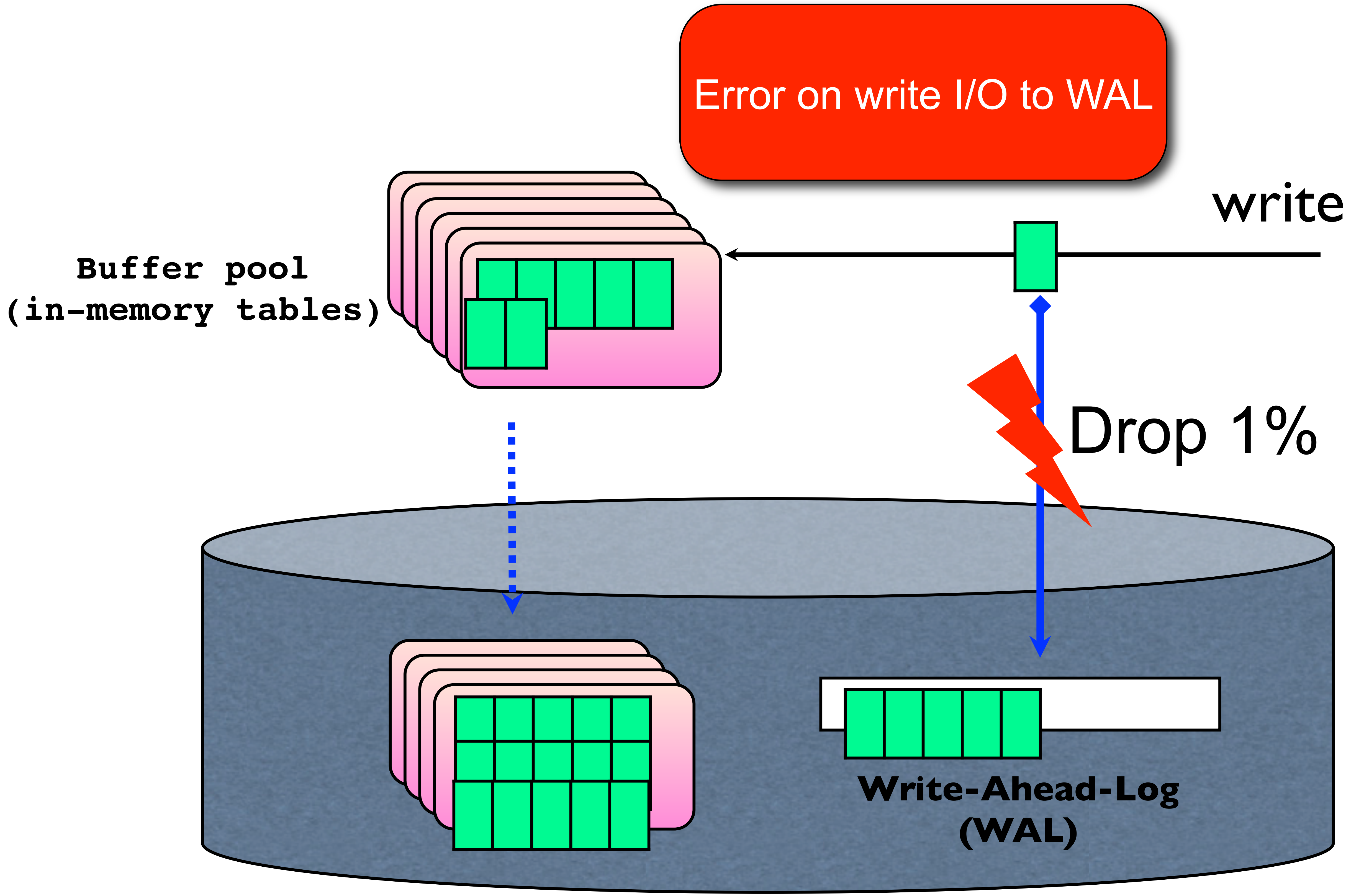
Evaluation

- Three distributed storage systems:
  - Hadoop Distributed Filesystem (HDFS)
  - HBase
  - **Cassandra**
- Write intensive workload of Yahoo Cloud Serving Benchmark (YCSB)
  - Similar to real-world scenarios
  - Stress the system

Experiment Setup



Cassandra write I/O path



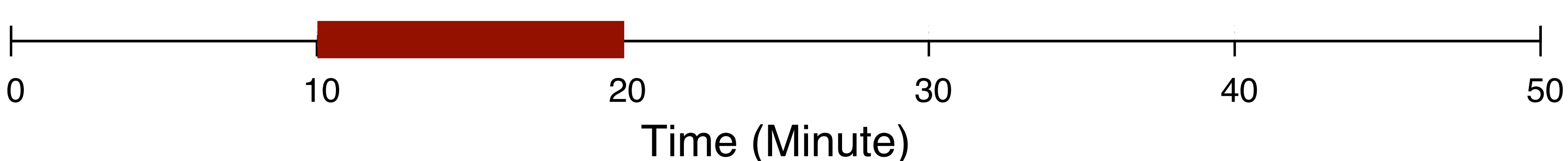
Error on write I/O ()

Fault: failing 1% of write to WAL

Cassandra 1

Execution Flow

	Log Id	Log Statement
Normal	L101	"memtable is already frozen; another thread must be flushing it"
	L102	"Applying"
	L103	"Adding hint for"
	L104	"... applied. Sending response to"
	L105	"applying mutation of row {}"
Anomalous	L101	"memtable is already frozen; another thread must be flushing it"



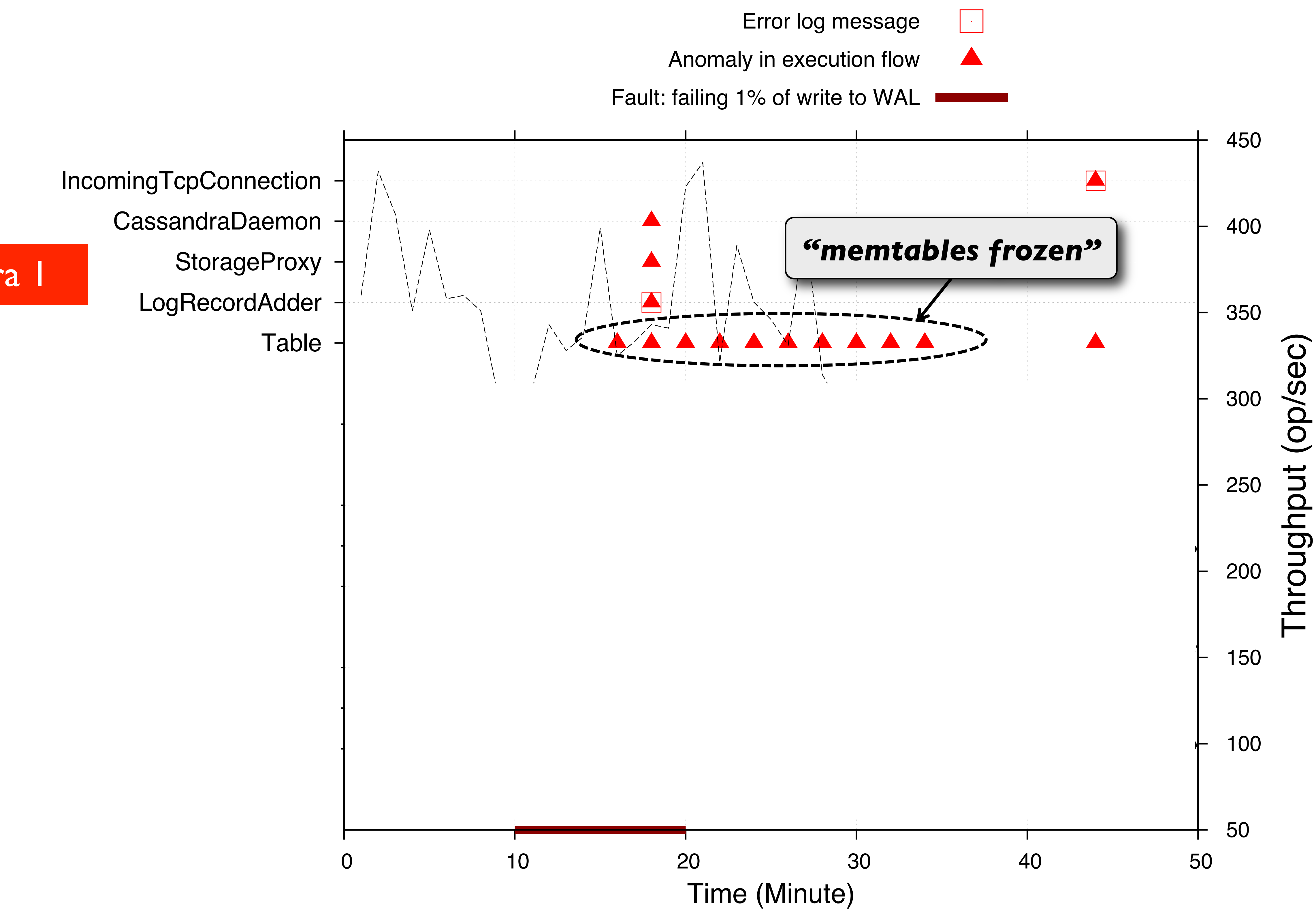
Error logs

X Input/output error

X Fatal exception – crash!

Error on write I/O

Cassandra 1



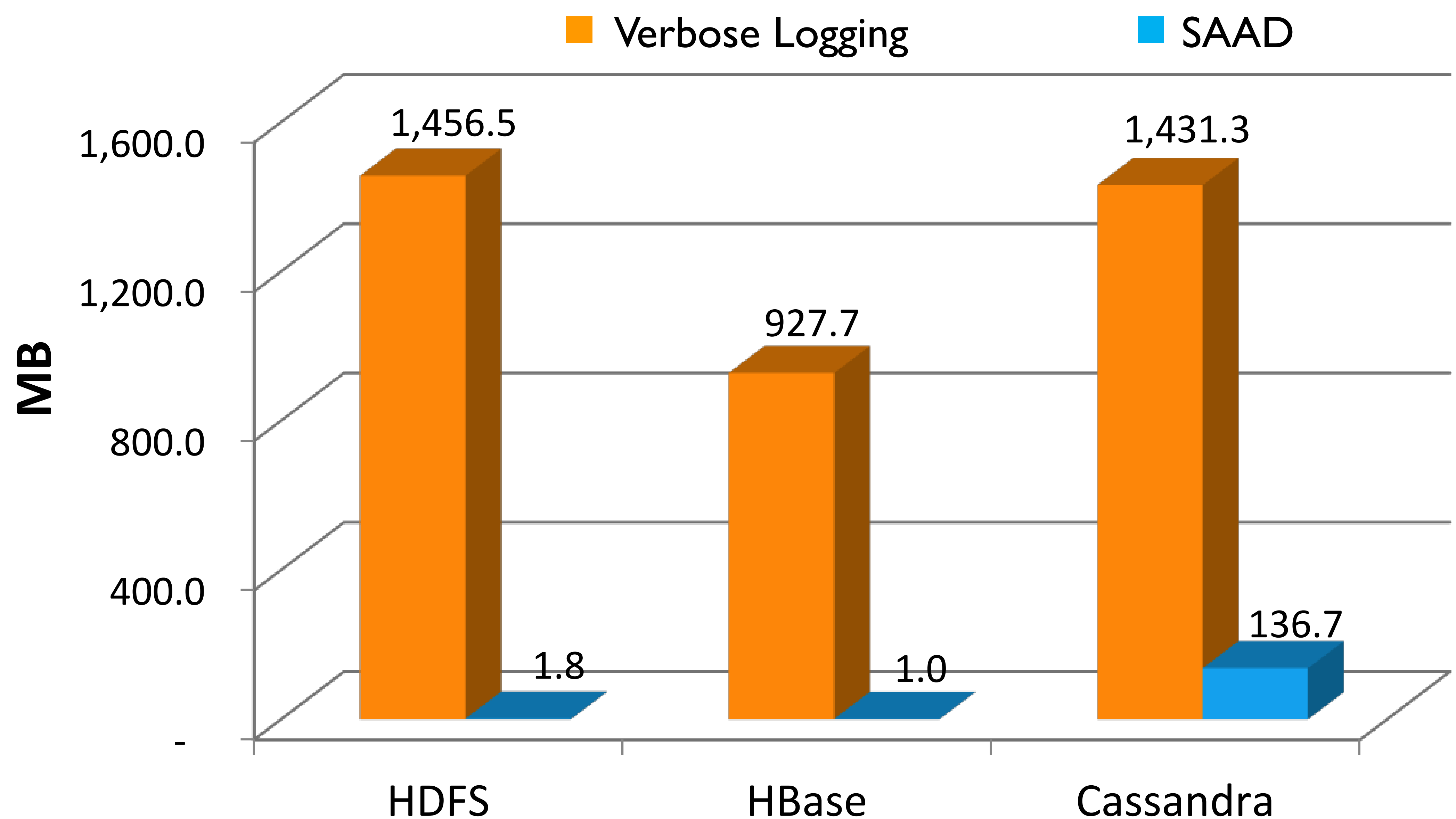
Uncovering Bugs & Misconfigurations

#	Type	Component	Description
1	Bug	HDFS Data Node	Empty Packet
2	Bug	Hbase Regionserver	Distributed Log splitting gets stuck
3	Misconfig.	HBase Regionserver	No live nodes contain current block
4	Misconfig.	Hbase Regionserver	Zookeeper missed heartbeat due to lengthy GC

Analysis Cost

- State of the Art, based on regular expression matching:
  - Offline processing of 1.6GB log data
    - 12 millions log messages
  - 12 Minutes on 8 cores full utilization
- Our solution
  - Real time on one core
  - Average of 3% CPU utilization

Storage Overhead



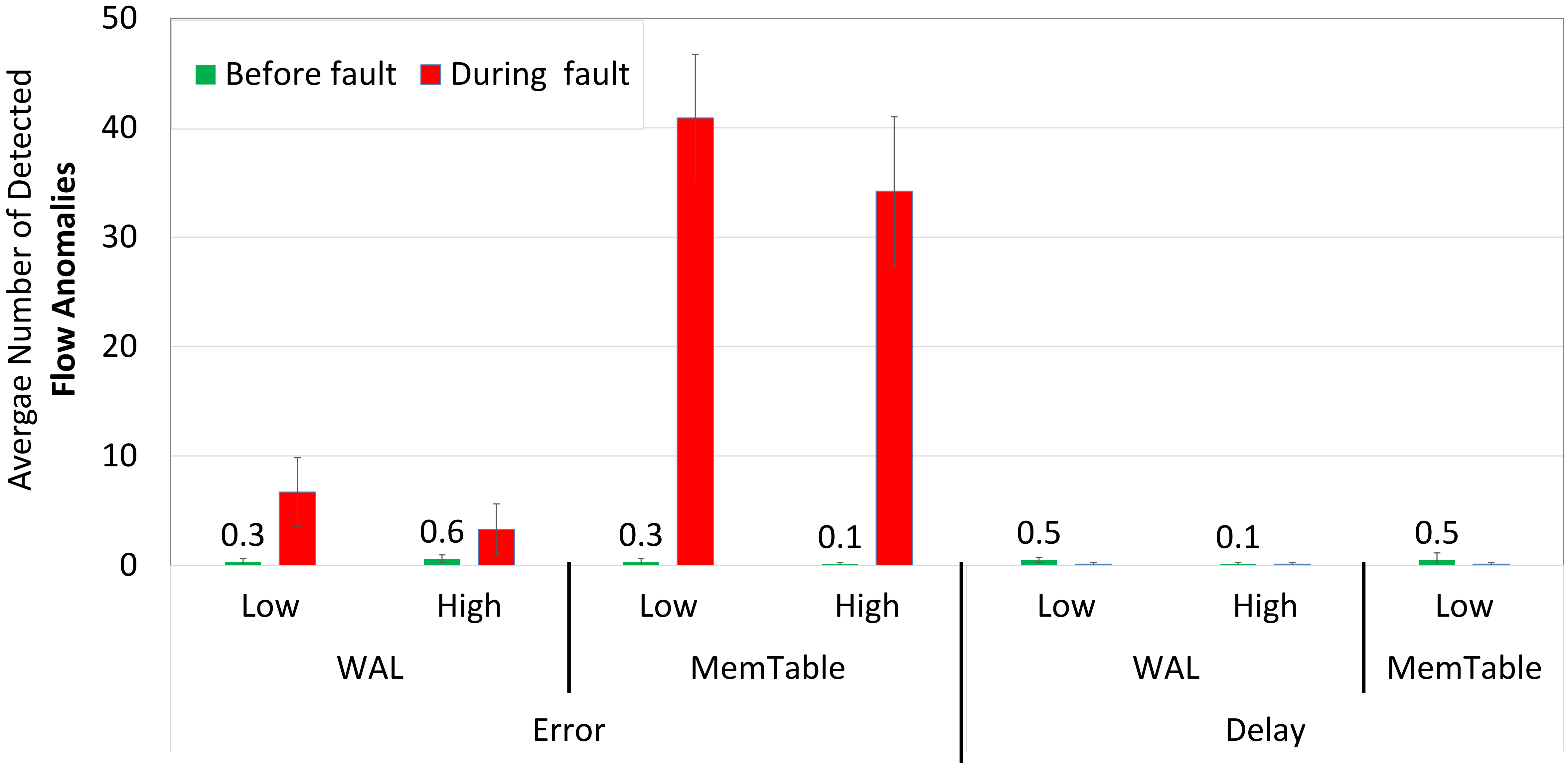
Up to 1000x storage saving vs. verbose logging

Runtime Overhead

No overhead when using SAAD vs. default (INFO) logging

False Positive: Flow anomalies

Each experiment : 90min (10 times)



1 false alarm per hour

Stage Aware Anomaly Detection

Provides context to the anomaly

- Code module (stage)
- Execution flow (log points)

In Real Time, with Low overhead

- Minimal runtime overhead
- Low storage cost (synopses instead of logs)
- Low processing cost

Portable, evaluated on 3 distributed storage systems

Questions

Details in Saeed Ghanbari’s PhD Thesis (2014)

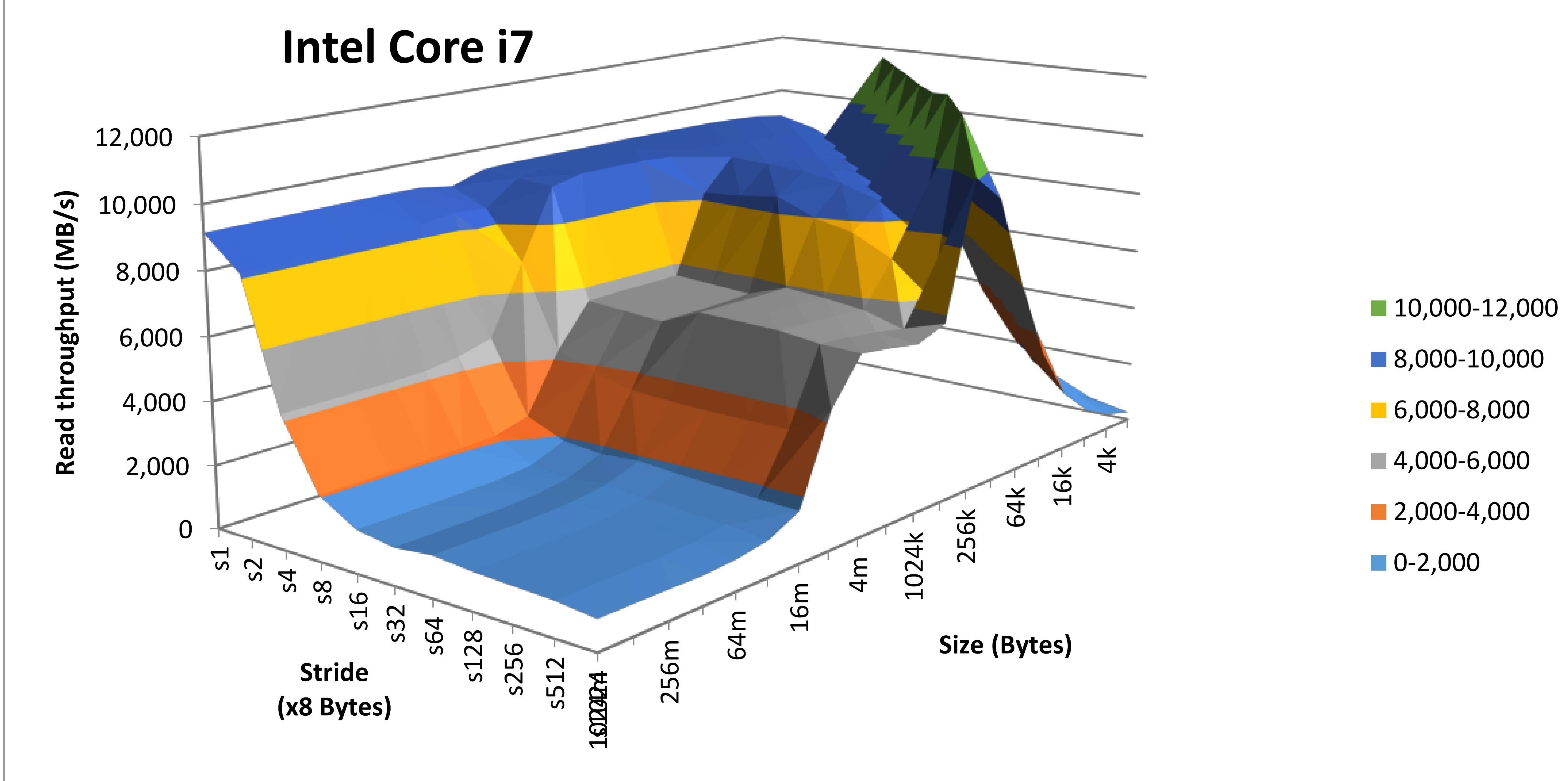
and in ACM/IFIP/Usenix Middleware 2014

In collaboration with

Saeed Ghanbari and Ali Hashemi

Ongoing Work: Model Transformation

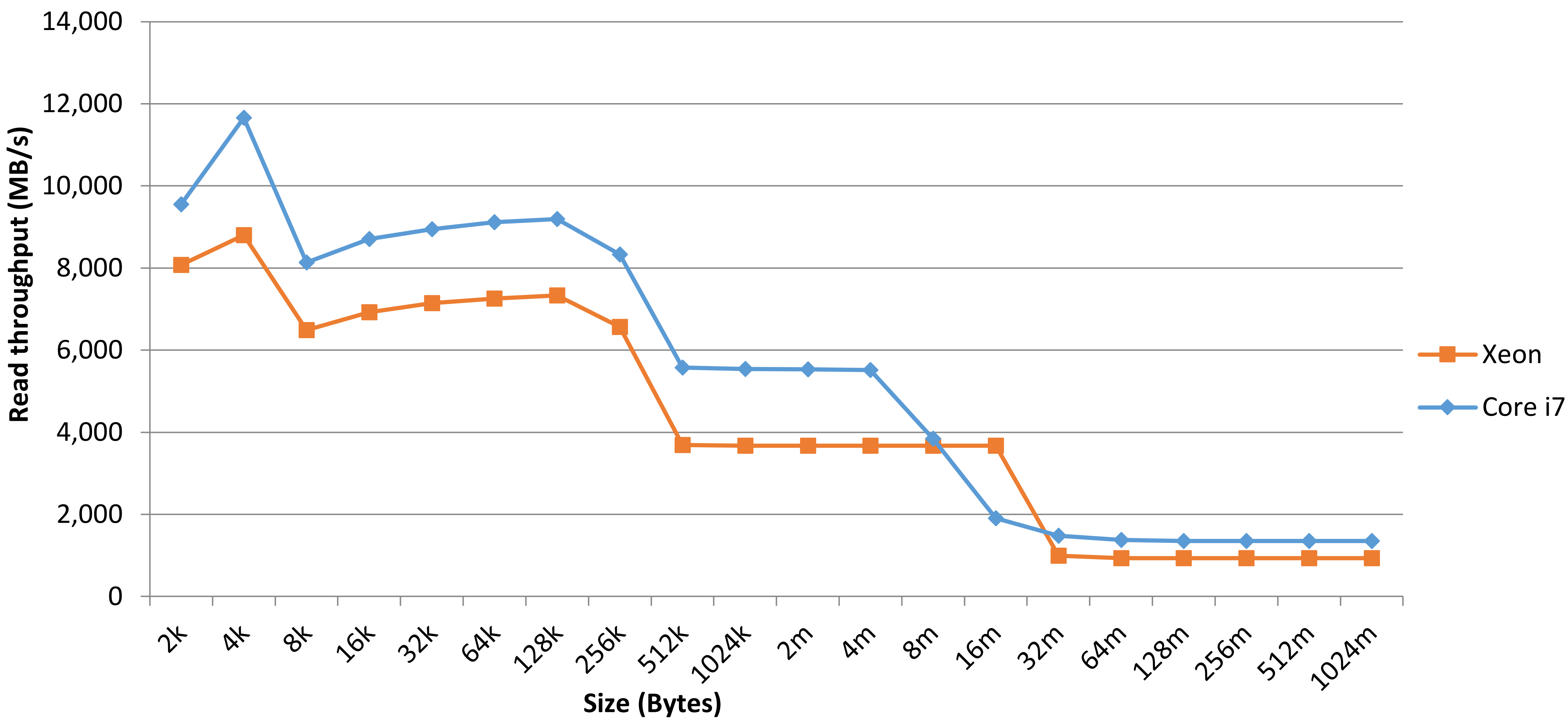
Cache	L1	L2	L3
Core i7	256 kB	1024 kB	8192 kB



Guidance: e.g., 3D to 2D reduction

	L1	L2	L3
Xeon	256 kB	2048 kB	20480 kB
Core i7	256 kB	1024 kB	8192 kB

s16



With minimum new samples …

Cache	L1	L2	L3
Xeon	256 kB	2048 kB	20480 kB