MittOS

Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface

Mingzhe Hao, Huaicheng Li, Michael Hao Tong,
Chrisma Pakha, Riza O. Suminto, Cesar A. Stuardo,
Andrew A. Chien, and Haryadi S. Gunawi

THE UNIVERSITY OF CHICAGO

CERES
Center for Unstoppable Computing

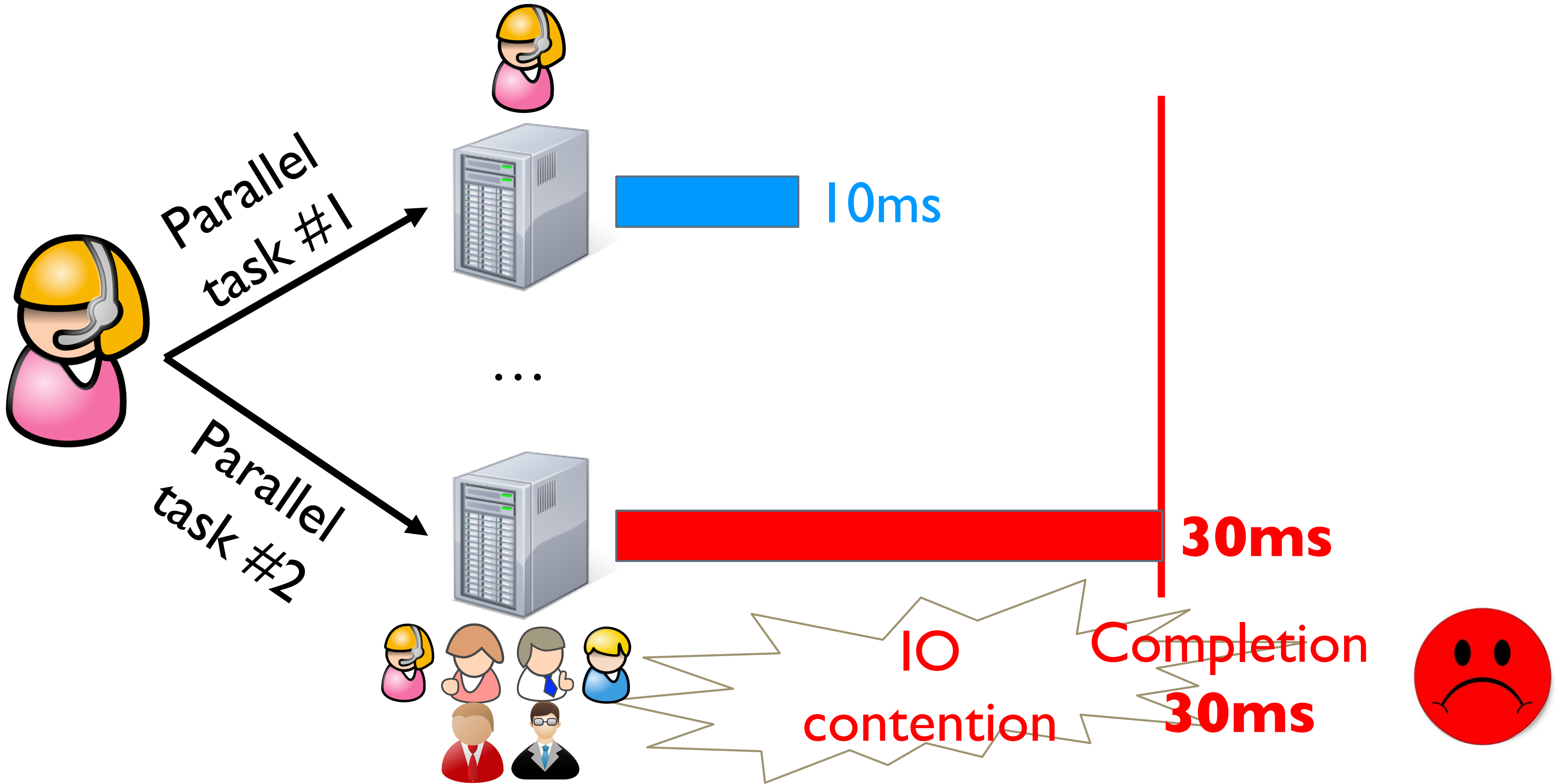
Millisecond Matters!

AMAZON: "EVERY 100MS OF LATENCY COSTS 1% IN SALES"

TABB GROUP: "BROKER COULD LOSE AS MUCH AS $4 MILLION IN REVENUES PER MILLISECOND IF ITS ELECTRONIC TRADING PLATFORM WAS ONLY 5MS BEHIND THE COMPETITION"

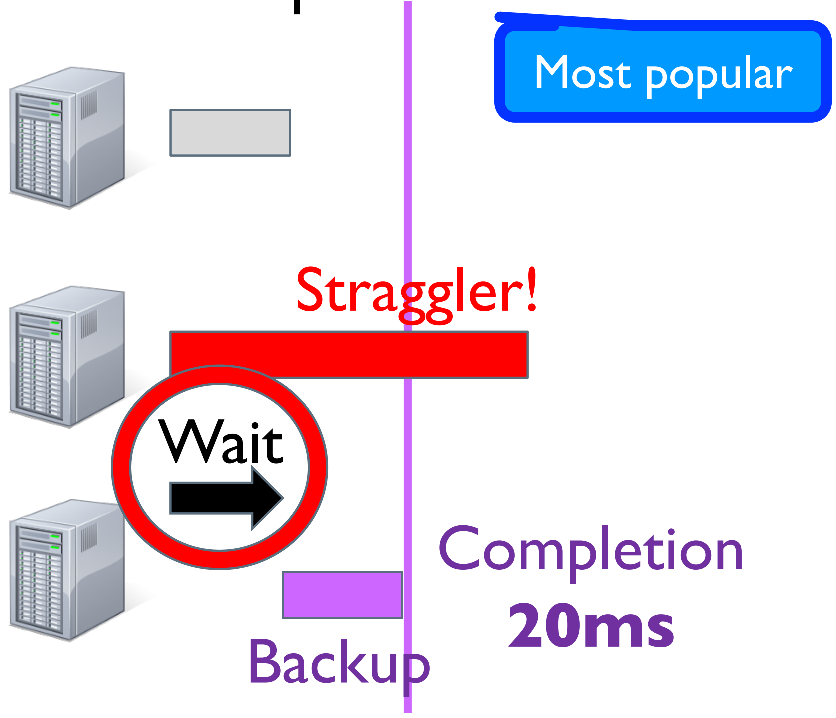
GOOGLE: "EXTRA 500MS IN SEARCH PAGE GENERATION TIME DROPPED TRAFFIC BY 20%"

Millisecond Tail Latency



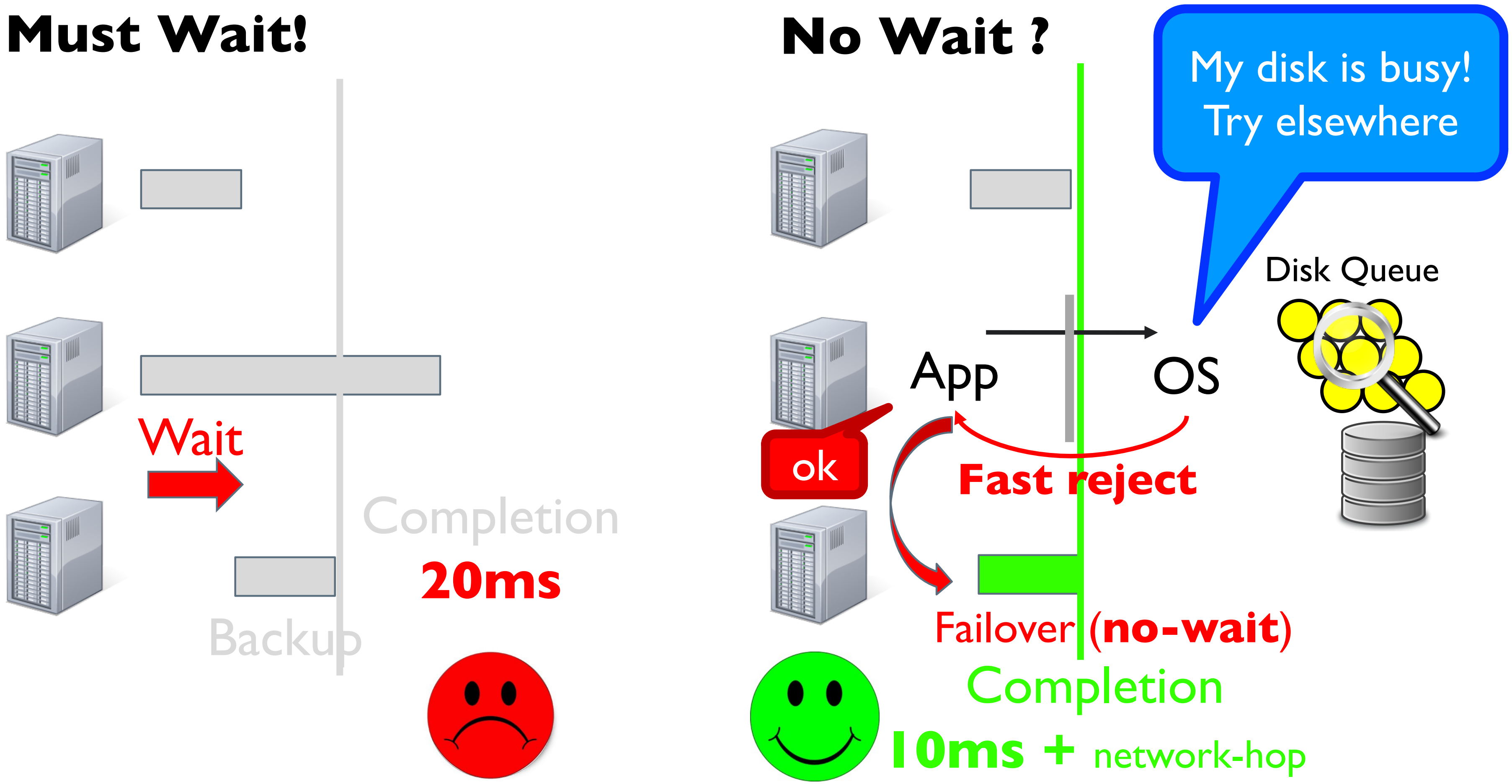
Current Tail-Tolerant Mechanisms

1. Speculation



2. Cloning
   - Introduces 2x workload

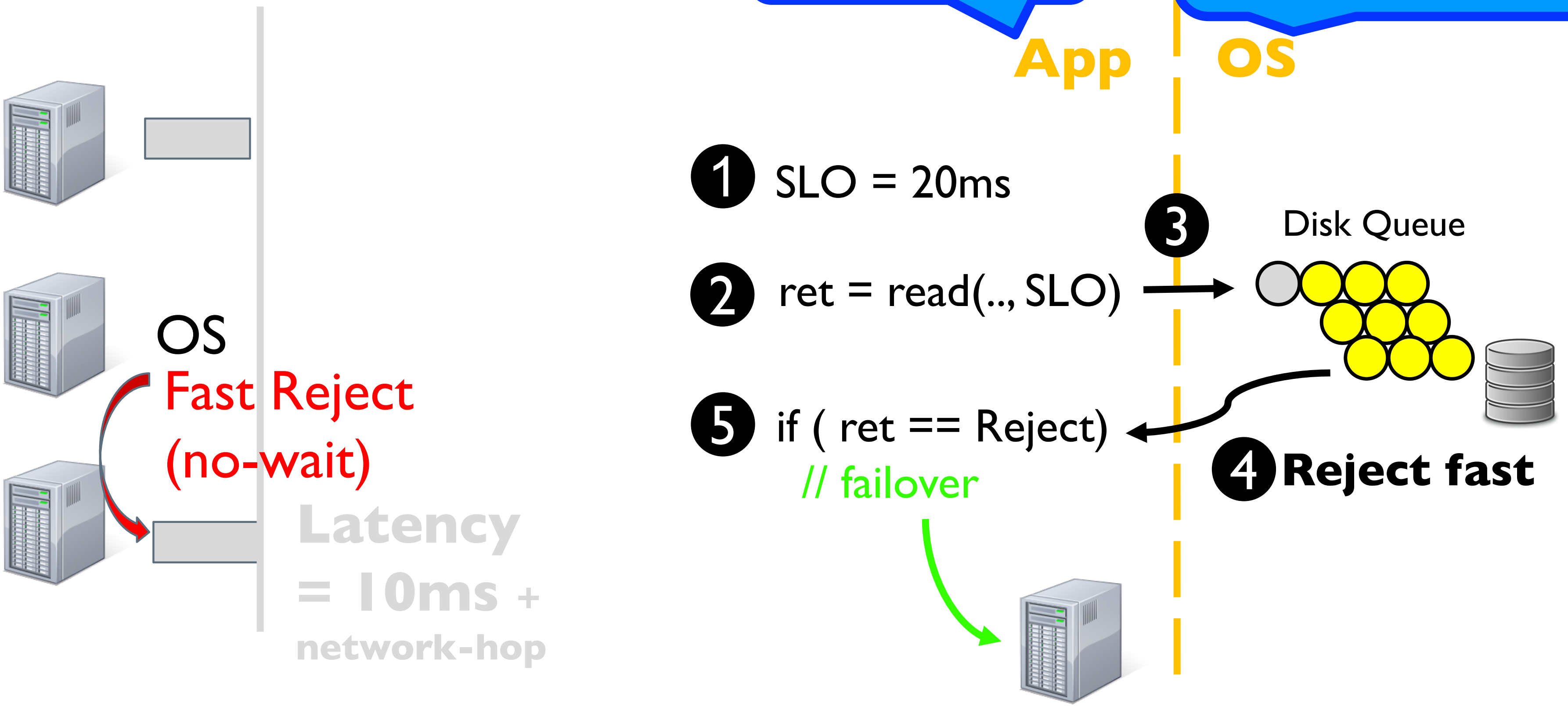
3. Snitching
   - Does not work when burstiness fluctuates in ms-level


Must Wait!
Wait
Backup
Completion
20ms
No Wait ?
My disk is busy!
Try elsewhere
Disk Queue
App
OS
ok
Fast reject
Failover (no-wait)
Completion
10ms + network-hop

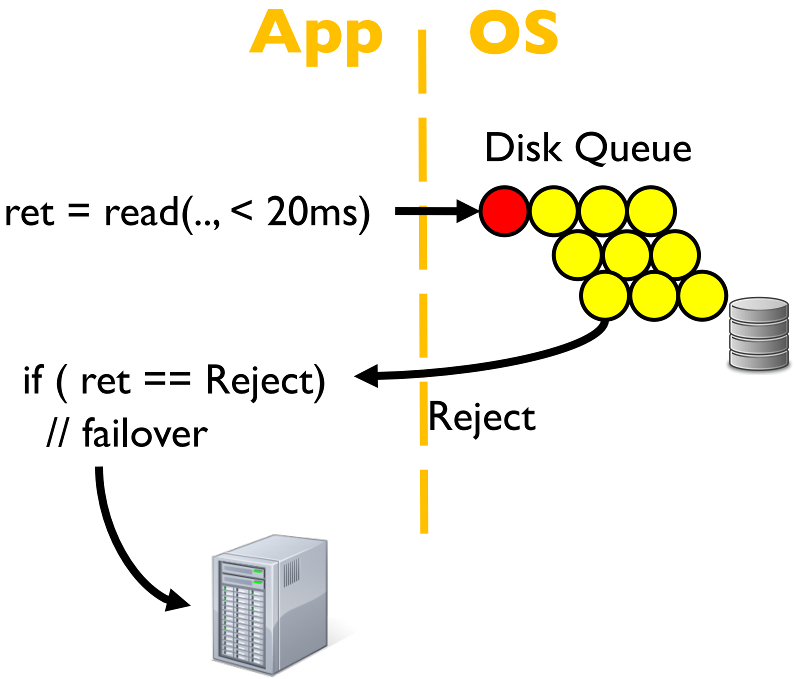
Use-Case

I want **< 20ms** latency

OS can see *"everything"* and tell app when **it is busy**



MittOS



- MittOS **Principles**
  - SLO-aware interface
  - Reject fast
  - *Transparent* of *busyness*
    - **PC** era: is best effort (cannot reject IOs)
    - **DC** era: Less-busy replicas available

Challenge

ret = read(.., < 20ms) **App**

OS

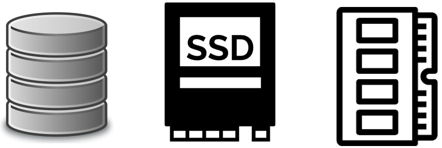
Should I reject this IO?

?

What is the OS ***queue policy?***

FIFO, elevator, CFQ, etc.

What is the **device type?**



Prediction depends on queue policy and device type

Contribution

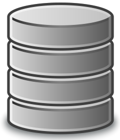
+50 LOC

MittOS-powered MongoDB riak

Fast Reject Interface

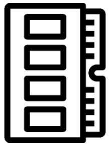
MittOS Latency Prediction

Disk



Open-Channel SSD

OS Cache



MittOS principle: Support fast rejecting SLO-aware interface

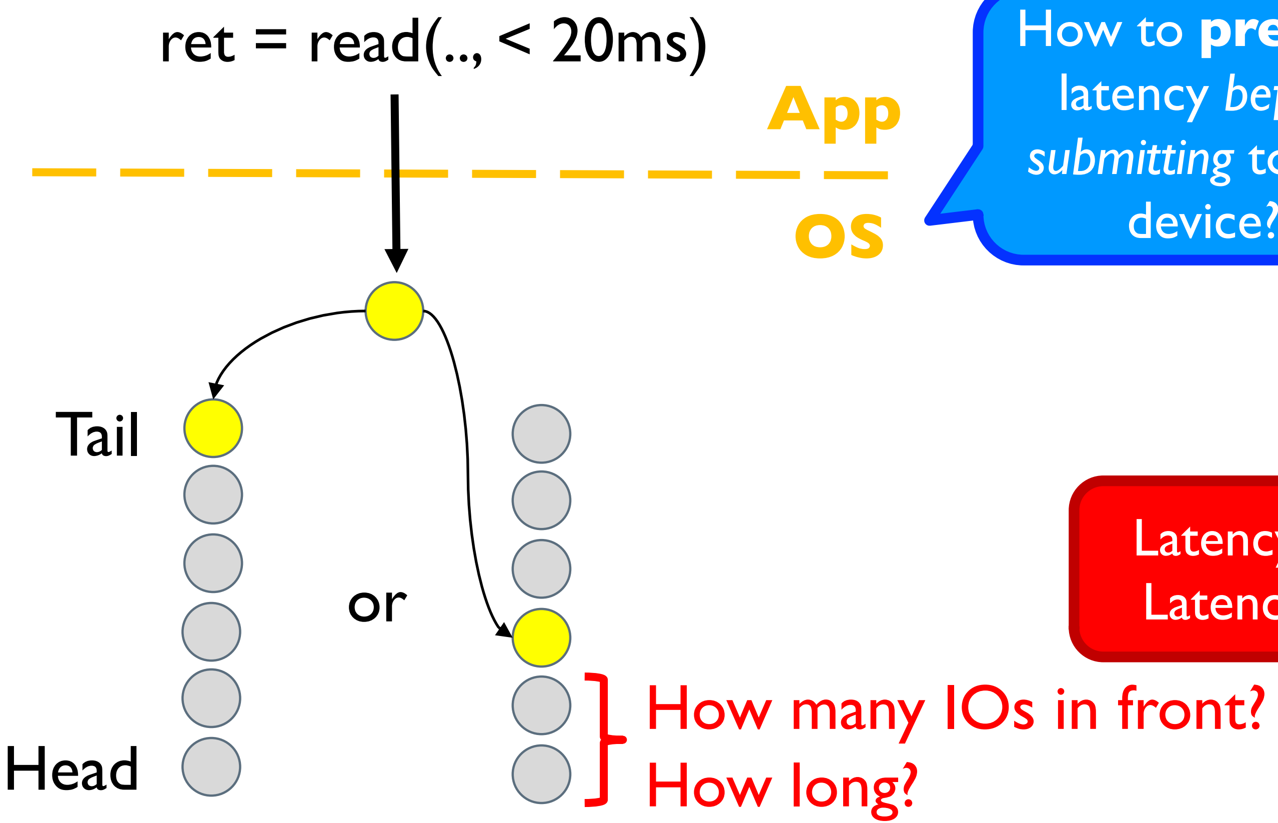
vs. **state of the art**: hedged requests, cloning, application timeout, etc.

Cut tail:
50% latency reduction above 75 percentile

Outline

- Introduction
- **Design**
  - Challenges
  - Solutions
- Evaluation
- Conclusion

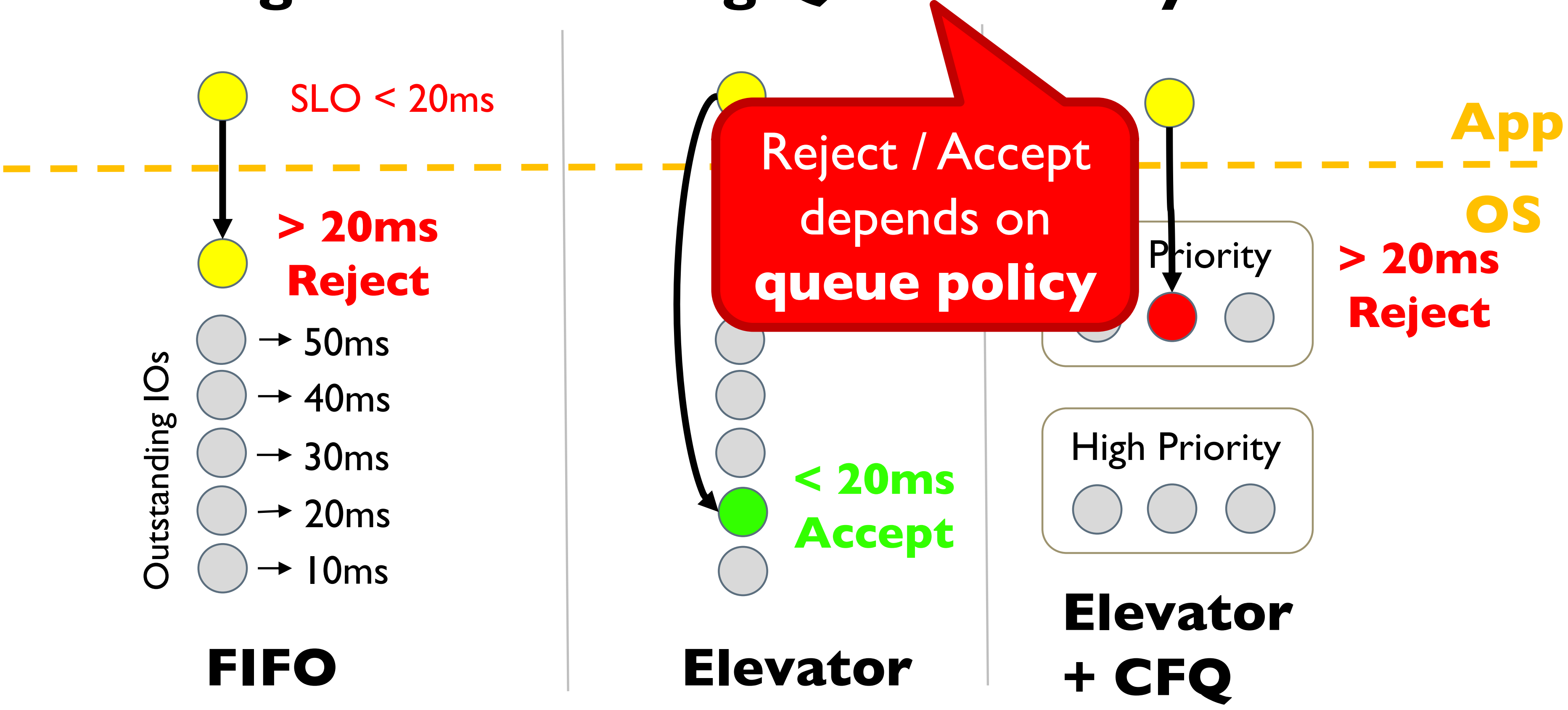
Prediction



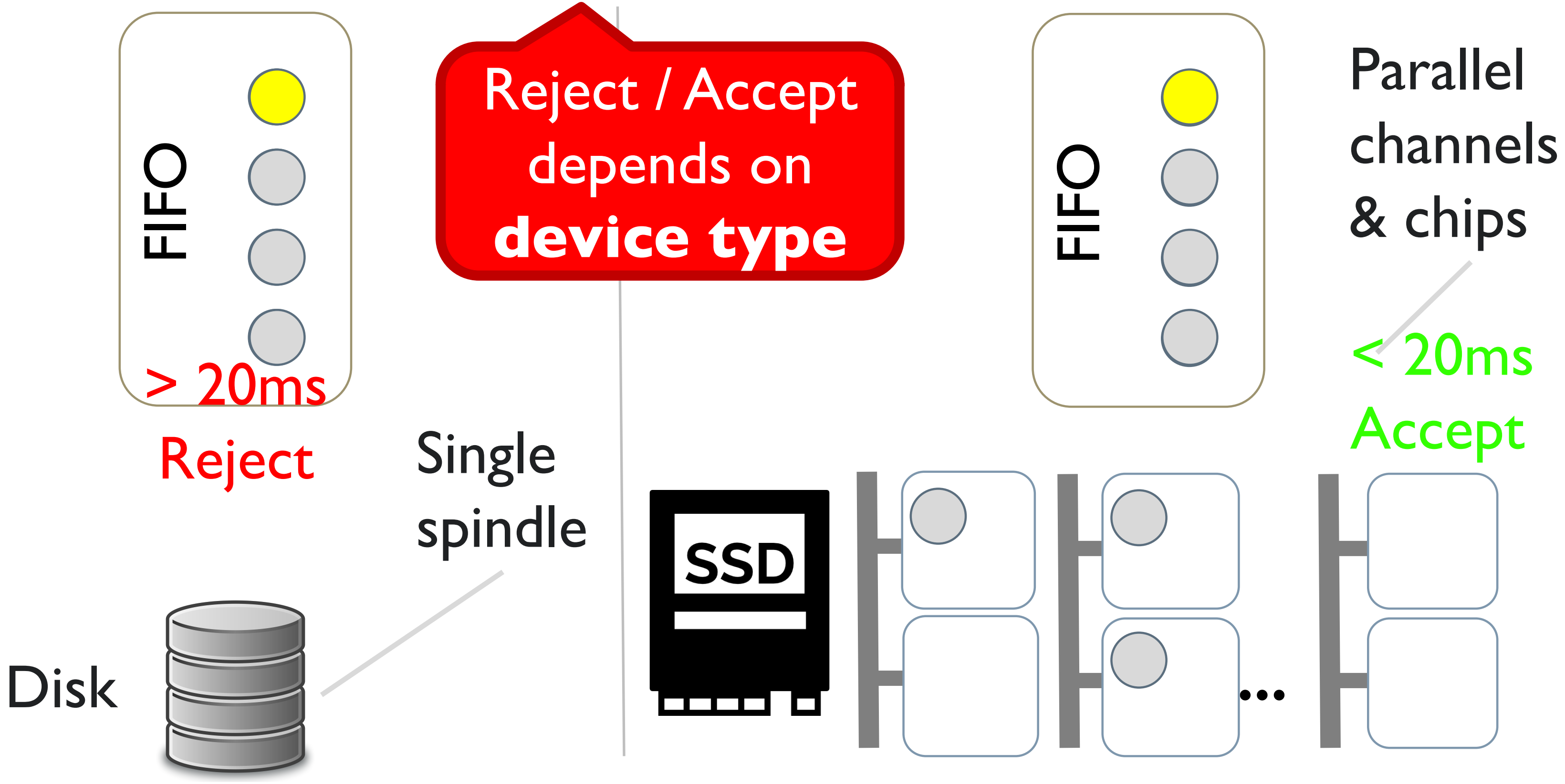
How to **predict** latency *before submitting* to the device?

Latency < SLO → **Accept**
Latency > SLO → **Reject**

Challenge #1: Modeling Queue Policy

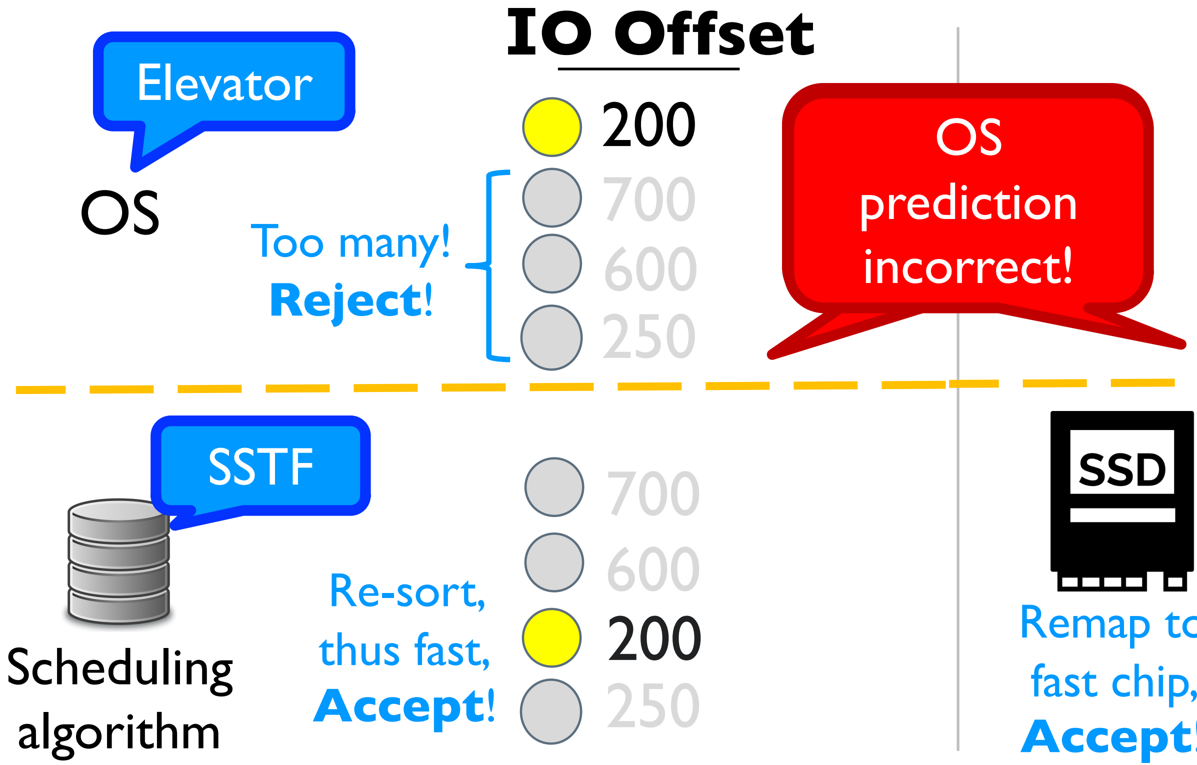


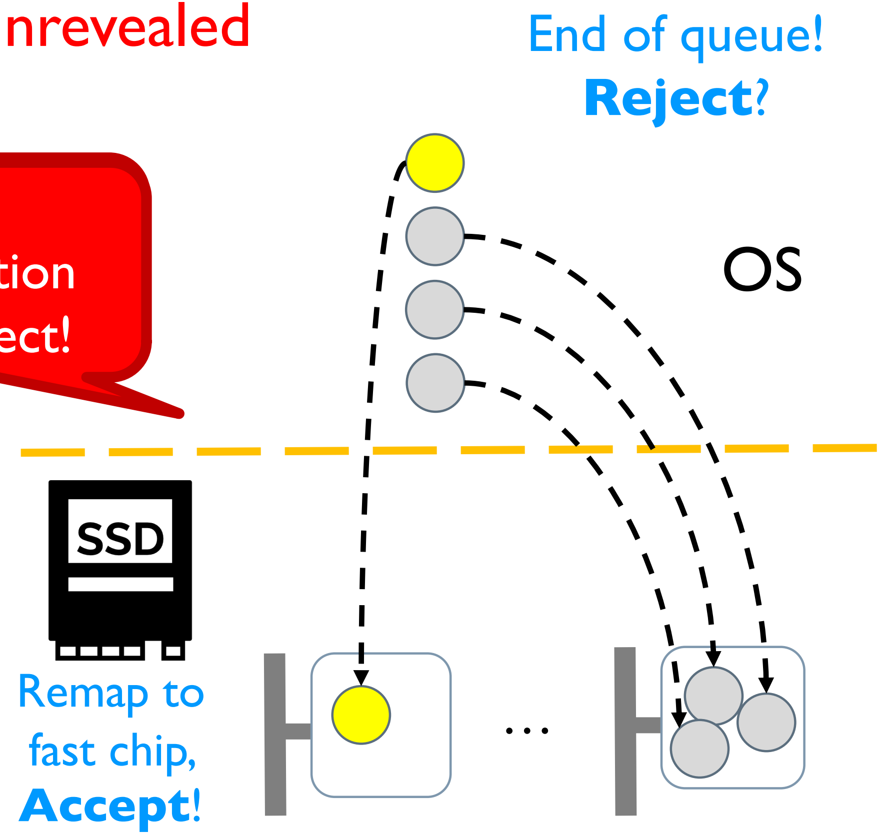
Challenge #2: Device Type



Challenge #2: Device Type

Idiosyncrasies of devices are mostly unrevealed





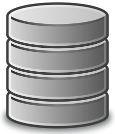
Outline

- Introduction
- **Design**
  - Challenges
  - Solutions
- Evaluation
- Conclusion

Reject/Latency Prediction

Reject? = f(SLO, queue policy, device type)

Get from source-code. e.g. CFQ, noop



Simple type

Profiling is enough

MittCFQ

Complicated type

White-box knowledge required

MittSSD

MittCFQ

Open-sourced

Reverse engineer

Which tree/queue each IO belongs to?

How many IOs in front?

...gement

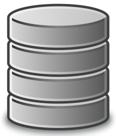
...es

...orities

Contains ~4500 LOC

CFQ

OS



Black box

Disk Scheduling?

Seek latency?
(depends on seek distance)

Transfer time?
(depends on IO size)

MittCFQ Profiling

For each interval in [100MB, 200MB, ..., 1GB] do:

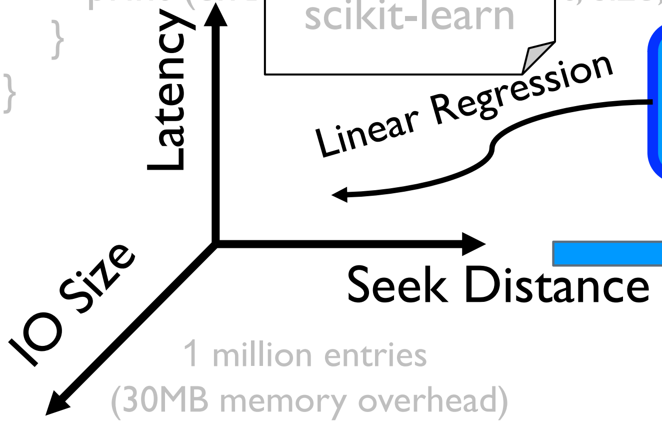
```
for (startOffset = 0; startOffset < maxOffset; startOffset += interval) {
  for (endOffset = 0; endOffset < maxOffset; endOffset += interval) {
    for (size = 0; size < maxSize; size += sizeInterval){
      start_ts = gettimeofday();
      seek (startOffset);
      read (endOffset, size);
      end_ts = gettimeofday();
      latency = start_ts - end_ts;
      print (endOffset, endOffset, size, latency);
    }
  }
}
```

Random seek

Random read

Collect latency

scikit-learn



1 million entries
(30MB memory overhead)
For 1TB drive

2 disk models
11-hour profiling

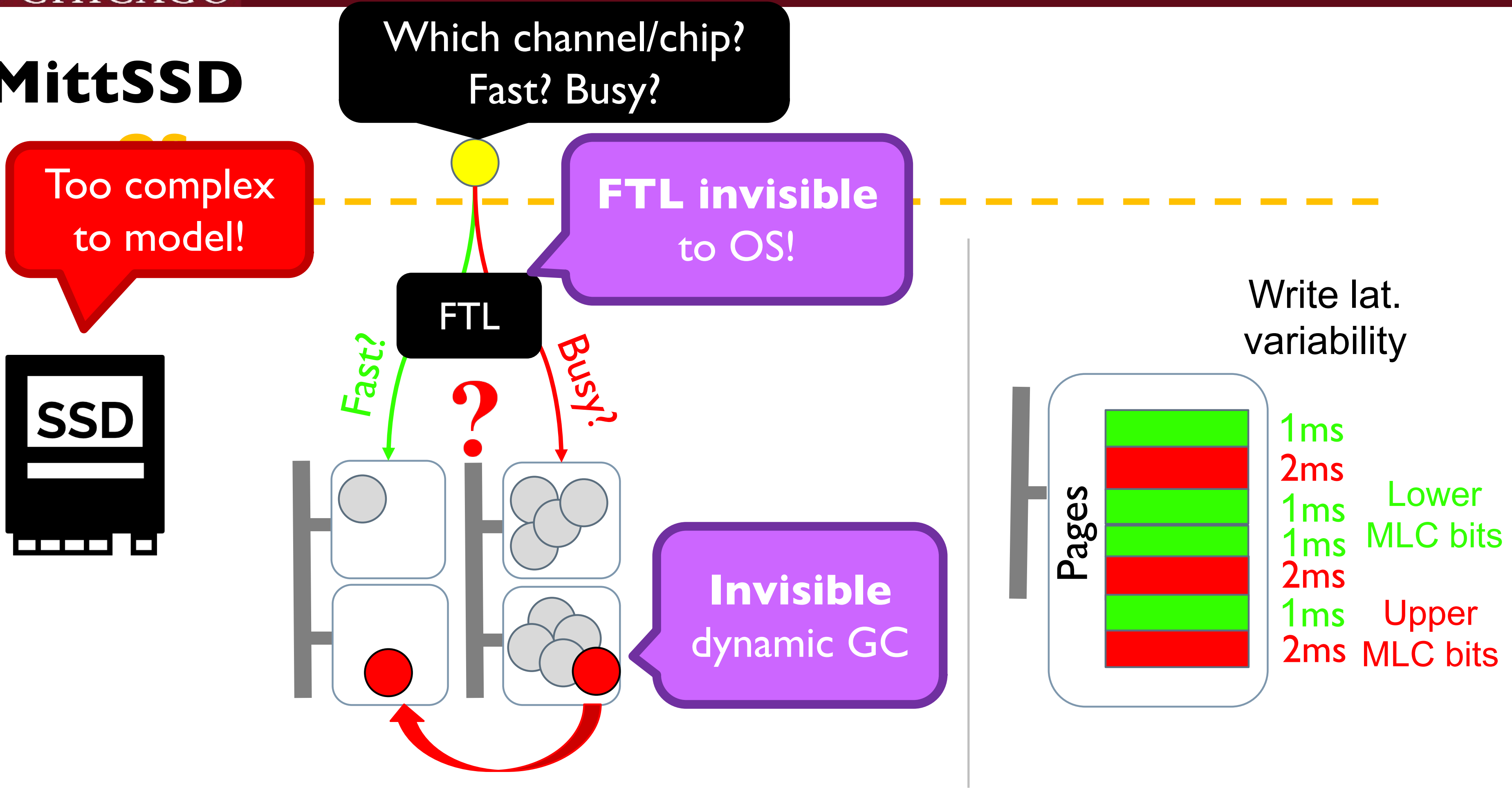
+ concurrent IO profiling

Infer

SSTF scheduling

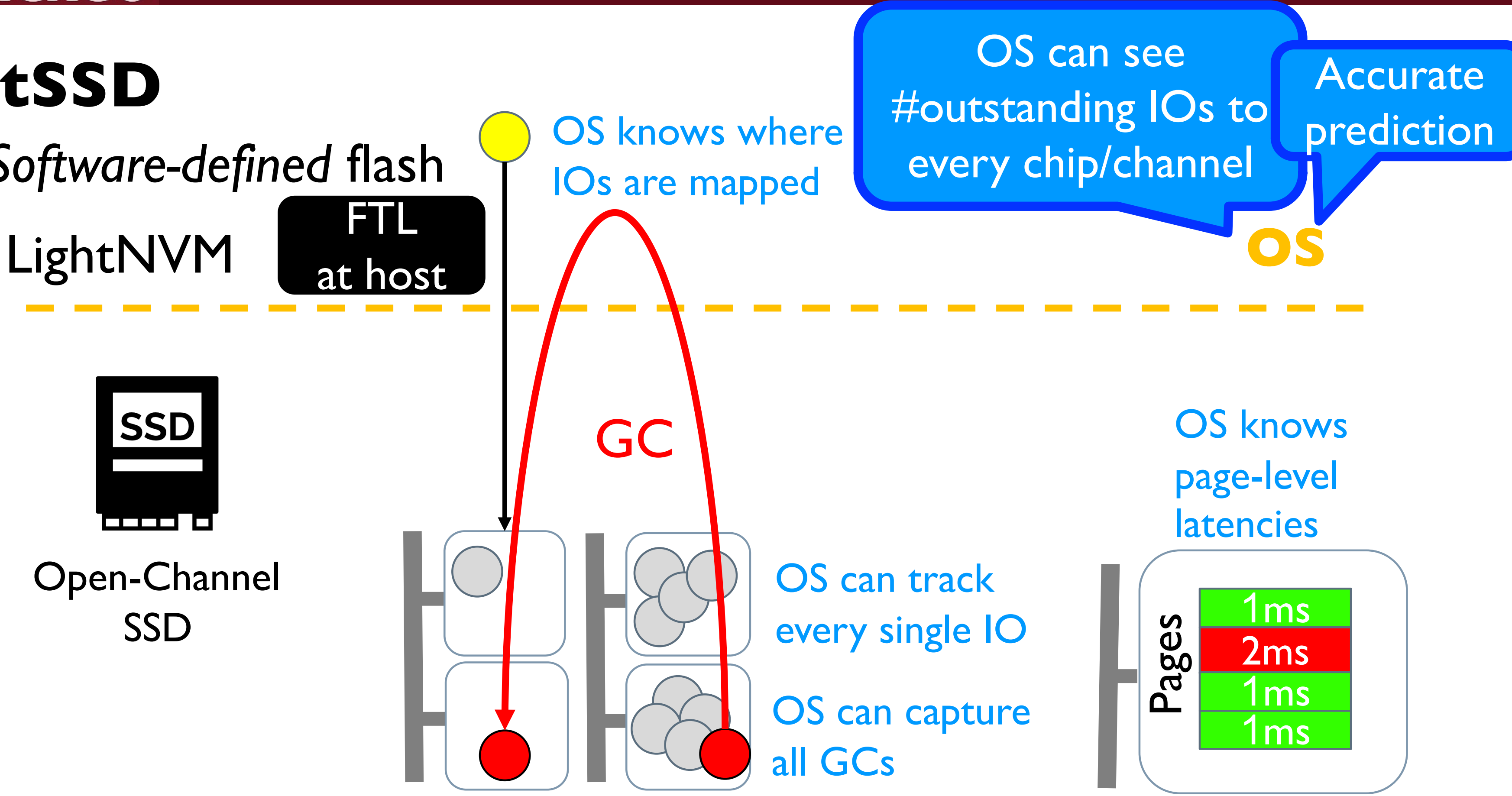
Accurate prediction

MittSSD


Which channel/chip?
Fast? Busy?
Too complex
to model!
FTL invisible
to OS!
SSD
FTL
Fast?
Busy?
?
Invisible
dynamic GC
Write lat.
variability
Pages
1ms
2ms
1ms
1ms
2ms
1ms
2ms
Lower
MLC bits
Upper
MLC bits

MittSSD

Software-defined flash



Reject/Latency Prediction

Reject? = $f($ SLO, queue policy, device type $)$

Reverse engineering based on source code

Simple type

Profiling is enough

MittCFQ ~**1800 LOC**

Complicated type

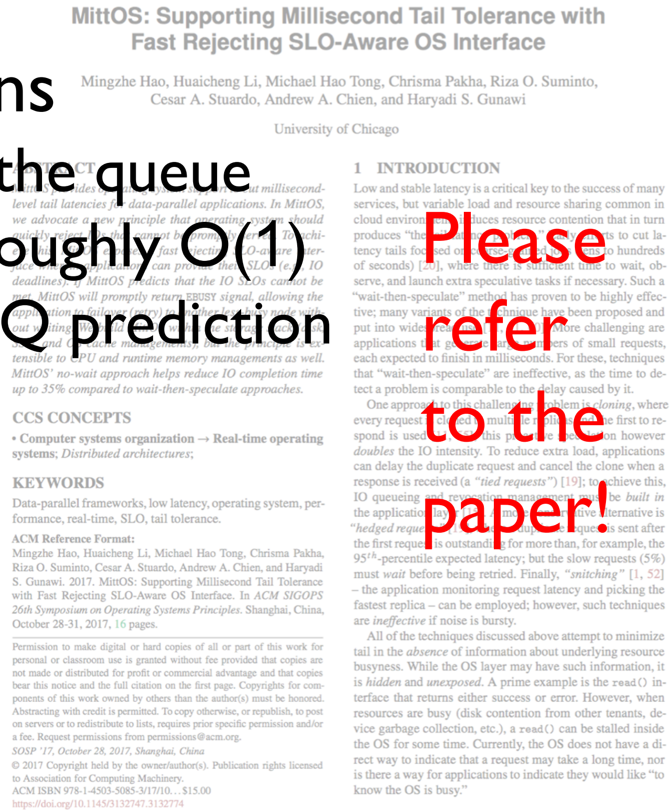
White-box knowledge required

MittSSD ~**1400 LOC**

LightNVM +
Open-Channel SSD

Other Solved Challenges

- Prediction overhead optimizations
  - Avoids going through every IO in the queue
  - Reduces overhead from O(n) to roughly O(1)
  - Shows < 5µs overhead for MittCFQ prediction
  - < 300ns for MittSSD prediction
- MittCache
  - Prediction for OS Cache


MittOS: Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface

Mingzhe Hao, Huaicheng Li, Michael Hao Tong, Chrisma Pakha, Riza O. Suminto, Cesar A. Stuardo, Andrew A. Chien, and Haryadi S. Gunawi

University of Chicago

CCS CONCEPTS

• Computer systems organization → Real-time operating systems; Distributed architectures;

KEYWORDS

Data-parallel frameworks, low latency, operating system, performance, real-time, SLO, tail tolerance.

ACM Reference Format:
Mingzhe Hao, Huaicheng Li, Michael Hao Tong, Chrisma Pakha, Riza O. Suminto, Cesar A. Stuardo, Andrew A. Chien, and Haryadi S. Gunawi. 2017. MittOS: Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface. In ACM SIGOPS 26th Symposium on Operating Systems Principles. Shanghai, China, October 28-31, 2017, 16 pages.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SOSP '17, October 28, 2017, Shanghai, China
© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-5085-3/17/10...$15.00
https://doi.org/10.1145/3132747.3132774

1 INTRODUCTION

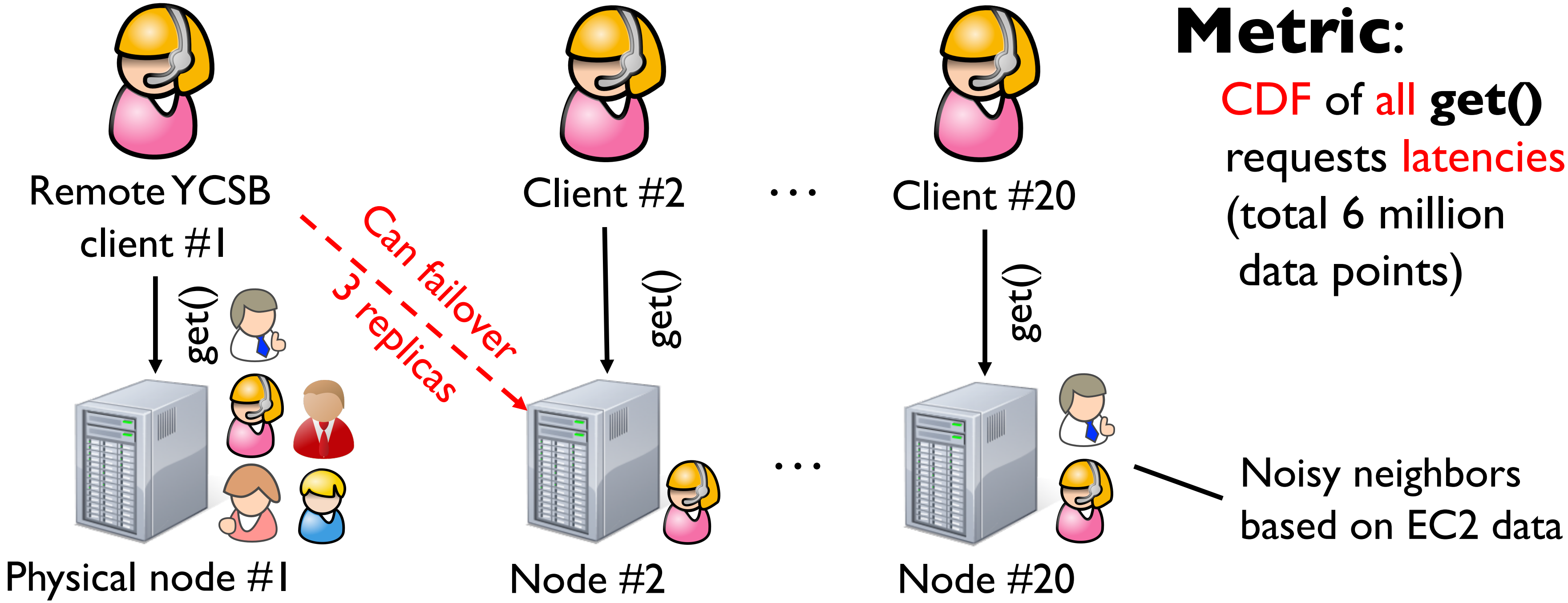
All of the techniques discussed above attempt to minimize tail in the absence of information about underlying resource busyness. While the OS layer may have such information, it is hidden and unexposed. A prime example is the read() interface that returns either success or error. However, when resources are busy (disk contention from other tenants, device garbage collection, etc.), a read() can be stalled inside the OS for some time. Currently, the OS does not have a direct way to indicate that a request may take a long time, nor is there a way for applications to indicate they would like "to know the OS is busy."

Please refer to the paper!

Outline

- Introduction
- Design
- **Evaluation**
  - Tail reduction
  - Latency prediction accuracy
- Conclusion

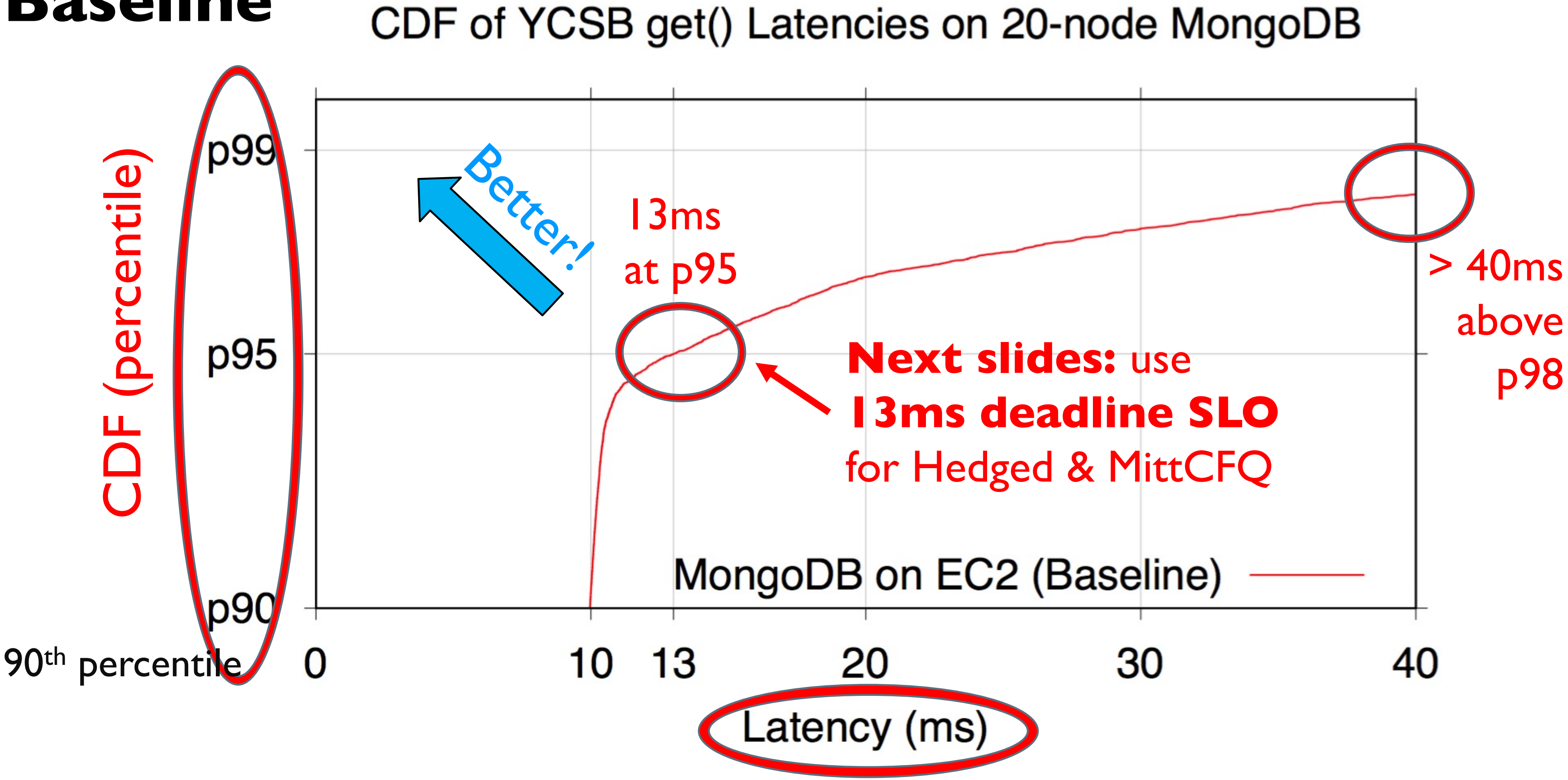
MittCFQ-powered MongoDB



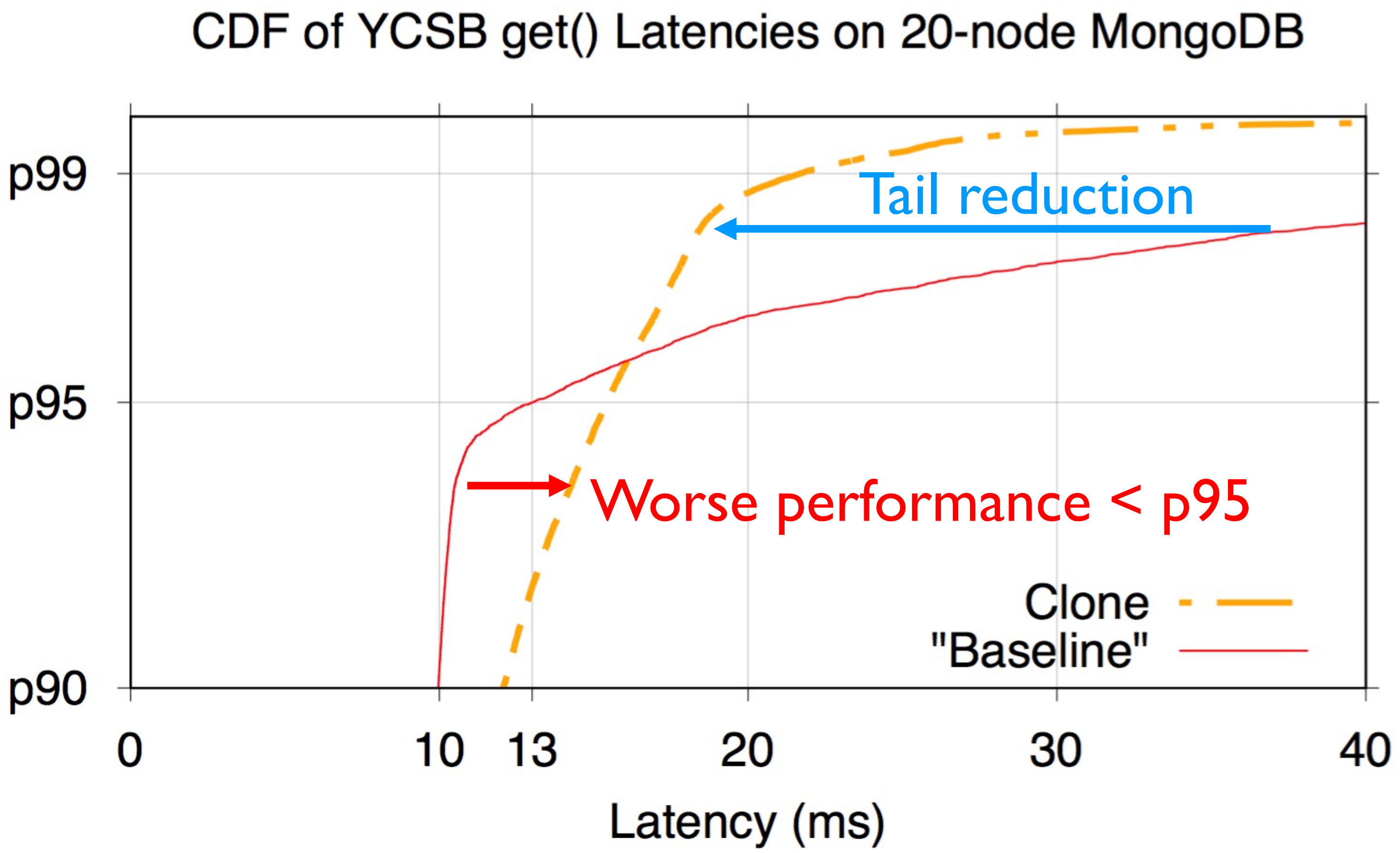
Metric:

CDF of all **get()** requests latencies (total 6 million data points)

Baseline


CDF of YCSB get() Latencies on 20-node MongoDB
CDF (percentile)
p99
p95
p90
90th percentile
Better!
13ms
at p95
> 40ms
above
p98
Next slides: use
13ms deadline SLO
for Hedged & MittCFQ
MongoDB on EC2 (Baseline)
0
10
13
20
30
40
Latency (ms)

Clone


CDF of YCSB get() Latencies on 20-node MongoDB
p99
p95
p90
0
10
13
20
30
40
Latency (ms)
Tail reduction
Worse performance < p95
Clone
"Baseline"

Hedged Requests

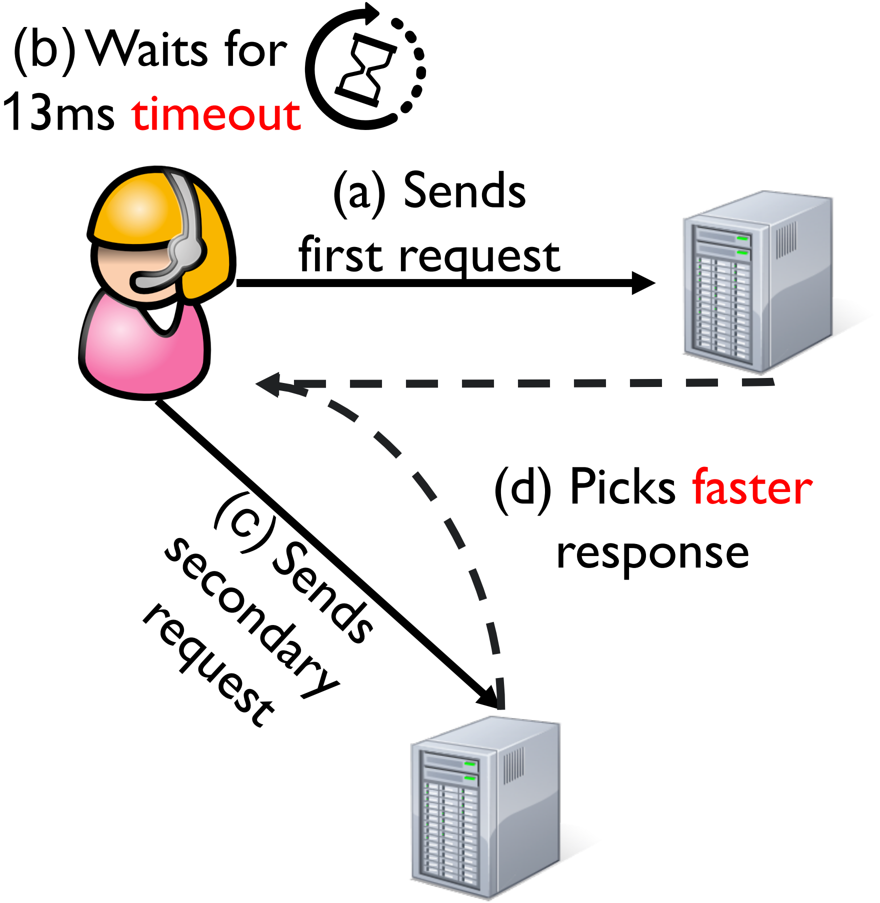

DOI:10.1145/2408776.2408794

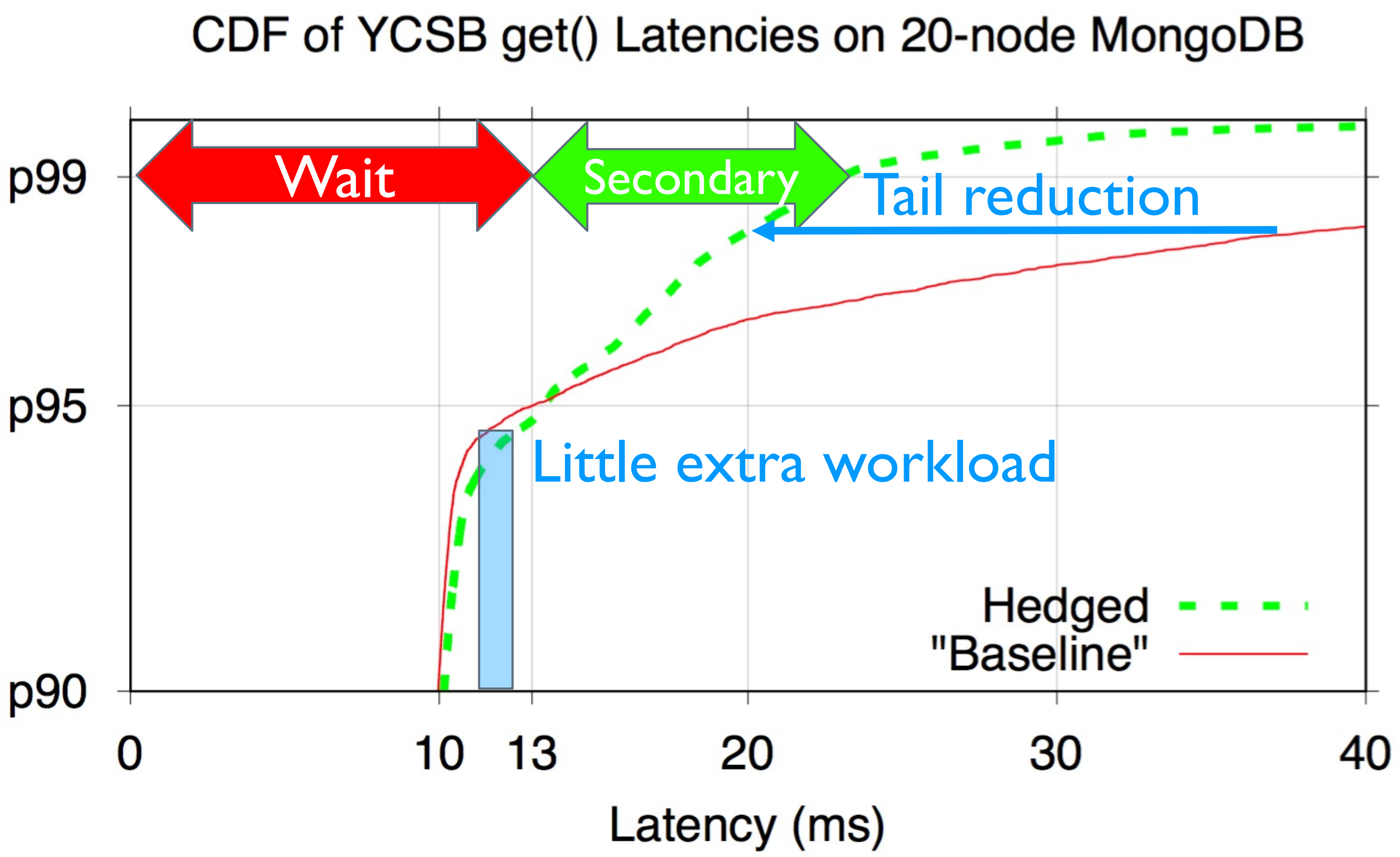
Software techniques that tolerate latency variability are vital to building responsive large-scale Web services.

BY JEFFREY DEAN AND LUIZ ANDRÉ BARROSO

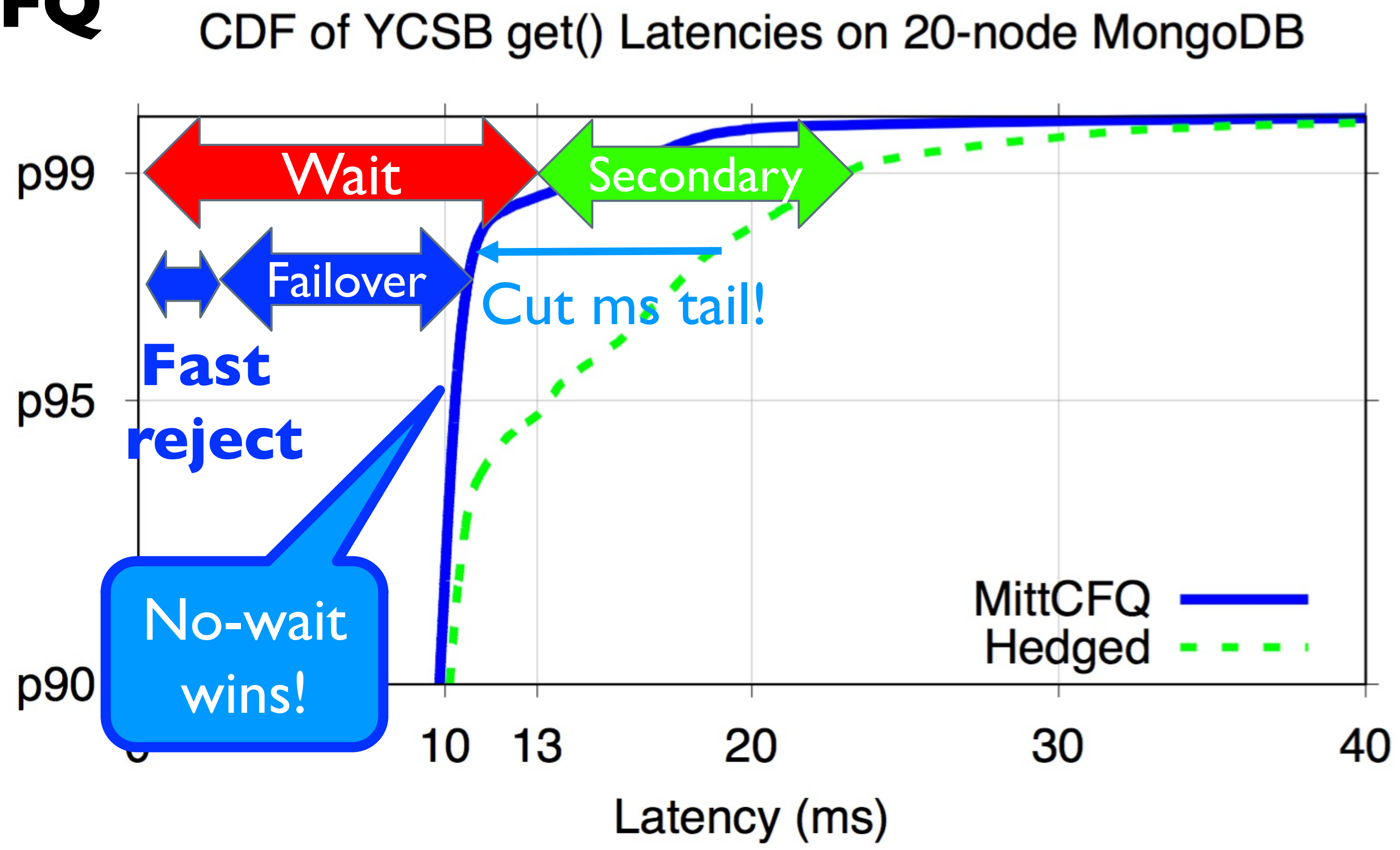
The Tail at Scale

Communications of the ACM, vol. 56 (2013), pp. 74-80

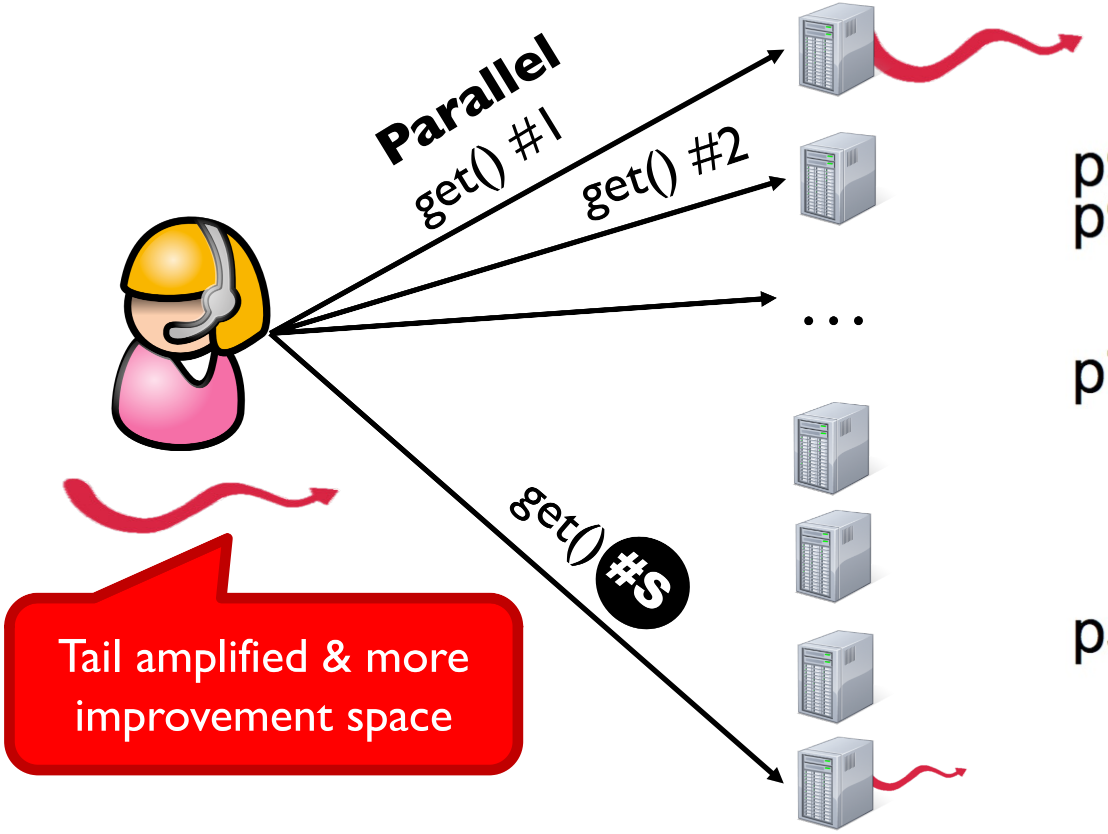


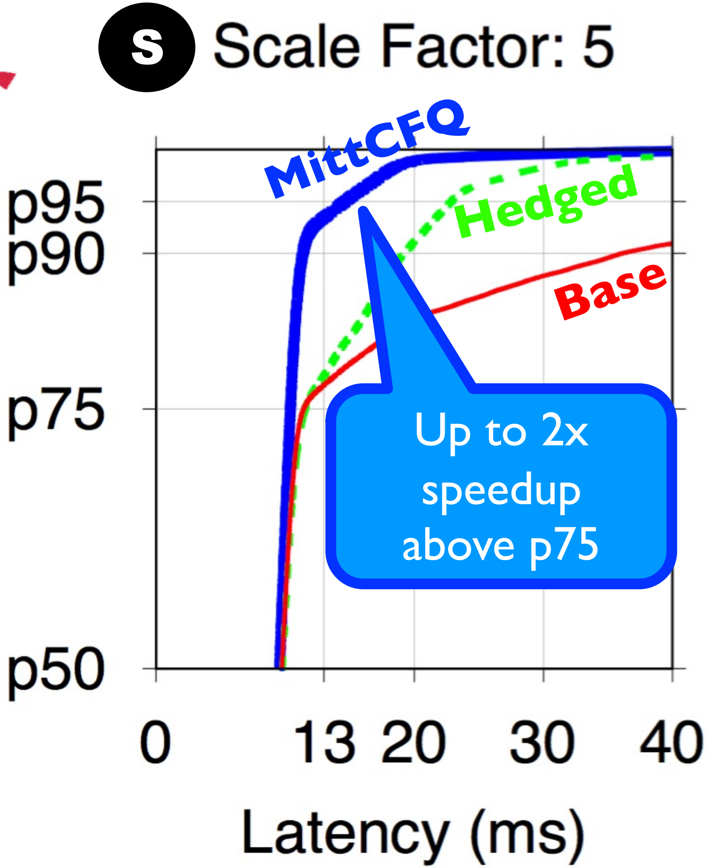

CDF of YCSB get() Latencies on 20-node MongoDB
Wait
Secondary
Tail reduction
Little extra workload
p99
p95
p90
0
10
13
20
30
40
Latency (ms)
Hedged
"Baseline"

MittCFQ


CDF of YCSB get() Latencies on 20-node MongoDB
p99
p95
p90
0
10
13
20
30
40
Latency (ms)
Wait
Secondary
Failover
Cut ms tail!
Fast reject
No-wait wins!
MittCFQ
Hedged

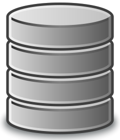
Tail amplified at Scale





Accuracy Evaluation

MittCFQ



Disk

MittSSD

Open-Channel SSD

5 real-world block-level traces

DAPPS

DTRS

TPCC

EXCH

LMBE

Metrics:

- False positive: IO rejected, but deadline is met
- False negative: Deadline violated, but IO is not rejected

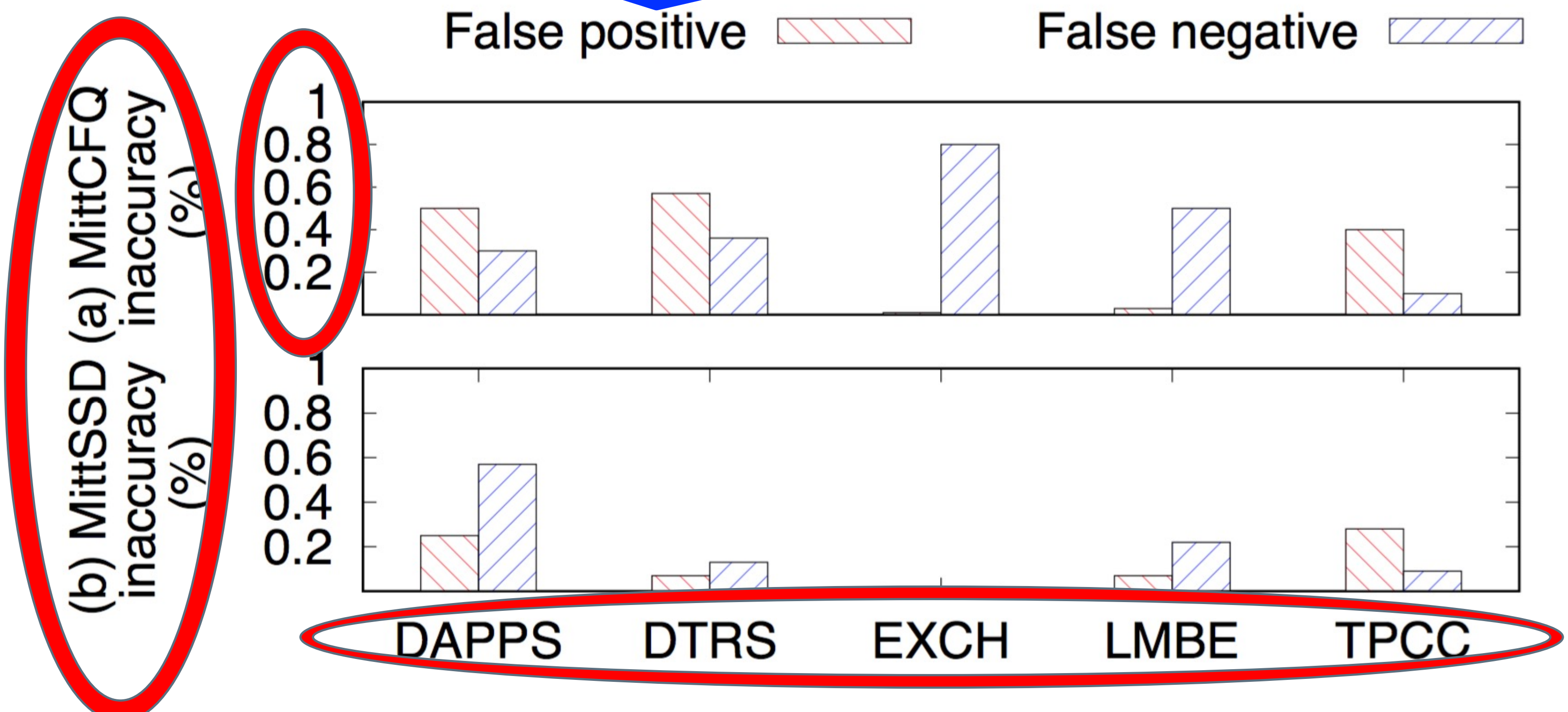
Accuracy Evaluation

Among *incorrect* cases:

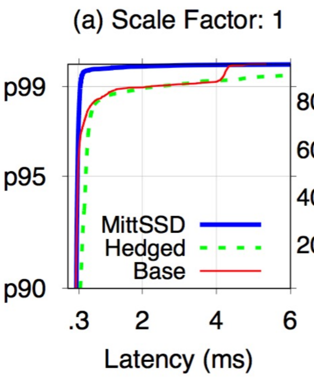
MittCFQ: < 3ms diff

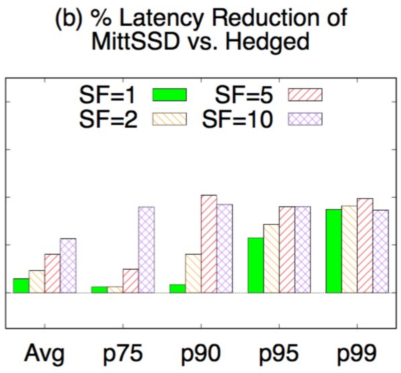
MittSSD: < 1ms diff

Only <1% inaccuracy!

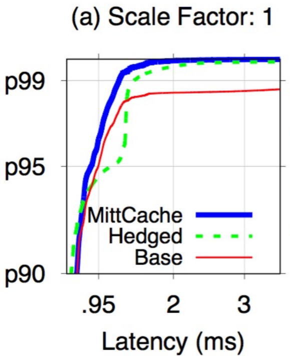


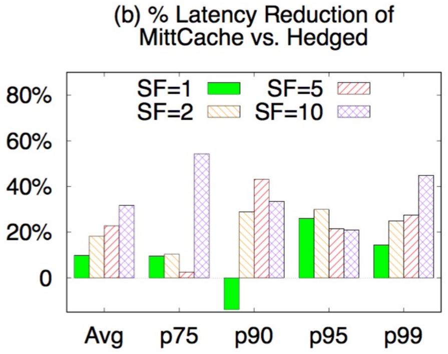
MittSSD



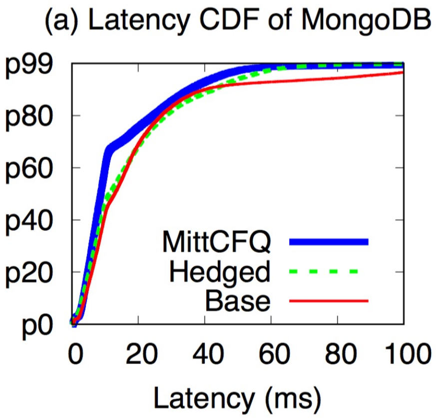


MittCache

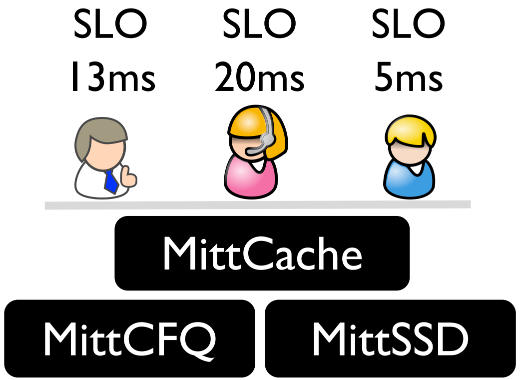




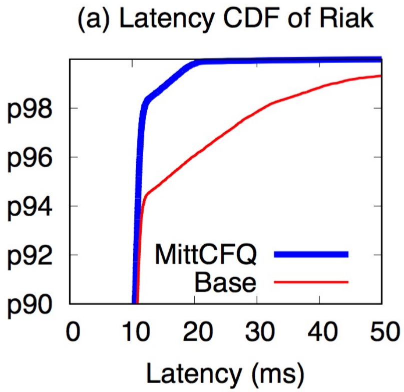
MongoDB + Filebench + Hadoop



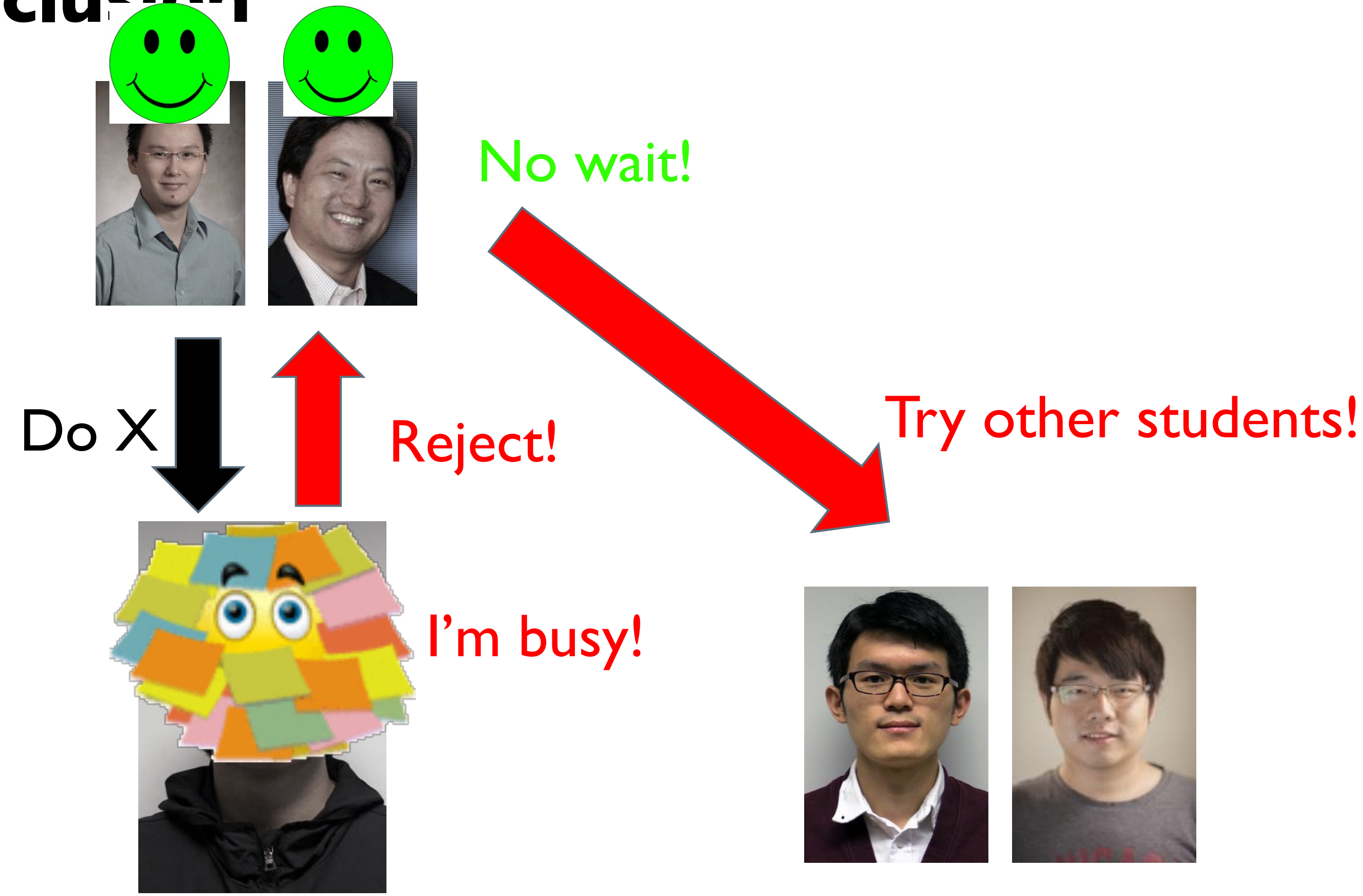
All in one



Riak



Conclusion



Conclusion

MittOS-powered **apps**

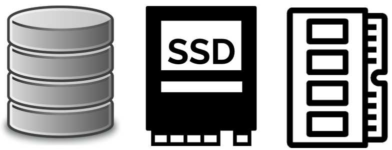
MongoDB

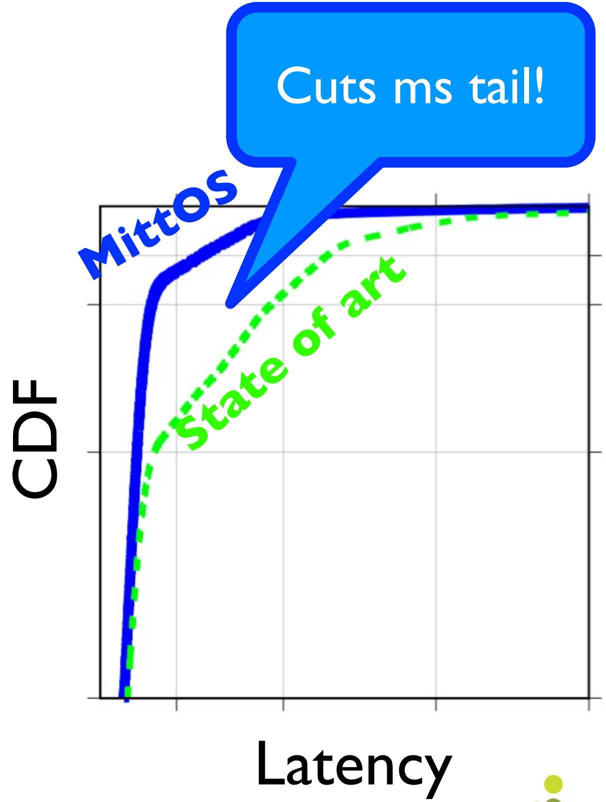
riak

Fast Reject (No-wait) Interface

MittOS

Latency Predictions





UCARE

http://ucare.cs.uchicago.edu

Thank you! Questions?

CERES
Center for Unstoppable Computing

https://ceres.uchicago.edu