Principled Schedulability Analysis for Distributed Storage Systems Using Thread Architecture Models

Suli Yang*, Jing Liu,

Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

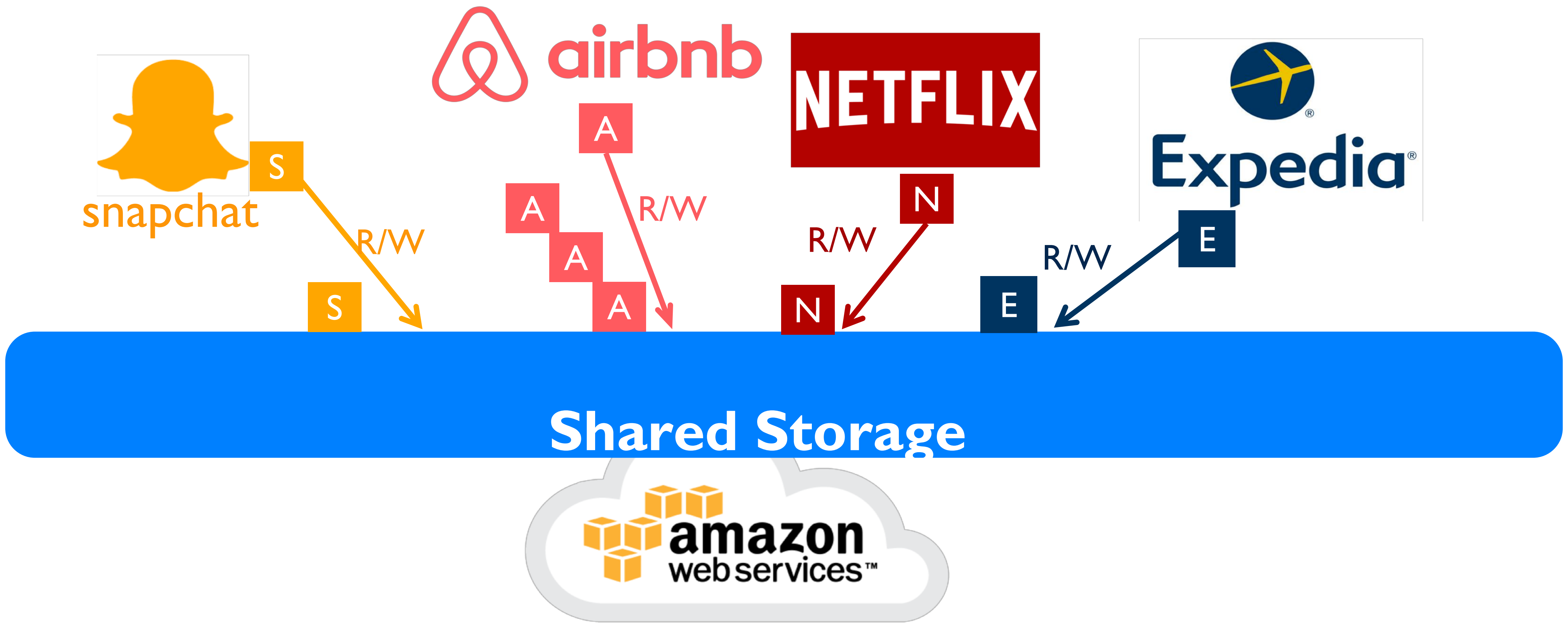
蚂蚁金服
ANT FINANCIAL

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

* work done while at UW-Madison

Scheduling: A Fundamental Primitive

- Modern storage systems are shared
- Correct and efficient request scheduling is indispensable



Broken Scheduling in Current Systems

- Popular storage systems have fundamental scheduling deficiencies

[MongoDB - #21858]:

“*A high throughput update workload … could cause starvation on secondary reads*”

[HBase - #8884]:

“ *… when the read load is high on a specific RS is high, the write throughput also get impacted dramatically, and even write data loss...*”

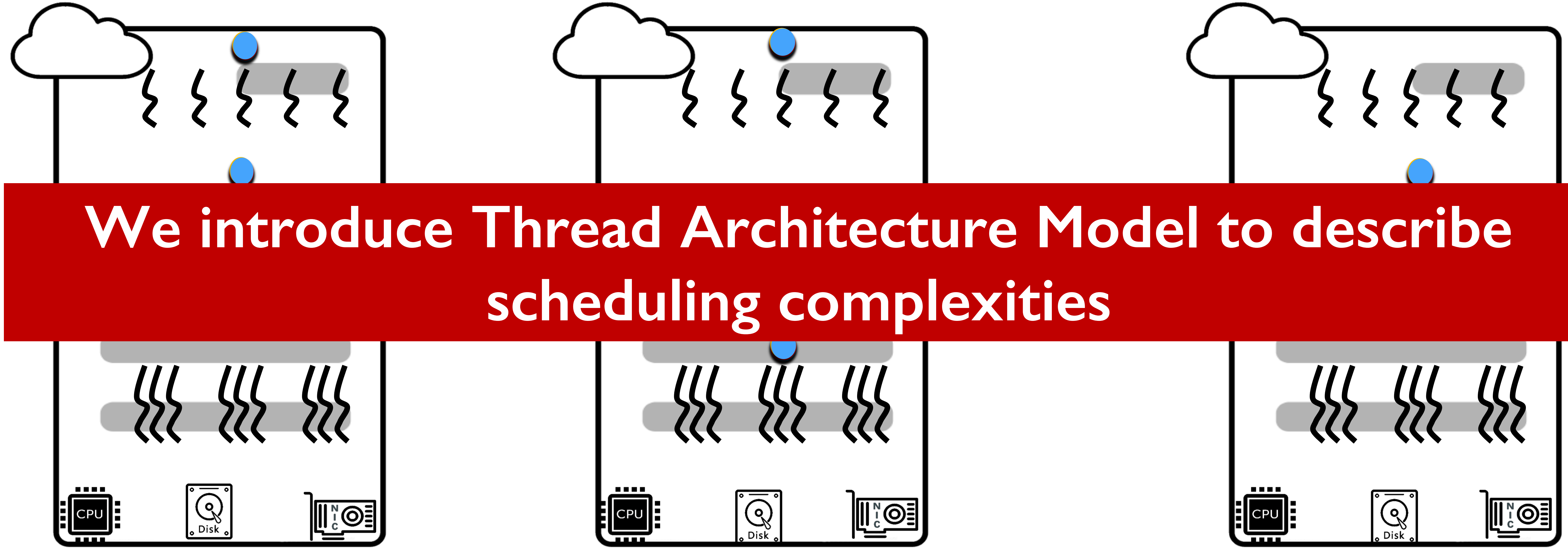
[Cassandra - #10989]:

“*inability to balance writes/reads/compaction/flushing…*”

etc.

Why Is Scheduling Broken?

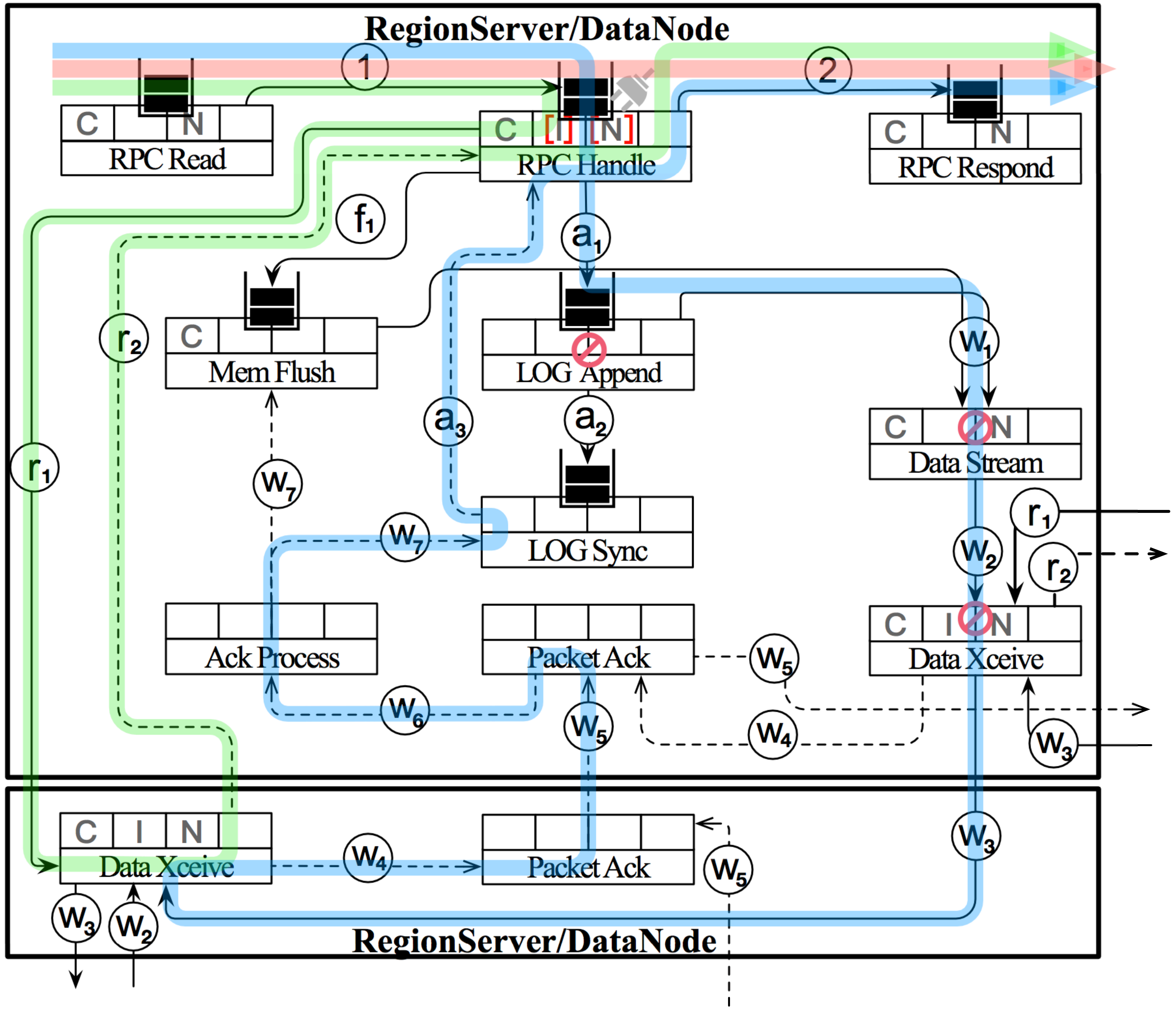
- The complexities in modern storage systems
  - Distributed: >1000 servers
  - Highly concurrent: ~1000 interacting threads in each server
  - Long execution path: requests traverses numerous threads across multiple machines



We introduce Thread Architecture Model to describe scheduling complexities

Thread Architecture Model (TAM)

- Encodes scheduling related info:
  - Request flows
  - Thread interactions
  - Resource consumption patterns
- Easy to obtain automatically
- From complicated systems to an understandable and analyzable model
  - HBase
  - Cassandra
  - MongoDB
  - Riak



TAM Exposes Scheduling Problems

- We discovered five categories of problems that happen in real systems
  - Lack of scheduling points
  - Unknown resource usage
  - Hidden contention between threads
  - Uncontrolled thread blocking
  - Ordering constraints upon requests

Fix Problems Leads to Effective Scheduling

- TAM-based simulation finds problem-free thread architectures
  - Provides schedulability: various desired scheduling policies can be realized
  - HBase ➡ Tamed-HBase
- Implementation transforms system to be schedulable
  - Muzzled-HBase: approximated implementation
  - Effective scheduling under YCSB and other workloads

Thread Architecture Model

enables

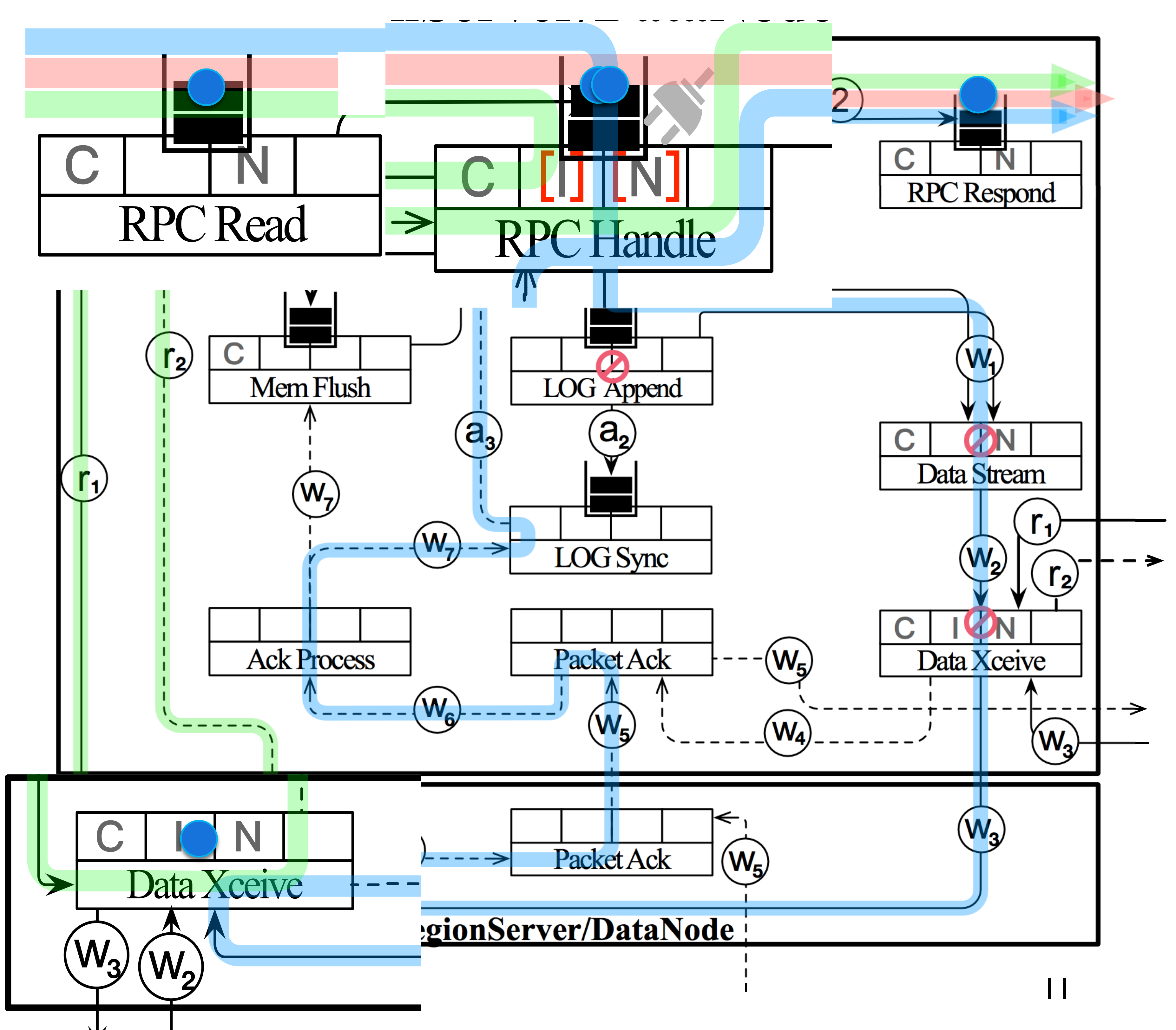
principled schedulability analysis

on general distributed storage systems

Outline

- Overview
- Thread Architecture Model
- Scheduling Problems
- Achieve Schedulability: A Case Study
- Conclusion

Thread Architecture Model



Name

C — CPU
I — I/O
N — network
L — Lock

stage (threads performing similar tasks)

Name

resource usage

request flow

request queue (scheduling point)

blocking

Thread Architecture Model

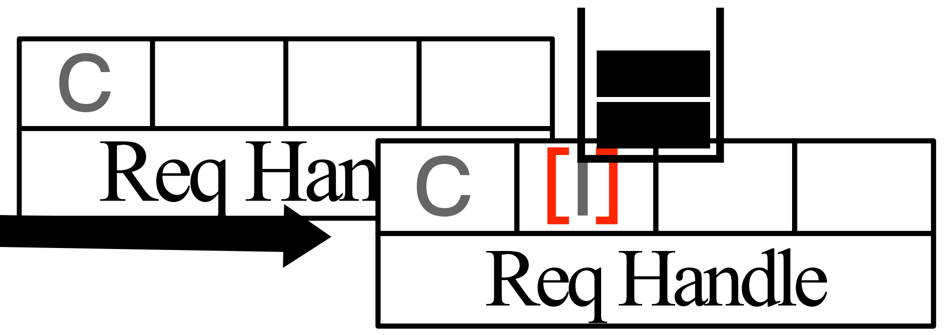
- TAM encodes scheduling related info:
  - Request flows
  - Thread interactions
  - Resource consumption patterns
- From complex systems to analyzable models
- TADalyzer: from live system to TAM automatically
  - Only 20-50 lines of user annotation code required

Outline

- Overview
- Thread Architecture Model
- Scheduling Problems
- Achieve Schedulability: A Case Study
- Conclusion

TAM Exposes Scheduling Problems

- No scheduling
- Unknown resource usage
- Hidden contention
- Blocking
- Ordering constraint



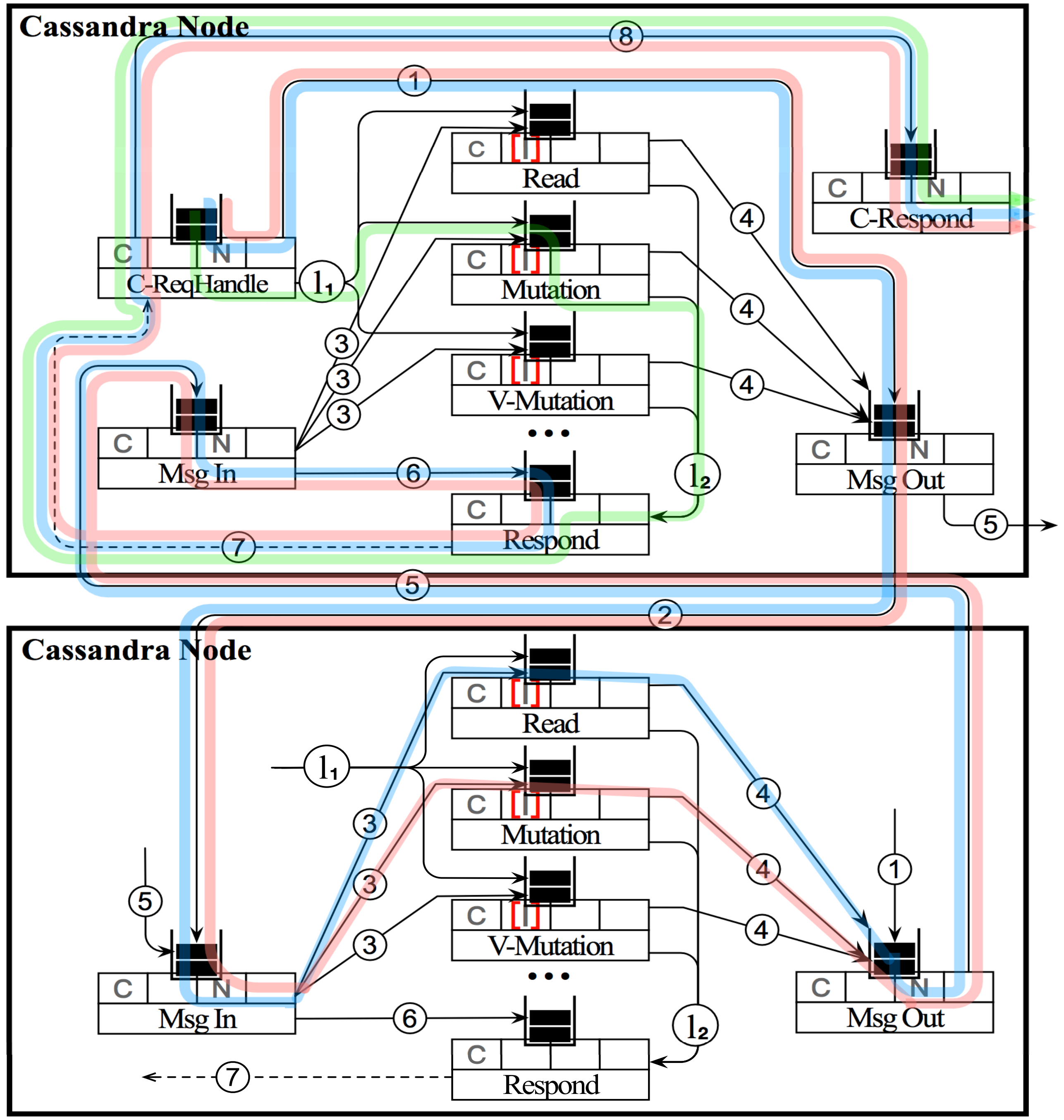
- Common in distributed storage systems
  - HBase, Cassandra, MongoDB, Riak…
- Directly identifiable from TAM
  - No low-level implementation details required

TAM Exposes Scheduling Problems

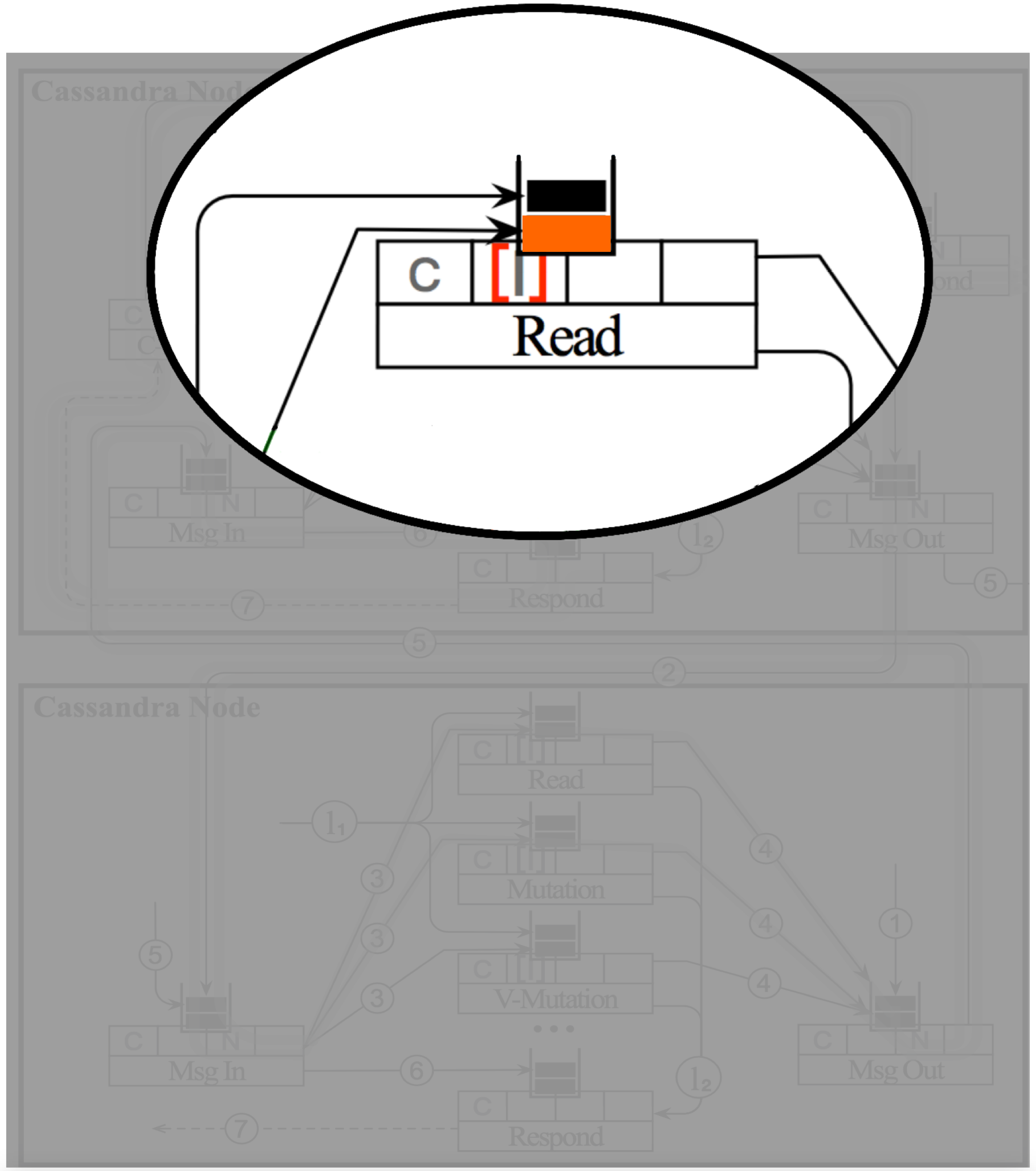
- No scheduling
- Unknown resource usage
- Hidden contention
- Blocking
- Ordering constraint

- Common in distributed storage systems
  - HBase, Cassandra, MongoDB, Riak…
- Directly identifiable from TAM
  - No low-level implementation details required

Scheduling Problem: Unknown Resource Usage



Scheduling Problem: Unknown Resource Usage



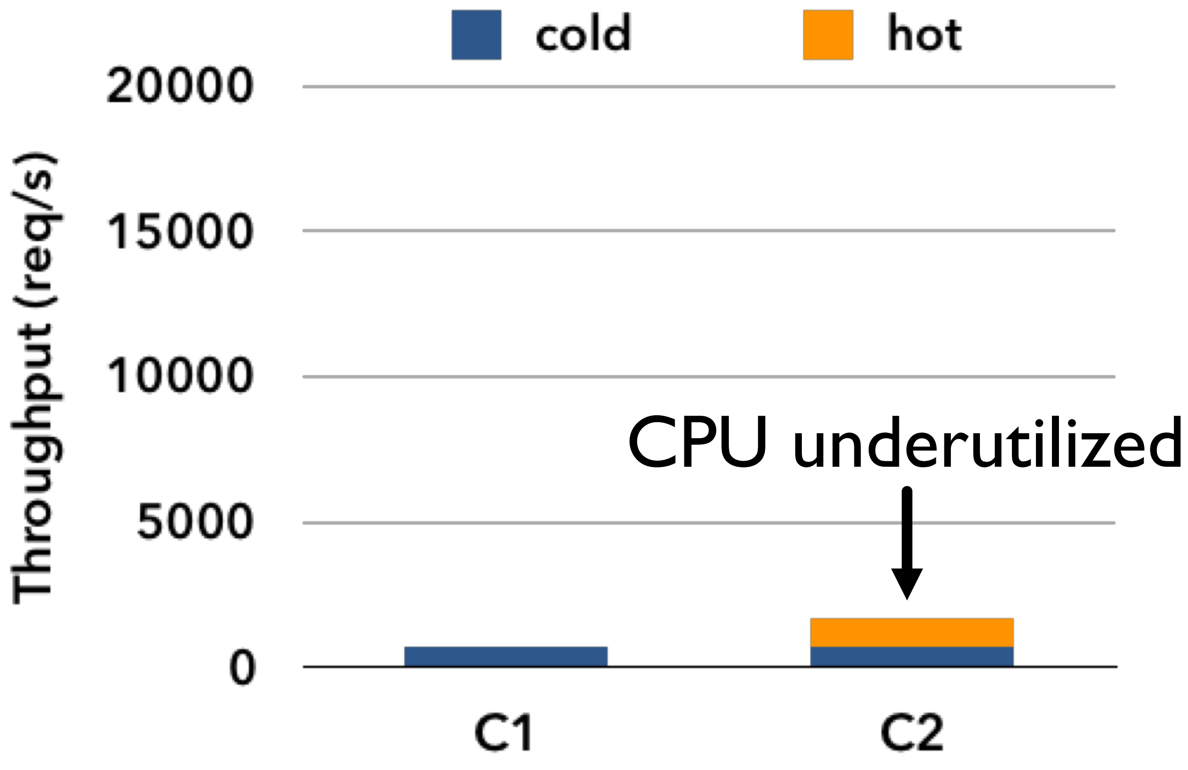
Workload:

C1: issues cold requests

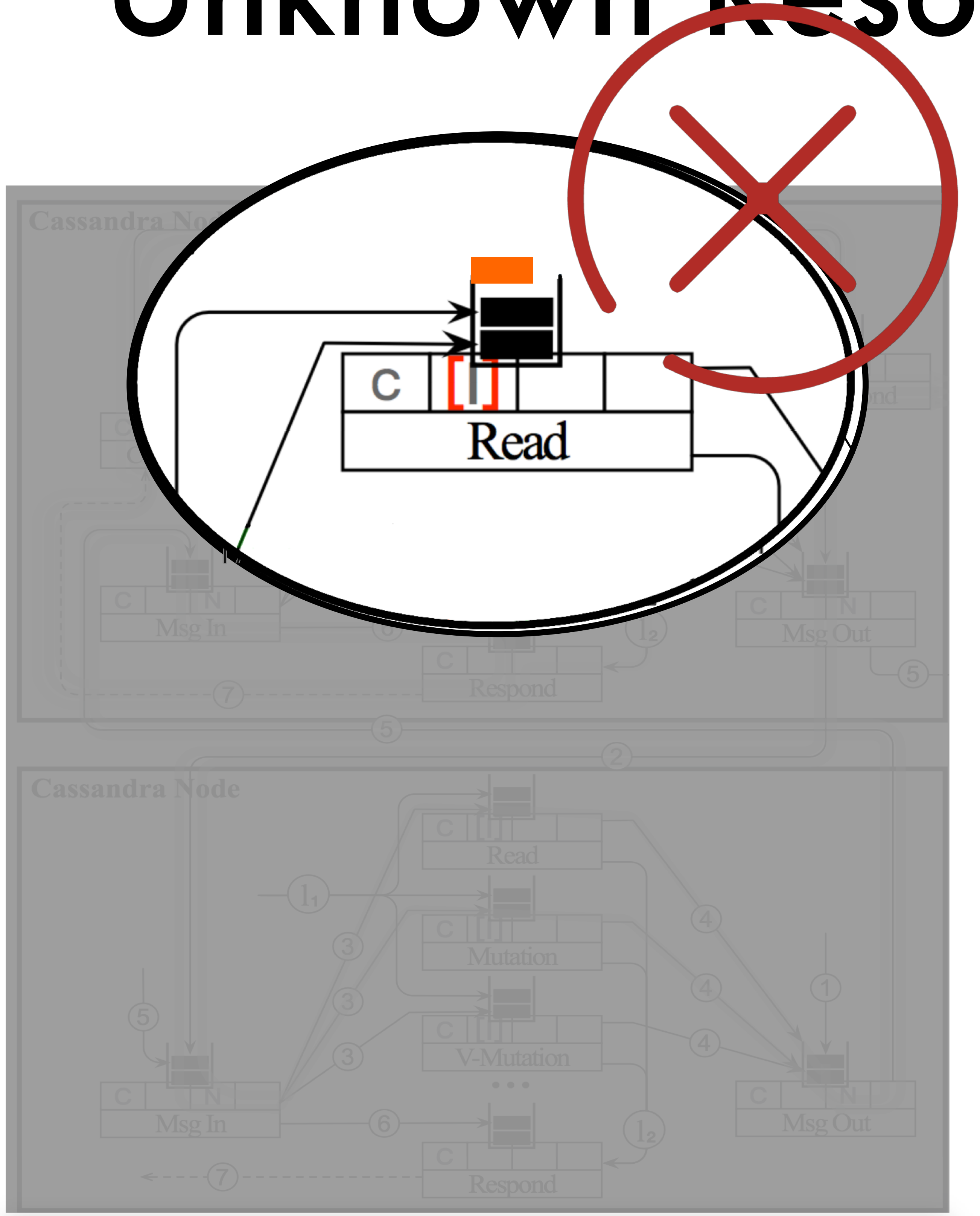
C2: issues cold *and* cached requests

Expectation:

C2 has much higher throughput (due to cached request)



Unknown Resource Usage: Solution



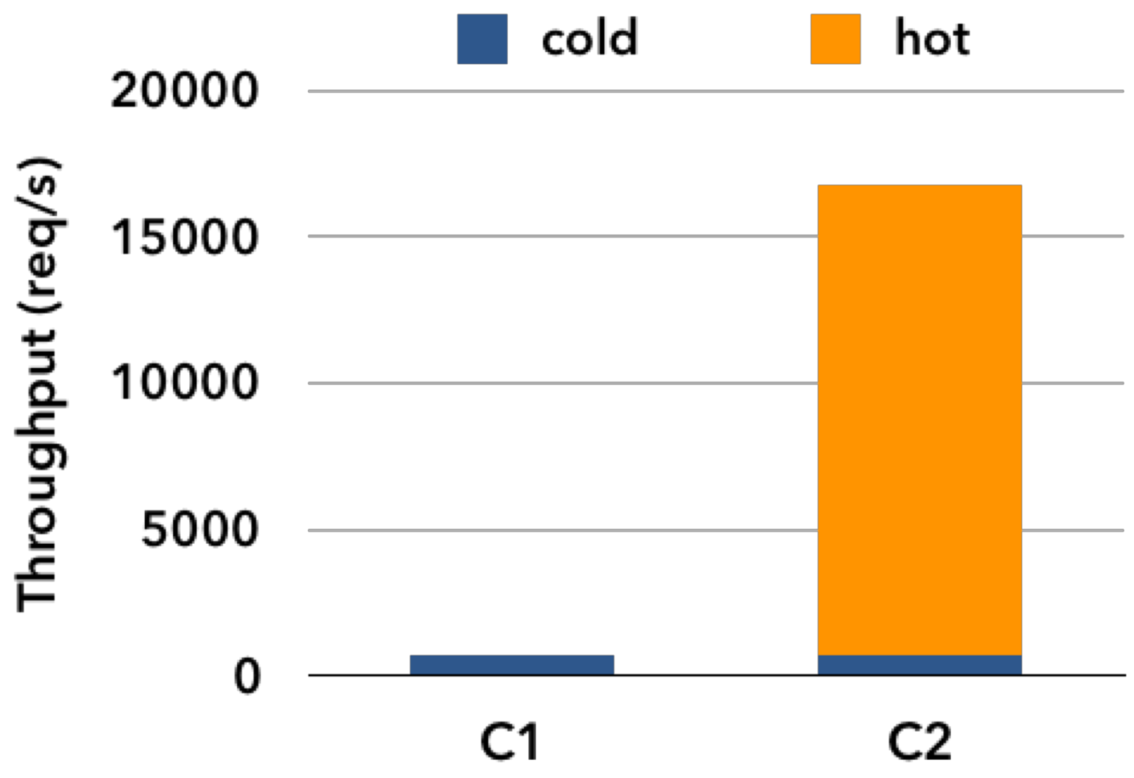
Workload:

C1: issues cold requests

C2: issues cold and cached requests

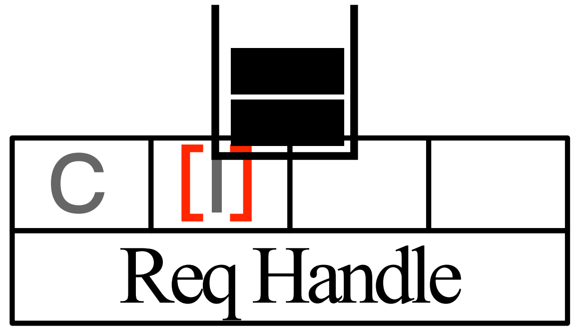
Expectation:

C2 has much higher throughput (due to cached request)

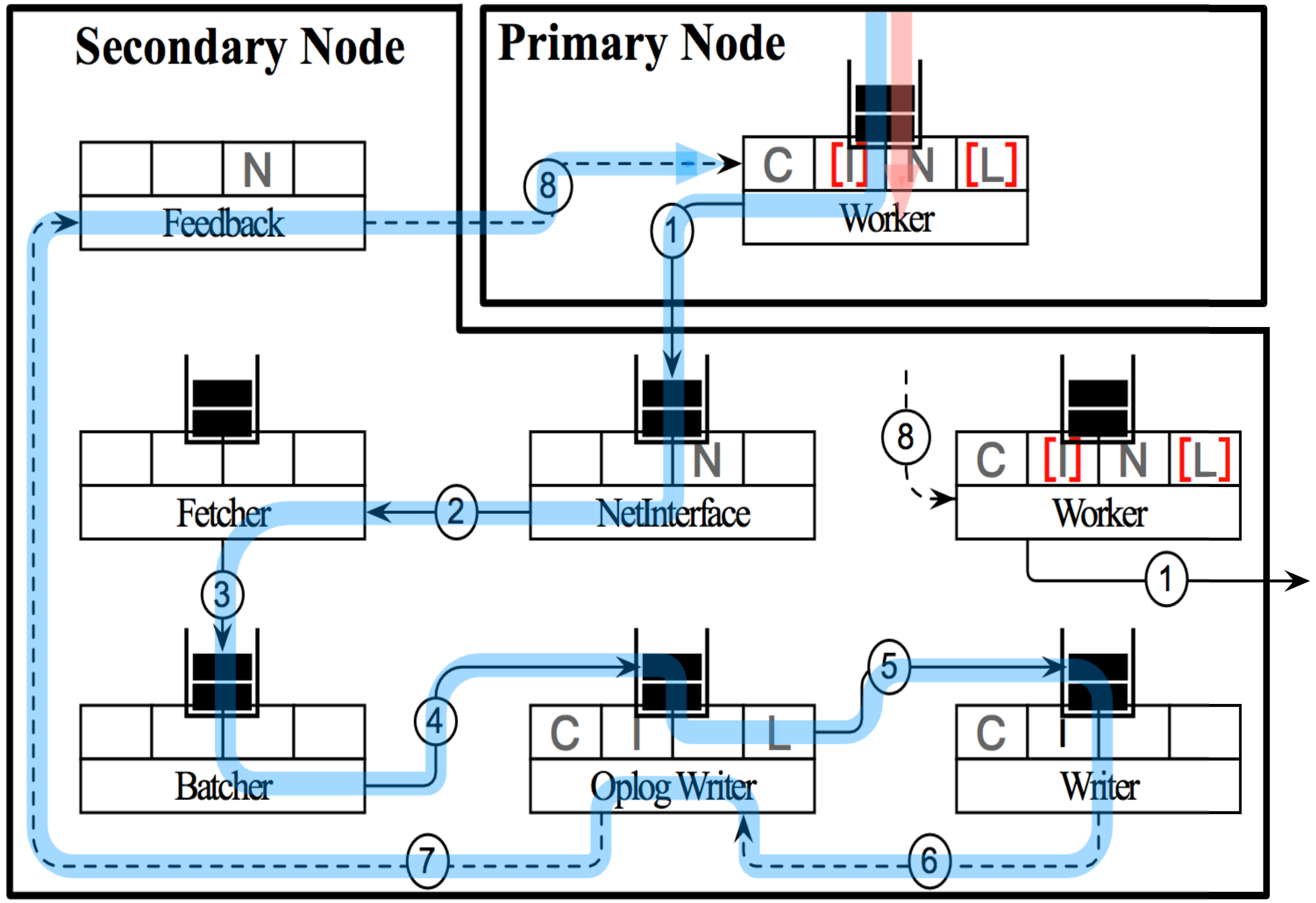


Scheduling Problem: Unknown Resource Usage

- Resource usage patterns unknown to schedulers until after the processing begins
- Forces schedulers to make decisions before information is available
- Identified as red square brackets around resource symbols in TAM

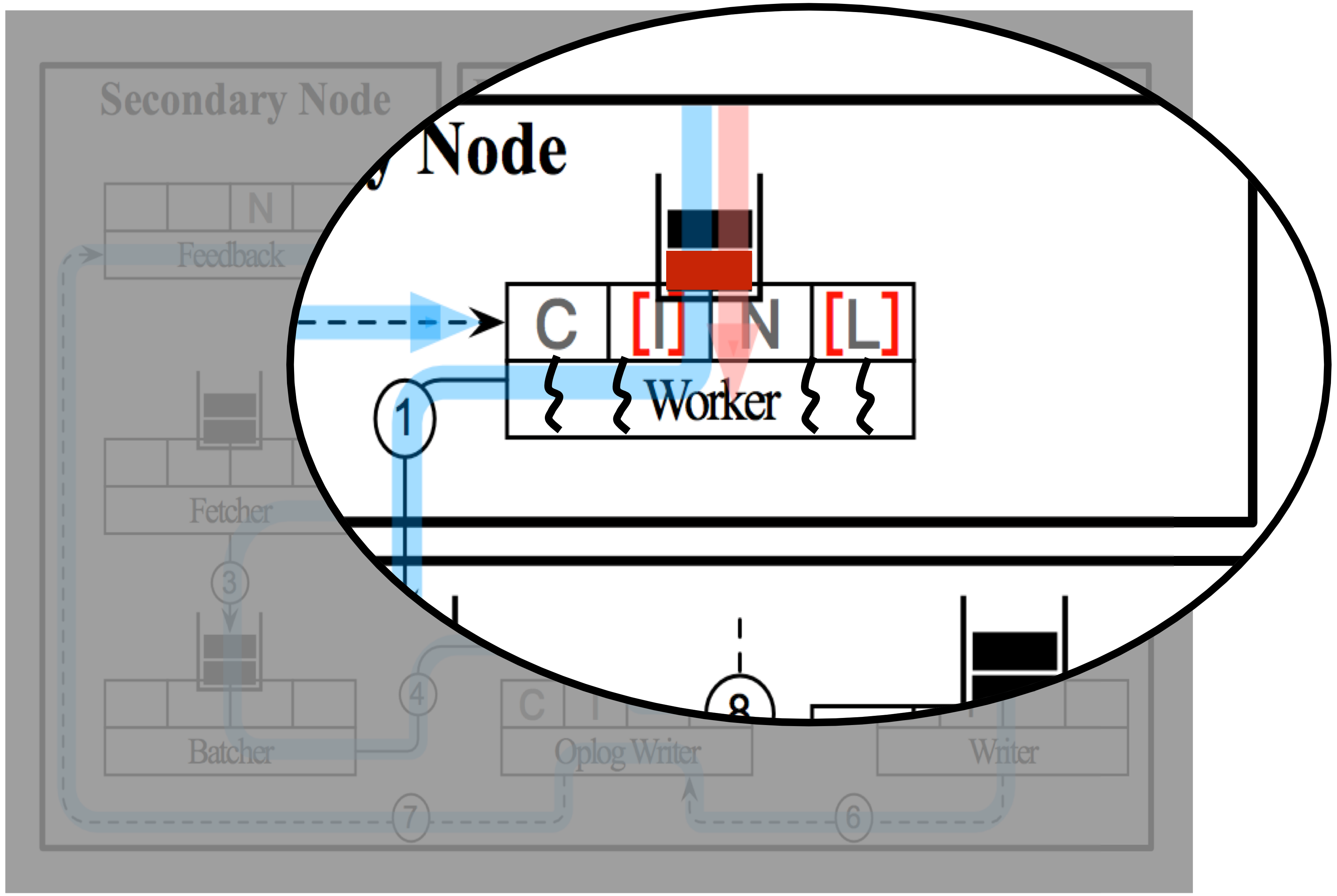


Scheduling Problem: Blocking



MongoDB

Scheduling Problem: Blocking



MongoDB

Workload:

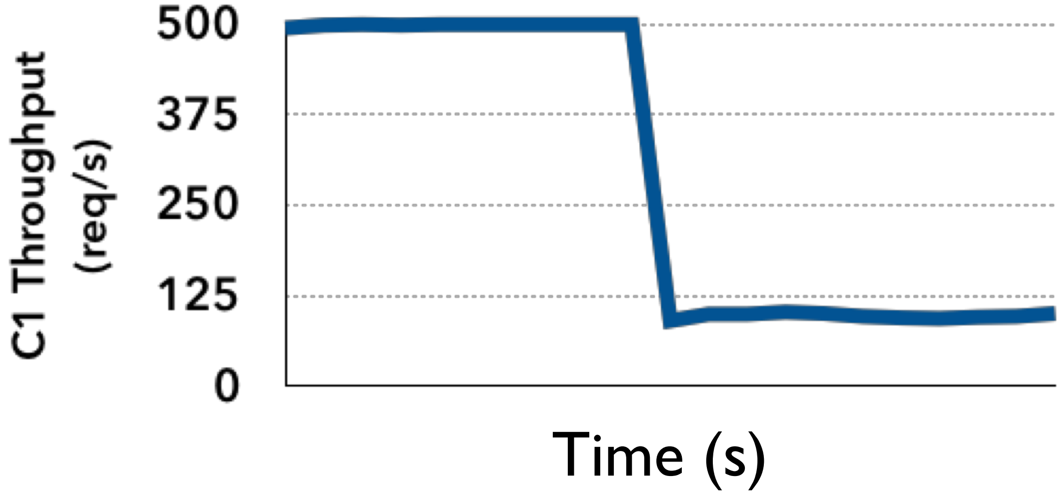
C1: reads from primary (does not go to secondary)

C2: writes to primary (replicate to secondary node)

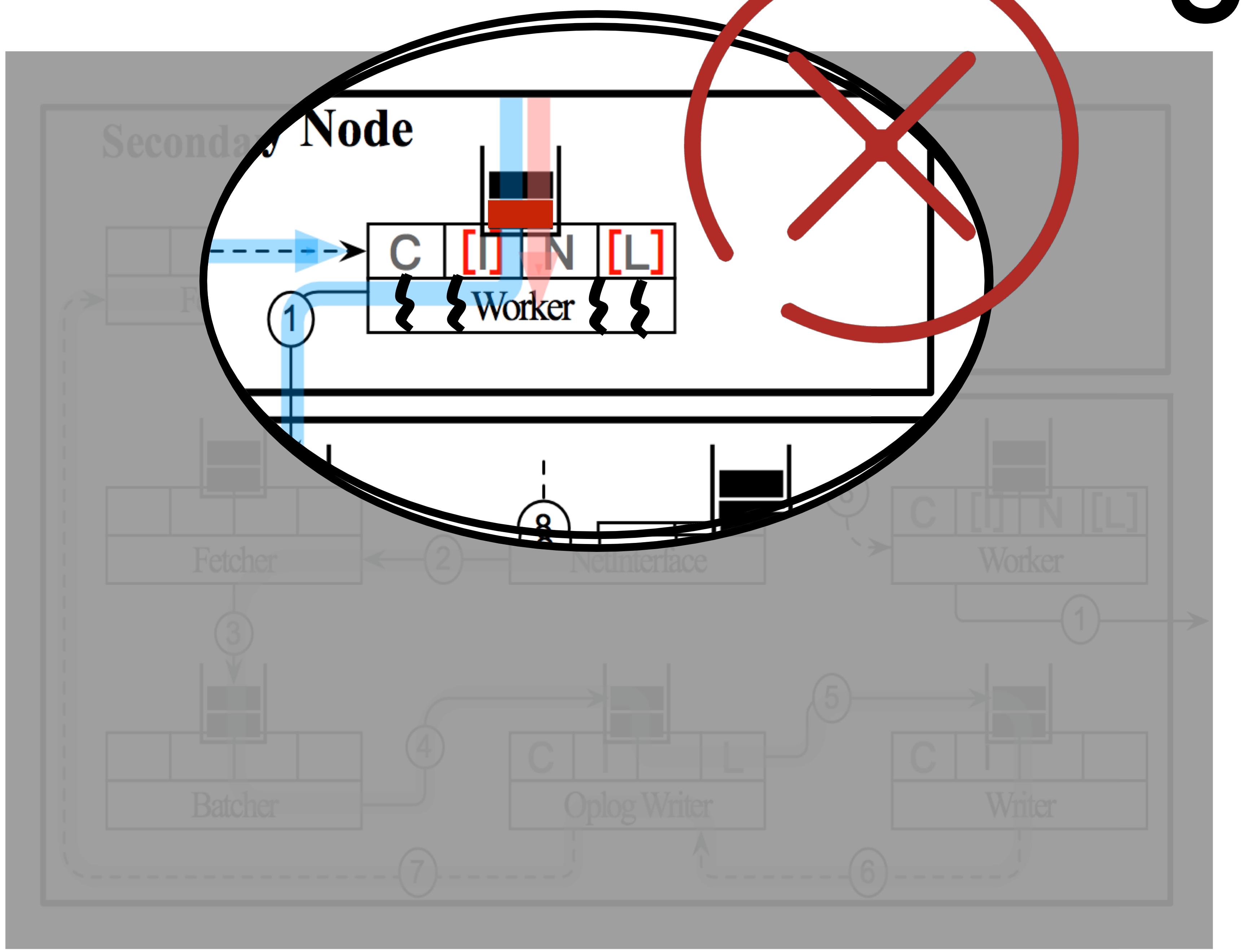
time 10: the secondary node slows down

Expectation:

C1 reads throughput remains stable



Blocking: Solution



MongoDB

Workload:

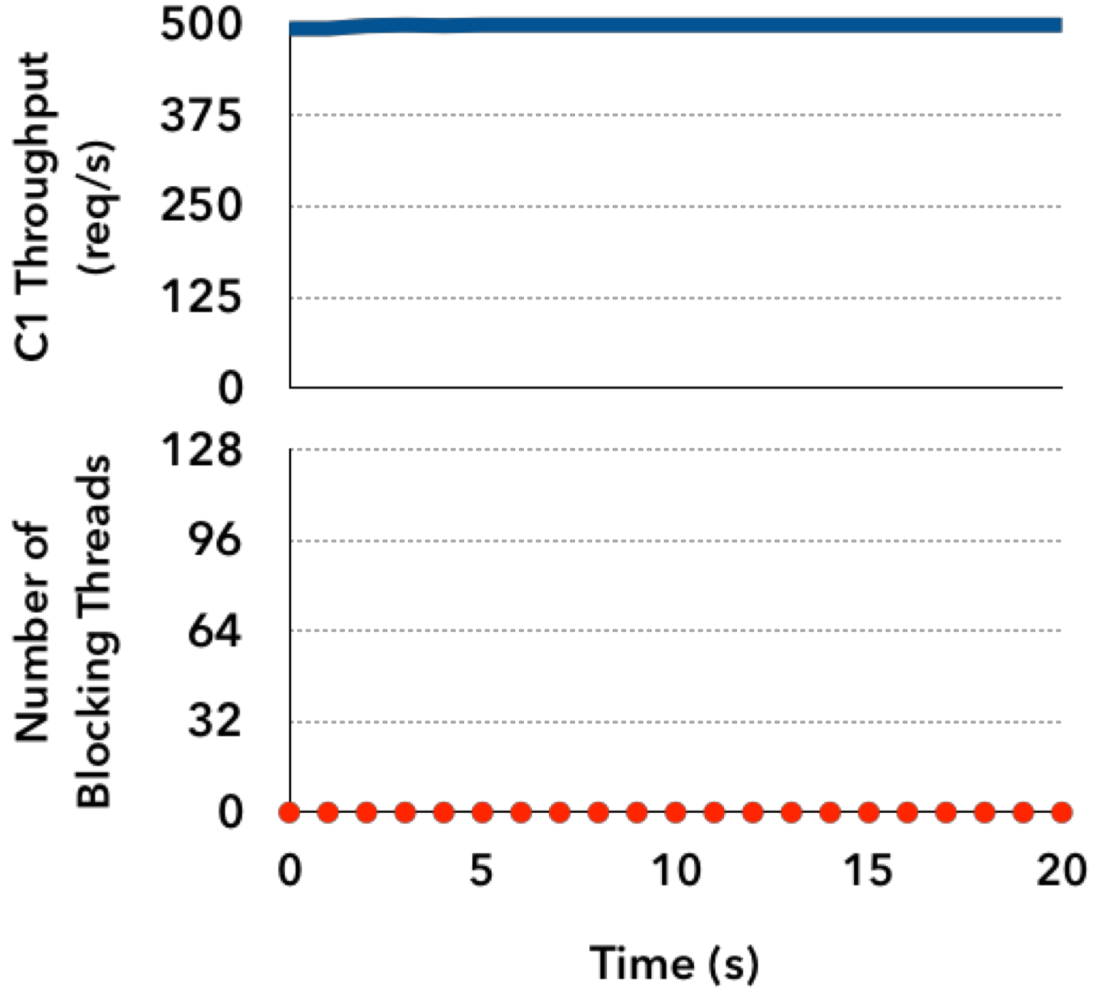
C1: reads

C2: writes (replicate to secondary node)

time 10: the secondary node slows down

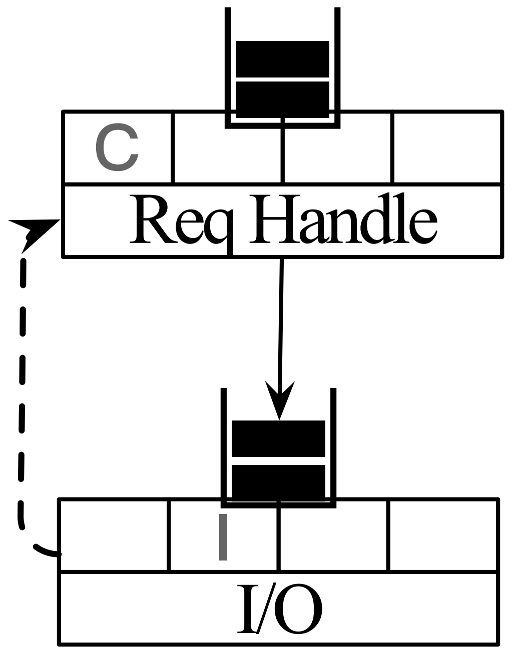
Expectation:

C1 reads throughput remains stable



Scheduling Problem: Blocking

- Stages with fixed number of threads block on other stages
- Unable to schedule requests that could have been completed because all threads block
- Identified as dashed arrow point to stages with queues in TAM



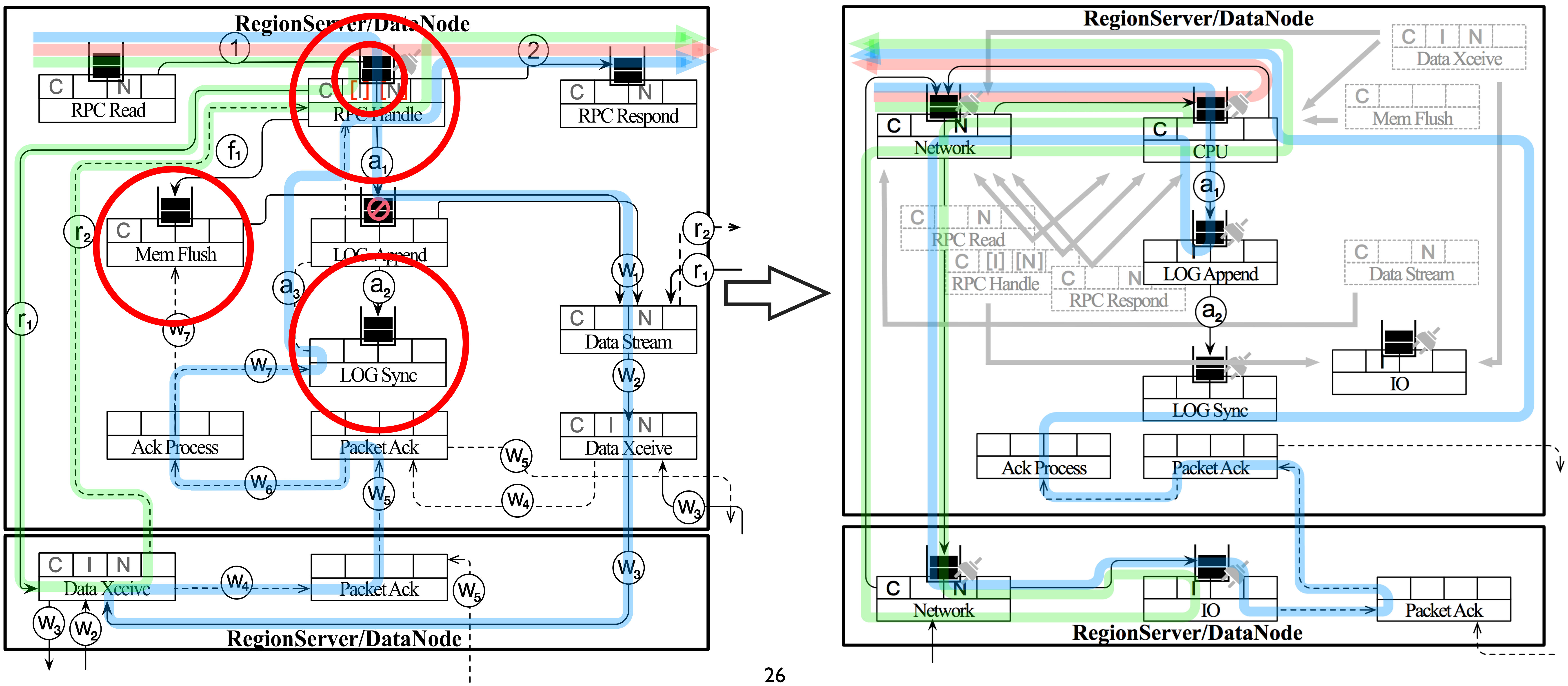
Outline

- Overview
- Thread Architecture Model
- Scheduling Problems
- Achieve Schedulability: A Case Study
- Conclusion

Fixing Problems Leads to Schedulability

- TAM-based simulation framework: explore thread architectures
  - Simulates how systems perform under workloads
  - Easily study architecture designs and scheduling policies
- Implementation: realize schedulable systems
  - Also validates that simulation matches the real world

Simulation: HBase to Tamed-HBase



Implementation : Tamed-HBase to Muzzled-HBase

- Some approximations to make implementation easier
- Supports multiple scheduling policies
- Proper scheduling under various workloads

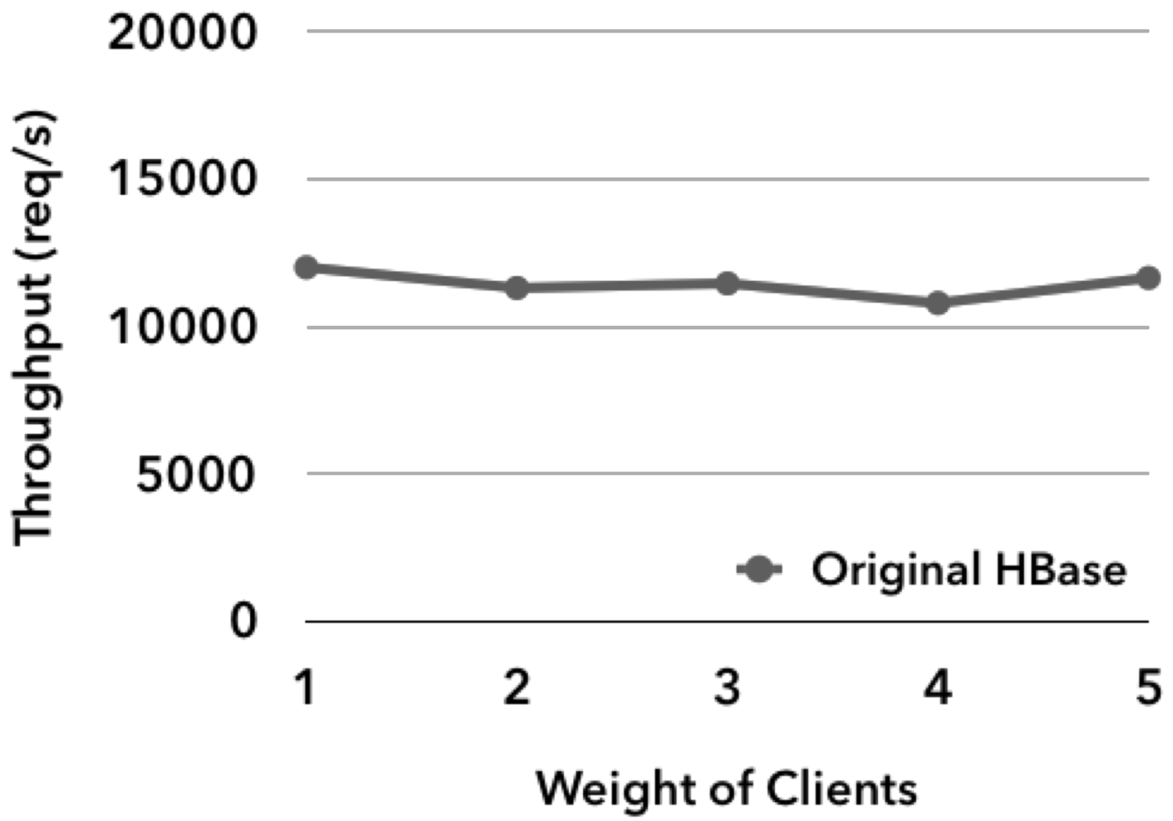
Muzzled-HBase: Weighted Fairness

Workloads:

Five clients, each with different weight , run YCSB (reads mostly)

Expectation:

Client receives throughput proportional to weight



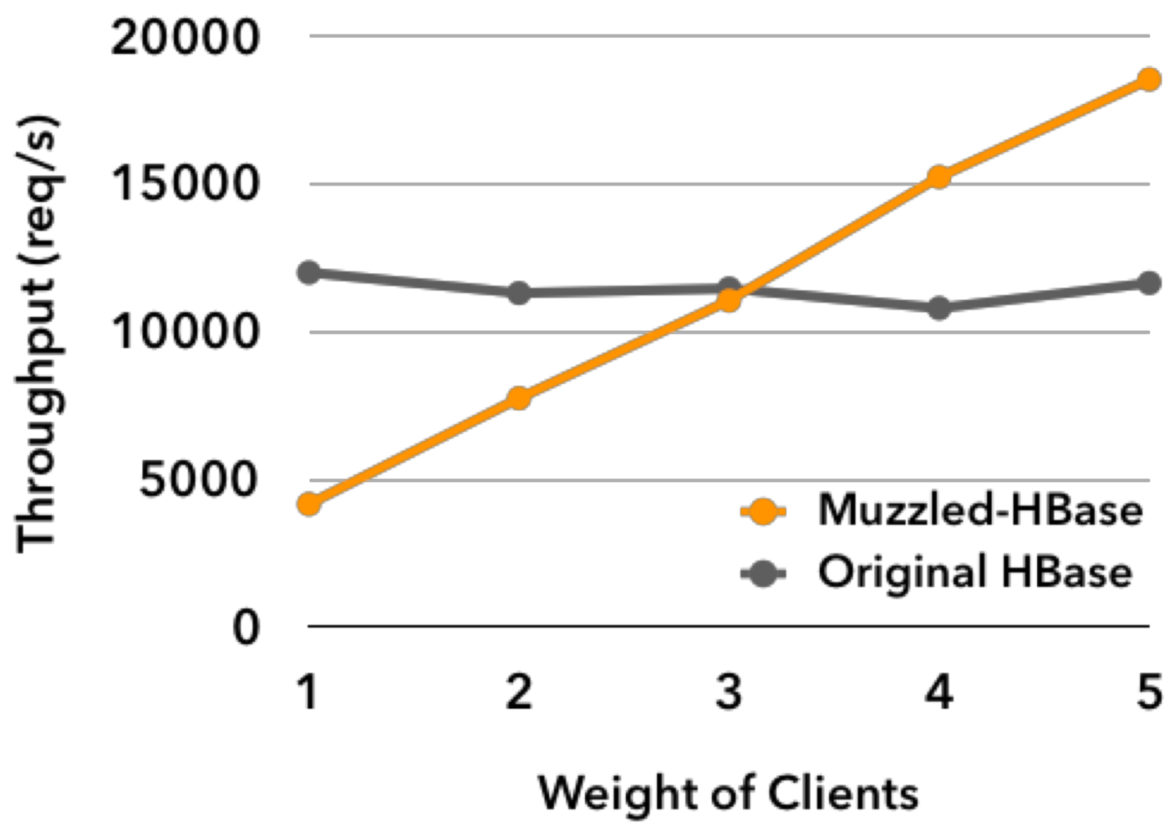
Muzzled-HBase: Weighted Fairness

<u>Workloads</u>:

Five clients, each with different weight , run YCSB (reads mostly)

<u>Expectation:</u>

Client receives throughput proportional to weight



Muzzled-HBase: Tail Latency Guarantee

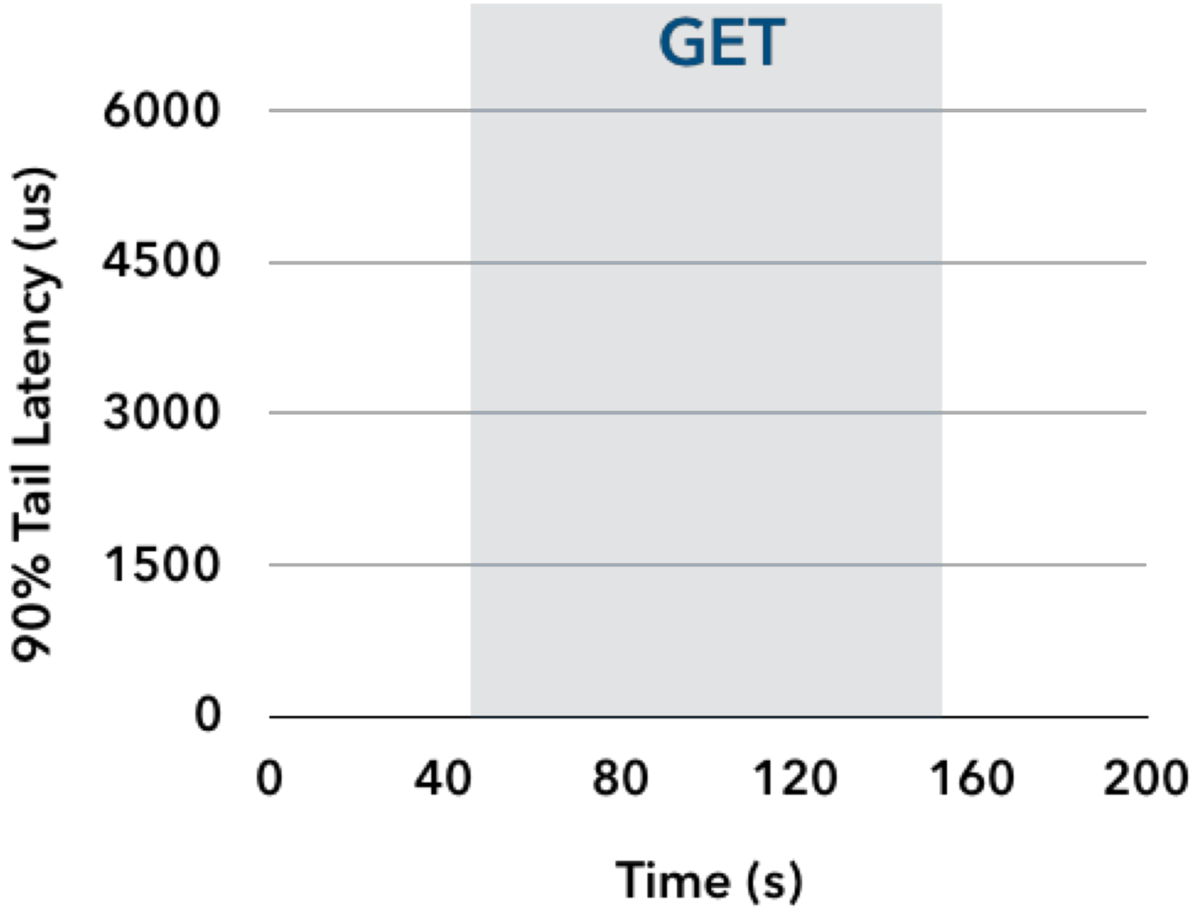
Workloads:

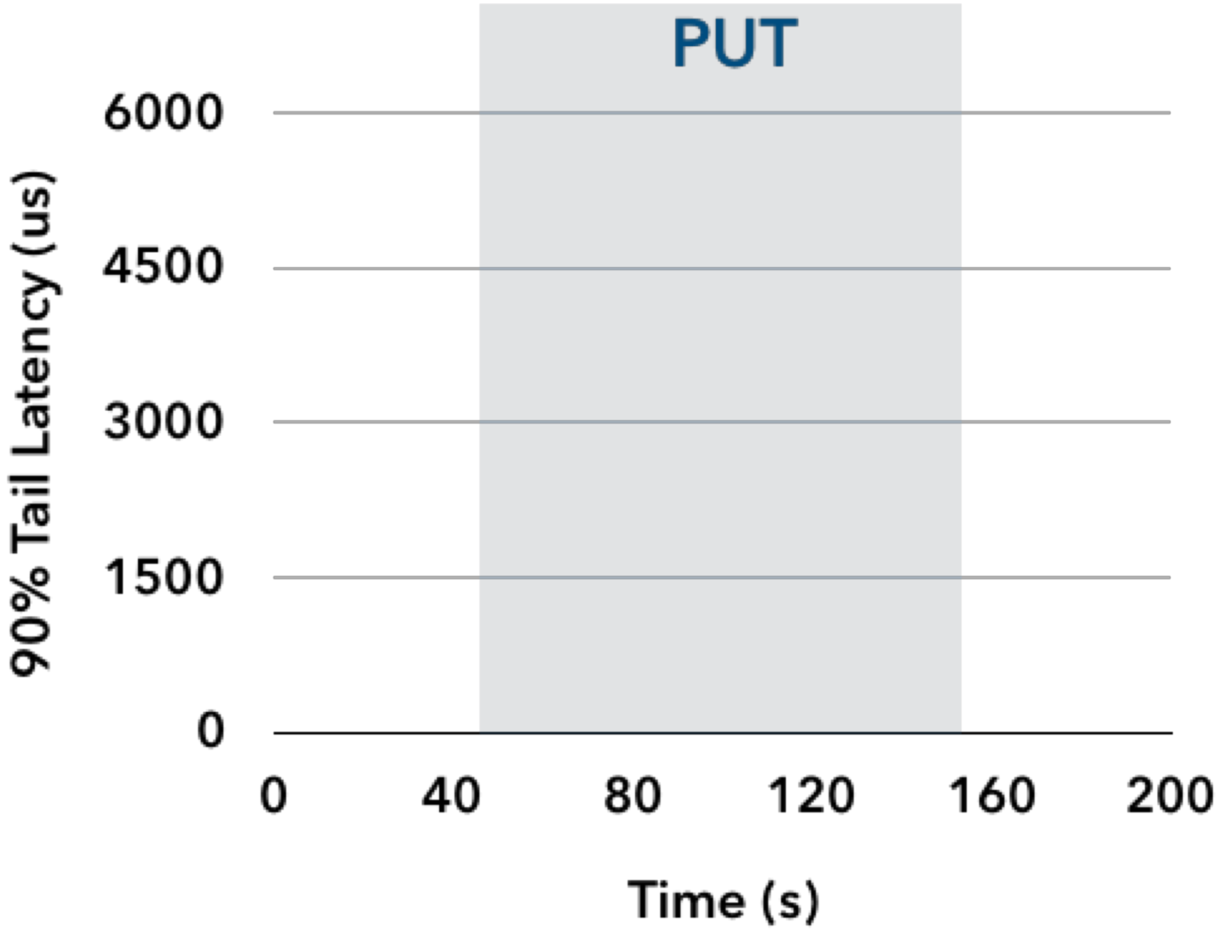
Foreground client: runs YCSB (update-heavy)

Background client: random Gets or Puts

Expectation:

Foreground latency remains stable





Muzzled-HBase: Tail Latency Guarantee

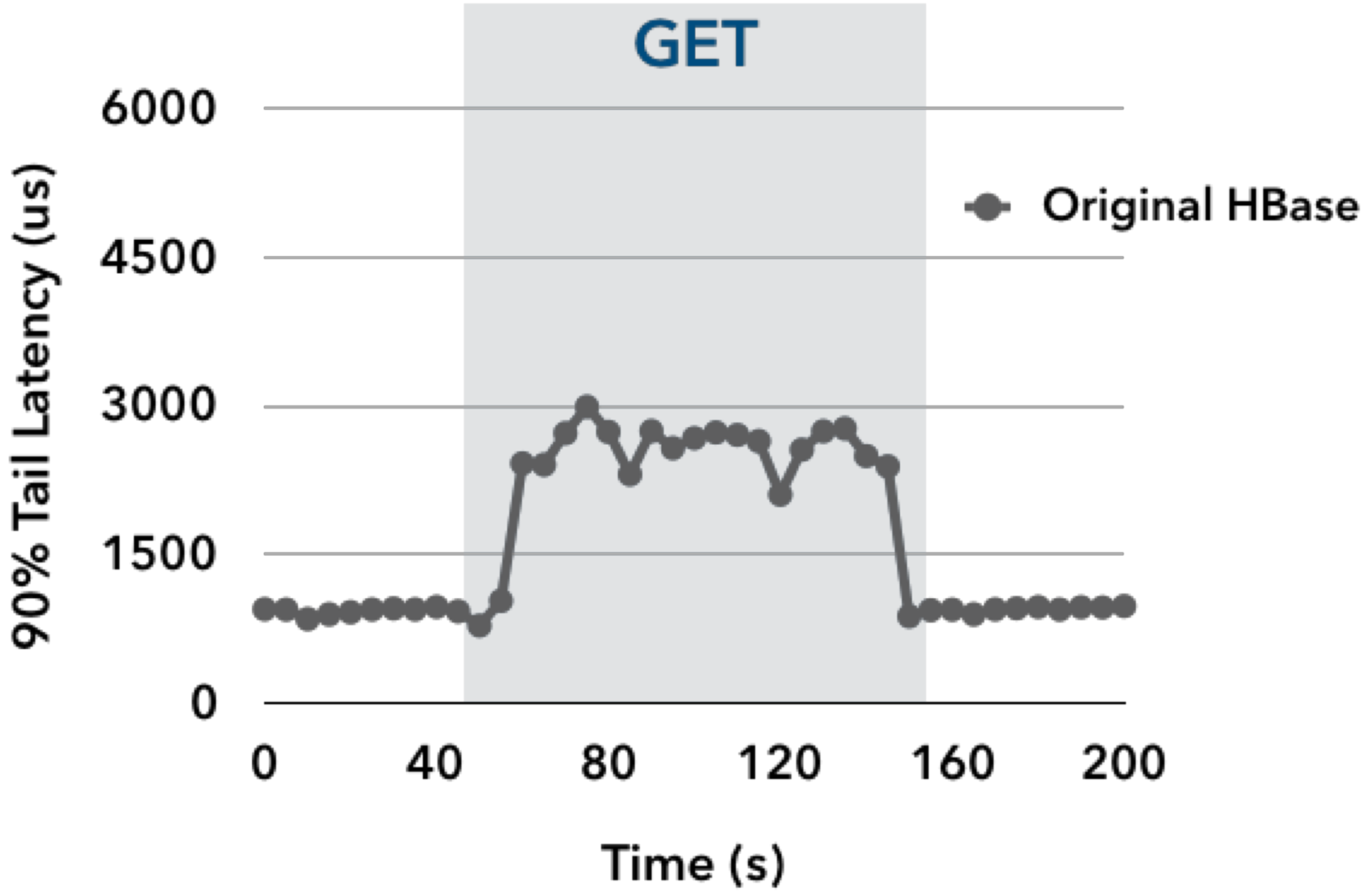
Workloads:

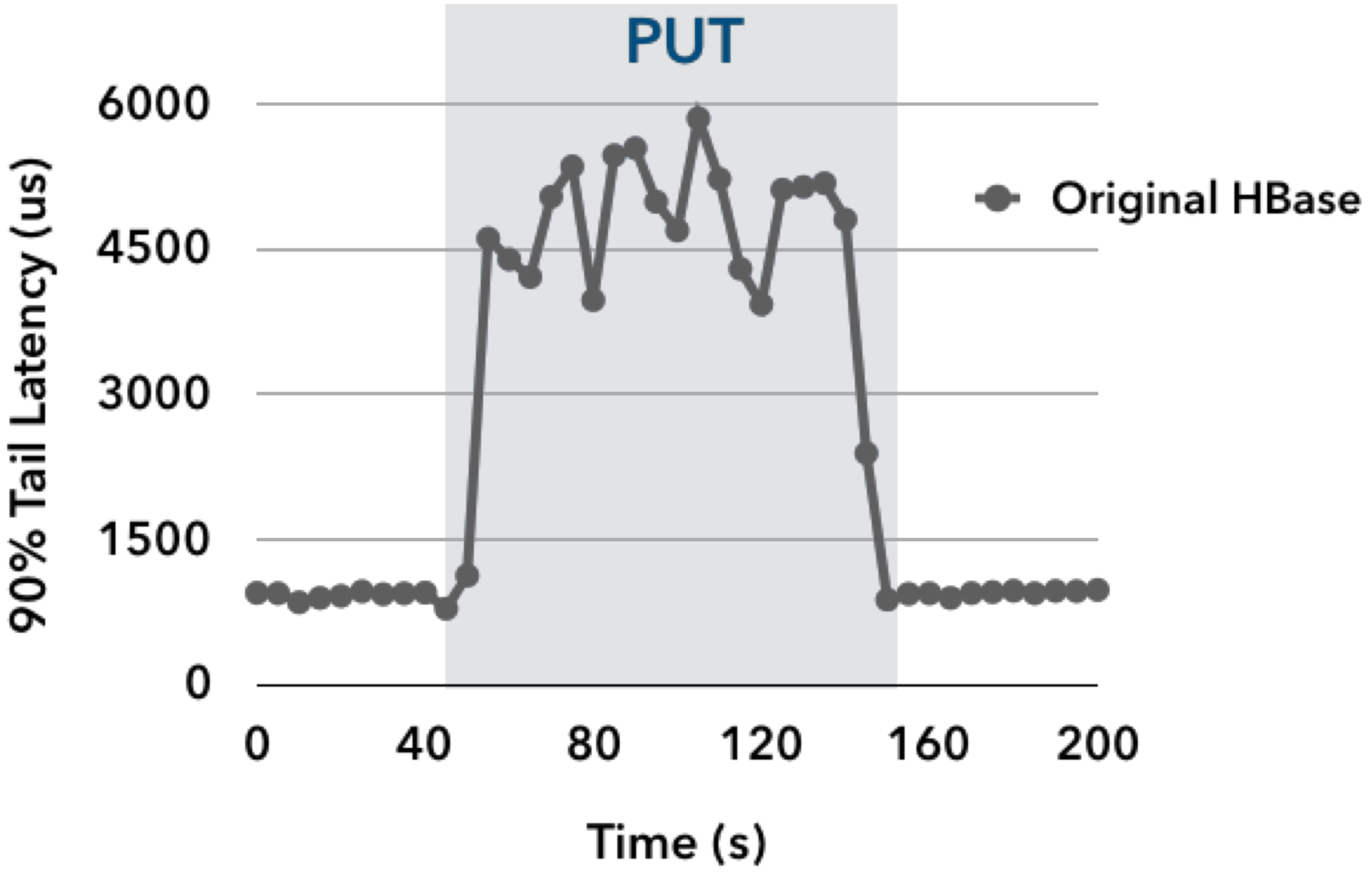
Foreground client: runs YCSB

Background client: random Gets or Puts

Expectation:

Foreground latency remains stable





Muzzled-HBase:
Tail Latency Guarantee

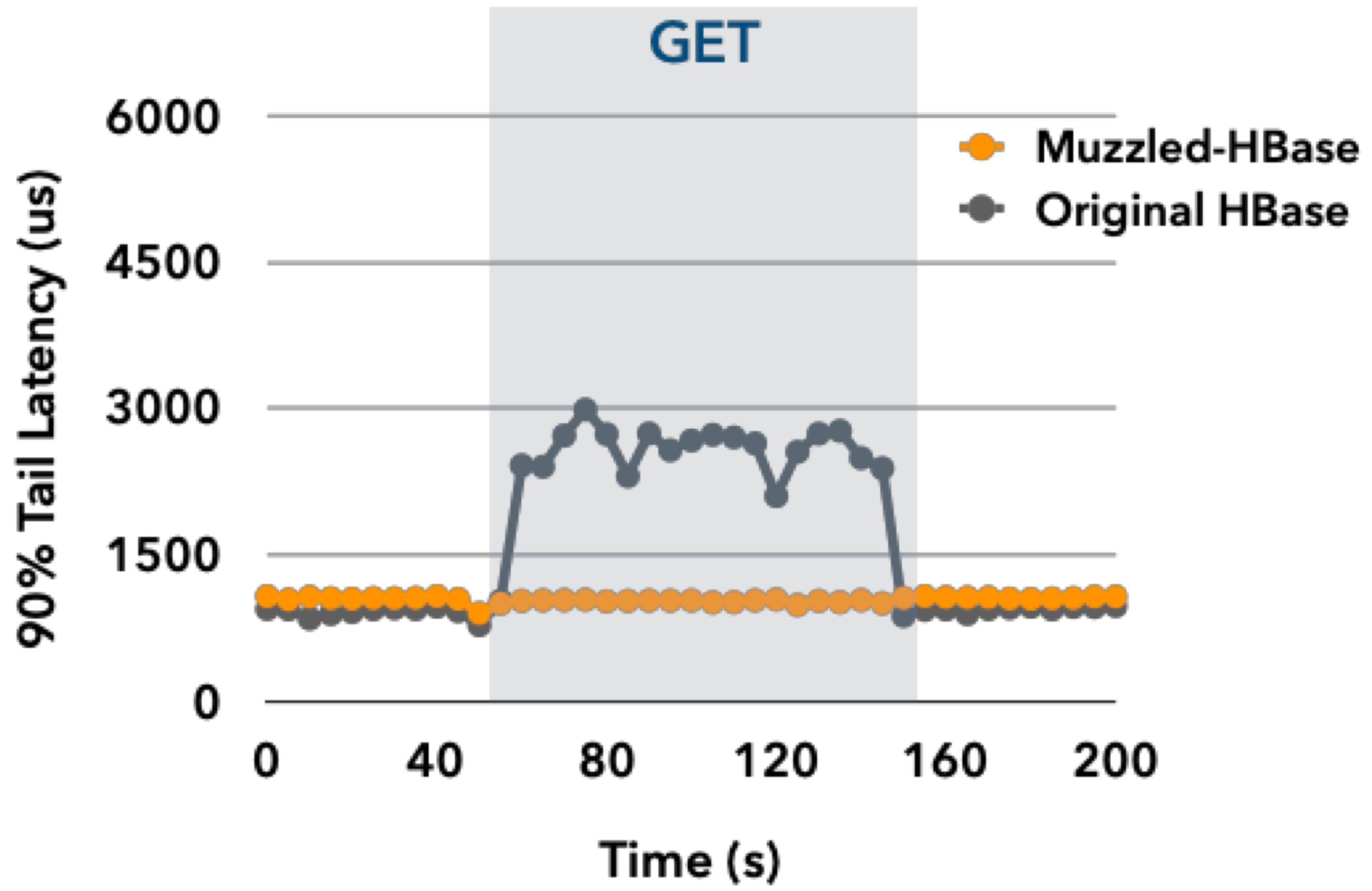
Workloads:

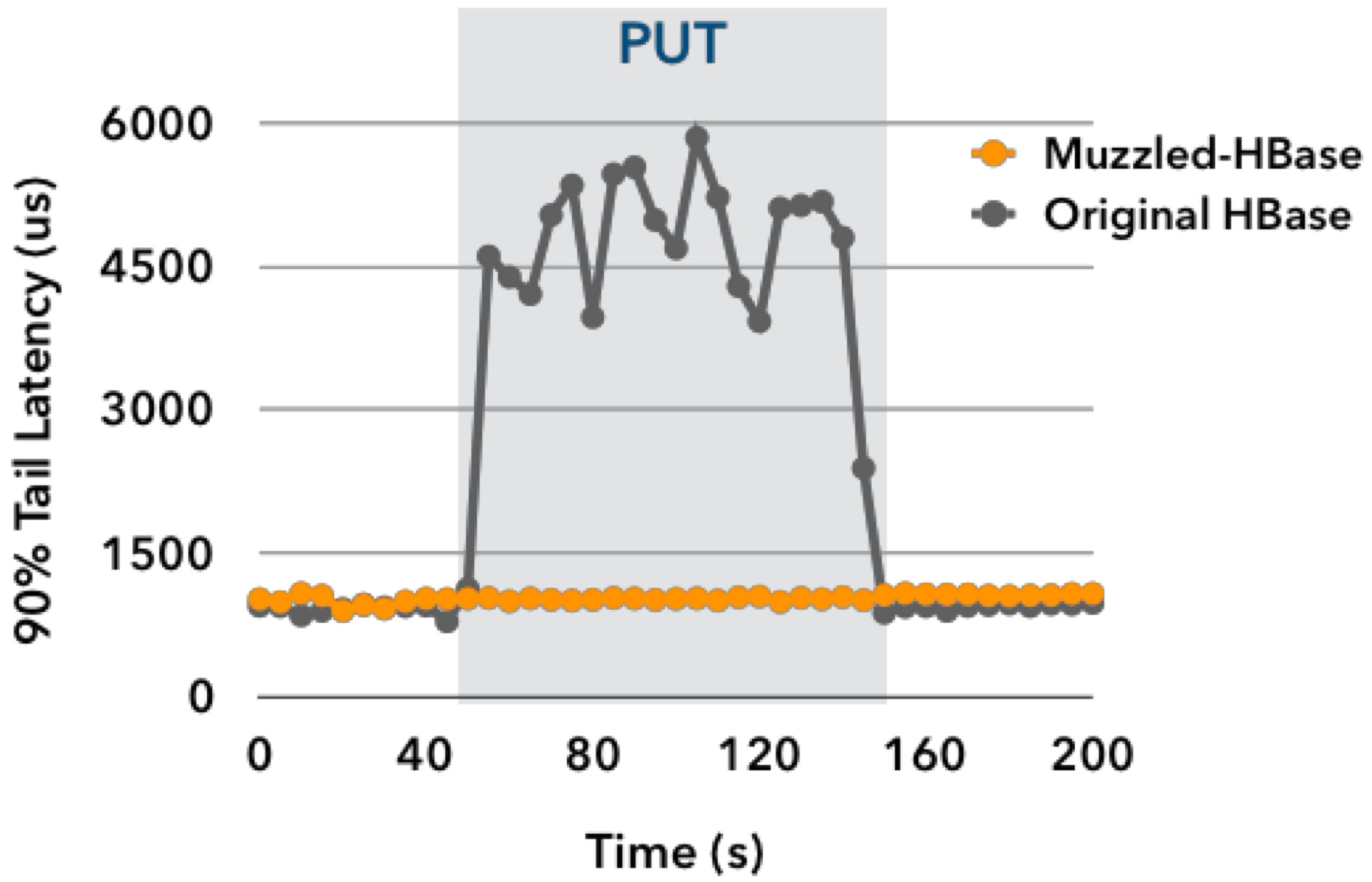
Foreground client: runs YCSB

Background client: random Gets or Puts

Expectation:

Foreground latency remains stable





Outline

- Overview
- Thread Architecture Model
- Scheduling Problems
- Achieve Schedulability: A Case Study
- Conclusion

Conclusion

- We introduce thread architecture models
  - Reduce complex distributed scheduling to an understandable representation
  - Enable schedulability analysis
- We discover five scheduling problems
  - Point to problematic architecture that exist in real systems
  - Fixing them enables effective scheduling
- Complex systems need to be built with the help of TAM
  - Analyze existing system and enable schedulability
  - Design systems that are problem-free and natively schedulable

Thank you! Questions?

(poster number: 28)

OceanBase: We are Hiring

Geo-scale relational database behind Alipay

42,000,000 SQLs per second

US and China based

Contact OceanBase-Public@list.alibaba-inc.com

OceanBase微信
公众号