original: April 17th, 2013
this version: Oct 2013

Omega: flexible, scalable schedulers for large compute clusters

Malte Schwarzkopf (University of Cambridge Computer Lab)

Andy Konwinski (UC Berkeley)

Michael Abd-El-Malek (Google)

John Wilkes (Google)

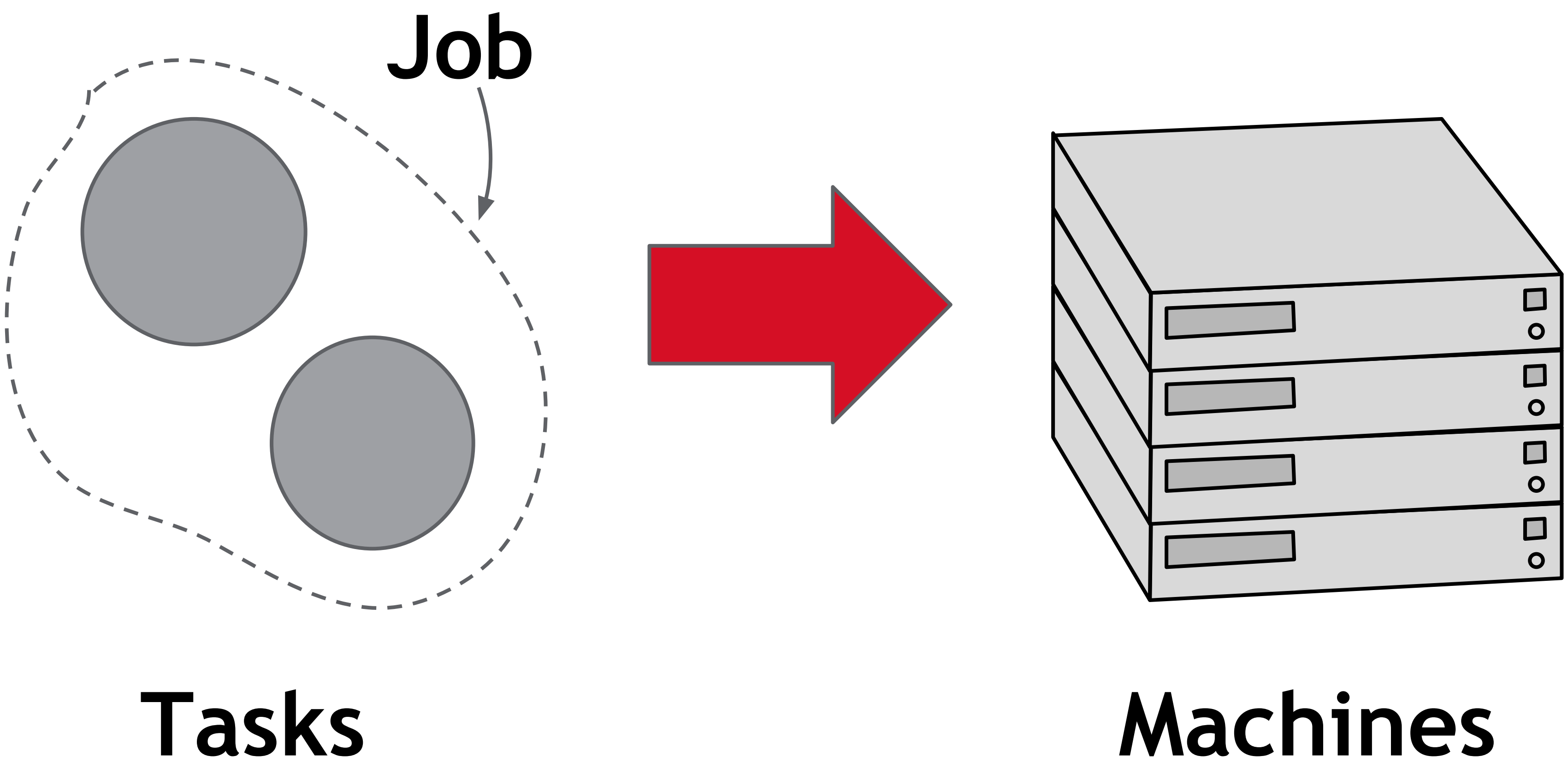
We own and operate data centers around the world

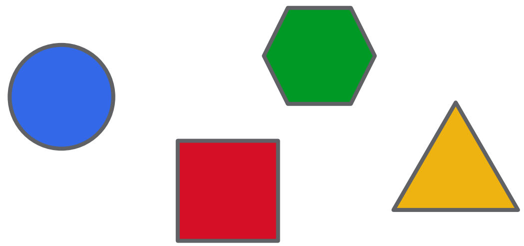
http://www.google.com/about/datacenters/inside/locations/


Google
google.com/datacenters

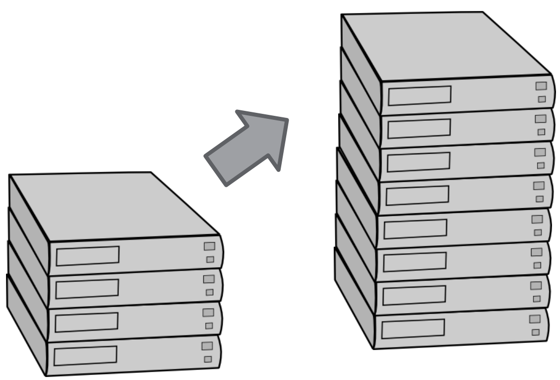

Google | google.com/datacenters


Google
google.com/datacenters

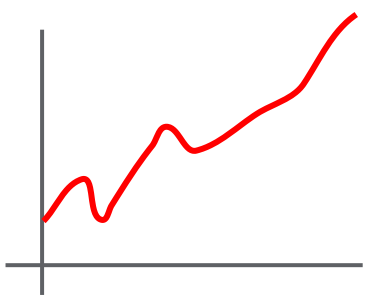

Job
Tasks
Machines



Diverse workloads

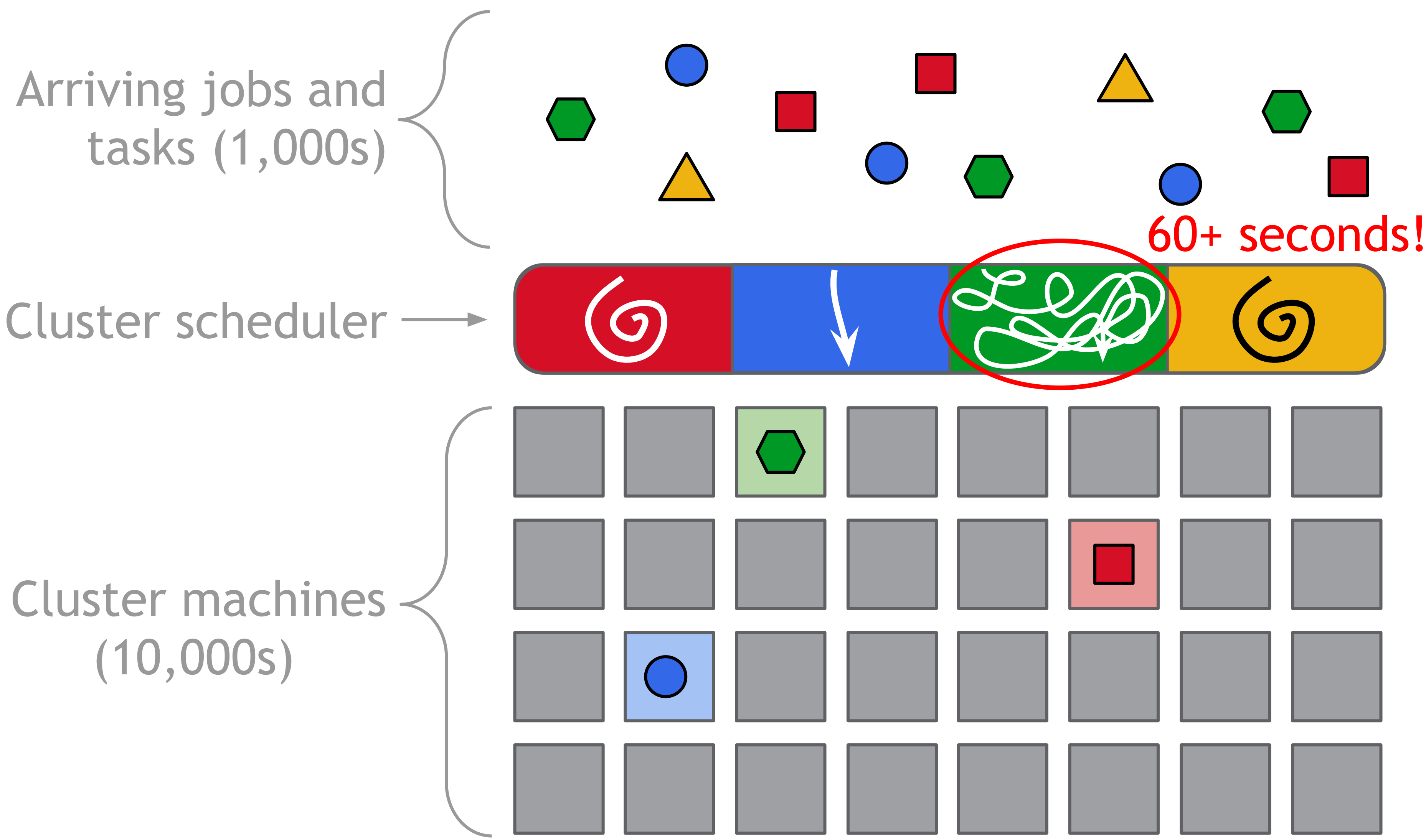


Increasing cluster sizes



Growing job arrival rates

why is this a problem?



why is this a problem?

Arriving jobs and tasks (1,000s)

Cluster scheduler

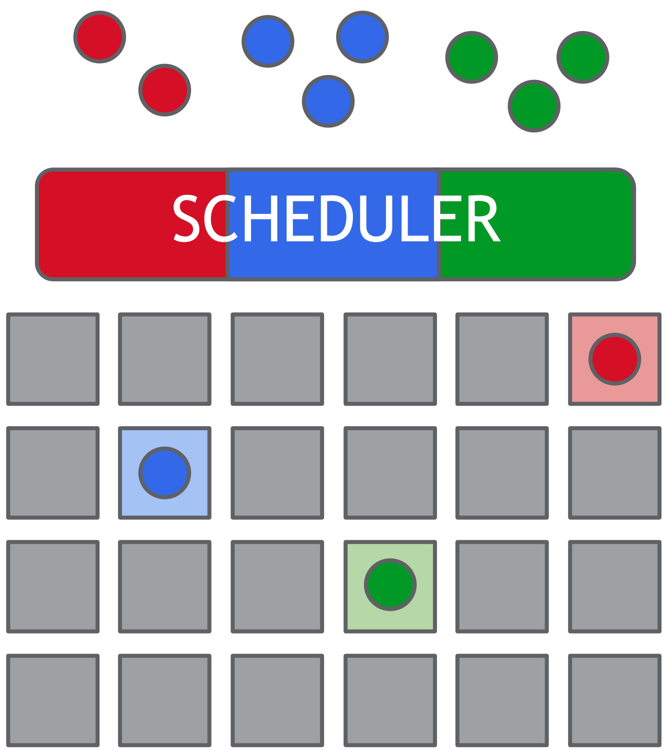
Cluste[r]
(10,000s)

Hence:
Break up into independent schedulers.

But: How do we arbitrate resources between schedulers?

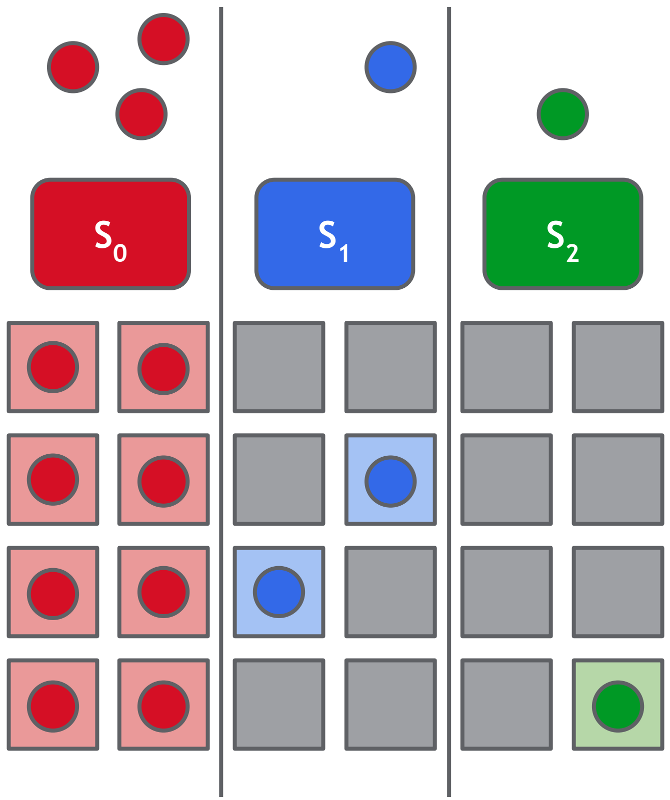
existing approaches

monolithic scheduler



- hard to diversify
- code growth
- scalability bottleneck

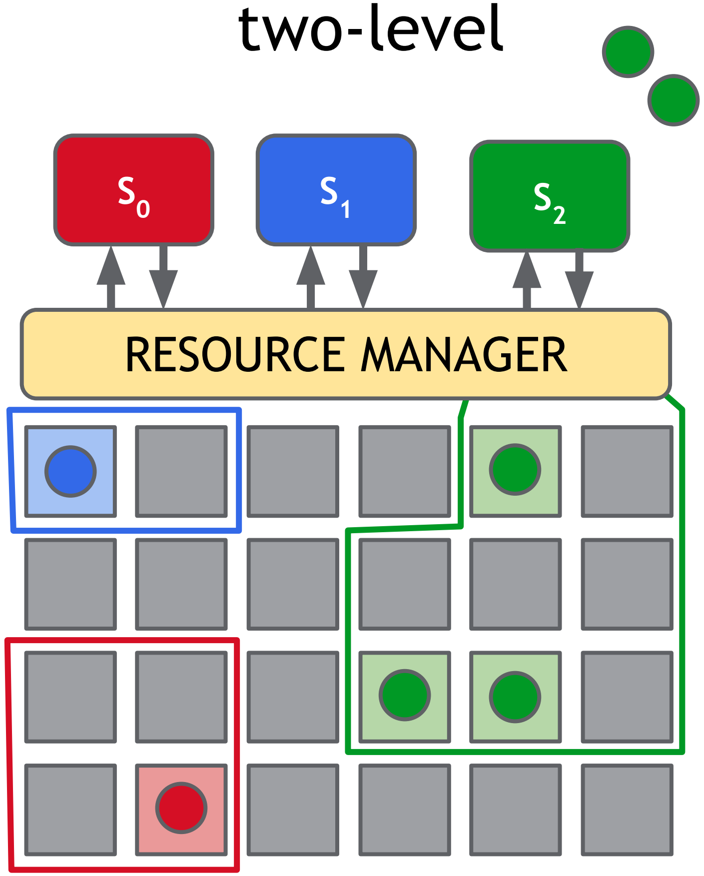
static partitioning



- poor utilization
- inflexible

existing approaches

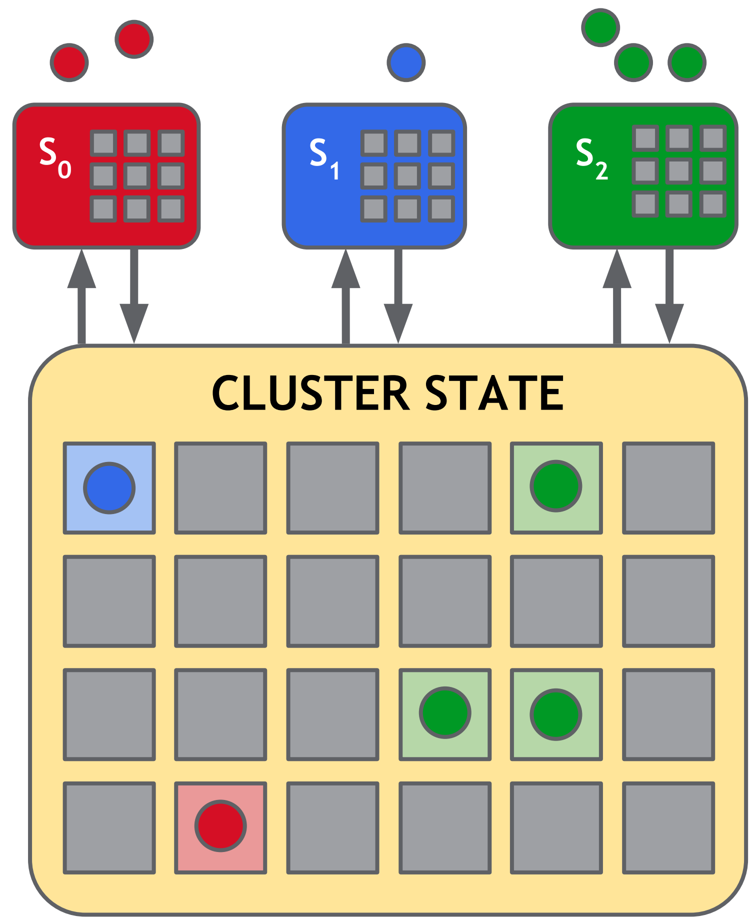
two-level



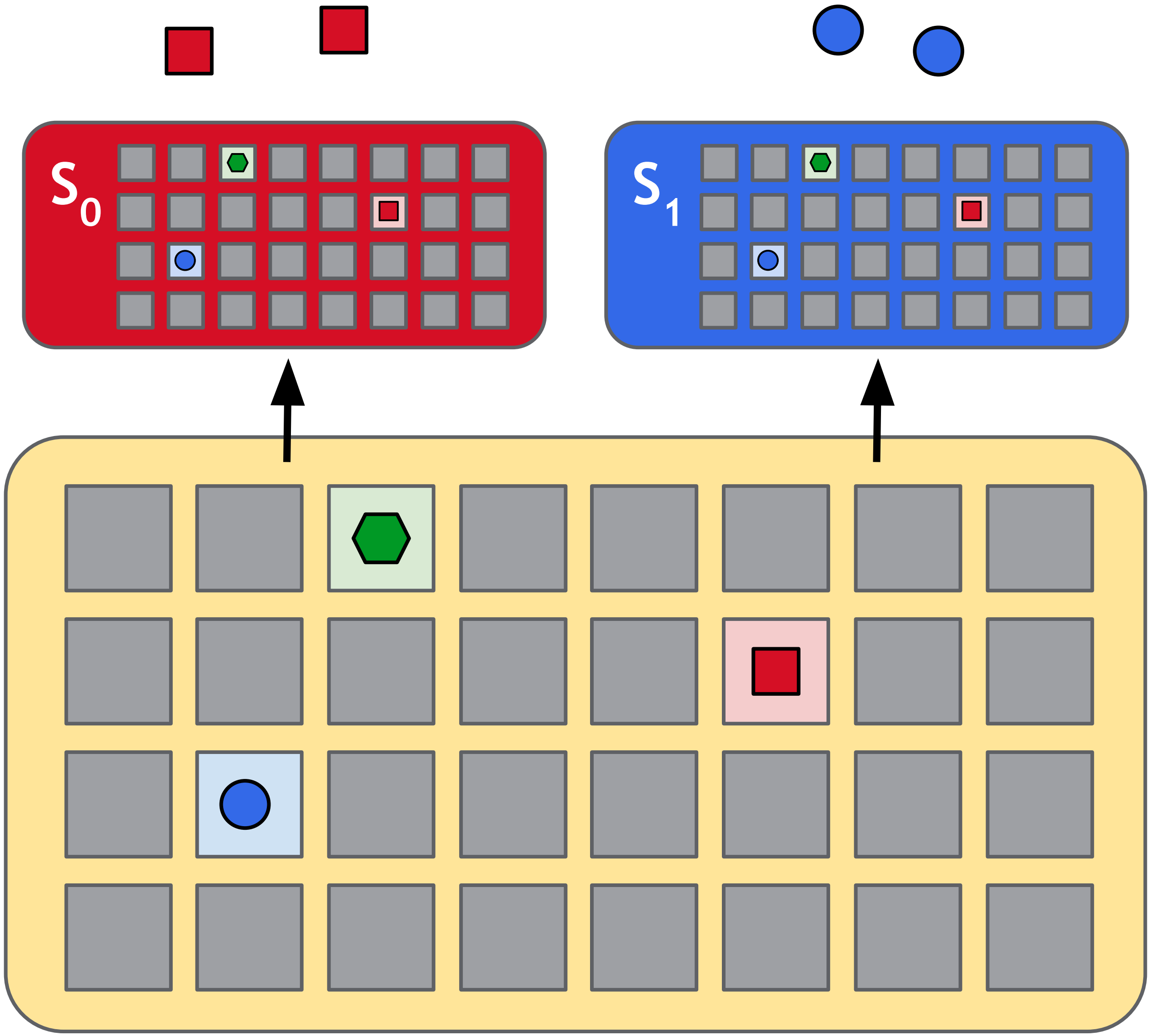
- hoarding
- information hiding

e.g. UCB Mesos [NSDI 2011]

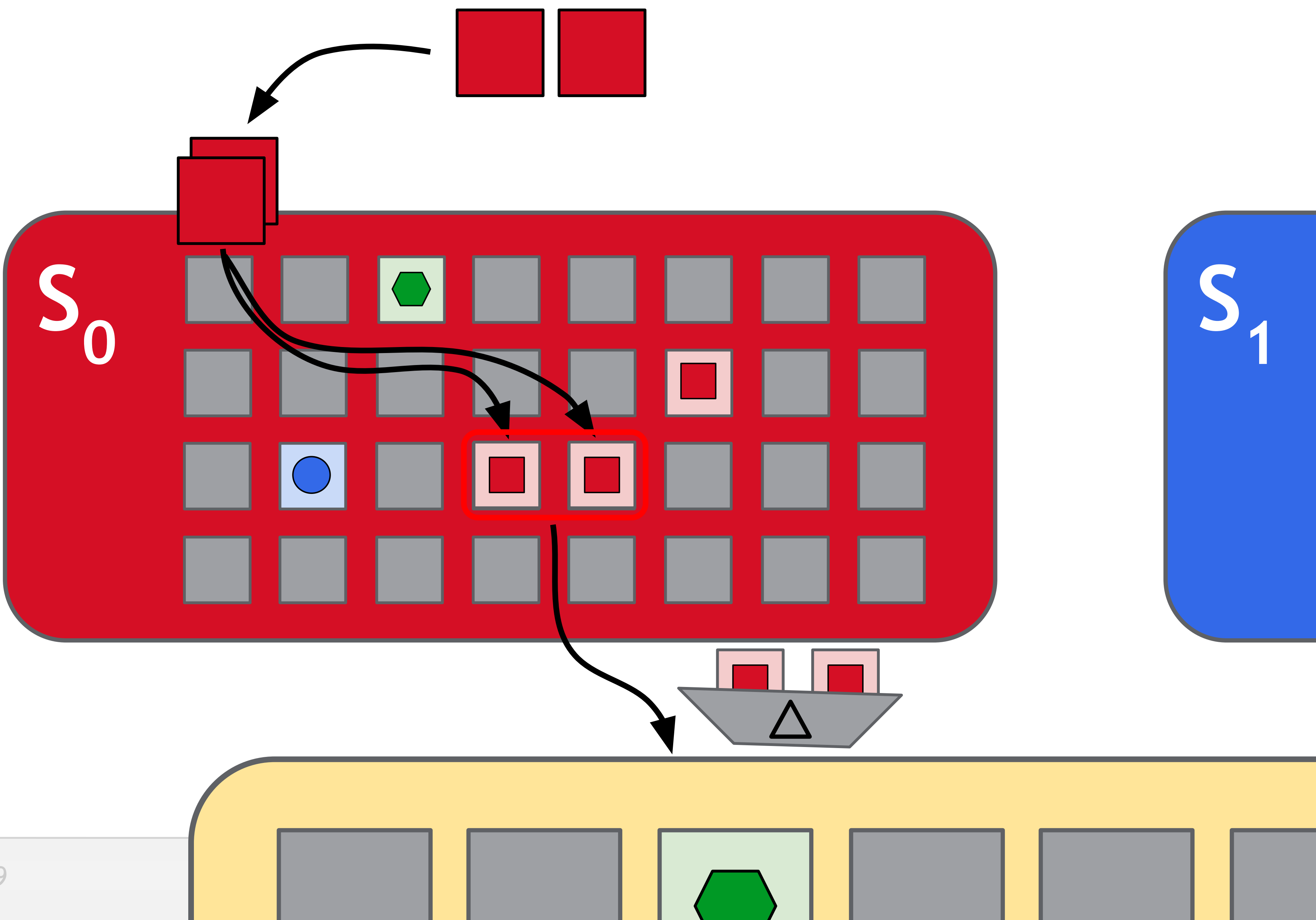
shared-state



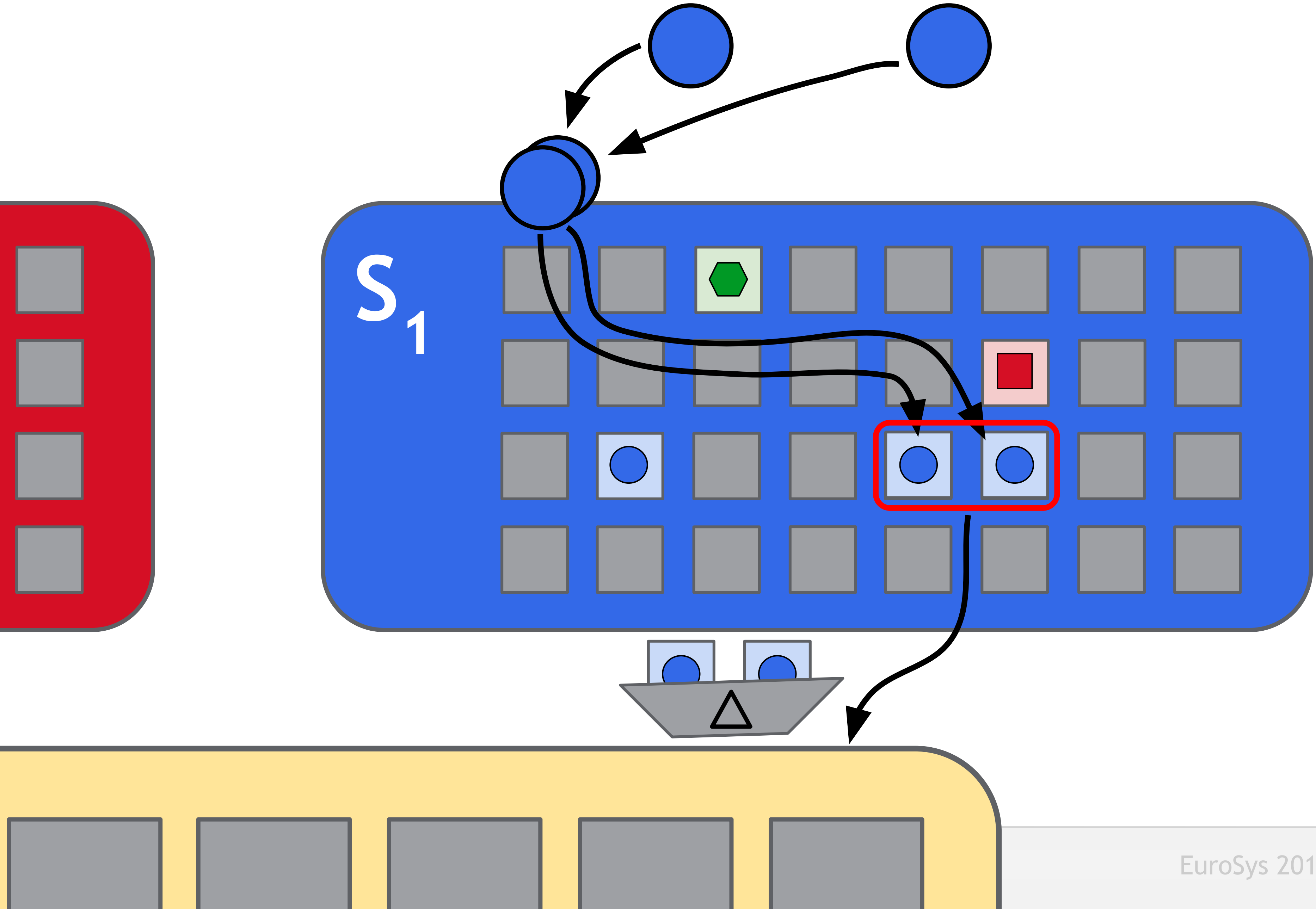
how does omega work?

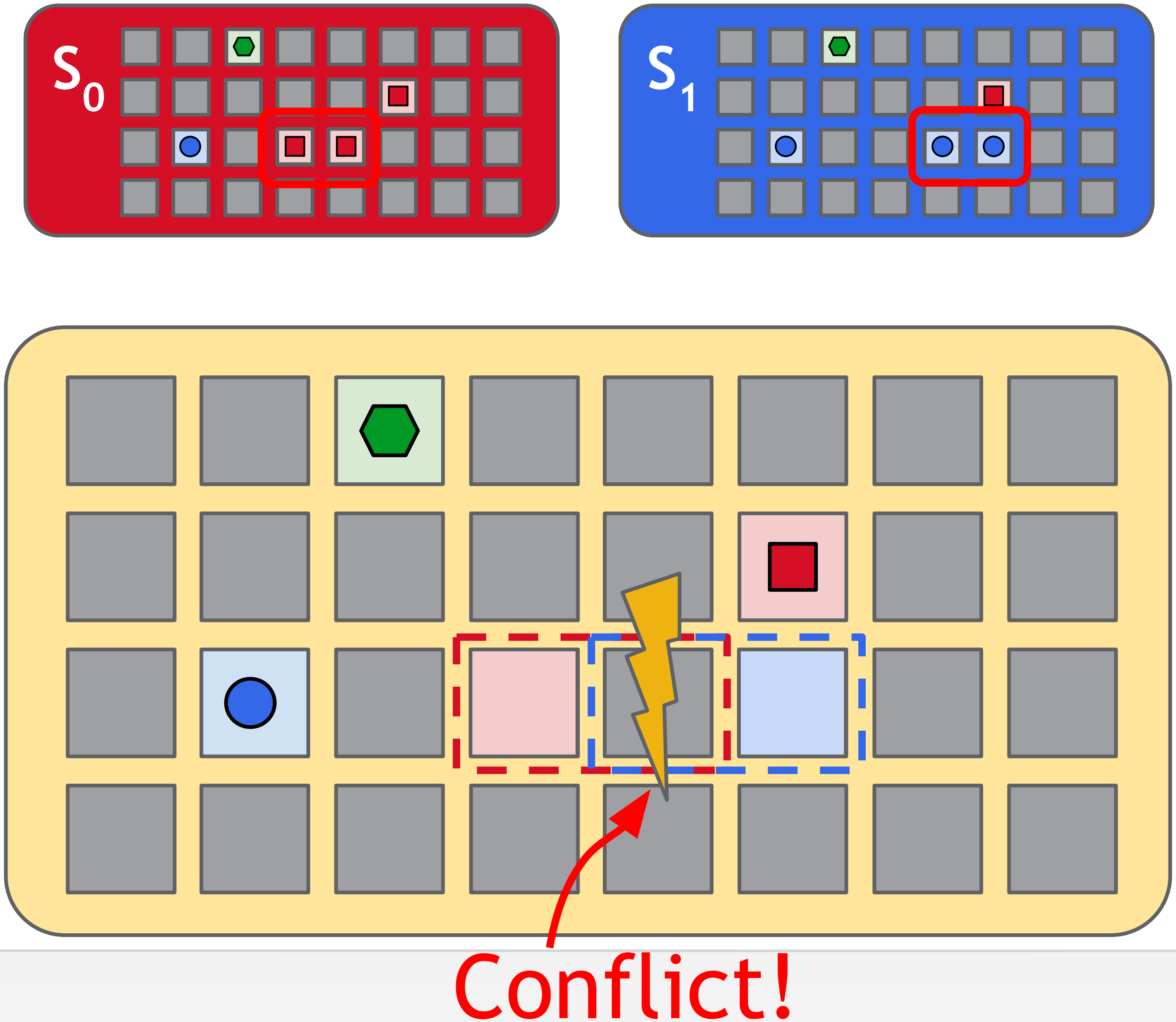


how does omega work?



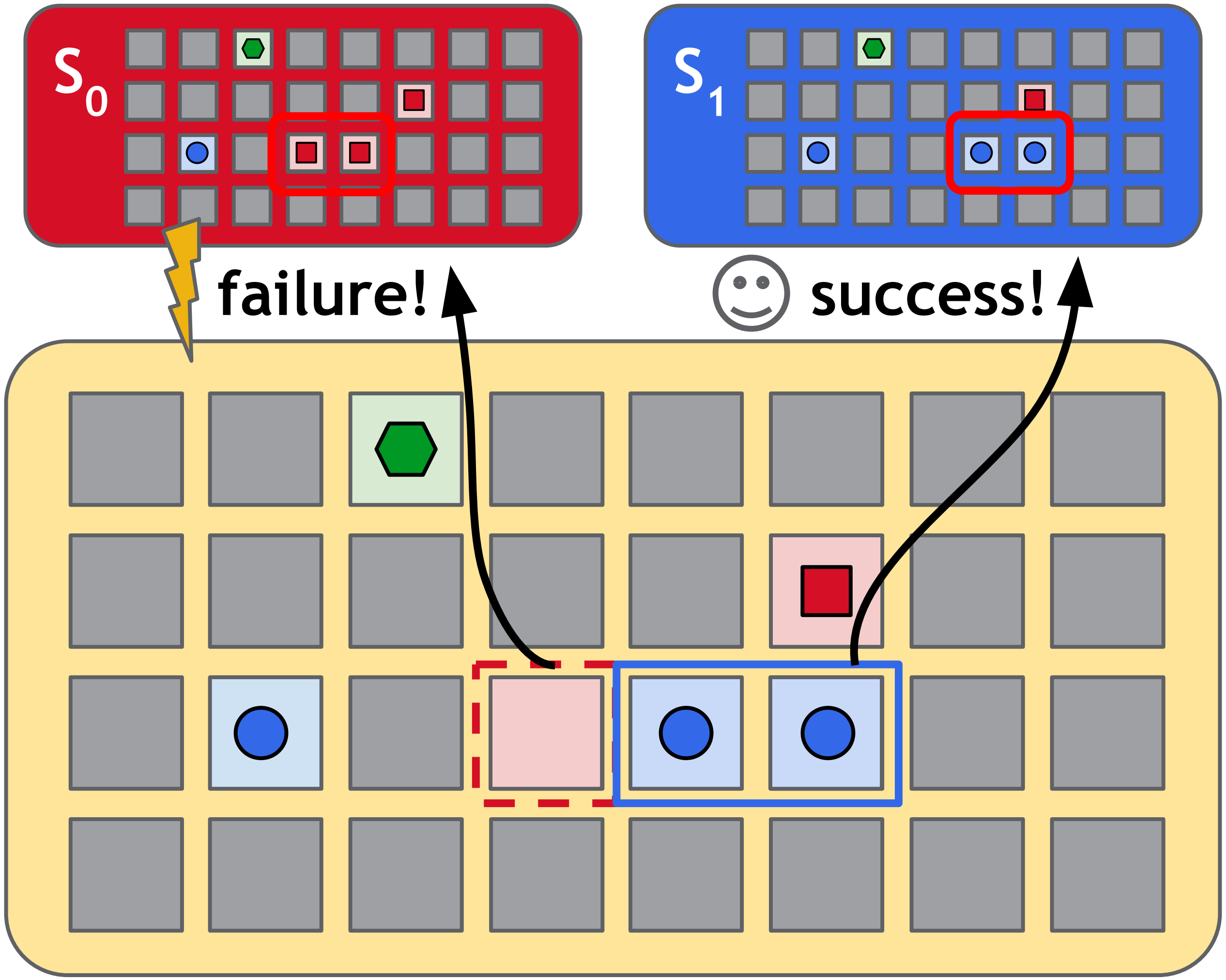
how does omega work?





S_0
S_1
Conflict!

how does omega work?



1) intro & motivation

2) workload characterization

3) comparison of approaches

4) trace-based simulation

5) flexibility case study

Batch

Service

workload: batch/service split

Cluster A

Medium size

Medium utilization

Cluster B

Large size

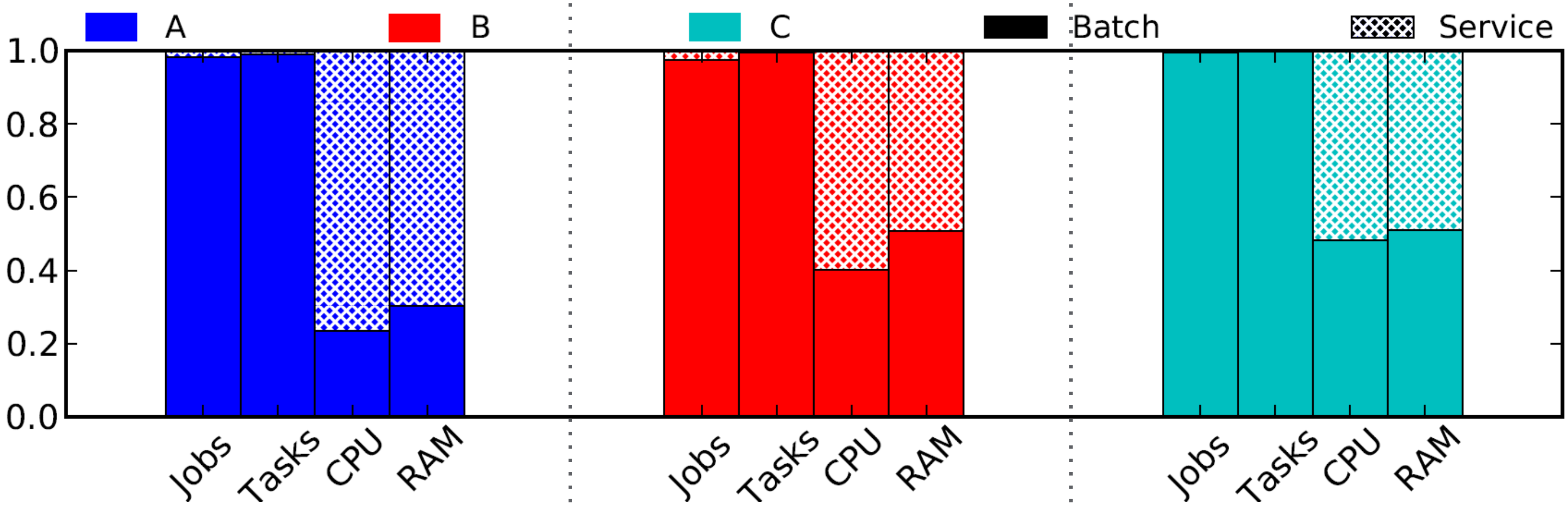
Medium utilization

Cluster C

Medium (12k mach.)

High utilization

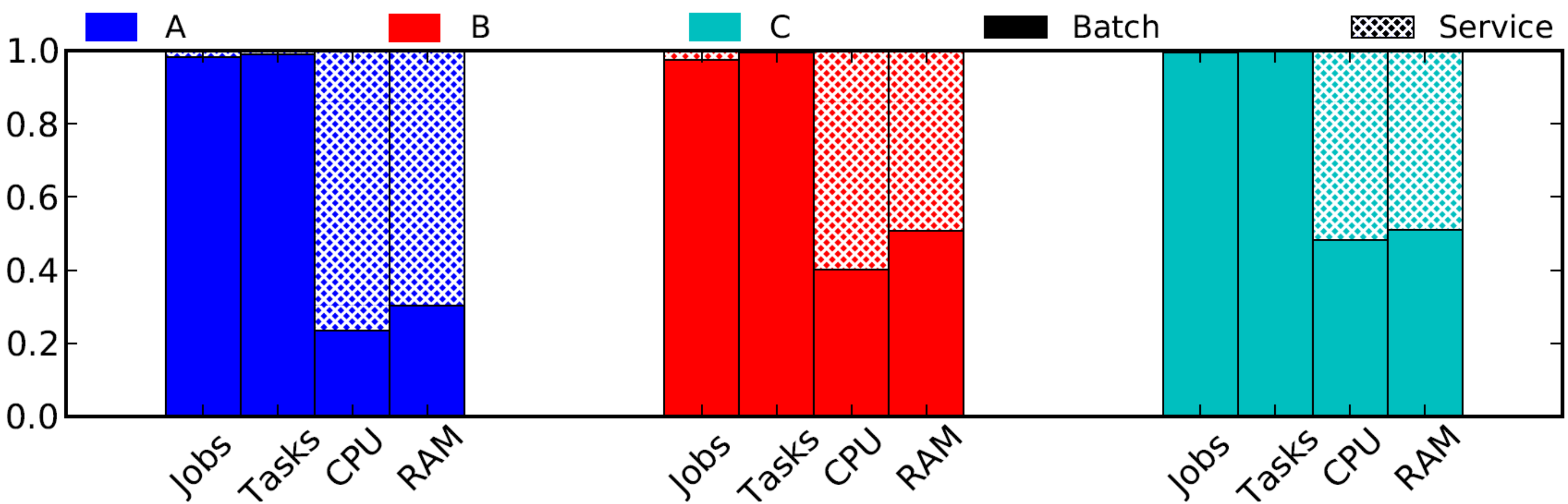
Public trace



Jobs/tasks: counts
CPU/RAM: resource seconds [i.e. resource job runtime in sec.]

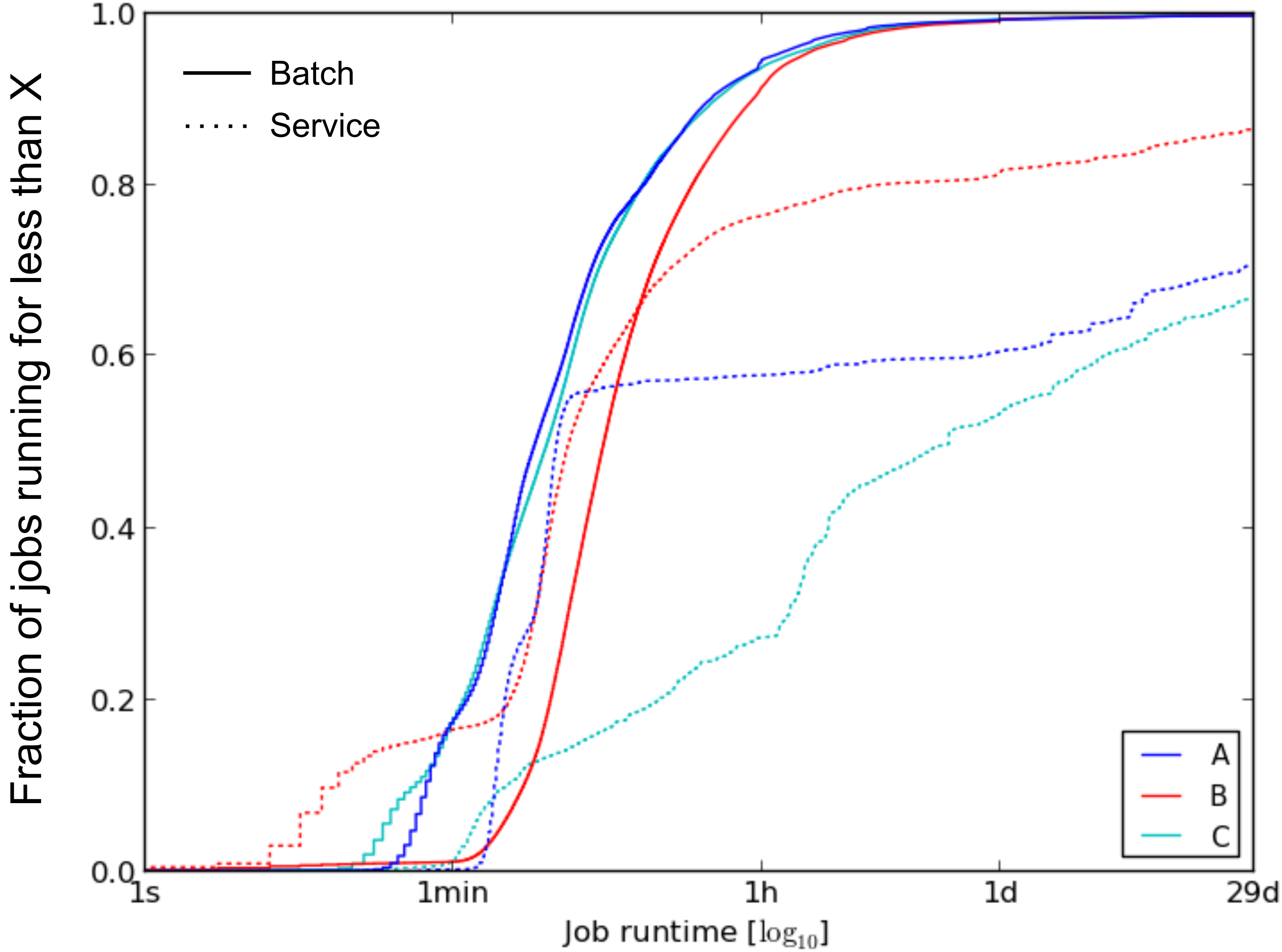
Takeaway

Most jobs are batch, but most resources are consumed by service jobs.

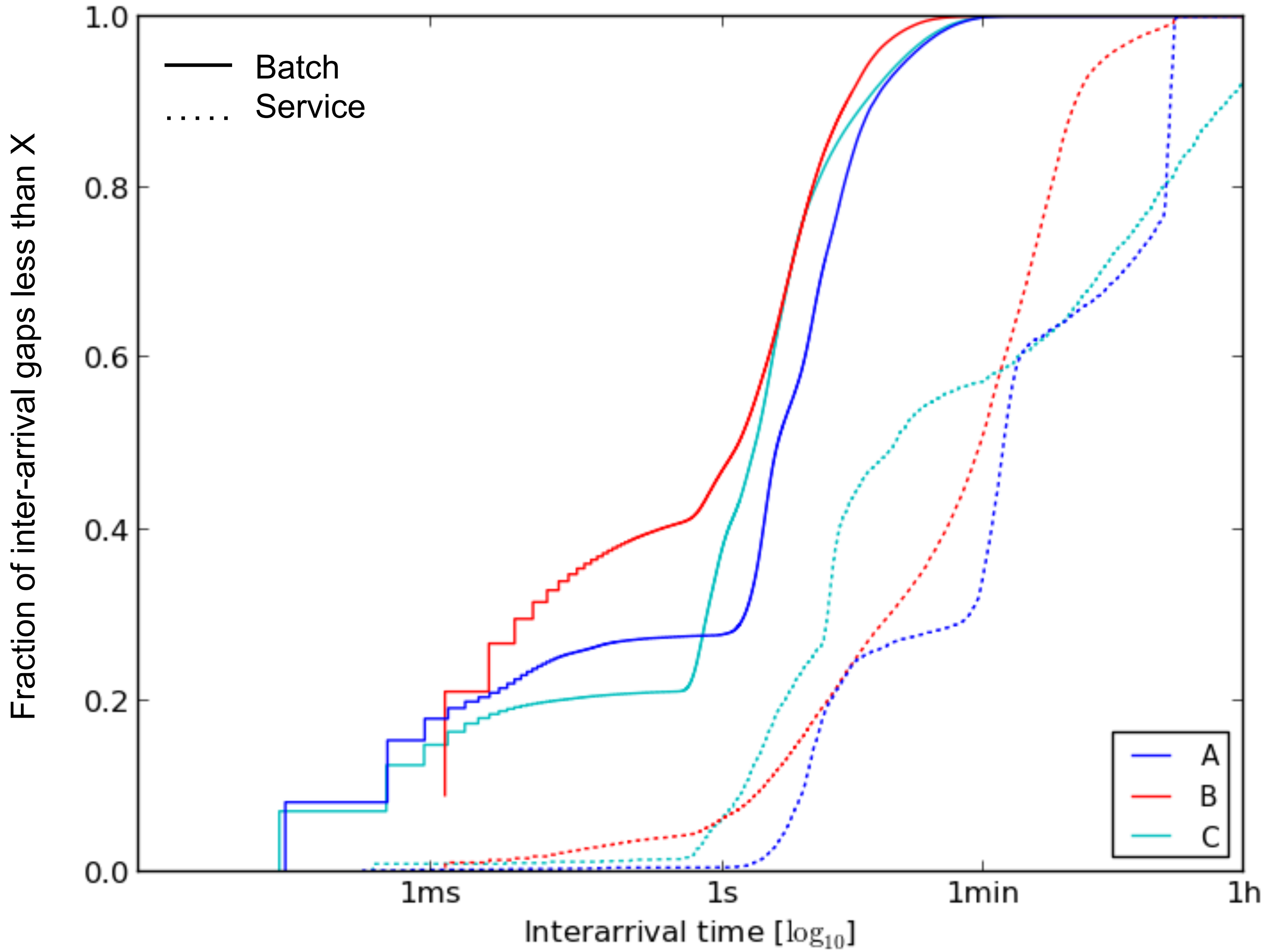


Jobs/tasks: counts
CPU/RAM: resource seconds [i.e. resource job runtime in sec.]

workload: job runtime distributions



workload: inter-arrival time distributions



Batch jobs	*Service jobs*
80th %ile runtime	
12-20 min.	**29 days**
80th %ile inter-arrival time	
4-7 sec.	**2-15 min.**

1) intro & motivation

2) workload characterization

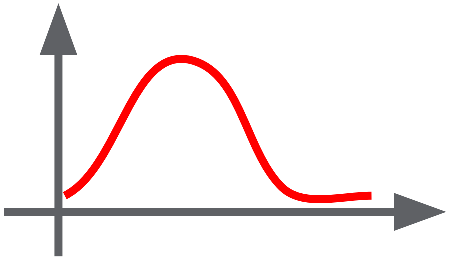
3) comparison of approaches

4) trace-based simulation

5) flexibility case study

simulation using

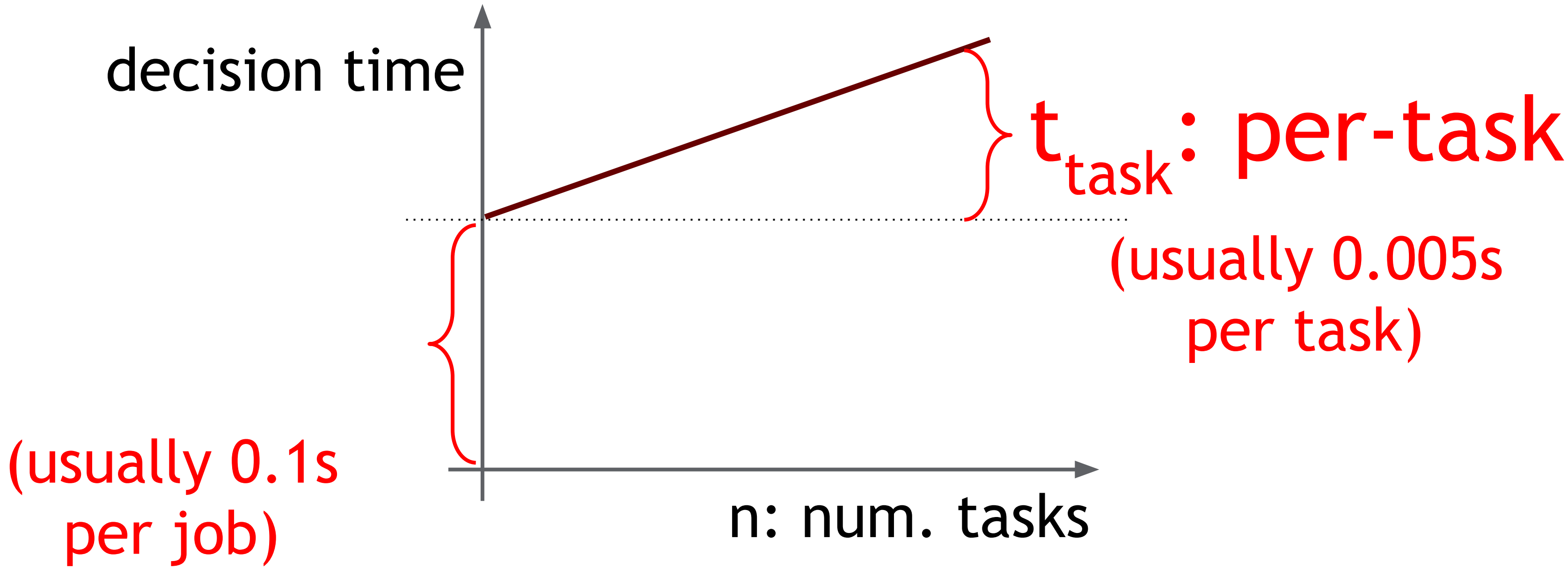
empirical workload parameters distributions



Code available:

`http://code.google.com/p/cluster-scheduler-simulator`

Scheduler decision time



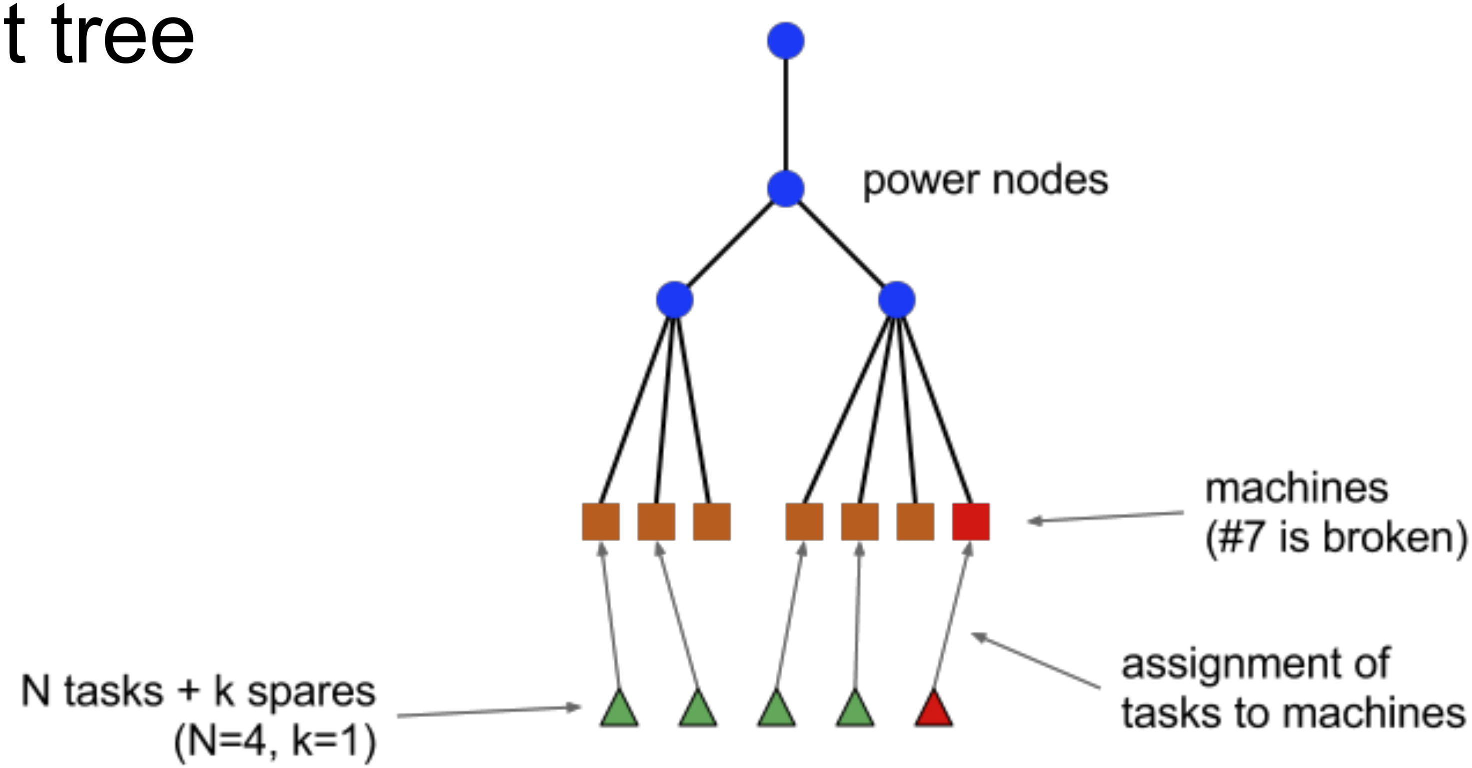
Why might scheduling take 60 seconds?

- Large jobs (tens of thousands of tasks)
- Optimization algorithms (constraints, bin packing with knock-on preemption)
- Picky jobs in a full cluster
- Monte Carlo simulations (fault tolerance)

Open issues: *failure tolerance*

Topology-aware scheduling
for concurrent outages

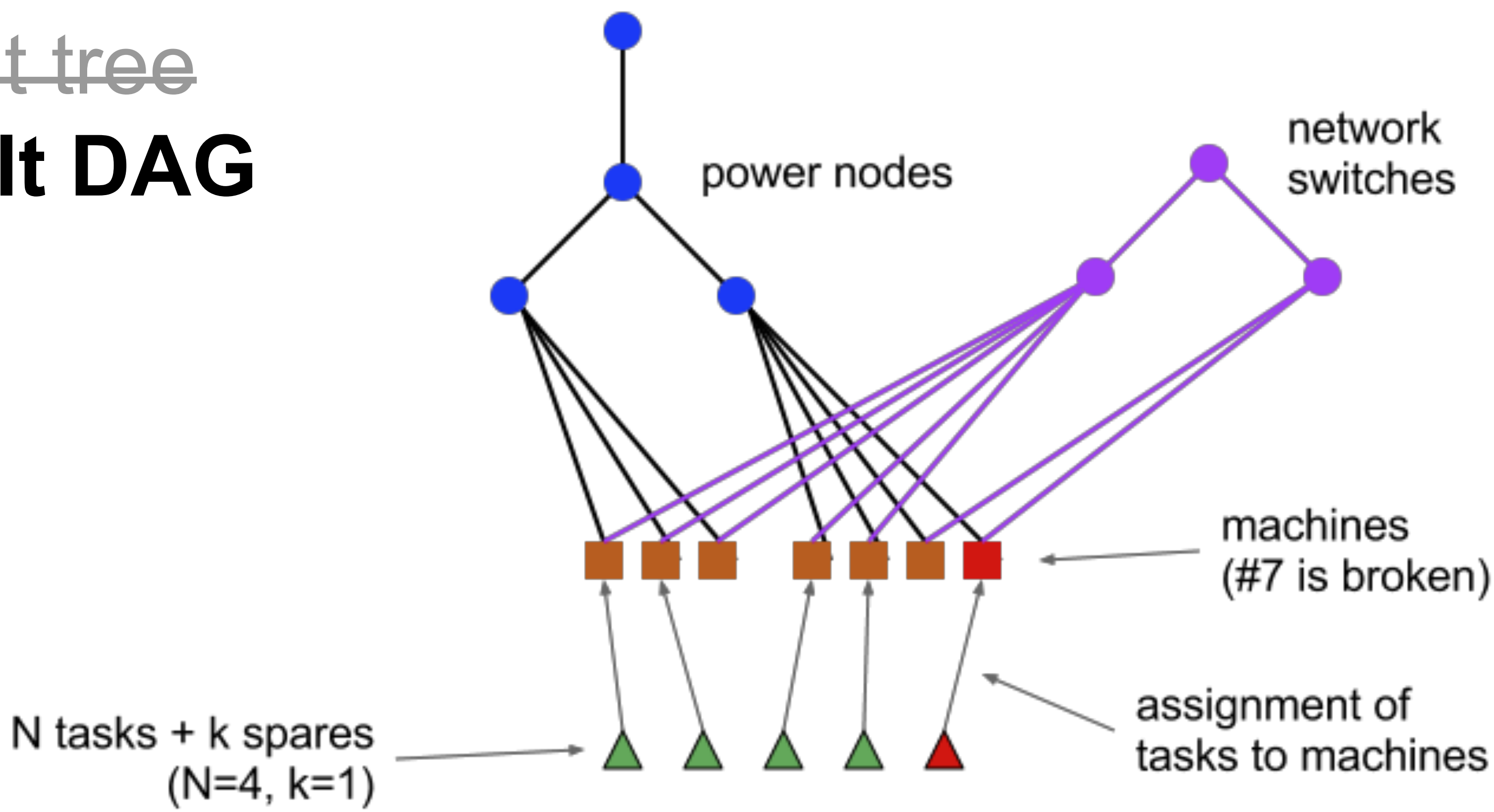
- a fault tree



Open issues: *failure tolerance*

Topology-aware scheduling
for concurrent outages

- ~~a fault tree~~
- **a fault DAG**

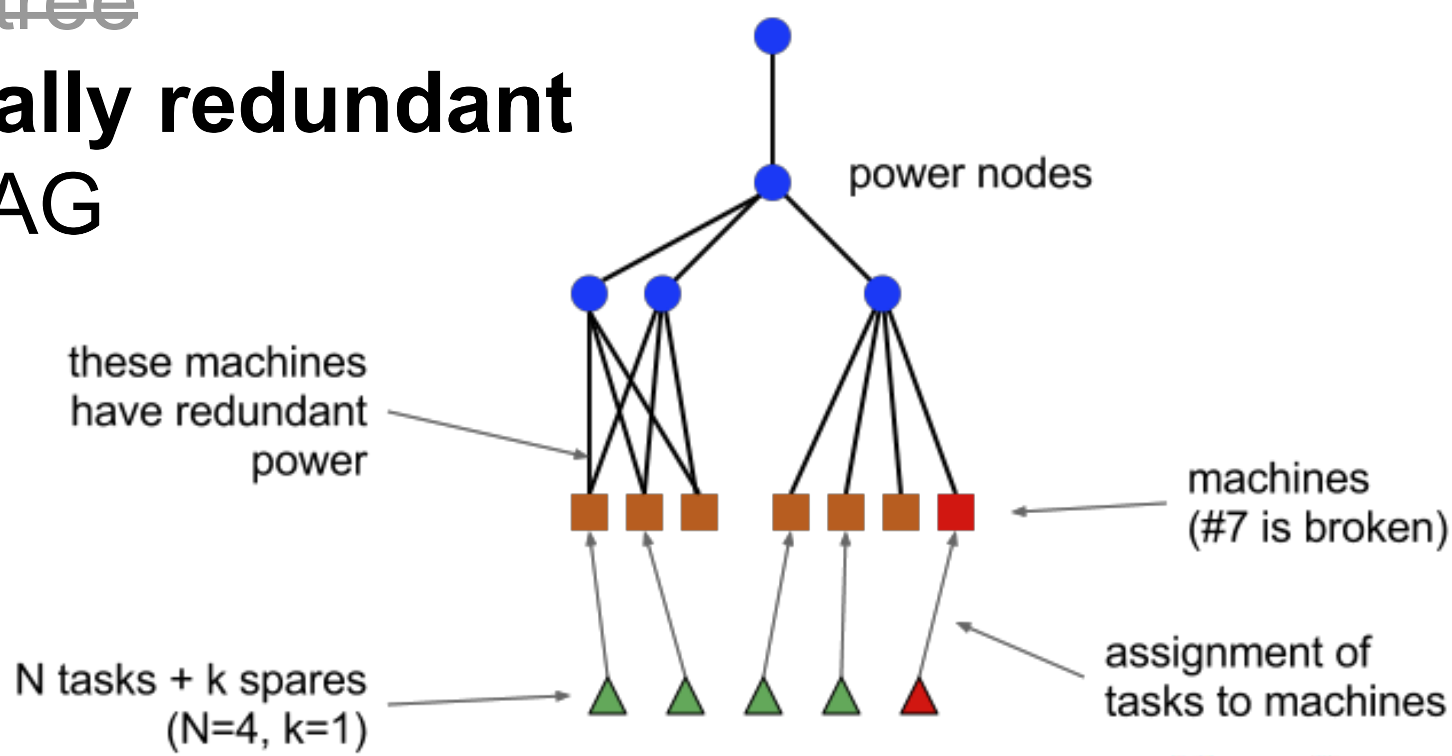


Google

Open issues: *failure tolerance*

Topology-aware scheduling
for concurrent outages

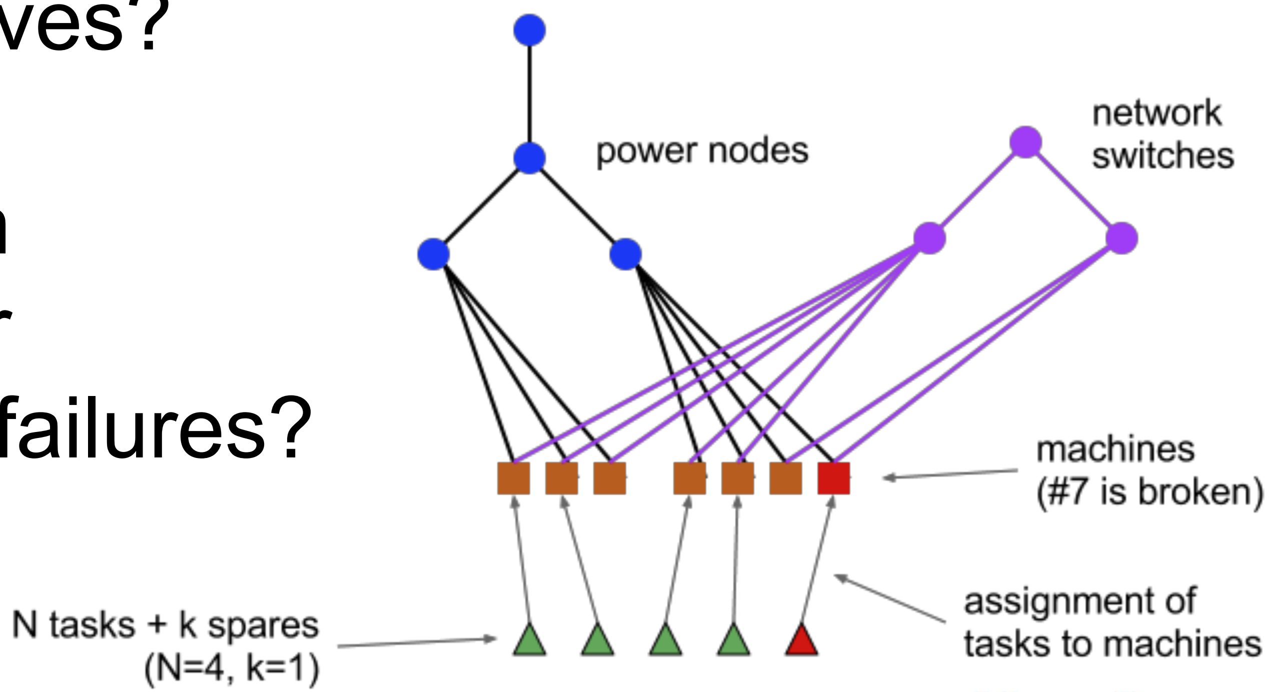
- ~~a fault tree~~
- a **partially redundant** fault DAG



Google

Open issues: *failure tolerance*

- real fault, or lost touch?
- time to detect vs. false positives?
- multiple information sources for correlated failures?



Google

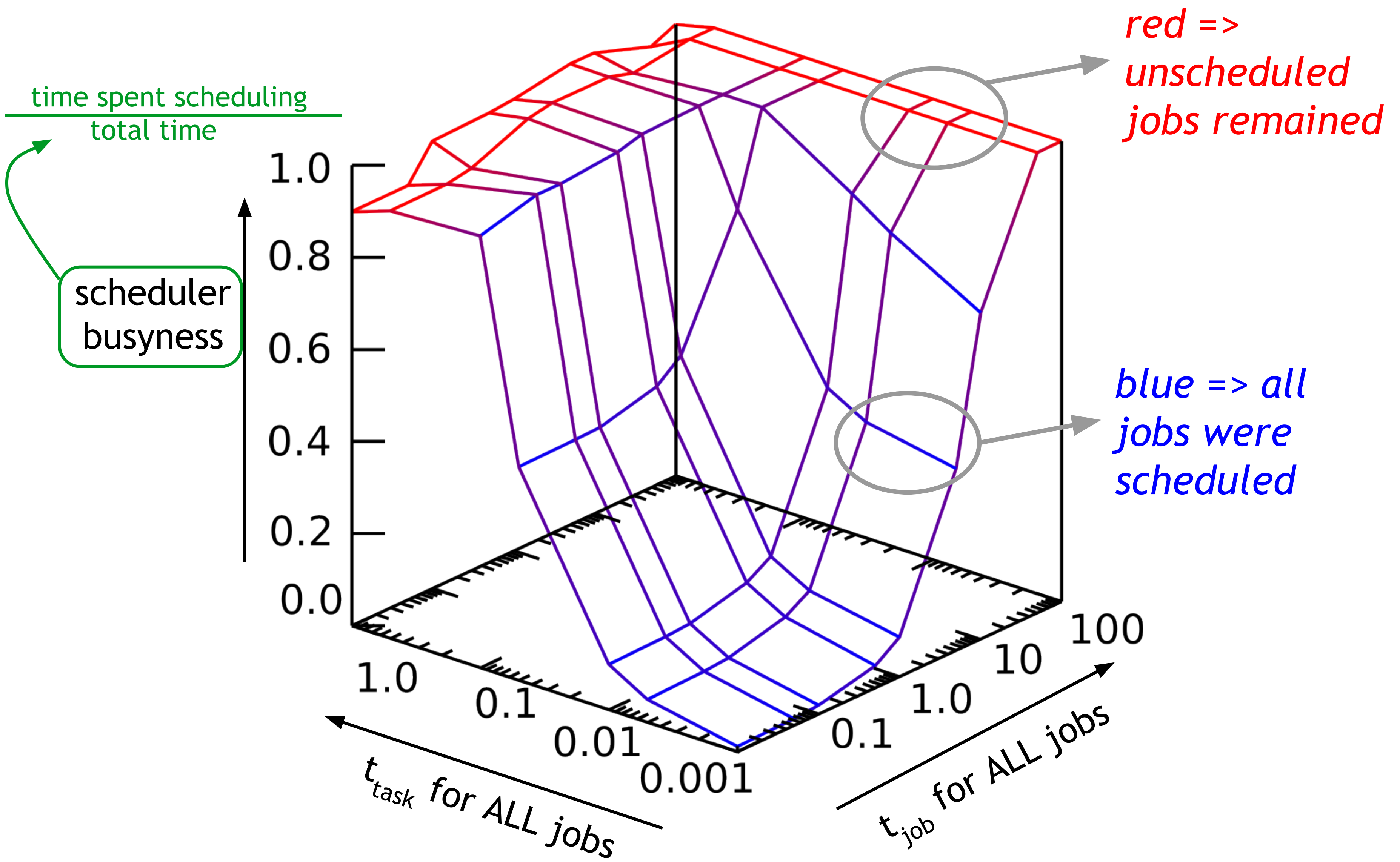
Experiment 1:

How do does the shared-state design compare with other architectures?

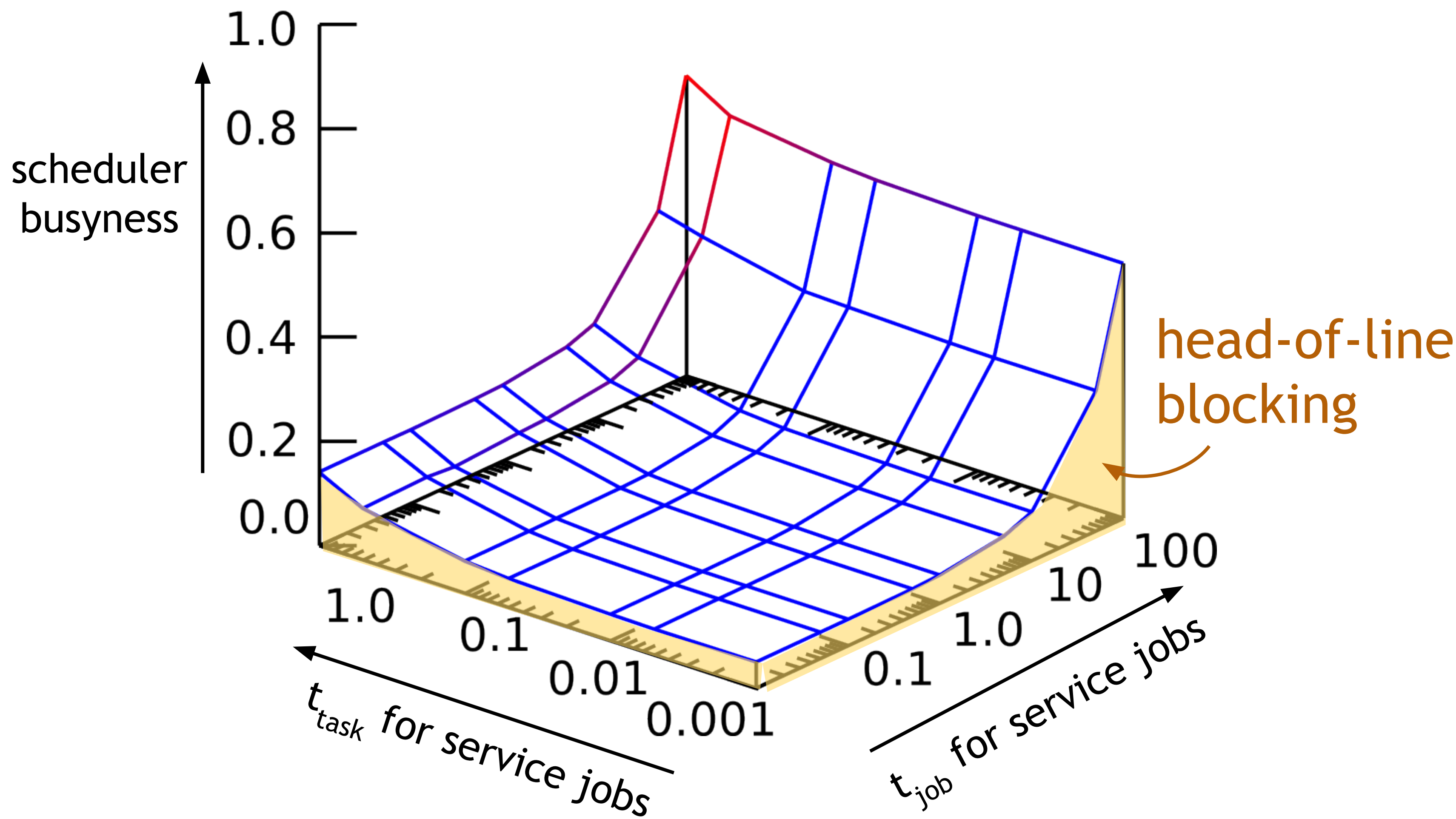
Experiment details:

- all clusters, 7 simulated days
- 2 schedulers
- varying **Service** scheduler

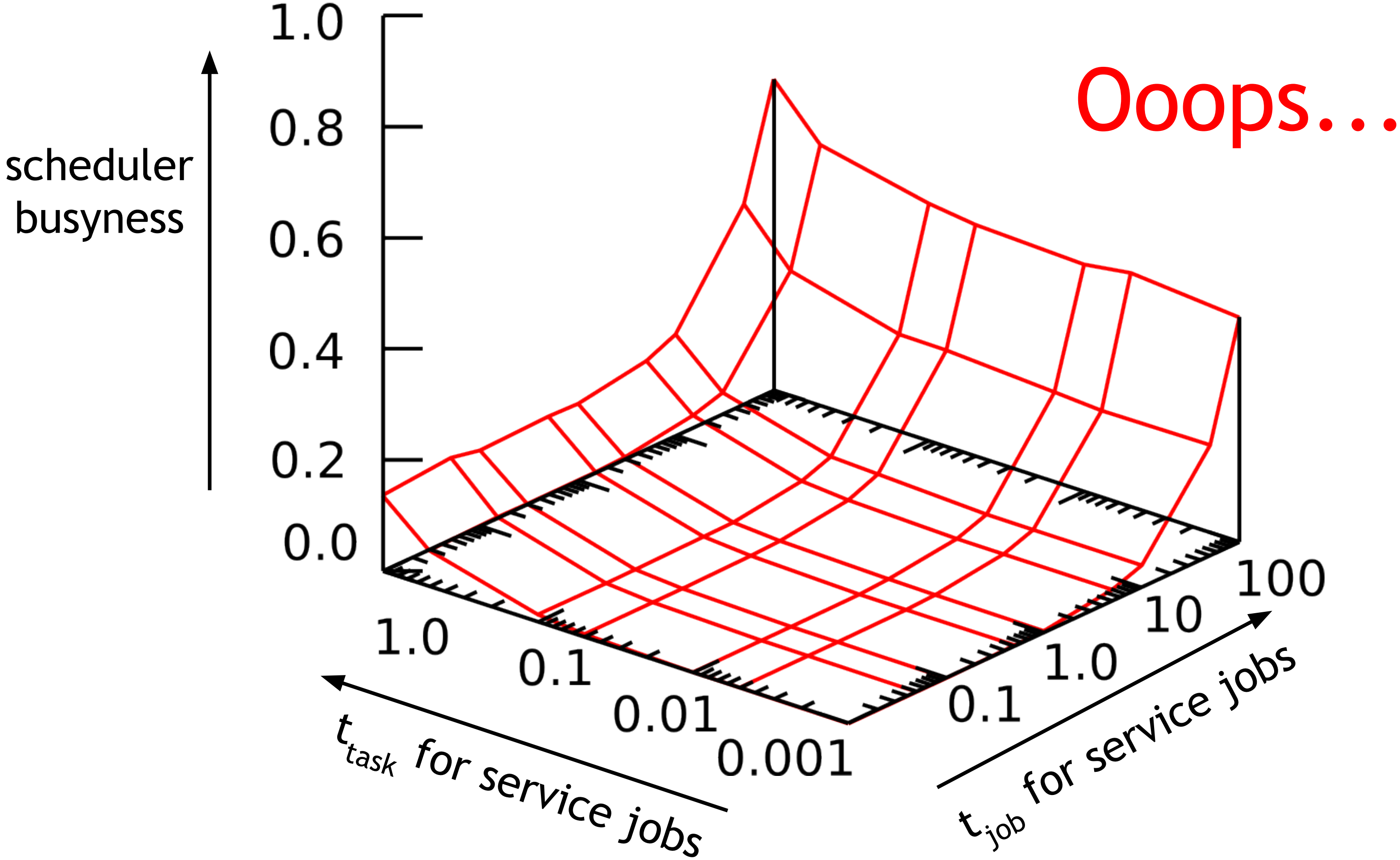
monolithic, uniform decision time (single logic)

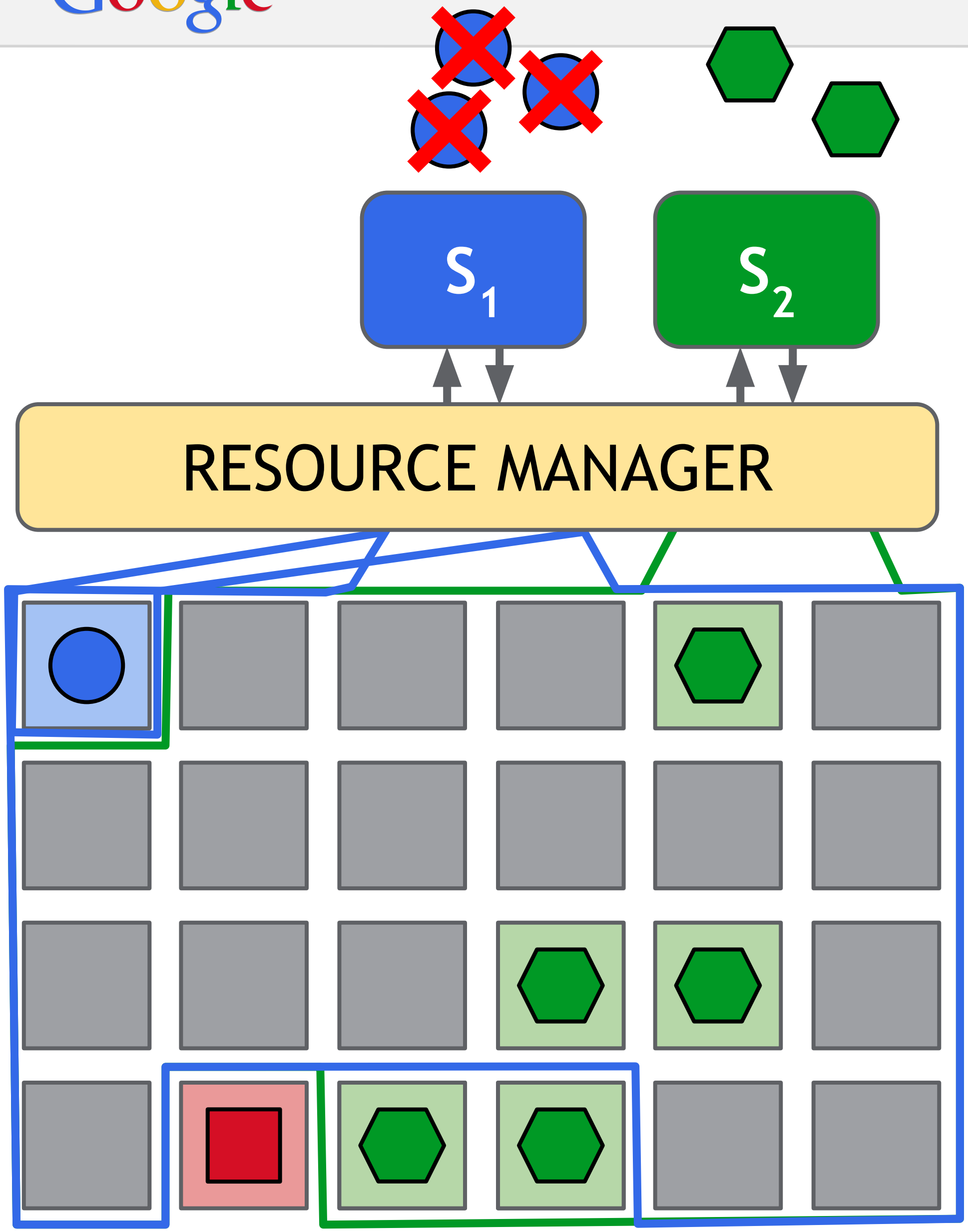


monolithic, fast-path batch decision time



mesos v0.9 (of May 2012)





1. Green receives offer of all available resources.

2. Blue's task finishes.

3. Blue receives tiny offer.

4. Blue cannot use it.

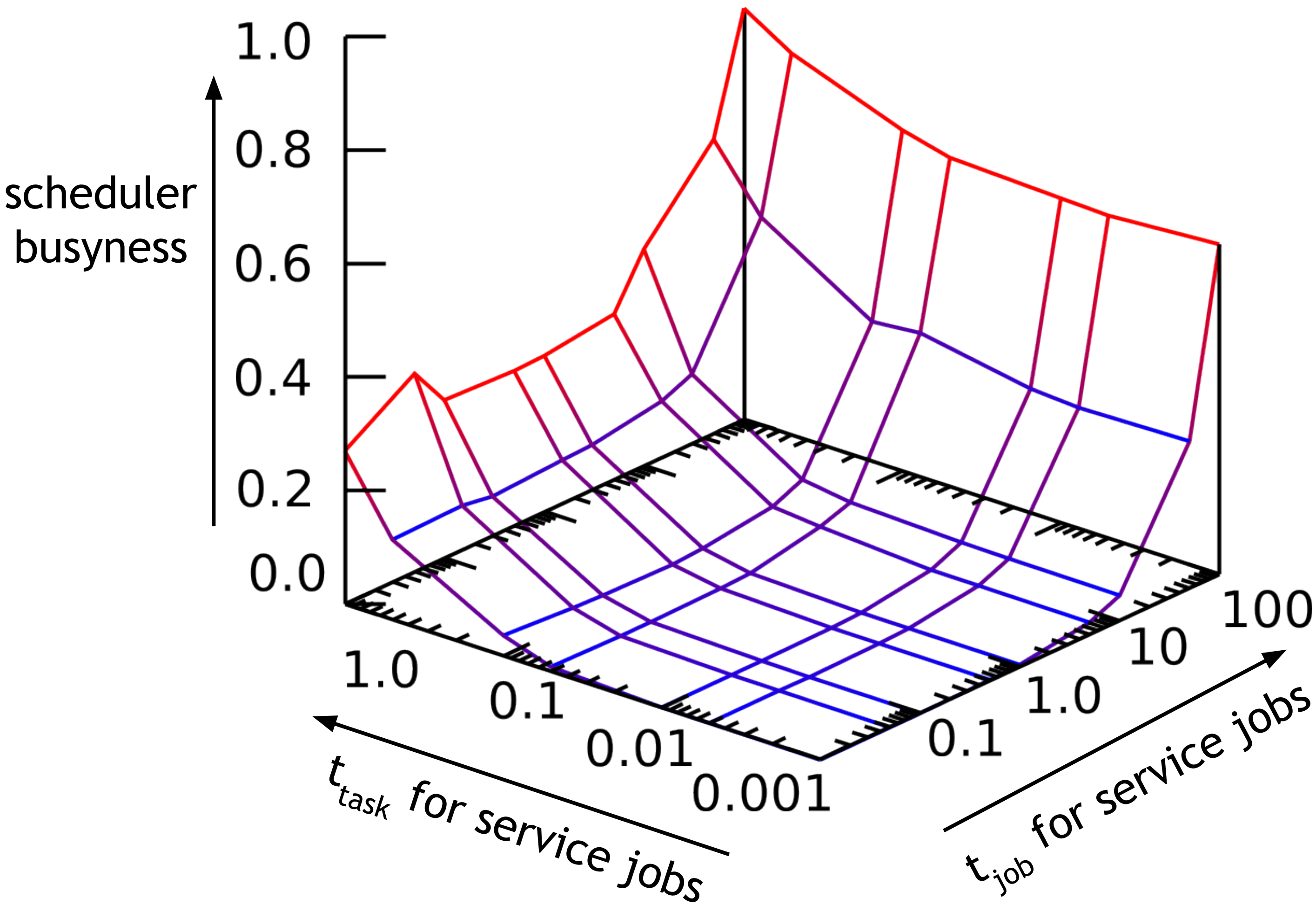
[repeat many times]

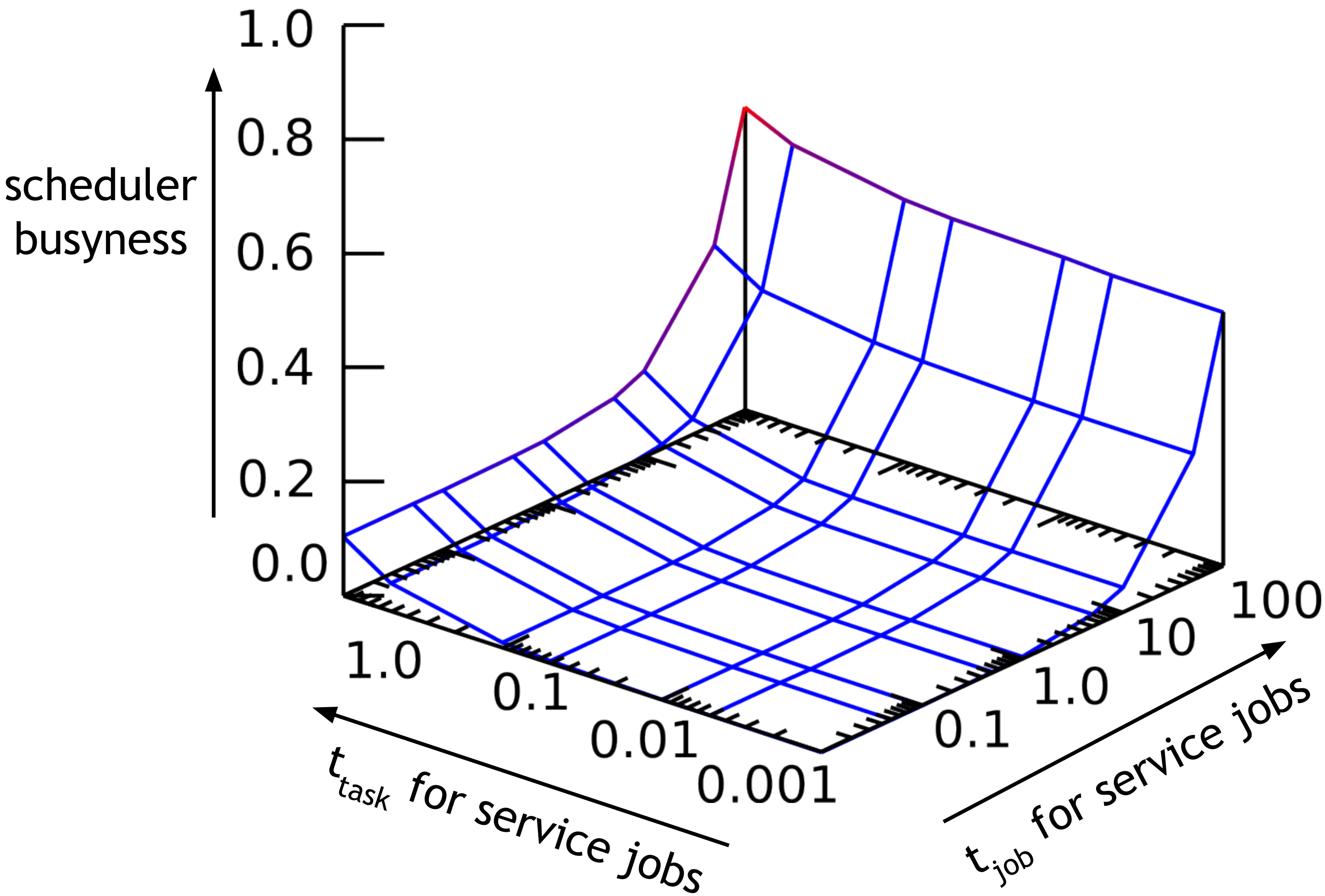
5. Green finishes scheduling.

6. Blue receives large offer.

By now, it has given up.

omega, no optimizations

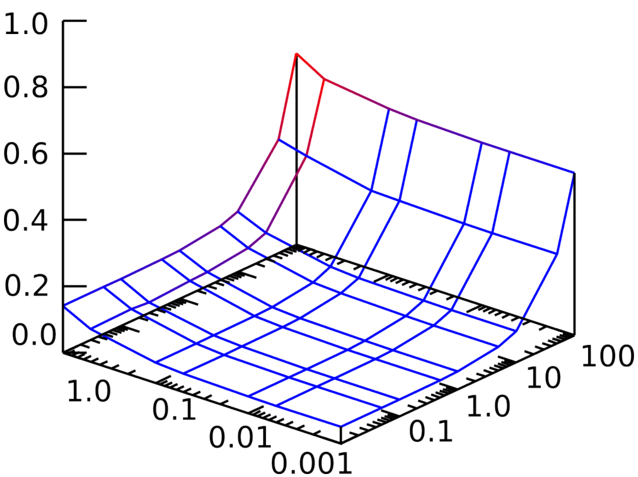




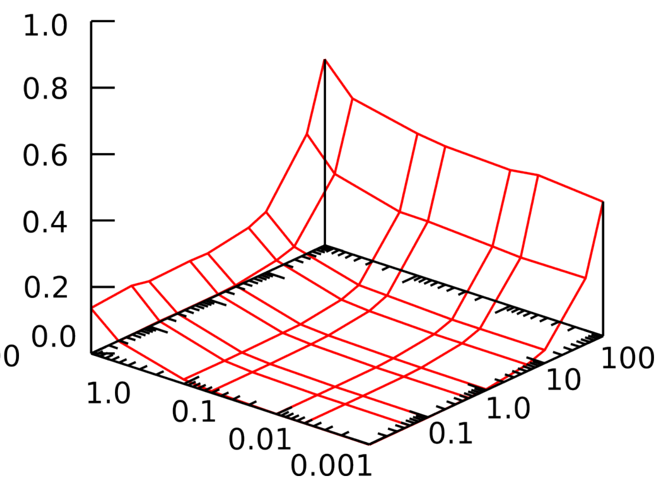
scheduler busyness
1.0
0.8
0.6
0.4
0.2
0.0
1.0
0.1
0.01
0.001
t_{task} for service jobs
0.1
1.0
10
100
t_{job} for service jobs

Takeaway

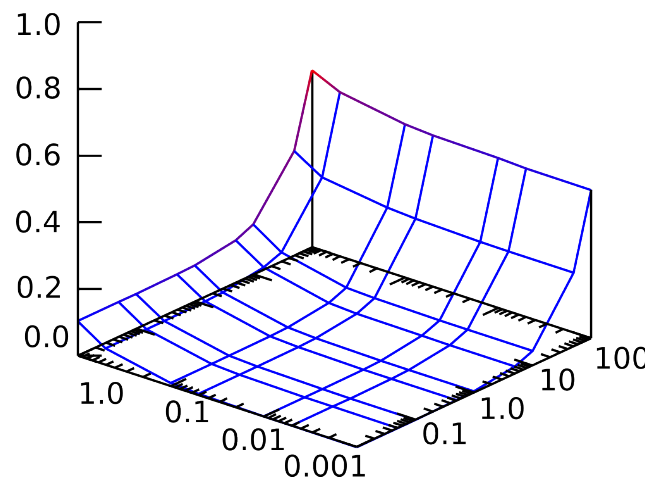
The Omega shared-state model performs as well as a (complex) monolithic multi-path scheduler.



Monolithic



Mesos

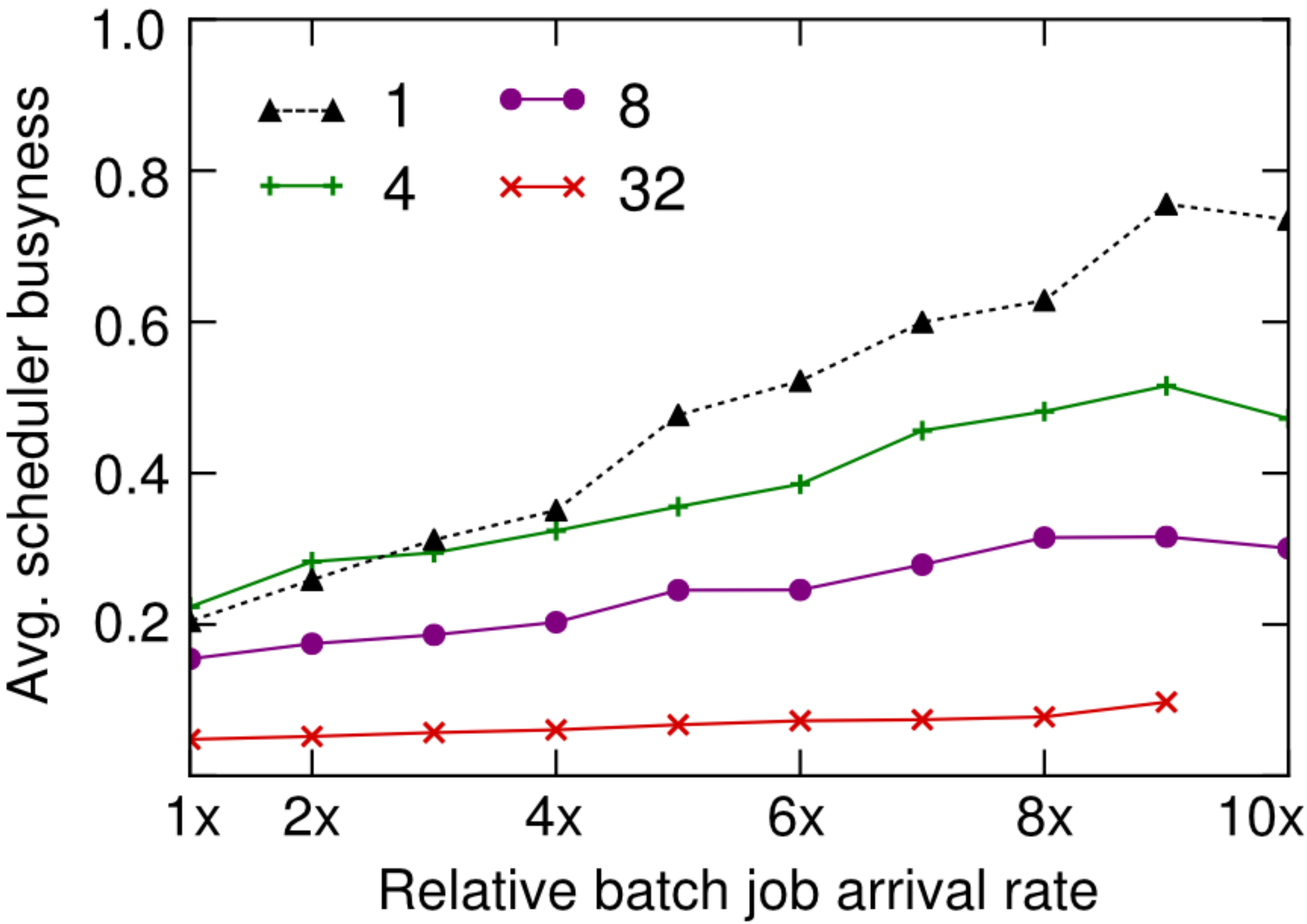


Omega

Experiment 2:

Does the shared-state design scale to many schedulers?

Experiment details:
- cluster B, 7 simulated days
- 2 schedulers
- varying **job arrival rate** and **number of schedulers**


1
8
4
32
1.0
0.8
0.6
0.4
0.2
Avg. scheduler busyness
1x
2x
4x
6x
8x
10x
Relative batch job arrival rate

1) intro & motivation

2) workload characterization

3) comparison of approaches

4) trace-based simulation

5) flexibility case study

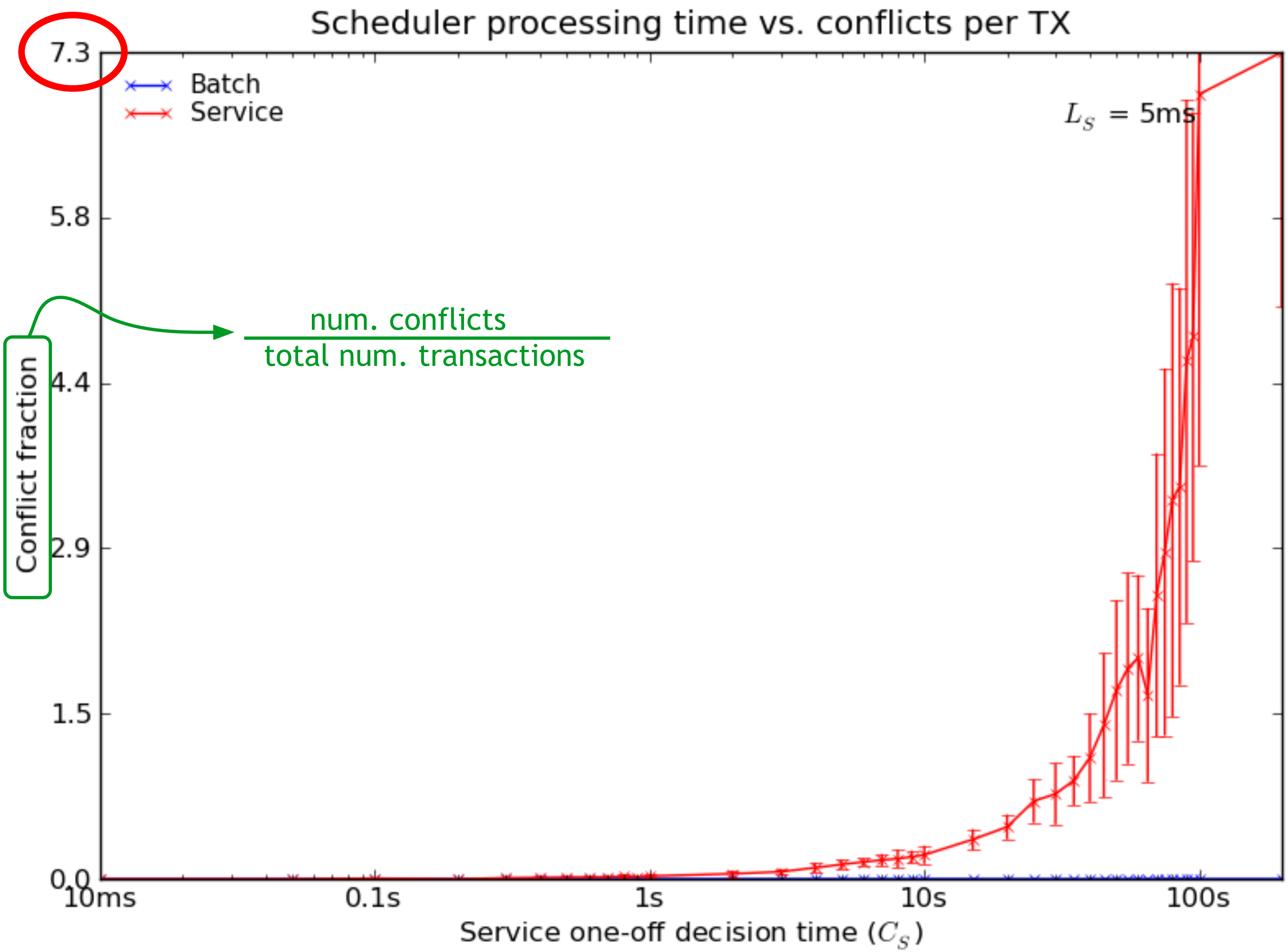
	lightweight simulator	***high-fidelity simulator***
machines	homogeneous	**real-world**
job parameters	empirical distribution	**workload trace**
constraints	not supported	**supported**
scheduling algorithm	random first fit	**Google algorithm**
runtime	fast (24h ≃ 5min)	**slow (24h ≃ 2h)**

Experiment 3:

How much scheduler interference do we see with real Google workloads?

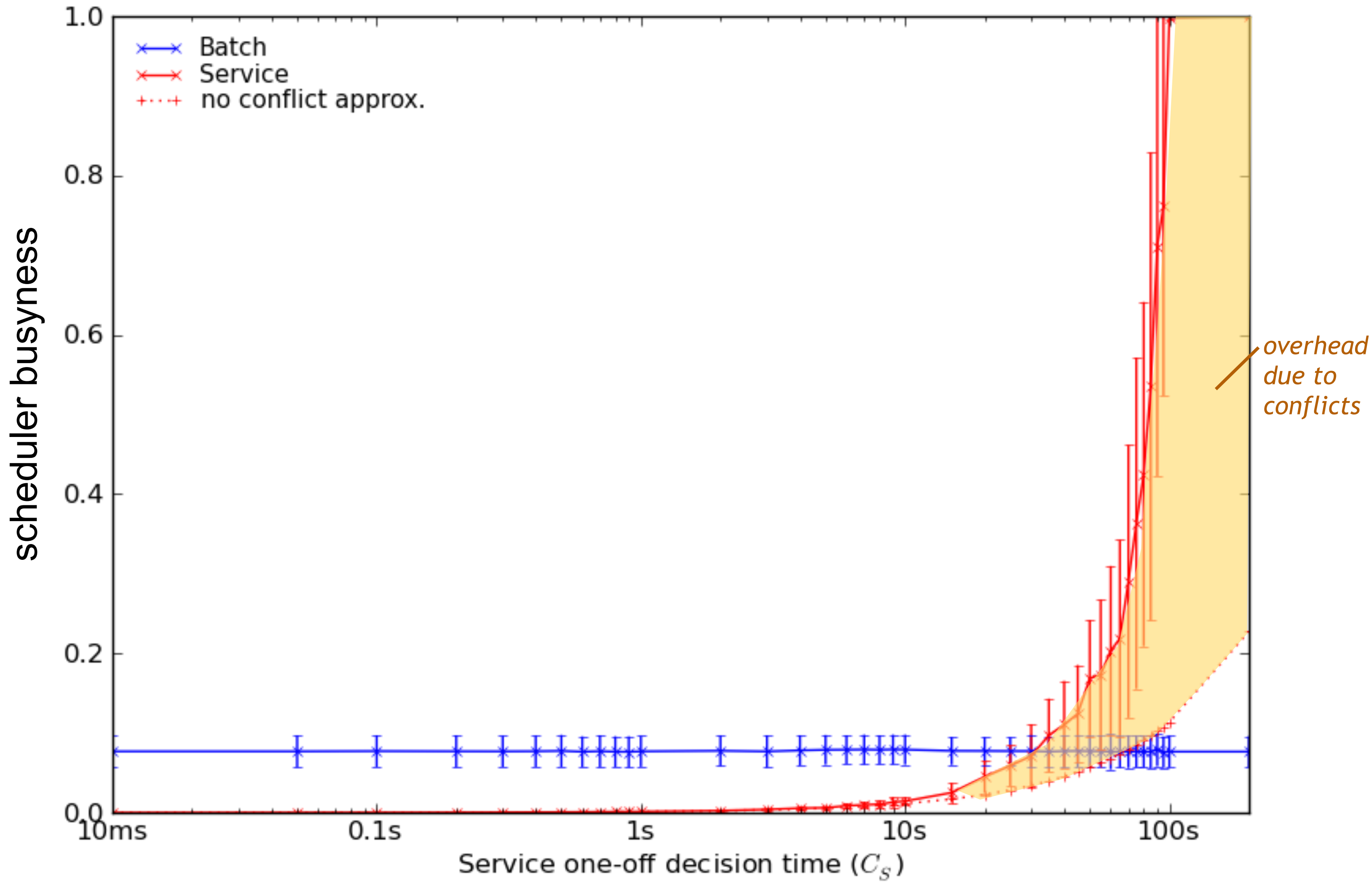
Experiment details:

- cluster C, 29 days
- 2 schedulers,
  non-uniform decision time
- varying **Service** scheduler



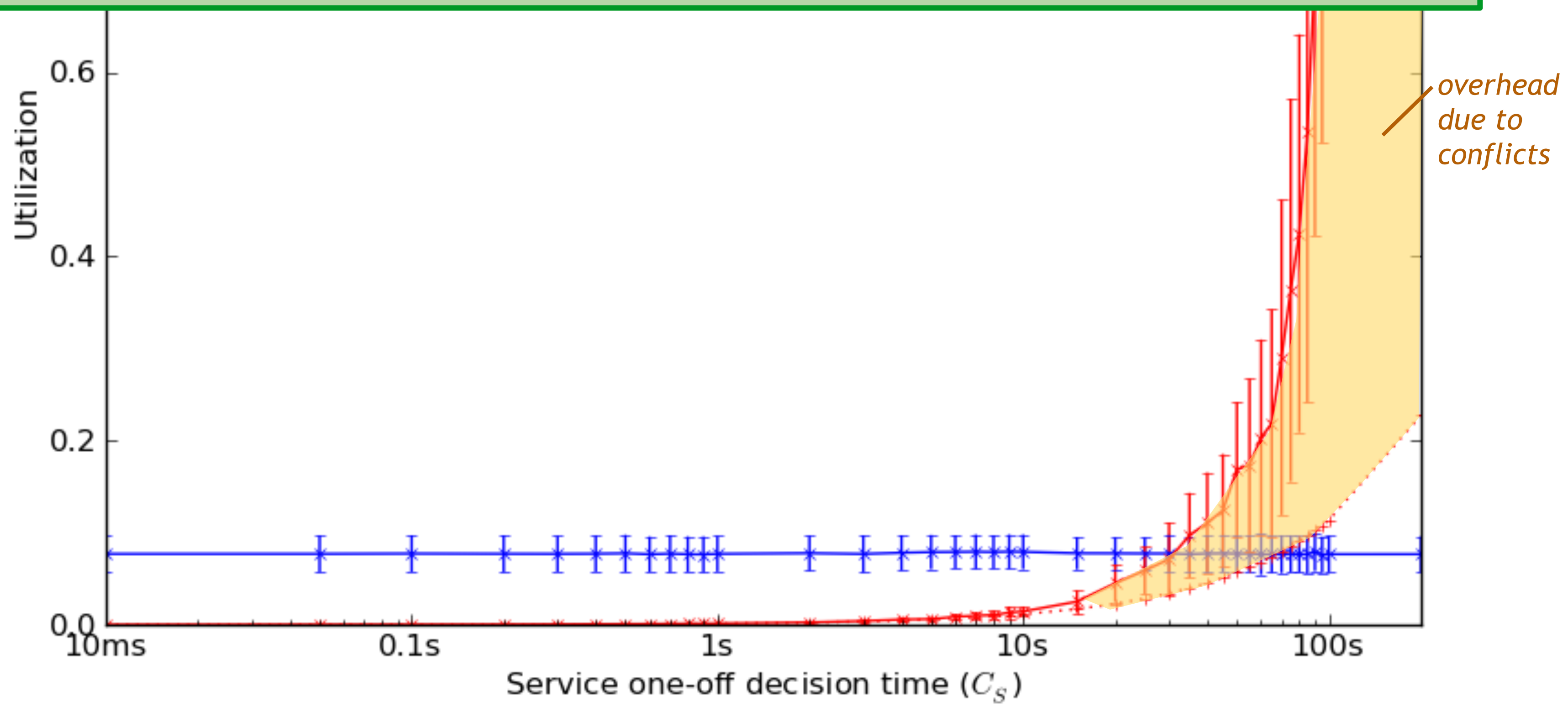
Scheduler processing time vs. conflicts per TX
7.3
5.8
4.4
2.9
1.5
0.0
Conflict fraction
num. conflicts / total num. transactions
Batch
Service
L_S = 5ms
10ms
0.1s
1s
10s
100s
Service one-off decision time (C_S)

scheduler busyness

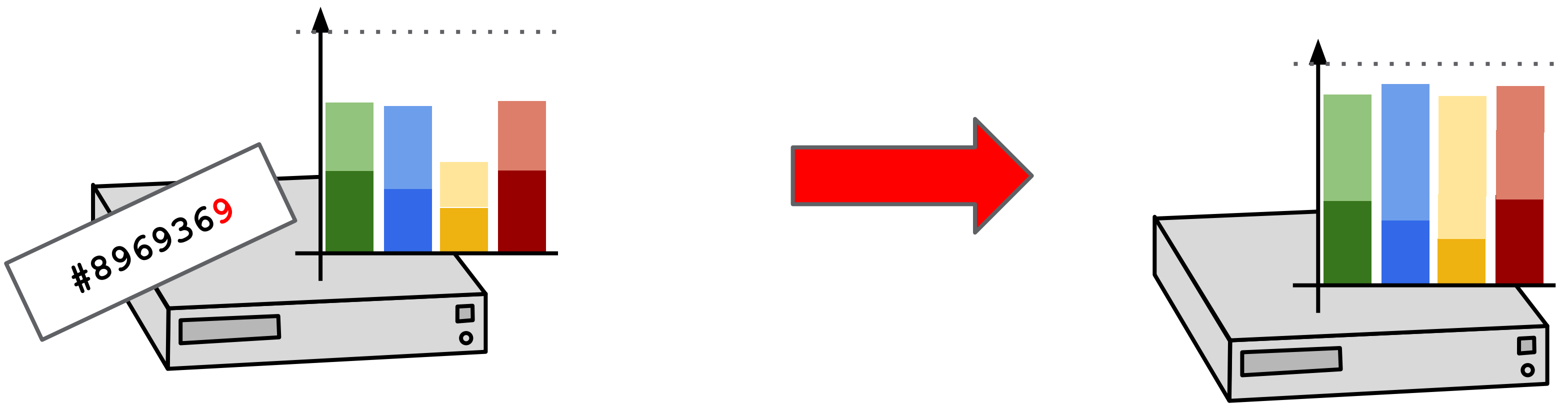


TAKEAWAY

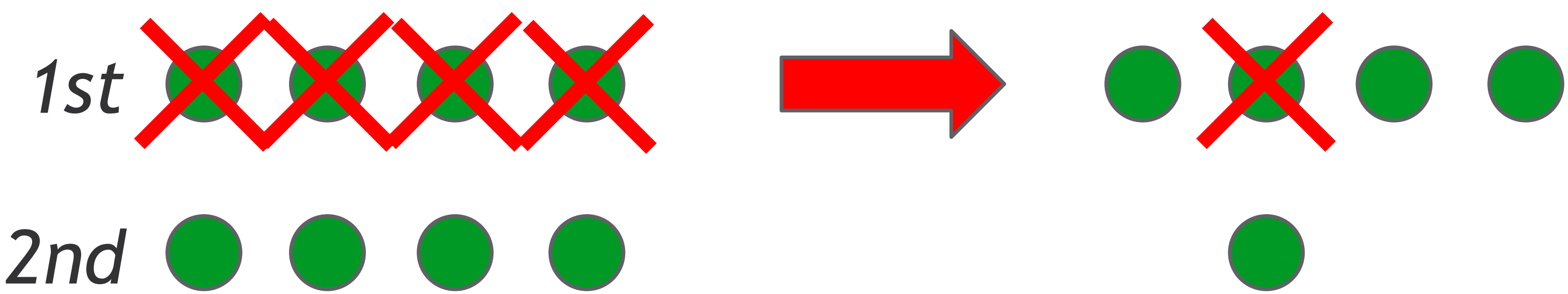
Interference is higher for real-world settings.



1. Fine-grained conflict detection



2. Incremental commits



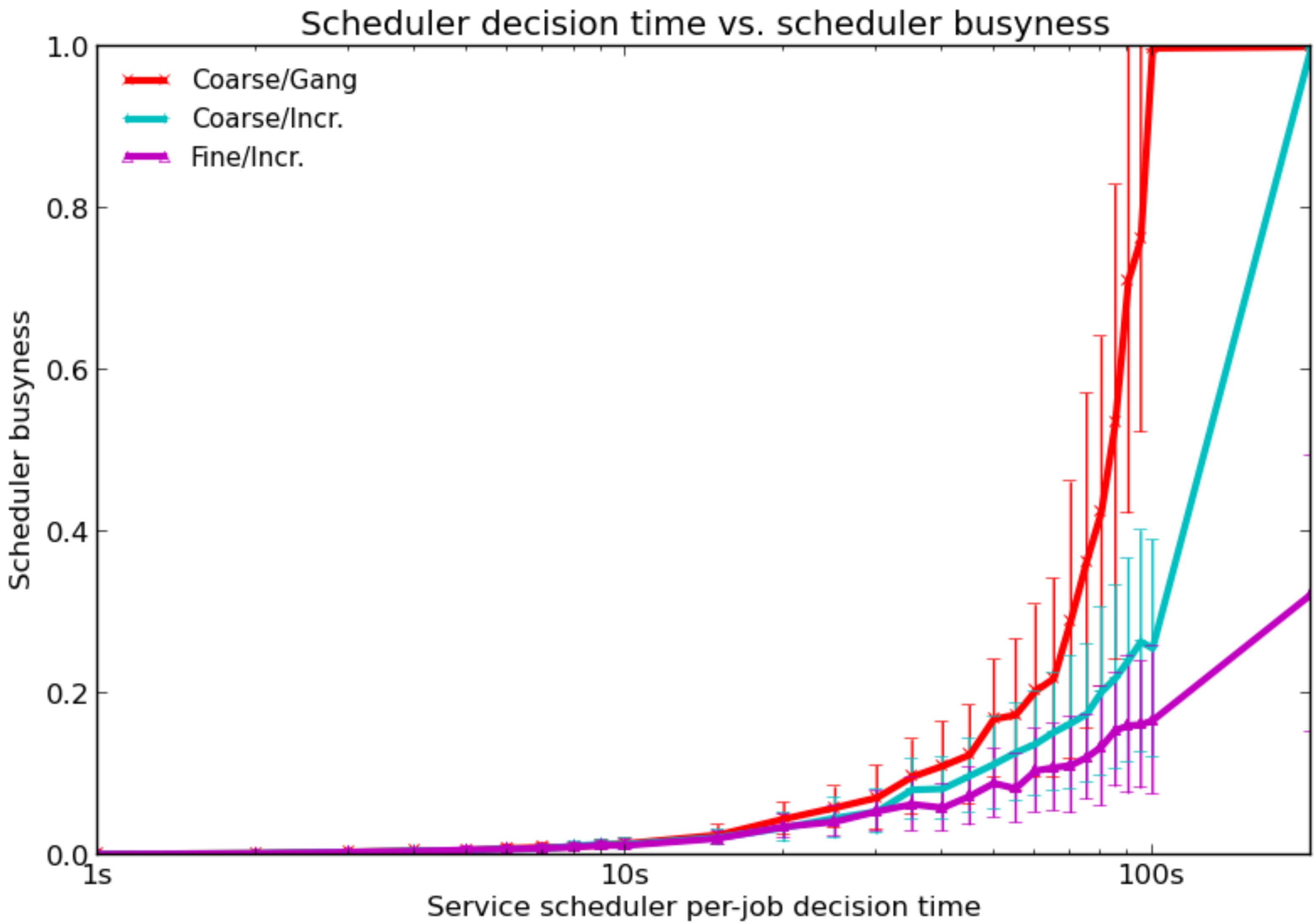
Experiment 4:

How do the optimizations affect performance?

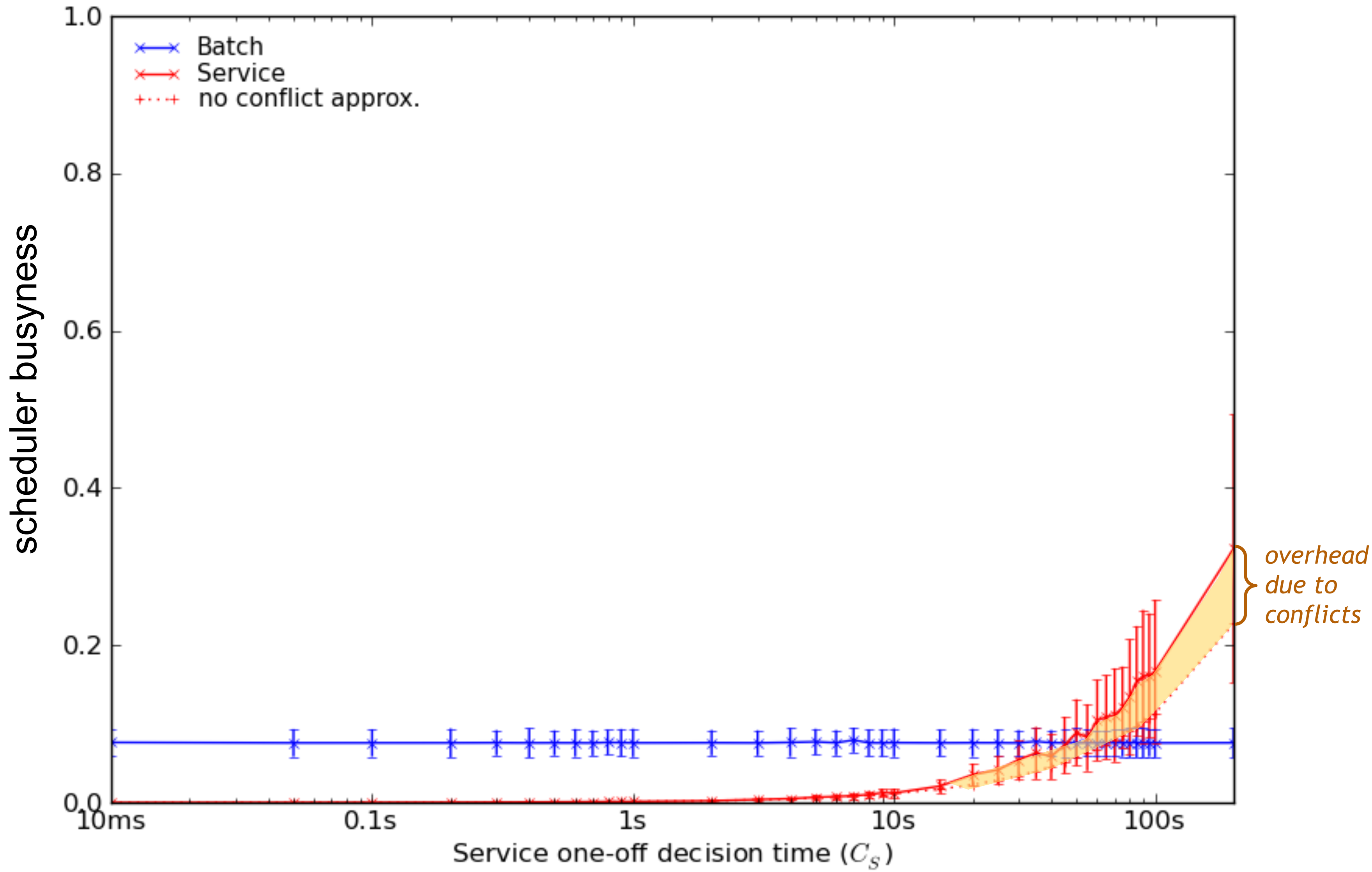
Experiment details:

- cluster C, 29 days
- 2 schedulers,
  non-uniform decision time
- varying **Service** scheduler

impact on scheduler utilization

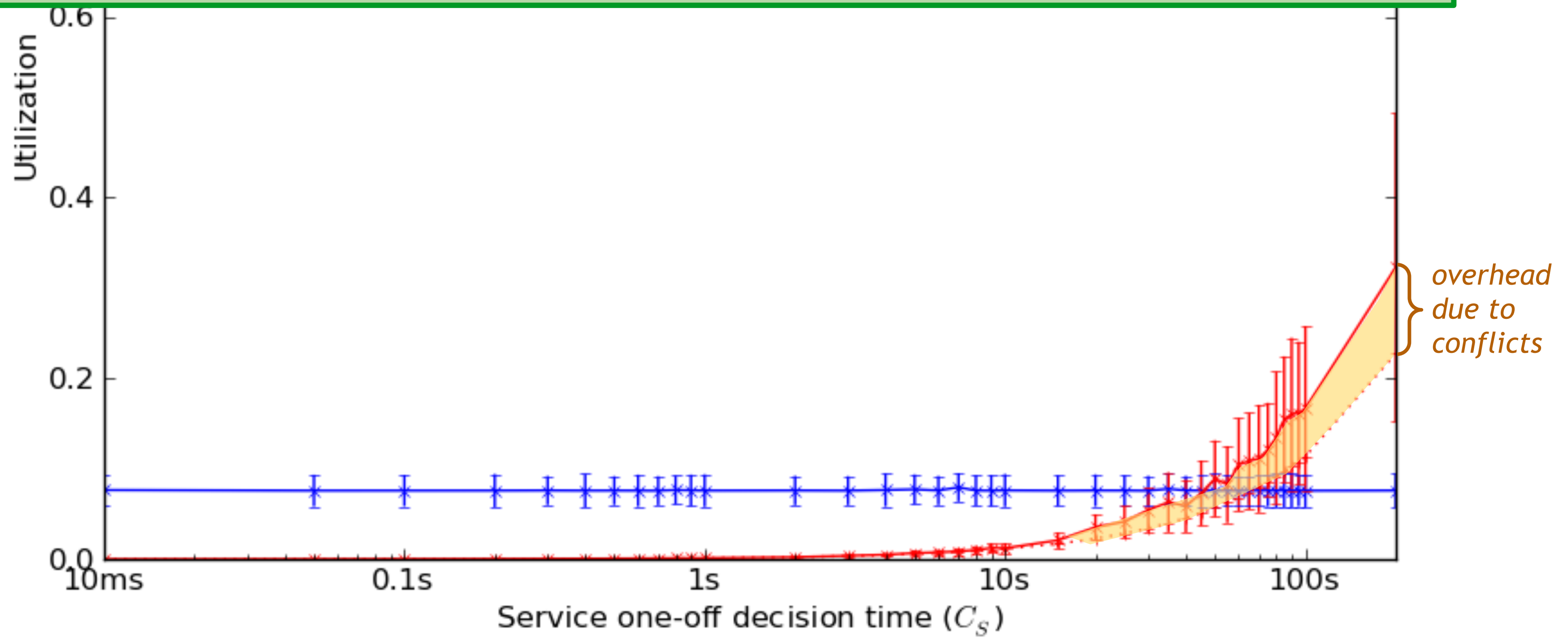


practical implications – scheduler utilization



TAKEAWAY

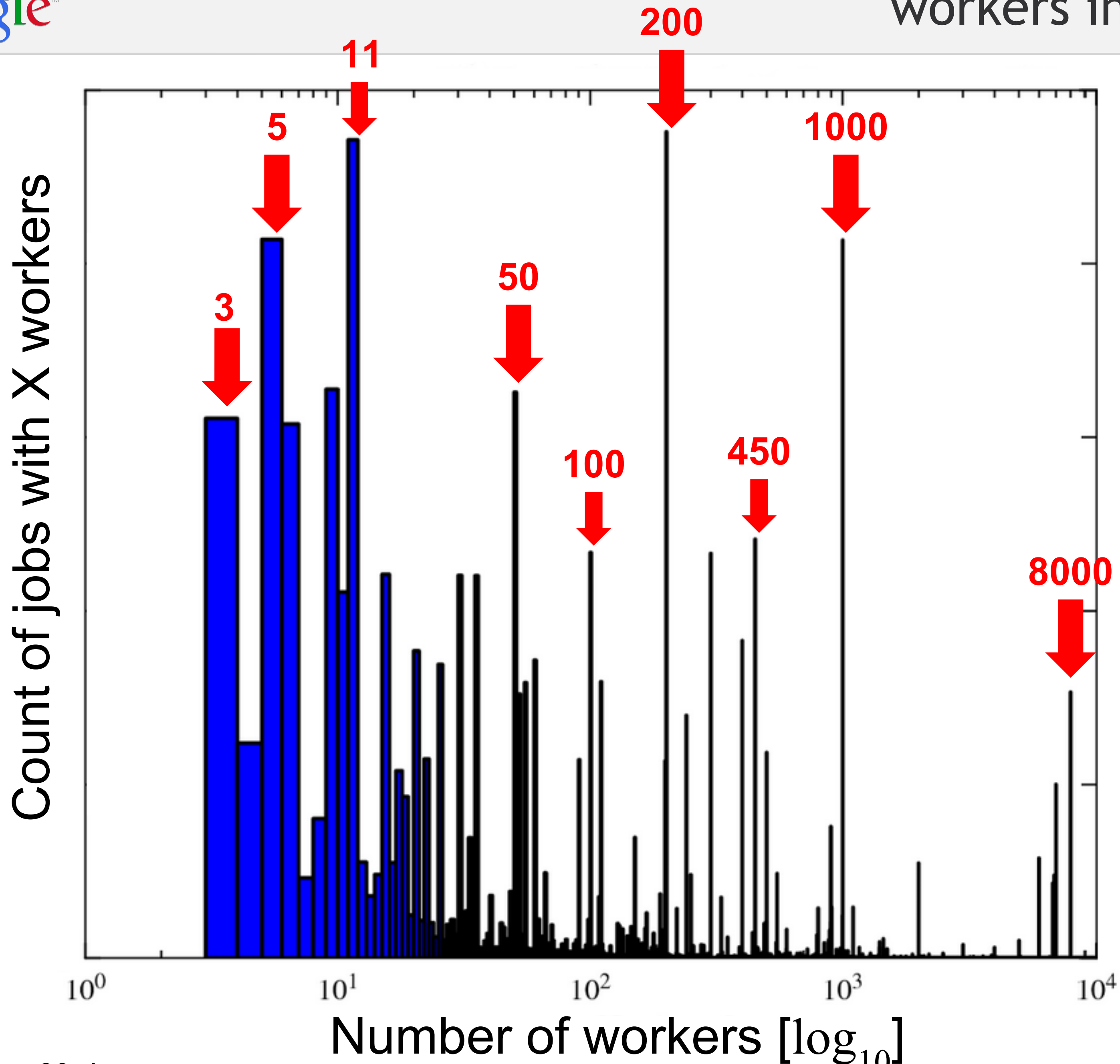
We can make simple improvements that significantly improve scalability.



Case study

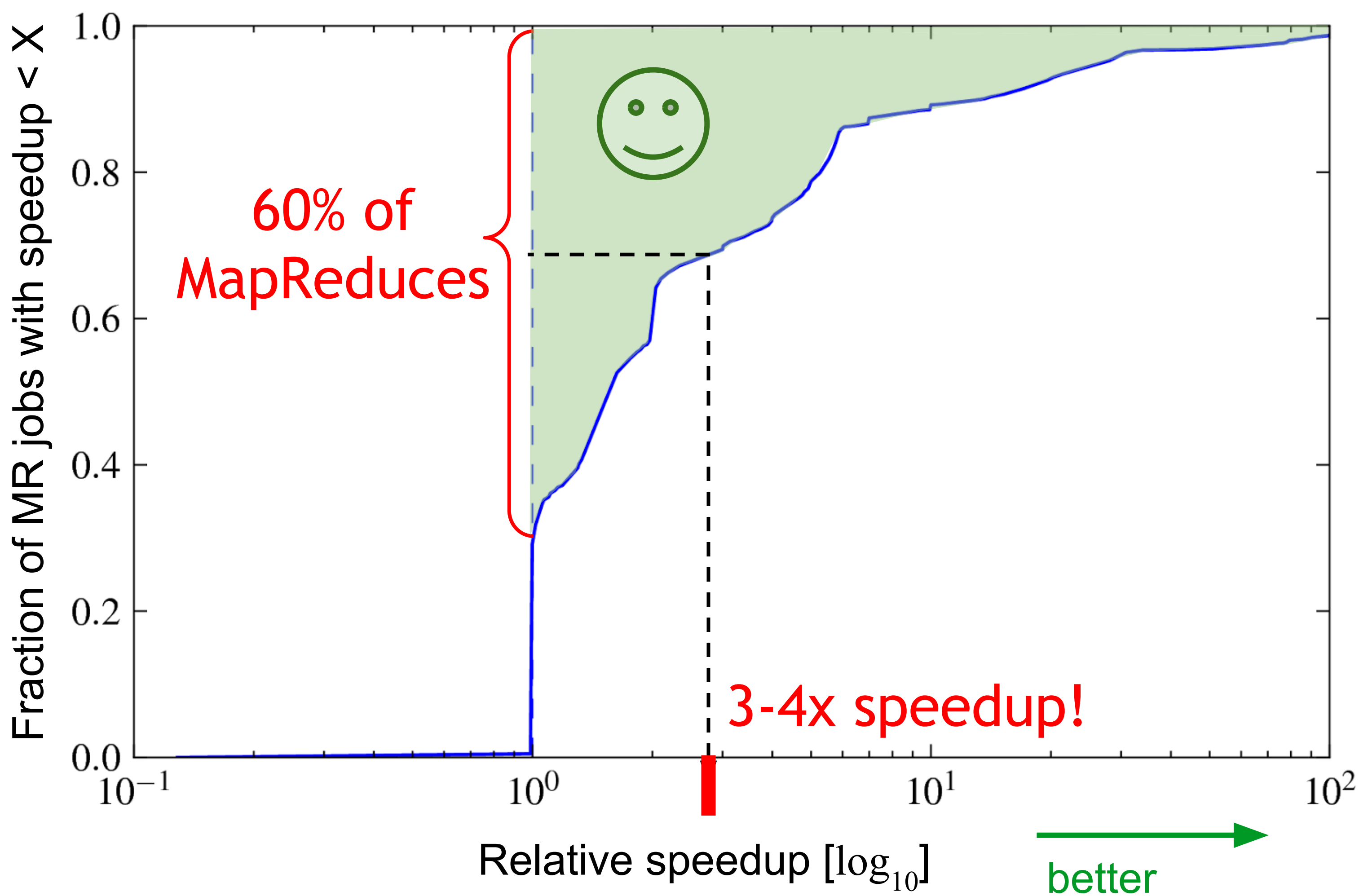
MapReduce scheduler with opportunistic extra resources

workers in MR jobs



Snapshot over 29 days

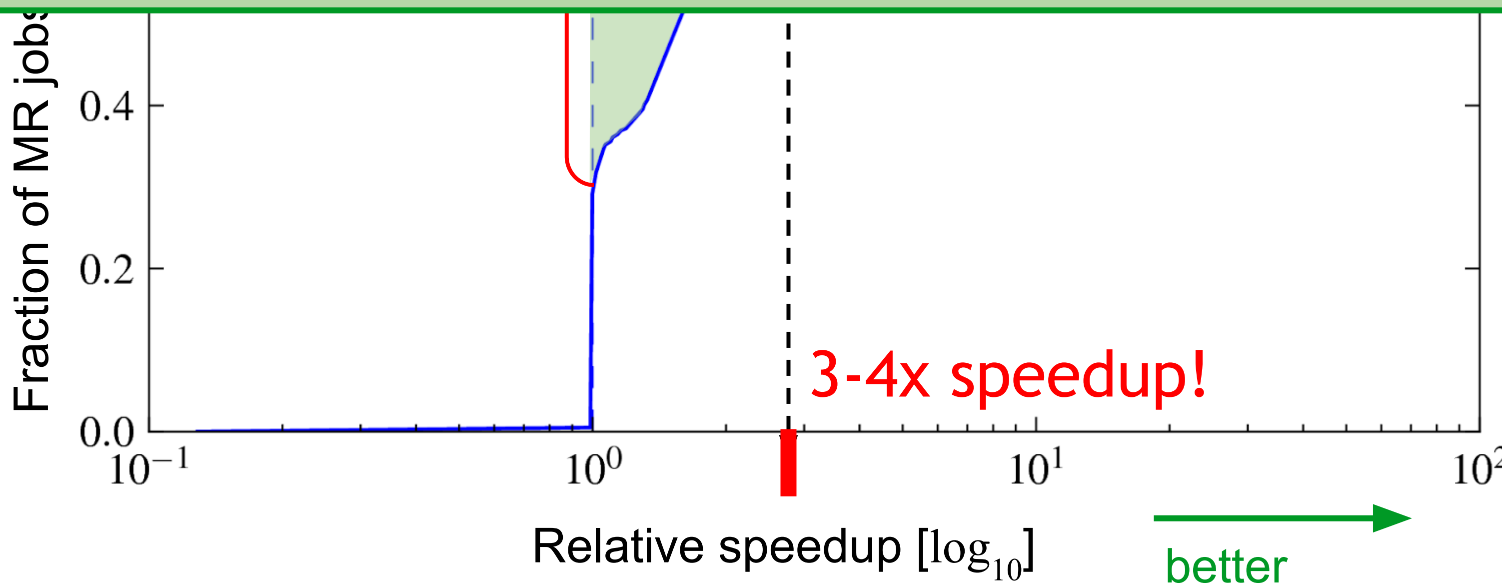
case study: a MapReduce scheduler



cluster C, 29 days

TAKEAWAY

The Omega approach gives us the flexibility to easily support custom policies.



TAKEAWAYS

Flexibility and scale require parallelism,

parallel scheduling works *if you do it right*, and

using shared state is the way to do it right!

BACKUP SLIDES

Apache Hadoop YARN: Yet Another Resource Negotiator. *ACM SoCC'13*

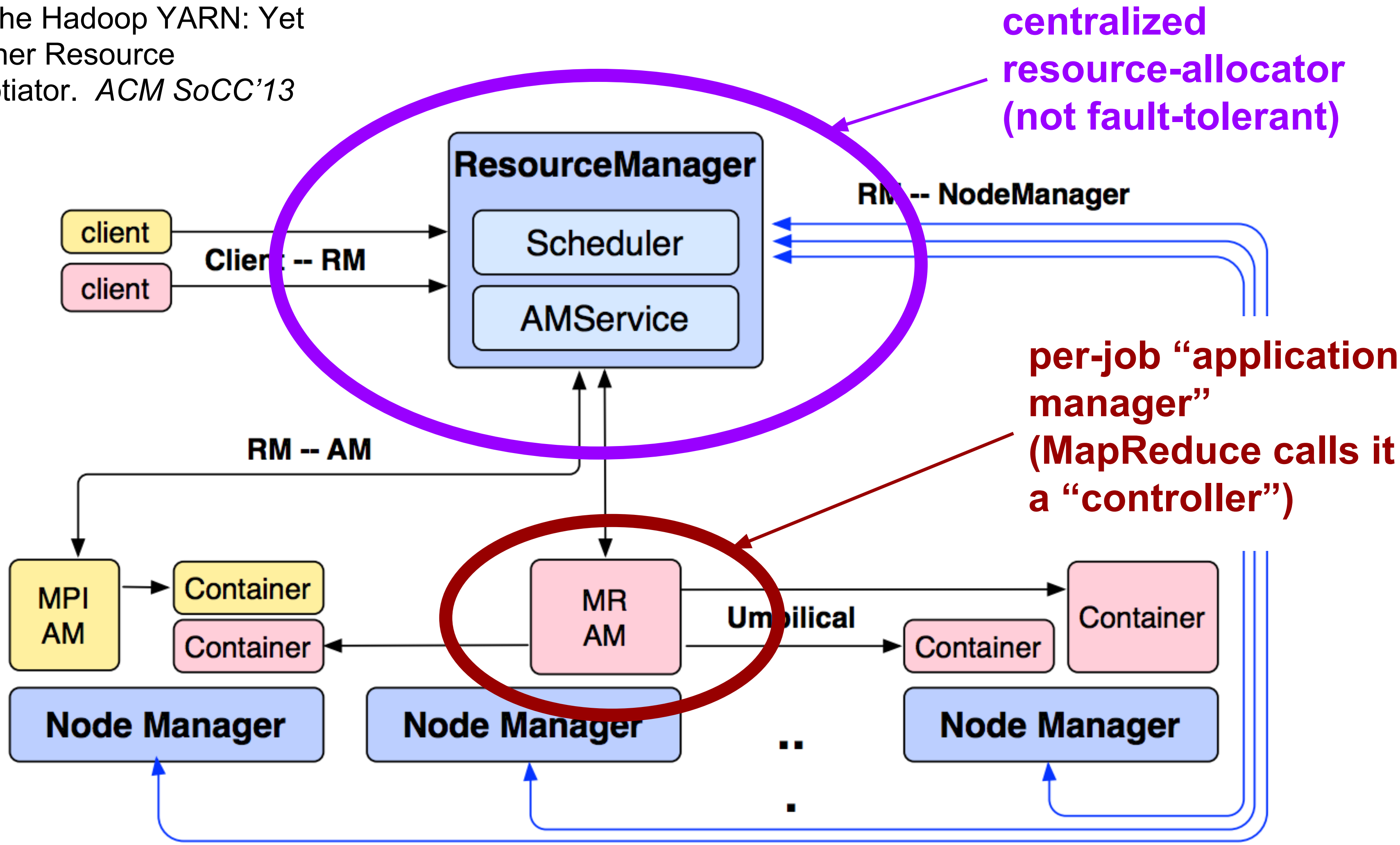
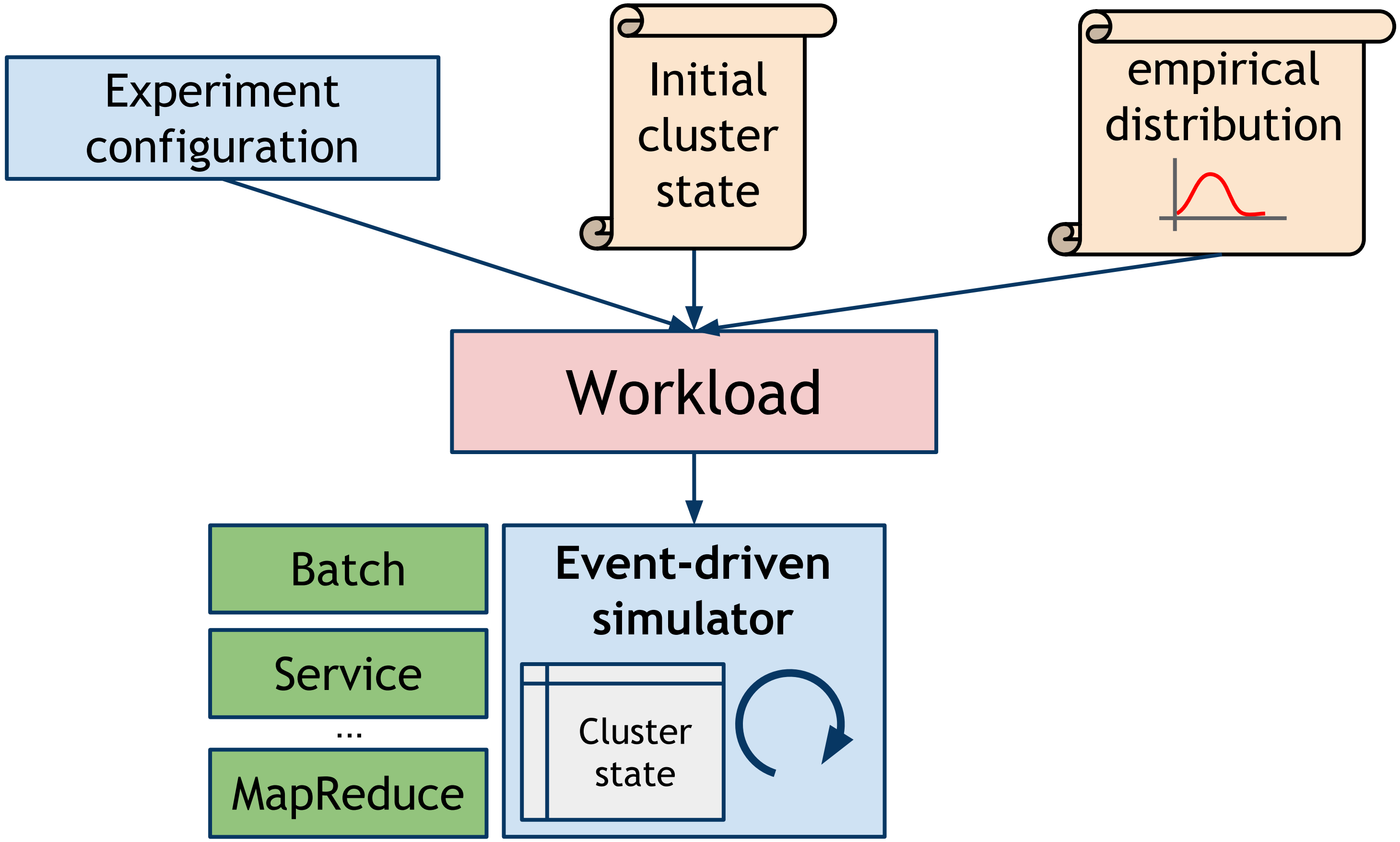


Figure 1: YARN Architecture (in blue the system components, and in yellow and pink two applications running.)

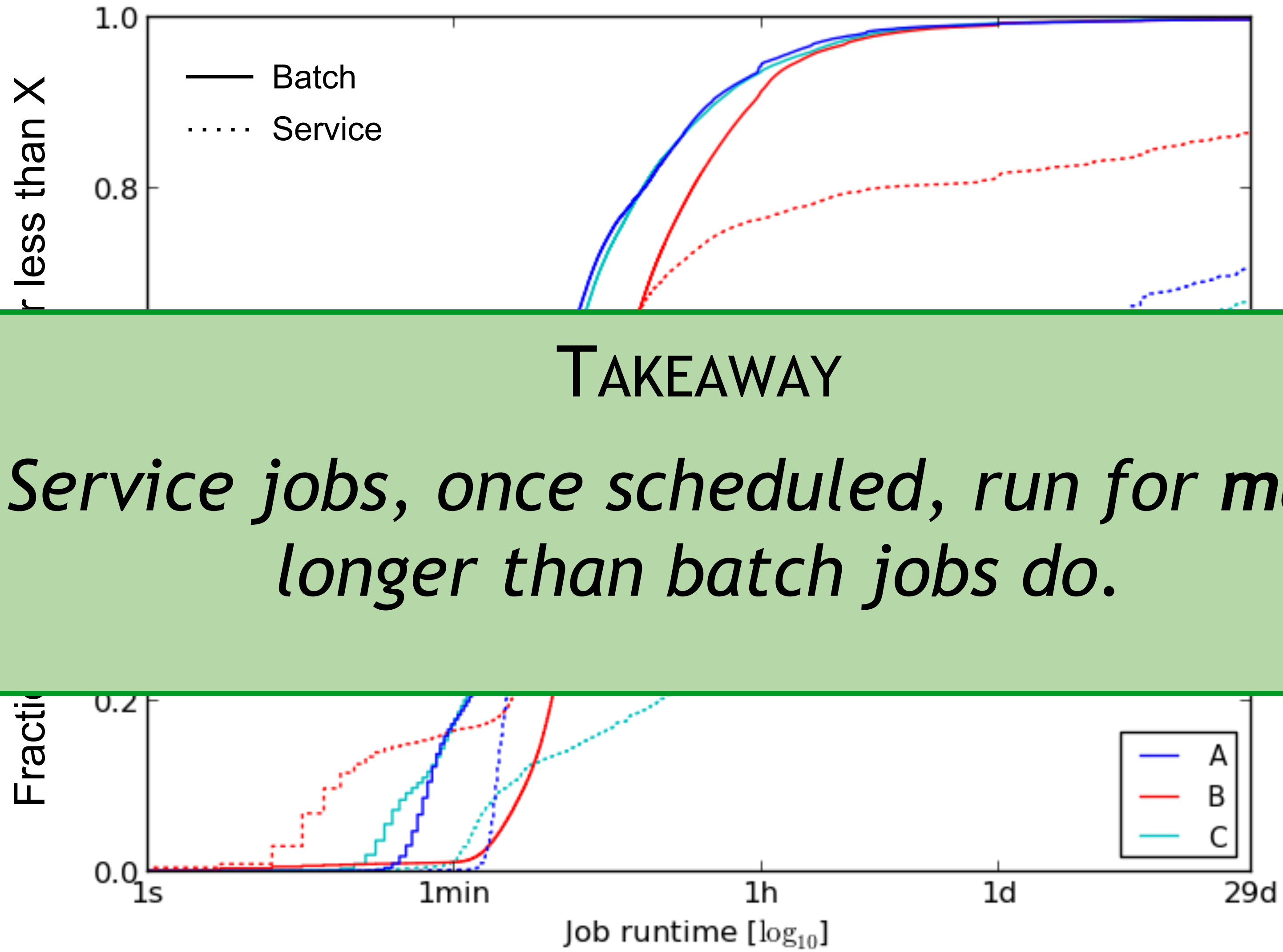
methodology: simulation



Code available:

`http://code.google.com/p/cluster-scheduler-simulator`

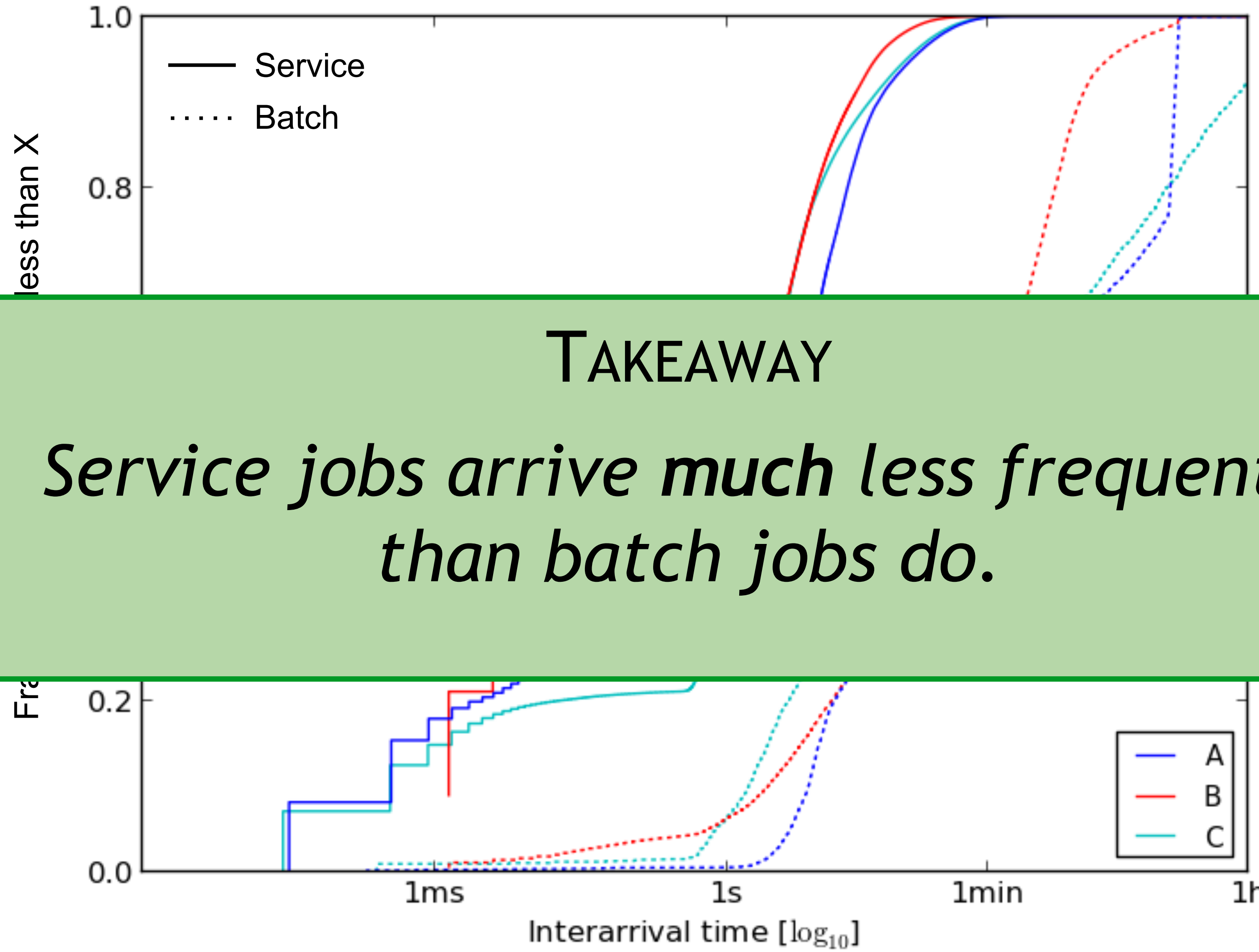
workload: job runtime distributions



TAKEAWAY

*Service jobs, once scheduled, run for **much** longer than batch jobs do.*

workload: inter-arrival time distributions



TAKEAWAY

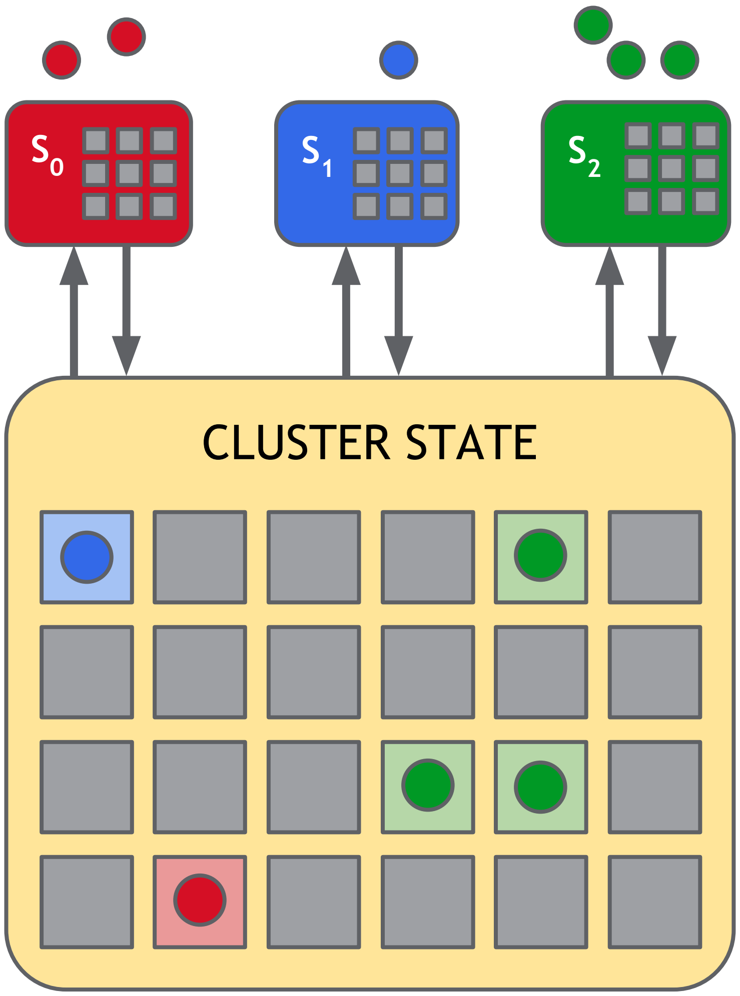
Service jobs arrive ***much*** *less frequently than batch jobs do.*

the omega approach

Shared state

Ωmega
Brave New Borg

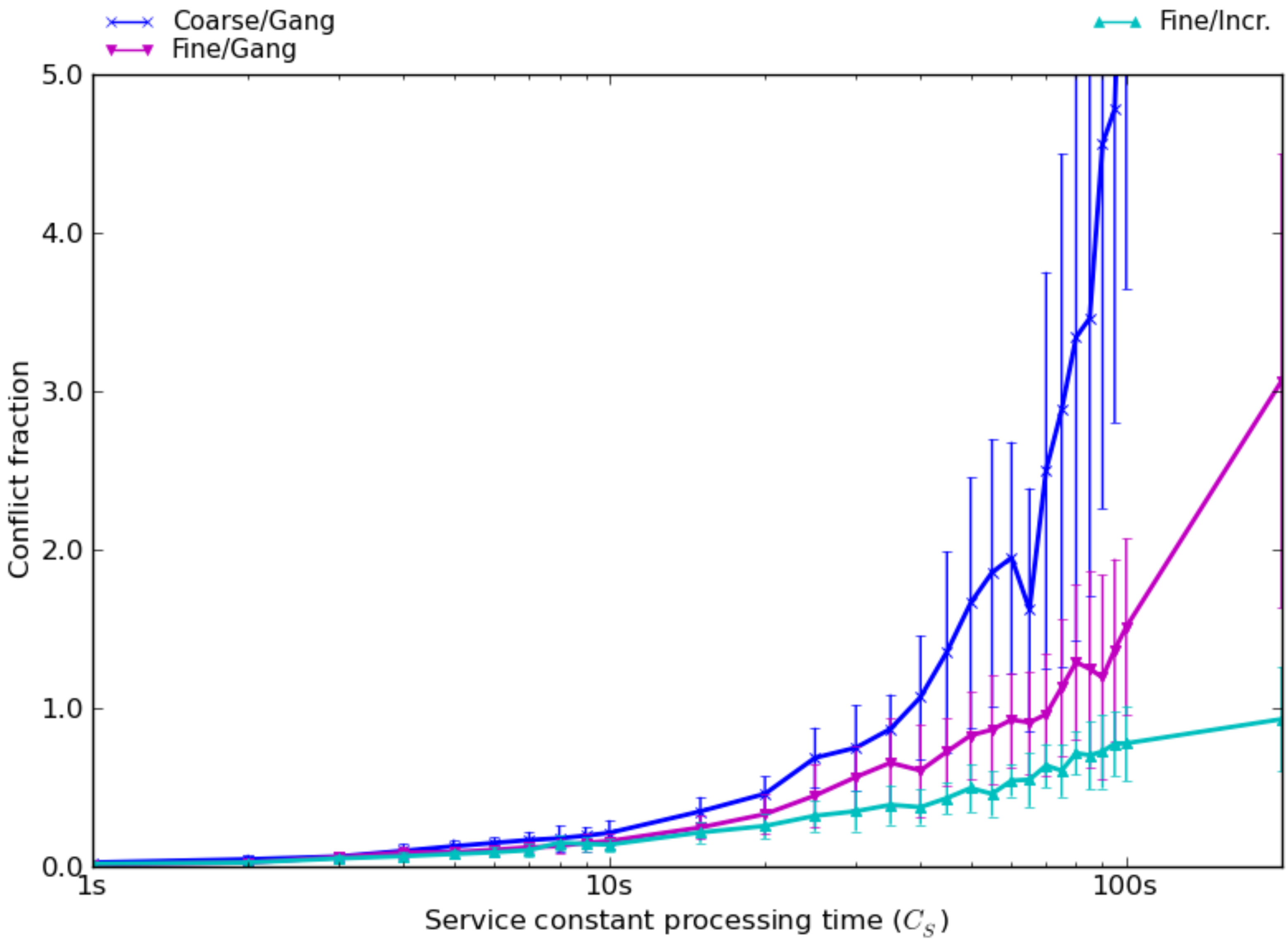
- Deltas against shared state
- Easy to develop & maintain
- Heterogeneous schedulers OK



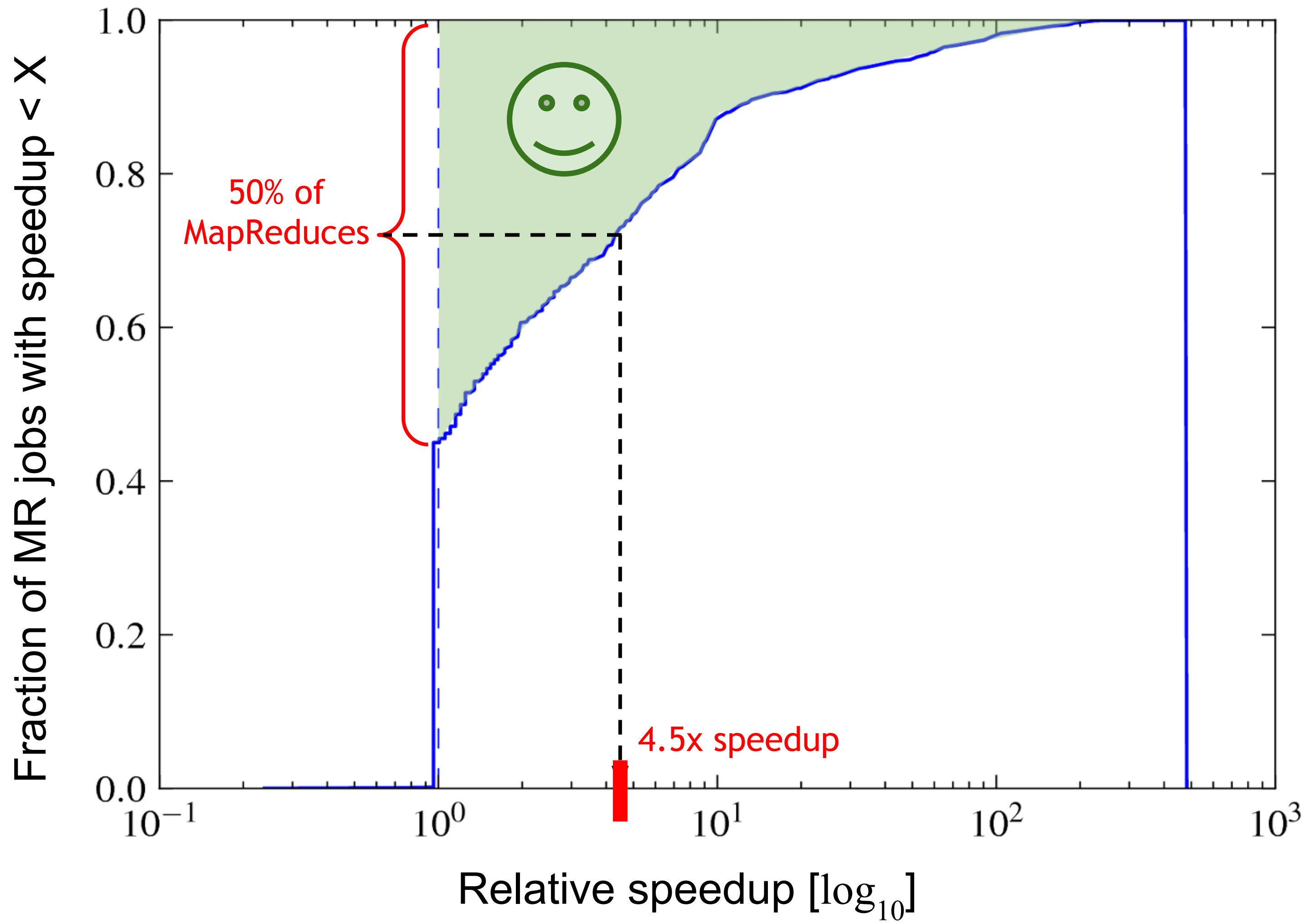
Optimistic concurrency

- No explicit coordination required
- Interference resolution (not prevention)
- Scales well

impact on conflict fraction



case study: a MapReduce scheduler



cluster A, 29 days

Possible problems...

- aggressive, systematically adverse workloads or schedulers
- small clusters with high overcommit

➡ **deal with using out-of-band or post-facto enforcement mechanisms**