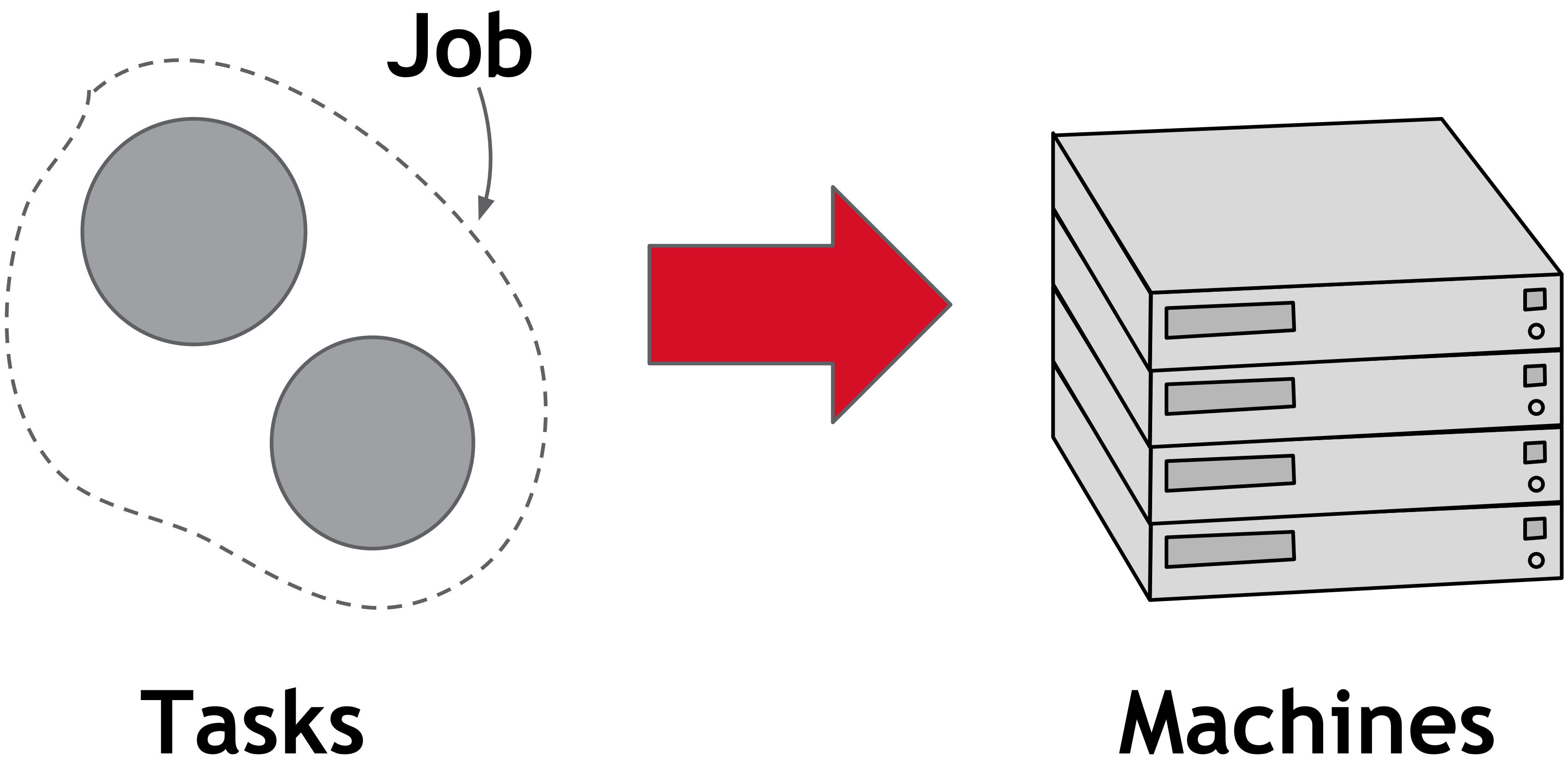
Omega: flexible, scalable schedulers for large compute clusters

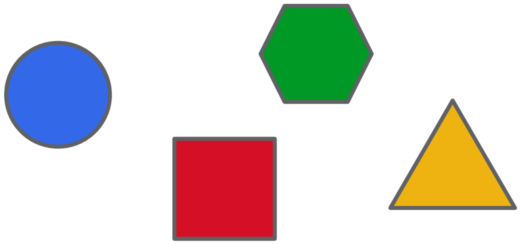
Malte Schwarzkopf (University of Cambridge Computer Lab)

Andy Konwinski (UC Berkeley)

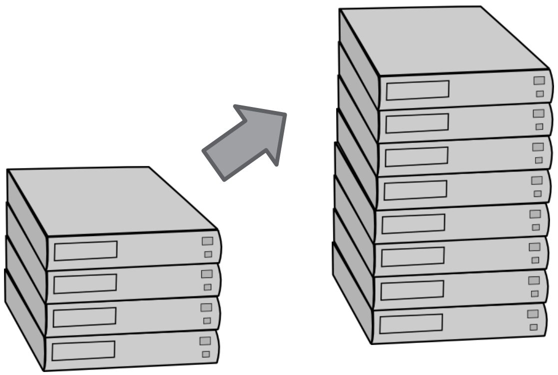
Michael Abd-El-Malek (Google)

John Wilkes (Google)

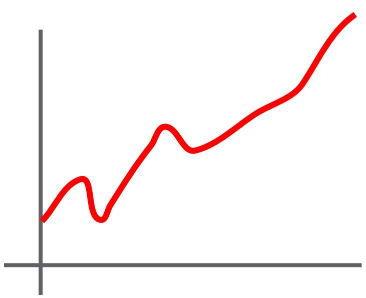

Job
Tasks
Machines



Diverse workloads

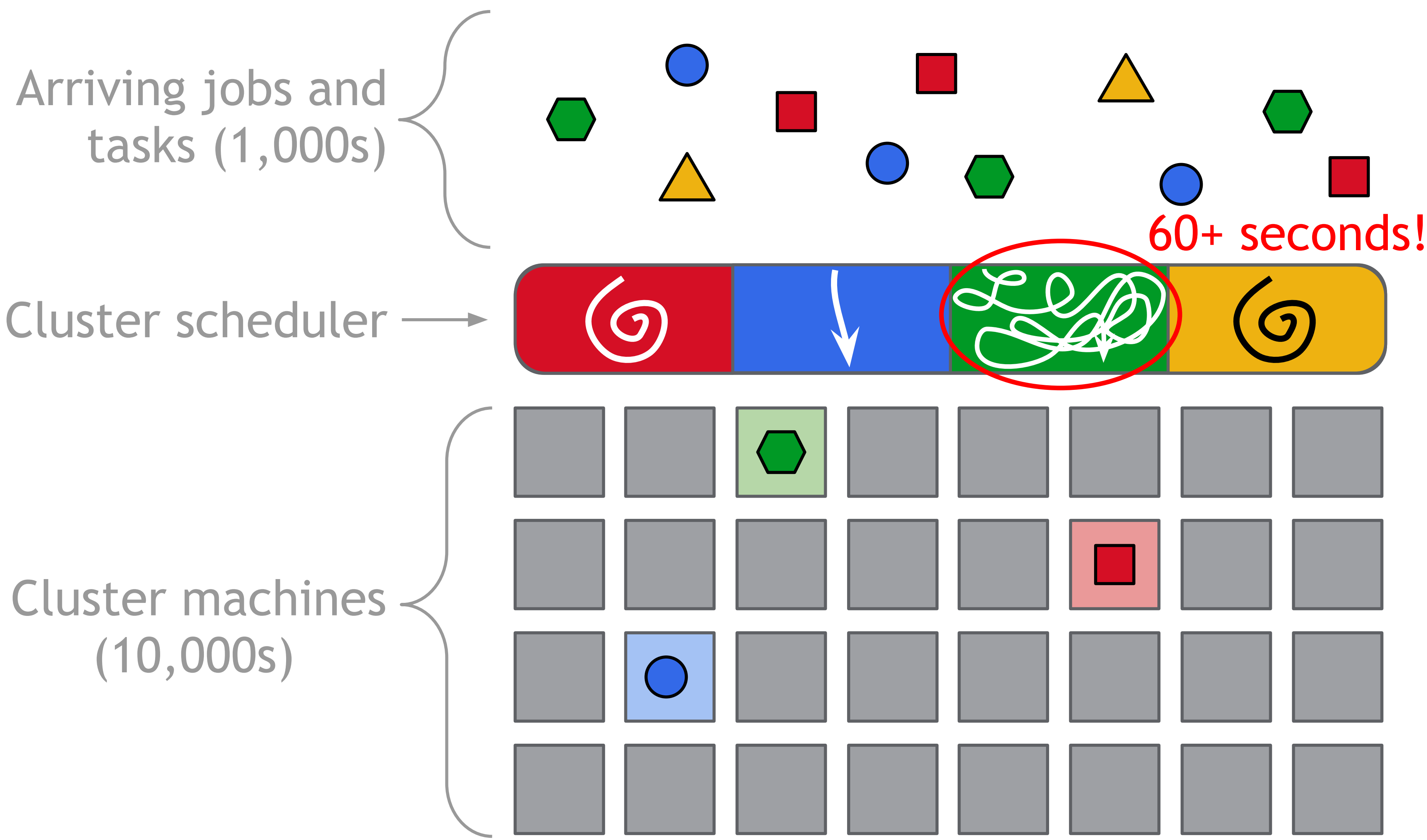


Increasing cluster sizes



Growing job arrival rates

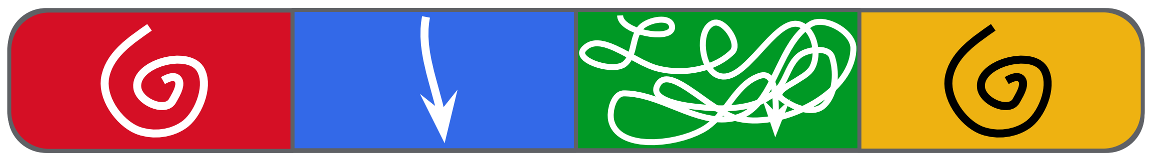
why is this a problem?



Arriving jobs and tasks (1,000s)

Increasing complexity!

Cluster scheduler

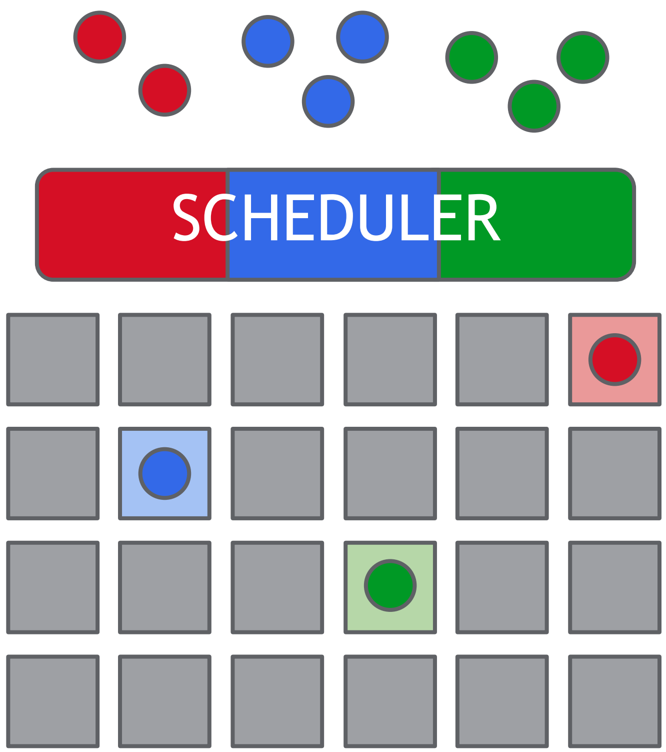


Cluste
(10

Hence:
Break up into independent schedulers.

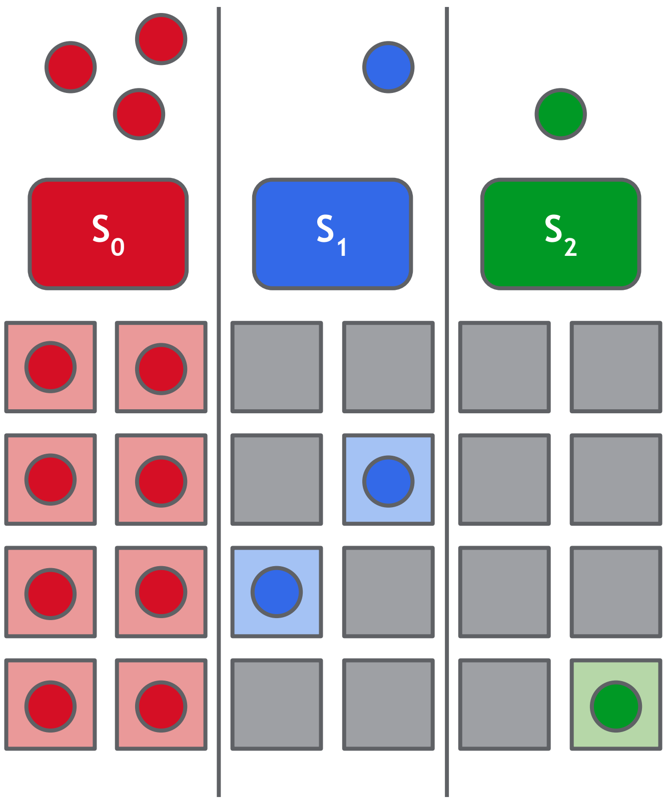
But: How do we arbitrate resources between schedulers?

monolithic scheduler



- hard to diversify
- code growth
- scalability bottleneck

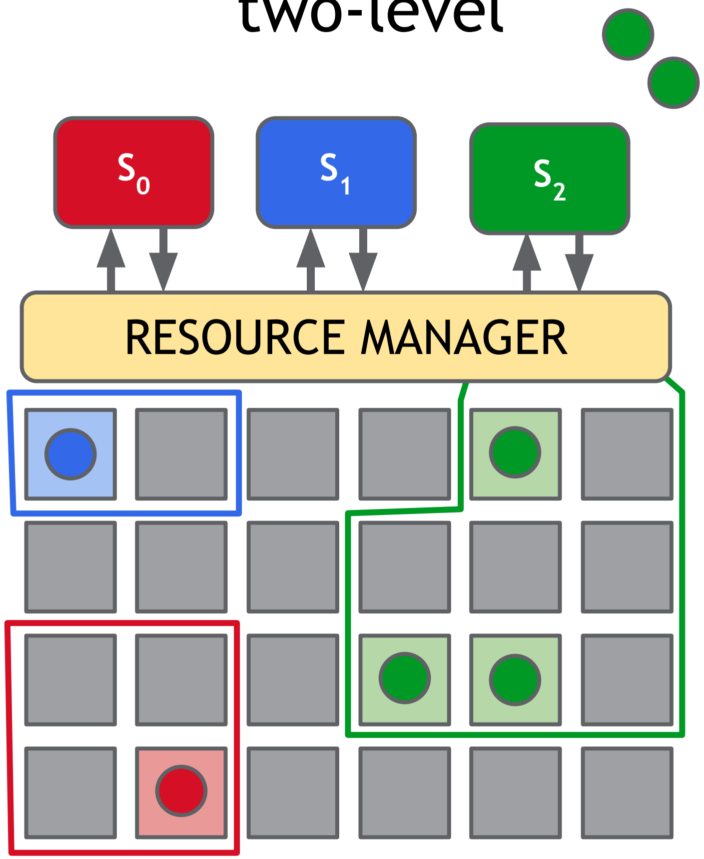
static partitioning



- poor utilization
- inflexible

existing approaches

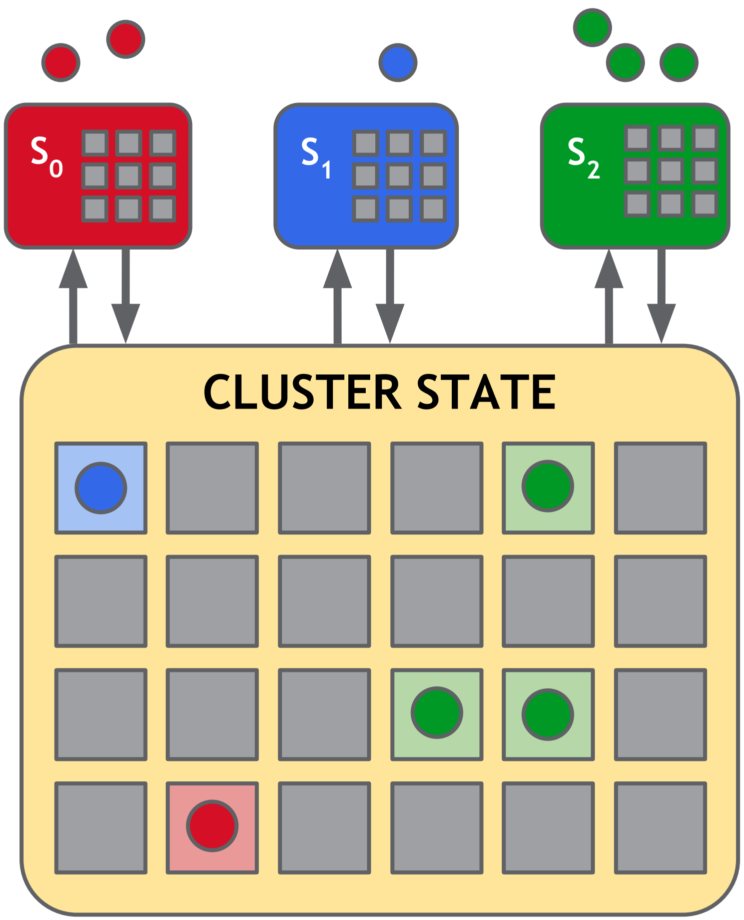
two-level



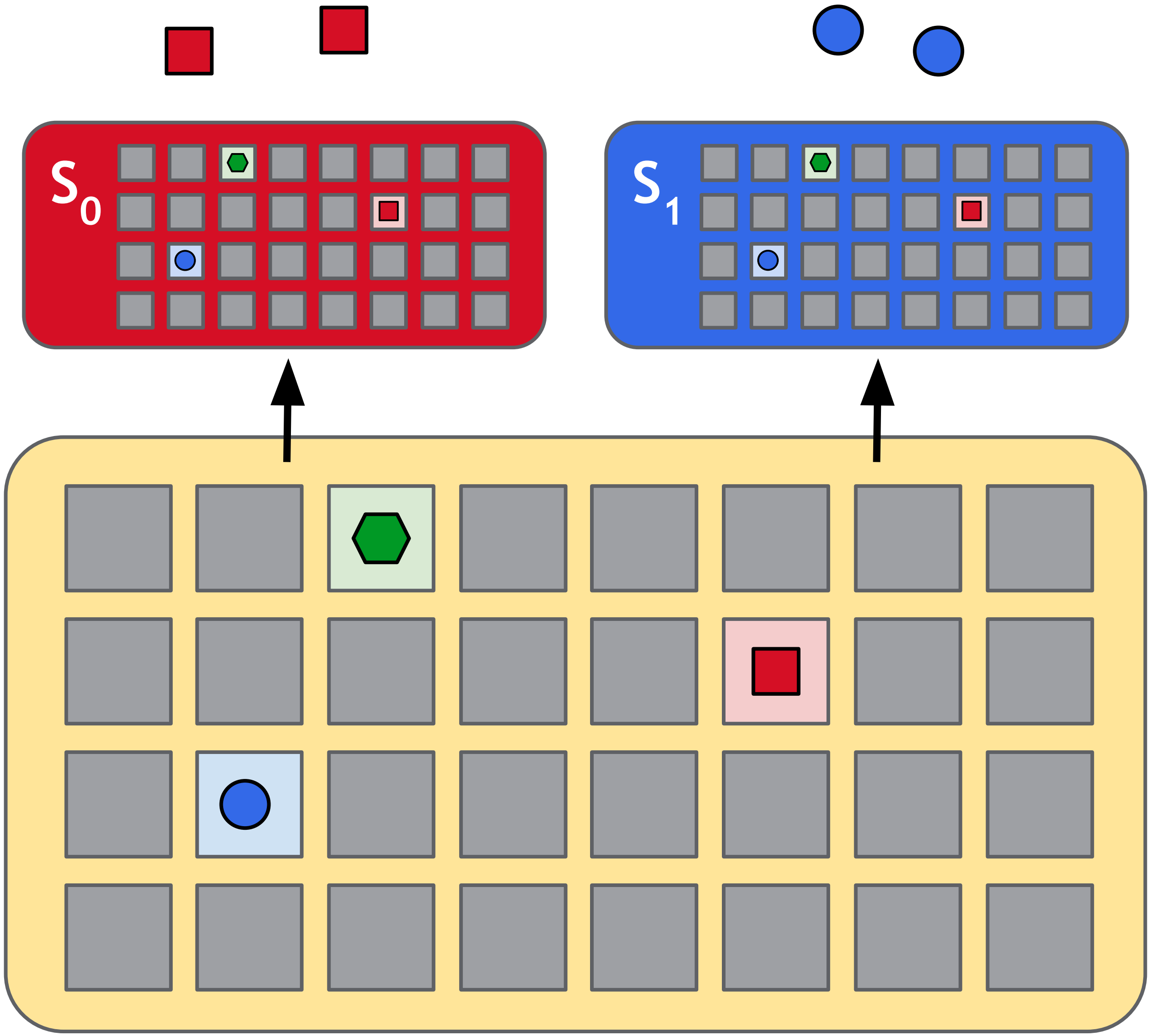
- hoarding
- information hiding

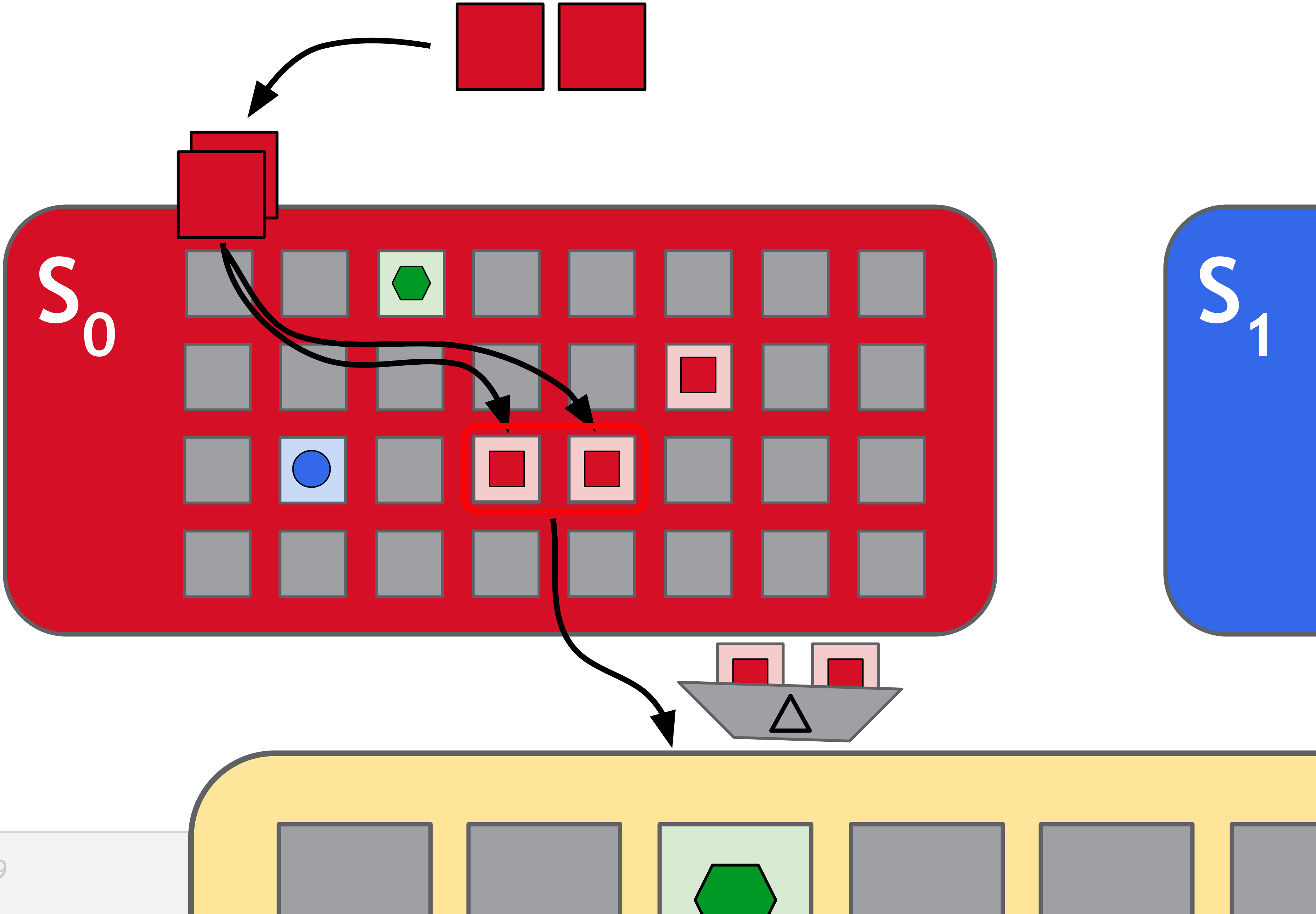
e.g. UCB Mesos [NSDI 2011]

shared-state



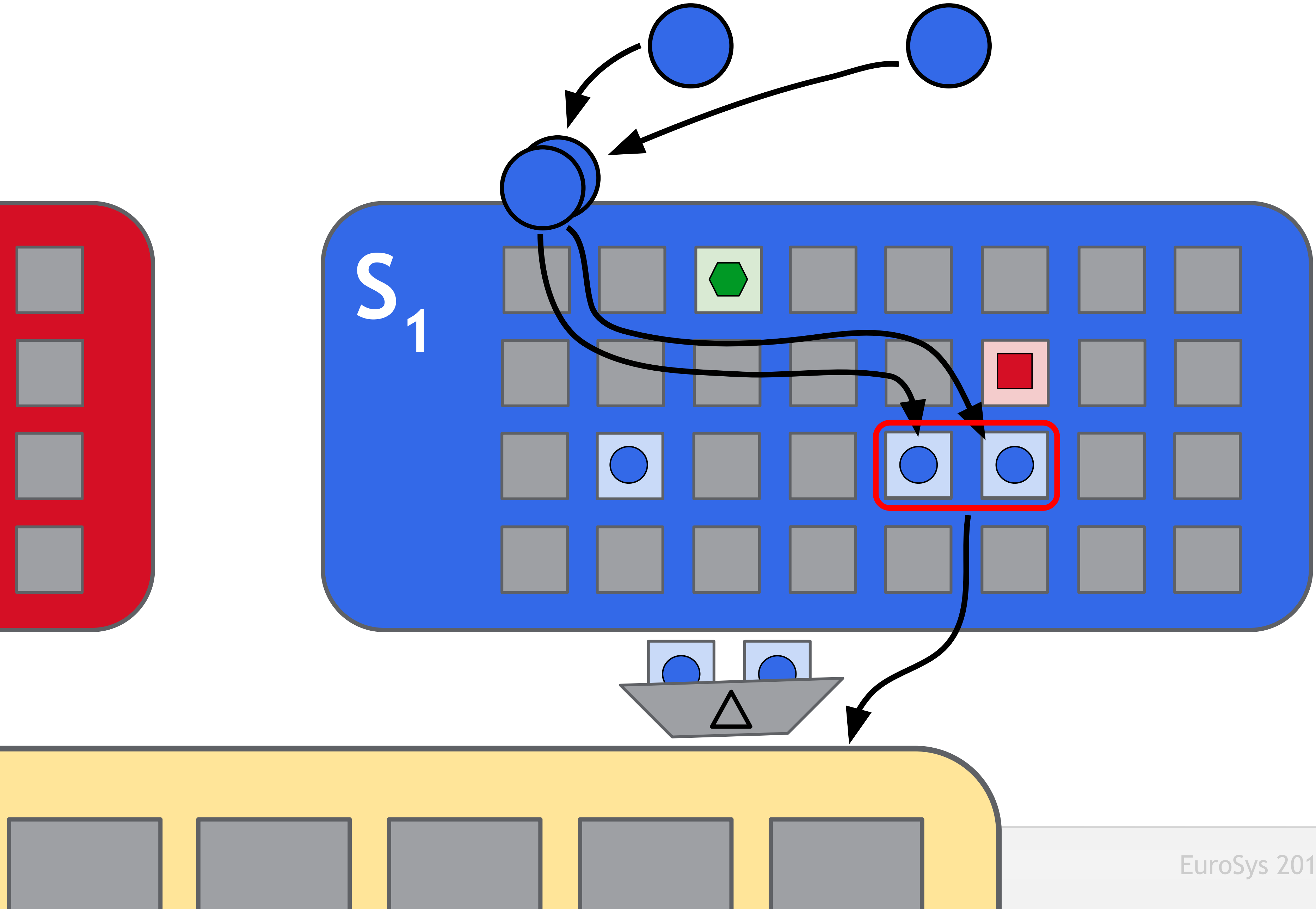
how does omega work?



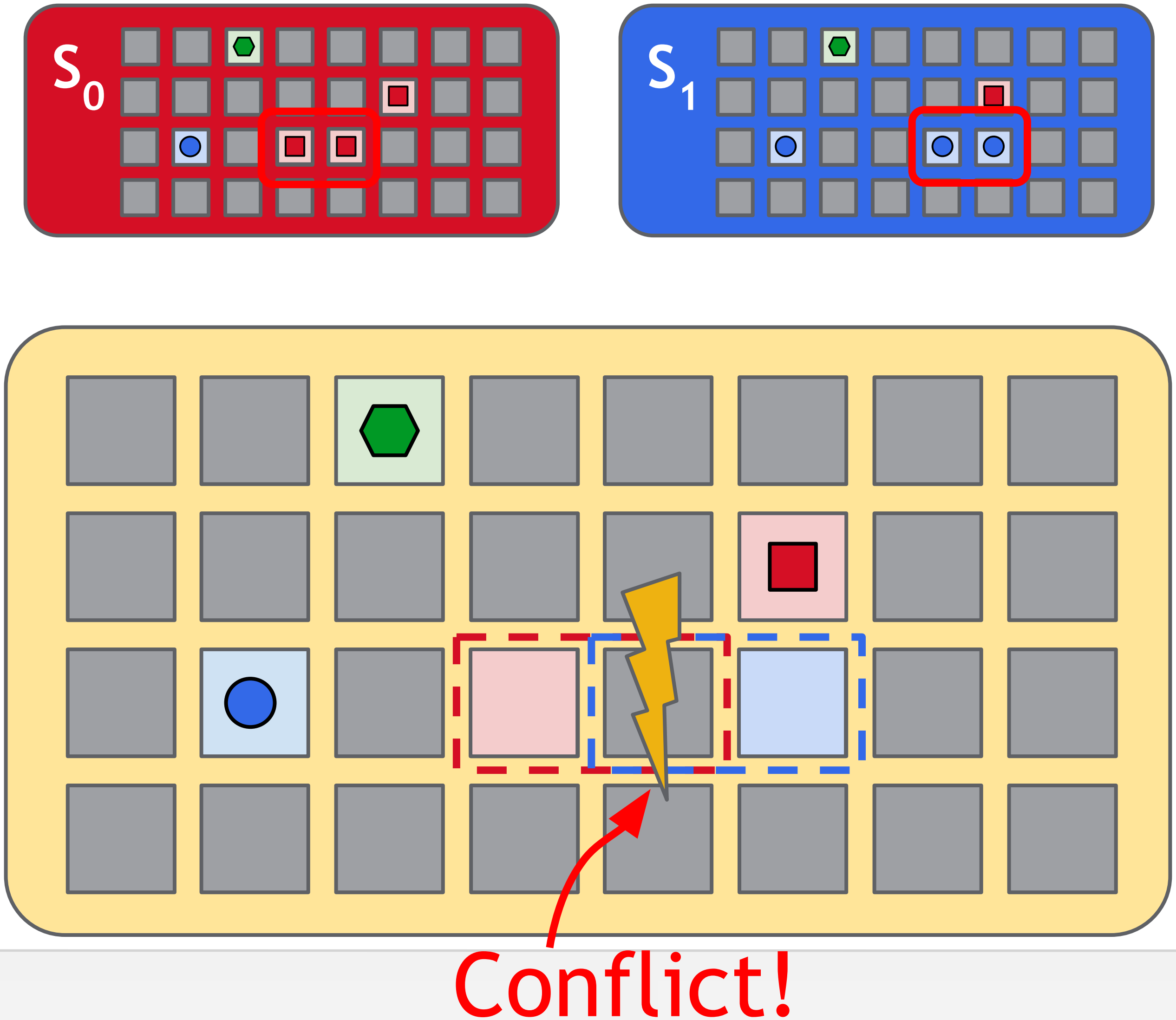


S_0
S_1

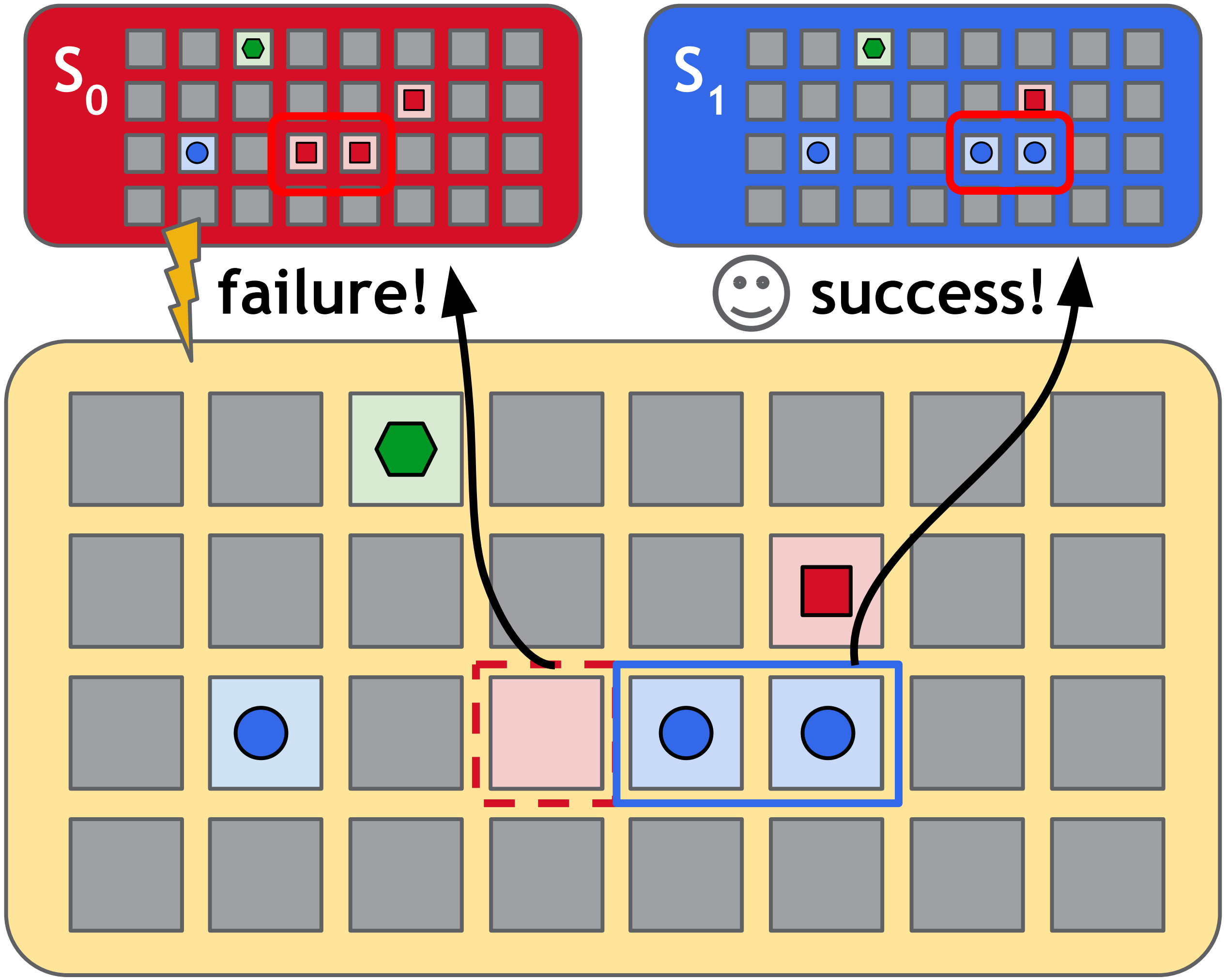
how does omega work?



how does omega work?



how does omega work?



1) intro & motivation

2) workload characterization

3) comparison of approaches

4) trace-based simulation

5) flexibility case study

Batch

Service

Cluster A

Medium size

Medium utilization

Cluster B

Large size

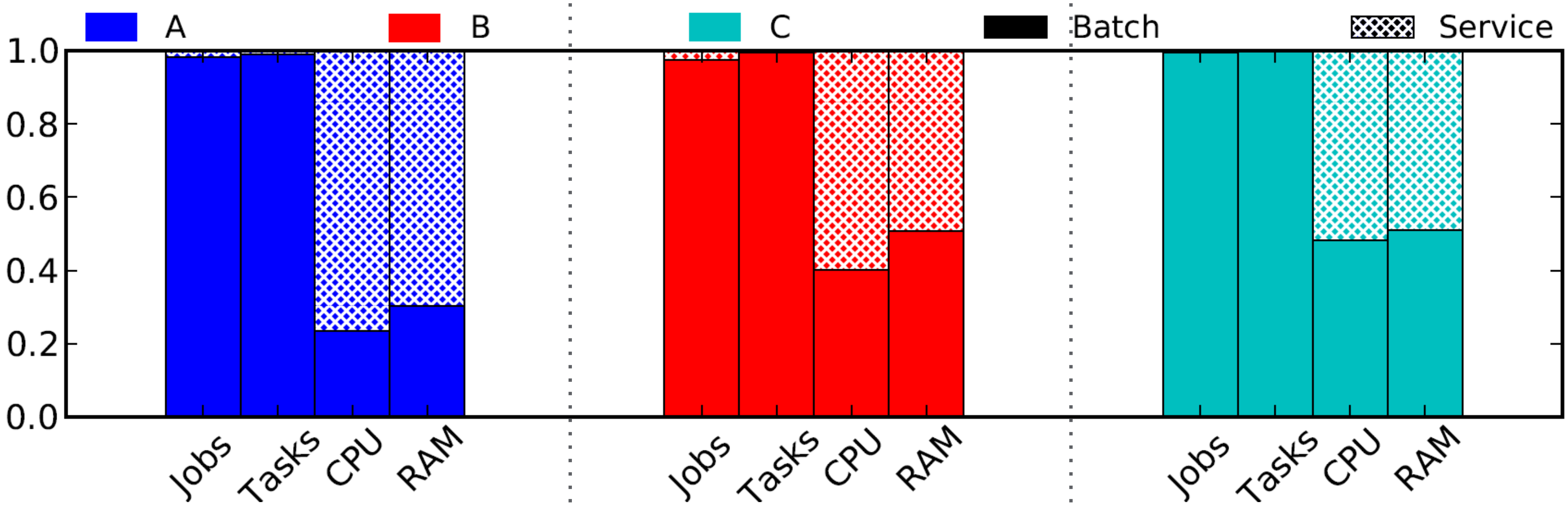
Medium utilization

Cluster C

Medium (12k mach.)

High utilization

Public trace



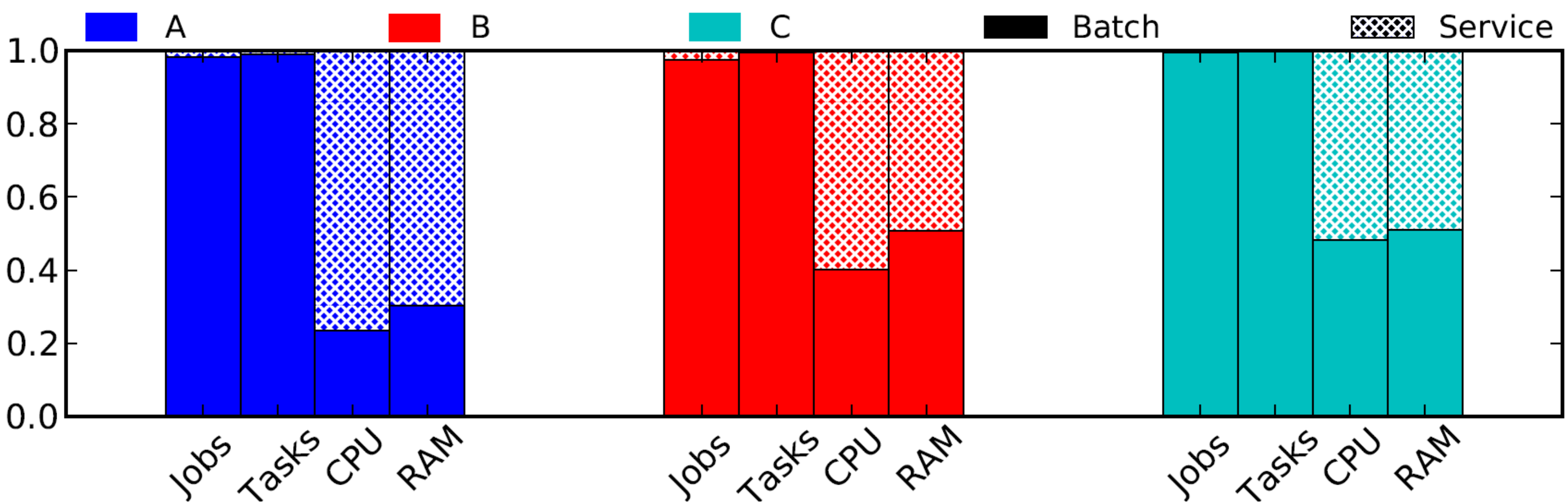
Jobs/tasks: counts
CPU/RAM: resource seconds [i.e. resource job runtime in sec.]

Google

workload: batch/service split

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.]

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

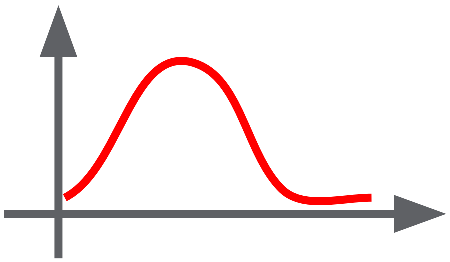
3) comparison of approaches

4) trace-based simulation

5) flexibility case study

simulation using

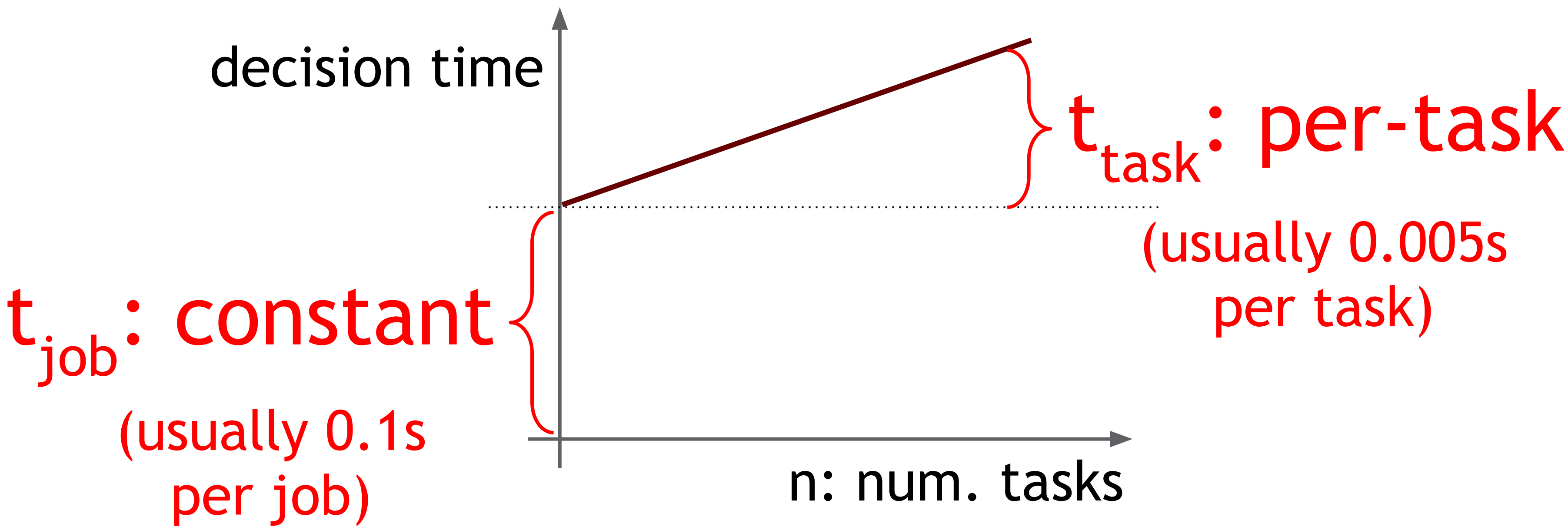
empirical workload parameters distributions



Code [soon to be] **available:**

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

Scheduler decision time



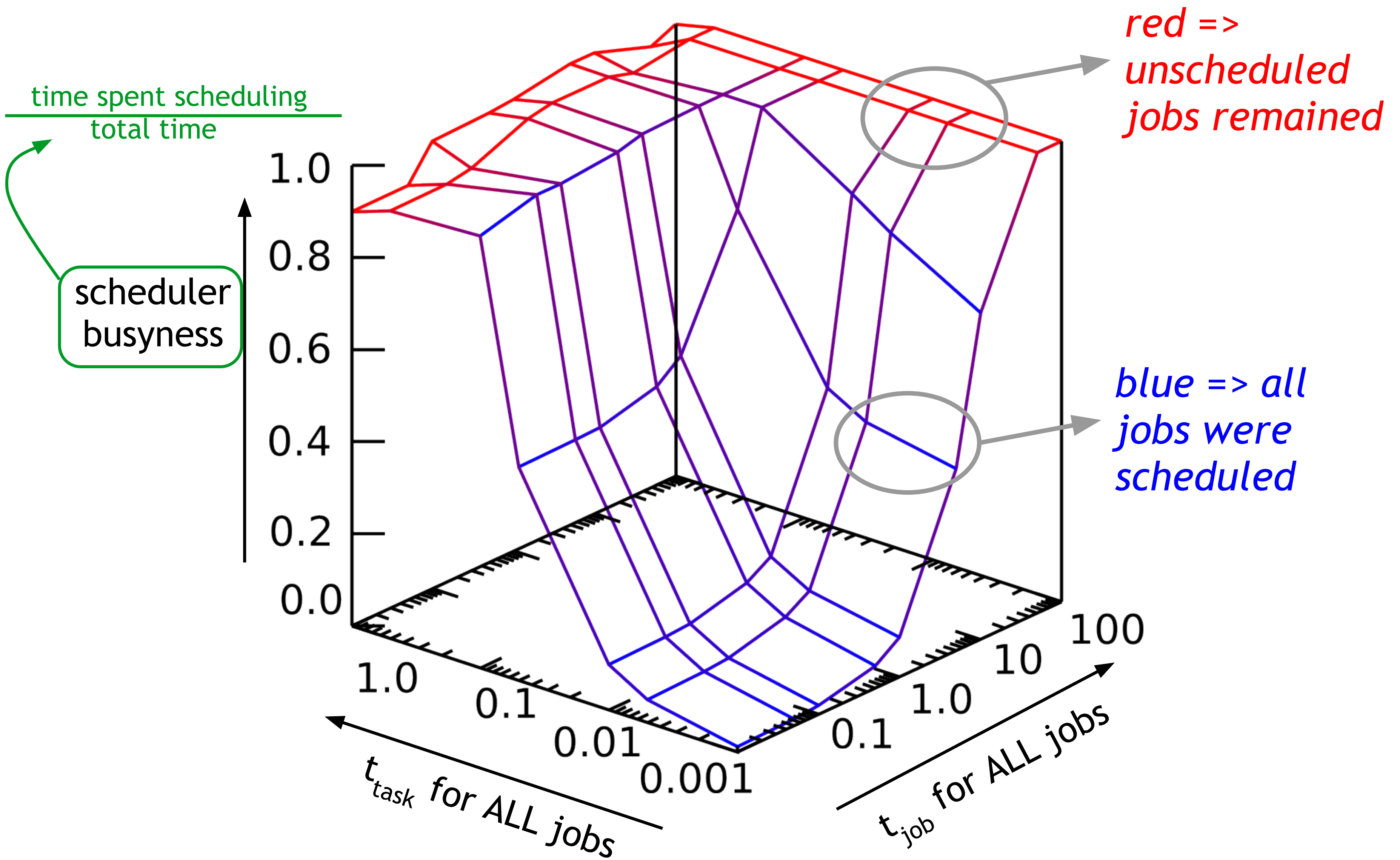
Experiment 1:

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

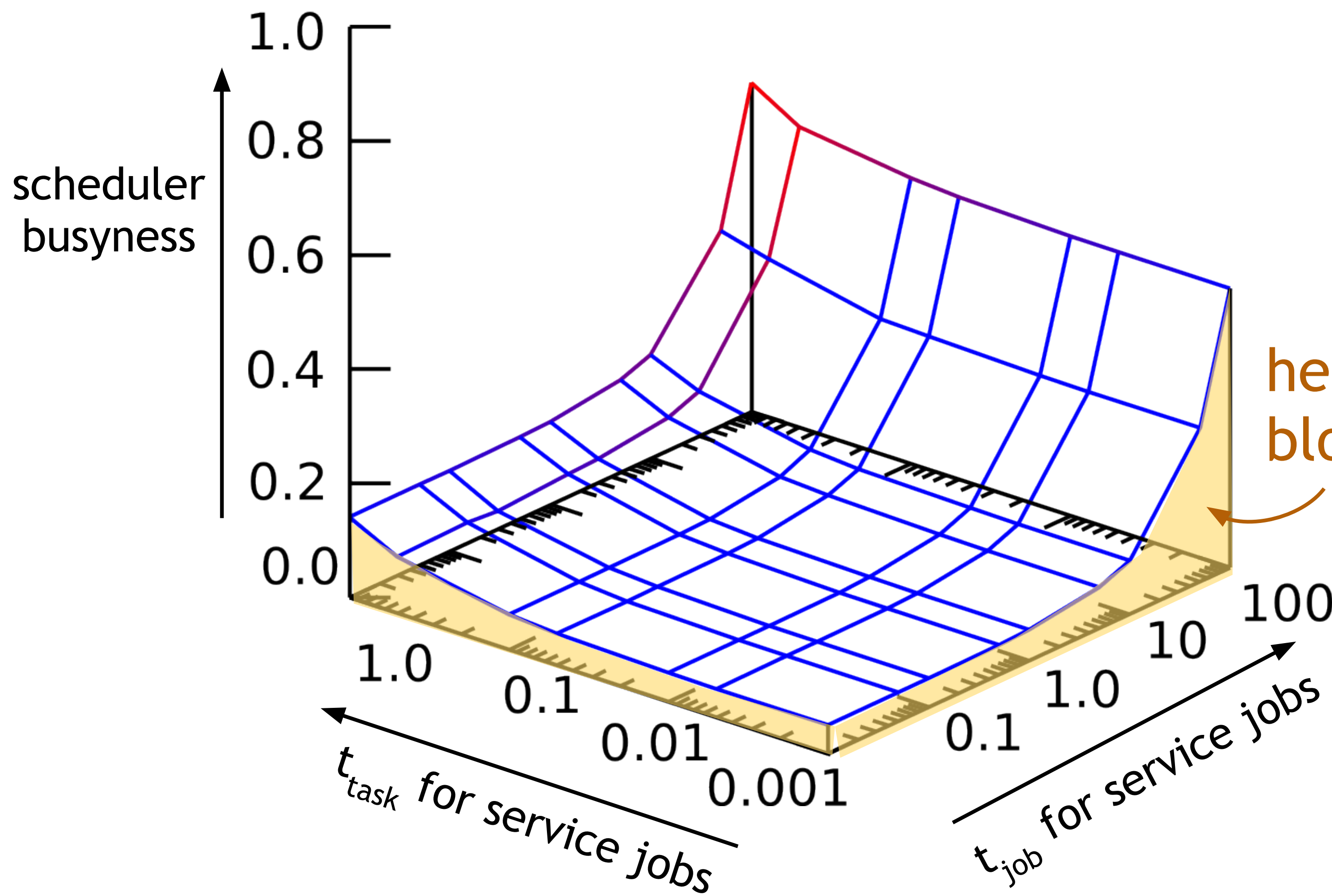
Experiment details:

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

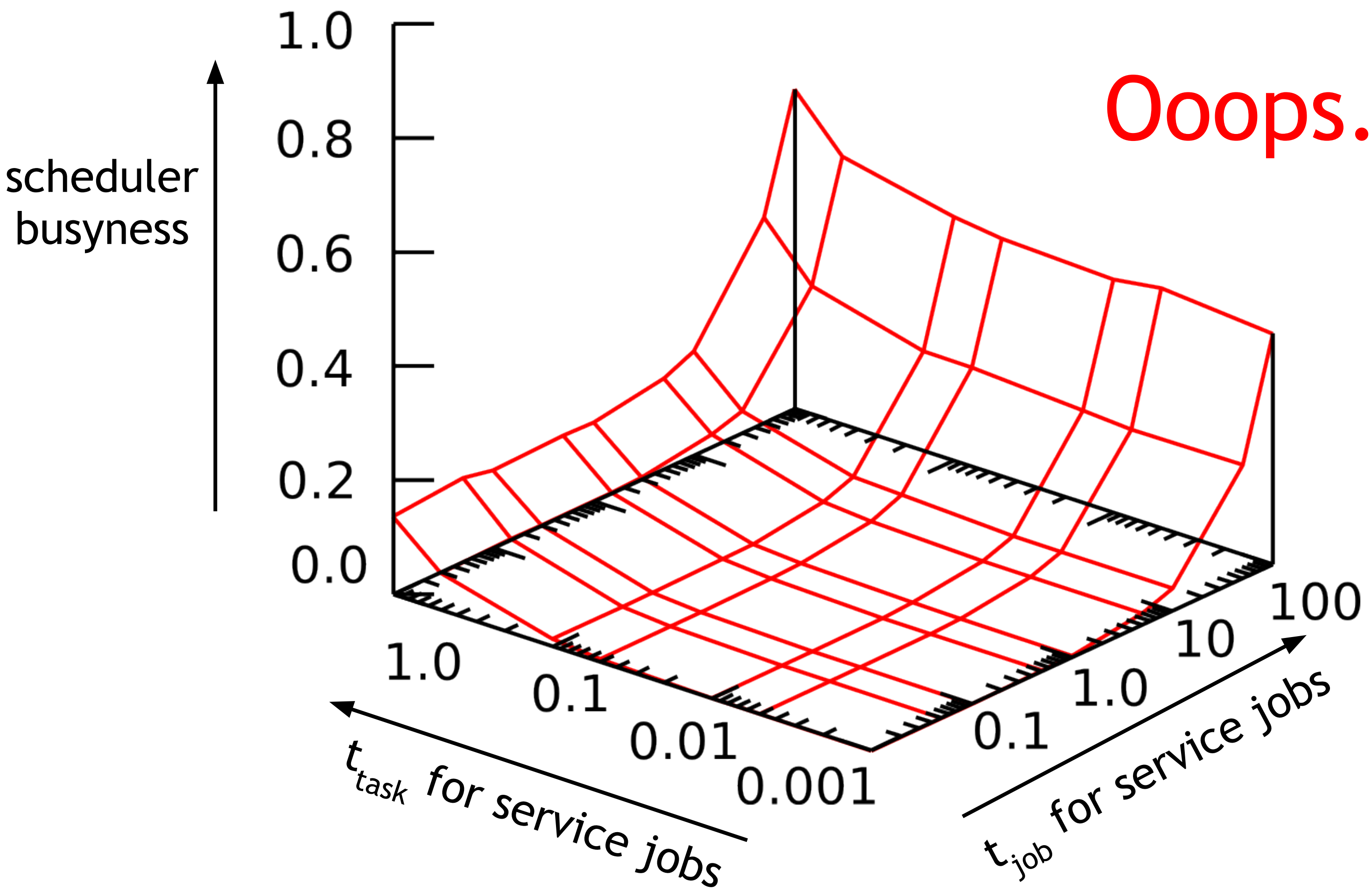
monolithic, uniform decision time (single logic)

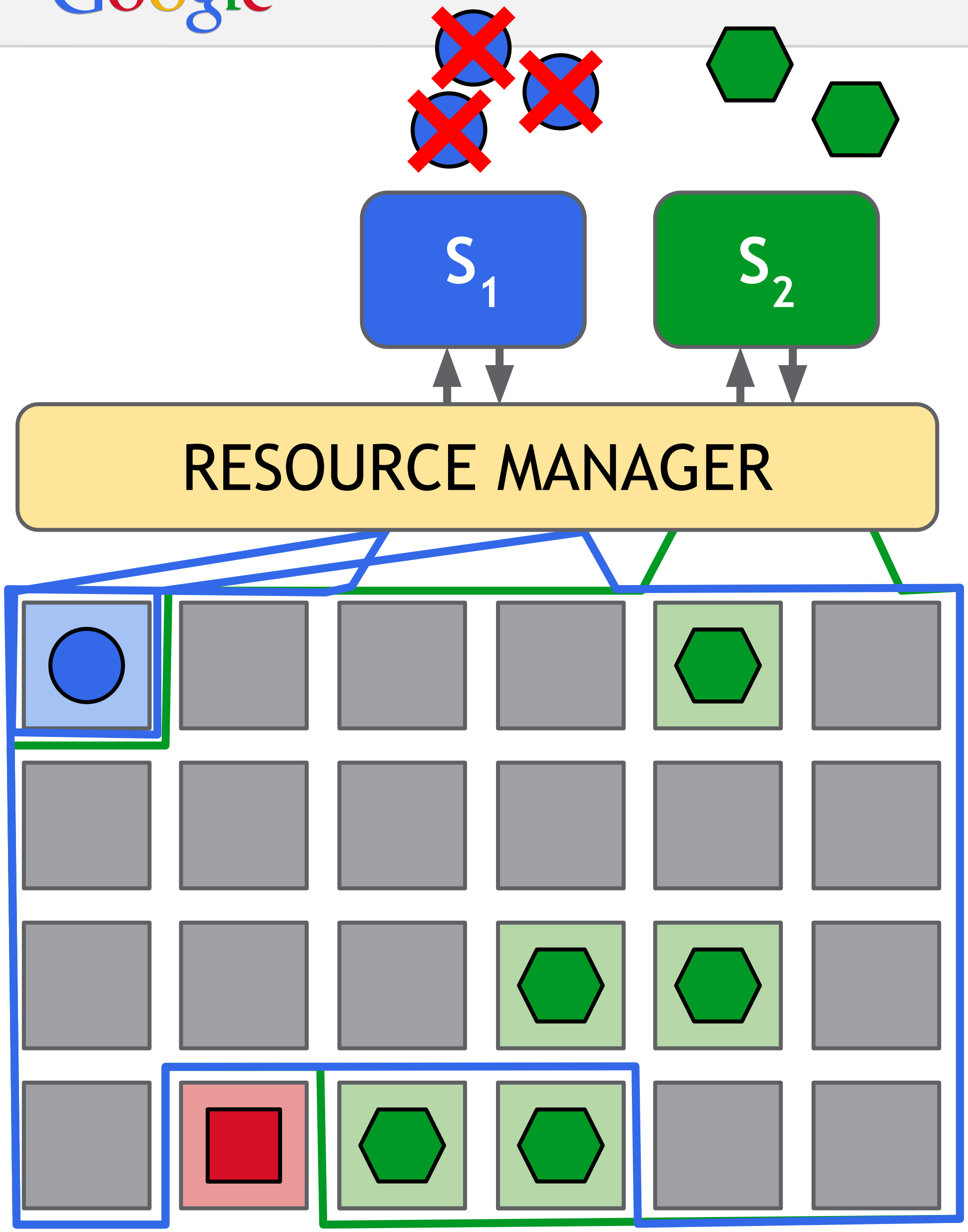


monolithic, fast-path batch decision time



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.

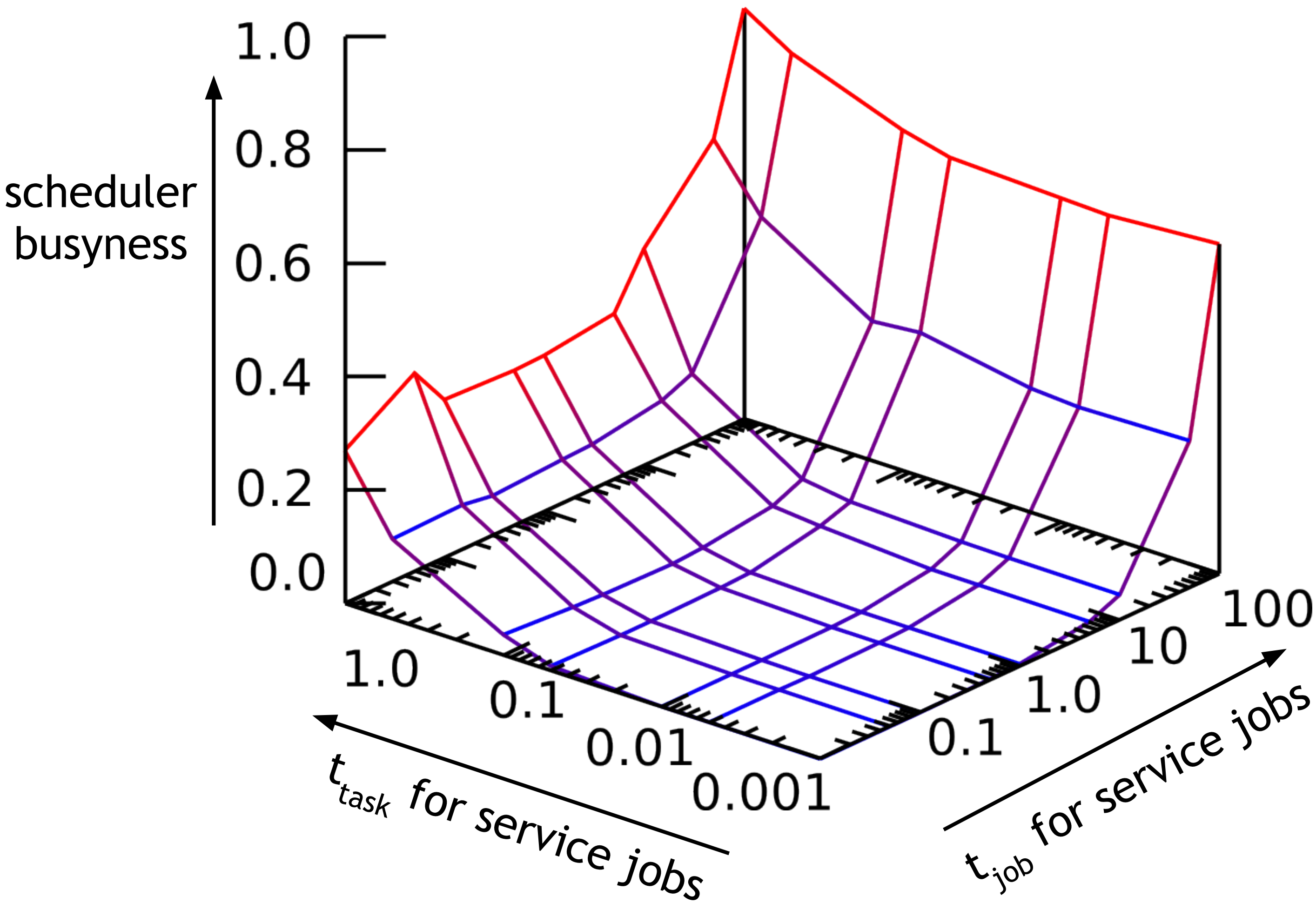
[repeat many times]

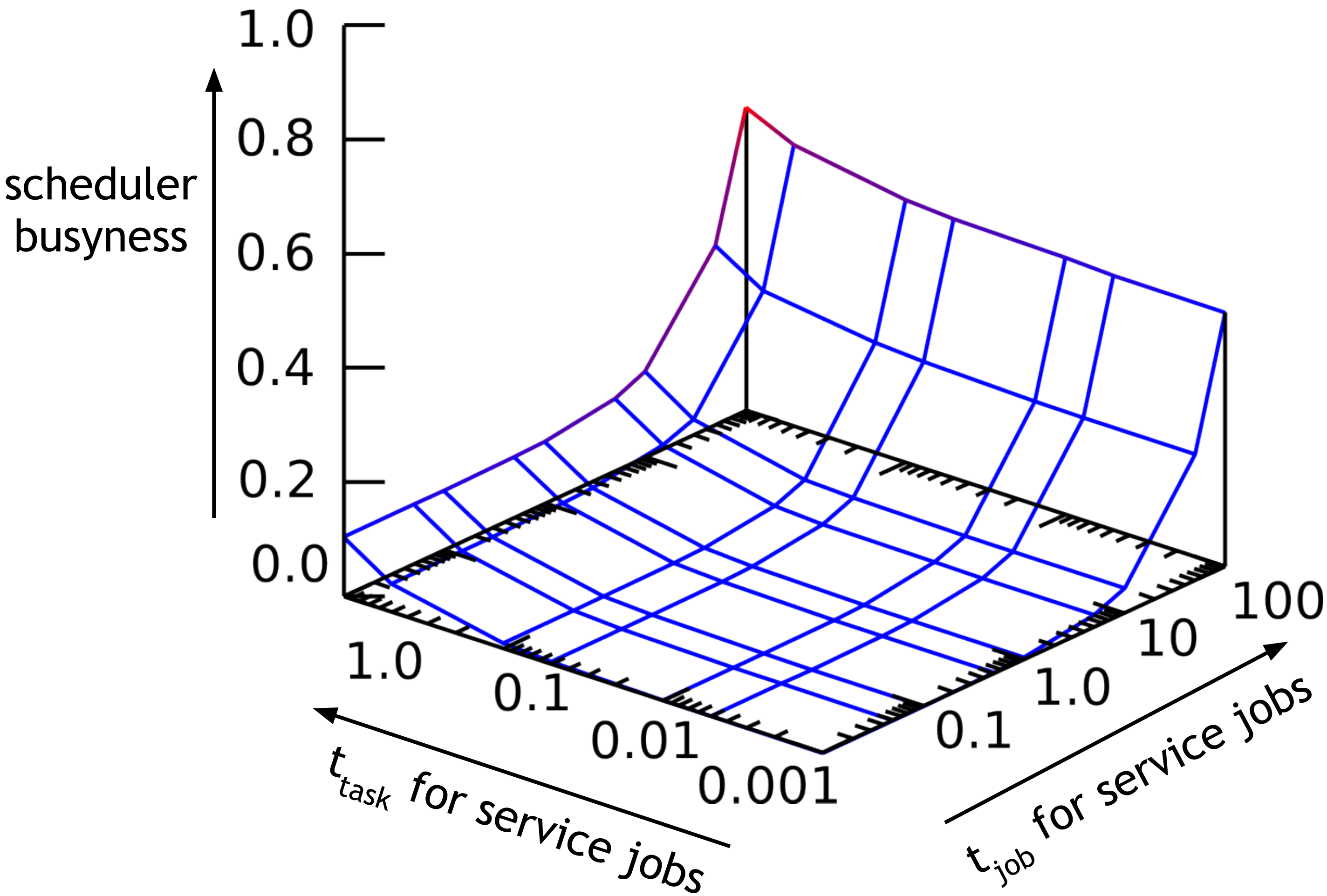
5. Green finishes scheduling.

6. Blue receives large offer.

By now, it has given up.

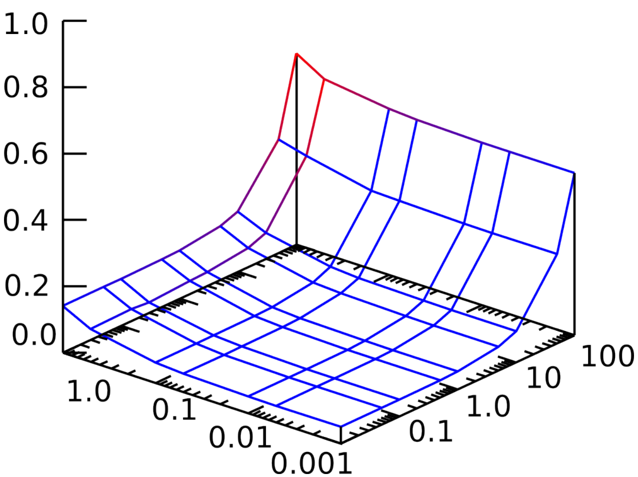
omega, no optimizations



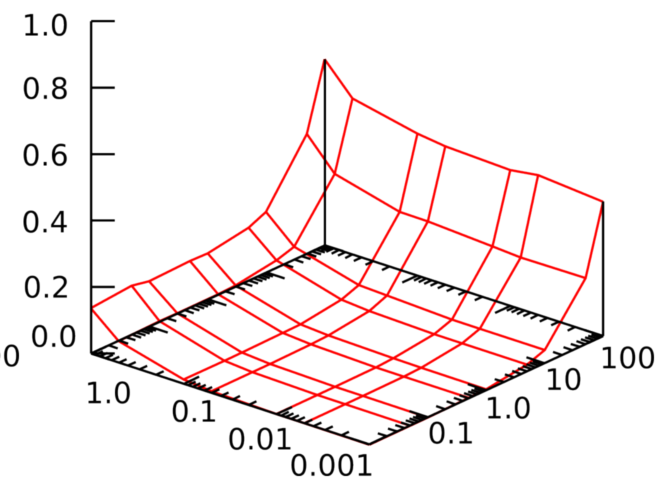

scheduler
busyness
1.0
0.8
0.6
0.4
0.2
0.0
1.0
0.1
0.01
0.001
t_{task} for service jobs
0.1
1.0
10
100
t_{job} for service jobs

Takeaway

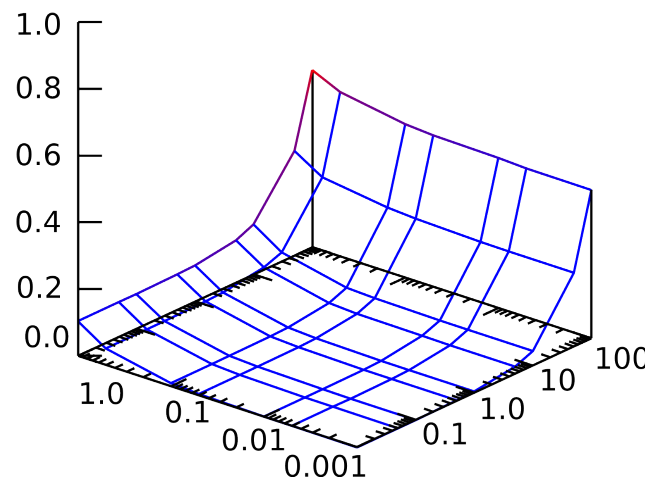
The Omega shared-state model performs as well as a (complex) monolithic multi-path scheduler.



Monolithic



Mesos



Omega

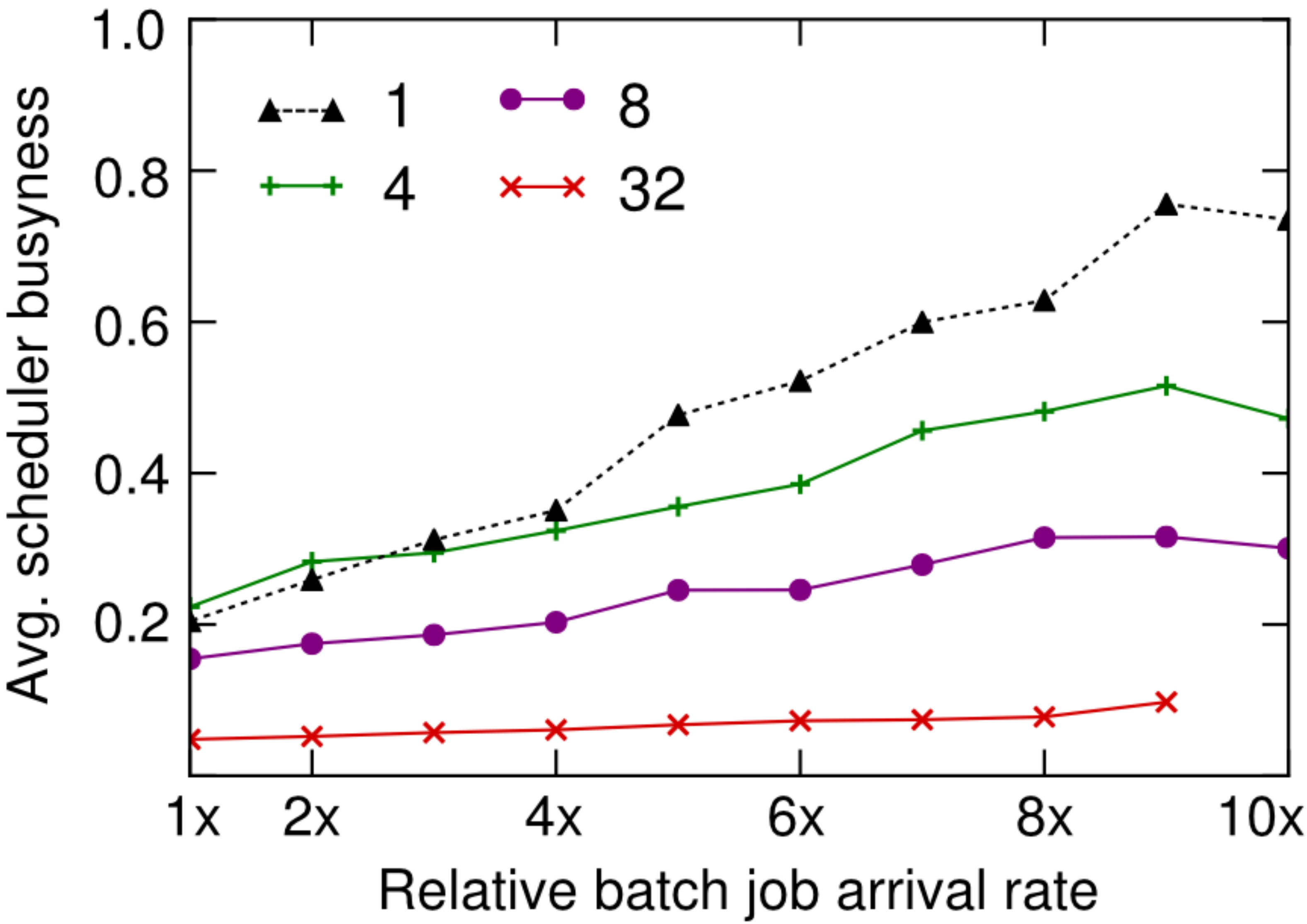
Experiment 2:

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**

scaling to many schedulers



1) intro & motivation

2) workload characterization

3) comparison of approaches

4) trace-based simulation

5) flexibility case study

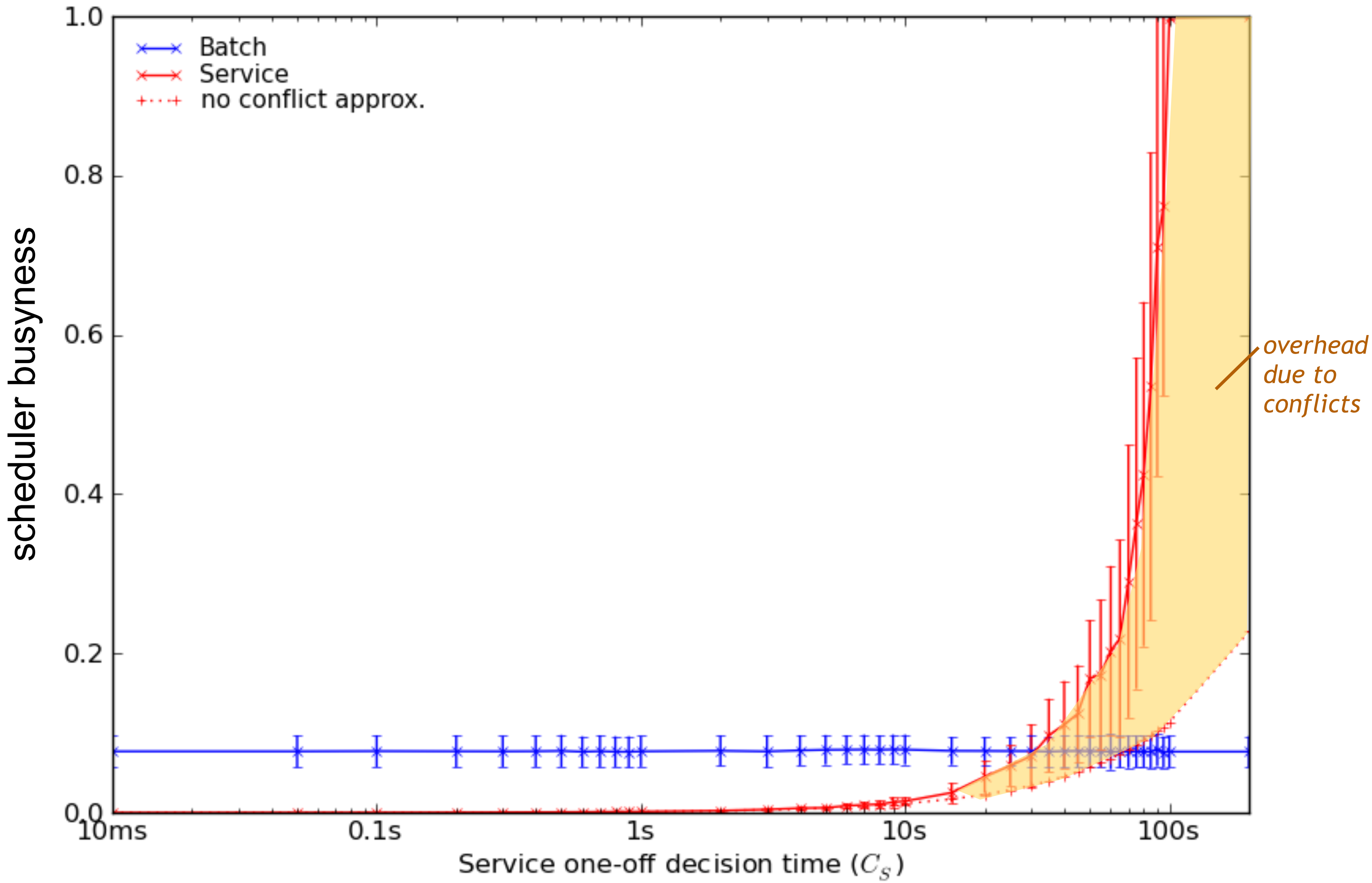
	lightweight simulator	*high-fidelity simulator*
machines	homogeneous	real-world
job parameters	empirical distribution	workload trace
constraints	not supported	supported
scheduling algorithm	random first fit	Google algorithm
runtime	fast (24h ≃ 5min)	slow (24h ≃ 2h)

Experiment 3:

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

Experiment details:

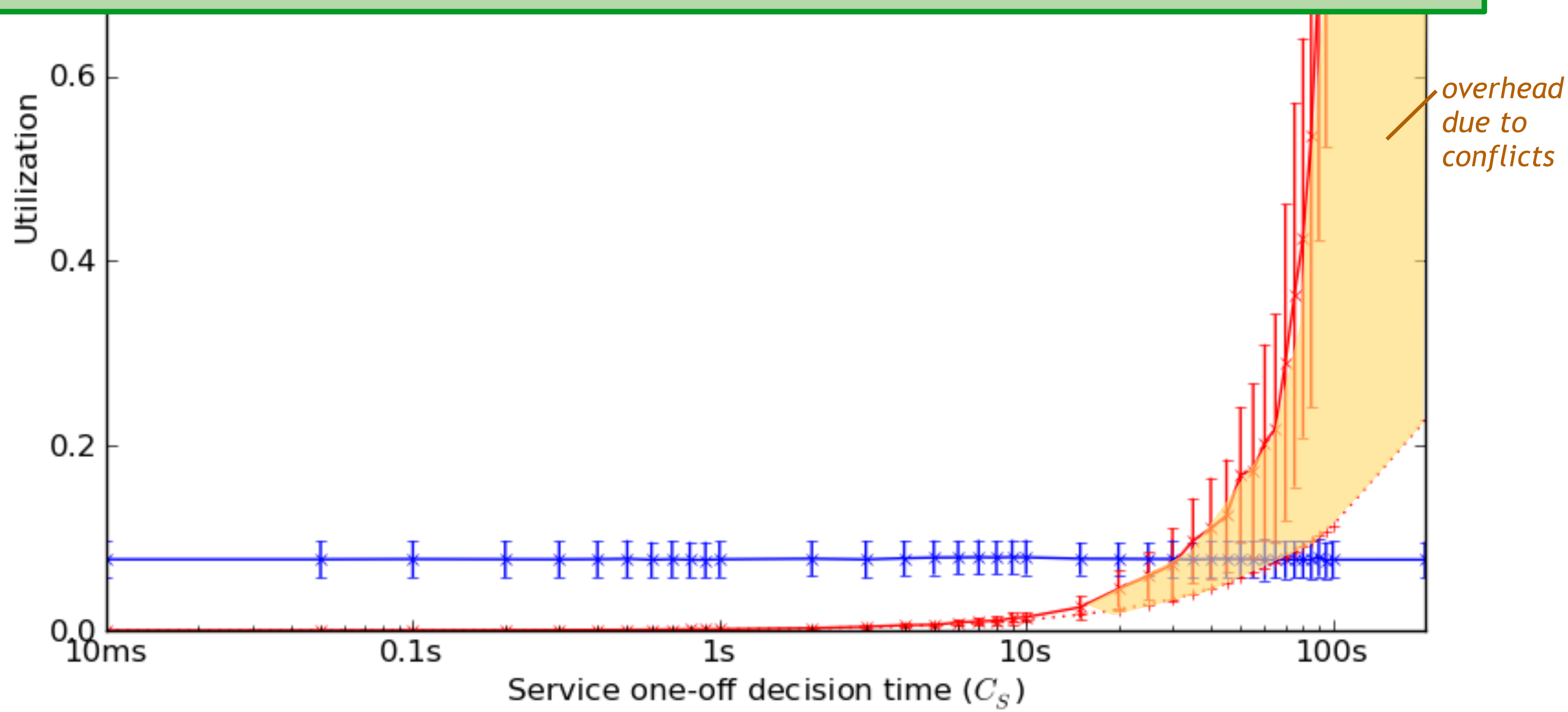
- cluster C, 29 days
- 2 schedulers,
  non-uniform decision time
- varying **Service** scheduler



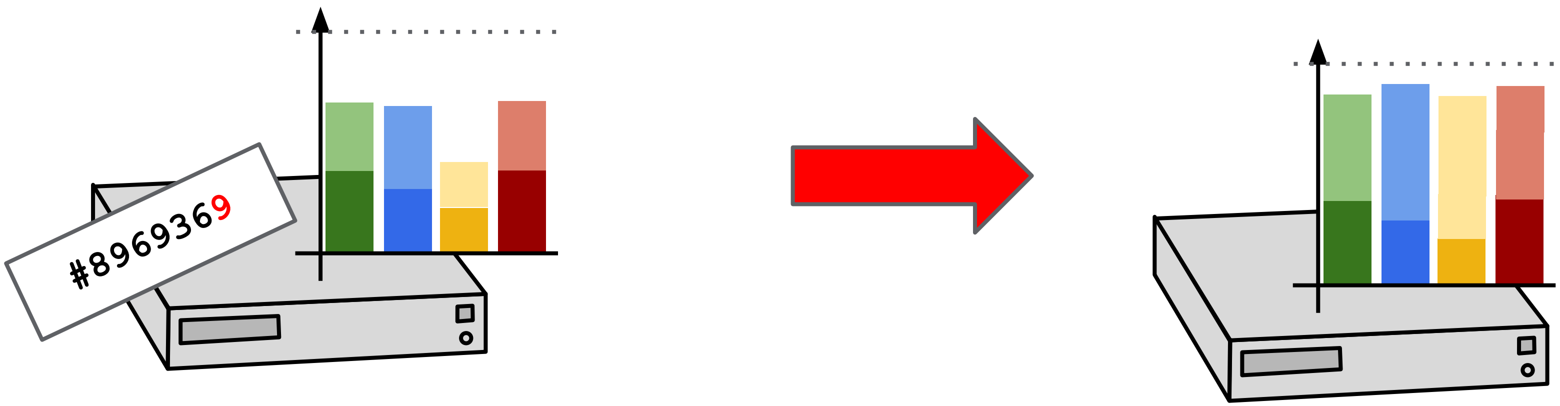
scheduler busyness
1.0
0.8
0.6
0.4
0.2
0.0
Batch
Service
no conflict approx.
10ms
0.1s
1s
10s
100s
Service one-off decision time (C_S)
overhead due to conflicts

TAKEAWAY

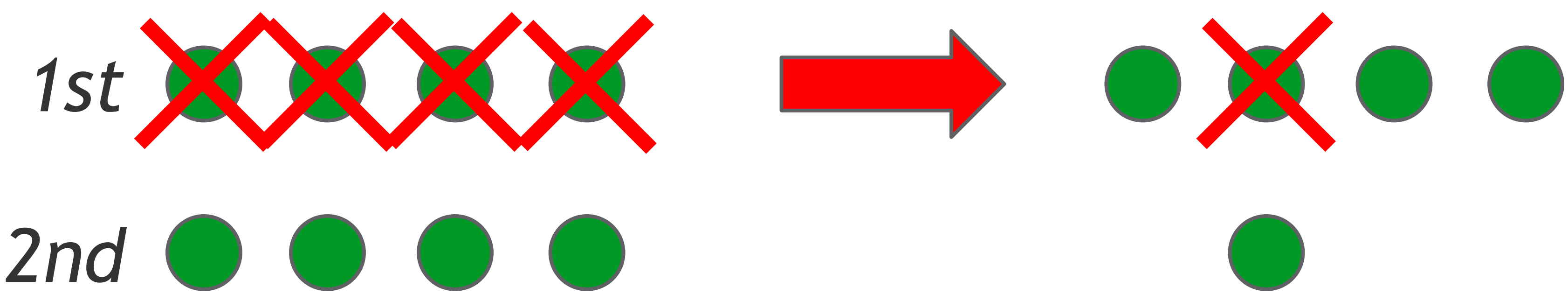
Interference is higher for real-world settings.



1. Fine-grained conflict detection



2. Incremental commits



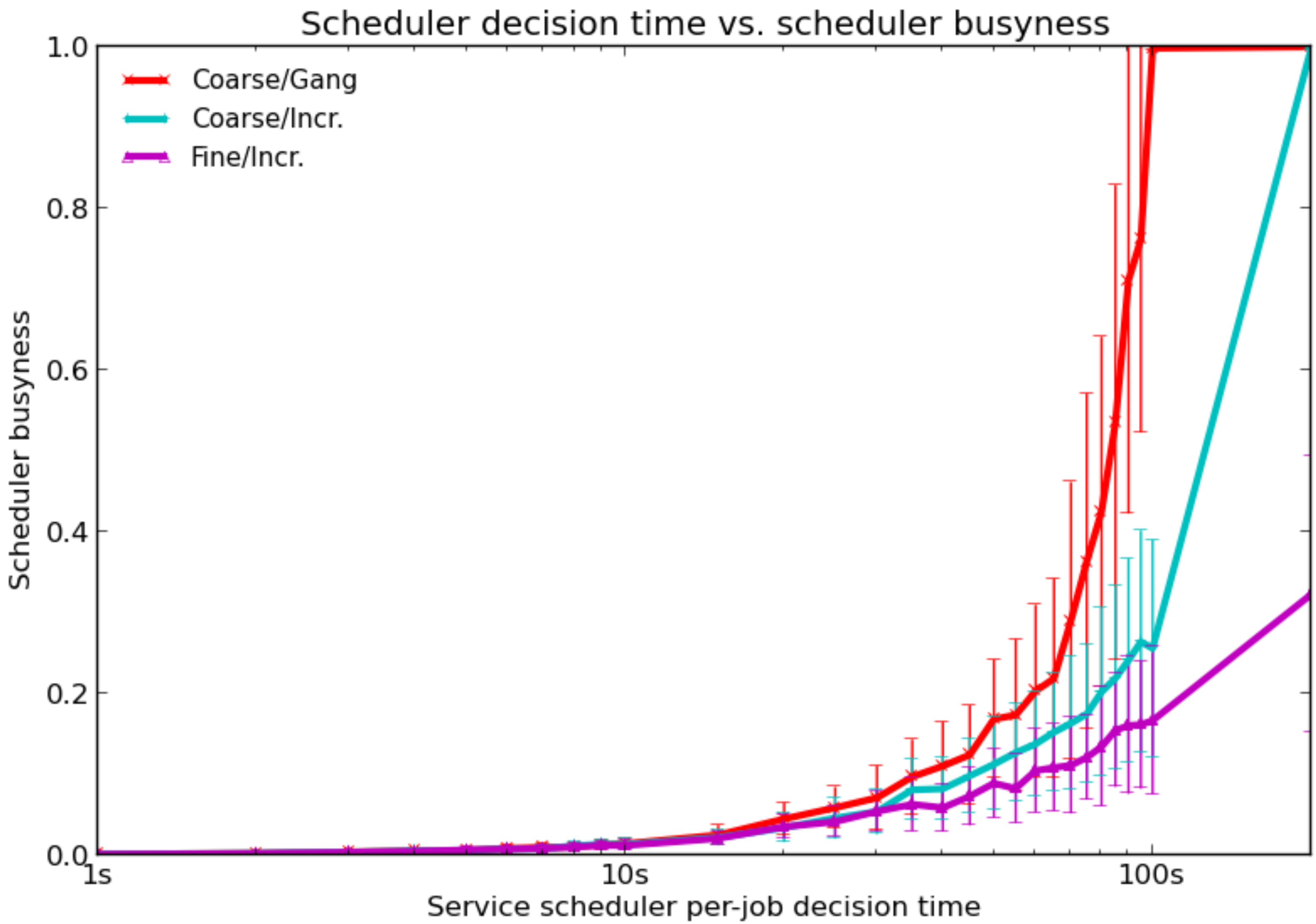
Experiment 4:

How do the optimizations affect performance?

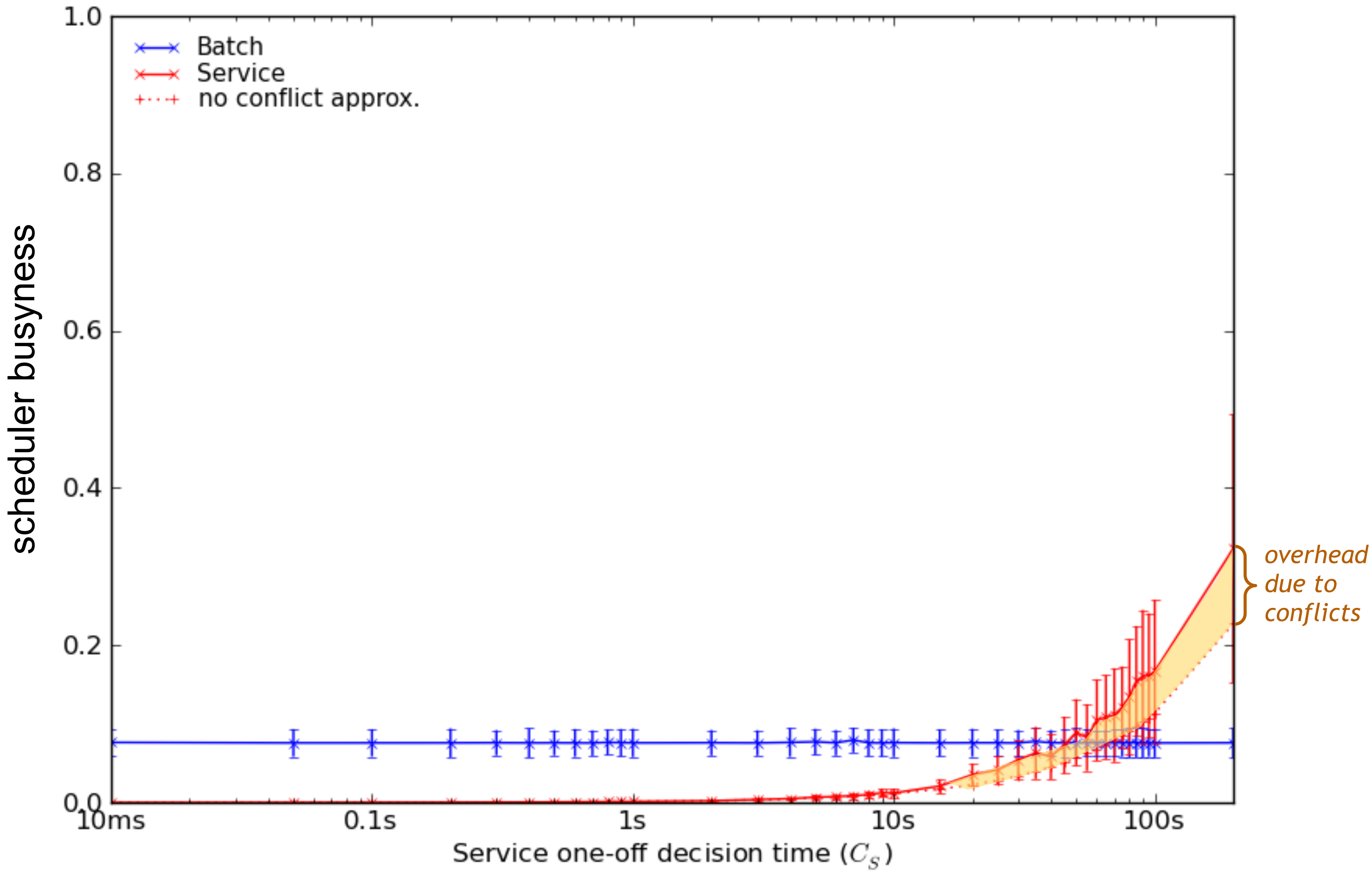
Experiment details:

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

impact on scheduler utilization

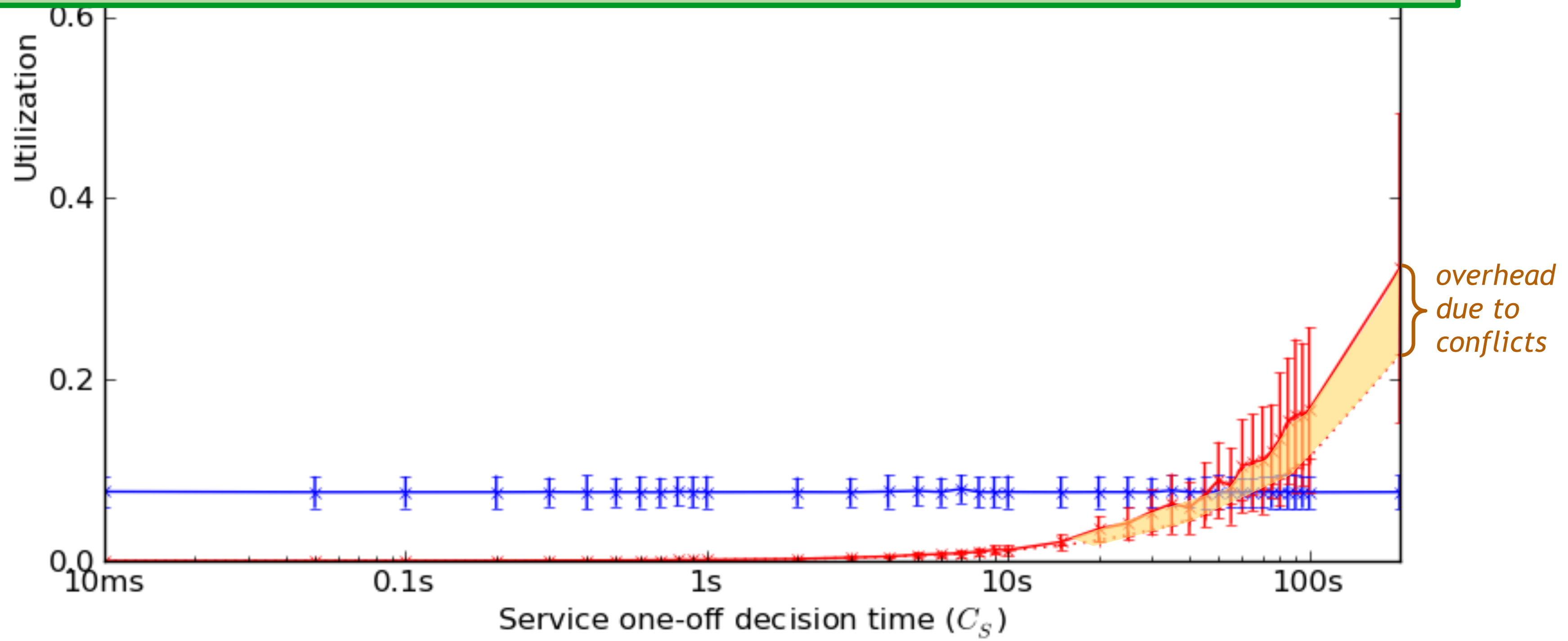


practical implications – scheduler utilization



TAKEAWAY

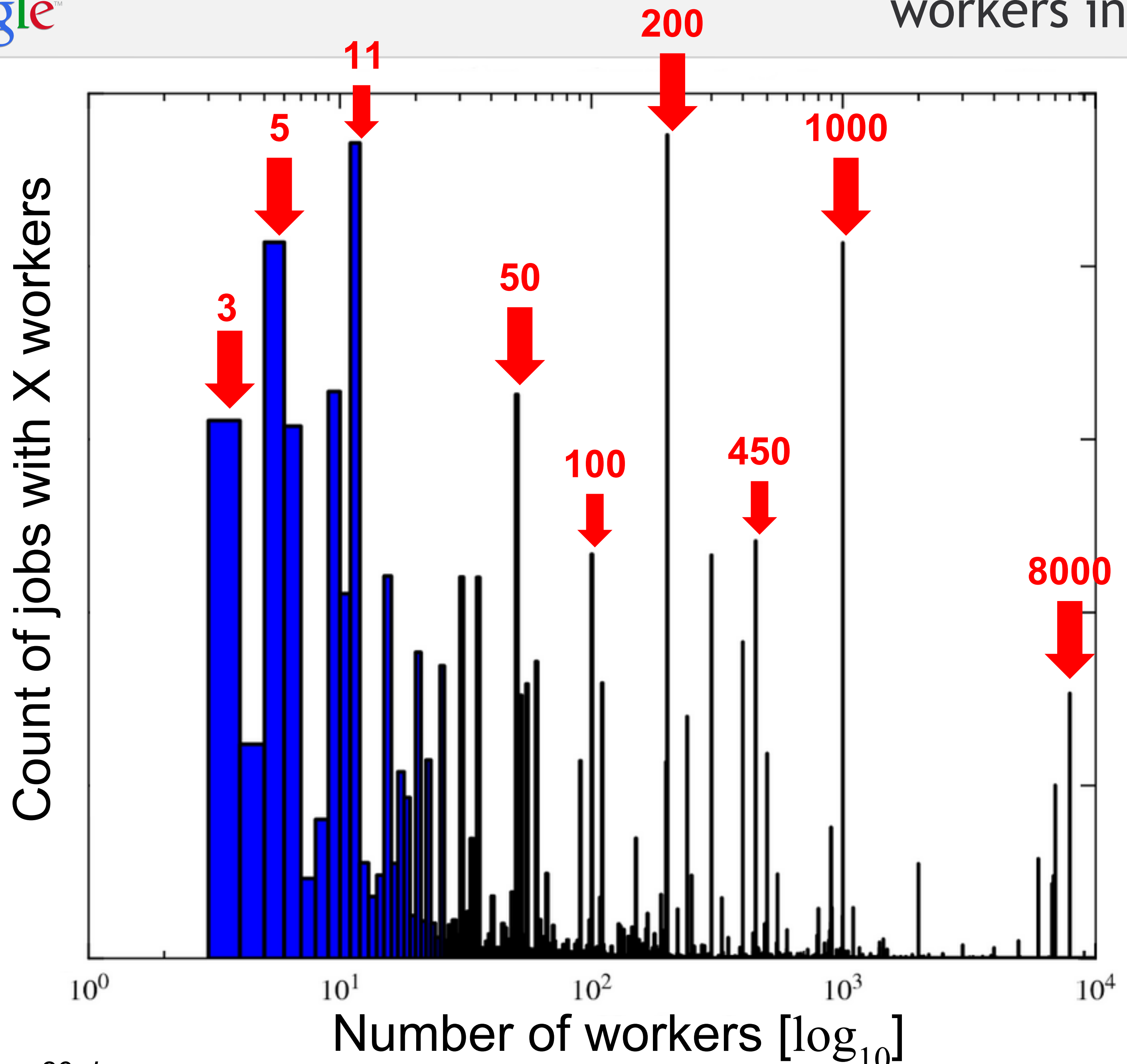
We can make simple improvements that significantly improve scalability.



Case study

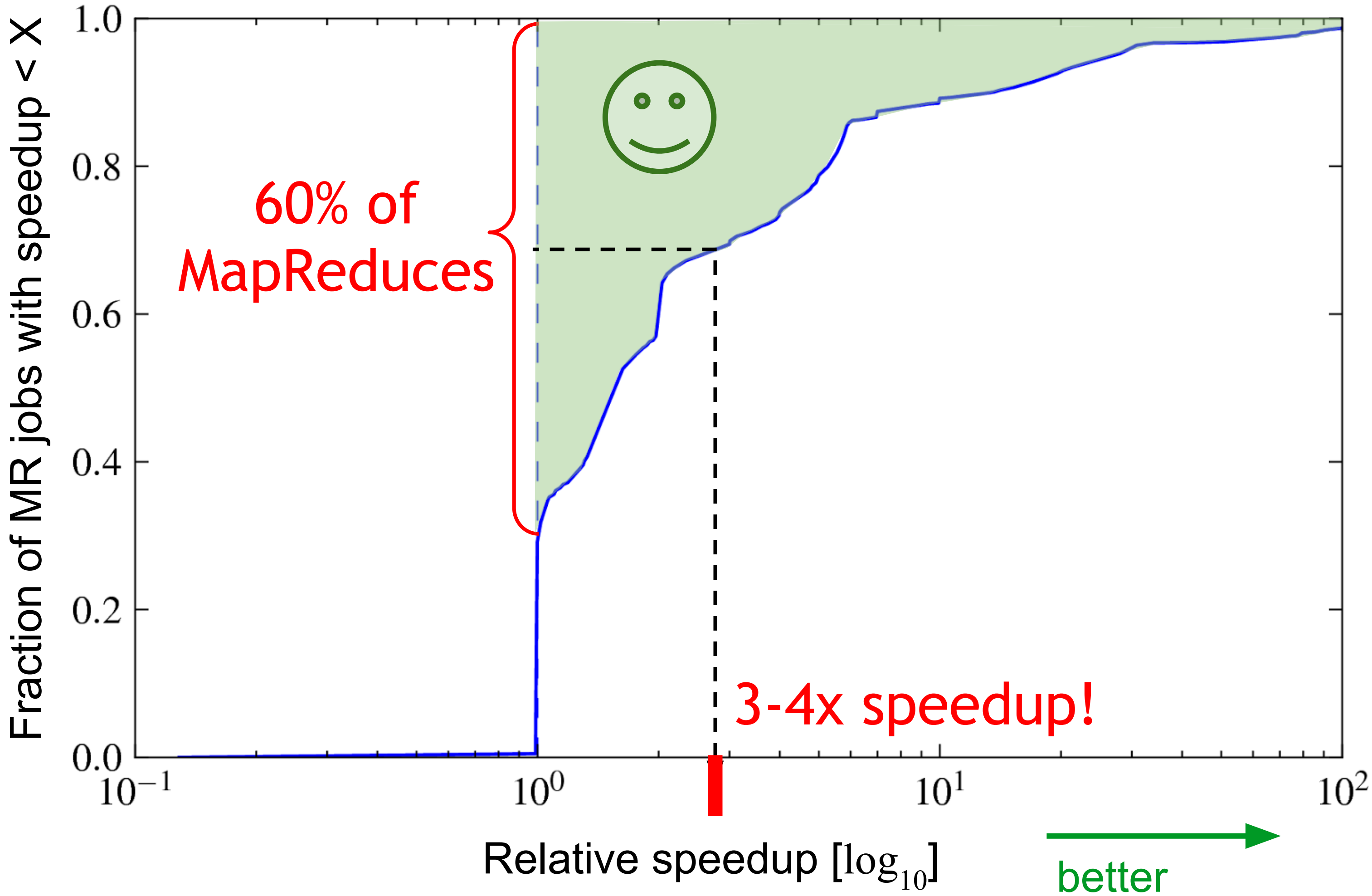
MapReduce scheduler with opportunistic extra resources

workers in MR jobs



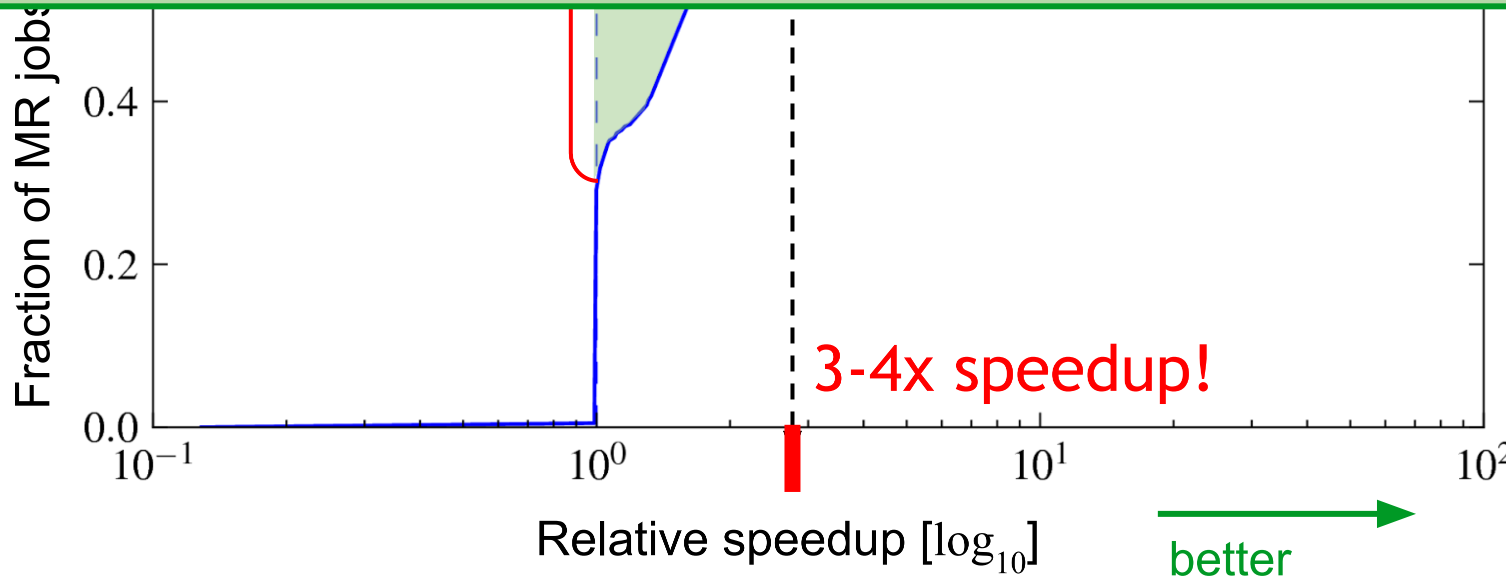
Snapshot over 29 days

case study: a MapReduce scheduler



TAKEAWAY

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



cluster C, 29 days

TAKEAWAYS

Flexibility and scale require parallelism,

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

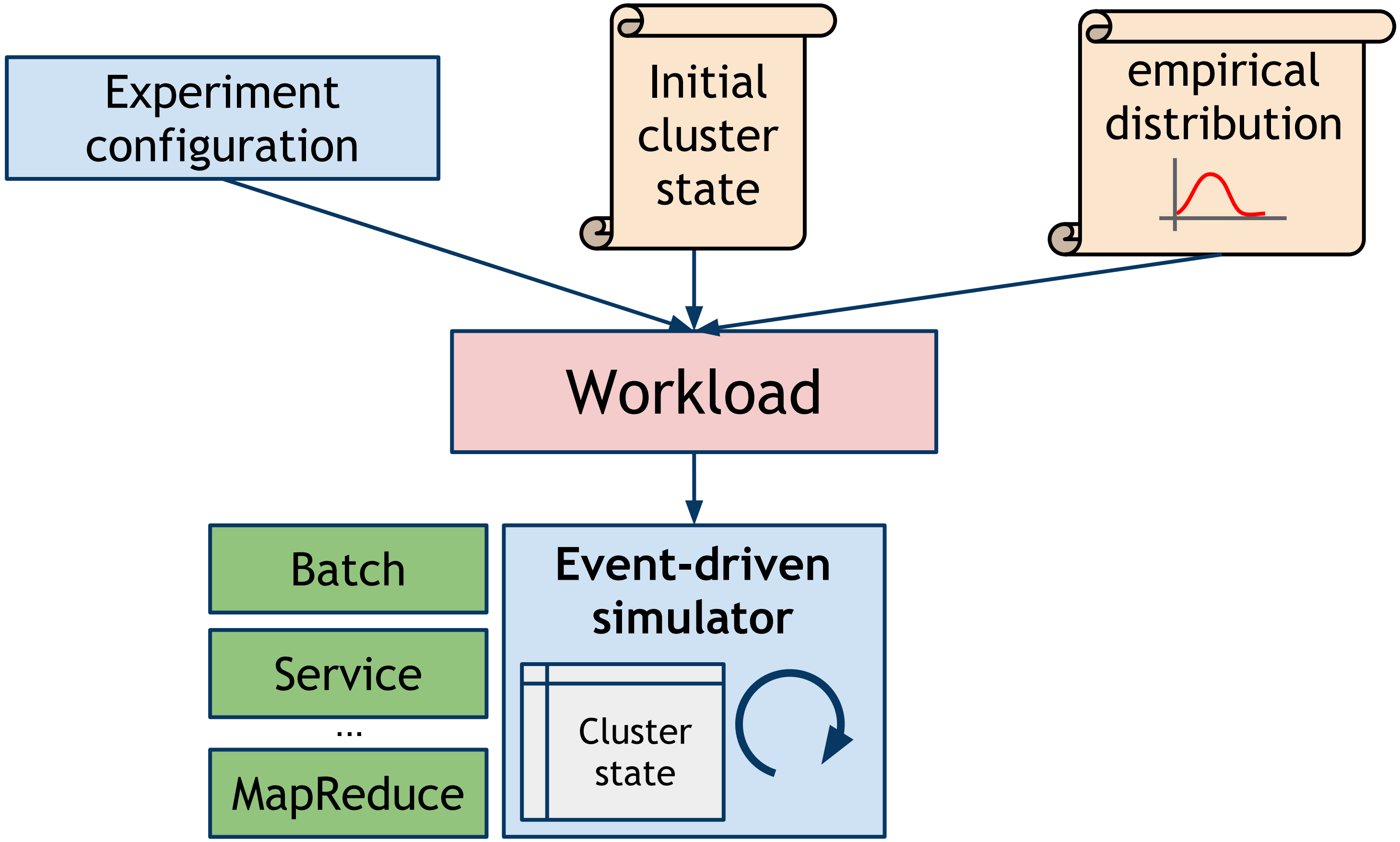
using shared state is the way to do it right!

Google

BACKUP SLIDES

Why might scheduling take 60 seconds?

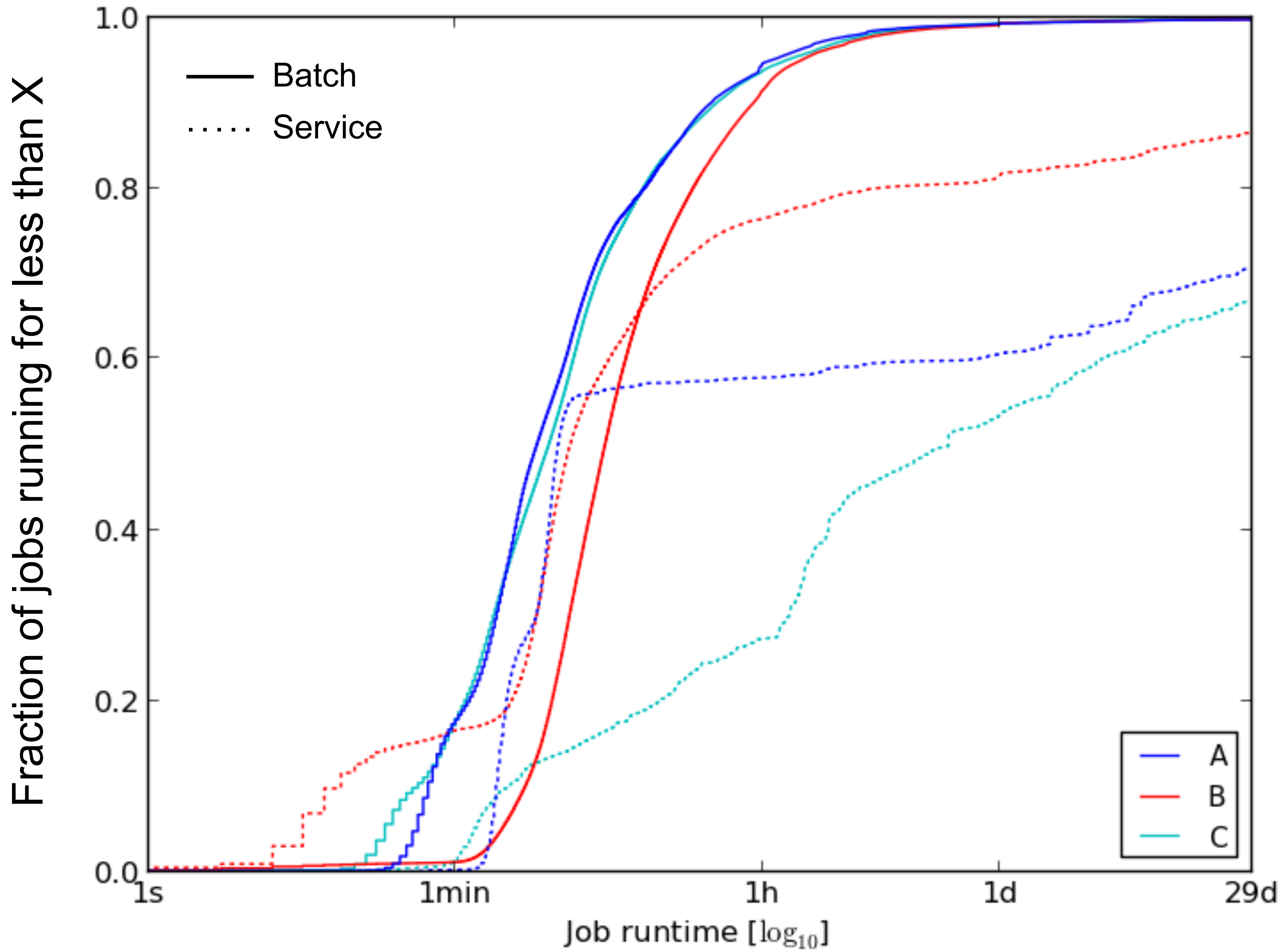
- Large jobs (1,000s of tasks)
- Optimization algorithms (constraint solving, bin packing)
- Very picky jobs in a full cluster (preemption consequences)
- Monte Carlo simulations (fault tolerance)



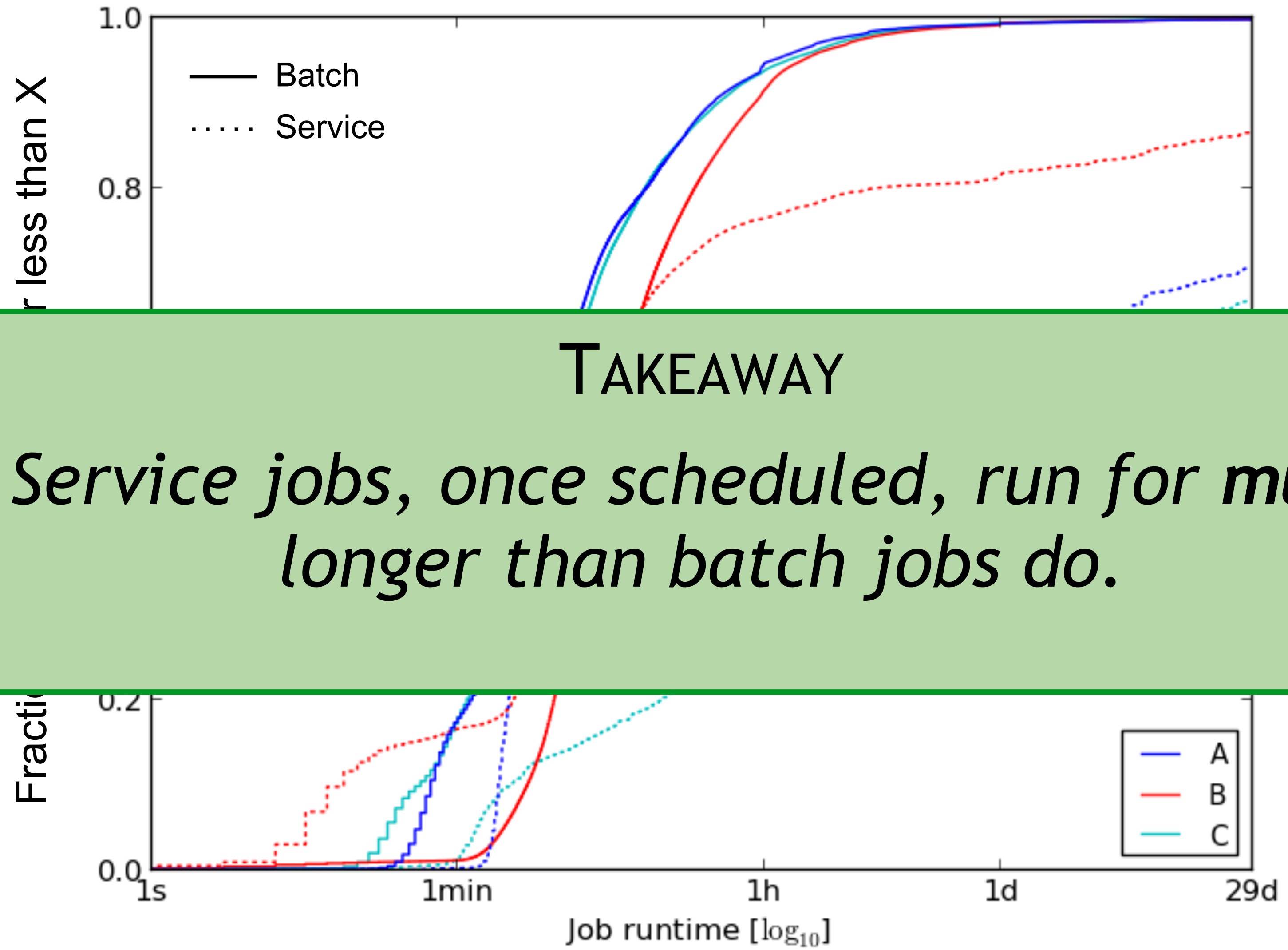
Code [soon to be] **available:**

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

workload: job runtime distributions



workload: job runtime distributions

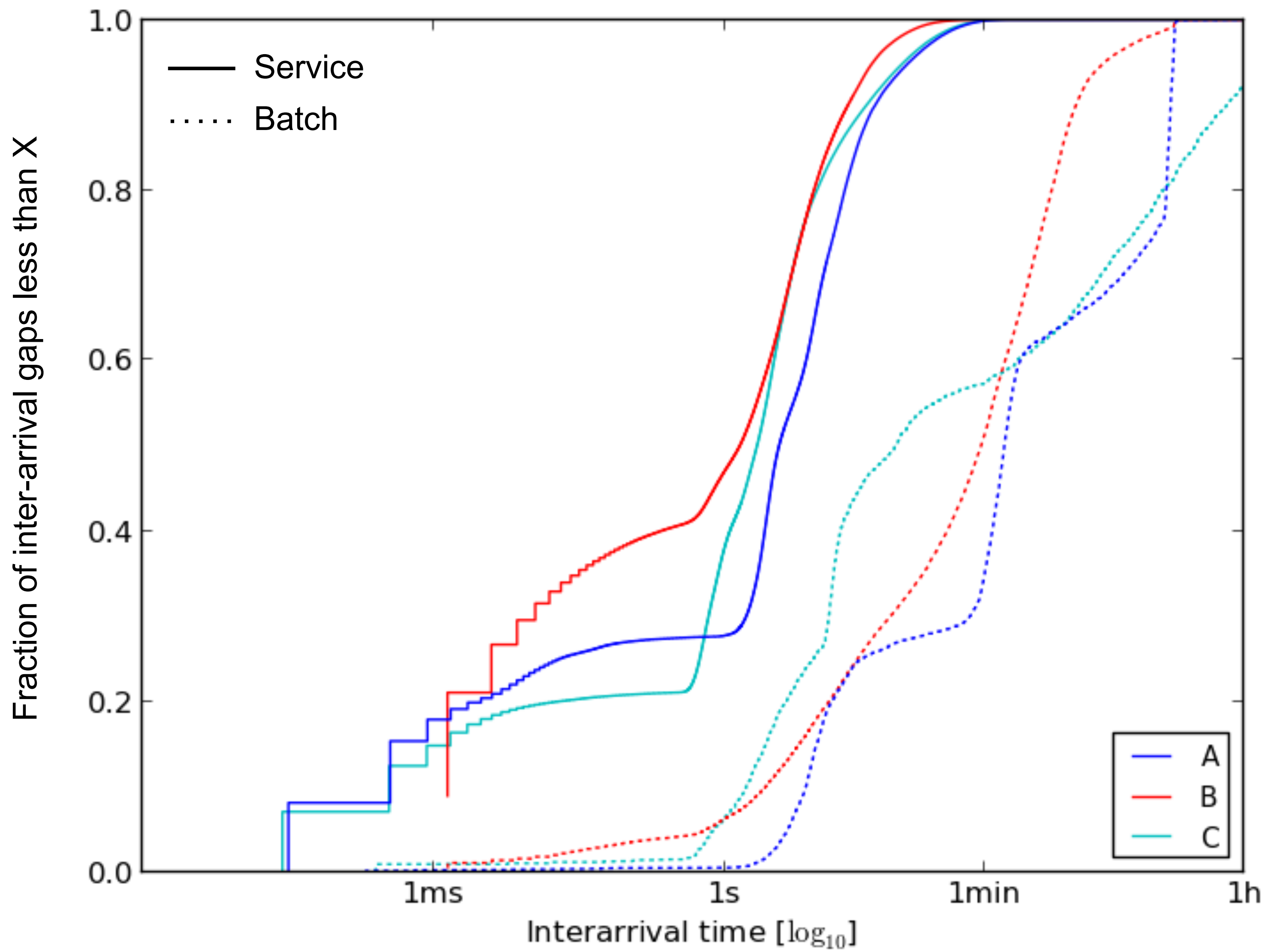


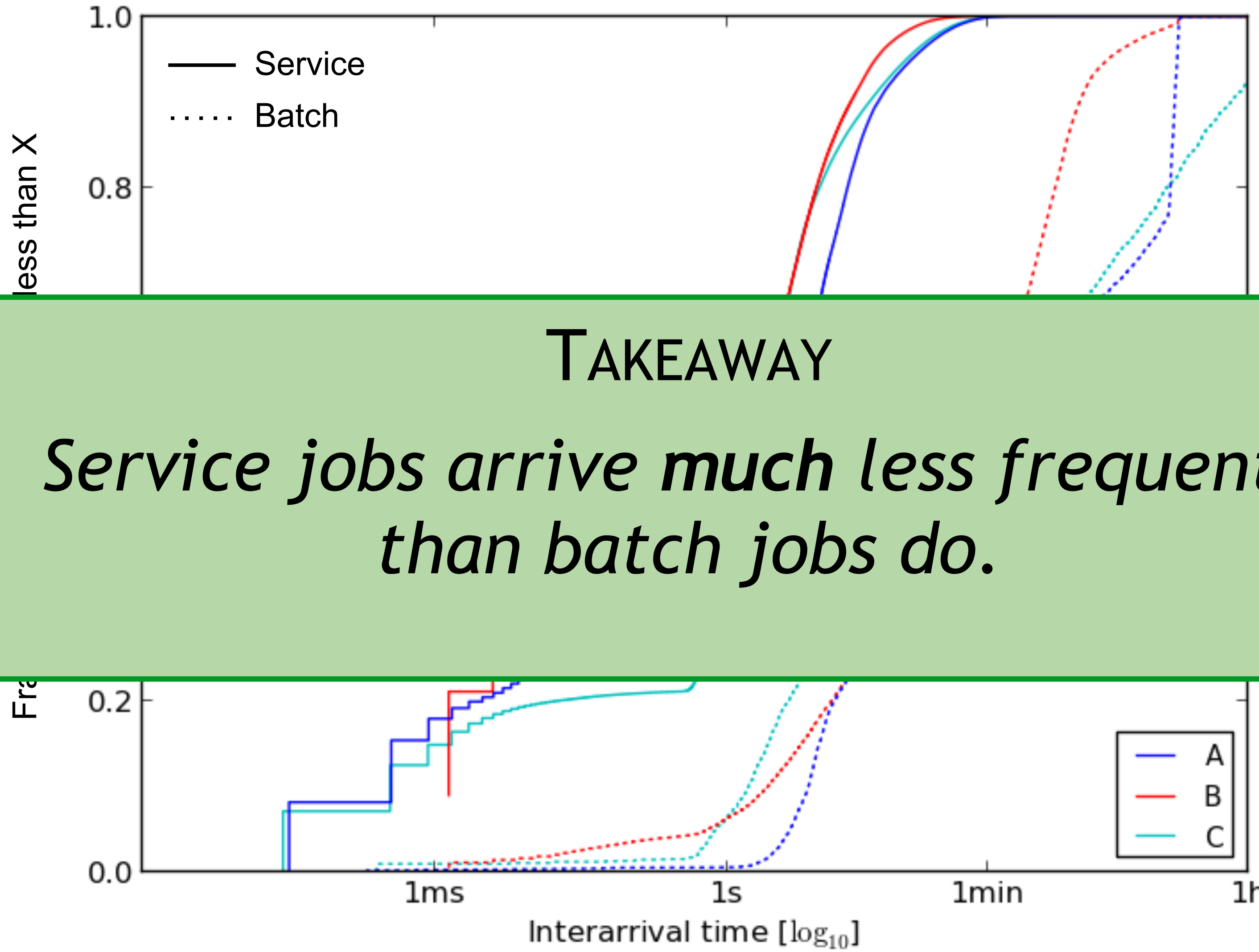
TAKEAWAY

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

Google

workload: inter-arrival time distributions





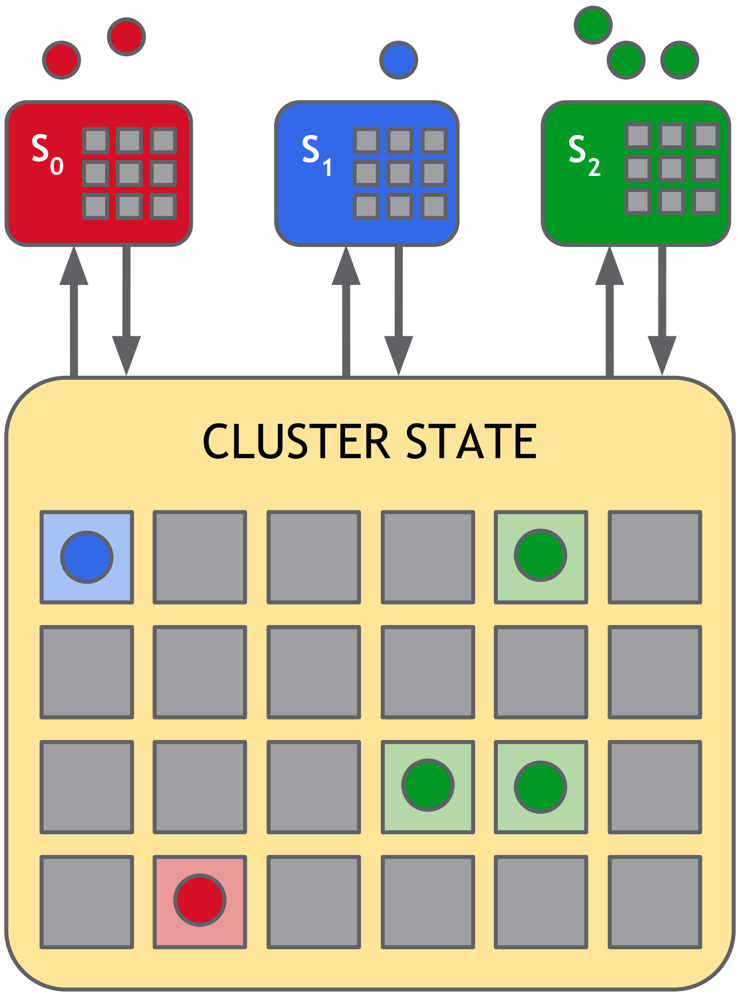
TAKEAWAY

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

Shared state

Omega
Brave New Borg

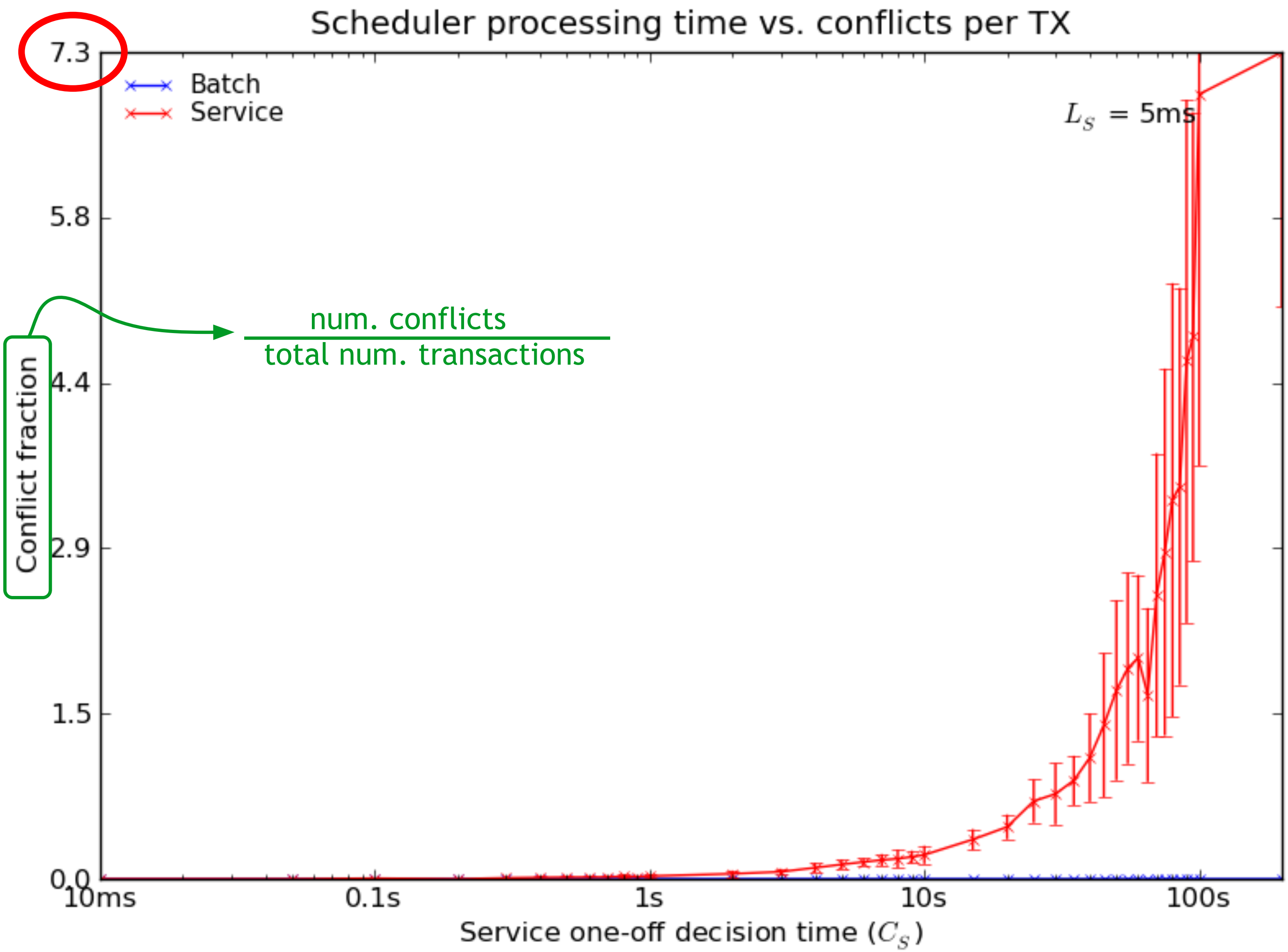
- Deltas against shared state
- Easy to develop & maintain
- Heterogeneous scheduling logic supported



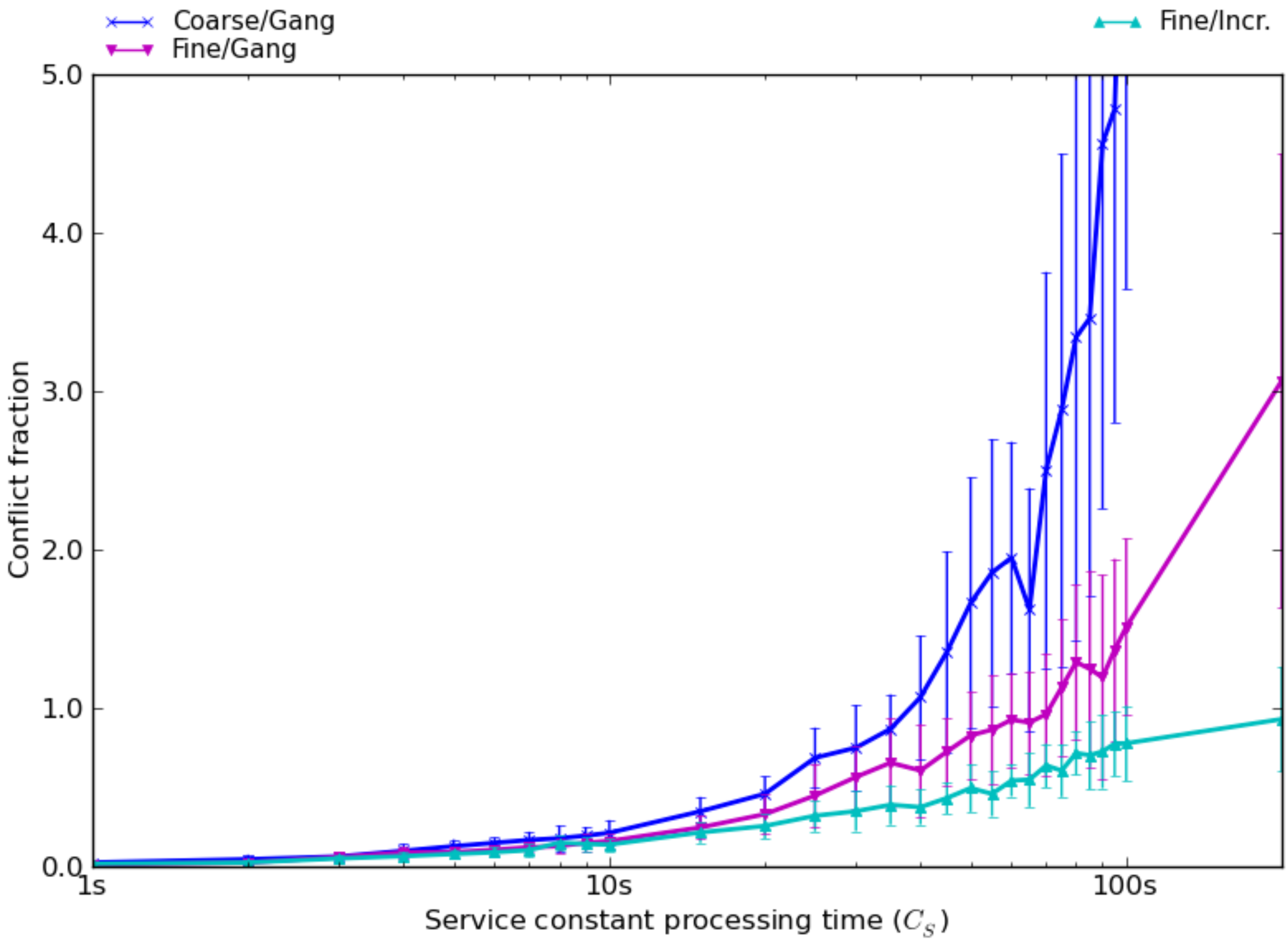
Optimistic concurrency

- No explicit coordination required
- Post-hoc interference resolution
- Scales well

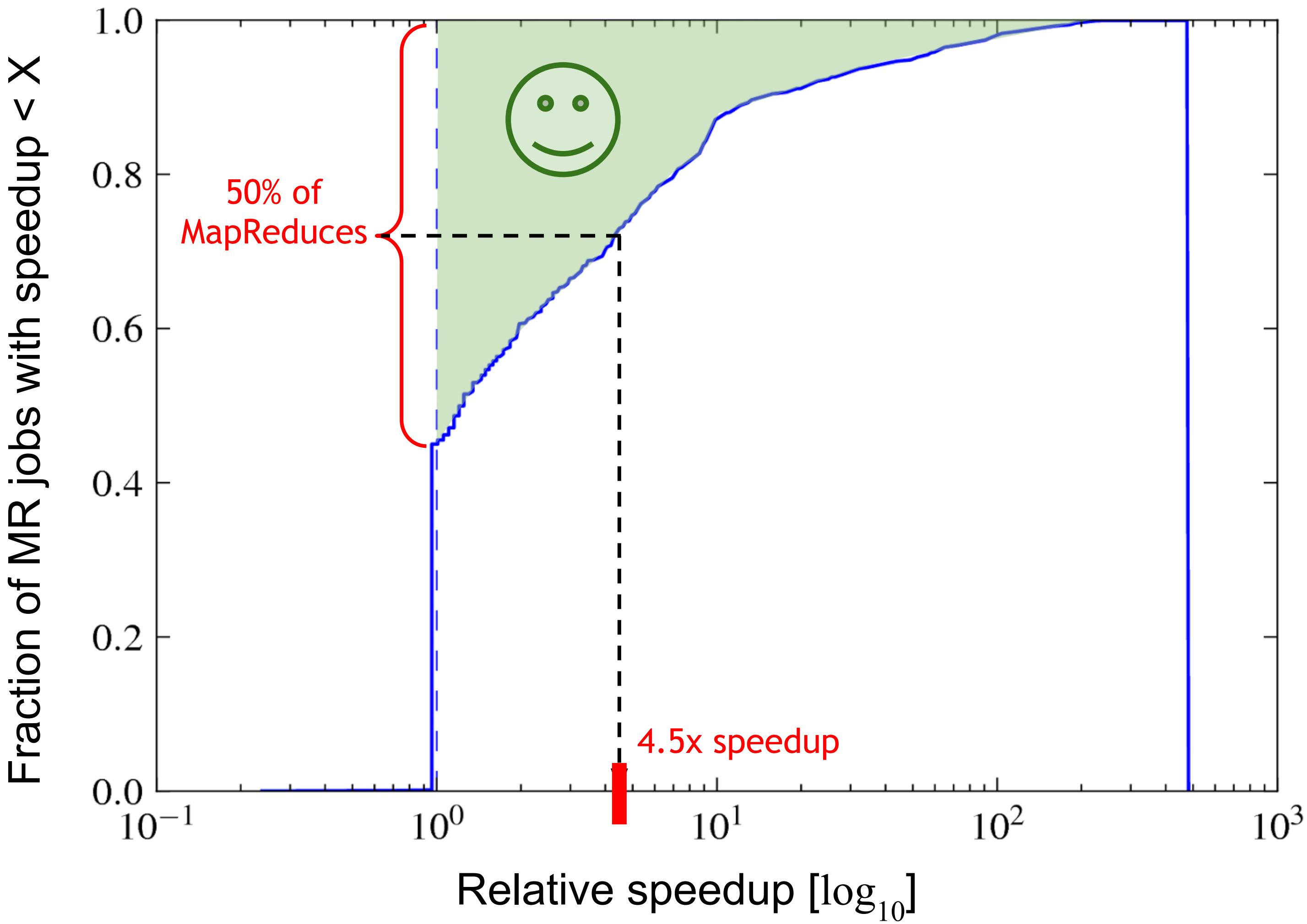
conflict fraction



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**