Firmament

Fast, centralized cluster scheduling at scale

Ionel Gog Malte Schwarzkopf Adam Gleave
Robert N. M. Watson Steven Hand

UNIVERSITY OF CAMBRIDGE
Google

Meet Sesame, Inc.

- Sesame’s employees run:

1. Interactive data analytics
   that must complete in seconds

2. Long-running services
   that must provide high performance

3. Batch processing jobs
   that increase resource utilization

The cluster scheduler must achieve:

1. **Good task placements**
   - high utilization without interference

2. **Low task scheduling latency**
   - support interactive tasks
   - no idle resources

State of the art

Good task placements ⟷ Low scheduling latency

Centralized

Sophisticated algorithms

[Borg, Quincy, Quasar]

Distributed

Simple heuristics

[Sparrow, Tarcil, Yaq-d]

Hybrid

Split workload, provide either

[Mercury, Hawk, Eagle]

Can’t get **both** good placements and low latency for the **entire workload**!

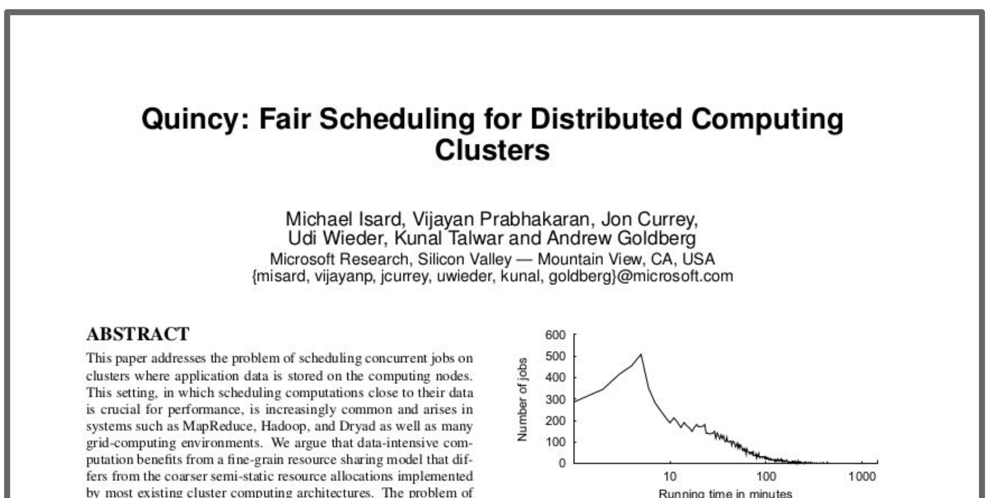
Firmament provides a solution!

- Centralized architecture
- Good task placements
- Low task scheduling latency
- Scales to 10,000+ machines

- ➢ Finds **optimal** task placements

- ➢ Min-cost **flow-based** centralized scheduler

[SOSP 2009]

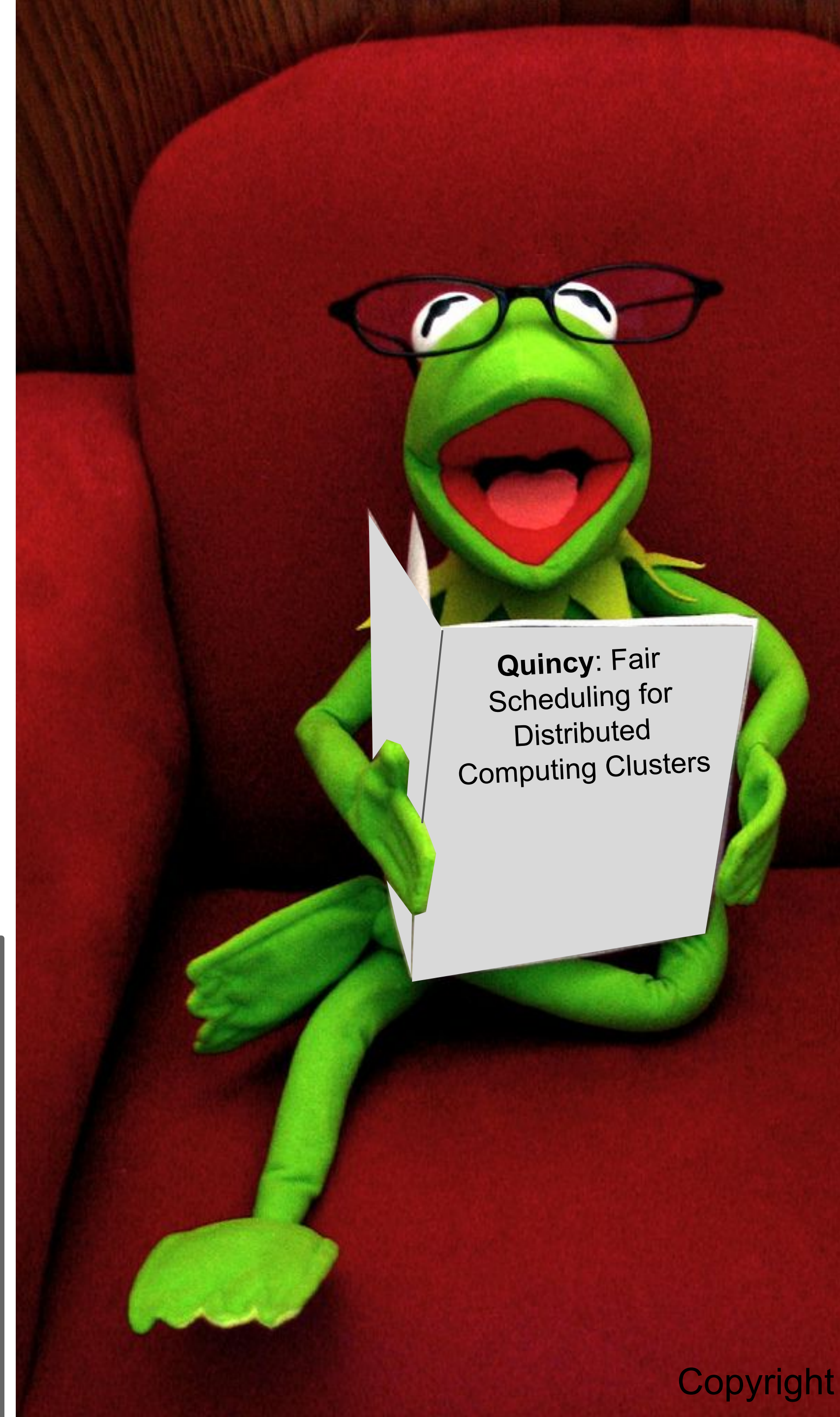


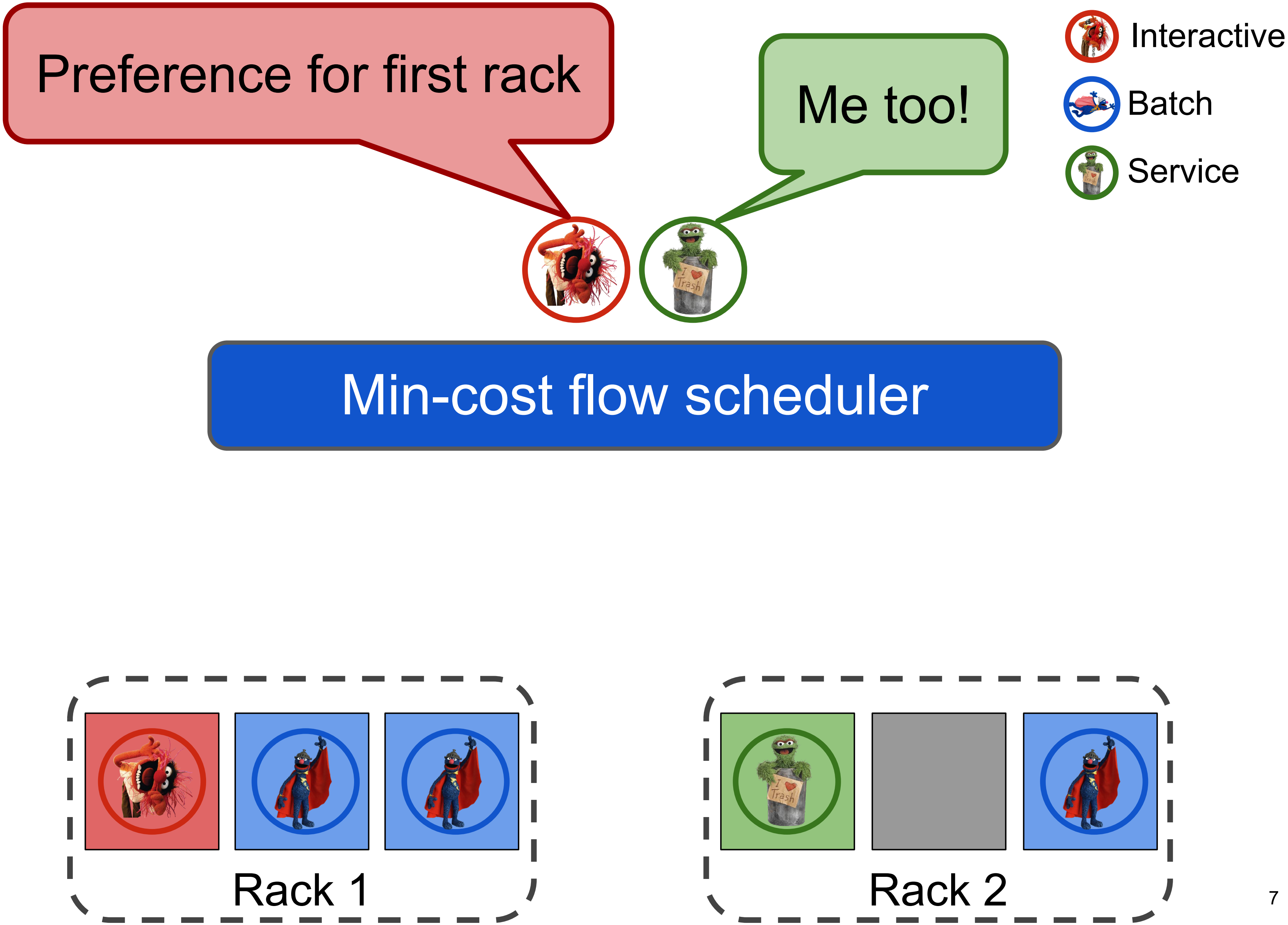
Quincy: Fair Scheduling for Distributed Computing Clusters

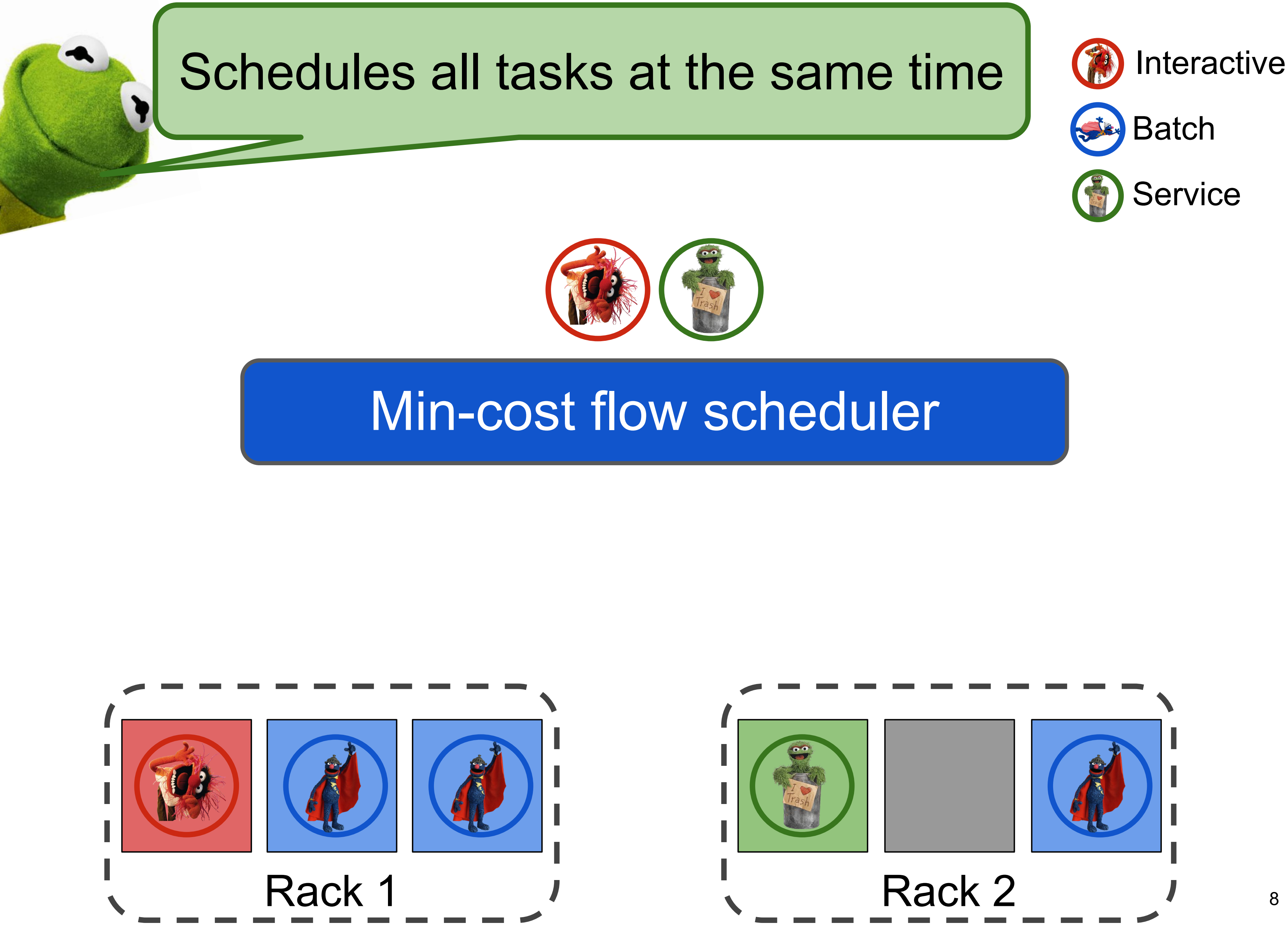
Michael Isard, Vijayan Prabhakaran, Jon Currey,
Udi Wieder, Kunal Talwar and Andrew Goldberg
Microsoft Research, Silicon Valley — Mountain View, CA, USA
{misard, vijayanp, jcurrey, uwieder, kunal, goldberg}@microsoft.com

ABSTRACT

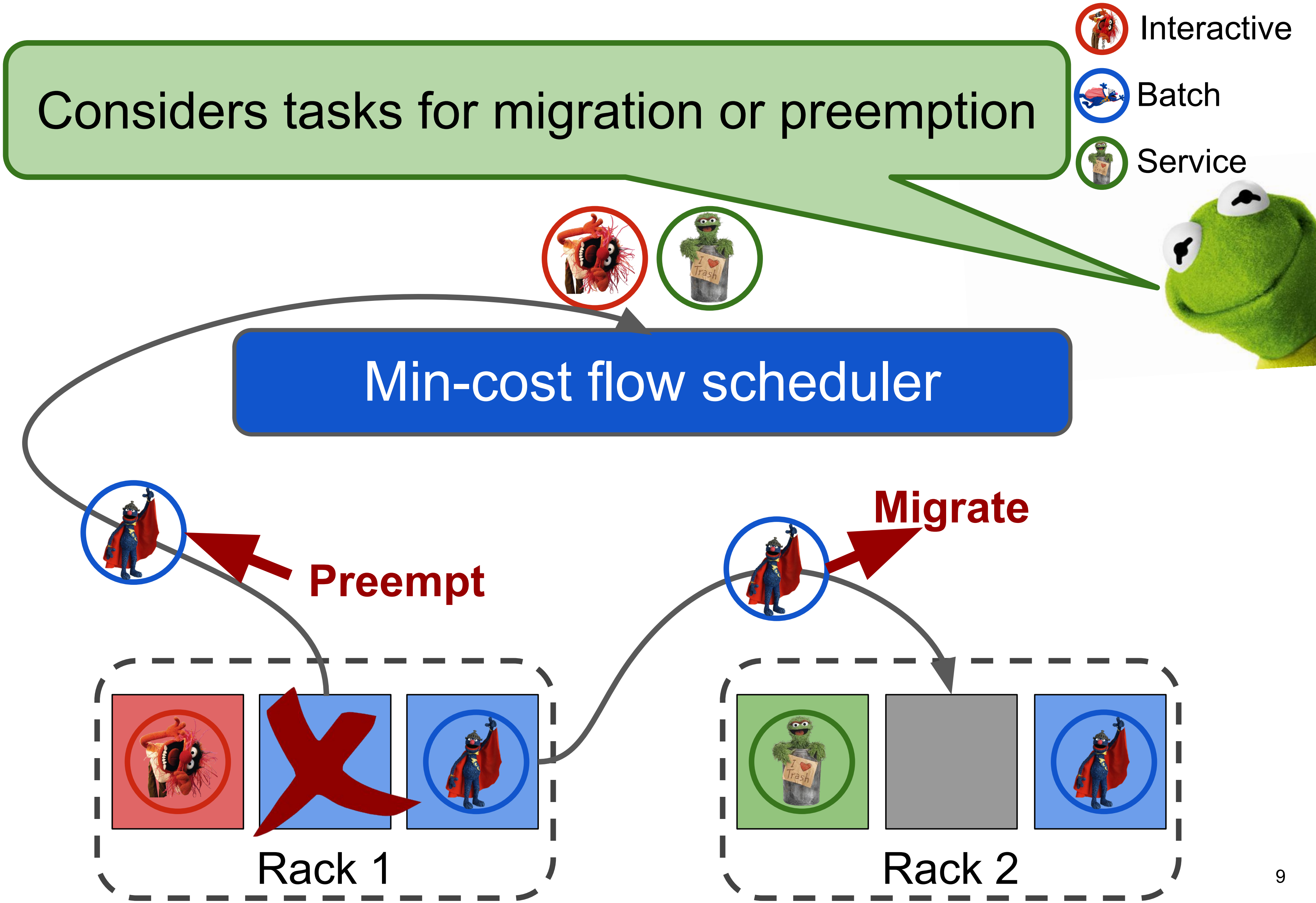
This paper addresses the problem of scheduling concurrent jobs on clusters where application data is stored on the computing nodes. This setting, in which scheduling computations close to their data is crucial for performance, is increasingly common and arises in systems such as MapReduce, Hadoop, and Dryad as well as many grid-computing environments. We argue that data-intensive computation benefits from a fine-grain resource sharing model that differs from the coarser semi-static resource allocations implemented by most existing cluster computing architectures. The problem of

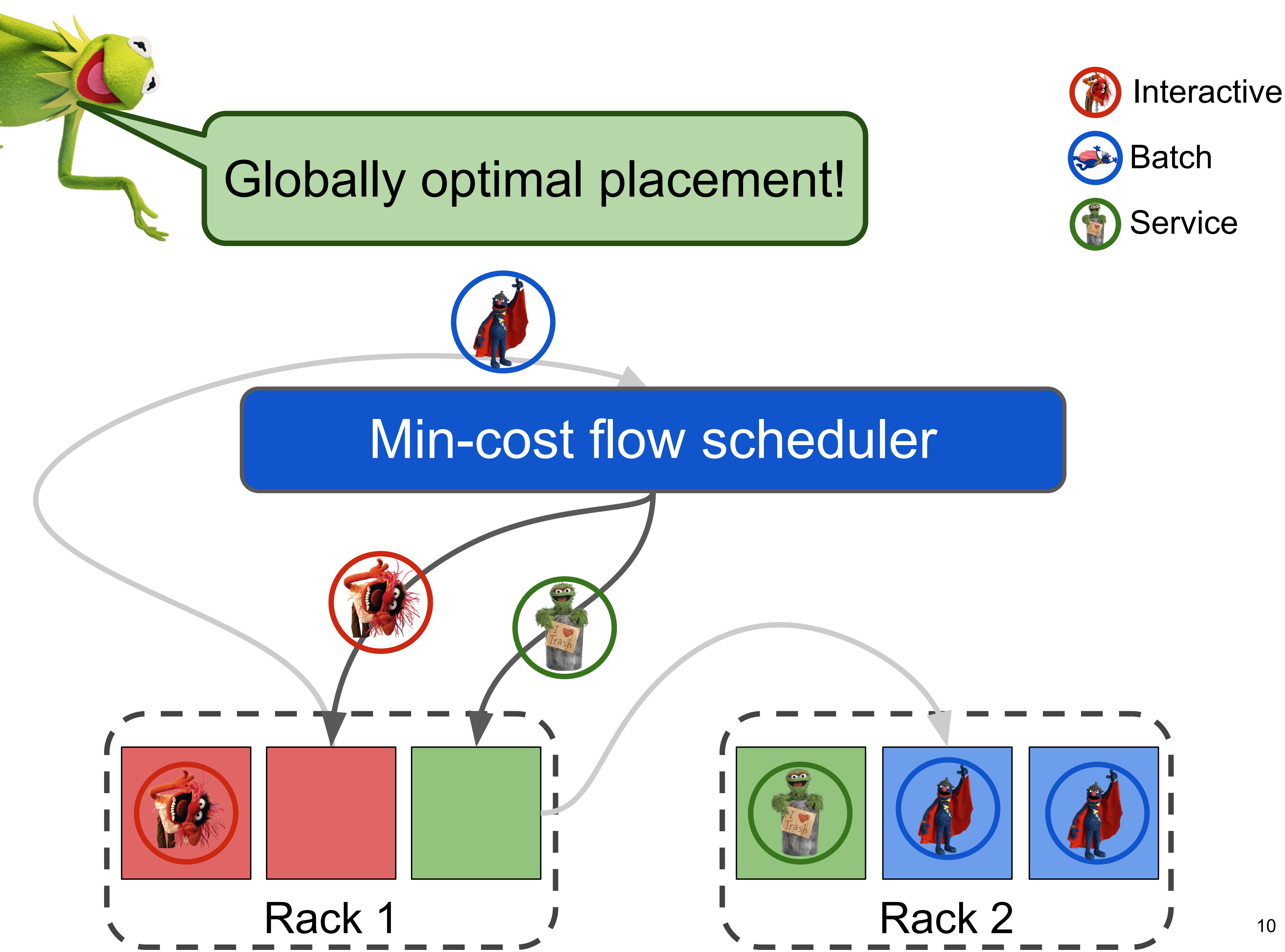



Preference for first rack
Me too!
Interactive
Batch
Service
Min-cost flow scheduler
Rack 1
Rack 2



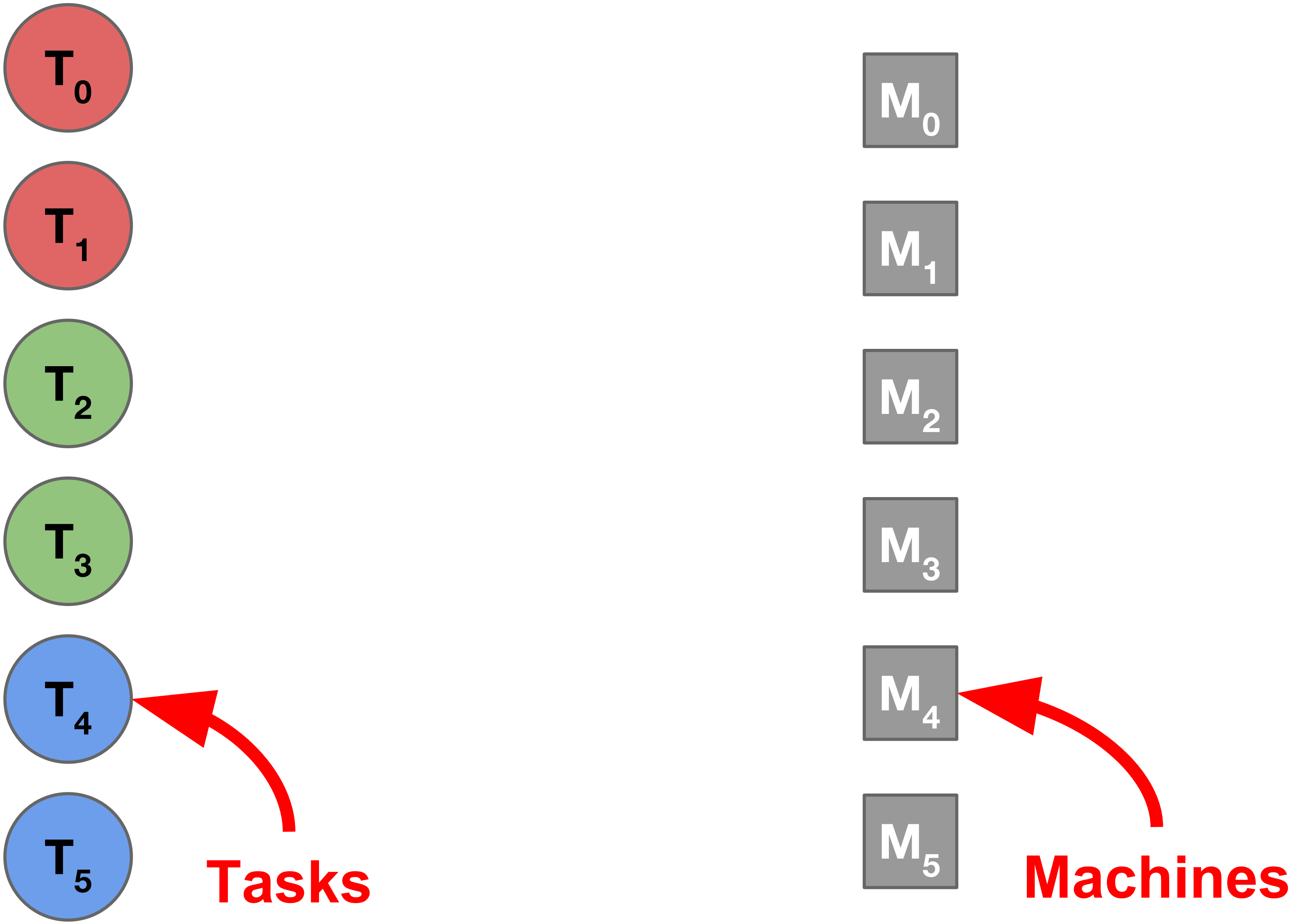
Schedules all tasks at the same time
Interactive
Batch
Service
Min-cost flow scheduler
Rack 1
Rack 2


Interactive
Batch
Service
Considers tasks for migration or preemption
Min-cost flow scheduler
Migrate
Preempt
Rack 1
Rack 2

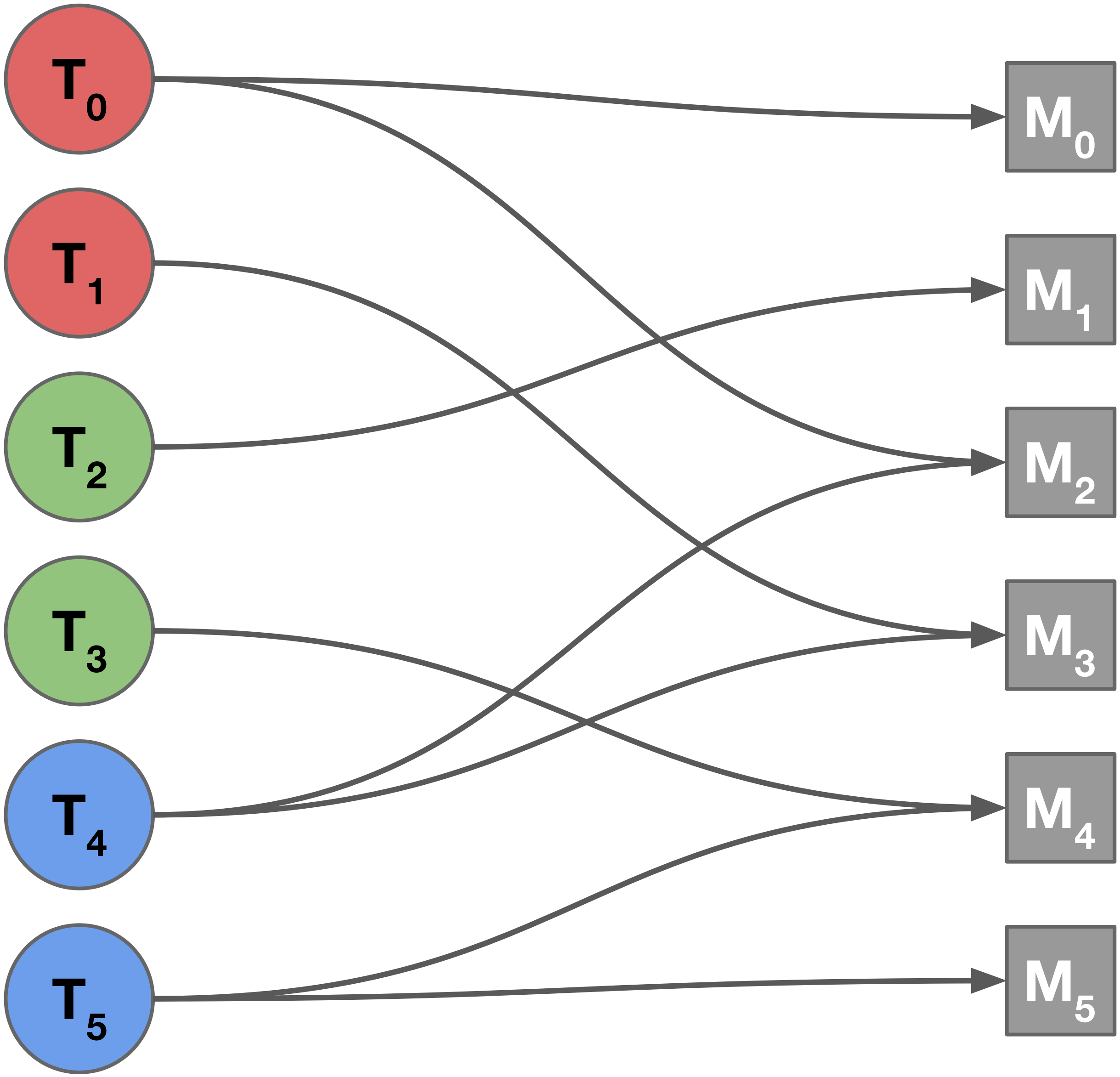


Globally optimal placement!
Interactive
Batch
Service
Min-cost flow scheduler
Rack 1
Rack 2

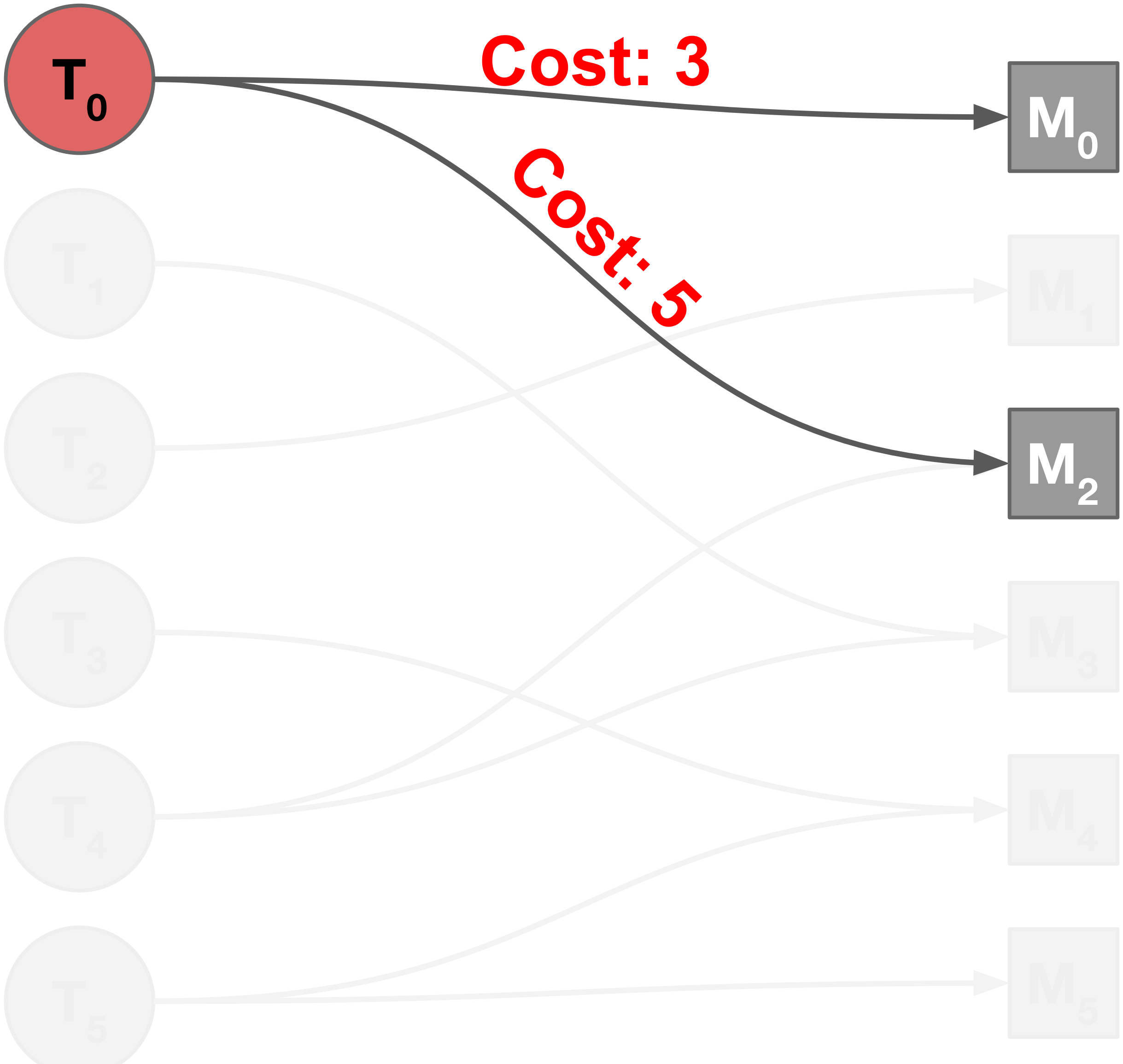
Introduction to min-cost flow scheduling



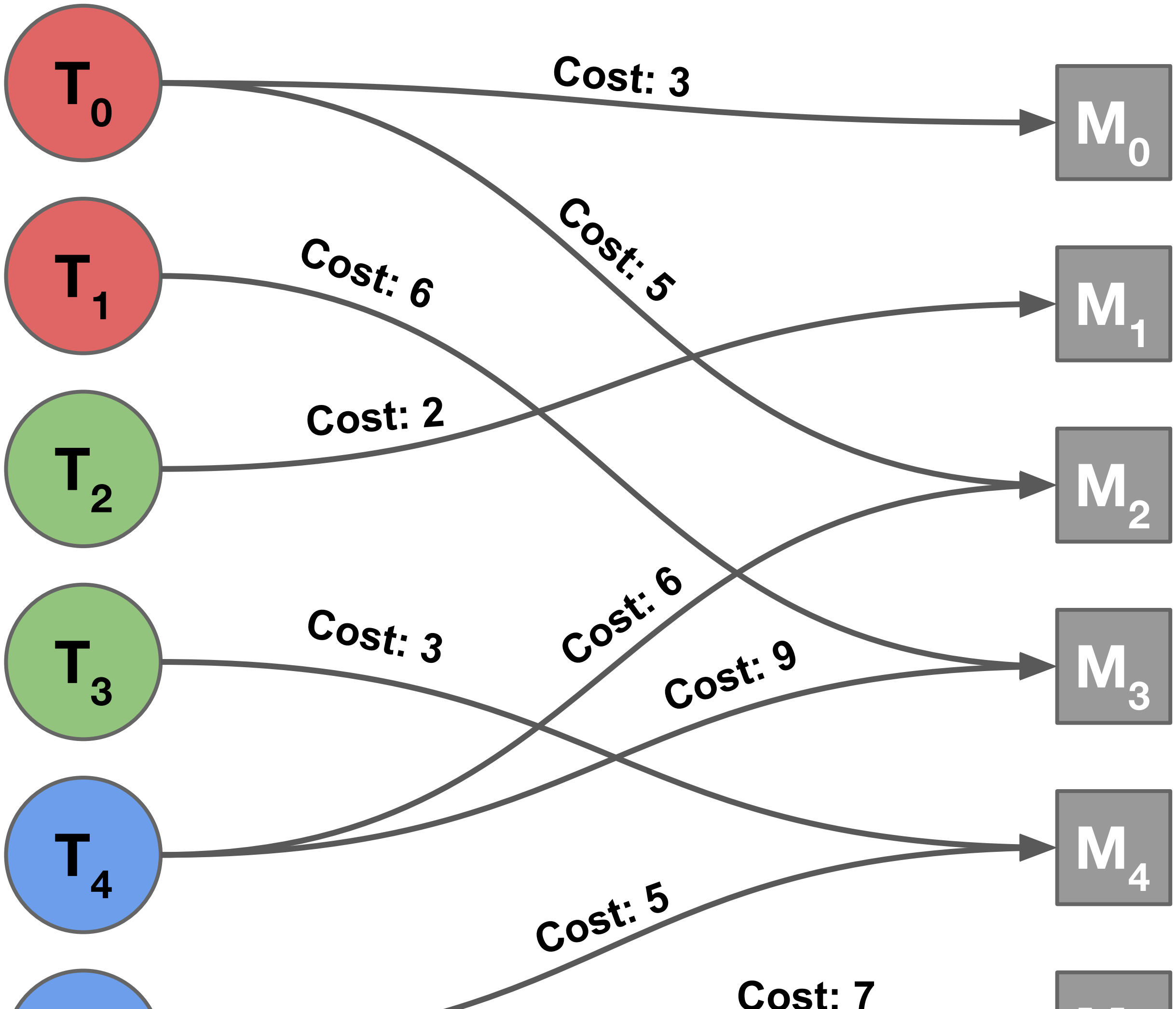
Introduction to min-cost flow scheduling



Introduction to min-cost flow scheduling

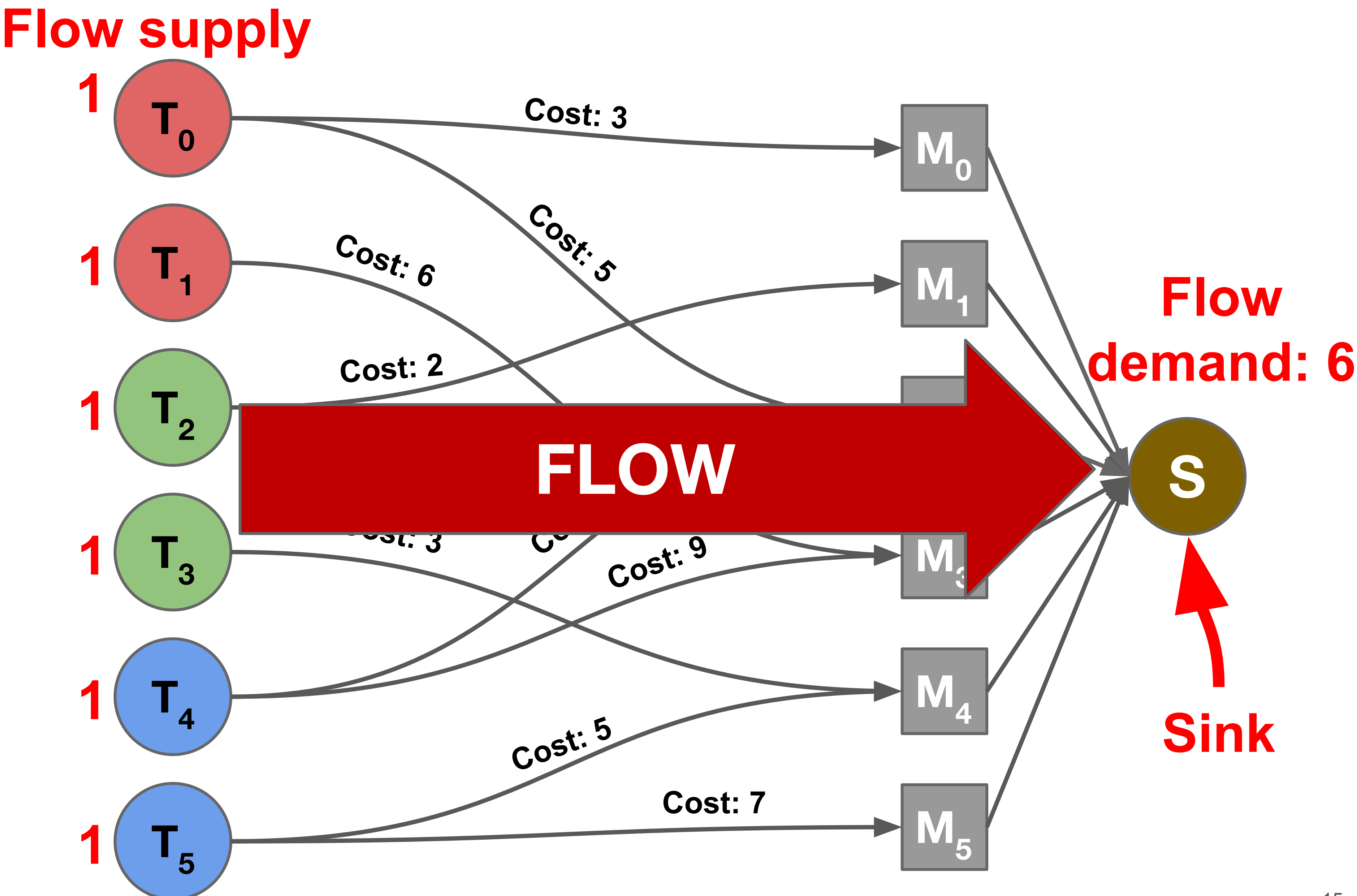


Introduction to min-cost flow scheduling

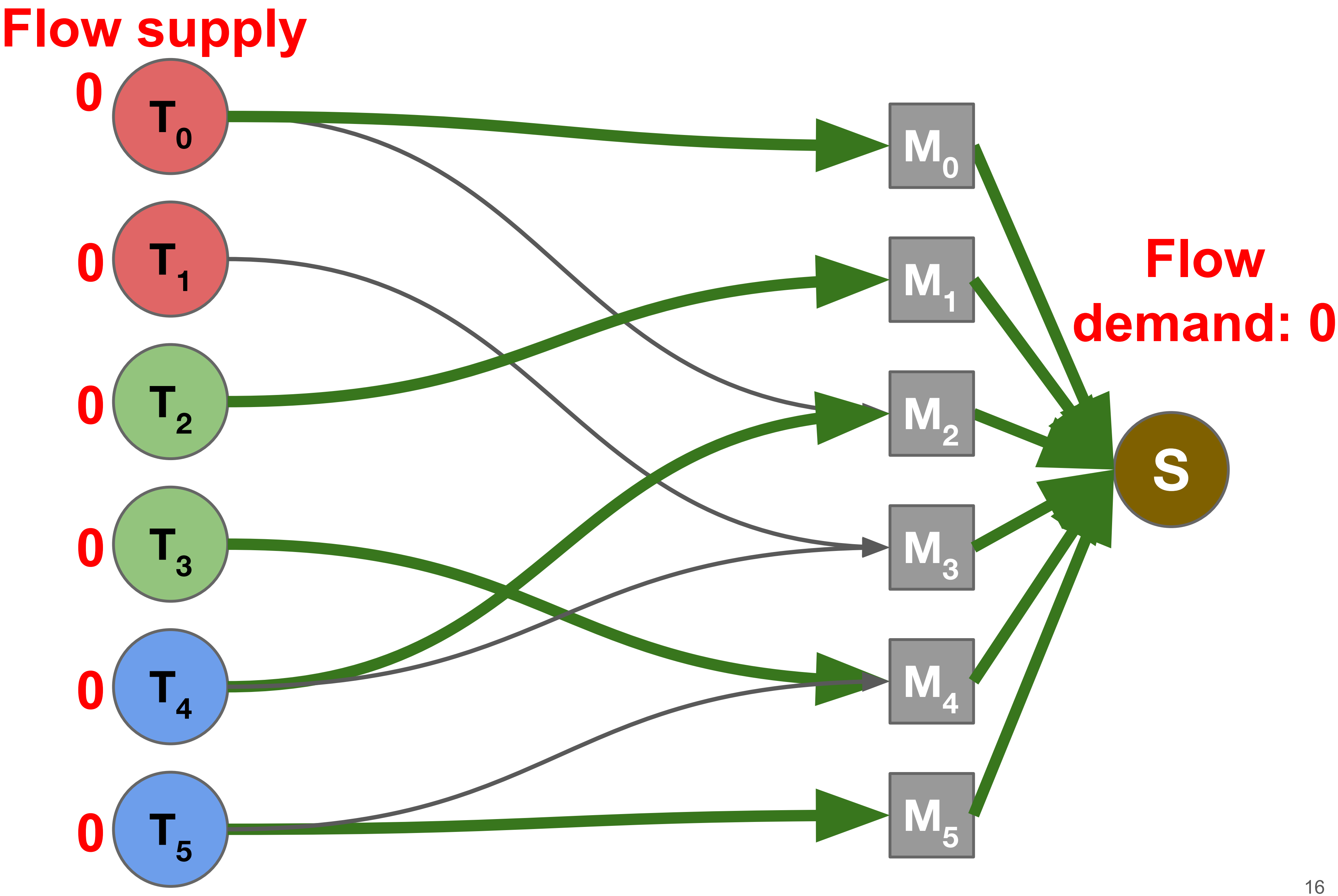


Min-cost flow places tasks with minimum overall cost

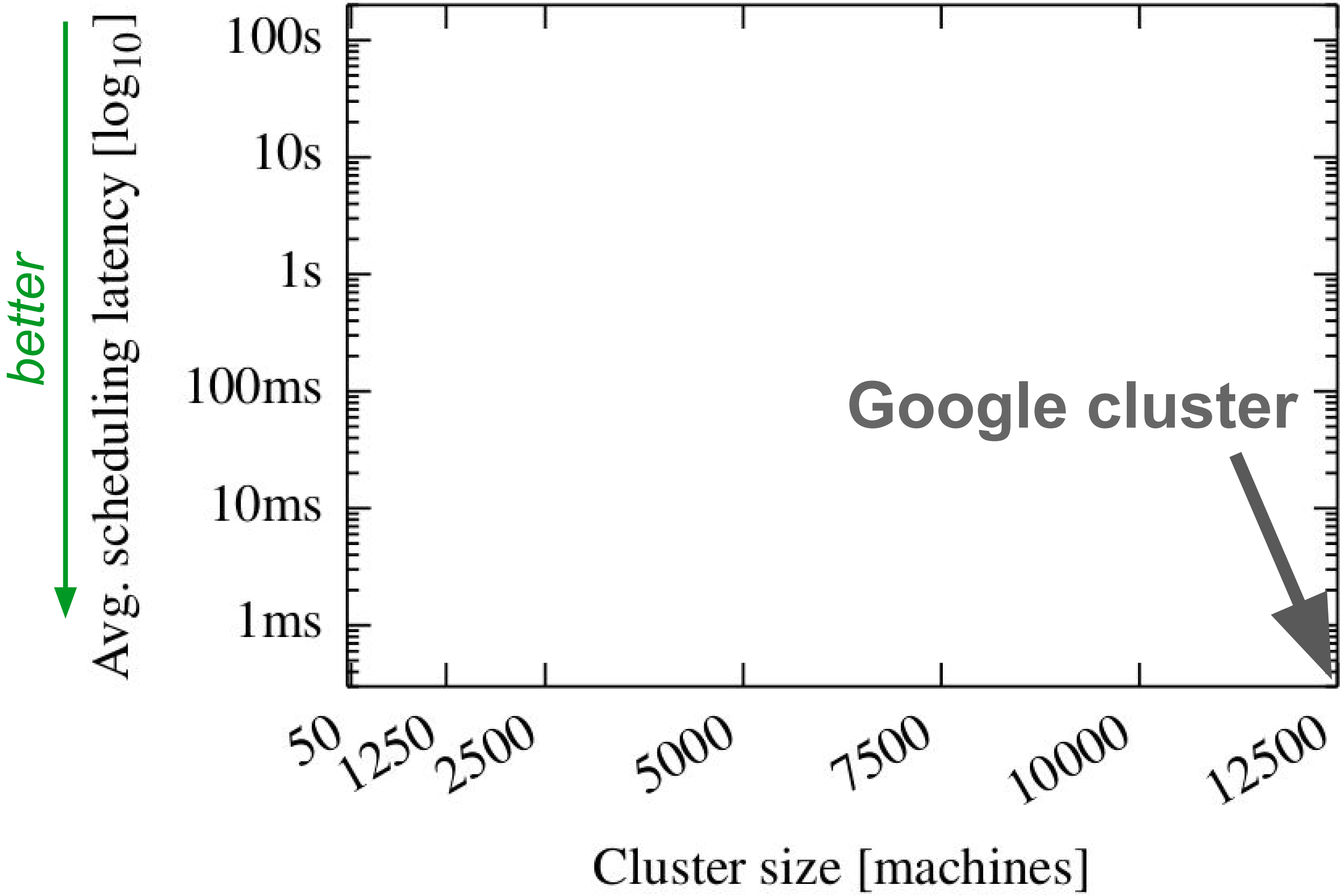
Introduction to min-cost flow scheduling



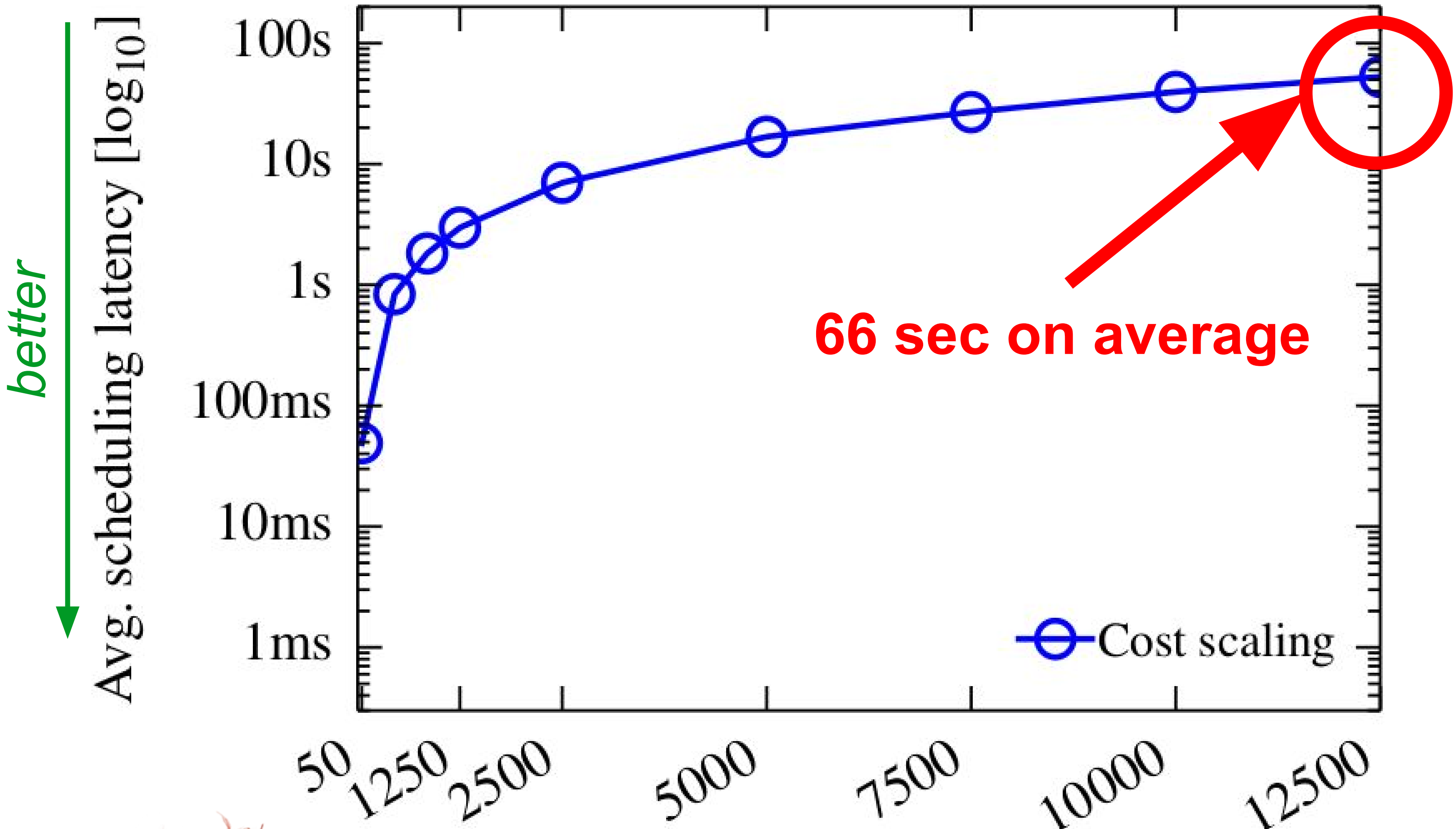
Introduction to min-cost flow scheduling



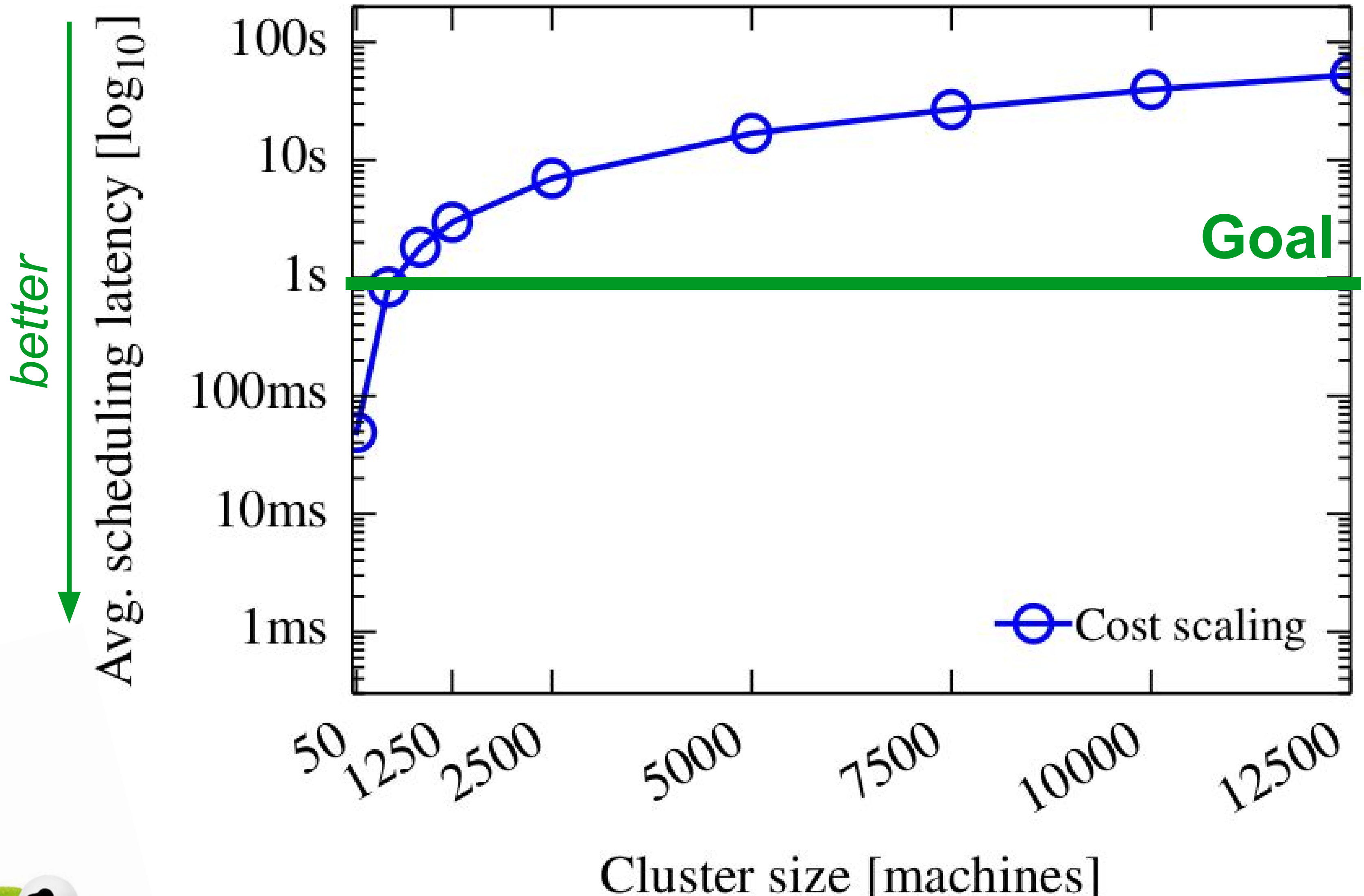
How well does the Quincy approach scale?



Simulated Quincy using Google trace, 50% utilization



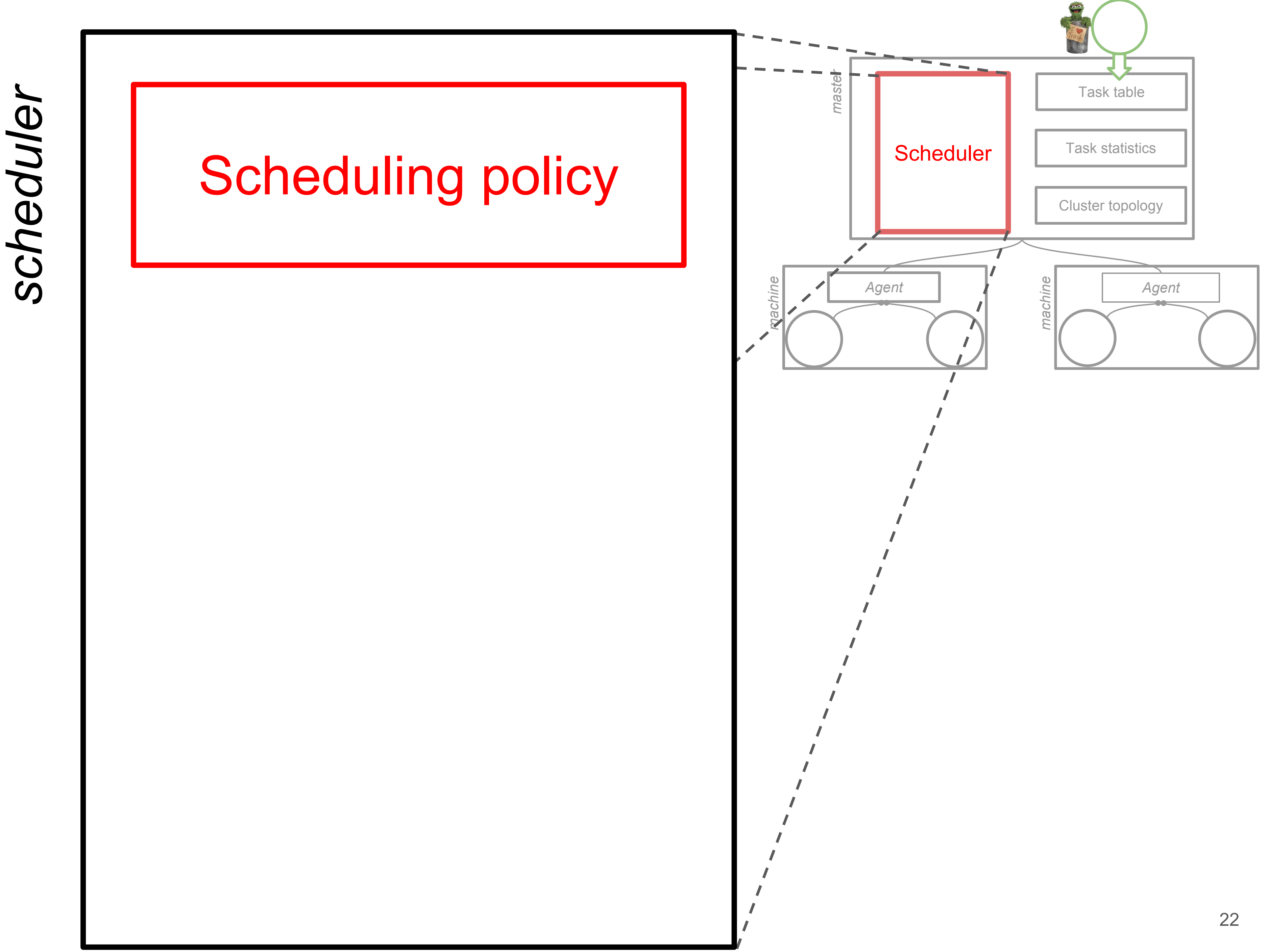
Too slow! **30%** of tasks wait to be scheduled for over **33%** of their runtime and waste resources



Goal: sub-second scheduling latency in common case

Contributions

- **Low task scheduling latency**
  - Uses best suited min-cost flow algorithm
  - Incrementally recomputes the solution
- **Good task placement**
  - Same optimal placements as Quincy
  - Customizable scheduling policies

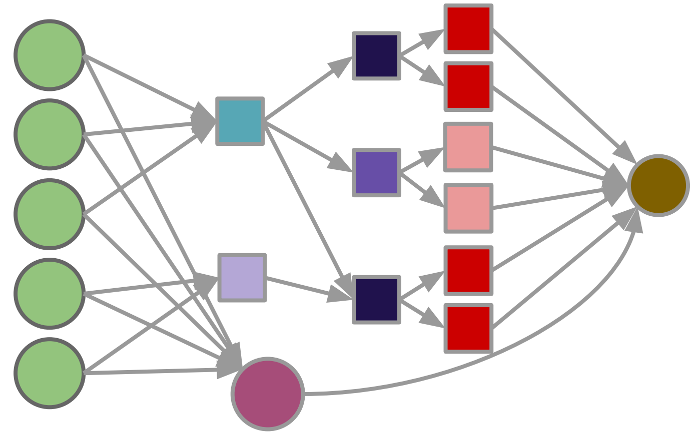

scheduler
Scheduling policy
master
Scheduler
Task table
Task statistics
Cluster topology
machine
Agent
machine
Agent

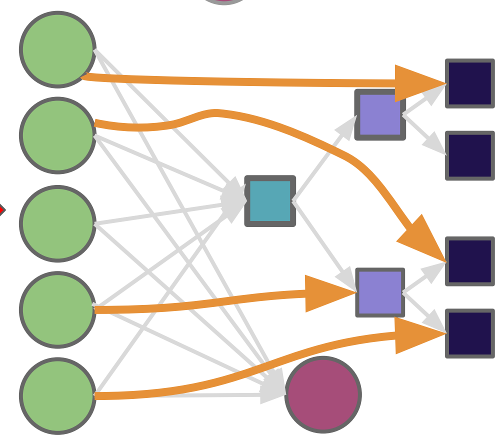
Specifying scheduling policies

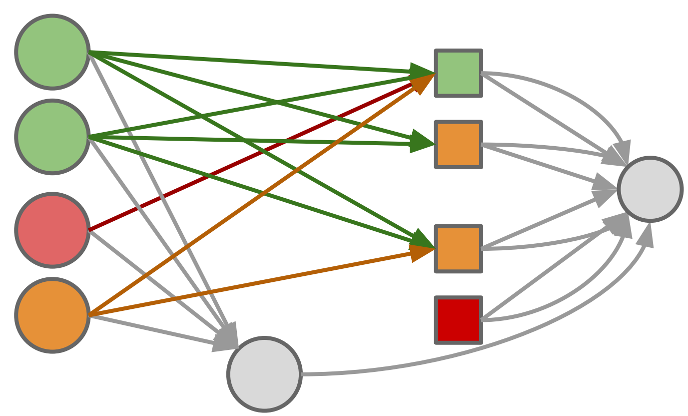
```
class QuincyPolicy {

  Cost_t TaskToResourceNodeCost(
   TaskID_t task_id) {
   return task_unscheduled_time *
     quincy_wait_time_factor;
  }
  ...
}
```

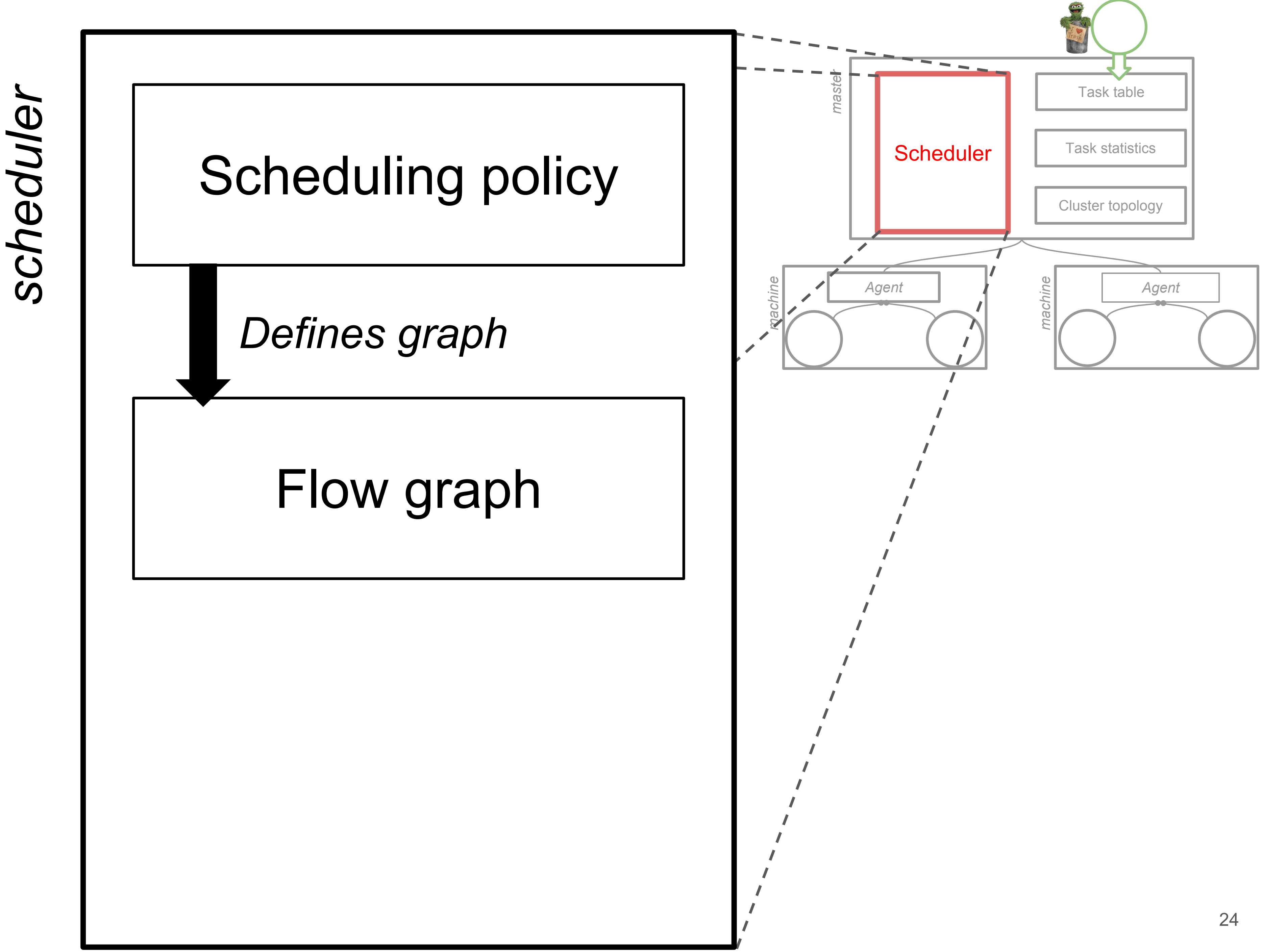
Defines flow graph

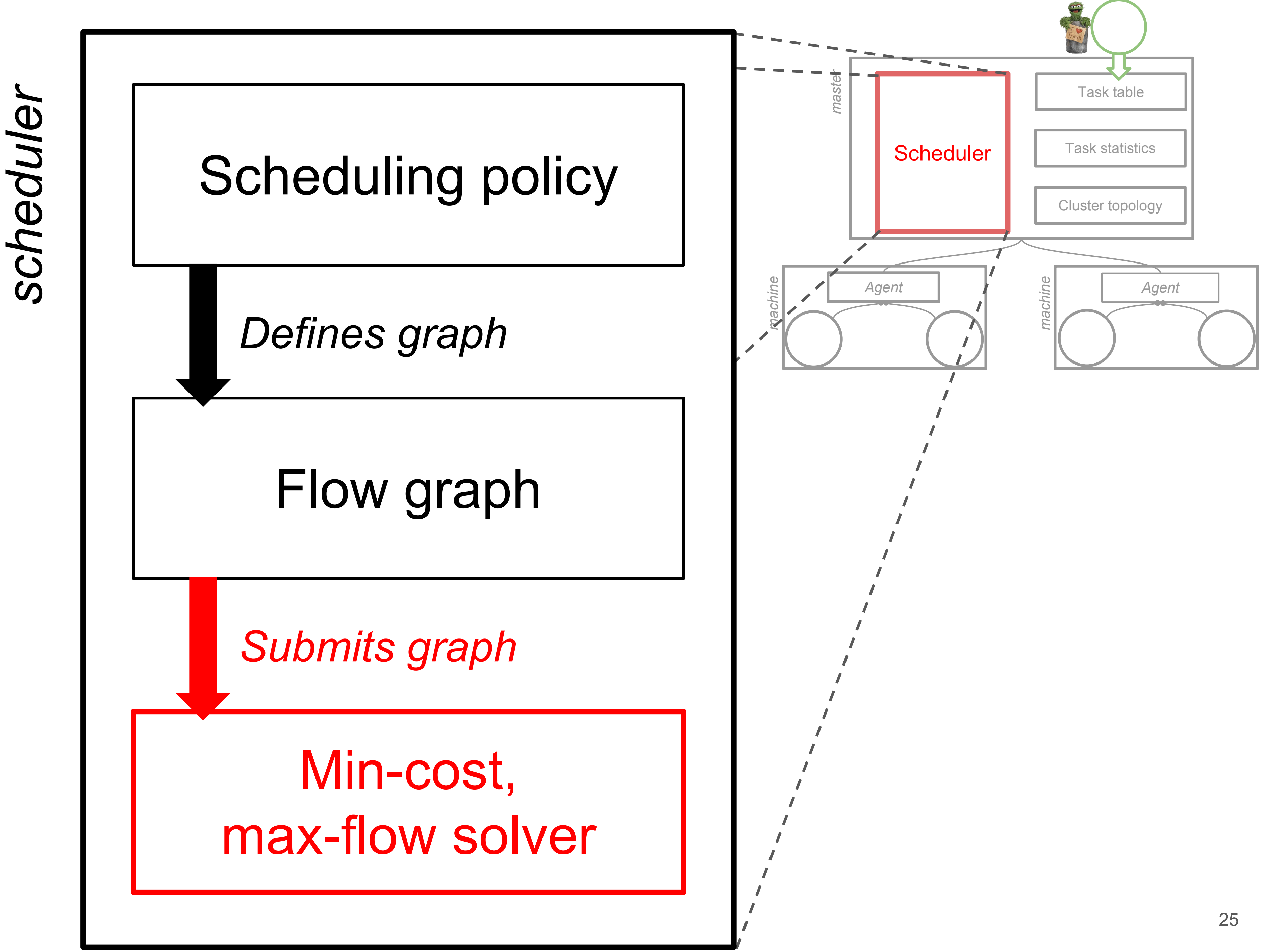


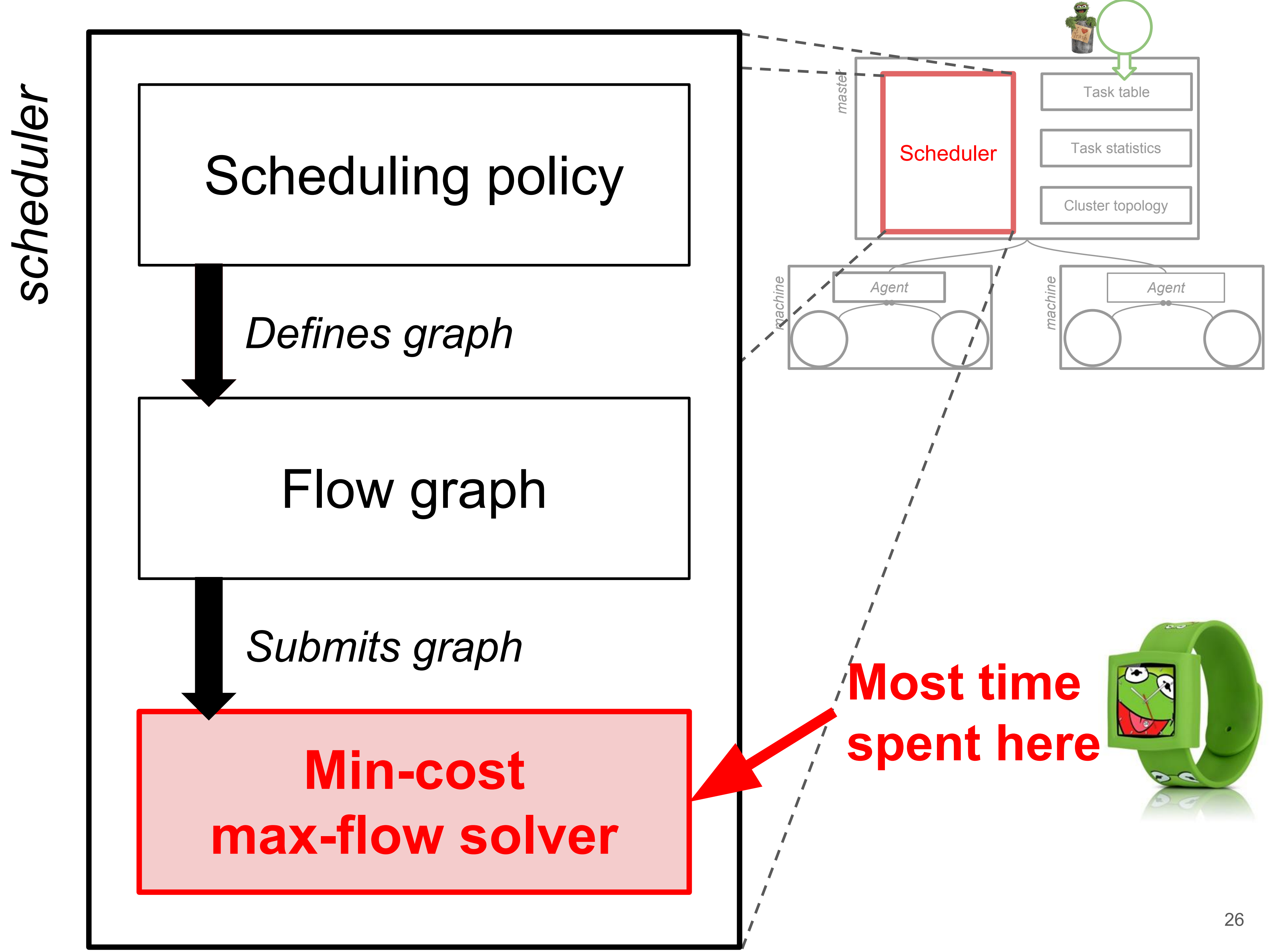




N.B: More details in the paper.


scheduler
Scheduling policy
Defines graph
Flow graph
master
Scheduler
Task table
Task statistics
Cluster topology
machine
Agent
machine
Agent


scheduler
Scheduling policy
Defines graph
Flow graph
Submits graph
Min-cost,
max-flow solver
master
Scheduler
Task table
Task statistics
Cluster topology
machine
Agent
machine
Agent


scheduler
Scheduling policy
Defines graph
Flow graph
Submits graph
Min-cost
max-flow solver
Most time
spent here
master
Scheduler
Task table
Task statistics
Cluster topology
machine
Agent
machine
Agent

Algorithm	Worst-case complexity
$\boxed{\text{Cost scaling}}$	$O(V^2 E log(VC))$

Used by Quincy

E: number of arcs

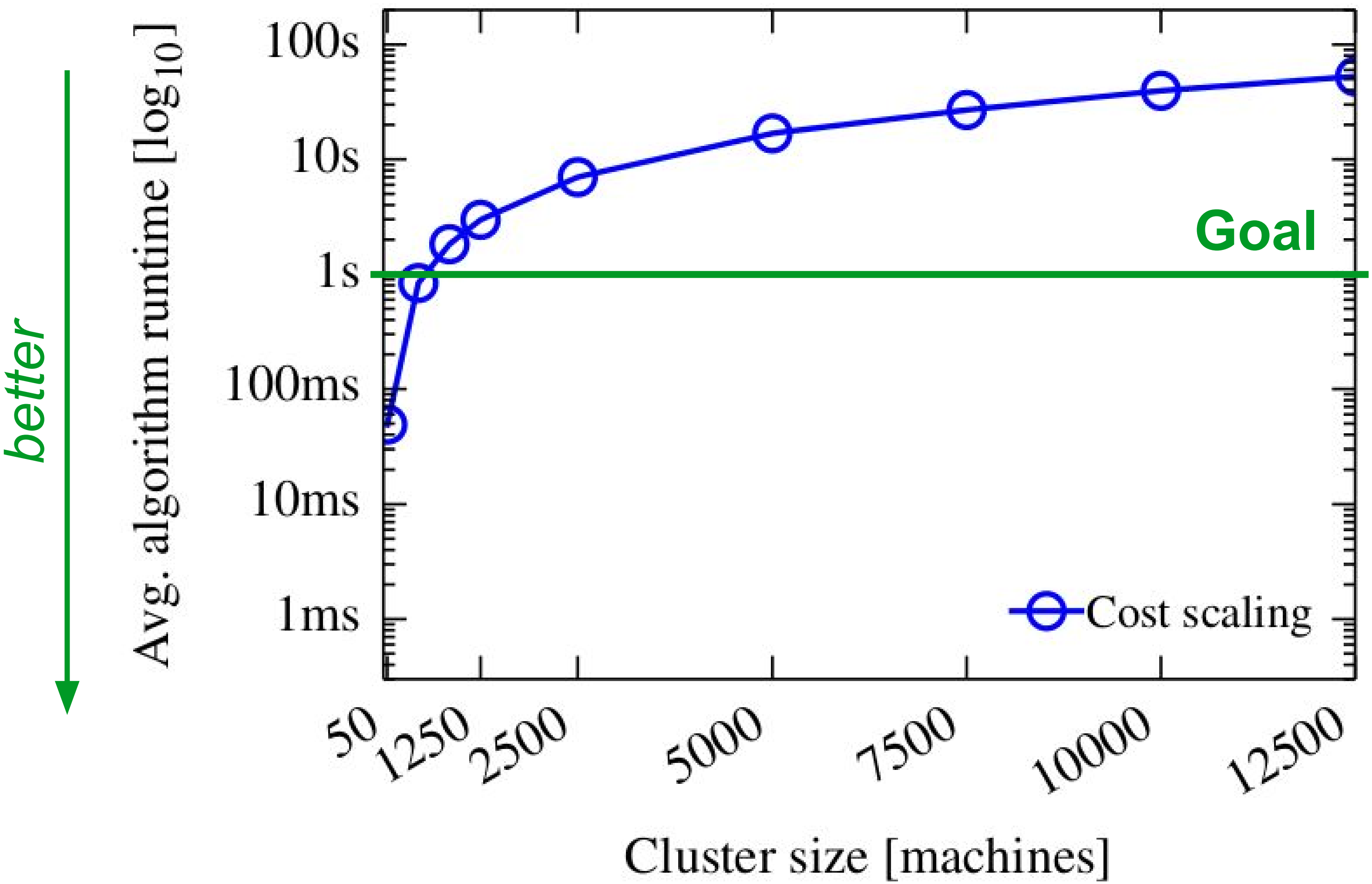
V: number of nodes

U: largest arc capacity

C: largest cost value

$E > V > C \cong U$

Subsampled Google trace, 50% slot utilization [Quincy policy]



Cost scaling is too slow beyond 1,000 machines

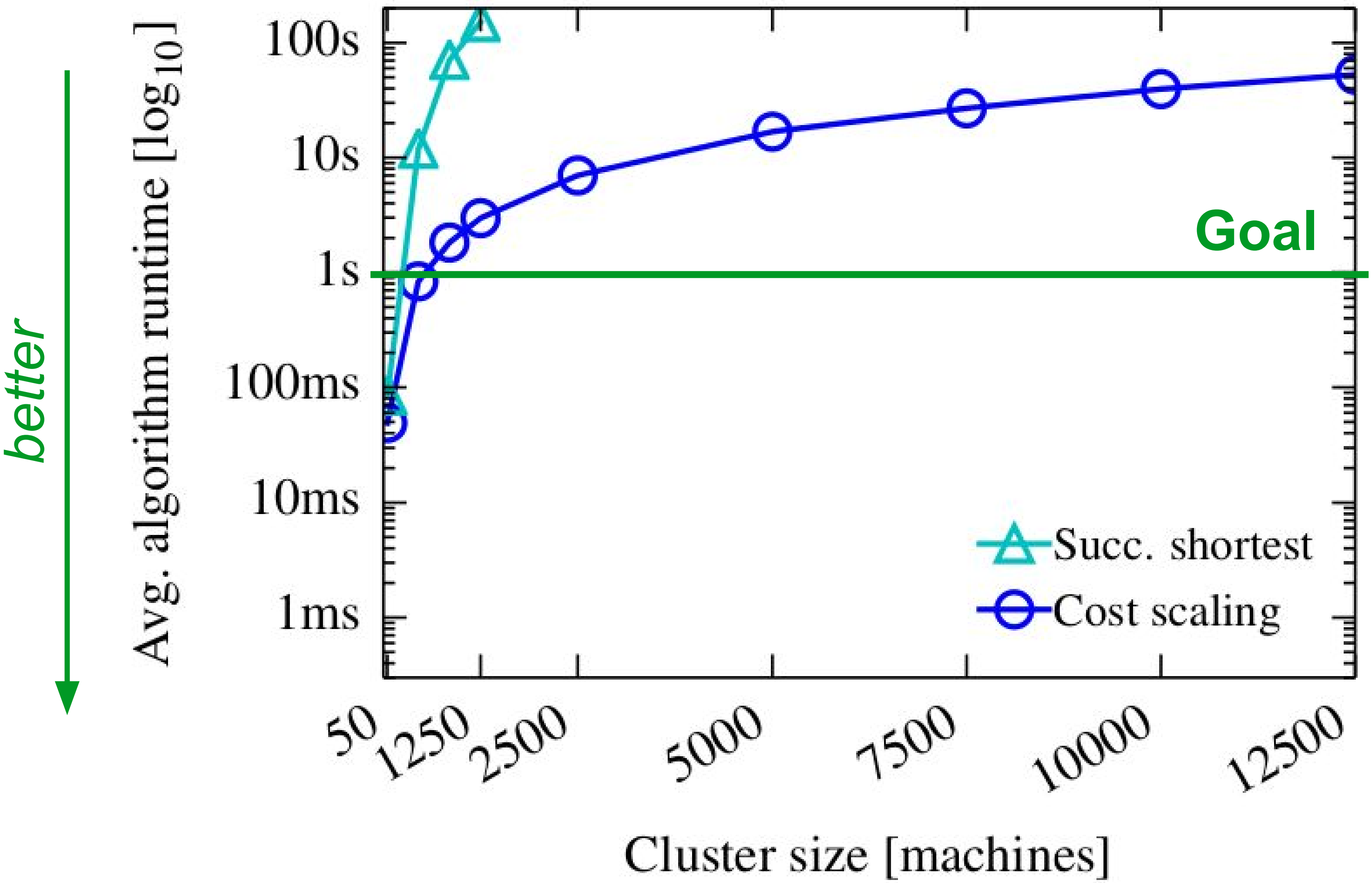
Algorithm	Worst-case complexity
Cost scaling	$O(V^2 E\log(VC))$
Successive shortest path	$\boxed{O(V^2 U\log(V))}$

Lower worst-case complexity

E: number of arcs
V: number of nodes
U: largest arc capacity
C: largest cost value

$$E > V > C \cong U$$

Subsampled Google trace, 50% slot utilization [Quincy policy]



Successive shortest path only scales to ~100 machines

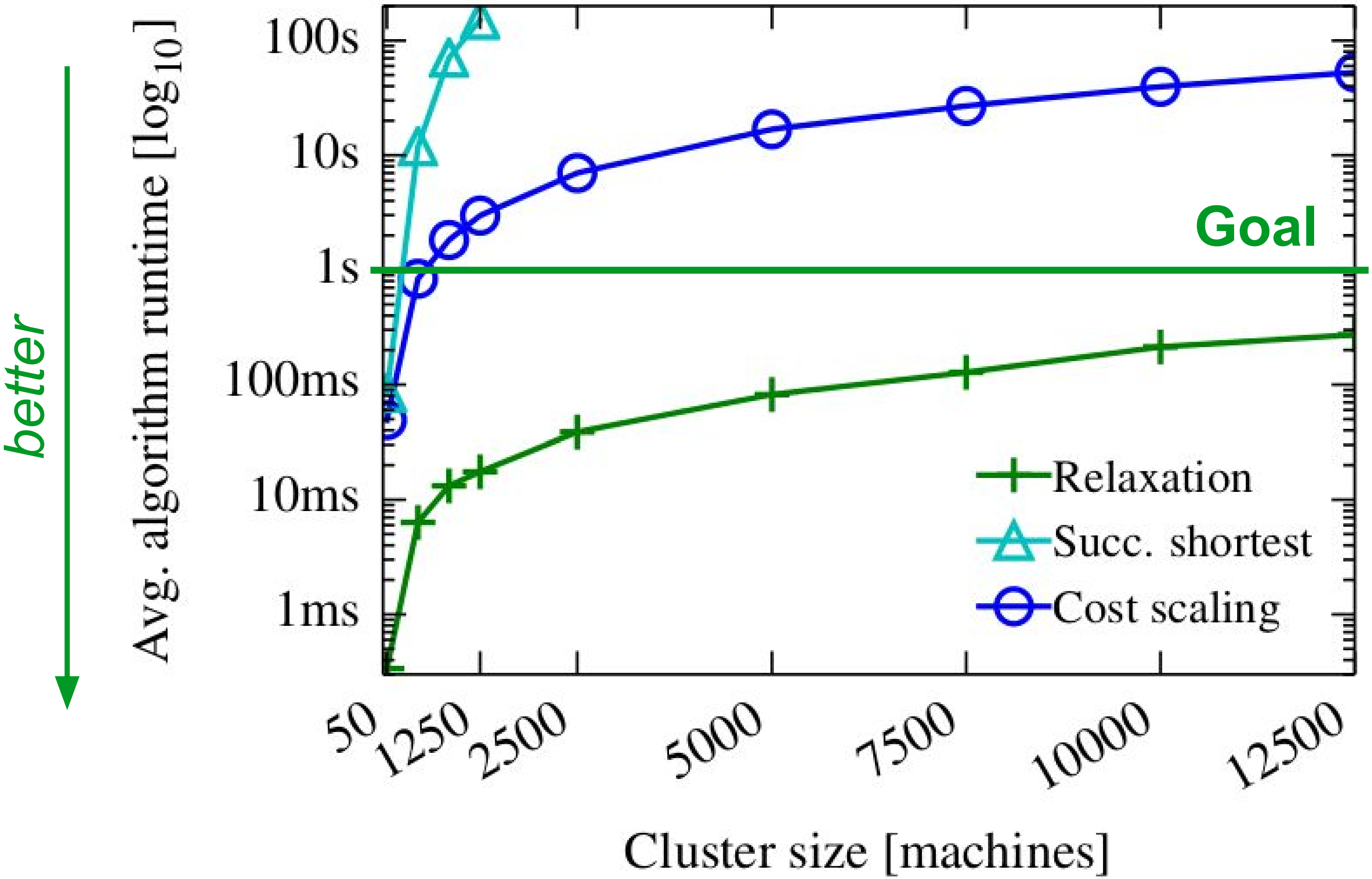
Algorithm	Worst-case complexity
Cost scaling	$O(V^2 E log(VC))$
Successive shortest path	$O(V^2 U log(V))$
Relaxation	$\boxed{O(E^3 C U^2)}$

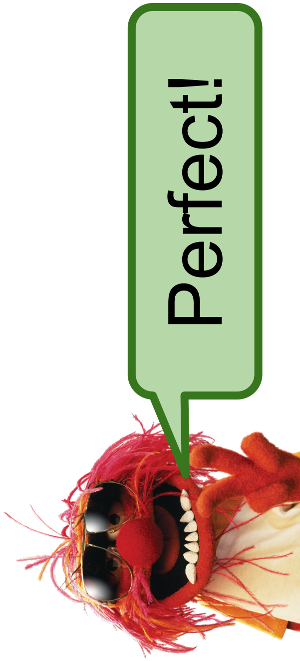
Highest complexity

E: number of arcs
V: number of nodes
U: largest arc capacity
C: largest cost value

$$E > V > C \cong U$$

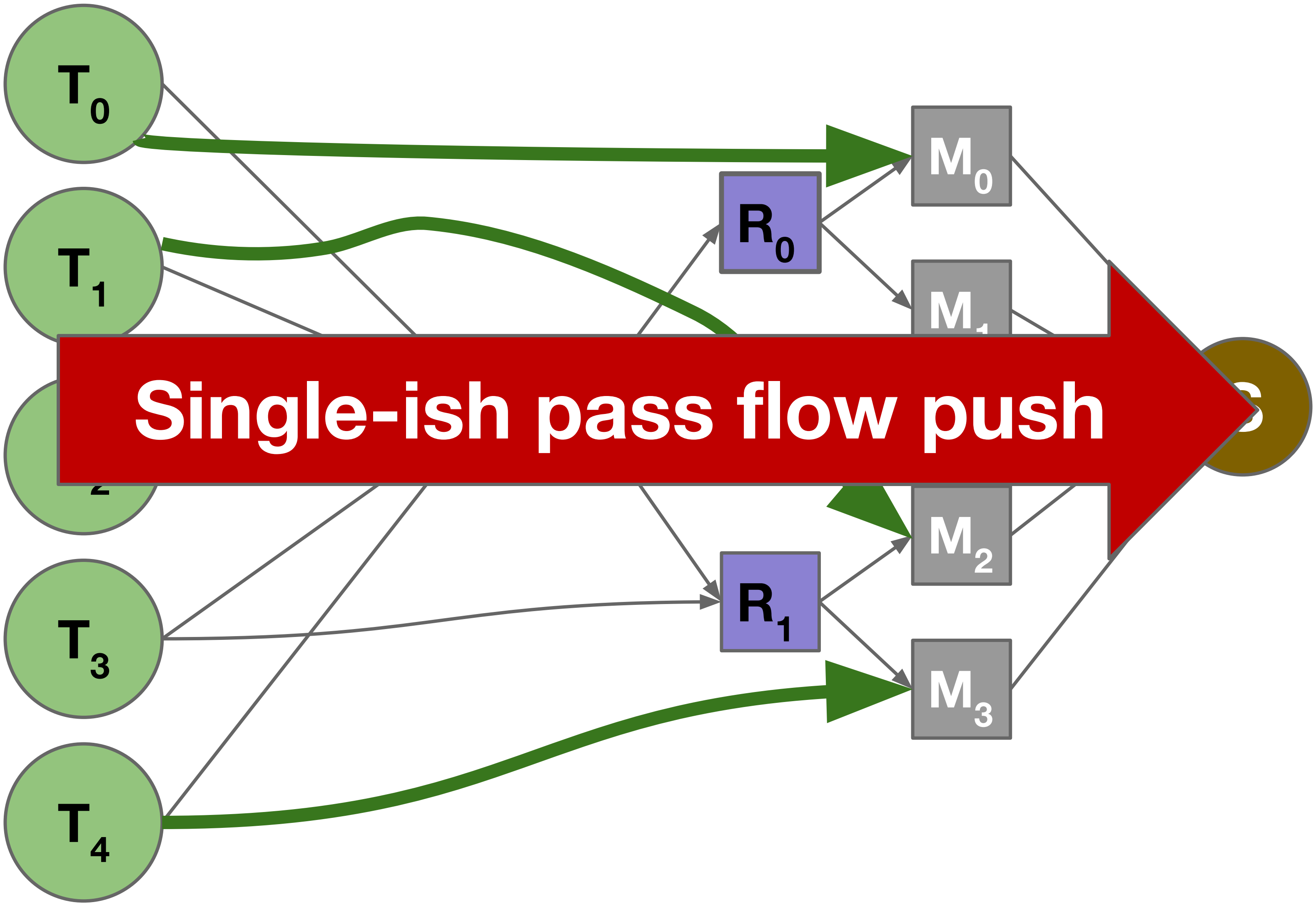
Subsampled Google trace, 50% slot utilization [Quincy policy]





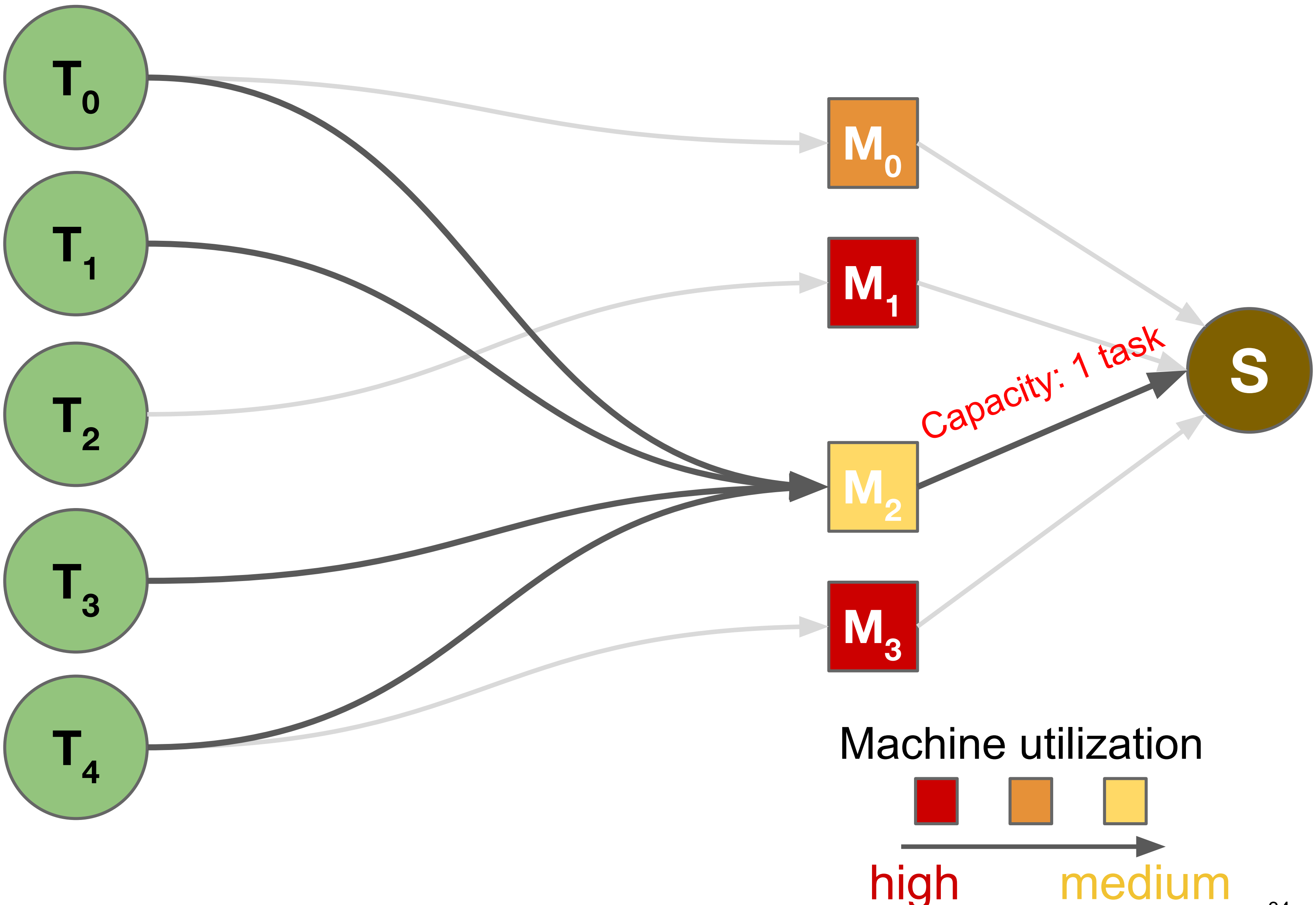
Relaxation meets our sub-second latency goal

Why is Relaxation fast?

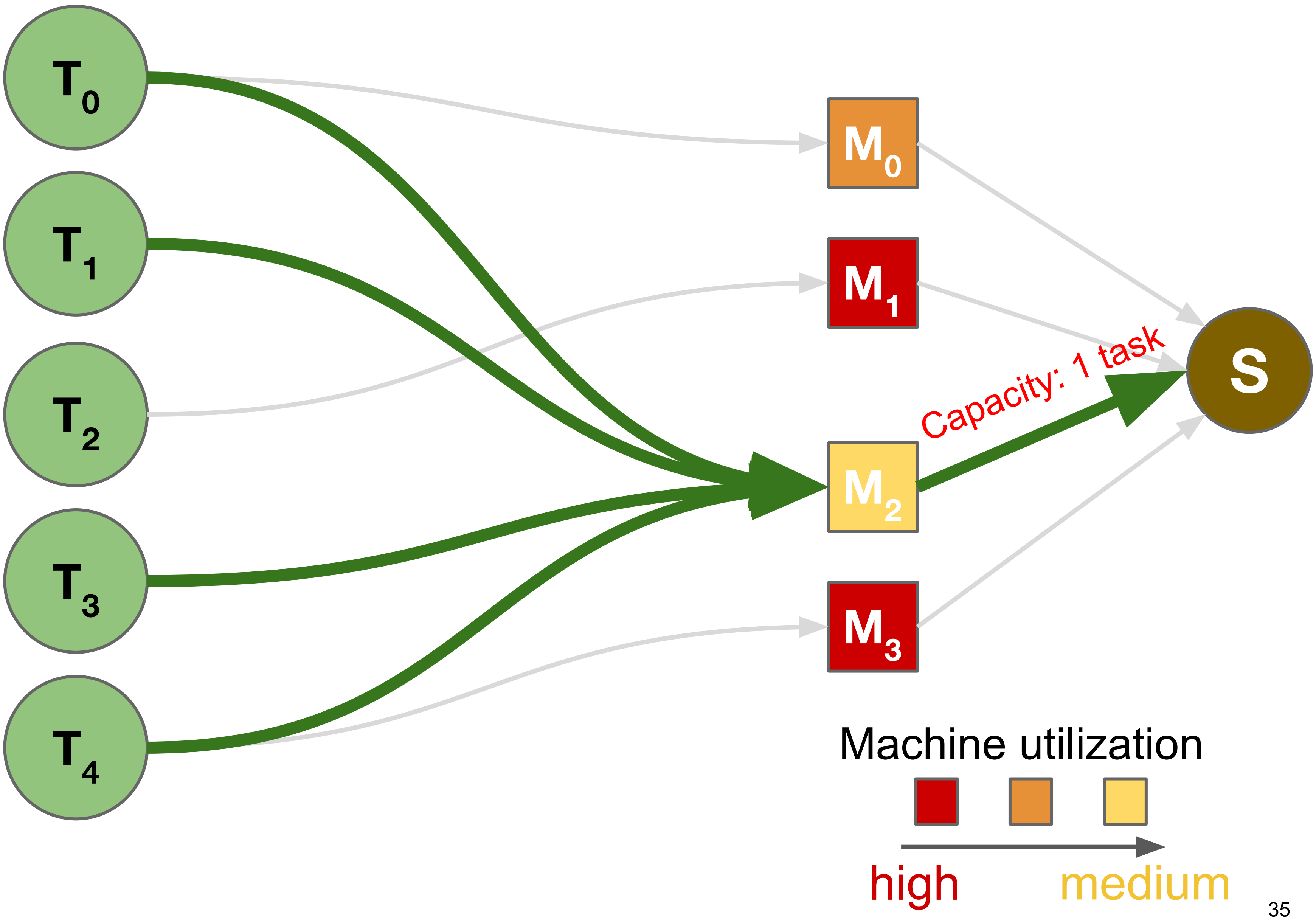


Relaxation is well-suited to the graph structure

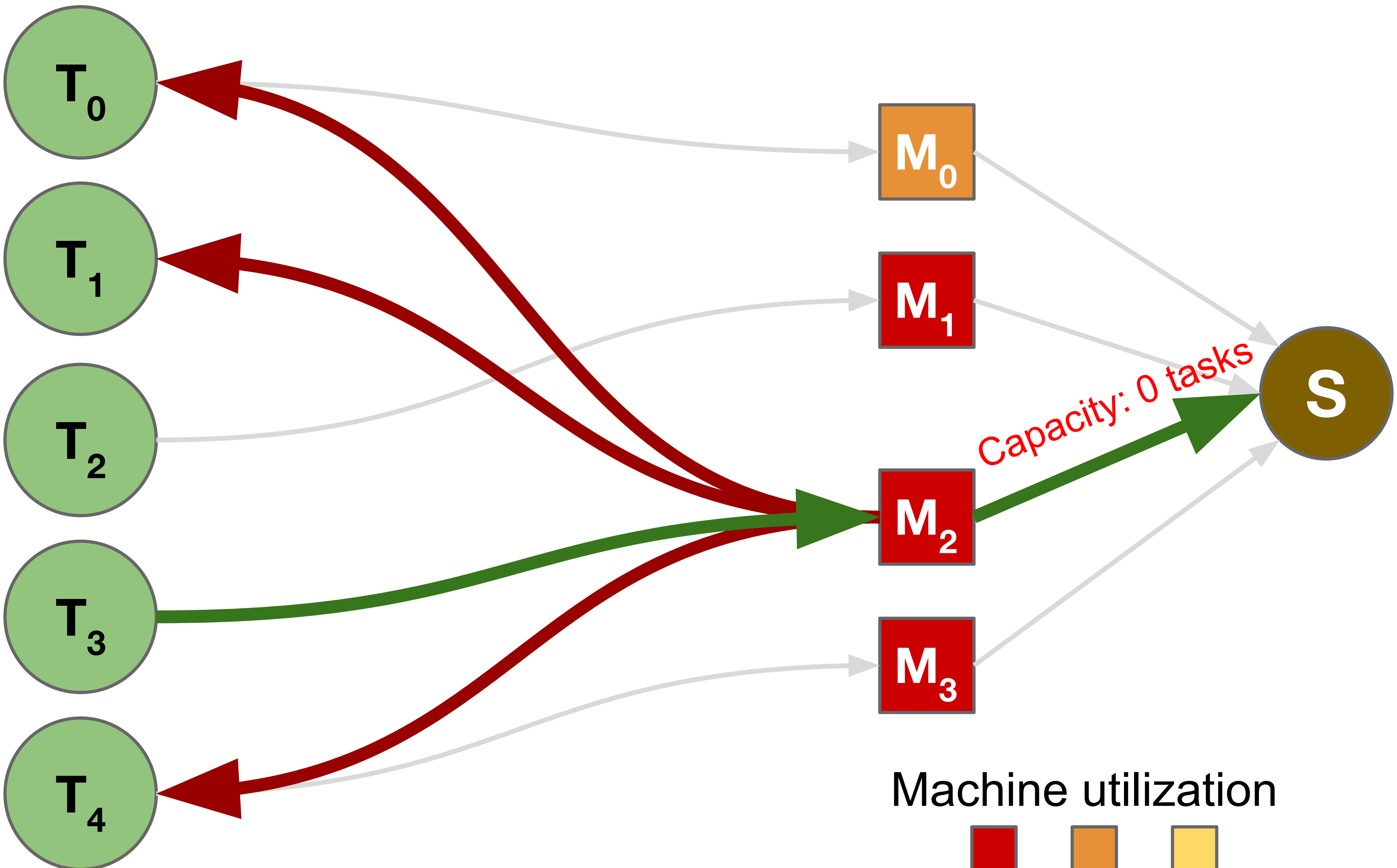
Relaxation suffers in pathological edge cases



Relaxation suffers in pathological edge cases



Relaxation suffers in pathological edge cases

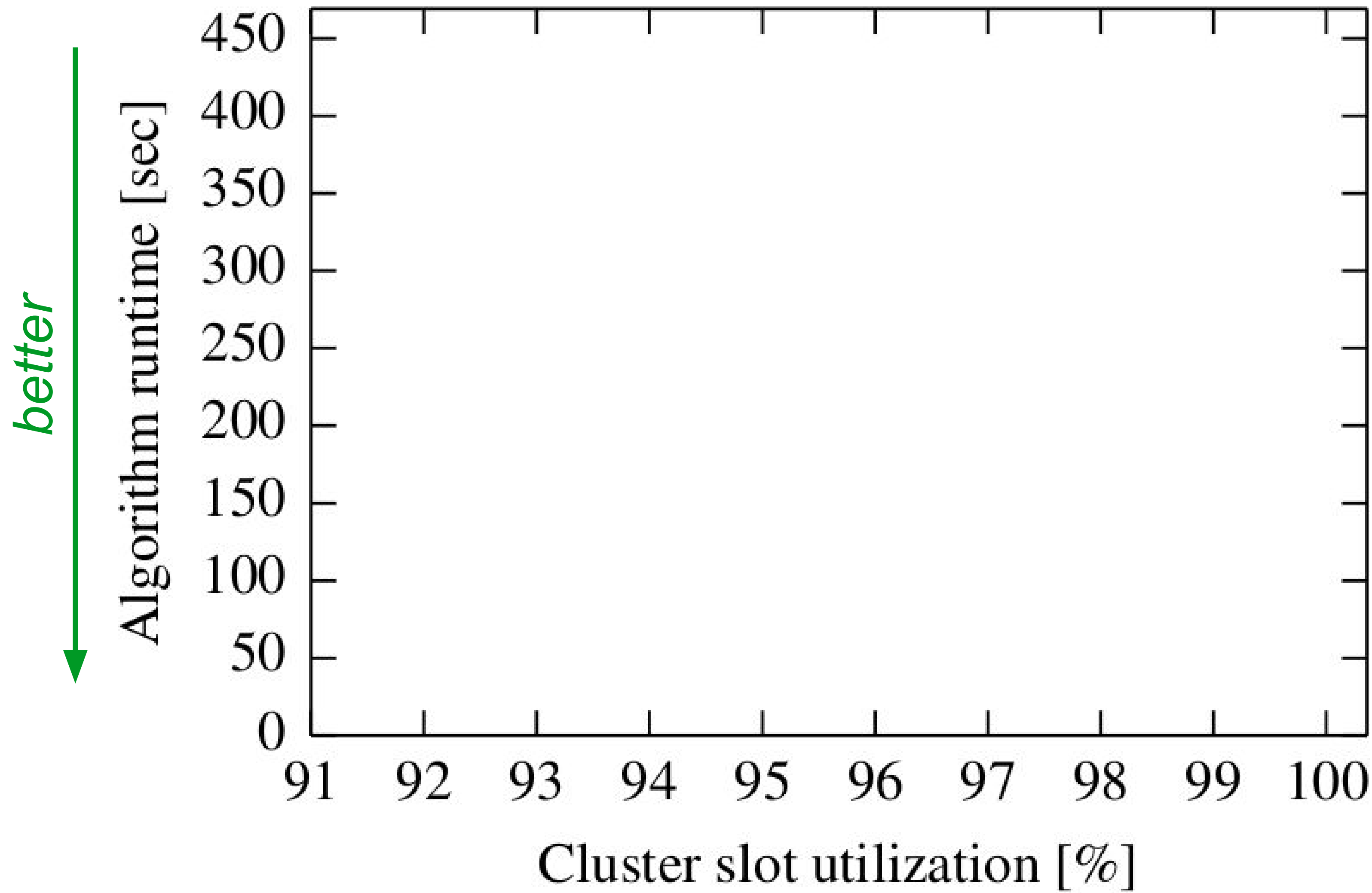


Relaxation cannot push flow in a single pass any more

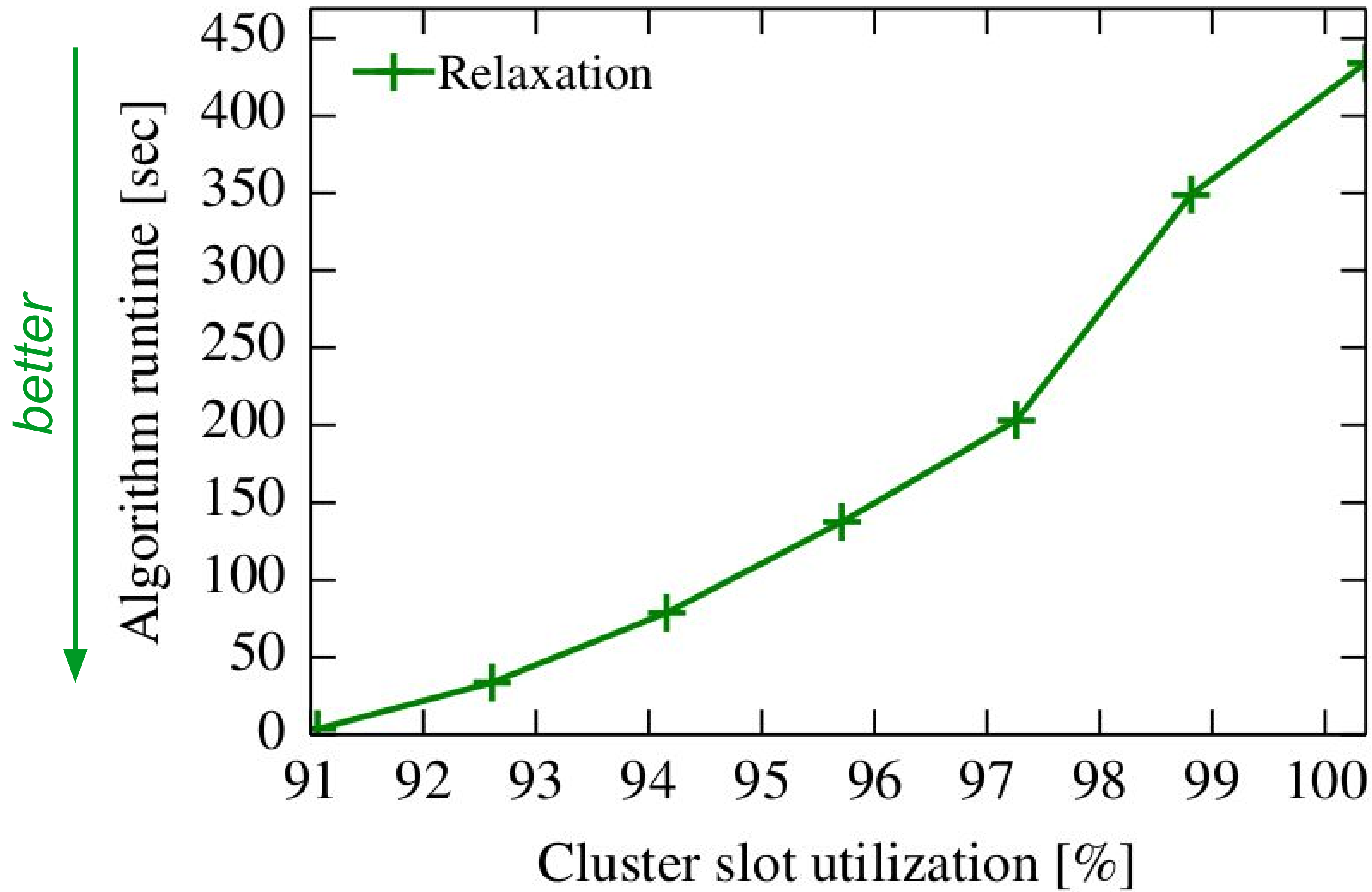
How bad does Relaxation's edge case get?

Experimental setup:
- Simulated 12,500 machine cluster
- Used the Quincy scheduling policy
- Utilization >90% to oversubscribed cluster

Quincy, 12,500 machines cluster, job of increasing size

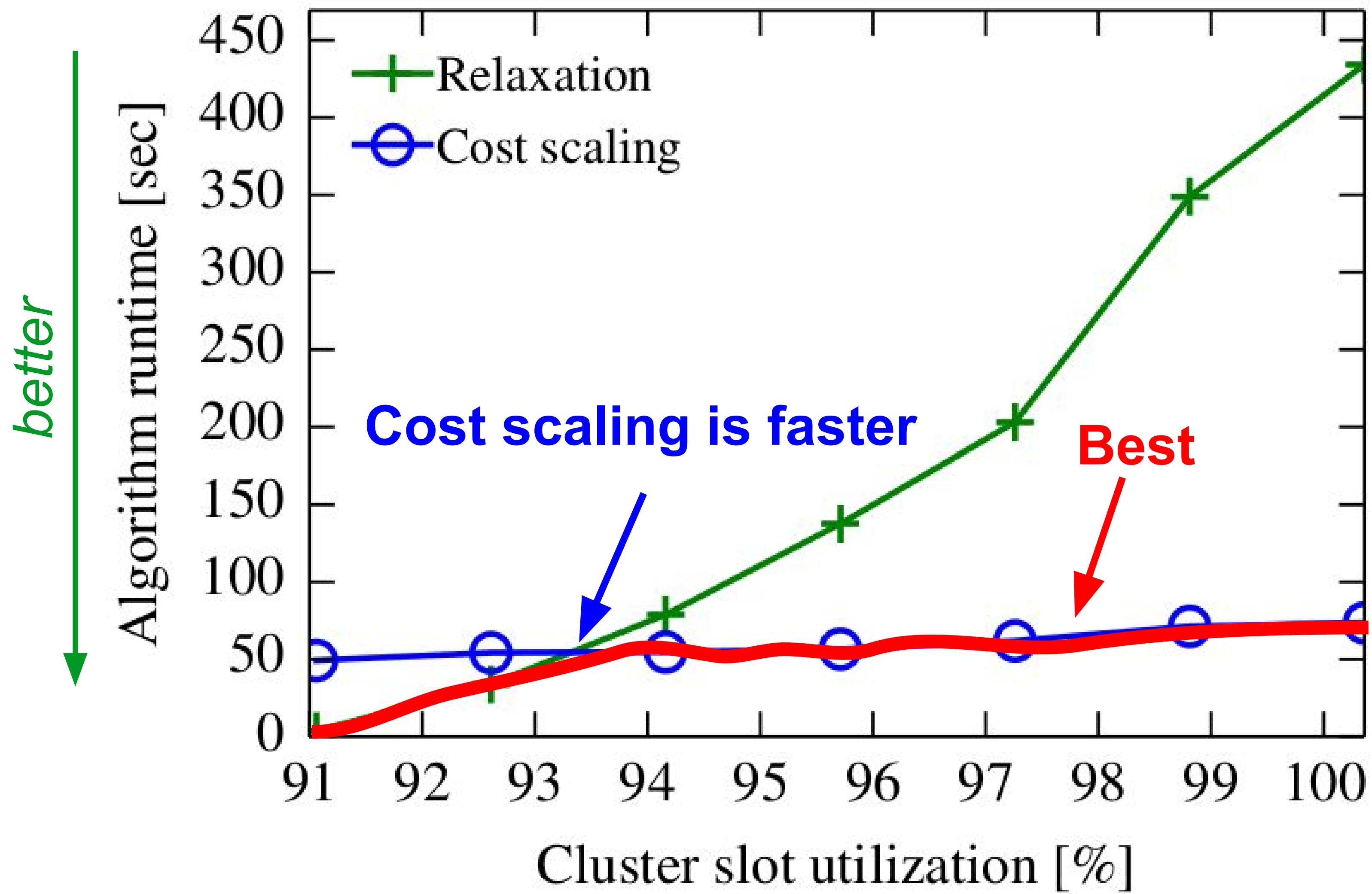


Quincy, 12,500 machines cluster, job of increasing size

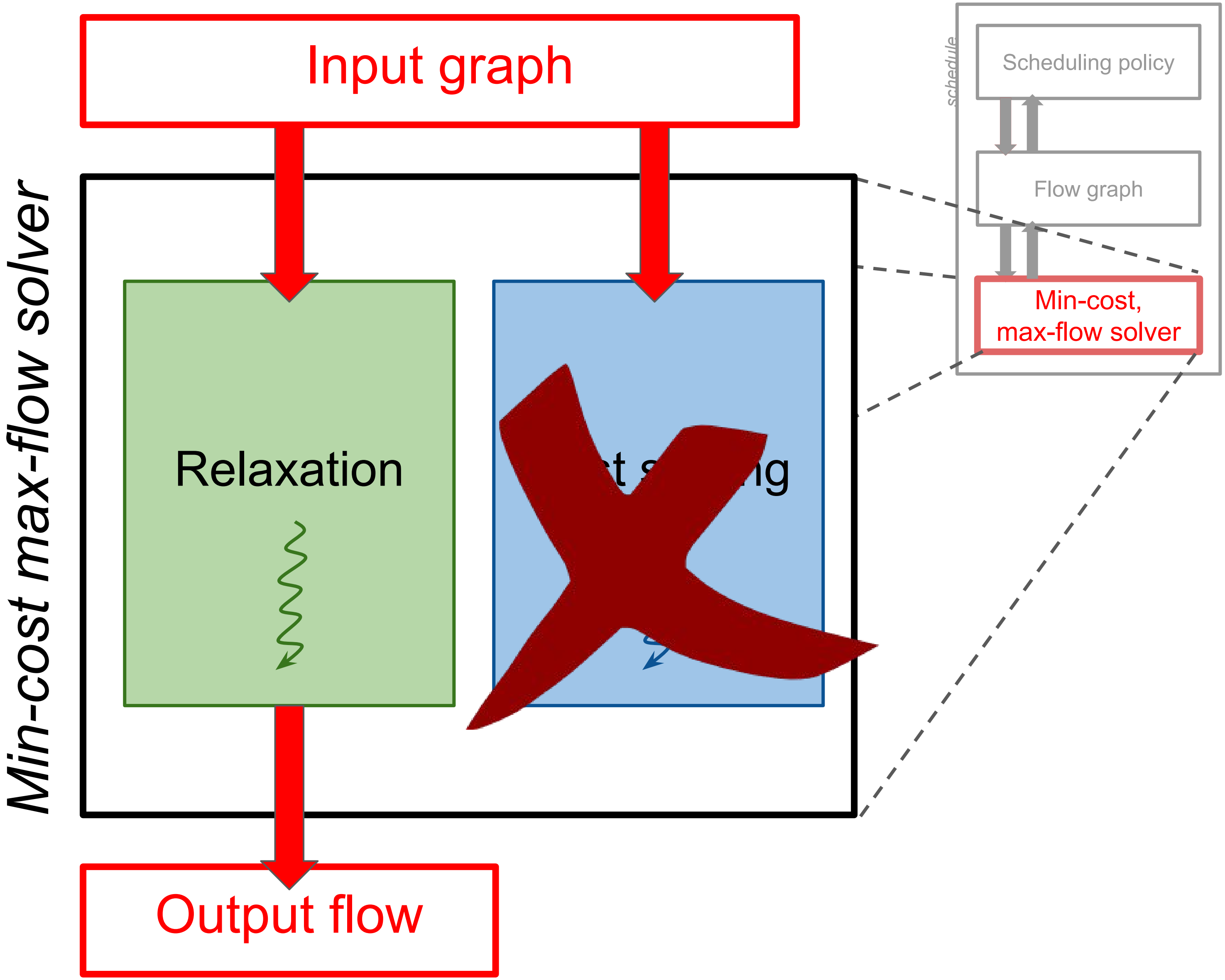


Relaxation's runtime increases with utilization

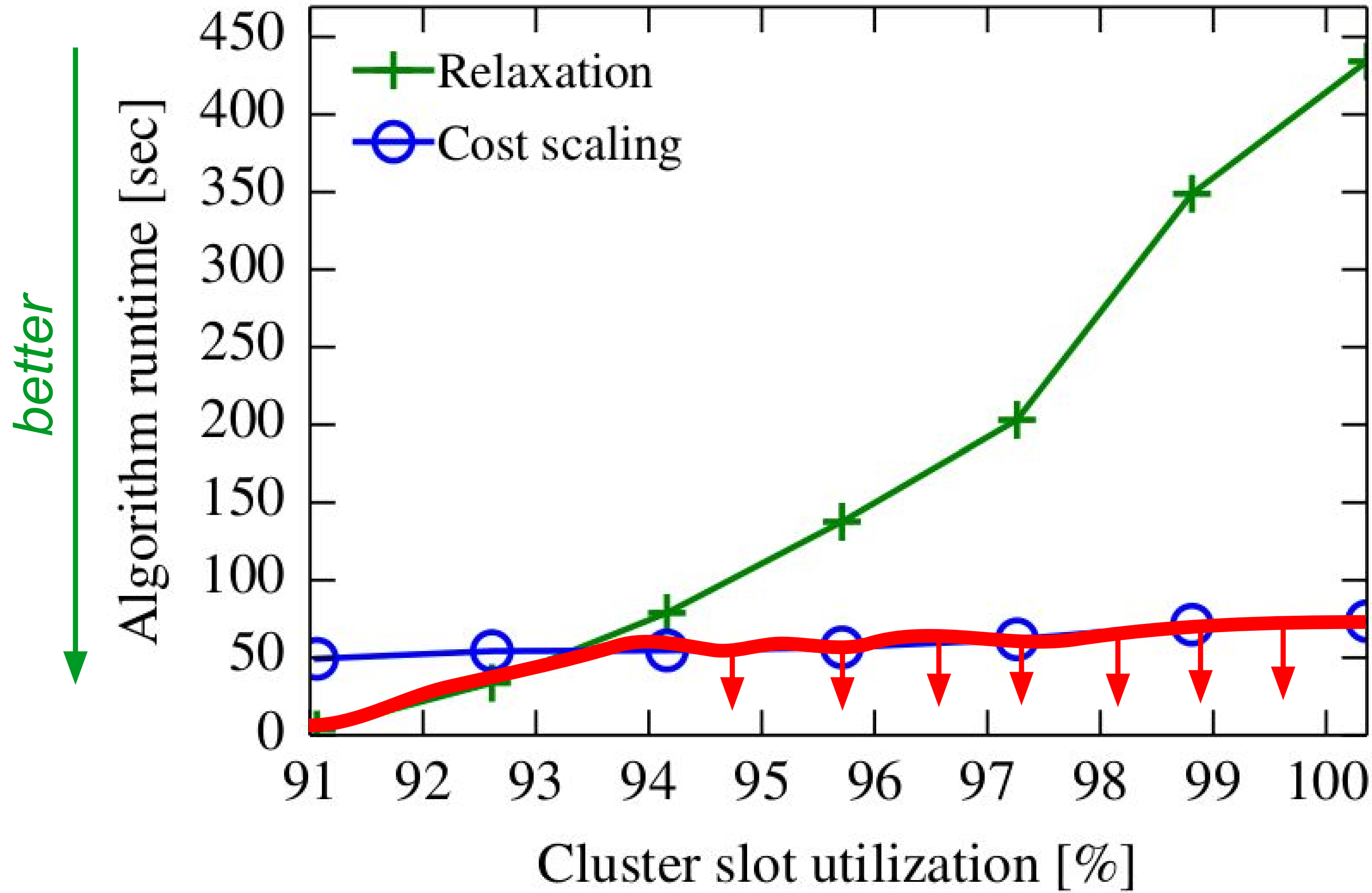
Quincy, 12,500 machines cluster, job of increasing size



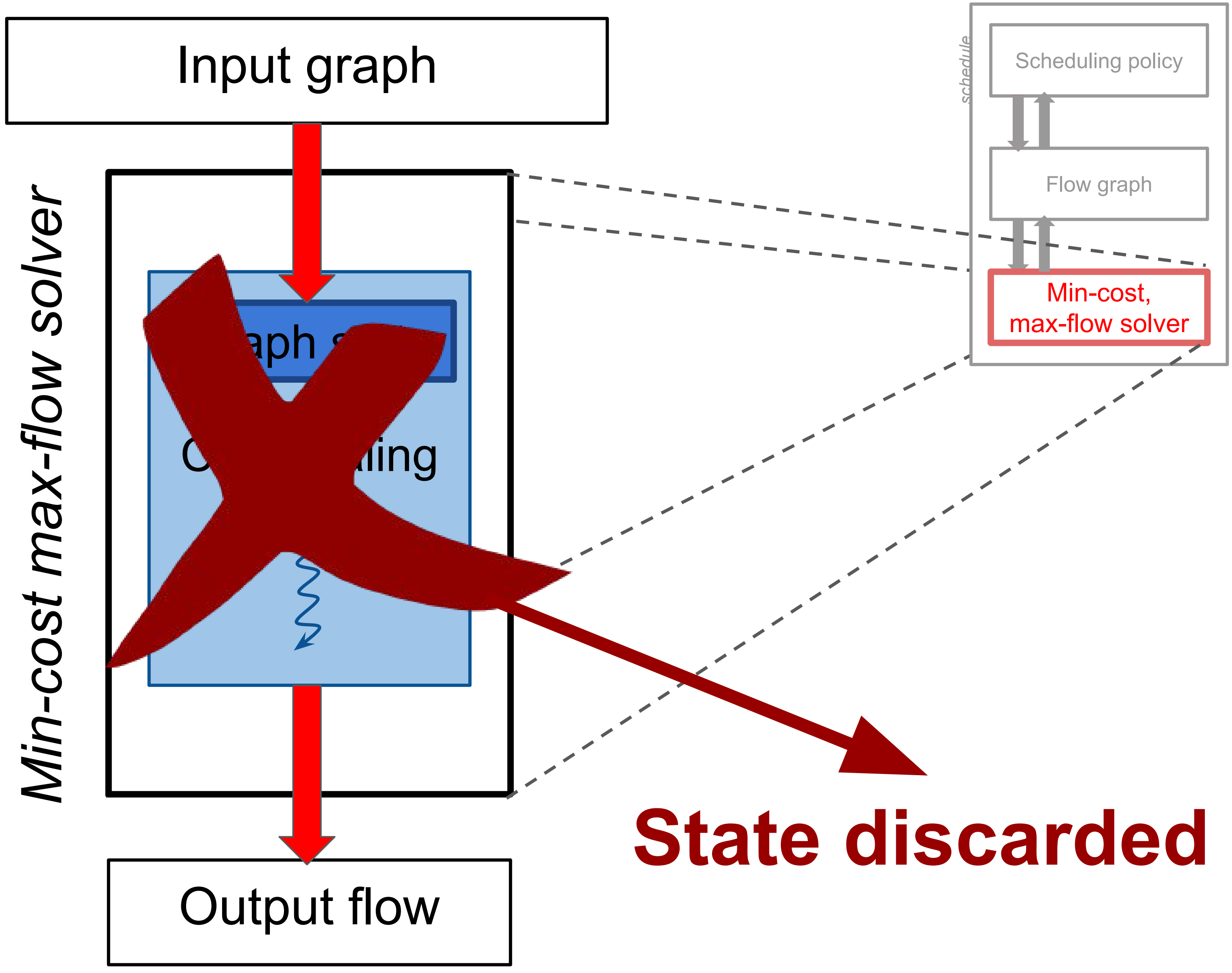
Cost scaling is unaffected by high utilization

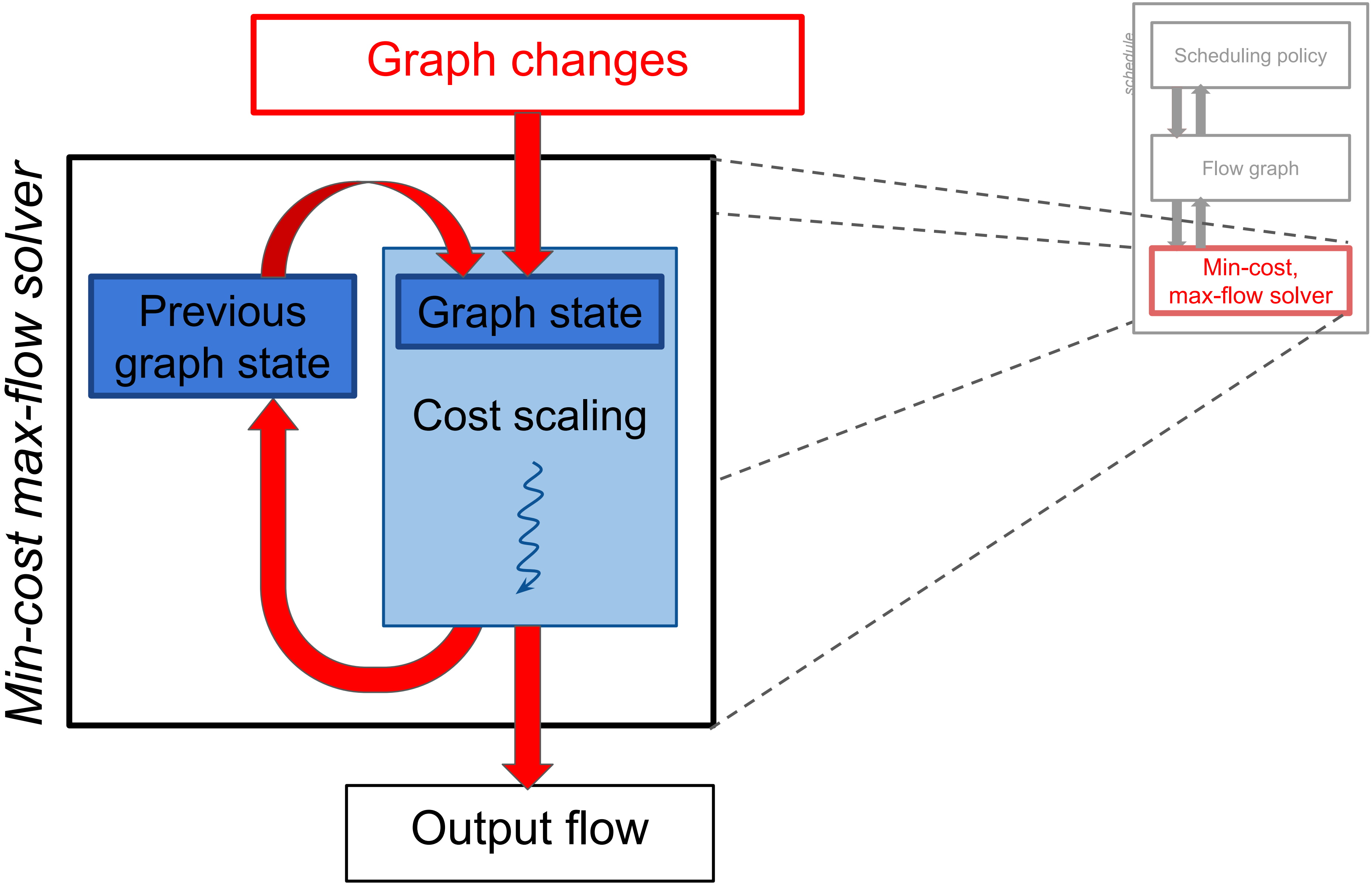

Input graph
Min-cost max-flow solver
Relaxation
Output flow
schedule
Scheduling policy
Flow graph
Min-cost,
max-flow solver

Quincy, 12,500 machines cluster, job of increasing size

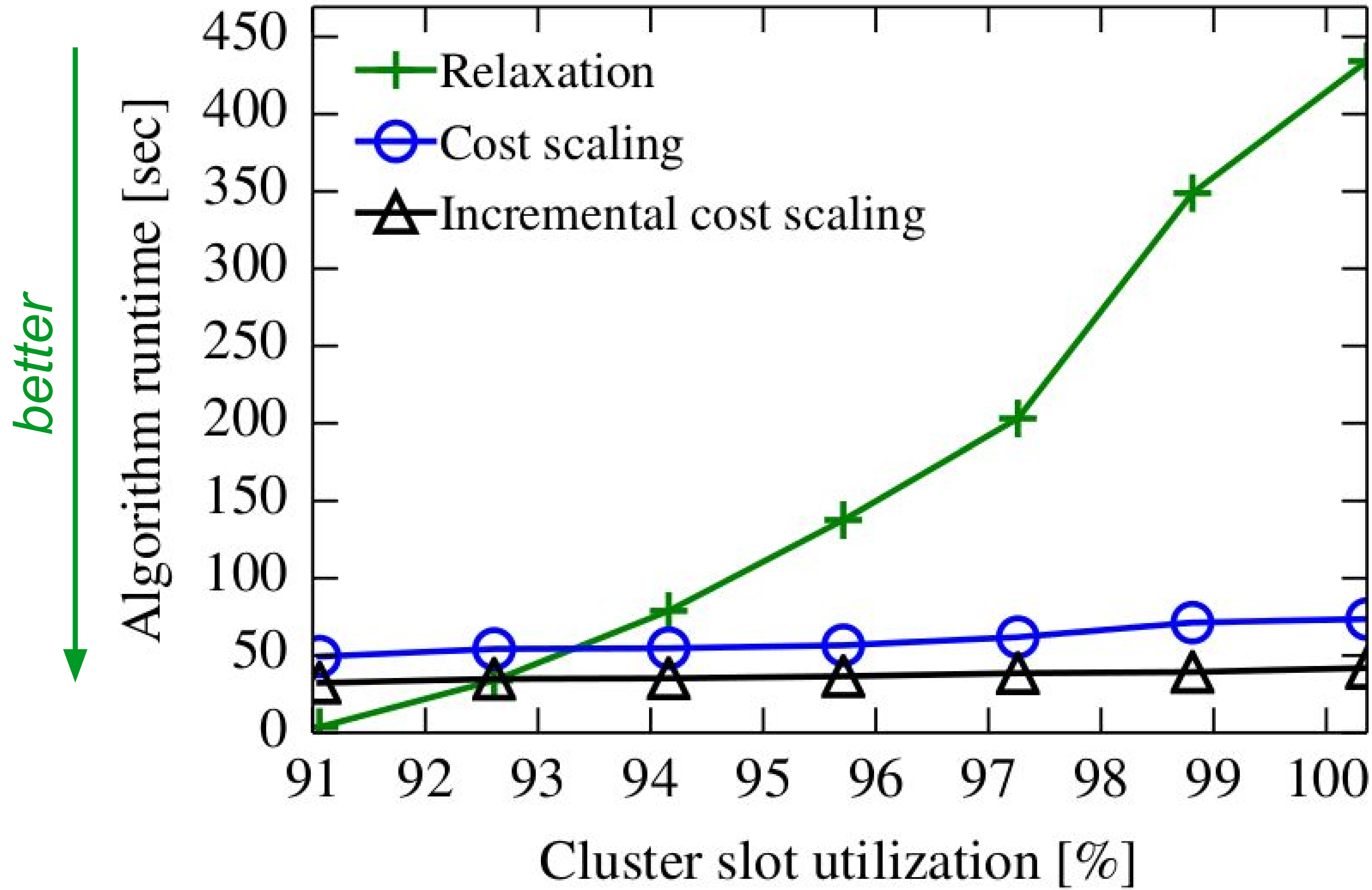


Algorithm runtime is still high at utilization > 94%


Input graph
Min-cost max-flow solver
aph s
C
ling
Output flow
State discarded
schedule
Scheduling policy
Flow graph
Min-cost,
max-flow solver


Graph changes
Min-cost max-flow solver
Previous graph state
Graph state
Cost scaling
Output flow
Scheduling policy
Flow graph
Min-cost, max-flow solver
schedule

Quincy, 12,500 machines cluster, job of increasing size



Incremental cost scaling is ~2x faster

Evaluation

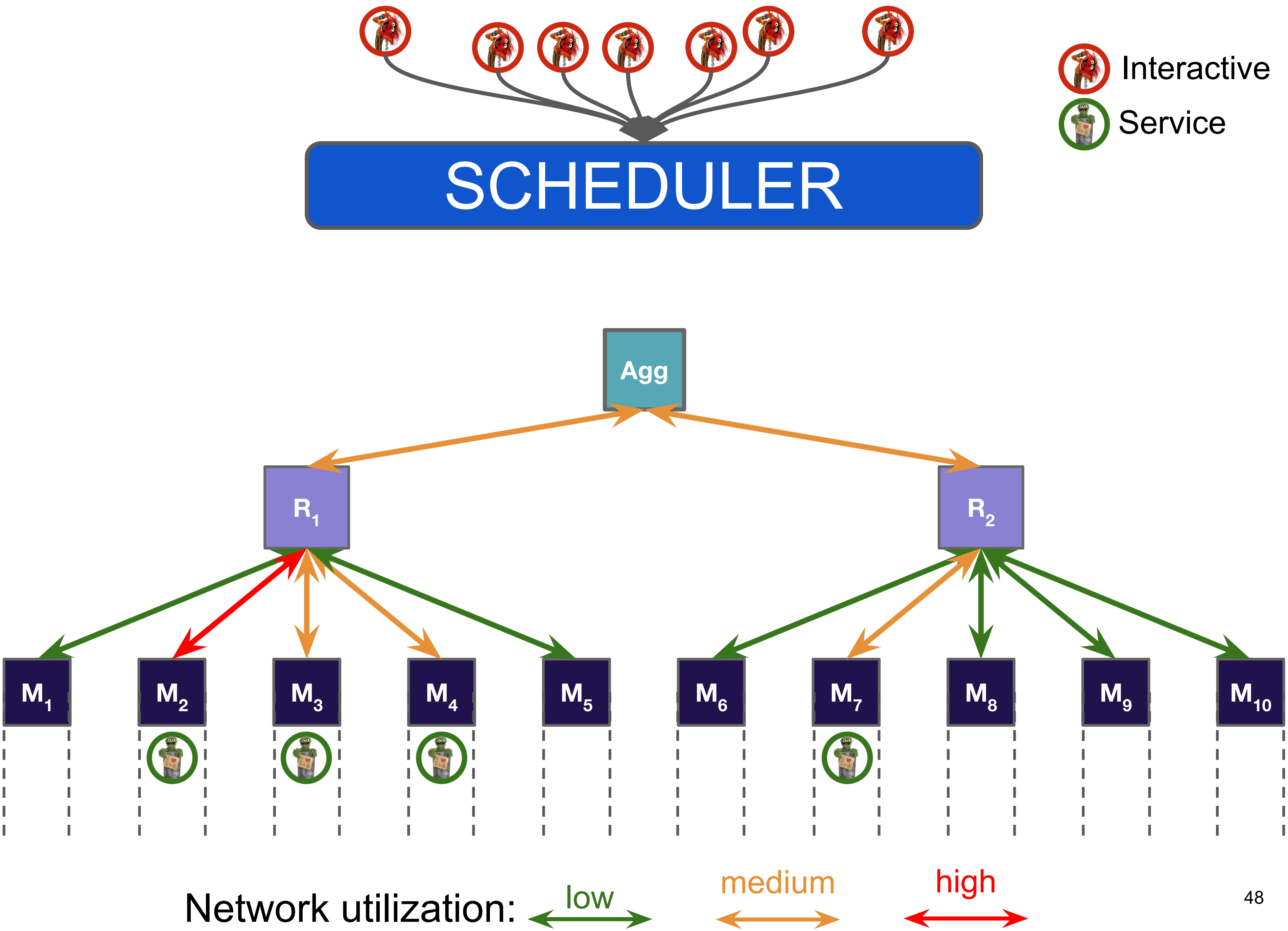
Does Firmament choose good placements with low latency?

Note: many additional experiments in the paper.

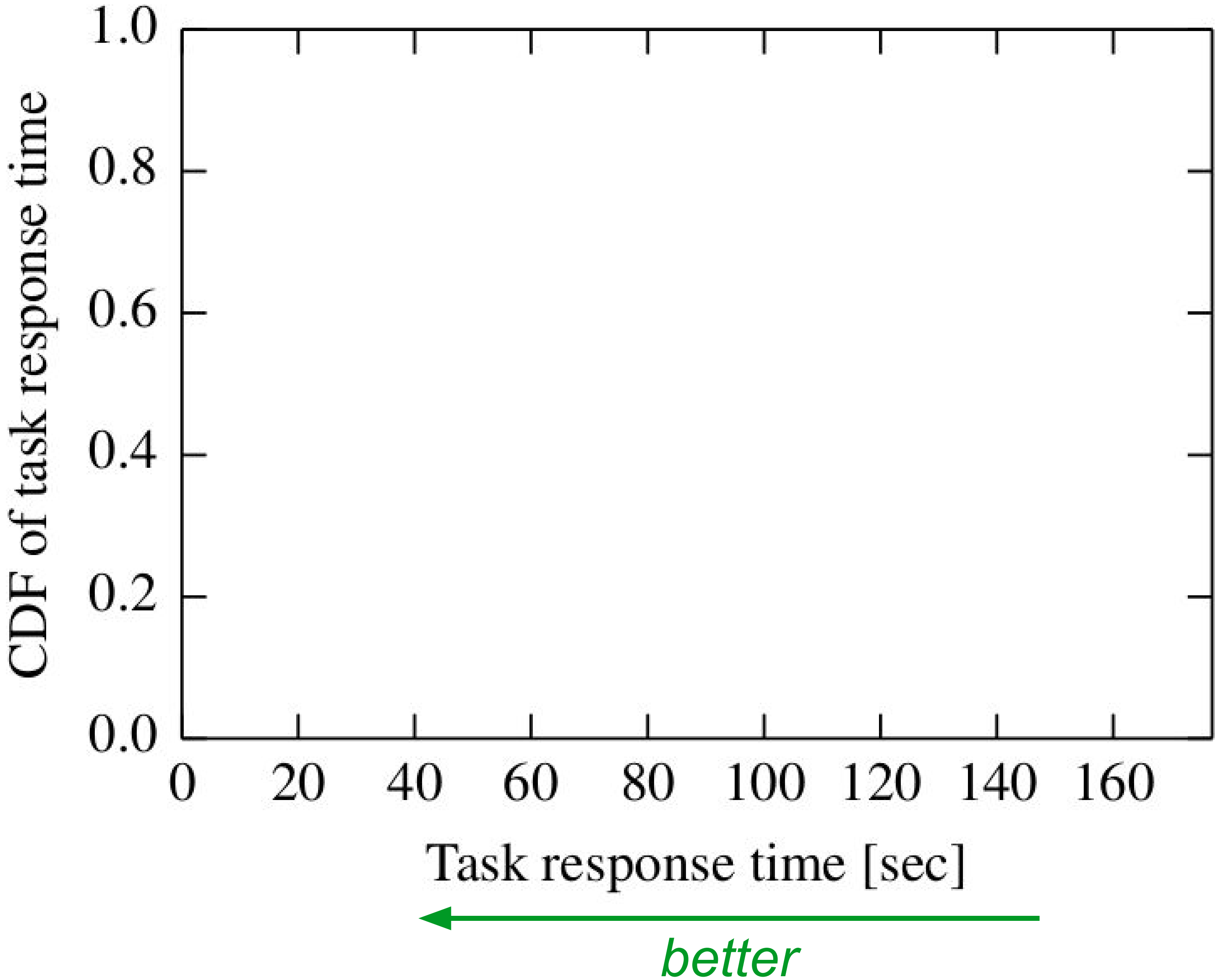
How do Firmament's placements compare to other schedulers?

Experimental setup:

- Homogeneous 40-machine cluster, 10G network
- Mixed batch/service/interactive workload

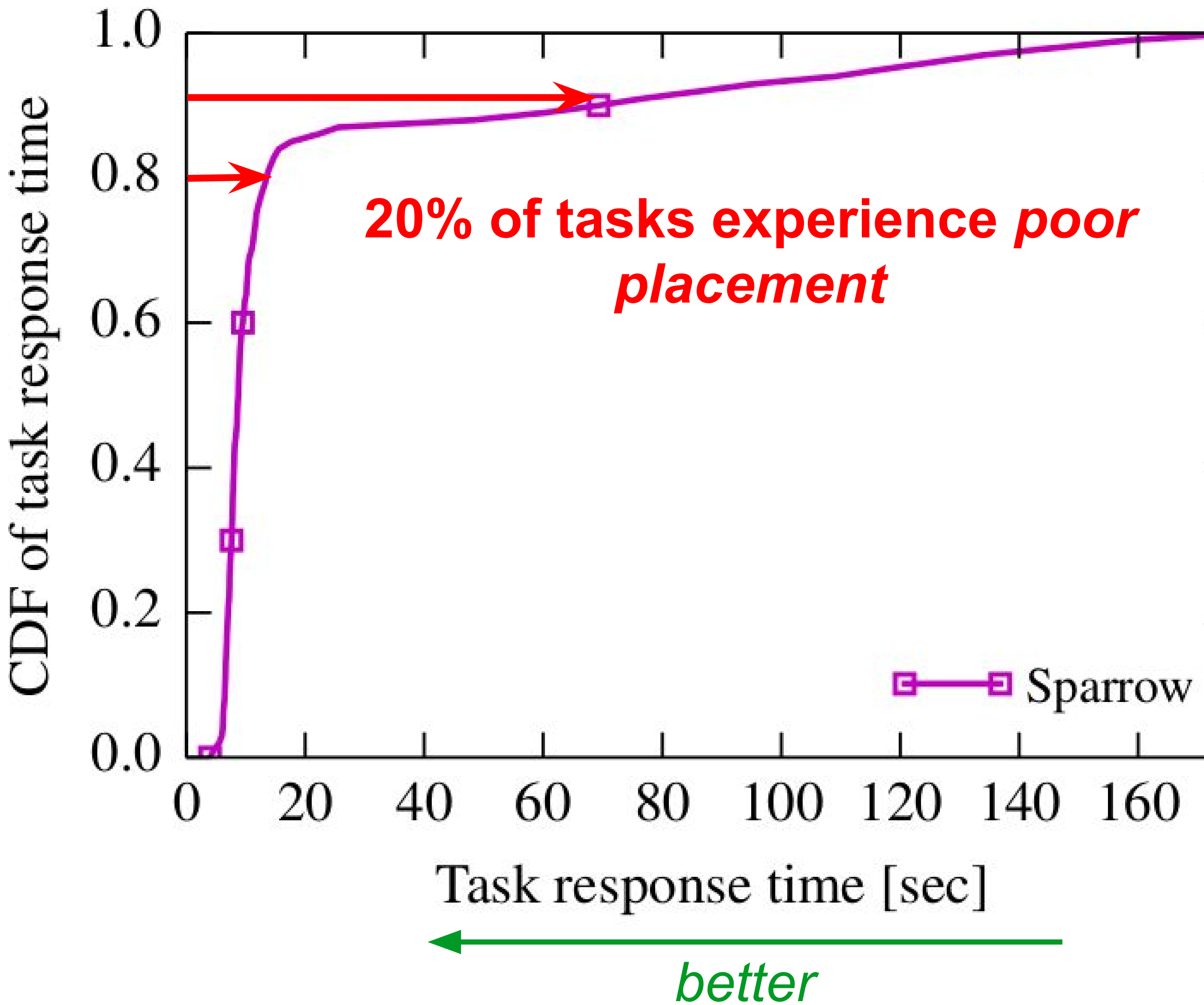

Interactive
Service
SCHEDULER
Agg
R1
R2
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
Network utilization:
low
medium
high

Firmament chooses good placements



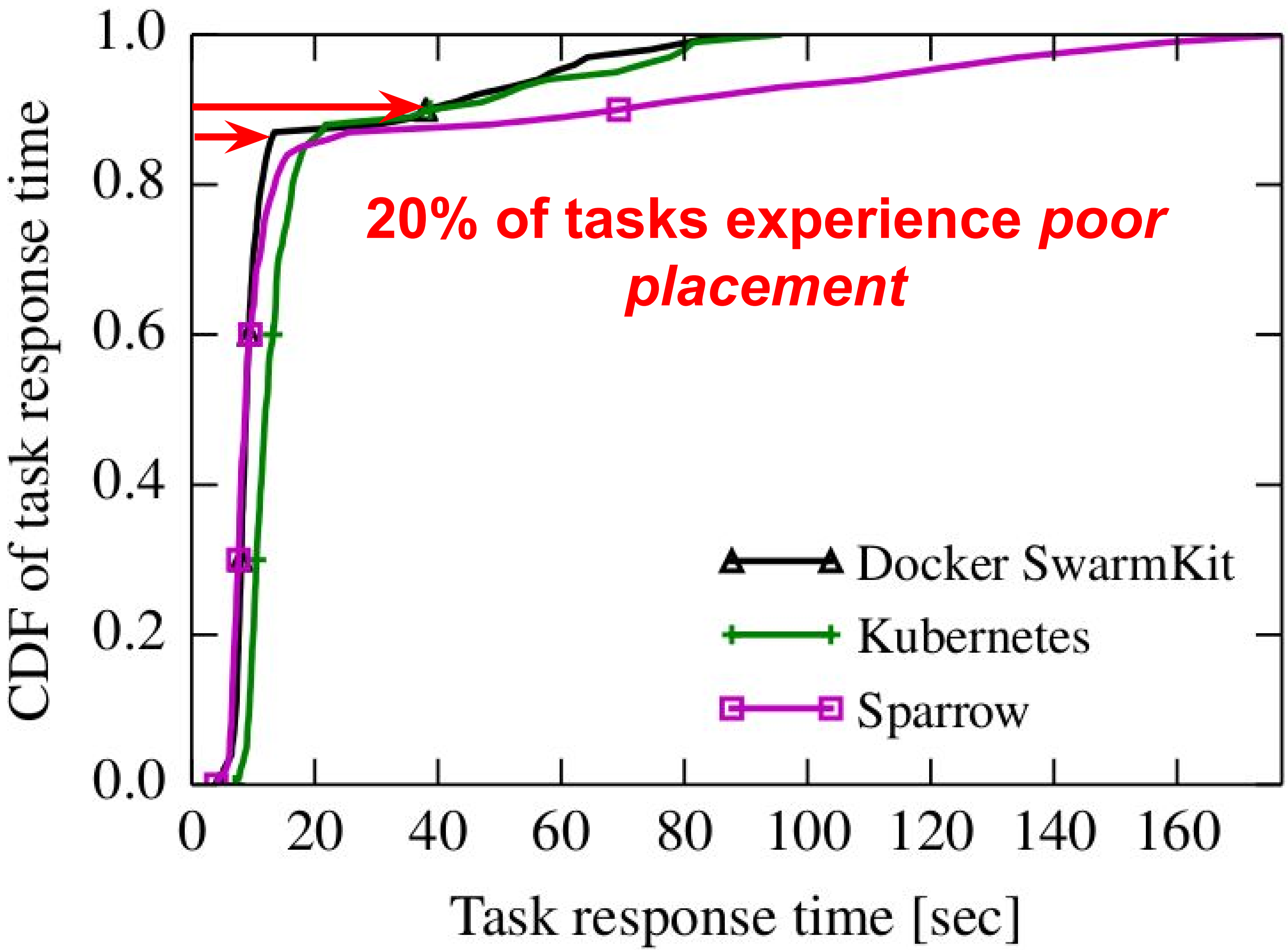
5 seconds task response time on idle cluster

Firmament chooses good placements



Sparrow is unaware of resource utilization

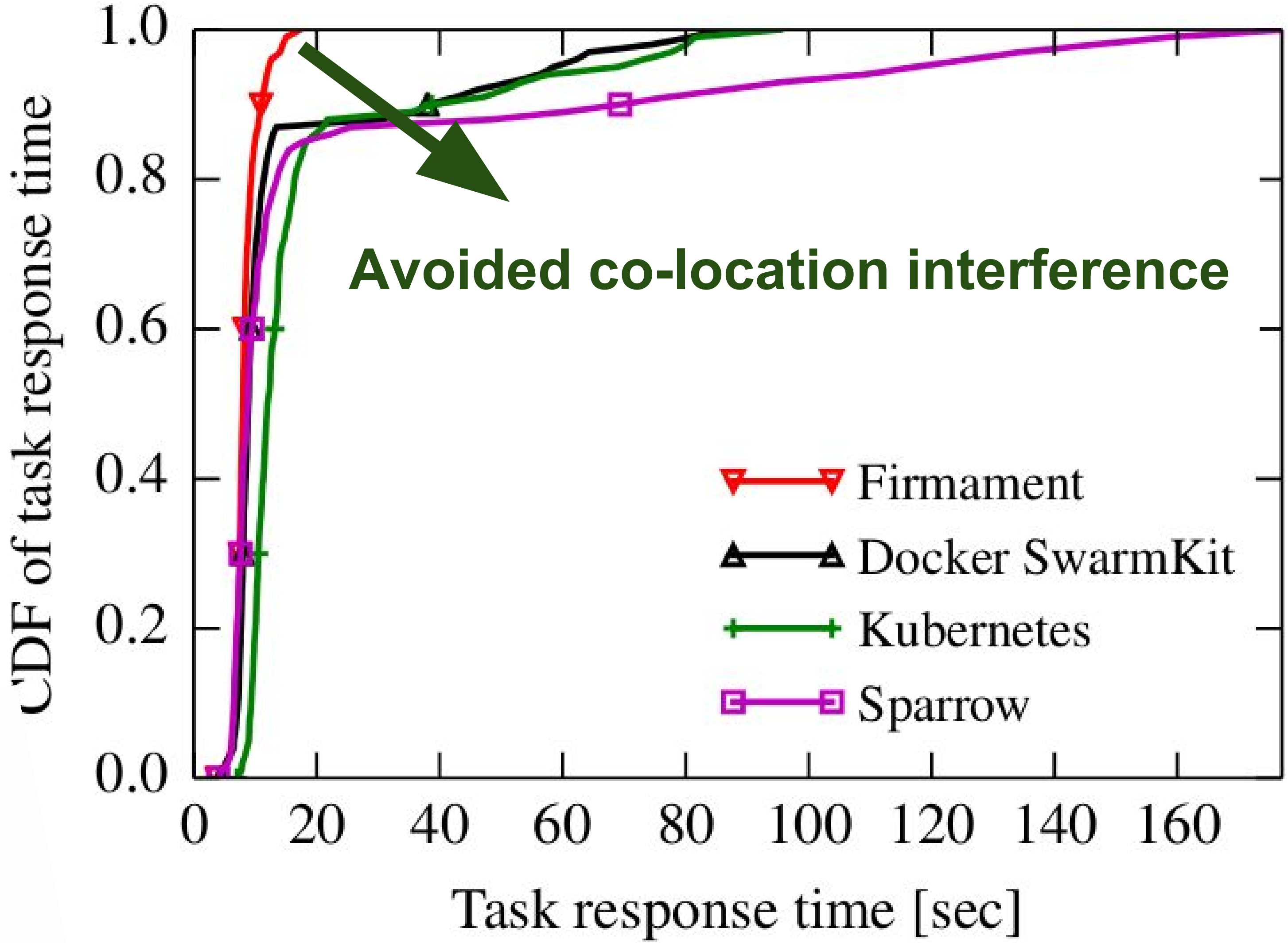
Firmament chooses good placements



better

Centralized Kubernetes and Docker still suffer

Firmament chooses good placements



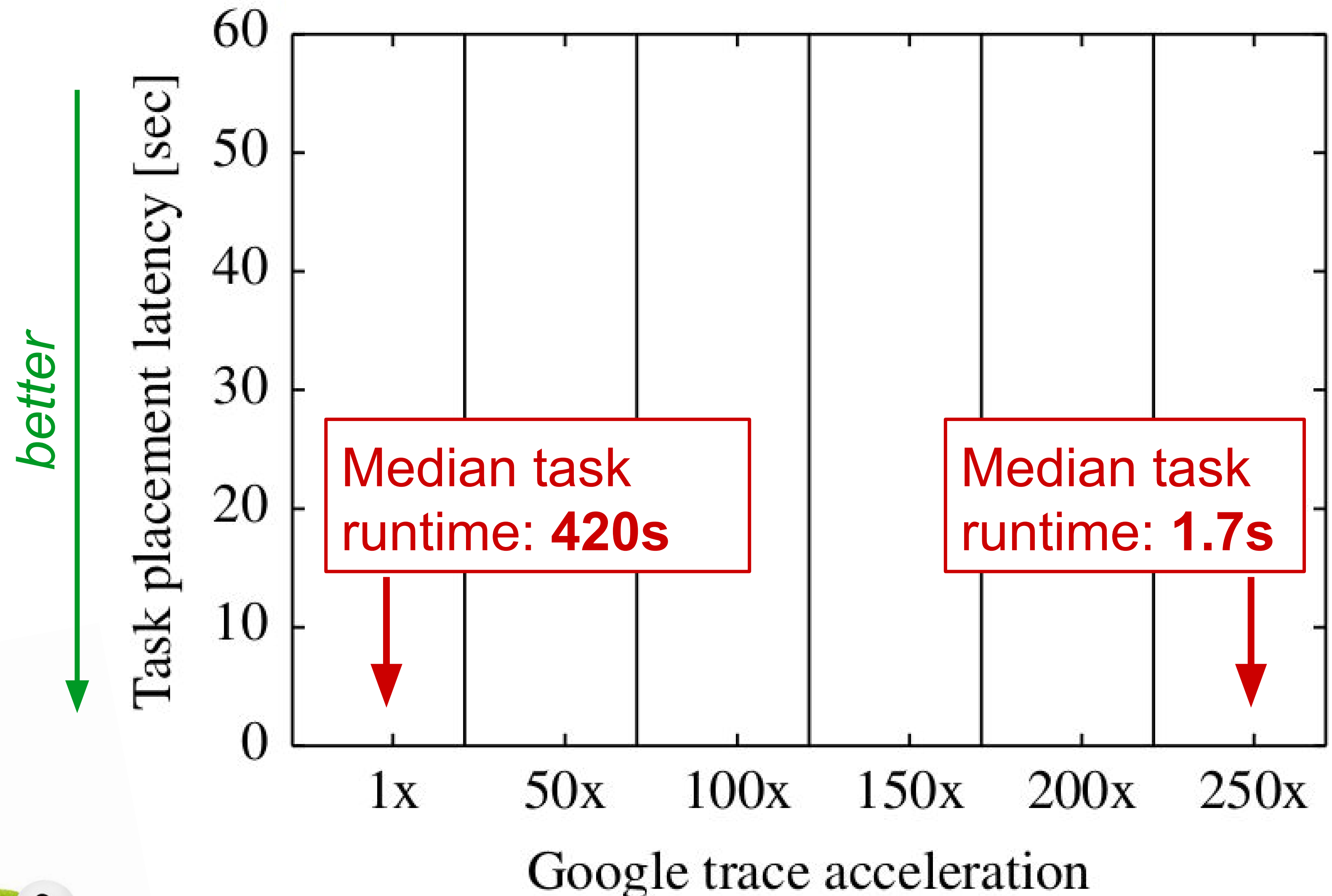
Firmament outperforms centralized and distributed schedulers

How well does Firmament handle challenging workloads?

Experimental setup:

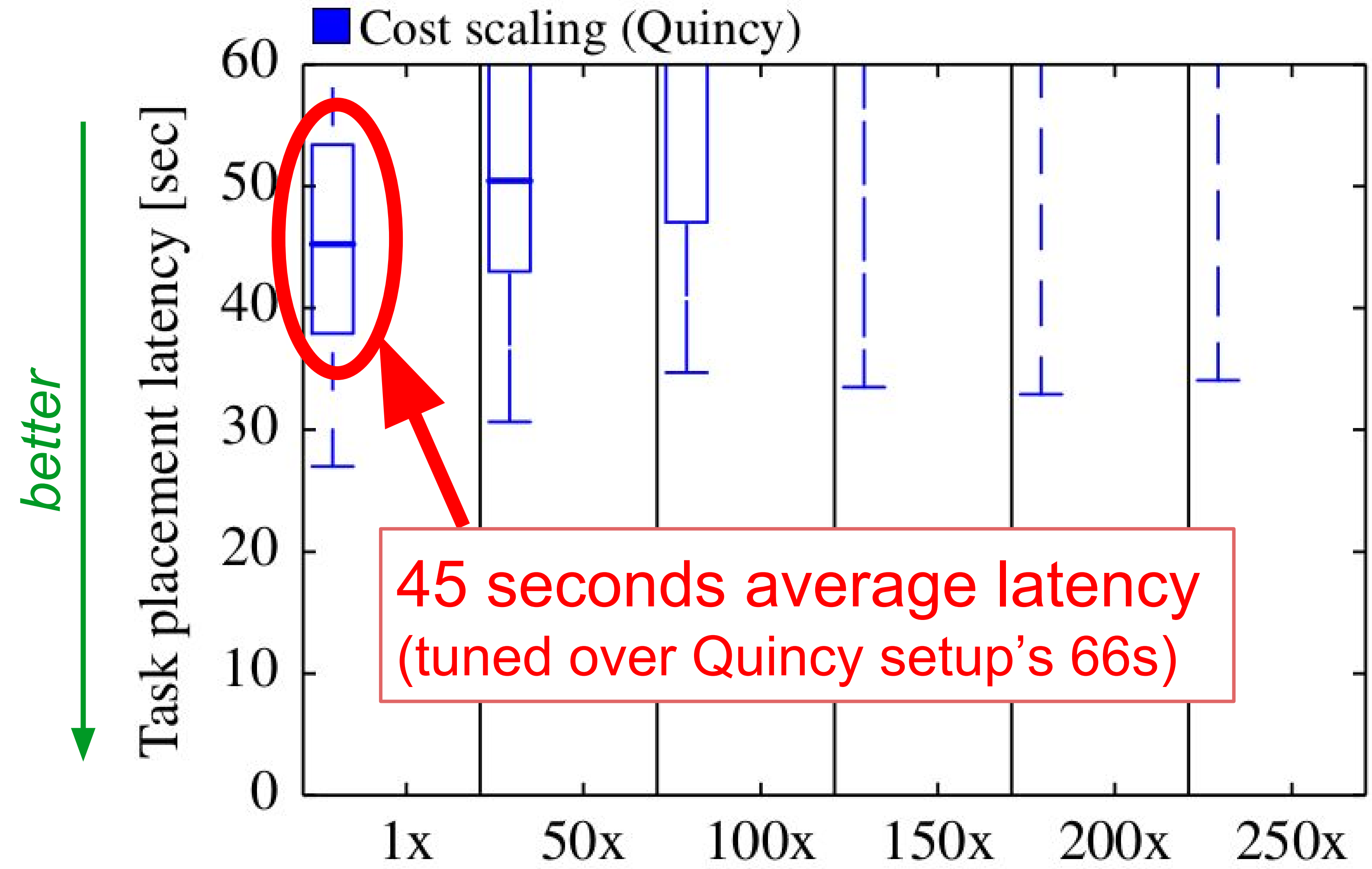
- Simulated 12,500 machine Google cluster
- Used the centralized Quincy scheduling policy
- Utilization varies between 75% and 95%

Firmament handles challenging workloads at low latency



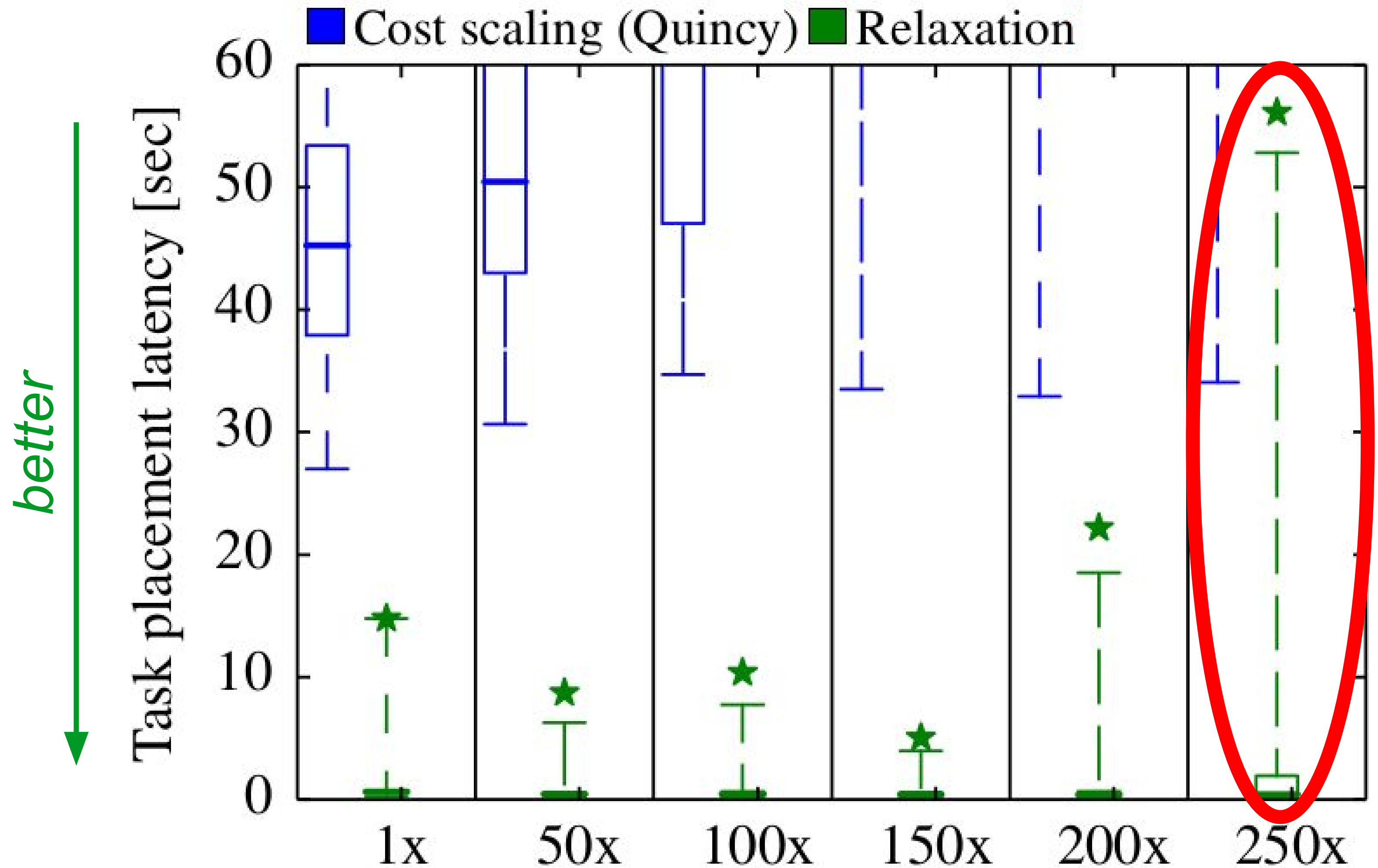
Simulate interactive workloads by scaling down task runtimes

Firmament handles challenging workloads at low latency



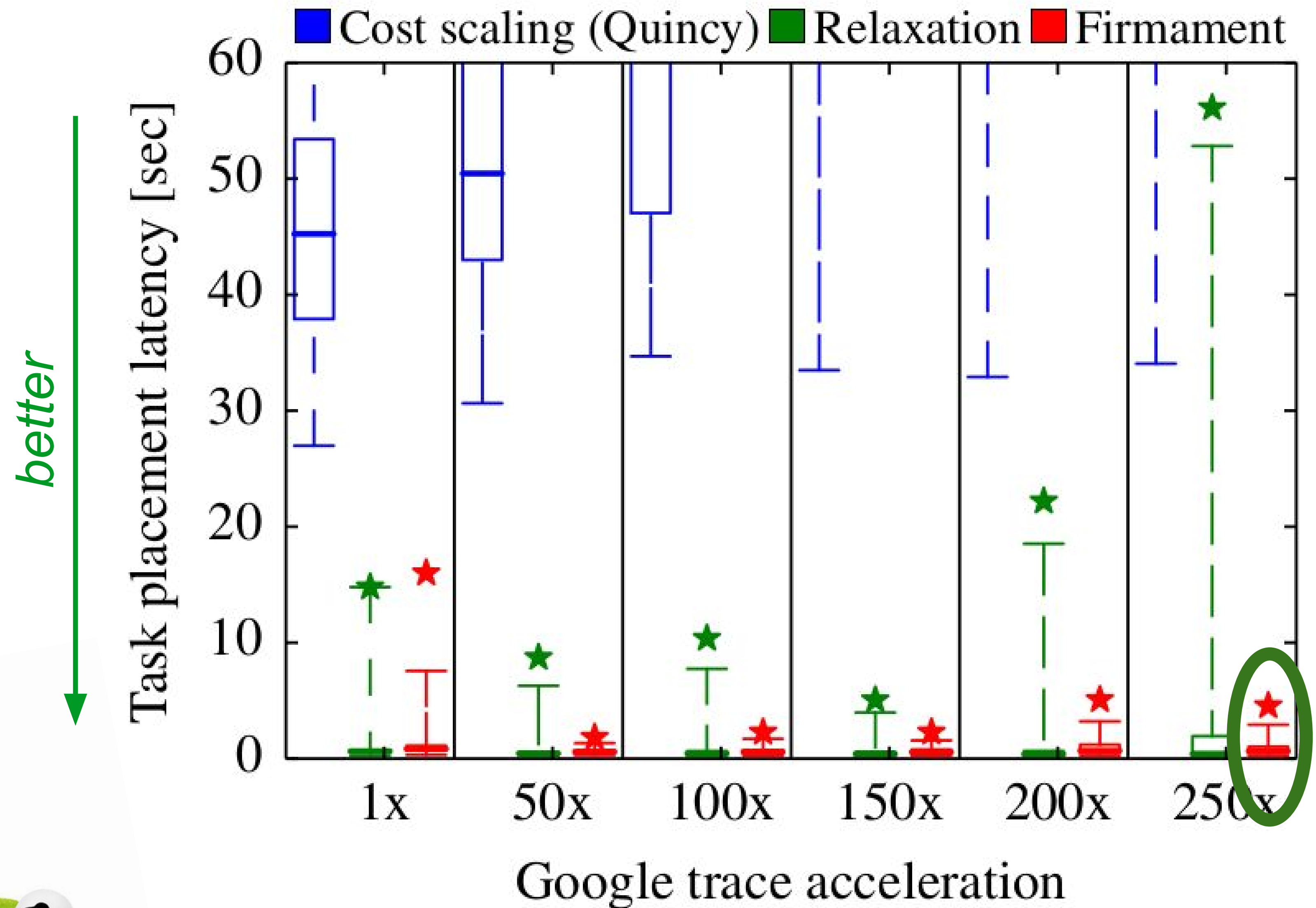
Average latency is too high even without many short tasks

Firmament handles challenging workloads at low latency



Latency with a 250x acceleration:
75th percentile: 2 sec
maximum: 57 sec

Firmament handles challenging workloads at low latency



Firmament's common-case latency is sub-second even at 250x acceleration

Conclusions

- **Low task scheduling latency**
  - Uses best algorithm at all times
  - Incrementally recomputes solution
- **Good task placement**
  - Same optimal placements as Quincy
  - Customizable scheduling policies

Open-source and available at:

firmament.io