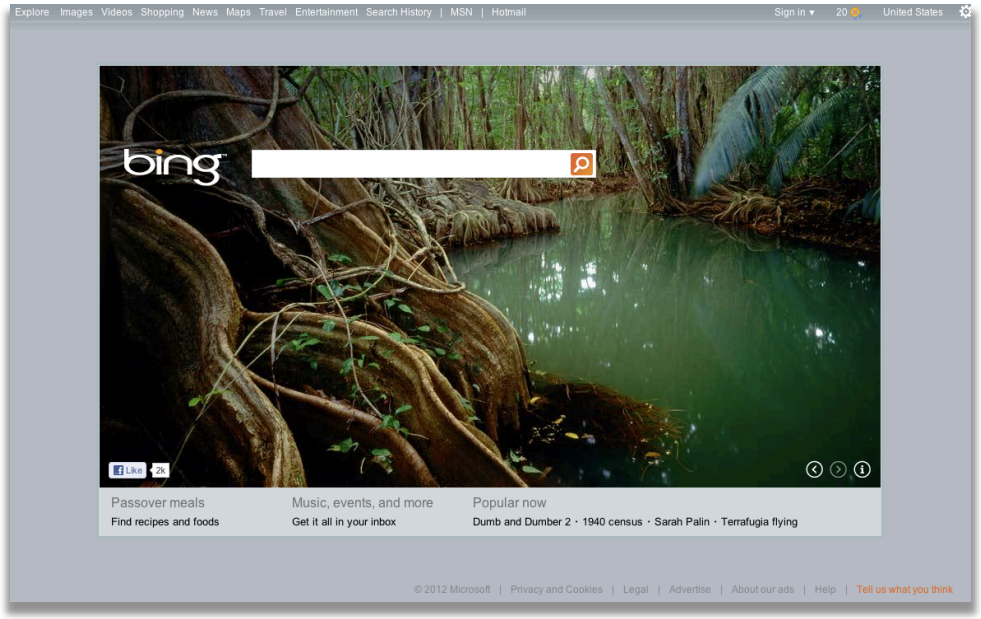
Jockey

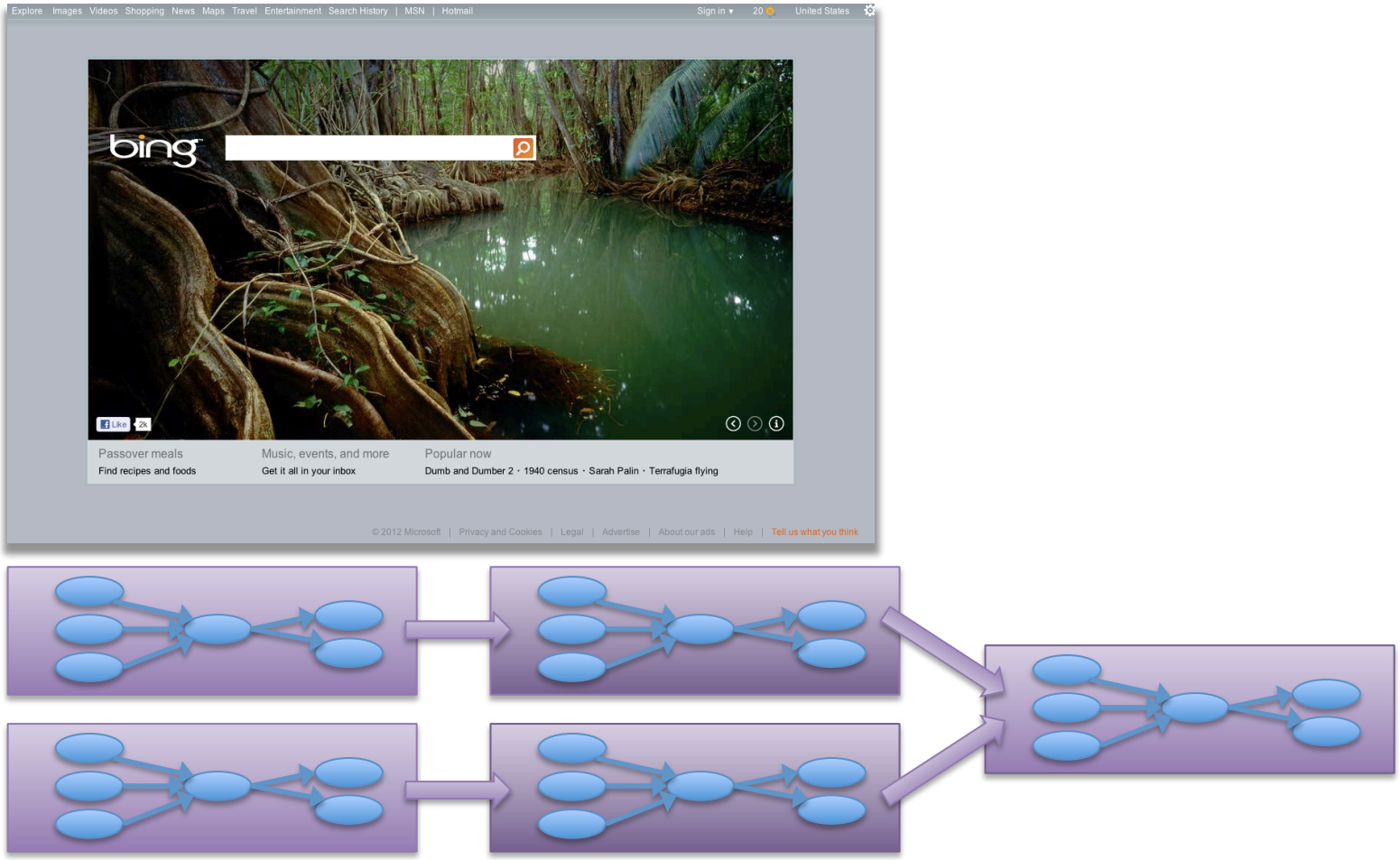
Guaranteed Job Latency in Data Parallel Clusters

Andrew Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca

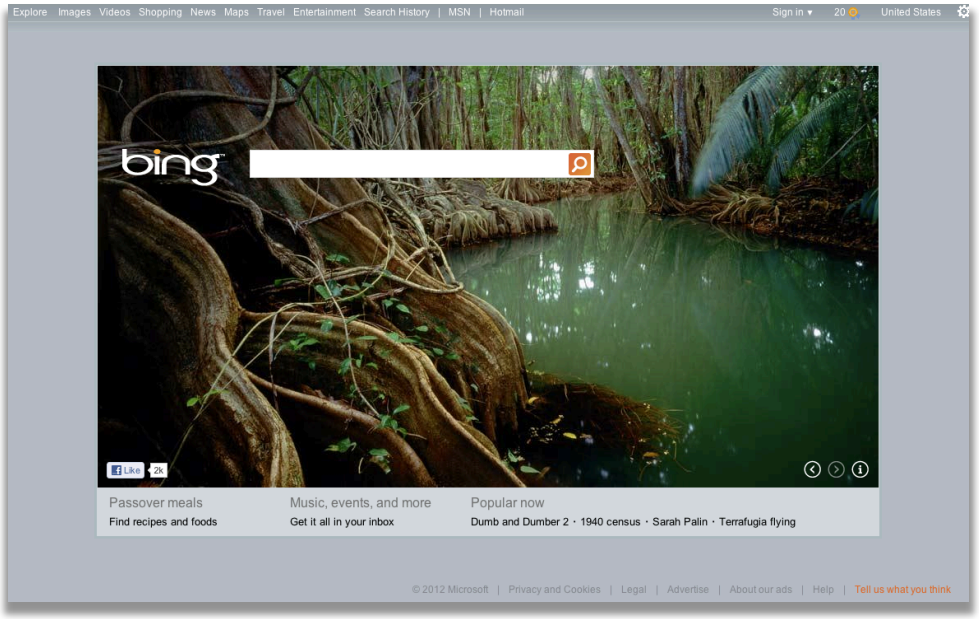
Microsoft® Research

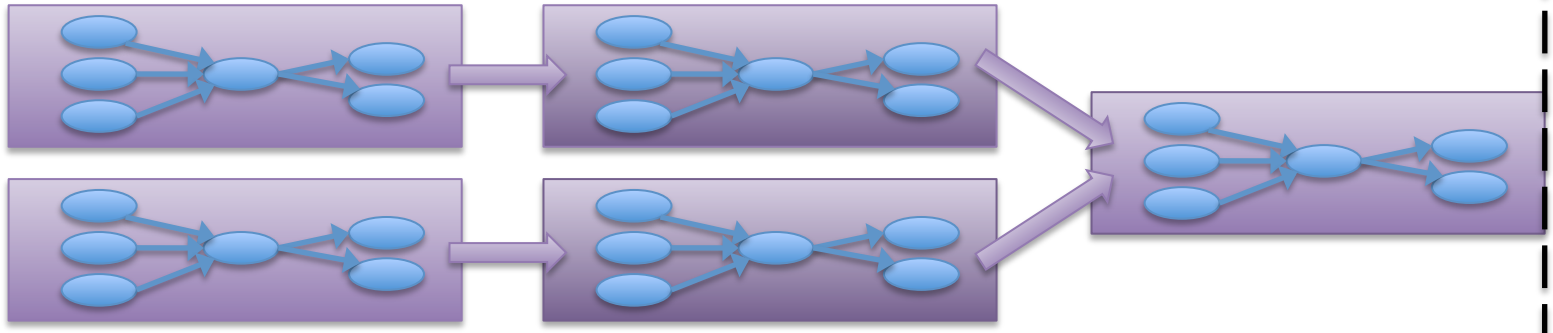


DATA PARALLEL CLUSTERS



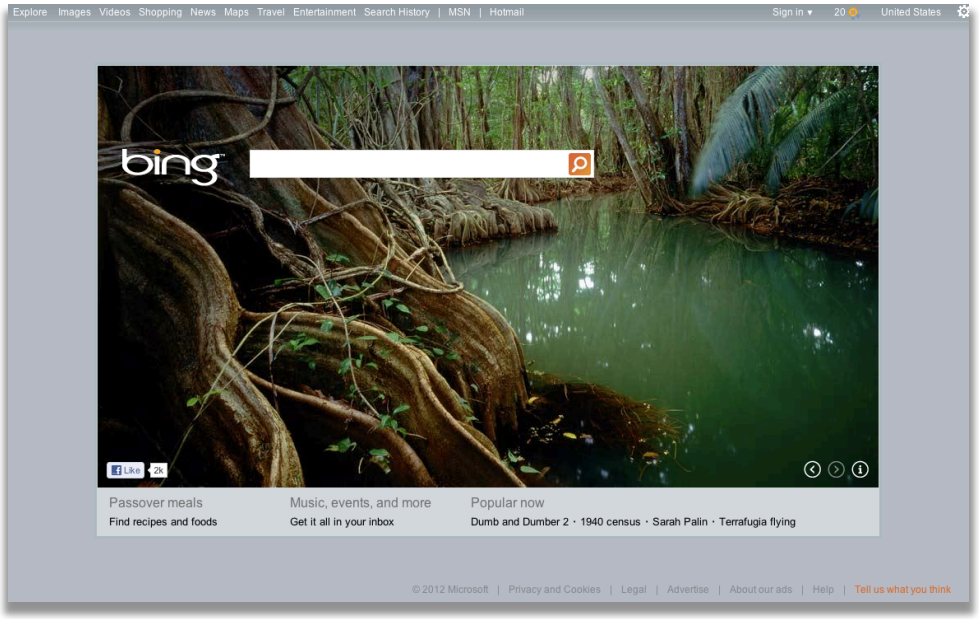
DATA PARALLEL CLUSTERS

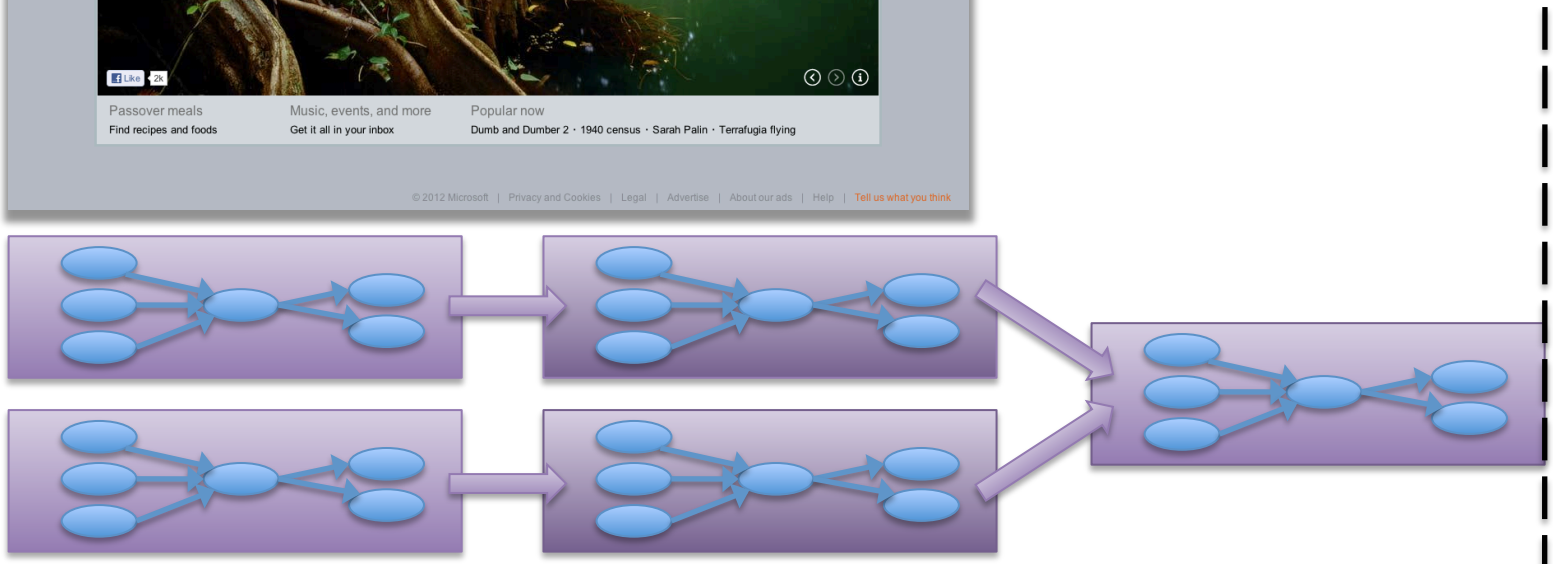




Predictability

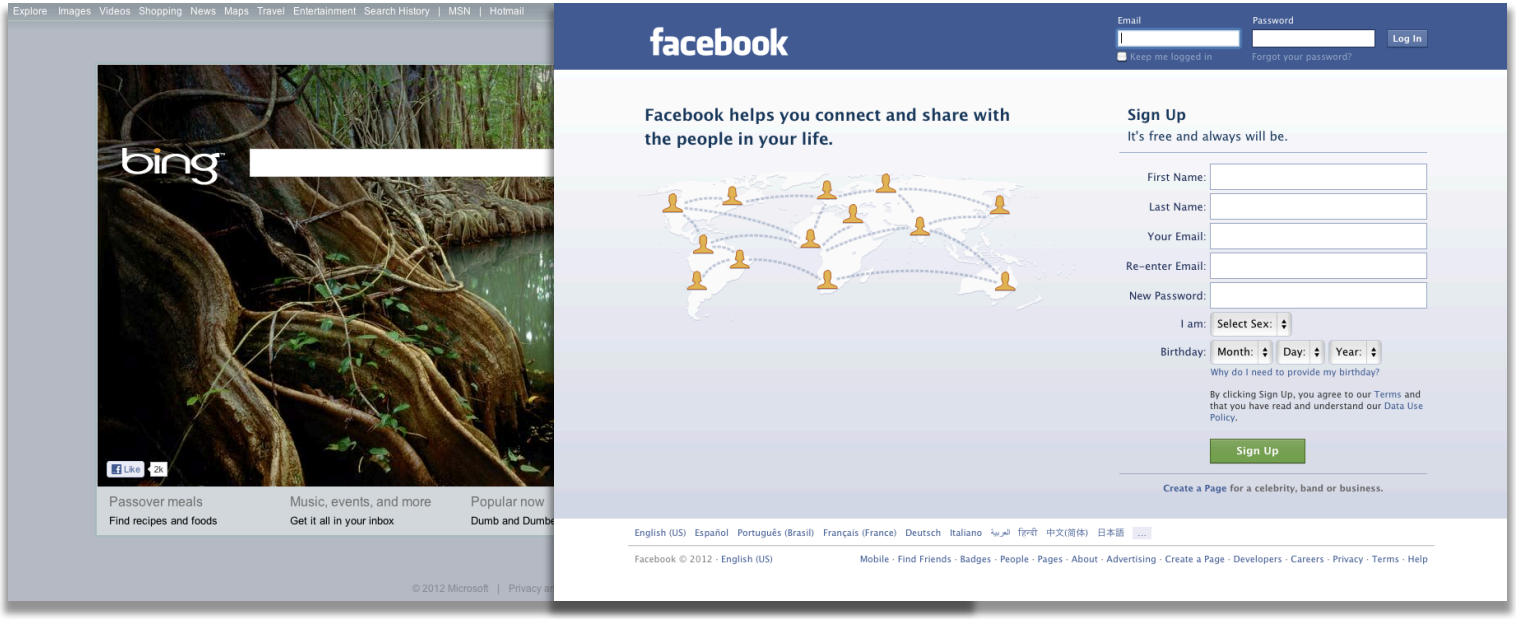
DATA PARALLEL CLUSTERS

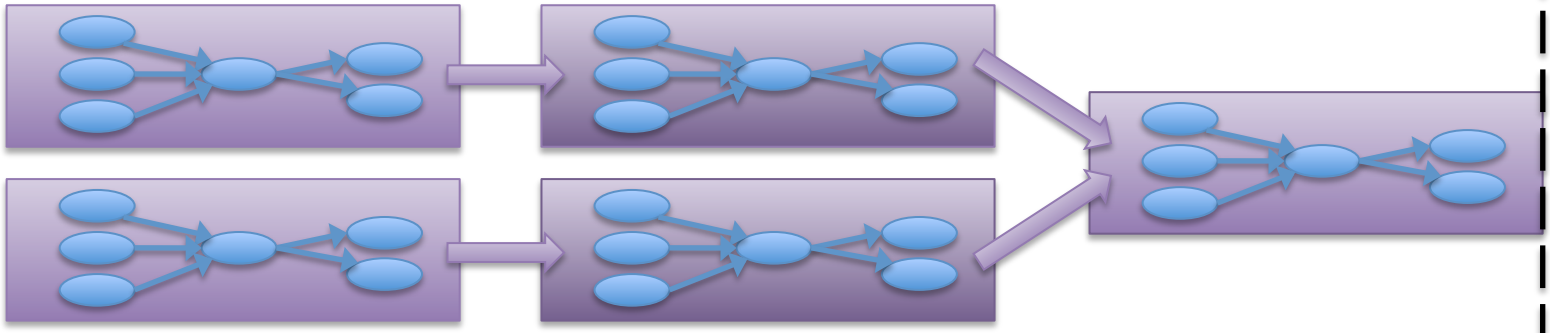




Deadline

DATA PARALLEL CLUSTERS

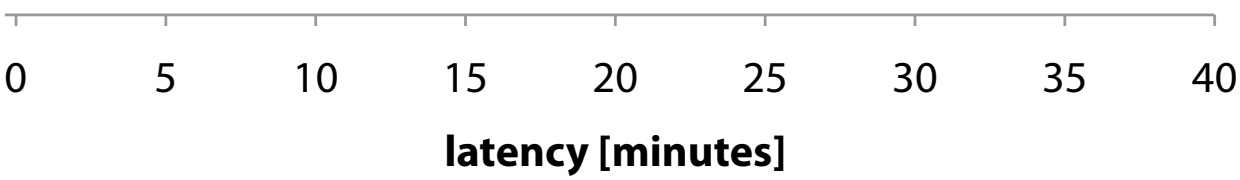




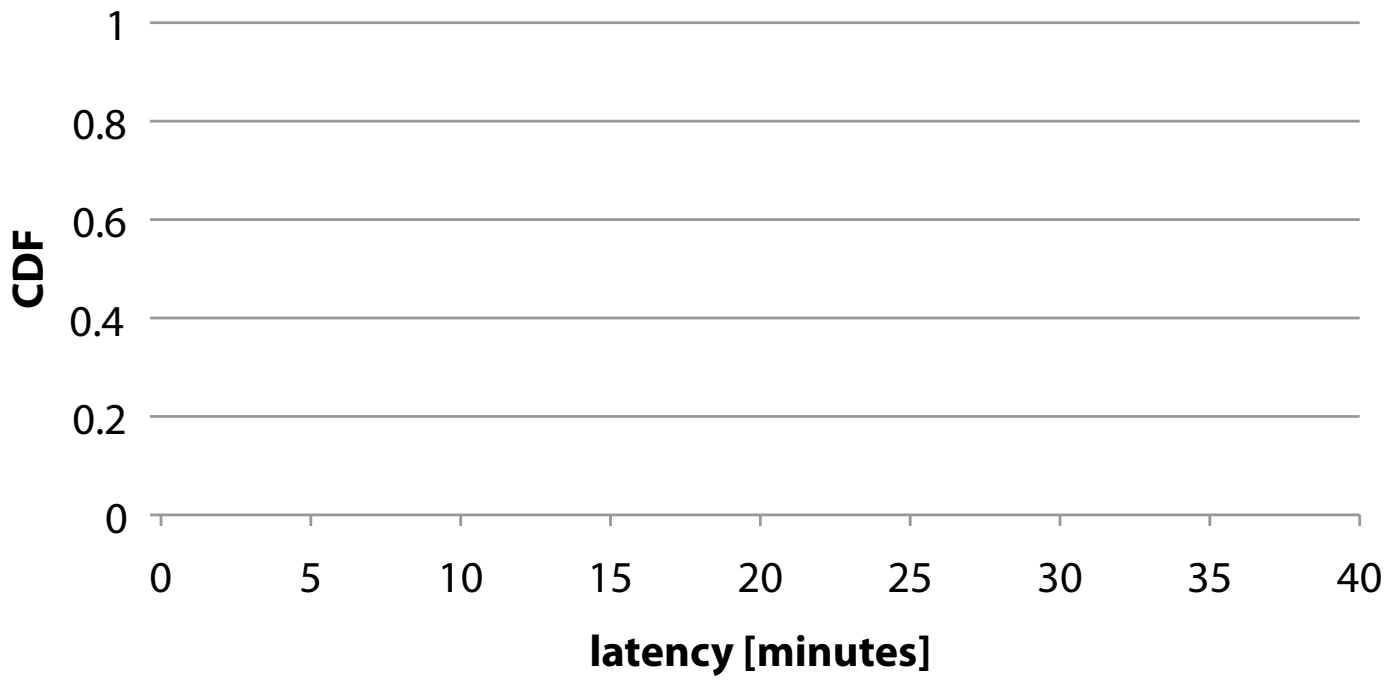
Deadline

DATA PARALLEL CLUSTERS

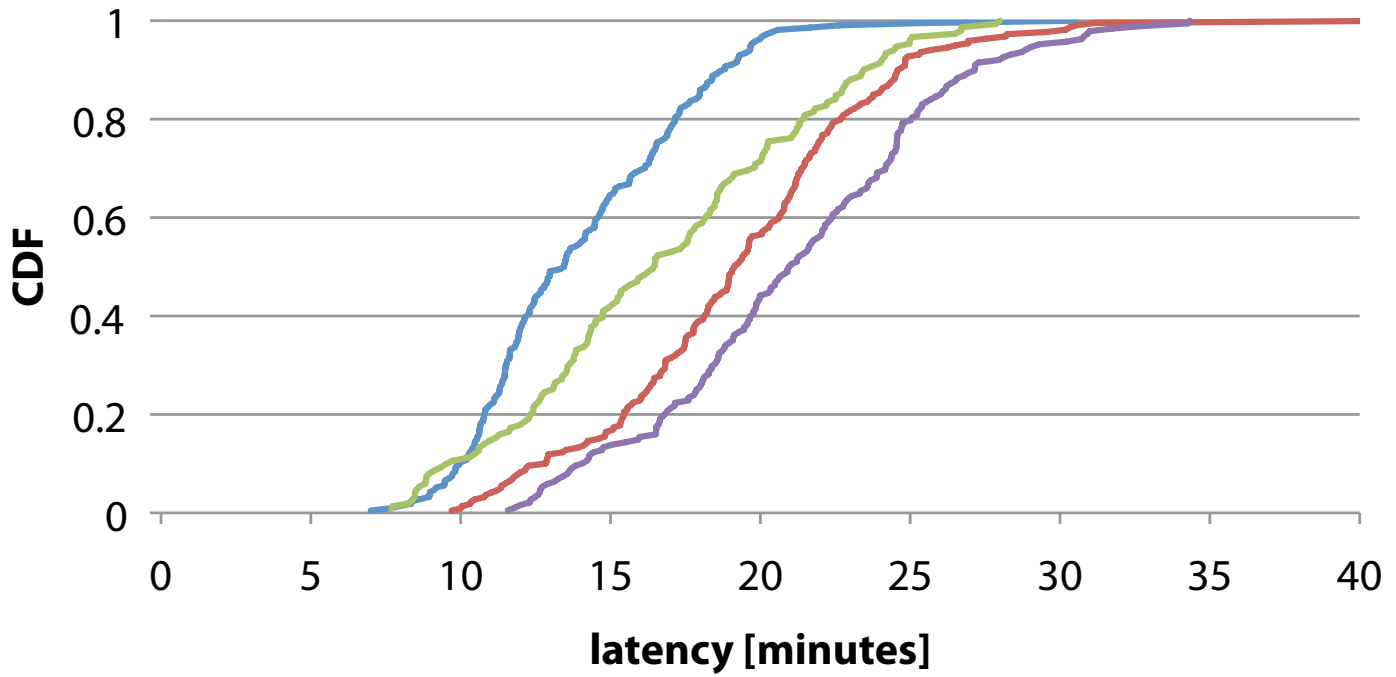
VARIABLE LATENCY



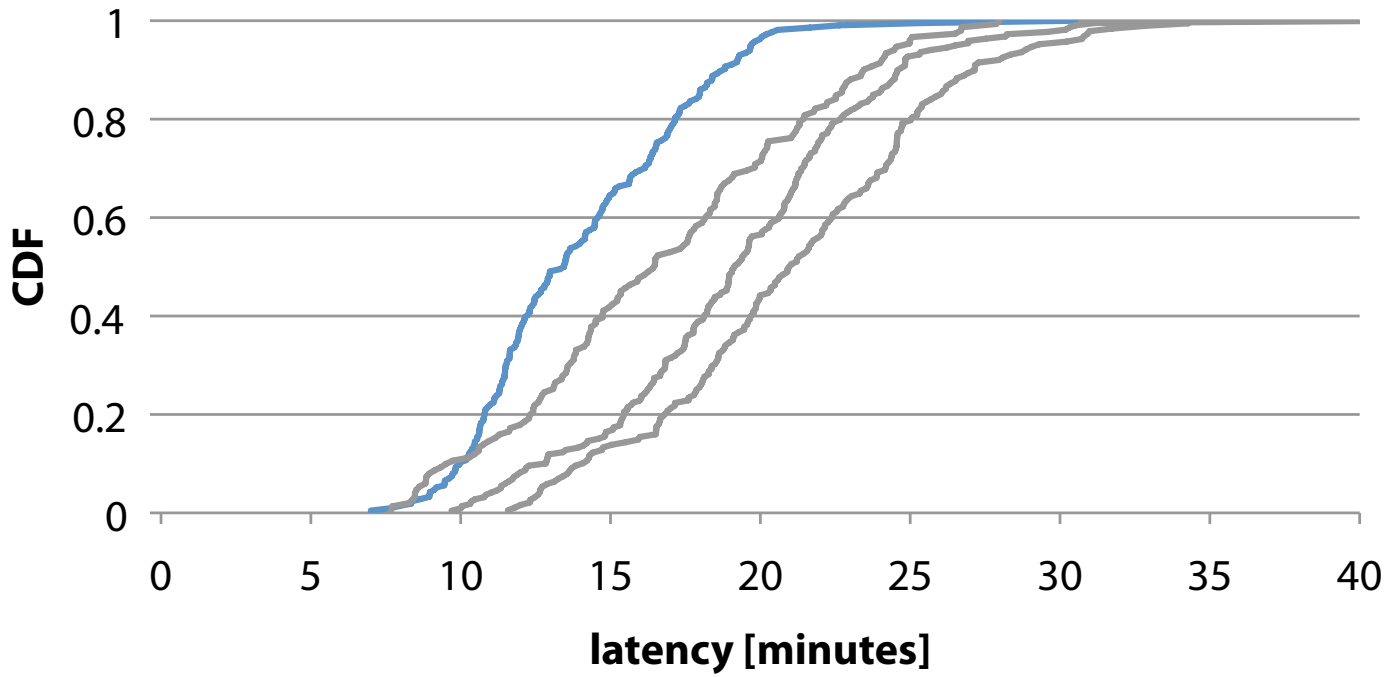
VARIABLE LATENCY



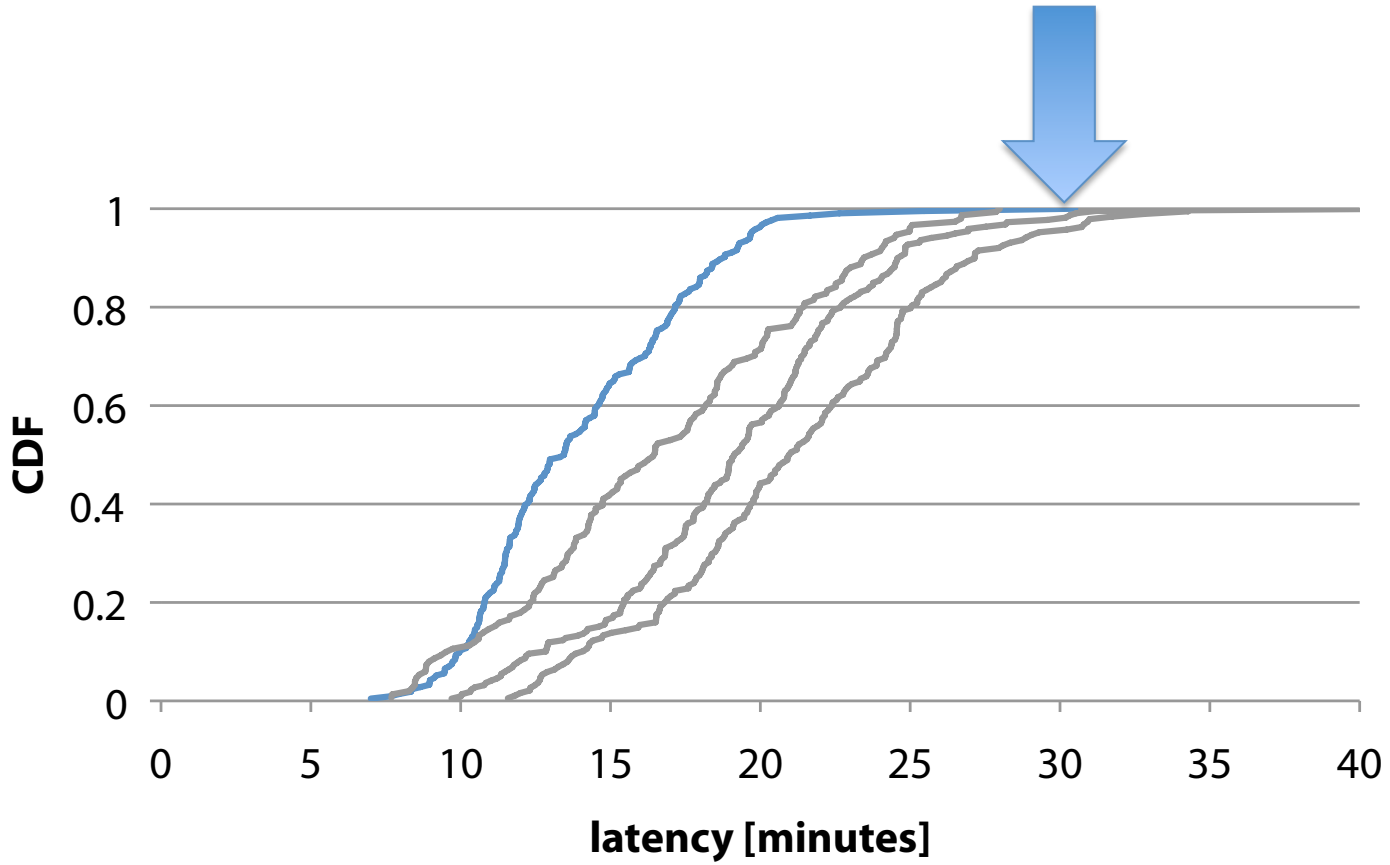
VARIABLE LATENCY



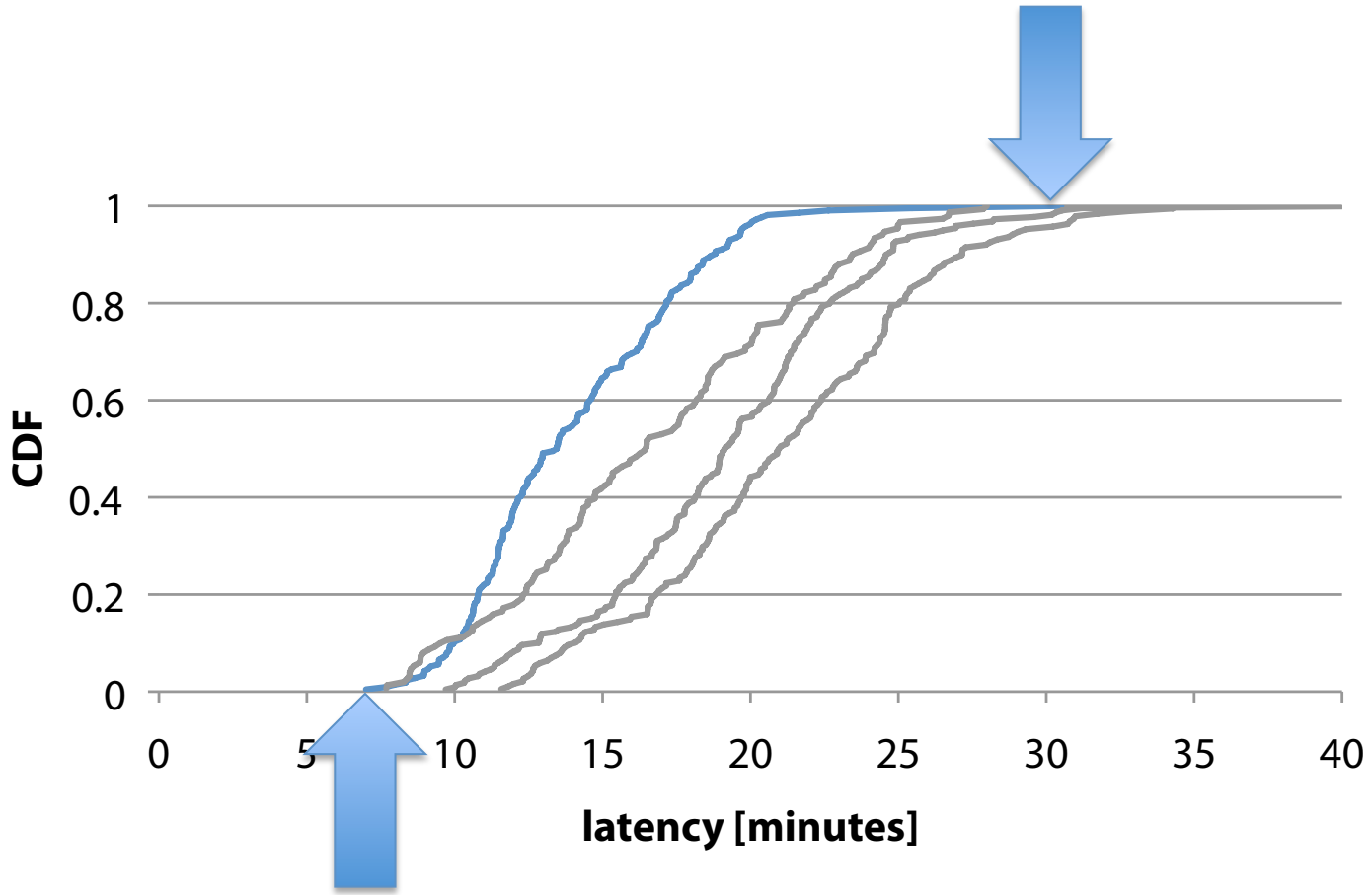
VARIABLE LATENCY



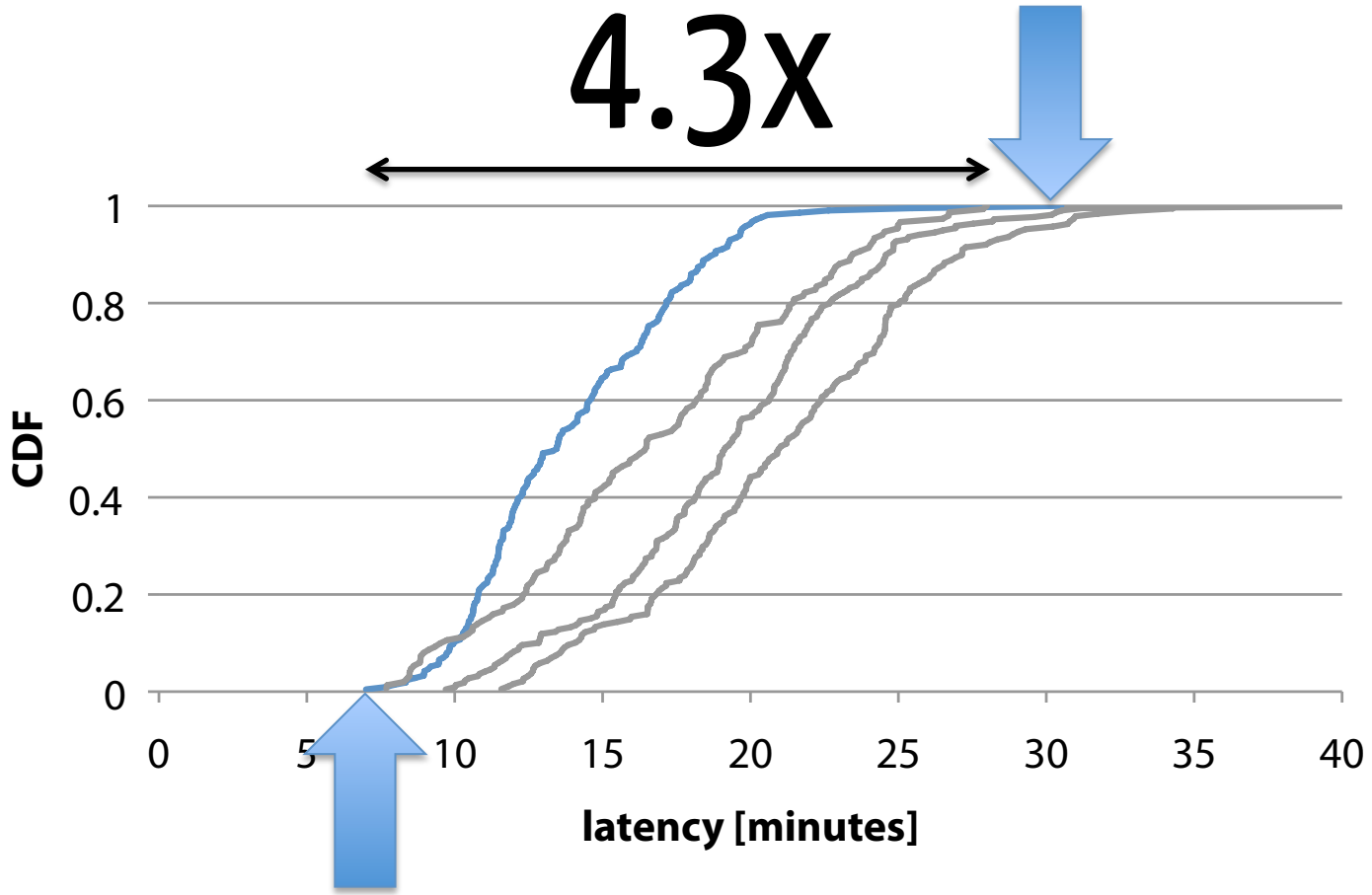
VARIABLE LATENCY



VARIABLE LATENCY



VARIABLE LATENCY



VARIABLE LATENCY

Why does latency vary?

1. Pipeline complexity
2. Noisy execution environment

Cosmos

MICROSOFT'S DATA PARALLEL CLUSTERS

- CosmosStore

Cosmos

MICROSOFT'S DATA PARALLEL CLUSTERS

- CosmosStore
- Dryad

Cosmos

MICROSOFT'S DATA PARALLEL CLUSTERS

- CosmosStore
- Dryad
- SCOPE

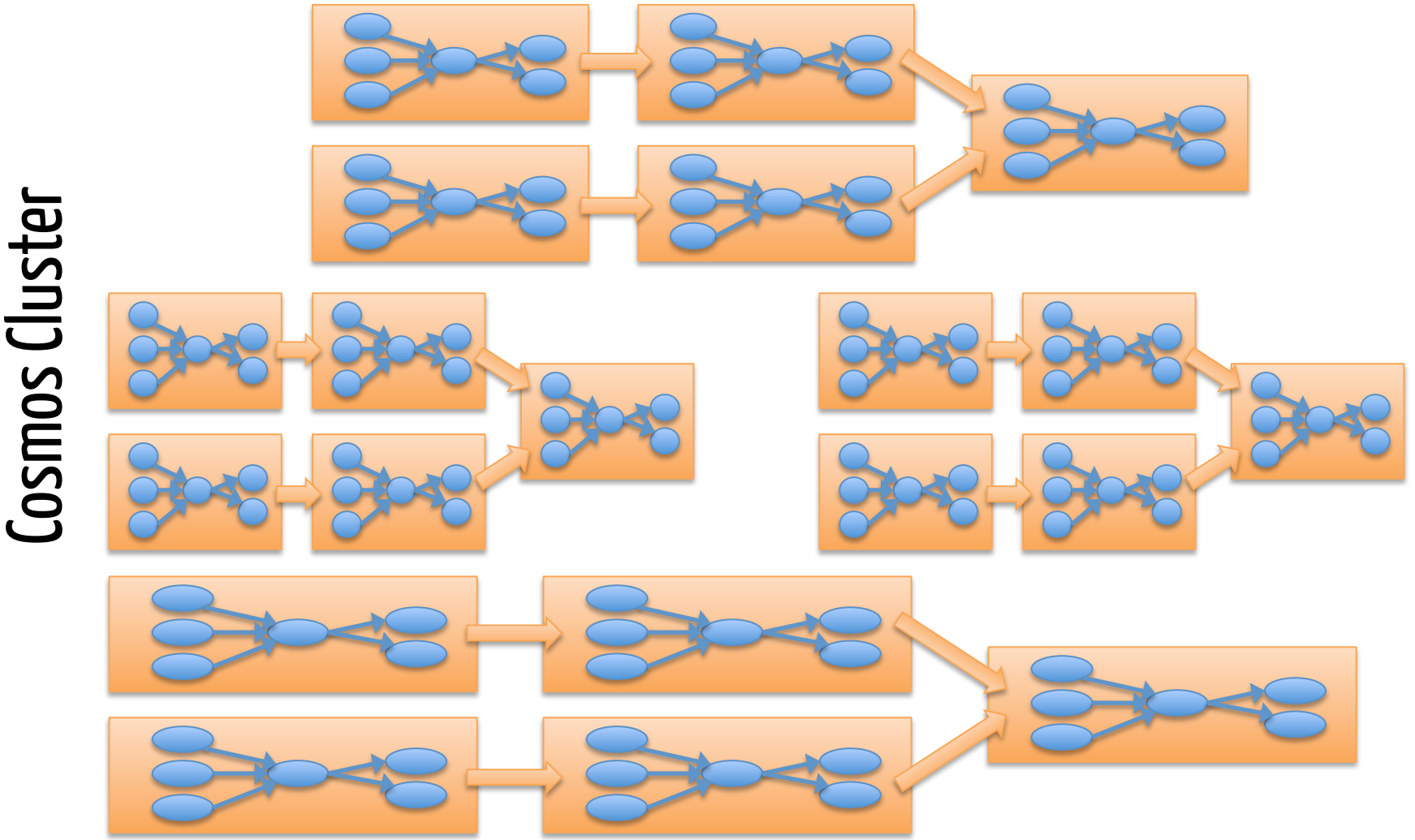
Cosmos

MICROSOFT'S DATA PARALLEL CLUSTERS

- CosmosStore
- **Dryad**
- SCOPE

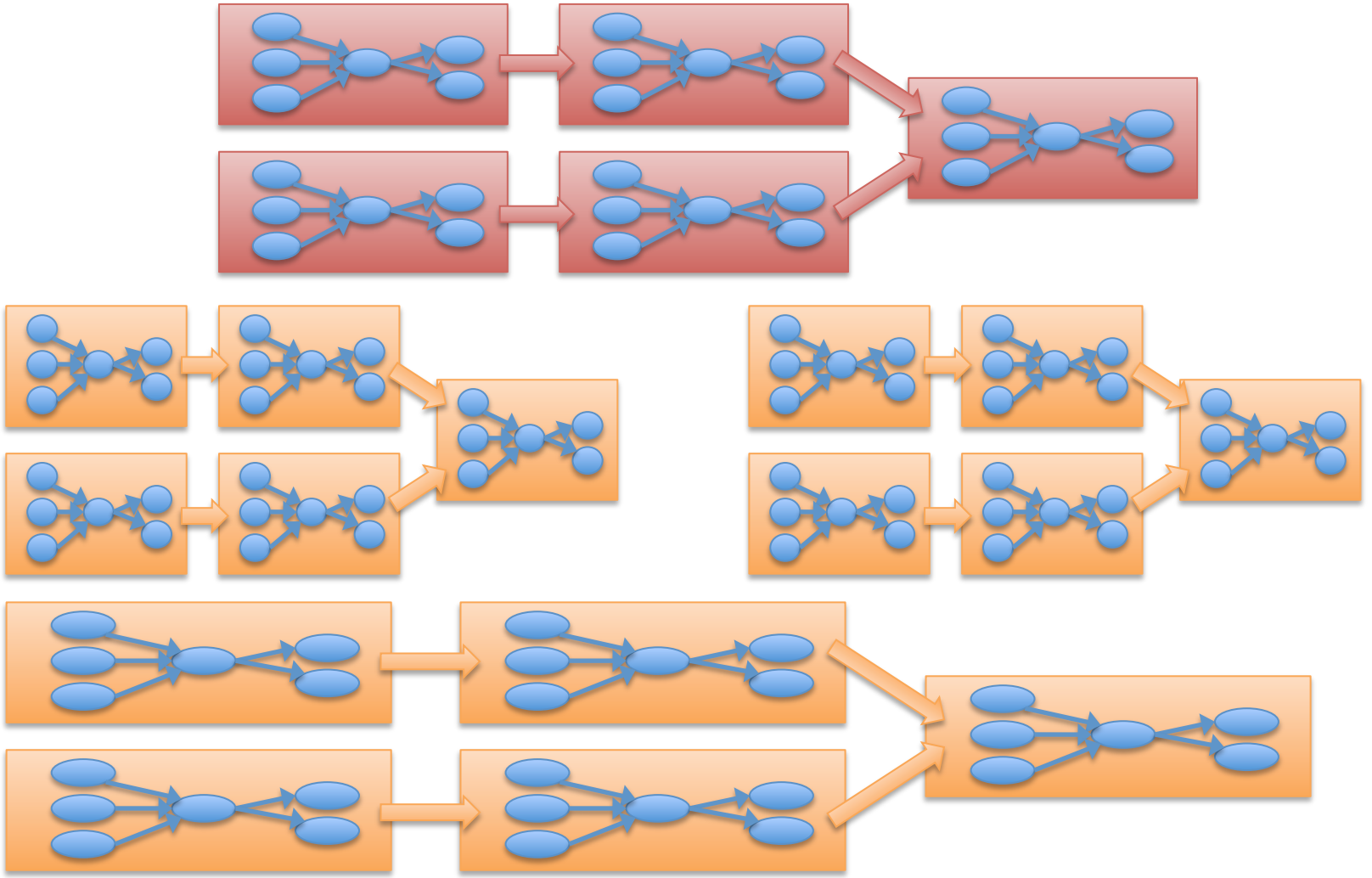
Cosmos

MICROSOFT'S DATA PARALLEL CLUSTERS

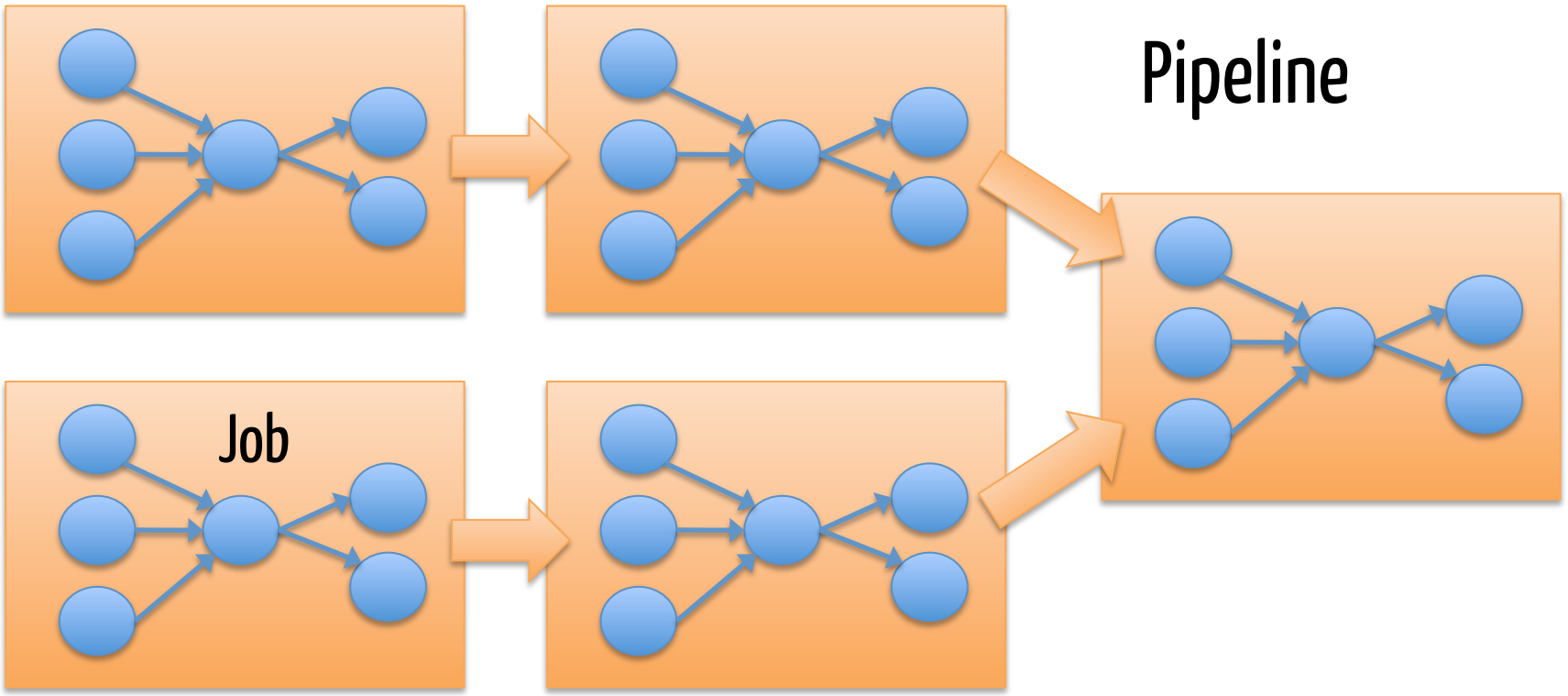


DRYAD'S DAG WORKFLOW

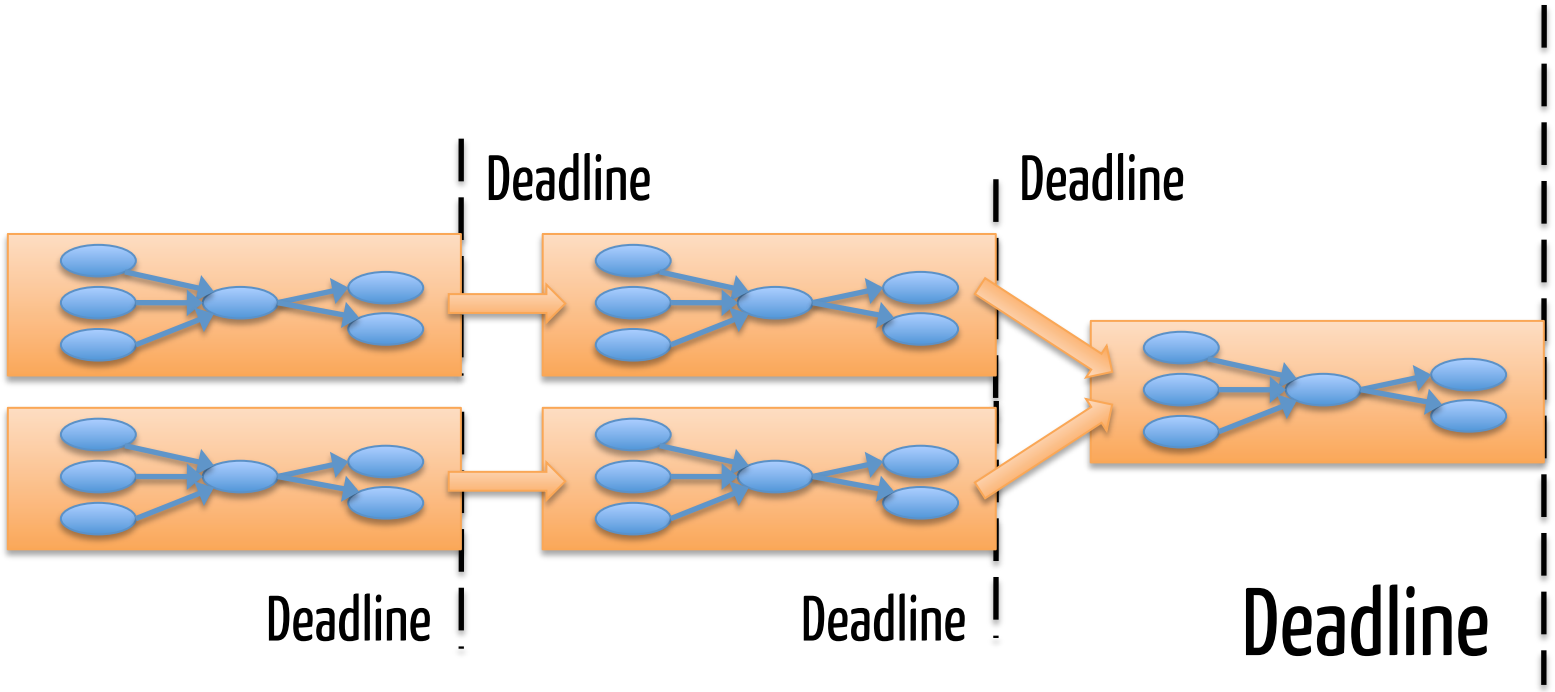
Cosmos Cluster



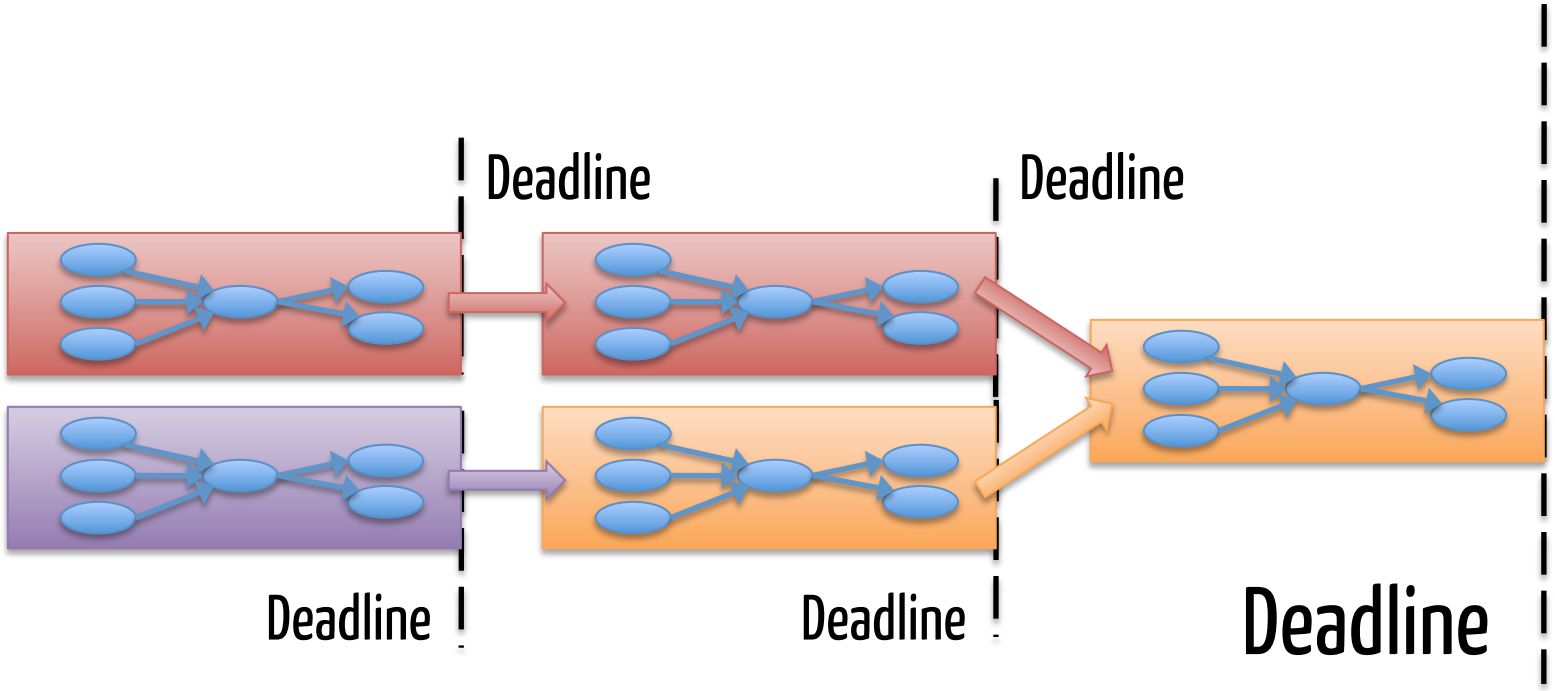
DRYAD'S DAG WORKFLOW



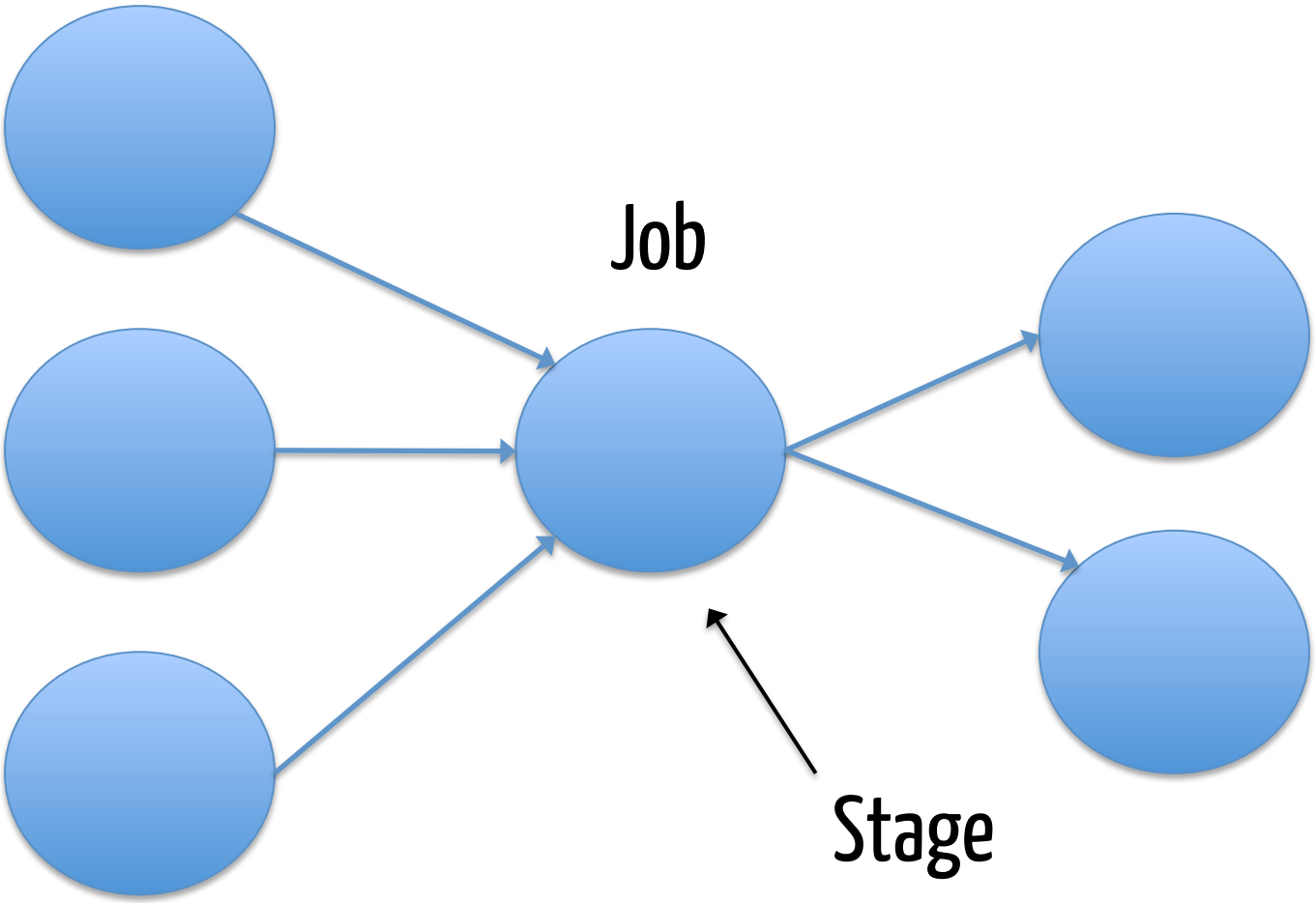
DRYAD'S DAG WORKFLOW



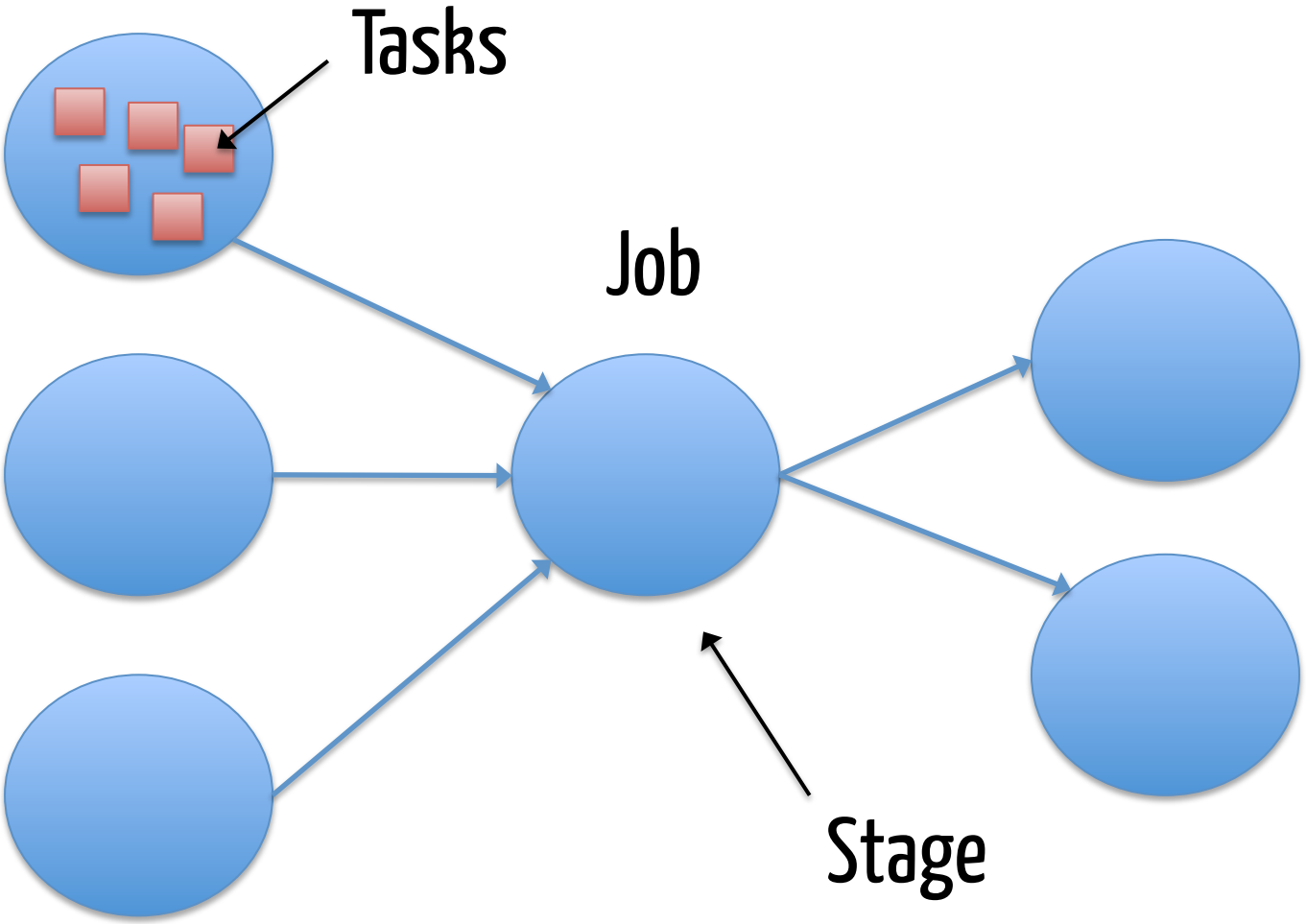
DRYAD'S DAG WORKFLOW



DRYAD'S DAG WORKFLOW



DRYAD'S DAG WORKFLOW



DRYAD'S DAG WORKFLOW

Priorities?

EXPRESSING PERFORMANCE TARGETS

Priorities? Not expressive enough

Weights?

EXPRESSING PERFORMANCE TARGETS

 Priorities? Not expressive enough

 Weights? Difficult for users to set

Utility curves?

EXPRESSING PERFORMANCE TARGETS

 Priorities? Not expressive enough

 Weights? Difficult for users to set

 Utility curves? Capture deadline & penalty

EXPRESSING PERFORMANCE TARGETS

OUR GOAL

Maximize utility

OUR GOAL

Maximize utility

while minimizing resources

OUR GOAL

Maximize utility

while minimizing resources

by dynamically adjusting the allocation

OUR GOAL

Jockey

- Large clusters

Jockey

- Large clusters
- Many users

Jockey

- Large clusters
- Many users
- Prior execution

Jockey

f(,) ->

JOCKEY – MODEL

f(job state,) ->

JOCKEY – MODEL

f(job state, allocation) ->

JOCKEY – MODEL

f(job state, allocation) -> remaining run time

JOCKEY – MODEL

JOCKEY – CONTROL LOOP

JOCKEY – CONTROL LOOP

JOCKEY – CONTROL LOOP

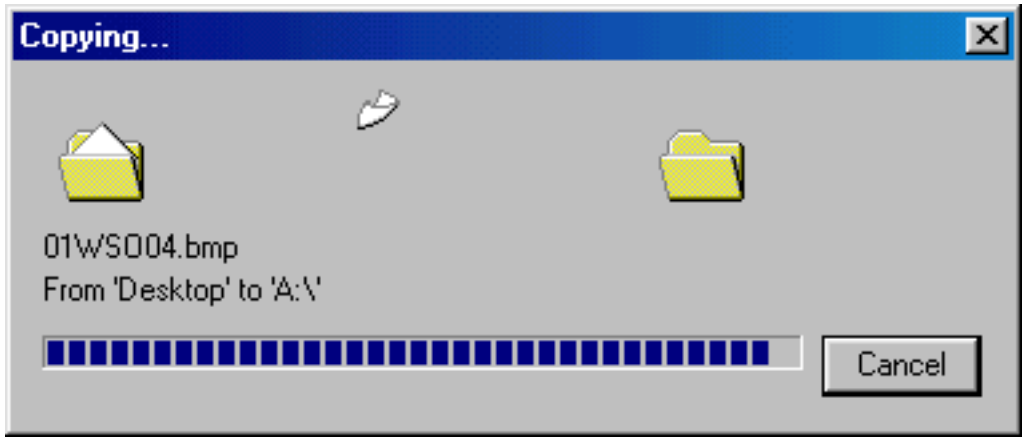
f(job state, allocation) -> remaining run time

JOCKEY – MODEL

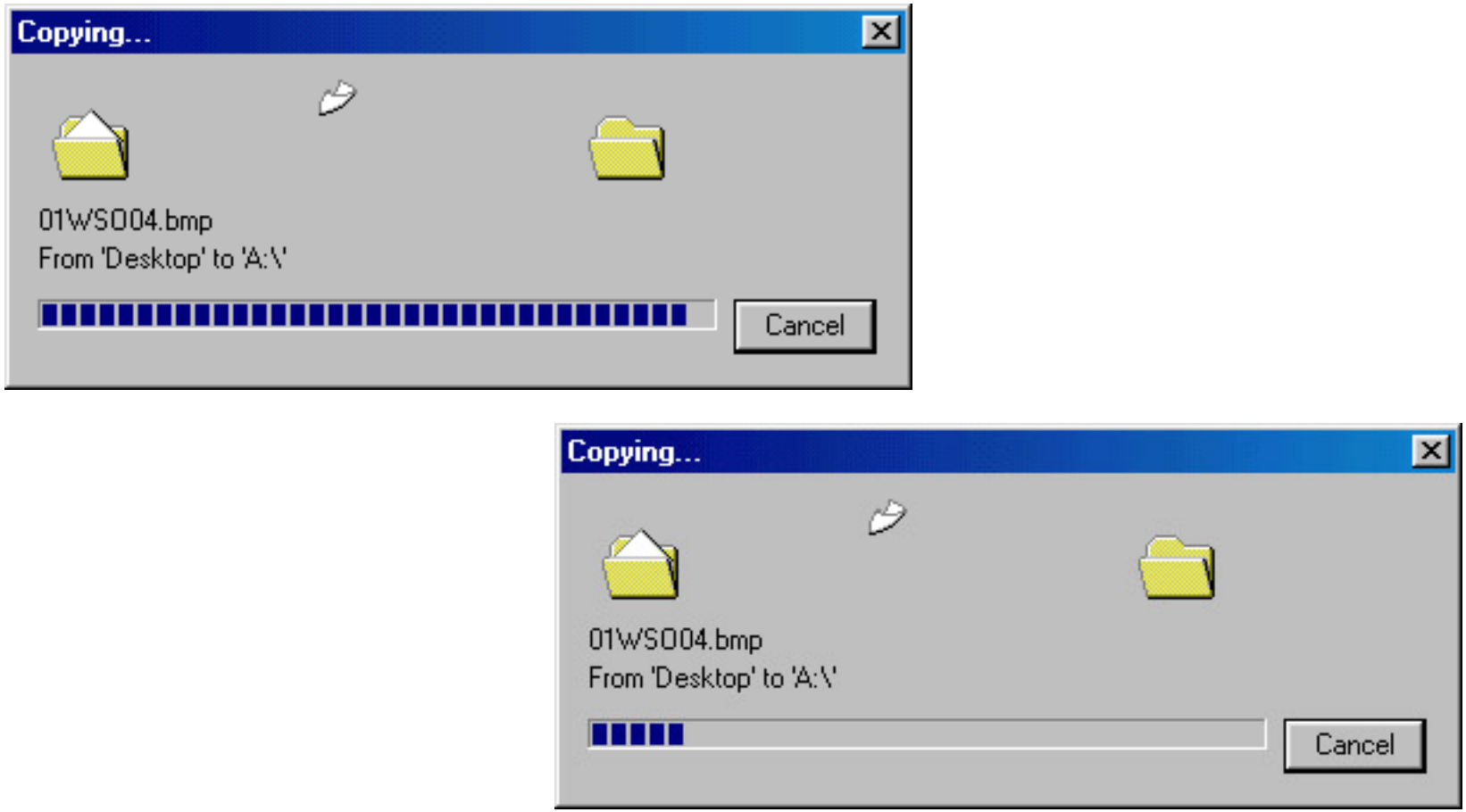
f(job state, allocation) -> remaining run time

f(progress, allocation) -> remaining run time

JOCKEY – MODEL



JOCKEY – PROGRESS INDICATOR



JOCKEY – PROGRESS INDICATOR

JOCKEY – PROGRESS INDICATOR

total running

JOCKEY – PROGRESS INDICATOR

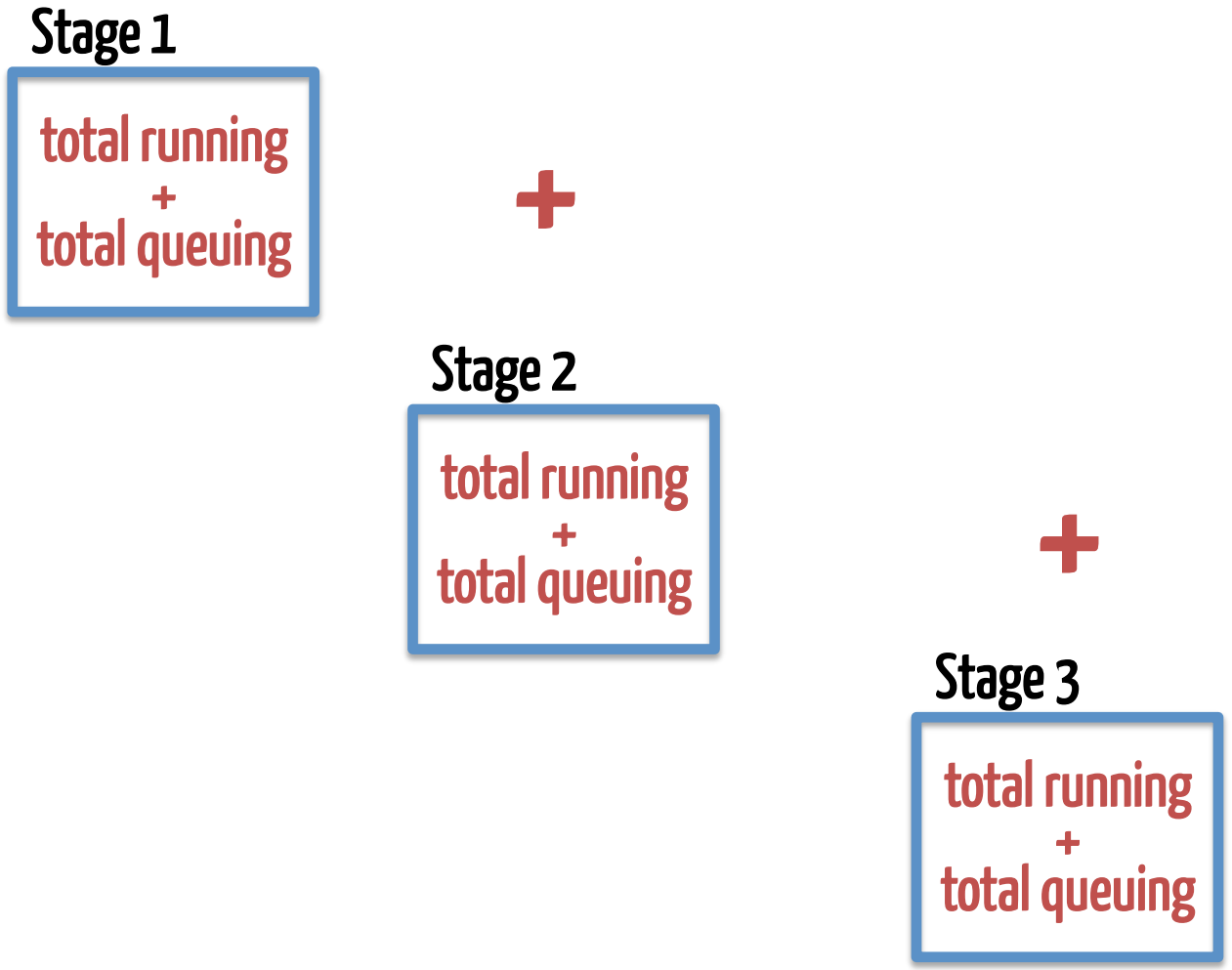
total running
+
total queuing

JOCKEY – PROGRESS INDICATOR

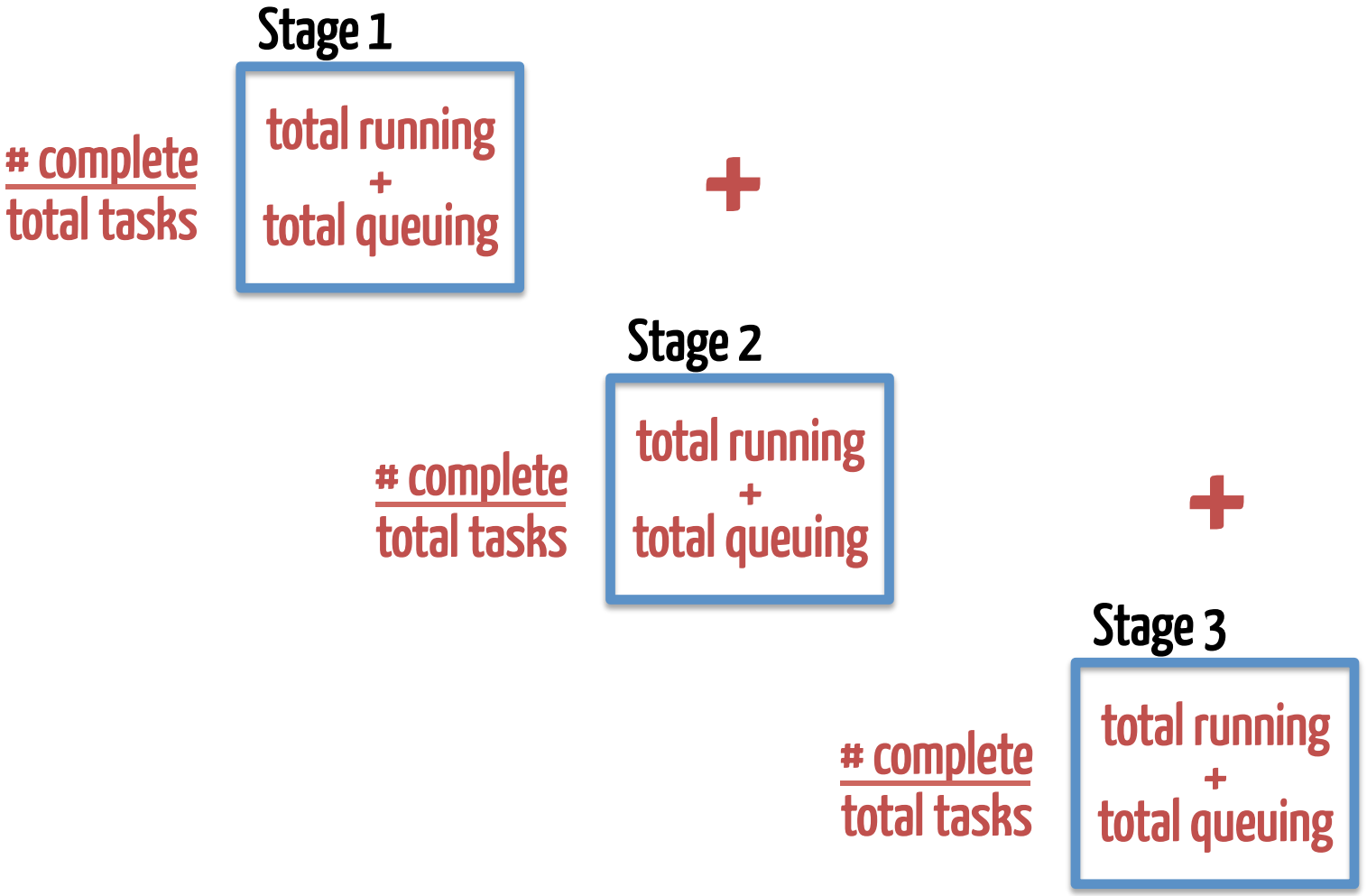
stage

total running
+
total queuing

JOCKEY – PROGRESS INDICATOR



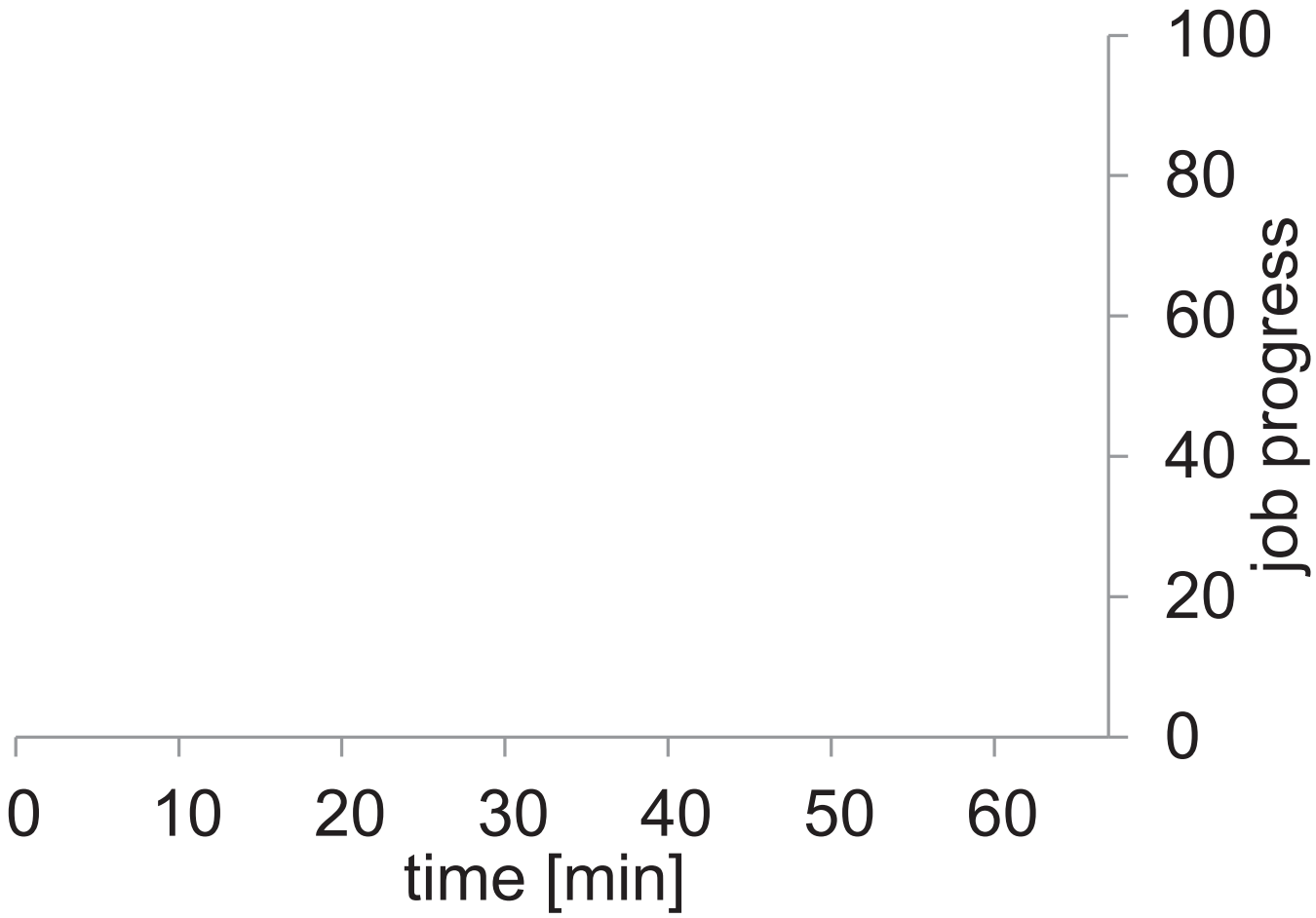
JOCKEY – PROGRESS INDICATOR



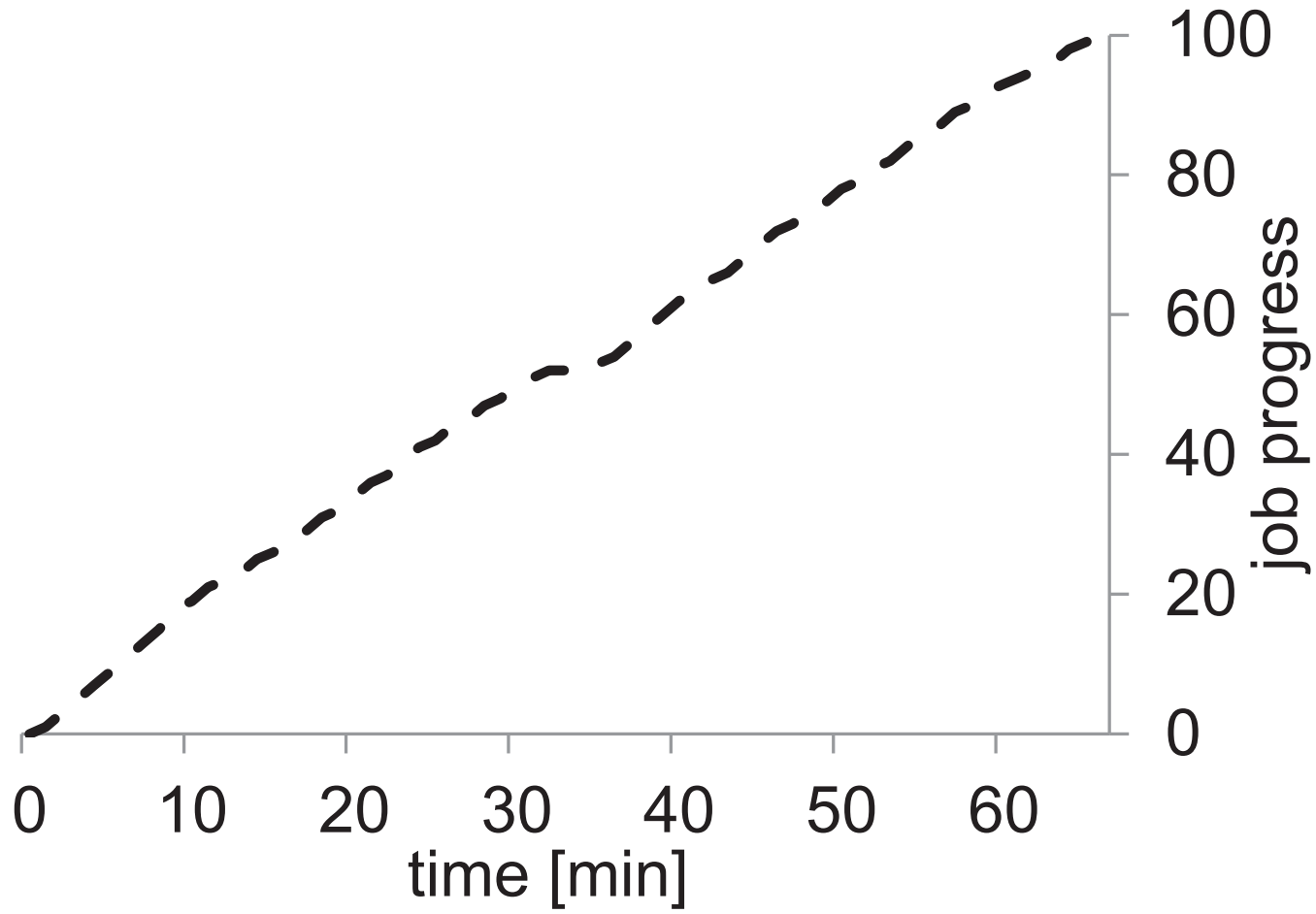
JOCKEY – PROGRESS INDICATOR

JOCKEY – PROGRESS INDICATOR

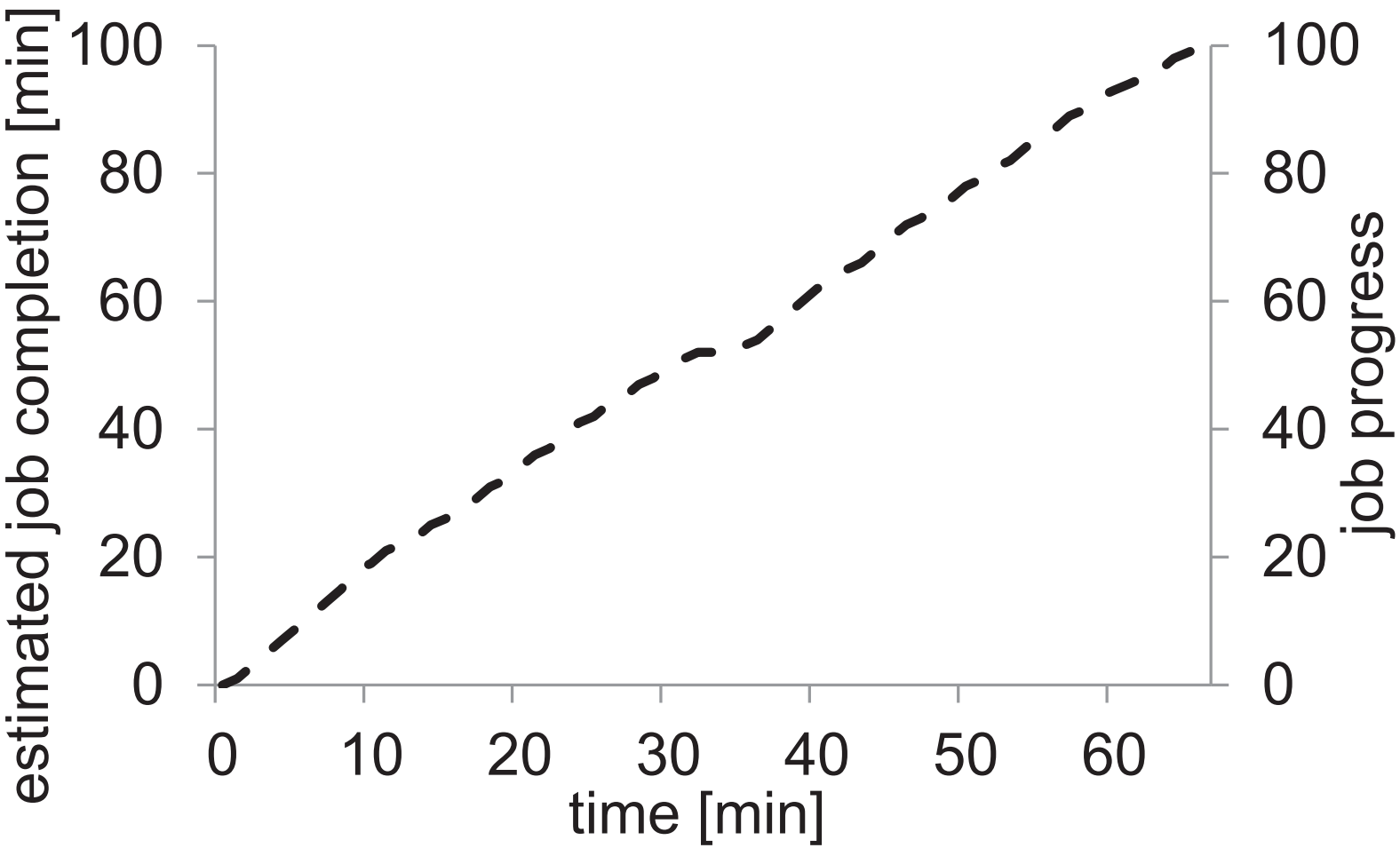
JOCKEY – PROGRESS INDICATOR



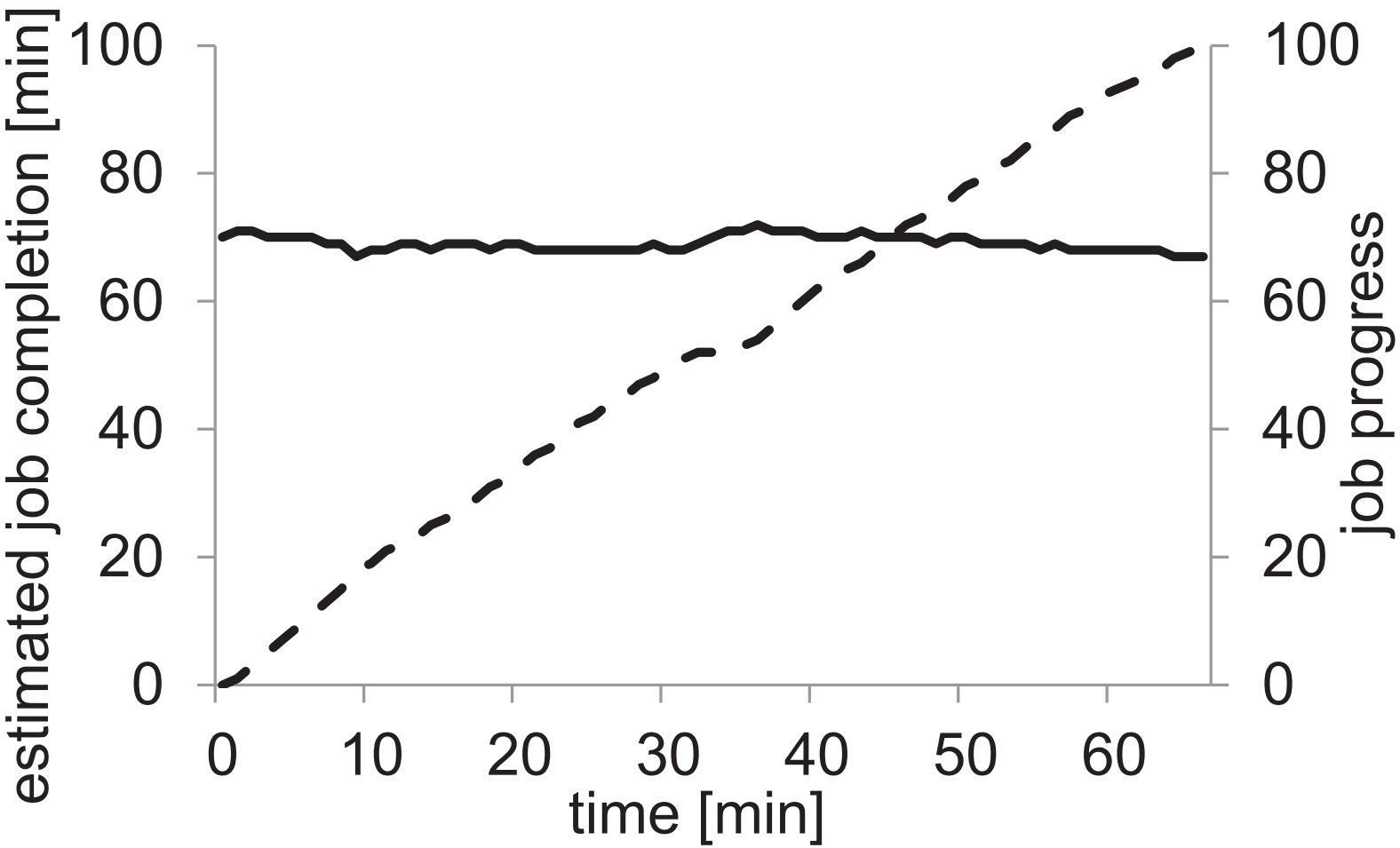
JOCKEY – PROGRESS INDICATOR



JOCKEY – PROGRESS INDICATOR



JOCKEY – PROGRESS INDICATOR



JOCKEY – PROGRESS INDICATOR

JOCKEY – CONTROL LOOP

Job model

1% complete			
2% complete			
3% complete			
4% complete			
5% complete			

JOCKEY – CONTROL LOOP

Job model

	10 nodes	20 nodes	30 nodes
1% complete			
2% complete			
3% complete			
4% complete			
5% complete			

JOCKEY – CONTROL LOOP

Job model

	10 nodes	20 nodes	30 nodes
1% complete	60 minutes	40 minutes	25 minutes
2% complete	59 minutes	39 minutes	24 minutes
3% complete	58 minutes	37 minutes	22 minutes
4% complete	56 minutes	36 minutes	21 minutes
5% complete	54 minutes	34 minutes	20 minutes

JOCKEY – CONTROL LOOP

Job model

	10 nodes	20 nodes	30 nodes
1% complete	60 minutes	40 minutes	25 minutes
2% complete	59 minutes	39 minutes	24 minutes
3% complete	58 minutes	37 minutes	22 minutes
4% complete	56 minutes	36 minutes	21 minutes
5% complete	54 minutes	34 minutes	20 minutes

Completion: 1%

Deadline: 50 min.

JOCKEY – CONTROL LOOP

Job model

	10 nodes	20 nodes	30 nodes
1% complete	60 minutes	40 minutes	25 minutes
2% complete	59 minutes	39 minutes	24 minutes
3% complete	58 minutes	37 minutes	22 minutes
4% complete	56 minutes	36 minutes	21 minutes
5% complete	54 minutes	34 minutes	20 minutes

Completion: 1%

Deadline: 50 min.

JOCKEY – CONTROL LOOP

Job model

	10 nodes	20 nodes	30 nodes
1% complete	60 minutes	40 minutes	25 minutes
2% complete	59 minutes	39 minutes	24 minutes
3% complete	58 minutes	37 minutes	22 minutes
4% complete	56 minutes	36 minutes	21 minutes
5% complete	54 minutes	34 minutes	20 minutes

Completion:

3%

Deadline:

40 min.

JOCKEY – CONTROL LOOP

Job model

	10 nodes	20 nodes	30 nodes
1% complete	60 minutes	40 minutes	25 minutes
2% complete	59 minutes	39 minutes	24 minutes
3% complete	58 minutes	37 minutes	22 minutes
4% complete	56 minutes	36 minutes	21 minutes
5% complete	54 minutes	34 minutes	20 minutes

Completion:

5%

Deadline:

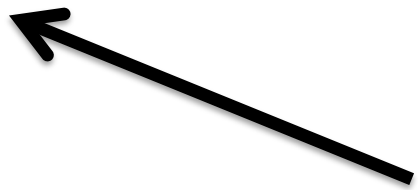
30 min.

JOCKEY – CONTROL LOOP

f(progress, allocation) -> remaining run time

JOCKEY – MODEL

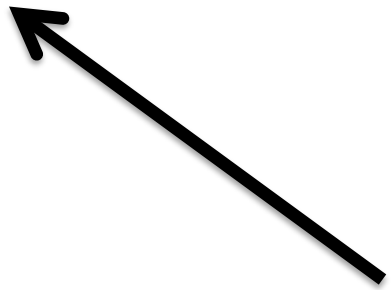
f(progress, allocation) -> remaining run time



analytic model?

JOCKEY – MODEL

f(progress, allocation) -> remaining run time

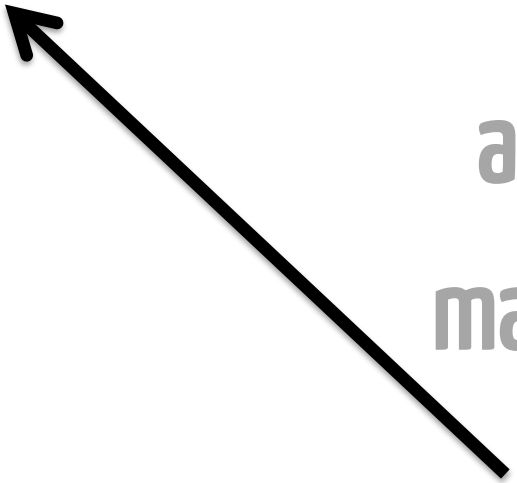


analytic model

machine learning?

JOCKEY – MODEL

f(progress, allocation) -> remaining run time



analytic model

machine learning

simulator

JOCKEY – MODEL

Problem	Solution

JOCKEY

Problem	Solution
Pipeline complexity	

JOCKEY

Problem	Solution
Pipeline complexity	Use a simulator

JOCKEY

Problem	Solution
Pipeline complexity	Use a simulator
Noisy environment	

JOCKEY

Problem	Solution
Pipeline complexity	Use a simulator
Noisy environment	Dynamic control

JOCKEY

Jockey in Action

- Real job

Jockey in Action

- Real job
- Production cluster

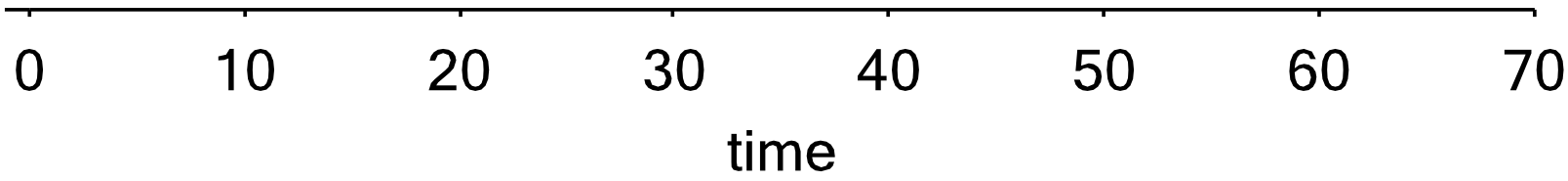
Jockey in Action

- Real job
- Production cluster
- CPU load: ~80%

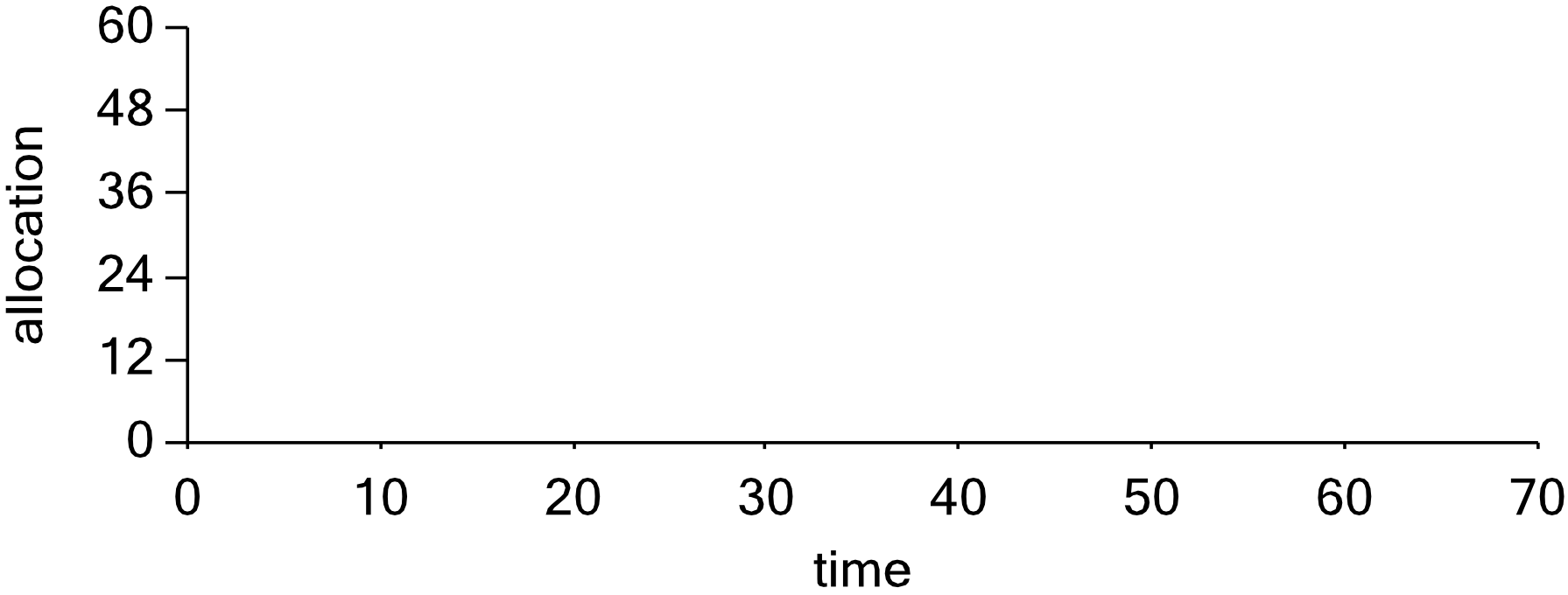
Jockey in Action

Jockey in Action

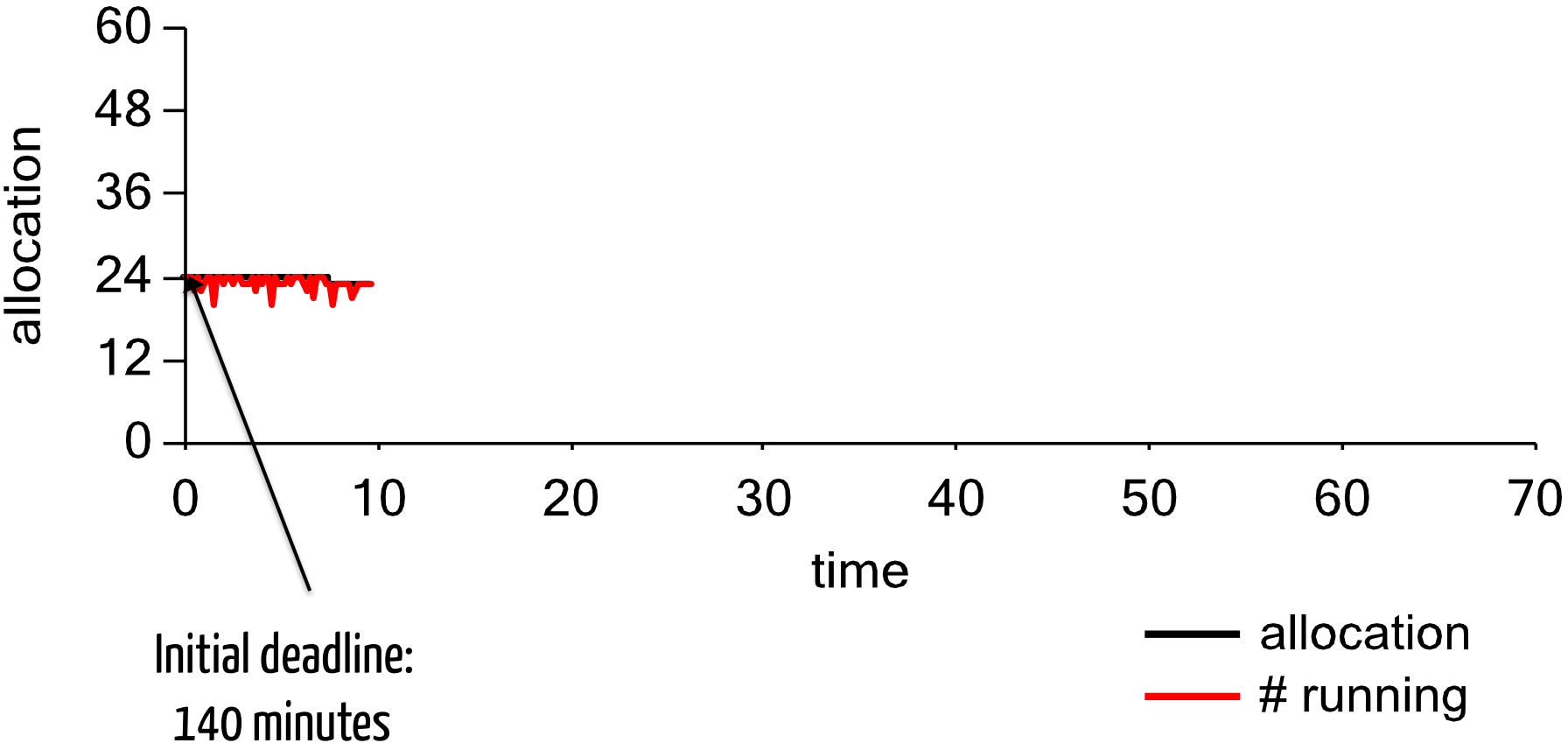
Jockey in Action



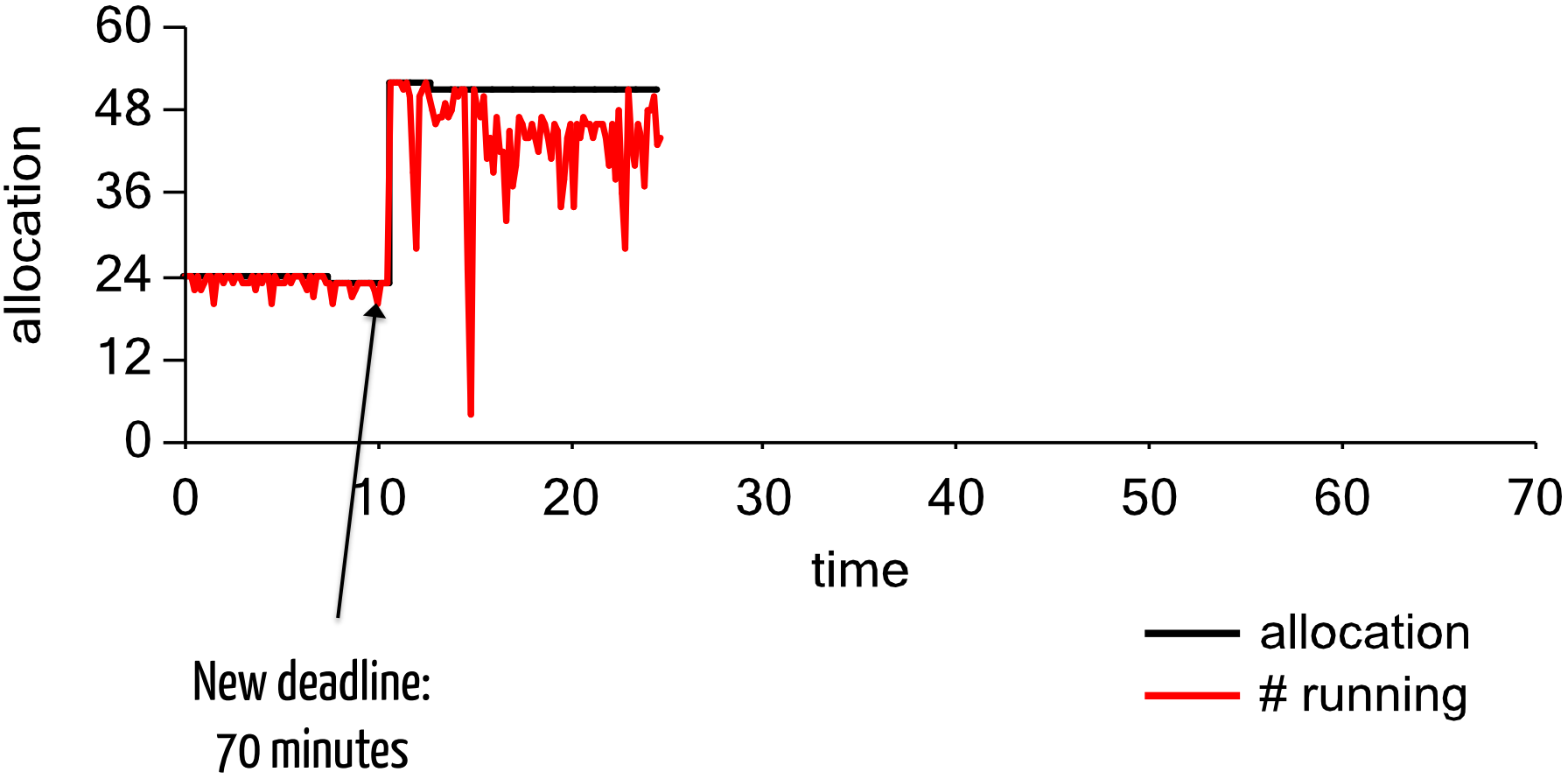
Jockey in Action



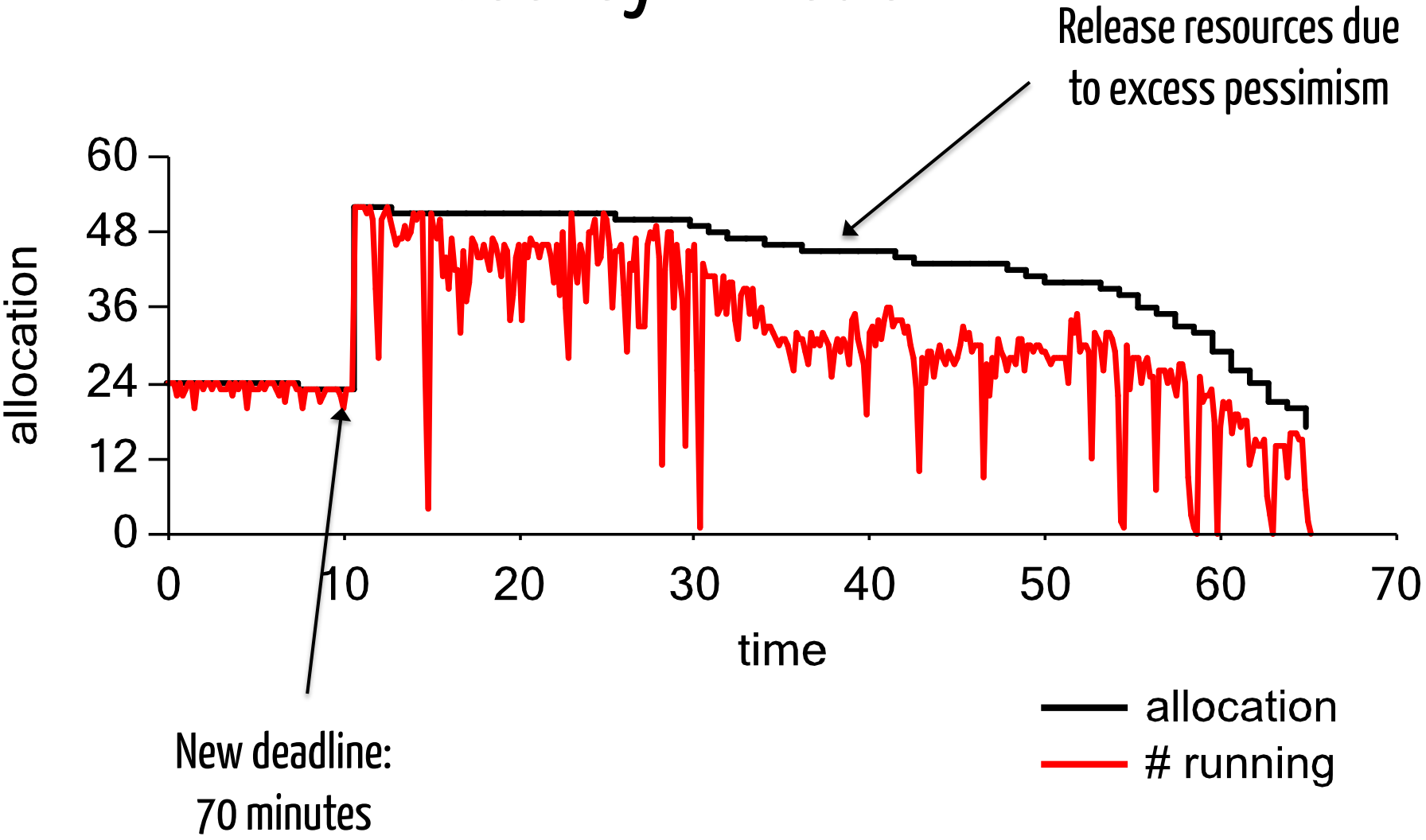
Jockey in Action



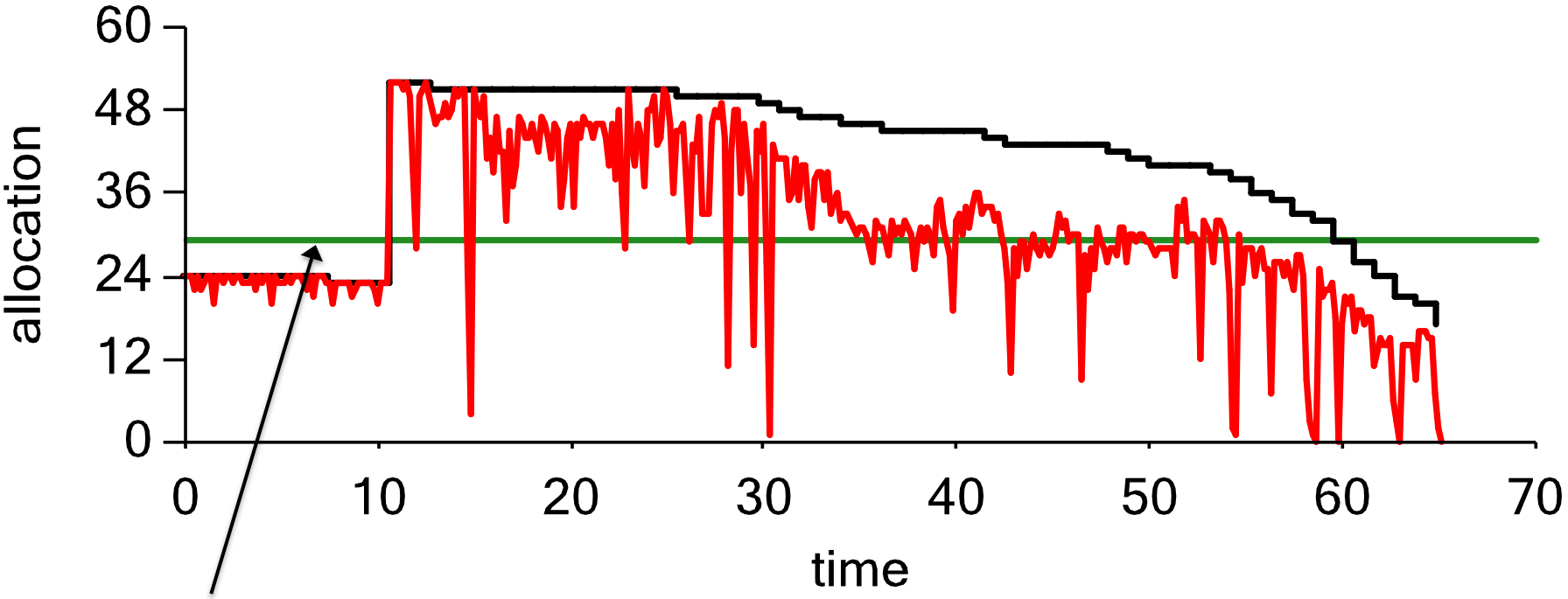
Jockey in Action



Jockey in Action



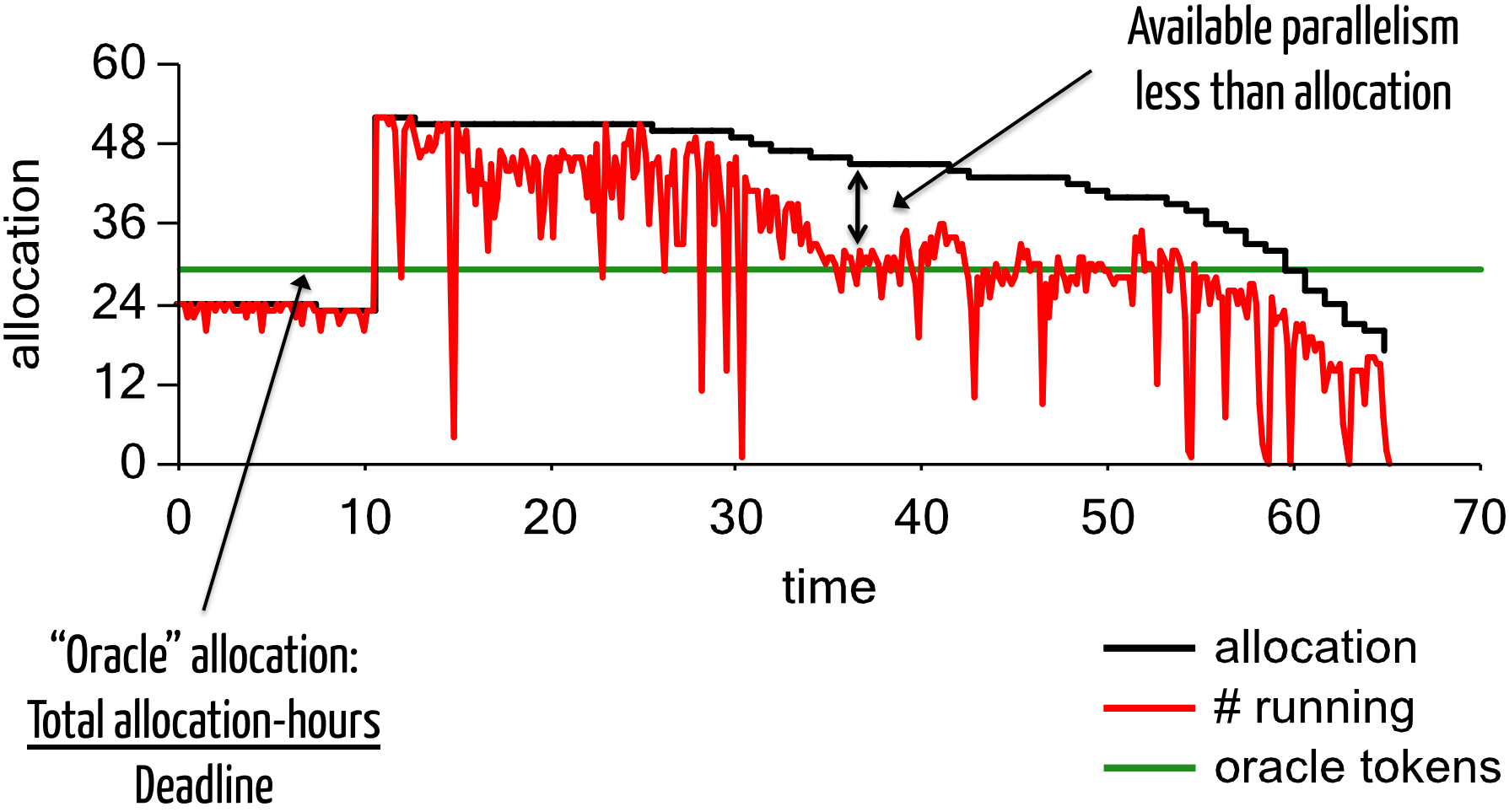
Jockey in Action



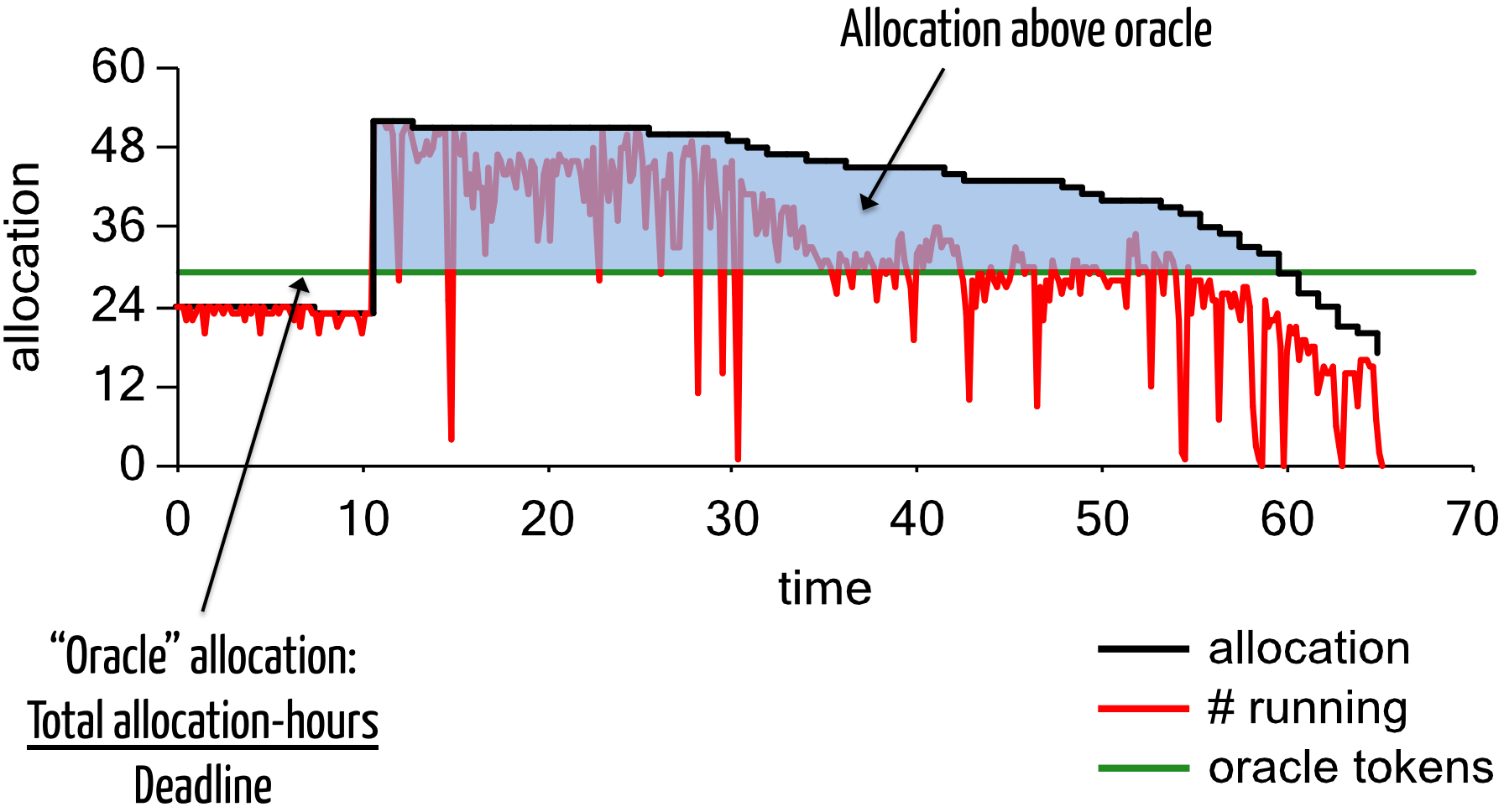
"Oracle" allocation:

$$\frac{\text{Total allocation-hours}}{\text{Deadline}}$$

Jockey in Action



Jockey in Action



Evaluation

- Production cluster

Evaluation

- Production cluster
- 21 jobs

Evaluation

- Production cluster
- 21 jobs
- SLO met?

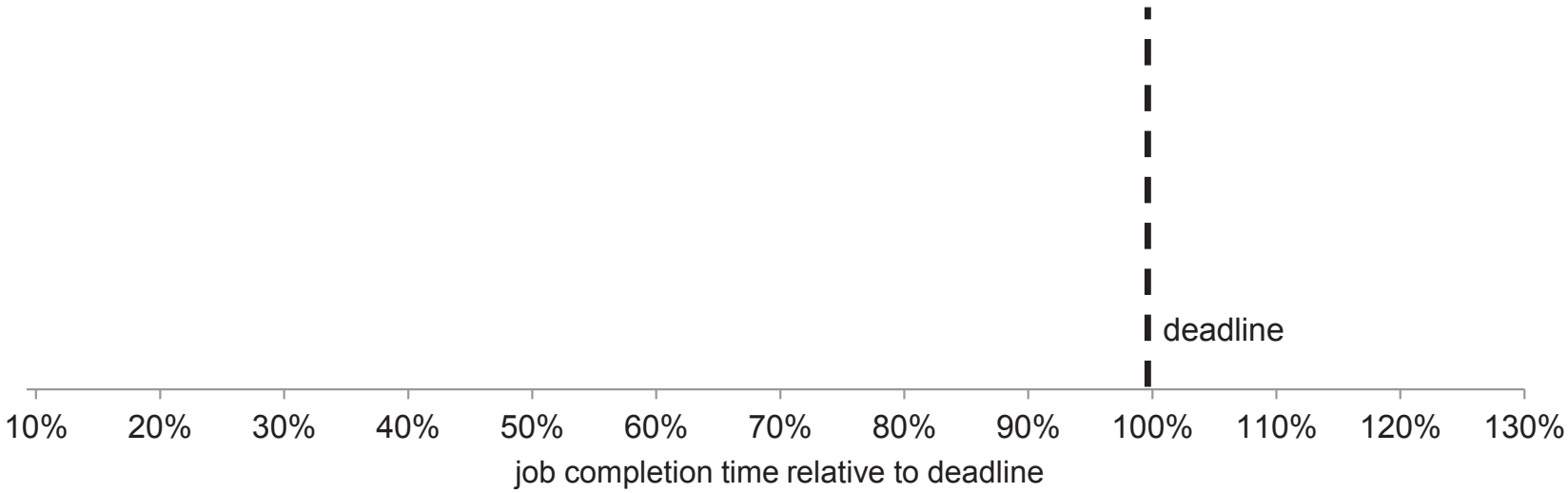
Evaluation

- Production cluster
- 21 jobs
- SLO met?
- Cluster impact?

Evaluation

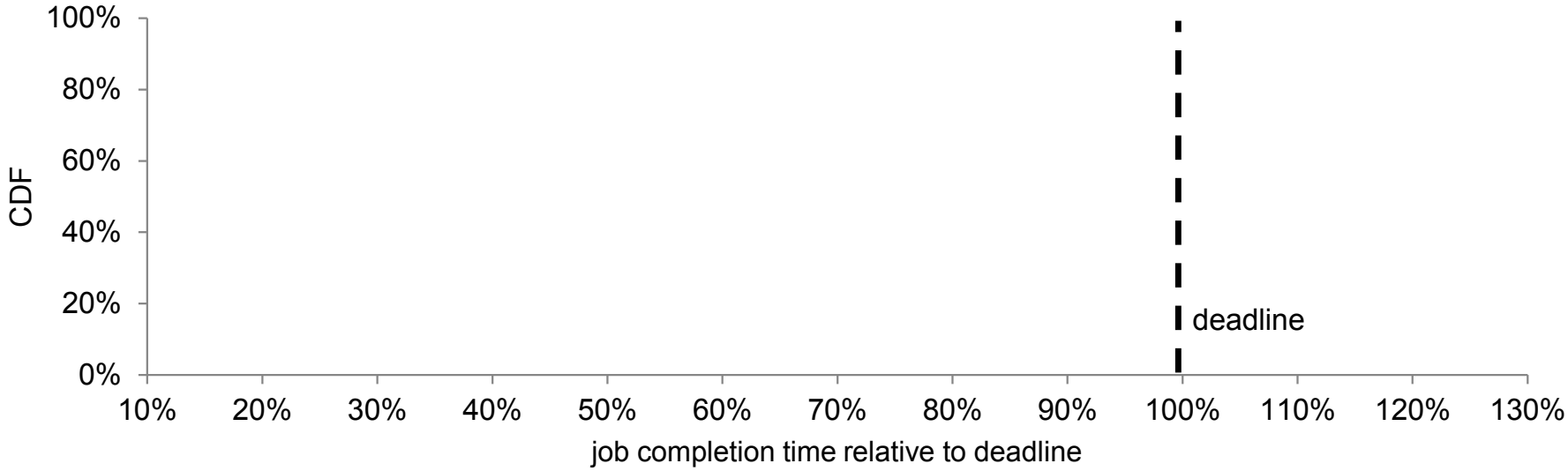
Evaluation

Evaluation



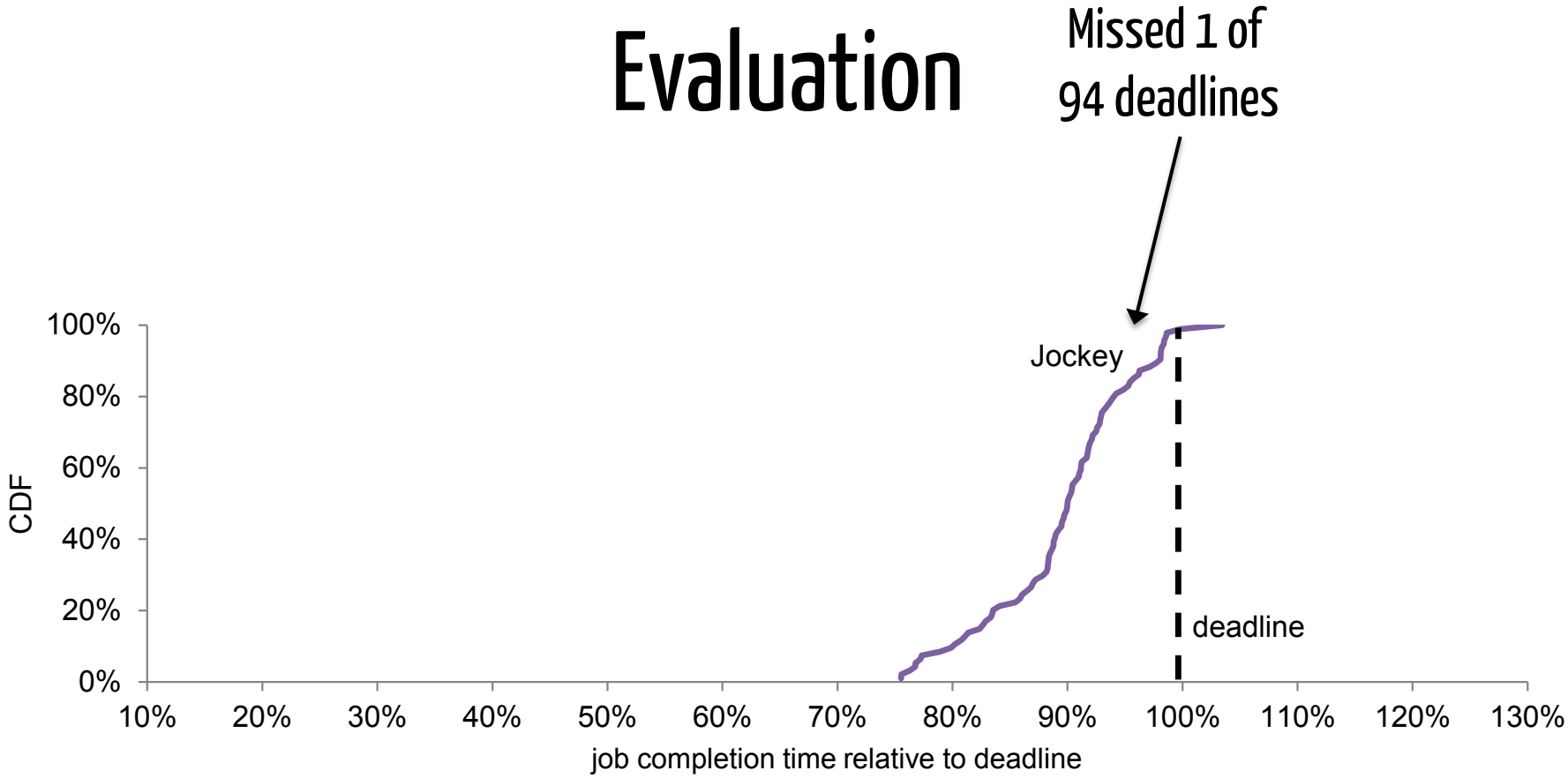
Jobs which met the SLO

Evaluation



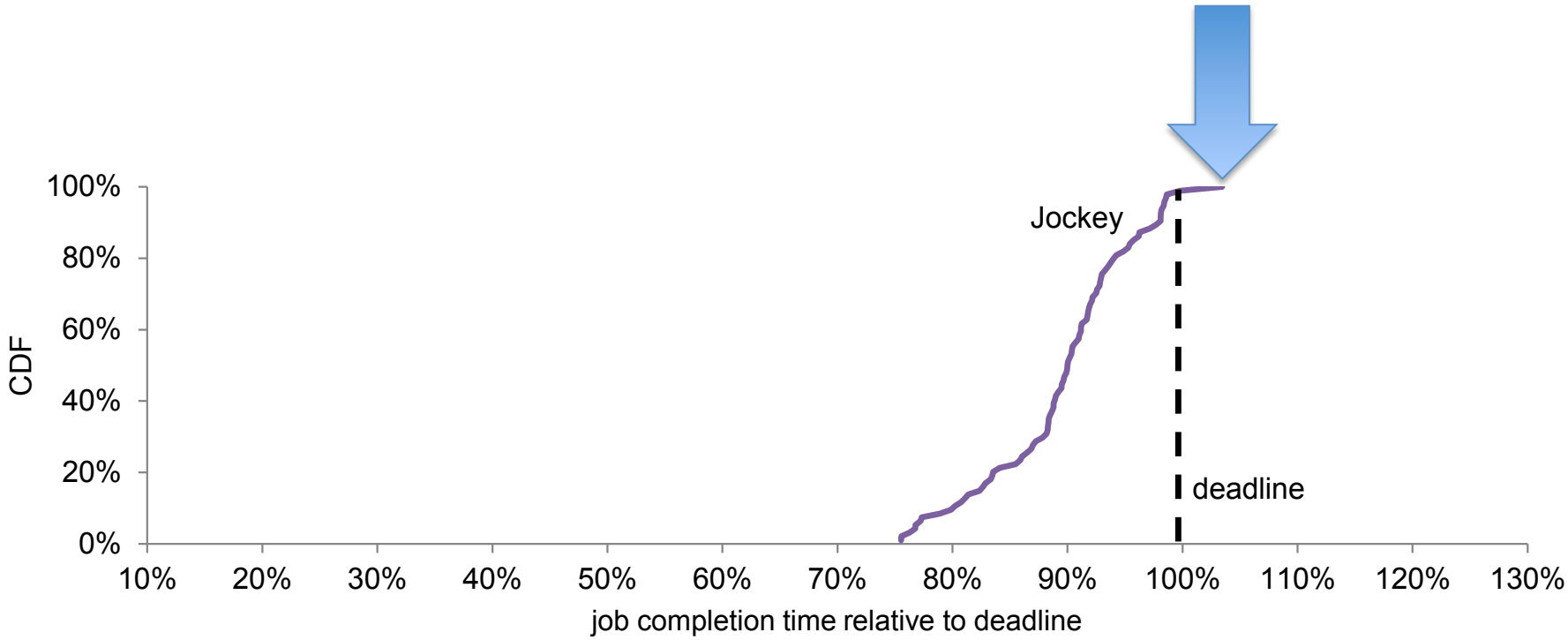
Jobs which met the SLO

Evaluation



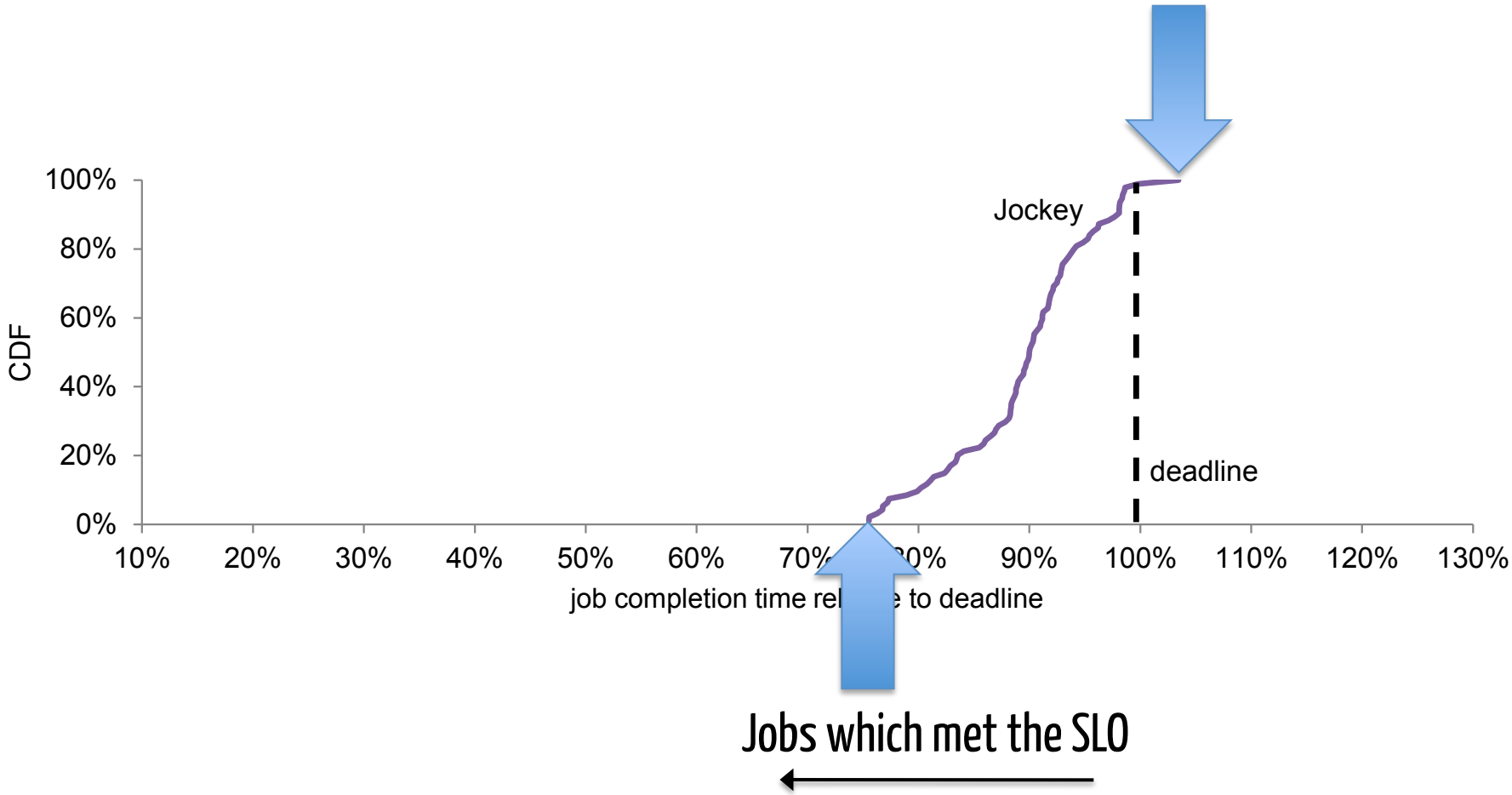
Jobs which met the SLO

Evaluation

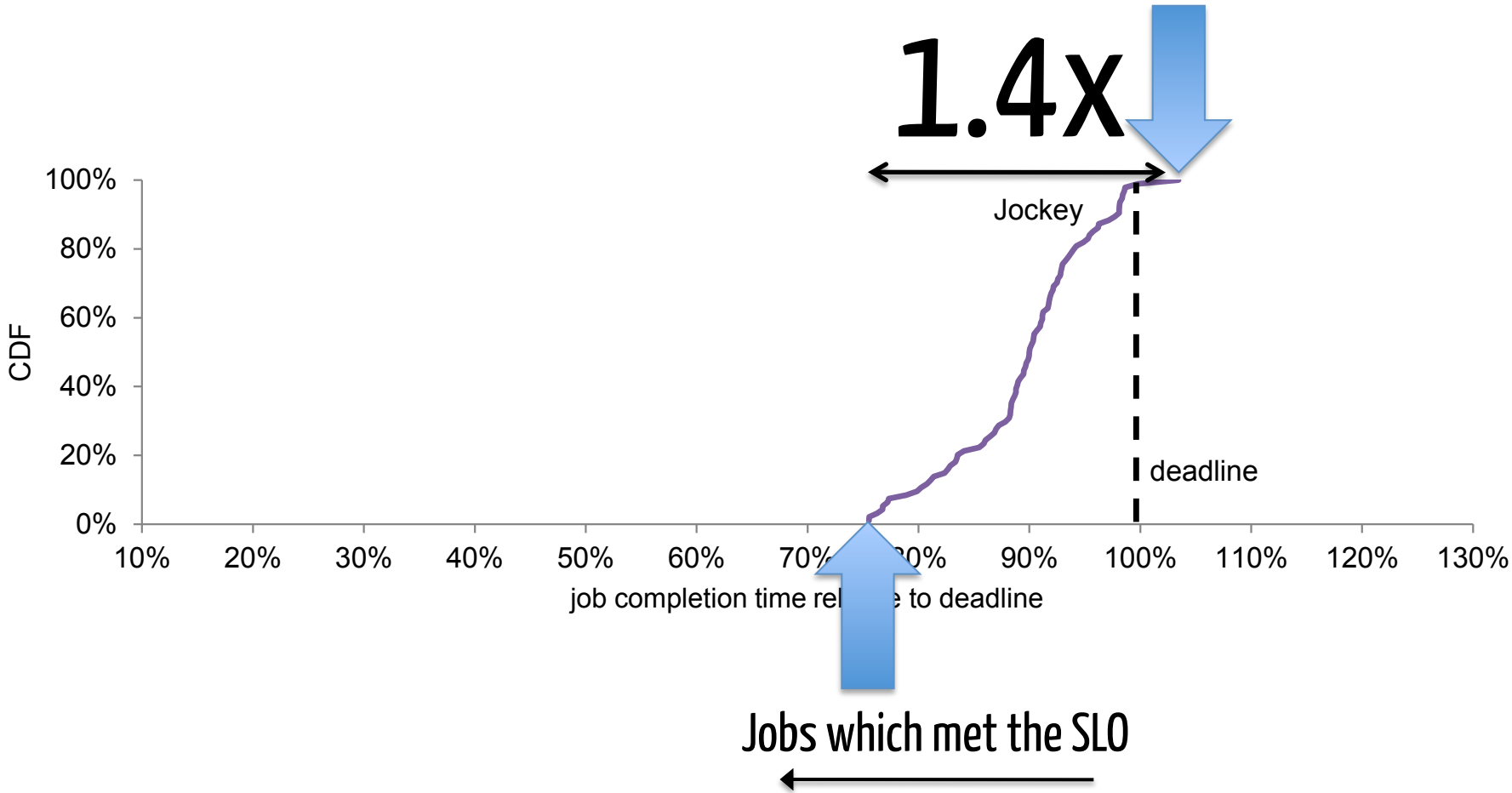


Jobs which met the SLO

Evaluation


100%
80%
60%
40%
20%
0%
CDF
Jockey
deadline
10%
20%
30%
40%
50%
60%
70%
90%
100%
110%
120%
130%
to deadline
Jobs which met the SLO

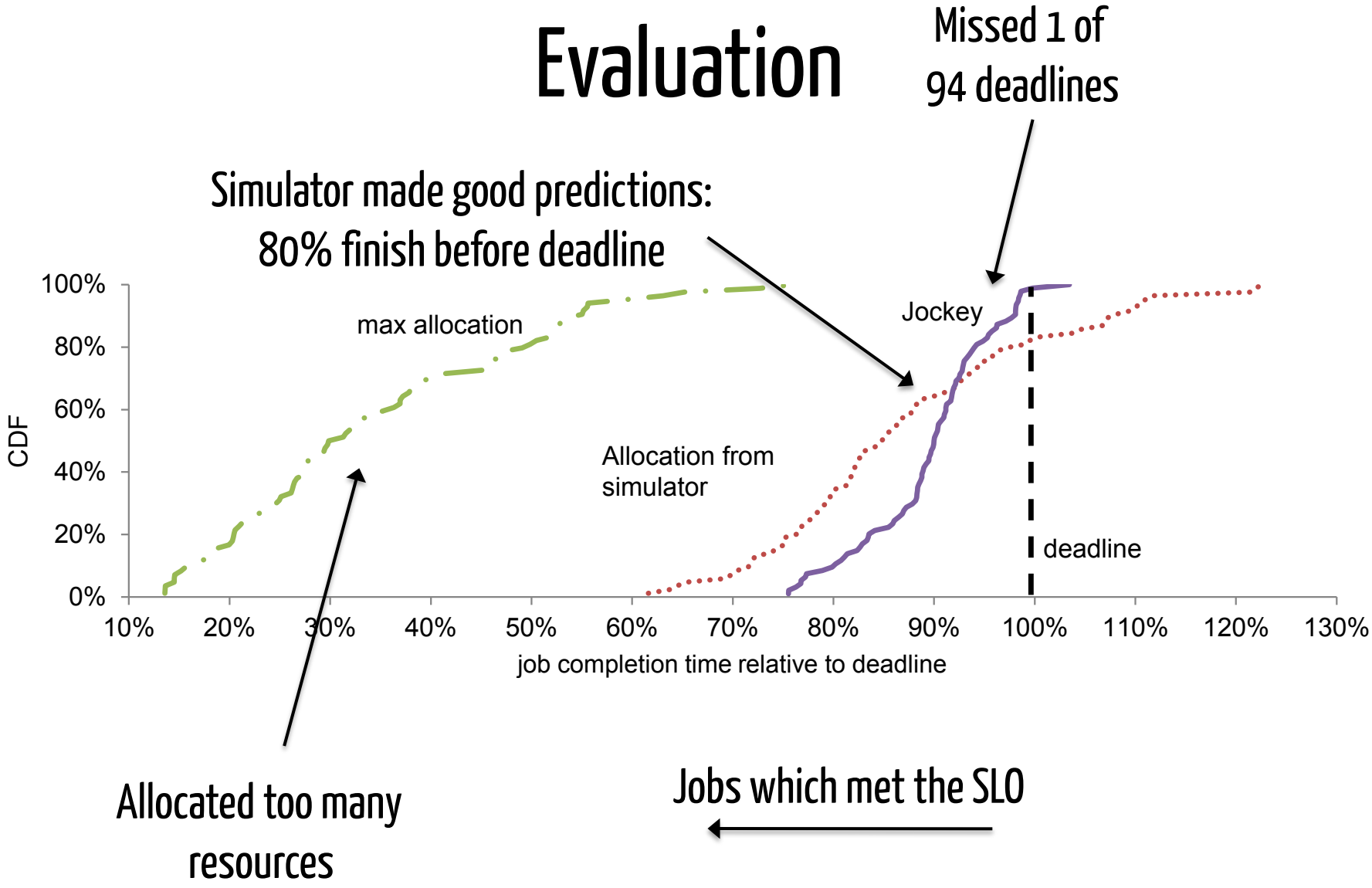
Evaluation


1.4x
Jockey
deadline
CDF
100%
80%
60%
40%
20%
0%
10%
20%
30%
40%
50%
60%
70%
90%
100%
110%
120%
130%
job completion time rel e to deadline
Jobs which met the SLO

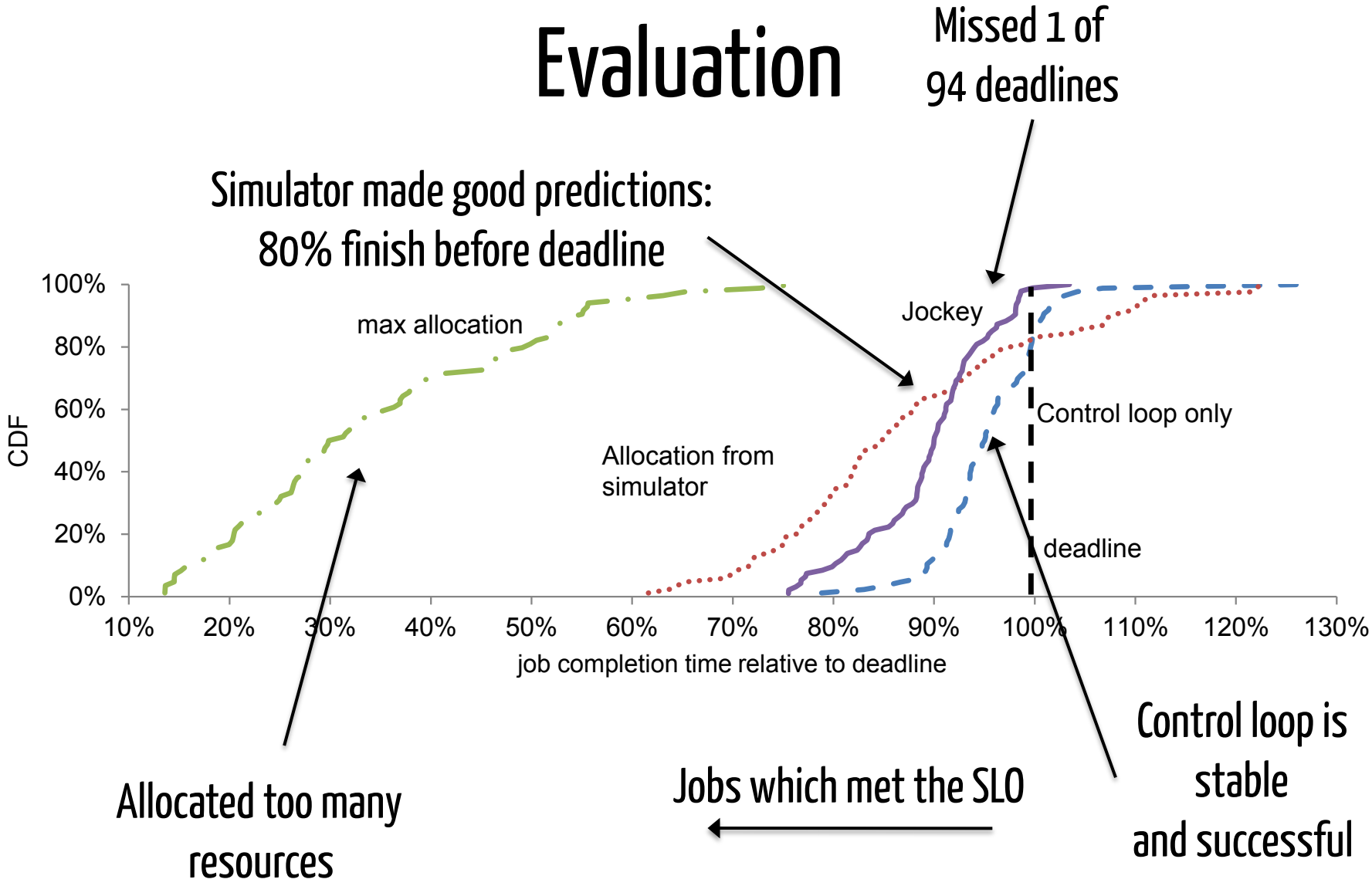
Evaluation


Missed 1 of
94 deadlines
100%
80%
60%
40%
20%
0%
CDF
max allocation
Jockey
deadline
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
110%
120%
130%
job completion time relative to deadline
Allocated too many
resources
Jobs which met the SLO

Evaluation

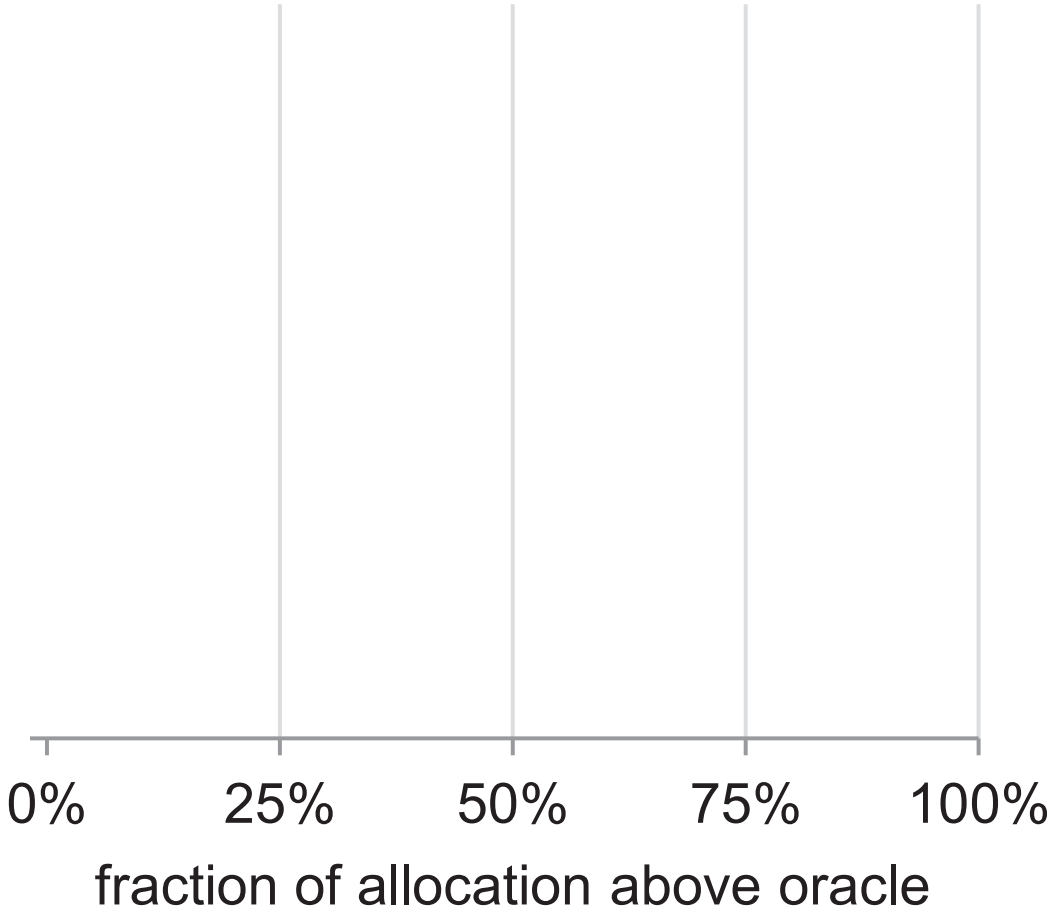

Missed 1 of 94 deadlines
Simulator made good predictions: 80% finish before deadline
100%
80%
60%
40%
20%
0%
CDF
max allocation
Jockey
Allocation from simulator
deadline
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
110%
120%
130%
job completion time relative to deadline
Allocated too many resources
Jobs which met the SLO

Evaluation

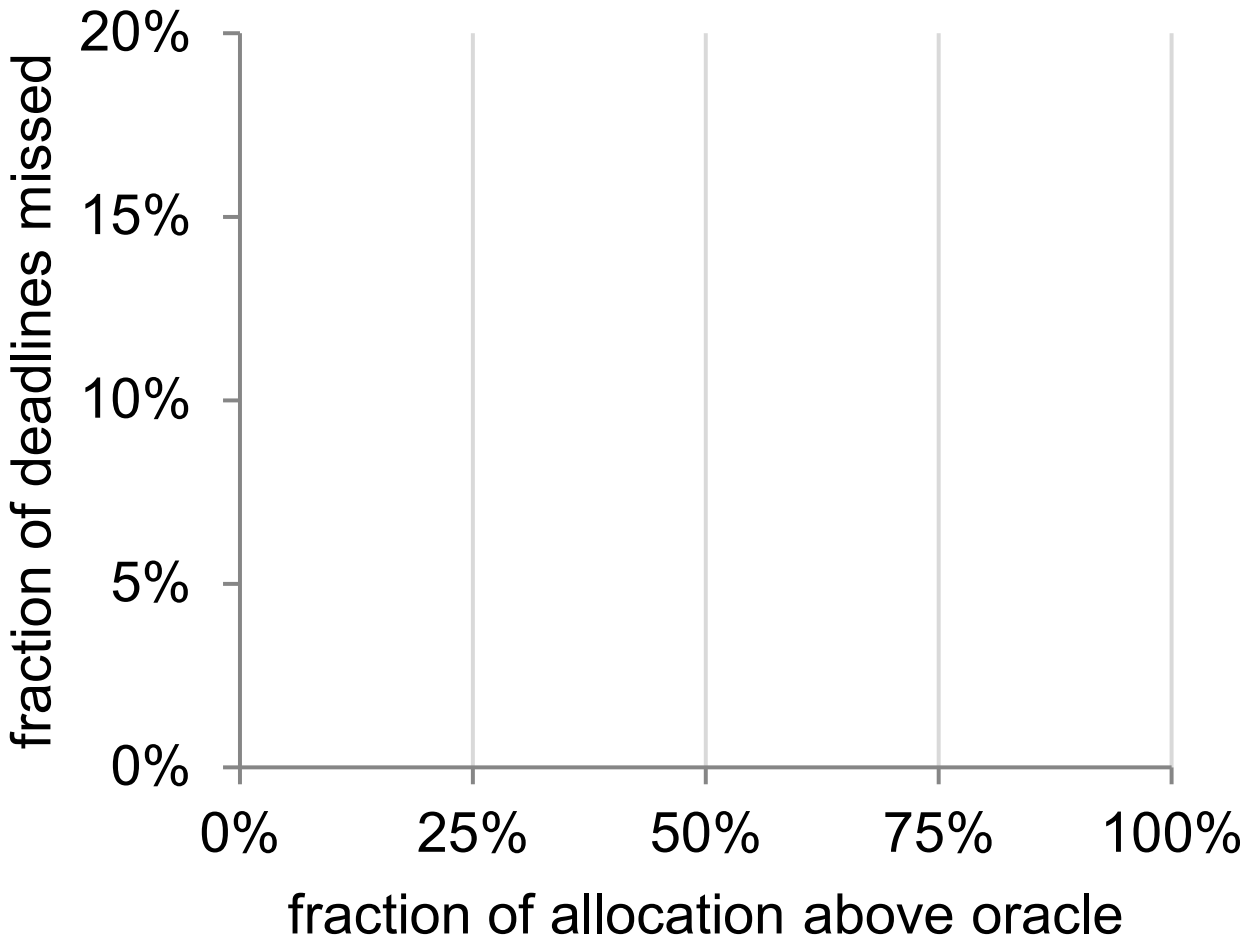

Missed 1 of 94 deadlines
Simulator made good predictions: 80% finish before deadline
100%
80%
60%
40%
20%
0%
CDF
max allocation
Jockey
Control loop only
Allocation from simulator
deadline
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
110%
120%
130%
job completion time relative to deadline
Allocated too many resources
Jobs which met the SLO
Control loop is stable and successful

Evaluation

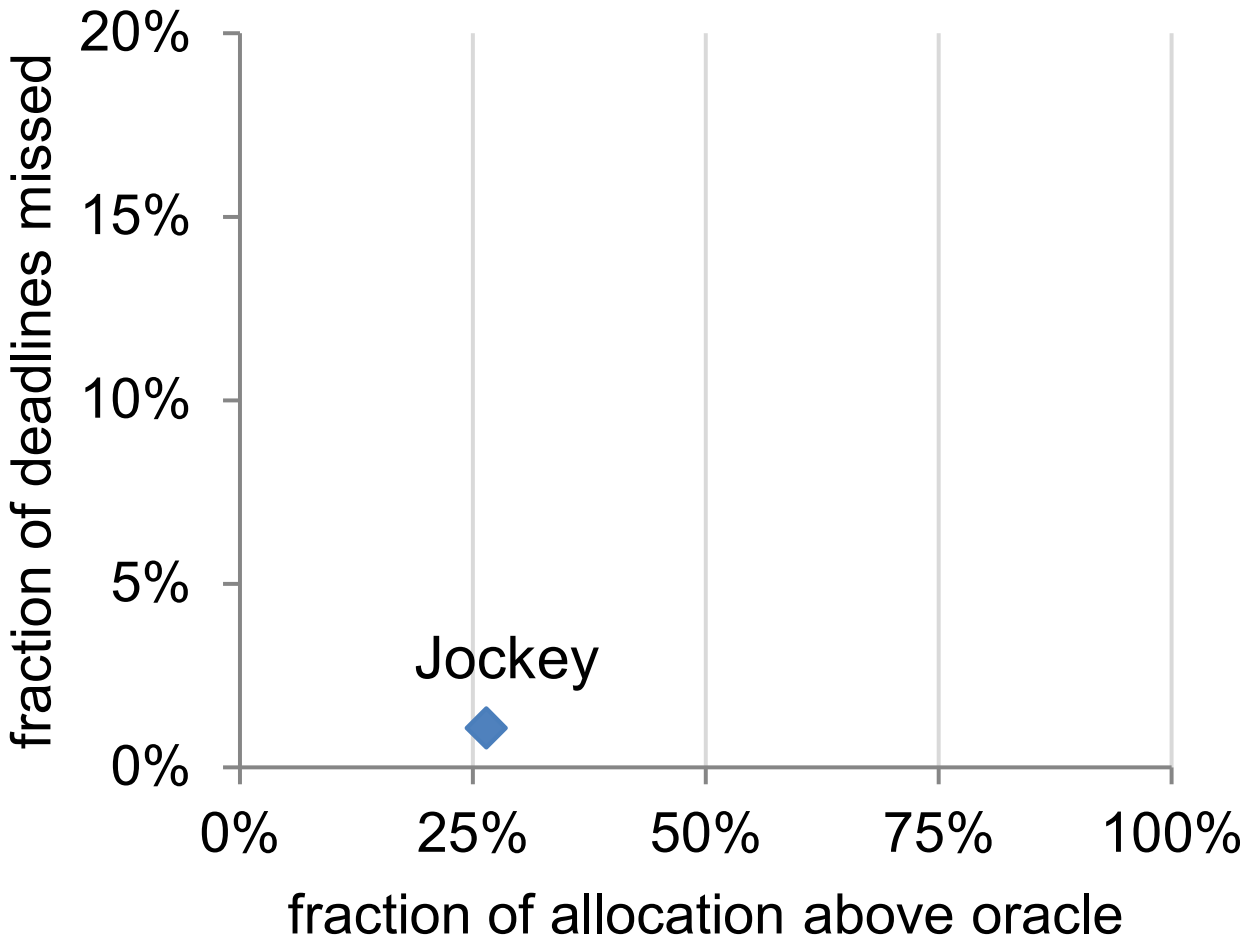
Evaluation



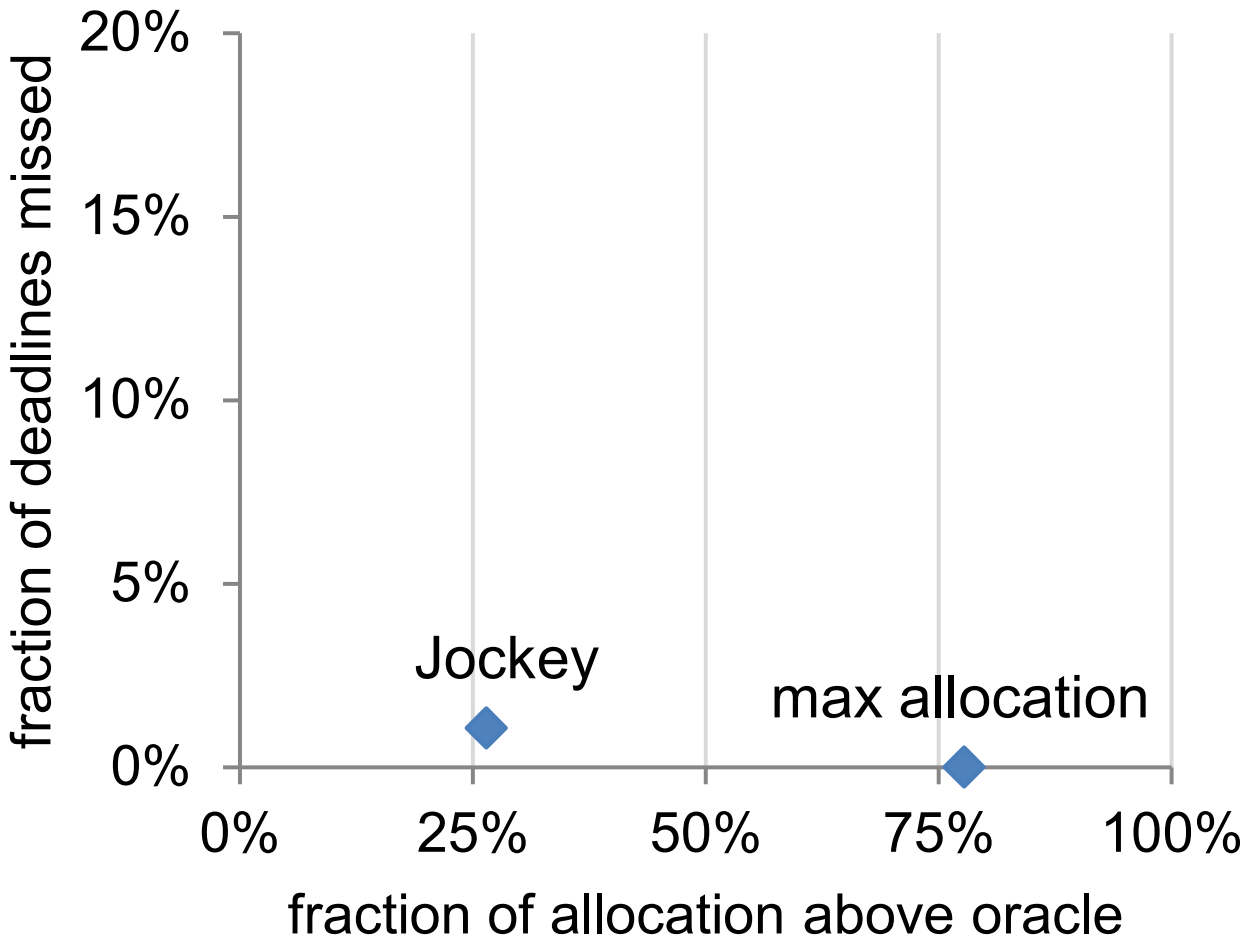
Evaluation


20%
15%
10%
5%
0%
fraction of deadlines missed
0%
25%
50%
75%
100%
fraction of allocation above oracle

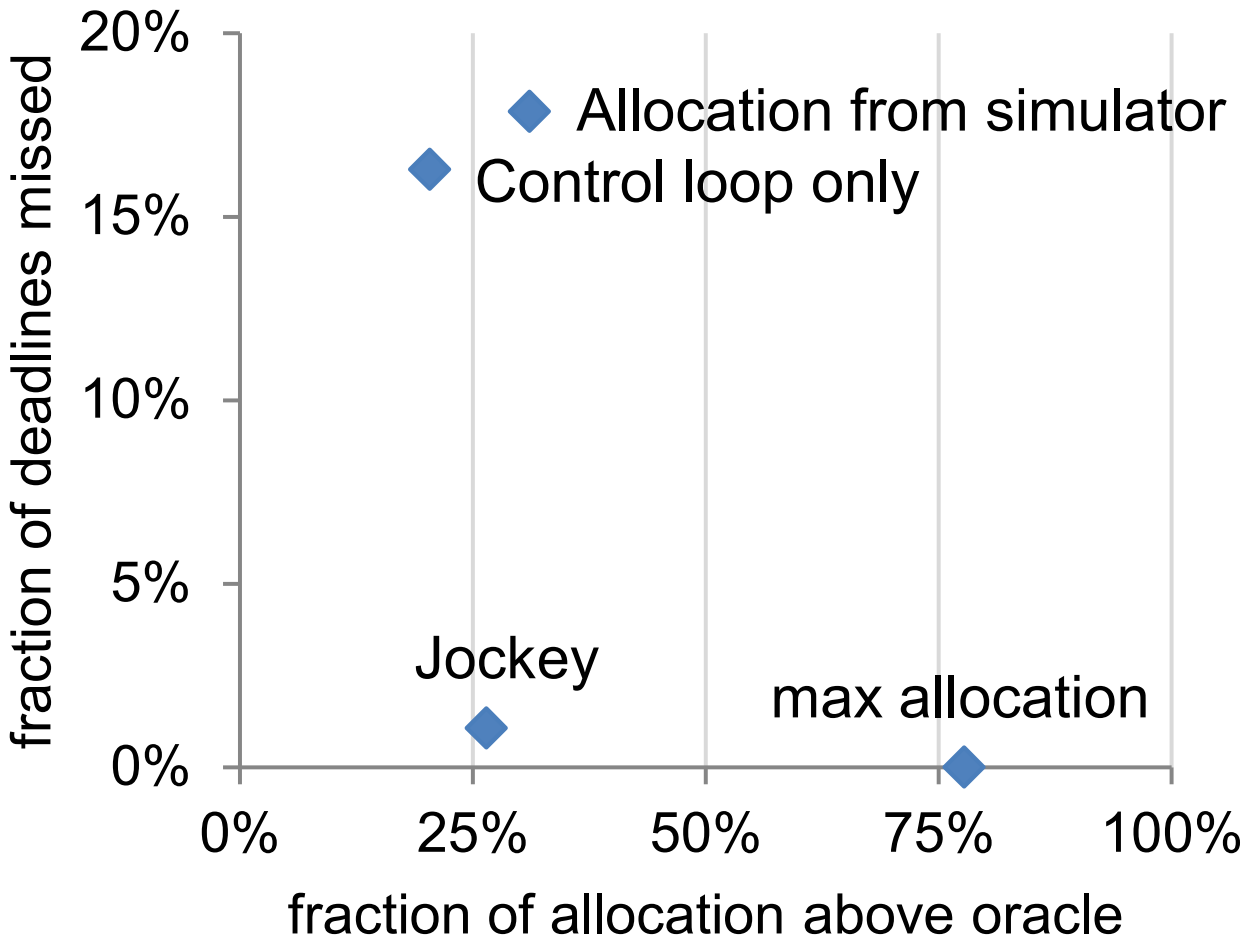
Evaluation


20%
15%
10%
5%
0%
fraction of deadlines missed
Jockey
0%
25%
50%
75%
100%
fraction of allocation above oracle

Evaluation


20%
15%
10%
5%
0%
fraction of deadlines missed
Jockey
max allocation
0%
25%
50%
75%
100%
fraction of allocation above oracle

Evaluation


20%
15%
10%
5%
0%
fraction of deadlines missed
Allocation from simulator
Control loop only
Jockey
max allocation
0%
25%
50%
75%
100%
fraction of allocation above oracle

Conclusion

Data parallel jobs are complex,

Data parallel jobs are complex,

yet users demand deadlines.

Data parallel jobs are complex,

yet users demand deadlines.

Jobs run in shared, noisy clusters,

Data parallel jobs are complex,

yet users demand deadlines.

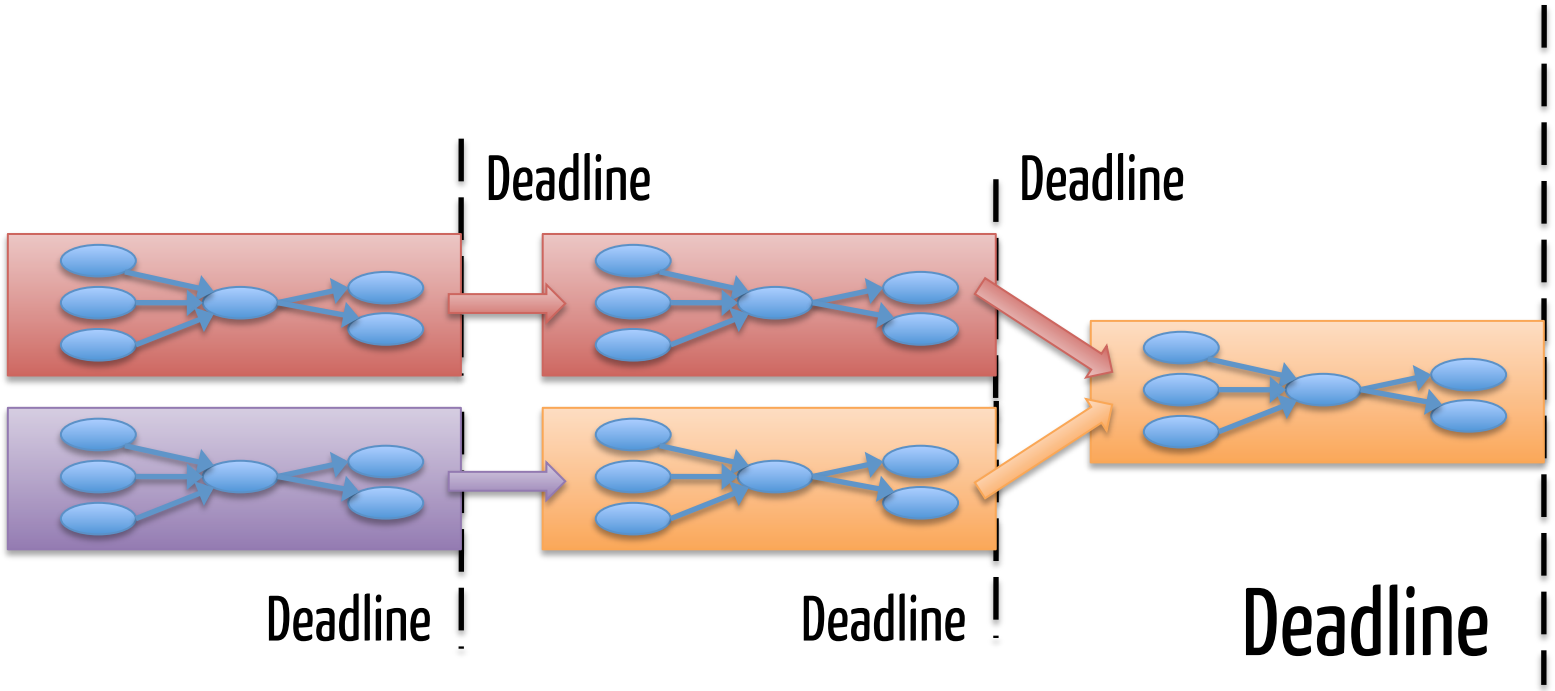
Jobs run in shared, noisy clusters,

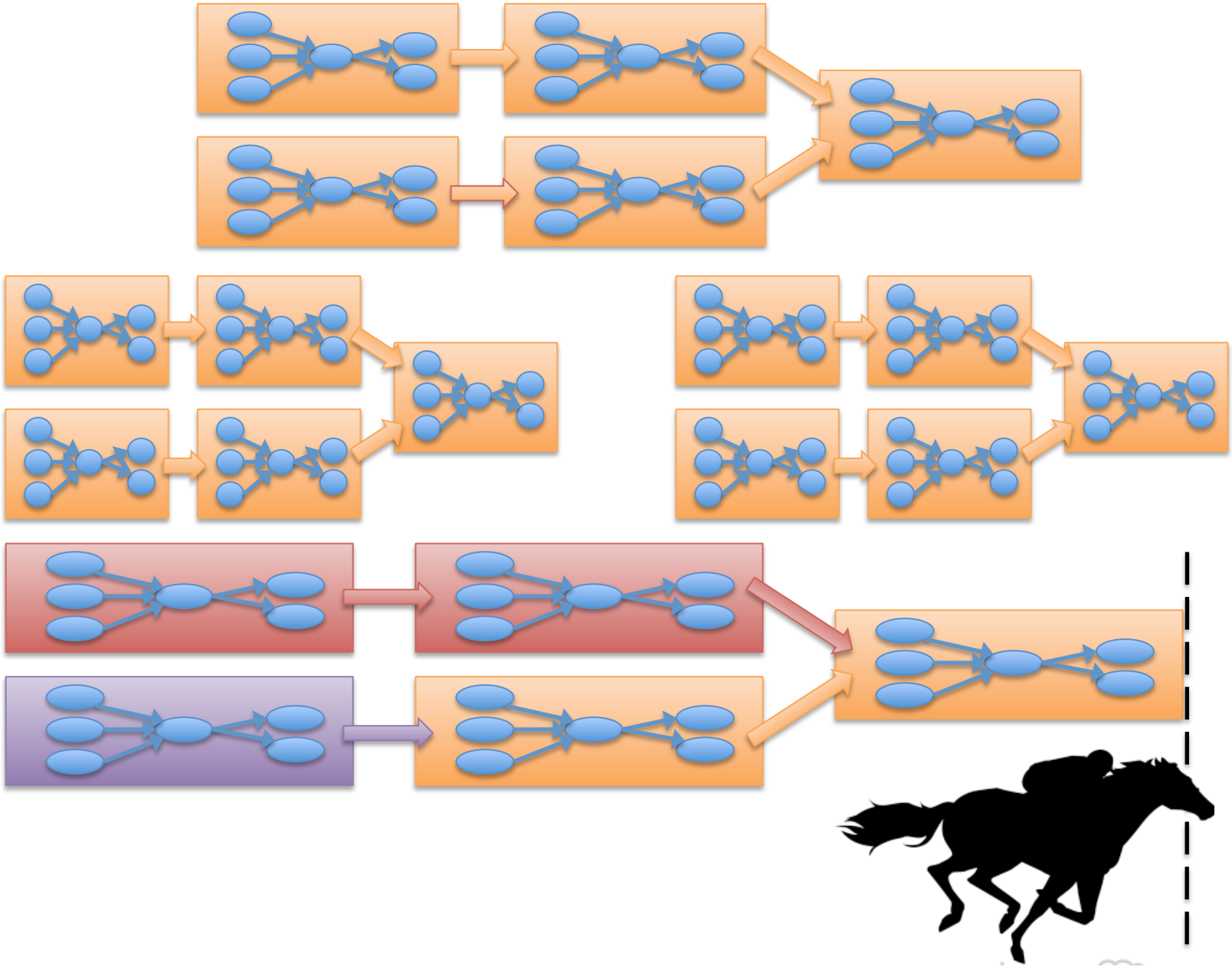
making simple models inaccurate.

Jockey

simulator

control-loop


Deadline
Deadline
Deadline
Deadline
Deadline



Questions?

Andrew Ferguson
adf@cs.brown.edu

Co-authors

- Peter Bodík (Microsoft Research)
- Srikanth Kandula (Microsoft Research)
- Eric Boutín (Microsoft)
- Rodrigo Fonseca (Brown)

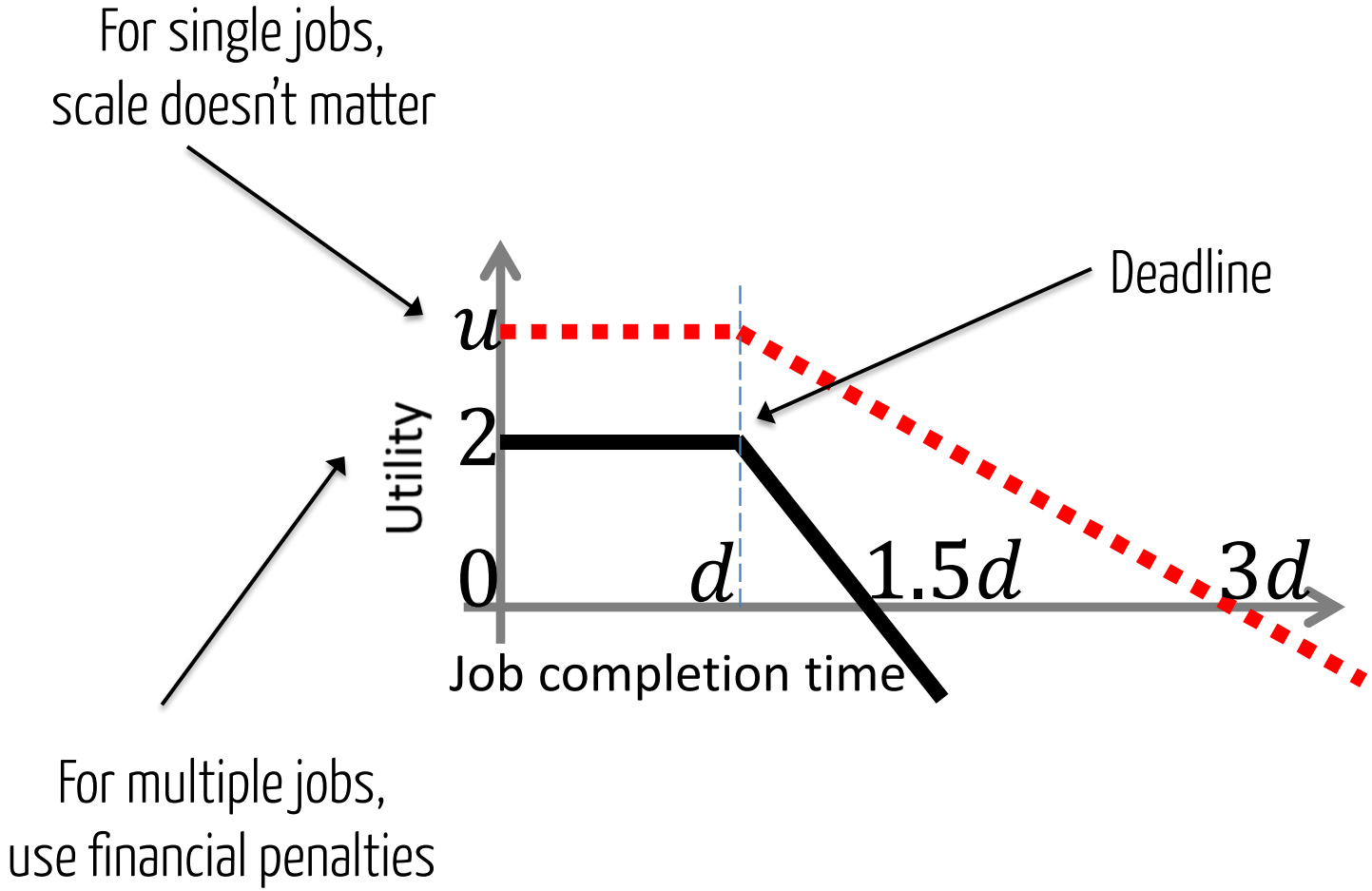



Questions?

Andrew Ferguson
adf@cs.brown.edu

Backup Slides

Utility Curves



Jockey

Resource allocation control loop

Utility

Run Time

Prediction

$$U_a = U(t_r + C(p, a))$$

$$A^r = \arg\min_a \{a : U_a = \max_b U_b\}$$

1. Slack
2. Hysteresis $$A_t^s = A_{t-1}^s + \alpha(A^r - A_{t-1}^s)$$
3. Dead Zone

Cosmos

Resource sharing in Cosmos

- Resources are allocated with a form of fair sharing across business groups and their jobs. (Like Hadoop FairScheduler or CapacityScheduler)
- Each job is guaranteed a number of *tokens* as dictated by cluster policy; each running or initializing task uses one token. Token released on task completion.
- A token is a guaranteed share of CPU and memory
- To increase efficiency, unused tokens are re-allocated to jobs with available work

Jockey

Progress indicator

- Can use many features of the job to build a progress indicator
- Earlier work (ParaTimer) concentrated on fraction of tasks completed
- Our indicator is very simple, but we found it performs best for Jockey's needs

Fraction of completed vertices

Total vertex initialization time

Total vertex run time

$$\sum_{\text{stage } s} f_s \left(Q_s + T_s \right)$$

Comparison with ARIA

- ARIA uses analytic models
- Designed for 3 stages: Map, Shuffle, Reduce
- Jockey's control loop is robust due to control-theory improvements
- ARIA tested on small (66-node) cluster without a network bottleneck
- We believe Jockey is a better match for production DAG frameworks such as Hive, Pig, etc.

Jockey

Latency prediction: C(p, a)

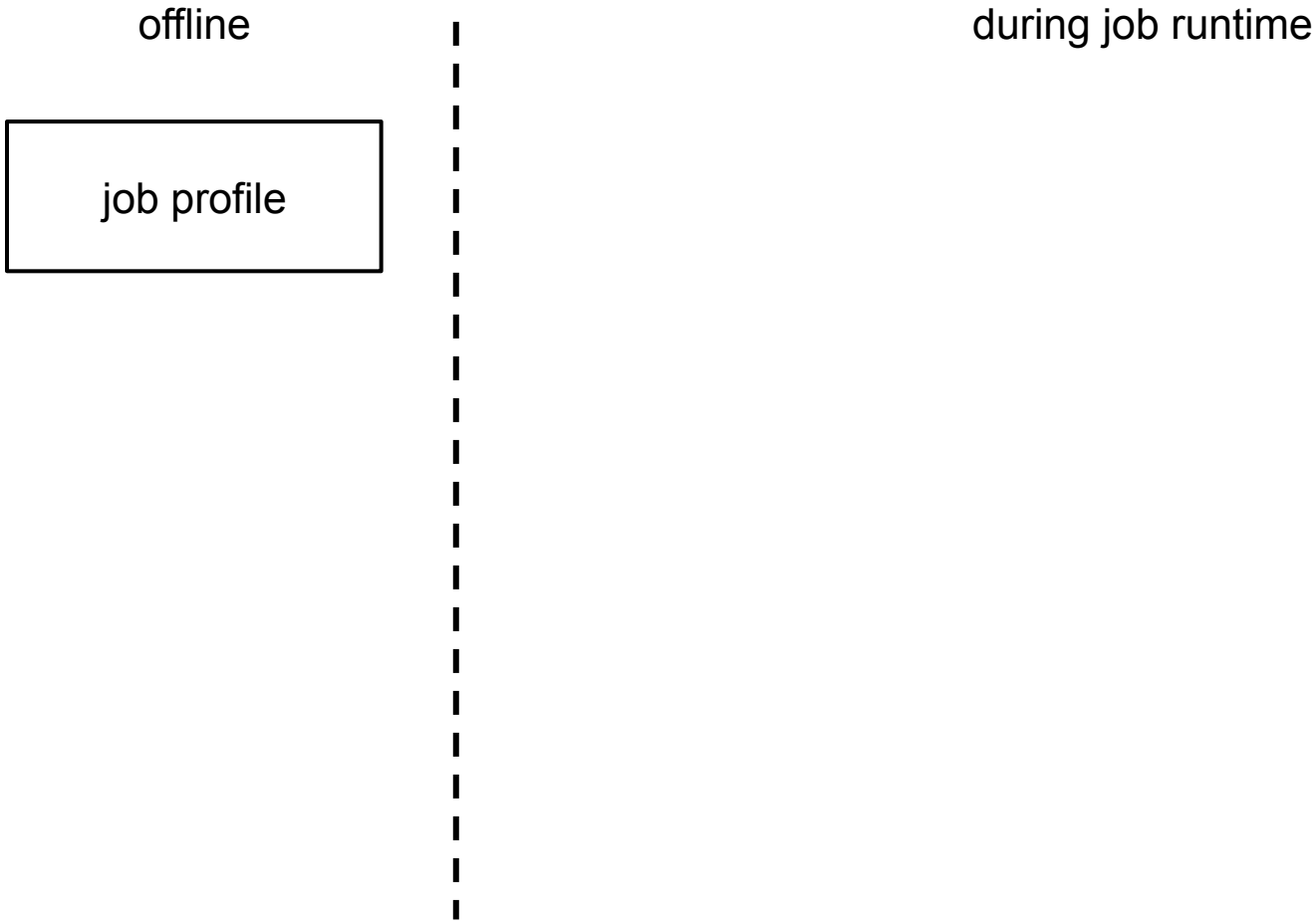
- Event-based simulator
  - Same scheduling logic as actual Job Manager
  - Captures important features of job progress
  - Does not model input size variation or speculative re-execution of stragglers
  - Inputs: job algebra, distributions of task timings, probabilities of failures, allocation
- Analytic model
  - Inspired by Amdahl's Law: T = S + P/N
  - S is remaining work on critical path, P is all remaining work, N is number of machines

Jockey

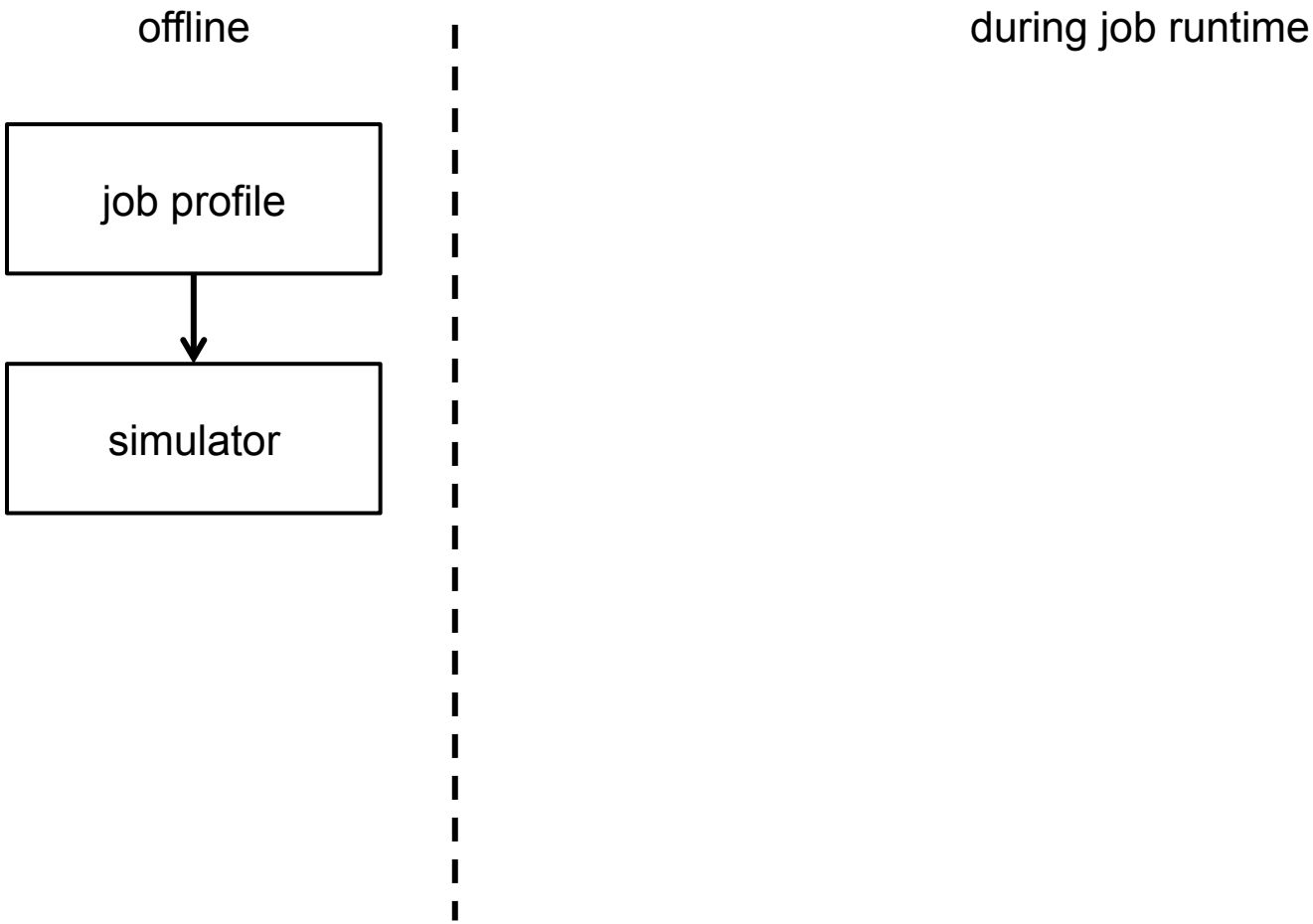
Resource allocation control loop

- Executes in Dryad's Job Manager
- Inputs: fraction of completed tasks in each stage, time job has spent running, utility function, precomputed values (for speedup)
- Output: Number of tokens to allocate
- Improved with techniques from control-theory

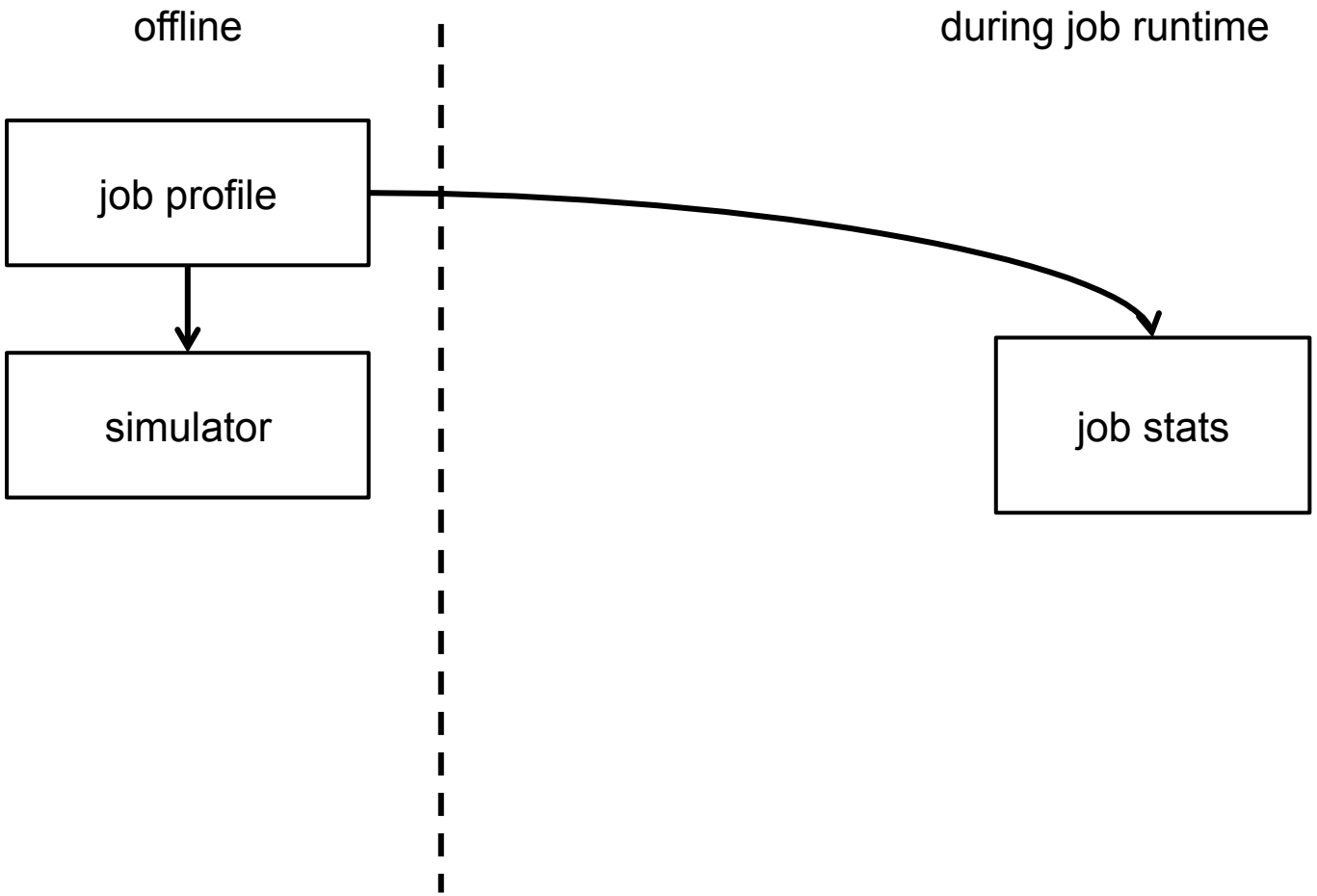
Jockey



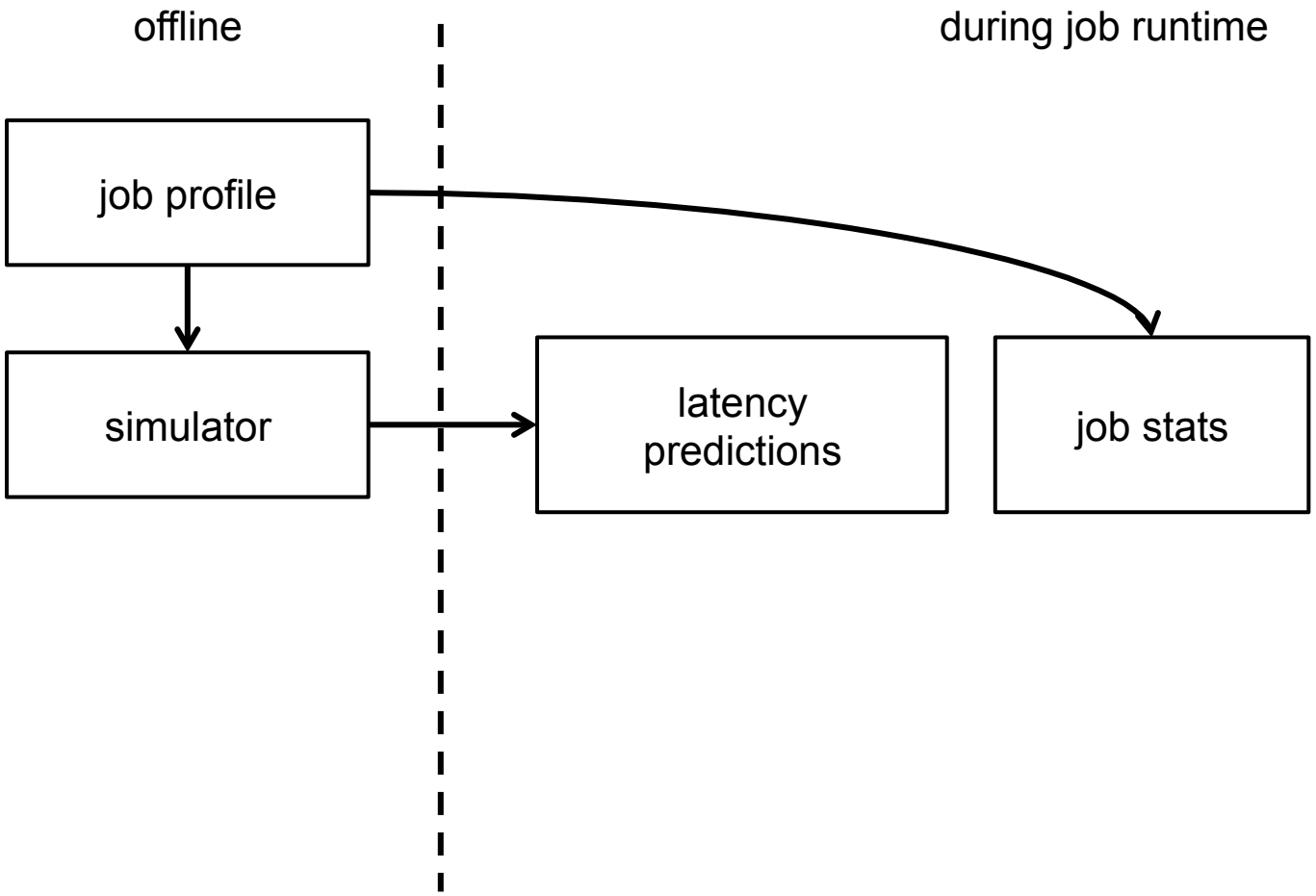
Jockey



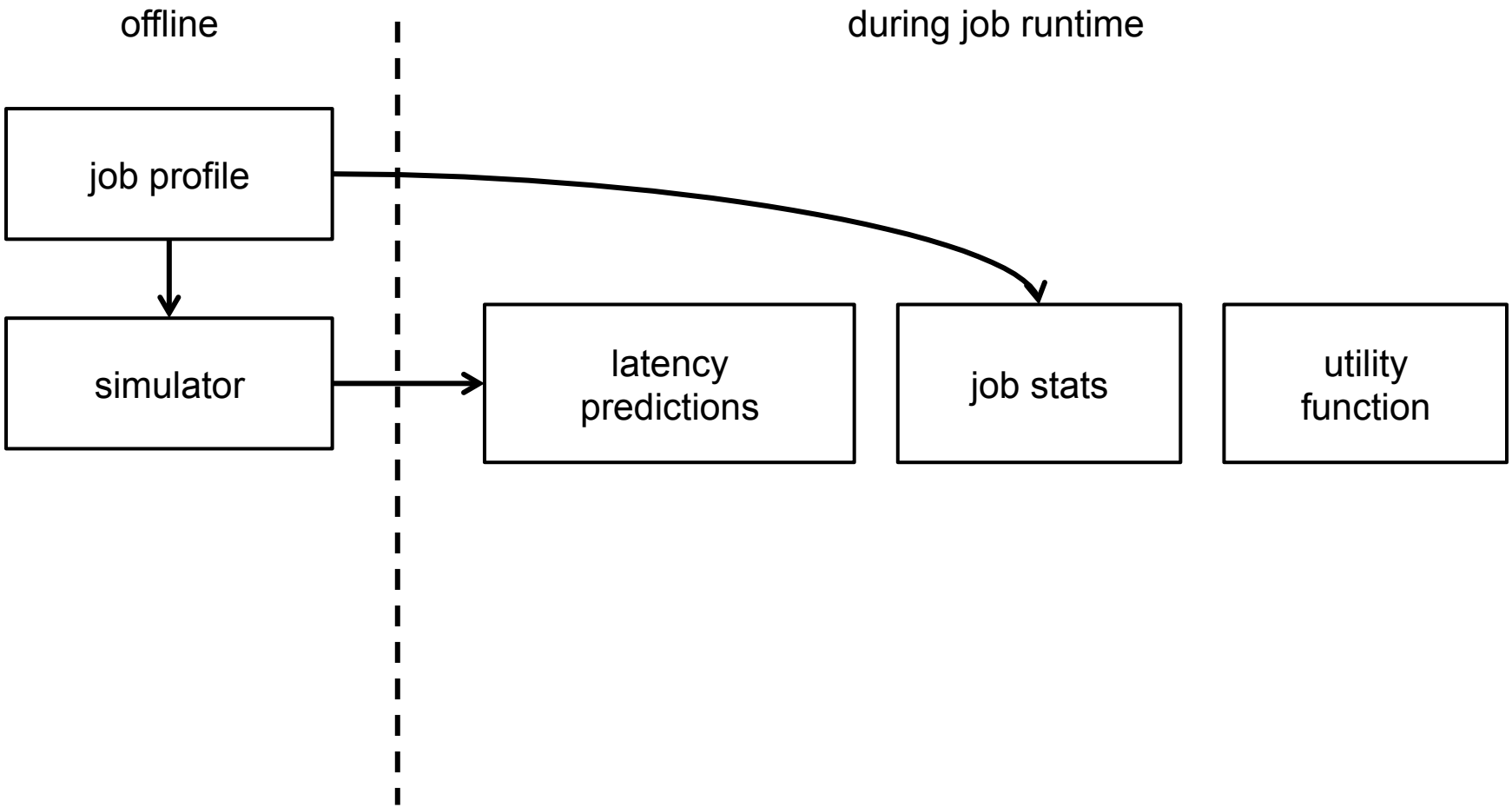
Jockey


offline
during job runtime
job profile
simulator
job stats

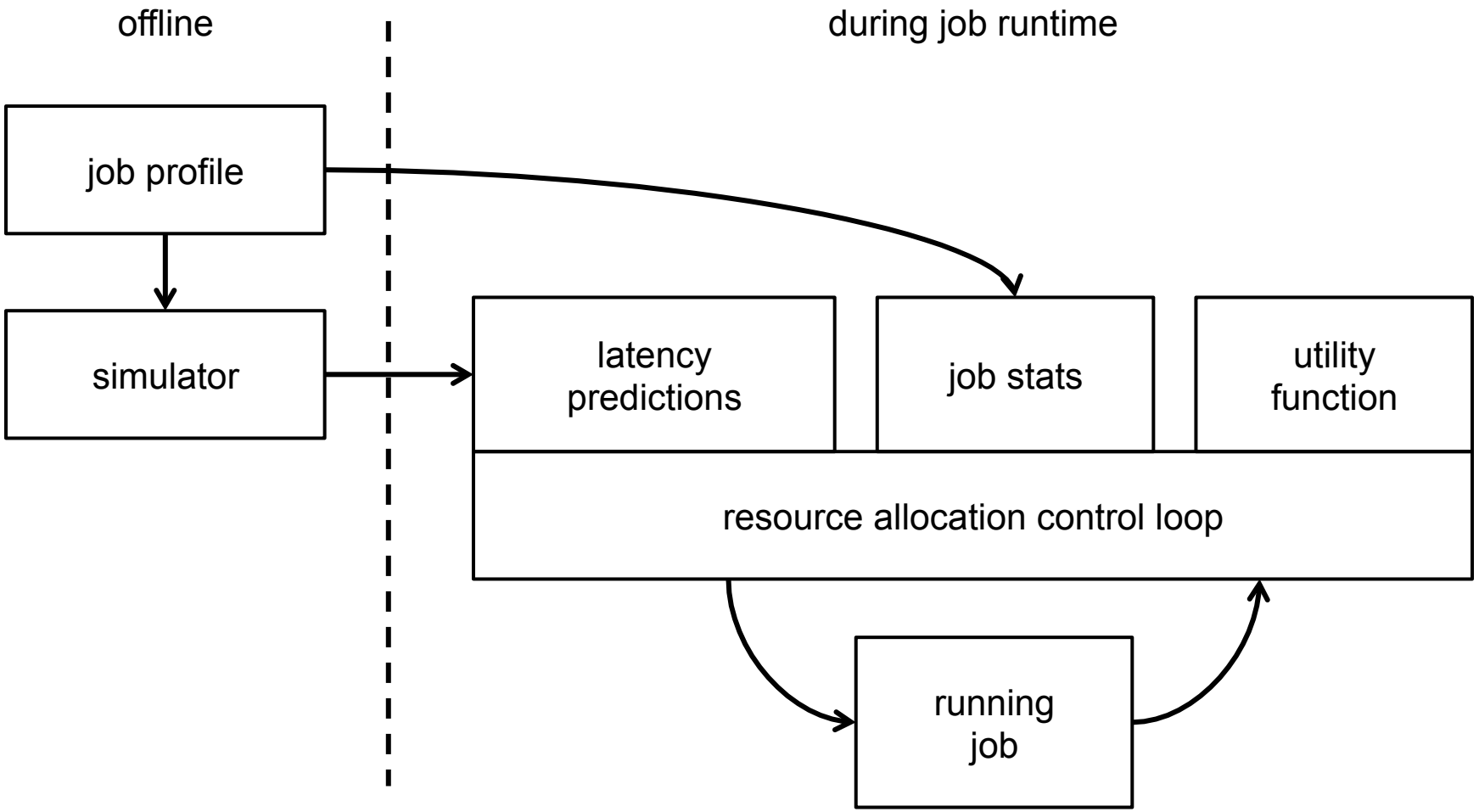
Jockey


offline
during job runtime
job profile
simulator
latency predictions
job stats

Jockey


offline
during job runtime
job profile
simulator
latency predictions
job stats
utility function

Jockey


offline
during job runtime
job profile
simulator
latency predictions
job stats
utility function
resource allocation control loop
running job