Systems@ETH zürich

OSDI | 18

Three steps is all you need

fast, accurate, automatic scaling decisions for distributed streaming dataflows

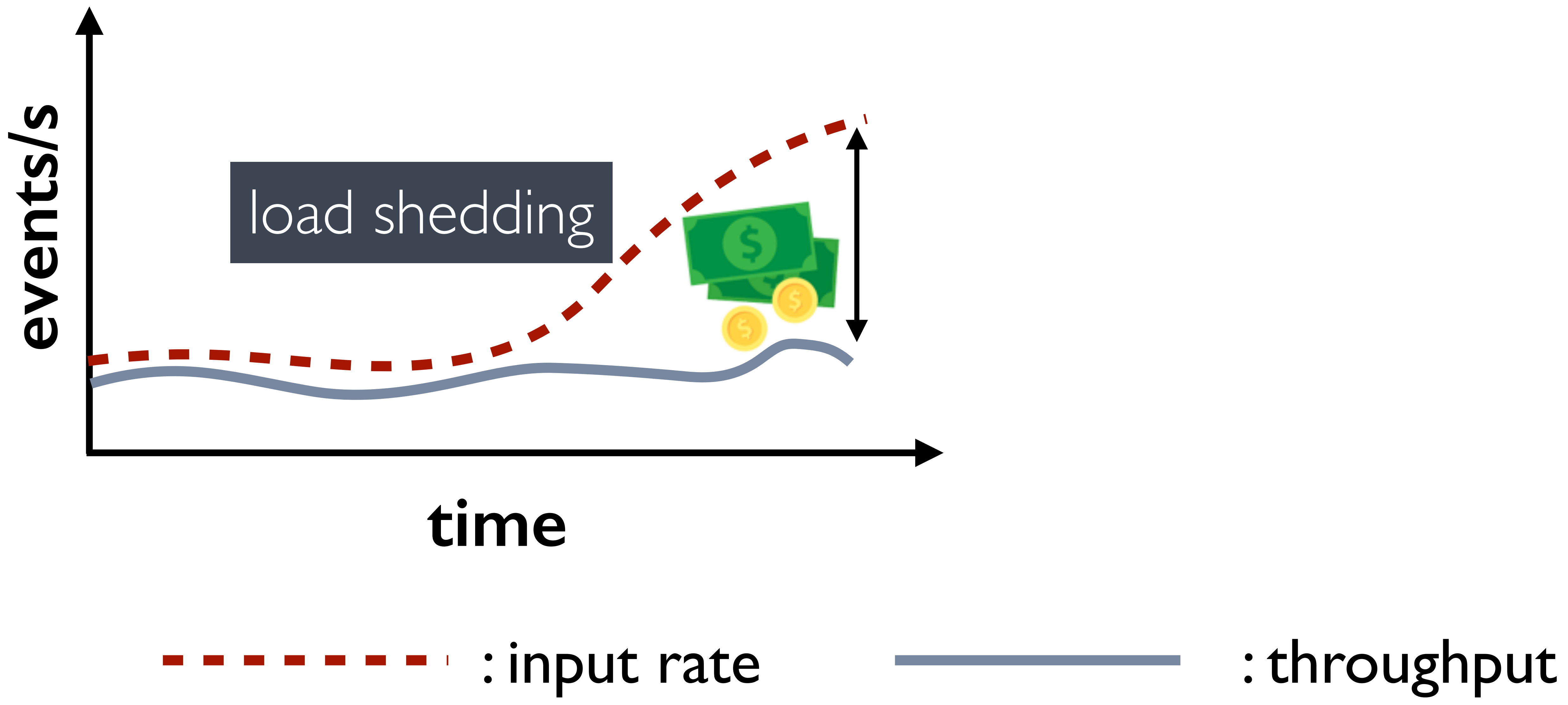
Vasiliki Kalavri†, John Liagouris†, Moritz Hoffmann†, Desislava Dimitrova†, Matthew Forshaw††, Timothy Roscoe†

†Systems Group, Department of Computer Science, ETH Zürich, firstname.lastname@inf.ethz.ch
††Newcastle University, firstname.lastname@newcastle.ac.uk

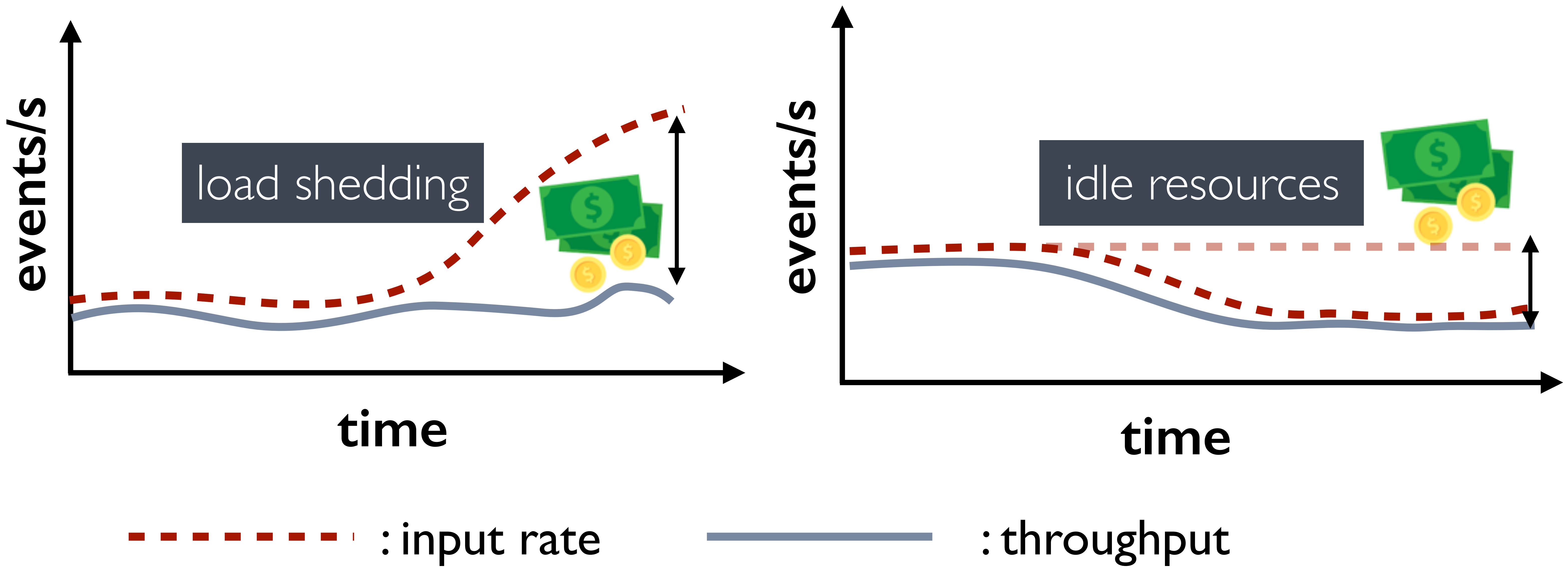
Support: FNSNF Fonds national suisse, Schweizerischer Nationalfonds, Fondo nazionale svizzero, Swiss National Science Foundation; Google; vmware Research

Any streaming job will inevitably become over- or under-provisioned in the future

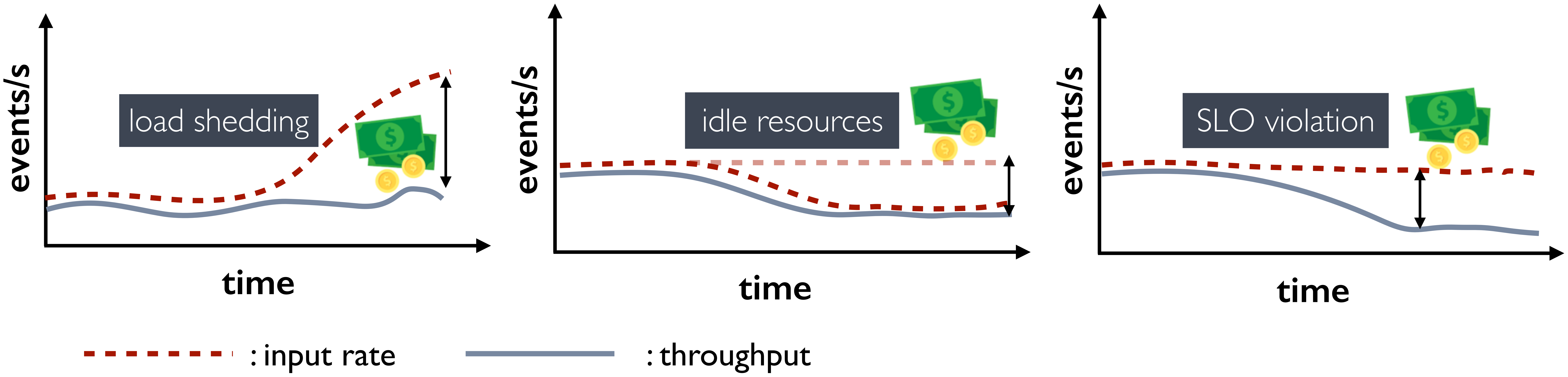
Any streaming job will inevitably become over- or under-provisioned in the future



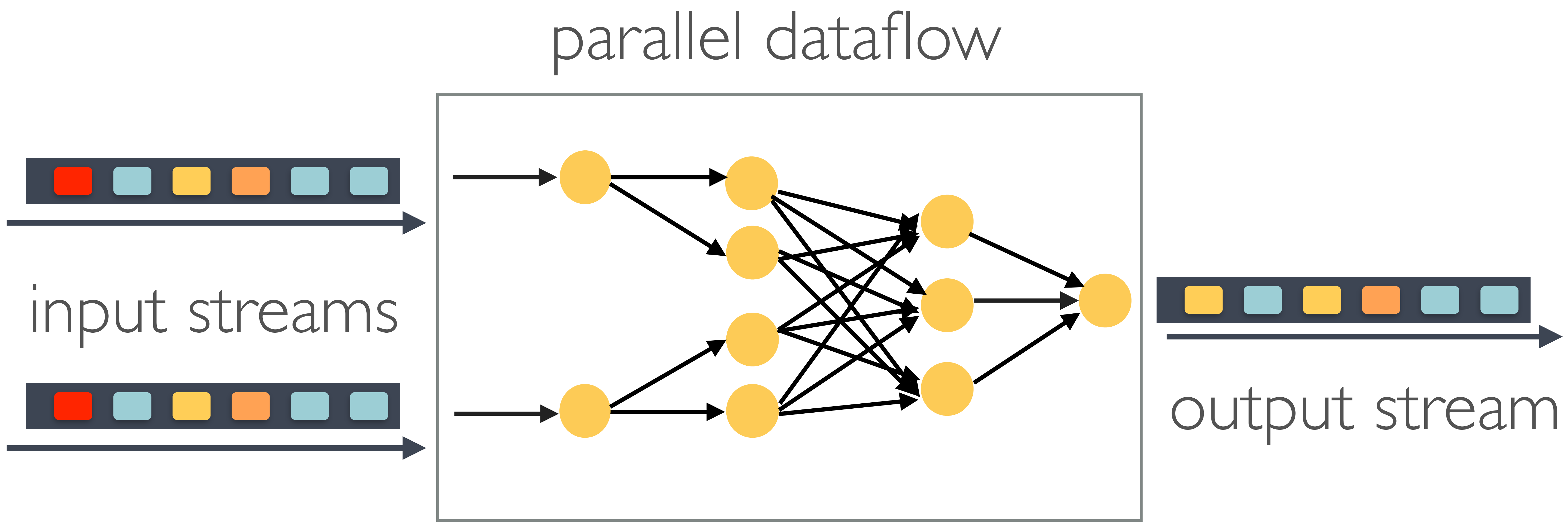
Any streaming job will inevitably become over- or under-provisioned in the future



Any streaming job will inevitably become over- or under-provisioned in the future

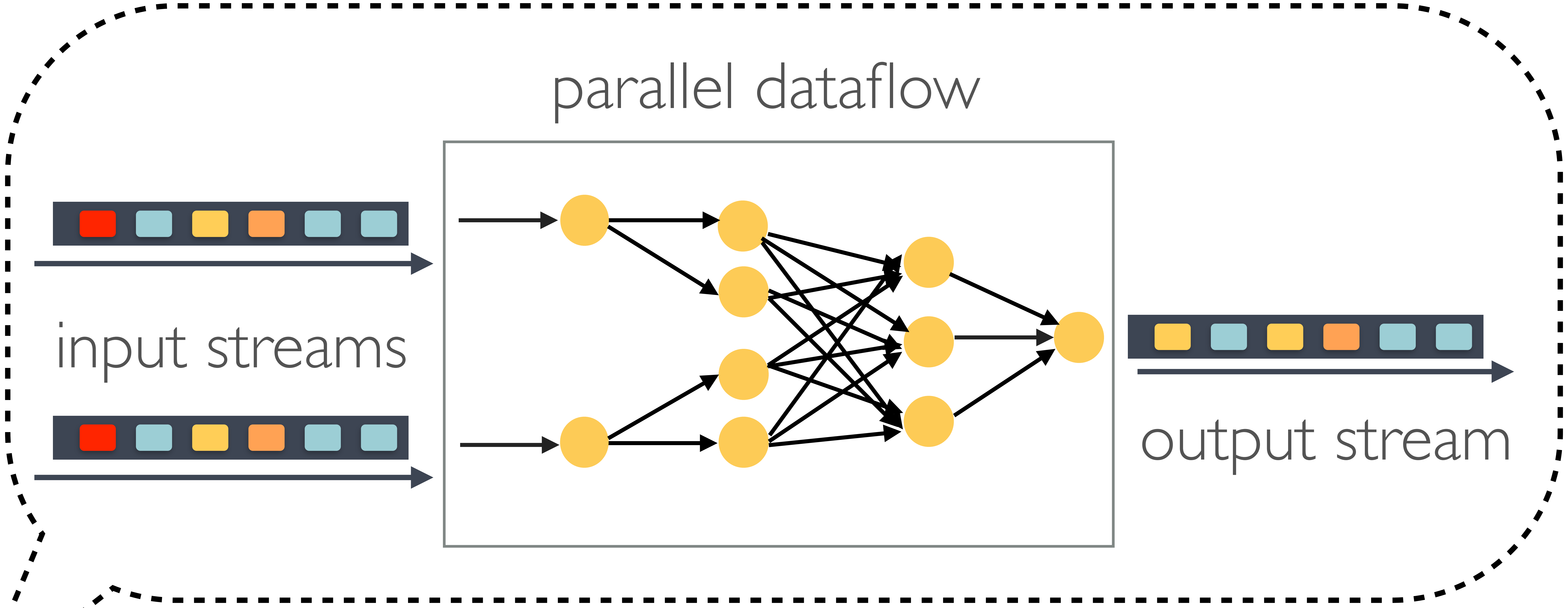


CONFIGURING PARALLELISM FOR A STREAMING JOB



CONFIGURING PARALLELISM FOR A STREAMING JOB

1. monitor event rates

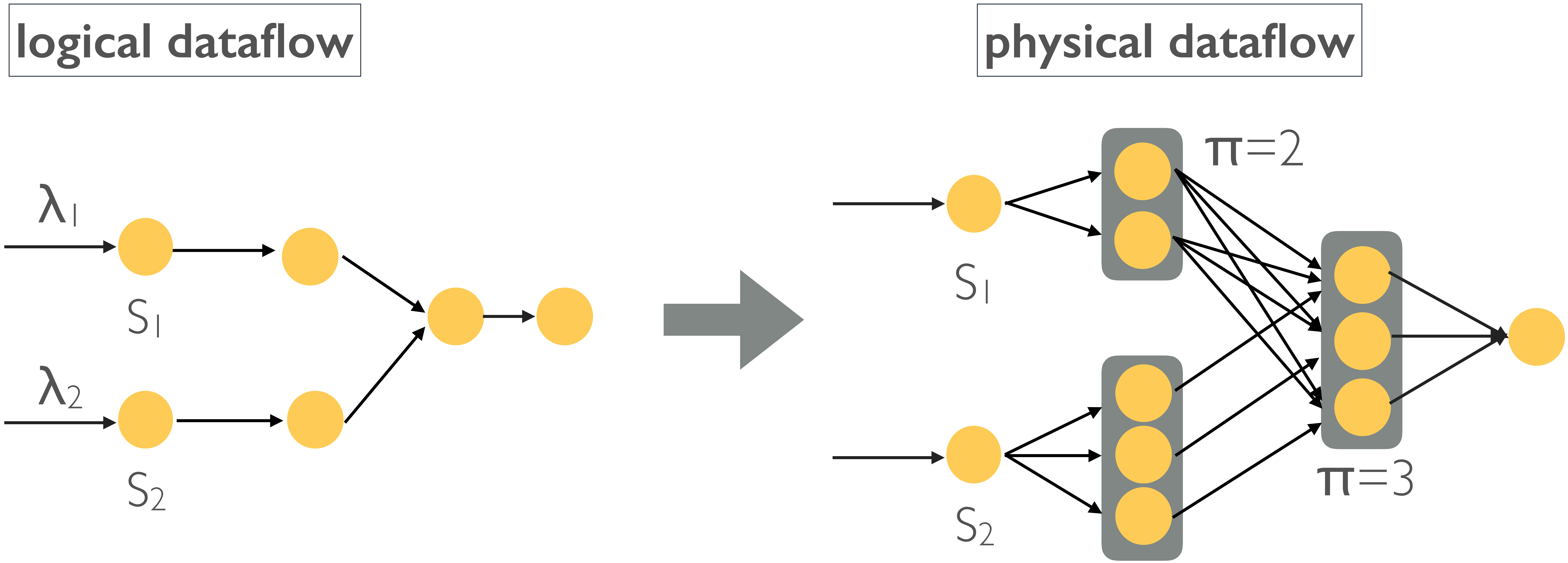


2. configure parallelism

3. deploy and test performance

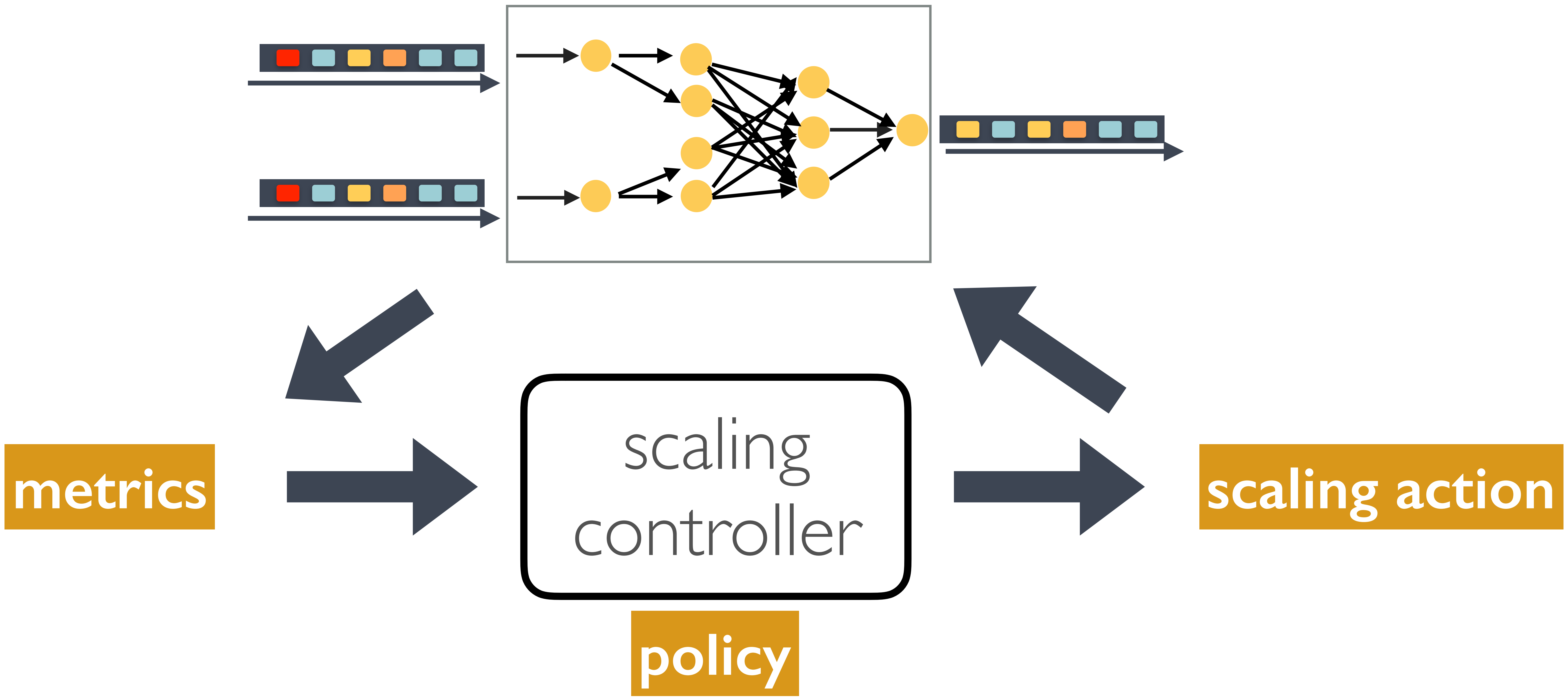
until the target throughput is met

THE SCALING PROBLEM

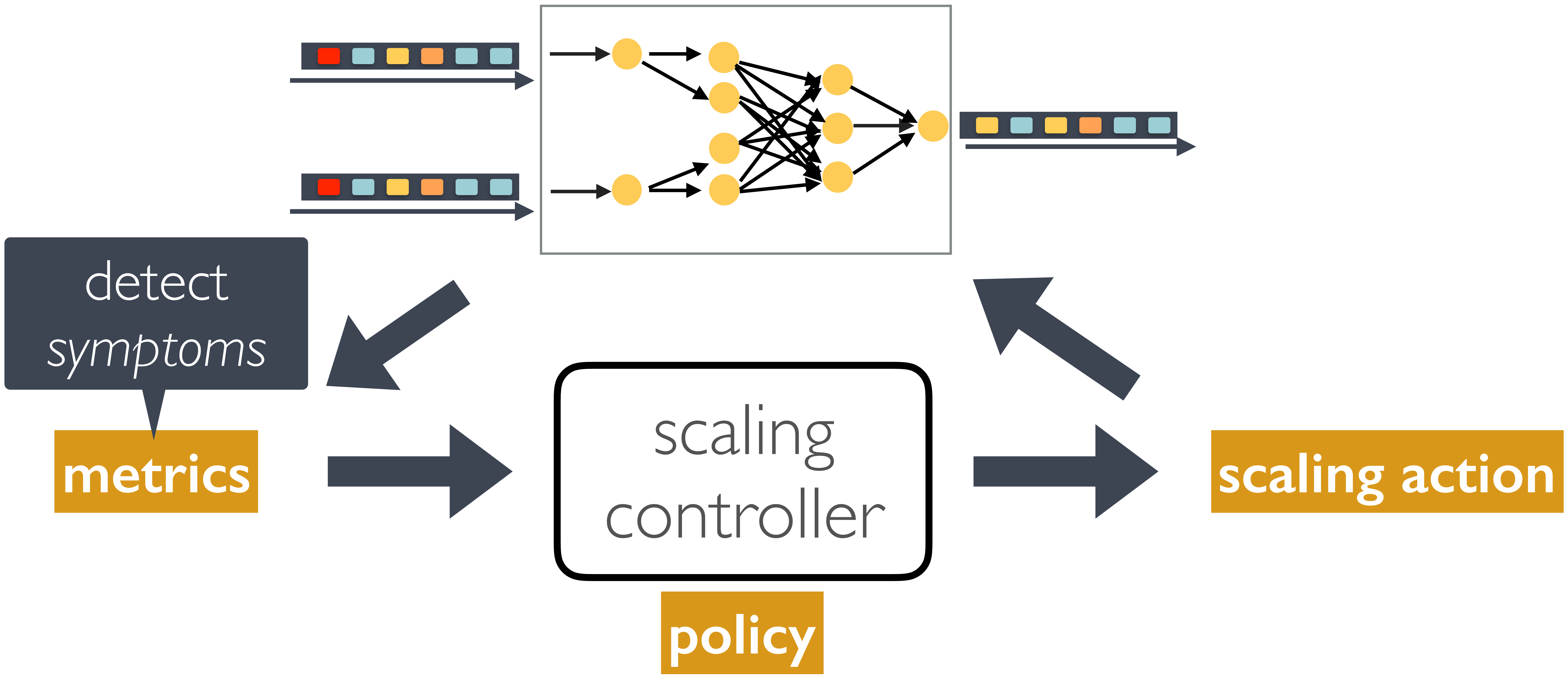


Given a logical dataflow with sources $S_1, S_2, \ldots S_n$ and rates $\lambda_1, \lambda_2, \ldots \lambda_n$ identify the minimum parallelism π_i per operator i, such that the physical dataflow can sustain all source rates.

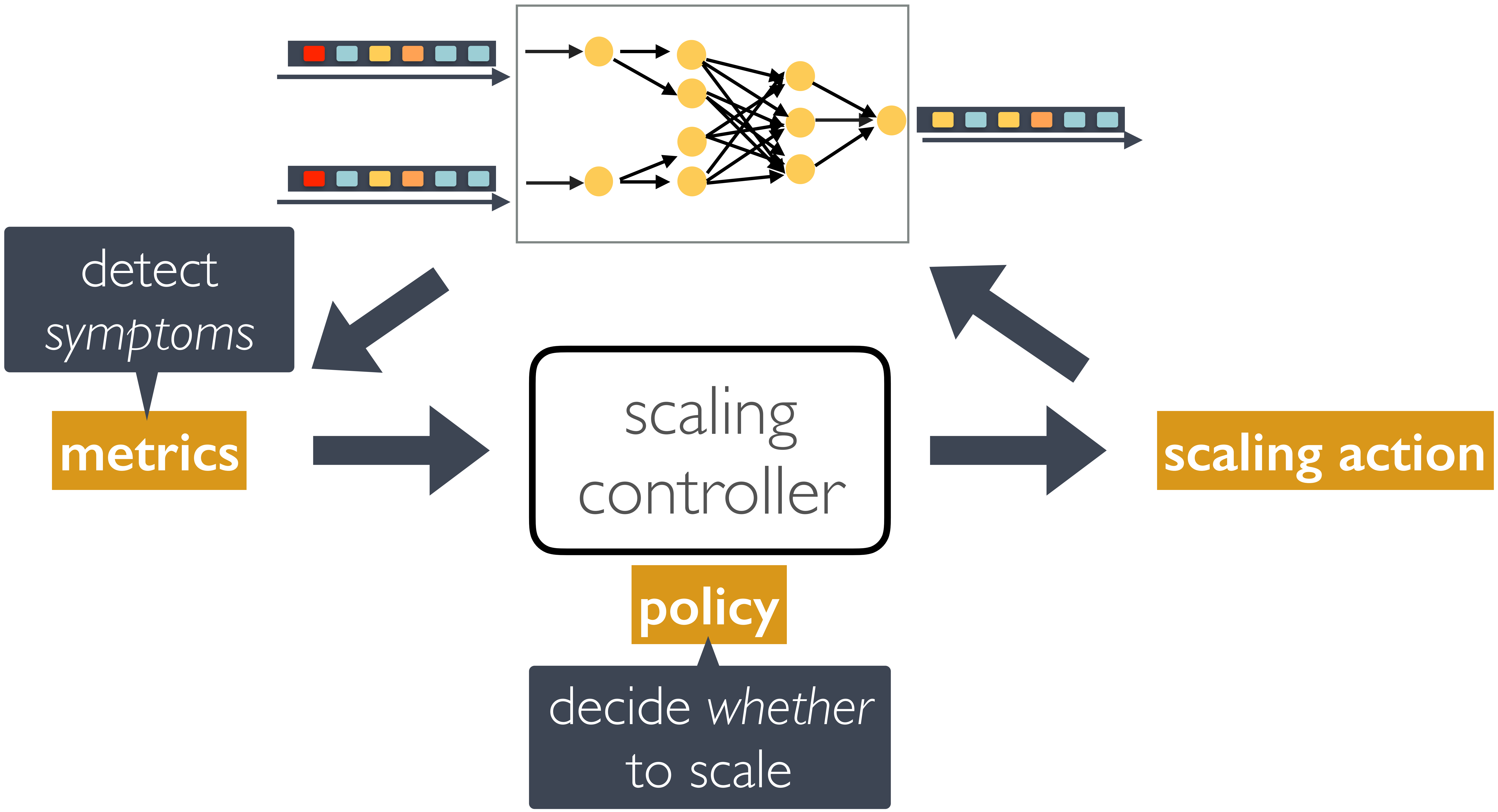
AUTOMATIC SCALING OVERVIEW



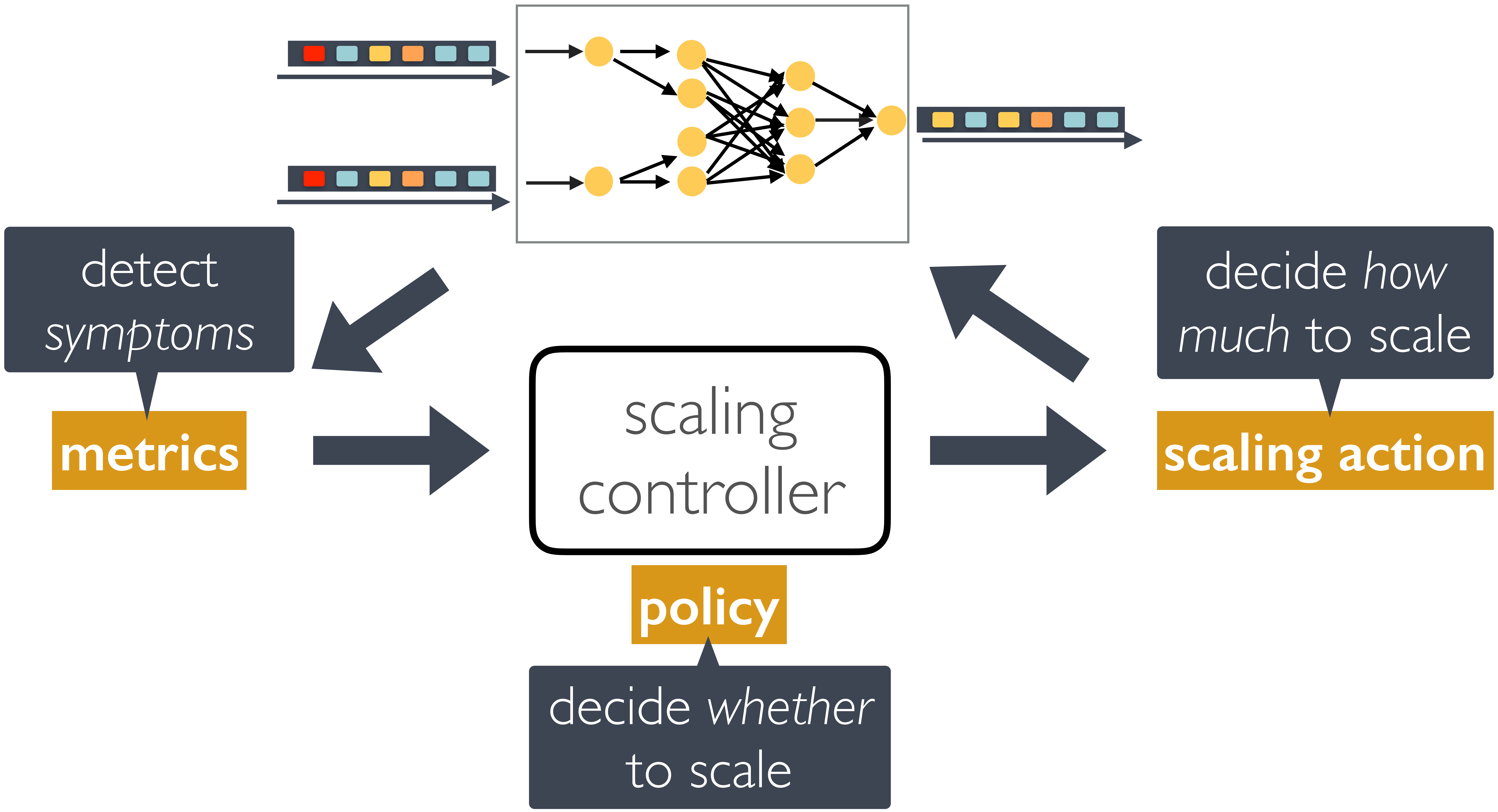
AUTOMATIC SCALING OVERVIEW



AUTOMATIC SCALING OVERVIEW



AUTOMATIC SCALING OVERVIEW



Existing approaches

- Borealis
- StreamCloud
- Seep
- IBM Streams
- Spark Streaming
- Google Dataflow
- Dhalion
- …

metrics | **policy** | **scaling action**

Existing approaches

- Borealis
- StreamCloud
- Seep
- IBM Streams
- Spark Streaming
- Google Dataflow
- Dhalion
- …

metrics

CPU utilization
backlog, tuples/s
backpressure signal

policy

scaling action

Existing approaches

- Borealis
- StreamCloud
- Seep
- IBM Streams
- Spark Streaming
- Google Dataflow
- Dhalion
- ...

metrics

CPU utilization
backlog, tuples/s
backpressure signal

policy

threshold and rule-based
if CPU > 80% => scale

scaling action

Existing approaches

Borealis

StreamCloud

Seep

IBM Streams

Spark Streaming

Google Dataflow

Dhalion

…

metrics

CPU utilization
backlog, tuples/s
backpressure signal

policy

threshold and rule-based
if CPU > 80% => scale

scaling action

small changes,
one operator at a time

Existing approaches

Borealis
StreamCloud
Seep
IBM Streams
Spark Streaming
Google Dataflow
Dhalion
...

metrics

CPU utilization
backlog, tuples/s
backpressure signal

problematic due to interference, multitenancy

policy

threshold and rule-based
if CPU > 80% => scale

scaling action

small changes,
one operator at a time

Existing approaches

Borealis

StreamCloud

Seep

IBM Streams

Spark Streaming

Google Dataflow

Dhalion

...

metrics

CPU utilization
backlog, tuples/s
backpressure signal

problematic due to interference, multitenancy

policy

threshold and rule-based
if CPU > 80% => scale

sensitive to noise, manual, hard to tune

scaling action

small changes,
one operator at a time

Existing approaches

- Borealis
- StreamCloud
- Seep
- IBM Streams
- Spark Streaming
- Google Dataflow
- Dhalion
- ...

metrics

CPU utilization
backlog, tuples/s
backpressure signal

problematic due to interference, multitenancy

policy

threshold and rule-based
if CPU > 80% => scale

sensitive to noise, manual, hard to tune

scaling action

small changes,
one operator at a time

non-predictive, speculative steps

Existing approaches

Borealis

StreamCloud

Seep

IBM Streams

Spark Streaming

Google Dataflow

Dhalion

…

metrics

CPU utilization
backlog, tuples/s
backpressure signal

problematic due to interference, multitenancy

policy

threshold and rule-based
if CPU > 80% => scale

sensitive to noise, manual, hard to tune

scaling action

small changes,
one operator at a time

non-predictive, speculative steps

✗ **oscillations**

Existing approaches

- Borealis
- StreamCloud
- Seep
- IBM Streams
- Spark Streaming
- Google Dataflow
- Dhalion
- …

metrics

CPU utilization
backlog, tuples/s
backpressure signal

problematic due to interference, multitenancy

policy

threshold and rule-based
if CPU > 80% => scale

sensitive to noise, manual, hard to tune

scaling action

small changes,
one operator at a time

non-predictive, speculative steps

✗ **oscillations**

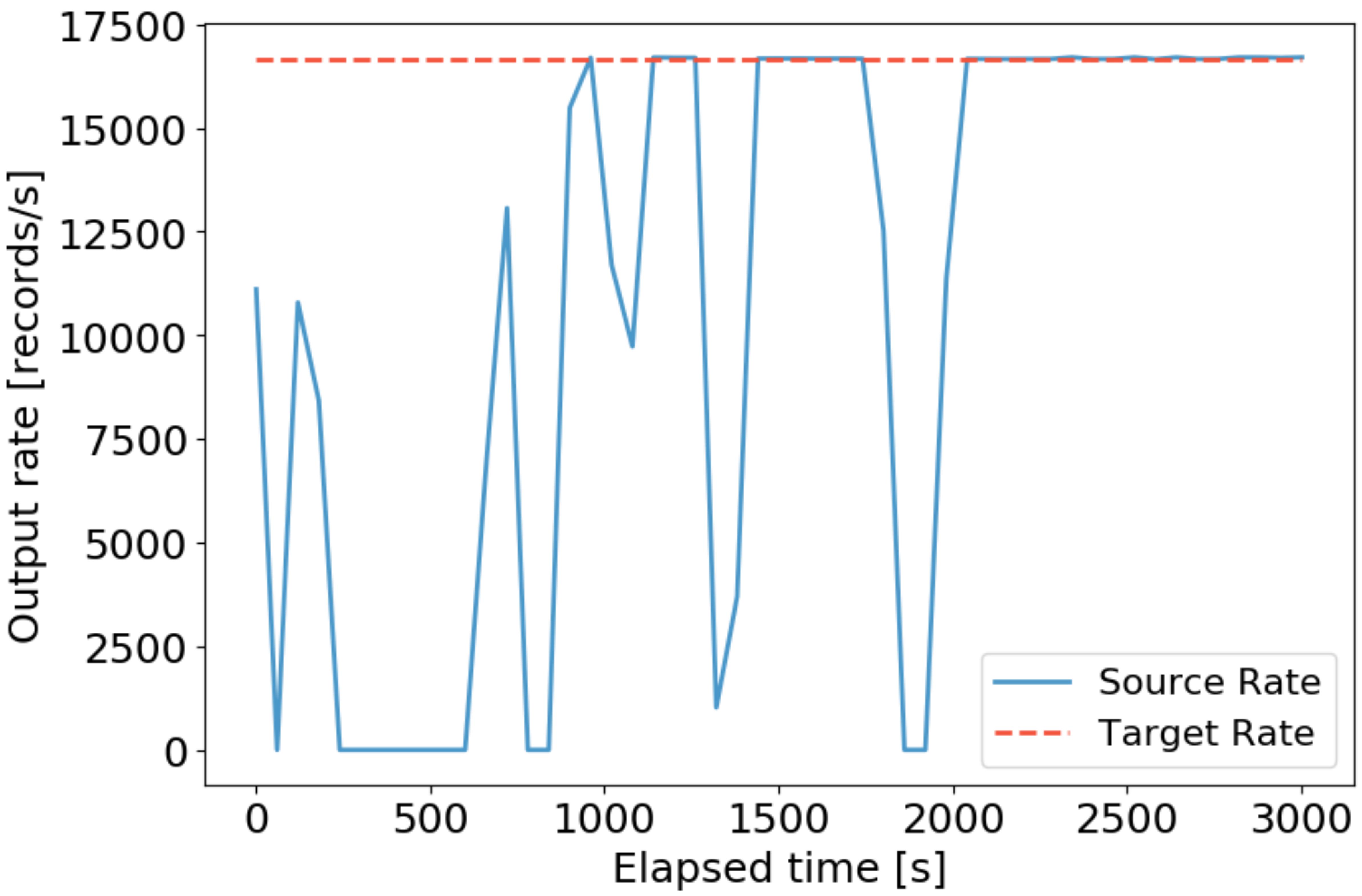
✗ **temporary over- and under-provisioning**

Existing approaches

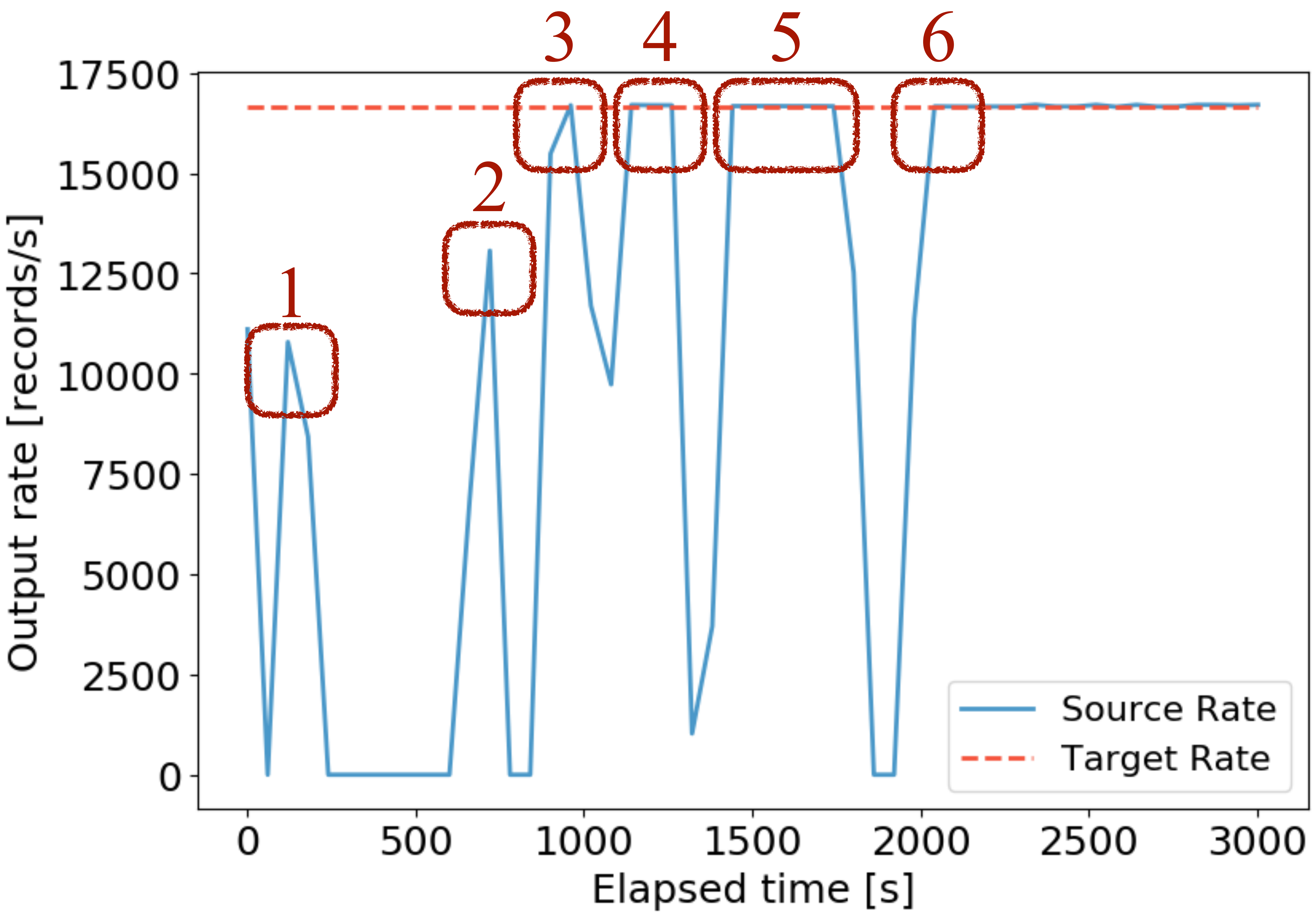
- Borealis
- StreamCloud
- Seep
- IBM Streams
- Spark Streaming
- Google Dataflow
- Dhalion
- …

metrics	**policy**	**scaling action**
CPU utilization backlog, tuples/s backpressure signal	threshold and rule-based *if CPU > 80% => scale*	small changes, one operator at a time
problematic due to interference, multitenancy	sensitive to noise, manual, hard to tune	non-predictive, speculative steps
✗ **oscillations**	✗ **temporary over- and under-provisioning**	✗ **slow convergence**

effect of Dhalion's scaling actions in an initially under-provisioned wordcount dataflow



effect of Dhalion's scaling actions in an initially under-provisioned wordcount dataflow



metrics

externally observed

policy

threshold-based

scaling action

non-predictive, single-operator

OUR APPROACH: DS2

metrics

~~externally observed~~

policy

~~threshold-based~~

scaling action

~~non-predictive, single-operator~~

true rates through instrumentation

dataflow dependency model

predictive, dataflow-wide actions

OUR APPROACH: DS2

metrics

~~externally observed~~

policy

~~threshold-based~~

scaling action

~~non-predictive, single-operator~~

true rates through instrumentation

dataflow dependency model

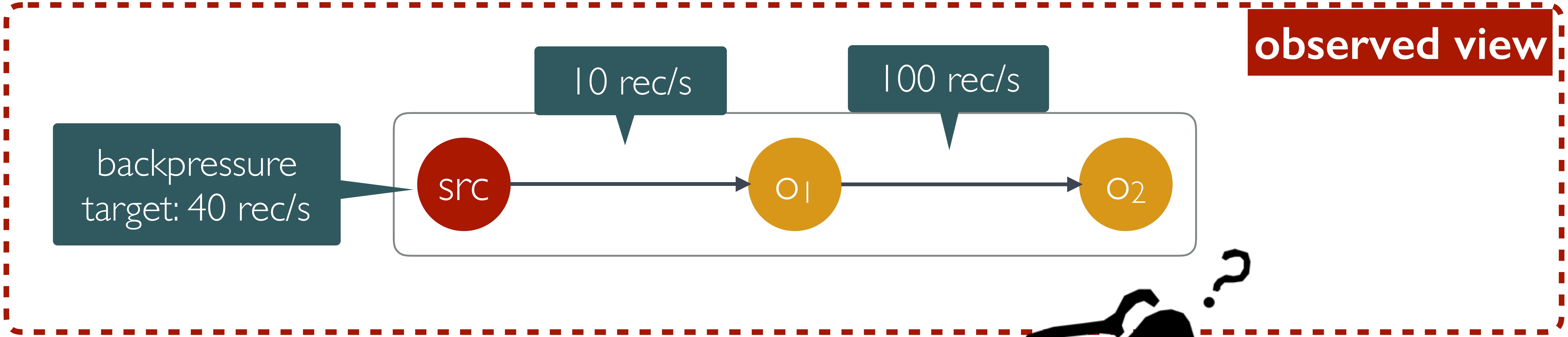
predictive, dataflow-wide actions

✓ **no oscillations**

✓ **true rates as bounds to avoid over/under-shoot**

✓ **fast convergence**

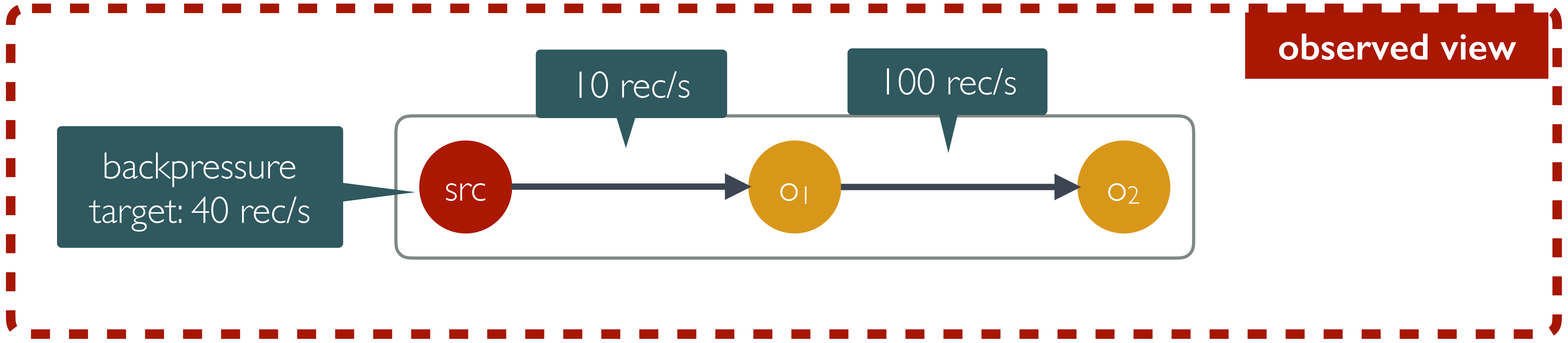

backpressure target: 40 rec/s
10 rec/s
100 rec/s
src
o1
o2

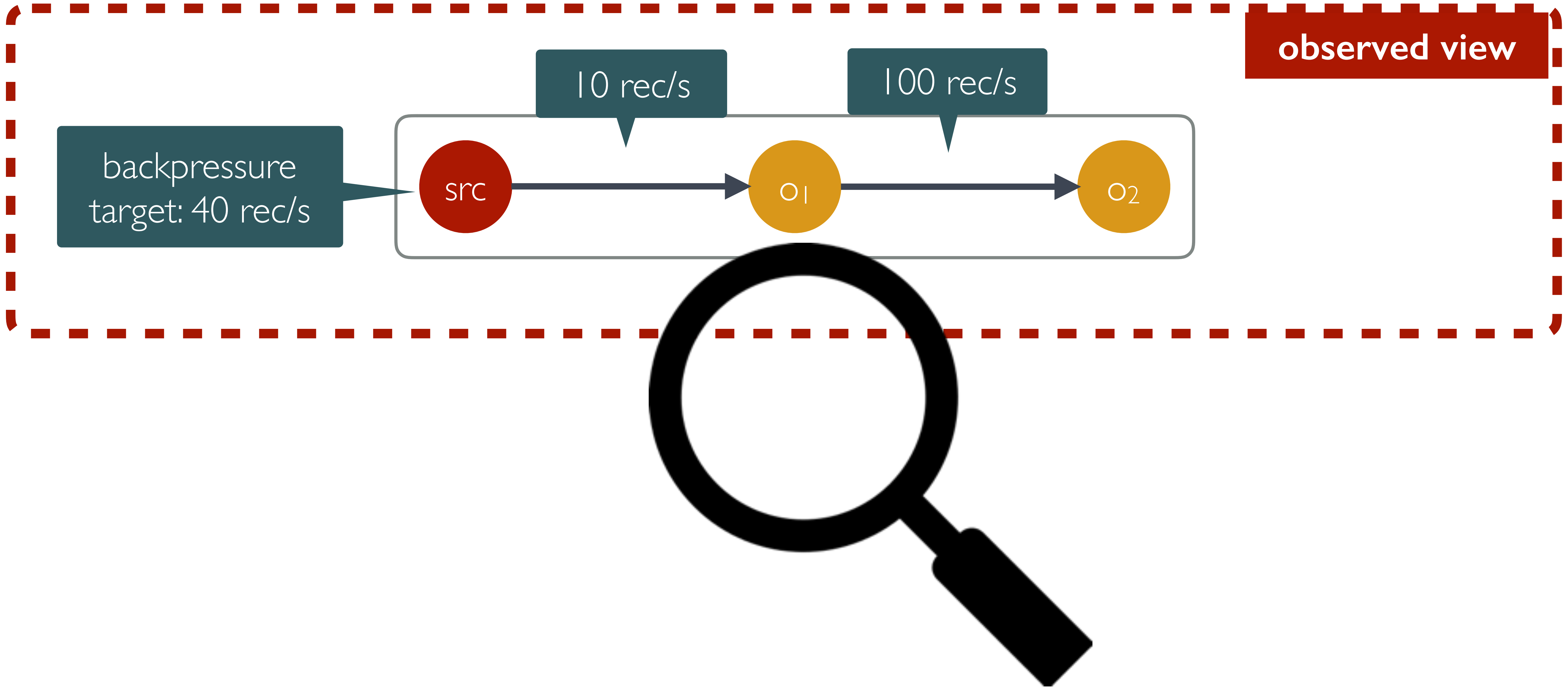


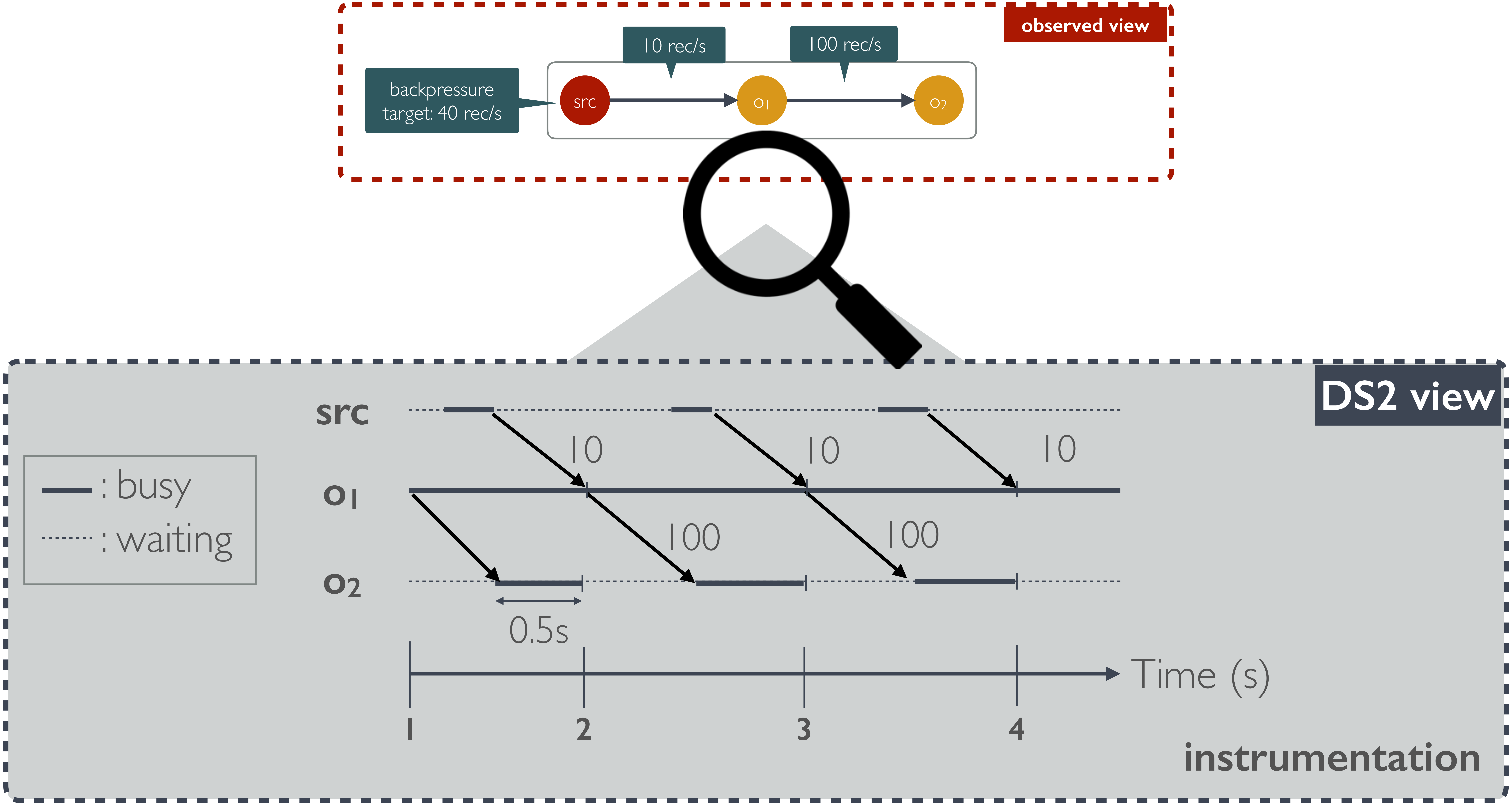
Which operator is the bottleneck?

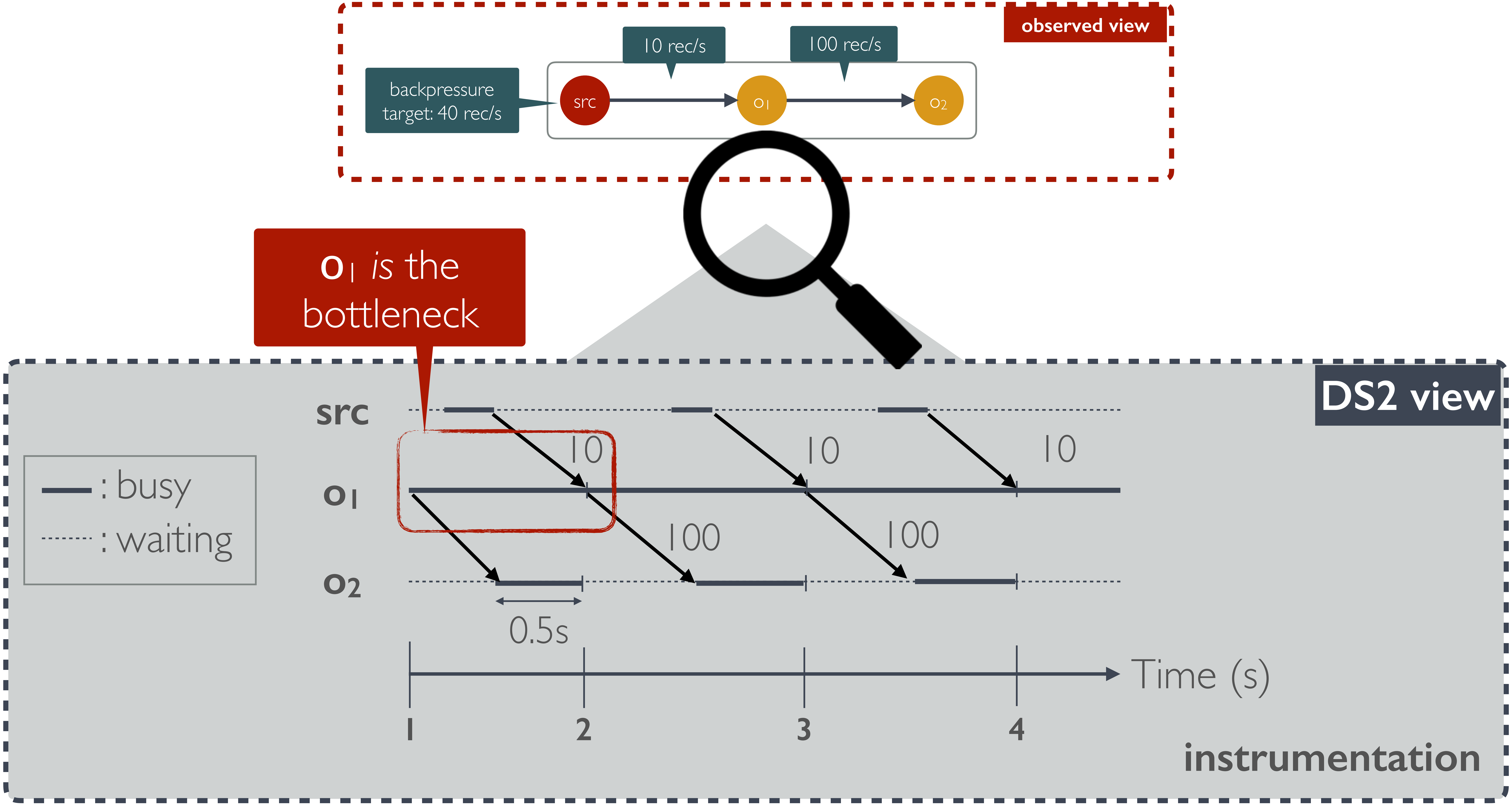
What if we scale o_1 x 4?

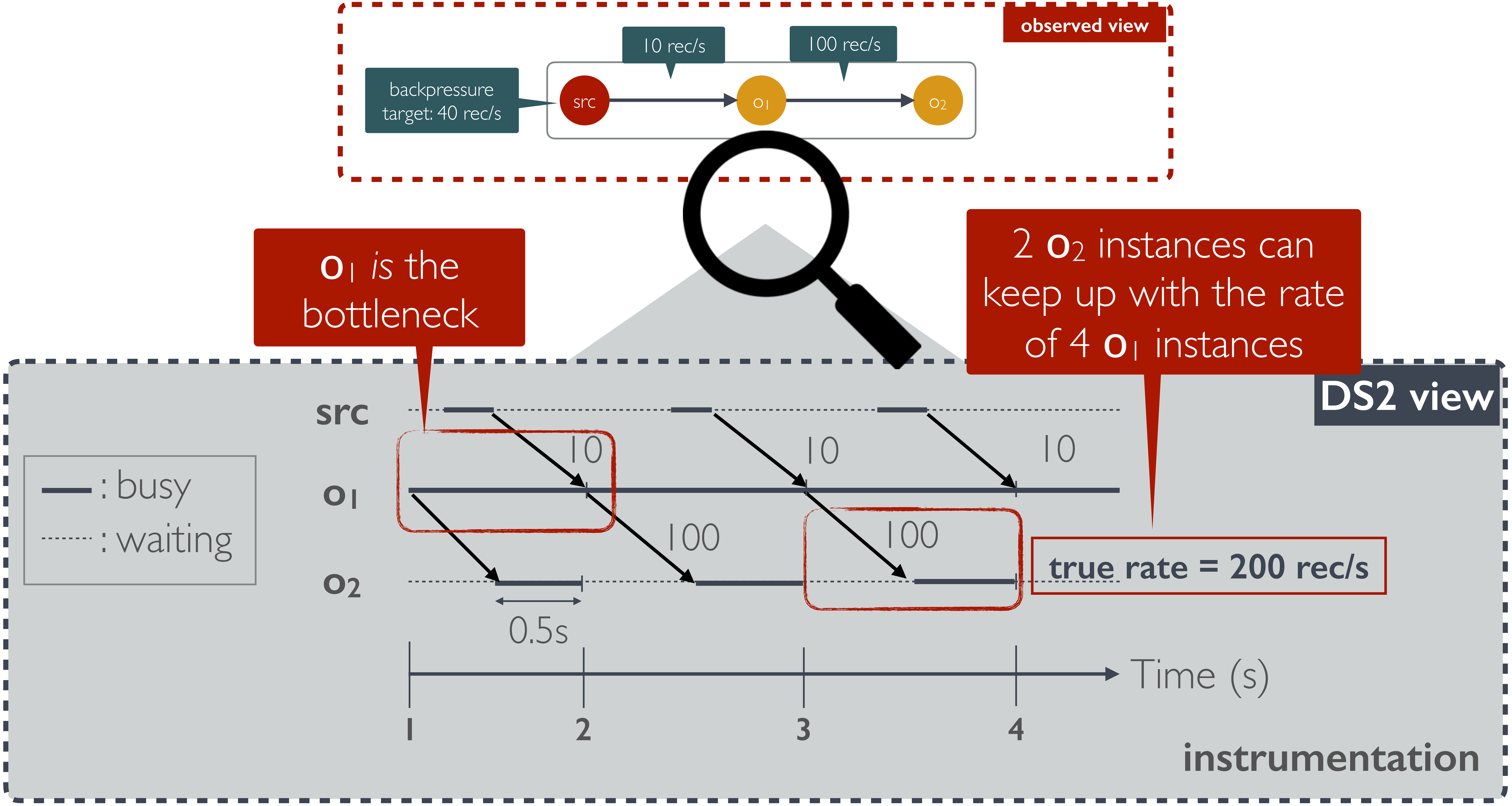
How much to scale o_2?


observed view
10 rec/s
100 rec/s
backpressure
target: 40 rec/s
src
o1
o2


observed view
10 rec/s
100 rec/s
backpressure
target: 40 rec/s
src
o1
o2


observed view
10 rec/s
100 rec/s
backpressure
target: 40 rec/s
src
o1
o2
DS2 view
src
o1
o2
: busy
: waiting
10
10
10
100
100
0.5s
Time (s)
1
2
3
4
instrumentation


observed view
10 rec/s
100 rec/s
backpressure target: 40 rec/s
src
o1
o2
o1 is the bottleneck
DS2 view
src
o1
o2
: busy
: waiting
10
10
10
100
100
0.5s
Time (s)
1
2
3
4
instrumentation


observed view
10 rec/s
100 rec/s
backpressure target: 40 rec/s
src
o1
o2
o1 is the bottleneck
2 o2 instances can keep up with the rate of 4 o1 instances
DS2 view
src
o1
o2
: busy
: waiting
10
10
10
100
100
true rate = 200 rec/s
0.5s
Time (s)
1
2
3
4
instrumentation

THE DS2 MODEL

THE DS2 MODEL

Useful time: The time spent by an operator instance in **deserialization**, **processing**, and **serialization** activities.

THE DS2 MODEL

Useful time: The time spent by an operator instance in **deserialization**, **processing**, and **serialization** activities.

True processing (resp. output) rate: The number of **records** an operator instance can process (resp. output) **per unit of useful time**.

THE DS2 MODEL

Useful time: The time spent by an operator instance in **deserialization**, **processing**, and **serialization** activities.

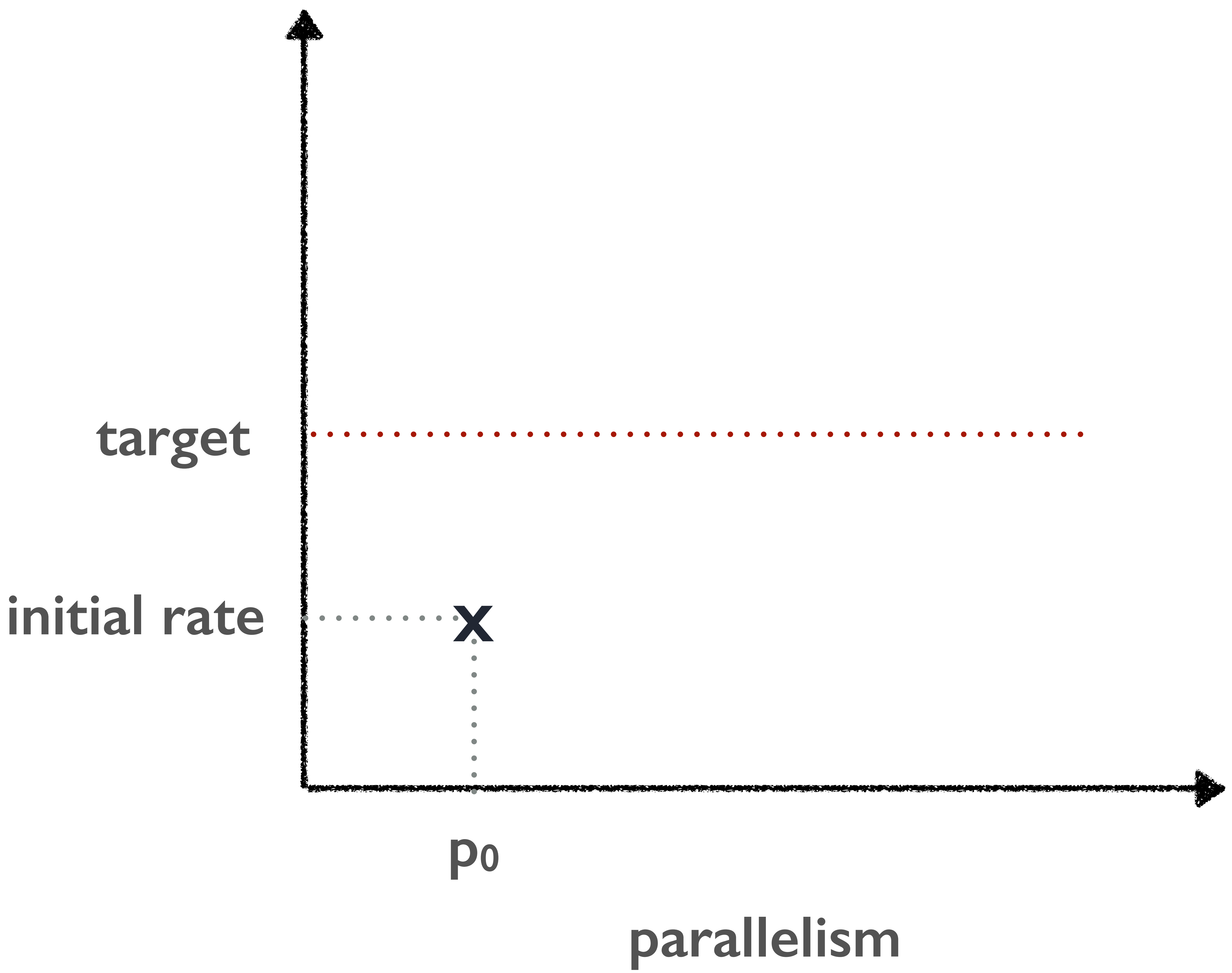
True processing (resp. output) rate: The number of **records** an operator instance can process (resp. output) **per unit of useful time**.

Optimal parallelism for o_i : $\dfrac{\text{aggregated true output rate of upstream ops}}{\text{average true processing rate of } o_i}$

CONVERGENCE STEPS

if the actual scaling is linear,
convergence takes **one** step

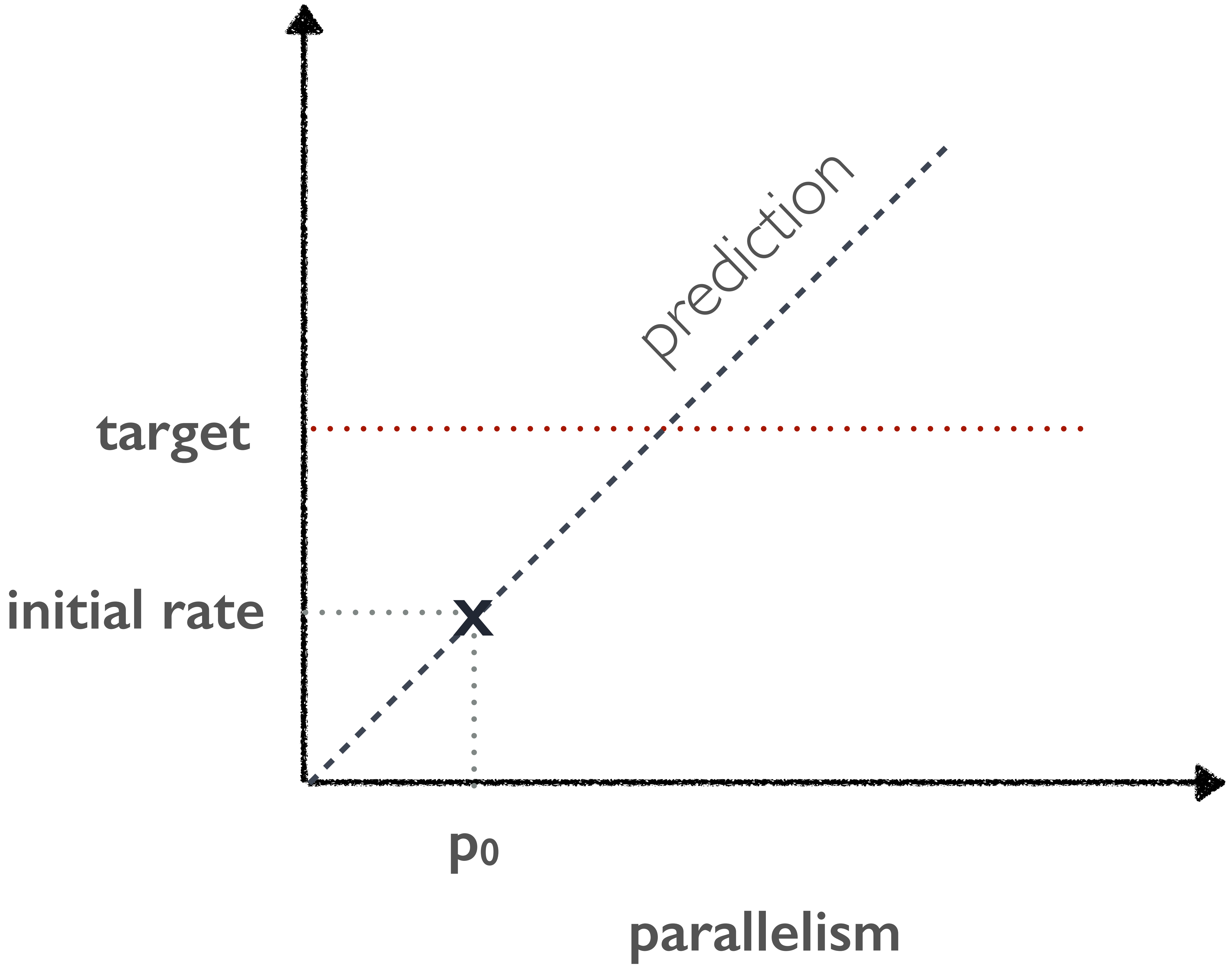
- **no overshoot** when scaling up: ideal rate is an *upper* bound
- **no undershoot** when scaling down: ideal rate is a *lower* bound



CONVERGENCE STEPS

if the actual scaling is linear, convergence takes **one** step

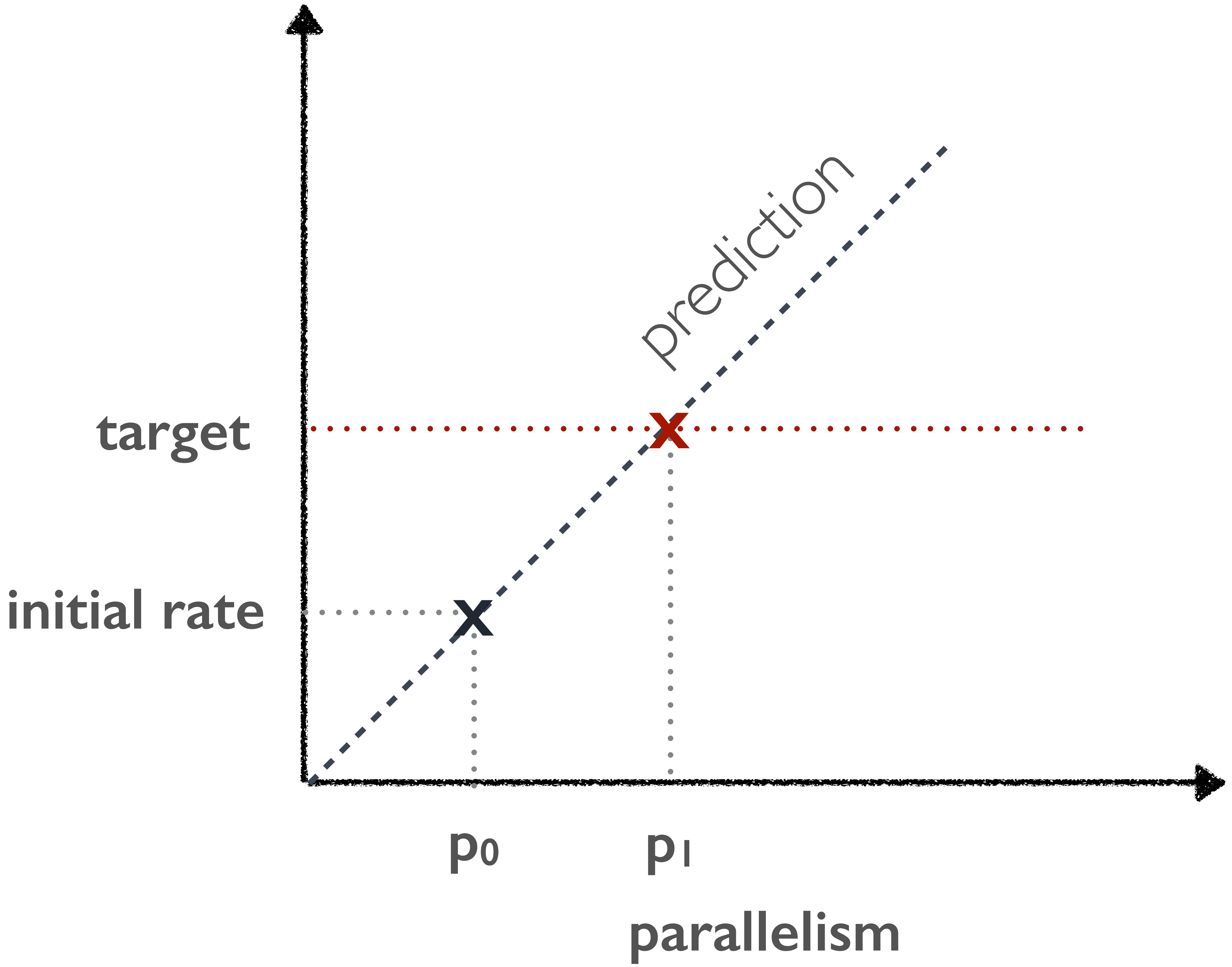
- **no overshoot** when scaling up: ideal rate is an *upper* bound
- **no undershoot** when scaling down: ideal rate is a *lower* bound



CONVERGENCE STEPS

if the actual scaling is linear,
convergence takes **one** step

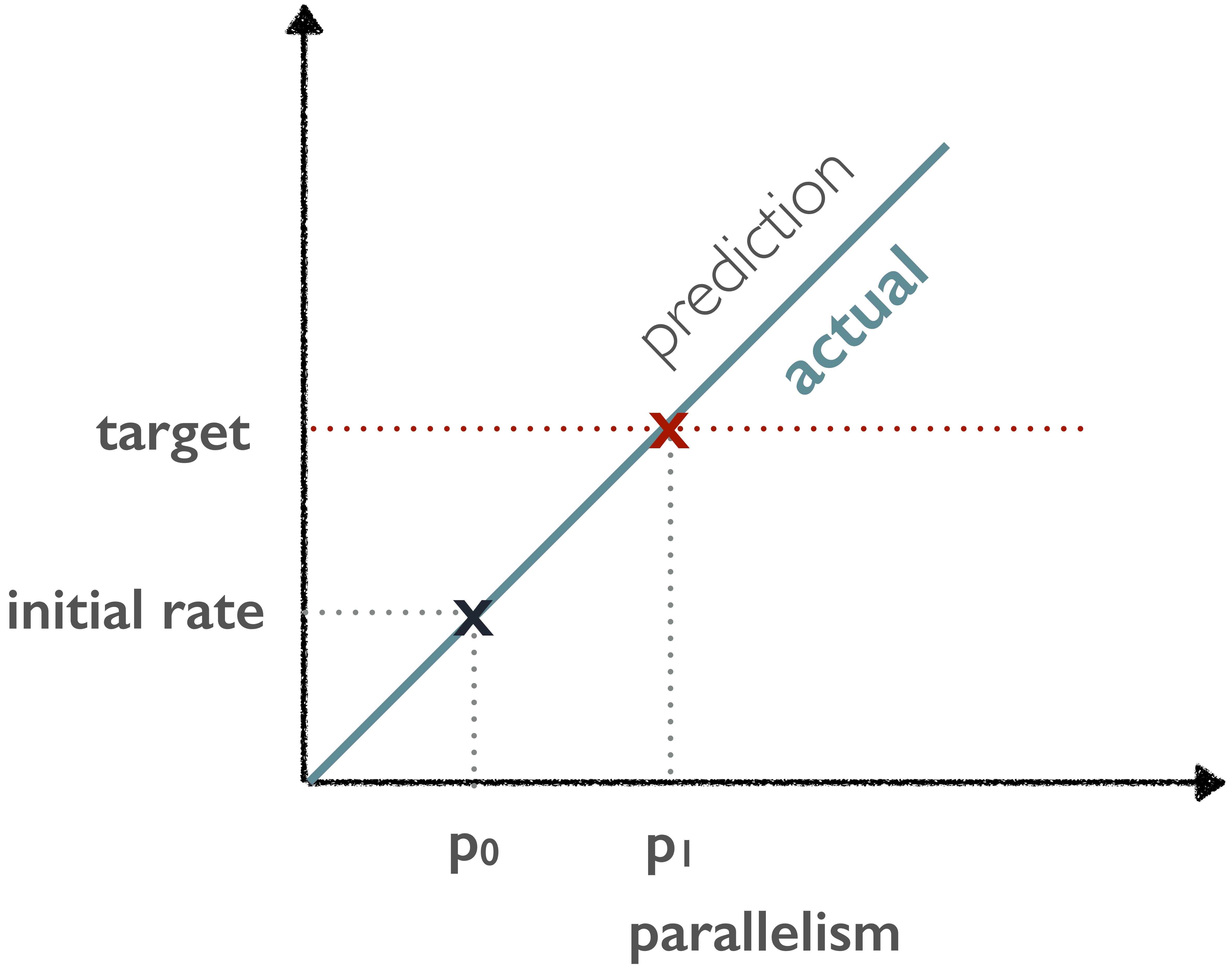
- **no overshoot** when scaling up: ideal rate is an *upper* bound
- **no undershoot** when scaling down: ideal rate is a *lower* bound



CONVERGENCE STEPS

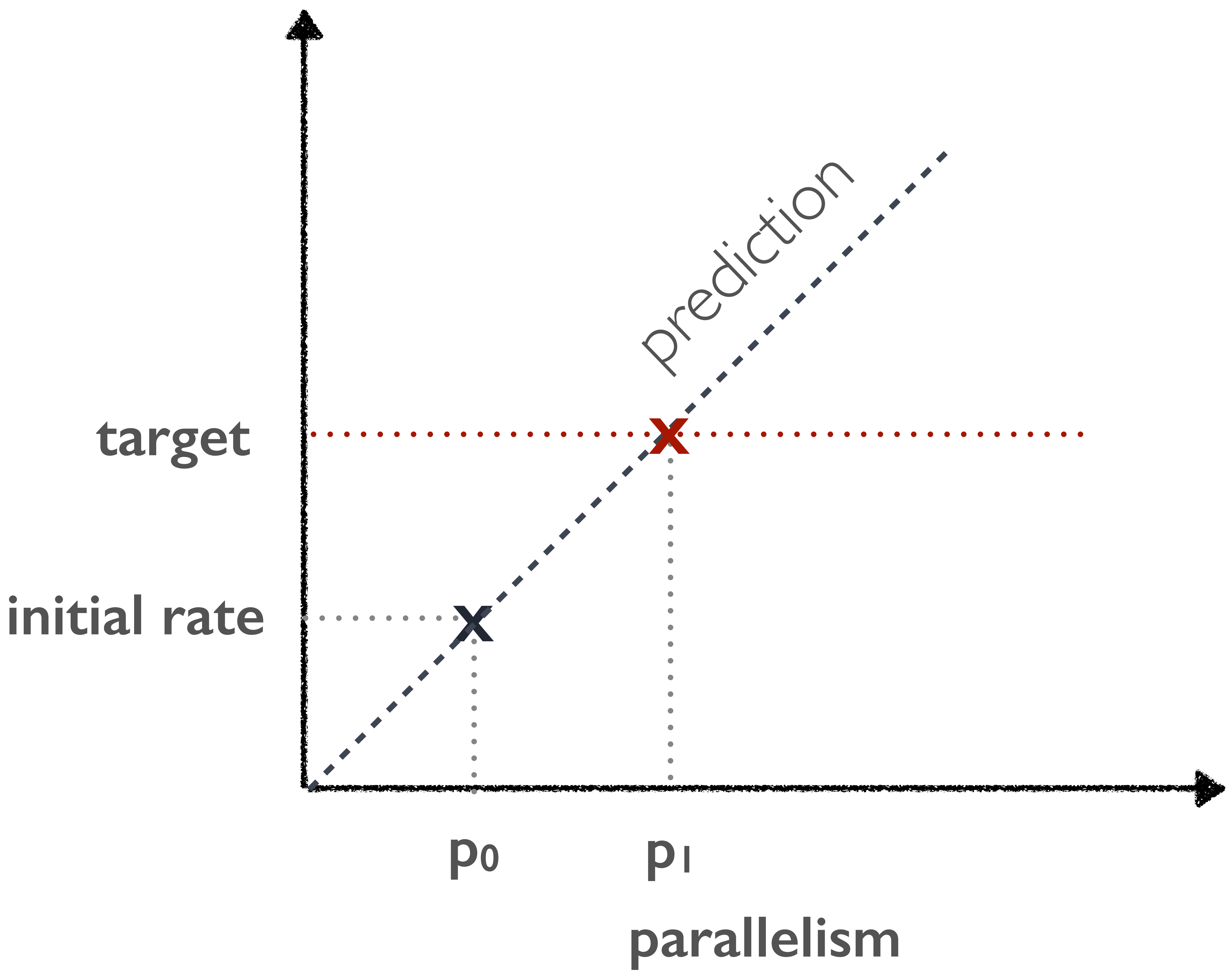
if the actual scaling is linear, convergence takes **one** step

- **no overshoot** when scaling up: ideal rate is an *upper* bound
- **no undershoot** when scaling down: ideal rate is a *lower* bound



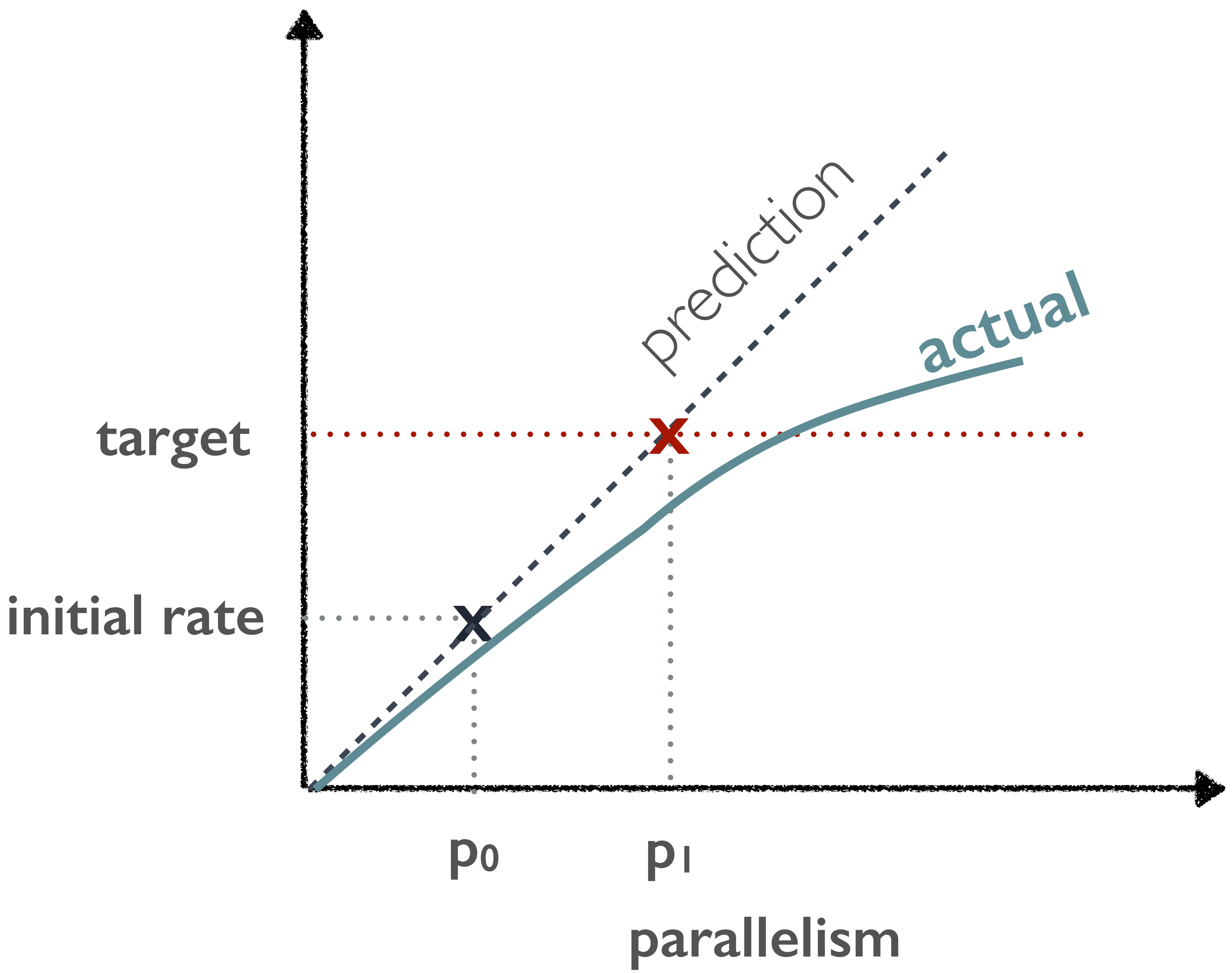
CONVERGENCE STEPS

In practice, rates are commonly **sub-linear** due to other overheads (e.g. worker coordination).



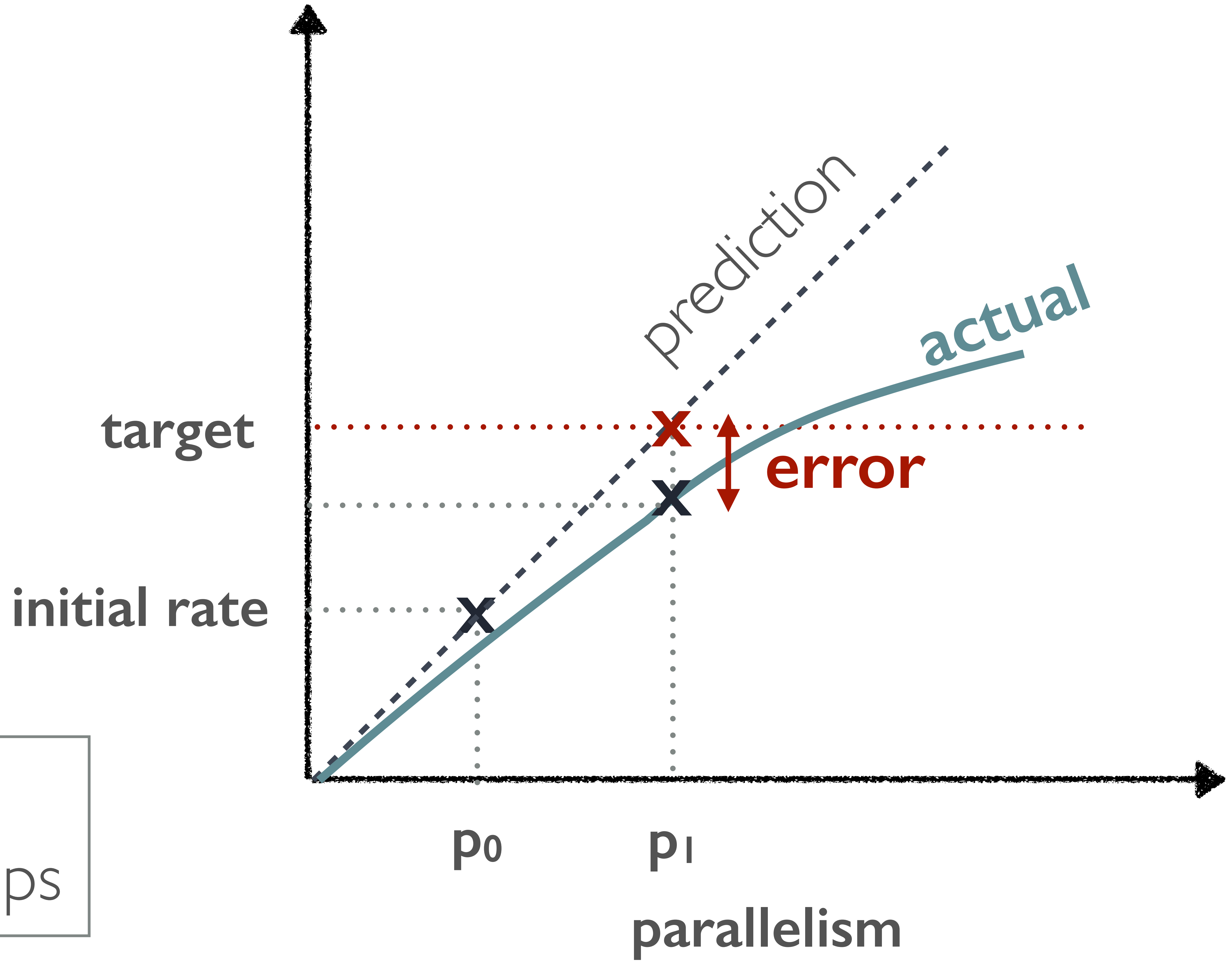
CONVERGENCE STEPS

In practice, rates are commonly **sub-linear** due to other overheads (e.g. worker coordination).



CONVERGENCE STEPS

In practice, rates are commonly **sub-linear** due to other overheads (e.g. worker coordination).

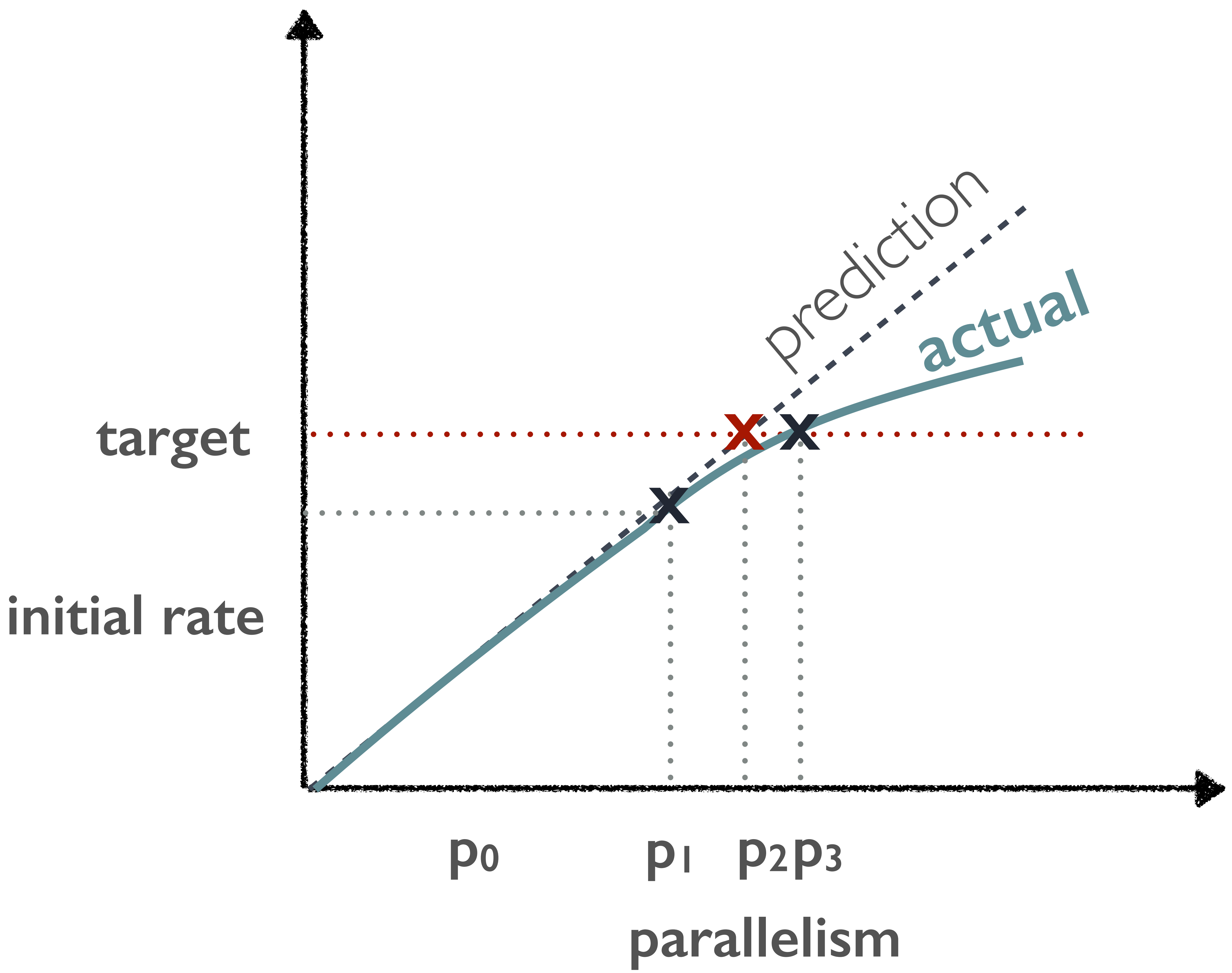


when the actual scaling is sub-linear, convergence takes **more than one** steps

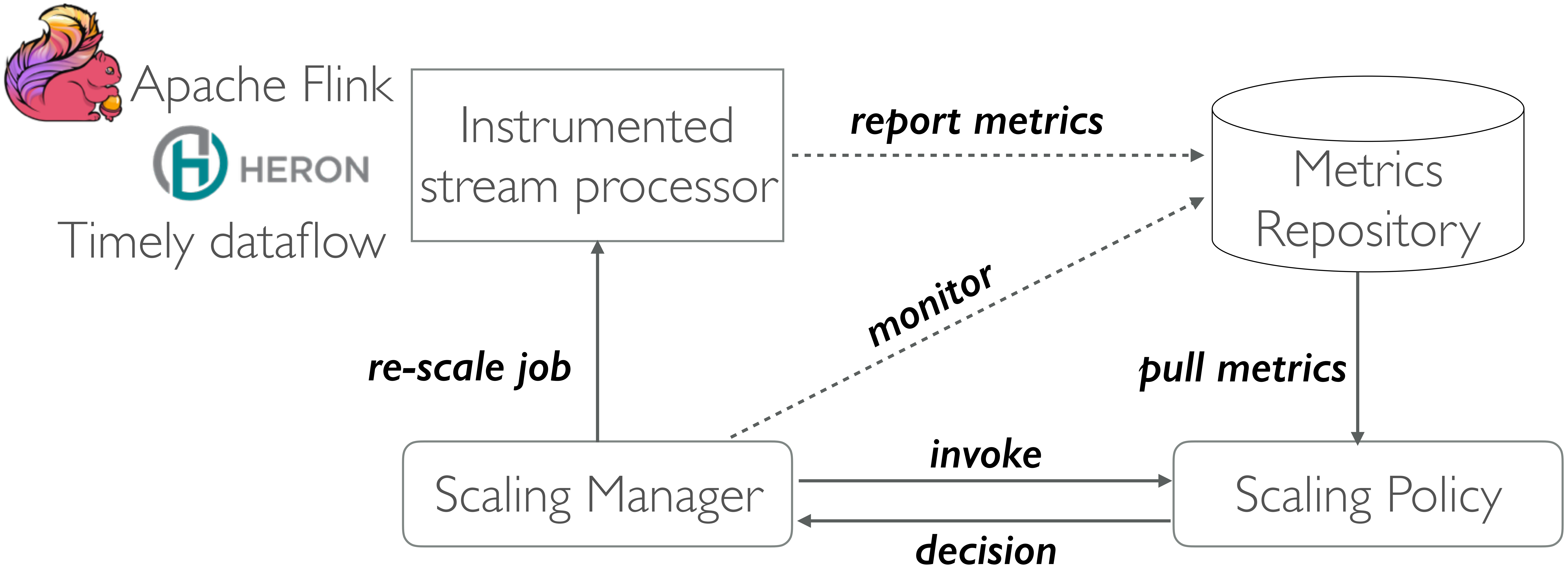
CONVERGENCE STEPS

In practice, rates are commonly **sub-linear** due to other overheads (e.g. worker coordination).

In our experiments, DS2 took **up to three steps** to converge for complex queries.

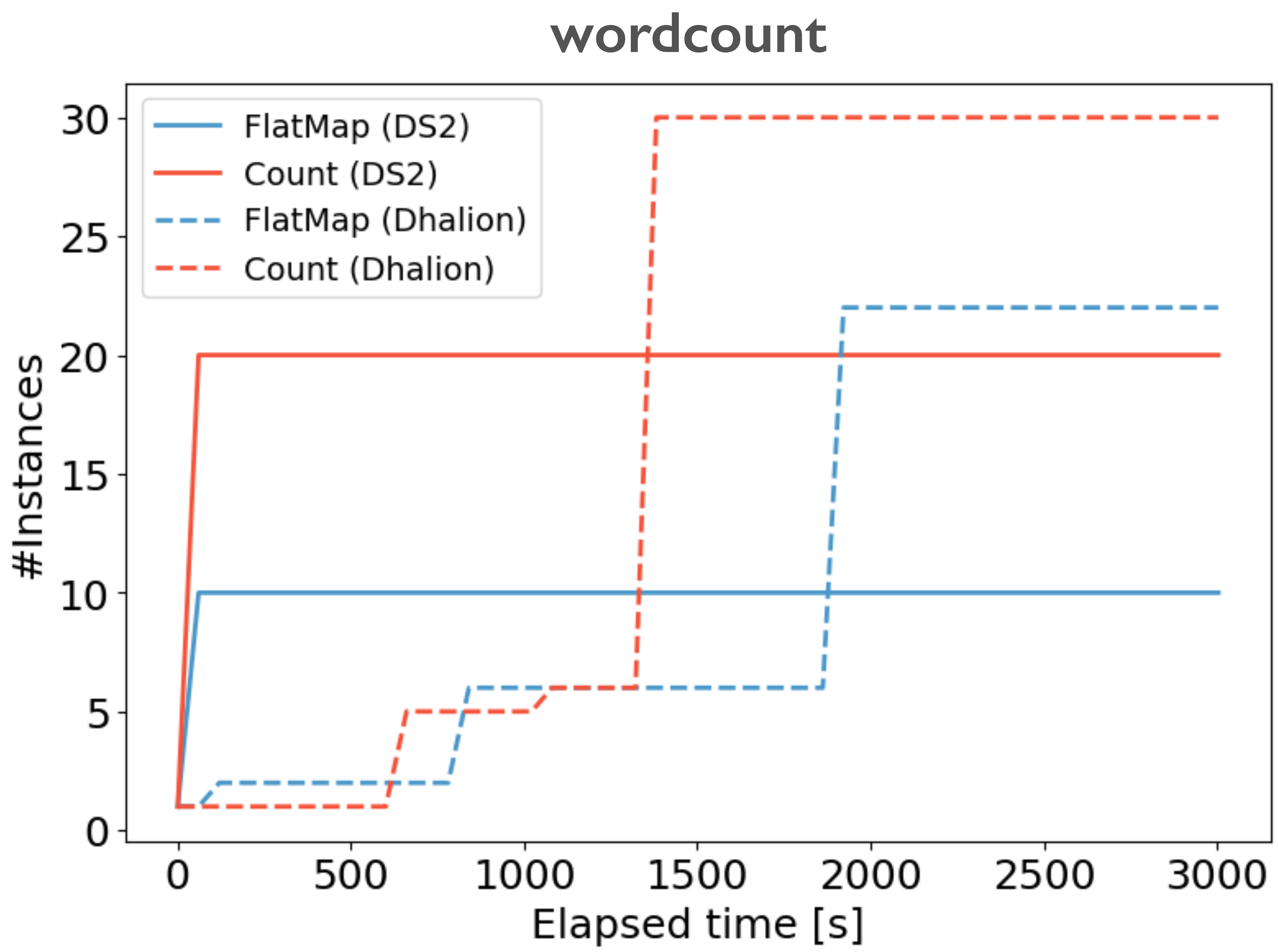


DS2 operates *online* in a *reactive* setting



EVALUATION

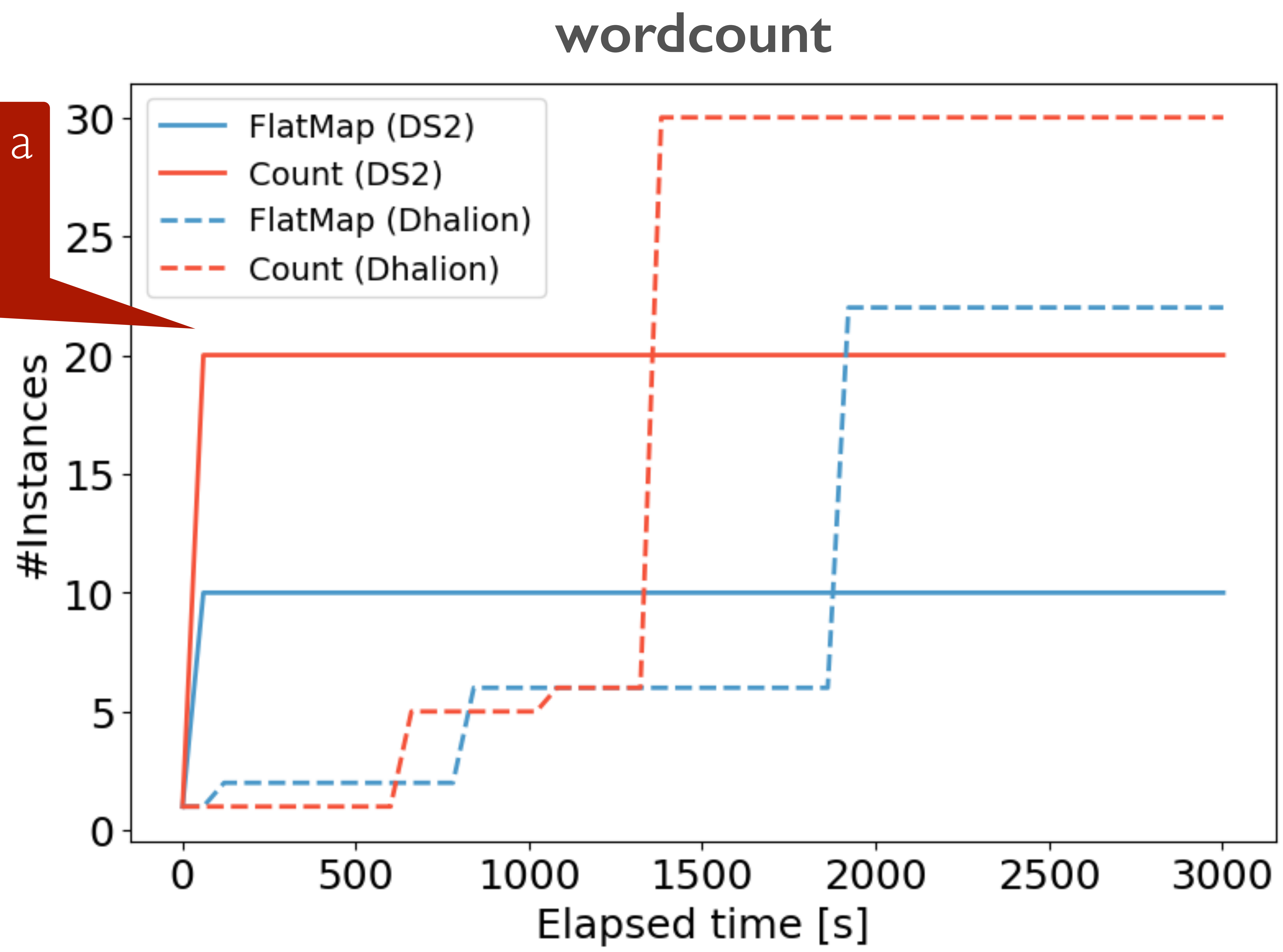
DS2 VS. DHALION ON HERON



Target rate: 16.700 rec/s

DS2 VS. DHALION ON HERON

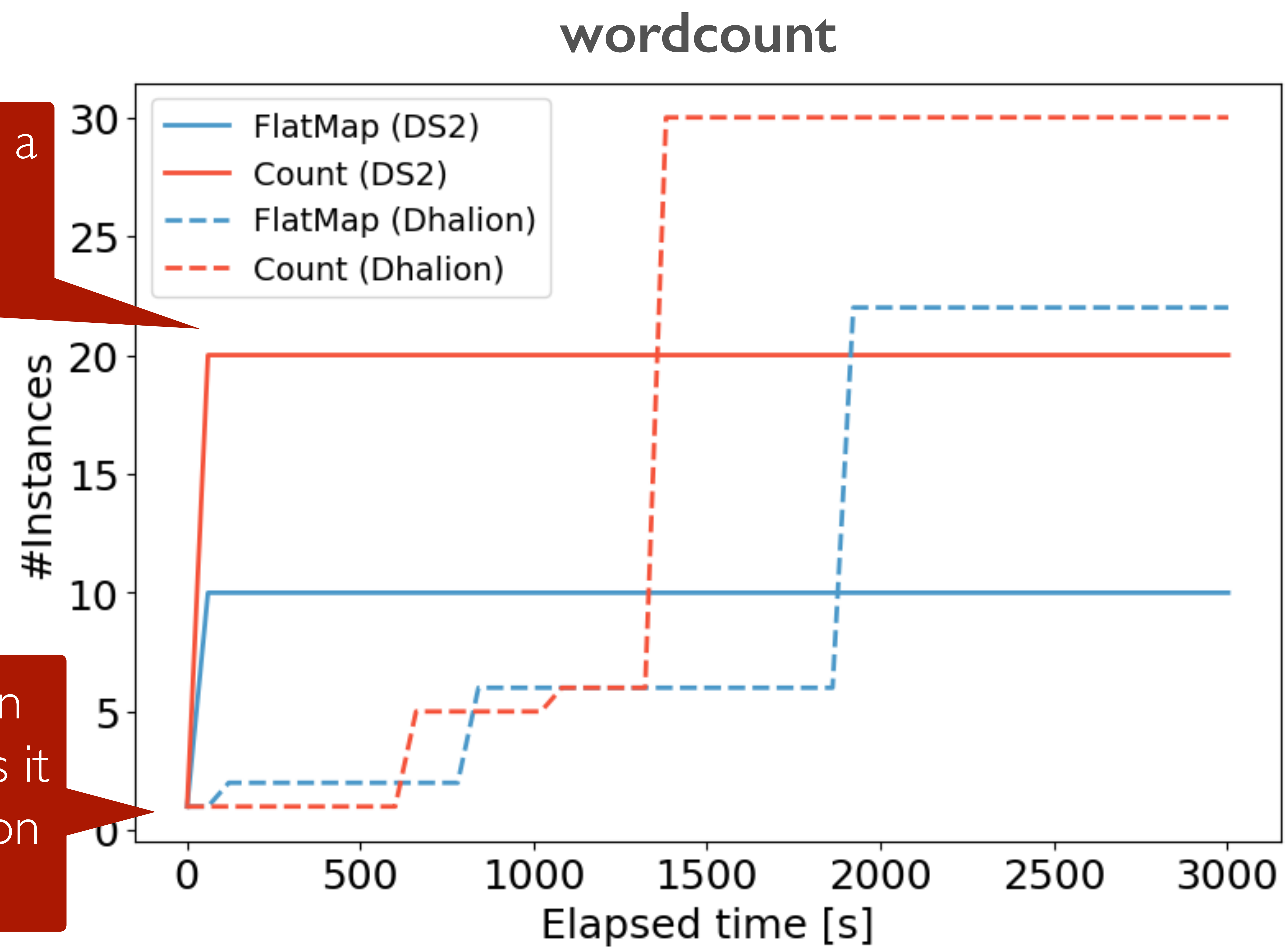
DS2 converges in a **single step** for both operators



Target rate: 16.700 rec/s

DS2 VS. DHALION ON HERON

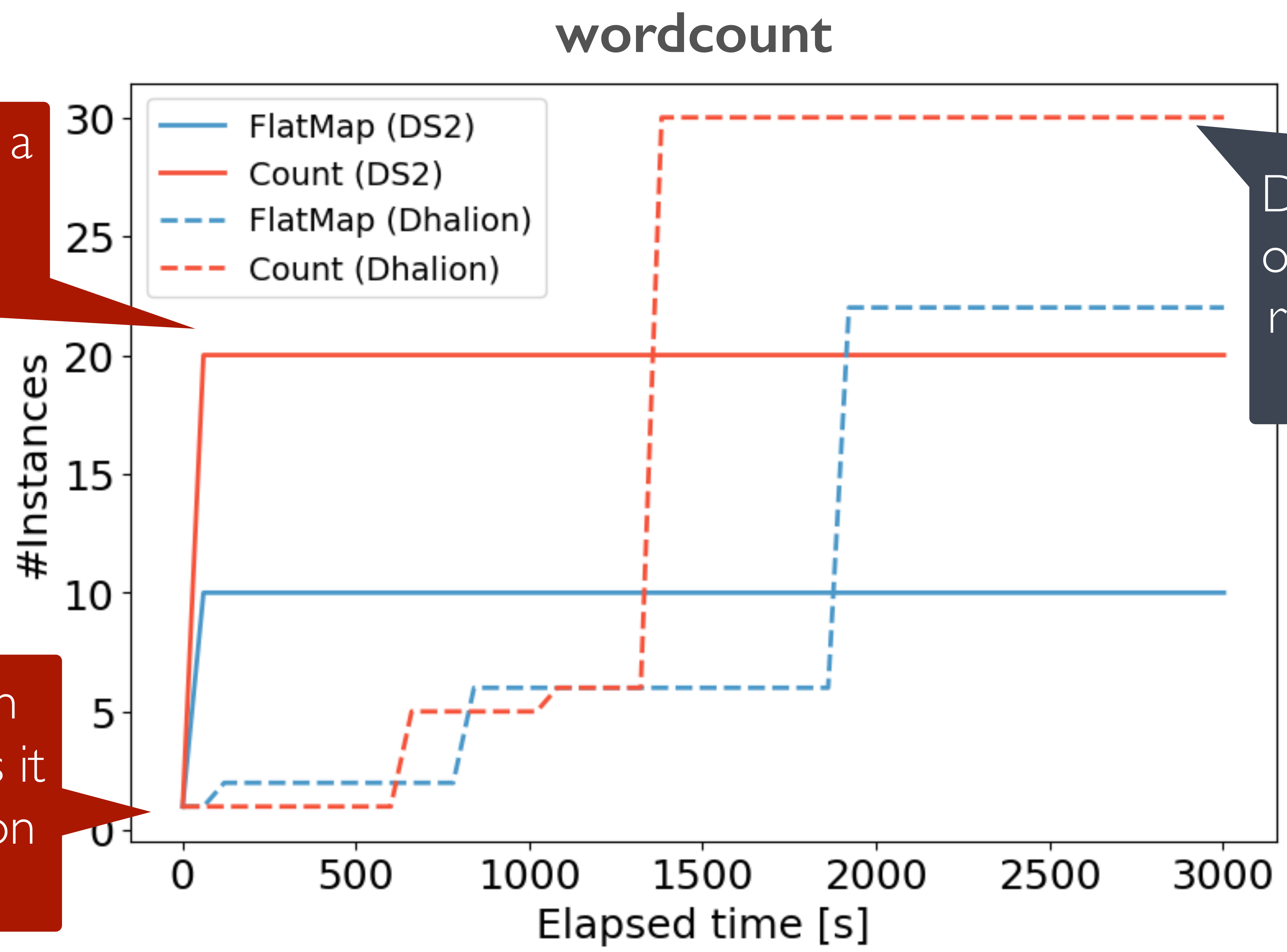
DS2 converges in a **single step** for both operators



DS2 converges in **60s**, i.e. as soon as it receives the Heron metrics

Target rate: 16.700 rec/s

DS2 VS. DHALION ON HERON



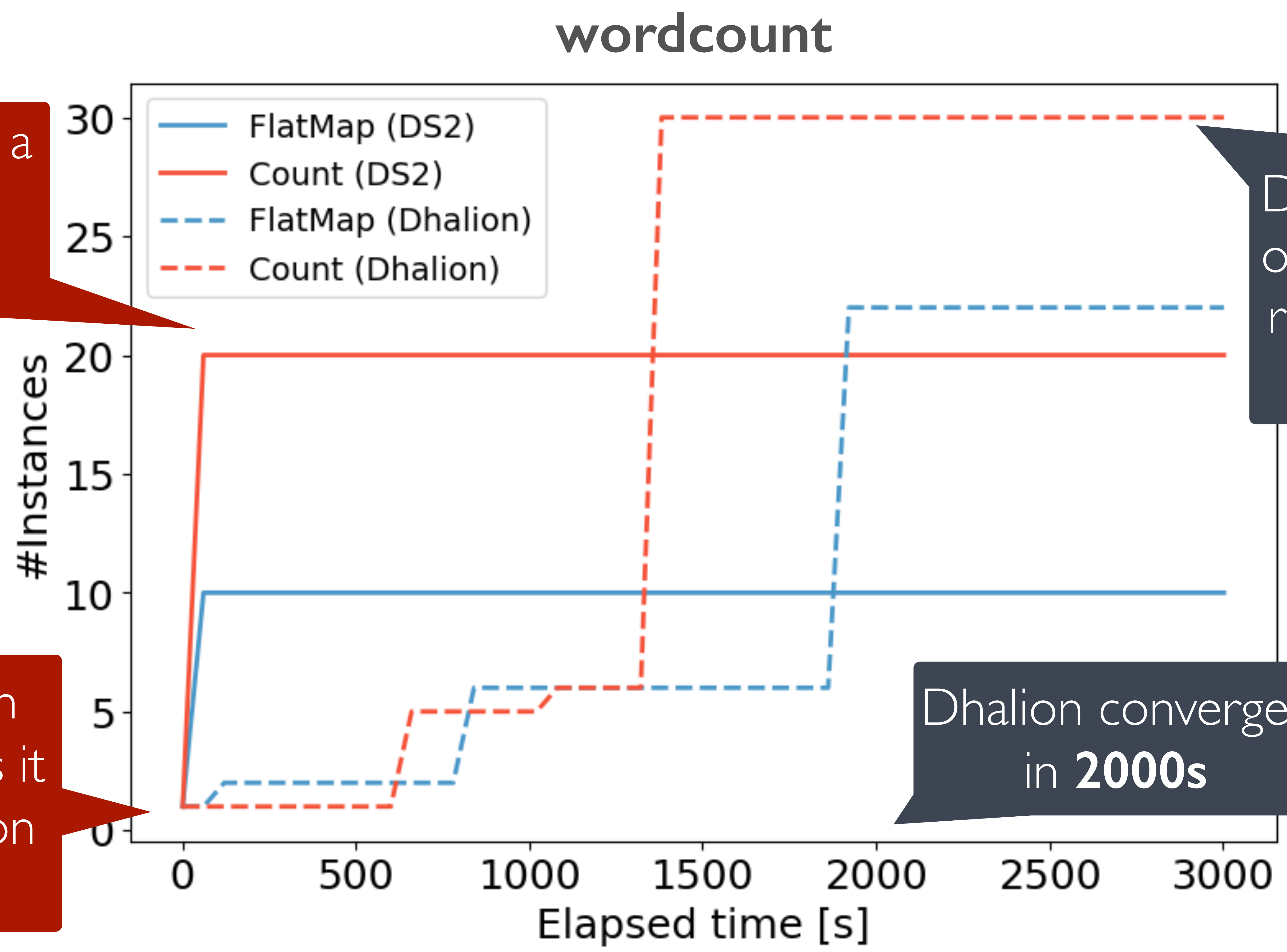
DS2 converges in a **single step** for both operators

DS2 converges in **60s**, i.e. as soon as it receives the Heron metrics

Dhalion scales one operator at a time, resulting to a total of **six steps**

Target rate: 16.700 rec/s

DS2 VS. DHALION ON HERON



DS2 converges in a **single step** for both operators

DS2 converges in **60s**, i.e. as soon as it receives the Heron metrics

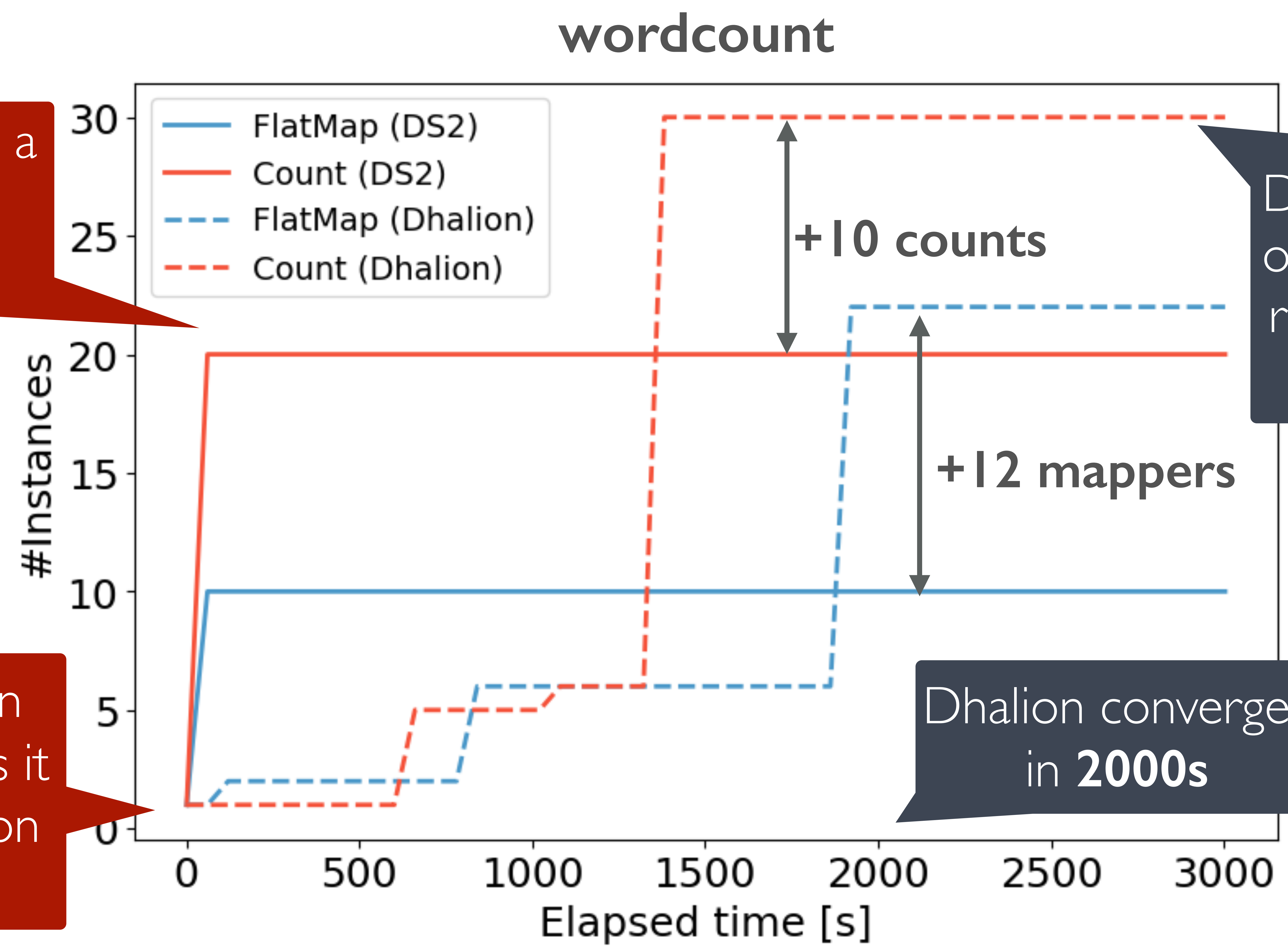
Dhalion scales one operator at a time, resulting to a total of **six steps**

Dhalion converges in **2000s**

Target rate: 16.700 rec/s

DS2 VS. DHALION ON HERON

DS2 converges in a **single step** for both operators



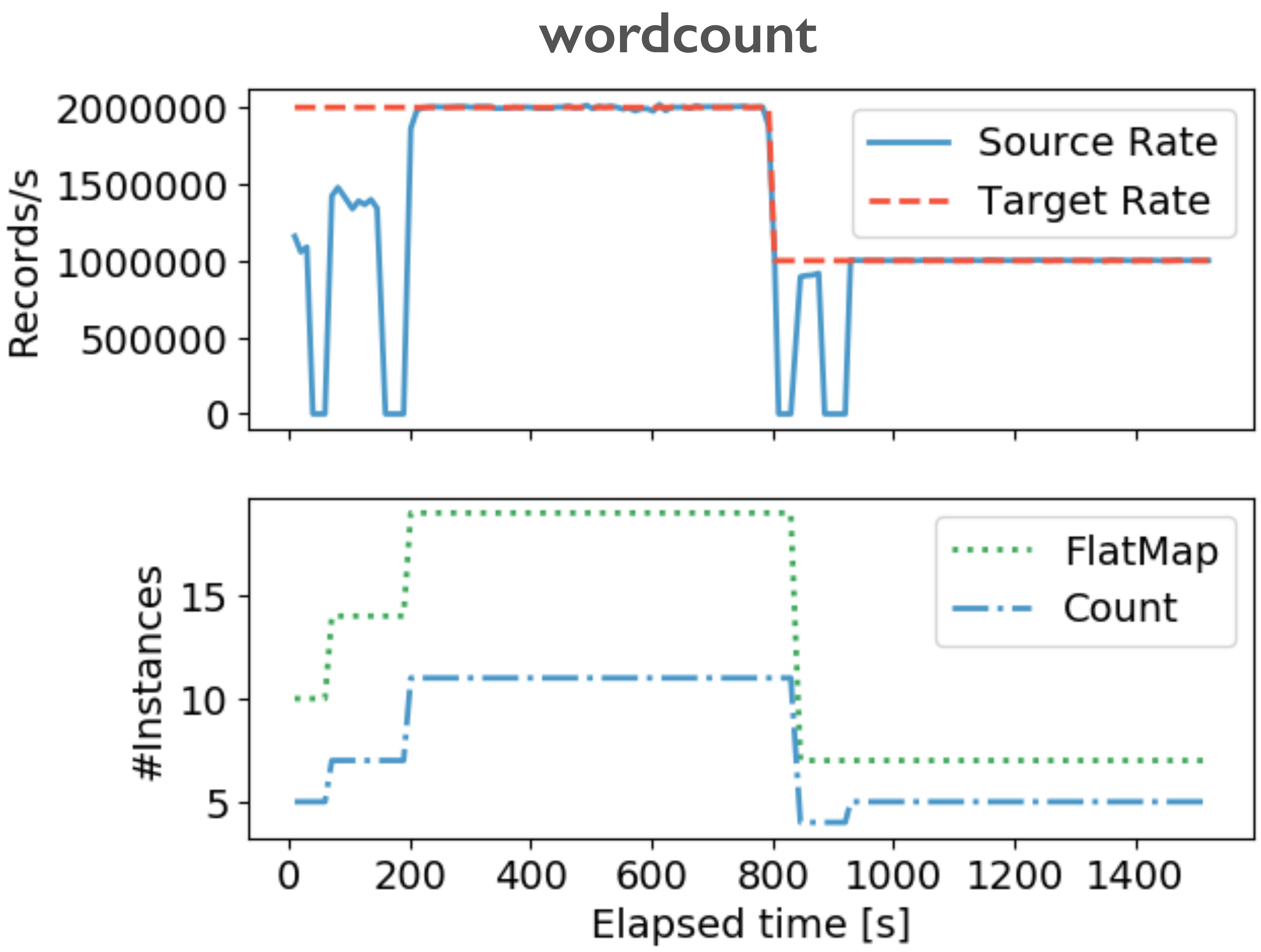
Dhalion scales one operator at a time, resulting to a total of **six steps**

DS2 converges in **60s**, i.e. as soon as it receives the Heron metrics

Dhalion converges in **2000s**

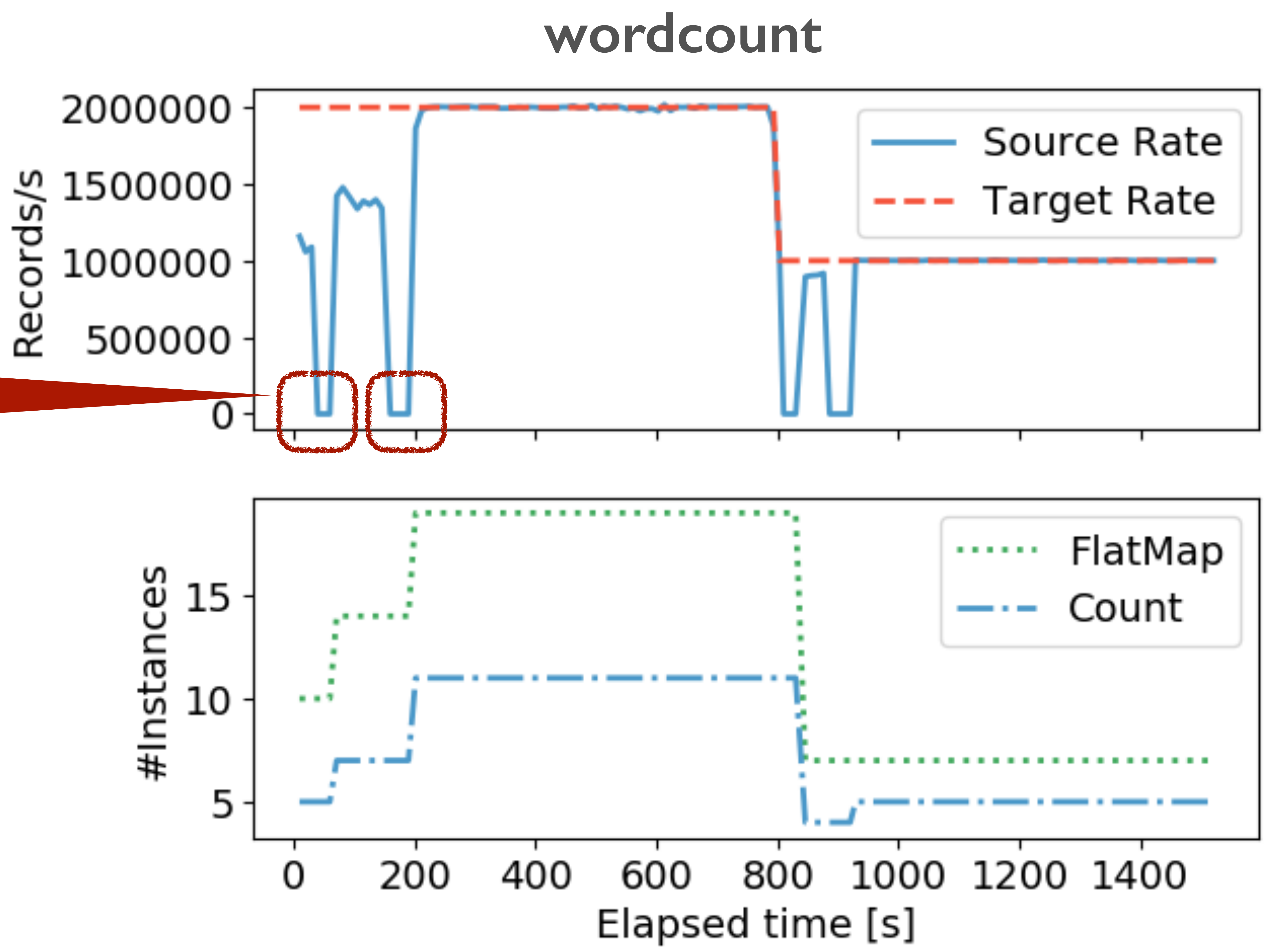
Target rate: 16.700 rec/s

DS2 ON APACHE FLINK



Target rate: 2.000.000 rec/s

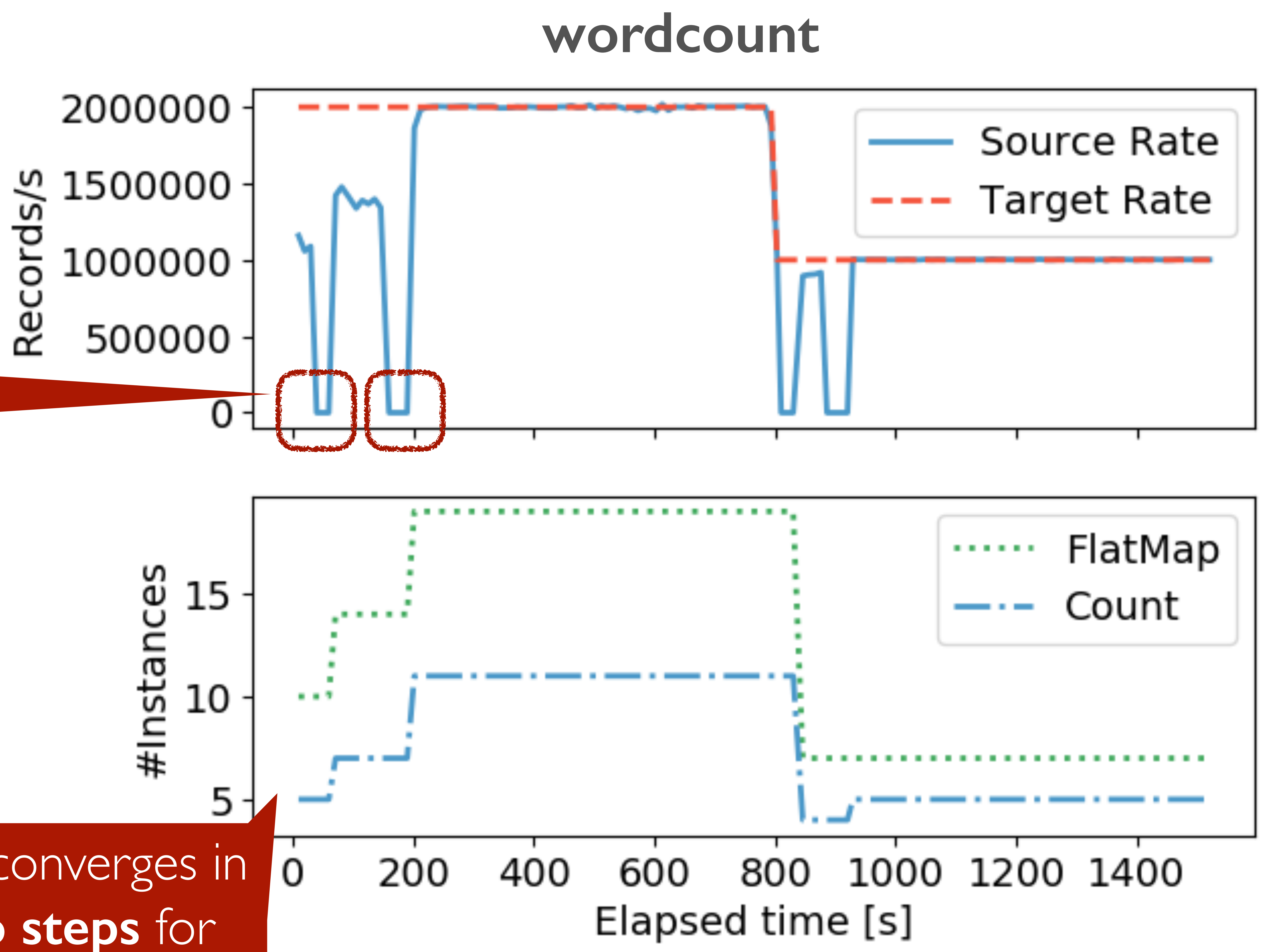
DS2 ON APACHE FLINK



Apache Flink savepoint and reconfiguration takes ~**30s**

Target rate: 2.000.000 rec/s

DS2 ON APACHE FLINK

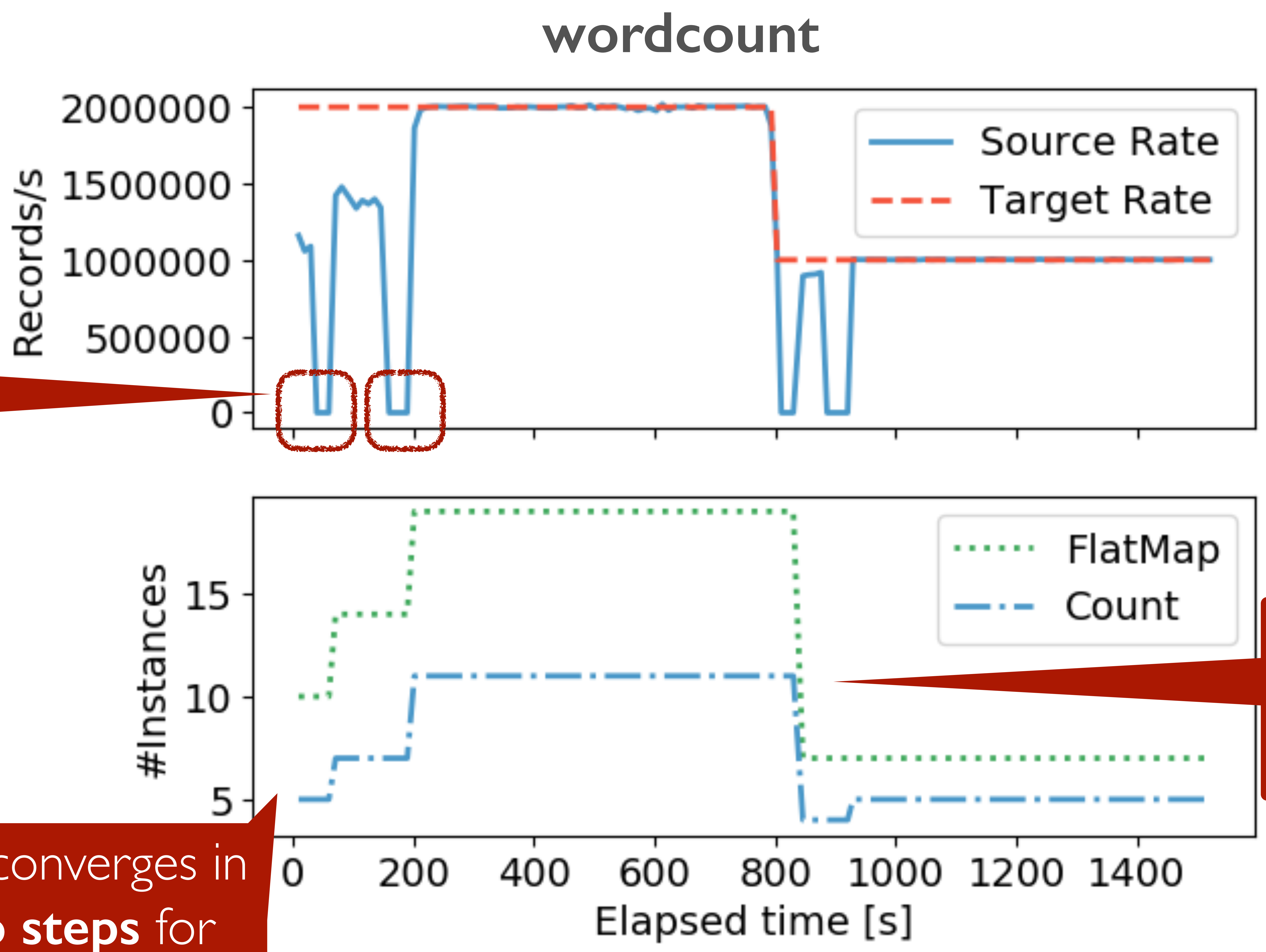


Apache Flink savepoint and reconfiguration takes ~**30s**

DS2 converges in **two steps** for both operators

Target rate: 2.000.000 rec/s

DS2 ON APACHE FLINK



Apache Flink savepoint and reconfiguration takes ~**30s**

DS2 reacts **3s** after the target rate has changed

DS2 converges in **two steps** for both operators

Target rate: 2.000.000 rec/s

CONVERGENCE - NEXMARK

initial parallelism	Q1: flatmap	Q2: filter	Q3: incremental join	Q5: tumbling window join	Q8: sliding window	Q11: session window
8 =>	12 => **16**	11 => 13 => **14**	16 => **20**	14 => 15 => **16**	**10**	12 => 22 => **28**
12 =>	**16**	**14**	18 => **20**	**16**	**10**	22 => **28**
16 =>	**16**	12 => **14**	**20**	**16**	8 => **10**	26 => **28**
20 =>	**16**	13 => **14**	**20**	14 => **16**	8 => **10**	**28**
24 =>	**16**	**14**	**20**	14 => **16**	8 => **10**	**28**
28 =>	**16**	**14**	**20**	13 => **16**	8 => **10**	**28**

=> : scaling action

CONVERGENCE - NEXMARK

initial parallelism	Q1: flatmap	Q2: filter	Q3: incremental join	Q5: tumbling window join	Q8: sliding window	Q11: session window
8 =>	12 => **16**	11 => 13 => **14**	16 => **20**	14 => 15 => **16**	**10**	12 => 22 => **28**
12 =>	**16**	**14**	18 => **20**	**16**	**10**	22 => **28**
16 =>	**16**	12 => **14**	**20**	**16**	8 => **10**	26 => **28**
20 =>	**16**	13 => **14**	**20**	14 => **16**	8 => **10**	**28**
24 =>	**16**	**14**	**20**	14 => **16**	8 => **10**	**28**
28 =>	**16**	**14**	**20**	13 => **16**	8 => **10**	**28**

scale-up

=> : scaling action

CONVERGENCE - NEXMARK

initial parallelism	Q1: flatmap	Q2: filter	Q3: incremental join	Q5: tumbling window join	Q8: sliding window	Q11: session window
8 =>	12 => **16**	11 => 13 => **14**	16 => **20**	14 => 15 => **16**	**10**	12 => 22 => **28**
12 =>	**16**	**14**	18 => **20**	**16**	**10**	22 => **28**
16 =>	**16**	12 => **14**	**20**	**16**	8 => **10**	26 => **28**
20 =>	**16**	13 => **14**	**20**	14 => **16**	8 => **10**	**28**
24 =>	**16**	**14**	**20**	14 => **16**	8 => **10**	**28**
28 =>	**16**	**14**	**20**	13 => **16**	8 => **10**	**28**

scale-up

scale-down

=> : scaling action

CONVERGENCE - NEXMARK

initial parallelism	Q1: flatmap	Q2: filter	Q3: incremental join	Q5: tumbling window join	Q8: sliding window	Q11: session window
8 =>	12 => **16**	11 => 13 => **14**	16 => **20**	14 => 15 => **16**	**10**	12 => 22 => **28**
12 =>	**16**	**14**	18 => **20**	**16**	**10**	22 => **28**
16 =>	**16**	12 => **14**	**20**	**16**	8 => **10**	26 => **28**
20 =>	**16**	13 => **14**	**20**	14 => **16**	8 => **10**	**28**
24 =>	**16**	**14**	**20**	14 => **16**	8 => **10**	**28**
28 =>	**16**	**14**	**20**	13 => **16**	8 => **10**	**28**

scale-up

scale-down

a single step for many queries and initial configurations

=> : scaling action

CONVERGENCE - NEXMARK

at most **3 steps**

initial parallelism	Q1: flatmap	Q2: filter	Q3: incremental join	Q5: tumbling window join	Q8: sliding window	Q11: session window
8 =>	12 => **16**	11 => 13 => **14**	16 => **20**	14 => 15 => **16**	**10**	12 => 22 => **28**
12 =>	**16**	**14**	18 => **20**	**16**	**10**	22 => **28**
16 =>	**16**	12 => **14**	**20**	**16**	8 => **10**	26 => **28**
20 =>	**16**	13 => **14**	**20**	14 => **16**	8 => **10**	**28**
24 =>	**16**	**14**	**20**	14 => **16**	8 => **10**	**28**
28 =>	**16**	**14**	**20**	13 => **16**	8 => **10**	**28**

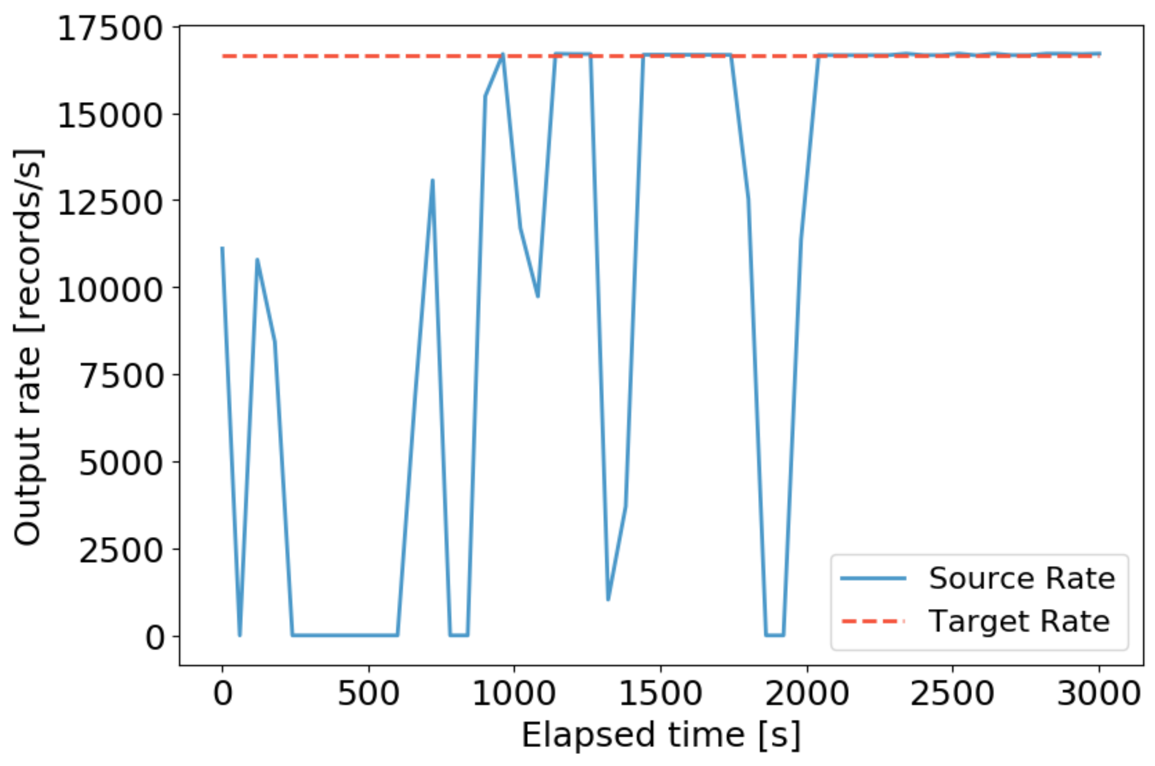
scale-up

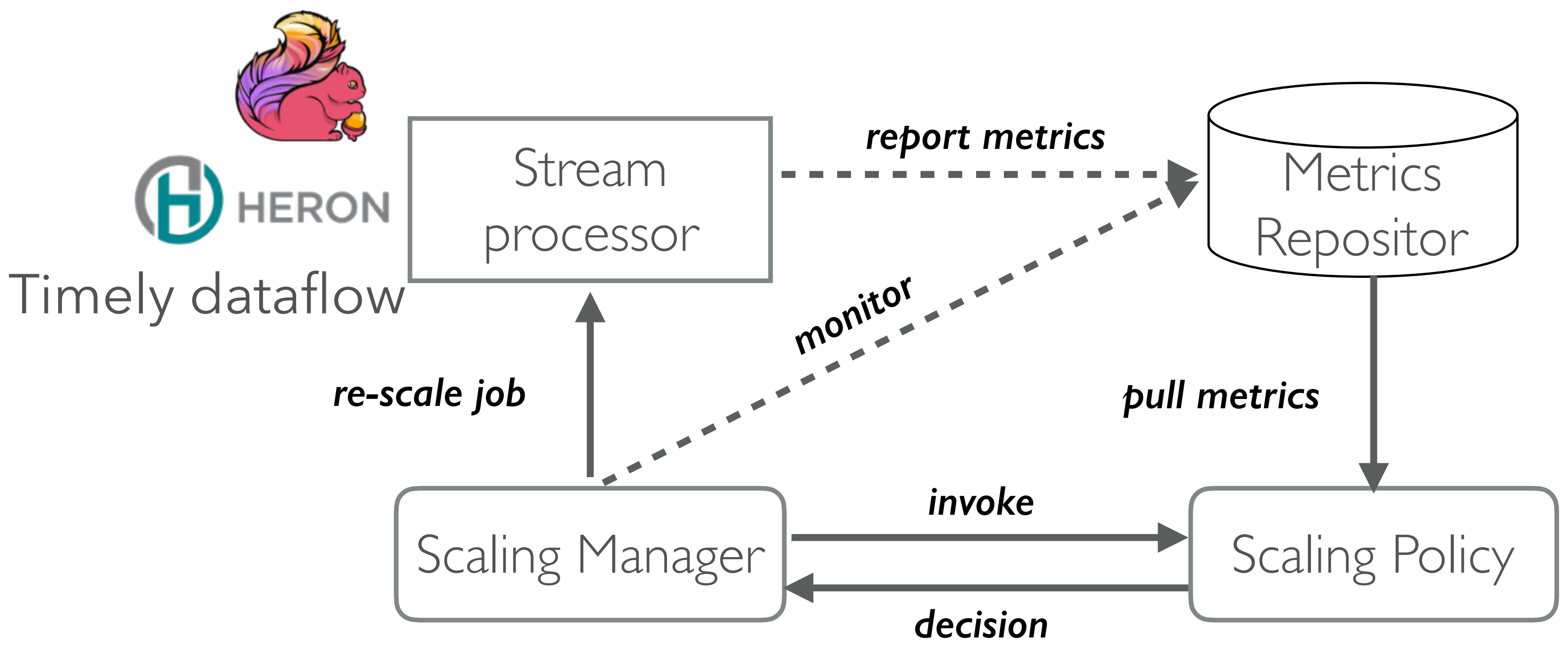
scale-down

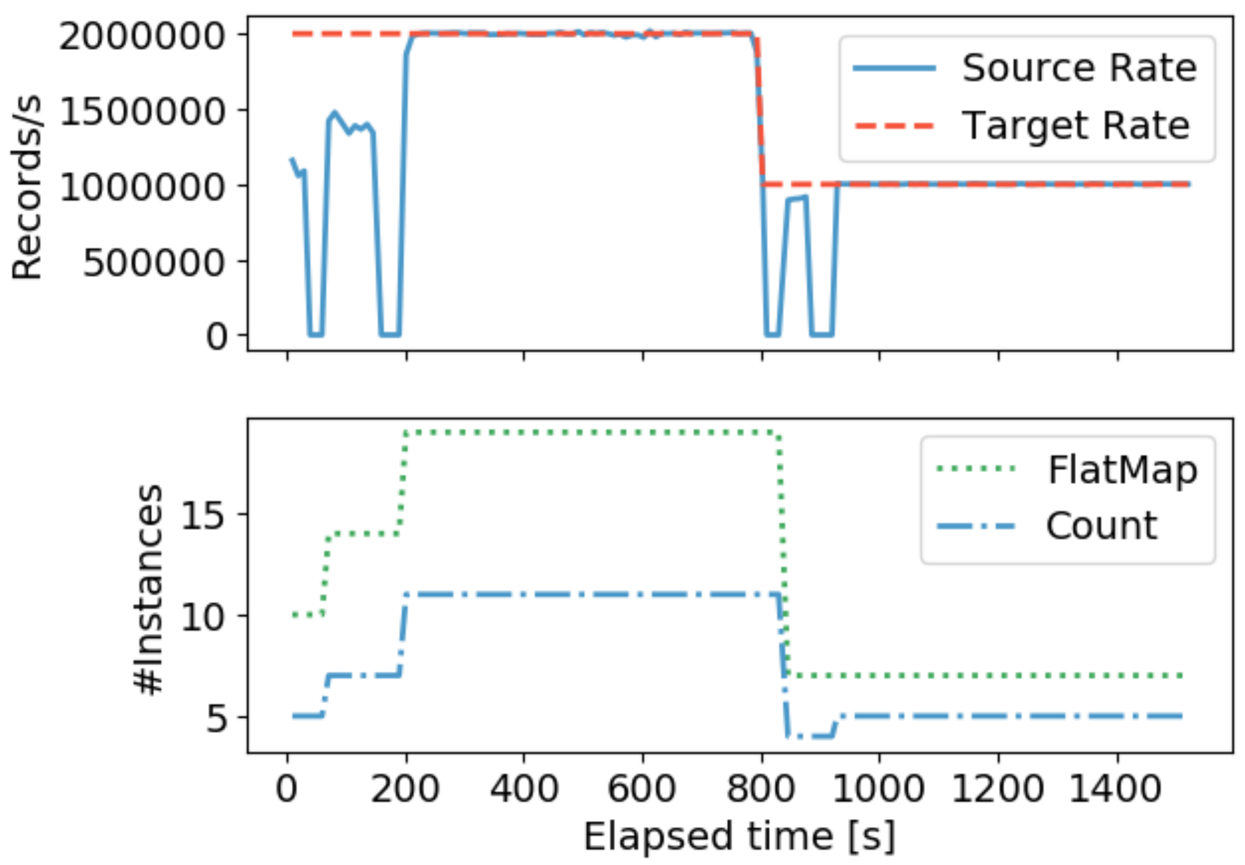
a single step for many queries and initial configurations

=> : scaling action

RECAP







Observed metrics **threshold-based policies** can lead to **oscillations**, **misconfiguration** and **slow convergence**.

DS2 uses **instrumentation** to measure **true processing** and **output rates** and estimate parallelism for all operators at once.

DS2 makes **fast** and **accurate** scaling decisions and converges in up to **three steps** even for non-linear, **complex dataflows**.

https://github.com/strymon-system/ds2

Three steps is all you need

fast, accurate, automatic scaling decisions
for distributed streaming dataflows

Vasiliki Kalavri[†], John Liagouris[†], Moritz Hoffmann[†],
Desislava Dimitrova[†], Matthew Forshaw[††], Timothy Roscoe[†]

[†]Systems Group, Department of Computer Science, ETH Zürich, firstname.lastname@inf.ethz.ch
[††]Newcastle University, firstname.lastname@newcastle.ac.uk

Support: FNSNF Fonds national suisse / Schweizerischer Nationalfonds / Fondo nazionale svizzero / Swiss National Science Foundation; Google; vmware Research