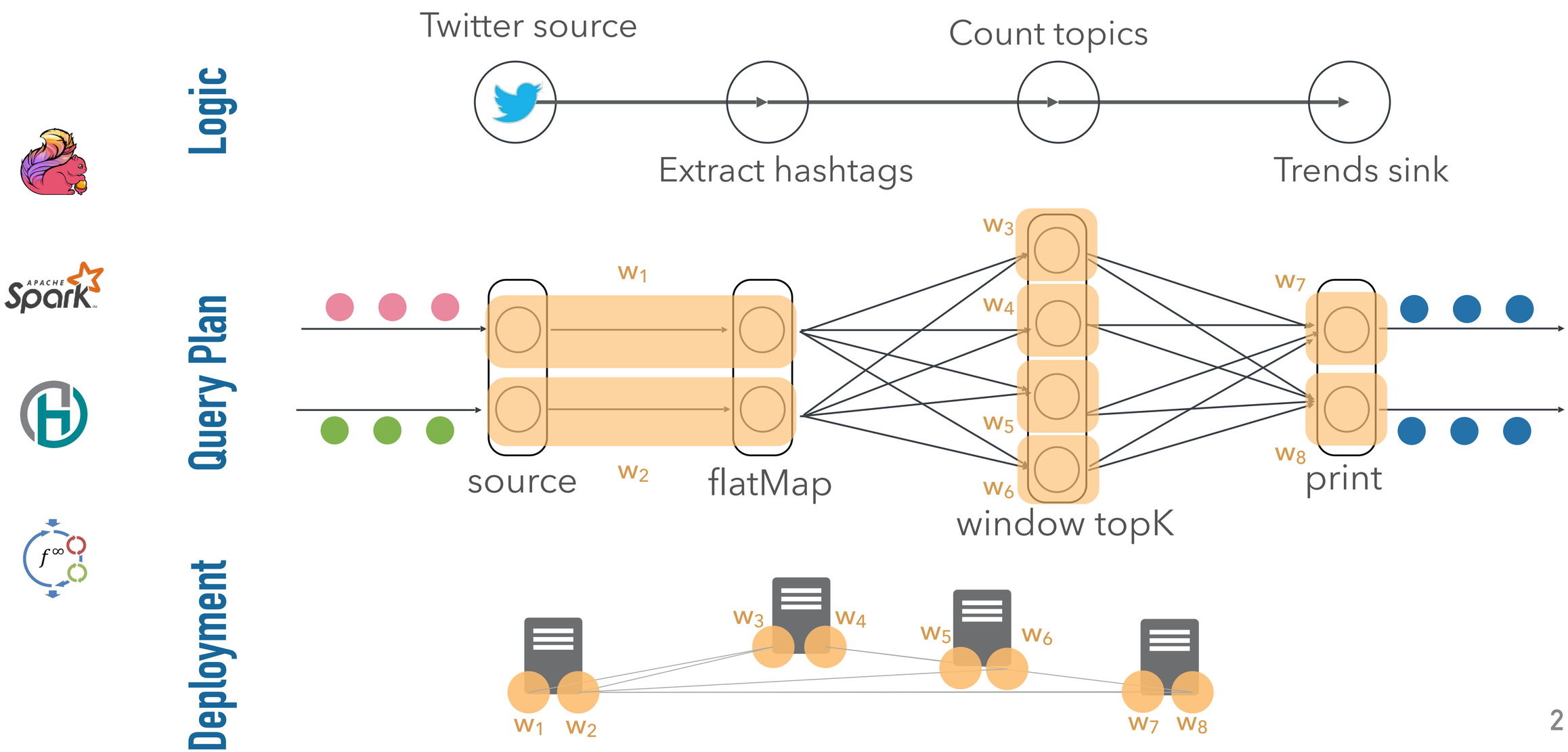
TOWARDS RECONFIGURABLE DATA STREAM PROCESSING

John Liagouris

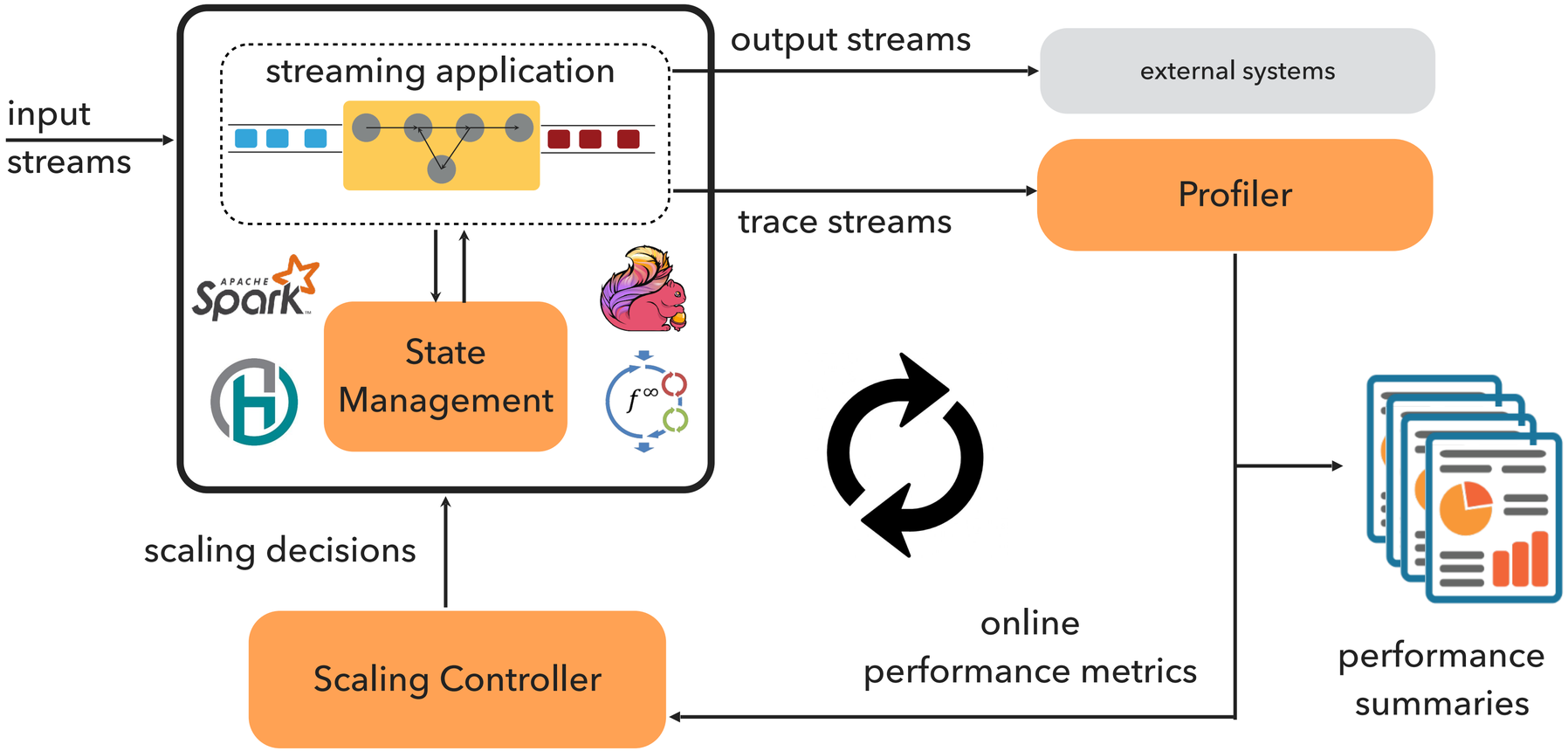
MSR Redmond

16 May 2019

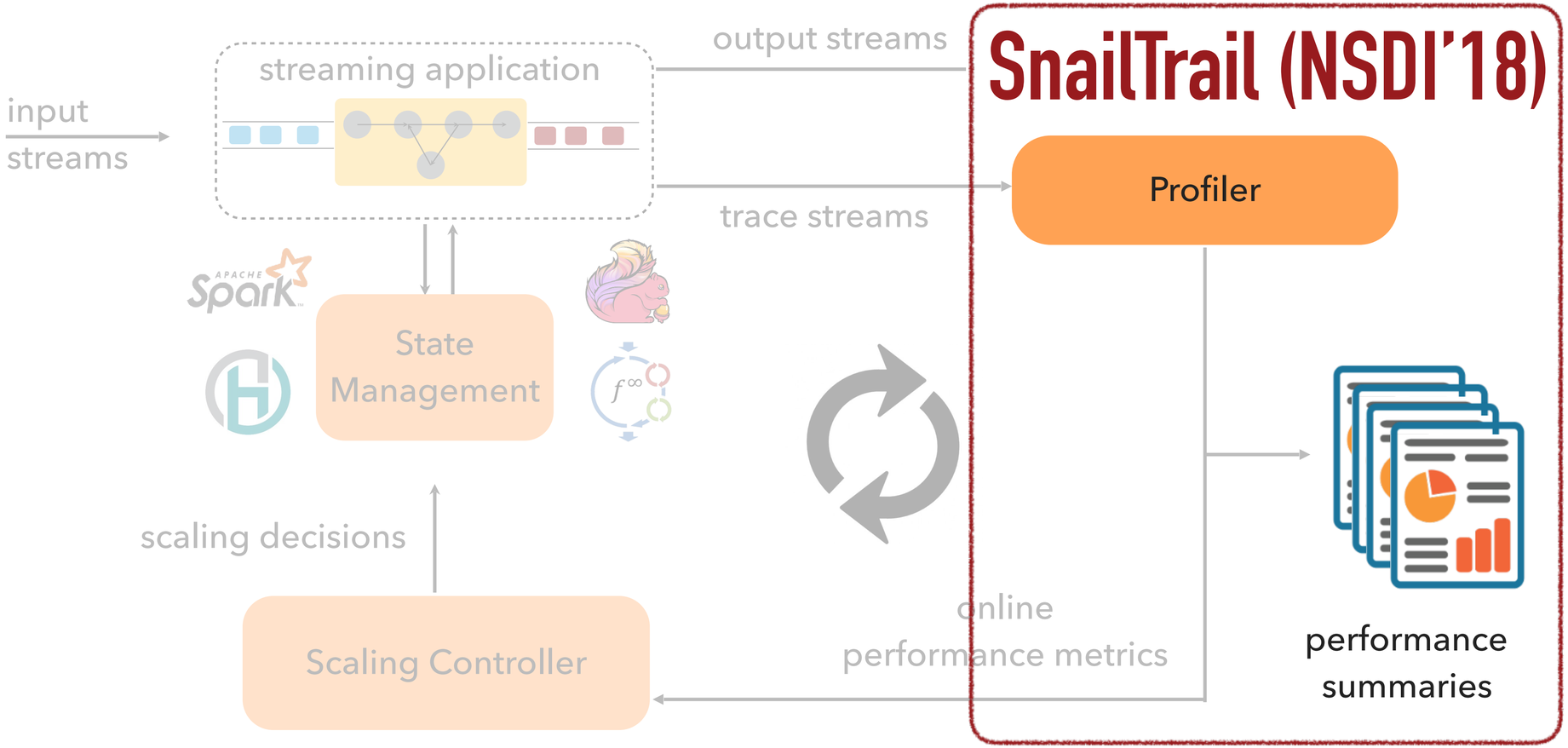
STREAMING DATAFLOWS



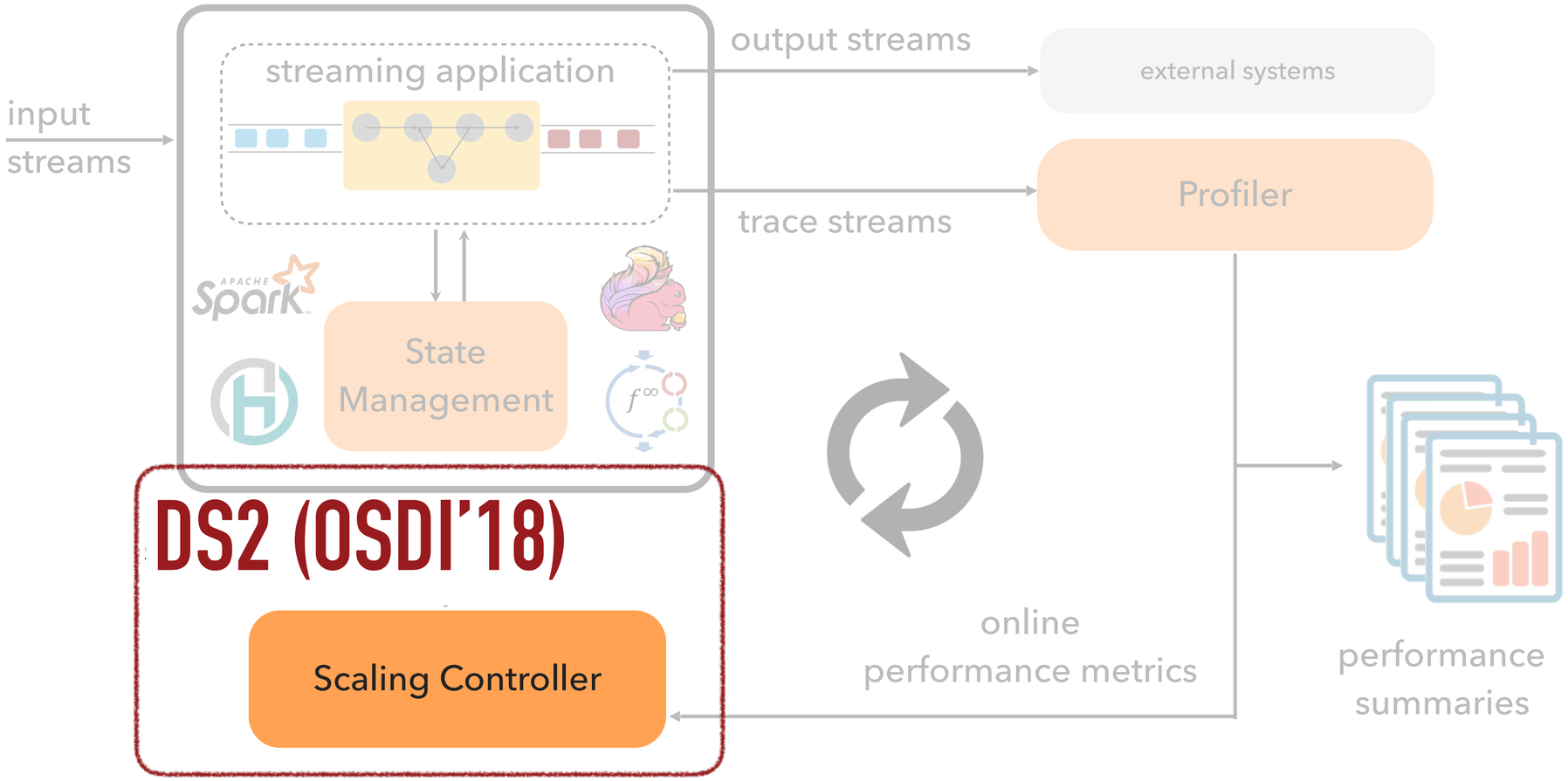
RECONFIGURABLE STREAM PROCESSING OVERVIEW



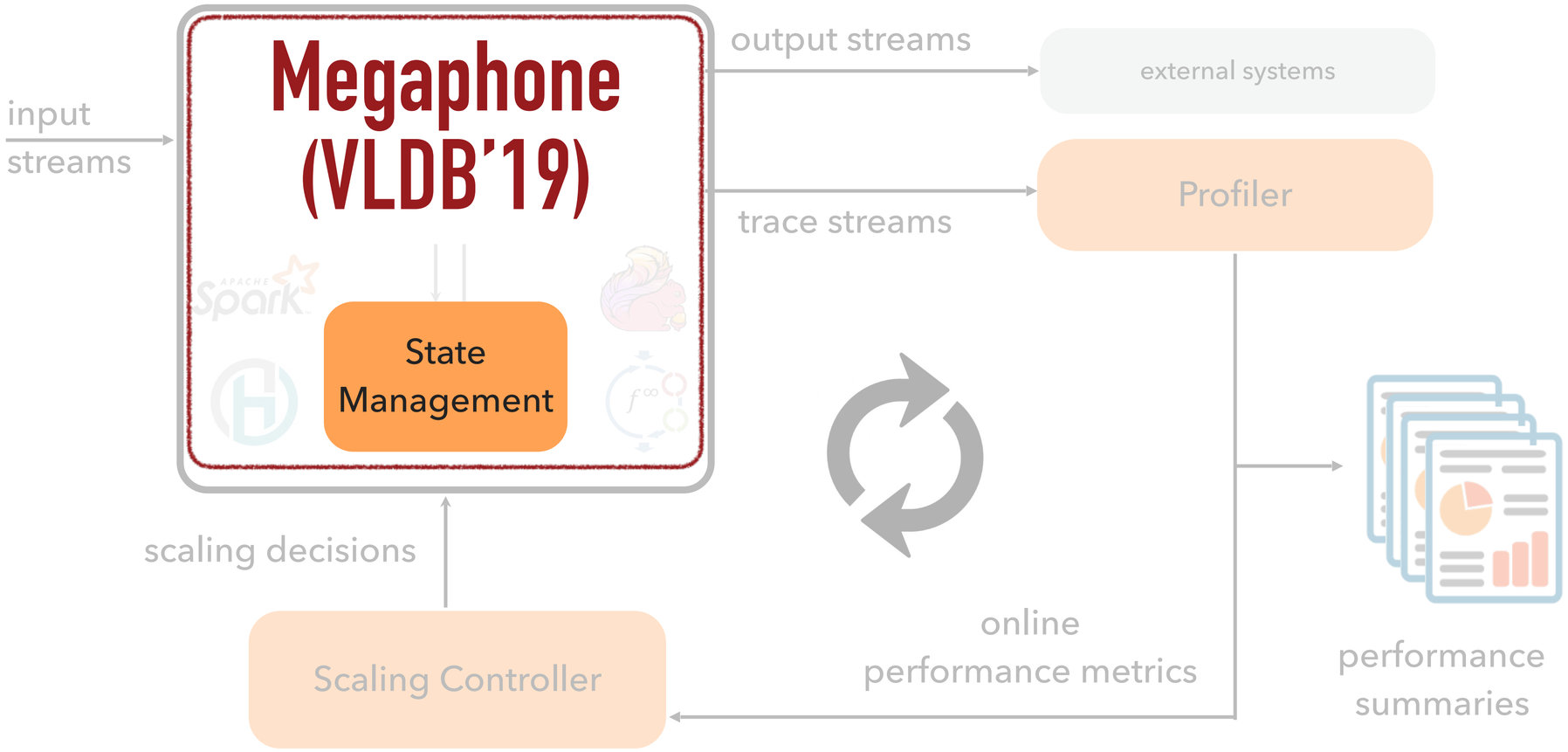
RECONFIGURABLE STREAM PROCESSING OVERVIEW



RECONFIGURABLE STREAM PROCESSING OVERVIEW



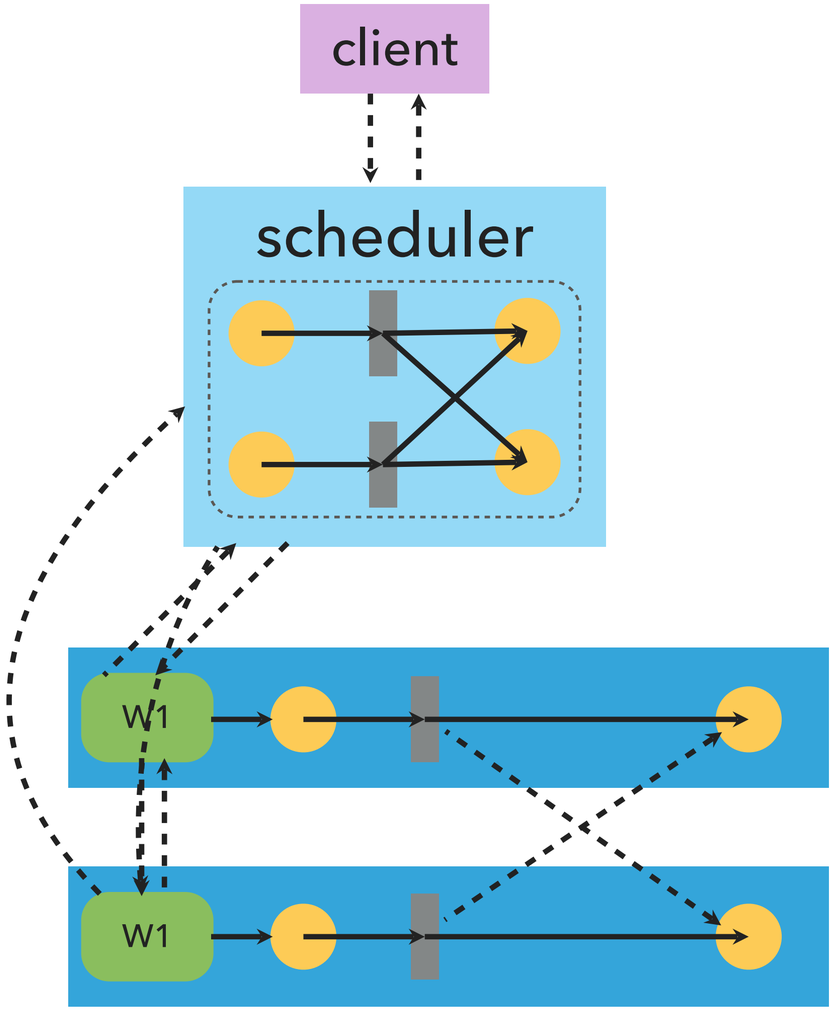
RECONFIGURABLE STREAM PROCESSING OVERVIEW



PART I

SNAILTRAIL: ONLINE CRITICAL PATH ANALYSIS FOR DISTRIBUTED STREAMING DATAFLOWS

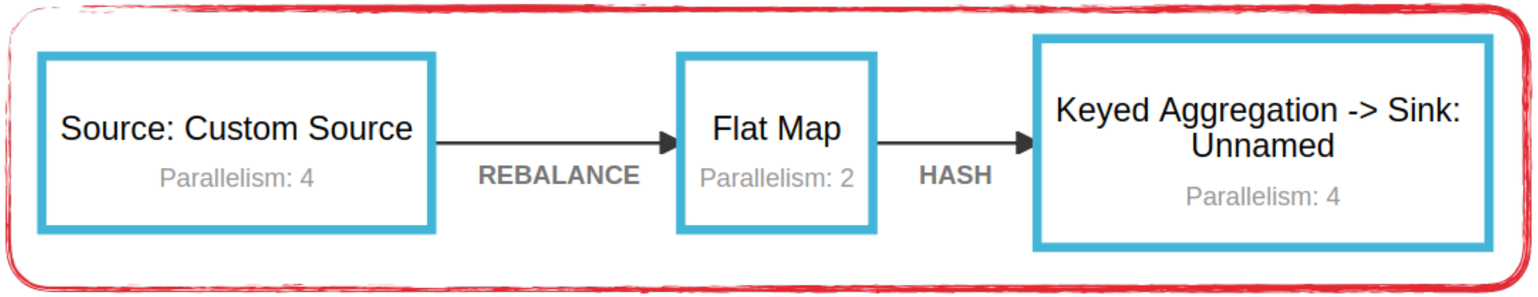
PROFILING DISTRIBUTED DATAFLOWS IS HARD



- long-running, dynamic workloads
- many tasks, activities, operators, dependencies
- bottleneck causes are usually not isolated but span multiple processes

PERFORMANCE METRICS IN EXISTING SYSTEMS (FLINK)

Dataflow graph



Custom aggregate metrics

Duration

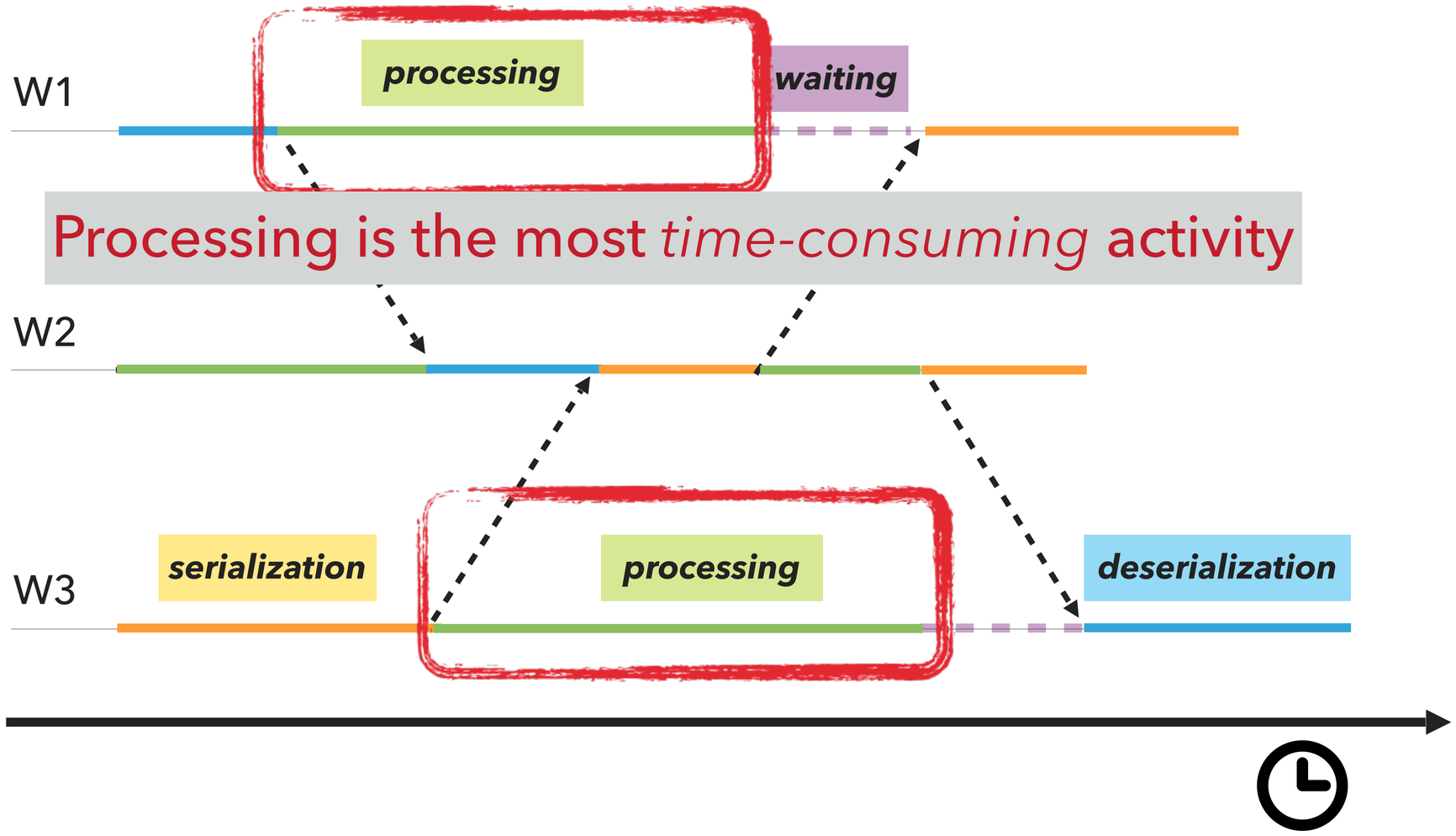
Aggregate data exchange

Subtasks | TaskManagers | Metrics | Accumulators | Checkpoints | Back Pressure

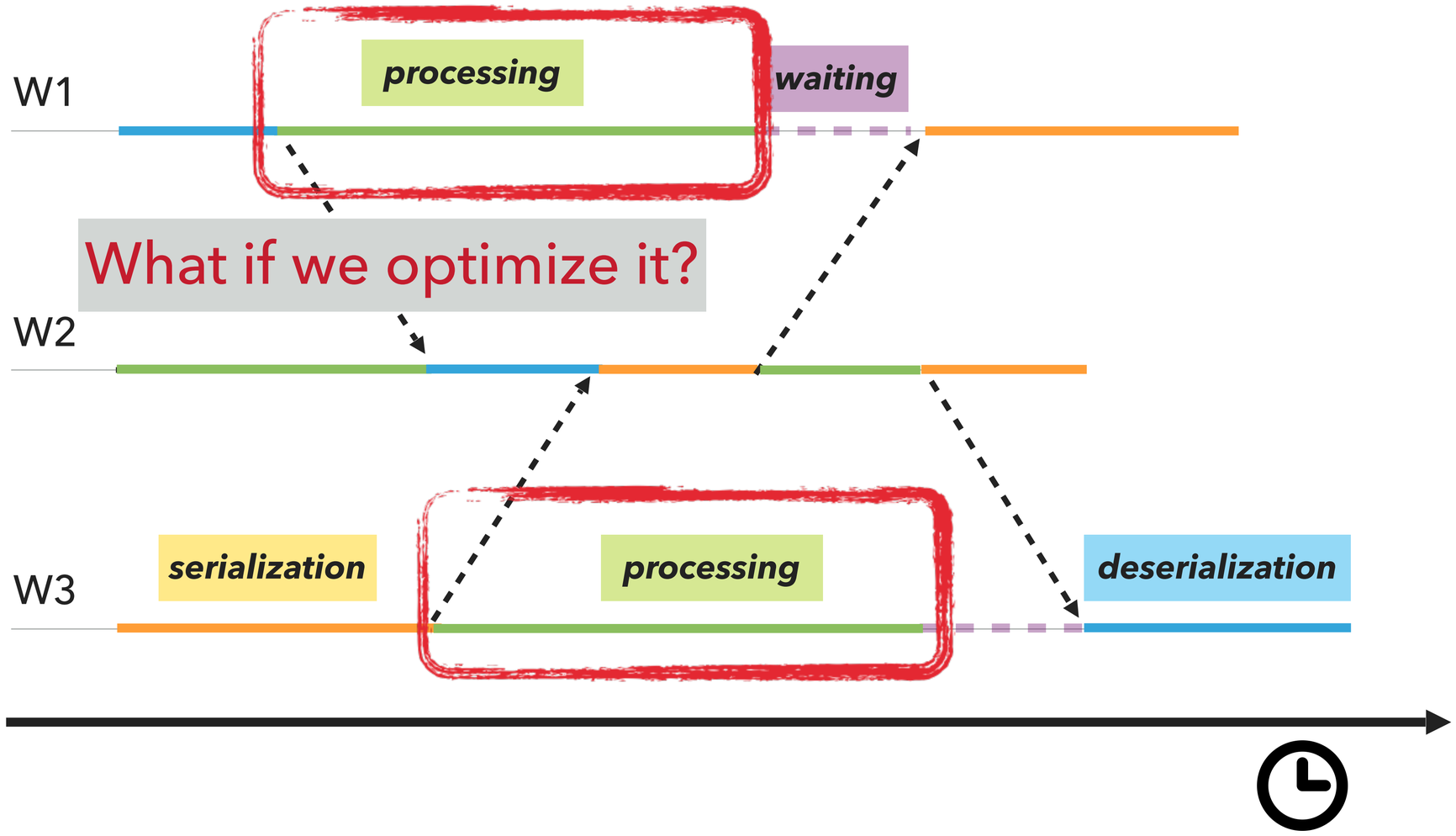
Start Time	End Time	Duration	Name	Bytes received	Records received	Bytes sent	Records sent	Parallelism	Tasks	Status
2017-10-19, 17:45:43	2017-10-19, 17:47:42	1m 58s	Source: Custom Source	0 B	0	2.56 GB	12,942,429	4	0 4 0 0 0 0	RUNNING
2017-10-19, 17:45:43	2017-10-19, 17:47:42	1m 58s	Flat Map	2.54 GB	12,844,494	4.59 GB	254,656,192	2	0 2 0 0 0 0	RUNNING
2017-10-19, 17:45:43	2017-10-19, 17:47:42	1m 58s	Keyed Aggregation -> Sink: Unnamed	4.59 GB	254,646,032	0 B	0	4	0 4 0 0 0 0	RUNNING

Start Time	End Time	Duration	Bytes received	Records received	Bytes sent	Records sent	Attempt	Host	Status
2017-10-19, 17:45:43		1m 58s	1.26 GB	6,401,851	2.29 GB	126,924,569	1	kalamari:43402	RUNNING
2017-10-19, 17:45:43		1m 58s	1.27 GB	6,442,643	2.30 GB	127,731,623	1	kalamari:43402	RUNNING

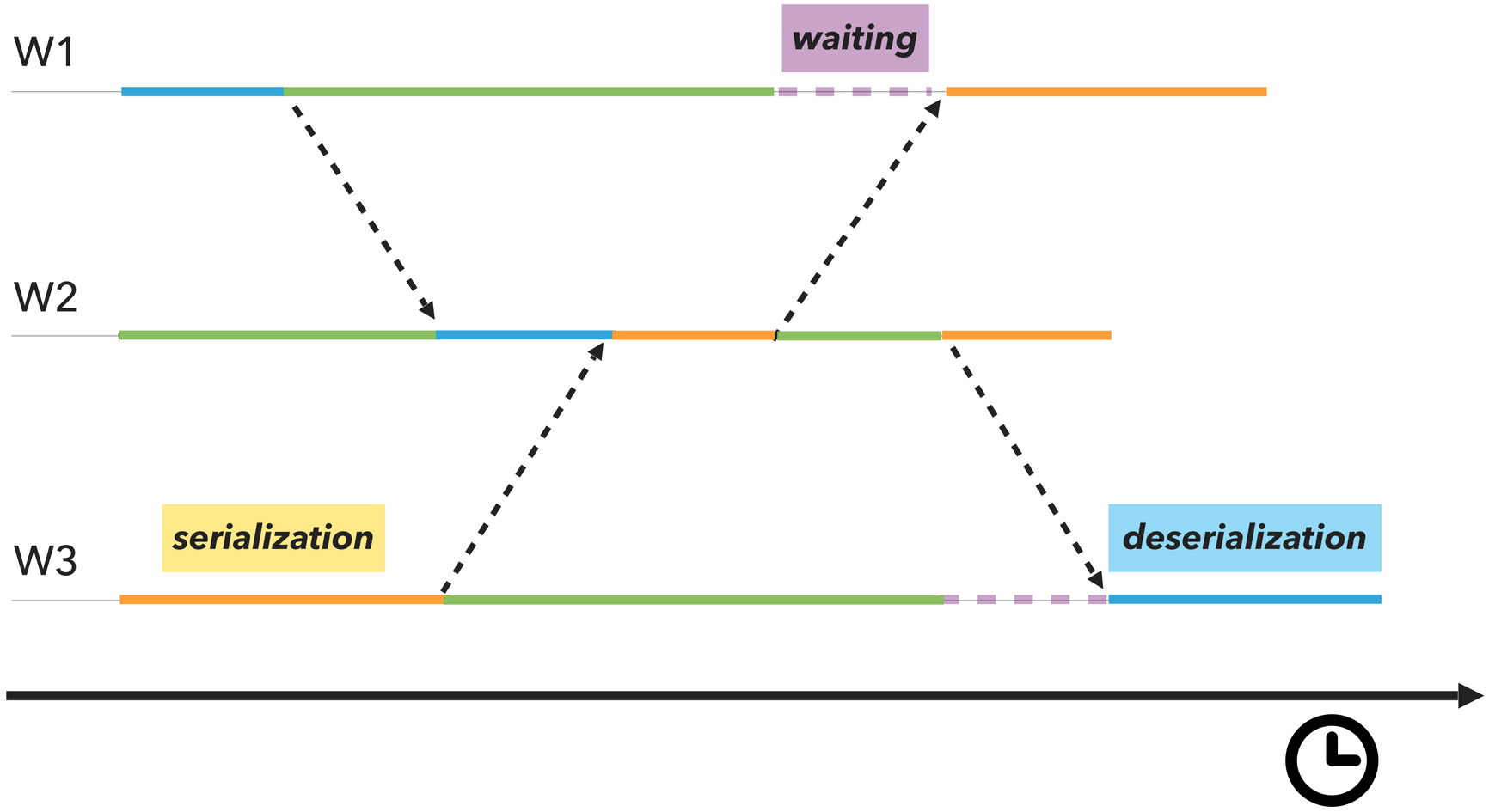
A PARALLEL EXECUTION



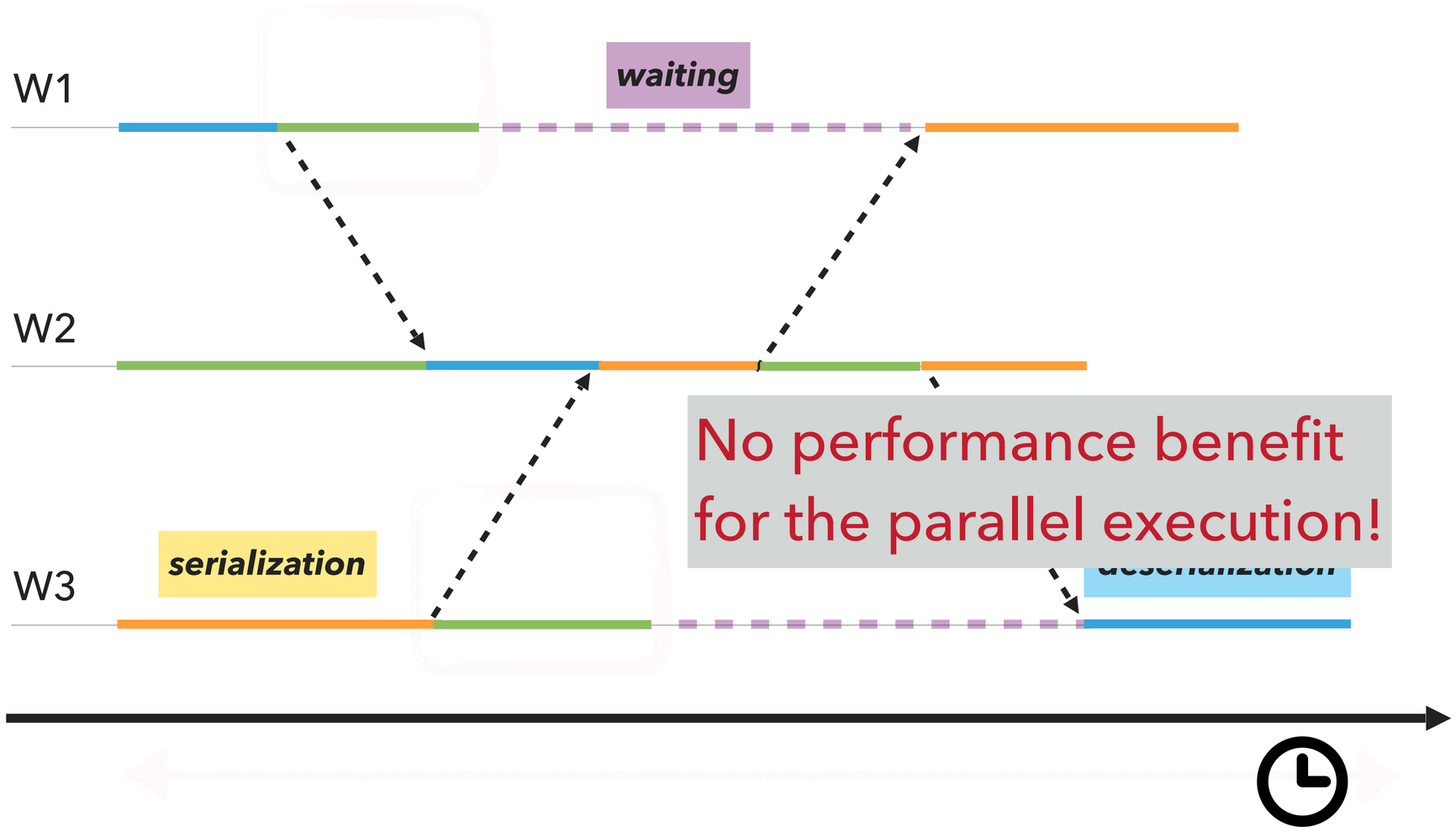
A PARALLEL EXECUTION



A PARALLEL EXECUTION



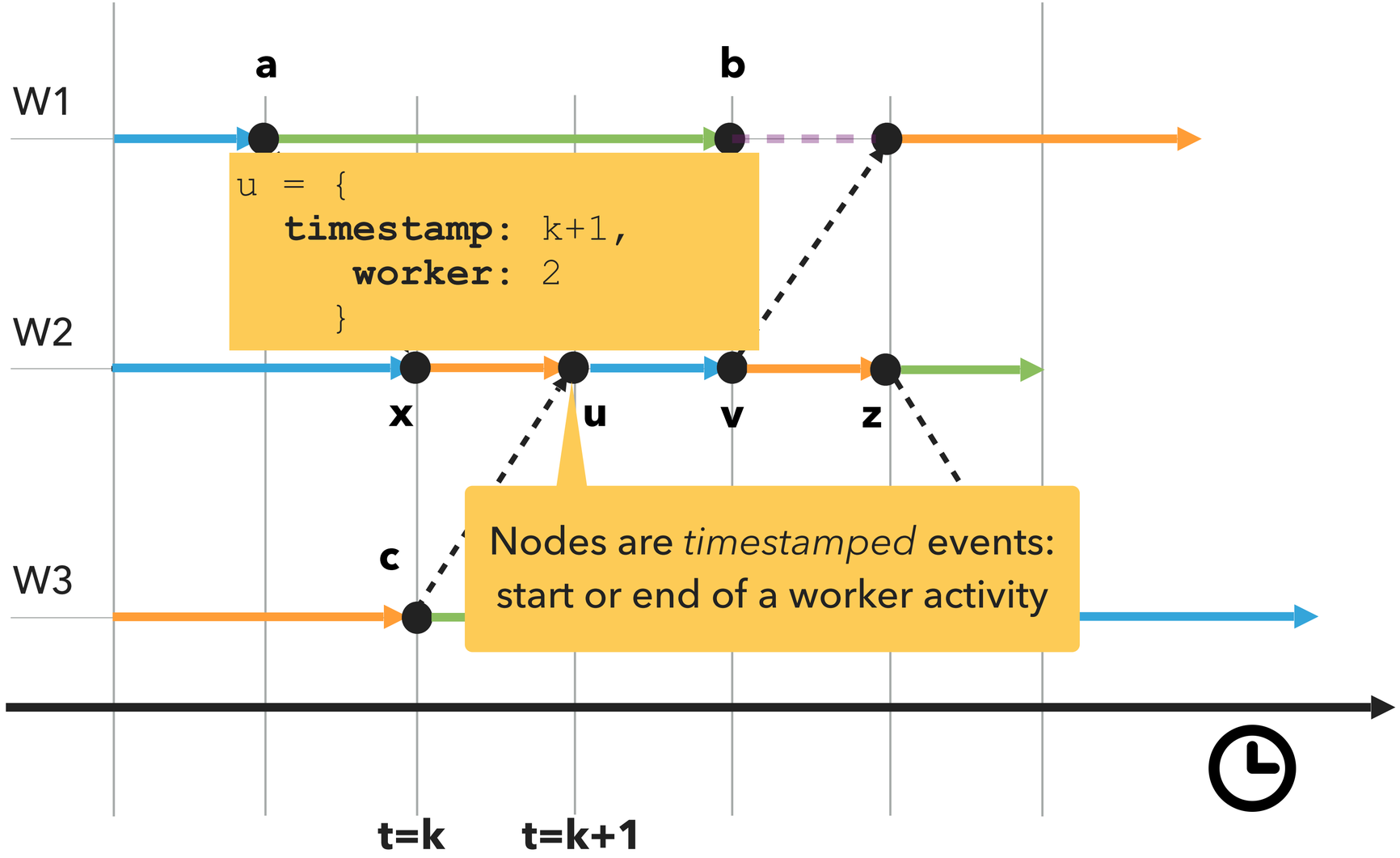
A PARALLEL EXECUTION



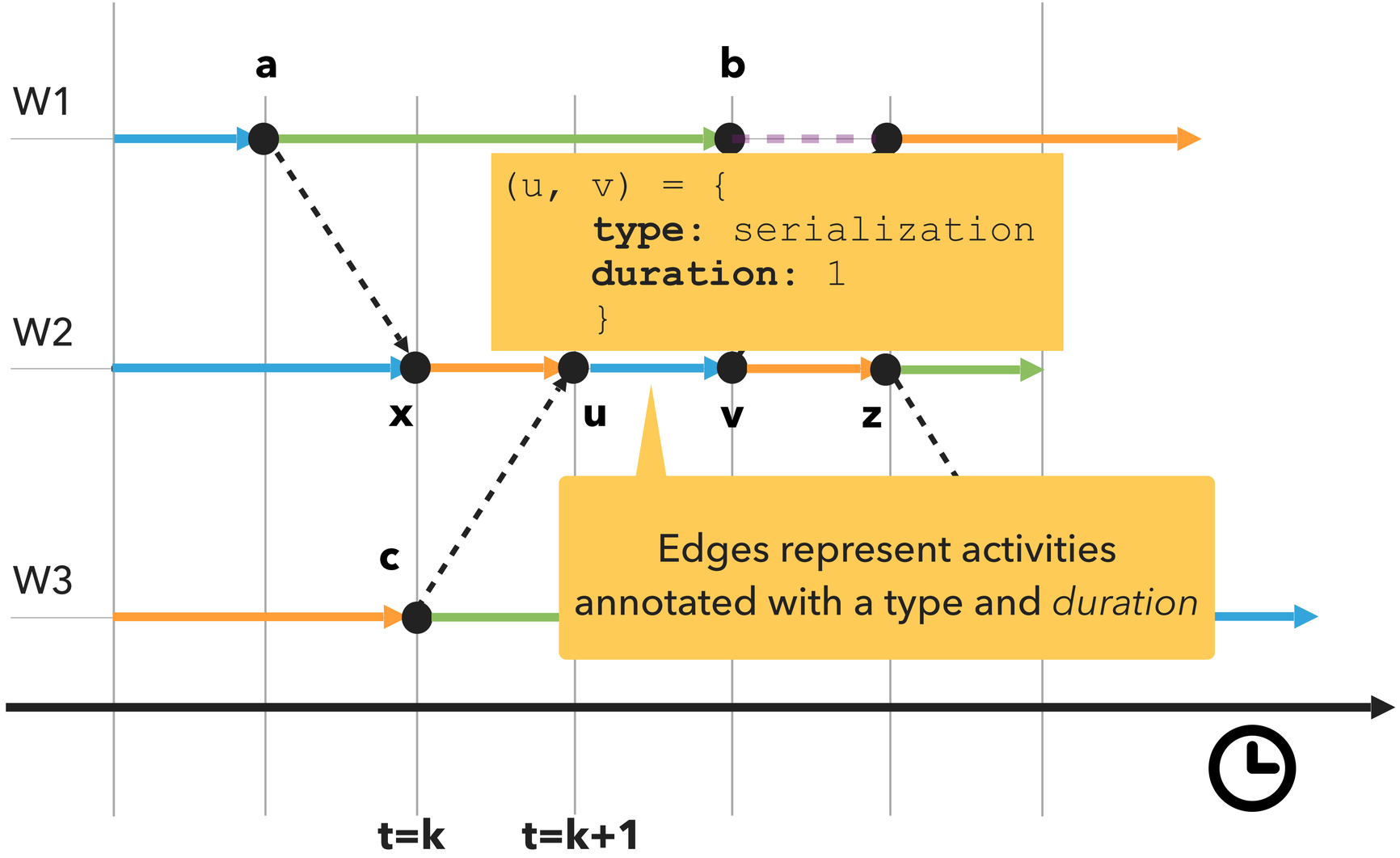
A PARALLEL EXECUTION

conventional profiling can be misleading

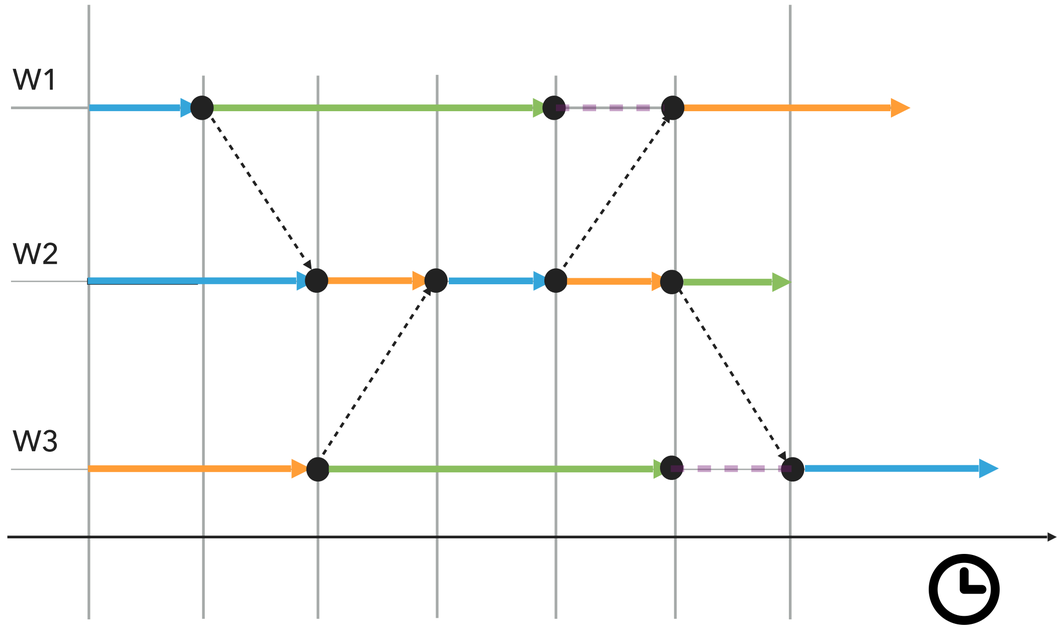
THE PROGRAM ACTIVITY GRAPH (PAG)



THE PROGRAM ACTIVITY GRAPH (PAG)

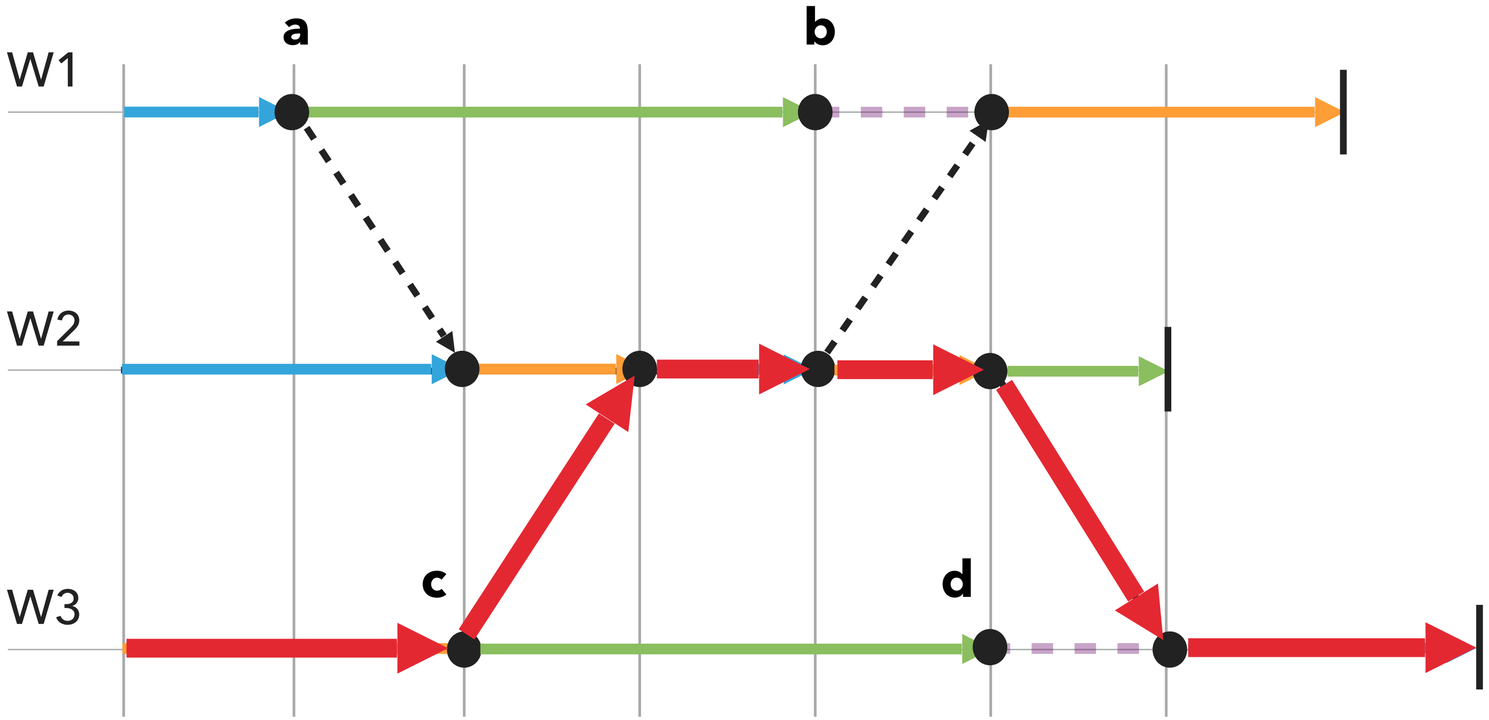


The Program Activity Graph captures computational dependencies among parallel workers



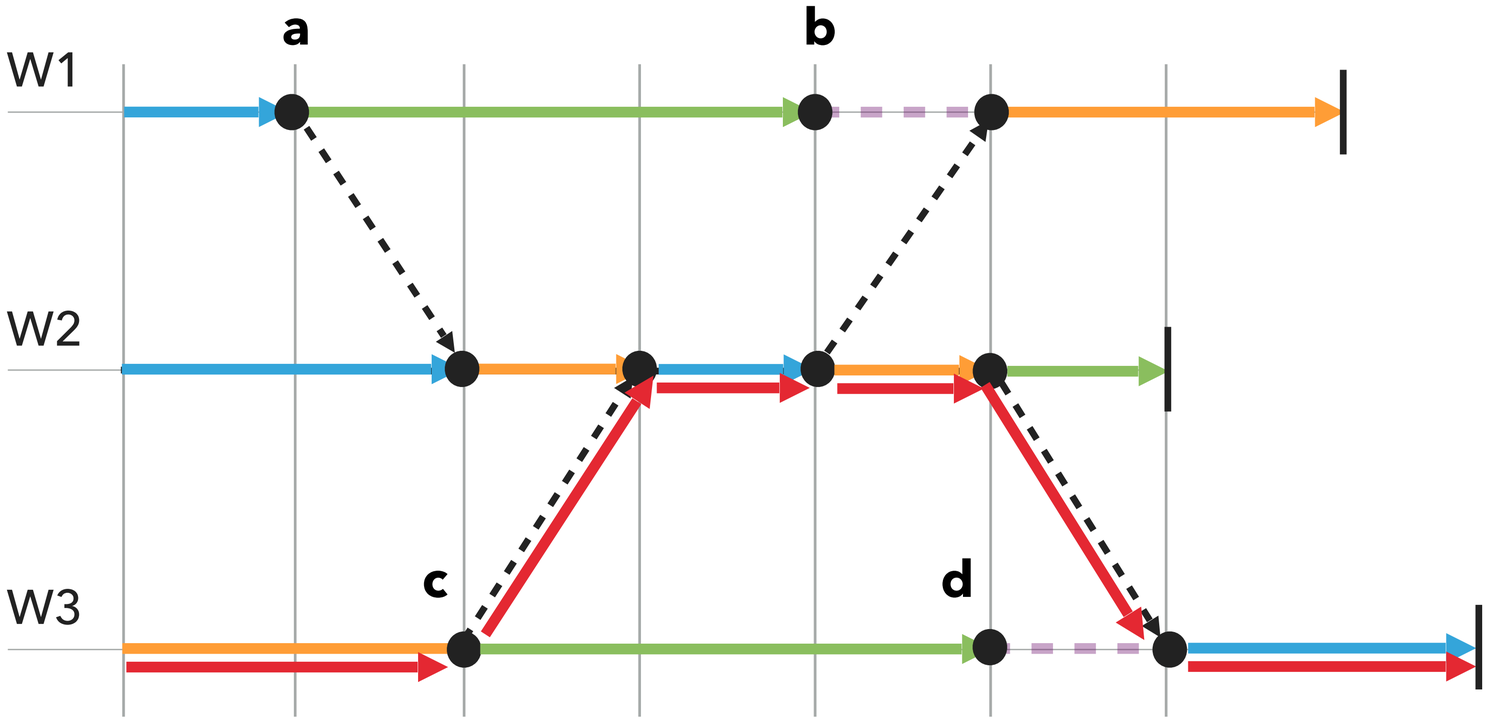
- Which activities delay the overall execution?
- i.e. which activities lie on the critical path of the execution?

CRITICAL PATH



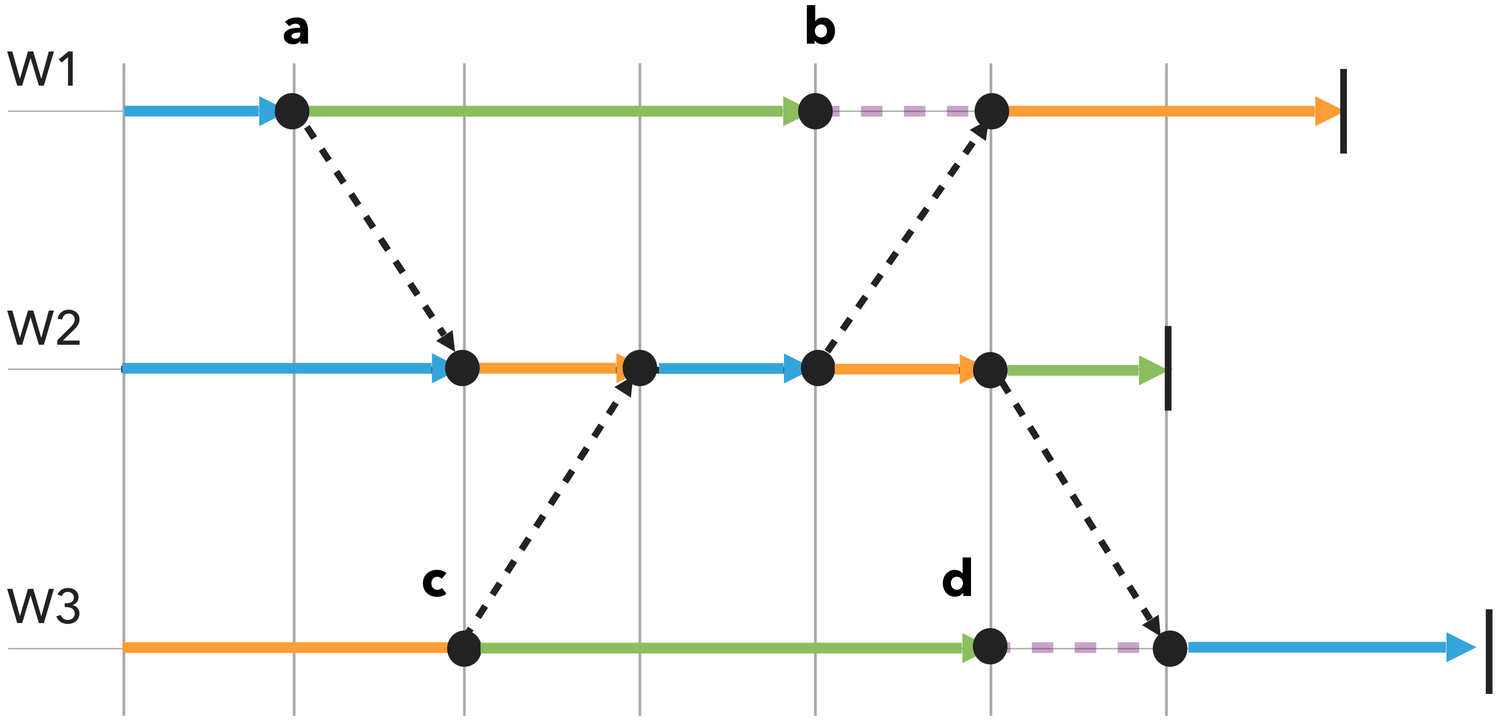
The longest path in the execution history
(not considering waiting activities)

CRITICAL PATH

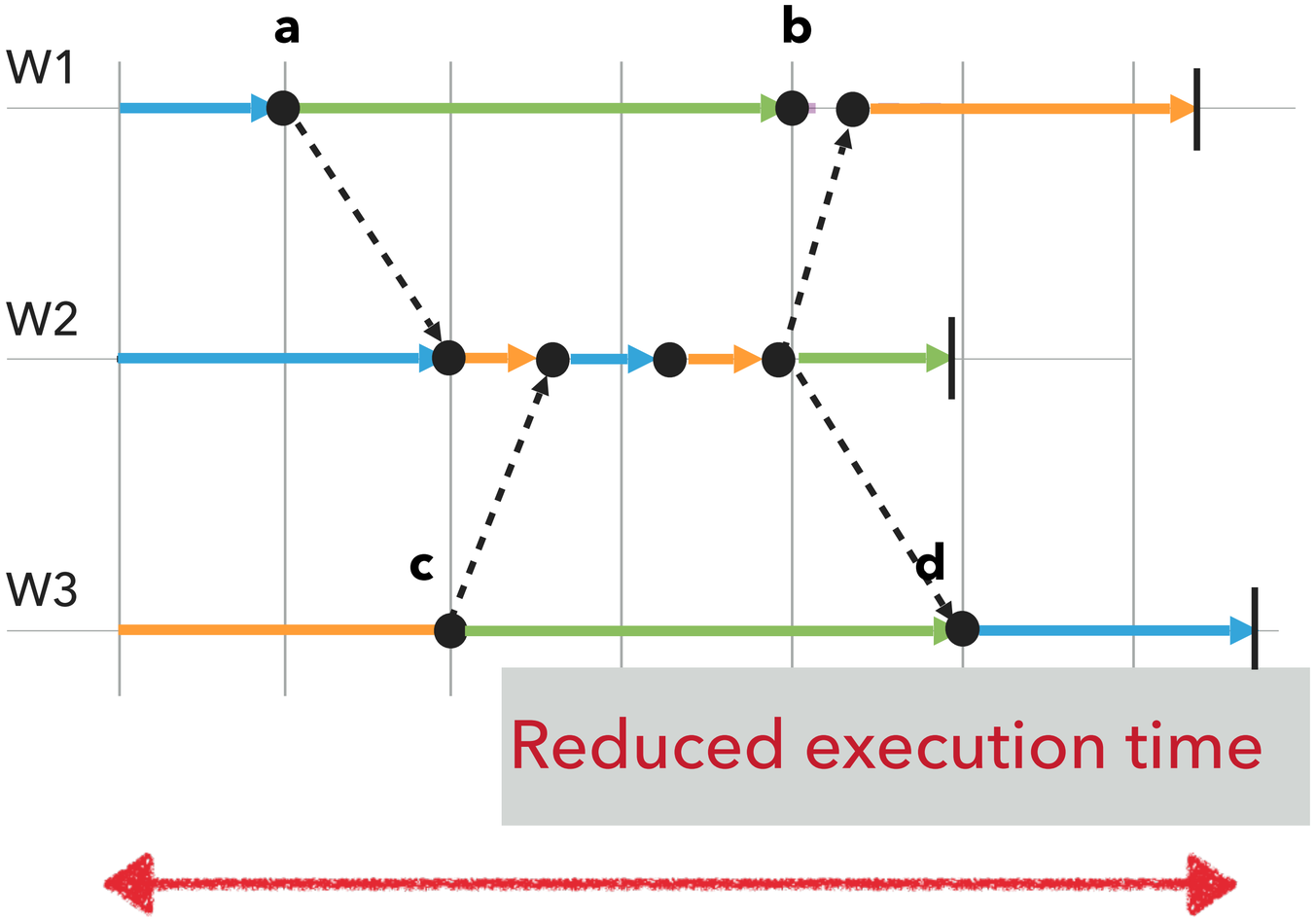


Optimizing activities on the critical path can potentially reduce the execution time

CRITICAL PATH

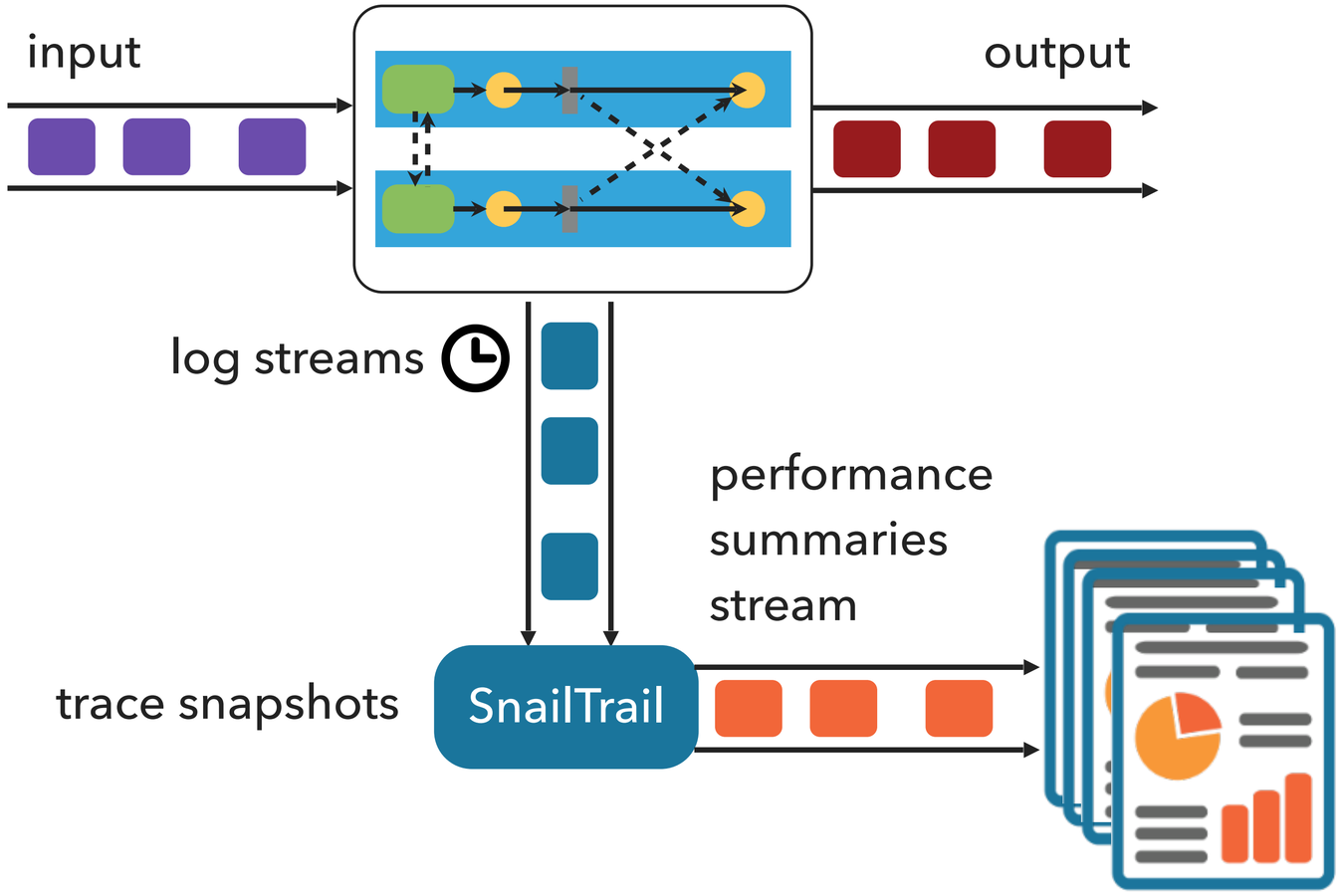


CRITICAL PATH

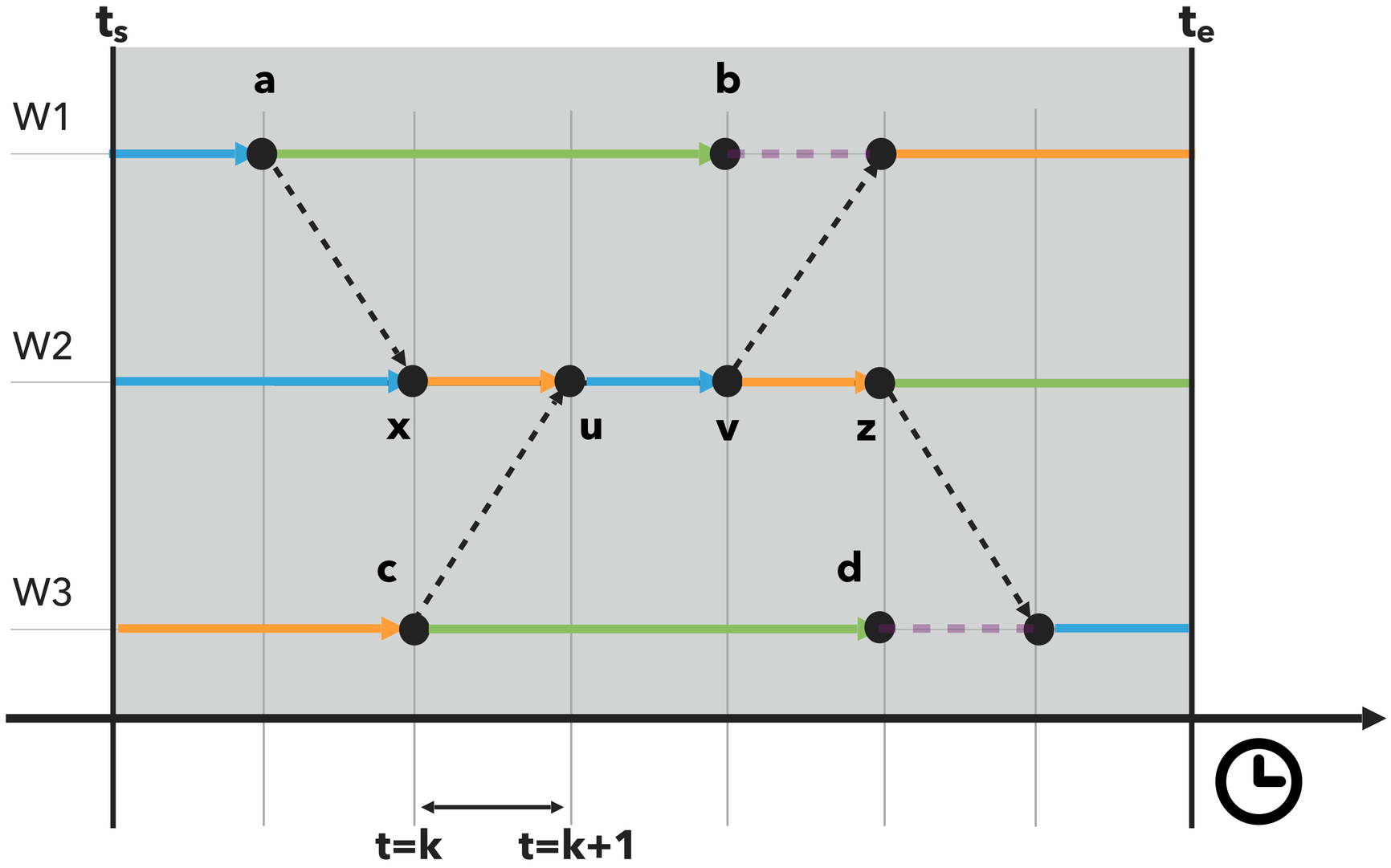


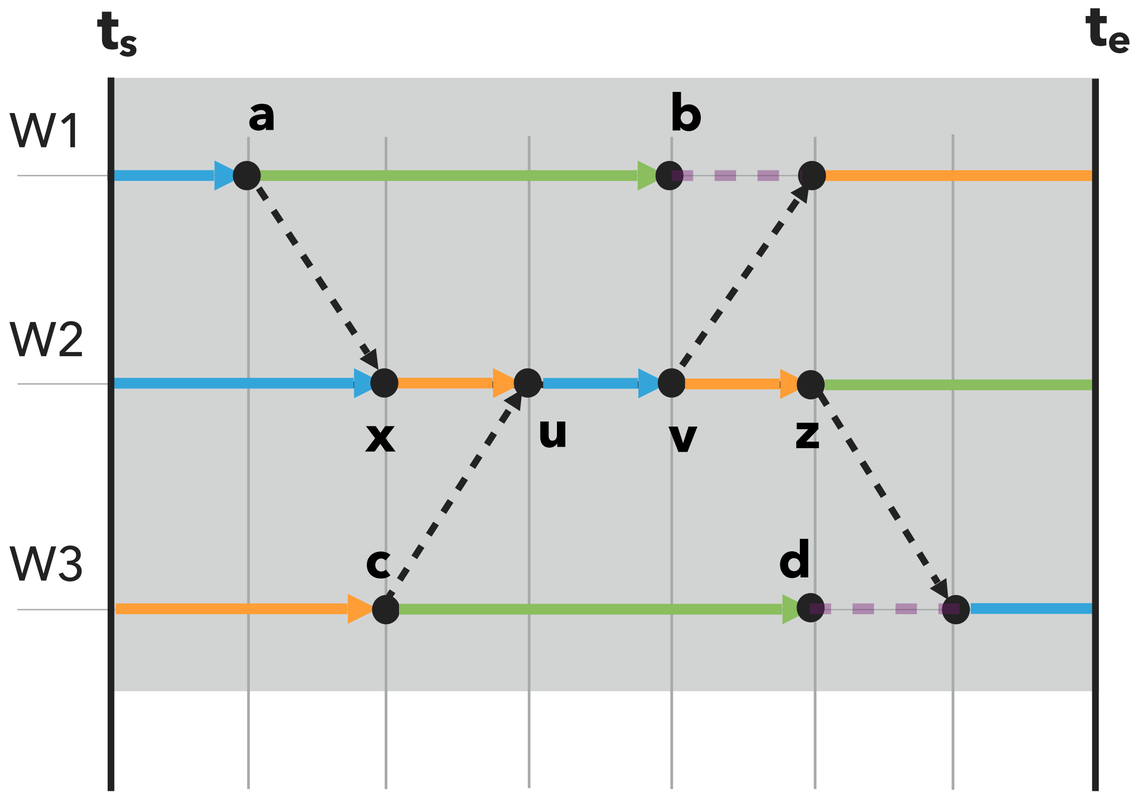
How to efficiently compute the critical path for continuously running, distributed dataflow applications with potentially unbounded input?

CRITICAL PATH ANALYSIS ON TRACE SNAPSHOTS

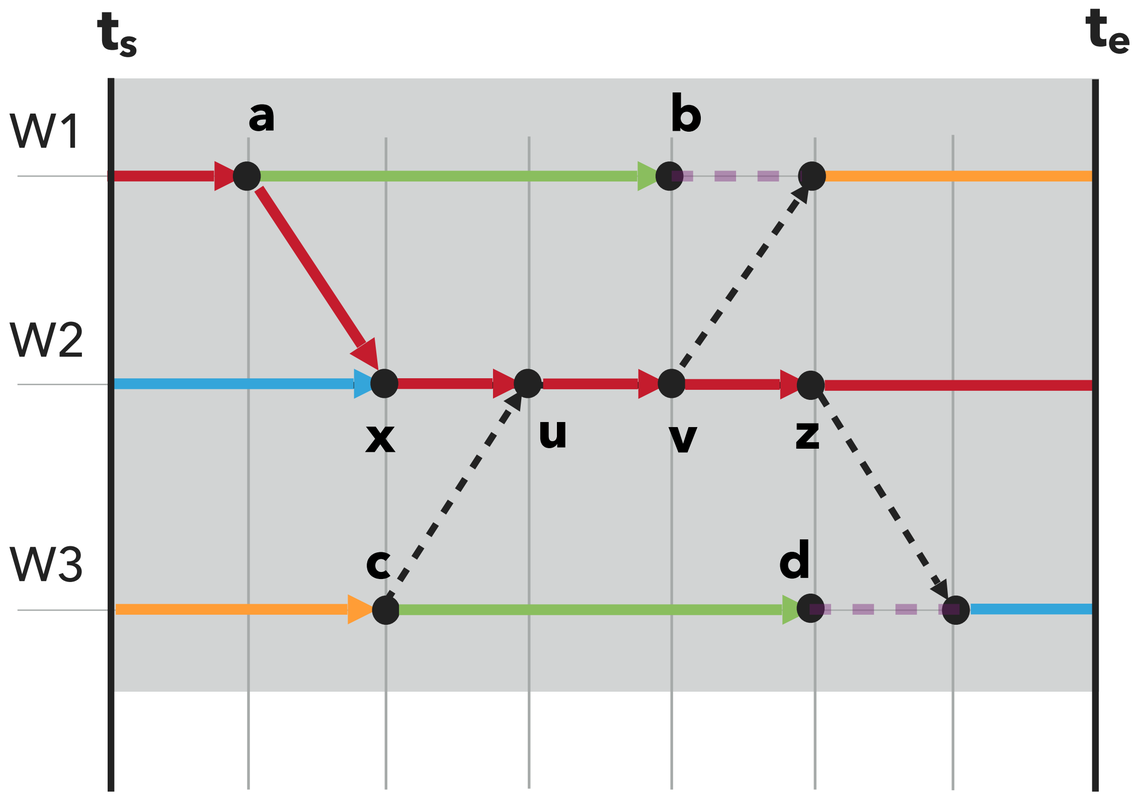


PROGRAM ACTIVITY GRAPH SNAPSHOT

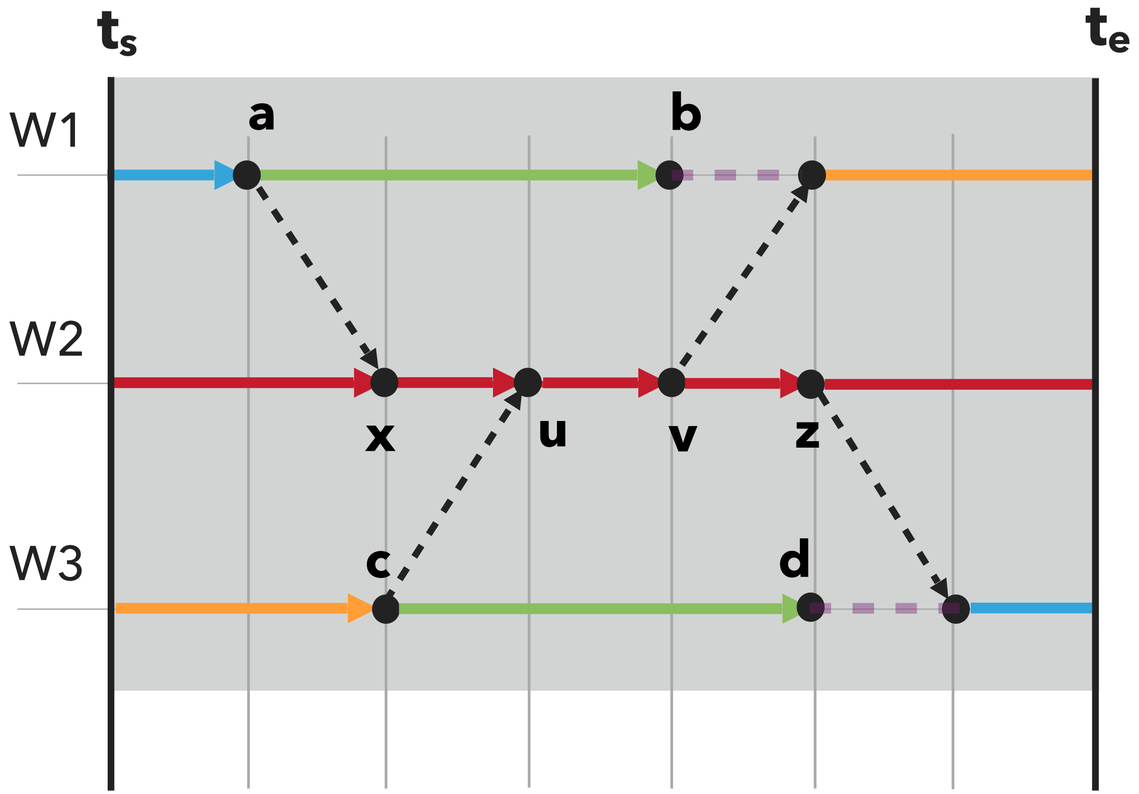




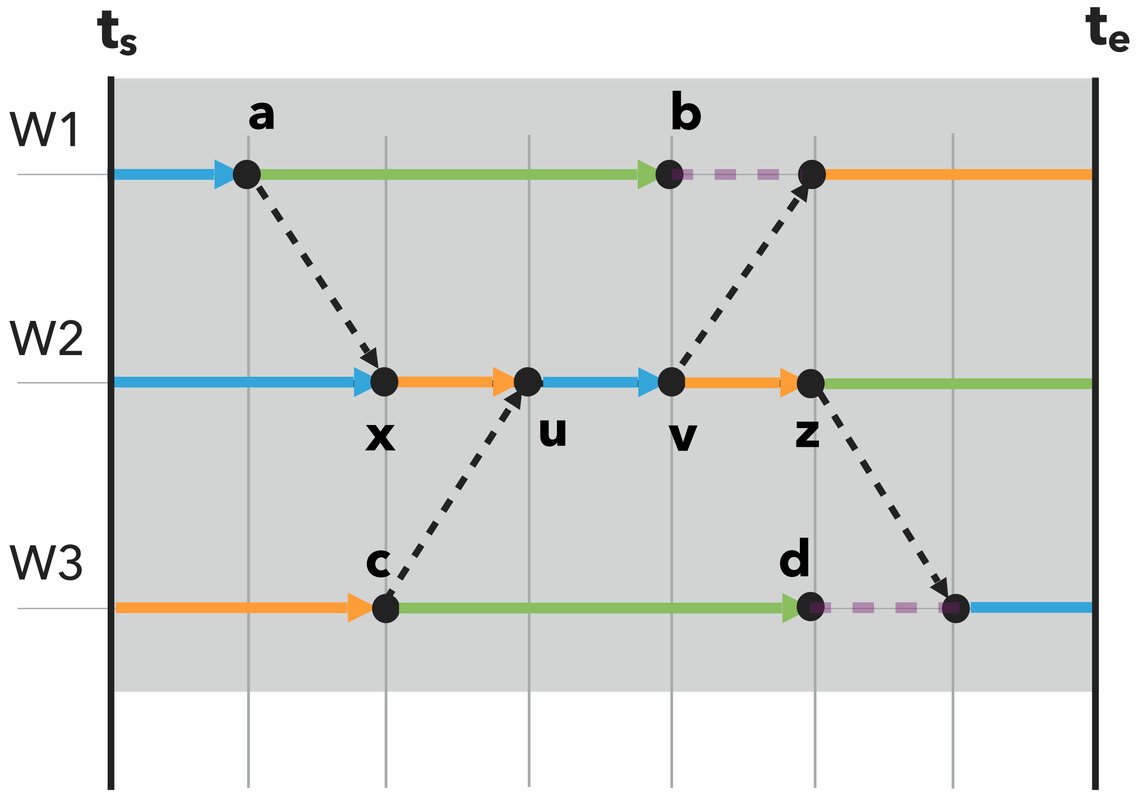
- All paths have the same length: $t_e - t_s$



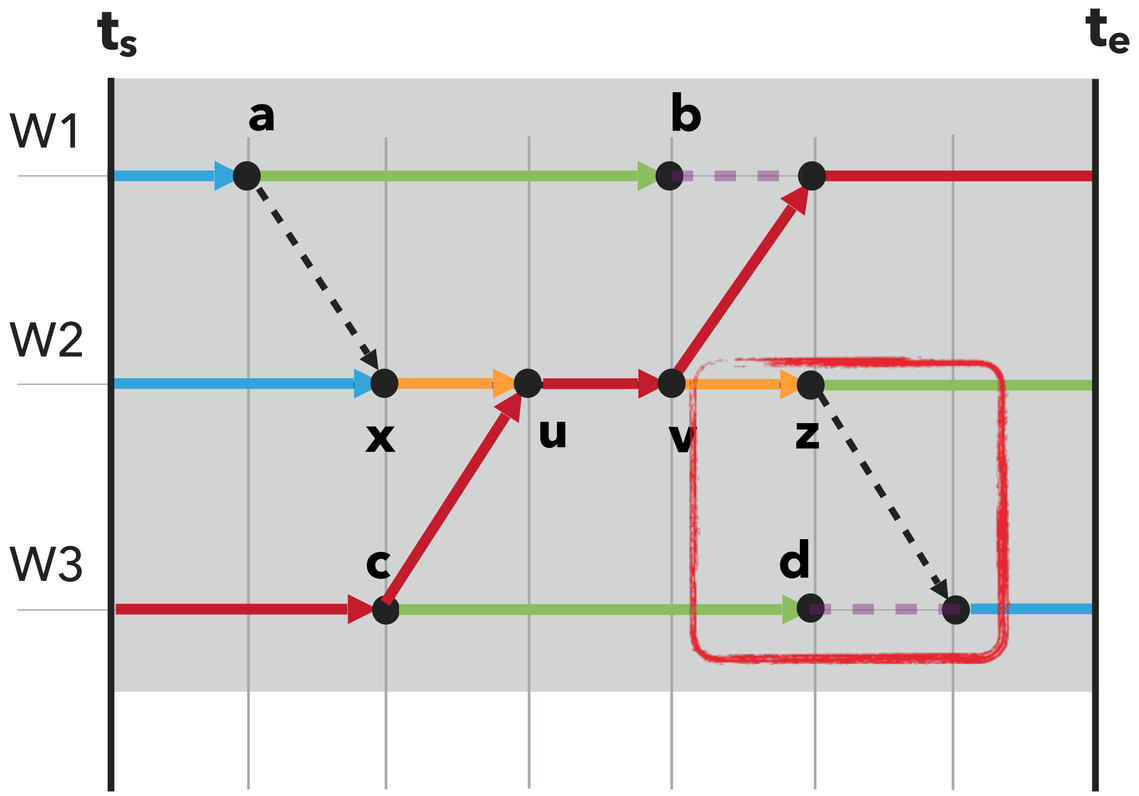
- All paths have the same length: $t_e - t_s$



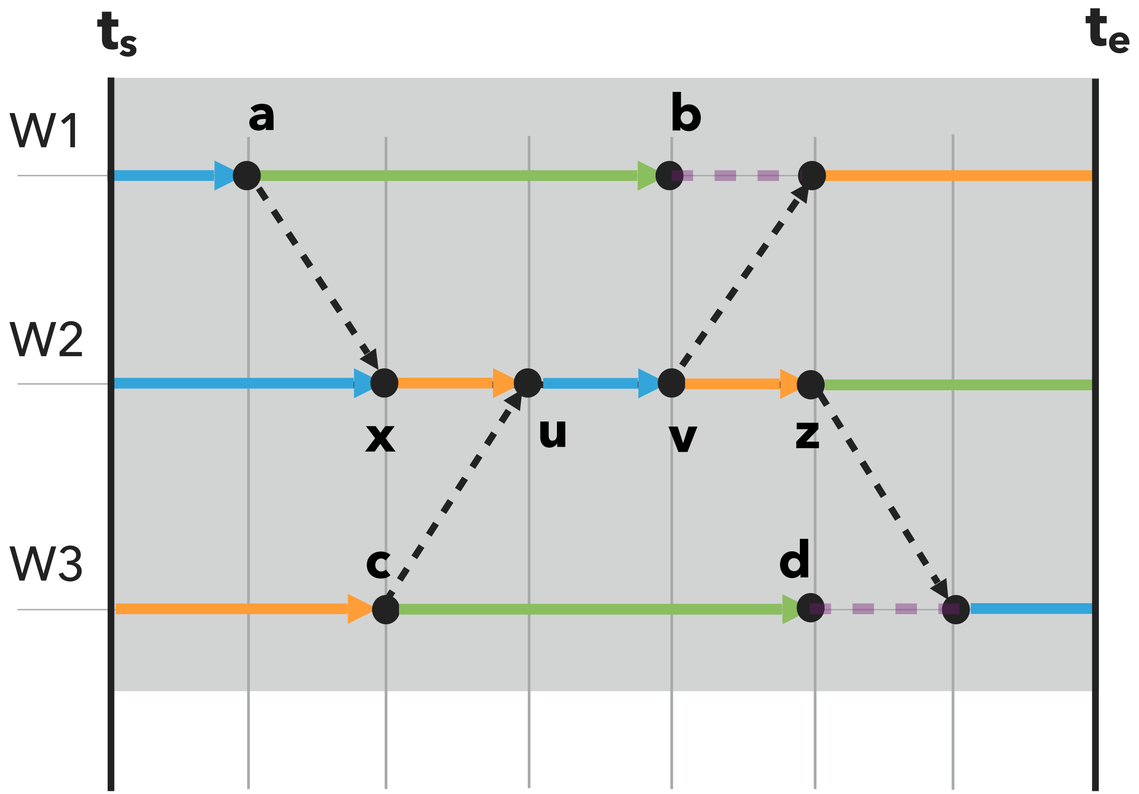
- All paths have the same length: $t_e - t_s$



- All paths have the same length: $t_e - t_s$
- Choosing a random path might miss critical activities

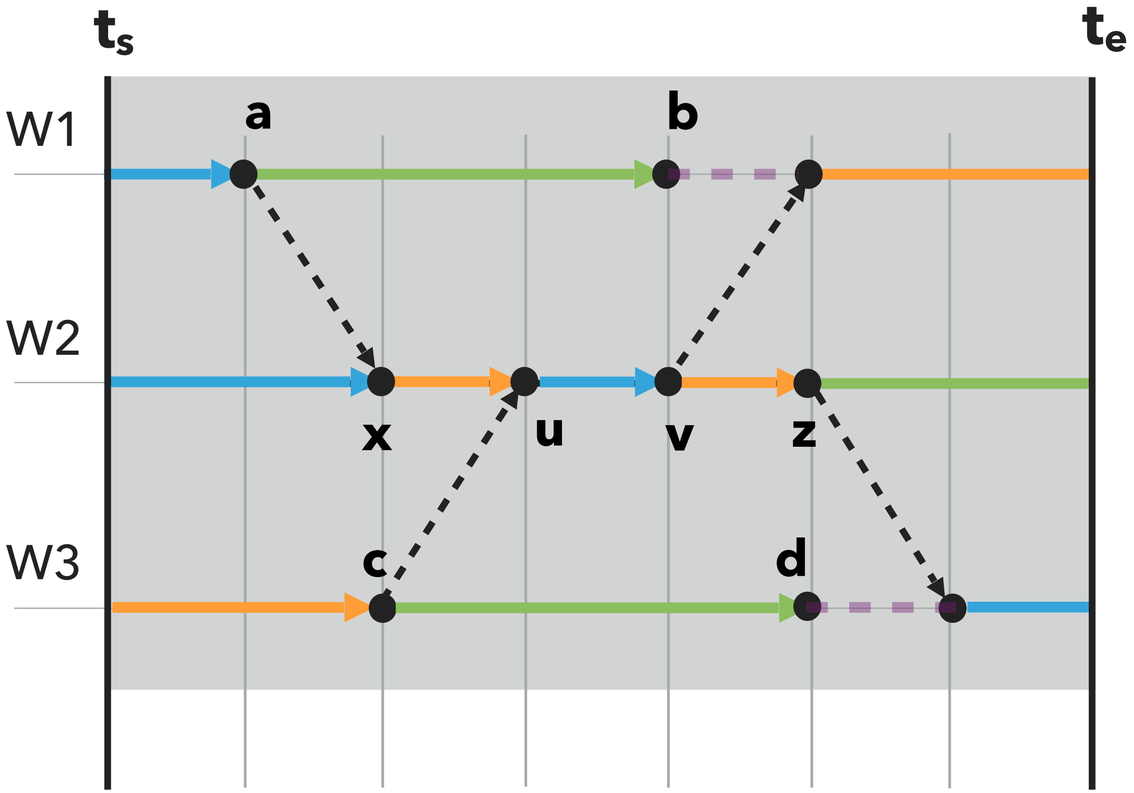


- All paths have the same length: $t_e - t_s$
- Choosing a random path might miss critical activities



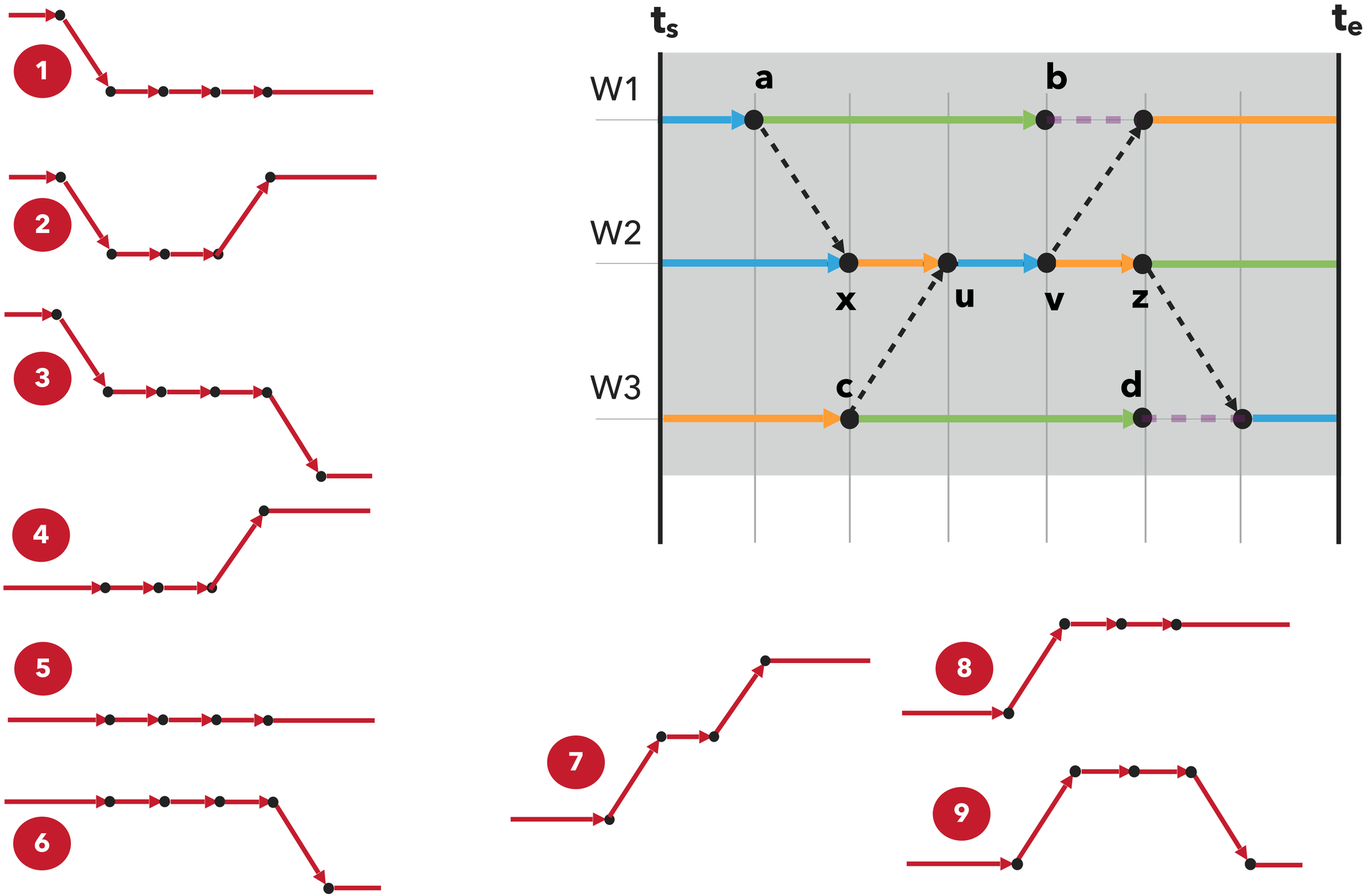
- All paths have the same length: $t_e - t_s$
- Choosing a random path might miss critical activities
- Enumerating all paths is impractical

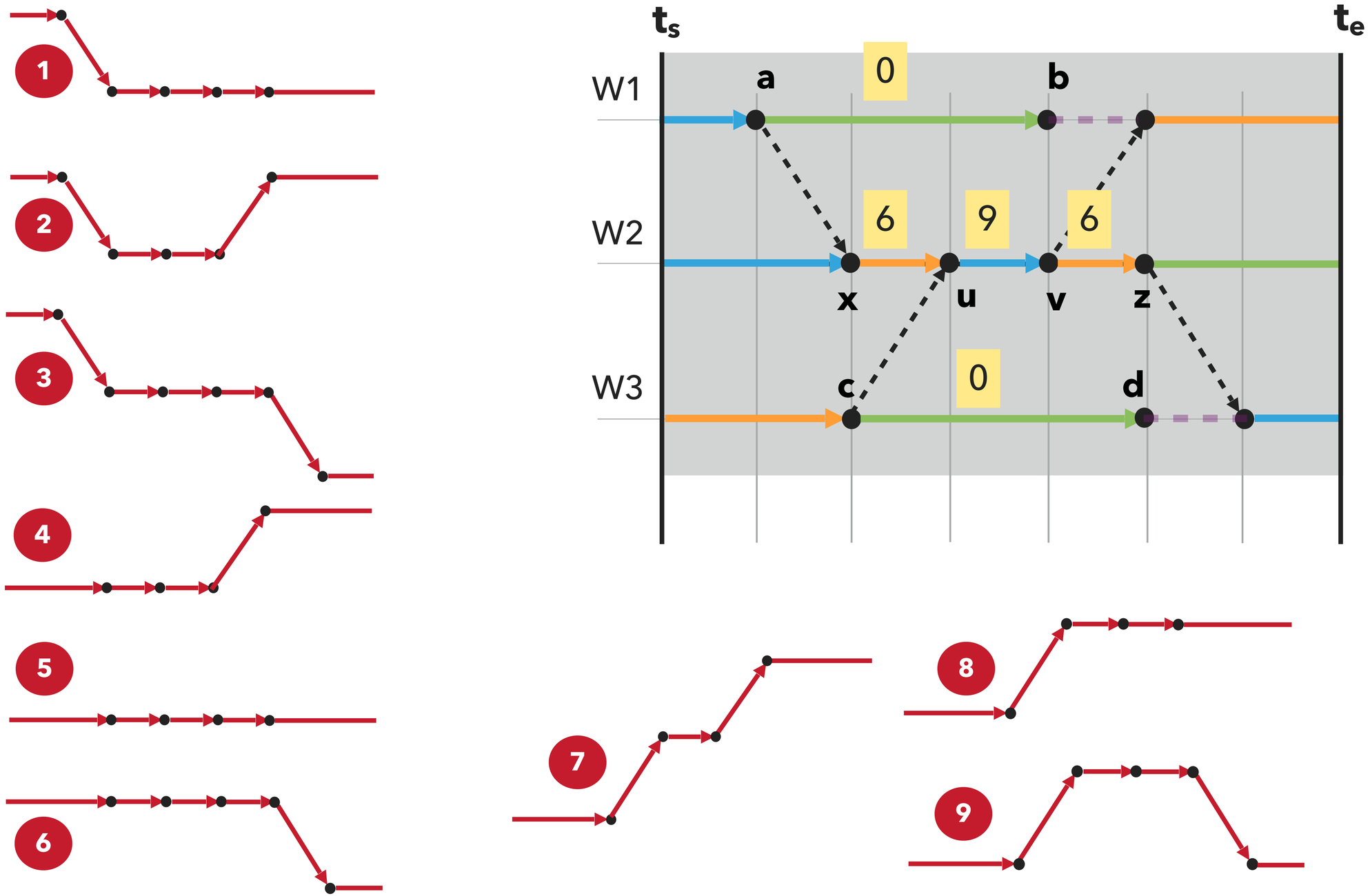
All paths are potentially part of the evolving critical path



How to rank activities with regard to criticality?

Intuition: the more paths an activity appears on the more probable it is critical


1
2
3
4
5
6
7
8
9
t_s
t_e
W1
W2
W3
a
b
x
u
v
z
c
d


1
2
3
4
5
6
7
8
9
t_s
t_e
W1
W2
W3
a
b
x
u
v
z
c
d
0
6
9
6
0

CRITICAL PARTICIPATION (CP METRIC)

An estimation of the activity's participation in the critical path

centrality: the number of paths this activity appears on

activity duration (edge weight)

$$CP_a = \frac{C(a) \cdot a_w}{N(t_e - t_s)}$$

total number of paths in the PAG snapshot

Can be computed without path materialization!

CRITICAL PARTICIPATION (CP METRIC)

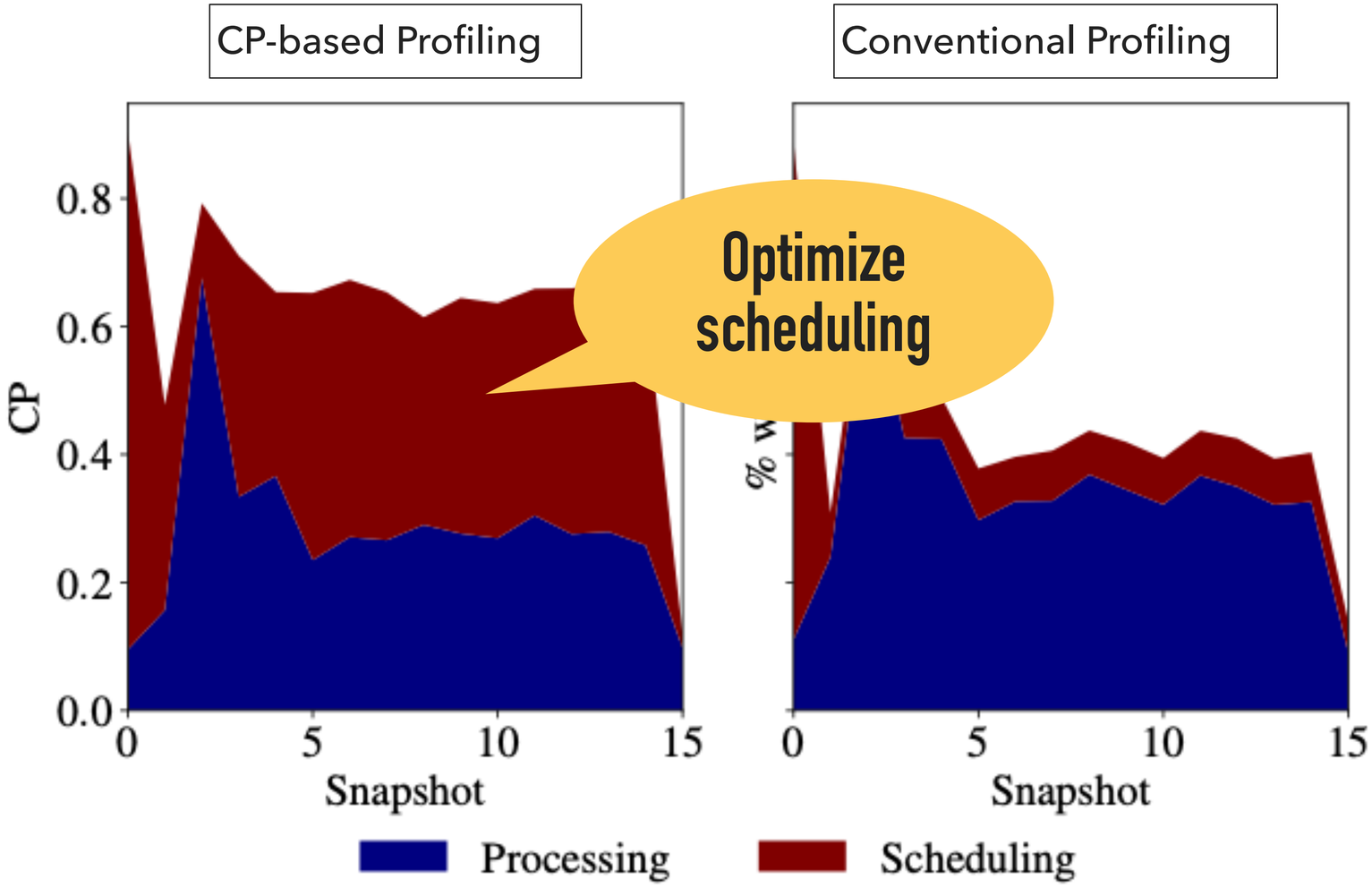
$$\sum_{\forall T \in G} \sum_{\forall a:T} CP_a = 1$$

The contribution of an activity to the critical path is normalized over all other activities in the same snapshot

T: type of activity a

G: the PAG snapshot

COMPARISON WITH CONVENTIONAL PROFILING



Apache Spark: Yahoo! Streaming Benchmark, 16 workers, 8s snapshots

See also: Venkataraman, S., Panda, A., Ousterhout, K., Ghodsi, A., Franklin, M. J., Recht, B., and Stoica, I. *Drizzle: Fast and Adaptable Stream Processing at Scale*. In SOSP, 2017.

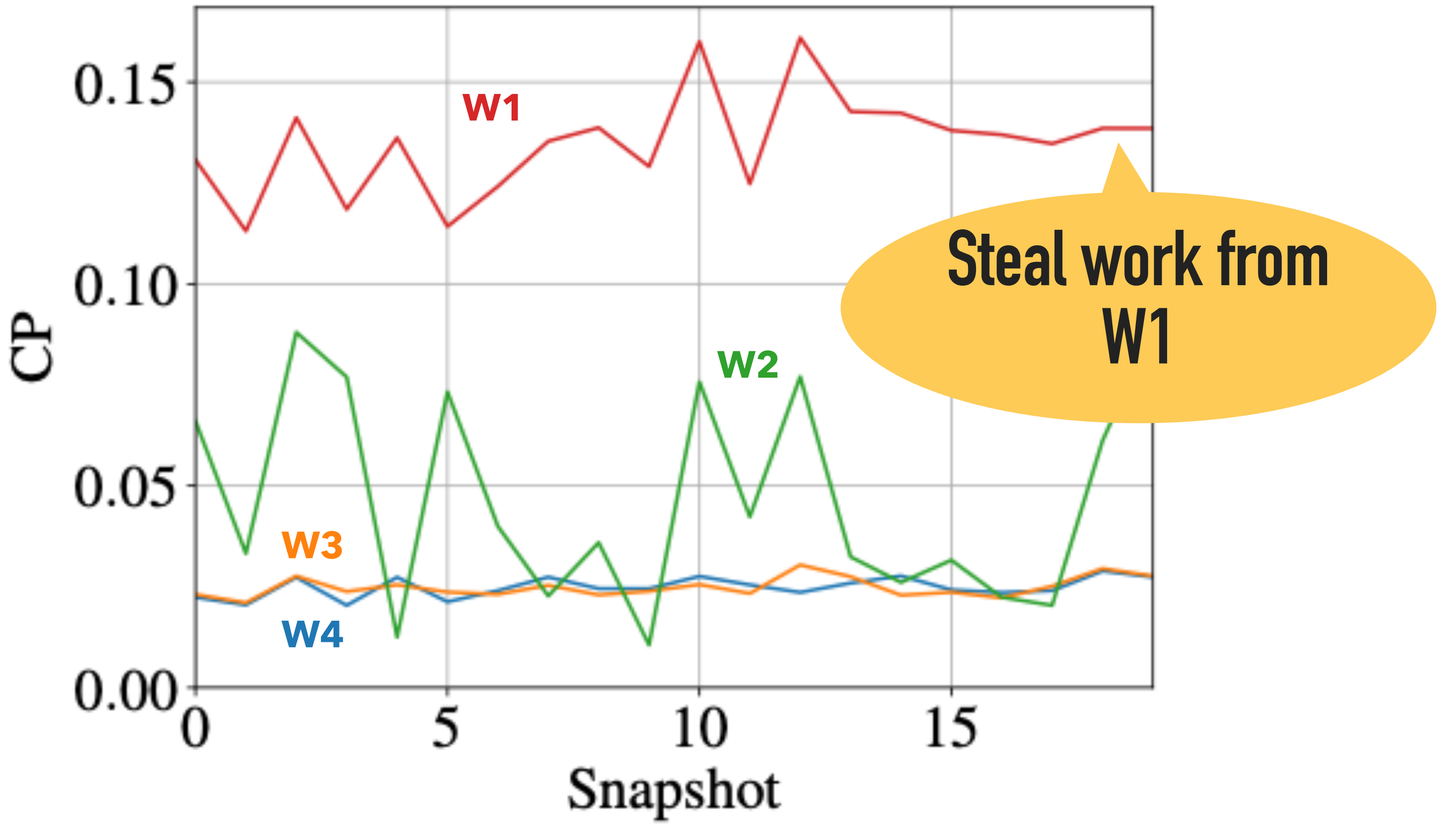
CP-BASED SUMMARIES

- Activity Summary
  - *which **activity type** is a bottleneck?*

CP-BASED SUMMARIES

- Activity Summary
  - *which **activity type** is a bottleneck?*
- Straggler Summary
  - *which **worker** is a bottleneck?*

Straggler Summary

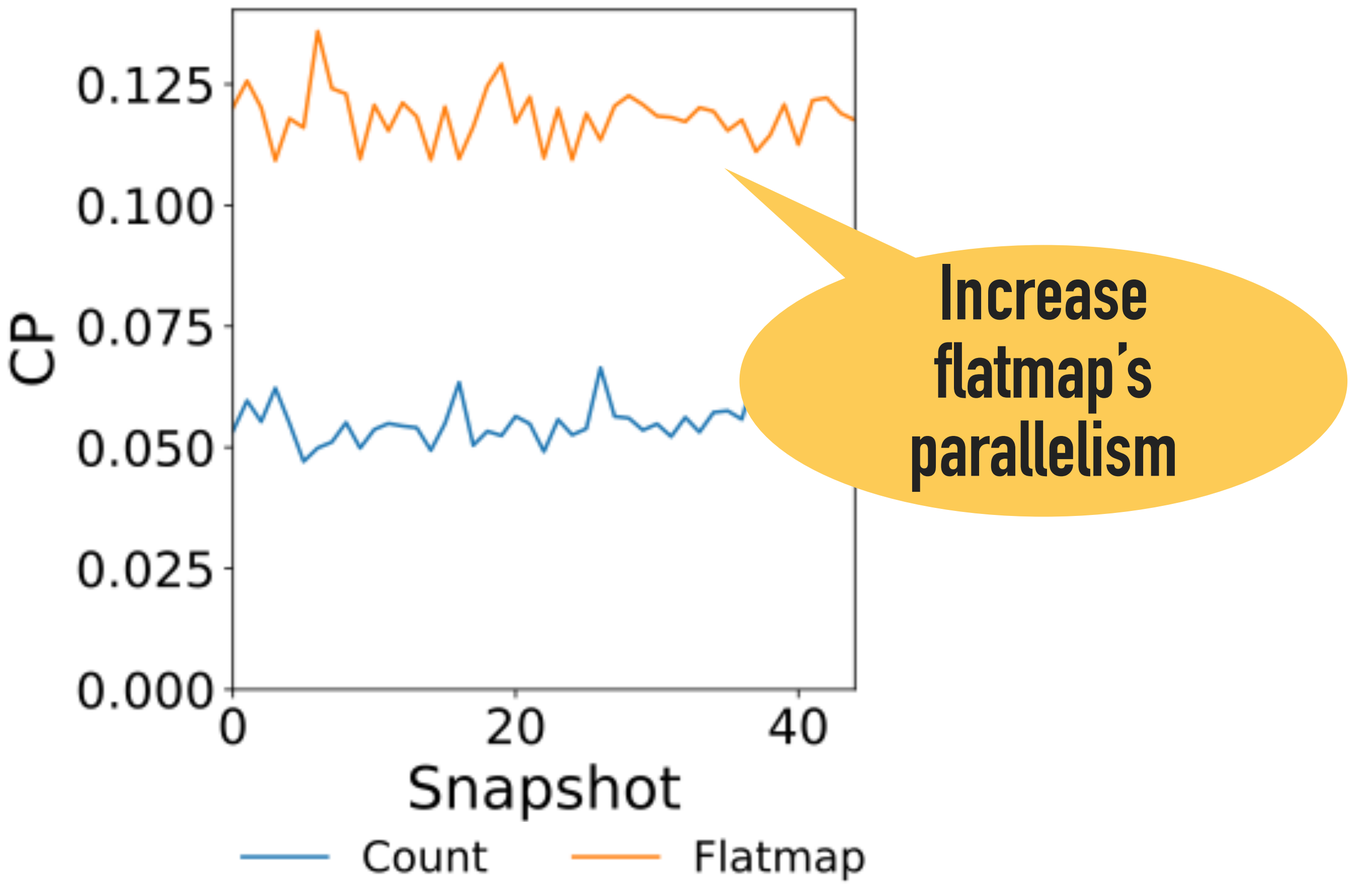


Apache Flink: Dhalion WordCount Benchmark, 4 workers, 1s snapshots

CP-BASED SUMMARIES

- Activity Summary
  - *which **activity type** is a bottleneck?*
- Straggler Summary
  - *which **worker** is a bottleneck?*
- Operator Summary
  - *which **operator** is a bottleneck?*

Operator Summary

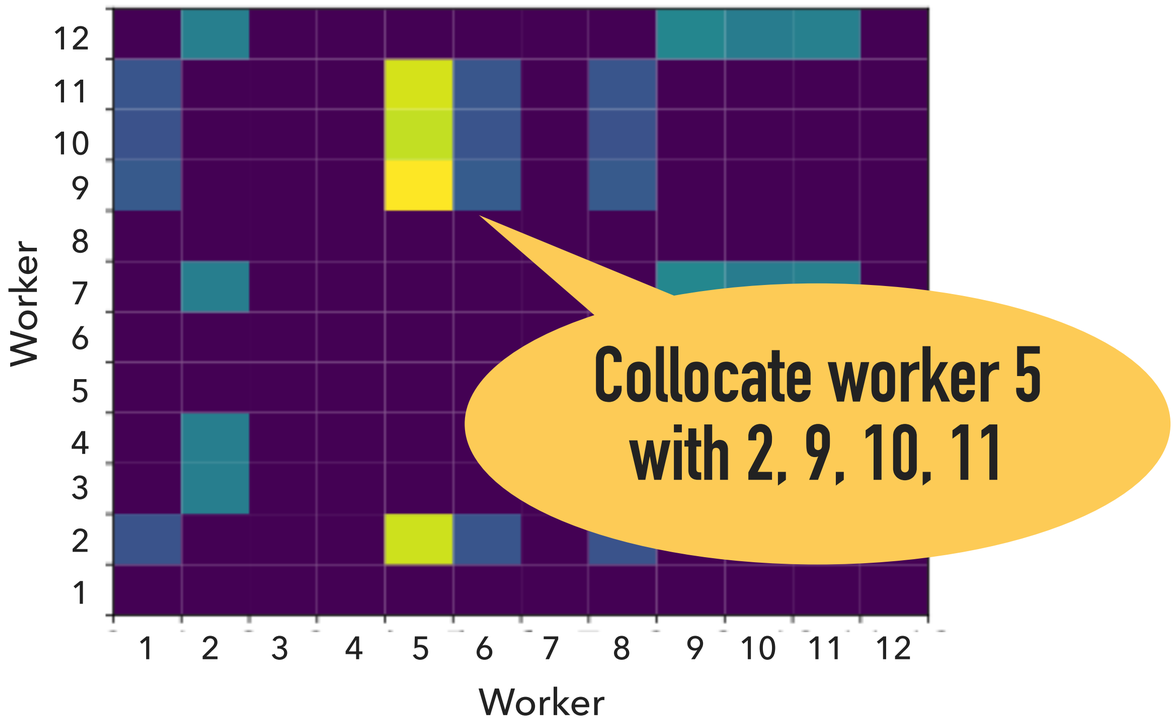


Apache Flink: Dhalion WordCount Benchmark, 10 workers, 1s snapshots

CP-BASED SUMMARIES

- Activity Summary
  - *which **activity type** is a bottleneck?*
- Straggler Summary
  - *which **worker** is a bottleneck?*
- Operator Summary
  - *which **operator** is a bottleneck?*
- Communication Summary
  - *which communication **channels** are bottlenecks?*

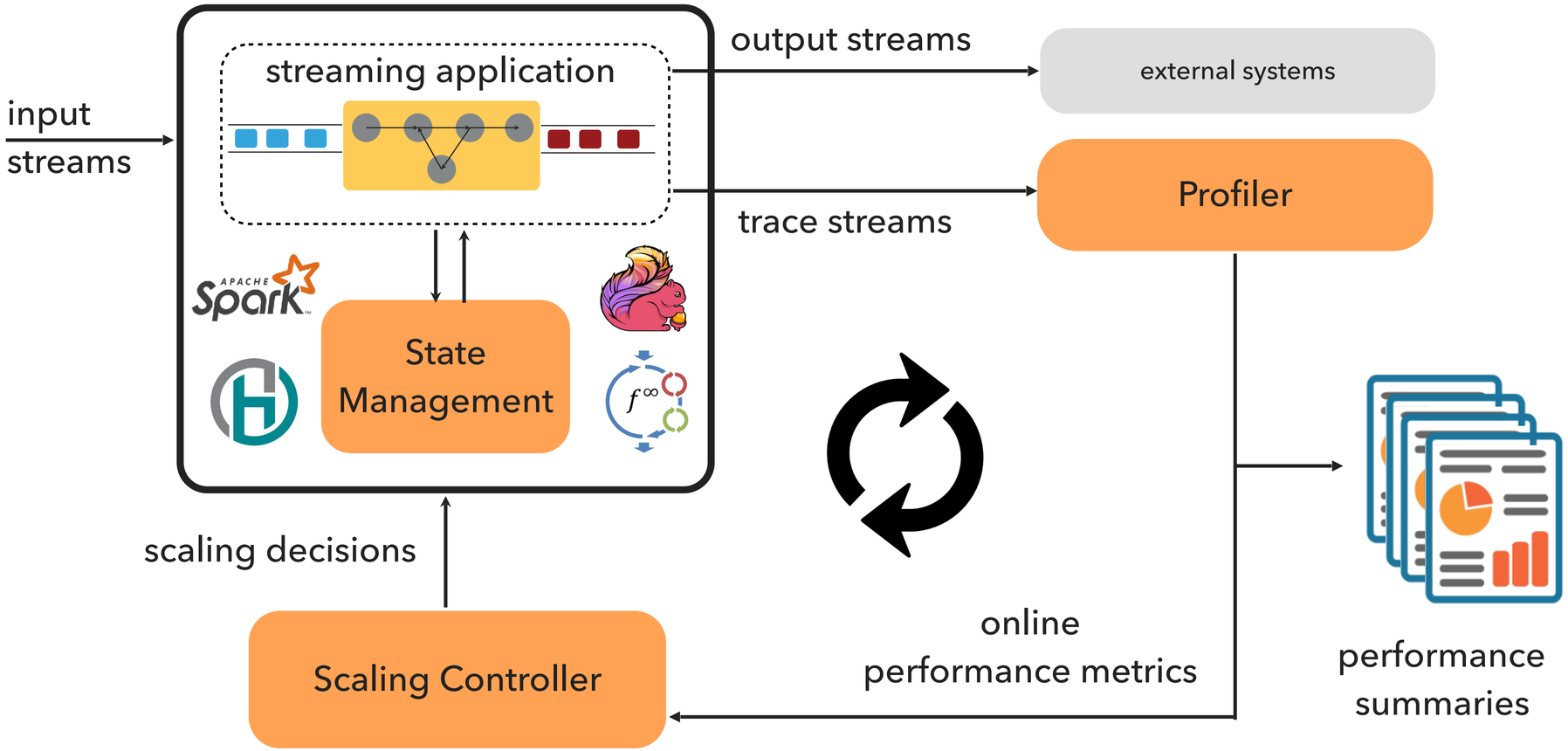
Communication Summary



Communication Criticality

Timely Dataflow: Breadth-First Search, 12 workers, 1s snapshots

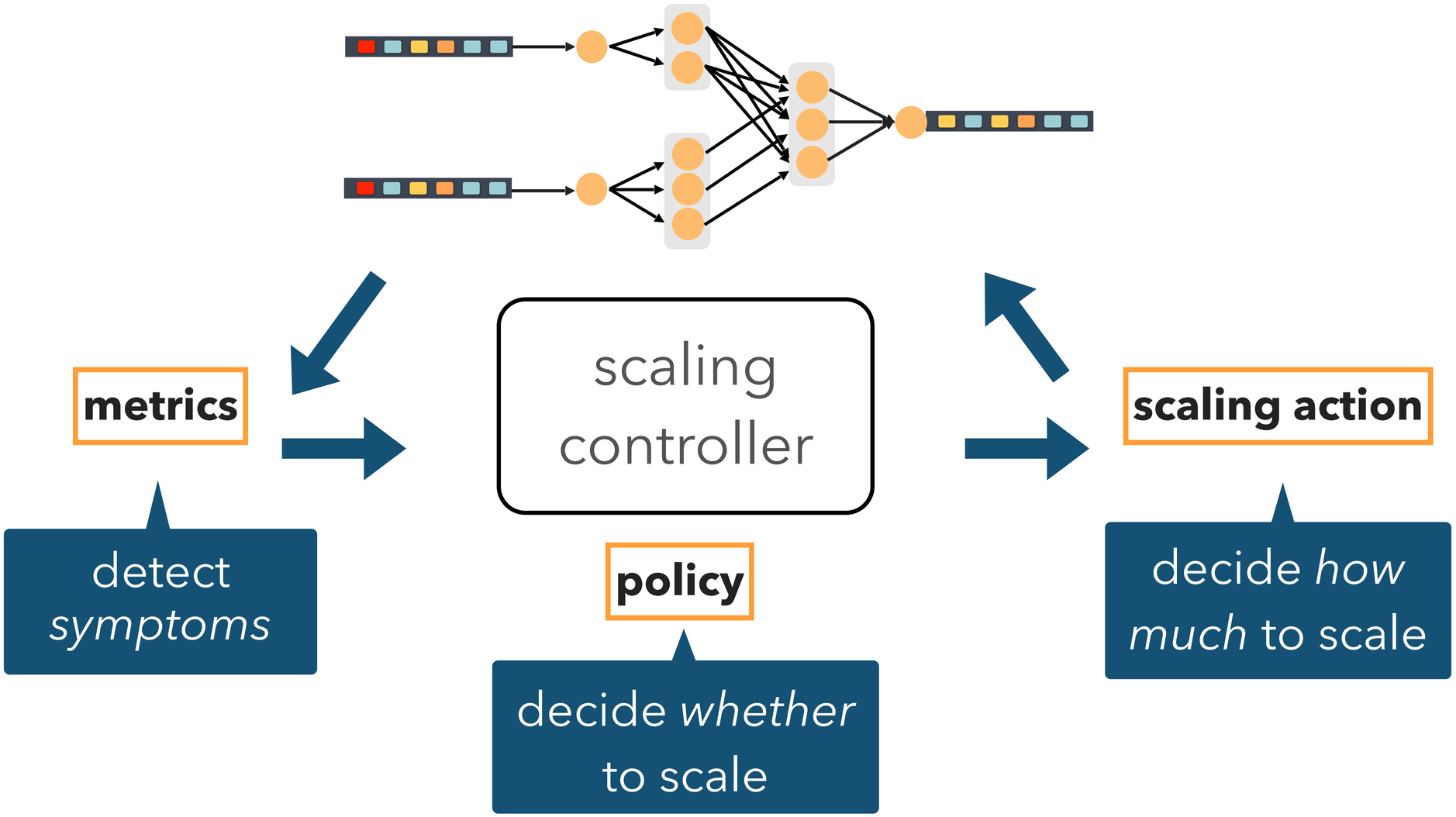
RECONFIGURABLE STREAM PROCESSING OVERVIEW



PART II

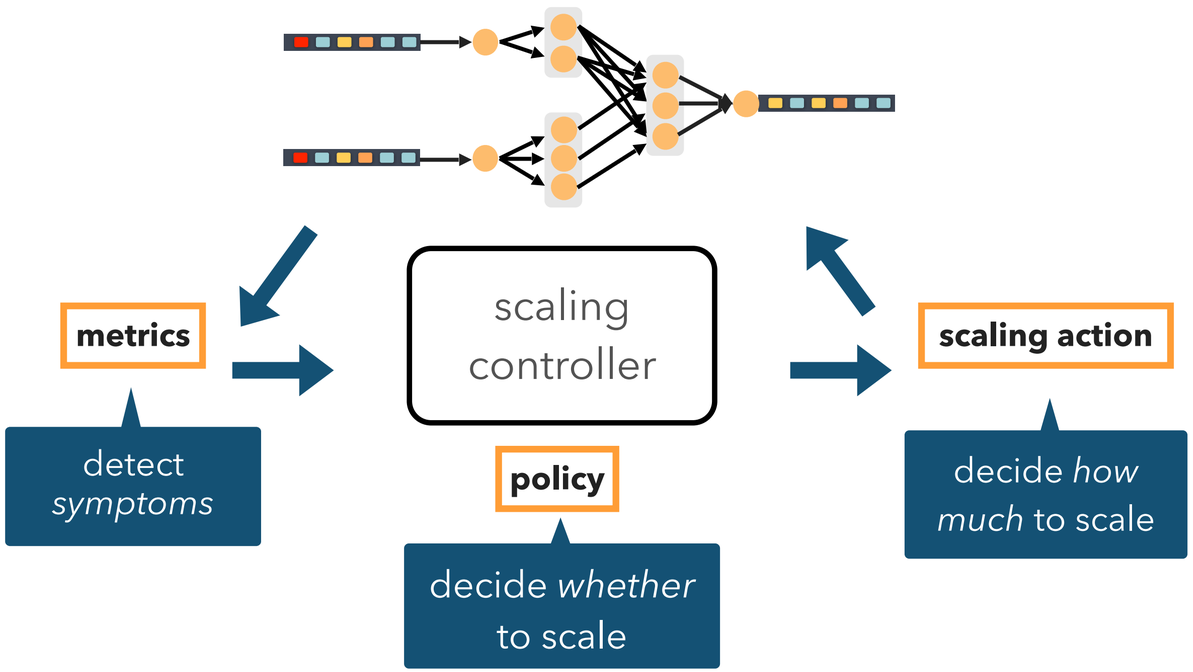
DS2: FAST, ACCURATE, AUTOMATIC SCALING DECISIONS FOR DISTRIBUTED STREAMING DATAFLOWS

AUTOMATIC SCALING OVERVIEW



AUTOMATIC SCALING REQUIREMENTS

- **Accuracy**
  - no over/under-provisioning
- **Stability**
  - no oscillations
- **Performance**
  - fast convergence



HEURISTIC SCALING APPROACHES

Borealis

StreamCloud

Seep

IBM Streams

Spark Streaming

Google Dataflow

Dhalion

metrics

CPU utilization
backlog, tuples/s
backpressure signal

Problematic under **interference**, **multi-tenancy**

policy

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

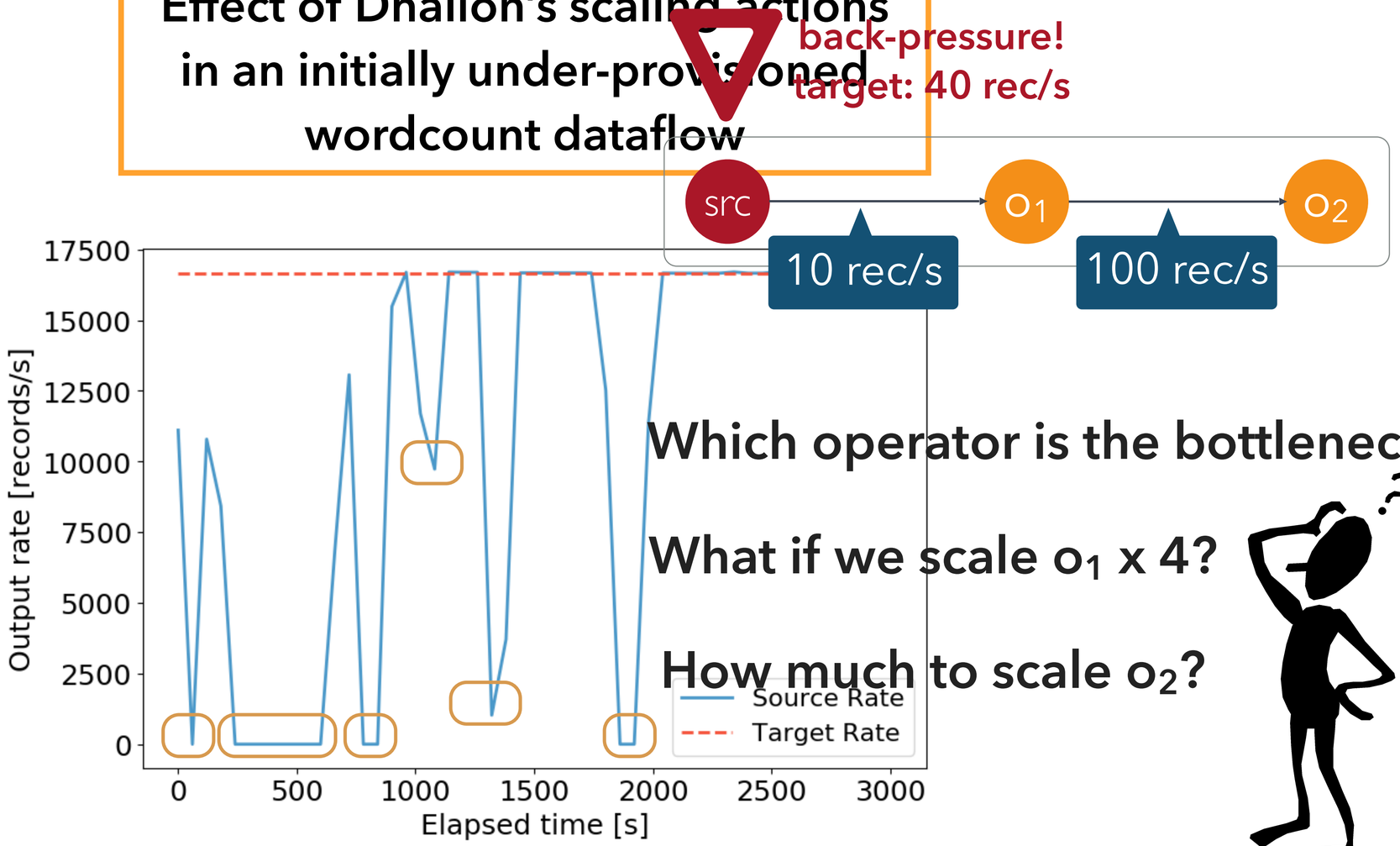
Sensitive to **noise**, manual, **hard to tune**

scaling action

small changes,
one operator
at a time

Non-predictive, **speculative** steps

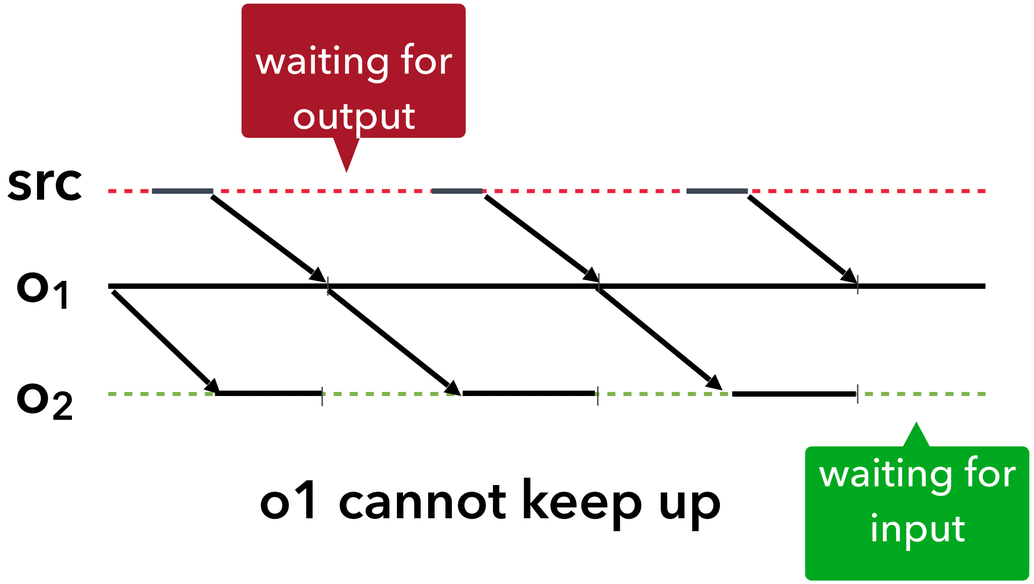
Effect of Dhalion's scaling actions in an initially under-provisioned wordcount dataflow



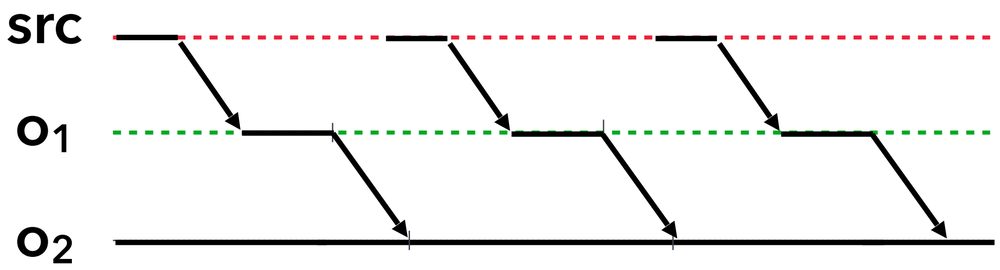
Which operator is the bottleneck?

What if we scale o_1 x 4?

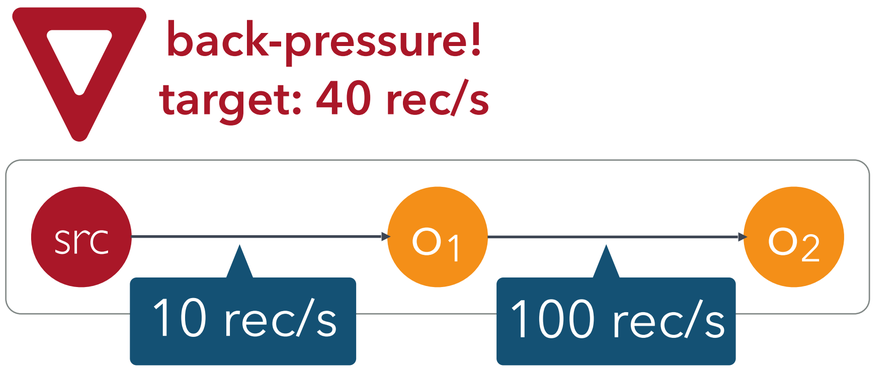
How much to scale o_2?



o1 cannot keep up



o2 cannot keep up



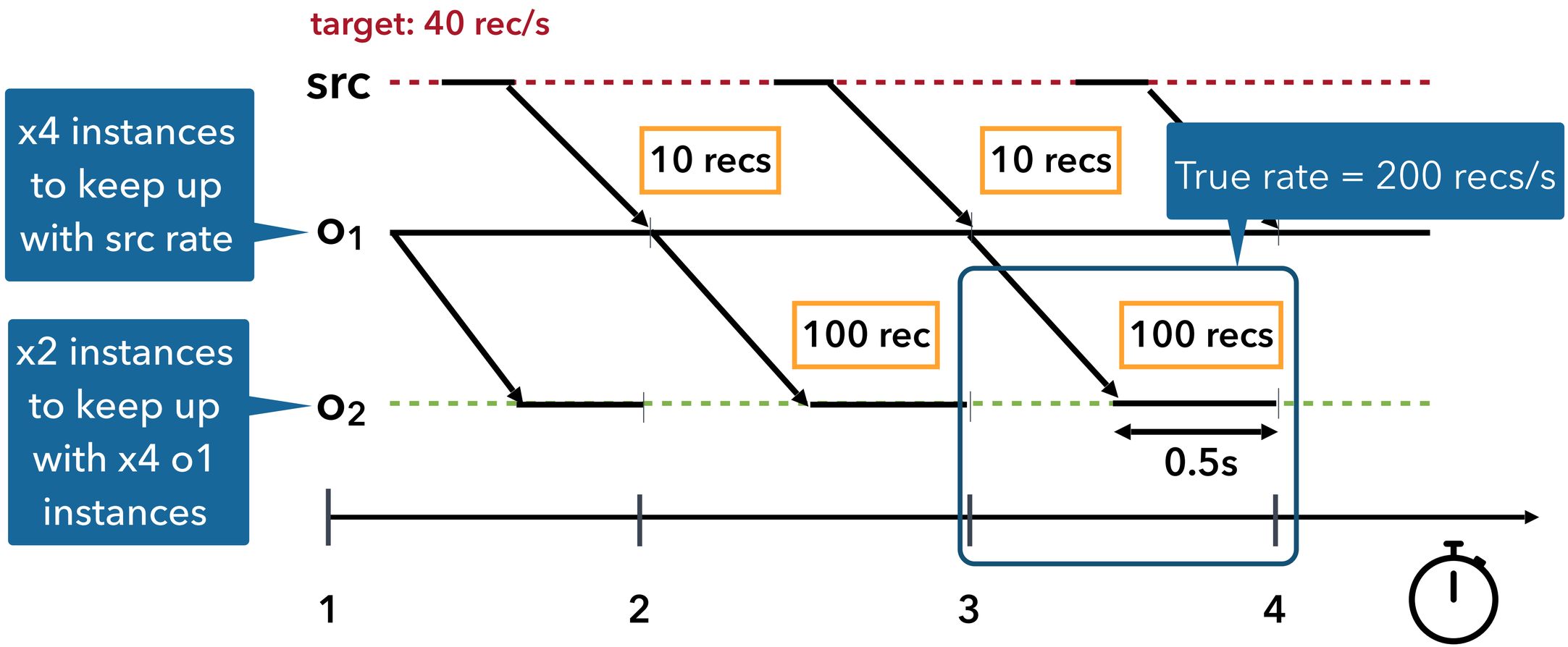
Which operator is the bottleneck?

What if we scale o_1 x 4?

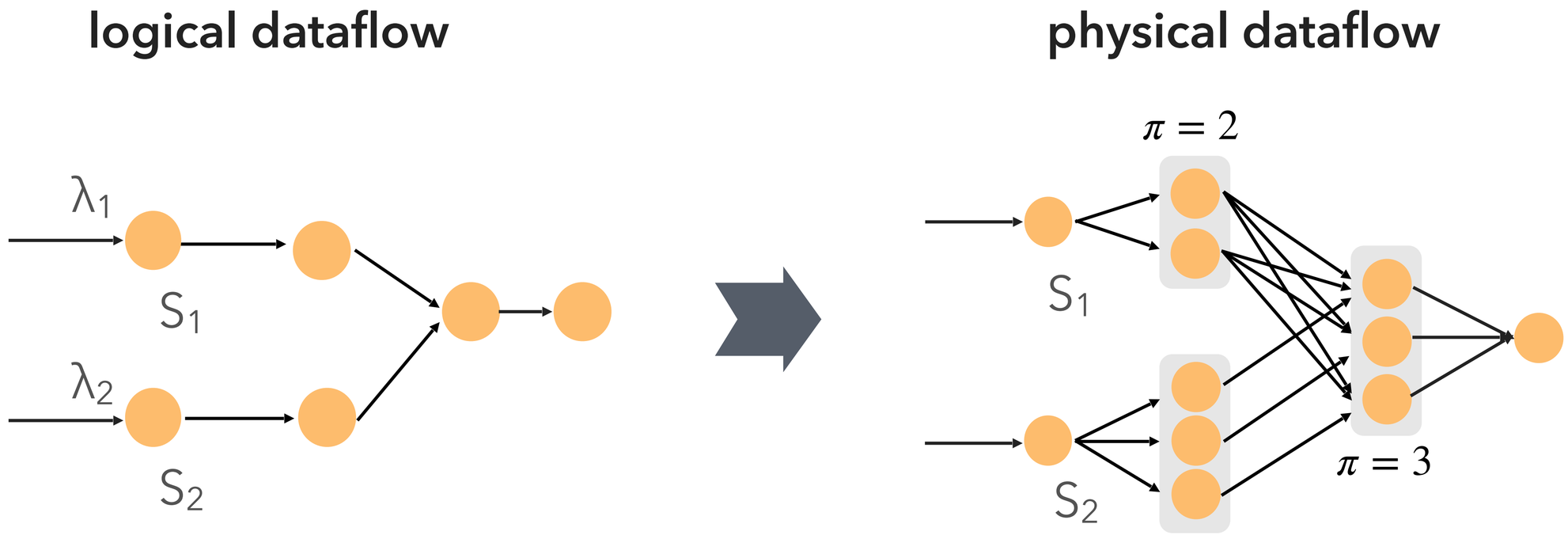
How much to scale o_2?

THE DS2 MODEL

<u>Intuition</u>: use the dataflow graph to extract **operator dependencies** and system **instrumentation** to collect **accurate**, representative metrics.



THE DATAFLOW 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**.

DS2 MODEL: USEFUL TIME AND TRUE RATES

Useful time W_u

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

True processing / output rates

$$\lambda_p = \frac{R_{pcd}}{W_u} \qquad \lambda_o = \frac{R_{psd}}{W_u}$$

Aggregated true processing / output rates

$$o_i[\lambda_p] = \sum_{k=1}^{k=p_i} \lambda_p^k \qquad o_i[\lambda_o] = \sum_{k=1}^{k=p_i} \lambda_o^k$$

DS2 MODEL: OPTIMAL PARALLELISM

Optimal parallelism per operator

$$\pi_i = \left\lceil \sum_{\forall j: j<i} A_{ji} \cdot o_j[\lambda_o]^* \cdot \left(\frac{o_i[\lambda_p]}{p_i} \right)^{-1} \right\rceil, n \leq i < m$$

Dataflow adjacency matrix

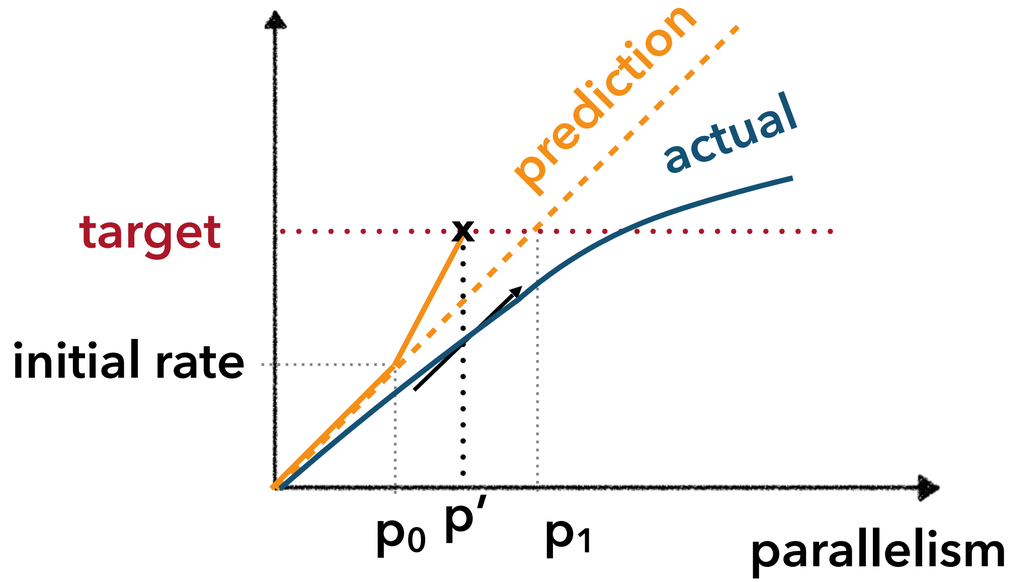
Aggregated true output rate of operator o_j considering optimal parallelism for all operators up to o_j

Measured true processing rate with current parallelism

DS2 MODEL PROPERTIES

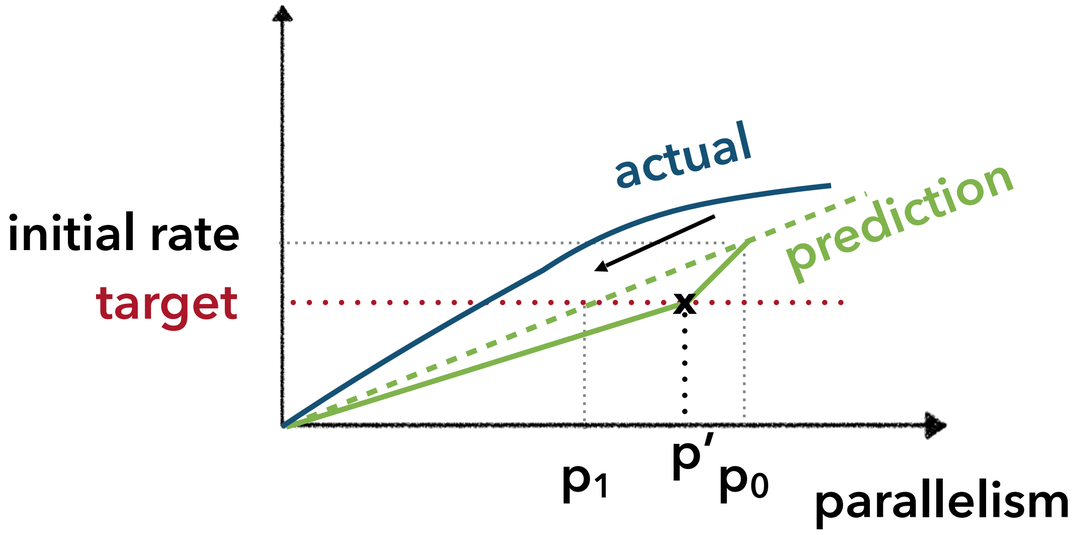
If operator scaling is **linear**, then:

- **no overshoot** when scaling up
- **no undershoot** when scaling down

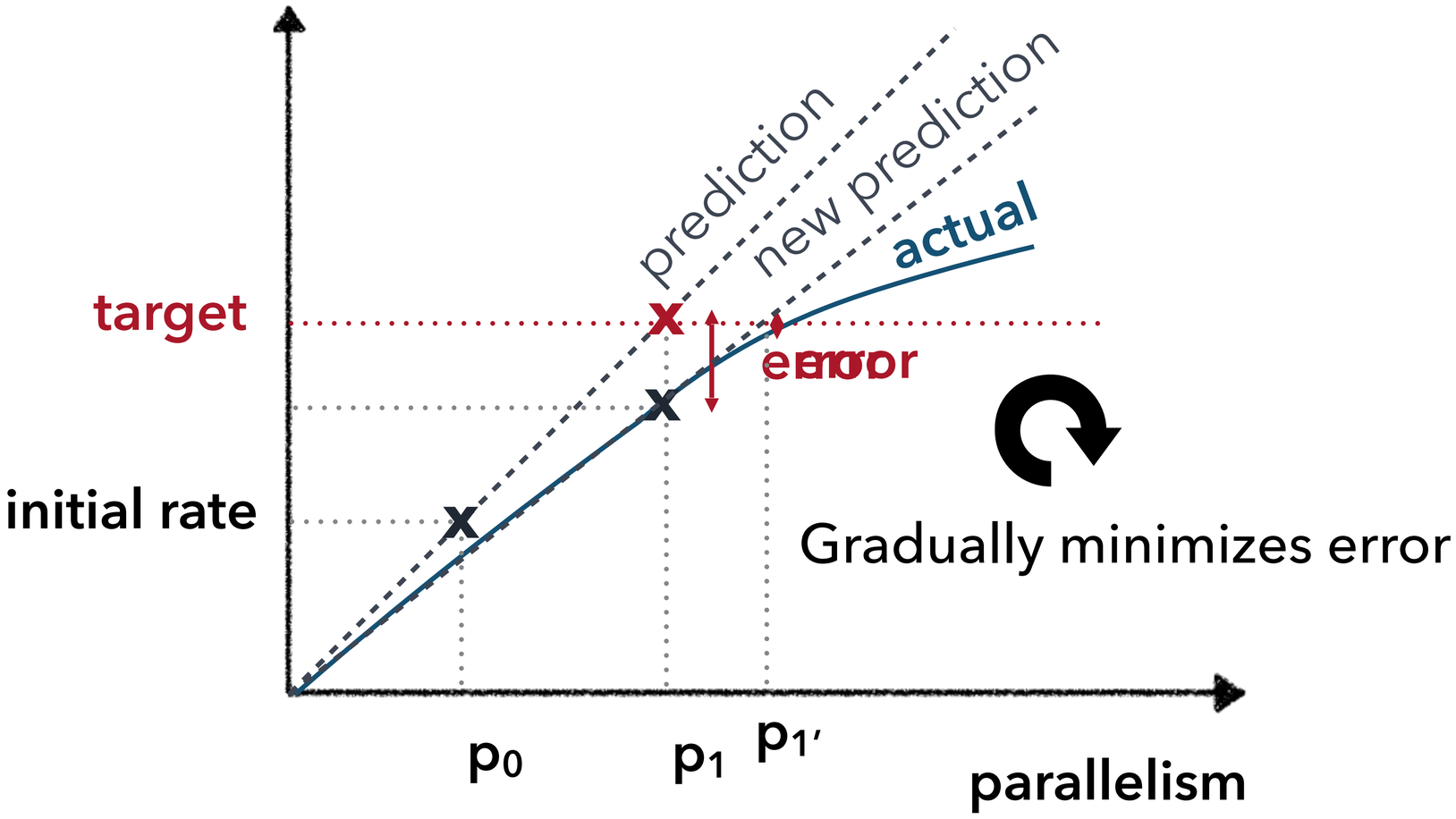


Ideal rates act as un **upper bound** when scaling up and as a **lower bound** when scaling down:

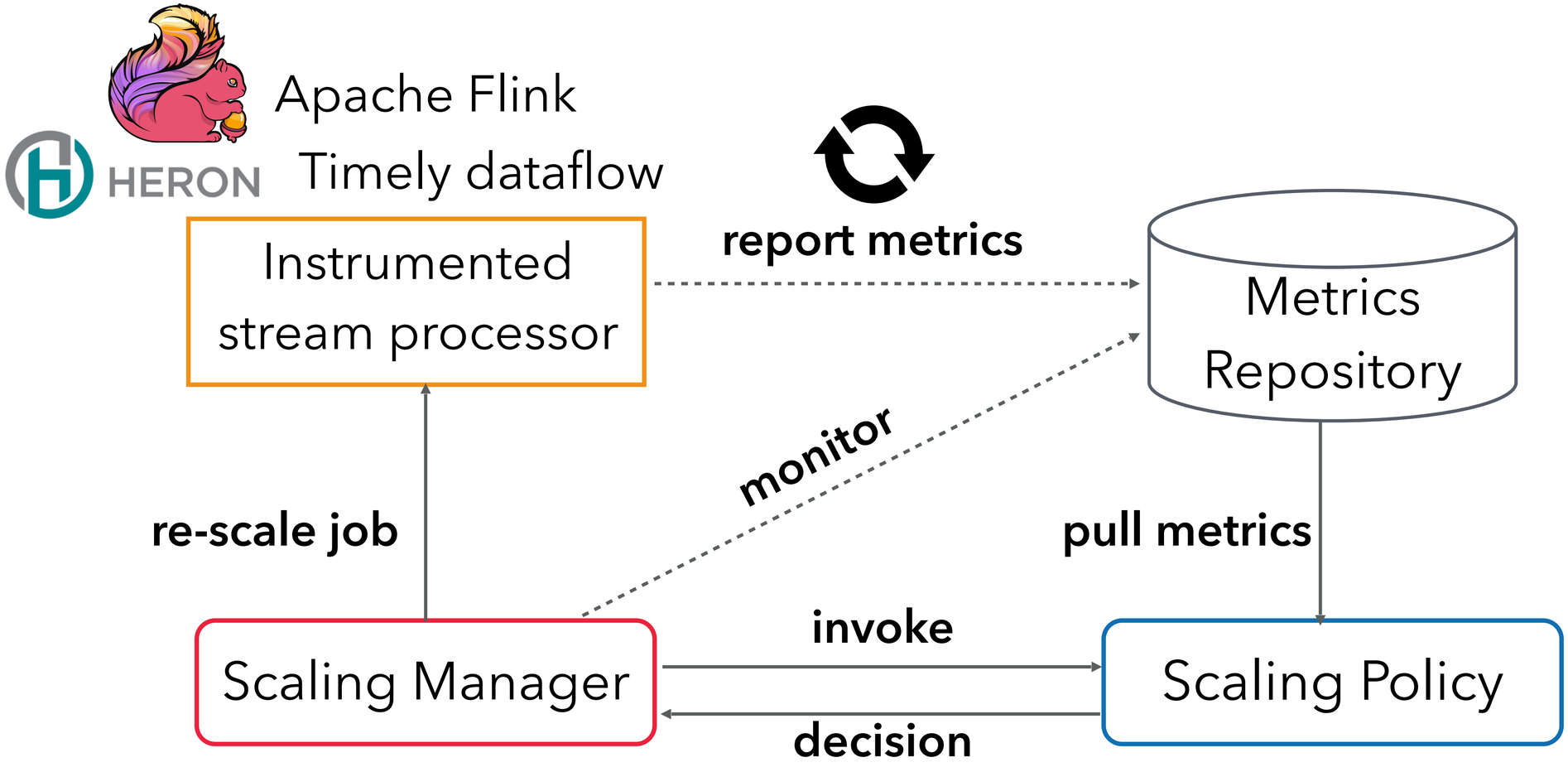
- DS2 will **converge monotonically** to the target rate



DS2 MODEL PROPERTIES



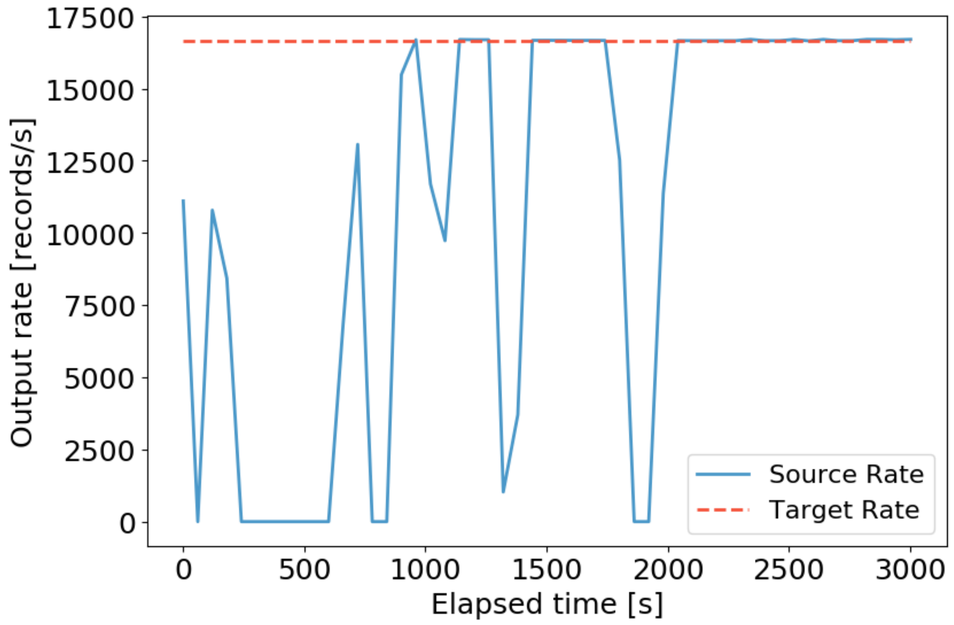
ONLINE AND REACTIVE OPERATION

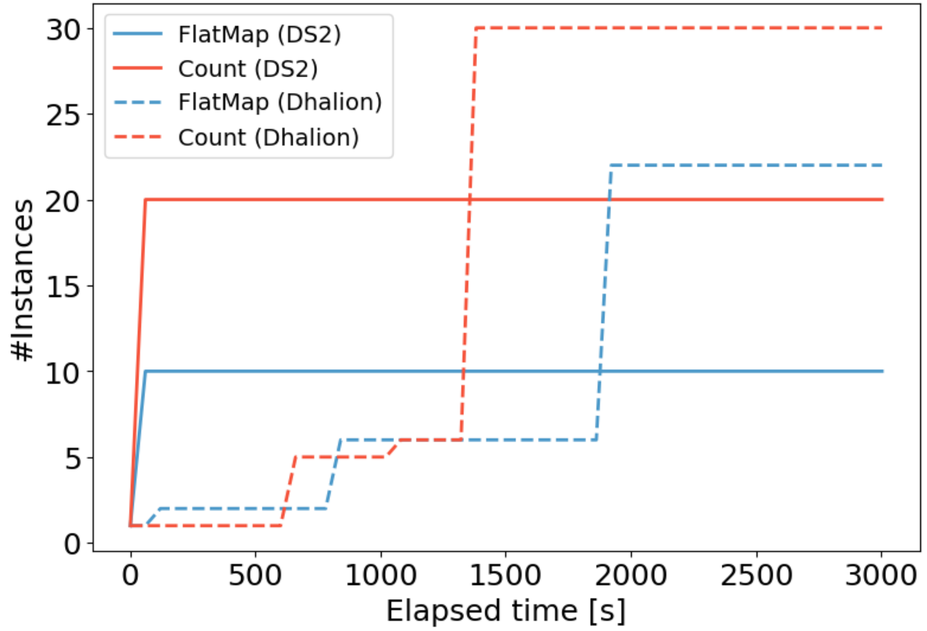


DS2 VS STATE-OF-THE-ART ON HERON

Initially under-provisioned wordcount dataflow

Target rate: 16.700 rec/s



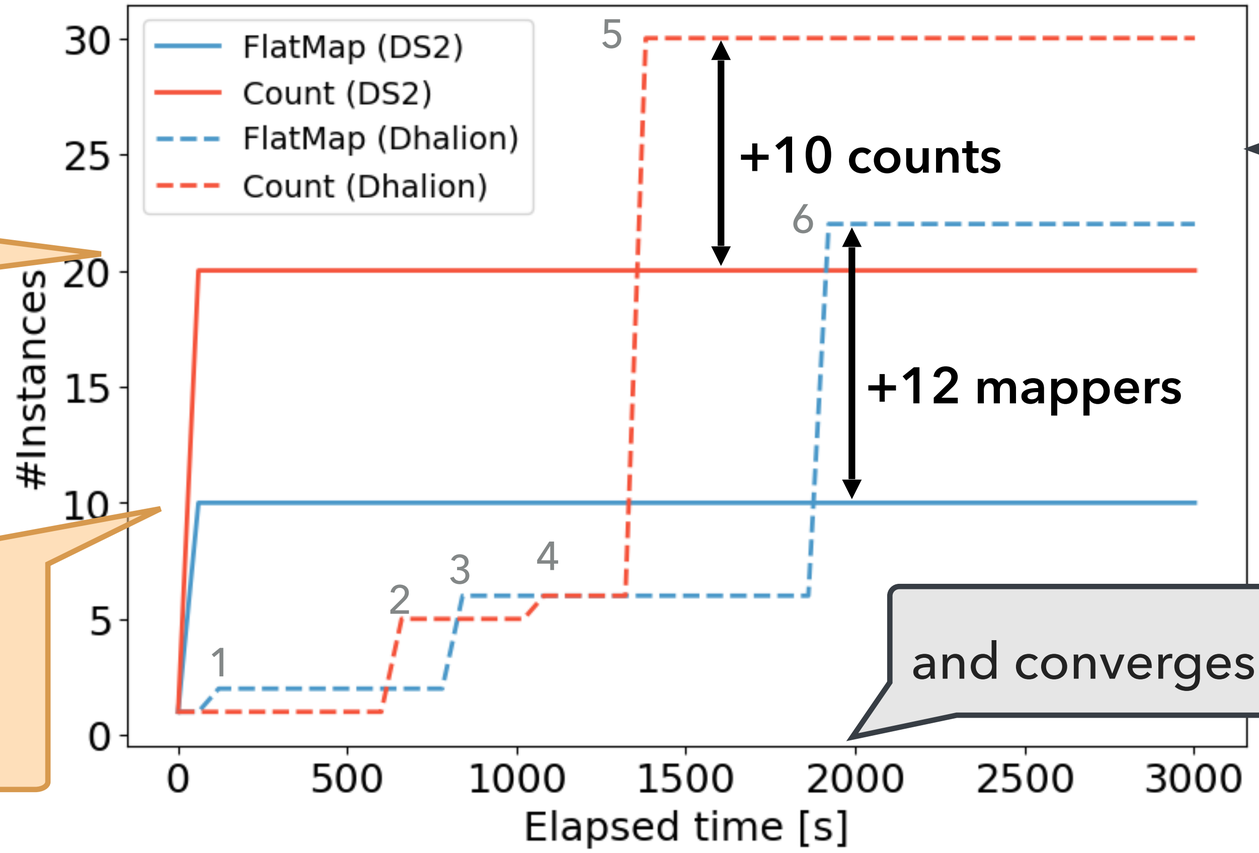


DS2 VS STATE-OF-THE-ART ON HERON

Initially under-provisioned wordcount dataflow

Target rate: 16.700 rec/s

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



Dhalion scales one operator at a time, and needs **six steps** in total

and converges in **60s**, as soon as it receives the Heron metrics

and converges in **2000s**

DS2 CONVERGENCE ON NEXMARK QUERIES

One step for many queries and initial configurations, at most **three**

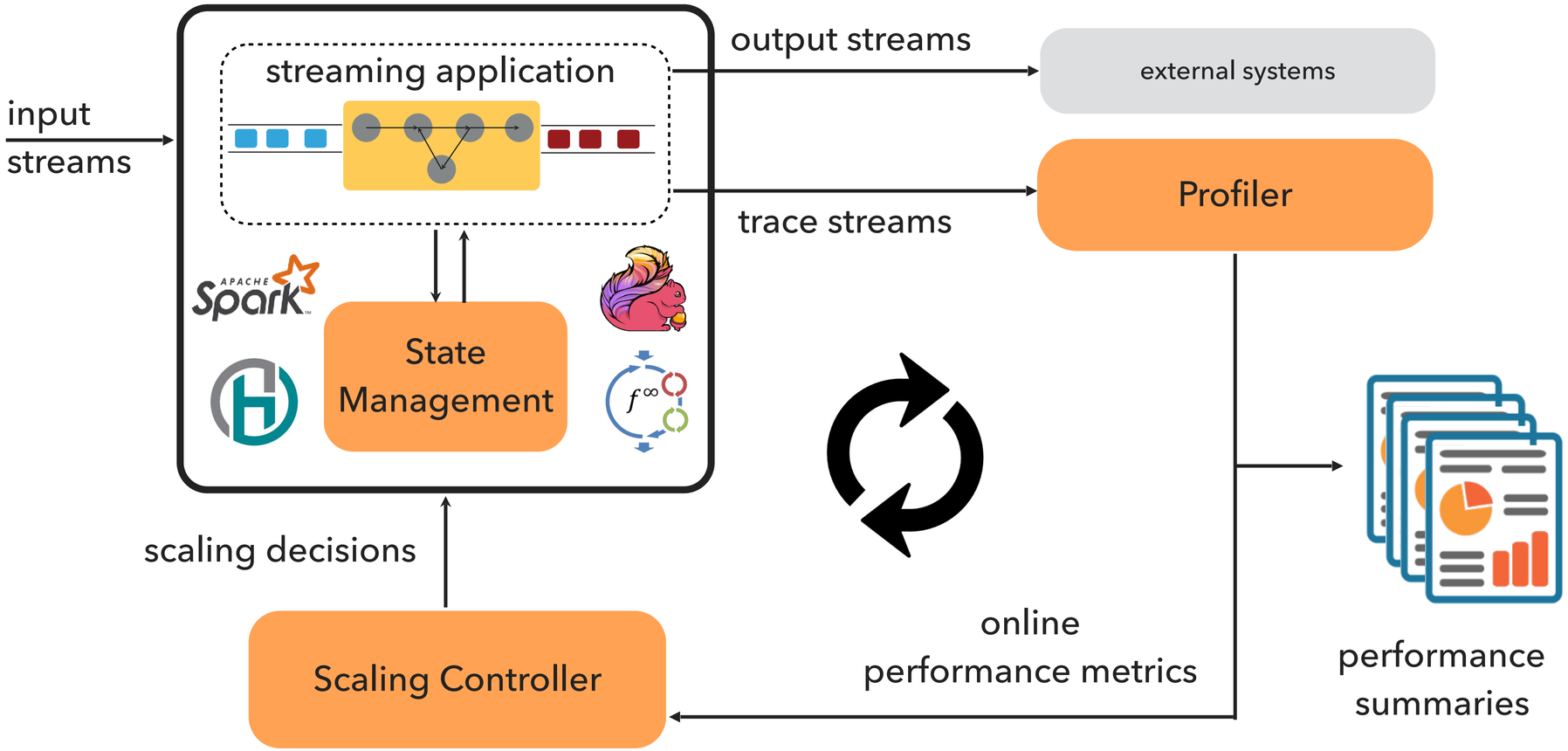
	initial parallelism	Q1: flatmap	Q2: filter	Q3: incremental join	Q5: tumbling window join	Q8: sliding window	Q11: session window
scale-up	8 →	12 → 16	11 → 13 → 14	16 → 20	14 → 15 → 16	10	12 → 22 → 28
	12 →	16	14	18 → 20	16	10	22 → 28
	16 →	16	12 → 14	20	16	8 → 10	26 → 28
	20 →	16	13 → 14	20	14 → 16	8 → 10	28
	24 →	16	14	20	14 → 16	8 → 10	28
scale-down	28 →	16	14	20	13 → 16	8 → 10	28

Convergence to the optimal configuration

No oscillations

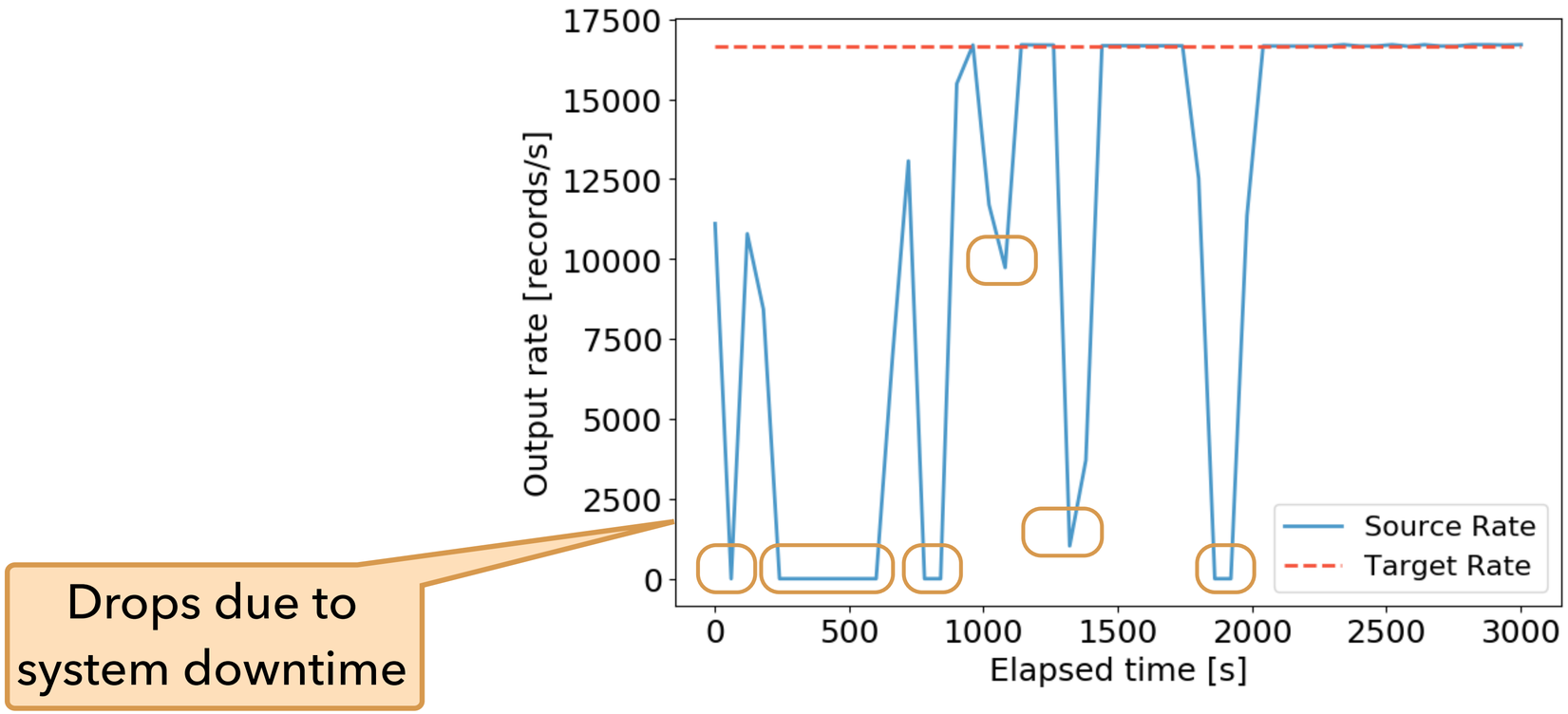
→ : scaling action

RECONFIGURABLE STREAM PROCESSING OVERVIEW

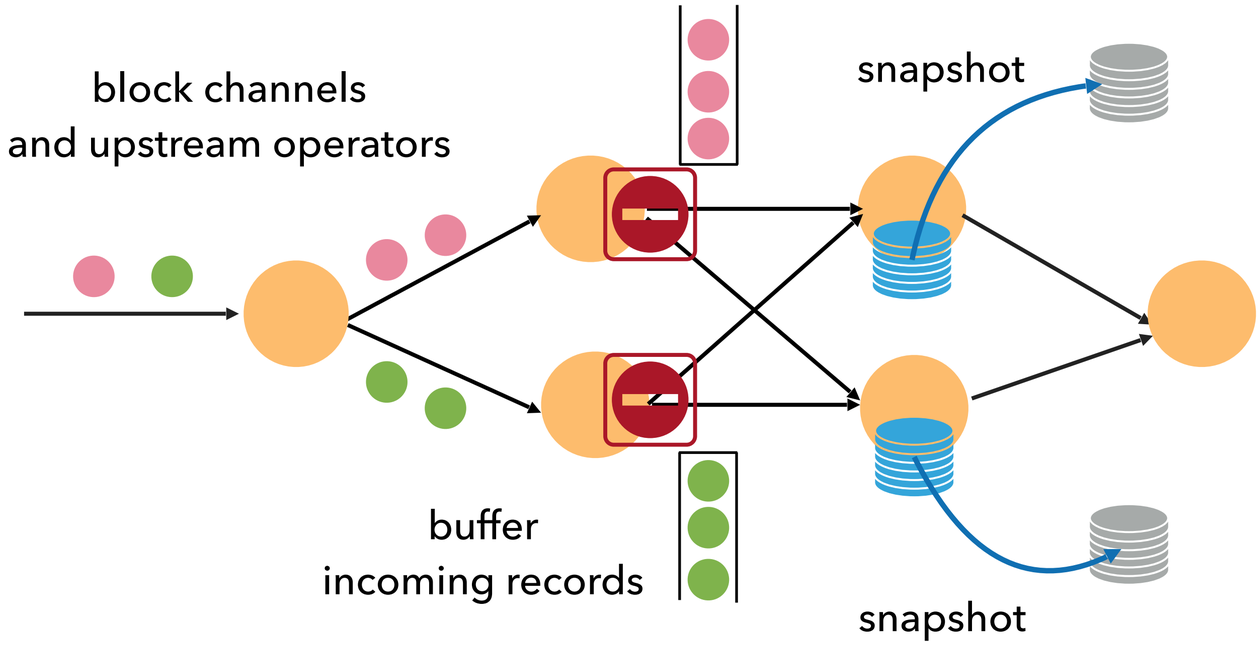


PART III

MEGAPHONE: LATENCY-CONSCIOUS STATE MIGRATION FOR DISTRIBUTED STREAMING DATAFLOWS

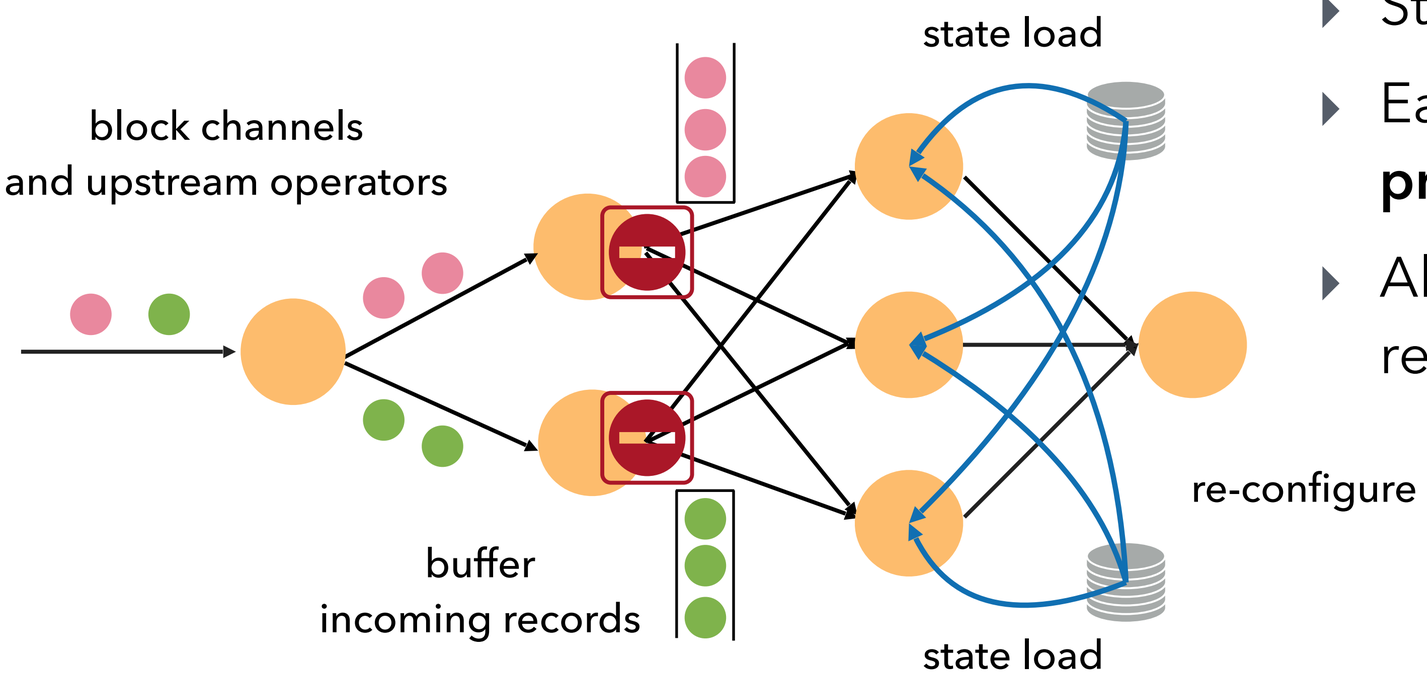

Drops due to system downtime
Output rate [records/s]
17500
15000
12500
10000
7500
5000
2500
0
0
500
1000
1500
2000
2500
3000
Elapsed time [s]
Source Rate
Target Rate

PAUSE-AND-RESTART STATE MIGRATION



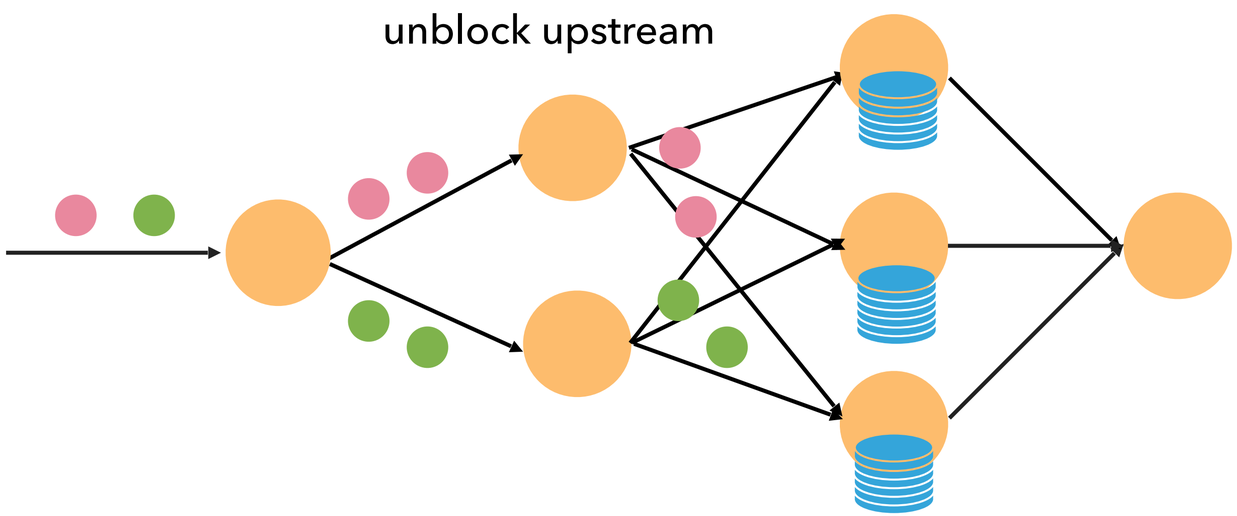
- State is **scoped** to a single task
- Each stateful task is responsible for **processing** *and* **state management**

PAUSE-AND-RESTART STATE MIGRATION



- State is **scoped** to a single task
- Each stateful task is responsible for **processing** *and* **state management**
- All affected operators **block** until the reconfiguration is complete

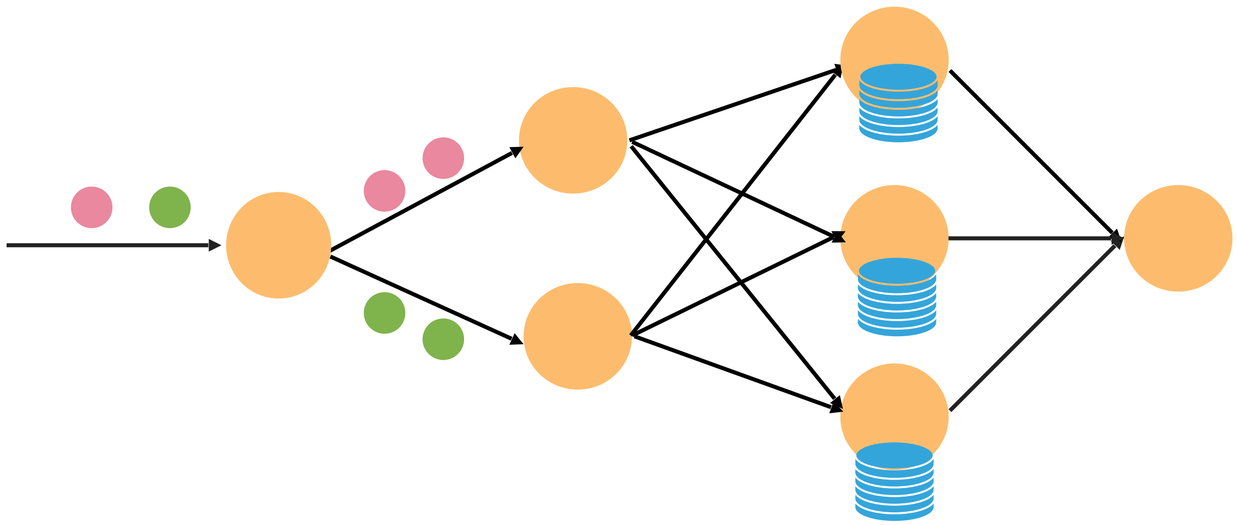
PAUSE-AND-RESTART STATE MIGRATION



- State is **scoped** to a single task
- Each stateful task is responsible for **processing** *and* **state management**
- All affected operators **block** until the reconfiguration is complete

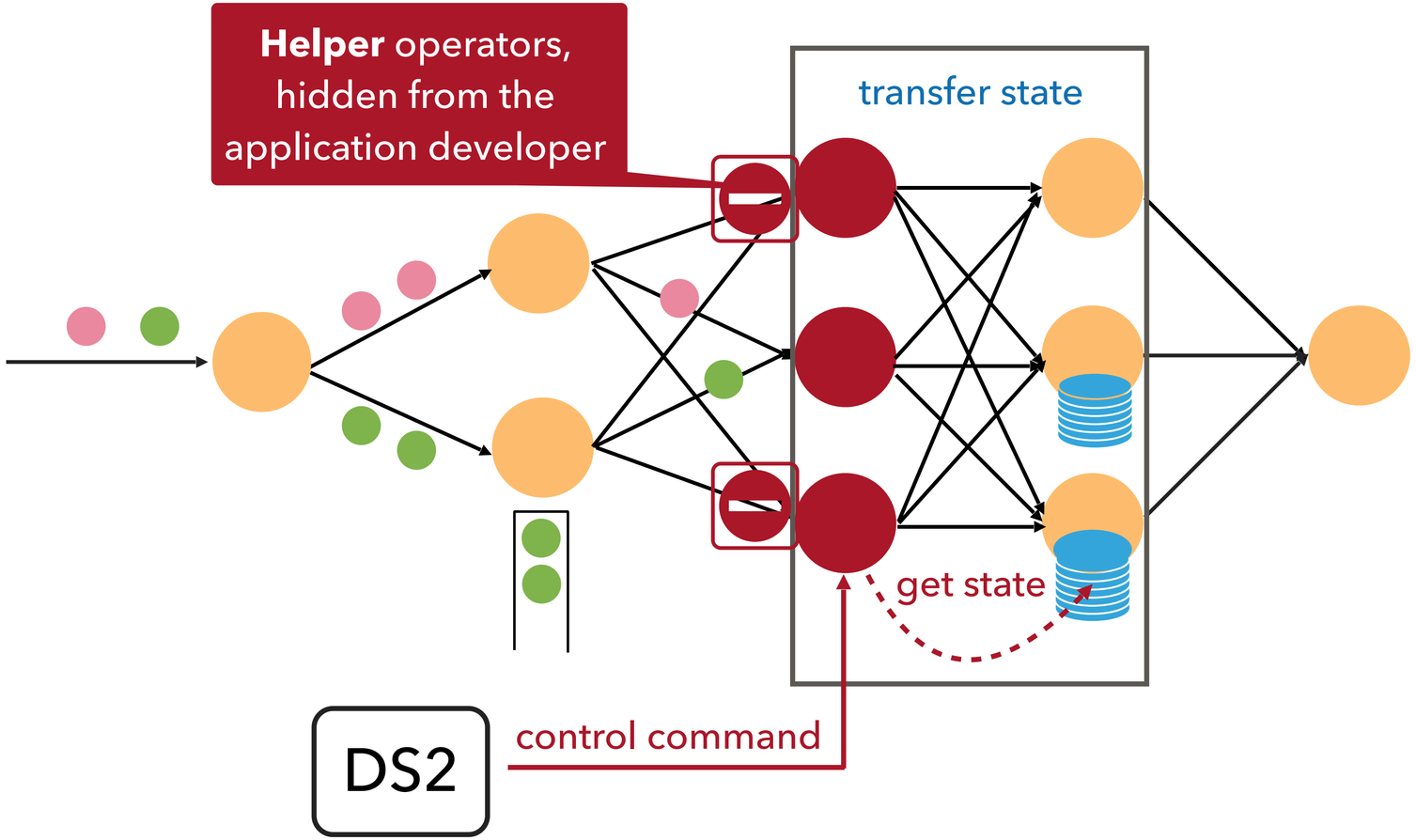
LIVE STATE MIGRATION

<u>Intuition</u>: treat state migration as **a dataflow operation** and *interleave* **fine-grained** state transfers with processing.



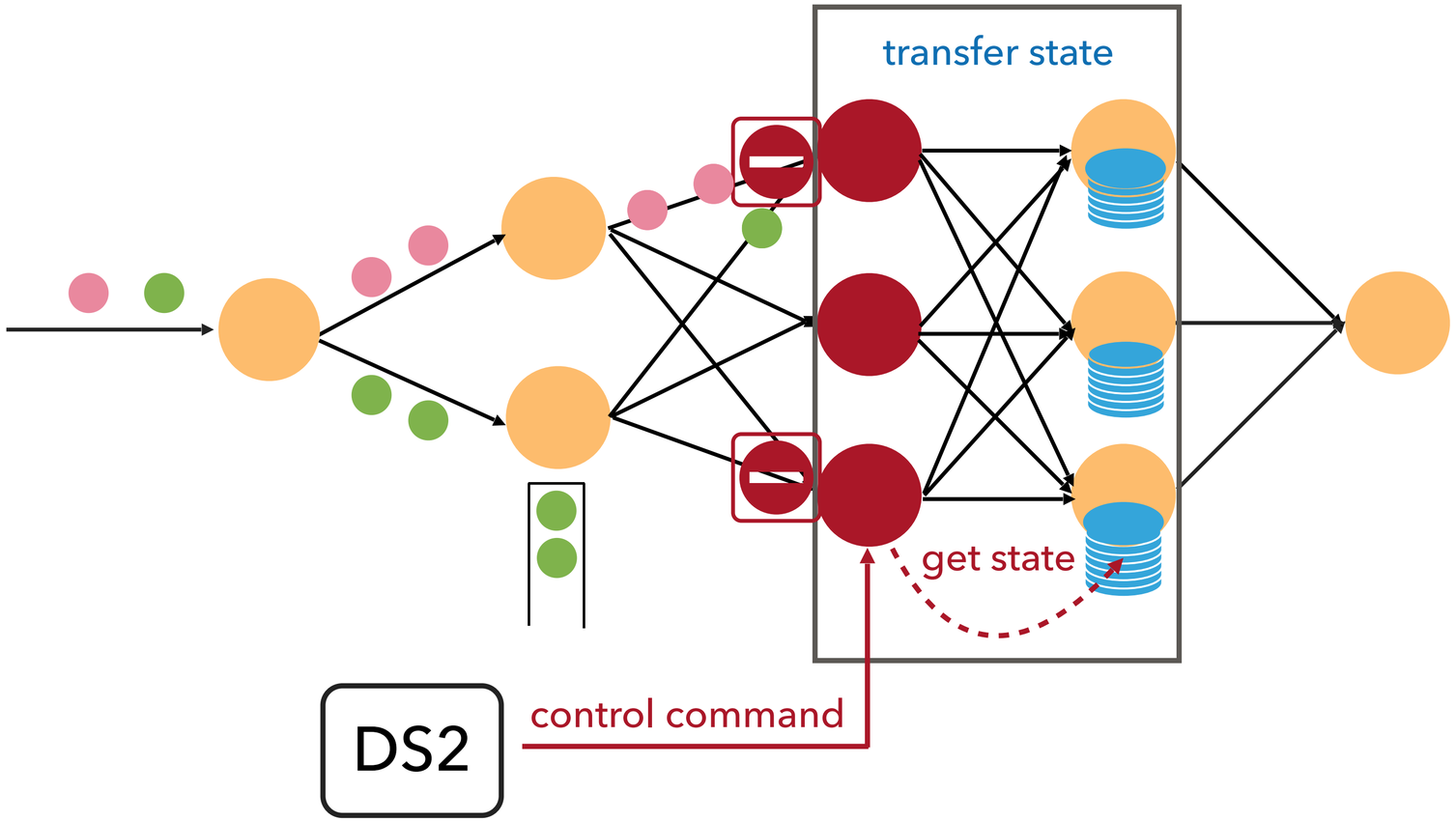
LIVE STATE MIGRATION

<u>Intuition</u>: treat state migration as **a dataflow operation** and *interleave* **fine-grained** state transfers with processing.



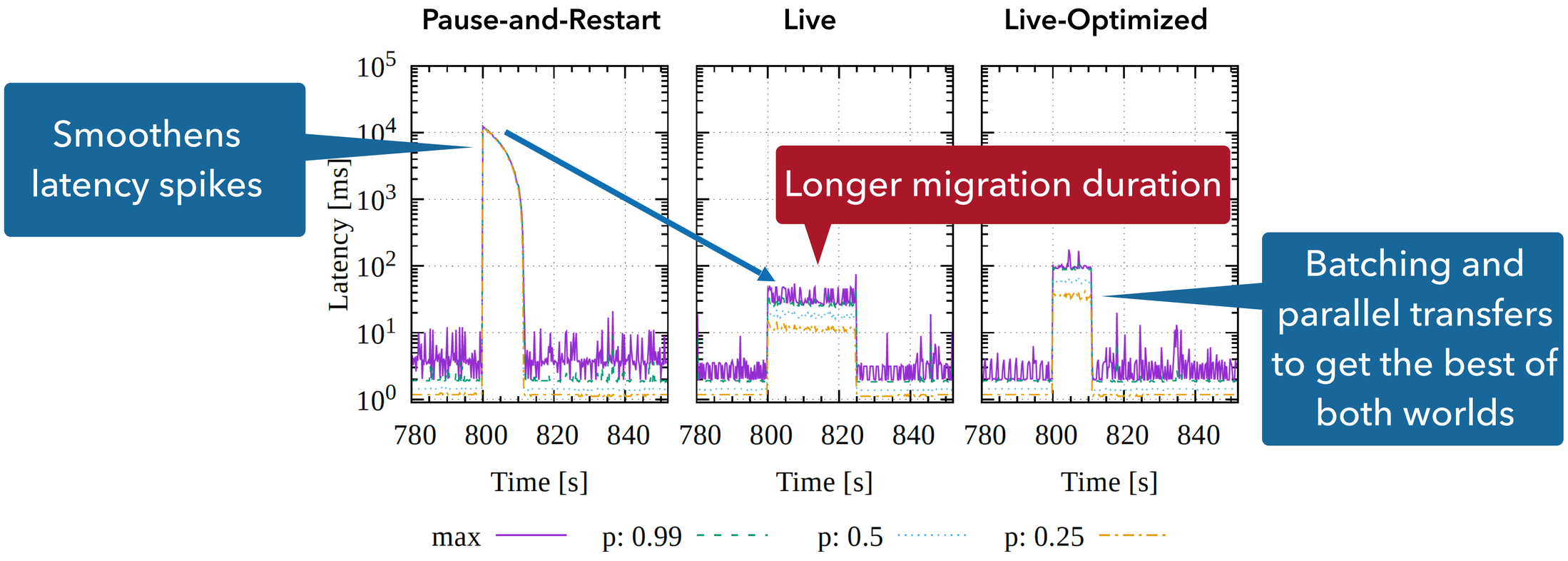
LIVE STATE MIGRATION

<u>Intuition</u>: treat state migration as **a dataflow operation** and *interleave* **fine-grained** state transfers with processing.



Helper operators manage record routing to **ensure correctness**

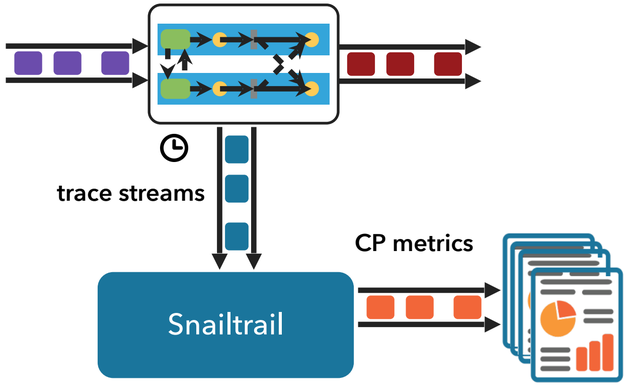
LIVE STATE MIGRATION IN ACTION



Migration of 1 billion keys, 8GB data

SUMMARY

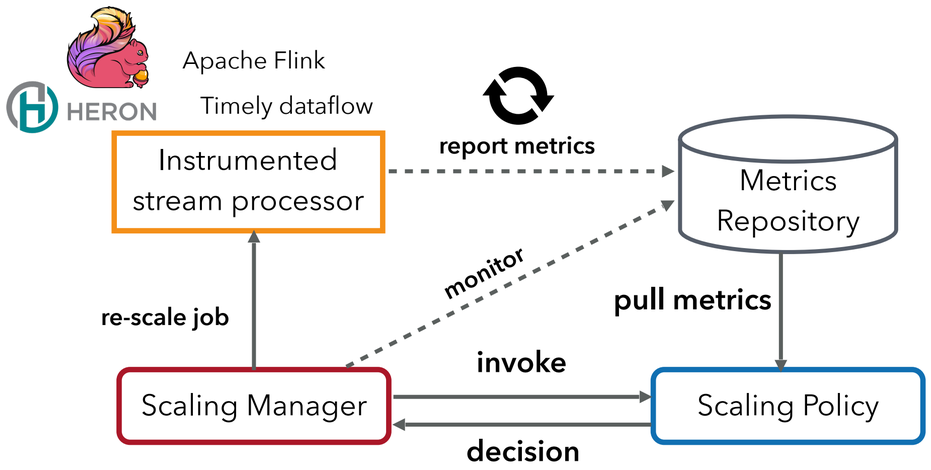
PART I: Online Critical Path Analysis



critical participation

real-time performance summaries

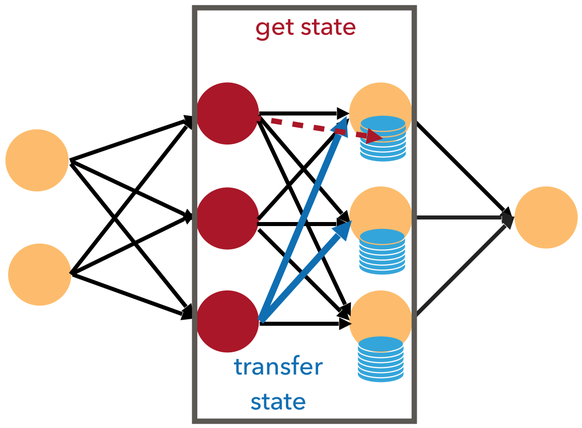
Part II: Automatic Scaling Decisions



true vs observed rates

fast convergence

Part III: Live State Migration



fluid migration

low latency spikes

https://github.com/strymon-system

COLLABORATORS










TOWARDS RECONFIGURABLE DATA STREAM PROCESSING

John Liagouris

MSR Redmond

16 May 2019