How to Measure Scalability of Distributed Stream Processing Engines?

Sören Henning and Wilhelm Hasselbring

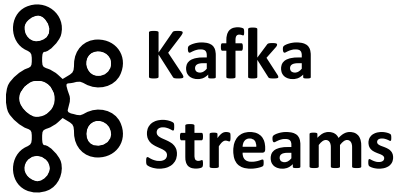
Kiel University
Christian-Albrechts-Universität zu Kiel

LTB 2021
Load Testing & Benchmarking

Big Data Stream Processing Engines

“Scales to any use case”

“… is a unified analytics engine for large-scale data processing.”



“Elastic, highly scalable, fault-tolerant”

“It is scalable, […]”

“Battle-tested at scale, […]”

Stream Processing Scalability Benchmarking

- Several performance benchmarking studies for stream processing engines
  [Karimov 2018, van Dongen 2020, Hesse 2021]

  → Throughput, latency, resource efficiency, ...
- Evaluation of performance attributes for different cluster sizes [Akidau 2013, Kulkarni 2015]
- Need for scalability benchmarking
  [van Dongen 2020,Hesse 2021]

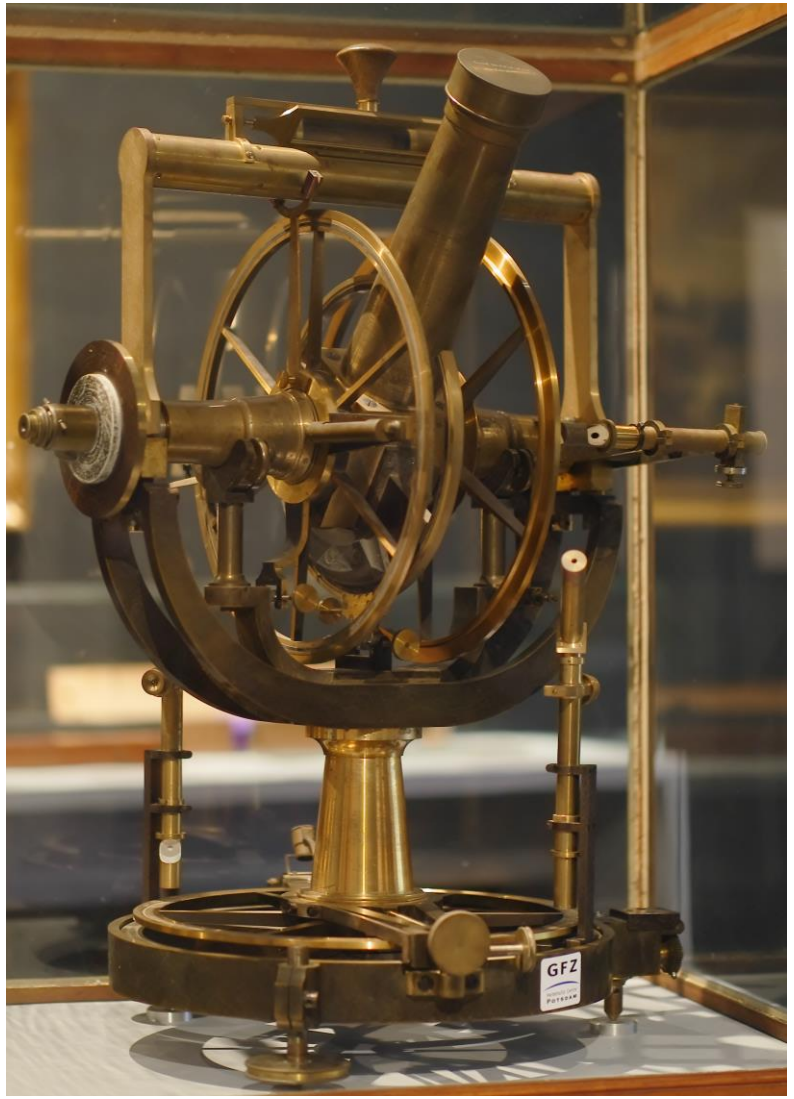
Stream Processing Scalability Benchmarking

- Several performance benchmarking studies for stream processing engines
  [Karimov 2018, van Dongen 2020, Hesse 2021]

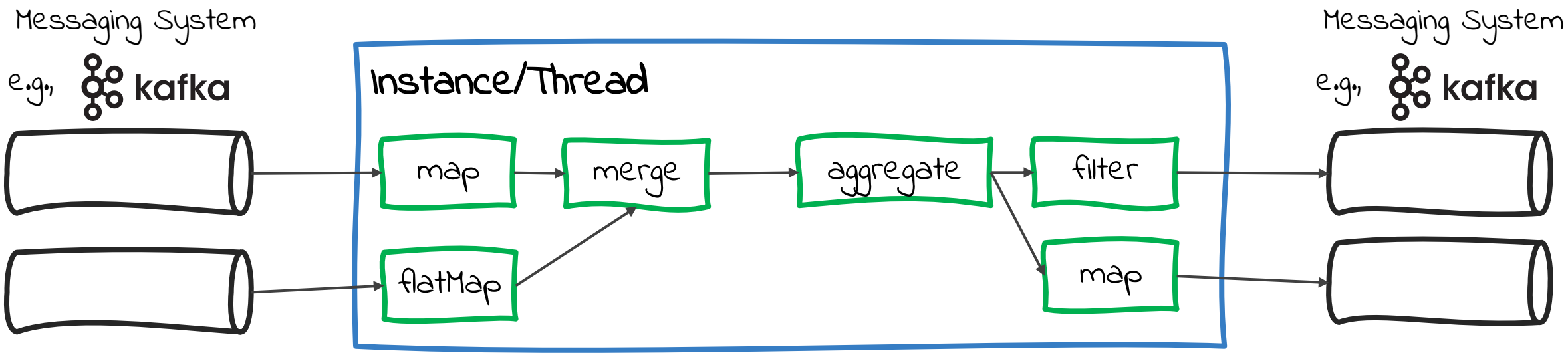
  → Throughput, latency, resource efficiency, ...
- Evaluation of performance attributes for different cluster sizes [Akidau 2013, Kulkarni 2015]
- Need for scalability benchmarking
  [van Dongen 2020,Hesse 2021]

Theodolite Benchmarking Tool

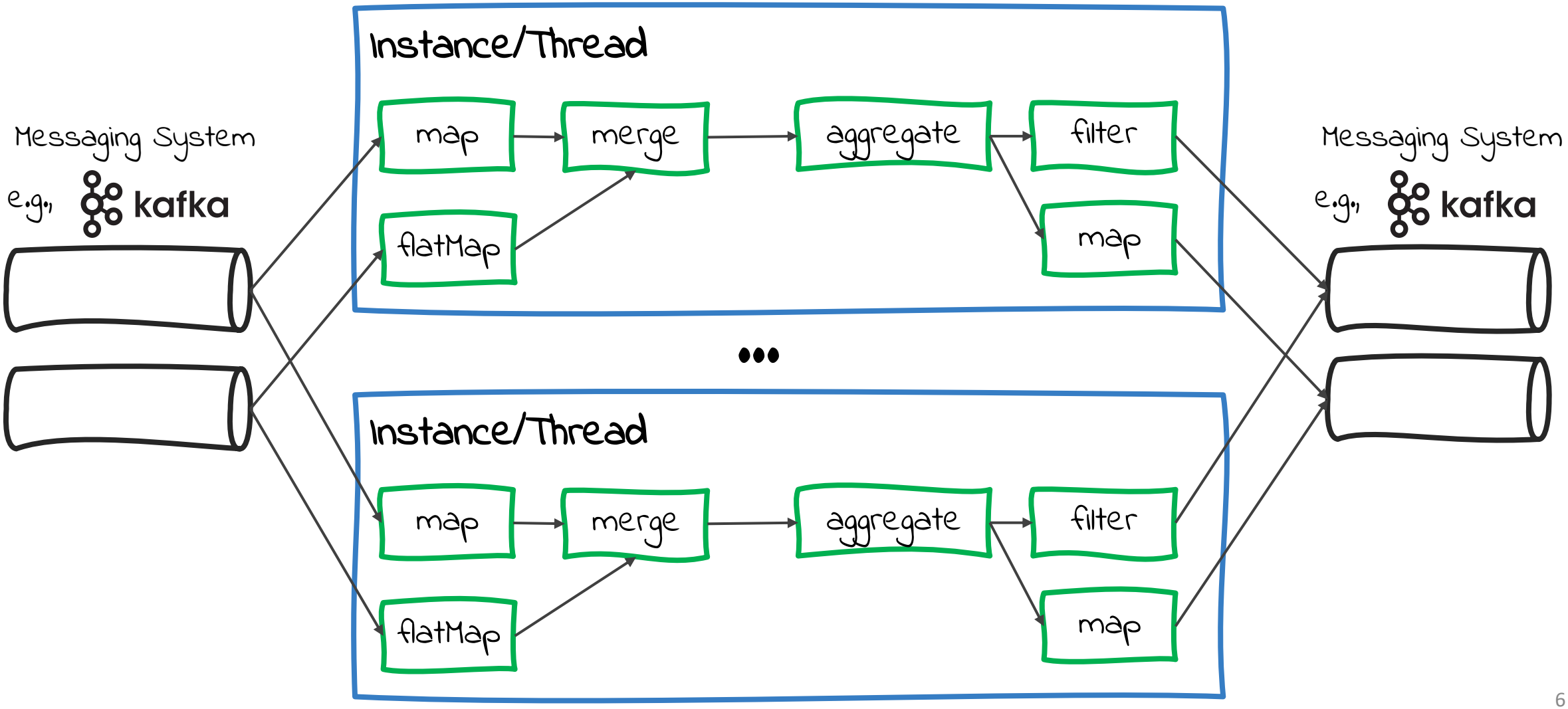
[Henning 2021]



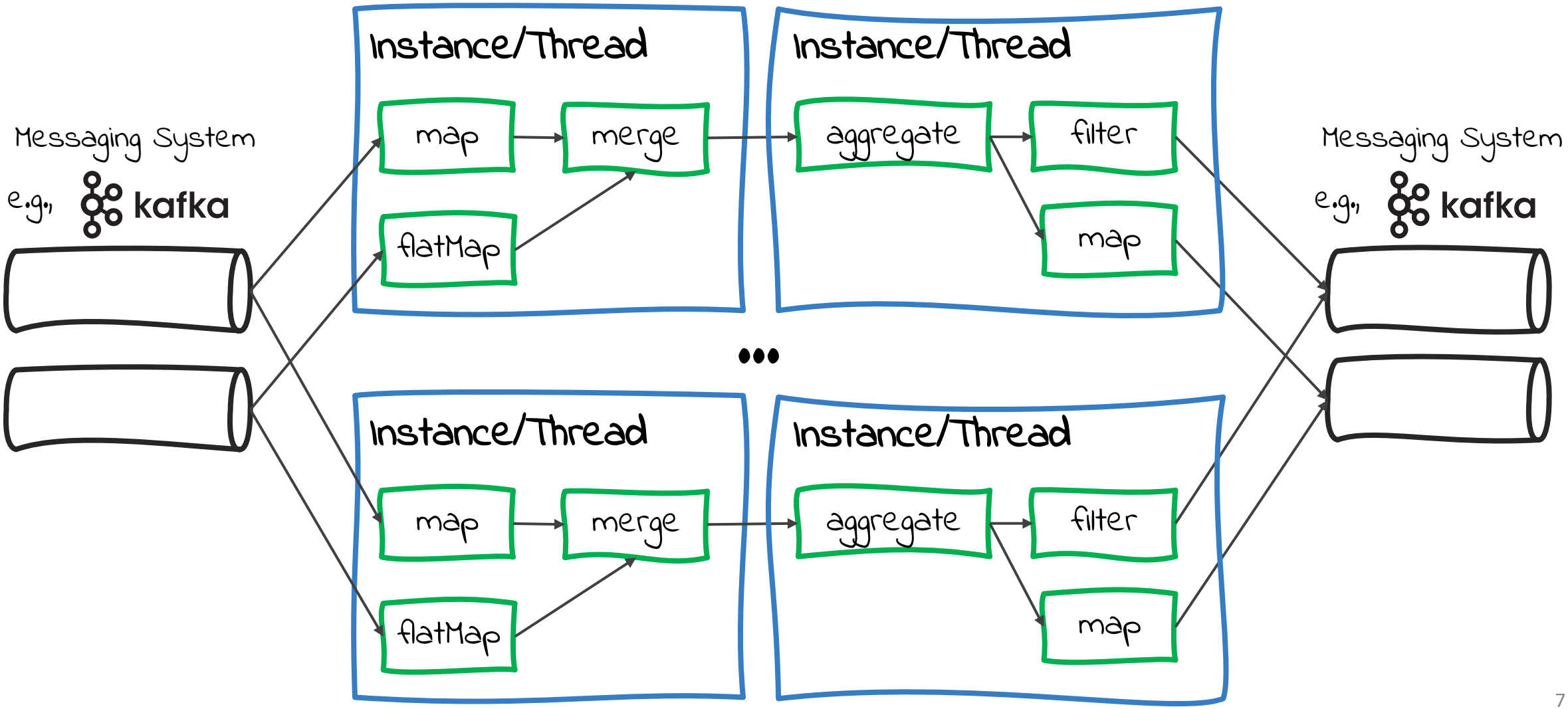
Distributed Stream Processing



Distributed Stream Processing



Distributed Stream Processing



Scalability in Cloud Computing

> *Scalability is the ability of [a] system to sustain increasing workloads by making use of additional resources […].*
>
> [Herbst 2013]

Scalability in Cloud Computing

> *Scalability is the ability of [a] system to sustain increasing workloads by making use of additional resources […].*
>
> [Herbst 2013]

Load intensity is the input variable a system is subject to. Scalability is evaluated within a range of load intensities

Service levels objectives (SLOs) are measurable quality criteria that have to be fulfilled for every load intensity.

Provisioned resources can be increased to meet the SLOs if load intensities increase.

[Weber 2014]

Scalability in Stream Processing

Load intensity $\hat{L} \subseteq L$

- Messages per second
- Message frequency
- Different message types (keys)
- …

Provisioned resources R

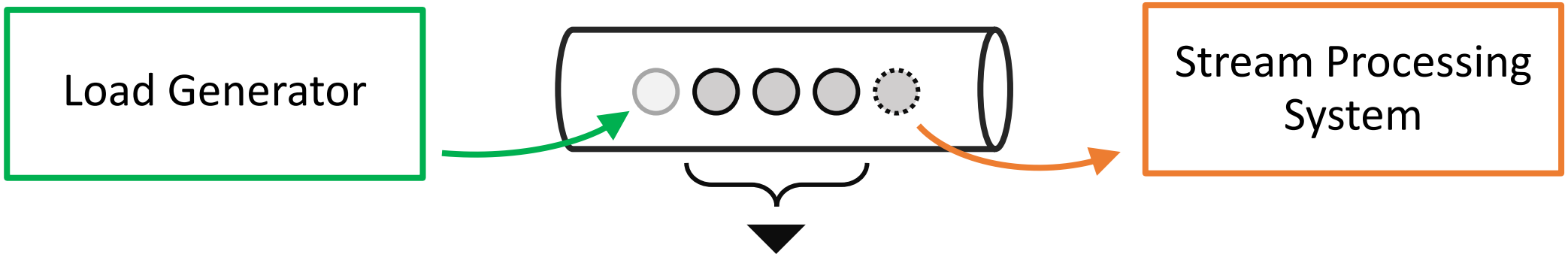
- Instances (e.g., Kubernetes Pods)
- Threads
- VMs / configurations (S → M → L)
- …

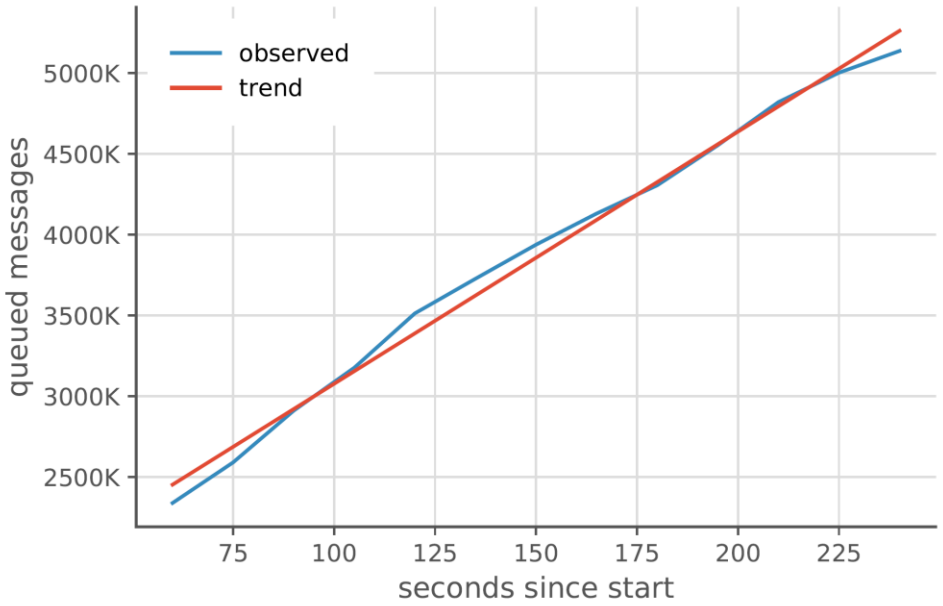
Service levels objectives (SLOs)

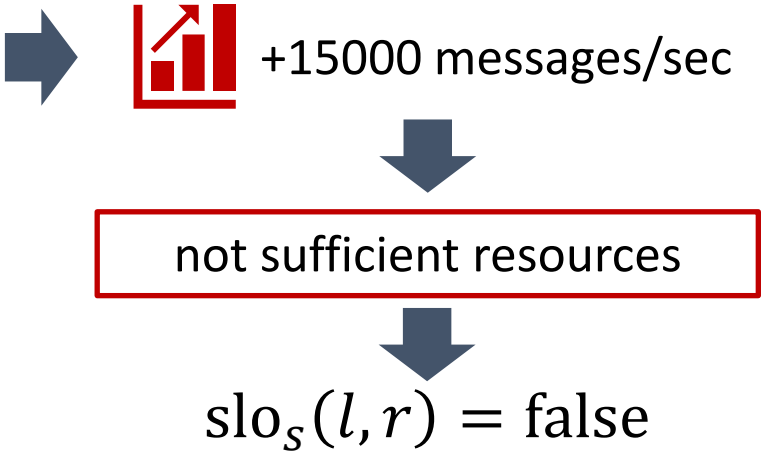
- *Lag Trend* (next slide)
- …

$$\forall s \in S\colon\ \mathrm{slo}_s\colon L \times R \to \{\mathrm{false}, \mathrm{true}\}$$

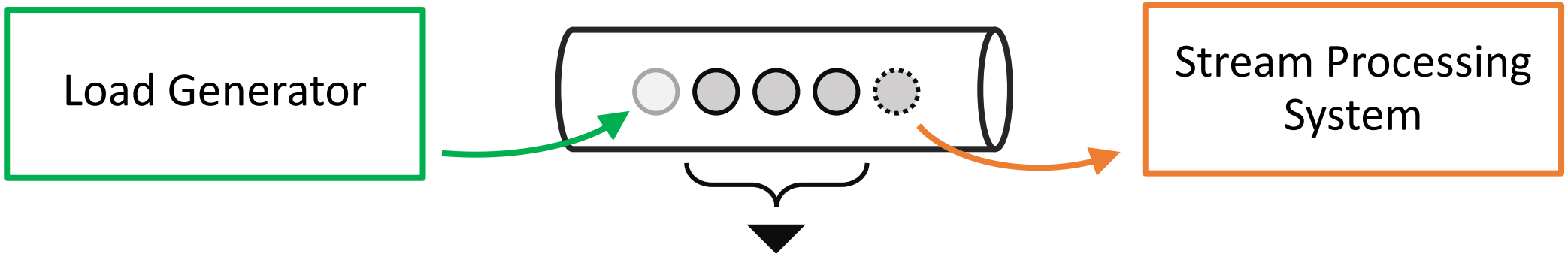
Lag Trend Metric as SLO

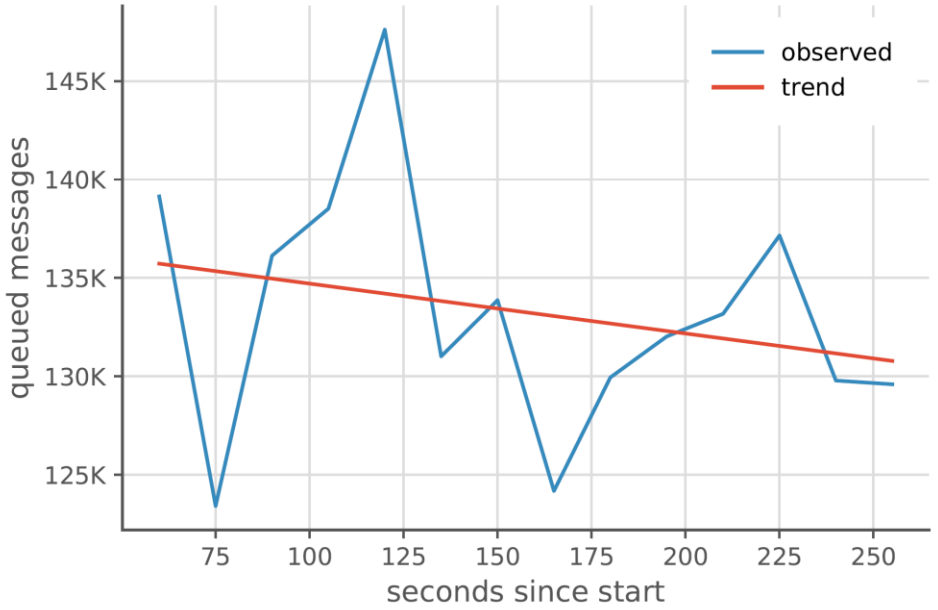


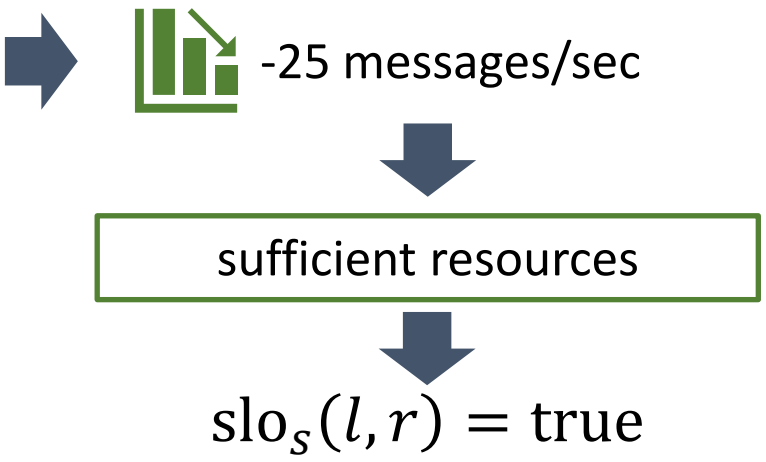




Lag Trend Metric as SLO







Scalability in Stream Processing

Load intensity $\hat{L} \subseteq L$

- Messages per second
- Message frequency
- Different message types (keys)
- …

Provisioned resources R

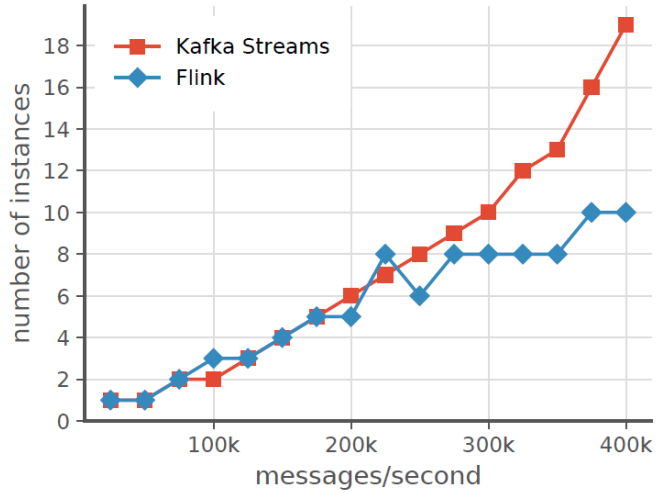
- Instances (e.g., Kubernetes Pods)
- Threads
- VMs / configurations (S → M → L)
- …

Service levels objectives (SLOs)

- *Lag Trend*
- …

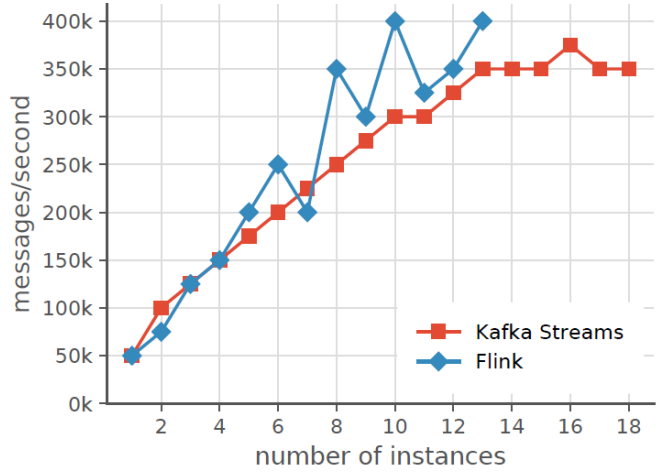
$$\forall s \in S\colon\ \mathrm{slo}_s\colon L \times R \to \{\mathrm{false}, \mathrm{true}\}$$

Scalability Metrics



Resource demand metric

$$\forall\, l \in \widehat{L} : \mathrm{demand}(l) \;=\; \min\left\{ l \in \widehat{R} \;\middle|\; \forall s \in S : \mathrm{slo}_s(l, r) = \mathrm{true} \right\}$$

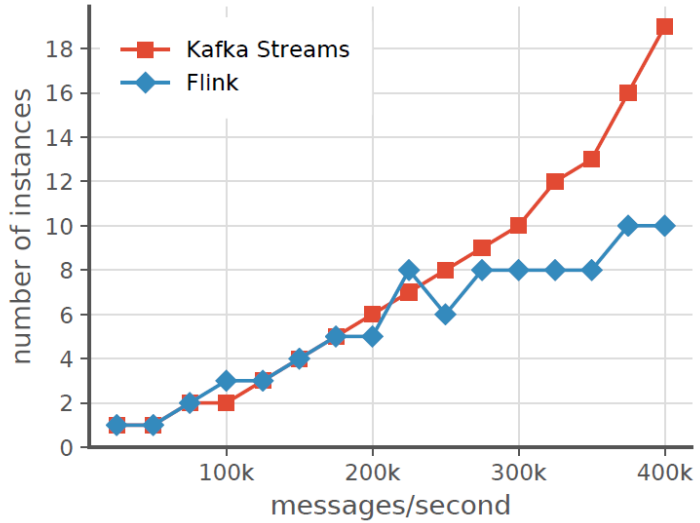


Load capacity metric

$$\forall\, r \in R : \mathrm{capacity}(r) \;=\; \max\left\{ l \in \widehat{L} \;\middle|\; \forall s \in S : \mathrm{slo}_s(l, r) = \mathrm{true} \right\}$$

based on [Henning 2021]

Scalability as a Function



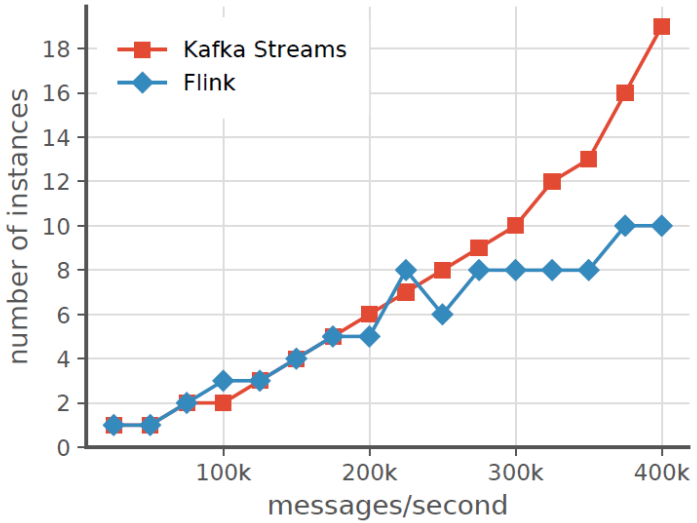
➡ Capacity does not grow at constant rate

[Sanders 2015, Brataas 2017]

vs.

speed-up: XX.X %

Scalability as a Function



vs.

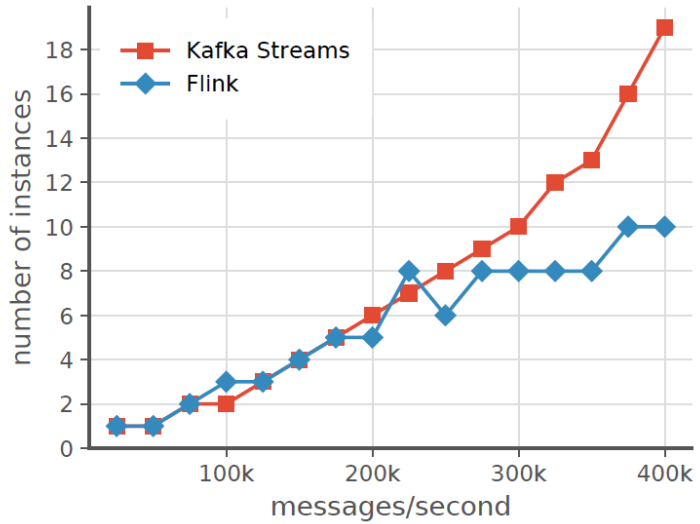
speed-up: XX.X %

➡ Capacity does not grow at constant rate

[Sanders 2015, Brataas 2017]

How to rank different systems?

Scalability as a Function



vs.

speed-up: XX.X %

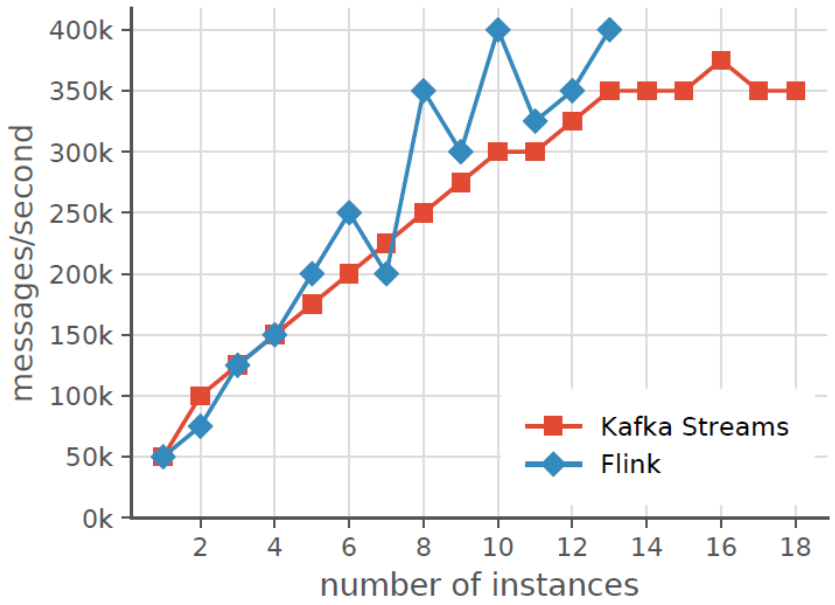
➡ Capacity does not grow at constant rate
[Sanders 2015, Brataas 2017]

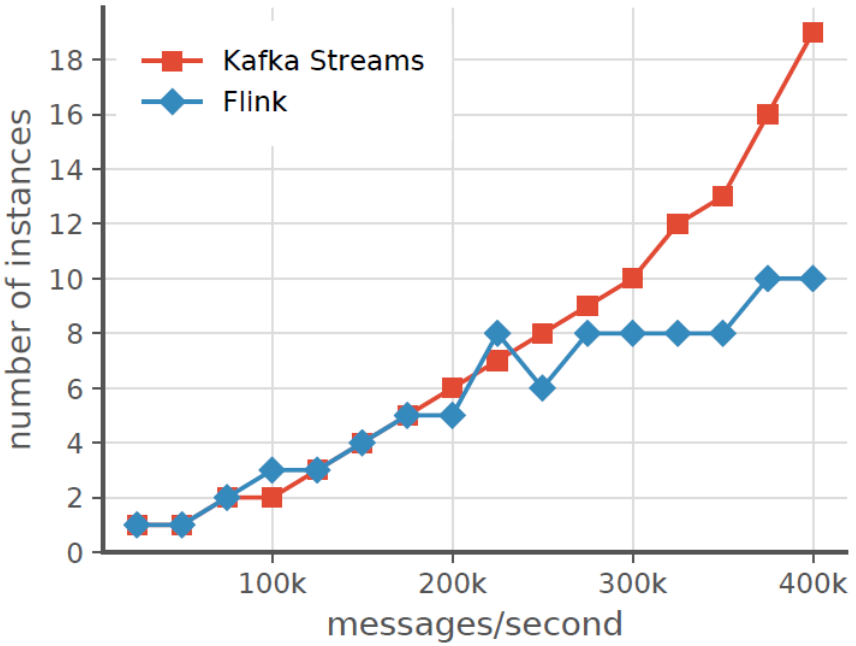
How to rank different systems?

- Visual comparison
- Clustering similar functions
- Derivative or axis intersection
- Universal Scalability Law [Gunther 2015]
  → Derive non-linear rational function
  → Contention and coherency coefficients
  → Applicable to stream processing?

Function of Load vs. Function of Resources

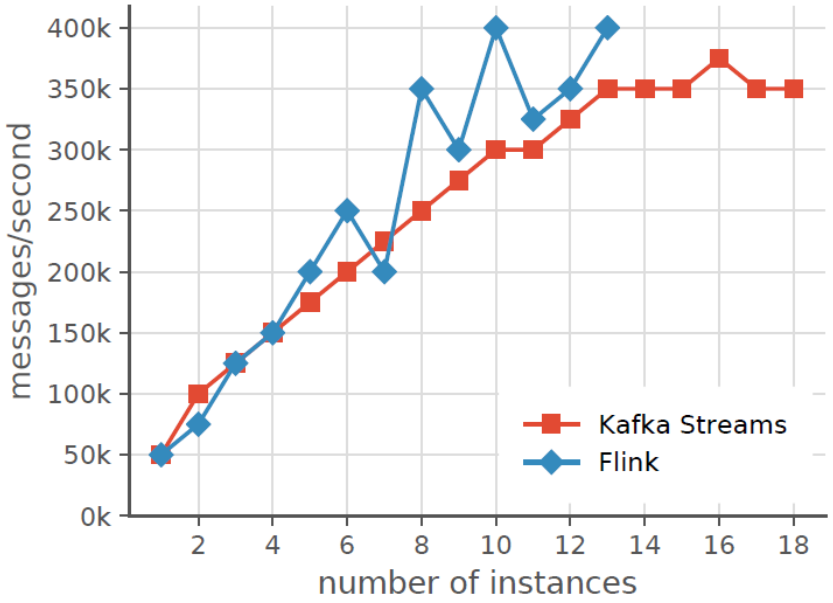
More common in literature



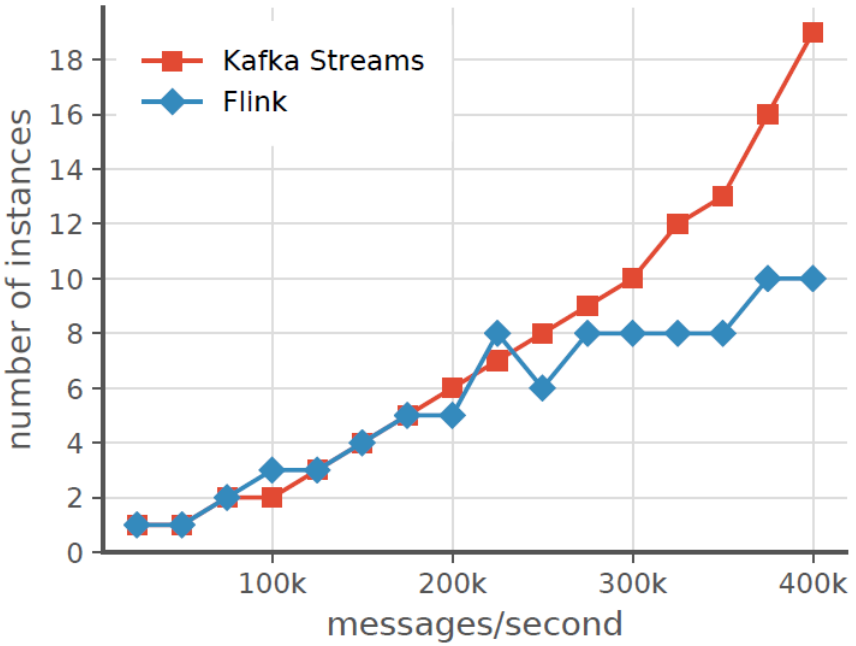


Function of Load vs. Function of Resources

More common in literature

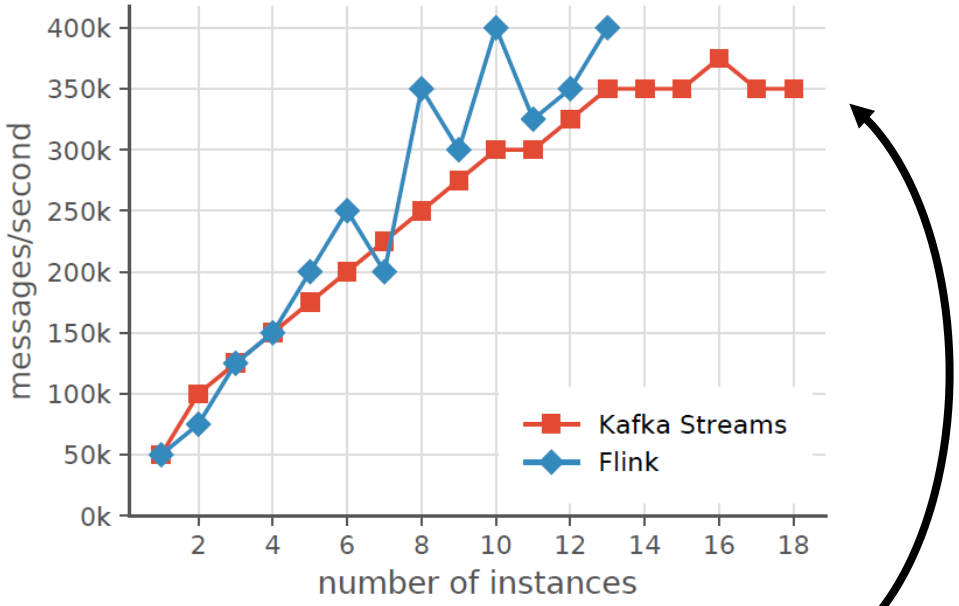


Aligned to scalability definitions:
Load is input variable



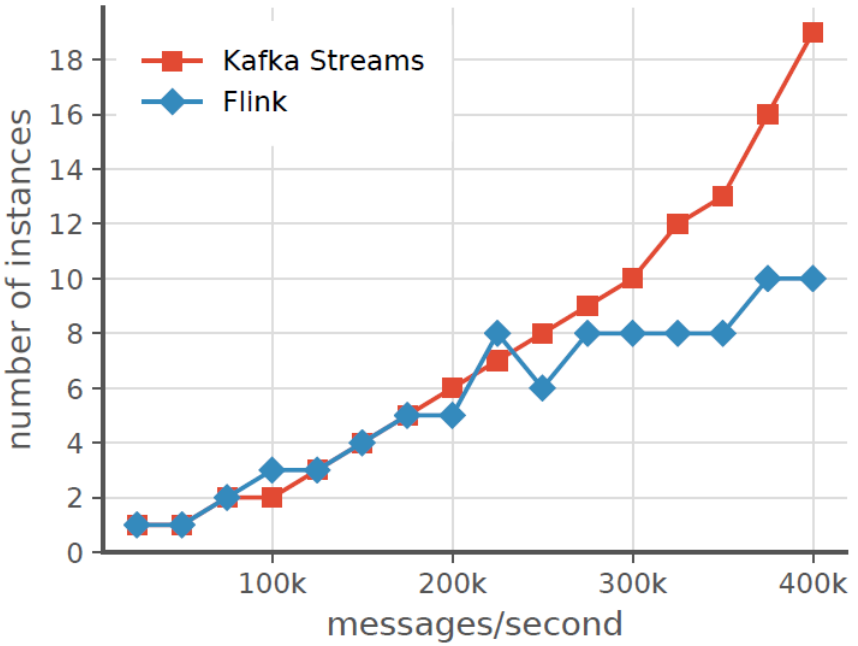
Function of Load vs. Function of Resources

More common in literature



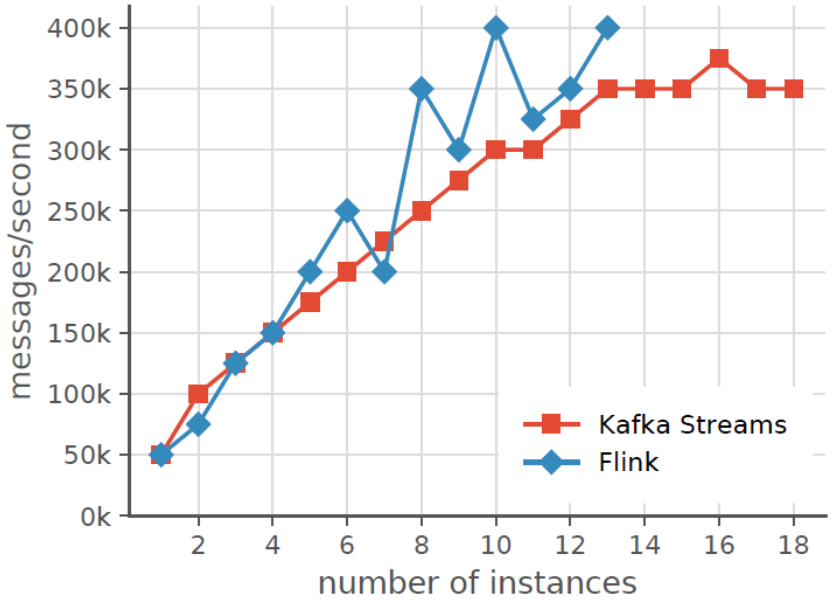
Shows drops in capacity

Aligned to scalability definitions:
Load is input variable



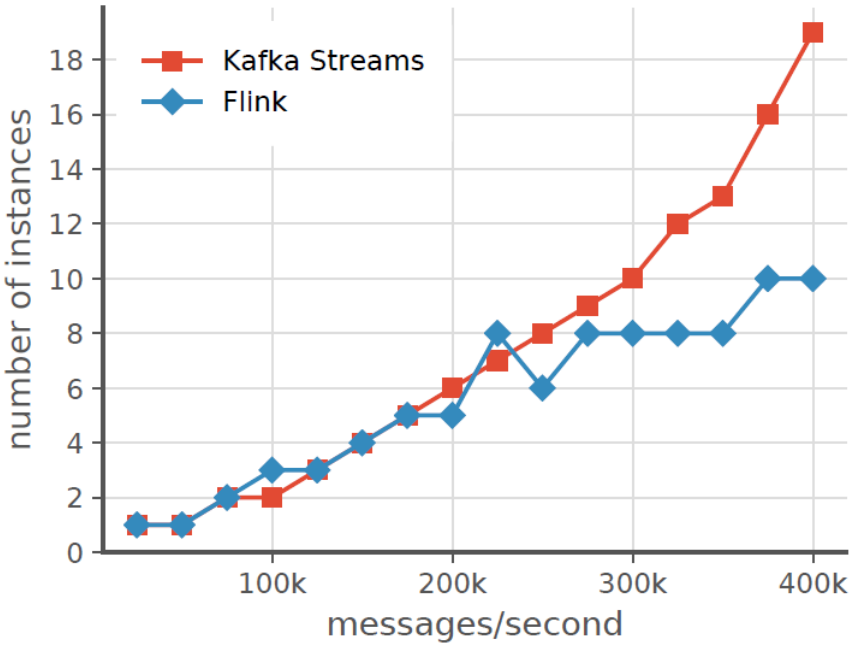
Function of Load vs. Function of Resources

More common in literature



Shows drops in capacity

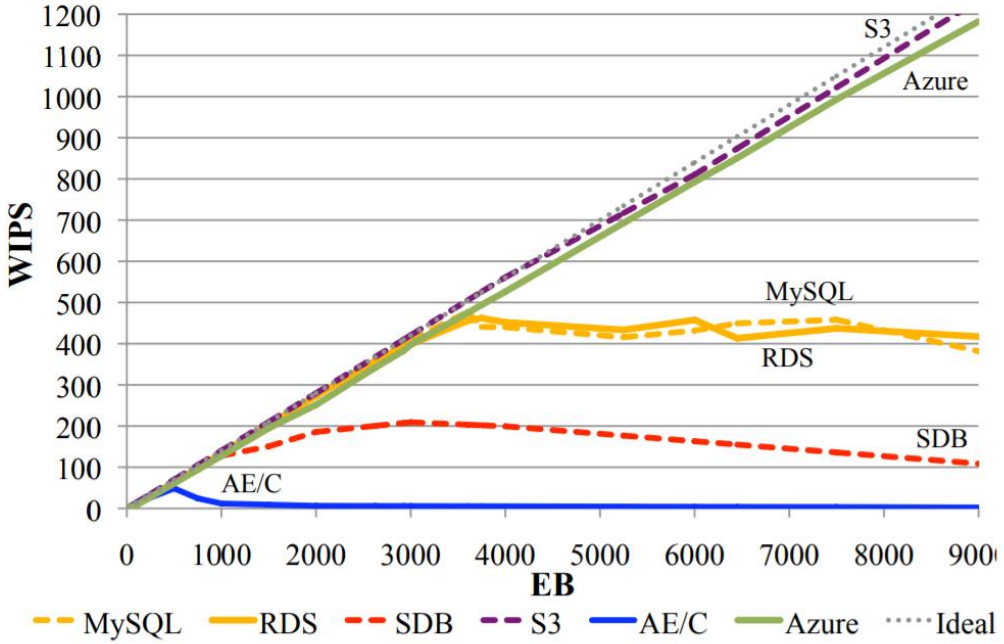
Aligned to scalability definitions:
Load is input variable



Allow for normalization [Gunther 2015]

→ exclude resource efficiency

Resources as a Function of Load

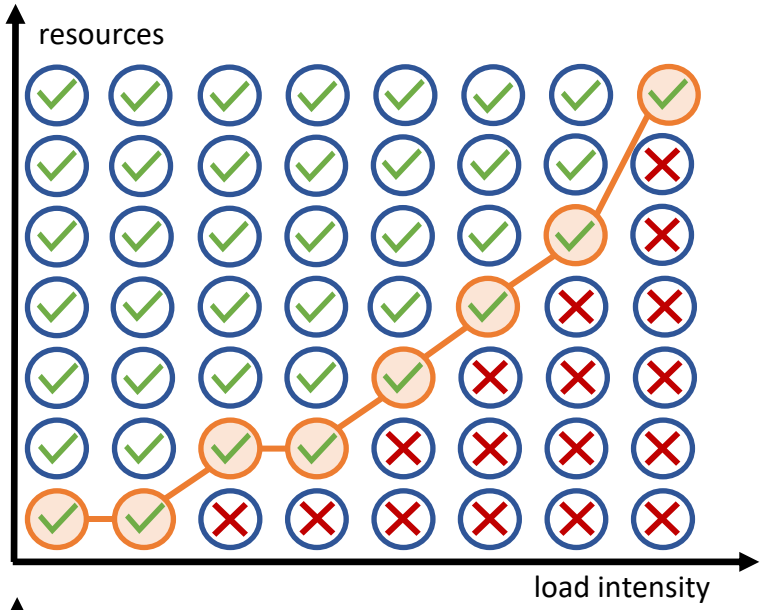


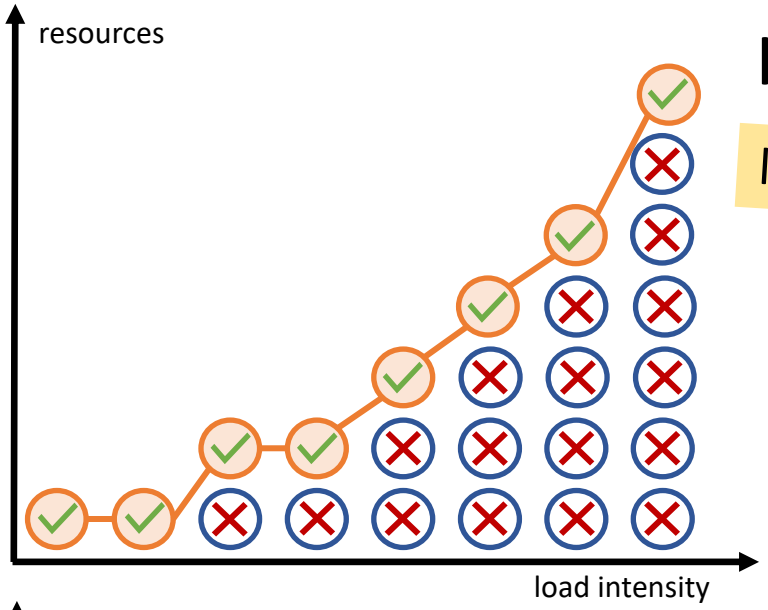
[Kossmann 2010]

- No binary decision on SLOs but instead evaluating the service level as a function of load
- Only feasible when auto-scaled in the background

  ➔ Contains evaluation of elasticity

Theodolite Measurement Method



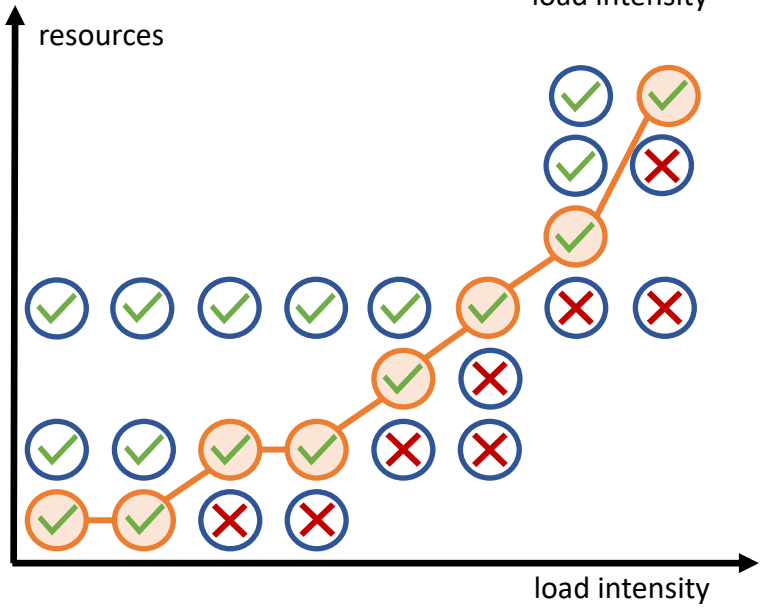


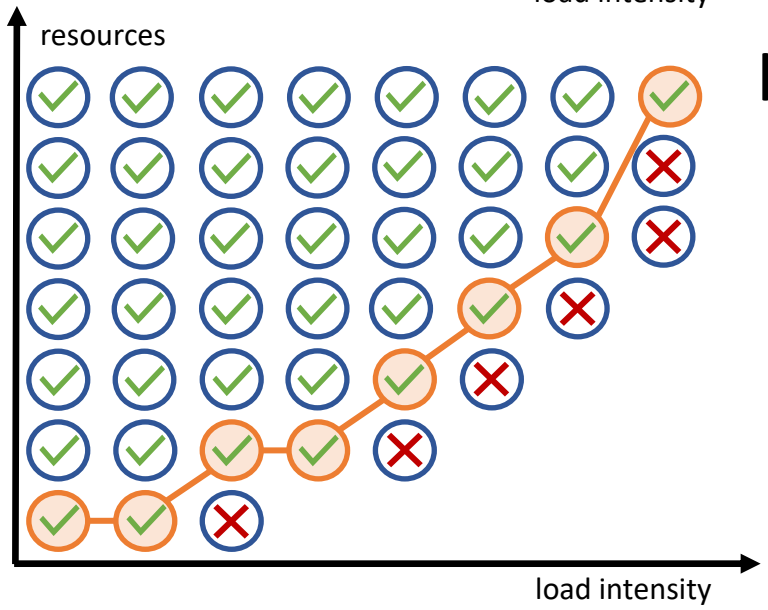
H1

linear search

H2

binary search



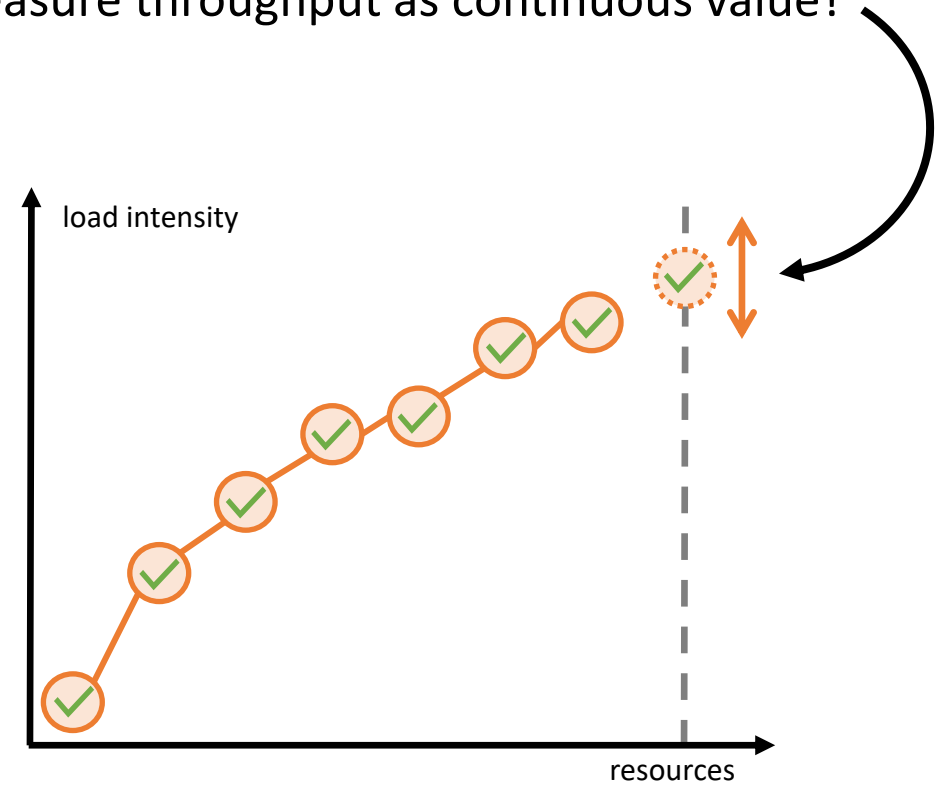


H3

assumption:
resource demand monotonically increasing

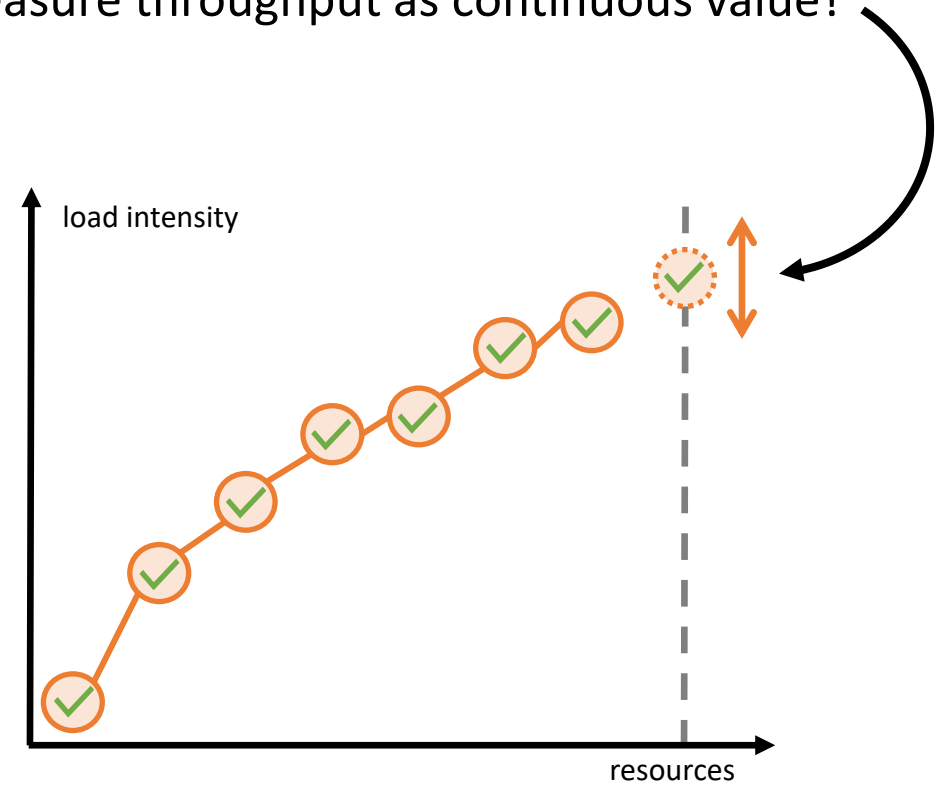
Capacity as Discrete Values

Measure throughput as continuous value?



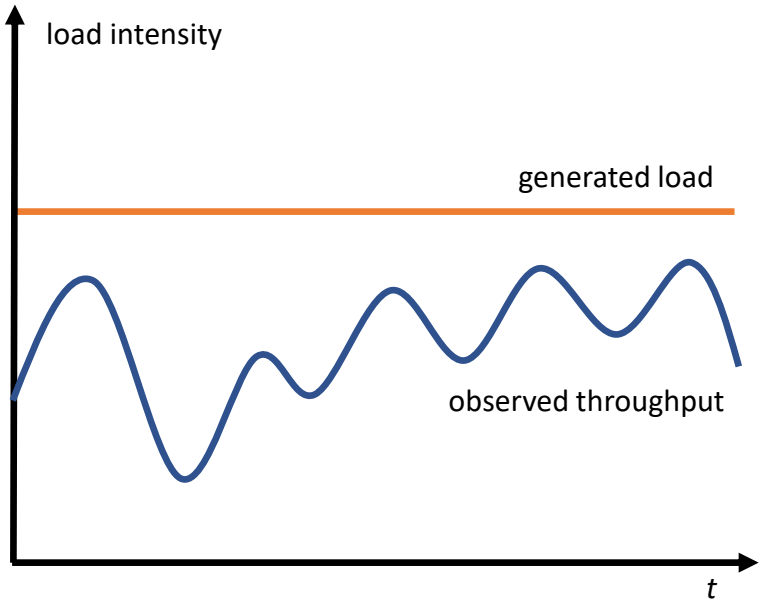
Capacity as Discrete Values

Measure throughput as continuous value?



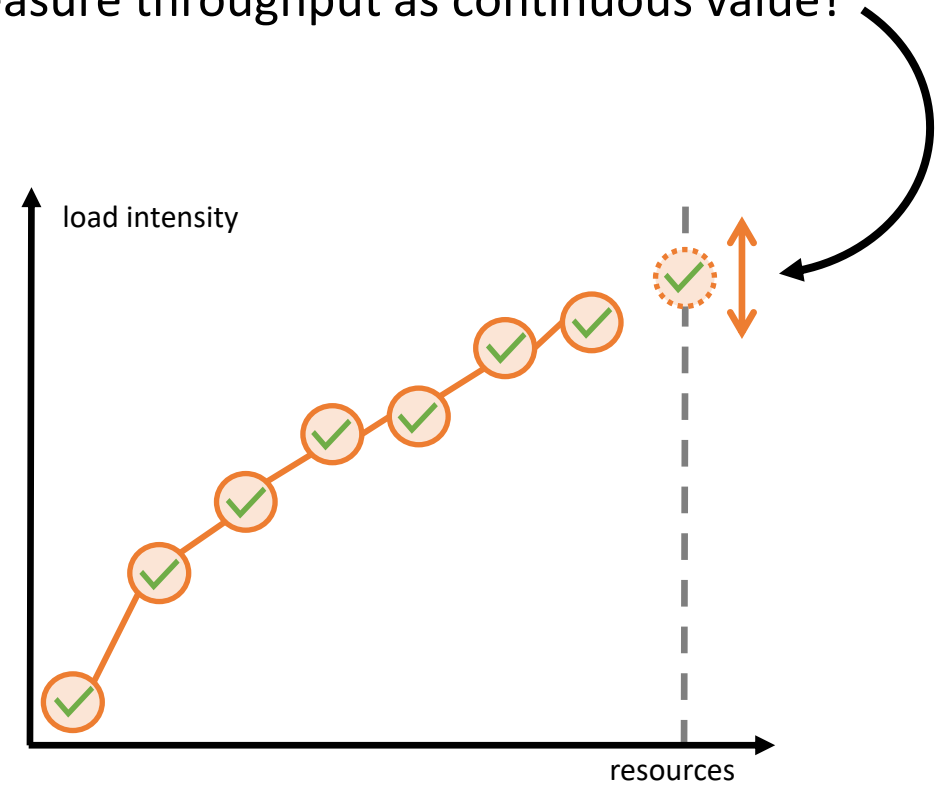
Alternative 1: [Karakaya 2017], [Nasiri 2019]

- Generate constant load
- Measure throughput



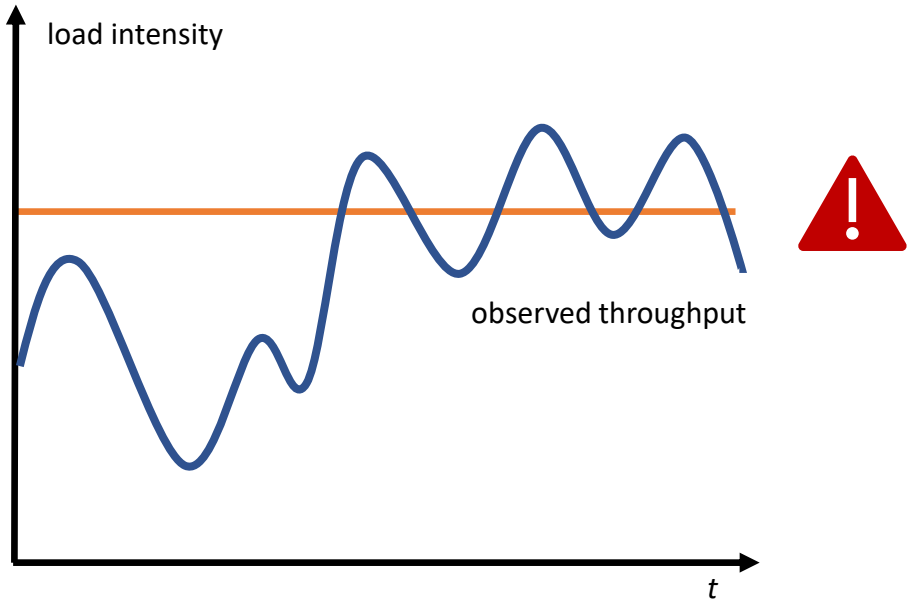
Capacity as Discrete Values

Measure throughput as continuous value?



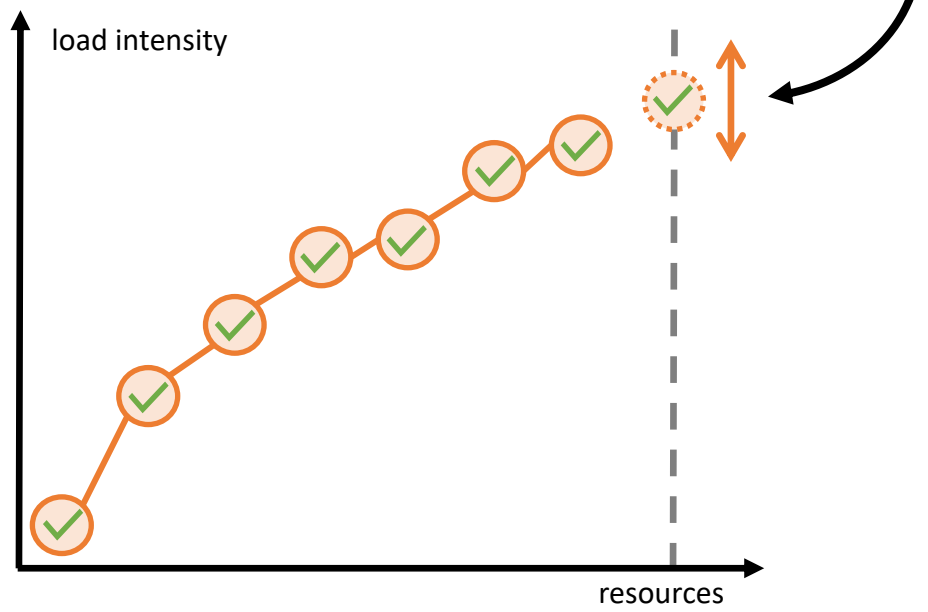
Alternative 1:

- Generate constant load
- Measure throughput



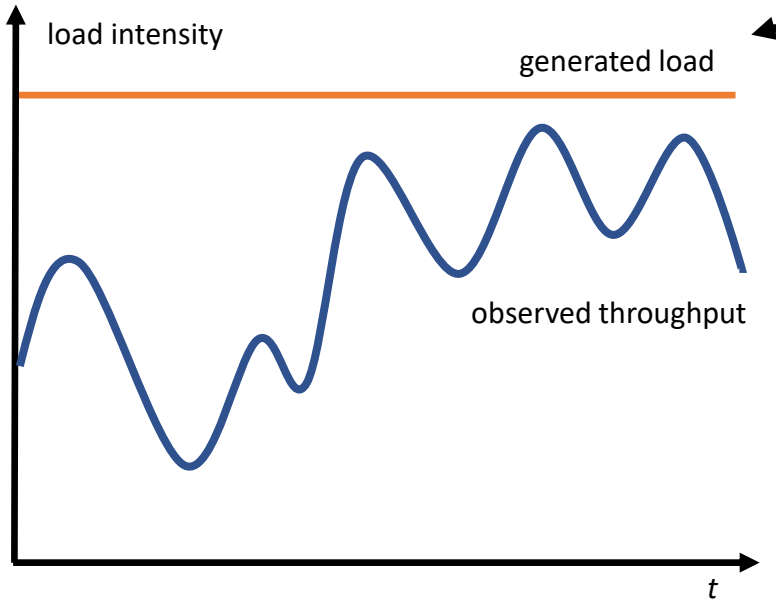
Capacity as Discrete Values

Measure throughput as continuous value?



Alternative 1:

- Generate constant load
- Measure throughput



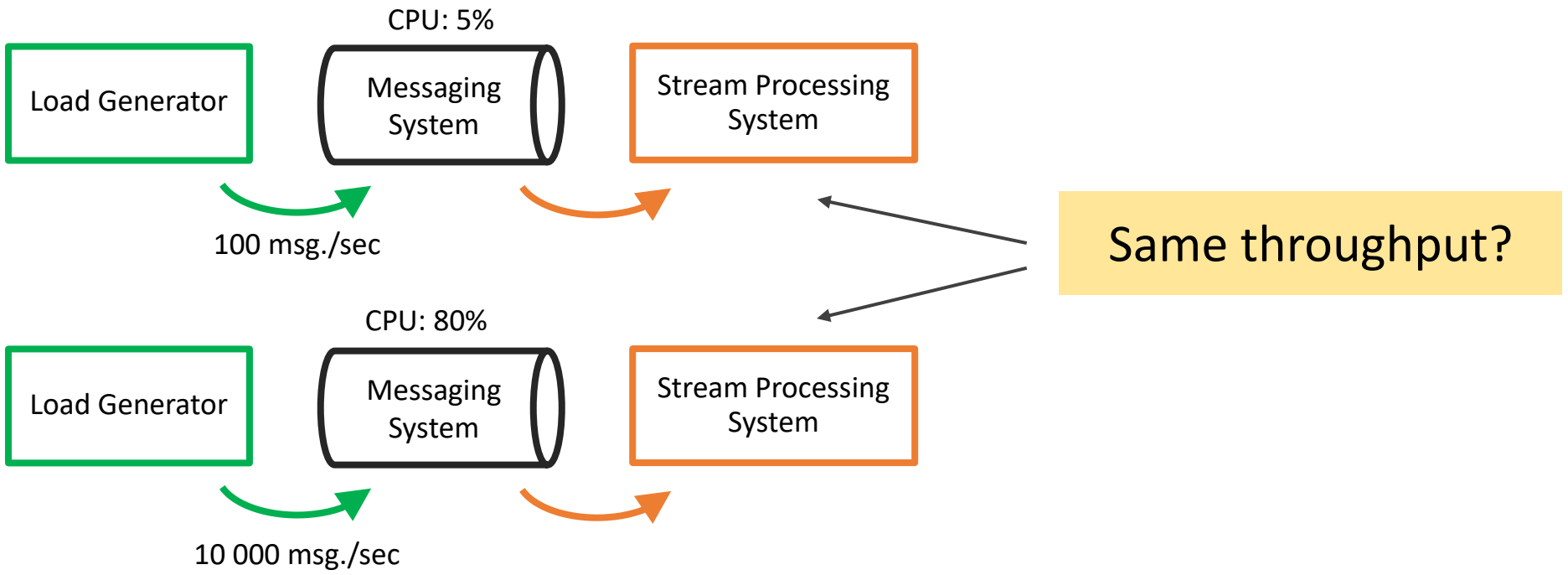
What constant load?

Capacity as Discrete Values

Measure throughput as continuous value?

Alternative 1:

- Generate constant load
- Measure throughput

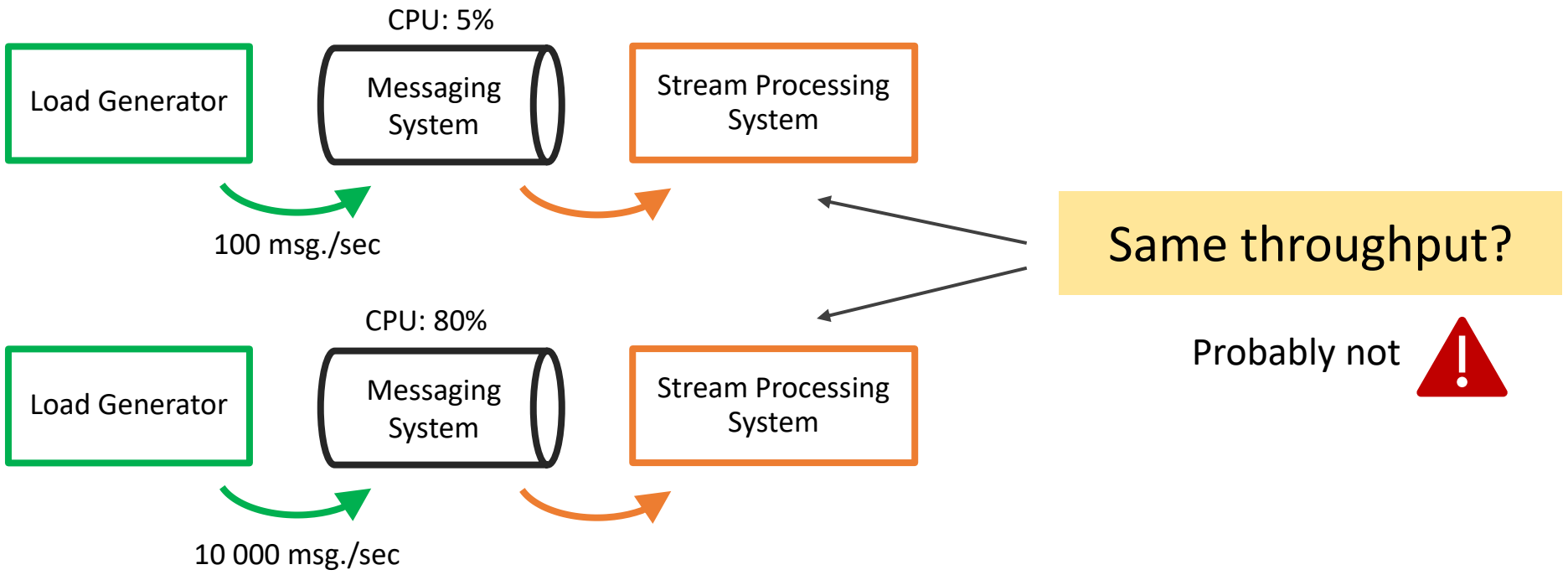


Capacity as Discrete Values

Measure throughput as continuous value?

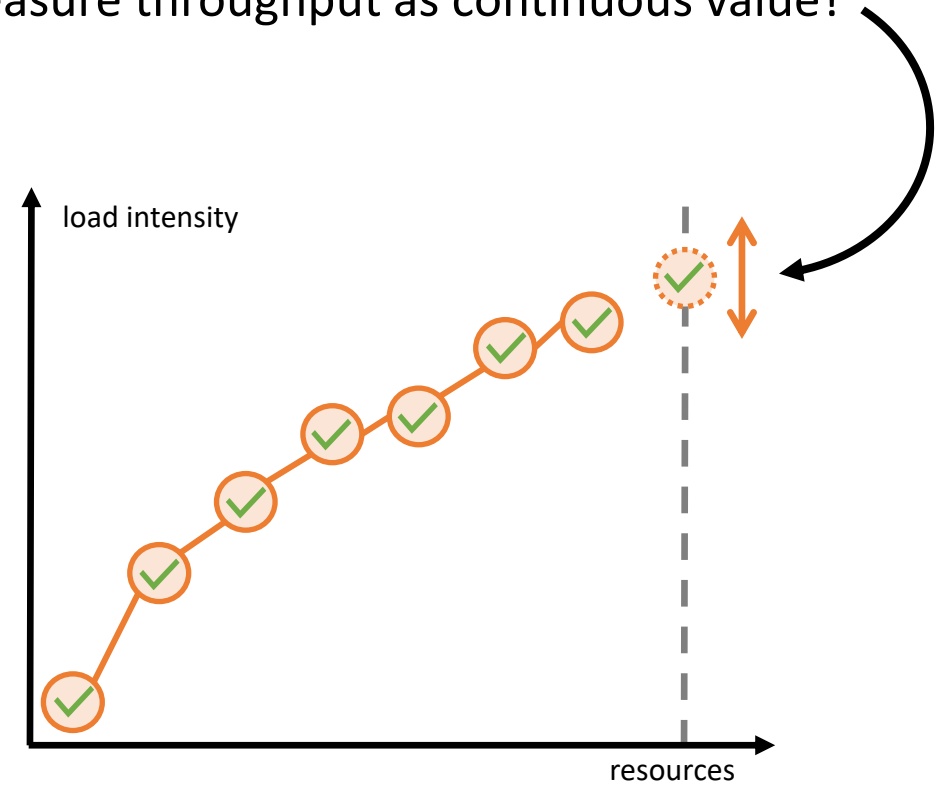
Alternative 1:

- Generate constant load
- Measure throughput



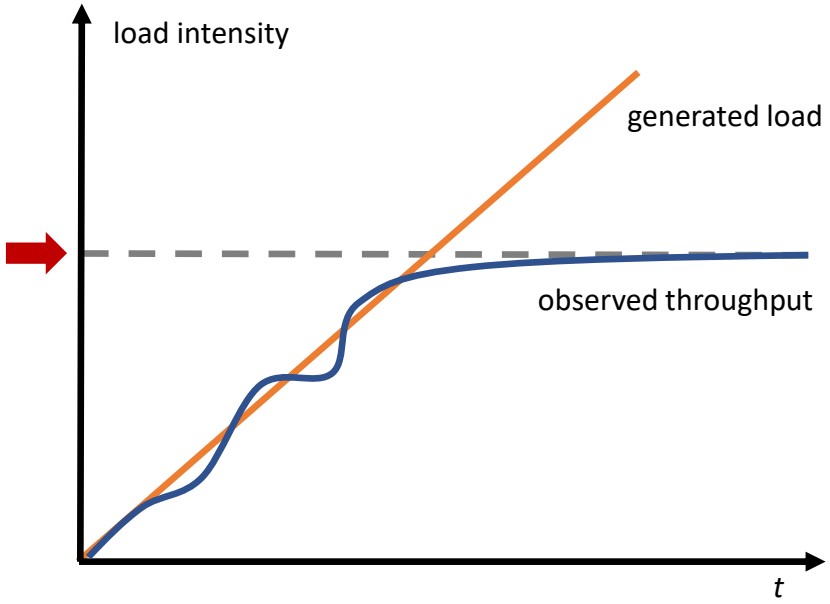
Capacity as Discrete Values

Measure throughput as continuous value?



Alternative 2: [Karimov 2017]

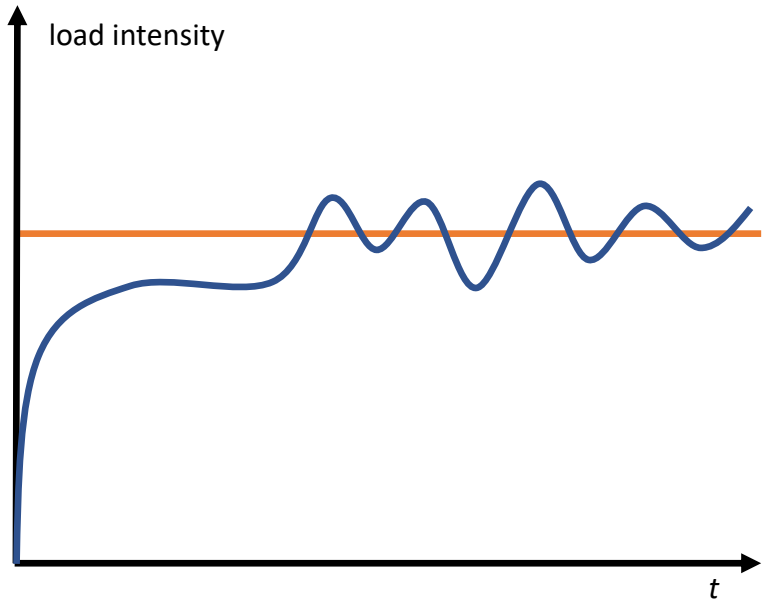
- Steadily increase load
- Determine when SLOs are not met anymore



Capacity as Discrete Values

Measure throughput as continuous value?

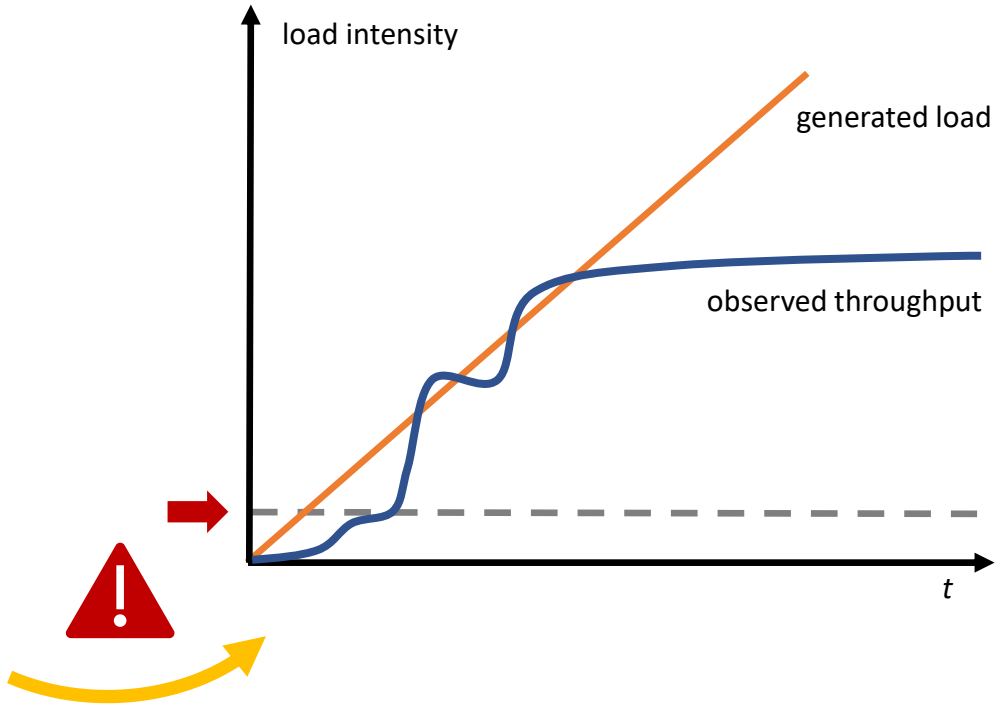
Even with constant load



[Henning 2021]

Alternative 2:

- Steadily increase load
- Determine when SLOs are not met anymore

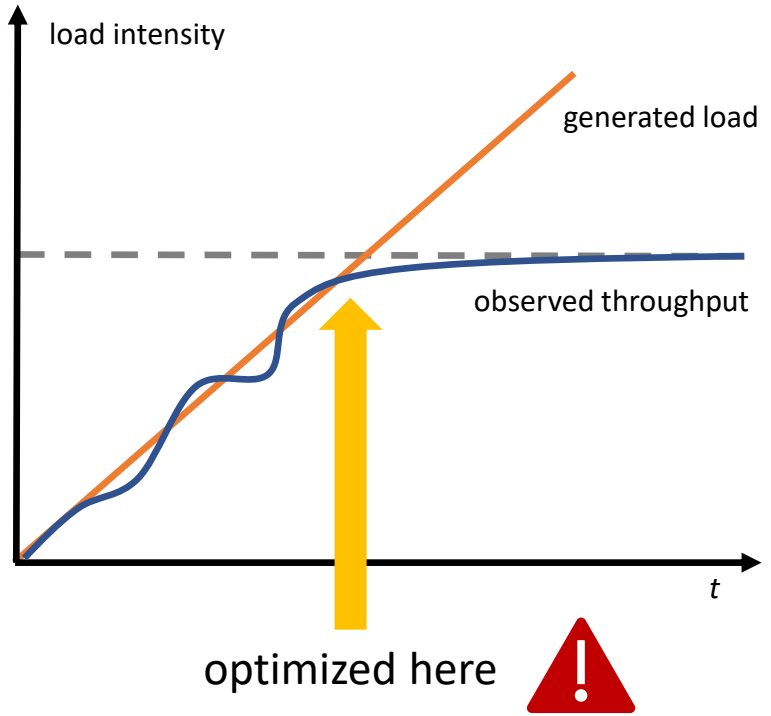


Capacity as Discrete Values

Measure throughput as continuous value?

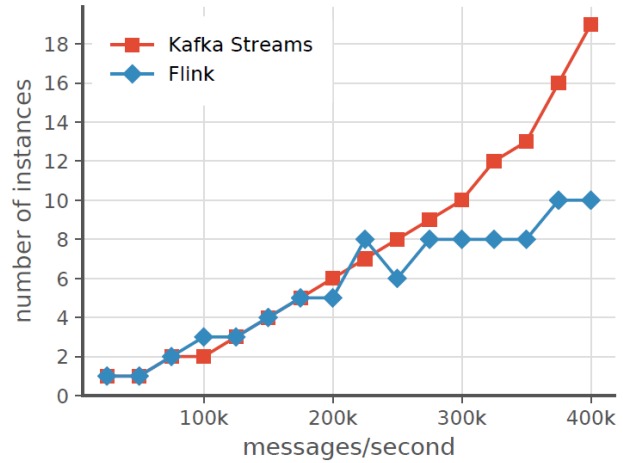
Alternative 2:

- Steadily increase load
- Determine when SLOs are not met anymore

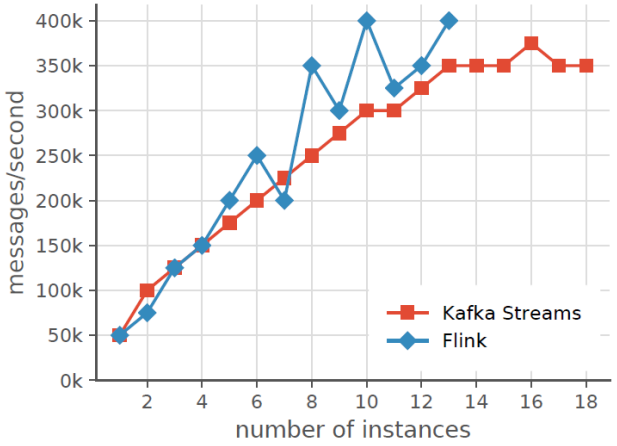


Conclusions

Resource demand metric



Load capacity metric



Scalability defined based on:

- Load intensities
- Resources
- SLOs (e.g., *Lag Trend*)

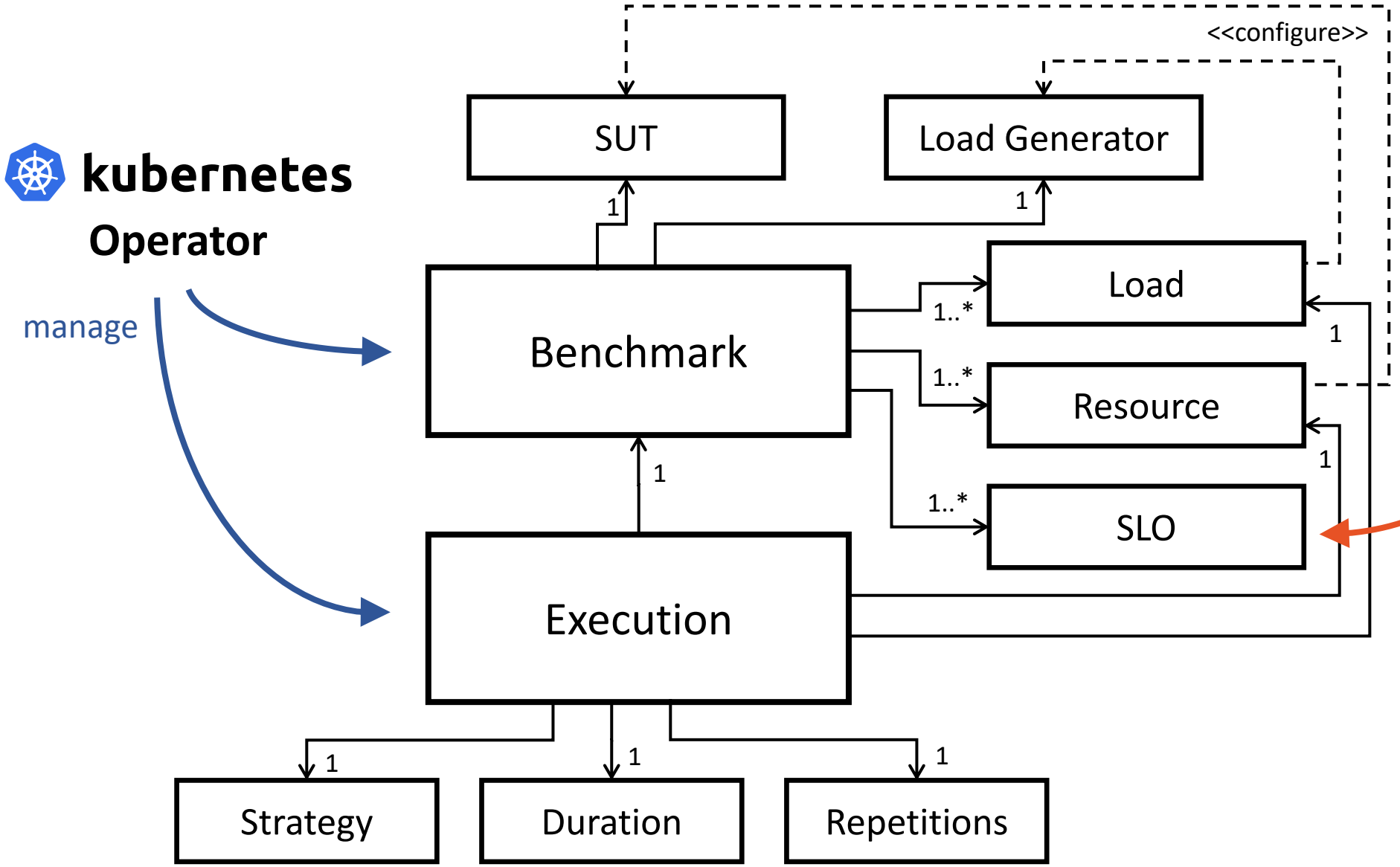
Scalability as a Function

Remand and capacity as discrete values

Isolated experiments for different load and resource combinations

Application

Outlook: *Theodolite* Benchmarking Tool



Prometheus

check SLOs

https://github.com/cau-se/theodolite

References

[Akidau 2013] T. Akidau, A. Balikov, K. Bekiroglu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, S. Whittle. 2013. Millwheel: fault-tolerant stream process-ing at internet scale. In Proc. VLDB Endow. 6.

[Brataas 2017] G. Brataas, N. Herbst, S. Ivanšek, and J. Polutnik. 2017. Scalability Analysis of Cloud Software Services. In Proc. International Conference on Autonomic Computing.

[Gunther 2015] N. J. Gunther, P. Puglia, and K. Tomasette. 2015. Hadoop Superlinear Scalability. Commun. ACM 58, 4 (2015).

[Henning 2021] S. Henning and W. Hasselbring. 2021. Theodolite: Scalability Benchmarking of Distributed Stream Processing Engines in Microservice Architectures. Big Data Research 25 (2021), 100209.

[Herbst 2013] N. R. Herbst, S. Kounev, and R. Reussner. 2013. Elasticity in Cloud Computing: What It Is, and What It Is Not. In Proc. Int. Conference on Autonomic Computing.

[Hesse 2021] G. Hesse, C. Matthies, M. Perscheid, M. Uflacker, and H. Plattner. 2021. ESPBench: The Enterprise Stream Processing Benchmark. In Proc. ACM/SPEC International Conference on Performance Engineering.

References

[Karakaya 2017] Z. Karakaya, A. Yazici, and M. Alayyoub. 2017. A Comparison of Stream Processing Frameworks. In Proc. International Conference on Computer and Applications.

[Karimov 2018] J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen, and V. Markl. 2018. Benchmarking Distributed Stream Data Processing Systems. In Proc. International Conference on Data Engineering.

[Kossmann 2010] D. Kossmann, T. Kraska, and S. Loesing. 2010. An Evaluation of Alternative Architectures for Transaction Processing in the Cloud. In Proc. SIGMOD International Conference on Management of Data.

[Kulkarni 2015] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J.M. Patel, K. Ramasamy, S. Taneja. 2015. Twitter Heron: stream processing at scale. In Proc. ACM SIGMOD International Conference on Management of Data.

[Nasiri 2019] H. Nasiri, S. Nasehi, and M. Goudarzi. 2019. Evaluation of distributed stream processing frameworks for IoT applications in Smart Cities. Journal of Big Data 6, 52 (2019).

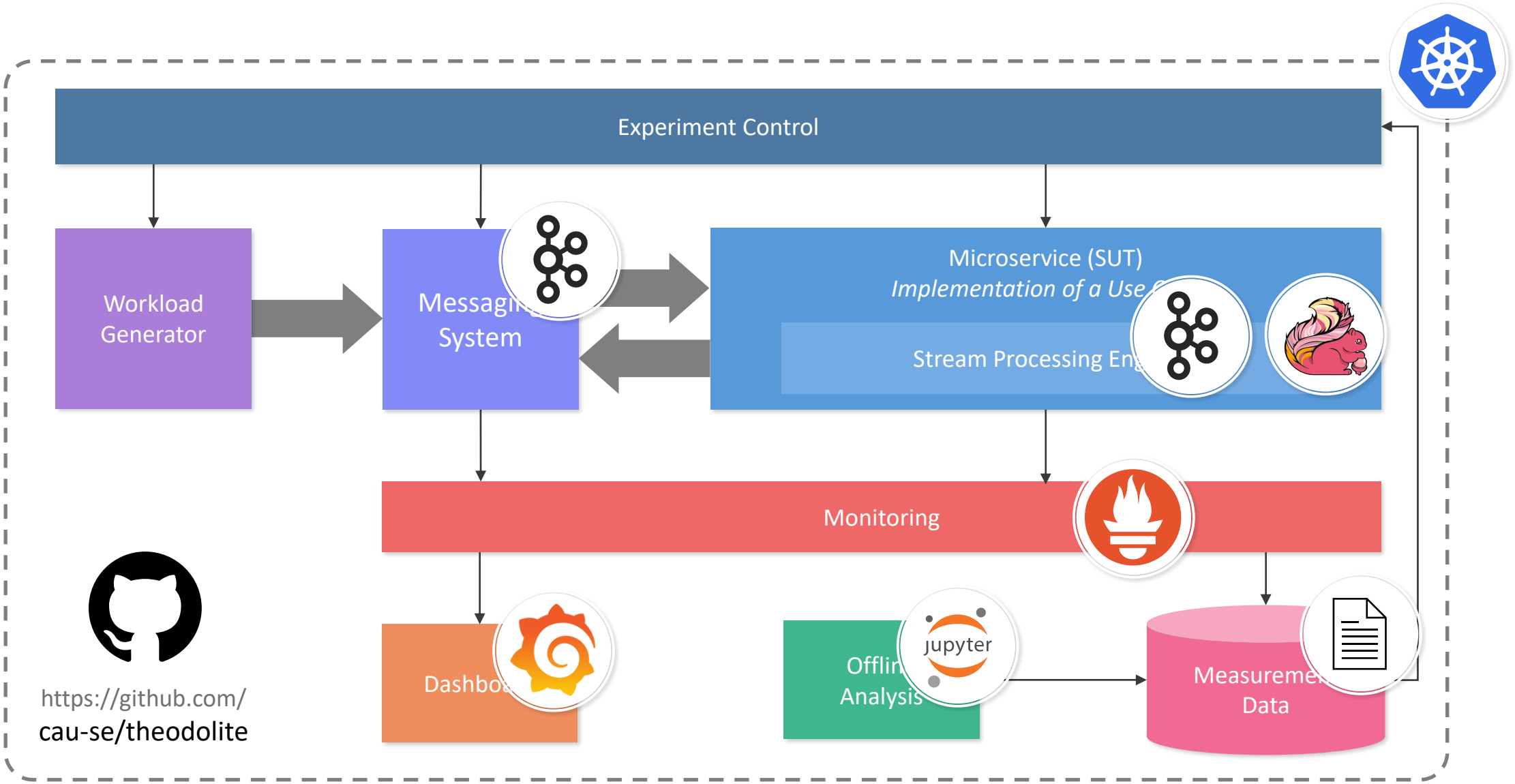
[Sanders 2015] R. Sanders, G. Brataas, M. Cecowski, K. Haslum, S. Ivanšek, J. Polutnik, and B. Viken. 2015. CloudStore – Towards Scalability Benchmarking in Cloud Computing. Procedia Comput. Sci. 68 (2015).

References

[Van Dongen 2020] G. Van Dongen and D.E. Van den Poel. 2021. Evaluation of stream processing frameworks. IEEE Trans. Parallel Distrib. Syst. 31 (2020).

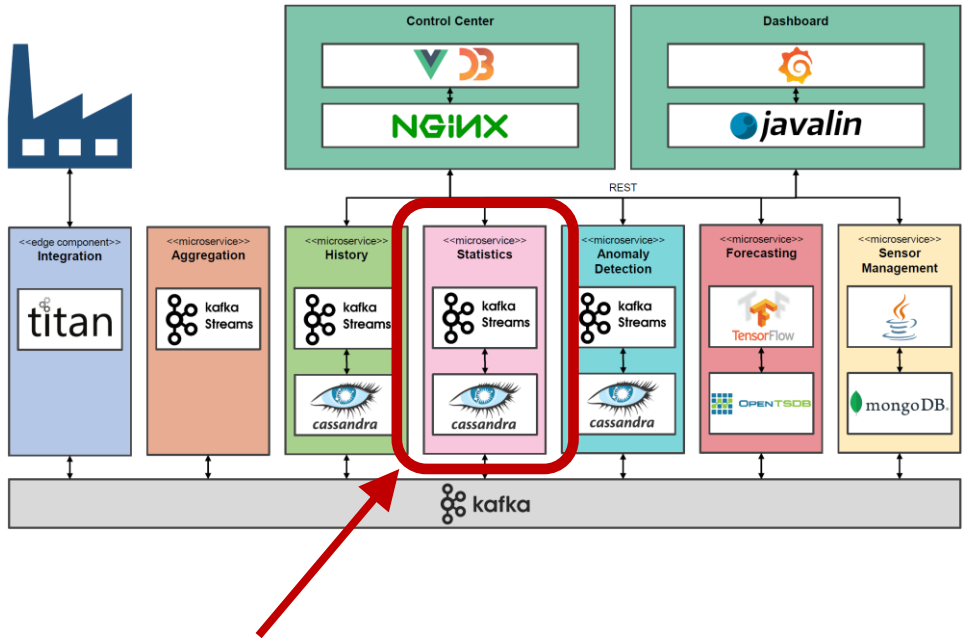
[Weber 2014] A. Weber, N. Herbst, H. Groenda, and S. Kounev. 2014. Towards a Resource Elasticity Benchmark for Cloud Environments. In Proc. InternationalWorkshop on Hot Topics in Cloud Service Scalability.

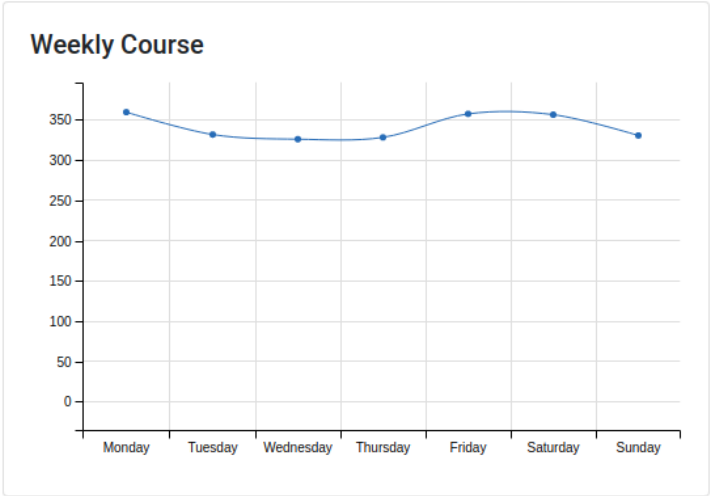
Theodolite's Framework Architecture

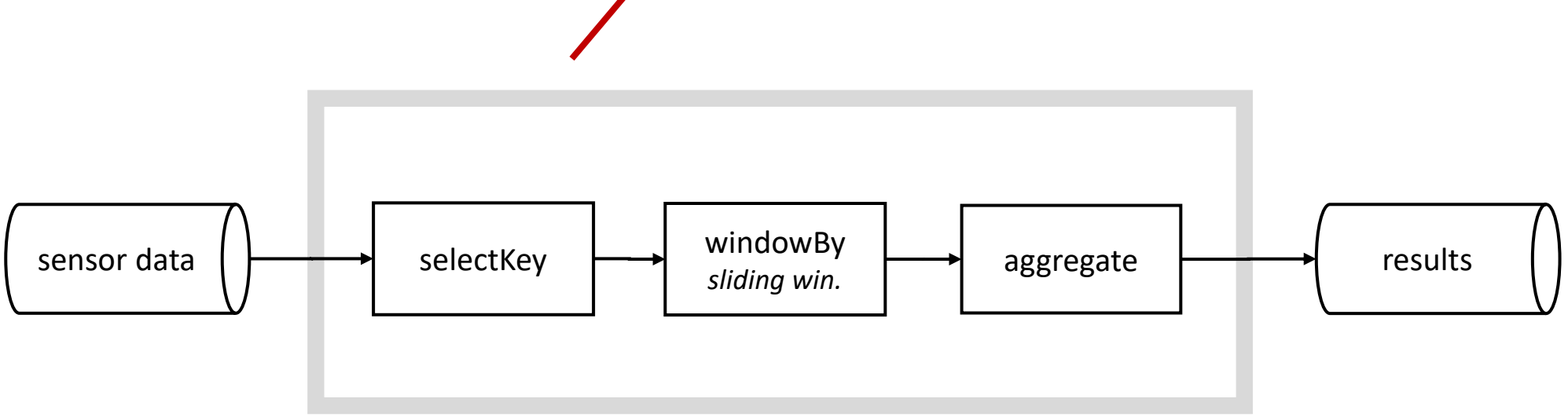


Theodolite's Benchmarks

titan
Control Center

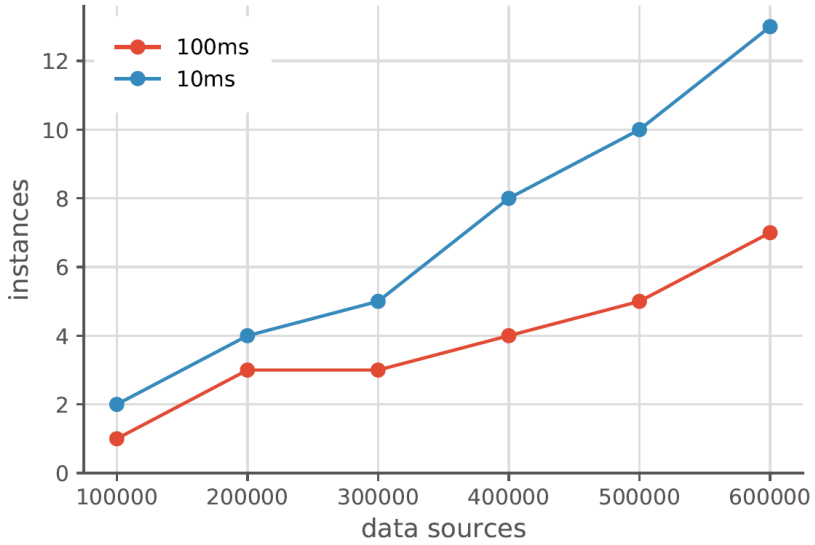


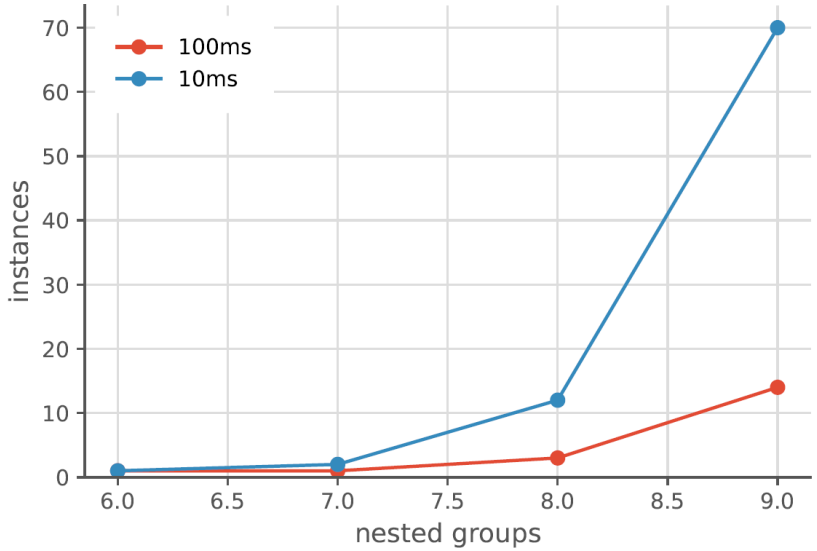




Example: Kafka Streams Commit Interval

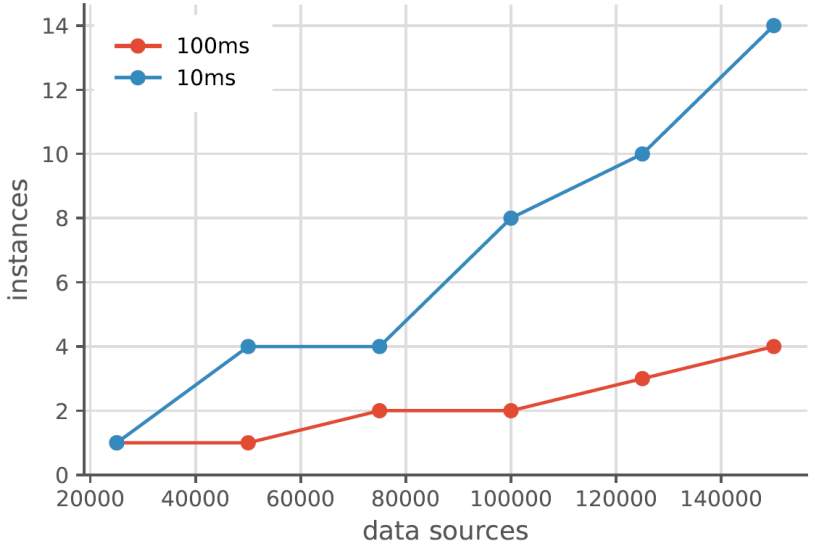
Database Storage

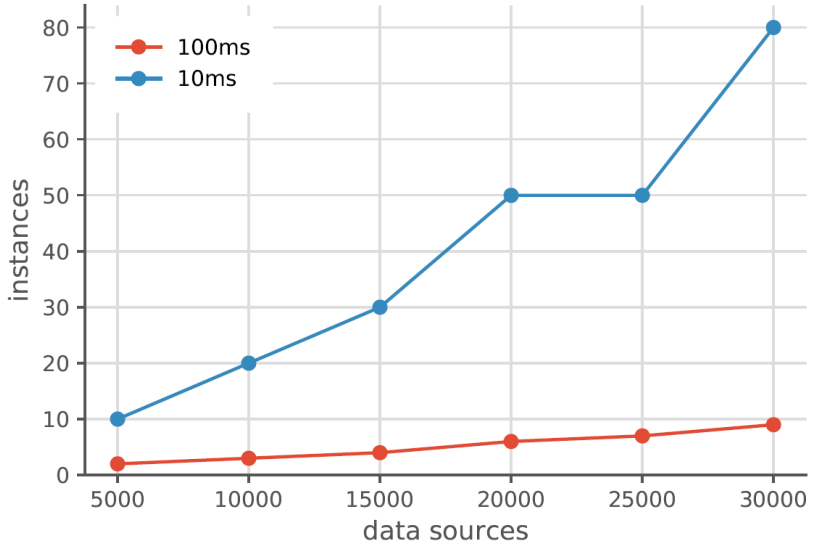




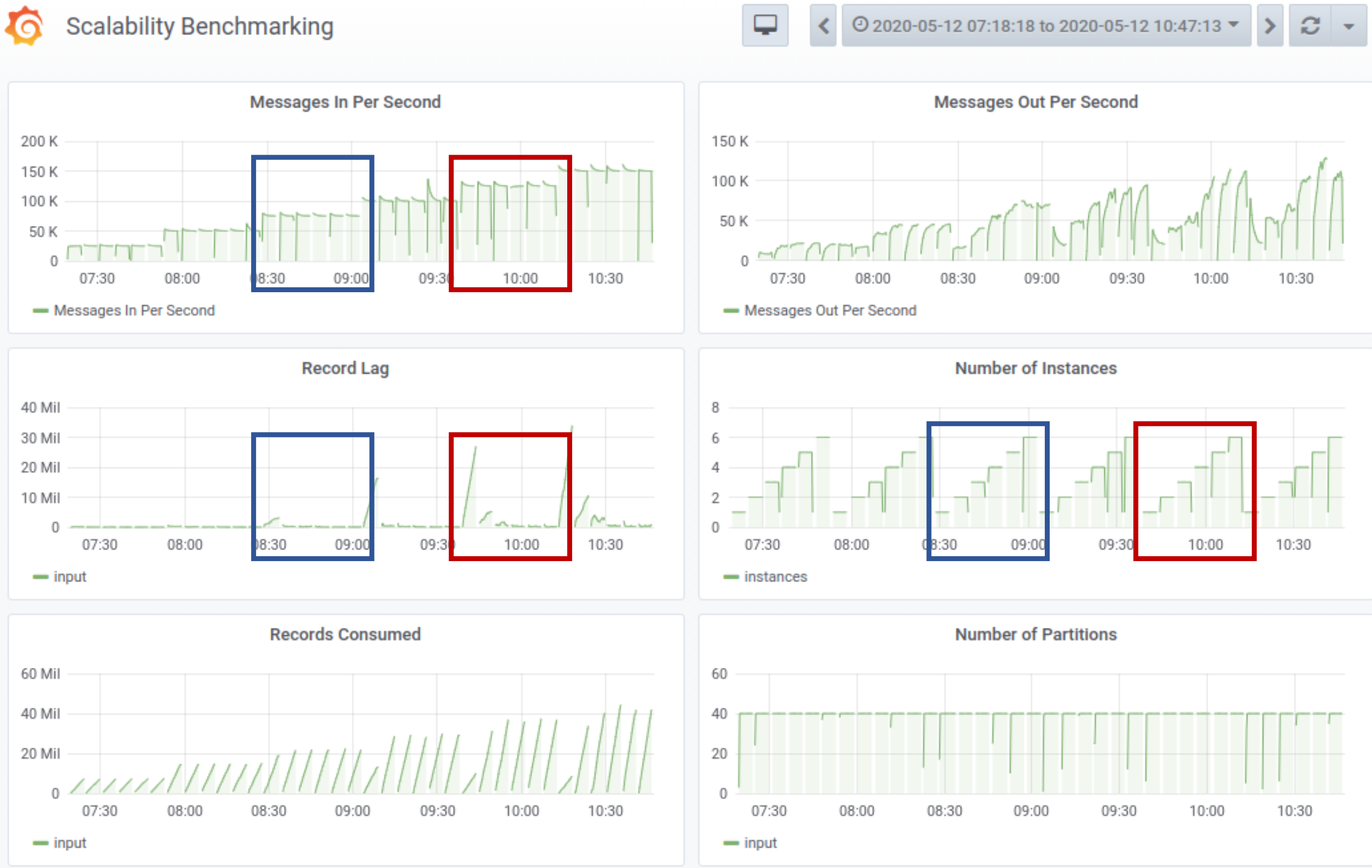
Hierarchical Aggregation

Downsampling

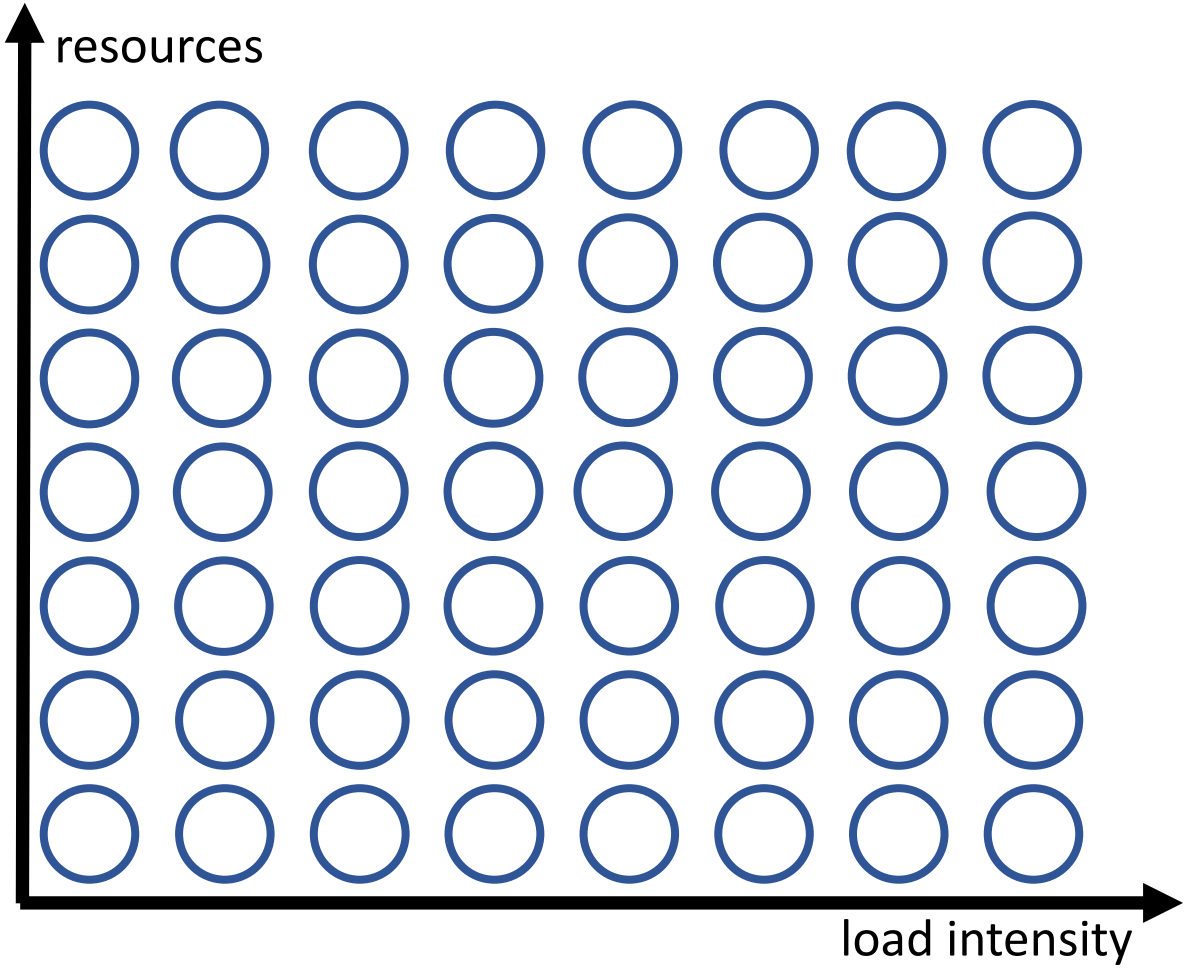




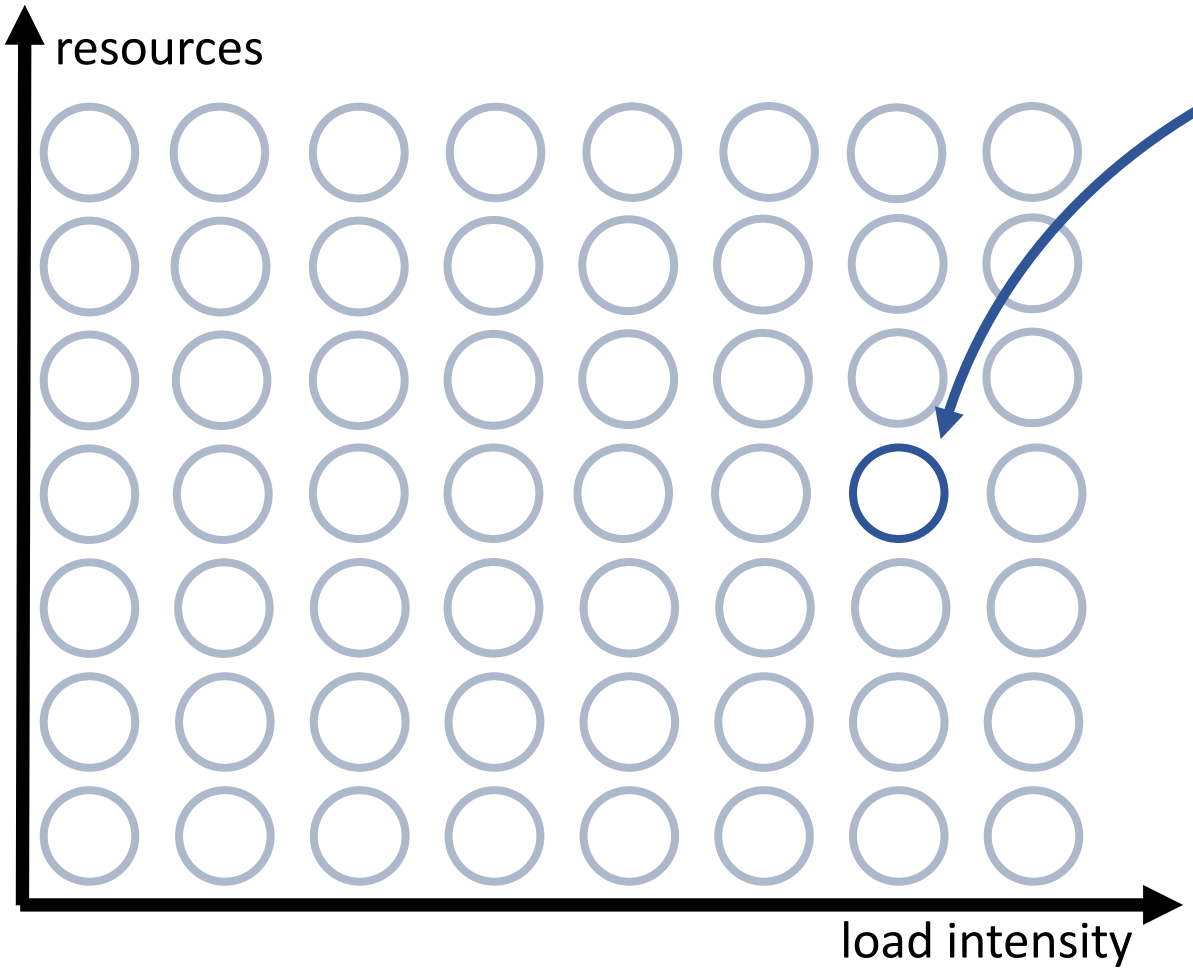
Aggregating Time Attributes


Scalability Benchmarking
2020-05-12 07:18:18 to 2020-05-12 10:47:13
Messages In Per Second
200 K
150 K
100 K
50 K
0
07:30
08:00
08:30
09:00
09:30
10:00
10:30
Messages In Per Second
Messages Out Per Second
150 K
100 K
50 K
0
07:30
08:00
08:30
09:00
09:30
10:00
10:30
Messages Out Per Second
Record Lag
40 Mil
30 Mil
20 Mil
10 Mil
0
07:30
08:00
08:30
09:00
09:30
10:00
10:30
input
Number of Instances
8
6
4
2
0
07:30
08:00
08:30
09:00
09:30
10:00
10:30
instances
Records Consumed
60 Mil
40 Mil
20 Mil
0
07:30
08:00
08:30
09:00
09:30
10:00
10:30
input
Number of Partitions
60
40
20
0
07:30
08:00
08:30
09:00
09:30
10:00
10:30
input

Theodolite’s Scalability Measurement Method



Theodolite's Scalability Measurement Method

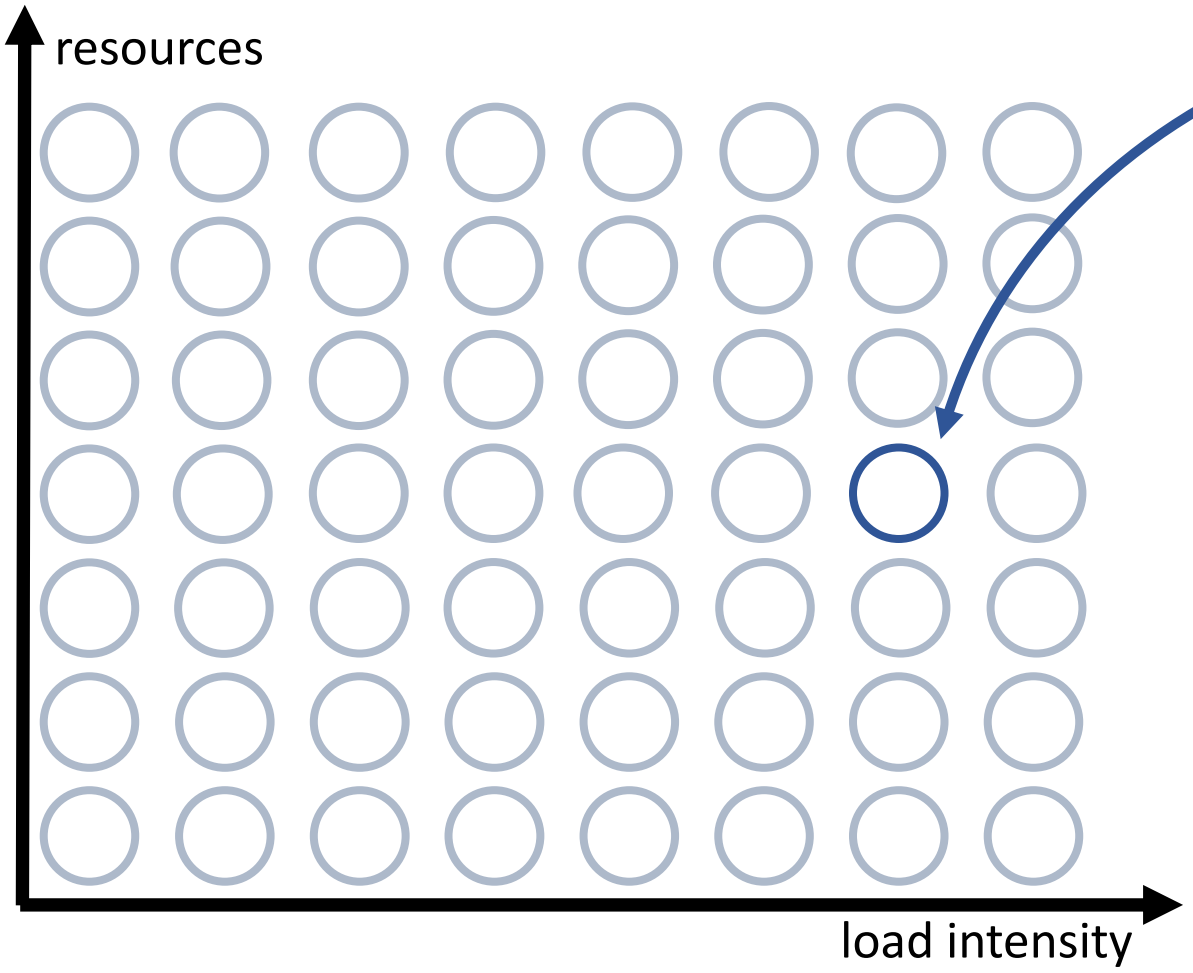


sufficient resources for load?

lag increase over time?

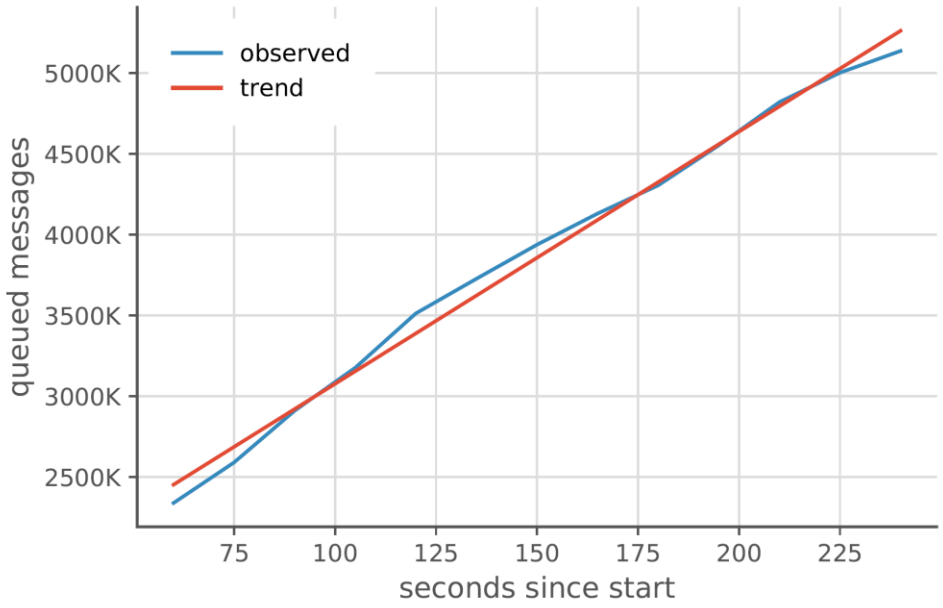
lag = queued messages

Theodolite's Scalability Measurement Method

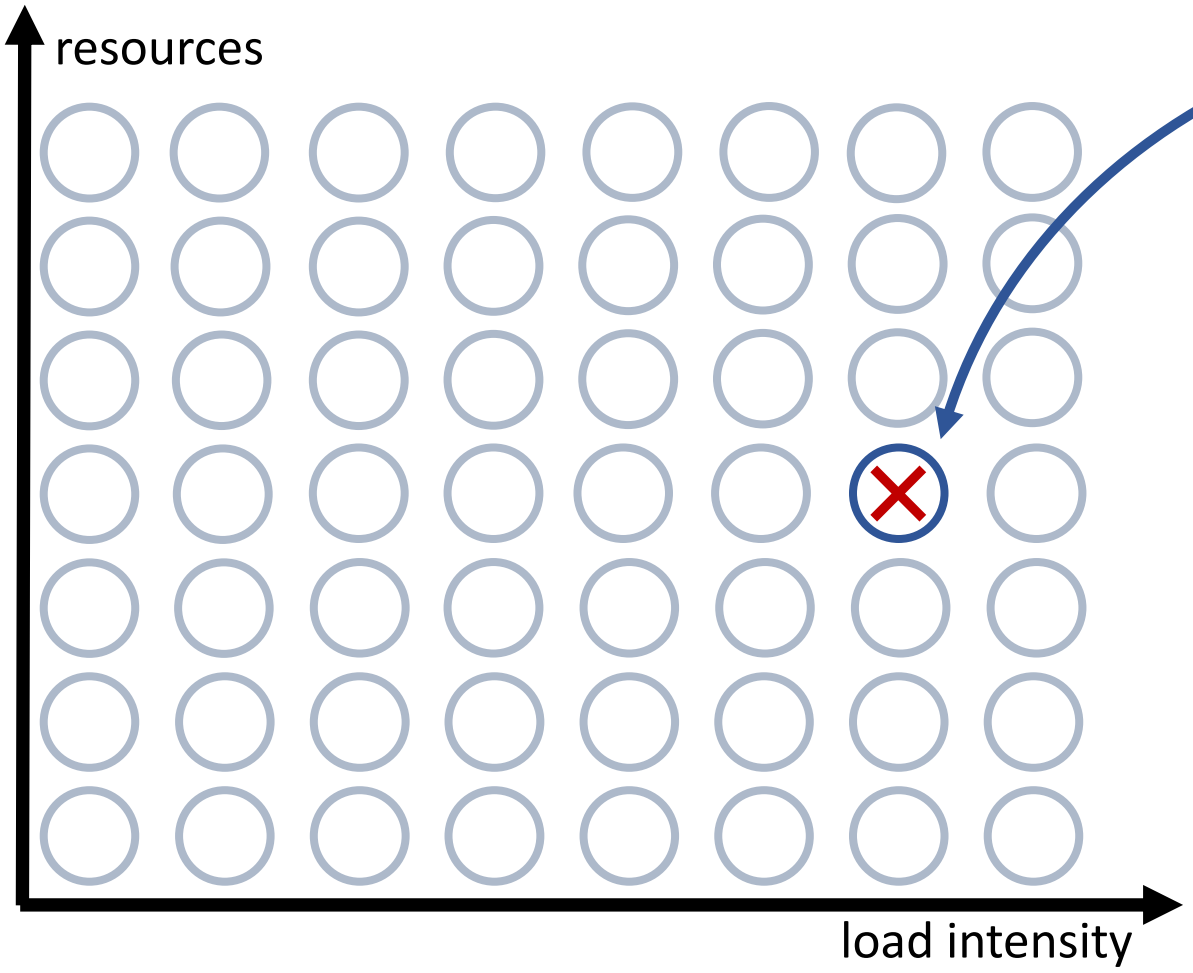


sufficient resources for load?

lag increase over time?

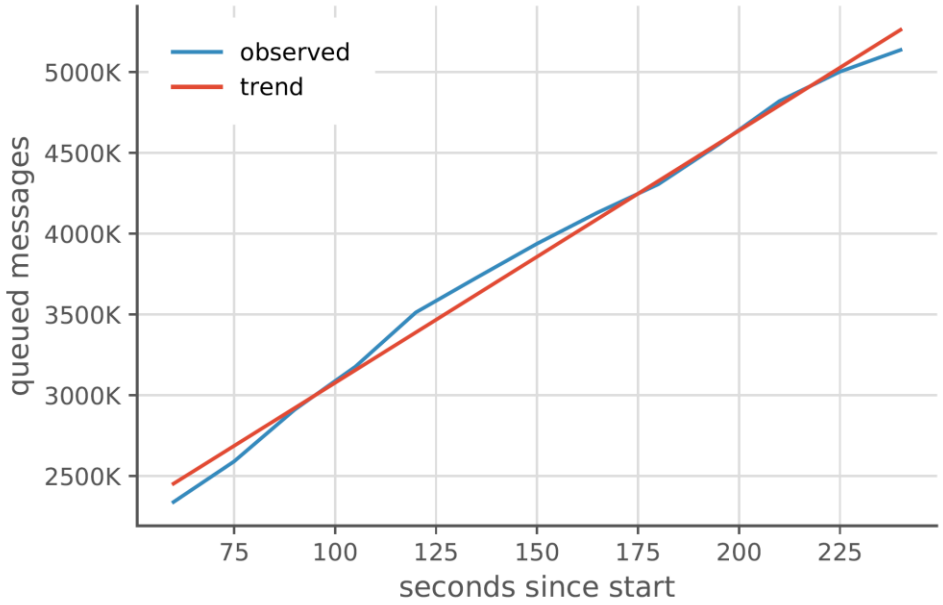


Theodolite's Scalability Measurement Method

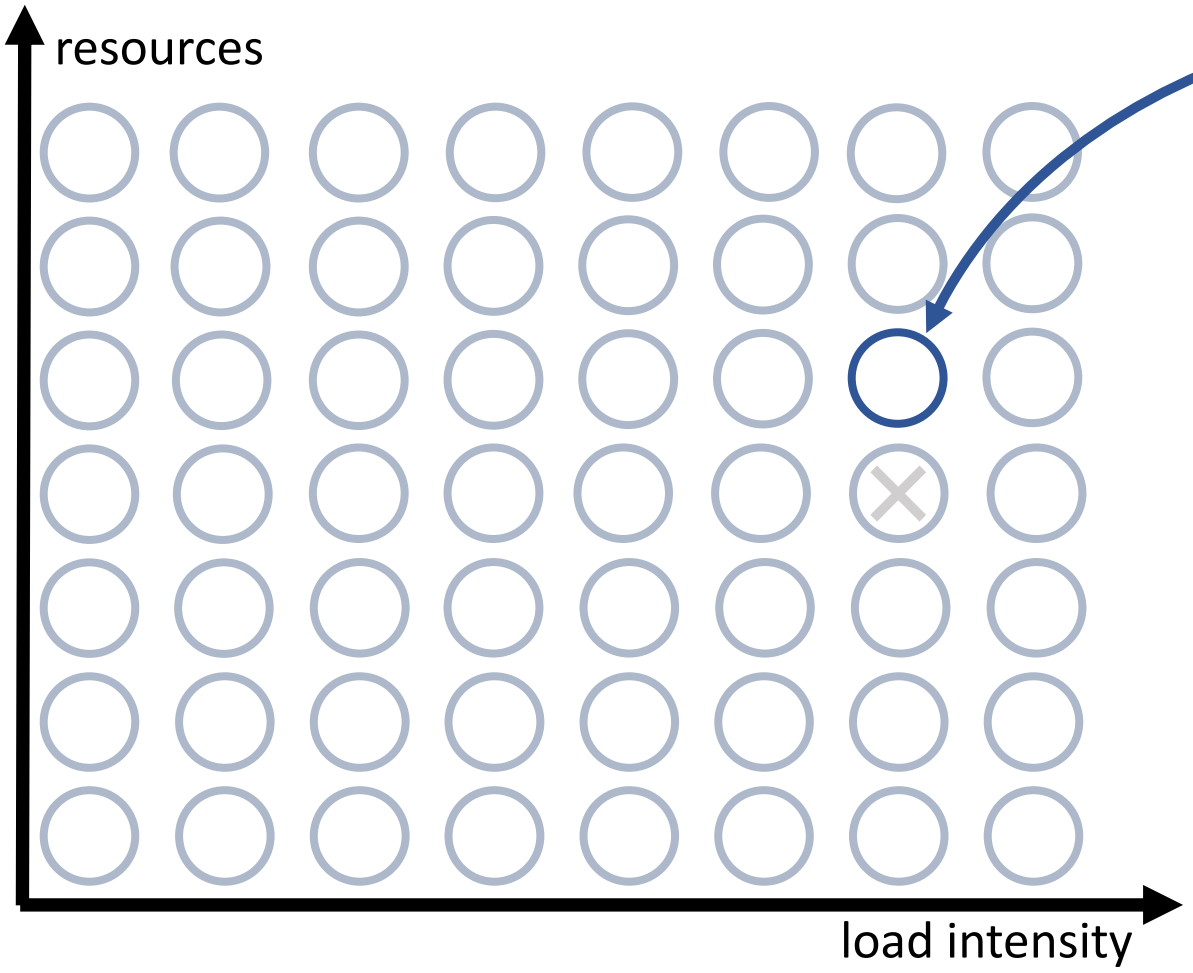


sufficient resources for load? No!

lag increase over time? Yes!

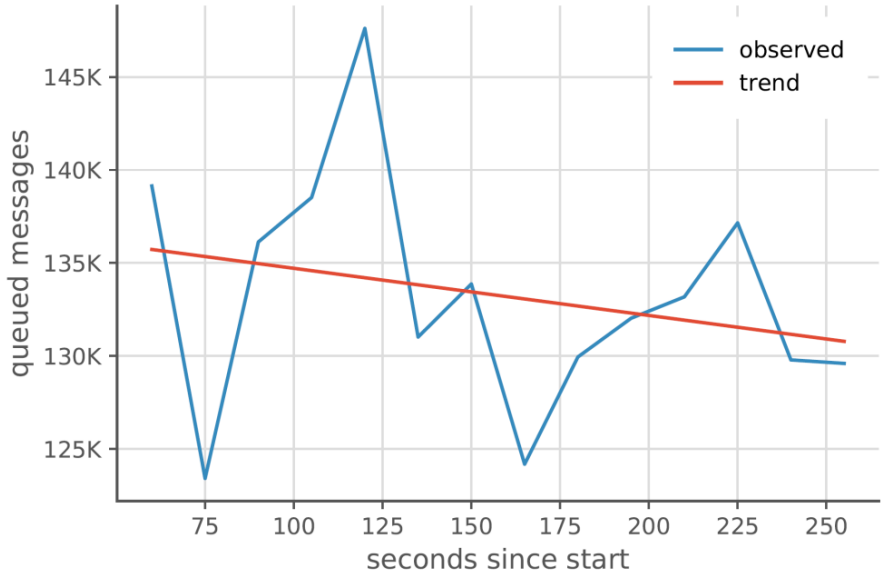


Theodolite's Scalability Measurement Method

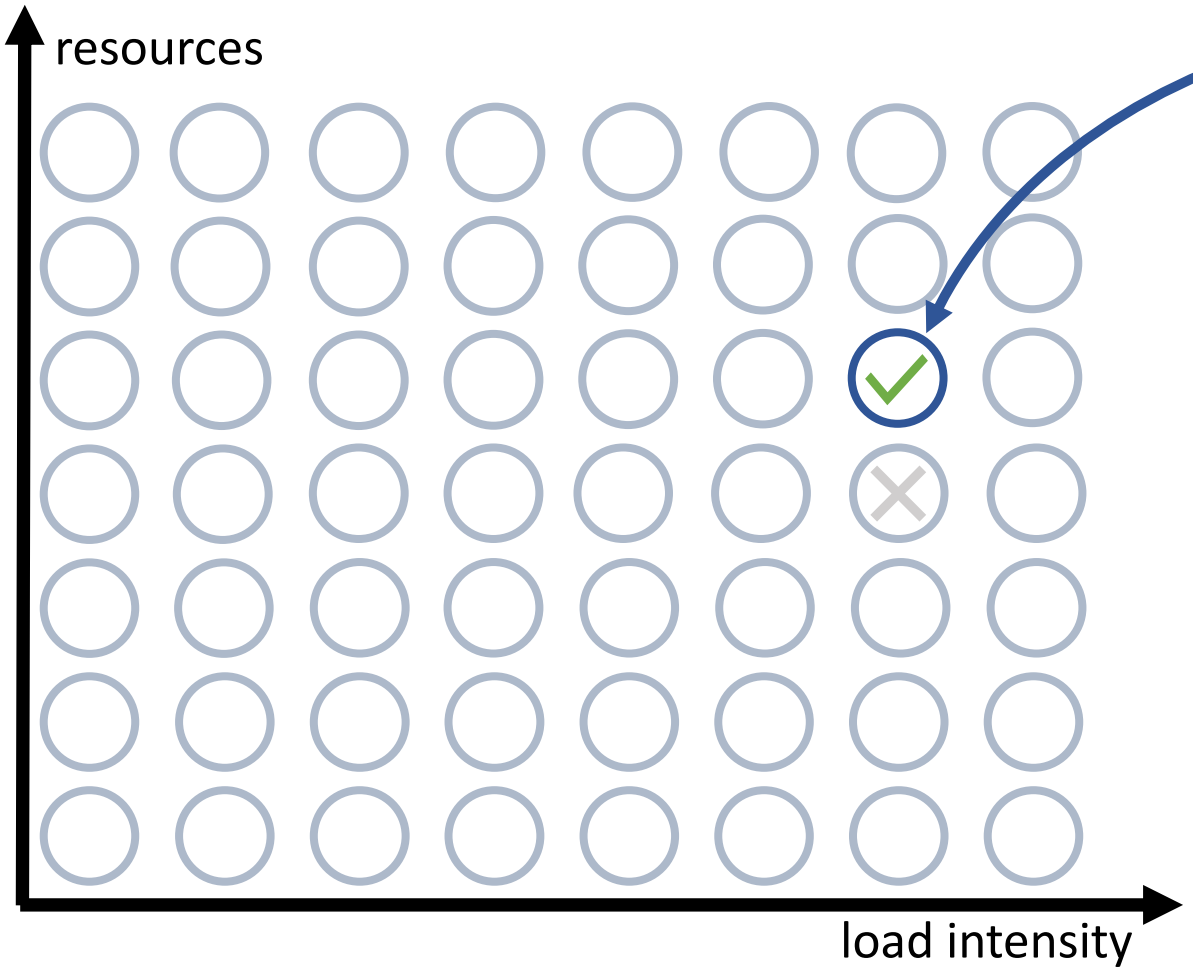


sufficient resources for load?

lag increase over time?

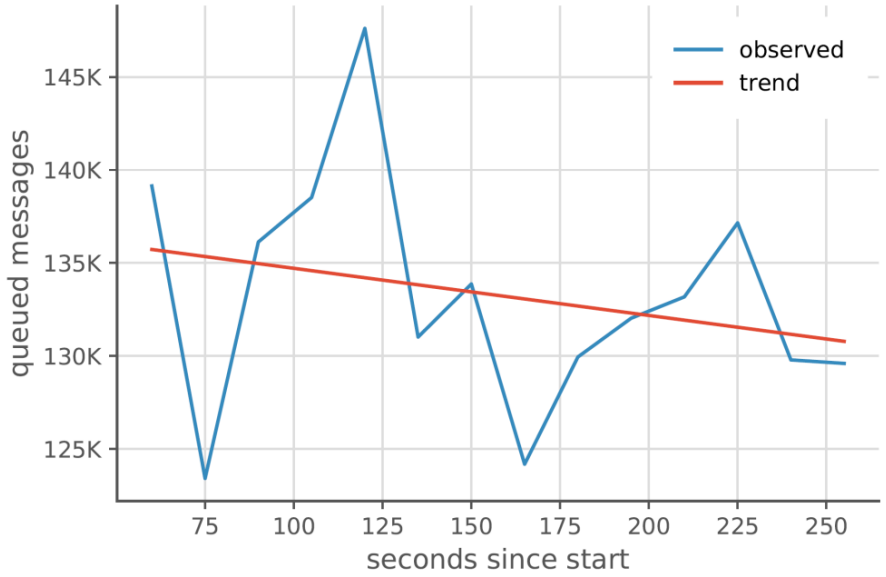


Theodolite's Scalability Measurement Method

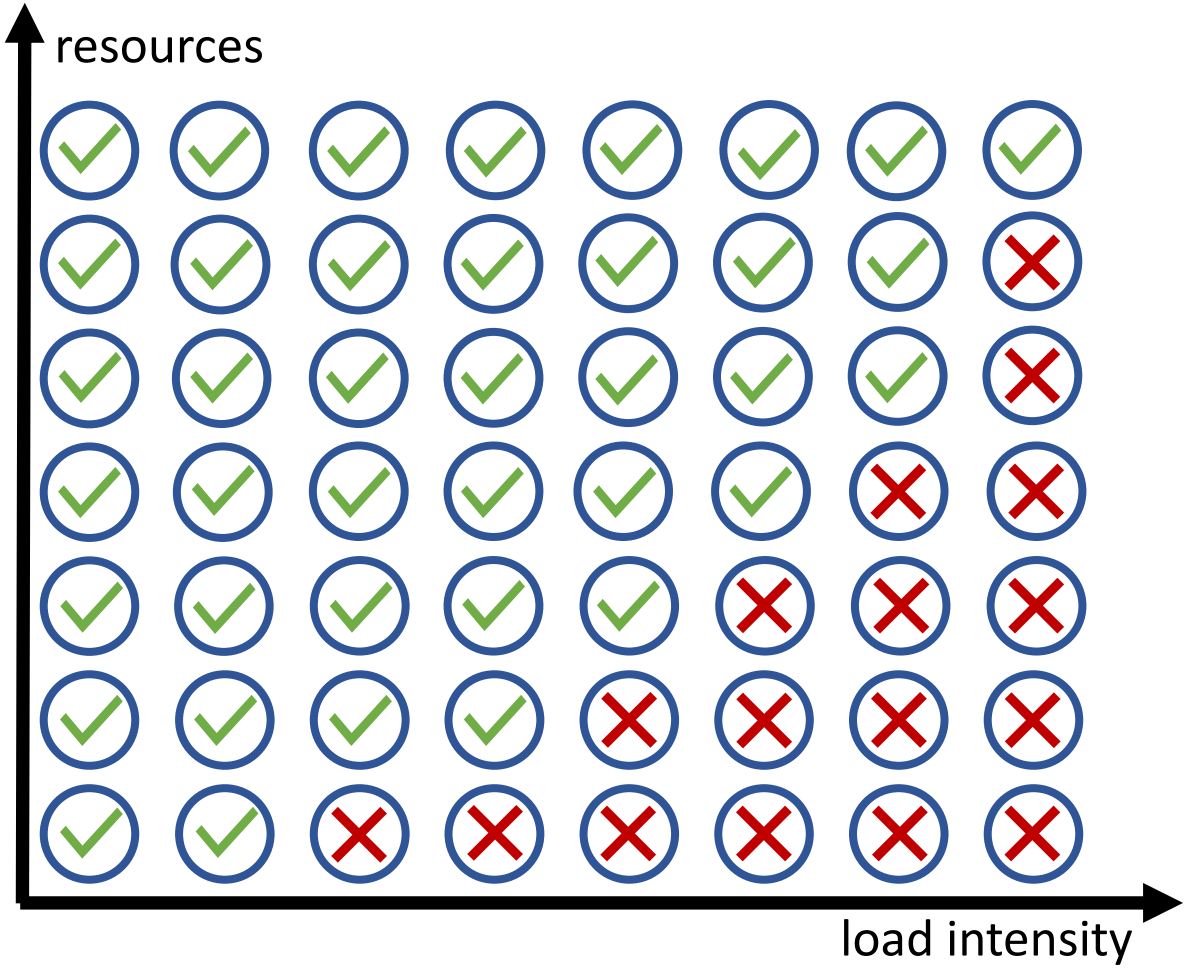


sufficient resources for load? Yes!

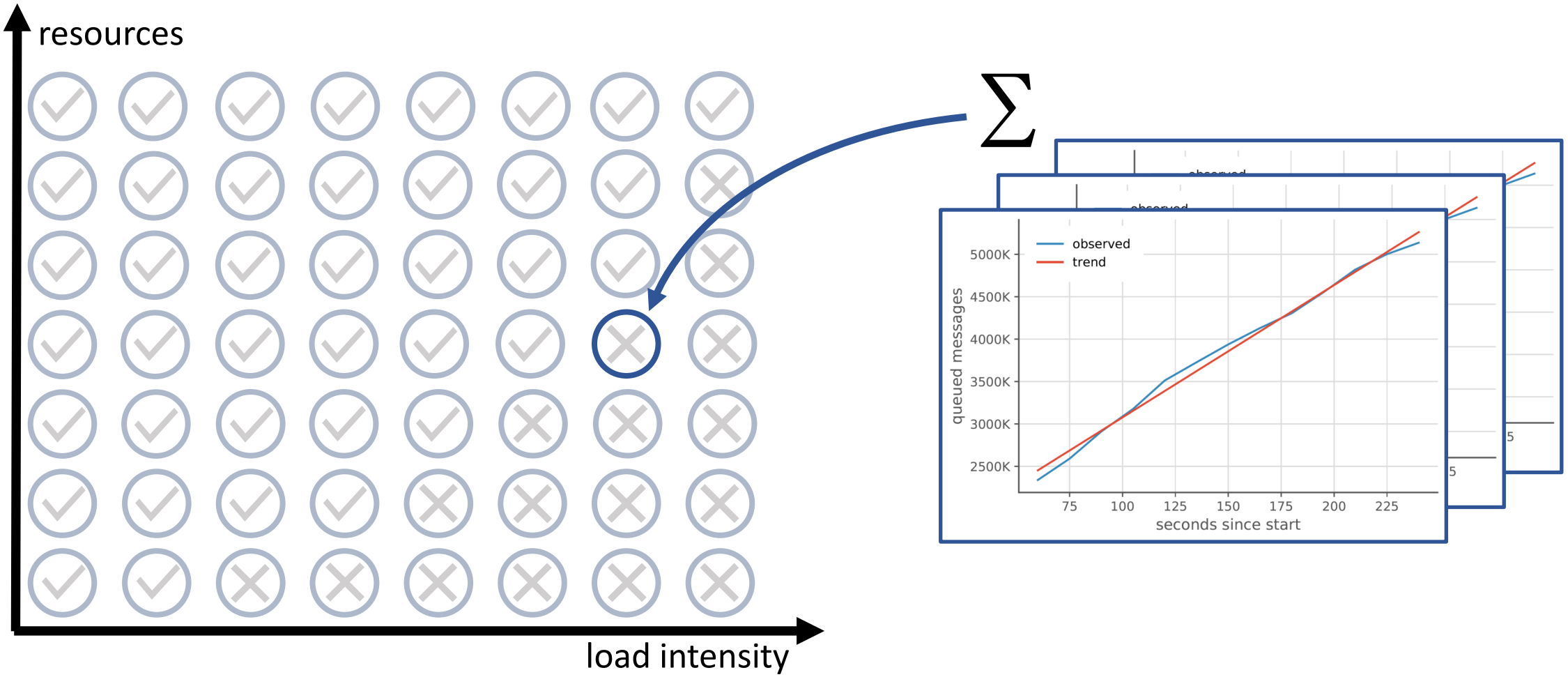
lag increase over time? No!



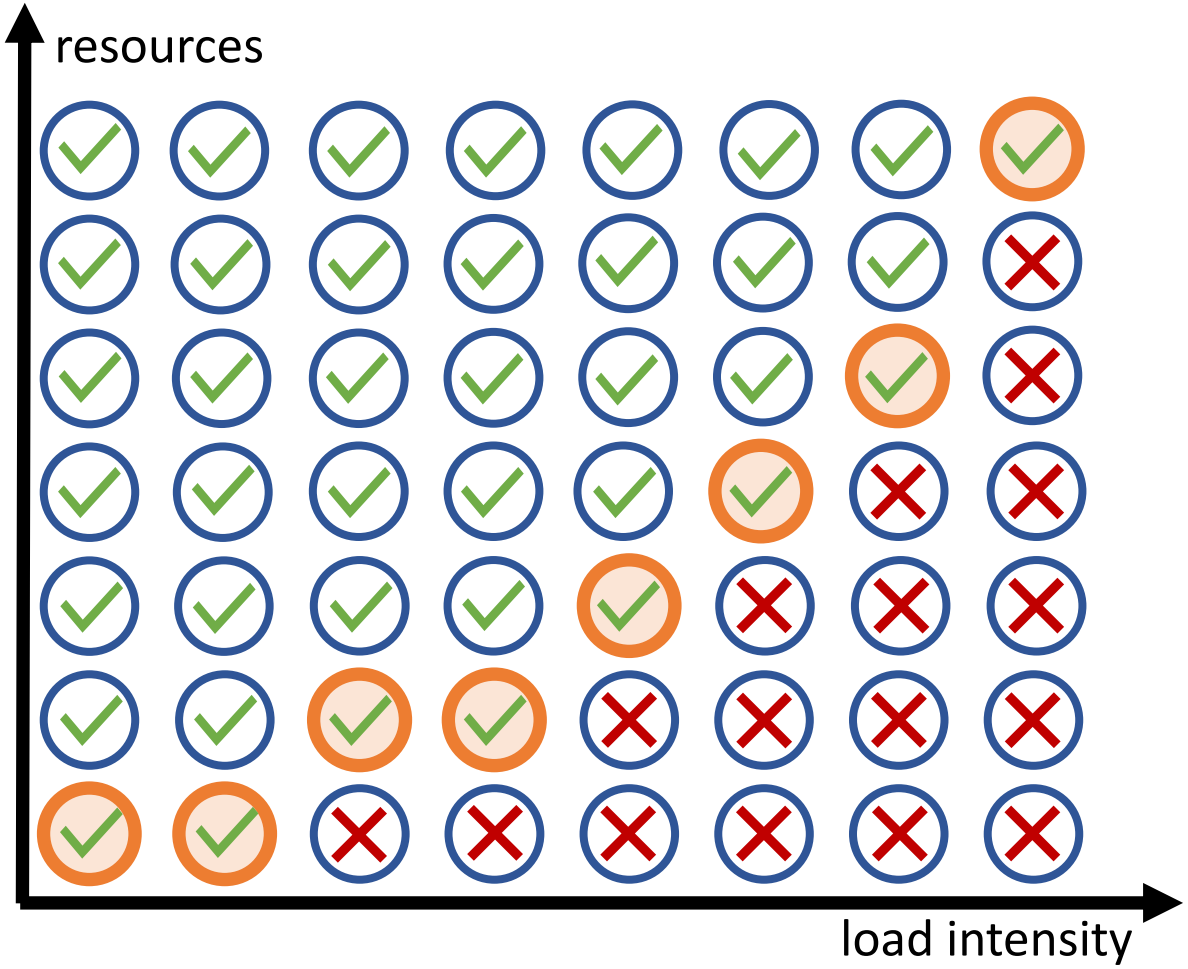
Theodolite’s Scalability Measurement Method



Theodolite's Scalability Measurement Method



Theodolite's Scalability Measurement Method



Identify minimal required resources per load intensity

Theodolite's Scalability Measurement Method