Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Descartes
research

Autoscaling Rethought: A Continuous, Decentralized Approach

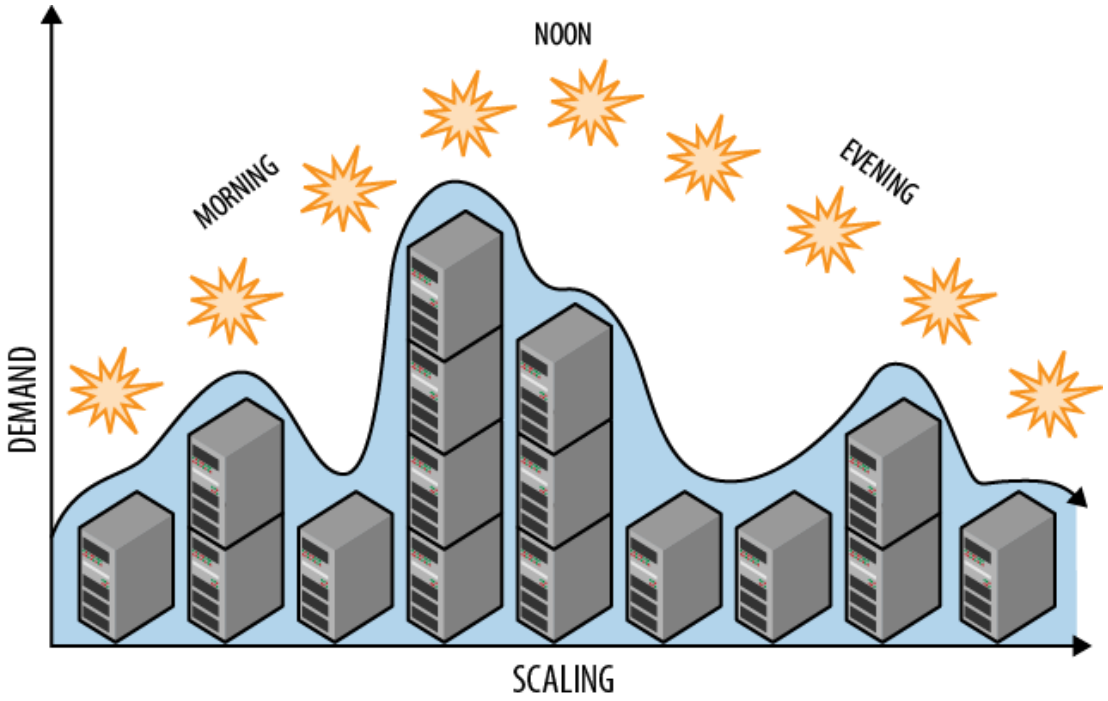
Martin Straesser

Intelligent Serverless and Cloud Applications Symposium

June 17-18, 2024, Zurich, Switzerland

Autoscaling

- Dynamically provision computing resources under varying load
- High-valued task in production systems
- Affects both operating costs and customer experience
- Across several sub-domains: traditional cloud computing, fog computing or serverless computing

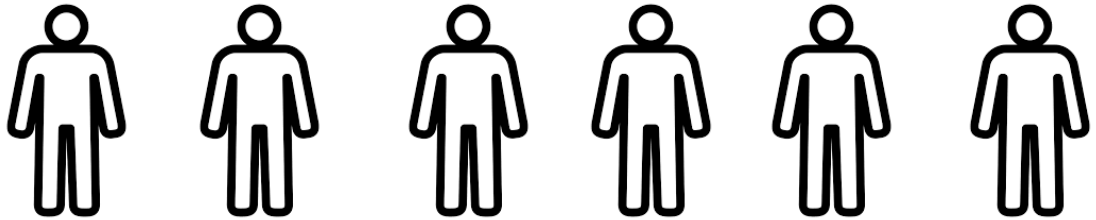


Continuous Decentralized Autoscaling - Context

- Vision paper: doi.org/10.1109/CCGridW59191.2023.00058
- Full paper under submission

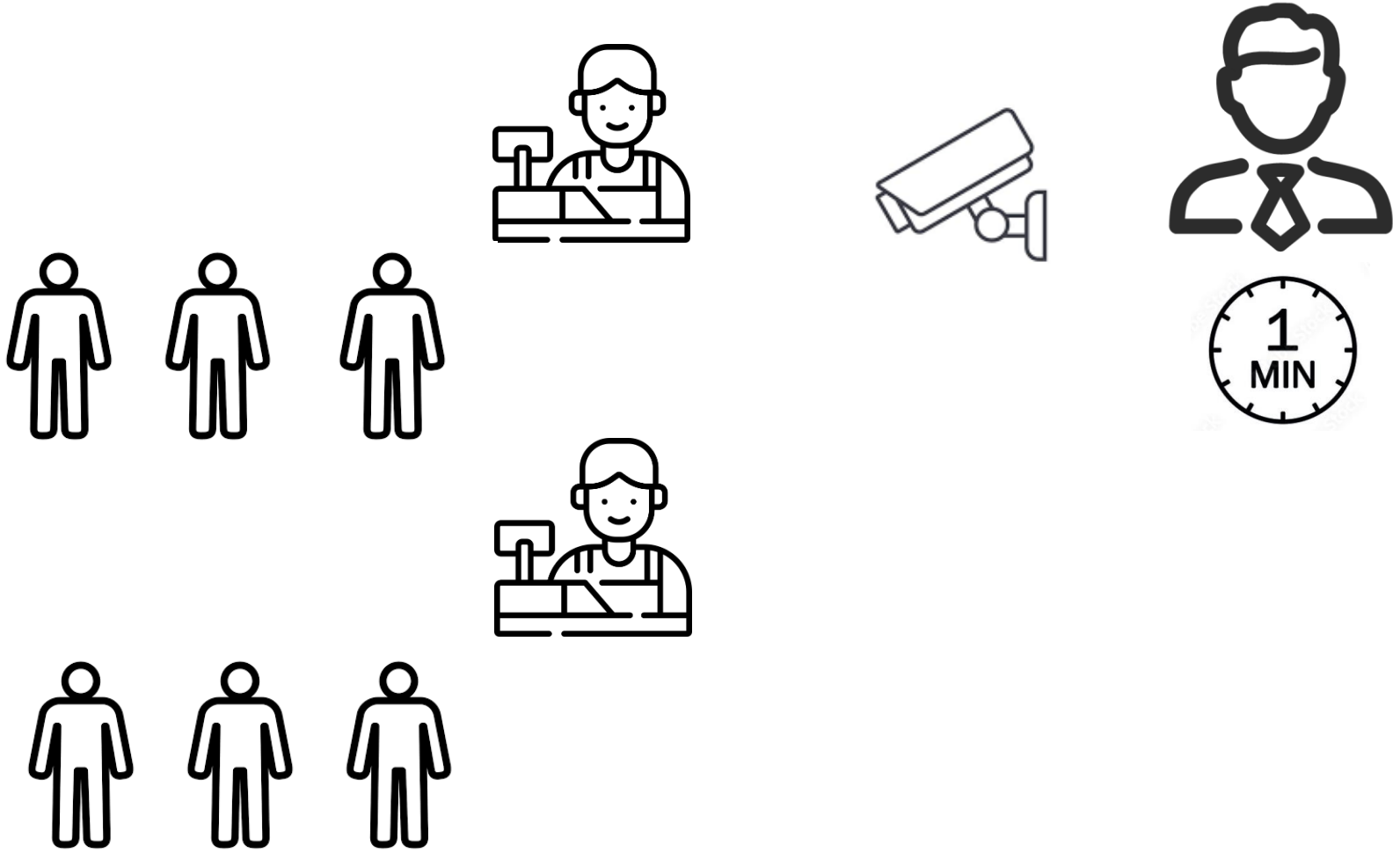
Continuous Decentralized Autoscaling – The Analogy

Conventional autoscaling



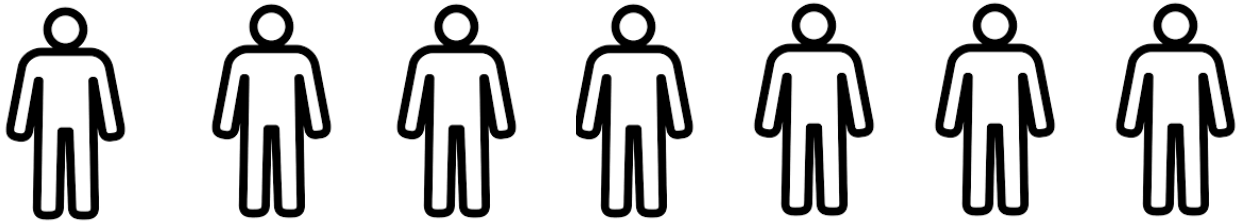
Continuous Decentralized Autoscaling – The Analogy

Conventional autoscaling



Continuous Decentralized Autoscaling – The Analogy

Conventional autoscaling

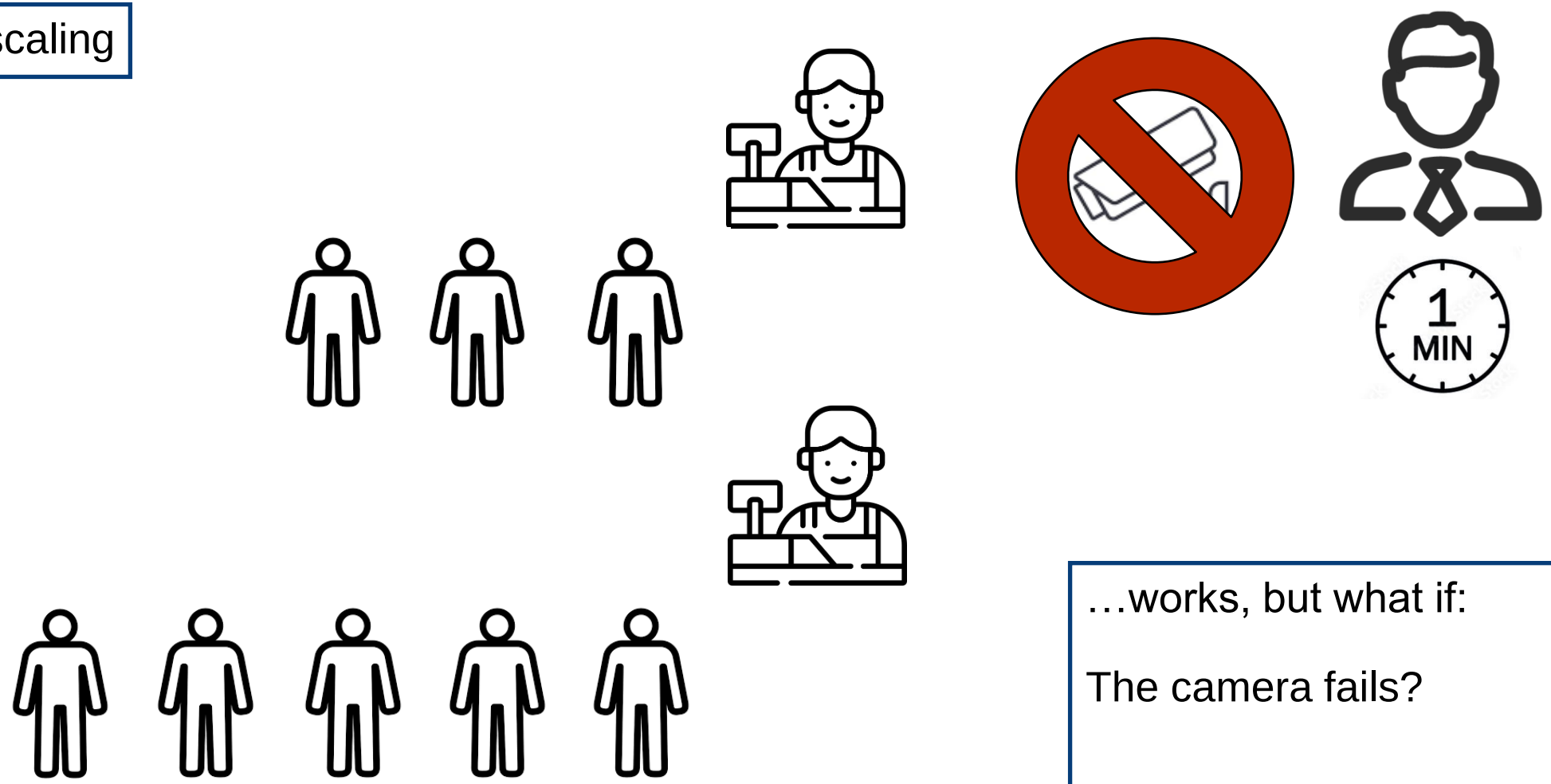


…works, but what if:

There is a sudden increase of customers at an unfortunate time?

Continuous Decentralized Autoscaling – The Analogy

Conventional autoscaling

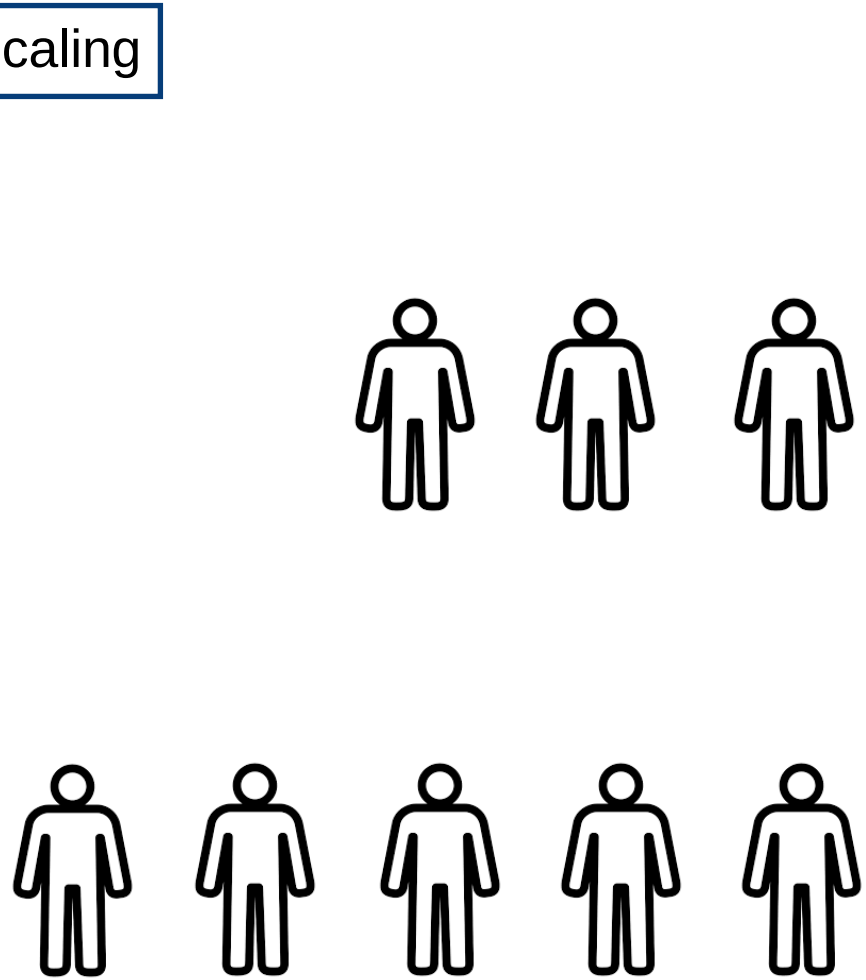


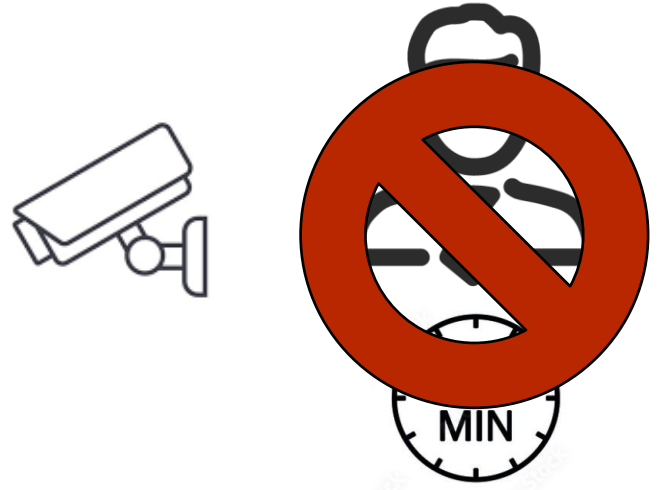
…works, but what if:

The camera fails?

Continuous Decentralized Autoscaling – The Analogy

Conventional autoscaling



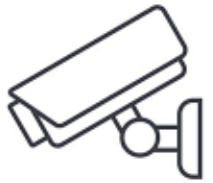


…works, but what if:

The manager fails?

Continuous Decentralized Autoscaling – The Analogy

Conventional autoscaling



Monitoring system

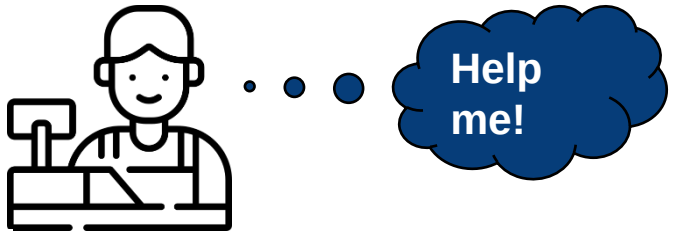
Autoscaler

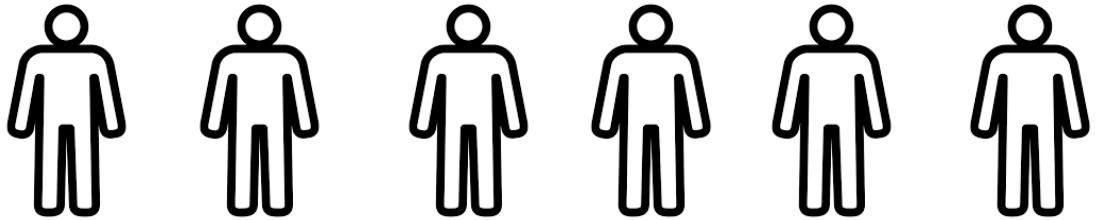
Arriving requests / Workload

Microservices / Functions

Continuous Decentralized Autoscaling – The Analogy

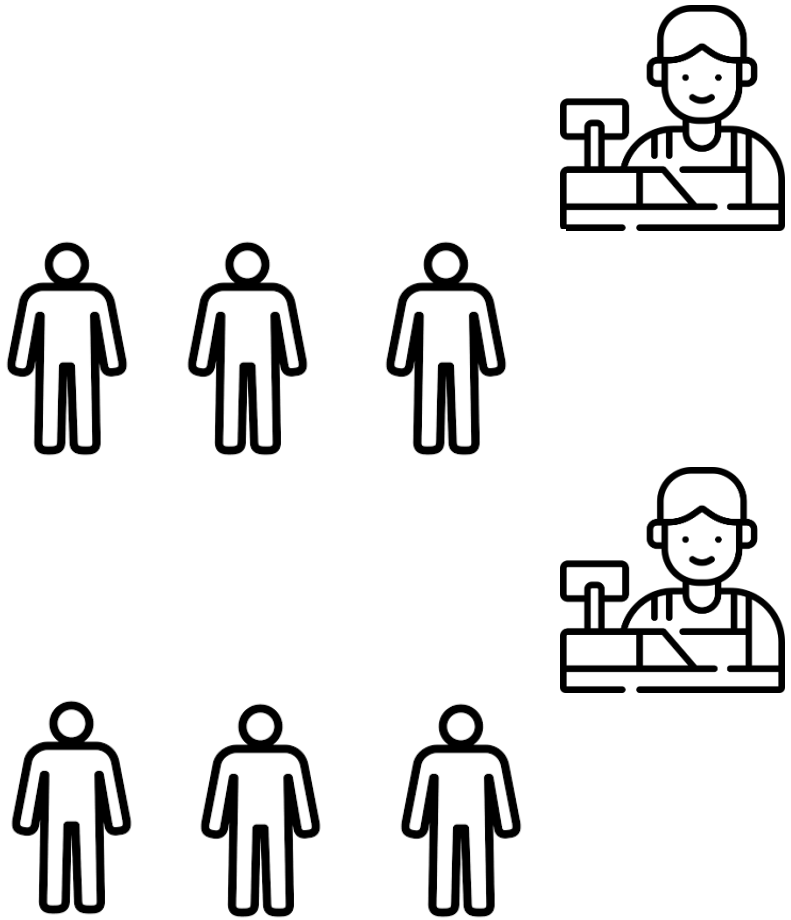
Decentralized autoscaling





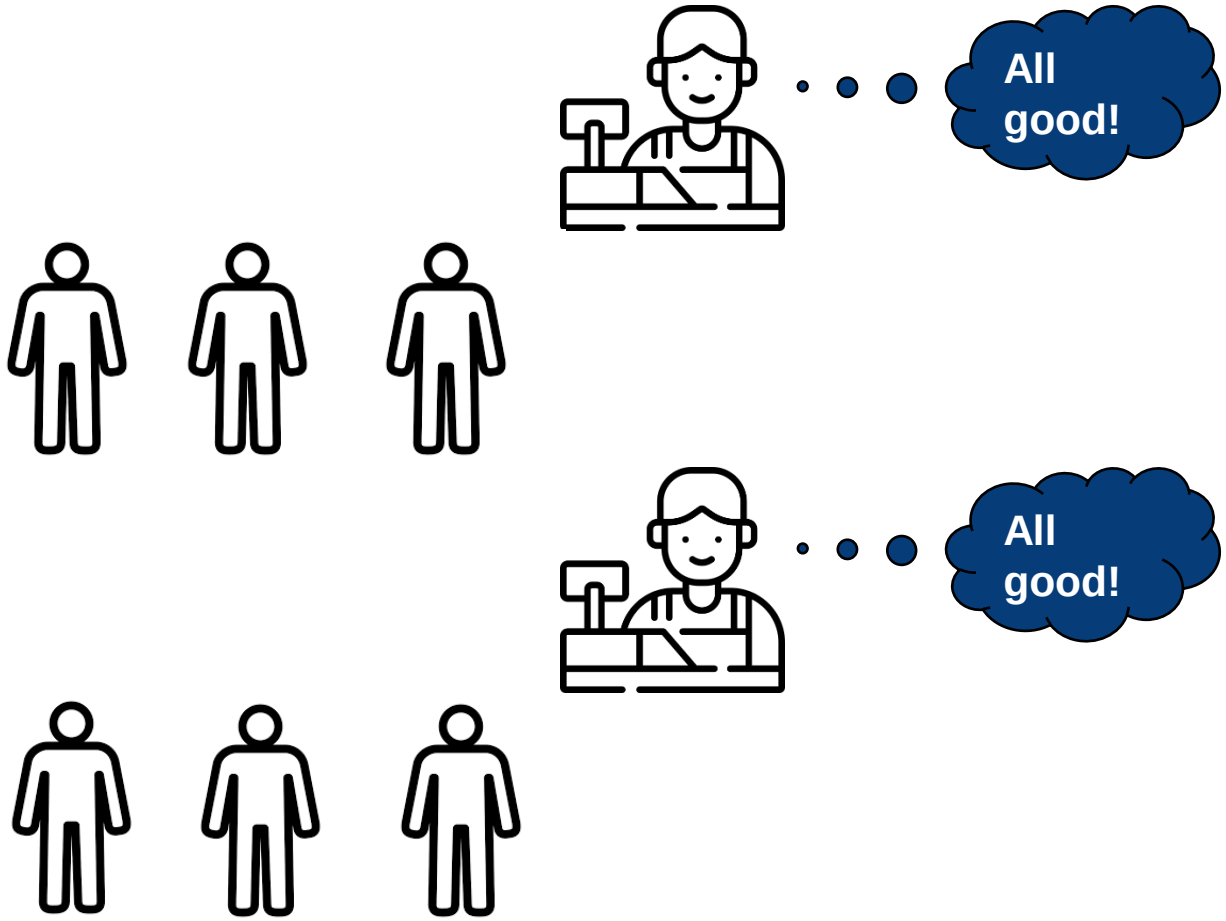
Continuous Decentralized Autoscaling – The Analogy

Decentralized autoscaling



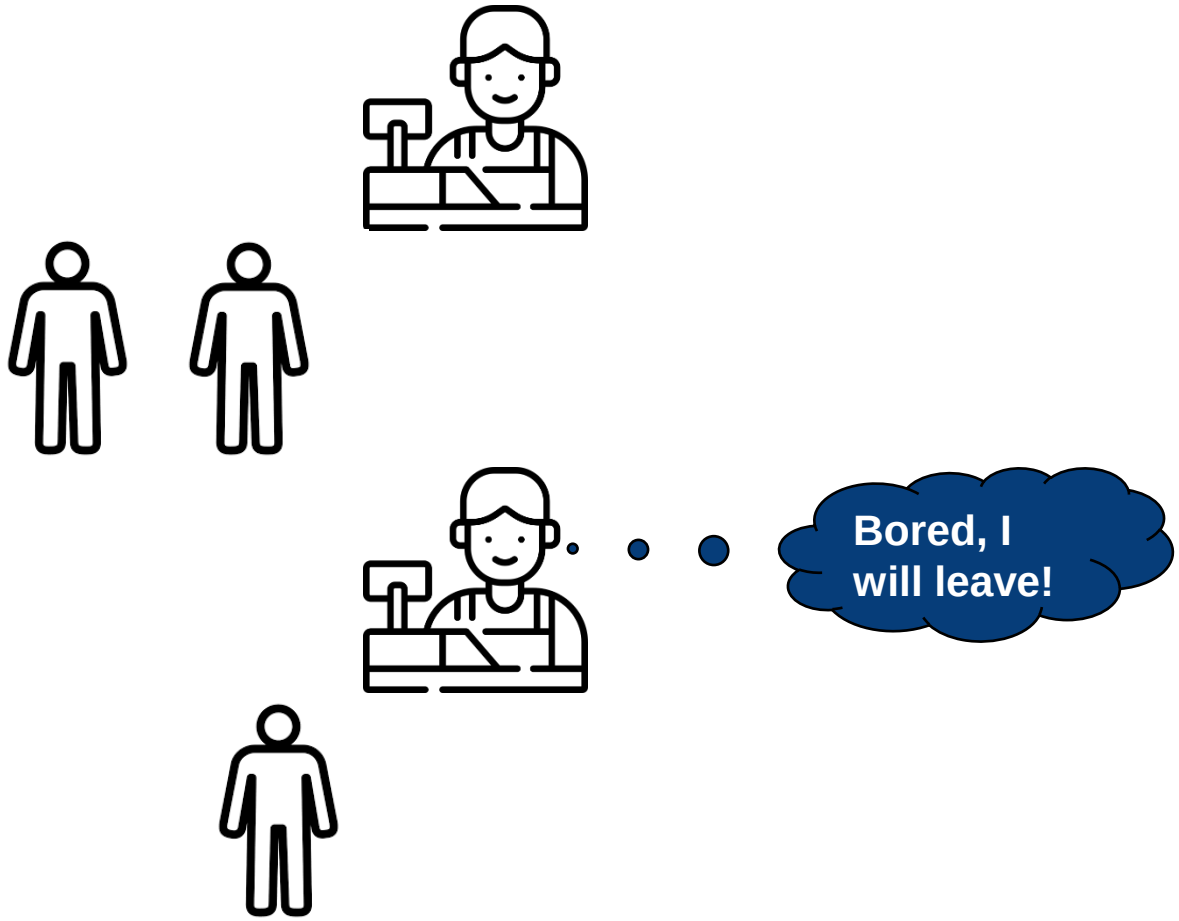
Continuous Decentralized Autoscaling – The Analogy

Decentralized autoscaling



Continuous Decentralized Autoscaling – The Analogy

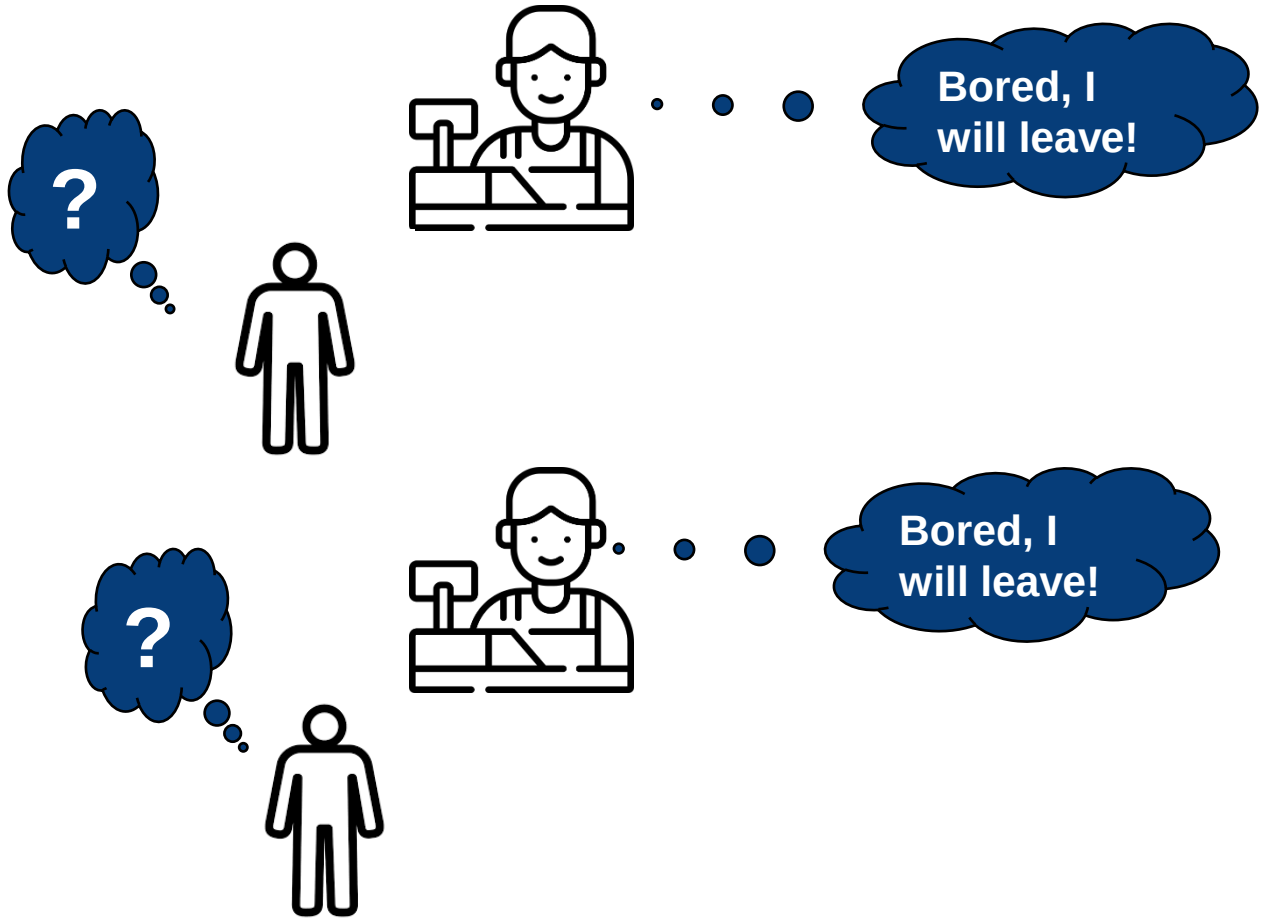
Decentralized autoscaling



Continuous Decentralized Autoscaling – The Analogy

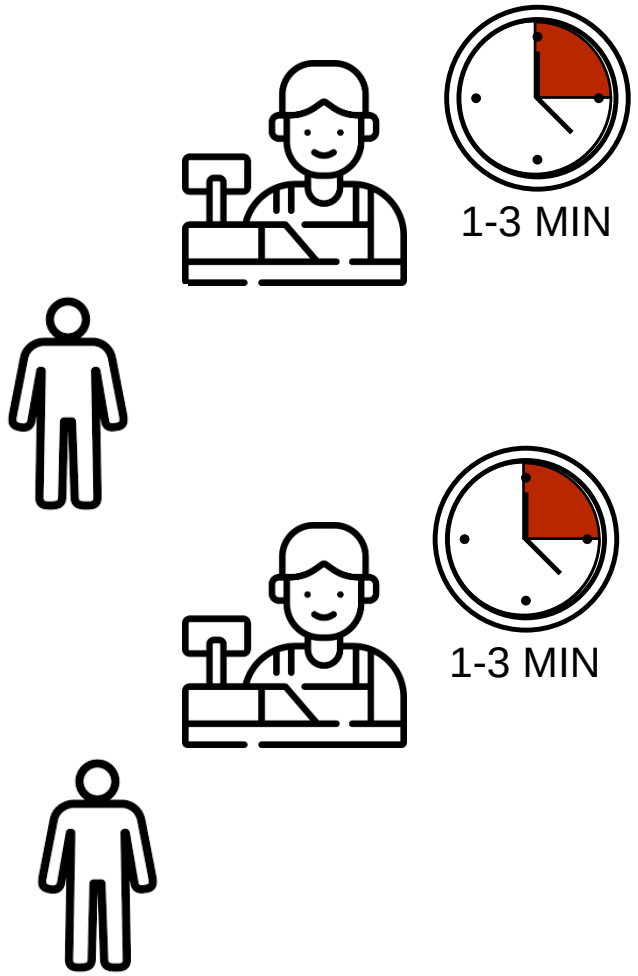
Decentralized autoscaling

Problematic situation



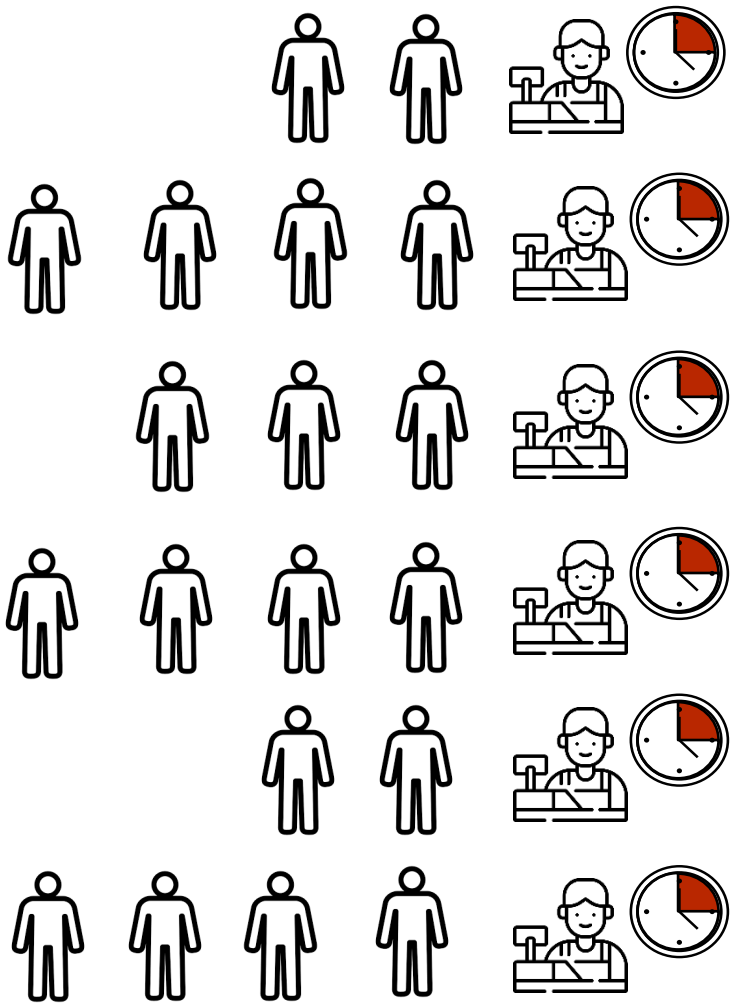
Continuous Decentralized Autoscaling – The Analogy

Continuous Decentralized Autoscaling



Continuous Decentralized Autoscaling – The Analogy

Continuous Decentralized Autoscaling



For a “large” number of instances we converge into a continuous process

For the example here with 6 cashiers taking decisions every 1-3 mins, the average time to the next decision is 21 seconds when we look at the system at an arbitrary time

Continuous Decentralized Autoscaling – Key Characteristics

Conventional Autoscalers	Continuous Decentralized Autoscaling
One autoscaler manages a group of service instances.	Each instance gets the right to trigger autoscaling.
Requires centralized monitoring system and data gathering.	Each instance processes own monitoring data, no centralized monitoring needed.
Monitoring and data processing overhead increases with system size.	This overhead is constant.
Scaling policies might be fitted to a specific system size.	As instances are not aware of total system size, scaling policies are always scale-invariant.
Autoscalers must choose scaling action from a large action space.	Instances just decide between UP, DOWN and HOLD.
Reaction times to unforeseen events are limited by design.	Reaction times to unforeseen events depend on system size.
Complex algorithms/learning are used to take correct decisions. Wrong decisions are costly.	Wrong scaling decisions are not costly because of the increased scaling frequency.

Continuous Decentralized Autoscaling – Algorithm & Config

Algorithm 1 Algorithm for Decentralized Autoscaling

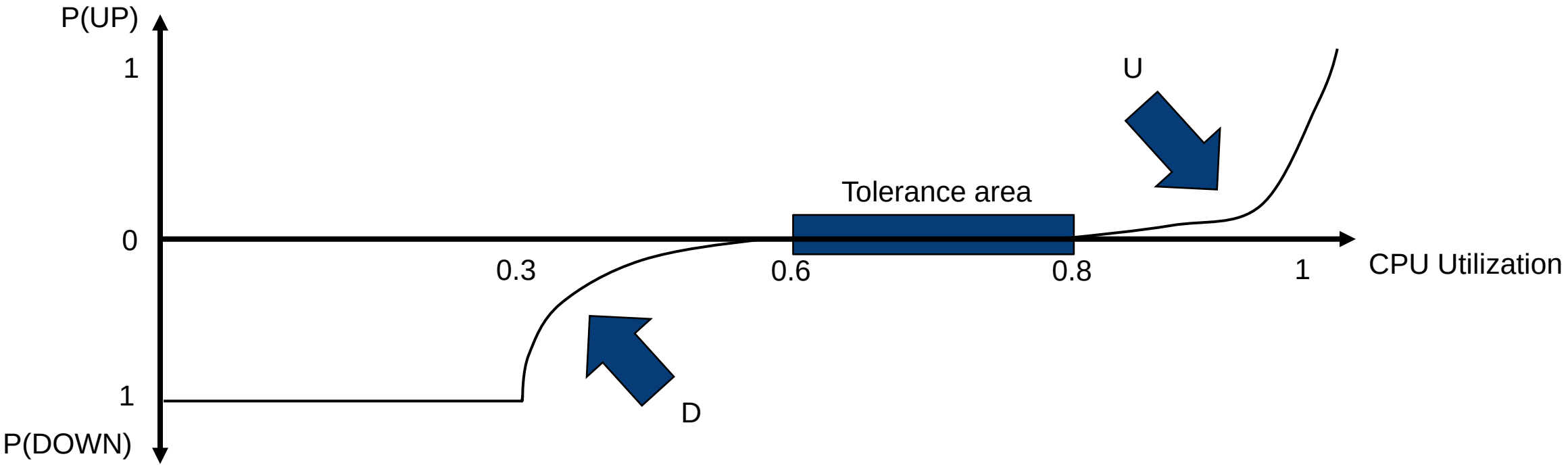
Input: $D : M \rightarrow [0;1]$, $U : M \rightarrow [0;1]$, Distribution W

1: **while** instance is running **do**
2: Collect recent monitoring data into m_t
3: Draw sample r from a uniform distribution in $[0;1]$
4: $P(UP) = U(m_t)$
5: **if** $P(UP) > 0$ **and** $r < P(UP)$ **then**
6: decision = UP
7: **else**
8: $P(DOWN) = D(m_t)$
9: **if** $P(DOWN) > 0$ **and** $r < P(DOWN)$ **then** decision = DOWN
10: **else** decision = HOLD
11: **end if**
12: Submit decision to scaling executor
13: Draw sample w from W and wait for time w
14: **end while**

- Parameters:
  - U – Upscaling function
  - D – Downscaling function
  - W – Waiting time distribution
- U and D take the current instance monitoring data as inputs and output a probability for the corresponding action

Continuous Decentralized Autoscaling – Scaling Policies

- In the following, we consider autoscaling based on CPU utilization
- Example policy:



Continuous Decentralized Autoscaling – Scaling Policies

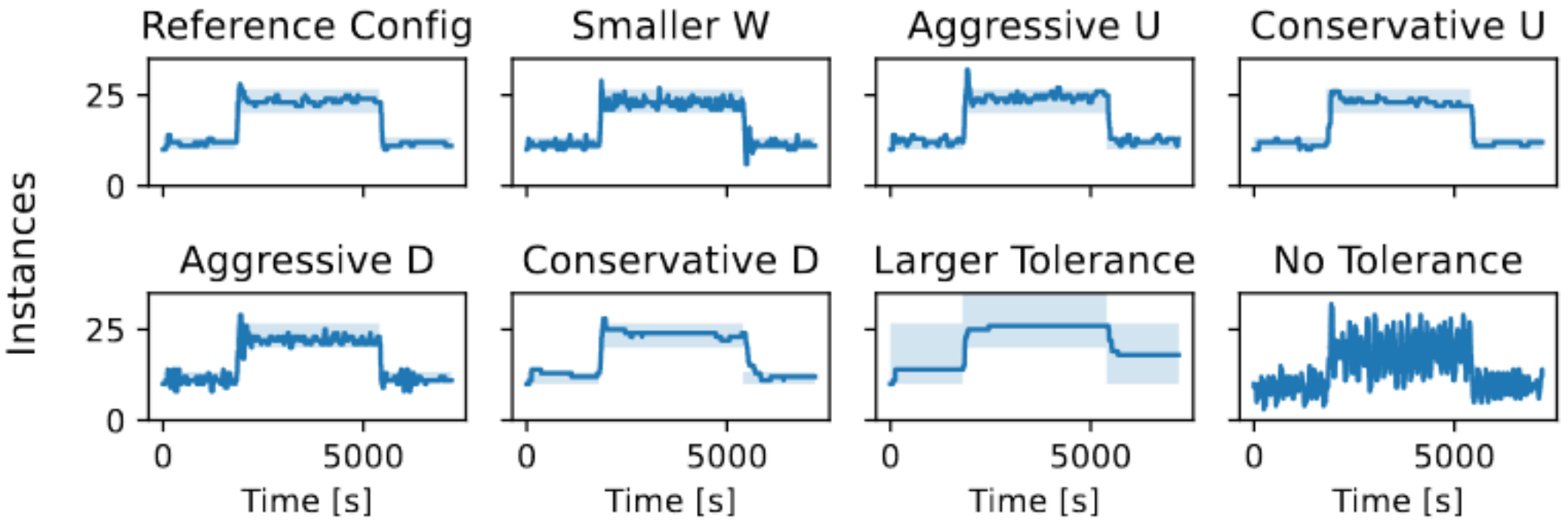


Fig. 7. Examples illustrating the influence of various configuration parameters.

Continuous Decentralized Autoscaling - Implementation

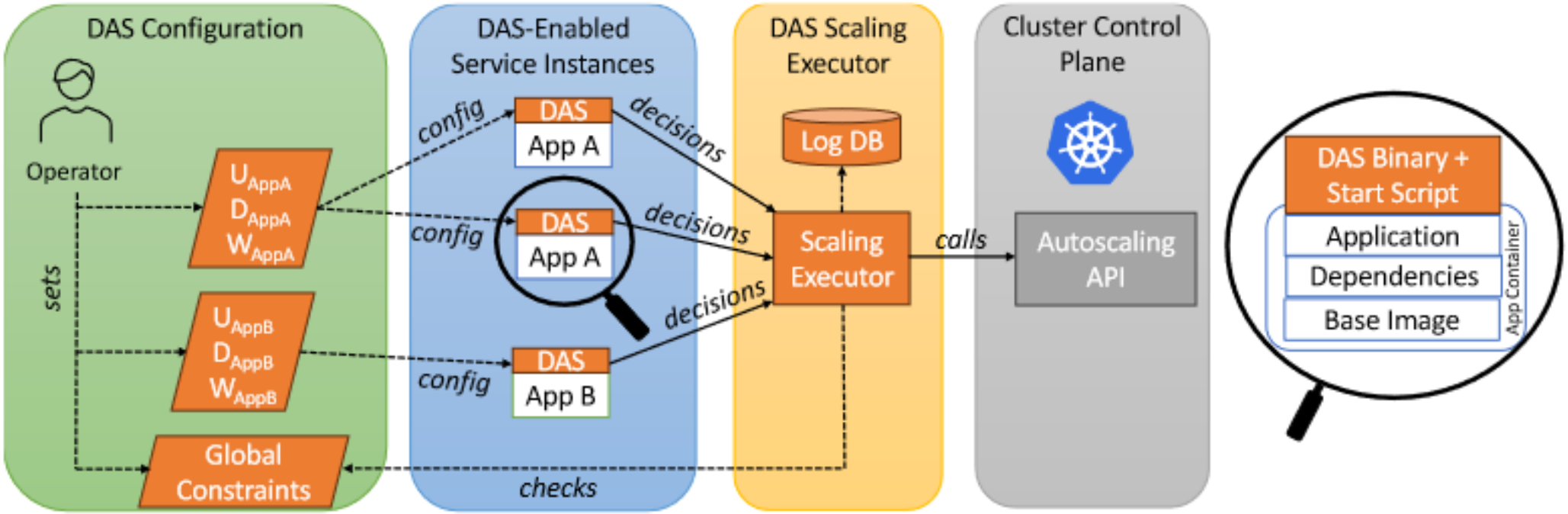
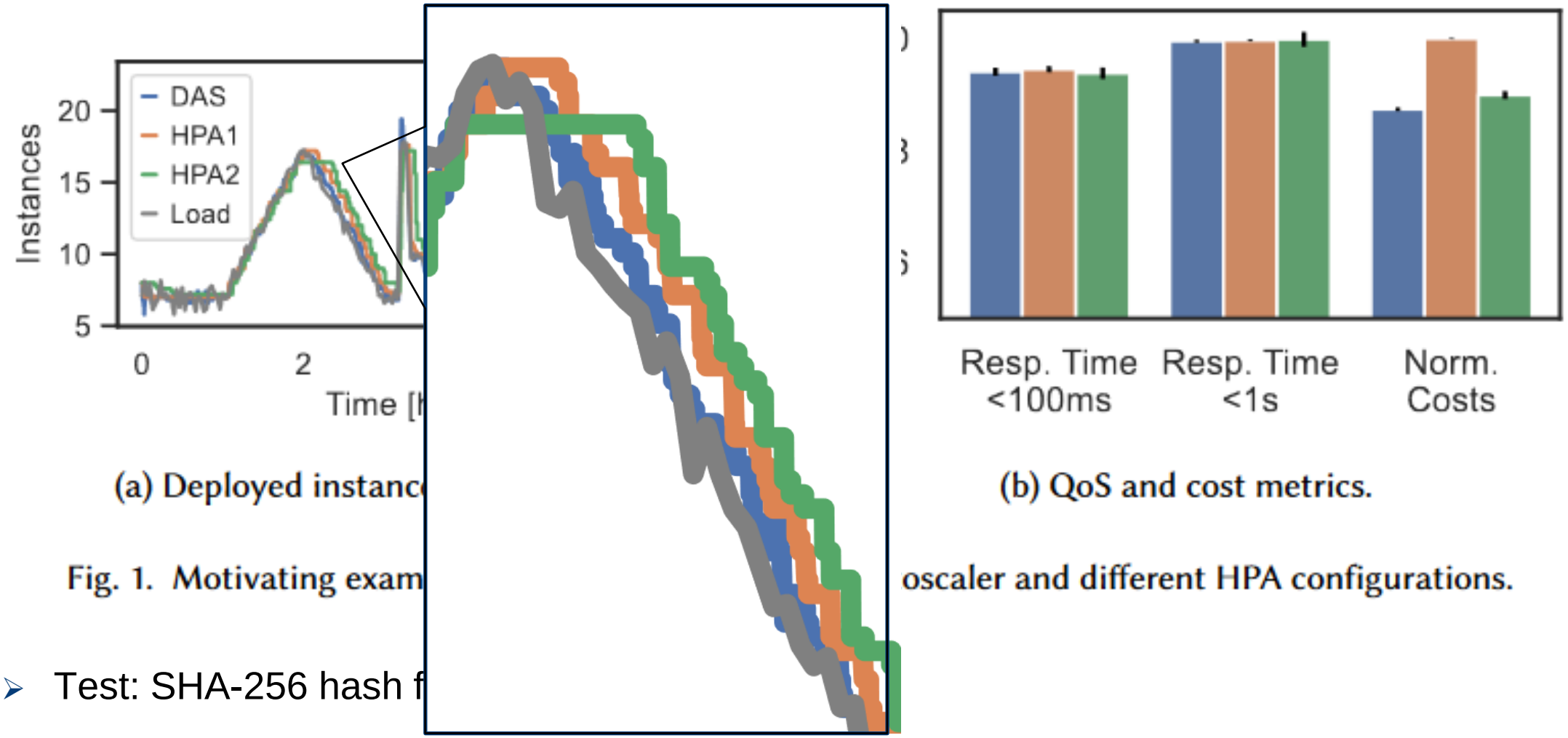


Fig. 9. Proof of Concept Implementation.

Continuous Decentralized Autoscaling – Evaluation Highlights



(a) Deployed instanc

(b) QoS and cost metrics.

Fig. 1. Motivating exam oscaler and different HPA configurations.

- Test: SHA-256 hash f

Continuous Decentralized Autoscaling – Evaluation Highlights

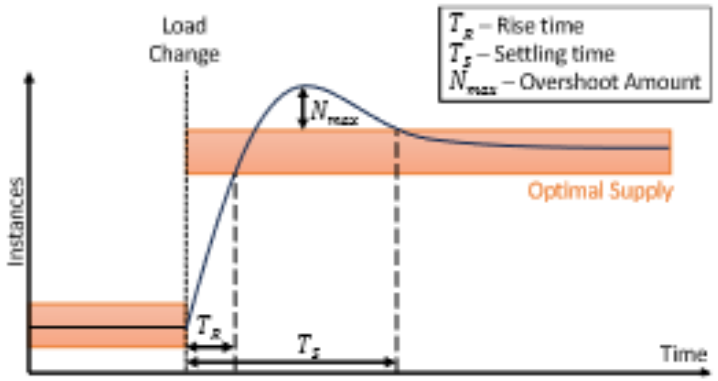


Fig. 4. Metrics for step response.

Scale Factor s	T_R [s]	T_S [s]	N_{max}/s
1	241.6 ± 24.8	268.3 ± 68.2	1.00 ± 1.26
5	246.5 ± 9.9	322.9 ± 26.8	0.64 ± 0.23
10	238.9 ± 10.1	300.0 ± 61.7	0.36 ± 0.42
50	237.8 ± 7.5	311.4 ± 10.2	0.27 ± 0.03
100	240.1 ± 6.1	310.2 ± 7.9	0.21 ± 0.05

Table 1. Scaling metrics for different system scales.

Peak load: 10,000+ req/s

- There is more in the paper…
  - A model for worst-case assessments using discrete-time analysis
  - A simulation that can be used for tuning configuration parameters
  - More real-world evaluation results, incl. Azure trace workloads in a Knative environment

Thank you!