Leveraging Kubernetes Source Code for Performance Simulation

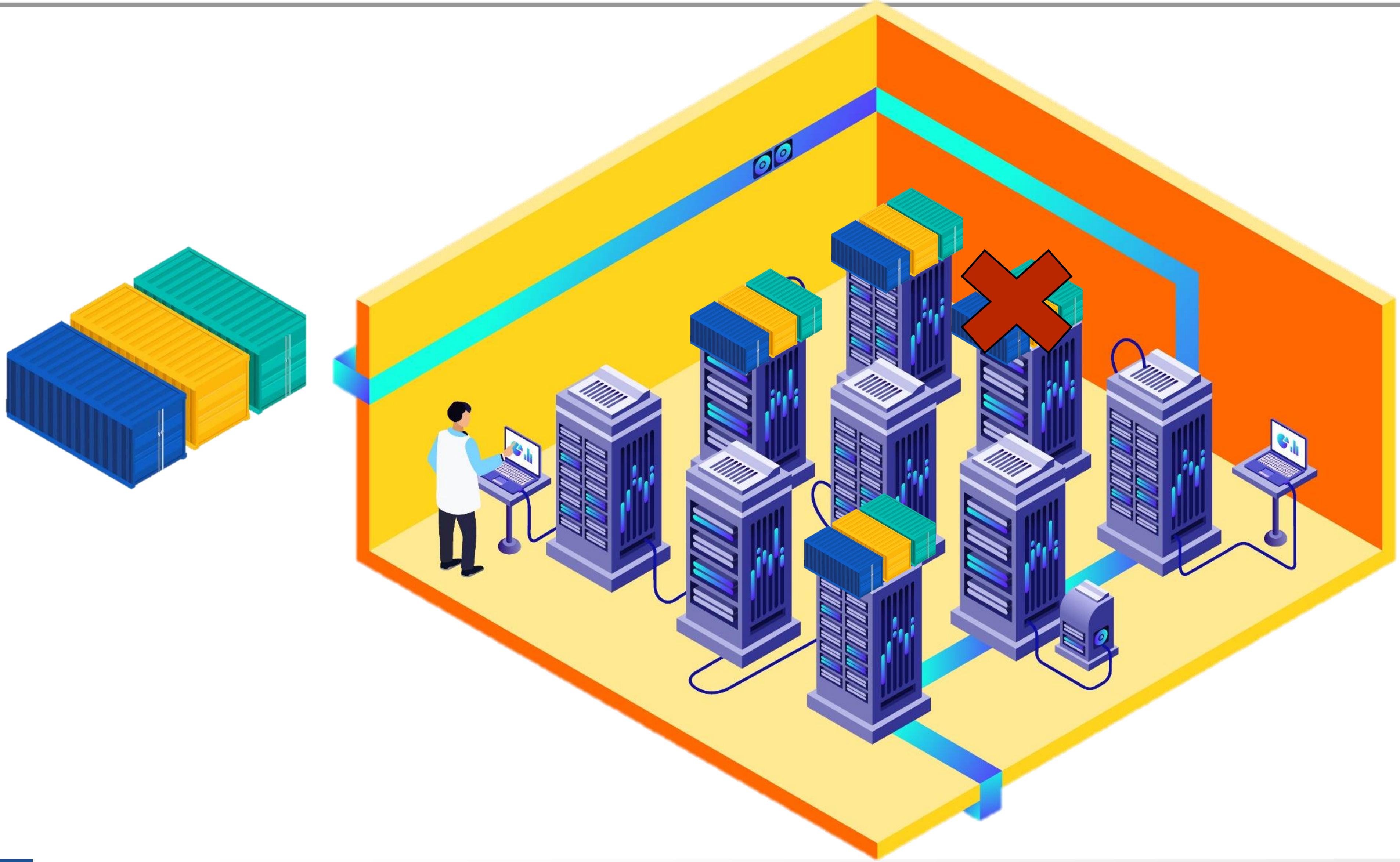
13th Symposium on Software Performance 2022
Session 4: Performance from Cloud to Edge

Martin Straesser, Patrick Haas, Samuel Kounev

09.11.2022

https://se.informatik.uni-wuerzburg.de

Introduction



Introduction

Container orchestration automates the deployment, management, scaling, and networking of containers. […]

Container orchestration is used to automate and manage tasks such as:

- Provisioning and deployment
- Configuration and scheduling
- Resource allocation
- Container availability
- Scaling […]
- Load balancing and traffic routing
- Monitoring container health
- Configuring applications based on the container in which they will run
- Keeping interactions between containers secure

Source: Red Hat Inc.
https://www.redhat.com/en/topics/containers/what-is-container-orchestration

Introduction

- Why container orchestration (CO) in software performance research?
  - CO mechanisms have implications on performance of managed applications [1, 2, 3]
  - Container orchestrators themselves are distributed applications with interesting performance characteristics [4, 5]
  - All of the mentioned tasks are non-trivial and be assessed using different approaches
  - Container orchestration is a high-valued task in production environments as it is (co-) responsible for availability, quality of service, operating costs, resilience, security etc. of cloud applications
- Challenges
  - Holistic view on container orchestration and interdependencies between CO tasks
  - Modeling is hard because of system complexity and continuous updates

Problem

- Goal
  - Test different container orchestration policies
- Obstacles
  - Needs complex technical setup
  - Needs suitable load generation
  - Limited reproducibility
  - Bound by costs
- **Proposed solution**: Performance simulation with integrated container orchestration functions using their original implementation
  - Save costs for experimental evaluation
  - Produce simulation results of high quality

Simulating Container Orchestration

- ➢ We build on the state-of-the-art microservice simulation MiSim [6]

MiSim Features

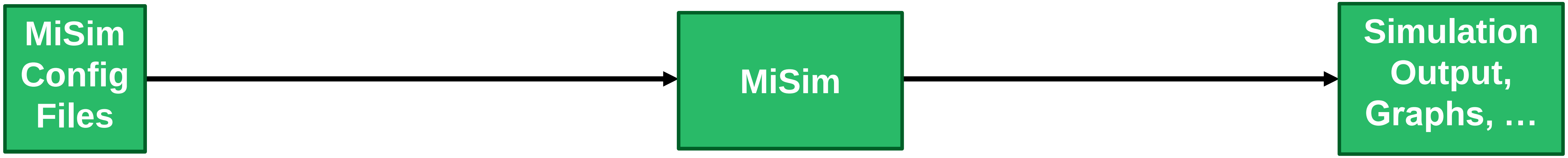
- ✓ Discrete Event Simulation
- ✓ Specialized on microservices
- ✓ Implements CPU performance model
- ✓ Implements several resilience mechanisms
- ✓ Supports fault injections and dynamic workloads

+

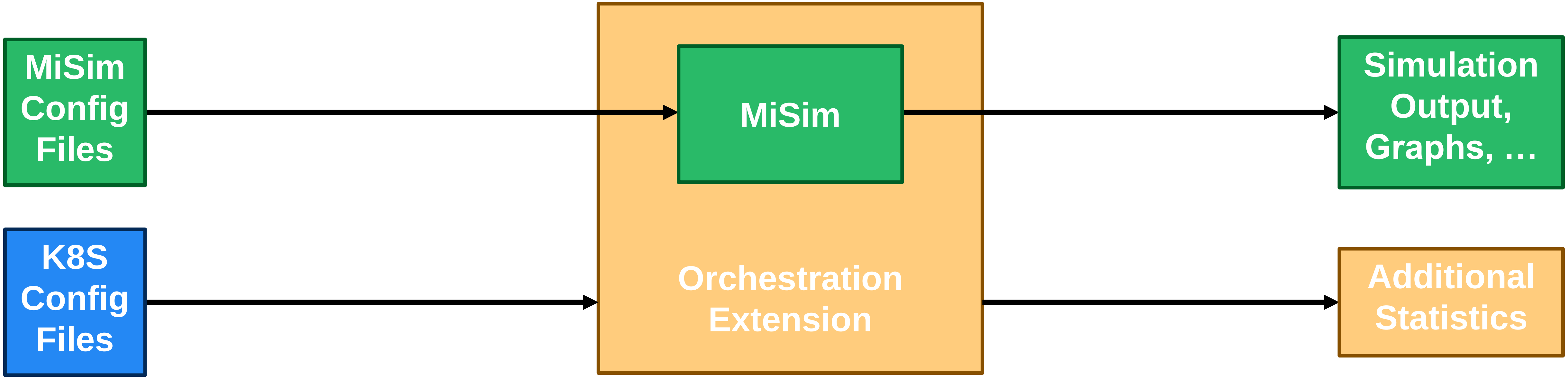
Orchestration Extension

- ✓ Supports all features of MiSim
- ✓ Adds model of nodes and containers
- ✓ Implements several CO mechanisms, e.g. health monitoring, scheduling, …
- ✓ Allows to include original Kubernetes configuration files for deployments, pods, autoscalers etc.
- ✓ Provides interfaces to use original kube-scheduler and cluster-autoscaler in simulation

Framework Overview



Framework Overview



MiSim Config Files

K8S Config Files

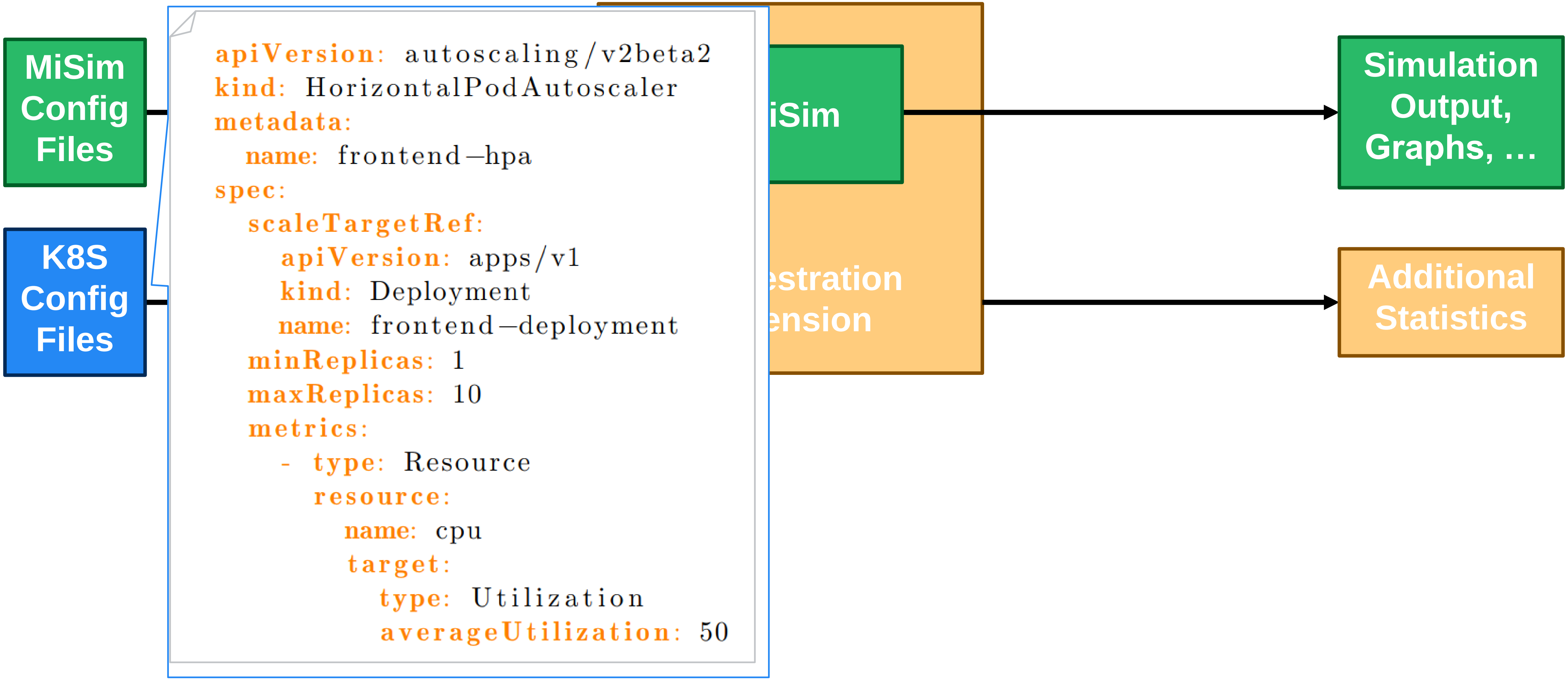
MiSim

Orchestration Extension

Simulation Output, Graphs, …

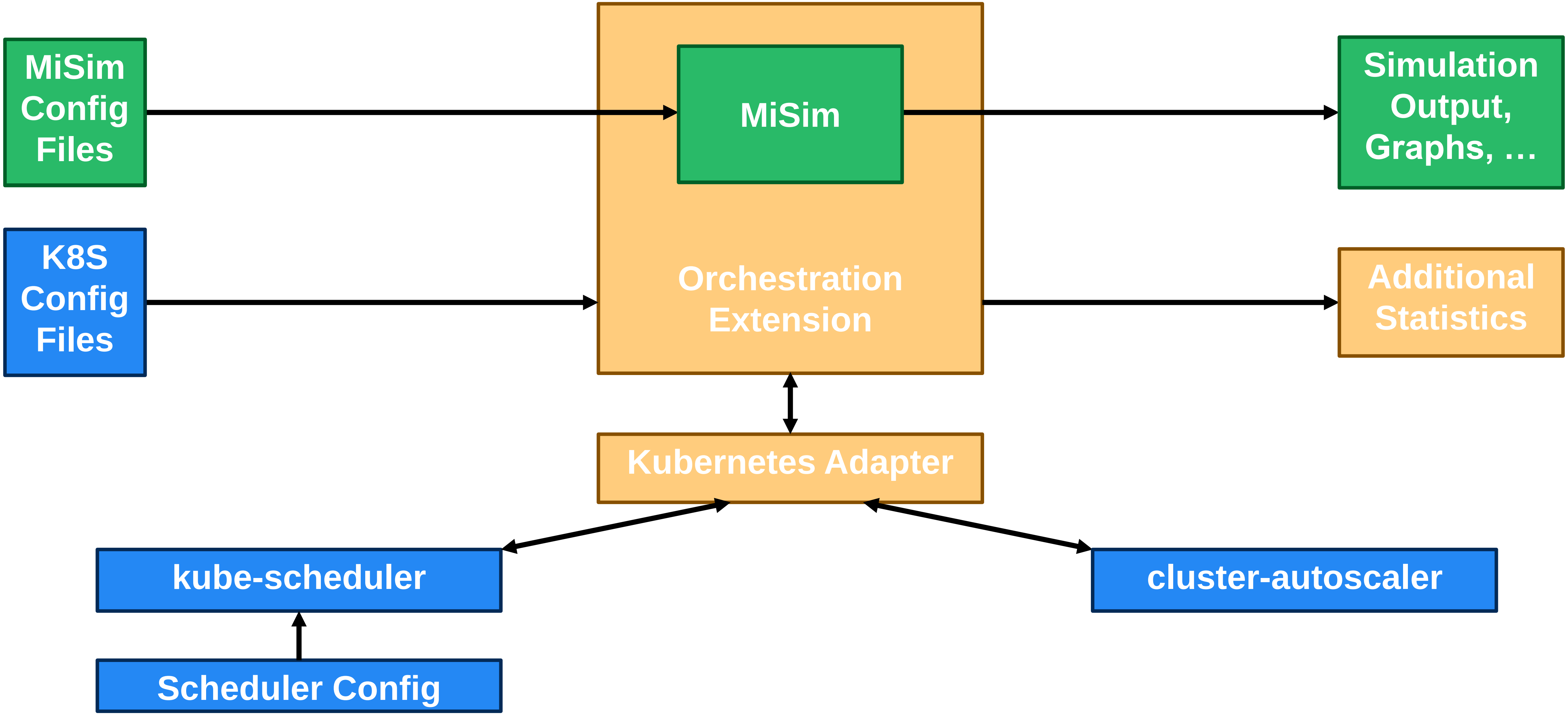
Additional Statistics

Framework Overview



```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: frontend-deployment
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
```

Framework Overview



MiSim Config Files

K8S Config Files

MiSim

Orchestration Extension

Simulation Output, Graphs, …

Additional Statistics

Kubernetes Adapter

kube-scheduler

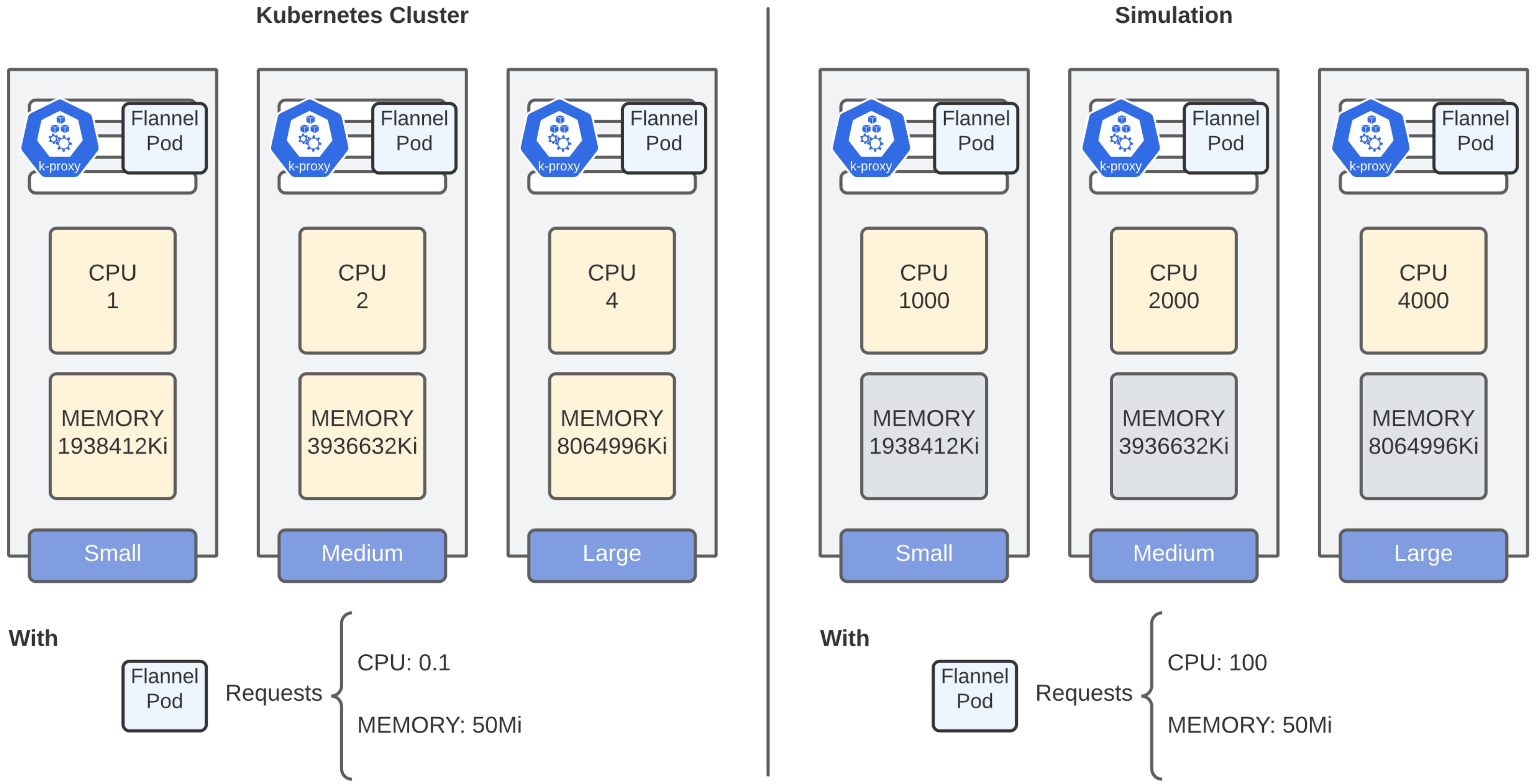
cluster-autoscaler

Scheduler Config

How It Works

- Question 1: Kubernetes components work normally in real-time with real cluster resources, how does this fit a discrete event simulation?
- Answer
  - By having a closer look at Kubernetes internal architecture, we note that communication is realized with event-based HTTP watch streams, which are emitted by the kube-apiserver
  - Our Kubernetes adapter implements parts of the kube-apiserver, it basically translates MiSim events to Kubernetes events and vice versa
- Question 2: What about the performance overhead?
- Answer
  - Performance overhead is around 10% in terms of simulation runtime, no significantly more resource usage

kube-scheduler - Evaluation Setup



Kubernetes Cluster

Small	Medium	Large
k-proxy Flannel Pod	k-proxy Flannel Pod	k-proxy Flannel Pod
CPU 1	CPU 2	CPU 4
MEMORY 1938412Ki	MEMORY 3936632Ki	MEMORY 8064996Ki

With Flannel Pod Requests

- CPU: 0.1
- MEMORY: 50Mi

Simulation

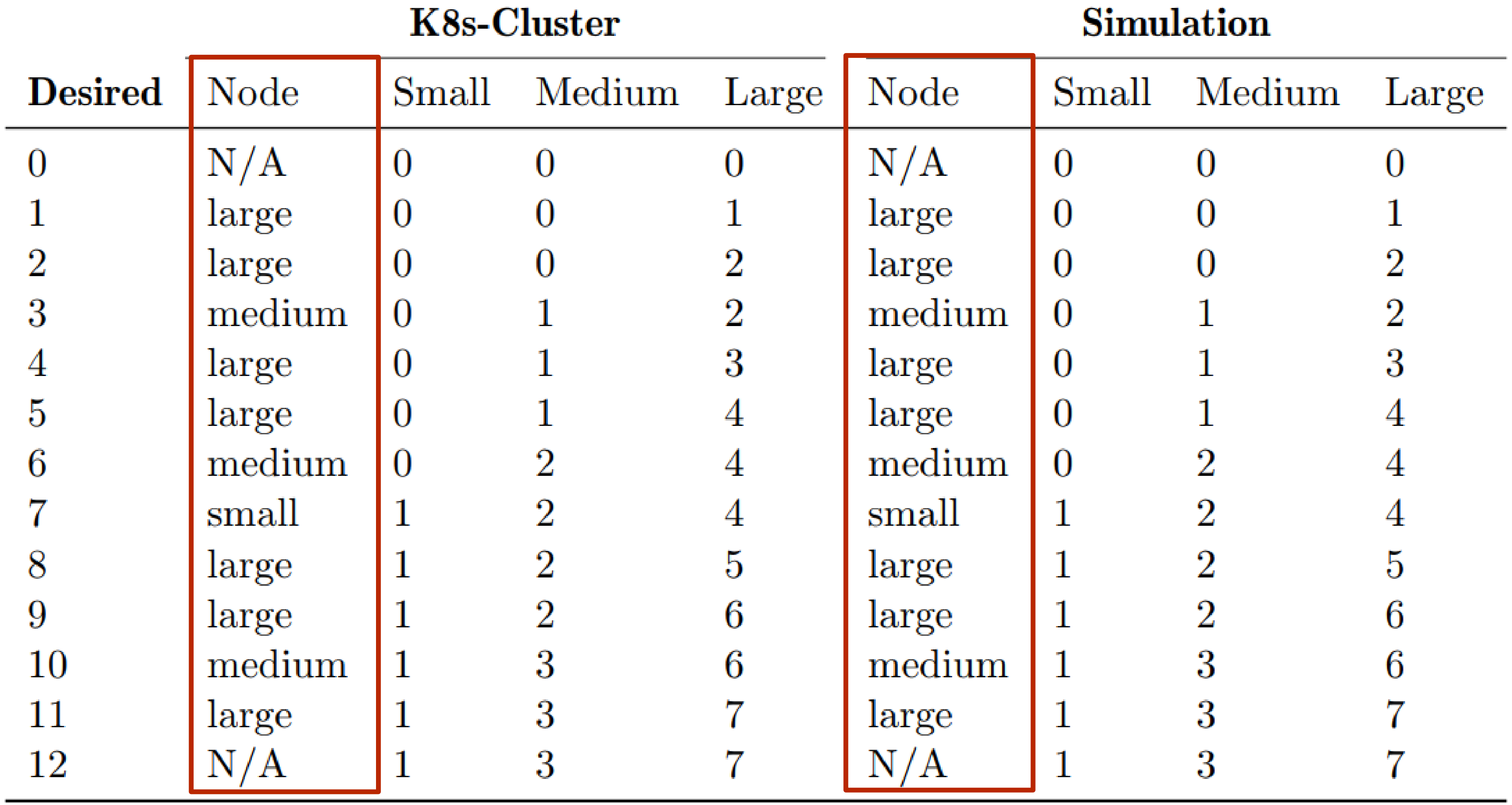
Small	Medium	Large
k-proxy Flannel Pod	k-proxy Flannel Pod	k-proxy Flannel Pod
CPU 1000	CPU 2000	CPU 4000
MEMORY 1938412Ki	MEMORY 3936632Ki	MEMORY 8064996Ki

With Flannel Pod Requests

- CPU: 100
- MEMORY: 50Mi

kube-scheduler - Evaluation

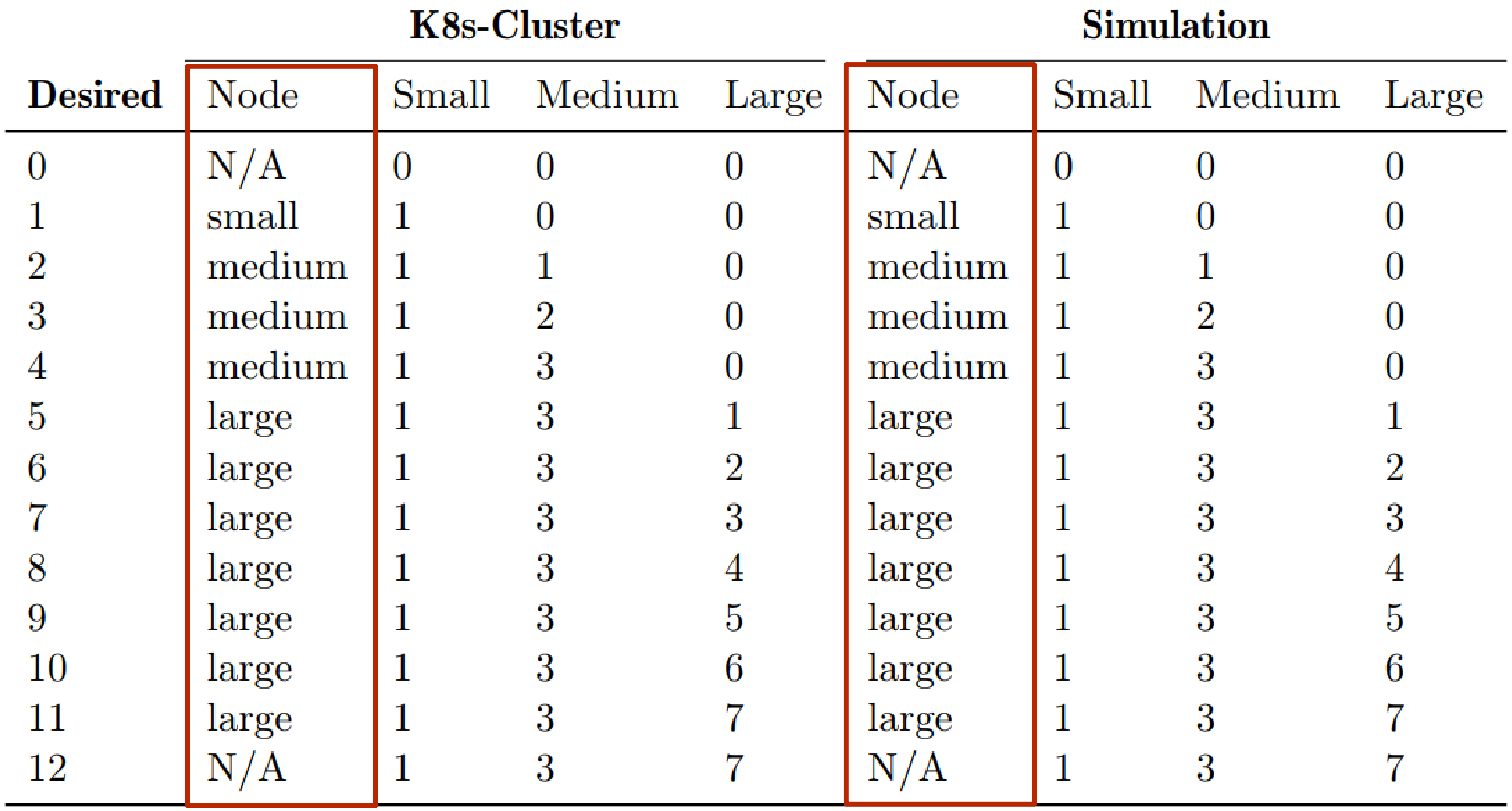
- Scheduling Policy: NodeResourceBalancedAllocation



	K8s-Cluster				Simulation			
Desired	Node	Small	Medium	Large	Node	Small	Medium	Large
0	N/A	0	0	0	N/A	0	0	0
1	large	0	0	1	large	0	0	1
2	large	0	0	2	large	0	0	2
3	medium	0	1	2	medium	0	1	2
4	large	0	1	3	large	0	1	3
5	large	0	1	4	large	0	1	4
6	medium	0	2	4	medium	0	2	4
7	small	1	2	4	small	1	2	4
8	large	1	2	5	large	1	2	5
9	large	1	2	6	large	1	2	6
10	medium	1	3	6	medium	1	3	6
11	large	1	3	7	large	1	3	7
12	N/A	1	3	7	N/A	1	3	7

kube-scheduler - Evaluation

- Scheduling Policy: NodeResourcesFit (CPU) – MostAllocated

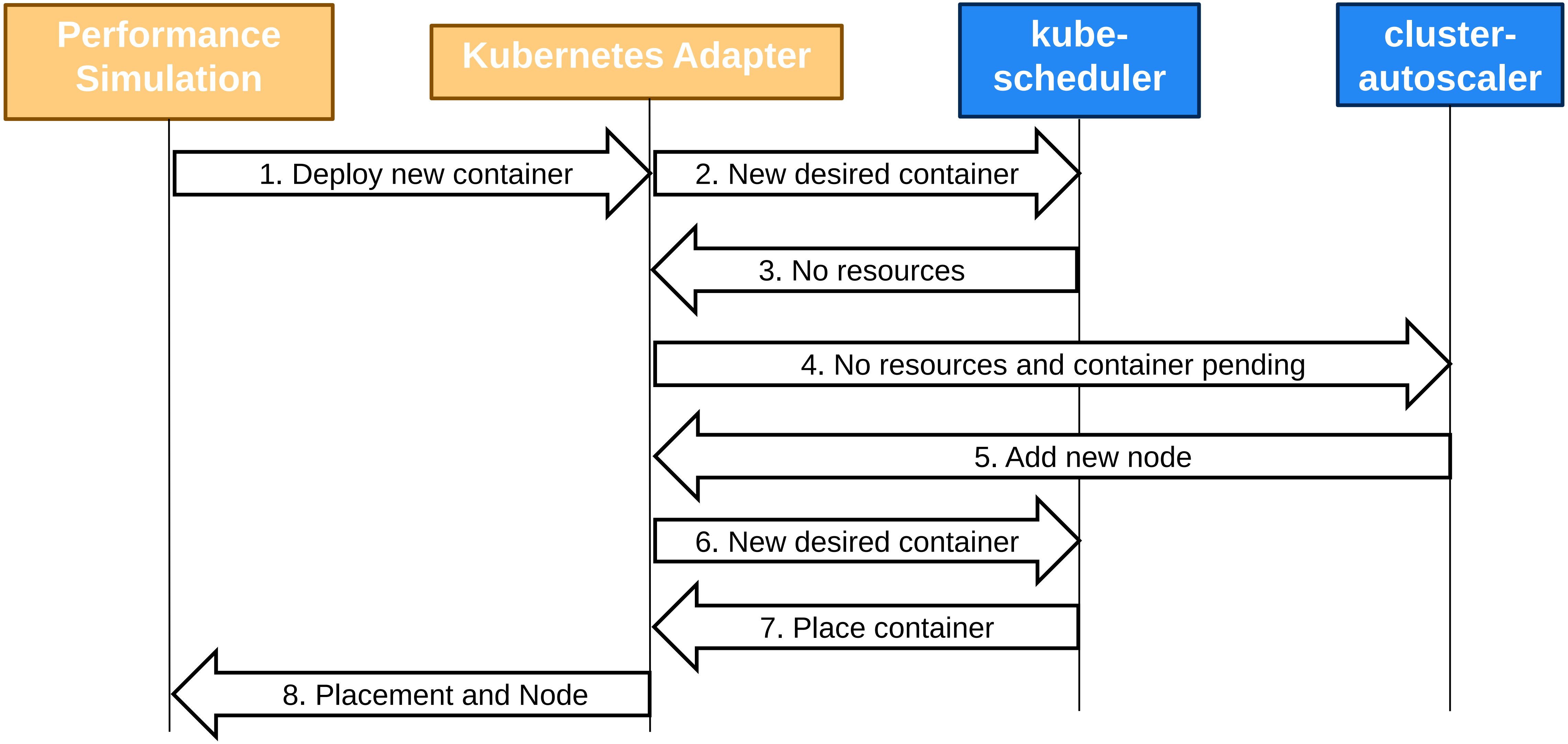


	K8s-Cluster				Simulation			
Desired	Node	Small	Medium	Large	Node	Small	Medium	Large
0	N/A	0	0	0	N/A	0	0	0
1	small	1	0	0	small	1	0	0
2	medium	1	1	0	medium	1	1	0
3	medium	1	2	0	medium	1	2	0
4	medium	1	3	0	medium	1	3	0
5	large	1	3	1	large	1	3	1
6	large	1	3	2	large	1	3	2
7	large	1	3	3	large	1	3	3
8	large	1	3	4	large	1	3	4
9	large	1	3	5	large	1	3	5
10	large	1	3	6	large	1	3	6
11	large	1	3	7	large	1	3	7
12	N/A	1	3	7	N/A	1	3	7

kube-scheduler - Evaluation

- Integrated kube-scheduler capable of reflecting different scheduling strategies in simulation
- We were even able to reproduce an active GitHub issue of the kube-scheduler
- However, not all custom scheduling plugins can be simulated out of the box (e.g. when nodes are grouped in zones and you want to have some “zone-aware” scheduling)
  - Solution: Provide “black-box” Kubernetes configuration files (e.g. node descriptions)
  - Simulation can not directly understand them but will forward the information to the kube-scheduler

Next step forward – cluster-autoscaler



Conclusion

Problem

- Container orchestrators fulfill many performance-relevant tasks
- Modeling hard, experimental evaluation expensive

Idea

- Performance simulation with integrated container orchestration functions using their original implementation

Benefits

- Simulation with increased accuracy and more use cases
- Analysis of container orchestration policies

References

[1] M. Straesser, J. Grohmann, J. von Kistowski, S. Eismann, A. Bauer, and S. Kounev. 2022. Why Is It Not Solved Yet? Challenges for Production-Ready Autoscaling. In Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering (ICPE '22).

[2] E. Truyen, D. Van Landuyt, B. Lagaisse, and W. Joosen. 2019. Performance overhead of container orchestration frameworks for management of multi-tenant database deployments. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19).

[3] E. Truyen, M. Bruzek, D. Van Landuyt, B. Lagaisse and W. Joosen. 2018. Evaluation of Container Orchestration Systems for Deploying and Managing NoSQL Database Clusters. In IEEE 11th International Conference on Cloud Computing (CLOUD).

[4] Y. Pan, I. Chen, F. Brasileiro, G. Jayaputera and R. Sinnott. 2019. A Performance Comparison of Cloud-Based Container Orchestration Tools. In IEEE International Conference on Big Knowledge (ICBK).

[5] I. M. A. Jawarneh et al. Container Orchestration Engines: A Thorough Functional and Performance Comparison. 2019. In IEEE International Conference on Communications (ICC).

[6] S. Frank, L. Wagner, A. Hakamian, M. Straesser and A. van Hoorn. 2022. MiSim: A Simulator for Resilience Assessment of Microservice-based Architectures. In IEEE International Conference on Software Quality, Reliability and Security (QRS).