@Large Research
Massivizing Computer Systems

http://atlarge.science

Continuum

Automate Infrastructure Deployment and Benchmarking in the Compute Continuum

Matthijs Jansen, Linus Wagner,
Animesh Trivedi, Alexandru Iosup

m.s.jansen@vu.nl
atlarge.science/offsense

VU VRIJE UNIVERSITEIT AMSTERDAM

Use Case: Video Processing

Requirement: Process live video using ML

Problem: Device may not have resources

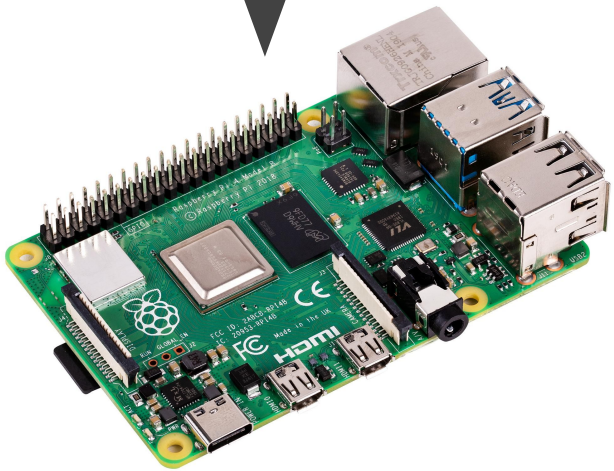
Solution: Offload data

Task Offloading

Offloading targets?

Available resources?

Requirements?

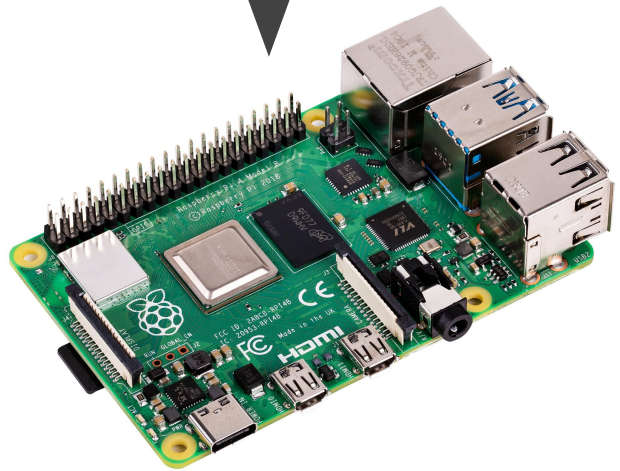


What Deployment Works?

Simple solution →

Let’s deploy

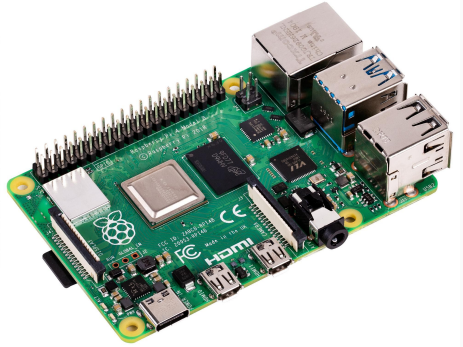
And find out



Let’s Deploy: Infrastructure

Requirement: Infrastructure
→ Very costly

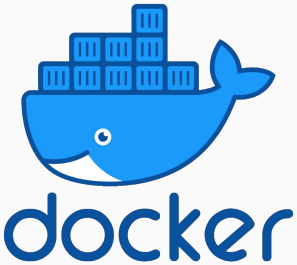
May be infeasible to get!



Let’s Deploy: Software

Requirement: Software
→ Very costly

May be infeasible to do!



L

We can not test every deployment by hand

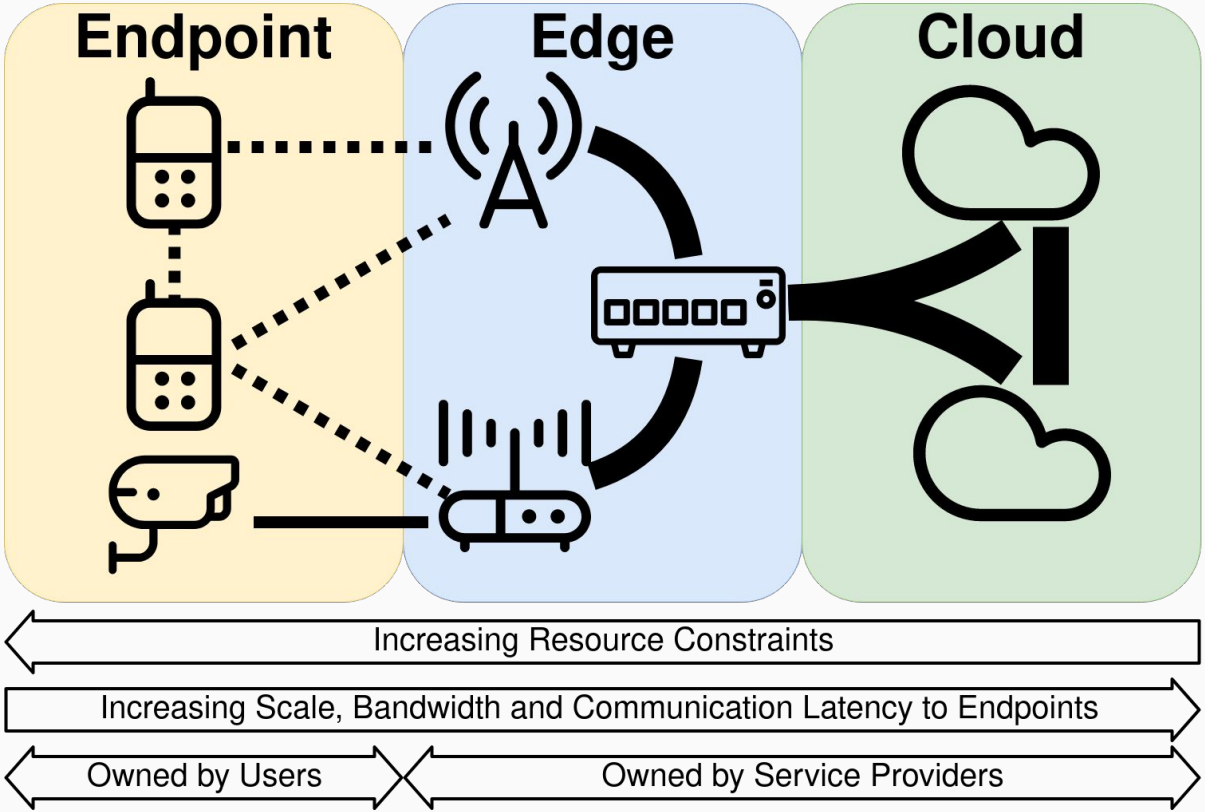
Design Space Exploration

Big performance differences between deployments

Can’t test all:

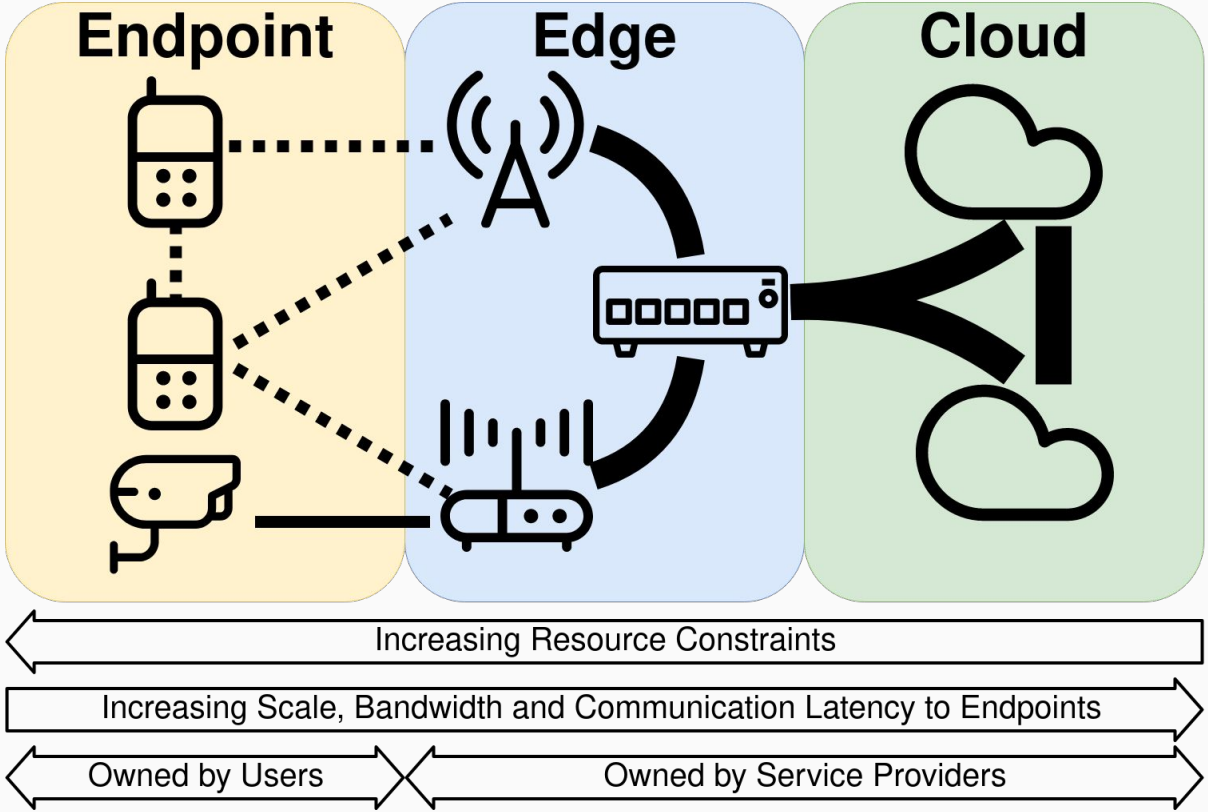
→ Analyze

→ Prune



Design Space Dimensions

- What to offload?
- Services to use?
- Parties involved?
- Resources?
- Networks?
- Requirements?
- …



Analysis is very costly and difficult

Problem Summary

1. Many deployments in the continuum

   → Big performance differences

2. Unlikely to find a satisfactory deploym. in one go

   → Even with expert knowledge

3. Need to iterate over many deployments quickly

   → To expensive with real-world deployments

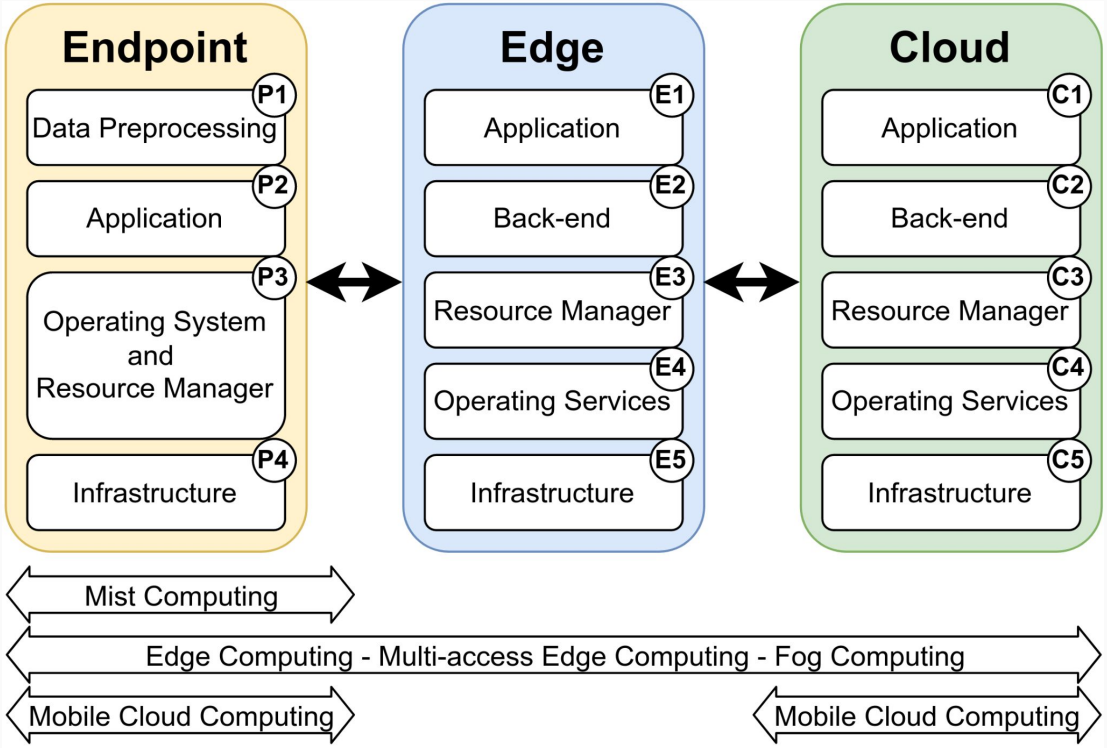
Continuum

Automate cloud-edge infrastructure deployment and benchmarking in the compute continuum

https://github.com/atlarge-research/continuum

SPEC-RG Reference Architecture for Cont.

Common components in continuum



The SPEC-RG Reference Architecture for the Compute Continuum, Matthijs Jansen et al, CCGRID'23

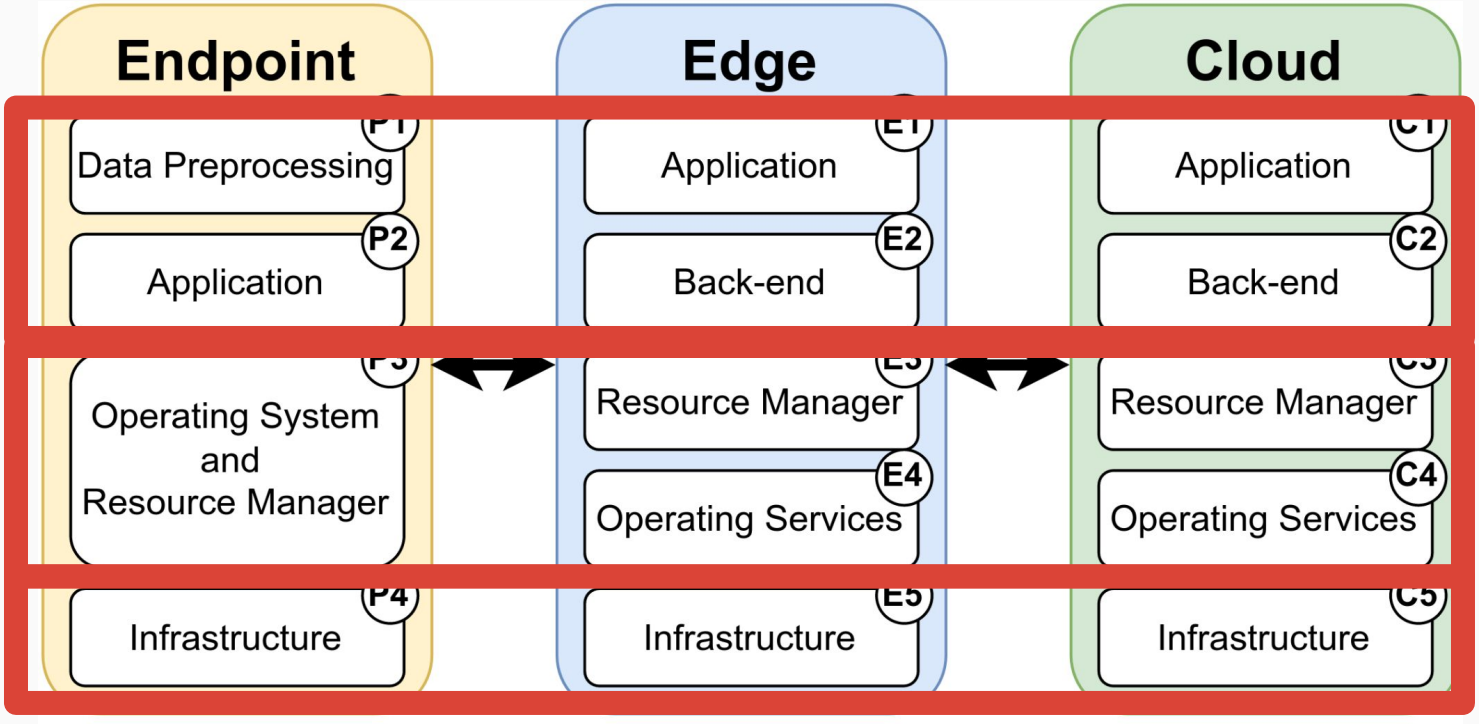
Implementation in specific components

SPEC-RG Reference Architecture

Application Benchmark

Software

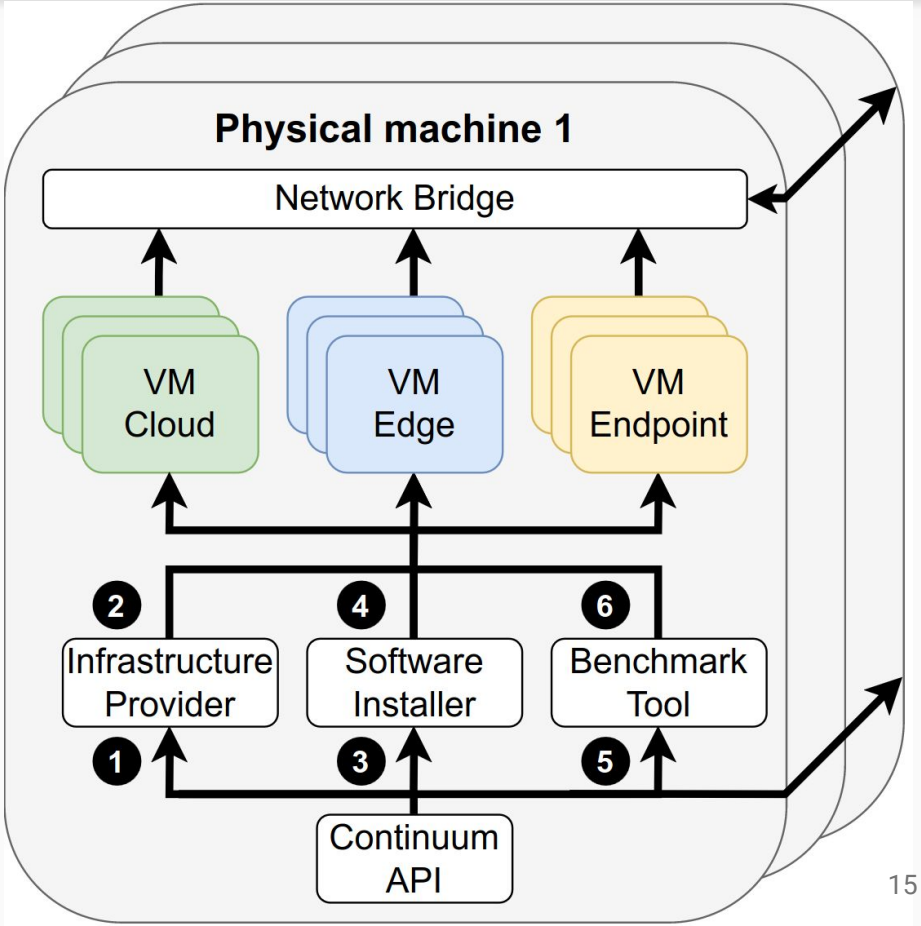
Infrastruct.



The Continuum Framework

Design principles

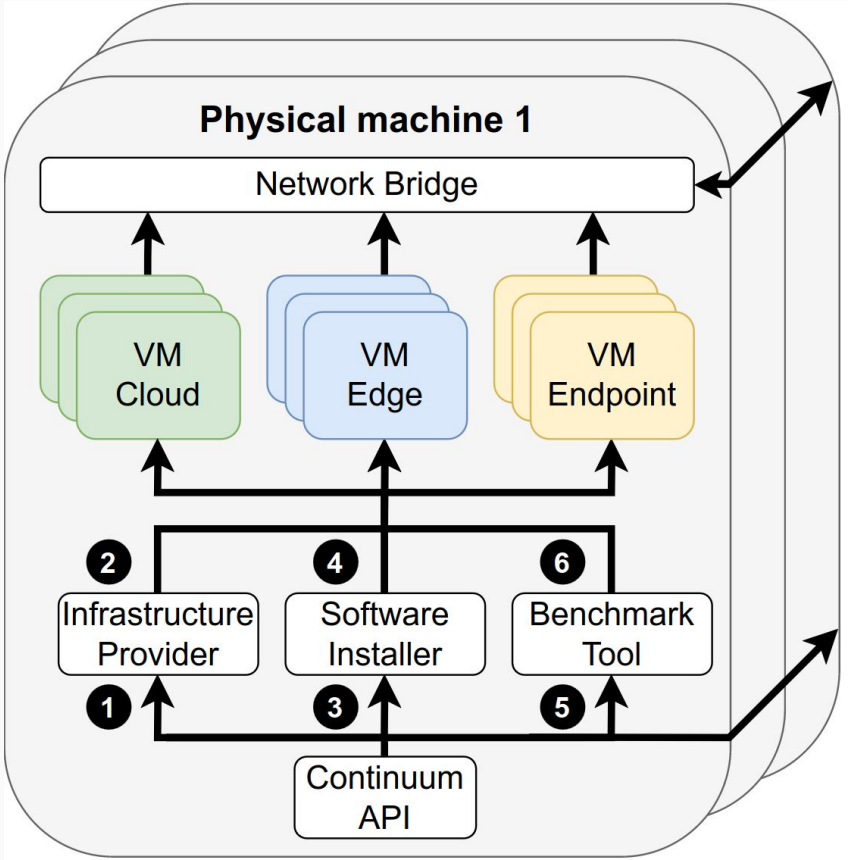
1. Accurate
   → Hardware deployment
2. Automated
   → Scripting
3. Extendable
   → Modular design
4. Flexible
   → Emulation



Step 1: Infrastructure Provisioning

Infrastructure providers:

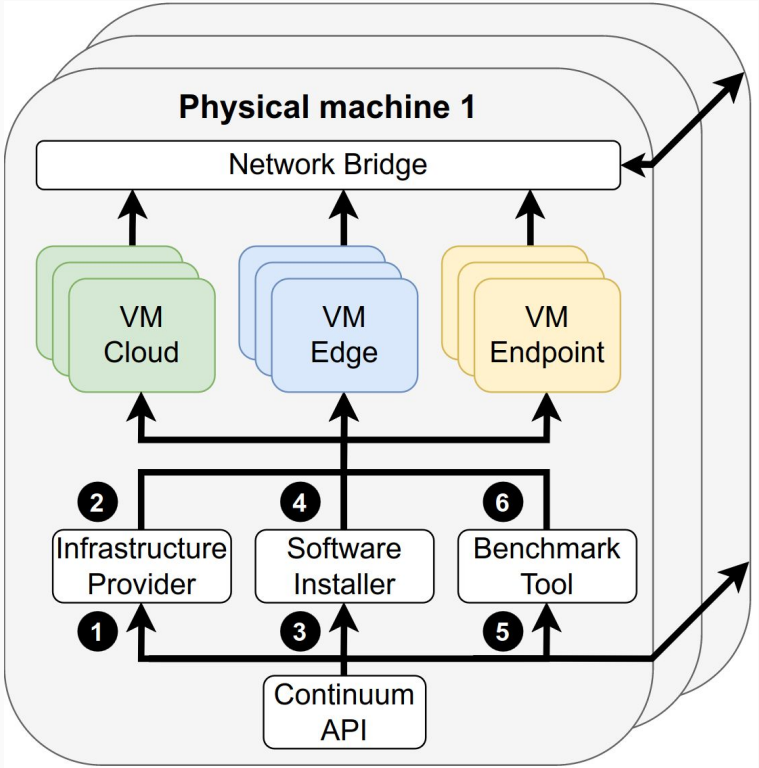
- Baremetal
- QEMU (VMs)
- Google Cloud (VMs)
- Docker (containers)



Step 1: Infrastructure Provisioning

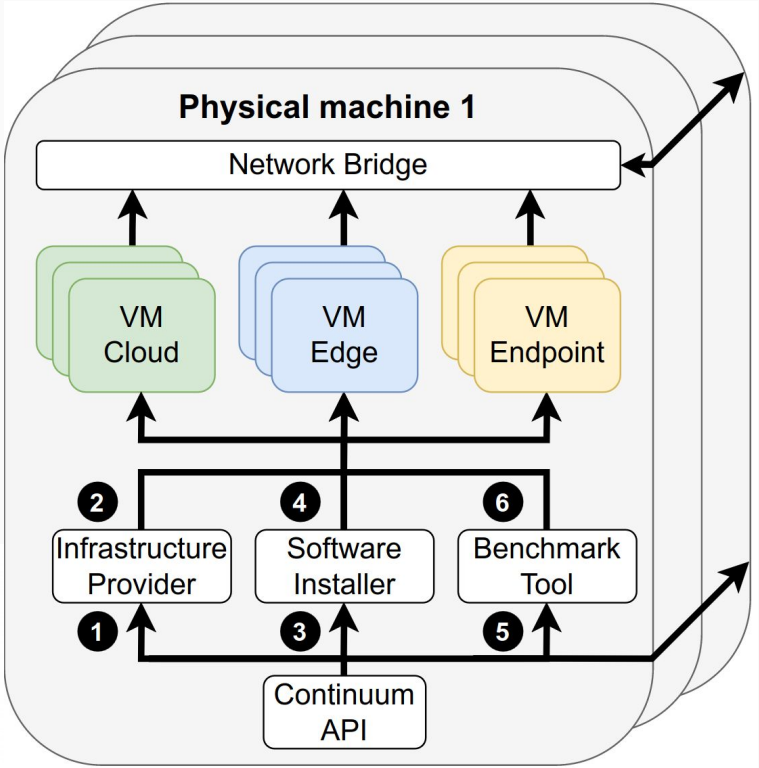
config-example.cfg

Provider	= GCP
Cloud-VMs	= 5
Cloud-cores	= 8
Cloud-memory	= 16 GB
Storage-read	= 1 GBps
Cloud-Edge-Latency	= 15 ms



Step 2: Software Installation

- Install software on infra
- Automate it once, repeat
- Minimal restrictions



kubernetes

docker

OPENFAAS

ANSIBLE

Step 2: Software Installation

config-example.cfg | Resource-manager = Kubernetes

- Create module
- Ansible automation

```
continuum/
├── infrastructure/
│   └── qemu/
│       ├── qemu.py
│       ├── cloud_start.yml
│       ├── edge_start.yml
│       └── endpoint_start.yml
└── resource_manager/
    └── kubernetes/
        ├── kubernetes.py
        ├── master_start.yml
        └── worker_start.yml
```

Step 2: Software Installation

master-start.yml

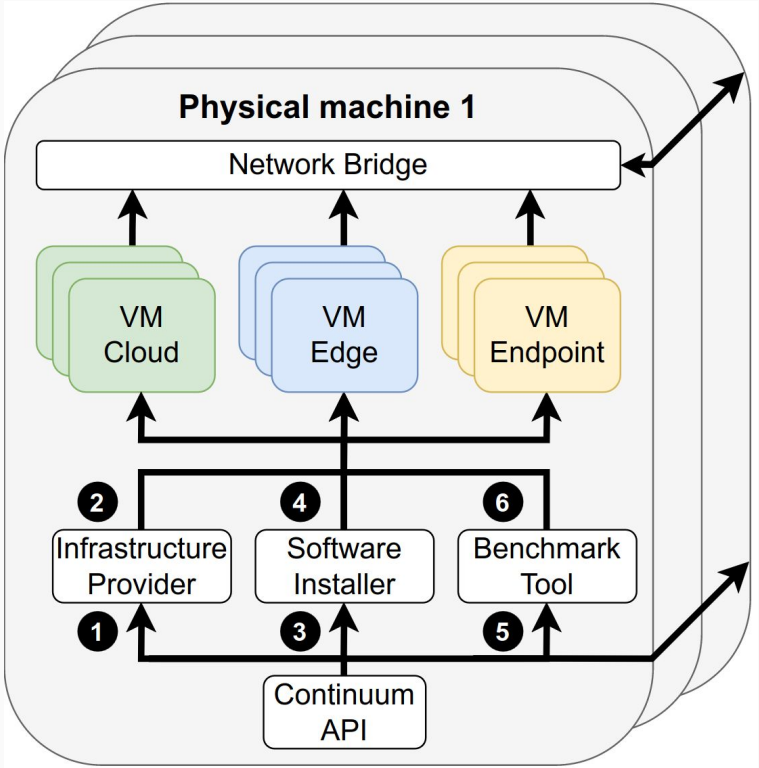
```
- hosts: cloudcontroller
  tasks:
  - name: Initialize the Kubernetes cluster using kubeadm
    command:
      kubeadm init
        --apiserver-advertise-address={{ cloud_ip }}
        --apiserver-cert-extra-sans={{ cloud_ip }}
        --node-name {{ ansible_hostname }}
        --pod-network-cidr=10.244.0.0/16
```

Step 3: Benchmark, Observe

Define:

- Application
- Deployment method
- Application arguments

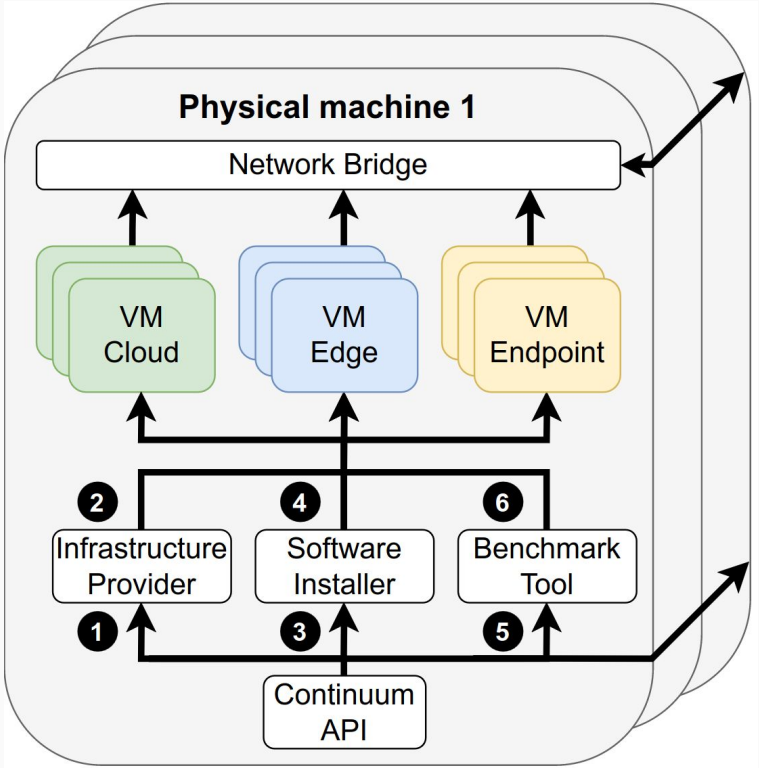
Built-in observability + custom



Step 3: Benchmark, Observe

config-example.cfg

```
Resource-manager  = Kubernetes
App               = my_app
#cloud-apps       = 10
#endpoint-apps    = 100
my-app-argument-1 = ...
```



Step 3: Benchmark, Observe

```
continuum/
└── application/
    └── myapp/
        ├── myapp.py
        └── deploy_kube.yml
```

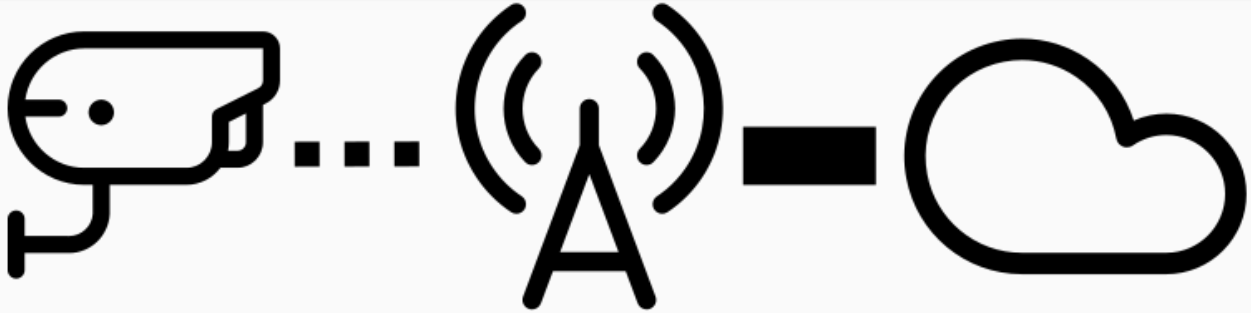
deploy_kube.yml

```
- hosts: cloudcontroller
  tasks:
    - name: Create job file
      shell:
        cat > "/home/{{ username }}/job.yaml" <<EOF
        kind: Job
        containers:
        - name: {{ app_name }}
          image: {{ image }}
          resources:
            memory: "{{ memory_req }}Mi"
            cpu: {{ cpu_req }}
          env:
          - name: MY_VARIABLE
            value: "{{ var_1 }}"
        EOF

    - name: Launch jobs
      command: kubectl create -f "/home/{{ username }}/jobs"
```

Evaluation

Use Case: Video Processing



Endpoint: Security camera, X images/second

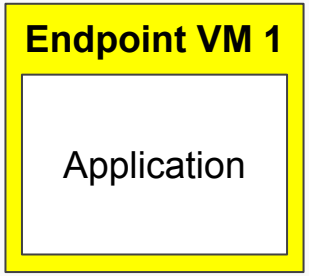
Processing: ML

Question: Where to process?

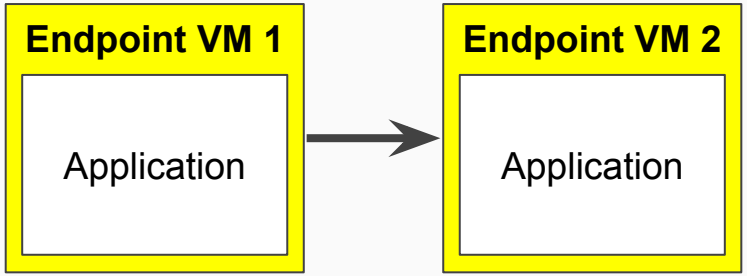
Metrics: End-to-end latency, utilization

Deployment Scenarios

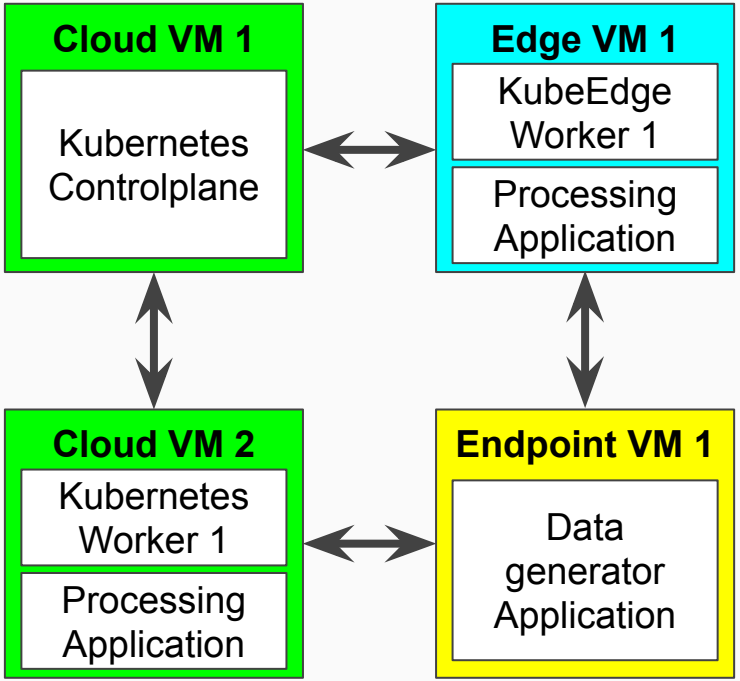
Local processing



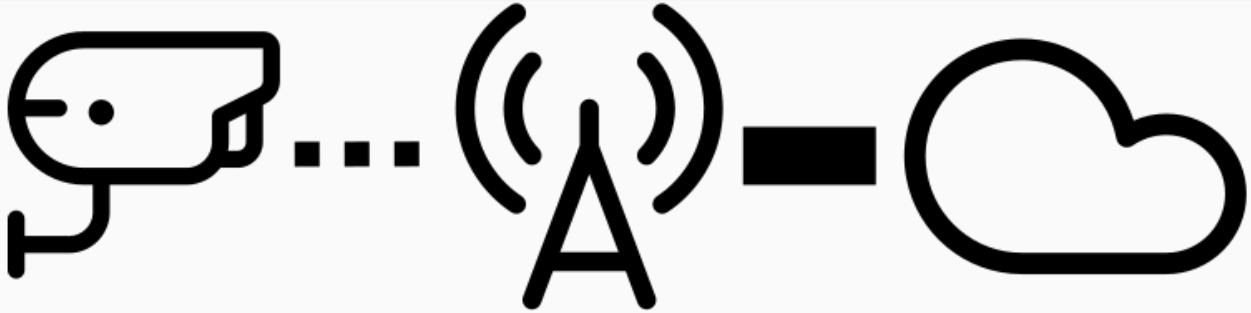
Endpoint offloading



Cloud / Edge offloading



Setting



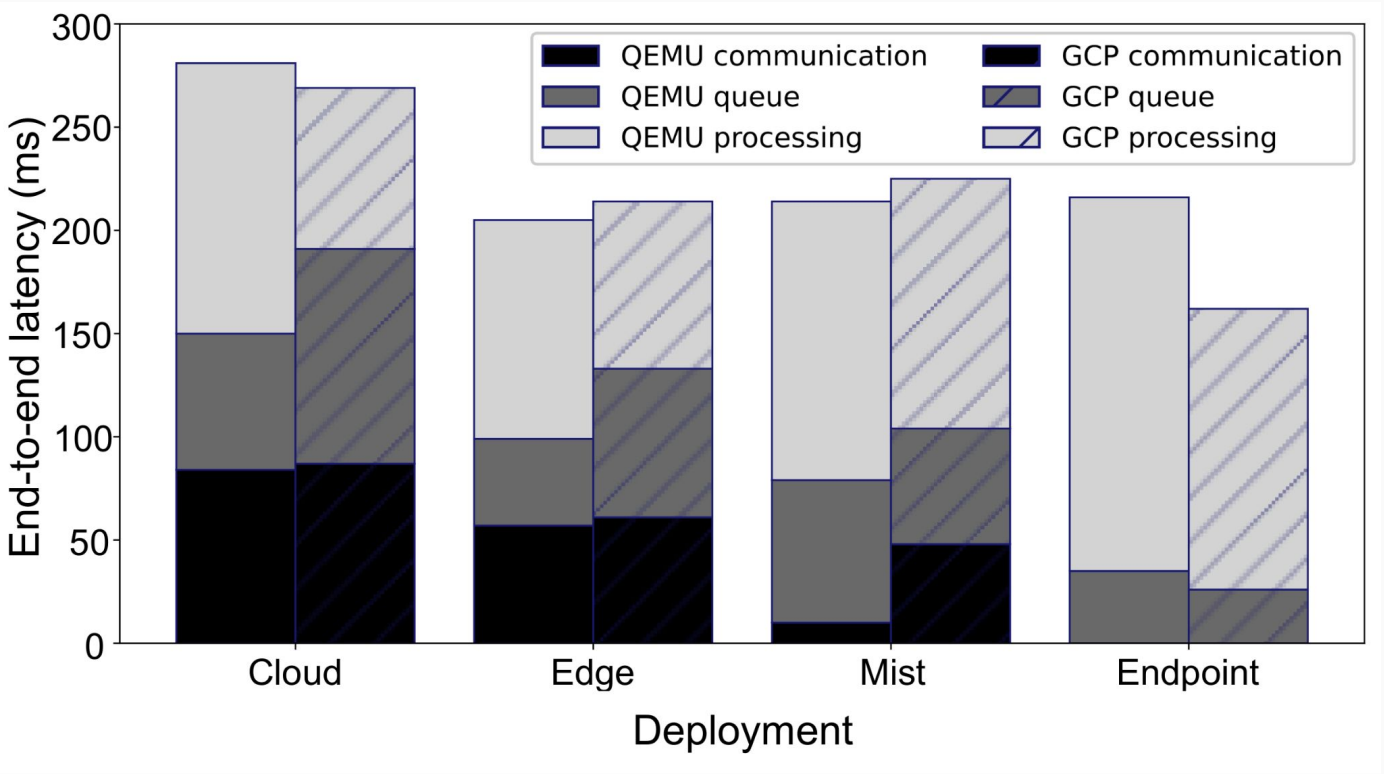
Infra Provider: Google Cloud, QEMU

Resources: Endpoints < Edge < Cloud

Latency to endpoint: Endpoint < Edge < Cloud

End-to-end Latency Breakdown

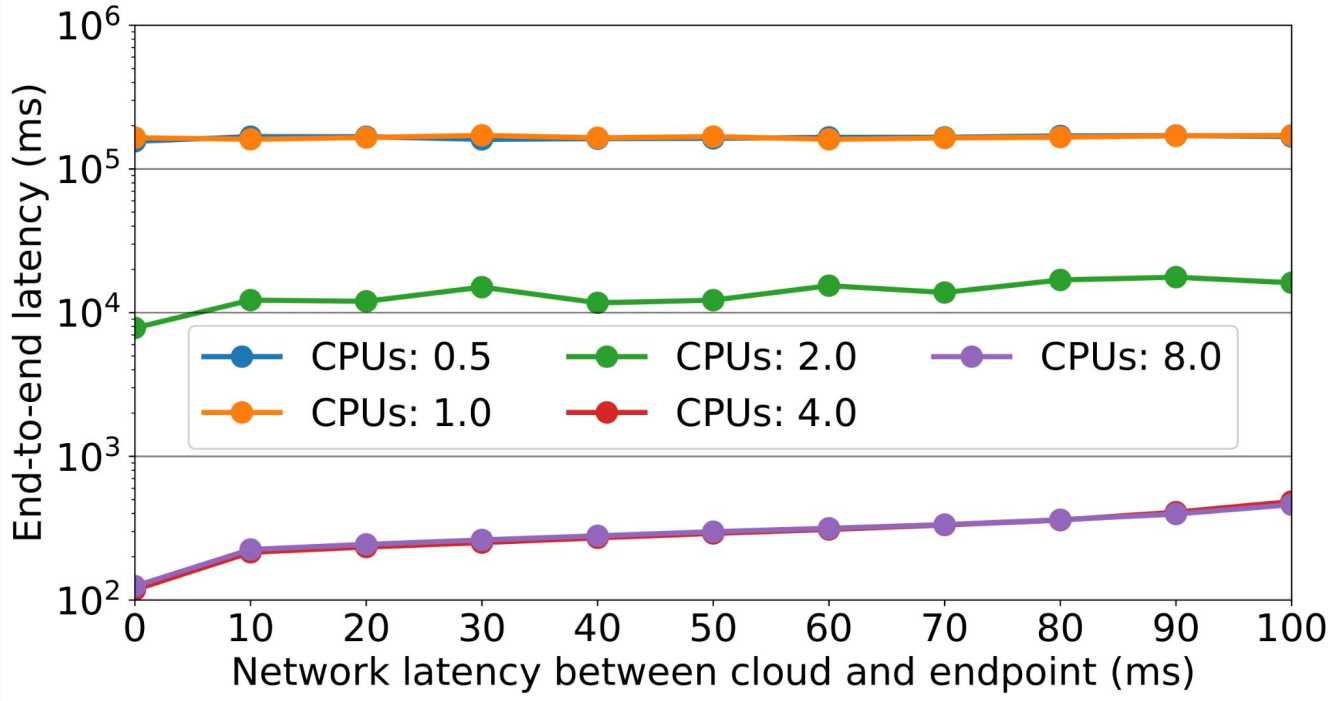
1. EP pre-processing
2. EP → Target
3. Queue
4. Processing
5. Target →EP



Network Latency and CPUs

Discover:

1. Performance trade-off
2. Metrics to focus on



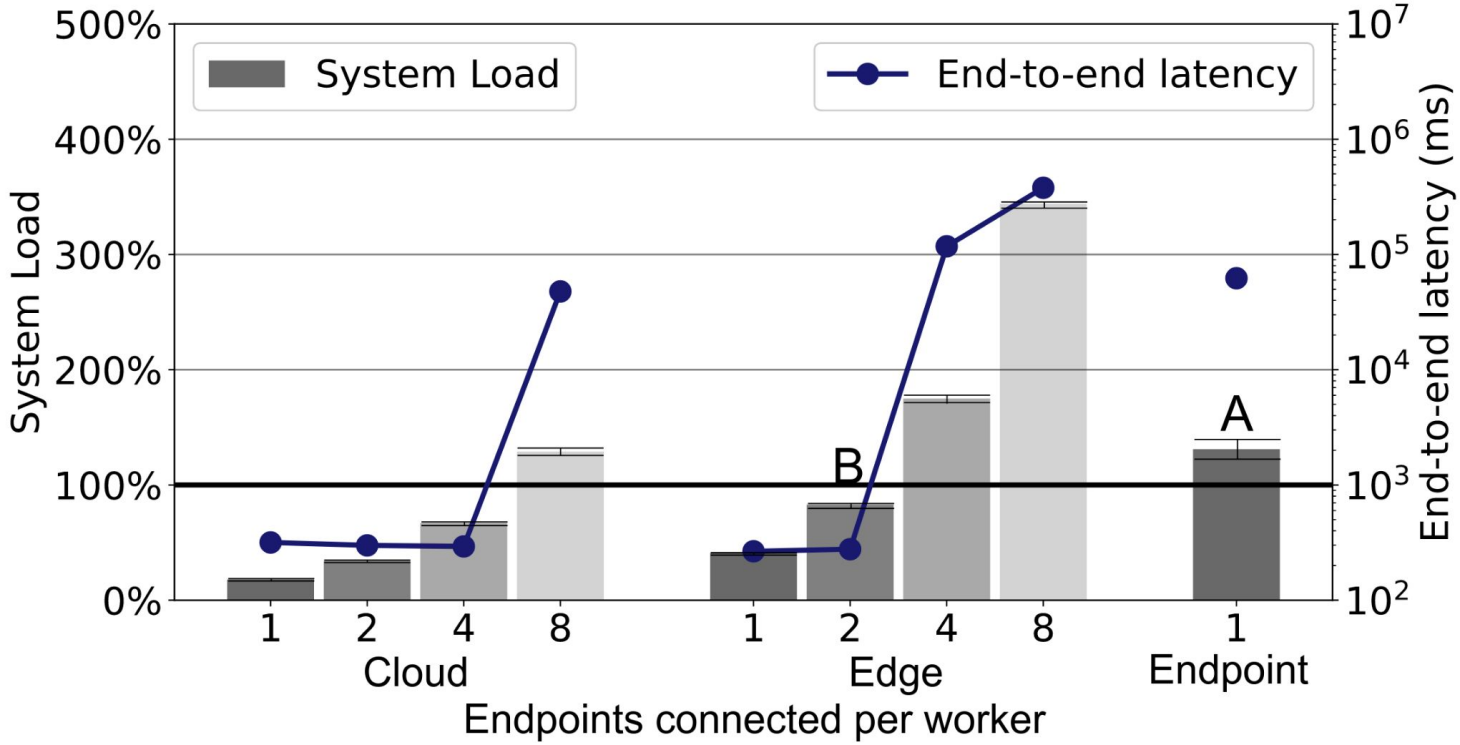
Results difficult to predict, benchmarking required!

Multiple Endpoints per Offload Target

System Load < 100%: Real-time processing

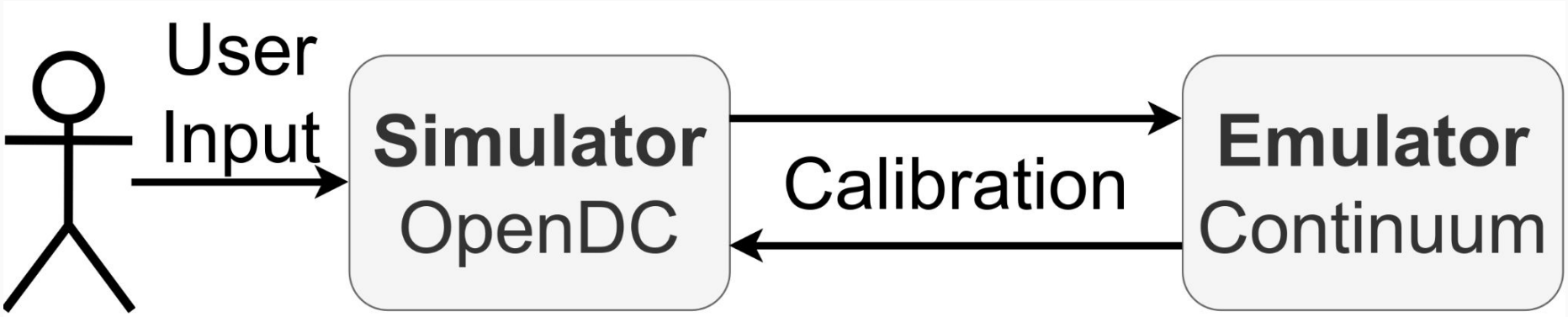
> 100%: Queue starts to form

E2E Latency



Future Work

Simulator → Fast
Emulator → Real-world metrics



More Ongoing and Future Work

1. Kubernetes scalability analysis
2. Energy modeling for resource management
3. Performance analysis of virtualization technologies
4. Scheduling for serverless edge computing
5. Domain-specific language for Continuum

Take-away message

Compute continuum is complex, difficult to navigate

We offer Continuum:

- Deploy *Infrastructure, Software, Benchmark*
- *Accurate, Automated, Extendable, Flexible*

Open Research Objects (ORO)

Research Objects Reviewed (ROR)

https://github.com/atlarge-research/continuum
https://atlarge-research.com/offsense/

This presentation was based of work from

1. **Matthijs Jansen**, Linus Wagner, Animesh Trivedi, and Alexandru Iosup. Continuum: Automate Infrastructure Deployment and Benchmarking in the Compute Continuum (2023). Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE'23). https://atlarge-research.com/pdfs/2023-fastcontinuum-continuum.pdf
2. **Matthijs Jansen**, Auday Al-Dulaimy, Alessandro V. Papadopoulos, Animesh Trivedi, and Alexandru Iosup (2023). The SPEC-RG Reference Architecture for the Compute Continuum. 2023 23th IEEE/ACM International Symposium on Cluster, Cloud, and Internet Computing. https://atlarge-research.com/pdfs/2023-ccgrid-refarch.pdf

Further reading

1. Alexandru Iosup, Alexandru Uta, Laurens Versluis, Georgios Andreadis, Erwin van Eyk, Tim Hegeman, Sacheendra Talluri, Vincent van Beek, and Lucian Toader (2018). Massivizing Computer Systems: a Vision to Understand, Design, and Engineer Computer Ecosystems through and beyond Modern Distributed Systems. CoRR. http://arxiv.org/abs/1802.05465