WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

CLOUD COMPUTING

Orchestration, Containers, and Kubernetes

PAUL TOWNEND

ASSOCIATE PROFESSOR, UMEÅ

YESTERDAY IN SUMMARY

Lot of topics at a fairly high level

Basic Cloud terminology, what are data centers

HDFS (distributed storage)

Hadoop MR (distributed batch processing)

Apache Spark (distributed in-memory processing)

Apache Storm (stream processing)

Edge, Fog, Serverless

TODAY IN SUMMARY

Orchestration

Containers and Kubernetes

Learn how to use the Ericsson Research Data Center

Cloud Economics

Keynote talk from Google

Assignment etc.

RECAP: ELASTICITY AND SCALABILITY

Clouds need to be highly elastic and scalable

Why?

Dynamic allocation of resources

Rapid deployment of applications

This helps us to quickly (and automatically) scale services to meet dynamic workloads

The solution to this has been **virtualisation.** **But how do we manage virtual resources?**

CLOUD ORCHESTRATION

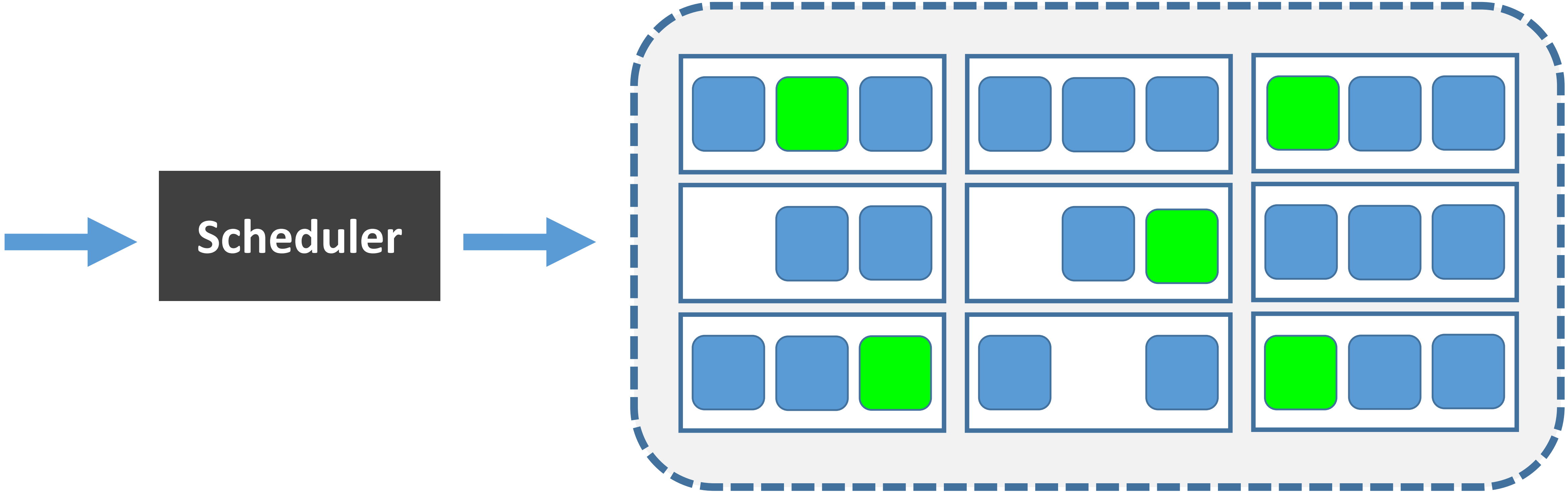
"Cloud orchestration consists in coordinating, at the software and hardware layer, the deployment of a set of virtualized services in order to fulfil operational and quality objectives of end users and Cloud providers"

A. Tosatto, P. Ruiu and A. Attanasio, "Container-Based Orchestration in Cloud: State of the Art and Challenges",9th International Conference on Complex, Intelligent, and Software Intensive Systems, 2015

Resource allocation Fulfil SLAs + enforce limits	Resource optimisation Maximise host resources	Performance Minimise overhead	Persistency / FT Same or different host
Security Minimise exposure	Supervision Monitoring + auto restart	Portability Isolation + heterogeneity	Efficiency Minimise interference etc

RECAP FROM FIRST LECTURE

Schedule virtual workloads in a more effective manner



ORCHESTRATION VS SCHEDULING

Orchestration is a broad term that refers to
container scheduling,
cluster management,
and possibly the provisioning of additional hosts.

J. Ellingwood, "The Docker Ecosystem: Scheduling and Orchestration", https://www.digitalocean.com/community/tutorials/the-docker-ecosystem-scheduling-and-orchestration

Containers in more detail

RECAP FROM FIRST LECTURE

Container 1	Container 2	Container 3
App	App	App
Container engine (Docker, etc.)		
Host operating system		
Physical server		

- "Light"
- Fast to create
- Do not have migration
- Typically short lived and single function

VM 1	VM 2	VM 3
App	App	App
Guest OS	Guest OS	Guest OS
Hypervisor		
Host operating system (if Type 2 Hypervisor)		
Physical server		

- "Heavy"
- Slow to create
- Slow to migrate
- Typically long lived and multi-function

TERMINOLOGY

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another

docker

A container is an isolated, lightweight silo for running an application on the host operating system

Microsoft

CONTAINERS IN MORE DETAIL

Containers don't exist

They are built on multiple kernel features

Namespaces	CGroups
Provide per-process isolation of OS resources. There are seven namespaces, covering different resources `pid, net, ipc, mnt, uts, user, cgroup`	A kernel feature that isolates resource usage, providing resource management and accounting. This controls: `Memory, CPU, Block I/O`

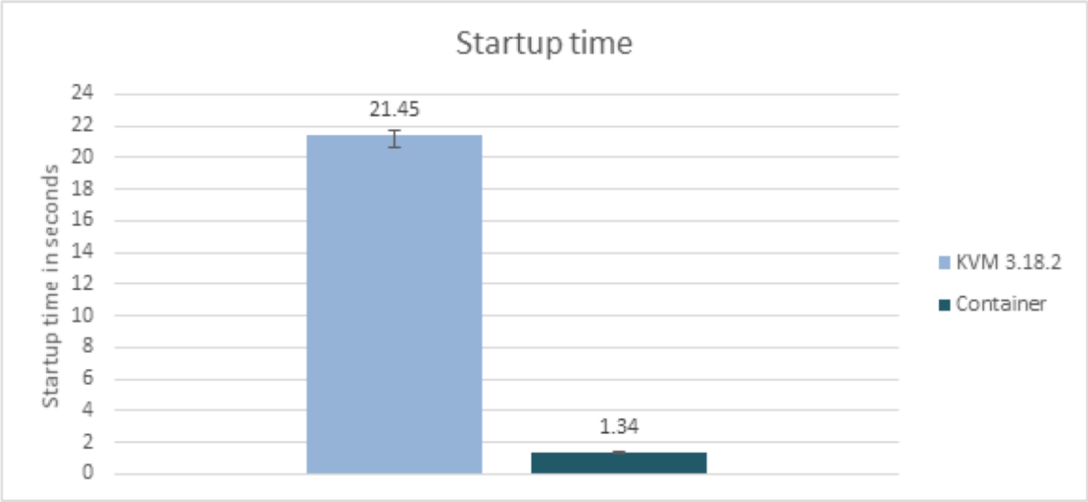
Chroot

Security policies (SELinux, AppArmor, etc.)

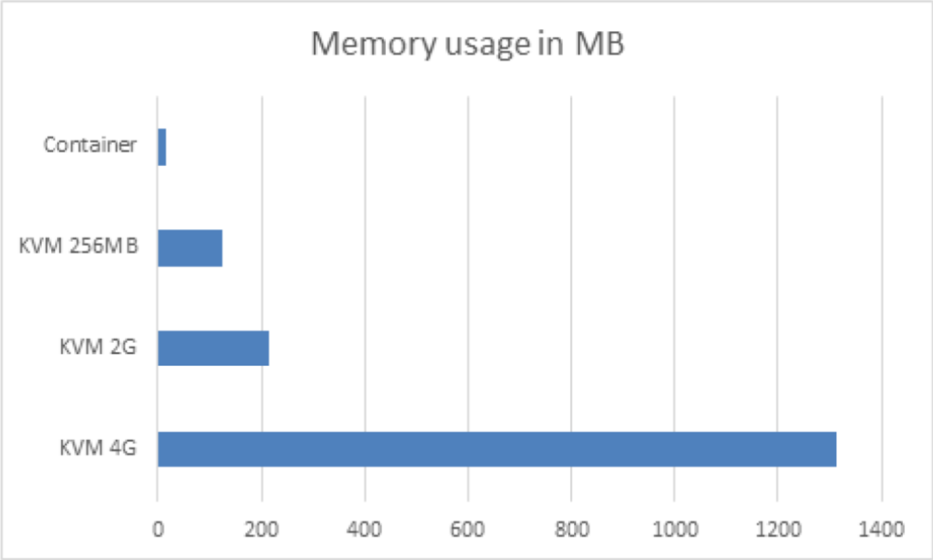
Modification of kernel

CONTAINERS VS VMS

Average Startup Time (Seconds) for a KVM Linux Virtual Machine and a Container Over Five Measurements



The Memory Used by a Container Versus the Memory Used by a KVM Virtual Machine with Varying Memory Sizes



Intel Corporation, "Container and Kernel-Based Virtual Machine (KVM) Virtualization for Network Function Virtualization (NFV)", builders.intel.com, Santa Clara, CA, 2015

CONTAINER TERMINOLOGY

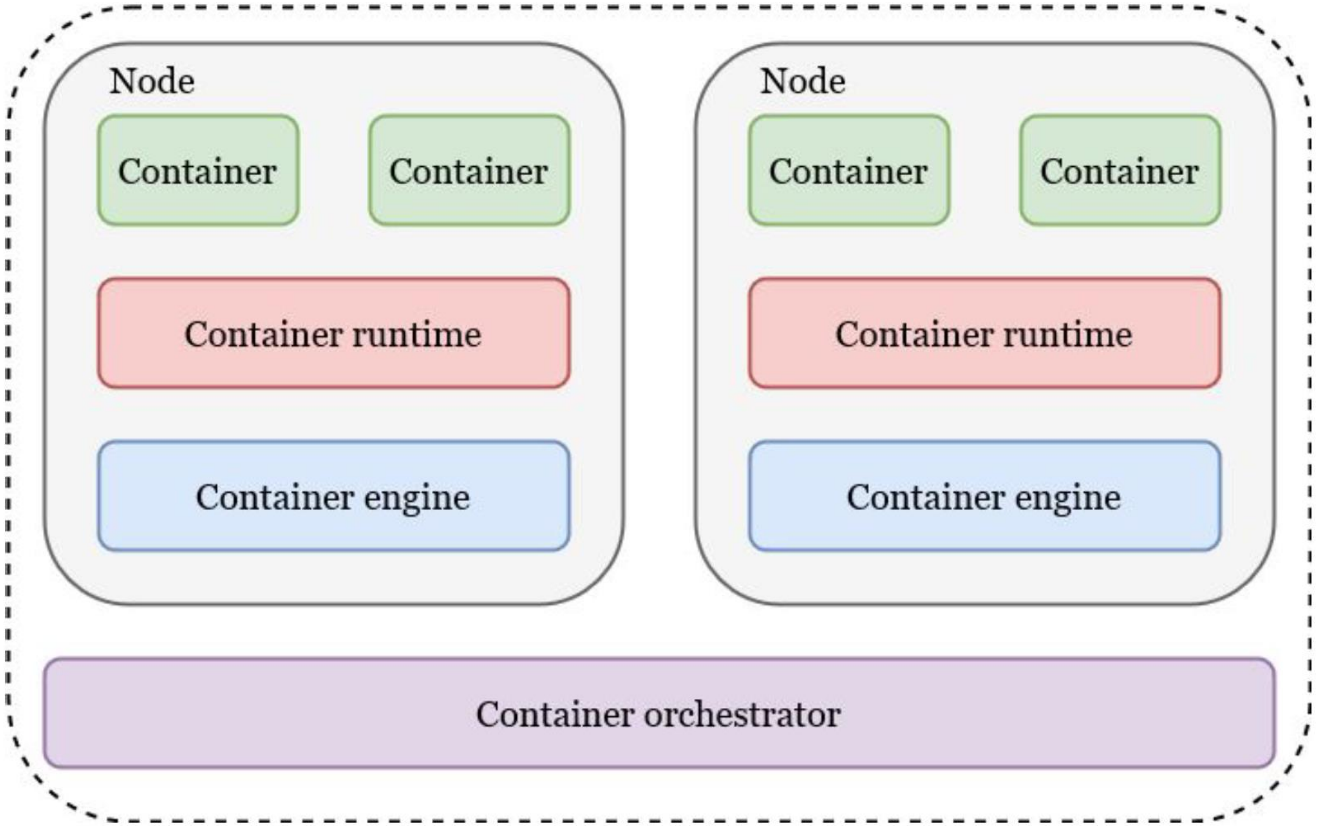
Conflicting definitions and terminology

We will use

Container engine

Container runtime

Container orchestrator



CONTAINER ENGINE

"A container engine is a piece of software that accepts user requests, including command line options, pulls images, and from the end user's perspective, runs the container"

Responsibilities

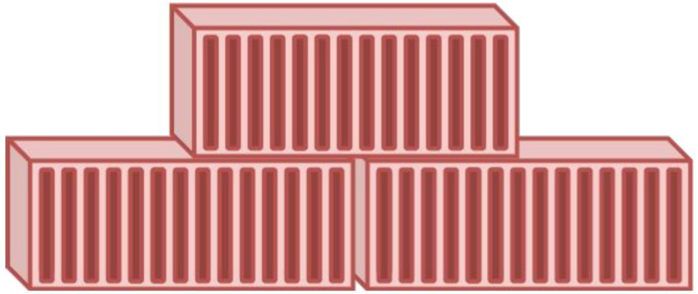
Interface / API (for users and orchestrators)

Images (instructions for creating + pulling from registry server)

Container mount point

Configuration

Calling the Container runtime



CONTAINER RUNTIME

A container runtime [is] a lower level component typically used in a Container Engine but can also be used by hand for testing

Provides higher-level abstraction for creating and running single or multiple containers within single host

Responsibilities

- Configuration / specification of containers
- Setting up Cgroups
- Setting up Linux namespaces
- Setting up Chroot
- Setting up SELinux Policy
- Setting up AppArmor rules etc.

Alternatives

Unmodified kernel

- Libcontainer
- LXC

Modified kernel

- OpenVZ
- Linux-VServer

TERMINOLOGY (3)

When at rest, a container is a file (or set of files) that is saved on disk → This is the **container image** or **container repository** (when collected)

When you start a container, the files are unpacked and sent to the Linux kernel → Remember, this is the job of the **container engine**

The kernel API call typically initiates extra isolation and mounts a copy of the files that were in the **container image** → When running, containers are a standard **linux process**

OPEN CONTAINER INITIATIVE

There are several competing **Container Image** formats, but industry moving forward with a standard

→

The **Open Container Initiative** (aka **OCI**)

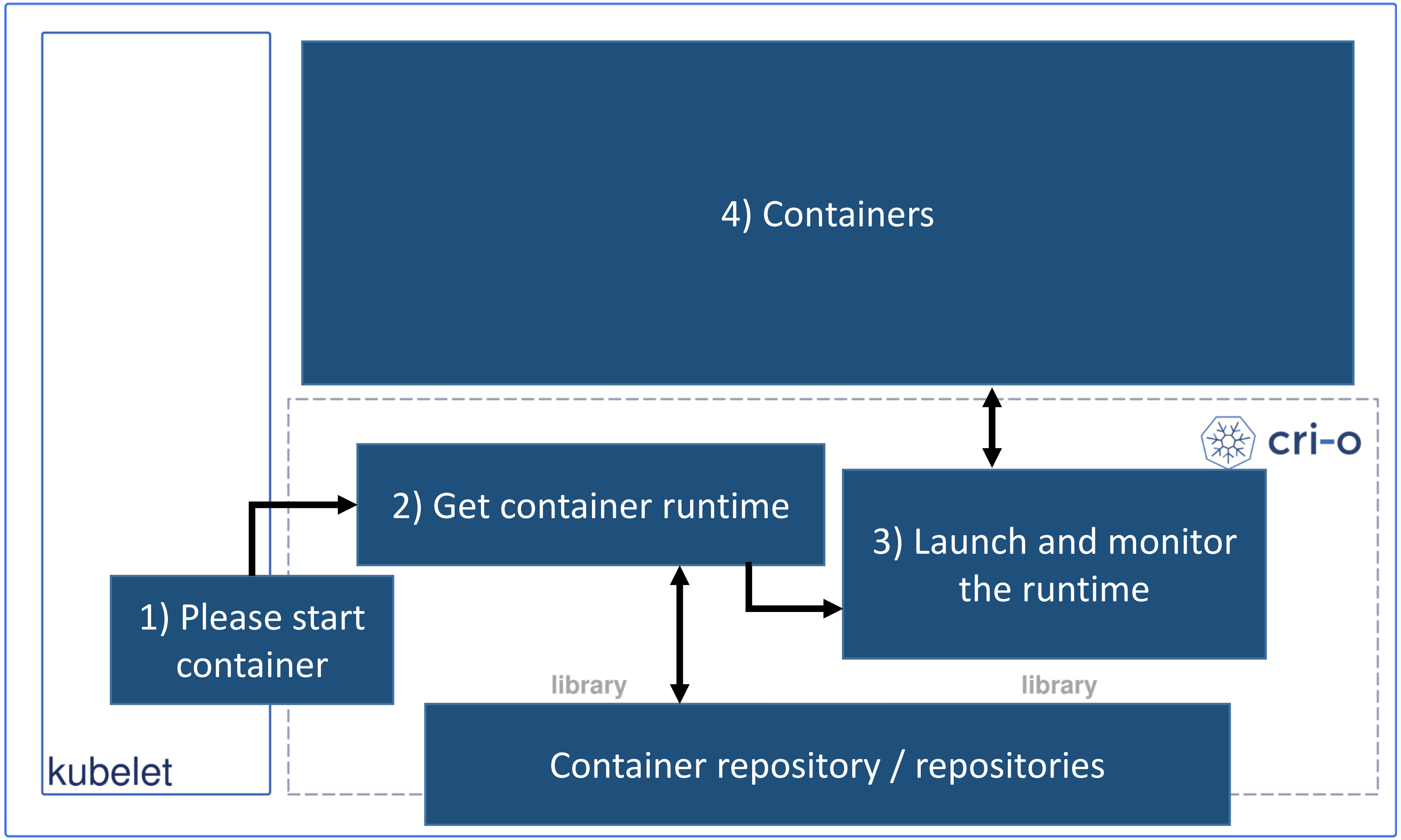
The scope of OCI includes:

The format of the **Container Image**
How **Container Engines** turn an image into a container (i.e. a running process)

CRI-O

A container engine that enables using OCI (Open Container Initiative) compatible runtimes.

“Lightweight alternative to docker” being developed by Red Hat, Intel, SUSE, Hyper, IBM, etc.



Container Orchestration

SO WHAT IS CONTAINER ORCHESTRATION?

Container orchestration platforms can be broadly defined as a system that provides an enterprise-level framework for integrating and managing containers at scale

A. Khan, "Key Characteristics of a Container Orchestration Platform to Enable a Modern Application," in IEEE Cloud Computing, vol. 4, no. 5, September/October 2017

Capabilities include

Cluster state management and scheduling
High availability and fault-tolerance
Security
Networking

Service discovery
Monitoring and governance
Facilitate continuous deployment

CLUSTER STATE MANAGEMENT AND SCHEDULING

Containers can run on multiple virtualised or physical instances - the cluster needs to be kept stable

Flexible scheduling of tasks across a cluster, making backups, garbage collection, file consolidation, index rebuilds.

Also, control mechanisms for algorithms (binning, affinity, etc.)

Reliable state management and repartitioning of data/resources across the cluster

Informing dependent systems of changes, and throttling system tasks/changes

HIGH AVAILABILITY AND FAULT-TOLERANCE

High availability requires the container platform ensure agreed QoS

Elimination of single points of failure (adding redundancy)

Reliable crossover (continue operating even if a component fails)

Detect failures as they occur, and ensure graceful degradation of QoS until failure is resolved

Load balancing is often effective to optimise resource use. Using multiple components (containers) with load balancing may increase reliability and availability through redundancy.

SECURITY

A container orchestrator needs to ensure integrity of deployed services and prevent/ detect intrusions

Container image "sanity"

Trusting images is a critical concern.

Best practice: ensure images are signed and originate in a trusted repository

Isolation

A container with root kernel access and see and access other containers.

Best practice: segment traffic on the network using a service mesh.

Access-control

Platforms should provide both coarse and fine-grained access.

The policy definition point should be a standard identity and access solution.

Run-time container defence and profiling

Containers could go rogue or be misconfigured and use significant resources.

This will unbalance the cluster.

NETWORKING

Orchestration platforms must provide efficient networking at scale

Network isolation is key for container security – but how to balance with network efficiency?

Containers must be allocated ports on the host IP – there is overhead in managing these ports, especially at scale

Dynamic port allocation is a solution, but introduces challenges such as service discovery and managing container level ports.

SERVICE DISCOVERY (1)

To communicate with a container, the network location needs to be known (IP + port)

Containers have dynamically assigned network locations. These also change due to auto-scaling, failures, etc.

How do we discover locations?

A **service registry** is used. It contains the network locations of service instances.

There should be multiple registries, using a replication protocol for consistency.

Examples of service registries:

Netflix Eureka

Etcd

Apache Zookeeper

SERVICE DISCOVERY (2)

Two types of service discovery: client-based and server-based

Client-based

Client is responsible for determining locations and load balancing across them

Server-based

The client makes a request to a service via a load balancer.

The load balancer queries the service registry.

Client-based discovery lets the client make intelligent, application-specific load balancing decisions.

Server-based results in loose coupling – clients do not need to write their own discovery/balancing.

CONTINUOUS DELIVERY + DEPLOYMENT

Process by which code changes are automatically built, tested and prepared/deployed for production

Containers make continuous deployment easier

Gets rid of “works on my computer” syndrome

CDD pipeline can be automated

Tools such as Jenkins for pipeline management and deployment

Container security is critical.

Testing should include security tests as defined by appropriate standards.

MONITORING AND GOVERNANCE

Monitoring (logs, traceroutes, network performance etc.) is very important in Container environments

Two places where monitoring is required

Physical infrastructure

Container activity

Infrastructure level monitoring

Monitor network, security, CPU, memory usage, disk IO, etc.

Container level monitoring

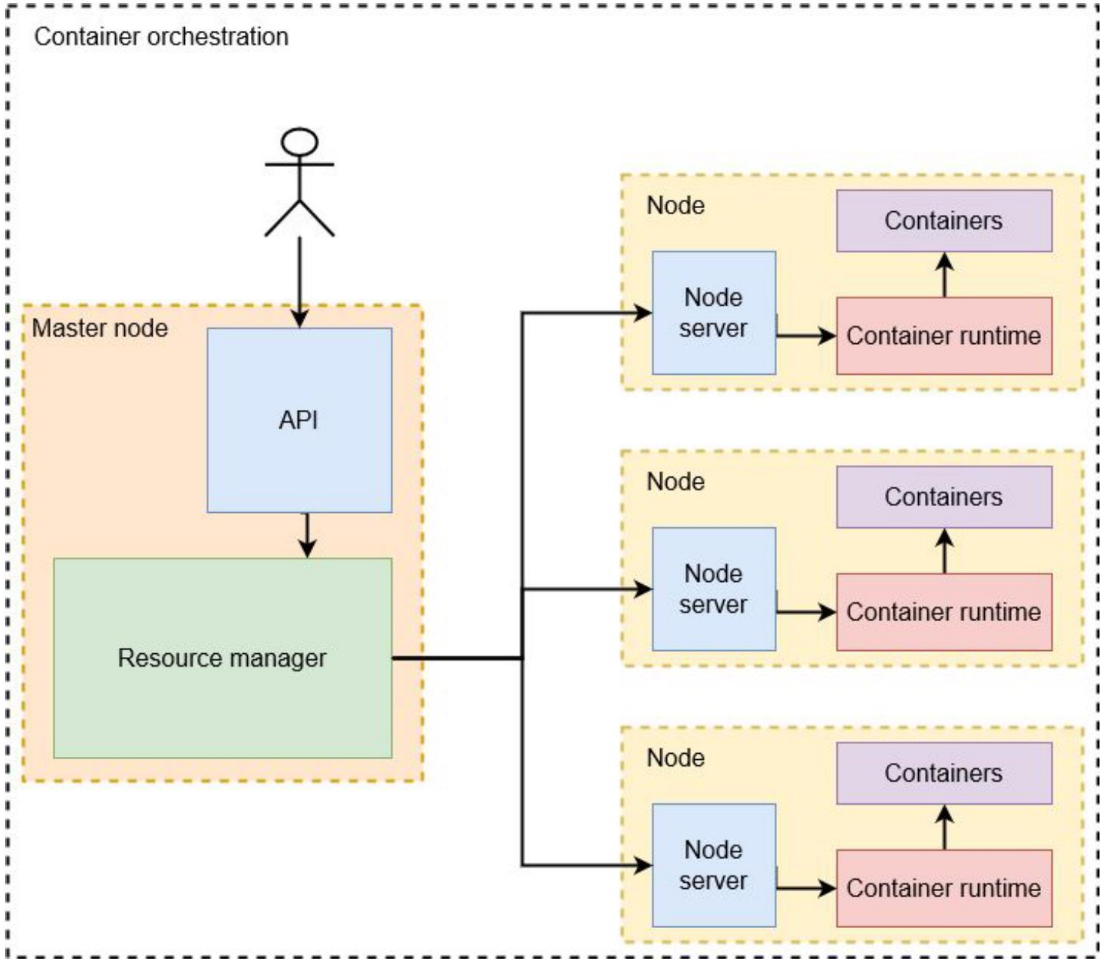
White box tracing of requests, logging events, monitoring performance

CONTAINER ORCHESTRATORS IN SUMMARY

Container orchestrators are crucial for deploying, managing, and monitoring container systems

Container engines deploy container images, running container runtimes

Container orchestrators manage the runtimes and the live system as a whole



POPULAR CONTAINER ORCHESTRATORS

docker SWARM

Nomad

kubernetes

RED HAT® OPENSHIFT

CONTAINER ORCHESTRATORS – CLOUD PROVIDER

Azure Kubernetes Service (AKS)

IBM **Cloud**
Kubernetes Service

Google Kubernetes Engine

Amazon ECS

Kubernetes

WHAT IS KUBERNETES?

By far the leading container orchestration platform in the world

Portable

Extensible

Open-Source

Huge ecosystem

KubeCon 2020

18,700 attendees

7,800 companies

208 media

(this photo is not from 2020!)

KUBERNETES TERMINOLOGY

A **pod** is the smallest unit of computing you can create and manage in k8s

A group of one or more containers with shared storage and network resource, plus a specification for how to run them.

The shared context of a **pod** is a set of Linux namespaces, cgroups, etc.

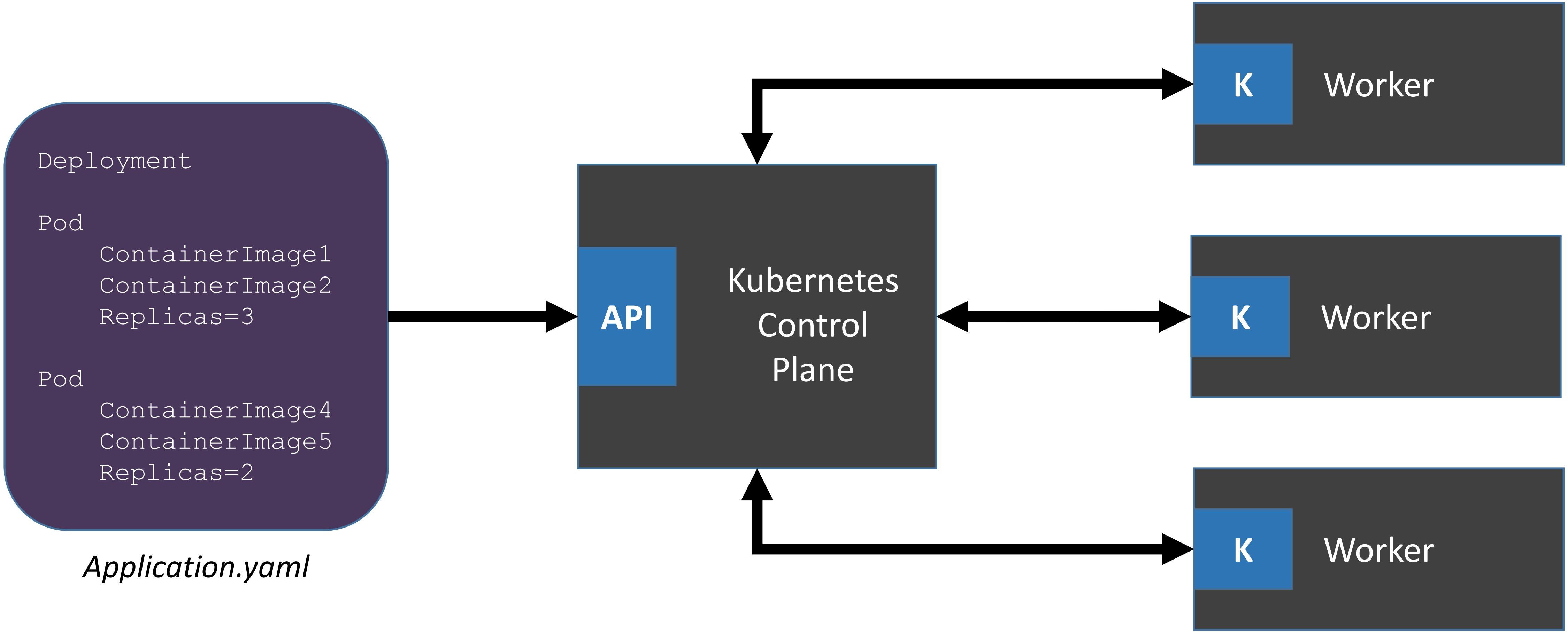
Single container pods

These are the most common pod. K8s manages pods not containers, so consider as a **wrapper**

Pods with multiple containers working together

Encapsulate an application composed of multiple co-located and tightly coupled containers. These form a single unit of service.

KUBERNETES AT A HIGH LEVEL



Deployments within Kubernetes are declarative

KUBERNETES CONTROL PLANE AND NODES

Workers

Provides the k8s runtime environment

Kubelet
An agent that makes sure containers are running in a Pod. Takes PodSpecs and ensures the containers described are running and healthy.

Kube-proxy
A network proxy maintaining network rules. Uses OS packet filtering if there is one, otherwise forwards traffic itself.

Control plane

Components that runs the controller processes:

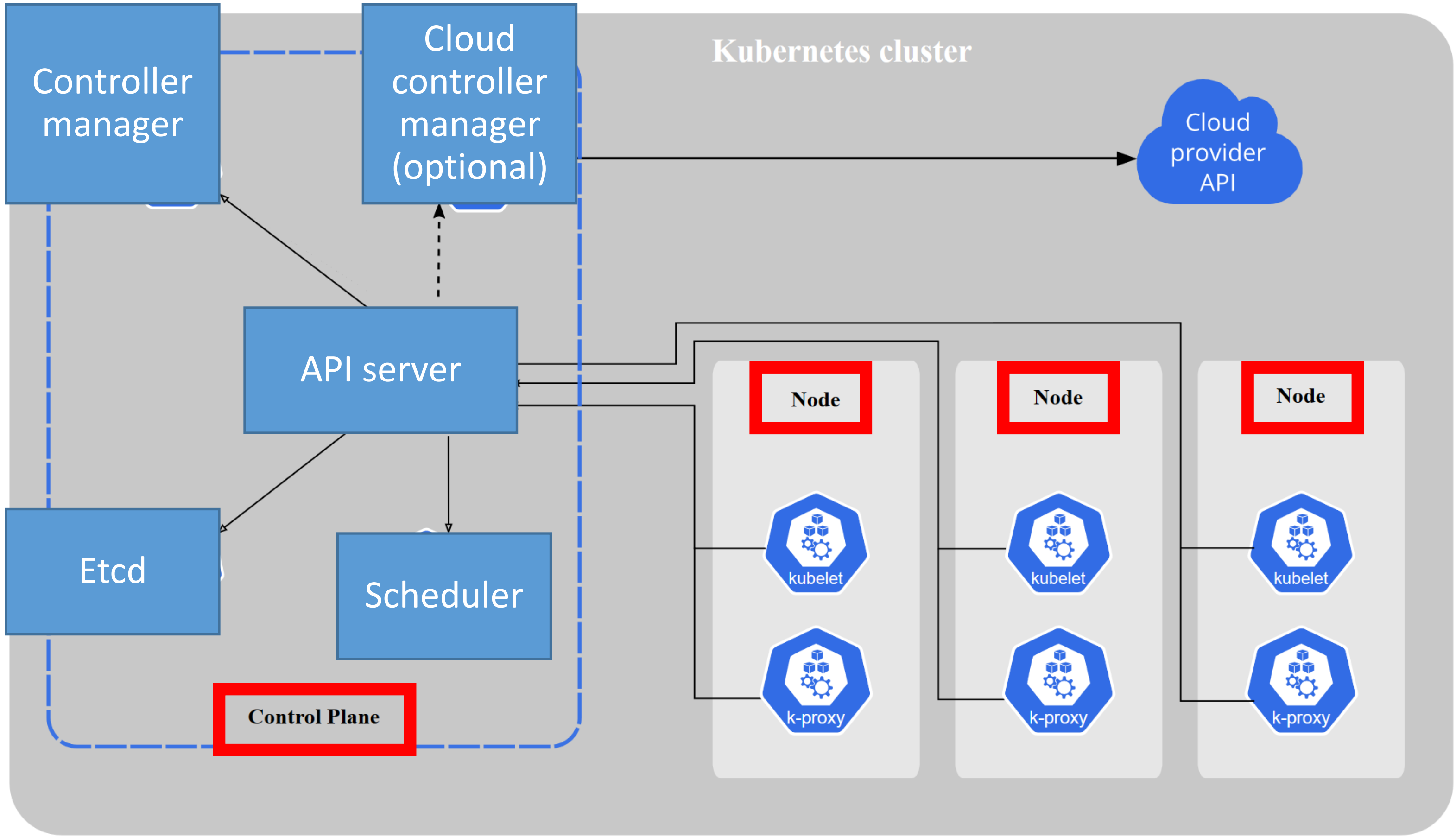
Node controller: Responsible for noticing and responding when nodes go down

Job controller: Watches for job objects and creates Pods to run them

Endpoints controller: Joins services and pods

Service account and Token controllers: Creates default accounts, API access tokens, etc.

KUBERNETES (HIGH-LEVEL) ARCHITECTURE



- API server
- Etcd
- Controller manager
- Cloud controller manager
- Scheduler

KUBERNETES CONTROL PLANE COMPONENTS

API Server

Exposes the k8s API – front end of the control plane. Validates and configures data for objects, including pods, services, etc.

Cloud Control Manager

Links your cluster to your cloud provider's API. If you're running on-prem, this won't be used. You can run multiple instances.

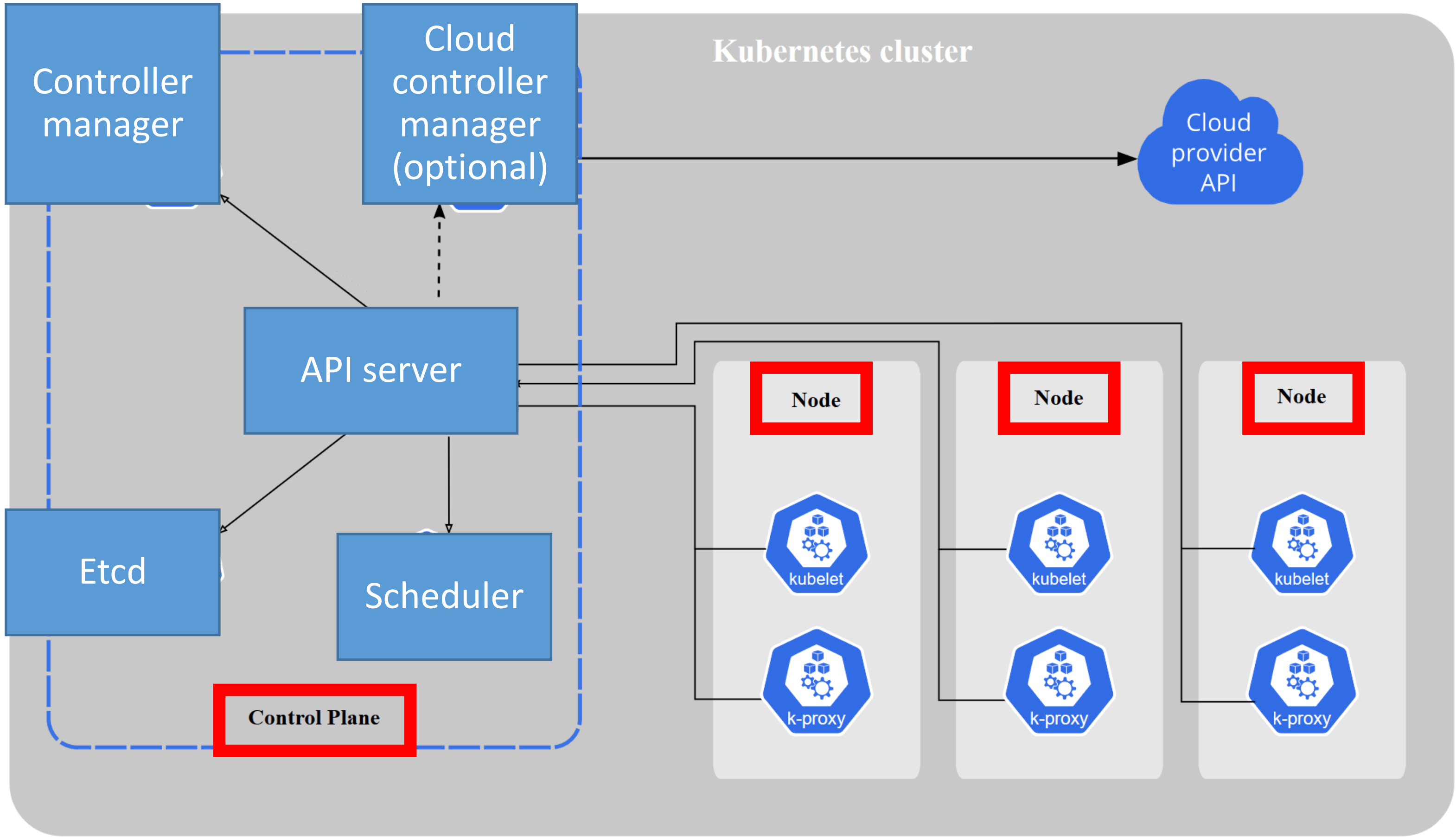
Etcd

Consistent and highly-available key/value store. This is used as backing store for all cluster data.

Control Manager

Runs all the controllers; node, job, endpoints, service account and token controllers.

RECAP: KUBERNETES (HIGH-LEVEL) ARCHITECTURE

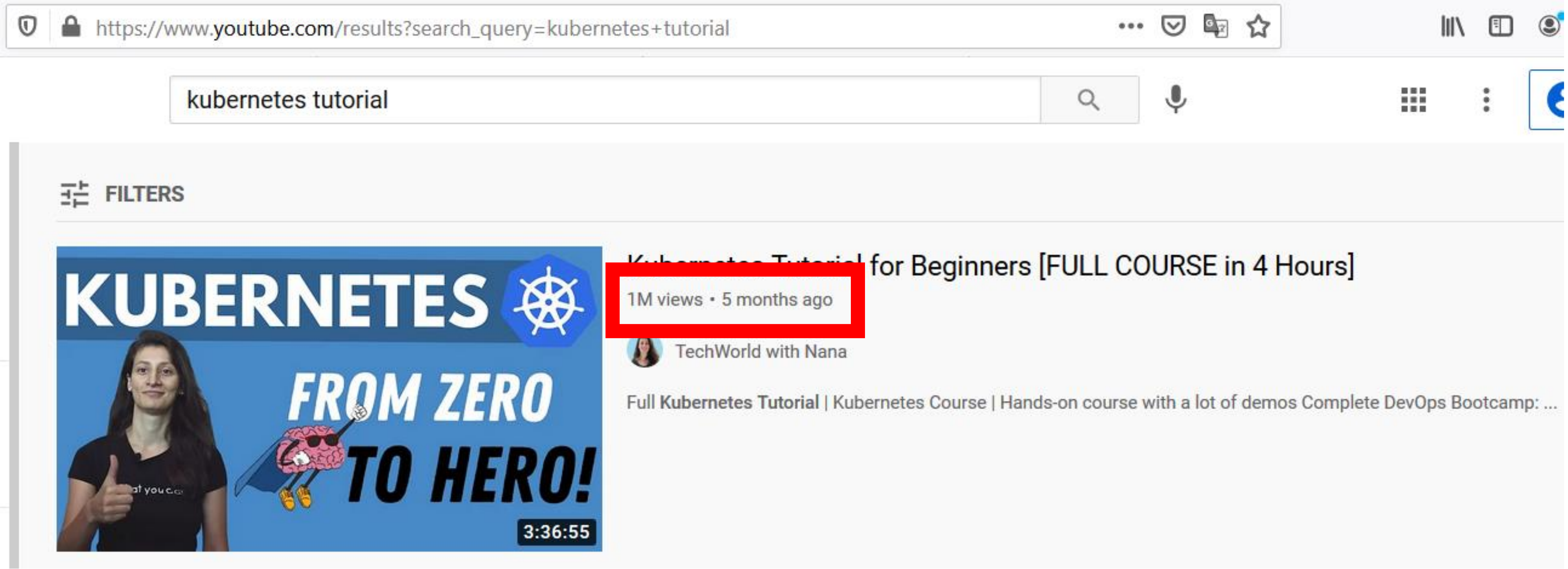


In production, the control plane usually runs across multiple machines.

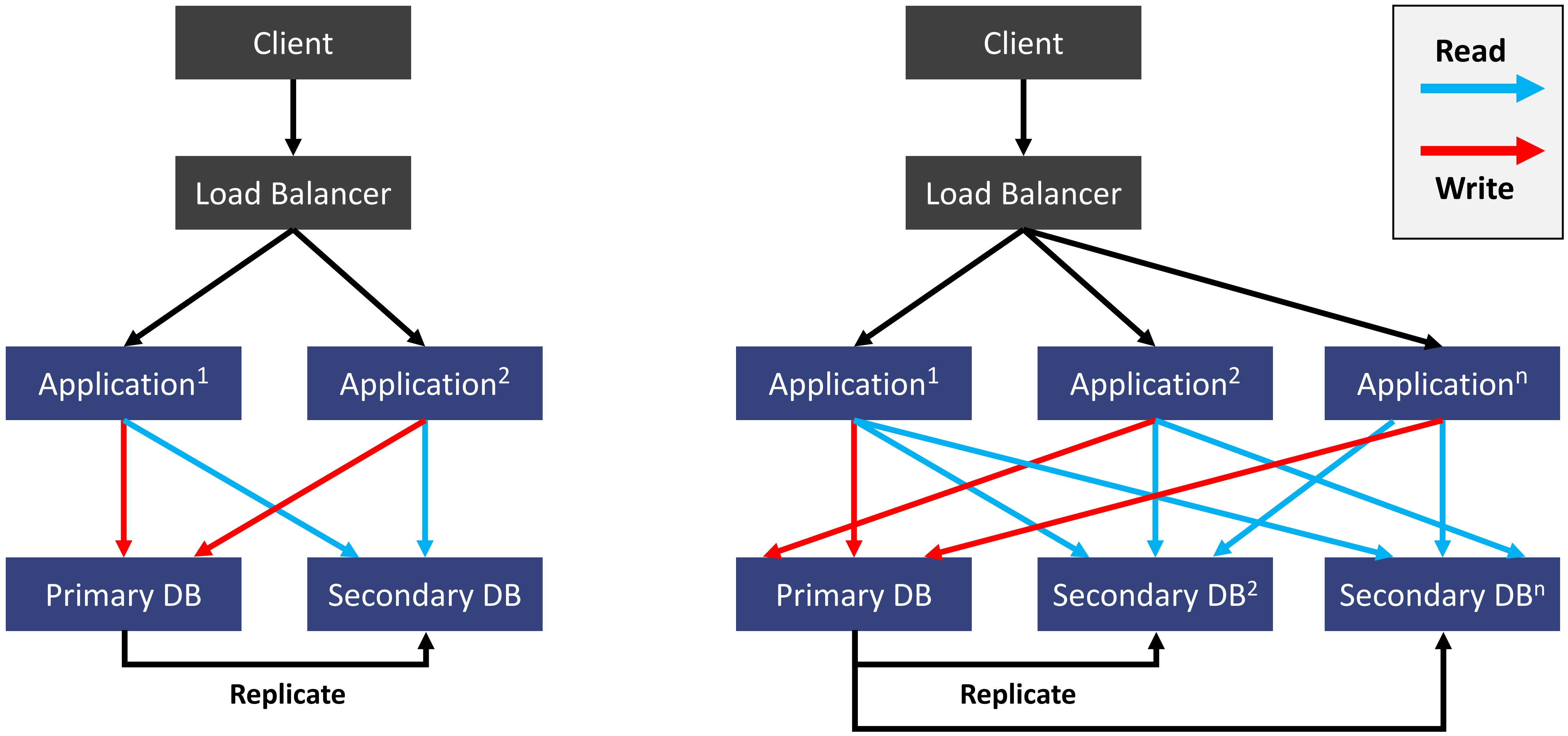
Nodes are run on multiple machines for fault-tolerance and availability (and load balancing)

Many add-ons are possible

IT'S INCREDIBLY "FASHIONABLE" RIGHT NOW



RECAP: AUTO-SCALING



Container scheduling for efficient data centers

Case study

A CASE STUDY

Commercial and Research Data Center

DCD Best Data Center Initiative 2017

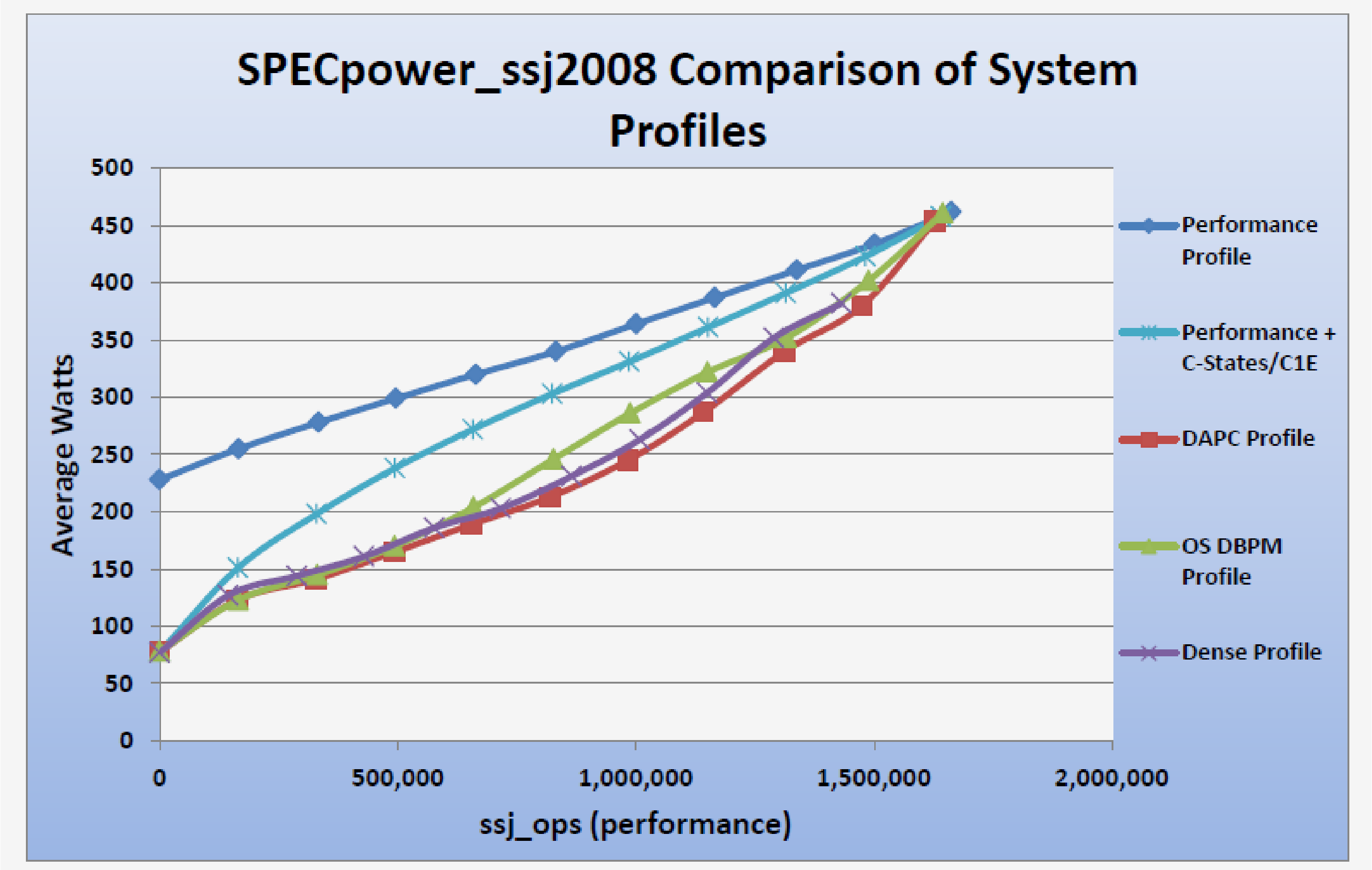
IEEE
Scale Award 2017

Building 2000+ Node Container Facility

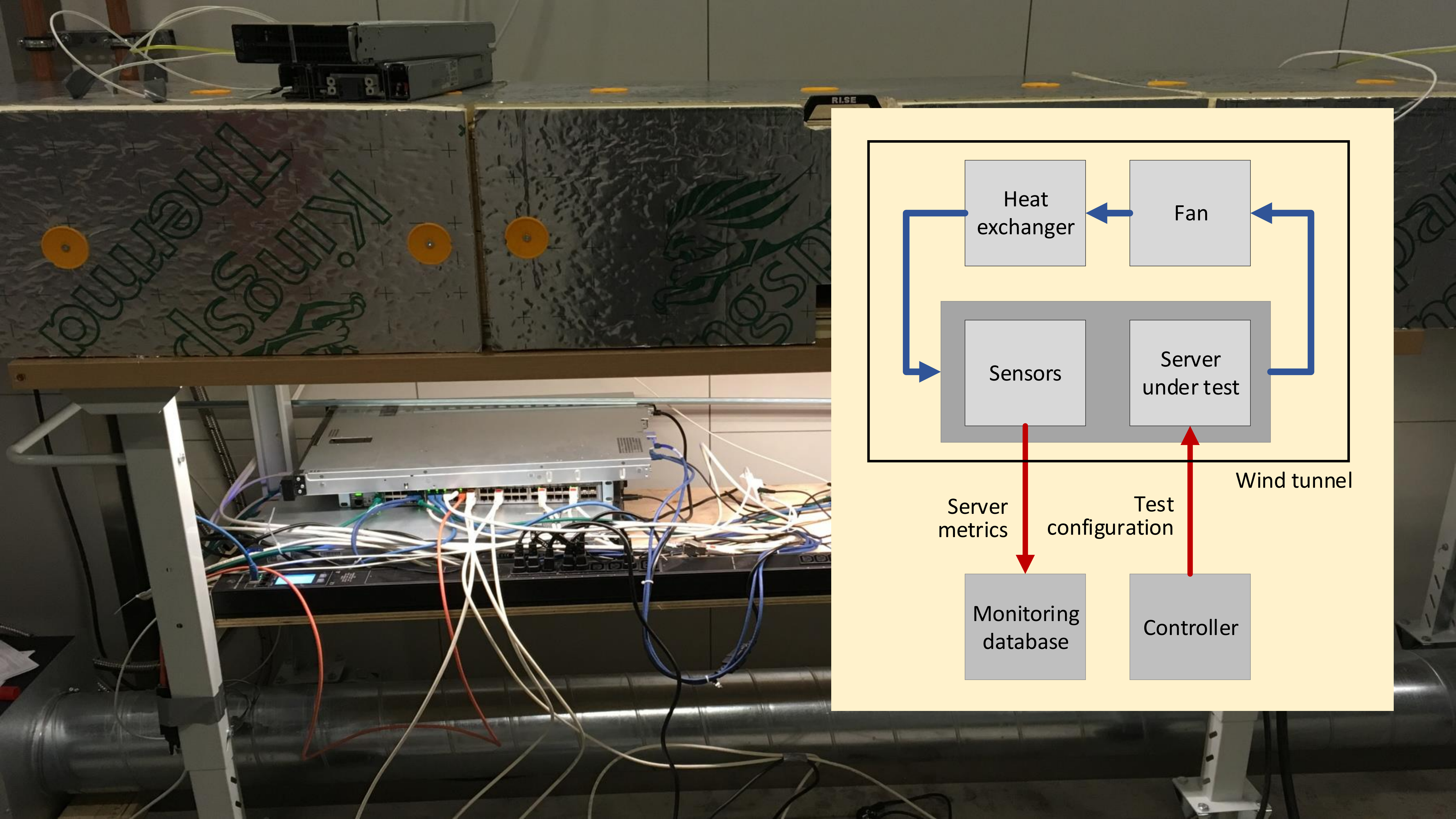
RISE DELL R430

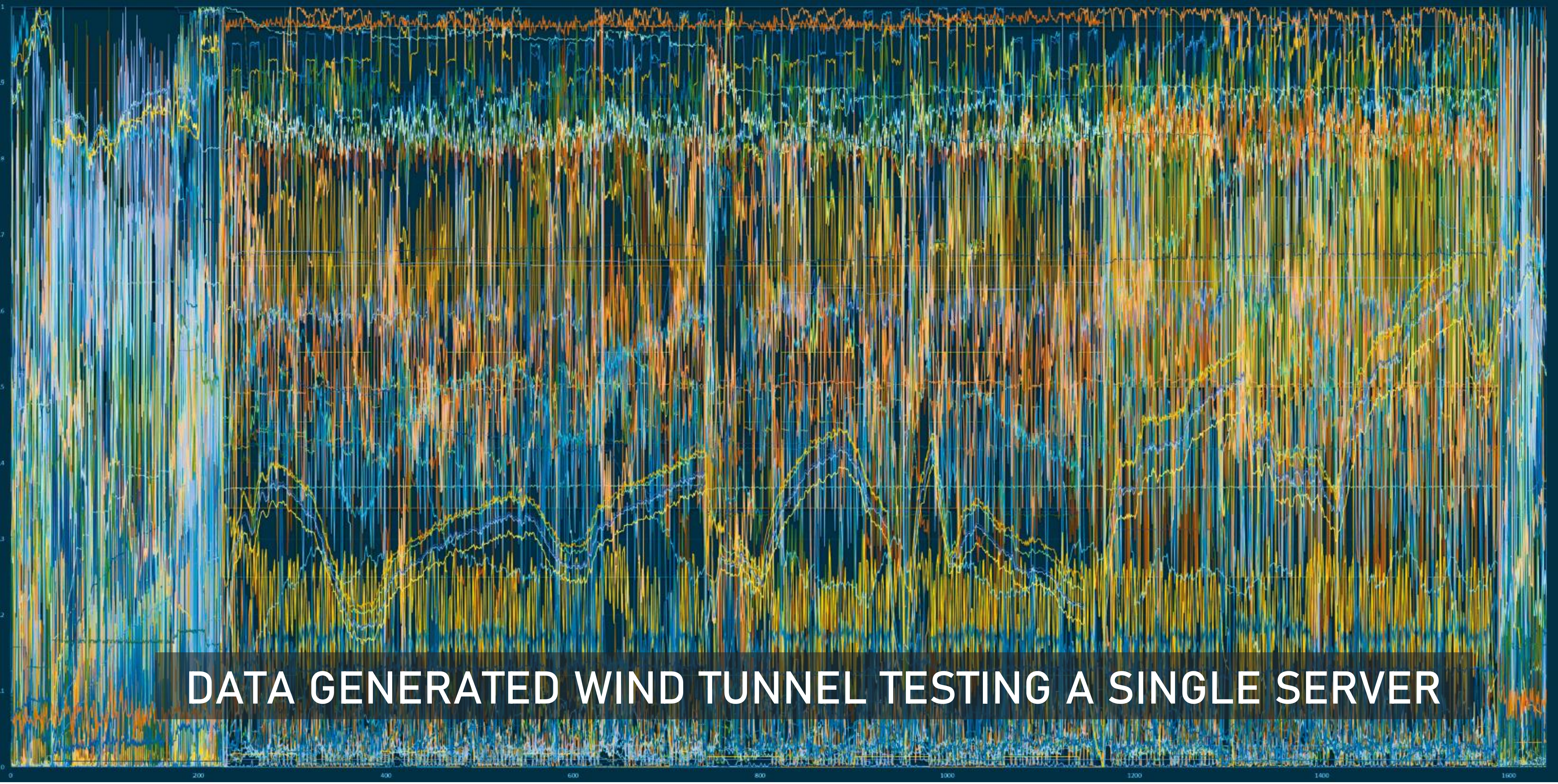
"Performance" BIOS	Uses large amounts of power even while idle Designed to maximise computing performance
"Efficiency" BIOS	Uses low power when idle, scaling up with load Designed to maximise power efficiency

SPECPOWER BENCHMARKS



WIND TUNNEL BASED SERVER MODELLING

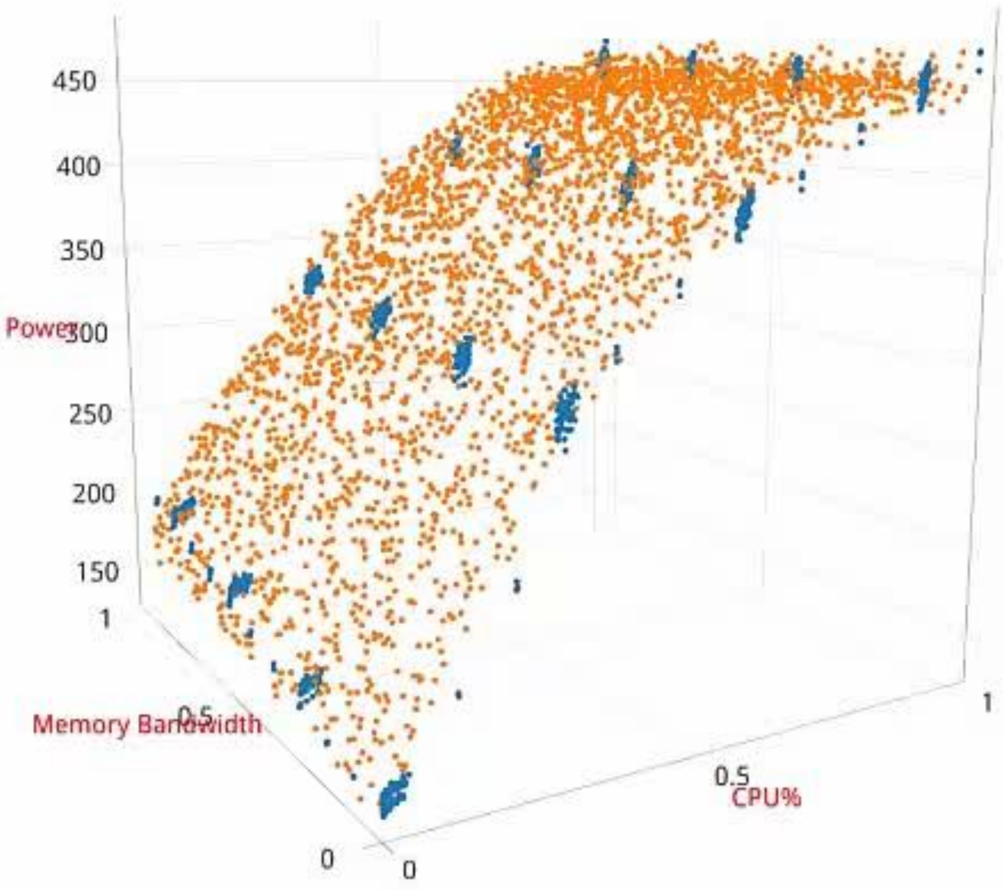

Heat exchanger
Fan
Sensors
Server under test
Wind tunnel
Server metrics
Test configuration
Monitoring database
Controller



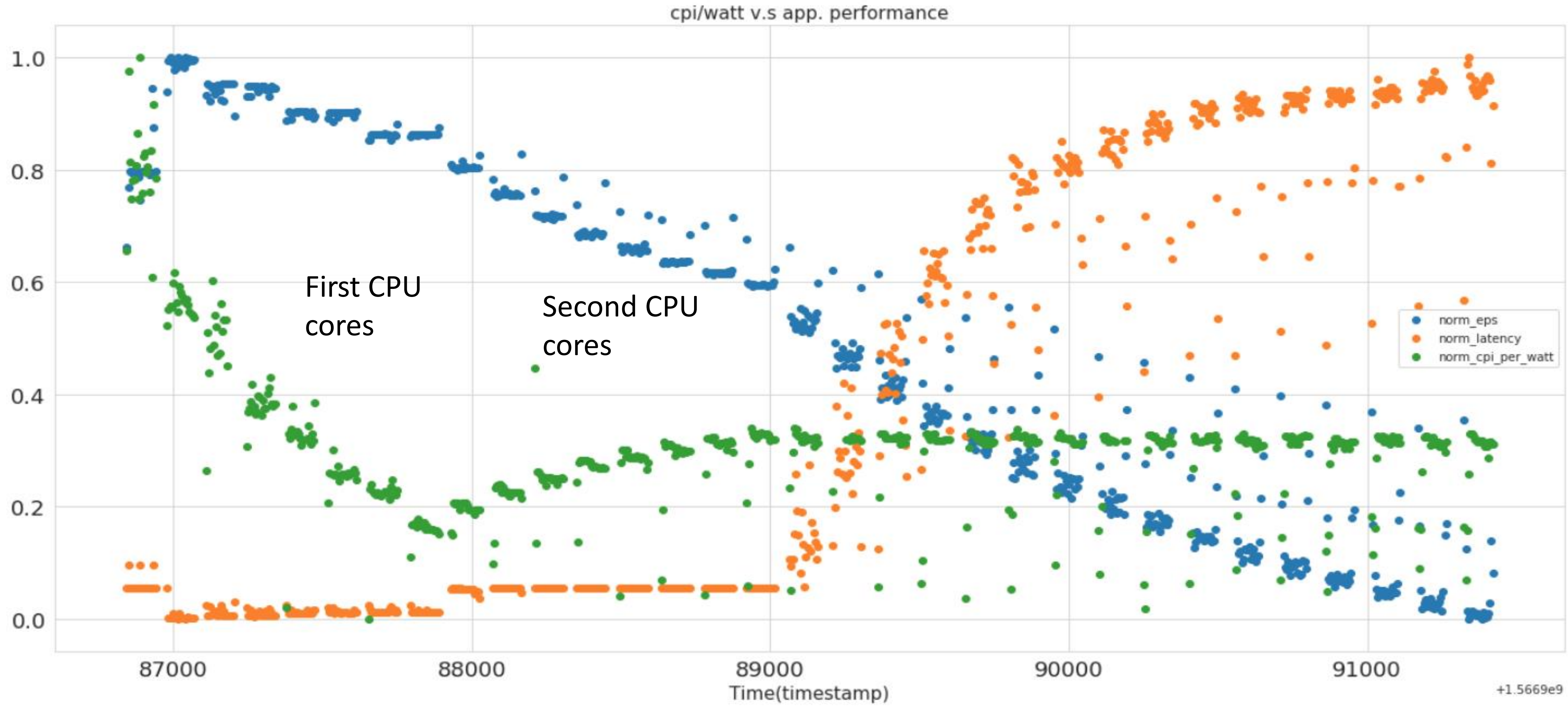
DATA GENERATED WIND TUNNEL TESTING A SINGLE SERVER

POWER MODEL AS FACTOR OF 2 CO-EFFICIENTS

Actual readings (in blue)
Predicted values (orange)



CPI / WATT VS APPLICATION PERFORMANCE SINGLE NODE



EXPERIMENTAL WORKFLOW

Set the BIOS on RISE Pod 2 machines

Create 215 node Kubernetes cluster

Deploy Edgetic scheduler

Submit tests to Workload Generator

Deploy replayable workloads

"Fill up" cluster to a given level of utilisation

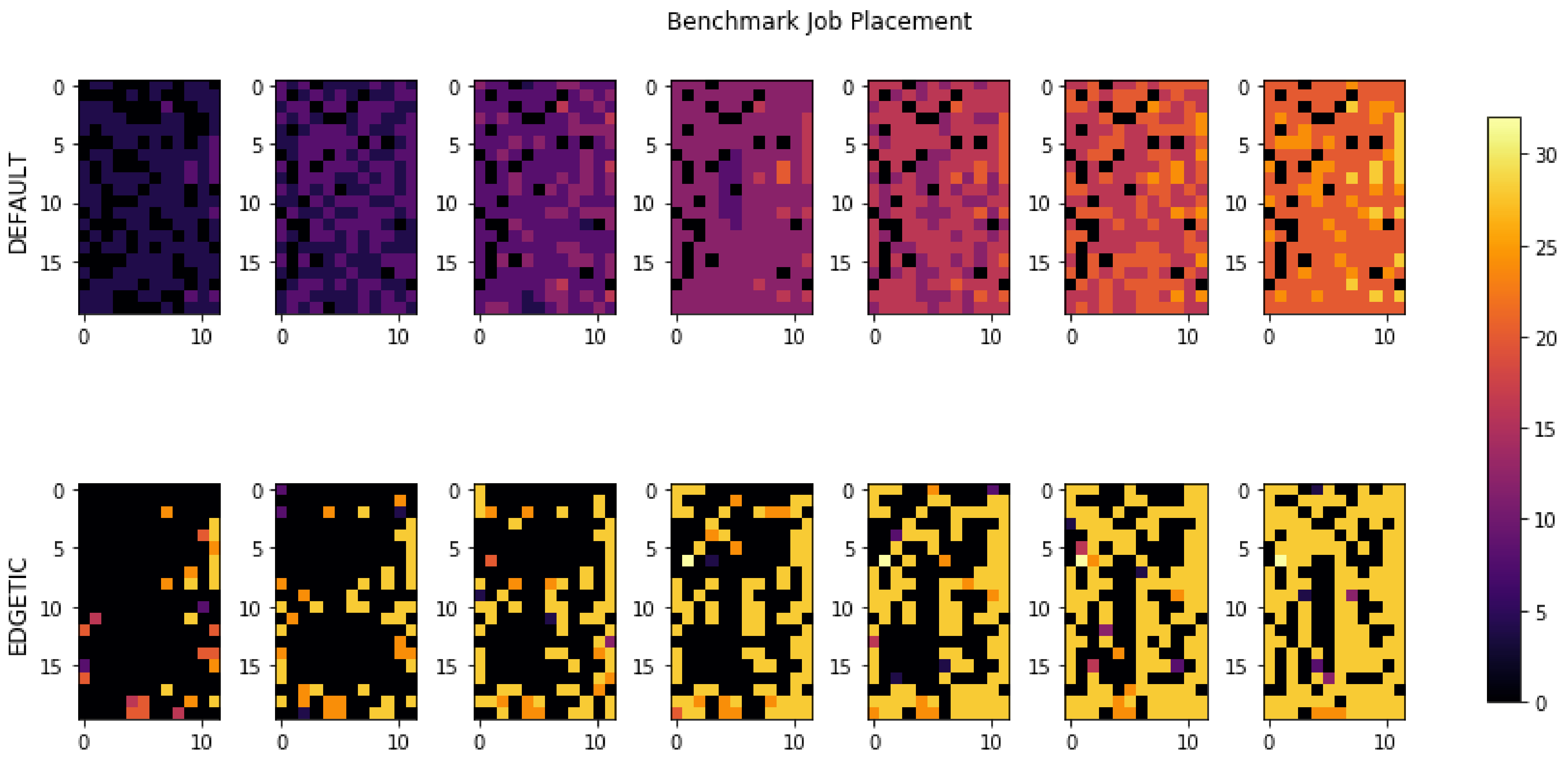
Measure power consumption

Measure performance of each task

Record results

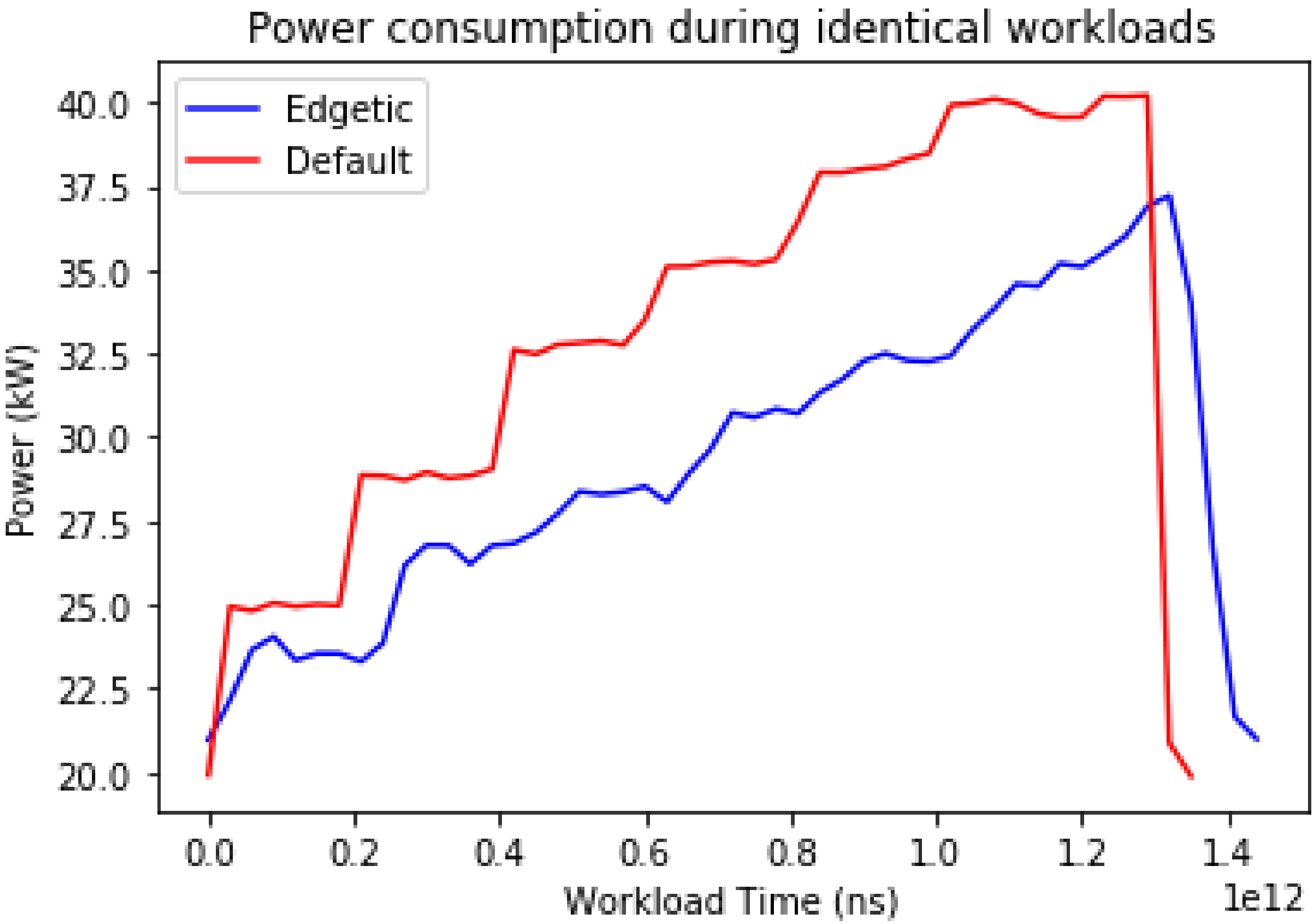
Let cluster cool then move to next test

UTILISATION AT DIFFERENT TIME PERIODS



WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

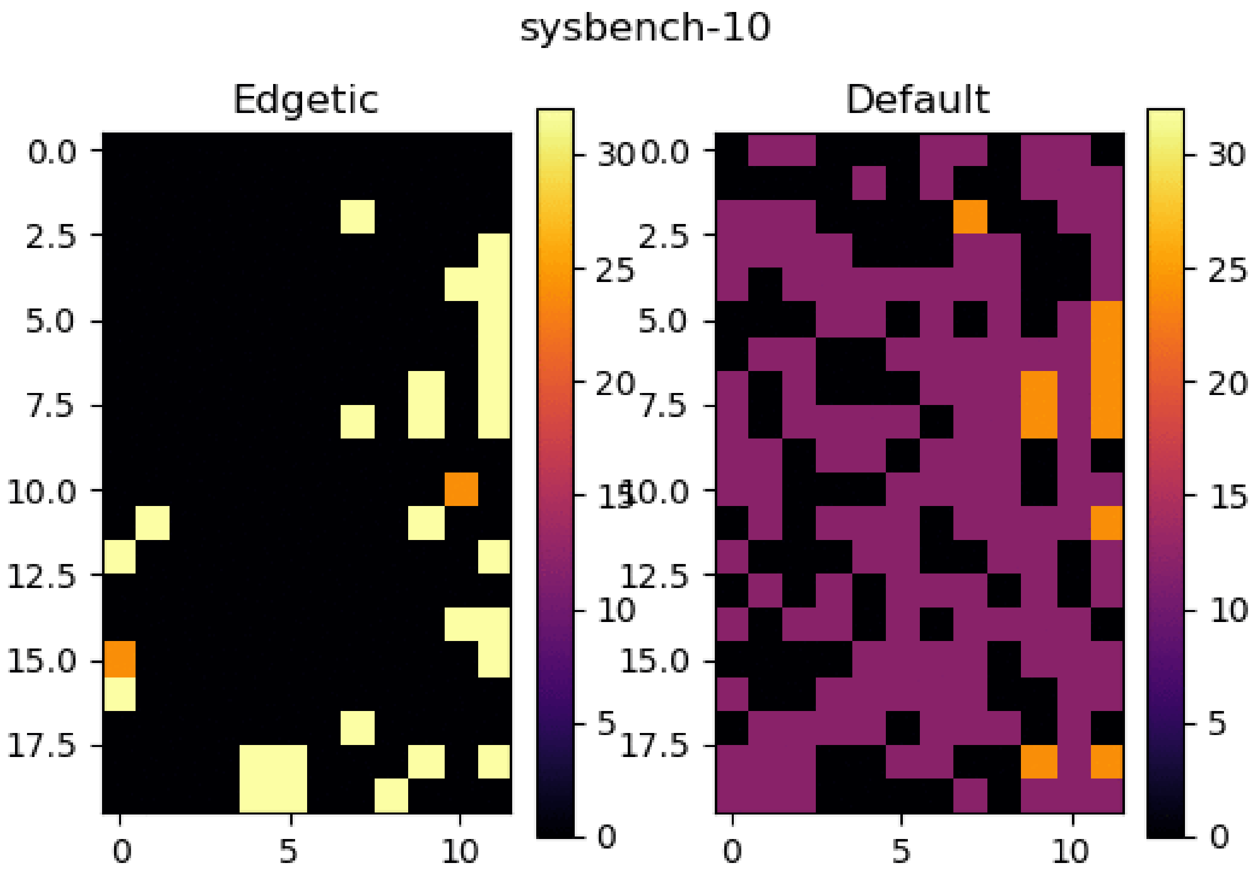
OUR SCHEDULER VS DEFAULT KUBERNETES



HOLISTIC SCHEDULER TESTING AT RISE

215 nodes
OCP Hardware
Variety of workloads
Kubernetes containers
No prior workload knowledge

10-20% power savings
12 terabytes of telemetry data



Overhead of Edgetic Scheduler: 10ms per incoming workload

FINAL THOUGHTS

Cloud orchestration is a powerful concept

Open Container Initiative are standardising container technology

Container orchestration has a lot of considerations

Kubernetes is dominating the container world

We looked at how we can use intelligent orchestration to reduce power consumption

What are you guys doing with k8s?