Methods of Cloud Computing

Pre-Exam Q&A

Complex and Distributed Systems
Faculty IV
Technische Universität Berlin

Operating Systems and Middleware
Hasso-Plattner-Institut
Universität Potsdam

Exam (1/2)

- Portfolio examination
  - Assignments for 40 points, test for 60 points
  - Grading scale: 1

- Written test
  - February 20, 9am, H 0104
  - 60 + 15 minutes
  - Questions in English, answers can be English or German

Exam (2/2)

- We will try to provide the test results quickly
- Chance to take a look at your graded exams on March 20, 4:00pm-5:30pm, H 3008, FCFS
- No second test date: oral examination instead, for everyone…
  - …unable to take the test on February 20 (due to illness)
  - or failed after the results of the written test are in

What the Lecture Discussed

- Goal: Understand what Cloud Computing entails on a technical level
  - Understanding the different levels of abstractions
  - Understanding the implications of resource sharing
  - Learning to take advantage of Cloud platforms
- Content
  - Virtual Resources
  - Managing Cloud Resources
  - Scalable and Fault-Tolerant Applications
  - Data-Intensive Applications
  - Platforms
  - Federations, Edge, IoT

Methods of Cloud Computing

Chapter 1: Introduction

Complex and Distributed Systems
Faculty IV
Technische Universität Berlin

Operating Systems and Middleware
Hasso-Plattner-Institut
Universität Potsdam

Background

Cluster

- Many computers in one Room

Grid

- Loosely coupled computers or clusters all over the world

Cloud

- IT resources as a utility

Edge

- Extending the Cloud to the edge of the network

Cluster

- Mostly homogeneous compute resources and software stacks
- Interconnected by a low-latency and high-bandwidth network
- Goal: improving availability and price/performance

- Examples
  - Analytics cluster at Facebook

Grid

- Heterogeneous compute resources
- Connected via a slow network (i.e. the internet)
- Heterogeneous software stacks unified by a middleware

- Examples
  - PanDA (CERN)
  - BOINC

NIST Definition of Cloud Computing

- NIST: National Institute of Standards and Technology
  - Agency of U.S. Department of Commerce
  - Responsible for standardization processes

- Definition of Cloud Computing according to NIST[4]:

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.
>
> This cloud model is composed of **five essential characteristics**, **three service models**, and **four deployment models**."

Five Characteristics of Cloud Computing (NIST)

- On-demand self-service
  - No human interaction required for resource provisioning
- Broad network access
  - Accessible over network with standard mechanisms
- Resource pooling
  - Pooled resources dynamically shared among several consumers, location independence
- Rapid elasticity
  - Capabilities can be provisioned/released on demand
- Measured service
  - Resource usage is monitored, controlled, and reported

Three Service Models of Cloud Computing (NIST)

- Software as a Service (SaaS)
  - Provider‘s application runs on cloud infrastructure
  - Consumer can access application over the network
  - Consumer does not control/manage underlying infrastructure
- Platform as a Service (PaaS)
  - Consumer can deploy custom application onto cloud infrastructure using programming languages, libraries, services and tools supported by provider
  - Consumer does not control/manage underlying infrastructure
- Infrastructure as a Service (IaaS)
  - Provider provisions processing, storage, network resources to consumer
  - Consumer does not control/manage underlying infrastructure but has control over operating systems, storage, and deployed applications

Four Deployment Models of Cloud Computing (NIST)

	Private Cloud	Community Cloud	Public Cloud	Hybrid Cloud
User of the cloud infrastructure?	Single organization	Organizations with shared concerns	Open for the general public	Composition of private/community/public cloud: • Remain distinct entities • Bound together by standard mechanisms • Goal: Enable data /application portability
Owner of the cloud infrastructure?	Organization, third party, combination thereof	Organizations, third party, combination thereof	Business, academic, government organization, combination thereof	
Location of the cloud infrastructure?	On premise, off premise	On premise, off premise	On premise of cloud provider	

Methods of Cloud Computing

Chapter 2: Virtual Resources

Complex and Distributed Systems
Faculty IV
Technische Universität Berlin

Operating Systems and Middleware
Hasso-Plattner-Institut
Universität Potsdam

Overview

- Virtual Resources and Infrastructure-as-a-Service
- Hardware Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
  - Case Study: Amazon EC2
- OS-Level Virtualization
  - Linux Containerization
  - LXC Containers and Docker
  - Comparison to Virtual Machines

Challenges for IaaS Provider

- Rapid provisioning
  - Resources must be available to the consumer quickly
  - No human interaction during provisioning
- Elasticity
  - Create illusion of infinite resources
  - Yet, manage data center in a cost-efficient manner
- Isolation of different consumers
  - Users must not interfere with each other
- Performance
  - Maintain good performance despite other challenges

What is Virtualization?

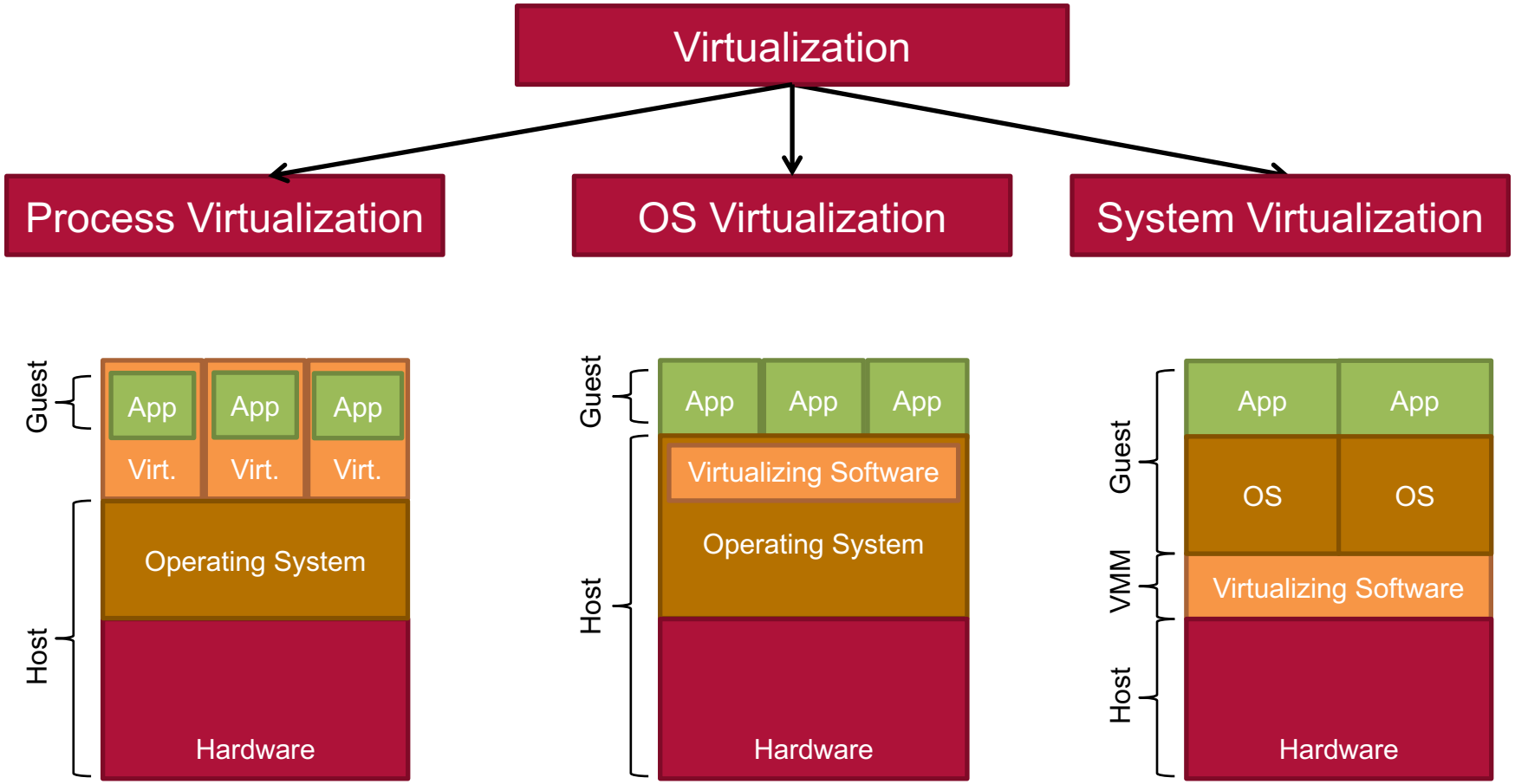
- Definition of virtualization according to NIST:

> "Virtualization is the simulation of the software and/or hardware upon which other software runs. This simulated environment is called a virtual machine (VM)."

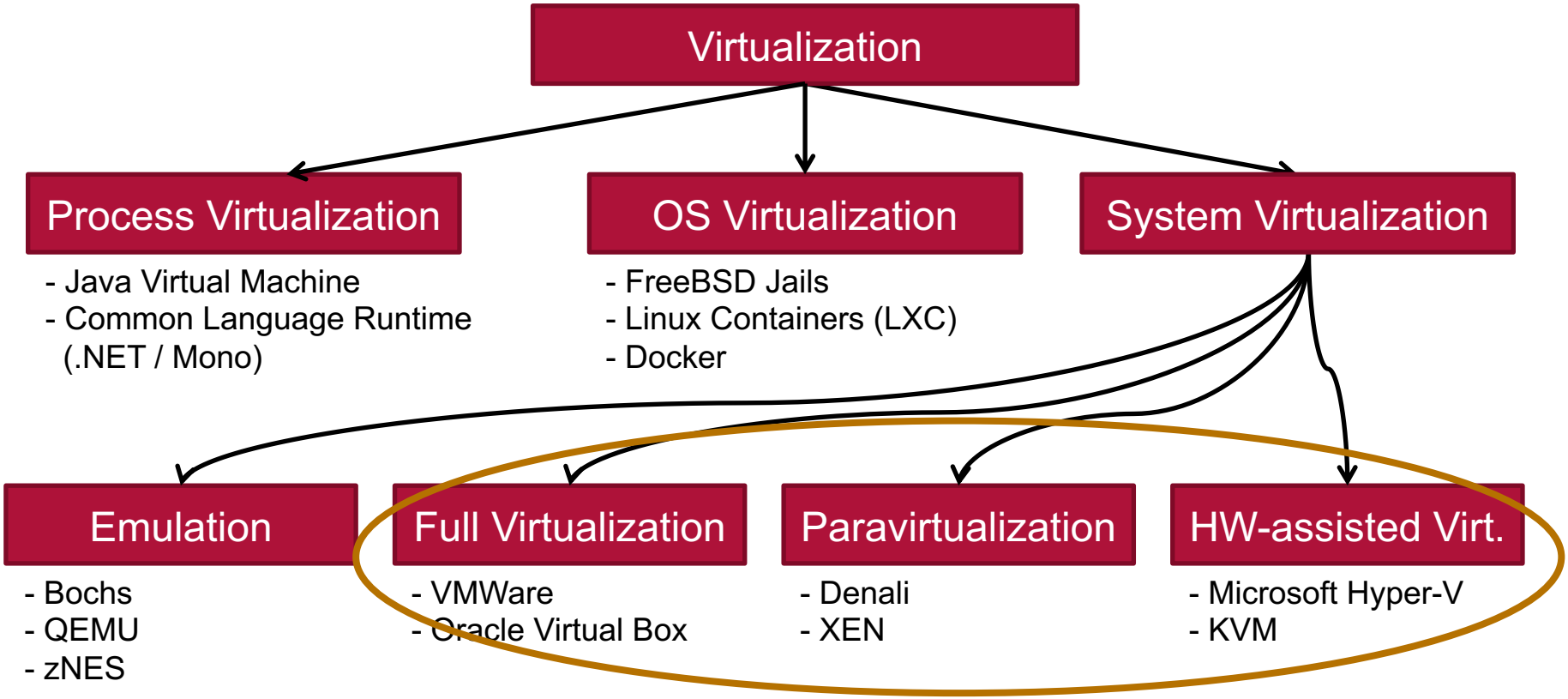
- Virtualization can transform a real system so
  - it looks like a different virtual system
  - multiple virtual systems

- **Real system** is often referred to as **host (system)**
- **Virtual system** is often referred to as **guest (system)**

Taxonomy of Virtualization (1/2)



Taxonomy of Virtualization (2/2)

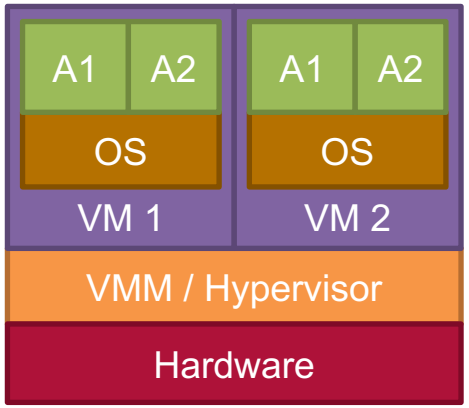


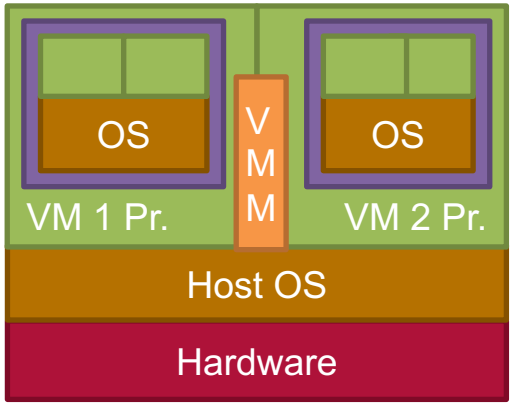
Relevant techniques for Infrastructure as a Services
(Also referred to as Hardware Virtualization)

Basic Designs for Hardware Virtualization

- VMM Type I
  - Directly on hardware
  - Basic OS to run VMs
  - Pro: More efficient
  - Con: Requires special device drivers
- VMM Type II
  - VMM as host OS proc.
  - VMs run as processes, supported by VMM
  - Pro: No special drivers
  - Con: More overhead

Also known as bare-metal/ hypervisor virtualization





Also known a hosted virtualization

Conditions for ISA Virtualizability (VMM Type I)

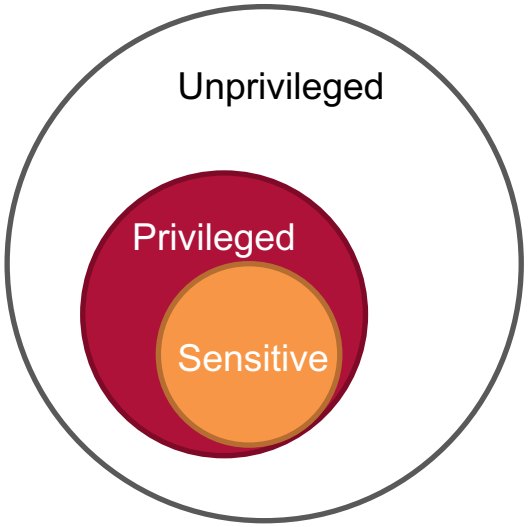
- Fundamental problem for hardware virtualization:
  - VMM must have ultimate control over hardware
  - Guest operating system must be disempowered without noticing

- Four assumptions in analysis of Popek and Goldberg[3]
  1. One processor and uniformly addressable memory
  2. Two processor modes: system and user mode
  3. Subset of instruction set only available in system mode
  4. Memory addressing is relative to relocation register

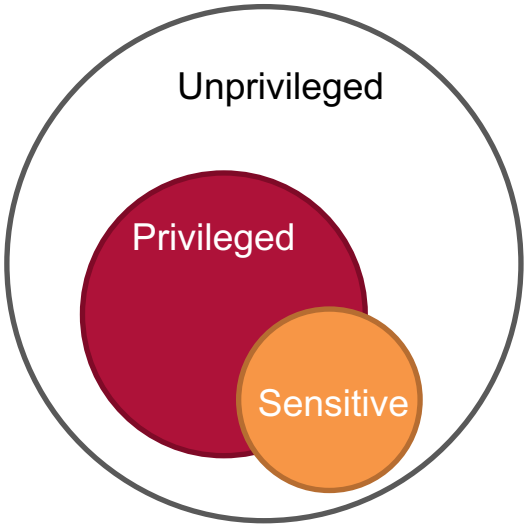
Popek and Goldberg's Theorem

- Basic condition for the construction of *efficient* VMMs

> "For any conventional third generation computer, a virtual machine monitor may be constructed if the set of **sensitive instructions** for that computer is a **subset** of the set of **privileged instructions**."



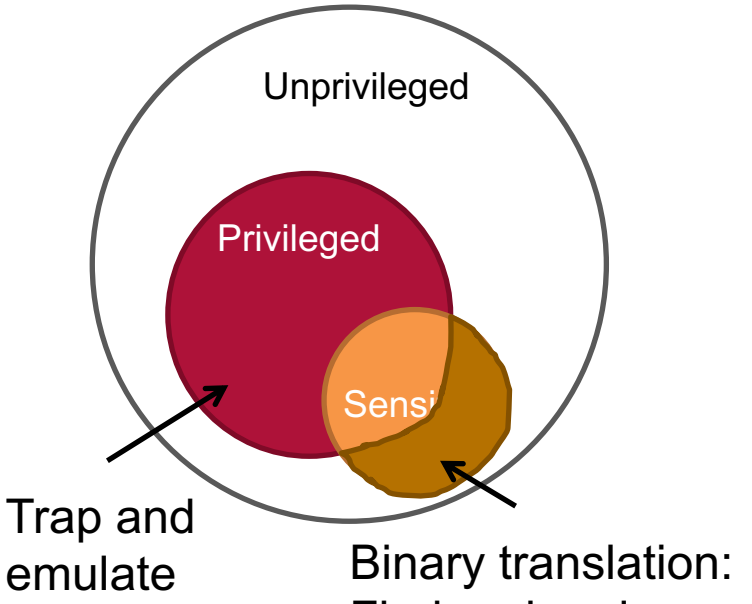
Condition satisfied



Condition unsatisfied

Full Virtualization using Binary Translation

- Translating a book word for word => inefficient
- Idea: Find critical instructions and replace them
  1. Run unprivileged instructions directly on CPU
  2. Trap and emulate privileged and sensitive instructions
  3. Find critical instructions and replace with exception



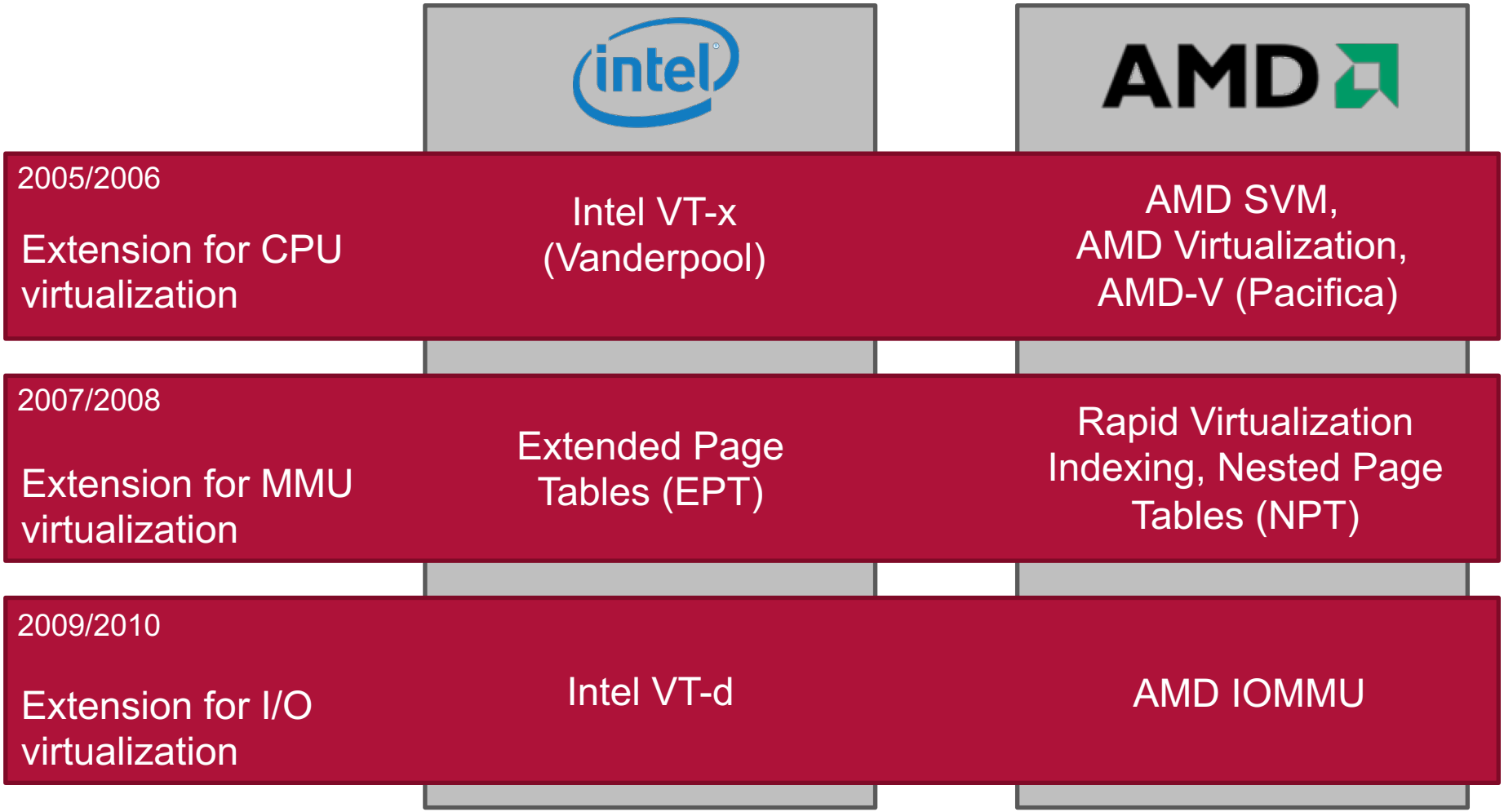
- Problem: Differentiation if critical or regular depends in some cases on the parameters used (e.g. LOAD-command)

  ➔Replacement must be done at runtime

VMWare Adaptive Binary Translation

- Modern CPUs are deeply pipelined
- Trapping privileged instructions can be too expensive
- Example: `rdtsc` (read time-stamp counter), Pentium 4[6]
  - Trap-and-emulate: 2030 cycles
  - Callout-and-emulate: 1254 cycles (*Callout method replaces traps with stored emulation functions*)
  - In-Translation Cache (In-TC) emulation: 216 cycles
- VMWare feature: Adaptive Binary Translation
  - Monitor frequency and costs of traps
  - Adaptively switch between different execution strategies at runtime

Incremental Hardware Support for Virtualization

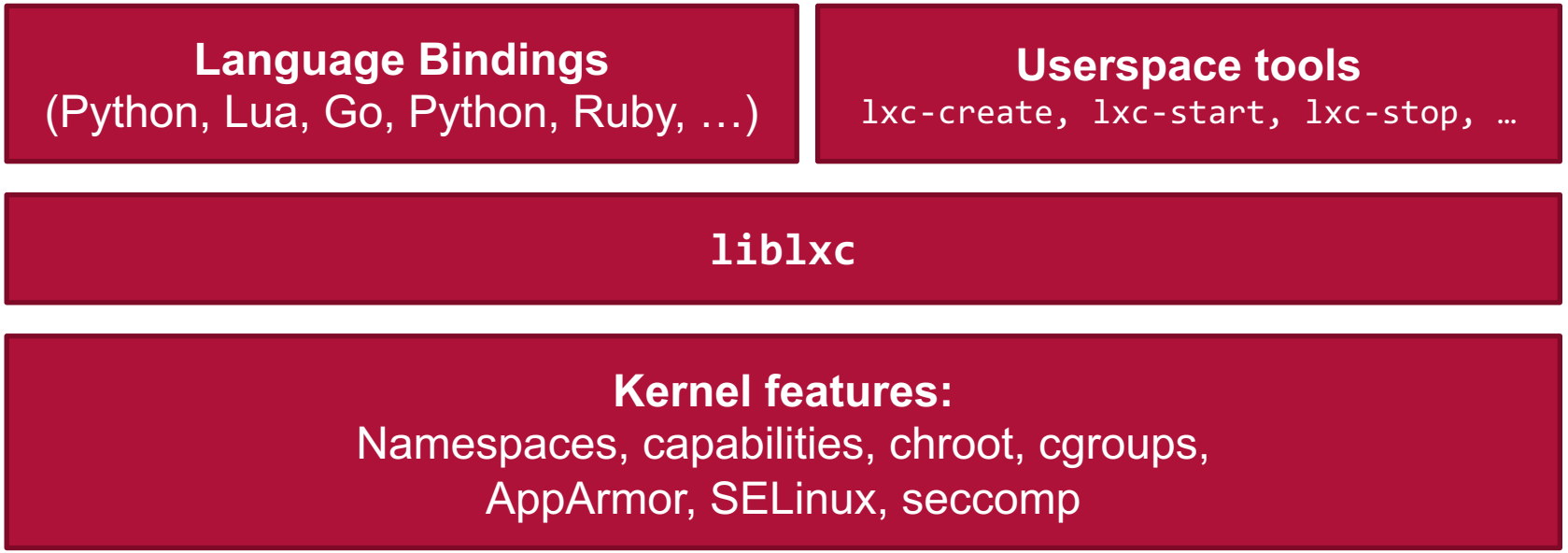


Linux Kernel Features for OS Virtualization

- Linux kernel contains large number of mechanisms for process isolation:
  - `chroot` system call
  - Namespaces
  - Capabilities
  - cgroups
  - SELinux
  - `seccomp`
  - …
- Partially overlapping functionality, but different configuration approaches

Linux Containers

- LXC: Library and userspace tools (bash scripts) to access kernel process isolation features
- No daemon, containers and their file systems are stored in `/var/lib/lxc`



Summary: Virtual Resources

- IaaS clouds let customers rent basic IT resources
  - Full control over OS and deployed applications in VMs
  - No long-term obligation or risk of over-/under-provisioning
- Virtualization as fundamental enabling technology
  - Several customers can share physical infrastructure
  - Different approaches to achieve virtualization
- Containers increasingly recognized as alternative or supplement to VMs

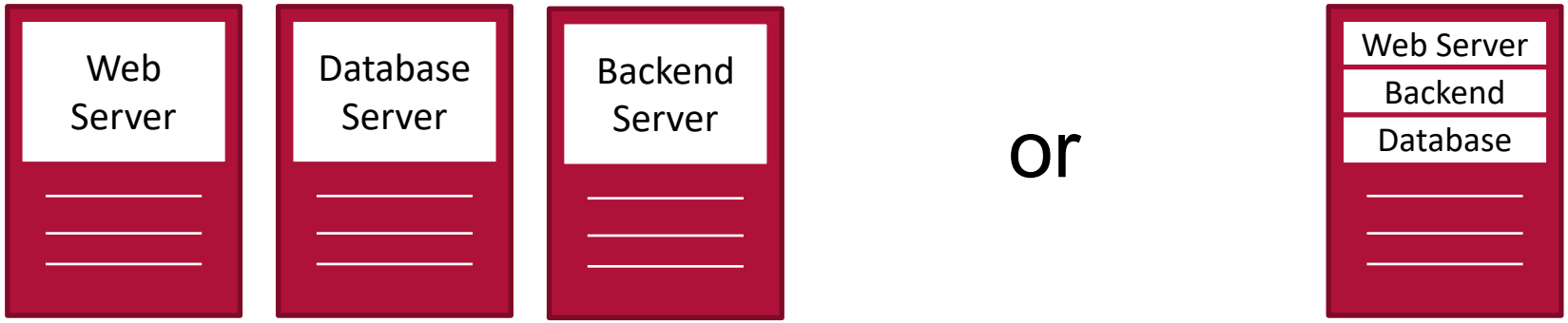
Methods of Cloud Computing

Chapter 3: Management of Virtual Resources

Complex and Distributed Systems
Faculty IV
Technische Universität Berlin

Operating Systems and Middleware
Hasso-Plattner-Institut
Universität Potsdam

“Iron Age”: Bare-Metal Servers



- Running components of an application on one or multiple physical servers:
  - Costs and agility: Purchase, housing, maintenance
  - Fault tolerance: Servers are single points of failure
  - Resource utilization: Servers were often underutilized, but sometimes also bottlenecks (i.e. w/ dynamic load)

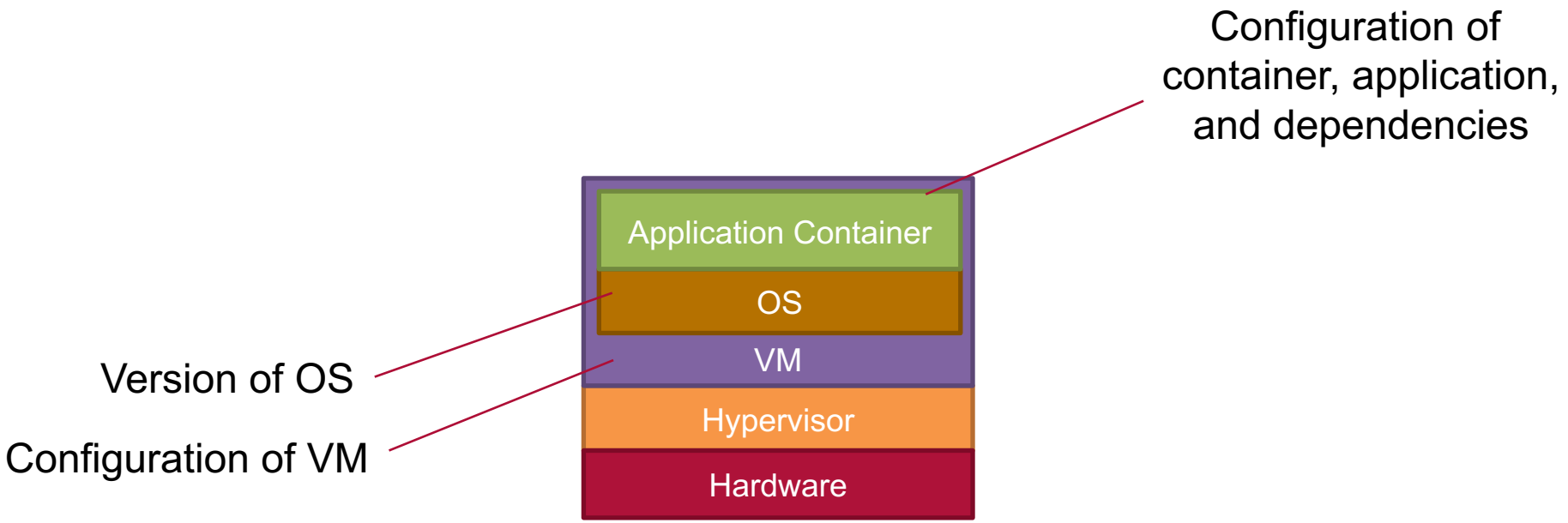
Management of Resources (1/2)

- Now we can create VMs and containers, but management of large sets of resources is still difficult:
  - Provisioning of VMs and containers
  - Configuration of systems
  - Monitoring and failure handling
  - Replication and load balancing

- Cloud Operating Systems: manage large sets of virtual resources running on large sets of physical resources

Management of Resources (2/2)

- Major remaining problems in the “Cloud Age”: server sprawl, configuration drift, snowflake servers… → Technical debt!



Cloud Operating Systems (1/3)

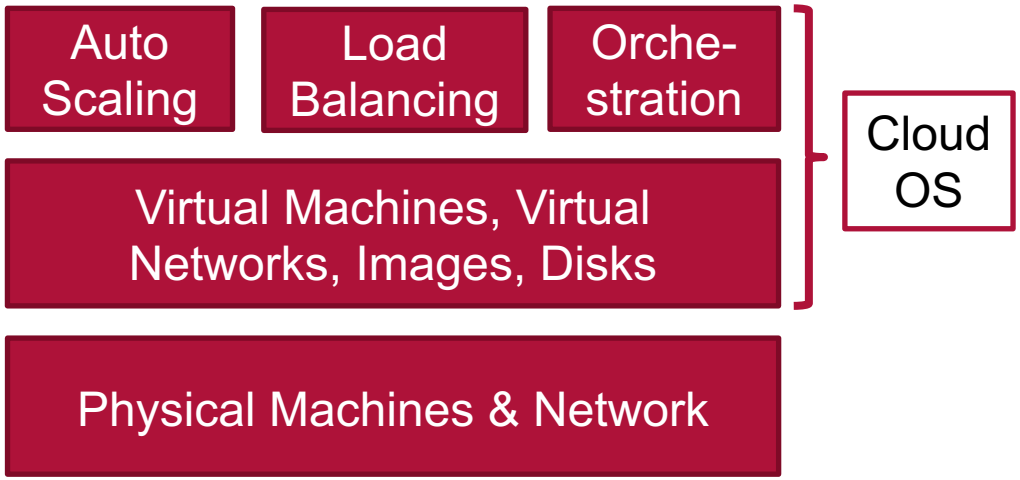
- Virtualization is immensely useful, but managing many virtual machines manually is impractical
- Therefore: Cloud Operating systems
  - Control large pools of compute, storage, and networking resources
  - Provides dashboards and APIs for datacenter operators (administration) and users (provisioning)

- Open Source cloud systems: OpenStack, OpenNebula
- Commercial public clouds: AWS EC2, GCE, Azure, Digital Ocean

Cloud Operating Systems (2/3)

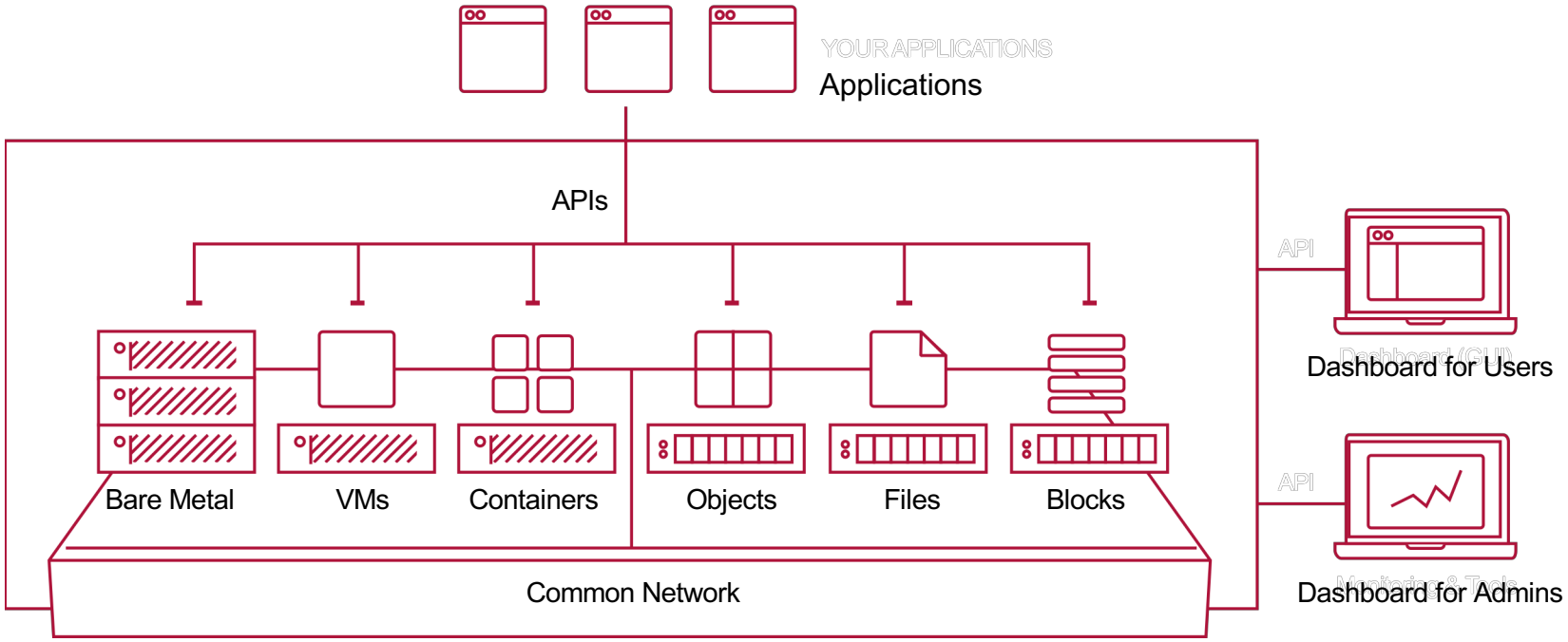
- User Interface and APIs for
  - spawn & maintain VMs,
  - virtual networking,
  - manage images & virtual disks

- Advanced features
  - load balancing,
  - auto scaling,
  - and orchestration



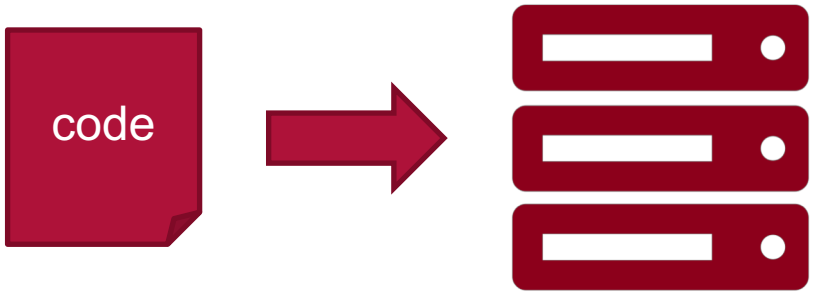
Cloud Operating Systems (3/3)

- Interfaces to manage and provision virtual resources



Infrastructure-as-Code

- Define servers, networking, and other infrastructure elements in source code:
  - Configuration of environments
  - Dependencies with specific versions



- Automation tools built around infrastructure definitions ➔ easily rebuild servers (instead of updating running servers: “immutable infrastructure”)
- Versioning and sharing of infrastructure definitions

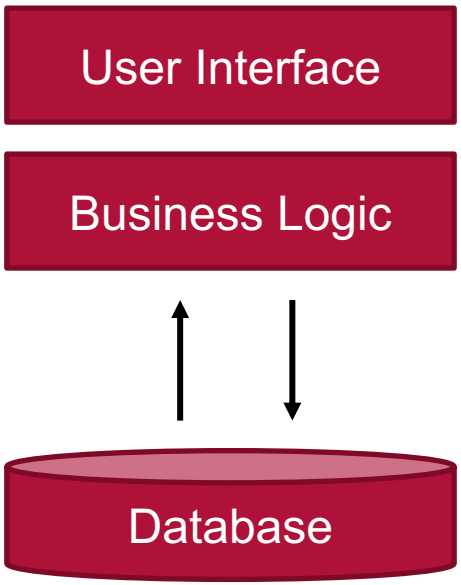
Ansible

- Automation language and engine
- Configuration management
  - i.e. configure existing servers
- Orchestration
  - i.e. allocate and provision new servers

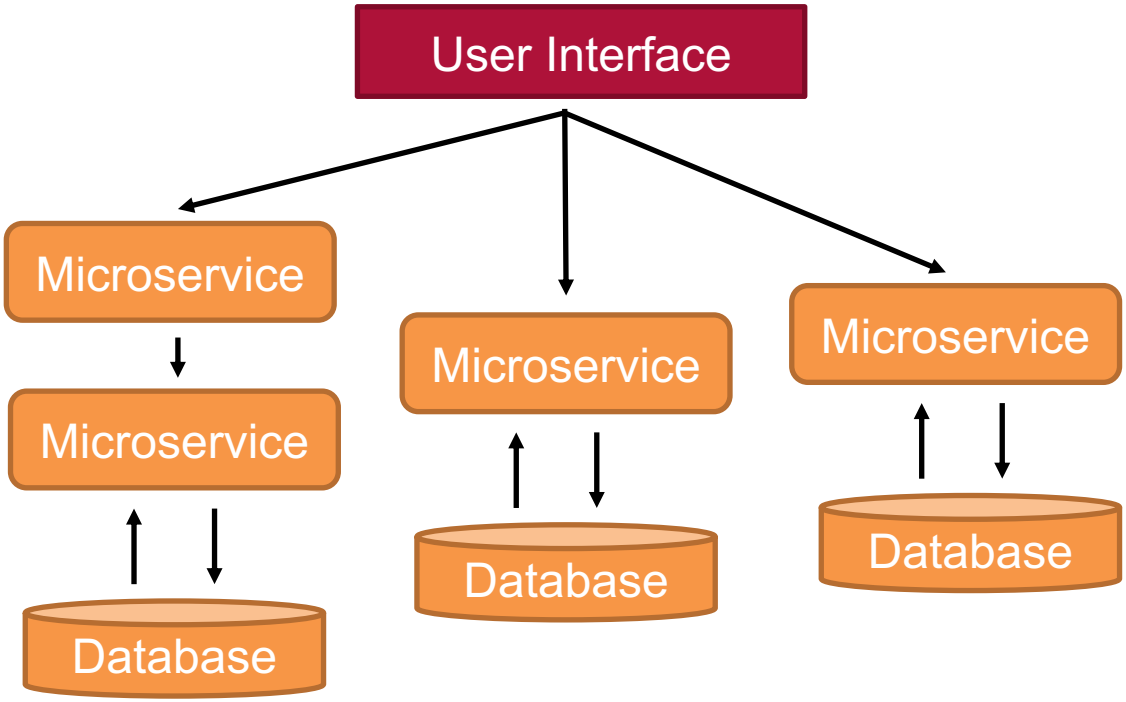
- Since 2012, acquired by Red Hat in October 2015
- Well-documented conventions and best practices, but no artificial limitations, so users can choose to ignore

Microservices (1/2)

Monolith

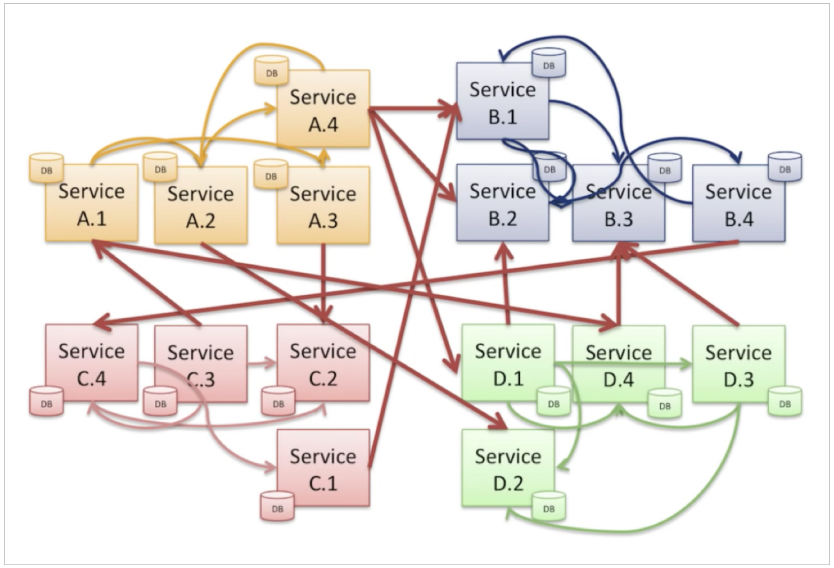


Microservices



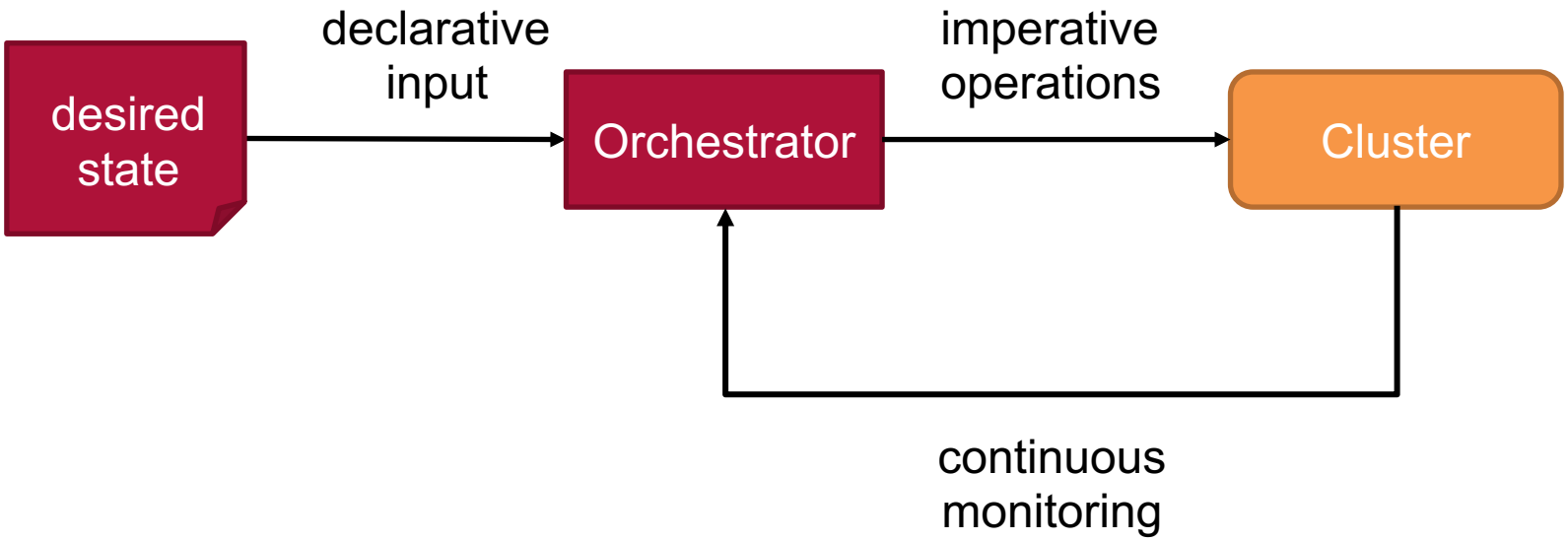
Microservices (2/2)

- Advantages
  - Independent development
  - Small teams
  - Fault isolation
  - Scalable!
- Disadvantages
  - Overhead (duplicated tech)
  - Management of services and networking

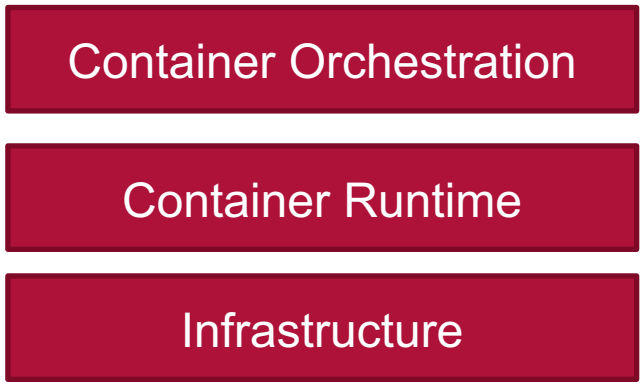


Orchestration

- Orchestration tools: Control systems for clusters



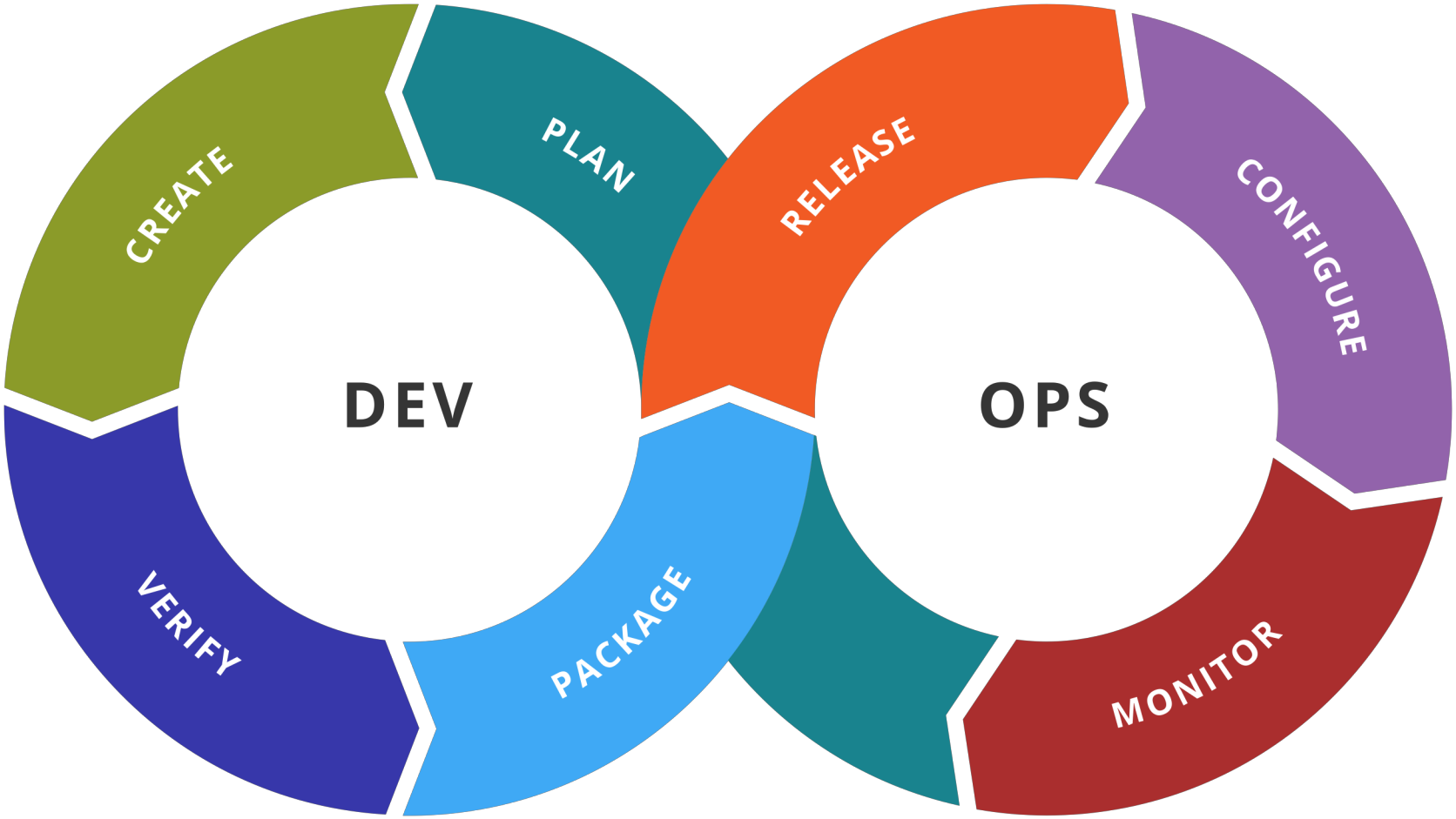
Container Orchestration



- Distributed container management
- Local container management
- Container-agnostic infrastructure

- Provisioning and deployment of containers
- Replication and availability of containers
- Configuration and networking of containers
- Load balancing across replicated containers
- Monitoring of replicated containers

DevOps


CREATE
PLAN
RELEASE
CONFIGURE
DEV
OPS
VERIFY
PACKAGE
MONITOR

Continuous Delivery

- Fast and reproducible software releases

- From the first principle of the agile manifesto:

  "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."

- CD Metric: How long does it take to release a single-line-code-change to production? *(cycle time)*
- Business case: Software development as investment → optimizing ROI through small cycle time

Summary

- Virtual machines and containers allow to quickly provision new servers, i.e. by using the interface of a Cloud OS
- Yet, server management and configuration remain very challenging (server sprawl, configuration drift, snowflake servers)
- Central Ideas
  - Infrastructure-as-Code: executable, understandable, and versioned descriptions of desired states as input to automation and orchestration tools
  - Cultural change to DevOps, for continuously and automatically deploying new versions to production

Methods of Cloud Computing

Programming Cloud Resources 1: Scalable and Fault-Tolerant Applications

Complex and Distributed Systems
Faculty IV
Technische Universität Berlin

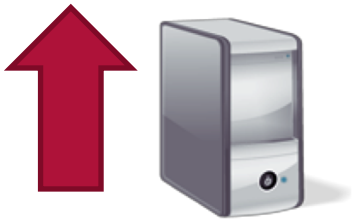
Operating Systems and Middleware
Hasso-Plattner-Institut
Universität Potsdam

Overview

- Intro
- Partitioning
- Replication and Consistency
- CAP Theorem
- Case Studies

How to Achieve Scalability / Availability?

- Two principal methods to scale



Vertical Scalability (Scaling Up)	
Idea	Increase performance of a single node (more CPUs, memory, …)
Pro:	Good speedup up to a particular point
Con:	Beyond that point, speedup becomes very expensive

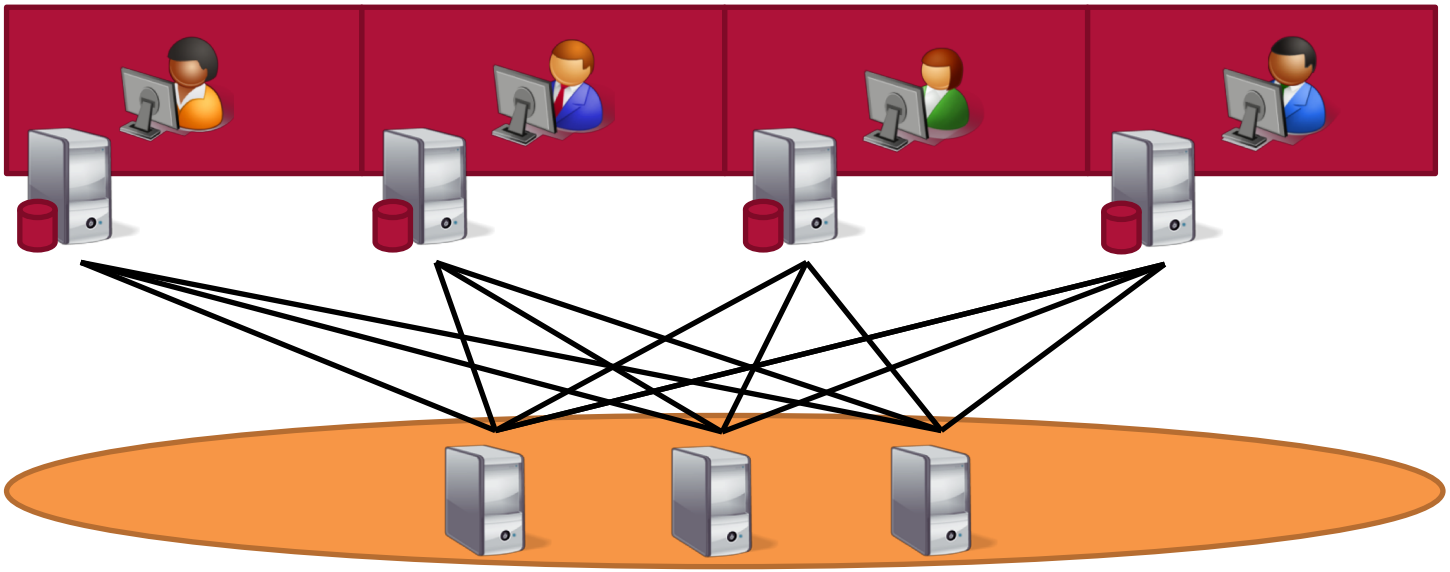
Horizontal Scalability (Scaling Out)	
Idea	Increase number of nodes
Pro:	Cheap to grow total amount of resources
Con:	More difficult for applications to efficiently utilize the resources

Popular Partitioning Schemes (1/2)

- Partitioning per tenant
  - Put different tenants on different machines
  - Pro: In clouds, tenants are expected to be isolated

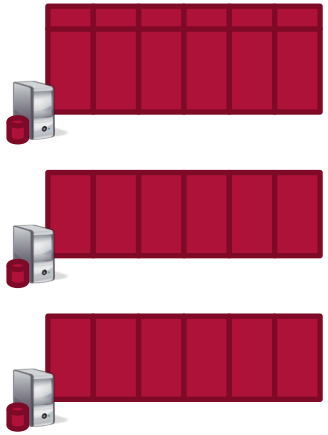
→No network traffic between machines, good scalability

→Con: Tenants cannot scale beyond one machine

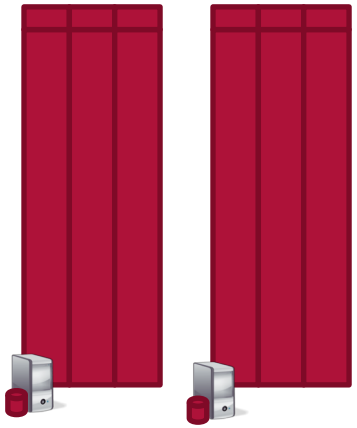


Popular Partitioning Schemes (2/2)

- Horizontal partitioning (relational databases)
  - Split table by rows
  - Put different rows on different machines
  - Reduced number of rows, reduced indices
  - Done by Google BigTable, MongoDB



- Vertical partitioning (relational databases)
  - Split table by columns
  - Not very common to improve scalability (?)



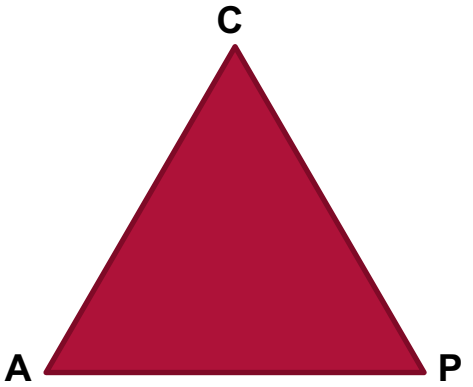
Two Views On Consistency

- Data-centric consistency models
  - Talk about consistency from a global perspective
  - Provides guarantees how a sequence of read/write operations are perceived by *multiple clients*

- Client-centric consistency models
  - Talk about consistency from a client‘s perspective
  - Provides guarantees how a *single client* perceives the state of a replicated data item

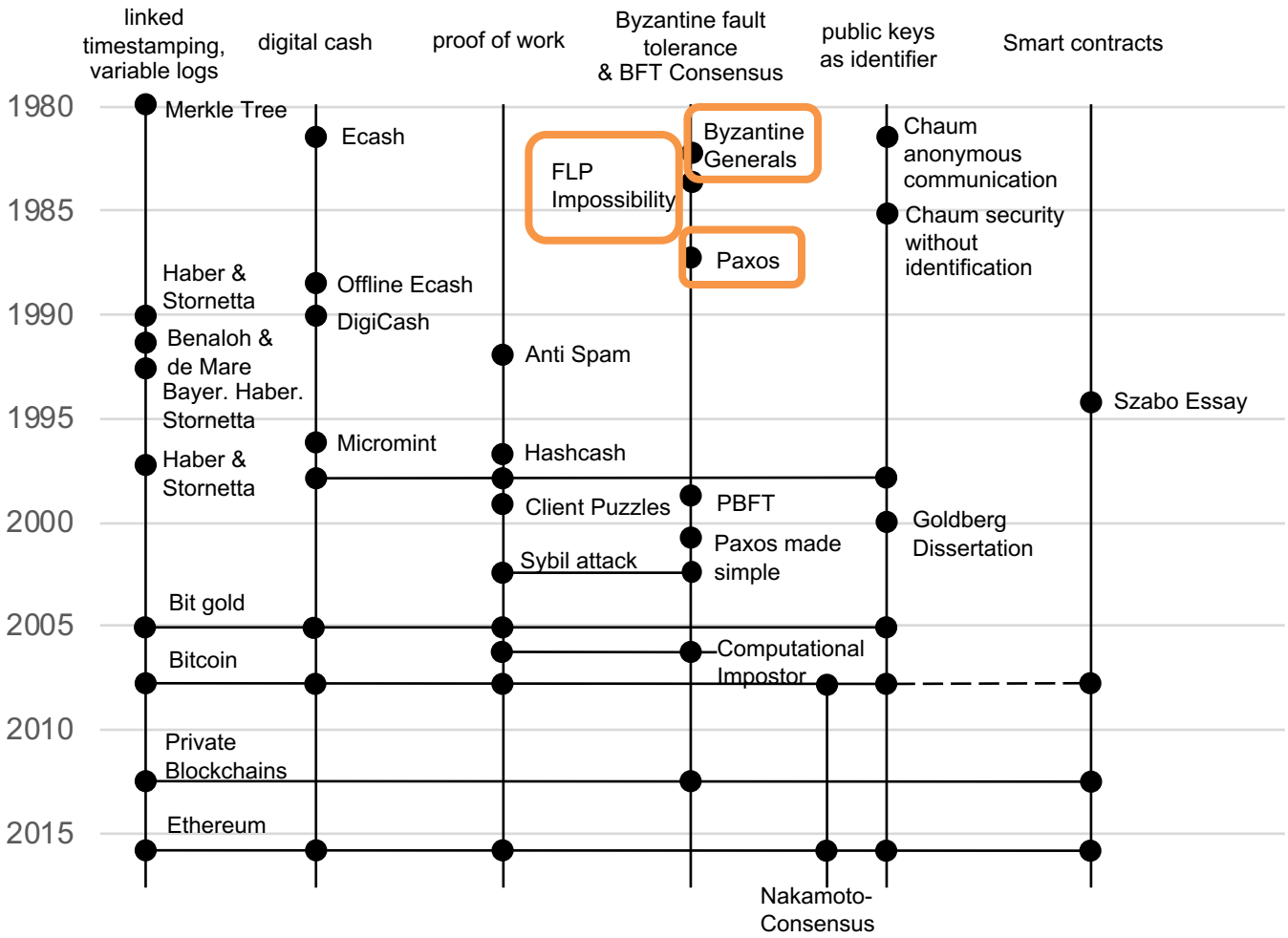
Brewer's CAP Theorem[2]

- In a distributed system, it is impossible to provide all three of the following guarantees at the same time:
  - Consistency: Write to one node, read from another node will return something no older than what was written
  - Availability: Non-failing node will send proper response (no error or timeout)
  - Partition tolerance: Keep promise of either consistency or availability in case of network partition



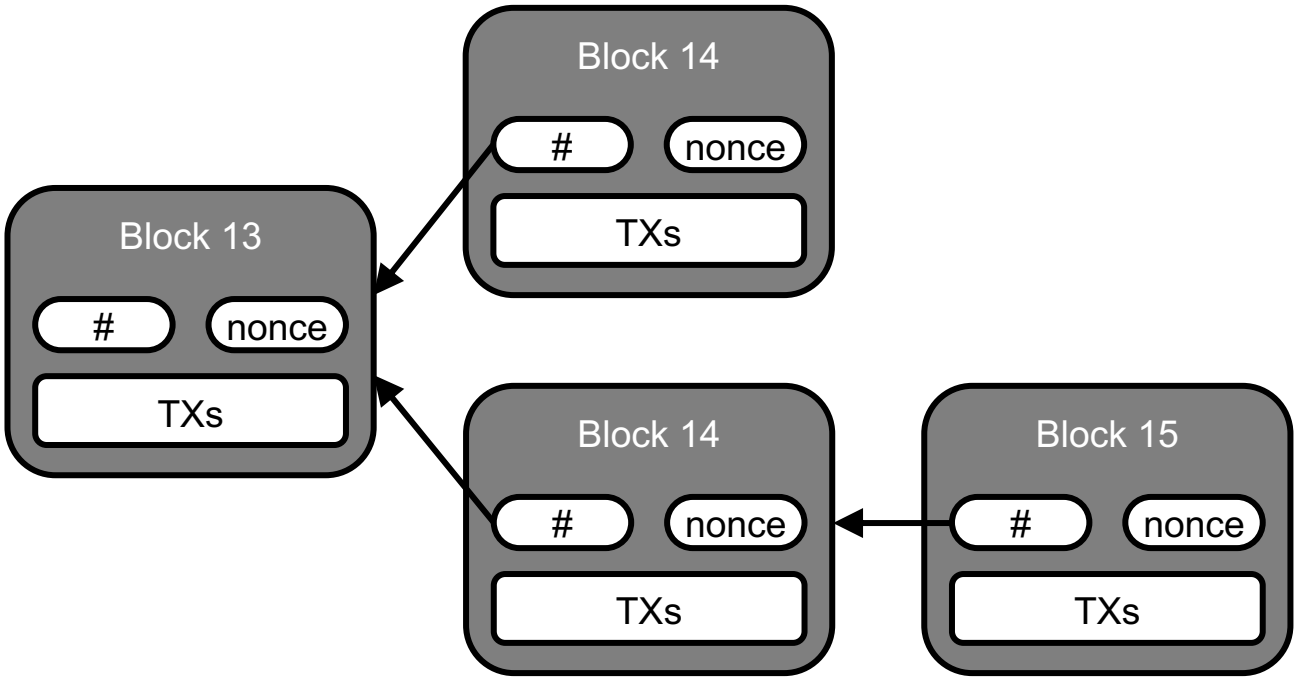
- Proof by Seth Gilbert and Nancy Lynch[3]

Blockchain: Background



Narayanan, Arvind, and Jeremy Clark. "Bitcoin's Academic Pedigree." *Queue* 15. 4 (2017): 20.

Double Spends



Nakamoto-Consensus

Voting is not explicit but implicit by signing blocks

→ voting weight proportional to computer performance

Summary

- Scaling-out to more virtual resources: scalability and fault tolerance
  - Load balancing for replicated stateless components
  - Load balancing *and* data consistency models for replicated stateful components
- The higher the consistency level, the less scalable replicated services are
- System components fail eventually, decide on either availability or consistency

Methods of Cloud Computing

Programming Cloud Resources 2: Data-Intensive Applications

Complex and Distributed Systems
Faculty IV
Technische Universität Berlin

Operating Systems and Middleware
Hasso-Plattner-Institut
Universität Potsdam

Overview

- Introduction
- Distributed Storage
  - GFS
  - HDFS
  - Bigtable
- Distributed Processing
  - MapReduce
  - Spark
  - Flink

Motivation

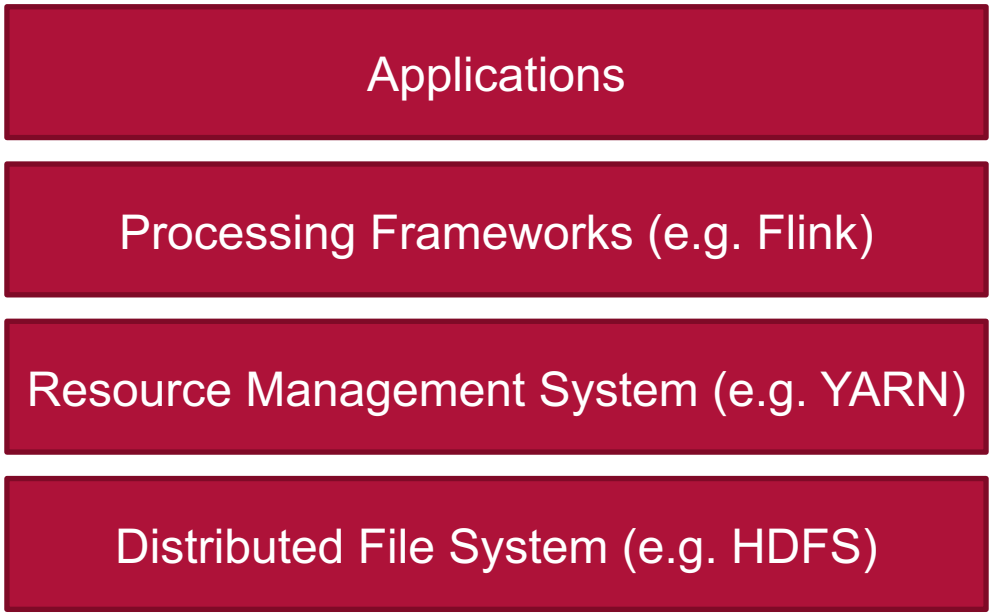
- Amount of digitally available data grows rapidly
  - WWW
    - Google processes hundreds of TB of Web pages and millions of clicks for its services[1,2]
  - Data warehousing
    - HP built 4 PB data store for Wal-Mart[3]
  - Scientific data
    - LHC produces 60 TB per day[3]
- Traditionally domain of Internet/search companies, e.g.
  - Google: GFS, MapReduce, Dataflow/Beam
  - Microsoft: Dryad, SCOPE

Challenges of Large-Scale Data Processing

- Large clusters/clouds have 100s to 1000s of servers
  - Extremely high performance/throughput possible
  - But jobs must be written to take advantage of the massively parallel and distributed environment
- Writing efficient parallel and distributed processing jobs is hard
  - Most developers are no experts in parallel programming, distributed systems, and data processing
  - Developers also don't want to care about concurrency and failures, but about extracting new information
- Needed: Suitable abstraction layers for developers

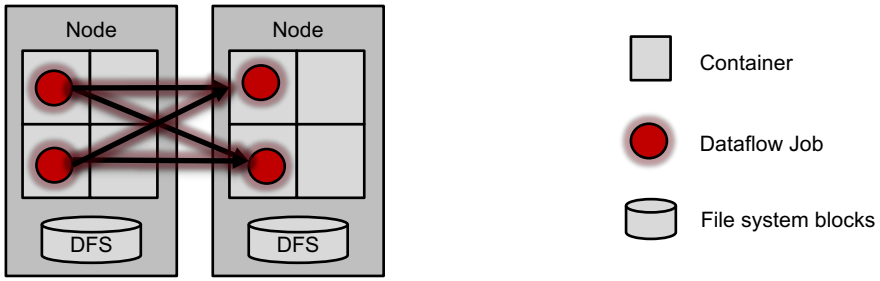
Analytics Cluster Setup (1/2)

- Analytics clusters often shared by multiple users and applications
- Shared via resource management and distributed file systems



Analytics Cluster Setup (2/2)

- Users reserve resources temporarily in containers from large sets of managed commodity nodes
- Containers: bundle of resources on a specific node assigned to a specific distributed job, often without any isolation (i.e. resource management “containers”, not necessarily OS-level containers)



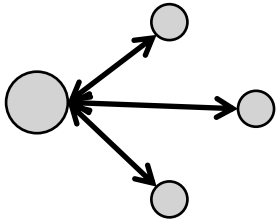
Comparison to HPC (1/2)

- Largely distinct communities and tools

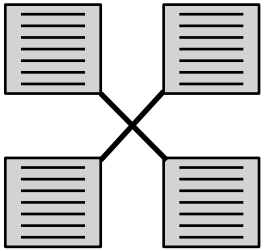
High-Performance Computing	Data-intensive Applications / Distributed Dataflows
Generic and low-level programming abstractions, explicit parallelism and synchronization and communication, optimized for specific hardware	High-level declarative abstractions, restricted programming models, comprehensive frameworks and runtime environments
Often more tightly coupled dependent computations, messaging and fine-grained updates to distributed state	Typically data-parallel problems with little coupling, parallelizable work that exceeds distributed communication
High-performance parallel computers and high-speed interconnects	Commodity nodes, used for both compute and storage, and commodity network technology

Comparison to HPC (2/2)

Flexible and fast low-level code

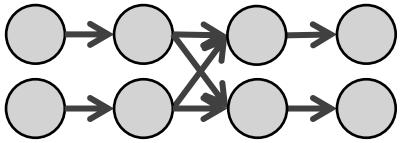


Architecture-specific implementations

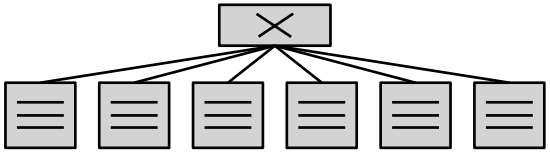


High-performance hardware

High-level dataflows



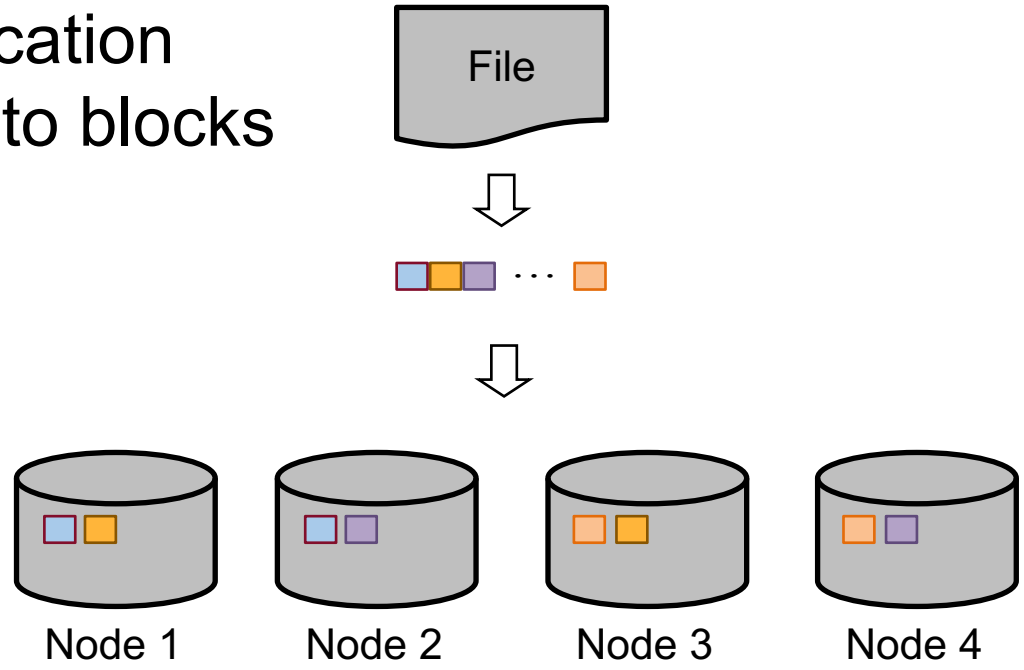
Scalable fault-tolerant distributed engines



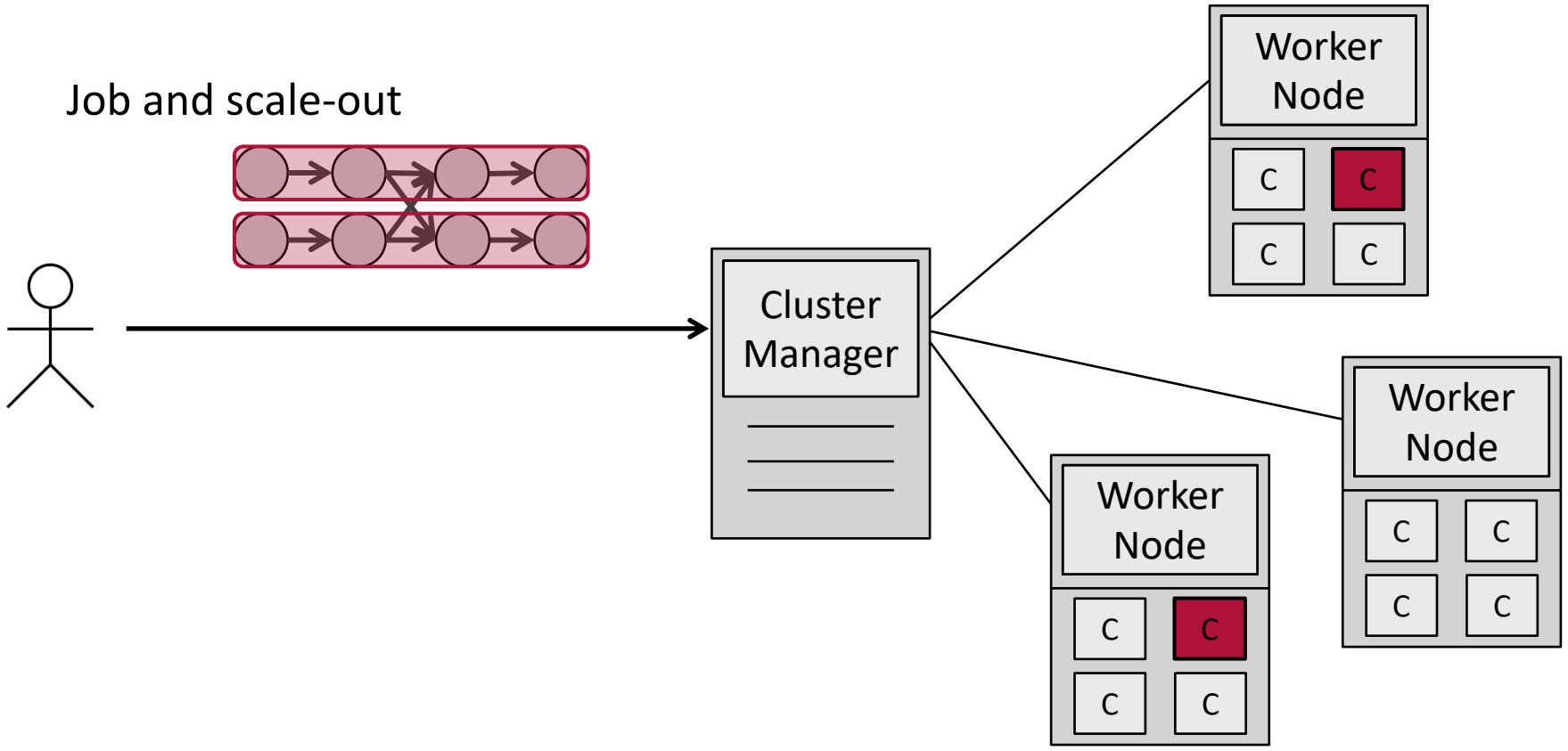
Commodity clusters

Distributed Data

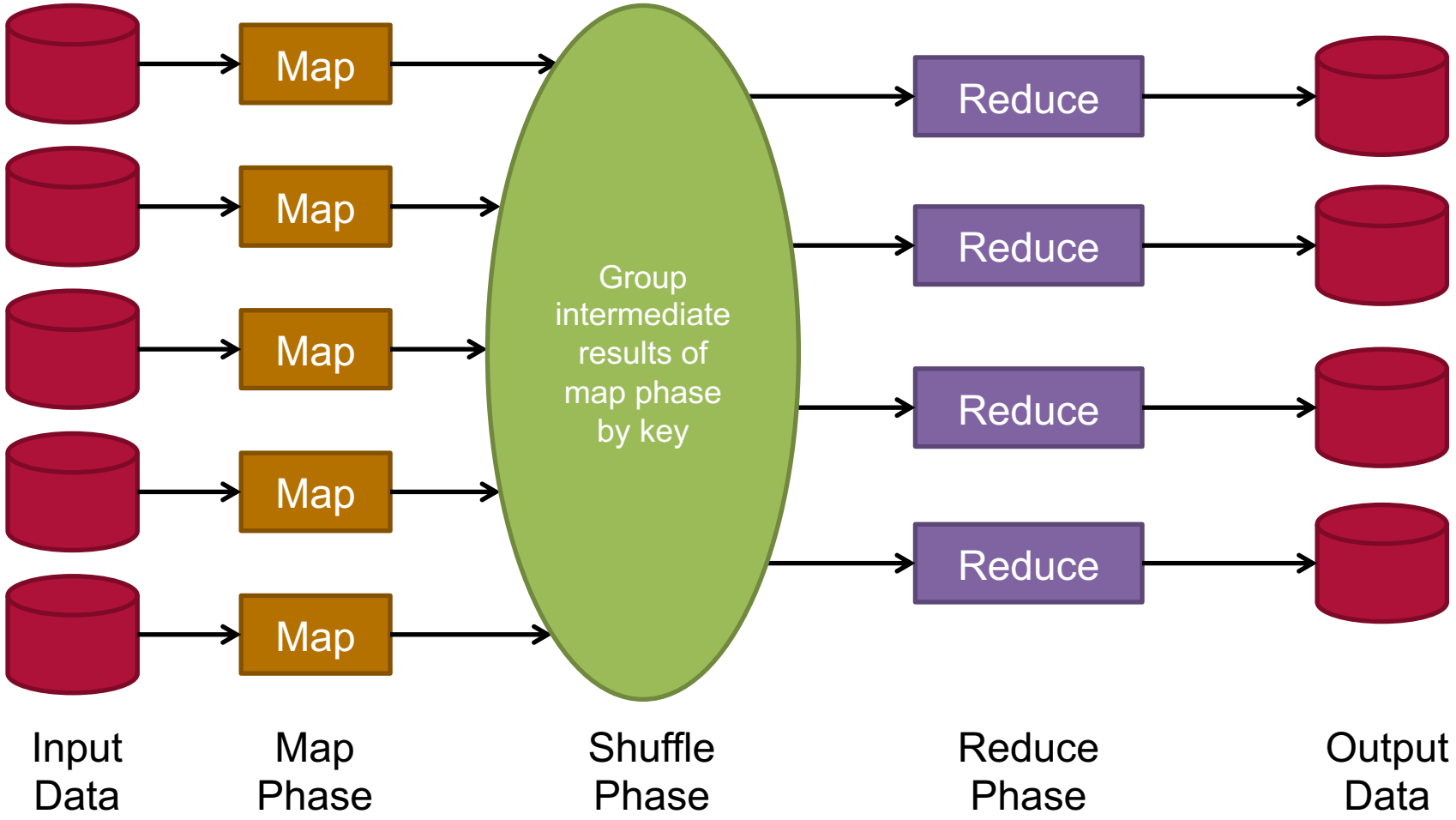
- Distributed file systems and systems on top (e.g. Bigtable on GFS), running on commodity servers
- Fault tolerance and parallel access through replication of data that is split into blocks



Distributed Execution on Commodity Nodes

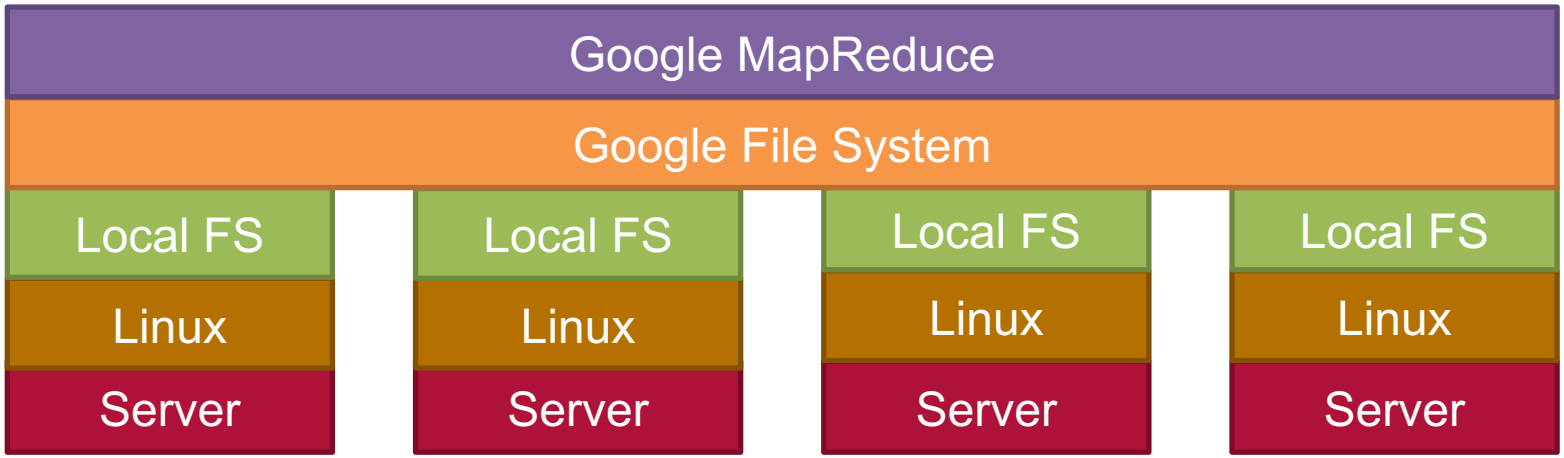


High-Level View on a MapReduce Program

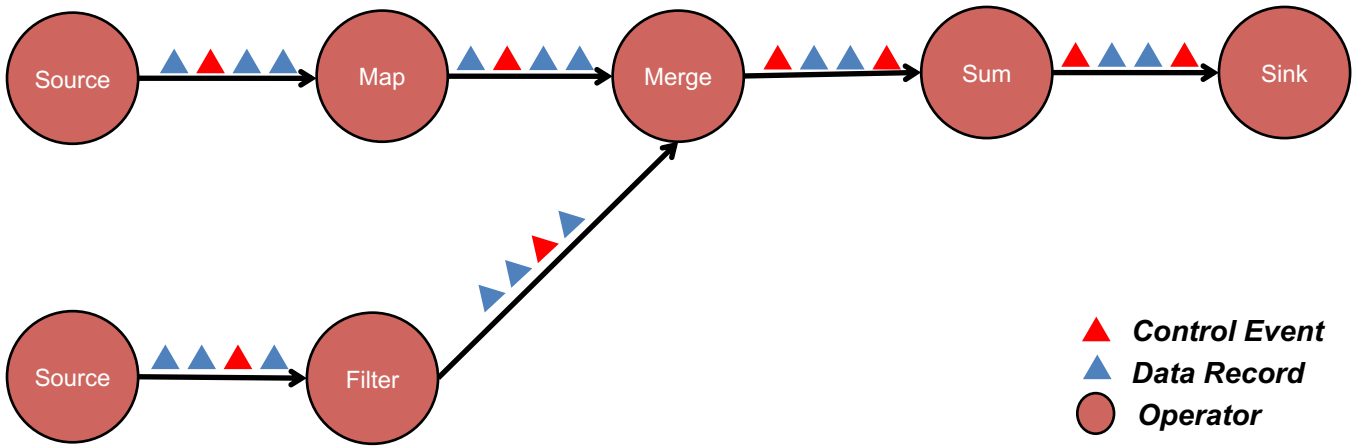


MapReduce Implementation (1/2)

- Designed to run on large sets of shared-nothing nodes
- Further assumptions
  - Expects distributed file system
    - Every node can potentially read every part of input
  - Individual nodes can fail
  - Prefers local data (computation is pushed to data)

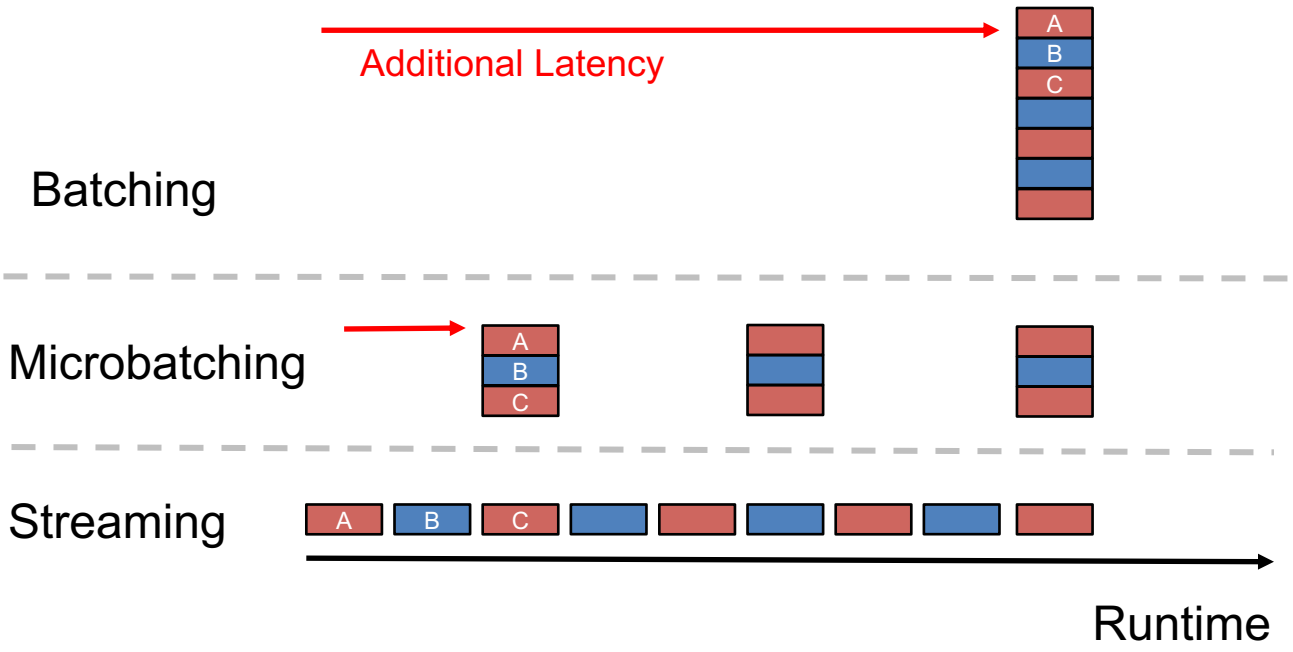


Apache Flink Execution Model (1/3)



- A job consists of:
  - A directed acyclic graph (DAG) of operators and
  - Intermediate streams of data records and control events flowing through the DAG of operators
- Pipelined/Streaming engine

Performance Comparison of Streaming Approaches (1/3)



- Microbatches: Processing of small batches of tuples
- “Real” Streaming: Tuple-wise processing (lower latencies possible, but might cost throughput)

Summary

- Major trends of the last two decades
  - Single processors stopped getting faster as previously
  - Clusters of commodity resources and access via VMs or containers in public clouds
  - More data is recorded and stored
- New distributed systems to utilize cluster resources
  - Distributed storage systems for files and also more structured data
  - Distributed analytics with dataflow systems for search and relational processing, graph analysis, machine learning etc.

Methods of Cloud Computing

Platforms / Platform-as-a-Service

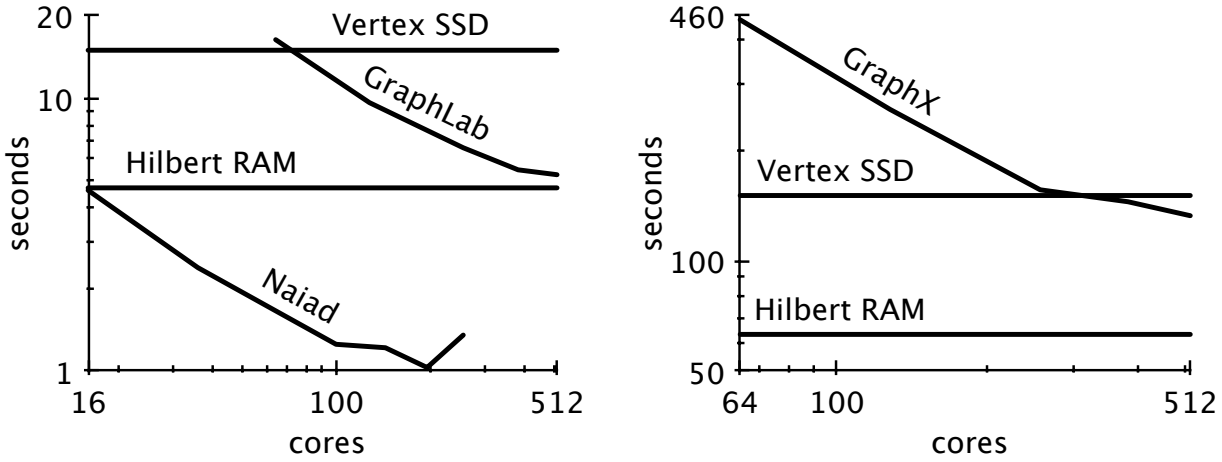
Complex and Distributed Systems
Faculty IV
Technische Universität Berlin

Operating Systems and Middleware
Hasso-Plattner-Institut
Universität Potsdam

Overview

- **Intro**
  - Cost of Scalability
  - Platforms vs Infrastructure
  - Abstraction Levels of Platforms
- Platforms
  - Azure
  - Amazon EMR and SageMaker
- Serverless Computing
  - Concept, FaaS, BaaS
  - Serverless Architectures and Implications
  - Examples

COST of Scalability: Scaling measurements



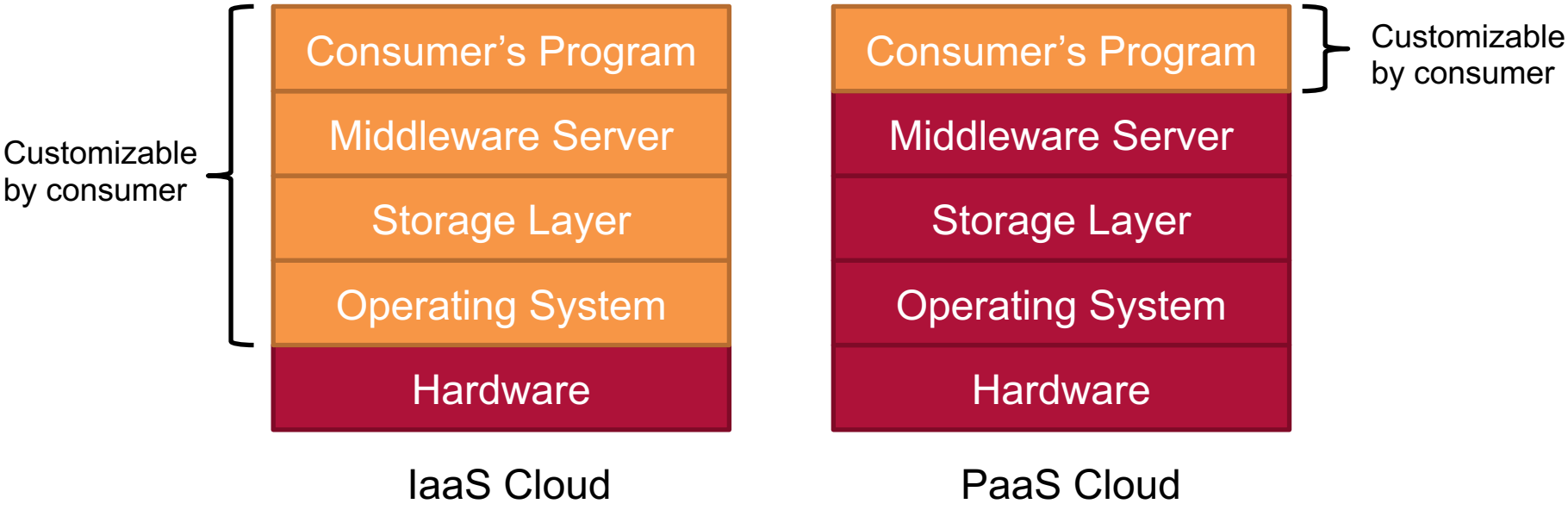
Naiad has COST of 16 Cores, GraphX unbounded COST

→ When to use scalable systems is an important decision

Also: Amdahl's Law, Gustafson's Law

IaaS vs. PaaS (1/2)

- PaaS offers higher abstraction level compared to IaaS
  - Less development/maintenance effort
  - Less flexibility, high provider dependence



PaaS Abstraction Levels

- Many levels of abstractions in PaaS
  - Execution environments (like on Heroku)
  - Databases
  - Distributed processing
  - Domain-specific workbenches (like SageMaker)
  - Complete services (like Rekognition)*
- A lot of new offerings are quite high-level PaaS

*almost SaaS, but mostly used by developers

PaaS Value Proposition

1. Maintenance
   - Hardware maintenance
   - OS patches
   - Middleware updates
2. Availability
   - Application/service will be available despite maintenance/outages
3. Scalability
   - Application/service will scale to thousands of concurrent users

Serverless Computing: Concept, FaaS, BaaS[11]

- Serverless computing is a cloud execution model
  - Provider runs the server-side and dynamically manages the resources
- Enabled by high-level PaaS offerings:
  - Backend as a Service
    - Use cloud database, authentication, etc. as backend to a rich client app
    - Often used for mobile apps
    - Examples: Firebase, Parse, AWS Cognito, ...
  - Function as a Service
    - Execute own code in event-triggered, stateless, and ephemeral compute containers
    - Examples: AWS Lambda, Google Cloud Functions, …

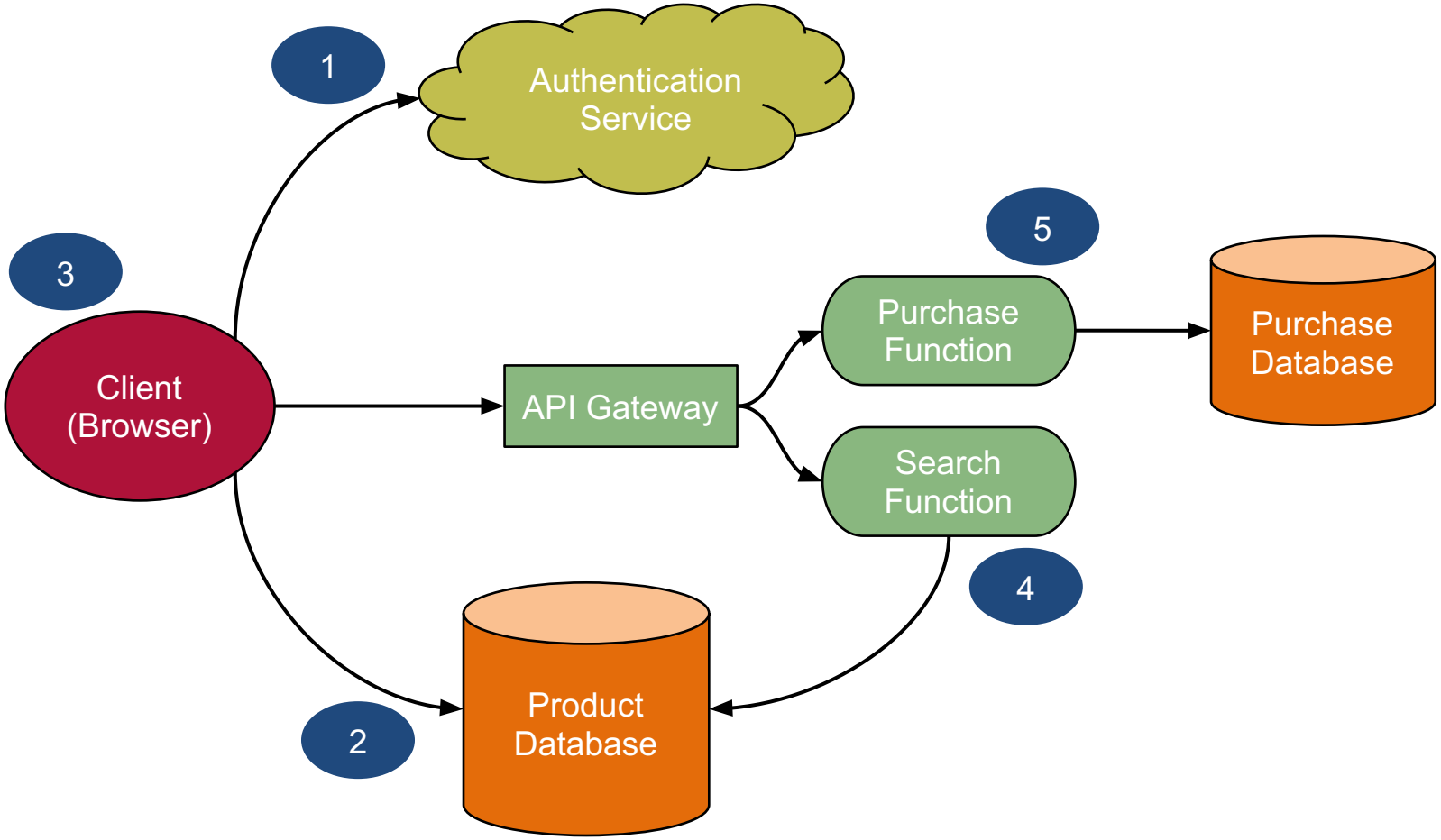
Serverless Architectures: Illustrating Examples (1/3)

A webshop as a traditional three-tier web application:

[11]

Serverless Architectures: Illustrating Examples (2/3)

A possible serverless architecture for the webshop:



[11]

Summary

- Managed platforms allow developers to focus more on their applications and less on operations
  - PaaS: infrastructure and runtime managed, yet users typically still allocate resources
  - FaaS: users only write and upload their functions, provider manages allocation and scaling of resources
- Platforms available for all kinds of use cases: e.g. long-running Web applications (e.g. Azure Platform), event processing (e.g. AWS Lambda), scalable data processing jobs (e.g. AWS EMR) → choose wisely!
- New cloud execution model: Serverless Computing

Methods of Cloud Computing

Beyond Cloud

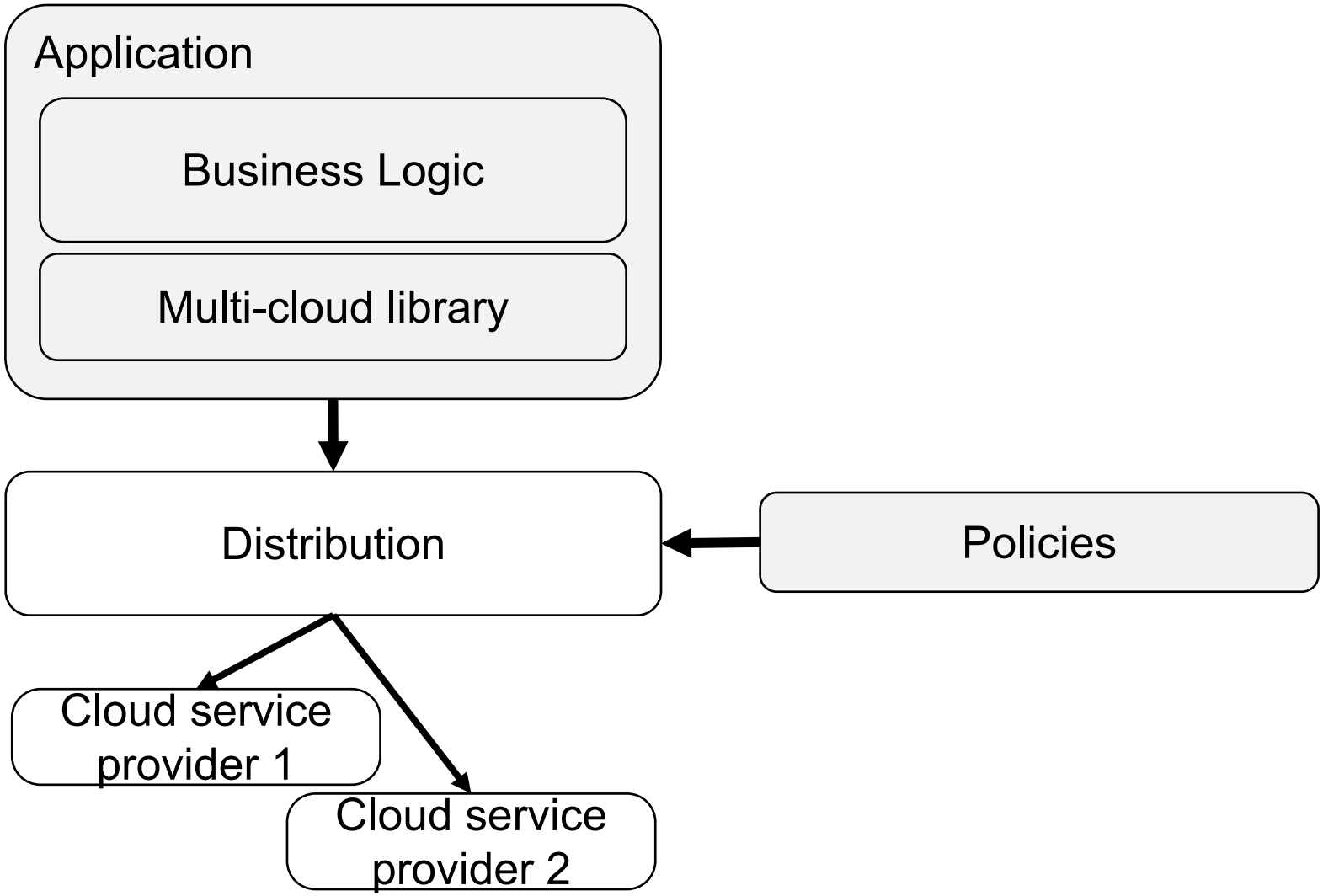
Complex and Distributed Systems
Faculty IV
Technische Universität Berlin

Operating Systems and Middleware
Hasso-Plattner-Institut
Universität Potsdam

Overview

- Intro
- Federated, Hybrid and Multi-Clouds
  - Concepts
  - Policy enforcement
- Fog, Edge and IoT
  - Concepts
  - Placement in fog architectures
  - Development practices: Testbed for fog computing

Cloud Federation



Background

Cluster

- Many Computers in one Room

Grid

- Loosely coupled computers or clusters all over the world

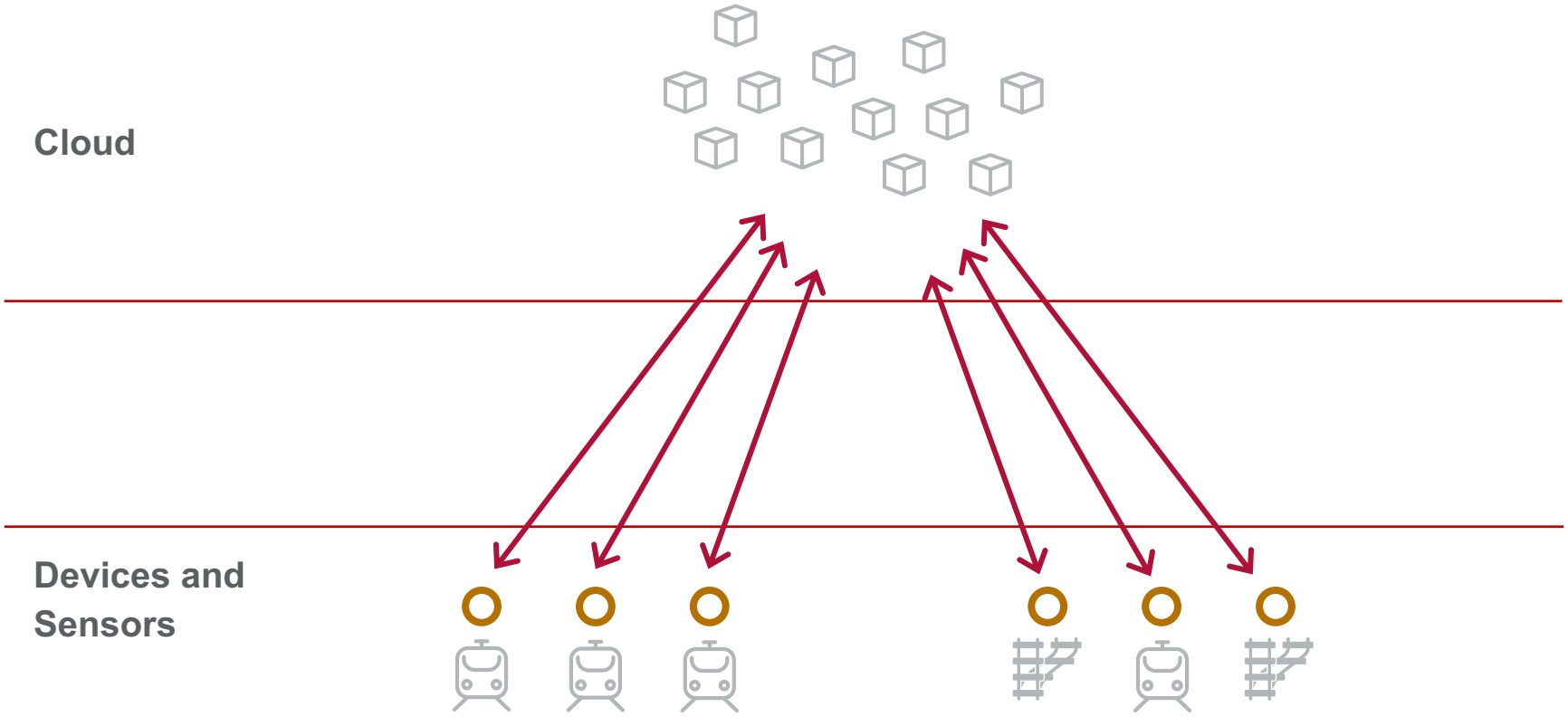
Cloud

- IT resources as a utility

Edge

- Extending the Cloud to the edge of the network

Edge Computing



Edge Computing