



UNIVERSITY OF LEEDS

COMP5123M

Cloud Computing Systems

Information on the Module

Staff

- Karim Djemame
karim@comp.leeds.ac.uk
scskd@leeds.ac.uk
K.Djemame@leeds.ac.uk
- Office: Room 3.25d, School of Computing, Bragg Building

Module Objectives

- **Demonstrate** an understanding of cloud computing techniques and technologies;
- **Identify** the paradigms that determine the requirements, capabilities and performance of Cloud systems;
- **Demonstrate** an understanding of the contexts in which big data systems are applied;
- **Design** a high-level framework of a Cloud architecture;
- **Use** a range of middleware tools to implement a cloud application;
- **Reason** about the significance of the new directions that Cloud computing is taking.

COMP5123M

- Lectures
 - Monday 5-6, Roger Stevens LT21
 - Friday 3-4, Chemistry LT A (2.15)
- Lab sessions shared with other modules,
2.05 Lab. Bragg Building
 - Tuesday 9-10
 - Wednesday 9-10, 1-3, 3-5
 - Thursday 3-5

Prerequisites

- The stated prerequisite for this module requires students
 - To have some background in distributed systems and networking
 - To be able to write code independently
- This module does not teach
 - the concepts of programming
 - the skills to program in a programming language

Prerequisites (2)

- General distributed systems concepts, e.g.
 - Terminology
 - Middleware
 - Service Oriented Architectures
 - Web services
 - Networking aspects

Assessment

- Coursework
 - Coursework 1
Out: 17/02/23 MCQ/Problem Solving 20%
 - Coursework 2
Out: 15/03/23 Problem Solving 20%
- Examination (60%)
 - Open book
 - May/June 2023

Text books

- *Cloud Computing for Science and Engineering.* I. Foster and D.B. Gannon. MIT Press, 2017
- *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things.* Hwang K, Fox G.C. & Dongarra J.J., 1st edition, Morgan Kaufmann, 2012
- *Cloud computing.* S. Bhowmik, Cambridge University Press, 2017

Text books – Other

- *Distributed Systems - Principles and Paradigms.* Tanenbaum A.S. and van Steen M., Prentice Hall, 3rd Edition, 2017
- *Computer Networking: A Top-Down Approach.* J. Kurose and K. Ross, 7th Edition, Pearson, 2017

Course Outline - Themes

1. Distributed system models and enabling technologies
2. **Cloud computing** – architectures, services, models, use cases
3. Public and private clouds
4. Virtualization of clusters/data centres, containers and unikernels
5. Virtual Infrastructure Management
6. Resource management and Kubernetes
7. Cloud programming and software environments
8. Serverless architectures
9. Cloud middleware and configuration management
10. **Big data.** Mapreduce and Hadoop
11. Energy Efficiency
12. Security and trust
13. Ubiquitous clouds and Internet of things
14. Edge Computing
15. Disaggregated computing
16. Clouds in the Exascale

Course Outline

Week	Lectures	Lab. Session
1	Module Introduction Distributed system models and enabling technologies	Introduction to virtualization and cloud simulation -
2	Cloud computing – architectures, services, models, use cases	Virtualization
3	Public and private clouds. Inter cloud resource management. Federated clouds Virtualization of clusters/data centres and containers	Cloud monitoring
4	Virtual Machine Scheduling Virtual Infrastructure Management: Openstack Kubernetes	VM and Virtualisation / Docker / Kubernetes
5	Cloud programming and software environments	Programming Model
6	Cloud middleware and configuration management Serverless architectures	Serverless architectures
7	Big data. Mapreduce and Hadoop	Hadoop and Spark
8	Energy Efficiency	Big data systems/Technologies
9	Security and trust	Energy Efficiency
10	Ubiquitous clouds and Internet of things Edge computing	Putting everything together
11	Current Trends – clouds in the exascale. Disaggregated computing. Revision session	-

Support

- Web-based resources
<http://www.minerva.leeds.ac.uk/>
- Weekly Lab. exercises
- Microsoft Teams Discussion Board

Minerva

- Module pages are available on Minerva
www.minerva.leeds.ac.uk/ - login and click on COMP5123
- Lecture Notes
- Lab. Sessions
- Coursework Specifications
- Reading List
- Past Exam Paper
- Links/Resources

Support for Practical Work: Microsoft Azure

- Bragg Teaching Lab. 2.05



- Microsoft Azure
 - The school has allocated [credit](#) (approximately £100)
 - Get free \$100 credit through a student subscription:
<https://azure.microsoft.com/en-gb/free/students/>



Keeping well with the Module

- **Reading**
 - Text books
 - Articles/reports on Minerva
- **Practical Exercises**
 - Get familiar with the cloud access/technical material
 - Complete weekly Lab. Exercises
 - Useful when you do the coursework
- **Time management**
- **Weekly revisions**

Plagiarism Policy

- Any student involved in plagiarism
 - will be reported to the Head of School
 - will be subject to a disciplinary process
- University of Leeds web page for further details
<http://library.leeds.ac.uk/skills-academic-integrity>
- Academic Integrity Tutorial
<https://www.leeds.ac.uk/vle/students/assess/academicintegrity/>

Microsoft LEARN Initiative

<https://learn.microsoft.com>

- Microsoft Azure **Fundamentals Certification**
<https://learn.microsoft.com/en-us/certifications/azure-fundamentals/>
- Microsoft Azure **AI Fundamentals Certification**
<https://learn.microsoft.com/en-us/certifications/azure-ai-fundamentals/>
- **Free access** to material and Azure cloud
- You need to find the time to prepare for the certification
- Certification exam (1 hour) to take place in June 2023 (**fee waived**)
- Details already circulated.

COMP5123M – Cloud Computing



Introduction to Cloud Computing:

*Enabling Technologies
and Distributed System Models*

Plan of the Lecture

Goals

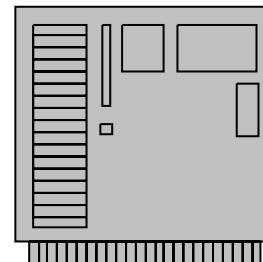
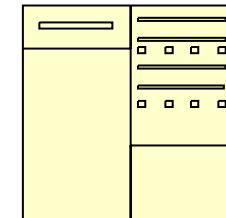
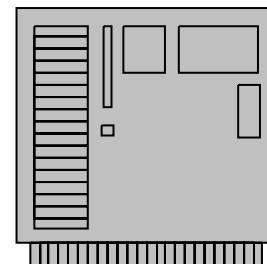
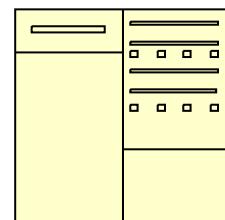
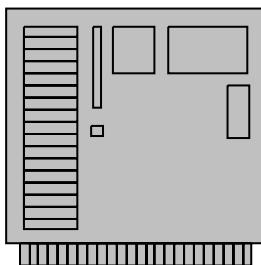
- Understand concepts of Cloud Computing

Overview

- Technology landscape
- Distributed computing evolution
- Towards a Definition of Cloud Computing
- Virtualised infrastructures
- Conceptual Cloud Architecture
- Conclusion

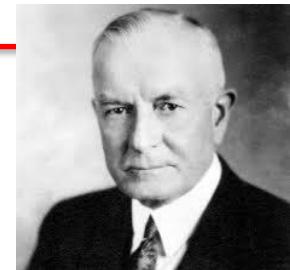
Just to start ... “Five computers”

- *"I think there is a world market for about five computers"* —
Remark attributed to Thomas J. Watson (Chairman of the
Board of International Business Machines) – 1943



Just to start ... “Five computers”

- *“I think there is a world market for about five computers”* — Remark attributed to Thomas J. Watson (Chairman of the Board of International Business Machines) – 1943



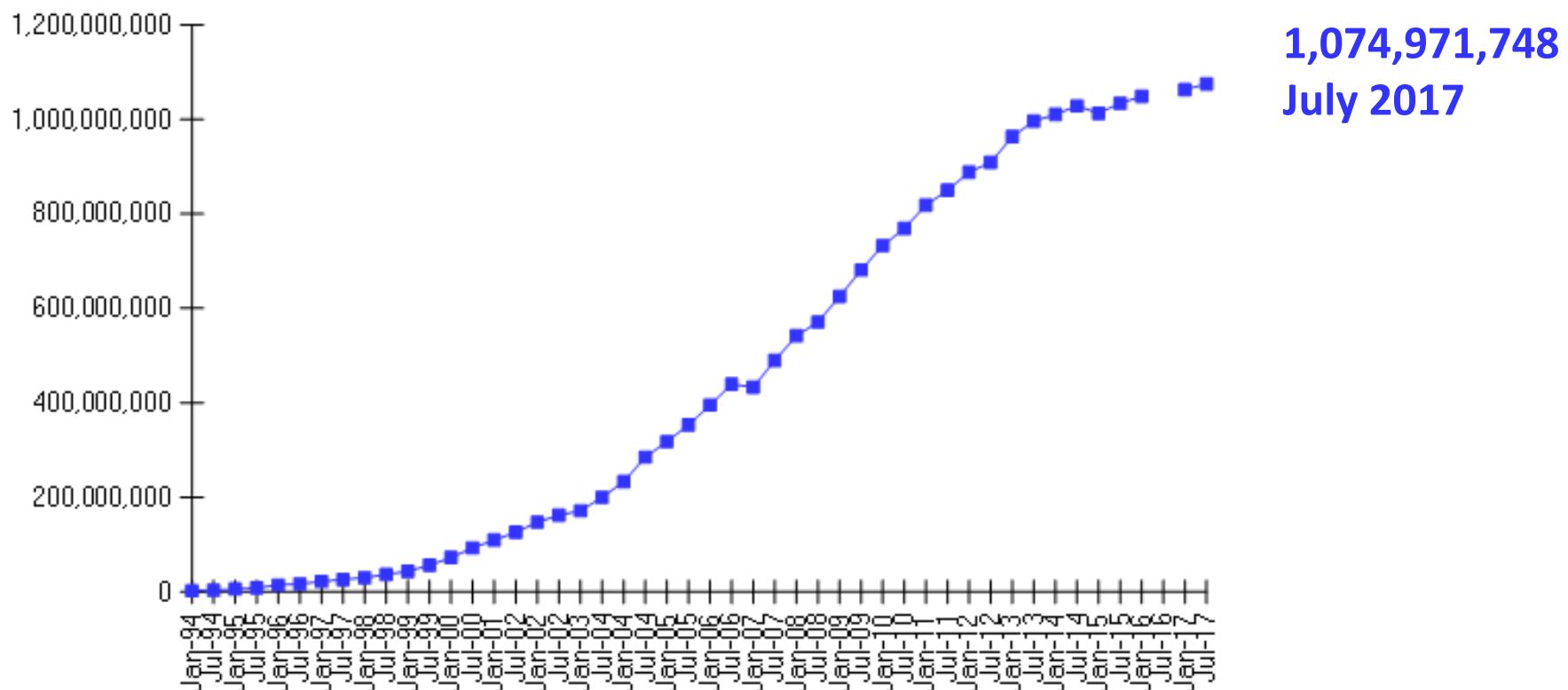
Length: 51 feet
Height: 8 feet
Weight: nearly 5 tons
Wire: 530 miles

Performance:
3 additions or subtractions in ... 1 second!

IBM Harvard Mark 1
Courtesy of IBM Archives

Scalability in Size

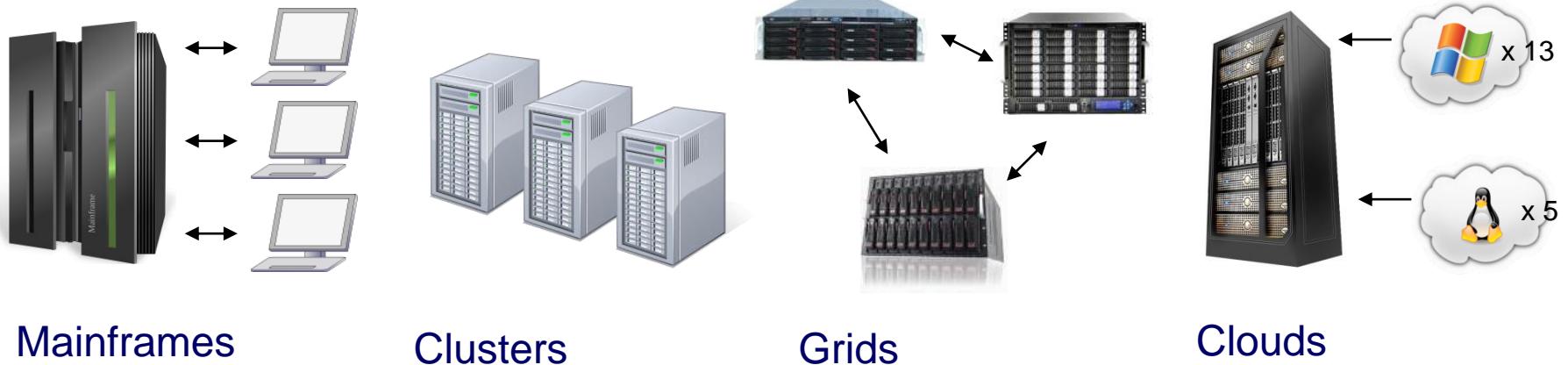
Internet Domain Survey Host Count



Source: Internet Systems Consortium (www.isc.org)

Source: www.isc.org/network/survey - now discontinued

The Evolution of Distributed Computing



Mainframes

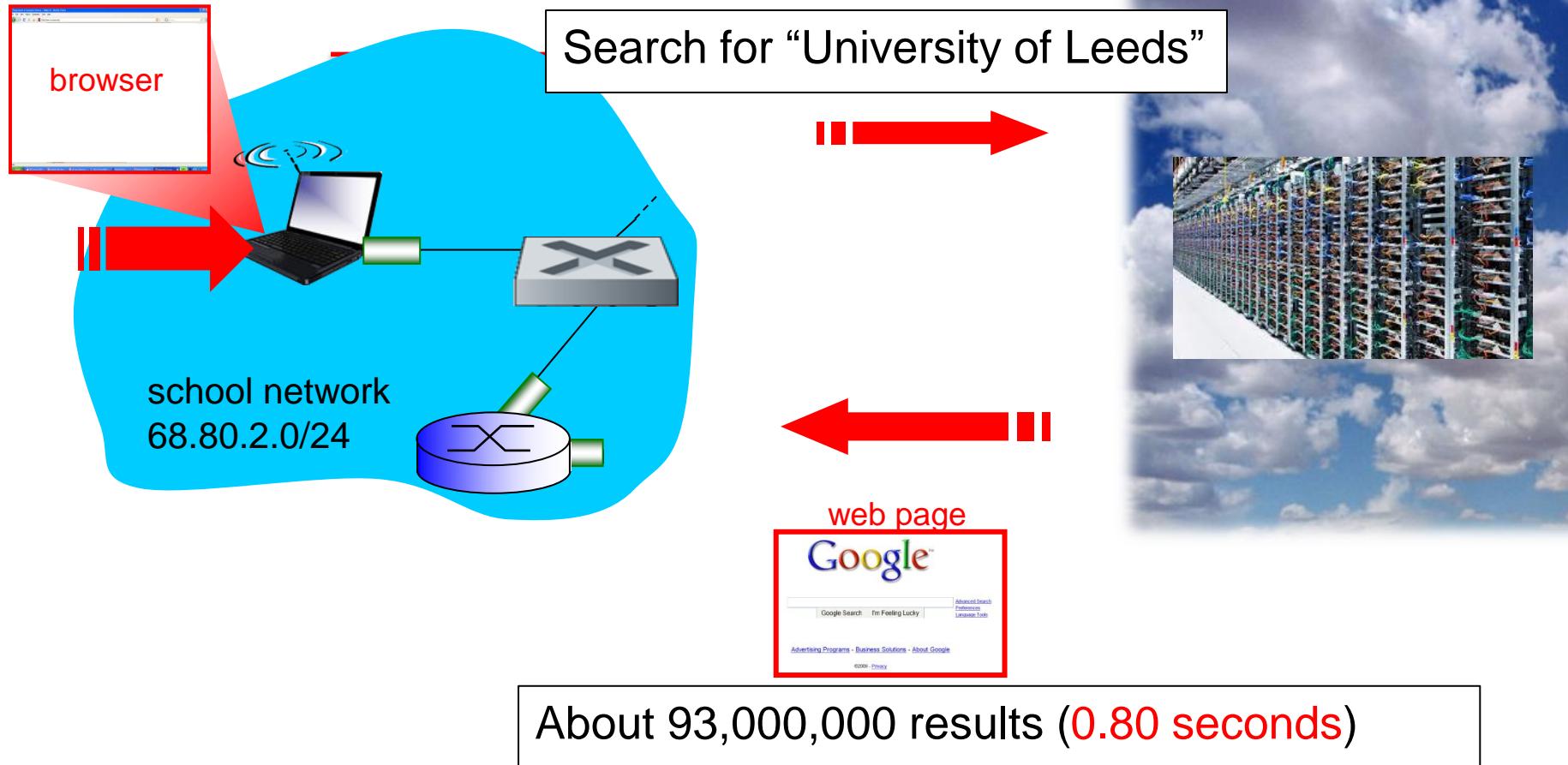
Clusters

Grids

Clouds

- Moving towards **Service Oriented Economy**
 - With new technological requirements
- Cloud Computing: “the next natural step in the evolution of **on-demand** information technology services...”
- Requires a paradigm shift to enable on-demand services
- **Question** : what made this possible?

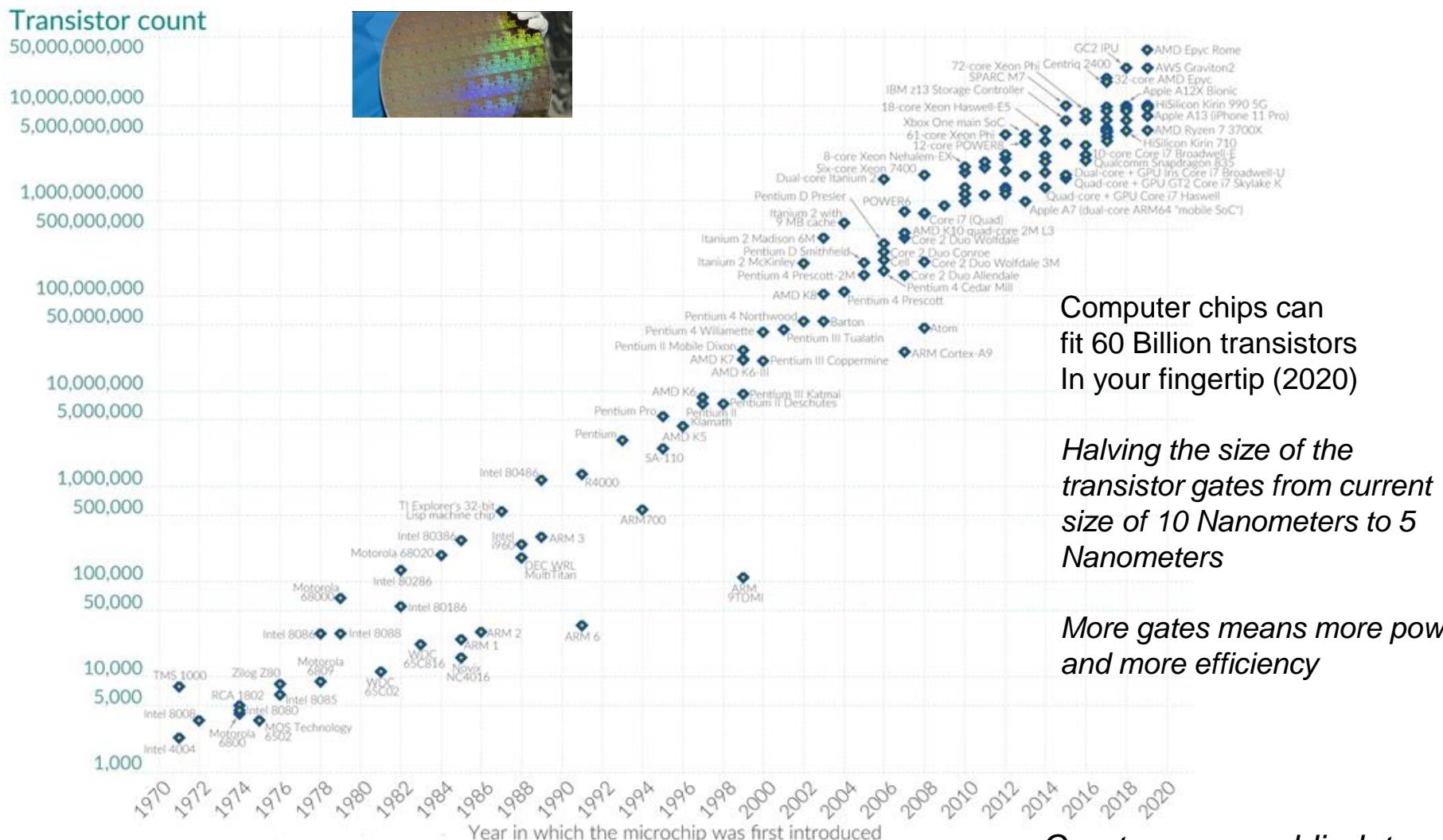
A Day in the Cloud: a Search on Google



WHAT made this possible? And **HOW**?

Living in an Exponential World: Moore's Law

- Exponential growth in the number of transistors per integrated circuit
 - predicted in 1965 by Gordon Moore that this trend would continue



Frontier: The First Exascale Supercomputer



- Breaks exascale barrier at 1.1 exaflops
- Full production January 2023
- See top500.org

Courtesy of Oak Ridge National Lab.

Living in an Exponential World: Computer Networks

- Network vs. computer performance
 - Computer speed doubles every 18 months
 - Network speed doubles every 9 months
- 1986 (64Kbs) → 2000 (2.5Gbps)
- 2001 → 2013 (40,160,320Gbps, 640Gbps)



Cisco Router CRS-X family

- 400 Gbps - one slot on the router's rack
- Each rack is scalable up to 6.4 Terabits per second
- Entire CRS-X system is capable of nearly 1 Petabit per second if multiple racks are set up in tandem.



Living in an Exponential World: Storage

- Storage density doubles every 12 months
- Dramatic growth in online data: 1 petabyte = 1000 terabyte = 1,000,000 gigabyte
 - 2000 ~0.5 petabyte
 - 2005 ~10 petabytes
 - 2010 ~100 petabytes
 - 2015 ~1000 petabytes = 1 Exabytes
 - Today: zetta era (10^{21} bytes)



How Much Data? – Daily Generation



15 Exabytes of
data stored +
processes 100 PBs
a day



40 PBs a year



300 PB of user
data + processes
600 TB/day



90 PB of data stored
+ processes
100PB/day



阿里云计算
Alibaba Cloud Computing

Note 1: these figures need to be revisited

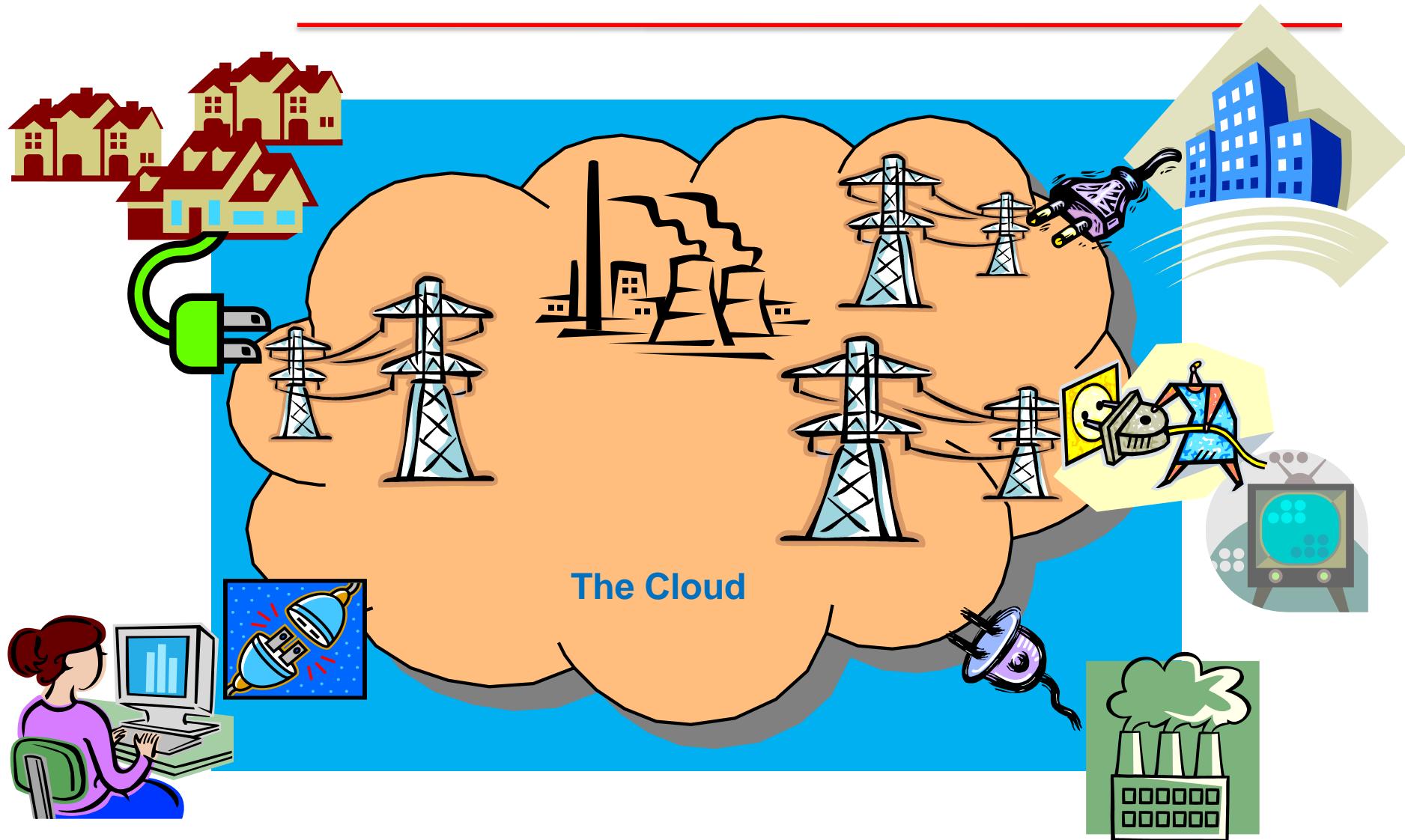
*Note 2: by 2025, the amount of data generated each day is expected to reach **400 exabytes** globally.*

Distributed Computing Paradigms

- First
 - We linked all machines together
 - Internet, TCP/IP
- Second
 - we linked all documents together
 - WWW, HTTP, HTML, XML
- Third
 - we linked all applications together
 - Web services, SOAP, WSDL, UDDI, REST
- Now
 - We are linking everything else together
 - Grid – decentralised infrastructures
 - Cloud – centralised infrastructures

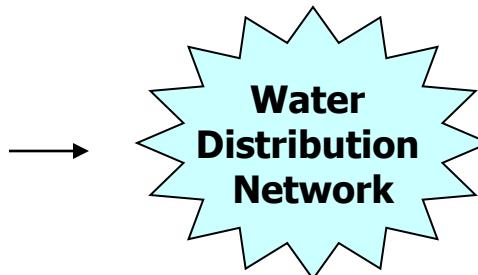
ICT Today : Power Grid Inspiration for Computing

Deliver ICT services as “computing utilities” to users

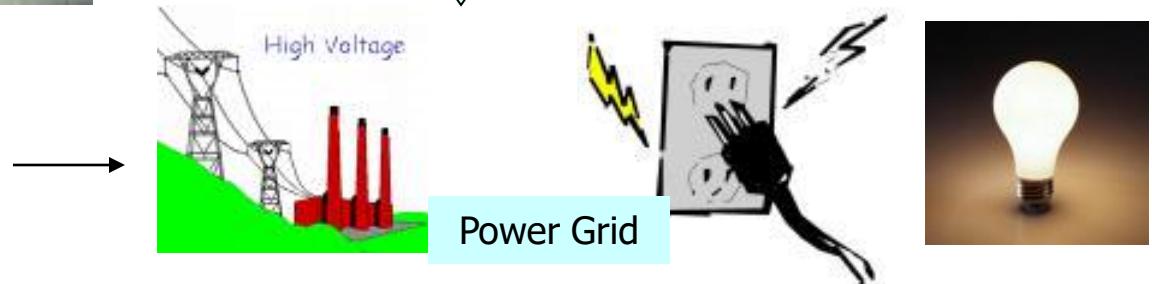


Today ... Essential Utilities and Delivery Networks

(1) Water



(2) Electricity



(3) Gas

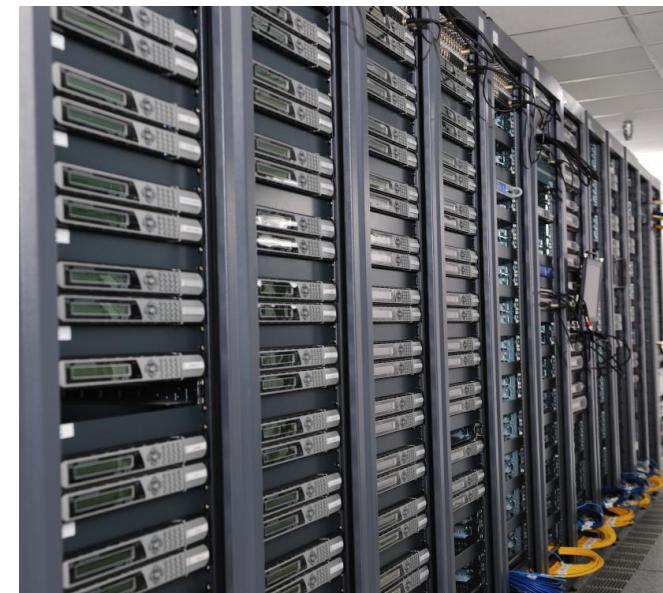


(4) Telephone



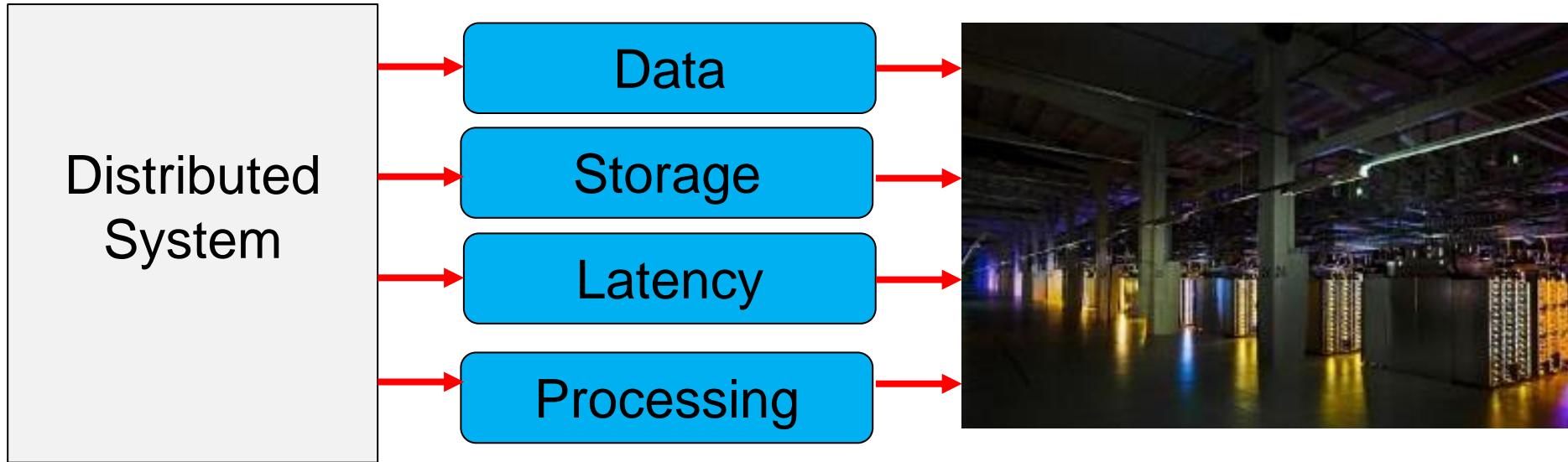
“Computer Utilities” Vision: Implications of the Internet

- 1969 – Leonard Kleinrock, ARPANET project
 - “As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of ‘computer utilities’, which, like present electric and telephone utilities, will service individual homes and offices across the country”
- Computers (and their role) Redefined
 - Proliferation of cloud data centres



Examples of a Distributed System

The Cloud



*Google Hamina,
Courtesy of Google*

Question 1: Biggest challenges cloud data centres are facing?

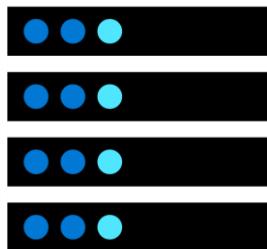
Question 2: Time to deploy underwater data centres?

<https://www.subseacloud.com/>

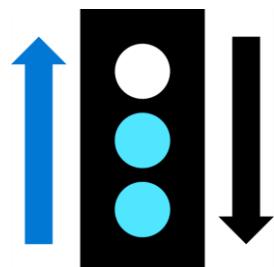
<https://news.microsoft.com/source/features/sustainability/project-natick-underwater-datacenter/>

Cloud computing – a Definition

- National Institute of Standards and Technology (NIST)
 - 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.



Compute



Networking



Storage

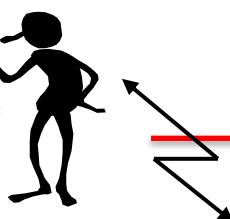


Analytics

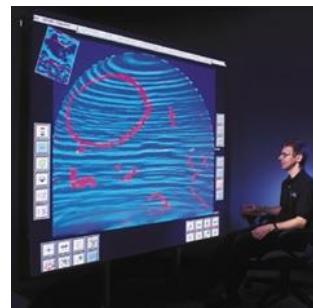
The Cloud Metaphor



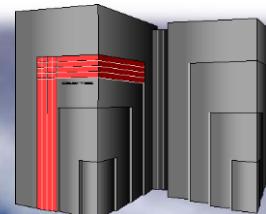
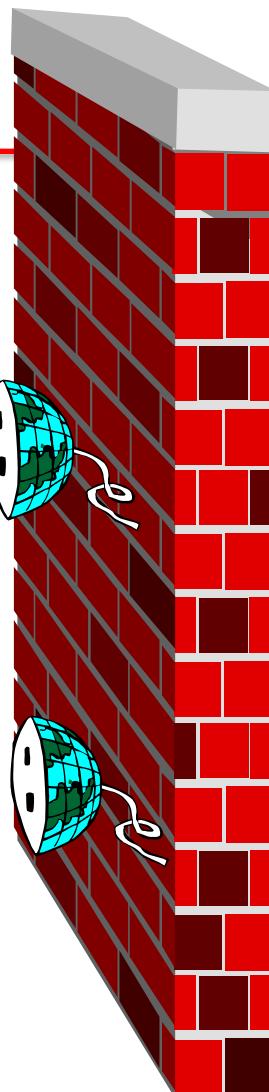
Mobile Access



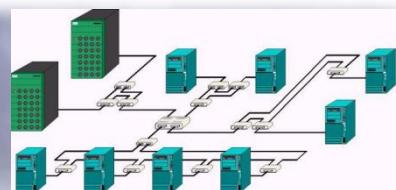
Workstation



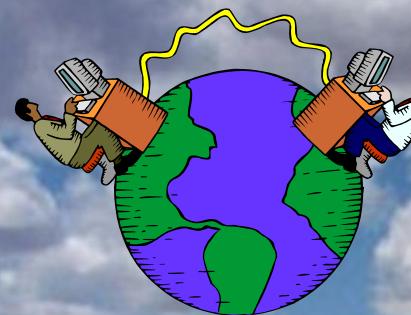
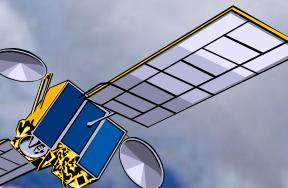
Application



Supercomputer, PC-Cluster



Data-storage, Sensors



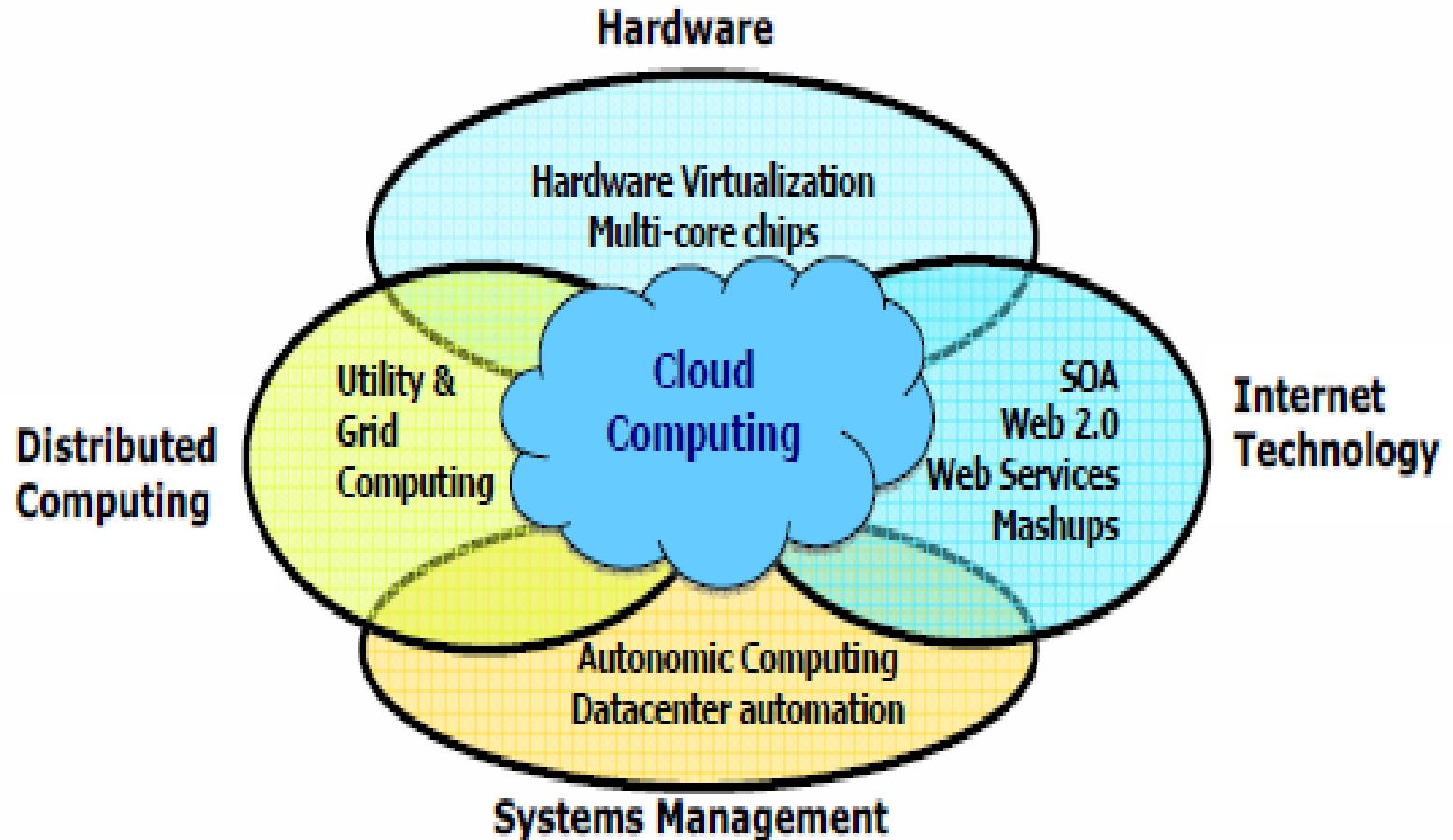
Internet, networks



Cloud computing – a Definition (2)

- The NIST definition lists **five** essential characteristics:
 1. On-demand self-service
 2. Broad network access
 3. Resource pooling
 4. Rapid elasticity or expansion
 5. Measured service
- It also lists **three** "service models" (software, platform and infrastructure), and **four** "deployment models" (private, community, public and hybrid) strategies – *more later*

New Challenges Enabled



The Cloud

- Historical roots in today's Internet applications
 - Search, email, social networks
 - File storage (Dropbox, Mobile Me, Flickr, ...)
- A cloud infrastructure provides a framework to manage scalable, reliable, **on-demand** access to applications
- A cloud is the “invisible” backend to many of our **mobile** applications



Cloud Data Centers

- 10's to 100's of thousands of hosts, often closely coupled, in close proximity:
 - e-business (e.g. Amazon)
 - content-servers (e.g., YouTube, Apple, Microsoft)
 - search engines, data mining (e.g., Google)
- ❖ challenges:
 - multiple applications, each serving massive numbers of clients
 - managing/balancing load, processing, networking, data management



Inside a 40-ft Microsoft container,
Chicago data center

See <http://www.google.co.uk/about/datacenters/>

Virtualised Infrastructures

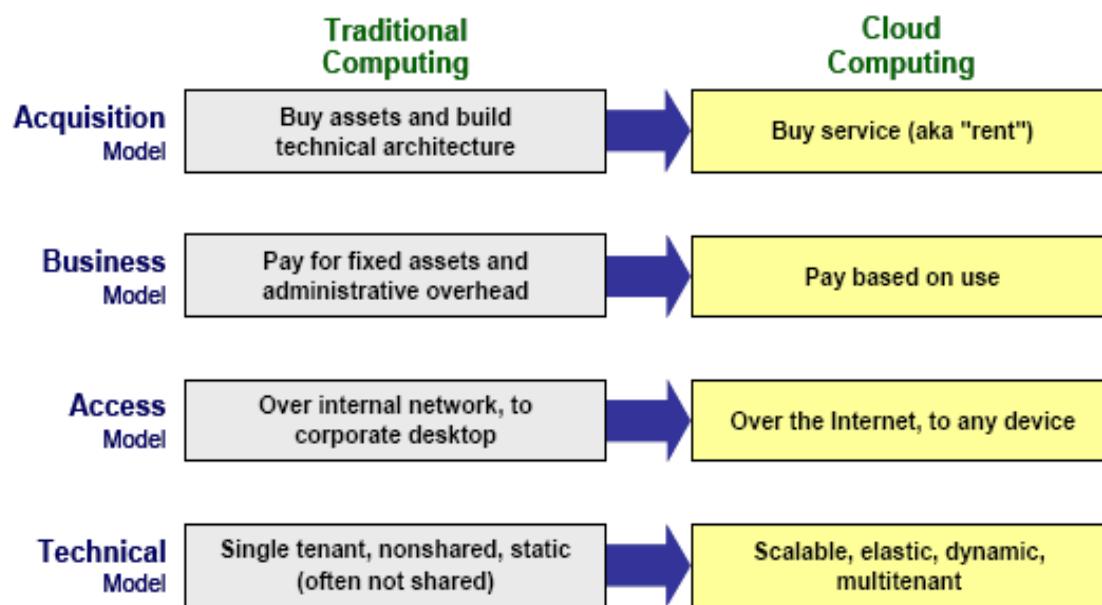
- **Virtualized Service Platforms**
 - Consolidation of server systems
 - Reduced costs
 - Reduced complexity
 - Simplified administration
 - Pay-per-usage
- **Key technology: virtualization**



Virtualised Infrastructures



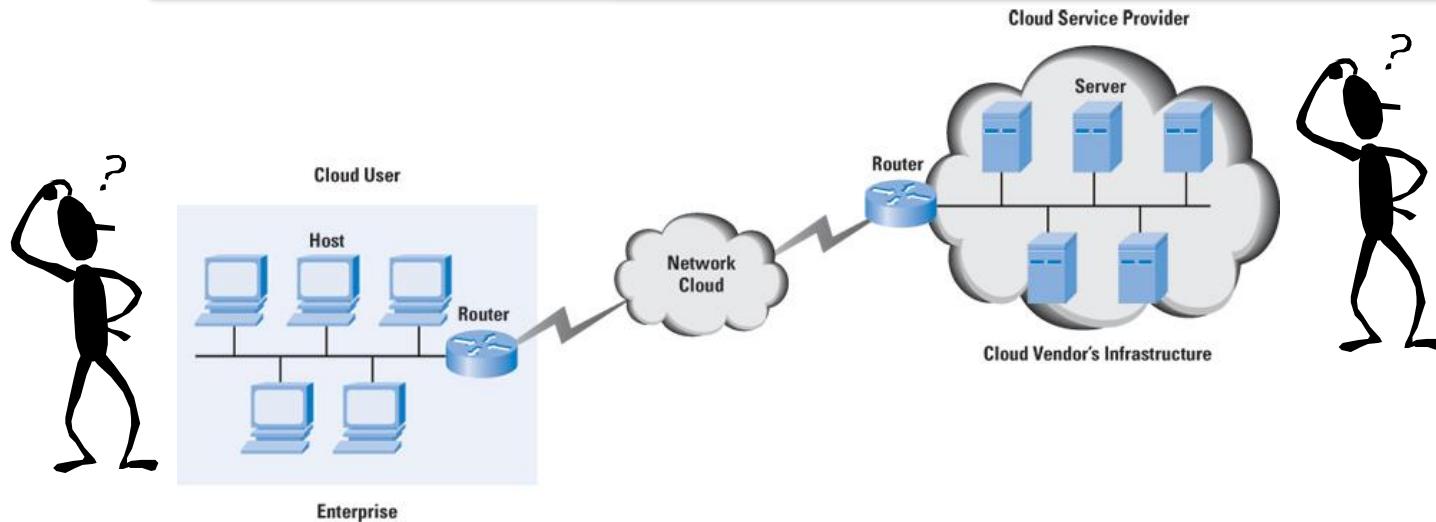
- An emerging computing paradigm where data and services reside in massively scalable data centers and can be ubiquitously accessed from any connected device over the Internet



Source: Gartner (September 2008)

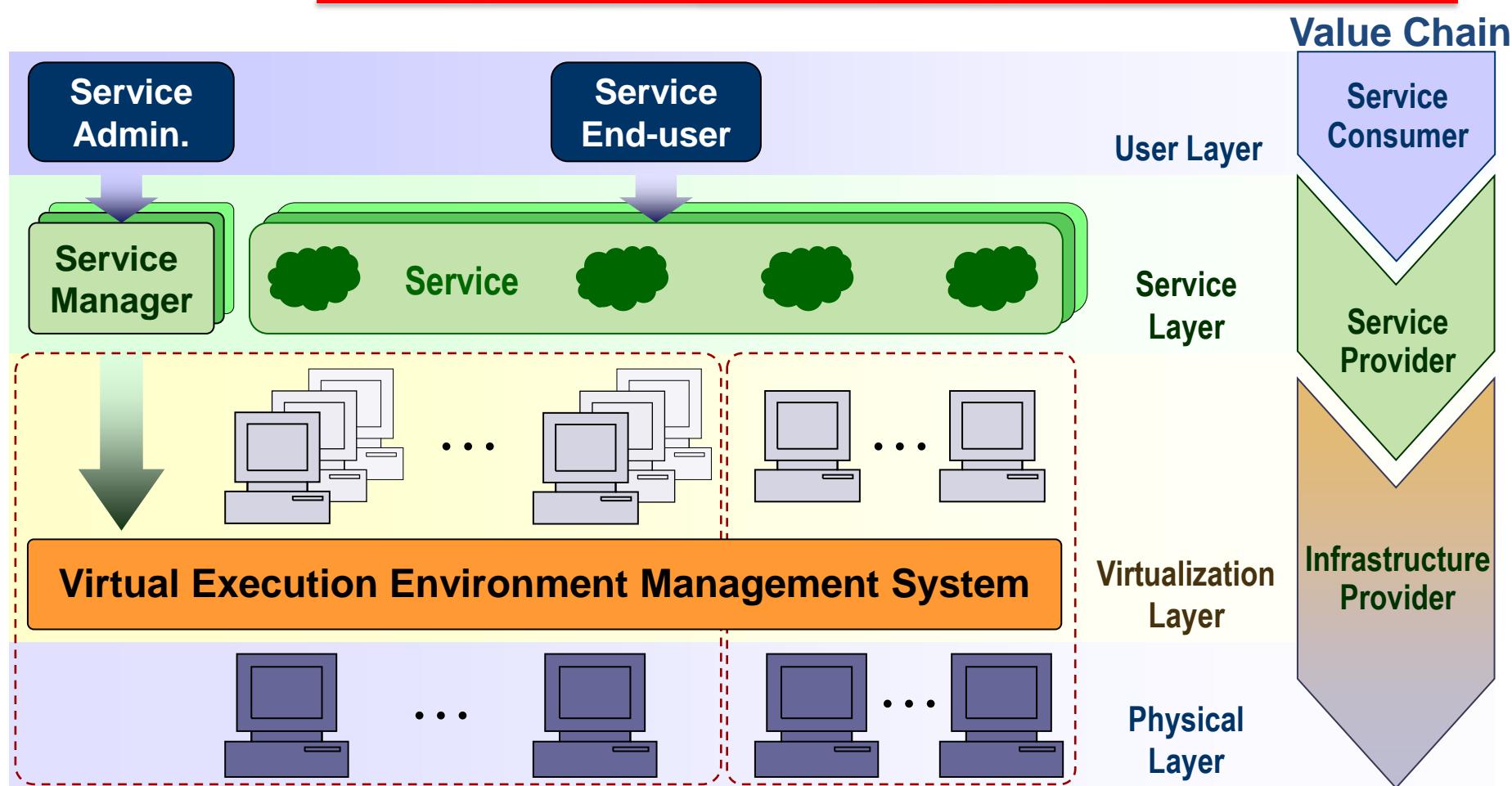


Realising the ‘Computer Utilities’ Vision: What Consumers and Providers Want?

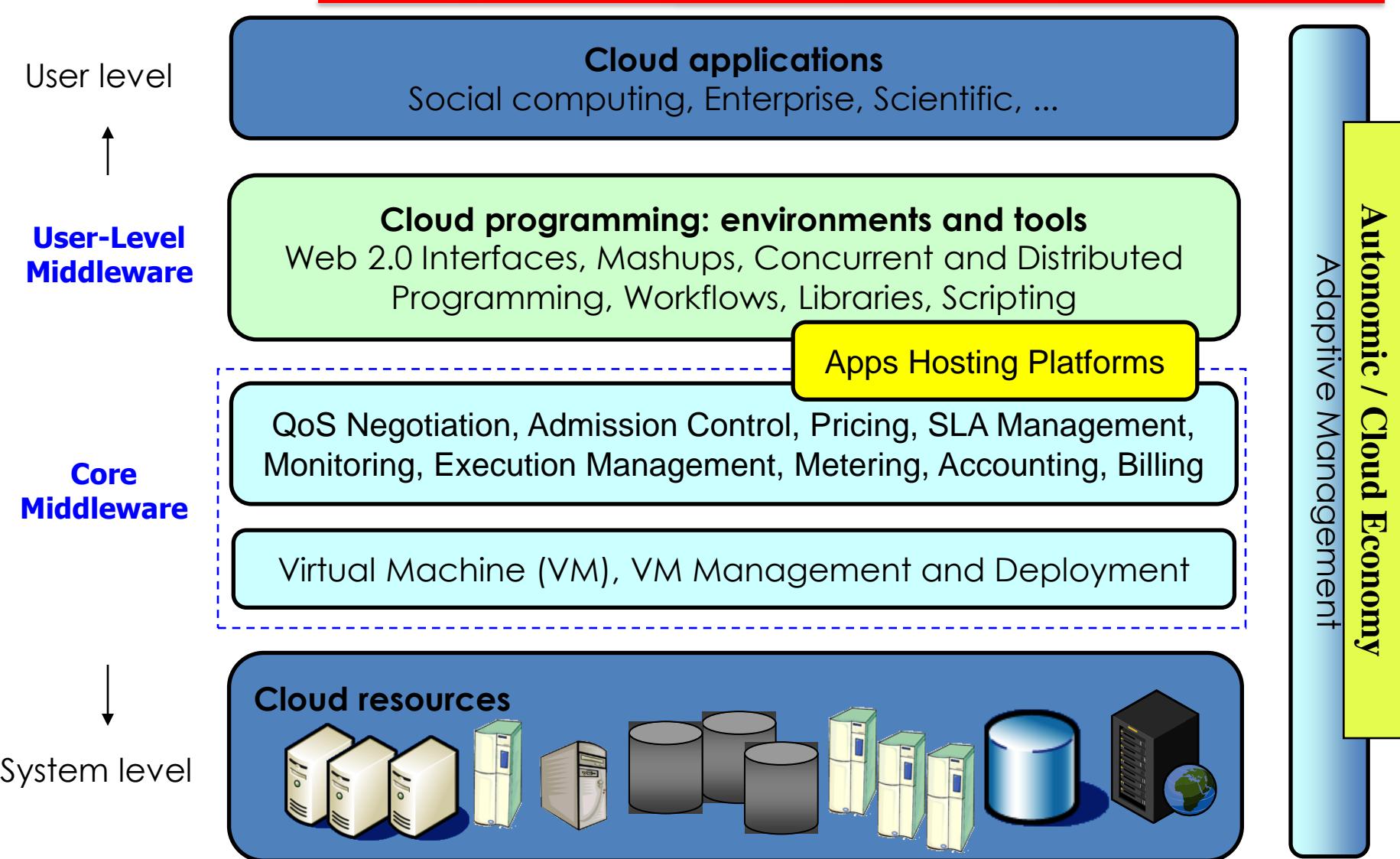


- Cloud Service Consumers
 - Meet Quality of Service
 - Minimise expenses
- Cloud Service Providers
 - Attract customers
 - Maximise profit

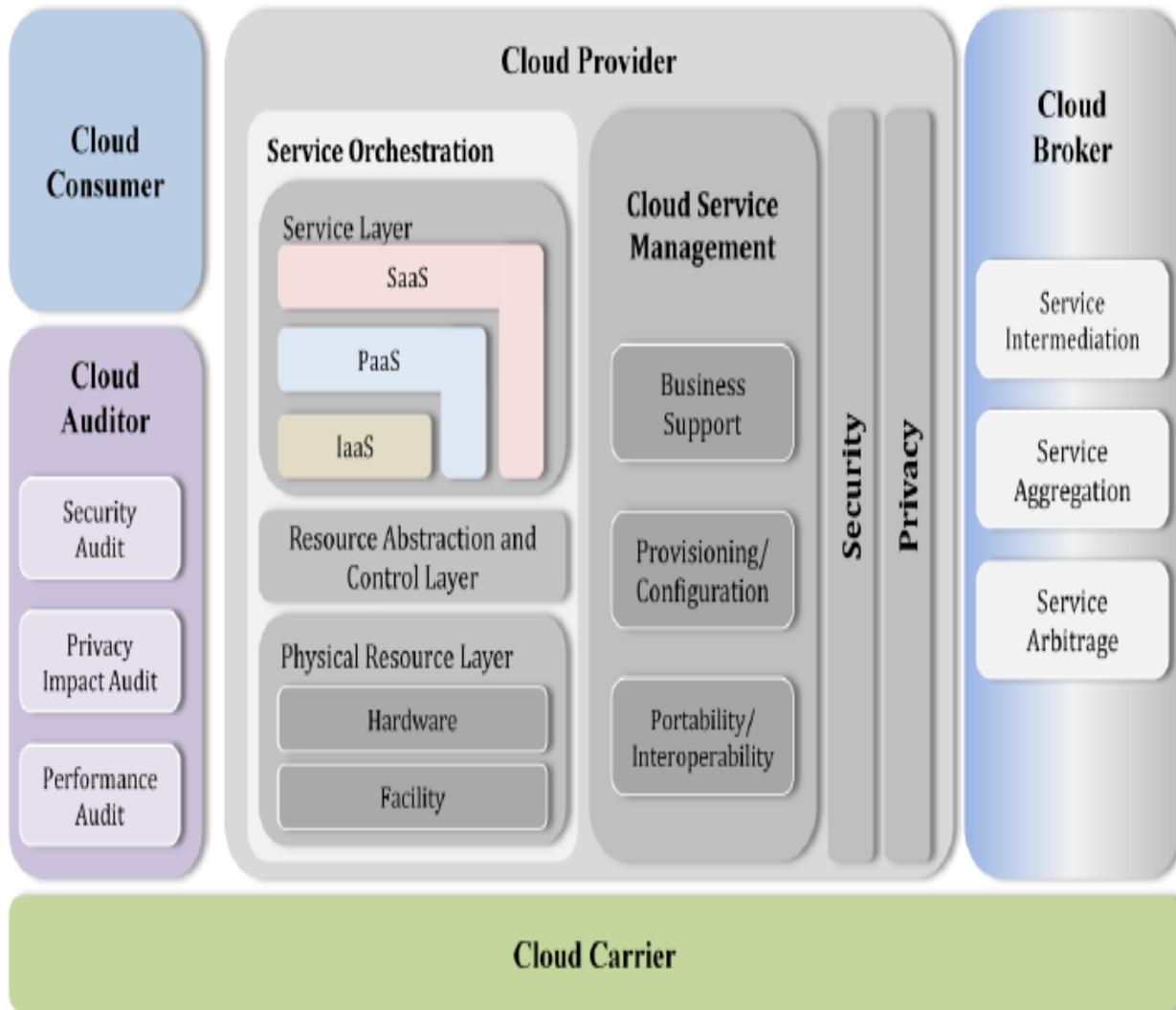
The Vision



A (Layered) Cloud Architecture



NIST Cloud Computing Reference Architecture



Benefits of Cloud Computing

- Question 1: can you identify the **technological** benefits of cloud computing
 - 1. .
 - 2. .
 - 3. .
 - 4. .
 - 5. .
- Question 2: what are the **economic** benefits?
 - 1. .
 - 2. .
 - 3. .
 - 4. .
 - 5. .

Conclusion

- Several Computing Platforms/Paradigms are promising to deliver “Computing Utilities” vision
 - Cloud Computing is the most recent kid in the block promising to turn vision into reality
 - Clouds built on: SOA, VMs, Web 2.0 technologies
 - Many exciting business and consumer applications enabled.
- The Cloud is **not a silver bullet** that can take any application and run it a 1000 times faster
 - Some applications may not be suitable to run on a cloud

References

- Cloud Computing for Science and Engineering, I. Foster and D. B. Gannon. MIT Press, 2017
- K. Hwang, G. Fox, and J. Dongarra, Distributed and Cloud Computing: from Parallel Processing to the Internet of Things Morgan Kauffmann Publishers, 2011
- A Manifesto for Future Generation Cloud Computing: Research Directions for the Next Decade. R. Buyya et al., ArXiv, 2017
<https://arxiv.org/abs/1711.09123>
- The NIST Definition of Cloud Computing. P. Mell and T. Grance, September 2011
<http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>
- S. Bhowmik, Cloud computing. Cambridge University Press, 2017

COMP5850 – Cloud Computing



Cloud Platform Architecture over Virtualised Datacenters

Plan of the Lecture

Goals

- Understand concepts of cloud platform architectures

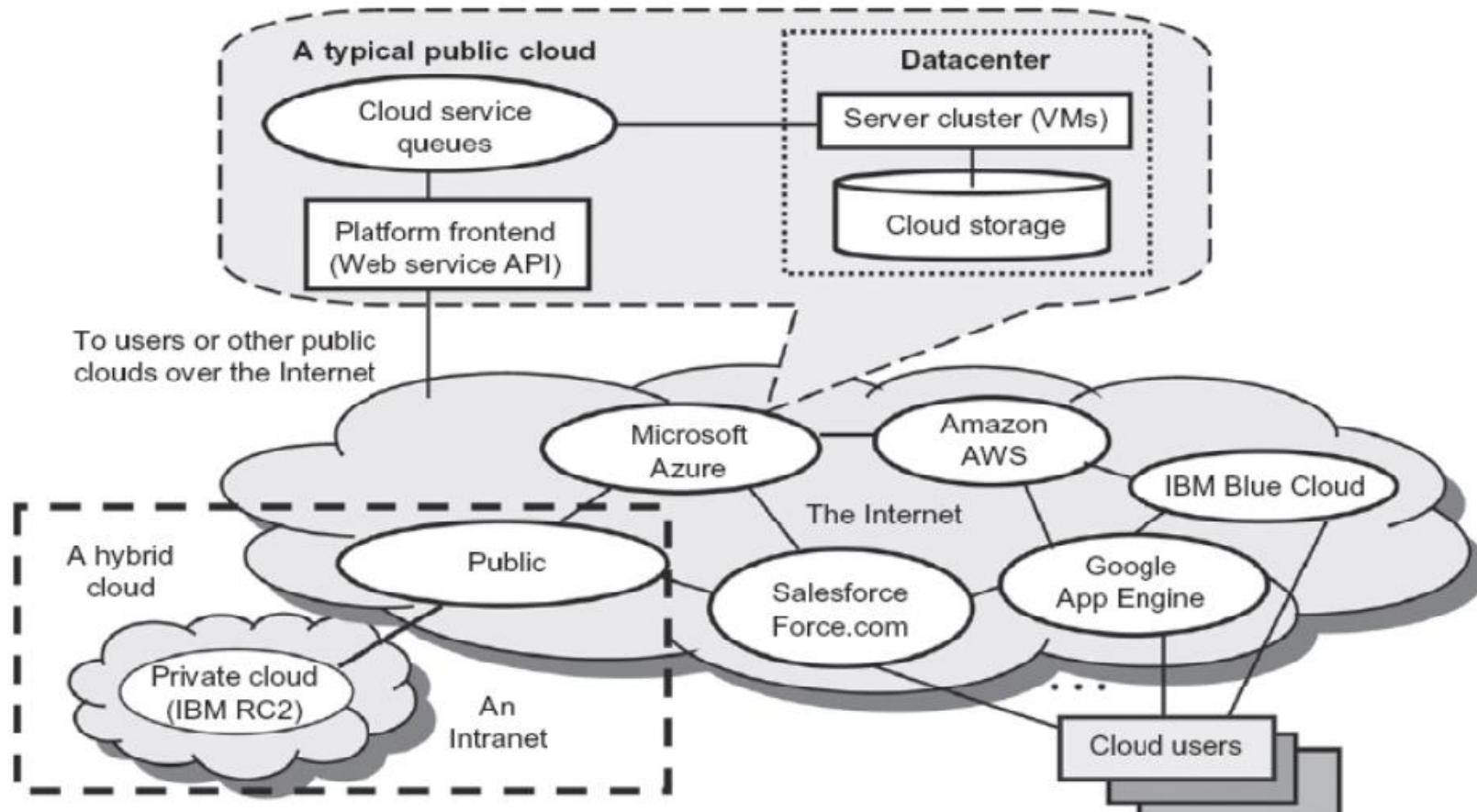
Overview

- Quick Recap.: Private, Public, and Hybrid Clouds
- Cloud computing as a service
- Data centre design
- Warehouse scale computing
- Cloud Architecture: System Level
 - Compute
 - Storage
 - Network
- Aspect of heterogeneity
- Conclusion

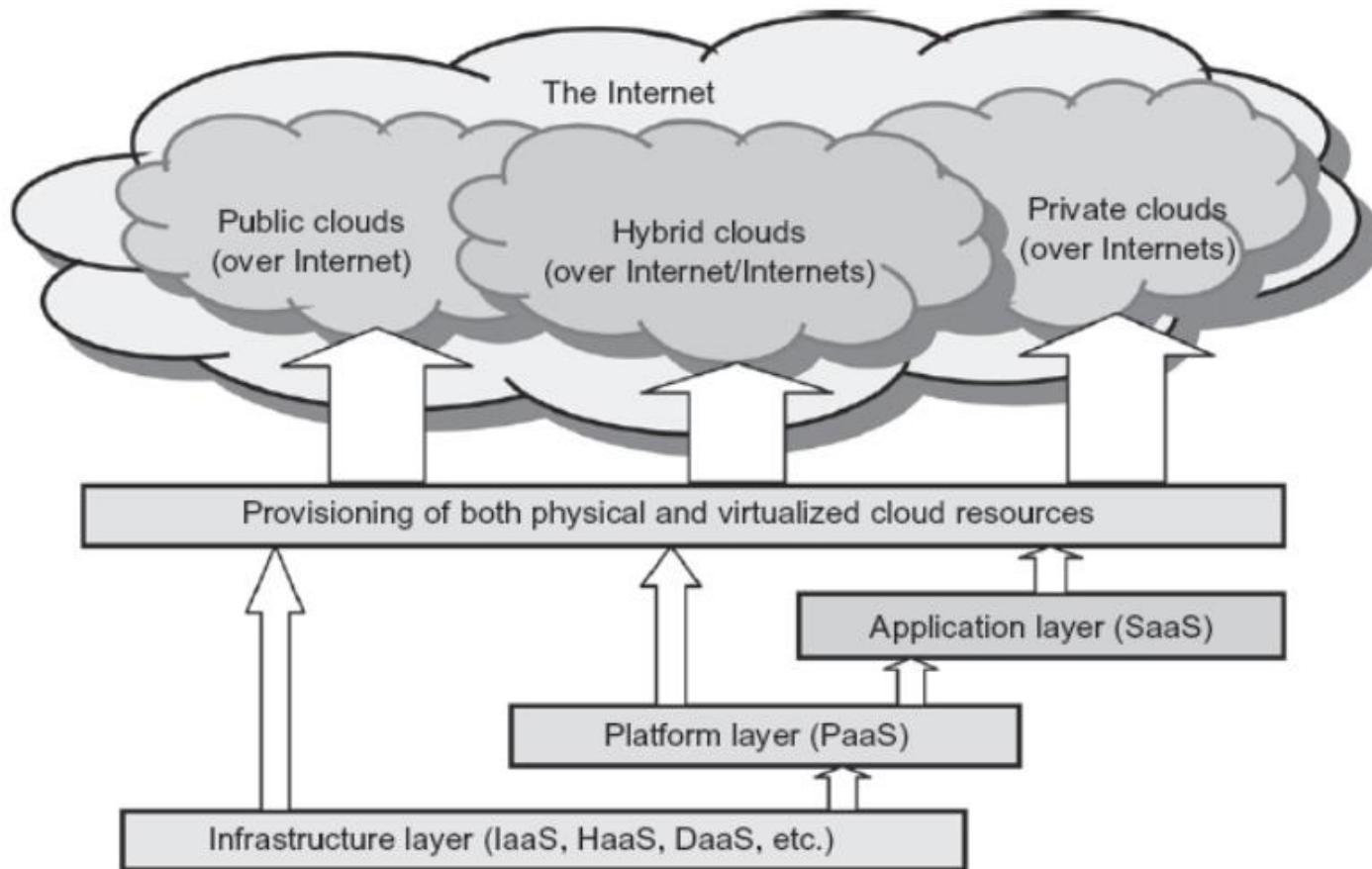
Three Aspects in Hardware that can benefit from Cloud Computing

1. The illusion of infinite computing resources available **on demand**, thereby eliminating the need for cloud users to plan far ahead for resource provisioning
2. The elimination of an up-front commitment by cloud users, thereby allowing them to start small and **increase** hardware resources when needed in the future.
3. The ability to pay the costs of computing resources on a **short-term** basis as needed (e.g., processors by the hour and storage by the day) and release them after done and thereby rewarding resource conservation.

Private, Public, and Hybrid Clouds

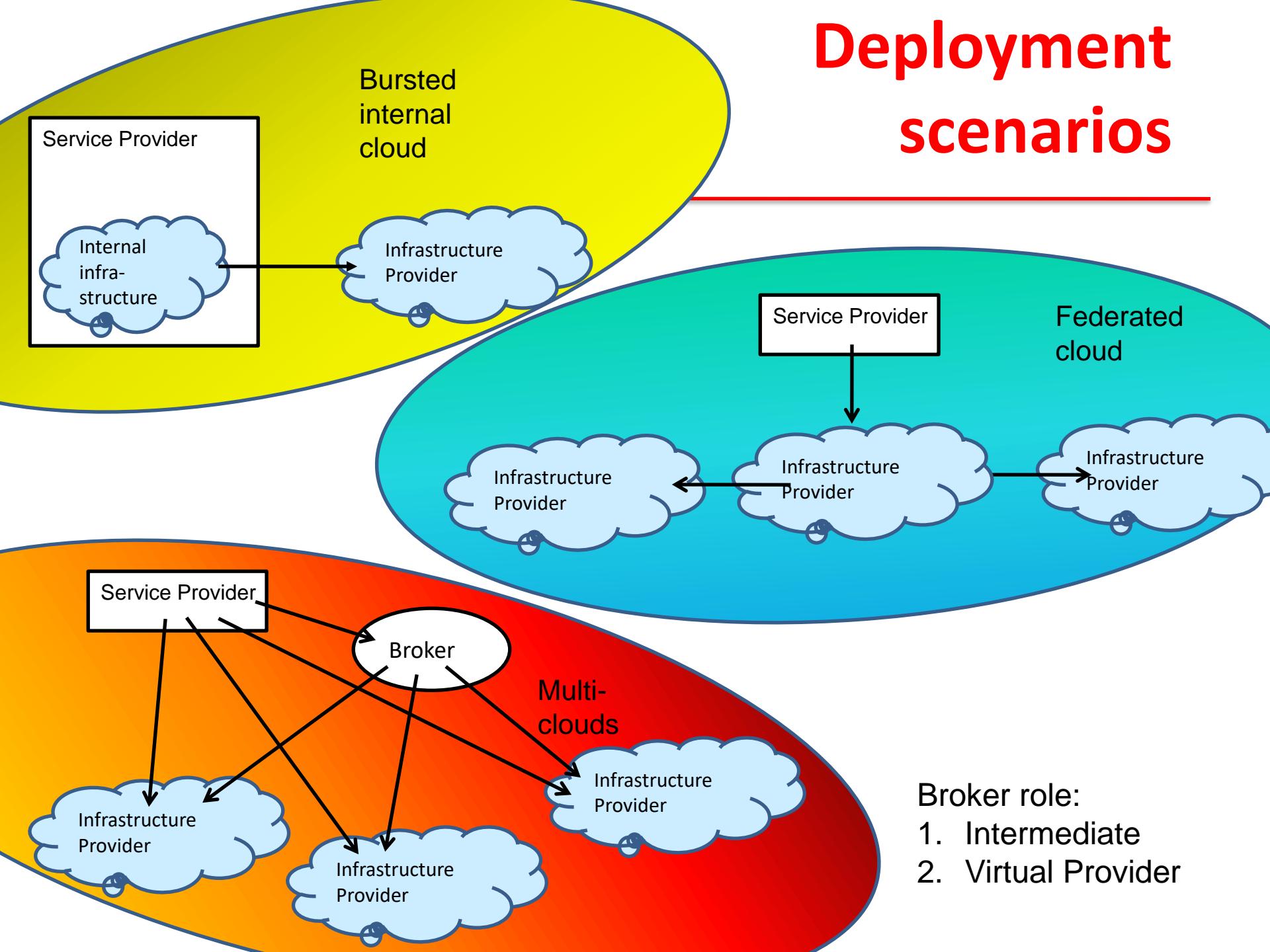


Cloud Computing as A Service



Layered architectural development of the cloud platform for IaaS, PaaS, and SaaS applications over the Internet.

Deployment scenarios



Cloud Resources: System Level

User level

Cloud applications



User-Level
Middleware

Cloud programming: environments and tools

Core
Middleware

Apps Hosting Platforms

QoS Negotiation, Admission Control, Pricing, SLA Management, Monitoring, Execution Management, Metering, Accounting, Billing

Virtual Machine (VM), VM Management and Deployment

System level

Cloud resources



Clouds built on Massive Datacenters

Range in size from “edge” facilities to megascale (100K to 1M servers)

Economies of scale

- Approximate costs for a small size center (1K servers) and a larger, 400K server center.



Technology	Cost in small-sized Data Center	Cost in Large Data Center
Network	\$95 per Mbps/ Month	\$13 per Mbps/ month
Storage	\$2.20 per GB/ Month	\$0.40 per GB/ month
Administration	~140 servers/ Administrator	>1000 Servers/ Administrator



This data center is
11.5 times
the size of a football field

Warehouse Scale Computing (WSC)

- Provides Internet services
 - Search, social networking, online maps, video sharing, online shopping, email, cloud computing, etc.
- Differences with HPC “clusters”
 - Clusters have higher performance processors and network
 - Clusters emphasize **thread-level** parallelism, WSCs emphasize **request-level** parallelism
- Datacenters
 - Consolidate different machines and software into one location
 - Emphasise virtual machines and hardware heterogeneity in order to serve varied customers

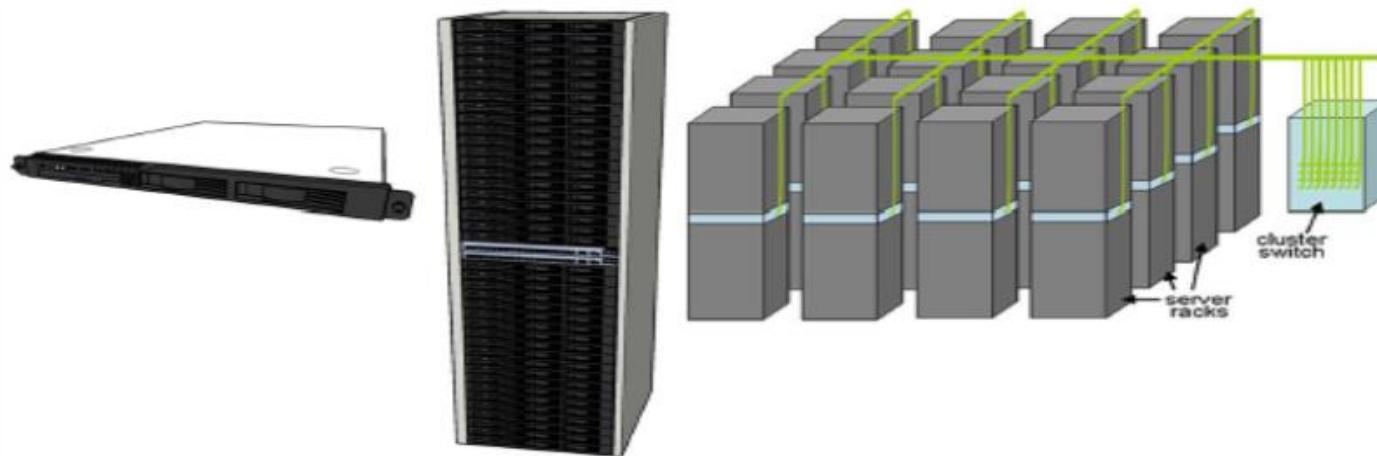
Design Considerations for WSC

- Cost-performance
 - Small savings add up
- Energy efficiency
 - Affects power distribution and cooling
- Dependability via redundancy
- Network I/O
- Interactive and batch processing workloads
- Operational costs count
 - Power consumption is a primary constraint when designing system
- Scale and its opportunities and problems



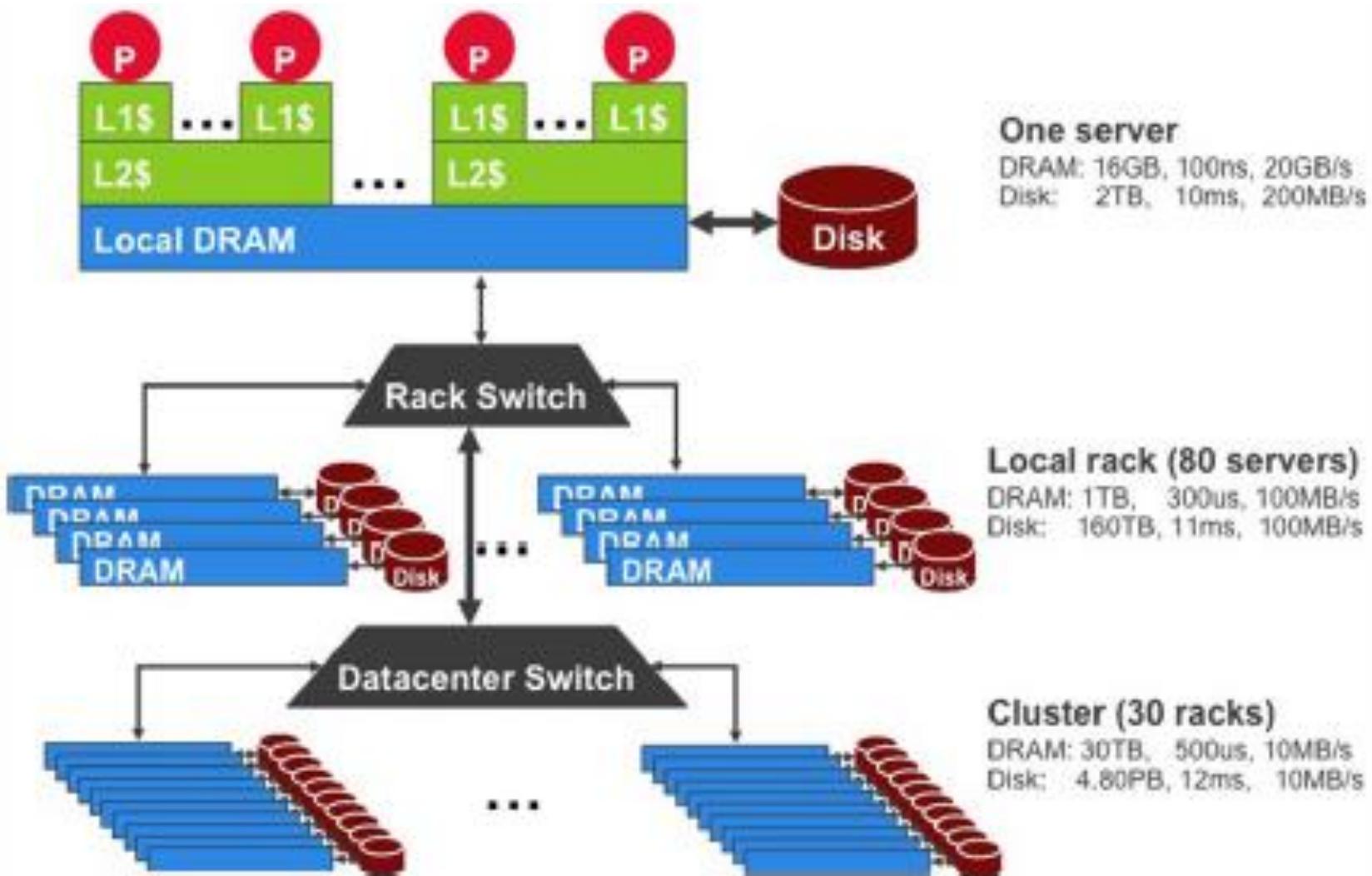
The Architecture of a Small Server Cluster (~ 1000 servers)

Interconnected by an Ethernet switch and housed in a warehouse or in a container environment



Typical elements in warehouse-scale systems: 1U server (left), 7' rack with Ethernet switch (middle), and diagram of a small cluster with a cluster-level Ethernet switch/router (right).

Servers, Racks, Clusters

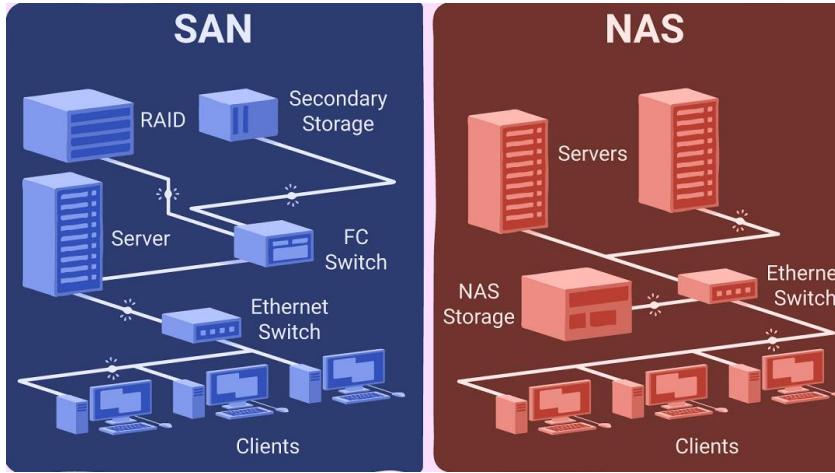


Larger Datacenter Growth

- One at a time:
 - 1 system
 - Racking & networking
- Rack at a time:
 - ~ 40 systems
 - Install & networking
- Container at a time:
 - ~1,000 systems
 - No packaging to remove
 - No floor space required
 - Power, network, & cooling only
- Weatherproof & easy to transport
- Datacenter construction takes 24+ months
 - Both new build & DC expansion require regulatory approval



Storage and Array Switch



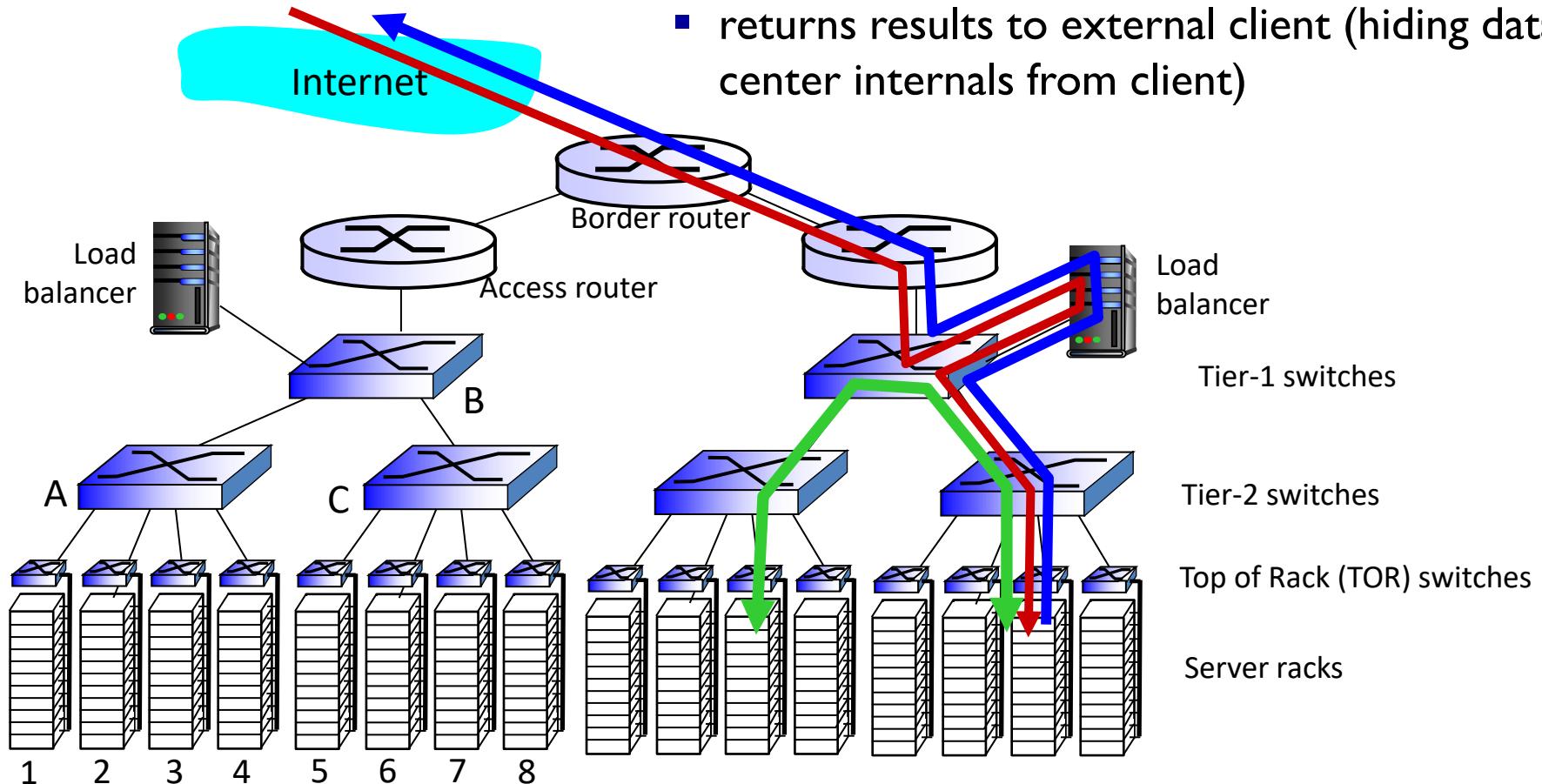
- Storage options:
 - Storage Area Network (**SAN**): local network that uses Fibre Channel to connect several data storage devices
 - Network Attached Storage (**NAS**): dedicated hardware device that connects to a local area network, usually through an Ethernet connection and operates on data files
- WSCs generally rely on local disks in servers
- Google File System (GFS) uses local disks and maintains at least three replicas.

Data Center Networks

load balancer: application-layer routing

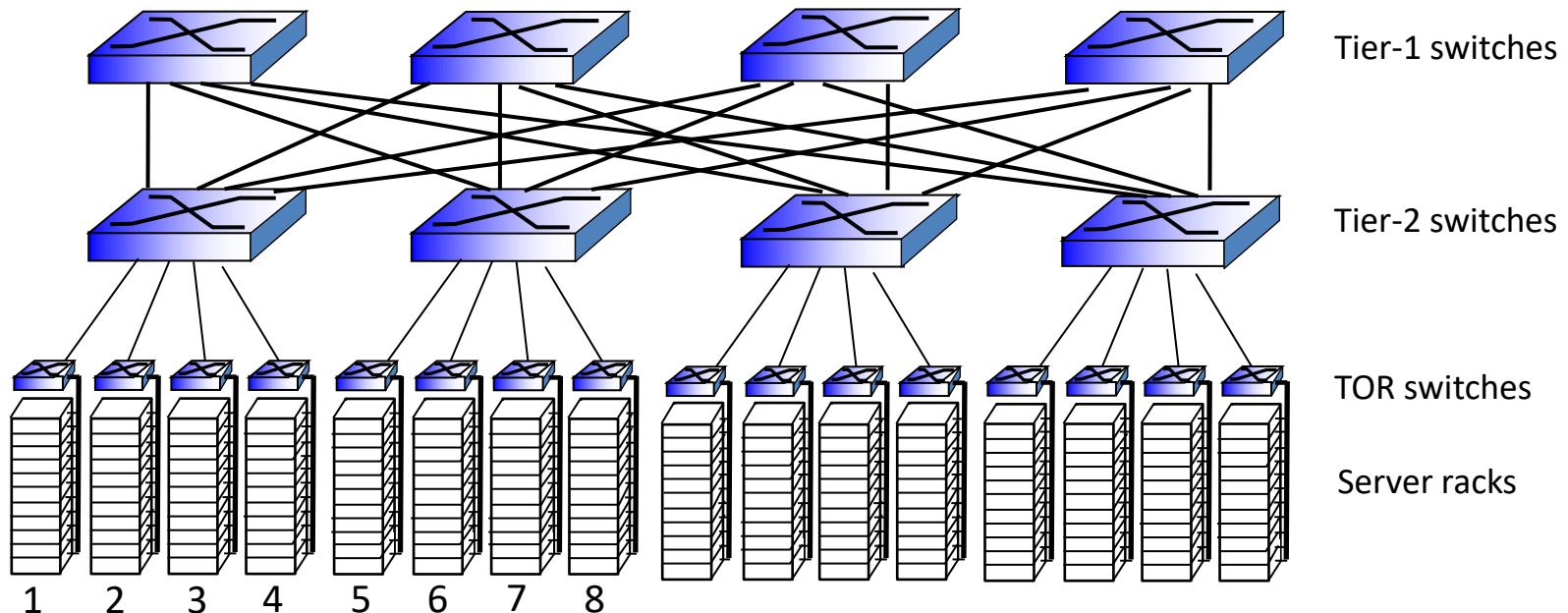
- receives external client requests
- directs workload within data center
- returns results to external client (hiding data center internals from client)

Query, e.g. Search



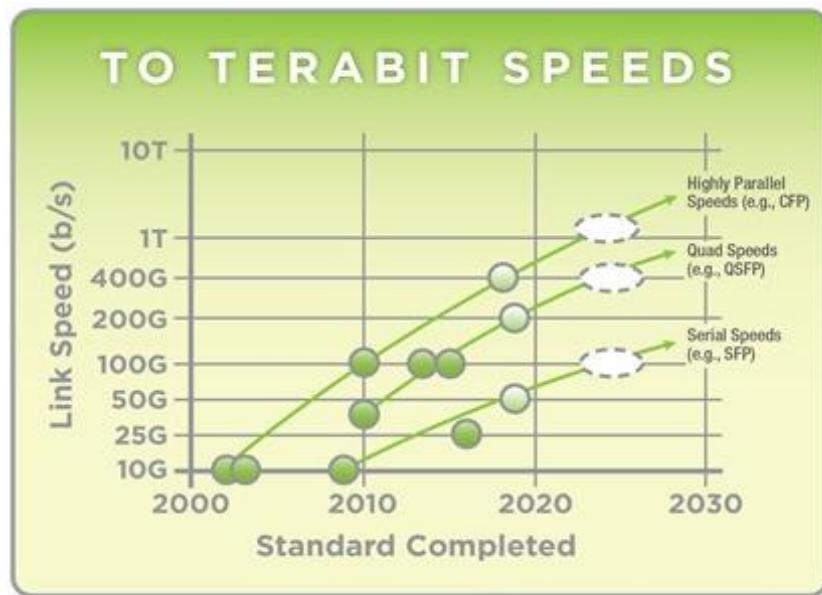
Data Center Networks

- ❖ rich interconnection among switches, racks:
 - increased throughput between racks (multiple routing paths possible)
 - increased reliability via redundancy

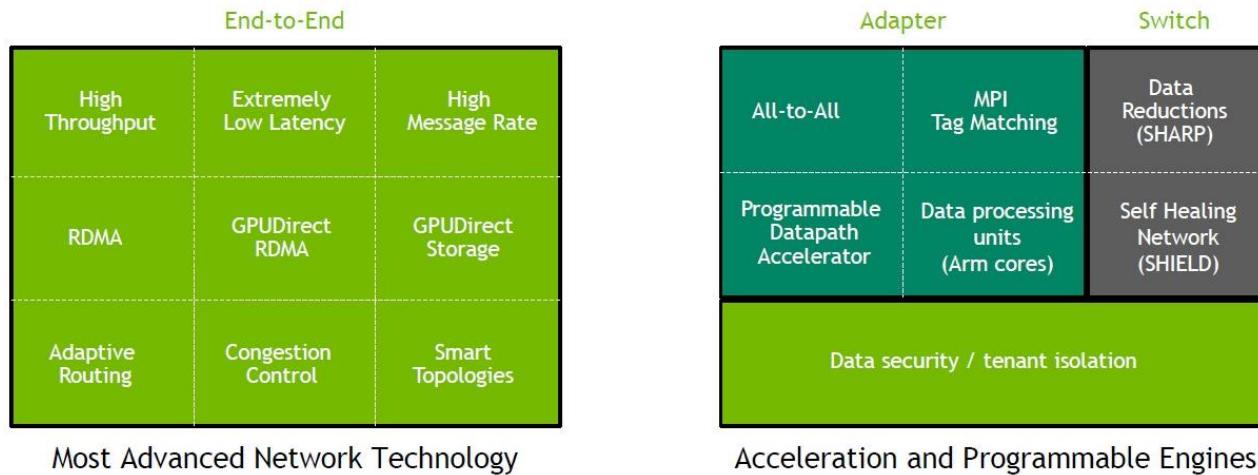


Ethernet - Trend

- Data center optics are rapidly transitioning to **400GE rates**
- Key requirements to handle the interfaces
 - New silicon and circuits e.g., linear broadband amplifiers and drivers
 - Adaptive digital equalizers
 - Development and ratification of standards
- Data Center Interconnect (DCI) ecosystem to move to **800GE rates**
- Bandwidth and performance demands on the network **will continue to accelerate**
 - **1.6Tb/sec?**



Infiniband – Software-Defined, Hardware Accelerated Network

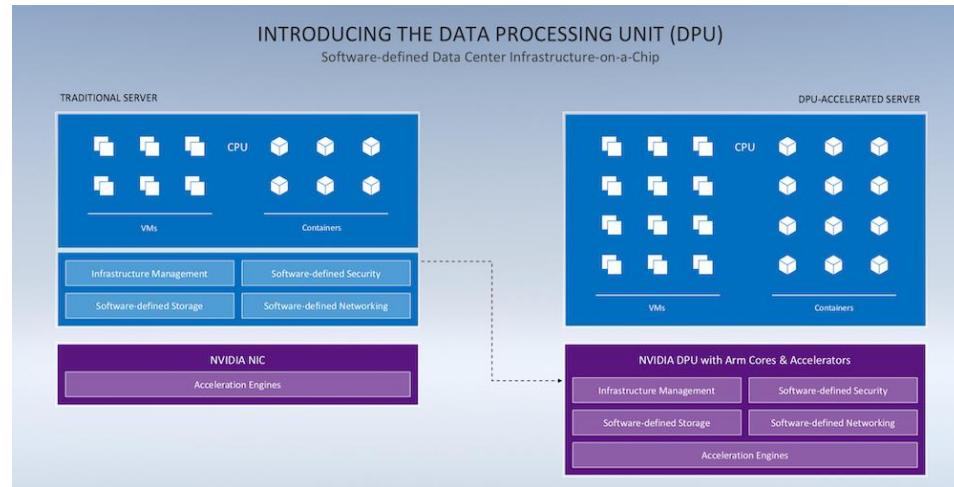


- **Move-off** work from general purpose CPU cores where application code runs (or manage the code offloaded to GPU accelerators) onto network devices
 - the switch **ASIC** (Application Specific Integrated Circuit itself), the network interface ASIC, or a full Data Processing Unit (**DPU**)
- DPU is a new part of the InfiniBand stack:
 - Is a smart NIC –has onboard CPU and GPU capabilities
 - Can virtualise networking and storage for the host
 - Can run security software without putting that burden on the host CPUs.

Infiniband – Software-Defined, Hardware Accelerated Network

In-Network Acceleration

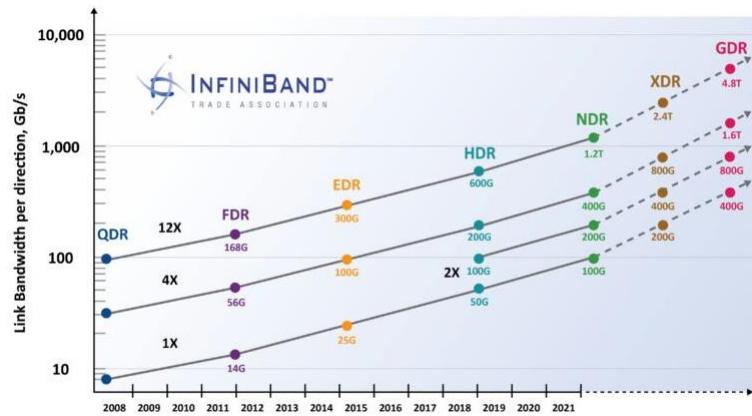
- **Bluefield DPU (application)**
 - collective operations, and offload active message processing, **smart MPI** (Message Passing Interface) progression, data compression, and user-defined algorithms
- **Scalable Hierarchical Aggregation Protocol (SHARP) (network)**
 - for data reductions inside the network
 - Developed for supercomputers, e.g. “Summit” (see top500.org)



Note: SHIELD is another innovative interconnect technology that improves data center fault recovery (self-healing).

Infiniband – What Next

- Today: **HDR (200Gb/s)**, **EDR (100Gb/s)** and **FDR (56Gb/s)**
- Like Ethernet, as bandwidth rates increase, forward error correction needs to compensate for higher bit error rates during transmission
 - latency across the switch will stay flat — at best — and will likely increase with each generation of technology

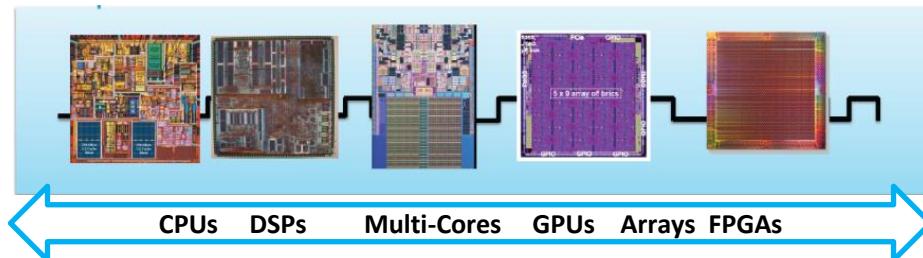


What next?

- **NDR**, with **400 Gb/sec ports** using four lanes — implemented by **NVIDIA with its Quantum-2 ASICs**
 - Deliver **64 ports at 400 Gb/sec (25.6 Tb/sec)**
- **XDR speed**, delivering **800 Gb/sec per port**
 - Projection: **1.6 Tb/sec**
 - 64 ports at 1.6Tb/sec = **102.4 Tb/sec**

Heterogeneity Aspects

- **Heterogeneity** has emerged as one of the most profound and challenging characteristics of parallel environments.
- Levels are identified:
 - **Macro** level: networks of distributed computers (clouds), composed by diverse node architectures (single, multi-core), are interconnected with potentially heterogeneous networks
 - **Micro** level: deeper memory hierarchies (main, cache, disk storage, tertiary storage) and various accelerator architectures (fixed, programmable, e.g. GPUs, and reconfigurable, e.g. FPGAs)



- **Other**: Software: OS, middleware, tools, ...

Computer architectures – classified regarding purpose

22



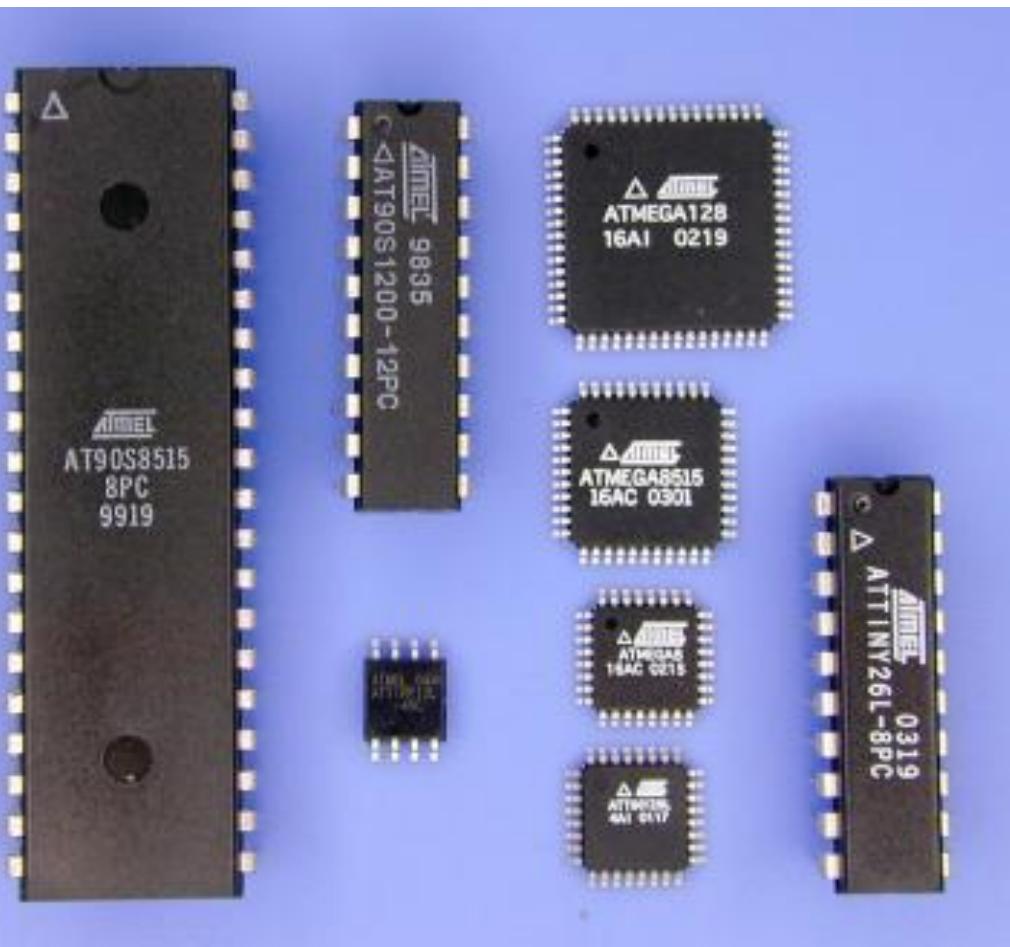
General Purpose Processors

23

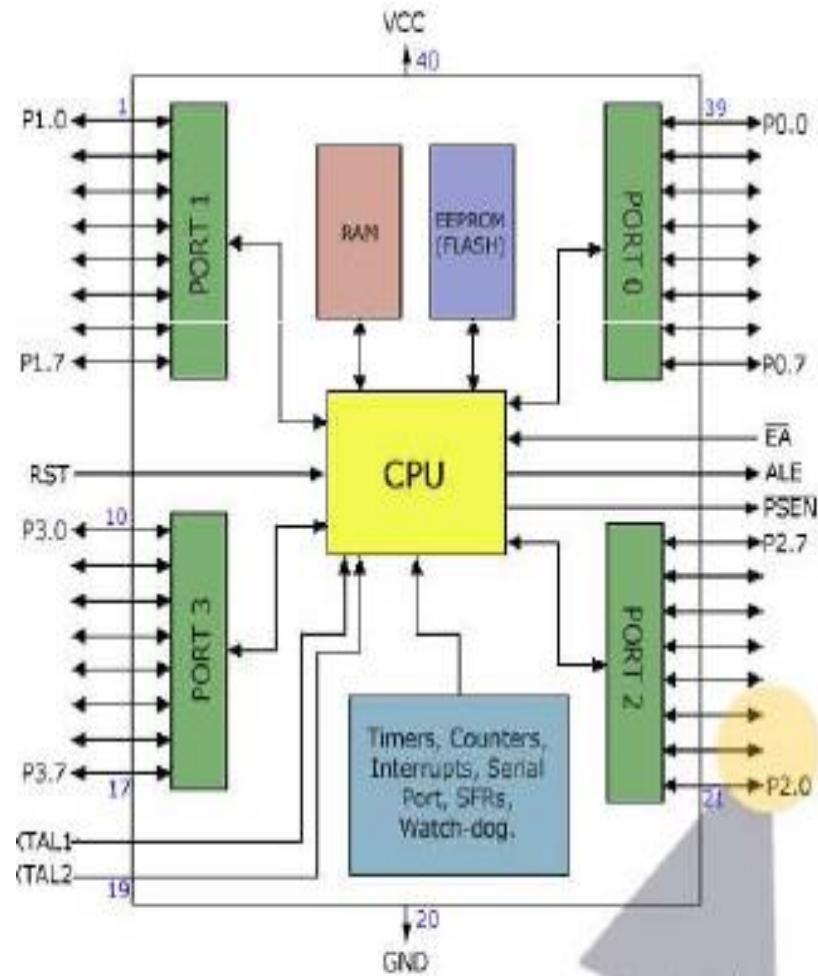
- General Purpose Processors (GPP)
 1. General Purpose Microprocessors - general purpose computers
 - Designed for general purpose computers such as PCs, workstations, Laptops, notepads etc.
 - Execute multiple applications and perform multiple tasks
 2. Microcontrollers - Embedded systems
 - Designed for specific tasks in embedded systems
 - Have control oriented peripherals
 - Have on chip CPU, fixed amount of RAM, ROM, I/O ports
 - Low cost, low performance, low power, smaller than microprocessors
 - Appropriate for applications in which cost, power and space are critical

General Purpose Processors – Microcontrollers (2)

24



Microcontrollers



Components of the Microcontroller

Application Specific Processors

25

- General Purpose Processors offer good performance for all different applications but specific purpose processors offer better for a specific task
- Application specific processors emerged as a solution for
 - higher performance
 - lower power consumption
 - Lower cost
- Application Specific Processors have become a part of our life and can be found almost in every device we use on a daily basis
 - Devices such as TVs, mobile phones and GPSs
- They are classified into
 1. Digital Signal Processor (DSPs)
 2. Application Specific Instruction Set Processors (ASIPs)
 3. Application Specific Integrated Circuit (ASICs)

Application Specific Processors

1. **DSP:** Programmable microprocessor for extensive real-time mathematical computations – they support Multiply-accumulate units
2. **ASIP:** Programmable microprocessor where hardware and instruction set are designed together for one special application
 - Instruction set, micro architecture and/or memory system are customised for an application or family of applications
 - better performance, lower cost, and lower power consumption than GPP
3. **ASIC:** Algorithm completely implemented in hardware
 - IC designed for a specific line of a company
 - proprietary by nature and not (always) available to the general public

Accelerators - Coprocessors

- Accelerators / co-processors are used to perform some functions more efficiently than the CPU

- Faster
 - Lower energy

➤ But harder to program

- Graphics Processing Unit (GPU)

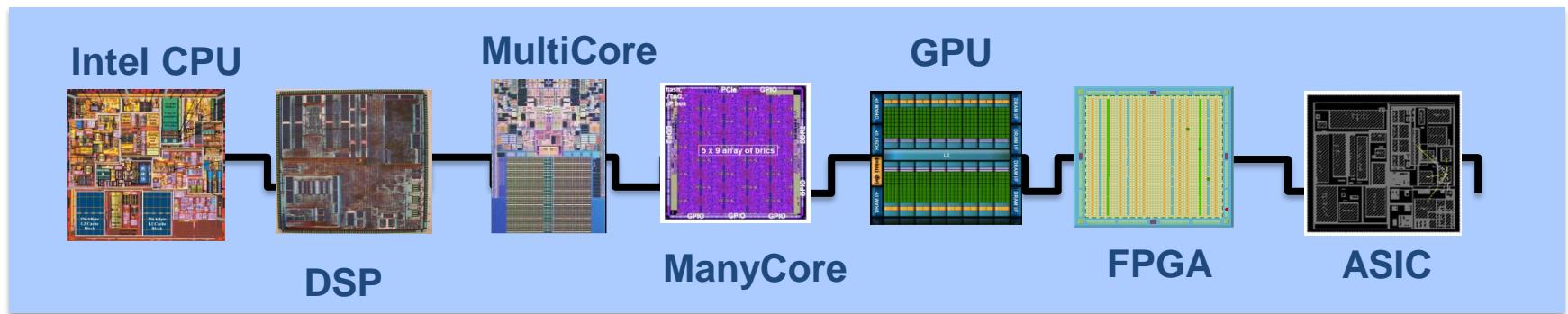
- Single Instruction Multiple Thread (SIMT) model – CUDA code
 - Efficient for
 - Data parallel applications
 - Throughput intensive applications - the algorithm is going to process lots of data elements

- **FPGA (Field Programmable Gate Array)** is an array of logic gates that can be hardware-programmed to fulfill user-specified tasks

- Part of the application can be implemented entirely in HW



Comparison



CPU:

- Market-agnostic
- Accessible to many programmers
- Flexible, portable

FPGA:

- Somewhat Restricted Market
- Harder to Program (VHDL, Verilog)
- More efficient than SW
- More expensive than ASIC

ASIC

- Market-specific
- Fewer programmers
- Rigid, less programmable
- Hard to build (physical)

Public Cloud providers – Compute Instance Examples

- Amazon Elastic Compute Cloud (EC2): Web service that provides secure, resizable compute capacity in the cloud
<https://aws.amazon.com/ec2/>
- Amazon EC2 Elastic **GPUs**: to attach low-cost graphics acceleration to a wide range of EC2 instances over the network
<https://aws.amazon.com/ec2/elastic-gpus/>
- Amazon EC2 F1: compute instance with **FPGAs** a user can program to create custom hardware accelerations for applications
<https://aws.amazon.com/ec2/instance-types/f1/>



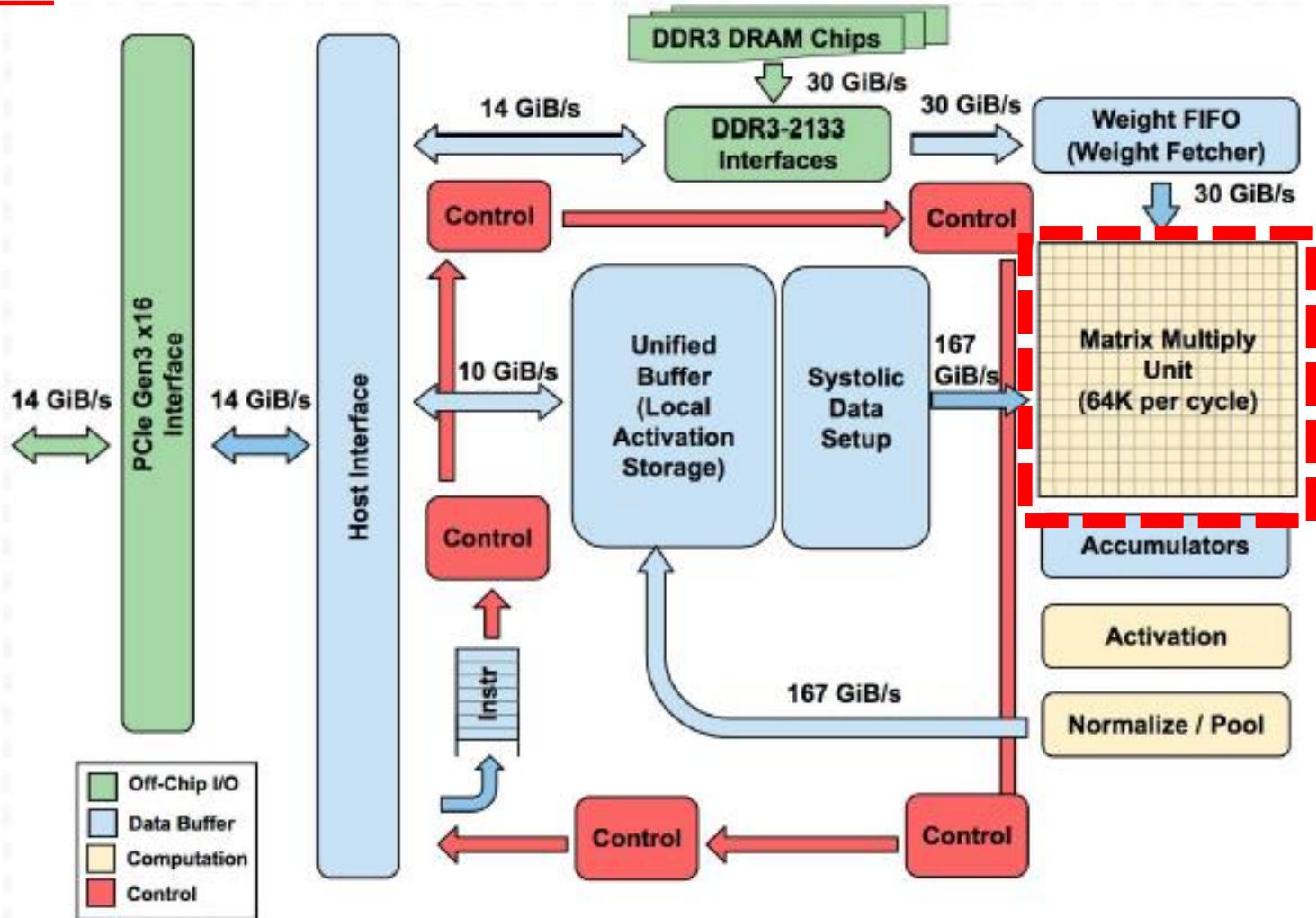
Custom Hardware: Google TPU

- Only a few applications could run on custom H/W (ASIC, FPGA, GPUs)
- Wide applicability of another computational paradigm: Neural Networks (NN)
 - could double the computation demands on datacenters
- Google Tensor Processing Unit - Up to 100 petaflops in performance
- Custom-designed machine learning ASIC that powers Google products
 - Translate, Photos, Search, Gmail
- Ideal to run machine learning workloads using machine learning frameworks, e.g. TensorFlow, Pytorch
- Currently a cloud service offering.



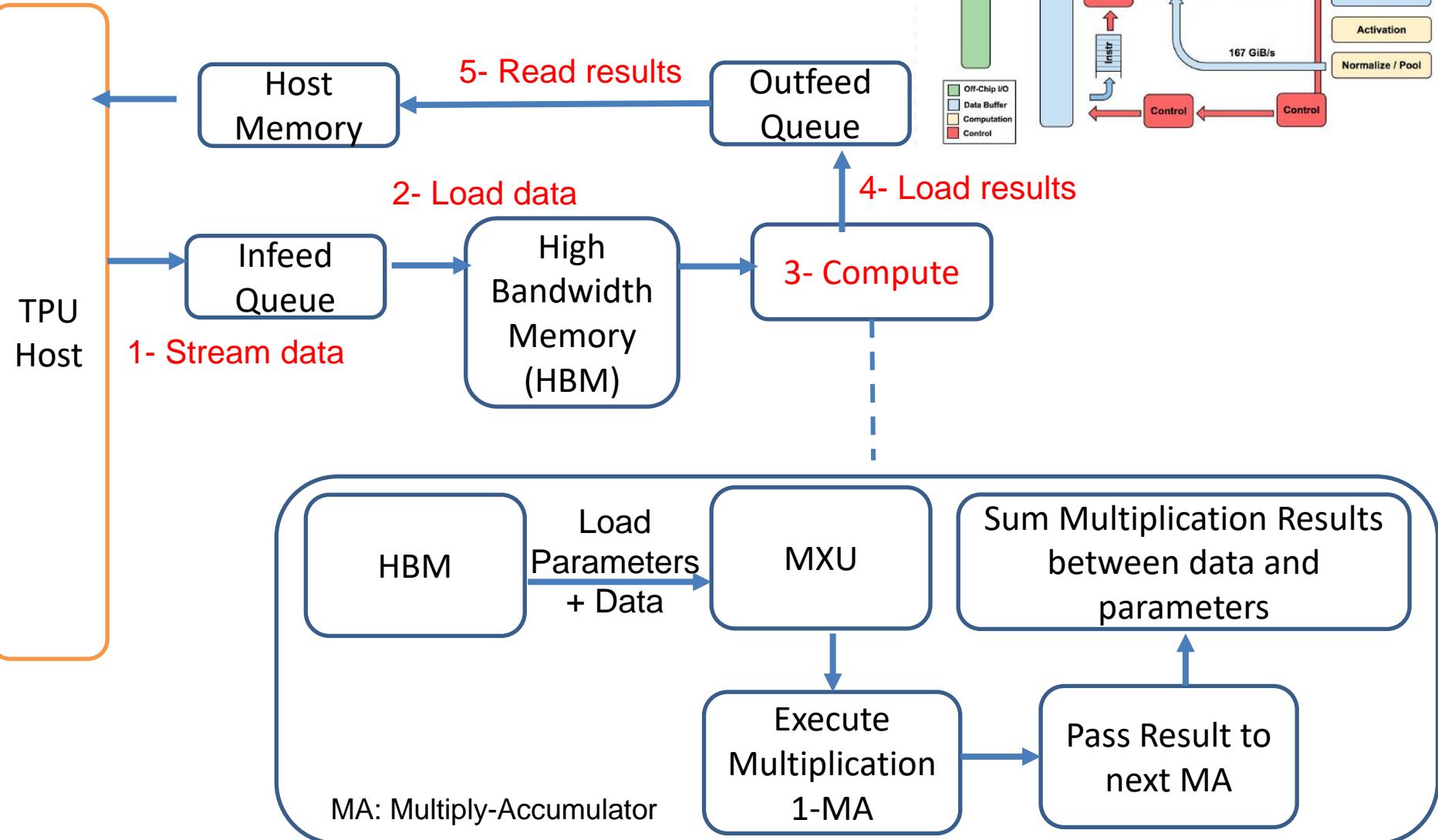
Courtesy of Google

TPU Architecture



- Matrix Multiply Init (MMU): contains 256 x 256 MACs (Multiplier-Accumulator)
- Instruction Set – Traditional Complex Instruction Set Computer (CISC)
- Dataflow from Unified Buffer
- Systolic system – network of processors compute/pass data

TPU Architecture



Conclusion

- Recap in relation to public, private, and hybrid clouds
- Reviewed design issues in relation to cloud datacenters at the system level:
 - Servers, racks, clusters
 - Storage
 - Networks
- Aspect of heterogeneity as well as specialised hardware are becoming important.

References

- Cloud Computing for Science and Engineering. I. Foster and D.B. Gannon. MIT Press, 2017
- S. Bhowmik, *Cloud computing*. Cambridge University Press, Chapter 7, 2017
- Distributed and Cloud Computing: From Parallel Processing to the Internet of Things. Hwang K, Fox G.C. & Dongarra J.J., 1st edition, Morgan Kaufmann, 2012
- RC2 – A Living Lab for Cloud Computing.
G. Ammons et al. IBM Technical Paper RC24947

https://www.usenix.org/legacy/events/lisa10/tech/full_papers/Ryu.pdf

COMP5850 – Cloud Computing



Virtualisation *Part I*

Plan of the Lecture

Goals

- Understand concepts of virtualisation

Overview

- Definitions
- Implementation levels of virtualisation
- Machine Level Virtualisation
- Hypervisor-based Virtualisation
 - Full virtualization
 - Para-virtualization
 - Hardware-assisted
- Operating System Level Virtualisation
- Examples
- Conclusion

Cloud Resources: Virtualisation Layer

User level

Cloud applications



User-Level
Middleware

Cloud programming: environments and tools

Apps Hosting Platforms

Core
Middleware

Execution Management

Virtual Machine (VM), VM Management and Deployment



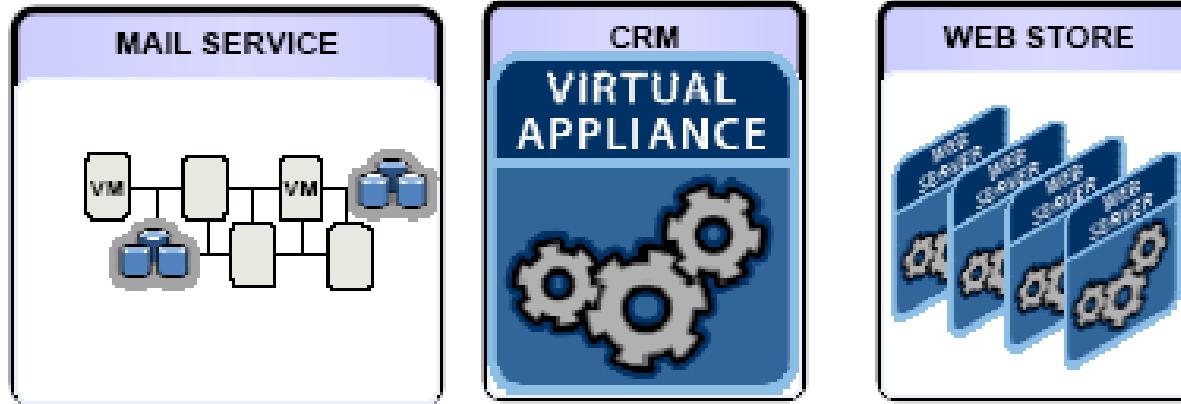
System level

Cloud resources



User's view of virtualization

LOGICAL VIEW



Virtualization Layer - Optimize HW utilization, power, etc.

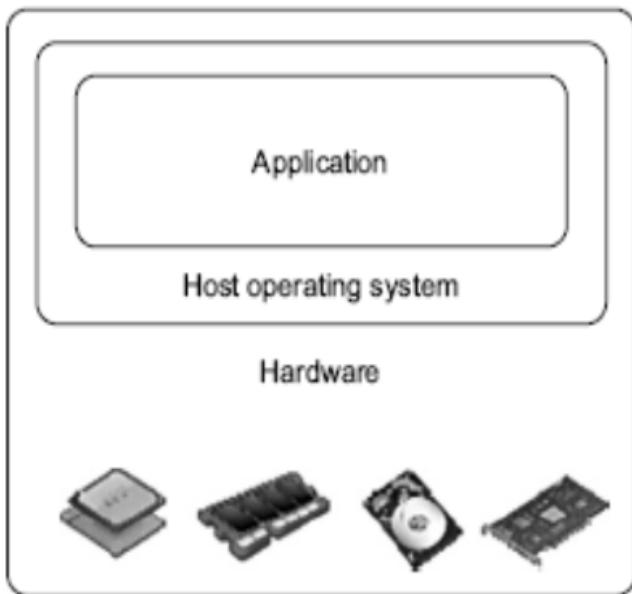
PHYSICAL VIEW



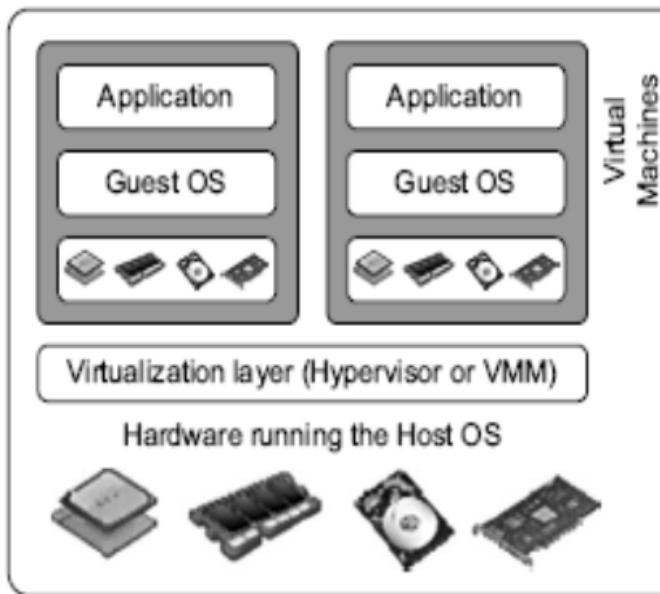
Advantages of virtualisation:

Better utilisation of resources, hardware cost reduction, improved fault tolerance, capacity expansion ...

Difference between Traditional Computer and Virtual machines



(a) Traditional computer



(b) After virtualization

The main technology enabling virtualisation is the **Hypervisor**

- Virtual Machine Manager (or Virtual Machine Monitor (VMM)) that partitions a physical host server transparently via emulation or hardware-assisted virtualisation

Implementation Levels of Virtualization

- Application
 - Virtualises an application as a VM
 - Process level virtualization
- Library (user-level API)
 - APIs become candidates for virtualization
 - Control of the communication link between applications and the rest of the system through API hooks
- Operating System
 - Create isolated containers on a single physical server and the OS instances to utilize the hardware and software
- Hardware Abstraction Layer (HAL)
 - Manages the hardware (processors, memory, I/O devices) through virtualization
 - Virtual hardware environment for VMs
- Instruction Set Architecture (ISA)
 - Emulates a given ISA by the ISA of the host machine
 - E.g. Run legacy binary code written for various processors on any new hardware host machine

Implementation Levels of Virtualization

Application Level

.NET CLR, JVM

Library (user level API) Level

WINE, vCUDA

Operating System Level

OpenVZ

Hardware Abstraction Layer (HAL) Level

Vmware, XEN

Instruction Set Architecture (ISA) Level

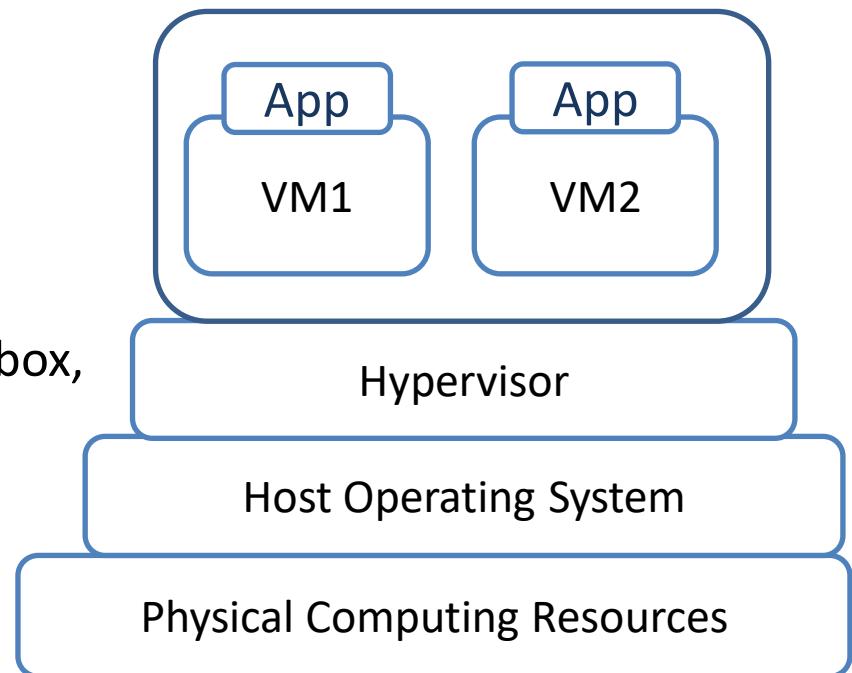
QEMU

Machine Level Virtualisation

- Two different techniques
 - Hosted approach
 - Bare metal approach
- The techniques differ depending on the type of hypervisor used
- Aim: create a platform where multiple VMs can share the same system resources

• Hosted Approach: Type 2 Hypervisor

- Run over a host operating system
- Is the second layer over the hardware.
- Guest operating systems run a layer over the hypervisor
- Example: VMWare Workstation, Virtualbox, Microsoft Virtual PC

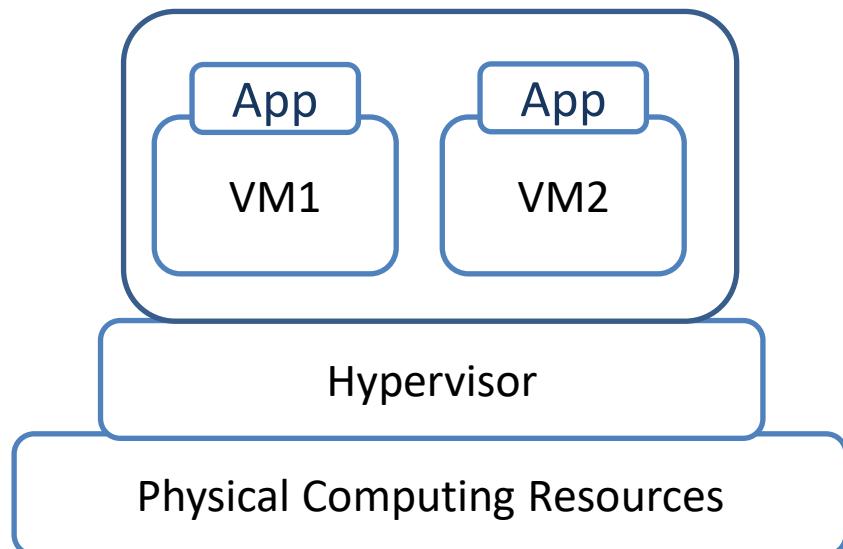


Question: Advantages/Drawbacks?

Machine Level Virtualisation

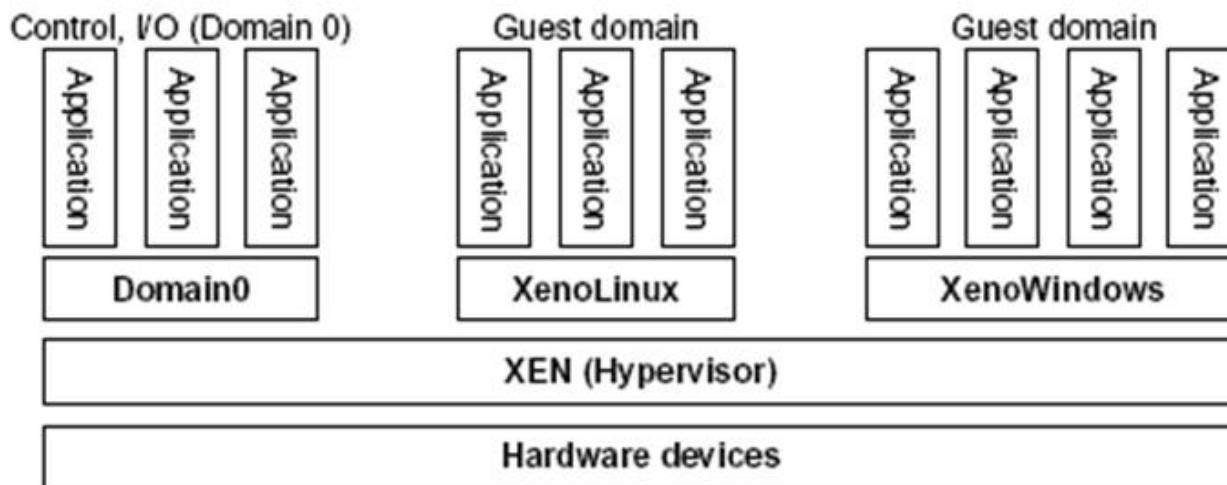
- **Bare Metal: Type 1 hypervisor**

- sits on the bare metal computer hardware like the CPU, memory, etc.
- All the guest operating systems are a layer above the hypervisor.
- Hypervisor is the first layer over the hardware
- Example: VMWare ESX, Xen, KVM, Microsoft Hyper-V.



Question: Advantages/Drawbacks?

The XEN Architecture

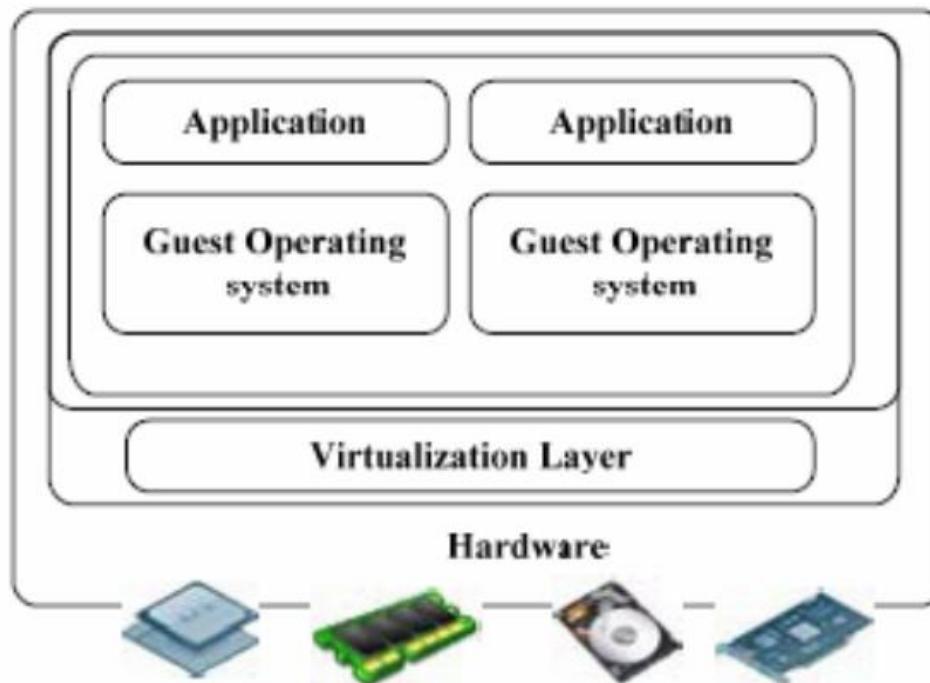


The Xen architecture's special domain 0 for control and I/O, and several guest domains for user applications.

Hypervisor-based Virtualisation

- The Hypervisor presents a virtual operating platform before the guest systems
- Monitors and manages the execution of guest systems and the Virtual Machines
- Hypervisor-based virtualization techniques can be divided into three categories:
 1. Full virtualisation
 2. Para-virtualisation
 3. Hardware-assisted virtualisation

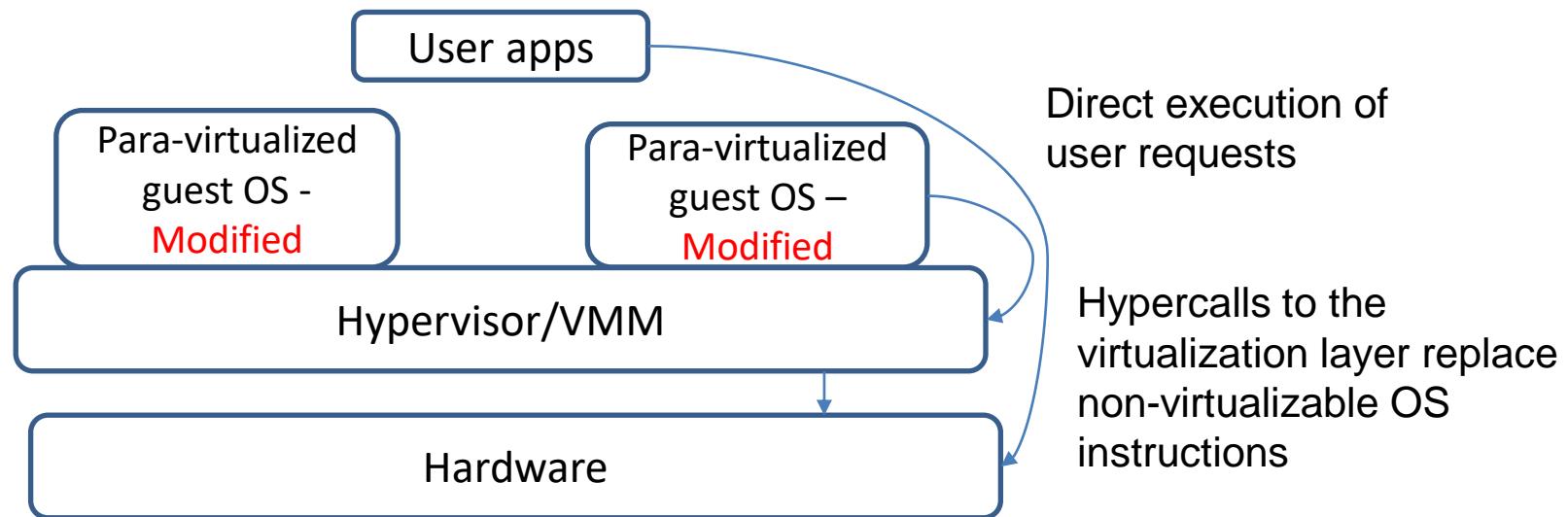
1- Full Virtualisation



- Unmodified versions of available Oss (e.g, Windows, Linux) run as guest OS over hypervisor
 - **Flexibility!**
- The hypervisor fully simulates or emulates the underlying hardware
 - Handles all guest-OS to hardware calls
- Critical instructions are emulated by software through the use of binary translation
- This approach of binary translation **slows down** the performance
 - e.g. VMWare, Microsoft Virtual Server

2- Para-Virtualisation

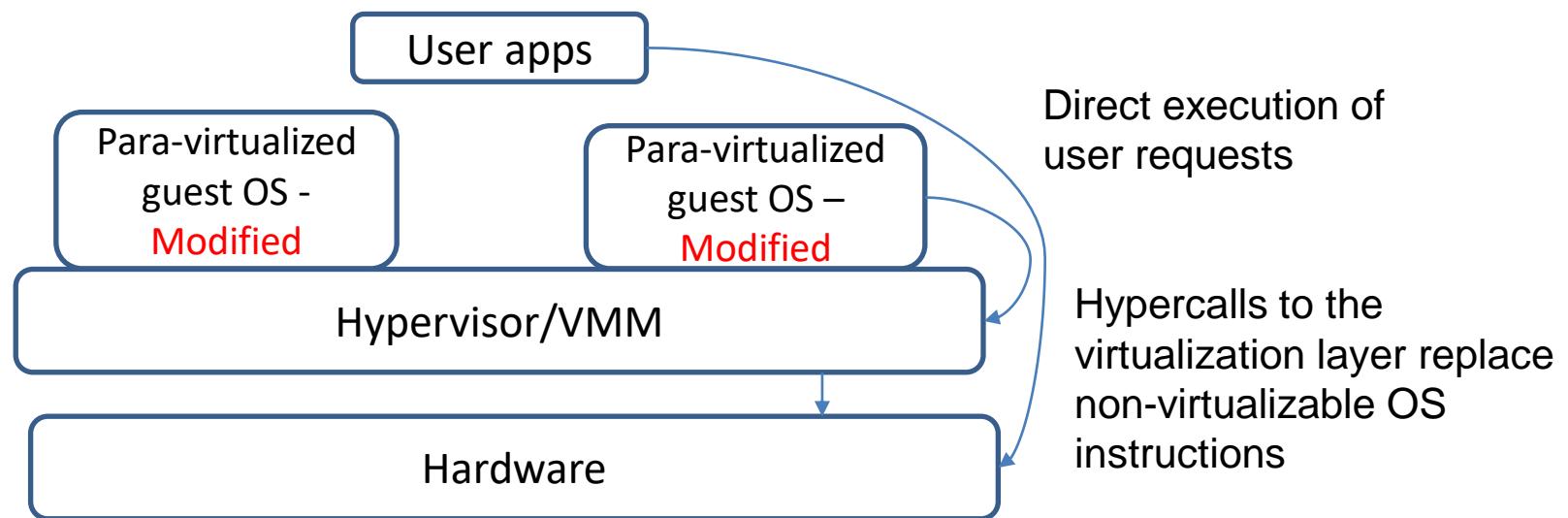
- A portion of the virtualization management task is transferred from the hypervisor towards the guest OS
 - Needs to modify guest OS for this capability inclusion: **porting**
 - Guest OS is explicitly ported for the para-Application Program Interface (API). Need to have prior knowledge
 - that it will run over the virtualized platform
 - on which particular hypervisor it will have to run



Para-Virtualisation (2)

- **Para-virtualization**

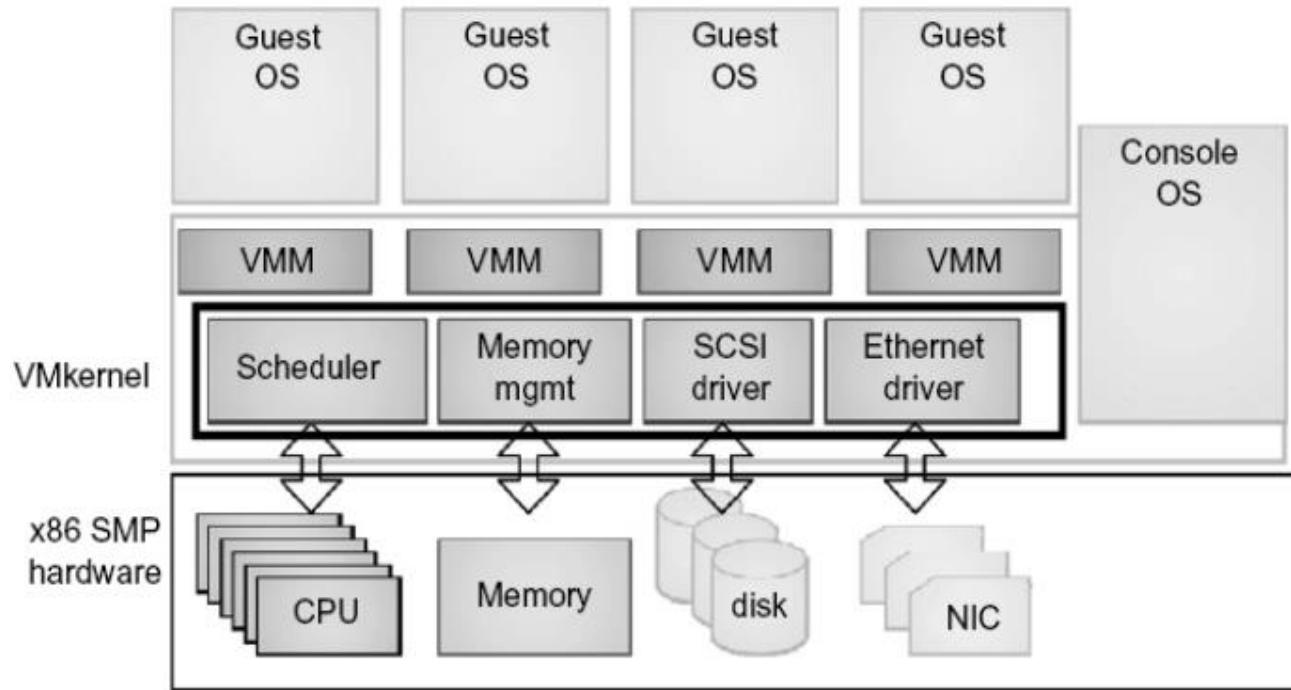
- Non-virtualizable instructions are replaced by hypercalls that communicate directly with the hypervisor or VMM
- Reduces the overhead of the hypervisor as compared to full virtualization
- Cost of maintaining paravirtualized OS is high.
- Supported by Xen, KVM, VMware ESX.



Para-Virtualization (3)

- The guest OS is **recompiled** prior to installation inside a VM
- Allows for an interface to the VM that can differ from that of the underlying hardware
 - provides specially defined '**hooks**' to do some tasks in the host and guest operating systems
- The hypervisor in a para-virtualized platform is simpler because the critical tasks are now performed in the OS rather than by the hypervisor
 - Since the virtualization **overhead decreases** the performance **increases**.
- Disadvantages
 - Reduced compatibility and portability because of the modified OS
 - High cost of maintenance because of the deep OS modifications.

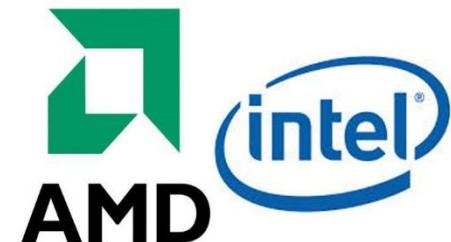
VMWare ESX Server for Para-Virtualization



The VMware ESX server architecture using para-virtualization.

3- Hardware-Assisted Virtualisation

- Manufacturing devices tailored to support virtualisation by hardware vendors
- Virtualisation features included in the processor
- Allow some privilege CPU calls from the guest OS to be directly handled by the CPU
 - No need for these calls to be translated by the hypervisors
- Example: AMD-Virtualisation (AMD-V) and Intel-Virtualisation Technology (Intel-VT)

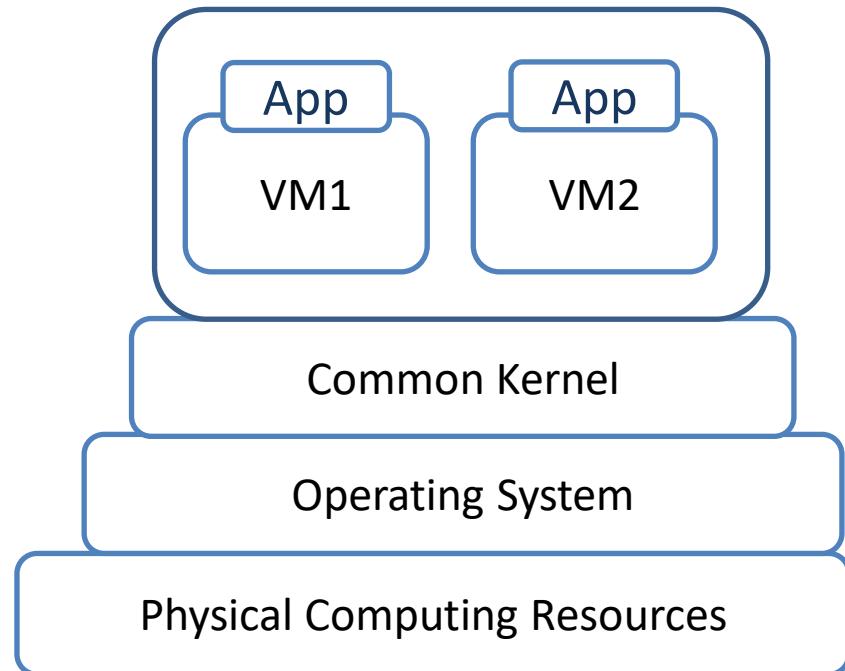


Major Hypervisor Providers

VMM Provider	Host CPU	Guest CPU	Host OS	Guest OS	VM Architecture
VMware Work-station	X86, x86-64	X86, x86-64	Windows, Linux	Windows, Linux, Solaris, FreeBSD, Netware, OS/2, SCO, BeOS, Darwin	Full Virtualization
VMware ESX Server	X86, x86-64	X86, x86-64	No host OS	The same as VMware workstation	Para-Virtualization
XEN	X86, x86-64, IA-64	X86, x86-64, IA-64	NetBSD, Linux, Solaris	FreeBSD, NetBSD, Linux, Solaris, windows XP and 2003 Server	Hypervisor
KVM	X86, x86-64, IA64, S390, PowerPC	X86, x86-64, IA64, S390, PowerPC	Linux	Linux, Windows, FreeBSD, Solaris	Para-Virtualization

Operating System Level Virtualisation

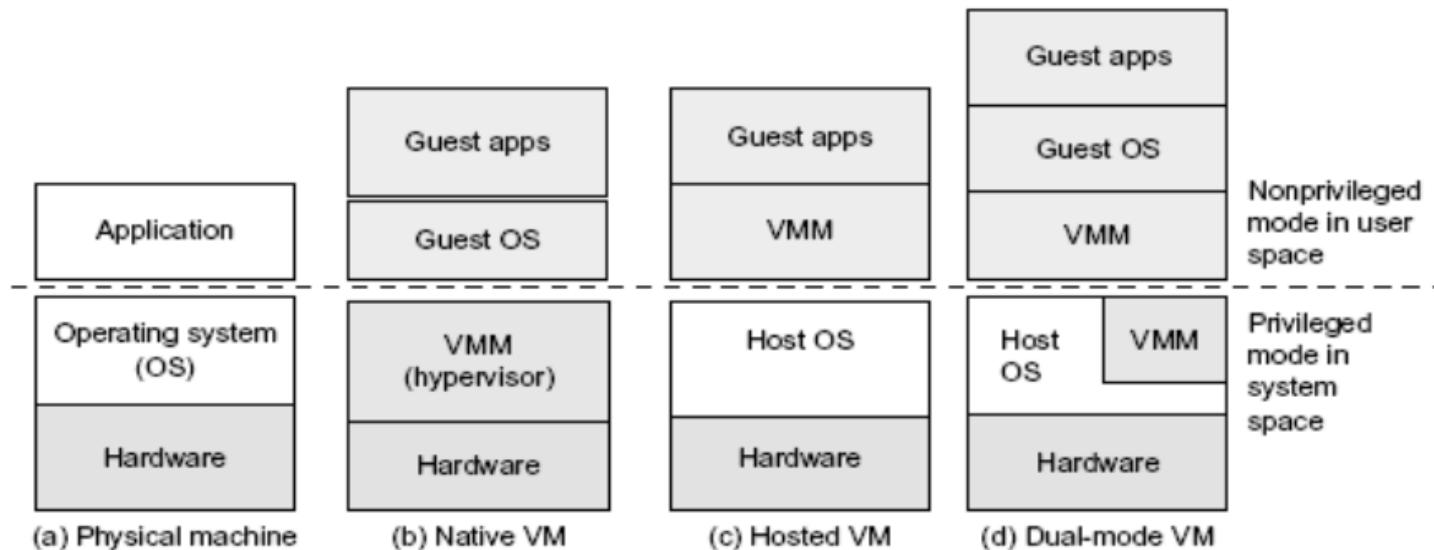
- Also known as system level virtualisation
- Virtual servers
 - enabled by the **shared** kernel of the OS
 - Have the same OS, the parent system
- Examples: FreeBSD jail, Linux Vserver, OpenVZ
- Advantages
 - Lighter in weight



See lectures on containerisation

A Note on VM Architectures

Three VM architectures compared with a traditional physical machine



- a) OS executes in a privileged mode
- b) VMM operates in a privileged mode
- c) Trusted host OS executes in a privileged mode in a hosted VM system.
Guest OS resides in the VM and operates at a less privileged level
- d) Parts of the VMM operate in the privileged mode.

A Quick Note on APIs

- Virtualisation platforms can be managed by toolkits, e.g.
 - **libvirt** project provides virtualisation APIs
 - <https://libvirt.org/>
- Accessible from C, Python, Perl, Java and more programming languages
- Supports KVM, QEMU, Xen, VMWare ESX, Linux Containers (LXC) ...
- Python Example:

```
def main(args)
    vm_name = args['<vm_name>']
    # get domain from libvirt
    con = libvirt.open('qemu:///system')
    domain = con.lookupByName(vm_name)
...

```

Conclusion

- Gave some definitions in relation to virtualization
- Defined the implementation levels of virtualization
- Many virtualization structures/tools and mechanisms
 - Full virtualization
 - Para-virtualization
 - Hardware-assisted
 - OS level
- Hypervisors: VMWare, Xen and KVM

References

- Cloud Computing for Science and Engineering. I. Foster and D.B. Gannon. MIT Press, 2017
- S. Bhowmik, *Cloud computing*. Cambridge University Press, Chapter 7, 2017

COMP5123M – Cloud Computing Systems



Virtualisation *Part II*

Plan of the Lecture

Goals

- Understand concepts of virtualisation

Overview

- Virtual Machines: Recap
- Container technologies
- VMs vs Containers
- Example: Docker
- The case for Unikernels
- Conclusion

Cloud Resources: Virtualisation Layer

User level

Cloud applications



User-Level
Middleware

Cloud programming: environments and tools

Apps Hosting Platforms

Core
Middleware

Execution Management

Virtual Machine (VM), Containers, Management and Deployment



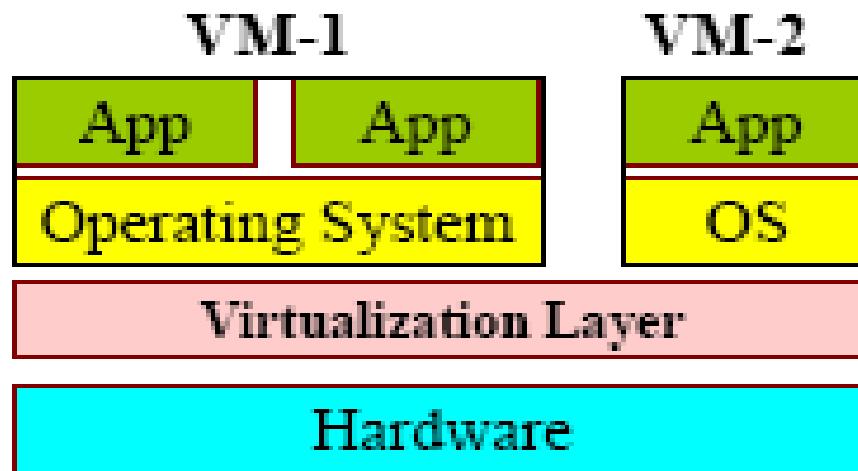
System level

Cloud resources



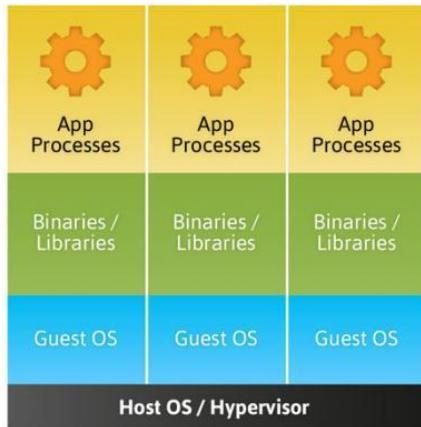
What is Virtualisation ?

- A level of indirection between hardware and software.

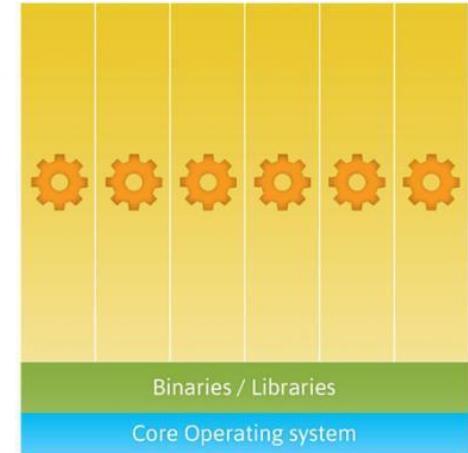


- Virtual Machine abstraction
 - Run all software written for physical machine.
- Virtualization layer is known as hypervisor or Virtual Machine Monitor (VMM)

Virtual Machines versus Containers



VM

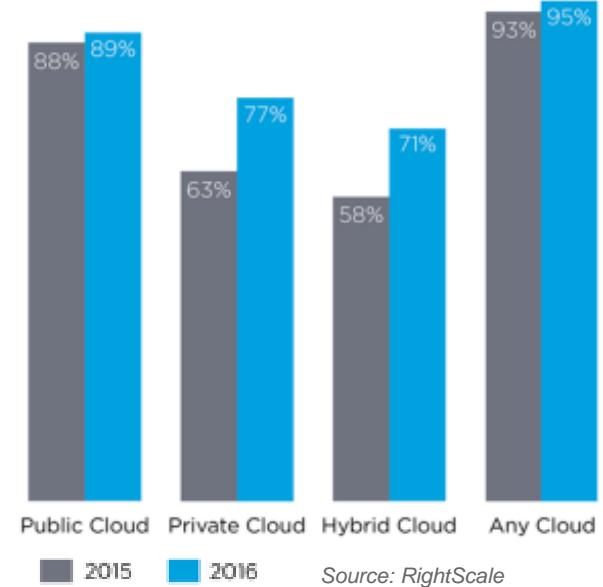


Container

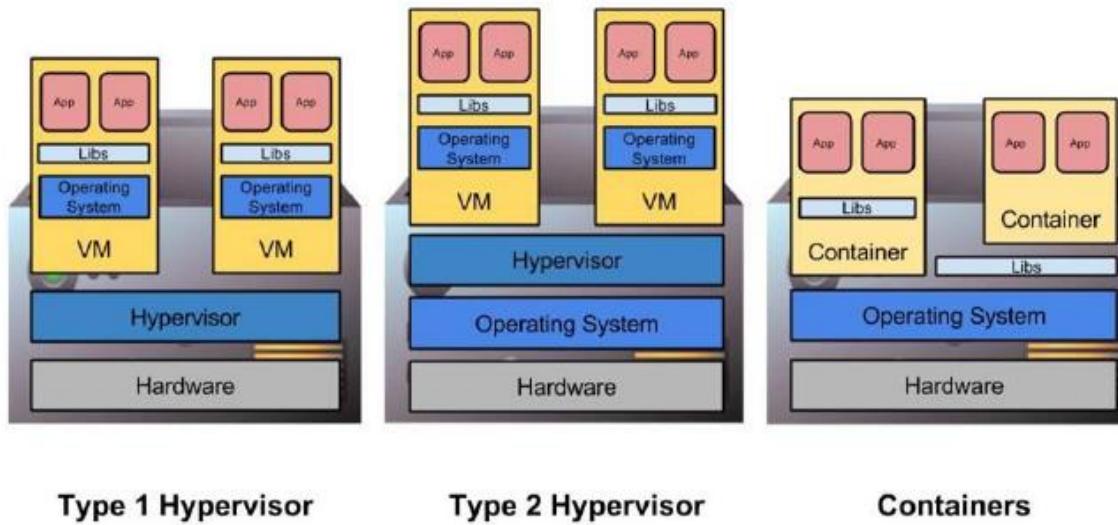
- Container virtualises the OS rather than the hardware
- Uses a single kernel to run multiple instances of an operating system
 - Each instance runs in a completely isolated environment
 - Is secure.
- Can offer greater efficiency and performance than a conventional hypervisor

Containers Leading the IT Transformation

- Cloud is widely adopted by most of the companies.
 - The interest in **Containers** is expanding rapidly
- Container History
 - Started on Unix with **chroot** in 1979.
 - Variants in the evolution:
 - 1998: FreeBSD jails
 - 2005: Solaris One, OpenVZ
 - 2008: LXC, modern Linux container, OS container
 - 2013: Docker

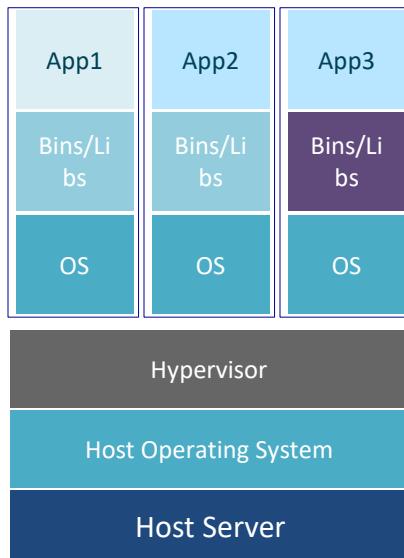


Containers and Hypervisors

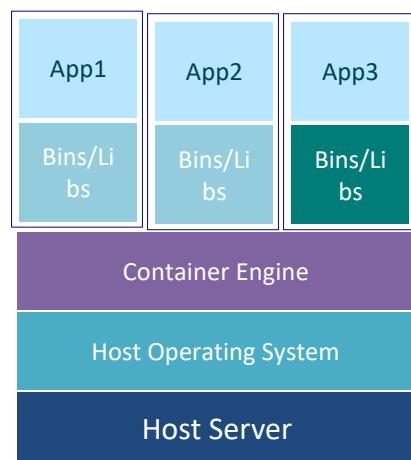


- Type 1 hypervisors insert layer below host OS
- Type 2 hypervisors work as or within the host OS
- Containers do not abstract hardware, instead provide “enhanced chroot” to create isolated environment using a common kernel
- Location of abstraction can have impact on performance
- All enable custom software stacks on existing hardware

Containers and VMs: Differences



Virtual Machines



Containers

Containers lighter and better performance:

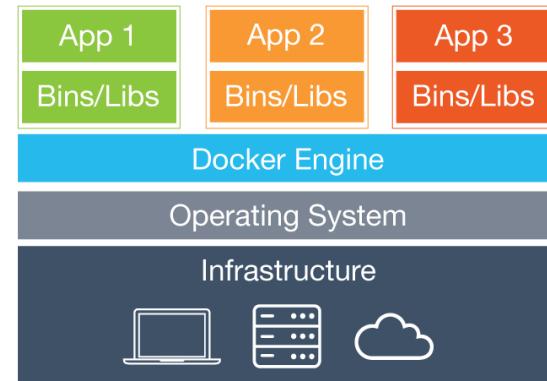
- **Portability:** VM (Gigabytes) vs. Container (Megabytes), VMs are constrained to Hypervisor and hardware-emulation
- **Performance:** Containers can boot and restart in seconds, compared to minutes for virtual machines
 - No extra overhead of a hypervisor and guest OS makes containers consume less CPU and memory
- **Management cost:** Each VM requires a full functional operating system, and then extra management

Clear advantage to use containers in:

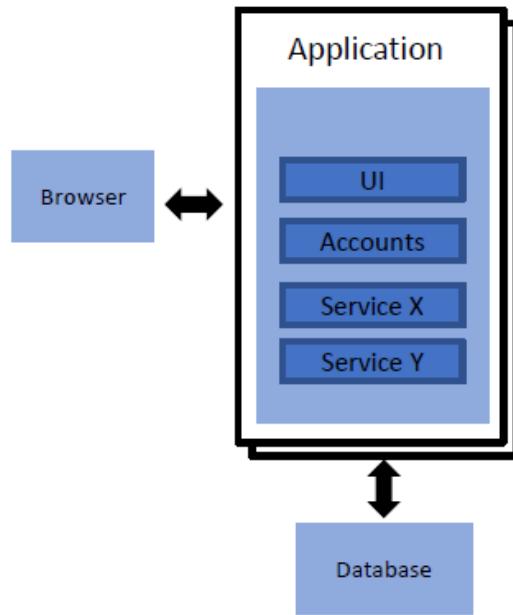
- DevOps
- Batch computing
- Microservices

The Case of Containers

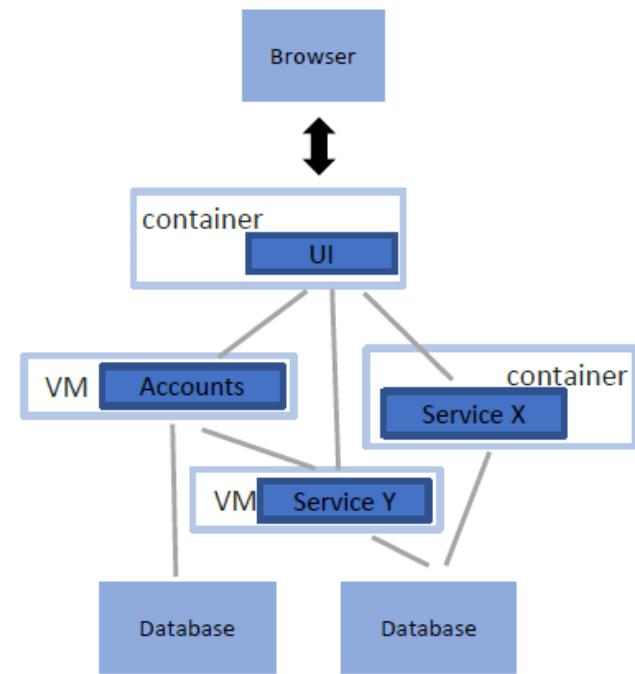
- Inside the container is a single executable service that is a small part of an application: **a micro-service**
- Micro-services provide a simple function from a single piece of executable code.
- Applications are made up of a collection of micro-services
 - each provides a small part of the overall application.
- Multiple copies of a single micro-service may be run in many containers, and those containers are linked together with message queuing or load balancing.
- Application itself is built to cope with individual container failure and each container has its own instance of the micro-service software code.
- Containers for the same micro-service run the same code.



The Case for Micro-services



Monolithic approach



Microservices approach

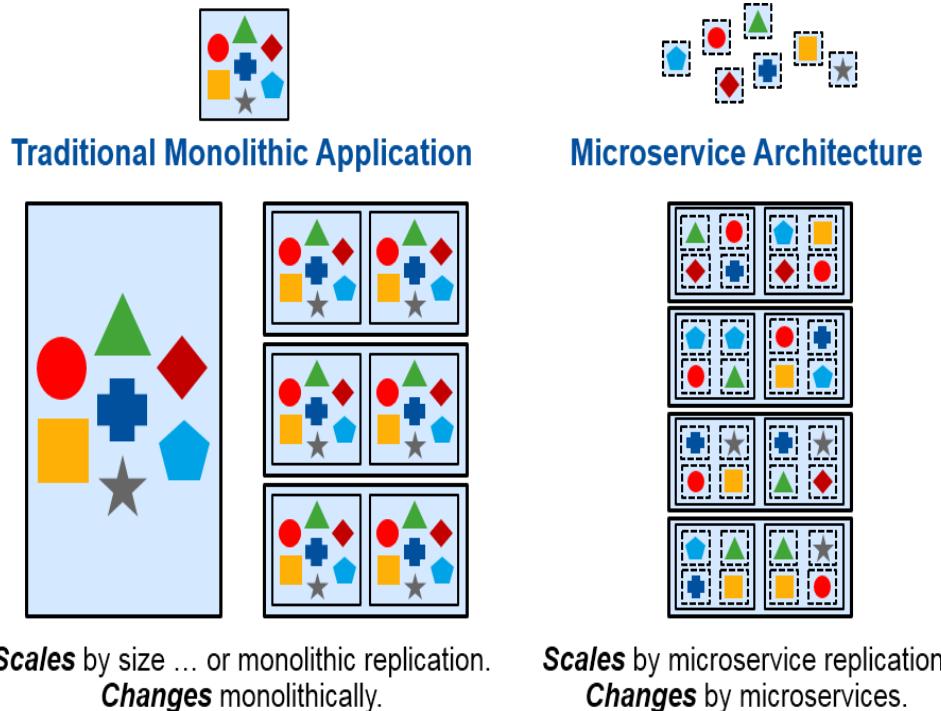
An architectural pattern

- Unit of distribution (single service granularity)
- Small autonomous services that work together
- Loosely-coupled components
- API-accessible
- Inter-component software access via a chosen protocol (e.g. HTTP)

Application Development Requires Efficient Container Management

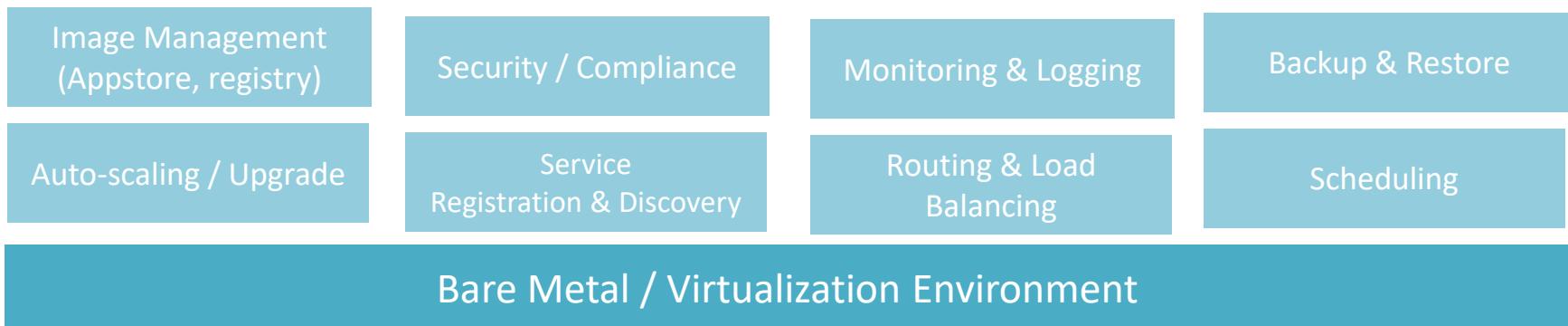
Microservices Architecture

- Decomposed into small pieces
- Easy to scale development
- Improved fault isolation
- Each service can be developed and deployed independently



Container Management Platforms are Emerging

- **Container Management Platforms** are also known as container managers, container orchestration engines (COEs), container orchestration platforms and containers-as-a-service (CaaS) platforms.
- Help enterprises address the challenges of deploying containers
- These platforms let administrators monitor, manage, secure, and scale containers similar to how they treat existing servers and virtual machines.

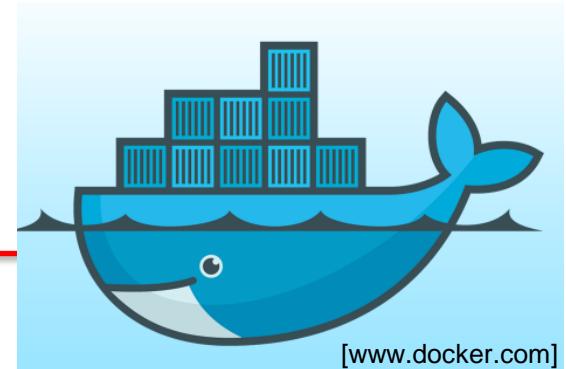


What is Docker?

Docker <https://www.docker.com/>

Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating system-level virtualization on Linux.

Docker: Name

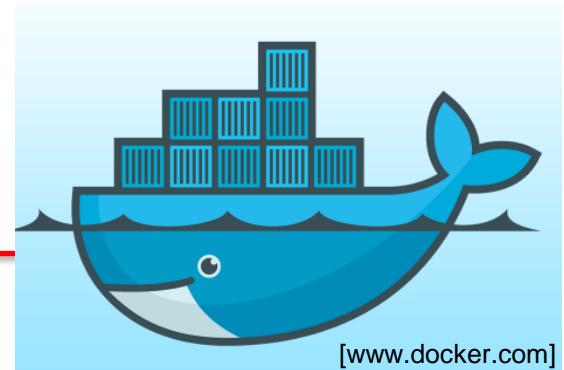


[www.docker.com]

docker: a person employed in a port to load and unload ships

- Provide a uniformed wrapper around a software package: «*Build, Ship and Run Any App, Anywhere*» [www.docker.com]
 - Similar to shipping containers: the container is always **the same**, regardless of the contents and thus fits on all trucks, cranes, ships, ...

Docker: Definitions



- **Image:** A read-only template that defines how to create a container
- **Container:** An instantiation of an image, a running instance
- **Container Runtime:** Tool or service to execute and manage containers
- **Registry:** A service that is used to store and distribute images

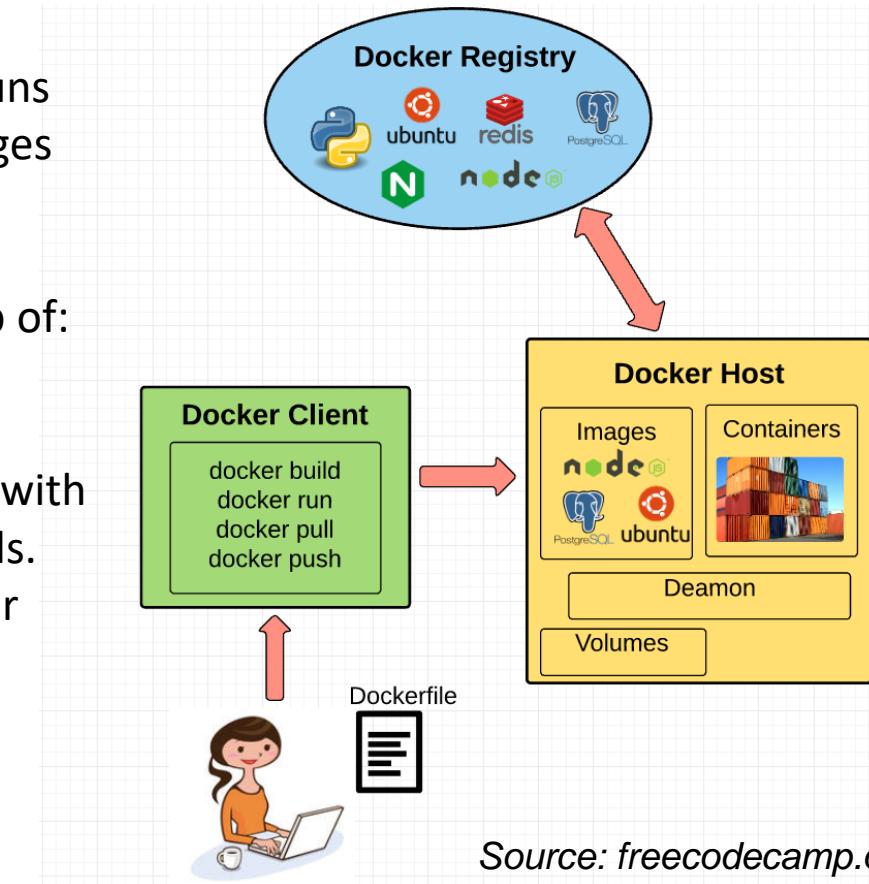
Docker Concepts

Docker Engine: is the layer on which Docker runs

- lightweight runtime and tooling that manages containers, images, builds, ...

Runs natively on Linux systems and is made up of:

1. **Docker Daemon** that runs in the host computer.
2. **Docker Client** that then communicates with the Docker Daemon to execute commands.
3. **REST API** for interacting with the Docker Daemon remotely.



Source: freecodecamp.org

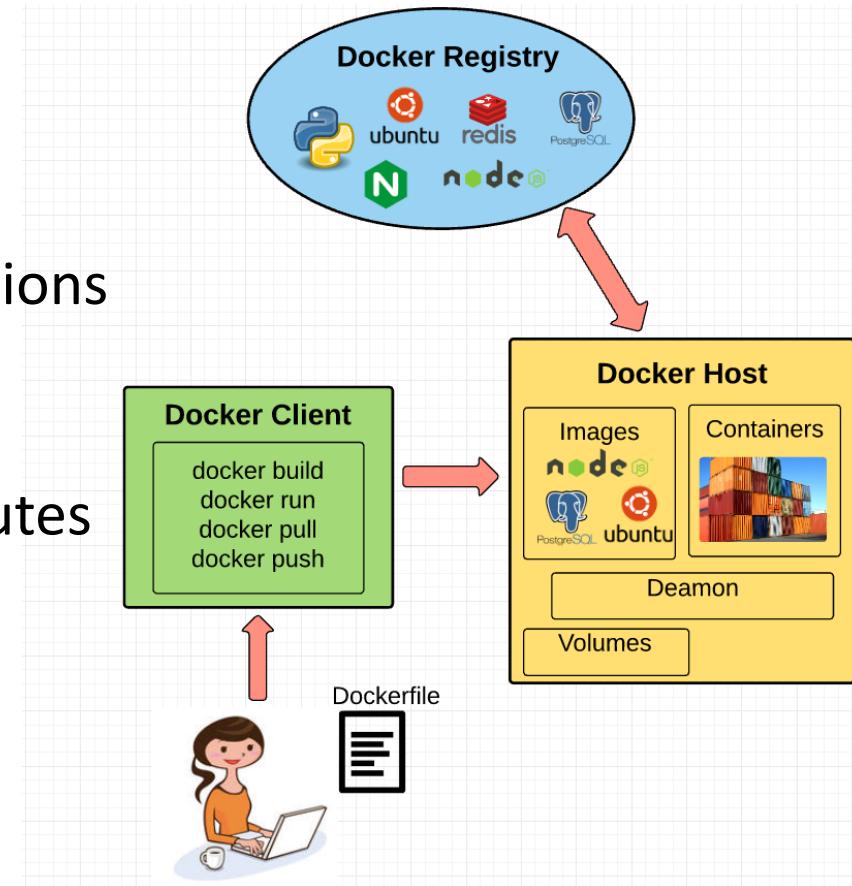
Docker Concepts

Docker Client: what the end-user of Docker, communicates with

- User Interface
- Communicates end-user's instructions to the Docker Daemon

Docker Daemon: what actually executes commands sent to the Docker Client

- Building, running, and distributing containers
- Runs on the host machine
- Docker Client can run on the host machine as well, but is not required to

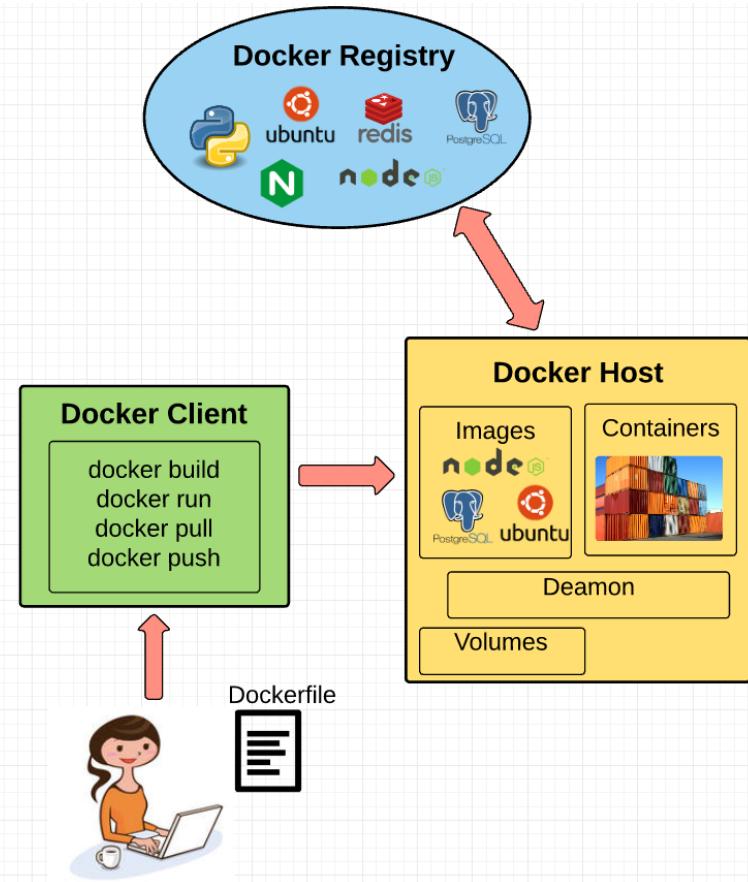


Docker Concepts

Dockerfile: is where the end-user writes the instructions to build a Docker image

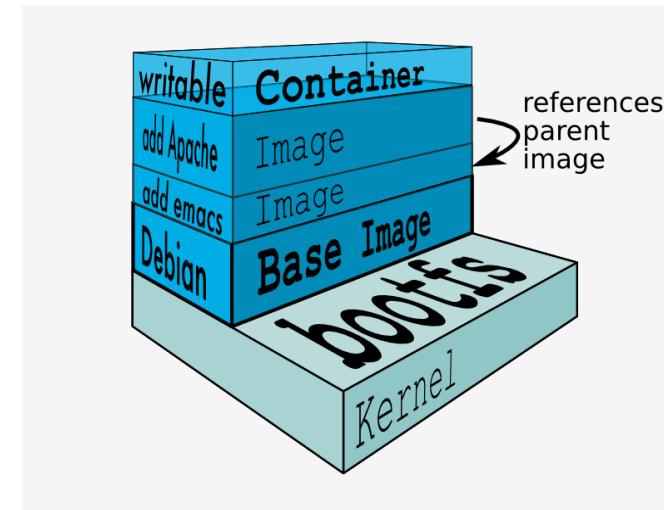
Docker Images: are read-only templates that the end-user builds from a set of instructions written in the Dockerfile

- Images define both what
 - the end-user's packaged application and its dependencies to look like
 - processes to run when the application is launched.



Docker Technology

- A Docker container **wraps** an application's software into an invisible box with everything the application needs to run
 - includes the operating system, application code, runtime, system tools, system libraries
- Since images are read-only, Docker adds a **read-write file system** over the read-only file system of the image to create a container.
- Docker creates a **network interface** so that the container can talk to the local host, attaches an **IP address** to the container, and **executes** the process that is specified to run the application when defining the image.



Docker Technology (2)

Docker Compose

- A tool for designing and running multi-container services
- Uses a YAML file to specify containers and their options
- More basic than solutions that came on later, e.g., Kubernetes (*more later ...*)

Example: Service requiring 2 containers

- CodiMD deployment (a markdown note server)
- Requires a database service for users and notes

Compose allows to

- Specify dependencies between containers
- Configure Docker volumes
- Configure Docker networks

```
services:  
  database:  
    image: postgres:11.6-alpine  
    [...]  
  codimd:  
    image: hackmdio/hackmd:2.1.0-cjk  
    [...]  
    depends_on:  
      - database  
  
volumes:  
  database-data: {}  
  upload-data: {}
```

Docker Technology (3)



Docker Hub:

- Docker images are located and stored in the Docker Hub
 - This is like Github
- Once Docker images are developed, their owners commit them and push them to Docker Hub so that they could be used by others.

Summary: Deployment Options

manageability



Dedicated HW

VM

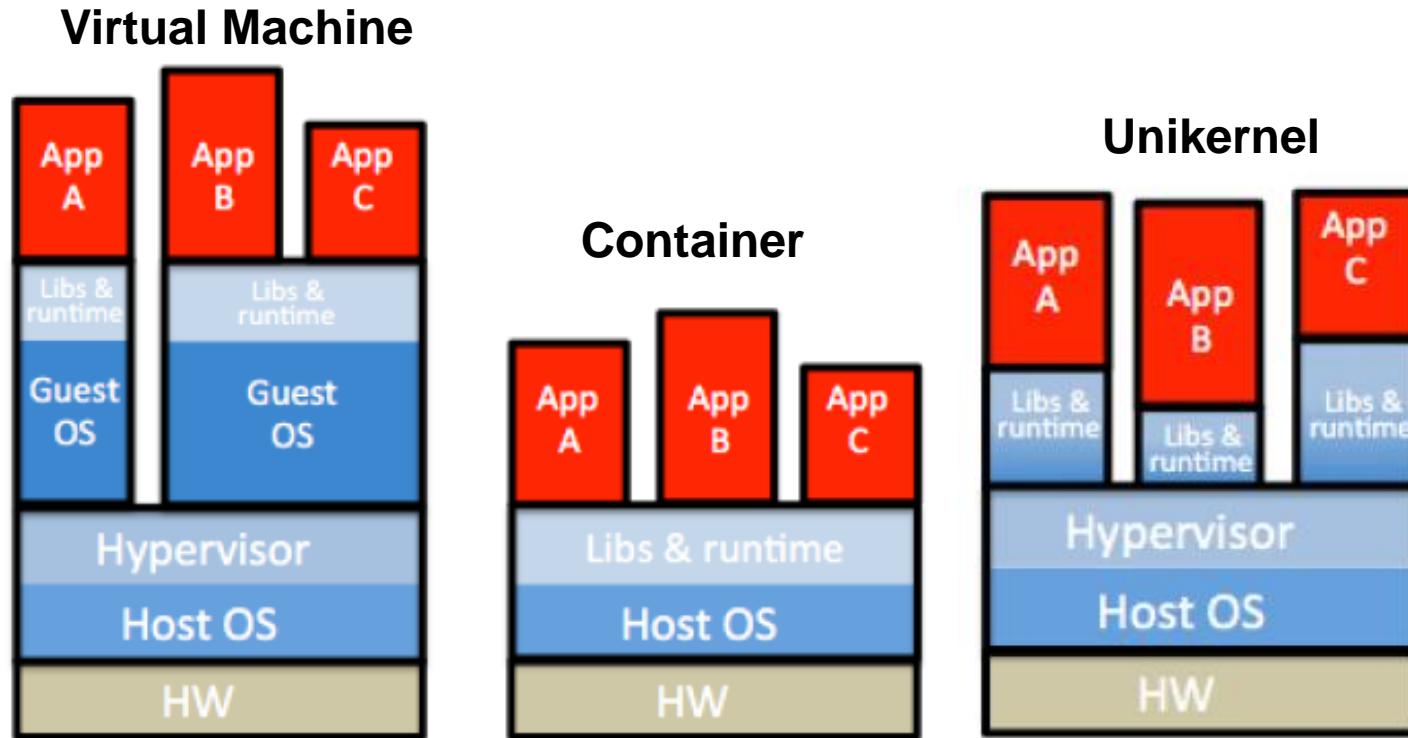
Container

Unikernel

efficiency

-
- Services can be placed in anywhere from dedicated hardware to just a software package
 - **Containers**: ideal for efficiency and management
 - most of the advantages of **Virtual Machines** (VMs)
 - Less overhead than VMs
 - Further improvement?
 - **Unikernels**

The Case for Unikernels



- Implement the bare minimum of traditional operating system functions
 - Are extremely **light**, allowing high density on commodity hardware
- Can run their own services that are **born** when the need appears, and **die** as soon as the need disappears
 - Some of these transient **microservices** may have lifespans measured in **seconds** or **even fractions of a second**
- **Are just-in-time** computing services, which exist only when there is work to do, therefore maximising the use of the computing infrastructure.

Conclusion

- Gave some definitions in relation to virtualization
- Defined the implementation levels of virtualization
- Introduced virtualisation structures/tools and mechanisms
- Container as an alternative to a Virtual Machine
 - comprises just the application and its dependencies
 - runs as an isolated process in user space on the host operating system, sharing the kernel with other containers
 - Enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient
- Note: Singularity as an alternative to Docker
<http://singularity.lbl.gov/>

References

- Cloud Computing for Science and Engineering. I. Foster and D.B. Gannon. MIT Press, 2017
- Cloud Computing. S. Bhowmik, Chapter 7. Cambridge, 2017
- Docker <https://www.docker.com/>
- Singularity: <http://singularity.lbl.gov/>

COMP5123M – Cloud Computing Systems



Cloud Virtual Infrastructure Management

Plan of the Lecture

Goals

- Understand Virtual Infrastructure Manager (VIM) features

Overview

- Introduction
- Virtual infrastructure Managers
- Architectural view
- Considerations
 - Compute, Storage, Network
 - Programming – APIs
- Conclusion

Architectural Layer: Where Are We?

User level



User-Level
Middleware

Cloud applications

Social computing, Enterprise, Scientific, ...

Cloud programming: environments and tools

Web 2.0 Interfaces, Mashups, Concurrent and Distributed Programming, Workflows, Libraries, Scripting

Apps Hosting Platforms

Core
Middleware

QoS Negotiation, Admission Control, Pricing, SLA Management, Monitoring, Execution Management, Metering, Accounting, Billing

Virtual Machine (VM), VM Management and Deployment

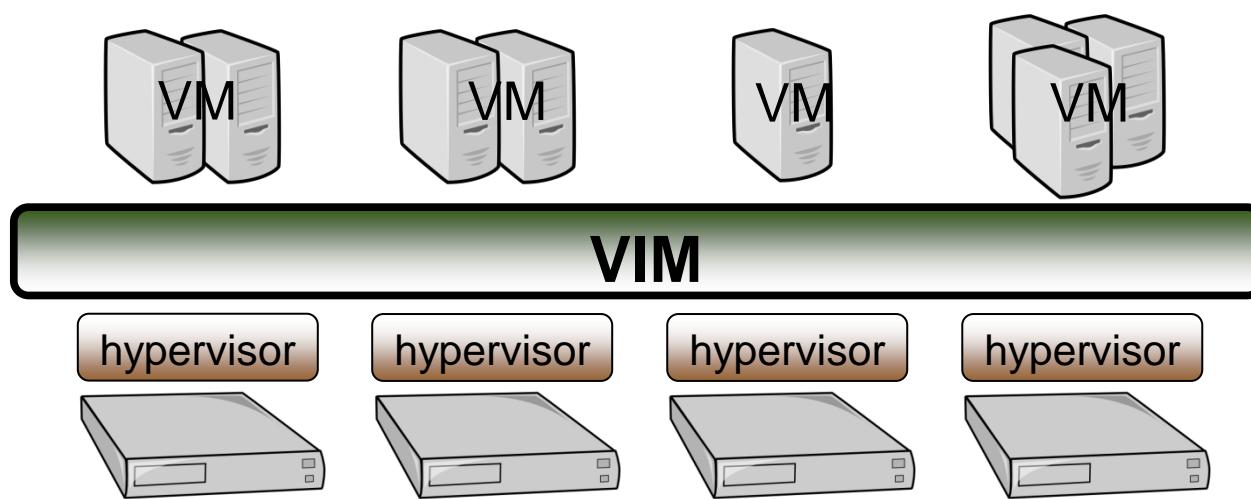
System level

Cloud resources



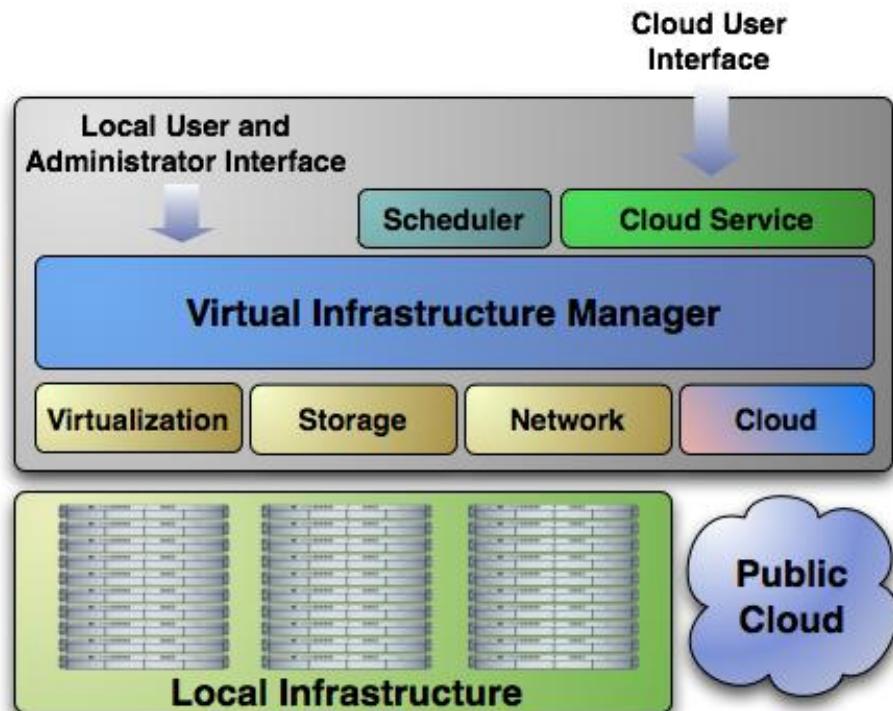
Why a Virtual Infrastructure Manager (VIM)?

- VMs are great...but something more is needed
 - Where did/do I put my VM? (*scheduling & monitoring*)
 - How do I provision a new cluster node? (*clone & context*)
 - What MAC addresses are available? (*networking*)
- Provides a *uniform view* of the resource pool
- *Life-cycle management* and monitoring of VM
- The VIM *integrates* Image, Network and Virtualisation



Extending the Benefits of Virtualisation to Clusters

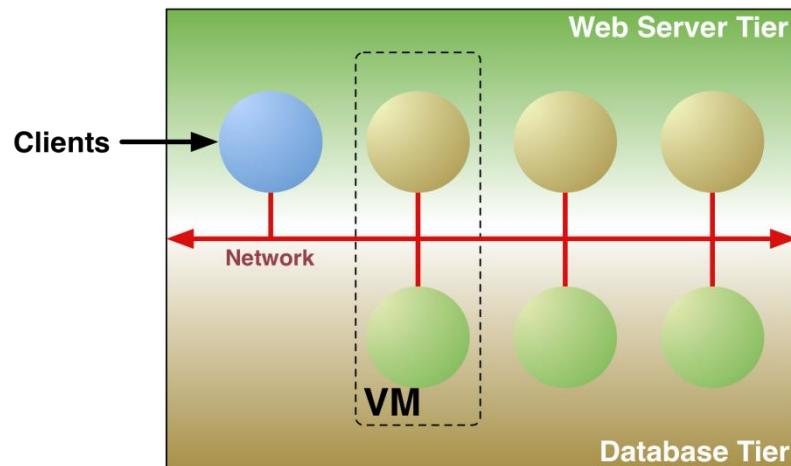
- Dynamic deployment and re-placement of virtual machines on a pool of physical resources
- Transform a rigid distributed physical infrastructure into a **flexible and agile virtual infrastructure**



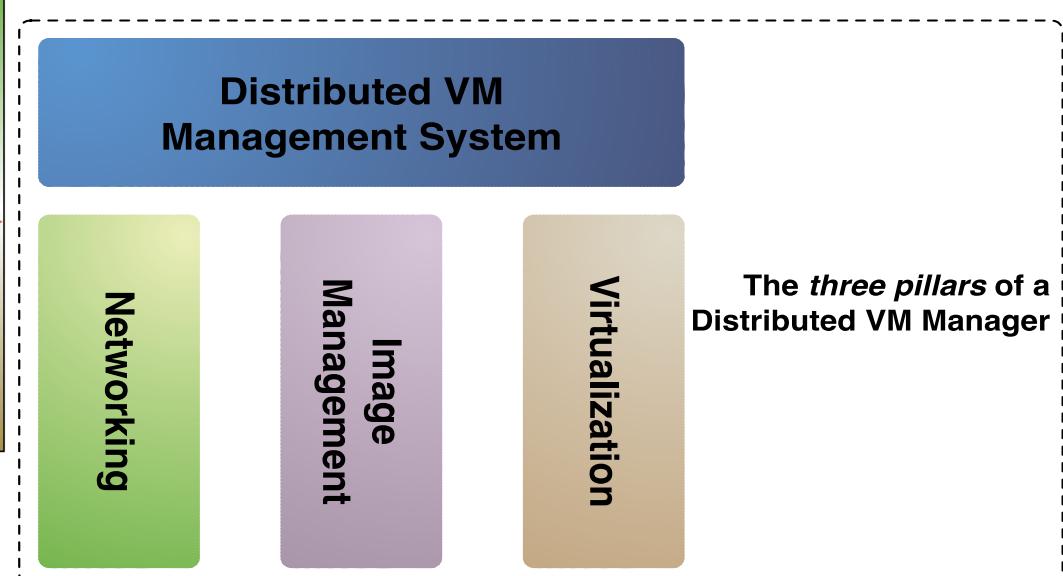
- **Backend of Public Cloud:** Internal management of the infrastructure
- **Private Cloud:** Virtualisation of cluster or data-center for internal users
- **Cloud Interoperation:** On-demand access to public clouds

Virtual Machine Management Model

Service as Management Entity



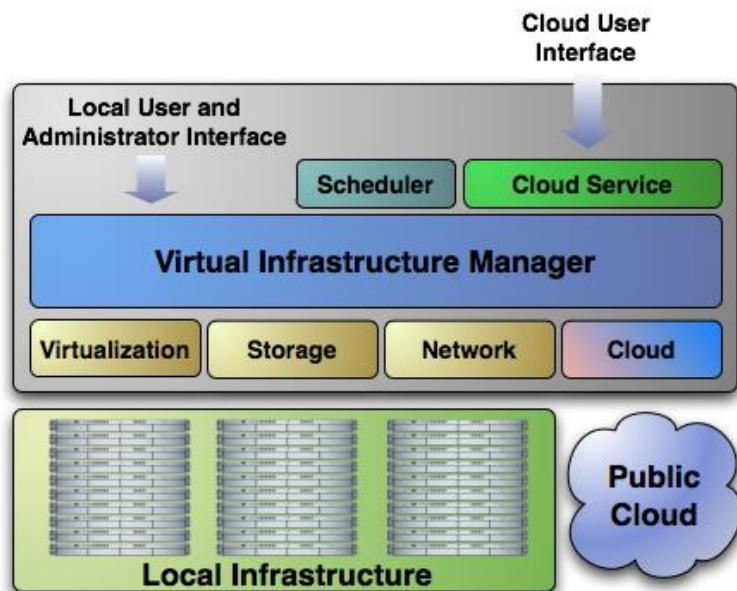
Distributed VM Management Model



- Service structure
 - Service components run in VMs
 - Inter-connection relationship
 - Placement constraints
- The VM Manager is **service agnostic**

The Benefits of VIM

- For the Infrastructure Manager
 - Centralised management of VM workload and distributed infrastructures
 - Support for VM placement policies: balance of workload, server consolidation
 - Dynamic resizing of the infrastructure
 - Dynamic partition and isolation of clusters
 - Dynamic scaling of private infrastructure to meet fluctuating demands
 - Lower infrastructure expenses combining local and remote Cloud resources



VM Management at Scale

The screenshot shows the Microsoft Azure portal interface for managing virtual machines. The main focus is on the 'Virtual machines' blade for a specific VM named 'kdvm1'. The top navigation bar includes links for 'All services', 'Virtual machines', and a search bar. On the left, a sidebar lists other VMs such as 'Faasdnos', 'First-VM', 'kdvm1', 'lab-2', etc. The main content area displays the VM details under the 'Essentials' tab, including its resource group ('uol_feps_soc_comp5850m_scskd'), operating system (Linux), status (Stopped), location (UK South), public IP address (51.132.222.81), subscription ('UoL-Teaching-SOC-MCC'), and subscription ID (66f773f7-ab57-4a46-9c43-c11a2caf7c9d). The 'Properties' tab is selected, showing options like 'Monitoring', 'Capabilities (7)', 'Recommendations', and 'Tutorials'. Below the main content, there are tabs for 'Microsoft Defender for Cloud' and 'Advisor recommendations'. The bottom of the screen shows a taskbar with various open files and a 'Show all' button.

Microsoft Azure

Search resources, services, and docs (G+)

All services > Virtual machines >

Virtual machines

University of Leeds (leeds365.onmicrosoft.com)

Create Switch to classic

Filter for any field...

Name

- Faasdnos
- First-VM
- kdvm1**
- lab-2
- lab-ubuntu-tomcat-ws
- Linux-Virtual-Machine
- linux-vm
- Linux-Container

Page 1 of 1

kdvm1

Virtual machine

Search

Connect Start Restart Stop Capture Delete Refresh

JSON View

Essentials

Resource group (move)	Operating system
uol_feps_soc_comp5850m_scskd	Linux
Status	Size
Stopped (deallocated)	Standard B2ms (2 vcpus, 8 GiB memory)
Location	Public IP address
UK South	51.132.222.81
Subscription (move)	Virtual network/subnet
UoL-Teaching-SOC-MCC	uol_feps_soc_comp5850m_scskd-vnet/defa...
Subscription ID	DNS name
66f773f7-ab57-4a46-9c43-c11a2caf7c9d	Not configured

Tags (edit) More (8)

Properties Monitoring Capabilities (7) Recommendations Tutorials

Microsoft Defender for Cloud

Advisor recommendations

qsort.c

COMP_IPE_2023_C....xlsx

kdvm1.rdp

AAM_Pre_Meeting....pdf

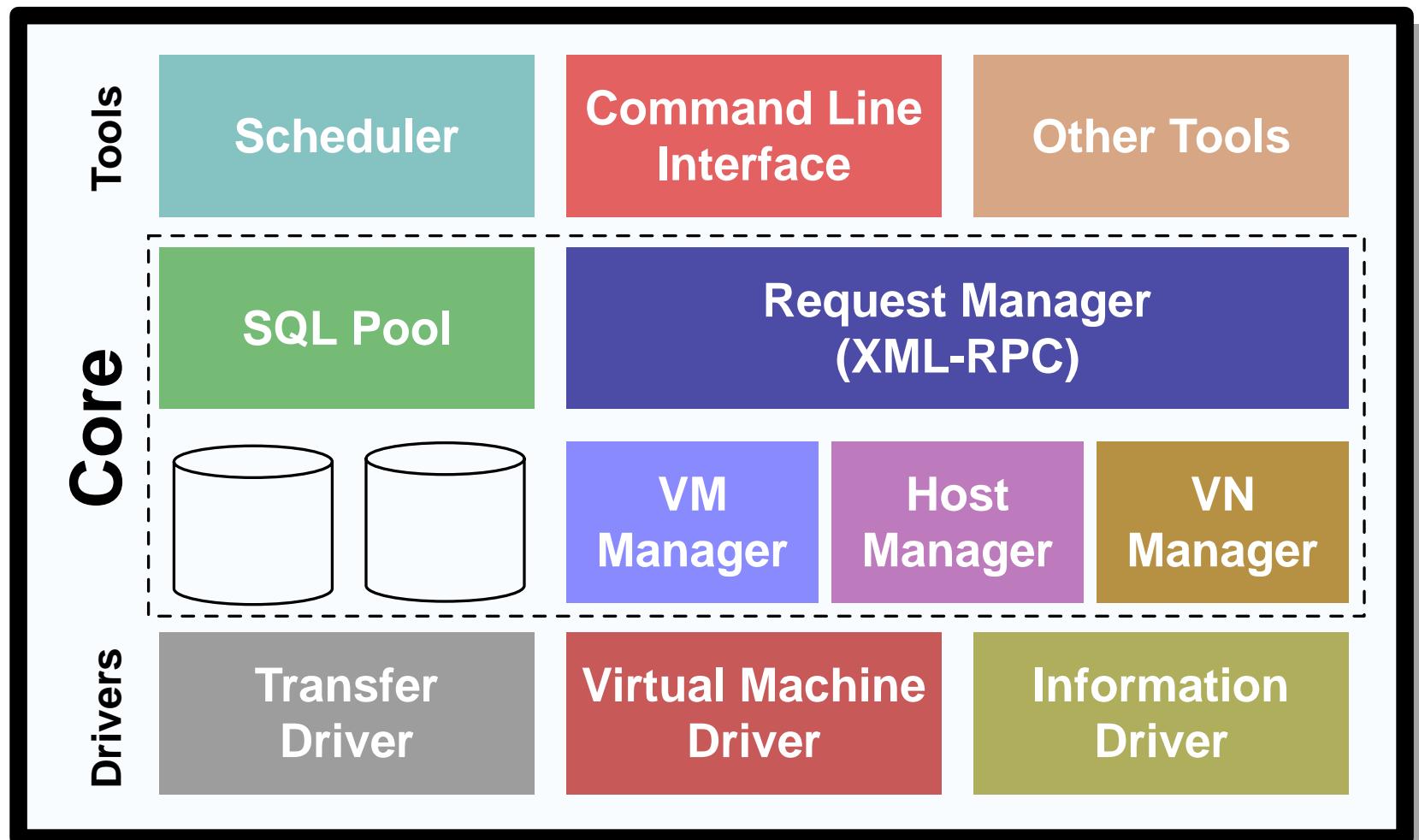
Show all

VIM Main features: OpenNebula

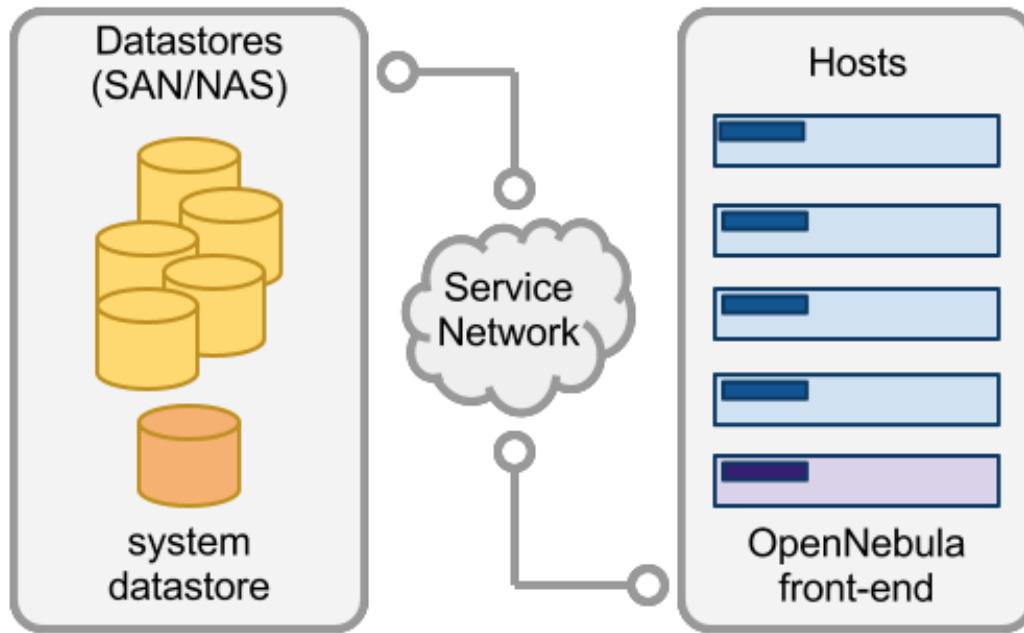
<https://opennebula.io/>

Feature	Function
Internal Interface	<ul style="list-style-type: none">Unix-like CLI for fully management of VM life-cycle and physical boxesXML-RPC API and libvirt virtualization API
Scheduler	<ul style="list-style-type: none">Requirement/rank matchmaker allowing the definition of workload and resource-aware allocation policies
Virtualization Management	<ul style="list-style-type: none">Xen, KVM, VirtualBox, and VMwareGeneric libvirt connector
Image Management	<ul style="list-style-type: none">General mechanisms to transfer and clone VM images
Network Management	<ul style="list-style-type: none">Definition of isolated virtual networks to interconnect VMs
Service Management and Contextualization	<ul style="list-style-type: none">Support for multi-tier services consisting of groups of inter-connected VMs, and their auto-configuration at boot time
Security	<ul style="list-style-type: none">Management of users by the infrastructure administrator
Fault Tolerance	<ul style="list-style-type: none">Persistent database backend to store host and VM information
Scalability	<ul style="list-style-type: none">Tested in the management of medium scale infrastructures with hundreds of servers and VMs (no scalability issues)
Installation	<ul style="list-style-type: none">Installation on a UNIX cluster front-end without requiring new services
Flexibility and Extensibility	<ul style="list-style-type: none">Open, flexible and extensible architecture, interfaces and components, allowing its integration with any product or tool

OpenNebula Architecture



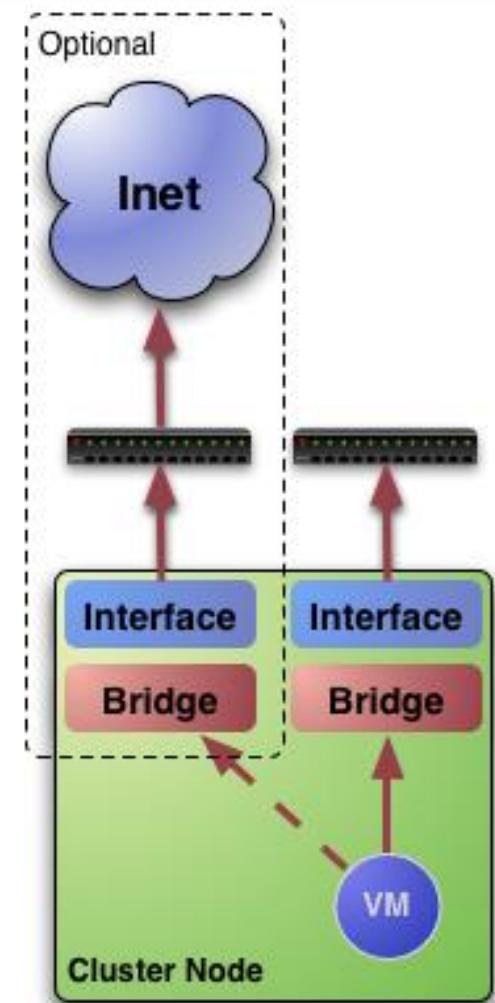
Storage Considerations



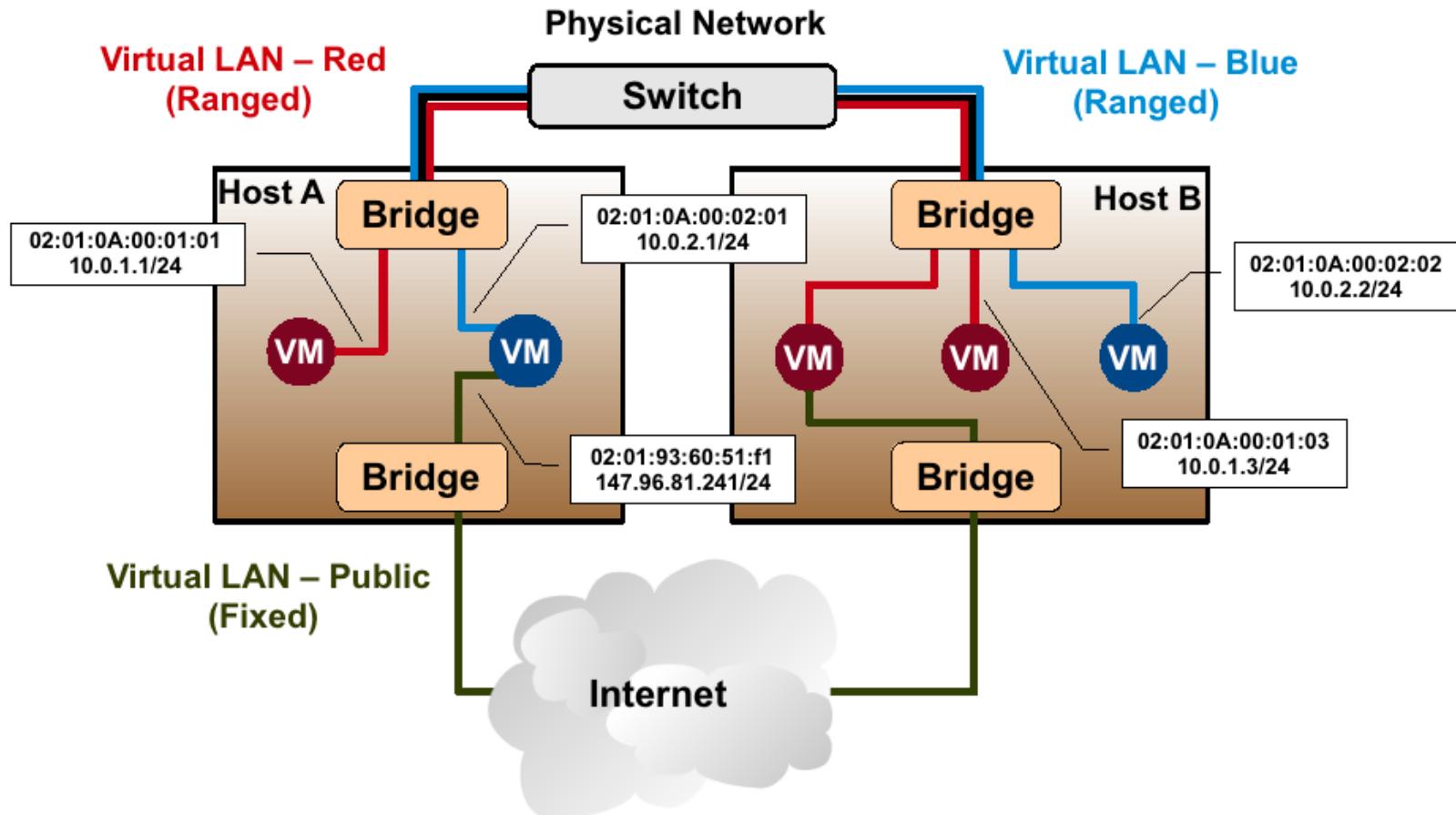
- Virtual machines and their images are represented as files
- Virtual machines and their images are represented as block devices (just like a disk)

Networking Considerations

- Management operations use **ssh** connections
- **Image traffic**, may require the movement of heavy files (VM images, checkpoints)
- **VM demands**, consider the typical requirements of the 1,000 of VMs
 - Network Interface Cards (NICs) to support the VM traffic
- Networking for the VMs, e.g. **bridge** networking.



Network Setup: Example



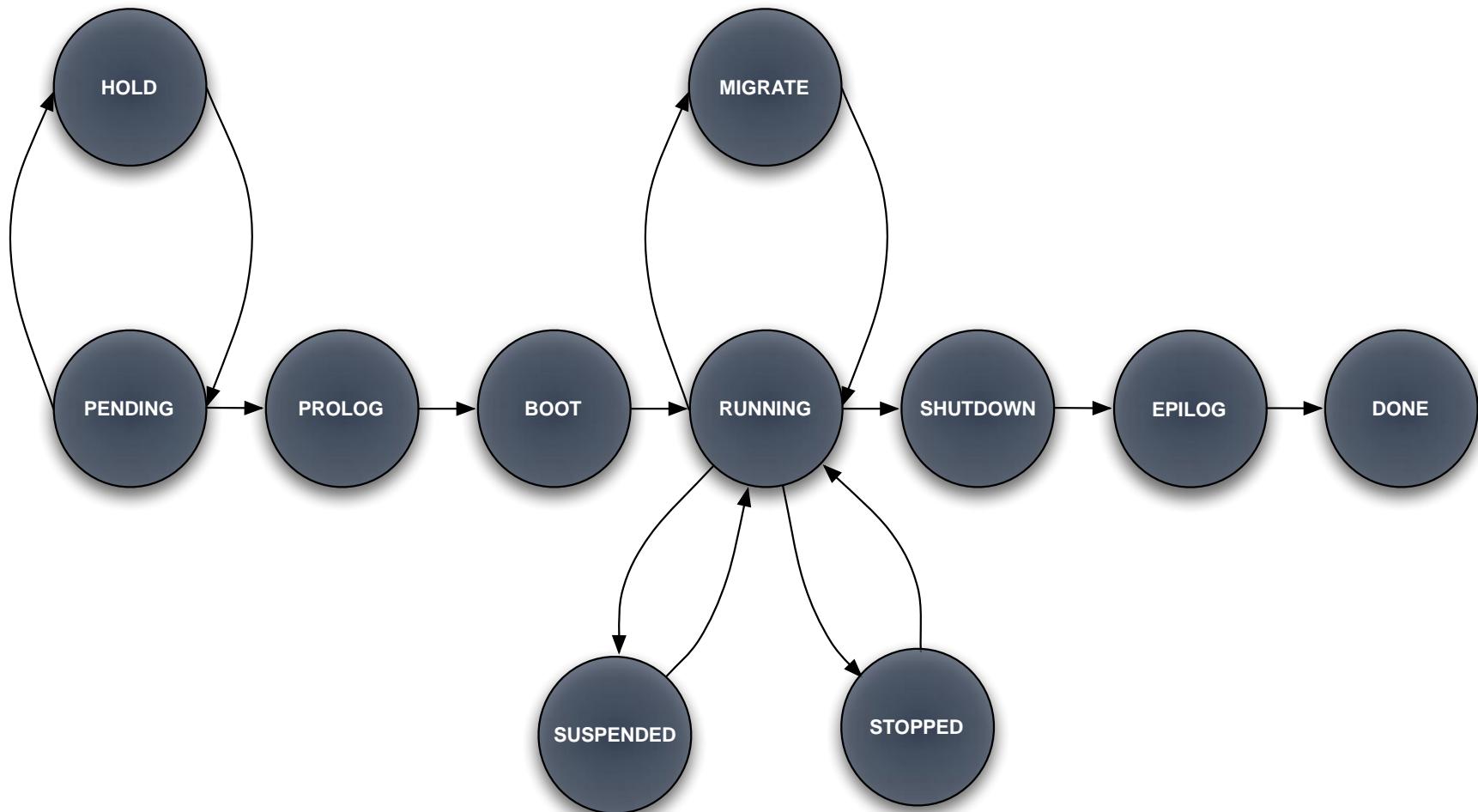
- A **virtual network** defines a separated MAC/IP address space to be used by VMs
- Each virtual network is associated with a **physical network** through a **bridge**

VM Description

Option	Description
NAME	<ul style="list-style-type: none">Name that the VM will get for description purposes.
CPU	<ul style="list-style-type: none">Percentage of CPU divided by 100 required for the Virtual Machine.
OS (KERNEL, INITRD)	<ul style="list-style-type: none">Path of the kernel and initrd files to boot from.
DISK (SOURCE, TARGET, CLONE, TYPE)	<ul style="list-style-type: none">Description of a disk image to attach to the VM.
NIC (NETWORK)	<ul style="list-style-type: none">Definition of a virtual network the VM will be attached to

- Can use any VM prepared for the target hypervisor
- Install once and deploy many; prepare master images

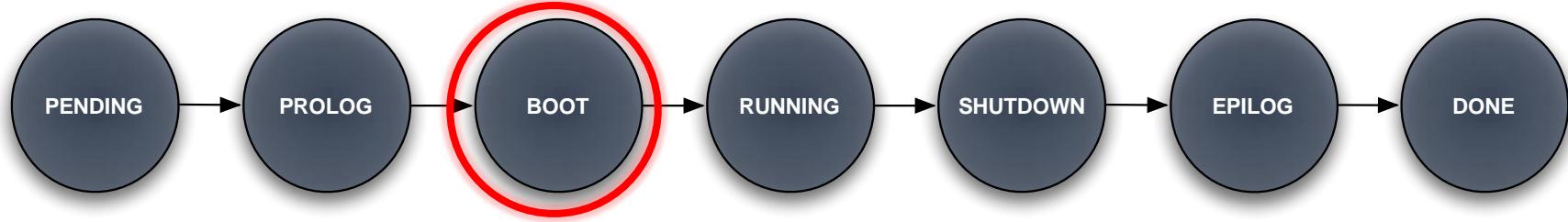
VM States overview



Note: these are VIM dependent

VM States Overview

- After submitting a VM description
 - VM instantiation
 - VM state is set to **PENDING**
- **PROLOG**: the Transfer Driver prepares the images to be used by the VM
- **BOOT**: a deployment file specific for the virtualisation technology configured for the physical host is generated using the information provided in the VM description file.





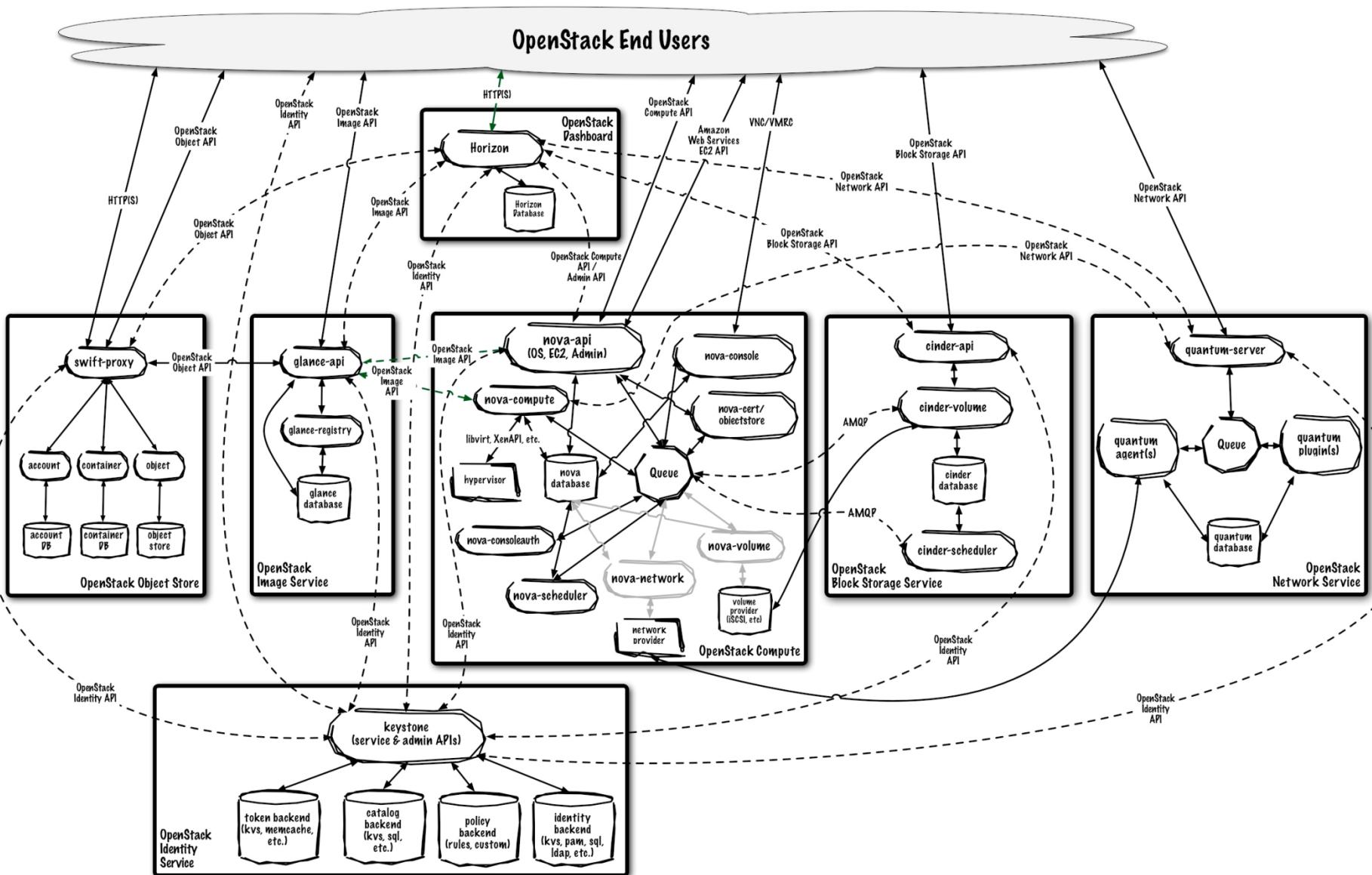
Virtual Infrastructure Manager – Openstack

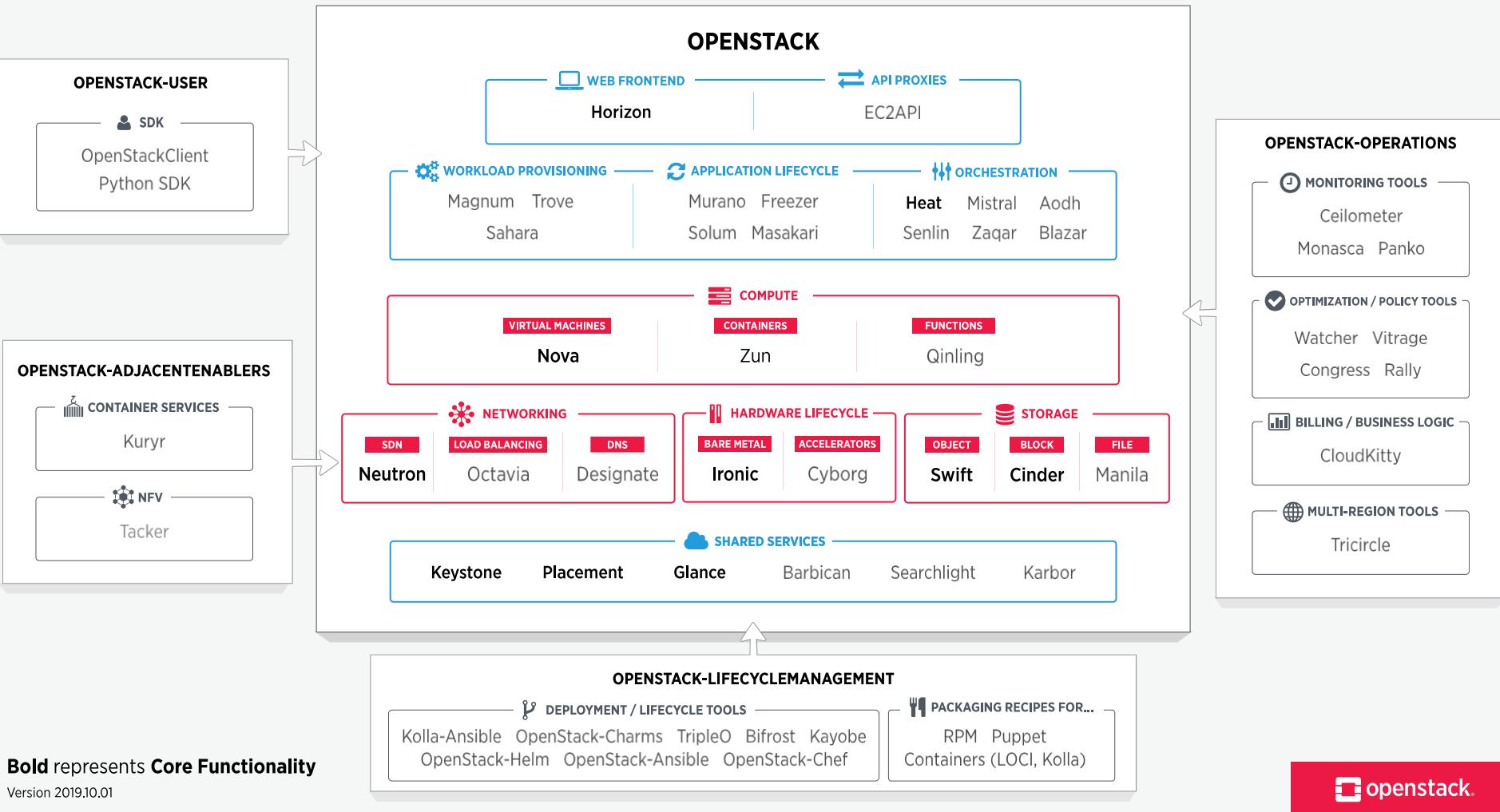
<https://www.openstack.org/>

- Mission: To produce the **ubiquitous open source cloud computing platform** that will meet the needs of public and private clouds regardless of size, by being simple to implement and massively scalable.
- Aim: to be the “Linux” in cloud computing systems
- Management layer that adds automation and control
- Empower administrators and users via service portals
- Empower developers to make apps cloud-aware via APIs
- Enables cloud federation

*OpenStack is the world's leading open-source cloud platform. It is used by hundreds of **local public cloud providers**, **telcos** and thousands of **enterprises**, with over 25 million cores running in production, according to the OpenStack User Survey 2021.*

OpenStack architecture is ... complex!



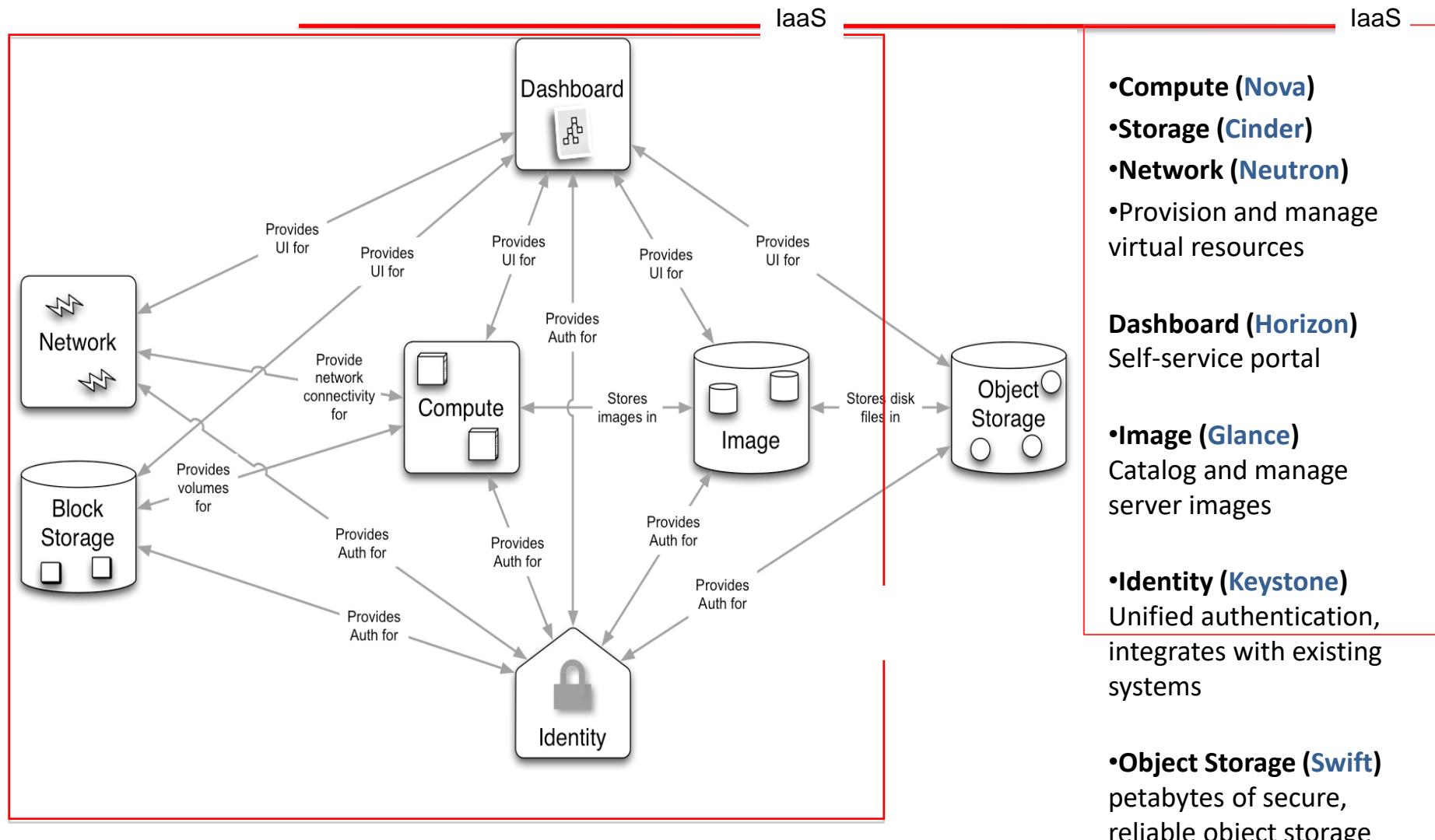


Bold represents **Core Functionality**

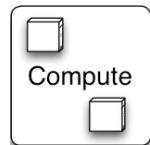
Version 2019.10.01



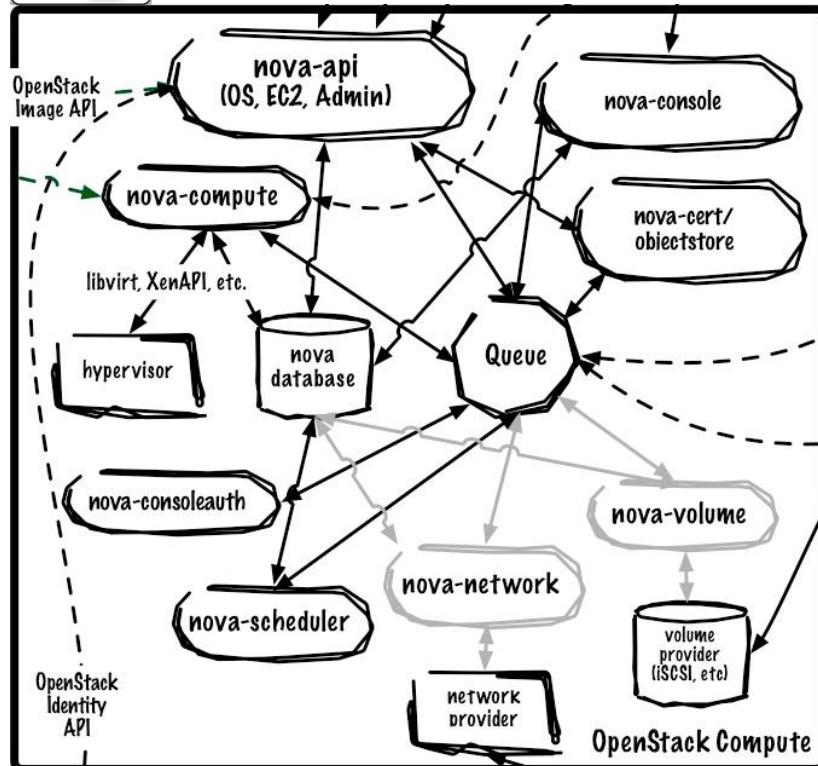
OpenStack is comprised of seven “core” projects that form a complete IaaS solution



Compute delivers a fully featured, and scalable cloud computing platform



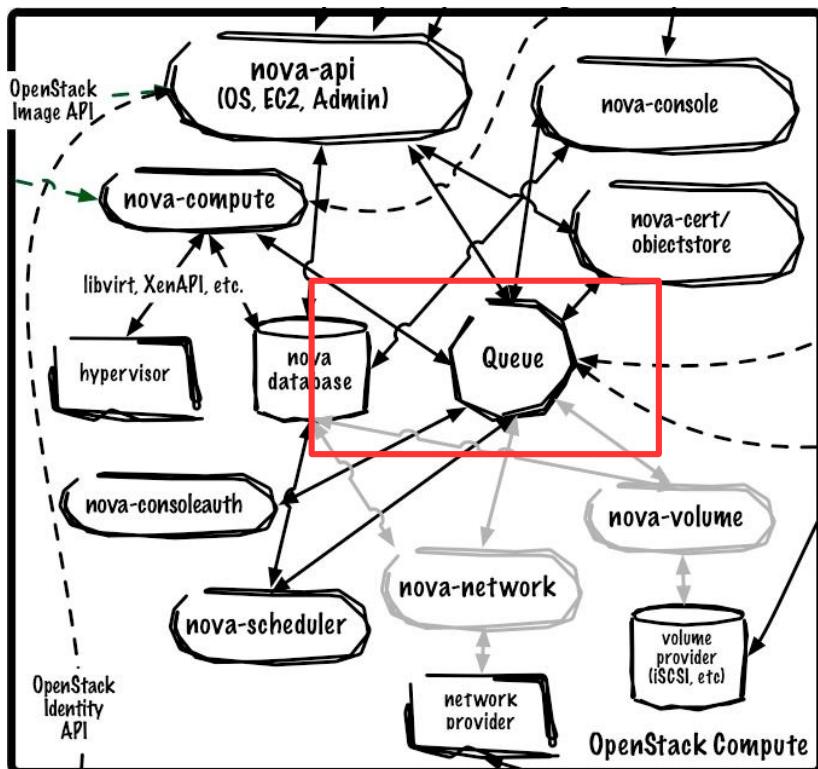
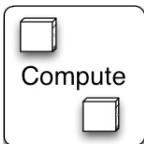
- Architecture



Key Capabilities:

- Manage virtualized server resources
 - CPU/Memory/Disk/Network Interfaces
- API with authentication
- Distributed and asynchronous architecture
 - Massively scalable and highly available system
- Live guest migration
 - Move running guests between physical hosts
- Live VM management (Instance)
 - Run, reboot, suspend, resize, terminate instances
- Security Groups
- Role Based Access Control (RBAC)
 - Ensure security by user, role and project
- Projects & Quotas

Compute management stack control built on queues and database



- Key Capabilities:

- Message Queues: responsible for providing inter-process communications hub and managing data persistence
- AMQP based messaging technology:
 - RabbitMQ is default queue, MySQL DB
 - Single “cell” (1 Queue, 1 Database) typically scales from 500 – 1000 physical machines
<https://www.rabbitmq.com>
- Communications route through queue
 - API requests are validated and placed on queue
 - Workers listen to queues based on role
 - Responses are dispatched back through queue.

Note: OpenStack supports other message queuing service back ends: Qpid and ZeroMQ.

Conclusion

- Presented Virtual Infrastructure Management features
- Introduced two examples: Opennebula, Openstack
- Both provide setting up private and hybrid clouds
- Openstack modular architecture:
 - Seven core projects
 - Other projects exist, e.g.
 - **Heat** for orchestration of multiple composite cloud applications
 - **Mistral** for workflow management
 - **Zun** for launching and managing containers
 - **Ceilometer** for billing systems (resource consumption)
 - **Trove**: Database as a Service
 - **Sahara**: to rapidly provision Hadoop clusters (more in big data lectures)
 - ...

COMP5123M – Cloud Computing Systems



Cloud Resource Management and Scheduling

Plan of the Lecture

Goals

- Understand concepts of resource management and scheduling in clouds

Overview

- Introduction
- Cloud Resource Management
- Scheduling Approaches
- Live Migration Issues
- Scaling Up/Out
- Example
- Conclusion

Architectural Layer

User level



User-Level
Middleware

Cloud applications

Social computing, Enterprise, Scientific, ...

Cloud programming: environments and tools

Web 2.0 Interfaces, Mashups, Concurrent and Distributed Programming, Workflows, Libraries, Scripting

Apps Hosting Platforms

Core
Middleware

QoS Negotiation, Admission Control, Pricing, SLA Management, Monitoring, Execution Management, Metering, Accounting, Billing

Virtual Machine (VM), VM Management and Deployment

System level

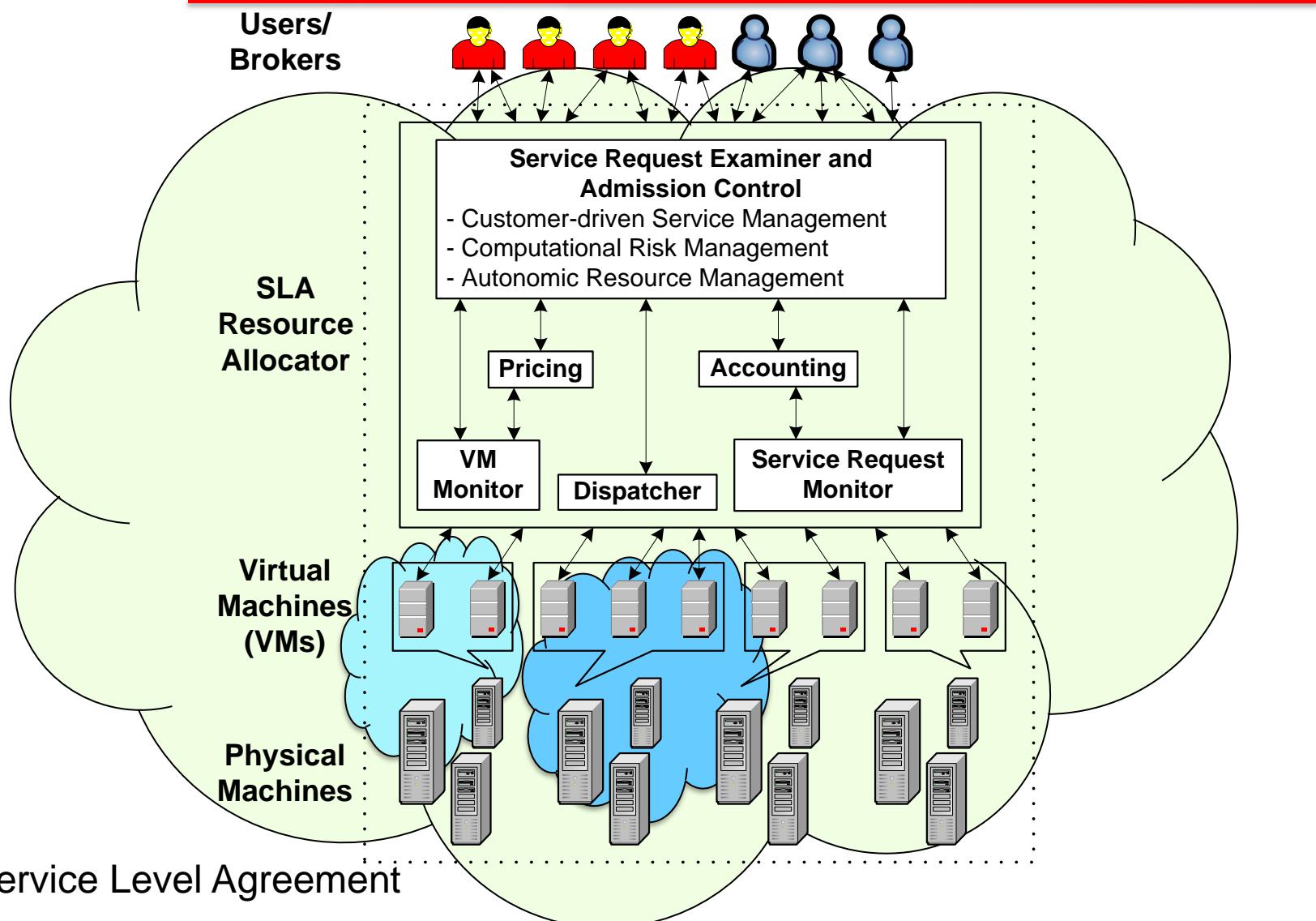
Cloud resources



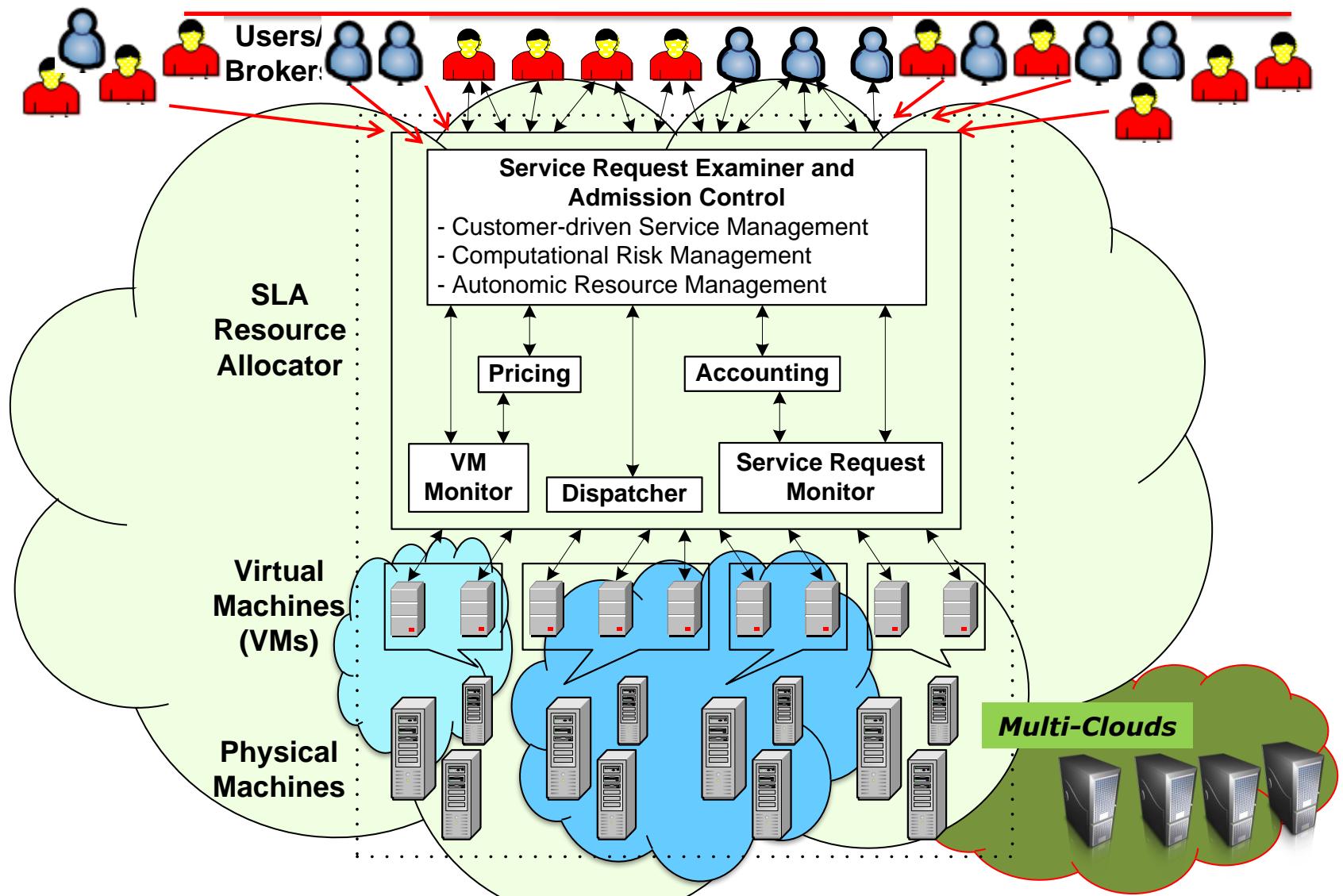
Resource Management and Scheduling

- Critical function of any man-made system.
- Affects the three basic criteria for the evaluation of a system:
 - Functionality
 - Performance
 - Cost
- **Scheduling** in a computing system: deciding how to allocate resources of a system, such as hosts, CPU cycles, memory, secondary storage space, I/O and network bandwidth, between users and tasks.
- **Policies and mechanisms** for resource allocation.
 - Policy: principles guiding decisions
 - Mechanisms: the means to implement policies.

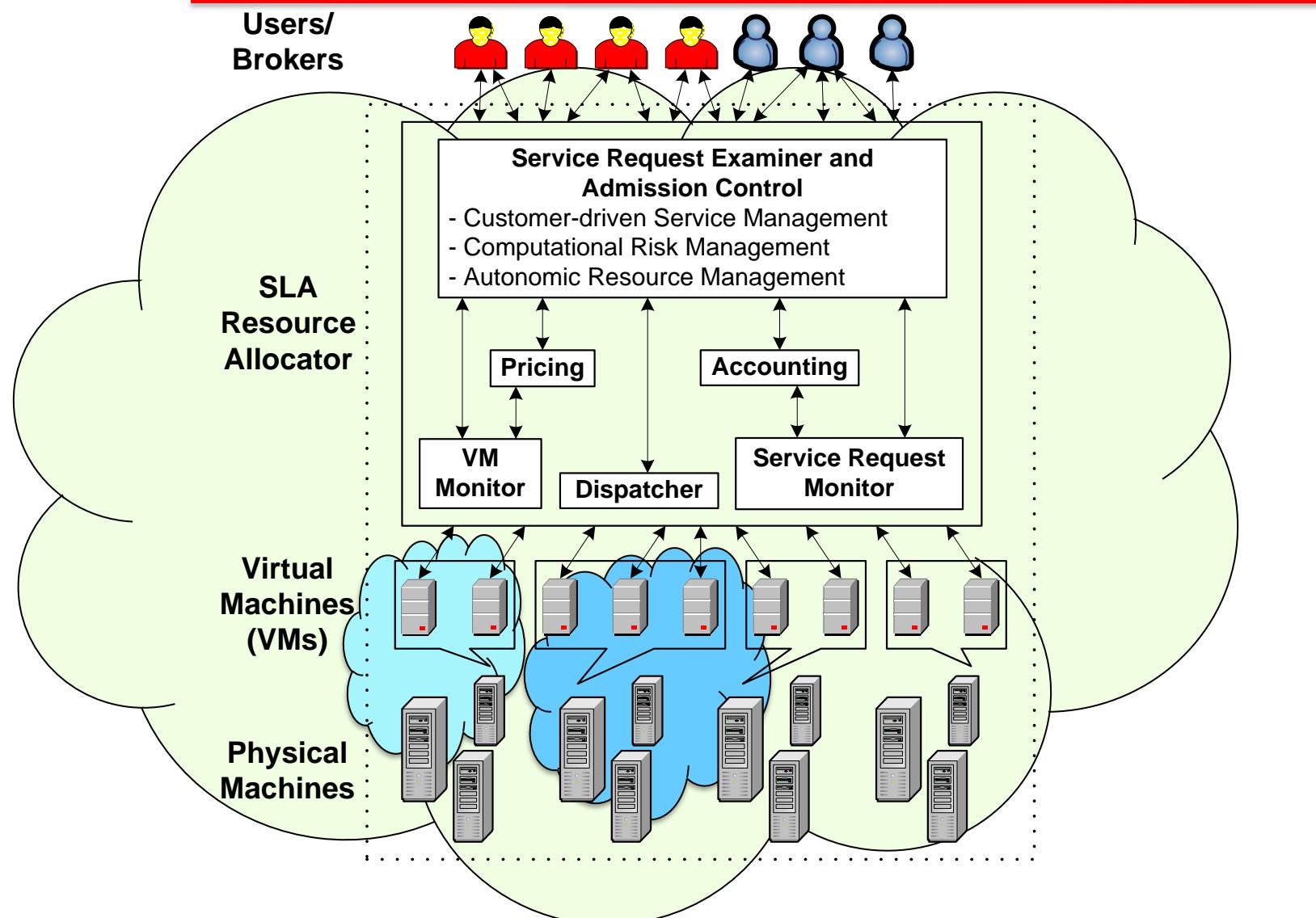
VM Resource Management



Cloud to Expand with Increase in Resource Demand

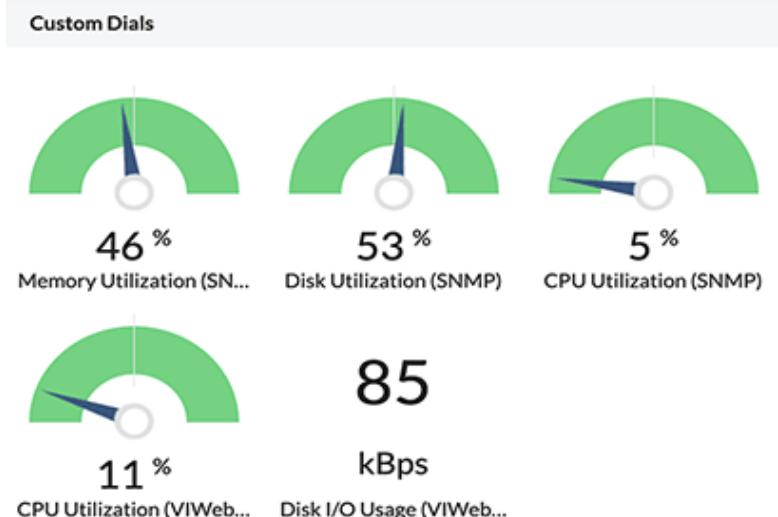


Cloud to Shrink with Decrease in Resource Demand



The Problem

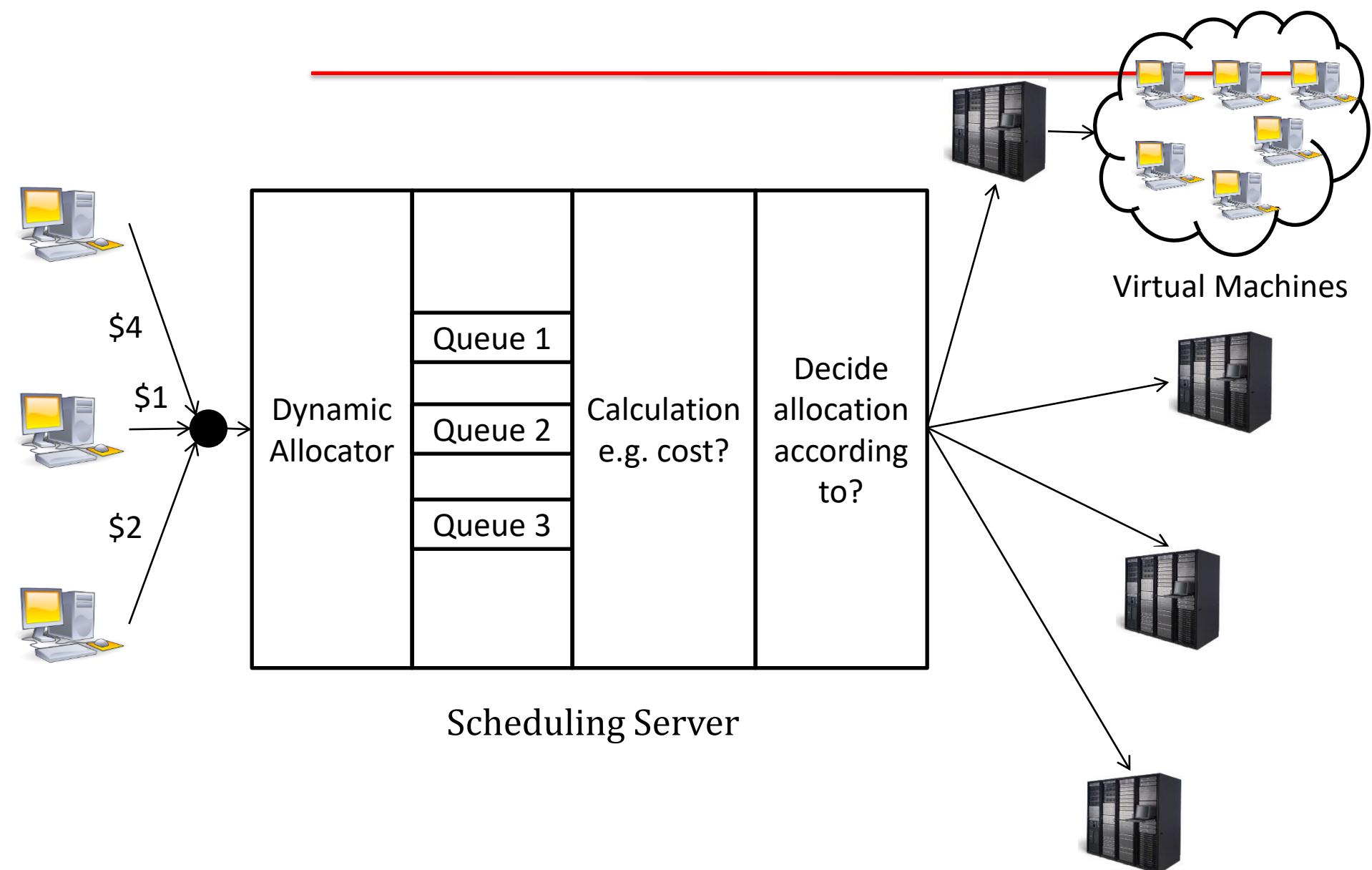
- Cloud resource management
 - Requires complex policies and decisions for **multi-objective optimisation**
 - Is challenging - the complexity of the system makes it impossible to have **accurate** global state information
 - Affected by **unpredictable** interactions with the environment, e.g., system failures, attacks
 - Cloud service providers are faced with large **fluctuating loads** which challenge the claim of cloud elasticity.



Cloud Resource Management Policies

- **Admission control:** prevent the system from accepting workload in violation of high-level system policies
- **Capacity allocation:** allocate resources for individual activations of a service
- **Load balancing:** distribute the workload evenly among the servers
- **Energy optimisation:** minimization of energy consumption
- **Quality of service (QoS) guarantees:** ability to satisfy timing or other conditions specified by a Service Level Agreement (SLA).

Overall Scenario

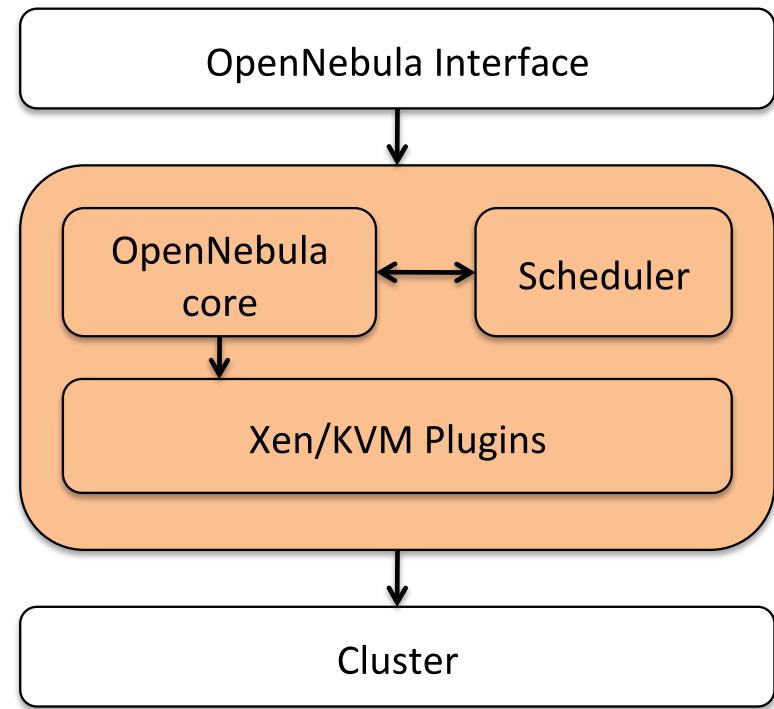


Cloud Scheduling

- Scheduling: responsible for resource sharing at several levels:
 - A Physical Host can be shared among several Virtual Machines
 - A Virtual Machine could support several applications
 - An application may consist of multiple threads
- Objectives of a scheduler:
 - Batch system: maximise throughput and minimise turnaround time
 - Real-time system: meet the deadlines and be predictable
- Best effort: batch applications and analytics
- Common algorithms for best effort applications:
 - Round-Robin (RR)
 - First-Come-First Serve (FCFS)
 - Shortest-Job-First (SJF)
 - Priority Algorithms

VM Scheduling in OpenNebula

- The Scheduler is in charge of the assignment between pending VMs and known Hosts
 - **match making** scheduler
 - implements the **Rank Scheduling Policy**
 - Goal of the policy is to prioritise those resources more suitable for the VM.

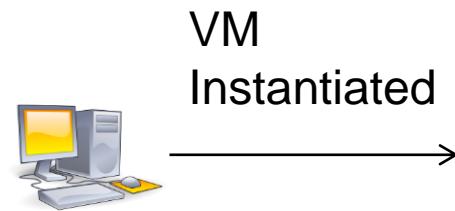


VM Scheduling Approaches: Examples

- **Power-aware**: the focus is how to minimise the power consumed by Physical Hosts in order to maximise revenues by cloud providers
- **Performance-aware**: the focus is on maintaining service levels while improving resource utilisation via dynamic placement mechanism
- **Network-aware**: the focus is how to minimise the effect of transfer time of data between VM instances and data storage
- **Heuristics**: these include First Come First Served (FCFS), Greedy and Round Robin mechanisms. For example, FCFS allocates a VM to the first Physical Host that can accommodate it employing the First Fit (FF) heuristic.



VM Scheduling Example: First Fit



VM
Instantiated

VM Scheduler:
How to allocate the VM?

VM Requirements:
4 vCPU, 16GB RAM

Node 1



	Total	Used
vCPU	16	14
Memory	48	24

	Total	Used
vCPU	16	8
Memory	48	16



Node 2

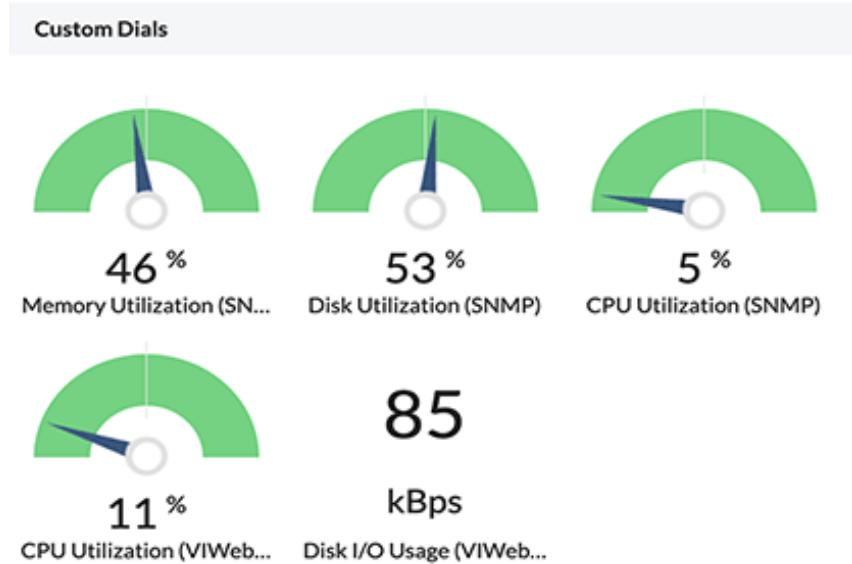
	Total	Used
vCPU	32	24
Memory	96	64

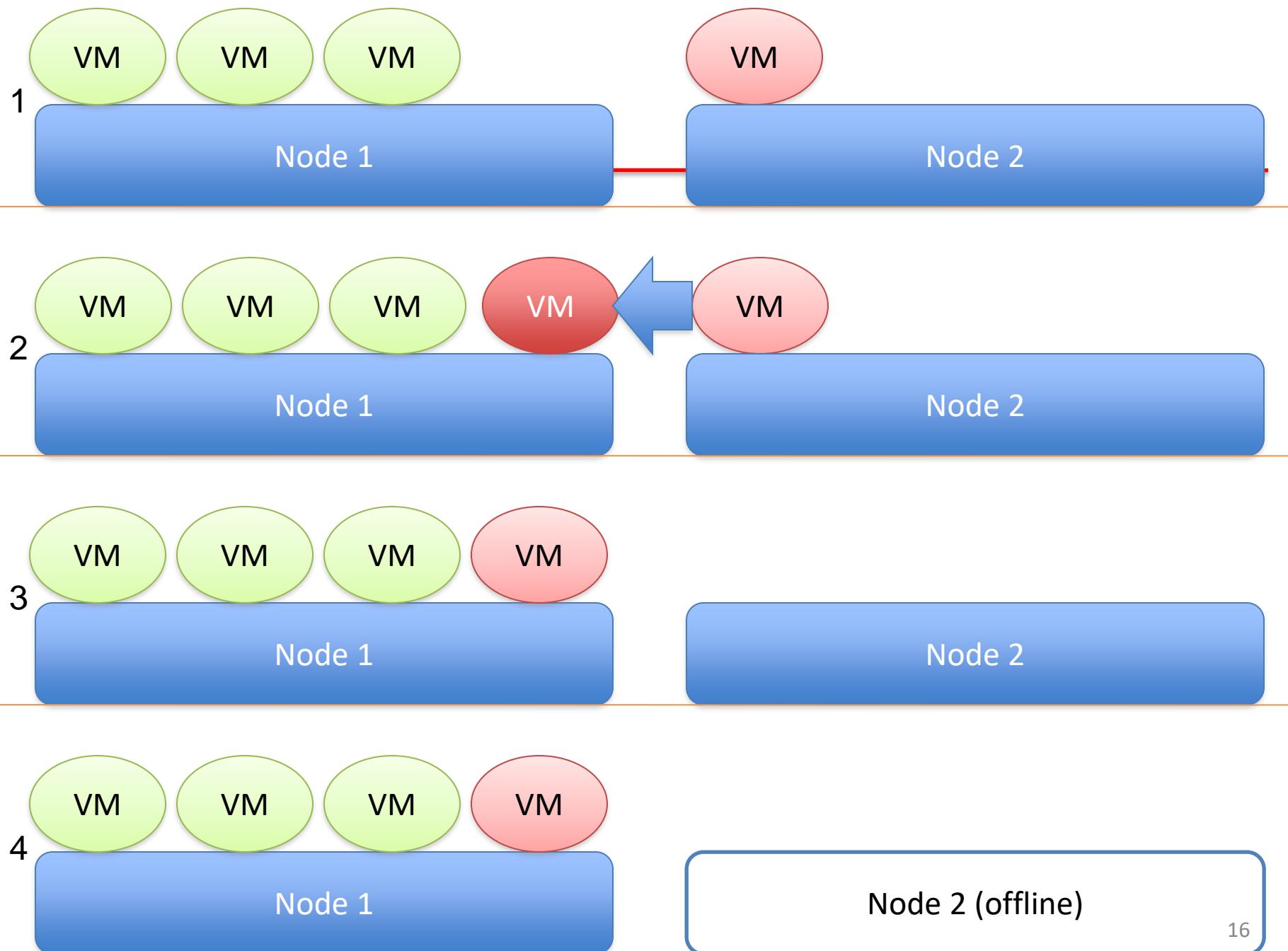


Node 3

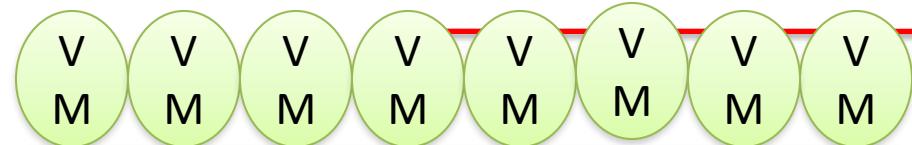
VM Management: Examples

- Monitor Cloud usage and load
- When load **increases**:
 - Start up waiting nodes
 - Schedule new VMs to new nodes.
- When load **decreases**:
 - Live migrate VMs to more utilised nodes
 - Shutdown unused nodes
 - This is server consolidation.





Example: Power-aware 485 Watts vs. 552 Watts



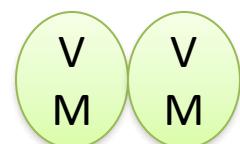
Node 1 @ 170W

Node 2 @ 105W

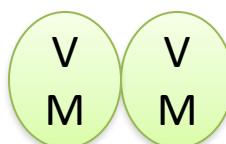
Node 3 @ 105W

Node 4 @ 105W

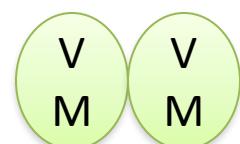
VS.



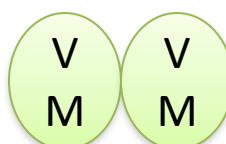
Node 1 @ 138W



Node 2 @ 138W



Node 3 @ 138W



Node 4 @ 138W

Power-Aware Algorithms

- Host overload detection
 - Adaptive utilisation **threshold** based algorithms
 - Median Absolute Deviation (MAD) algorithm
 - Regression based algorithms
 - Local Regression (LR) algorithm
- Host underload detection algorithms
 - Migrate the VMs from **the least utilised host**
- VM selection algorithms
 - Minimum Migration Time (MMT) policy
 - Random Selection (RS) policy
- VM placement algorithms
 - **Heuristic for the bin-packing problem (Power-Aware Best Fit)**

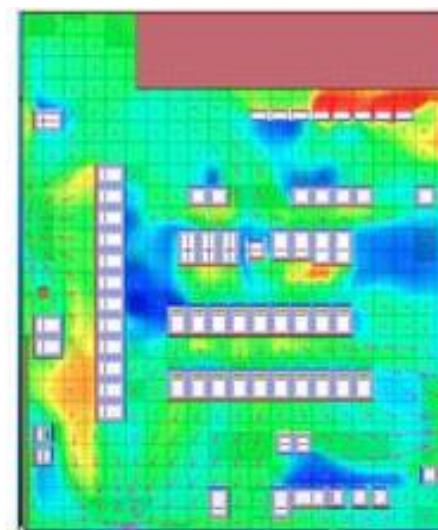
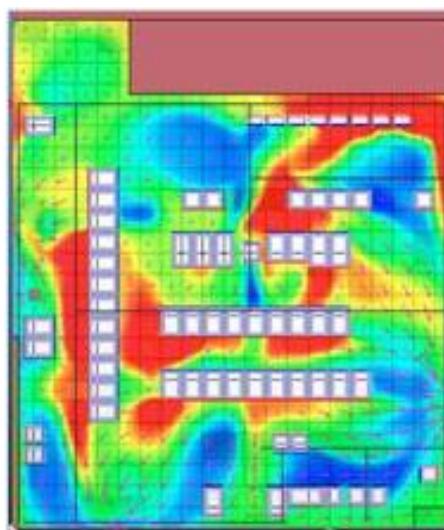
Virtualisation and Scheduling

Virtualisation

- Smarter planning and allocation of resources

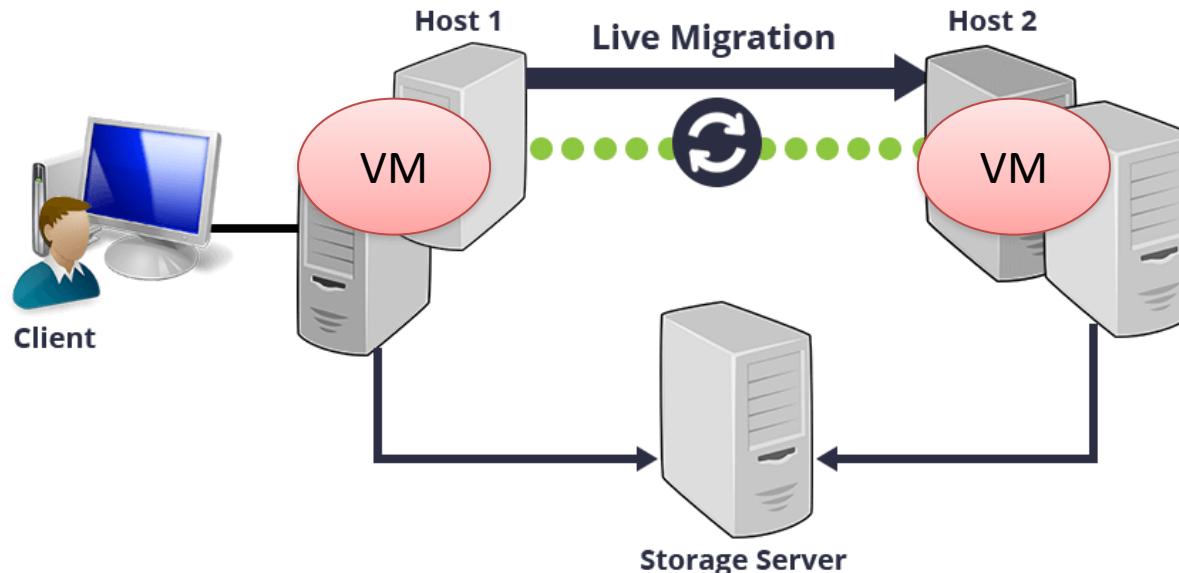
Scheduling and Migration

- Allow to avoid/minimise data centre hotspots
- Migration requires additional cost and energy



VM Migration

- Transfer executing VMs between physical hosts without disconnecting the client or application
- Memory, storage and network connectivity of the VM are transferred from the original host to the destination
- Key operation in cloud resource provisioning
 - Commonly used for planned activities
 - e.g. machine maintenance



The Problem ...

- **When** to migrate VMs?

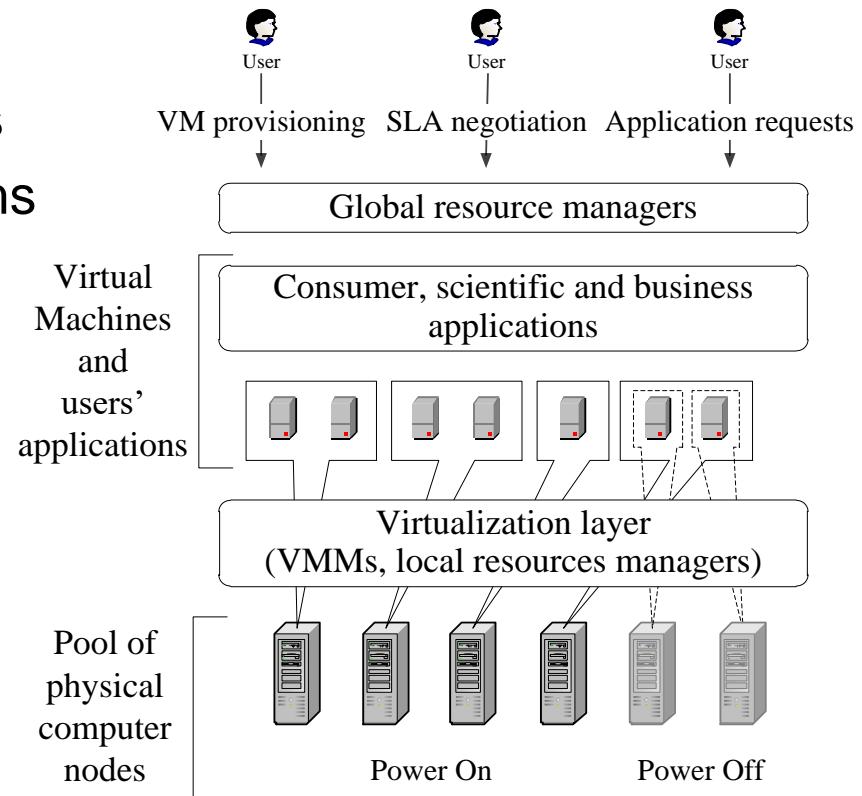
- Host overload detection algorithms
- Host underload detection algorithms

- **Which** VMs to migrate?

- VM selection algorithms

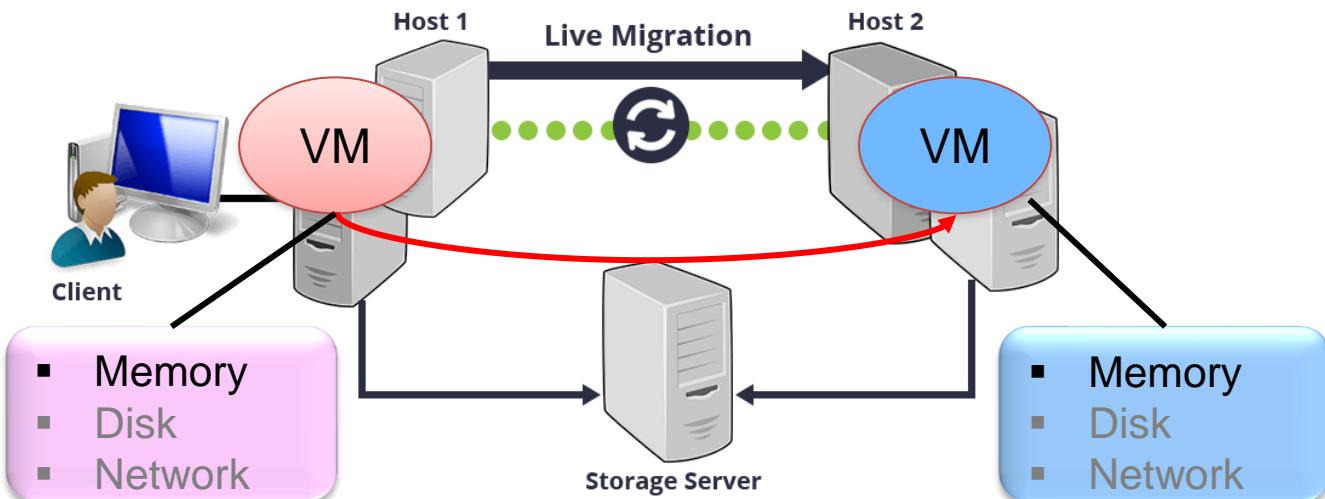
- **Where** to migrate VMs?

- VM placement algorithms



Issues: Memory, Network, Storage

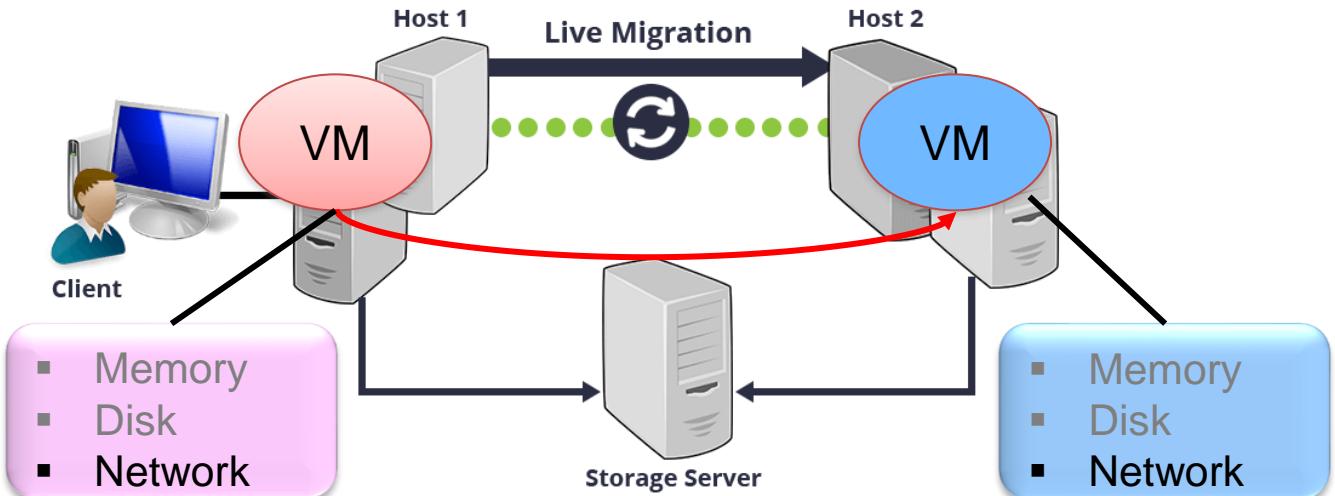
- Several replication and migration approaches exist to achieve fault tolerance and high availability
- Copy of a VM implies the complete state from the original VM should be transferred to the copy including memory, disk, and network connections
- Memory state Migration Methods
 - **Precopy**: first copies the memory state to the destination, through a repetitive process, after which its processor state is transferred to the target
 - **Postcopy**: transfers a VM's memory contents after its processor state has been sent to the target host



Issues: Memory, Network, Storage (2)

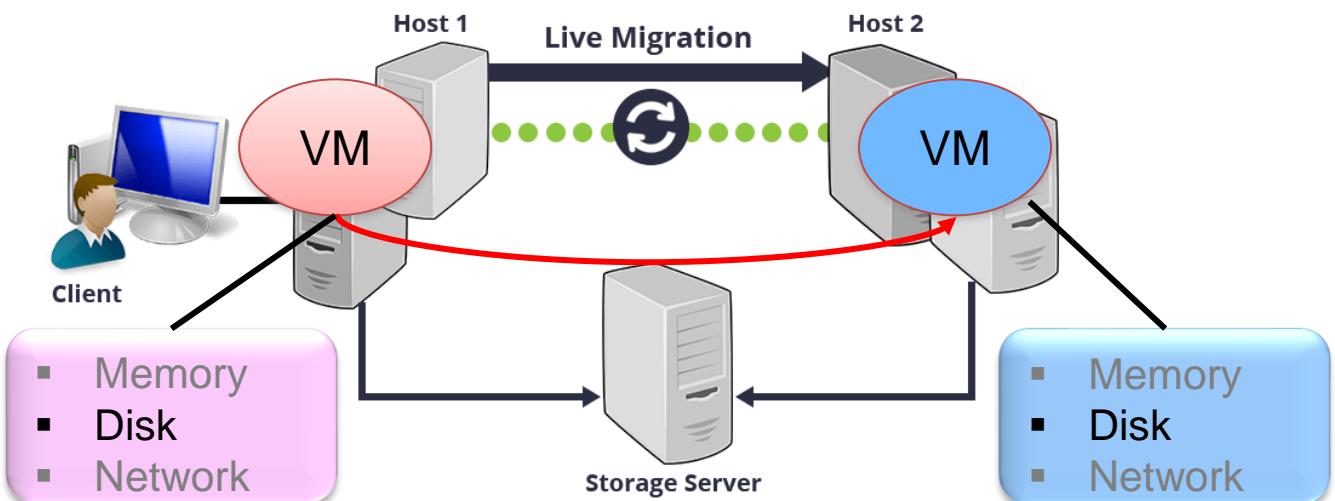
Network Migration Methods

- To maintain network connectivity after migration, it is necessary to preserve open connections
- Network clients should be attended with minimal disruption
- Migration is within the same LAN (Local Area Network)
 - a VM should retain its original IP address after migration, by generating an unsolicited Address Resolution Protocol (ARP) reply advertising the new location for the migrated VM's IP
- Migration is over a WAN (Wide Area Network)
 - the use of network technologies such Virtual Private Networks (VPNs), tunnelling and Domain Name System (DNS) servers is useful



Issues: Memory, Network, Storage (3)

- Disk state migration is sometimes not considered
 - it is assumed that a Storage Area Network (SAN) or Network Attached Storage (NAS) strategy is used among VMs
 - but a WAN environment should consider the migration of local disk.
- Migrating disk state typically represents the **largest component** of the overall migration time
 - the disk state may be in the tens or hundreds of gigabytes
- Many strategies to migrate storage are used
 - e.g. Distributed Replicated Block storage solution

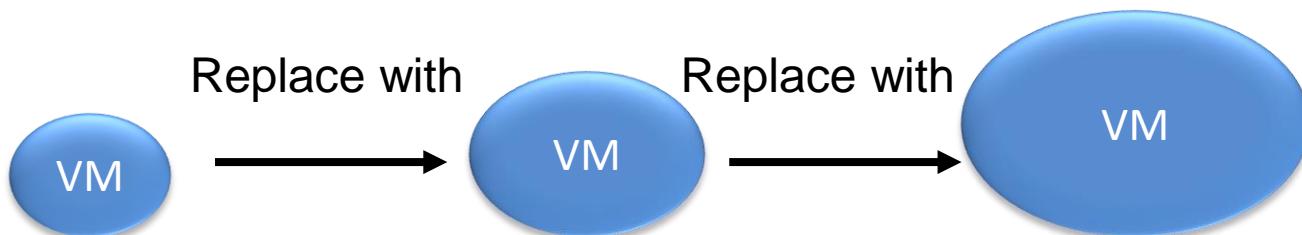


VM Scale Up or Scale Out?

- Scaling Out: getting additional VM(s) to spread application processing load



- Scaling Up: upgrading the current hardware components within the VM, or replacing the VM itself with a better performance / more expensive version



Conclusion

- Reviewed important aspects of cloud resource management and scheduling
- A large number of VM scheduling approaches are found in the literature
 - Still an important research problem
- Issues with VMs migration

References

- M. Ibrahim et al. (2020). An In-Depth Empirical Investigation of State-of-the-Art Scheduling Approaches for Cloud Computing. In IEEE Access, vol. 8, pp. 128282-128294.
<https://ieeexplore.ieee.org/document/9133407>
- Zhi-Hui Zhan, Xiao-Fang Liu, Yue-Jiao Gong, Jun Zhang, Henry Shu-Hung Chung, and Yun Li (2015). Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches. *ACM Computing Surveys*. 47, 4, Article 63.
- V. Medina and J. M. Garcia. A survey of migration mechanisms of virtual machines (2014). *ACM Computing Surveys (CSUR)*, Volume 46 Issue 3, Article No. 30
<https://dl.acm.org/citation.cfm?id=2492705>

COMP5123M – Cloud Computing Systems



Container Management - Kubernetes

Plan of the Lecture

Goals

- Understand the concepts of container orchestration

Overview

- Kubernetes: what is it?
- Definitions
- Architecture
- Key concepts
- Pods and containers
- Auto-scaling
- References

Cloud Resources: Virtualisation Layer

User level

Cloud applications



User-Level
Middleware

Cloud programming: environments and tools

Apps Hosting Platforms

Core
Middleware

Workload and Services Management, Execution Management

Virtual Machine (VM), VM Management and Deployment



System level

Cloud resources



Kubernetes

Kubernetes

κυβερνήτης: Greek for "pilot" or "helmsman of a ship"
the open source cluster manager from Google



- A declarative language for launching containers
- A highly collaborative open source project originally conceived by Google
 - Google has 10+ years experience with containerised apps

Sometimes called:

- kube
- k8s (that's 'k' + 8 letters + 's')
- Start, stop, update, and manage a cluster of machines running containers in a consistent and maintainable fashion

Kubernetes

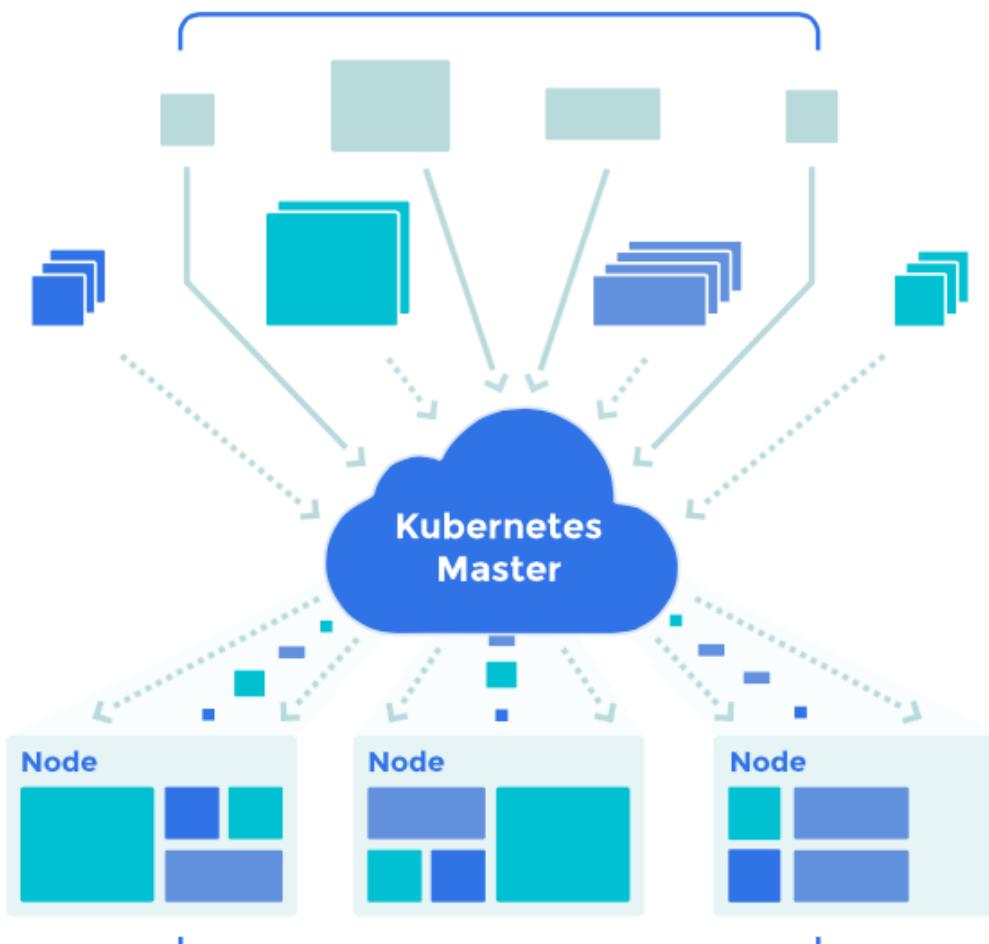
Kubernetes

κυβερνήτης: *Greek for "pilot" or "helmsman of a ship"*
the open source cluster manager from Google



- Docker application orchestration
- Google Cloud, Rackspace, Azure providers
- Easily deployable on OS
- Container replication.

An ocean of
user containers



Scheduled and packed
dynamically onto nodes

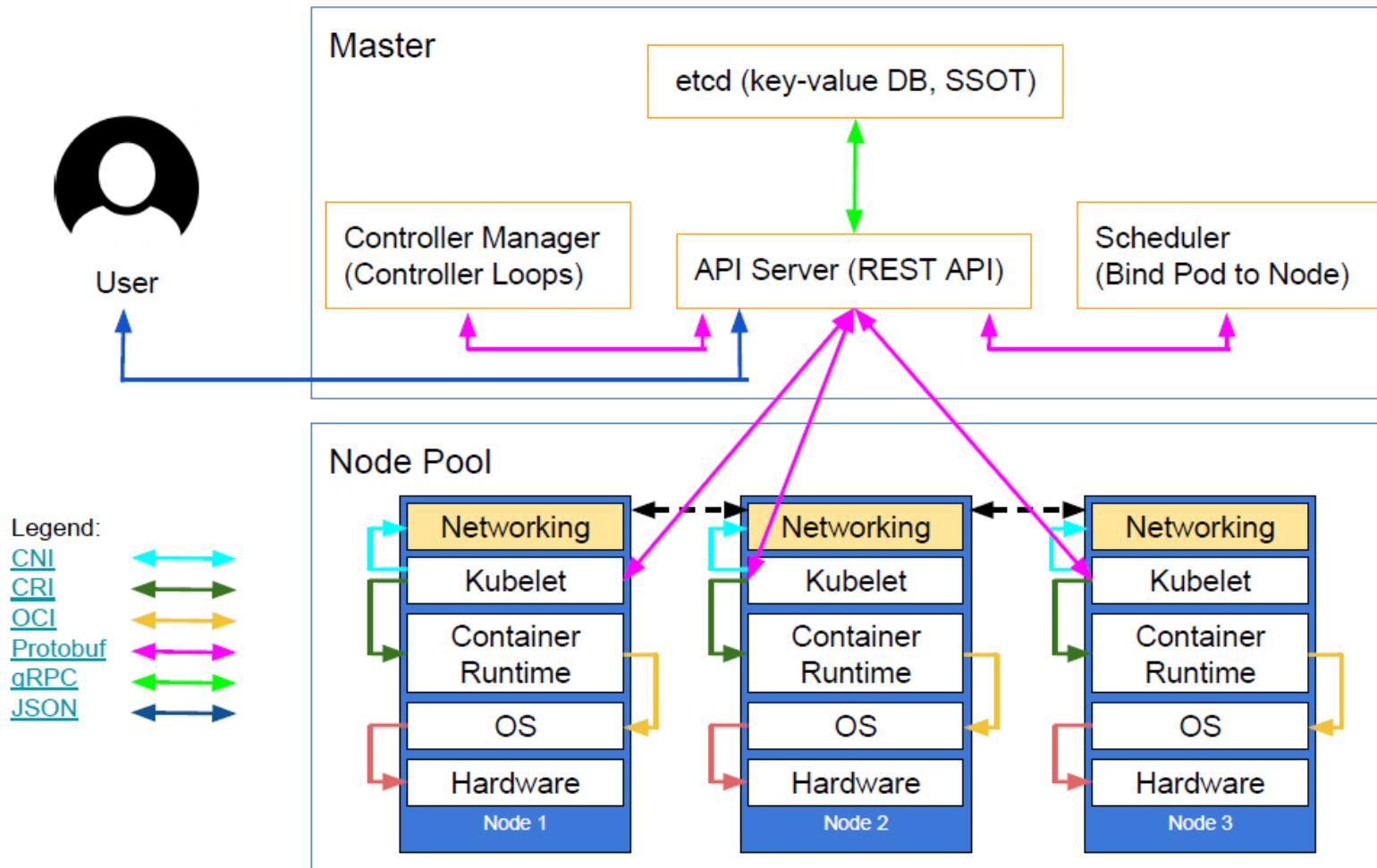
Kubernetes Role

- Improves **reliability**
 - Continuously monitors and manages your containers
 - Will scale your application to handle changes in load
- Better use of **infrastructure resources**
 - Helps reduce infrastructure requirements by gracefully scaling up and down your entire platform
- Coordinates what containers run where and when across your system
 - How do all the different types of containers in a system talk to each other?
- Easily **coordinate** deployments of your system
 - Which containers need to be deployed
 - Where should the containers be deployed

Kubernetes - Definitions

- **Cluster:** Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit. Consists of:
 - The **Master** coordinates the cluster
 - The **Nodes** are the workers that run applications
- **Pod**
 - smallest deployable unit of compute
 - consists of one or more containers that are always co-located, co-scheduled & run in a shared context

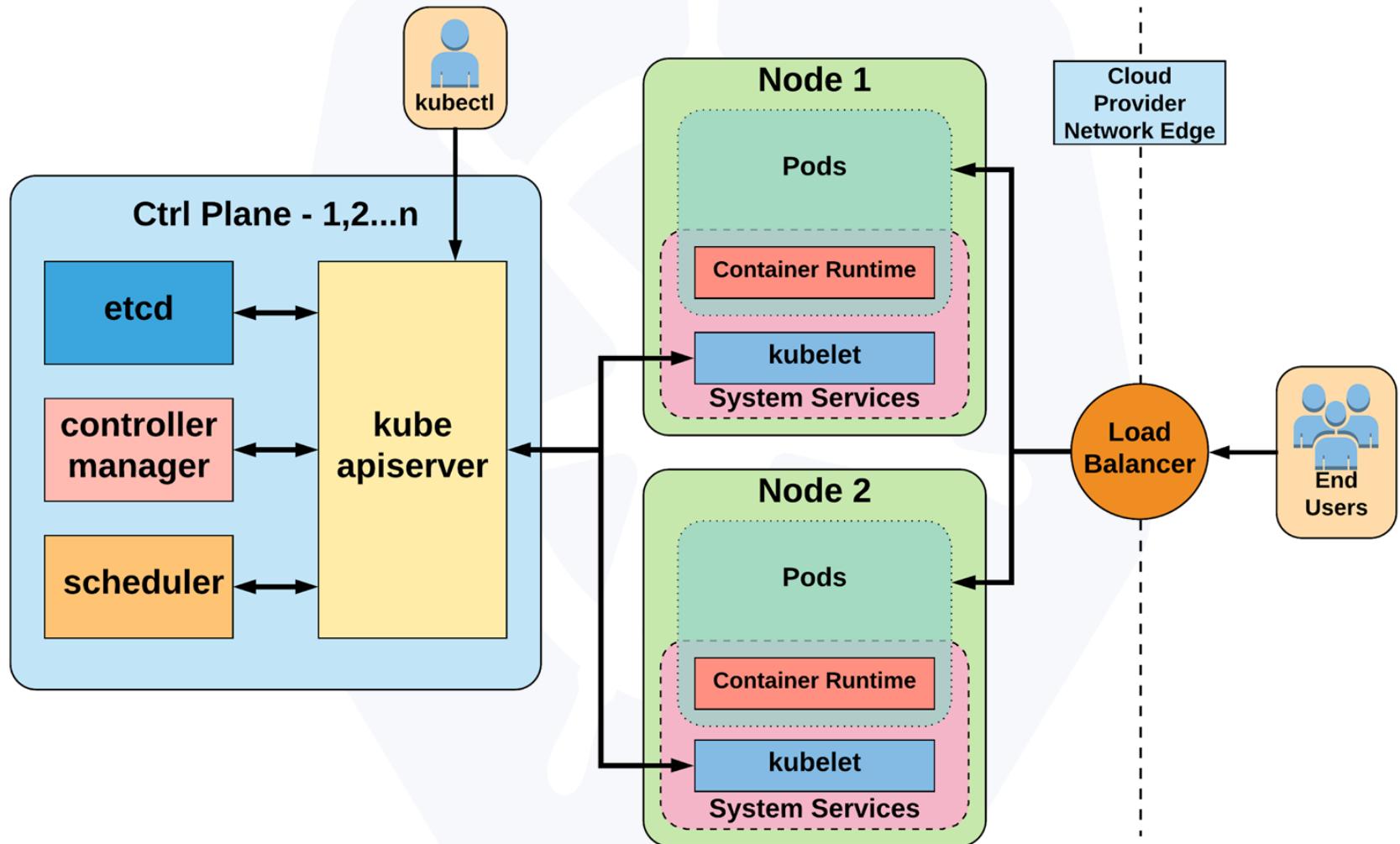
High-Level Architecture



CNI (Container Network Interface)
CRI (Container Runtime Interface)
OCI (Open Container Initiative)

etcd: open source, distributed key-value database from CoreOs. Acts as the Single Source of Truth (SSOT) for all components

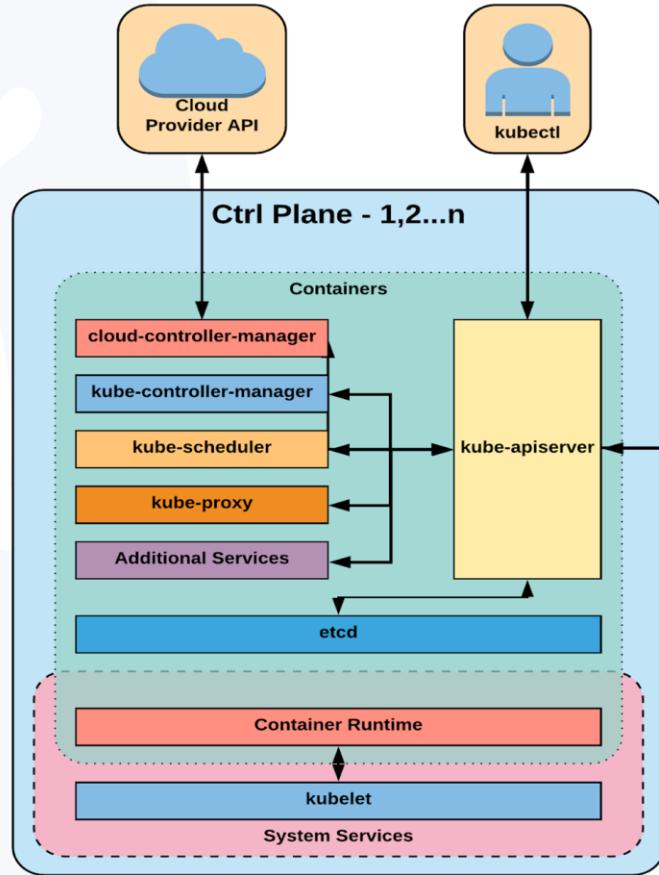
High-Level Architecture (2)



Control Plane Components

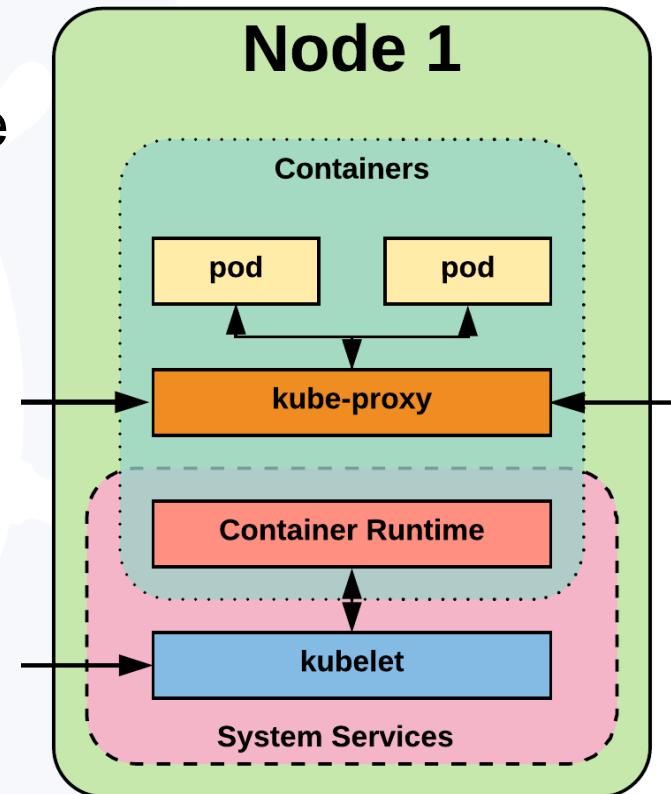
- **kube-apiserver**
 - forward facing REST API into the kubernetes control plane and datastore
 - handles authentication and authorisation, request validation and admission control
- **etcd**
 - provides a strong, consistent and highly available key-value store for persisting cluster state
- **kube-controller-manager**
 - primary daemon that manages all core component control loops
- **kube-scheduler**

assigns pods to nodes. Default scheduler uses bin packing



Node Components

- Kubelet
 - Runs pods
 - Acts as the node agent responsible for managing the lifecycle of every pod on its host
- kube-proxy
 - TCP/UDP forwarding
 - Manages the network rules on each node
- Container Runtime Engine
 - application that executes and manages containers.

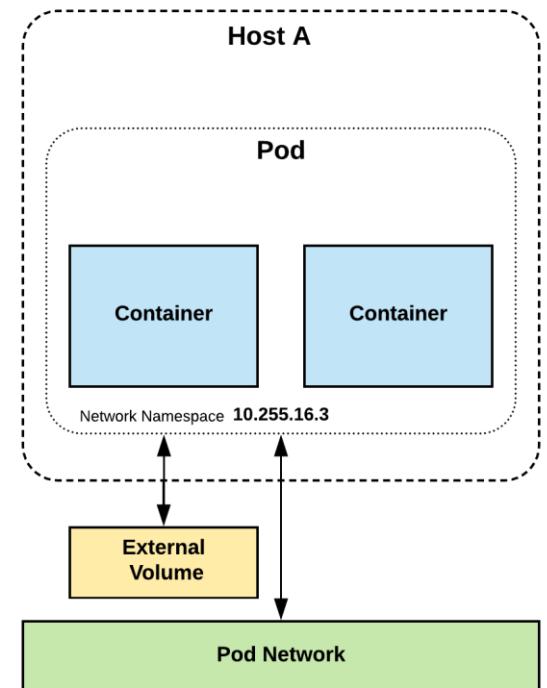


More Concepts

- **Deployment:** Includes a Pod template and a replicas field
 - Kubernetes will make sure the actual state (amount of replicas, Pod template) always matches the desired state
 - When a deployment is updated a “rolling update” is performed
- **Service:** Selects Pods by a matching label selector and provides a stable way to talk to the application by using the **internal IP** or DNS name.
- **Namespace:** A logical isolation method, most resources are namespace-scoped
 - similar workloads can be grouped logically
 - various policies can be enforced.

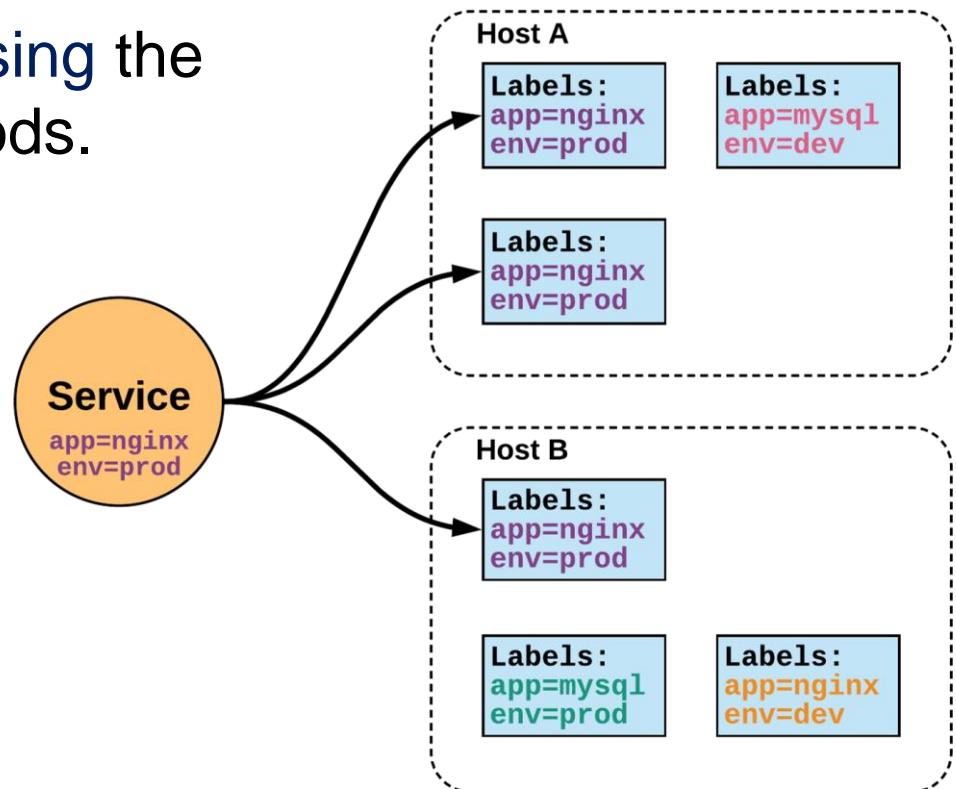
Pods

- Atomic unit or smallest “*unit of work*” of Kubernetes
- Pods are one or MORE containers that share volumes, a network namespace, and are a part of a single context.



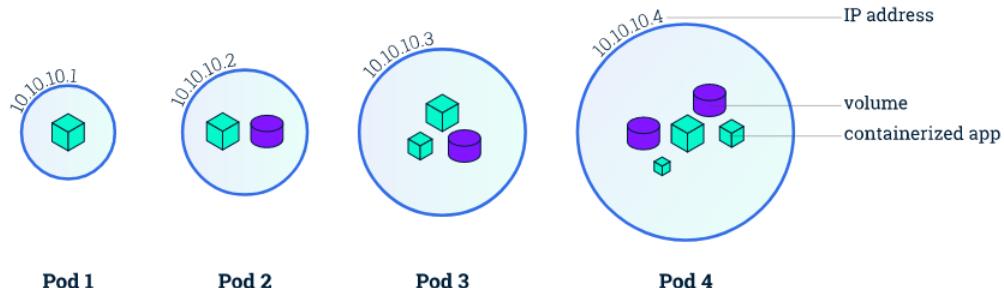
Services

- Unified method of accessing the exposed workloads of Pods.
- Durable resource
 - static cluster IP
 - static namespaced DNS name



Pods and Nodes

- A **Pod** always runs on a **Node**



- A **Node** is a worker machine and may be either a virtual or a physical machine

- Each Node is managed by the Master
 - A Node can have multiple pods

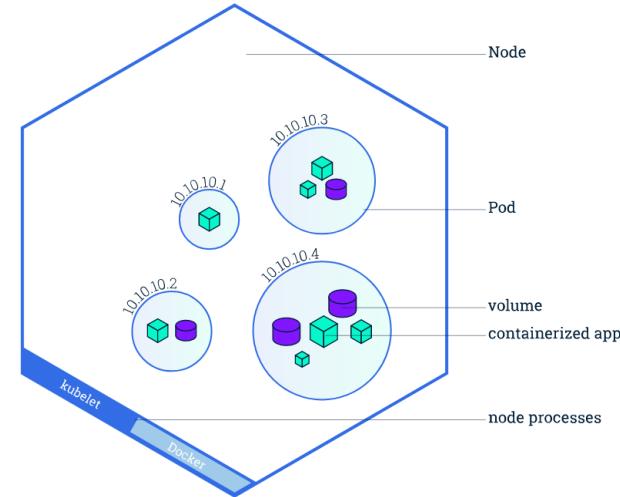
- The **Master** automatically handles scheduling the pods across the nodes in the cluster

- The automatic scheduling takes into account the available resources on each node

- **Kubelet:** process responsible for communication between the Kubernetes Master and the Node

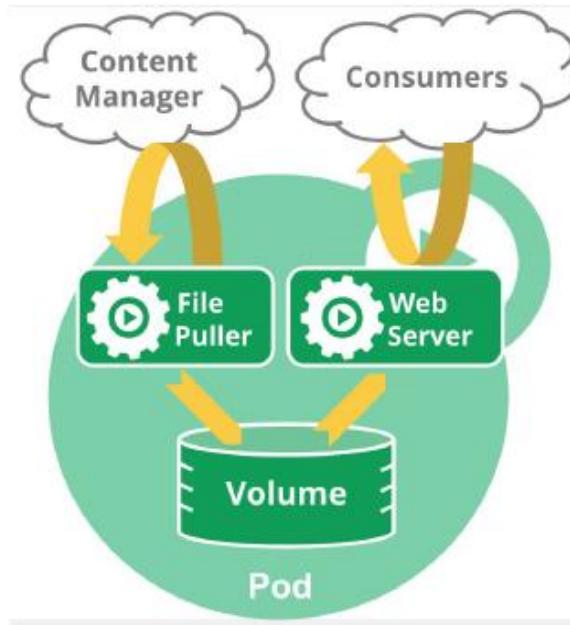
- manages the Pods and the containers running on a machine

- A container runtime e.g. **Docker** is responsible for pulling the container image from a registry, unpacking the container, and running the application.



Pod and Containers (1)

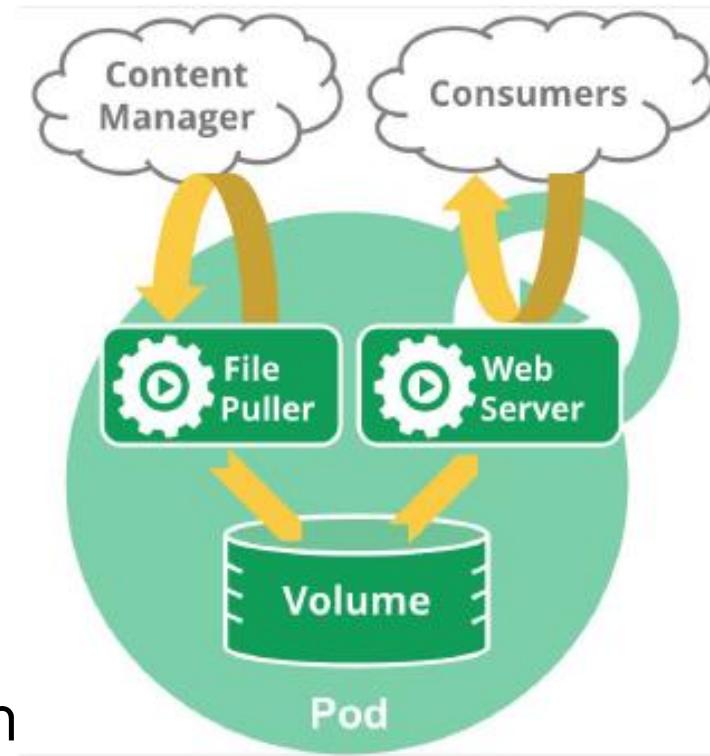
- Containers may be tightly coupled, e.g.
 - a container that acts as a web server for files in a shared volume, and a separate “sidecar” container that updates those files from a remote source



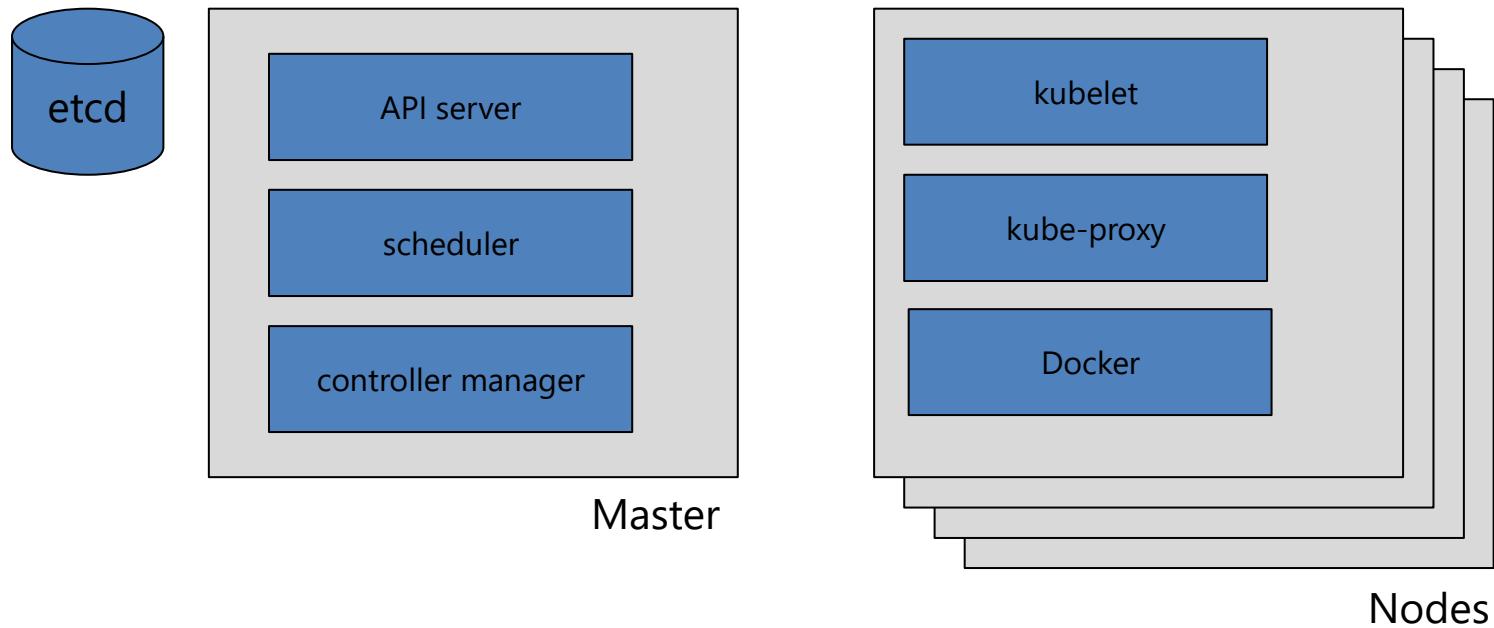
- Pods provide two kinds of shared resources for their constituent containers: **networking** and **storage**.

Pod and Containers (2)

- Every container in a Pod shares the **network namespace**, including the IP address and network ports.
- Containers inside a Pod can communicate with one another using ***localhost***
- A Pod can specify a set of **shared storage volumes**
 - All containers in the Pod can access the shared volumes
- Volumes also allow **persistent** data in a Pod to survive in case one of the containers within needs to be restarted.



Pods: Scaling and Deployments



System Performance

Scale up/down the number of pods based on CPU load or other criteria

System Monitoring

Probes to check the health of each pod

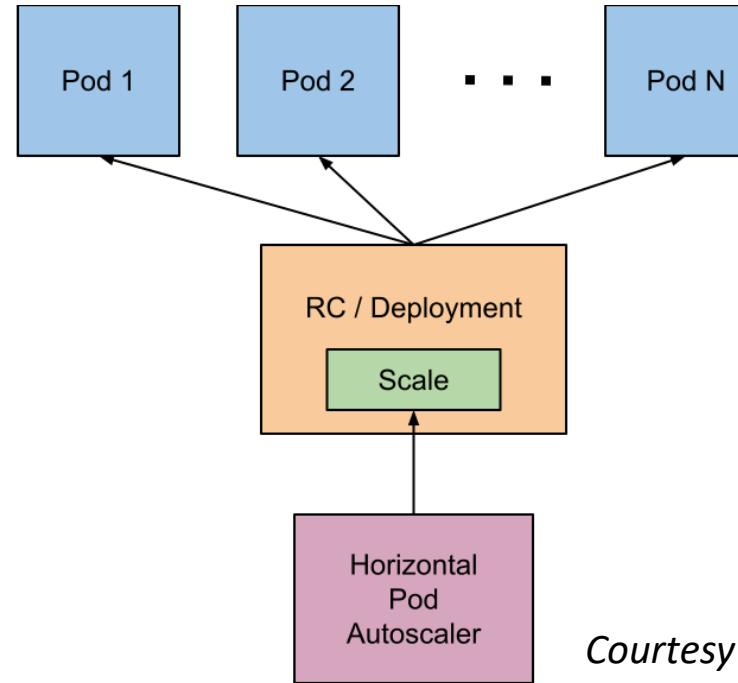
Any unhealthy ones get killed and new pod is put into service

Deployments

Deploy new versions of the container

Control traffic to the new pods to test the new version

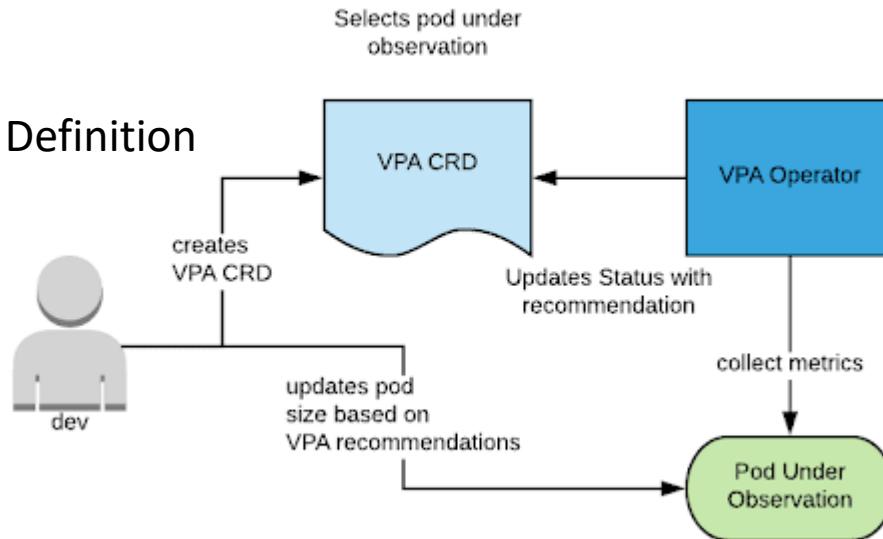
Pods: Horizontal Scaling



- Built-in feature -implemented as a control loop that runs intermittently
- Automatically shrink/increase based on certain parameters
- Gathers information from containers
- Works on top of
 - **ReplicaSets**: makes sure a given number of identical pods are up at any time
 - **Deployments**: manages replica set through time and versions for pod specification

Pods: Vertical Scaling

CRD: Custom Resource Definition



Courtesy of Google

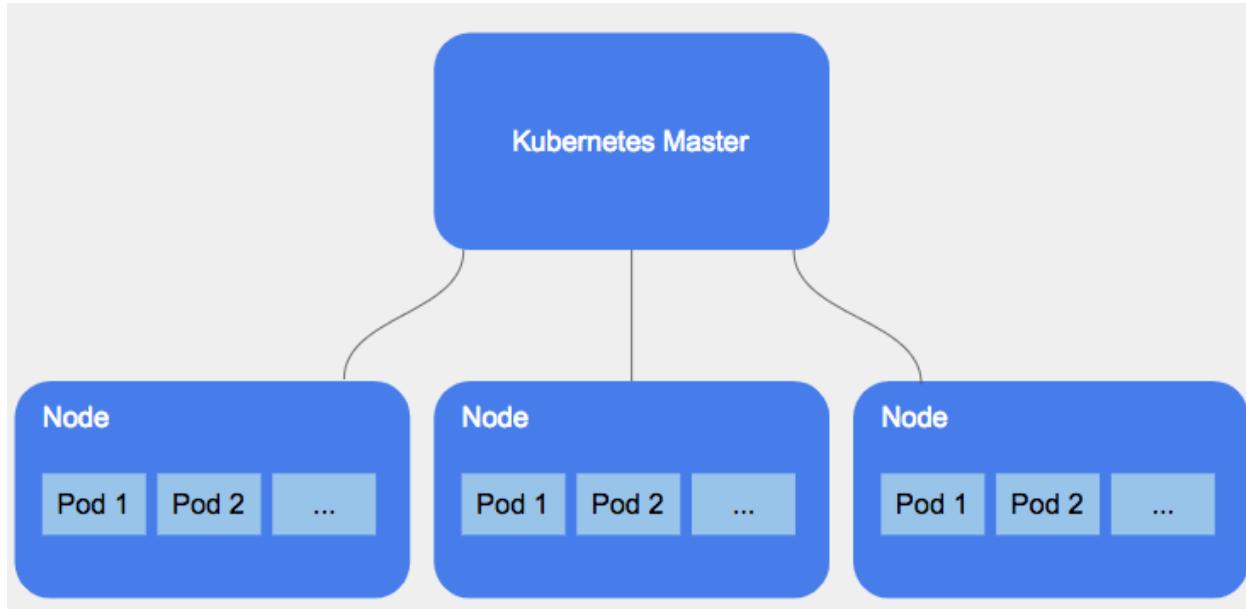
The Vertical Pod Autoscaler (VPA)

- helps size Pods for the optimal **CPU** and **memory** resources required by the Pods
- uses live data to set limits on container resources.
- increases and decreases the requests made by pod containers to ensure actual usage is in line with available **CPU** and **memory** resources.

A VPA deployment

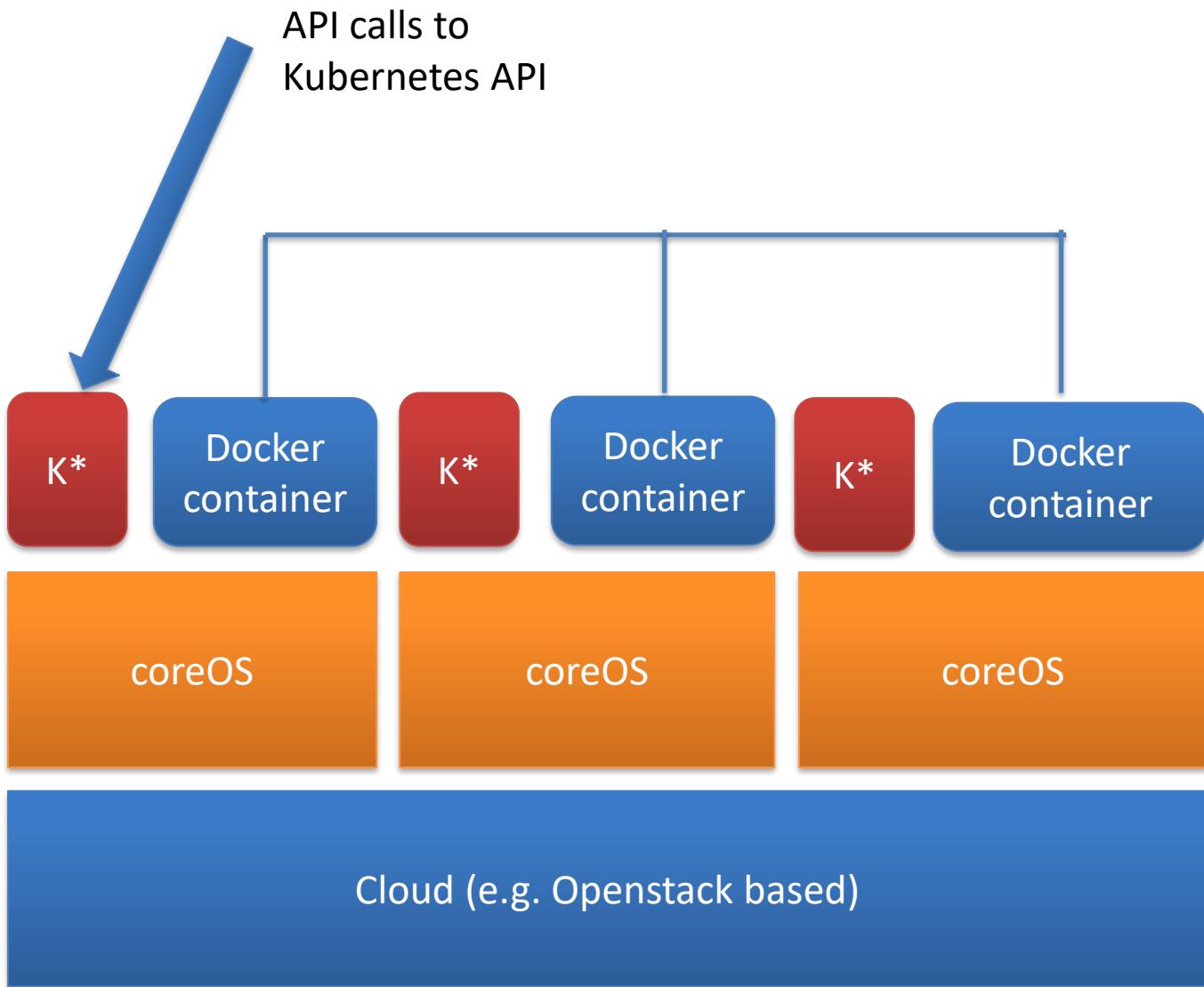
- calculates target values by monitoring resource utilisation, using its **recommender** component
- Its **updater** component evicts pods that must be updated with new resource limits.
- VPA **admission controller** overwrites the pod resource requests when they are created.

Cluster Auto Scaler



The Cluster Autoscaler

- Changes the number of cluster nodes
 - while HPA scales the number of running cluster pods
- Seeks un-schedulable pods
- Tries to consolidate pods that are currently deployed on only a few nodes
- Loops through these two tasks constantly.



Try it out!

See Lab. Session material, week 5

COMP5123M – Cloud Computing Systems



Cloud Programming Landscape

Plan of the Lecture

Goals

- Present the cloud programming landscape

Overview

- PaaS, middleware and configuration
- Cloud Applications
- The Service Life Cycle
- Infrastructure as Code
- Cloud Programming
 - MPI and OpenMP
 - MapReduce and Hadoop
 - TensorFlow
- Other frameworks
 - Amazon Web Services
 - Conclusion

Architectural Layer: Where Are We?

User level



User-Level
Middleware

Cloud applications

Social computing, Enterprise, Scientific, ...

Cloud programming: environments and tools

Web 2.0 Interfaces, Mashups, Concurrent and Distributed Programming, Workflows, Libraries, Scripting

Apps Hosting Platforms

Core
Middleware

QoS Negotiation, Admission Control, Pricing, SLA Management, Monitoring, Execution Management, Metering, Accounting, Billing

Virtual Machine (VM), VM Management and Deployment

Cloud resources



System level

Revisiting the Cloud Service Life-cycle

Construction

- Develop
- Compose
- Configure

Deployment

- Select Provider
- Deploy
- Contextualize

Operation

- Optimize / Schedule
- Execute
- Recontextualize

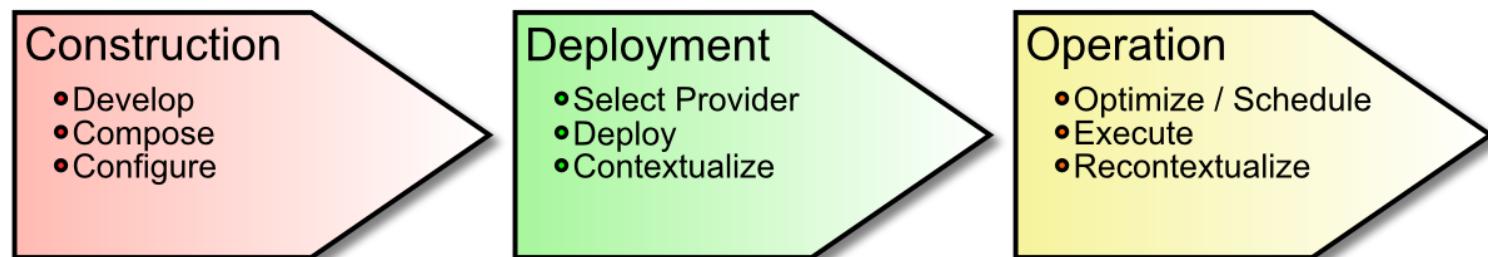
Platform as a Service (PaaS)

- PaaS definition:
 - Any services providing above and beyond basic virtual resource (i.e. virtual machines, block storage) management.
- What is it?
 - A form of Cloud **Middleware**
 - Provides a software solution stack as a service
 - Can aggregate other PaaS and IaaS providers services
 - Often comprised of Tools and/or libraries (APIs)
- What is it used for?
 - Simplifies application **development** via abstraction
 - Facilitates application **deployment** via simplified management



Infrastructure as Code (IaC)

- Modern **IaC languages**: Terraform, TOSCA, Ansible, Chef, Puppet
- Facilitate the provisioning, deployment and configuration process of cloud applications
- Need to build IaC **models** – this requires in-depth knowledge about both the language itself and the characteristics of the operation environment (cloud)
- Ideal solution: generalise different IaC languages and provide a user-friendly language model to manage the development and operation process.



Architectural Layer: Where Are We?

User level



User-Level
Middleware

Cloud applications

Social computing, Enterprise, Scientific, ...

Cloud programming: environments and tools

Apps Hosting Platforms

QoS Negotiation, Admission Control, Pricing, SLA Management, Monitoring, Execution Management, Metering, Accounting, Billing

Virtual Machine (VM), VM Management and Deployment

Core
Middleware



System level

Cloud resources



Cloud Applications

Science and Technical Applications

- Scientific/Tech Applications
- Business Applications
- Consumer/Social Applications

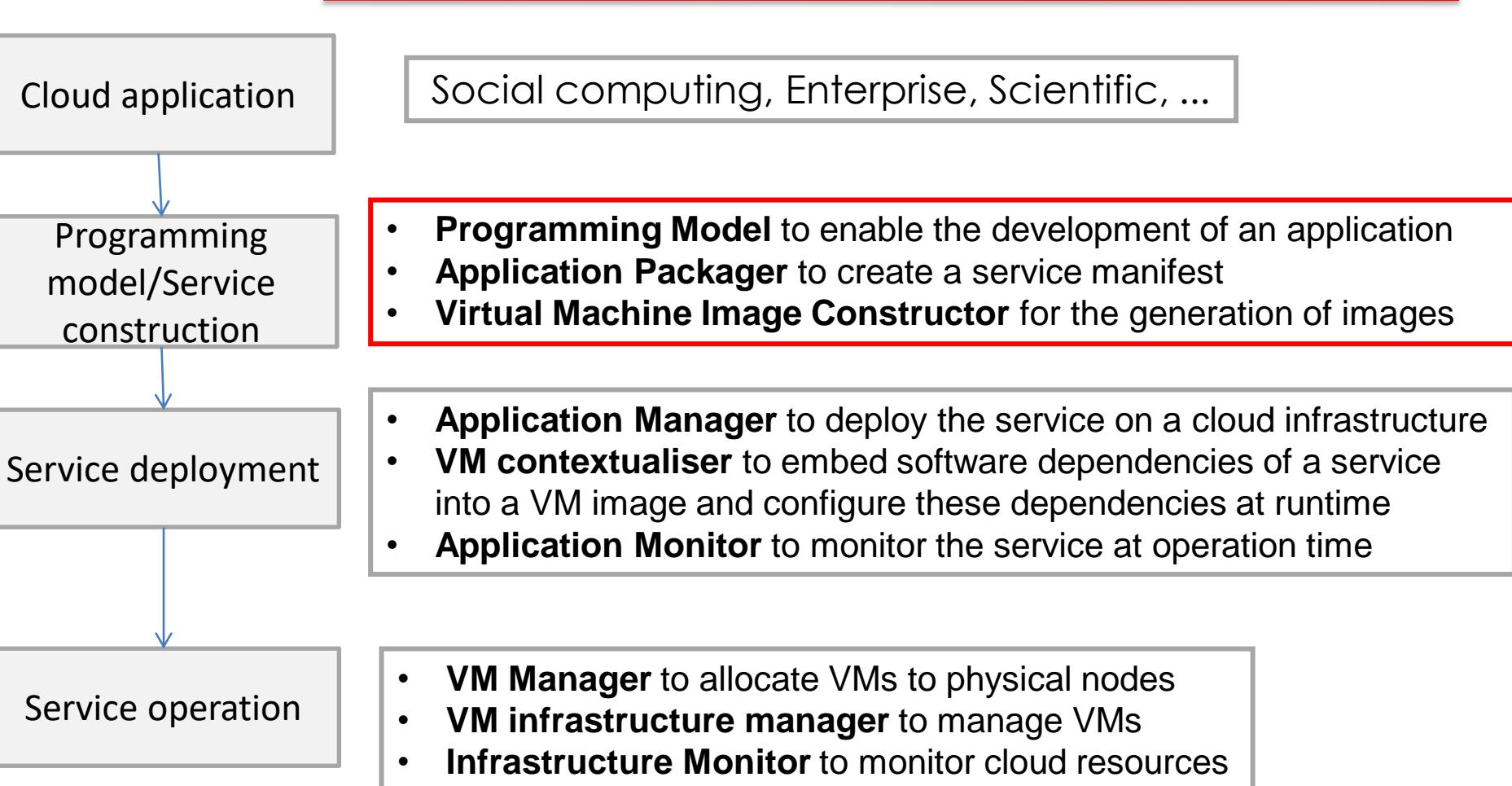


Consumer/Social Applications



Business Applications

Revisiting the Cloud Service Life-cycle



Programming of Applications and Services

- Frameworks that enable the programming of applications and services and their execution in the cloud already exist
- Classified in many **categories**:
 1. High Performance Computing
 - MPI (Message Passing Interface)
 - OpenMP (Open Multi-Processing)
 2. Non-general purpose programming models
 - MapReduce – more on this next week
 - TensorFlow: library to develop and train Machine Learning models
 - Platforms and frameworks for developing distributed applications on the Cloud, e.g. Aneka
 3. New APIs to develop applications
 - E.g. Azure, Google App Engine

Cloud Service Construction (1)

- Cloud services are developed, orchestrated and configured for deployment on cloud infrastructures.

Activities performed

- Develop the service: the applications that deliver the functionality to end-users are developed.
- Use a programming model for simplifying the creation of services, e.g.
 - MapReduce
 - MPI (Message Passing Interface)
- Once the service logic is implemented, Virtual Machine (VM) images are prepared to support the developed applications
- Specify and configure the service requirements describing its functional and non-functional parameters
 - capacity requirements, location constraints, energy efficiency constraints

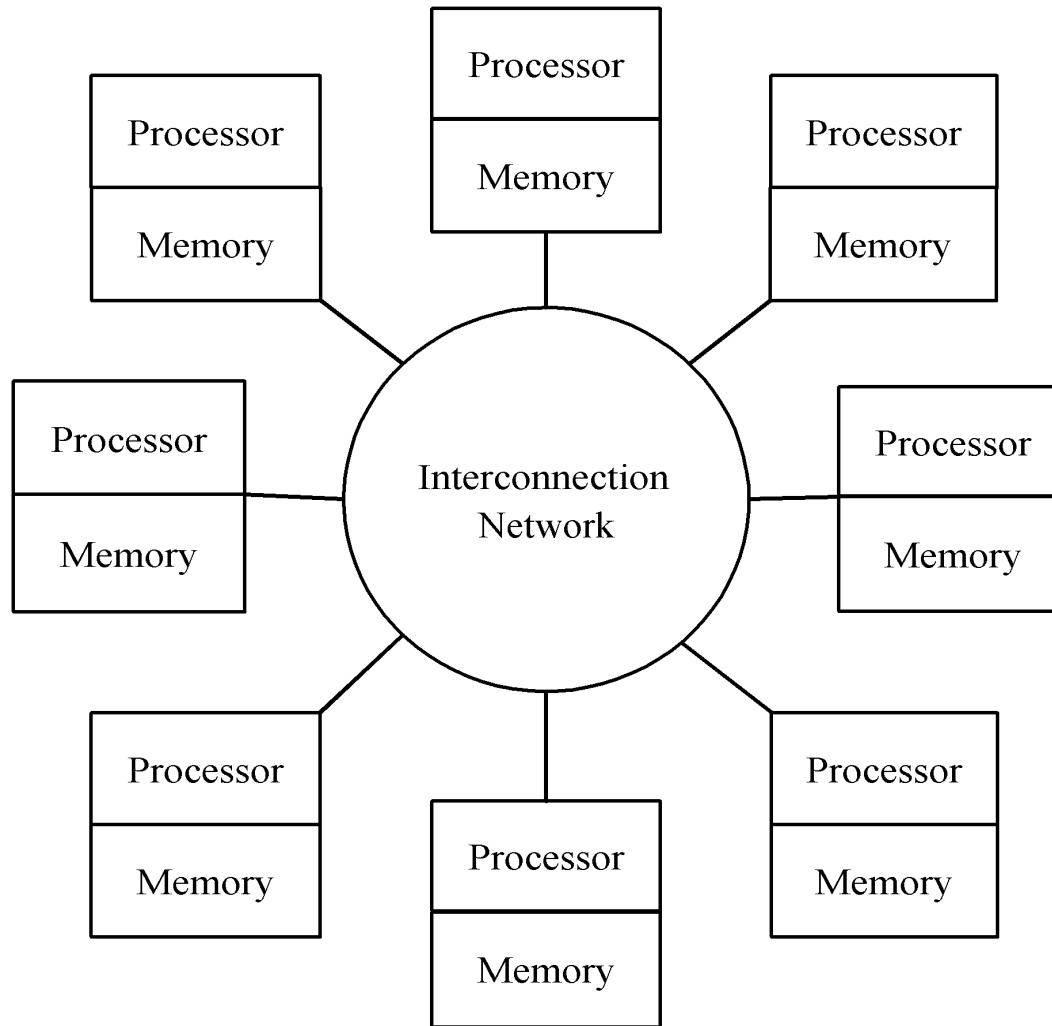
Cloud Service Construction (2)

- Makes abstraction of the **programming language**
 - C, C++, Java, Python, etc
- Any programming language tailored for the cloud?
- The case for **Go**
 - Programming language by Google <https://code.google.com/p/go/>
 - Is the language of choice of cloud projects like Docker and Kubernetes
 - Has ability to provide concurrent operations, as well as other features that exploit the provisioning models of clouds
 - Trying to improve C++

Example 1: High Performance Computing

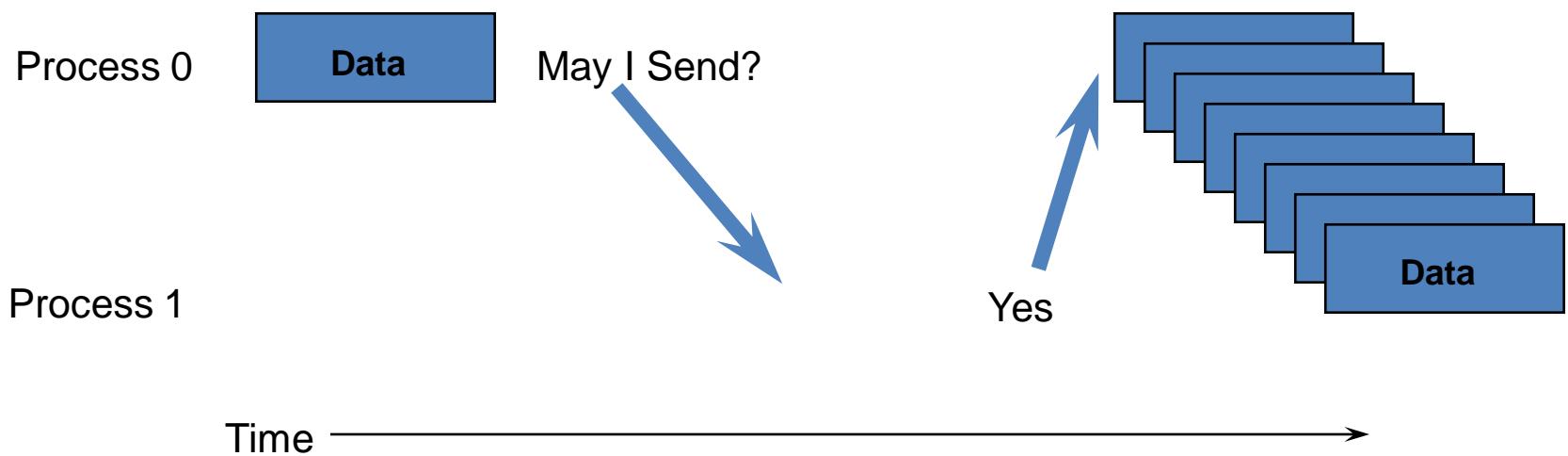
- Advent of high performance multi-computers
- Need for message-oriented primitives to easily write highly efficient applications
- Problem: most interconnection networks and high performance multi-computers were shipped with proprietary communication libraries
 - Need to be hardware independent
- MPI – Message Passing Interface
 - Designed for parallel applications
 - Makes use of underlying network
 - Assumes communication takes place within a known group of processes

Message-passing Model



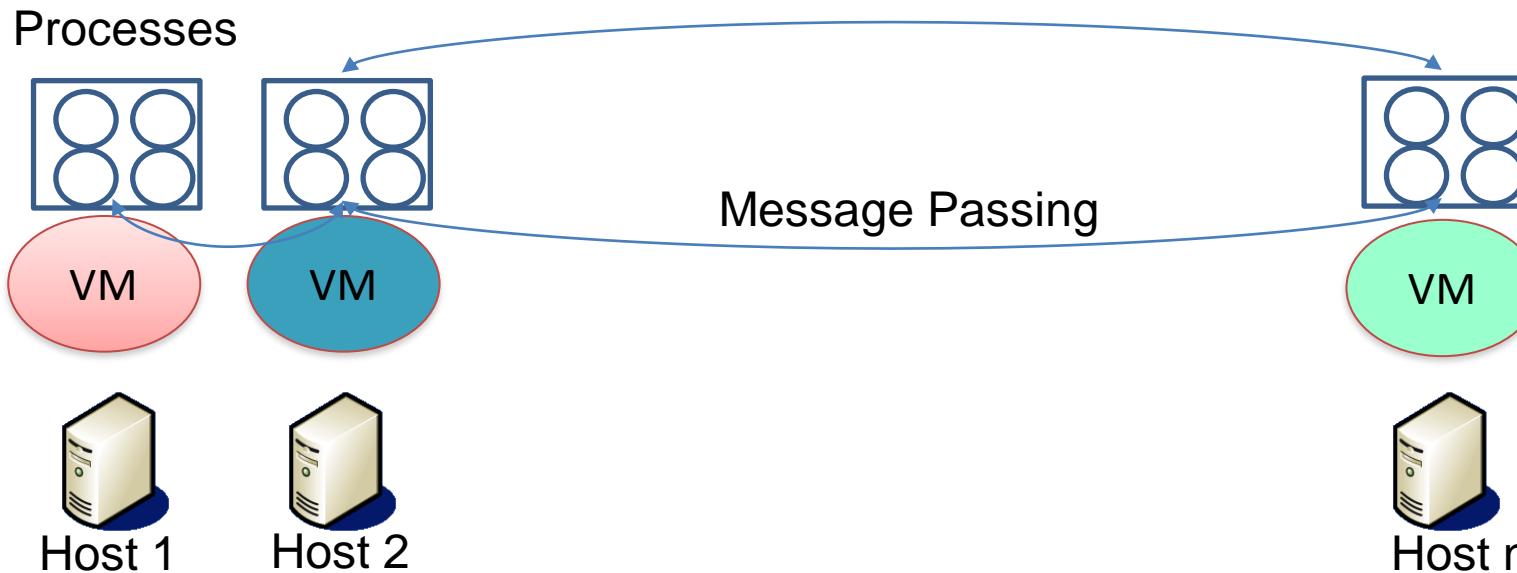
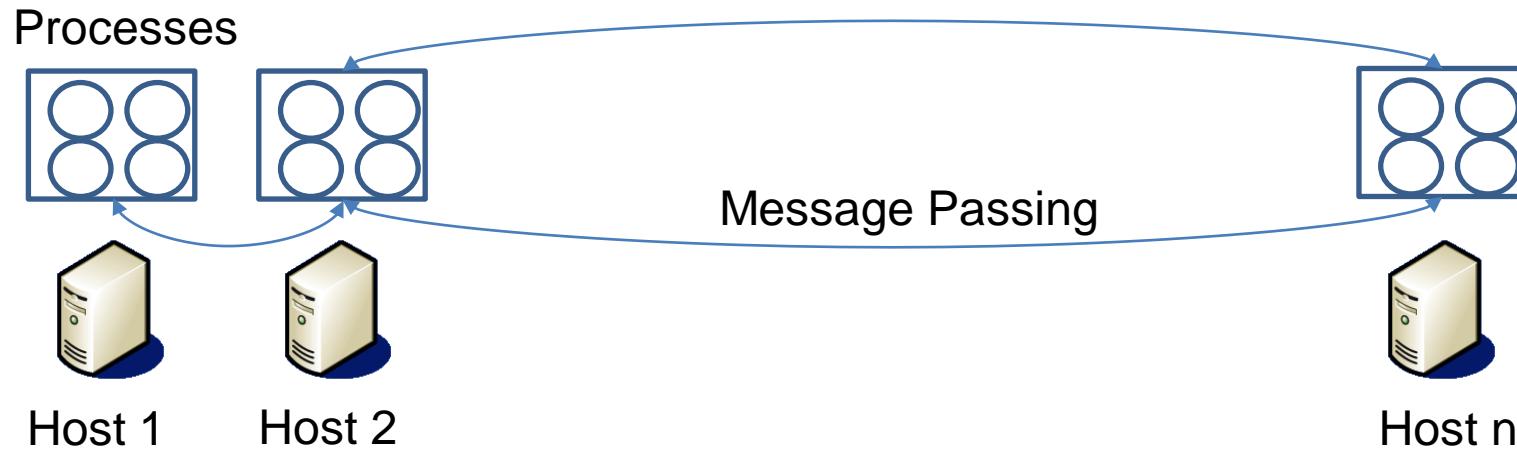
Message-passing Model (2)

- Data transfer plus synchronisation



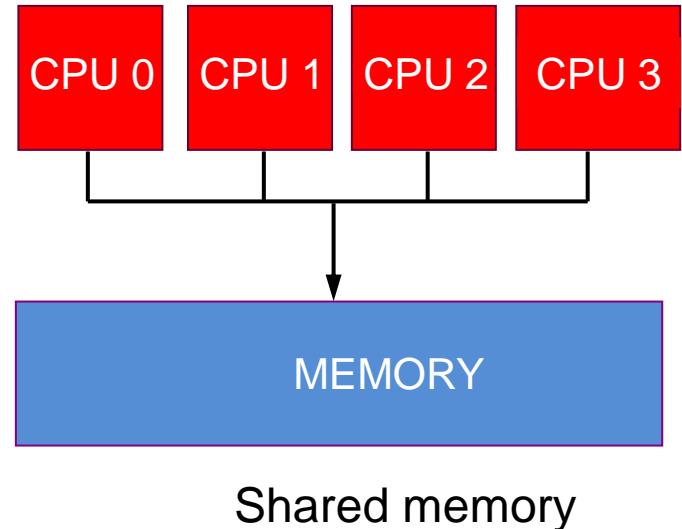
- Large number of primitives, e.g. MPI_send, MPI_recv
- Requires cooperation of sender and receiver
- Cooperation not always apparent in code
- One-to-one vs collective communication

Process Execution: Bare Metal vs Cloud Virtualised Environment



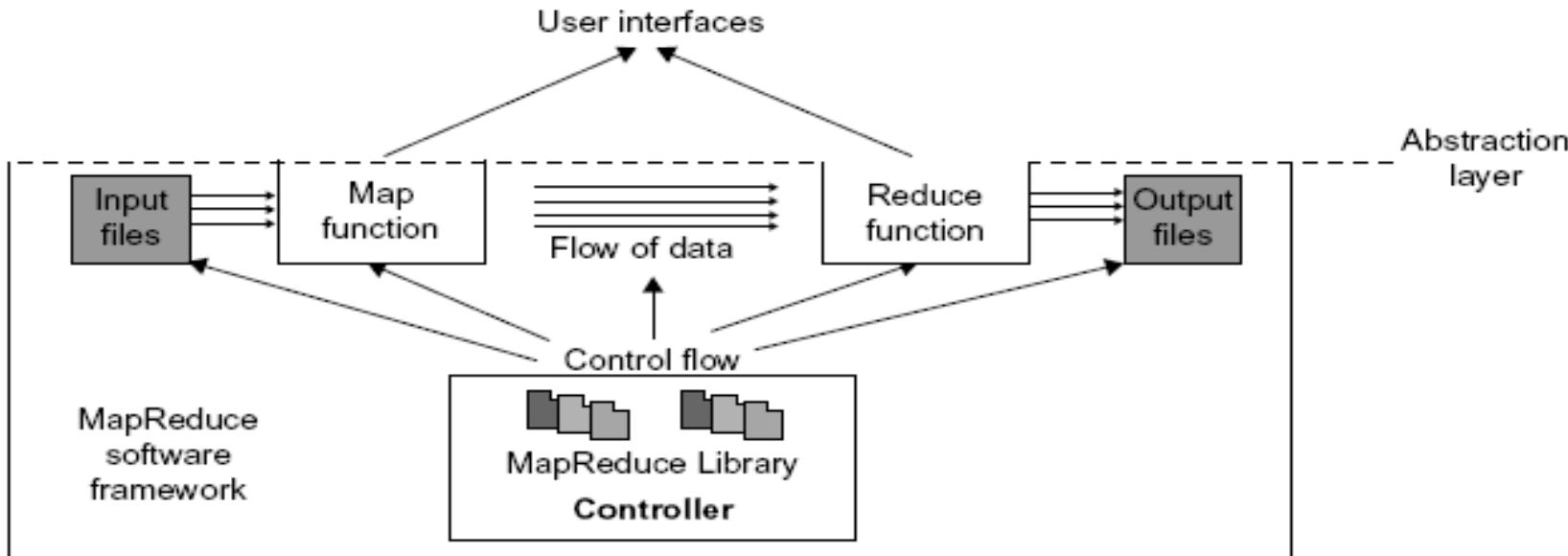
OpenMP

- An alternative to MPI – hence distributed memory
- OpenMP: Application Programming Interface (API) for **multi-threaded** parallelisation consisting of
 - Source code directives
 - Functions
 - Environment variables
- Multiple threads: a process spawns additional tasks (threads) with same memory address space
- Goal: distribute work among threads
 - Loop-level: specified loops are parallelised.



Example 2: MapReduce : Scalable Data Processing on Large Clusters

- A programming model for fast processing large datasets
- Applied in Web-scale search and cloud computing applications
- Users specify a *map function* to generate intermediate key/value pairs
- Users use a *reduce function* to merge all intermediate values with the same key.



Hadoop:

A software platform originally developed by Yahoo to enable users write and run applications over vast distributed data

Attractive Features in Hadoop :

- **Scalable** : can easily scale to store and process petabytes of data in the Web space
- **Economical** : An open-source MapReduce minimises the overheads in task spawning and massive data communication.
- **Efficient**: Processing data with high-degree of parallelism across a large number of commodity nodes
- **Reliable** : Automatically maintains multiple copies of data to facilitate redeployment of computing tasks on failures



- More on this next lecture ...

Example 3: TensorFlow

- TensorFlow is an interface for expressing Machine Learning algorithms, and an implementation for executing such algorithms
- Open source library for numerical computation using **data flow graphs**
- Developed by Google Brain Team to conduct machine learning research
- **Key idea:** express a numeric computation as a **graph**
 - Graph nodes are **operations** with any number of inputs and outputs
 - Graph edges are **tensors** which flow between nodes

TensorFlow Programming Model: *Example*



Neural network with one hidden layer

The **function** to compute: $h = \text{ReLU}(Wx + b)$

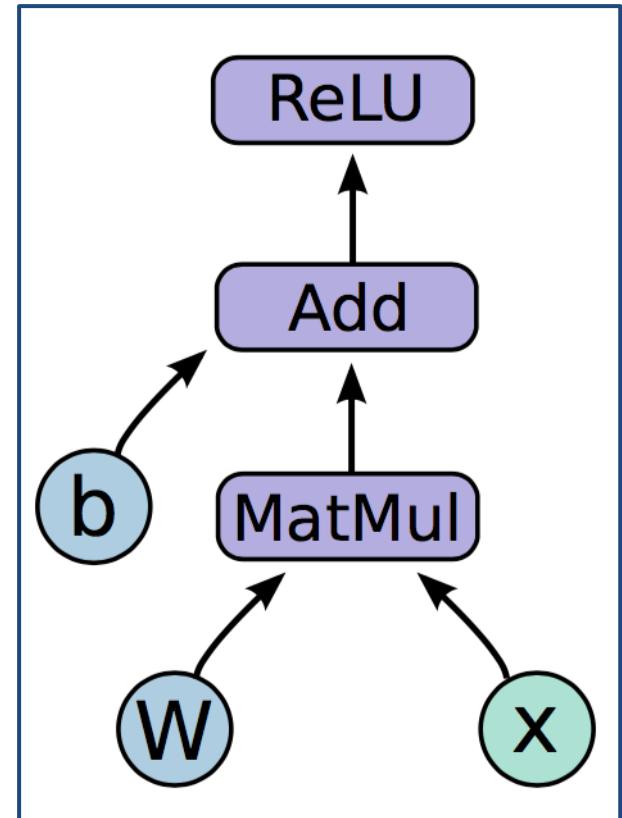
Relu is an activation function that takes the max of 0 and the input, applies a nonlinearity to the network

The graph shows how this function would look like in a tensorflow graph

b and W are Variables: stateful nodes which output their current value. State is retained across multiple executions of a graph (b, W)

X is Placeholder: node whose value is fed in at execution time

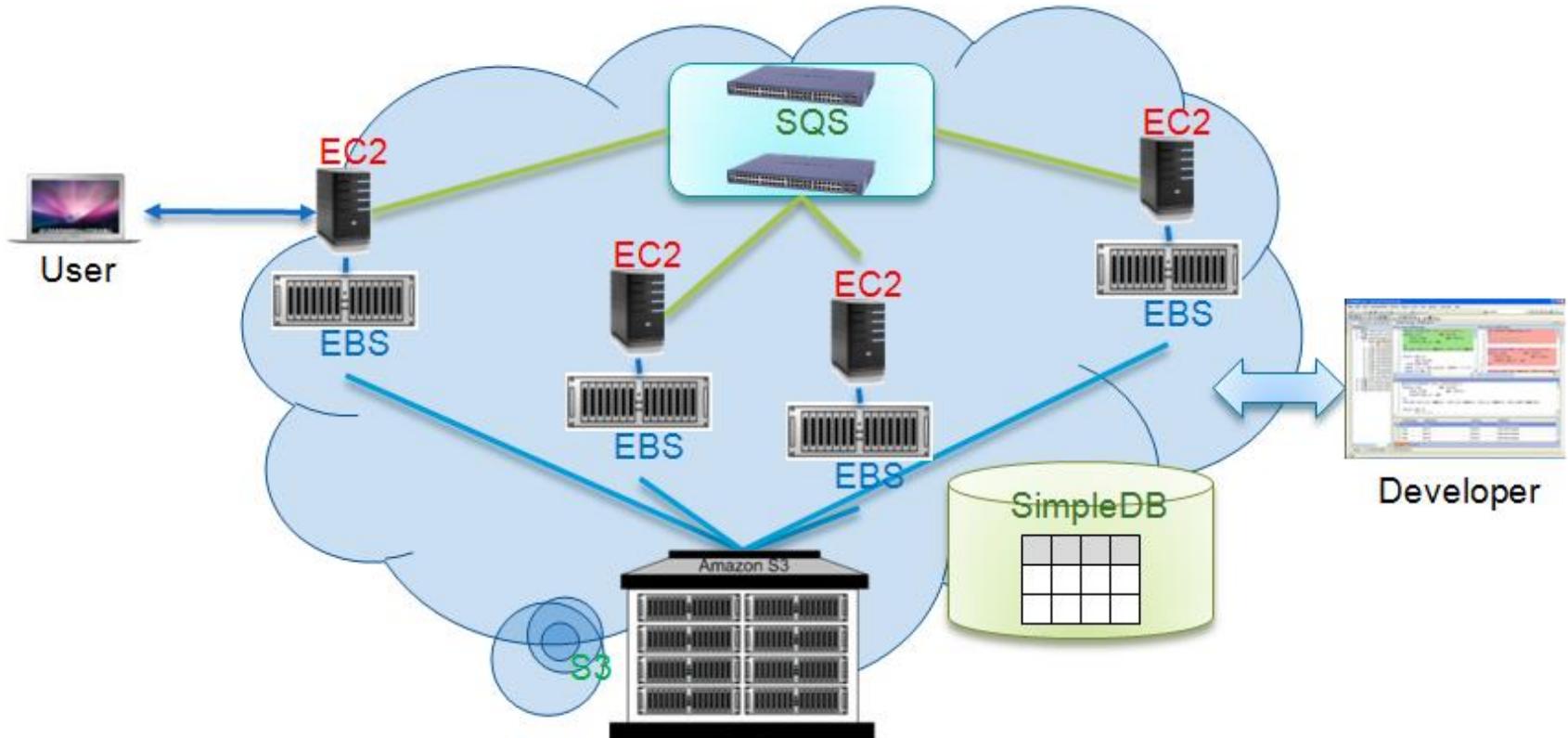
Matmul, addition, and ReLU are operations in the graph that combine the inputs and parameters to produce new nodes.



Example 4: Cloud APIs

The Amazon Web Service (AWS) Platform

Simple Queue Service (SQS): fast, reliable, scalable, fully managed message queuing service.



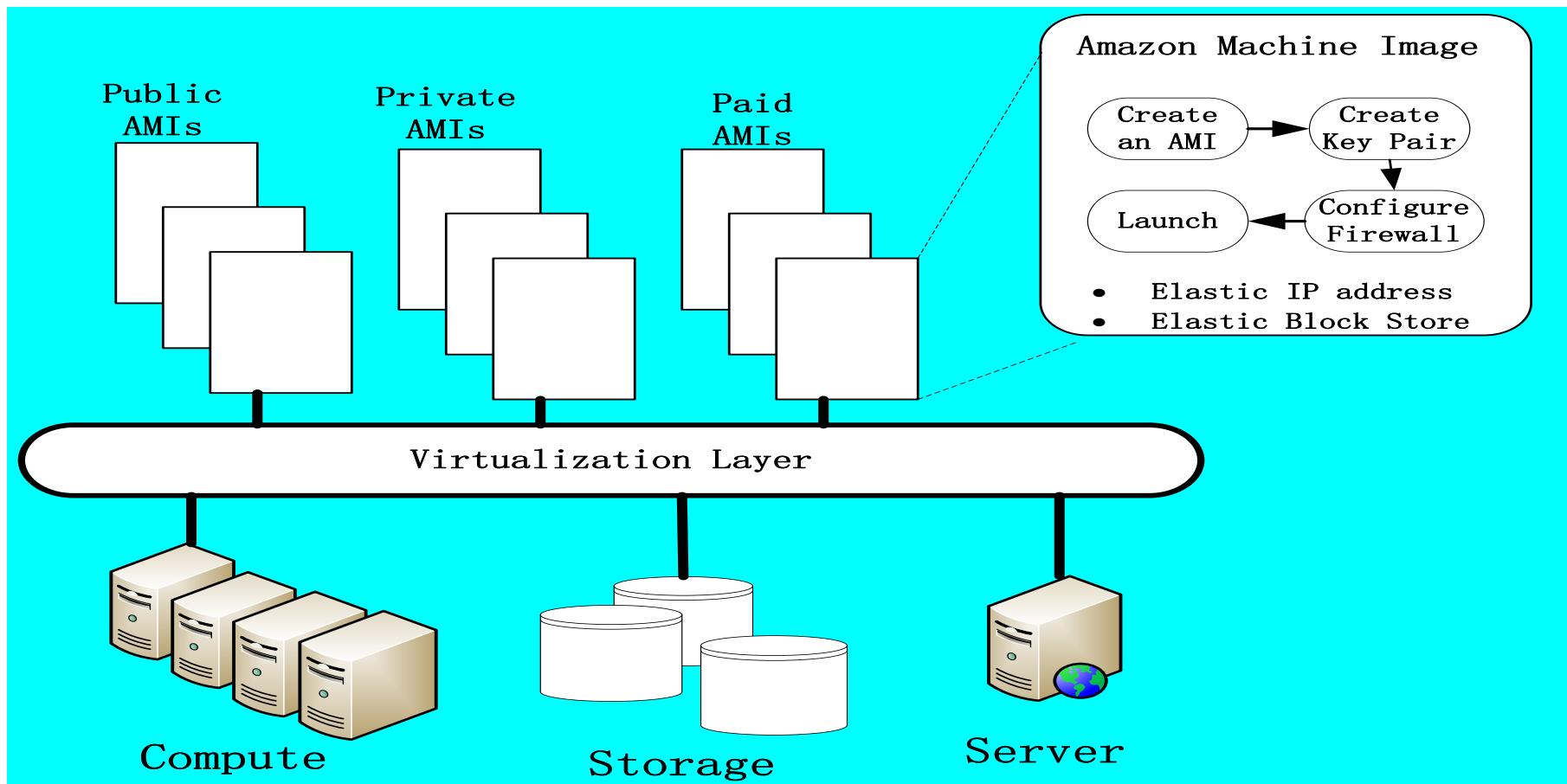
- **AWS Cloud Control APIs:** make it easy for developers to manage the lifecycle of AWS and third-party services
- Provide **five operations** for developers to Create, Read, Update, Delete, and List (CRUDL) their cloud infrastructure.

Parallel Programming on Amazon Web Service (AWS)

(Some) Amazon Platforms and Service Offerings

- AWS Elastic Compute Cloud (**EC2**)
 - is a Web service that provides resizable compute capacity
 - designed to make web-scale cloud computing easy for developers
- AWS Simple Storage Service (**S3**)
 - provides users with secure, durable, highly-scalable object storage
 - simple web services interface to store and retrieve any amount of data from anywhere on the Web
- AWS Elastic Block Store (**EBS**)
 - Provides persistent block level storage volumes for use with Amazon EC2 instances
 - Each Amazon EBS volume is automatically replicated

Amazon EC2 Execution Environment



Amazon Machine Images (AMI)

Image Type	Definition
Private	Images created by you, which are private by default. You can grant access to other users to launch your private images.
Public	Images created by users and released to the Amazon Web Services community, so anyone can launch instances based on them and use them any way they like. The Amazon Web Services Developer Connection Web site lists all public images.
Paid	You can create images providing specific functions that can be launched by anyone willing to pay you per each hour of usage on top of Amazon charges.

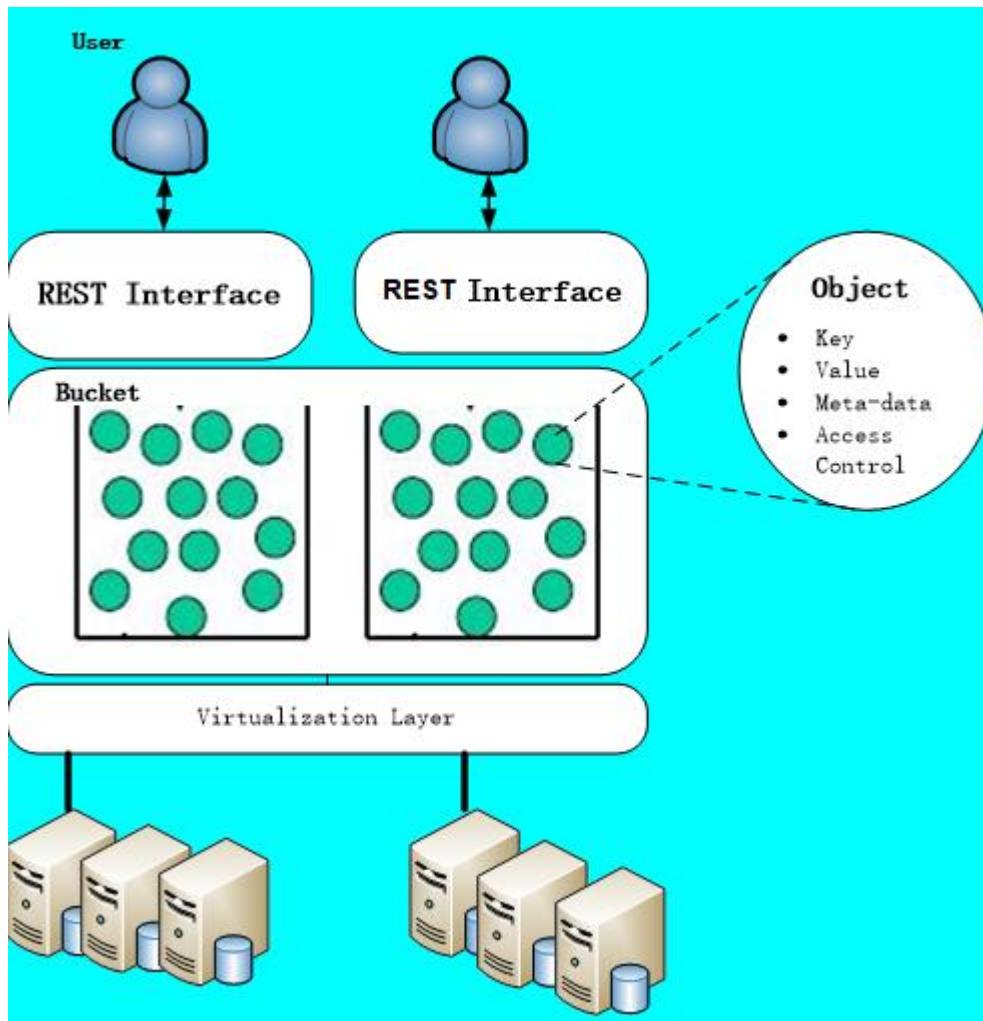
- AMI is a packaged server environment in EC2, based on Linux running any user software or application. AMIs are the templates for VM instances.

S3: Simple Storage Service

Storage Cloud Service: Amazon

- Object-Based Storage
- 1 B – 5 GB / object
- Redundant through geographic dispersion
- 99.99% Availability Goal
- Private or Public
- Per-object URLs
- BitTorrent Support

Amazon S3 for Storage Provisioning



- Object is the basic unit of data
- Bucket for storing objects
- Key for data object retrieval
- Object is attributed to value, metadata, and access control

Conclusion

- Reviewed the cloud programming landscape
- Classified frameworks that enable the programming of applications and services and their execution in the cloud
- Reviewed well known/used programming frameworks and models
 - MPI / OPenMP
 - MapReduce/Hadoop, TensorFlow
 - Amazon Web Services

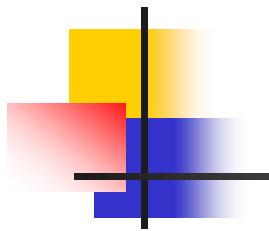
References

- [Hadoop: The Definitive Guide](#), 4th Edition. Tom White. O'Reilly Media, 2015
- Go programming language <https://code.google.com/p/go/>
- Aneka: http://www.manjrasoft.com/aneka_architecture.html
- TensorFlow: <https://www.tensorflow.org/>
- Amazon EC2 - <http://aws.amazon.com/ec2/>
- Amazon S3 - <http://aws.amazon.com/s3/>
- Amazon EBS - <http://aws.amazon.com/ebs/>

COMP5123M – Cloud Computing Systems

Serverless Architectures





Plan of the Lecture

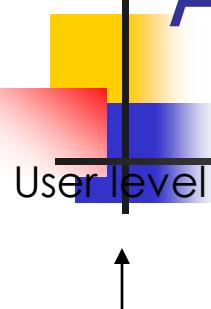
Goals

- Understand the concept of serverless architectures

Overview

- Introduction
- Definition
- Function as a Service
- Architectural Support
- Solutions
 - Commercial
 - Open source
- Summary

Architectural Layer: Where Are We?



User-Level
Middleware

Core
Middleware

System level

Cloud applications

Social computing, Enterprise, Scientific, ...

Cloud programming: environments and tools

Apps Hosting Platforms

QoS Negotiation, Admission Control, Pricing, SLA Management, Monitoring, Execution Management, Metering, Accounting, Billing

Virtual Machine (VM), VM Management and Deployment

Cloud resources



Serverless Computing

- **Serverless computing** simply means that you, the developer, do not have to *deal with* the server.
- A serverless computing platform like **AWS Lambda** allows you to build your code and deploy it without ever needing to configure or manage underlying servers.
- Your unit of deployment is your code; not the container that hosts the code, or the server that runs the code, but simply the code itself.

Serverless Computing

<https://developer.ibm.com/openwhisk/what-is-serverless-computing/>

Serverless computing refers to a model where the existence of servers is simply hidden from developers

- Serverless architecture and **Function-as-a-Service (FaaS)** platforms provide for software developers a whole new world of simplicity, speed and flexibility
- “Serverless” refers to the software architecture, whereas “function-as-a-service” describes the **key mechanism** by which a developer implements the business logic in that architecture.

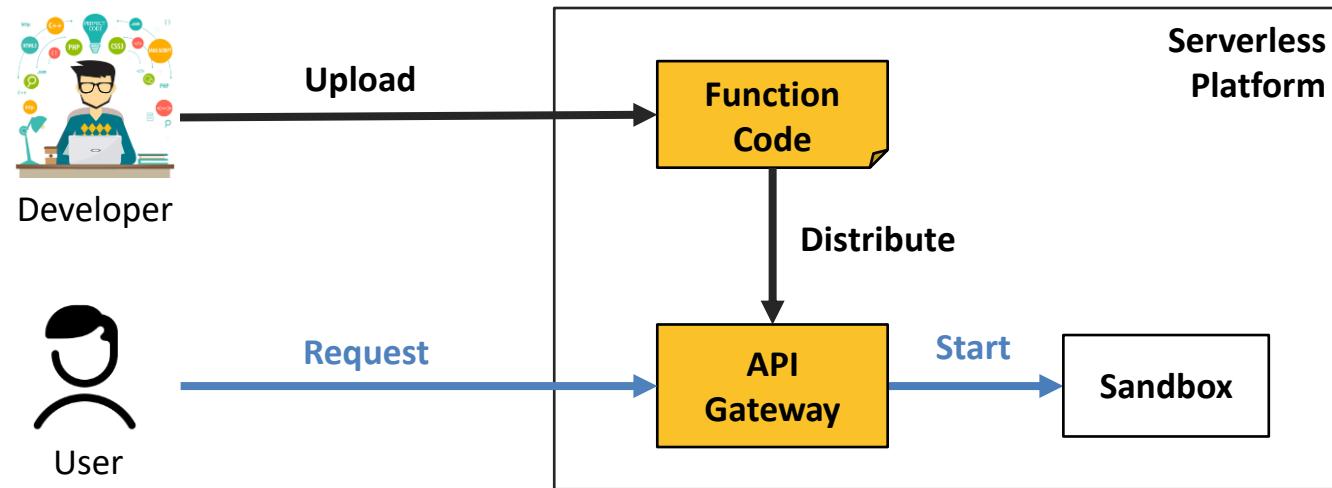
IaaS, PaaS, SaaS and Function as a Service (FaaS)

Private Cloud	Infrastructure (as a service)	Platform (as a service)	Function (as a service) (serverless arch)	Software (as a service)
Functions	Functions	Function	Functions	Functions
Data	Data	Data	Data	Data
Application	Data	Application	Application	Application
Runtime	Runtime	Runtime	Runtime	Runtime
Backend Code	Backend Code	Backend Code	Backend Code	Backend Code
OS	OS	OS	OS	OS
Virtualization	Virtualization	Virtualization	Virtualization	Virtualization
Server Machines	Server Machines	Server Machines	Server Machines	Server Machines
Storage	Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking	Networking

 Public Cloud Provider - responsibility

 Application Writer - responsibility

FaaS: Core of Serverless Computing



AWS Lambda



Microsoft Azure

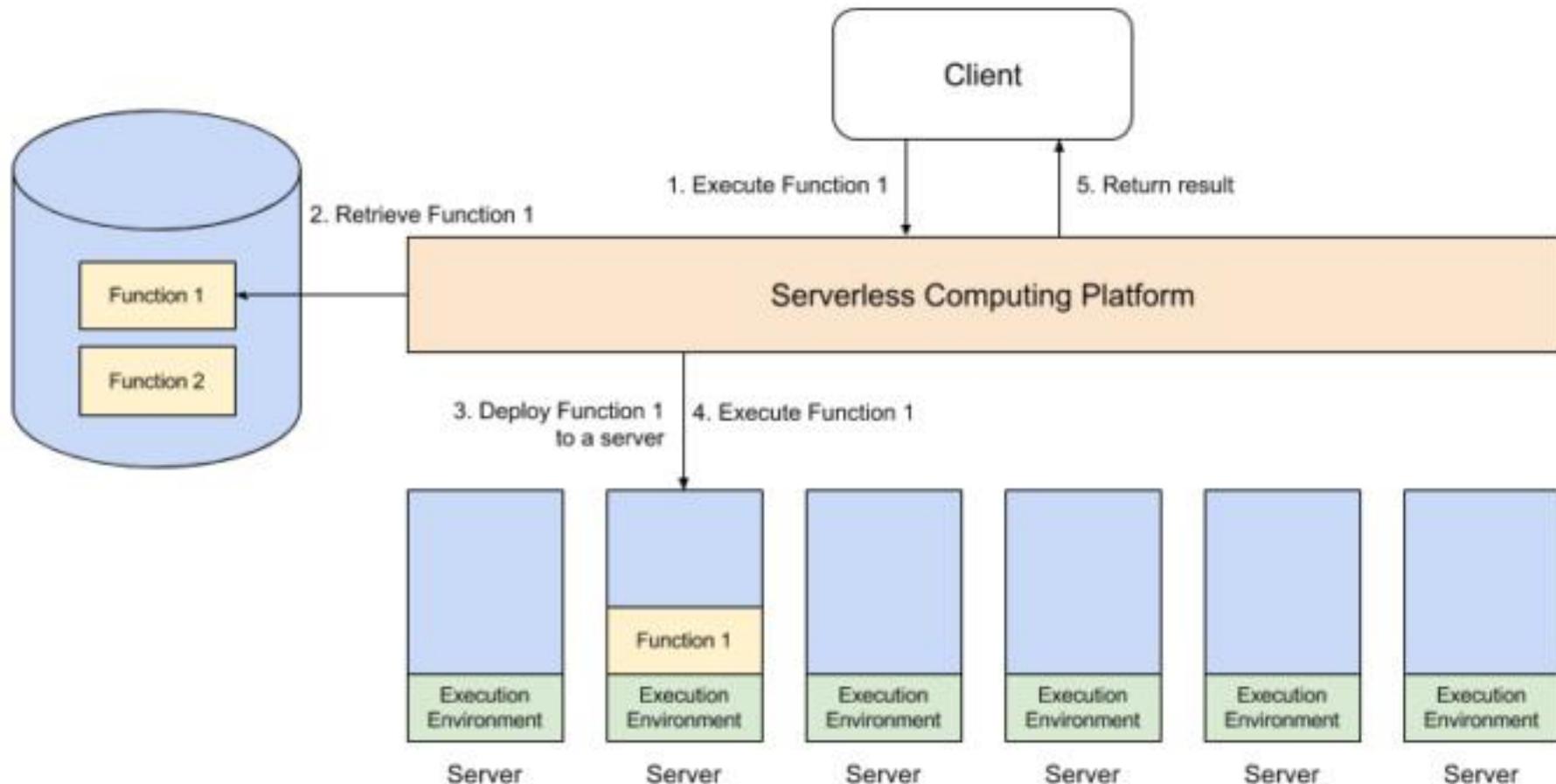


Google Functions



IBM OpenWhisk

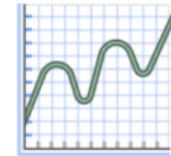
Serverless Computing Execution Model



The serverless computing platform first checks to see if the function is running on any of its servers. If the function isn't already running, then the platform loads the function from a data store.

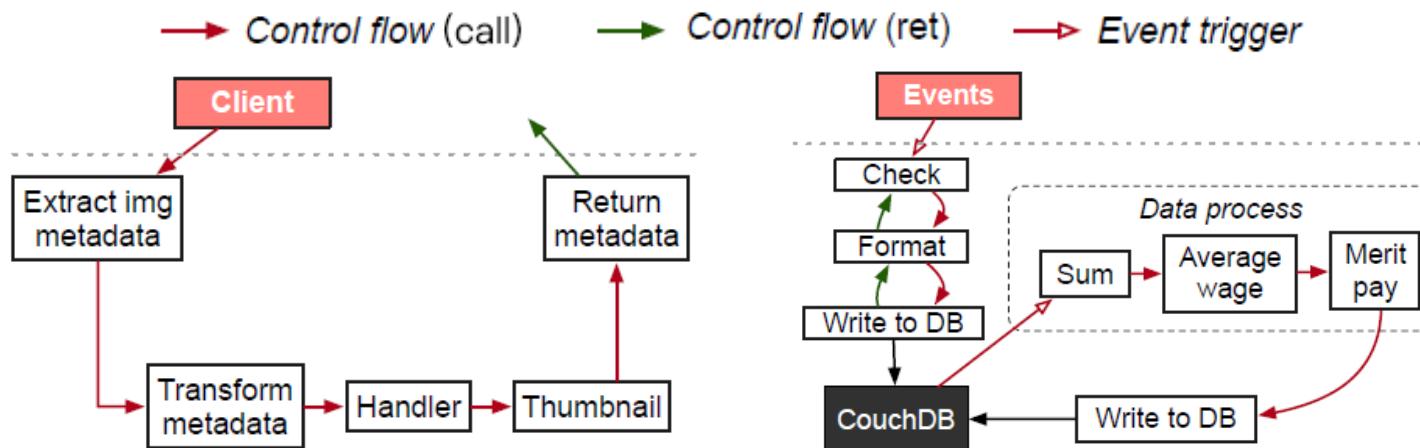
Serverless Computing - Benefits

- No Servers to Manage
- Continuous Scaling
- Dynamic allocation of resources
- Avoid overallocation of resources
- Never Pay for Idle: pay-per-usage



Serverless Applications

- **Single-function applications:** Alexa skills, image resizing, etc.
- **Multiple-function applications:** IoT, Image recognition, database analysis, etc.
- **Emerging applications:** Parallel processing, machine learning



Serverless Functions

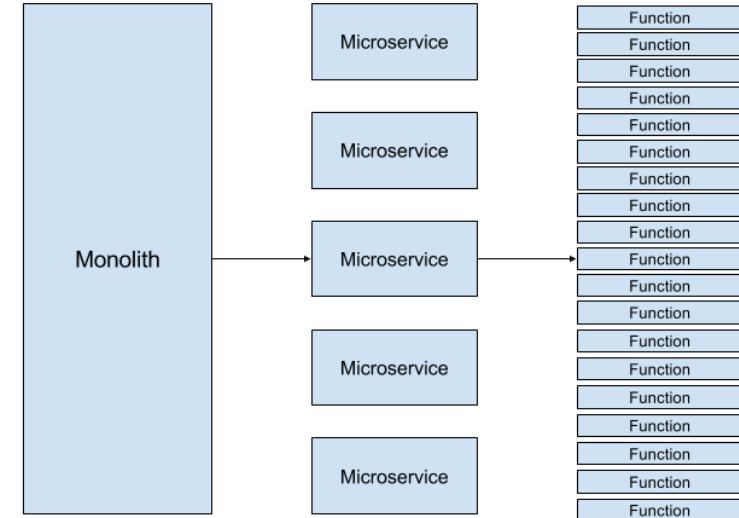
Go from monoliths to microservices to functions (nanoservices)?

Microservices

- Smaller-grained services
- Specified Functions
- Defined Capabilities

Microservices Architecture

- Event handler
- Serverless back ends
- Data processing



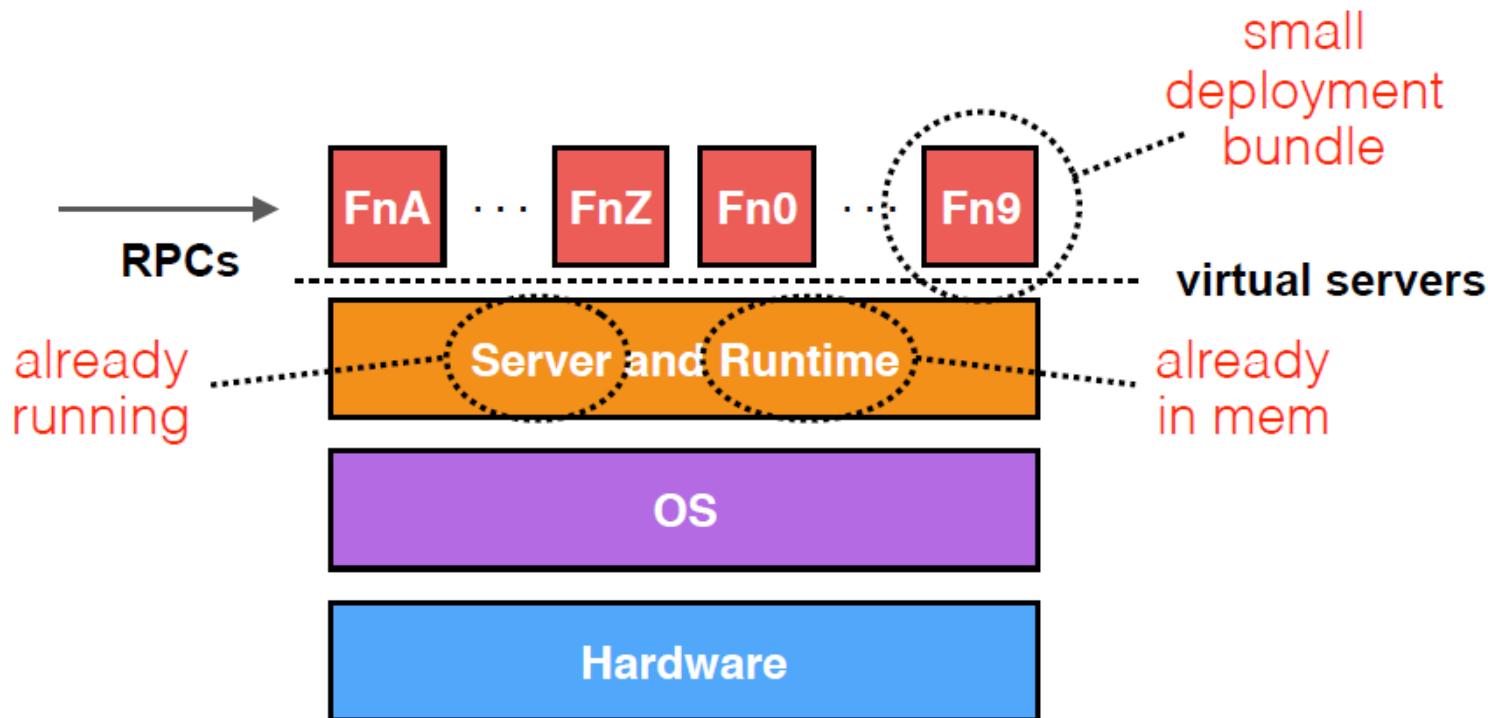
Serverless functions

- Services that are event-driven and instantaneously scalable
- flexible (de)composition and interoperability of software and data at run-time

Performance Issue:

- Cold start

Cold Start vs Warm Start



advantages:

- very fast startup
- agile deployment
- share memory

Issue:

function is running on server -> **warm start** of the container
Function not running _> **cold start**,
container needs to be instantiated
(overhead)

Serverless Architecture Solutions

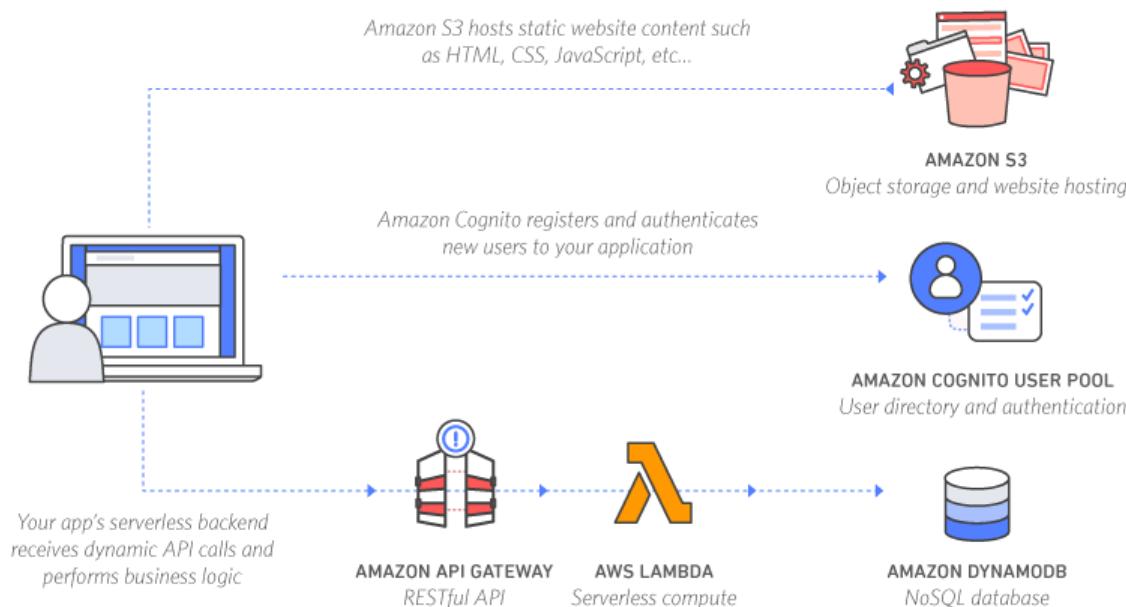
Commercial and Open Source

- Amazon Lambda
- IBM Openwhisk
- Google cloud functions
- Microsoft Azure functions
- Apache Openwhisk
- OpenFaaS
- Iron.io, Fission
- Other

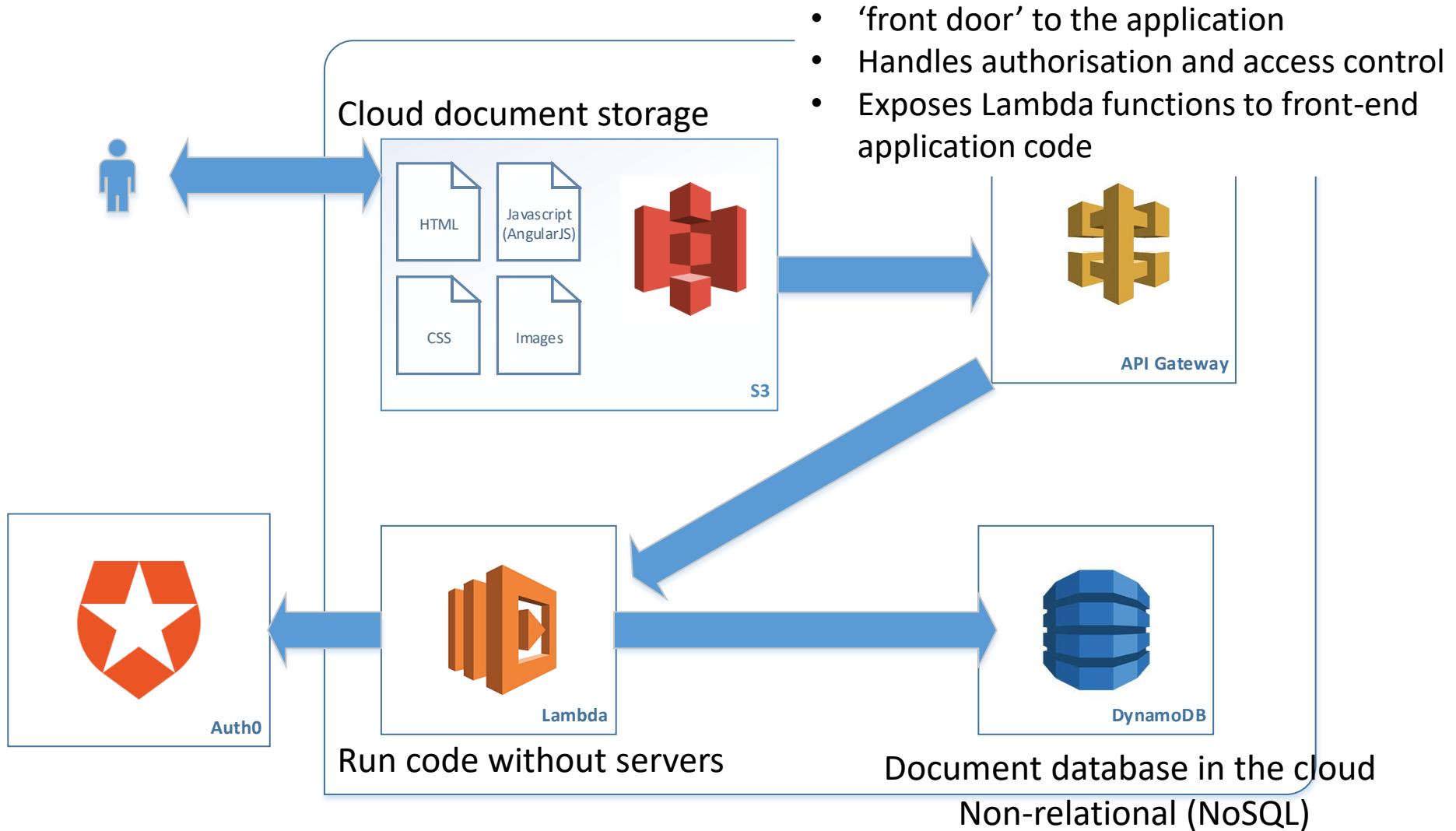
AWS Lambda

<https://aws.amazon.com/lambda/>

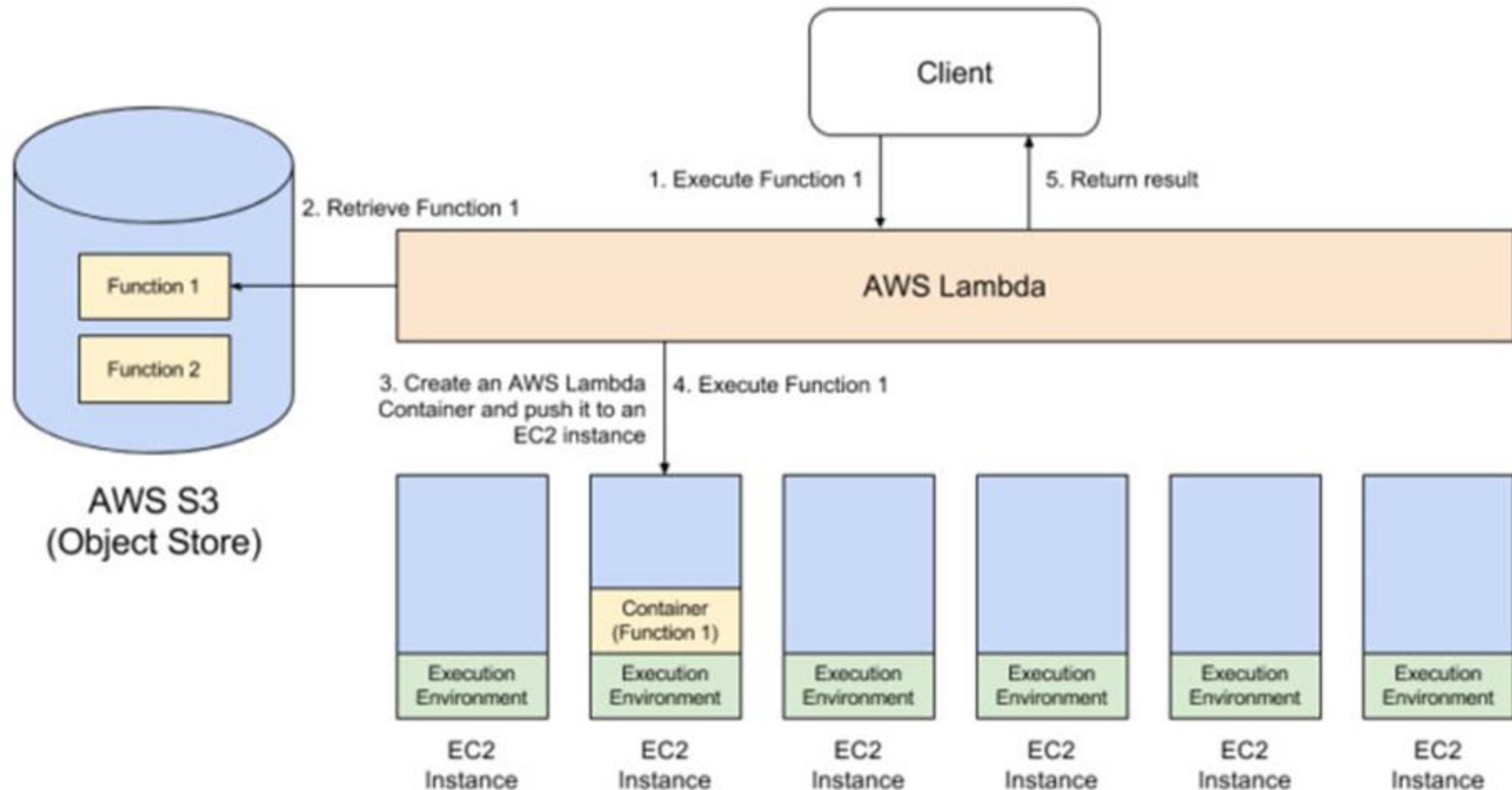
- Allows software developer to execute code without administrating services
 - No necessity to admin any server
 - Automatic scaling
- How it works
 - Event (trigger) based
 - Events can be from different sources: AWS Services, http-endpoints, or internal to the app
 - Code runs in container
 - For each lambda function a container is created
 - User specifies the amount of RAM, max execution time via Programming Model.



Architecture



AWS Lambda – In Practice



AWS Lambda

<https://aws.amazon.com/lambda/>

- Container execution
 - When a container is created the first time, it has some initialization penalties (**cold start**), after that uptime is quicker (**warm start**)
 - Concurrent executions
 - **Stream based** event sources, e.g. Amazon Kinesis Streams or Dynamo DB Streams
 - As many containers as needed will be created
 - **No-Stream based** event sources
 - A container per event will be created
 - Containers are re-used if the previous event has finished
- Detailed monitoring
- Pay per processing unit

Microsoft Azure Functions

<https://docs.microsoft.com/azure/azure-functions/functions-overview>

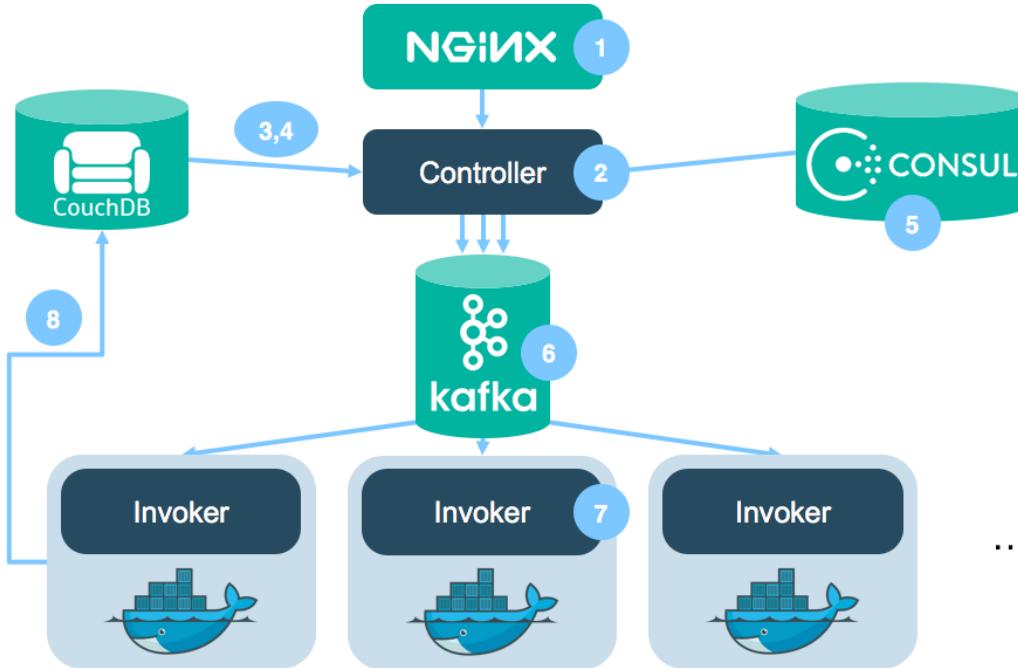
- Code implemented in several languages (not only .NET based ones...)
- Triggers code execution on Azure based on events on Azure, Third-party services or on-premises
 - Time triggers (cron), Azure Storage events, Azure Queues, messages from Service Bus, and HTTP triggers
- Based on Webjobs SDK
- Open Source
- Code can be in several sources, including Github or Bitbucket, so allows continuously deployment

Apache OpenWhisk

- <https://openwhisk.apache.org/>
- OpenWhisk follows a simple event-driven architecture
- Open source cloud platform that allows functions being triggered in response to events originating from direct invocations to the OpenWhisk API
- Handles the infrastructure and servers
- Sample applications

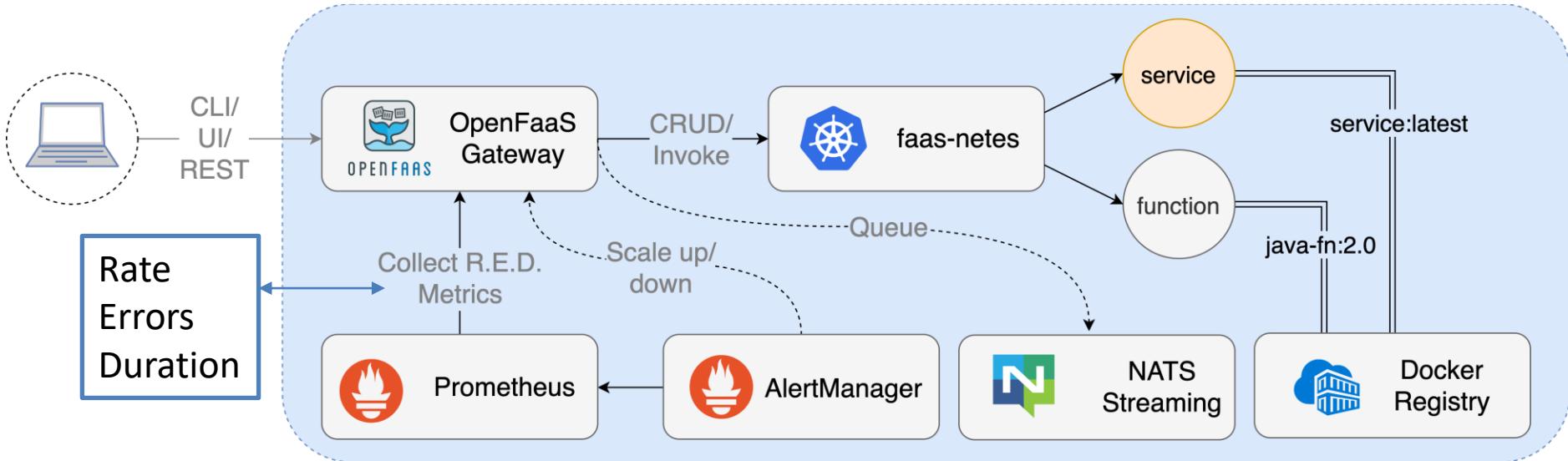
<https://github.com/apache/incubator-openwhisk-external-resources#applications>

Apache OpenWhisk - Architecture



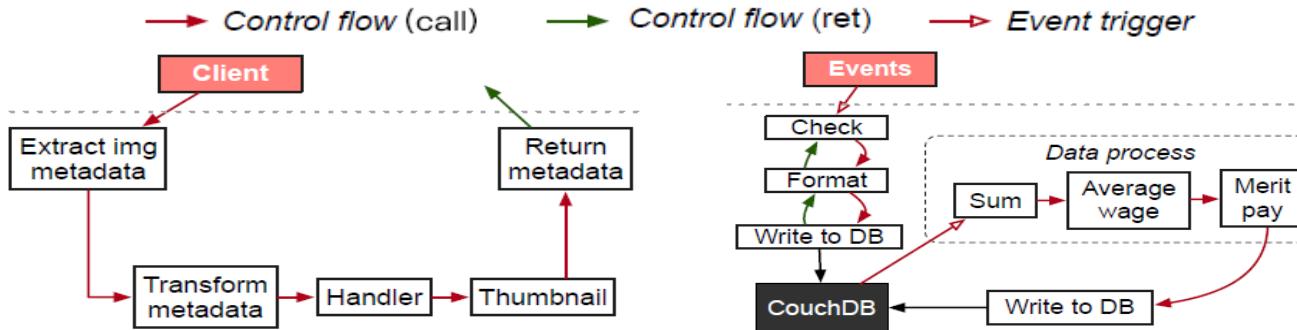
1. **Nginx**: exposes HTTP endpoints to clients so functions can be requested
2. Request hits the controller which performs authorisation and authentication of every request
3. and 4. The controller interacts with a **CouchDB** instance to verify credentials, namespaces associated with the requested function
5. The controller interacts with a **Consul** instance for service discovery
6. Real-time data pipeline tool **Kafka** is used for communication between the controller and invokers
7. and 8. Invokers perform all the heavy lifting, handling container deployment and resource allocation to create a runtime for function execution.

OpenFaaS - Architecture



- <https://www.openfaas.com/>
- OpenFaaS - management and auto-scaling of compute – PaaS / FaaS, a developer-friendly abstraction on top of Kubernetes
 - Each function or microservice is built as an immutable Docker container
 - NATS - asynchronous message bus / queue for deferred execution
 - Kubernetes - declarative, extensible, scale-out, self-healing clustering
 - Prometheus - a time-series database used to capture metrics
 - Linux - The base Operating System to provide containers.

Serverless Workflows



- **Commercial** workflow products provided by cloud vendors, e.g. AWS Step Functions, Azure Durable Functions, Google Cloud Composer
- **Open-source** projects include Apache Airflow, HyperFlow and Cloud Native Computing Foundation (CNCF) Serverless Workflow

Features:

- **Workflow definition**, e.g. using
 - a Domain-Specific Language (DSL) that is written in YAML or JSON
 - SDKs in a programming language such as Python and JavaScript.
- **Operators**: Tasks that a workflow orchestrates, such as local functions, cloud functions and other vendor-specific services
- **Error handling strategy**: when a task fails in a workflow
- **Parallelism support**: Whether the workflow system supports executing tasks in parallel mode
- **Invocation method**: dashboard, executing commands on the terminal or triggered by events.
- **Autoscaling strategy**: automatically scales the workflow systems up when receiving more invocations and scales down when the workflow is idle.

Research Categories and Investigation Context



Function Execution

Platform deployment environment

Testing & observability

Benchmarking

Costs optimisation

Programming models

Research-centric platforms

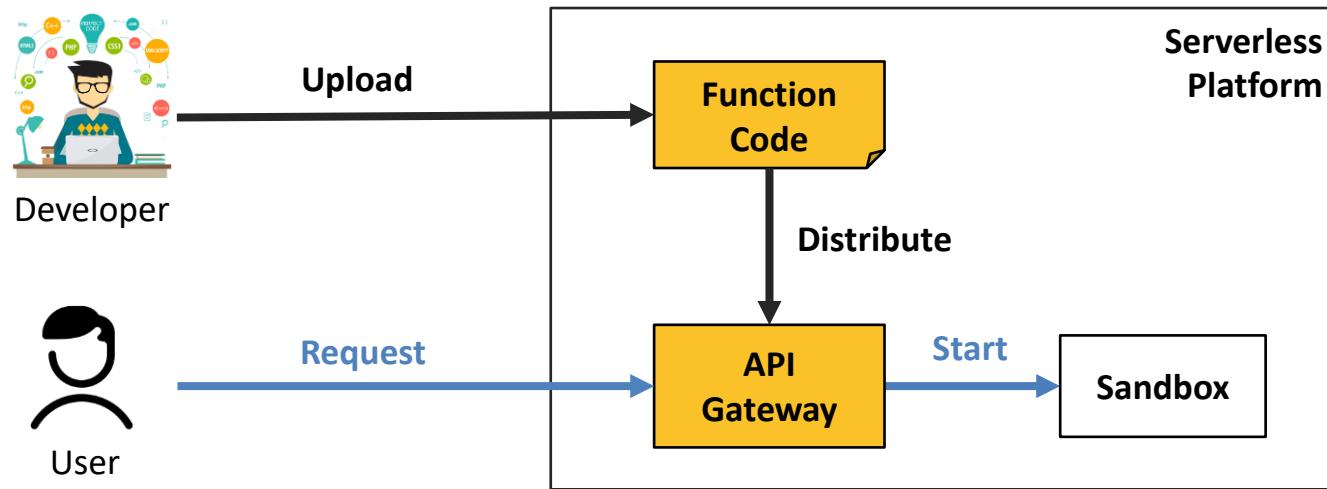
Deployment automation

Migration

Continuous integration / Continuous delivery pipeline

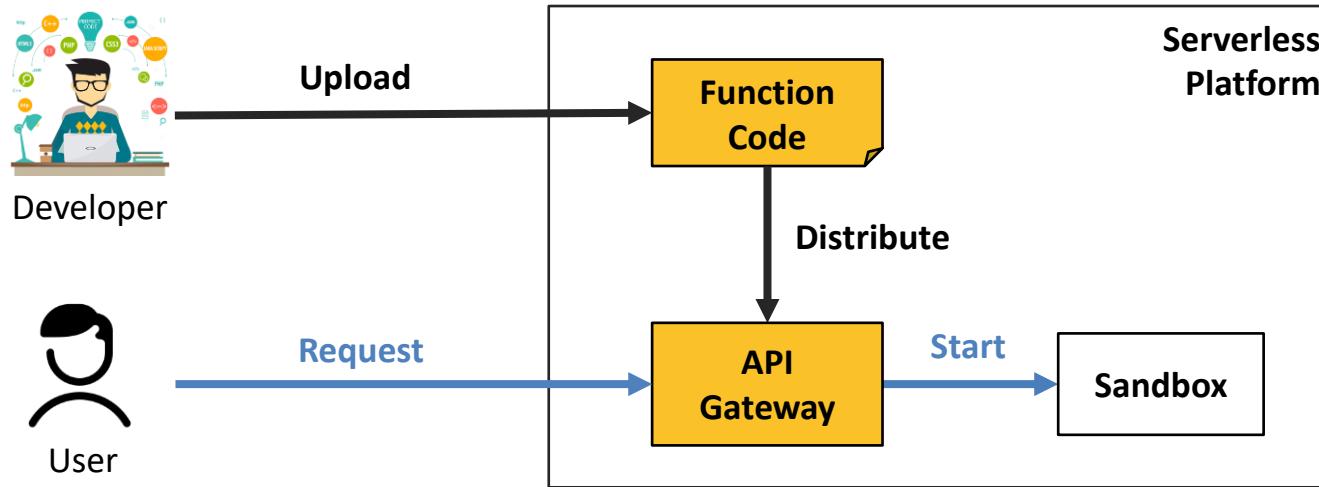
- Performance
- Security
- Long-running tasks support
- Fault tolerance mechanisms
- Function composition support
- Language runtime support

Serverless Key Metrics



- **1. Communication Performance**
 - Function composition: used for complex serverless applications
 - Examples of composition models: *sequence chain* and *nested chain*
- **2. Startup Latency: cold vs warm start**
 - Startup latency is added to each request processing time, as the execution environment is prepared on-demand
 - Function execution time is usually short (in seconds or milliseconds)
 - Autoscaling makes it harder to keep tail latency low

Serverless Key Metrics (2)



- **3. Stateless Overhead**
 - **Explicit State** (needed by function logic): being passed using external storage services (e.g., AWS S3)
 - **Implicit State** (e.g., session cache): the loss of implicit states might hurt the performance
- **4. Resource Efficiency**
 - For platform: ability to co-locate serverless functions with other workloads for utilisation improvement
 - For users: how to provision resources for better performance and economy benefit

Conclusion

- Introduced serverless architectures
- Learned how they employ nanoservices (versus microservices) to increase application scalability, while lowering the price of delivery
- Introduced the serverless computing execution model and the Function-as-a-Service concept
- Explained the relationship of functions, as they are used in AWS Lambda, to functional programming.

References

- Vladimir Yussupov et al. A Systematic Mapping Study on Engineering Function-as-a-Service Platforms and Tools. In Proceedings of UCC'2019, Auckland, New Zealand, December 2019, pages 229–240

<https://dl.acm.org/doi/pdf/10.1145/3344341.3368803>

Lab. Session

- Introduction to Azure Functions
- This is important for coursework 2.



Distributed Systems and Services Group

UNIVERSITY OF LEEDS

COMP5123M - CLOUD COMPUTING SYSTEMS

INTRODUCTION TO BIG DATA SYSTEMS



Goals

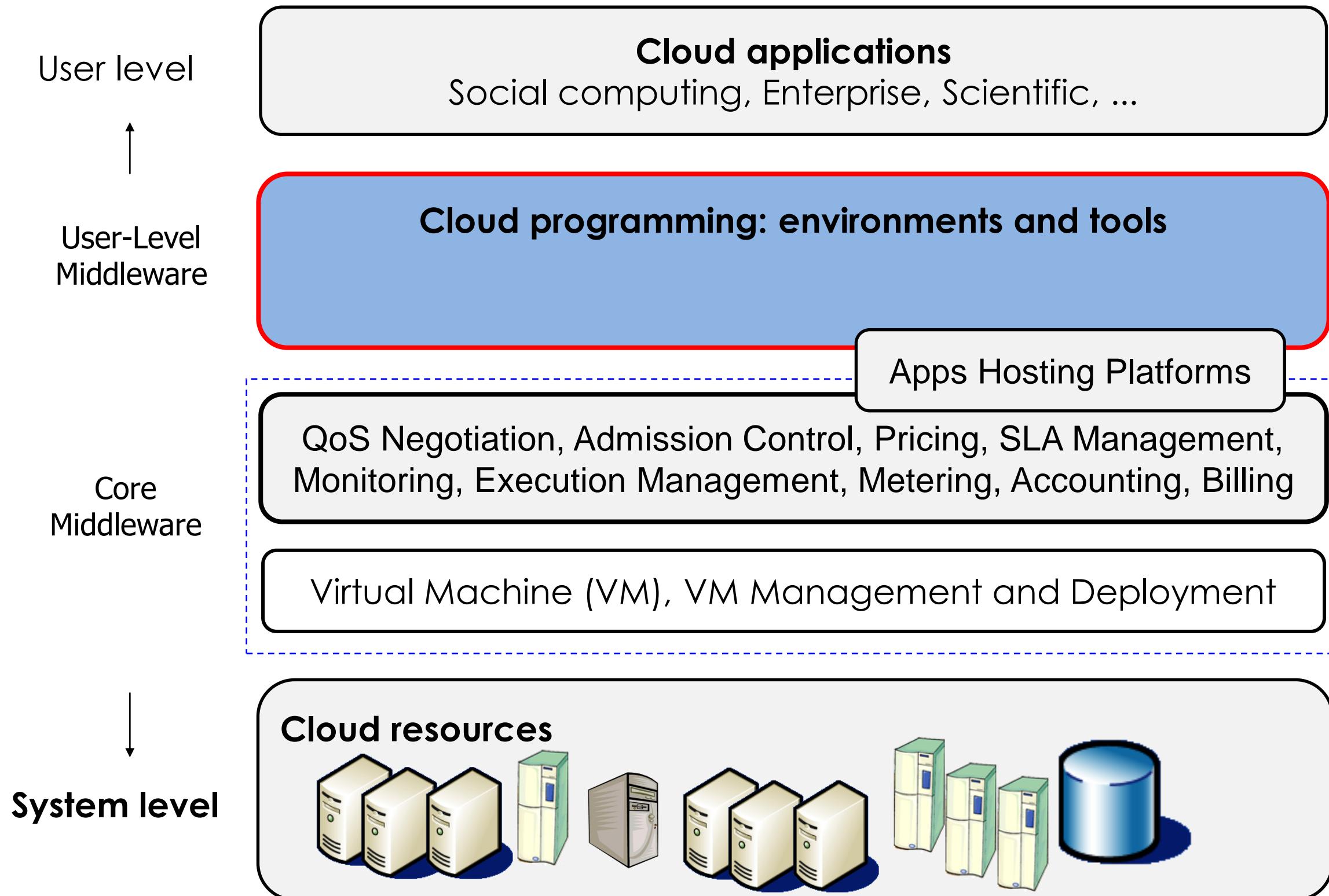
- Appreciate the generation and use of big data through examples
- Understand the technology landscape

Overview

- Introduction
- Big data classification
- Example scenarios
- Technology landscape
- Conclusion



Architectural Layer: Where Are We?



Big Data Everywhere!





Types of Data

UNIVERSITY OF LEEDS

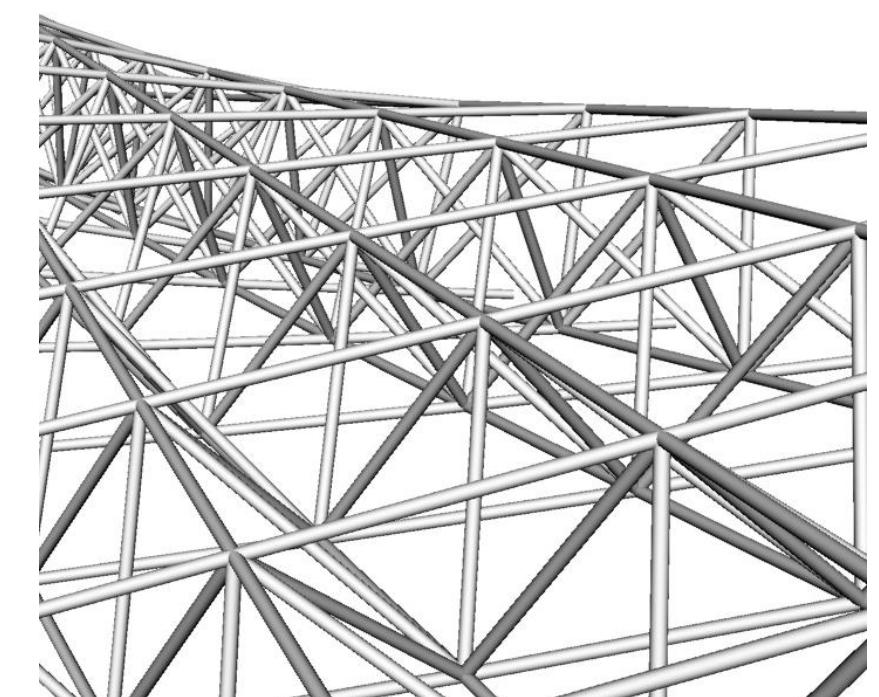
- Relational Data
 - Tables/Transactions/Legacy)
- Text Data (Web)
- Semi-structured Data (XML)
- Graph Data
 - (Social Network, Semantic Web)
- Streaming Data
 - Can only be scanned once



Large



Complex



Semi-structured

How Much Data? – Daily Generation



15 Exabytes of data
stored + processes
100 PBs a day



300 PB of user
data + processes
600 TB/day



90 PB of data stored +
processes 100PB/day



40 PBs a year



阿里云计算
Alibaba Cloud Computing

Note 1: these figures need to be revisited

Note 2: by 2025, the amount of data generated each day is expected to reach 400 exabytes globally.

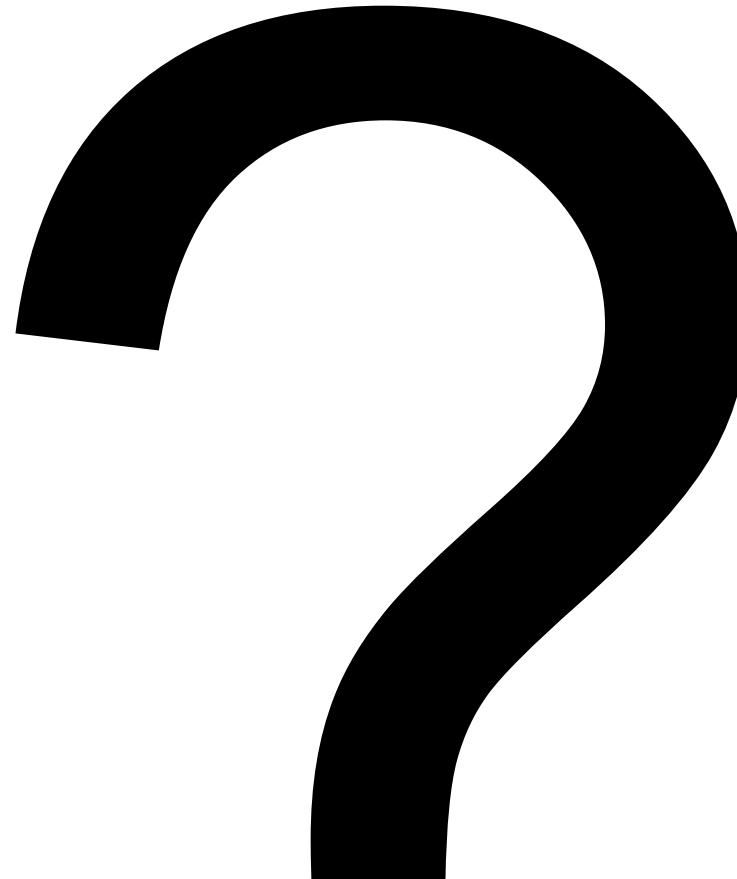


Why's this important?

UNIVERSITY OF LEEDS



Business
Intelligence (BI)



Discovery



Environmental

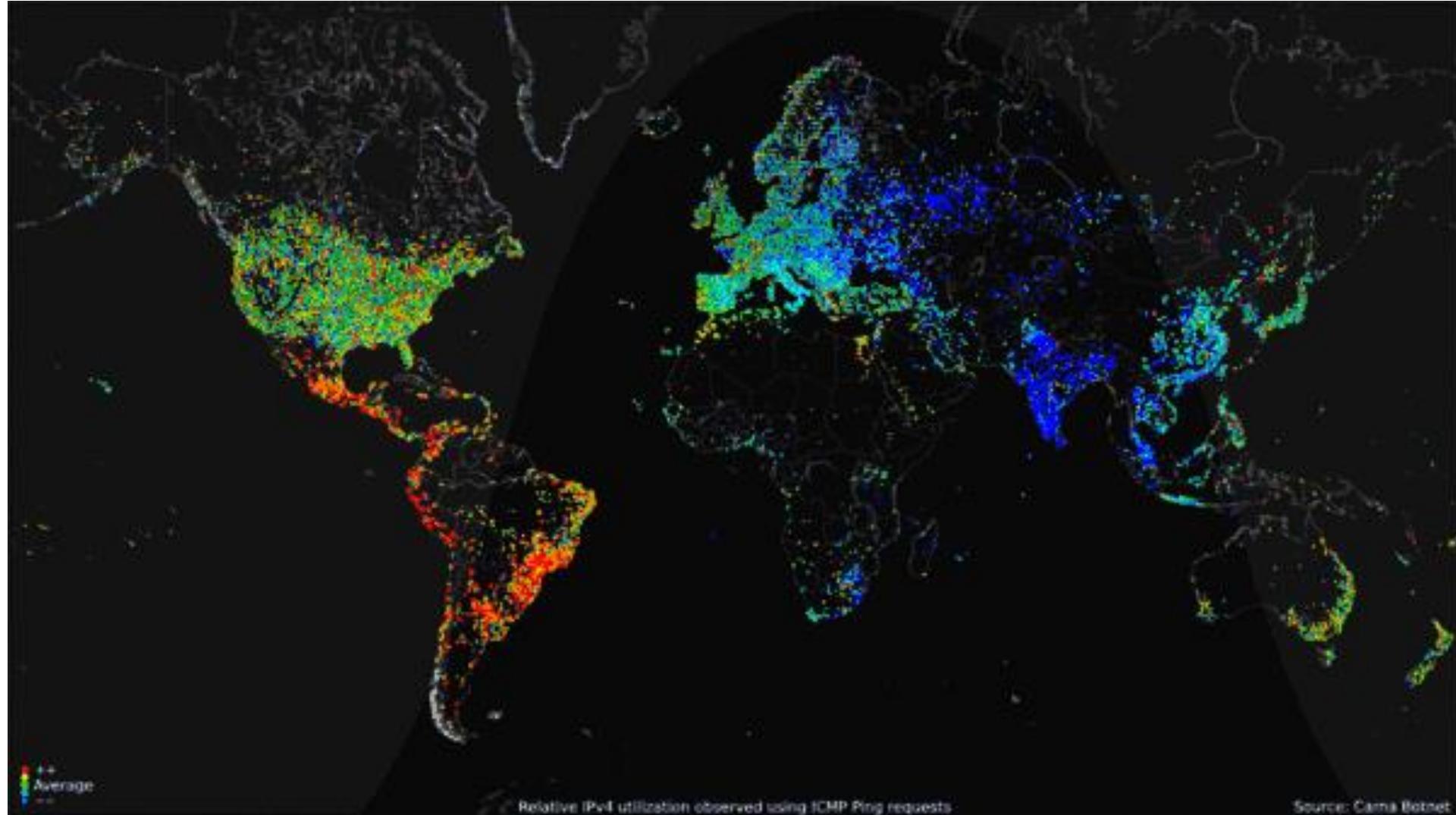


Monetary



- What are the behavioural patterns of users?
- How economically efficient are components within the system?
- What aspects of the system are draining resources needlessly?
- How will future business growth impact system operation?
- What applications are popular; when and why?





Real-time data visualization and non-intuitive correlations of scientific data.



Specific uses

- **Indexing, searching, querying**
 - Keyword based search
 - Pattern matching (XML/RDF)
 - Deep learning
- **Knowledge discovery**
 - Data mining
 - Statistical modelling
- **Aggregation and Statistics**
- **Modeling and forecasting**



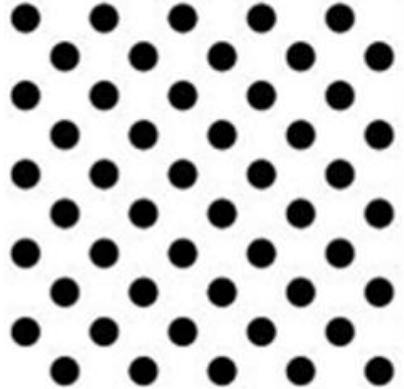
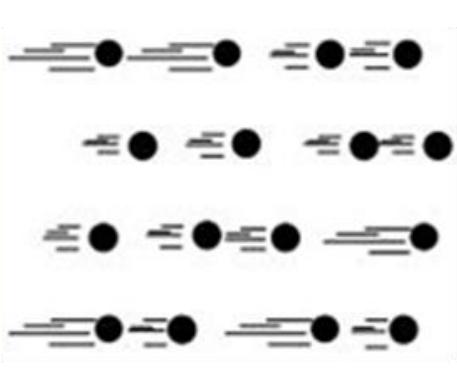
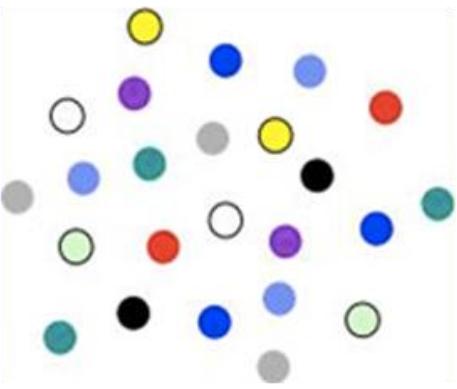
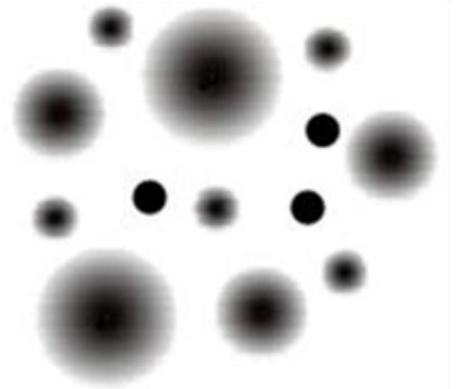
What is “Big Data”

1. Ability to manage, analyze, summarize, visualize, and discover knowledge from collected data in a timely manner and in a scalable fashion
2. Doing “**something**” **beneficial** with it

How big is it?

- Complete works of Shakespeare
- US Library of Congress
- Facebook uploads
- Daily data production by humanity

We have seen that Big Data is not a single technology, technique, or initiative. Rather, it is a **characterisable trend**.

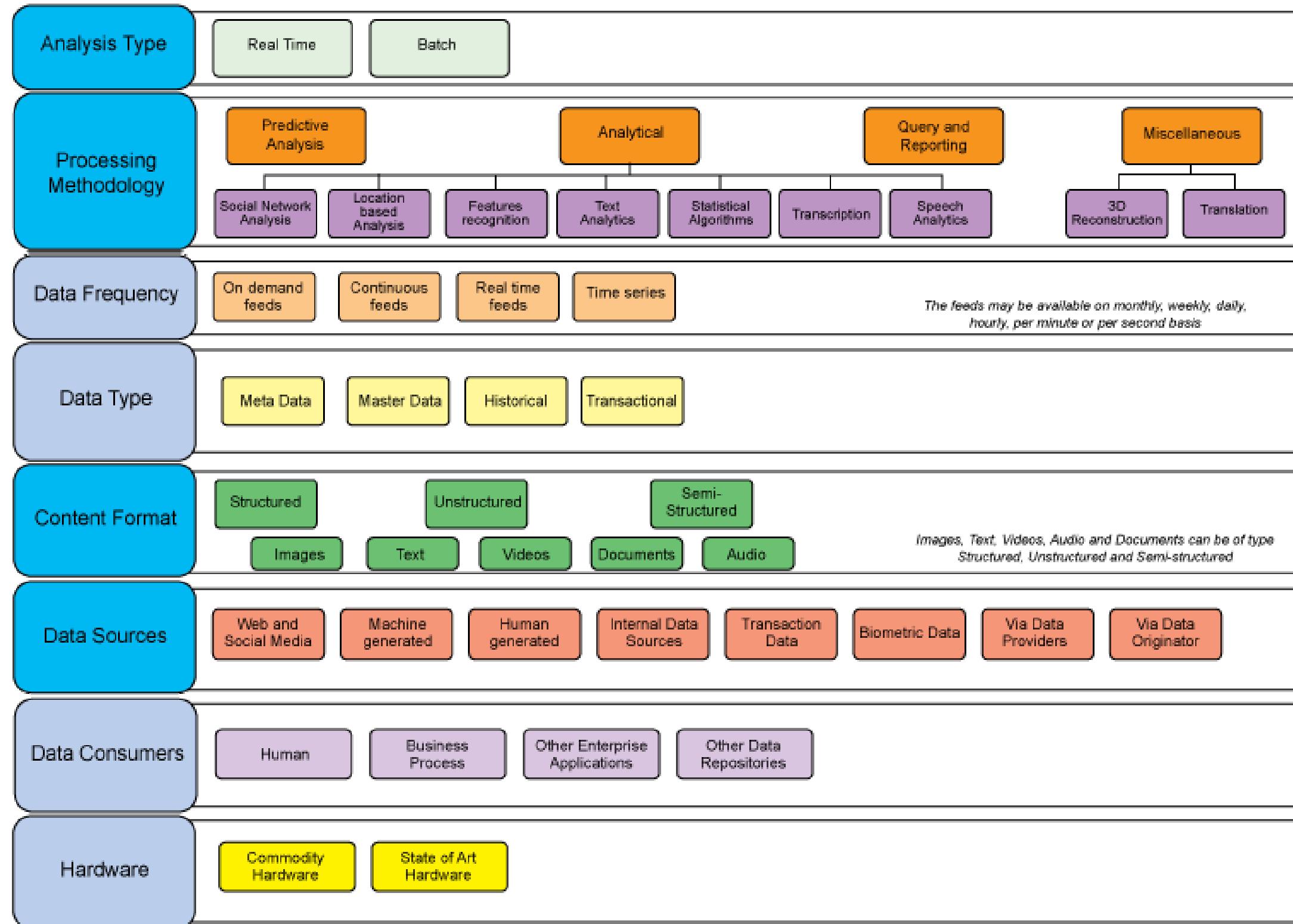
Volume	Velocity	Variety	Veracity	Value
				
Data at Rest Terabytes to Exabytes of existing data to process	Data in Motion Streaming data, requiring milliseconds to seconds to respond	Data in Many Forms Structured, unstructured, text, multimedia,...	Data in Doubt Uncertainty due to data inconsistency & incompleteness, ambiguities, latency, deception, model approximations	Data into Money Business models can be associated to the data

1. Data manipulation and analysis
 - Perform computations and ask well-defined questions
2. Data mining
 - Find patterns in data
3. Machine learning
 - Make inferences or predictions for missing or future data
4. Data visualisation
 - Depict the data graphically
 - Get Tableau, it's free for students:
<https://www.tableau.com/academic/students>
5. Data collection and preparation
 - Filling in missing values, removing suspicious data, formatting, etc.

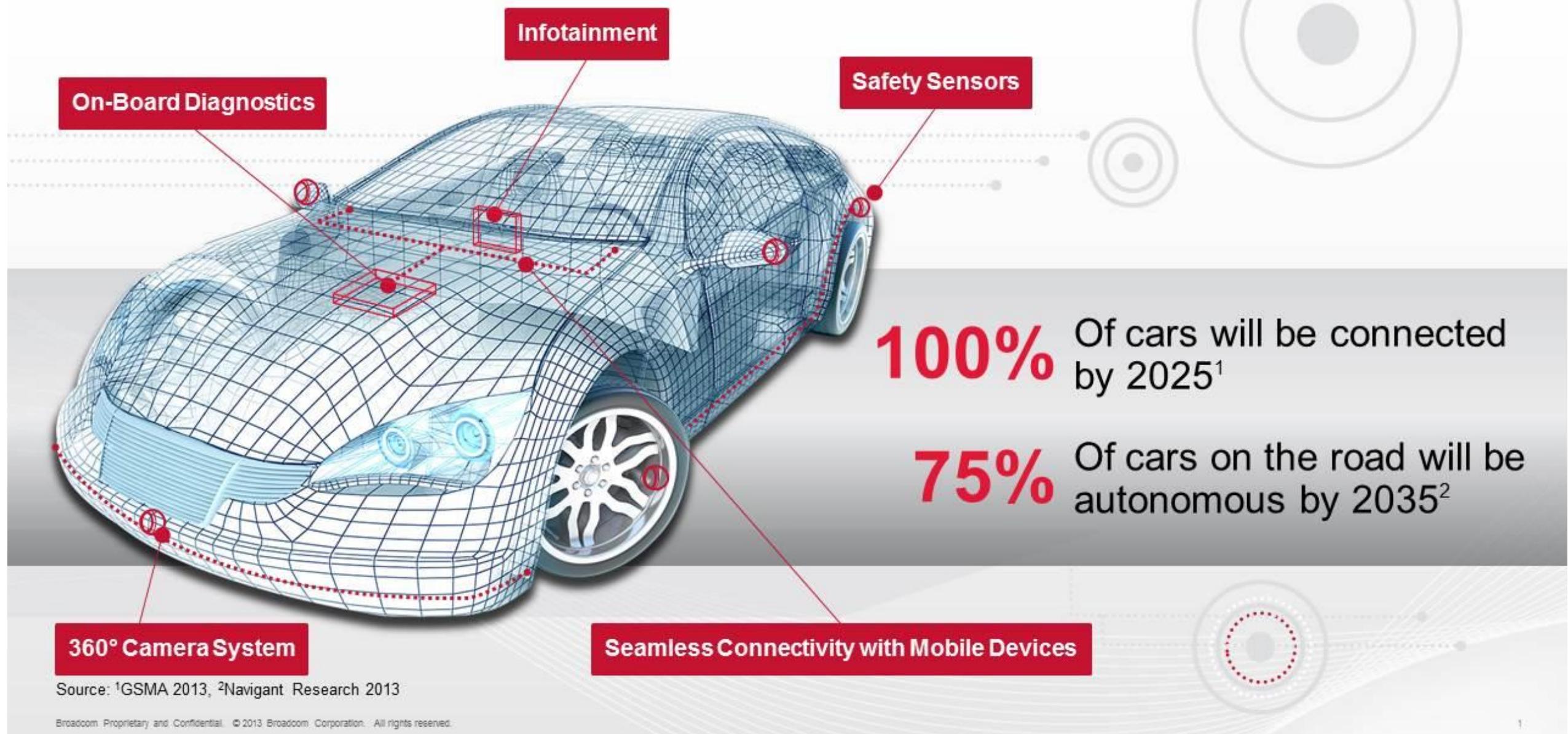


BIG DATA CLASSIFICATION

UNIVERSITY OF LEEDS



THE CONNECTED CAR



How does this relate to the 5 V's?

EXAMPLE: INTELLIGENT TRANSPORT



UNIVERSITY OF LEEDS



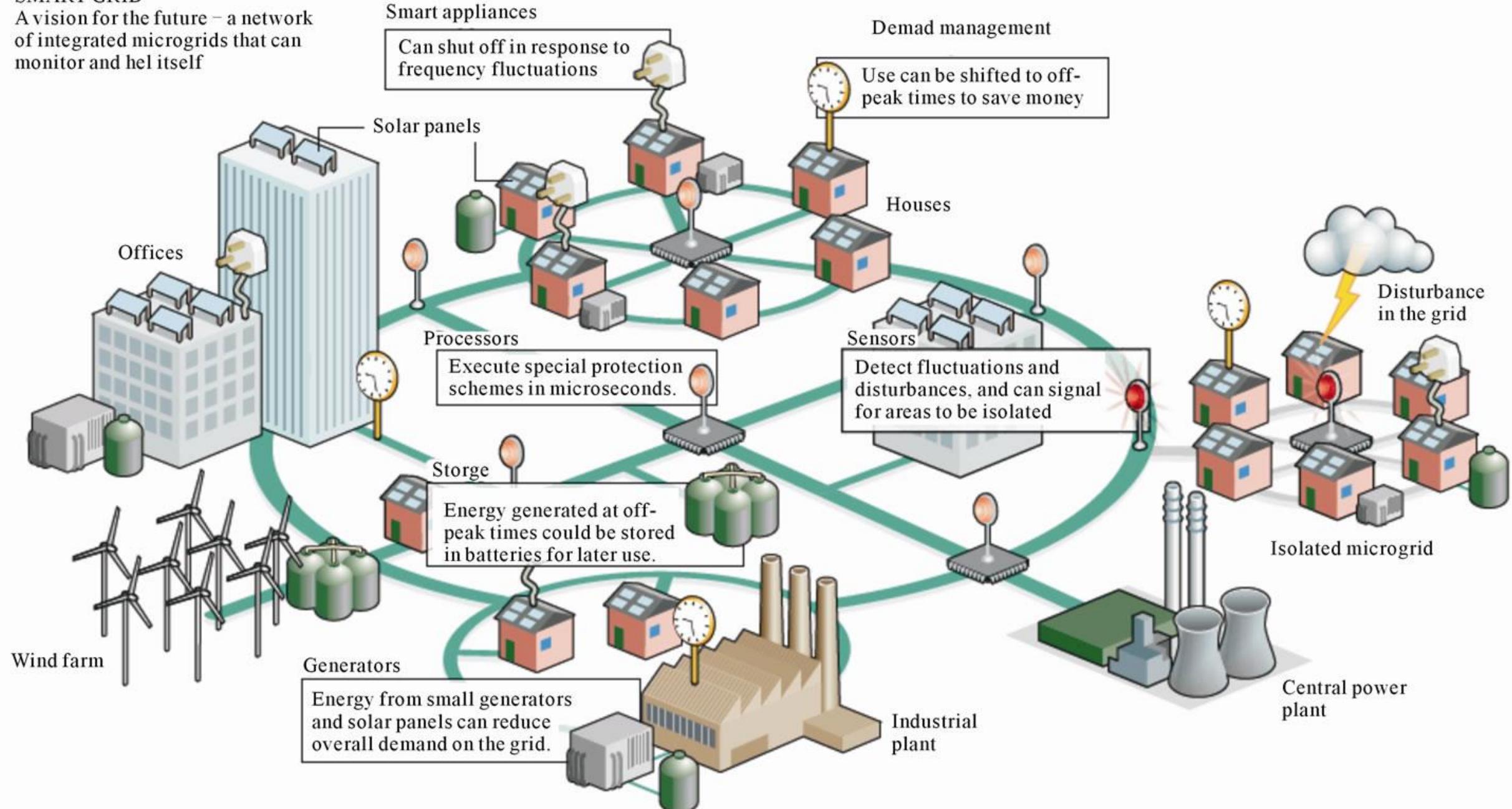
How does this relate to the 5 V's?

EXAMPLE: SMART GRID



SMART GRID

A vision for the future – a network of integrated microgrids that can monitor and help itself

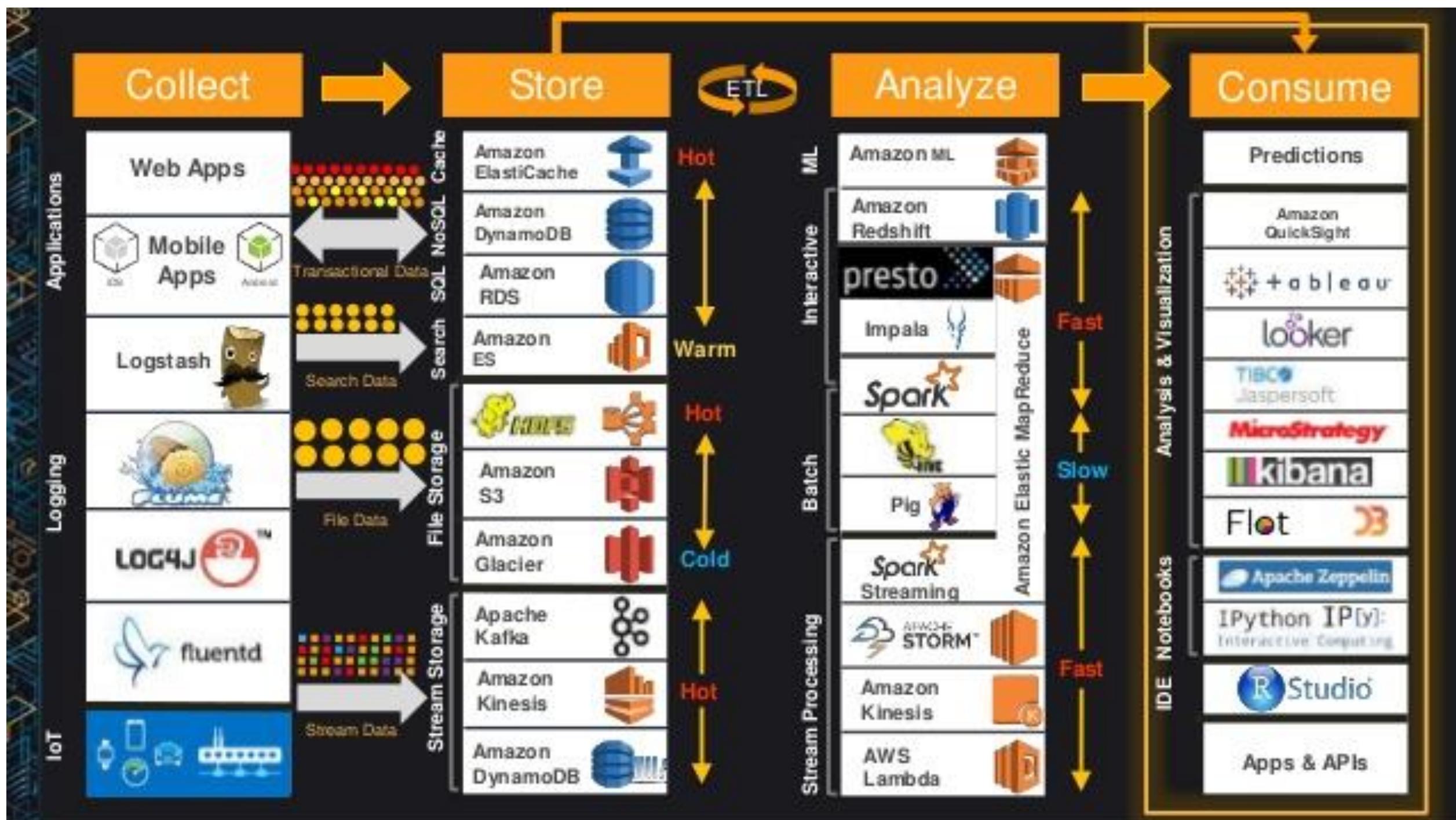


How does this relate to the 5 V's?

EXAMPLE: RECOMMENDER SYSTEMS



UNIVERSITY OF LEEDS

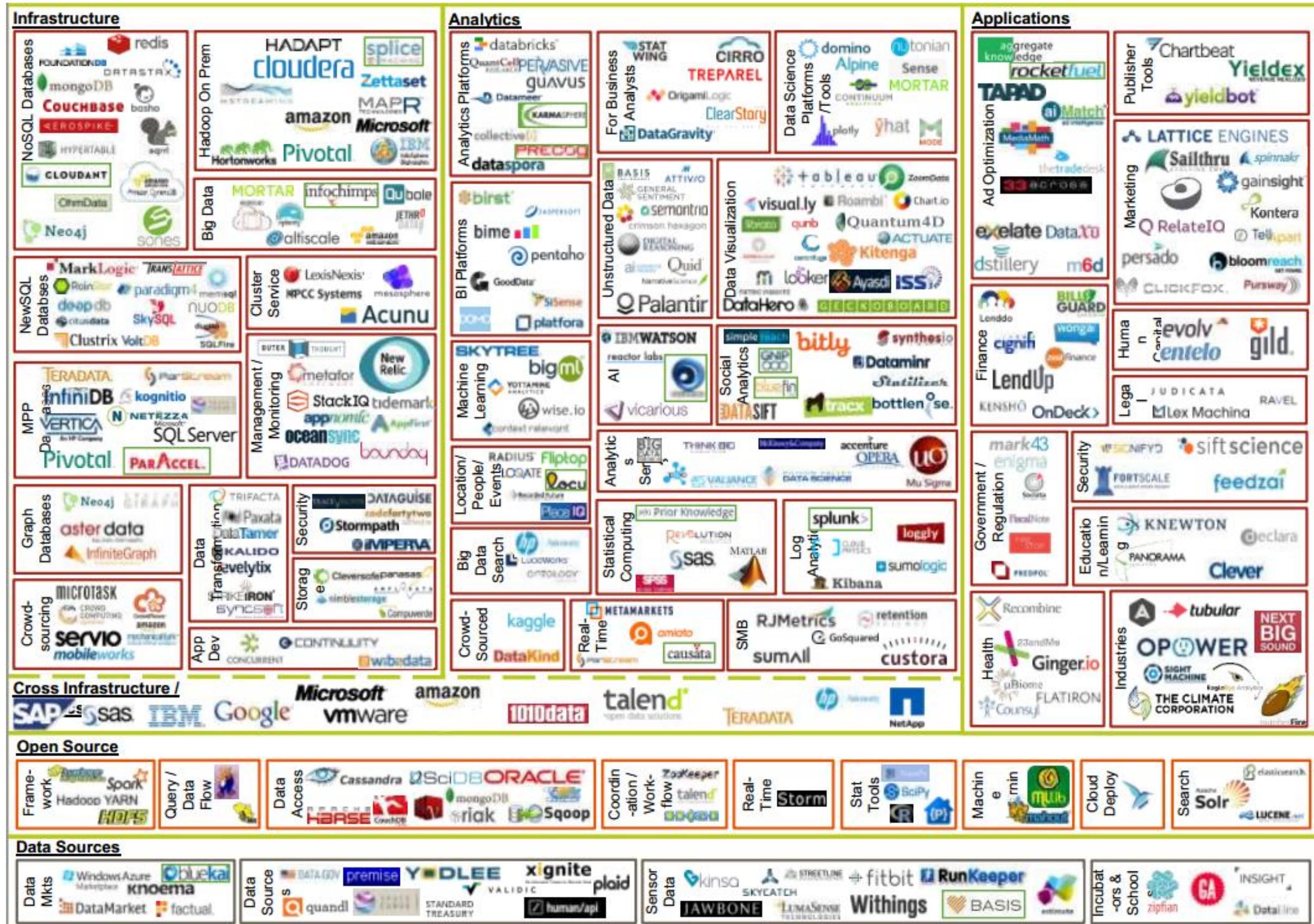


How does this relate to the 5 V's?

TECHNOLOGY LANDSCAPE SNAPSHOT



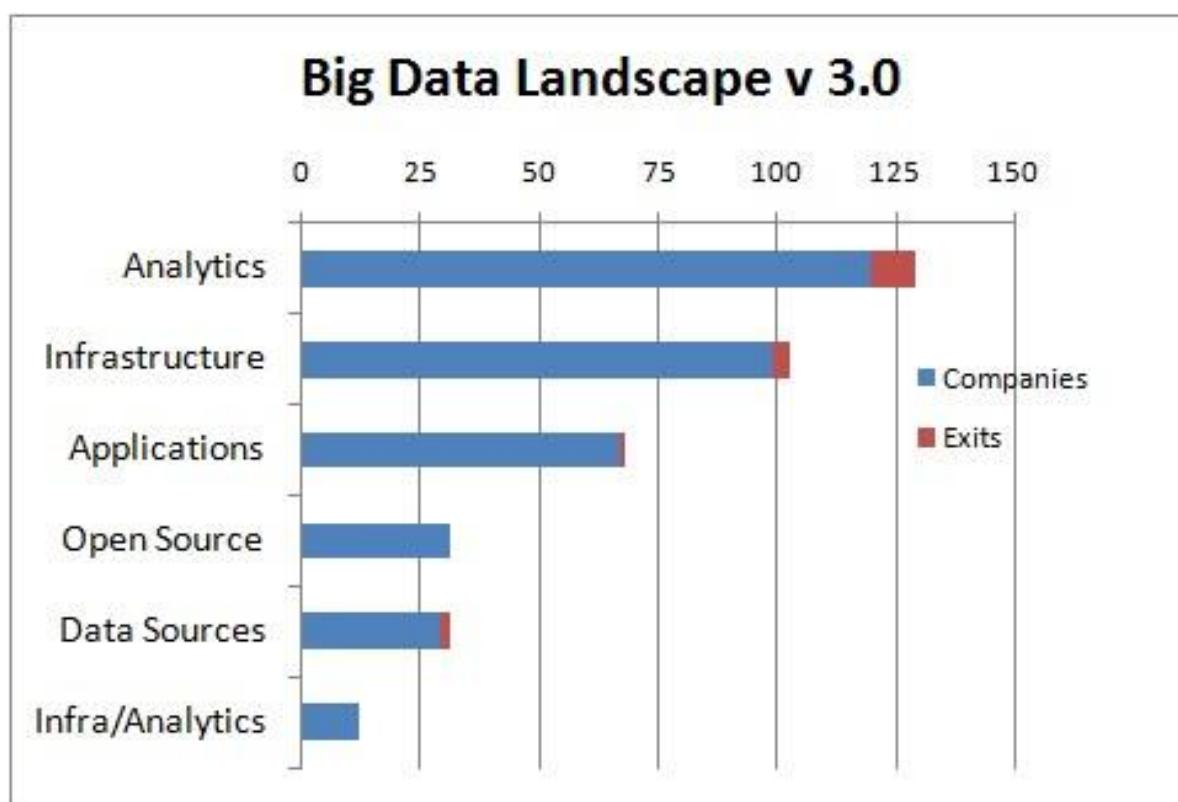
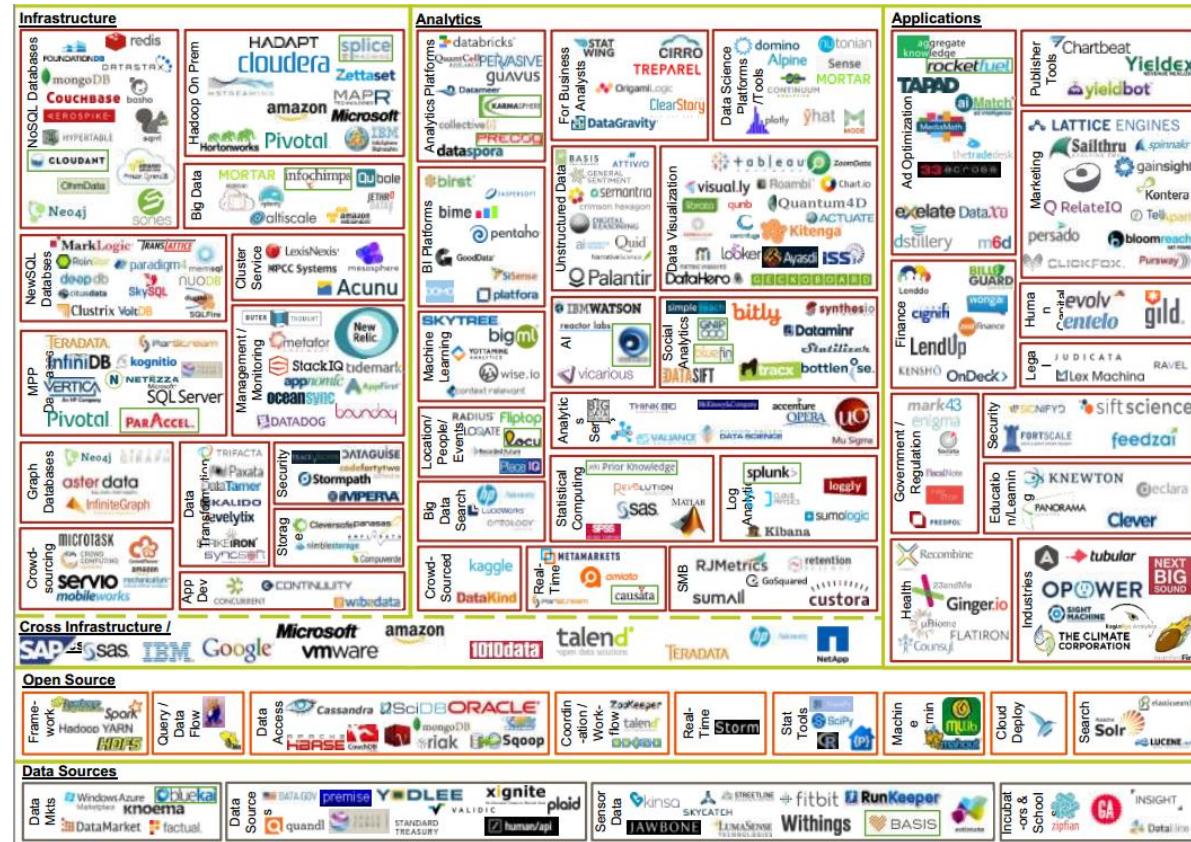
UNIVERSITY OF LEEDS



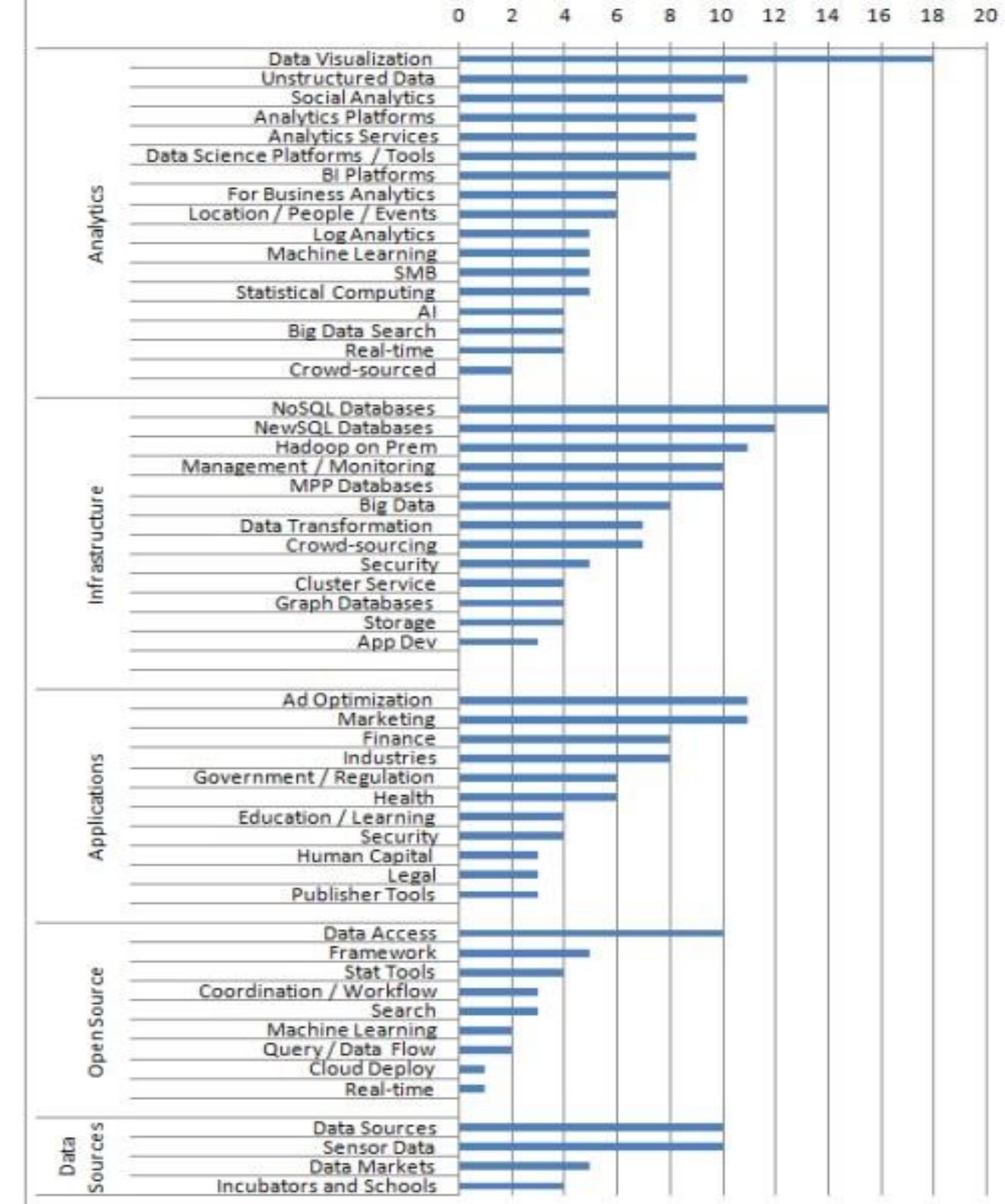
TECHNOLOGY LANDSCAPE SNAPSHOT (2)



UNIVERSITY OF LEEDS



Big Data Landscape v 3.0 by Sub-Categories



- Reviewed real-life big data examples
- Presented the technology landscape
 - More on this in the next lectures
- Reading
 - Chapter 1: a New Paradigm for Big Data. In: Big data : principles and best practices of scalable real-time data systems. Nathan Marz and James Warren. Manning, 2015
 - Hadoop: The Definitive Guide, 4th Edition, Chapter 1. Tom White. O'Reilly Media, 2015
- Practical Exercises
 - See Practical Exercises on Minerva



Distributed Systems and Services Group

UNIVERSITY OF LEEDS

COMP5123M Cloud Computing Systems

BIG DATA SYSTEMS – HIGH LEVEL ARCHITECTURE

Goals

- Understand key concepts in a big data architecture

Overview

- Techniques towards big data
- Layered architecture overview
 1. Sources
 2. Data messaging
 3. Analysis
 4. Consumption
- Information integration
- Data governance
- System management
- Example
- Conclusion

The challenge of Big Data is **how can we possibly deal with data that fits these characteristics**, both individually and collectively.

Volume

Velocity

Variety

Veracity

Value

Storage

Retrieval

Fusion

Analysis

The core of the answer lies in using **distributed resources**

Infrastructure

Multiple nodes

CPU or GPUs

FPGAs

ASICs

Memory

Storage

Network

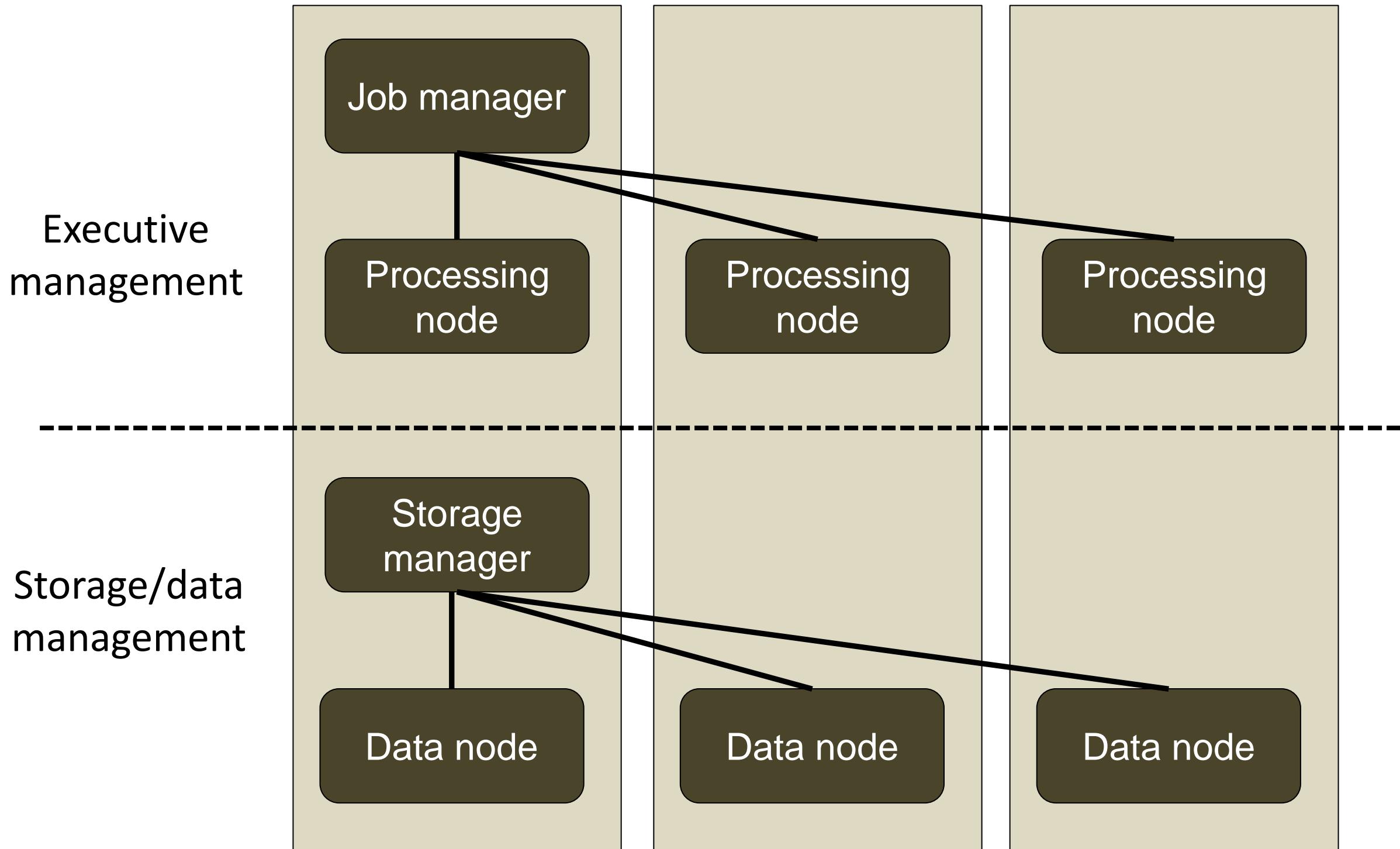
Methods

Task parallelisation

Data parallelisation

- Massive parallelism
- Huge data volumes storage
- Data distribution
- High-speed networks
- High-performance computing
- Task and thread management
- Data mining and analytics
- Data retrieval
- Machine learning
- Data visualisation

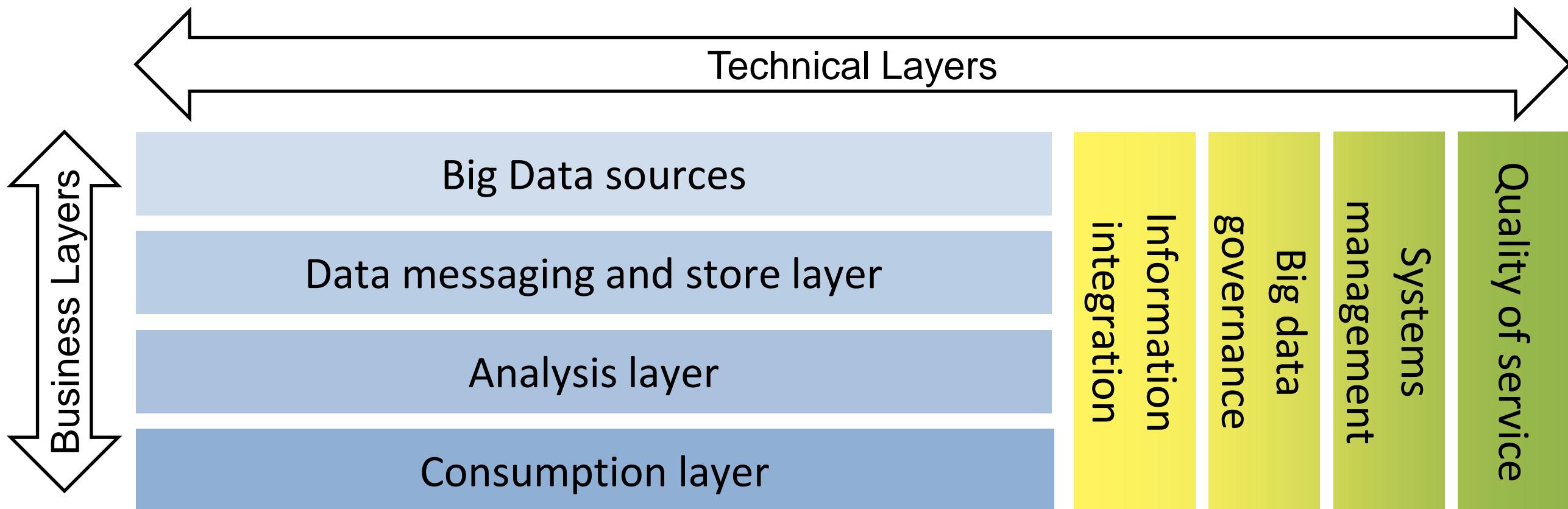
GENERIC HIGH-LEVEL DISTRIBUTED ARCHITECTURE



When designing an actual architecture for a Big Data system, it is best to organise with a **layered approach**.

Layers are a way to organise components that perform specific functions – they are **logical** and do not imply that the functions that support each layer are run on separate machines or processes.

A typical Big Data solution is typically architected around logical layers:





The sources layer refers to **all of the data available for analysis**, coming in from all channels. The type of analysis and sources are closely related.

FORMAT

Structured, semi-structured, or unstructured

VELOCITY AND VOLUME

The **speed** that data arrives and the **rate** at which it's delivered varies according to data source

COLLECTION POINT

Where the data is collected (directly or through data providers), in **real time** or in **batch** mode. The data can come from a primary or a secondary source.

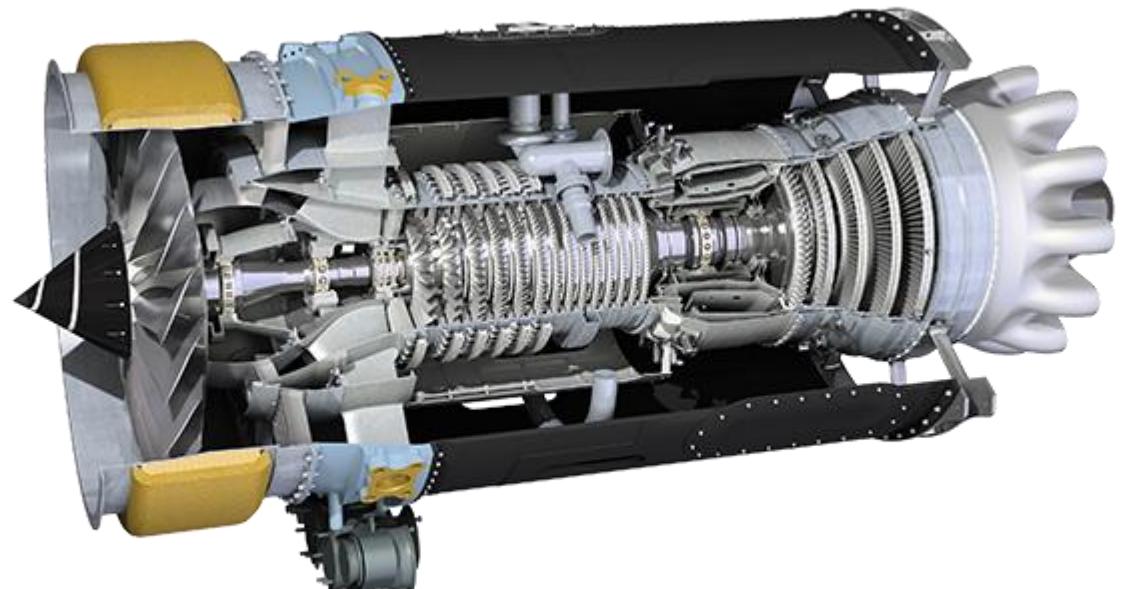
LOCATION OF DATA SOURCE

Data sources can be **inside** an enterprise or **external**. Identify the data to which you have limited-access, since access to data affects the scope of data available for analysis



Airline Example

- Real-Time Monitoring
 - Airline Data-Centre
 - Fleet data
 - Maintenance data
 - On-board systems
 - Avionics
 - Flight data recorder
- In-flight analysis
 - Feed engine data to data-centre
 - Analyse
 - Identify potential faults/vibrations etc.
 - Alert engineers (in real-time, at destination)



100,000 RR engines flying every day
1 engine = 1 GB data per flight

This layer is responsible for **acquiring data from data sources**.

If necessary, the layer will also be responsible for **converting the data to a format that suits how the data will be analysed**.

Compliance regulations and governance policies dictate the appropriate storage for different types of data.

Data Acquisition

- Acquires data from data sources and sends it to the data digest component or **stores** it in specified locations
- Must be intelligent enough to choose **whether and where to store the incoming data**.

Data Digest

- Massage data into a format required to achieve the purpose of the analysis
- This component can have **simple transformation logic or complex statistical algorithms to convert source data**
- The analysis engine determines **the specific data formats** that are required
- Challenge: how to accommodate **unstructured data formats**.

Distributed storage

- Responsible for **storing the data** from data sources.
- Often, multiple data storage **options** are available in this layer, such as distributed file storage (DFS), cloud, structured data sources, NoSQL, etc.

The analysis layer **reads the data digested by the data massaging and store layer** (in some cases, it accesses data directly from the data source).

Designing the analysis layer requires careful forethought and planning. Decisions must be made with regard to how to manage the tasks to:

Produce the desired analytics

Derive insight from the data

Find the entities required

Locate the data sources that can provide data for these entities

Understand what algorithms and tools are required

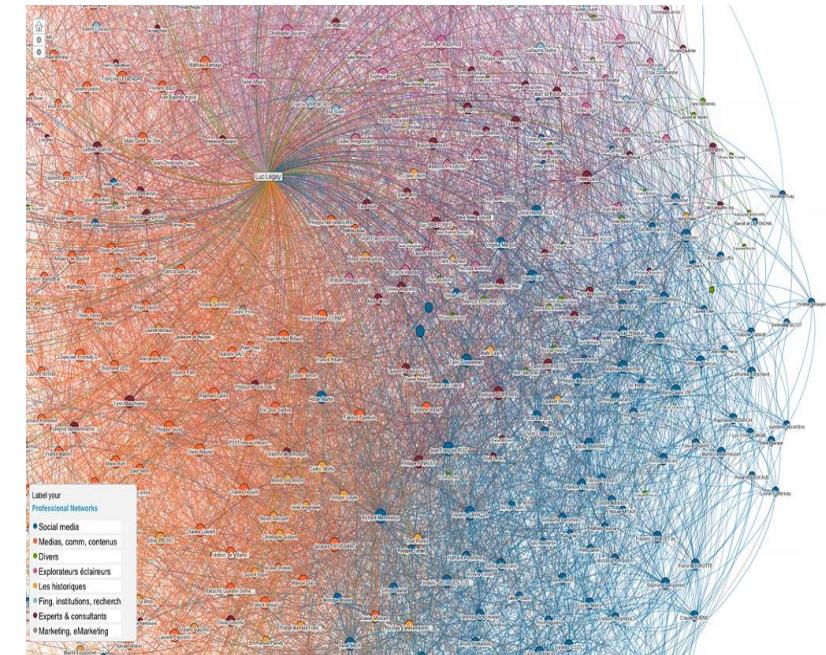
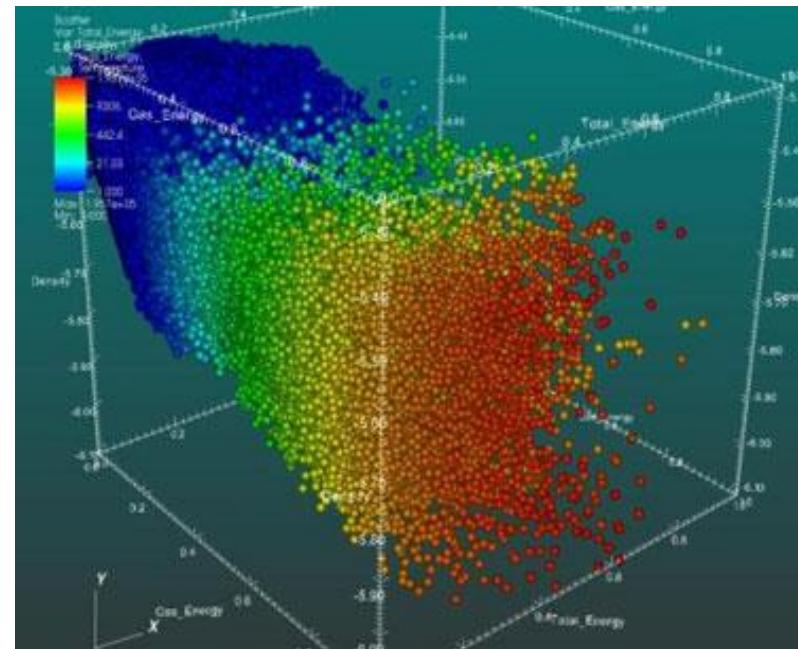
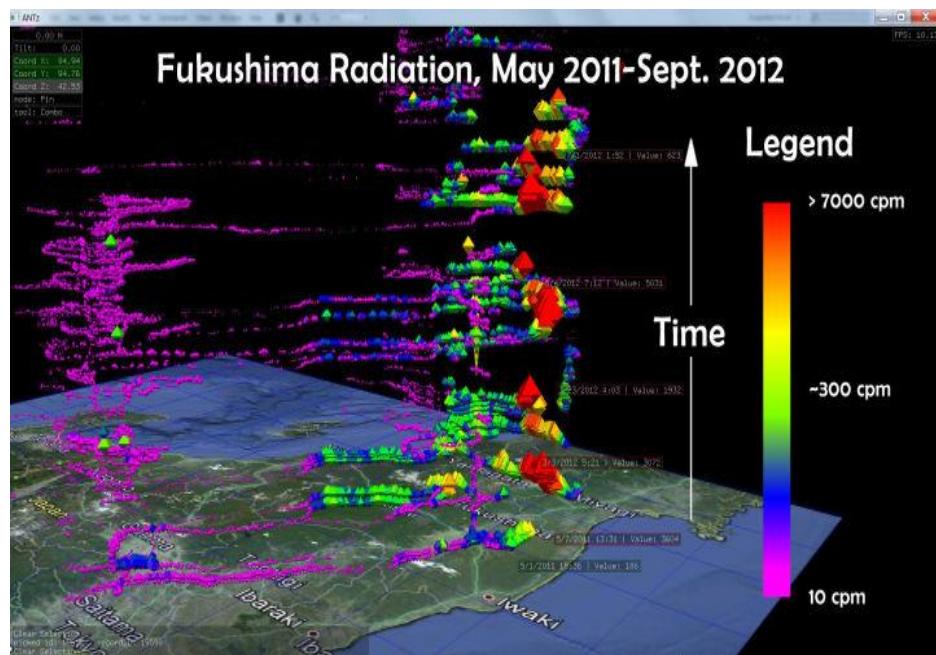


This layer consumes the output provided by the analysis layer.

The consumers can be visualization applications, human beings, business processes, or services.

It can be challenging to visualize the outcome of the analysis layer.

Sometimes it's helpful to look at what competitors in similar markets are doing.



CONSUMPTION LAYER

Transaction interceptor

Intercepts high-volume transactions **in real time** and converts them into a suitable format for real-time analysis.

Business process management processes

Insight from the analysis layer can be consumed by APIs and other processes **to drive value** by **automating functions** for upstream and downstream applications, people, and processes.

Real-time monitoring

Alerts can be generated using the data coming out of the analysis layer. These can be sent to interested consumers and devices.

Reporting engine

Producing reports similar to traditional business intelligence reports is critical.

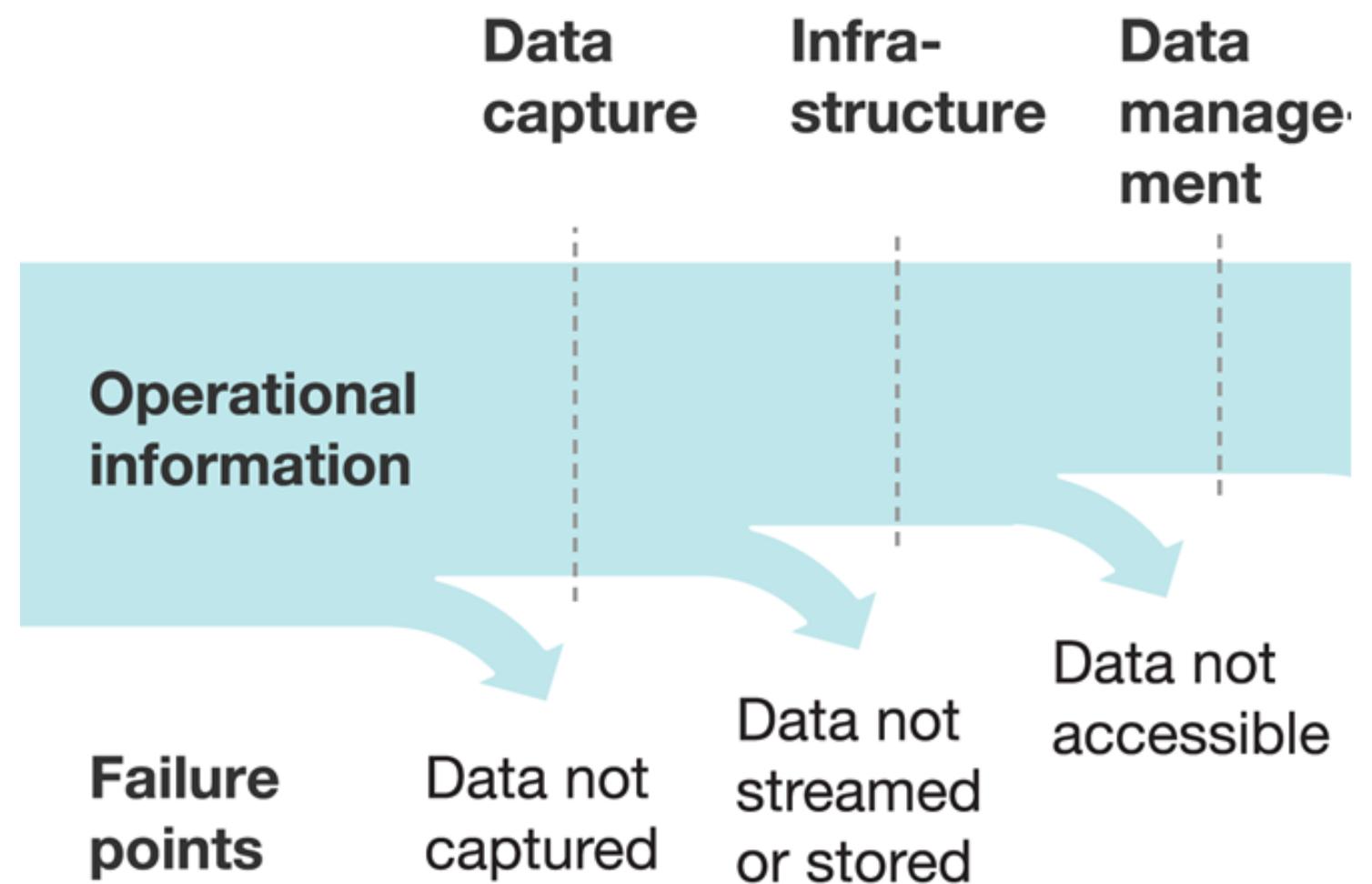
Recommendation engine

Based on analysis outputs, offers personalised, real-time, & relevant **recommendations** to users.

Visualisation and discovery

The data can vary in content and format, and can be **combined** for visualization.

- Responsible for connecting to various data sources.
- Requires quality connectors & adapters:
 - Communication protocols
 - APIs
 - Web Services
 - Others....?
- “Industry 4.0” platforms are designed to ease this part



Source: McKinsey & Company

Core principles

1. Lawfulness, fairness, & transparency
2. Purpose limitation
3. Data minimisation
4. Accuracy
5. Storage limitation
6. Integrity & confidentiality
7. Accountability

Must consider at least

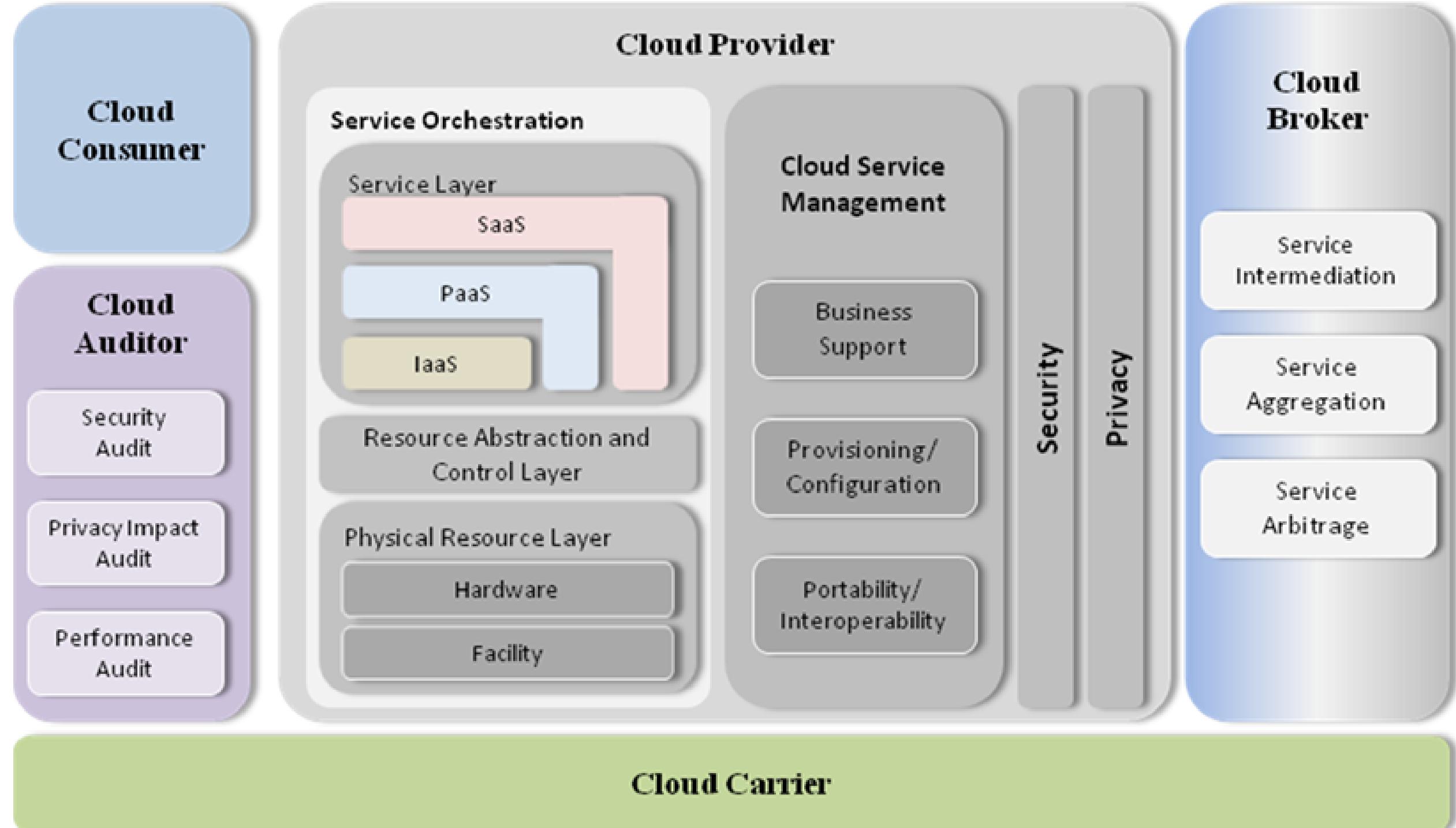
1. Data discovery
2. Security measures
3. Consent management
4. Data minimisation
5. Usage monitoring
6. Breach notification

GDPR: General Data Protection Regulation

Applies to processing of personal data

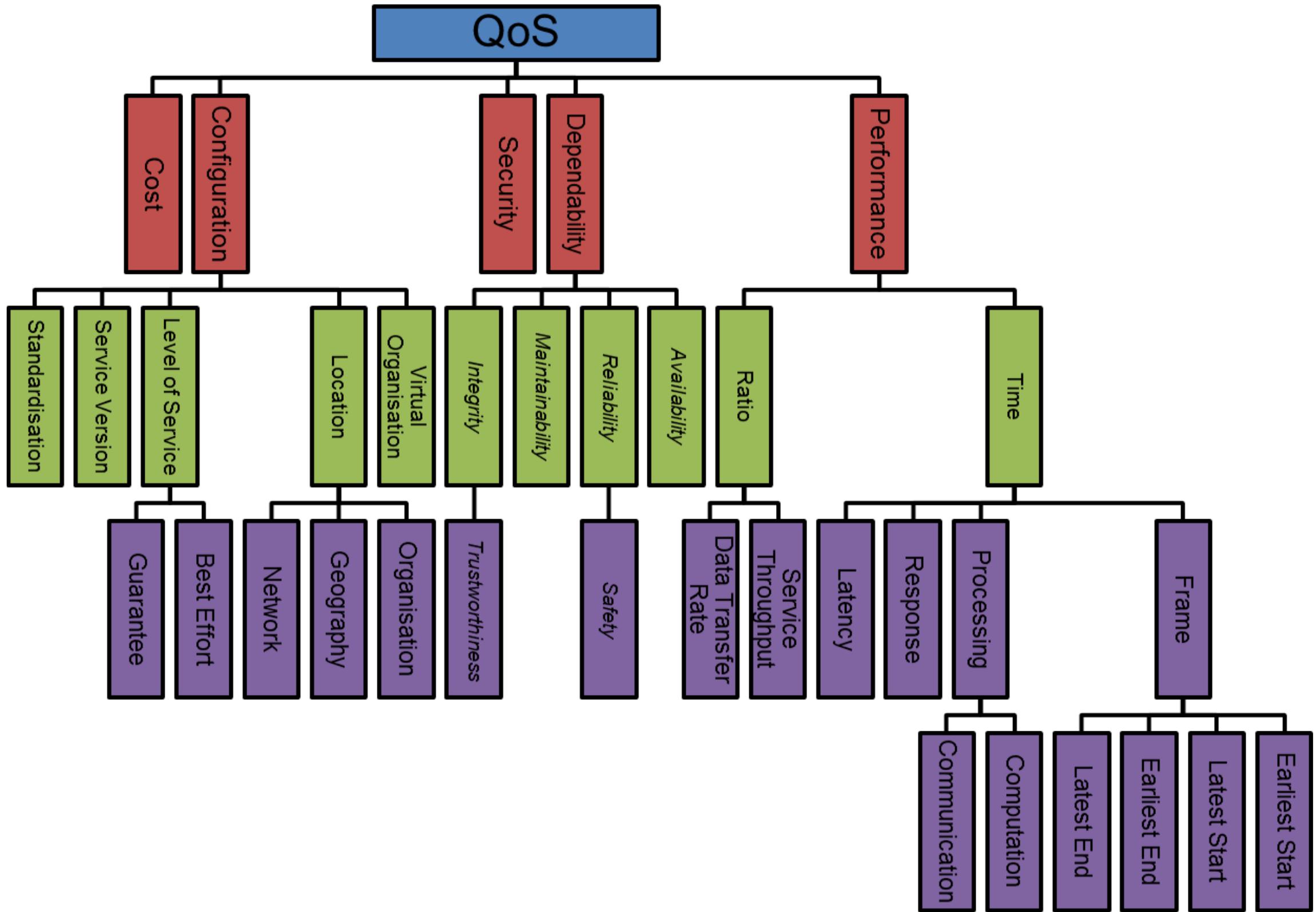
2 parties:

1. Data controller
2. Data processor





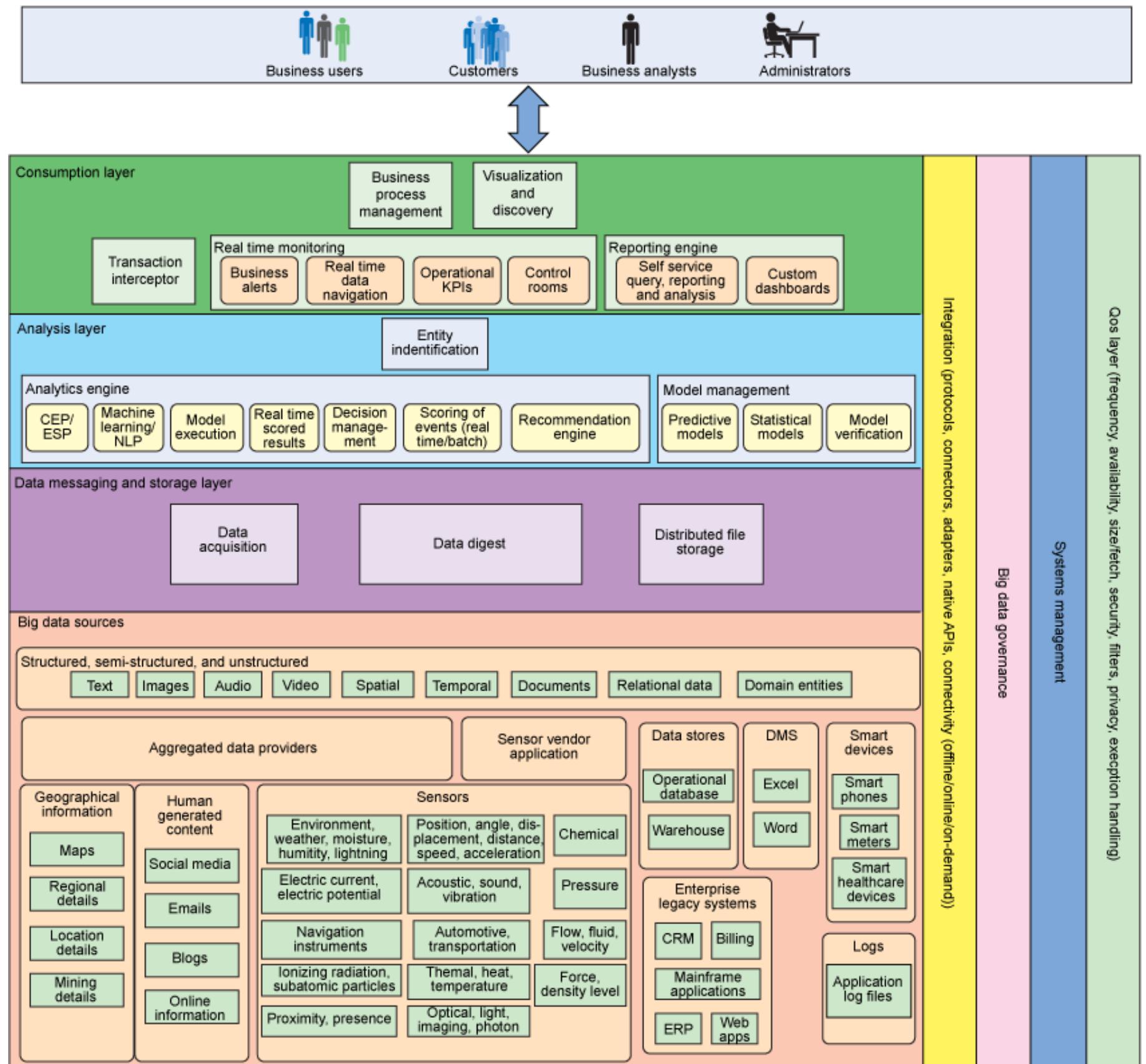
QUALITY OF SERVICE (QoS) LAYER



BIG DATA LAYERED ARCHITECTURE (IBM)



UNIVERSITY OF LEEDS



Information integration

Responsible for connecting to various data sources. Requires quality connectors & adapters

Big data governance

Strong guidelines & processes are required to monitor, structure, store, & secure data

Systems management

Monitoring and managing the health of the overall big data ecosystem.

Quality of service

Defines data quality, policies around privacy and security, frequency of data, etc.

	Traditional	Big Data
APPLICATION DEVELOPMENT	Applications that take advantage of parallelism developed by highly skilled HPC experts with optimisation and tuning.	Simplified application execution model (distributed file system, programming model, distributed database, and scheduler).
PLATFORM	Uses high-cost massively parallel processing (MPP) computers , utilising high-bandwidth networks and massive I/O devices	Innovative methods of creating scalable yet elastic virtualised platforms.
DATA MANAGEMENT	Limited to file-based or RDBMS using standard row-oriented data layouts	Alternate models for data management (often NoSQL) with variety of methods for managing information according to need.
RESOURCES	Requires large capital investment in purchasing high-end hardware to be installed and managed in-house.	Ability to deploy systems on virtualised platforms , especially clouds , giving a low barrier for entry.



Prescribed reading

- Big data : principles and best practices of scalable real-time data systems. Chapter 2. Nathan Marz and James Warren. Manning, 2015
- Hadoop: The Definitive Guide, 4th Edition, Chapter 1. Tom White. O'Reilly Media, 2015



COMP5123M Cloud Computing Systems

BIG DATA SYSTEMS – MAPREDUCE AND HADOOP



Plan of the Lecture

UNIVERSITY OF LEEDS

Goals

- Understand concepts of MapReduce/Hadoop
- Illustrate how MapReduce fits in Big Data Systems

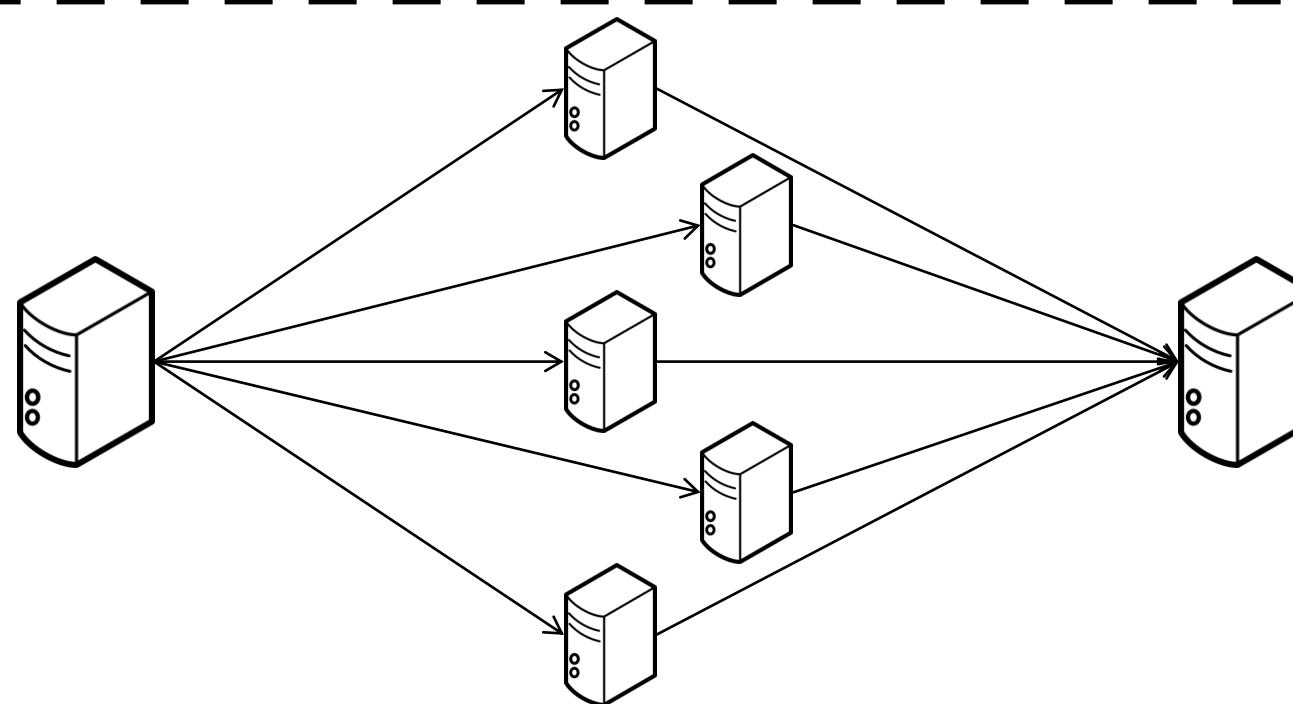
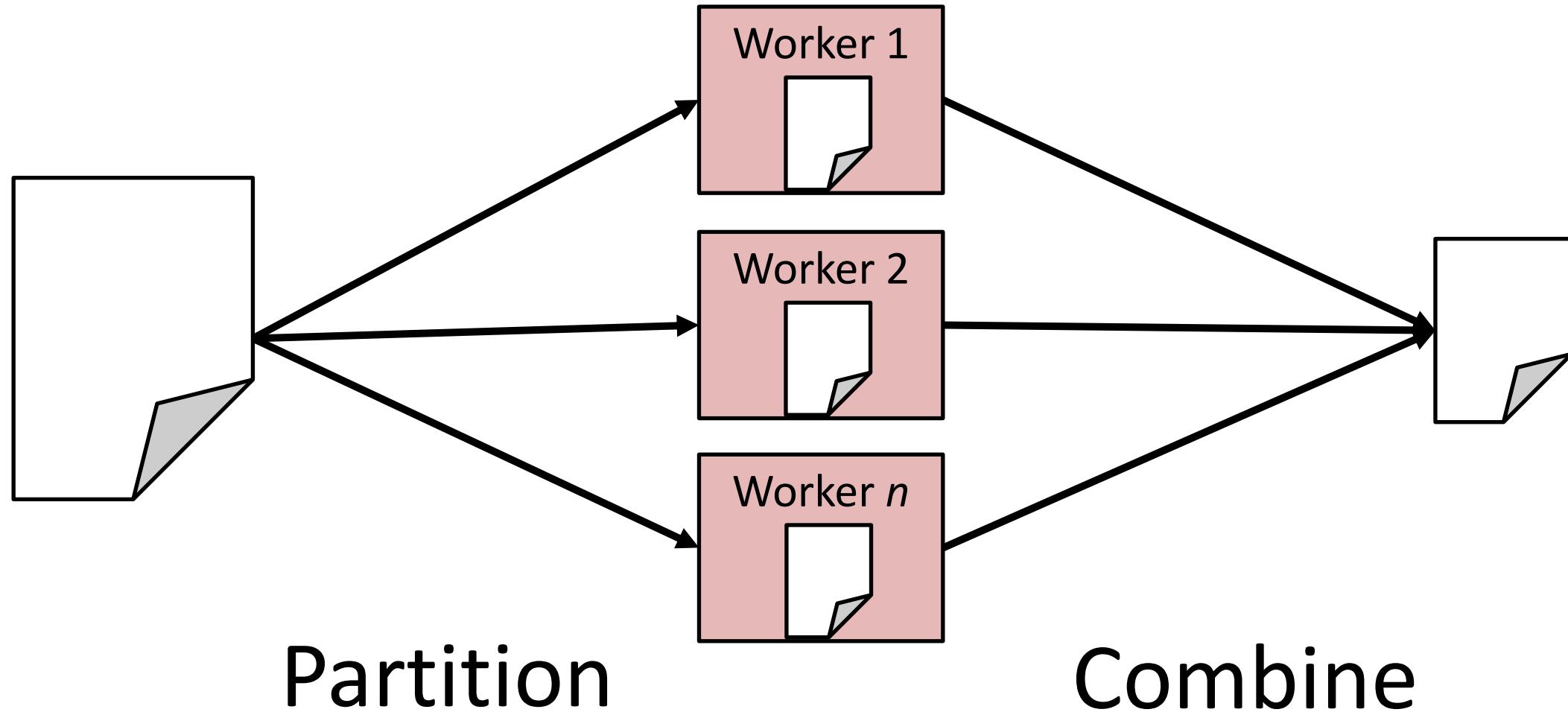
Overview

- Big Data challenges
- MapReduce
- Hadoop
- Example
- Conclusion



- **Massive volumes of data**
 - Rate of data production and types is rapidly increasing.
- **Data description**
 - Metadata and semantics for describing content.
- **Searching**
 - Semantically-enabled search engines.
- **Storage and retention**
 - Easier to recreate than to store
 - Useful/noise ratio
 - Companies typically store everything.
- **Governing data collection**
 - Information and knowledge
 - Quality control.

Potential solution: Parallelization



- **Scheduling**
 - How do we assign work to individual workers?
- **Availability**
 - What if there are more work units than available workers?
- **Dependencies**
 - What if workers need to share partial results?
 - How do we know when all workers have completed?
- **Fault-tolerance**
 - What if workers fail mid-execution?
- **Communication and synchronisation**
 - Between workers to exchange state, access shared resources (i.e. data).

Require a synchronization mechanism



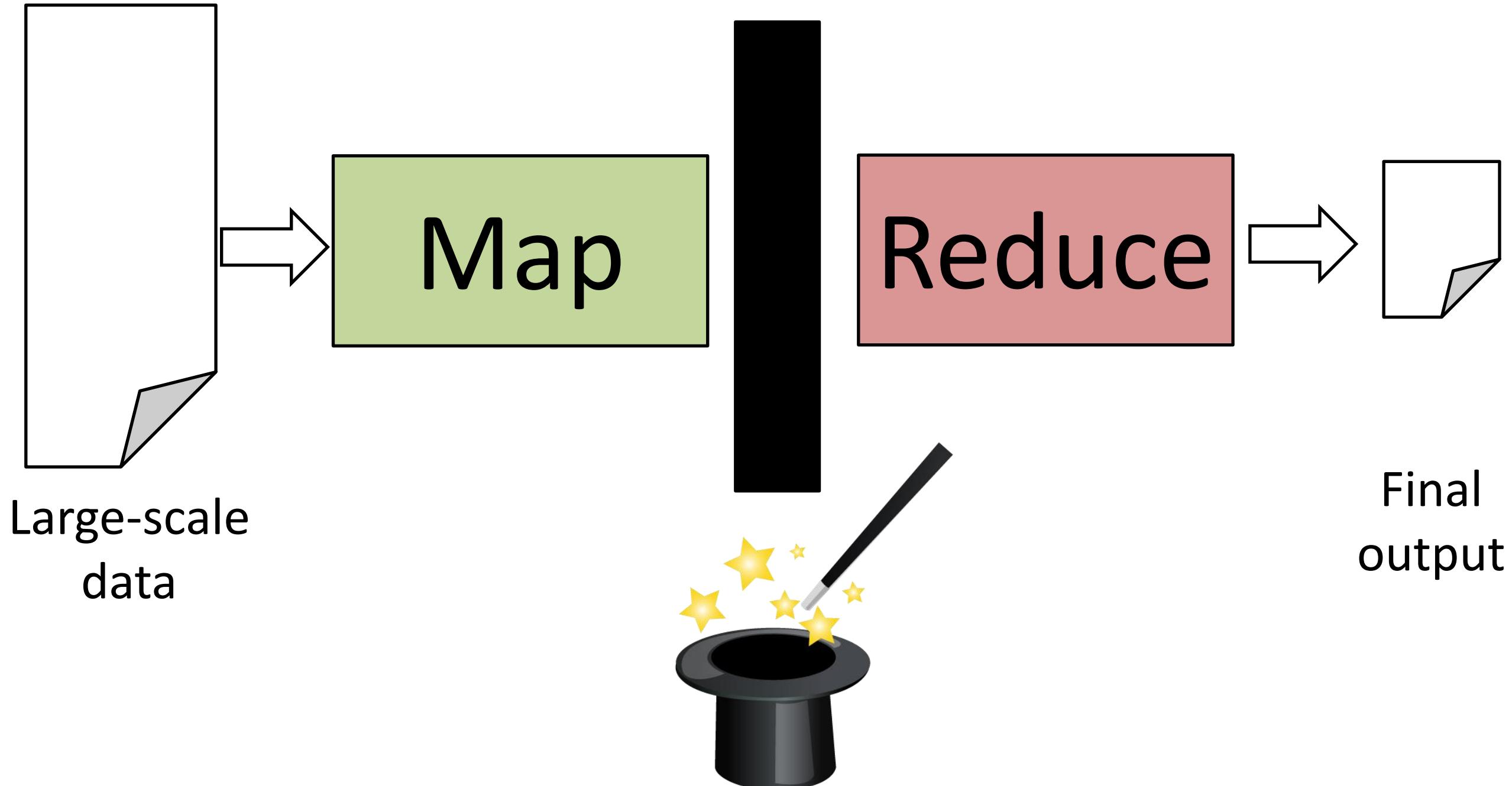
- **Concurrency is difficult when considering**
 - Datacenter scale
 - Faults and failures
 - Multiple interacting services.
- **Shifting the burden**
 - Developer specifies the computation that needs to be performed.
 - Execution framework (at runtime) that handles actual execution.



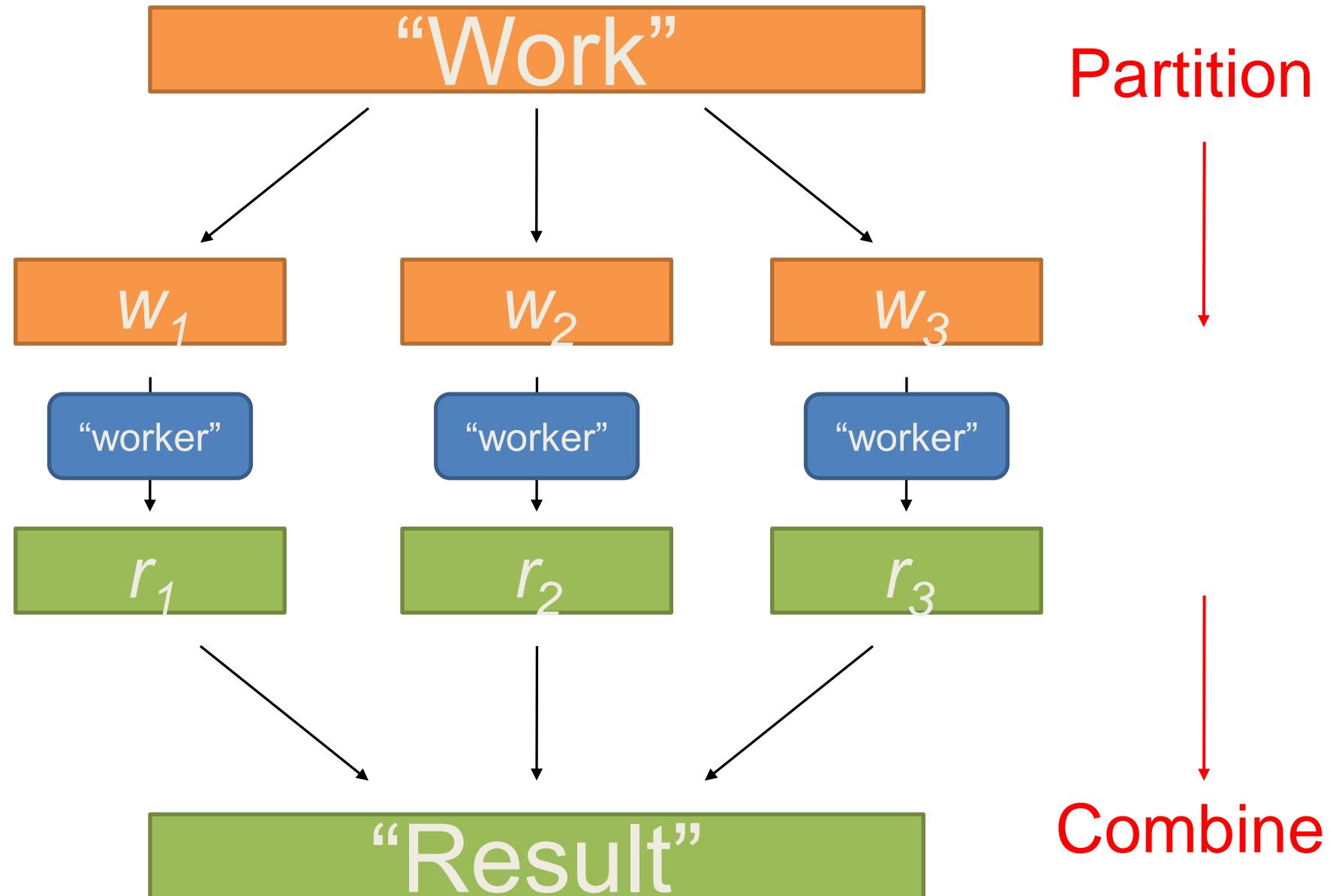


- Programming model for expressing distributed computation in massive-scale systems.
- Execution framework for organizing and performing computation
 - runs on commodity hardware in a reliable, fault-tolerant manner
- Originally developed by Google
- Open-source implementation called Hadoop.





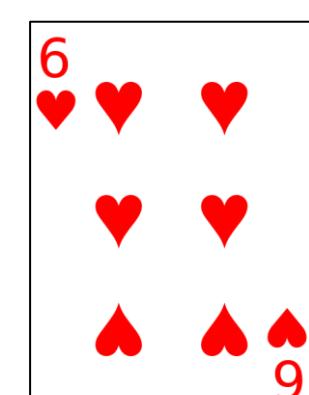
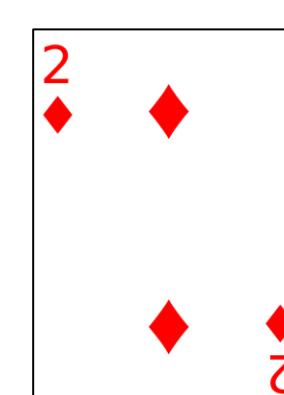
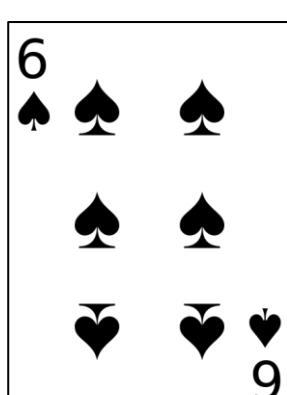
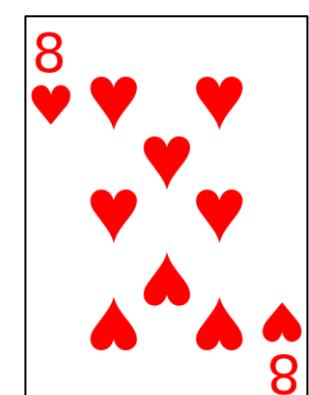
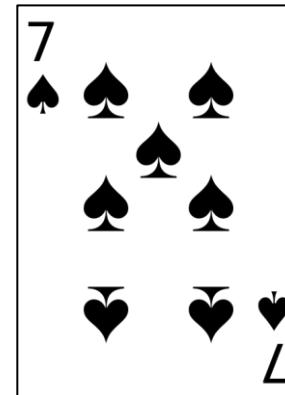
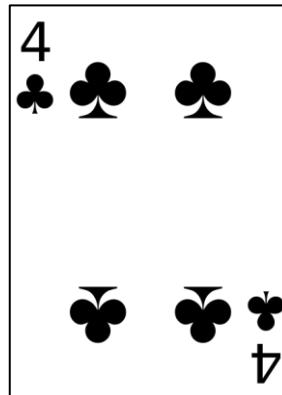
How do we scale up? Divide and Conquer





MapReduce process

- Iterate over a large number of records.
 - Extract something of interest from each. **Map**
 - Shuffle and sort intermediate results.
 - Aggregate intermediate results into something usable. **Reduce**
 - Generate final output.
-
- **Example**
 - Count the sum of card numbers, grouped by card suit.
 - Remove any picture cards
 - Map (Filter), Shuffle (sort), Reduce (count)





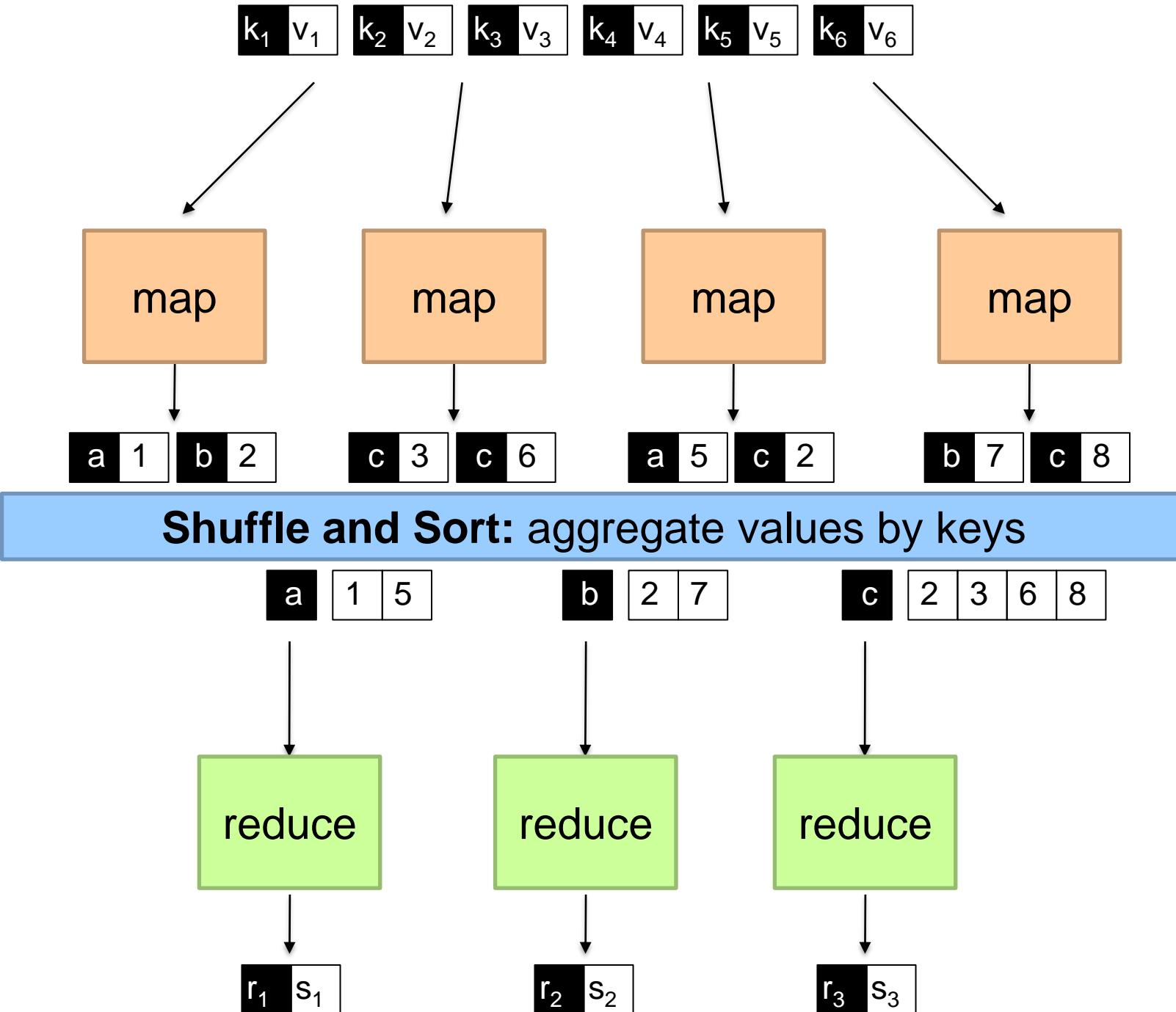
- Programmers specify two functions:
map $(k, v) \rightarrow \langle k', v' \rangle^*$
reduce $(k', v') \rightarrow \langle k', v' \rangle^*$
 - Value v is mapped to key k .
 - All values with the same key are sent to the same reducer.
- The execution framework handles everything else...

MapReduce “Runtime”

- Handles scheduling
 - Assigns workers to map and reduce tasks
- Handles “data distribution”
 - Moves processes to data
- Handles synchronization
 - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
 - Detects worker failures and restarts

MapReduce

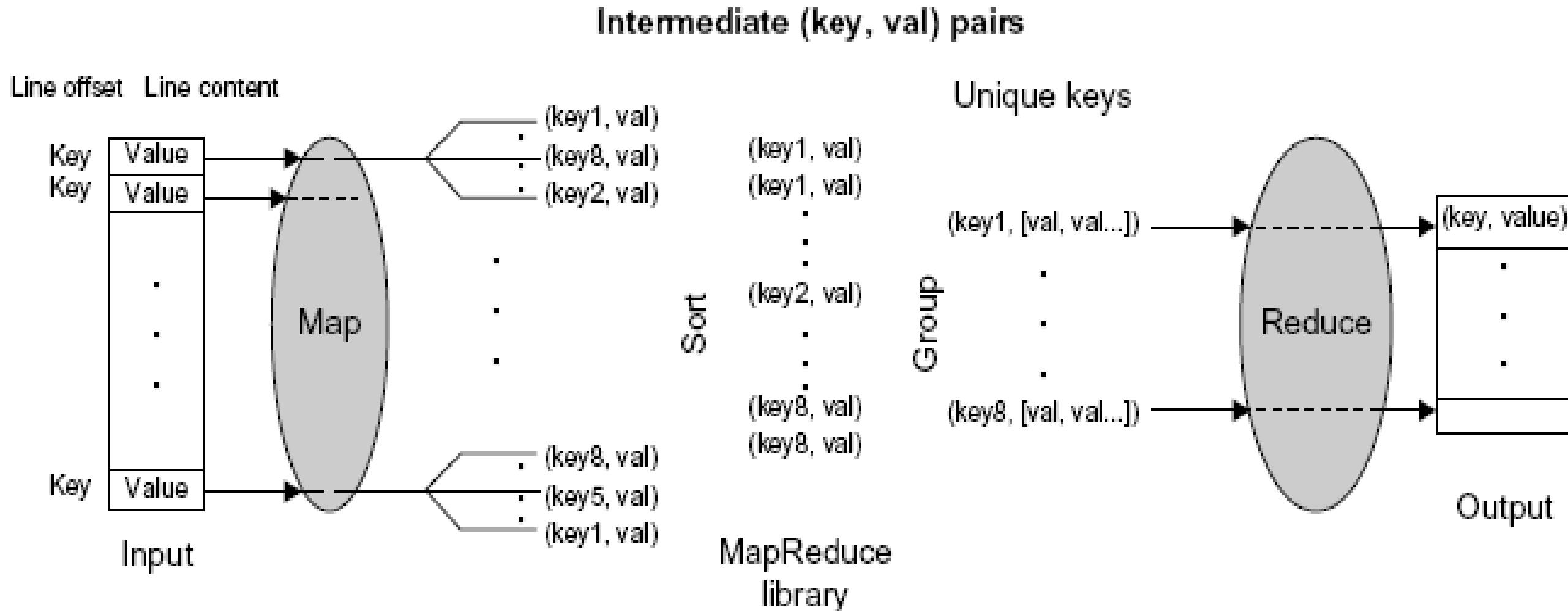
- Programmers specify two functions:
 - map** $(k, v) \rightarrow \langle k', v' \rangle^*$
 - reduce** $(k', v') \rightarrow \langle k', v' \rangle^*$
 - All values with the same key are reduced together
- The execution framework handles everything else...
- Not quite...usually, programmers also specify:
 - partition** $(k', \text{number of partitions}) \rightarrow \text{partition for } k'$
 - Divides up key space for parallel reduce operations
 - combine** $(k', v') \rightarrow \langle k', v' \rangle^*$
 - Mini-reducers that run in memory after the map phase
 - Used as an optimization to reduce network traffic



Logical Data Flow in 5 Processing Steps in MapReduce Process



UNIVERSITY OF LEEDS





Counting the number of occurrences of each word in a large collection of documents.

```
map(String key, String value):  
    // key: document name  
    // value: document contents  
    for each word w in value:  
        EmitIntermediate(w, "1");
```

The **map** function emits each word w plus an associated count of occurrences (just a “1” is recorded in this pseudo-code).



Pseudo Code : Reduce

UNIVERSITY OF LEEDS

Counting the number of occurrences of each word in a large collection of documents.

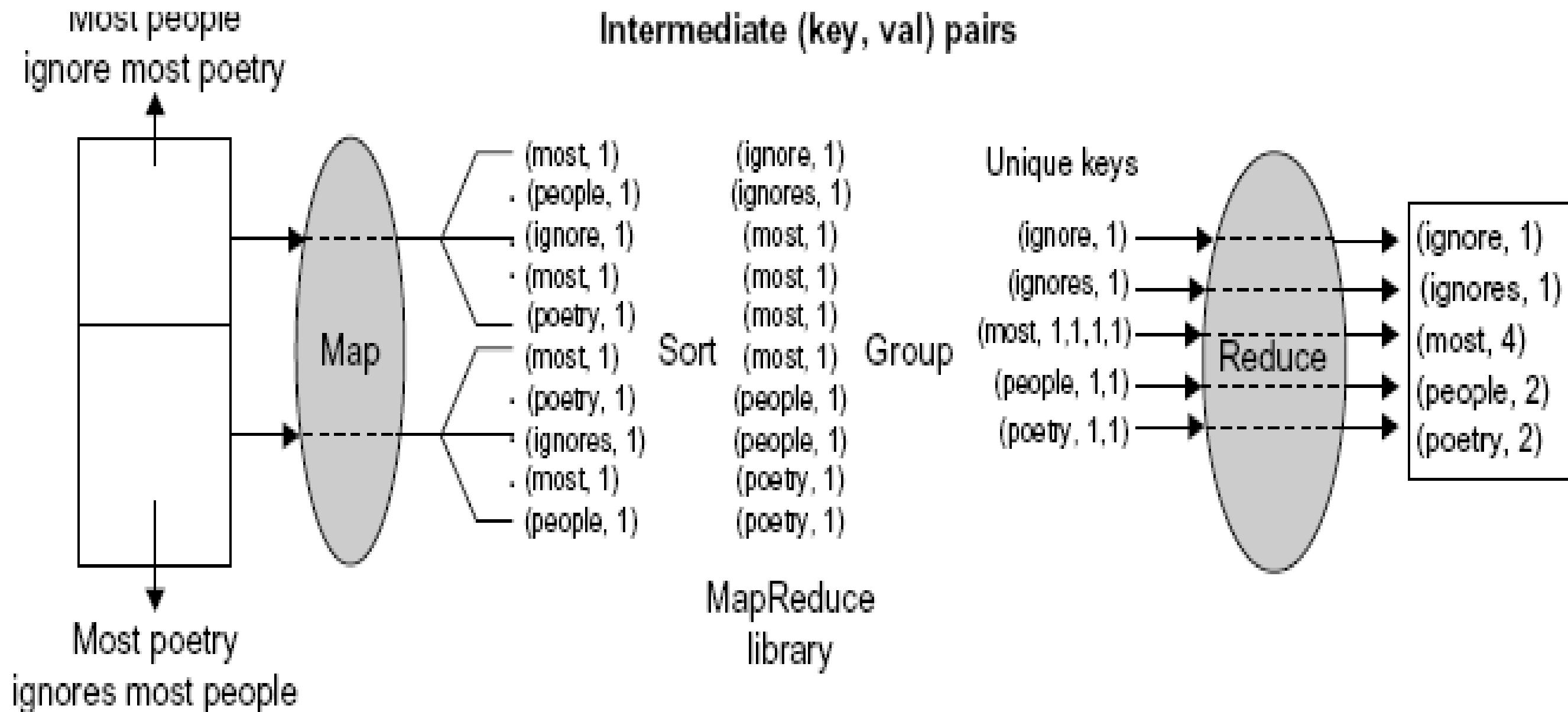
```
reduce (String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt (v);
    Emit (AsString(result));
```

The **reduce** function sums together all counts emitted for a particular word.

A Word Counting Example on <Key, Count> Distribution



UNIVERSITY OF LEEDS



- Google has a proprietary implementation in C++
 - Bindings in Java, Python
- Hadoop is an open-source implementation in Java
 - Development led by Yahoo, used in production
 - Now an Apache project
 - Rapidly expanding software
 - **Spark:** run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk
 - Other: **Pig, Hive, Storm**
- Lots of custom research implementations
 - For GPUs, cell processors, etc.



Distributed File System (DFS)

UNIVERSITY OF LEEDS

- Don't move data to workers... move workers to the data!
 - Store data on the local disks of nodes in the cluster
 - Start up the workers on the node that has the data local
- Why?
 - Not enough RAM to hold all the data in memory
 - Disk access is slow, but disk throughput is reasonable
- A distributed file system is the answer
 - GFS (Google File System) for Google's MapReduce
 - HDFS (Hadoop Distributed File System) for Hadoop

Google





The outputs of Map and inputs/outputs of Reduce **are always <key, value> pairs**

MapReduce is great for problems where the “sub problems” are NOT interdependent
(e.g. the output of one mapper should NOT depend on the output of another)

The reduce phase does not begin execution until ALL mappers have finished

MapReduce takes care of **scheduling tasks, monitoring them and re-executing failed tasks.**

All data elements in MapReduce are **immutable** – they cannot be updated.

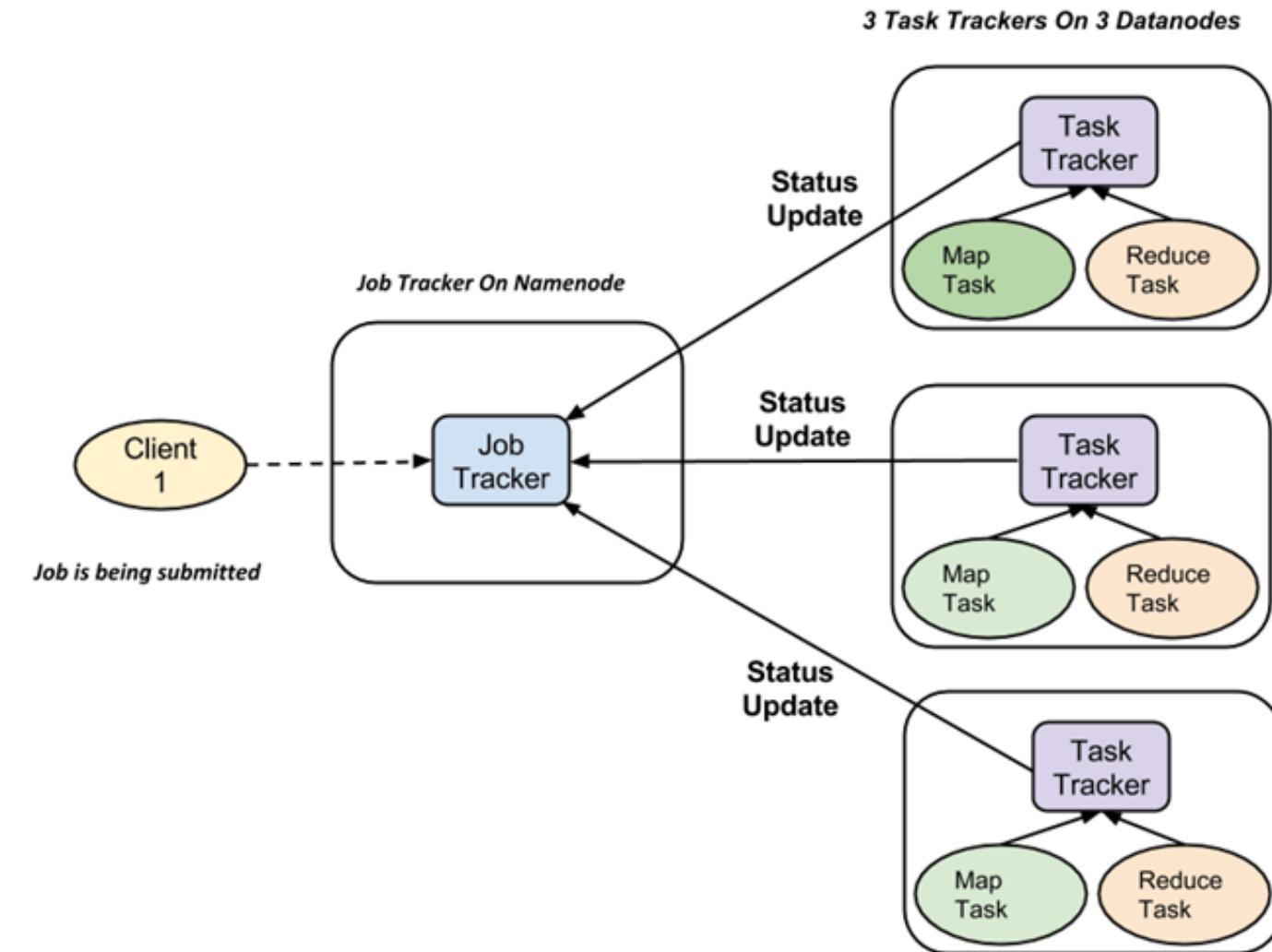
In Hadoop, MapReduce is Rack and HDFS aware



The Client submits the MapReduce job to the **Job Tracker**.

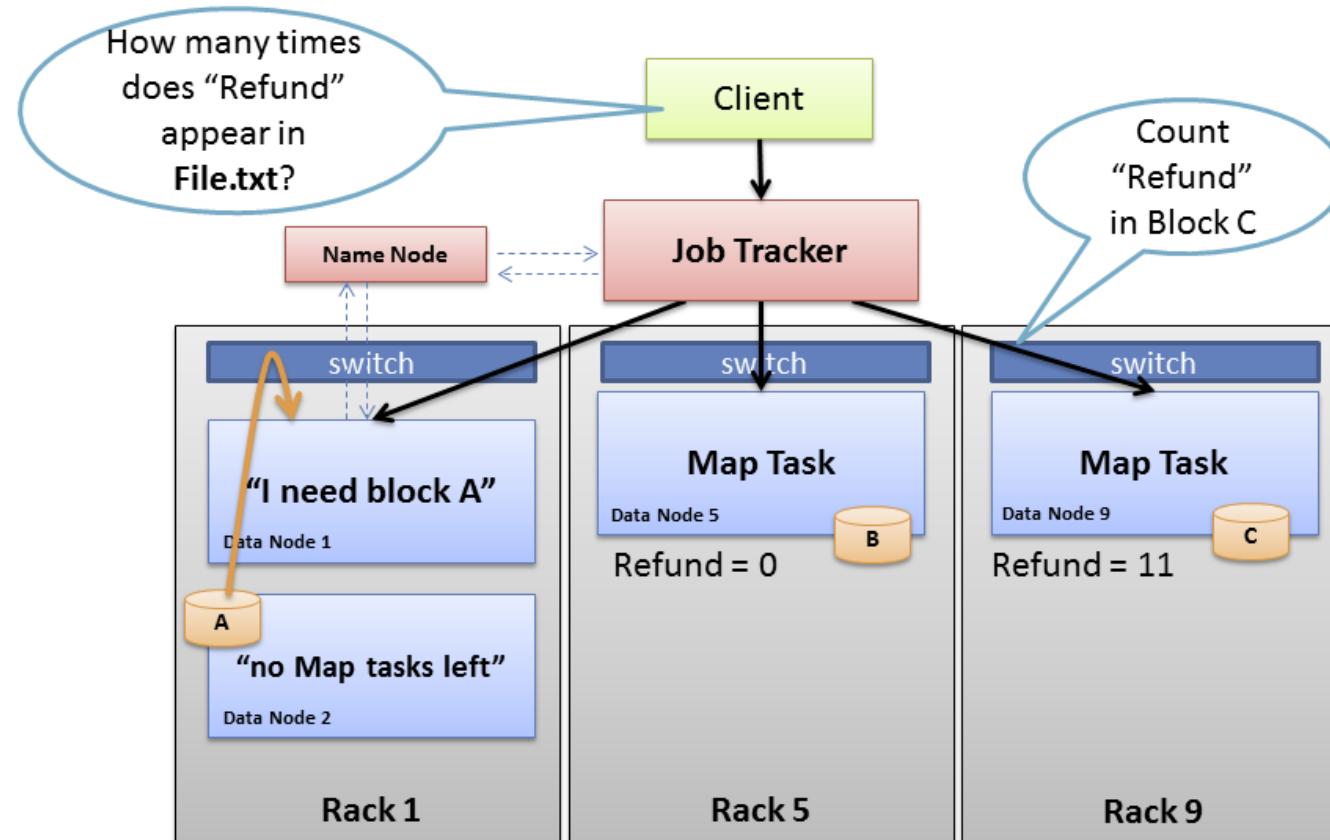
The **Job Tracker** consults the **NameNode** to learn which **DataNodes** have the file blocks.

The **Job Tracker** then provides the **Task Tracker** running on those nodes with the **Java code** required to execute the Map computation on their local data.



The **Task Tracker** starts a Map task and monitors its progress. It provides heartbeats and task status back to the **Job Tracker**.

As each Map task completes, its node stores the result in temporary storage (**intermediate data**). When all Maps complete, this is sent over the network to nodes running reduce tasks.



The **Job Tracker** will always try to pick nodes with local data for a Map task.

This might not be possible
(for example, the nodes with local data might be already running too many other tasks)

In this situation, the **Job Tracker** will try to assign the task to a node in the same rack as the data (using Rack Awareness) wherever possible.

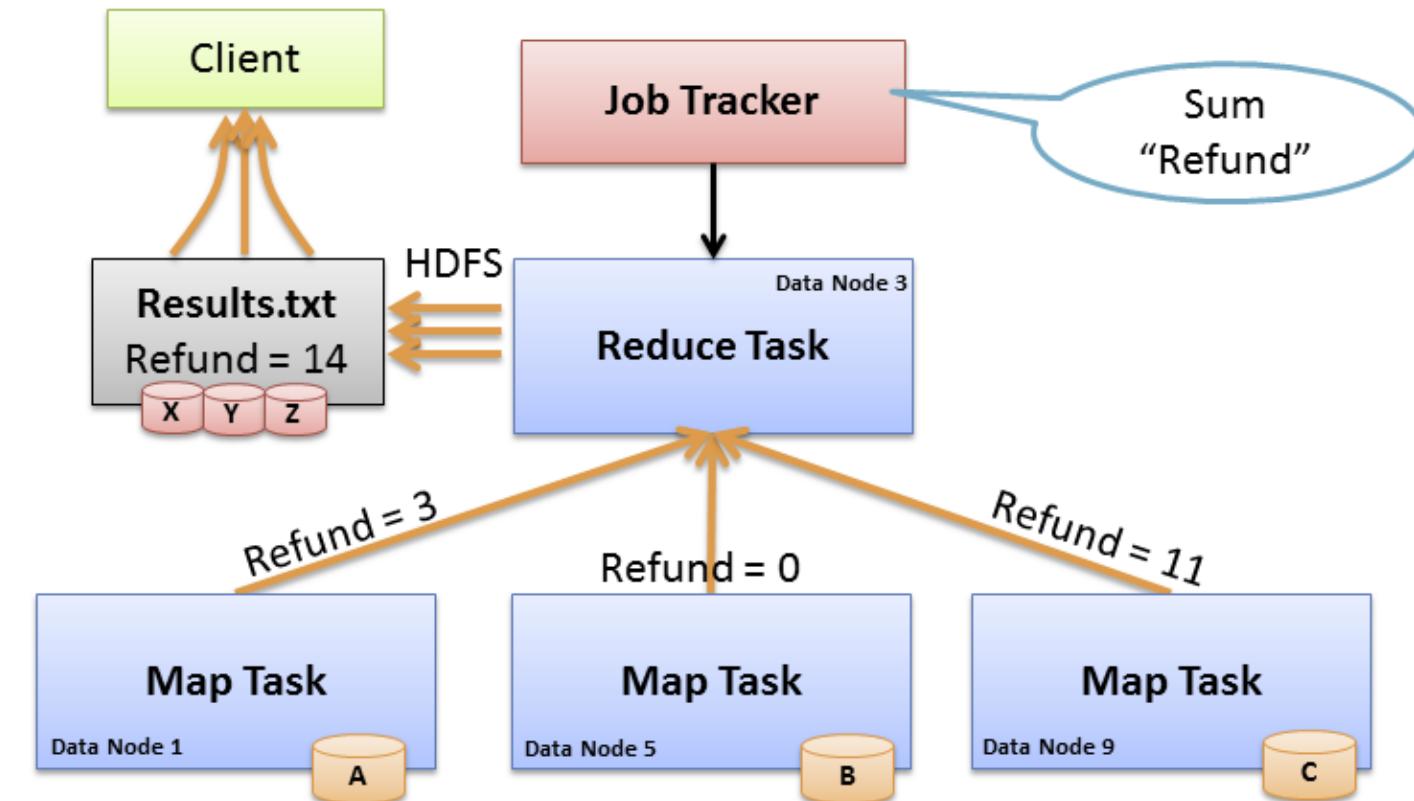
The **NameNode** will instruct the node to copy the data from the relevant **DataNode**.



The **Job Tracker** starts a Reduce task on any of the nodes in the cluster

It instructs the task to copy **intermediate data** from every completed Map task needed.

If Map tasks respond almost instantly and flood the Reduce task with TCP data, “**Incast**” occurs



To handle this, it is important that the cluster network switches have **good internal traffic management capabilities and buffers**

The outputs from the Reducer tasks are written to **HDFS** as per what we have covered (splitting into blocks, pipeline replication, etc.)

Hadoop supports methods to merge multiple Reduce Task jobs if required.



Facebook has a “friends in common” feature.

This list doesn’t change frequently, so to improve performance why don’t we use MapReduce to **calculate everyone’s friends in common once per day and store that result.**

When someone visits a page, this stored result can then be displayed.

Assume friends are stored as: **Person → [List of Friends]**

FINDING FRIENDS EXAMPLE (2)



UNIVERSITY OF LEEDS

Person \rightarrow [List of Friends]

```
A -> B C D  
B -> A C D E  
C -> A B D E  
D -> A B C E  
E -> B C D
```

facebook

Assume each line will be an input split to a mapper.

The mapper will output <Friend Person, List of Friends>
As an example, the first two lines will have the result:

```
(A B) -> B C D  
(A C) -> B C D  
(A D) -> B C D
```

```
(A B) -> A C D E  
(B C) -> A C D E  
(B D) -> A C D E  
(B E) -> A C D E
```

As an exercise, please now complete the mapping for all input splits



FINDING FRIENDS EXAMPLE (3)

UNIVERSITY OF LEEDS

Person \rightarrow [List of Friends]

A \rightarrow B C D

B \rightarrow A C D E

C \rightarrow A B D E

D \rightarrow A B C E

E \rightarrow B C D

Map and Sort

We are assuming that the list on the right has already been **sorted** by the MapReduce framework, ready for **partitioning** and **shuffling**.

(A B) \rightarrow (A C D E)

(A B) \rightarrow (B C D)

(A C) \rightarrow (A B D E)

(A C) \rightarrow (B C D)

(A D) \rightarrow (A B C E)

(A D) \rightarrow (B C D)

(B C) \rightarrow (A B D E)

(B C) \rightarrow (A C D E)

(B D) \rightarrow (A B C E)

(B D) \rightarrow (A C D E)

(B E) \rightarrow (A C D E)

(B E) \rightarrow (B C D)

(C D) \rightarrow (A B C E)

(C D) \rightarrow (A B D E)

(C E) \rightarrow (A B D E)

(C E) \rightarrow (B C D)

(D E) \rightarrow (A B C E)

(D E) \rightarrow (B C D)

The reducer function will intersect lists of values and output the same key with this result.

Now try this yourself.

FINDING FRIENDS EXAMPLE (4)



UNIVERSITY OF LEEDS

(A B) -> (A C D E)
(A B) -> (B C D)
(A C) -> (A B D E)
(A C) -> (B C D)
(A D) -> (A B C E)
(A D) -> (B C D)
(B C) -> (A B D E)
(B C) -> (A C D E)
(B D) -> (A B C E)
(B D) -> (A C D E)
(B E) -> (A C D E)
(B E) -> (B C D)
(C D) -> (A B C E)
(C D) -> (A B D E)
(C E) -> (A B D E)
(C E) -> (B C D)
(D E) -> (A B C E)
(D E) -> (B C D)

Partition,
Shuffle,
Reduce



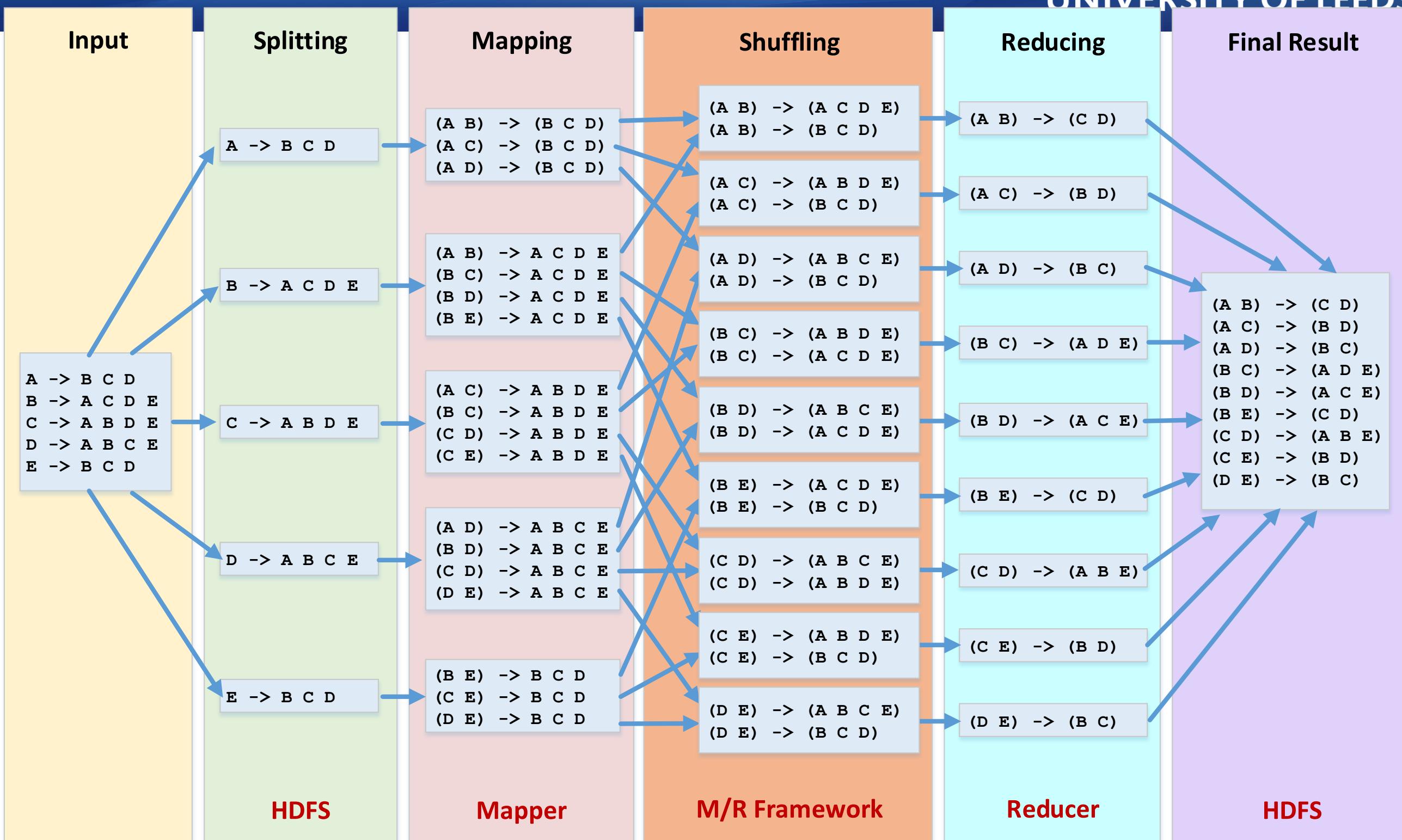
(A B) -> (C D)
(A C) -> (B D)
(A D) -> (B C)
(B C) -> (A D E)
(B D) -> (A C E)
(B E) -> (C D)
(C D) -> (A B E)
(C E) -> (B D)
(D E) -> (B C)

Now - for example - when A visits B's profile, we can quickly look up (A B) and see that they have two friends in common, (C D)

FINDING FRIENDS EXAMPLE (5)



UNIVERSITY OF LEEDS



Imagine Facebook has 1 billion users and this operation took 10 hours on a single machine.
If we used MapReduce to process this with 10000 nodes, it would take around **4 seconds**.



Summary

- What is MapReduce
- Importance of the underlying distributed file system
- MapReduce illustrated through an example

Prescribed Reading

- [Hadoop: The Definitive Guide](#), 4rd Edition. Tom White. O'Reilly Media, 2015 (chapters 2 and 7)
- IBM Hadoop Developers. <https://developer.ibm.com/hadoop/>



COMP5123M Cloud computing Systems

EVOLUTION OF HADOOP AND MAPREDUCE – INTRODUCTION TO SPARK

Goals:

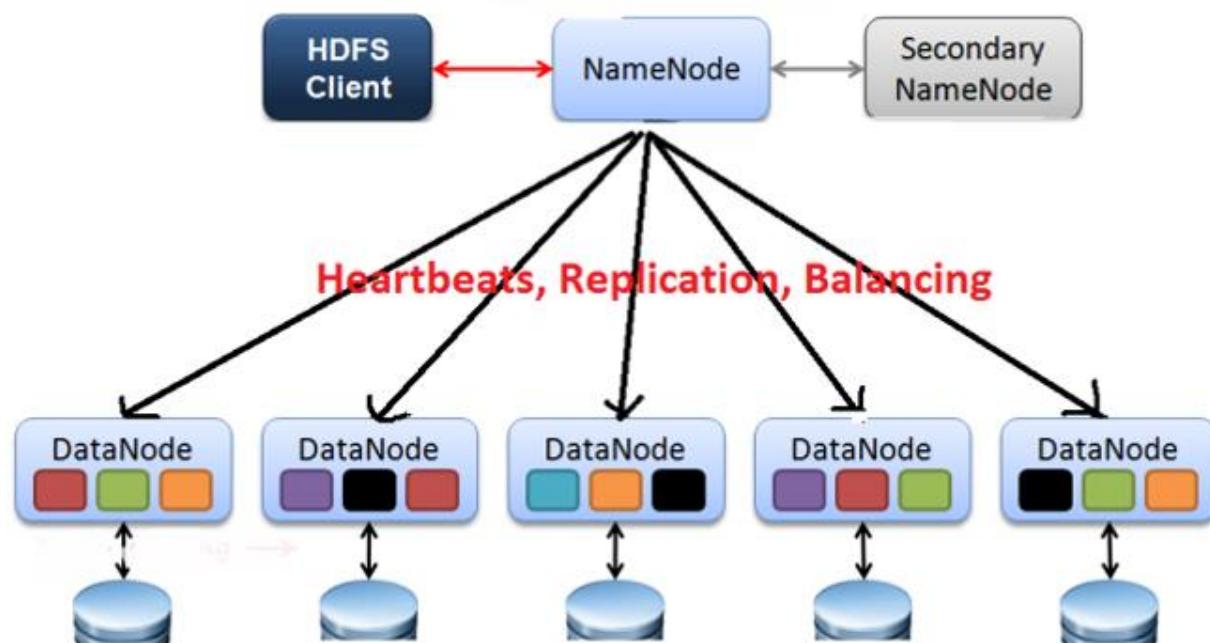
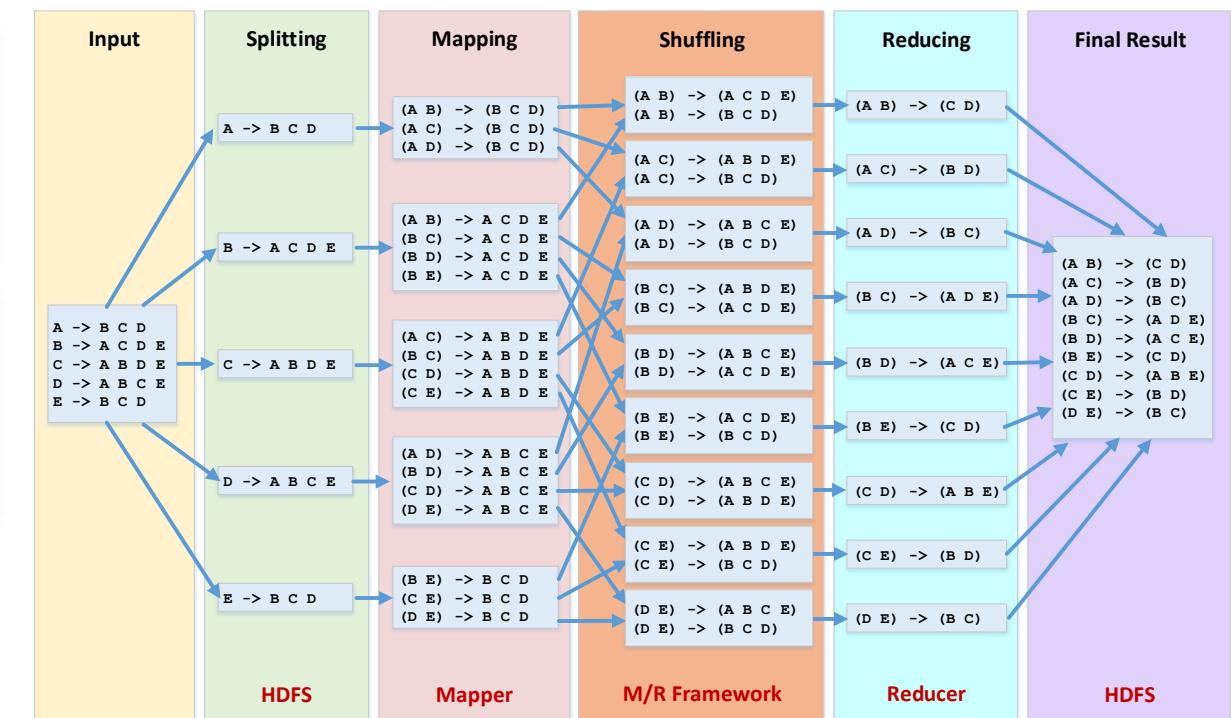
- Understand evolution of original Hadoop

Plan:

- Original Hadoop limitations
- HDFS Federation
- YARN
- Spark
- Summary

Map Reduce Distributed Programming Framework

HDFS Hadoop Distributed File System



Point of Interest
Hadoop 3.1.4 released
on August 3rd 2020



The two major limitations of MapReduce are

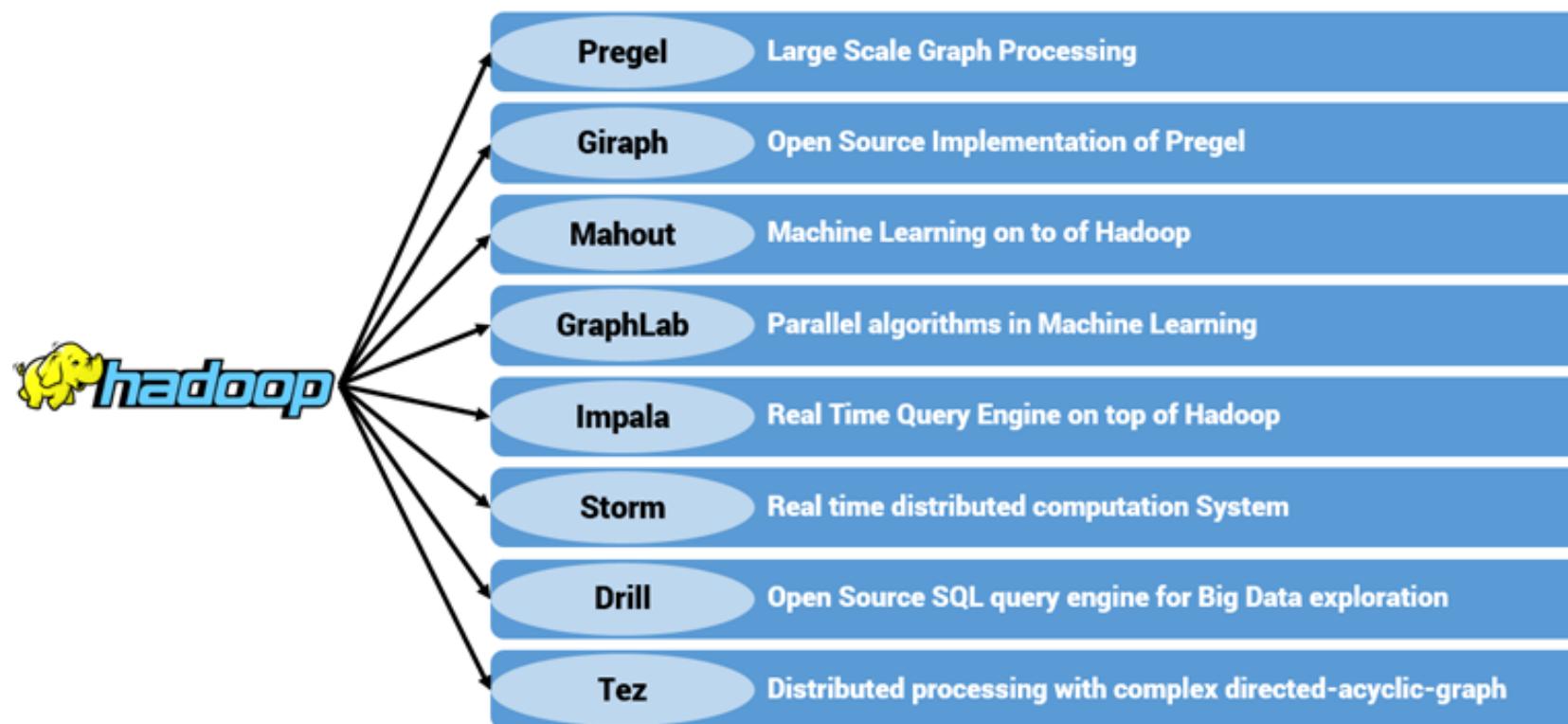
Difficulty

Many people struggle with programming pure MapReduce tasks.

Performance

There are often bottlenecks in performance (and batch processing doesn't fit the required use case)

The result of this is that MapReduce does not compose well for large applications. This has resulted in the development of multiple specialised systems.

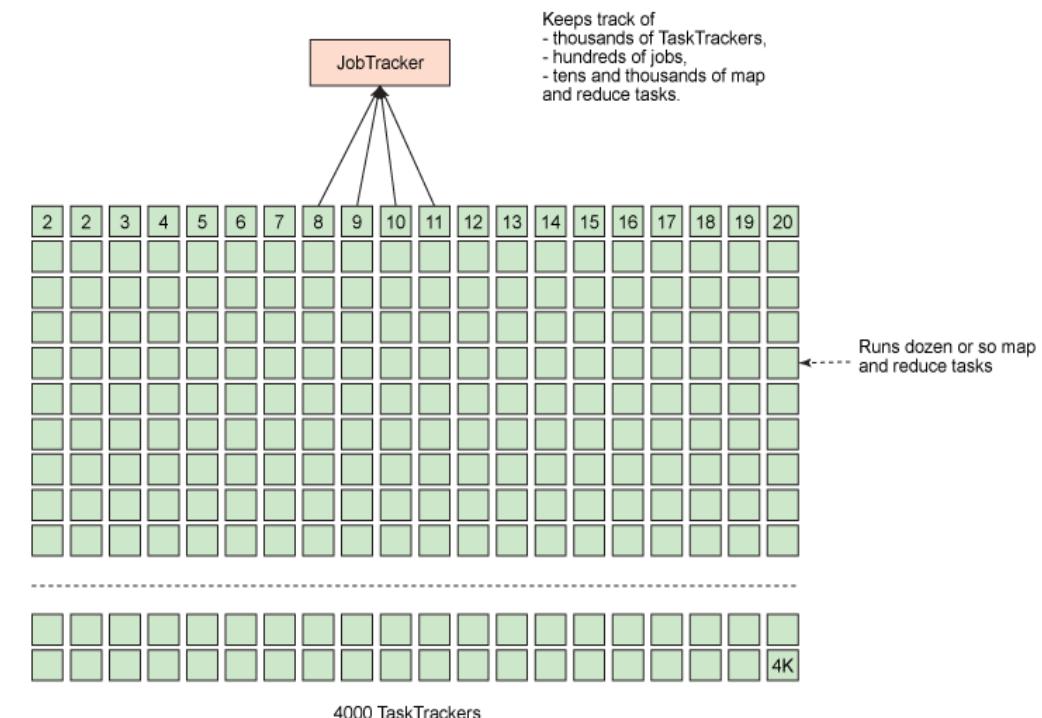


LIMITATIONS OF THE ORIGINAL HADOOP

In original Hadoop, there is tight coupling between Cluster Resource Management and MapReduce.

JobTracker runs on a single machine.
Limits scalability - could be too many DataNodes

JobTracker is a single point of availability.
If the JobTracker fails, all jobs must restart.

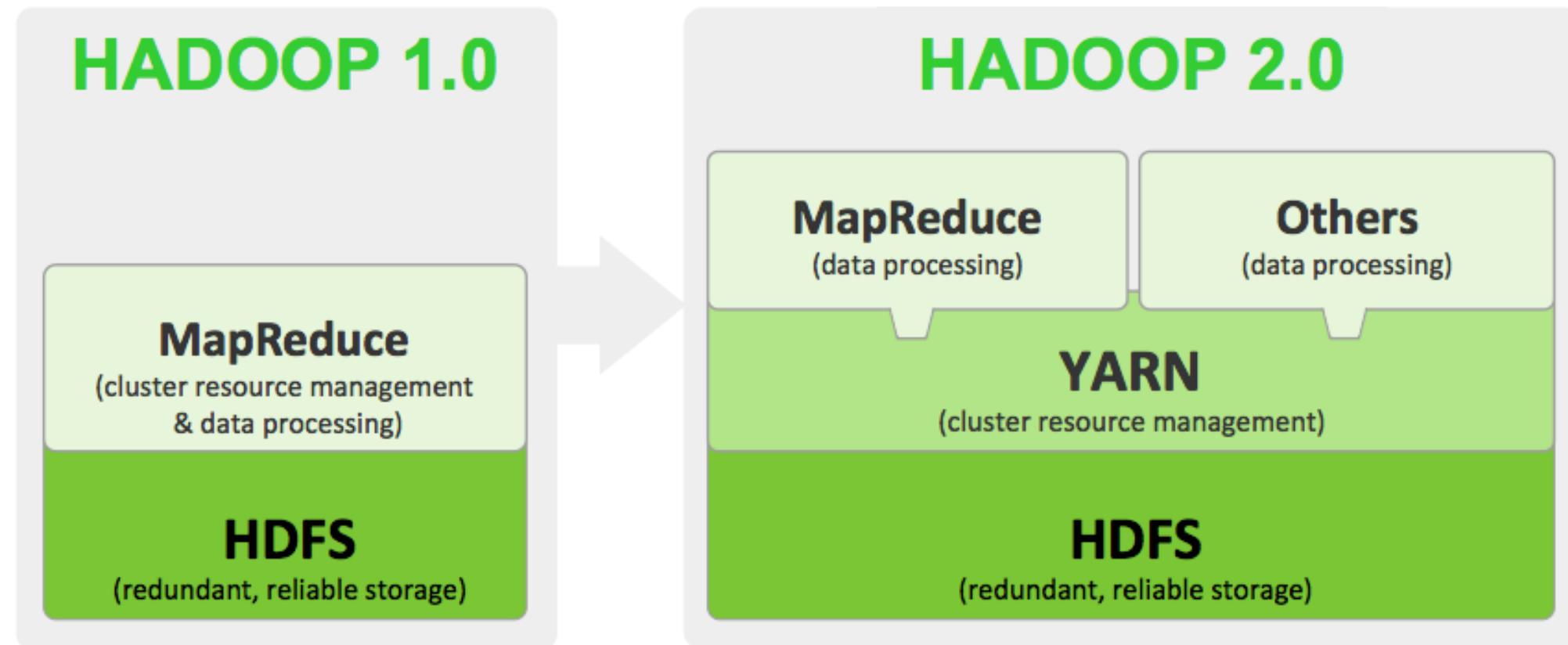


Yahoo estimated the limits of this design to be 5000 nodes and 40,000 concurrent tasks

There are also problems with resource utilization

Hadoop had a pre-defined number of map and reduce slots.
Map slots might be 'full' while Reduce slots are 'empty' (and vice versa)

The number of files is also – surprisingly – limited. **NameNodes** hold all metadata in main memory, so are typically limited to 50-100 million files per cluster.



Hadoop 2 moves from a restricted batch-oriented model
to **more interactive and specialized processing models**

The biggest changes in Hadoop 2 are
HDFS Federation, **YARN**, a highly available **NameNode**, and the concept of **Containers**

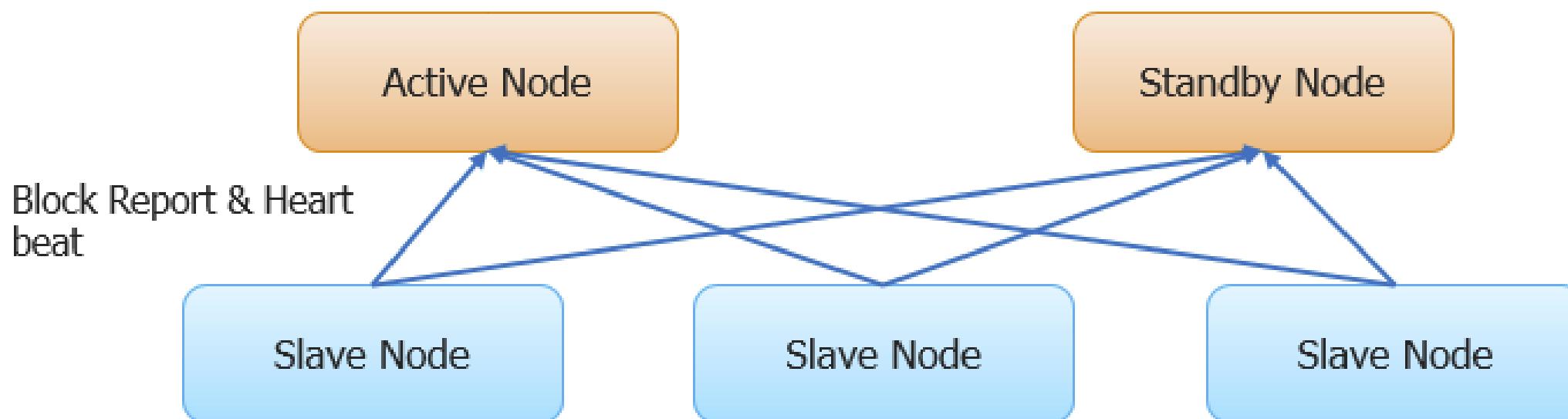


In Hadoop 1, the **NameNode** was a single point of failure in an HDFS cluster.

Each cluster has a single **NameNode**, and if that machine or process becomes unavailable, the cluster as a whole becomes unavailable until another **NameNode** is created or restarted.

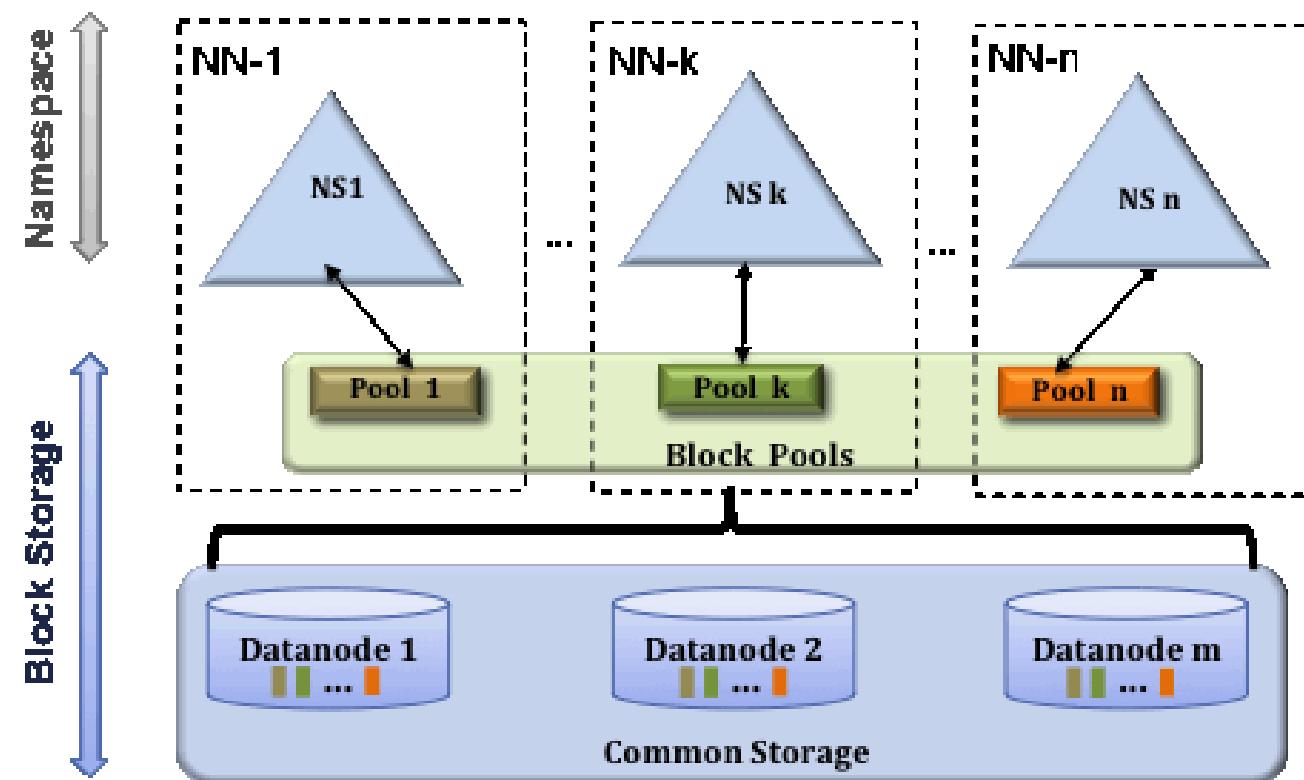
Hadoop 2 has a **HDFS High Availability** feature to address this issue.

It provides the option of running **two redundant NameNodes** in the same cluster in an Active/Passive configuration.



Even with high availability, two NameNodes alone do not provide horizontal scalability

A solution to this issue is to federate multiple independent NameNodes



Federated NameNodes are independent and do not require coordination.

DataNodes are used as common storage for blocks by all NameNodes.

Each DataNode registers with all NameNodes in a cluster.

DataNodes send heartbeats, block reports, and handle commands, from all the NameNodes.

The fundamental idea of YARN is to
Split Hadoop resource management and job scheduling into separate processes (daemons).

Many different types of application can be submitted to YARN (MapReduce, Giraph, etc.)
An application is either a single job or a Directed Acyclic Graph (DAG) of jobs.

ResourceManager is the authority that arbitrates resources among all applications. Replaces the **JobTracker**.

NodeManager is a per-machine framework responsible for **containers**, monitoring resource usage, and reporting to the **ResourceManager**. Each machine in a cluster is a NodeManager and a DataNode.

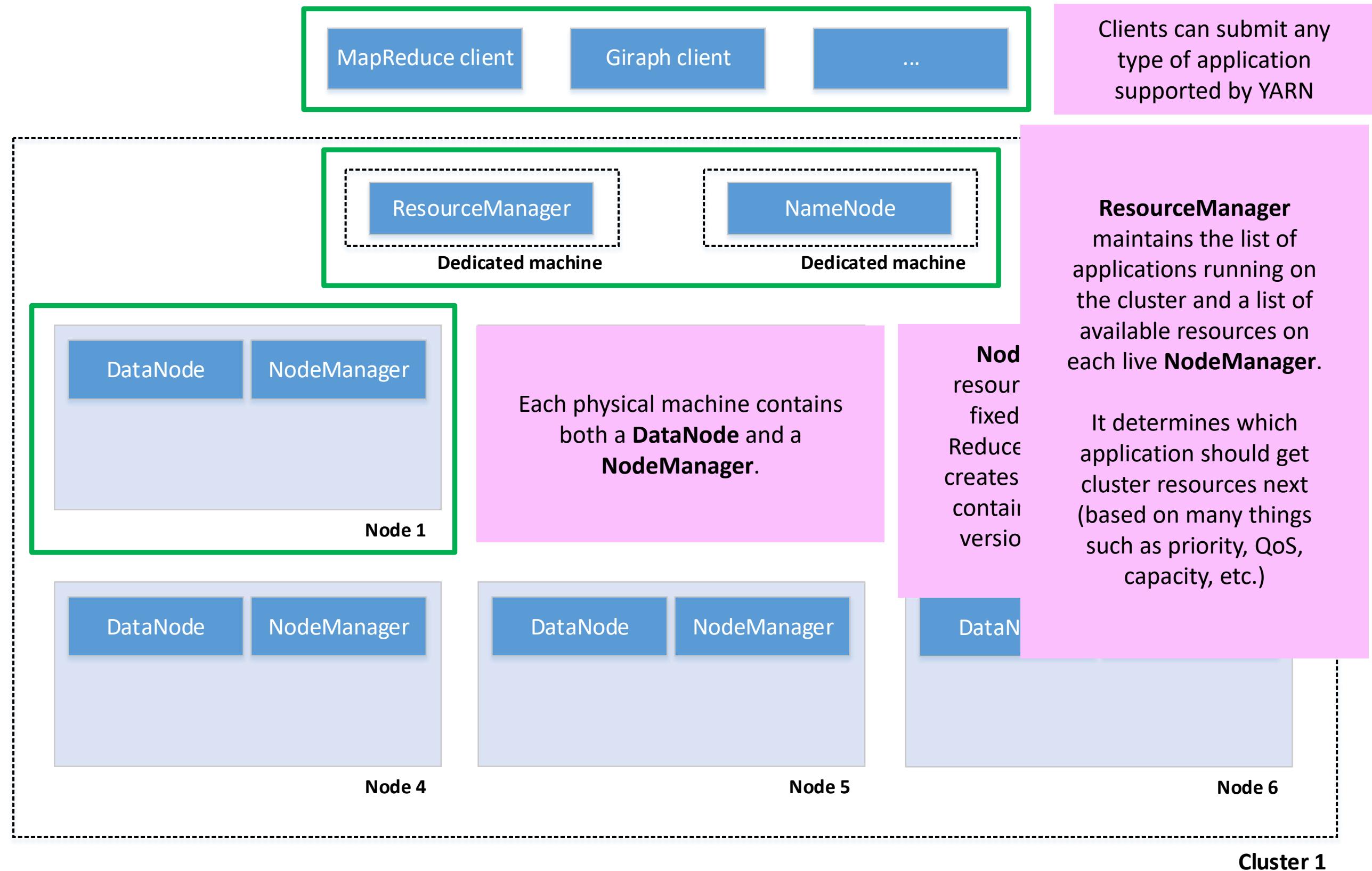
ApplicationMaster is tasked with negotiating resources from the **ResourceManager** and working with **NodeManagers** to execute & monitor tasks.

This allows more jobs to be run in parallel, and scalability is dramatically increased

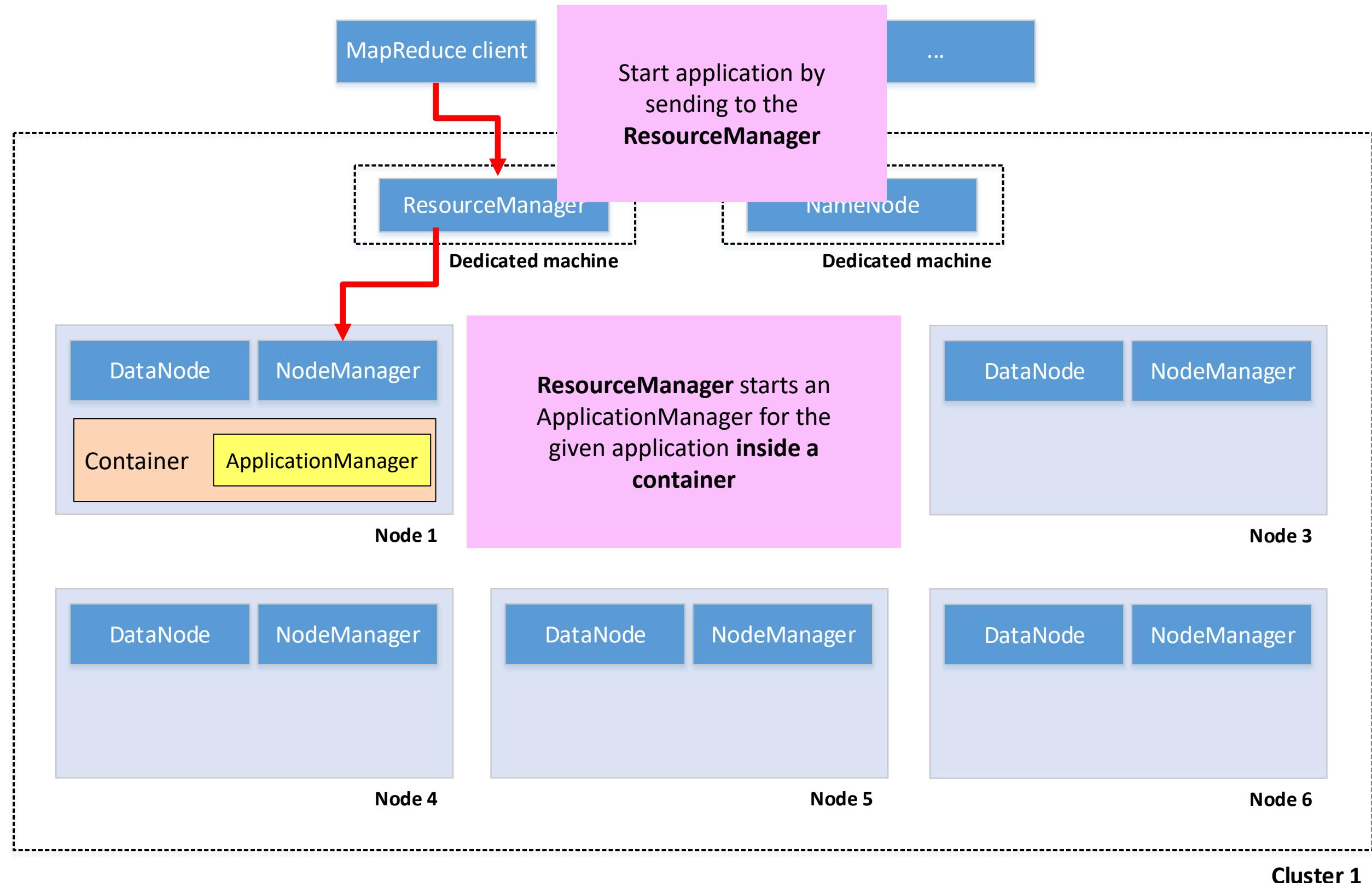
APPLICATION SUBMISSION IN YARN



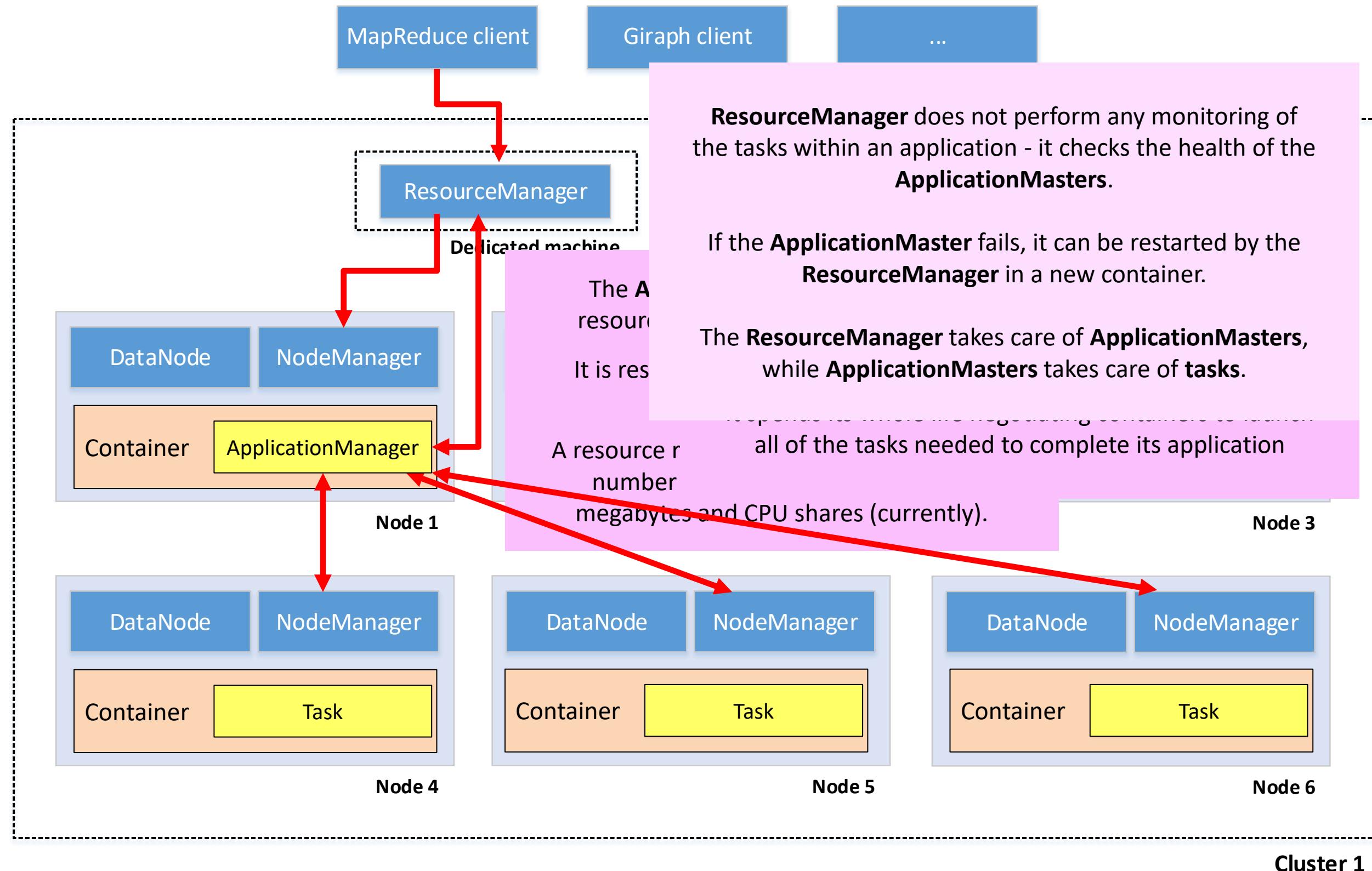
UNIVERSITY OF LEEDS



APPLICATION SUBMISSION IN YARN



APPLICATION SUBMISSION IN YARN



Apache Spark is a general-purpose data processing engine.

It combines the specialities of the stream processing
into a single component with a common set of APIs.

Has been designed for a range of requirements

Faster **batch** processing

Apps requiring **interactive** query processing

Processing of **streaming** data

Systems that require **iterative** algorithms

Features of Spark

In Memory computation engine

Almost 100x faster than Hadoop MapReduce with in-memory computations

Almost 10x faster than Hadoop MapReduce using computations with Disk IO

Apache Spark doesn't provide any storage (like HDFS) or Resource Management capabilities.

It is just a unified framework for processing large amount of data near to real time.

The Spark framework is organized into three major layers.

Core Layer

The generalized layer of the framework. It defines all basic functions. All other functionalities and extensions are built on top of this.

Ecosystems Layer

Contains libraries operating on top of the Spark Core.

Resource Management layer

Spark manages its own resource in standalone mode (a single node cluster setup). But for distributed cluster mode it can be integrated with resource management modules like **YARN**.



Resource Management

Standalone

YARN

Mesos

Spark Ecosystem

Spark SQL

Spark Streaming

BlinkDB

Spark ML

GraphX

Tachyon

Spark Core

Spark DataFrame API

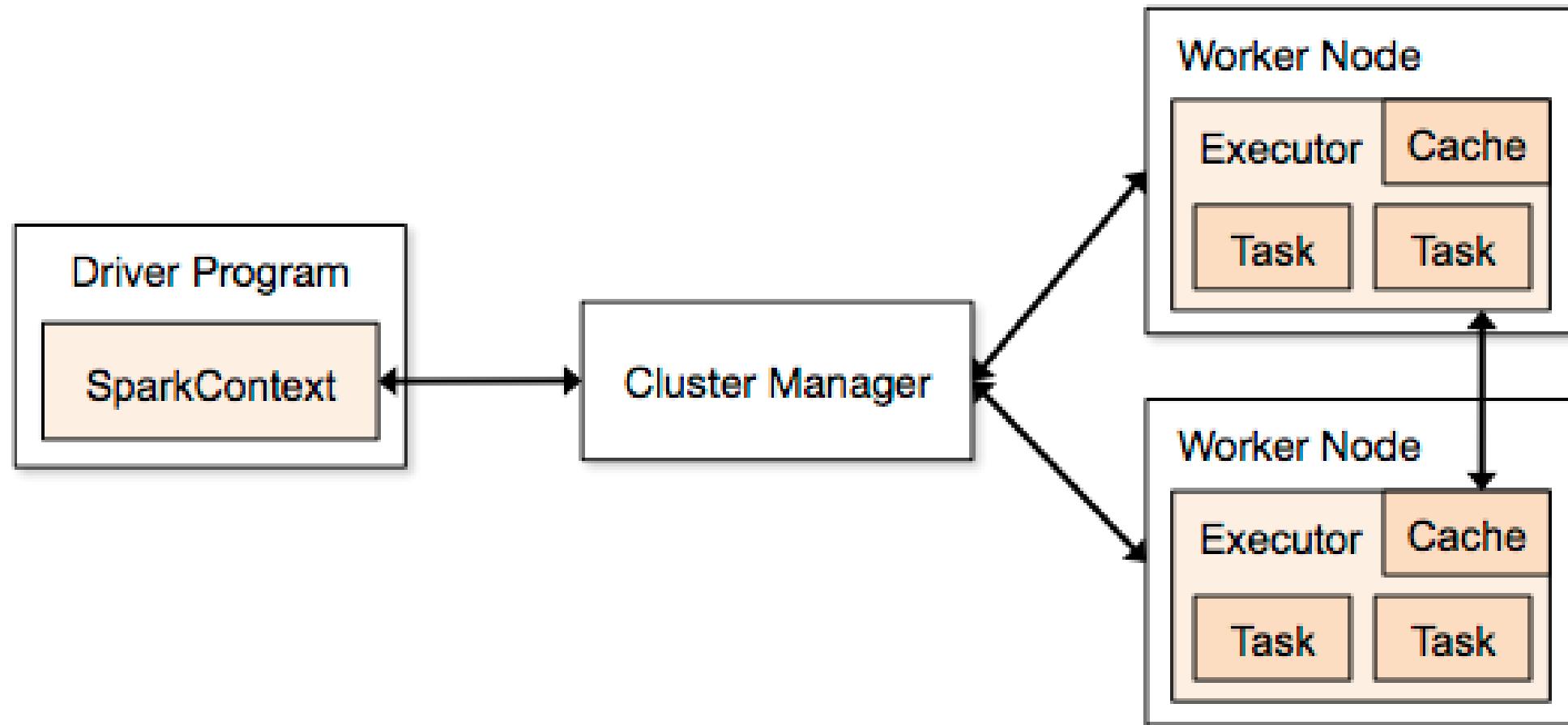
Java

Scala

Python

R

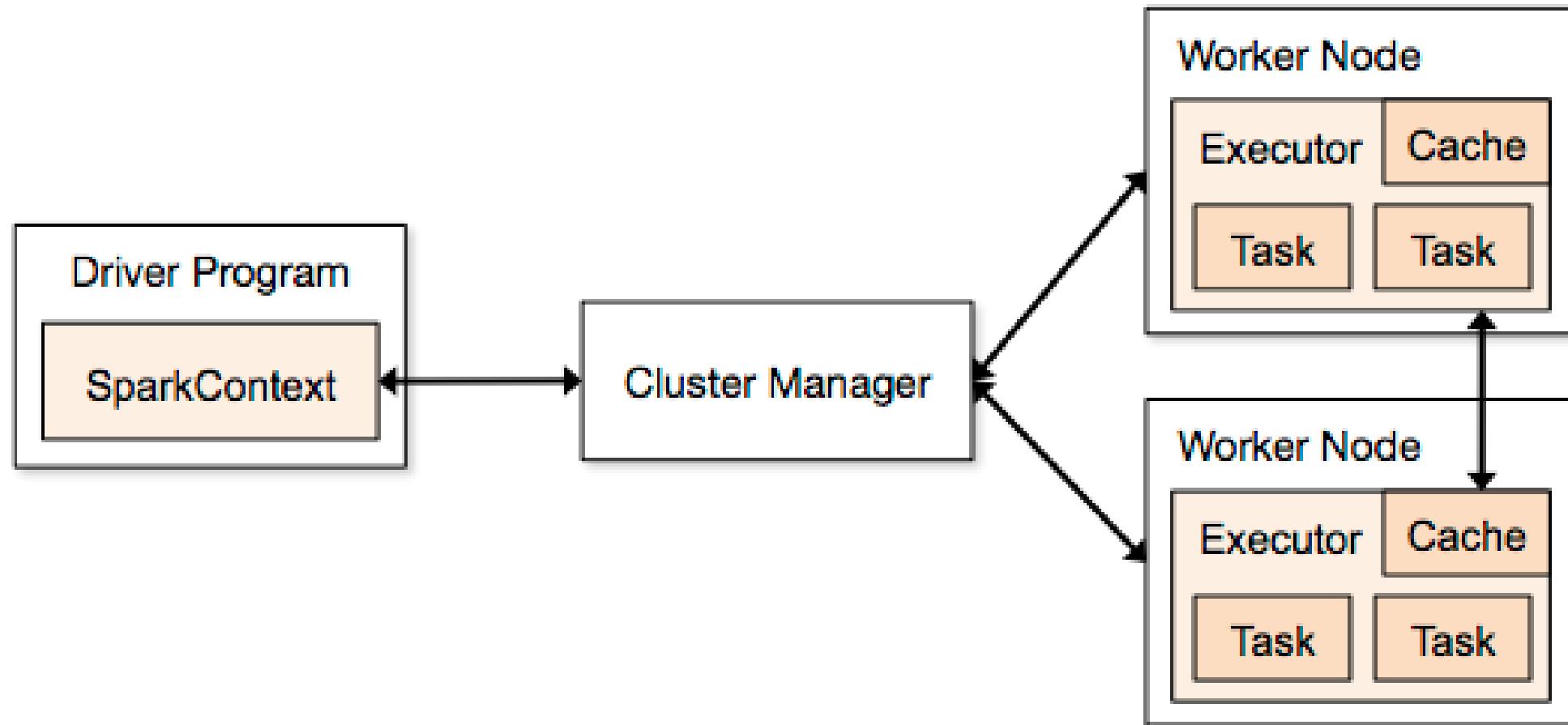
Spark Core



Spark applications run as independent sets of processes on a cluster, coordinated by the **SparkContext** object (aka Driver Program)

SparkContext sends application code (JAR or Python files) and tasks to run to the Executors.

Each driver program has a web UI, typically on port 4040, that displays information about running tasks, executors, and storage usage



Each application gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads.

This has the benefit of isolating applications from each other - each driver schedules its own tasks, and tasks from different applications run in different JVMs.

However, it also means that data cannot be shared across different Spark applications (instances of SparkContext) without writing it to an external storage system.

Spark provides high-level APIs in **Java**, **Scala**, and **Python**

It provides an optimised engine that supports general execution graphs, and high-level tools for structured data processing, etc.

Another important aspect is the **interactive shell** (REPL). Using REPL, you can test the outcome of each line of code without first needing to code and execute the entire job.

Spark API

Scala

```
val spark = new SparkContext()
val lines    = spark.textFile("hdfs://docs/")    // RDD[String]
val nonEmpty = lines.filter(l => l.nonEmpty())   // RDD[String]
val count = nonEmpty.count
```

Java 8

```
SparkContext spark = new SparkContext();
JavaRDD<String> lines    = spark.textFile("hdfs://docs/")
JavaRDD<String> nonEmpty = lines.filter(l -> l.length() > 0);
long count = nonEmpty.count();
```

Python

```
spark = SparkContext()
lines = spark.textFile("hdfs://docs/")
nonEmpty = lines.filter(lambda line: len(line) > 0)
count = nonEmpty.count()
```

The **Spark Core** is the heart of spark. It deals with:

memory management and fault recovery

scheduling, distributing and monitoring jobs on a cluster

interacting with storage systems

It also implements the key concept of **Resilient Distributed Databases (RDDs)**

RDDs are **immutable fault tolerant distributed collections of objects**
that can be operated on in parallel.

An **RDD** can contain any type of object and is created by loading an external dataset or
distributing a collection from the driver program

An **RDD** is a representation of a dataset that is distributed throughout the cluster.



Summary

- Reviewed Hadoop V1 limitations
- Explained the concept of HDFS Federation
- Introduced YARN in Hadoop 2 to improve MapReduce implementation
- Introduced Spark.

References

Hadoop the Definitive Guide, Chapter 4. Tom White,
O'Reilly, 4th Edition. 2015

COMP5850 – Cloud Computing



Energy Efficiency in Clouds

Plan of the Lecture

Goals

Understand concepts surrounding energy efficiency in clouds

Overview

- Introduction
- Powering the Cloud Infrastructure: Energy Consumption, Costs, Implications
- Power-Aware Computing: Trends and Issues
 - The role of hardware
 - The role of software
- Towards energy efficient Clouds
- Review of energy efficiency metrics
- Conclusion

Ordinary Things Ordinary People Do ...

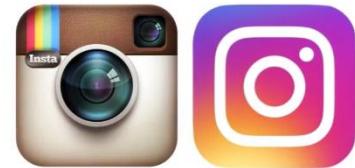
- “Liking” something on Facebook



- Streaming the latest movie

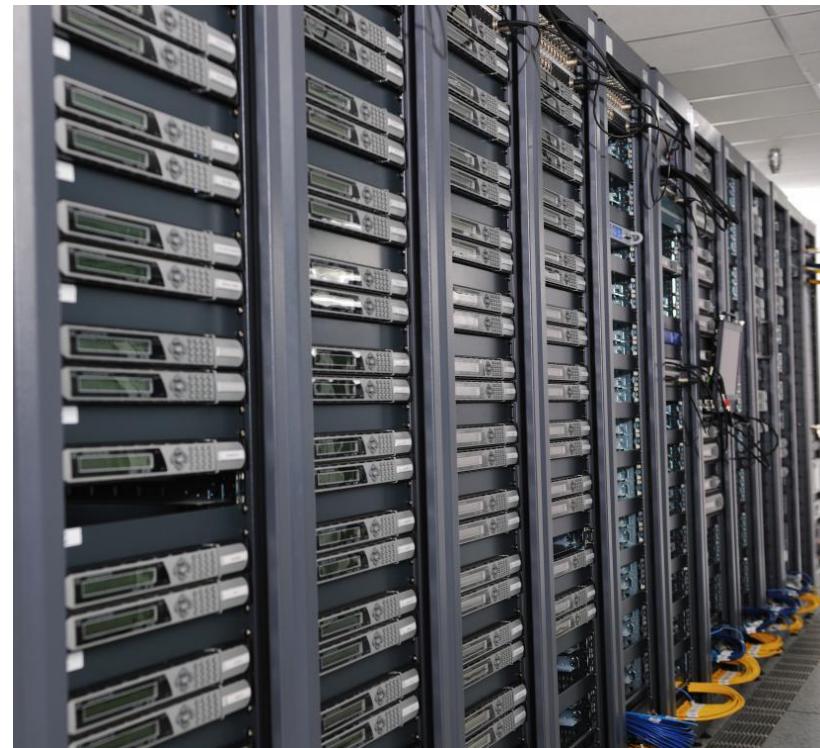


- Posting on instagram while on holiday
- ... every internet activity involves huge amounts of data that needs to be processed/stored somewhere
- The “internet of everything” brings further innovations
 - Example: driverless cars
- The vast network of cloud data centres that have sprung up in the past decade will spread.

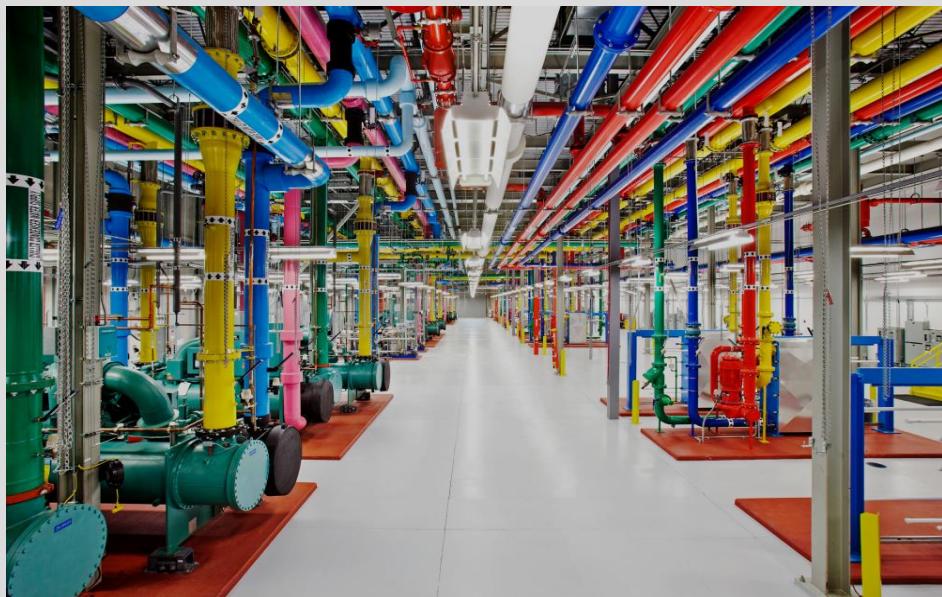
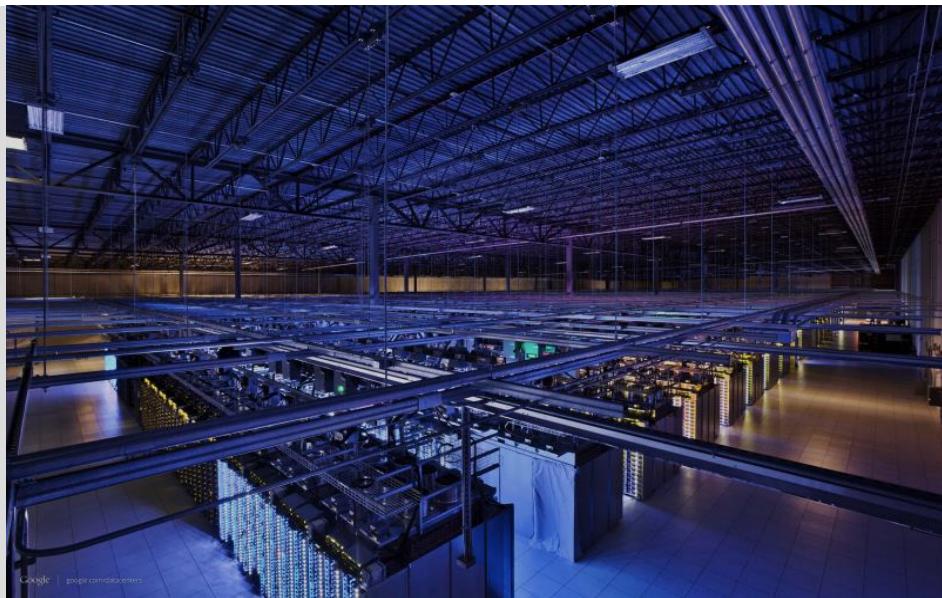


Powering the Cloud Infrastructure

- Modern data centres, operating under the Cloud computing model, are hosting a variety of applications, e.g.
 - those that run for **a few seconds** (e.g. serving requests of Web applications such as e-commerce and social networks portals)
 - those that run for **longer periods of time** (e.g. simulations or large dataset processing).
- However, Cloud Data Centers (DCs) consume **excessive** amount of energy:
 - The total energy bill for data centres is over \$15 billion



Data Centre Infrastructure and Growth



Think Environment?



GB

baby shark song

Time magazine reported that it costs 0.0002kWh of energy to stream 1 minute of video from the Youtube data centre.

0.01kWh of energy is consumed on average in downloading 1MB over the Internet.

The average Internet device energy consumption is around 0.001kWh in 1 minute of video stream

Using the number of downloads, transferring this 17MB file and streaming for 4 minutes gives the full energy cost in streaming this one video since June 2016!

2097.6 GWh = annual electricity consumption of Togo
(1,239 GWh, pop. 8.65M) and Lesotho **(791 GWh, pop. 2.3M)** in 2018

The Dark side ...

- Data centres have
 - Consume 3% of the global electricity supply
 - Contribute 2% of world's total CO2 emissions



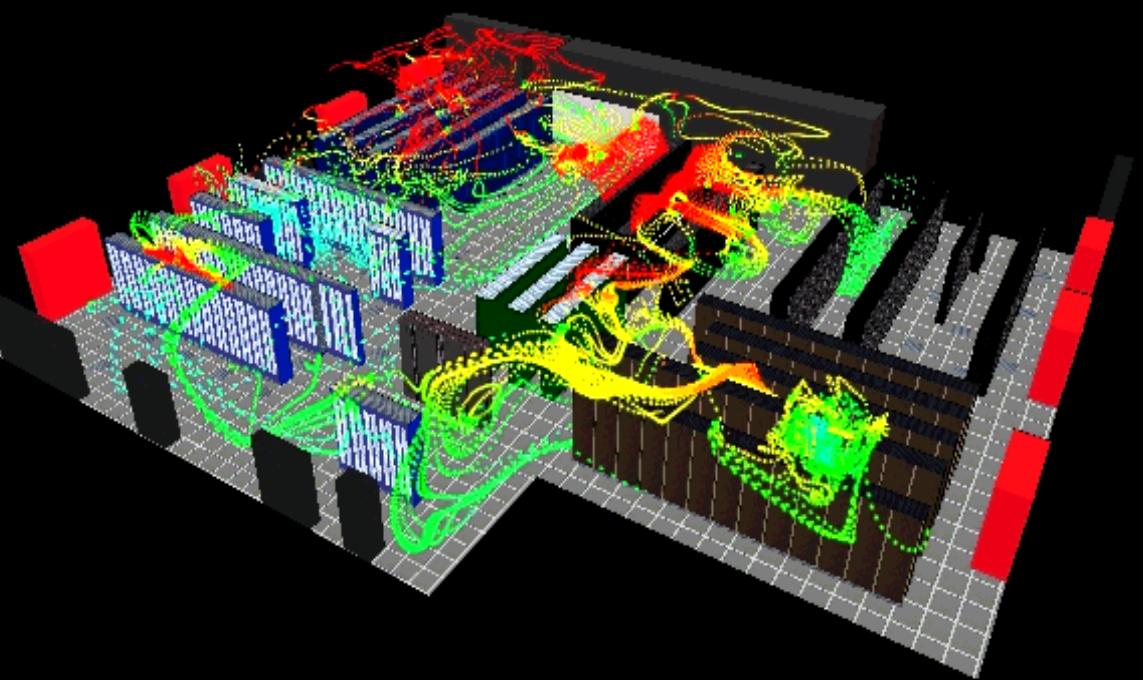
- 416.2 Terawatt hours of electricity the **world's data centres** used in **2017** was significantly **higher** than the **UK's total consumption** of about 300 Terawatt hours.
- Amount of energy used by data centres likely to **double** every 5 years

Some Facts

- It is estimated that the IT sector will be the world's most energy consuming industry
 - **% of global CO2 emissions** likely to increase
- **Apple's \$1bn "iDataCenter"** in North Carolina is estimated to require as much energy as 250,000 EU homes
<http://www.datacenterknowledge.com/the-apple-data-center-faq/>
- **Facebook** has built a major data centre in the far north of Sweden (Lulea)
 - 70 miles from the Arctic Circle
 - 84-acre site houses tens of thousands of computer servers
 - Still require 500 huge fans to cool them.
- Global internet traffic surged by almost 40% between **February and mid-April 2020**, driven by growth in video streaming, video conferencing, online gaming, and social networking ☹

Where Does the Power Go in a Data Center?

Power Consumption in the Datacenter



Server/Storage	50%
Computer Rm. AC	34%
Conversion	7%
Network	7%
Lighting	2%

Compute resources and particularly servers are at the heart of a complex, evolving system!

Power, Energy, and Energy Efficiency

- **Energy**: physical currency used to accomplish a particular task
 - Can be of various forms, e.g. Electrical, mechanical
- **Power**: the instantaneous rate of energy use, or equivalent
- Energy and power relationship:
$$\text{Energy} = \text{avgPower} \times \text{Time}$$
- **Energy Efficiency (EE)**: ratio of work done per unit of energy consumed
 - In computing energy is invariably delivered as electricity
 - Typical units
 - **Watts-Hour** for Energy
 - **Watts** for Power

$$\text{EE} = \text{Work Done} / (\text{Power} \times \text{Time})$$

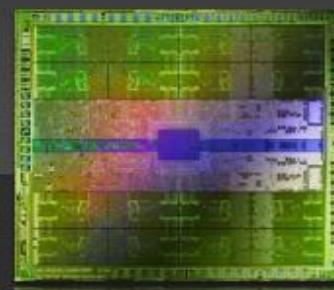
$$\text{Work Done} / \text{Energy}$$

$$\text{Performance} / \text{Power}$$

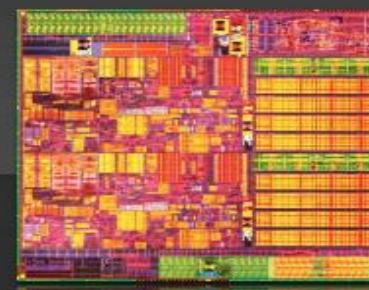
Power is THE Problem

- 1 Data Movement Dominates Power
- 2 Optimize the Storage Hierarchy
- 3 Tailor Memory to the Application

GPU
200pJ/Instruction



CPU
2nJ/Instruction



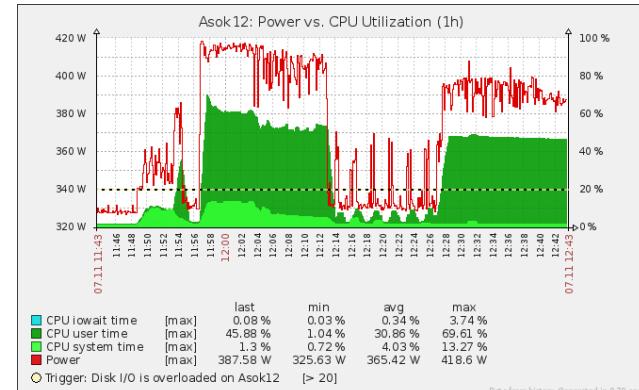
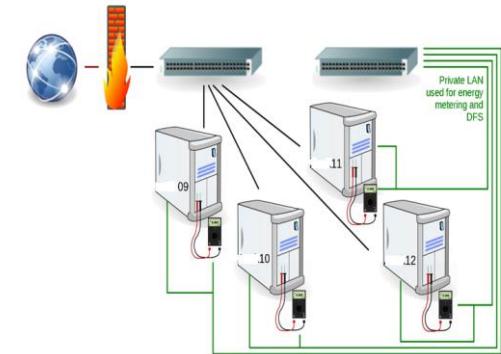
Pico=
 10^{-12}

Nano=
 10^{-9}

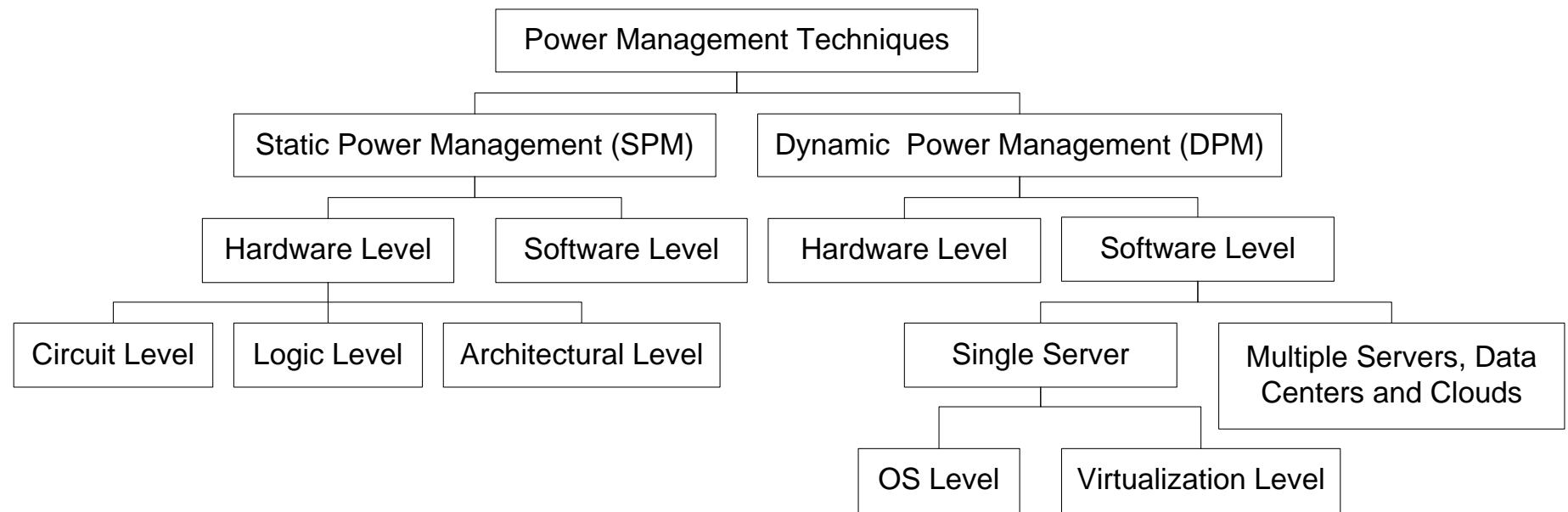
Joule: The work required to produce one **Watt** of power for one second

Ways to Measure Energy Consumption in Computing Systems

- Power meter attached to server
 - Electricity power usage monitor used to status tracking: server's Voltage, Current(Amps), Power(W), Energy (kWh), Frequency(Hz) ...
- Tools: direct measurement of power consumption based on sensor's integrated into the server
 - Intelligent Platform Management Interface (IPMI)
 - Running Average Power Limit (RAPL)
 - Other: Powerstat, Powertop



Power Management Taxonomy



Ways to Reduce Energy Consumption in Computing Systems

- Hardware specialisation
 - ASICs, FPGAs, accelerators – **less flexibility**
- Hardware optimisation
 - Instruction Level Parallelism, hardware threading – **diminishing returns**
- Hardware software co-design
 - Large one-off engineering cost – **can be expensive**
- Software closer to the hardware
 - Less abstractions – harder to program
 - Current tools are performance oriented, **not energy oriented**
- Optimal software operation
 - Off-line and On-line overheads – **can pay off**



Who is to blame? Hardware or Software?

- Hardware is often extensively optimised for power, performance, or cost
- Optimising for energy requires trade-off between power and performance
- Energy-optimised programmable hardware is ultimately controlled by software
- Levels of software optimisation
 1. Software design
 2. Software operation



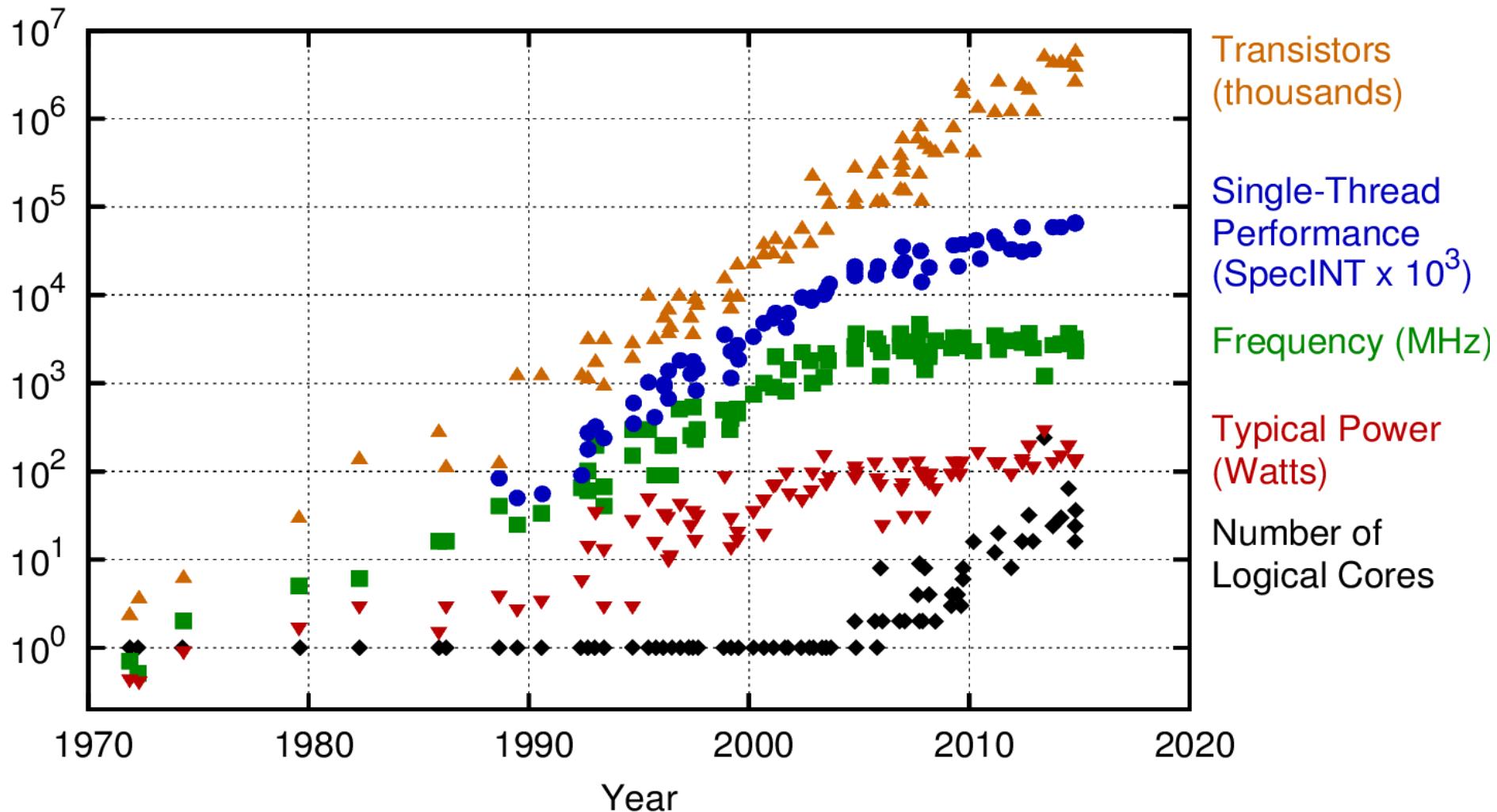
A screenshot of the Java Code Export v1.0.0 application window. The left pane shows the input Java code for the `Jim` class, which includes imports for `java.awt.event`, `java.awt.swing.event`, and `java.awt.tree`, along with a copyright notice. The right pane shows the output HTML code generated by the converter, which includes the DOCTYPE declaration, a link to the W3C Transitional DTD, and the converted Java code structure using spans and styles to maintain the original layout.

```
Java Code Export v1.0.0 - [G:\Projects\Code Converter\Java\Jim- v.2 (src)\Jim\source\Jim.java]
File Edit Single File Tools Help
Convert Indent Print
Save Output Preview
Input
Output
Copyright, 2007 Jason Dominicali.
This file is part of JIM - The Java Instant Messenger.
JIM is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as
published by
the Free Software Foundation; either version 2, or (at your
option) any later version.
JIM is distributed in the hope that it will be useful,
but
WITHOUT ANY WARRANTY; without even the implied warranty
of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.
You should have received a copy of the GNU General Public
License along with JIM; if not, write to the
Free Software Foundation, Inc., 59 Temple Place - Suite 330,
Boston, MA 02111-1307 USA.
<!DOCTYPE html PUBLIC "-//IETF//DTD HTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>G:\Projects\Code Converter\Java\Jim- v.2 (src)\Jim\source\Jim.java</title>
<style type="text/css">
span.c0 {color: #000000}
span.c1 {color: #000000}
span.c2 {color: #000000; font-family: Tahoma; font-size: 70%}
b.c3 {color: #000000}
span.c4 {color: #000000; font-family: Tahoma; font-size: 70%}
b.c5 {color: #000000}
</style>
<body>
<pre>
package Jim;
import java.awt.event.*;
import java.awt.swing.event.*;
import java.awt.*;
import java.awt.swing.tree.*;
/*
Copyright, 2007 Jason Dominicali.
This file is part of JIM - The Java Instant Messenger.
JIM is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as
published by
the Free Software Foundation; either version 2, or (at your
option) any later version.
JIM is distributed in the hope that it will be useful,
but
WITHOUT ANY WARRANTY; without even the implied warranty
of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.
You should have received a copy of the GNU General Public
License along with JIM; if not, write to the
Free Software Foundation, Inc., 59 Temple Place - Suite 330,
Boston, MA 02111-1307 USA.
</pre>
</body>
</html>

```

Learning from History

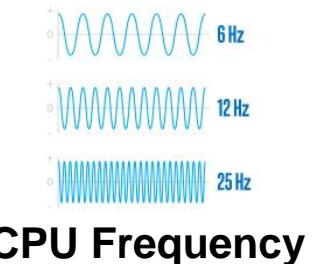
40 Years of Microprocessor Trend Data



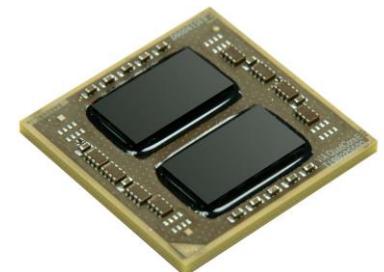
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Software Operation

- Energy-optimal software operation requires a careful configuration choice
- Hardware resources to monitor:
 - Hard disk: disk bytes/sec, disk read bytes/sec
 - CPU: percentage of processor usage
 - Memory: private bytes, working set, usage
 - Network: bytes total/sec, bytes sent/sec, bytes received/sec
 - IO: IO data (bytes/sec), IO read (bytes/sec), IO write (bytes/sec).
- Operation space is large!
 - Number of cores / threads
 - Optimising frequency of the cores
 - Type of core – heterogeneous computing



CPU Frequency



Multi-core processor

Software Operation

- Each point (configuration) in the software operation space poses a different **trade-off between power and performance**
 - Energy = Power / Performance, OR
 - Energy = Power x Time
- Power and performance are affected **in different ways** by operating frequency and the number / type of cores

Operating frequency and Voltage

- ↑ improves performance
 - much more power
- ↓ reduces power
 - less performance

Number and type of cores

- ↑ improves performance
 - more power

Energy Efficiency in Clouds: Towards Solutions

1. Data Centre Design Considerations
2. Dynamic Voltage and Frequency scaling
3. Server Consolidation
4. Virtualisation and Scheduling
5. Data Analytics
6. Software Design

Cloud Architecture – Where to Support Energy Efficiency?

User level

Cloud applications
Enterprise, Scientific, Social ...

User-Level
Middleware

Cloud programming: environments and tools
Interfaces, Concurrent and Distributed Programming,
Workflows, Libraries, Scripting

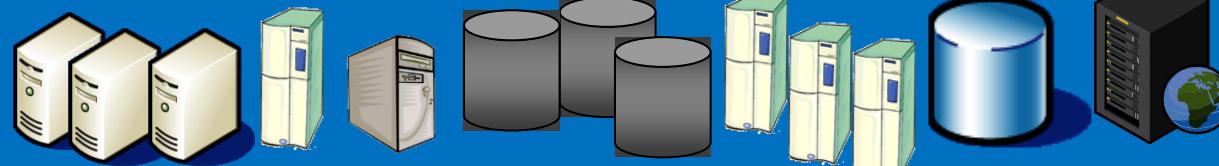
Apps Hosting
Platforms

QoS Negotiation, Admission Control, Pricing, SLA
Management, Monitoring, Execution Management, Metering,
Accounting, Billing

Core
Middleware

Virtual Machine (VM), VM Management and Deployment

Cloud resources



System level

Cloud Power-aware Computing

Hardware

e.g. DVFS (Dynamic Voltage and Frequency Scaling)
Cooling systems

Operating System

e.g. Power saving techniques

Energy Efficient Software Design

Datacenters

Turn-off idle servers
Virtual Machine consolidation
VM scheduling ...

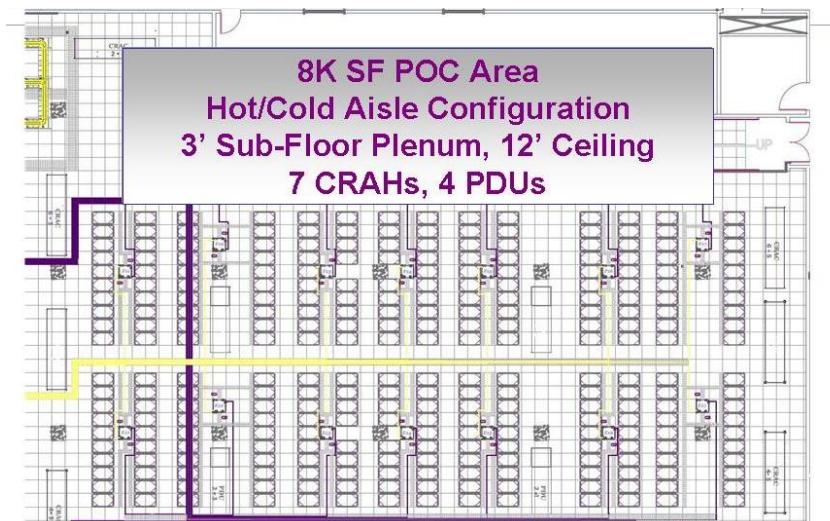
Virtualization

e.g. Xen Hypervisor

Energy efficient network protocols

1- Data Centre Design Considerations

- Cost-performance
- Datacenter layout
- Cooling system
- Server architecture
- Dependability via redundancy
- Network I/O
- Interactive and batch processing workloads
- **Server lifecycle** (approximately 18 months)

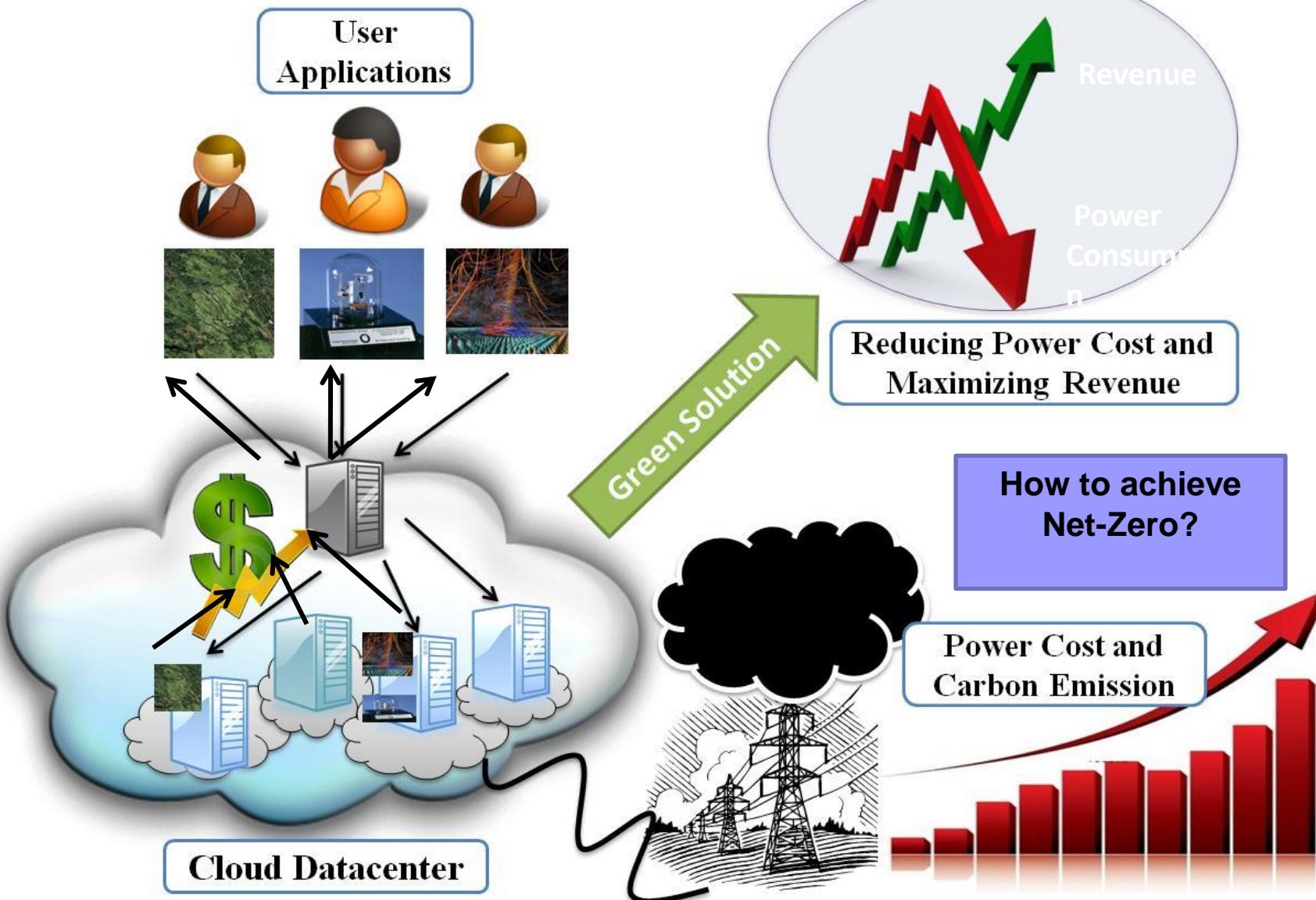


Towards Green Cloud Computing: Cloud Providers Measures

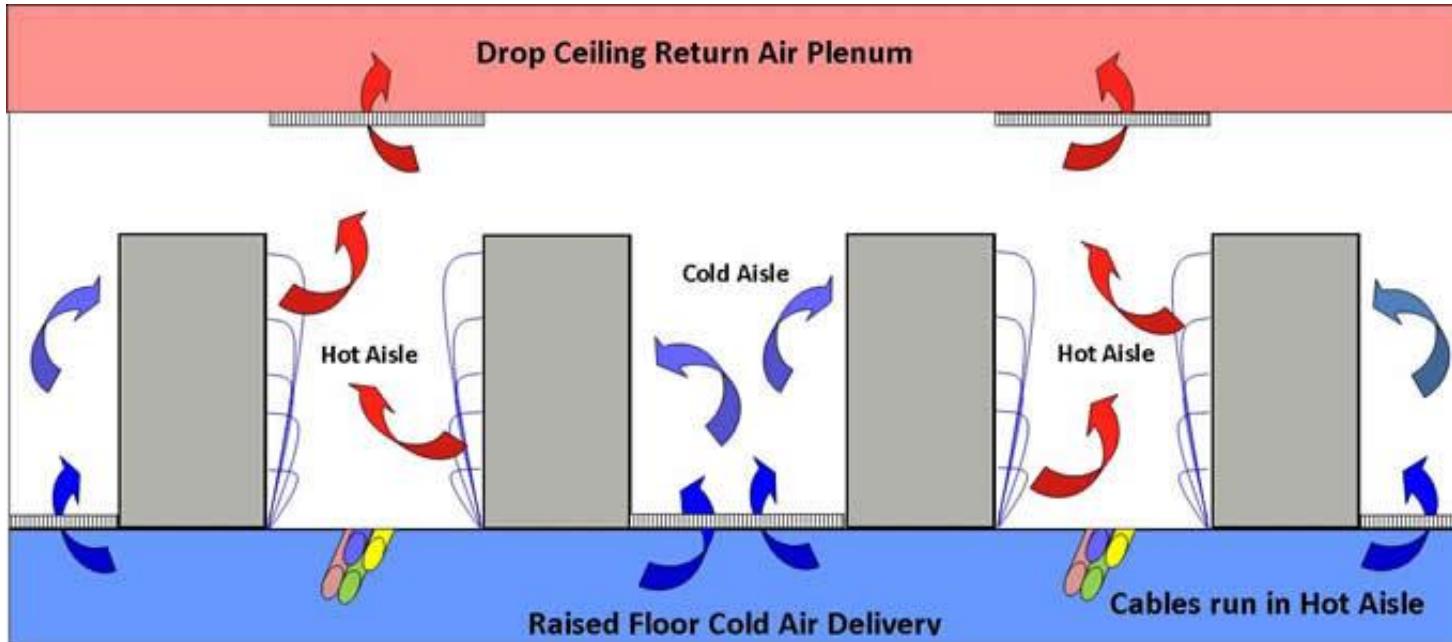
- Cloud service providers need to adopt **measures** to ensure that their profit margin is not dramatically reduced due to high energy costs.
 - Amazon estimate the energy-related costs of its data centres amount to 42% of the total budget that include both direct power consumption and the cooling infrastructure amortized over a 15-year period.
 - Google, Microsoft, and Apple are building large data centres in places where they can exploit **cheap hydroelectric power**.
- There is also increasing pressure from Governments worldwide to reduce carbon footprints, which have a significant impact on climate change.



Green Cloud Computing – Solutions?



Cooling



Cooling

Critical aspect of Cloud Data Centre infrastructure

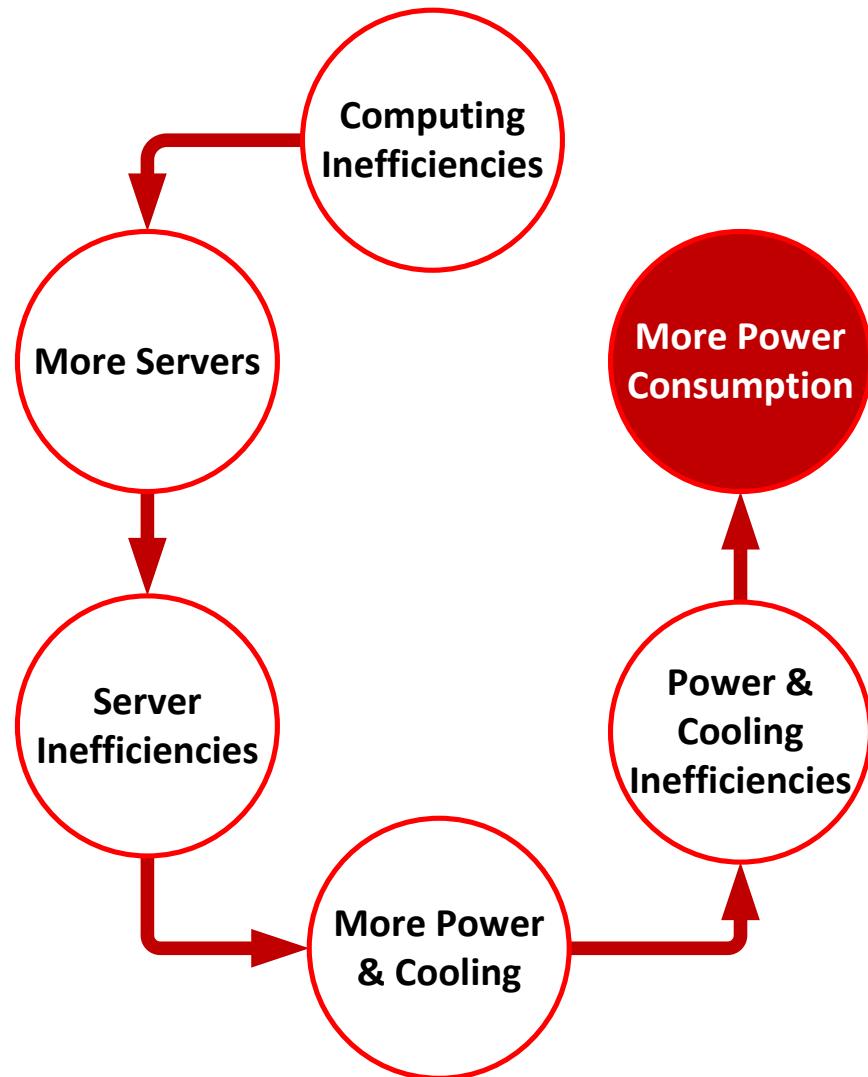
- Hot servers cause a number of problems (i.e. performance, damage).
- Humidity and pressure also factors to consider.

Different types of cooling available

- Direct Contact Liquid Cooling
- Evaporation (70,000 to 200,000 gallons per day for an 8 MW facility)
- Air
- Submerged liquid



Inefficiency Chain



- In the past, it was common to invest more financial resources to add **additional** servers, networking equipment, and storage to ensure uptime and security
- Challenging due to the lack of physical space, power supply, cooling resources and funding

Measuring Efficiency of a Data Centre: PUE

Power Utilisation Effectiveness (PUE)

- Total facility power / IT equipment power
- Median PUE (2022) study was still at 1.55, ideally should approach 1.1

Performance

- Latency is important metric because it is seen by users
- Service Level Objectives (SLOs)/Service Level Agreements (SLAs)
- E.g. 99% of requests must be below 100 ms

PUE doesn't tell the whole story...

- i.e 100 Watts of IT equipment might not all be useful
- Techniques to deflate the PUE (some companies reported a PUE of 1.02)

References

- Lago, P, Gu, Q & Bozzelli, P (2014), **A systematic literature review of green software metrics.** VU Technical Report
<https://research.vu.nl/ws/portalfiles/portal/910331>
- Rajkumar Buyya List of Papers:
<http://www.cloudbus.org/publications-years.html>
- Google Green Computing. <http://www.google.co.uk/green/>
- Google's Green Computing: Efficiency at Scale.
<http://static.googleusercontent.com/media/www.google.com/en//green/pdfs/google-green-computing.pdf>

COMP5850 – Cloud Computing



Energy Efficiency in Clouds

Part II

Plan of the Lecture

Goals

Understand concepts surrounding energy efficiency in clouds

Overview

- Introduction
- Powering the Cloud Infrastructure: Energy Consumption, Costs, Implications
- Power-Aware Computing: Trends and Issues
 - The role of hardware
 - The role of software
- Towards energy efficient Clouds
- Review of energy efficiency metrics
- Conclusion

Cloud Architecture – Where to Support Energy Efficiency?

User level

Cloud applications
Enterprise, Scientific, Social ...

User-Level
Middleware

Cloud programming: environments and tools
Interfaces, Concurrent and Distributed Programming,
Workflows, Libraries, Scripting

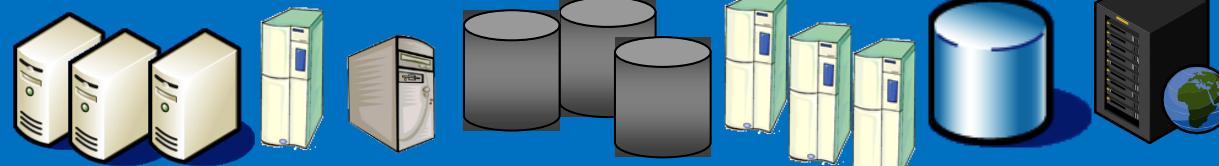
Apps Hosting
Platforms

QoS Negotiation, Admission Control, Pricing, SLA
Management, Monitoring, Execution Management, Metering,
Accounting, Billing

Core
Middleware

Virtual Machine (VM), VM Management and Deployment

Cloud resources



System level

Cloud Power-aware Computing

Hardware

e.g. DVFS (Dynamic Voltage and Frequency Scaling)
Cooling systems

Operating System

e.g. Power saving techniques

Energy Efficient Software Design

Datacenters

Turn-off idle servers
Virtual Machine consolidation
VM scheduling ...

Virtualization

e.g. Xen Hypervisor

Energy efficient network protocols

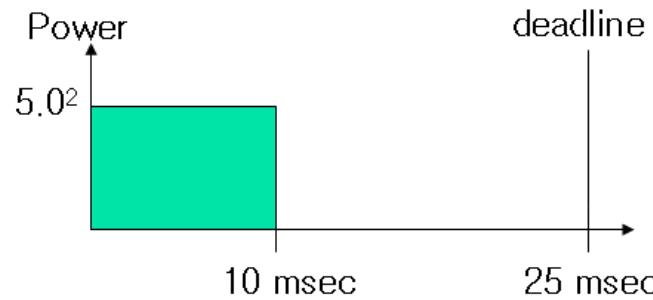
Energy Efficiency in Clouds: Towards Solutions

1. Data Centre Design Considerations
2. Dynamic Voltage and Frequency scaling
3. Server Consolidation
4. Virtualisation and Scheduling
5. Data Analytics
6. Software Design

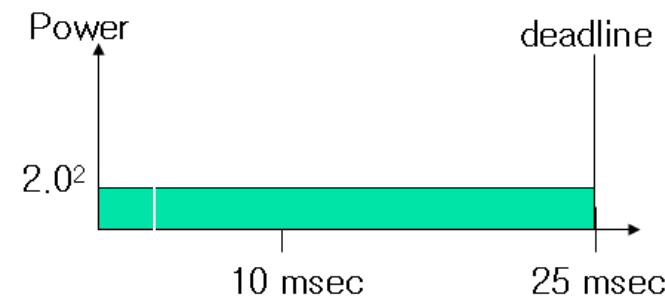
2- DVS (Dynamic Voltage Scaling)

- DVS (Dynamic Voltage Scaling) technique
 - Reducing the dynamic energy consumption by lowering the supply voltage at the cost of performance degradation
 - Recent processors support such ability to adjust the supply voltage dynamically.
 - The dynamic energy consumption = $\alpha * Vdd^2 * Ncycle$
 - Vdd : the supply voltage
 - $Ncycle$: the number of clock cycles

Note: **increasing** the clock frequency **increases** the work that a digital system can accomplish per unit time



(a) Supply voltage = 5.0 V



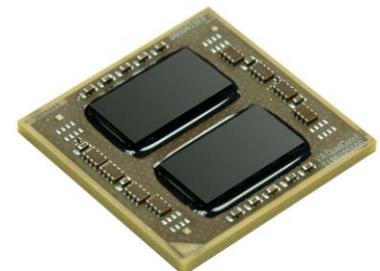
(b) Supply voltage = 2.0 V

DVFS-based Power Aware Scheduling

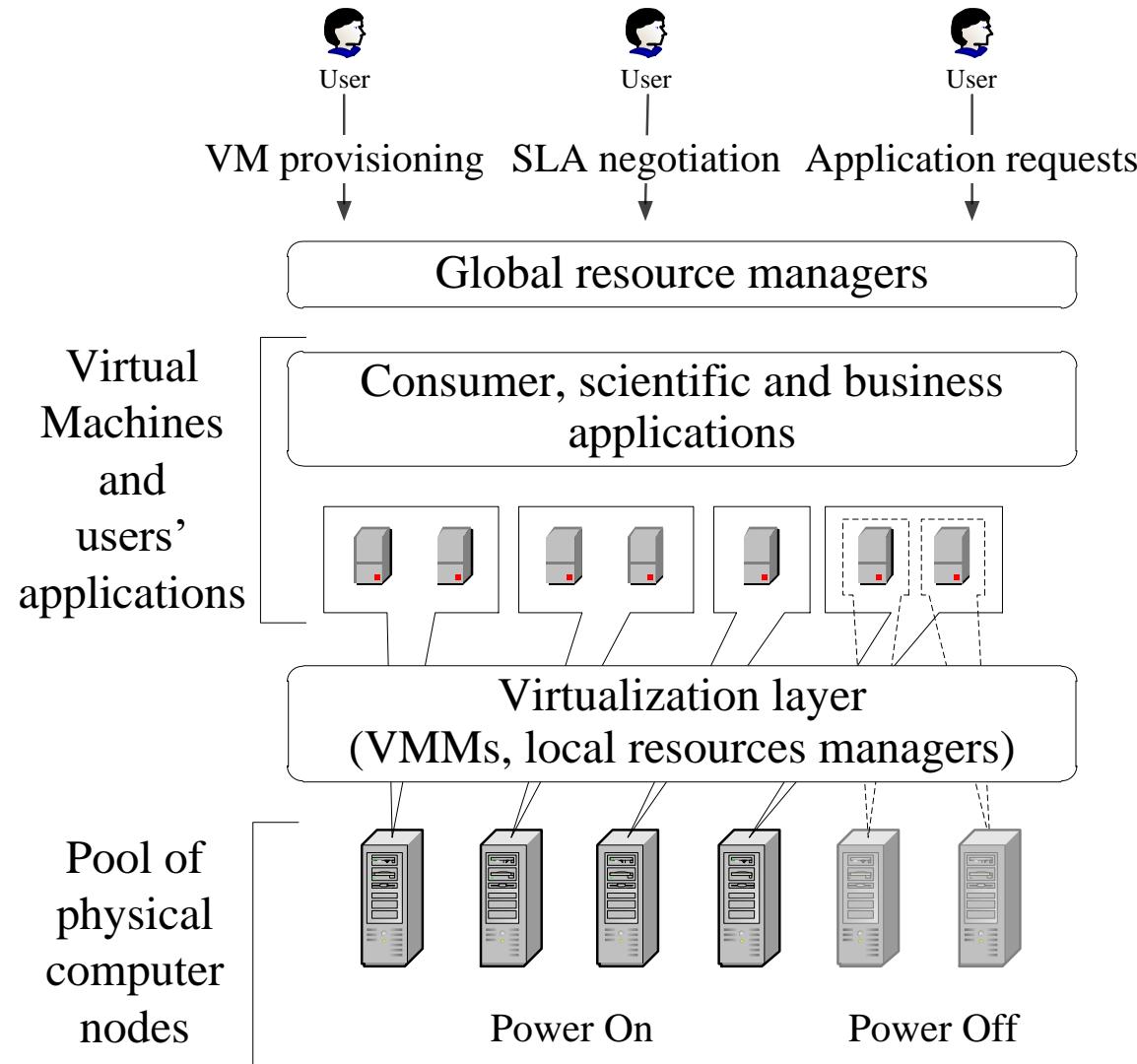
- Dynamic Voltage and Frequency scaling (DVFS): power-management technique where the clock frequency of a processor is **decreased** to allow a corresponding reduction in the supply voltage
- Motivation:
 - Develop Resource Management and Scheduling Algorithms that aims at
 1. minimising the energy consumption
 2. meet the job deadline.

Example: DVFS

- can **reduce** voltage and frequency by, e.g. 10%
- can **slow a program** by, e.g. 8%, but ... **reduce**
 - dynamic power by 27%
 - total power by, e.g. 23%
 - total energy by 17%



3- Server Consolidation



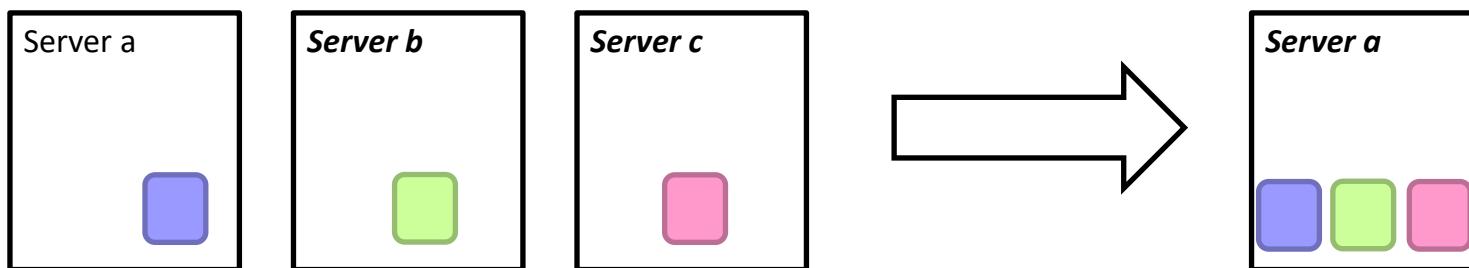
Consolidation

Consolidation

- Most servers utilisation is less than 5%
- Idle servers can consume up to 70% of max. energy output

Can't turn servers off!

- It breaks the hot-cold isle.
- They might not turn back on again when needed.
- System admins unsure how to use effectively.



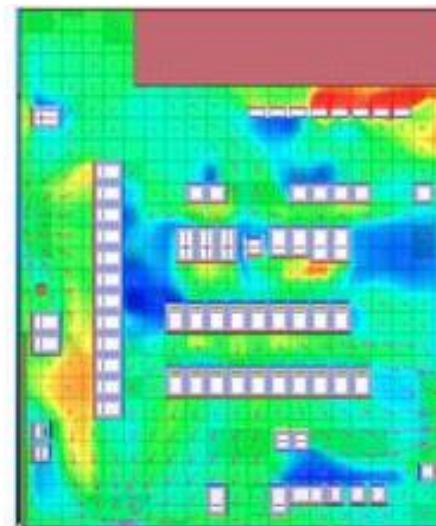
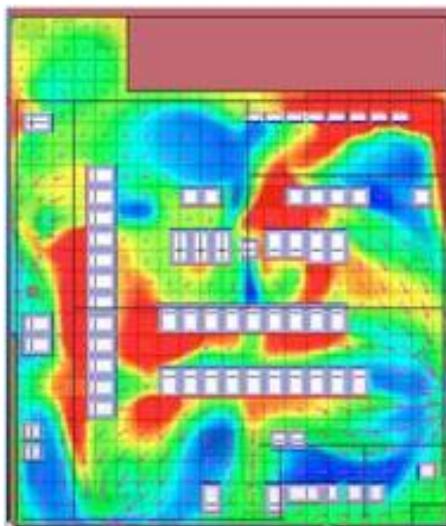
4- Virtualisation and Scheduling

Virtualisation

- Allows many applications on a single server.
- Smarter planning and allocation of resources.

Scheduling and Migration

- Allow to avoid/minimize data centre hotspots
- Migration requires additional cost and energy



The Problem

- **When** to migrate VMs?

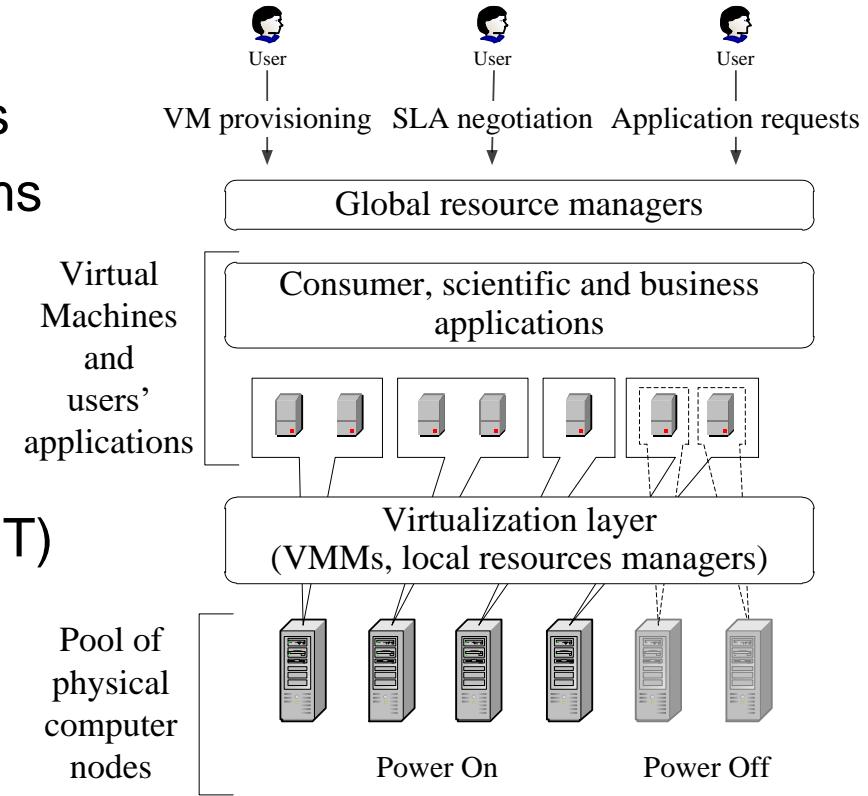
- Host overload detection algorithms
- Host underload detection algorithms

- **Which** VMs to migrate?

- VM selection algorithms
- E.g. Minimum Migration Time (MMT) policy

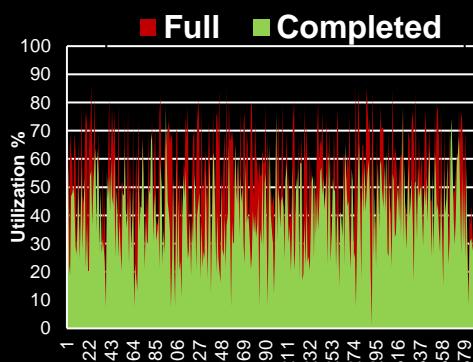
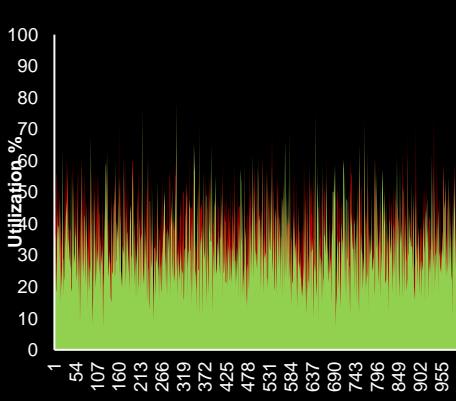
- **Where** to migrate VMs?

- E.g. Heuristic for the bin-packing problem
 - Power-Aware Best Fit

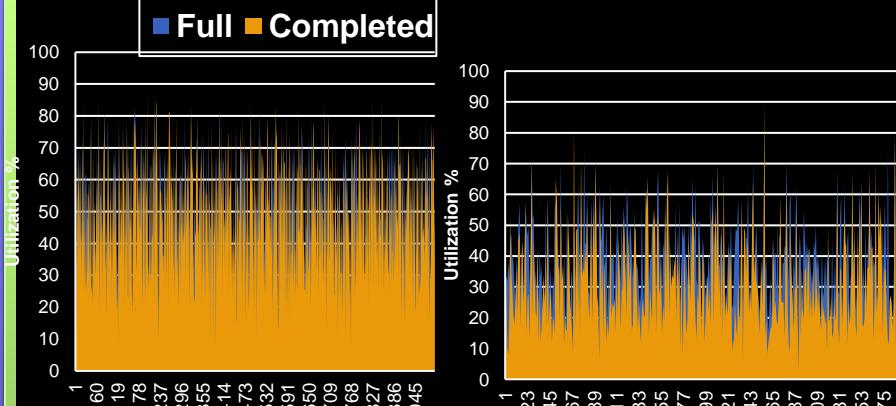
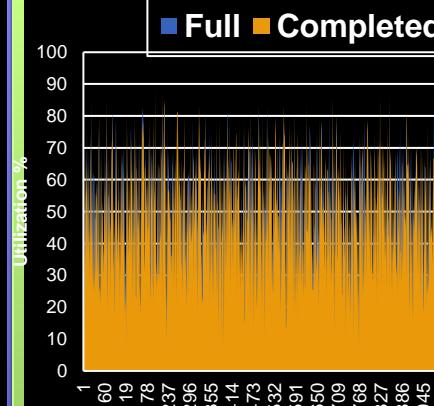


5- Data Analytics

Server CPU Utilization



Server Memory Utilization



- Use Big Data techniques to identify points of inefficiency

Server architecture types exhibit different resource inefficiencies.

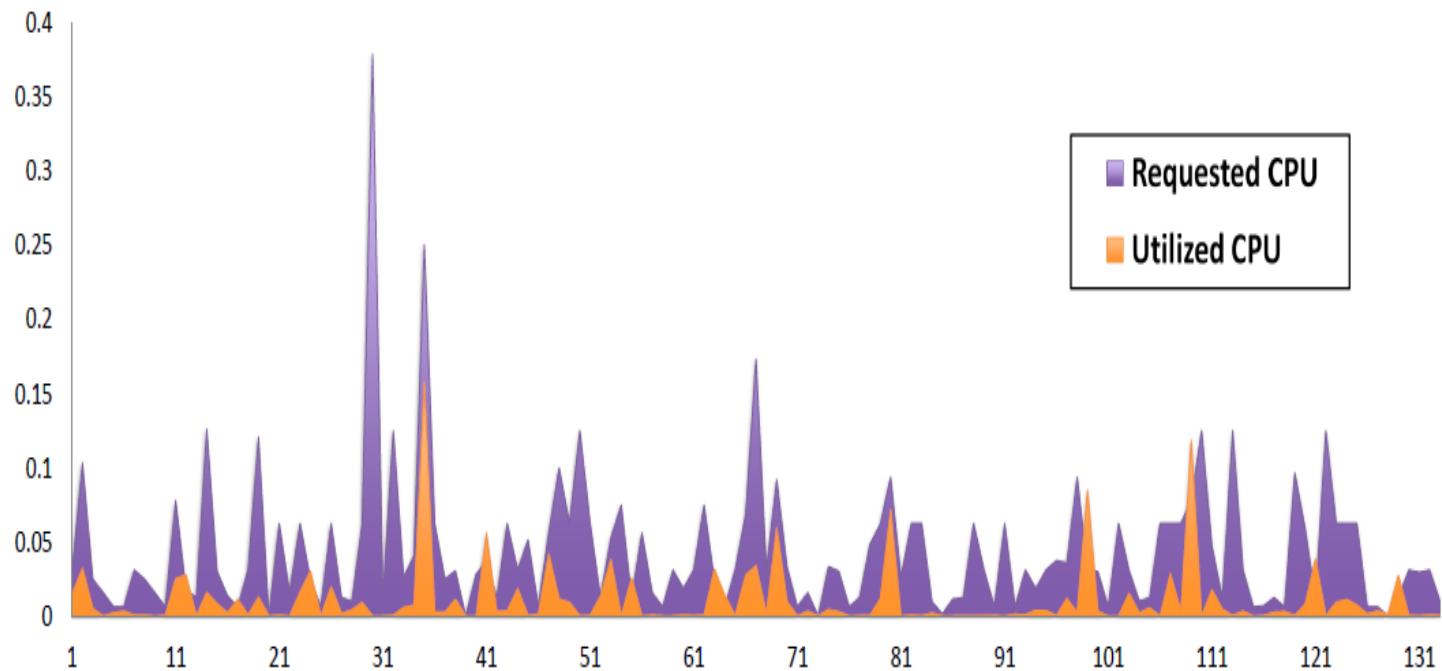
- **4.33-14.22%** waste for CPU
- **1.29-7.61%** waste for Memory

Use Big Data analytics to identify potential areas of improvement in resource utilization within specific servers and server types.

Can use Data Centre Infrastructure Managers (DCIMs) for monitoring.

User Resource Estimation

- Users always overestimate what they need (just in case, naive).
- Results in large amounts of idle resources
- Can use over provisioning (similar to overbooking flight seats).



CPU patterns in Google datacentre per user

VM Power Modelling

- Estimate the amount of energy consumption when it cannot be directly measured
- Use CPU Utilisation:
 - Consider CPU utilisation data for each VM
 - Assign the energy usage by the ratio produced by the utilisation data

$$VM_P_x = Host_P \times \frac{VM_Util_x}{\sum_{y=1}^{VM_Count} VM_Util_y}$$

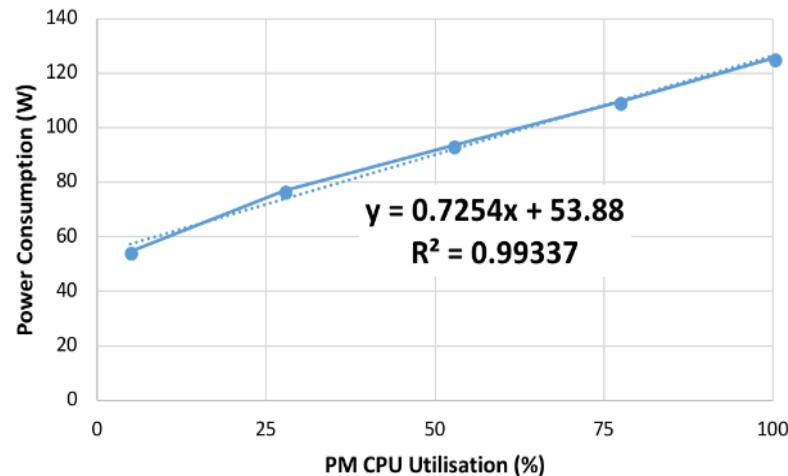
- VM_Px : named VM's power consumption
- $Host_P$: measured host power consumption
- VM_Util_x : named VMs CPU utilization
- VM_Count : number of VMs on the host machine
- VM_Util_y : CPU utilisation of a member of the set of VMs on the named host.

Physical Machine Power Modelling (1)

- Linear Model

$\text{PM}x_{\text{PredPwr}}$

$$= (\alpha \times (\text{PM}x_{\text{PredUtil}}) + \beta)$$



*CPU Utilisation vs Power Consumption –
Linear Regression Model*

$\text{PM}x_{\text{PredPwr}}$ (Watt) identified using a linear relation with the predicted PMs CPU utilisation, $\text{PM}x_{\text{PredUtil}}$

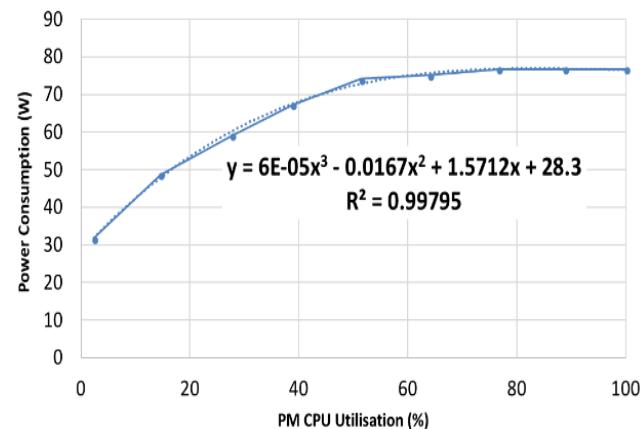
α and β are the slope and interceptor values obtained from the regression relation.

Physical Machine Power Modelling (2)

- Polynomial Model

$$\begin{aligned} \text{PM}_x_{\text{PredPwr}} \\ = & (\alpha(\text{PM}_x_{\text{PredUtil}})^3 - \gamma(\text{PM}_x_{\text{PredUtil}})^2 \\ & + \delta(\text{PM}_x_{\text{PredUtil}}) + \beta) \end{aligned}$$

- Used to characterise the relation between the power consumption and CPU utilisation of the PM
- α , γ and δ are all slopes
- β is the intercept
- $\text{PM}_x_{\text{PredUtil}}$ is predicted PM CPU utilisation.



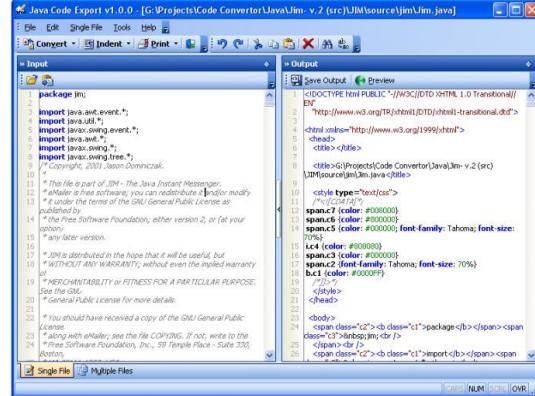
CPU Utilisation vs Power Consumption –
Polynomial Regression Model

6- Energy Efficiency and Software Design

- Can service applications **be built** with energy awareness in mind?
- Can we **relate** software design and energy use in operation?
 - Energy efficient service application at requirements/design stage
 - Service application instrumented to cooperate with novel software, platform, and infrastructure components
- Context: cloud-based services
 - Emphasis on shared software components which are likely to be used and reused many times in many different applications
 - Imperative that the software to be developed is as energy efficient as it possibly can be.



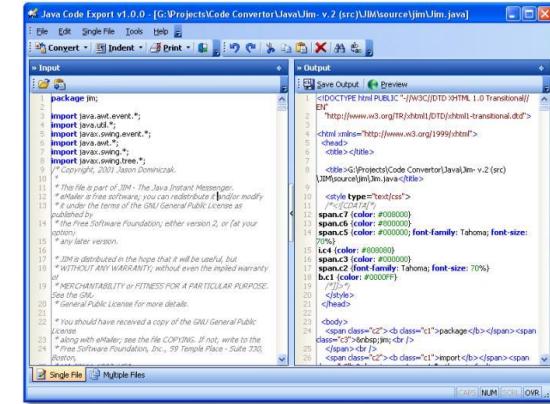
ENERGY EFFICIENCY



```
Java Code Export v1.0.0 - [G:\Projects\Code Converter\Java\Jim- v. 2 (src)\Jim\source\jim\Jim.java]
File Edit Single File Tools Help
Convert Indent Print
Input Output
<DOCTYPE HTML PUBLIC "-//IETF//DTD/xhtml-transitional.dtd">
<html>
<head>
<title>G:\Projects\Code Converter\Java\Jim- v. 2 (src)\Jim\source\jim\Jim.java</title>
<link href="http://www.w3.org/1999/xhtml/DTD/xhtml-transitional.dtd" type="text/css" rel="stylesheet"/>
<style type="text/css">
span{color: #000000}
span2{color: #000000}
span3{color: #000000; font-family: Tahoma; font-size: 70%}
b{color: #000000}
b2{color: #0000FF}
b3{color: #000000; font-family: Tahoma; font-size: 70%}
b4{color: #0000FF}
</style>
<body>
<span class="c2"><b>package</b></span><span class="c2"><b>import</b></span><span class="c2"><b>*</span>
```

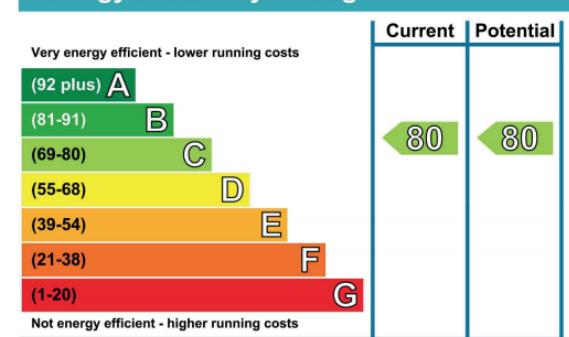
Programming Model, Hotspots and Patterns

- Software developer constructs and analyses an application using the Programming Model
- Uses the Programming Model Plug-in to develop the application
- Analyses the resultant program using a **Code Optimiser** Plug-in to identify
 - **Patterns** in the code
 - Potential energy **hotspots**
- Analysis provides feedback to the software developer thus enabling an adaptive software development methodology through
 - **Monitor:** the code's performance
 - **Analyse:** identifying energy hotspots
 - **Plan:** potential changes to the code to improve its performance
 - **Execute:** recompiling the code enabling further monitoring



The screenshot shows a software interface titled "Java Code Export v1.0.0". On the left, there is an "Input" pane displaying Java code for a class named "jm". The code includes imports for various Java packages and annotations like "@JM". On the right, there is an "Output" pane showing the generated HTML code. The HTML includes doctype declarations, meta tags, and a body section with spans containing the Java code. The CSS styles for these spans are visible, such as "color: #000000" and "font-family: Tahoma; font-size: 70%".

Energy Efficiency Rating



Energy Awareness across Cloud Layers

User level

Cloud applications
Enterprise, Scientific, Social ...

User-Level
Middleware

Cloud programming: environments and tools
Interfaces, Concurrent and Distributed Programming,
Workflows, Libraries, Scripting

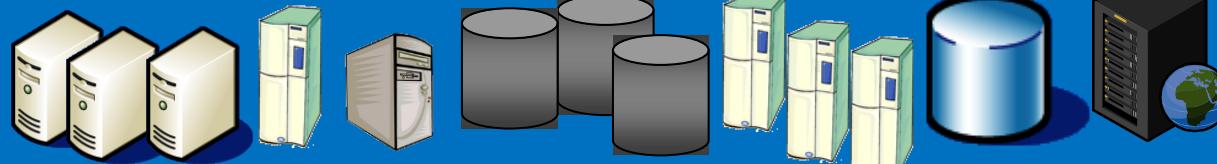
Core
Middleware

QoS Negotiation, Admission Control, Pricing, SLA
Management, Monitoring, Execution Management, Metering,
Accounting, Billing

Virtual Machine (VM), VM Management and Deployment

System level

Cloud resources



Cloud Stack Adaptive Management

Energy Efficiency

A Final Note on Policy

Societal enforcement and policies

- Many companies want to be seen as “green”
- Increased pressure for enforcement to restrict datacentre energy costs

Currently, demand trumps savings

- **Jevons paradox** = efficiency results in increased use!
- The easier you make it to consume the product the greater the consumption will be!



Summary

- Clouds are essentially Data Centres hosting application services offered on a subscription basis
- However, they consume high energy to maintain their operations.
 - → high operational cost + environmental impact
- Reviewed some power-aware computing solutions
 - e.g. DVFS, VM consolidation to reduce energy consumption, scheduling
- Open Issues:
 - EE Algorithms, software design
 - EE Resource Management for other workloads (e.g., workflows)
 - Importance of understanding metrics.

References

- K. Hwang, G. Fox, and J. Dongarra, Distributed and Cloud Computing: from Parallel Processing to the Internet of Things Morgan Kauffmann Publishers, 2011
- Rajkumar Buyya List of Papers:
<http://www.cloudbus.org/publications-years.html>
- Google Green Computing. <http://www.google.co.uk/green/>
- Google's Green Computing: Efficiency at Scale.
<http://static.googleusercontent.com/media/www.google.com/en//green/pdfs/google-green-computing.pdf>
- Lago, P, Gu, Q & Bozzelli, P (2014), **A systematic literature review of green software metrics.** VU Technical Report
<https://research.vu.nl/ws/portalfiles/portal/910331>

Metrics for Measuring Software Energy Consumption

Metrics Type	Measurement Unit(s)	Example Metric
Energy	Joule (J) Watt (W) Ampere (A) Kilowatt-hour (kWh)	Application Performance: measures energy consumption per computing application unit.
Performance	GFLOPS/kWh Computing Unit/kWh Percentage (%) Seconds (s)	Throughput: number of service requests served at a given time period.
Utilisation	Percentage (%) Megabyte (MB) Megahertz (MHz) GB/s	CPU Usage: deals with the relative CPU utilisation of specific applications.
Economics	Currency (£, \$...)	Lifecycle Cost: measures the total process lifecycle expenditures, e.g. costs of conceptual modelling, design, development, deployment ...
Performance/Energy	GFLOPS/Watt	Power Efficiency: evaluates how efficiently the power is used within a system – energy consumption with respect to the service throughput
Pollution	CO2 units	Supply Chain: defines the index of carbon emissions, being caused by logistics etc. needed for the execution of services

Metrics Context for Measuring Energy Consumption

Metrics Context	Example Metric
Application	Computational Energy Cost : evaluates the energy cost due to CPU processing, memory access, I/O operations of an application.
Architecture	Distributed System Energy Consumption : estimates the energy consumed by a distributed system as the sum of the energy consumed by its constituent components and connectors.
Service	Execution Plan Energy Efficiency : measures how efficiently a service uses energy.
Virtual Machine	Disk Energy Model : energy consumed by the disk over time duration for a single virtual machine.
Data Centre	Data Centre Energy Productivity : measures the number of bytes that are processed (useful work) per kWh of electric energy with respect to the whole data centre.
Embedded Software	Executed Instructions Count Measure : evaluates the energy consumption dealing with the number of executed assembly instructions and considering a typical embedded integer processor core.
Server	Server Power Utilisation : evaluates the amount of power used by a server with respect to the server CPU utilisation.
DBMS	Aggregated Cost : evaluates a power-aware query plan, in order to improve the energy efficiency of the executed queries.

COMP5850 – Cloud Computing



The Internet of Things and Ubiquitous Clouds

Plan of the Lecture

Goals

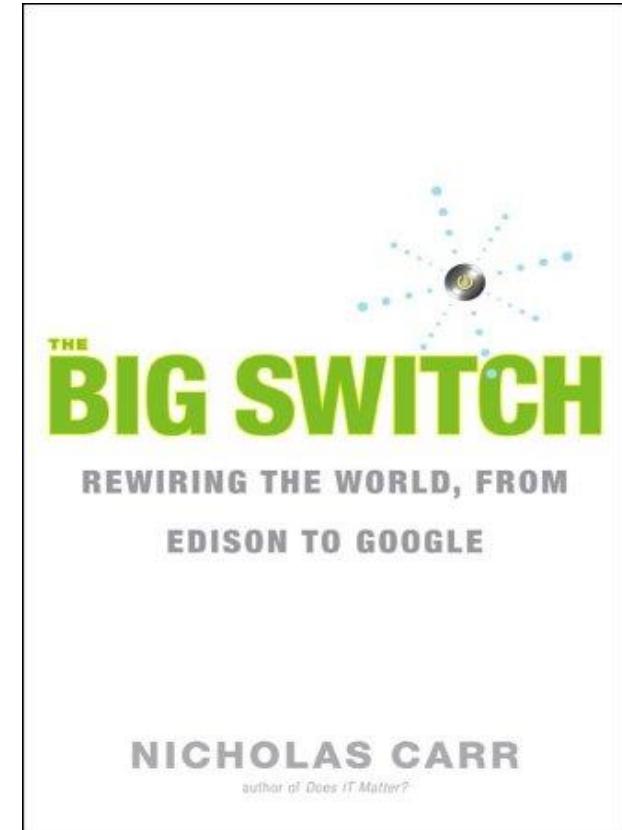
Understand concepts surrounding ubiquitous clouds and the Internet of Things (IoT)

Overview

- Introduction
- IoT Definition
- Application Examples
- IoT Challenges
- IoT and Data
- IoT Architecture
- Cloud Computing Support
- Technologies
- Conclusion

The Big Switch in Early 21st Century

- More powerful computers
- Infinite storage
- High bandwidth network and pervasive connectivity
 - Petabit network in sight
 - Wideband wireless mobility
- Industries race to build massive datacenters (capacity)
- Virtualisation helps realise the economics of scale



Major Technological Challenges in Developing Future Internet

- Programmable Networking Architectures
 - Software Defined Networks
- Fusion of The Internet, Mobile and TV Networks
- Named Data Networking beyond the TCP/IP
- Intelligent Routing and Content Distribution
- Enforced Security and Privacy Protection
- ...



Clouds for the Future Internet and Social Networking

- Clouds will be the base to provide future Internet services
 - Future Internet covers not only **people** and **machines** but also any **objects** or **things**.
- Internet of Things (IoT) are emerging - interesting IoT applications must leverage clouds in processing and storage of massive amount of data, dynamically.
- Clouds, IoT and social networks are reshaping human relations, affect our daily life, and impact the global economy, political system ...

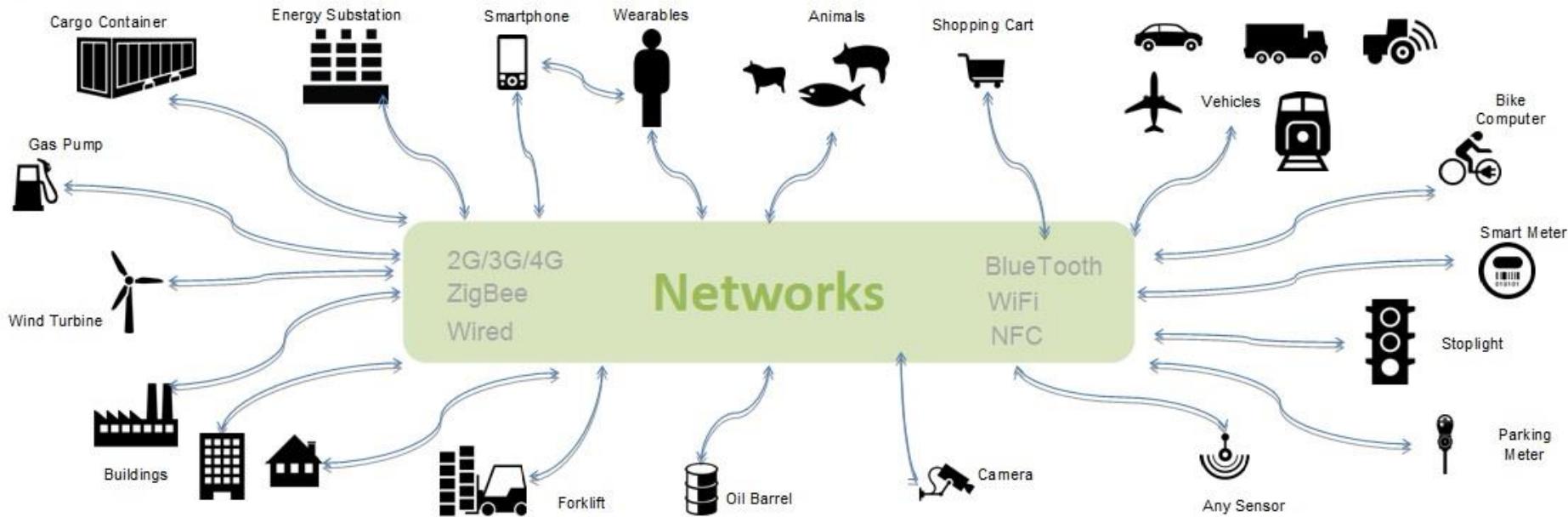
What IoT Actually Means

Kevin Ashton coined "*Internet of Things*" phrase to describe a system where the Internet is connected to the physical world via **ubiquitous sensors**



What are Things?

“Things” refer to any physical object with a device that has its **own IP address** and can **connect & send/receive** data via a **network**



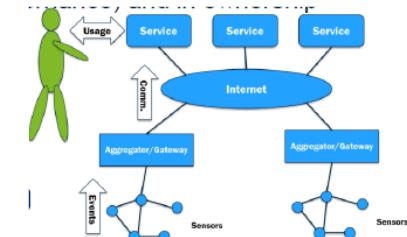
Courtesy of IBM <https://ibmcai.files.wordpress.com/2014/06/iot-network.jpg>

What is IoT: Two Different Views (View 1)

The Internet of Things (IoT) envisions systems made out of networked sensors and smart objects whose purpose is to measure/control/operate on an environment in such a way to make it intelligent, usable, and programmable and capable of providing useful services to humans.

Single Administrative Domain

- In a single administrative domain IoT envisions a system comprising sensors/actuators, aggregators and gateways, service control. These components use Internet protocols and/or specific sensor protocols to **communicate**.
- These systems could be quite **large** in size and **complex** in technologies (even if we tend to use a few of them), but are **homogeneous** from a management perspective (at least in processes and governance) and in ownership

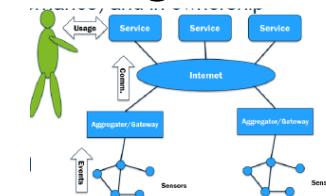


What is IoT: Two Different Views (View 2)

The Internet of Things (IoT) envisions systems made out of networked sensors and smart objects whose purpose is to measure/control/operate on an environment in such a way to make it intelligent, usable, and programmable and capable of providing useful services to humans.

Multiple Administrative Domain

- The IoT envisions the integration of several **heterogeneous** systems (i.e., networks of networks), each one using different technologies, interfaces and protocols and governed/managed by different Actors by means of different processes and managements functions.
- The IoT in a multi domain envisions a **self-configuring and adaptive** complex system made out of networks of sensors and smart objects whose purpose is to interconnect “all” things, including every day and industrial objects in such a way to make them intelligent, programmable and more capable of providing useful services to humans.



How Ubiquitous?

A forecast by International Data Corporation (IDC) estimates that there will be **41.6 billion** IoT devices in 2025, capable of generating **79.4 zettabytes** (ZB) of data.

- Every mobile
- Every auto
 - Every door
 - Every room
- Every part, on every parts list
- Every sensor in every device ... in every bed, chair or bracelet ... in every home, office, building or hospital room ... in every city and village ... on Earth ...

What are Internet-Connected Things? Active/Passive, With/Without Context

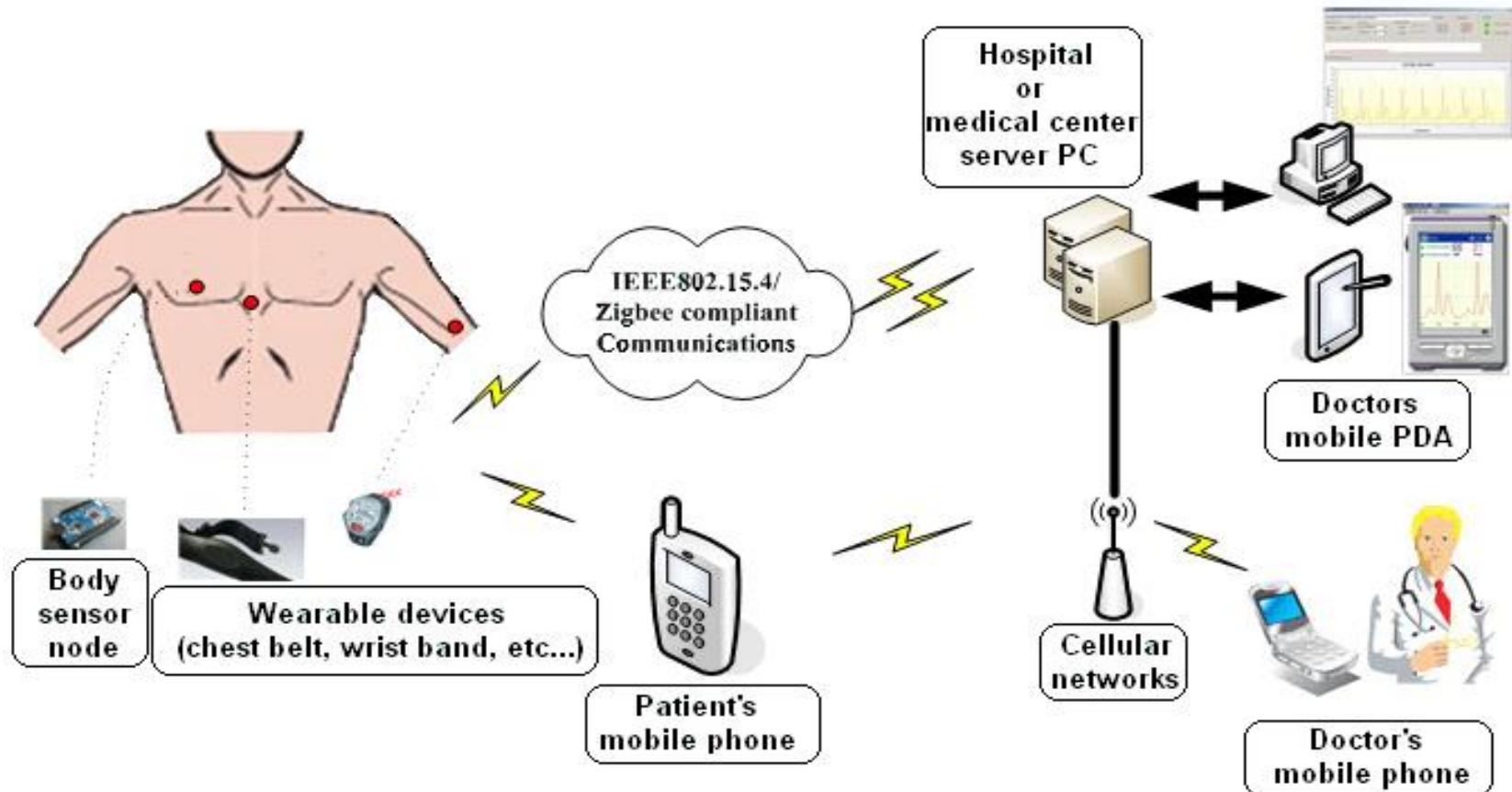
	Generic Info	Contextualized Info
Passive Objects	A Tag, A pointer to some information	Info + a location
Reactive Objects	A switch at home (turn it on/off), A smart meter	Home Automation (when temperature reaches 20 C stop heating)
Autonomous Objects	A Vending Machine, An Intelligent Fridge	A Cleaning Robot

Sensors: Examples in E-Health



Courtesy of Libelium

Example Application 1: Telemedicine



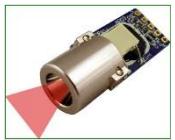
Patient Data Transferred Using a Wireless Sensor Network.

Example 2: How to Integrate a Fridge into IoT (1)?



- Just to start ... it is just a fridge
 - Invented for home use in 1913
 - Keeps things cold and fresh
- Add the **basics** ... some **sensors** built in that tells
 - when the water filter need replacing
 - if the doors are shut
 - if they are open for a certain amount of time:
 - start a timer
 - An alarm goes off
 - ... this requires some kind of **computational intelligence**
 - etc

How to Integrate a Fridge into IoT (2)?



- Make use of barcode and RFID (radio Frequency Identification) scanning
- Add more **sensors**, e.g. weight and ... **cameras**
- **Example 1:** A camera in the butter tray can sense when the butter is low
 - Test the weight of the butter
 - Send a warning when this happens
- **Example 2:** Have a set of recipes programmed into the fridge
 - It can suggest recipes based on the food it **sees** inside
- Nice features to have which require
 - Local sensing
 - local processing
 - ... again this requires some kind of **computational intelligence**

How to Integrate a Fridge into IoT (3)?

IoT fridge = computational intelligence + network connectivity



- Network connectivity you can add another set of features
- When the stock is low
 - Voice (talking fridge)
 - Send an SMS
 - Order food
- How can food be ordered?
 - go straight to a designated online supermarket
 - Can check first for the cheapest (using Web services?)
 - Place the order, grab money out of the bank account, transfer it to whoever food is purchased from
- Food is delivered the next day on the doorstep.

The Challenges

Every one of those sensor and control points is generating data. Often, it's very informative and very private data. Systems are needed to help those devices *talk to each other, manage all that data, and enforce proper access control.*



Big Data means BIG Challenges

All of the *messaging, management, and access control* technologies used in these large-scale device networks must be massively **scalable**.



Open Protocols

Current Internet and software methods are

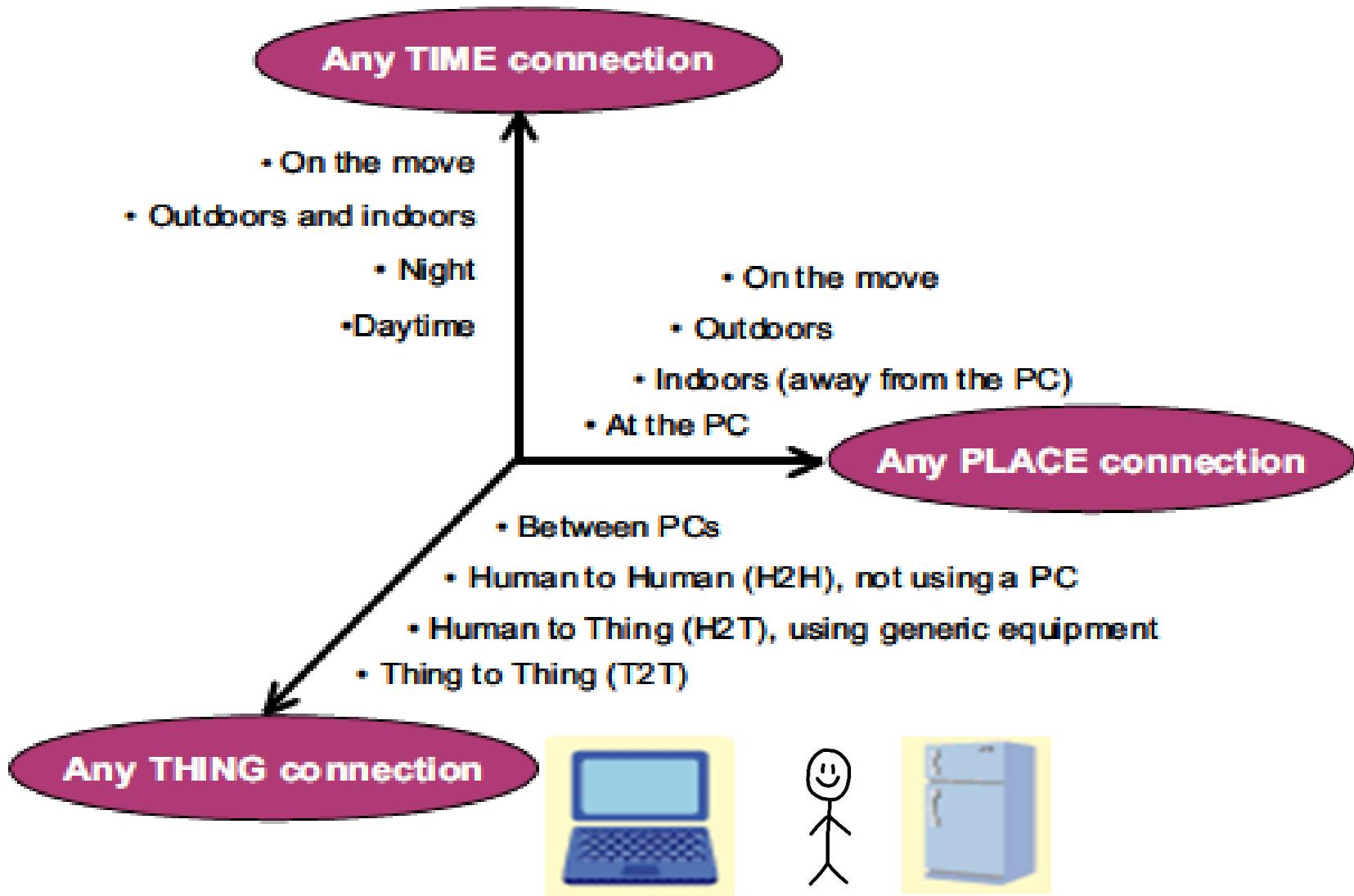
- highly modular (APIs)
- highly distributed (Cloud), and
- "loosely coupled" (SOA), microservices, serverless functions

In today's systems, *every LEGO brick comes from a different source* – and they all still must snap together.

This requires *open, rapid and safe* development methods.



Opportunities of IOT in 3 Dimensions



Sensors: Examples in IoT Data and Identity of Things

Things have Identities (and Owners)



“My” Smart Thing

People have Identities and use Things



Me



Third Parties

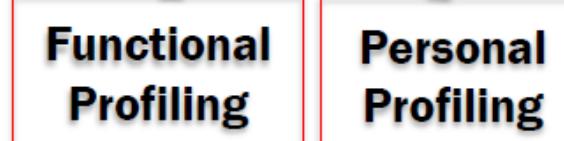
Identity Relation



Functional Relation (events and cmds)



Raw data to be
transformed into
Info



Who, Where, When, What, Why, ...

How Much Data and Traffic?

Device Type	2014	2019
Nonsmartphone	22 MB/month	105 MB/month
M2M Module	70 MB/month = 2.33 MB /Day	366 MB/month = 12.2 MB /Day
Wearable Device	141 MB/month	479 MB/month
Smartphone	819 MB/month	3,981 MB/month
4G Smartphone	2,000 MB/month	5,458 MB/month
Tablet	2,076 MB/month	10,767 MB/month
4G Tablet	2,913 MB/month	12,314 MB/month
Laptop	2,641 MB/month	5,589 MB/month

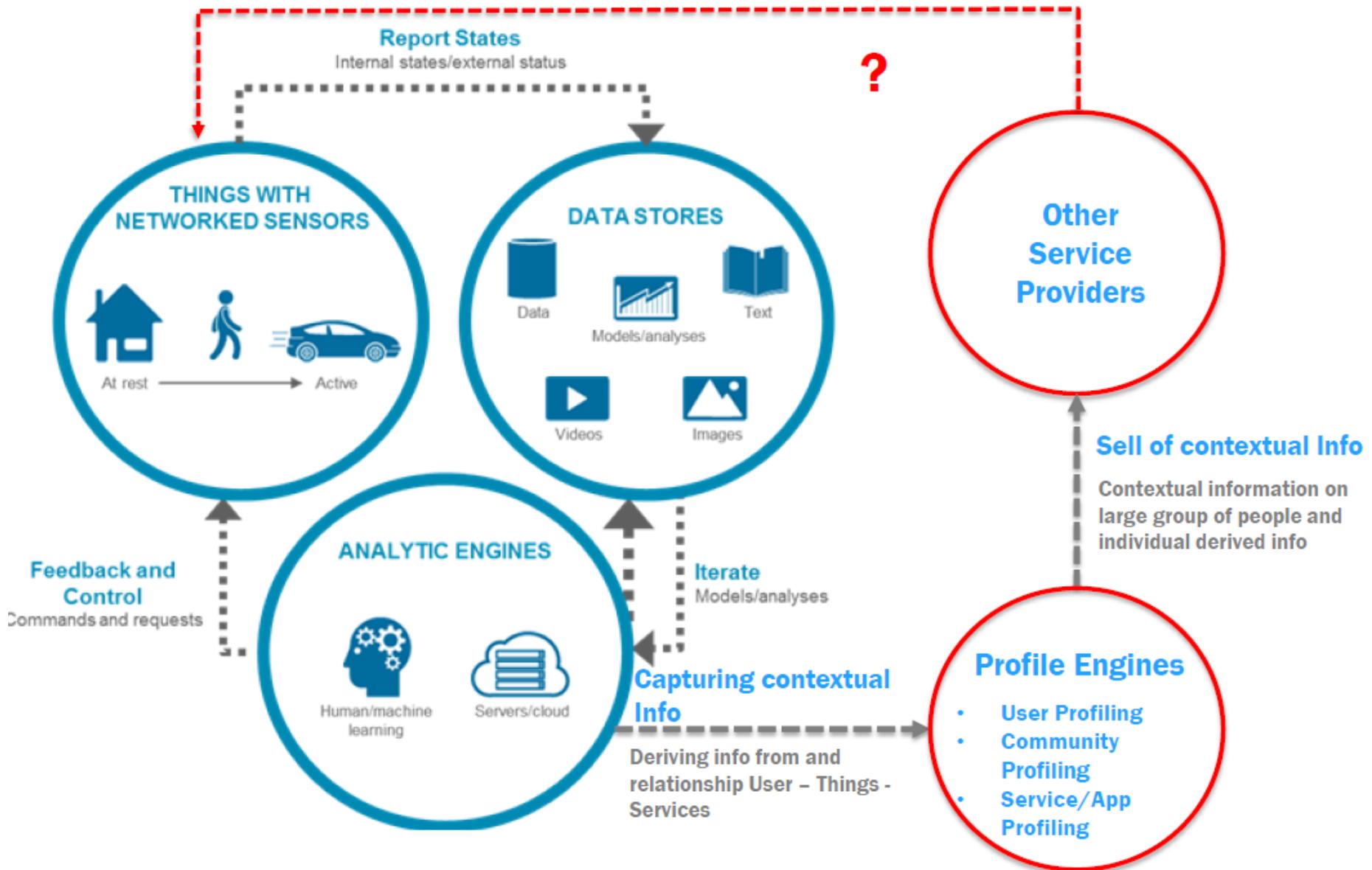
Source: Cisco VNI Mobile, 2015

= 27 Byte /s

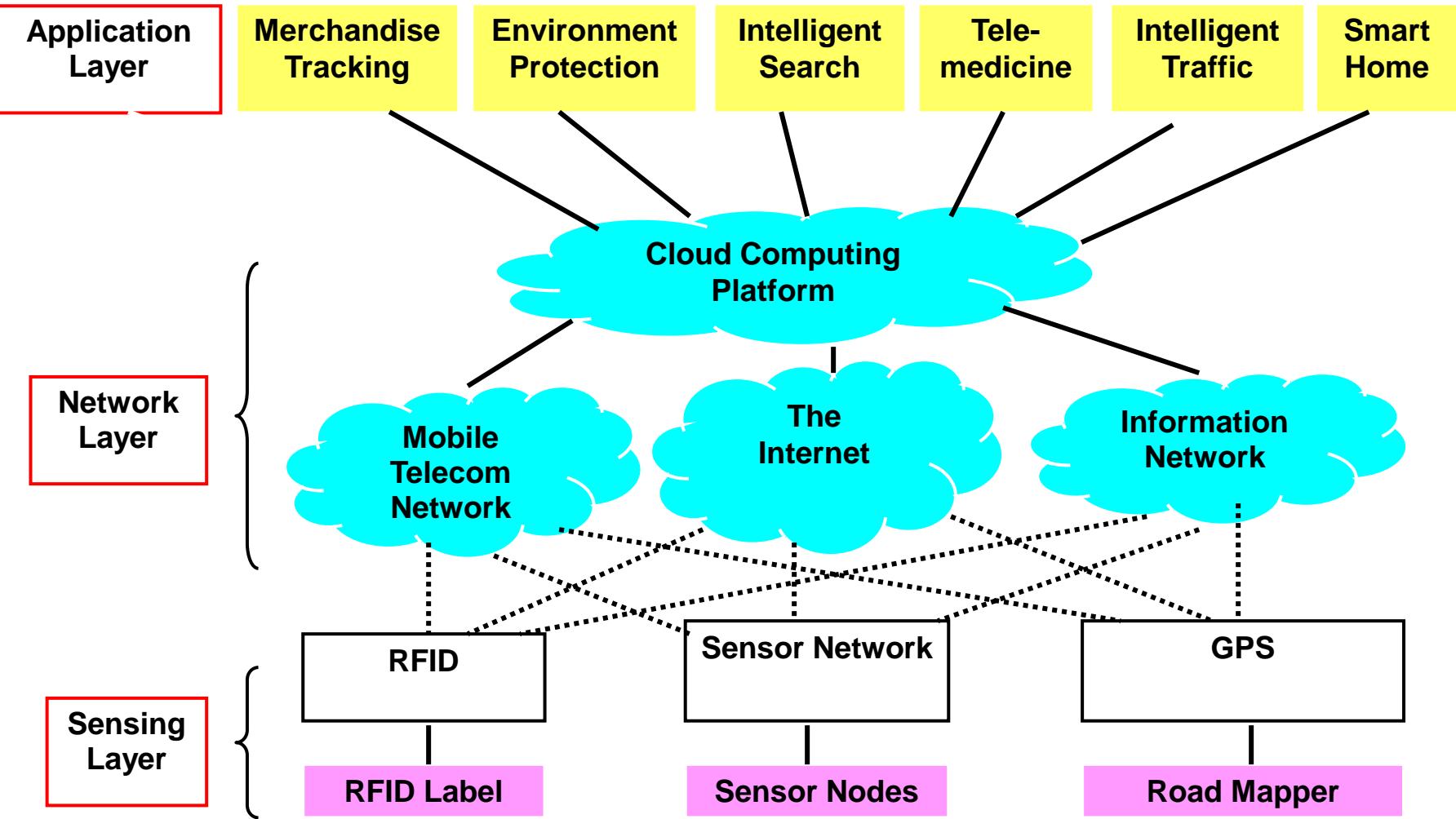
= 141 Byte /s

http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html#Trend_3_Measuring_Mobile_IoE

What to do with Data?

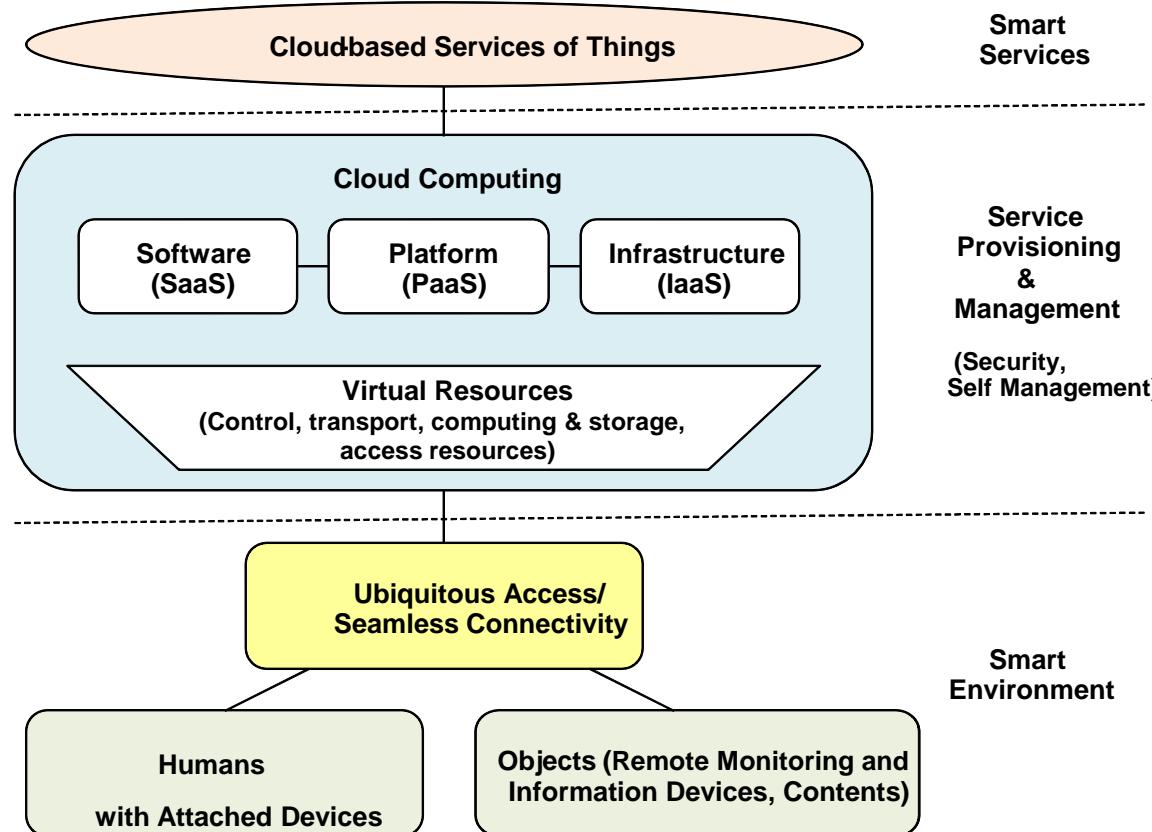


Architecture of The Internet of Things

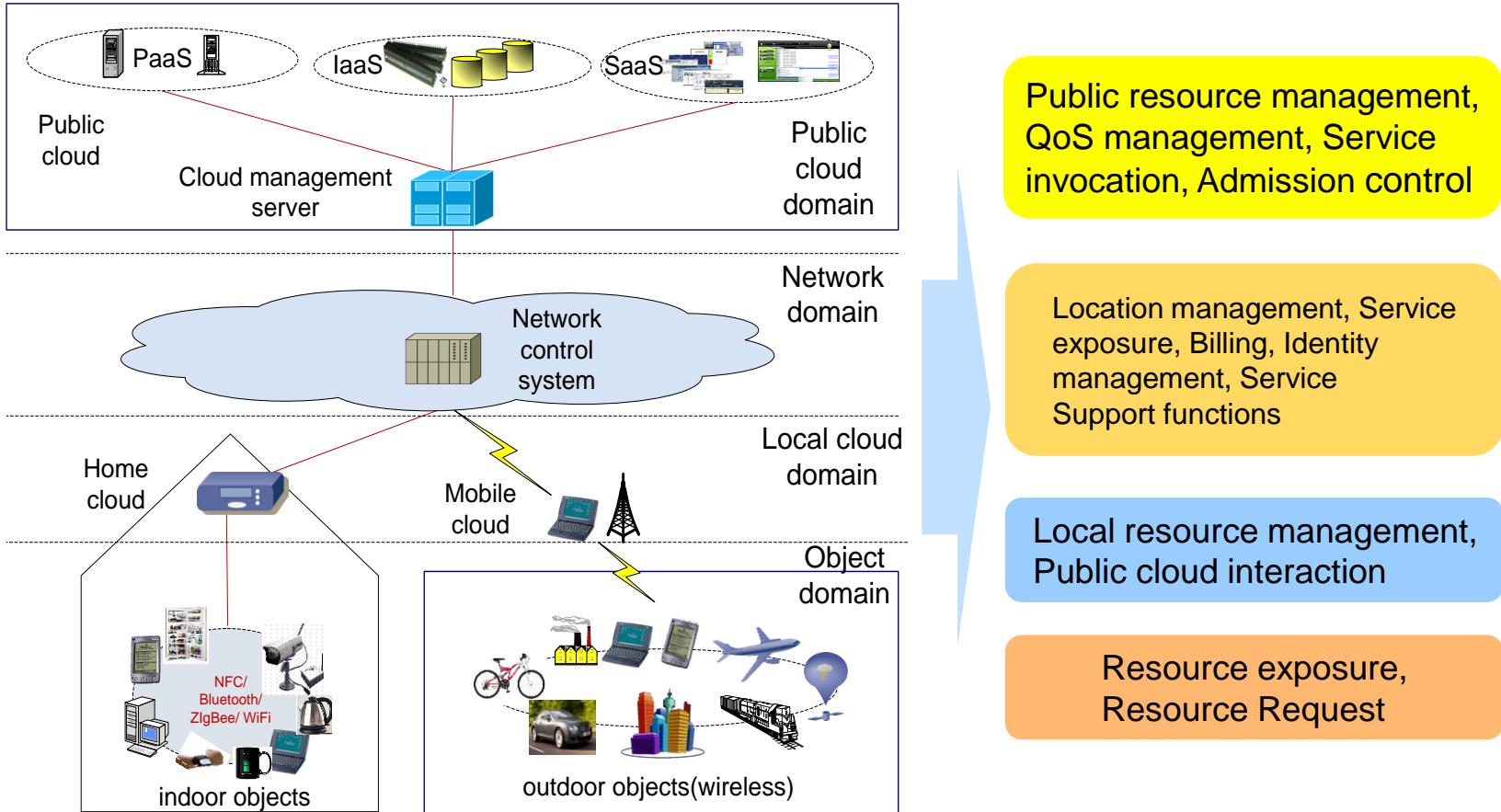


RFID: Radio Frequency Identification Technology

Conceptual Diagram: Cloud-based IoT



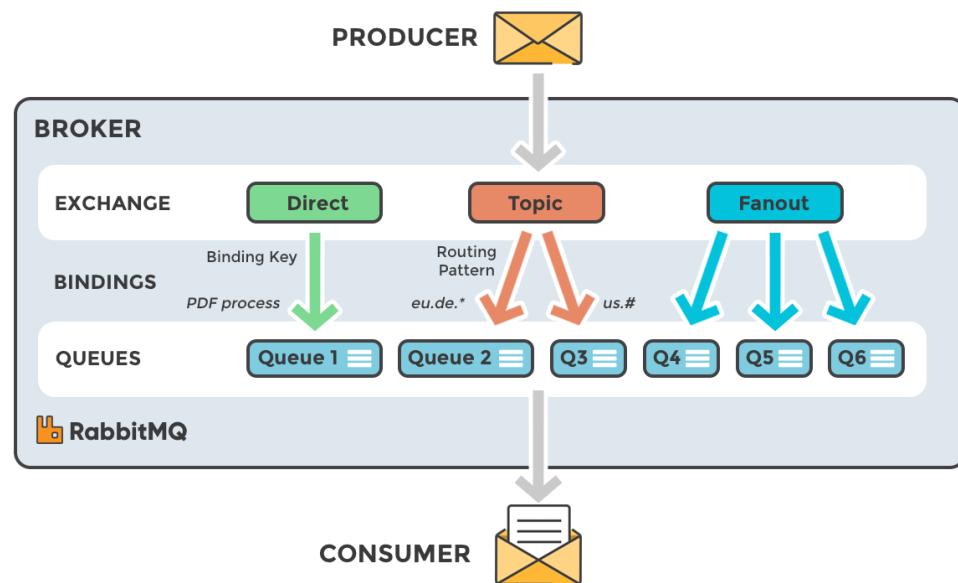
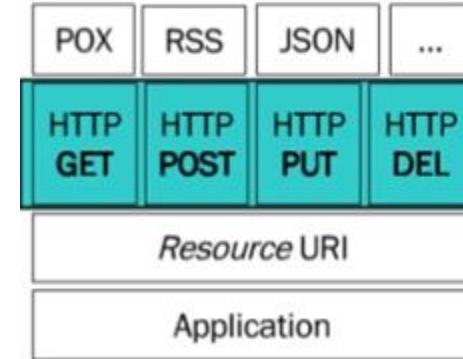
The IoT using Distributed Clouds



- Wireless connections:
 - RFID
 - Bluetooth/ZigBee
 - Wi-Fi
 - Radio
 - 5G, 6G and beyond
 - IEEE 802....
- Message Protocols:
 - TCP/IP
 - HTTP/HTTPS/WebSocket
 - REST
 - WS-*
 - AMQP (see below)
- Dedicated Real-Time Information:
 - Data Distribution Service
 - High Level Architecture (IEEE 1516.2010) for simulation
- Asynchronous Message Queues:
 - RabbitMQ
 - ActiveMQ
 - Apache kafka
- Application specific APIs

Task: compare some of the technologies listed here

REST



References

1. What is IoT? Oracle [Online].
<https://www.oracle.com/uk/internet-of-things/what-is-iot/>
 2. Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions, Future Generation Computer Systems, Volume 29, Issue 7, 2013, pages 1645-1660
- Libelium Case Studies – Connecting Sensors to the Cloud
<http://www.libelium.com/case-studies/>

COMP5123M – Cloud Computing Systems



The Internet of Things and Ubiquitous Clouds
Part II

Plan of the Lecture

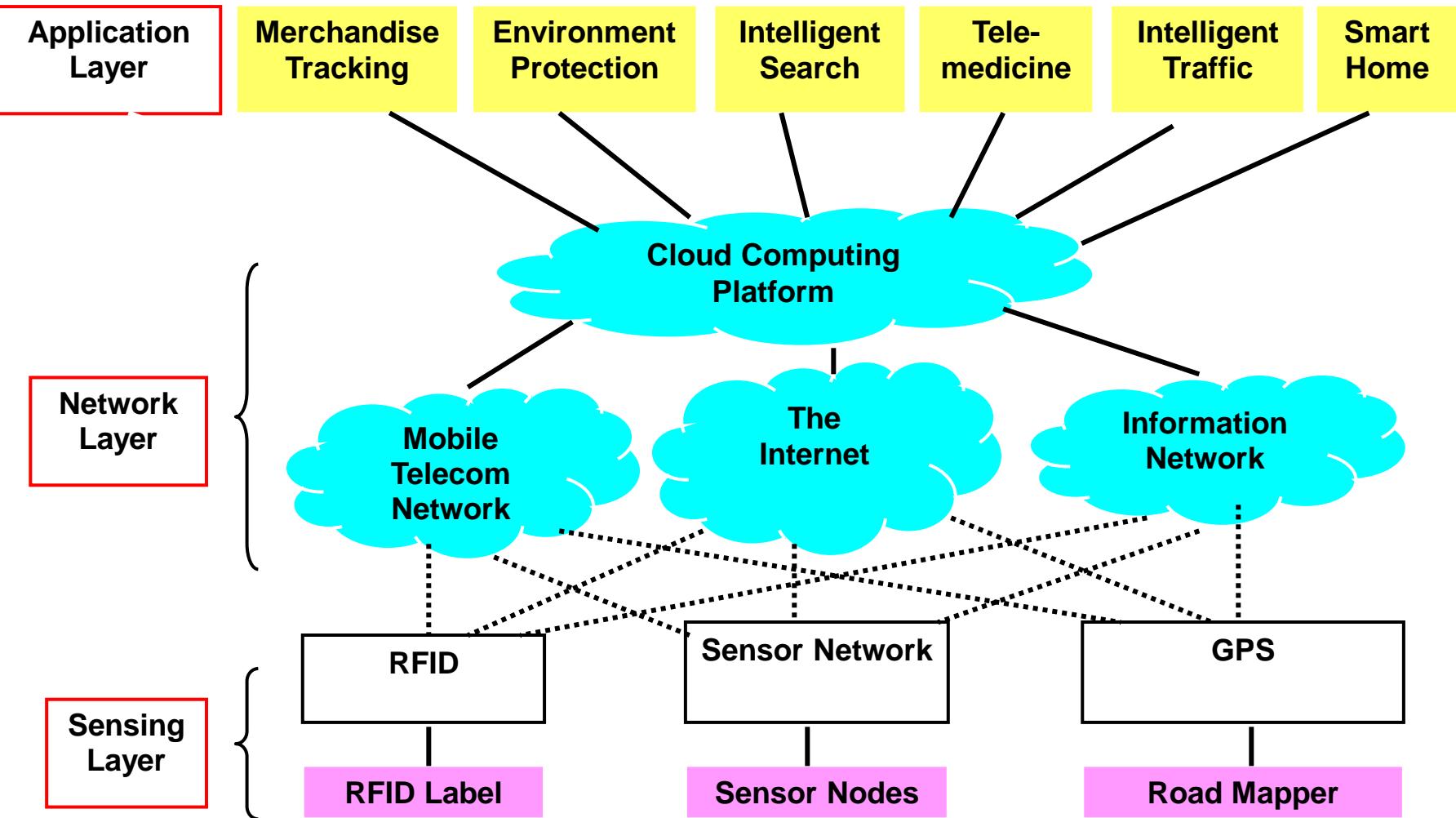
Goals

Understand concepts surrounding ubiquitous clouds and the Internet of Things (IoT)

Overview

- Introduction
- IoT Definition
- Application Examples
- IoT Challenges
- IoT and Data
- IoT Architecture
- Cloud Computing Support
- Networking Technologies
- Conclusion

Architecture of The Internet of Things



RFID: Radio Frequency Identification Technology

Wireless Networks for Supporting Ubiquitous Computing

Wireless Networks for Supporting Ubiquitous Computing

Market Name Standard	ZigBee 802.15.4	GSM/GPRS CDMA/1XRTT	WiFi 802.11g	Bluetooth 802.15.1
Application focus	Monitoring and control	Wide area voice and data	Web, e-mail, video	Cable replacement
System Resources	4 KB–32 KB	18 MB+	1 MB+	250 KB+
Battery Life (days)	100–1,000+	1–7	0.5–5	1–7
Network Size	Unlimited (2^{64})	1	32	7
Bandwidth (Kbps)	20–250	64–128+	54,000+	720
Range (meters)	1–100+	1,000+	1–100	1–10+
Success Metrics	Reliability, power, cost	Reach, quality	Speed, flexibility	Cost, convenience

GSM: Global System for Mobile Communication

GPRS: General Packet Radio Service

CDMA: Code Division Multiple Access

ZigBee Architecture

○ Zigbee Devices

- Full Function Devices (FFD's) - ZigBee Coordinator , ZigBee Router
- Reduced Function Devices (RFD's) - ZigBee End Device

○ ZigBee Coordinator (ZC)

- Only one required for each ZB network, Initiates network
- Acts as 802.15.4 2003 coordinator
- May act as router once network is formed.

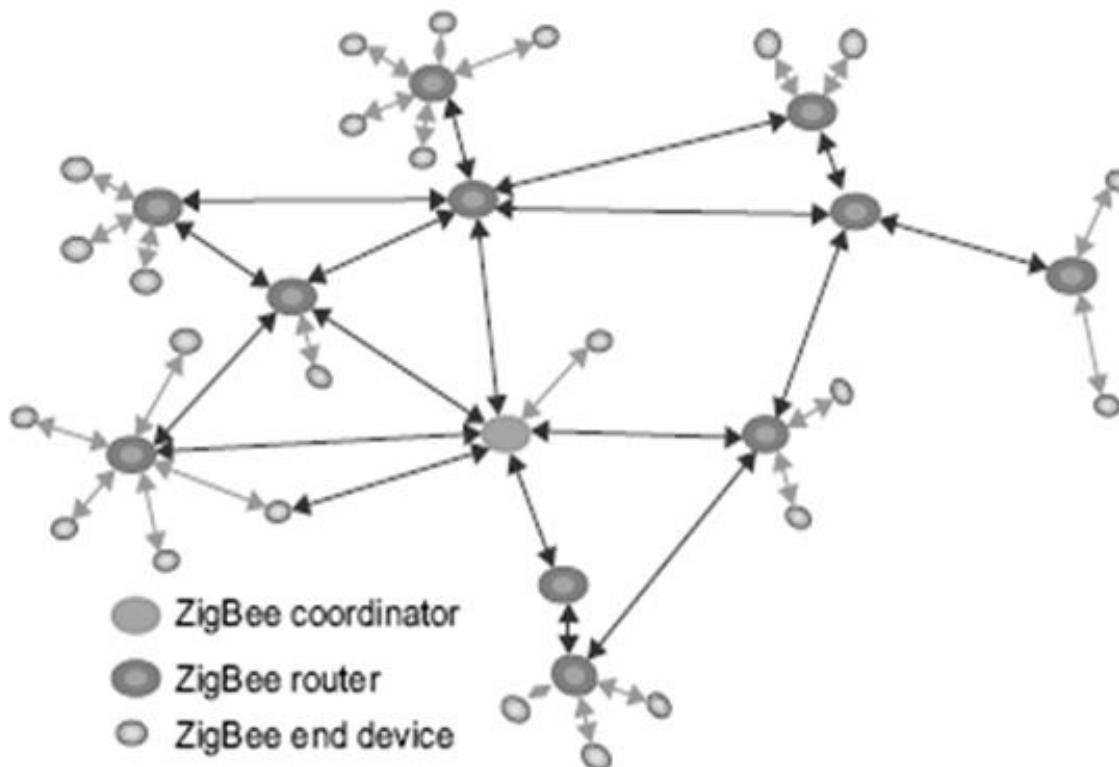
○ ZigBee Router (ZR)

- Optional component, may associate with ZC
- Multihop routing of messages.

○ ZigBee End Device (ZED)

- Optional network component
- Does not participate in routing.

Typical Zigbee Architecture

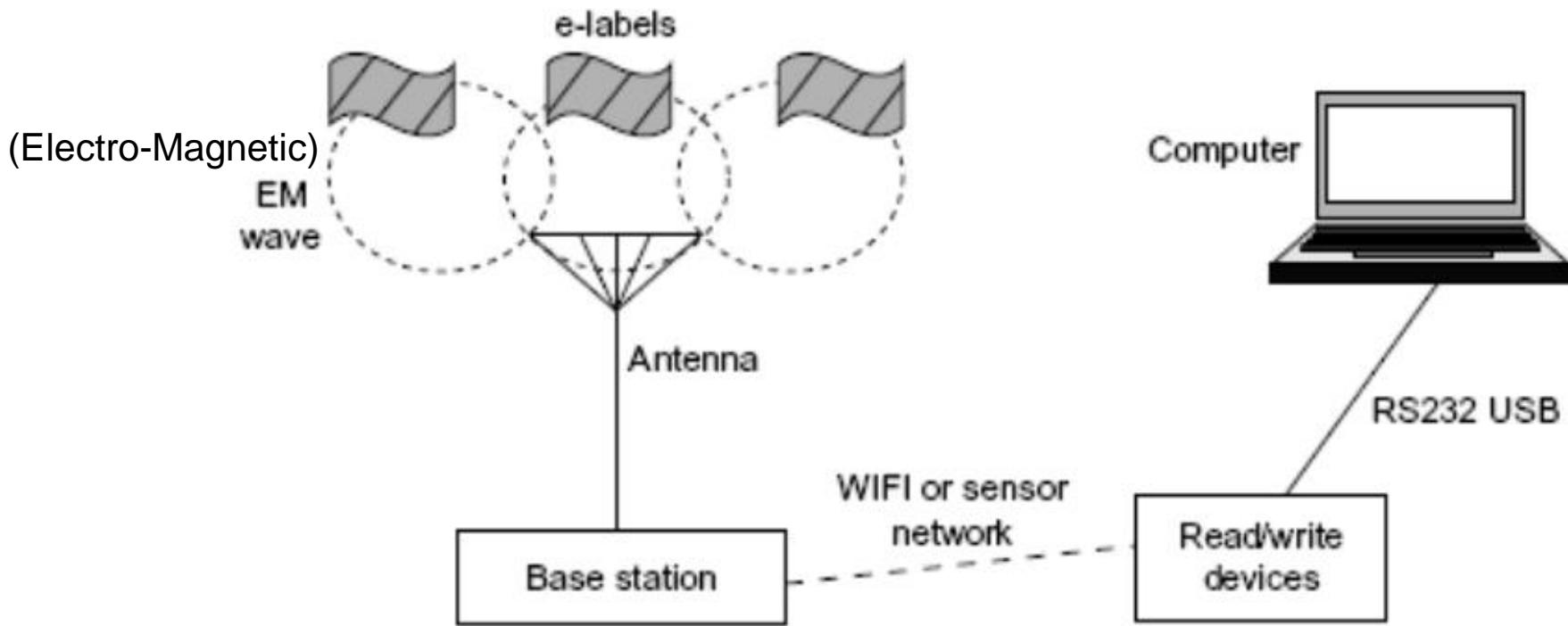


Structure of a Typical ZigBee Network.

Radio Frequency Identification Technology (RFID) Technology

A wireless system comprised of two components: **tags** and **readers**

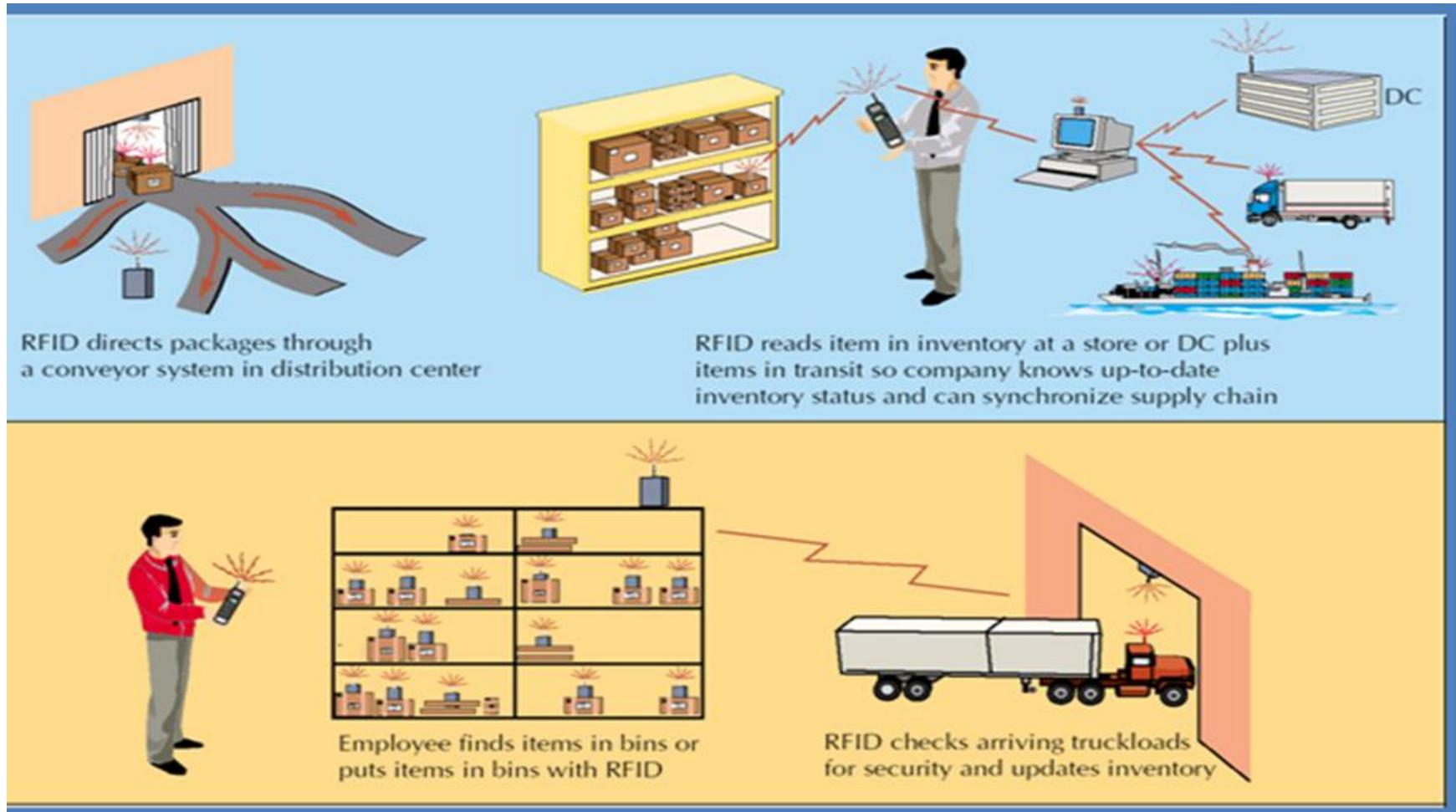
The reader is a device that has one or more antennas that emit **radio waves** and receive signals back from the RFID tag.



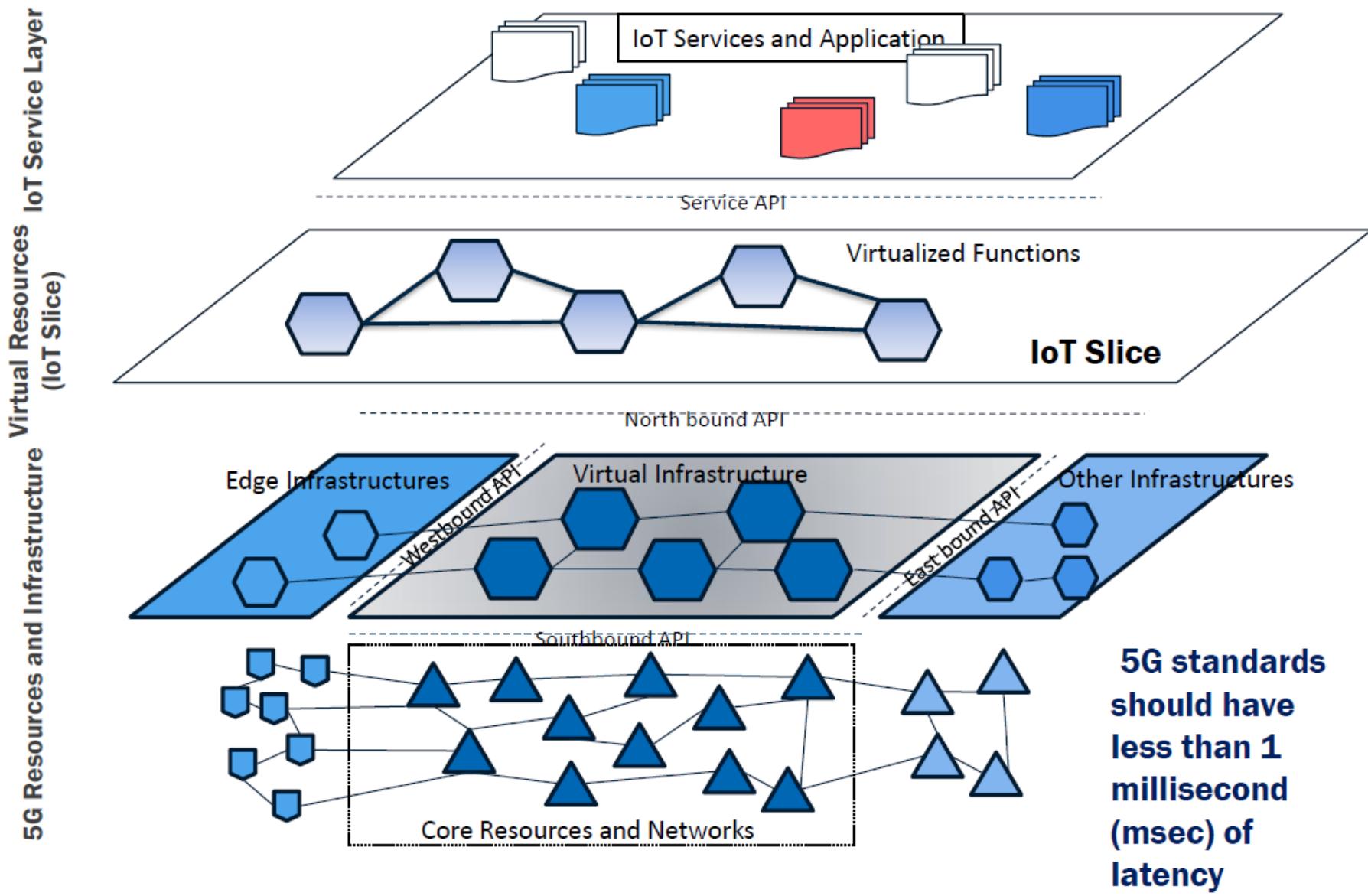
RFID tagging and signals read/write through sensor or WiFi networks.

RS232: standard communication protocol computer - devices

RFID Merchandise Tracking in a Distribution Center



5G Slicing Support for IoT



IoT: Smart Everything



Monitor vineyard conditions to prevent plant diseases



Monitor radiation levels generated by space phenomena, e.g. sun storms



Monitor radiation levels in nuclear power plants



Free parking spaces and pollution monitoring



Monitor water quality

More data, more processing ... **more clouds!**

Conclusion

- Computing clouds, IoT, and social networks affect the entire service industry and thus the future Internet evolution and global economy.
- The cloud ecosystems demand ubiquity, efficiency, security, trustworthiness, and user acceptance.
- Clouds are crucial to shape the future of Internet
 - IoT and social networks are becoming a common practice in business, government, education, entertainment, and many more.

References

1. K. Hwang, G. Fox, and J. Dongarra, Distributed and Cloud Computing: from Parallel Processing to the Internet of Things Morgan Kauffmann Publishers, 2011
2. Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions, Future Generation Computer Systems, Volume 29, Issue 7, 2013, pages 1645-1660

□ Libelium Case Studies – Connecting Sensors to the Cloud
<http://www.libelium.com/case-studies/>

COMP5123M – Cloud Computing Systems



Edge Computing

Plan of the Lecture

Goals

- Have an appreciation of what edge computing is and how it supports cloud computing

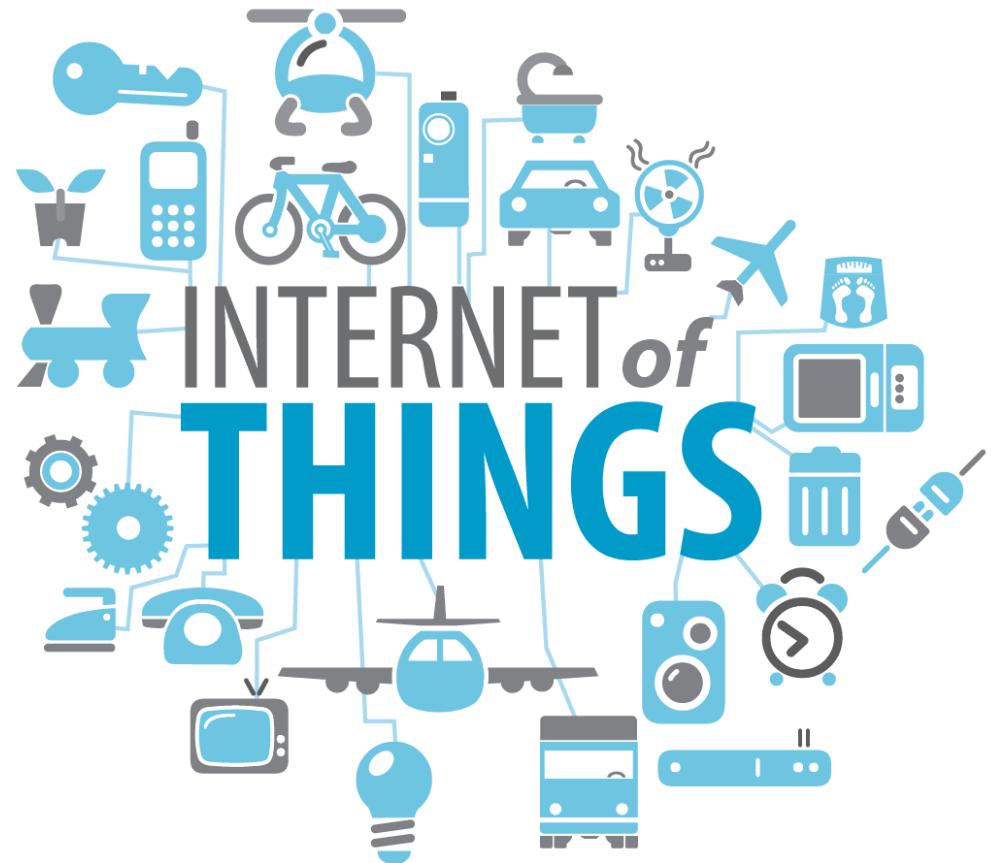
Overview

- Recap on IoT – Examples
- IoT and Industry 4.0
- Fog Computing
- Edge Computing
- Actors and Views
- Applications
- Some Challenges
 - Hardware
 - AI at the edge
 - Carbon neutrality
- Conclusion

Disruptive Applications: New Requirements

- IoT applications and devices continue to proliferate
- The old “data warehouse” model cannot
 - keep up with the data volume and velocity created by IoT devices
 - meet the **low latency** response times that users demand.
- Sending the data to the cloud for analysis also poses a risk of
 - data bottlenecks
 - security concerns
- Increase in **network traffic** causes the problem of data congestion
- New business models need **data analytics** in a minute or less (with some use cases of even less than a second)

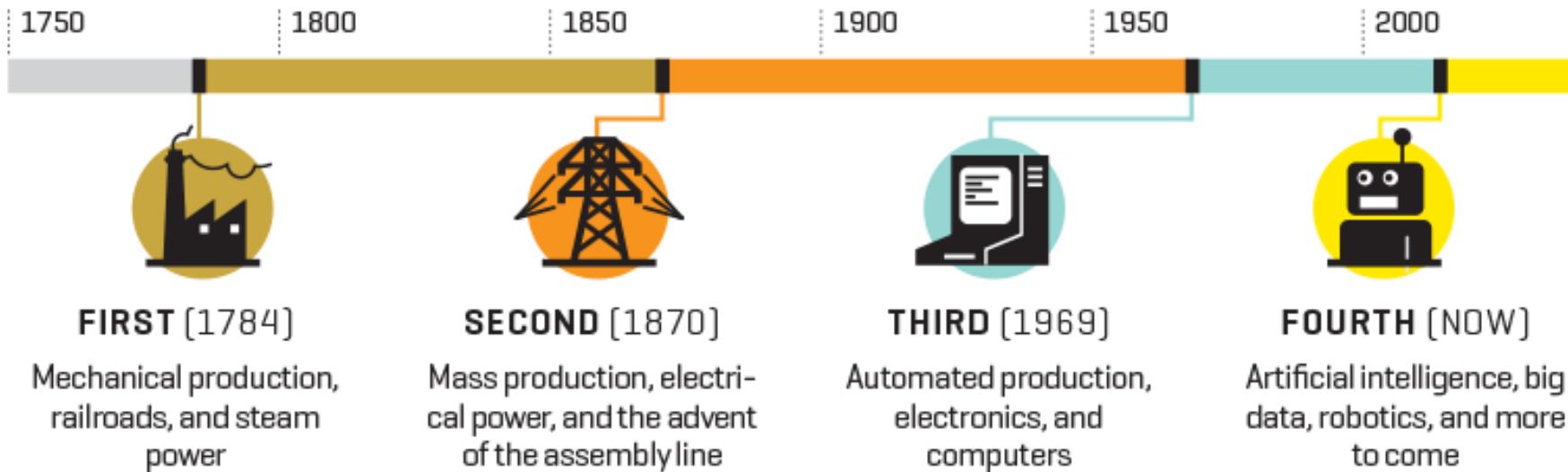
- Internet of Things: consumer based
 - Low-cost end-point devices
- Industrial IoT: enterprise focussed
 - High-cost industrial assets





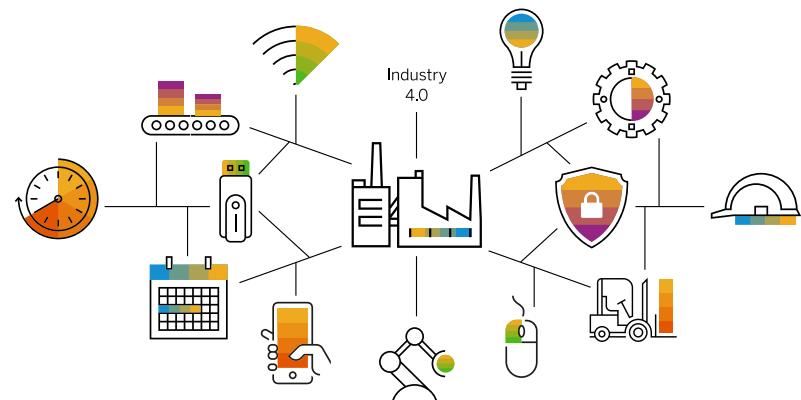
1. The Industrial Revolution – mechanised production, steam power
2. 2nd Industrial Revolution – mass production, electrification
3. Internet Revolution – automation, electronics & IT
4. Industry 4.0 – (I)IoT, digital integration, Big Data, AI, etc.

THE FOUR INDUSTRIAL REVOLUTIONS



Industry 4.0 / 5.0

- Industry in which **computers and automation will come together** in an entirely new way
 - E.g. with robotics connected remotely to computer systems equipped with machine learning algorithms that can learn and control the robotics with very little input from human operators.
- Industry 4.0 introduces what has been called the “**smart factory**,” in which Cyber-Physical Systems
 - monitor the physical processes of the factory
 - make decentralised decisions
- The physical systems become **Internet of Things**, communicating and cooperating both with each other and with humans in real time via the wireless Web.



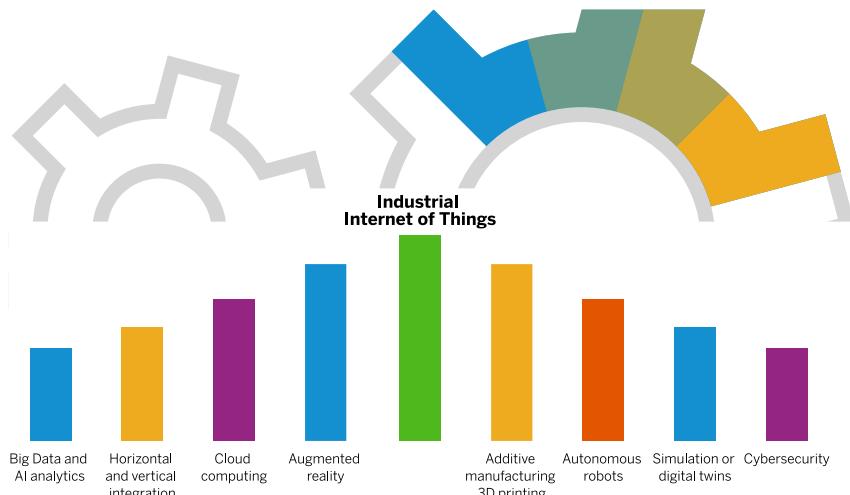
Courtesy of SAP

The term **Industry 5.0** refers to people working with smart machines and robots. These help humans work faster by leveraging advanced technologies such as big data analytics.

Industry 4.0

Six Design Principles

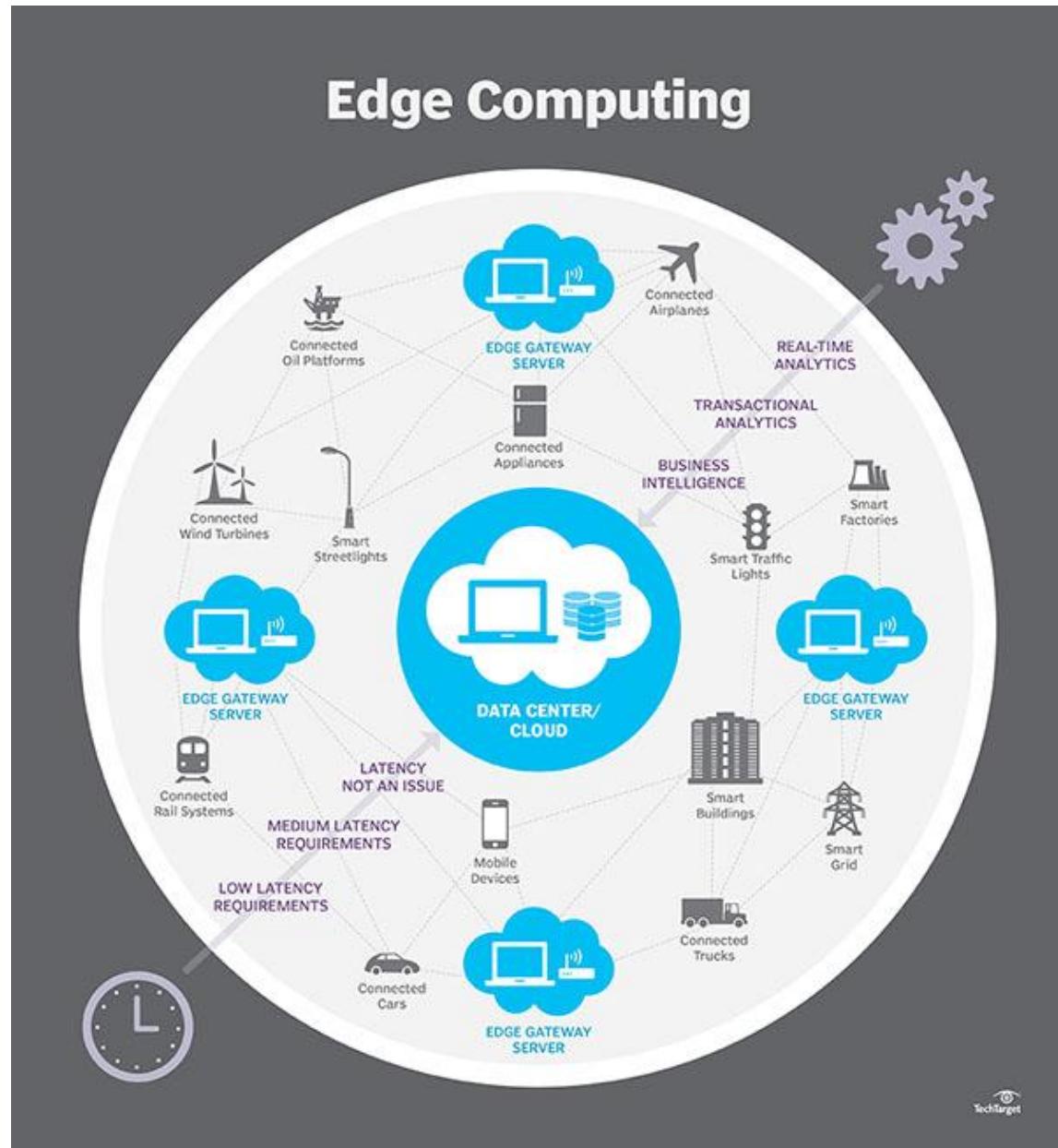
- **Interoperability:** the ability of **cyber-physical systems** (i.e. work piece carriers, assembly stations and products), humans and Smart Factories to connect and communicate with each other via the **Internet of Things**
- **Virtualisation:** a virtual copy of the Smart Factory which is created by linking sensor data (from monitoring physical processes) with virtual plant models and simulation models
- **Decentralization:** the ability of **cyber-physical systems** within Smart Factories to make decisions on their own
- **Real-Time Capability:** the capability to collect and analyse data and provide the insights immediately
- **Service Orientation:** offering of services (of **cyber-physical systems**, humans and Smart Factories) via the **Internet of Services**
- **Modularity:** flexible adaptation of Smart Factories for changing requirements of individual modules.



Courtesy of SAP

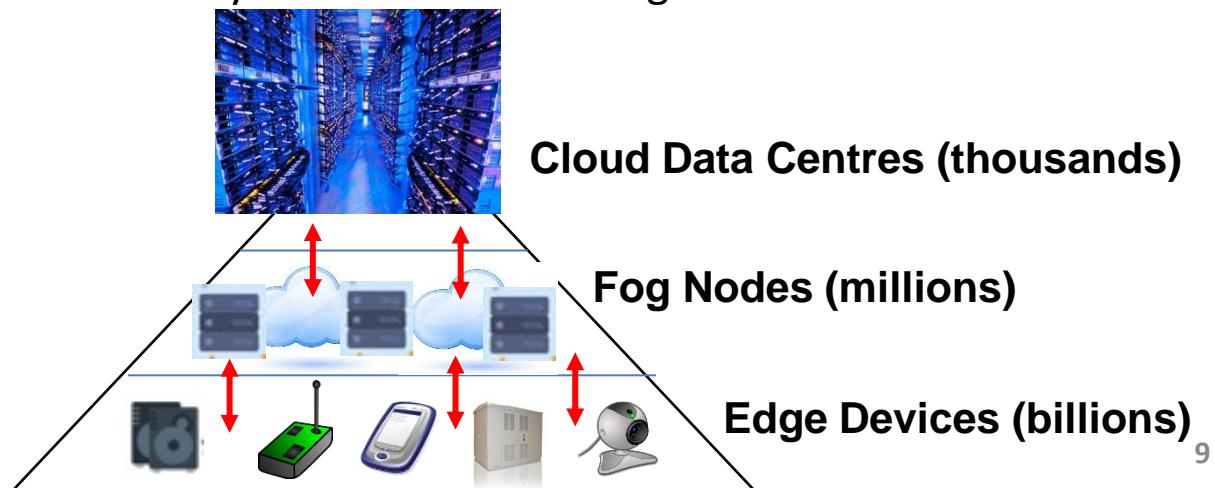
IoT brings together:

- Cloud computing
 - Data-centre based
 - Typically accessed via TCP/IP
 - Massively scalable using virtualised infrastructure
- Edge computing
 - Geographically localised servers
 - Brings computing closer to end devices
- Fog computing
 - Unites Cloud & Edge
 - Uses decentralised computing across both

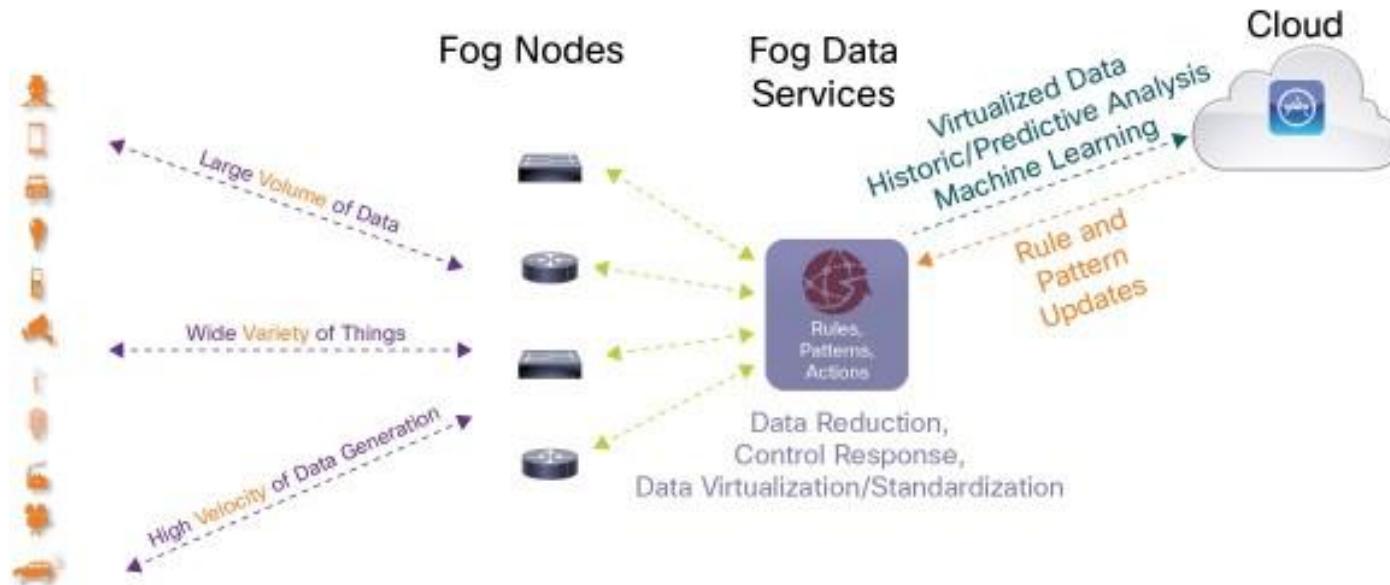


Fog / Edge Computing

- Many in industry use the terms **fog** computing and edge computing (or **edge** processing) interchangeably
- Both fog computing and edge computing involve pushing intelligence and processing capabilities down **closer to where the data originates** from pumps, motors, sensors, relays ...
- Key difference between the two architectures is exactly where that intelligence and computing power is placed
 - **Fog**: pushes intelligence down to the local area network level of network architecture, processing data in a fog node or IoT gateway.
 - **Edge**: pushes the intelligence, processing power and communication capabilities of an edge gateway or appliance directly into devices like Programmable Automation Controllers (PACs).



Fog Computing – Cisco Architecture



Fog platform: dense computational architecture at the network's edge.

Characteristics:

- Low latency
- Location awareness
- Use of wireless access.

Example:

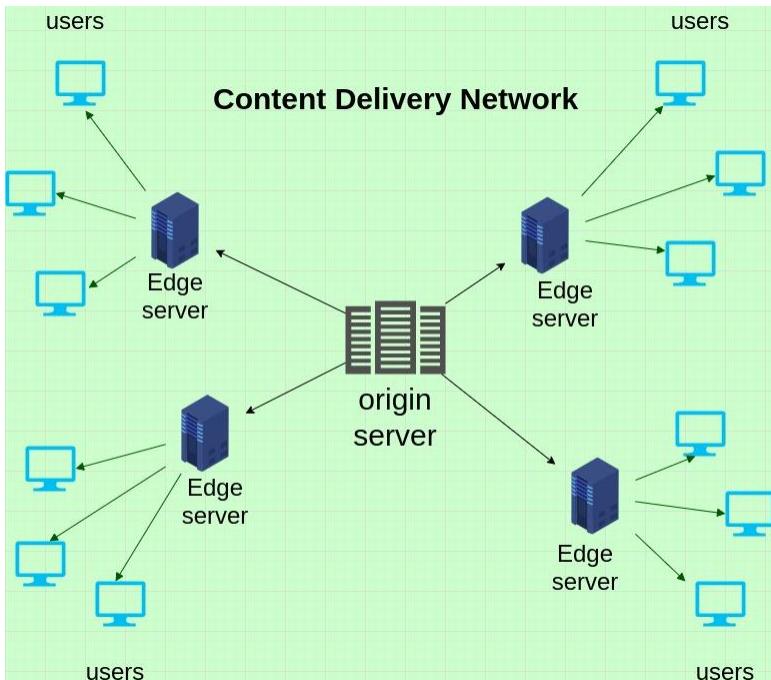
- smart traffic light system, which can change its signals based on surveillance of incoming traffic to prevent accidents or reduce congestion
- Data could be sent to the cloud for longer-term analytics.

Benefits:

- Real-time analytics
- Improved security.

Edge Computing

- An early example is Akamai, with servers around the world to distribute web site content from locations close to the user (**Content Delivery Networks**, or CDNs)



Content Delivery Network is a system of distributed networks that deliver pages and other web content to a user, based on the geographic location of the requesting user.

- The closer the CDN server is to the user geographically, as a result, the faster the content will be delivered to the requesting user.

Edge Computing - Drivers

Latency

- data processing close to where it originates avoids round-trip time to the cloud

Bandwidth

- optimisation of communication to and from the cloud

Privacy/security

- sensitive data stays local

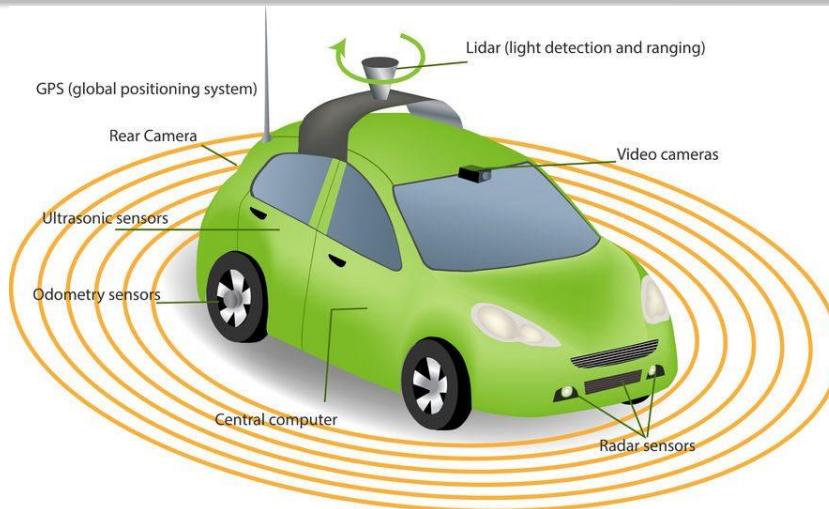
Connectivity

- continued processing (in some cases) despite lack of connectivity to the cloud

Local dependencies

- data processing close to points of interaction with end users and other system components

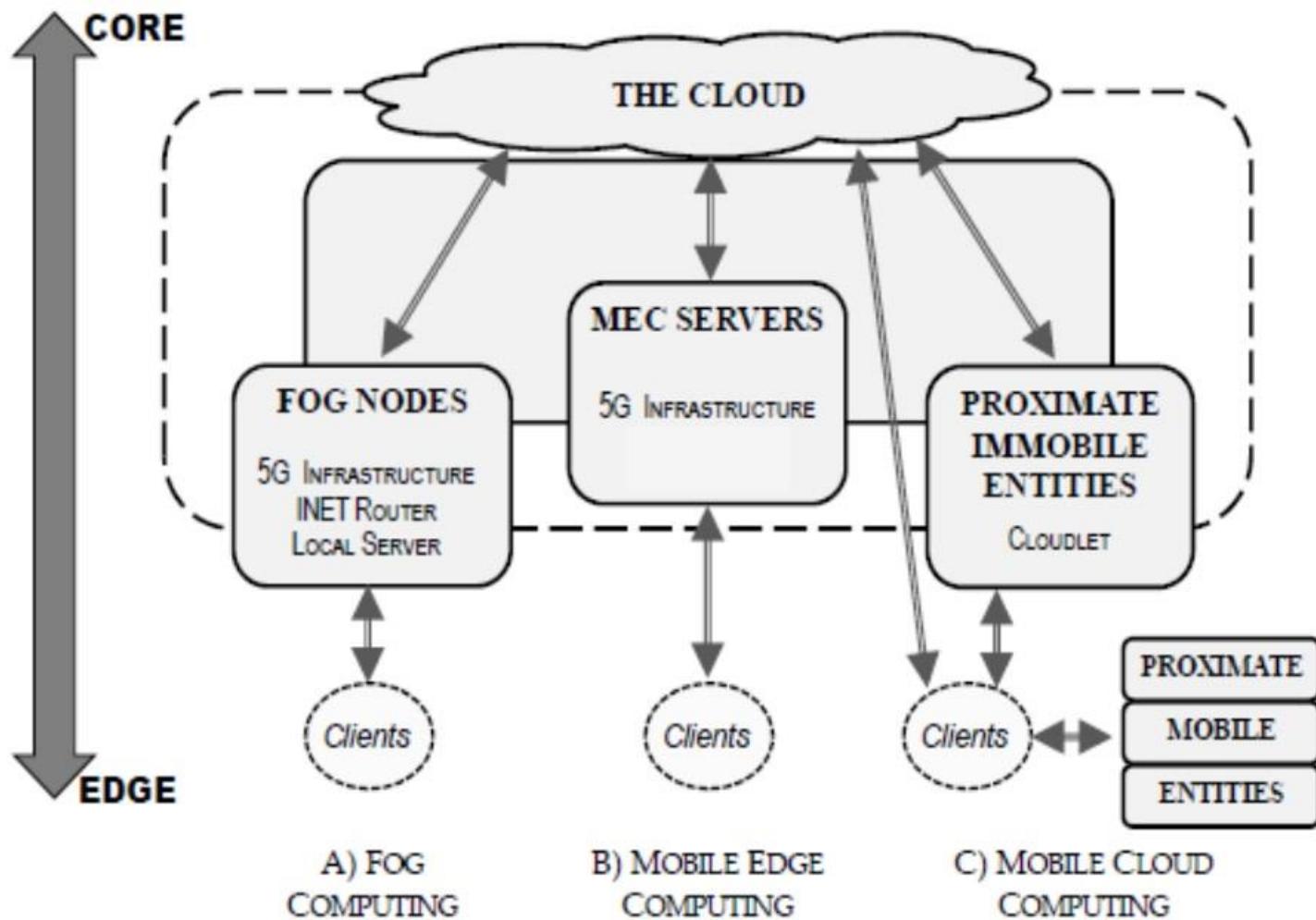
Example: Driverless Cars



Courtesy of dreamstime.com

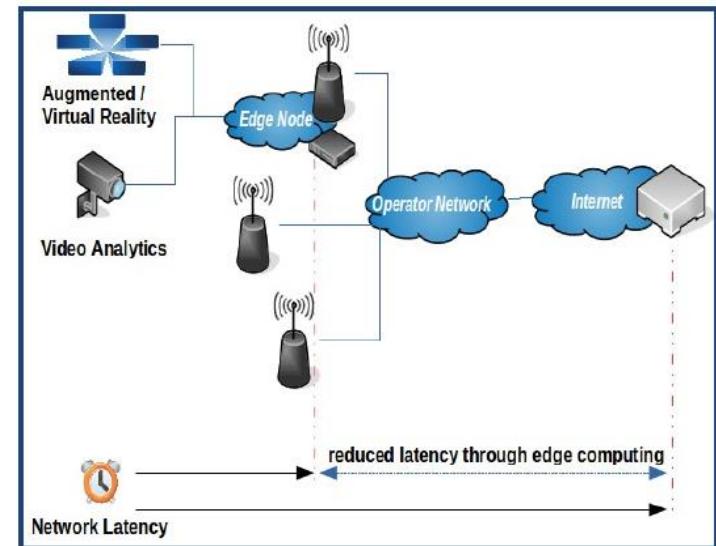
- A combination of enormous amounts of sensor data, critical local processing power
- Essential need to connect back to more advanced data analysis tools in the cloud
- Appearance of entirely new types of distributed computing architectures that can break up large workloads across different elements
 - big data analytics
 - real-time applications
 - ...
- Coordination of several different layers of computing, including pieces that live out on the **edge**.

(Mobile) Edge Computing



Edge Computing – The Telco View

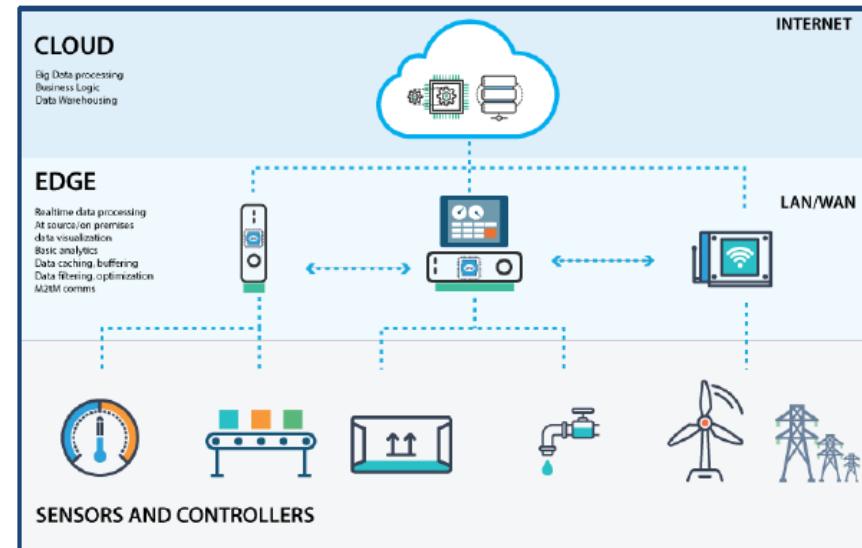
- Opportunity for providing edge computing devices in existing infrastructure, e.g.
 - micro data centers at the base of cellular towers
- Multiple organisations seeking standardization, e.g.
 - Multi-Access Edge Computing (MEC), Open Edge Computing (OEC)
- Business model is not clear: Who pays for the service? Consumer? Content Provider?



Edge Computing – The Cloud Provider View

Goal is mainly to provide

- Content Delivery Network (**CDN**) services
- IoT data processing and aggregation for data in transit to the cloud



Examples

- **Azure IoT Edge** —deploy business logic to edge devices and monitor from the cloud <https://azure.microsoft.com/en-gb/products/iot-edge/>
- **Amazon** <https://aws.amazon.com/iot/solutions/iot-edge/>
 - AWS CloudFront - CDN Service, includes Lambda@Edge
 - AWS Greengrass - connected IoT devices can run AWS Lambda functions and other code on locally-collected data

Edge Computing – The Appliance View

- Goal is to provide a “data center in a box” to push cloud computing capabilities to the edge
- Often combined with networking capabilities such as edge gateways and smart routers
- Many players in this space, such as Amazon, Cisco, Dell, HPE

Disconnected Operations

AWS Snowball Edge — large-scale data transfer service with an embedded computing platform (based on AWS Greengrass plus Lambda functions)



<https://docs.aws.amazon.com/snowball/latest/developer-guide/using-device.html>

Computation and Data in Edge Environments

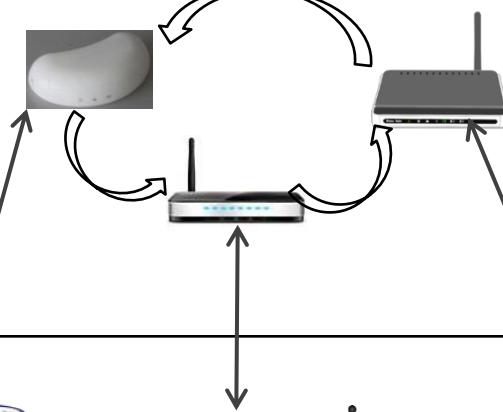
Data Center
(Cloud)



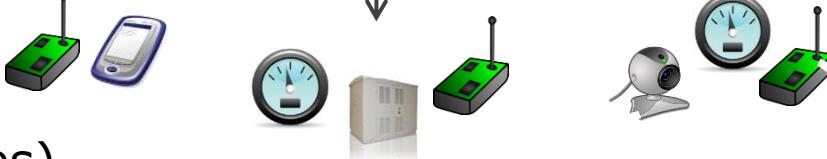
Network Core



Multi-service Edge
(Gateways,
Edge routers)



Embedded Systems &
Sensors
(M2M devices)



Providing computation-intensive capabilities and data at the edge when there is no access to the cloud

- Speech recognition
- Face recognition
- Speech translation
- Image recognition
- Image processing
- Air/water quality analysis

Network-edge computing

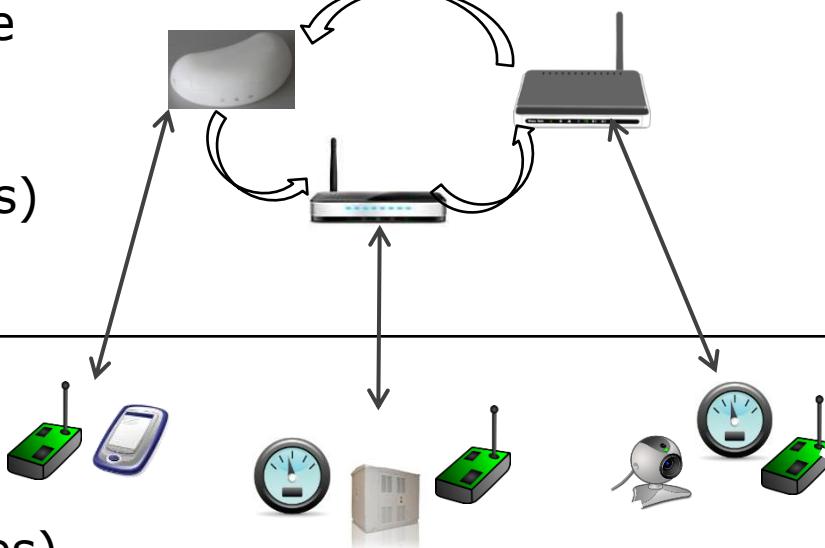
Data Center
(Cloud)



Network Core



Multi-service Edge
(Gateways,
Edge routers)



Embedded Systems &
Sensors
(M2M devices)

- Data is
- Monitored
 - Analysed
 - Reduced
 - Cached

Network-edge computing

Data Center
(Cloud)

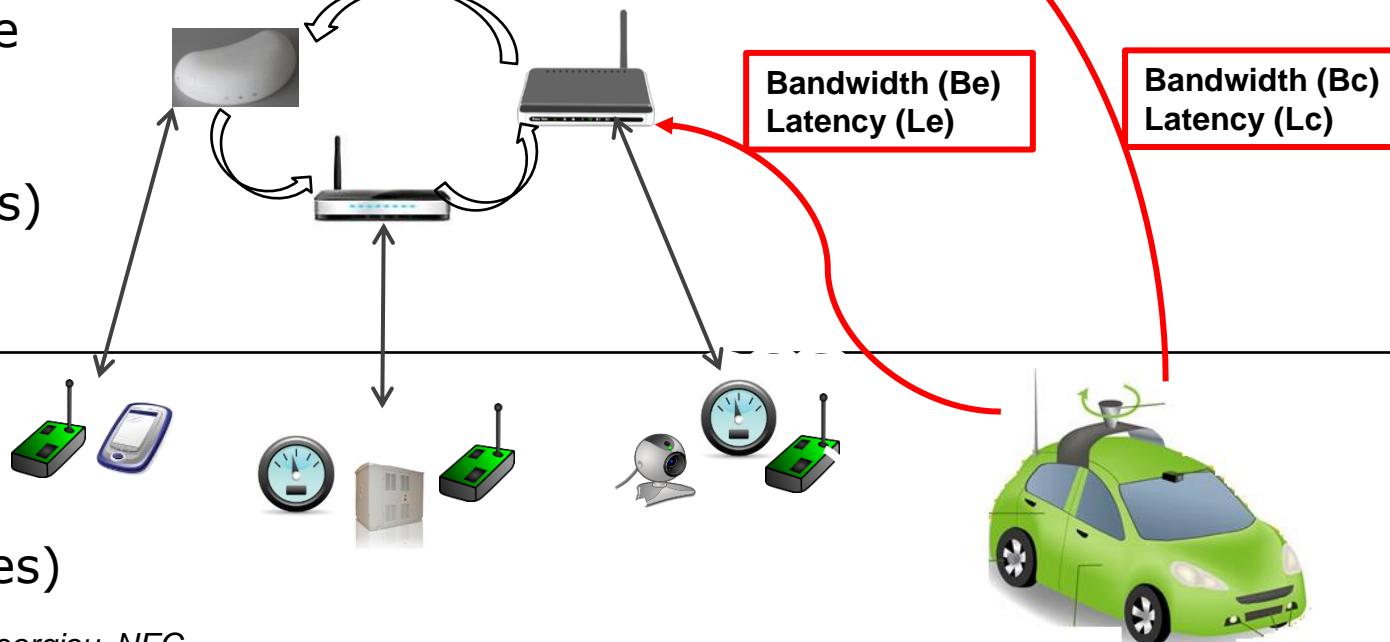


- Data is
- Monitored
 - Analysed
 - Reduced
 - Cached

Network
Core



Multi-service
Edge
(Gateways,
Edge routers)



Summary

- Reviewed directions cloud computing is taking, especially in the context of IoT and edge computing
- Edge Computing is about pushing applications, data and computing power to the edge of the Internet, in close proximity to mobile devices, sensors, and end users
- Edge Computing via “appliances” can provide computation and data to support a wide variety of missions and applications.

References

- A Manifesto for Future Generation Cloud Computing: Research Directions for the Next Decade. R. Buyya et al., ArXiv, 2017 <https://arxiv.org/abs/1711.09123>
- Fog and Edge Computing: Principles and Paradigms (Wiley Series on Parallel and Distributed Computing), Rajkumar Buyya (Editor), 2019
- Edge-centric Computing: Vision and Challenges. P. Garcia Lopez et al. SIGCOMM Comput. Commun. Rev. October 2015, Vol. 45, No. 5, pp. 37-42

COMP5123M – Cloud Computing Systems



Edge Computing *Part II*

Plan of the Lecture

Goals

- Have an appreciation of what edge computing is and how it supports cloud computing

Overview

- Recap on IoT – Examples
- IoT and Industry 4.0
- Fog Computing
- Edge Computing
- Actors and Views
- Applications
- Some Challenges
 - Hardware
 - AI at the edge
 - Carbon neutrality
- Conclusion

Network-edge computing

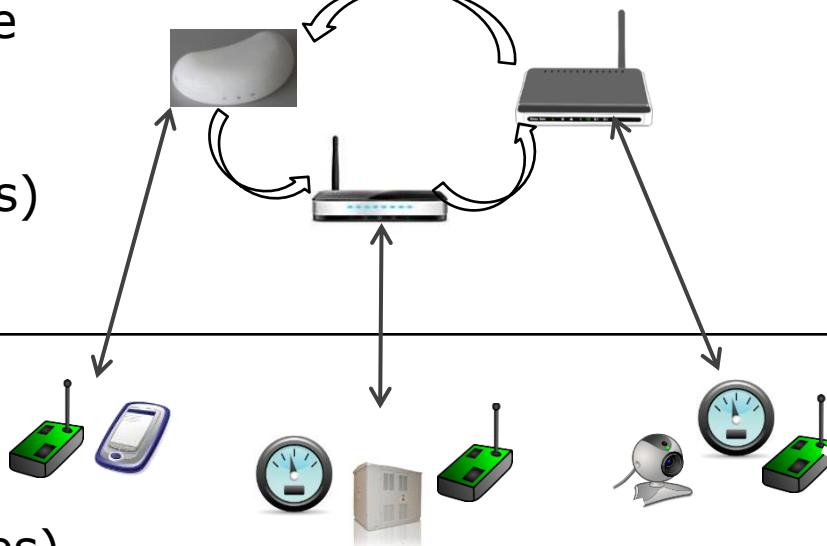
Data Center
(Cloud)



Network Core



Multi-service Edge
(Gateways,
Edge routers)



- Data is
- Monitored
 - Analysed
 - Reduced
 - Cached

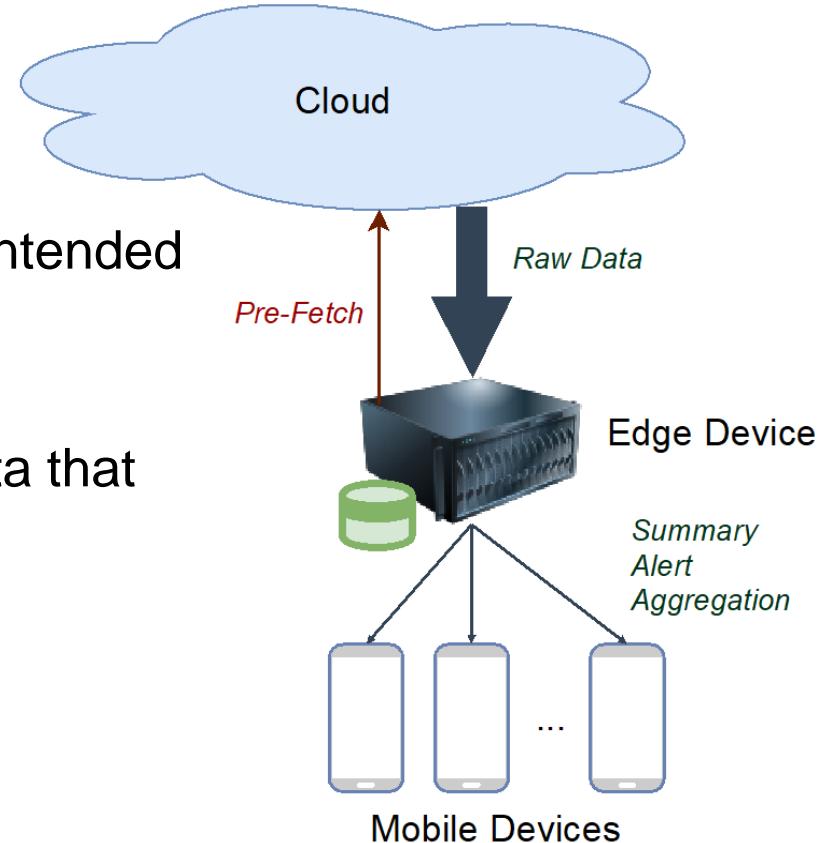
Data Pre-Processing, Filtering, and Pre-Fetching (Cloud to Edge)

Using edge devices to

- pre-process,
- pre-fetch, or
- filter unnecessary data from streams intended for mobile devices

Goal: Mobile devices receive only the data that they need, when they need it

- reduced bandwidth
- reduced latency
- reduced load

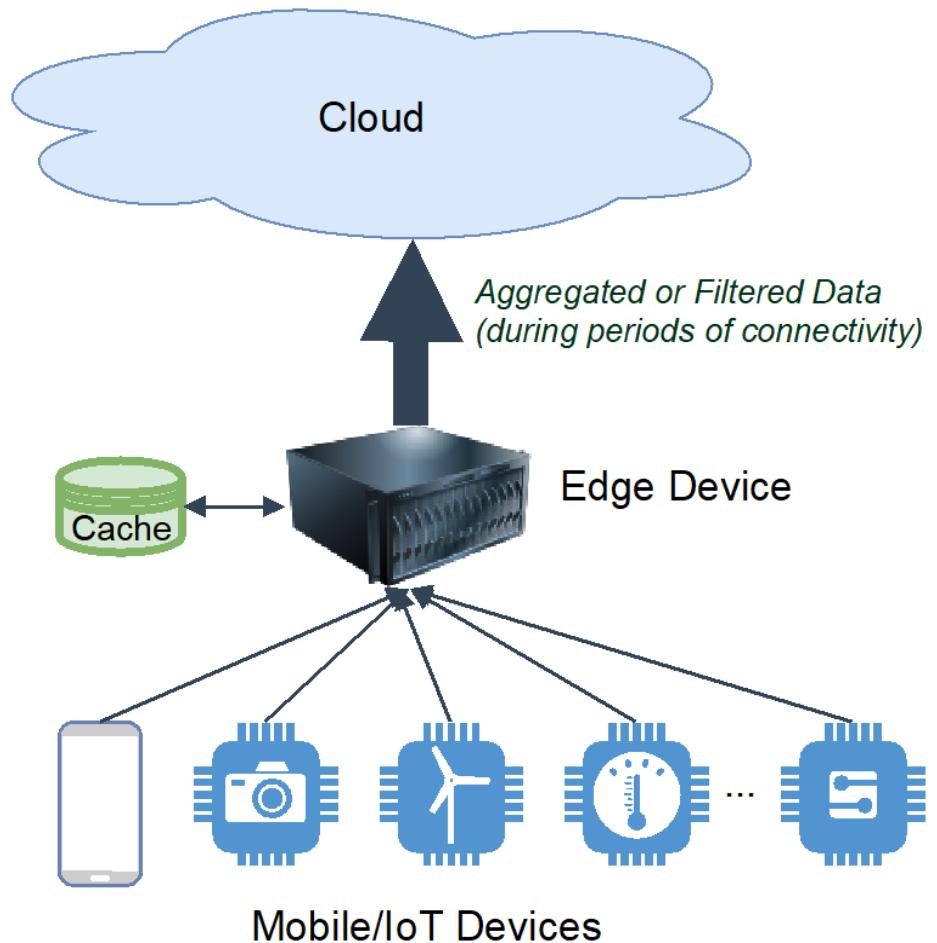


Data Pre-Processing and Caching (Edge to Cloud)

Using edge devices to

- pre-process, or
- cache

data heading for enterprise
repositories



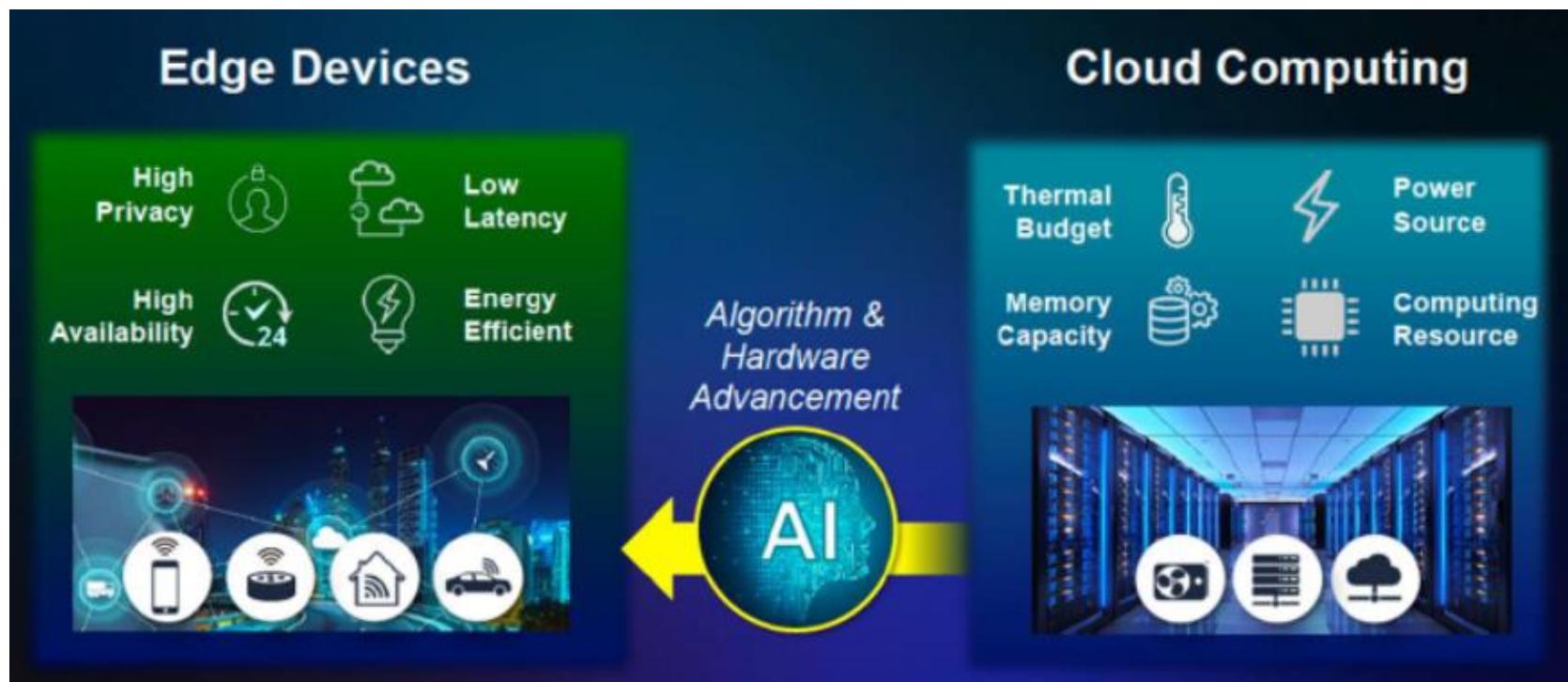
Challenges

- Resource management
- Resource discovery
- Data and computation allocation to edge devices
- Privacy
- Security
- Mobility
- Billing and accounting
- Energy efficiency and carbon neutrality
- Hardware
 - Especially in the context of Edge AI (see next slide)
 - Google Edge TPU: purpose-built ASIC designed to run inference at the edge.
<https://cloud.google.com/edge-tpu>



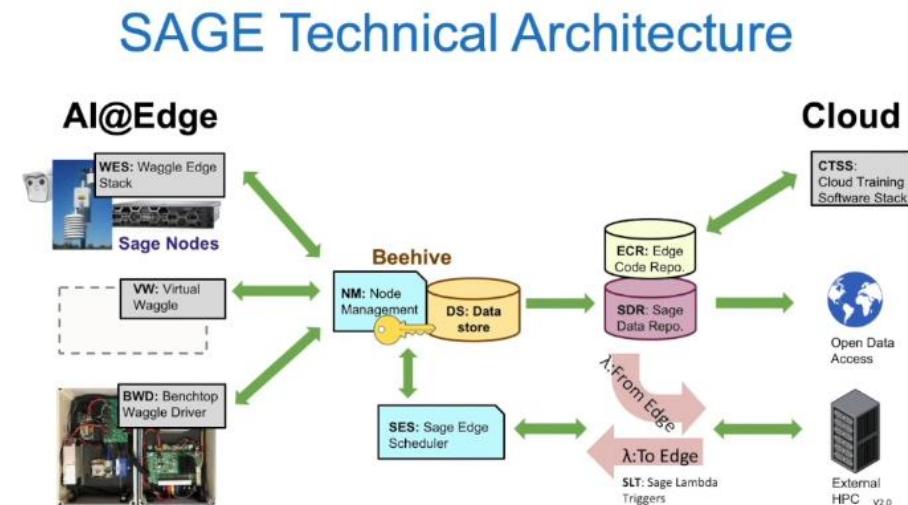
Artificial Intelligence at the Edge

- AI is becoming more pervasive, from consumer to enterprise applications, e.g. IoT
- There is an explosive growth of connected devices, combined with a demand for privacy/confidentiality, low latency, and bandwidth constraints
- **AI models** trained in the cloud increasingly need to be run at the edge.



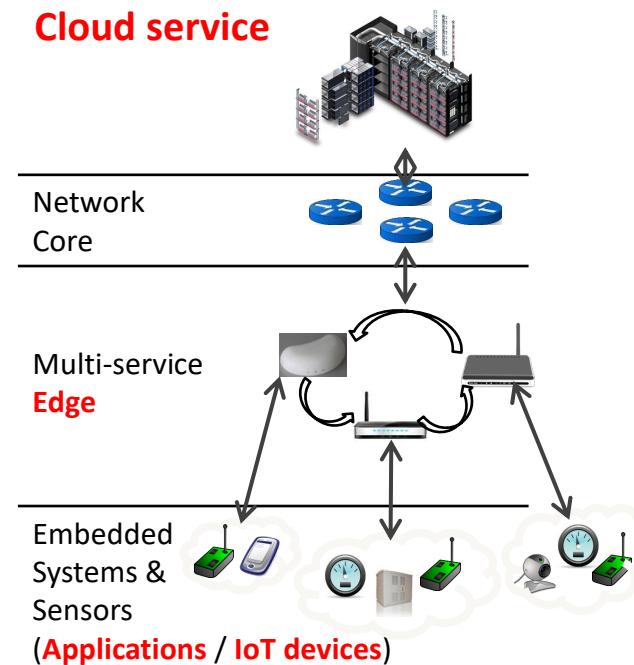
Edge-Driven Future for Computing - Example

- **Waggle:** Scientific AI at the Edge Computing <https://wa8.gl/>
 - AI-enabled edge computing platform that crunches incoming data using an Nvidia Xavier NX GPU
- **SAGE:** <https://sagecontinuum.org/>
Cyberinfrastructure for AI at the Edge
- Software-Defined Sensor Network
<https://www.anl.gov/mcs/sage-a-softwaredefined-sensor-network>
 - Waggle edge sensors used in networks to build software-defined sensors



Billing Structure (1)

- Assumption: application, edge, cloud
 - Cloud service provider
 - Edge computing service provider
- **Note:** both providers may belong to the same entity
- Monitoring of edge computing resources, accounting, and billing are necessary to
 - ensure satisfactory **QoS (Quality of Service)**
 - properly **charging** the user/application for the offered services by the edge computing service provider
- Monitoring, accounting, and billing require a sustainable **business model** for the edge computing service providers
- Designing such a business model is **challenging** because of
 - Scope of the service, e.g. short versus long service usage
 - Mobile nature of the user



Billing Structure (2)

Scope of the service,
e.g.

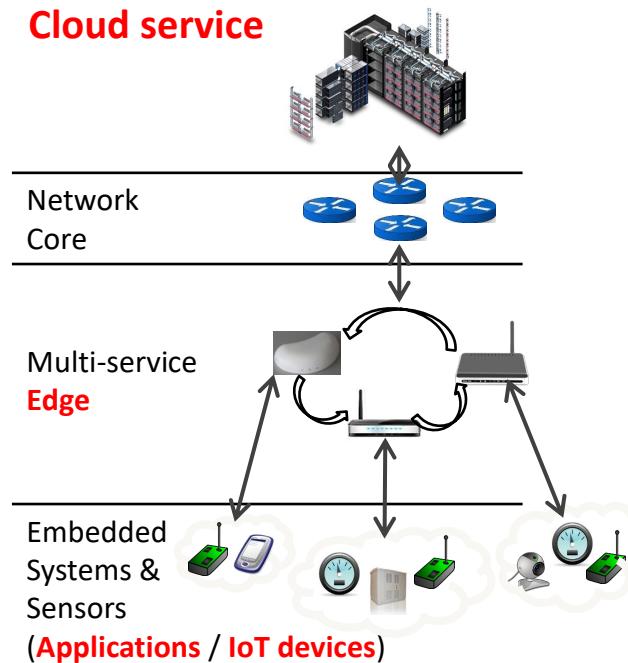
- Type of resource consumption
 - Compute, storage, network
- Short versus long service usage

State information

- ◊ What?
- ◊ Where?
- ◊ When?
- ◊ How much?
- ◊ How many?

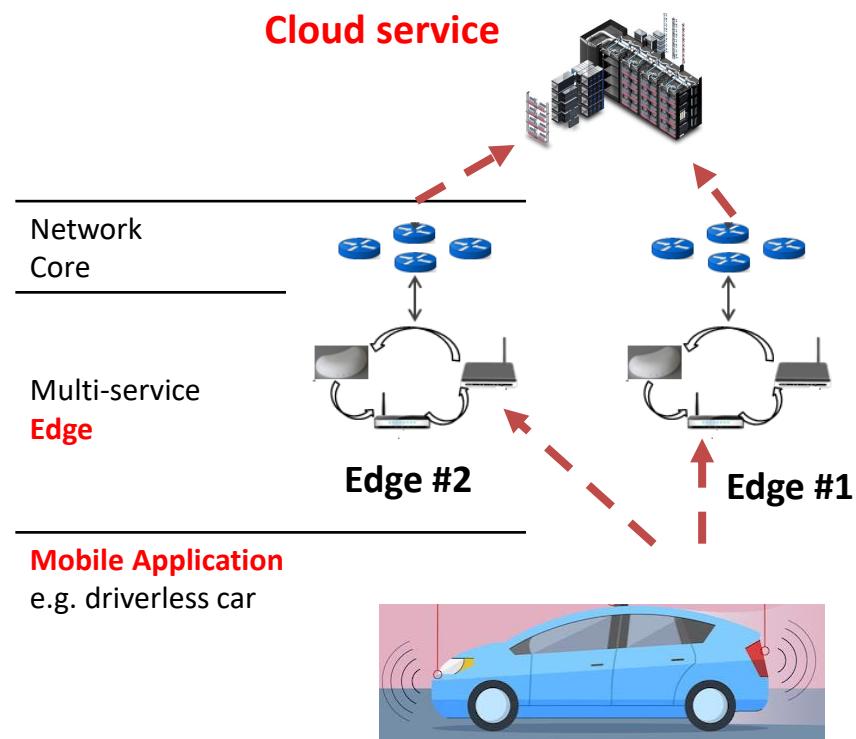
State information

- ◊ What?
- ◊ Where?
- ◊ When?
- ◊ How much?
- ◊ How many?



Billing Structure (3)

- Mobile nature of the user
 - a mobile user is facilitated through the provider's roaming Edge services
 - when the user moves and execution migrates from one edge platform to another, the division of the charges among the involved edge computing service providers raises an **accounting / billing challenge**



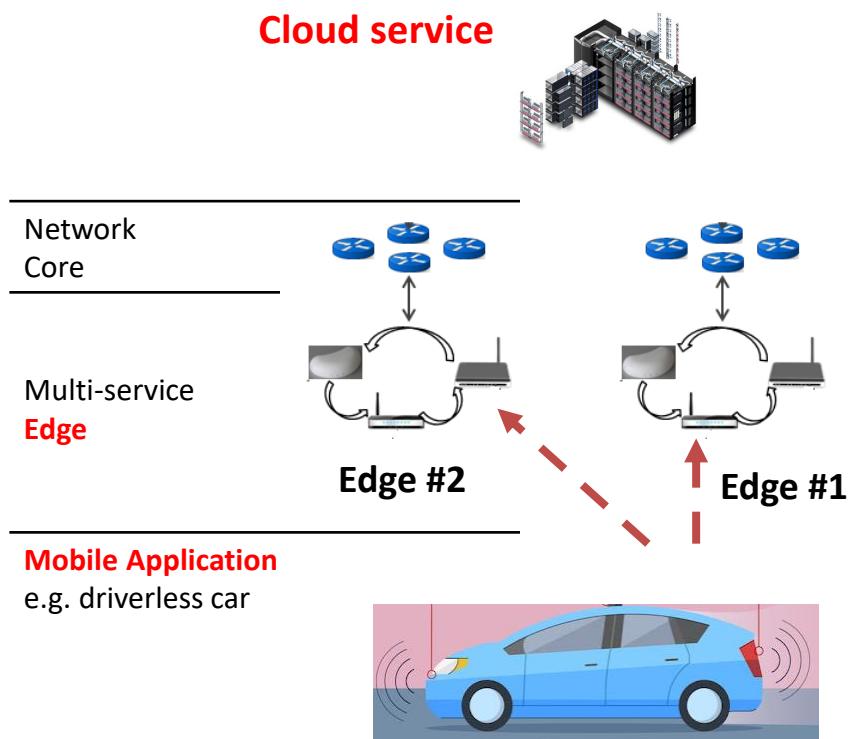
Dynamic Billing

A number of different mobile users may request from the cloud through the Edge based systems

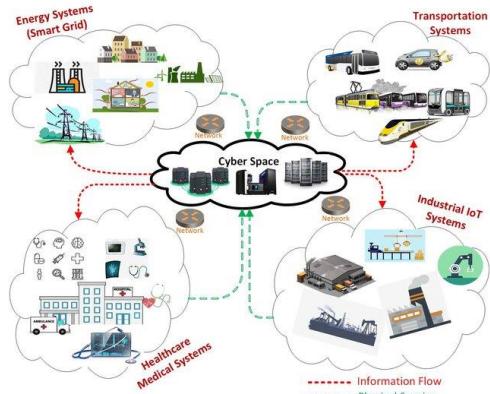
- specific resources (e.g., CPU, memory requirements)
- with various network parameters (e.g., bandwidth requirement and availability for data transmission, latency, delay and level of security as per application requirements etc.)

Three factors considered while developing the dynamic billing mechanism

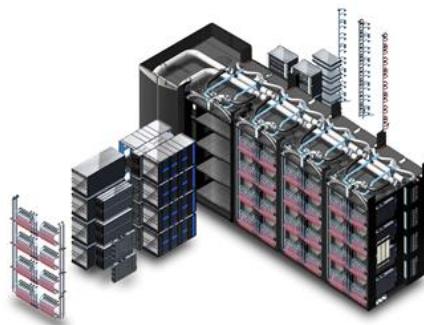
1. **resource availability** e.g. billing charges may vary in case of more users presence which will result in high resources demand
2. **frequency of resource usage** e.g., how frequently a particular user use resources
3. **duration of resource usage** e.g., charges may differ for users with long duration resource usage as compared to users with short duration resources usage due-to management overheads.



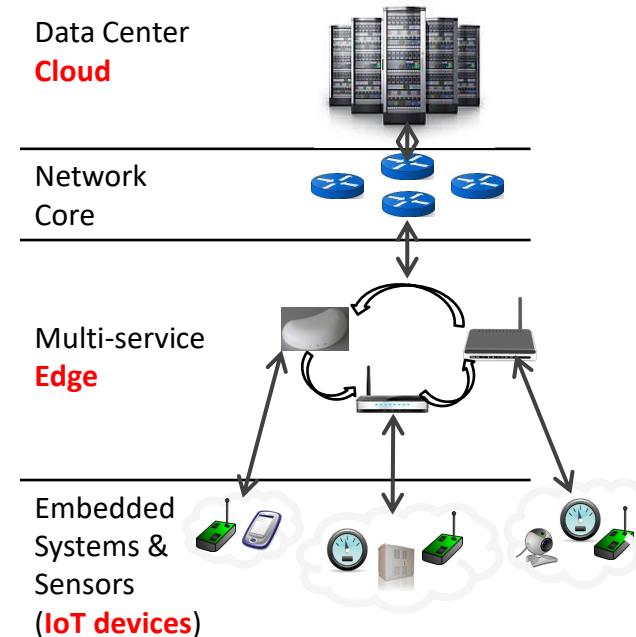
Edge-Driven Future for Supercomputing



New **Disruptive** Applications



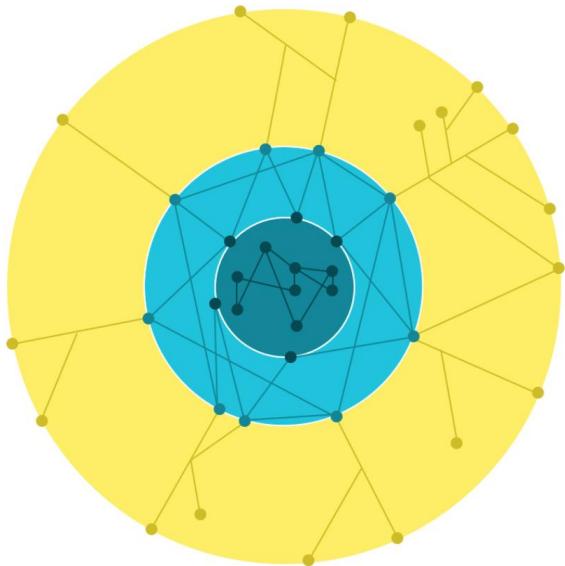
High Performance
Exascale Computing



- Edge computing is a crucial part of delivering value for exascale
 - is seen as a device for turning an **I/O-bound** problem into a **compute-bound** problem
- It is more efficient to examine data at the edge and filter only the important or interesting data to supercomputers for heavy-duty analysis.

Edge Infrastructures: On Carbon Neutrality

Google



Google's network infrastructure has three distinct elements:

- Core data centers
- Edge Points of Presence (PoPs)
- Edge caching and services nodes (Google Global Cache, or GGC)

Source: <https://peering.google.com/#/infrastructure>



Watson IoT



AWS CloudFront — Content Delivery Network Service, includes Lambda@Edge
AWS Greengrass — connected IoT devices can run AWS Lambda functions and other code on locally-collected data

On Carbon Neutrality in Edge-Driven Future for Computing

- Lots of hype regarding carbon neutrality!
- Lesson learnt through an example: Google
 - 21 hyperscale data centres around the globe
 - Subsidised renewable energy projects
 - The 100% **carbon neutral claim** only covers Google's data centre, not the energy caused their products as soon as the data packets have left the data centre
 - To bring data closer to high traffic areas (e.g. high bandwidth/low latency video content), it uses
 - smaller **Edge Points of Presence** (PoP) data centres
 - Third-party **Content Delivery Networks** (CDN)



These **ARE NOT COVERED** by the carbon neutrality claim.

Carbon Neutrality: Take Way Message

In an Edge / Cloud computing environment that is carbon neutral, Edge Computing is yet other devices on the edge of the network, consuming resources, increasing the need for **bandwidth**, increasing **power consumption** and the need to roll out **more network infrastructure**.

Solving the problem caused by technology with technology simply **shifts** the carbon neutrality issue elsewhere ...

Summary

- Reviewed directions cloud computing is taking, especially in the context of IoT and edge computing
- Edge Computing is about pushing applications, data and computing power to the edge of the Internet, in close proximity to mobile devices, sensors, and end users
- Edge Computing via “appliances” can provide computation and data to support a wide variety of missions and applications.

References

- A Manifesto for Future Generation Cloud Computing: Research Directions for the Next Decade. R. Buyya et al., ArXiv, 2017 <https://arxiv.org/abs/1711.09123>
- Fog and Edge Computing: Principles and Paradigms (Wiley Series on Parallel and Distributed Computing), Rajkumar Buyya (Editor), 2019
- Edge-centric Computing: Vision and Challenges. P. Garcia Lopez et al. SIGCOMM Comput. Commun. Rev. October 2015, Vol. 45, No. 5, pp. 37-42

COMP5123M – Cloud Computing Systems



On Disaggregation

Plan of the Lecture

Goals

- Have an appreciation of what disaggregation supports cloud computing

Overview

- Introduction
- Disaggregation in Data Centre
- Software Defined Networks
- Conclusion

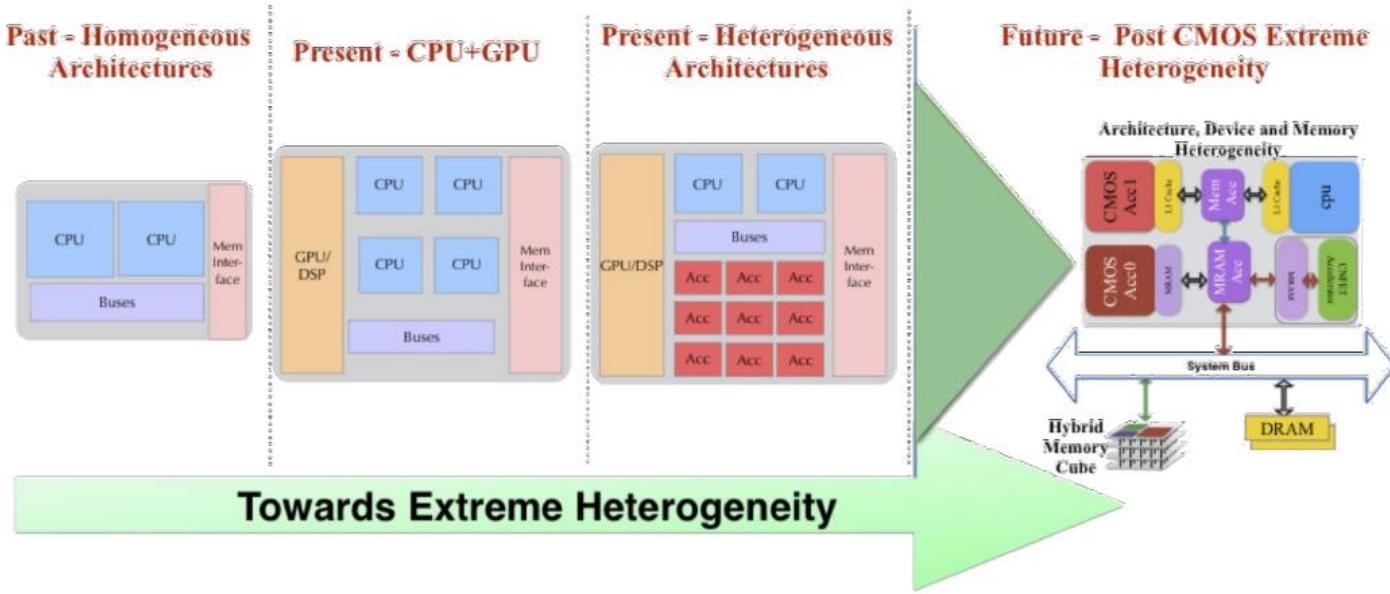
Future Computing / Extreme Heterogeneity

A good start to the subject of **extreme heterogeneity** is:

- Extreme Heterogeneity 2018. Productive Computational Science in the Era of Extreme Heterogeneity. DOE ASCR Basic Research Needs Workshop on Extreme Heterogeneity. J.S. Vetter et al. " US Department of Energy, Office of Science, Advanced Scientific Computing Research, 2018, doi:10.2172/1473756.

<https://orau.gov/exheterogeneity2018/2018-Extreme-Heterogeneity-BRN-report-final.pdf>

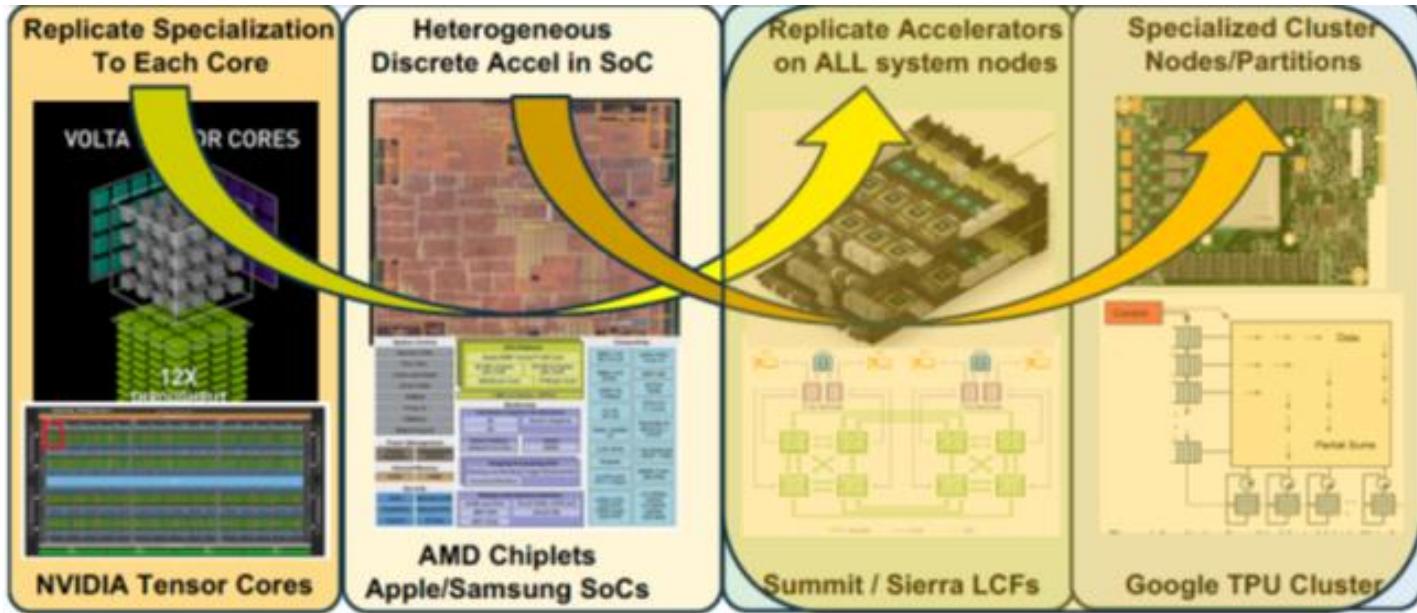
Future Direction of Computing



- Classical technological driver that has underpinned **Moore's law** is already flattening
- Consequence: many forms of **heterogeneous accelerators** (no longer just GPU accelerators)
- Extracting continued performance improvements is to use transistors more efficiently by **specialising the architecture** to the target scientific **problem/application**.

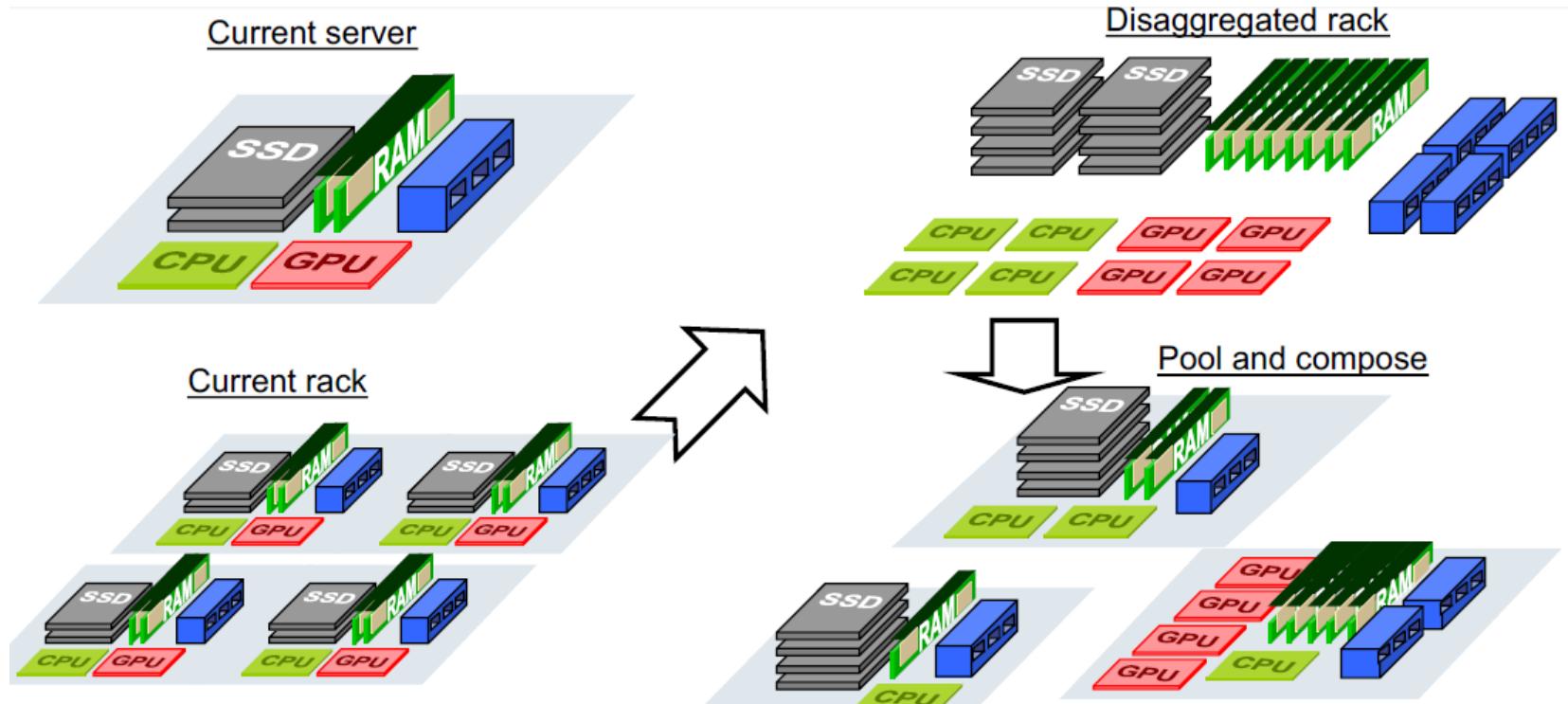
Courtesy of J. Shalf, PADAL'19

How Do We Arrange the Specialisations?



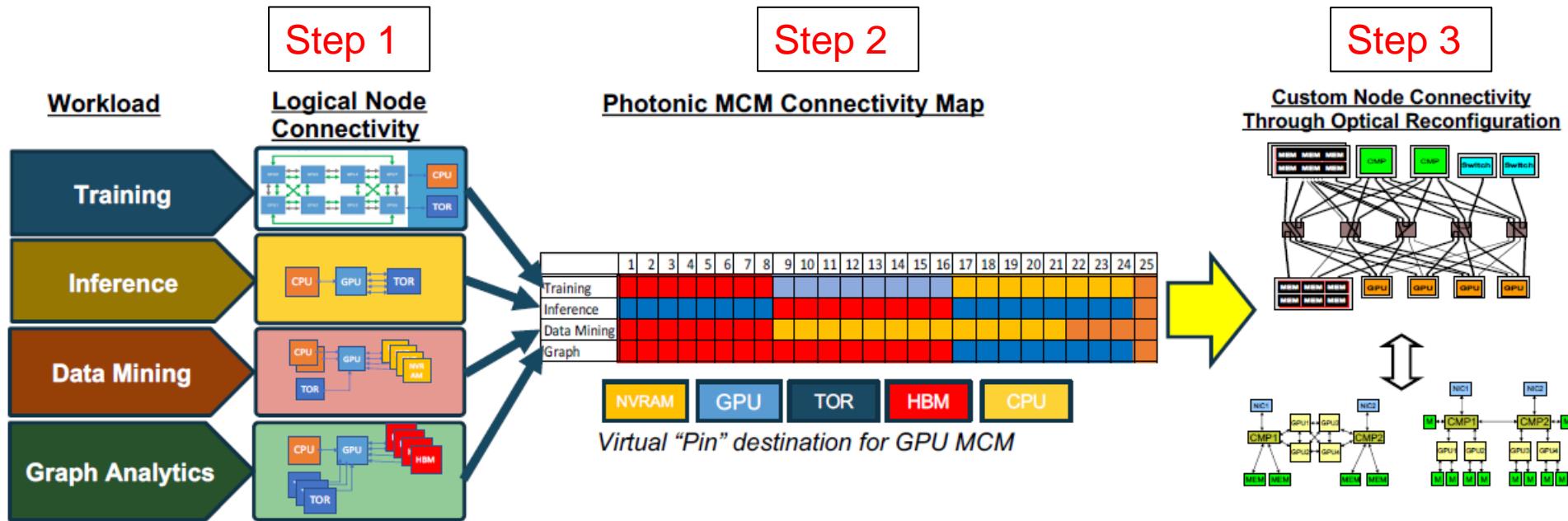
Examples of Supercomputers

Disaggregated Node / Rack Architecture



Most current disaggregation solutions use interconnect bandwidth (1 – 10 GB/s)
But this is significantly inferior to RAM bandwidth (100 GB/s – 1 TB/s)

Case for Disaggregation from a “Workload” Perspective

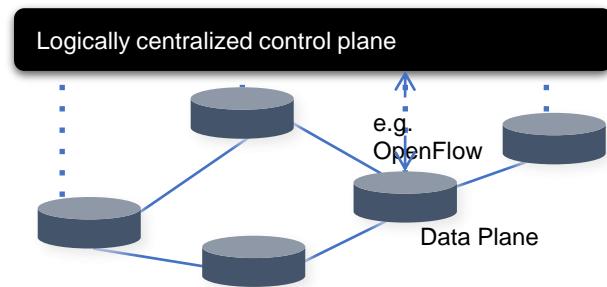


To mitigate data movement costs, reliance on new computing architectures and advanced packaging technologies such as

- monolithic three-dimensional integration (building chips in the third dimension)
- Photonic co-packaging.

Software Defined Networks –Definition

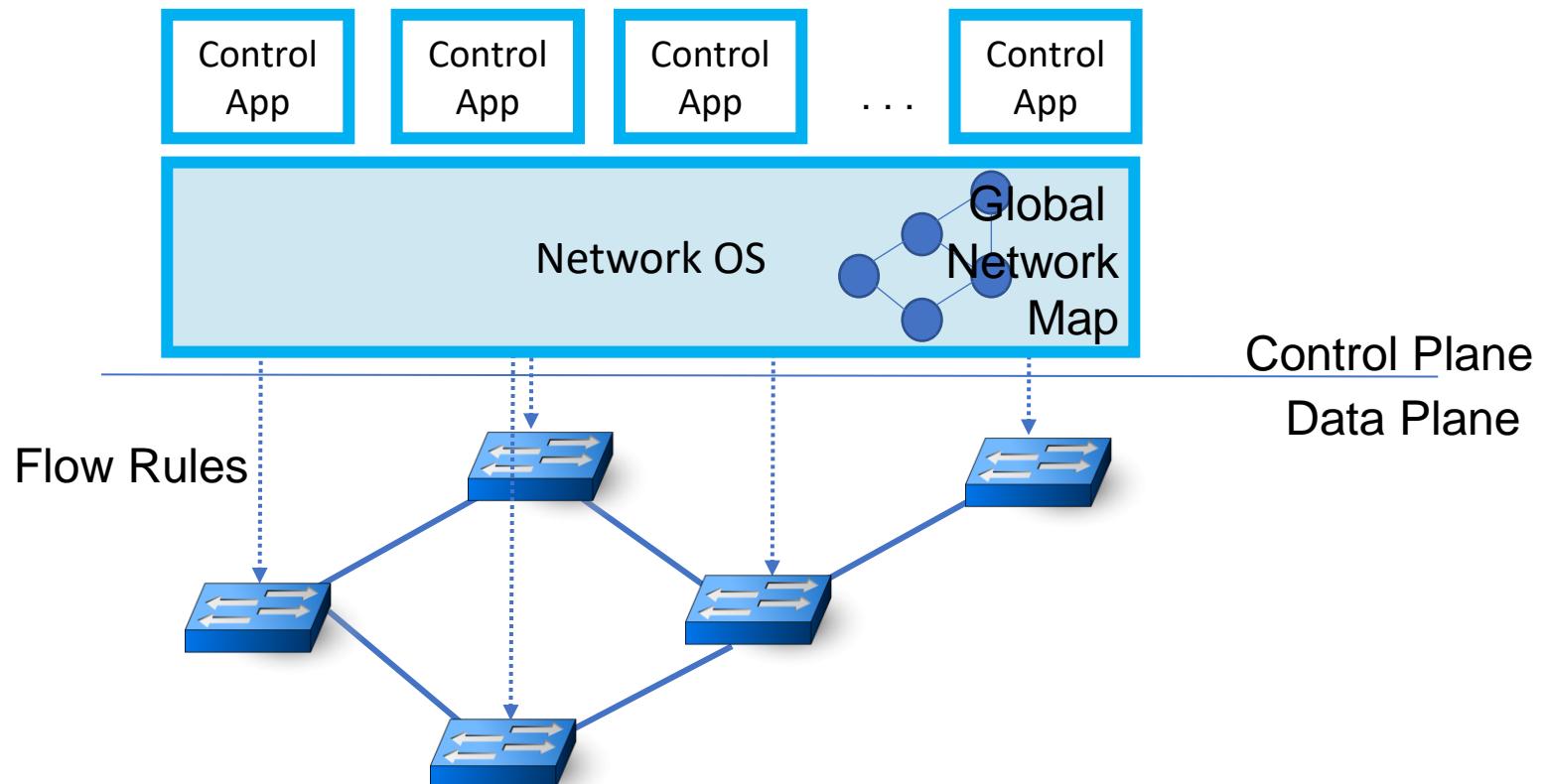
- SDN (software-defined networking) is the separation of **control** and **data** planes
- The separation allows control topology to be independent of physical network topology
- Question is:
 - Why would anyone want to do this?



Key SDN Insights

- Centralizing the control plane enables more powerful abstractions
 - E.g. X and Y should be able to communicate
 - Express intent network-wide
- Distributed systems techniques to make central control scalable and fault tolerant
- Central control means a single API for the network, rather than an API per box
- Networks provisioned by software, not humans
- Disaggregation → innovation
- Network-wide intent → better security

SDN Control and Data Planes

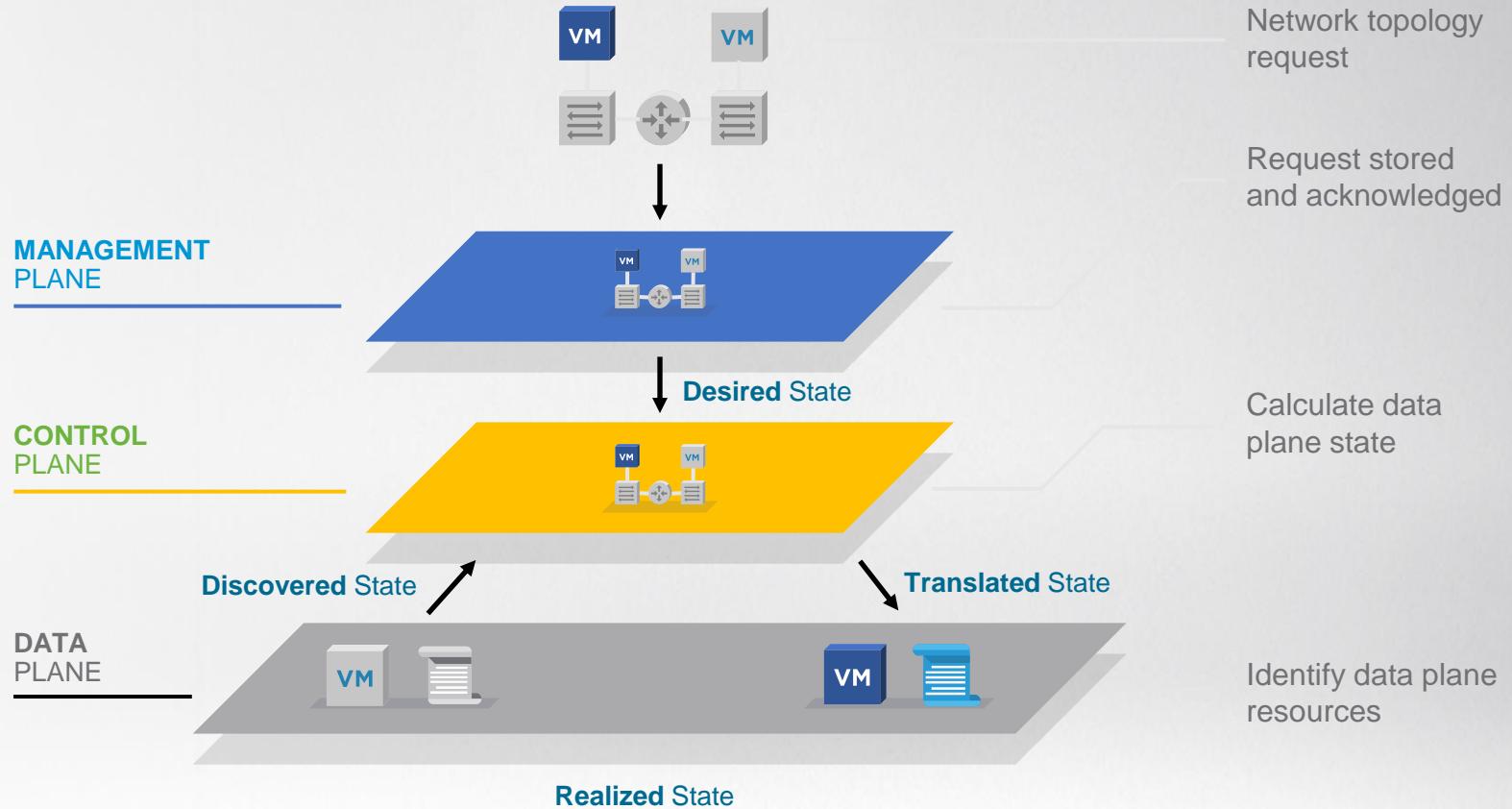


Virtual Machines to Virtual Networks

- ▶ Virtual Data Centers
- ▶ “Network hypervisor”
- ▶ Virtualization layer
- ▶ Network, storage, compute



Management, Control and Data Planes



Summary

Definition of SDN

A network in which the control plane is physically separate from the forwarding plane, and a single control plane controls several forwarding devices. – *Nick McKeown (2013)*

Dimensions

- **Disaggregated** Control and Data planes
- Centralized / Decentralized Control Plane
- Fixed-Function vs Programmable Data Plane

References

- The future of computing beyond Moore (2020). J. Shalf. Philosophical Transactions of the Royal Society A: 3782019006120190061, <http://doi.org/10.1098/rsta.2019.0061>
- dReDBox - Disaggregated Data Center in a Box. EU H2020 Project

<http://www.dredbox.eu/>

<http://www.dredbox.eu/publications/>