Cloud Computing (INFS3208)

Lecture 13: Course Review

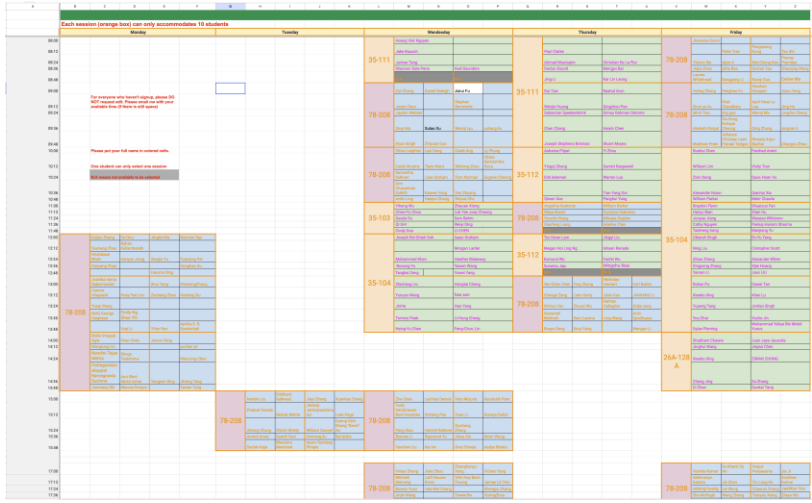
Lecturer: Dr Sen Wang

School of Electrical Engineering and Computer Science

Faculty of Engineering, Architecture and Information Technology

The University of Queensland

Individual Project Marking & Competition



Marking sessions from Monday to Friday this week.

- 360+ students have signed up for the sessions in Week 13

Approved Extensions:

- Case 1: New due date is before this Friday – find out if there exist available marking sessions (by contacting tutor Jason)
- Case 2: New due date is after this Friday OR unavailable in Week 13
  – we will move the marking to Week 14 (Time: TBA)
- Proof of Approval (Case 1 & Case 2)

Late submission (without the approval of an extension)

- 10% penalty per day after the grace period up to 7 calendar days.
- More detailed policy on extension and late submission in the course profile.
- Some exceptional cases (cannot submit twice for proposal before due)

Competition: The outcome of the competition will be available after the marking sessions (after week 14).

Weighted Voting:

- Teaching staff: 10% each
- Students: 20%

Outline

Course Revision

- Final Exam - when, how, and what!
- Topic Review

Final Exam – When and How

INFS3208 Final Exam (on-campus)

- Time: **2:30 PM AEST, 10 November 2025**
- Examination Duration: 120 minutes
- Reading Time: 10 minutes (planning)
- Closed book (No materials permitted; no calculator needed)
  - On-campus Exam: Central Paper-based Exam
- Total 50 points and **double pass** (>= 25 out of 50)

Final Exam – What

*1. **30** Multiple Choice Questions (MCQ): 15 marks (0.5 marks per question)*

Part A – Multiple choice questions (15 marks, 0.5 mark per question)
Answer all questions on the Gradescope Bubble sheet.

1. Which of the following is NOT the business driver of Cloud Computing?

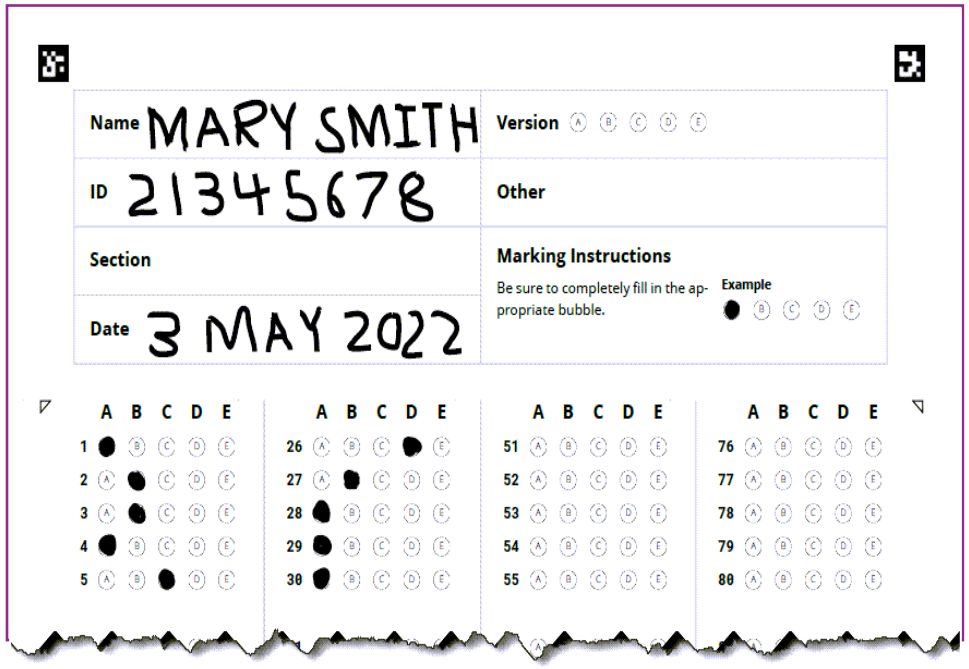
   a. Ease of use

   b. Capacity planning

   c. Cost reduction

   d. Organisational agility

Gradescope Bubble Sheet tips



1. On-campus exam will use a Gradescope Bubble Sheet
2. Enter your **FIRST NAME** & **LAST NAME** in block capitals
3. Enter your eight digit UQ student **ID**
4. Enter the date of your exam
5. Fill in the bubbles using a **2B pencil**
6. To change an answer use an **eraser**

Final Exam – What

2. Short Answer Questions: 35 marks (10 questions)

Part B – Short answer questions 35 marks)
There are 7 questions in this part. Please answer all questions in the spaces provided on this examination paper.

Question 1: (**3 marks**) What are horizontal scaling and vertical scaling methods?

Final Exam – What topics are covered?

Covered Topics will be from

- Lecture 1 to Lecture 12
- Tutorial/Practical Week 2 to Week 11

Useful preparation materials:

- Lecture Slides (Lec 13***) & Recordings
- Tutorial questions & MCQ questions (MCQs in homeworks)
- Past final exam papers

Option 1: Log in to the Library home page

Log in to see your **personalised learning resources** on the Library home page. It includes **links to past exams**, course reading lists, and relevant Library guides for specific courses. You can use the learning resources search box to find courses if yours are not listed.

This learning resources option is also available in your Library account menu at the top of each page.

Option 2: Search by course code

Use Library Search from the home page.

1. Select **Past exam papers** from the Library Search options
2. Enter a full or partial course code, such as ENGG1500 or ENGG150
3. Click **Search**.

Access a past exam paper

1. Select the link for an available exam
2. Log in with your UQ username and password if you haven't already
3. Papers will be available as a PDF or as a link to the Inspera exam paper (papers will be listed from most recent to oldest).

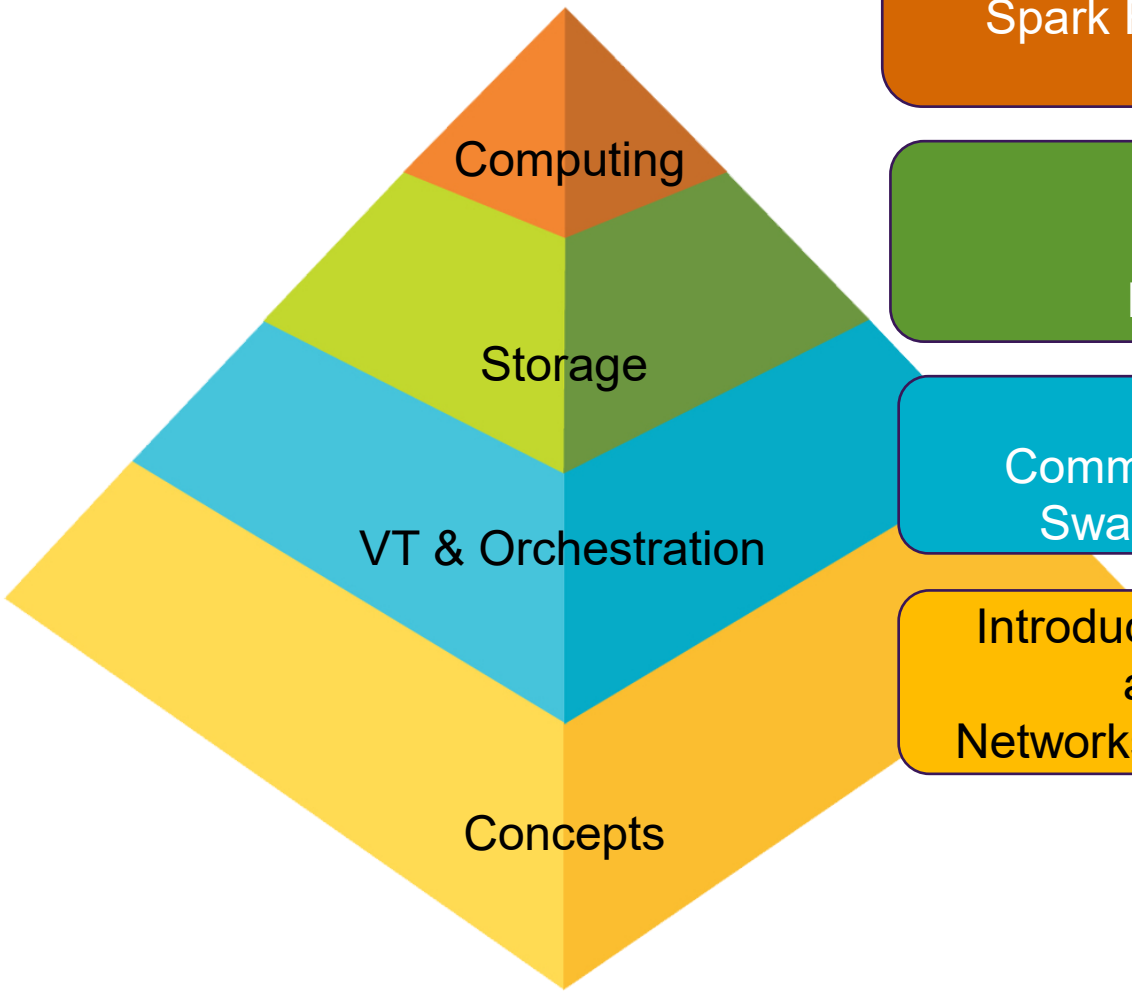
Outline

Course Revision

- Final Exam - when, how, and what!
- Topic Review

Note: the topics that are mentioned are strongly recommended to be focused.

Course Overview – Lectures



Computing:
Hadoop & MapReduce
Spark RDD, SparkSQL, SparkML

Storage:
DBs in CC
DFS in CC

VT: Docker
Commands, Compose, Swarm, Kubernetes

Introduction, adv. topics & applications
Networks & Load Balancing

Security & Privacy

Lecture 1: Introduction to Cloud Computing

ID	Topics	Pages	Lecture
1	Business Drivers	34-36	1
2	Pre-existing Technologies	38-40	1
3	Basic Concepts and Terminology in the cloud	42-43,46	1
4	Two Scaling methods	44-45	1
5	Cloud Characteristics	47-48	1
6	Cloud Delivery Models	50-57	1
7	Cloud Deploy Models	58-59	1
8	Cloud-Enabling Technology II: Virtualisation	63-67	1
9	Cloud-Enabling Technology V: Multitenancy	75-77	1
10	Goals and Benefits of Cloud Computing	79	1

Lecture 1: Introduction to Cloud Computing

History & Definitions

- Definition: Cloud computing is a specialised form of distributed computing that introduces utilisation models for remotely provisioning scalable and measured resources

Business Drivers (***ID 1, Lec1, 32-34***):

- Capacity Planning (CP), Cost Reduction, and Organisational Agility

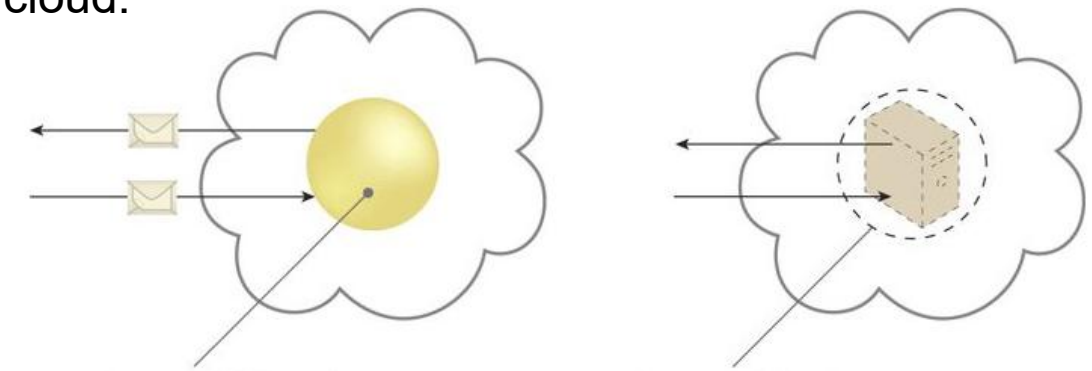
Basic Concepts and Terminology in the Cloud (***ID 3, Lec1, 40-41, 44***):

- Cloud
  - A distinct IT environment that is designed to provision scalable and measured IT resources remotely.
- Cloud vs Internet
  - Cloud has a clear and finite boundary
  - Cloud is usually private and offers metered IT resources
  - Cloud often provides back-end processing capabilities

Lecture 1: Introduction to Cloud Computing

Basic Concepts and Terminology in the Cloud (***ID 3, Lec1, 40-41, 44***):

- Cloud Service:
  - any IT resource that is made remotely accessible via a cloud.
  - Broad context:
    - a simple Web-based software program
    - a remote access point for administrative tools
    - larger environments
    - etc
  - Driving Motivation -- to provide IT resources as services
    - types of cloud services can be labelled with the "as-a-service"
    - Examples: Infrastructure-as-a-service (IaaS), PaaS, and SaaS

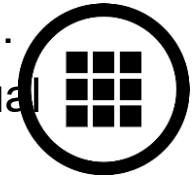


Lecture 1: Introduction to Cloud Computing

Technology Innovations (***ID 2, Lec1, 36-38***) :

- **Clustering**
  - A cluster is a group of independent IT resources that are interconnected and work as a single system.
  - Cluster nodes have reasonably identical hardware and OS

- **Grid Computing**
  - GC provides a platform in which computing resources are organized into one or more logical pools.
  - These pools are collectively coordinated to provide a high-performance distributed grid (super virtual computer).

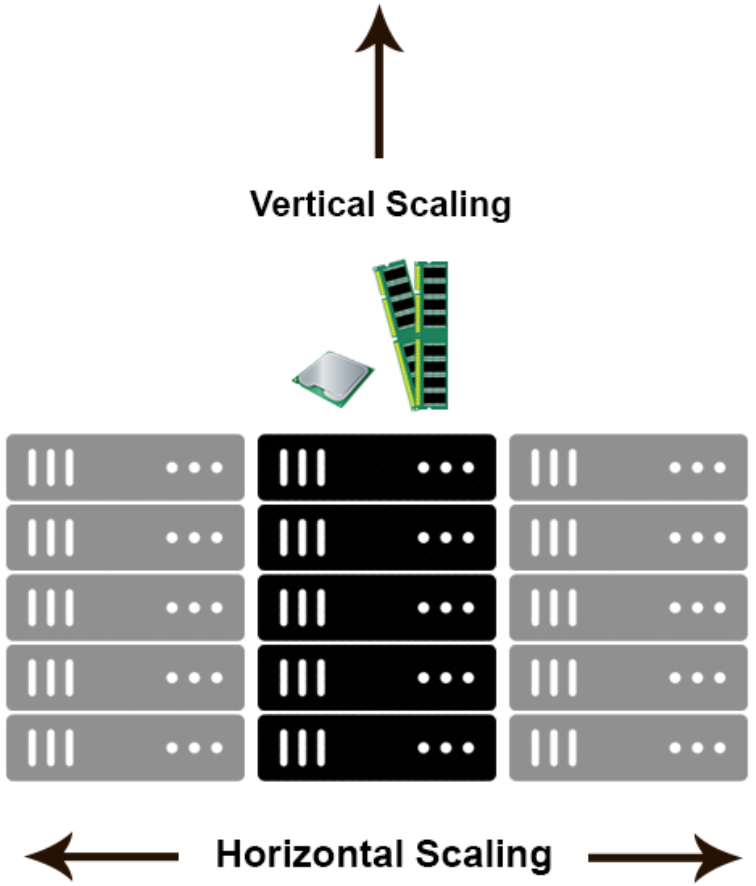


- **Virtualisation**
  - VT is used to create virtual instances of IT resources.
    - A layer of virtualization software allows physical IT resources to provide multiple virtual images of themselves
    - Underlying processing capabilities that can be shared by multiple users.

Lecture 1: Introduction to Cloud Computing

Two Scaling Methods (***ID 4, Lec1, 42-43***):

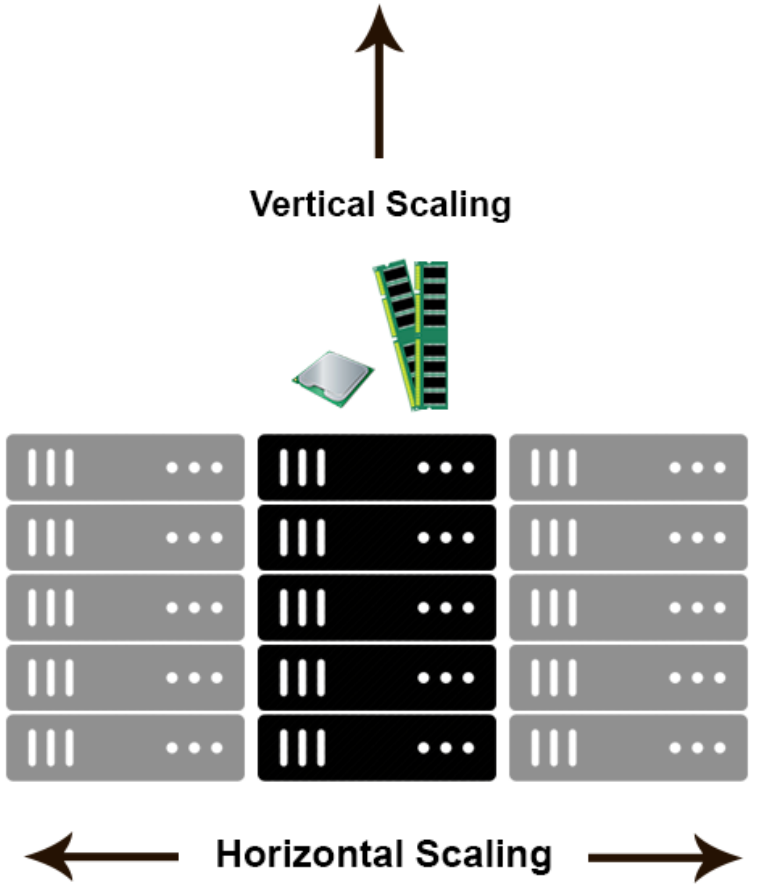
- Scaling
  - the ability of the IT resource to handle increased or decreased usage demands.
- Scaling Types:
  - **Horizontal scaling**: allocating or releasing of IT resources (same type)
  - **Vertical scaling**: higher or lower capacity of the current IT resources (less common due to downtime)



Lecture 1: Introduction to Cloud Computing

Two Scaling Methods (***ID 4, Lec1, 42-43***):

- **Horizontal scaling VS Vertical scaling**



Horizontal Scaling	Vertical Scaling
Less expensive	More expensive
IT resources instantly available	IT resources normally instantly available
Resource replication and automated scaling	Additional setup is normally needed
Additional IT resources needed	No additional IT resources needed
Not limited by hardware capacity	Limited by maximum hardware capacity

A comparison of horizontal and vertical scaling.

Lecture 1: Introduction to Cloud Computing

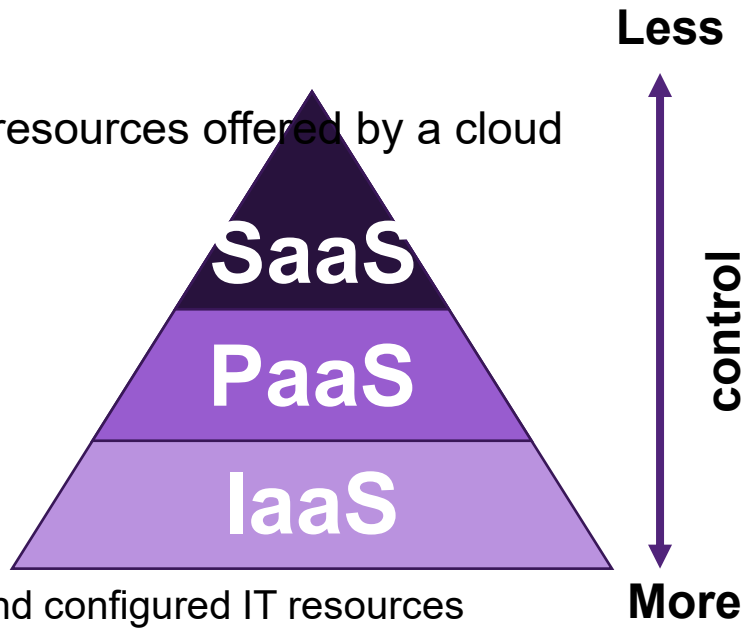
Cloud Characteristics (***ID 5, Lec1, 45-46***):

- On-Demand Self-service Usage
- Ubiquitous Access
- Multitenancy
- Elasticity
- Measured Usage
- Resiliency

Lecture 1: More on Cloud Computing

Cloud Delivery Models (***ID 6, Lec1, 50-57***):

- A *cloud delivery model* represents a specific, pre-packaged combination of IT resources offered by a cloud provider.
- Three common cloud delivery models:
  - **Infrastructure-as-a-Service (IaaS)**
    - a self-contained virtual environment consists of infrastructure-centric IT resources
    - users: system admins
    - examples: Amazon EC2, Google Compute Engine (GCE), Microsoft Azure
  - **Platform-as-a-Service (PaaS)**
    - a pre-defined "ready-to-use" environment typically consists of already deployed and configured IT resources
    - users: developers
    - examples: AWS Elastic Beanstalk and Google app engine (GAE)
  - **Software-as-a-Service (SaaS)**
    - a shared cloud service and made available as a "product"
    - users: end-users
    - examples: Google ecosystem docs/sheets/mails/calendar/etc Microsoft Office 365



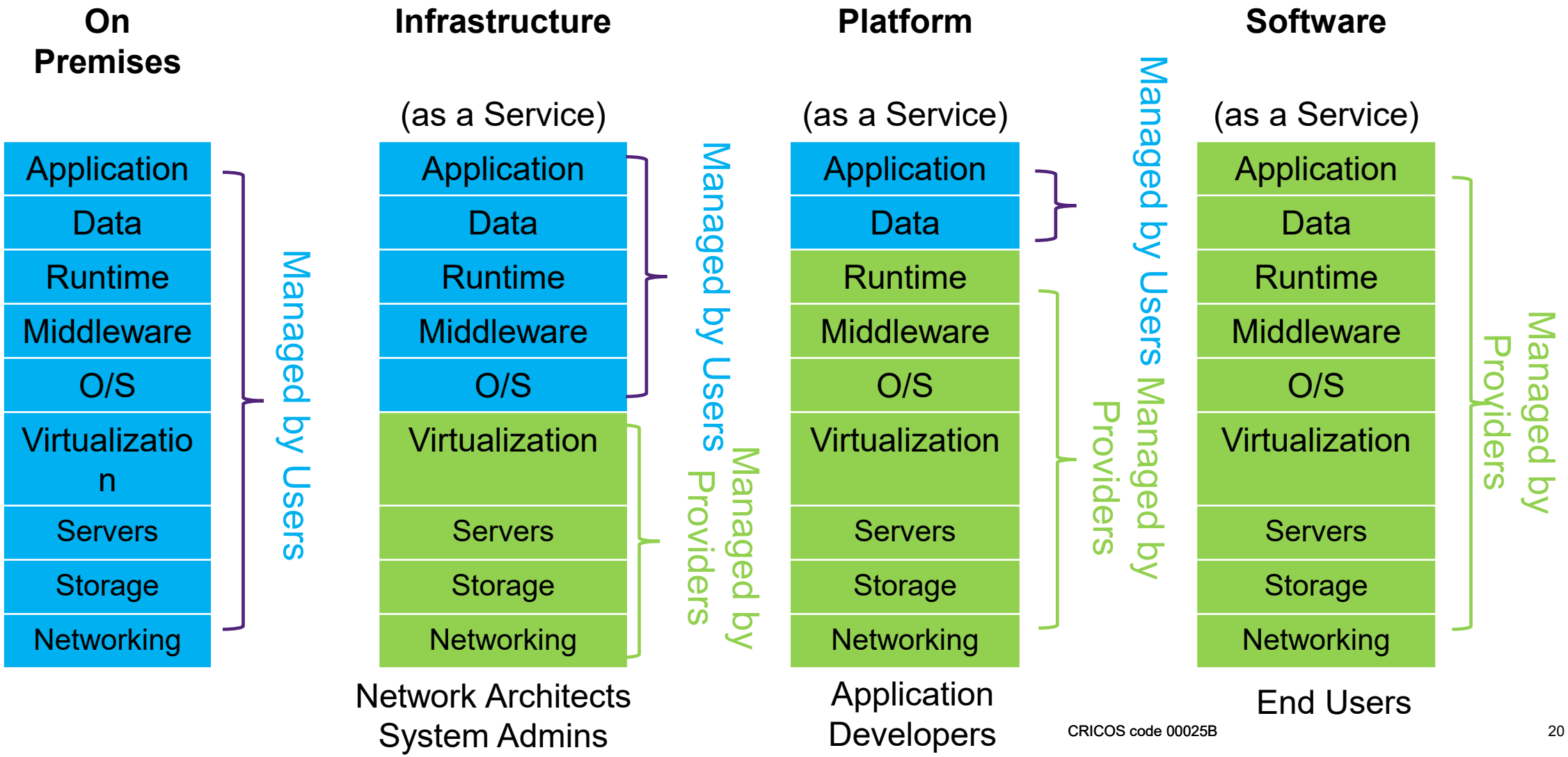
Lecture 1: More on Cloud Computing

Cloud Delivery Models (***ID 6, Lec1, 50-57***):

- Three models comparison:

Delivery Model	Control Level	Functionality	Consumer Activities	Provider Activities
SaaS	Usage and usage-related configuration	Access to front-end user-interface	Uses and configures cloud services	Implements, manages, and maintains cloud service Monitors usage by cloud consumers
PaaS	Limited administrative	Moderate level of administrative control over IT resources relevant to cloud consumer's usage of platform	Develops, tests, deploys, and manages cloud services and cloud-based solutions	Pre-configures platform and provisions underlying infrastructure, middleware, and other needed IT resources, as necessary Monitors usage by cloud consumers
IaaS	Full administrative	Full access to virtualized infrastructure-related IT resources and possibly to underlying physical IT resources	Sets up and configures bare infrastructure, and installs, manages, and monitors any needed software	Provisions and manages the physical processing, storage, networking, and hosting required Monitors usage by cloud consumers

Lecture 1: More on Cloud Computing



Lecture 1: More on Cloud Computing

Amazon's AWS:

- leading company in Cloud Computing
- provides IaaS and PaaS
- famous for EC2

Google Cloud:

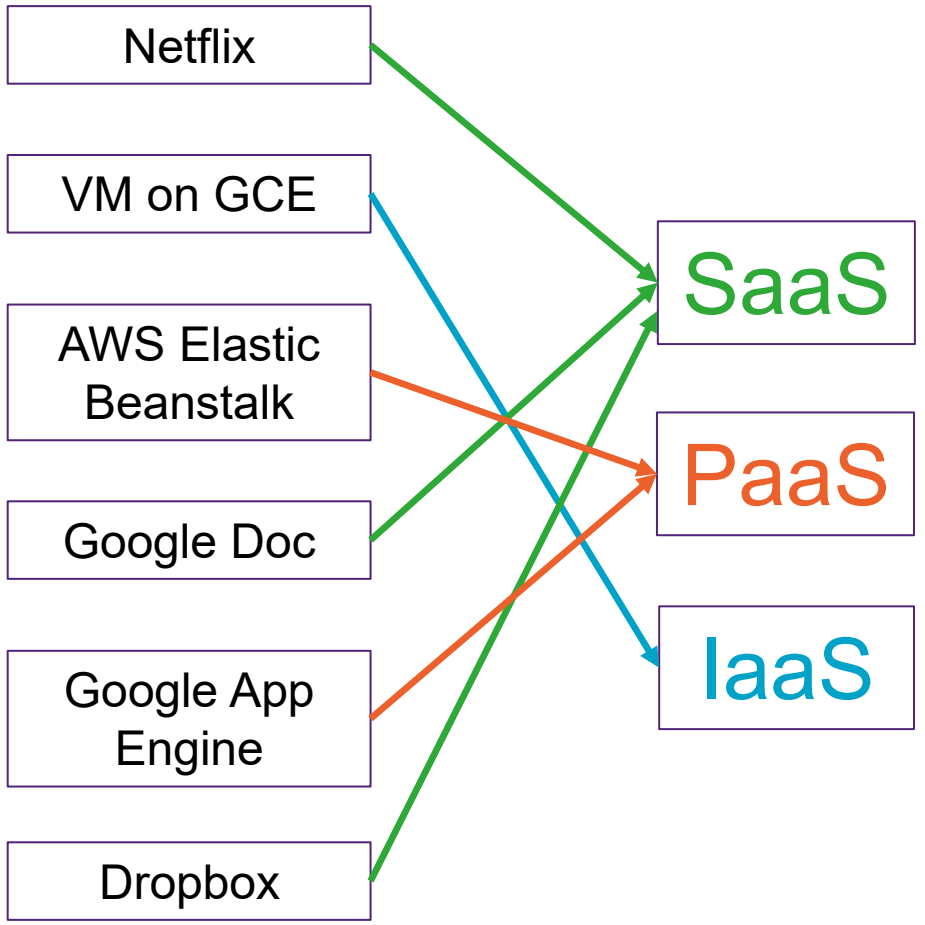
- offers IaaS, PaaS (GAE), and SaaS (Google docs/sheets/calendar/gmail)

Microsoft Azure:

- provides IaaS, PaaS, and SaaS (Office 365)

iCloud:

- majorly for Apple products (Macbook, iPad, iPhone, etc)
- store and backup users' documents online.
- Storage-as-a-service (aka STaaS)



Lecture 1: More on Cloud Computing

Cloud Deploy Models (***ID 7, Lec1, 58-59***):

- **Public cloud**:
  - a publicly accessible cloud environment owned by a third-party cloud provider
  - typical examples: GCP, AWS, AZURE, etc.
  - UQcloud?
- **Community cloud**:
  - is similar to a public cloud except that its access is limited to a community of cloud consumers.
  - typical examples: Cloud for multiple governmental departments.
- **Private cloud**:
  - is owned by a single organization and enables an organization to use CC technology to access to IT resources by different parts, locations, or departments.
  - Typical examples: UQCloud (VMs) and UQRDM cloud (STaaS)
- **Hybrid cloud**:
  - is a cloud environment comprised of two or more different cloud deployment models.
  - Example: private cloud (sensitive data) + public cloud (less sensitive cloud services)

Lecture 1: More on Cloud Computing

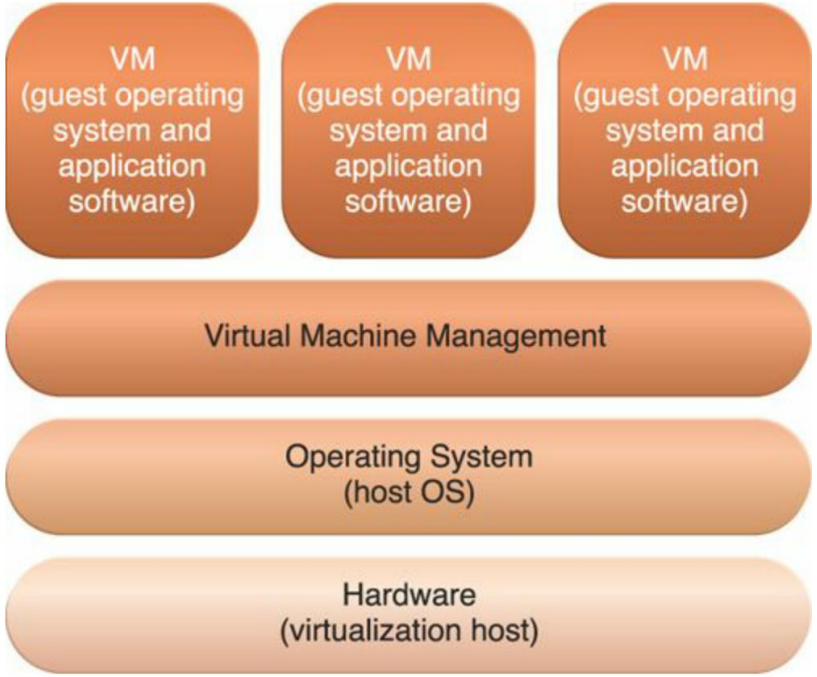
Cloud-Enabling Technology:

- Broadband Networks and Internet Architecture
- Virtualisation Technology (VT) (***ID 8, Lec1, 63-67***):
- Data Centre Technology
- Web Technology
- Multitenant Technology (***ID 9, Lec1, 75-77***):

Lecture 1: More on Cloud Computing

Cloud-Enabling Technology II: Virtualisation (***ID 8, Lec1, 63-67***):

- VT is the process of converting a physical IT resource into a virtual IT resource.
- A physical server is called a *host* or *physical host.*
- A software that manages VMs and hardware is called as *Virtual Machine Monitor (VMM)*, also known as *hypervisor* in cloud computing context.
- An operating system in a virtual machine is called as *guest OS*.

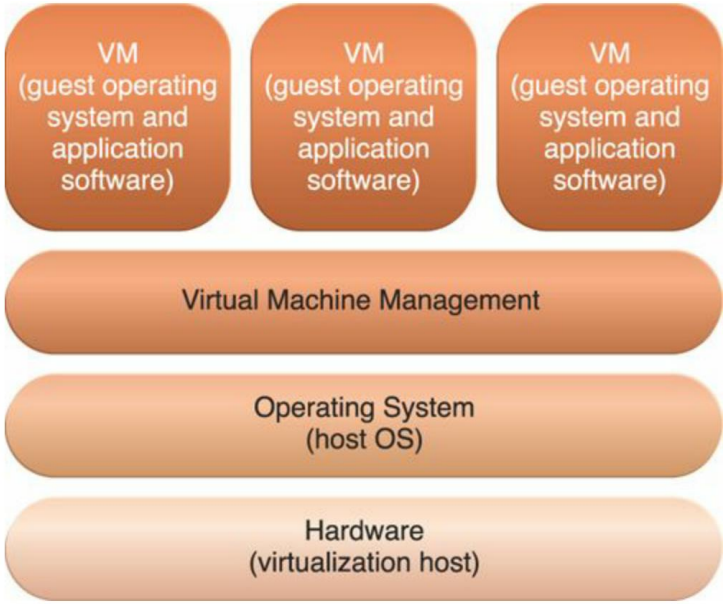


CET II: Virtualisation Technology (VT)

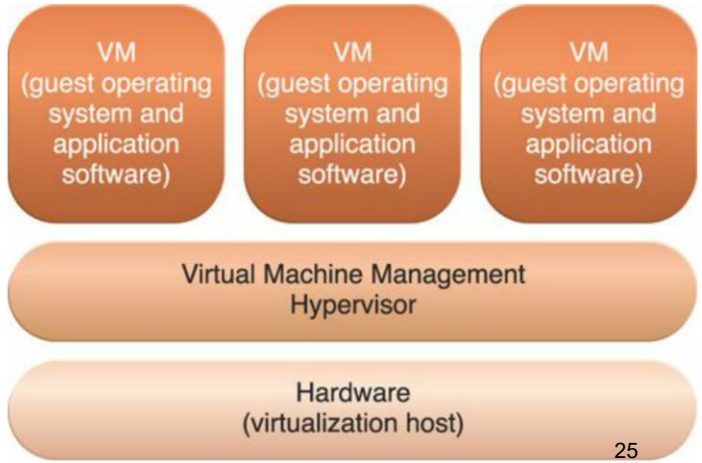
Cloud-Enabling Technology II: Virtualisation (***ID 8, Lec1, 63-67***):

- **Operating System-Based Virtualisation**
  - virtualisation is the installation of virtualisation software in a pre-existing operating system (called the host operating system)
  - example: Install ubuntu on Windows with VMware/VirtualBox
  - processing overhead: virtualisation software and host OS.
- **Hardware-Based Virtualisation**
  - the installation of virtualisation software directly on the physical host hardware bypassing the host OS
  - example: Oracle VM Server for x86 (up to 384 CPUs and 6TB RAM)
  - VMM is also named as Hypervisor
  - more efficient (no hosting OS), but compatible issues.

operating system-based virtualisation



Hardware-based virtualisation



Lecture 1: More on Cloud Computing

Cloud-Enabling Technology V: Multitenancy (***ID 9, Lec1, 75-77***):

- Multitenant application enables multiple users (tenants) to access the same application logic simultaneously.
- Each tenant has its own view of the application that it uses, administers, and customises as a dedicated instance of the software while remaining unaware of other tenants that are using the same application.
- Multitenant applications ensure that tenants do not have access to data and configuration information that is not their own.
- Tenants can individually customise features of the application:
  - User Interface – Tenants can define a specialised “look and feel” for their application interface.
  - Business Process – Tenants can customise the rules, logic, and workflows of the business processes that are implemented in the application.
  - Data Model – Tenants can extend the data schema of the application to include, exclude, or rename fields in the application data structures.
  - Access Control – Tenants can independently control the access rights for users and groups.

Lecture 1: More on Cloud Computing

Goals and Benefits of Cloud Computing (***ID10, Lec1, 79***)

- **Reduced Investments and Proportional Costs**
  - Capital reduced by cloud provider (mass-inquisition, hardware/software sharing, and data centre deployments)
  - Elimination or minimization of IT investments (focus on core business)
  - Common measurable benefits, e.g. on-demand access to pay-as-you-go computing resources (CPU by hr)
- **Increased Scalability**
  - can instantly and dynamically allocate IT resources
  - always meet and fulfil unpredictable demands avoids potential loss
- **Increased Availability and Reliability**
  - resilient IT resources
  - failover support (recovery)

IPC: Investments & Proportional Cost

ScAR: Scalability, Availabity & Reliability

Lecture 2: VPC Network, Load Balancing & Cloud Architectures

ID	Topics	Pages	Lecture
11	VPC definition and key techs	6-7	2
12	Regions and Zones	8-12	2
13	Routes and Firewall Rules	15	2
14	Load Balancing	20-21	2
15	LB Algorithms	24-29	2
16	Layer 7 and Layer 4 Load Balancing	45-47	2

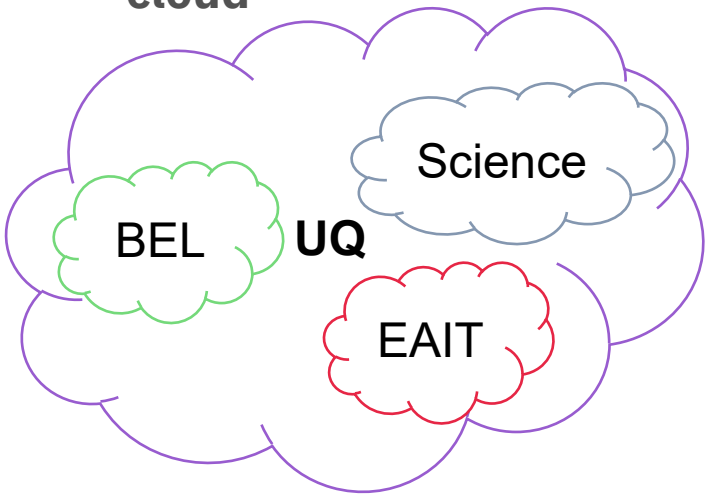
Virtual Private Cloud (VPC)

Cloud Deployment models (***ID11, Lec2, 6-7***):

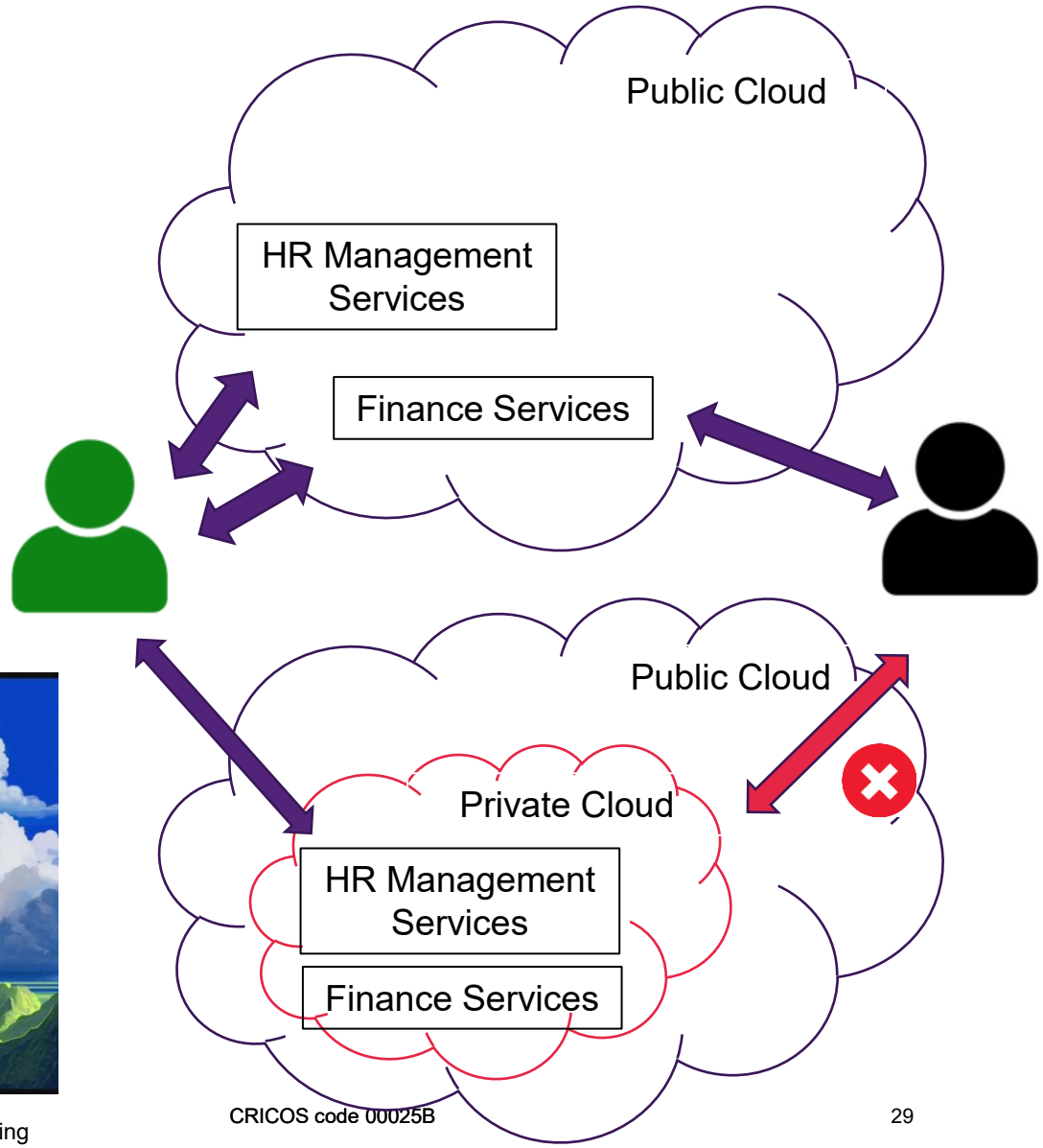
- Public Cloud (e.g. AWS, GCP) vs. Private Cloud (UQCloud)
- Human Resource department vs. Finance department in one company

A virtual private cloud (VPC) is a virtualized private cloud within a public cloud (GCP, AWS) for an organization

Advantages of VPC: Better Security + **All benefits of public cloud**



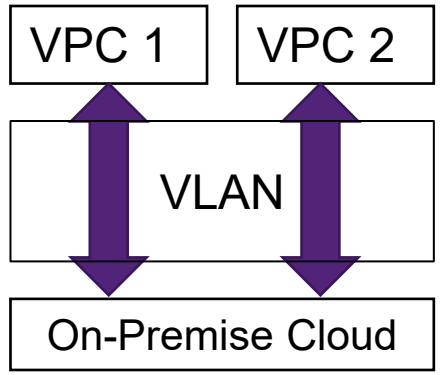
https://ichthyoid.writeas.com/castle-in-the-sky-a-study-in-world-building



Virtual Private Cloud (VPC)

The key technologies for isolating a VPC from the rest of the public cloud are (***ID11, Lec2, 6-7***):

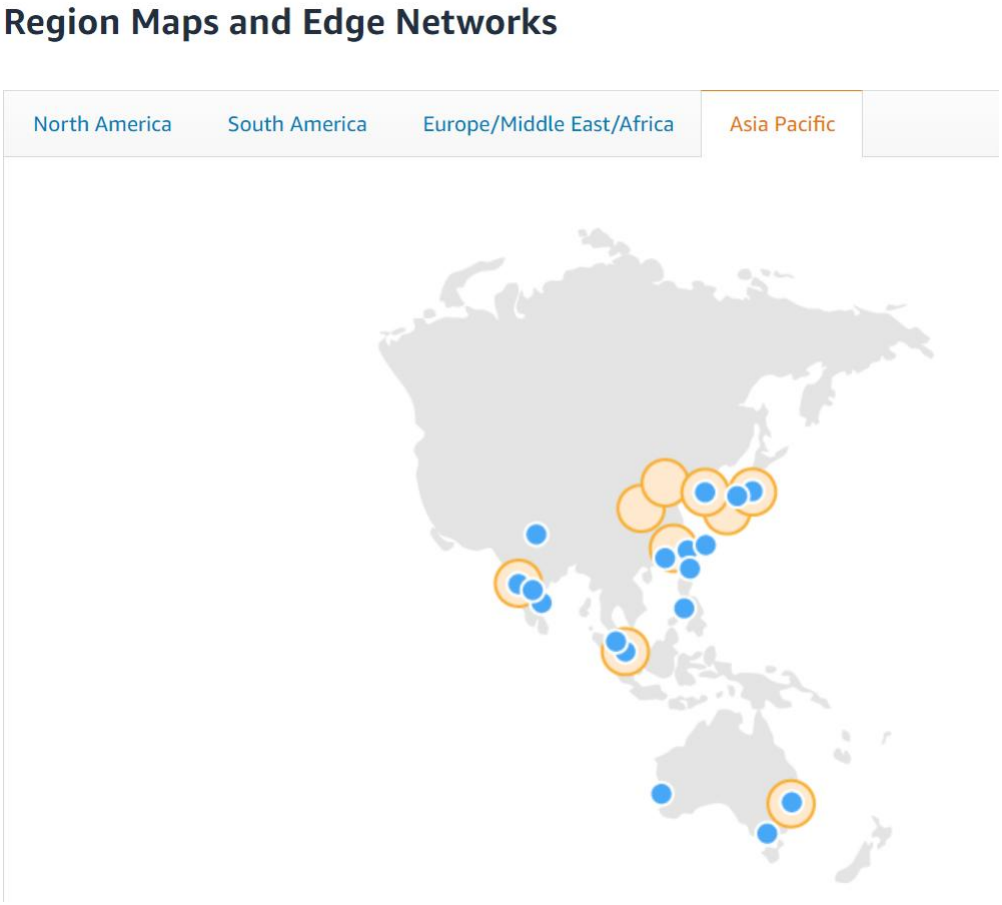
- **Subnets:**
  - A subnet (a range of IP addresses) is reserved (not available to everyone) within the network - for private use.
  - In a VPC, cloud providers will allocate private IPs (not accessible via the public Internet, e.g. 10.xxx.xxx.xxx).
- **VPN:**
  - A virtual private network (VPN) uses encryption to create a private network.
  - VPN traffic passes through publicly shared Internet infrastructure – routers, switches, etc.
- **VLAN (Virtual Local Area Network):**
  - A VLAN is a virtual LAN and it's used to partition a network.
  - Performance solution for hybrid connection setups (e.g. VPCs + On-Premise Private Cloud)
- **NAT (Network Address Translation):**
  - NAT matches private IP addresses to a public IP address for connections with the public Internet.
  - With NAT, a public-facing website or application could run in a VPC.



Regions and Zones

Regions and Zones (***ID12, Lec2, 8-12***)

- Cloud Providers organize IT resources by regions and zones
- Availability Regions
  - the geographic locations of the data centres
    - E.g. China, North America, Southeast Asia, East Asia, Europe, Middle East, etc.
  - collection of zones
  - Specific location to run resources
- Availability Zones
  - one or more discrete data centers with redundancy in a Region
  - Multiple zones are interconnected with encryption
- Prices of IT resources in different zones and regions could be very different!



https://aws.amazon.com/about-aws/global-infrastructure/regions_az/?nc1=h_ls

A Multi-Region Application

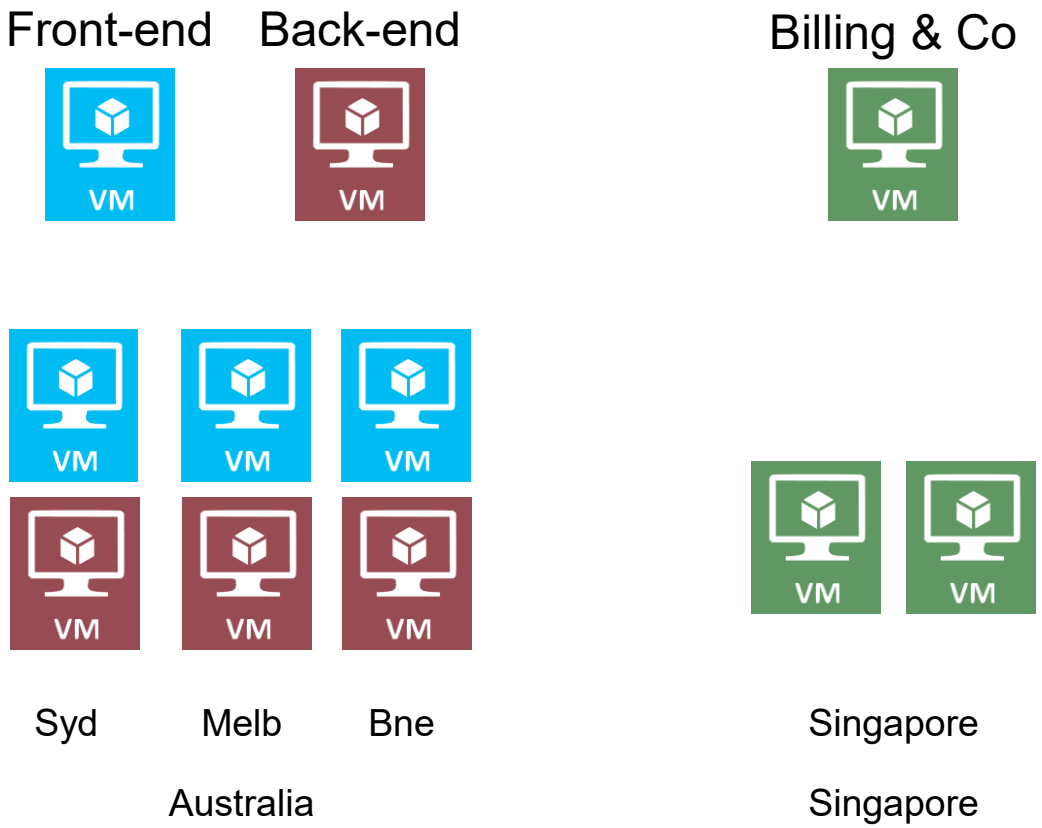
An Application that has three main functions:

- Front-end
- Back-end
- Other Functions (Billing & Co)

The customers are from Three Major Cities:

- Sydney
- Melbourne
- Brisbane

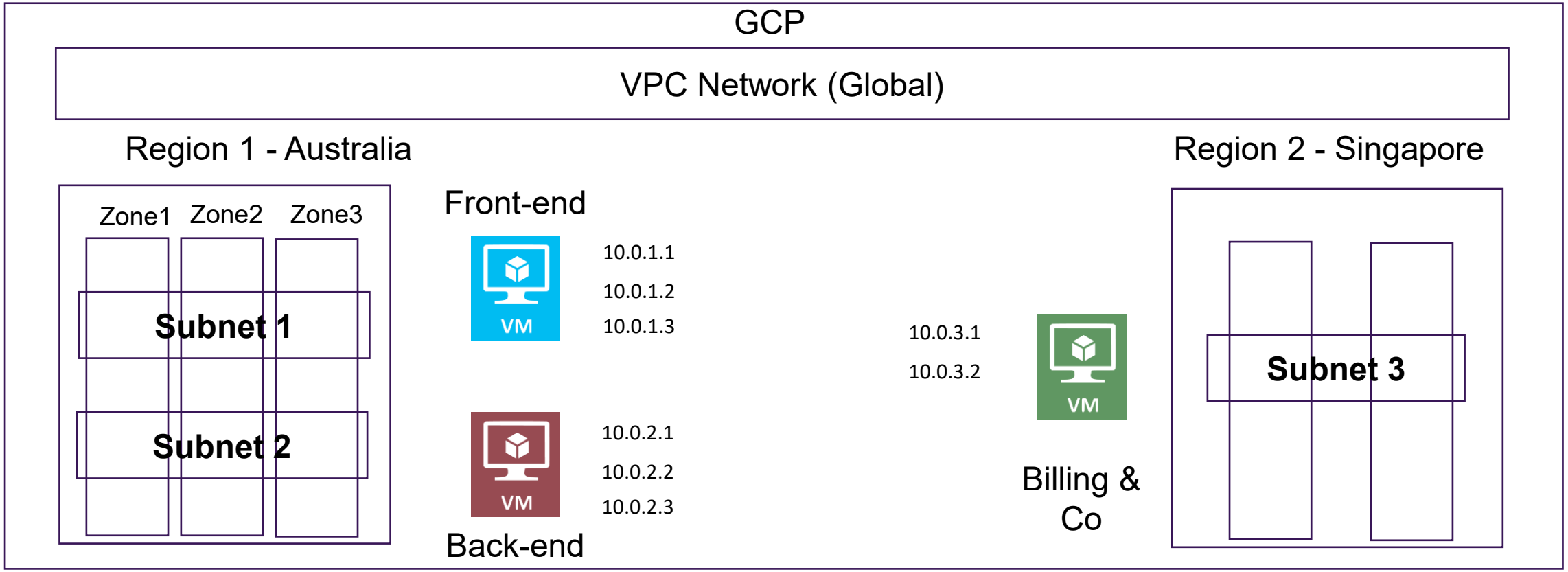
The HQ is in Singapore



VPC and Subnets in GCP and AWS

Subnets and VPC in GCP and AWS are differently organized:

- VPC in GCP is global (automatic routing for traffic) and Subnet is regional in GCP

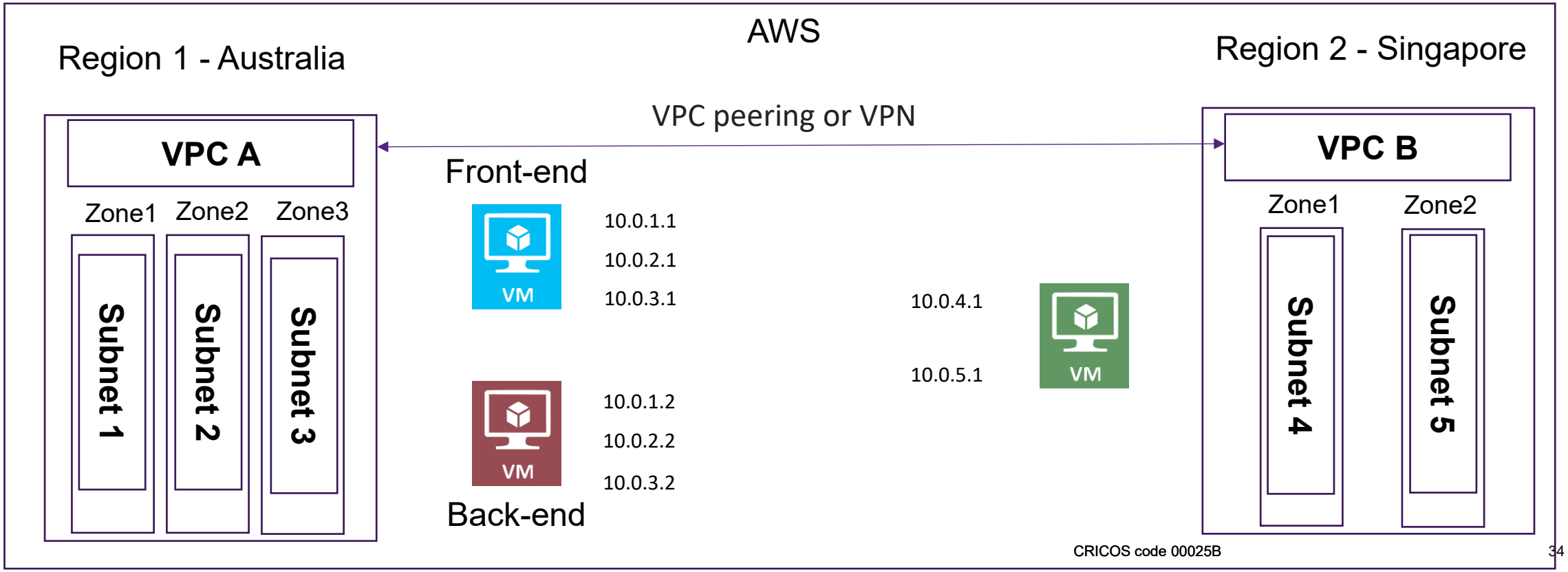


Subnets and VPC in GCP and AWS

Subnets and VPC in GCP and AWS are differently organized:

- VPC in AWS is regional (needs VPC peering setup);
- Subnet is confined in zones in AWS (needs routing setup)

Question:
Can Subnets in AWS overlap each other?



Routes and Firewall Rules

Regions and Zones (***ID13, Lec2, 15***)

- Routes define paths for packets leaving instances.
- Routes in Google Cloud are divided into two categories:
  - system-generated and custom.
- Firewall rules aim to protect your VPCs.
- Firewall rules apply to both outgoing (egress) and incoming (ingress) traffic in the network.
- Firewall rules control traffic even if it is entirely within the network.
- In GCP, every VPC network has implied firewall rules;
  - two implied IPv4 firewall rules,
  - two implied IPv6 firewall rules.
  - the implied egress rules allow most egress traffic, and the implied ingress rules deny all ingress traffic.
  - you cannot delete the implied rules, but you can override them with your own rules.
- To monitor which firewall rule allowed or denied a particular connection, see Firewall Rules Logging.

Load Balancing

What is load balancing? (***ID14, Lec2, 20-21***)

- Load balancing improves the distribution of workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units, or disk drives.
- Load balancing aims to optimise resource use, maximize throughput, minimize response time, and avoid overload of any single resource.
- Using multiple components with load balancing instead of a single component may increase reliability and availability through redundancy.
- Load balancing usually involves dedicated software or hardware, such as a multilayer switch or a Domain Name System server process.

Why should it be load-balanced?

- Improve resource utilisation
- Improve system performance
- Improve energy efficiency

Unbalanced

Load Balancing Problems

Server computing capacity problem – The thin clients waged too many applications.

Single-point Data Storage Problem - When a single data resource is (unexpected) demanded by an overwhelming number of clients, (i.e., a single data item is to be requested by many users).

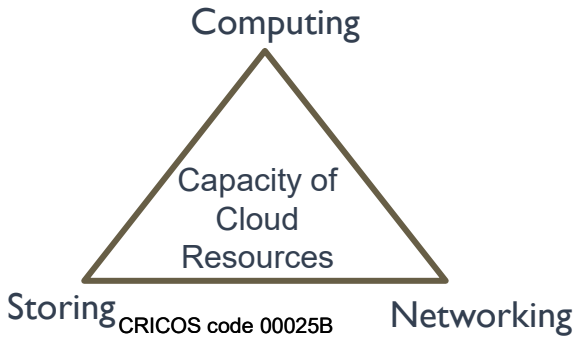
Traffic Problem - When a destination Web Server is to be visited by too many clients.

Storage and Traffic Problem - When a server needs to maintain far too many incoming data streams (upstream) or outgoing data streams (down streams) for file exchanges.

Network Congestion Problem – When the demands of the (web) services is over the server's capacity.

Dynamic Change of the Clients Demands – The clients' demands of services are unpredictable and may change dramatically.

Load balancing is the process of finding overloaded nodes and then transferring the extra load to other nodes.



http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.429.996&rep=rep1&type=pdf

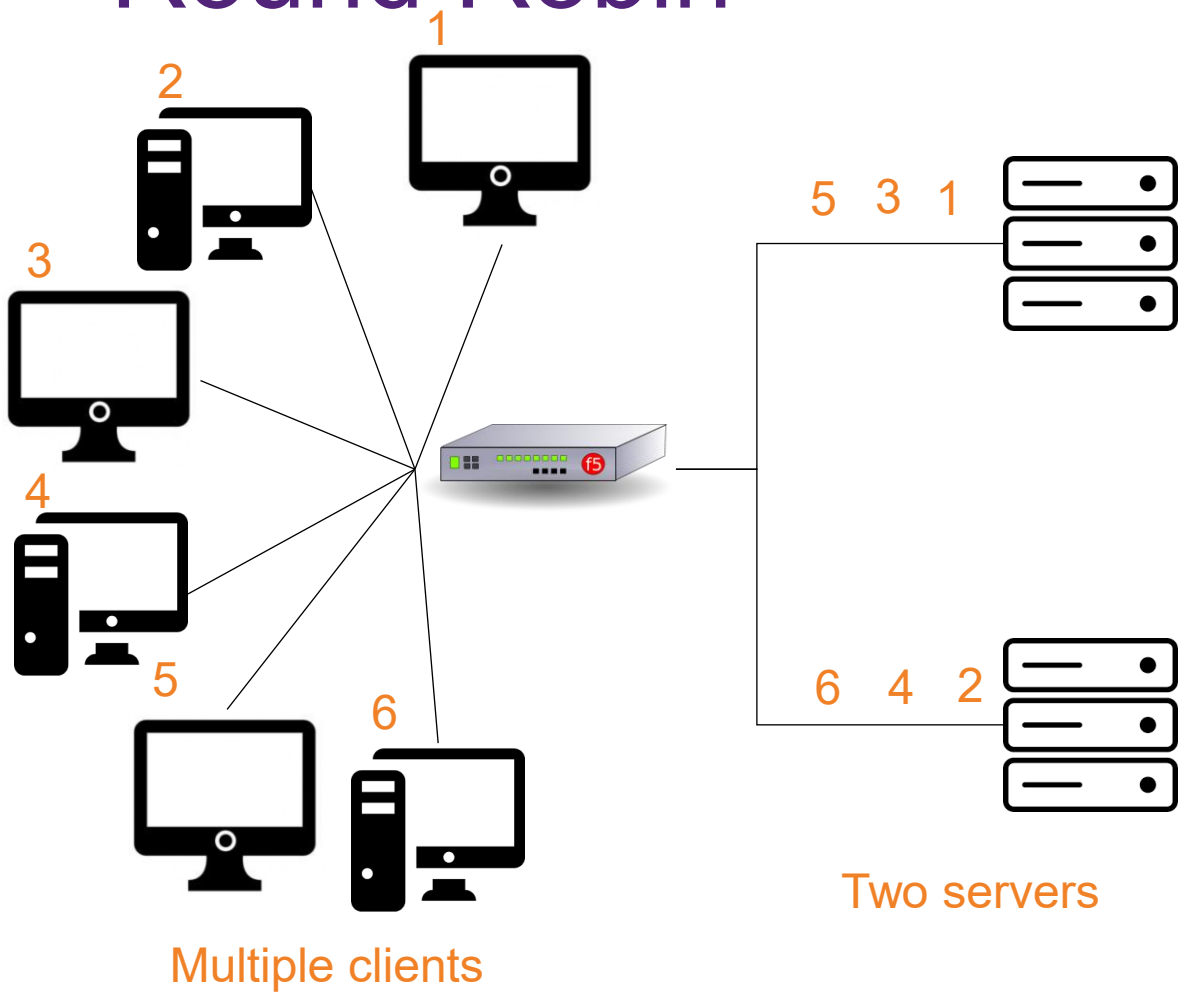
Load Balancing Algorithm – Round Robin

LB Algorithms (***ID15, Lec2, 24-29***)

Round-robin load balancing is one of the simplest methods for distributing client requests across a group of servers.

Going down the list of servers in the group, the round-robin load balancer forwards a client request to each server in turn.

When it reaches the end of the list, the load balancer loops back and goes down the list again (sends the next request to the first listed server, the one after that to the second server, and so on).



Singh, Navpreet, and Kanwalvir Singh Dhindsa. "Load Balancing in Cloud Computing Environment: A Comparative Study of Service Models and Scheduling Algorithms." *International Journal of Advanced Networking and Applications*8, no. 6 (2017): 3246.

Load Balancing Algorithm – Weighted Round Robin

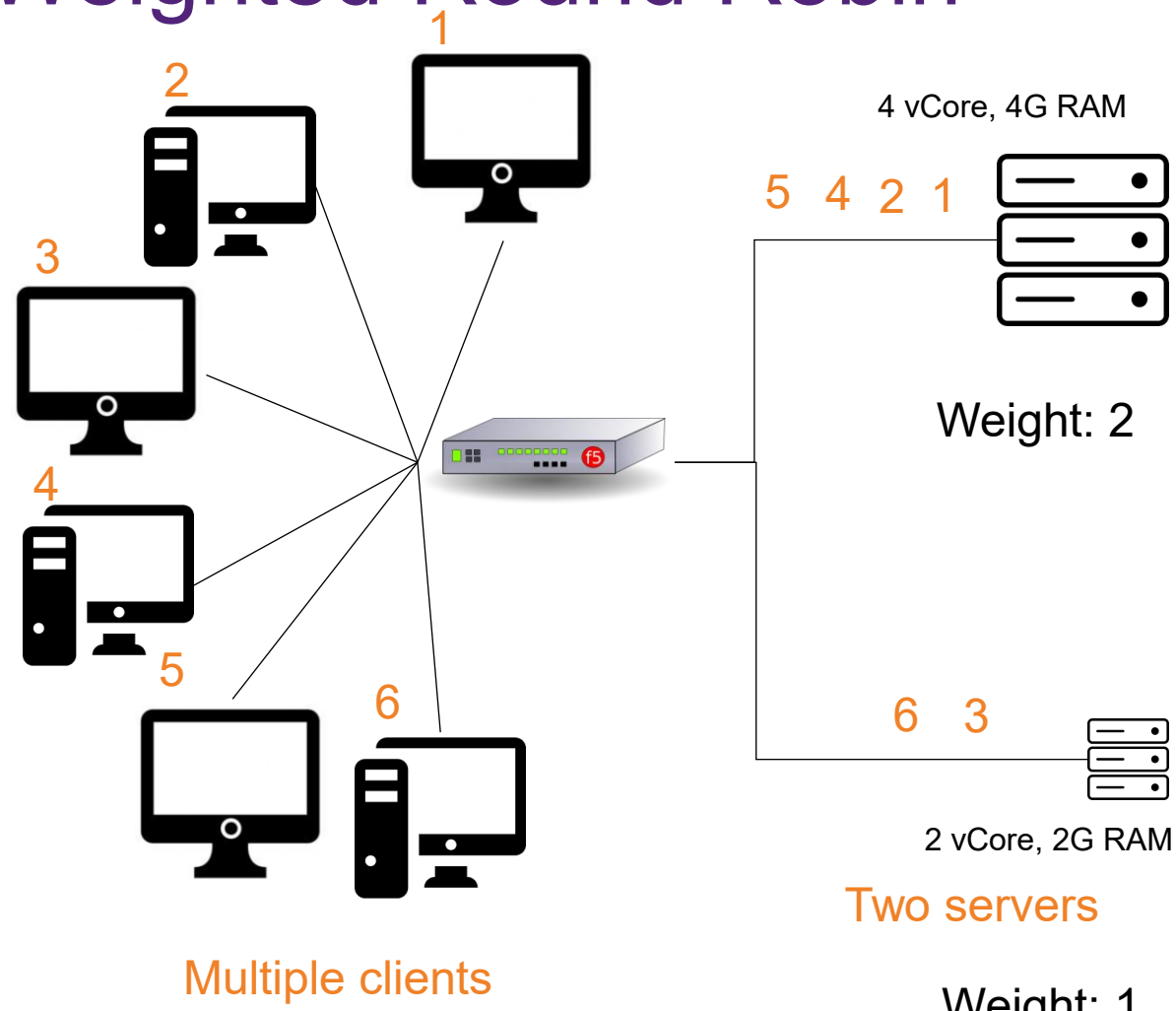
Weighted Round Robin load balancing is similar to the Round Robin (cyclic distribution).

The node with the higher specs will be apportioned a greater number of requests.

Set up the load balancer with assigned "weights" to each node according to hardware specs.

Higher specs, higher weight.

For example, if Server 1's capacity is 2x more than Server 2's, then you can assign Server 1 a weight of 2 and Server 2 a weight of 1.



Singh, Navpreet, and Kanwalvir Singh Dhindsa. "Load Balancing in Cloud Computing Environment: A Comparative Study of Service Models and Scheduling Algorithms." *International Journal of Advanced Networking and Applications*8, no. 6 (2017): 3246.

Load Balancing Algorithm – Least Connections

Identical hardware specs, but different occupied durations by client. E.g. clients connecting to Server 2 stay connected much longer than those connecting to Server 1.

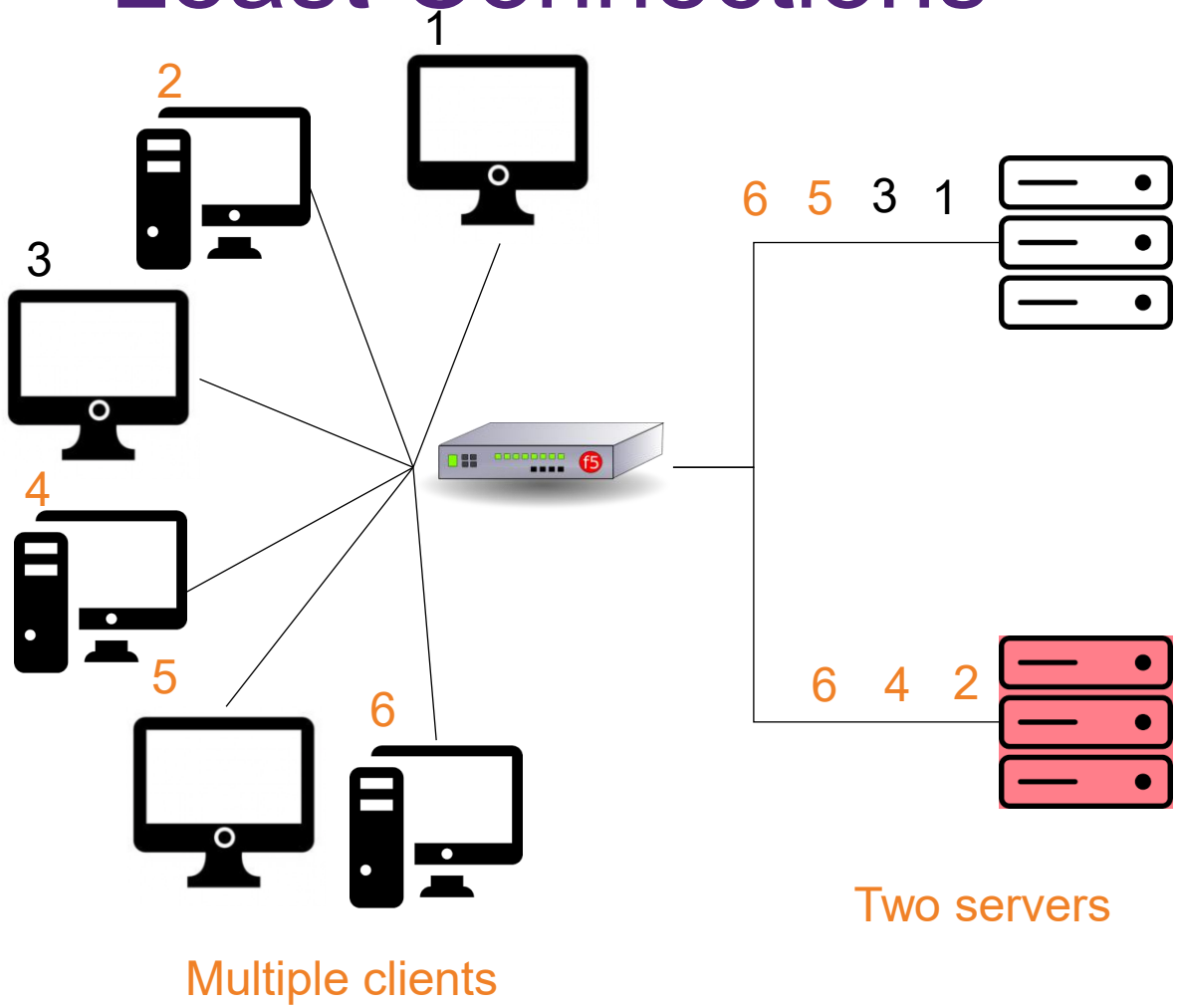
Congestion in Server 2 makes resources run out faster.

Example: clients 1 and 3 already disconnect, while 2, 4, 5, and 6 are still connected.

Least Connections algorithm consider the number of current connections each server has when load balancing.

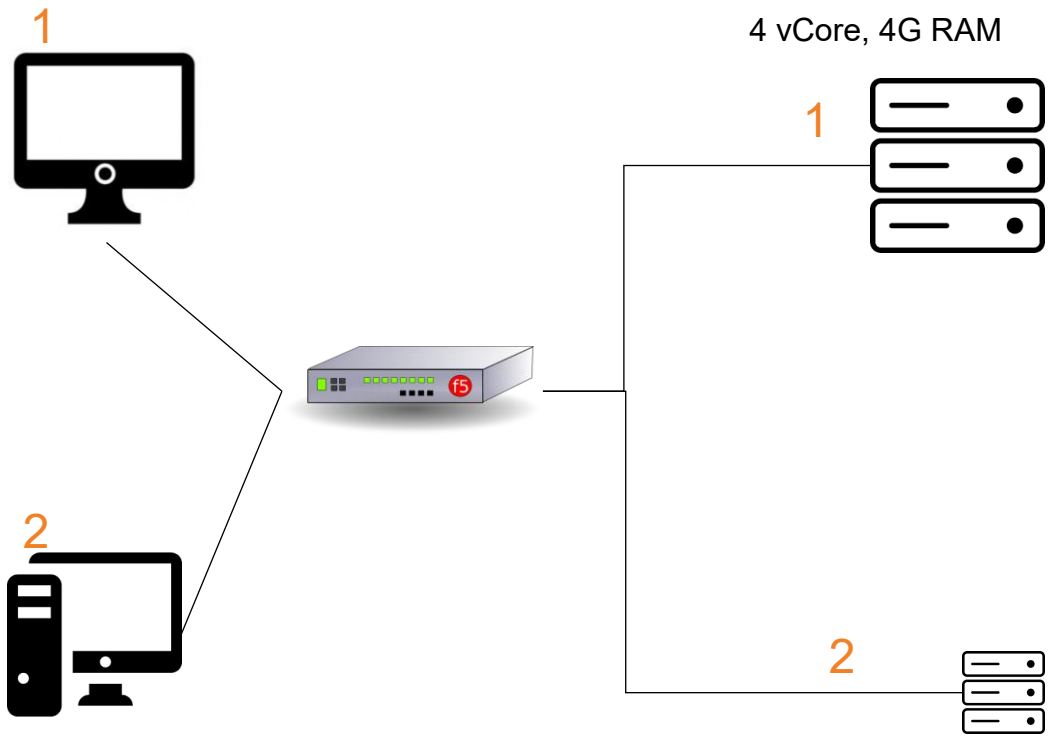
Less connection, higher priority for assignment.

Example, when using Least Connections algorithm, client 6 will be directed to Server 1 instead of Server 2.



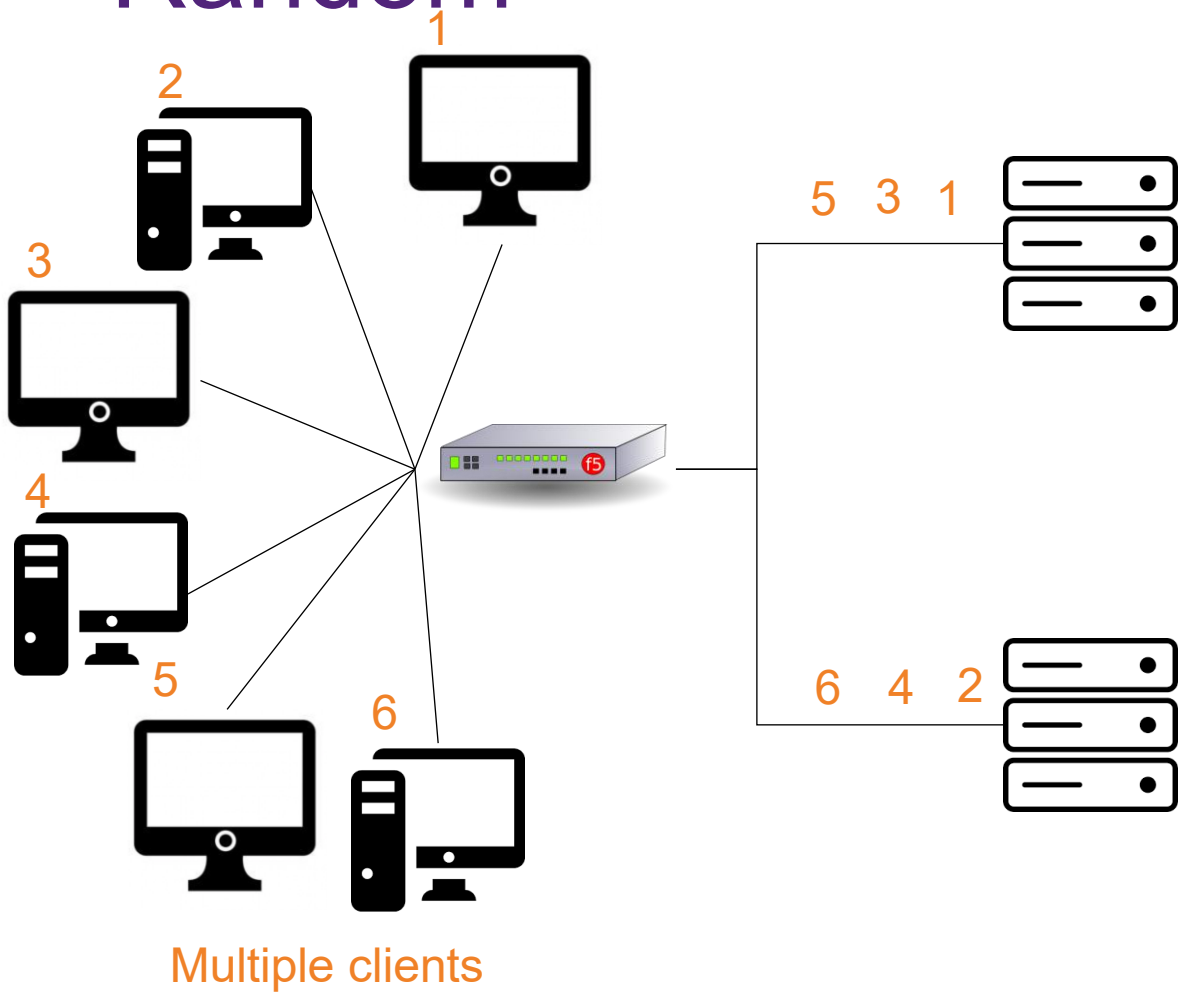
Load Balancing Algorithm – Weighted Least Connections

- The Weighted Least Connections algorithm applies a "weight" component based on the computing capacities of each server.
- Similar with Weighted Round Robin, setup a weight for each server.
- When directing an access request, a load balancer now considers two things:
  - the weights of each server
  - the number of clients currently connected to each server.



Load Balancing Algorithm – Random

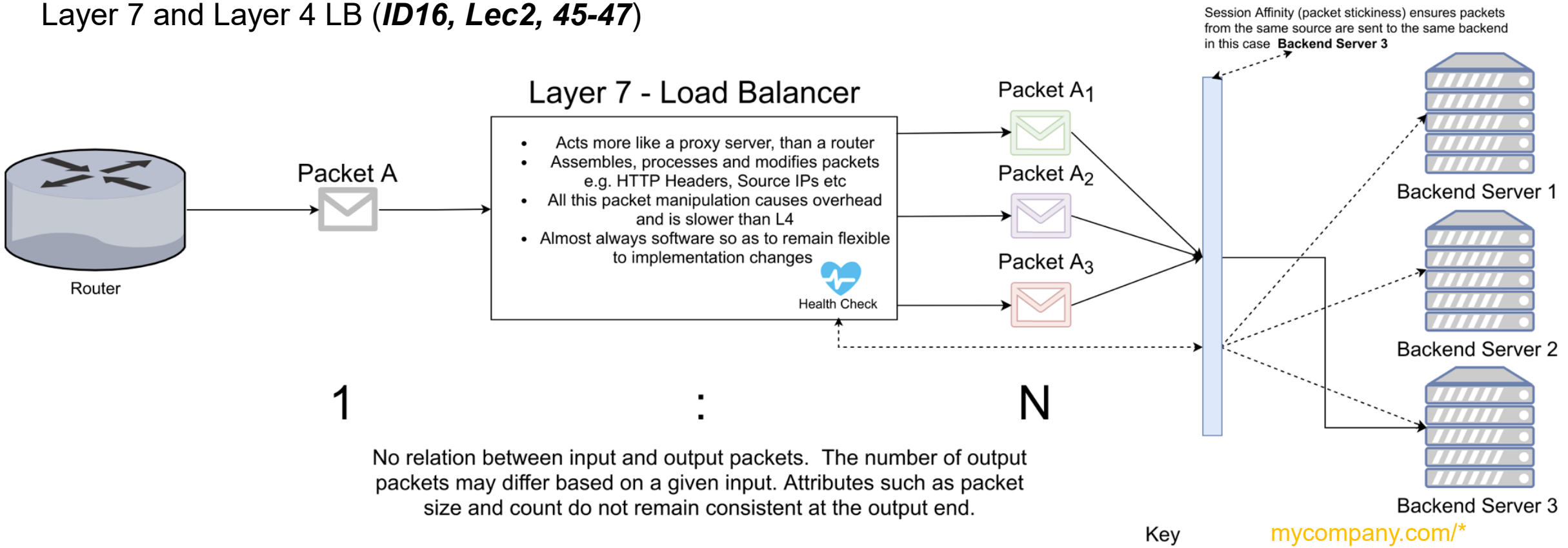
- As its name implies, this algorithm matches clients and servers by random, i.e. using an underlying random number generator.
- In cases wherein the load balancer receives a large number of requests, a Random algorithm will be able to distribute the requests evenly to the nodes.
- Like Round Robin, the Random algorithm is suitable for clusters consisting of nodes with similar configurations (CPU, RAM, etc).



HTTPs (Layer 7) Load Balancing

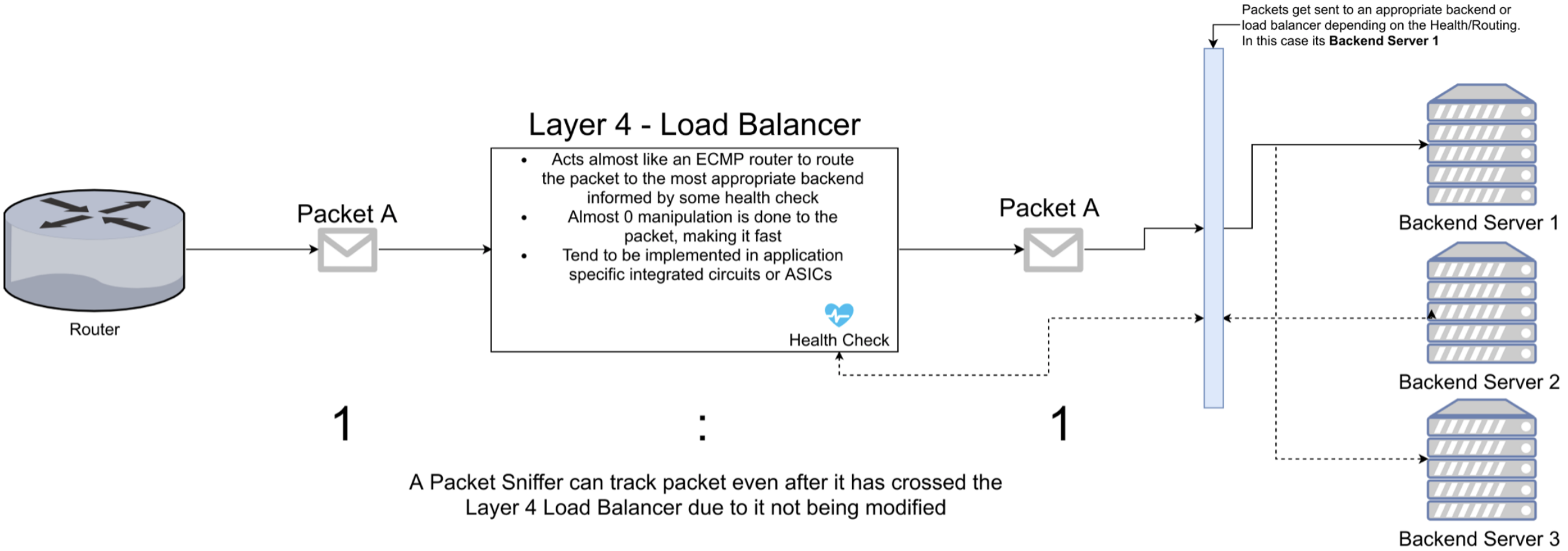
mycompany.com/upload/videos

Layer 7 and Layer 4 LB (***ID16, Lec2, 45-47***)



TCP vs HTTP(S) Load Balancing. https://medium.com/martinomburajr/distributed-computing-tcp-vs-http-s-load-balancing-7b3e9efc6167

TCP (Layer 4) Load Balancing



TCP vs HTTP(S) Load Balancing. https://medium.com/martinomburajr/distributed-computing-tcp-vs-http-s-load-balancing-7b3e9efc6167

Differences between Layer 4 and Layer 7 Load Balancing

	Layer 4 LB (TCP)	Layer 7 LB (HTTPs)
Layer	Transport Layer	Application Layer
Packet Manipulation	No	Yes
SSL Traffic	No	Yes
Logging & Monitoring	Not suitable	Yes
Implementation	Dedicated hardware	Typically software
Throughput Speed	Fast	Relatively lower

TCP vs HTTP(S) Load Balancing. https://medium.com/martinomburajr/distributed-computing-tcp-vs-http-s-load-balancing-7b3e9efc6167

Lectures 3,4 & 5: Docker Technology and Kubernetes

ID	Topics	Pages	Lecture
17	Containers	6,13,14	3
18	Linux Namespaces and Cgroups	7-11	3
19	Basic Concepts in Docker	21-26	3
20	Basic Docker commands	28-33	3
21	Containerisation and Dockerfile	36,38	3
22	Microservices	7-10	4
23	Docker-compose	13-22,24	4
24	Docker Swarm & Docker Machine	26-30, 35,36	4
25	Raft Consensus Algorithm	31-34	4
26	Deploy Services to a Swarm	49-55	4
27	Kubernetes Concepts and Components	8-17	5
28	Kubernetes Workflow	18, Tutorial 5 (Week 6)	5
29	Reconciliation Loop	22-24	5
30	Kubernetes Objects	33-66	5

Jump to Storage

Lecture 3: Docker I: Container, Docker Commands, and Dockerfile

Containers (***ID17, Lec3, 6,13,14***)

- Containers offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run.
- Deploy container-based applications more easily and consistently than running VMs
- Multiple containers running atop the OS kernel directly – far more lightweight:
  - share the OS kernel
  - start much faster (in seconds)
  - use a fraction of the memory (compared to booting an entire OS)
  - one physical machine/server can run much more containers than VMs
- **Run Anywhere:**
  - Containers are able to run virtually anywhere, greatly easing development and deployment:
- **Isolation:**
  - Containers virtualise hardware at the OS-level with a sandboxed view of the OS logically isolated from other applications.

Lecture 3: Docker I: Container, Docker Commands, and Dockerfile

Containers (***ID17, Lec3, 6,13,14***)

Features	Containers	Virtual Machines
Isolation	**Process-level** isolation using namespaces and cgroups	**Full OS-level** isolation with a separate guest OS
Resource Efficiency	**High**, due to sharing of the host OS kernel	**Lower**, as each VM runs a full OS
Startup Time	**Very fast** (seconds or less)	**Slower** (minutes)
Performance	**Near-native** performance	**Overhead** from hypervisor and full OS
Storage	Uses **layered file systems** (e.g., overlayfs)	Requires dedicated disk space for each VM
Deployment	**Lightweight** and quick to deploy	**Heavier** and slower to deploy
Scalability	**Easily scalable**	**Less flexible**, more resource-intensive to scale
Portability	**High portability** (consistent environments)	**Moderate portability** (more dependencies)
Security	**Less secure** than VMs (shared kernel vulnerabilities)	**More secure** (isolated OS kernel)
Use Cases	Microservices, DevOps, CI/CD, lightweight apps	Legacy applications, full OS environments, high-security isolation
Management Tools	Docker, Kubernetes	VMware, Hyper-V, VirtualBox
Overhead	**Minimal** overhead (shares host kernel)	**Significant** overhead (runs separate OS for each VM)

Lecture 3: Docker I: Container, Docker Commands, and Dockerfile

Linux Namespaces and Cgroups (***ID18, Lec3, 7-11***)

- Namespaces are a feature of the Linux kernel that partition kernel resources such that one set of processes sees one set of resources, while another set of processes sees a different set.
- Namespaces provides a process with its own isolated view of the system.
- Namespaces are utilized to give a container the illusion that it's running on its own, separate machine.

Key namespaces used by containers include:

- PID (Process ID): Isolates the process tree. Inside a container, the main process has a PID of 1, but on the host machine, it has a different PID.
- NET (Network): Provides each container with its own network stack (IP address, routing tables, etc.).
- MNT (Mount): Gives each container its own filesystem.
- UTS (UNIX Time-sharing System): Allows each container to have its own hostname.
- IPC (Inter-Process Communication): Isolates IPC resources.
- USER: Isolates user and group IDs.

Lecture 3: Docker I: Container, Docker Commands, and Dockerfile

Linux Namespaces and Cgroups (***ID18, Lec3, 7-11***)

- Control Groups (cgroups) are a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes.
- While namespaces handle what a container can see, cgroups control what a container can use.
- Cgroups prevent a single container from consuming all of the host's resources and impacting other containers or the host system itself.

- Resource Limiting: You can set hard limits on the amount of memory or CPU a container can use.
- Prioritization: You can allocate more or fewer resources to certain containers.
- Accounting: Cgroups monitor and report on the resource usage of containers.
- Control: You can freeze and resume all processes within a cgroup.

Lecture 3: Docker I: Container, Docker Commands, and Dockerfile

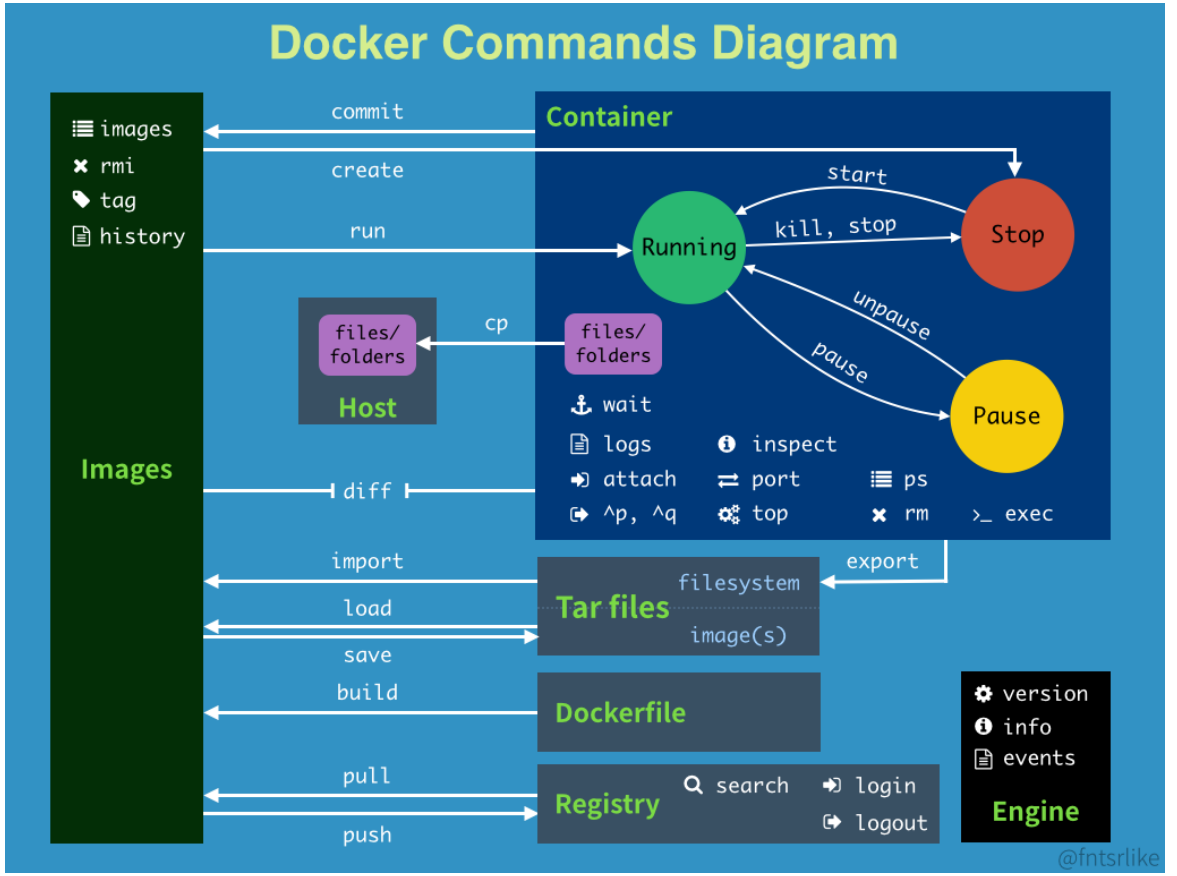
Basic Concepts in Docker (*ID19, Lec3, 21-26*)

- **Docker image** is an object that contains
  - an OS filesystem
  - and one or more applications
- An **image** is a template that is readable only to build a container.
- **Container**: a running instance of an image
  - Class vs Instances (in Objective Oriented Programming)
  - E.g. House Blueprints vs House
- Docker Image vs Container: One to Many
- After releases of docker images, a registry is needed to store and manage images for sharing.
- Docker images are stored in a **Docker registry**:
  - Public Docker registry: Docker Hub and Google Container Registry
  - Private Docker registry.

Lecture 3: Docker I: Container, Docker Commands, and Dockerfile

Basic Docker commands (***ID20, Lec3, 28-33***)

- **List images or containers**:
  docker images or docker ps
- **Remove one or more images or containers**:
  docker rmi or docker rm
- **Kill/Stop one or more running containers**:
  docker container kill/stop
  Log in running container with a bash:
  docker exec
- **Build an image from a Dockerfile**:
  docker build
- **Pull/push an image or a repository from a registry**:
  docker pull or docker push
- **Search the Docker Hub for images:**
  docker search



Lecture 3: Docker I: Container, Docker Commands, and Dockerfile

Containerisation and Dockerfile (***ID21, Lec3, 36,38***)

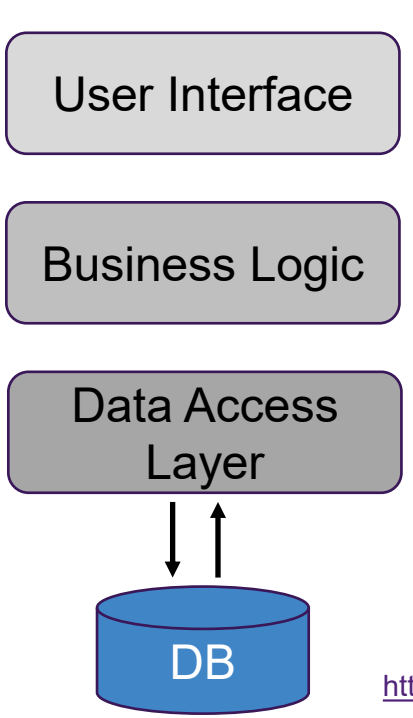
- The process of taking an application and configuring it to run as a container is called "containerizing" or "Dockerizing".
- Containers are all about apps! In particular, they're about making apps simple to build, ship, and run.
- **Dockerfile** is a text file that defines the environment inside the container
- **Dockerfile** is a collection of instructions and commands (blueprint) and clearly tells what are contained in the image.
- Docker can build images automatically by reading the instructions from a Dockerfile
  - Usage example: docker build <dockerfile>

Lecture 4: Docker II: Docker Compose and Docker Swarm

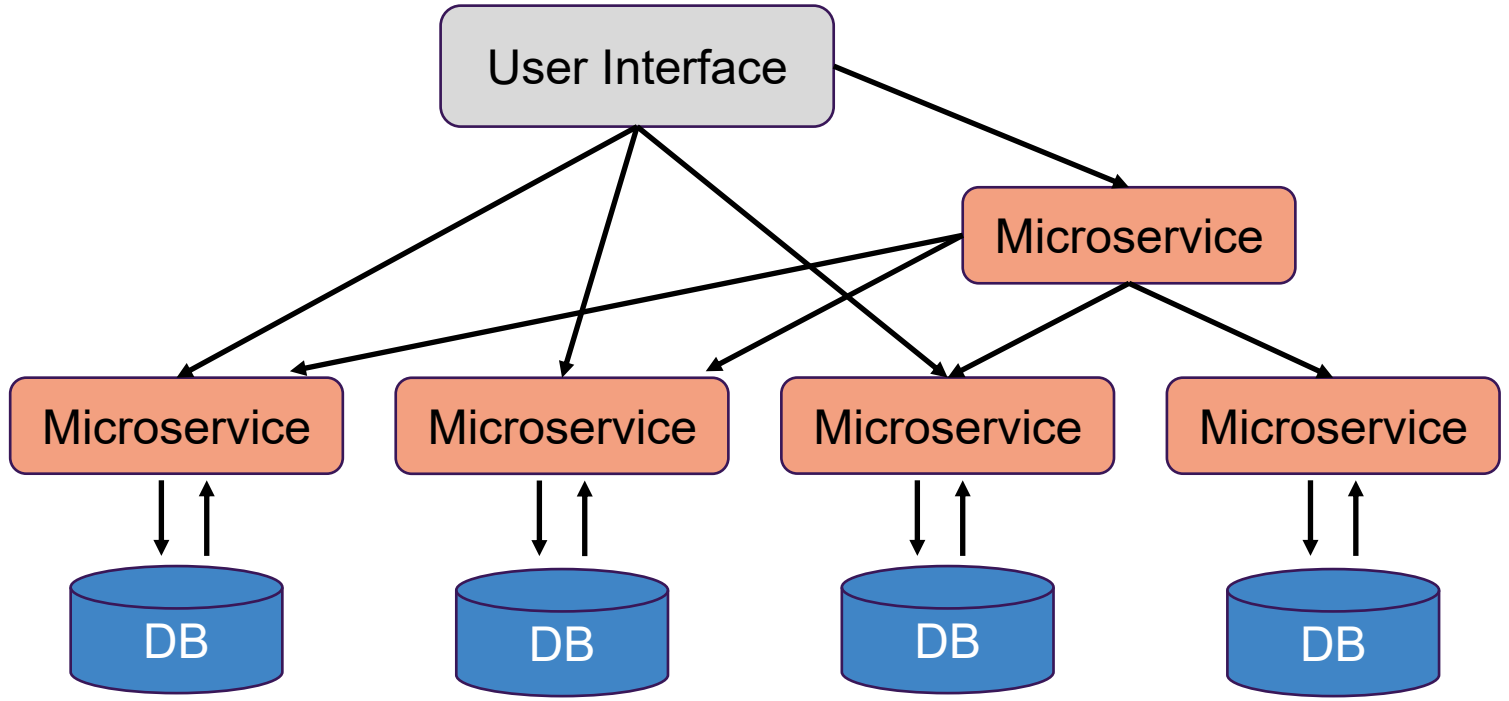
Microservices (***ID22, Lec4, 7-10***)

- Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs.

Monolithic Architecture



Microservices Architecture



https://aws.amazon.com/microservices/?nc1=h_ls

Three-Layer, Three-Tier, & Microservice Architectures

- **Microservices Architecture** breaks down an application into **small (atomic)**, **independent**, and **loosely coupled services**. Each service is responsible for **a specific piece of functionality** and can be **developed, deployed, and scaled independently**.
- **Three-Layer Architecture** (mainly taught in INFS3202/7202) is a **logical separation of** an application into three layers: Data Access Layer (Model), Presentation Layer (View), and Business Logic Layer (Controller).
  - Separation of Concerns, Maintainability, Not Focused on Scalability.
- **Three-Tier Architecture** is a **physical extension** of the three-layer arch, where each layer is **physically separated** and can be **hosted on different servers**: Presentation Tier, Application Tier, and Data Tier.
  - Physical Separation and Scalability

Differences:

- **Granularity**: Microservices break down the application into **finer-grained services**.
- **Scalability**: Microservices and Three-Tier architectures are designed with **scalability**.
- **Development Complexity**: Microservices may introduce additional complexity in deployment, monitoring, and inter-service communication.
- **Technology Stack**: Microservices allow for the use of different technologies for each service, while the other two architectures often use a consistent technology stack across layers or tiers.

https://aws.amazon.com/microservices/?nc1=h_ls

Lecture 4: Docker II: Docker Compose and Docker Swarm

Docker compose (***ID23, Lec4, 13-22,24***)

- Compose is a tool for defining and running multi-container Docker applications.
- Use a Compose file (docker-compose.yml) to configure application's services.
- Use a single command to create and start all the services from the configuration (docker-compose up).

docker-compose –f docker-compose.json up

- **Three-step process**:
  1. Define your app's environment with a Dockerfile
  2. Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
  3. Lastly, run docker-compose up and Compose will start and run your entire app.

```yaml
version: "3.7"

services:
  app:
    image: node:12-alpine
    command: sh -c "yarn install && yarn run dev"
    ports:
      - 3000:3000
    working_dir: /app
    volumes:
      - ./:/app
    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: root
      MYSQL_PASSWORD: secret
      MYSQL_DB: todos

  mysql:
    image: mysql:5.7
    volumes:
      - todo-mysql-data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: todos

volumes:
  todo-mysql-data:
```

An example of docker-compose.yml

Lecture 4: Docker II: Docker Compose and Docker Swarm

docker-compose up [options] [--scale SERVICE=NUM...] [SERVICE...]

docker-compose build

docker-compose down [options]

docker-compose run [SERVICE…]

docker-compose start [SERVICE…]

docker-compose stop, restart, kill, rm (un)paused

docker-compose logs [options] [SERVICE…]

docker-compose images

docker-compose ps [options] [SERVICE…]

docker-compose top

Compose files

Multi-service applications are defined in a configuration file

- docker-compose.yml (by default, but can use "-f" to read .yml file with a customised file name).
- YAML is a superset of JSON.
- consists of multiple layers that are split using tab stops or spaces
- Four main components in each Compose-File:
  - Compose file's version
  - Services (containers)
  - Volumes (storage)
  - Networks (linking)

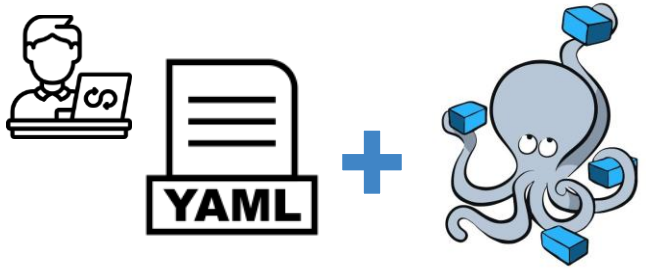
```yaml
version: '3.3'

services:
   db:
     image: mariadb:latest
     volumes:
       - database:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: MyWP123
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress
     networks:
       - app-network

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "80:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
       WORDPRESS_DB_NAME: wordpress
     networks:
       - app-network
volumes:
    database: {}
networks:
    app-network:
       driver: bridge
```

Benefits

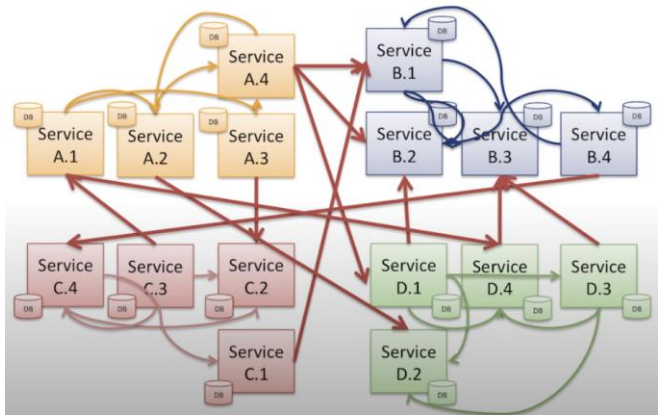
- **Simplified Configuration**: With Docker Compose, you can describe your system's **entire configuration**, including services, networks, and volumes, in a single YAML file, making it easier to **understand** and **maintain** your system.
- **Ease of Deployment**: A single command (**docker-compose up**) with a configuration file, which ensures that all containers are started in the **correct order**, with the **proper settings (env variables, networks, volumes, etc)**.
- **Scalability and Load Balancing**: With a few simple commands, Docker Compose enables you to scale specific services up or down to handle changes in load.
- **Resource Allocation**: You can specify **CPU, memory limits, and other resources** on a per-service basis, giving you fine-grained control over resource allocation.
- ...

```yaml
services:
  fronted:
    image: awesome/webapp
    deploy:
      mode: replicated
      replicas: 6
```

```yaml
services:
  frontend:
    image: awesome/webapp
    deploy:
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
          pids: 1
        reservations:
          cpus: '0.25'
          memory: 20M
```



docker Compose

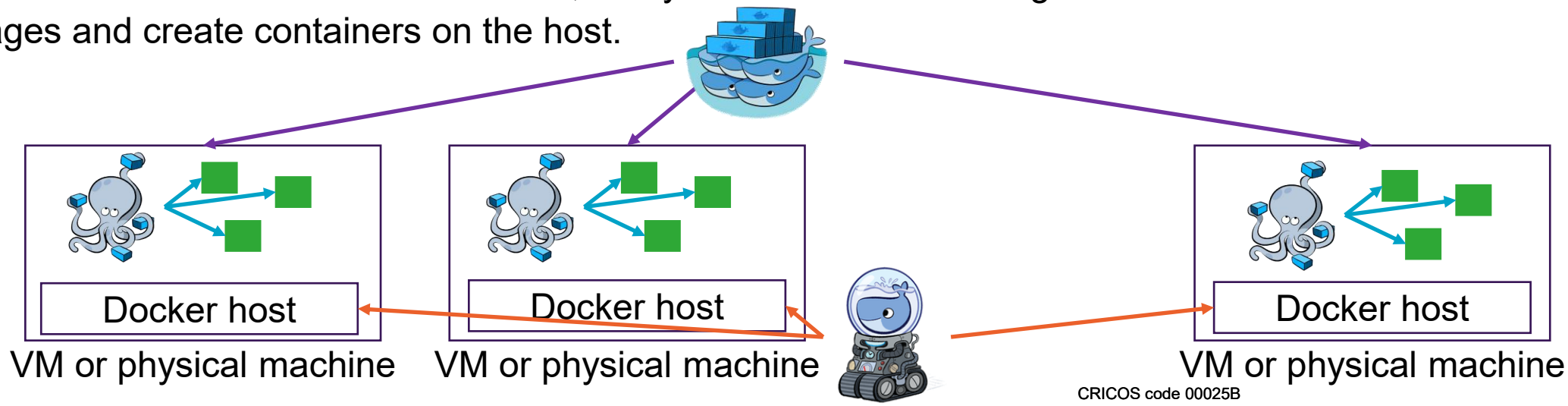


https://docs.docker.com/compose/compose-file/deploy/

Lecture 4: Docker II: Docker Compose and Docker Swarm

Docker Swarm & Docker Machine (***ID24, Lec4, 26-30,35,36***)

- **Docker Swarm** manages a cluster of Docker **nodes** and schedules containers
- Basically, Docker Swarm is an orchestration tool in charge of two things:
  - an enterprise-grade secure cluster of Docker hosts: Machines (aka nodes) joined the cluster
  - an engine for orchestrating microservices apps: Swarm manager controls the activities of the cluster
- **Docker Machine** allows you to provision Docker machines in a variety of environments,
- Docker Machine creates a Docker host, and you use the Docker Engine client as needed to build images and create containers on the host.



Lecture 4: Docker II: Docker Compose and Docker Swarm

Raft Consensus Algorithm (***ID25, Lec4, 31-34***)

What is a Raft Consensus Algorithm?

- In a distributed system (like a Docker Swarm cluster), multiple computers need to agree on a shared state, even if some of them fail.
- A consensus algorithm is the process they use to achieve this agreement.

Why Does Docker Swarm Need It?

- Docker Swarm's manager nodes are responsible for the entire cluster's state.
- All manager nodes must have the exact same information

Lecture 4: Docker II: Docker Compose and Docker Swarm

Raft Consensus Algorithm (***ID25, Lec4, 31-34***)

Three States for a Node:

- Follower: Passively waits for instructions. All nodes start as followers.
- Candidate: Tries to become the leader.
- Leader: The single, active manager of the cluster.

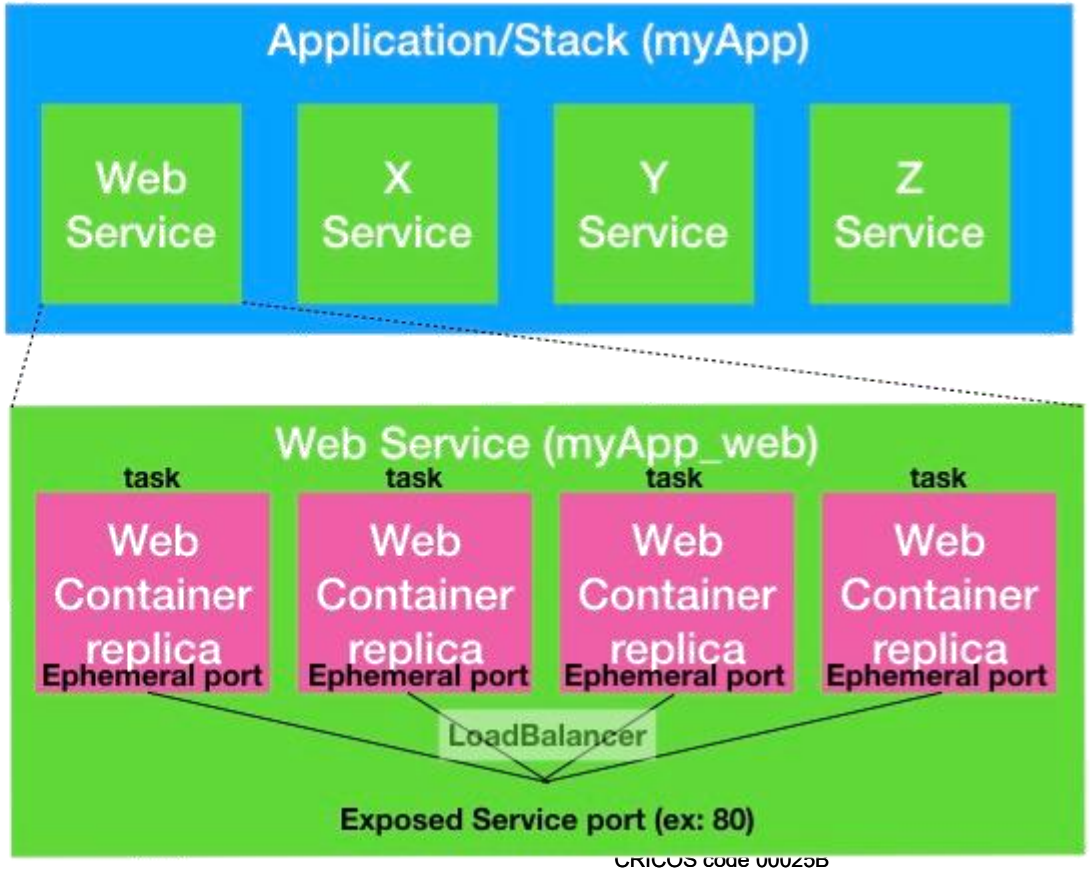
- Leader Election:
  - Followers start election if no heartbeat. Majority vote (>50%) elects Leader.
- Log Replication:
  - Leader appends commands, replicates to Followers, commits on majority.
- Fault Tolerance:
  - If Leader fails, new election; new Leader keeps committed logs.

Lecture 4: Docker II: Docker Compose and Docker Swarm

Deploy Services to a Swarm (***ID26, Lec4, 49-55***)

- Deploy a Stack to a Swarm

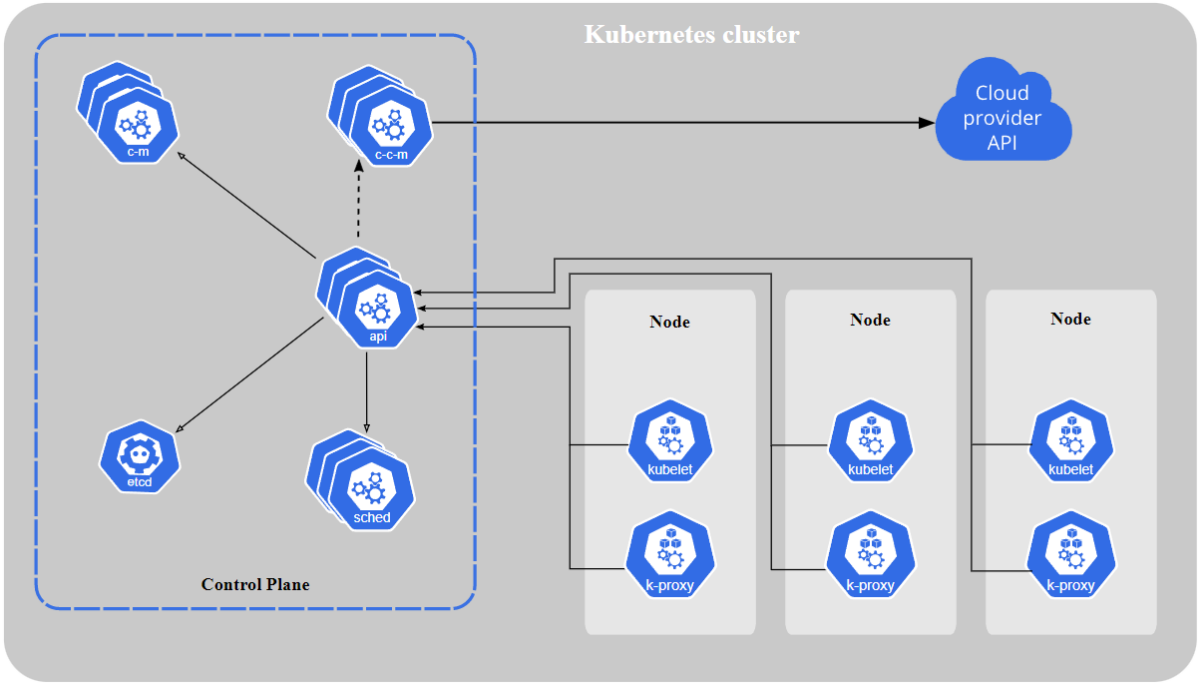
Describe the application in a yml
```
docker-compose.yml
```
Init host as a swarm host
```
docker swarm init
```
Deploy application
```
docker stack deploy -c docker-compose.yml myApp
```
List services
```
docker service ls
docker stack services myApp
```
List tasks
```
docker service ps myApp_web
docker container ls -q
docker stack ps myApp
```
Stop application
```
docker stack rm myApp
```
Take down swarm
```
docker swarm leave --force
```



Lecture 5: Kubernetes

Kubernetes Concepts and Components (***ID27, Lec5, 8-17***)

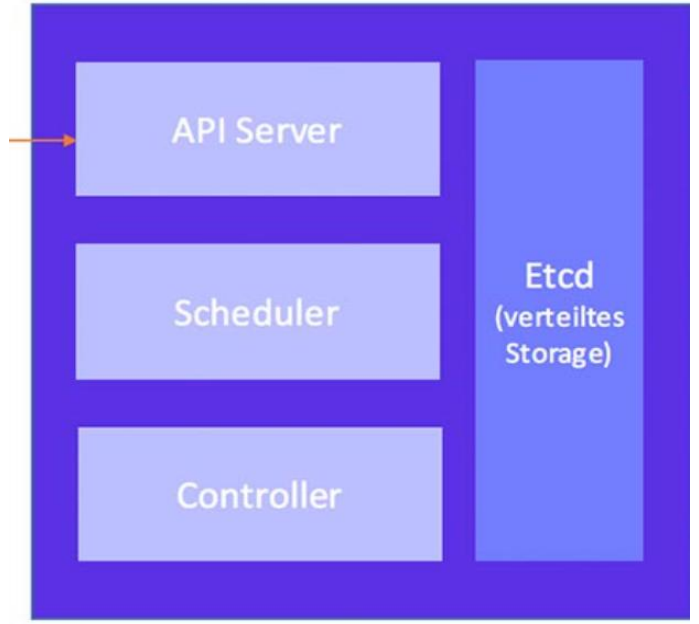
- Kubernetes (commonly stylised as K8s) is an **open-source container-orchestration system**
- K8s provides automated deployment, scaling and management of containerised applications.



Lecture 5: Kubernetes

Kubernetes Concepts and Components (***ID27, Lec5, 8-17***)

- **Control Plane (Master Node)**
  - Manages the (worker) nodes
    - Schedule (pod),
    - detect or respond to cluster events
  - Provides User Interface
  - FOUR processes need to run in Control Plane:
    - API Server, Scheduler, Controller Manager, Etcd
  - Multi-master high availability (**HA**) is a must-have
- **Worker Nodes (Slave Nodes)**
  - Conduct actual works with multiple pods
  - THREE processes need to run on every node
    - Container runtime, Kubelet, Kube proxy

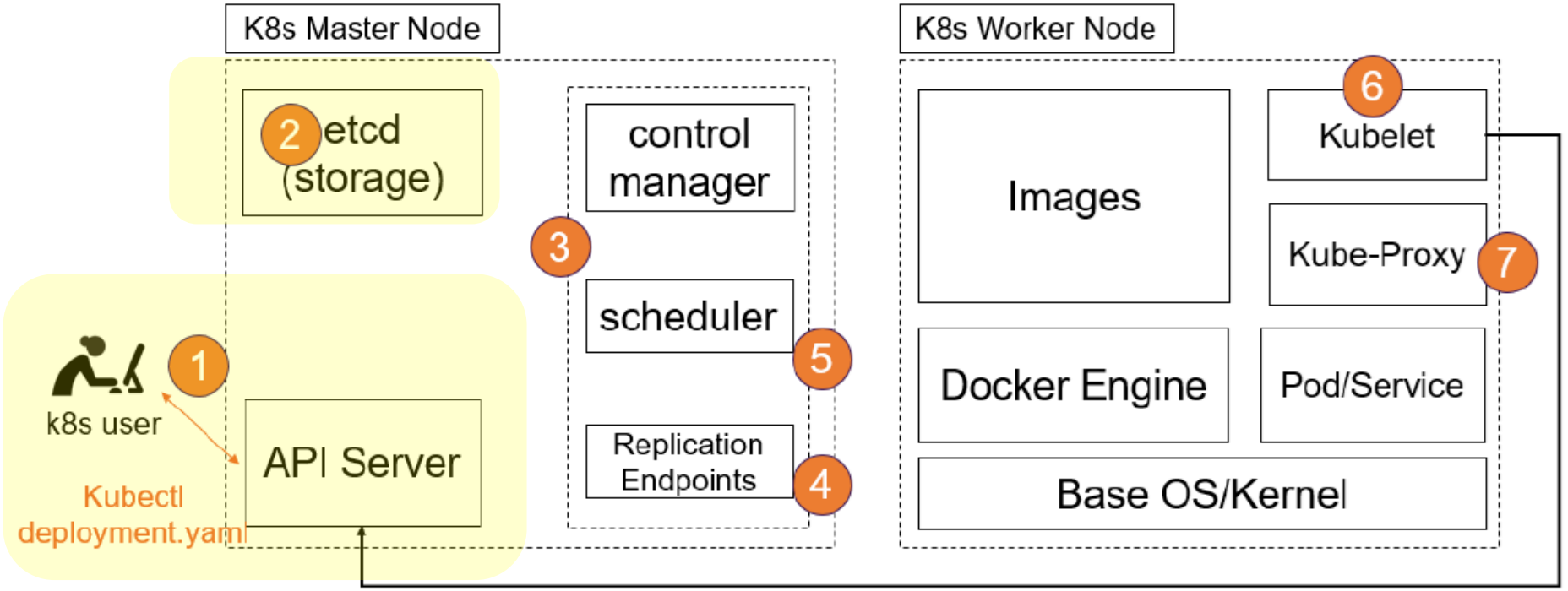


Lecture 5: Kubernetes

1. User via "kubectl" deploys a new application. Kubectl sends the request to the API Server.

Workflow (***ID28, Lec5, Tutorial 5 in Week 6***)

2. API server receives the request and stores it in the data store (etcd). Once the request is written to data store, the API server is done with the request.

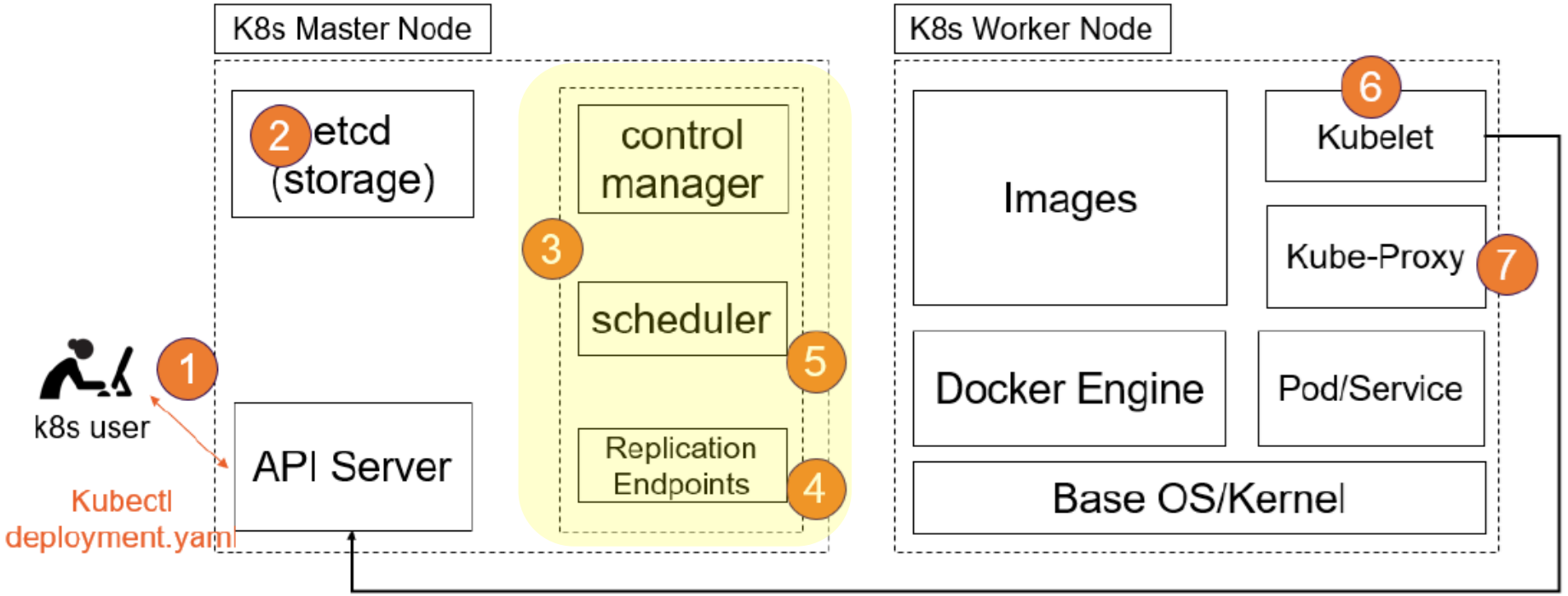


Lecture 5: Kubernetes

Workflow (***ID28, Lec5, Tutorial 5 in Week 6***)

3. Watchers detects the resource changes and send a notification to controller to act upon it

4. Controller detects the new app and creates new pods to match the desired number of instances. Any changes to the stored model will be picked up to create or delete Pods.



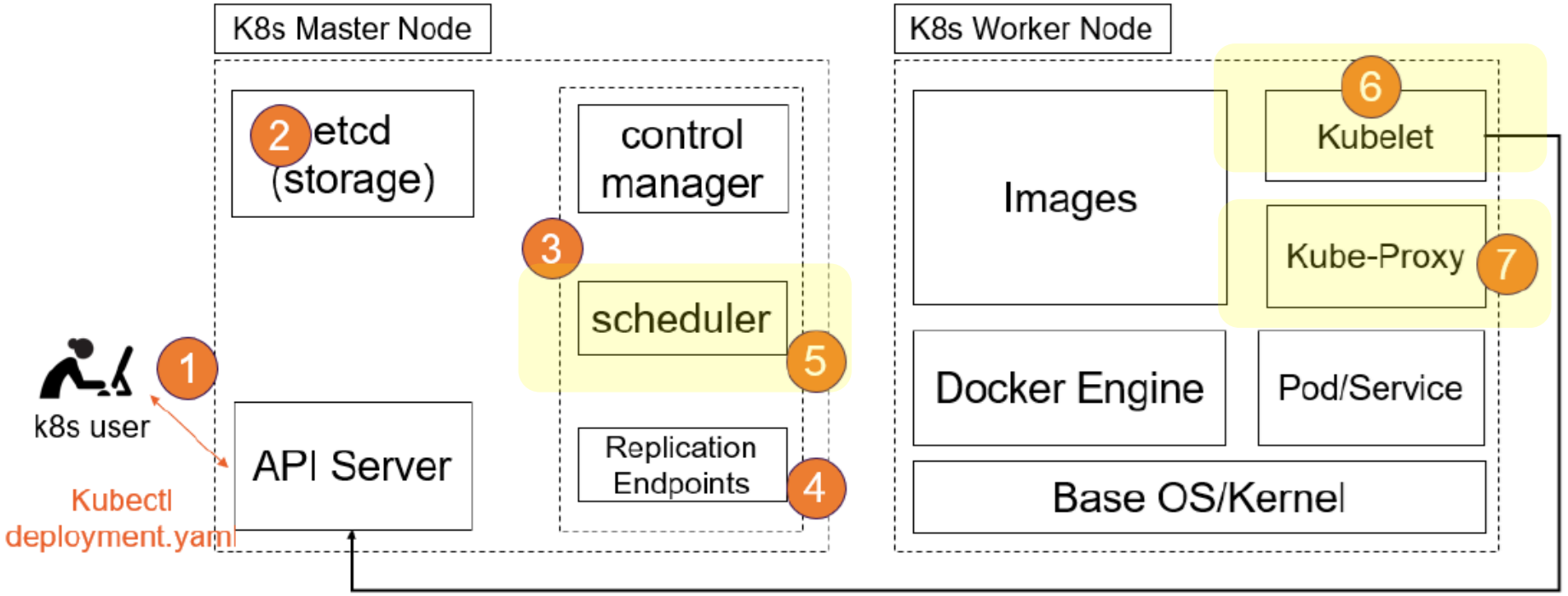
Lecture 5: Kubernetes

Workflow (***ID28, Lec5, Tutorial 5***)

5. Scheduler assigns new pods to a Node based on a criteria. Scheduler makes decisions to run Pods on specific Nodes in the cluster.

6. Kubelet on a node detects a pod with an assignment to itself, and deploys the requested containers via the container runtime (e.g. Docker).

7. Kubeproxy manages network traffic for the pods - including service discovery and load-balancing. Kubeproxy is responsible for communication between Pods that want to interact.



Lecture 5: Kubernetes

Reconciliation Loop (***ID 29, Lecture 5, 22-24***)

In Kubernetes, the Reconciliation Loop ensures that the system's actual state always matches the user's desired state. By continuously detecting and correcting differences, it provides self-healing and consistency in a dynamic, unreliable environment.

Goal: Make the cluster's current state match the desired state.

How it works:

- Observe – Check what's running now.
- Compare – See if it matches the YAML config.
- Act – Fix differences (e.g., start or stop pods).

https://kubernetes.io/docs/concepts/workloads/pods/

Lecture 5: Kubernetes

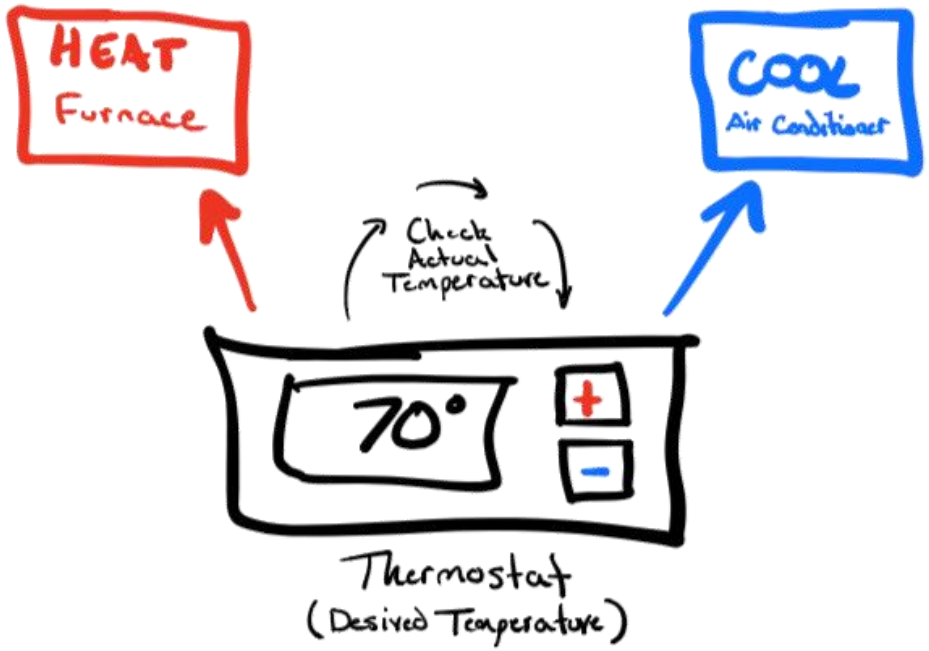
Reconciliation Loop (***ID 29, Lecture 5, 22-24***)

Desired State: You set the thermostat to 70°C. This is the state you want.

Current State: The thermostat's sensor constantly measures the actual room temperature, let's say it’s 70°C.

The Reconciliation Loop in Action:

- Observe: The thermostat reads the current temperature (60°C).
- Compare: It compares 60°C to your desired temperature of 70°C. They don't match.
- Act: Because the current temperature is too low, the thermostat sends a signal to turn on the heater.

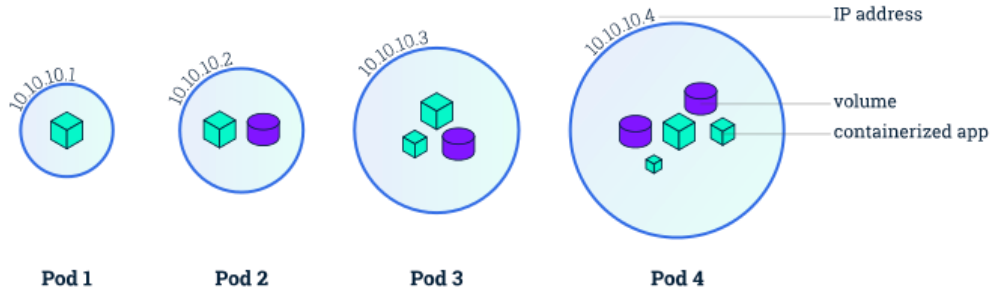


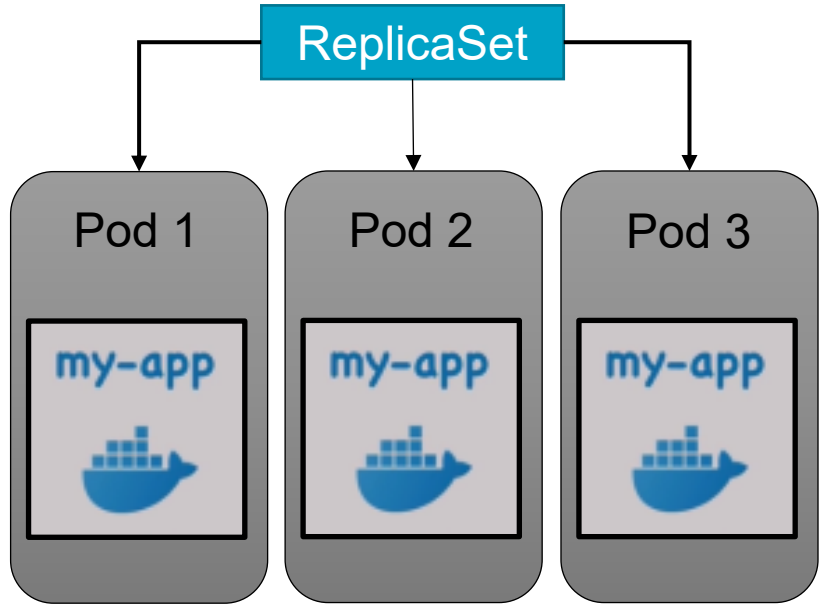
https://kubernetes.io/docs/concepts/workloads/pods/

Lecture 5: Kubernetes

Kubernetes Objects (***ID 30, Lecture 5, 33-66***)

- Pod
  - The atomic (smallest) unit of scheduling in K8s
  - Pods are ephemeral
  - The abstraction over a container(s)
  - Containers must **always** run inside Pods
  - One container per Pod vs multiple containers per Pod
  - Networking, resource sharing, etc.
- ReplicaSet
  - Maintains a stable set of replica Pods running at any given time (self-healing and scaling).
  - Created by Deployment



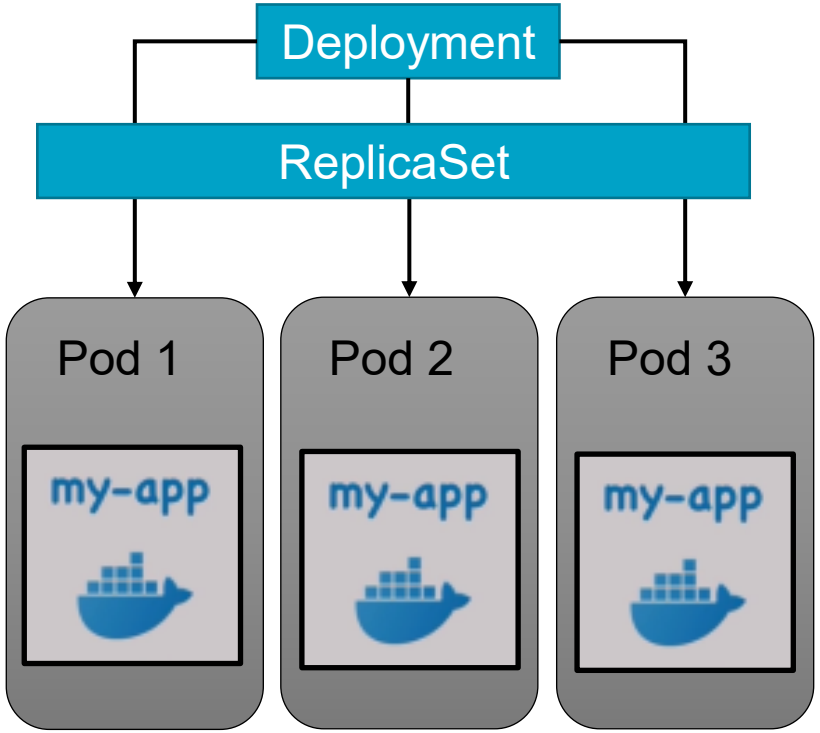


https://kubernetes.io/docs/concepts/workloads/pods/

Lecture 5: Kubernetes

Kubernetes Objects (***ID 27, Lecture 5, 29-62***)

- Deployment
  - provides declarative updates for Pods and ReplicaSets.
  - Blueprint for pods (including replica number of the pod)
  - You create deployments rather than pods (in practice no directly manipulating pods)
  - ReplicaSet is to maintain a stable set of replica Pods (identical) running at any given time.
  - Deployment can manage ReplicaSet (you may never need to manipulate ReplicaSet),
  - Use a Deployment directly and define your application in it.

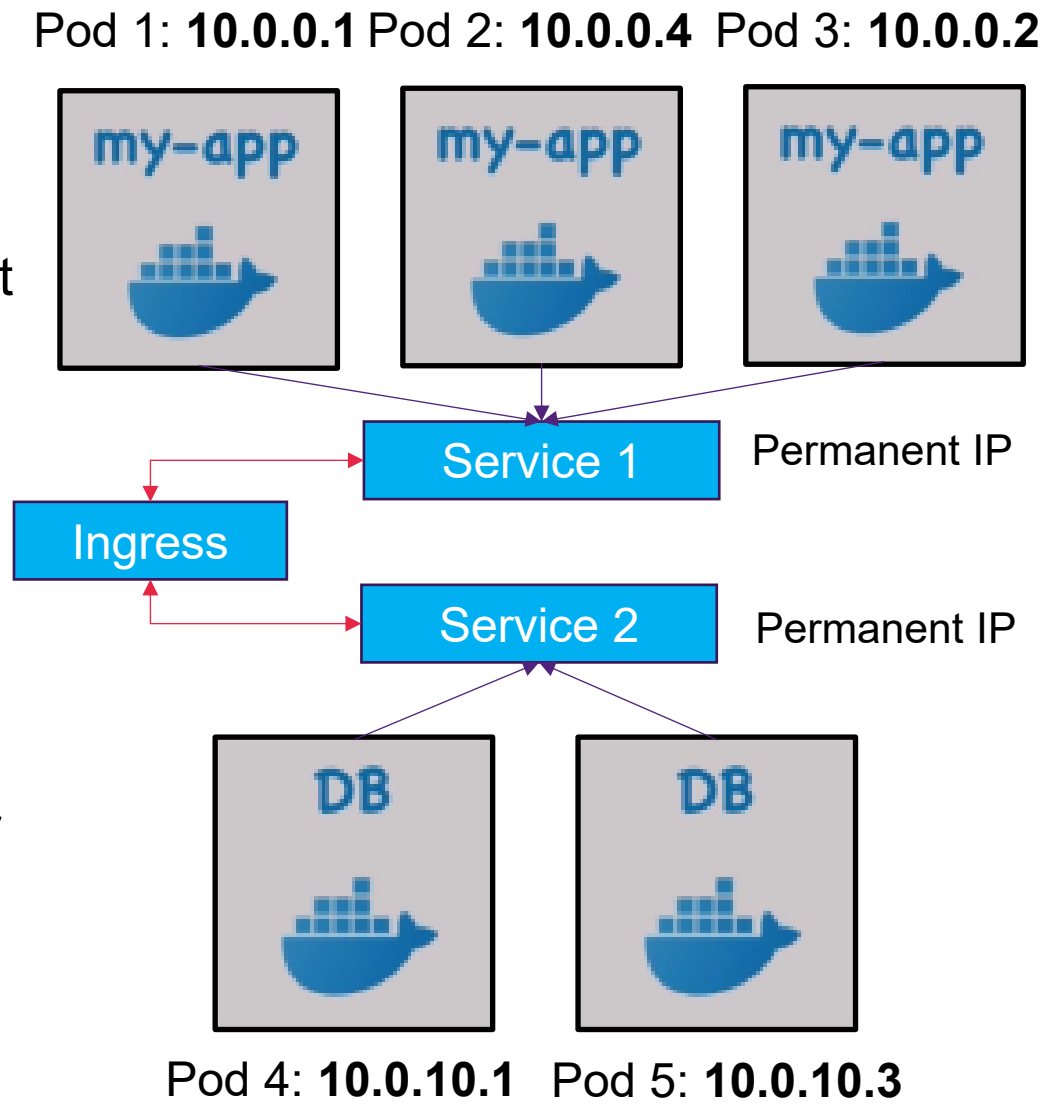


https://kubernetes.io/docs/concepts/workloads/pods/

Lecture 5: Kubernetes

Kubernetes Objects (***ID 27, Lecture 5, 29-62***)

- Services
  - Abstraction way to expose an app running on a set of pods as a networking services
  - Provide reliable networking for a set of Pods
  - Load-balancing requests across abstracted pods
  - Two ways of creating a Service
    - Expose ports & define services in a YAML file
  - Types of Services:
    - ClusterIP (default), NodePort, and LoadBalancer
- Ingress (DNS)
  - manages external access to the services in a cluster, typically HTTP.



Lecture 5: Kubernetes

Kubernetes Objects (*ID 27, Lec5, 29-62*)

	Design Role	Functionality	Usage
Deployment	Pod abstraction **Scalable** (up or down) Blueprint (template)	• Manages the **desired state** for your Pods and **ReplicaSets**. • Allows you to perform rolling updates and rollbacks to your application.	• Ensures a certain number of instances of your application are running; to update your application, or to rollback an update. • Deployment does **NOT** deal directly with **networking or exposing** your application.
Service	Pod abstraction Load-balancer (internal & external)	• Provides a **stable endpoint** (IP address and port) to communicate with a set of Pods, typically exposed by a Deployment. • Abstracts away the **ephemeral nature** of Pods, which might be created and destroyed frequently.	• Needs **a persistent way** to access your application, irrespective of the changes in the Pods (like creation, deletion, scaling). • Service deals with networking and can expose your application **internally** (within the cluster) or **externally** (to the internet or outside world).

Lecture 5: Kubernetes

Kubernetes Objects (***ID 27, Lec5, 29-62***)

- ConfigMap
  - Used to store non-confidential data in key-value pairs.
  - Consumed (by Pods) as environment variables, command-line arguments, or as configuration files in a volume.
- Secret
  - Similar to ConfigMap, but is used to store secret data (user name, password, tokens, keys, etc)
  - Independently created (less risk of being exposed)
  - base64 encoded.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

Configmap.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
type: Opaque
data:
  mysql-root-pwd: cm9vdHB3ZA==
```

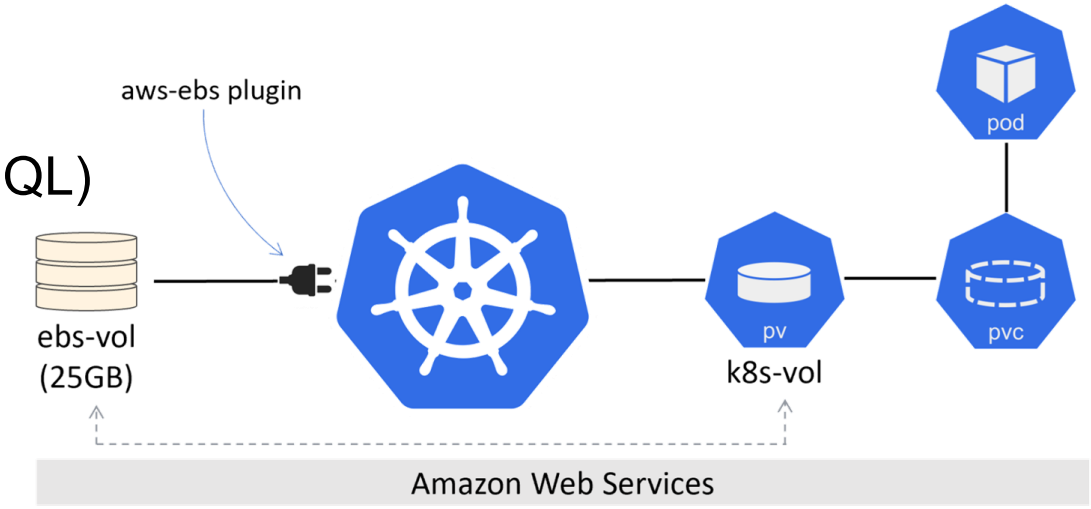
Secret.yaml

Lecture 5: Kubernetes

Kubernetes Objects (***ID 27, Lec5, 29-62***)

- Persistent Volume (PV):
  - how you map external storage onto the cluster; resource (admin/ops)
- Persistent Volume Claim (PVC):
  - a request for storage by a user; claim checks to the resource (user/dev)
- StatefulSet:
  - manages stateful applications (e.g. SQL & NoSQL)
  - deploys and scales a set of Pods (DBs)
  - provides guarantees about the ordering and uniqueness of these Pods.

1. Create the PV.
2. Create the PVC.
3. Define the volume.
4. Mount it into a container.



Goto VecDB

Lectures 6 and 7: NoSQL and Vector DBs in CC

ID	Topics	Pages	Lecture
31	NoSQL	23-24	6
32	Types of NoSQL	26-35	6
33	Database Partition	35-38	6
34	CAP Theorem	39-42	6
35	BASE in NoSQL	43-44	6
36	Comparison between NoSQL, and to SQL	80, T6 Q1	6
37	Vector Database	4	7
38	Vector Database Advantages	6	7
39	Indexing Techniques for Vector Databases	23-33	7
40	Types of Vector Databases	35-36	7
41	Retrieval Augmented Generation	45-46	7

Lecture 6: Databases in Cloud Computing

NoSQL (***ID31, Lec6, 25-26***)

- **A class of database management systems (DBMS), which**
  - does not use SQL as a querying language
  - is distributed, fault-tolerant architecture
  - has no fixed schema (formally described structure)
  - No **joins** (typical in databases operated with SQL)
  - **It's not a replacement for an RDBMS but compliments it**
- Support large numbers of concurrent users (tens of thousands, perhaps millions)
- Deliver highly responsive experiences to a globally distributed base of users (horizontally scalable)
- Be always available – no downtime
- Handle semi- and unstructured data

Lecture 6: Databases in Cloud Computing

Types of NoSQL (***ID32, Lec6, 27-37***)

- Key-value stores
  - Key is a string while value can be in different types
  - Representative Products: Redis, Riak, Memcached, etc.
- Document stores
  - Similar to key-value stores, but value is a document (nested values)
  - Representative products: MongoDB, CouchDB, etc.
- Wide-Column (Column-Family) stores
  - A column family is a collection of rows and columns, where each row has a unique key and each column has a name, value, and timestamp.
  - Representative products: BigTable, Hbase, **Cassandra**, etc.
- Graph databases
  - Nodes and relationships (edges) are the bases of graph databases.
  - Representative products: Neo4j, OrientDB, InfoGrid, etc.

Lecture 6: Databases in Cloud Computing

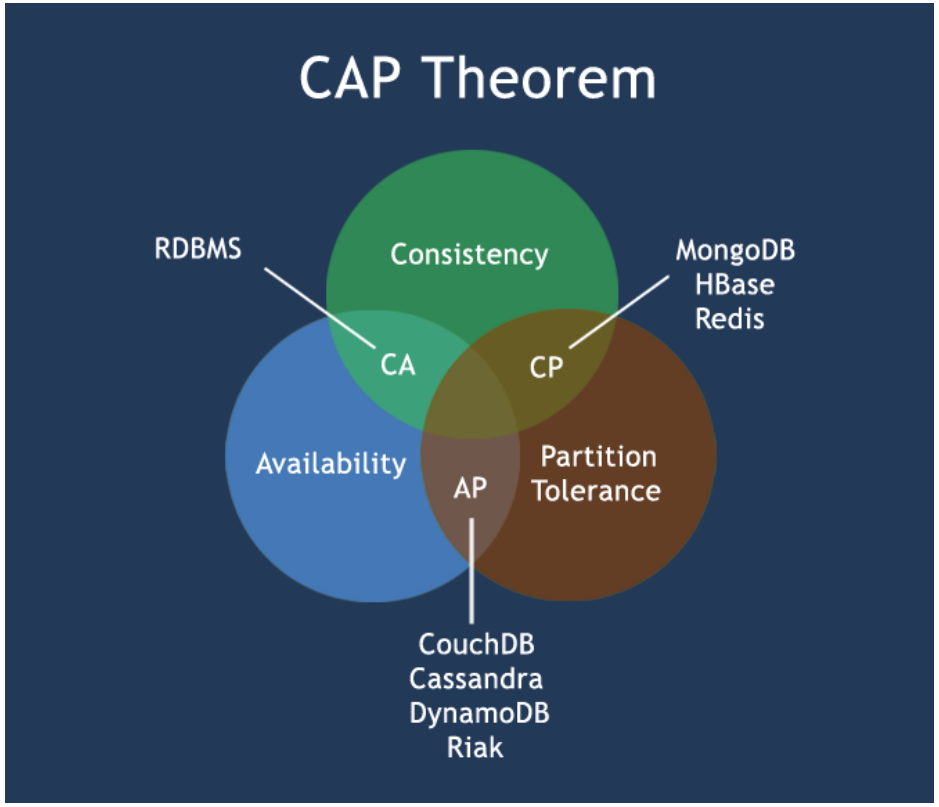
Database Partition (***ID33, Lec6, 38-40***)

- A database can be logically divided into multiple partitions.
- Partitions are distinct and independent parts that spread over multiple nodes.
- Popular partition methods in distributed DBMS:
  - Vertical Partitioning
    - Each partition holds a subset of the fields for items in the data store. (SQL vs NoSQL)
    - Frequent fields are placed in one vertical partition (e.g. ProductName)
  - Horizontal Partitioning (or Sharding)
    - Each partition (aka a shard) is a separate data store (a subset of the entire database)
- Database partitioning aims at improvements in
  - Scalability & Performance & Availability
  - Security & Operational flexibility

Lecture 6: Databases in Cloud Computing

CAP Theorem (***ID34, Lec6, 41-44***)

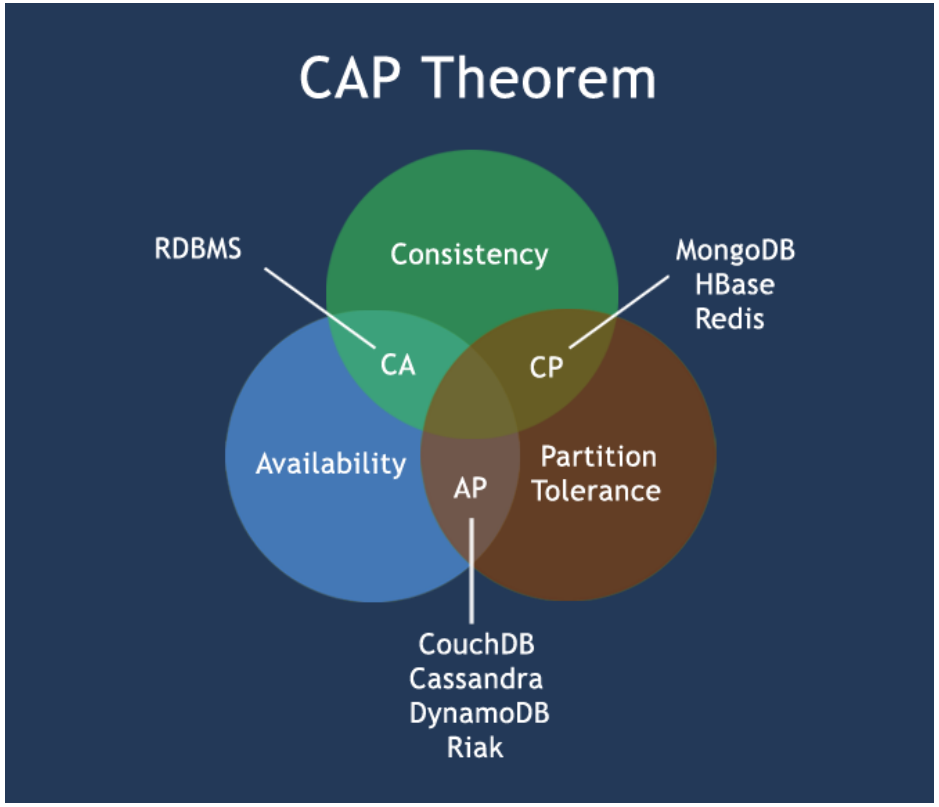
- The CAP theorem states that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:
  - **C**onsistency: Every read receives the most recent write or an error
  - **A**vailability: Every request receives a (non-error) response – without the guarantee that it contains the most recent write
  - **P**artition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes



Lecture 6: Databases in Cloud Computing

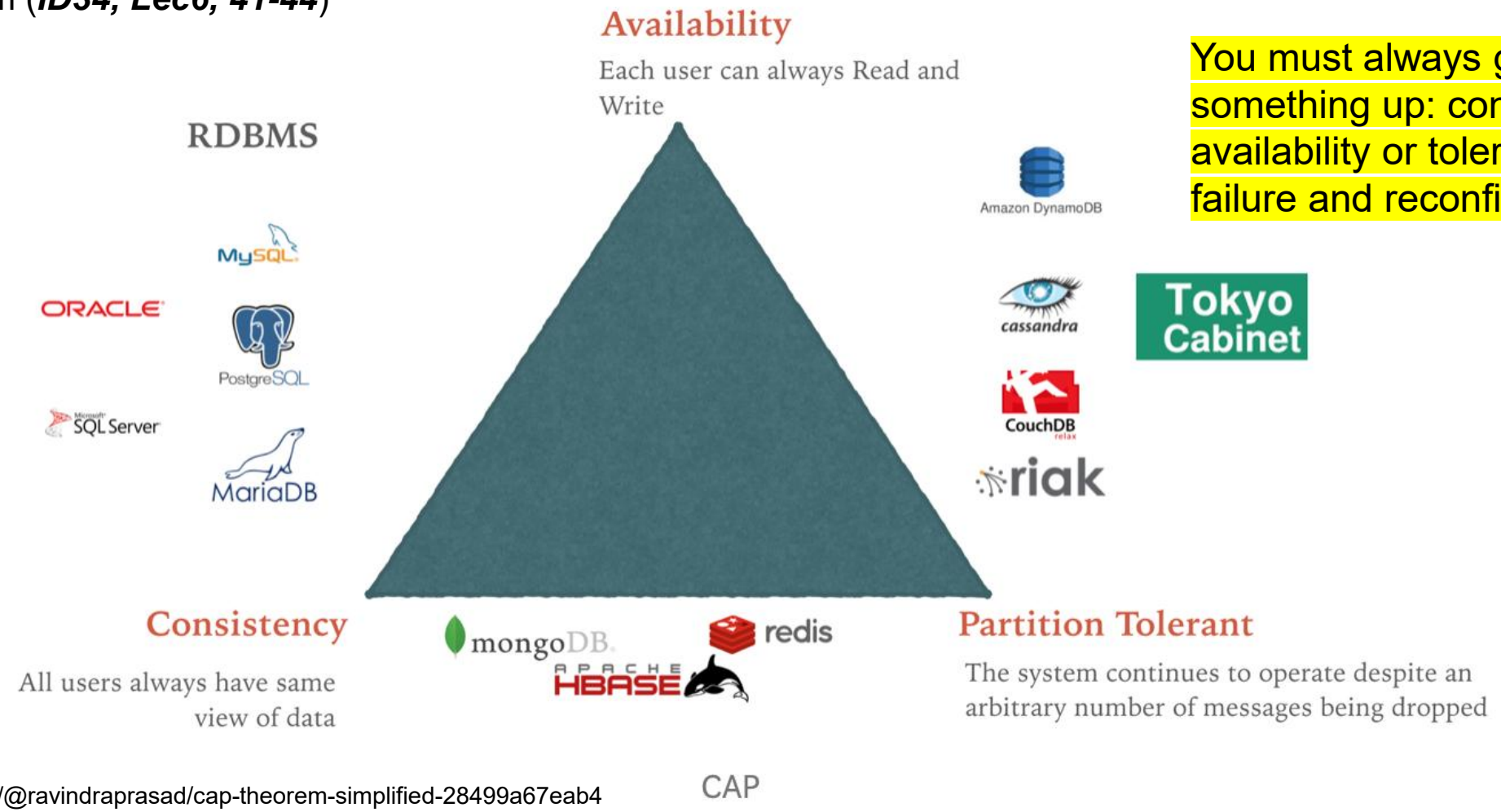
CAP Theorem (***ID34, Lec7, 41-44***)

- **CA** (*Consistency & Availability*)
  - All clients always have the same view on the data
  - Each Client can always read and write
  - The system may not tolerance to failure and reconfiguration
- **AP** (*Availability & Partition Tolerance*)
  - Each Client can always read and write
  - The system works well despite the physical partitions
  - Clients may have inconsistent views on the data
- **CP** (*Consistency & Partition Tolerance*)
  - All clients always have the same view on the data
  - The system works well despite the physical partitions
  - Clients sometimes may not be able to access data



Lecture 6: Databases in Cloud Computing

CAP Theorem (***ID34, Lec6, 41-44***)



You must always give something up: consistency, availability or tolerance to failure and reconfiguration

https://medium.com/@ravindraprasad/cap-theorem-simplified-28499a67eab4

Lecture 6: Databases in Cloud Computing

BASE in NoSQL (***ID35, Lec6, 45-46***)

- The opposite of ACID for transactions in relational databases is BASE:
  - **B**asically **A**vailable, **S**oft-state, **E**ventually consistent
- **B**asically **A**vailable:
  - one distributed system has failed parts, but the total system is still working properly.
  - E.g. online shopping on Black Friday
- **S**oft-State: corresponds to hard-state, which guarantees consistency and durability in RDBMS and allows delays or outages (short period)
- **E**ventually consistent:
  - Strong consistency: the data should be consistent after every transaction.
  - Rather than requiring consistency after every transaction, it is enough for the distributed database to eventually be in a consistent state.

Choose the appropriate NoSQL for your apps

Scenario 1: Social Media "Likes" Counter

Requirement: The system must be always available for users to like a post, even if some data centers face issues. An exact, immediate count isn't necessary; it's okay if different users see slightly different like counts for a brief period.

Scenario 2: E-commerce Product Catalog - An online shopping site displays a catalog of products.

Requirement: The product catalog should be available at all times for users to browse, even if there are network issues. It's acceptable if some users don't see the very latest products immediately after they're added.

Scenario 3: E-commerce Transaction System.

Requirement: Every transaction needs to be accurately reflected in the account balance. Even if there's a network partition, it's preferable to deny transactions rather than process them without ensuring consistency.

Scenario 4: Airline Seat Reservation System - An online system for booking airline seats.

Requirement: It's vital to ensure that the same seat isn't booked by multiple passengers. If there's uncertainty due to a network partition, it's better to temporarily halt bookings rather than double-book a seat.

Lecture 6: Databases in Cloud Computing

Comparison of NoSQL (***ID36, Lec6, 82***)

	Cassandra	HBase	MongoDB	Redis
CAP	AP	CP	CP	CP
Network	Masterless (P2P)	Master /Slave	Master/Slave	Master/Slave
Data Store	Wide-column	Wide-column	Document	Key-value
Architecture	Peer-to-peer	Hierarchical	Nested (Hierarchical)	
Data Lake	Too Complex	Best Choice	OK	not typically used as a data lake.
IoT or Web	Best Choice	Lack of Record-Level Indexing	Best Choice	suitable for real-time analytics for IoT and web.
Text Data	Good	Good	Good	Good
Schema	Yes (can replace RDB)	No	No	No (schema-less)
AWS	Amazon offers "Amazon Keyspaces	Amazon EMR (Elastic MapReduce)	Amazon DocumentDB	Amazon ElastiCache
GCP	No native support, but can be deployed on GCP	Google's Bigtable	GCP **does not** have a native MongoDB, but "MongoDB Atlas", can be deployed on GCP.	Cloud Memorystore
Azure	Azure Cosmos DB" provides a Cassandra API	Azure HDInsight	Azure Cosmos DB provides MongoDB API	Azure Cache

Lecture 6: Databases in Cloud Computing

Comparison between NoSQL and SQL (***ID36, T6 Q1***)

Tutorial 6: Databases in Cloud Computing

Question Set:

Q1. What are the differences between SQL and NoSQL?

SQL	NoSQL
Table-based databases	Document-based, key-value pairs, graph databases or wide-column stores
Predefined schema	Dynamic schema for unstructured data
Use structured query language for defining and manipulating the data	Queries are focused on collection of documents
MySQL, Oracle, SQLite, Postgres and MS-SQL	MongoDB, BigTable, Redis, RavenDB, Cassandra, Hbase
Not best fit for hierarchical data storage	Fits better for the hierarchical data storage
SQL databases are the best fit for heavy-duty transactional type applications	As compared to SQL, NoSQL is not best fit for heavy-duty transactional type applications
Follows ACID properties	Follows CAP theorem
Excellent support is available for all SQL database from their vendors	Limited support and sometime depend on community support

Goto Spark

Lecture 7: Vector Databases

Vector databases(***ID37, Lec7, 4***)

A vector database is any database that allows you to store, index, and query vector embeddings, or numerical representations of unstructured data, such as text, images, or audio.

https://cloud.google.com/discover/what-is-a-vector-database

Lecture 7: Vector Databases

Vector database advantages (***ID38, Lec7, 6***)

Data perspective

- Support complex data, e.g., *geospatial data*, *genomic data*, *social media likes*
- Embedding unstructured data - 80% of the world's data is unstructured
- High-dimensional data handling

Method perspective

- Efficient and accurate similarity search and retrieval
- Sophisticated query capabilities

Application perspective

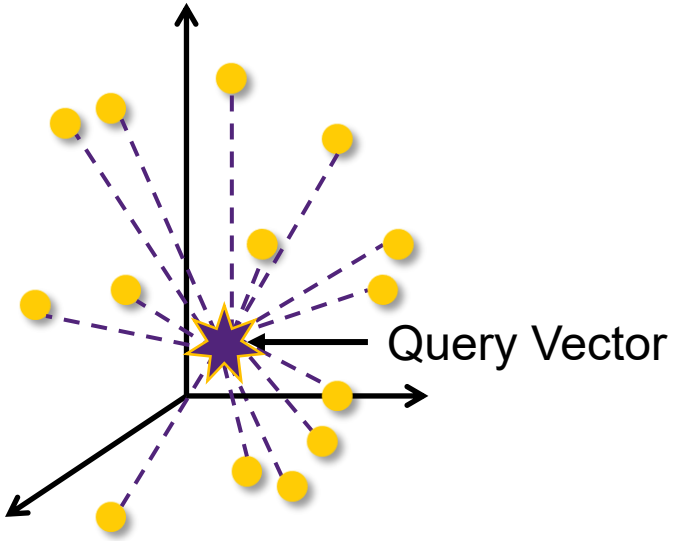
- AI and machine learning integration
- Scalability for modern applications, e.g., biology, healthcare, e-commerce, etc.

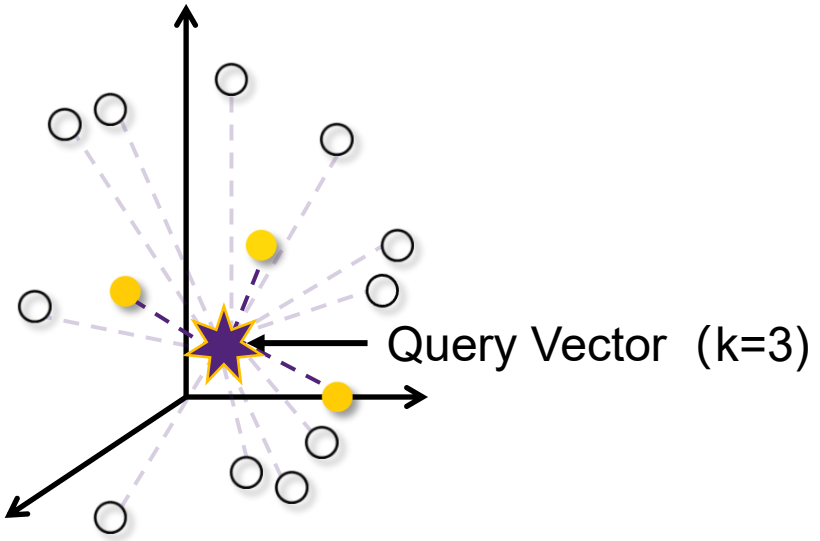
Lecture 7: Vector Databases

Vector database Indexing (***ID39, Lec7, 23-33***)

Flat Index (Brute Force**)**

- Flat indices are a *direct representation* of the vector embedding
- Deliver the best accuracy of all indexing methods, but notoriously slow
- Search is *exhaustive:* it's performed from the query vector across *every single vector embedding* and distances are calculated for each pair.





Lecture 7: Vector Databases

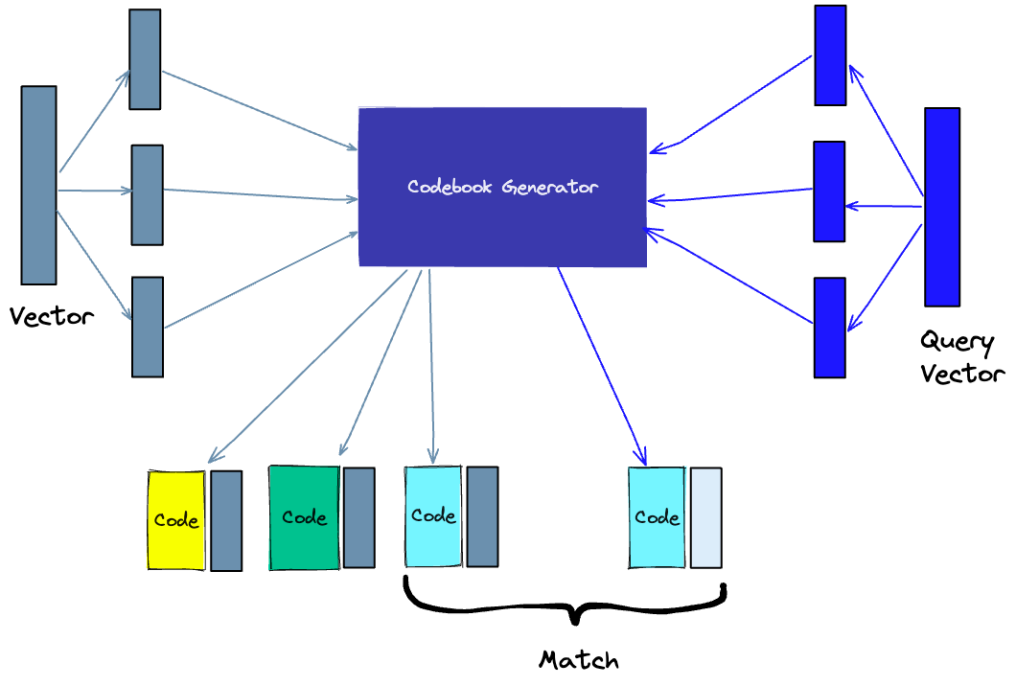
Flat Index (Brute Force**)** is desirable when**:**

- **Low-dimensional data**: With low-dimensional vectors, typically up to a few dozen dimensions, flat indices can be sufficiently fast while maintaining high accuracy.
- **Small-Scale Databases**: When the database contains a relatively small number of vectors, a flat index can be sufficient to provide quick retrieval times.
- **Simple Querying**: If the primary use case involves simple queries, a flat index can offer competitive performance compared to other indices without the complexity
- **Real-time Data Ingestion**: When vectors are continuously added to the database, they must be indexed quickly. Due to the simplicity of flat indexing, minimal computation is needed to generate the new indices.
- **Low Query Volume**: If the rate of queries being performed on the database is low, a flat index can handle these queries effectively
- **Benchmarking Comparisons**: In situations where you want to evaluate the accuracy of other index methods, using the perfectly accurate flat index can be used as a benchmark for comparison purposes.

Lecture 7: Vector Databases

Inverted File Product Quantization Index:

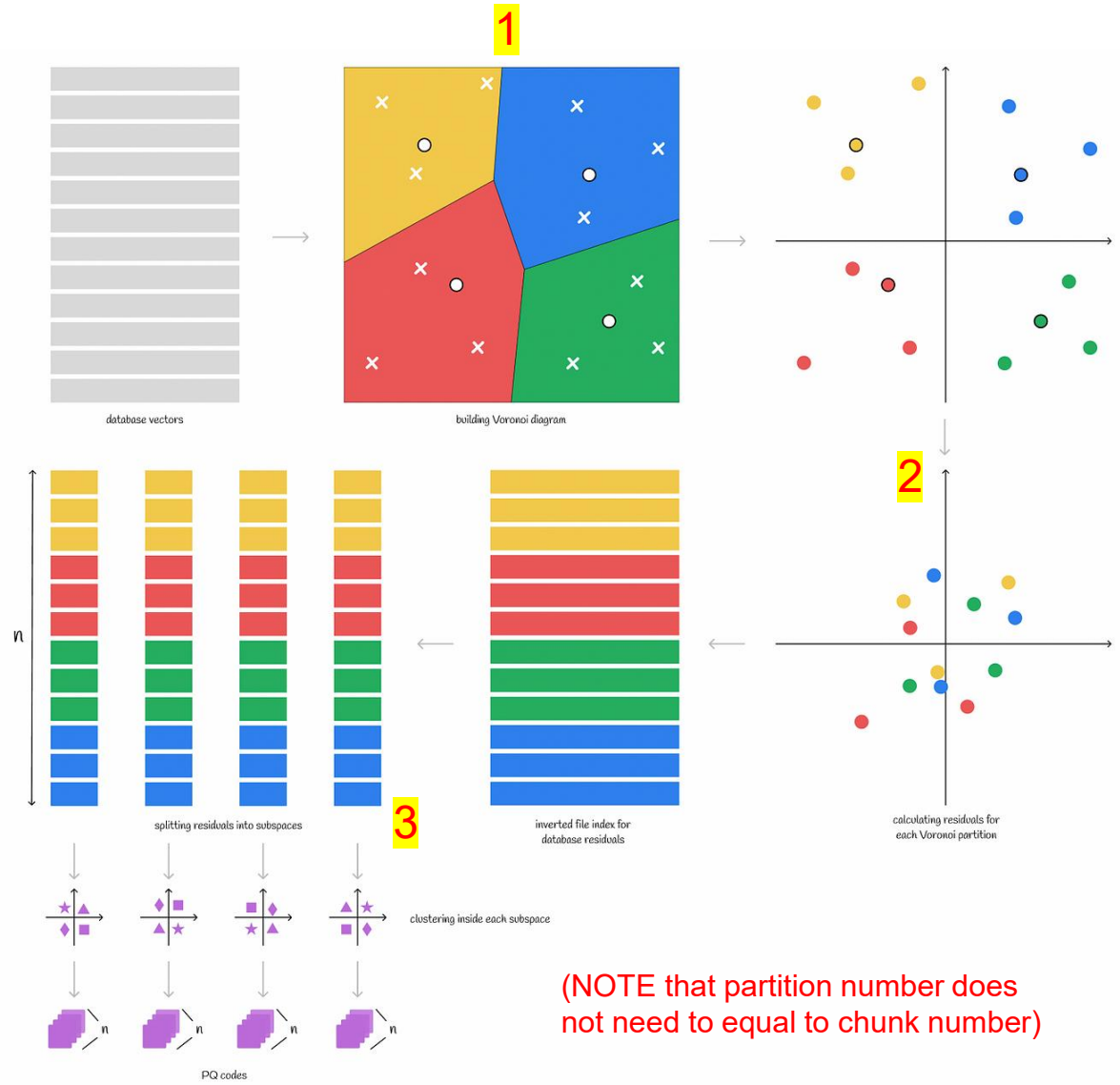
- A ***lossy*** compression technique for high-dimensional vectors
- It takes the original vector, breaks it up into sub-vectors (smaller chunks**)**, simplifies the representation of each chunk by creating a representative “code” for each chunk, and then puts all the chunks back together - without losing information that is vital for similarity operations.



Lecture 7: Vector Databases

Inverted File Product Quantization Index:

- ***Training*** - we build a "codebook" for each chunk.
  1. An **inverted file index** is constructed: dividing the set of vectors into k Voronoi partitions. (reduce the search space)
  2. Inside each Voronoi partition, each vector subtracted the coordinates of its **centroid vector**.
  3. The **product quantization algorithm** is run on vectors from all the partitions:
     a. Splitting high-dimensional vectors into smaller **sub-vectors**.
     b. Quantizing these **sub-vectors** into a finite number of **clusters or bins**, each represented by a **centroid**.
     c. Using a codebook for each sub-vector to enable efficient **storage** and approximate distance calculation.

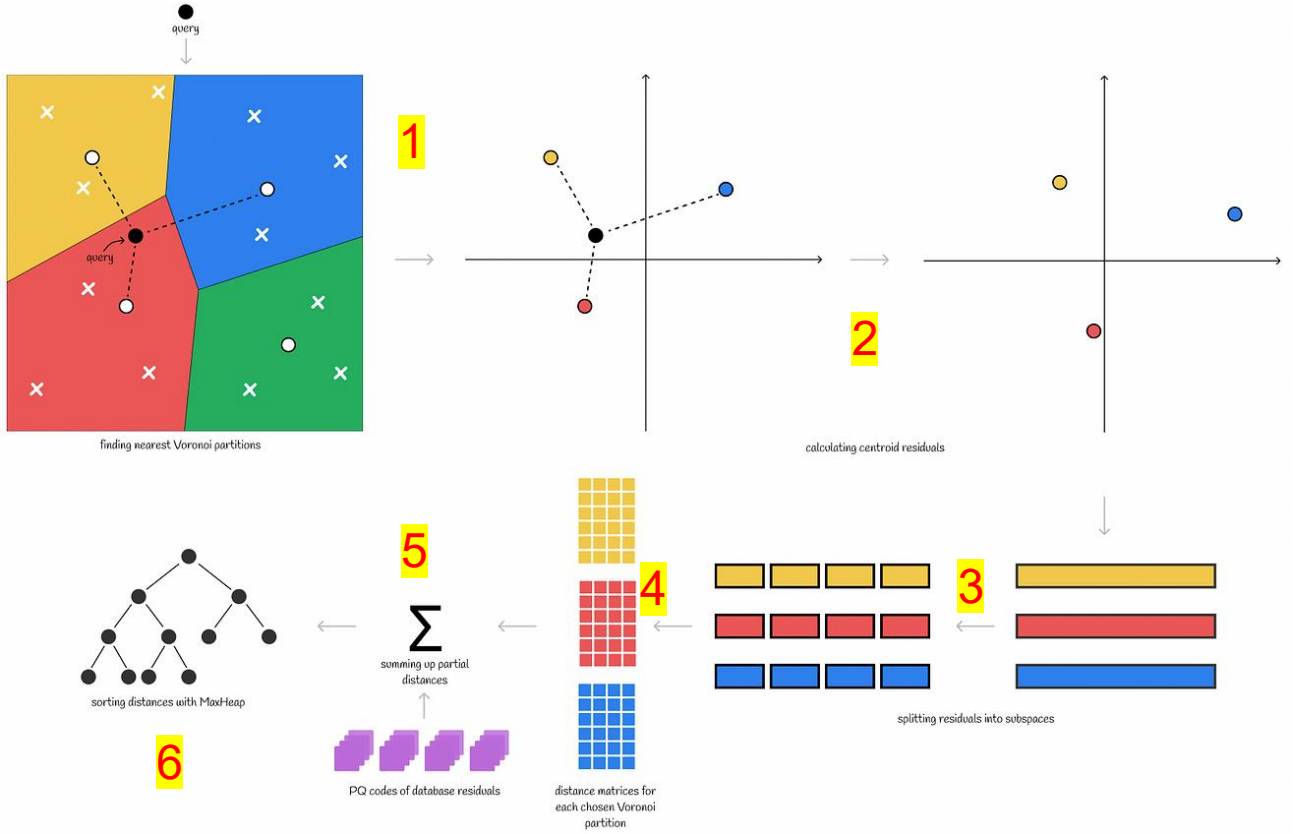


(NOTE that partition number does not need to equal to chunk number)

Lecture 7: Vector Databases

Inverted File Product Quantization Index:

- ***Inference***
  1. Find k nearest **centroids** of Voronoi partitions.
  2. Calculate the **query residual** separately for k Voronoi partitions
  3. The **query residual** is then split into **subvectors**.
  4. Calculate the **partial distance** to each partition
  5. Partial distances are **summed up**.
  6. Maintaining a **MaxHeap** data structure for **sorting** the results. (Store n smallest structure at each step)



Lecture 7: Vector Databases

Types of Vector Databases (***ID40, Lec7, 35-36***)

In-memory vector database (RedisAI, Torchserve**)**

- Store vectors directly in memory
- Enable swift read-and-write operations
- Support real-time analytics and recommendation systems

Disk-based vector database (Annoy, Milvus, ScanNN**)**

- Store vectors on disk
- Suitable for large data sets
- Using indexing and compression techniques

Distributed vector database (FAISS, Elastics Search With Vector Plugin, Dask-ML**)**

- Spread vector data across multiple nodes or servers
- Allows for horizontal scalability and fault tolerance
- Suitable for managing massive data sets and high-throughput tasks

Lecture 7: Vector Databases

In-memory vector database

Disk-based vector database

Distributed vector database

Graph-based vector database (Neo4j, TigerGraph, Amazon Neptune**)**

- Model data as a graph
- Nodes and edges represent vector attributes or embeddings
- Excel at capturing complex relationships
- Facilitate graph analytics

Time-series vector database (InfluxDB, TimescaleDB, Prometheus**)**

- Represent data collected over time as vectors
- Serve as tools for temporal pattern and anomaly analysis

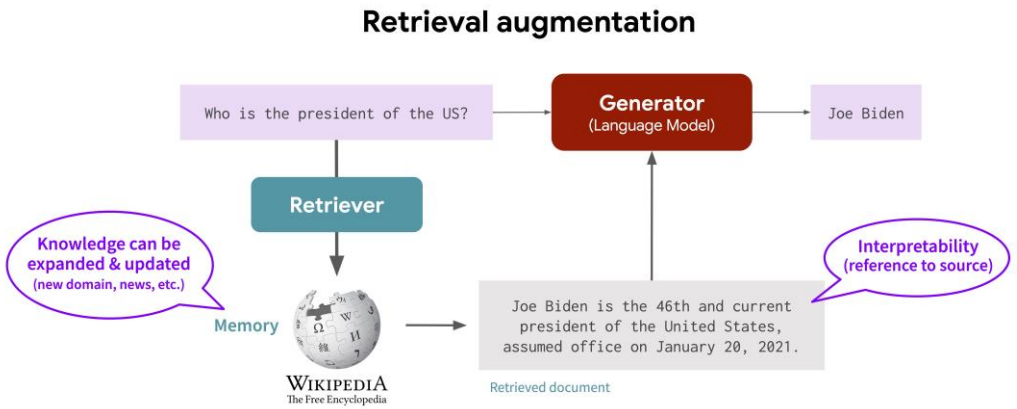
Lecture 7: Vector Databases

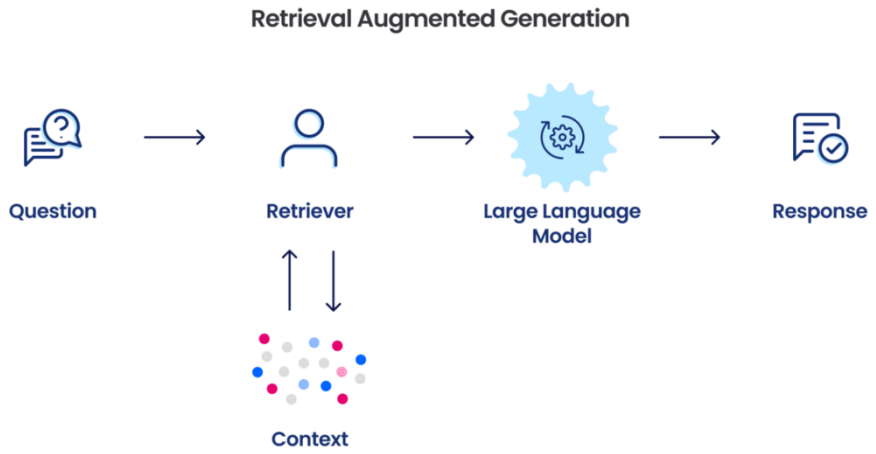
Retrieval Augmented Generation (***ID41, Lec7, 45-46***)

Definition:

- RAG is a method that combines *retrieval-based models* with *generative models* to enhance the generation of *contextually relevant and accurate* responses.

- RAG uses a retrieval mechanism to fetch relevant documents or pieces of information which are then used to condition the generative model.

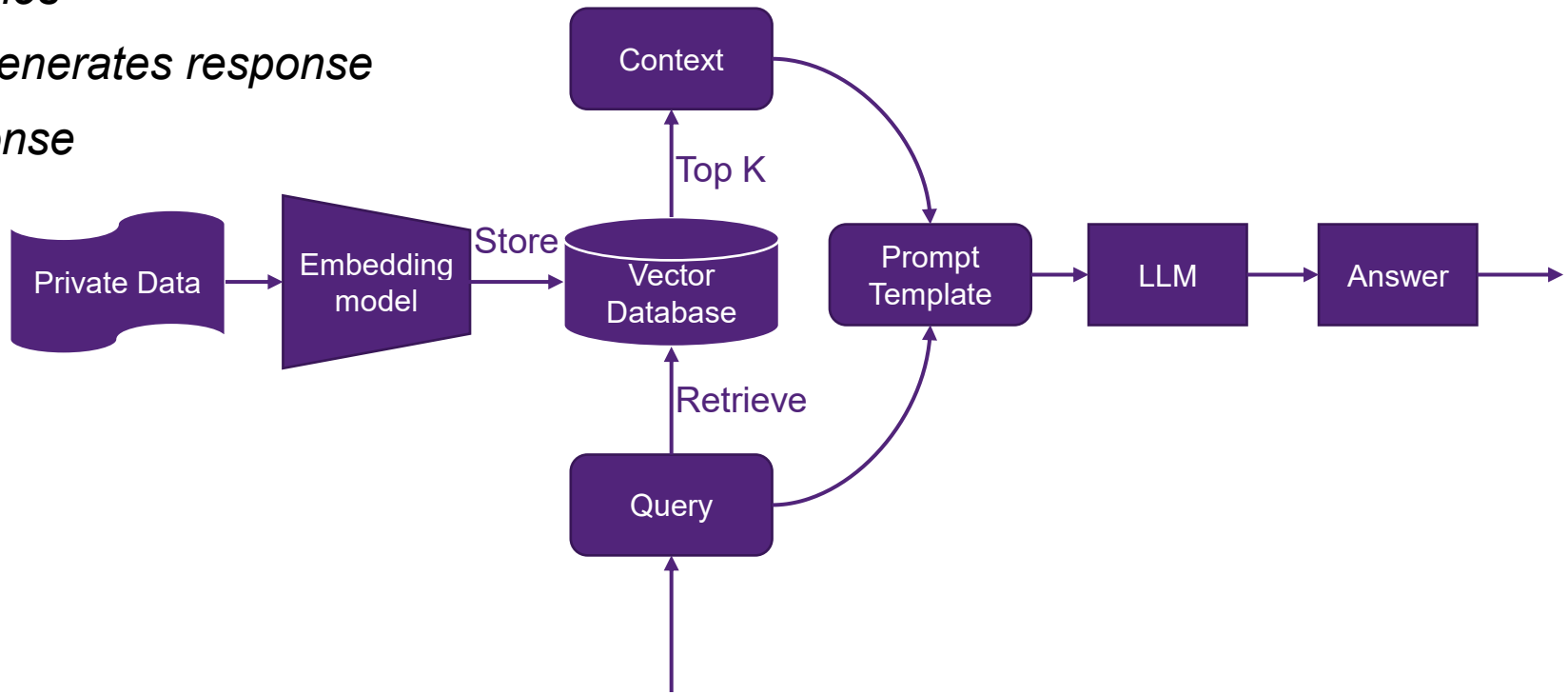




Lecture 7: Vector Databases

Generalized RAG Approach

- *Step 1 – User sends query*
- *Step 2 – App forwards queries*
- *Step 3 – RAG retrieves + generates response*
- *Step 4 – LLM returns response*



Goto End

Lectures 8 & 9: Spark and RDD Programming

ID	Topics	Pages	Lecture
42	Spark and its characteristics	9-21	8
43	RDD and its operations	23-28	8
44	Lazy Evaluation	30	8
45	RDD Lineage Graph	31	8
46	RDD Persistence and Caching	33-34	8
47	Terms in Spark	36-41	8
48	Directed Acyclic Graph	43-44	8
49	Narrow and Wide Dependencies	46-47	8
50	Shuffle	49-51	8
51	Spark Workflow	57	8
52	RDD Programming Fundamentals	5-44	9
53	RDD Programming Examples	46-55	9
54	RDD vs DataFrame	Tut 9 (Week 10)	9

Lecture 8: Introduction to Spark Framework

Spark and its characteristics (***ID42, Lec8, 9-21***)

- Apache Spark is an open-source distributed general-purpose cluster-computing framework.
- Spark aims to be fast: in-memory computing
- Simple differences: Spark vs MapReduce
  - Much faster (in-memory, low latency)
  - Wider range of workloads (e.g. iterative algorithms)
- **Characteristics**:
  - Speed (in memory)
  - Ease of use (more operations than MapReduce, four languages supported)
  - A Unified Stack (spark core, SQL, ML, Stream, GraphX)
  - Runs Everywhere
    - works with many clusters: Standalone, MESOS, YARN
    - work with many data sources: HDFS, Datawarehouse, NoSQL/SQL database)

Lecture 8: Introduction to Spark Framework

RDD and its operations (***ID43, Lec8, 23-28***)

- RDD is a fundamental data structure of Spark.
- ***RDD is a read-only (i.e. immutable) distributed collection of objects/elements***.
- RDD can be self recovered in case of failure (support rebuild if a partition is destroyed).
- There are two types of RDD Operations: Transformation and Action
  - Transformations
    - Transformations are operations on RDDs that return a new RDD.
    - Transformed RDDs are computed lazily (only when you use them in an action)
    - Many transformations are *element-wise* (working on one element at a time)
  - Actions
    - trigger job execution that forces the evaluation of all the transformations and must return a final value
    - the values of action are stored to drivers or to the external storage system.
    - It brings laziness of RDD into motion.

Lecture 8: Introduction to Spark Framework

RDD and its operations (***ID43, Lec8, 23-28***)

- Typical Transformations:
  - map(func), filter(func), flatMap(func), mapValues(func), union(func), reduceByKey(func)
- Typical Actions:
  - Collect(), count(), reduce(func), take(n), first()

Lecture 8: Introduction to Spark Framework

Lazy Evaluation (***ID44, Lec8, 30***)

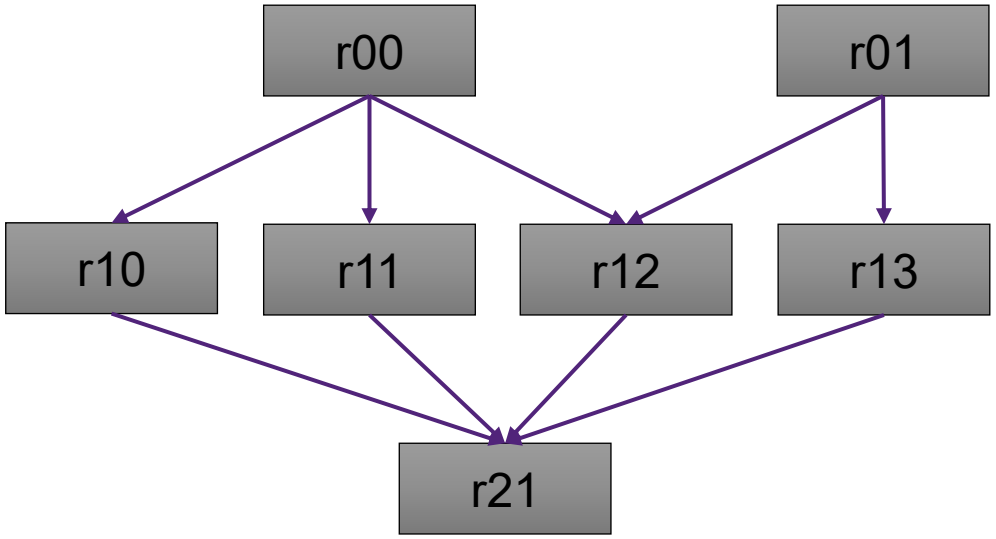
- Transformations on RDDs are evaluated or computed in a lazy manner:
  - Spark will not begin to execute until it sees an action.
- Lazy evaluation means that when a transformation on an RDD is called, the operation is not immediately performed.
- Spark internally records metadata to indicate that this operation has been requested.
- Spark can decide what the best way is to perform a series of transformations that are recorded.
- Spark uses lazy evaluation to reduce the number of passes (storage on disk)

Lecture 8: Introduction to Spark Framework

RDD Lineage Graph (***ID45, Lec8, 31***)

- Because of lazy nature of RDD, dependencies between RDDs are logged in a lineage graph (or RDD operator graph or RDD dependency graph).
- Lineage graph can be regarded as a logical execution plan of RDD transformations.
- When you run into action, this local plan is submitted to an optimiser, which is going to do optimization and implement it into a physical plan containing stages.
- Spark logs all transformations with a graph structure which can be optimized by graph optimization technology.
- **Lineage graph can be used to re-build the RDDs**

Fault Tolerance



Lecture 8: Introduction to Spark Framework

RDD Persistence and Caching (***ID46, Lec8, 33-34***)

- To reduce computation overhead, RDD persistence is to save the intermediate result.
- Methods:
  - cache(): store all the RDD in-memory (default storage level: MEMORY_ONLY).
  - persist(level): can cache in memory, on disk, or off-heap memory.
- Use cases:
  - iterative machine learning applications
  - standalone Spark applications (multiple actions will perform on the same RDD)
  - When too many transformations on RDD or some computations are very expensive

Lecture 8: Introduction to Spark Framework

Terms in Spark (***ID47, Lec8, 36-41***)

- **Job**: A piece of code that reads some input from HDFS or local, performs some computation on the data and writes some output data.
- **Stages**: Jobs are divided into stages.
- **Tasks**: Each stage has some tasks, one task per partition. One task is executed on one partition of data on one executor.
- **Spark Driver** (program driver): A separate process to execute user applications
- **Executors**: Run tasks scheduled by the driver, store computation results in memory, on disk or off-heap memory, interact with storage systems
- **Master**: The machine on which the Driver program runs
- **Slave**: The machine on which the Executor program runs

Lecture 8: Introduction to Spark Framework

Terms in Spark (***ID47, Lec8, 36-41***)

- **SparkContext:** the main entry point to spark functionality
- **Cluster Manager**:
  - is to divide resources across applications and works as an external service for acquiring resources on the cluster.
  - Spark supports pluggable cluster management, which handles starting executor processes, including:
    - Standalone Cluster Manager
    - Hadoop YARN
    - Apache Mesos

Lecture 8: Introduction to Spark Framework

Directed Acyclic Graph (***ID48, Lec8, 43-44***)

- Directed Acyclic Graph (DAG) is a set of vertices and edges
  - vertices represent the RDDs;
  - edges represent the operation to be applied on RDD.
- DAG is a finite directed graph (finite vertices and edges) with no directed cycles.
- It contains a sequence of vertices such that every edge is directed from earlier to later in the sequence.
- DAG operations can do better global optimization than other systems like MapReduce.
- On the calling of Action, the created DAG submits to DAG Scheduler which further splits the graph into the stages of the job.
- The DAGScheduler splits the Spark RDD into stages based on applied transformation.
- Fault Tolerance in RDD is achieved using DAG.

Lineage Graph v.s. Directed Acyclic Graph (DAG)

Directed Acyclic Graph (***ID48, Lec8, 43-44***)

Lineage Graph:

- Captures the sequence of transformations applied to the data.
- A lineage graph is a record of how a particular piece of data was derived. For each RDD (Resilient Distributed Dataset), there's a lineage graph that traces back the sequence of transformations (like `map`, `filter`, etc.) that produced it from the source data.
- The lineage graph aids in recovery during data loss. If a partition of an RDD is lost, Spark can recompute it from the source data using the lineage graph, thus ensuring fault tolerance without data replication.
- A lineage graph represents individual transformations and their dependencies.

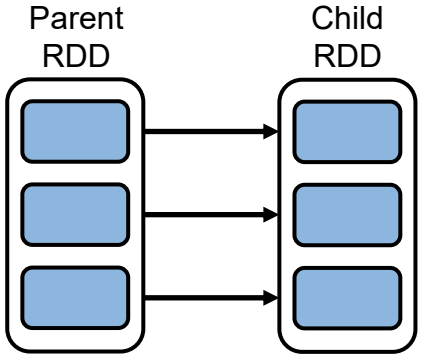
Directed Acyclic Graph (DAG):

- Models the entire execution plan for a Spark job.
- When an action (like `count`, `collect`, etc.) is called in Spark, it constructs a DAG to model the stages and tasks required for that action. This DAG represents the optimized execution plan for the computation.
- The DAG scheduler divides the DAG into stages. Each stage contains as many transformations as possible that have narrow dependencies. Wide dependencies introduce stage boundaries. This means that transformations which can be done in parallel without shuffling data are grouped into a single stage.
- The DAG represents the entire job, and Spark executes the job by executing these stages in topological order. If one stage fails, only that stage is recomputed, leveraging the DAG's ability to identify the minimum set of tasks to recompute.

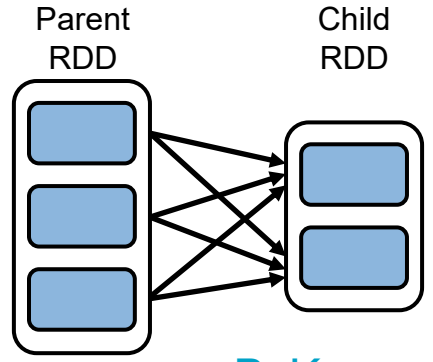
Lecture 8: Introduction to Spark Framework

Narrow and Wide Dependencies (***ID49, Lec8, 46-47***)

- **Narrow transformation (dependencies)**
  - each partition of the parent RDD is used by at most one partition of the child RDD
  - allow for pipelined execution on one cluster node
  - failure recovery is more efficient as only lost parent partitions need to be recomputed
  - *Example: map, flatmap, filter, sample, union, etc.*
- **Wide transformation (dependencies)**
  - multiple child partitions may depend on one parent partition
  - require data from all parent partitions to be available and to be shuffled across the nodes
  - a complete re-computation is needed, if some partition is lost from all the ancestors
  - *Example: groupbyKey()* and *reducebyKey()*.



map, filter



groupByKey

Lecture 8: Introduction to Spark Framework

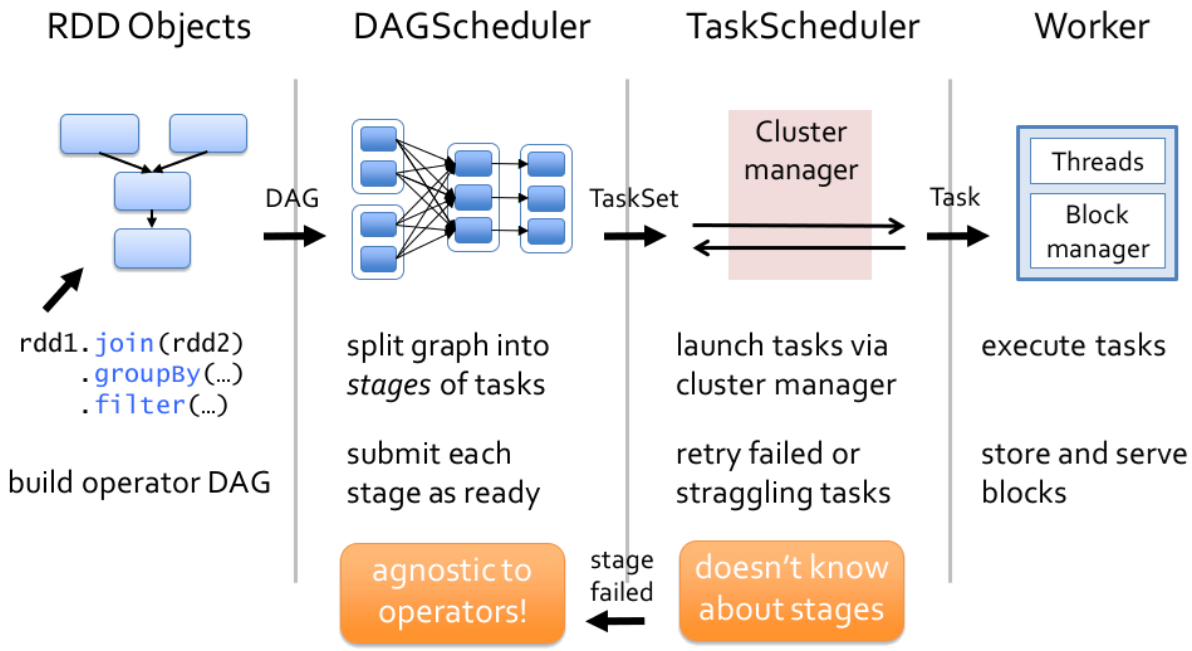
Shuffle (***ID50, Lec8, 49-51***)

- In Spark, we can group the records with the same key to one partition for a further operation (e.g. reduceByKey, groupByKey).
- To do this computation, we need to re-distribute (shuffle) data so that it's grouped differently across partitions.
- In Spark, the huge database is distributedly stored in RDDs across partitions in different nodes, some transformation operations will trigger shuffle (re-distributing) data.
- Shuffle typically involves copying data across executors and machines, making the shuffle a complex and costly operation.
- Shuffle is expensive since it involves disk I/O, data serialization, and network I/O.
- Shuffle also generates a large number of intermediate files on disk.
- Transformations that ***cause communications across nodes when repartitioning*** will cause a shuffle in Spark: join, groupByKey, reduceByKey, combineByKey, etc.

Lecture 8: Introduction to Spark Framework

Spark workflow (***ID51, Lec8, 57***)

1. Create an RDD object;
2. SparkContext is responsible for calculating the dependencies between RDDs and building DAGs;
3. DAG Scheduler is responsible for decomposing the DAG graph into multiple stages, each stage containing multiple tasks,
4. Task scheduler launches tasks to distribute across the worker nodes via cluster manager (Standalone or Mesos or YARN). The task scheduler does not know about dependencies among stages.



Lecture 9: RDD Programming

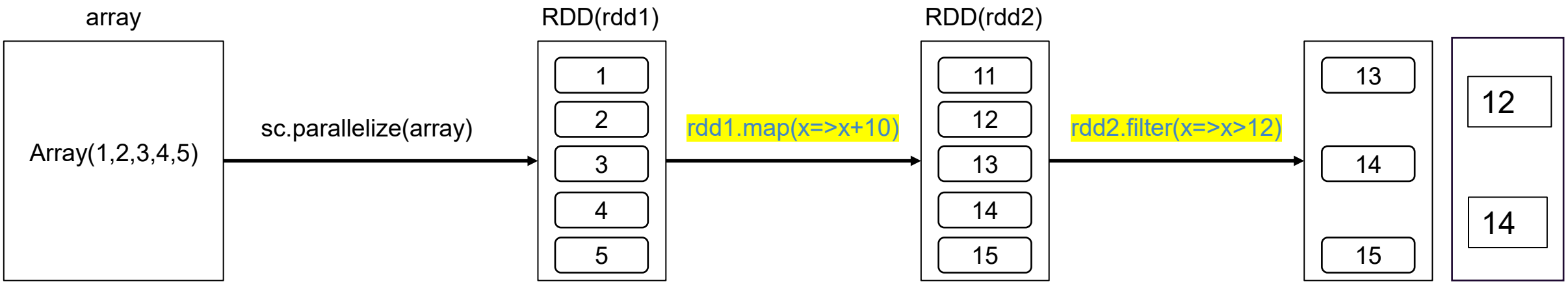
RDD Programming Fundamentals (***ID52, Lec9, 6-44***)

- RDD Creation: textFile & parallelize methods
- **RDD Transformation: Map, filter, flatMap, distinct, union, intersection, subtract, and cartesian**
- **RDD Action: Collect, count, reduce, first, take, foreach**
- RDD Persistence and Caching
- Key/Value Pairs: Creating KV Pairs
- Transformations and actions on Key/Value Pairs
  - reduceByKey, groupByKey, keys, values, sortByKey, mapValues, and join
- Partition for Paired RDDs

Lecture 9: RDD Programming

RDD Programming Fundamentals (***ID52, Lec9, 6-44***)

- RDD Creation: textFile & parallelize methods
- RDD Transformation: Map, filter, flatMap, distinct, union, intersection, subtract, and cartesian



(x) => (x+10) is equivalent to

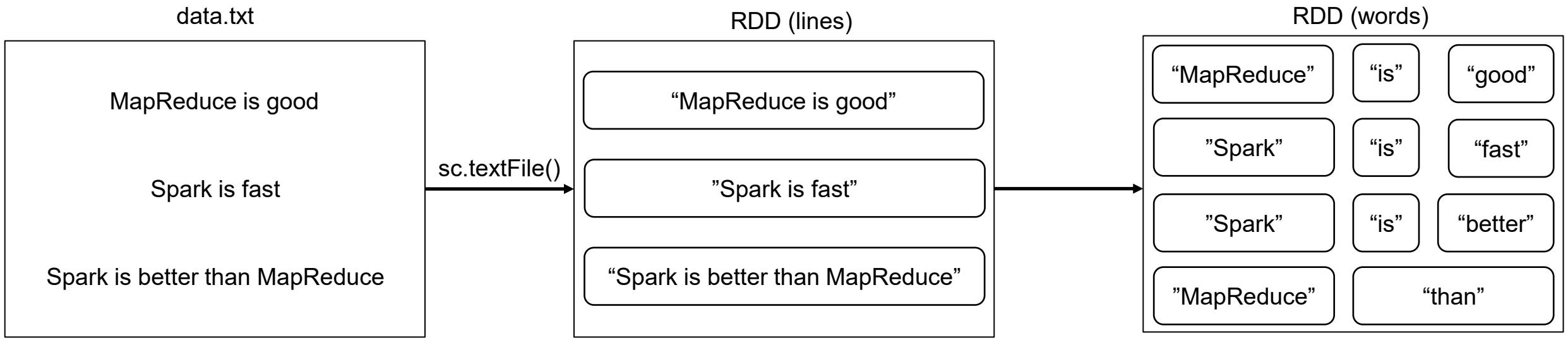
func(x) { return x+10 }

x=>x%2==0

Lecture 9: RDD Programming

RDD Programming Fundamentals (***ID52, Lec9, 6-44***)

- flatMap(func): Similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item).
- A simple usage of flatMap() is splitting up an input string into words,

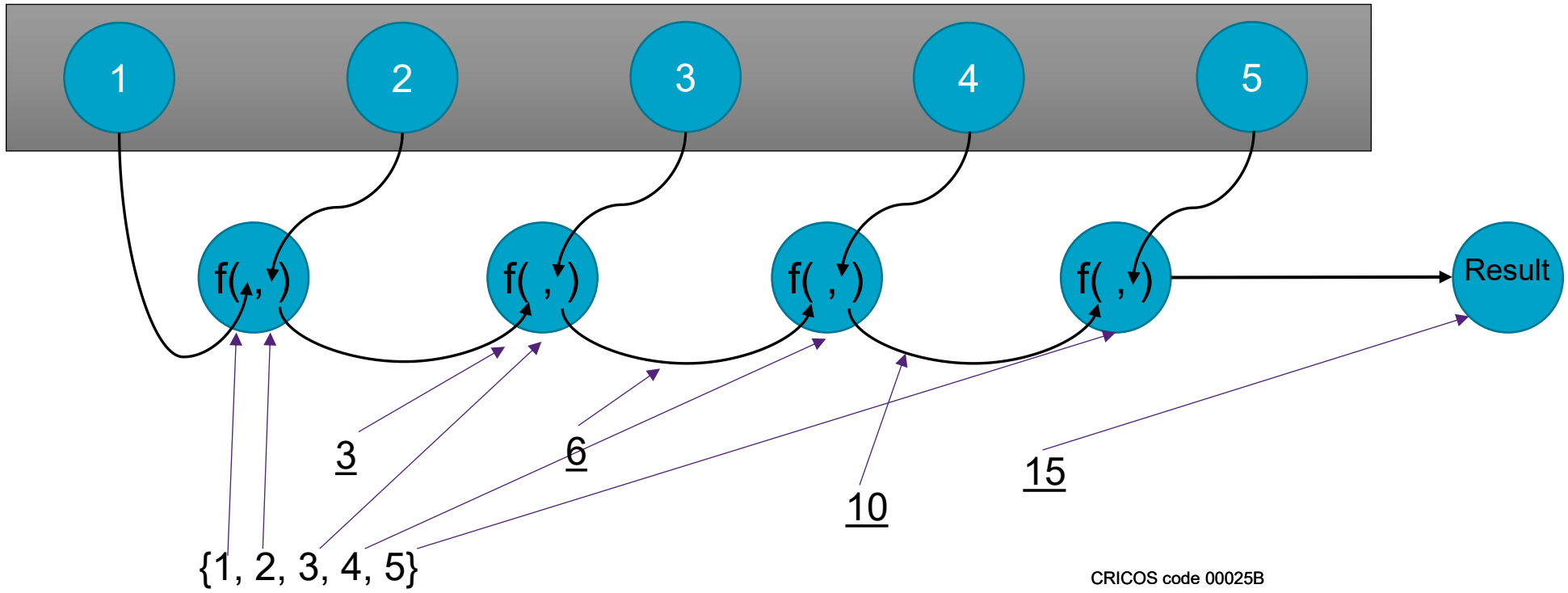


lines.flatMap(line=>line.split(“ “))

Lecture 9: RDD Programming

RDD Programming Fundamentals (***ID52, Lec9, 6-44***)

- Given an RDD rdd = {1, 2, 3, 4, 5}
- reduce(func): `val sum = rdd.reduce((x, y) => x + y)`



Lecture 9: RDD Programming

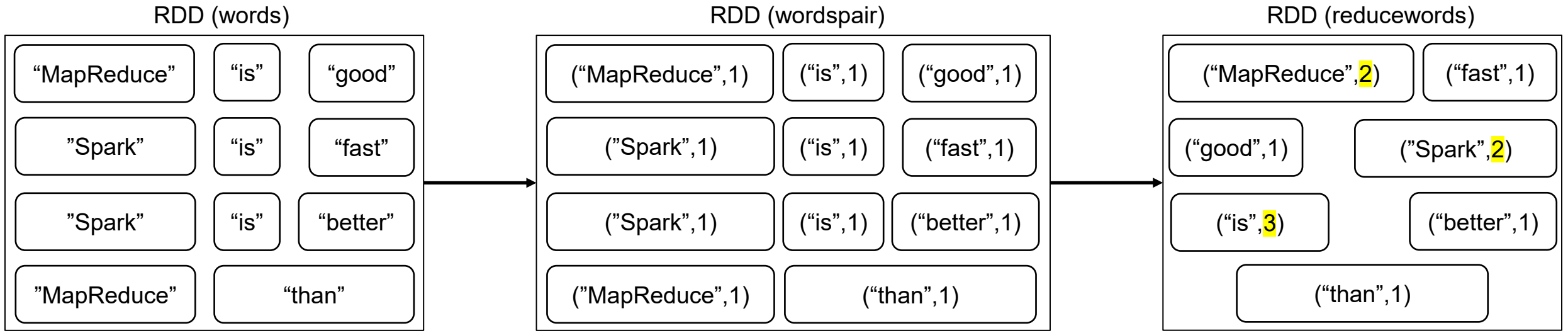
RDD Programming Fundamentals (***ID52, Lec9, 6-44***)

- RDDs of key/value pairs are a common data type required for many operations in Spark.
  - (“Spark”, 2), (“is”, 3), (“is”, (1,1,1)), etc.
- Key/value RDDs are commonly used to perform aggregations.
  - counting up reviews for each product,
  - grouping together data with the same key,
  - grouping together two different RDDs.
- Spark provides special operations on RDDs containing key/value pairs.
  - reduceByKey() method can aggregate data separately for each key
  - join() method can merge two RDDs together by grouping elements with the same key
- ._1 & ._2 stand for key and value, respectively.
  - E.g. ._1 stands for “Spark” or “is”, while ._2 stands for 2 or 3.

Lecture 9: RDD Programming

RDD Programming Fundamentals (***ID52, Lec9, 6-44***)

- reduceByKey(**func**): called on a dataset of (K, V) pairs,
- func must be of type (V,V) => V, e.g. (a,b) => a+b.

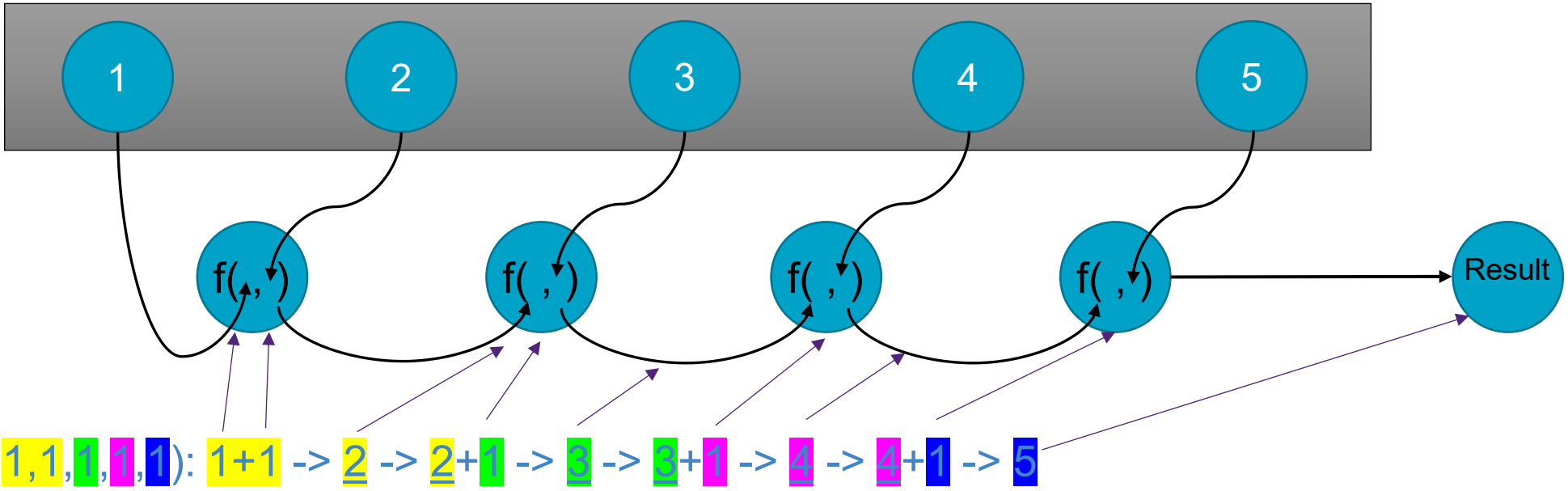


Lecture 9: RDD Programming

RDD Programming Fundamentals (***ID52, Lec9, 6-44***)

f(a, b) { return a + b }

- reduceByKey(**func**):
  - **wordspair.reduceByKey((a,b)=>a+b) on** {(”is”, 1), (”is”, 1), (”is”, 1), (”is”, 1), (”is”, 1)}



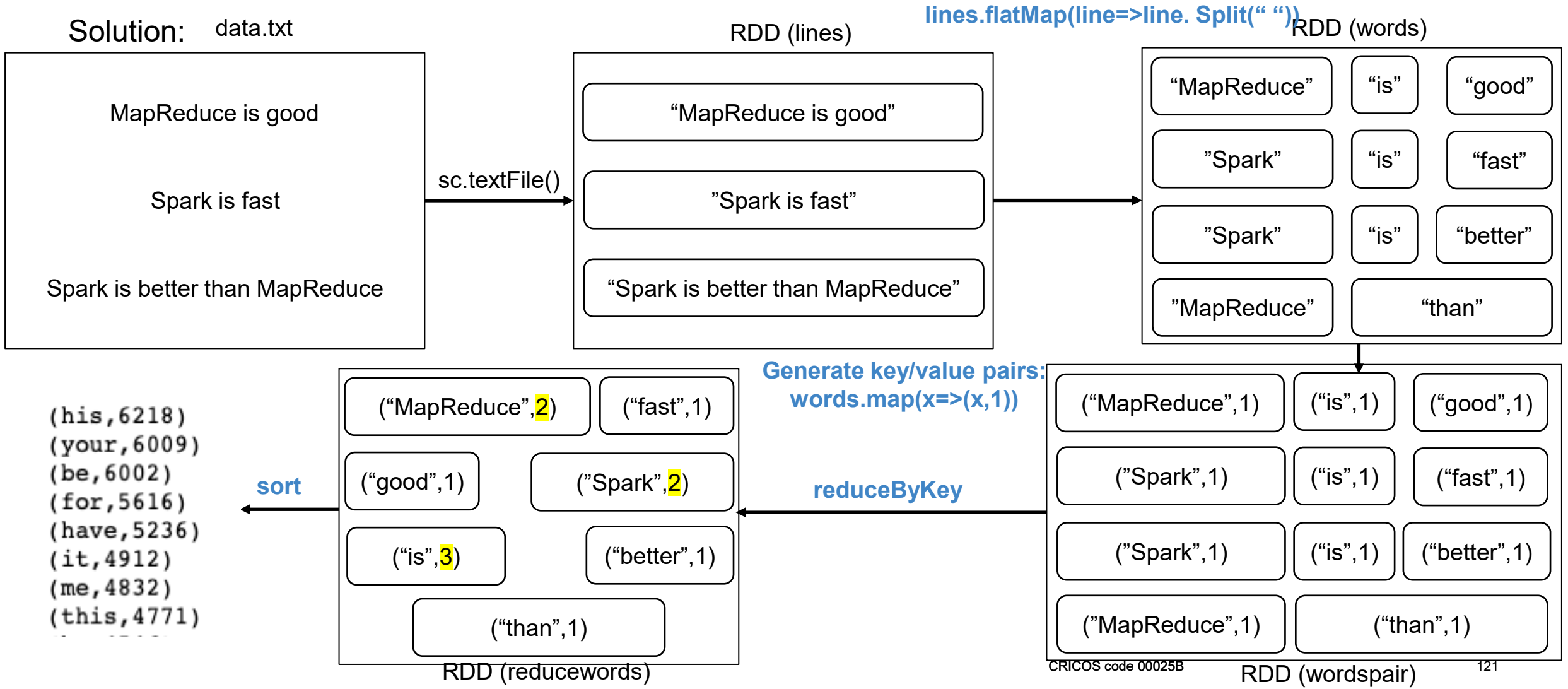
  - (1,1,1,1,1): 1+1 -> 2 -> 2+1 -> 3 -> 3+1 -> 4 -> 4+1 -> 5

Lecture 9: RDD Programming

RDD Programming Examples (***ID53, Lec9, 46-56***)

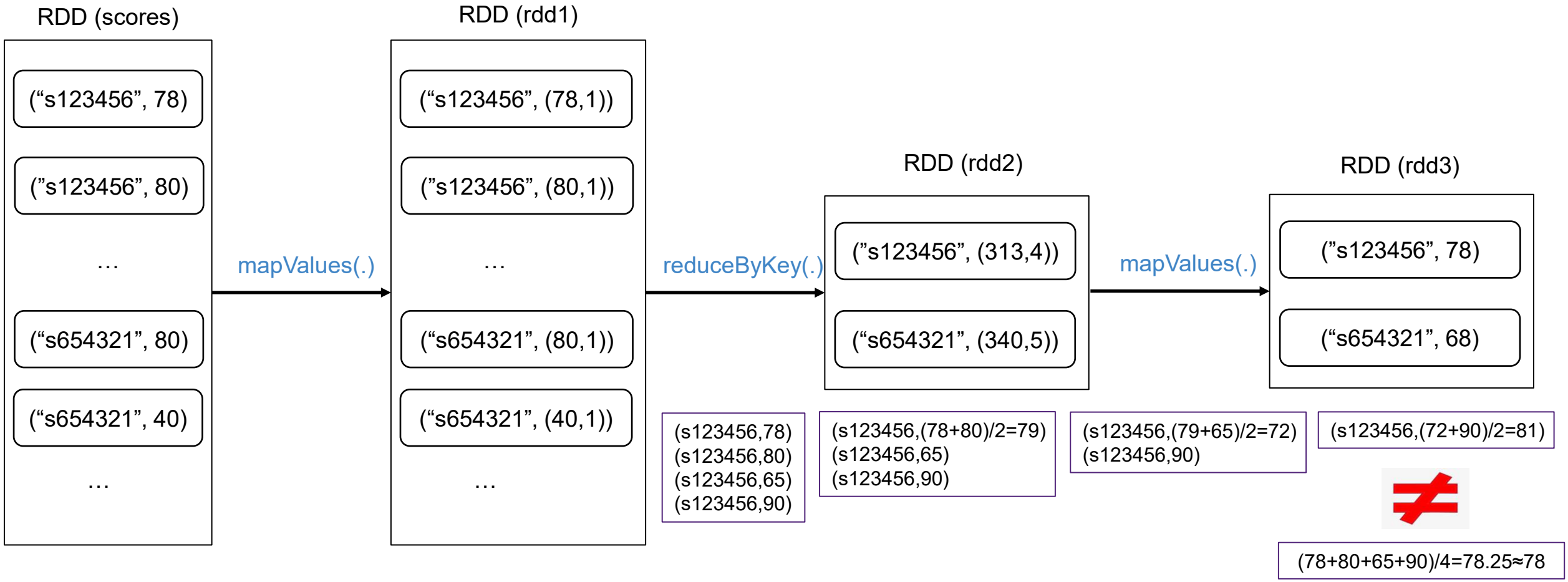
- I. Word count
- II. Average marks calculation
- III. Get top-5 values
- IV. File sorting
- V. Movie Rating

RDD Programming Example I: Count words in Shakespeare



RDD Programming Example II: Calculate averaged marks

Solution:

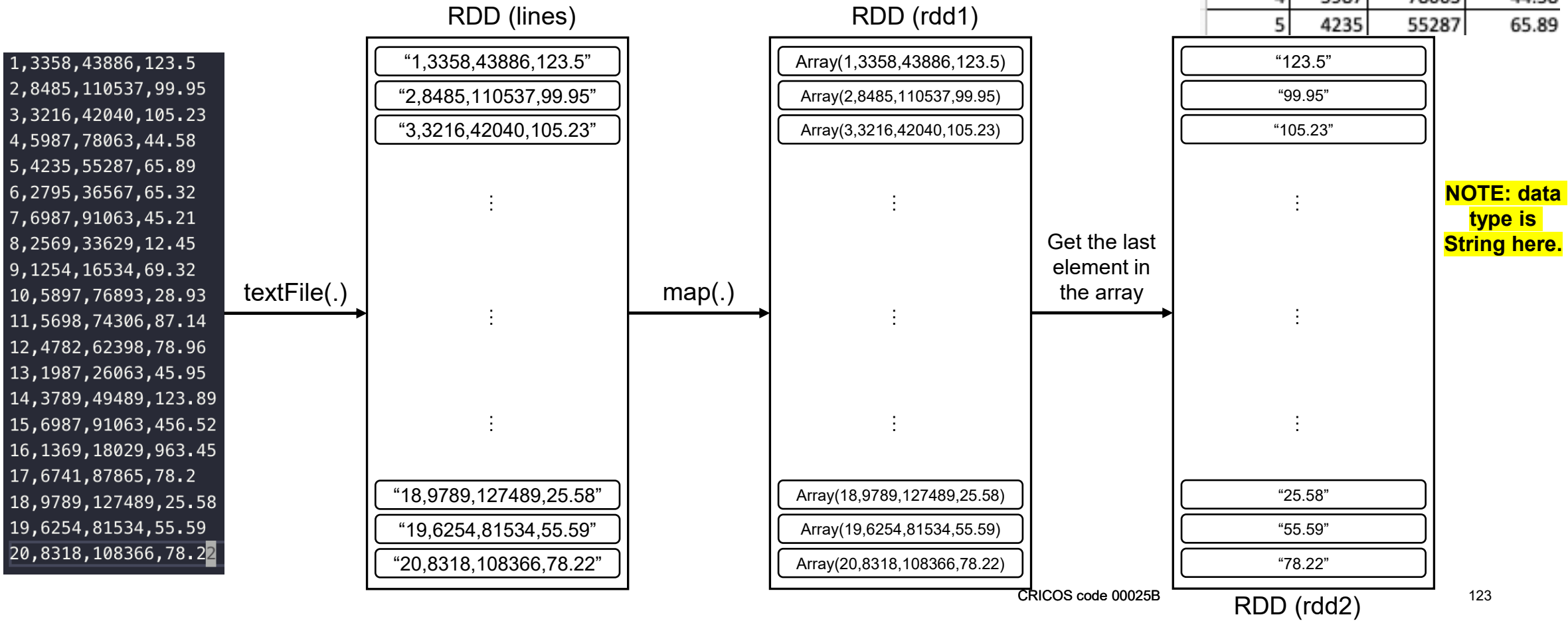


RDD Programming Example III: Get Top-5 Values

Get the Top value of RDD

Given a dataset of sale records: sales(orderId, userId, productid, payment)

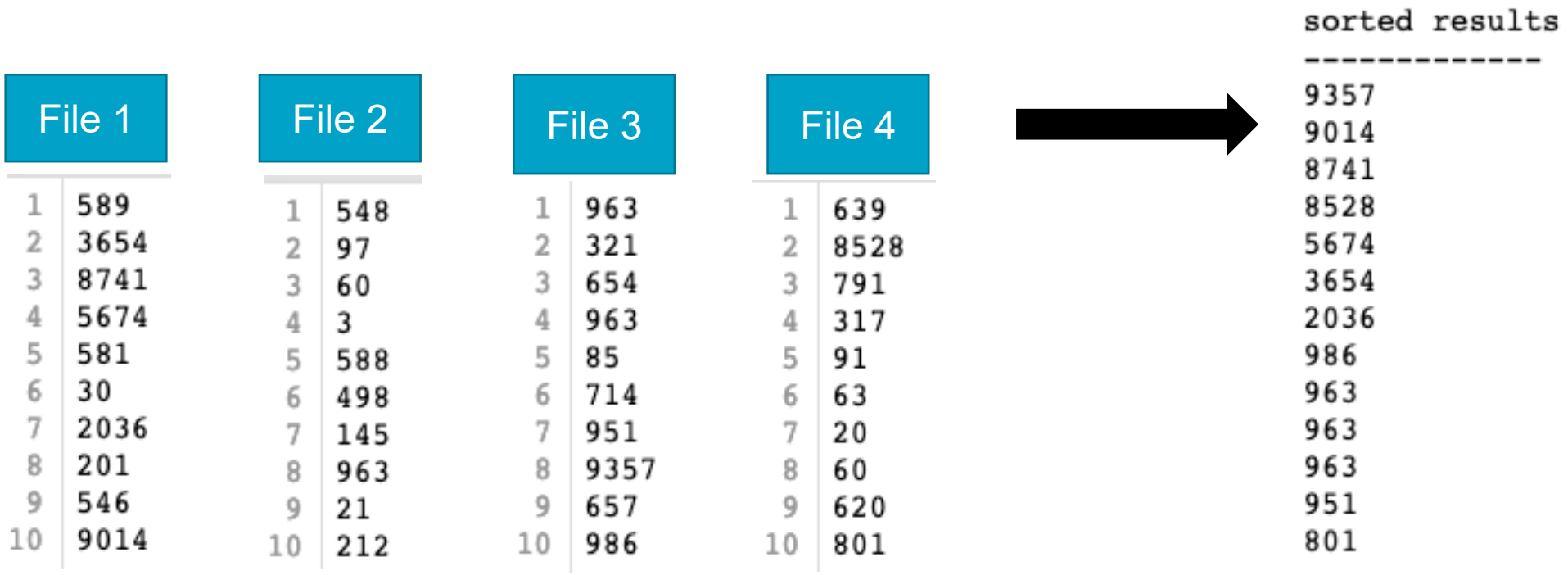
orderId	userId	productId	payment
1	3358	43886	123.5
2	8485	110537	99.95
3	3216	42040	105.23
4	5987	78063	44.58
5	4235	55287	65.89



RDD Programming Example: Get Top-5 Values

RDD Programming Example IV: File Sorting

Given multiple files (local file system or distributed file system), sort contents in the files.



RDD Programming Example V: Movie Rating

MovieLens

- GroupLens Research has collected and made available rating data sets: MovieLens 20M
  - 20 million ratings and 465,000 tag applications applied to 27,000 movies by 138,000 users.
  - Includes tag genome data with 12 million relevance scores across 1,100 tags.
  - Released 4/2015; updated 10/2016 to update links.csv and add tag genome data.

Goal:

- Given user ratings (more than 20,000,000 ratings by 138,000 users), calculate the average rating score for each movie (27,000).
- Display the titles and averaged rating scores for top 100 movies

movies(movieId, title, genres)

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama|Romance
5	Father of the Bride Part II (1995)	Comedy
6	Heat (1995)	Action|Crime|Thriller
7	Sabrina (1995)	Comedy|Romance
8	Tom and Huck (1995)	Adventure|Children
9	Sudden Death (1995)	Action
10	GoldenEye (1995)	Action|Adventure|Thriller

ratings(userId, movieId, rating, timestamp)

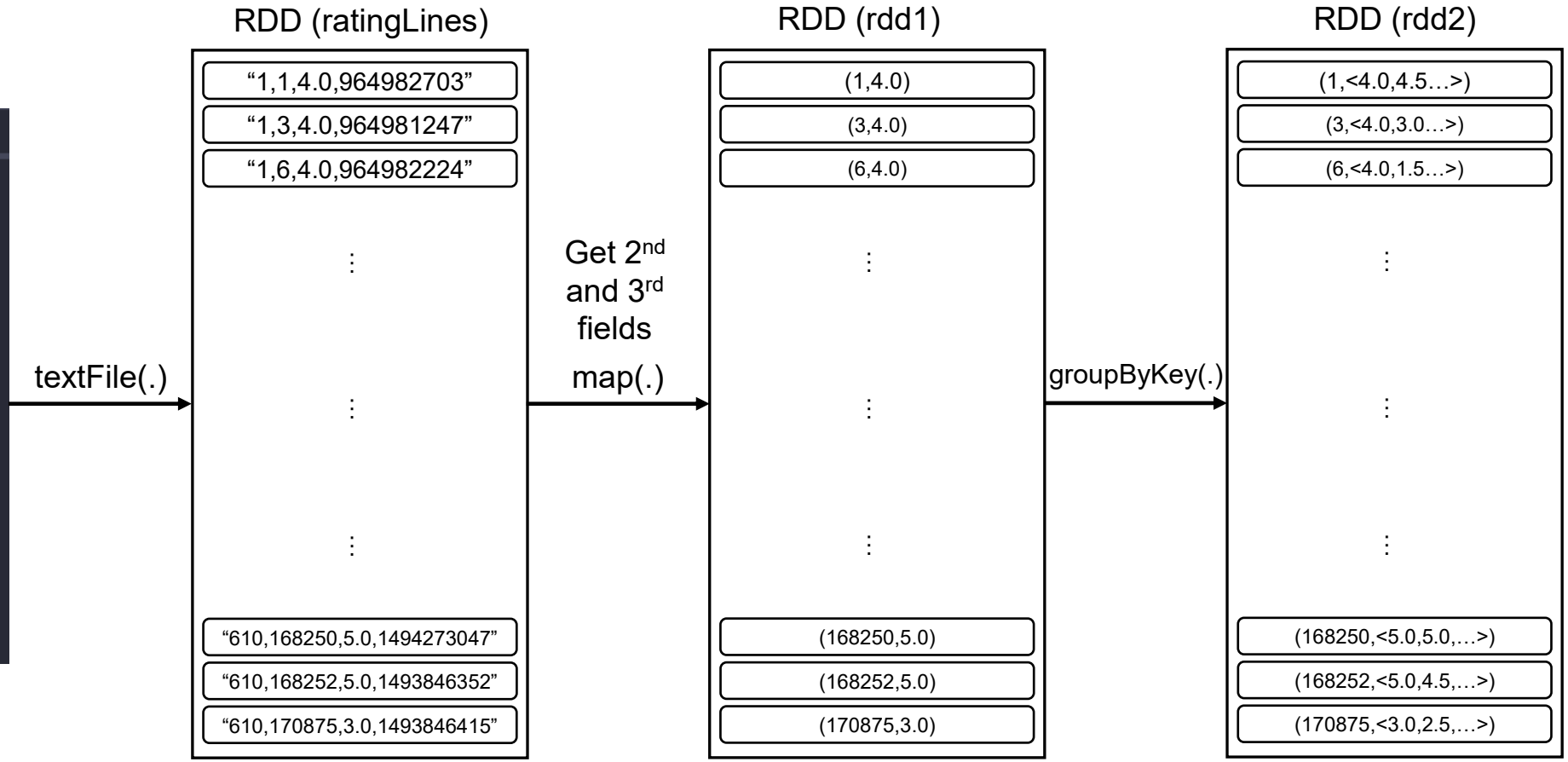
	userId	movieId	rating	timestamp
2	1	1	4	964982703
3	1	3	4	964981247
4	1	6	4	964982224
5	1	47	5	964983815
6	1	50	5	964982931
7	1	70	3	964982400
8	1	101	5	964980868
9	1	110	4	964982176
10	1	151	5	964984041

https://grouplens.org/datasets/movielens/

RDD Programming Example: Movie Rating

Solution:

```
1    1,1,4.0,964982703
2    1,3,4.0,964981247
3    1,6,4.0,964982224
4    1,47,5.0,964983815
5    1,50,5.0,964982931
6    1,70,3.0,964982400
7    1,101,5.0,964980868
8    1,110,4.0,964982176
9    1,151,5.0,964984041
10   1,157,5.0,964984100
```

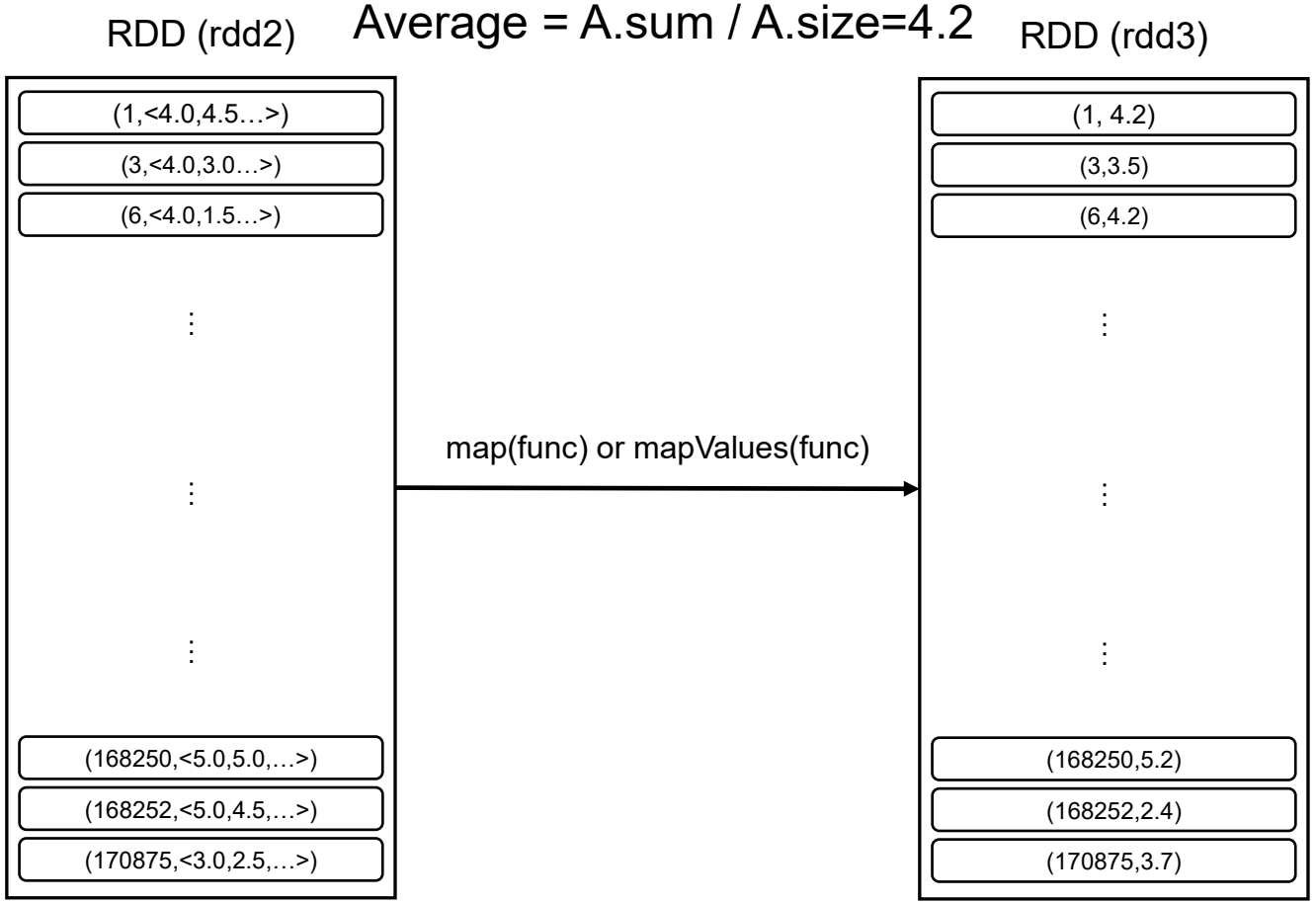


https://grouplens.org/datasets/movielens/

RDD Programming Example: Movie Rating

Solution:

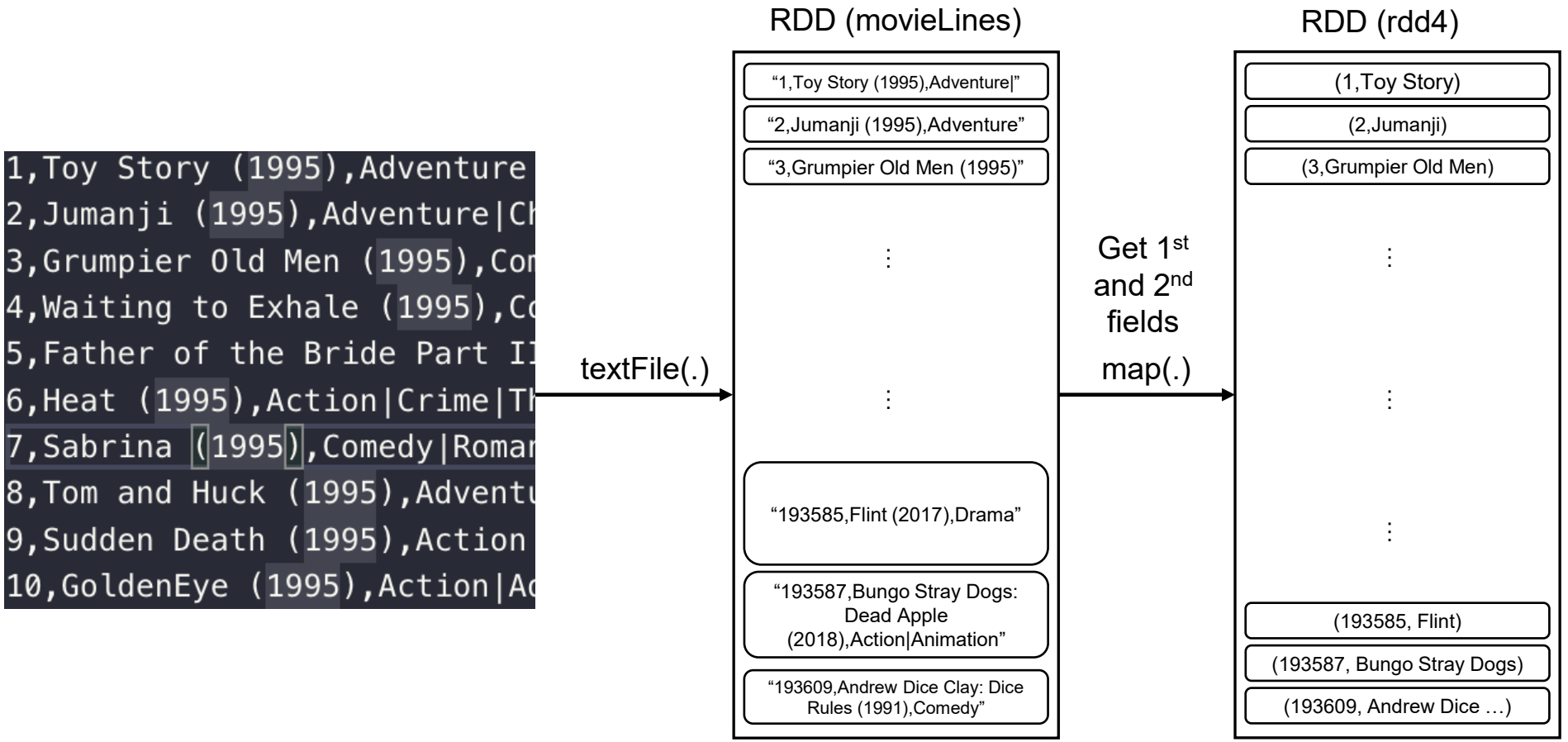
A = <4.0,4.5,4.0,4.0,4.5>
Average = A.sum / A.size=4.2



https://grouplens.org/datasets/movielens/

RDD Programming Example: Movie Rating

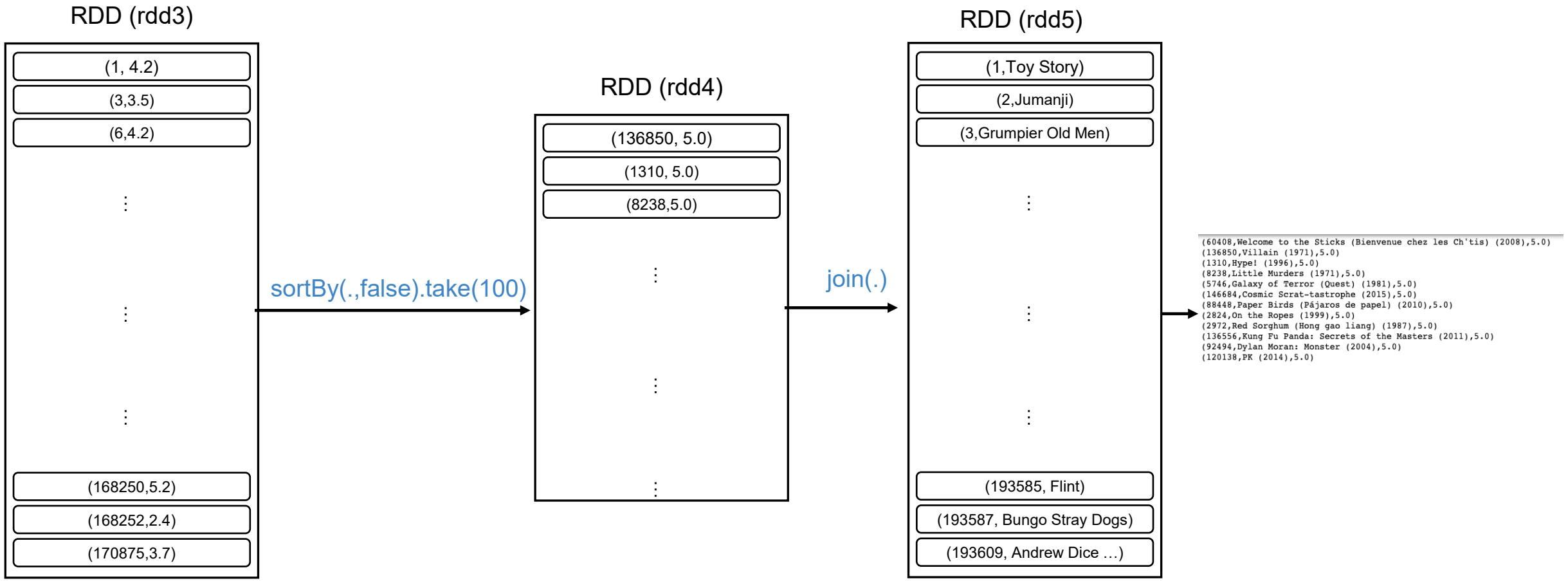
Solution:



https://grouplens.org/datasets/movielens/

RDD Programming Example: Movie Rating

Solution:



https://grouplens.org/datasets/movielens/

Lecture 9: RDD Programming

RDD vs DataFrame (***ID54, Tutorial 9 in Week 10***)

- DataFrame is a distributed collection of data, which is organized into named columns (inspired from DataFrame in R Programming and Pandas in Python).
- DataFrame still has immutability, resilient, in-memory, distributed computing abilities.
- RDD vs DataFrame:
  - RDD is an immutable distributed collection of elements of your data, partitioned across nodes in your cluster that can be operated in parallel with a low-level API that offers transformations and actions.
  - DataFrame is an immutable distributed collection of data. But it is organized into named columns, like a table in a relational database.
  - DataFrame is designed to make large data sets processing even easier in some tasks (a better version of RDD).
  - No-structured data manipulation (RDD) vs large structured data query (DataFrame)

What's Next for Spark Programming?

In Apache Spark, both **RDD** and **DataFrame** are fundamental data structures, but …

- RDD (Build foundation for Spark Programming):
  - It's a **low-level** data structure in Spark, providing an object-oriented interface.
  - Requires more extensive coding for operations and transformations
  - Gives you a fine-grained control over data and its distribution across nodes.
  - RDDs are type-safe: the compiler will validate types while compiling (early error detection).
  - When you need fine-grained control over your dataset and its partitioning, or when you're dealing with data that doesn't fit into a structured model.
- DataFrame (**Will be introduced with Spark SQL&ML in Tutorial**):
  - It's a **higher-level** abstraction representing a distributed collection of data organized into **named columns**. It's conceptually **equivalent to a table** in a relational database.
  - Allows users to perform operations using SQL-like syntax, making it more **user-friendly**.
  - Can be easily integrated with various data sources (like Hive, ORC, JSON, JDBC, and others).
  - Not type-safe (returned as Array of rows) --> type-related errors can only be caught at runtime.
  - When you are working with structured data, or when you need to leverage Spark's in-built optimizations and want to perform SQL-like operations.

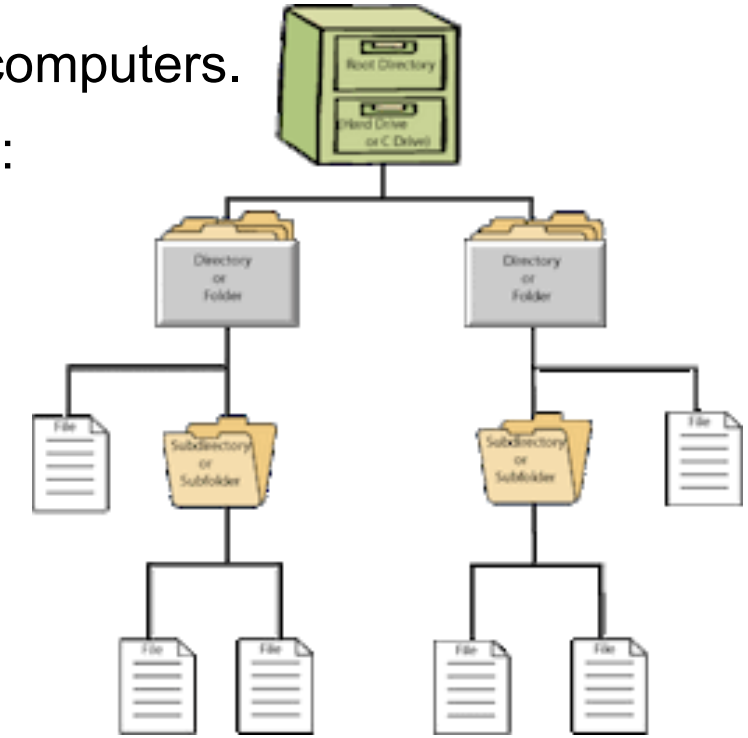
Lecture 10: DFS

ID	Topics	Pages	Lecture
55	DFS	15-16	10
56	SUN's NFS	17-20	10
57	CFS & Differences among DFS,NFS, CFS	23-27	10
58	GFS	29-31	10
59	Read and Write in GFS	32-34	10
60	HDFS	39-51	10
61	HDFS vs GFS	52	10

Lecture 10: Distributed File System

DFS (***ID55, Lec10, 15-16***)

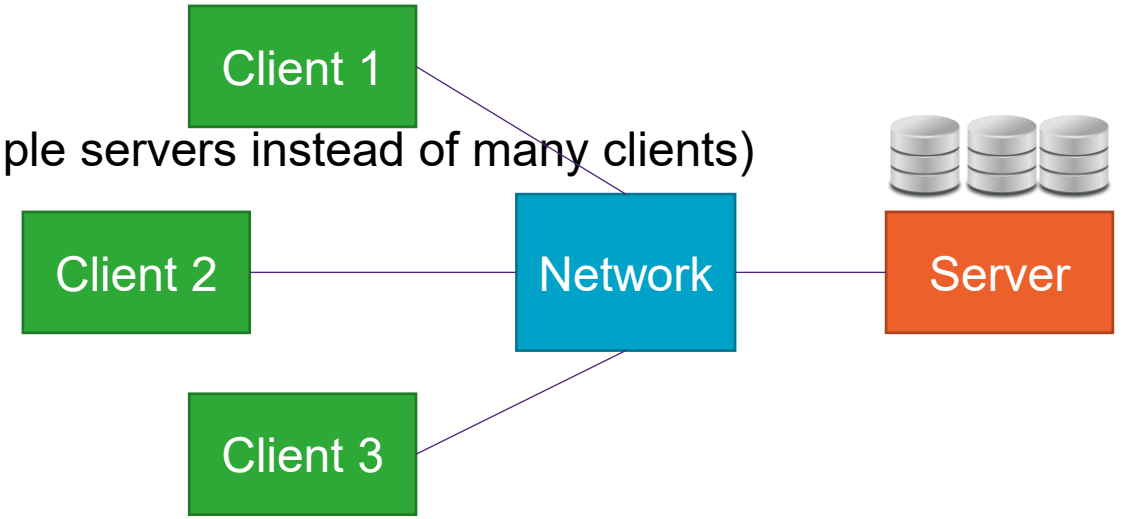
- DFS is a distributed implementation of the classical time-sharing model of a file system, where multiple users share files and storage resources.
- A distributed file system spreads over multiple, autonomous computers.
- A distributed file system should have following characteristics:
  - Access transparency
  - Location transparency
  - Concurrency transparency
  - Failure transparency
  - Replication transparency
  - Migration transparency
  - Heterogeneity
  - Scalability



Lecture 10: Distributed File System

SUN's NFS (***ID56, Lec10, 17-20***)

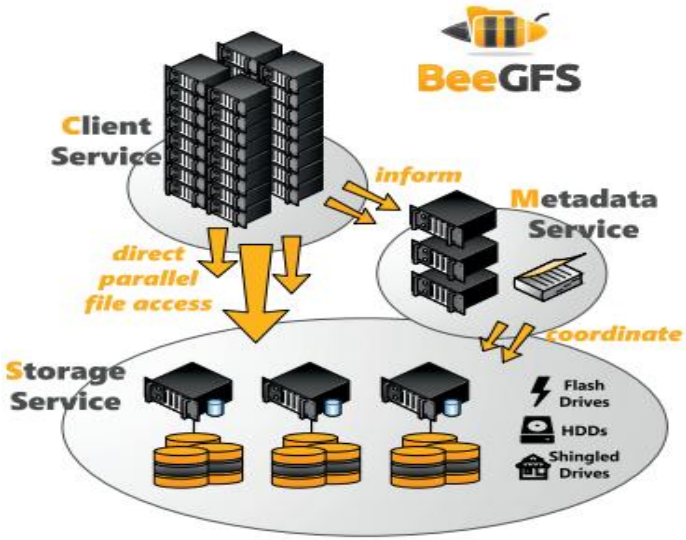
- Network File System (NFS) is a distributed file system protocol originally developed by Sun Microsystems in 1984.
- NFS allows a user on a client computer to access files over a computer network much like local storage is accessed.
- NFS is a client-server application, where a user can view, store and update the files on a remote computer.
- Advantages:
  - easy sharing of data across clients
  - centralized administration (backup done on multiple servers instead of many clients)
  - security (put server behind firewall)



Lecture 10: Distributed File System

CFS (***ID57, Lec10, 23-27***)

- Clustered File System (CFS) is not a single server with a set of clients, but instead a cluster of servers that all work together to provide high performance service to their clients.
- CFS is a distributed file system that consists of several **simultaneously mounted servers** that share the responsibilities of the system, as opposed to a single server (possibly replicated).
- The design decisions for a CFS are mostly related to how the data is distributed **across the cluster** and how it is managed.



Lecture 10: Distributed File System

GFS (***ID58, Lec5, 29-31***)

- **Google File System** is a scalable distributed file system developed by Google to provide efficient, reliable access to data using large clusters of commodity hardware.
- Shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability.
- GFS/HDFS is a typical example of CFS.
- Design Motivations:
  - Thousands of commodity computers (cheap hardware but distributed)
  - GFS has high component failure rates
  - Modest number of huge files (million files, 100MB or larger) & No need to optimize for small files
  - Workloads : two kinds of reads and writes
    - Large streaming reads (1MB or more) and small random reads (a few KBs)
    - Sequential appends to files by hundreds of data producers
  - High sustained throughput is more important than latency

Lecture 10: Distributed File System

Read and Write in GFS (***ID59, Lec10, 32-34***)

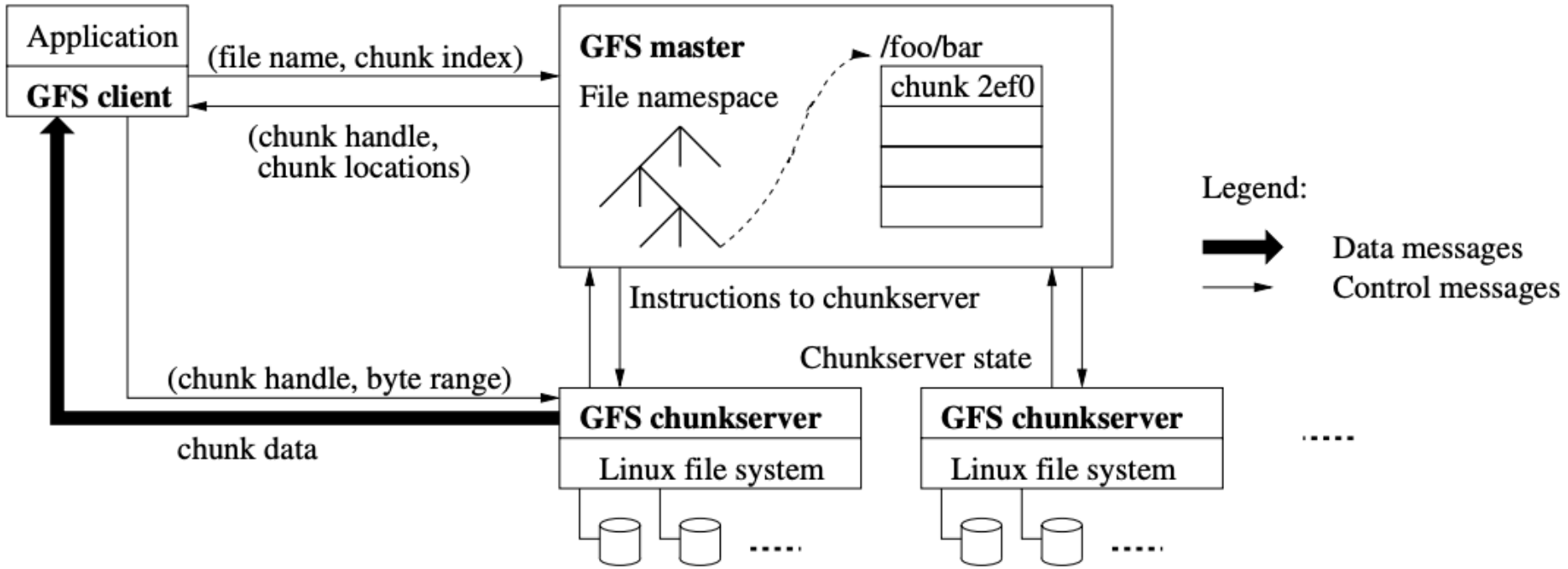


Figure 1: GFS Architecture

GFS Architecture – Read

The reading process in Google's File System (GFS) follows a series of specific key steps below:

1. **Client Request**: The client issues a request to read a file from GFS. The request specifies the file name and the byte range to be read.
2. **Communicating with the Master Server**:

- The client contacts the GFS Master server to obtain metadata about the file. Specifically, the client requests the file's chunk locations, including which chunk servers hold the replicas of the required file chunks.
- The Master responds by providing the client with the chunk metadata (i.e., the chunk handle and the locations of replicas on chunk servers).

3. **Contacting the Chunk Servers**:

- Once the client knows the chunk locations, it directly contacts the closest chunk server that holds the relevant replica of the chunk it needs to read.
- If multiple replicas exist, the client can select the closest chunk server to minimise network latency.

4. **Reading the Data**:

- The client sends a read request to the chosen chunk server, specifying the chunk handle and the byte range of the chunk it needs.
- The chunk server responds by reading the requested chunk data and sending it back to the client.

Handling Larger Files: If the requested data spans multiple chunks, the client will repeat the process for subsequent chunks by contacting other chunk servers until it has read the entire requested byte range.
Caching Metadata: The client caches the metadata (chunk locations) provided by the Master to reduce communication overhead for future reads, avoiding repetitive requests to the Master server.
Data Integrity Check: GFS includes mechanisms to ensure data integrity. Each chunk server stores a checksum for every chunk. When a chunk server sends data to the client, it verifies the checksum to ensure that the data has not been corrupted during storage or transmission.
This process ensures that file reads are optimized for performance, with **minimal involvement of the Master server** after the initial metadata lookup.

Lecture 10: Distributed File System

Read and Write in GFS (***ID59, Lec10, 32-34***)

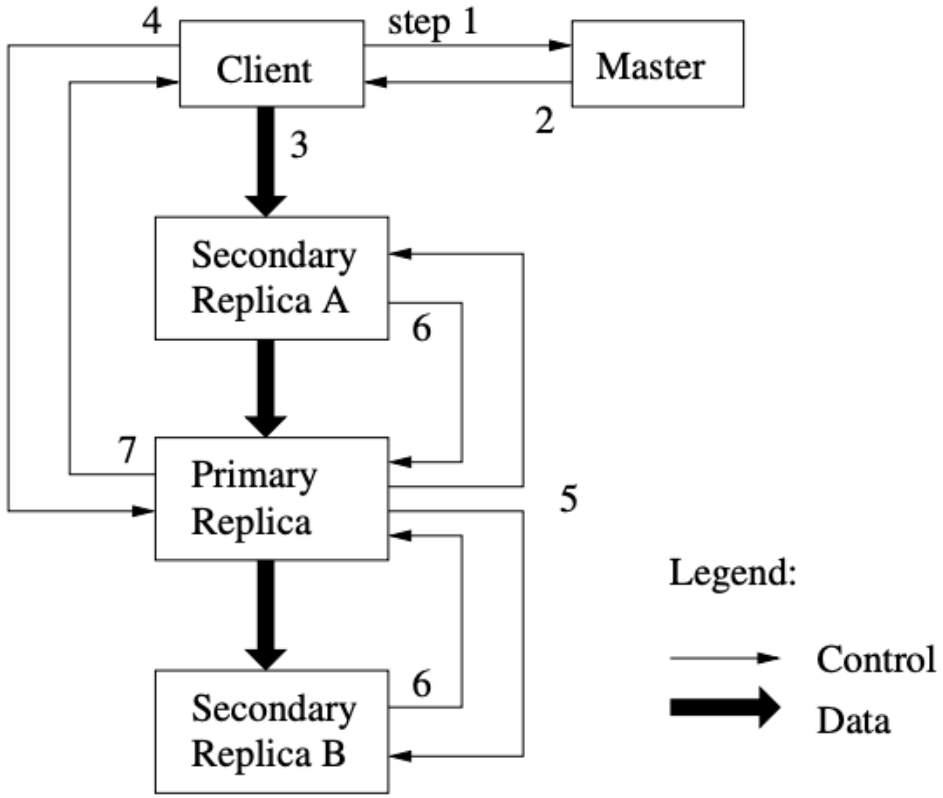


Figure 2: Write Control and Data Flow

1.Client asks master which chunk server holds current lease of chunk and locations of other replicas.

2.Master replies with identity of primary and locations of secondary replicas.

3.Client pushes data to all replicas

4.Once all replicas have acknowledged receiving the data, client sends write request to primary. The primary assigns consecutive serial numbers to all the mutations it receives, providing serialization. It applies mutations in serial number order.

5.Primary forwards write request to all secondary replicas. They apply mutations in the same serial number order.

6.Secondary recplicas reply to primary indicating they have completed operation

7.Primary replies to the client with success or error message

Lecture 10: Distributed File System

HDFS (***ID60, Lec10, 39-51***)

- Similar motivations:
  - Many inexpensive commodity hardware and failures are very common
  - Many big files: millions of files, ranging from MBs to GBs
  - Two types of reads
    - Large streaming reads
    - Small random reads
  - Once written, files are seldom modified
    - Random writes are supported but do not have to be efficient
  - High sustained bandwidth is more important than low latency
  - Block size is 128MB (by default)

Lecture 10: Distributed File System

HDFS vs GFS (***ID61, Lec10, 52***)

	Hadoop Distributed File System (HDFS)	Google File System (GFS)
Platform	Cross Platform (Linux, Mac, Windows)	Linux
Development	Developed in Java environment	Developed in C,C++ environment
Chunk Size	128 MB	64 MB
Node	NameNode / DataNodes	Master node & Chunk Server
Log	Editlog	Operational log
Write Operation	**No more than one writer at one time**	Can have multiple writers to one file at one time.
File Deletion	Deleted files are renamed into particular folder and then it will removed via garbage	Deleted files are not reclaimed immediately and are renamed in hidden name space and it will deleted after three days if it's not in use

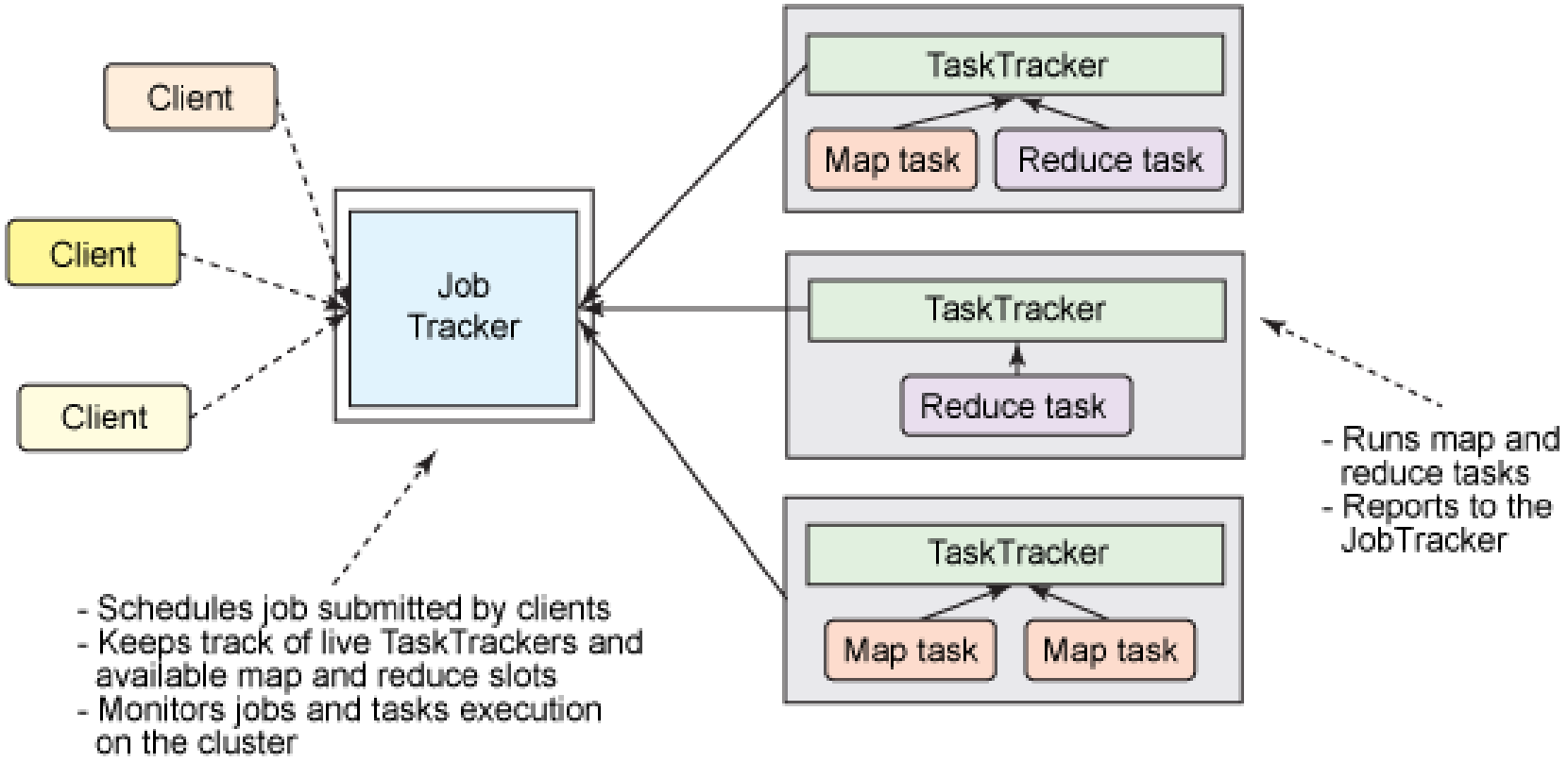
Lecture 11: Hadoop MapReduce and its Ecosystem

ID	Topics	Pages	Lecture
61	Hadoop MapReduce	15-22	11
62	Inputs and Outputs	23-25	11
63	MapReduce Examples	27-50	11
64	Hive and Pig	52-53	11

Lecture 11: Hadoop MapReduce and its Ecosystem

MapReduce Model (***ID61, Lec11, 15-22***)

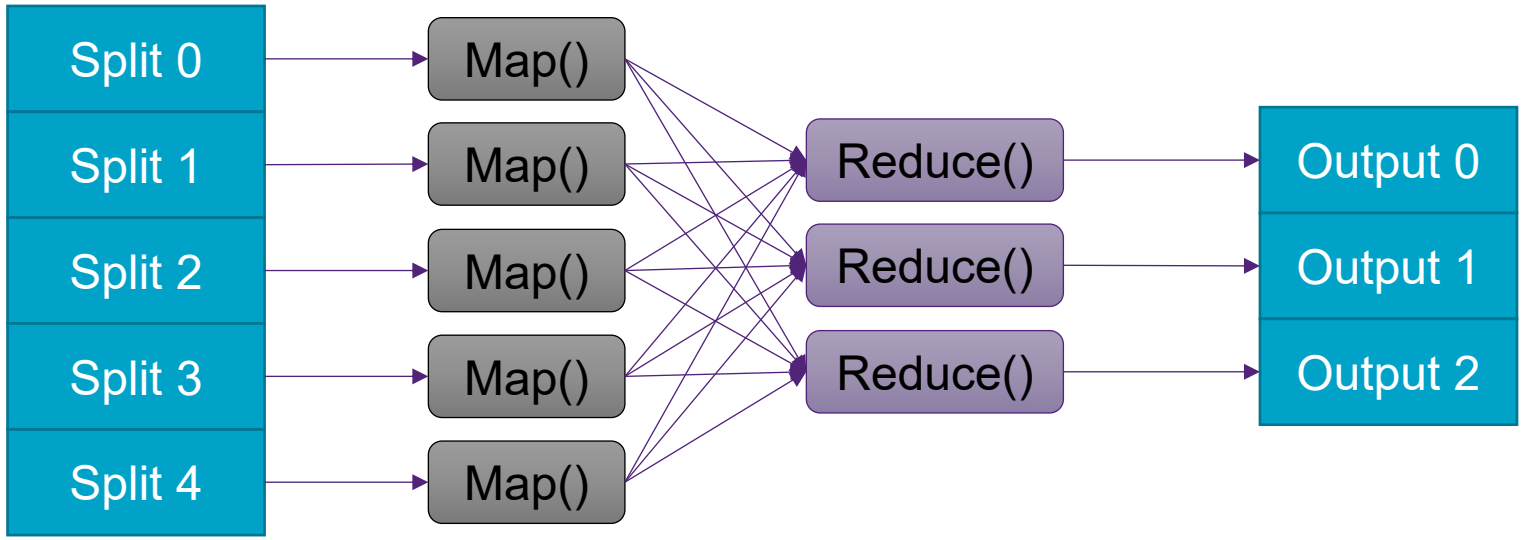
- Components:
  - Job Client
  - Job Tracker
  - Task Tracker
  - Task



Lecture 11: Hadoop MapReduce and its Ecosystem

MapReduce Model (***ID61, Lec11, 15-22***)

- Divide and Conquer
  - No communications between Map tasks
  - No communications between Reduce tasks
  - Need shuffle process to transfer data to reducer



Lecture 11: Hadoop MapReduce and its Ecosystem

Inputs and Outputs (***ID62, Lec11, 23-25***)

- The MapReduce framework operates exclusively on **<key, value>** pairs;
- The framework views the input to the job as a set of **<key, value>** pairs and produces a set of **<key, value>** pairs as the output of the job.
- Input and Output types of a MapReduce job:

Function	Input	Output	Note
Map	<k_1, v_1> E.g. <lineNum, "a b c b c a a c c">	List(<k_2, v_2>) e.g. (<"a", 1>, <"b", 1>, <"c", 1>, <"b", 1>, <"c",1>, <"a",1>, <"a",1>, <"c",1>, <"c",1>)	1. Convert splits of data into a list of <key, value> pairs. 2. Each input <k_1, v_1> will output a list of key/value pairs as intermediate results
Reduce	<k_2, List(v_2)> e.g. <"a", <1, 1, 1> >	<k_3, v_3> e.g. <"a", 3>, <"b", 2>, <"c", 4>	The value of <k_2, List(v_2)> in the intermediate result, List(v_2), represents the values of the same key k_2.

Lecture 11: Hadoop MapReduce and its Ecosystem

MapReduce Examples (***ID63, Lec11, 27-50***)

- Word Count ***
- Find the Highest/Average Temperature
- Find Word Length Distribution
- Find Common Friends
- Inverted Indexing

Lecture 11: Hadoop MapReduce and its Ecosystem

MapReduce Examples (***ID63, Lec11, 27-50***)

- Word Count **

Sample: Given the following text input, please use the below Map-Reduce framework to perform **word count** (case insensitive). Note that a combiner should be in use if necessary.

Spark is a model

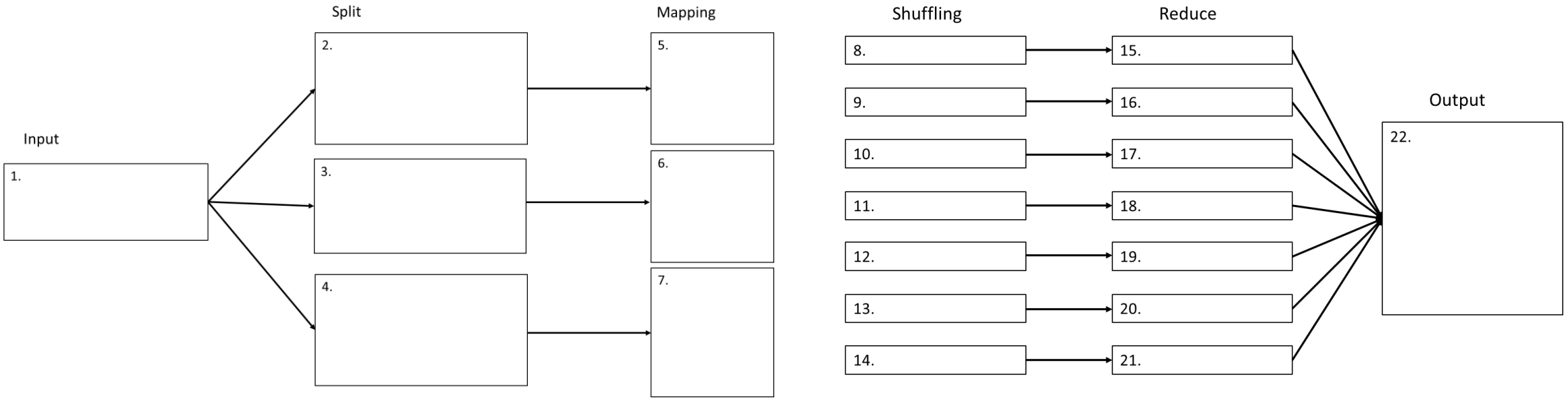
Model MapReduce is a model

HDFS is not a model

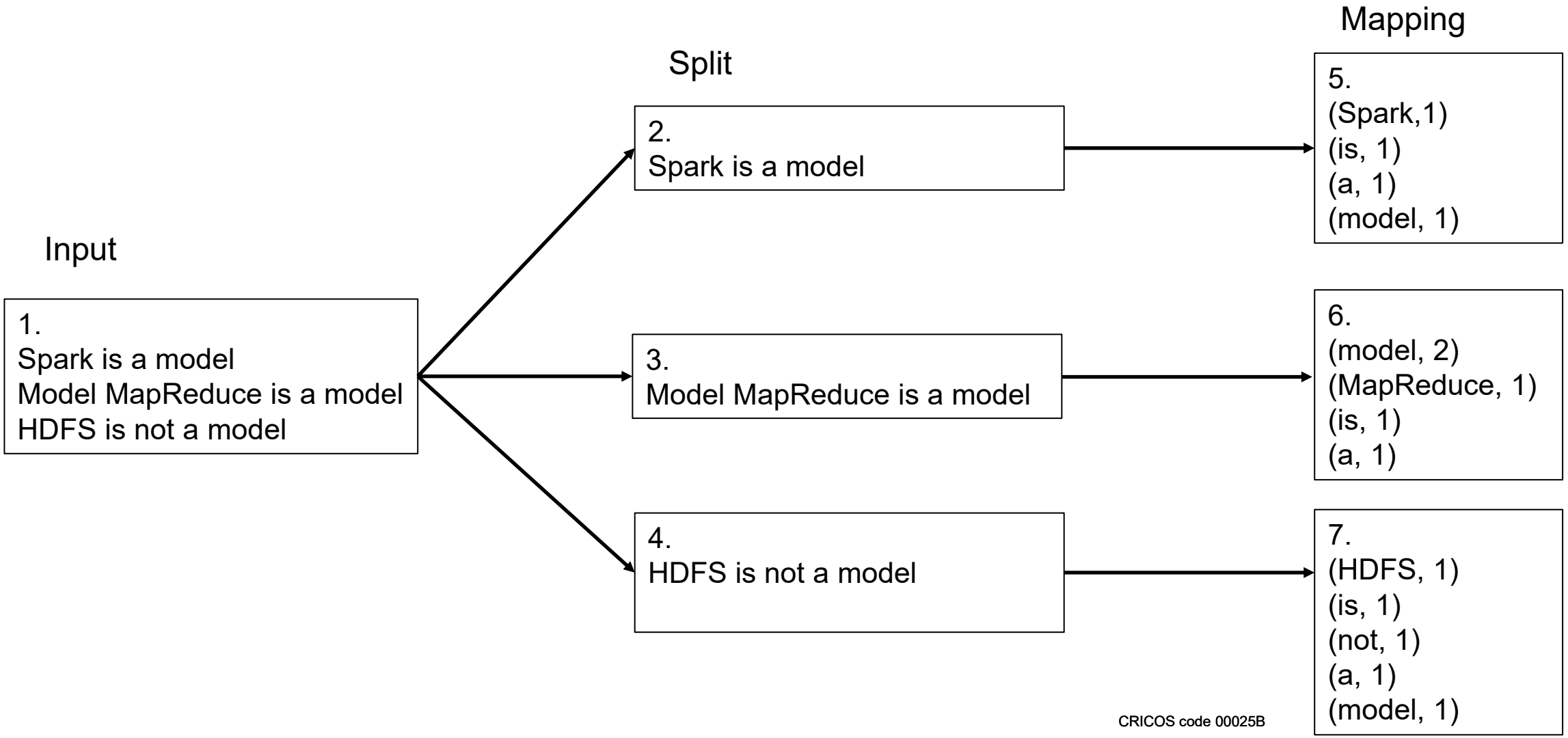
Processes: Input -> Split -> Map (3 mappers) -> Shuffle -> Reduce (7 reducers) -> Output

Question: where should we add a combiner to?

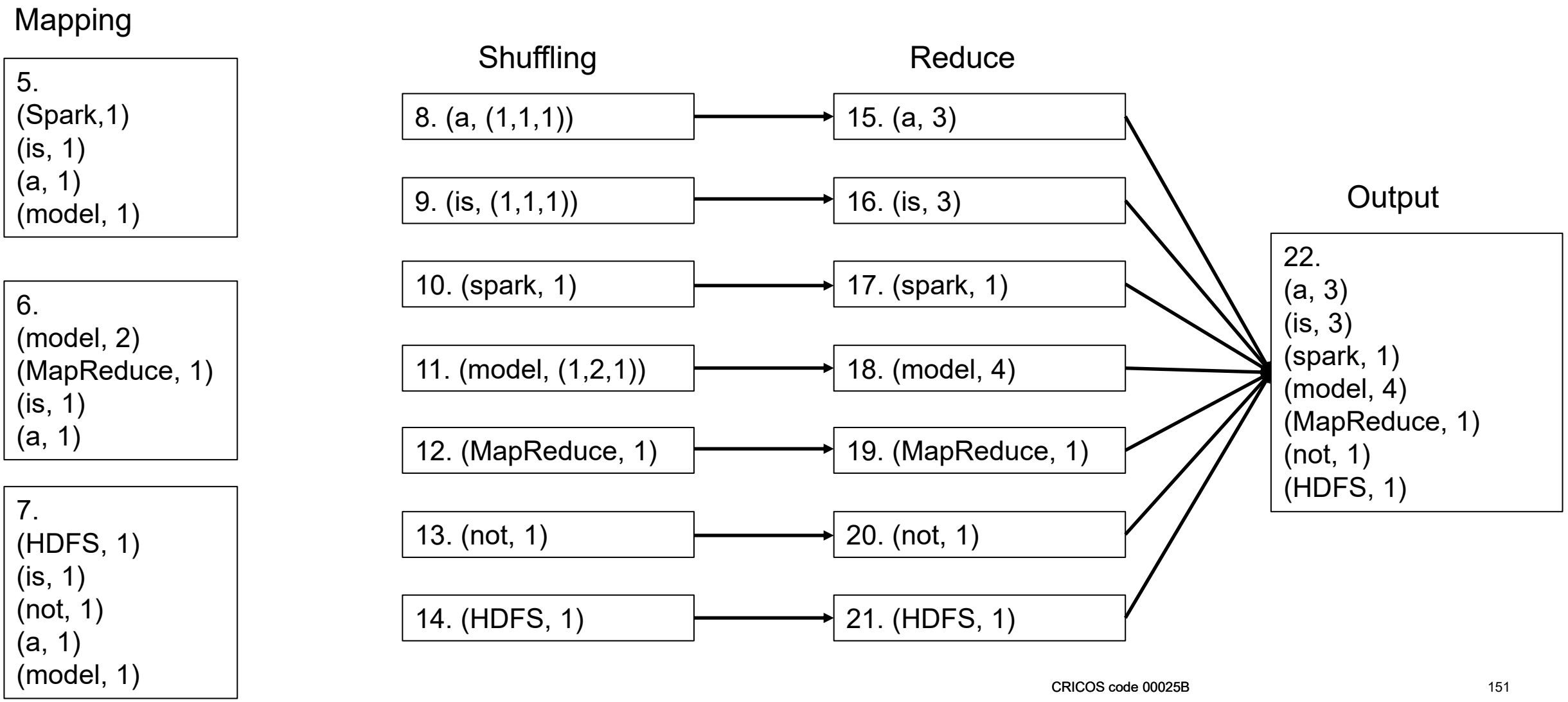
Lecture 11: Hadoop MapReduce and its Ecosystem



Lecture 11: Hadoop MapReduce and its Ecosystem



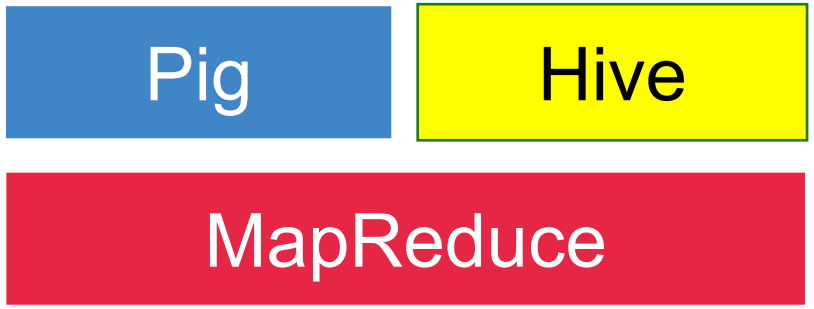
Lecture 11: Hadoop MapReduce and its Ecosystem



Lecture 11: Hadoop MapReduce and its Ecosystem

Hive and Pig (***ID64, Lec11, 52-53***)

- Hadoop is great for large-data processing!
  - But writing Java programs for everything is verbose and slow
  - Not everyone wants to (or can) write Java code
- Solution: develop higher-level data processing languages and convert to Hadoop jobs
- Hive: HQL is like SQL
  - Query language is HQL, a variant of SQL
  - Tables stored on HDFS as flat files
  - Developed by Facebook, now open source
- Pig:
  - Pig Latin is a bit like Perl
  - Scripts are written in Pig Latin, a dataflow language
  - Developed by Yahoo!, now open source
  - Roughly 1/3 of all Yahoo! internal jobs



Lecture 12: Security & Privacy

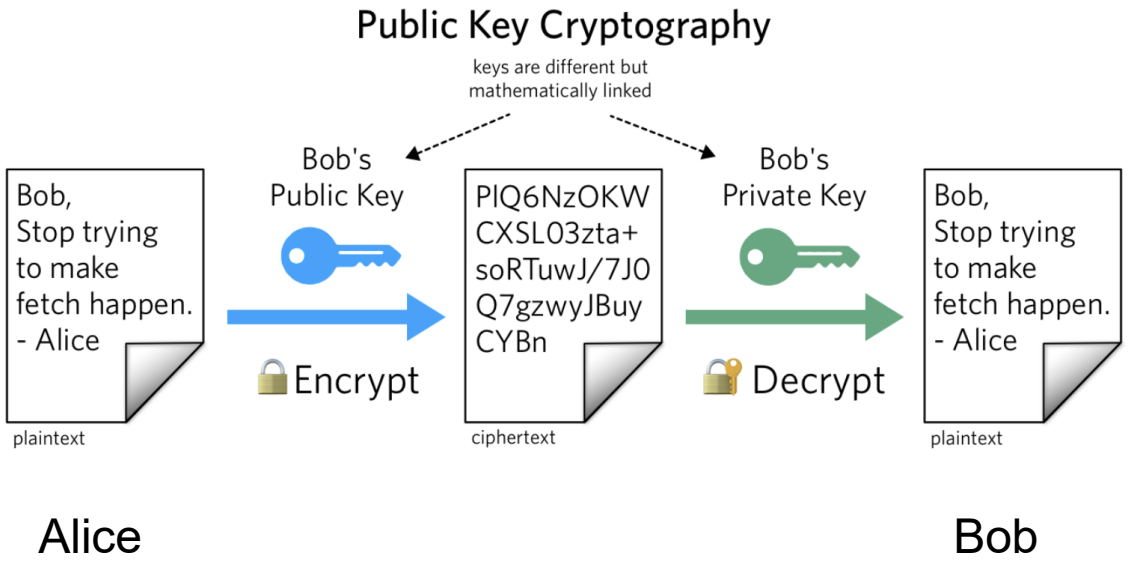
ID	Topics	Pages	Lecture
65	Asymmetric Encryption	24-26	12
66	Hashing	28-29	12
67	Digital Signature	34-37	12
68	PKI and CA	38-45	12

The first part of the guest lecture in Week 12 will **NOT** be assessed in the final exam.
Thus, only the second part (assessed content) has been kept in the current version.
For the exam preparation, please refer to the page numbers in the **reduced version**.
If you're interested in the first part of the guest lecture, you can download it in BB.

Lecture 12: Security & Privacy

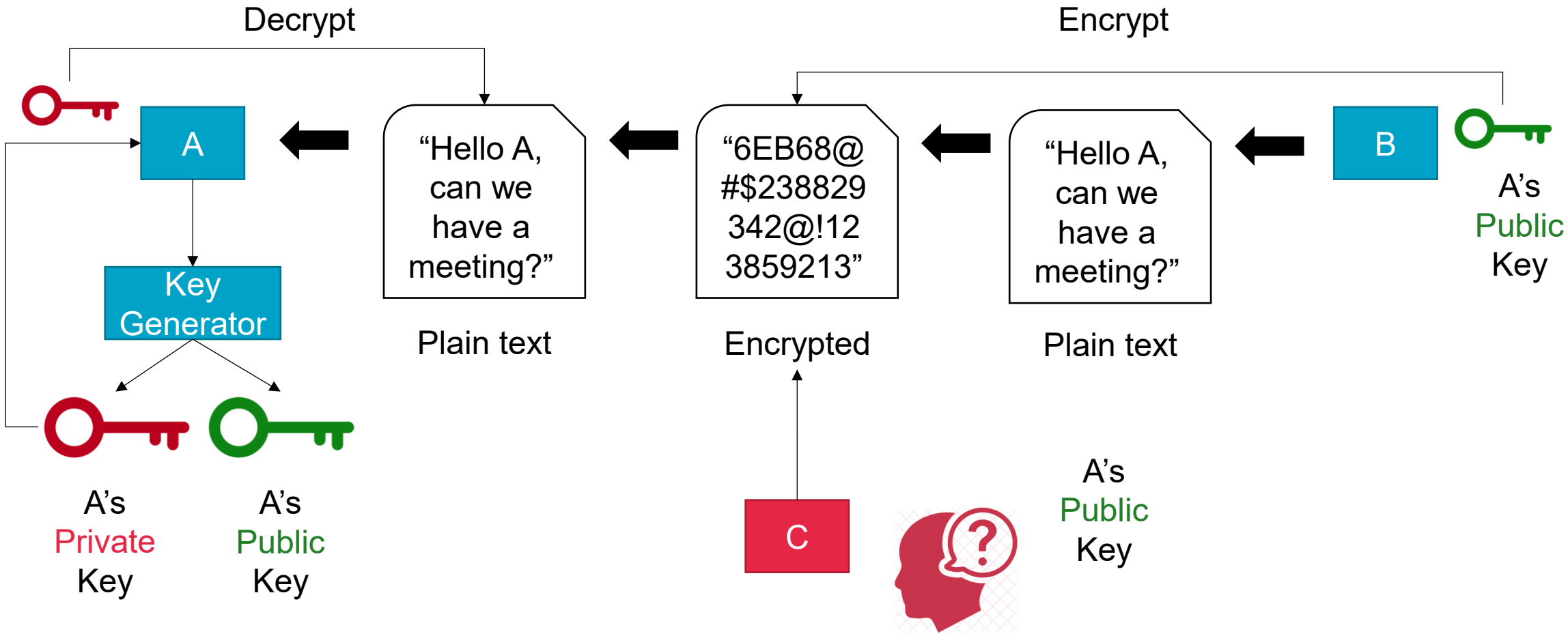
Asymmetric Encryption (***ID65, Lec12, 24-26***)

- Asymmetric encryption relies on the use of two different keys, namely a private key and a public key.
- The private key is known only to its owner
- the public key is commonly available.
- A document that was encrypted with a public key can only be correctly decrypted with the corresponding private key.
- asymmetric encryption is almost always computationally slower than symmetric encryption.
- Typical algorithms: RSA etc.



Lecture 12: Security & Privacy

Asymmetric Encryption (***ID65, Lec12, 24-26***)



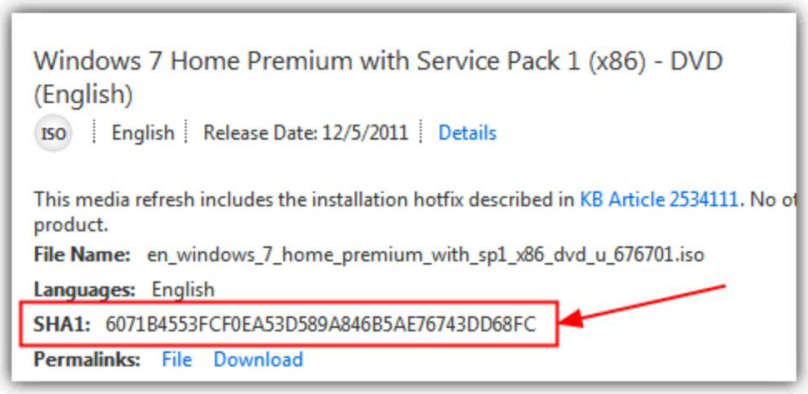
Lecture 12: Security & Privacy

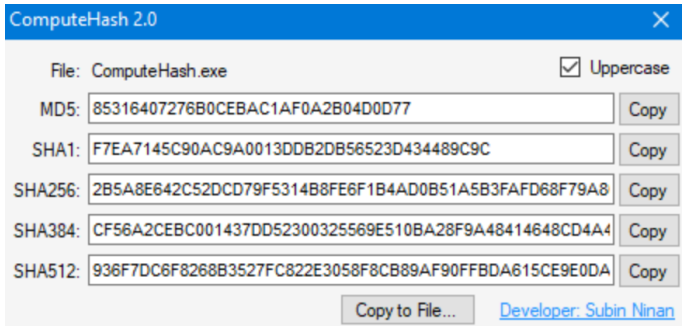
Hashing (***ID66, Lec12, 28-29***)

"Alex" -> a08372b70196c21a9229cf04db6b7ceb

- The *hashing* mechanism is used when a one-way, non-reversible form of data protection is required.
- Once hashing has been applied to a message, it is locked and no key is provided for the message to be unlocked.
- Hashing technology can be used to derive a hashing code or *message digest* from a message, which is often of a fixed length.
- The message sender can then utilize the hashing mechanism to attach the message digest to the message.
- The recipient applies the same hash function to the message to verify that the produced message digest is identical to the one that accompanied the message.
- Any alteration to the original data results in an entirely different message digest and clearly indicates that tampering has occurred.
- E.g. the storage of passwords, data integrity validation
- Typical algorithm: MD5, SHA-1, and SHA-2 (224, 256, 384, or 512 bits)

Microsoft



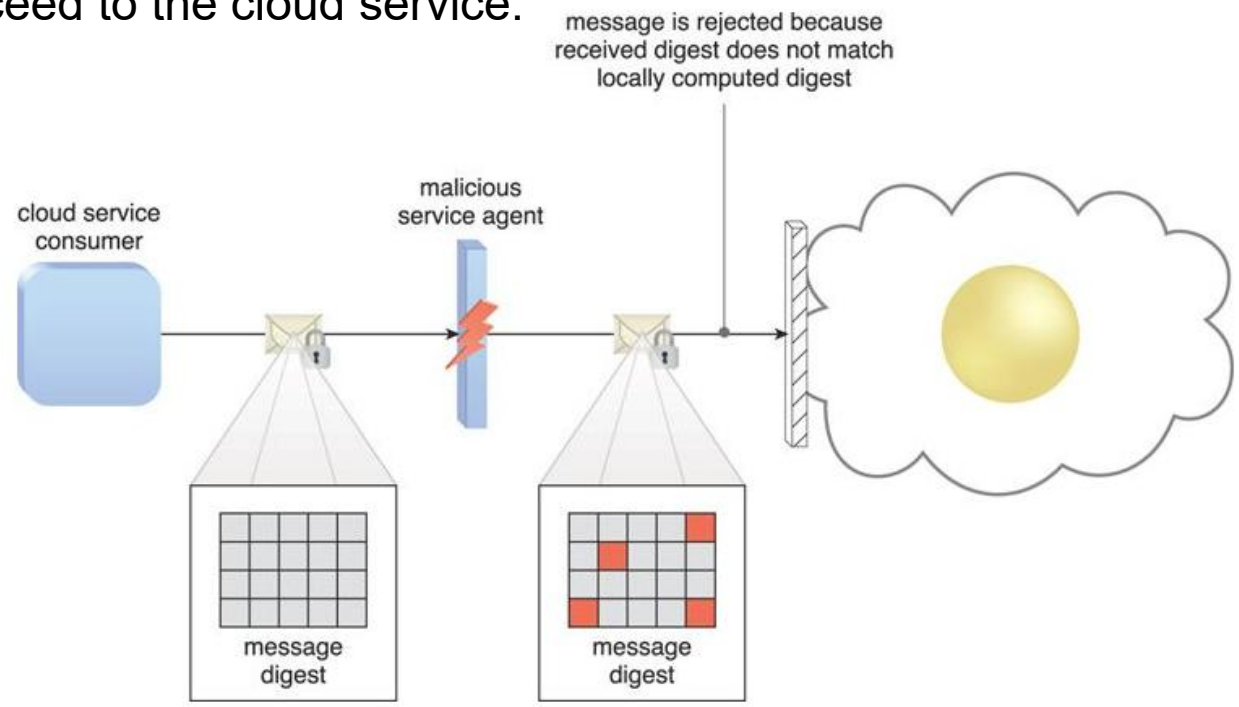


https://www.raymond.cc/blog/7-tools-verify-file-integrity-using-md5-sha1-hashes/

Lecture 12: Security & Privacy

Hashing (***ID66, Lec12, 28-29***)

- A hashing function is applied to protect the integrity of a message that is intercepted and altered by a malicious service agent, before it is forwarded.
- The firewall can be configured to determine that the message has been altered, thereby enabling it to reject the message before it can proceed to the cloud service.



Lecture 12: Security & Privacy

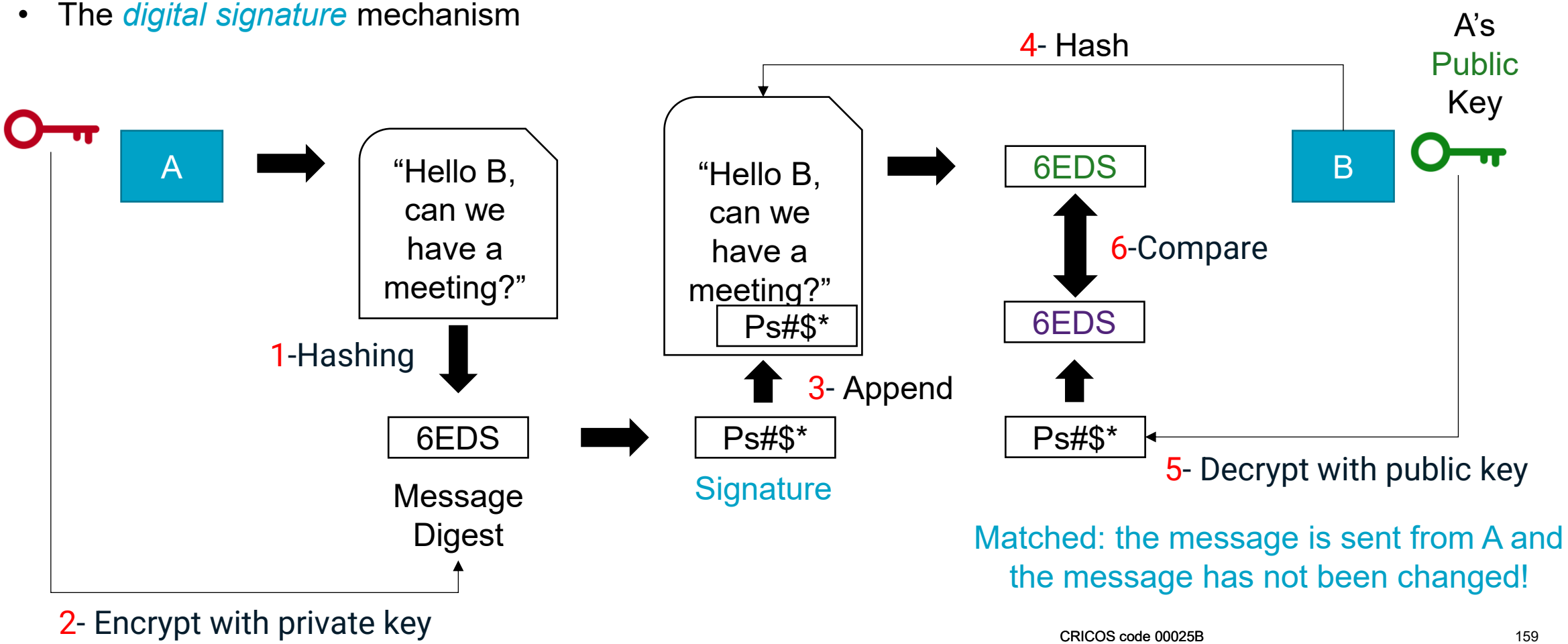
Digital Signature (***ID67, Lec12, 34-37***)

- The *digital signature* mechanism is a means of providing data authenticity and integrity through authentication and non-repudiation.
  - A message is assigned a digital signature prior to transmission,
  - The signature will become invalid if the message has unauthorized modifications.
  - A digital signature provides evidence that the message received is the same as the one created by its rightful sender.
- Both hashing and asymmetrical encryption are involved in the creation of a digital signature
  - a message digest is encrypted by a private key and appended to the original message.
  - The recipient verifies the signature validity and uses the corresponding public key to decrypt the digital signature, which produces the message digest.
  - The hashing mechanism can also be applied to the original message to produce this message digest. Identical results from the two different processes indicate that the message maintained its integrity.

Lecture 12: Security & Privacy

Digital Signature (***ID67, Lec12, 34-37***)

- The *digital signature* mechanism



Lecture 12: Security & Privacy

Digital Signature (***ID67, Lec12, 34-37***)

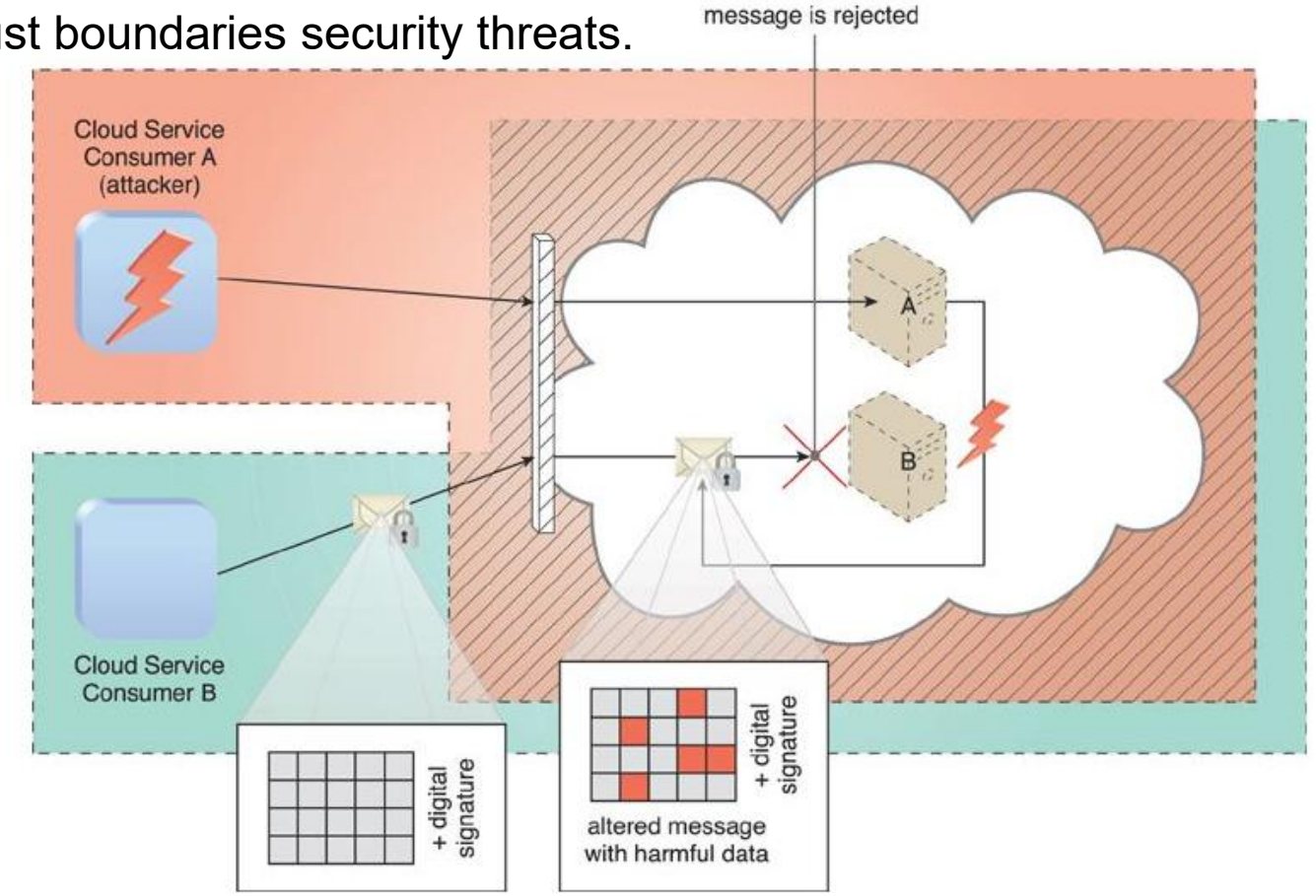
Process:

1. The sender selects the file to be digitally signed in the document platform or application and calculates the unique hash value of the file content (1-hashing).
2. This hashed value is encrypted with the sender's private key (2- encrypt with private key) to create the digital signature
3. The digital signature is appended to the plain text message (3-append).
4. The receiver use the same hash function to calculates the hash value of the plain text messages (4-hashing)
5. The receiver decrypt the digital signature with sender's public key (5- decrypt with public key).
6. The receiver compares the hash value of the plain text in Step 4 and the decrypted digital signature in Step 5 (6-Compare).

Lecture 12: Security & Privacy

Digital Signature (***ID67, Lec12, 34-37***)

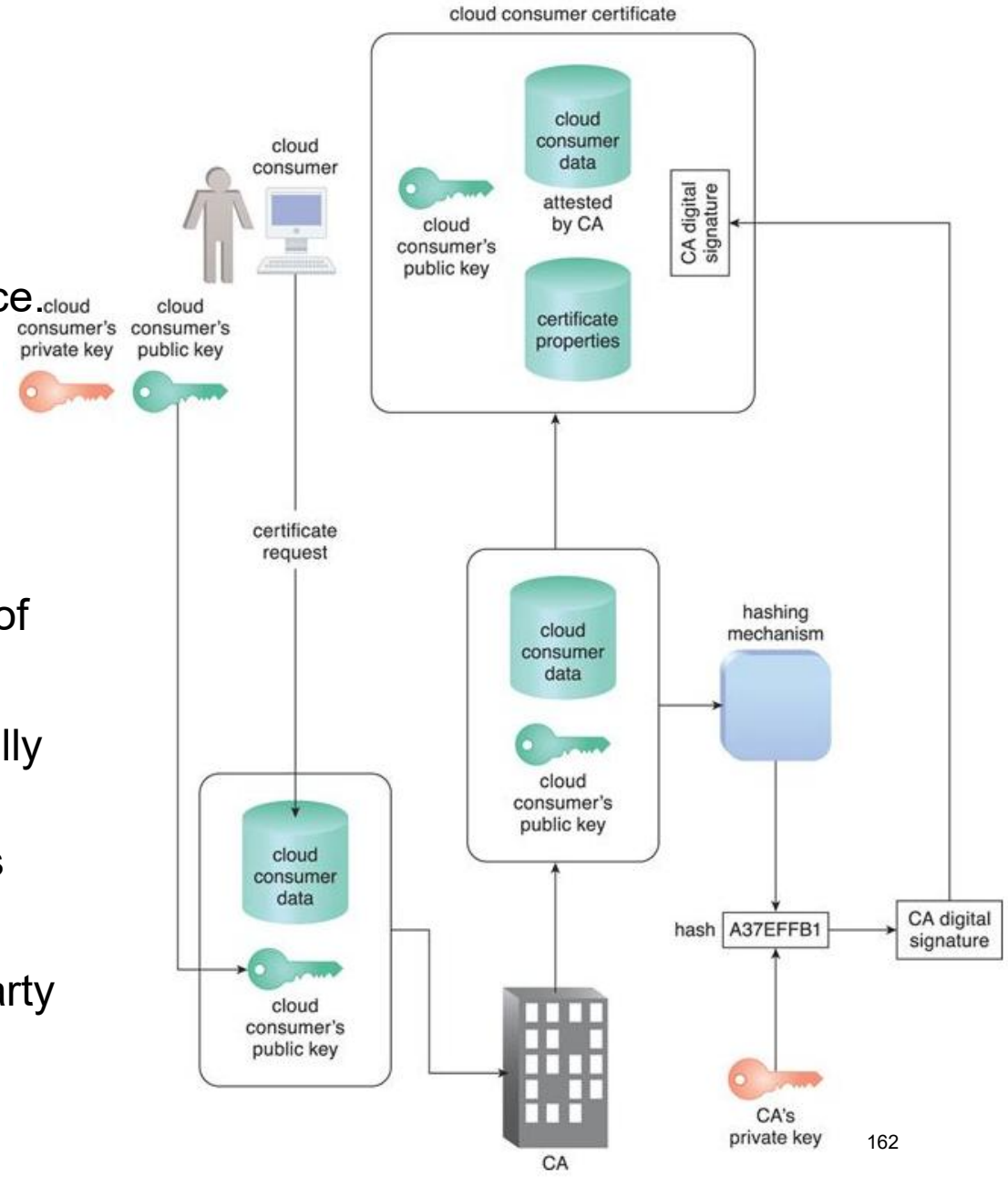
- The digital signature mechanism helps mitigate the malicious intermediary, insufficient authorization, and overlapping trust boundaries security threats.



Lecture 12: Security & Privacy

PKI and CA (***ID68, Lec12, 38-45***)

- PKI is the cornerstone of secured Internet and e-commerce.
- PKI mechanism exists as a system of protocols, data formats, rules, and practices that enable large-scale systems to securely use public key cryptography.
- This system is used to associate public keys with their corresponding key owners while enabling the verification of key validity.
- PKIs rely on the use of digital certificates, which are digitally signed data structures that bind public keys to certificate owner identities, as well as to related information, such as validity periods.
- Digital certificates are usually digitally signed by a third-party certificate authority (CA).



Lecture 12: Security & Privacy

PKI and CA (***ID68, Lec12, 38-45***)

Certification authority (CA): issues a **digital certificate** and binds a public key to particular entity E (e.g., Jennifer).

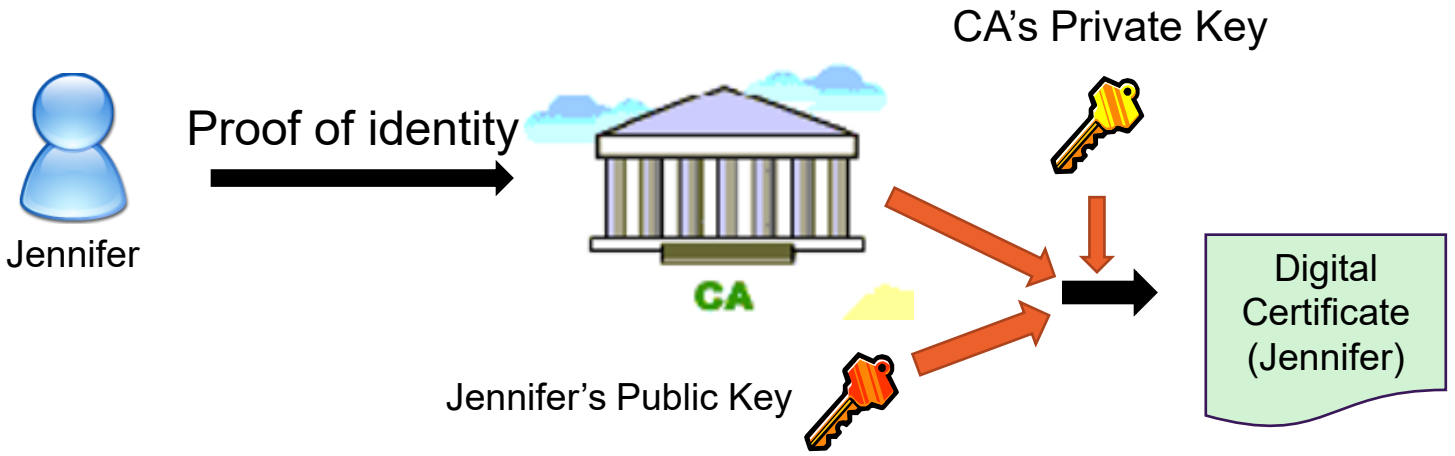
E (e.g. person, router) registers its public key with CA.

E provides “proof of identity” to CA.

CA creates a certificate binding E to its public key.

The certificate containing E’s public key digitally signed by a specific CA

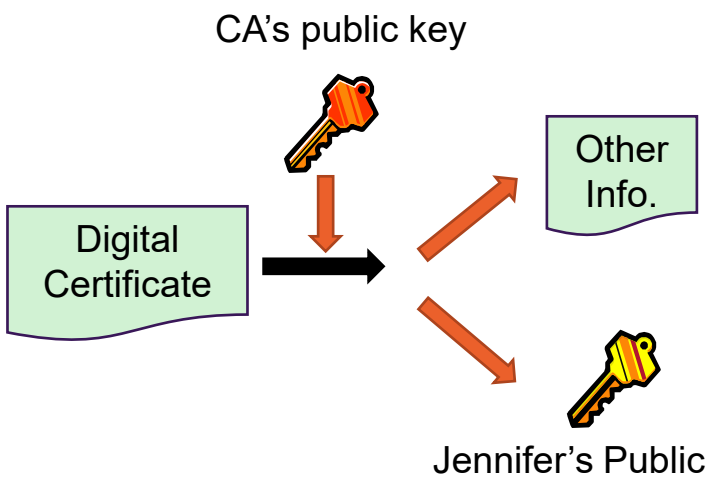
- CA says “this is E’s public key”

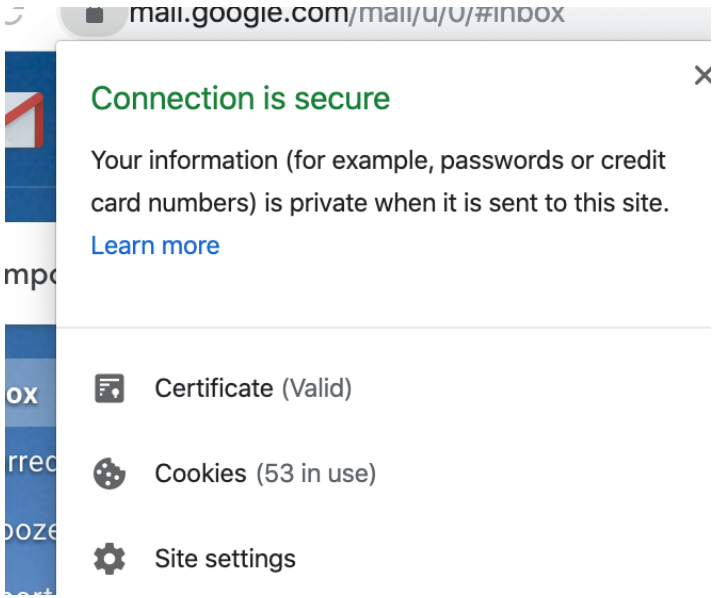


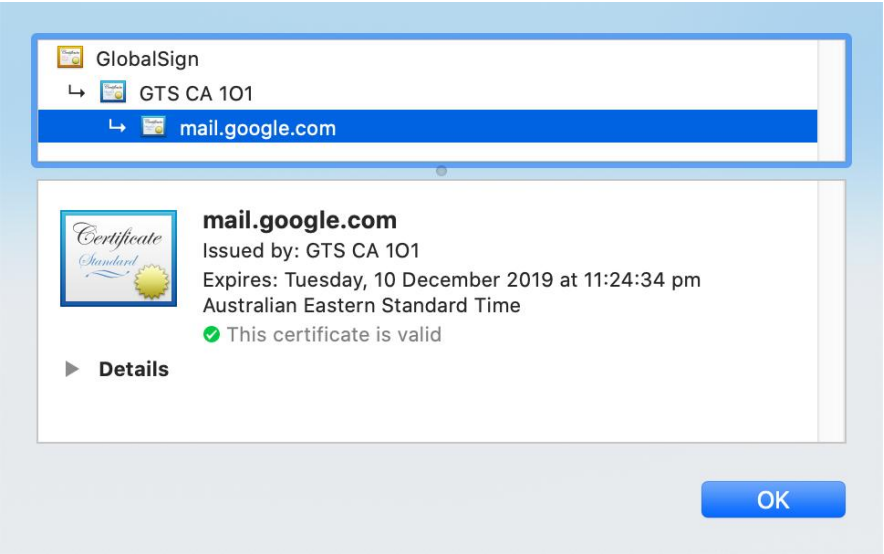
Lecture 12: Security & Privacy

PKI and CA (***ID68, Lec12, 38-45***)

In order to get the public key of Jennifer:







For Example: In google chrome, go to the https web page (say https://mail.google.com), click on the lock next to the URL, then click on "certificate information", click on the "Details" tab, and then find "Subject Public Key Info".

Back to Docker

https://security.stackexchange.com/questions/16085/how-to-get-public-key-of-a-secure-webpage

Go to Docker

Career Path and Certificates

Professional Data Engineer

A **Professional Data Engineer** enables data-driven decision making by collecting, transforming, and visualizing data. The Data Engineer designs, builds and maintains data processing systems with focus on the security, reliability, fault-tolerance, scalability, fidelity, and efficiency of such systems. The Data Engineer also analyzes data to gain insight into business outcomes, builds statistical models to support decision-making, and creates machine learning models to automate and simplify key business processes.

Professional Cloud Architect

A **Professional Cloud Architect** enables organisations to leverage Google Cloud technologies. With a thorough understanding of cloud architecture and Google Cloud Platform, this individual can design, develop, and manage robust, secure, scalable, highly available, and dynamic solutions to drive business objectives.

Associate Cloud Engineer

An **Associate Cloud Engineer** deploys applications, monitors operations, and manages enterprise solutions. They use Google Cloud Console and the command-line interface to perform common platform-based tasks to maintain one or more deployed solutions that leverage Google-managed or self-managed services on Google Cloud.

https://inthecloud.withgoogle.com/certification-journey-22/dl-cd.html

SECaT

Good luck with your final exam!