COMP 3010

Week 7, Tuesday Oct 17
Dr. Sara Rouhani

Today

Performance

Failure

Attacks

Cache

Why we do caching?

What things we store?

Cache

Why we do caching?

Key-value pairs

Temporary store *things* which we can then fetch.

Memcached is a high-performance, distributed memory object caching system.

What is the difference if we store sessions in memcached vs databases?

Where we do caching?

Client?

Web Server?

Between?

What things are often cached?

What things are often cached?

User profiles, User info from user id.

Responses to requests, e.g. Database query results

Configuration data.

Web content that **does not change on every request**.

...

What is good to cache

What is good to cache

Is frequently accessed.

Takes time or resources to generate or fetch.

Does not change frequently, or if it does, the application can tolerate serving slightly stale data for a short period.

Suggestions (Netflix/Amazon)

Trending things (items/movies)

Why can we not cache POST requests?

Why can we not cache POST requests?

POST request is to make a change or initiate an action on the server

Safety and Idempotence

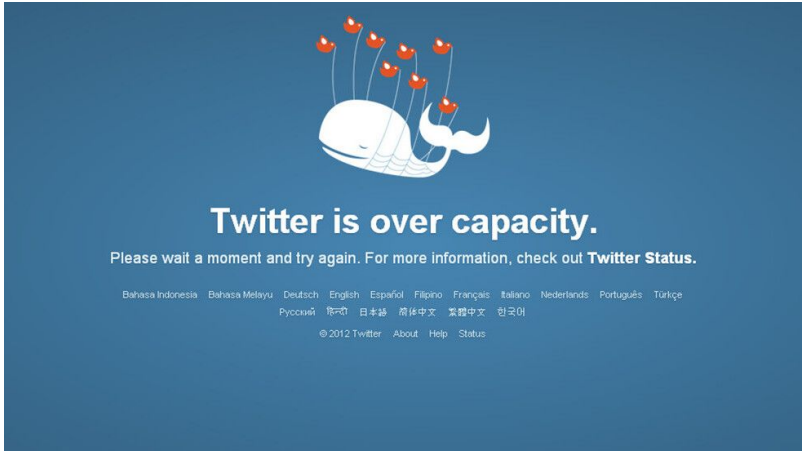
Different responses

Sharing errors

Should recover well? Send a fail whale?

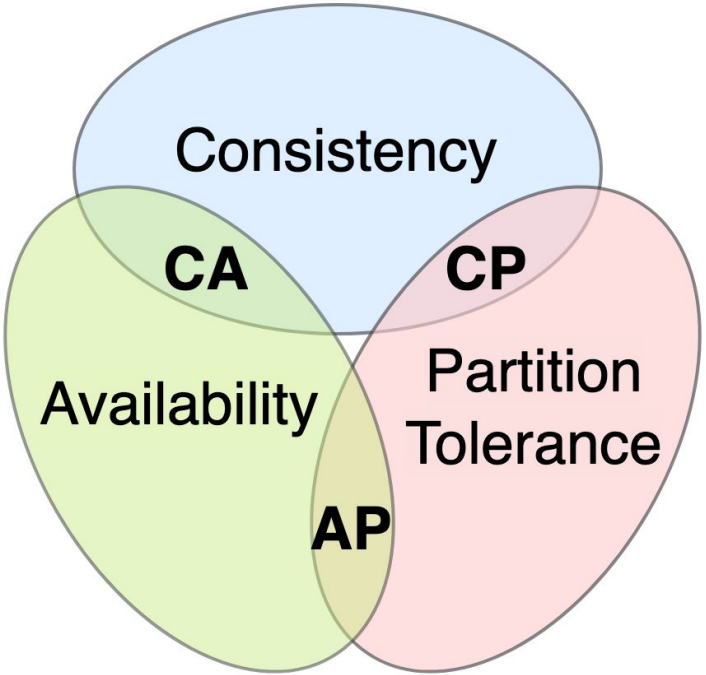
Can we predict all these extended failures?

We should be able to deal with some…



Can we assume that all components are working at all times?

CAP Theorem



https://en.wikipedia.org/wiki/CAP_theorem#/media/File:CAP_Theorem_Venn_Diagram.png

Example Error

Database is down... what can we do?

What if the database is down and memcache is still running?

Vice versa

DoS/DDoS

Dos Vs DDOS

What are the best requests for DDoS-ing? Why?

DoS/DDoS

Dos Vs DDOS

What are the best requests for DDoS-ing? Why?

Small request size, large response size

Attack surfaces

What parts of a system can be DoS'd?

Attack surfaces

What parts of a system can be DoS'd?

CPU time

Network

- Bandwidth
- Connections

Memory

How do we address each problem? - CPU

How do we deal with CPU issues?

How do we address each problem? - CPU

How do we deal with CPU issues?

Write efficient and optimized code.

Add more machines (horizontal scaling) or better machines

Break up the problem, make a new layer (microservices architecture or modularization)

How do we address each problem? - Bandwidth

How do we deal with Network Bandwidth issues?

How do we address each problem? - Bandwidth

Load Balancer, Multiple Locations

Can We Pass Less Data? Optimizing the data being transmitted

Rate Limiting (More Later)

How do we address each problem? - Connections

Called "port saturation" or "depletion" - how do we do it? Fix it?

How do we address each problem? - Connections

Called "port saturation" or "depletion" - how do we do it? Fix it?

Shorter timeouts (kick em out)... if possible

Horizontal scaling

How do we address each problem? - Memory

Rate limiting

A technique used to control the amount of requests a user can make to a service within a specified time period.

- Protect services from being overwhelmed by too many requests, which can lead to degraded performance or service outages.
- Limit DOS and DDOS attacks
- Manage resource allocation, ensuring that all users get a fair share of access.

Rate limiting

429 status code

HTTP/1.1 429 Too Many Requests

Retry-After: 90

Content-Type: application/json

...

Reddit Hug of Death

https://dezgo.com/

ism
A Kyndryl Company

A review of (some)

Architecture Styles & Design Patterns

for distributed systems on the cloud

Vahid Pourheidari

ism

agenda

- **Introduction**

PART I

- **Application Architecture Styles**
- **Design Patterns**

PART II

- **Full Stack Development Learning Paths**

450+

employees from British Columbia to New Brunswick

over 50%

of employees develop & support new technology

serving over

60

unique customers across Canada & globally

ism a kyndryl company

50 years

experience as an end-to-end managed IT service provider

320+

Microsoft certifications

leaders

in full stack solution integration for customers in diverse industries

unionized

delivery teams in Saskatchewan & British Columbia

innovative

top tech partnerships in cloud, network, apps

solution map

cloud

Migrate
Power Platform

Modern Workplace
365
Azure VD

Infrastructure
IaaS, BaaS, DRaaS, CaaS
Multicloud
AppOps

network

Management
SD-WAN
WAN
LAN
Voice

Infrastructure Support
Software Defined
Public Cloud Devices
Digital Bridge

End to End
Network Management
SIEM & NPM

applications

Development
DevOps
Modernization

Application Management
Architecture

Apps
CRM
ERP

infrastructure

Data Centre
Unix/Windows/Linux
Hardware
Storage

Mainframe
Mainframe Ops
Batch Control

Live Operations
Deskside
Service Desk

cross pillar services

delivery	defence	analytics
Assessment	Compliance	Operational reporting
Adoption	Governance	Predictive models
Migration & deployment	Monitoring	Machine Learning & AI
Integration	Security assessment	Insight Navigator
Staff Augmentation	Security tools	

Architecture

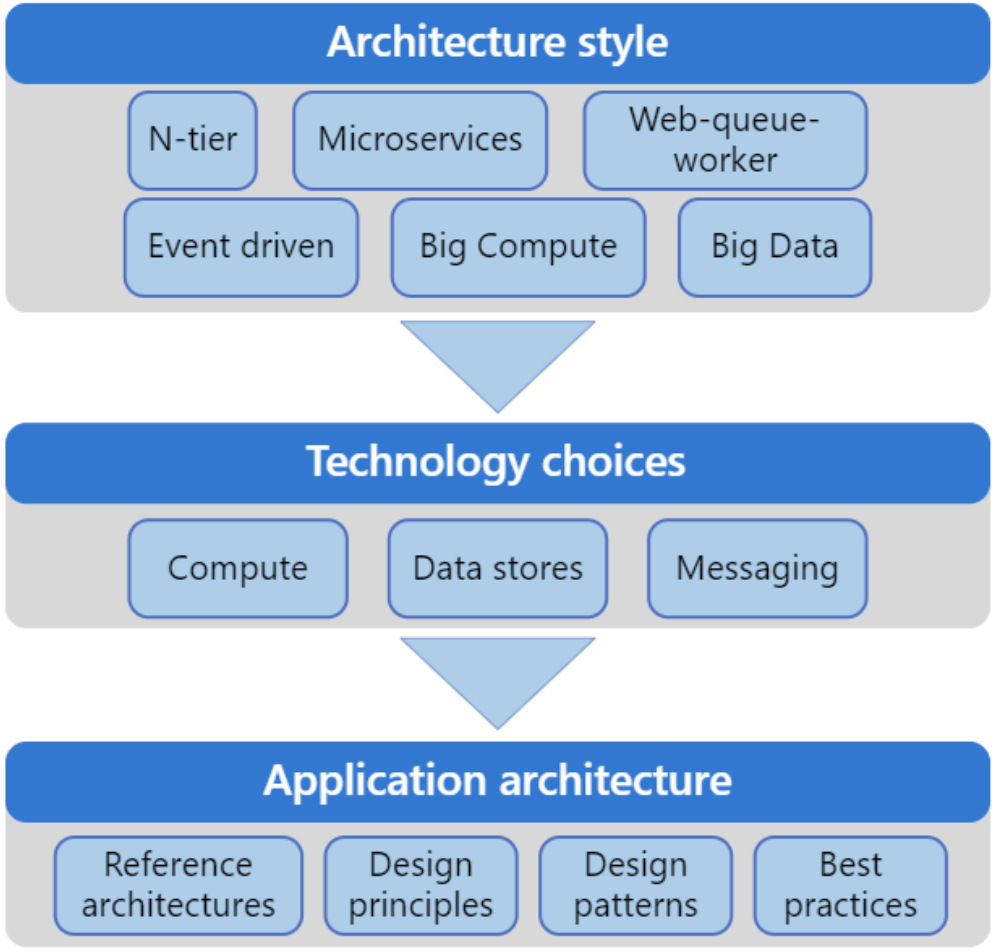
As an artifact

The Architecture of a system describes its overall static structure and dynamic behavior. It models the system's elements (which for IT systems are software, hardware and its human users), the externally manifested properties of those elements, and the static and dynamic relationships among them.

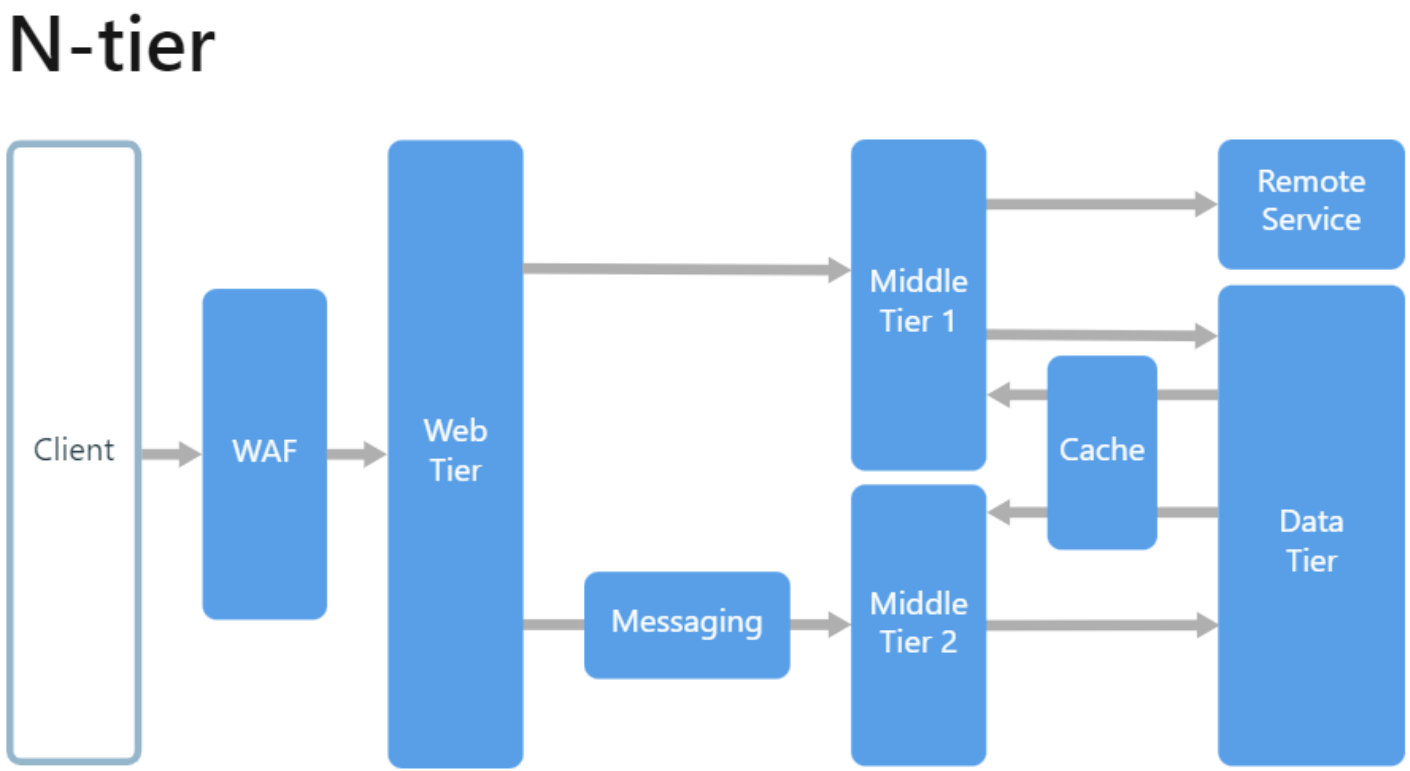
As a discipline

Architecture is an engineering discipline that studies methods of designing IT systems that provide a solution to a business problem The solution must satisfy functional and non-functional requirements in a way that best balances competing stakeholders' concerns and must take constraints into account.

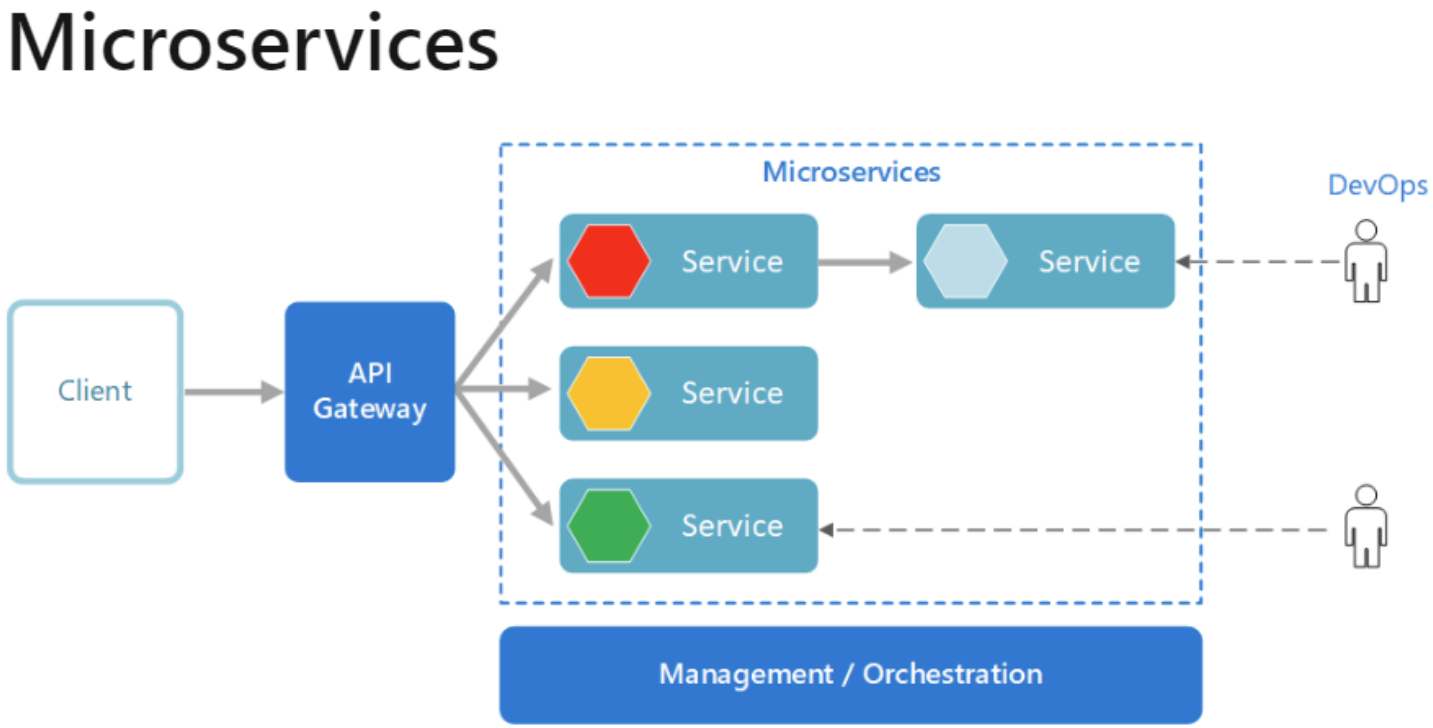
Architecture



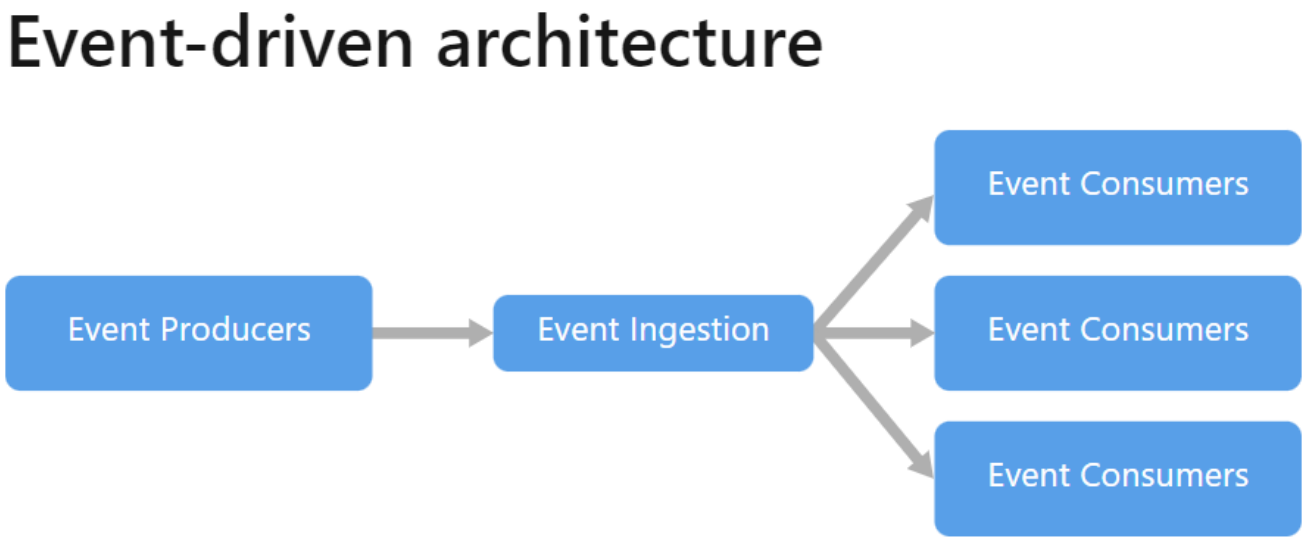
Architecture Style



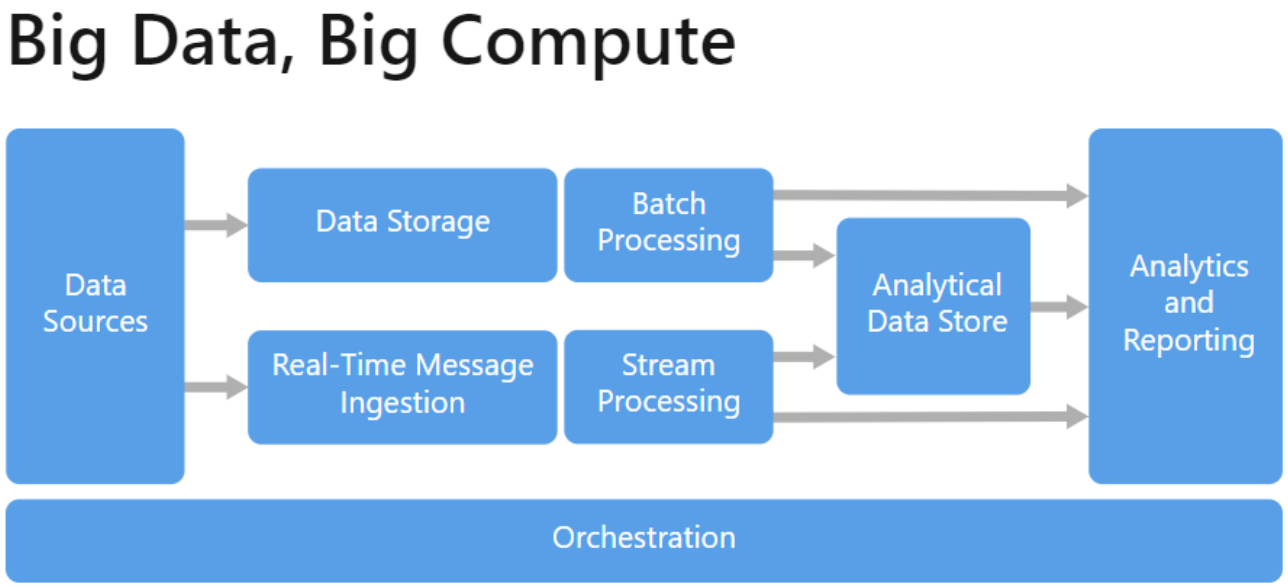
Architecture Style



Architecture Style



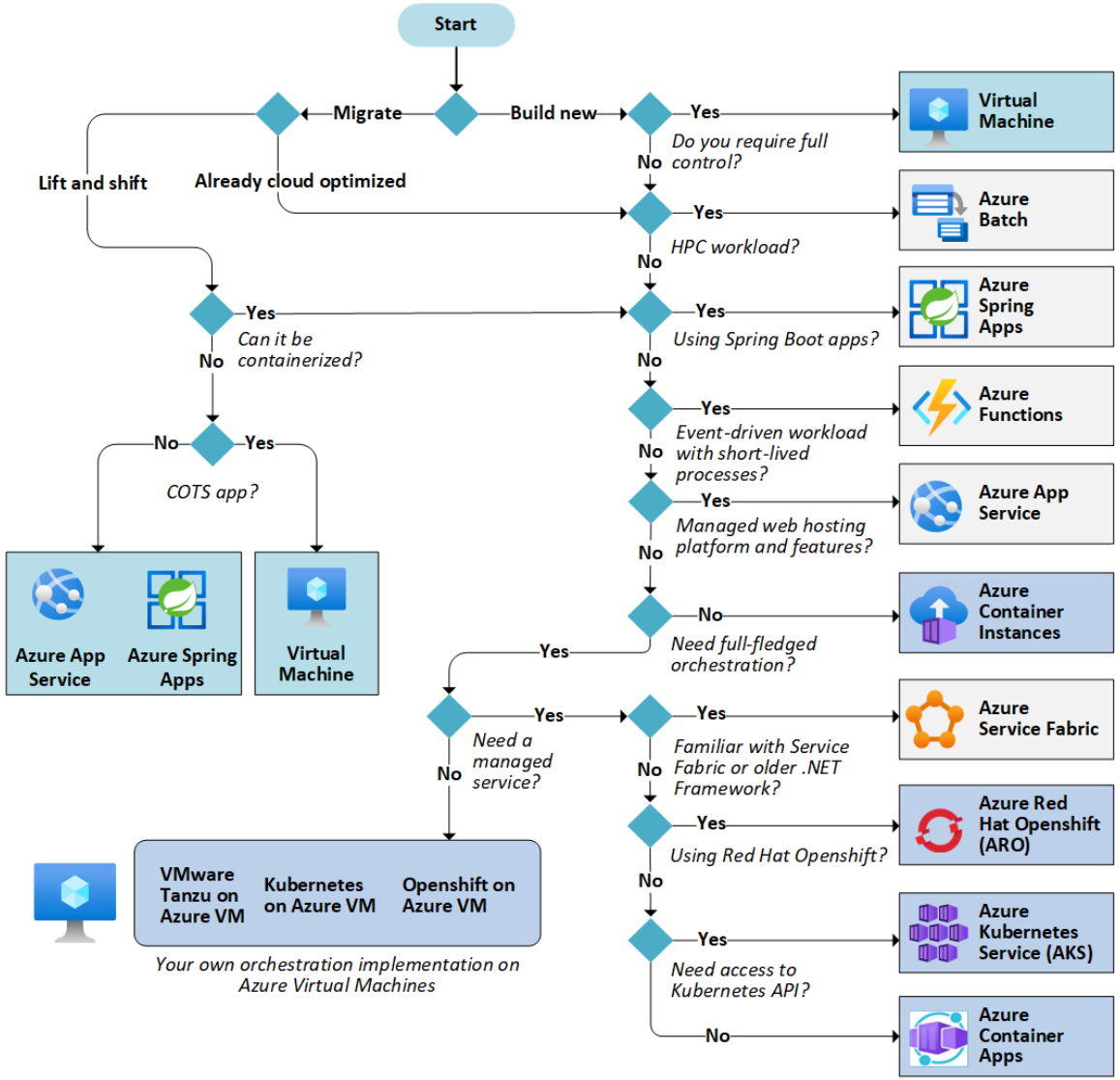
Architecture Style



Architecture Style

Architecture Style	Description	Domain
N-tier	Horizontal tiers divided by subnet	Traditional business domain. Frequency of updates is low.
Microservices	Decomposed services that call each other through APIs	Complicated domain. Frequent updates.
Event-driven	Producer/consumer. Independent view per sub-system.	IoT and real-time systems.
Big data	Divide a huge dataset into small chunks. Parallel processing on local datasets.	Batch and real-time data analysis. Predictive analysis using ML.
Big compute	Data allocation to thousands of cores.	Compute intensive domains such as simulations.

Choose a candidate service



Container exclusive services

- Azure Batch
- Azure Functions
- Azure App Service
- Azure Spring Apps
- Azure Service Fabric
- Kubernetes on Azure VM
- VMware Tanzu on Azure VM
- Openshift on Azure VM

Container compatible services

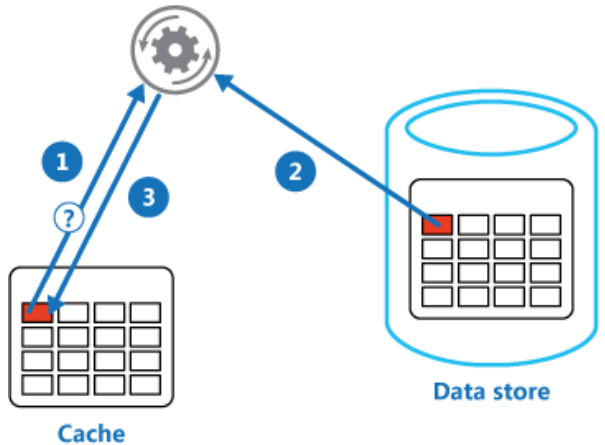
- Azure Batch
- Azure Functions
- Azure Service Fabric
- Azure Spring Apps
- Azure App Service

Design Pattern – Data Management

Cache-Aside Pattern

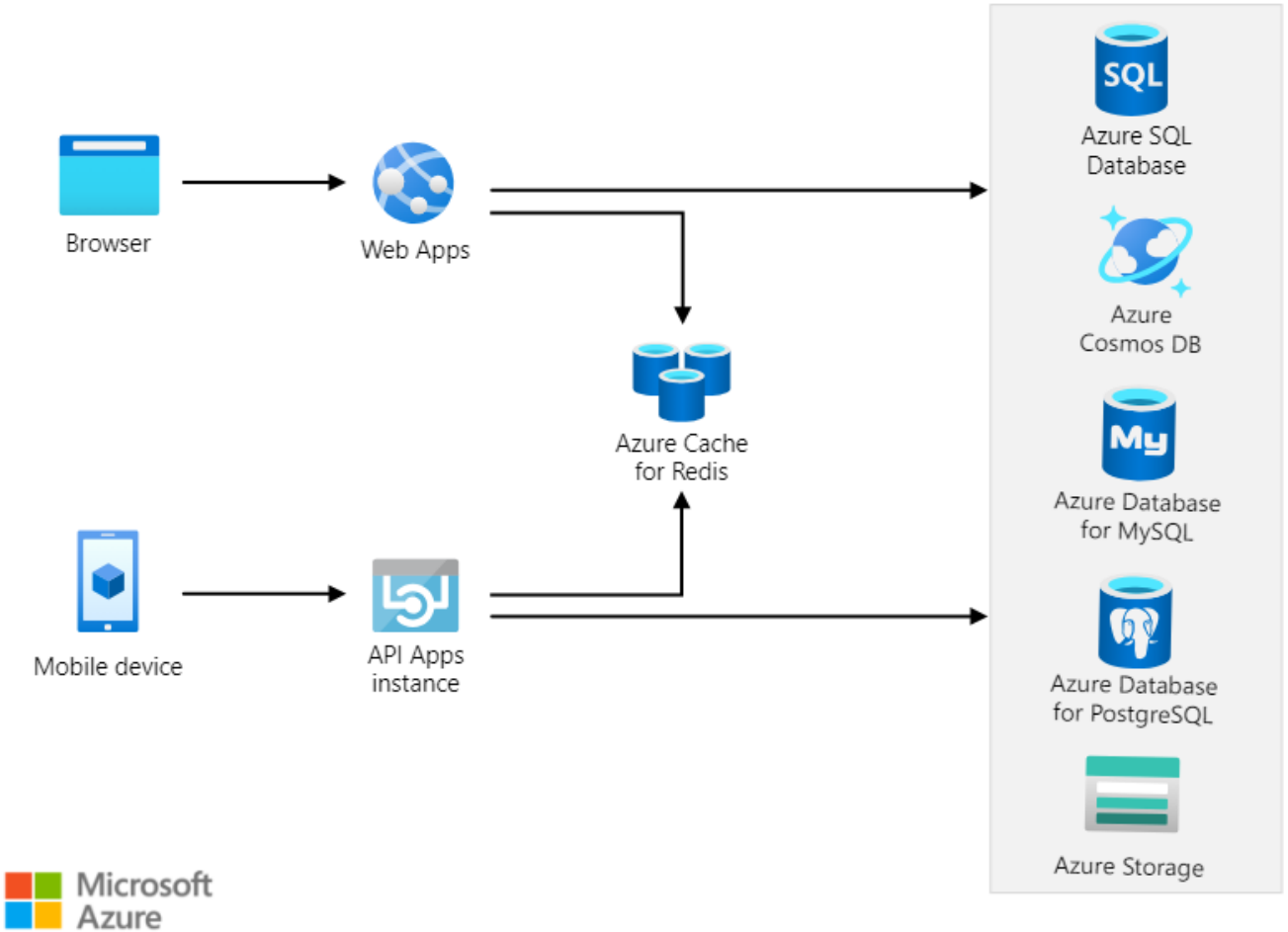
- Resource demand is unpredictable
- You have a cache which doesn't provide native read/write-through operations.

When the cached data set is statistic.



1: Determine whether the item is currently held in the cache.
2: If the item is not currently in the cache, read the item from the data store.
3: Store a copy of the item in the cache.

Design Pattern – Data Management

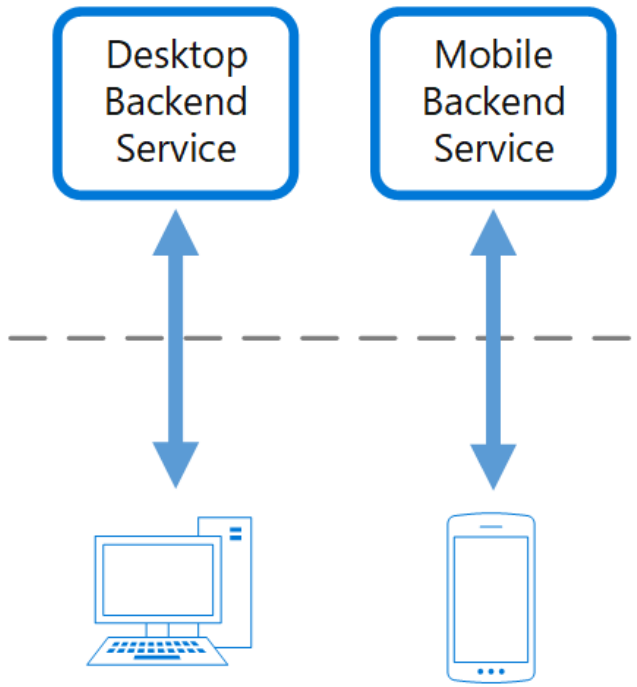


Design Pattern – Design & Implementation

Backends for Frontends

- Optimize the backend for the requirement of a specific interface.
- An alternative language is better suited for the backend of a different user interface.
- A shared backend service needs a significant development overhead.

- When different interfaces make the same or similar request to the backend.
- When you have only one interface interacting with the backend.

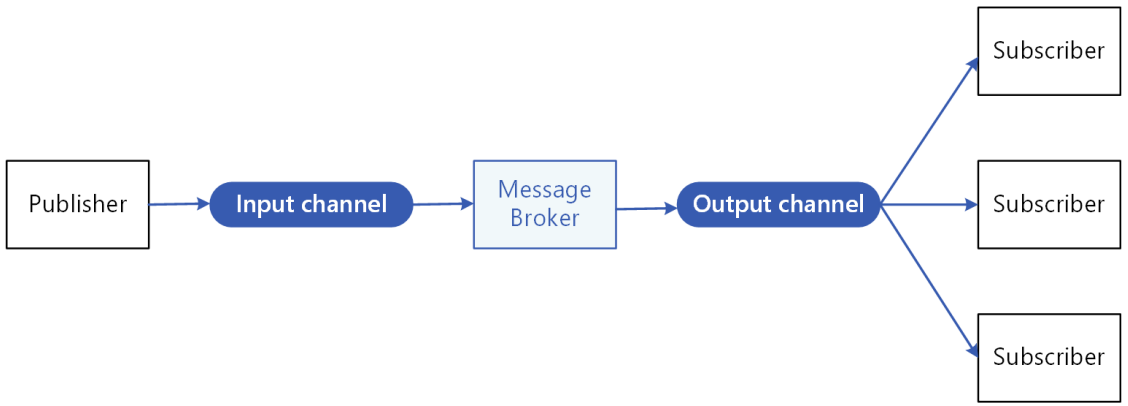


Design Pattern – Messaging

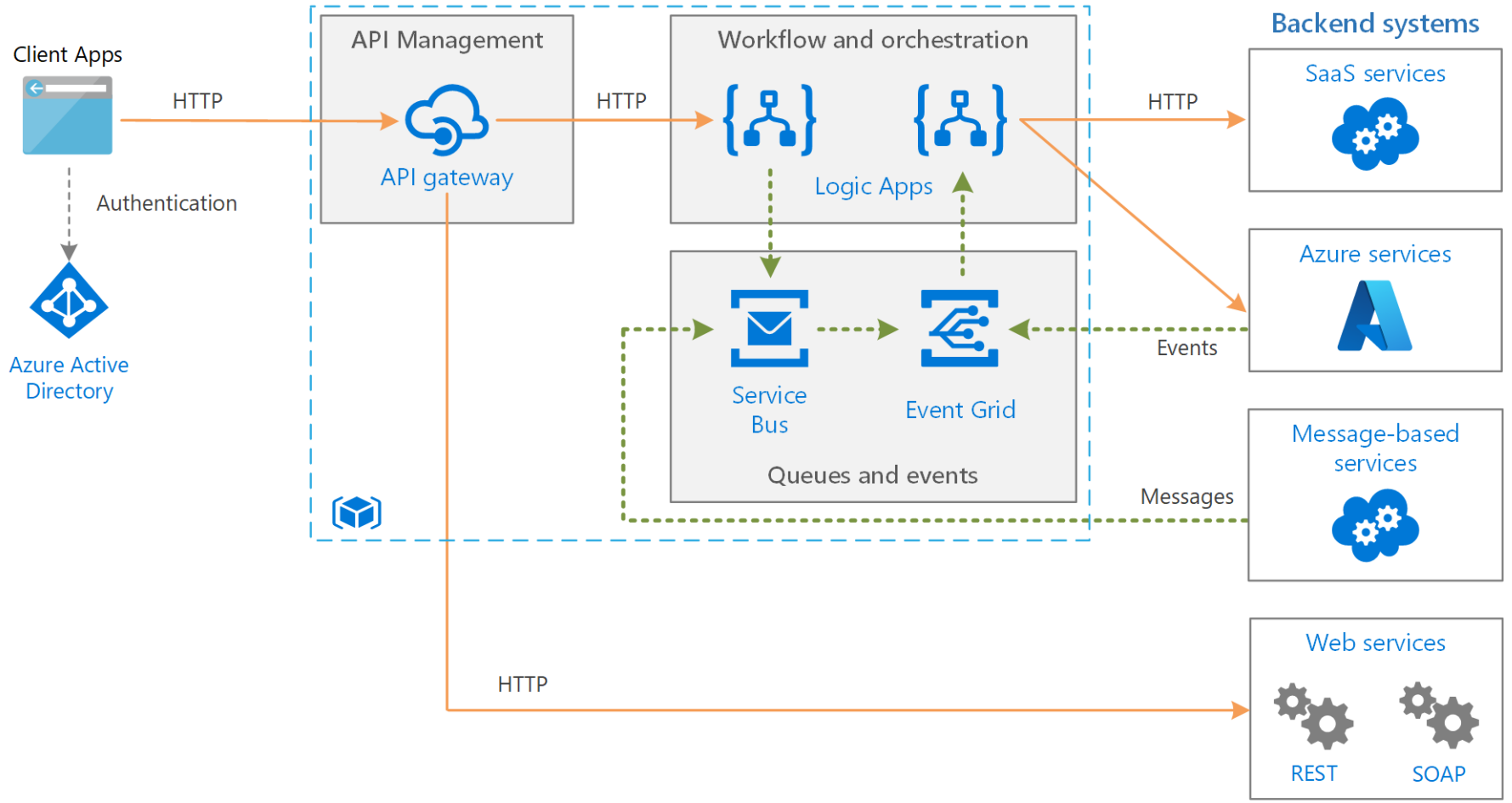
Publisher - Subscriber

- A need to broadcast information to multiple consumers.
- An application needs to communicate with one or more independent apps or services.
- There is no need for real-time responses.

- A few consumers who need different information.
- Near real-time interaction is a necessity.



Design Pattern – Messaging



A Kyndryl Company

Funding Your Way

A Map of Full-Stack Development Learning Paths

Vahid Pourheidari

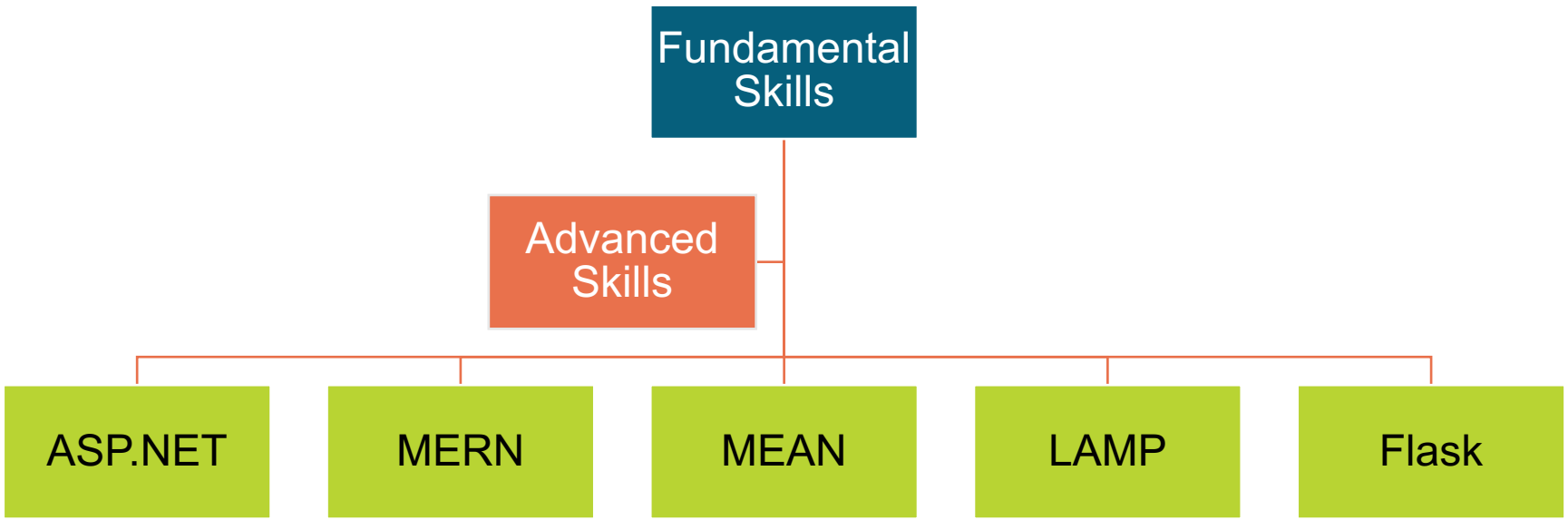
Why do learning paths matter?

Level Up Your Skills: The Power of Learning Paths

- At many companies, developers work on multiple projects with unique technical requirements, often using new toolsets for each project.
- Creating a personalized learning plan is crucial to make continuous progress and avoid confusion with the many available technologies and tools.
- Having a clear learning path helps focus on relevant skills and technologies for career goals.
- Continuously learning and improving skills leads to personal growth and increased job satisfaction.
- A skilled workforce benefits the organization's goals.

How Do these paths work?

Navigating the Journey



Essential Skills for Full Stack Developers

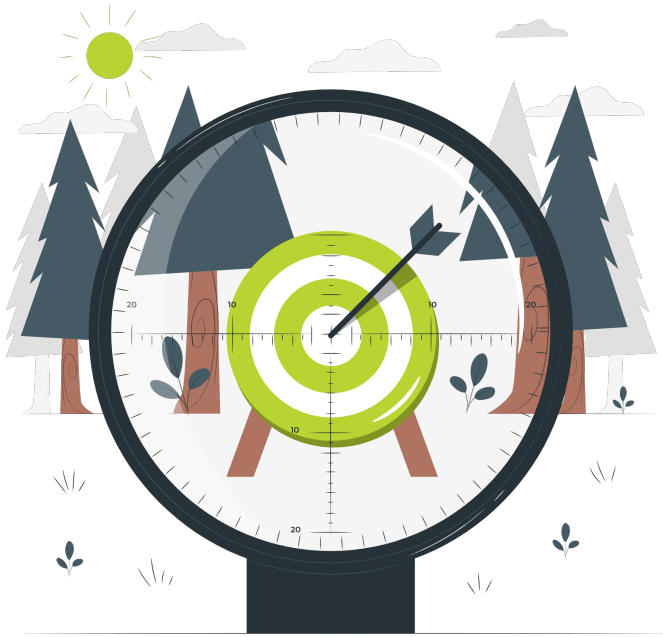
What You Need to Know to Succeed

Fundamental Skills:

- At least one back-end programming language for server-side development.
- An API framework such as Express.js for Node.js or Flask for Python to enable communication between the front-end and back-end components.
- HTML, CSS, and JavaScript for creating user interfaces.
- Proficiency with at least one front-end framework to streamline the development process and enhance the user interface.
- Familiarity with relational databases such as MySQL or PostgreSQL to efficiently store and retrieve data.
- Knowledge and implementation of software development best practices, including version control, unit testing, test-driven development, continuous integration/continuous deployment, code reviews, basic Linux commands, and agile methodology to ensure high-quality and efficient development.

Essential Skills for Full Stack Developers

What You Need to Know to Succeed

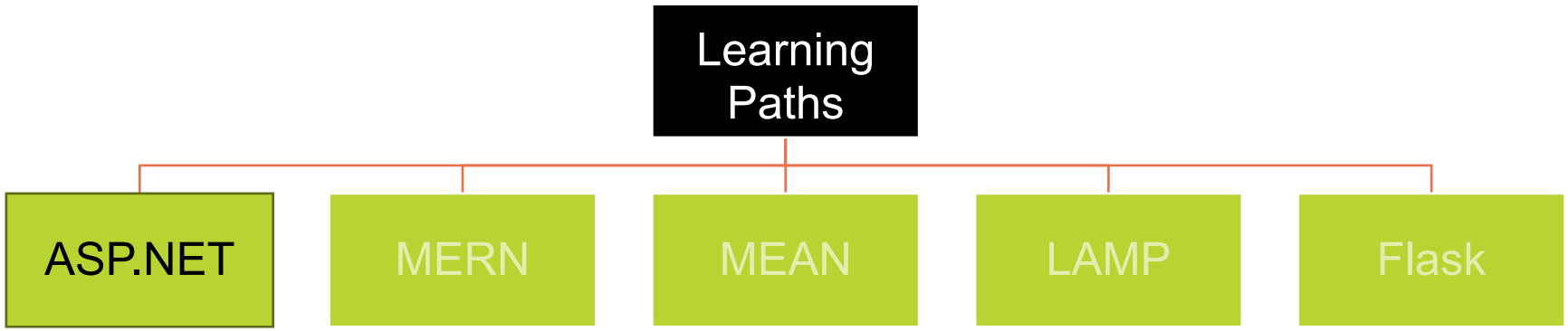


Advanced Skills

- Cloud Computing
- Virtualization
- Containerization
- GraphQL
- DevOps
- Security
- Software Development Architectures
- Domain-Specific Knowledge: IoT, Machine Learning, CRM

Recommended Learning Paths to Follow

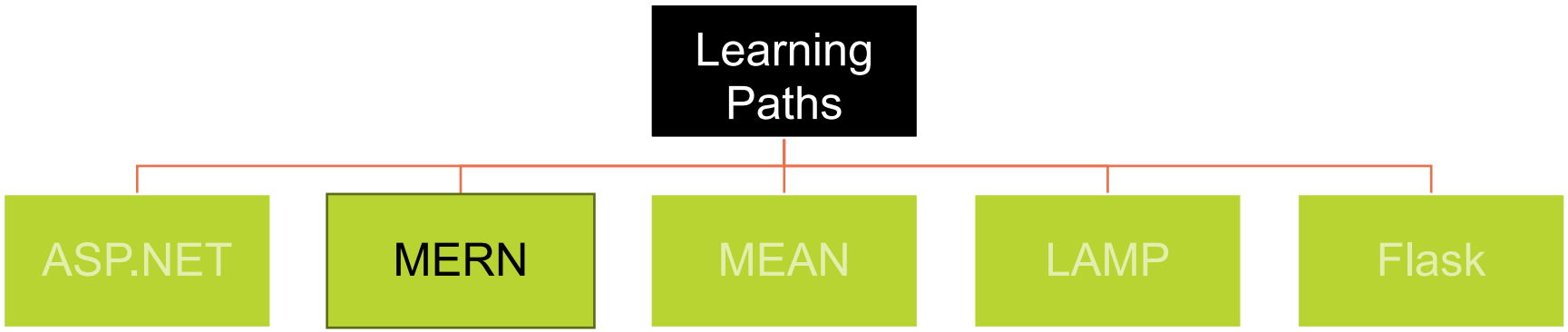
The roadmap!



1. Learn C# programming language
2. Familiarize yourself with HTML/CSS
3. Learn basic SQL for database access
4. Learn ASP.NET framework
5. Use Microsoft SQL Server as the database engine

Recommended Learning Paths to Follow

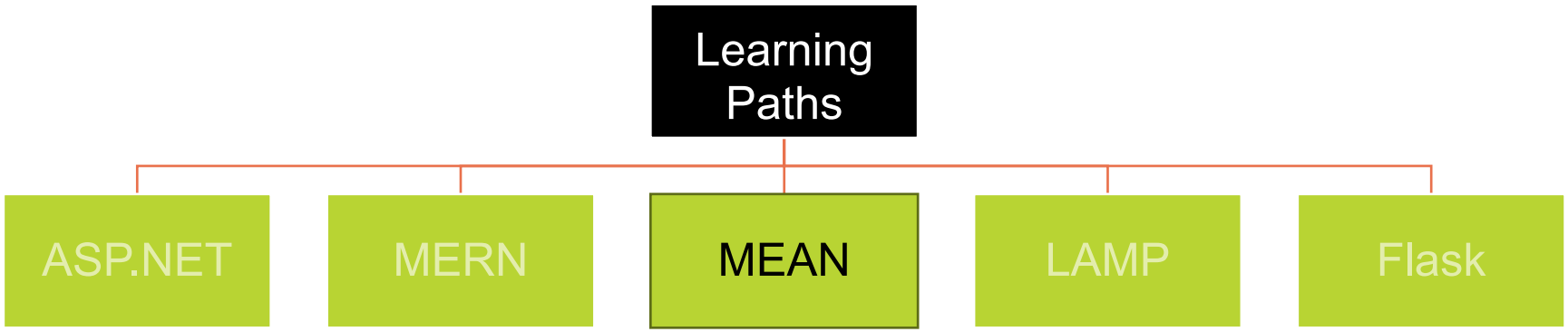
The roadmap!



1. Learn JavaScript programming language
2. Learn Node.js runtime environment
3. Learn React JavaScript library
4. Learn MongoDB NoSQL database engine

Recommended Learning Paths to Follow

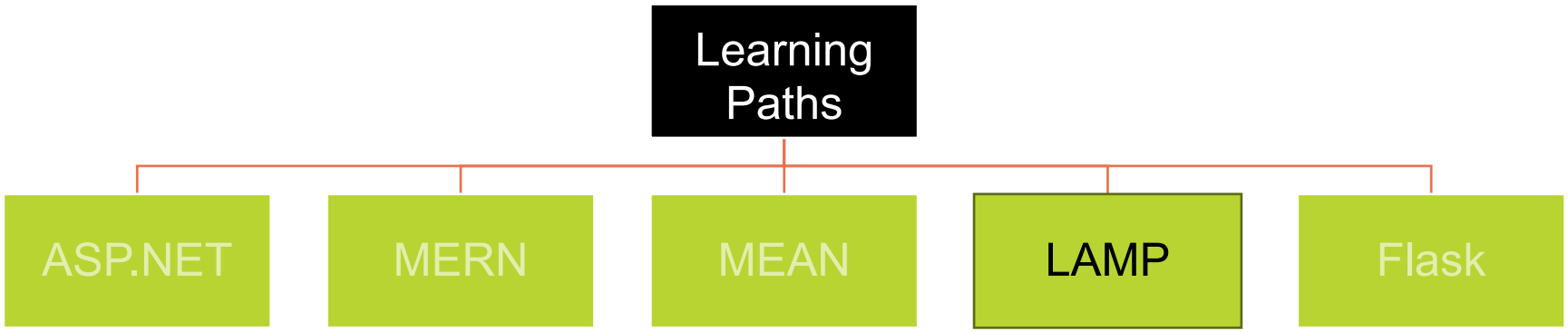
The roadmap!



1. Learn JavaScript programming language
2. Learn Node.js runtime environment
3. Learn Angular JavaScript framework
4. Learn MongoDB NoSQL database engine

Recommended Learning Paths to Follow

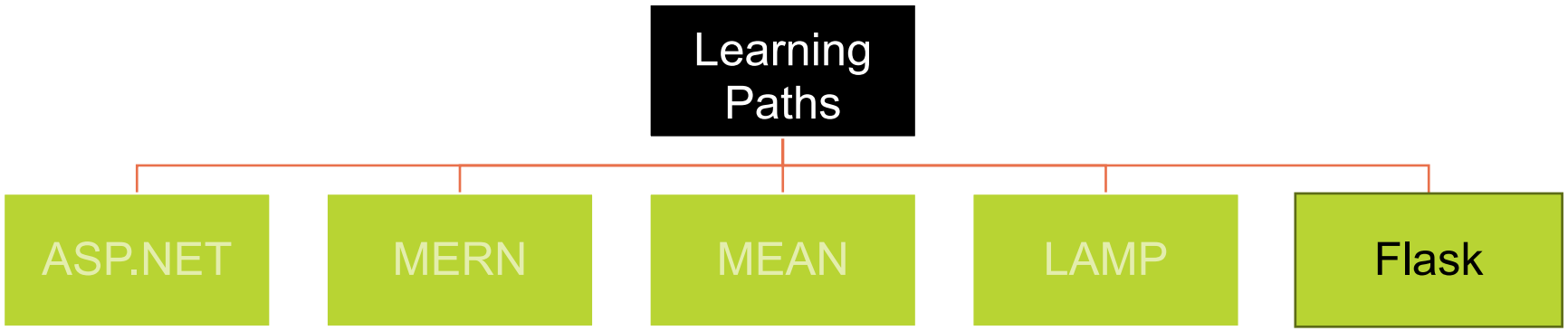
The roadmap!



1. Learn Linux operating system
2. Learn Apache web server
3. Learn MySQL database engine
4. Learn PHP scripting language

Recommended Learning Paths to Follow

The roadmap!



1. Learn Python programming language
2. Familiarize yourself with HTML/CSS
3. Learn basic SQL for database access
4. Learn Flask framework
5. Use MySQL as the database engine

How to Tailor Your Learning Path

Making it Personal

- Consider utilizing the learning resources that are available to you.
- Explore other learning platforms that offer free courses.
- Reach out to the senior members.

Thank You!

vahid.pourheidari@kyndryl.com

COMP 3010

Week 8
Dr. Sara Rouhani

Today

Consistency

Peer 2 Peer network

Election

Why it is important to have replication?

Why it is important to have replication?

Reliability and fault tolerance

Performance

What is the challenge?

What is consistency?

What is consistency?

All replicas of a data item have the same value at a given point in time.

Why it is important to have replication?

Reliability and fault tolerance

Performance

What is the challenge?

- Consistency problem due to:
  - Networks delays
  - Partial failure
  - Asynchronous communication

Consistency levels

Strong Consistency

Eventual (weak) Consistency

Causal Consistency

Consistency levels

Strong Consistency

Eventual Consistency

Causal Consistency

Think about examples! 3 minutes ...

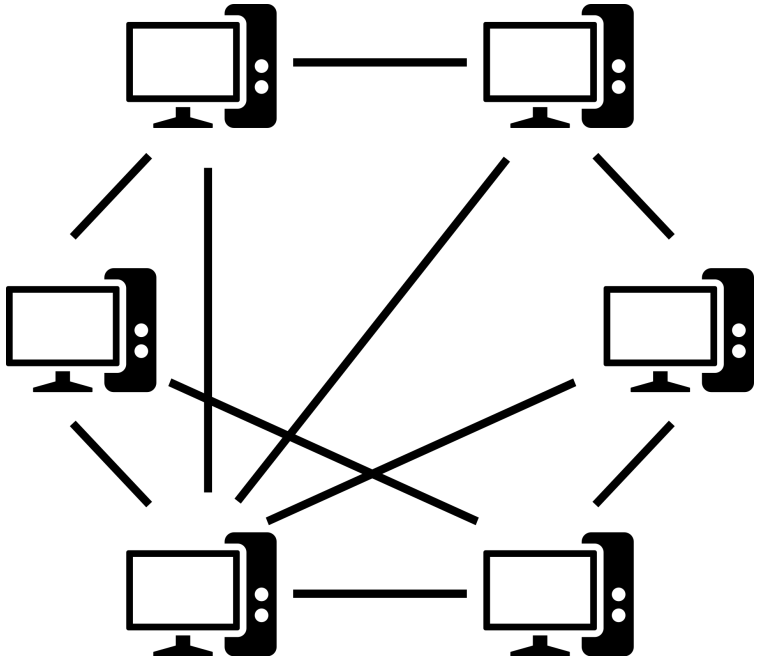
Peer 2 Peer network

A distributed application architecture that partitions tasks or workloads between peers.

Peers share their resources, such as processing power, disk storage or network bandwidth.

Peers are both suppliers (servers) and consumers (clients) of resources.

P2P does not need central coordination (servers)!!!



https://en.wikipedia.org/wiki/Peer-to-peer

What are the pros and cons of P2P systems over traditional client-server models?

Think about it! 4 Minutes…

What are the pros and cons of P2P systems over traditional client-server models?

Pros:

Resilience

Redundancy

Scalability

Load Distribution

Cons:

Redundancy

Complexity (node churn, variable peer capacity, security, ...)

Traffic Overhead

Shift toward decentralization

Peer 2 Peer network

P2P does not need central coordination (servers)!!!

So, how do the peers get coordinated?

There is no central coordinator, but there might be a temporary coordinator (leader).

How select the leader?

Through Leader Election algorithms

- Bully algorithm
- Ring algorithm
- There are more, Many versions of each, Raft, Zookeeper, ...

You want to design a leader election algorithm what factors you should consider?

You want to design a leader election algorithm what factors you should consider?

Fairness

Efficiency

Scalability

Resilience

Fairness

The algorithm should be fair in the sense that all peers have an equal chance of becoming the leader.

Efficiency

How quickly and resource-effectively the system can elect a leader

Algorithm complexity

Minimize number of messages

Amount of data exchanged

Required time for selecting the leader

Optimize utilizing resources for selecting the leader

Scalability

Scalability refers to the ability of algorithm to function effectively and efficiently as the system grows in size.

Growth in number of nodes, messages, ...

Failure handling

Resilience

The algorithm must be able to continue to elect a leader even if some of the peers in the network fail.

Failure detection

Leader Failure

Partitioning

COMP 3010

Week 9, Tuesday Oct 31th
Dr. Sara Rouhani

Today

Byzantine generals

Byzantine Fault Tolerance (BFT)

Consensus

Introduction to some consensus algorithms

Byzantine generals

Byzantine army with multiple generals.

Generals need to **communicate** to each other and agree on a common strategy.

What are the challenges?

Byzantine generals

Byzantine army with multiple generals.

Generals need to **communicate** to each other and agree on a common strategy.

Challenges:

- Some of the generals might be traitors (malicious) and could send deceptive messages.
- Messages can be lost or tampered with during transmission.

Byzantine consensus

These systems need to be able to **agree** on a common strategy, even if some of the generals are traitors.

Byzantine consensus in distributed systems

In distributed systems:

- Nodes can fail
- Messages between nodes can be delayed or lost
- Malicious nodes exist

These systems need to be able to **agree** on a common state, even if some of the nodes are faulty/malicious.

Byzantine fault tolerant (BFT)

How many total nodes are needed in a system to tolerate m faulty nodes?

Or in other words, What is the maximum number of faulty/malicious nodes that a Byzantine fault tolerant system can handle?

Byzantine fault tolerant (BFT)

How many total nodes are needed in a system to tolerate m faulty nodes?

Or in other words, What is the maximum number of faulty/malicious nodes that a Byzantine fault tolerant system can handle?

$N \geq 3M+1$

N: number of total nodes

M: number of faulty nodes

At least 3M+1 nodes to tolerate M faulty nodes in a Byzantine fault tolerant system.

Byzantine fault tolerant (BFT)

We have 3 nodes and 1 is faulty node. Can the system reach consensus?

Byzantine fault tolerant (BFT)

We have 3 nodes and 1 is faulty? This is unsolvable

We need at least 4 nodes if one of them is malicious

What if we have two malicious nodes? How many nodes/generals we need?

Improvements

Digital signatures

Consensus algorithms

Solve the Byzantine consensus problems.

Consensus algorithms are required to achieve agreement on a single data value/ state.

Consensus algorithms are essential for maintaining **consistency** across distributed systems.

Consensus algorithm properties

Safety: It essentially that nothing bad will happen, or in other words,

the behavior of the algorithm is correct.

Liveness: It ensure that eventually something good will happen. Non-faulty nodes must eventually reach consensus.

Trade-offs in Consensus Algorithms

Why we have so many consensus mechanism?

Trade-offs in Consensus Algorithms

Performance

Computation requirement

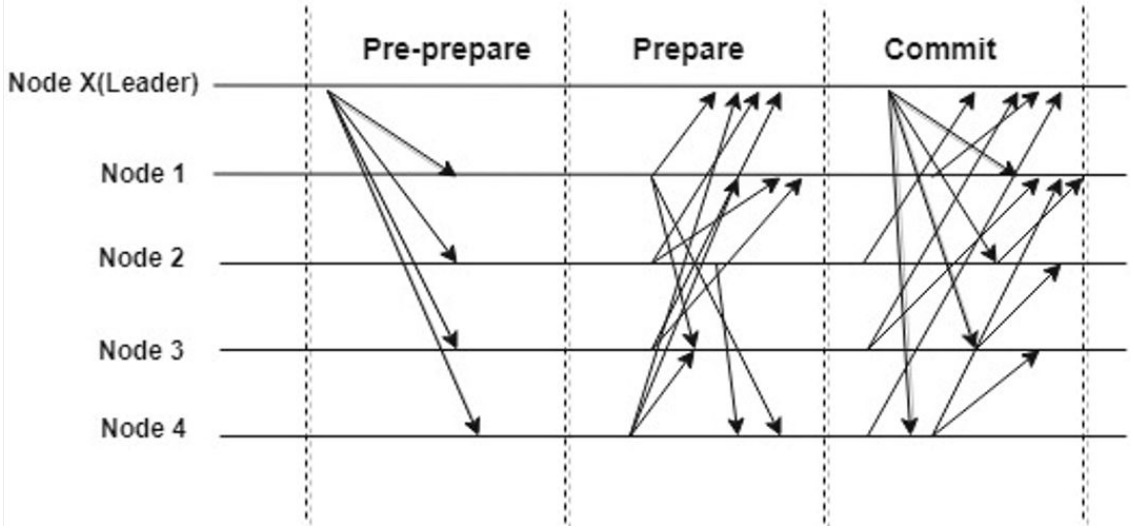
Security

Scalability

PBFT (Practical Byzantine Fault Tolerant)

The leader is temporary.

1. The leader proposes a new value/transactions to the nodes.
2. The other nodes broadcast their votes to the leader and other nodes
3. If at least two-third of nodes say yes, the new value/block will be approved.



PoW (Proof of work)

Nodes (miners) should solve a complicated mathematical puzzle to earn rewards.

After one miner solves the puzzle, the block is broadcasted to other nodes for verification.

After that block passes verification step, the block will be committed to the blockchain.

PoW is very resilient against tampering, but requires a high computation and energy power.

PoS (Proof of Stake)

It is based on the proof of ownership of the cryptocurrency.

In each round, the miner is chosen based on nodes' stake values.

There is a risk of rich gets richer.

COMP 3010

Week 9, Thursday
Dr. Sara Rouhani

Today

Menti Quiz

Blockchain introduction

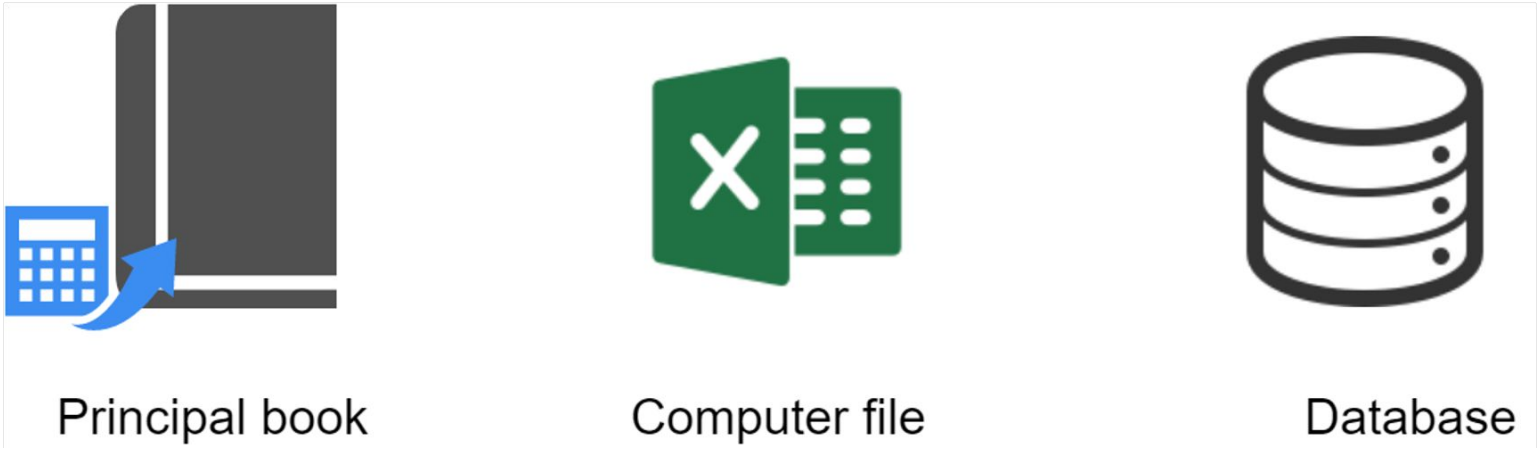
Distributed ledger

What is ledger?

Ledger

It is a place to record all transactions that happen in the system.

It is open to and trusted by all system participants.



Blockchain is a distributed ledger technology

Misconceptions:

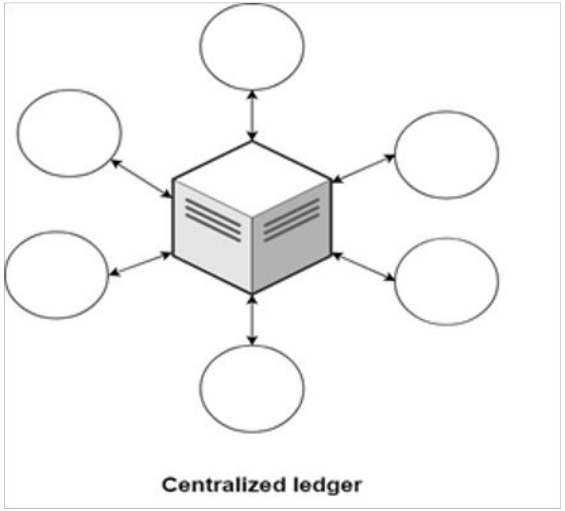
- Blockchain and distributed ledger technology are not same
- Blockchain is not the only distributed ledger technology

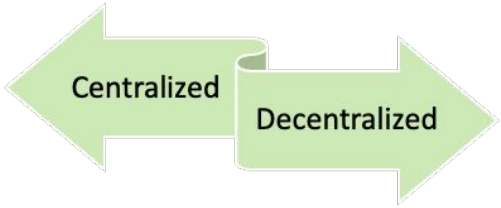
Other examples of distributed ledger technologies

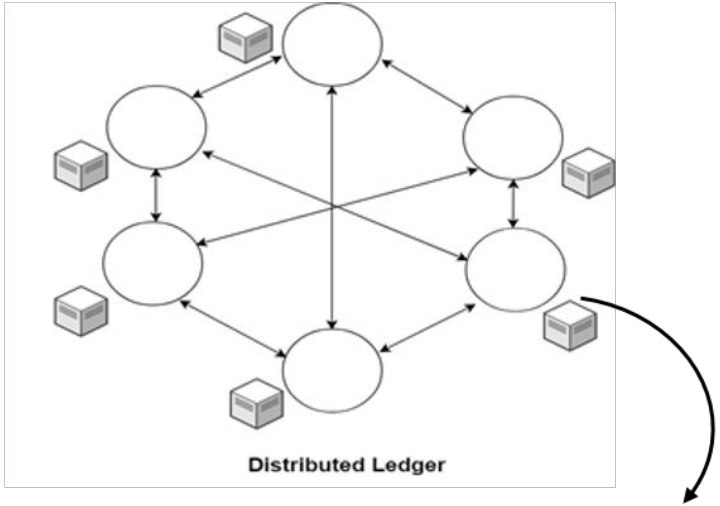
Directed Acyclic Graph (DAG), IOTA (Tangle)

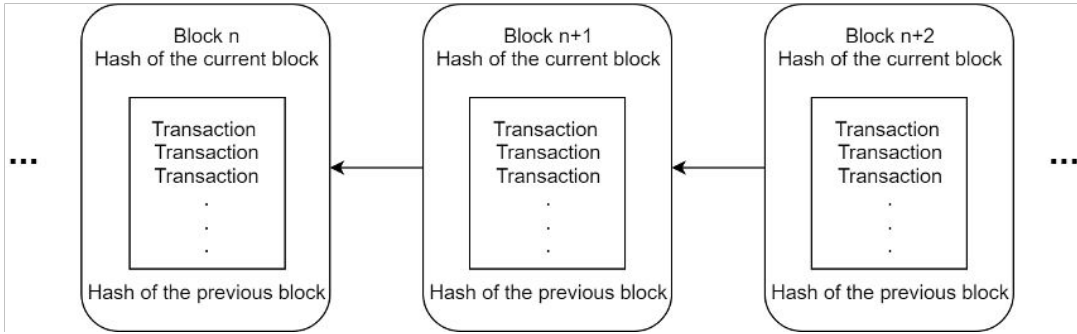
Hashgraph, Hedera platform

Centralized vs Decentralized



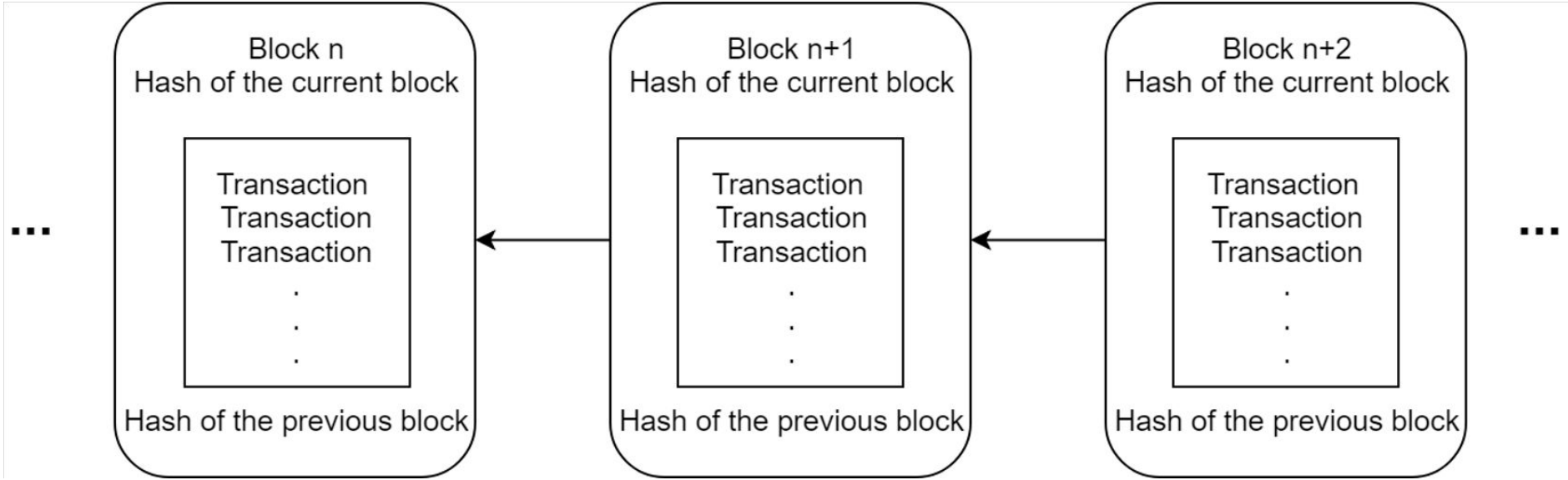






Blockchain data structure

Each block is connected to all the blocks before and after it. How?



Hash function

It is a mathematical algorithm that maps data of arbitrary size to a bit string of a fixed size (the "hash value", "hash“, or “Digest”).

SHA1, SHA2, SHA-256, SHA-512

https://passwordsgenerator.net/sha1-hash-generator/

Hash function

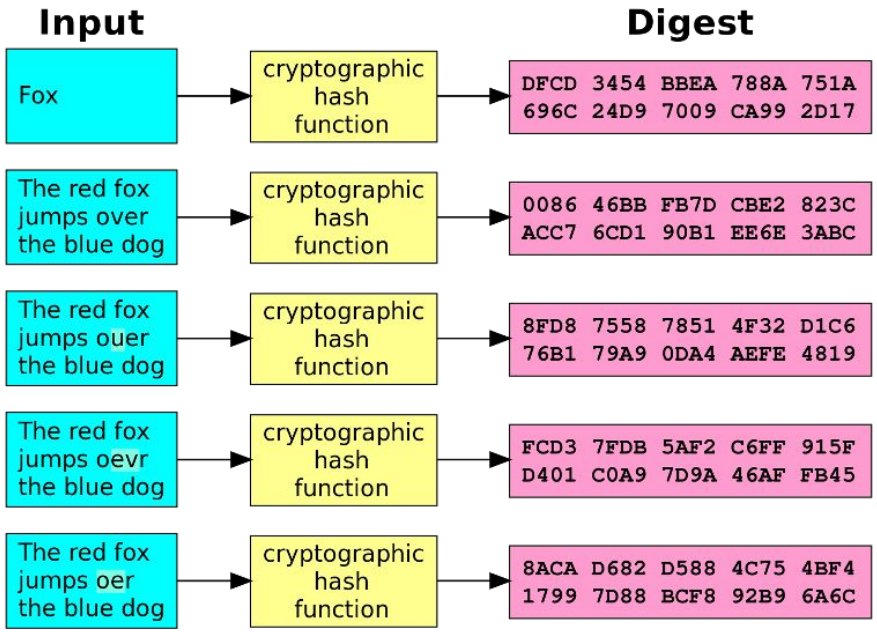
It is deterministic

It is quick to compute the hash value for any given message.

It is one-way.

It is infeasible to find two different messages with the same hash value

The hash function generates very different hash values for similar strings.



https://en.wikipedia.org/wiki/Cryptographic_hash_function

PoW and mining difficulty target

Each hash of block must start with number of zeros

It is adjusted every every 2016 blocks (~2 weeks) and decides on the number of leading zeros

Consensus and Longest chain

It states that the blockchain with the most accumulated proof-of-work (longest chain) is the valid one and the network should reach consensus based on that.

The consensus here is not based on majorities opinion.

Why network can have different lengths of chains?

Longest chain

It states that the blockchain with the most accumulated proof-of-work (longest chain) is the valid one.

Why network can have different lengths of chains?

Network latency

Forking, race condition

51% Attack

When an attacker controls more than 50% of the computing power on the network and can leads to:

Double spending

Block Withholding

Reversing transactions


BLOCKCHAIN?
THAT JUST SOUNDS LIKE A LINKED LIST WITH EXTRA STEPS
makeameme.org

Blockchain Characteristics

DECENTRALIZED

SECURE

IMMUTABLE

TAMPER RESISTANT

DATA PROVENANCE

CONSENSUS

Distributed peer to peer network

- The peers are computers that are connected directly via the internet.
- Data can be shared directly without the need of centralized server.
- The systems can also share computing power.

Advantages:

Easy to setup

Tolerance to faults attacks

Load balancing

Cost efficiency

Hard to control

Security

Blockchain blocks, linked and secured using cryptography methods.

The data stores using sophisticated math and innovative rules that makes it extremely difficult for attackers to manipulate it.

You don’t have to put your trust on single entity.

Immutable

- The ability of blockchain to record transactions permanently.
- In blockchain you can only add new transactions but not remove, modify, or reorder existing ones. (append only)

Tamper-proof

- Each transaction on a blockchain is secured with a digital signature of the creator.
- The way that data are added to the blockchain using hash values linked together makes it impossible for adversaries to tamper the data.

Provenance

As we mentioned blockchain is a chain of chronological blocks.

It provides an infrastructure to record sequence of events or history for transactions.

Main use case supply chain.

Increase customers’ knowledge of products origin and history.

Consensus

How reach agreement on the new state of the blockchain?

Who get the chance to add/append a new block to the blockchain?

Blockchain applies consensus mechanism to record only valid blocks and reaching consensus between all nodes.

Block configuration

Mining difficulty, time of block generation, latency and throughput

Size of the blocks (numbers or complexities of transactions)

Trade-off between speed of replication, inter-block time and throughput.

One of the factors that impacts the scalability of platform

Big blocks?

High block limits?

Blockchain generations

First generation:

- Cryptocurrency transactions.
- Decentralization of financial transactions.
- Proof-of-Work consensus mechanism.
- Limited scripting language (e.g., Bitcoin Script).

Second generation: Second Generation: Smart Contracts and Decentralized Applications

Third Generation: Scalability and Interoperability

COMP 3010

Week 10, Thursday Nov 9th
Dr. Sara Rouhani

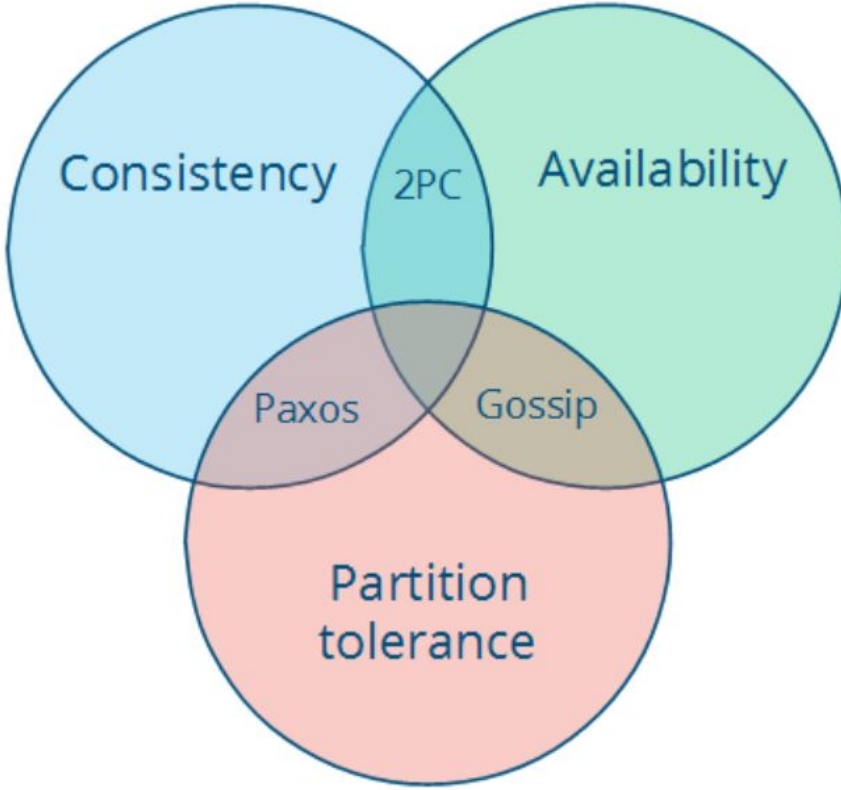
Today

CAP Theorem

CAP Theorem

Pick two out of three.

- CA: consistency + availability
- CP: consistency + partition tolerance
- AP: availability + partition tolerance



Distributed systems: for fun and profit" by Mikito Takada

https://hackernoon.com/exploring-the-cap-theorem-the-ultimate-battle-of-trade-offs-in-distributed-systems

CA and CP both offer the strong consistency model

CA cannot tolerate any node failures.

What happens if a failure happen in CA?

What is possible in terms of read and write?

What about scalability for CA systems?

CP can tolerate up to f faults given 2f+1 nodes.

Basically **majority rule**.

IRC (Internet Relay Chat)

What features of CAP theorem does IRC have?

Youtube

What features of CAP theorem does Youtube have?

What happens when partition happens?

File system

Design file systems in three ways: CP, CA, AP

Think about it 10 minutes

What design changes would you have to make?

File system, CP

How ensures consistency during partition?

What happens if network partition happens?

File system, CP

How ensures consistency during partition?

Consensus mechanism, majority rule

What happens if network partition happens?

It might not be available during these partitions.

File system, CA

Strong Consistency: Two-Phase Commit (2PC) or Three-Phase Commit (3PC).

Locking

CA cannot tolerate any node failure, it uses redundant hardware and network paths to minimize the chance of partitions. But it is never 100 % guaranteed, so what?

File system, AP

Allowing for eventual consistency

Requires strategies for resolving writing conflicts.

What about blockchain?

COMP 3010

Week 11, Nov 21
Dr.Sara Rouhani

Today

Election algorithm

Ring

DHT

Elections

Select a leader (coordinator) to coordinate actions and manage resources among a set of distributed nodes in a network.

Distribute tasks

Ensure consistency

Reliability

Why we need an election for leader?

Elections

Select a leader (coordinator) to coordinate actions and manage resources among a set of distributed nodes in a network. Algorithms such as Bully, Ring (discussed), and Raft (Last Week)

Ensure consistency

Reliability

Why we need an election for leader?

Get everyone chance to get elected

Current leader fails

System topology changes

Ring algorithm

Nodes are organized in a logical ring.

Nodes only communicate with their immediate logical neighbors.

What happens if the leader gets disconnected?

Ring algorithm

Nodes are organized in a logical ring.

Nodes only communicate with their immediate logical neighbors.

What happens if the leader gets disconnected?

1.Detect the failure of the leader

2. Initiate the election process

3. Propagate election messages

4. Complete the circuit and the new leader is selected

5. Acknowledger of the new leader

Ring data structure

It's an easy topology!

Why would we create a network as a ring?

Ring data structure

It's an easy topology!

Why would we create a network as a ring?

Simple

Predictable and deterministic traffic

Efficient resource utilization

Less network congestion

Fallacies

1. Host Drop
2. Leader Failure
3. Zero latencies
4. Topology doesn't change

More ring topology

Distributed Hash Table based on Chord algorithm

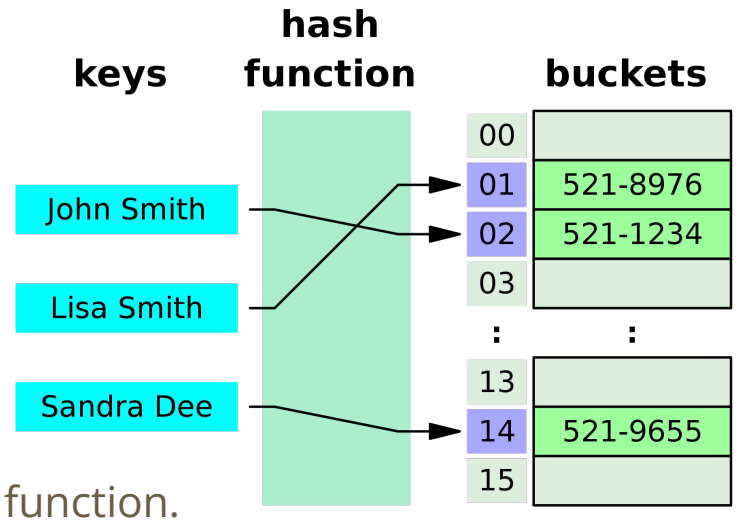
Lets see...

Hash table data structure

Efficiently locate and search an element

any data in HT is a tuple (K, V)

- K is the key that is mapped from the data by a hash function.
- V denotes the original data



https://en.wikipedia.org/wiki/Hash_table

Distributed Hash Table

Hash tables are distributed across different nodes.

Every node in a distributed hash table is responsible for a set of keys and their associated values.

Each node also has a key called ID of the node in the DHT space.

The DHT space is split into slots; each node in a DHT system maintains the data that are mapped into this node's slot.

Two primitive operations:

- put() is a function that puts data V into the DHT space
- K. get() is a function that gets the original data using a given key K.

DHT

Highly efficient: inherits the excellent properties of hash table

Decentralized

Scalable

There are different DHT algorithms; **Chord**, Kademlia, and Pastry.

Chord

Original paper: Stoica, Ion, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. "Chord: **A scalable peer-to-peer lookup service for internet applications.**" *ACM SIGCOMM computer communication review* 31, no. 4 (2001): 149-160.

Chord is a **distributed lookup protocol**
how to efficiently **locate the node** which stores a particular data item
Uses the **ring** topology
both nodes and data are mapped into this ring space by a pre-determined hash function (SHA256, MD5, …)

Nodes key

How IDs of the nodes can be generated?

Nodes key

How IDs of the nodes can be generated?

applying a hash function to unique attributes of individual nodes, such as their IPs or their IDS.

The hash function maps each node to a large, fixed-size space of IDs and provides a uniform distribution.

node N_i with its ID being i, on the clockwise ring
its previous node predecessor and define its next node as successor.

node with the maximal ID chooses the node with the minimal ID as its successor. Additionally, node with the maximal ID is the predecessor of the node with the minimal ID as its successor.

Join the network

What is bootstrap?

Join the network

What is bootstrap? The well-known entry point of the network

after the network is running should we only contact the bootstrap to join the network?

Join the network

What is bootstrap? The well-known entry point of the network

after the network is running should we only contact the bootstrap to join the network?

We can contact any running node and find our place and join. If we still know our predecessor we can re-join via that host.

What if a node goes offline?

What if a node goes offline?

Its immediate successor and predecessor in the network need to be updated.

Nodes in a DHT often periodically check the status of their successor and predecessor. For example, a node might regularly ask its successor by messages like Who is your predecessor?

Correct response

Incorrect response or no response→ rejoin the ring

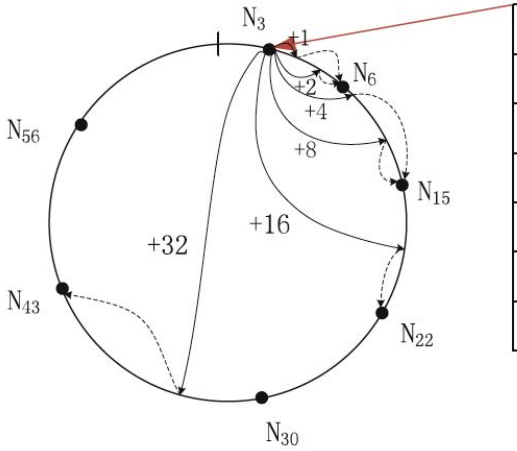
In Chord, does each node know only its immediate predecessor and successor in the ring? What problems could this cause?

Solution: *finger table*

Each node has a finger table, and each finger table maintains up to m nodes

- efficient routing
- increasing the connectivity of the graph

More specifically, the time complexity of the DHT lookup operation is reduced from O(n) to O(log n) due to increasing the connectivity from 1 to O(m).



finger table	
N_3+1	N_6
N_3+2	N_6
N_3+4	N_{15}
N_3+8	N_{15}
N_3+16	N_{22}
N_3+32	N_{43}

Distributed Hash Table Theory, Platforms and Applications by Hao Zhang, Yonggang Wen, Haiyong Xie, Nenghai Yu

Reference

Distributed Hash Table Theory, Platforms and Applications by Hao Zhang, Yonggang Wen, Haiyong Xie, Nenghai Yu

Available through UM library