DISTRIBUTED SYSTEMS ARCHITECTURE

CS435 Distributed Systems

Basit Qureshi PhD, FHEA, SMIEEE, MACM
https://www.drbasit.org/

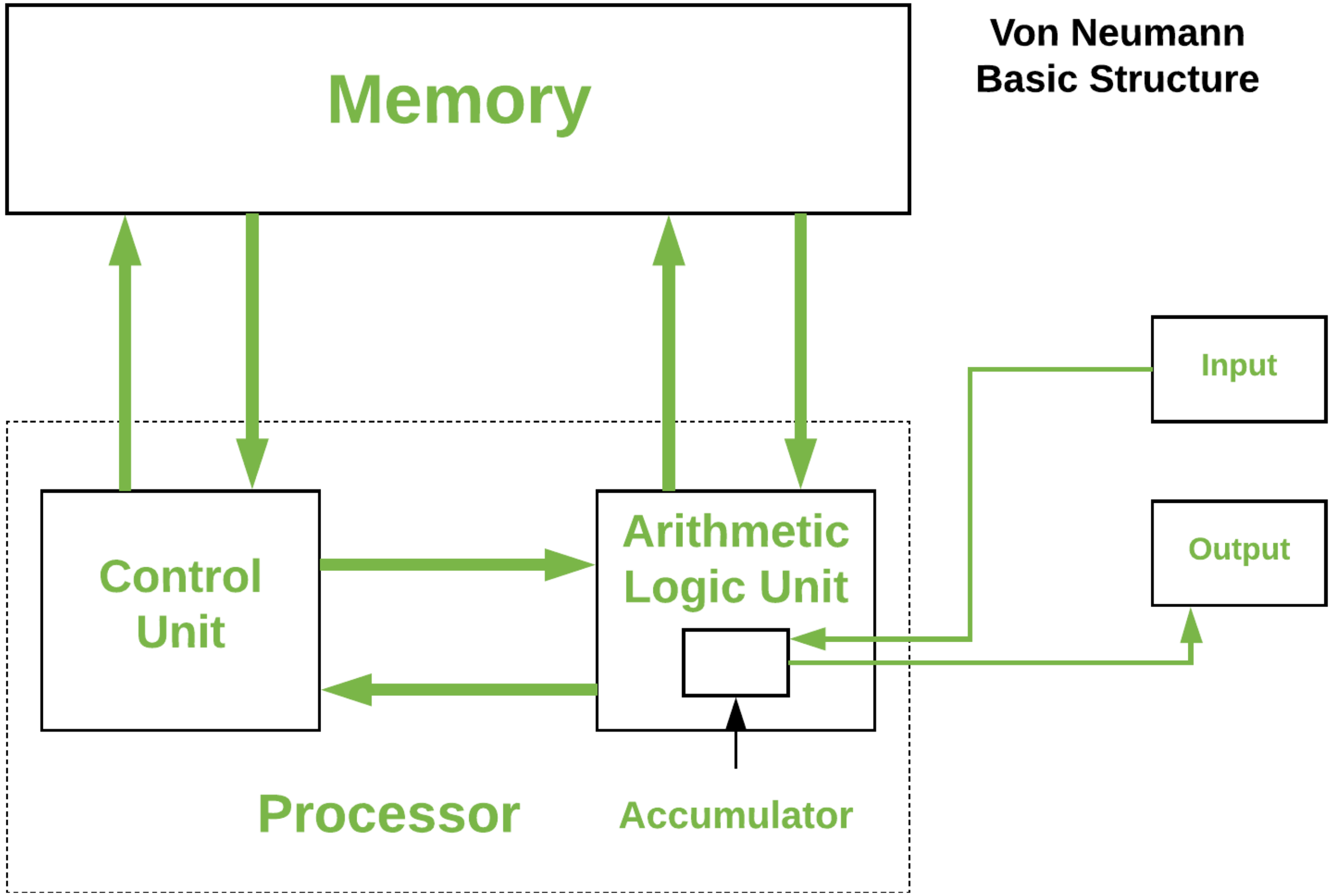
جامعة الامير سلطان
PRINCE SULTAN UNIVERSITY

TOPICS

- Operating Systems, a quick review
- Distributed Systems Themes
- Dist. Sys. Challenges
- Dist. Sys. Architecture
  - Software Architecture
  - Systems Architecture
    - Client-server arch.
    - P2P arch.
    - Hybrid approach
- Distributed Services

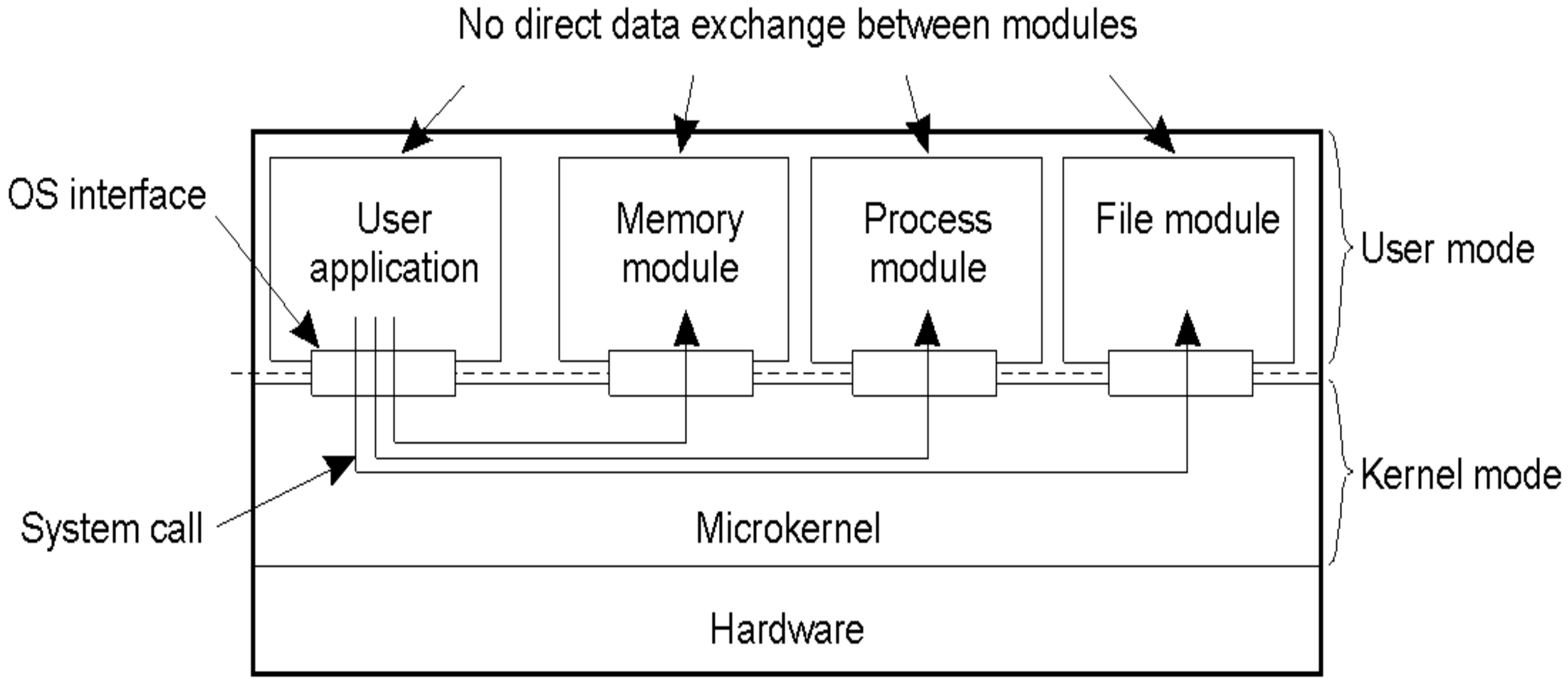
OPERATING SYSTEMS, A QUICK REVIEW

- Computer Organization



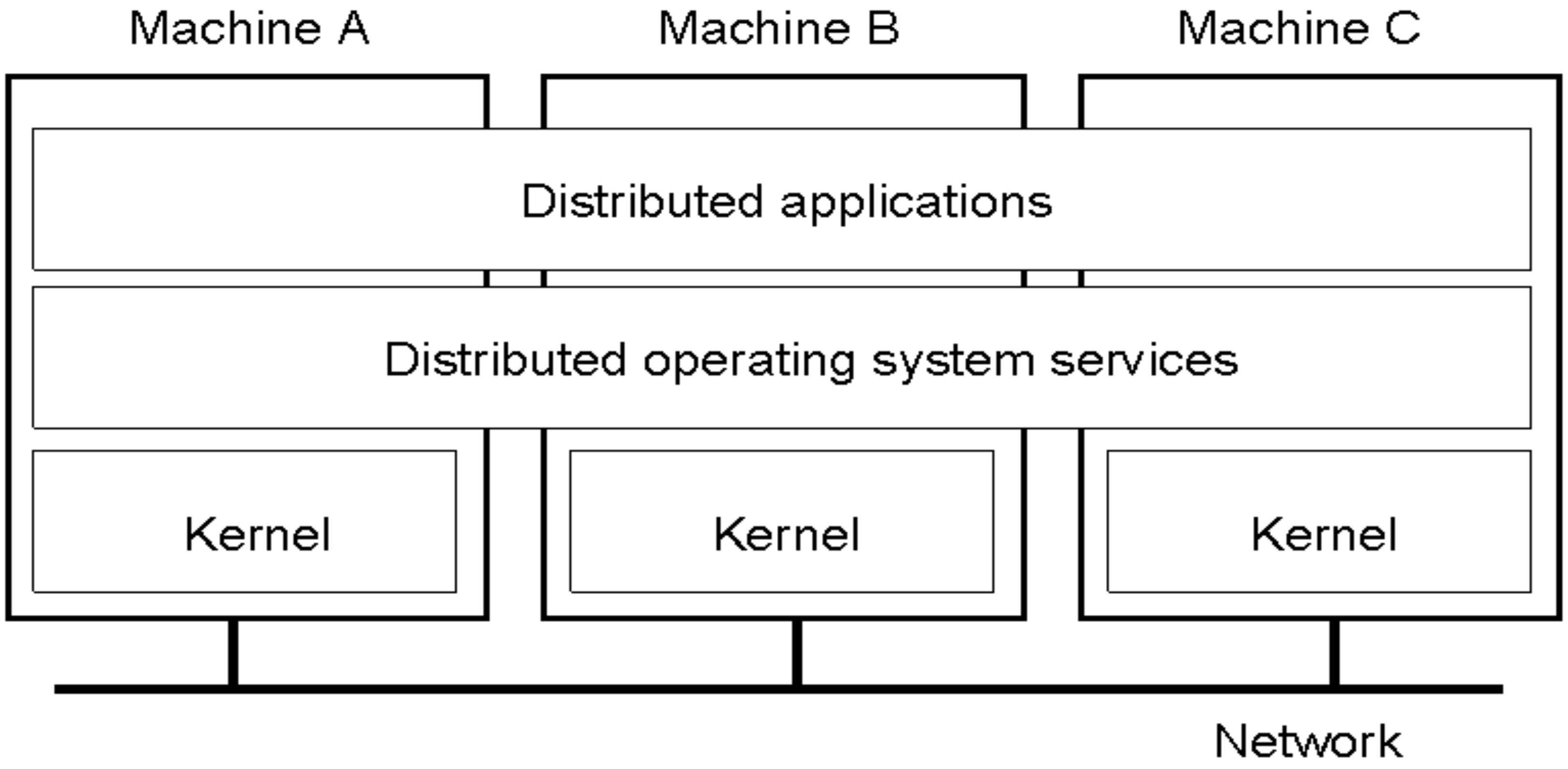
OPERATING SYSTEMS, A QUICK REVIEW

- Uni-Computer Operating Systems
  - Application, Memory, Processor, File-system resources, all on one machine



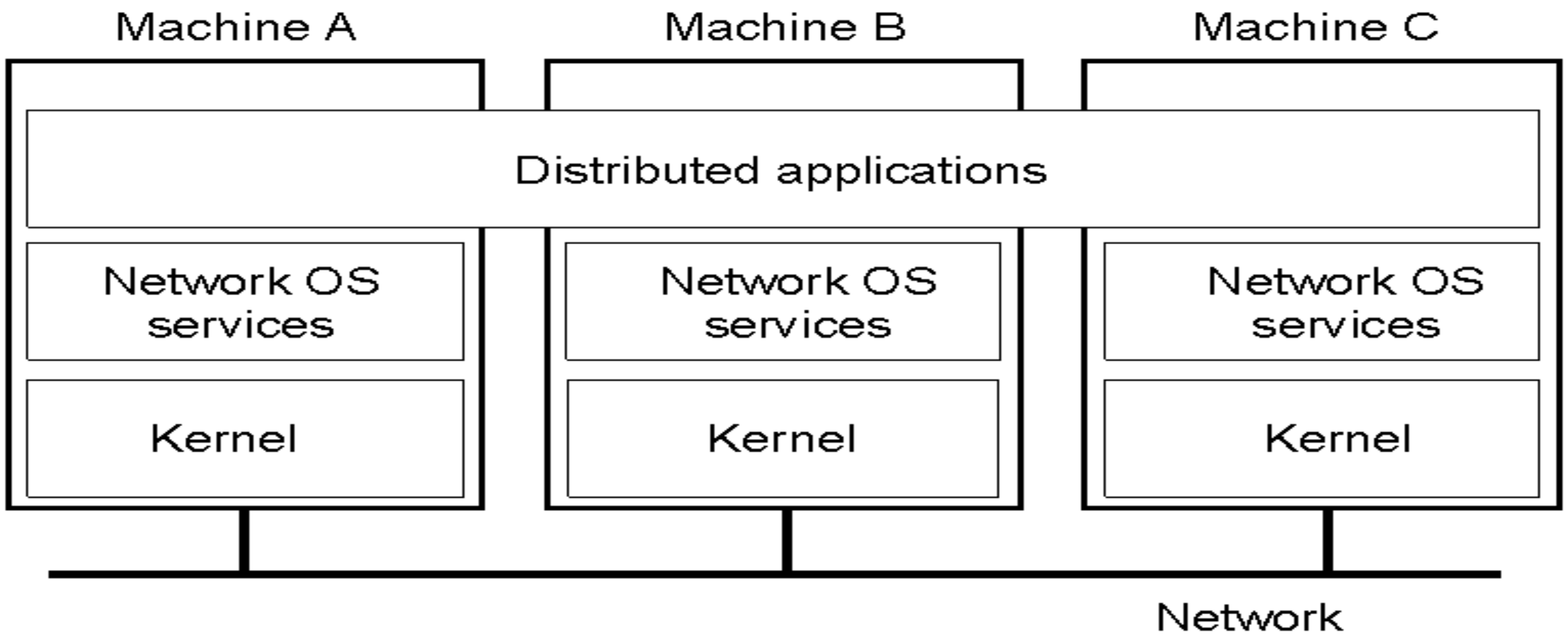
OPERATING SYSTEMS, A QUICK REVIEW

- Multi-Computer Operating System
  - All computers run using the same OS.
  - Memory shared between processors.
  - Dist. Applications run sharing Memory and CPU resources



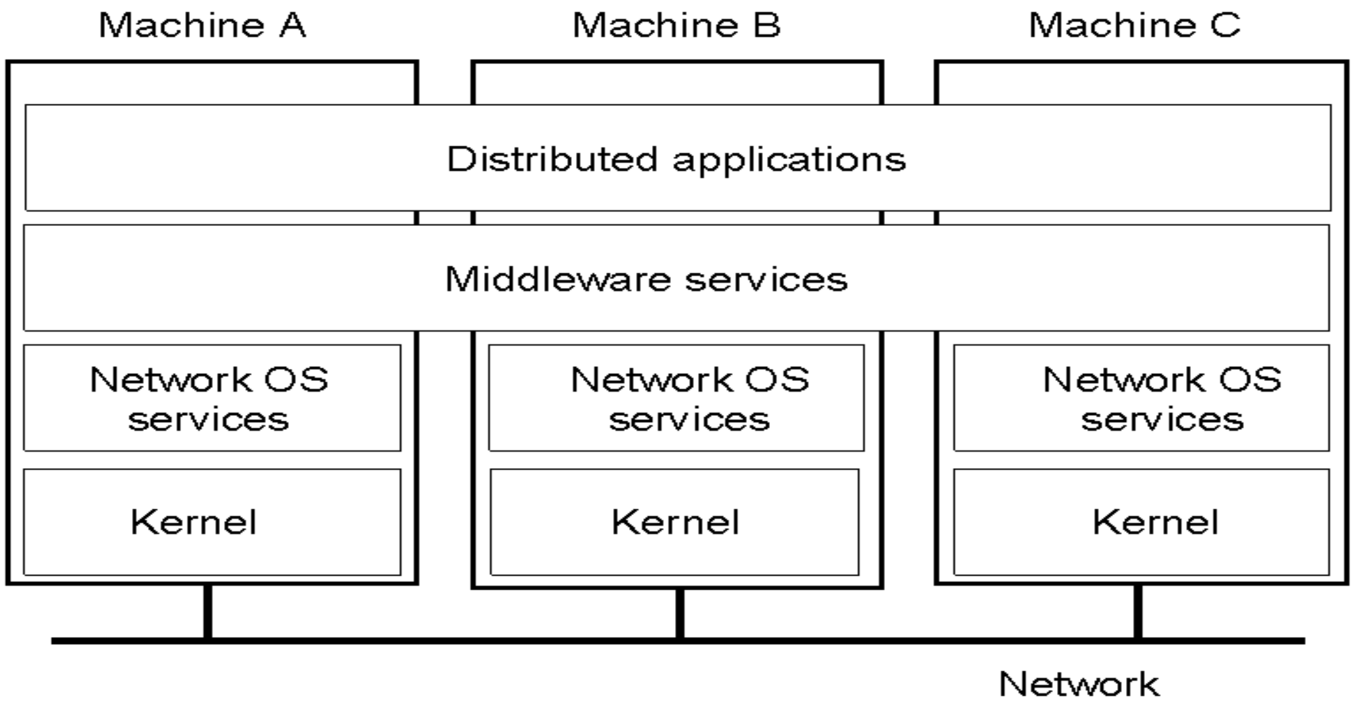
OPERATING SYSTEMS, A QUICK REVIEW

- Network Operating Systems
  - Network File system mounting on individual machines.
  - Resources accessible via network.
  - Hard to maintain a consistent view.
  - Relatively primitive set of services provided (e.g. Printers)
  - Configuration overhead/complexity



OPERATING SYSTEMS, A QUICK REVIEW

- Middleware-based Operating Systems
  - Middleware provides a set of services and communication protocols
  - Abstracts the complexities of distributed computing, making it easier for developers to design and implement distributed applications. E.g. Socket APIs



OPERATING SYSTEMS, A QUICK REVIEW

- Comparing Operating Systems

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

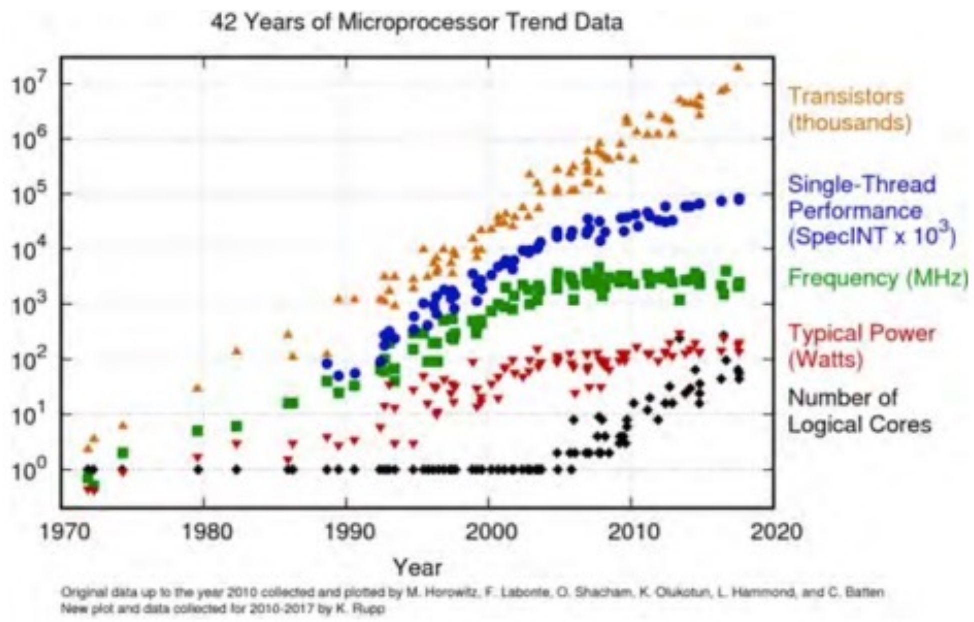
DISTRIBUTED SYSTEM THEMES

DISTRIBUTED SYSTEM THEMES

- Distributed Systems are a collection of independent computers that appears as a single system to the user(s)
  - Independent = autonomous, self-contained
  - Single system = user not aware of distribution
- GOALS
  - Scaling
  - Collaboration
  - Latency
  - Acessibility
  - Availability
  - Transparency

1. SCALING

- Vertical Scaling (Powerful systems)
  - Increases in processor performance have not been keeping up with Moore's Law since around 2005.
  - Adding more processor cores helped improve performance; but need to write multi-threaded programs
    - Intel Xeon 8490h 1.90GHz~3.50GHz 60Core/120Thread Processor (15000 USD)
    - Apple M3 Ultra 32-core CPU/ 80 Core GPU
    - Nvidia Geforce RTX 4090 Ti 18,432 CUDA cores



- Horizontal Scaling
  - Distributed load across more systems
    - Pixar Movie Rendering: 2000 machines with 24000+ cores.
    - Google: A single Google query uses 1,000 computers in 0.2 seconds to retrieve an answer

2. COLLABORATION

- Collaborate
  - Make content
  - Social connectivity
  - E-Commerce
  - News & media

YouTube

WhatsApp

NETFLIX

X

Microsoft Teams

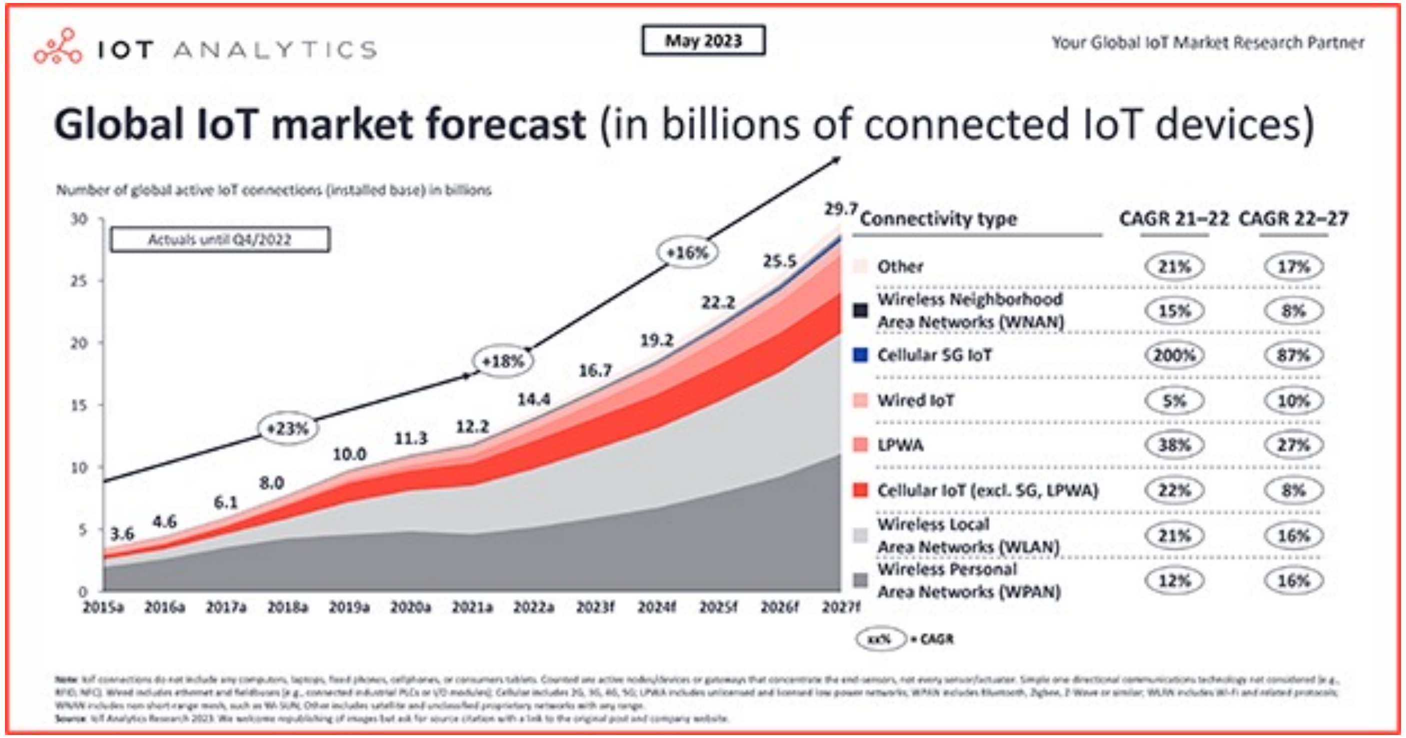
Spotify

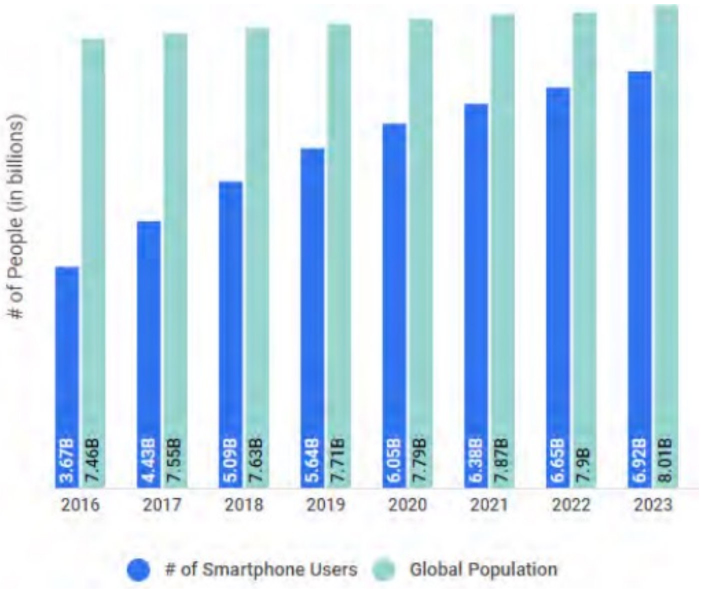
3. LATENCY

- Caching
  - Keep the data close to where it is needed
- Replication
  - Make multiple copies
- *Caching vs. replication*
  - Caching: temporary copies of frequently accessed data closer to where it's needed
  - Replication: multiple copies of data for increased fault tolerance

4. ACCESSIBILITY

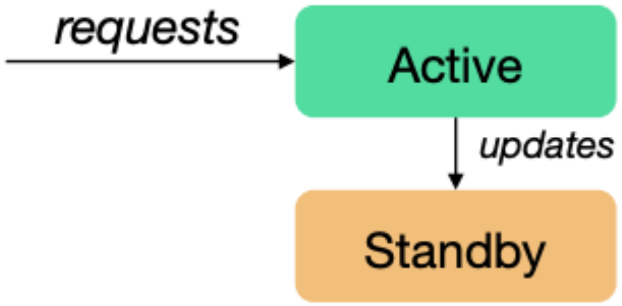
- Distributed Systems are accessible through Systems, IoT devices, Smart-phones etc.
- IoT = Internet of Things
  - 2023: 16.7 Billion devices
- Smart-Phones
  - 2023: 6.2 Billion devices

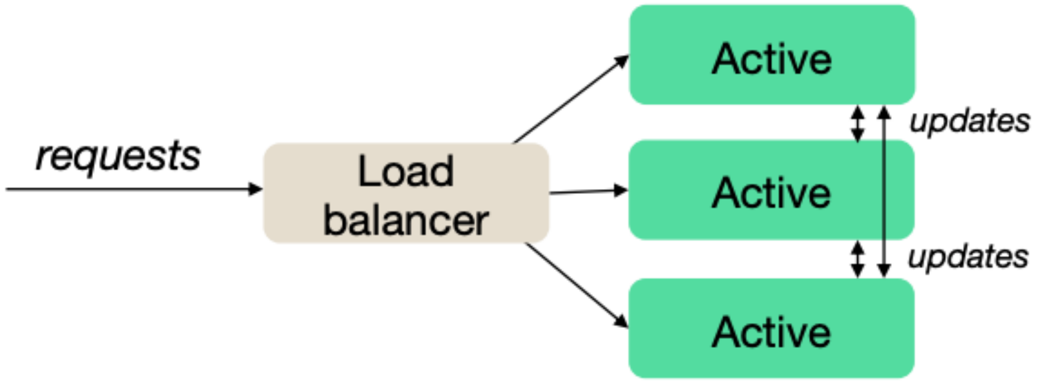




5. AVAILABILITY

- System Components Fail
  - Computers, processes, disks, memory, data centers etc
  - Replicas can take over
- Fault tolerance
  - Identify & recover from component failures
- Recoverability
  - Software can restart and function – May involve restoring state





6. TRANSPARENCY

- High level: hide distribution from users
- Low level: hide distribution from software
  - Location transparency Users don’t care where resources are
  - Migration transparency Resources move at will
  - Replication transparency Users cannot tell whether there are copies of resources
  - Concurrency transparency Users share resources transparently
  - Parallelism transparency Operations take place in parallel without user’s knowledge
  - Failure transparency Lower-level software works around any failures – things just work

DISTRIBUTED SYSTEM CHALLENGES

DISTRIBUTED SYSTEM CHALLENGES

- Concurrency
- Latency
- Partial Failure
- Security

DISTRIBUTED SYSTEM CHALLENGES

- Concurrency
  - Lots of requests may occur at the same time
  - Need to deal with concurrent requests
  - Need to ensure consistency of all data
  - Understand critical sections & mutual exclusion
    - Beware: mutual exclusion (locking) can **affect performance**
  - Caching and replication costs
    - Complex; synchronization, message-delivery, check-sums etc

DISTRIBUTED SYSTEM CHALLENGES

- Latency
  - Network delays
    - Synchronous: Use time-stamps to determine time to respond
    - Partial synchronous: Protocols operate correctly only if all messages are received within some time.
    - Asynchronous:
      - Messages may take an unpredictable amount of time
      - Lost or delayed?
      - Re-transmission -> duplicate messages
      - Mess up perception of time
      - Message arrive in different order
  - Caching: Keep data close to where it's processed to maximize efficiency
    - Memory vs. disk
    - Local disk vs. remote server
    - Remote memory vs. remote disk

DISTRIBUTED SYSTEM CHALLENGES

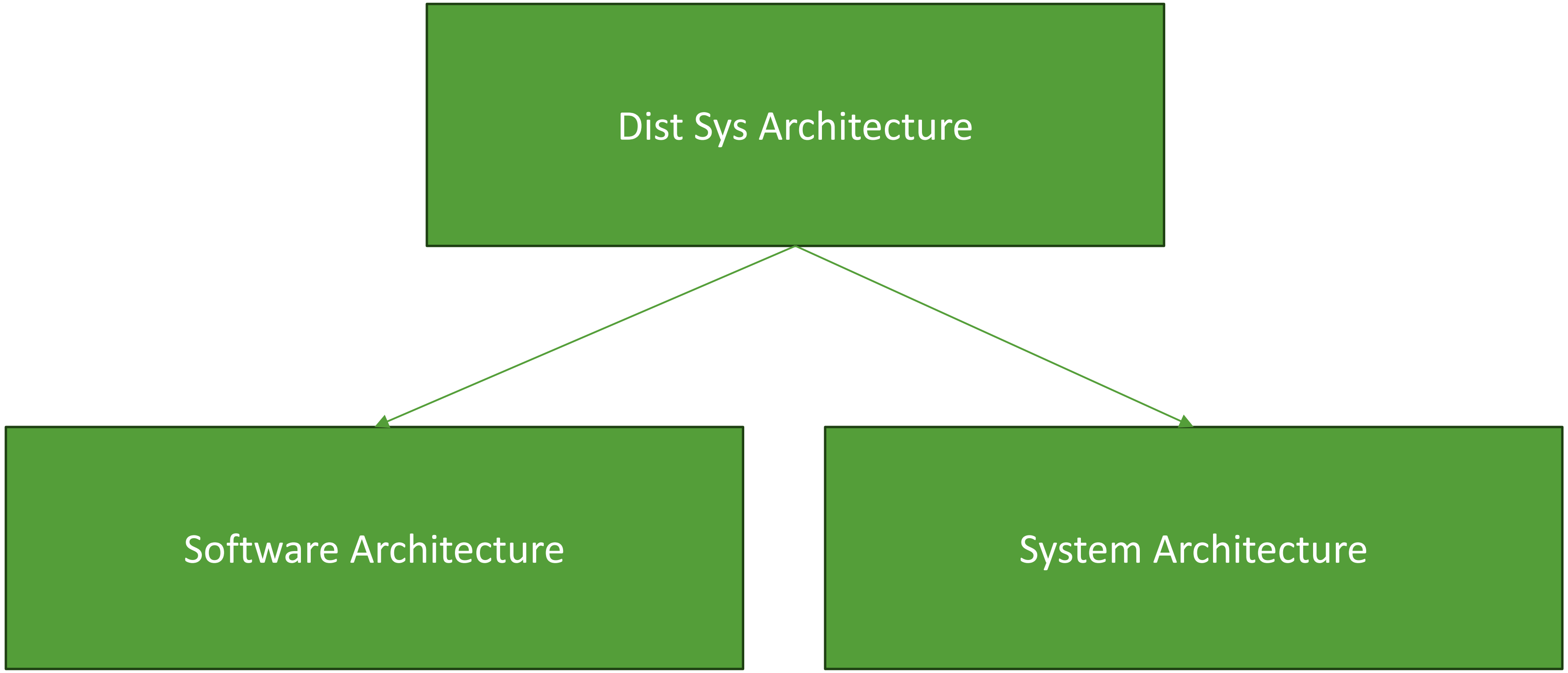
- Partial Failure
  - In local systems, failure is usually total (all-or-nothing)
  - In distributed systems, we get partial failure
    - A component can fail while others continue to work
    - Failure of a network link is indistinguishable from a remote server failure
    - Send a request but don't get a response ⇒ what happened?
  - No global state
    - There is no global state that can be examined to determine errors
    - There is no agent that can determine which components failed and inform everyone else
  - Need to ensure the state of the entire system is consistent after a failure

DISTRIBUTED SYSTEM CHALLENGES

- Security
  - Traditionally managed by operating systems
    - Users authenticate themselves to the system
    - Each user has a unique user ID (UID)
    - Access permissions = f(UID)
  - Now applications must take responsibility for
    - Identification, Authentication, Access control, Encryption, tamper detection, Audit trail
  - The environment
    - Public networks, remotely-managed services, 3rd party services
    - Trust: do you trust how the 3rd party services are written & managed?
  - Some issues:
    - Malicious interference, bad user input, impersonation of users & services
    - Protocol attacks, input validation attacks, time-based attacks, replay attacks
  - Rely on cryptography (hashes, cryptography) for identity management, authentication, encryption, tamper detection … and also rely on good defensive programming!

ARCHITECTURE

DISTRIBUTED SYSTEMS ARCHITECTURE

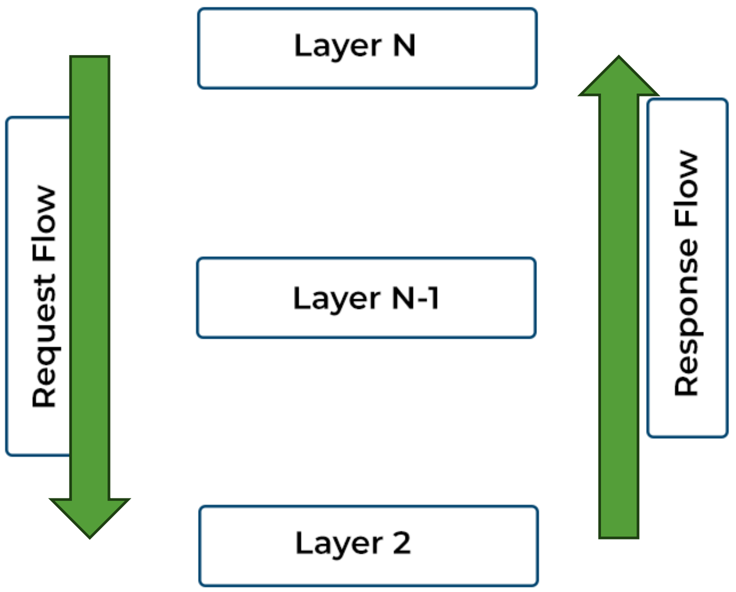


SOFTWARE ARCHITECTURE

- “The logical organization of software components and their interaction with other structures”
- Focuses entirely on “components”. Example: website front-end.
- Four common types
  - Layered
  - Object based
  - Data centric
  - Event based

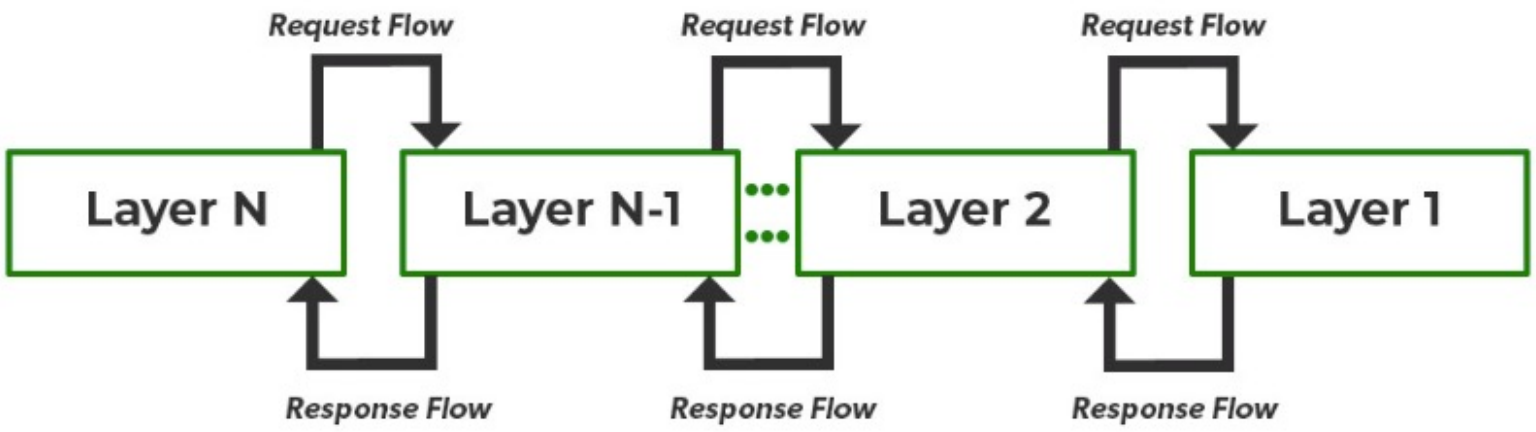
SOFTWARE ARCHITECTURE

- Layered architecture
  - Provides a modular approach to software
  - Software components organized as layers
  - Information flows through layers. E.g. a request goes from the top down, and the response goes from the bottom up.
  - Any layer can not directly communicate with another layer
  - No intermediate layer can be skipped!



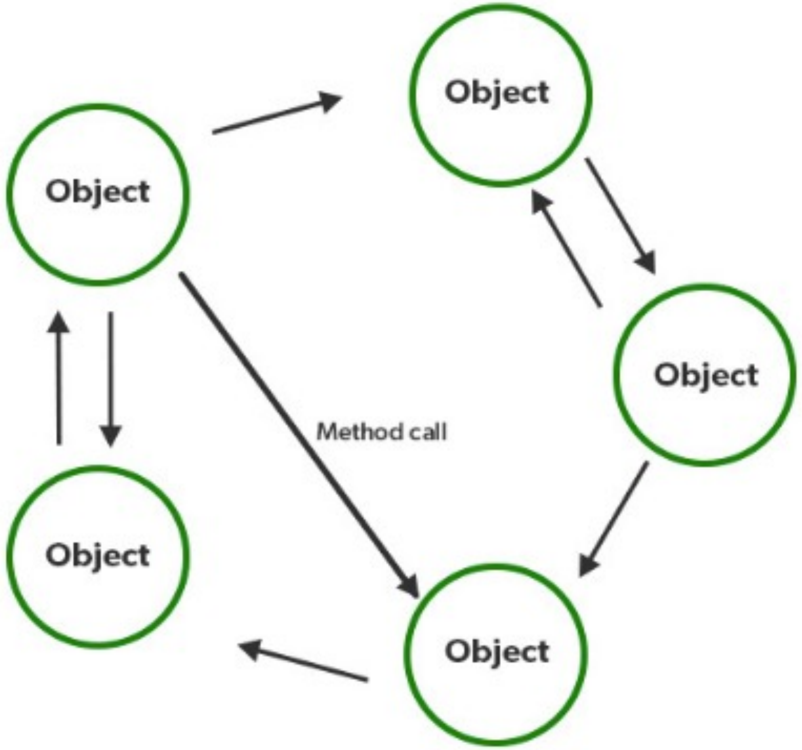
SOFTWARE ARCHITECTURE

- Layered architecture
  - Advantage:
    - 1. Each layer can be modified independently without affecting the whole system.
    - 2. Calls always follow a predetermined path and that each layer is simple to replace or modify without affecting the architecture as a whole.
  - This type of architecture is used in Open System Interconnection (OSI) model.
  - In some cases, layered architecture is in cross-layer coordination. In a cross-layer, any adjacent layer can be skipped until it fulfils the request and provides better performance results.



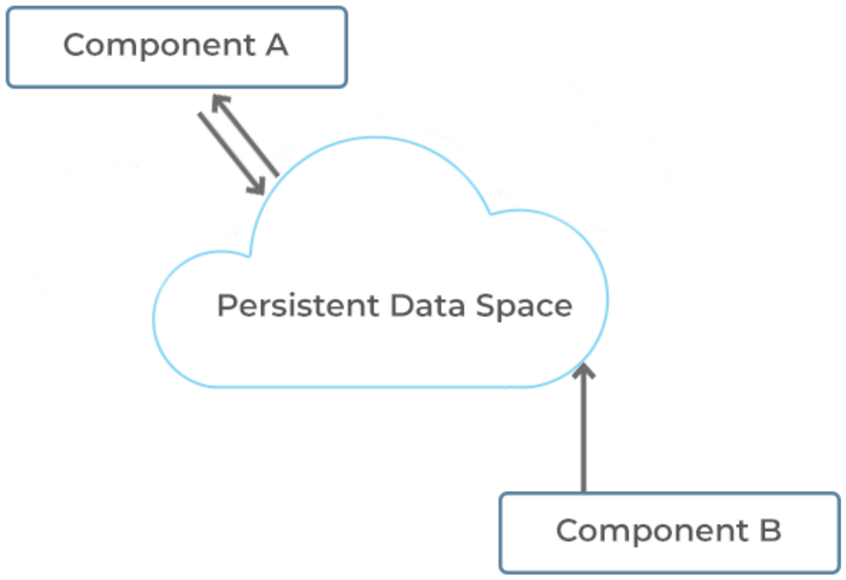
SOFTWARE ARCHITECTURE

- Object based architecture
  - Components are treated as objects which convey information to each other.
  - Contains an arrangement of loosely coupled objects.
  - Objects can interact with each other through method calls e.g. Remote Procedure Call (RPC) mechanism or Remote Method Invocation (RMI) mechanism.
  - Examples: REST API Calls, Web Services, Java RMI



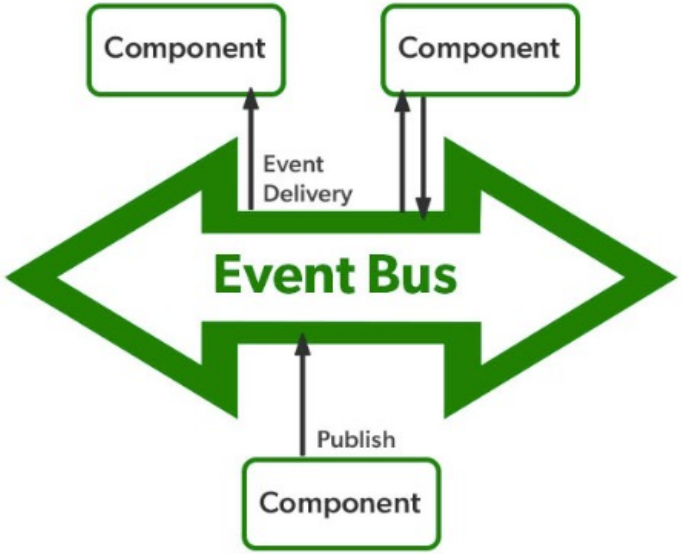
SOFTWARE ARCHITECTURE

- Data centric architecture
  - Works on a central data repository, either actively or passively
  - All the components are connected to this data repository.
  - Producer-consumer communication model:
    - Producer produces items to the common data repository
    - Consumer (individual) can request data from the common data repository
  - Example: Web-based E-commerce systems



SOFTWARE ARCHITECTURE

- Event based architecture
  - Similar to Data centered architecture (replaces Data with events)
  - Events are present at the center in the Event bus and delivered to the required component whenever needed
  - When an event occurs, the system, as well as the receiver, get notified. Data, URLs etc are transmitted through events.
  - Components are loosely coupled. i.e., it's easy to add, remove, and modify components.
  - Example: Enterprise services buses; akka.io



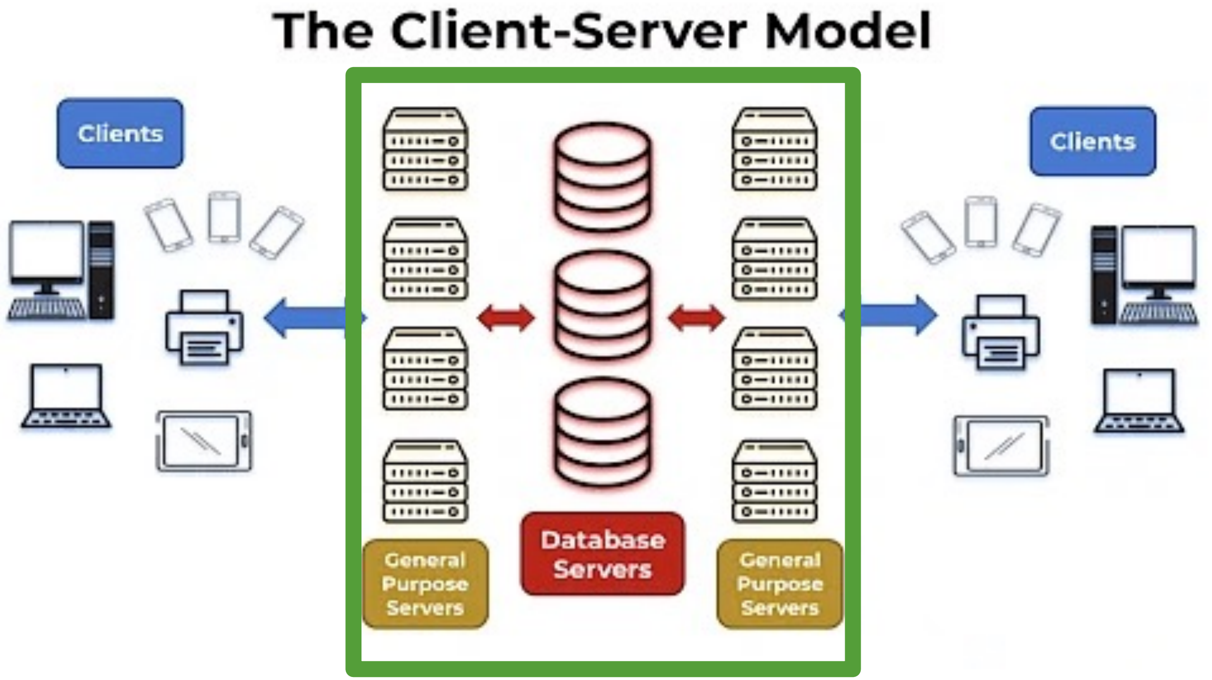
SYSTEM ARCHITECTURE

- “The placement of components of a distributed system across multiple machines”
- Three possible types
  - Centralized: Client-Server
  - De-centralized: Peer-to-Peer
  - Hybrid

SYSTEM ARCHITECTURE: CLIENT-SERVER MODEL

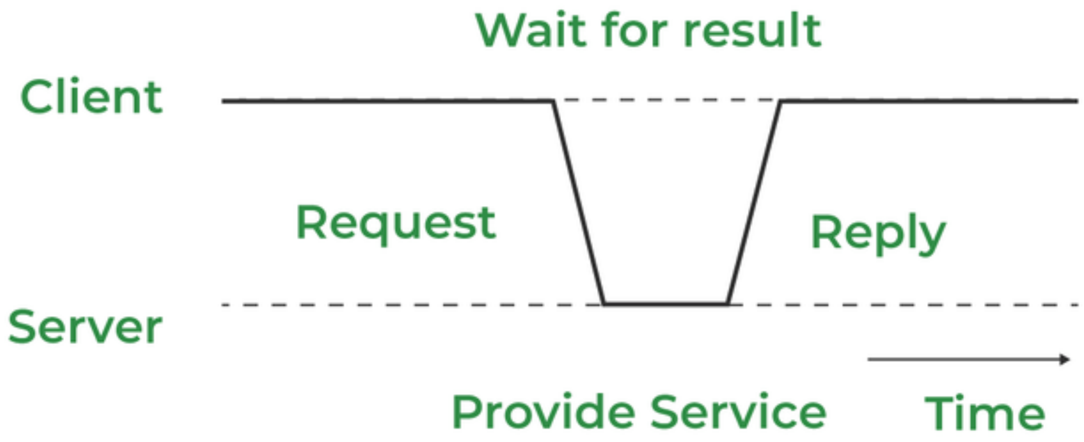
SYSTEM ARCHITECTURE: CLIENT-SERVER MODEL

- Centralized / Client-Server model
- Every node is connected to a central coordination system
  - Client – This is the first process that issues a request to the second process i.e. the server.
  - Server – This is the second process that receives the request, carries it out, and sends a reply to the client.



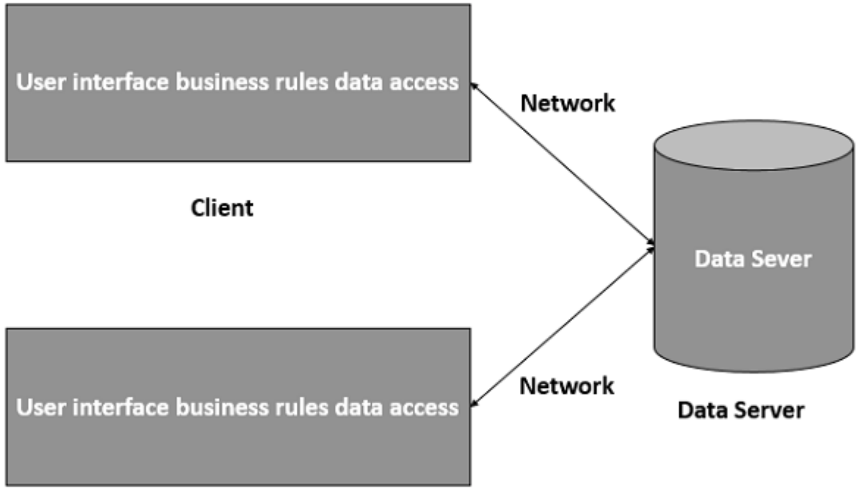
SYSTEM ARCHITECTURE: CLIENT-SERVER MODEL

- Client-server interaction/request-reply behavior.
  - Server: a process that implements a service (exp: file system service, database service).
  - Client: a process that requests a service from a server
- Communication between a client and a server can be:
  - Connectionless protocol [if reliable connection available].
  - Connection oriented protocol [otherwise, e.g. TCP/IP].



SYSTEM ARCHITECTURE: CLIENT-SERVER MODEL

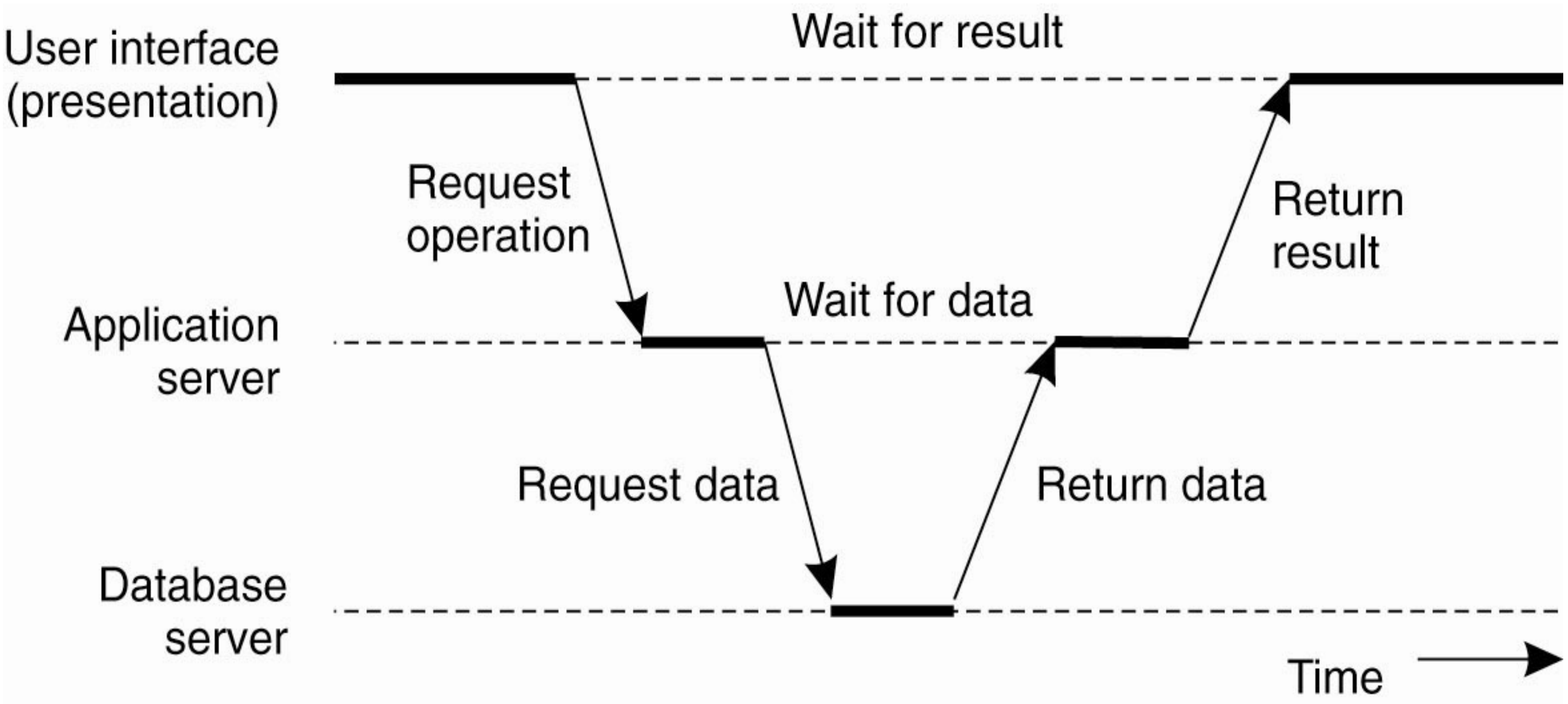
- 2-tier Client server architecture
  - The servers need not know about clients
  - The clients must know the identity of servers
  - Mapping of processors to processes is not necessarily 1
- Thin Client Model
  - Server: Application processing and data management
  - Client: Provide interface of the application
- Thick Client Model
  - Server: Data management only
  - Client: Complex data processing and interface



2 – Tier Client Server Architecture

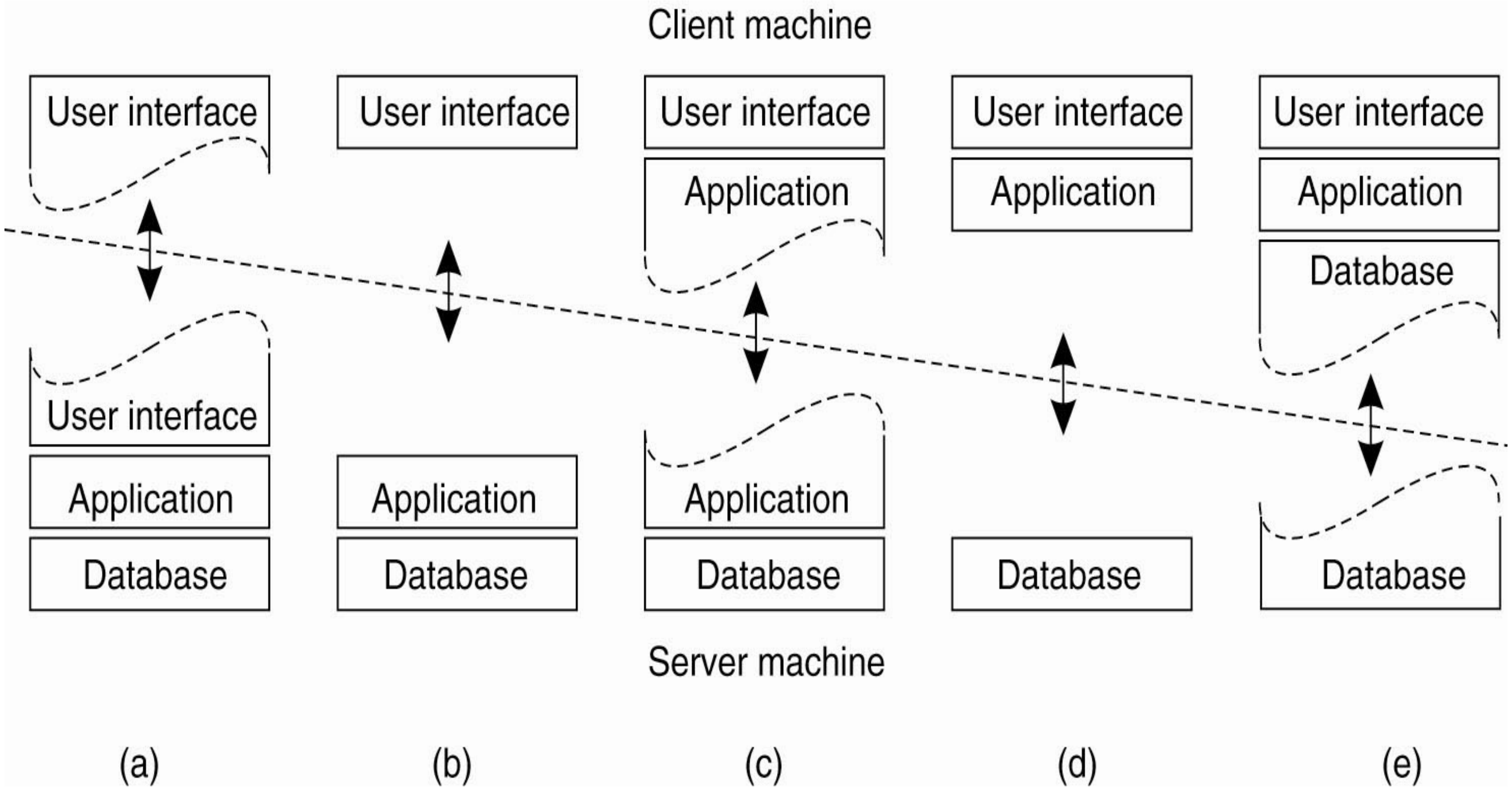
SYSTEM ARCHITECTURE: CLIENT-SERVER MODEL

- n-tier Client server architecture
  - Multi tier allows separate tier for a functionality of an application
  - 3-tier is common with Web/App-server, DB-server and Client-browser



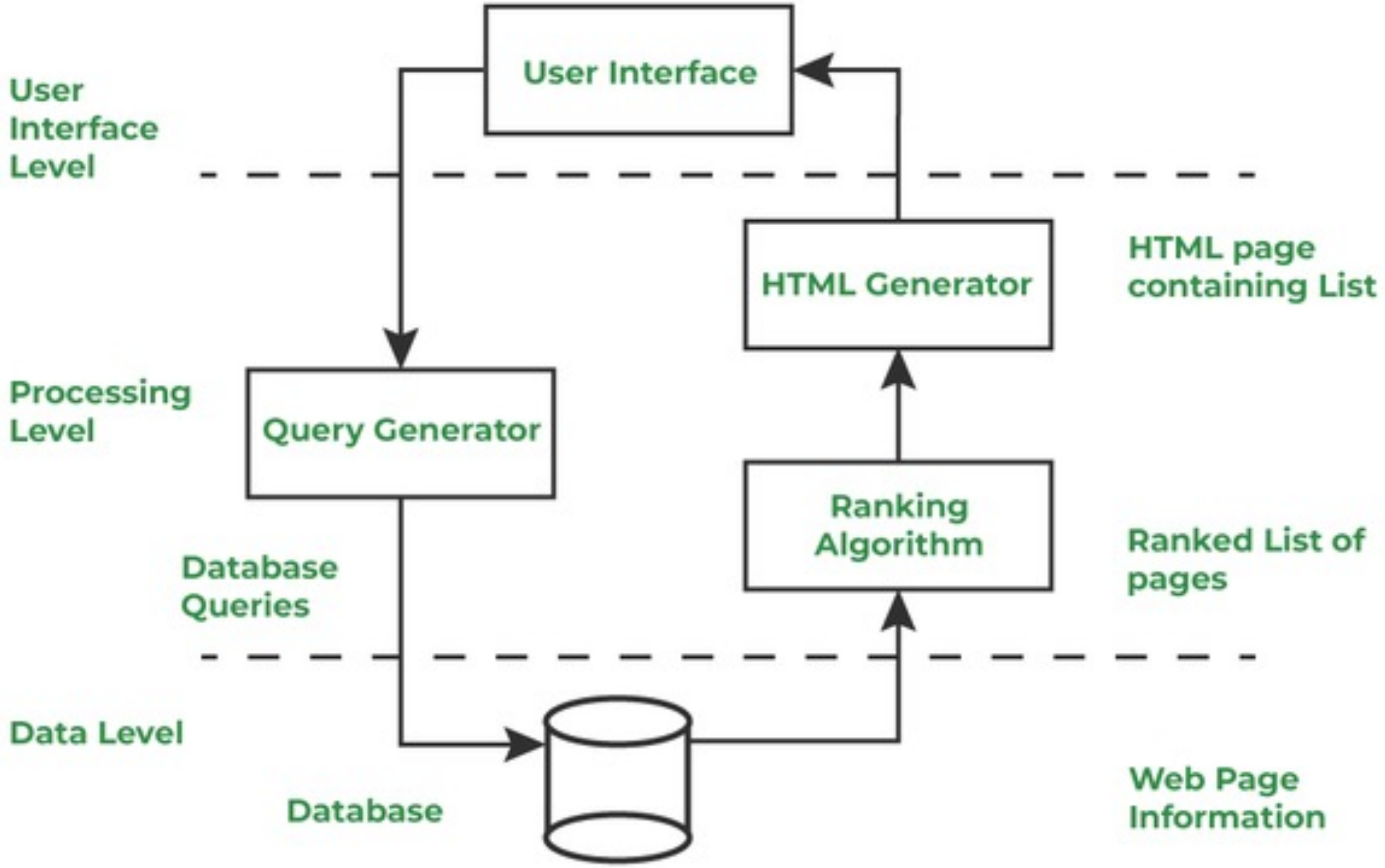
SYSTEM ARCHITECTURE: CLIENT-SERVER MODEL

- n-tier Client server architecture



SYSTEM ARCHITECTURE: CLIENT-SERVER MODEL

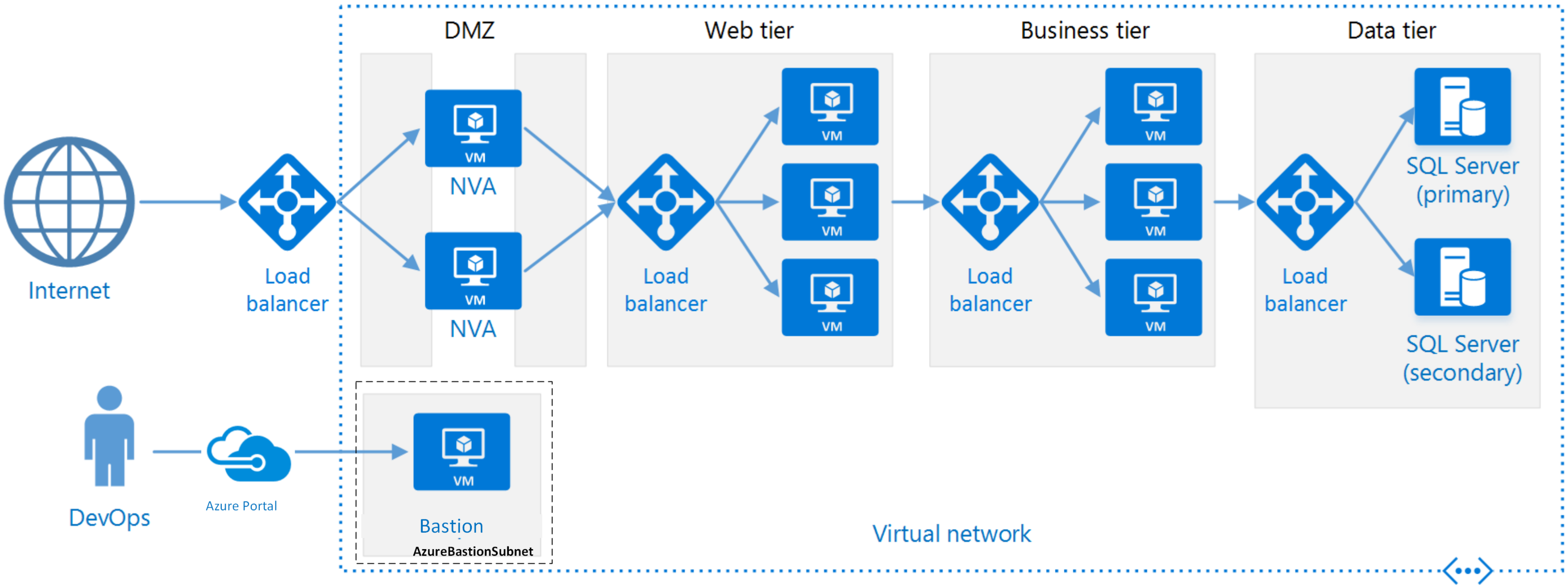
- 3-tier arch. example:
  - Internet Search Engine



Internet search engine into three different layers

SYSTEM ARCHITECTURE: CLIENT-SERVER MODEL

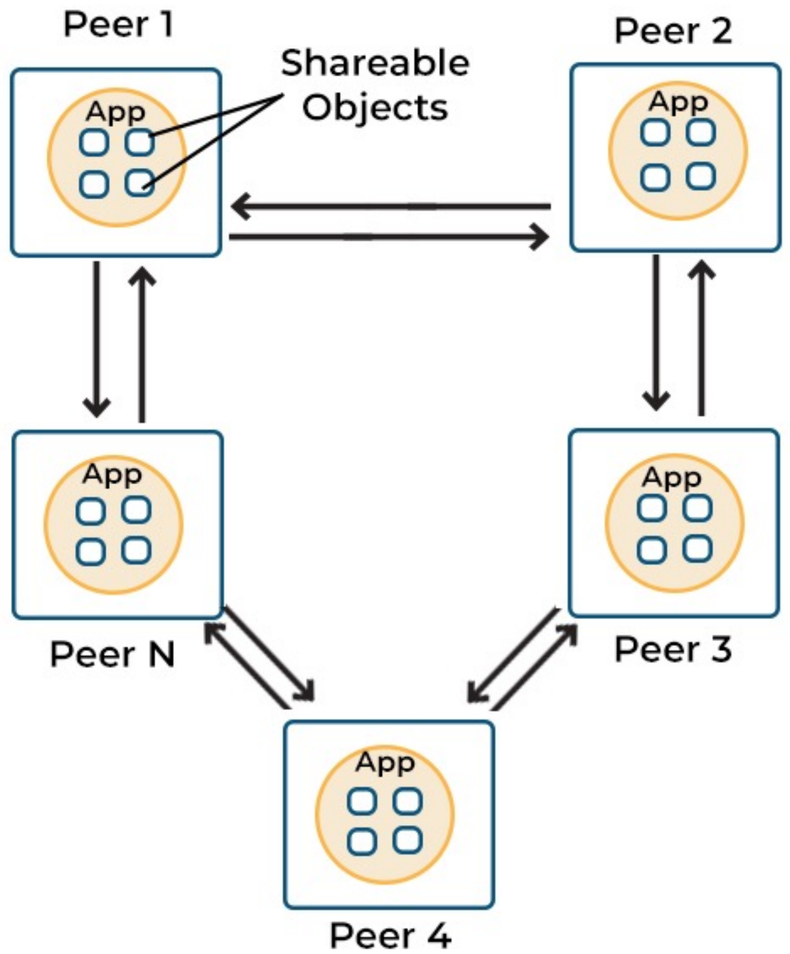
- n-tier arch. example:
  - An MS Azure application using multiple Virtual Machines



SYSTEM ARCHITECTURE: P2P MODEL

SYSTEM ARCHITECTURE: P2P MODEL

- De-Centralized / Peer to Peer model
  - No central control
  - A node can either act as a client or server at any given time once it joins the network
  - Each node in the network has the same set of responsibilities and capabilities.
  - Design of a P2P Dist Sys conforms to Application and Network overlay requirements to keep track of all nodes

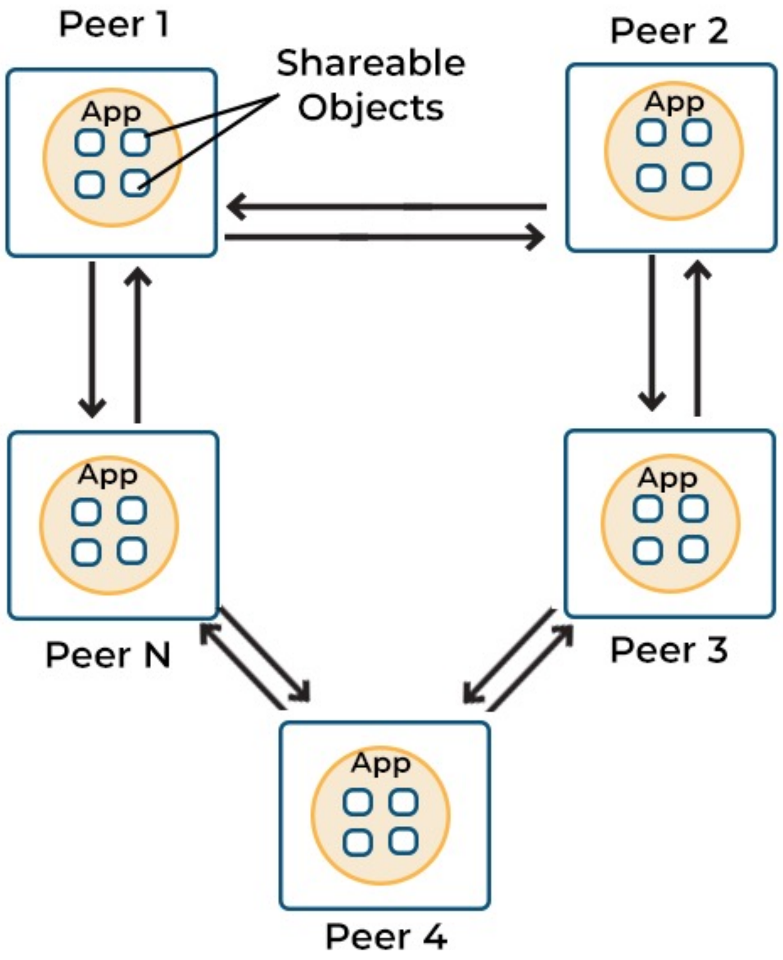


SYSTEM ARCHITECTURE: P2P MODEL

- Benefits:
  - Autonomy: Each node is independent of the other.
  - Less costly: No need to buy an expensive server.
  - No network manager
  - Adding nodes is easy: Adding, deleting, and repairing nodes in this network is easy.
  - Less network traffic than in a client/ server network.
- Challenges:
  - Less secure
  - Data is vulnerable. Stored in various nodes.
  - Slow performance

SYSTEM ARCHITECTURE: P2P MODEL

- Organization
  - The nodes (i.e., processes) are organized in an overlay that adheres to a specific, deterministic topology: a ring, a binary tree, a grid, etc.
  - Deterministic schemes can be deployed for routing messages between processes. Used to efficiently look up data.
  - Data item is uniquely associated with a key, and this key is subsequently used as an index.
  - The peer-to-peer system as a whole is now responsible for storing (key, value) pairs.
  - Each node is assigned an identifier, and each node is responsible for storing data associated with a specific subset of keys.

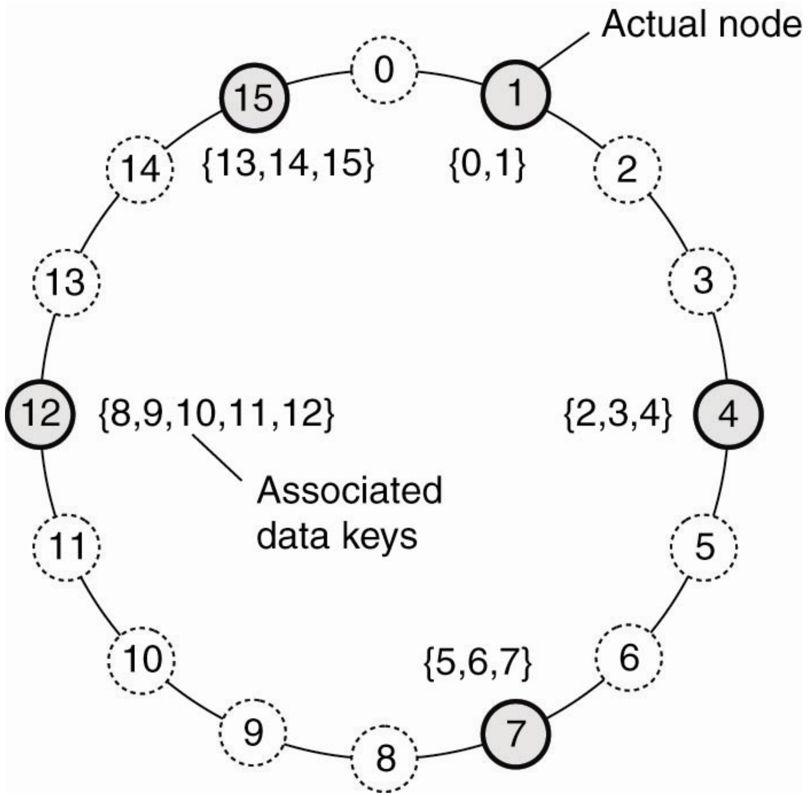


SYSTEM ARCHITECTURE: P2P MODEL

- Organization:
  - **Structured P2P**: Nodes adhere to a predefined distributed data structure.
  - **Unstructured P2P**: Networks feature nodes that randomly select their neighbors.
  - **Hybrid P2P**: Systems combine elements of both, with certain nodes assigned unique, organized functions.

SYSTEM ARCHITECTURE: P2P MODEL

- Structured P2P:
  - Typically maintains a Distributed Hash Table (DHT)
  - Each peer is responsible for a specific part of the content in the network.
  - Network use hash functions and assign values to every content and every peer in the network
  - A global protocol determines which peer is responsible for which content.
  - Whenever a peer wants to search for data, it uses the global protocol to determine the peers responsible for the data and then directs the search towards the responsible peers.



SYSTEM ARCHITECTURE: P2P MODEL

Structured P2P:

1. Node Identification and Routing:

- Each node in the network is assigned a unique identifier using a hash function.
- This identifier is typically a large number or a string of characters.
- A consistent hash function is used to map both data and node identifiers to the same address space.

2. Distributed Hash Table (DHT):

- DHT is a distributed database that maps keys to values, where keys and values are typically associated with nodes and data items.
- The DHT is divided into buckets or partitions, and each node is responsible for a specific range of keys.

3. Routing Algorithm:

- Use a routing algorithm in the network to communicate.
- E.g. Chord, Kademlia, and Pastry.

SYSTEM ARCHITECTURE: P2P MODEL

Structured P2P:

4. Joining and Leaving the Network:

- Each new node is given a unique identifier and integrated into the DHT.
- New nodes get redistributed responsibilities among existing nodes to maintain load balance.
- When a node leaves or fails, its responsibilities need to be transferred to other nodes.

5. Data Storage and Retrieval:

- Data is typically stored in the network using its hash value as a key.
- To retrieve data, a lookup in the DHT using the hash of the desired key.

6. Fault Tolerance:

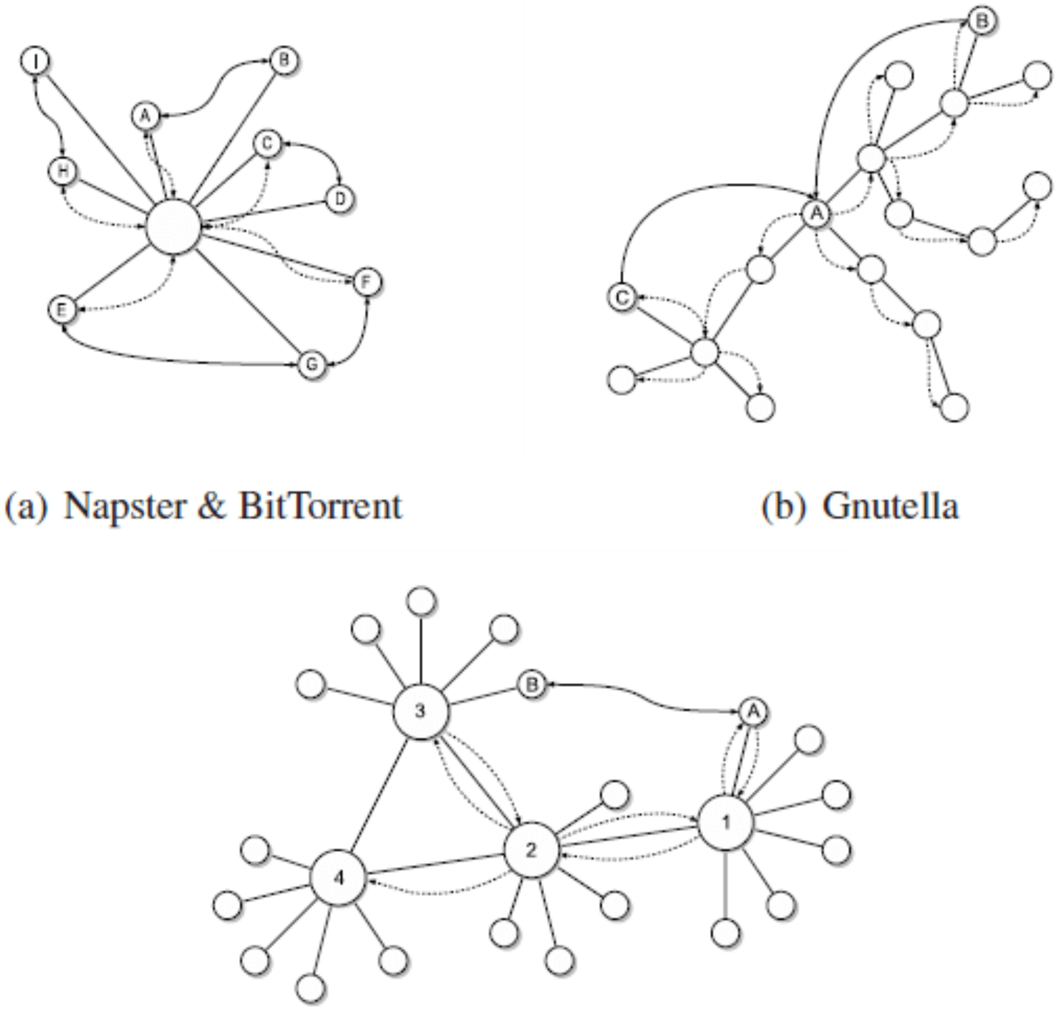
- Redundancy, replication, or backup mechanisms are employed.

7. Security and Privacy:

- Encrypted communication, authentication, and authorization mechanisms.
- Ensure the integrity and confidentiality of data and communications.

SYSTEM ARCHITECTURE: P2P MODEL

- Un-structured P2P:
  - Lack a predefined organization or topology for how nodes are connected.
  - Do not rely on distributed hash table (DHT).
  - More flexible and dynamic.
  - They are often used for applications where the focus is on simplicity, ease of deployment, and adaptability.



(a) Napster & BitTorrent

(b) Gnutella

(c) Gnutella/Overnet/eDonkey2000

SYSTEM ARCHITECTURE: P2P MODEL

Un-structured P2P:

1. Node Discovery:
    - Nodes join the network without adhering to a specific organizational structure.
    - Nodes may join and leave the network at any time.
    - There's no central authority or fixed rules for how nodes connect.
    - Some form of discovery mechanism is used to find and connect to other nodes.
2. Overlay Network:
    - An overlay network is formed where nodes are connected to each other.
    - No organization; connections between nodes are established based on various criteria.
3. Search and Communication:
    - Often rely on random or heuristic-based search methods.
    - A node broadcasts a query to its neighbors or all nodes.
    - Keeps propagating packets until the resource is found or a timeout occurs.

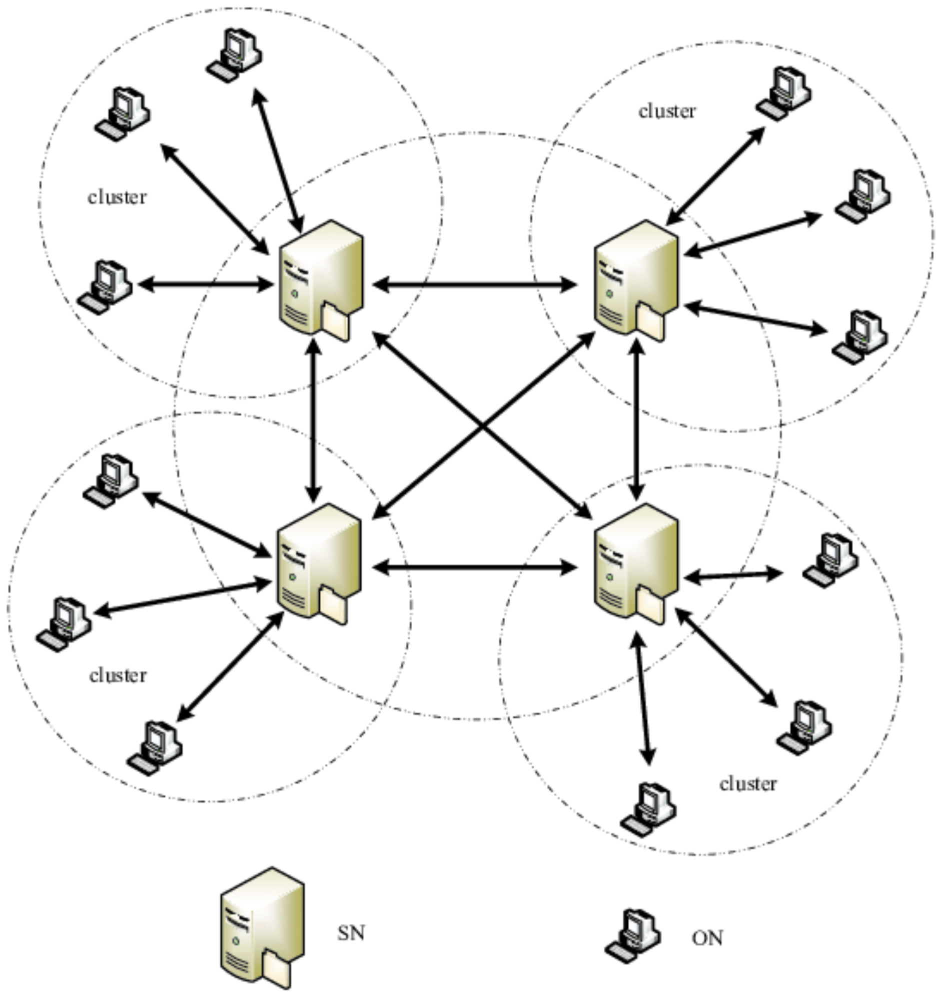
SYSTEM ARCHITECTURE: P2P MODEL

4. Data Storage and Retrieval:
    - Allow nodes to store and retrieve data without a predefined organization.
    - Nodes may replicate or cache data locally.
    - Searches for data are often performed by flooding the network with queries.
    - Efficiency of data retrieval may vary, and there could be redundancy in the storage of data.
5. Scalability and Adaptability:
    - Often more scalable and adaptable than structured systems.
    - Nodes can be added or removed without affecting the overall structure of the network.
    - Quick to adapt to changes in the network, such as in file-sharing applications.
6. Challenges:
    - Scalability issues
    - Increased search
    - Efficiency
    - Reliability.

SYSTEM ARCHITECTURE: HYBRID APPROACH

SYSTEM ARCHITECTURE: HYBRID MODEL

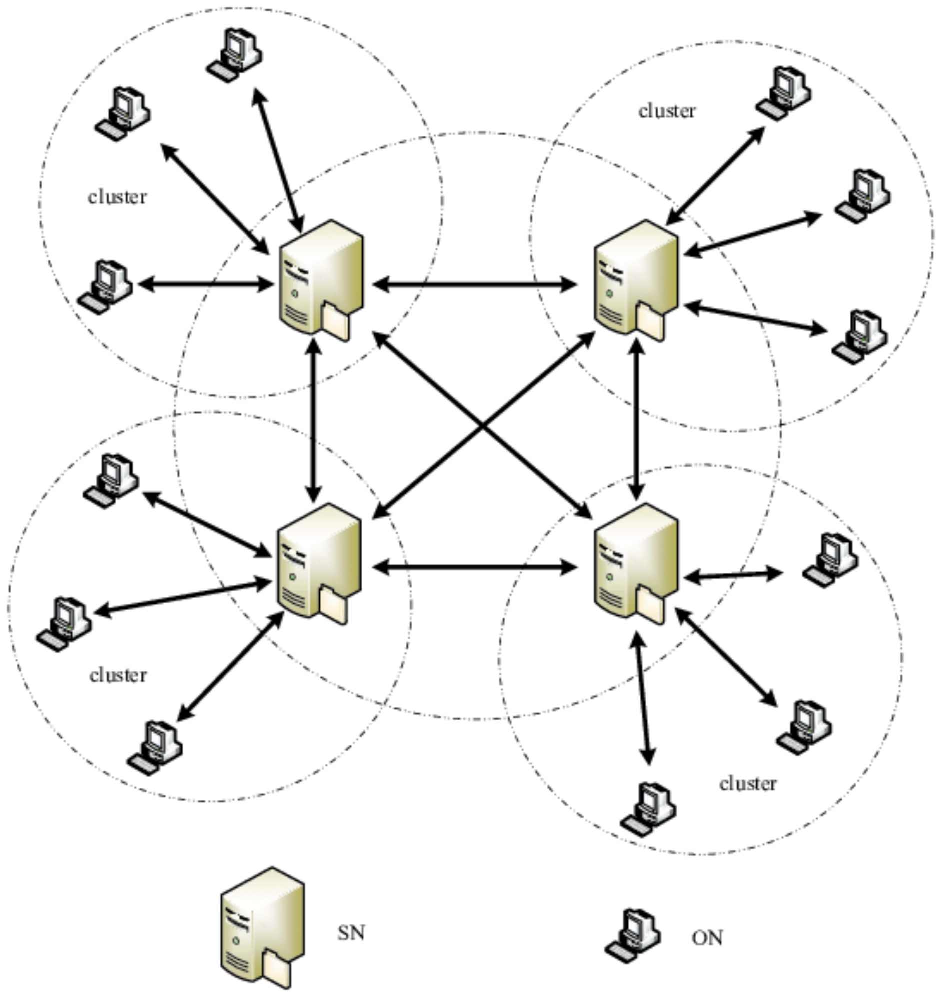
- Hybrid P2P/Client Server:
  - A combination of peer-to-peer and client–server models.
  - A common hybrid model is to have a central server that helps peers find each other
  - There are a variety of hybrid models, all of which make trade-offs between the centralized functionality provided by a structured server/client network and the node equality afforded by the pure peer-to-peer unstructured networks.
  - Currently, hybrid models have better performance than either pure unstructured networks or pure structured networks.



Shunzhi Wang, Zhanyou Ma, Rong Wang et al. Performance analysis of a queueing system based on working vacation with repairable fault in the P2P network, 21 September 2022, Supercomputing [https://doi.org/10.21203/rs.3.rs-1864515/v2]

SYSTEM ARCHITECTURE: HYBRID MODEL

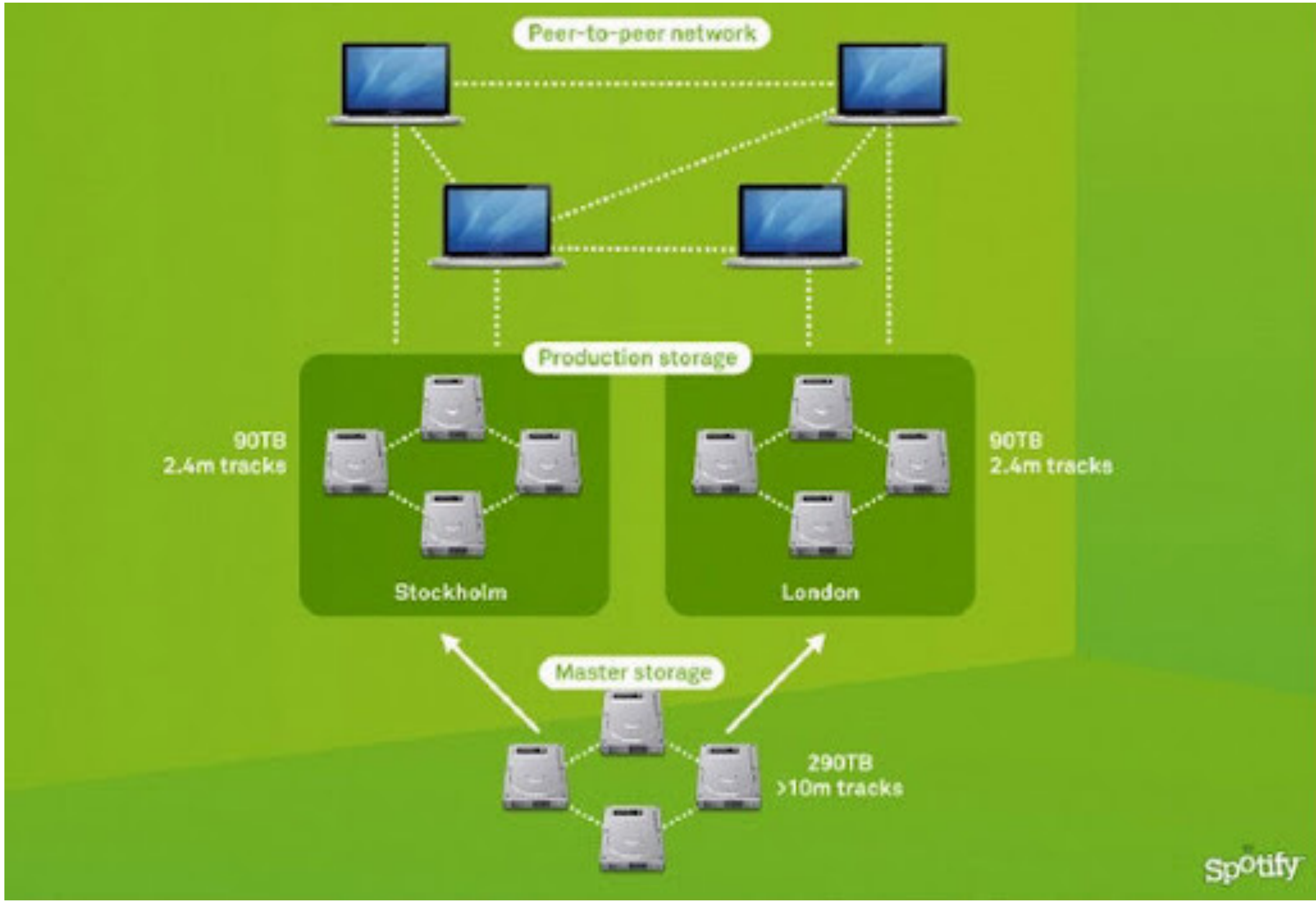
- Benefits
  - Efficient Data Retrieval
  - Scalability
  - Adaptability and Flexibility
  - Fault Tolerance
  - Load Balancing
  - Dynamic Resource Discovery
- Challenges
  - Complexity
  - Overhead
  - Consistency
  - Increased Latency
  - Resource Utilization
  - Security and Privacy Concerns



Shunzhi Wang, Zhanyou Ma, Rong Wang et al. Performance analysis of a queueing system based on working vacation with repairable fault in the P2P network, 21 September 2022, Supercomputing [https://doi.org/10.21203/rs.3.rs-1864515/v2]

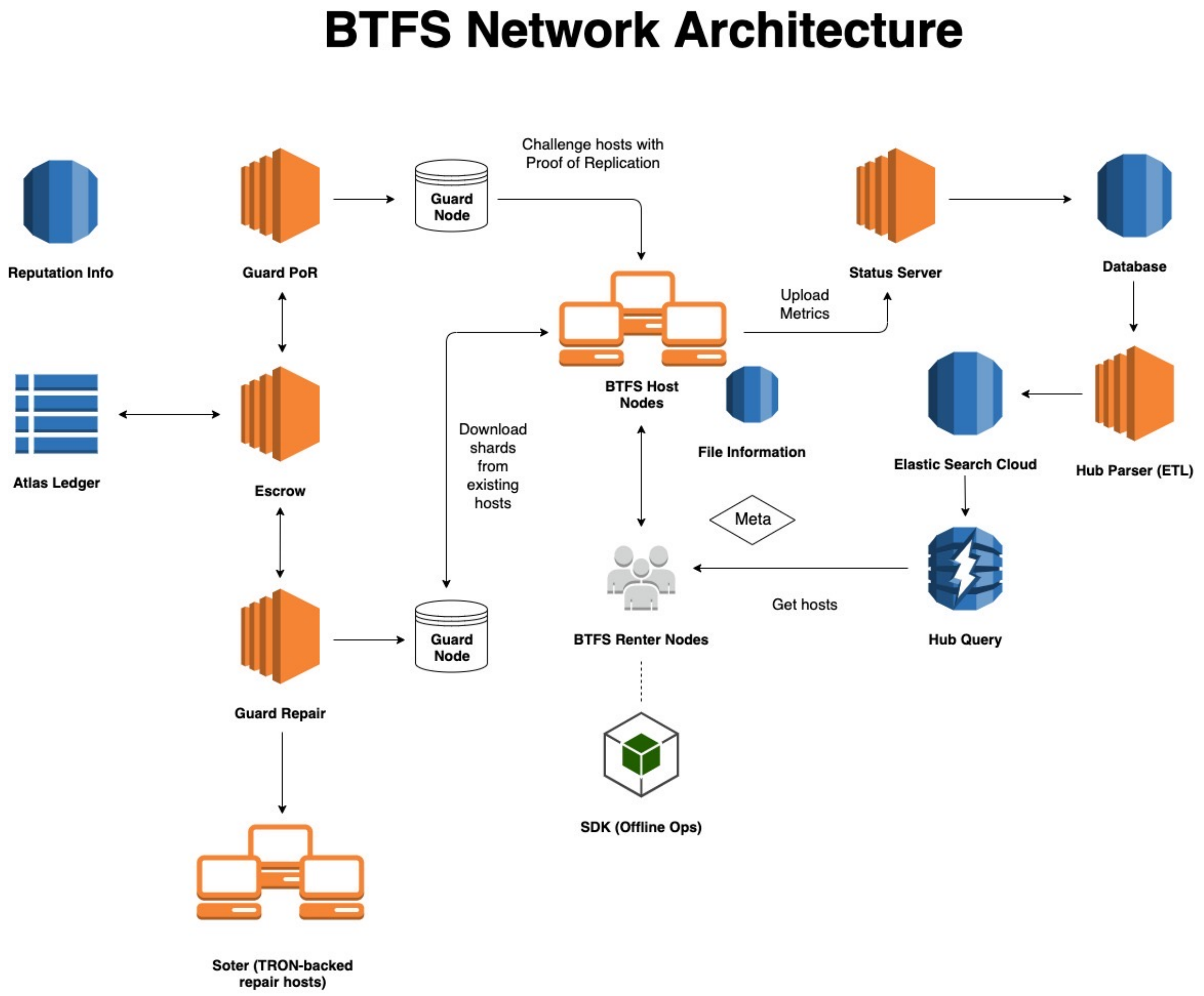
SYSTEM ARCHITECTURE: HYBRID MODEL

- Hybrid P2P-Client-Srvr:
  - Example: Spotify (before 2014)



SYSTEM ARCHITECTURE: HYBRID MODEL

- Hybrid P2P-Client-Srvr:
  - Example: Bittorent



https://docs.btfs.io/v1.0/docs/what-is-btfs#architecture

SYSTEM ARCHITECTURE: HYBRID MODEL

- Hybrid P2P-Client-Srvr:
  - Example: Deep Torrent crawler

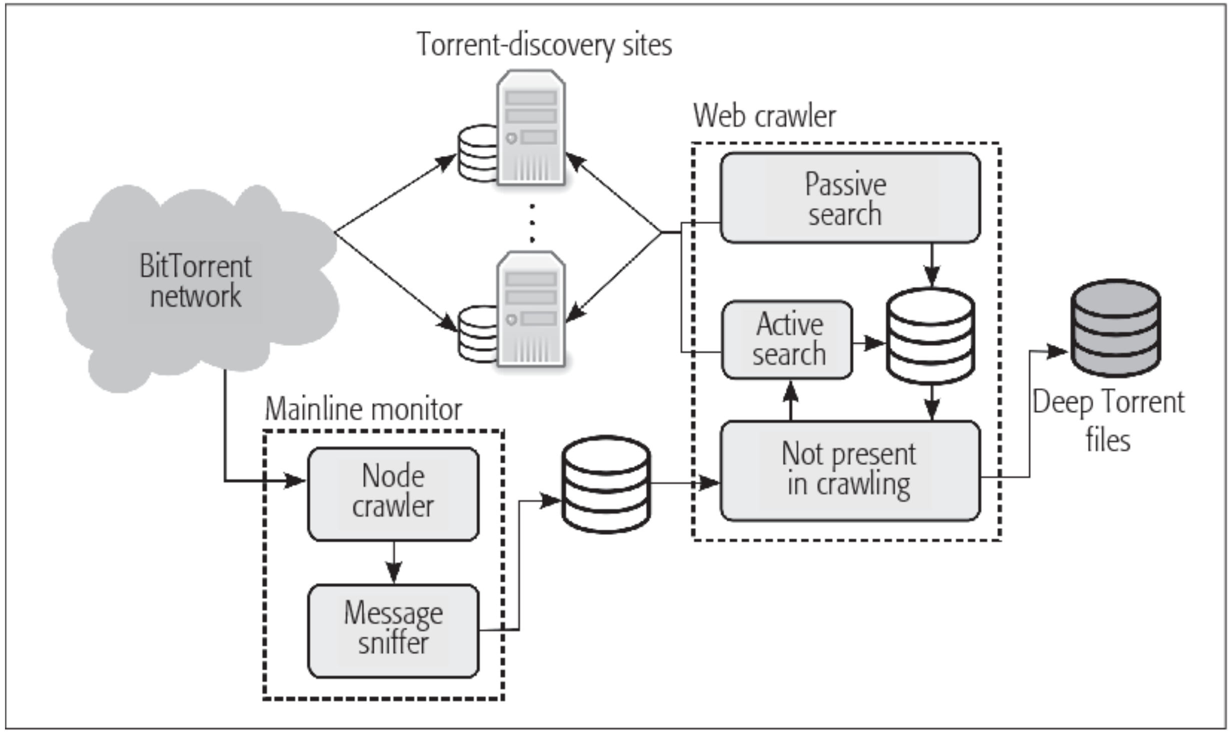
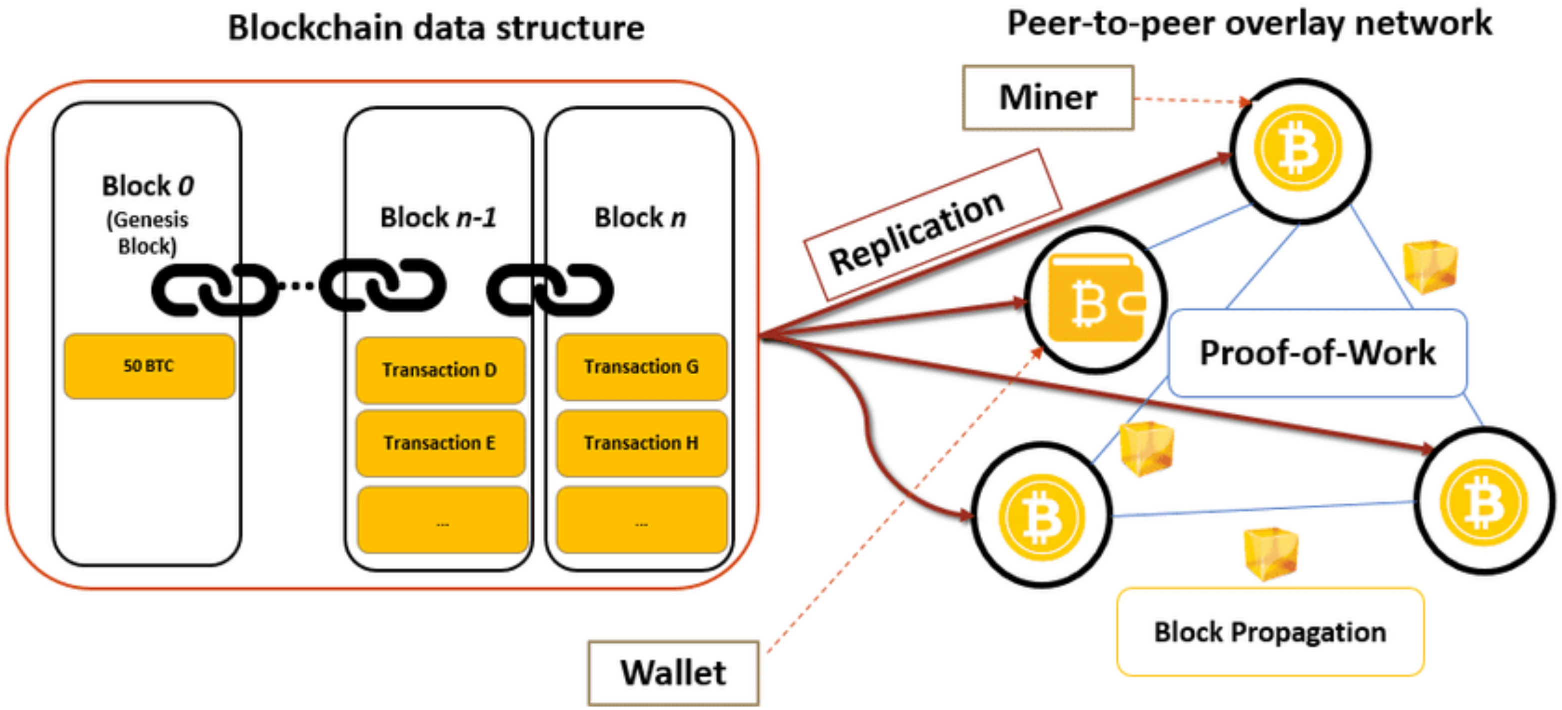


Figure 1. Functional architecture of the Deep Torrent crawler.

Rodríguez-Gómez, Rafael et al. "On Understanding the Existence of a Deep Torrent." *IEEE Communications Magazine* 55 (2017): 64-69.

SYSTEM ARCHITECTURE: HYBRID MODEL

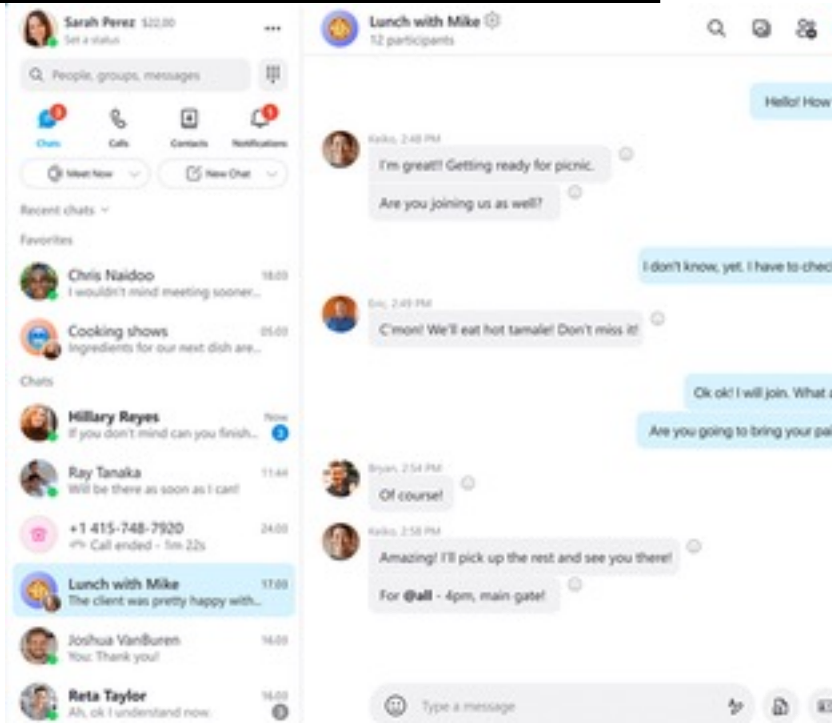
- Hybrid P2P-Client-Srvr:
  - Example: Bitcoin, Etherium Blockchain



Y. Shahsavari, K. Zhang and C. Talhi, "Performance Modeling and Analysis of the Bitcoin Inventory Protocol," *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, Newark, CA, USA, 2019, pp. 79-88, doi: 10.1109/DAPPCON.2019.00019.

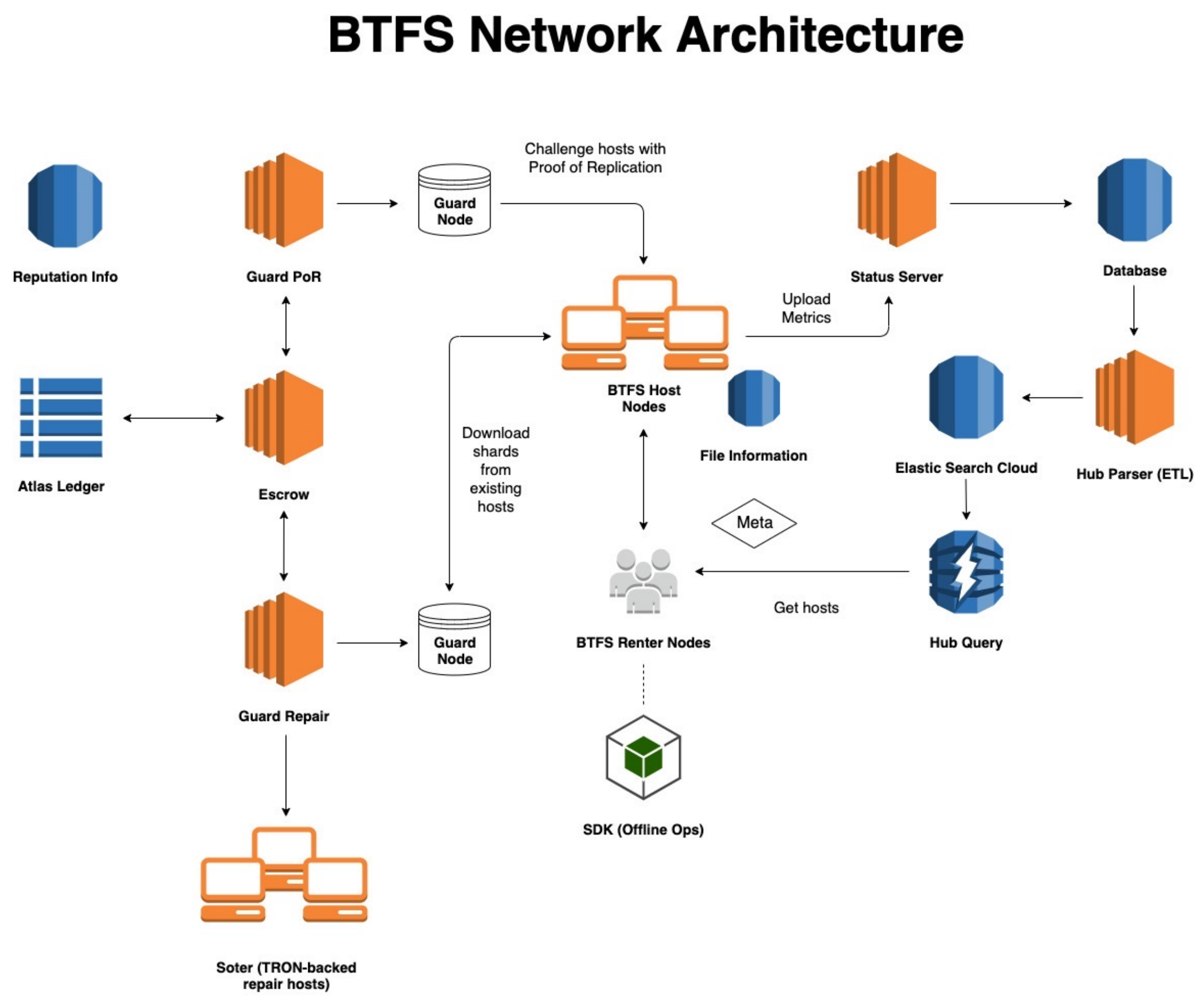
SYSTEM ARCHITECTURE: HYBRID MODEL

- Hybrid P2P-Client-Srvr:
  - Other Examples: Gnutella, eDonkey, Kazaa, Napster, Skype etc



SYSTEM ARCHITECTURE: HYBRID MODEL

- Hybrid P2P-Client-Srvr:
  - Example: Bittorent

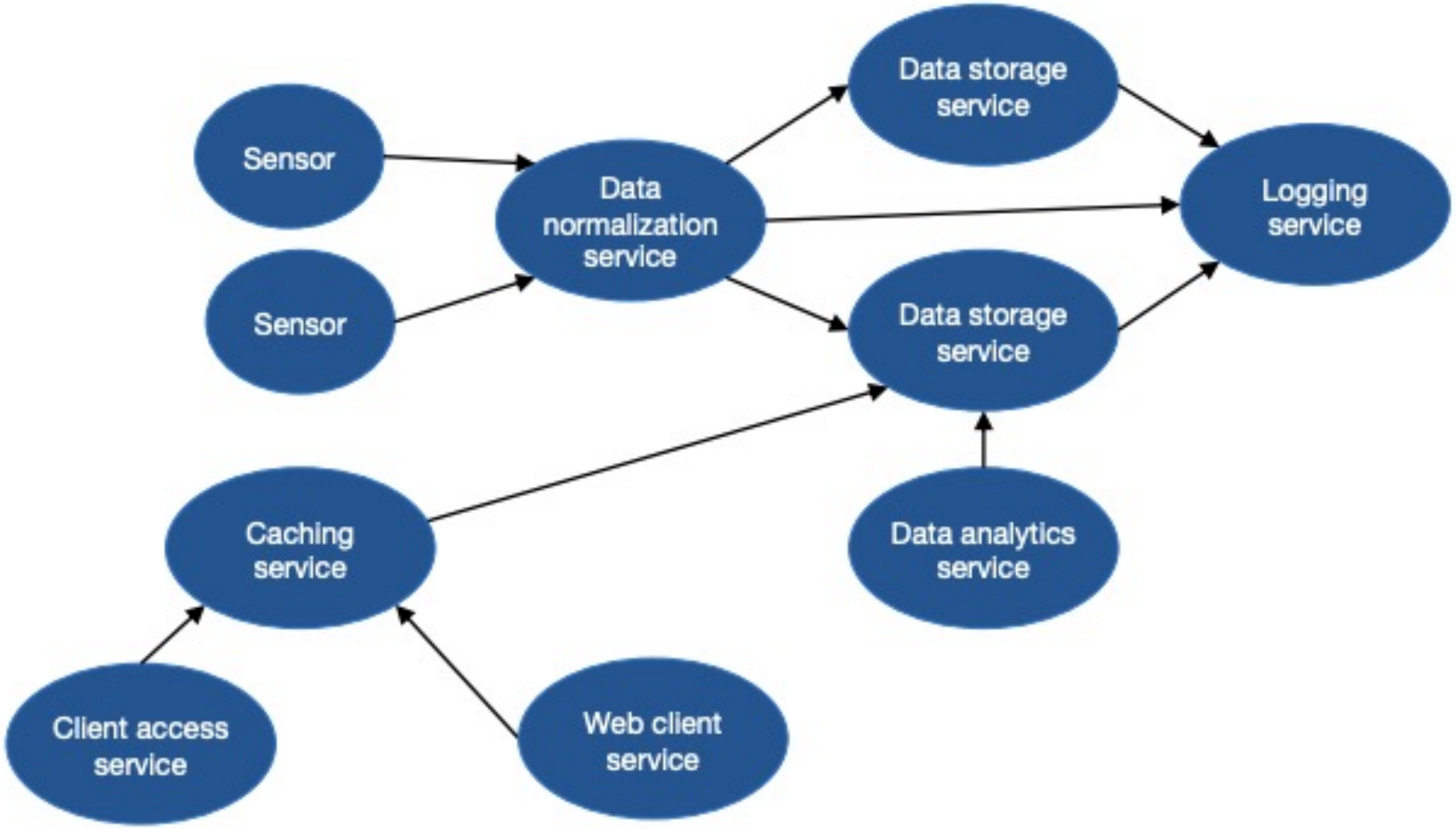


https://docs.btfs.io/v1.0/docs/what-is-btfs#architecture

DISTRIBUTED SYSTEMS SERVICES

DISTRIBUTED SYSTEMS SERVICES

- A distributed system is a collection of services accessed via network interfaces



DISTRIBUTED SERVICES

- Serverless Computing: The rise of serverless computing, where developers can focus on writing code without worrying about infrastructure management.
- Edge Computing: Bringing computing resources closer to the data source, enabling faster processing and reduced latency.
- Container Orchestration: Simplifying the deployment and management of distributed services using container orchestration platforms like Kubernetes.

DISTRIBUTED SERVICES

- Serverless Computing:
  - Depends on underlying physical servers, however there is no server hardware or operating system environment to manage for developers or IT engineers.
  - Abstracts applications from the underlying server and operating system, serverless functions are easier to deploy and manage
  - Event-driven computing; use resources as you go; deploy serverless functions and APIs
  - More efficient than conventional applications that run constantly
  - Auto-scaling enabled, cost-effective

DISTRIBUTED SERVICES

- Edge Computing:
  - Moves some portion of storage and compute resources out of the central data center and closer to the source of the data itself.
  - Compute, Store, Network, Service closer to the data-source.
  - Lighter, faster, efficient, cheaper.
  - Examples: Security system monitoring, IoT devices, Self-driving cars, Medical monitoring devices, Video conferencing etc.

CLOUDFLARE IBM

DISTRIBUTED SERVICES

- Kubernetes and Container Orchestration
  - A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing unit
  - Container orchestration automatically provisions, deploys, scales, and manages containerized applications without worrying about the underlying infrastructure.
  - Developers can implement container orchestration anywhere containers are, allowing them to automate the life cycle management of containers.

SUMMARY

- Distributed Systems Themes
- Dist. Sys. Challenges
- Dist. Sys. Architecture
  - Software Architecture
  - Systems Architecture
    - Client-server arch.
    - P2P arch.
    - Hybrid approach
- Distributed Services