Lecturer: Dr. Michael J. May
Kinneret College

Introduction to Distributed Systems, Client/Server and Peer to Peer

3 November 2024
Lecture 1

Most Slide Credits: Maarten van Steen

Distributed System: Definition

A distributed system is a piece of software that ensures that:

> ***a collection of autonomous computing elements that appears to its users as a single coherent system***

Two aspects: (1) Autonomous computing elements, also referred to as nodes, be they hardware devices or software processes and (2) Single coherent system: users or applications perceive a single system → nodes need to collaborate.

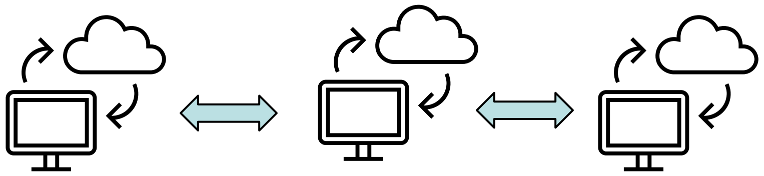
Each node is autonomous:

- Its own notion of time → there is no global clock
- Leads to fundamental synchronization and coordination problems.

Collection of nodes and group:

- How to manage group membership?
- How to know you are communicating with an authorized member?

Why build them?

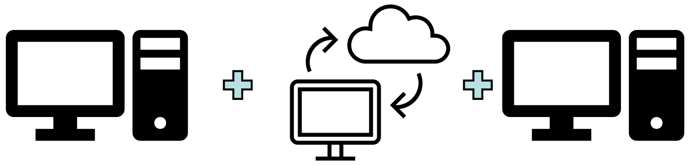


Integrative view:

Connecting existing networked computer systems into a larger a system.

Expansive view:

An existing networked computer systems is extended with additional computers



Decentralized vs. Distributed

Decentralized

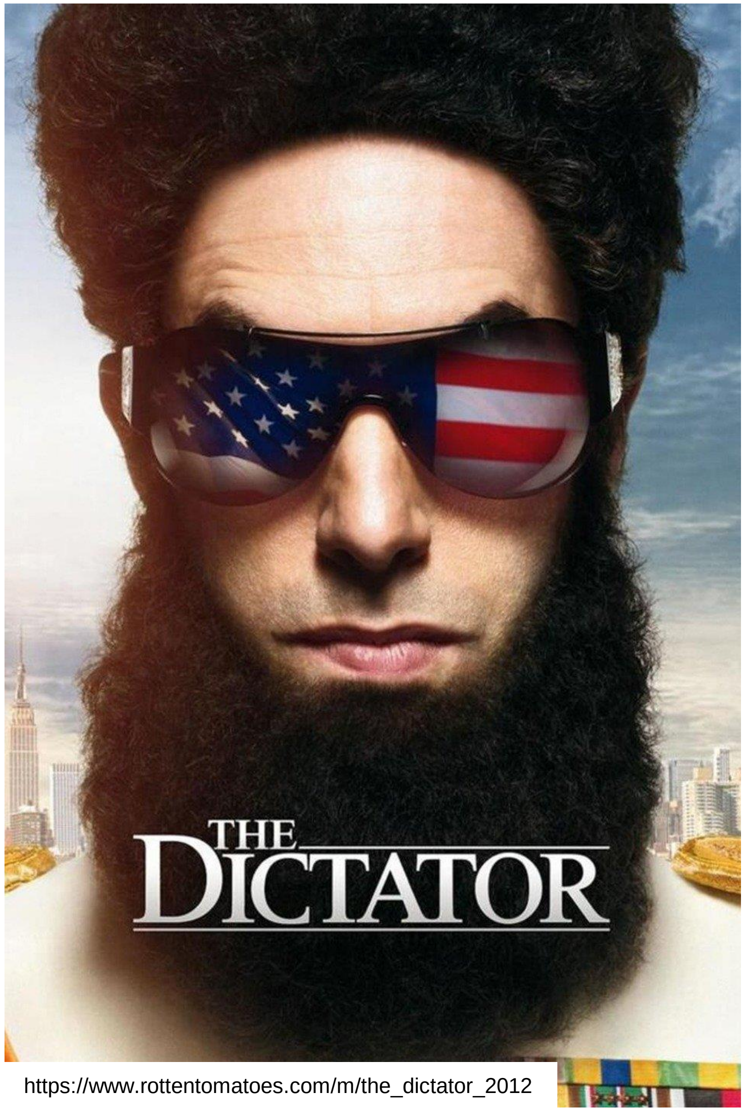
- A networked computer system in which processes and resources are necessarily spread across multiple computers.
- We must spread processes and resources across computers

Distributed

- A networked computer system in which processes and resources are sufficiently spread across multiple computers.
- We spread until we have reached some level of sufficiency – it’s distributed enough

Centralized Solutions

- A single computer that decides and rules
  - Order
  - Simplicity



https://www.rottentomatoes.com/m/the_dictator_2012

Centralized Pluses

- Easy coordination
- Assignment of responsibility

Centralized Misconceptions 🤓

Misconception: Centralized solutions do not scale

Distinguish logically and physically centralized.

Example: The roots of the Domain Name System:

- logically centralized
- physically (massively) distributed
- decentralized across several organizations

DNS

Centralized Misconceptions 🤓

Misconception: Centralized solutions have a single point of failure

Generally not true (e.g., the root of DNS). A single point of failure is often:

- easier to manage
- easier to make more robust

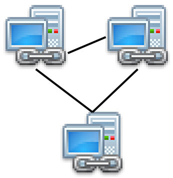
Organization

Overlay network

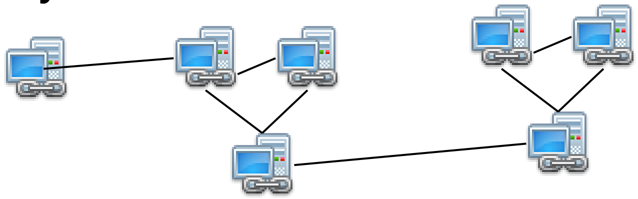
- Each node in the collection communicates only with other nodes in the system, its neighbors. The set of neighbors may be dynamic, or may even be known only implicitly (i.e., requires a lookup).

Overlay types

- Well-known example of overlay networks: peer-to-peer systems.
  - Structured: each node has a well-defined set of neighbors with whom it can communicate (tree, ring).



  - Unstructured: each node has references to randomly selected other nodes from the system.



Coherent system

- The collection of nodes as a whole operates the same, no matter where, when, and how interaction between a user and the system takes place.
- Examples:

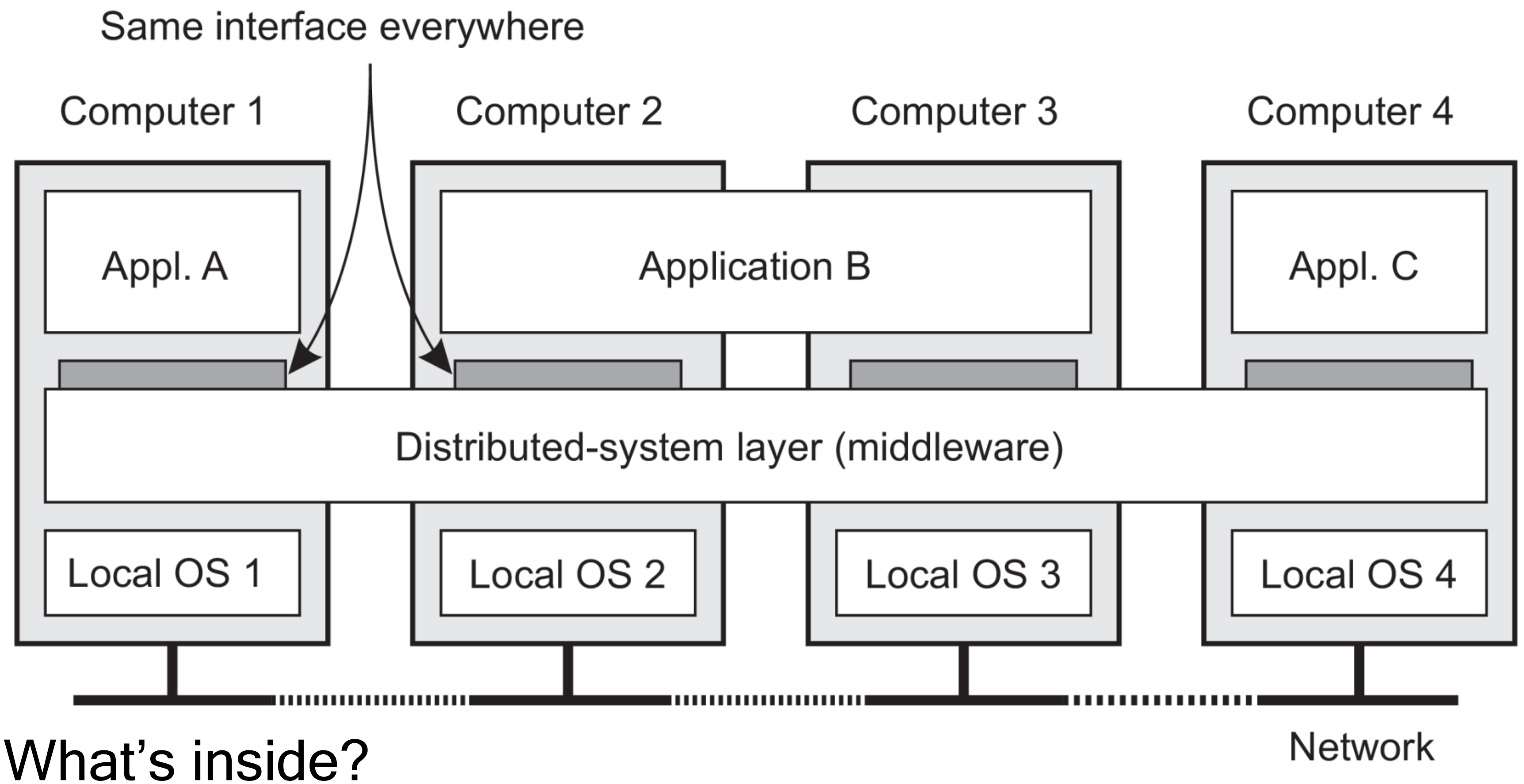
1. An end user cannot tell where a computation is taking place
2. Where data is exactly stored should be irrelevant to an application
3. Whether data has been replicated or not is completely hidden

Key: Distribution Transparency

Snag: Partial Failures

- It's inevitable that at any time only a part of the distributed system fails. Hiding partial failures and their recovery is often very difficult and in general impossible to hide.

Middleware: OS of Distributed Systems



What's inside?

- Commonly used components and functions that need not be implemented by applications separately.

What do we want to achieve?

Supporting sharing of resources

Distribution transparency

Openness

Scalability

Sharing resources

- Canonical examples
  - Cloud-based shared storage and files
  - Peer-to-peer assisted multimedia streaming
  - Shared mail services (think of outsourced mail systems)
  - Shared Web hosting (think of content distribution networks)

Observation:

- “The network is the computer”

  (quote from John Gage, then at Sun Microsystems)

Distribution Transparency

Transparency	**Description**
Access	Hide differences in data representation and how an object is accessed
Location	Hide where an object is located
Relocation	Hide that an object may be moved to another location while in use
Migration	Hide that an object may move (itself) to another location
Replication	Hide that an object is replicated
Concurrency	Hide that an object may be shared by several independent users
Failure	Hides failure and recovery of objects

Note: Distribution transparency is a nice goal, but aiming at full distribution transparency may be too much

Degree of Transparency

Observation: Aiming at full distribution transparency may be too much.

- There are communication latencies that cannot be hidden
- Completely hiding failures of networks and nodes is (theoretically and practically) impossible
  - You cannot distinguish a slow computer from a failing one
  - You can never be sure that a server actually performed an operation before a crash
- Full transparency will cost performance, exposing distribution of the system
  - Keeping replicas exactly up-to-date with the master takes time
  - Immediately flushing write operations to disk for fault tolerance

Exposing Distribution

- Exposing distribution may be good
  - Making use of location-based services (finding your nearby friends)
  - When dealing with users in different time zones
  - When it makes it easier for a user to understand what's going on (when e.g., a server does not respond for a long time, report it as failing).

- Conclusion:
  - Distribution transparency is a nice a goal, but achieving it is a different story, and it should often not even be aimed at.

Openness of Distributed Systems

Open distributed system

Be able to interact with services from other open systems, irrespective of the underlying environment:

- Systems should conform to well-defined interfaces
- Systems should support portability of applications
- Systems should easily interoperate

Achieving openness

At least make the distributed system independent from heterogeneity of the underlying environment:

- Hardware
- Platforms
- Languages

Openness of Distributed Systems

This is hard

Scale in Distributed Systems

Observation

Many developers of modern distributed systems easily use the adjective "scalable" without making clear **why** their system actually scales.

Scalability

At least three components:

- Number of users and/or processes (size scalability)
- Maximum distance between nodes (geographical scalability)
- Number of administrative domains (administrative scalability)

Observation

Most systems account only, to a certain extent, for size scalability. The (non)solution: powerful servers. Today, the challenge lies in geographical and administrative scalability.

Geographical scalability

Cannot simply go from LAN to WAN:	WAN links are often inherently unreliable	Lack of multipoint communication
• Many distributed systems assume synchronous client-server interactions • Client sends request and waits for an answer. • Latency may easily prohibit this scheme.	• Simply moving streaming video from LAN to WAN will fail.	• A simple search broadcast cannot be deployed. • Solution: Develop separate naming and directory services (having their own scalability problems).

Administrative scalability

Essence: Conflicting policies concerning usage (and thus payment), management, and security

Examples:

- Computational grids: share expensive resources between different domains.
- Shared equipment: how to control, manage, and use a shared radio telescope constructed as large-scale shared sensor network?

Exception: Several peer-to-peer networks

- File-sharing systems (based, e.g., on BitTorrent)
- Peer-to-peer telephony (Skype)
- Peer-assisted audio streaming (Spotify)

Note: End users collaborate and not administrative entities.

Size Scalability

- Root causes for scalability problems with centralized solutions

Computational capacity, limited by the CPUs

Storage capacity, including the transfer rate between CPUs and disks

Network between the user and the centralized service

Size Scaling Technique 1

Hide communication latencies

Avoid waiting for responses; do something else:

Make use of asynchronous communication

Have separate handler for incoming response

Problem: not every application fits this model

Size Scaling Technique 2

Replication/caching

Make copies of data available at different machines:

- Replicated file servers and databases
- Mirrored Web sites
- Web caches (in browsers and proxies)
- File caching (at server and client)

Replication Problems

Observation

Applying scaling techniques is easy, except for one thing:

- Having multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.
- Always keeping copies consistent and in a general way requires global synchronization on each modification.
- Global synchronization precludes large-scale solutions.

Observation

If we can tolerate inconsistencies, we may reduce the need for global synchronization, but tolerating inconsistencies is application dependent.

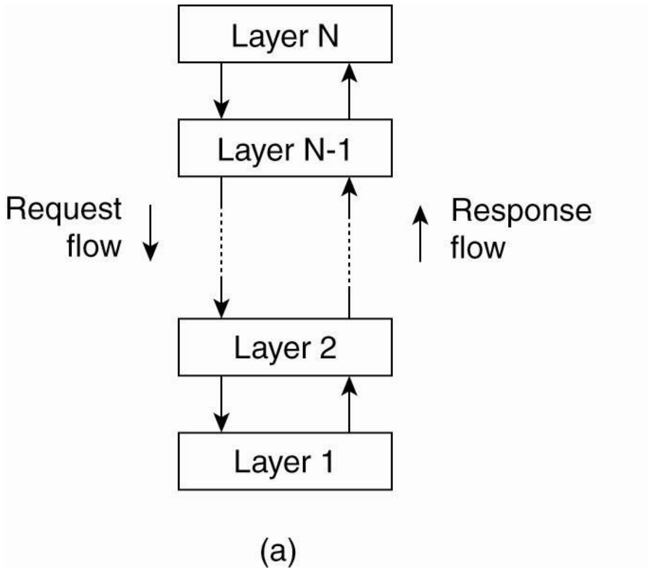
So Far

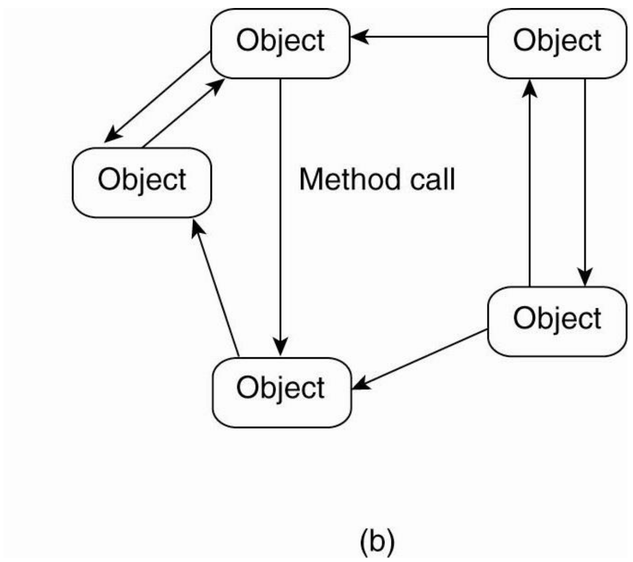
- Definitions and Goals
- Architectural Styles
- System Architectures
- Peer to Peer
- Edge Computing
- Blockchain basics

Architectural Styles

Basic Idea

Organize into logically different components and distribute those components over the various machines





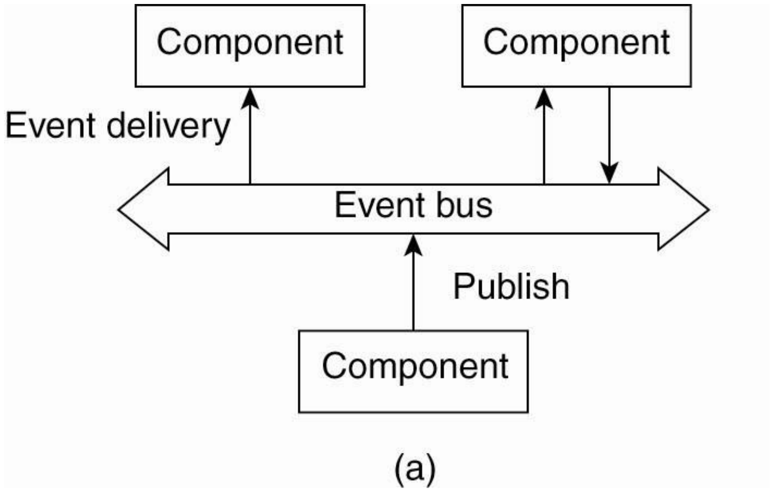
(a) Layered style is used for client-server system

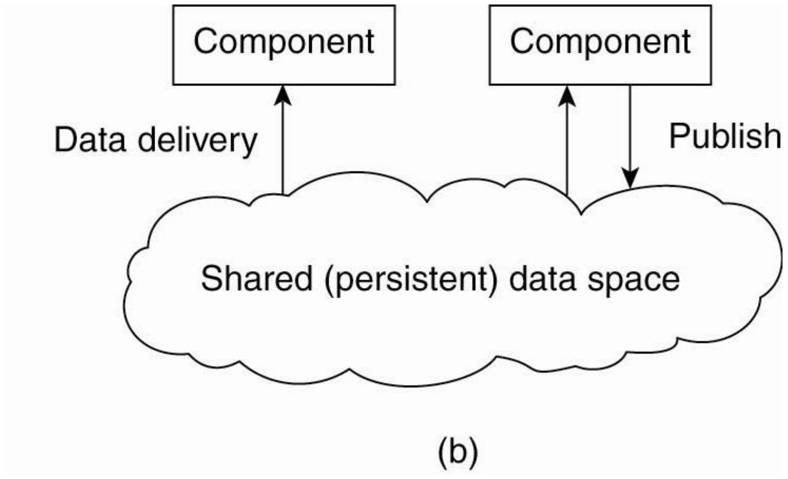
(b) Object-based style for distributed object systems

Architectural Styles

Observation

Decoupling processes (“anonymous”) and removing time constraints (“asynchronous”) led to alternative styles.





(a) Publish/Subscribe [anonymous]

(b) Shared dataspace [anonymous and asynchronous]

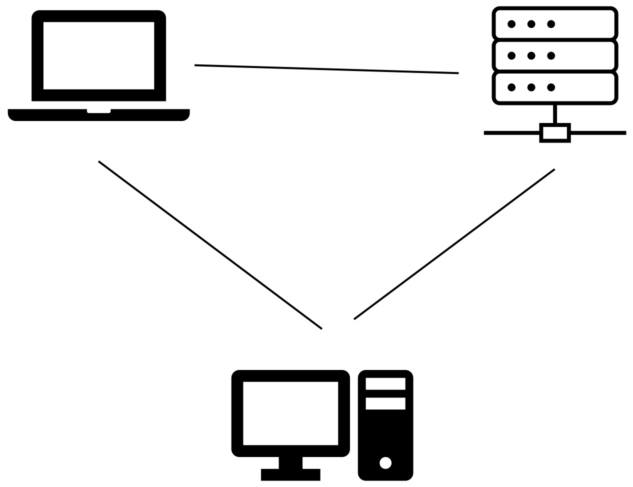
So Far

- Definitions and Goals
- Architectural Styles
- System Architectures
- Peer to Peer
- Edge Computing
- Blockchain basics

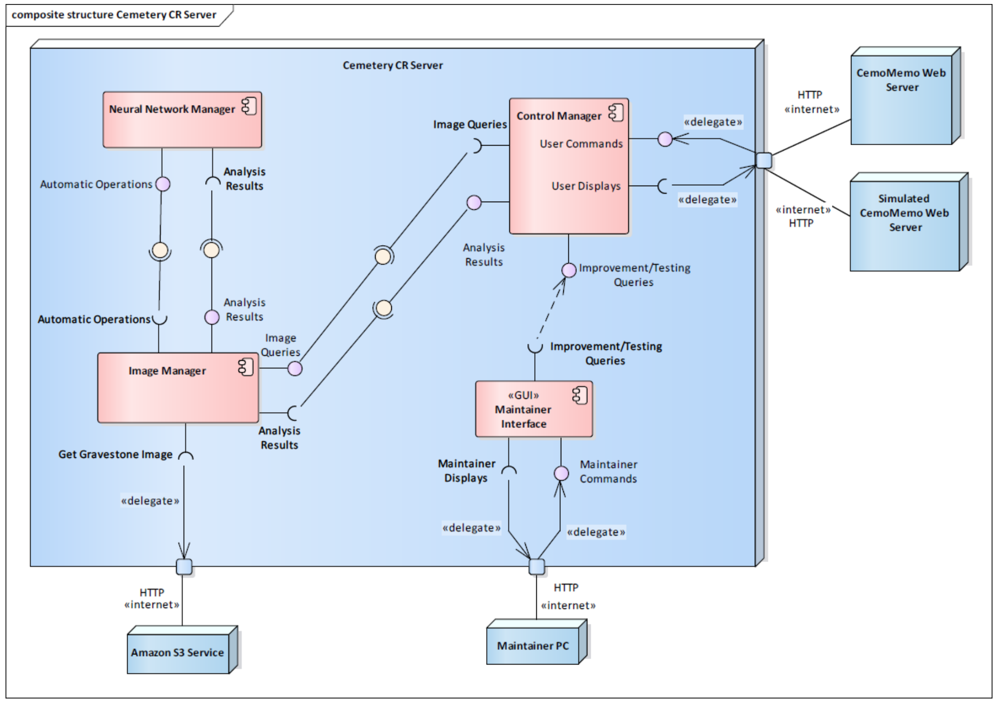
Discern

Physical architecture

- How many computers
- Where they are located



Logical architecture & Deployment

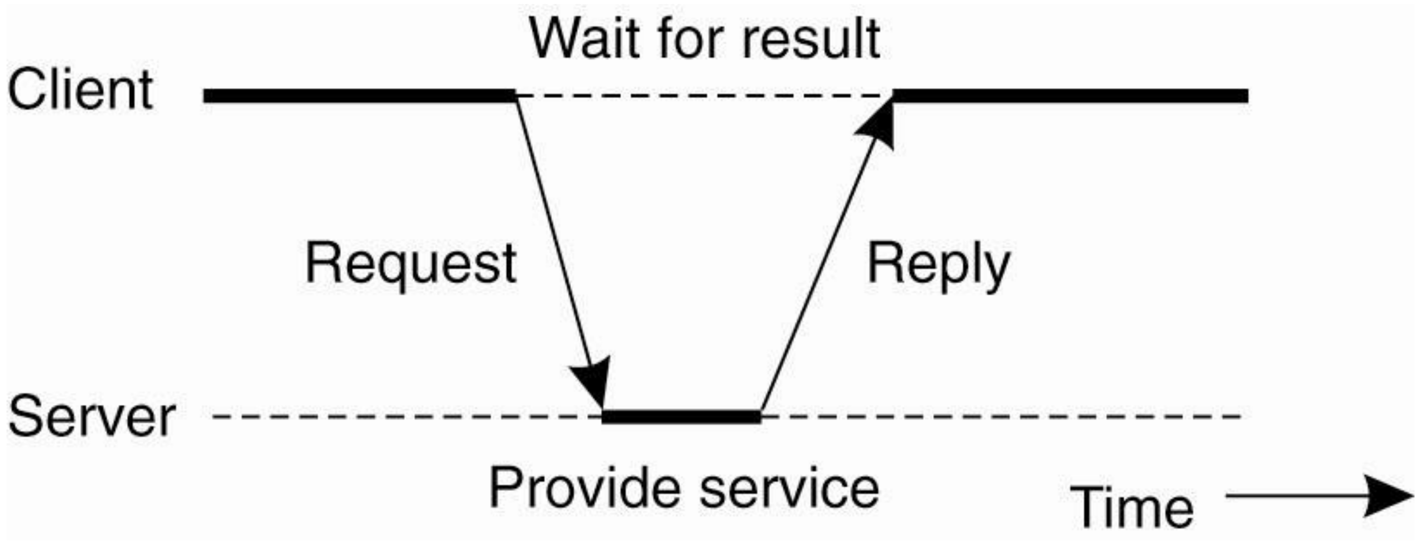


Phys: Centralized Architectures

Basic Client-Server Model

Characteristics:

- There are processes offering services (servers)
- There are processes that use services (clients)
- Clients and servers can be on different machines
- Clients follow request/reply model with respect to using services



Log: Application Layering

Traditional three-layered view

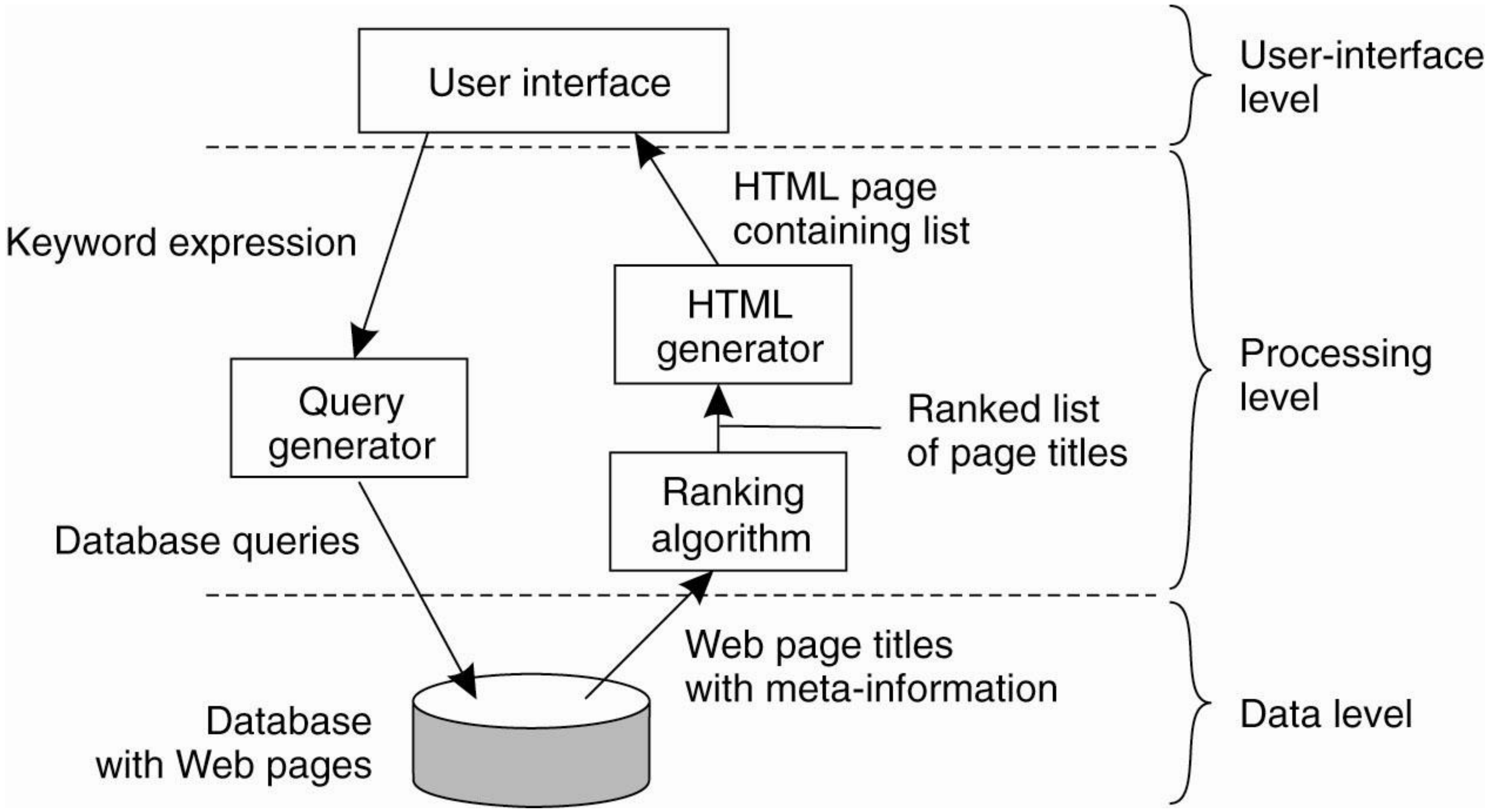
- User-interface layer contains units for an application’s user interface
- Processing layer contains the functions of an application, i.e. without specific data
- Data layer contains the data that a client wants to manipulate through the application components

---

Observation

This layering is found in many distributed information systems, using traditional database technology and accompanying applications

Application Layering



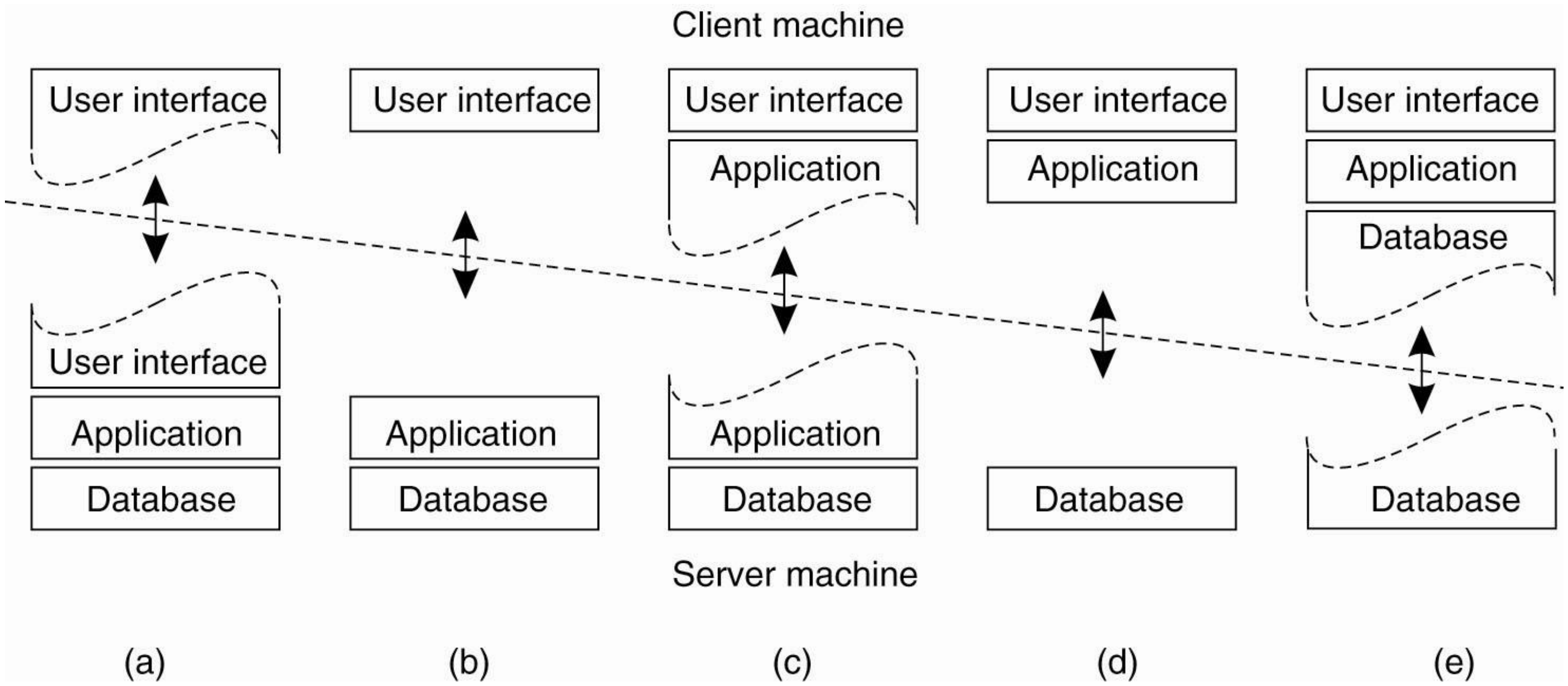
Dep: Multi-Tiered Architectures

Single-tiered: dumb terminal/mainframe configuration

Two-tiered: client/single server configuration

Three-tiered: each layer on a separate machine

Traditional two-tiered configurations:



Phys: Decentralized Arch

Observation

Tremendous growth in peer-to-peer systems

- Structured P2P: nodes are organized following a specific distributed data structure
- Unstructured P2P: nodes have randomly selected neighbors
- Hybrid P2P: some nodes are appointed special functions in a well-organized fashion

---

Note

In virtually all cases, we are dealing with overlay networks: data is routed over connections setup between the nodes (via application level multicasting)

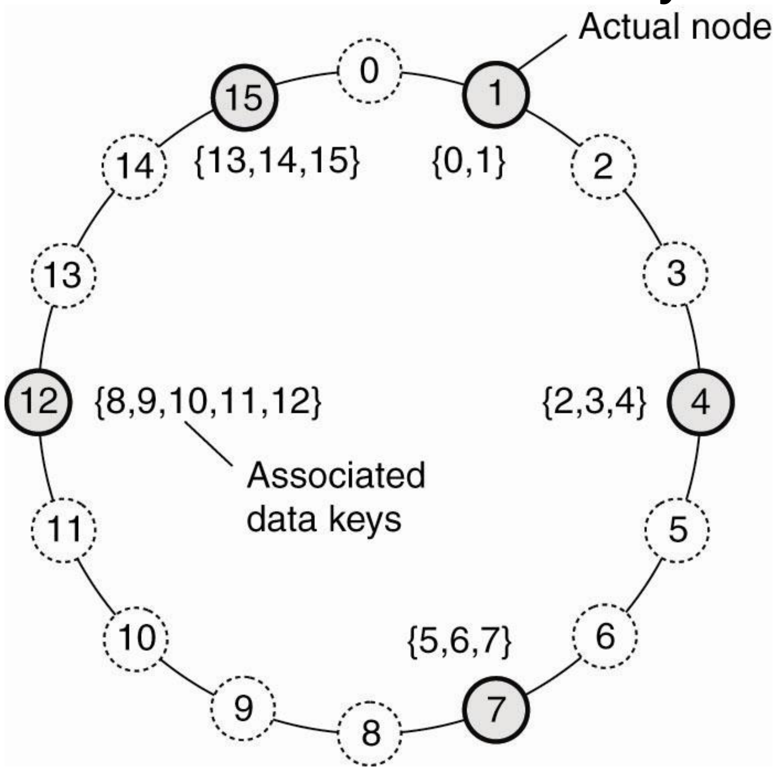
So Far

- Definitions and Goals
- Architectural Styles
- System Architectures
- Peer to Peer
- Edge Computing
- Blockchain basics

Structured P2P Systems

Basic idea

Organize the nodes in a structured overlay network such as a logical ring and make specific nodes responsible for services based only on their ID

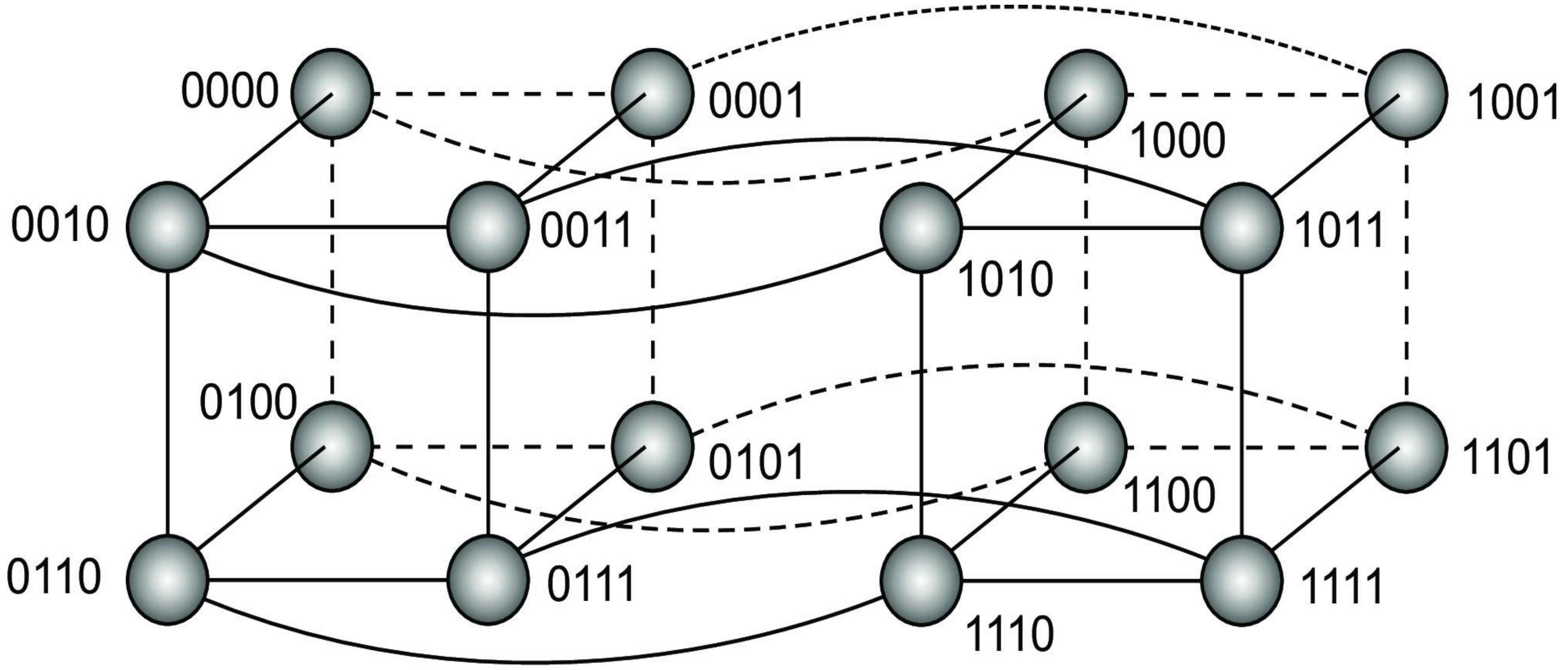


Note
The system provides an operation LOOKUP(key) that will efficiently route the lookup request to the associated node.

Structured P2P Systems

Other example

Organize nodes in a hypercube



Some structured P2P Algorithms

Apache Cassandra	• https://cassandra.apache.org/
BATON Overlay	• http://www.eecs.umich.edu./db/files/p661-jagadish.pdf
Mainline DHT	• Based on Kademlia, used by BitTorrent, http://www.bittorrent.org/beps/bep_0005.html
Content addressable network (CAN)	• http://www.eecs.berkeley.edu/~sylvia/papers/cans.pdf
Chord	• https://github.com/sit/dht/wiki
Koorde	• Derived from Chord, http://iptps03.cs.berkeley.edu/final-papers/koorde.ps
Kademlia	• http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html
Pastry	• http://freepastry.org/
P-Grid	• http://www.p-grid.com/
Riak	• https://riak.com/
Tapestry	• http://pdos.csail.mit.edu/~strib/docs/tapestry/tapestry_jsac03.pdf
TomP2P	• https://tomp2p.net/
Voldemort	• http://www.project-voldemort.com/voldemort/

Unstructured P2P Systems

Observation

Many unstructured P2P systems attempt to maintain a random overlay; two nodes are linked with probability p.

Basic principles

Each node is required to contact a randomly selected other node:

- Let each peer maintain a partial view of the network, consisting of c other nodes
- Each node P periodically selects a node Q from its partial view
- P and Q exchange information and exchange members from their respective partial views

Note

It turns out that, depending on the exchange, randomness, but also robustness can be maintained.

Unstructured P2P Systems

Observation

We can no longer look up information deterministically; we must resort to searching.

Model:

- Assume N nodes and that each data item is replicated across r randomly chosen nodes.

Two basic ideas:

Random Walk

- Randomly select a neighbor v. If v has the answer, it replies, otherwise v randomly selects one of its neighbors.
  - Variation: parallel random walk. Works well with replicated data.

Analysis

- $P[k]$ probability that item is found after k attempts:

$$P[k] = \frac{r}{N}\left(1 - \frac{r}{N}\right)^{k-1}$$

S ("search size") is expected number of nodes that need to be probed:

$$S = \sum_{k=1}^{N} k \times P[k] = \sum_{k=1}^{N} k \times \frac{r}{N}\left(1 - \frac{r}{N}\right)^{k-1} \approx \frac{N}{r}$$

for $1 \ll r \leq N$

Flooding

- Node u sends a lookup query to all of its neighbors. A neighbor responds, or forwards (floods) the request.
- Variations:
  - Limited flooding (maximal number of forwarding)
  - Probabilistic flooding (flood only with a certain probability).

Analysis:

- Limited Flood to d randomly chosen neighbors
- After k steps, some

$$R(k) = d \times (d-1)^{k-1}$$

will have been reached (assuming k is small).

- With fraction $\frac{r}{N}$ nodes having data, if

$$\frac{r}{N} \times R(k) \geq 1$$

- we will have found the data item

Comparing Random Walk vs. Flooding

Random Walk

If $\frac{r}{N} = 0.001$, then $S \approx 1000$

Flooding

With $d = 10, k = 4$, we contact 7290 nodes.

Random walks are more communication efficient but might take longer before they find the result.

Unstructured P2P Systems

Observation

Many unstructured P2P systems attempt to maintain a random overlay; two nodes are linked with probability p.

Basic principles

Each node is required to contact a randomly selected other node:

- Let each peer maintain a partial view of the network, consisting of c other nodes
- Each node P periodically selects a node Q from its partial view
- P and Q exchange information and exchange members from their respective partial views

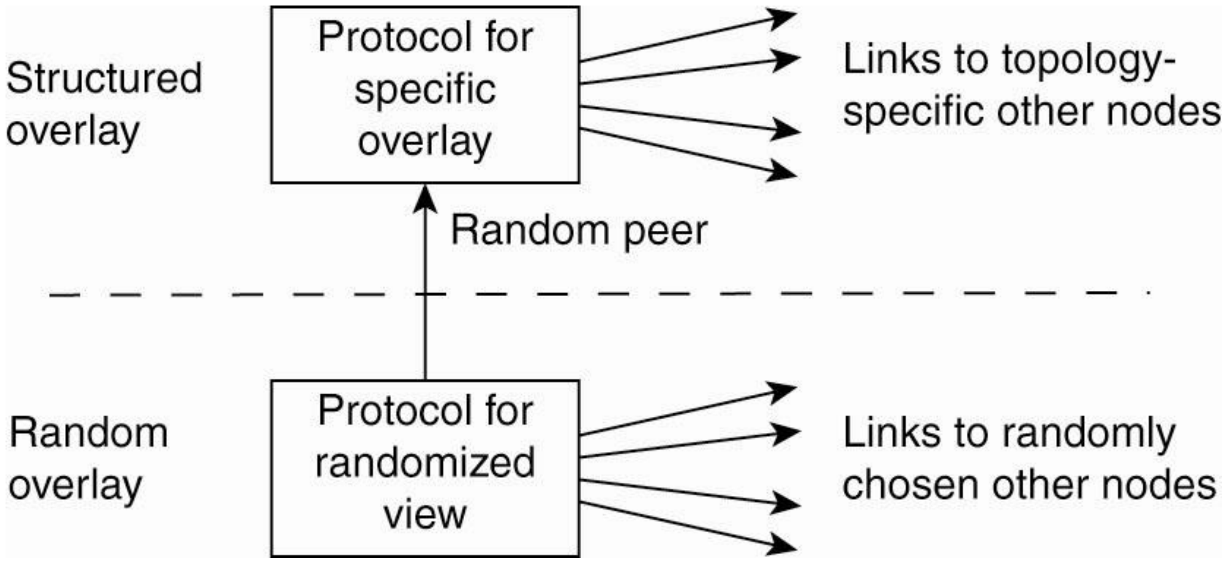
Note

It turns out that, depending on the exchange, randomness, but also robustness can be maintained.

Topology Management of Overlay Networks

Basic idea

Distinguish two layers: (1) maintain random partial views in lowest layer; (2) be selective on who you keep in higher-layer partial view.


Structured overlay
Protocol for specific overlay
Links to topology-specific other nodes
Random peer
Random overlay
Protocol for randomized view
Links to randomly chosen other nodes

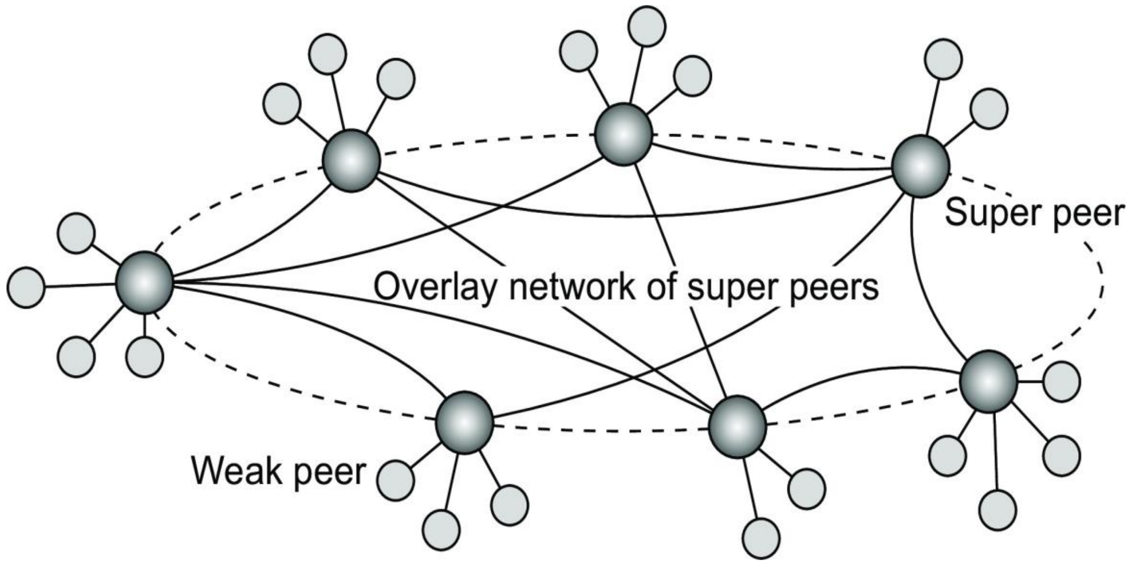
Note

Lower layer feeds upper layer with random nodes; upper layer is selective when it comes to keeping references.

Superpeers

Observation

Sometimes it helps to select a few nodes to do specific work: superpeer

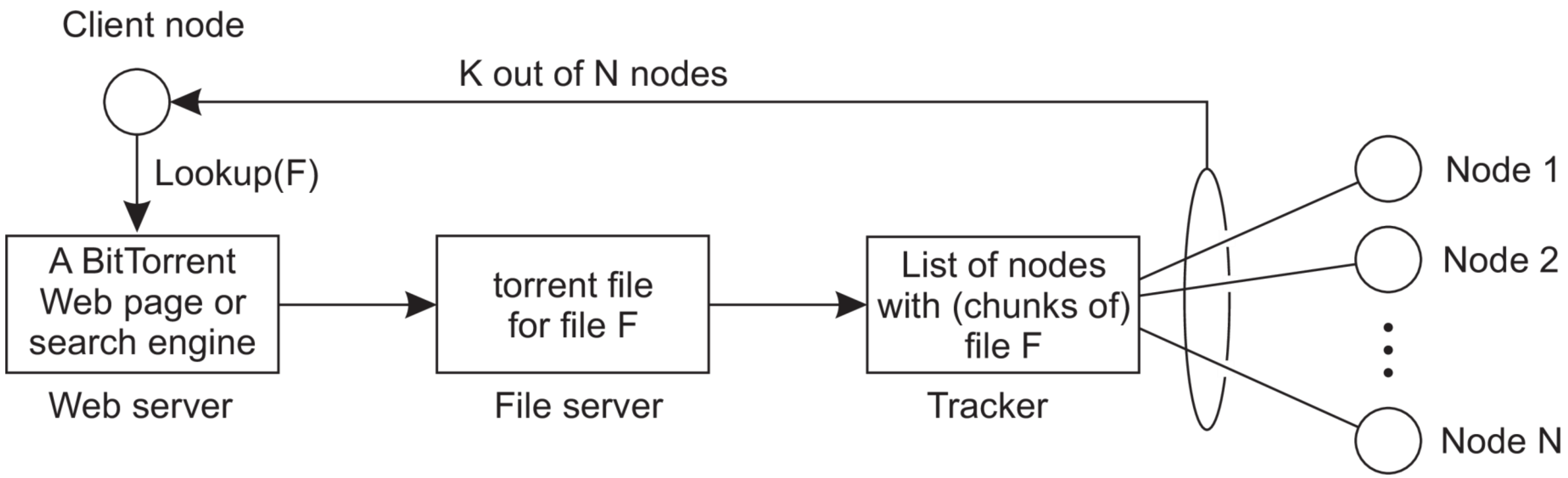


Examples

- Peers maintaining an index (for search)
- Peers monitoring the state of the network
- Peers able to setup connections

Collaboration: BitTorrent

- Principle: search for a file *F*
  - Lookup file at a global directory ⇒ returns a torrent file
  - Torrent file contains reference to tracker: a server keeping an accurate account of active nodes that have (chunks of) *F*.
  - *P* can join swarm, get a chunk for free, and then trade a copy of that chunk for another one with a peer *Q* also in the swarm



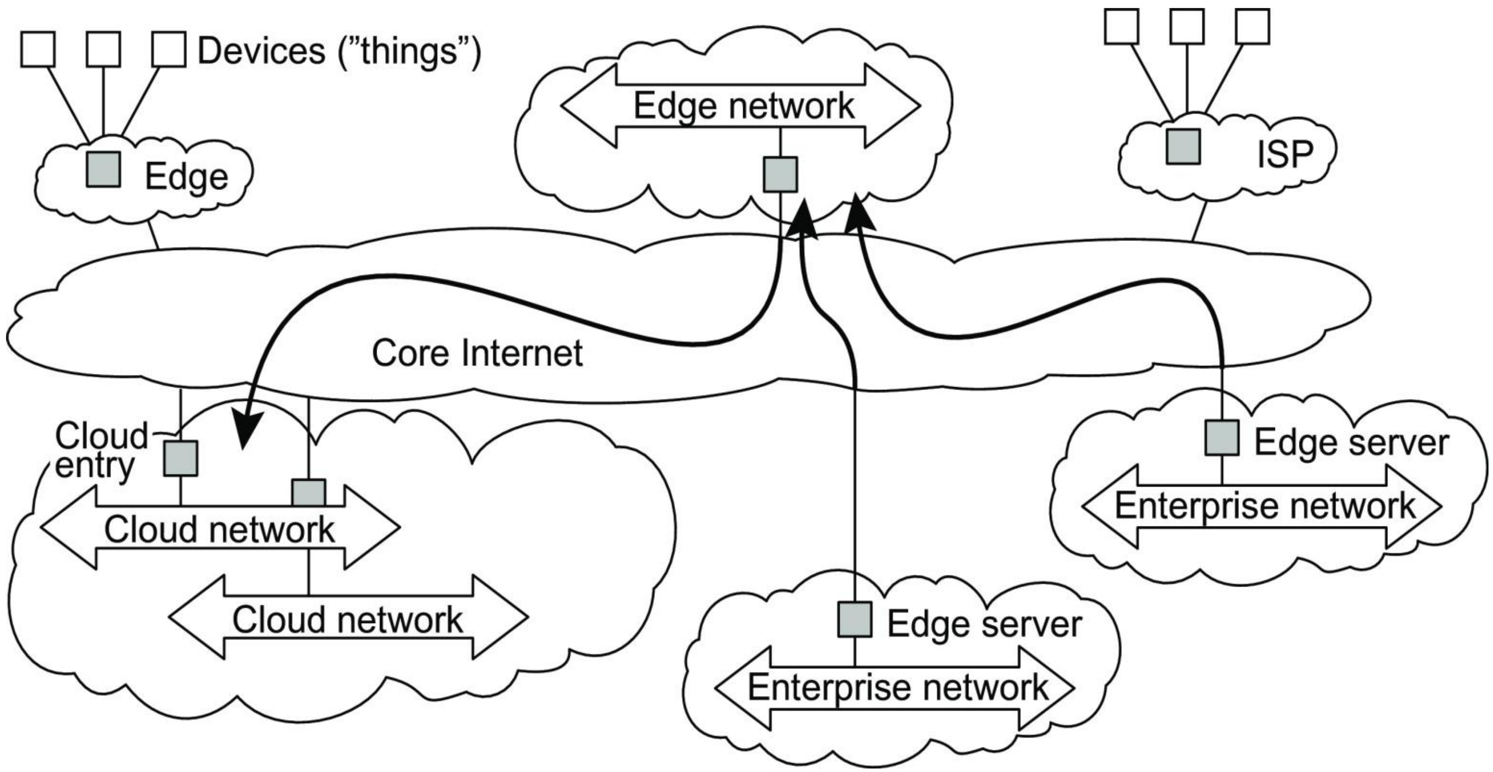
So Far

- Definitions and Goals
- Architectural Styles
- System Architectures
- Peer to Peer
- Edge Computing
- Blockchain basics

Edge-server architecture

Essence

Systems deployed on the Internet where servers are placed at the edge of the network: the boundary between enterprise networks and the actual Internet.



Why edge?

Latency and bandwidth

- Especially important for certain real-time applications, such as augmented/virtual reality applications.
- Many underestimate the latency and bandwidth to the cloud.

Reliability

- Connection to the cloud is often assumed to be unreliable, which may be a false assumption.
- May be critical situations in which extremely high connectivity guarantees are needed.

Security and privacy

- The implicit assumption is often that when assets are nearby, they can be made better protected.
- Practice shows this is generally false.
- However, securely handling data operations in the cloud may be trickier than within your own organization.

Edge orchestration

Managing resources at the edge may be trickier than in the cloud

- Resource allocation: we need to guarantee the availability of the resources required to perform a service.
- Service placement: we need to decide when and where to place a service. This is notably relevant for mobile applications.
- Edge selection: we need to decide which edge infrastructure should be used when a service needs to be offered. The closest one may not be the best one.

A little aside

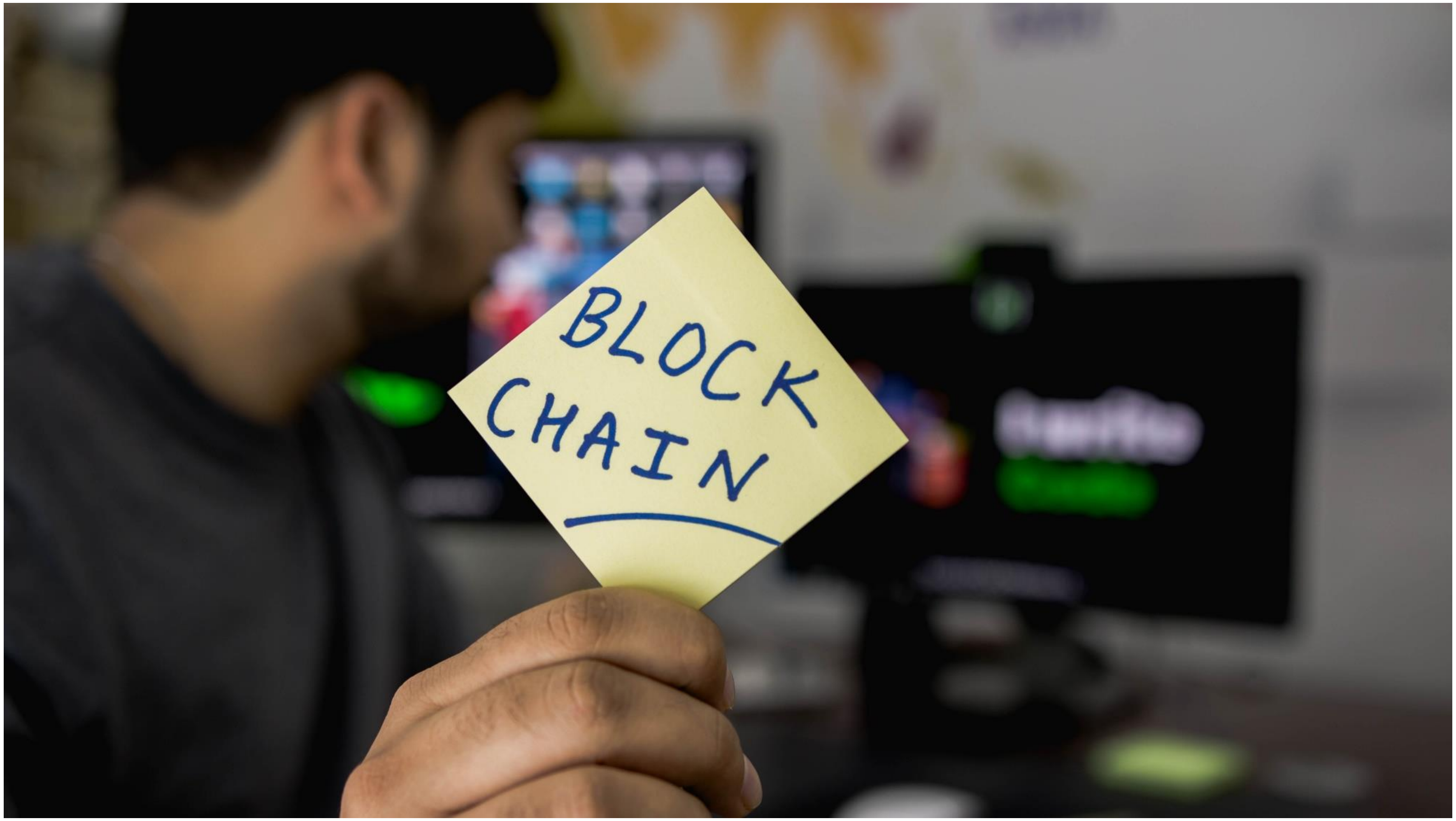
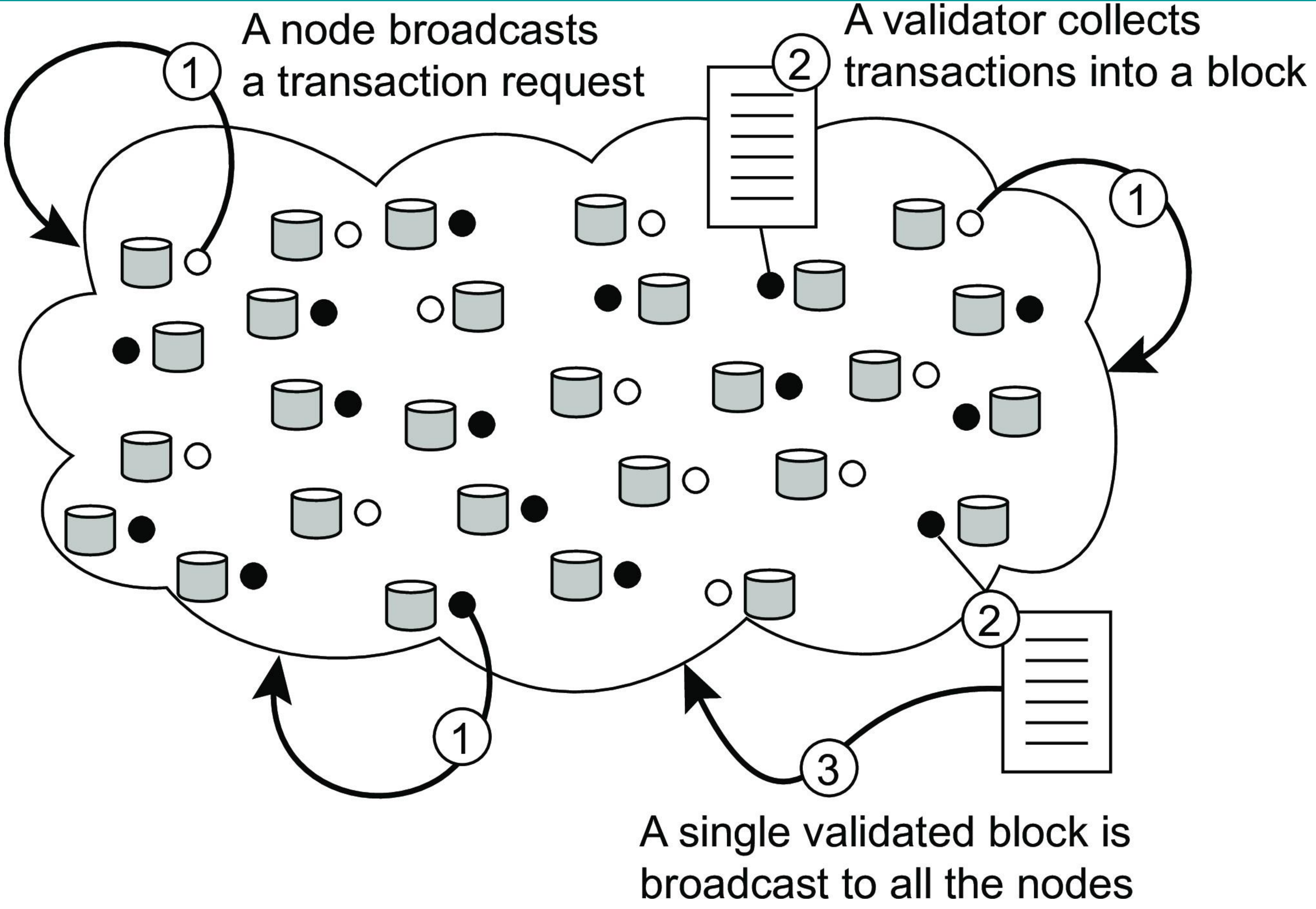
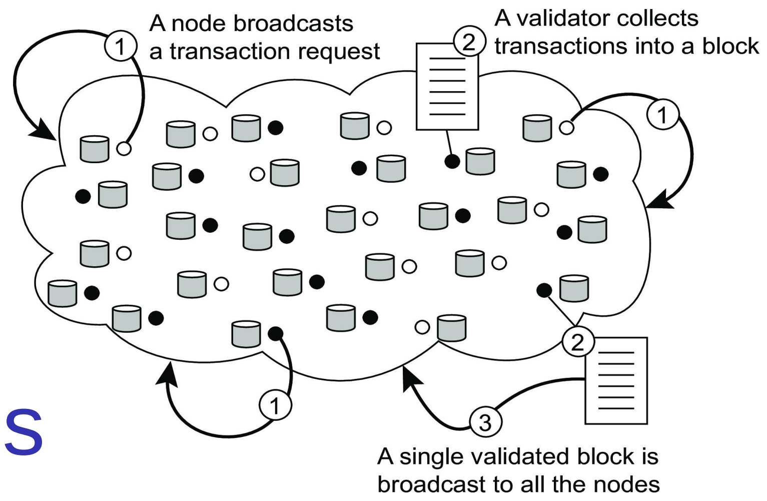


Photo by Hitesh Choudhary on Unsplash

Blockchain Basics

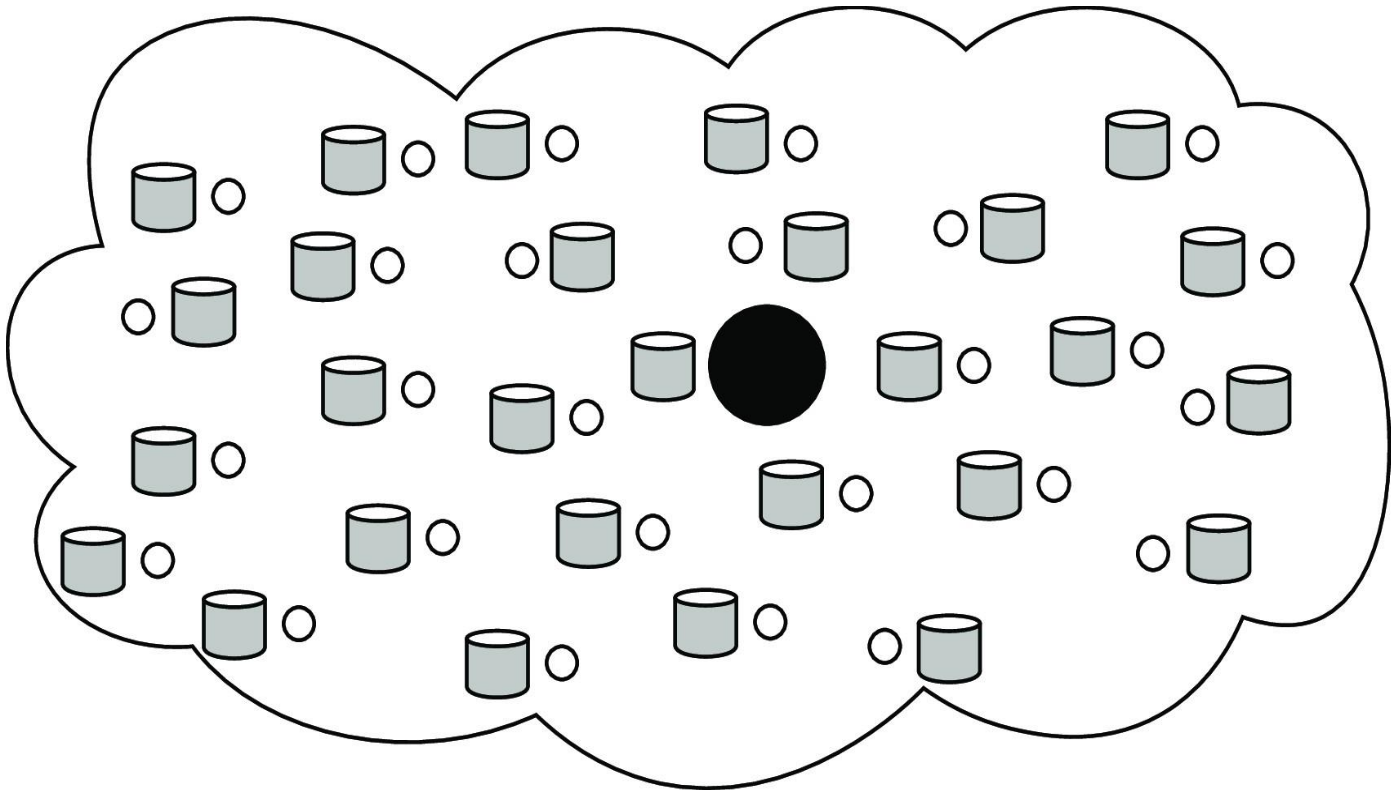

1
A node broadcasts a transaction request
2
A validator collects transactions into a block
1
1
2
3
A single validated block is broadcast to all the nodes

Blockchain Basics


1
A node broadcasts a transaction request
2
A validator collects transactions into a block
1
1
2
3
A single validated block is broadcast to all the nodes

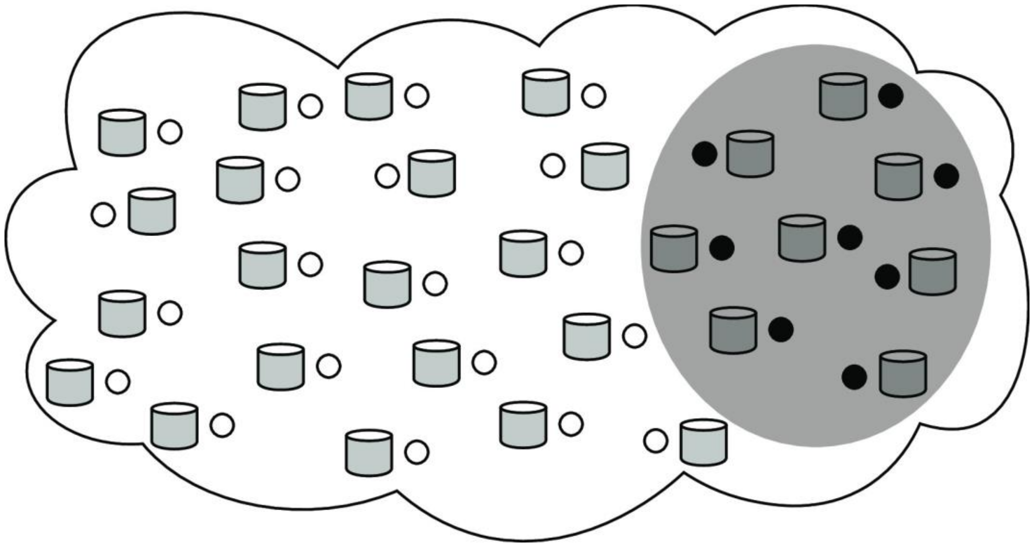
- Observations
  - Blocks are organized into an unforgeable append-only chain
  - Each block in the blockchain is immutable ⇒ massive replication
  - The real snag lies in who is allowed to append a block to a chain

Appending a block: Centralized



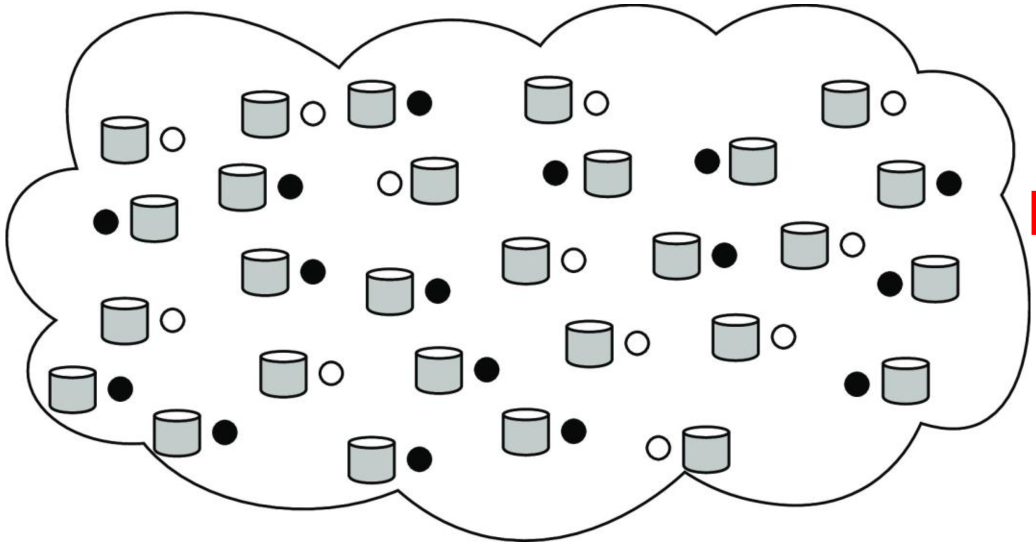
- A single entity decides on which validator can go ahead and append a block.
- Does not fit the design goals of blockchains.

Appending a block: Distributed



- A selected, relatively small group of servers jointly reach consensus on which validator can go ahead.
- None of the servers needs to be trusted, if $\approx \frac{2}{3}$ behave according to specification.
- In practice, only 10s of servers can be accommodated.

Appending a block: Distributed



Permissionless

- Participants collectively engage in a leader election. Only the elected leader is allowed to append a block of validated transactions.
- Large-scale, decentralized leader election that is fair, robust, secure, etc. is not trivial.

Conclusion

- Definitions and Goals
- Architectural Styles
- System Architectures
- Peer to Peer
- Edge Computing
- Blockchain basics