

Client/Server and Peer to Peer

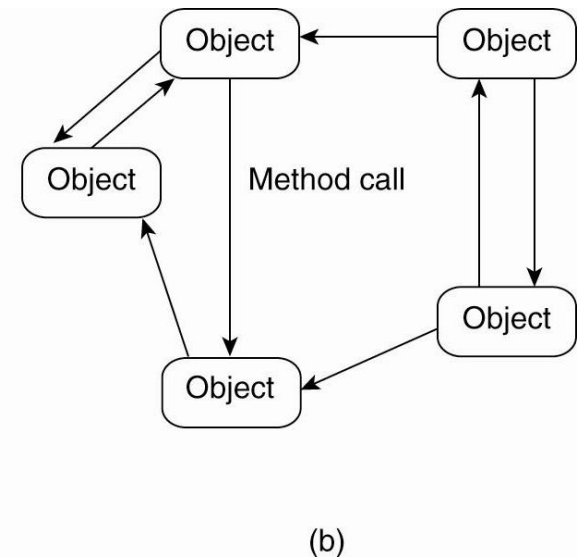
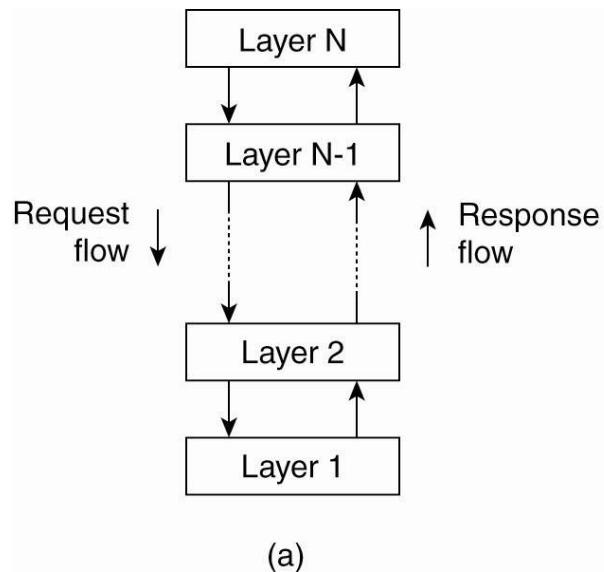
28 October 2026
Lecture 2

Most Slide Credits: Maarten van Steen

Architectural Styles

Basic Idea

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



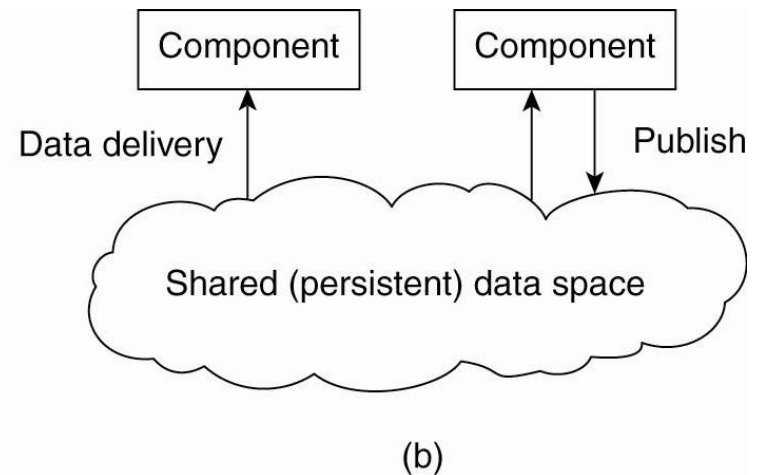
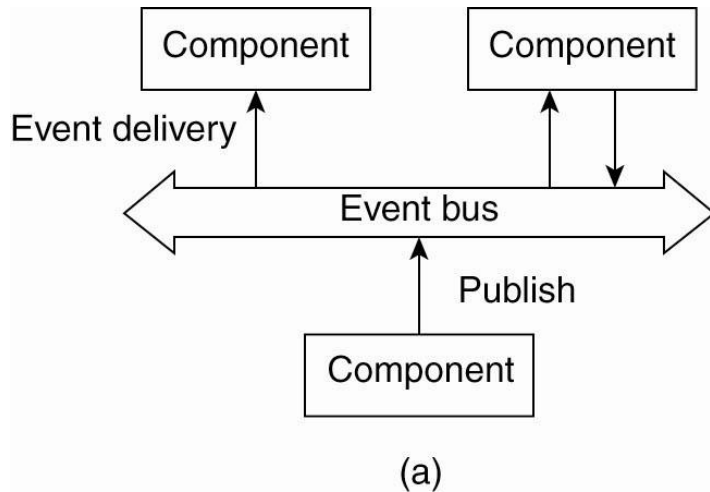
(a) Layered style is used for client-server system

(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**]

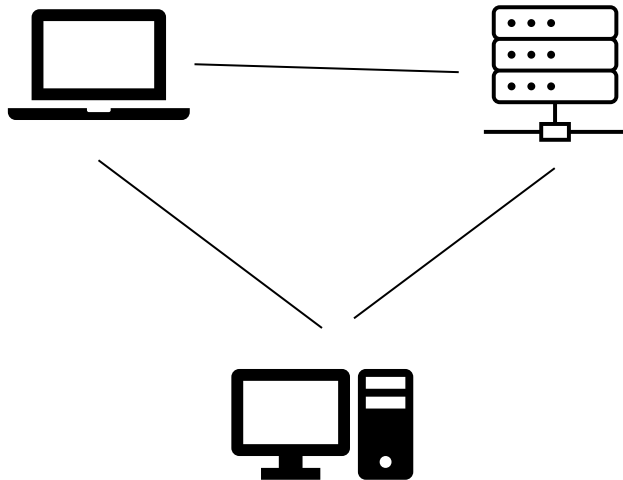
So Far

- Architectural Styles
- System Architectures
- Peer to Peer
- Edge Computing

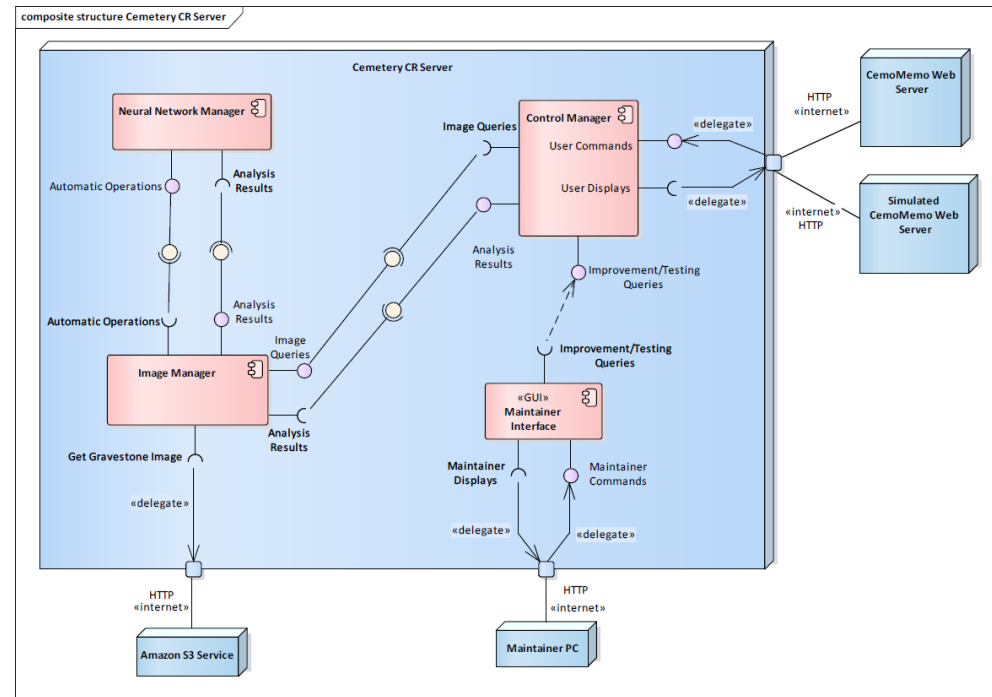
Discern

Physical architecture

- How many computers
- Where they are located



Logical architecture & Deployment

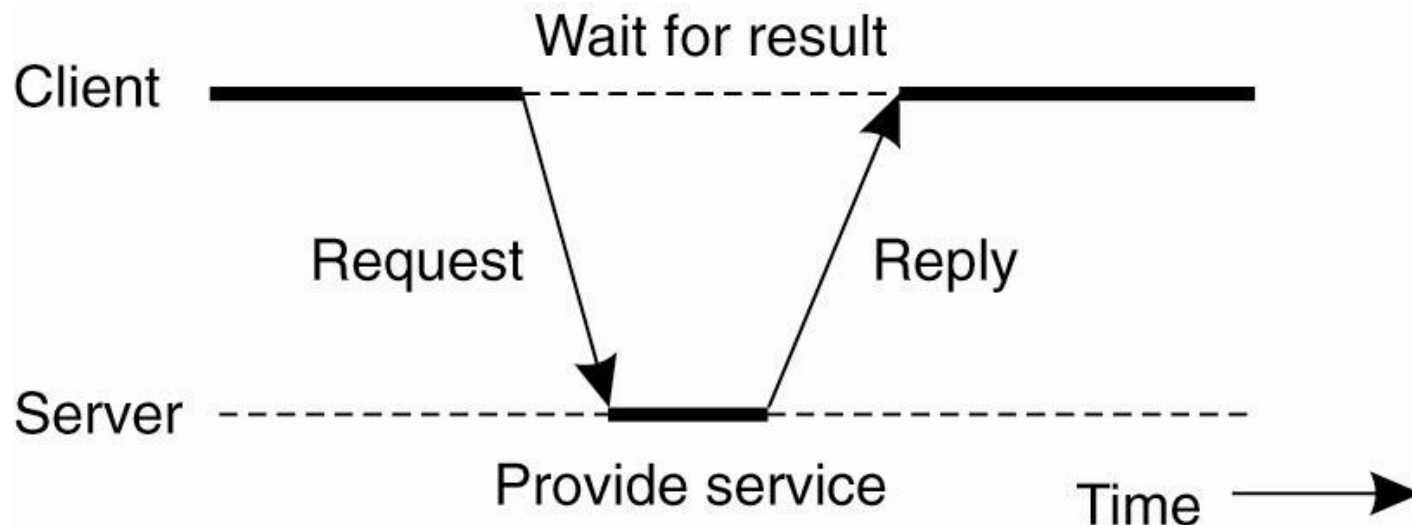


Phys: Centralized Architectures

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

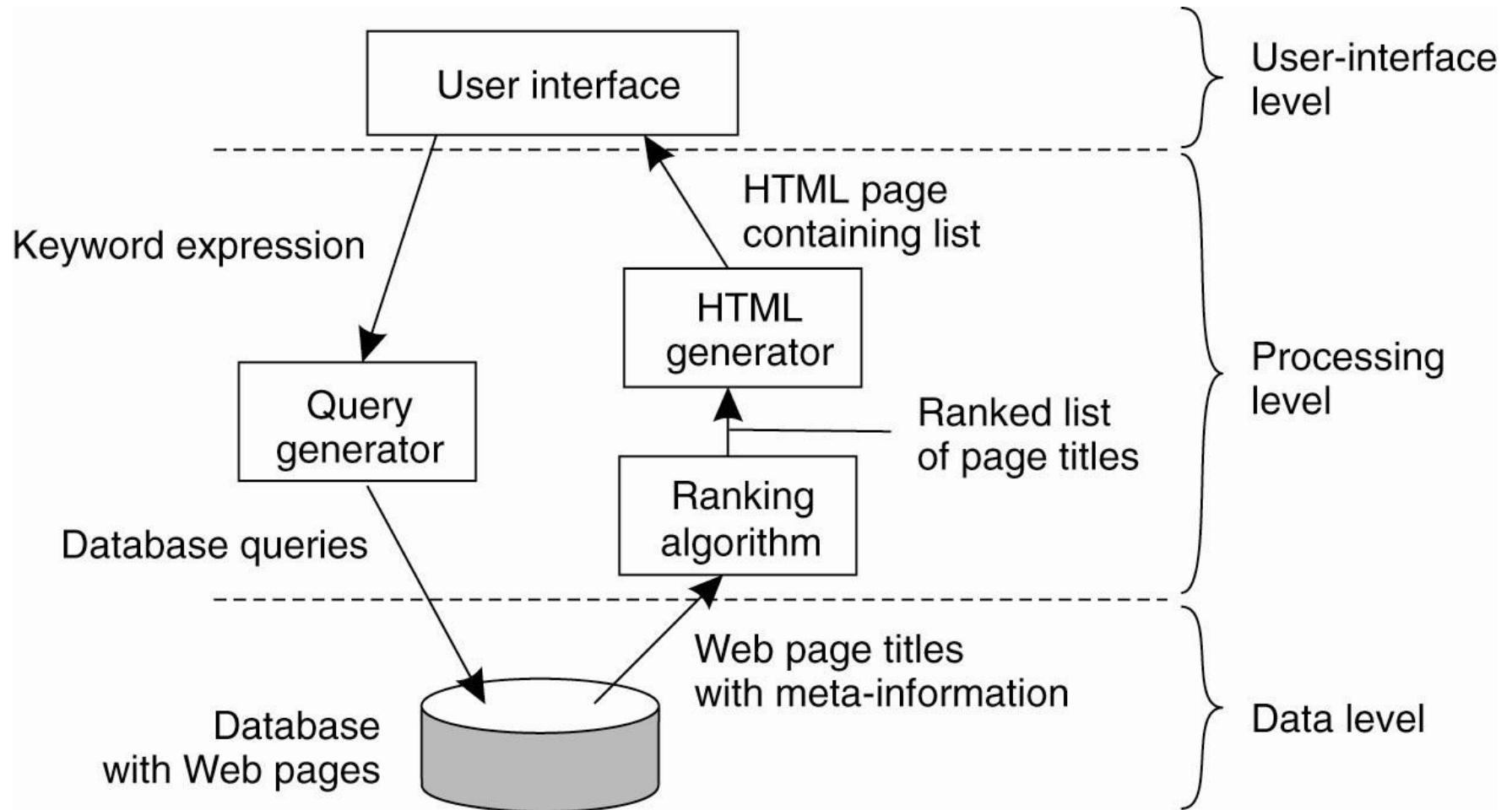
Traditional three-layered view

- 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



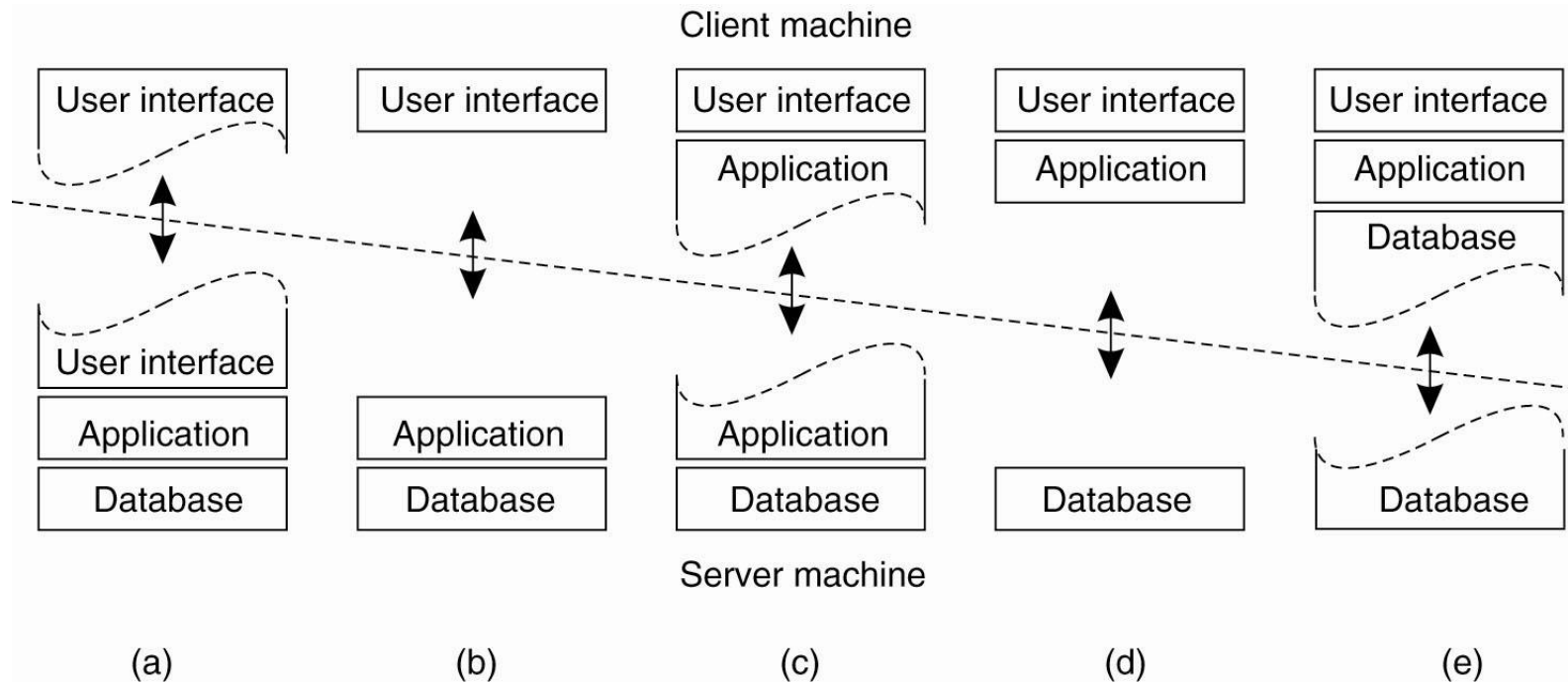
Dep: Multi-Tiered Architectures

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)

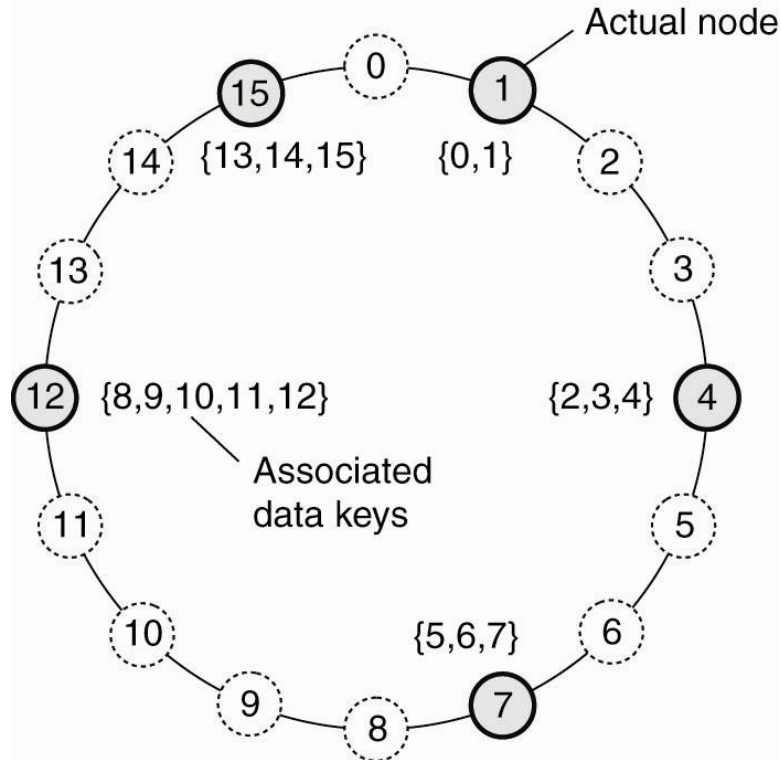
So Far

- Architectural Styles
- System Architectures
- Peer to Peer
- Edge Computing

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



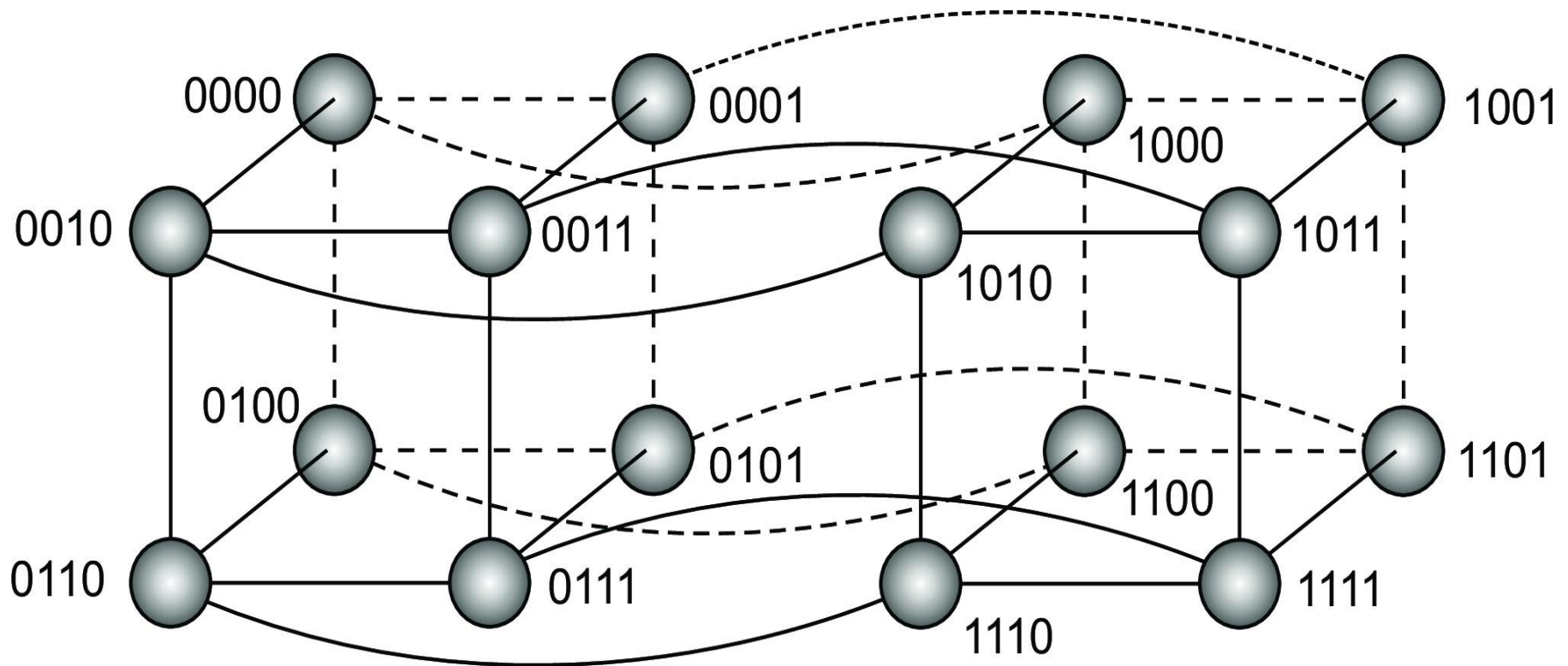
Note

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/~srib/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:

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

$$\text{for } 1 \ll r \leq N$$

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
-

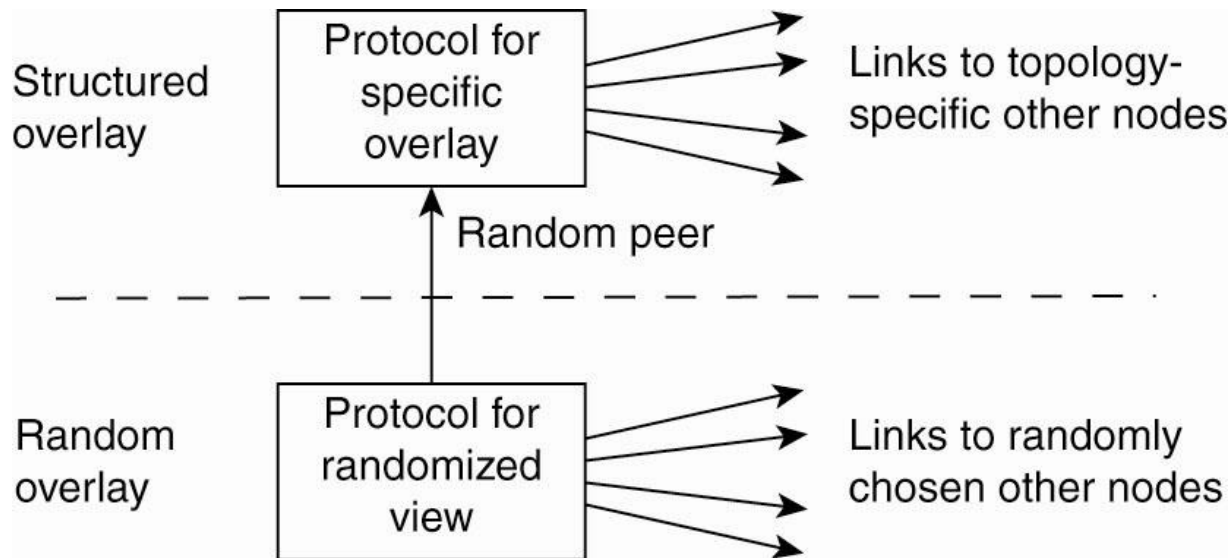
Note

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.



Note

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

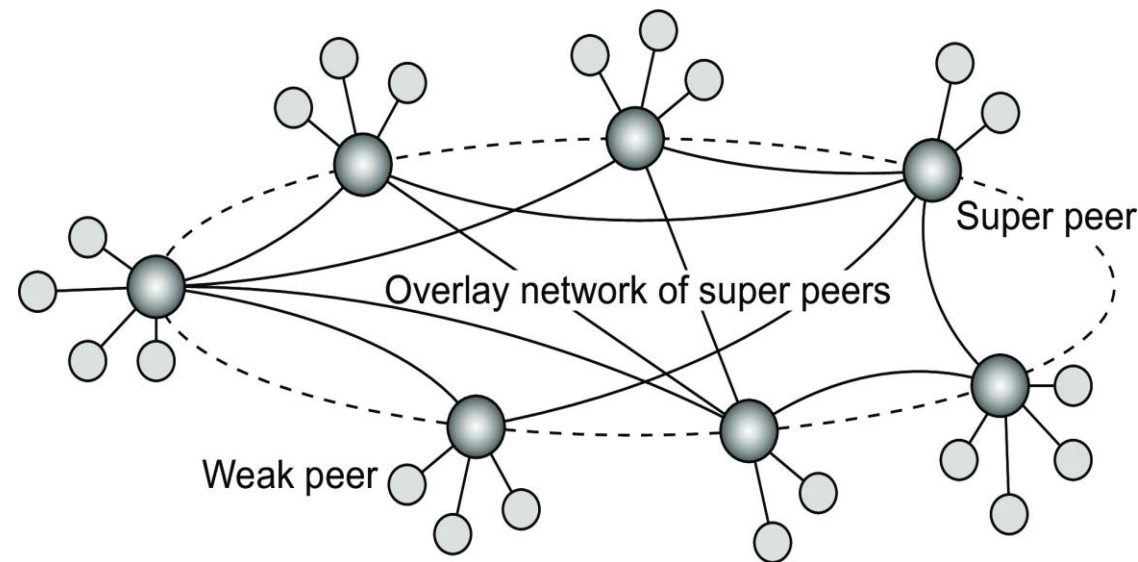
Superpeers

Observation

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

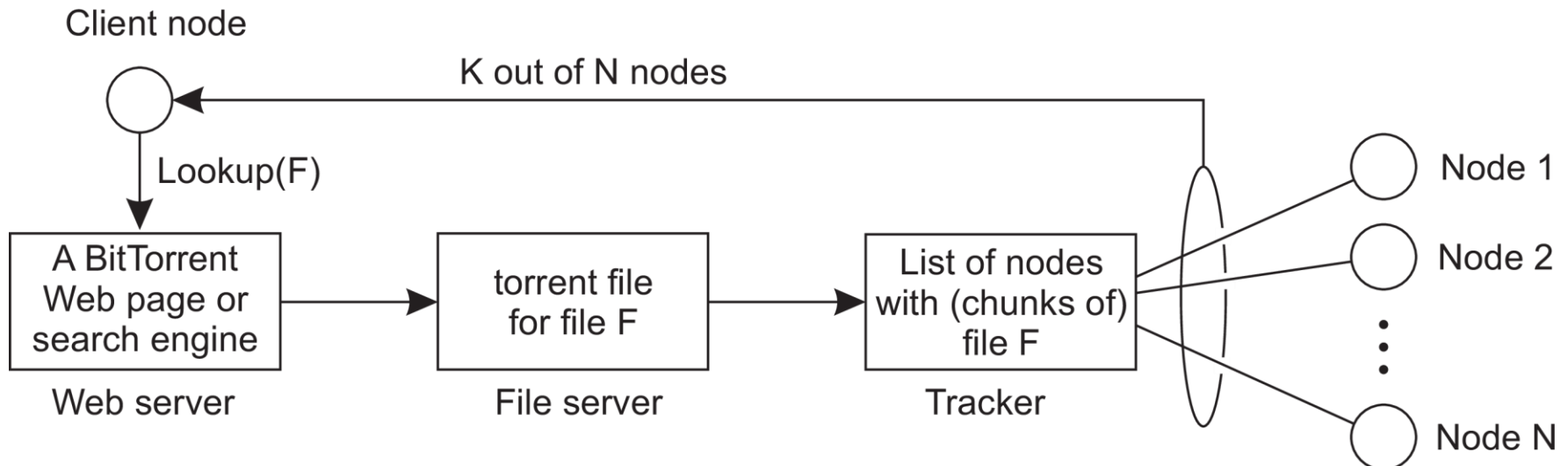
Examples

- 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 $\Rightarrow$ 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



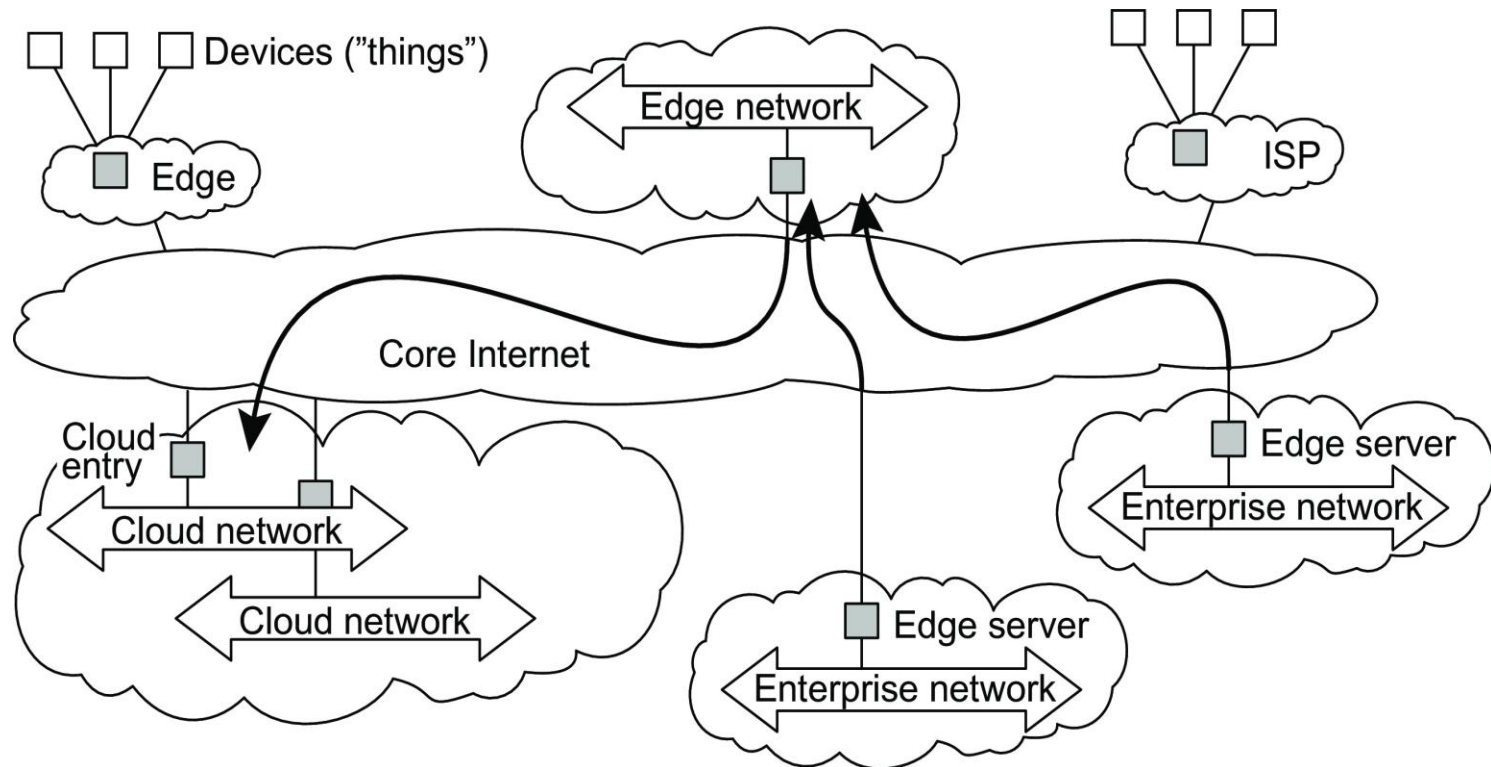
So Far

- Architectural Styles
- System Architectures
- Peer to Peer
- **Edge Computing**

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.

Conclusion

- Architectural Styles
- System Architectures
- Peer to Peer
- Edge Computing