Distributed Systems Architectures

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

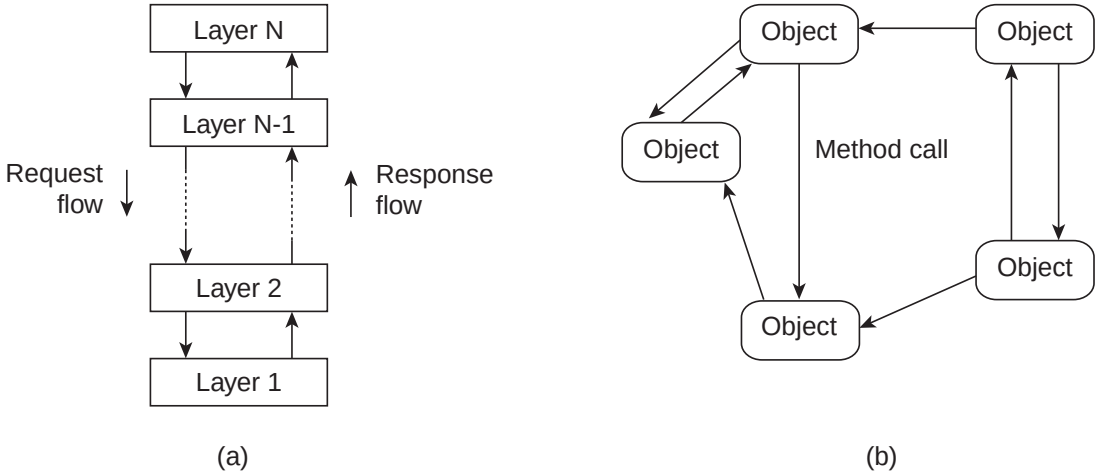
Based on slides by Maarten Van Steen

Basic Idea

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

Basic Idea

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



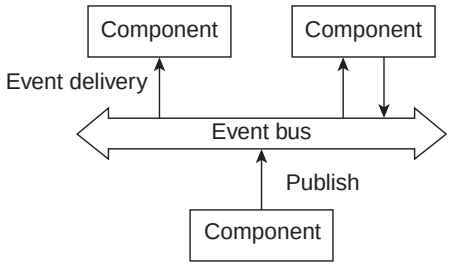
(a) Layered style is used for client-server system
(b) Object-based style for distributed object systems

Architectural Styles

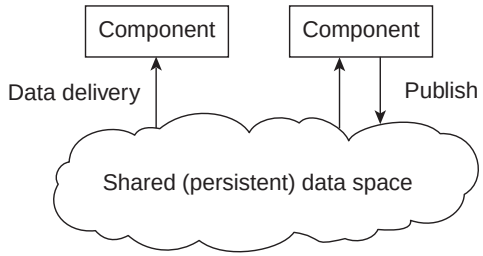
- Decoupling processes in space (anonymous) and also time (asynchronous) has led to alternative styles.

Architectural Styles

- Decoupling processes in space (anonymous) and also time (asynchronous) has led to alternative styles.



(a)



(b)

(a) Publish/subscribe: decoupled in space
(b) Shared dataspace: decoupled in space and time

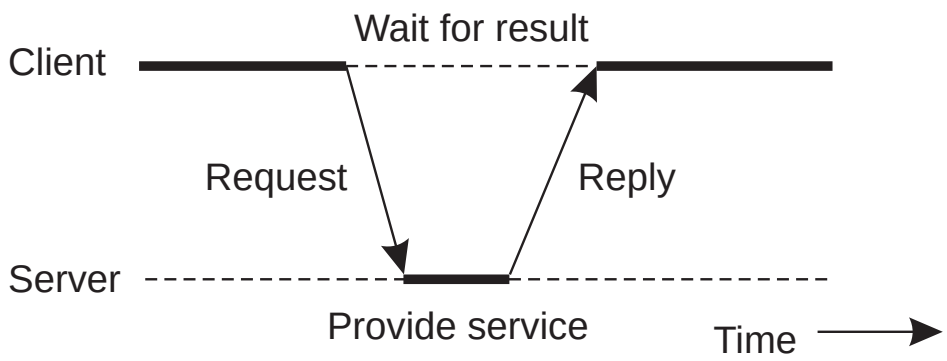
System Architectures

- Centralized architectures
- Decentralized architectures
- Hybrid architectures

Centralized Architectures

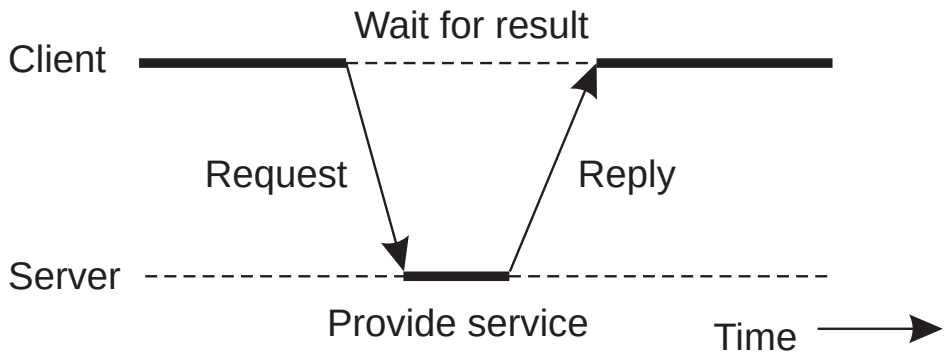
Centralized Architectures

- Basic client-server model



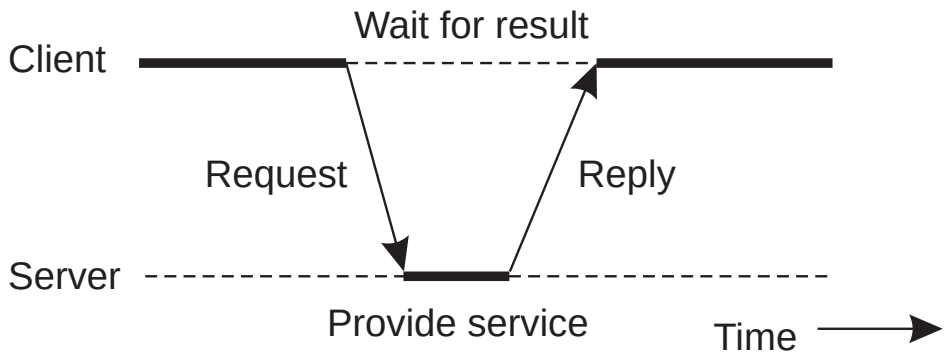
Centralized Architectures

- Basic client-server model
- Characteristics:
  - There are processes offering services: (servers)



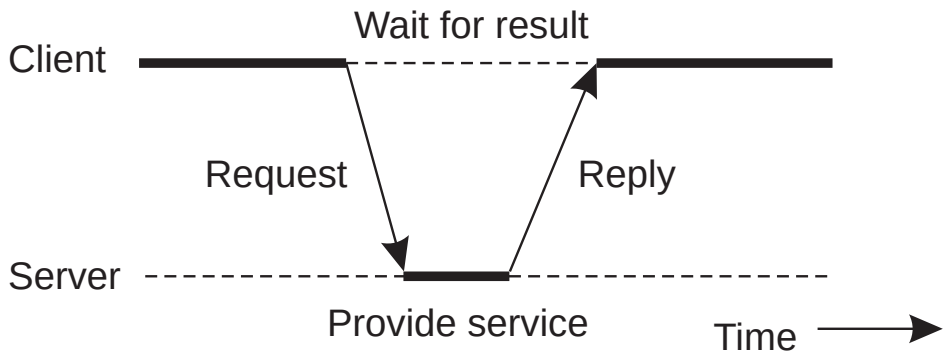
Centralized Architectures

- Basic client-server model
- Characteristics:
  - There are processes offering services: (servers)
  - There are processes that use services: (clients)



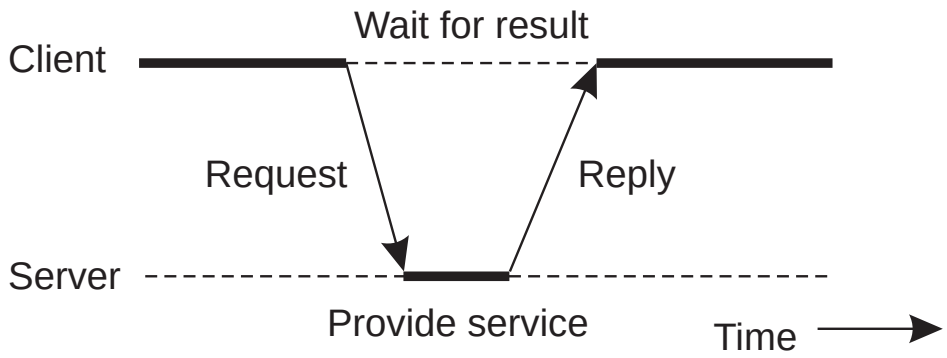
Centralized Architectures

- 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



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 w.r.t to using services



Application Layering (1/2)

- Traditional three-layered view:

Application Layering (1/2)

- Traditional three-layered view:
  - User-interface layer: contains units for an application's user interface.

Application Layering (1/2)

- 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.

Application Layering (1/2)

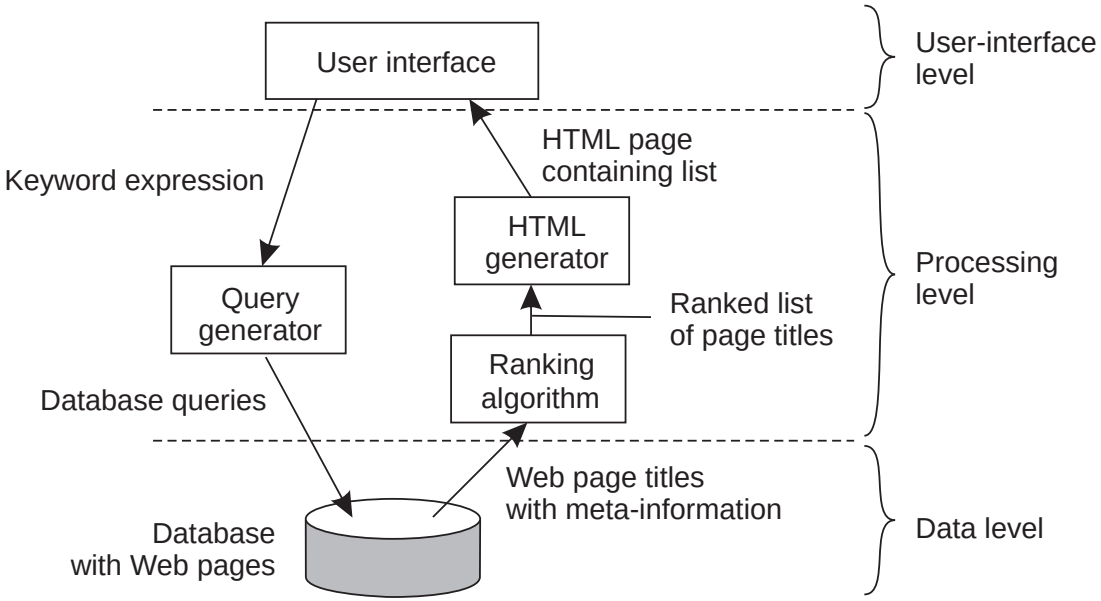
- Traditional three-layered view:
  - User-interface layer: contains units for an application's user interface.
  - Processing layer: contains the functions of an application, i.e., without specific data.
  - Data layer: contains the data that a client wants to manipulate through the application components.

Application Layering (1/2)

- 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.

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

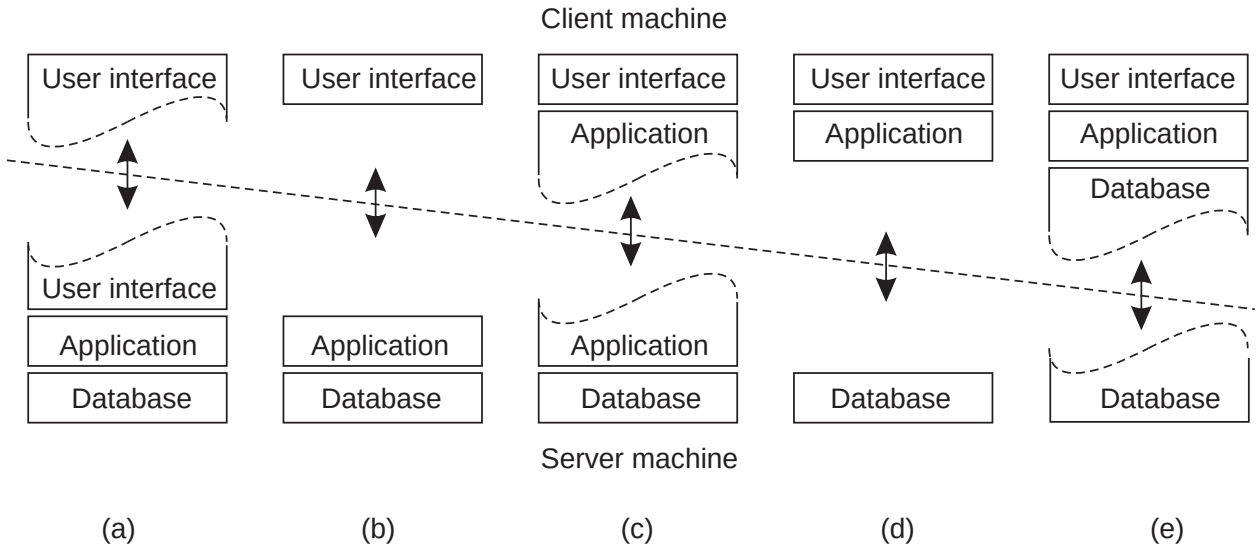
Application Layering (2/2)



Multi-Tiered Architectures

- Single-tiered: dumb terminal/mainframe configuration
- Two-tiered: client/single server configuration
- Three-tiered: each layer on separate machine

Traditional Two-Tiered Configurations



Decentralized Architectures

Decentralized Architectures

- Peer-to-Peer (P2P) systems

Decentralized Architectures

- Peer-to-Peer (P2P) systems
  - Structured P2P: nodes are organized following a specific distributed data structure

Decentralized Architectures

- Peer-to-Peer (P2P) systems
  - Structured P2P: nodes are organized following a specific distributed data structure
  - Unstructured P2P: nodes have randomly selected neighbors

Decentralized Architectures

- Peer-to-Peer (P2P) 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

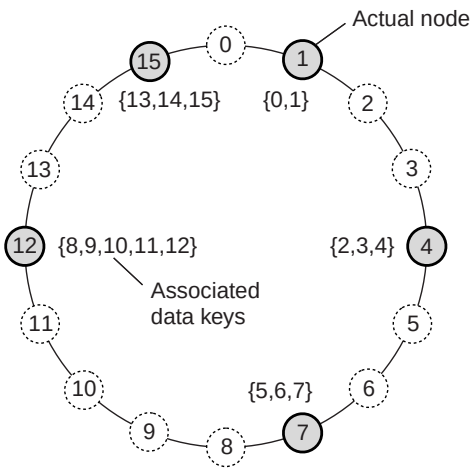
Decentralized Architectures

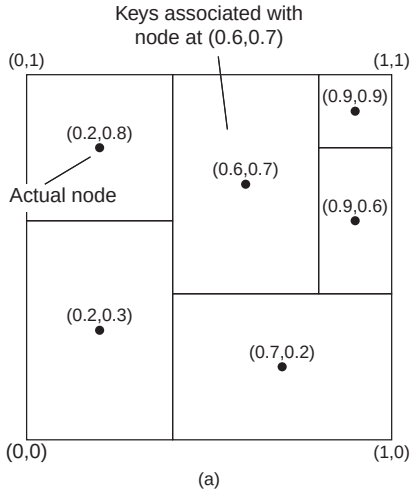
- Peer-to-Peer (P2P) systems
  - Structured P2P: nodes are organized following a specific distributed data structure
  - Unstructured P2P: nodes have randomly selected neighbors
  - Hybrid P2P: some nodes are appointed special functions in a well-organized fashion
- In all cases, we are dealing with overlay networks: data is routed over connections setup between the nodes.

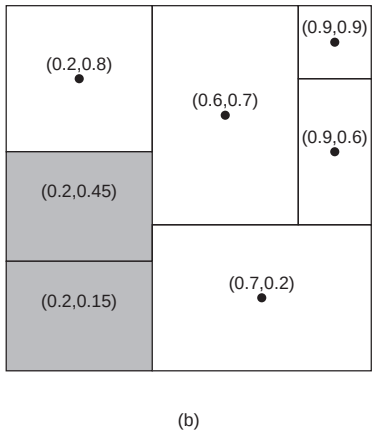
Structured P2P Systems

Structured P2P Systems

- Organize the nodes in a structured overlay network, e.g., logical ring or a *d*-dimensional space, and make specific nodes responsible for services based only on their ID.



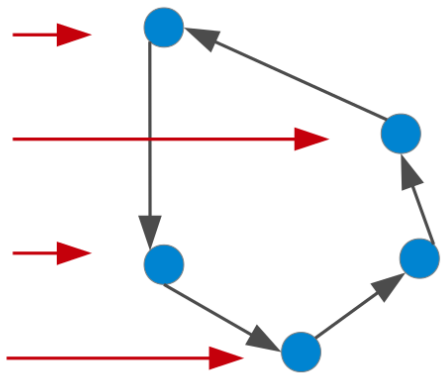




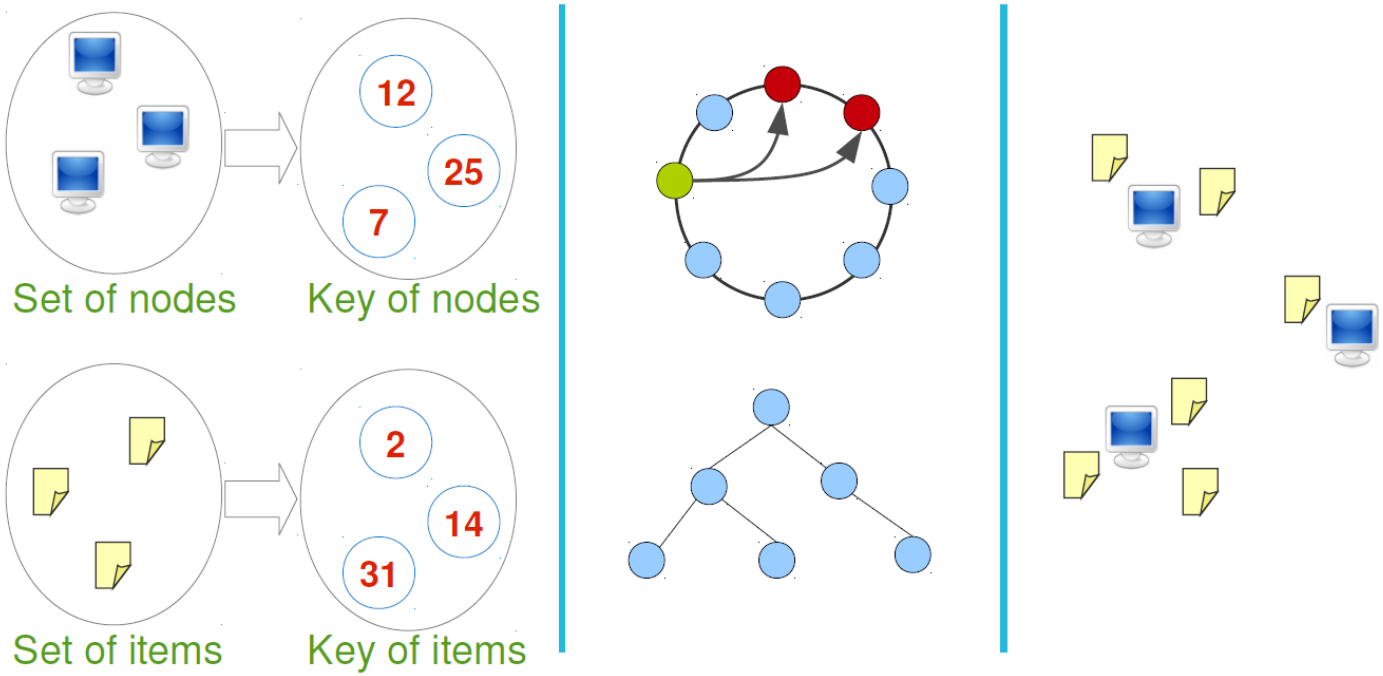
Distributed Hash Table

- An ordinary hash-table, which is distributed.

Key	Value
Fatemeh	Stockholm
Ali	California
Tallat	Islamabad
Cosmin	Bucharest
Seif	Stockholm
Amir	Tehran

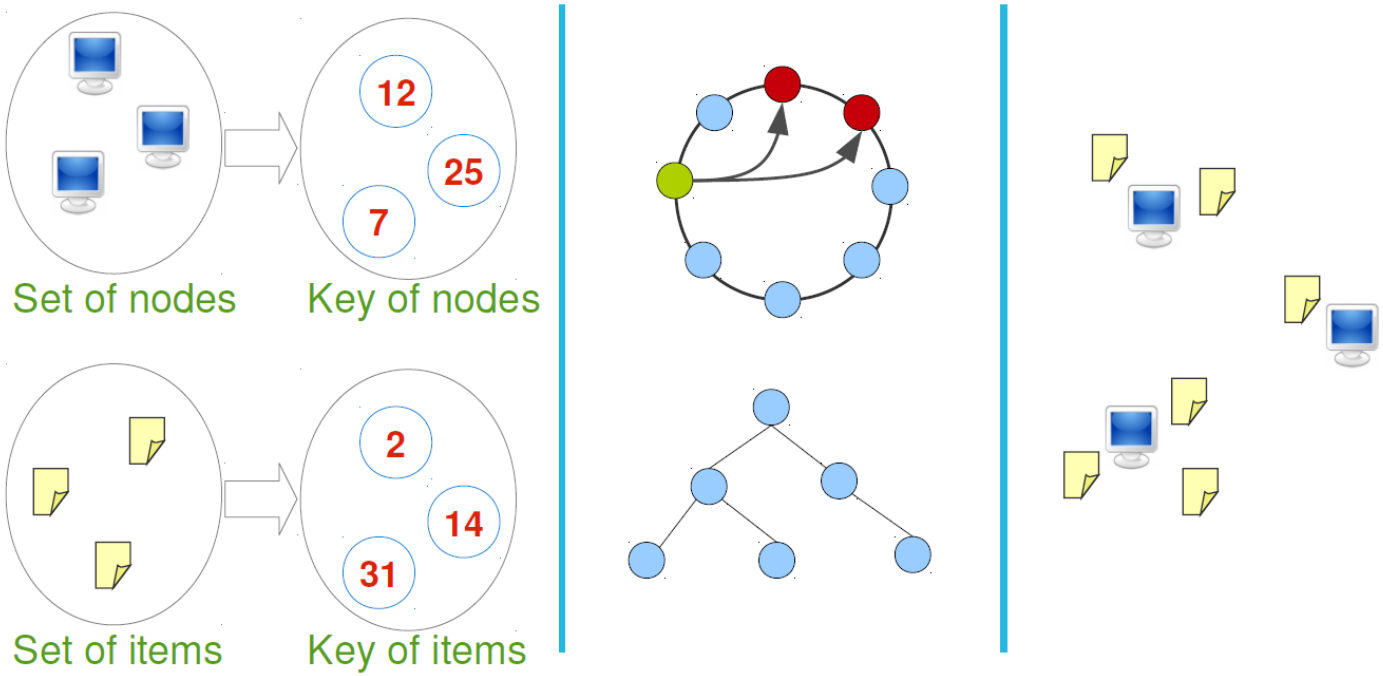


Steps to Build a DHT



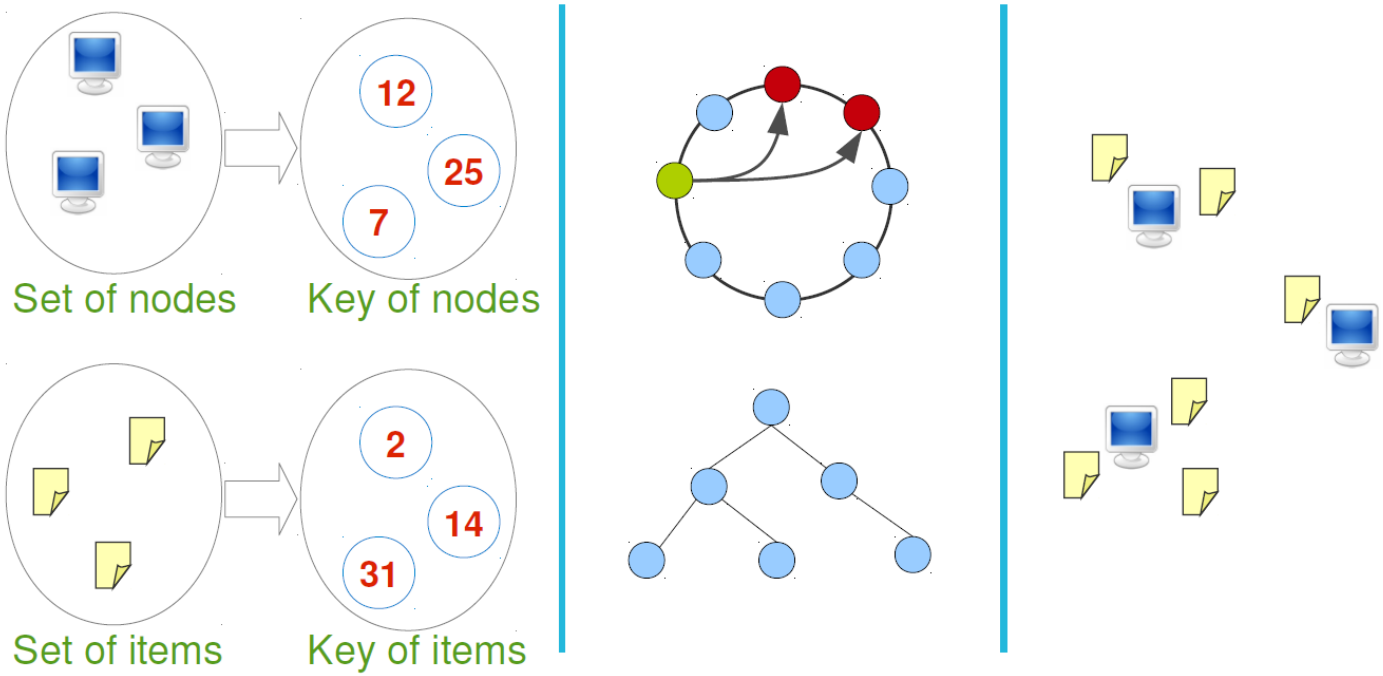
- Step 1: decide on common key space for nodes and values.

Steps to Build a DHT



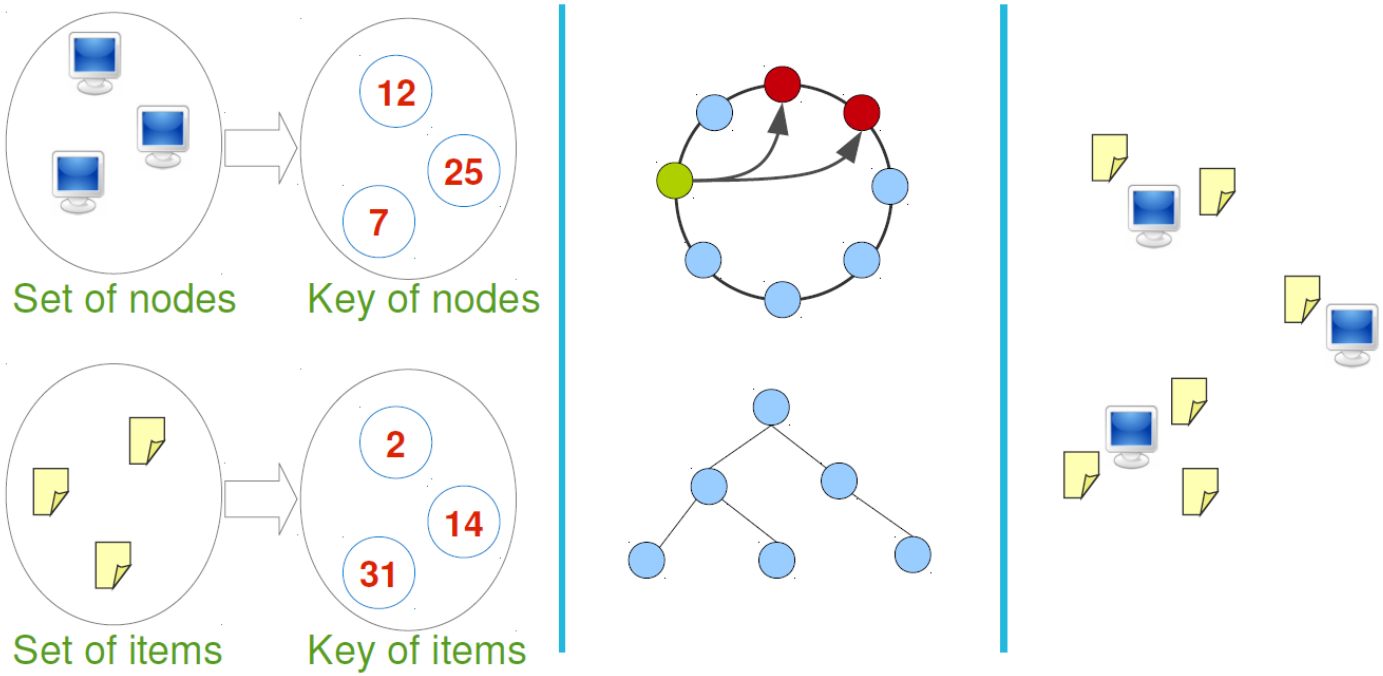
- Step 1: decide on common key space for nodes and values.
- Step 2: connect the nodes smartly.

Steps to Build a DHT



- Step 1: decide on common key space for nodes and values.
- Step 2: connect the nodes smartly.
- Step 3: make a strategy for assigning items to nodes.

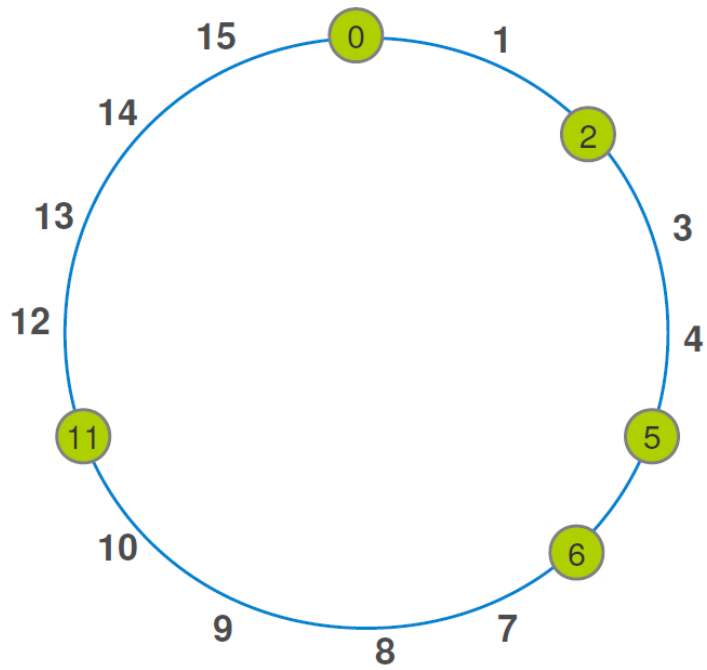
Steps to Build a DHT



- Step 1: decide on common key space for nodes and values.
- Step 2: connect the nodes smartly.
- Step 3: make a strategy for assigning items to nodes.
- Chord: an example of a DHT

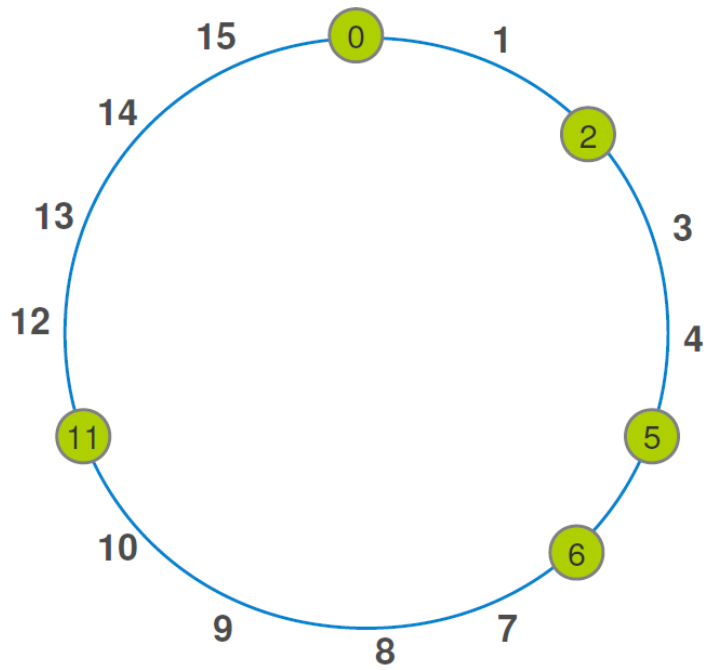
Construct Chord - Step 1

- Use a logical name space, called the id space, consisting of identifiers $\{0, 1, 2, \cdots, N-1\}$.



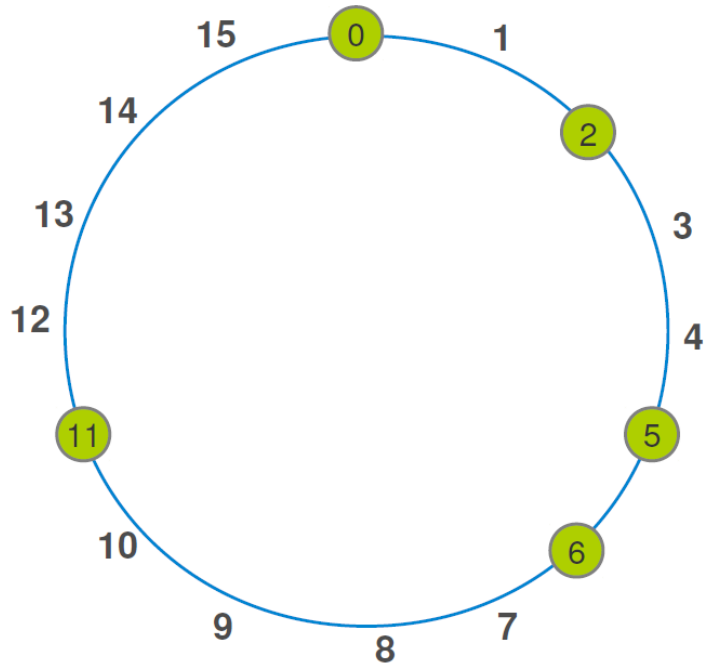
Construct Chord - Step 1

- Use a logical name space, called the id space, consisting of identifiers $\{0, 1, 2, \cdots, N-1\}$.
- Id space is a logical ring modulo N.



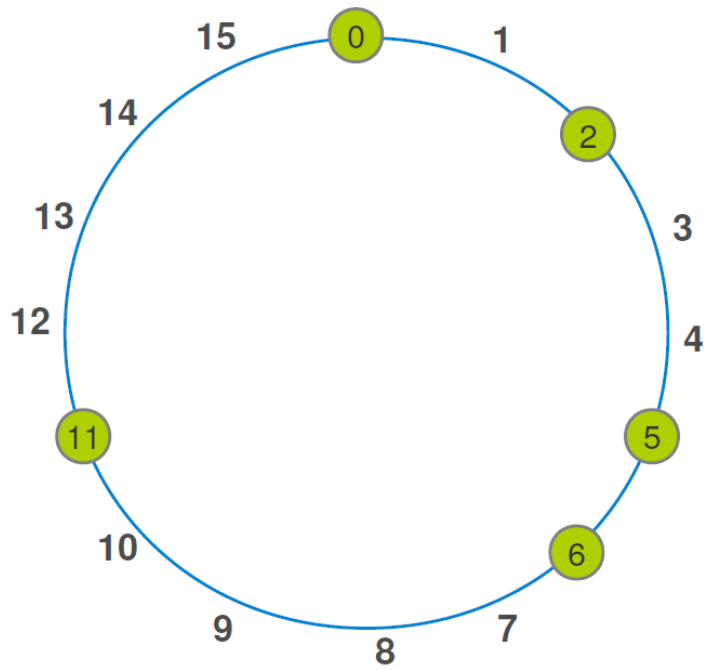
Construct Chord - Step 1

- Use a logical name space, called the id space, consisting of identifiers $\{0, 1, 2, \cdots, N-1\}$.
- Id space is a logical ring modulo N.
- Every node picks a random id though Hash H.



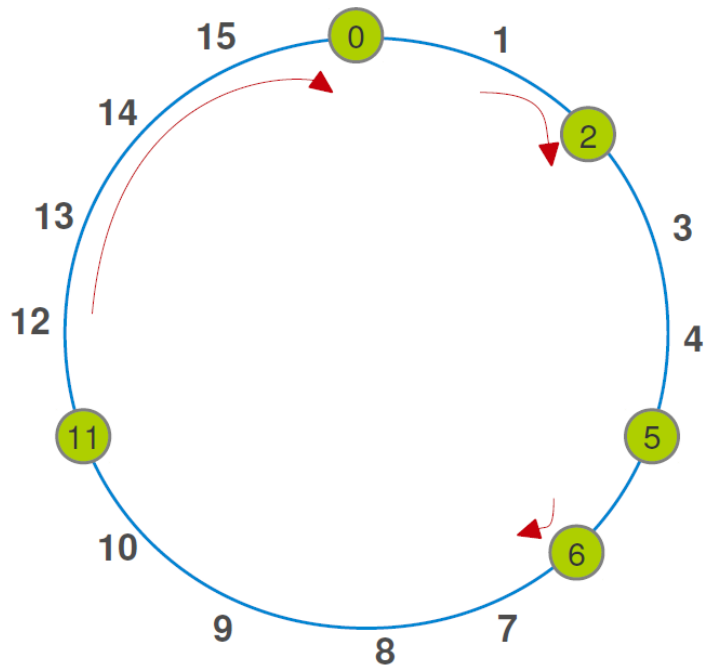
Construct Chord - Step 1

- Use a logical name space, called the id space, consisting of identifiers $\{0, 1, 2, \cdots, N-1\}$.
- Id space is a logical ring modulo N.
- Every node picks a random id though Hash H.
- Example:
  - Space $N = 16\{0, \cdots, 15\}$
  - Five nodes a, b, c, d, e.
  - $H(a) = 6$
  - $H(b) = 5$
  - $H(c) = 0$
  - $H(d) = 11$
  - $H(e) = 2$



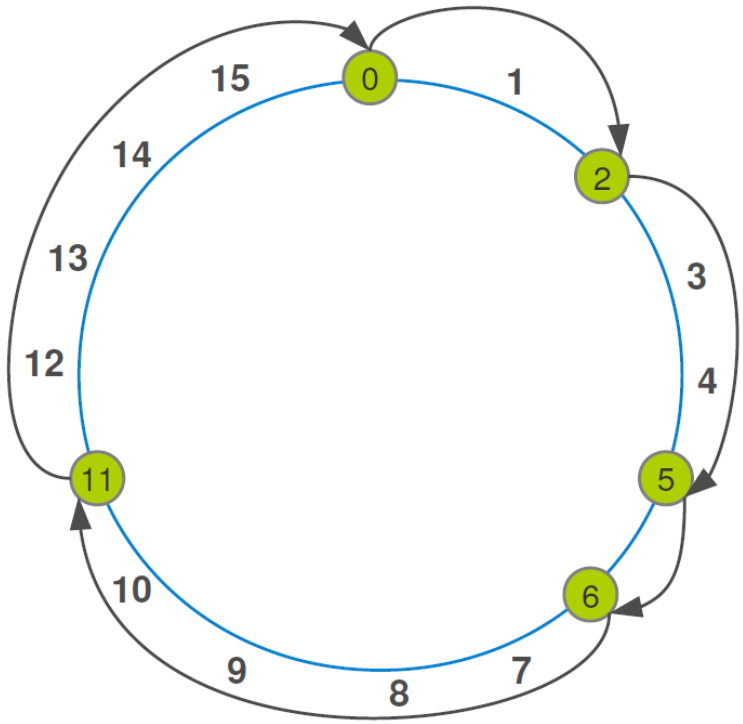
Construct Chord - Step 2 (1/2)

- The successor of an id is the first node met going in clockwise direction starting at the id.
- $succ(x)$: is the first node on the ring with id greater than or equal x.
  - $succ(12) = 0$
  - $succ(1) = 2$
  - $succ(6) = 6$



Construct Chord - Step 2 (2/2)

- Each node points to its successor.
- The successor of a node n is $succ(n+1)$.
  - 0's successor is $succ(1) = 2$.
  - 2's successor is $succ(3) = 5$.
  - 11's successor is $succ(12) = 0$.

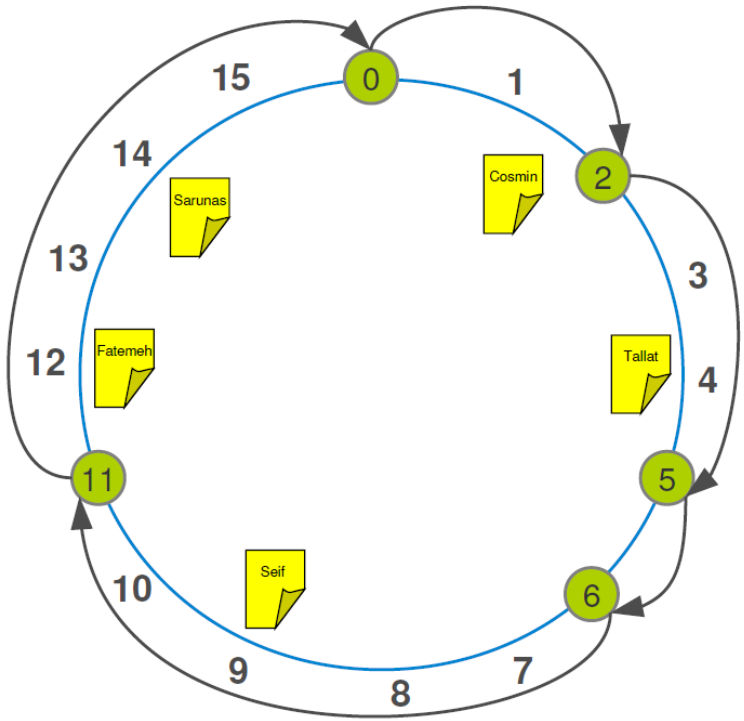


Construct Chord - Step 3

- Where to store data?

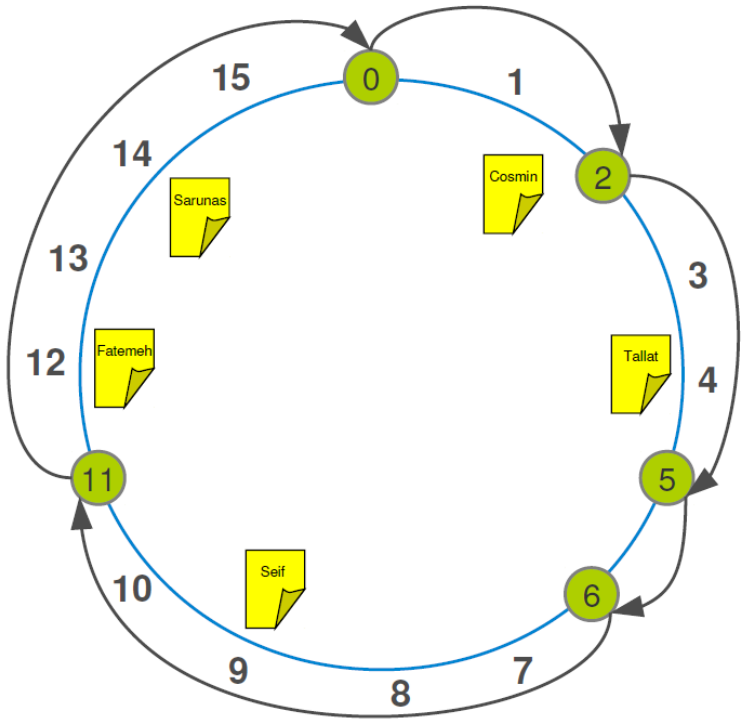
Construct Chord - Step 3

- Where to store data?
- Use globally known hash function H.



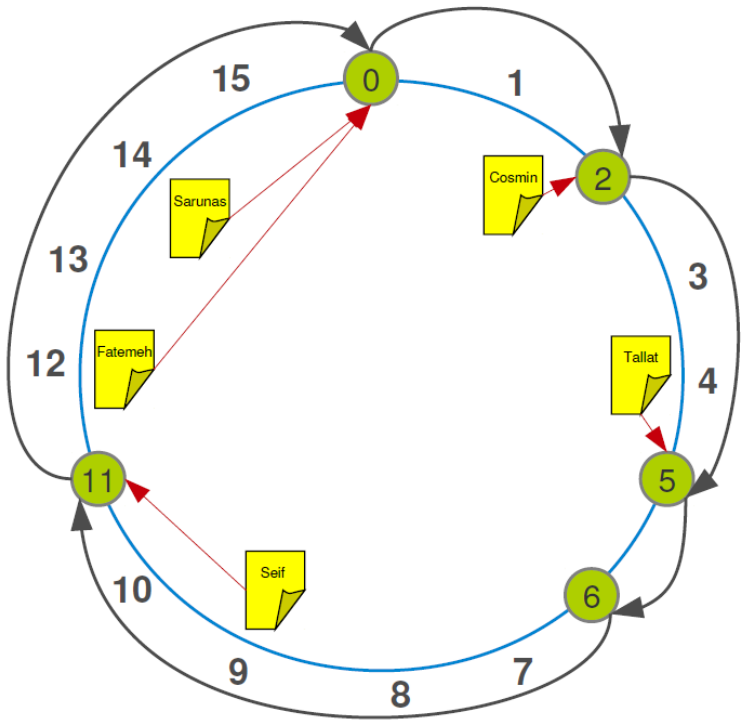
Construct Chord - Step 3

- Where to store data?
- Use globally known hash function H.
- Each item $\langle key, value \rangle$ gets identifier $H(key) = k$.
  - Space $N = 16\{0, \cdots, 15\}$
  - Five nodes a, b, c, d, e.
  - $H(Fatemeh) = 12$
  - $H(Cosmin) = 2$
  - $H(Seif) = 9$
  - $H(Sarunas) = 14$
  - $H(Tallat) = 4$



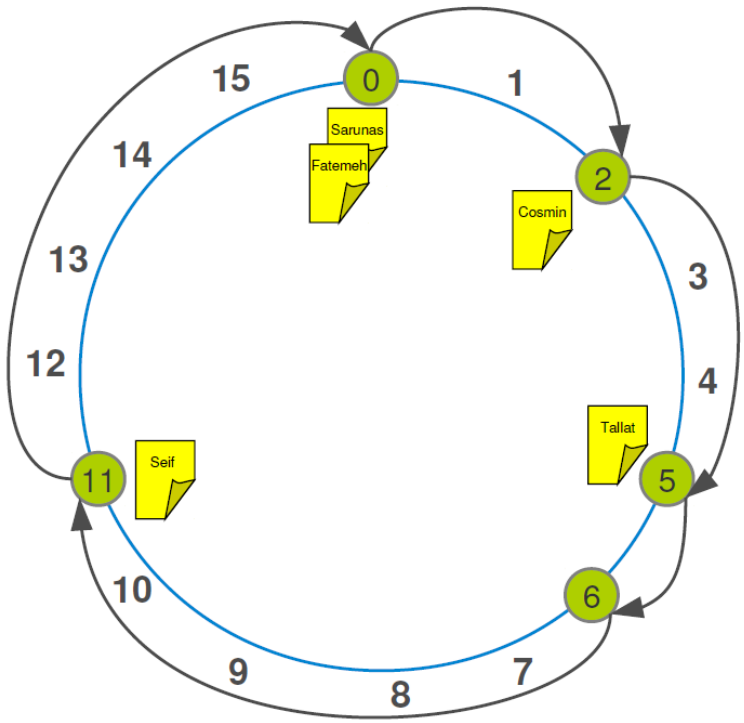
Construct Chord - Step 3

- Where to store data?
- Use globally known hash function H.
- Each item $\langle key, value \rangle$ gets identifier $H(key) = k$.
  - Space $N = 16\{0, \cdots, 15\}$
  - Five nodes a, b, c, d, e.
  - $H(Fatemeh) = 12$
  - $H(Cosmin) = 2$
  - $H(Seif) = 9$
  - $H(Sarunas) = 14$
  - $H(Tallat) = 4$
- Store each item at its successor.



Construct Chord - Step 3

- Where to store data?
- Use globally known hash function H.
- Each item $\langle key, value \rangle$ gets identifier $H(key) = k$.
  - Space $N = 16\{0, \cdots, 15\}$
  - Five nodes a, b, c, d, e.
  - $H(Fatemeh) = 12$
  - $H(Cosmin) = 2$
  - $H(Seif) = 9$
  - $H(Sarunas) = 14$
  - $H(Tallat) = 4$
- Store each item at its successor.



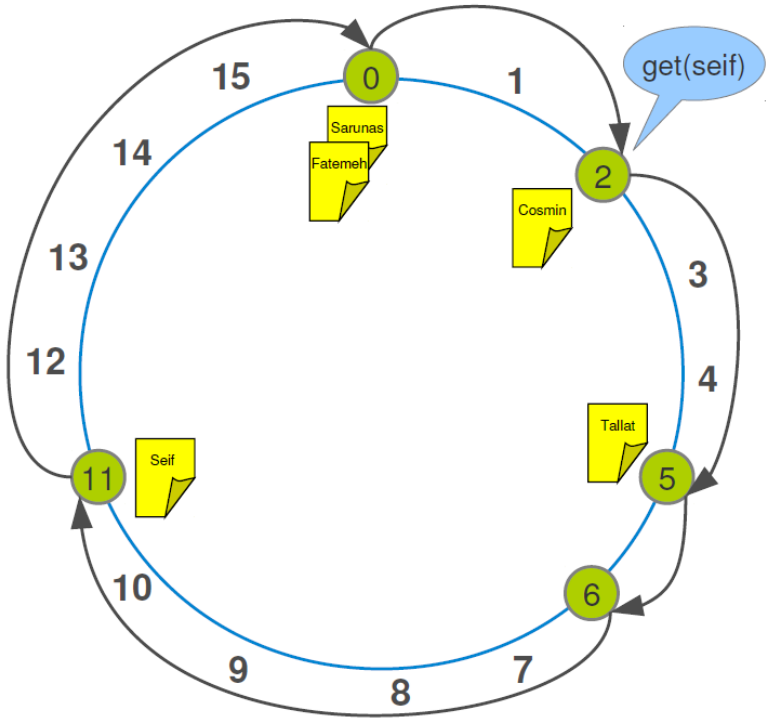
How to Lookup?

- To lookup a key k:
  - Calculate $H(k)$.
  - Follow succ pointers until item k is found.

How to Lookup?

- To lookup a key k:
  - Calculate $H(k)$.
  - Follow succ pointers until item k is found.
- Example:
  - Lookup Seif at node 2.
  - $H(Seif) = 9$
  - Traverse nodes: 2, 5, 6, 11
  - Return Stockholm to initiator

Key	Value
Seif	Stockholm



Unstructured P2P Systems

Unstructured P2P Systems

- Unstructured P2P systems attempt to maintain a random graph.

Unstructured P2P Systems

- Unstructured P2P systems attempt to maintain a random graph.
- Basic principle: each node is required to contact a randomly selected other node.

Unstructured P2P Systems

- Unstructured P2P systems attempt to maintain a random graph.
- Basic principle: 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.

Unstructured P2P Systems

- Unstructured P2P systems attempt to maintain a random graph.
- Basic principle: 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.
- It turns out that, depending on the exchange, randomness, but also robustness of the network can be maintained.

Gossiping and Aggregation

What is Gossiping?

Active thread

Passive thread

What is Gossiping?

Active thread

```
selectPeer(&B);
```

Passive thread

- `selectPeer`: randomly select a neighbor from partial view.

What is Gossiping?

Active thread

```
selectPeer(&B);
selectToSend(&bufs);
```

Passive thread

- `selectPeer`: randomly select a neighbor from partial view.
- `selectToSend`: select s entries from local cache.

What is Gossiping?

Active thread

```
selectPeer(&B);
selectToSend(&bufs);
sendTo(B, bufs);
```

Passive thread

```
receiveFromAny(&A, &bufr);
```

- `selectPeer`: randomly select a neighbor from partial view.
- `selectToSend`: select s entries from local cache.
- `selectToKeep`:
  1. add received entries to local cache.
  2. remove repeated items.
  3. shrink cache to size c (according to some strategy).

What is Gossiping?

Active thread

```
selectPeer(&B);
selectToSend(&bufs);
sendTo(B, bufs);
```

Passive thread

```
receiveFromAny(&A, &bufr);
selectToSend(&bufs);
```

- `selectPeer`: randomly select a neighbor from partial view.
- `selectToSend`: select s entries from local cache.
- `selectToKeep`:
  1. add received entries to local cache.
  2. remove repeated items.
  3. shrink cache to size c (according to some strategy).

What is Gossiping?

Active thread

```
selectPeer(&B);
selectToSend(&bufs);
sendTo(B, bufs);

receiveFrom(B, &bufr);
```

Passive thread

```
receiveFromAny(&A, &bufr);
selectToSend(&bufs);
sendTo(A, bufs);
```

- `selectPeer`: randomly select a neighbor from partial view.
- `selectToSend`: select s entries from local cache.
- `selectToKeep`:
  1. add received entries to local cache.
  2. remove repeated items.
  3. shrink cache to size c (according to some strategy).

What is Gossiping?

Active thread

```
selectPeer(&B);
selectToSend(&bufs);
sendTo(B, bufs);

receiveFrom(B, &bufr);
selectToKeep(cache, bufr);
```

Passive thread

```
receiveFromAny(&A, &bufr);
selectToSend(&bufs);
sendTo(A, bufs);
selectToKeep(cache, bufr);
```

- `selectPeer`: randomly select a neighbor from partial view.
- `selectToSend`: select s entries from local cache.
- `selectToKeep`:
  1. add received entries to local cache.
  2. remove repeated items.
  3. shrink cache to size c (according to some strategy).

Aggregation

- Aggregation provides a summary of some global system property.

Aggregation

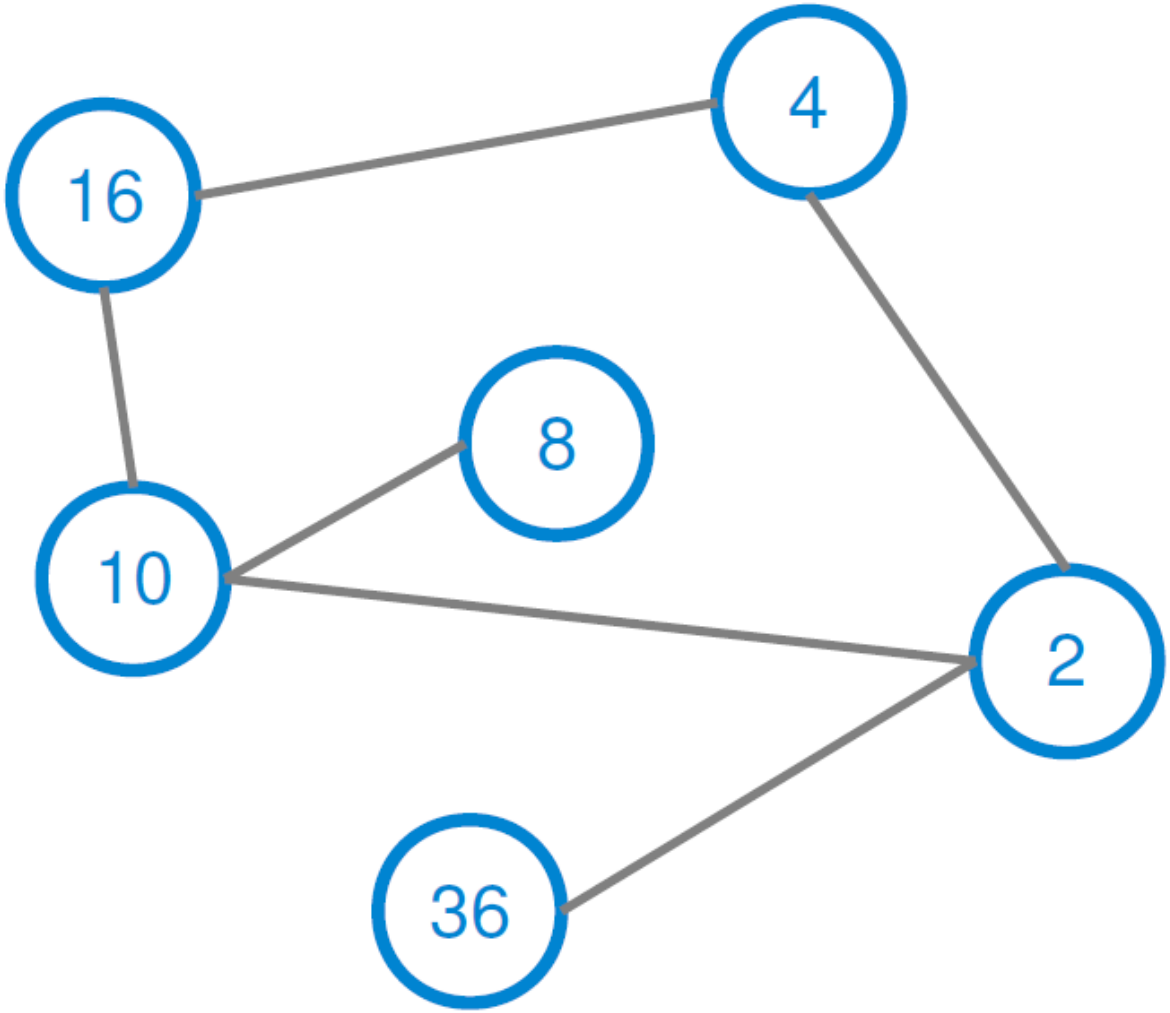
- Aggregation provides a summary of some global system property.
- It allows local access to global information.

Aggregation

- Aggregation provides a summary of some global system property.
- It allows local access to global information.
- Examples of aggregation functions:
  - The average load of nodes in a cluster.
  - The sum of free space in a distributed storage.
  - The total number of nodes in a P2P system.

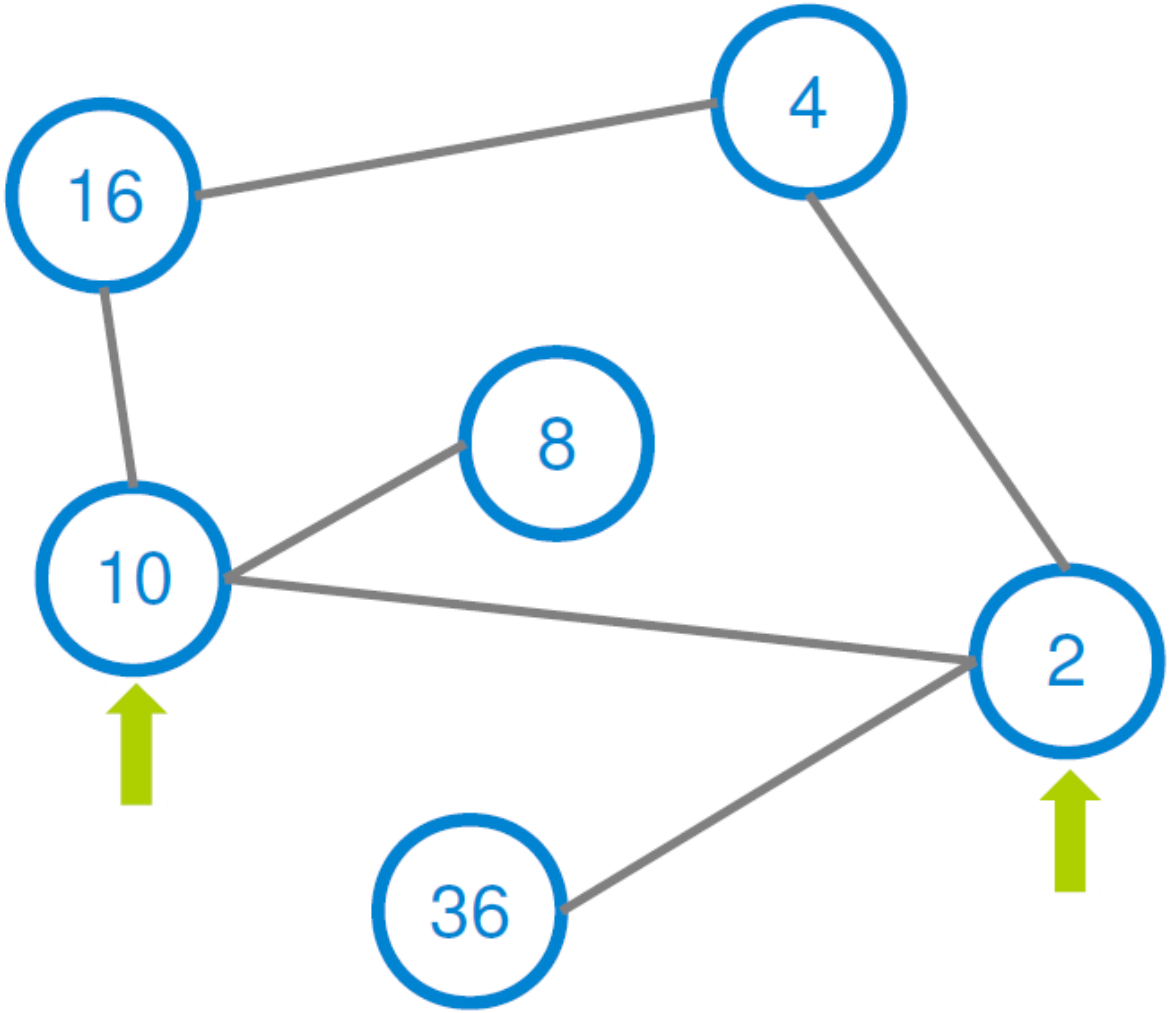
Aggregation Example (1/5)

- Taking the average of the numbers in the nodes.



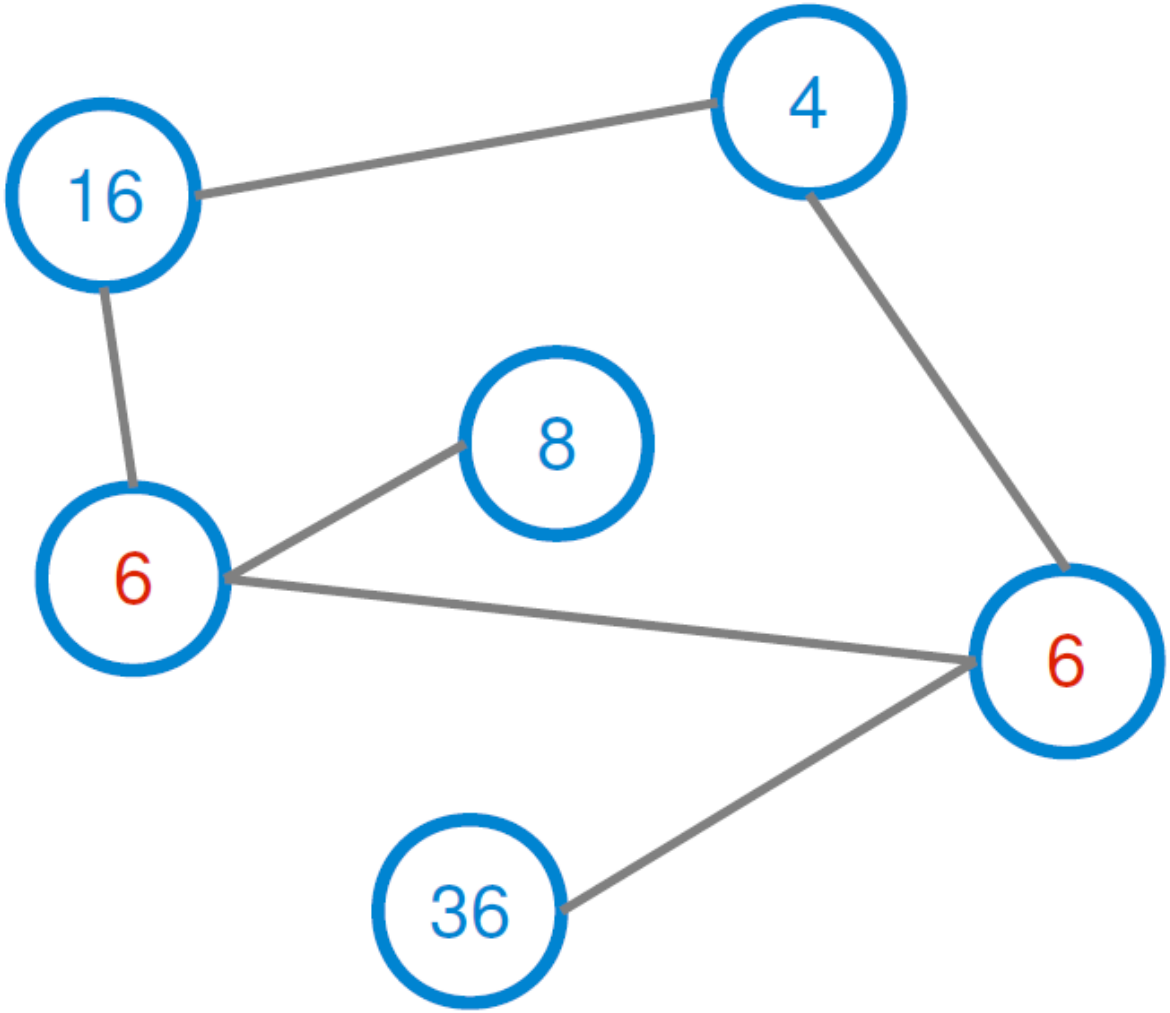
Aggregation Example (2/5)

- Taking the average of the numbers in the nodes.



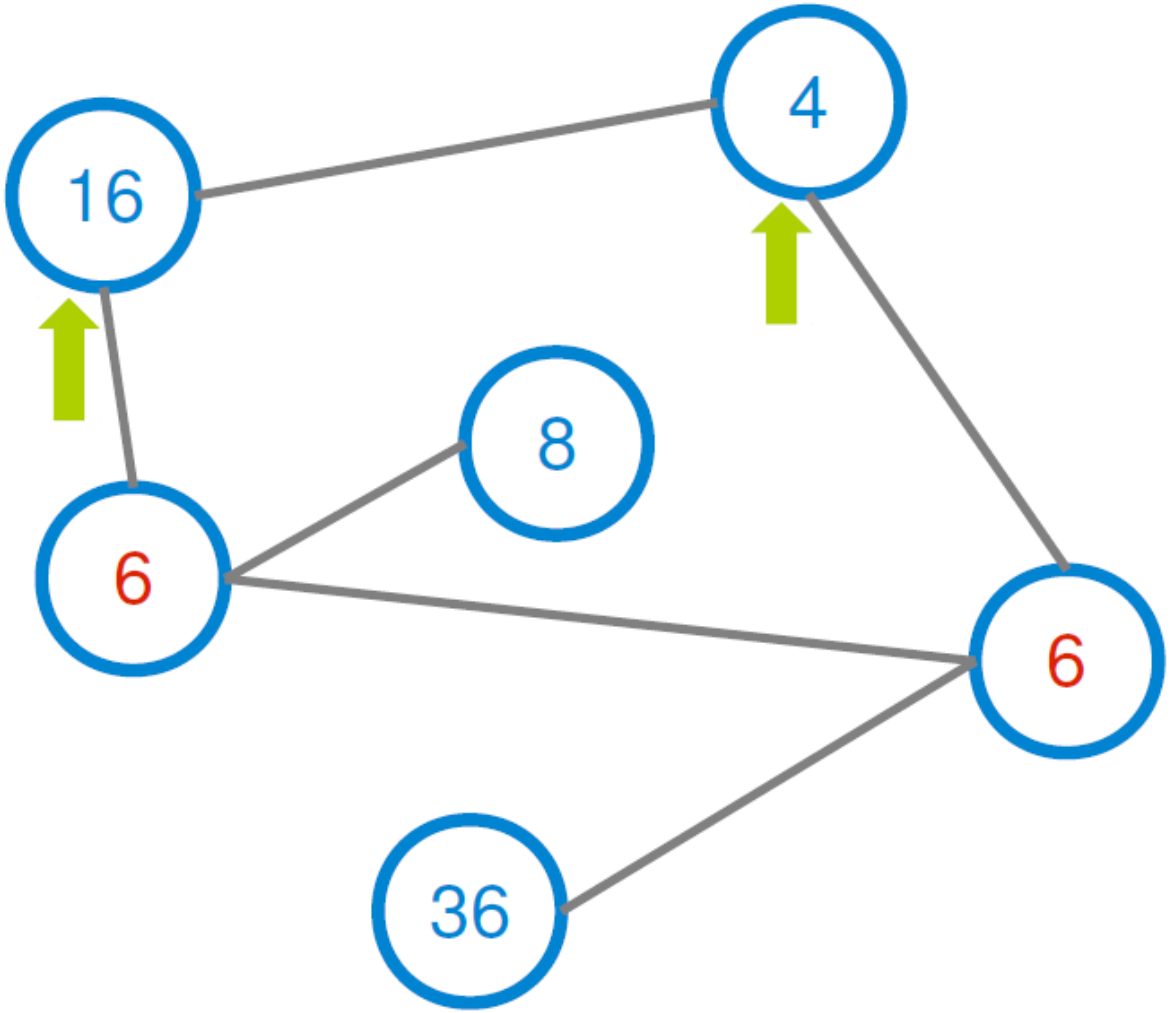
Aggregation Example (3/5)

- Taking the average of the numbers in the nodes.



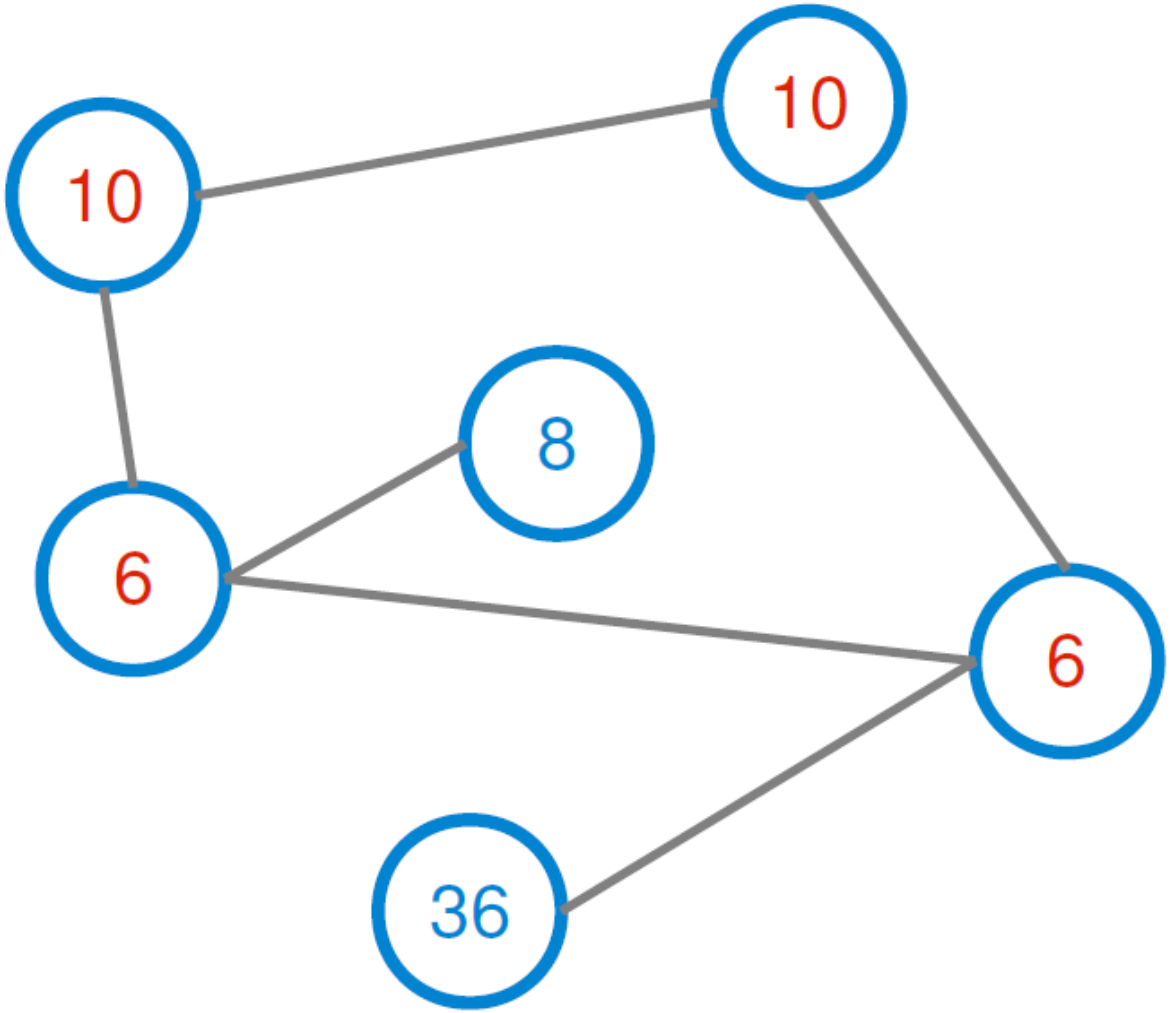
Aggregation Example (4/5)

- Taking the average of the numbers in the nodes.



Aggregation Example (5/5)

- Taking the average of the numbers in the nodes.



Gossiping-Based Peer Sampling

Gossip Protocols

- In a gossip protocol, each node in the system periodically exchanges information with a subset of nodes.

Gossip Protocols

- In a gossip protocol, each node in the system periodically exchanges information with a subset of nodes.
- The choice of this subset is crucial.

Gossip Protocols

- In a gossip protocol, each node in the system periodically exchanges information with a subset of nodes.
- The choice of this subset is crucial.
- Ideally, the nodes should be selected following a uniform random sample of all nodes currently in the system.

Achieving a Uniform Random Sample

- Each node may be assumed to know every other node in the system.

Achieving a Uniform Random Sample

- Each node may be assumed to know every other node in the system.
- Providing each node with a complete membership table is unrealistic in a large scale dynamic system.

Peer Sampling

- An alternative solution.

Peer Sampling

- An alternative solution.
- Every node maintains a relatively small local membership table that provides a partial view on the complete set of nodes.

Peer Sampling

- An alternative solution.
- Every node maintains a relatively small local membership table that provides a partial view on the complete set of nodes.
- Periodically refreshes the table using a gossiping procedure.

Gossip-based Peer Sampling

- Unify partial view and local cache $\Rightarrow$ exchange neighbors

Active thread

Passive thread

Gossip-based Peer Sampling

- Unify partial view and local cache $\Rightarrow$ exchange neighbors

Active thread

```
selectPeer(&B);
```

Passive thread

- `selectPeer`: randomly select a neighbor.

Gossip-based Peer Sampling

- Unify partial view and local cache $\Rightarrow$ exchange neighbors

Active thread

```
selectPeer(&B);
selectToSend(&peers_s);
```

Passive thread

- `selectPeer`: randomly select a neighbor.
- `selectToSend`: select s references to neighbors.

Gossip-based Peer Sampling

- Unify partial view and local cache $\Rightarrow$ exchange neighbors

Active thread

```
selectPeer(&B);
selectToSend(&peers_s);
sendTo(B, peers_s);
```

Passive thread

```
receiveFromAny(&A, &peers_r);
```

- `selectPeer`: randomly select a neighbor.
- `selectToSend`: select s references to neighbors.
- `selectToKeep`:
  1. add received references to partial view.
  2. remove repeated refs.
  3. shrink view to size c by randomly removing sent refs (but never received ones).

Gossip-based Peer Sampling

- Unify partial view and local cache $\Rightarrow$ exchange neighbors

Active thread

```
selectPeer(&B);
selectToSend(&peers_s);
sendTo(B, peers_s);
```

Passive thread

```
receiveFromAny(&A, &peers_r);
selectToSend(&peers_s);
```

- `selectPeer`: randomly select a neighbor.
- `selectToSend`: select s references to neighbors.
- `selectToKeep`:
  1. add received references to partial view.
  2. remove repeated refs.
  3. shrink view to size c by randomly removing sent refs (but never received ones).

Gossip-based Peer Sampling

- Unify partial view and local cache $\Rightarrow$ exchange neighbors

Active thread

```
selectPeer(&B);
selectToSend(&peers_s);
sendTo(B, peers_s);

receiveFrom(B, &peers_r);
```

Passive thread

```
receiveFromAny(&A, &peers_r);
selectToSend(&peers_s);
sendTo(A, peers_s);
```

- `selectPeer`: randomly select a neighbor.
- `selectToSend`: select s references to neighbors.
- `selectToKeep`:
  1. add received references to partial view.
  2. remove repeated refs.
  3. shrink view to size c by randomly removing sent refs (but never received ones).

Gossip-based Peer Sampling

- Unify partial view and local cache $\Rightarrow$ exchange neighbors

Active thread

```
selectPeer(&B);
selectToSend(&peers_s);
sendTo(B, peers_s);

receiveFrom(B, &peers_r);
selectToKeep(pview, peers_r);
```

Passive thread

```
receiveFromAny(&A, &peers_r);
selectToSend(&peers_s);
sendTo(A, peers_s);
selectToKeep(pview, peers_r);
```

- `selectPeer`: randomly select a neighbor.
- `selectToSend`: select s references to neighbors.
- `selectToKeep`:
  1. add received references to partial view.
  2. remove repeated refs.
  3. shrink view to size c by randomly removing sent refs (but never received ones).

Topology Management

Topology Management of Overlay Networks (1/3)

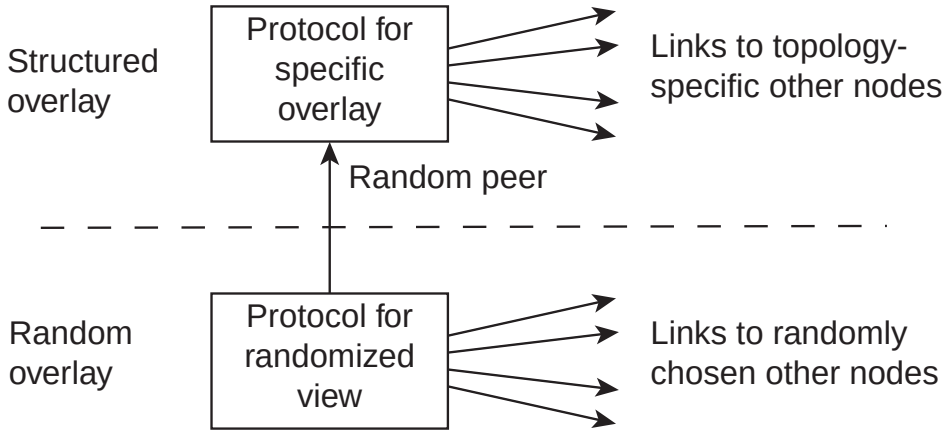
- A protocol to construct and maintain any topology with the help of a ranking function.

Topology Management of Overlay Networks (1/3)

- A protocol to construct and maintain any topology with the help of a ranking function.
- The ranking function orders any set of nodes according to their desirability to be neighbors of a given node.

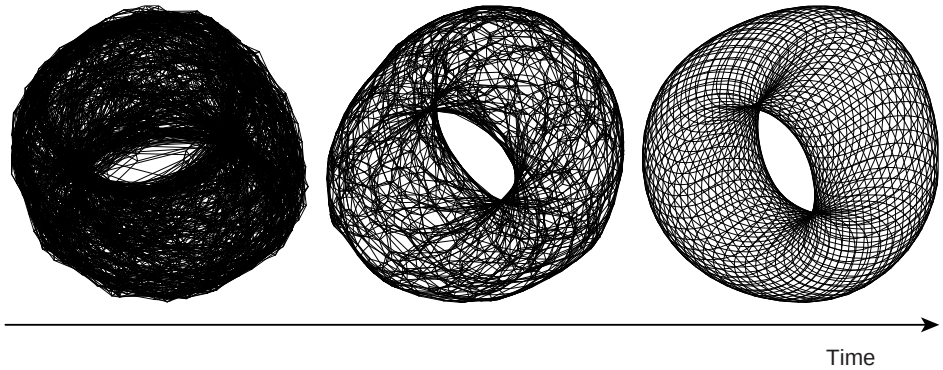
Topology Management of Overlay Networks (2/3)

- Distinguish two layers:
  1. The lower layer: maintains random partial views in lowest layer
  2. The upper layer: be selective on who you keep in higher-layer partial view
- Lower layer feeds upper layer with random nodes; upper layer is selective when it comes to keeping references.



Topology Management of Overlay Networks (3/3)

- Constructing a torus: consider a $N \times N$ grid.
- Keep only references to nearest neighbors in the upper layer:
  - Line: $d(a, b) = |a - b|$
  - Ring: $d(a, b) = min(N - |a - b|, |a - b|)$



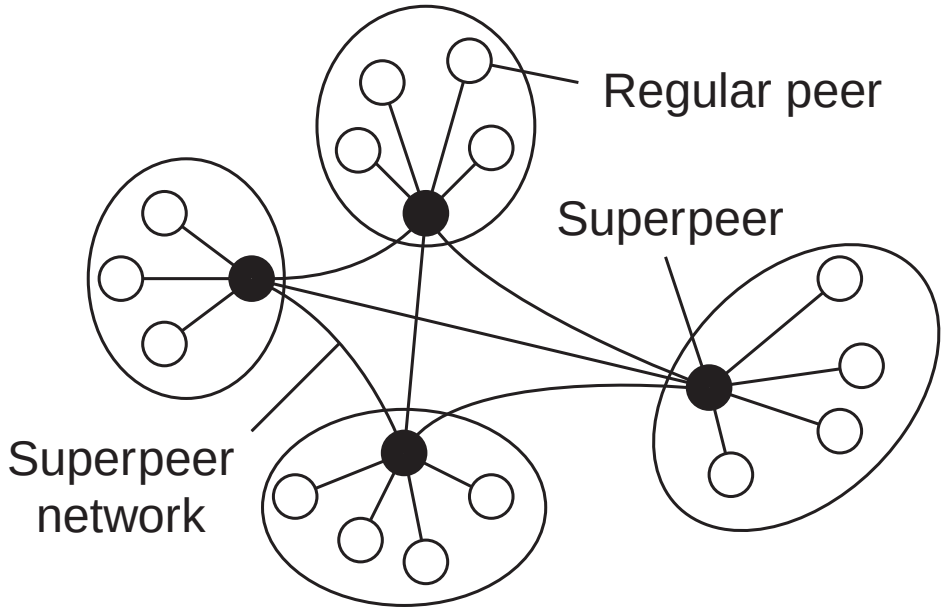
Hybrid P2P Systems

Superpeers

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

Superpeers

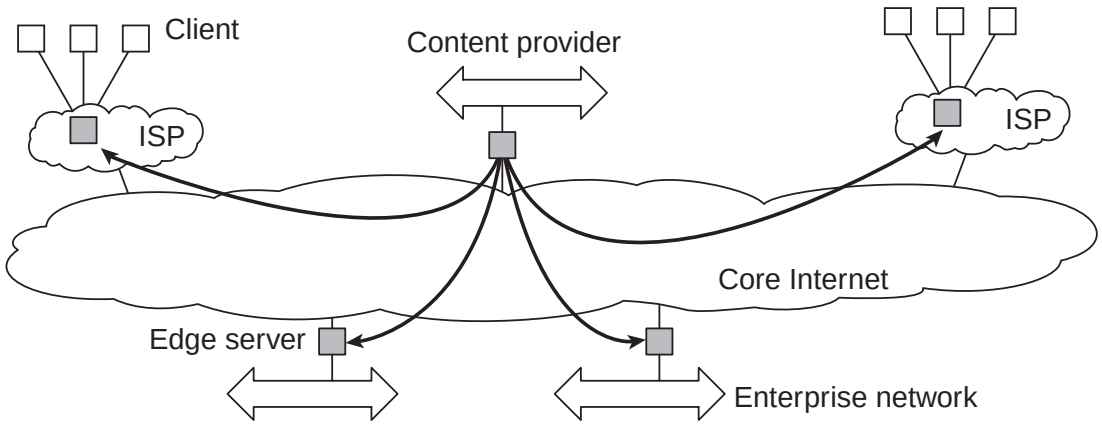
- Sometimes it helps to select a few nodes to do specific work: superpeer.
- Examples:
  - Peers maintaining an index (for search)
  - Peers monitoring the state of the network
  - Peers being able to setup connections



Hybrid Architectures

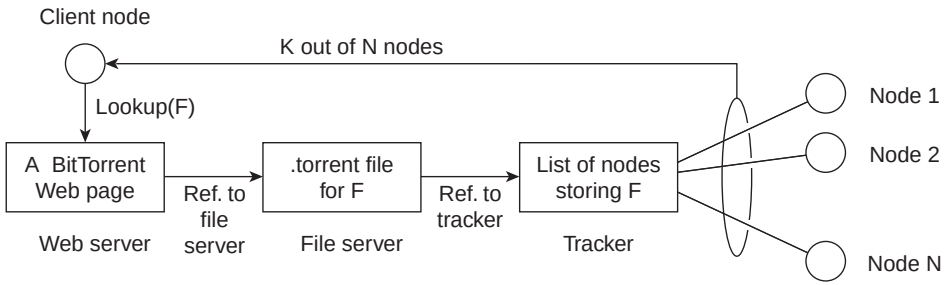
Hybrid Architectures (1/2)

- Client-server combined with P2P
- Edge-server architectures, which are often used for Content Delivery Networks (CDN)



Hybrid Architectures (2/2)

- Example: Bittorrent
- Once a node has identified where to download a file from, it joins a swarm of downloaders who in parallel get file chunks from the source, but also distribute these chunks amongst each other.



Summary

Summary

- Client-Server
  - Application layers, e.g., two-tier, three-tier
- P2P
  - Structured: DHT
  - Unstructured: gossip, peer sampling, topology management
  - Hybrid: superpeers
- Hybrid P2P and client-server: CDN + P2P

Reading

- Chapter 2 of the Distributed Systems: Principles and Paradigms.
- Stoica, Ion, et al., Chord: a scalable peer-to-peer lookup protocol for internet applications, Networking, IEEE/ACM Transactions on 11.1 (2003): 17-32.
- Jelasity, Mark, and Alberto Montresor, Epidemic-style proactive aggregation in large overlay networks, Distributed Computing Systems, 2004. Proceedings. 24th International Conference on. IEEE, 2004.
- Jelasity, Mark, et al., Gossip-based peer sampling, ACM Transactions on Computer Systems (TOCS) 25.3 (2007): 8.
- Jelasity, Mark, and Ozalp Babaoglu., T-Man: Gossip-based overlay topology management, Engineering Self-Organising Systems. Springer Berlin Heidelberg, 2006. 1-15.

Questions?