Architectures for Distributed Systems

Chapter 2

Definitions

Software Architectures

System Architectures

Definitions

- **Software Architectures**
  - describe the organization and interaction of software components
  - focuses on logical organization of software (component interaction, etc.)

Definitions

- **System Architectures**
  - describe the placement of software components on physical machines
  - Centralized most components located on a single machine
  - Decentralized most machines have approximately the same functionality
  - Hybrid some combination.

Architectural Styles
(Software Architectures)

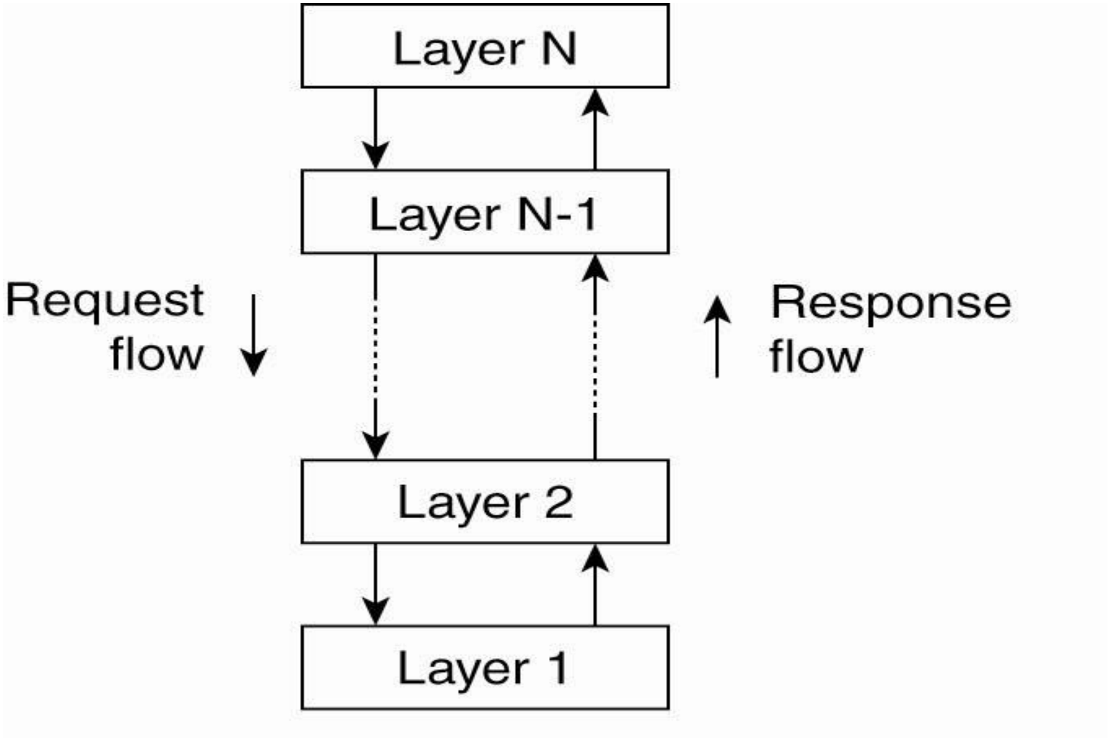
- An **architectural style** describes a particular way to configure a collection of components and connectors.

  - **Component -** a module with well-defined interfaces; reusable, replaceable
  - **Connector -** communication link between modules

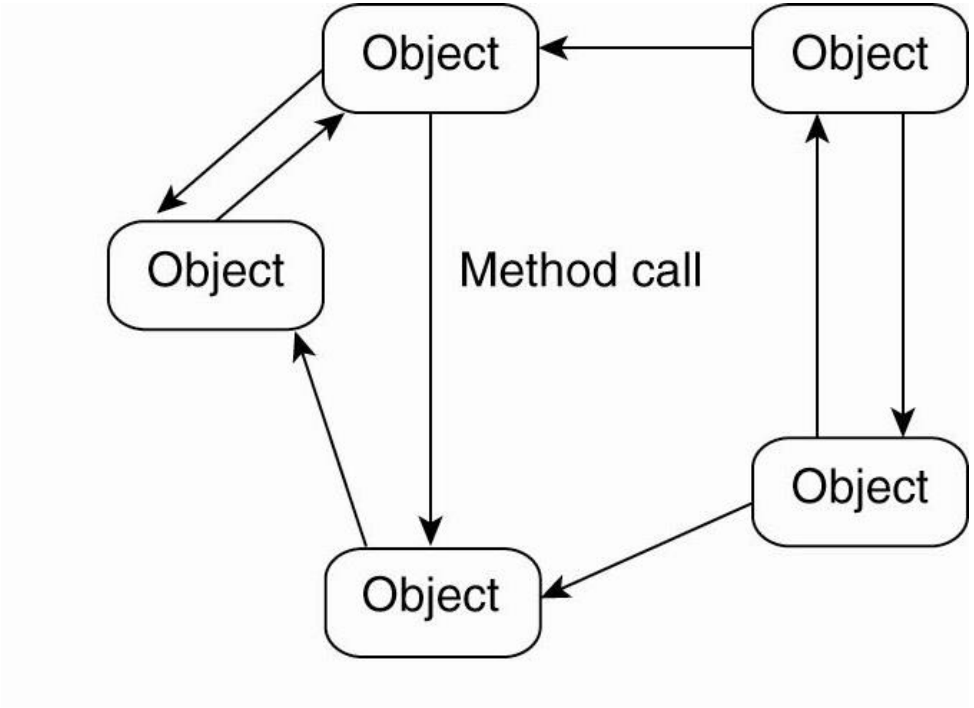
Architectural Styles

- Architectures suitable for distributed systems:
  - Layered architectures
  - Object-based architectures
  - Data-centered architectures
  - Event-based architectures

Layered architectures



Object based

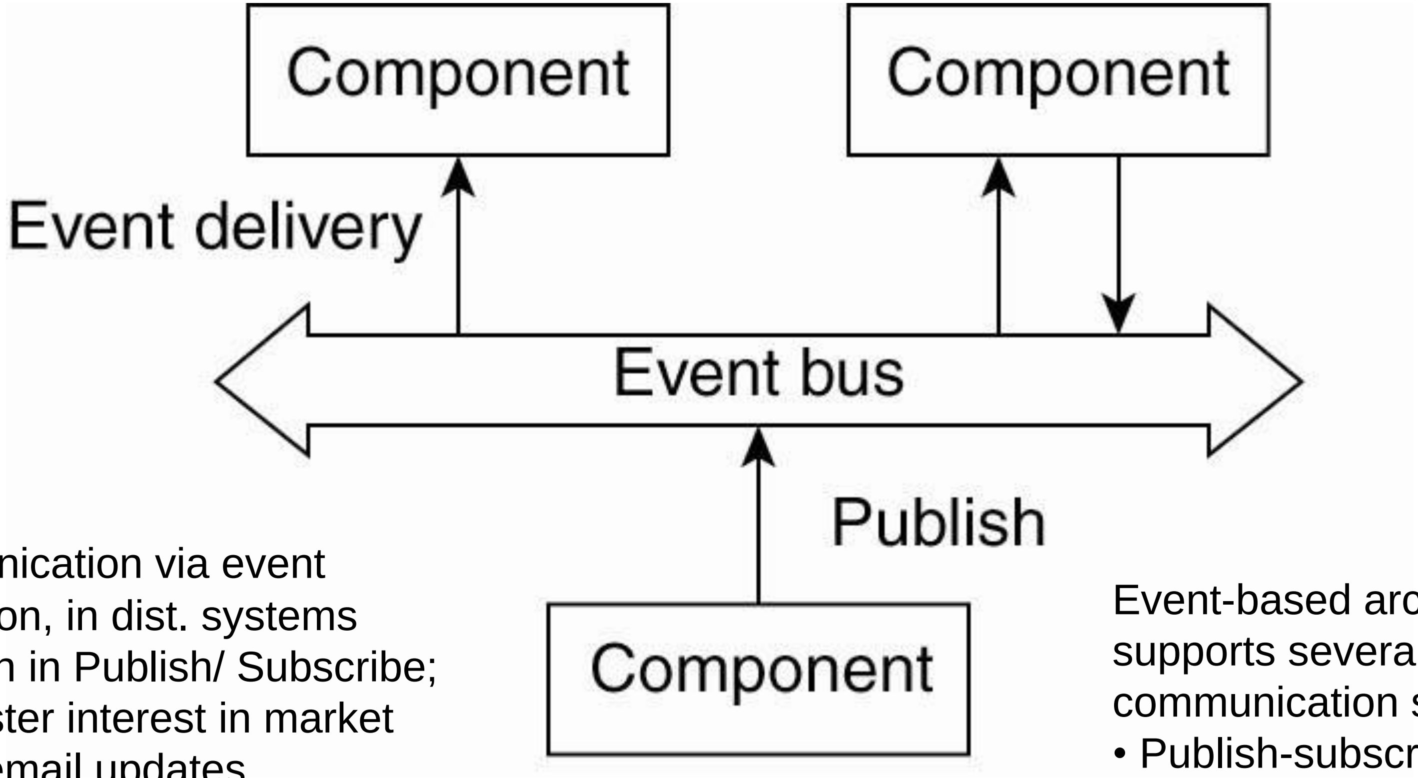


Object based is less structured
component = object
connector = RPC or RMI

Data-Centered Architectures

- Main purpose: **data** access and update
- Processes interact by reading and modifying data in some shared repository (active or passive)
- Example: web-based distributed systems where communication is through web services

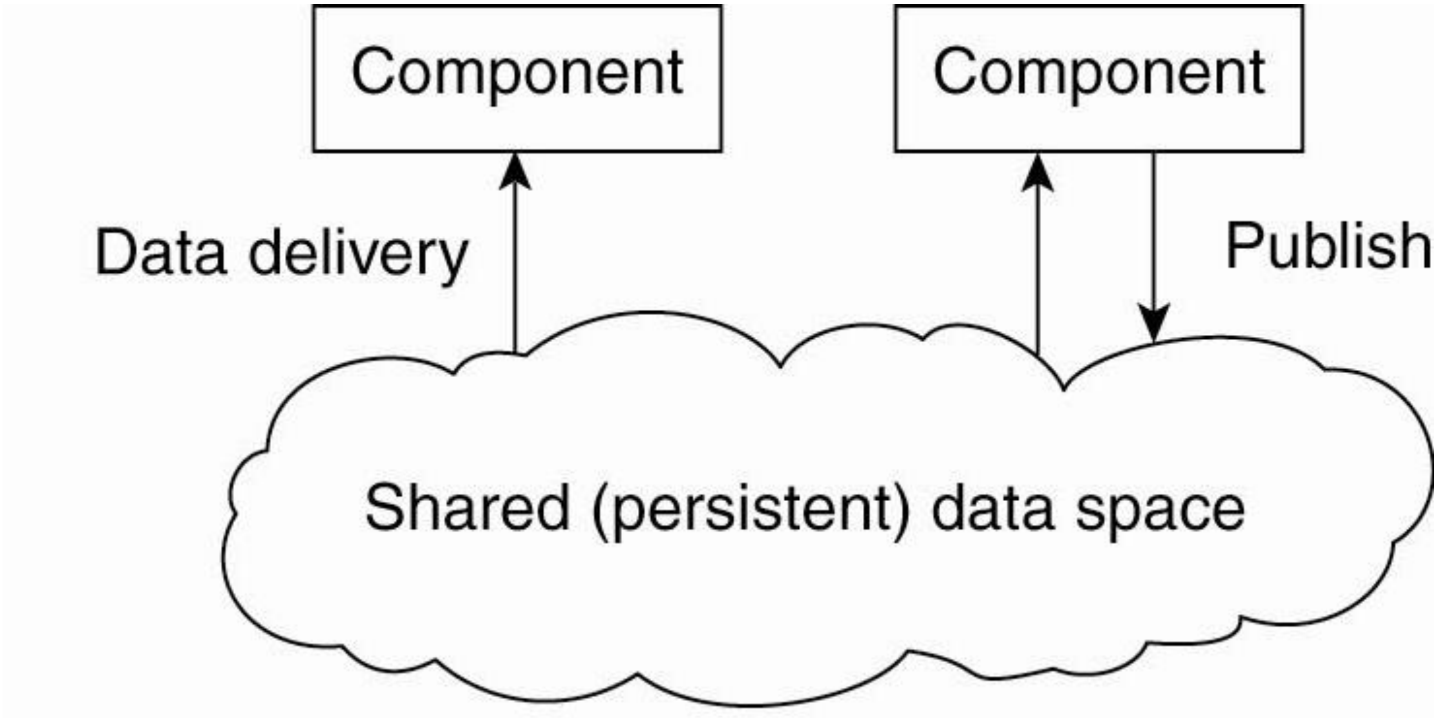
Event-based architectural



- Communication via event propagation, in dist. systems seen often in Publish/ Subscribe; *e.g.,* register interest in market info; get email updates
- Decouples sender & receiver; asynchronous communication

Event-based arch. supports several communication styles:
- Publish-subscribe
- Broadcast
- Point-to-point

Shared data-space architectural



Combination of data-centered and event based architectures

Processes communicate asynchronously

Distribution Transparency

- An important characteristic of software architectures in distributed systems is that they are designed to support distribution transparency.
- Transparency involves trade-offs
  - Performance
  - Fault tolerance
  - Ease-of-programming
- Different distributed applications require different solutions/architectures

System Architectures for Distributed Systems

- **Centralized**: traditional client-server structure
  - Vertical (or hierarchical) organization of communication and control paths (as in layered software architectures)
  - Logical separation of functions into client (requesting process) and server (responder)
- **Decentralized**: peer-to-peer
  - Horizontal rather than hierarchical comm. and control
  - Communication paths are less structured; symmetric functionality
- **Hybrid:** combine elements of C/S and P2P
  - Edge-server systems
  - Collaborative distributed systems.
- Classification of a system as centralized or decentralized refers to communication and control organization, primarily.

Traditional Client-Server

- Processes are divided into two groups (clients and servers).
- Synchronous communication: request-reply protocol
- In LANs, often implemented with a <u>connectionless</u> protocol
- In WANs, communication is typically <u>connection-oriented</u> TCP/IP
  - High likelihood of communication failures

C/S Architectures

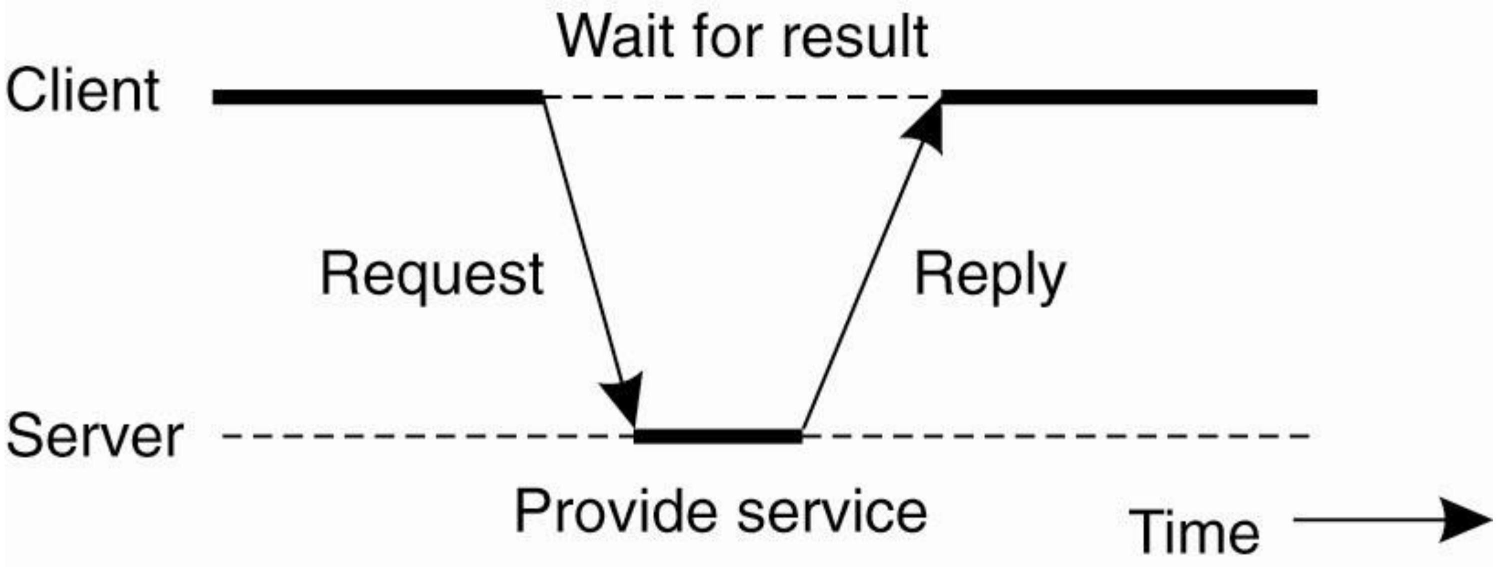


Figure 2-3. General interaction between a client and a server.

Transmission Failures

- With connectionless transmissions, failure of any sort means no reply
- Possibilities:
  - Request message was lost
  - Reply message was lost
  - Server failed either before, during or after performing the service
- Can the client tell which of the above errors took place?

Idempotency

- Typical response to lost request in connectionless communication:

  re-transmission

- Consider effect of re-sending a message such as "Increment X by 1000"
  - If first message was acted on, now the operation has been performed twice
- **Idempotent** operations: can be performed multiple times without harm
  - e.g., "Return current value of X"; check on availability of a product
  - Non-idempotent: "increment X", order a product

Layered (software) Architecture for Client-Server Systems

- **User-interface level**: GUI’s (usually) for interacting with end users
- **Processing level**: data processing applications – the core functionality
- **Data level**: interacts with data base or file system
  - Data usually is persistent; exists even if no client is accessing it
  - File or database system

Examples

- Web search engine
  - Interface: type in a keyword string
  - Processing level: processes to generate DB queries, rank replies, format response
  - Data level: database of web pages
- Stock broker’s decision support system
  - Interface: likely more complex than simple search
  - Processing: programs to analyze data; rely on statistics, AI perhaps, may require large simulations
  - Data level: DB of financial information

Application Layering

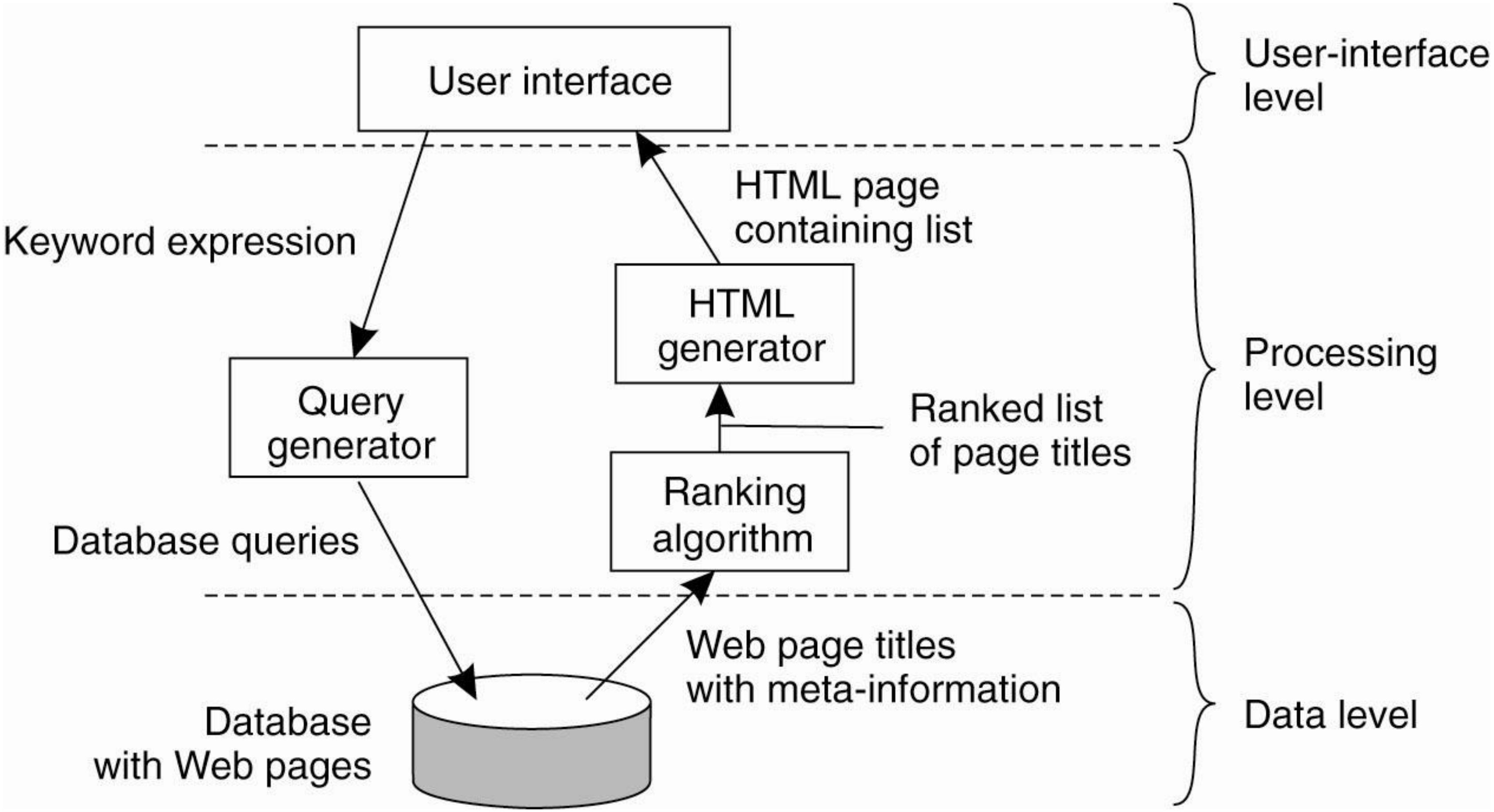


Figure 2-4. The simplified organization of an Internet search engine into three different layers.

Multi-tiered architectures

- *Layer* and *tier* are roughly equivalent terms, but *layer* typically implies software and *tier* is more likely to refer to hardware.

- Two-tier and three-tier are the most common

Two-tiered C/S Architectures

- Server provides processing and data management; client provides simple graphical display (**thin-client**)
  - Perceived performance loss at client
  - Easier to manage, more reliable, client machines don’t need to be so large and powerful
- At the other extreme, all application processing and some data resides at the client (**fat-client** approach)
  - reduces work load at server; more scalable
  - harder to manage by system admin, less secure

Multitiered Architectures

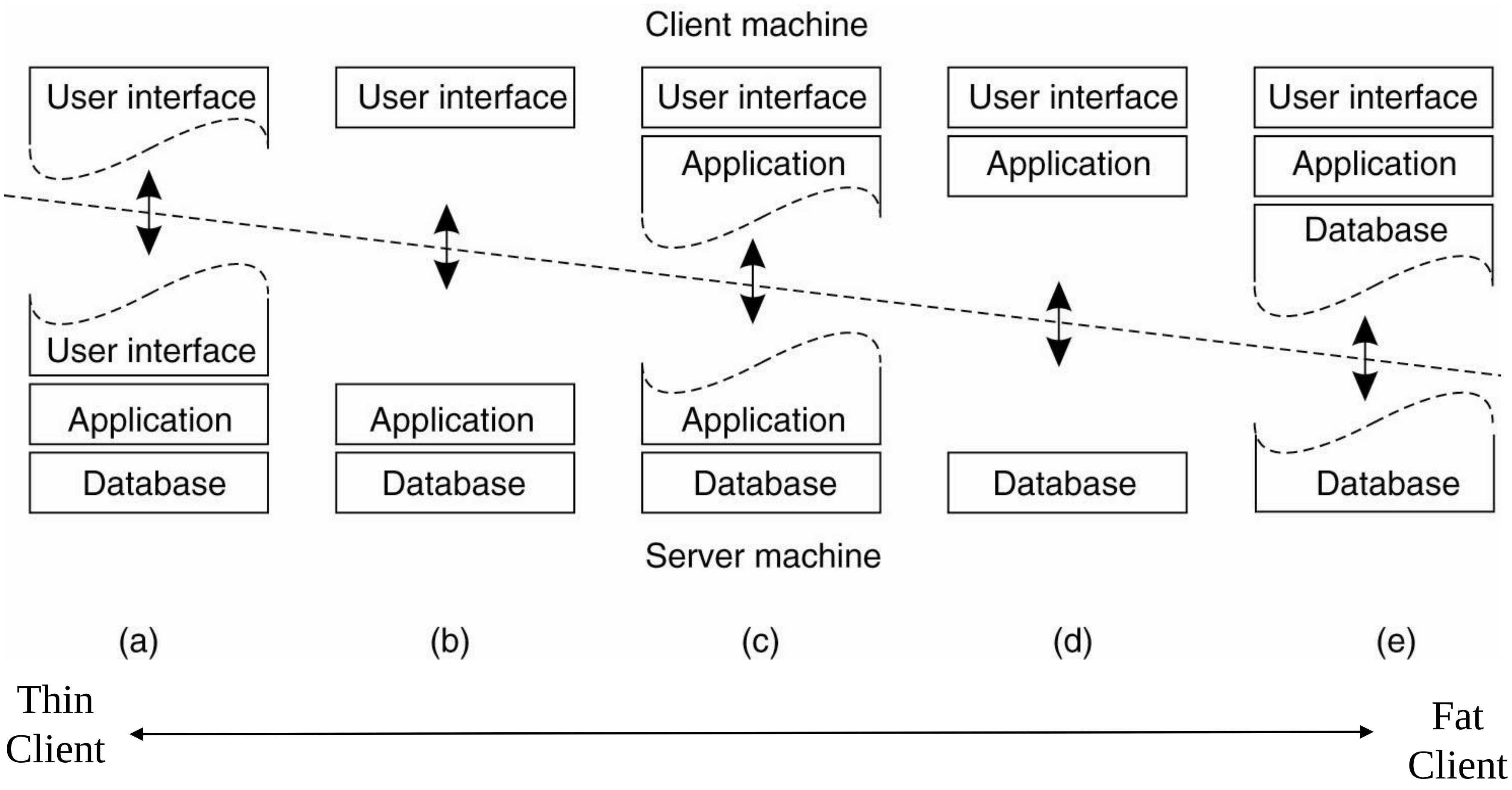


Figure 2-5. Alternative client-server organizations (a)–(e).

Three-tiered Architectures

- In some applications servers may also need to be clients, leading to a three level architecture
  - Distributed transaction processing
  - Web servers that interact with database servers

- Distribute functionality across three levels of machines instead of two.

Multi-tiered Architectures (3 Tier Architecture)

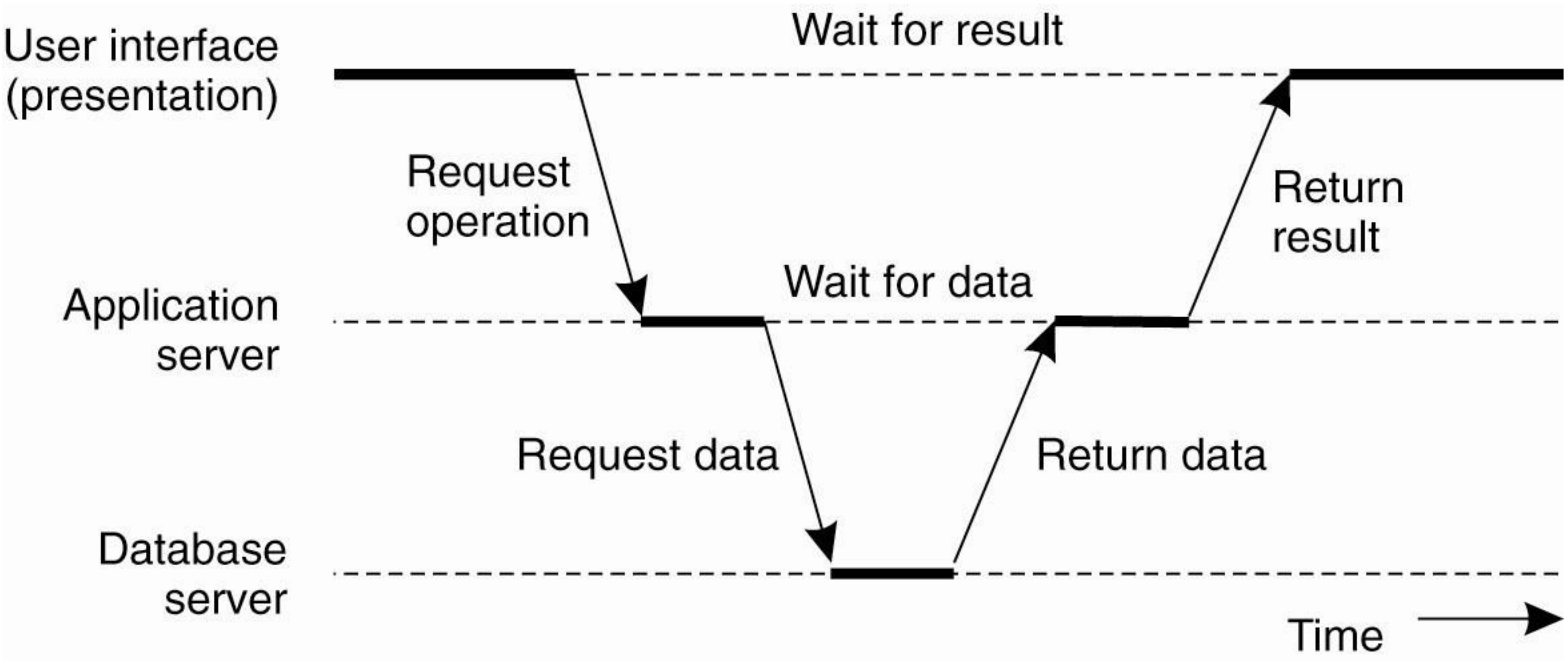


Figure 2-6. An example of a server acting as client.

Centralized vs Decentralized Architectures

- Traditional client-server architectures exhibit **vertical distribution**. Each level serves a <u>different purpose</u> in the system.
  - *Logically* different components reside on different nodes
- **Horizontal distribution** (P2P): each node has roughly the same processing capabilities and stores/manages part of the total system data.
  - Better load balancing, more resistant to denial-of-service attacks, harder to manage than C/S
  - Communication & control is not hierarchical; all about equal

Peer-to-Peer

- Nodes act as both client and server; interaction is symmetric
- Each node acts as a server for part of the total system data
- **Overlay networks** connect nodes in the P2P system
  - Nodes in the overlay use their own addressing system for storing and retrieving data in the system
  - Nodes can route requests to locations that may not be known by the requester.

Overlay Networks

- Are logical or *virtual* networks, built on top of a physical network

- A link between two nodes in the overlay may consist of several physical links.

- Messages in the overlay are sent to <u>logical</u> addresses, not physical (IP) addresses

- Various approaches used to resolve logical addresses to physical.

Overlay Networks

- Each node in a P2P system knows how to contact several other nodes.

- The overlay network may be
  - Structured (nodes and content are connected according to some design that simplifies later lookups)
  - Unstructured (content is assigned to nodes without regard to the network topology.)

Structured P2P Architectures

- A common approach is to use a **distributed hash table** (DHT) to organize the nodes
- Traditional hash functions convert a key to a hash value, which can be used as an index into a hash table.
    - Keys are unique – each represents an object to store in the table; e.g., at UAH, your A-number
    - The hash function value is used to insert an object in the hash table and to retrieve it.

Structured P2P Architectures

- In a DHT, data objects and nodes are each assigned a key which hashes to a random number from a very large identifier space (to ensure uniqueness)

- A mapping function assigns objects to nodes, based on the hash function value.

- A lookup, also based on hash function value, returns the network address of the node that stores the requested object.

Characteristics of DHT

- Scalable – to thousands, even millions of network nodes
  - Search time increases more slowly than size; usually $O(\log(N))$
- Fault tolerant – able to re-organize itself when nodes fail
- Decentralized – no central coordinator

Chord Routing Algorithm
Structured P2P

- Nodes are logically arranged in a circle
- Nodes and data items have m-bit identifiers (keys) from a 2^m namespace.
  - *e.g.*, a node's key is a hash of its IP address and a file's key might be the hash of its name or of its content or other unique key.
  - The hash function is *consistent*; which means that keys are distributed evenly across the nodes, with high probability.

Inserting Items in the DHT

- A data item with key value `k` is mapped to the node with the smallest identifier `id` such that `id ≥ k` (mod 2^m)

- This node is the successor of `k`, or `succ(k)`

- Modular arithmetic is used

Structured Peer-to-Peer Architectures

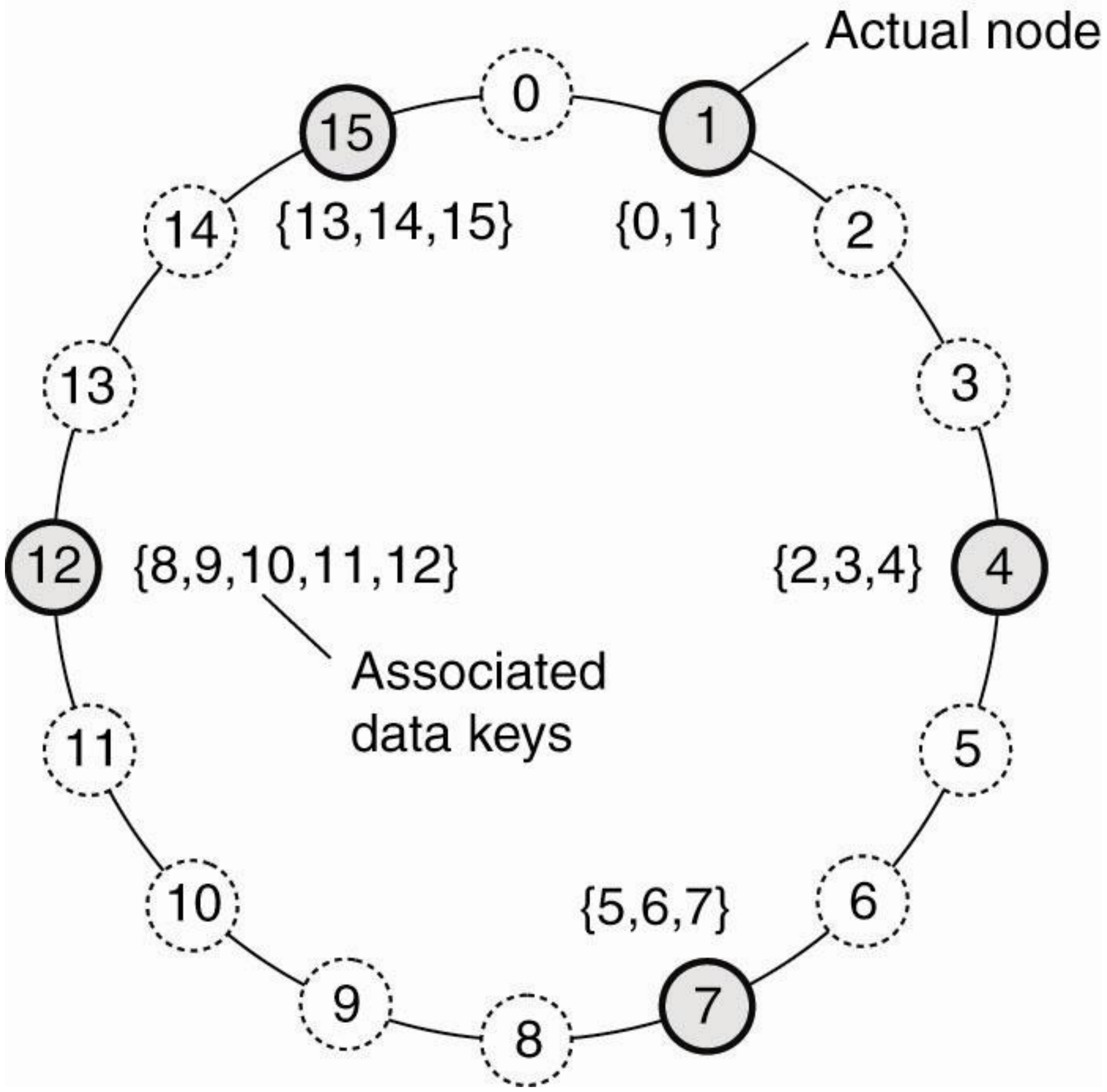


Figure 2-7. The mapping of data items onto nodes in Chord for m = 4

Finding Items in the DHT

- Each node in the network knows the location of some fraction of other nodes.
  - If the desired key is stored at one of these nodes, ask for it directly
  - Otherwise, ask one of the nodes you know to look in *its* set of known nodes.
  - The request will propagate through the overlay network until the desired key is located
  - Lookup time is $O(log(N))$

Joining & Leaving the Network

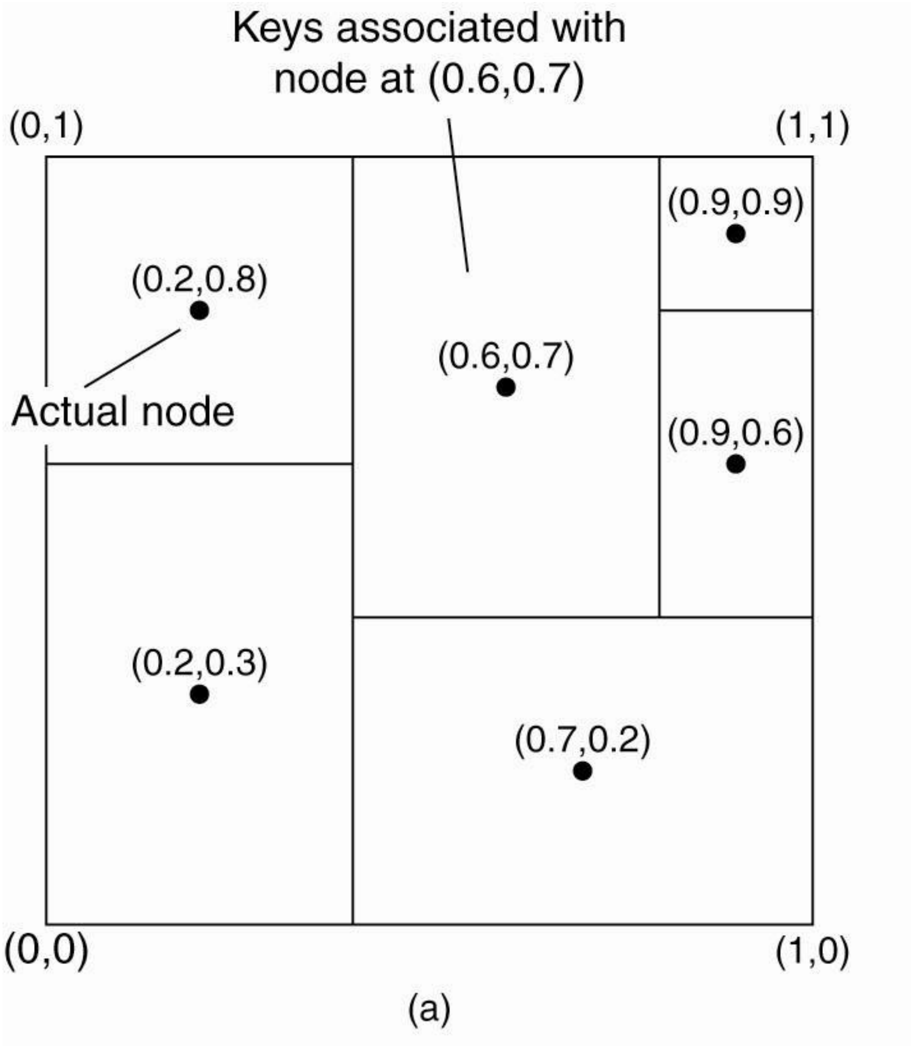
- Join
  - Generate the node's random identifier, `id`, using the distributed hash function
  - Use the lookup function to locate `succ(id)`
  - Contact `succ(id)` and its predecessor to insert self into ring.
  - Assume data items from `succ(id)`
- Leave (normally)
  - Notify predecessor & successor;
  - Shift data to `succ(id)`
- Leave (due to failure)
  - Periodically, nodes can run "self-healing" algorithms

Content Addressable Networks
Structured P2P

- A d-dimensional space is partitioned among all nodes
- Each node & each data item is assigned a point in the space.
- Data lookup is equivalent to knowing region boundary points and the responsible node for each region.

Structured Peer-to-Peer Architectures

- 2-dim space [0,1] x [0,1] is divided among 6 nodes
- Each node has an associated region
- Every data item in CAN will be assigned a unique point in space
- A node is responsible for all data elements mapped to its region



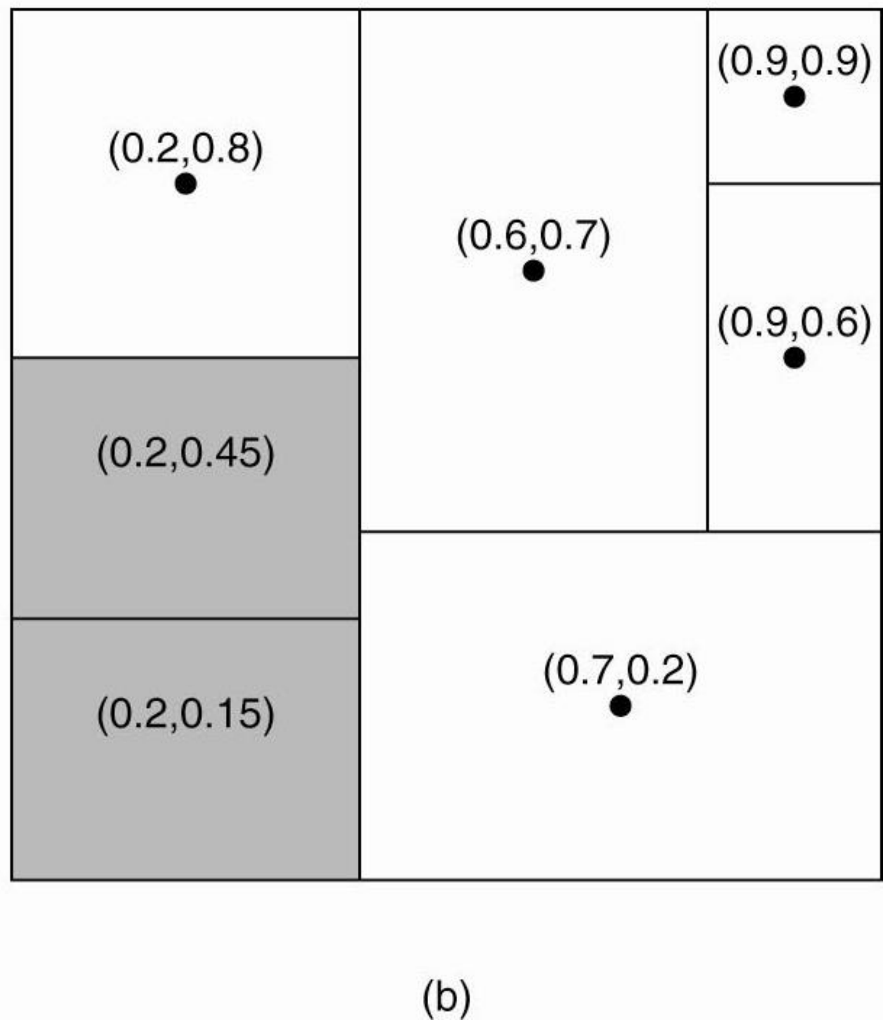
- Figure 2-8. (a) The mapping of data items onto nodes in CAN (Content Addressable Network).

Structured Peer-to-Peer Architectures

•To add a new region, split the region

•To remove an existing region, neighbor will take over

- Figure 2-8. (b) Splitting a region when a node joins.



Summary

- Deterministic: If an item is in the system it will be found
- No need to know where an item is stored
- Lookup operations are relatively efficient
- DHT-based P2P systems scale well

Unstructured P2P

- Unstructured P2P organizes the overlay network as a random graph.
- Each node knows about a subset of nodes, its “neighbors”.
  - Neighbors are chosen in different ways: physically close nodes, nodes that joined at about the same time, etc. –
- Data items are randomly mapped to some node in the system & lookup is random, unlike the structured lookup in Chord.

Locating a Data Object by Flooding

- Send a request to all known neighbors
  - If not found, neighbors forward the request to their neighbors
- Works well in small to medium sized networks, doesn't scale well
- "Time-to-live" counter can be used to control number of hops
- Example system: Freenet (Freenet uses a caching system to improve performance)

Comparison

- Structured networks typically guarantee that if an object is in the network it will be located in a bounded amount of time – usually *O(log(N))*
- Unstructured networks offer no guarantees.
  - For example, some will only forward search requests a specific number of hops
  - Random graph approach means there may be loops
  - Graph may become disconnected

Superpeers

- Maintain indexes to some or all nodes in the system
- Supports resource discovery
- Act as servers to regular peer nodes, peers to other **superpeers**
- Improve scalability by controlling floods
- Can also monitor state of network

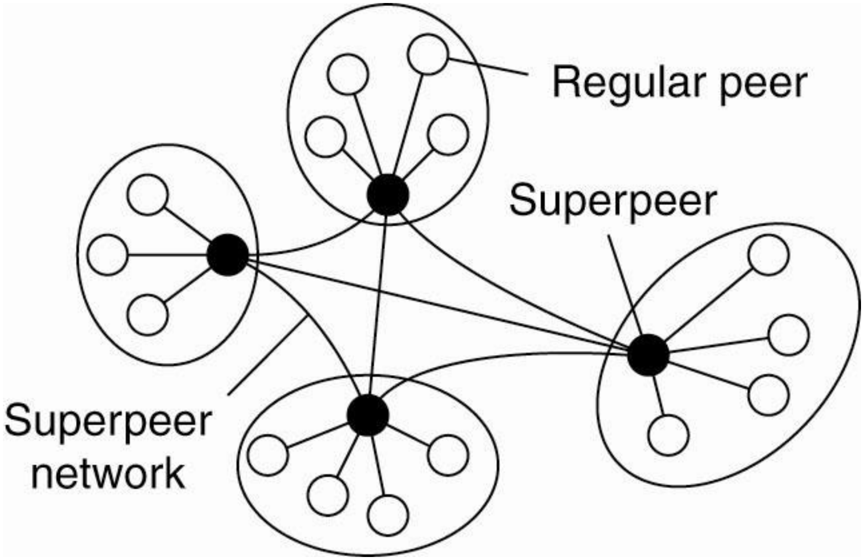


Figure 2-12.

Hybrid Architectures

- Combine client-server and P2P architectures
  - **Edge-server systems**; e.g. ISPs, which act as servers to their clients, but cooperate with other edge servers to host shared content
  - **Collaborative distributed systems**; e.g., BitTorrent, which supports parallel downloading and uploading of chunks of a file. First, interact with C/S system, then operate in decentralized manner.

Edge-Server Systems

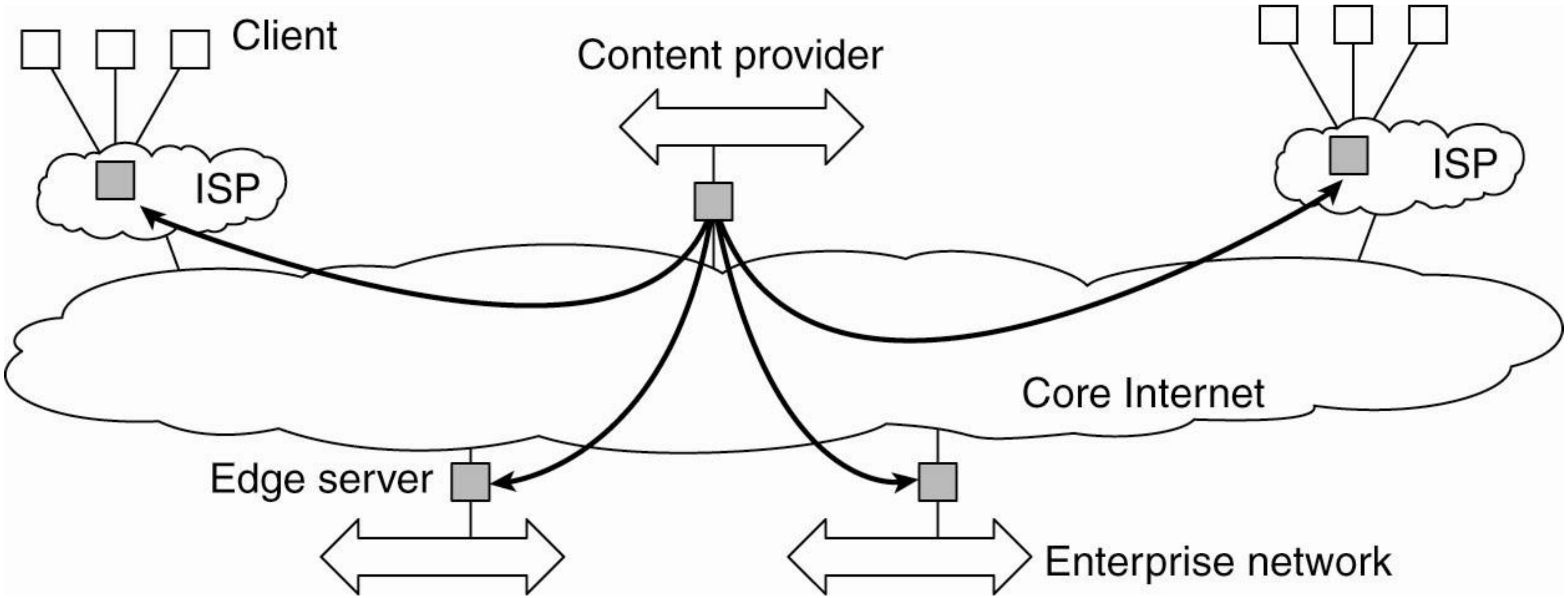


Figure 2-13. Viewing the Internet as consisting of a collection of edge servers.

Collaborative Distributed Systems
BitTorrent

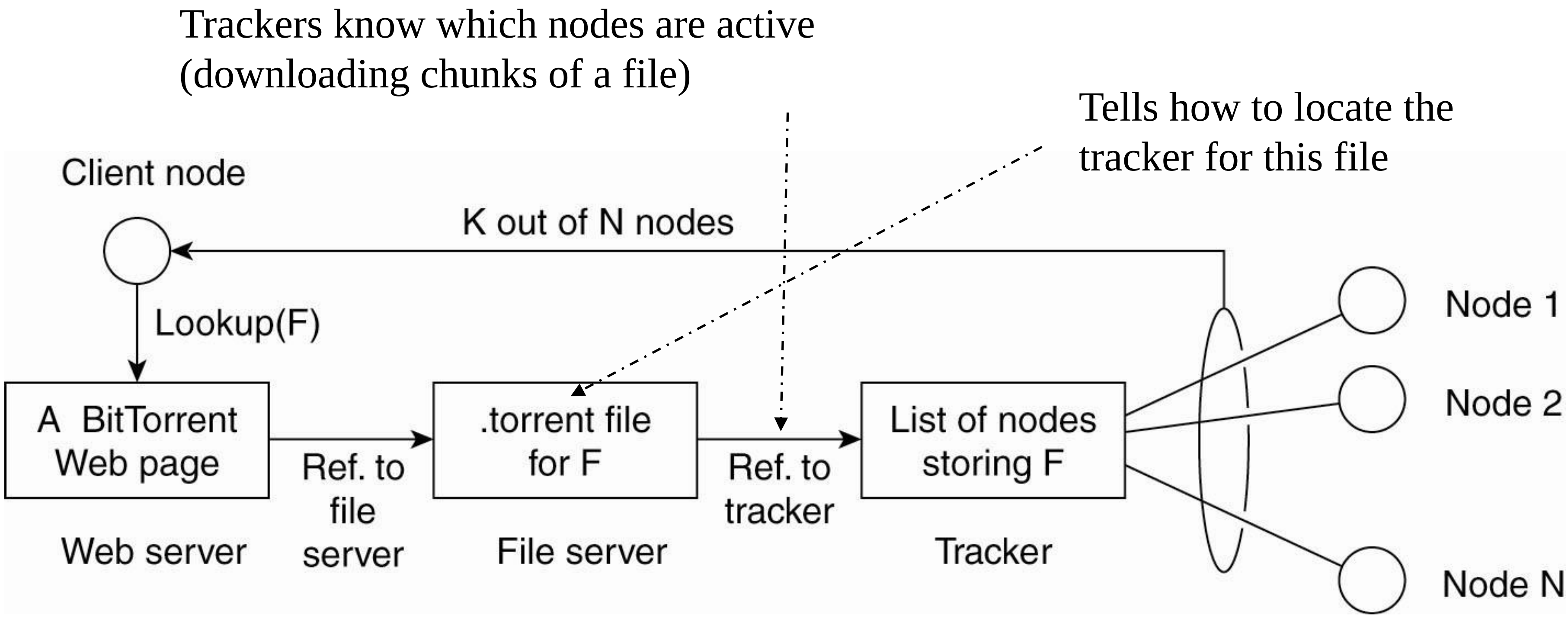
- Clients contact a global directory (Web server) to locate a *.torrent* file with the information needed to locate a **tracker**

- A server that can supply a list of active nodes that have chunks of the desired file.

- Using information from the tracker, clients can download the file in chunks from multiple sites in the network.

- Clients must also provide file chunks to other users.

Collaborative Distributed Systems



- Figure 2-14. The principal working of BitTorrent [adapted with permission from Pouwelse et al. (2004)].

BitTorrent - Justification

- Designed to force users of file-sharing systems to participate in sharing.
  - When a user downloads your file, he becomes in turn a server who can upload the file to other requesters.
  - Share the load – doesn’t swamp your server

P2P vs Client/Server

- P2P computing allows end users to communicate without a dedicated server.
- Communication is still usually synchronous
- There is less likelihood of performance bottlenecks since communication is more distributed.
  - Data distribution leads to workload distribution.
- Resource discovery is more difficult than in centralized client-server computing & look-up/retrieval is slower
- P2P can be more fault tolerant, more resistant to denial of service attacks because network content is distributed.
  - Individual hosts may be unreliable, but overall, the system should maintain a consistent level of service

Architecture versus Middleware

- Where does middleware fit into an architecture?
- Middleware: the software layer between user applications and distributed platforms.
- Purpose: to provide distribution transparency
  - Applications can access programs running on remote nodes without understanding the remote environment

Architecture versus Middleware

- Middleware may also have an architecture
  - e.g., CORBA has an object-oriented style.

- Use of a specific architectural style can make it easier to develop applications, but it may also lead to a less flexible system.

- Possible solution: develop middleware that can be customized as needed for different applications.

Interceptors

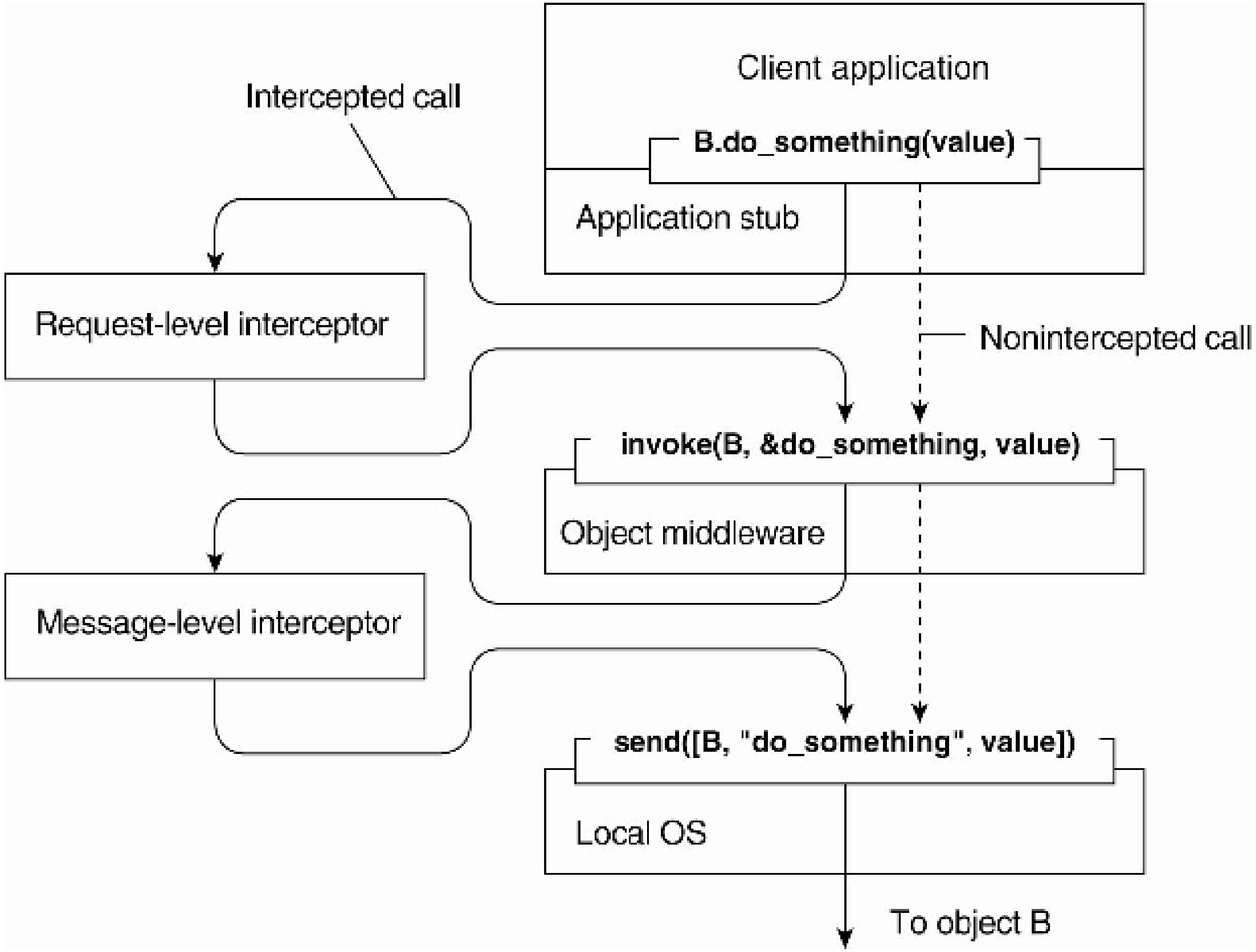


Figure 2-15. Using interceptors to handle remote-object invocations

General Approaches to Adaptive Software

Three basic approaches to adaptive software:

- Separation of concerns
- Computational reflection
- Component-based design