TDTS04/TDDD93: Distributed Systems

Instructor: Niklas Carlsson

Email: niklas.carlsson@liu.se

Notes derived from "Distributed Systems: Principles and Paradigms", by Andrew S. Tanenbaum and Maarten Van Steen, Pearson Int. Ed.

The slides are adapted and modified based on slides used by other instructors, including slides used in previous years by Juha Takkinen, as well as slides used by various colleagues from the distributed systems and networks research community.

Goals

- Study concepts that build the foundations of large-scale systems
- Learn about tradeoffs when building large-scale systems
- Learn from case studies, example systems
- Get exposure to system building and (if time) distributed systems research

Distributed systems

> "A collection of independent computers that appears to its users as a single coherent system"

- Hardware view
  - Multiple independent but cooperating resources
- Software view
  - Single unified system

Distributed systems

"A collection of independent computers that appears to its users as a single coherent system"

- Examples include …
  - Web
  - File-sharing
  - Scientific computing
  - Peer-to-peer
  - … (many many many more) ...
    - 2014: CDN example

Distributed systems

- Benefits?

- Problems?

Distributed systems

- Benefits?
  - Performance
  - Distribution
  - Reliability
  - Incremental growth
  - Sharing of data/resources
- Problems?

Distributed systems

- Benefits?
  - Performance
  - Distribution
  - Reliability
  - Incremental growth
  - Sharing of data/resources
- Problems?
  - Difficulties developing software
  - Network problems
  - Security problems

Distributed systems

- Goals
  - Sharing (incl. openness and heterogeneity)
  - Transparency
  - Scalability (incl. communication)
  - Reliability

Sharing

- Multiple users can share and access remote resources
  - Hardware, files, data, etc.
- Open standardized interface
  - Often heterogeneous environment (hardware, software, devices, underlying network protocols, etc.)
  - Middleware layer to mask heterogeneity
- Separate policies from mechanisms

Transparency

- Hide the distributed nature of system from users
- Several types:
  - **Location:** Hide where a resource is located
  - **Migration:** Resources can be moved
  - **Relocation:** Resources can be moved while being used
  - **Replication:** Multiple copies of same resource can exist
  - **Failure:** Hide failures of remote resources
  - …

Transparency in a Distributed System

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that multiple copies of a resource exist
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Different forms of transparency in a distributed system.

(**Bold** mentioned on previous slide too.)

Scalability

- Allow the system to become bigger without negatively affecting performance

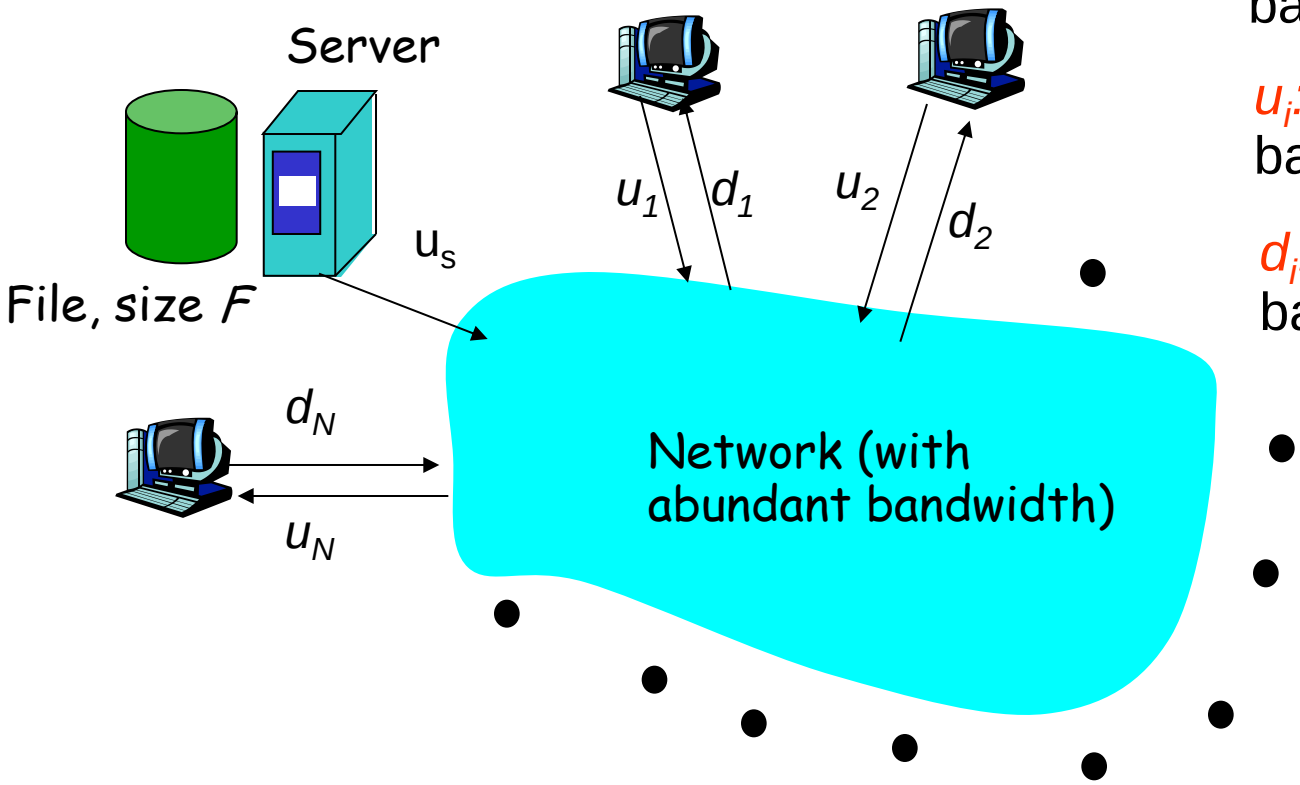
- Multiple dimensions:
  - **Size:** Adding more resources and users
  - **Geographic:** Dispersed across locations
  - **Administrative:** Spanning multiple administrative domains

Scalability

- Scalability problems appear as performance problems
  - System load, storage requirements, communication overhead, ...
- Some common techniques:
  - Divide and conquer
  - Replication
  - Distributed operation
  - Service aggregation
  - Asynchronous communication
  - Multicast

Scalability File Distribution Example: Server-Client vs P2P

Question : How much time to distribute file from one server to *N peers*?



u_s: server upload bandwidth

u_i: peer i upload bandwidth

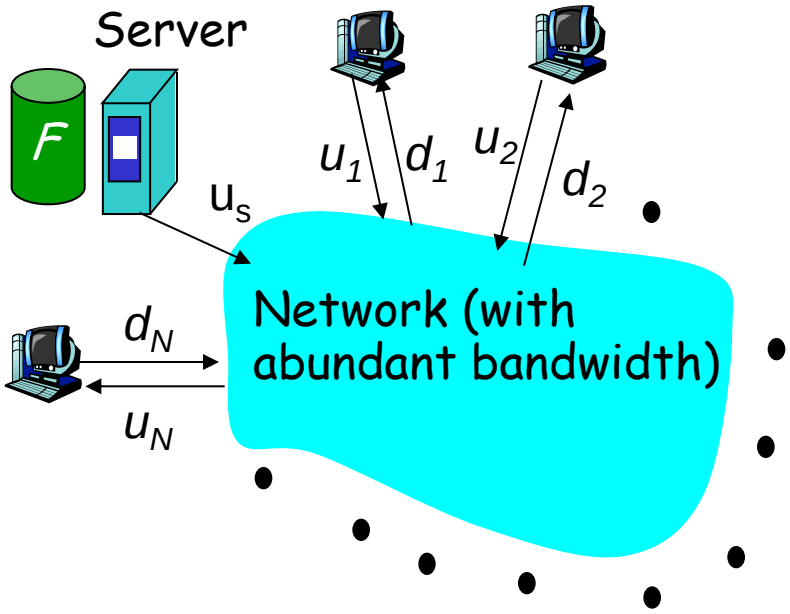
d_i: peer i download bandwidth

File distribution time: server-client

server must upload N copies:

- NF/u_s time

client i takes F/d_i time to download



Time to distribute F to N clients using client/server approach $= d_{cs} = \max_i \{ NF/u_s, F/\min(d_i) \}$

increases linearly in N (for large N)

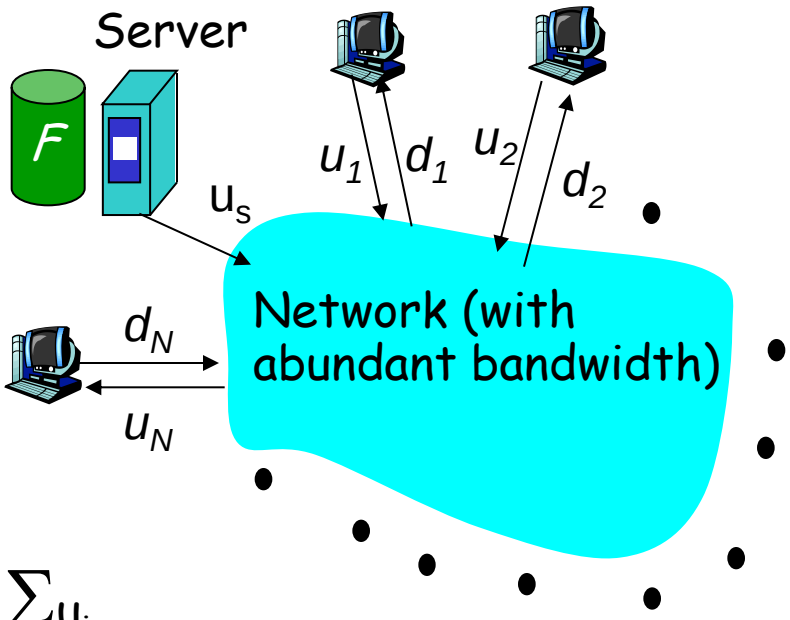
File distribution time: P2P

server must send one copy: F/u_s time

client i takes F/d_i time to download

NF bits must be downloaded (aggregate)

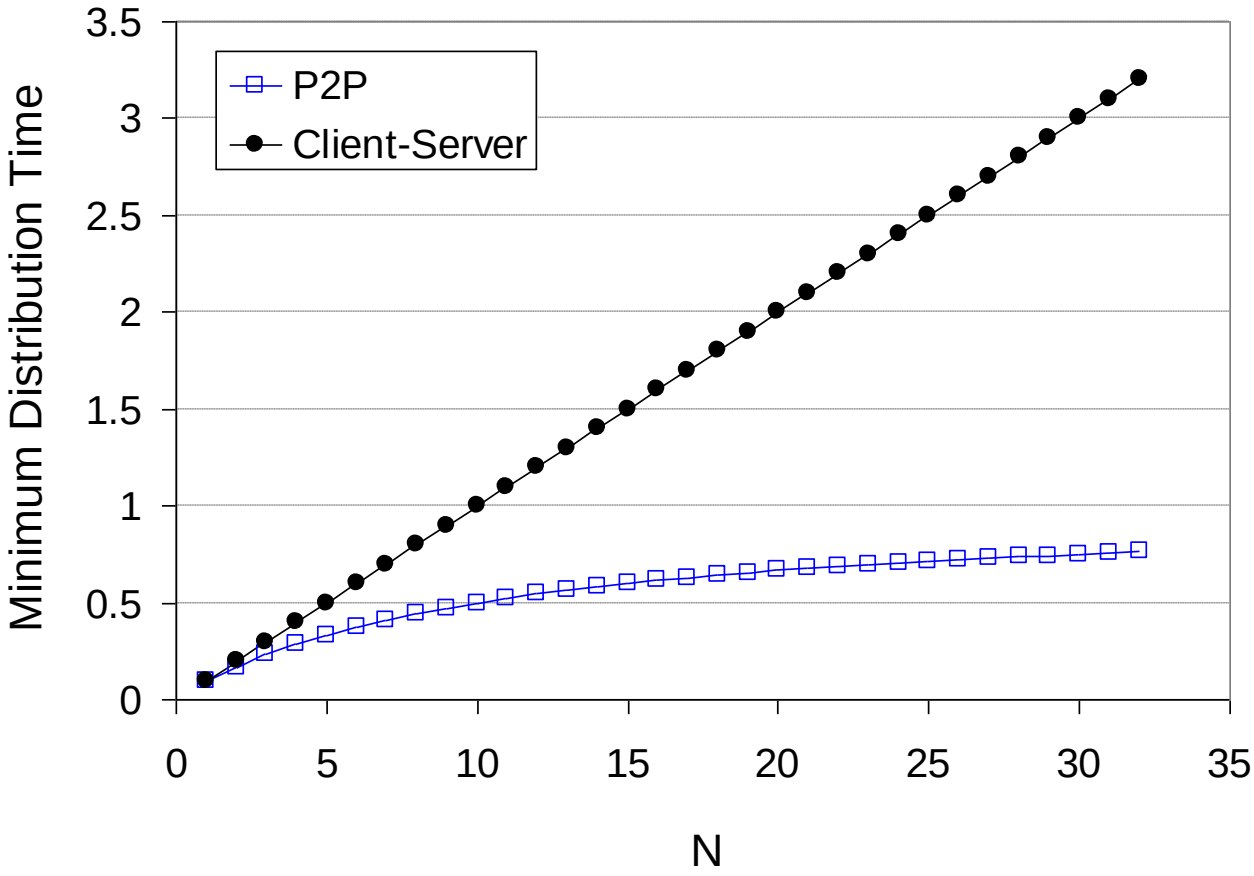
- fastest possible upload rate: $u_s + \sum u_i$



$$d_{P2P} = \max_i \{ F/u_s, F/\min(d_i) , NF/(u_s + \sum u_i) \}$$

Server-client vs. P2P: example

Client upload rate = u, F/u = 1 hour, u_s = 10u, $d_{min} \geq u_s$



Reliability

- Availability
  - If a machines goes down, the system should work with the reduced amount of resources
  - Replication used to ensure that data is not lost (should be consistent)
- Fault tolerance
  - The system must be able to detect faults, mask faults (if possible), or gracefully fail (if needed)

Common Pitfalls (bad/dangerous assumptions!)

- The network is reliable
- The network is secure
- The network is homogenous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport cost is zero
- There is one administrator

Distributed system architecture

- A distributed application runs across multiple machines
  - How to organize the various pieces of the application?
  - Where is the user interface, computation, data?
  - How do different pieces interact with each other?

Architectures

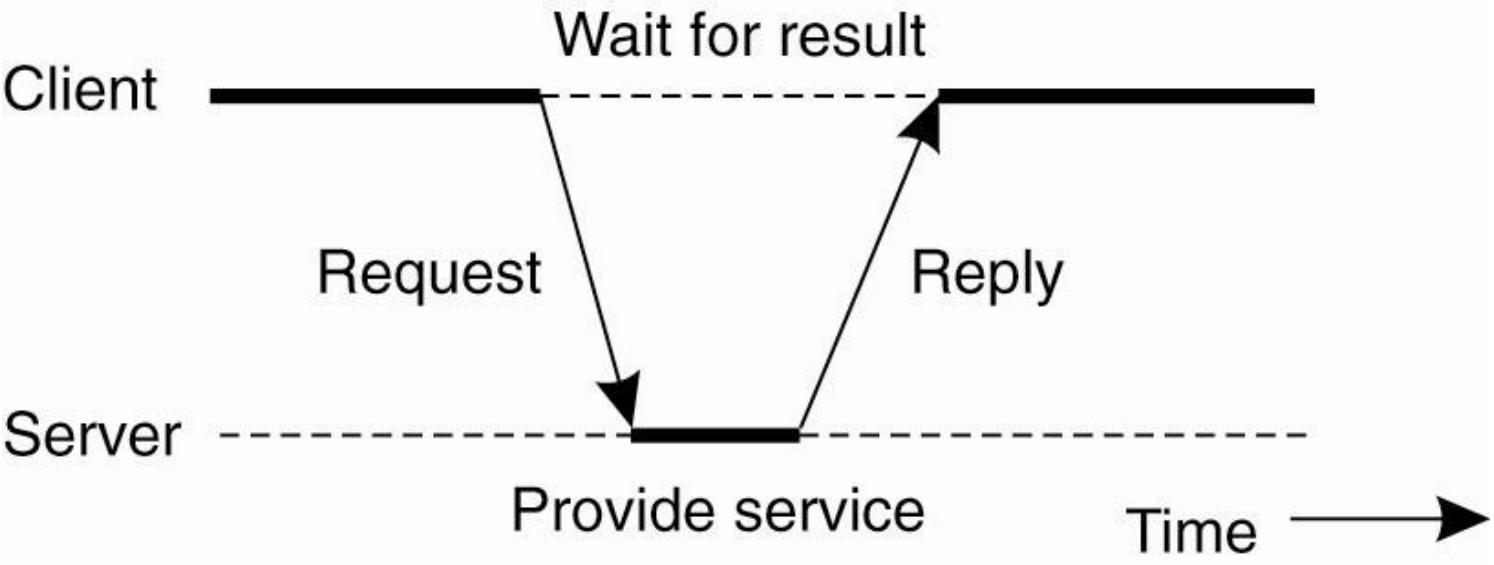
- **Centralized:** Most functionality is in a single machine
- **Distributed:** Functionality is spread across symmetrical machines
- **Hybrid:** Combination of the two

Centralized architecture

- Client-server
  - Client implements the user interface
  - Server has most of the functionality
    - Computation, data
  - E.g.: Web

Centralized architectures

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



Server design issues

Server organization; e.g., How to process client requests?

Client contact; e.g., how to contact end point (port)

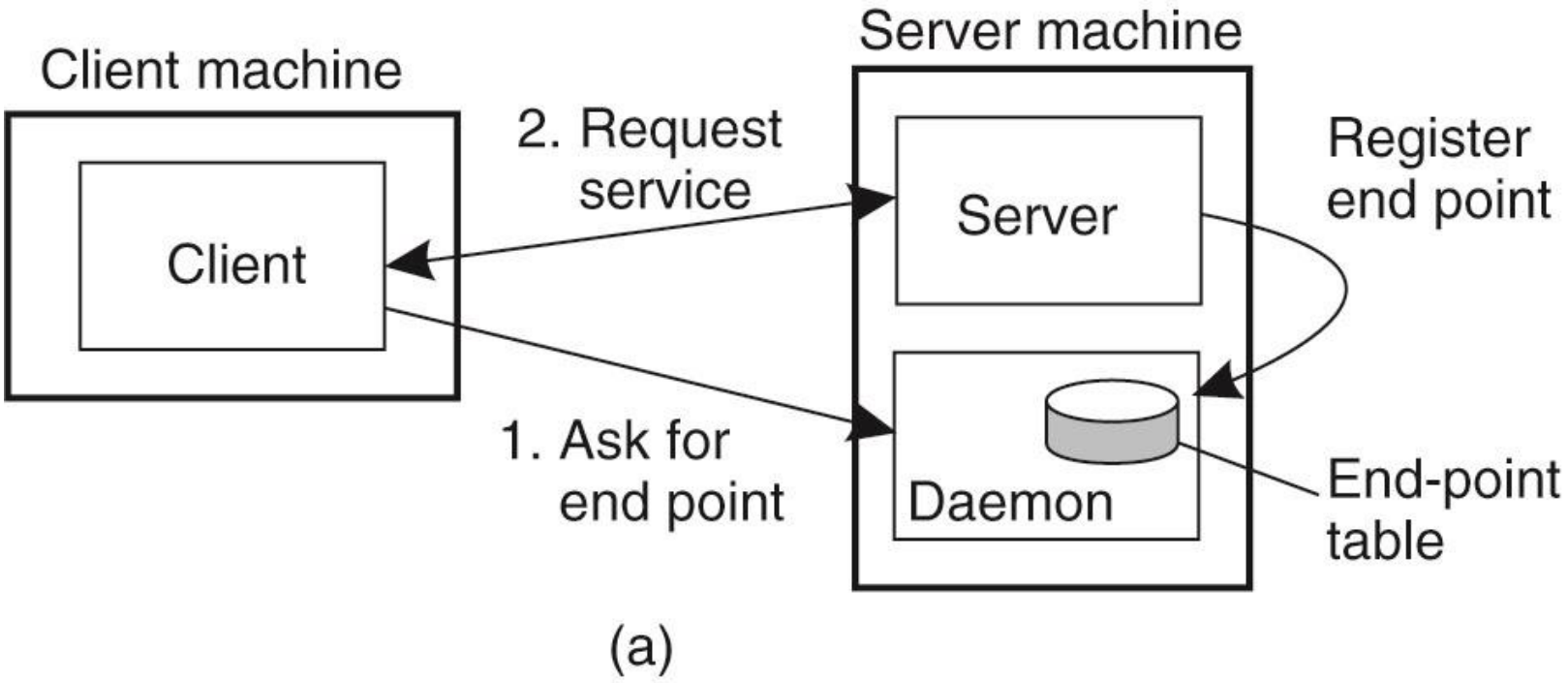
Server design issues

- Server organization; e.g., How to process client requests?
  - Iterative
  - Concurrent
    - Multithreaded
    - Fork (unix)
  - Stateless or stateful
- Client contact; e.g., how to contact end point (port)

Server design issues

- Server organization; e.g., How to process client requests?
  - Iterative
  - Concurrent
    - Multithreaded
    - Fork (unix)
  - Stateless or stateful
- Client contact; e.g., how to contact end point (port)
  - Well-known
  - Dynamic: daemon; superserver (unix)

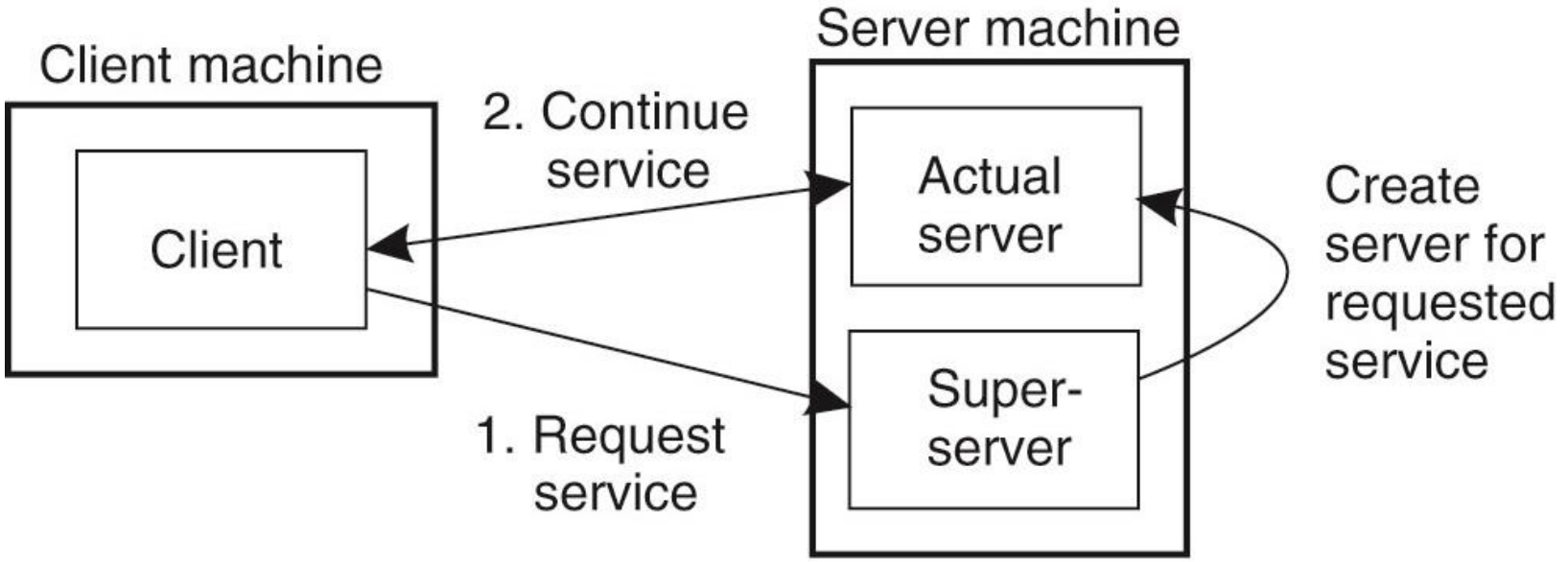
End point, general design issues



- Figure 3-11. (a) Client-to-server binding using a daemon.

End point, general design issues

Figure 3-11. (b) Client-to-server binding using a superserver.



(b)

Client-server architecture

- Application is vertically distributed
- Distribution along functionality
- Logically different component at different place

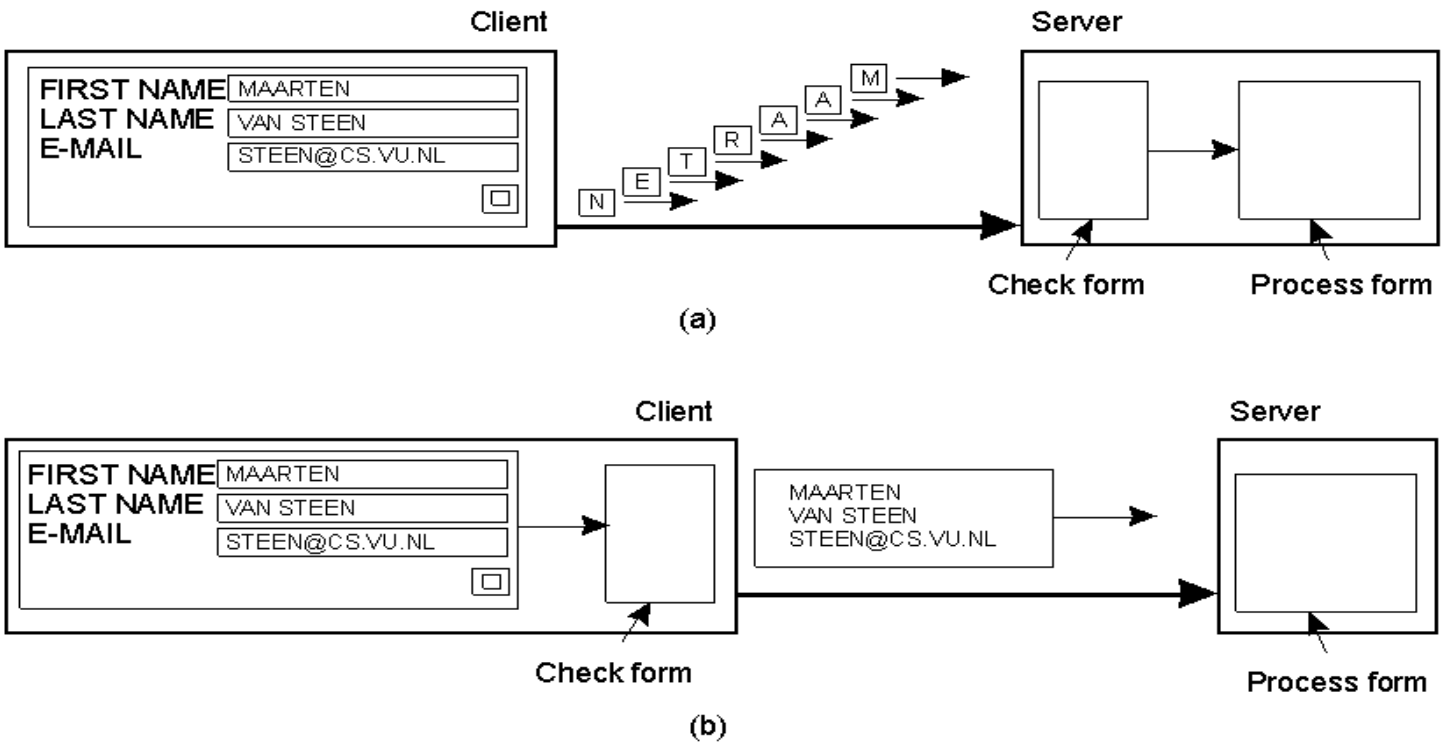
Decentralized architectures

- Vertical distribution
- Horizontal distribution
  - E.g., Peer-to-peer distribution

Component distribution

- Could have variations on component distribution
- Different amount of functionality between client-server
  - Only UI at client
  - UI+partial processing at client
  - UI+processing at client, data at server

Server offloading



The difference between letting:

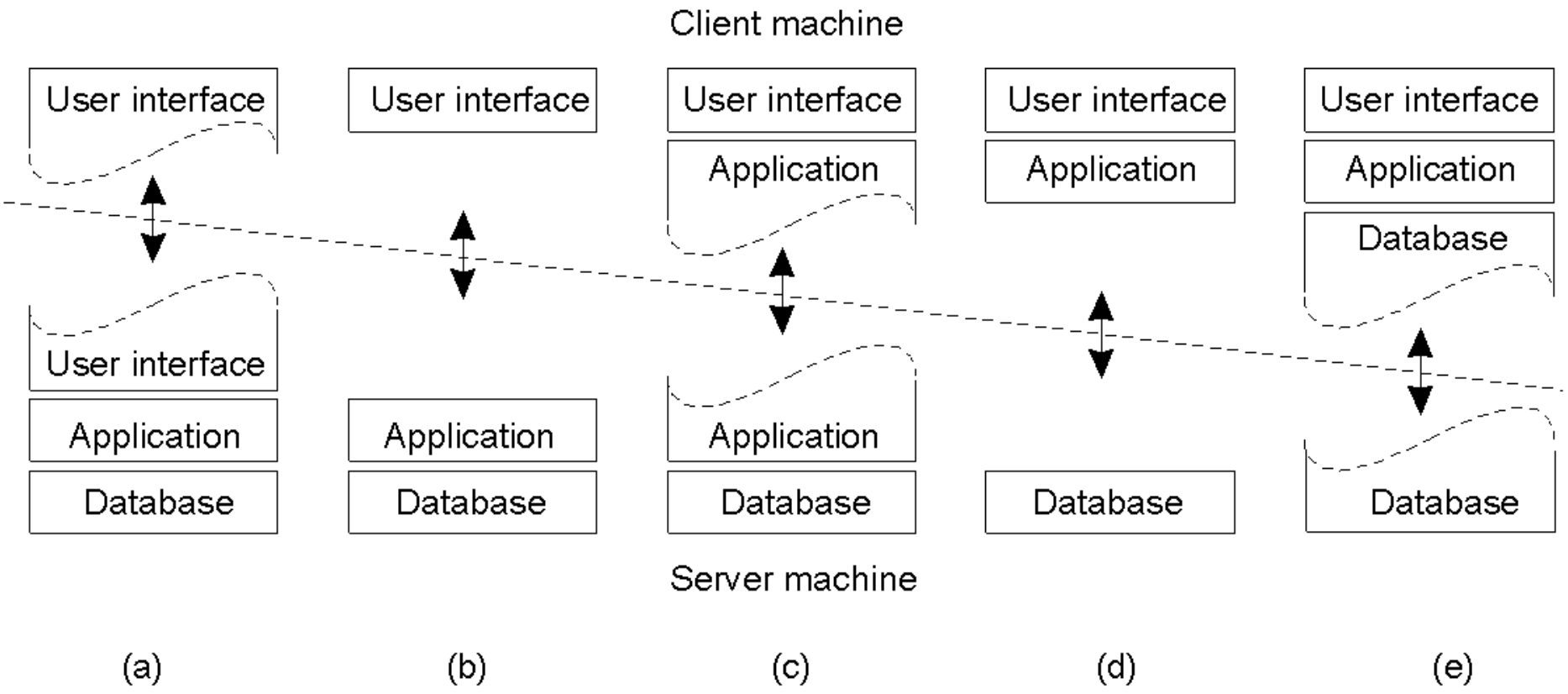
a) a server or

b) a client check forms as they are being filled

Multi-tiered servers

- Server may not be a single machine
- Multi-tiered architecture:
  - Front-end
  - Application server
  - Database

Physical two-tired architectures

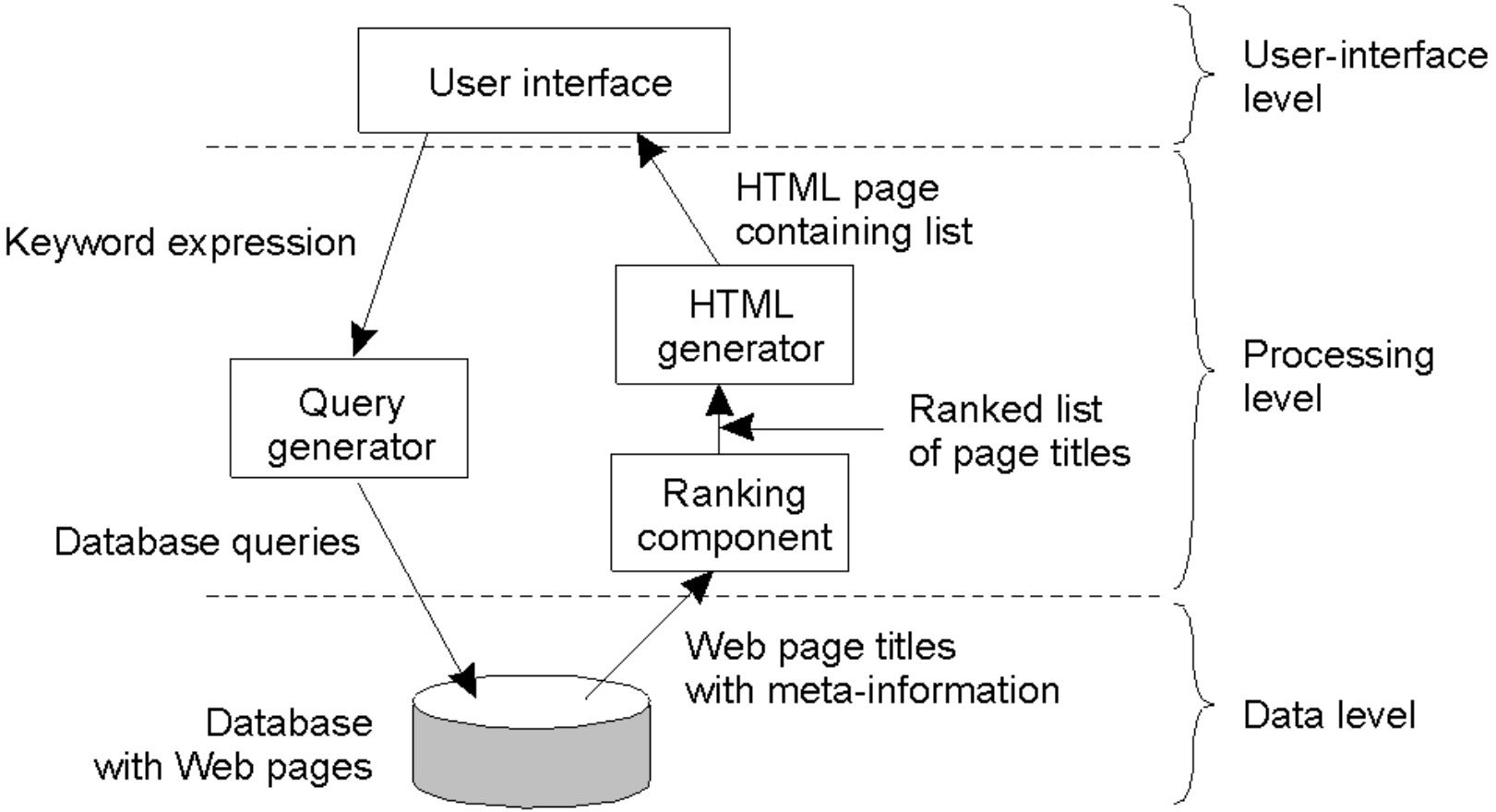


Alternative client-server organizations (a) – (e).

Application layering

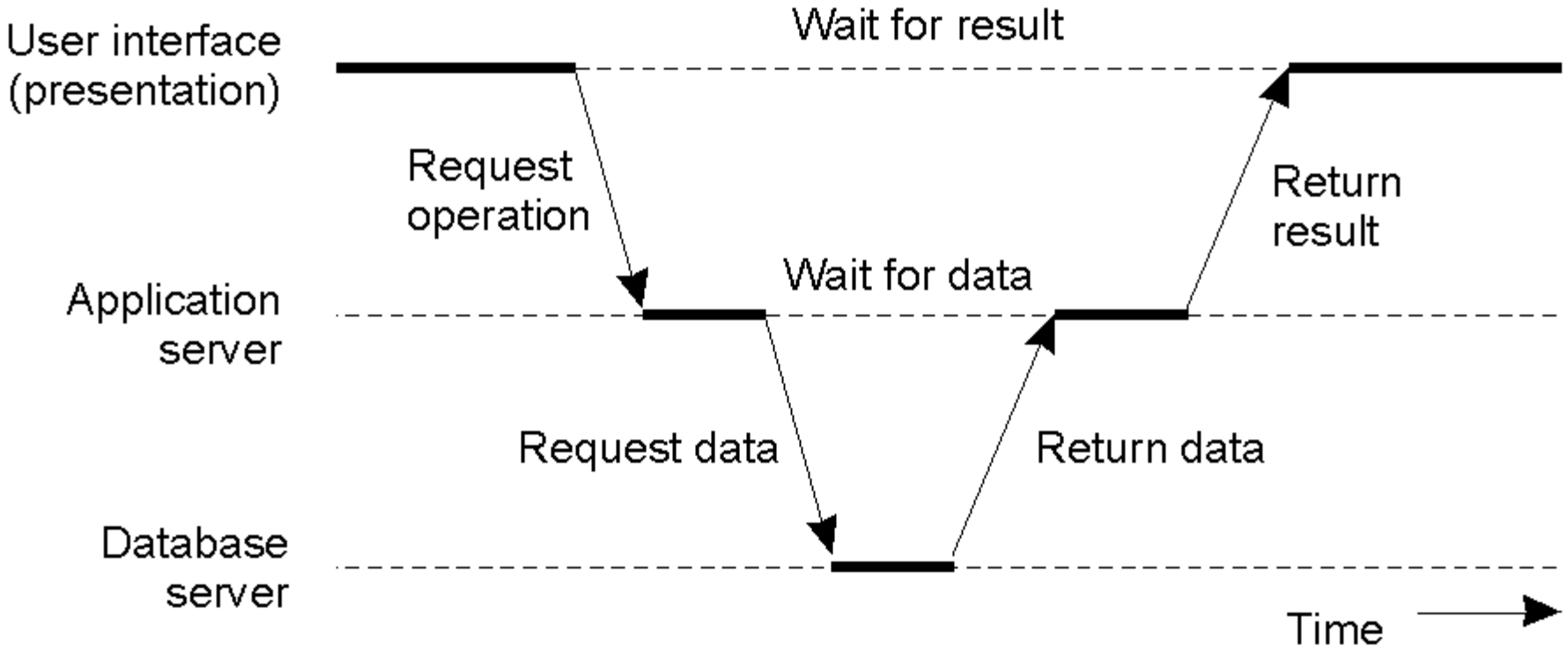
- The user-interface level
- The processing level
- The data level

Application layering



The general organization of an Internet search engine into three different layers

Multi-tiered architectures



An example of a server acting as a client.

Server clusters

- Replication of functionality across machines
  - Multiple front-ends, app servers, databases
- Client requests are distributed among the servers
  - Load balancing
  - Content-aware forwarding

Server clusters

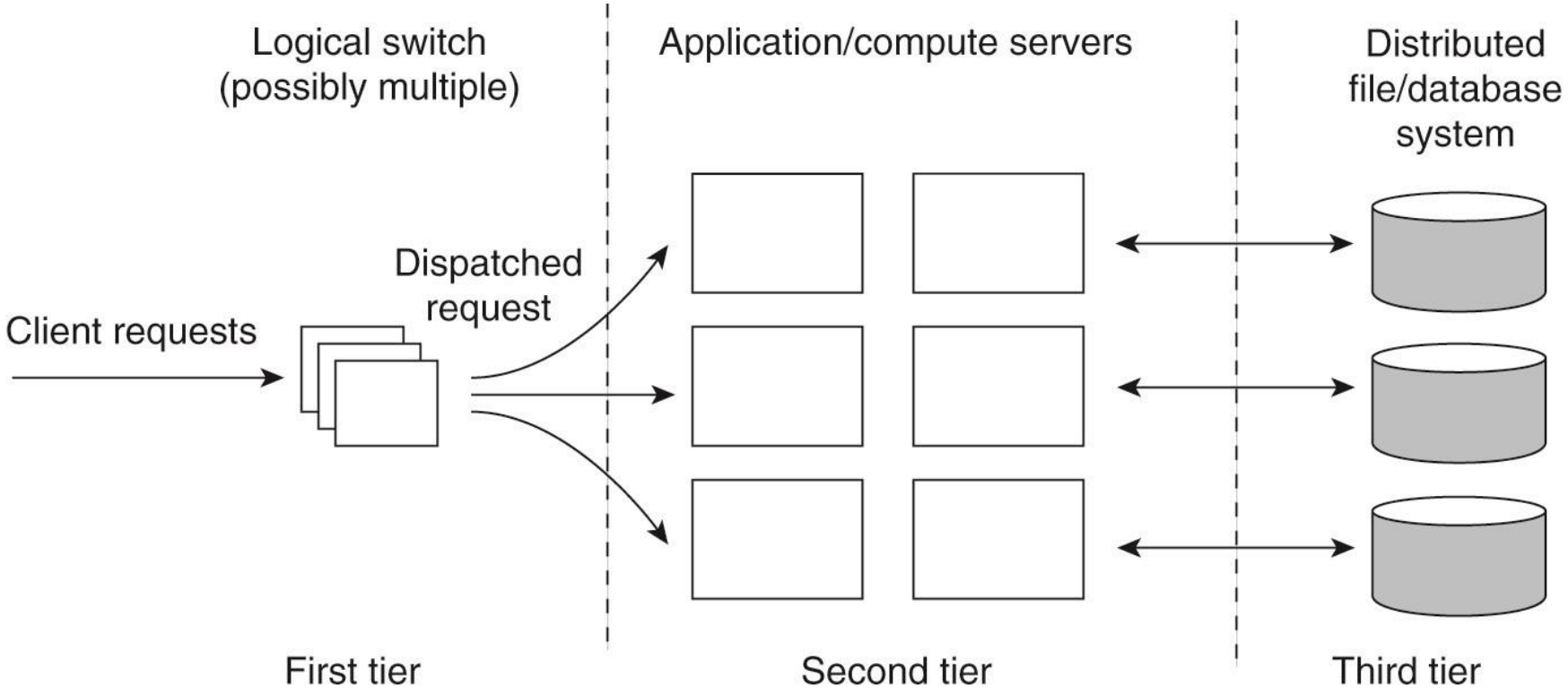


Figure 3-12. The general organization of a three-tiered server cluster.

Server clusters

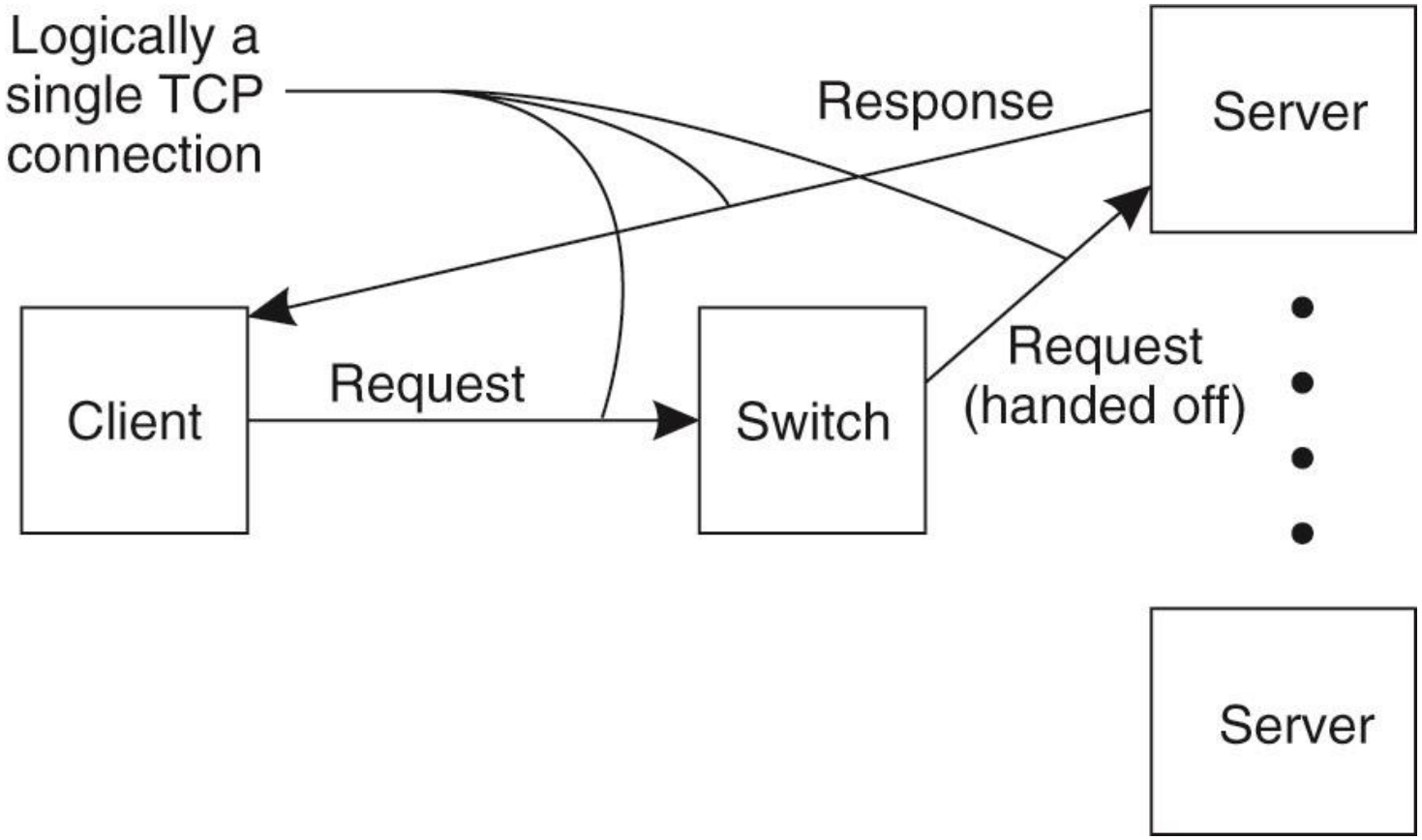
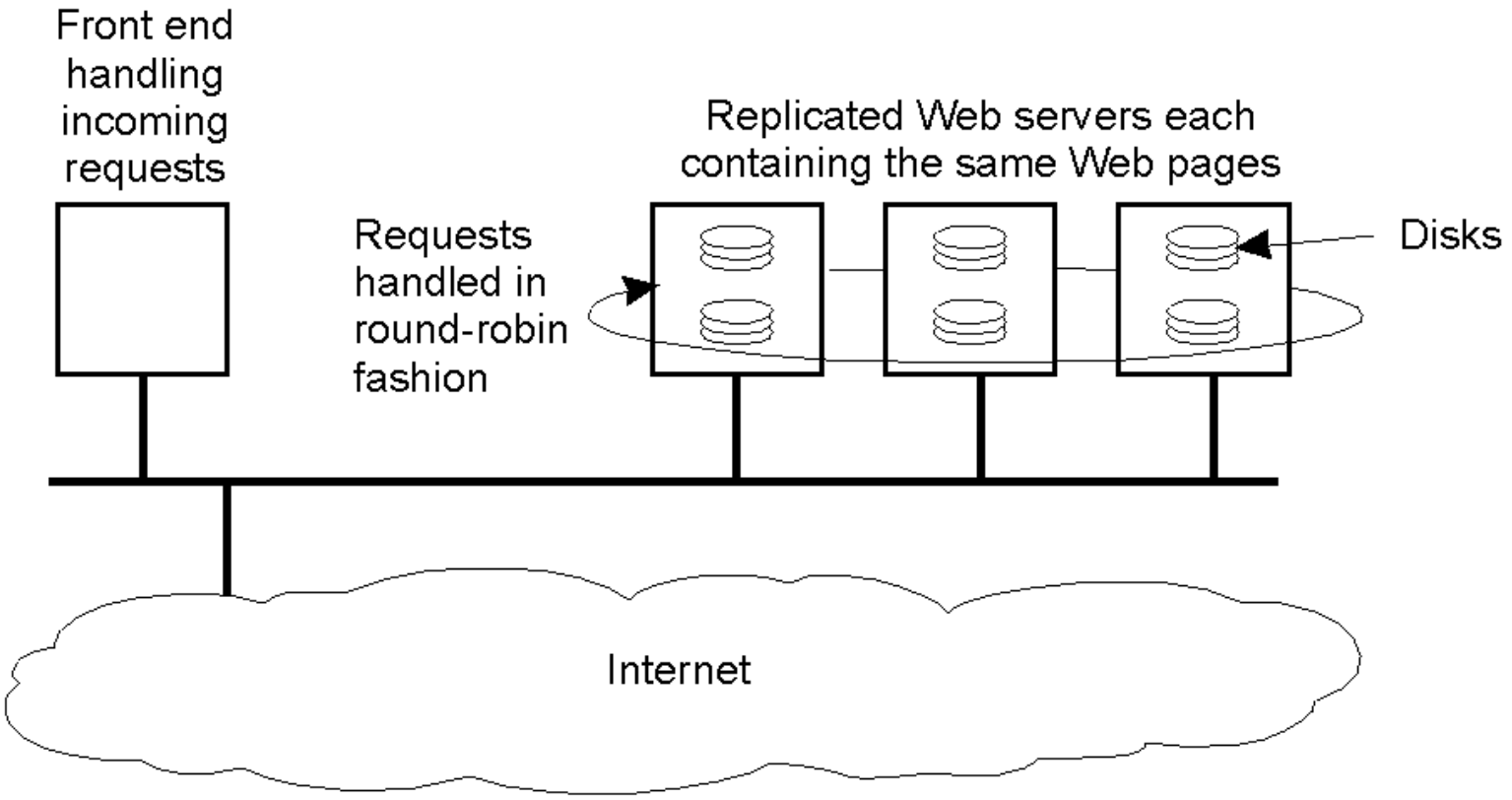


Figure 3-13. The principle of TCP handoff.

Modern Architectures



An example of horizontal distribution of a Web service.

Replica selection

- Round robin
- Load-based policies
- Payload-based methods (e.g., priorities)
- Energy/resource usage aware policies (e.g., costs)
- …

Decentralized architectures

- Horizontal distribution of application
- Each component is identical in functionality
- Differ in the portion of data they operate on
- E.g.: DNS, File-sharing, parallel processing

Hierarchical architectures

- Tree of nodes
- Centralized architecture between parent and children
- More scalable than a centralized architecture
  - Each node handles only part of the network

Peer-to-peer systems

- Each component is symmetric in functionality
- Peer: Combination of server-client
- How does a node find the other?
  - No “well-known” centralized server

Overlay networks

- A logical network consisting of participant components (processes/machines)
- Built on top of physical network
- Can be thought of as a graph
- Nodes are processes/machines, links are communication channels (e.g., TCP connections)

Types of peer-to-peer systems

- **Unstructured:** Built in a random manner
  - Each node can end up with any sets of neighbors, any part of application data
  - E.g.: Gnutella, Kazaa
- **Structured:** Built in a deterministic manner
  - Each node has well-defined set of neighbors, handles specific part of application data
  - E.g.: CAN, Chord, Pastry

Hybrid architectures

- Combination of centralized and distributed architectures
  - Some parts of the system organized as client-servers
  - Other parts organized in decentralized manner

Content distribution networks (CDNs)

- Provide localized content to users
- Decentralized set of content servers, may have P2P relationship
- Client-Server relation to the users
- E.g., Akamai

Collaborative distributed systems

- Work by user collaboration
- P2P in functionality
- Startup is done in a client-server manner
- E.g., Bittorrent, Napster

Other service model variations

- Multiple servers and caches (proxies)
- Mobile code
- Mobile agents
- Low-cost computers at client side (networked computers, and thin clients)
- Mobile devices
- …

More slides …

Unstructured peer-to-peer architectures

- Each node has a list of neighbors to which it is connected
- Communication to other nodes in the network happens through neighbors
- Neighbors are discovered in a random manner
- Exchange information with other nodes to maintain neighbor lists
- Application data is randomly spread across the nodes
- Flooding: To search for a specific item

Structured peer-to-peer architectures

- Nodes and data are organized deterministically
- Distributed Hash Tables (DHT)
  - Each node has a well-defined ID
  - Each data item also has a key
  - A data item resides in the node with nearest key
- Each node has information about neighbors in the ID space
- Searching for a data item:
  - Routing through the DHT overlay