TDTS04/TDDE35: 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.

Why study distributed systems?

- Understand the foundation of large-scale systems
- Understand tradeoffs when building large-scale systems
- Get knowledge useable in a broad set of applications
- Understand how the modern/connected world operates behind the scenes
- Appy knowledge of computer networking

Goals with these four lectures

- 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

?

Distributed systems

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

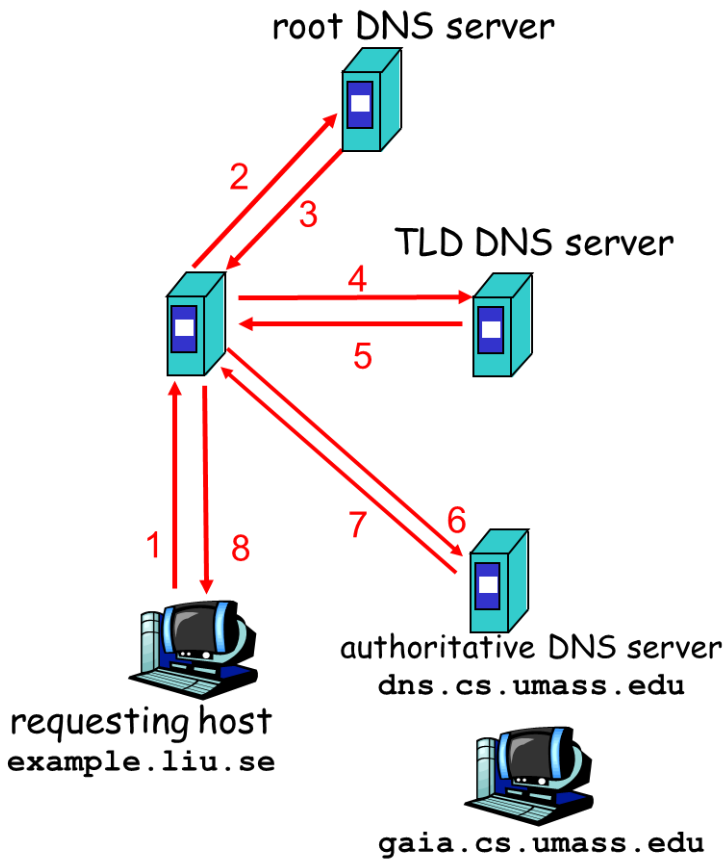
Distributed systems

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

- Examples include …

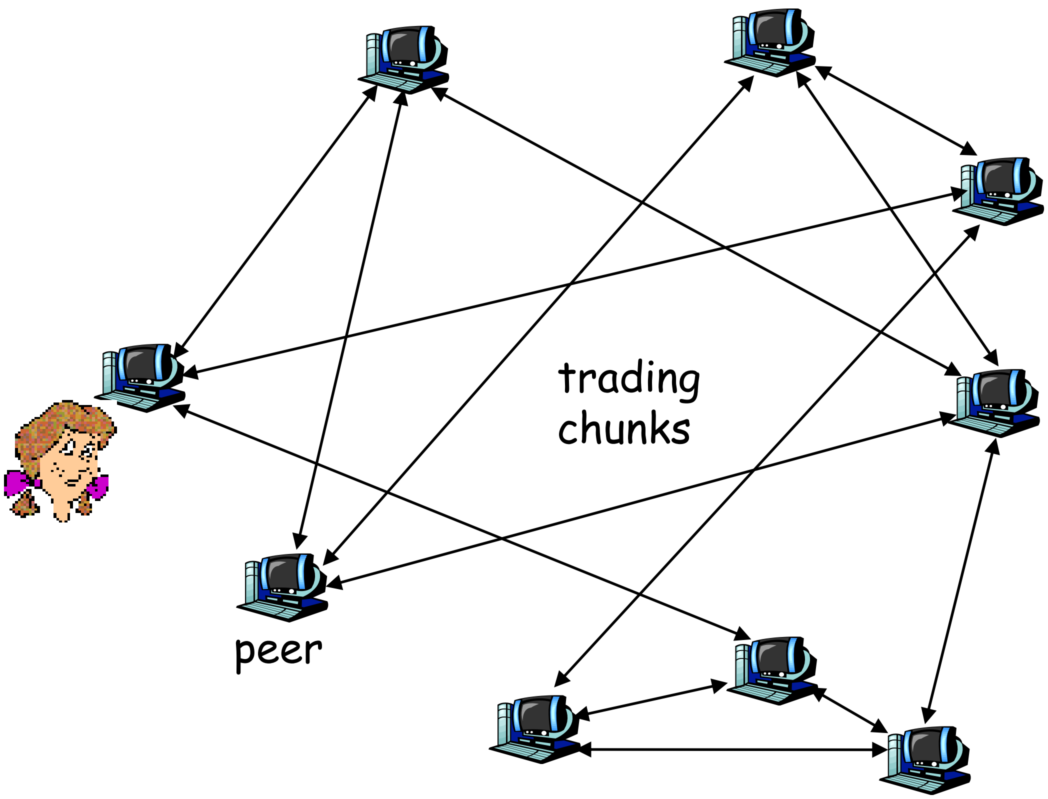
Distributed systems

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



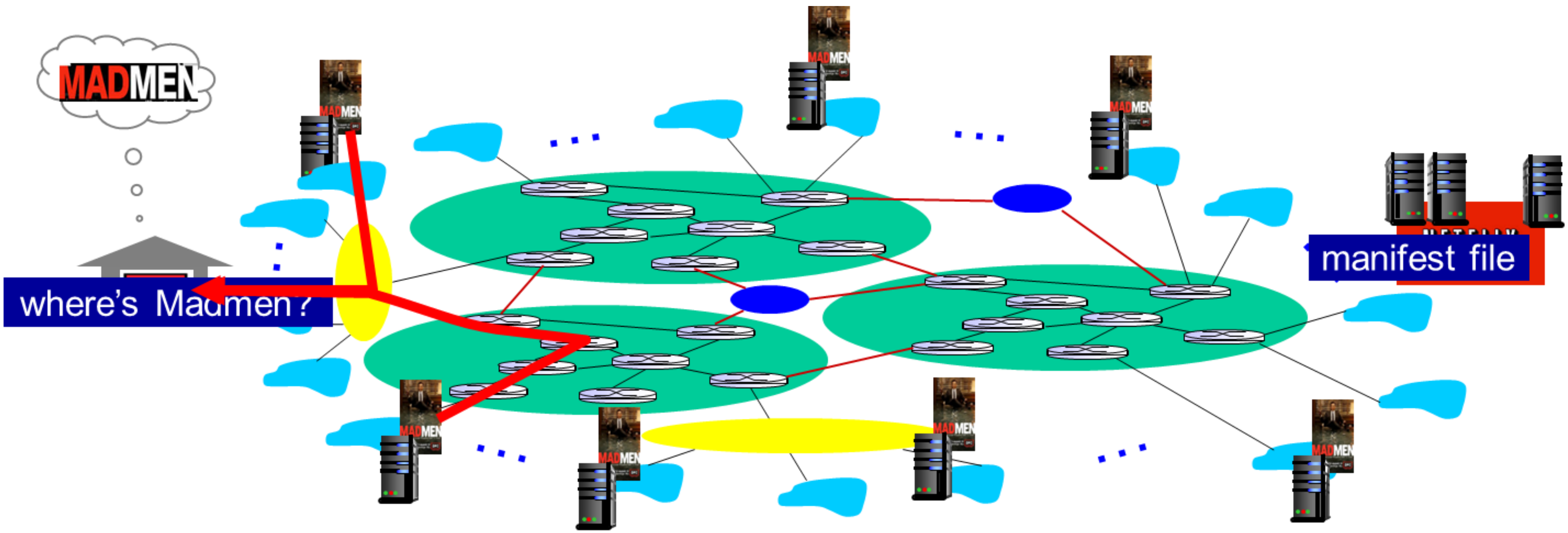
Distributed systems

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



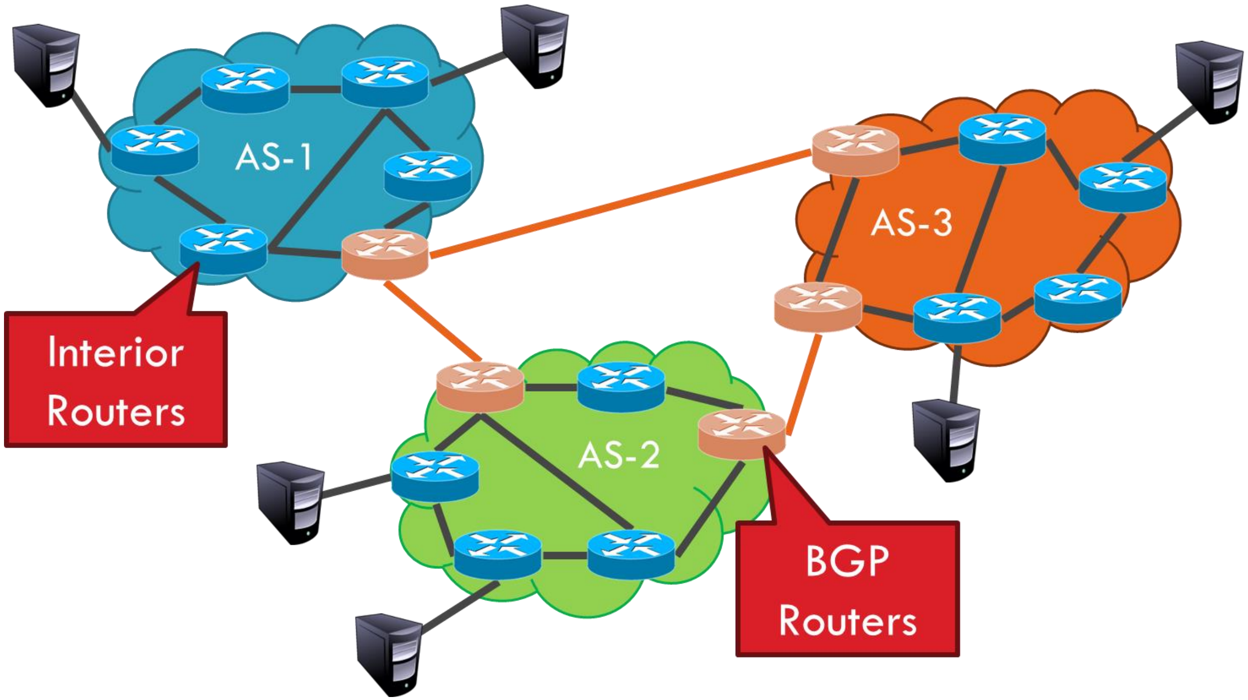
Distributed systems

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



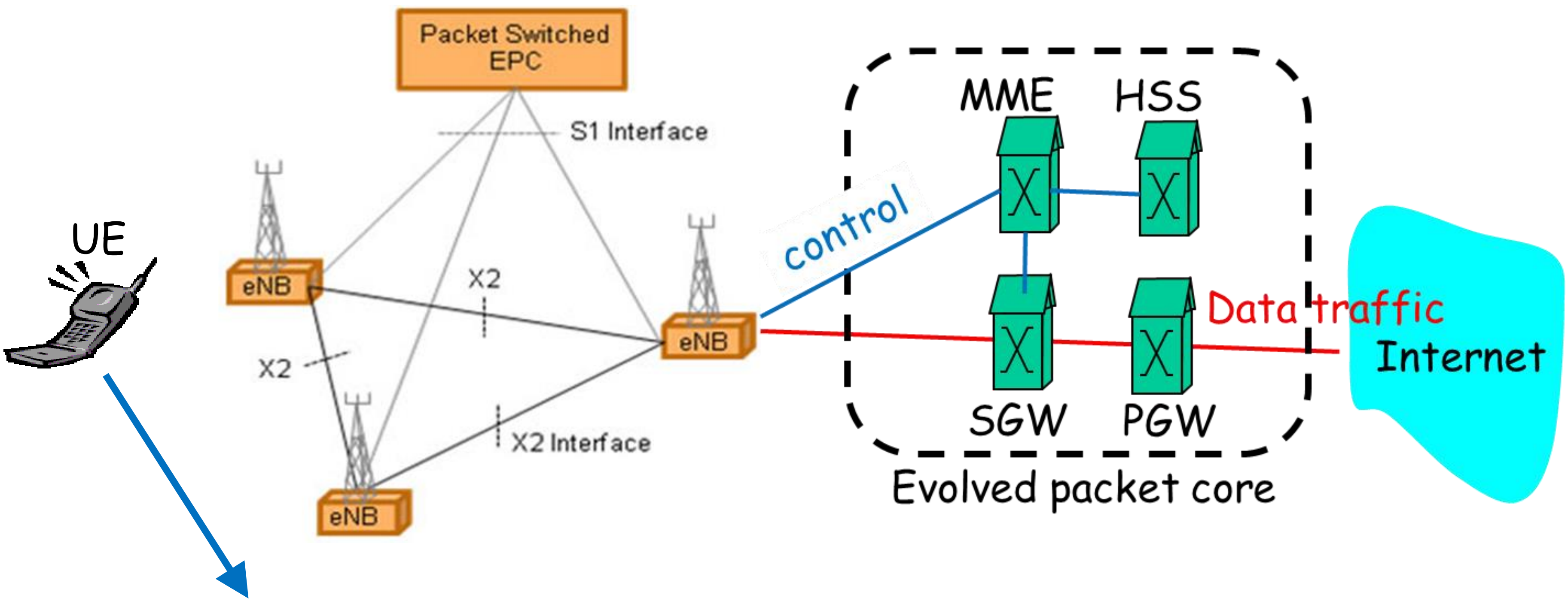
Distributed systems

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



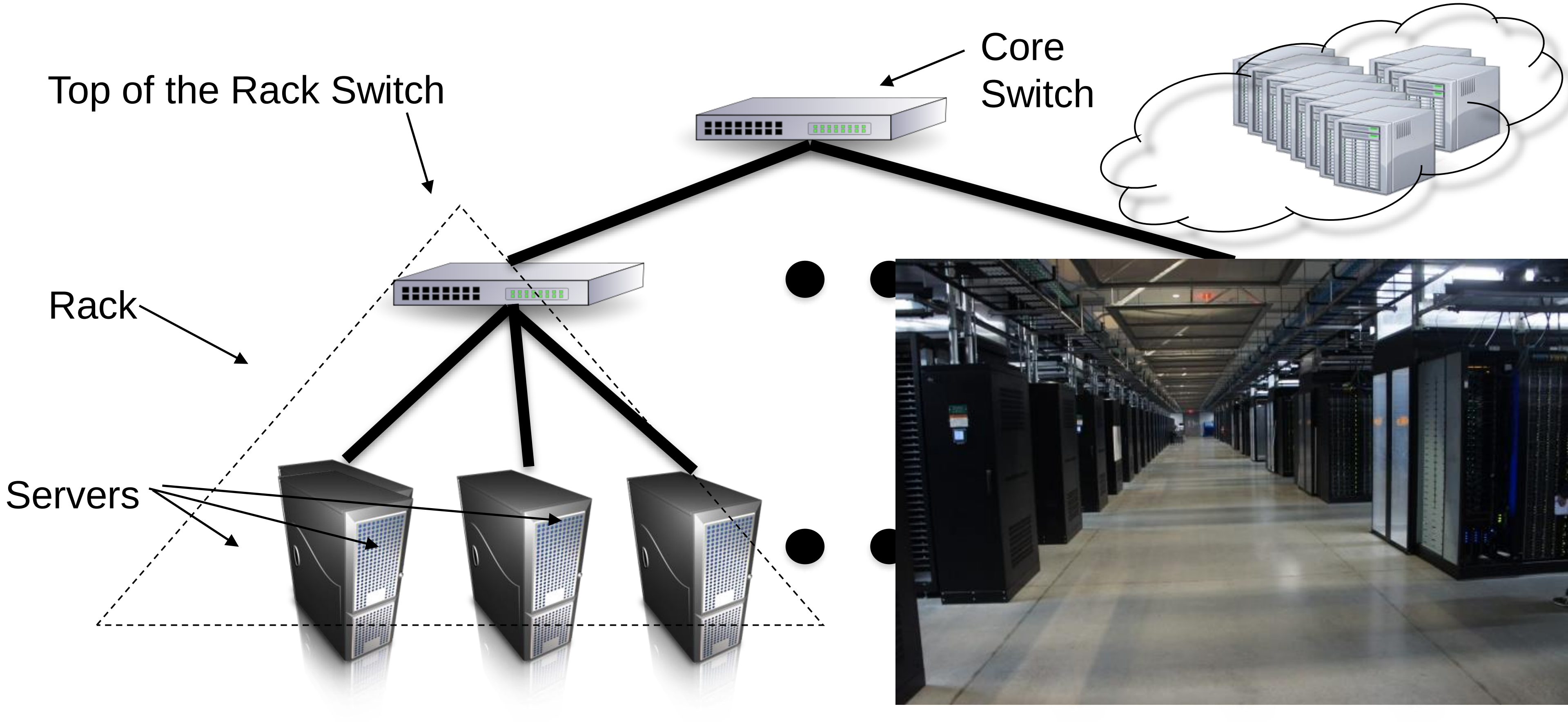
Distributed systems

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



Distributed systems

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



Distributed systems

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

- Examples include …
  - Web, web search, social networks, …
  - Peer-to-peer, file-sharing, …
  - Cloud services, scientific computing, …
  - Finance, healthcare, education, transportation/logistics, environmental engineering, entertainment/gaming, …
  - … (many many many more) ...

Distributed systems

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

- Networks of computers are everywhere!
  - Mobile phone networks
  - Corporate networks
  - Factory networks
  - Campus networks
  - Home networks
  - In-car networks
  - On-board networks in planes and trains
  - …

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 (e.g., application)
- Small vs large (full spectrum)
  - E.g., Multiprocessor vs. world-wide

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

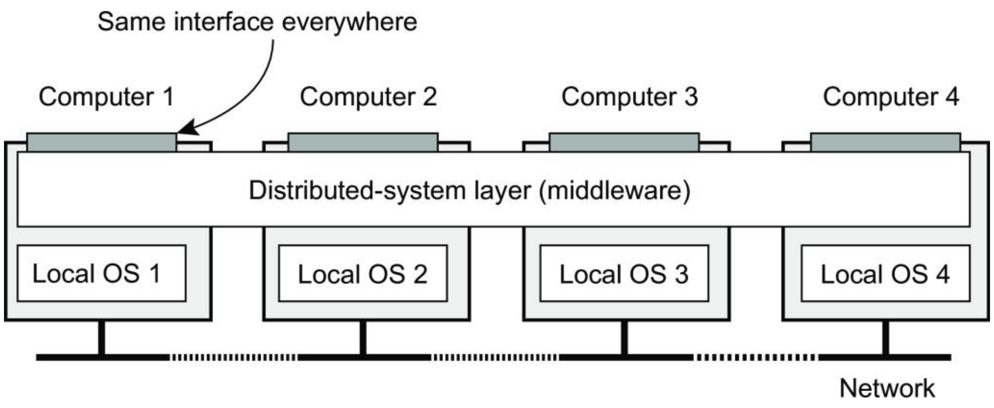
Common Distributed Systems Design Goals

- Heterogeneity – can the system handle a large variety of types of PCs and devices?
- Robustness – is the system resilient to host crashes and failures, and to the network dropping messages?
- Availability – are data+services always there for clients?
- Transparency – can the system hide its internal workings from the users?
- Concurrency – can the server handle multiple clients simultaneously?
- Efficiency – is the service fast enough? Does it utilize 100% of all resources?
- Scalability – can it handle 100 million nodes without degrading service? (nodes=clients and/or servers) How about 6 B? More?
- Security – can the system withstand hacker attacks?
- Openness – is the system extensible?
- Reliability – is the system available and fault tolerant?

Some examples …
Sharing (heterogeneity, openness, …)

- Multiple users can share + 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

Distribution transparency



What is transparency?

The phenomenon by which a distributed system attempts to hide the fact that its processes and resources are physically distributed across multiple computers, possibly separated by large distances.

Observation

Distribution transparency is handled through many different techniques in a layer between applications and operating systems: a middleware layer

Distribution transparency

Types

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 to another location
Replication	Hide that an object is replicated
Concurrency	Hide that an object may be shared by several independent users
Failure	Hide the failure and recovery of an object

Degree of transparency

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

Degree of transparency

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 goal, but achieving it is a different story, and it should often not even be aimed at.

Scalability

- Allow the system to become bigger without negatively affecting performance

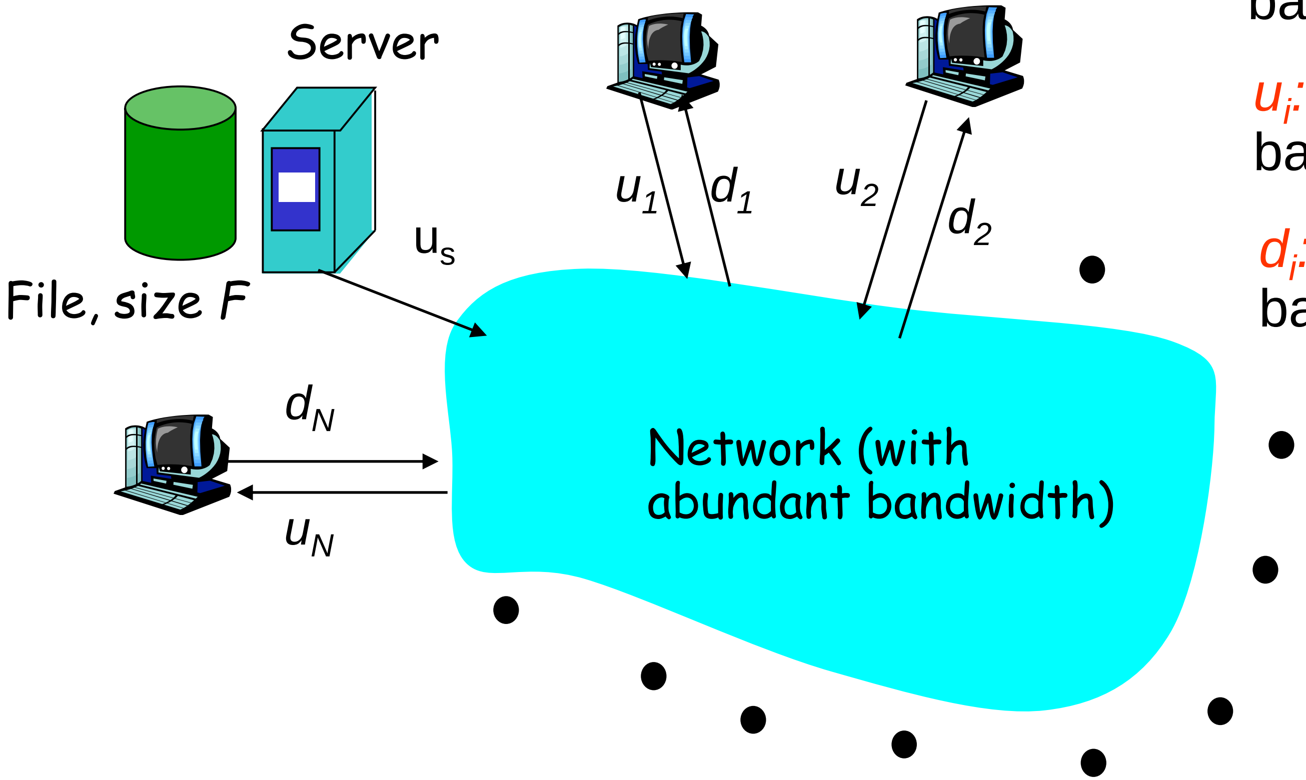
- Multiple dimensions:
  - **Size:** Adding more resources and users
  - **Geographic:** Dispersed across locations
  - **Administrative:** Spanning multiple administrative domains

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: Client-server vs P2P

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



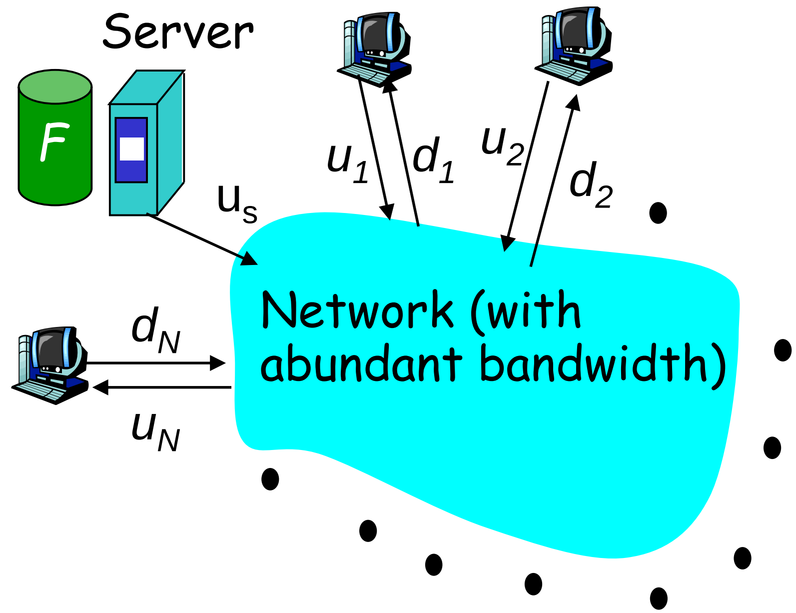
u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

File distribution time: Client-server

- server must upload N copies:
  - NF/u_s time
- client i takes F/d_i time to download

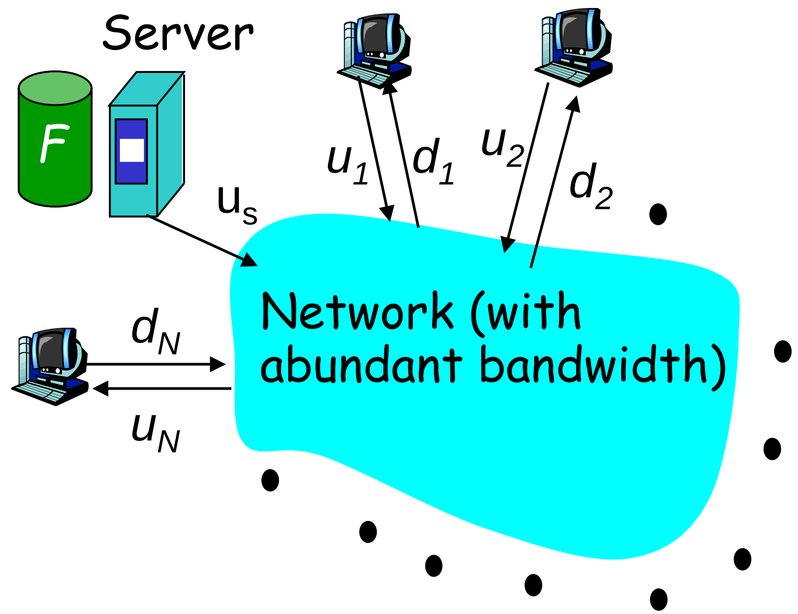


Time to distribute F to N clients using client/server approach $= d_{cs} = \max_i \{ NF/u_s, F/\min_i(d_i) \}$

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)

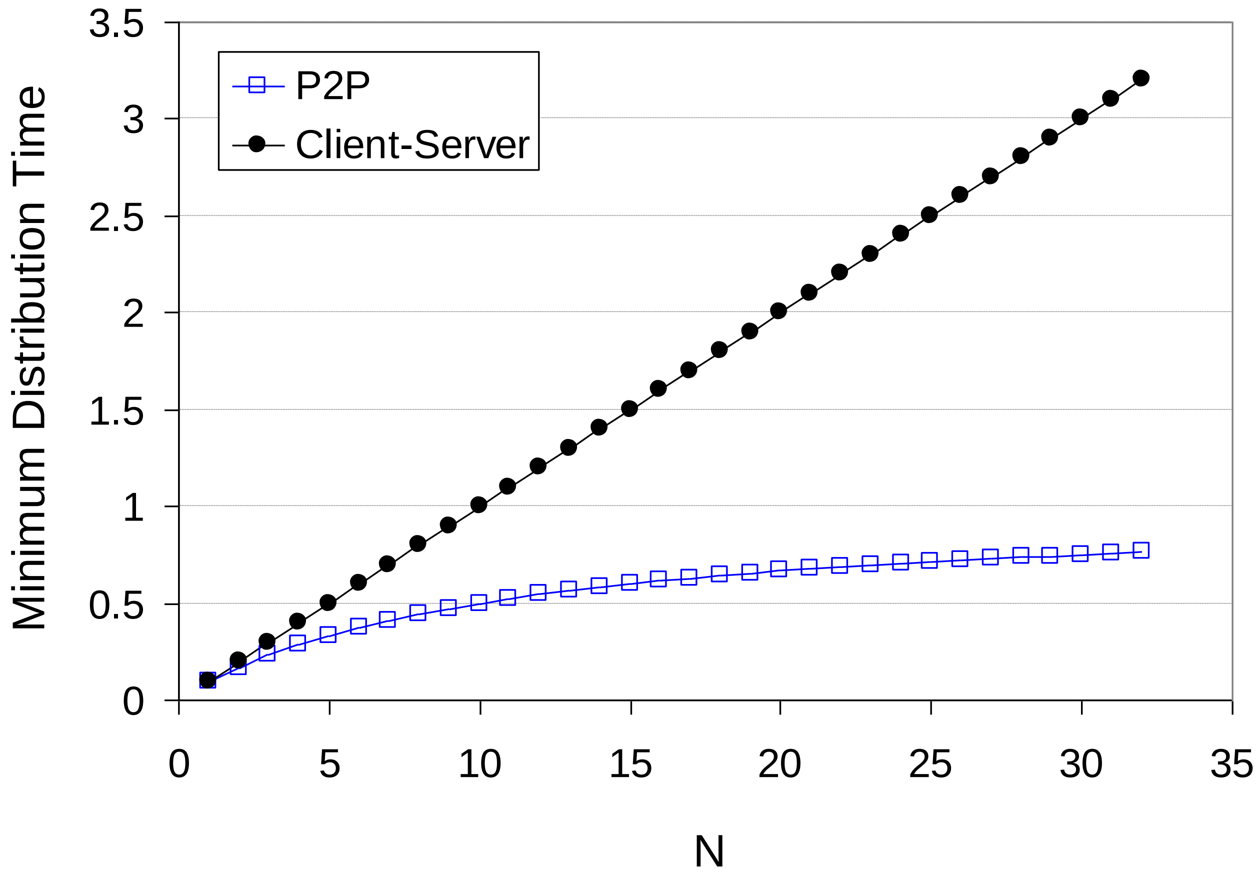


- fastest possible upload rate: $u_s + \sum u_i$

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

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)

Distributed systems

- Remember the goals just discussed ...
  - Heterogeneity, Robustness, Availability, Transparency, Concurrency, Efficiency, Scalability, Security, Openness, Reliability, …

Question: What complicates these goals?

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

Perspectives on distributed systems

Distributed systems are complex: take perspectives

- Architecture: common organizations
- Process: what kind of processes, and their relationships
- Communication: facilities for exchanging data
- Coordination: application-independent algorithms
- Naming: how do you identify resources?
- Consistency and replication: performance requires of data, which need to be the same
- Fault tolerance: keep running in the presence of partial failures
- Security: ensure authorized access to resources

Let’s organize …

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

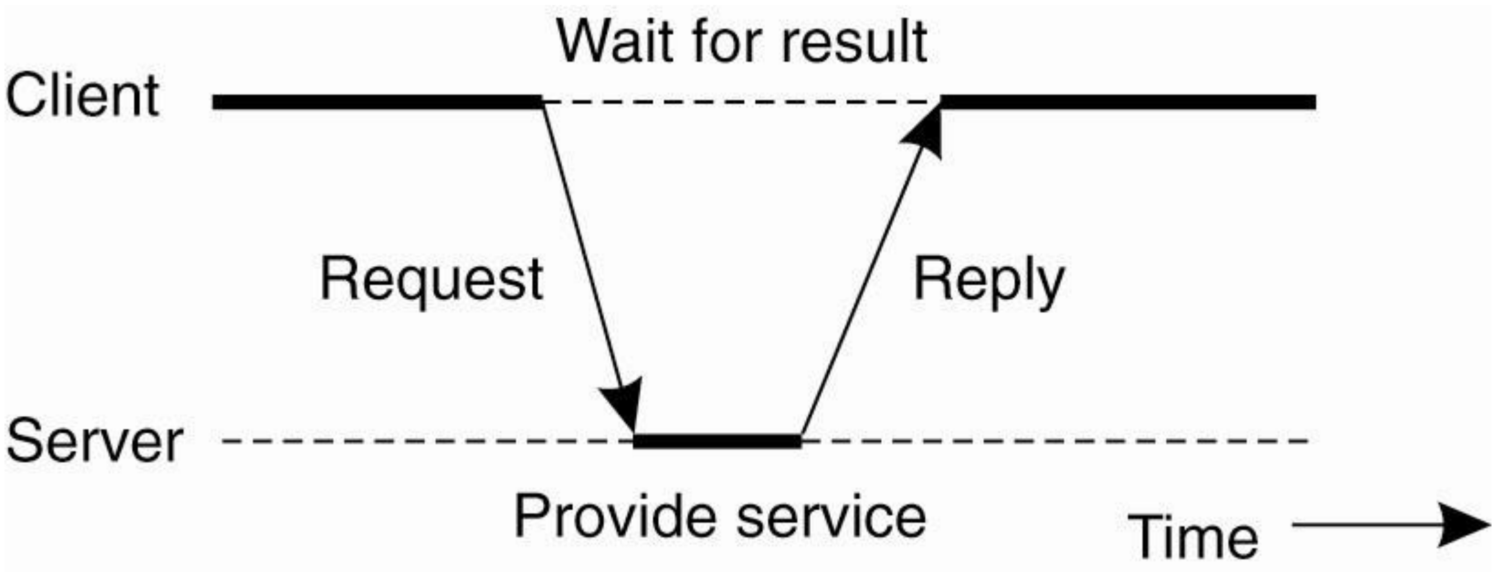
- **Centralized:** Most functionality is in a single machine
- **Distributed:** Functionality is spread across symmetrical machines
- **Hybrid:** Combination of the two

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.



Decentralized architectures

- Vertical distribution
  - Distribution along functionality
- Horizontal distribution
  - E.g., Peer-to-peer distribution

Client-server architecture

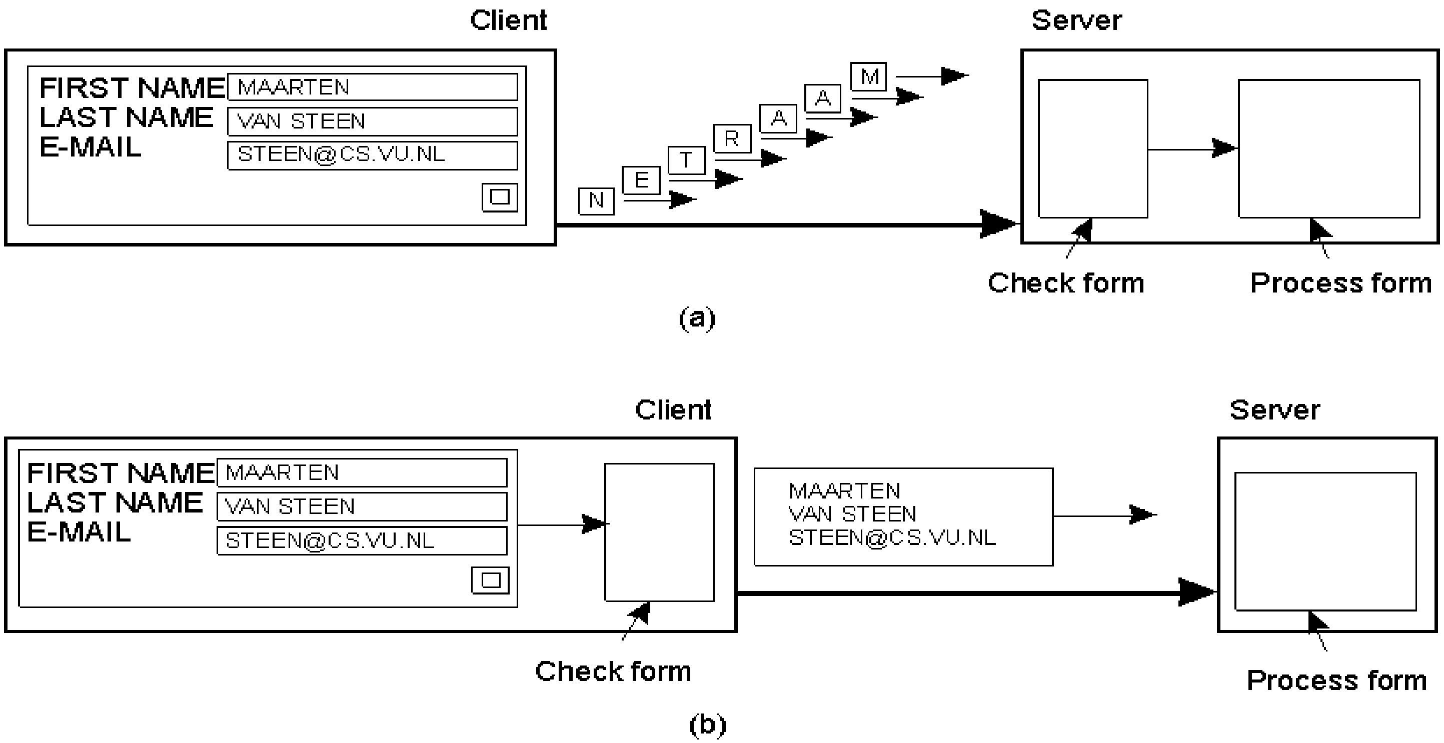
- Application is vertically distributed
  - Distribution along functionality

- Logically different component at different place

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

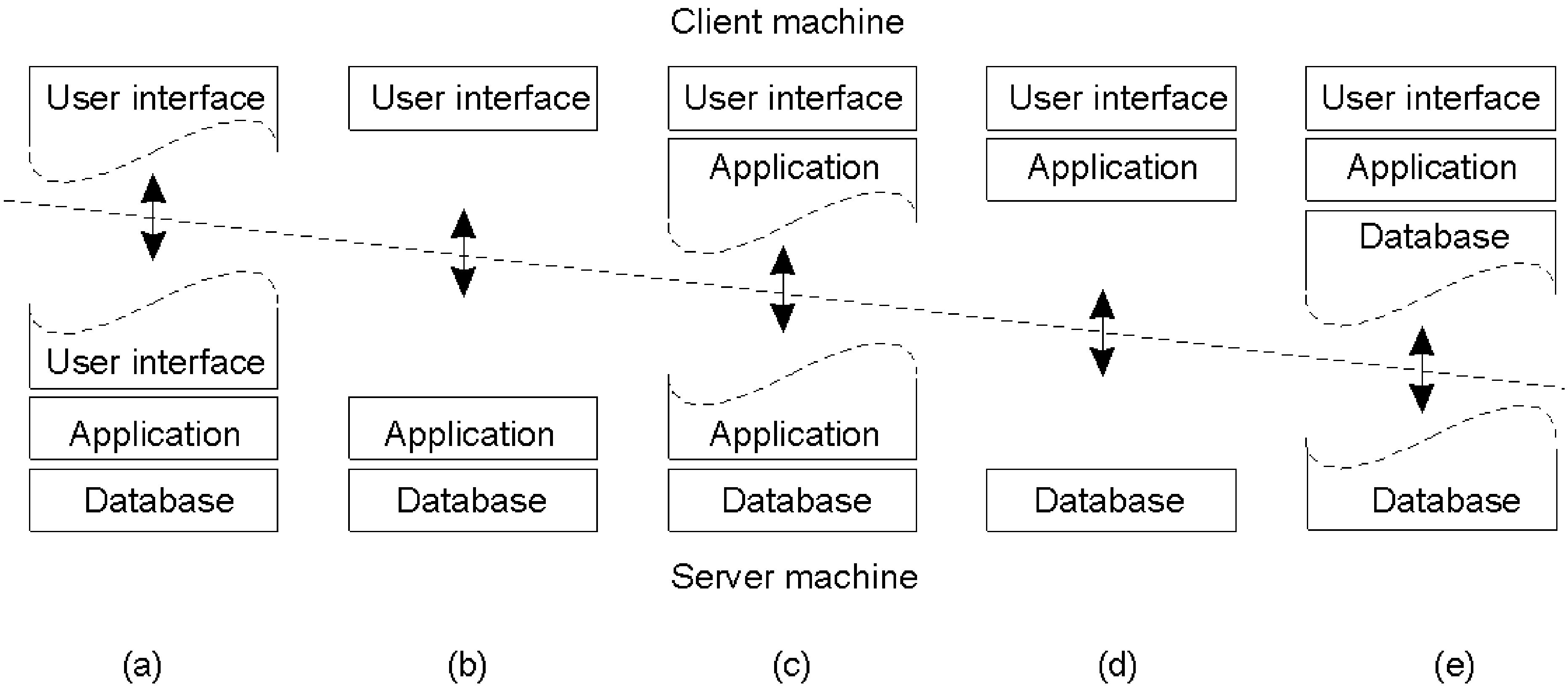


The difference between letting:

a) a server or

b) a client check forms as they are being filled

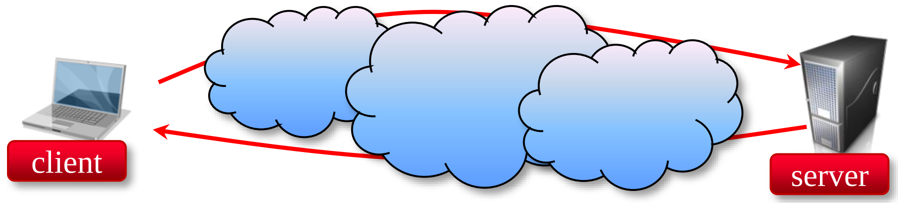
Physical two-tired architectures



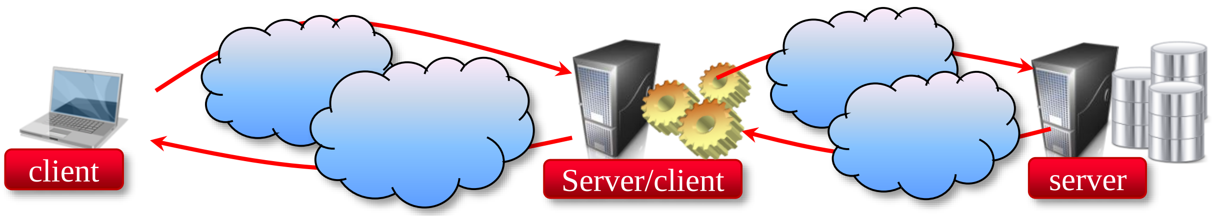
Alternative client-server organizations (a) – (e).

Client-Server Architecture (Tiered architecture)

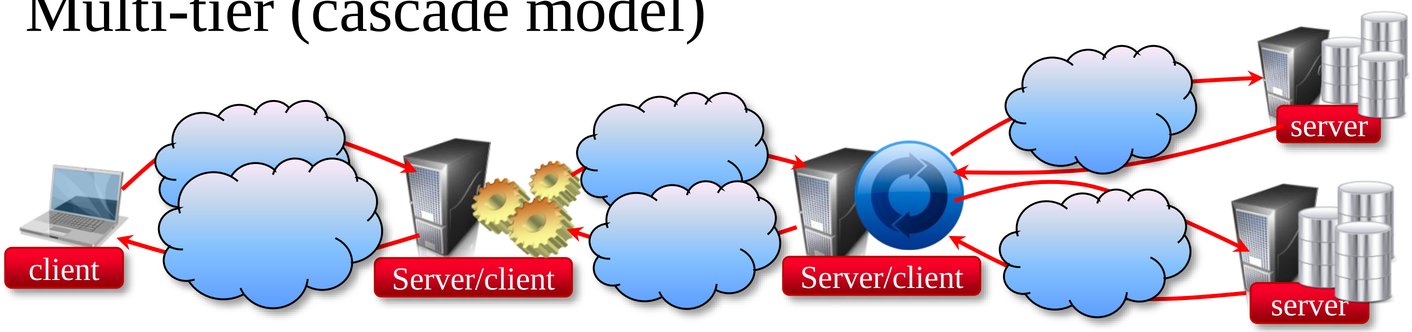
- Two-tier model (classic)



- Three-tier (when the server, becomes a client)



- Multi-tier (cascade model)



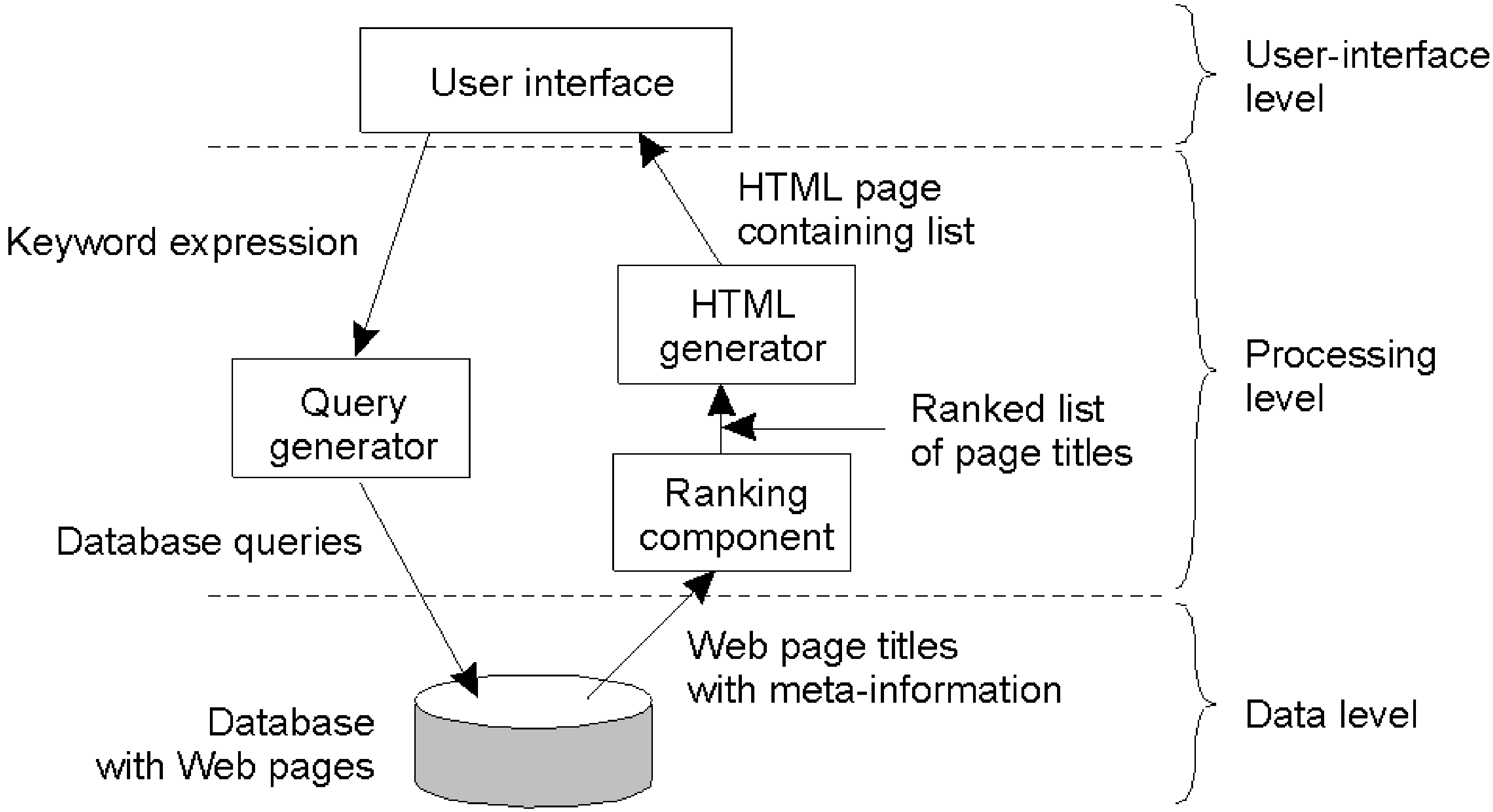
Multi-tiered servers

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

Application layering

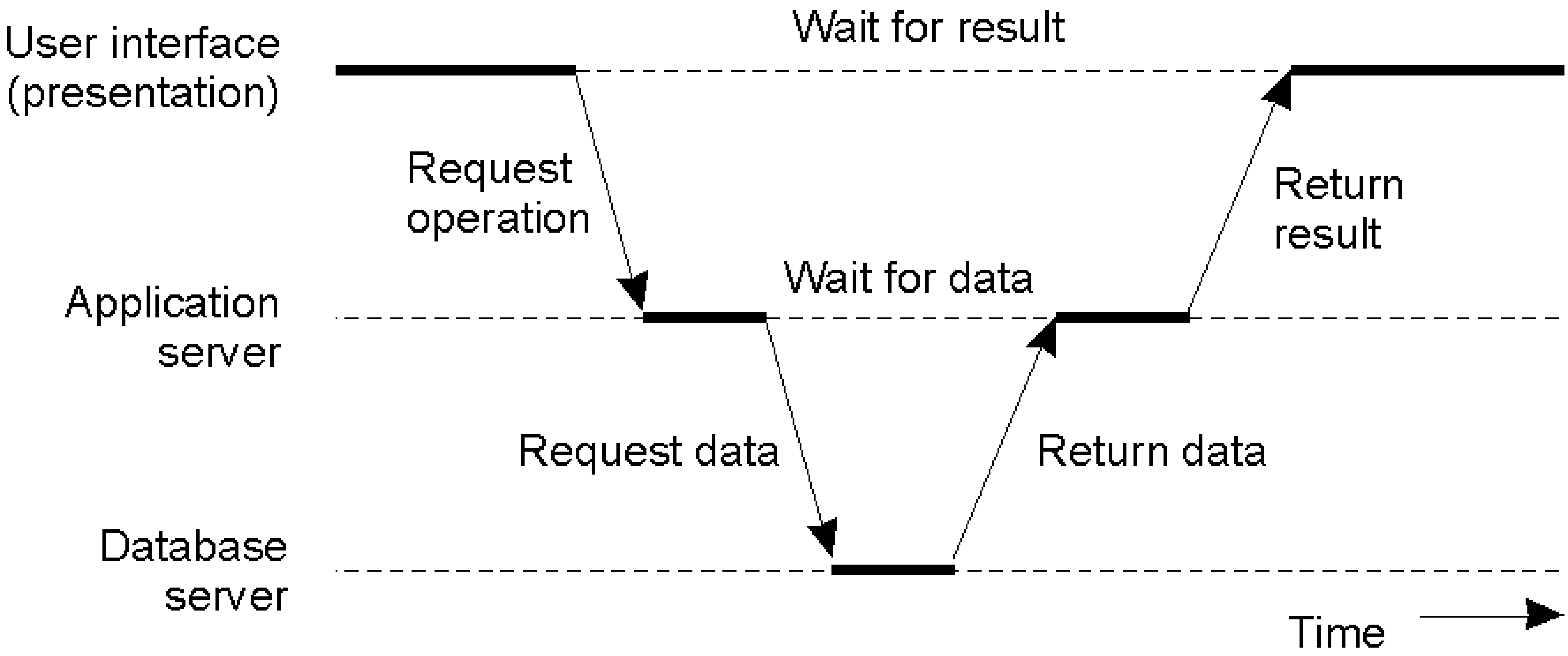
- The user-interface level
- The processing level
- The data level

Application layering



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

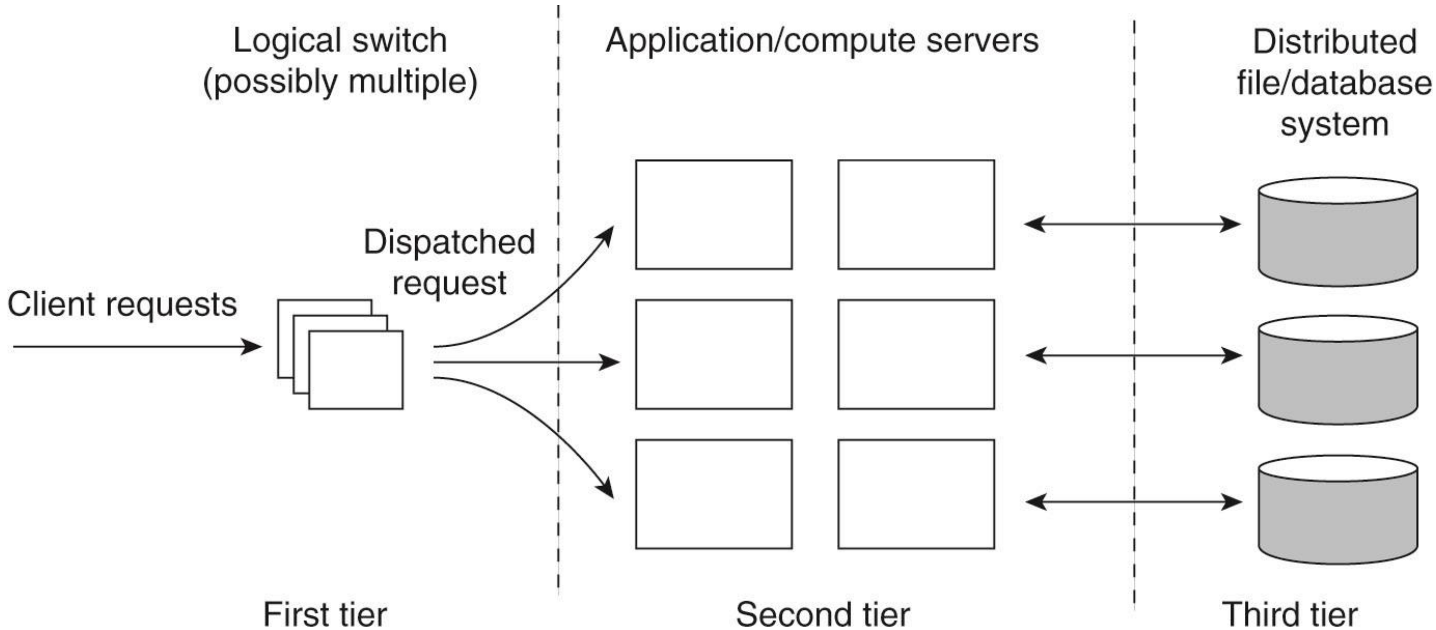
Multi-tiered architectures



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

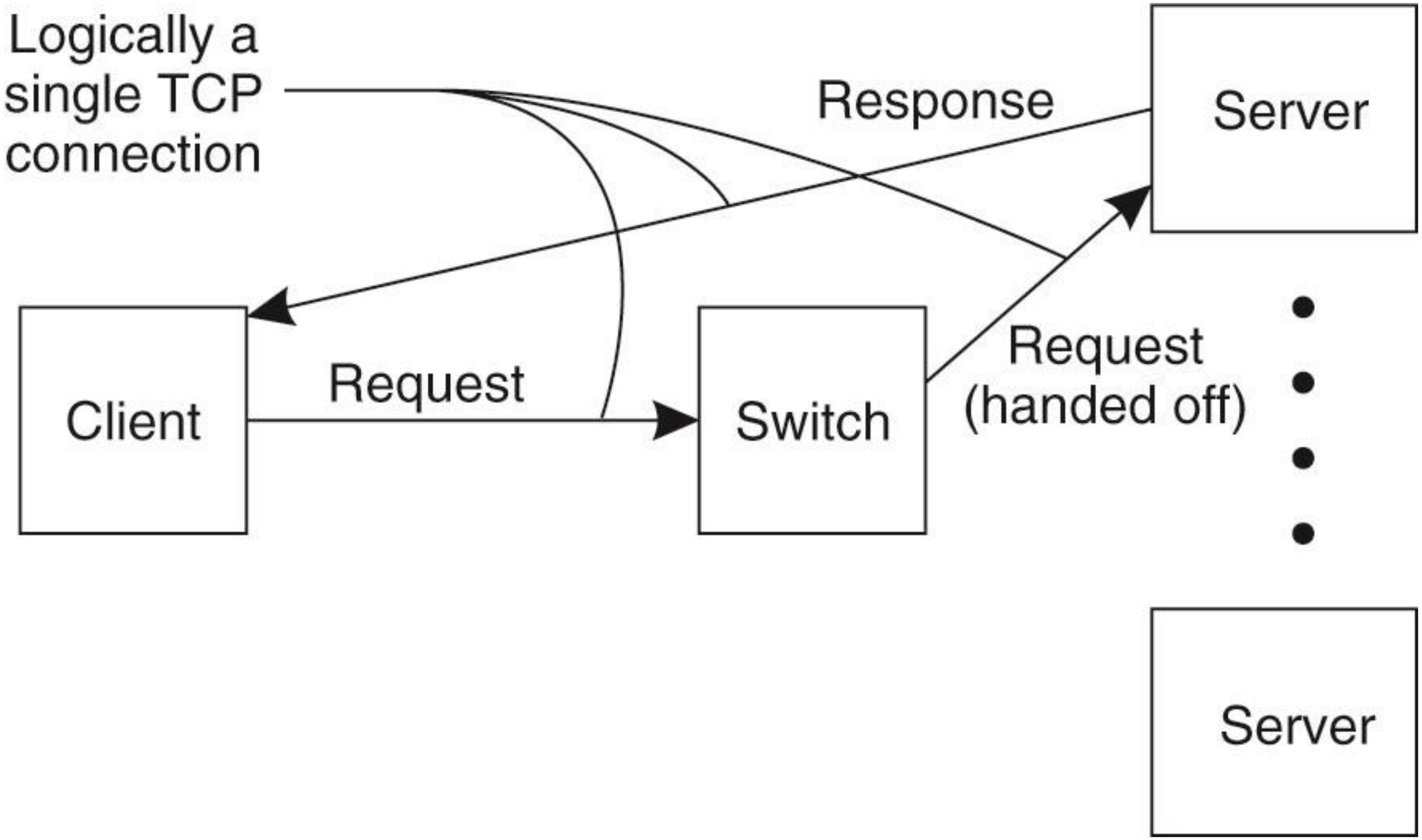
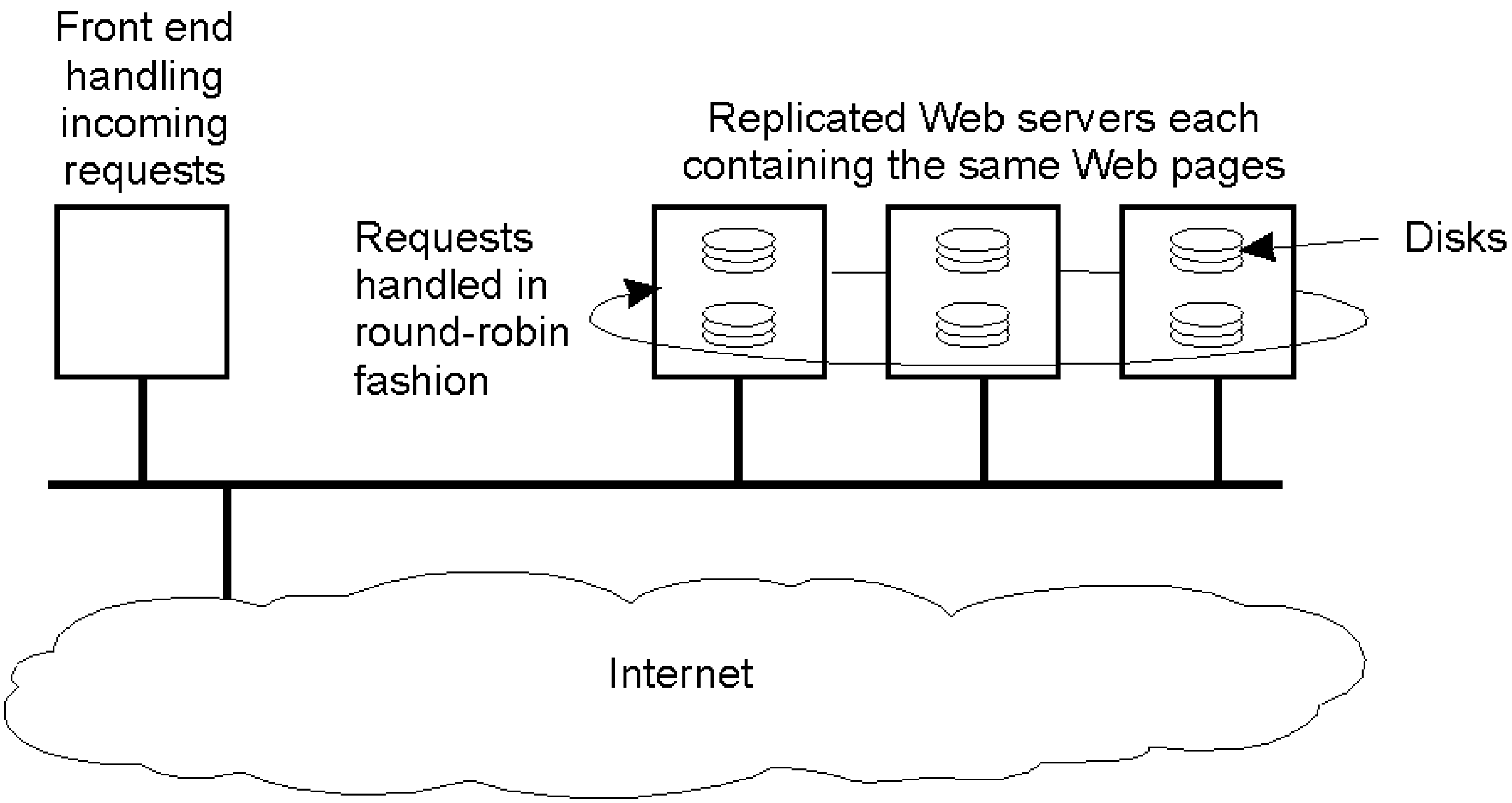


Figure 3-13. The principle of TCP handoff.

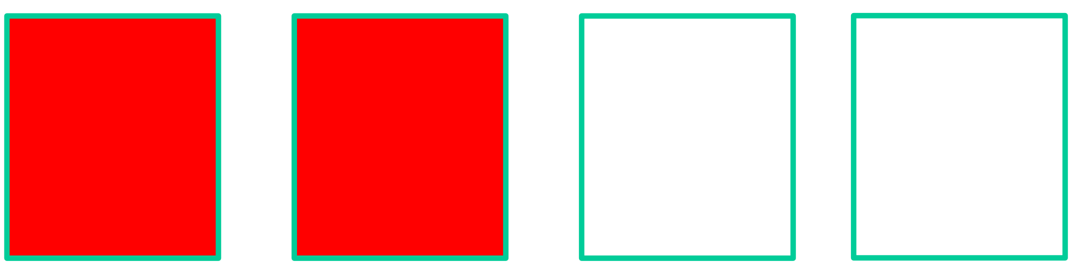
Modern Architectures



An example of horizontal distribution of a Web service.

Replica selection Examples

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



Cloud computing