Distributed Systems

Example: Amazon's Simple Storage Service

Typical operations

- All operations are carried out by sending HTTP requests:
- Create a bucket/object: POST, along with the URI
- Update a bucket/object: PUT/PATCH, along with the URI
- Listing objects: GET on a bucket name
- Reading an object: GET on a full URI
- Removing an object: Delete on a bucket ID

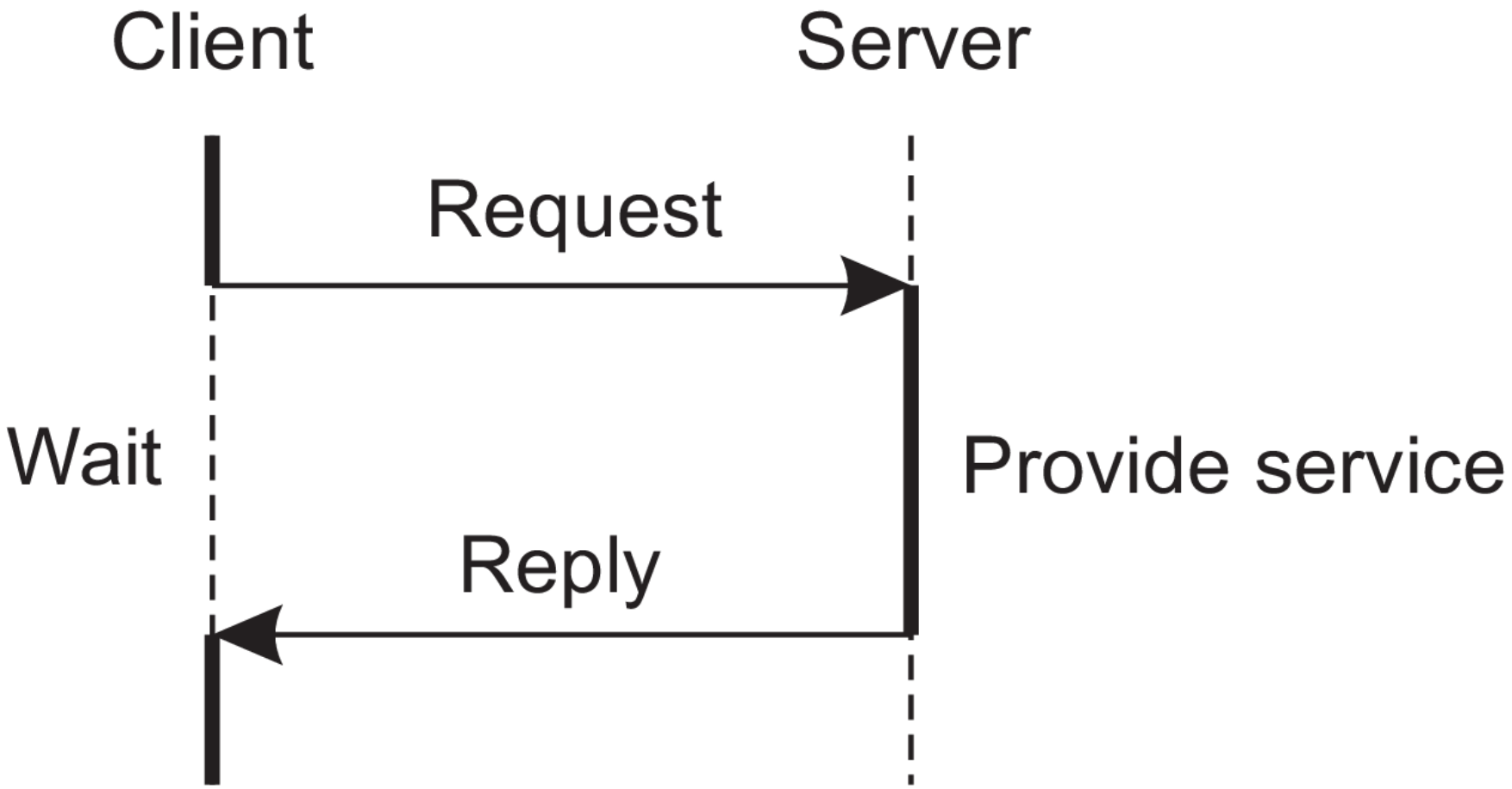
Centralized system architectures

Basic Client–Server Model

Characteristics:

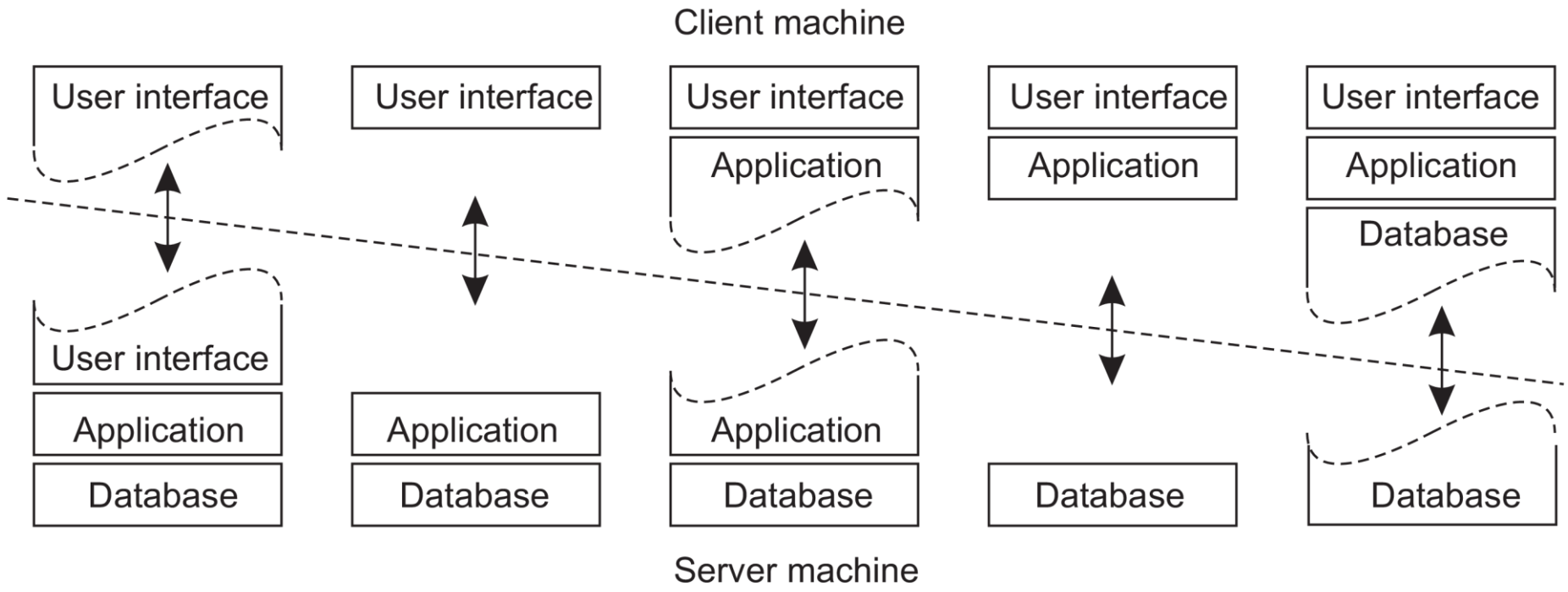
- There are processes offering services (servers)
- There are processes that use services (clients)
- Clients and servers can be on different machines
- Clients follow request/reply model with respect to using services

Centralized system architectures



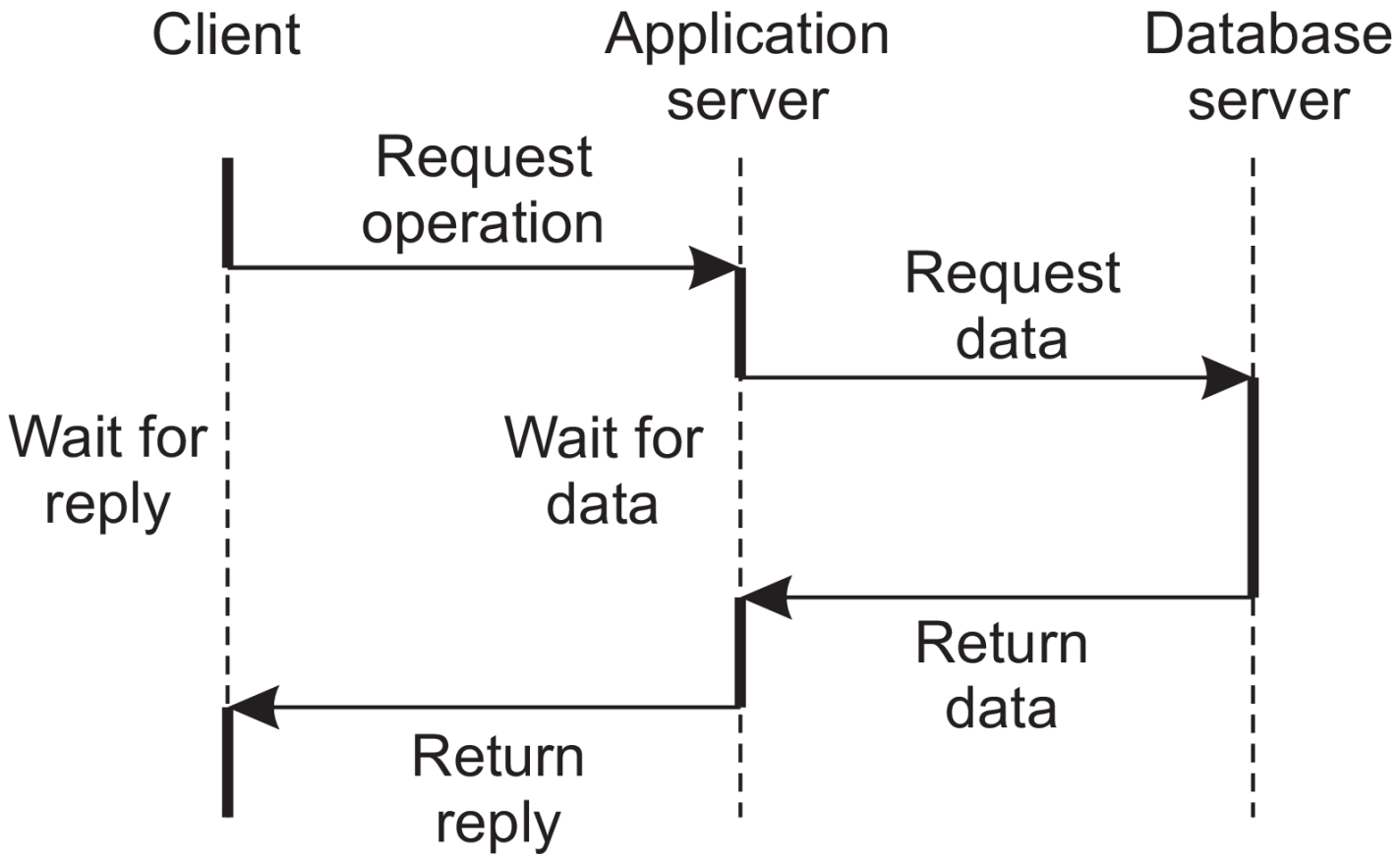
Multi-tiered centralized system architectures

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



Being client and server at the same time

- Three-tiered architecture



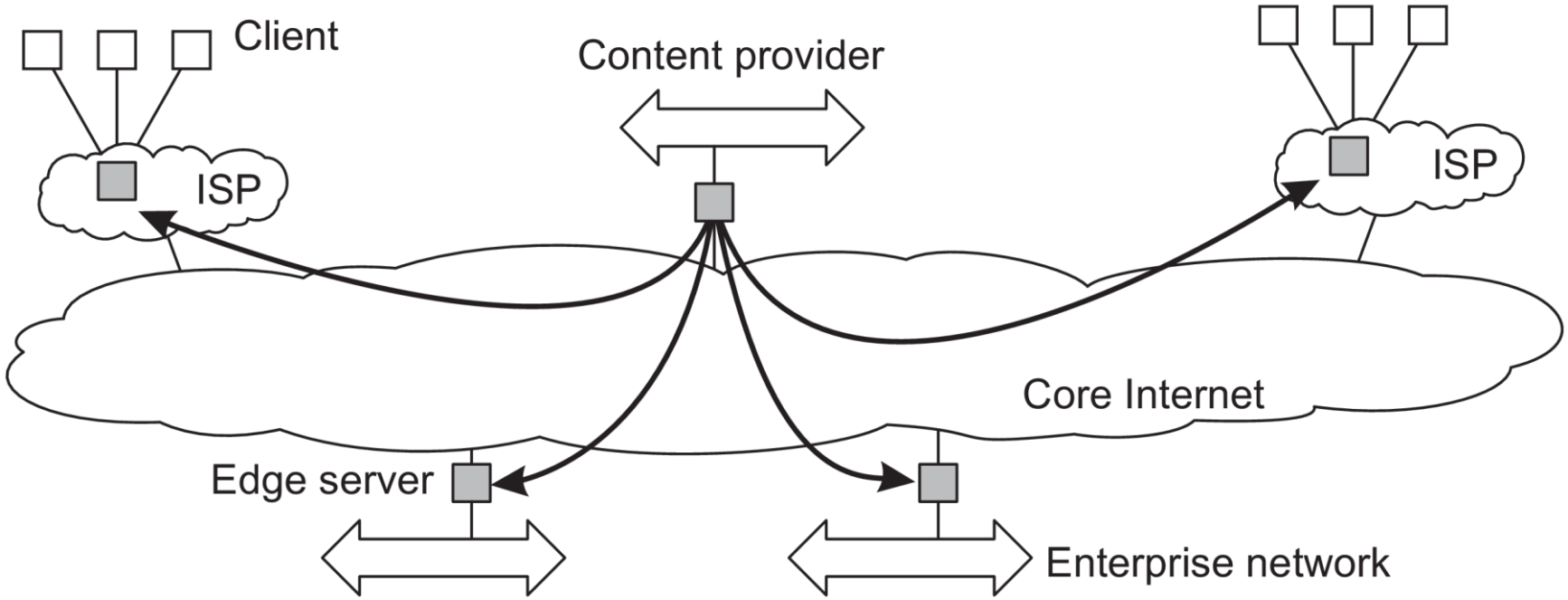
Skype’s principle operation: A wants to contact B

Both A and B are on the public Internet

- A TCP connection is set up between A and B for control packets.
- The actual call takes place using UDP packets between negotiated ports

Edge-server architecture

- Systems deployed on the Internet where servers are placed at the edge of the network:
- The boundary between enterprise networks and the actual Internet.

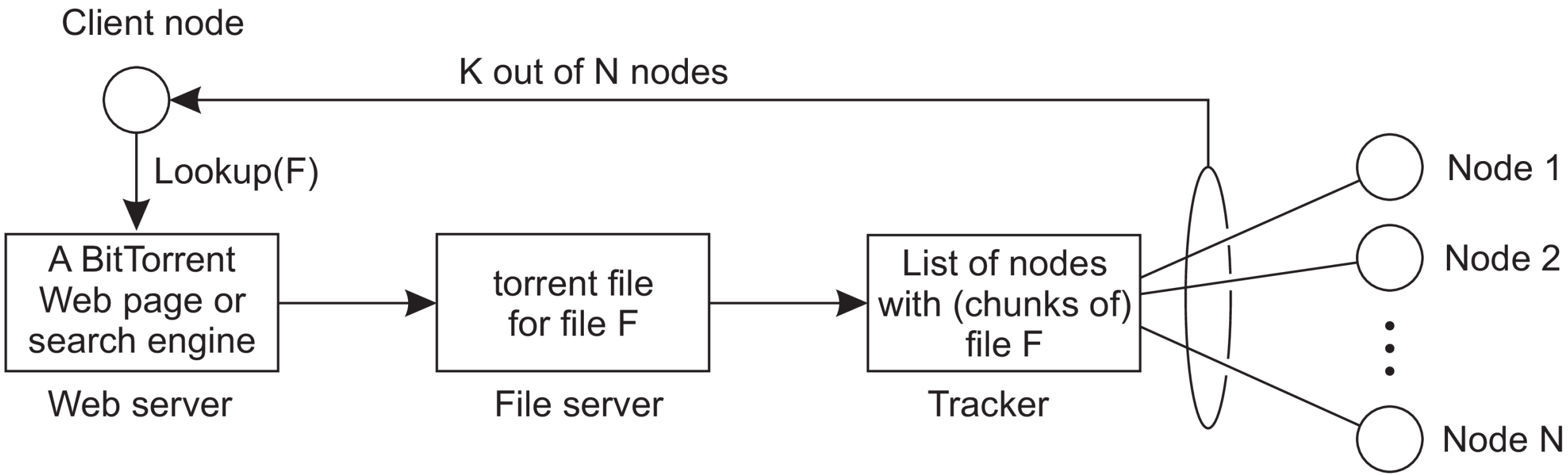


Collaboration: The BitTorrent case

Principle: search for a file F

- Lookup file at a global directory ⇒ 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.

Collaboration: The BitTorrent case



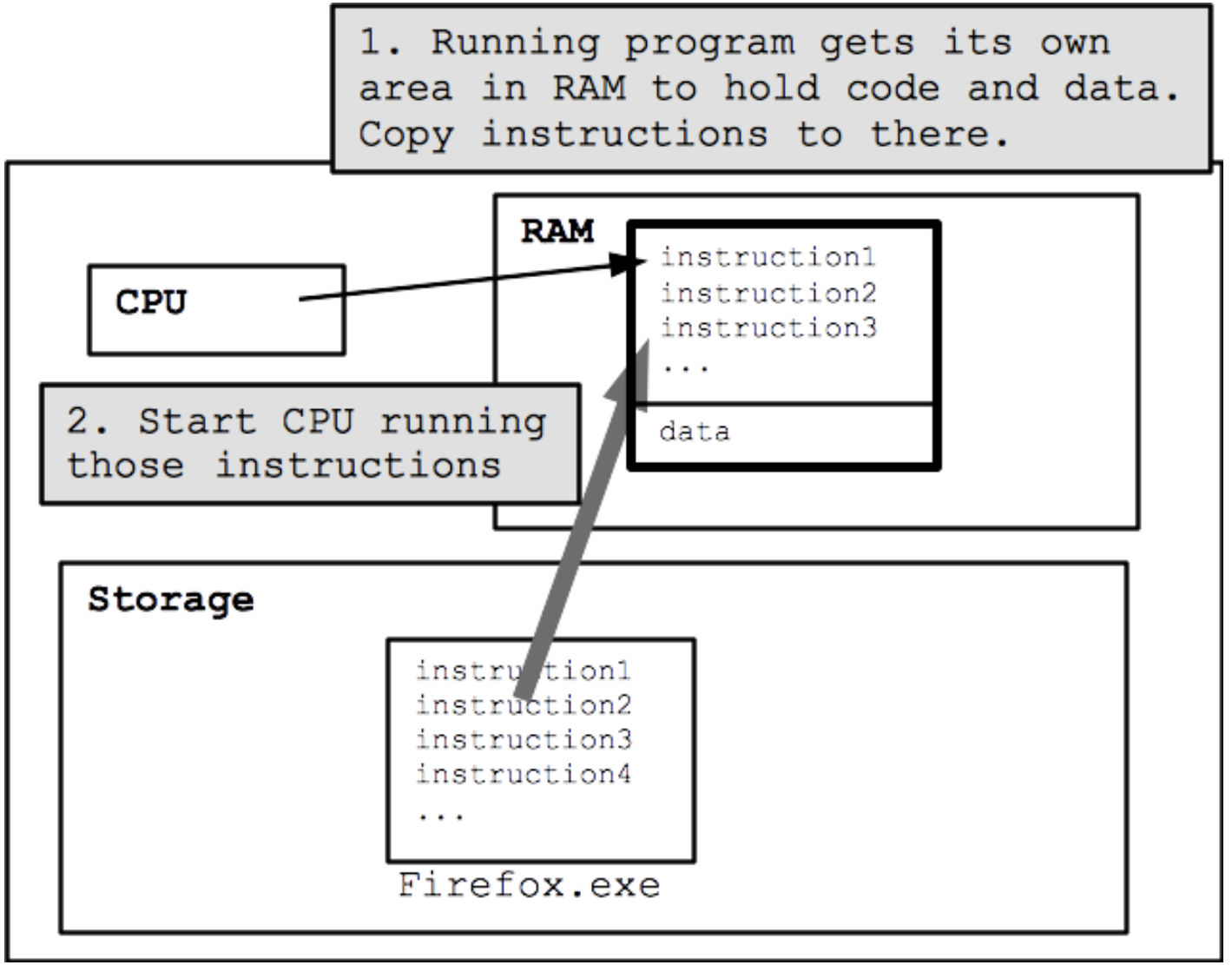
Introduction to threads

We build virtual processors in software, on top of physical processors:

Processor:

- Provides a set of instructions along with the capability of automatically executing a series of those instructions.

Processor

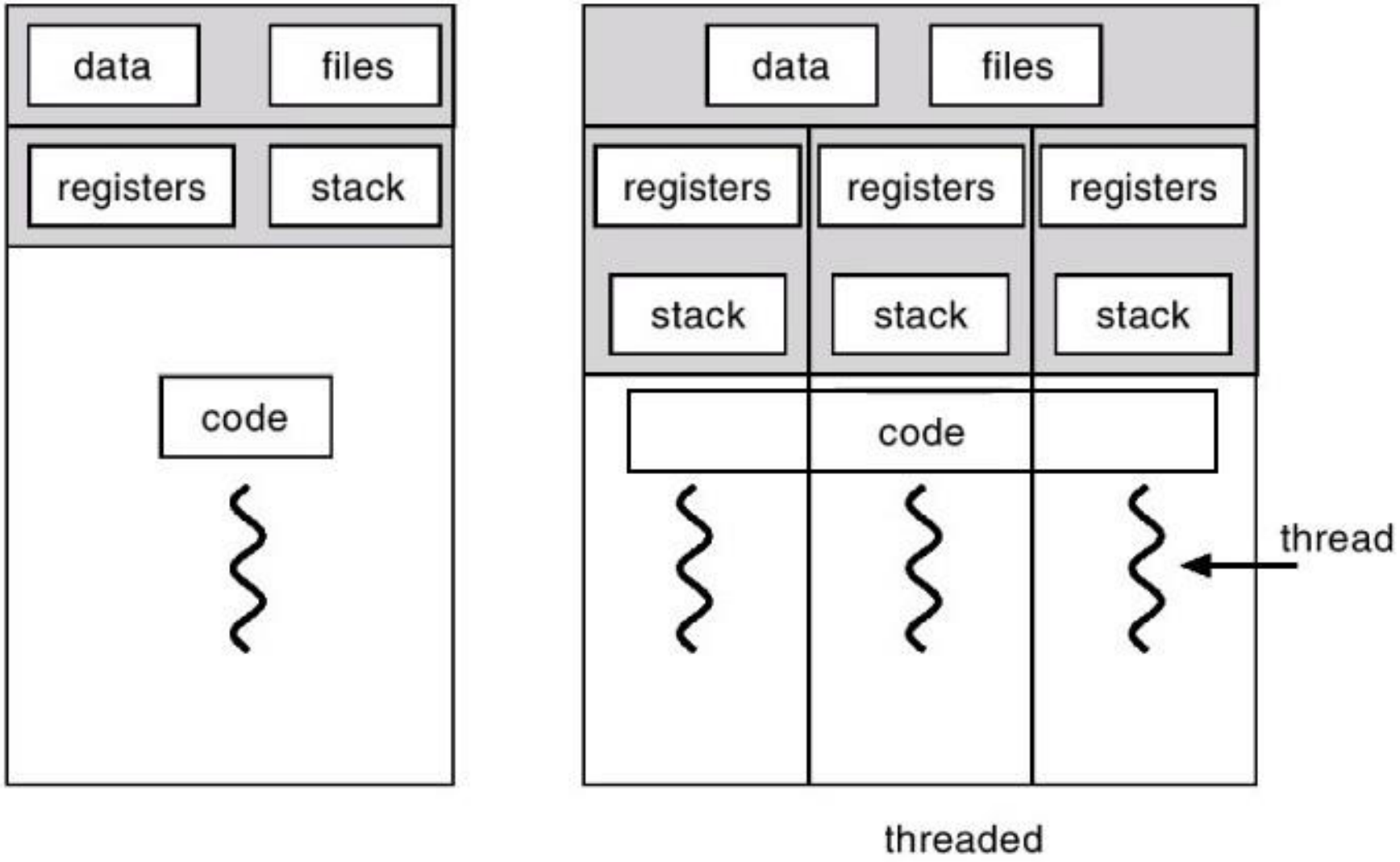

1. Running program gets its own area in RAM to hold code and data. Copy instructions to there.
RAM
instruction1
instruction2
instruction3
...
data
CPU
2. Start CPU running those instructions
Storage
instruction1
instruction2
instruction3
instruction4
...
Firefox.exe

Introduction to threads

Thread:

- A minimal software processor in whose context a series of instructions can be executed. Saving a thread context implies stopping the current execution and saving all the data needed to continue the execution at a later stage.

Thread

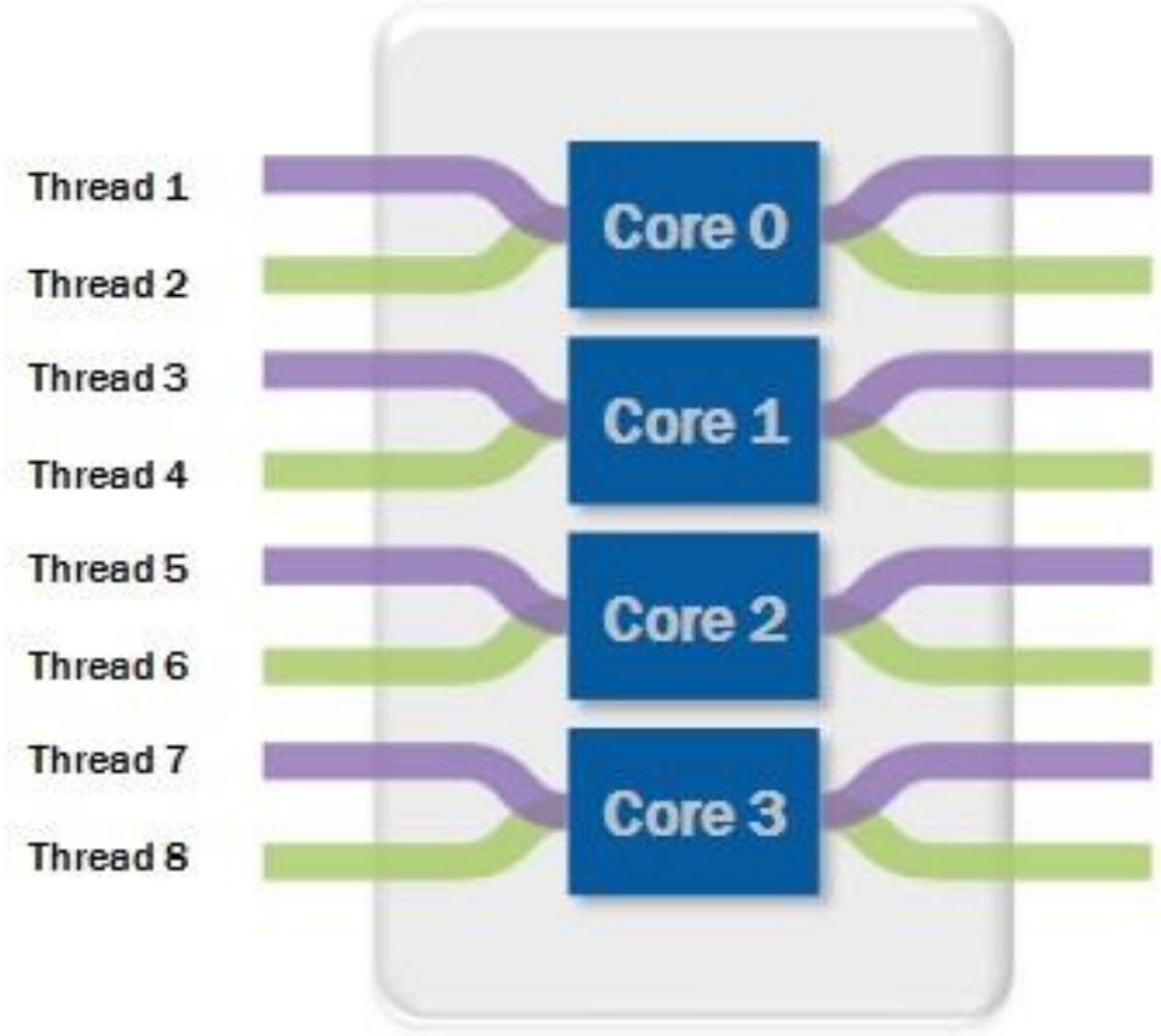

data
files
registers
stack
code
data
files
registers
registers
registers
stack
stack
stack
code
thread
threaded

Thread and Cores

4 MB Cache	6 MB Cache	8 MB Cache	16 MB Cache
2 Cores	4 Cores	4 Cores	8 Cores
4 Thread	8 Thread	8 Thread	16 Thread

What is the difference between a ‘Thread’ and a ‘Core’?

- In hardware, a ‘thread’ usually means a logical core. It may or may not be a physical core
- In software, a ‘thread’ depends on the OS type, but is a single continuous piece of code executing



Introduction to threads

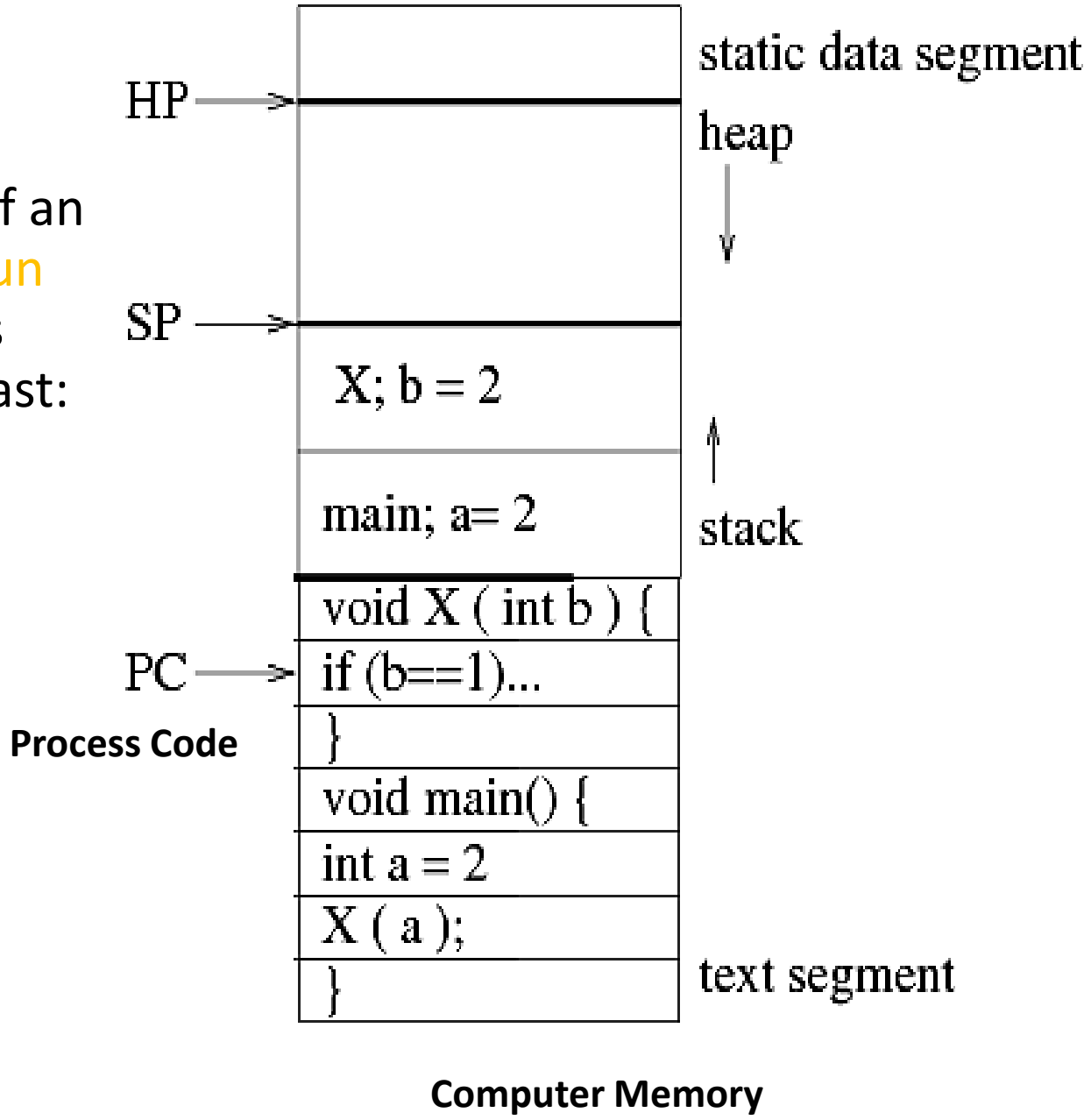
Process:

- A software processor in whose context one or more threads may be executed. Executing a thread, means executing a series of instructions in the context of that thread

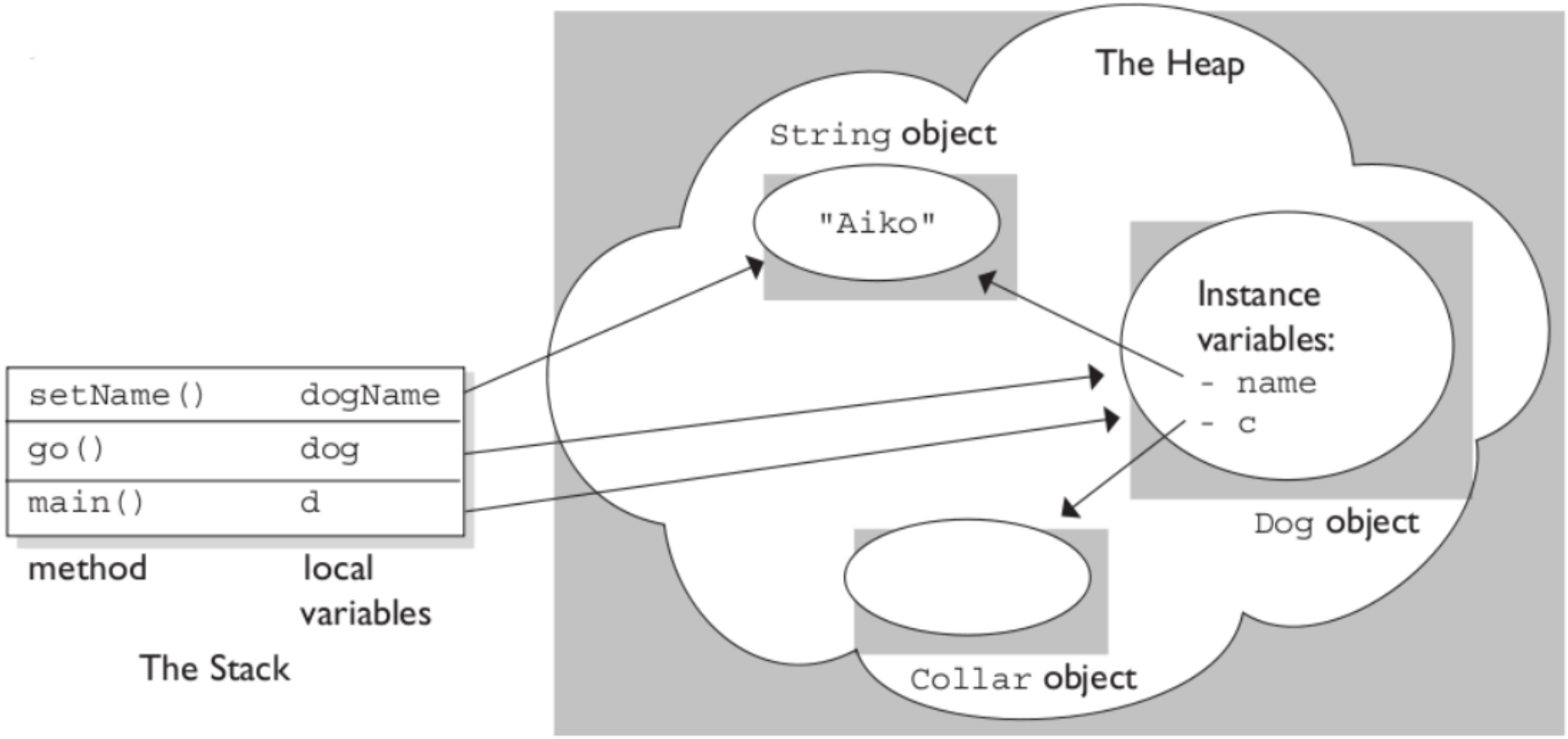
Process consists

A process is the dynamic execution context of an executing program. Several processes may run concurrently, each as a distinct entity with its own state. The process state consists of at least:

- the code for the running program
- the static data for running program
- a space in the heap for dynamic data
- the address of next instruction
- execution stack
- values of the CPU registers
- OS resources (e.g. open files)
- process execution state



Stack Vs Heap



Context switching

- Processor context: The minimal collection of values stored in the registers of a processor used for the execution of a series of instructions (e.g., stack pointer, addressing registers, program counter).
- Thread context: The minimal collection of values stored in registers and memory, used for the execution of a series of instructions (i.e., processor context, state).
- Process context: The minimal collection of values stored in registers and memory, used for the execution of a thread (i.e., thread context).

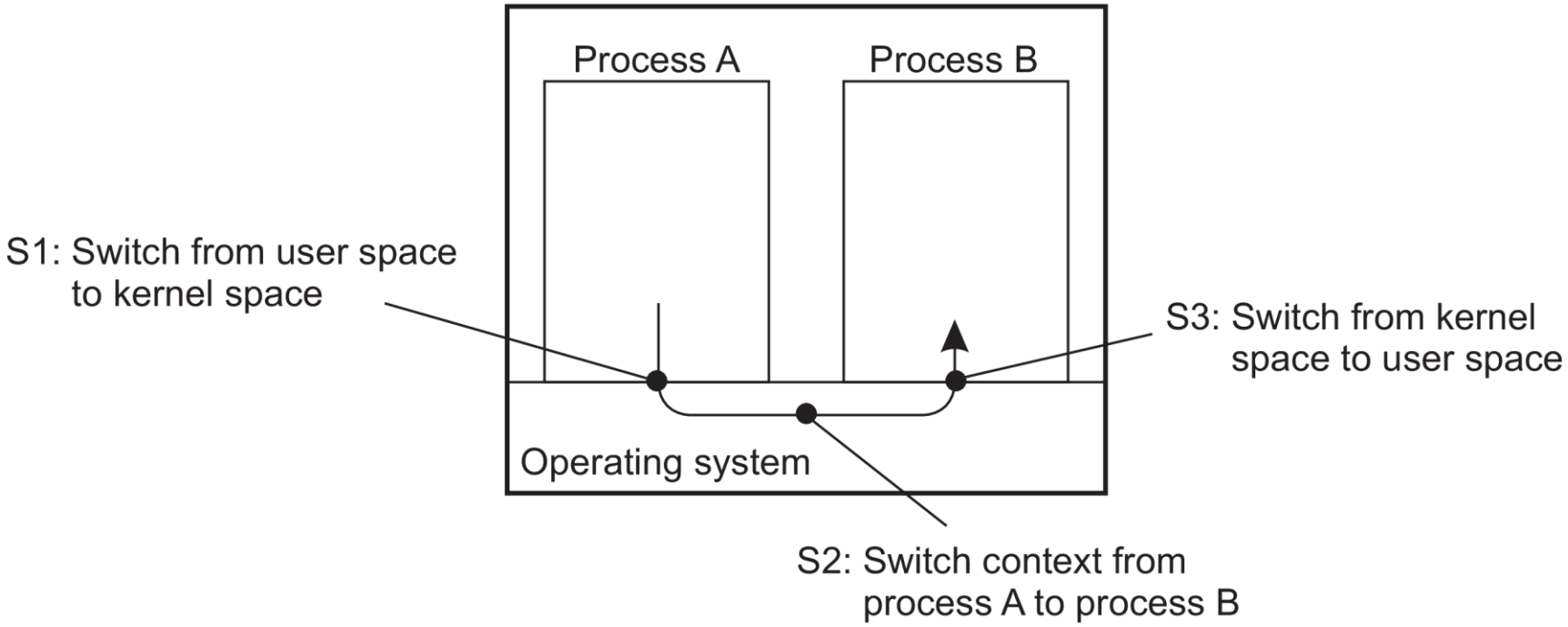
Context switching

- Threads share the same address space. Thread context switching can be done entirely independent of the operating system.
- Process switching is generally (somewhat) more expensive as it involves getting the OS in the loop, i.e., trapping to the kernel.
- Creating and destroying threads is much cheaper than doing so for processes.

Why use threads

- Avoid needless blocking: a single-threaded process will block when doing I/O; in a multi-threaded process, the operating system can switch the CPU to another thread in that process.
- Exploit parallelism: the threads in a multi-threaded process can be scheduled to run in parallel on a multiprocessor or multicore processor.
- Avoid process switching: structure large applications not as a collection of processes, but through multiple threads.

Avoid process switching



Thread context switching may be faster than process context switching

Using threads at the client side

Hiding network latencies:

- Web browser scans an incoming HTML page, and finds that more files need to be fetched.
- Each file is fetched by a separate thread, each doing a (blocking) HTTP request.
- As files come in, the browser displays them

Using threads at the server side

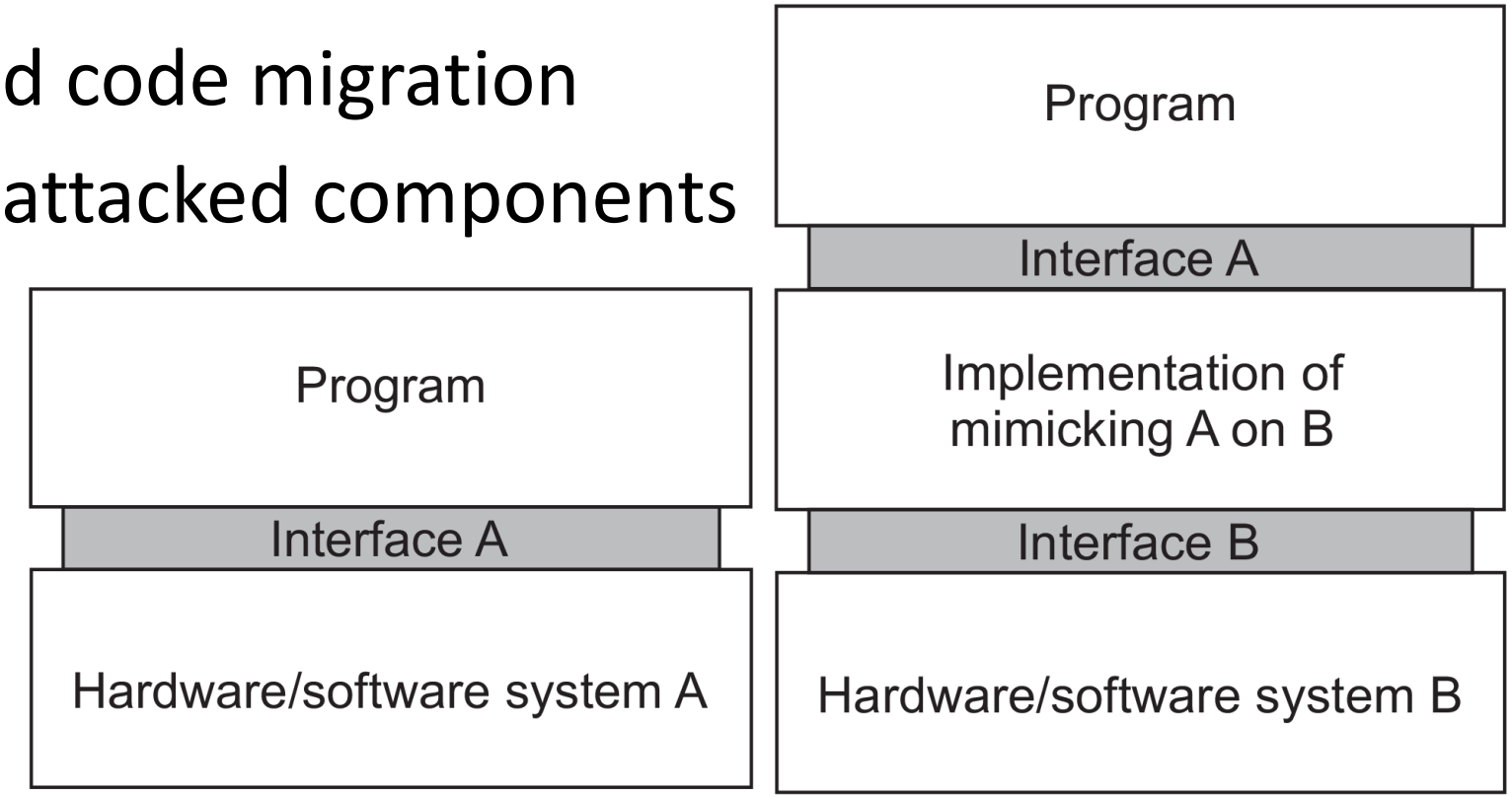
Improve performance

- Starting a thread is cheaper than starting a new process.
- Having a single-threaded server prohibits simple scale-up to a multiprocessor system.
- As with clients: hide network latency by reacting to next request while previous one is being replied.

Virtualization

Virtualization is important:

- Hardware changes faster than software
- Ease of portability and code migration
- Isolation of failing or attacked components



VMs and cloud computing

Three types of cloud services

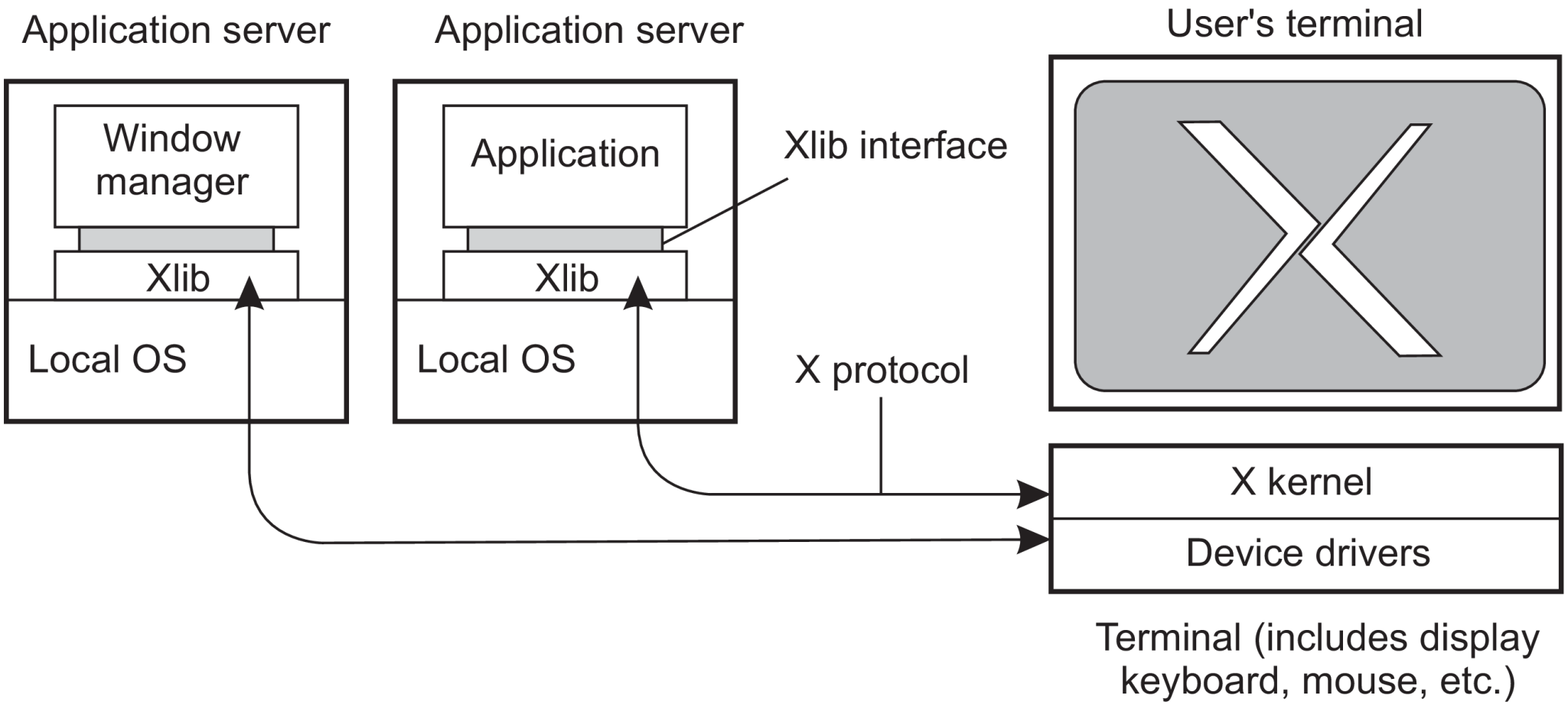
- Infrastructure-as-a-Service covering the basic infrastructure
- Platform-as-a-Service covering system-level services
- Software-as-a-Service containing actual applications

IaaS

Instead of renting out a physical machine, a cloud provider will rent out a VM

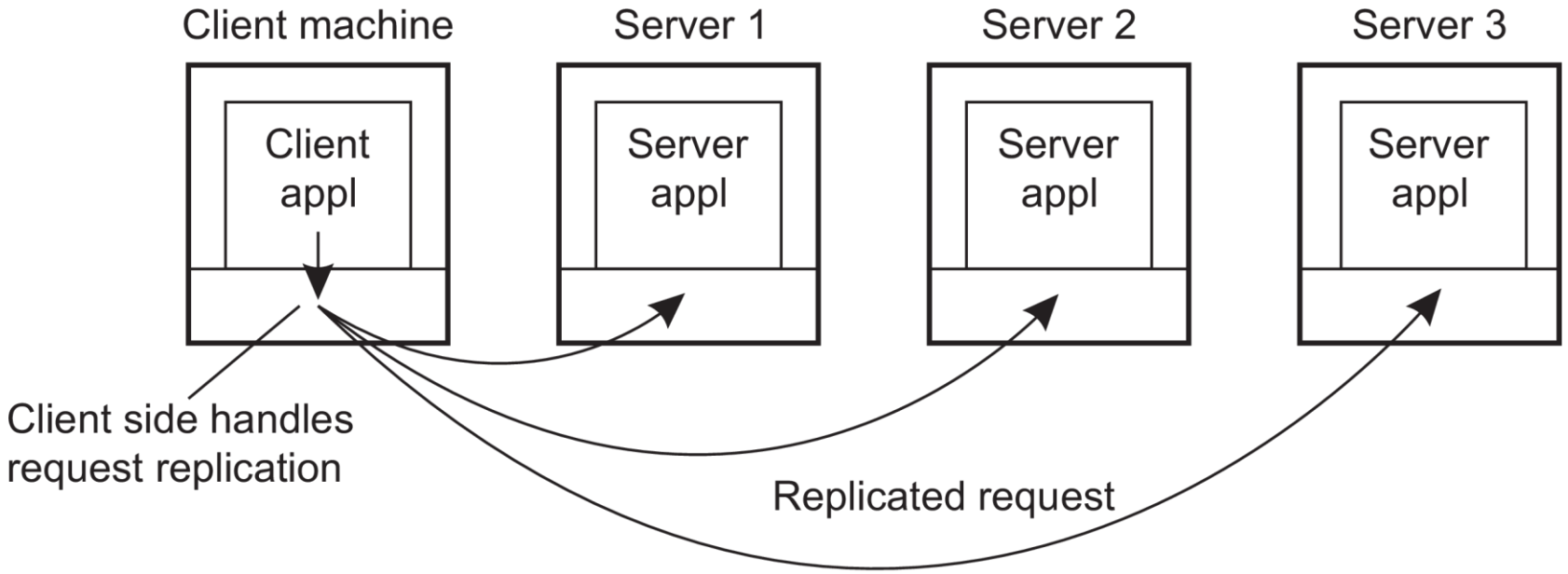
Almost complete isolation between customers

Example X Windows



Client-side software

- Access transparency: client-side stubs for RPCs
- Location/migration transparency: let client-side software keep track of actual location
- Replication transparency: multiple invocations handled by client stub:



Servers

- A process implementing a specific service on behalf of a collection of clients.
- It waits for an incoming request from a client and subsequently ensures that the request is taken care of,
- After which it waits for the next incoming request.

Concurrent servers

- Iterative server:

  Server handles the request before attending a next request.

- Concurrent server:

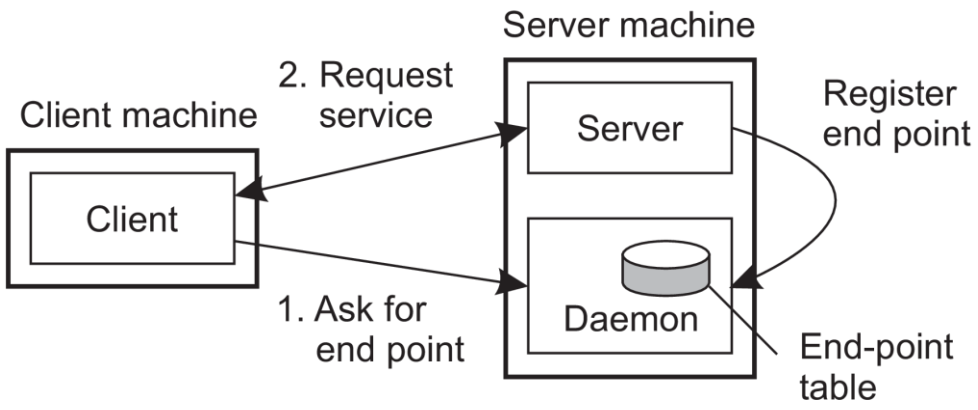
  Uses a dispatcher, which picks up an incoming request that is then passed on to a separate thread/process.

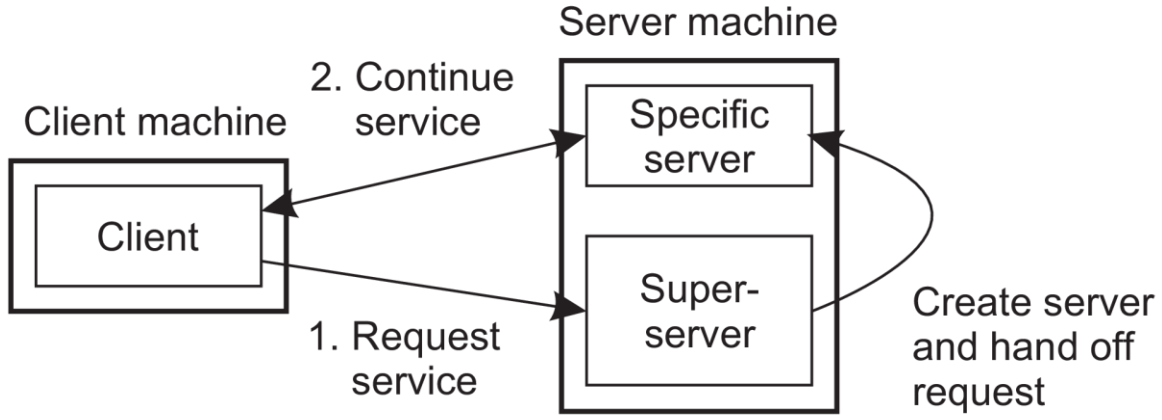
Concurrent servers: they can easily handle multiple requests, notably in the presence of blocking operations

Contacting a server

- Observation: most services are tied to a specific port

ftp-data	20	File Transfer [Default Data]
ftp	21	File Transfer [Control]
telnet	23	Telnet
Smtp	25	Simple Mail Transfer
www	80	Web (HTTP)





Out-of-band communication

Issue

- Is it possible to interrupt a server once it has accepted (or is in the process of accepting) a service request?

Solution 1: Use a separate port for urgent data

- Server has a separate thread/process for urgent messages
- Urgent message comes in ⇒ associated request is put on hold
- Note: we require OS supports priority-based scheduling

Solution 2: Use facilities of the transport layer

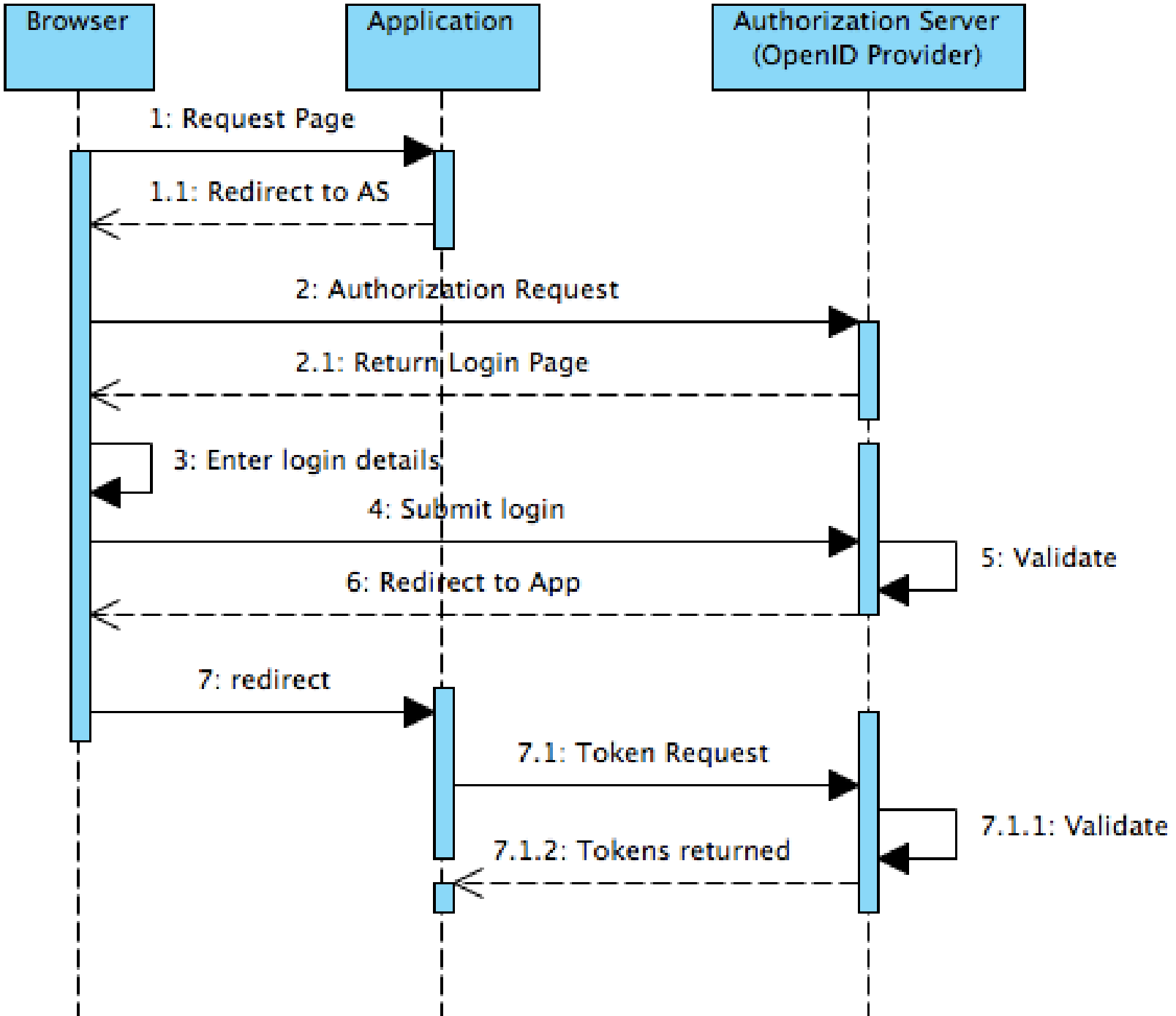
- Example: TCP allows for urgent messages in same connection
- Urgent messages can be caught using OS signaling techniques

Servers and state

Stateless servers

- Never keep accurate information about the status of a client after having handled a request:
- Don’t record whether a file has been opened (simply close it again after access)
- Don’t promise to invalidate a client’s cache
- Don’t keep track of your clients

Client Request Server



Servers and state

Consequences

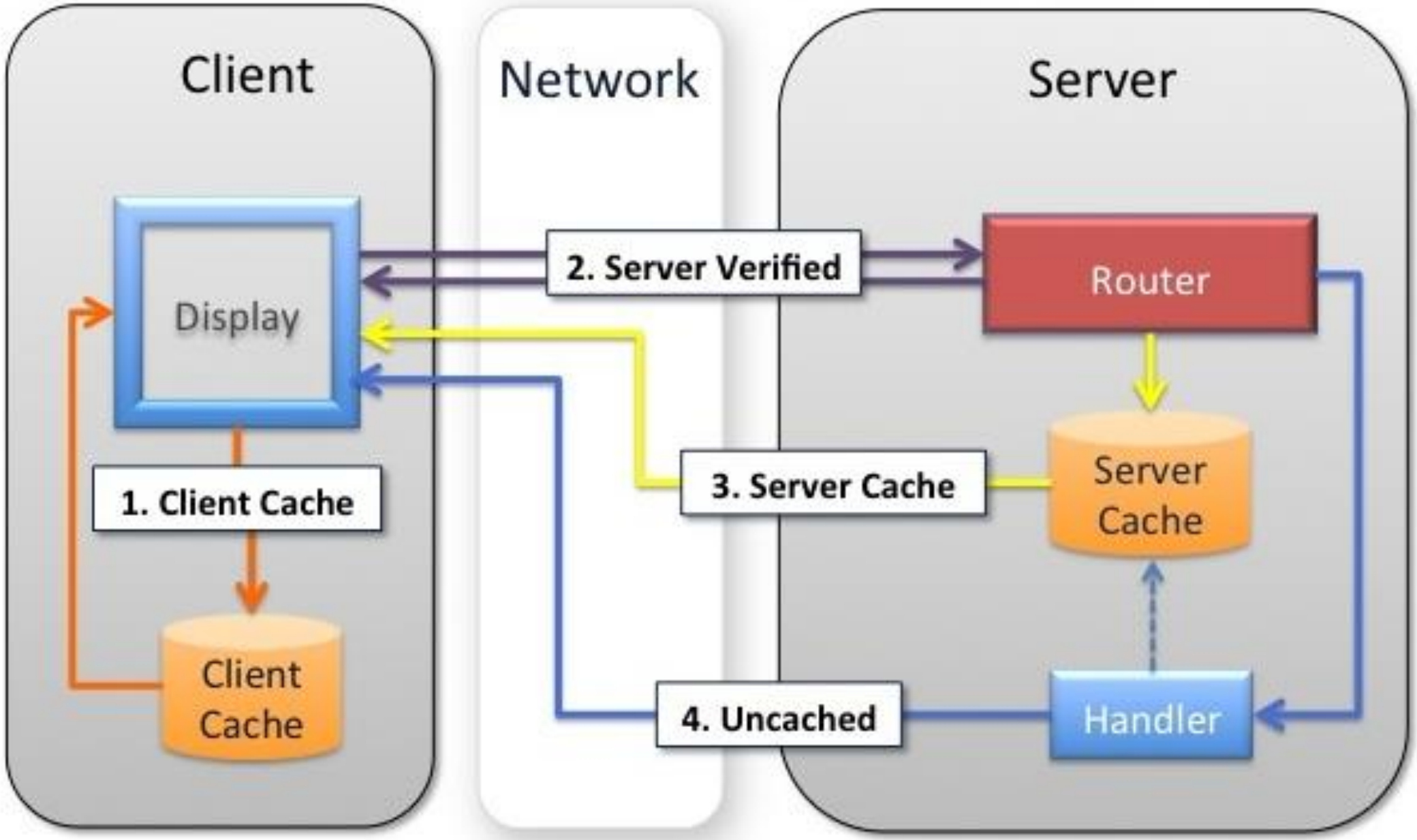
- Clients and servers are completely independent
- State inconsistencies due to client or server crashes are reduced
- Possible loss of performance because, e.g., a server cannot anticipate client behavior (think of prefetching file blocks)

Servers and state

Statefull servers

- Keeps track of the status of its clients:
- Record that a file has been opened, so that prefetching can be done
- Knows which data a client has cached, and allows clients to keep local copies of shared data

Appweb Caching Options

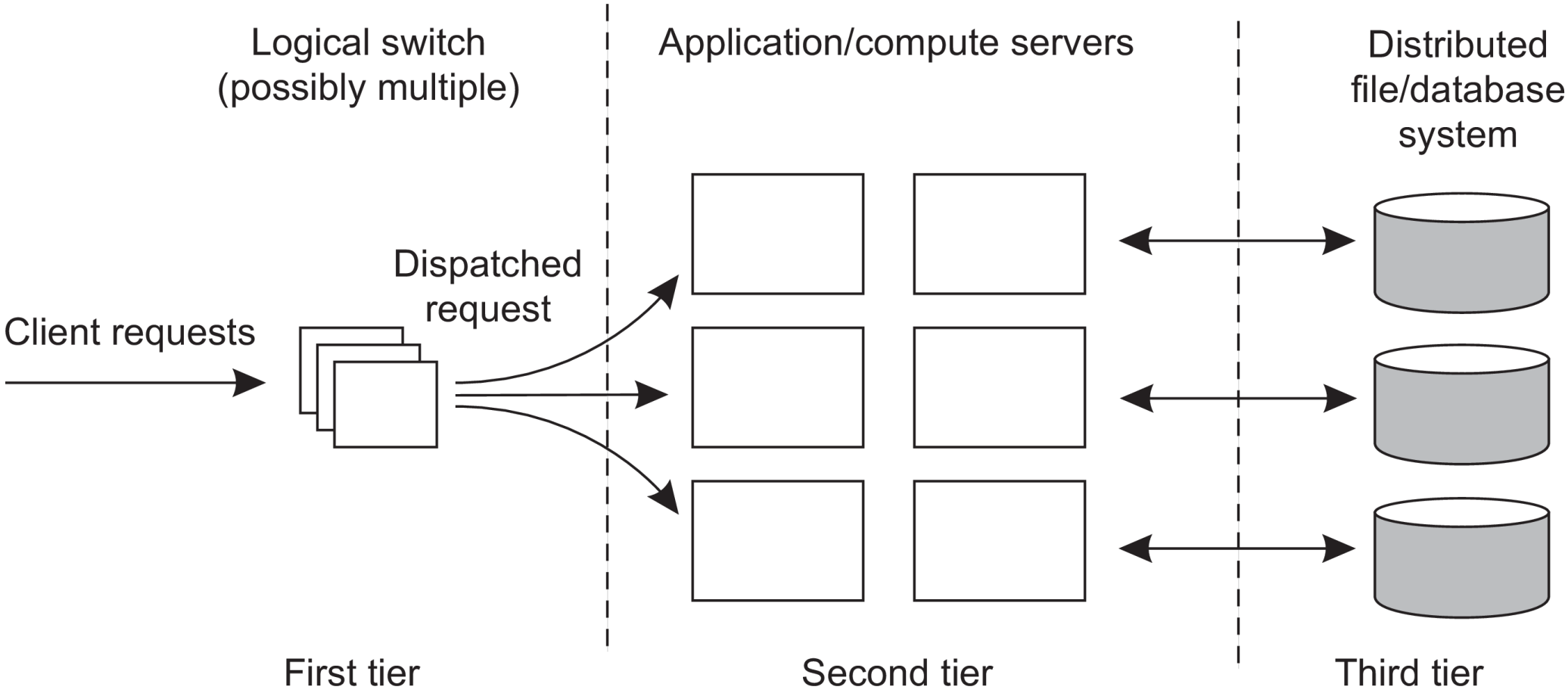


Servers and state

Observation

- The performance of statefull servers can be extremely high,
- Provided clients are allowed to keep local copies.
- As it turns out, reliability is often not a major problem.

Three different tiers



The first tier is generally responsible for passing requests to an appropriate server: request dispatching

Request Handling

- Having the first tier handle all communication from/to the cluster may lead to a bottleneck: A solution: TCP handoff