Distributed Systems Design
COMP 6231

Processes

Lecture 3 – Part 2

Essam Mansour

Today…

- Last Session:
  - Processes
  - Threads
  - Virtualization
- Today's Session:
  - Client
  - Server
  - Code migration

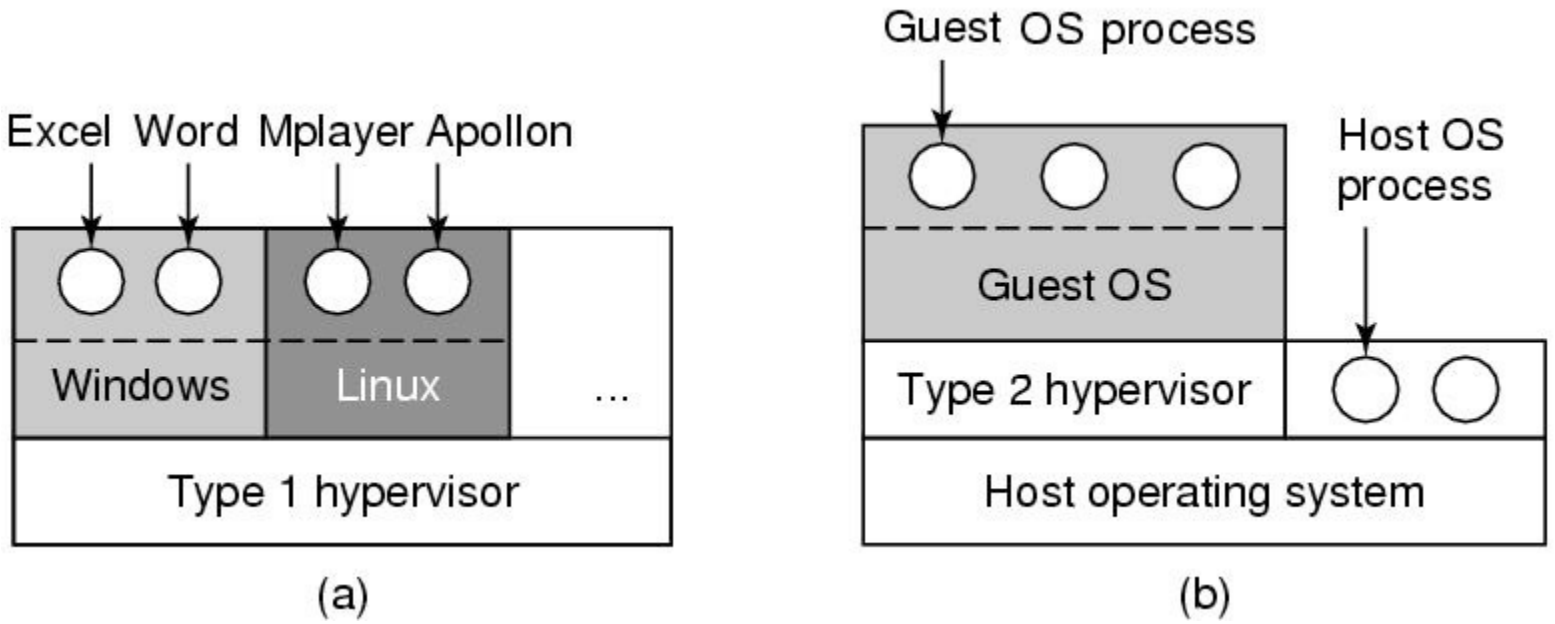
Use of Virtualization Today

- Data centers:
  - server consolidation: pack multiple virtual servers onto a smaller number of physical server
    - saves hardware costs, power and cooling costs
- Cloud computing: rent virtual servers
  - cloud provider controls physical machines and mapping of virtual servers to physical hosts
  - User gets root access on virtual server
- Desktop computing:
  - Multi-platform software development
  - Testing machines
  - Run apps from another platform

Types of virtualization

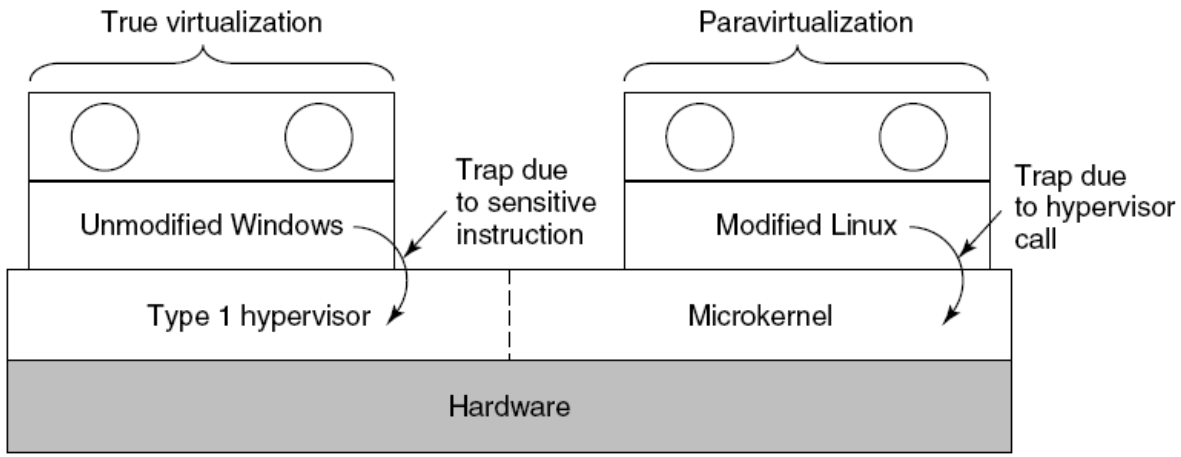
- Para-virtualization
  - VM does not simulate hardware
  - Use special API that a modified guest OS must use
  - Hypercalls trapped by the Hypervisor and serviced
- OS-level virtualization
  - OS allows multiple secure virtual servers to be run
  - Guest OS is the same as the host OS, but appears isolated
    - apps see an isolated OS
  - Solaris Containers, BSD Jails, Linux Vserver, Linux containers, Docker
- Application level virtualization
  - Application is gives its own copy of components that are not shared
    - (E.g., own registry files, global objects) - VE prevents conflicts
  - JVM, Rosetta on Mac (also emulation), WINE

Types of Hypervisors



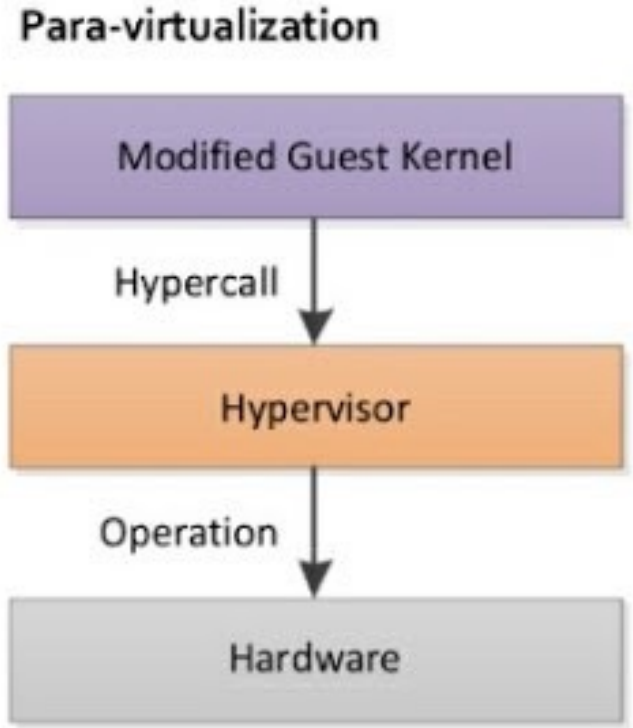
- Type 1: hypervisor runs on “bare metal”
- Type 2: hypervisor runs on a host OS
  - Guest OS runs inside hypervisor
- Both VM types act like real hardware

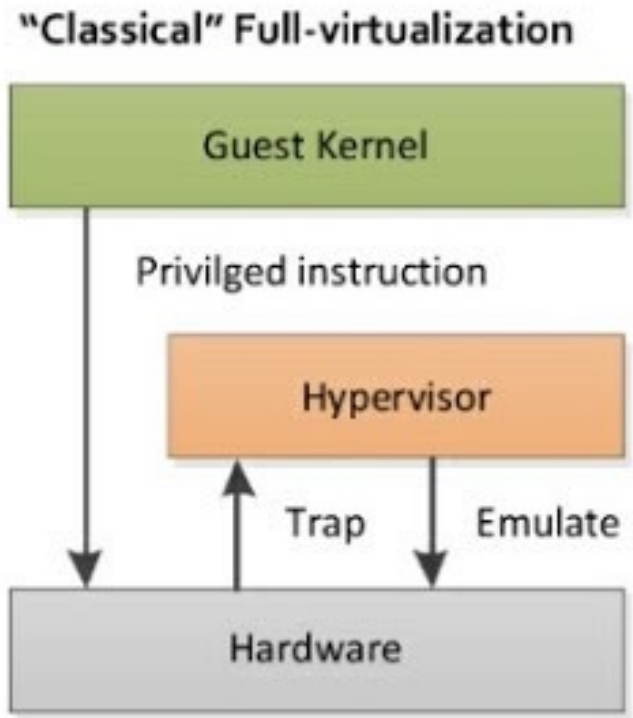
Paravirtualization



- Both type 1 and 2 hypervisors work on unmodified OS
- Paravirtualization: modify OS kernel to replace all sensitive instructions with hypercalls
  - OS behaves like a user program making system calls
  - Hypervisor executes the privileged operation invoked by hypercall.

Paravirtualization





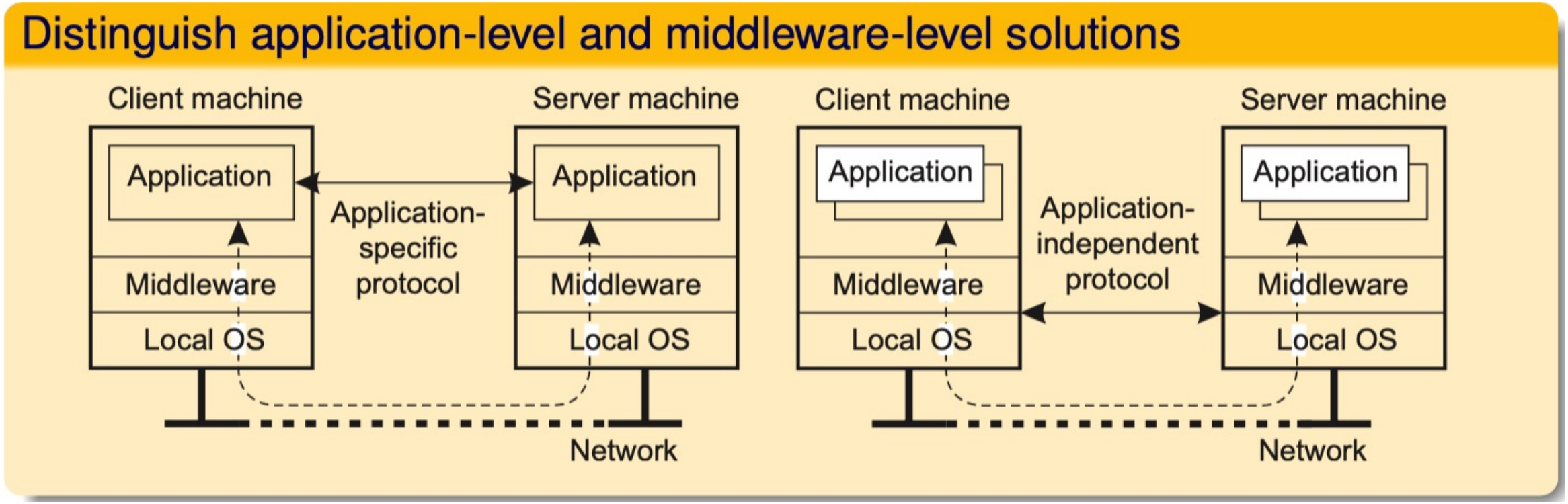
Memory virtualization

- OS manages page tables
  - Create new pagetable is sensitive -> traps to hypervisor
- hypervisor manages multiple OS
  - Need a second shadow page table
  - OS: VM virtual pages to VM's physical pages
  - Hypervisor maps to actual page in shadow page table
  - Two level mapping
  - Need to catch changes to page table (not privileged)
    - Change PT to read-only - page fault
    - Paravirtualized - use hypercalls to inform

I/O Virtualization

- Each guest OS thinks it “owns” the disk
- Hypervisor creates “virtual disks”
  - Large empty files on the physical disk that appear as “disks” to the guest OS
    - Hypervisor converts block # to file offset for I/O
  - DMA need physical addresses
    - Hypervisor needs to translate

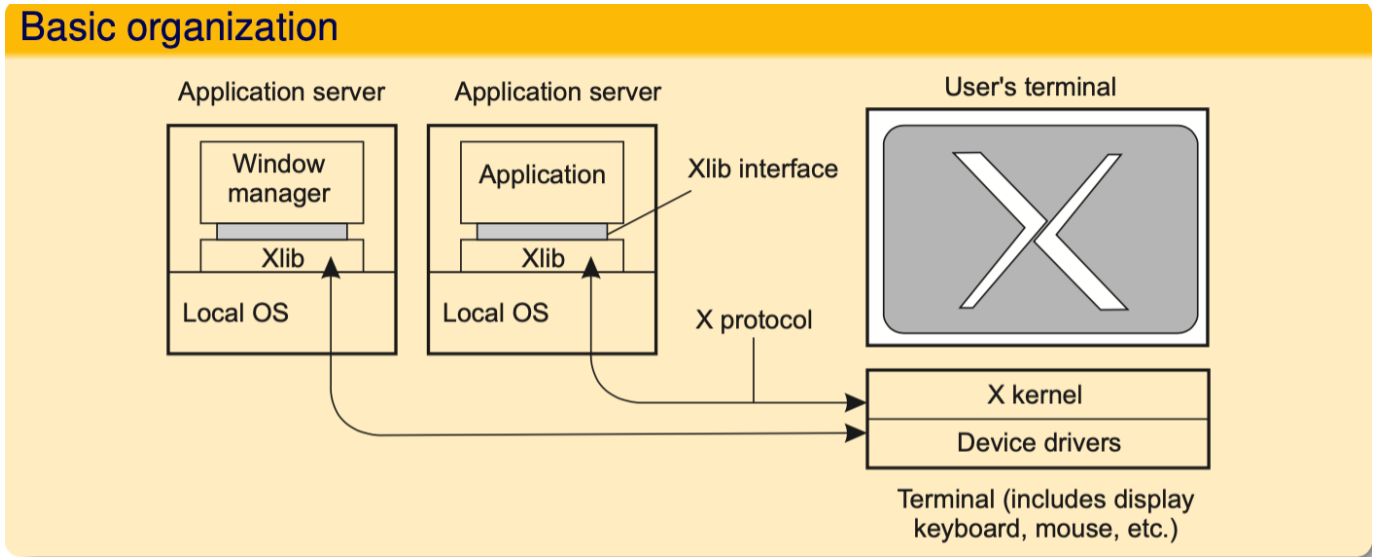
Client-server interaction

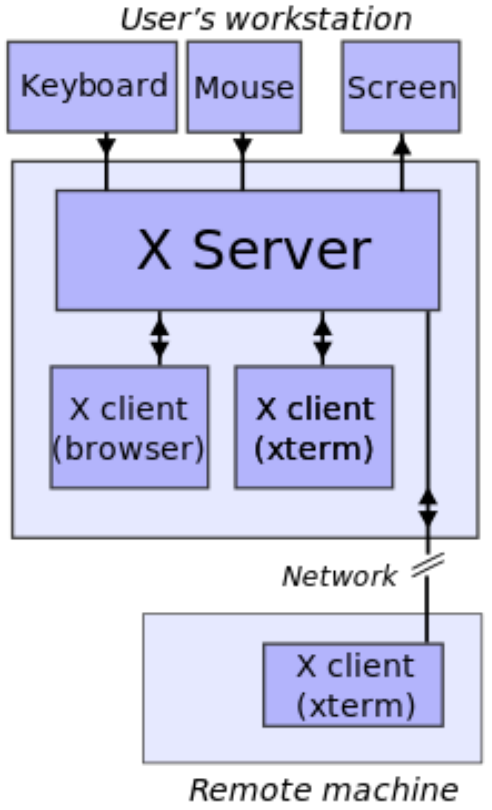


- A calendar app in a smart phone synchronizes with the remote shared calendar
- It needs its own protocol

- The client is used as a terminal with no need to a local storage
- A general solution to allow access to remote applications

Example: The XWindow System

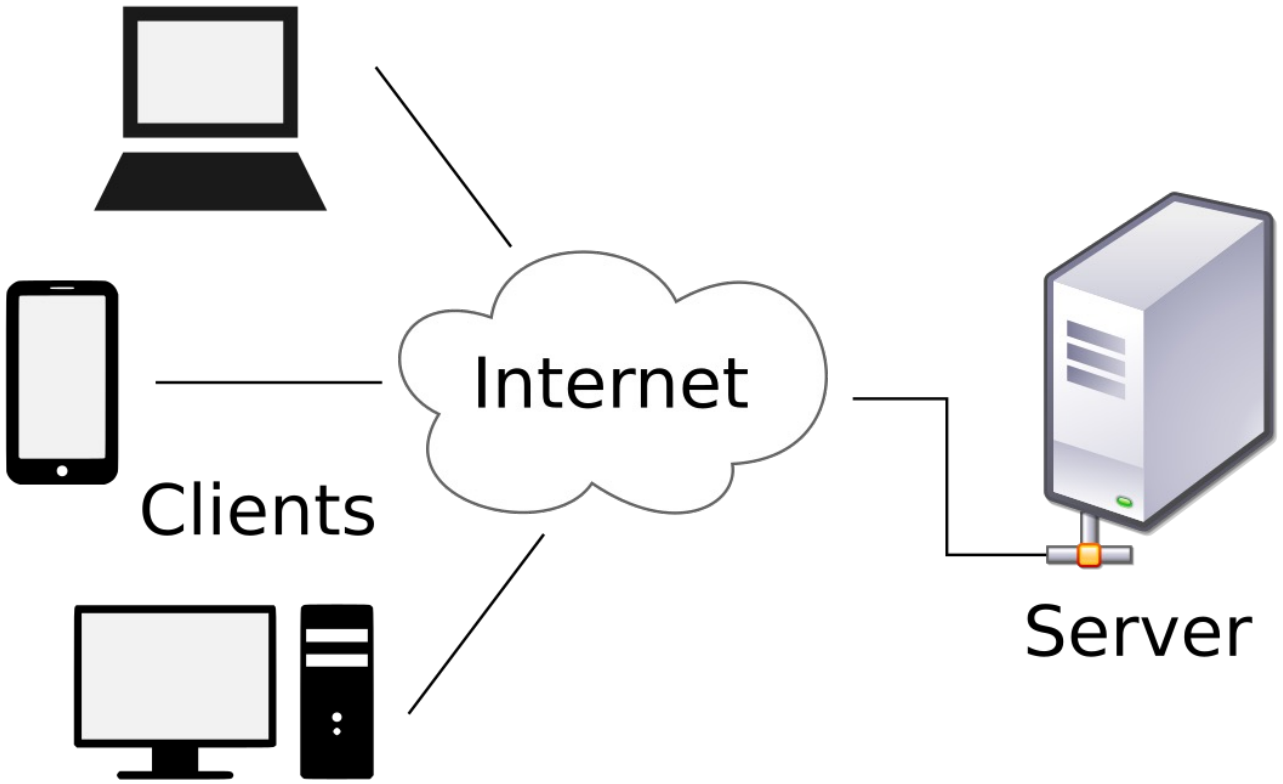




X Client and Server

The application acts as a client to the X-kernel, the latter running as a server on the client's machine.

Servers: General organization



- 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.

Servers: Two basic types

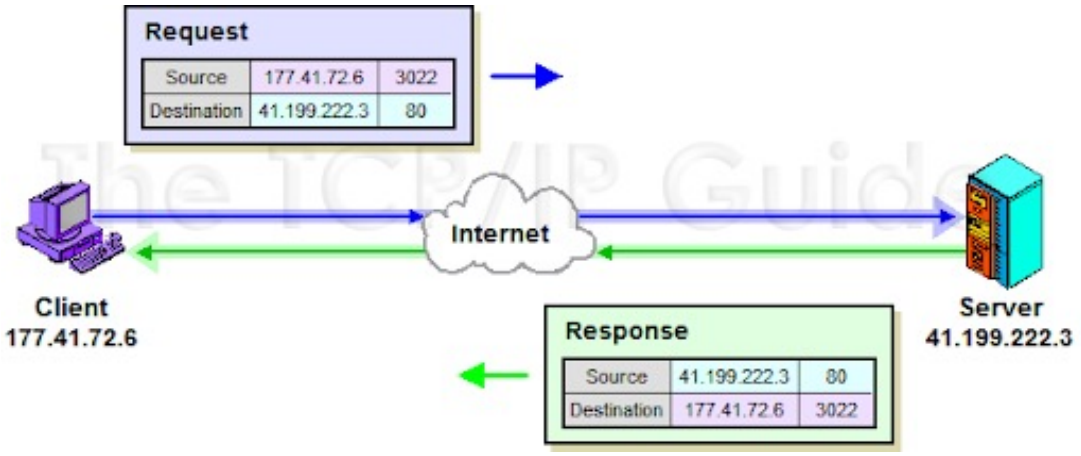
- Iterative Server
  - Process one request at a time
  - Easy to build
  - Unnecessary delay
- Concurrent server
  - Handles multiple request at one time
  - More effort is required to design and build
  - Better performance

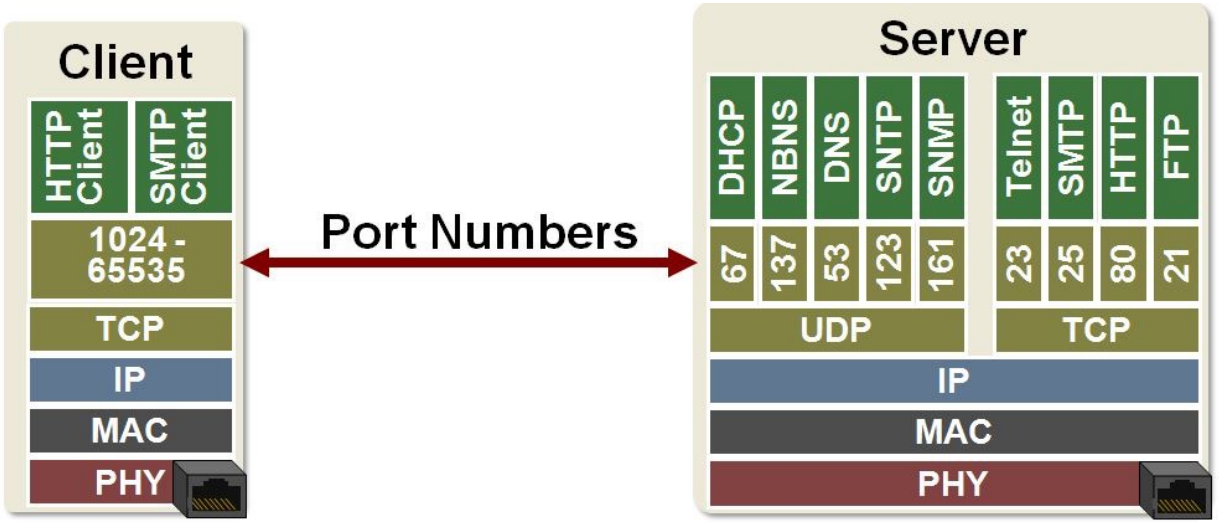
Observation

Concurrent servers are the norm: they can easily handle multiple requests, notably in the presence of blocking operations (to disks or other servers).

Contacting a server

- IP address + Port number = Socket





An **Internet Protocol address** (**IP address**) is a numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication.

A **media access control address** (**MAC address**) is a unique identifier assigned to a network interface controller (NIC) for use as a network address in communications within a network segment.

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

Stateful or Stateless?

- Stateful server
  - Maintain state of connected clients
  - Sessions in web servers
- Stateless server
  - No state for clients
- Soft state
  - Maintain state for a limited time; discarding state does not impact correctness

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

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)

Question

Does connection-oriented communication fit into a stateless design?

Servers and state

Stateful servers

Keeps track of the status of its clients:

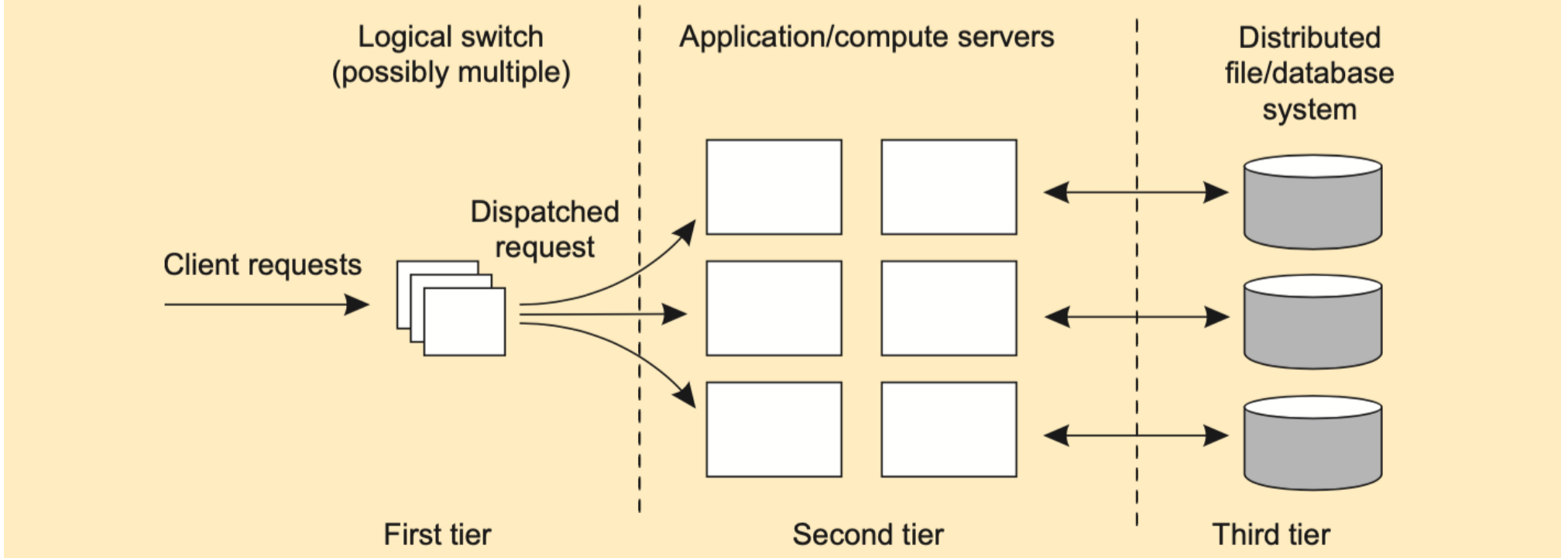
- Record that a file has been opened, so that prefetching can be done
- Knows which data a client has cached, and allows clients to keep local copies of shared data

Observation

The performance of stateful 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

Common organization



Crucial element

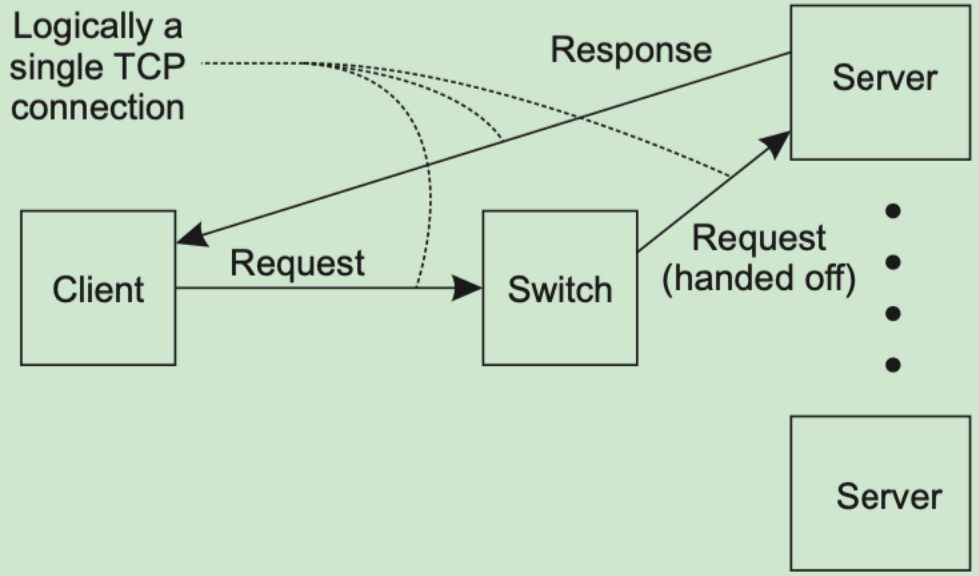
The first tier is generally responsible for passing requests to an appropriate server: request dispatching

Request Handling

Observation

Having the first tier handle all communication from/to the cluster may lead to a bottleneck.

A solution: TCP handoff

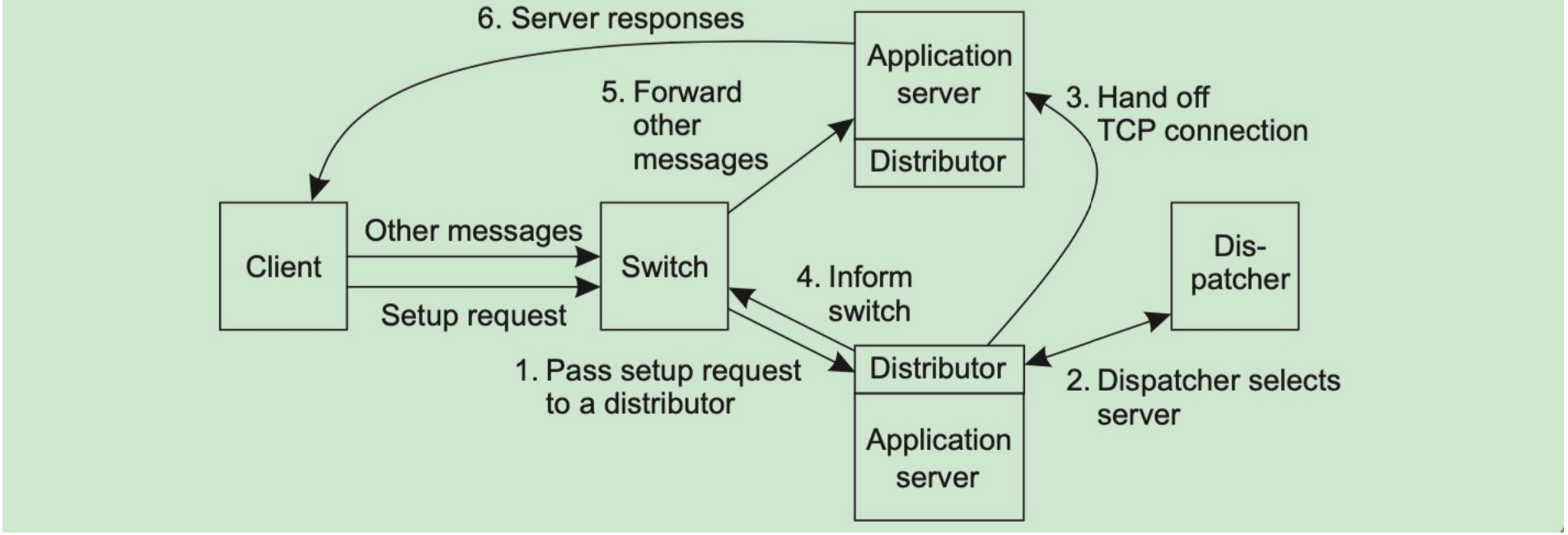


Server clusters

The front end may easily get overloaded: special measures may be needed

- Transport-layer switching: Front end simply passes the TCP request to one of the servers, taking some performance metric into account.
- Content-aware distribution: Front end reads the content of the request and then selects the best server.

Combining two solutions



When servers are spread across the Internet

Observation

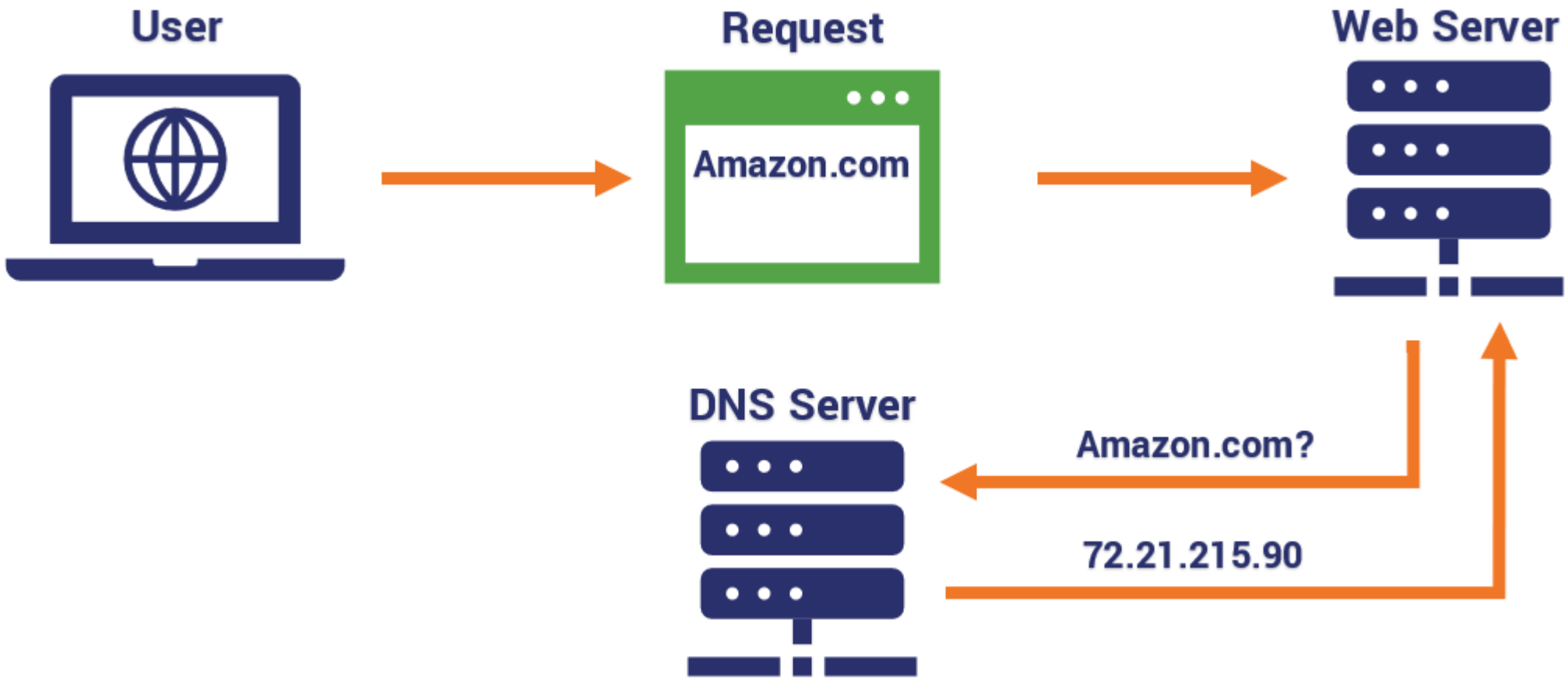
Spreading servers across the Internet may introduce administrative problems. These can be largely circumvented by using data centers from a single cloud provider.

Request dispatching: if locality is important

Common approach: use Domain Name System (DNS):

- Client looks up specific service through DNS - client's IP address is part of request
- DNS server keeps track of replica servers for the requested service, and returns address of most local server.

When servers are spread across the Internet



When servers are spread across the Internet

Observation

Spreading servers across the Internet may introduce administrative problems. These can be largely circumvented by using data centers from a single cloud provider.

Request dispatching: if locality is important

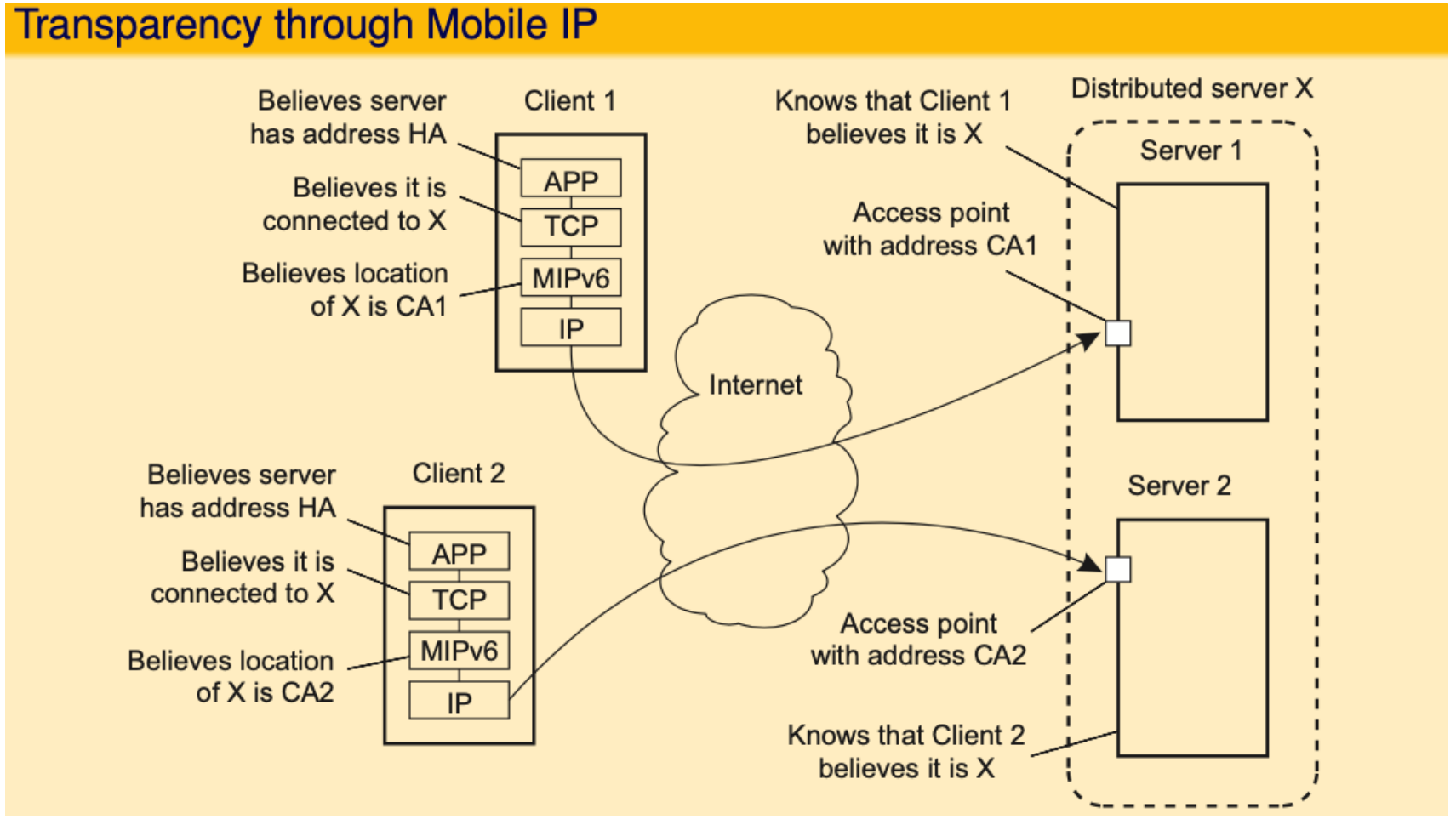
Common approach: use Domain Name System (DNS):

- Client looks up specific service through DNS - client's IP address is part of request
- DNS server keeps track of replica servers for the requested service, and returns address of most local server.

Client transparency

To keep client unaware of distribution, let DNS resolver act on behalf of client. Problem is that the resolver may actually be far from local to the actual client.

Distributed servers with stable IPv6 address(es)



Route optimization can be used to make different clients believe they are communicating with a single server, where, in fact, each client is communicating with a different member node of the distributed server, as shown

Case Study: PlanetLab

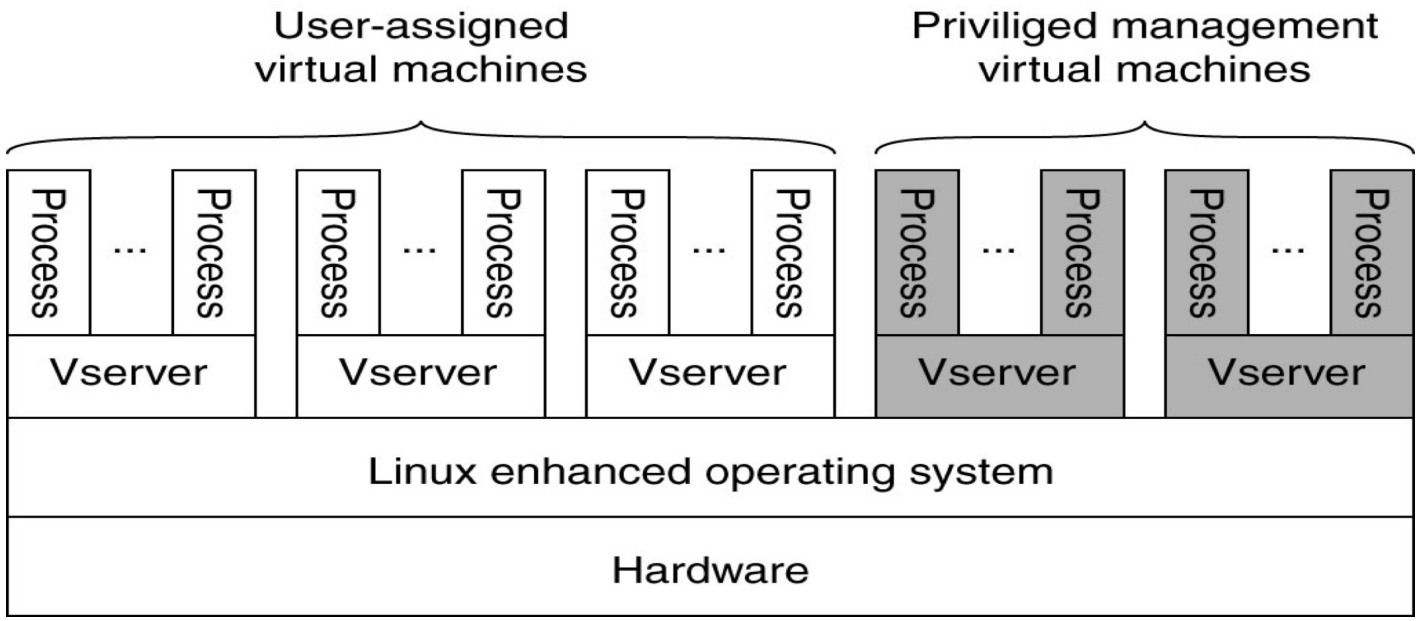
Essence

Different organizations contribute machines, which they subsequently share for various experiments.

Problem

We need to ensure that different distributed applications do not get into each other's way ⇒ virtualization

PlanetLab basic organization



Overview

- Distributed cluster across universities
  - Used for experimental research by students and faculty in networking and distributed systems
- Uses a virtualized architecture
  - Linux Vservers
  - Node manager per machine
  - Obtain a “slice” for an experiment: slice creation service

Case Study: PlanetLab

- PlanetLab management issues:
- Nodes belong to different organizations.
  - Each organization should be allowed to specify who is allowed to run applications on their nodes,
  - And restrict resource usage appropriately.
- Monitoring tools available assume a very specific combination of hardware and software.
  - All tailored to be used within a single organization.
- Programs from different slices but running on the same node should not interfere with each other.

Case Study: PlanetLab

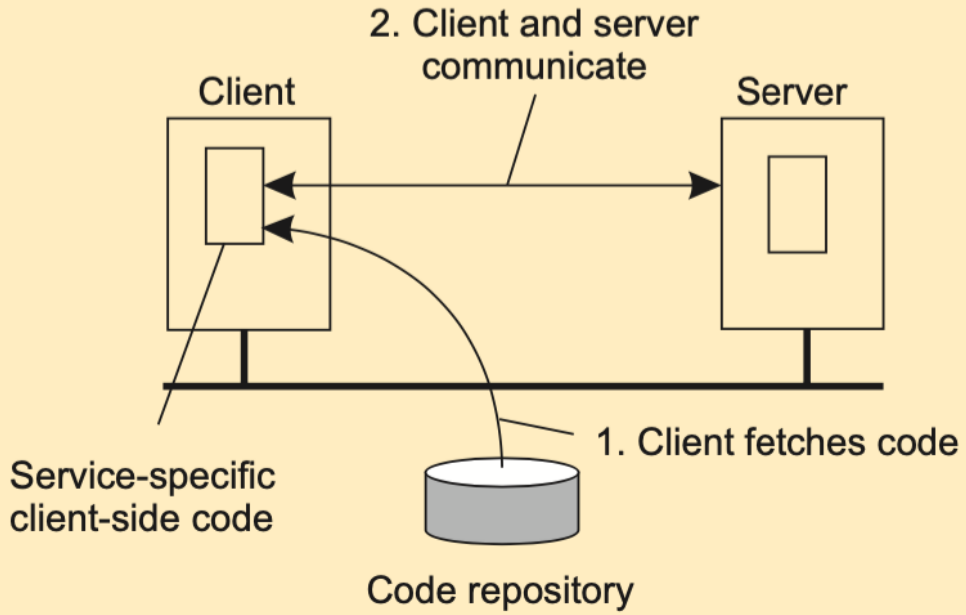
- Relationships between PlanetLab entities:
- A service provider contacts a slice authority to create a slice on a collection of nodes.
- The slice authority needs to authenticate the service provider.
- A node owner provides a slice creation service for a slice authority to create slices. It essentially delegates resource management to the slice authority.
- A management authority delegates the creation of slices to a slice authority.

Reasons for Migrating Code

Load distribution

- Ensuring that servers in a data center are sufficiently loaded (e.g., to prevent waste of energy)
- Minimizing communication by ensuring that computations are close to where the data is (think of mobile computing).

Flexibility: moving code to a client when needed



Strong and weak mobility

Object Component

- Code segment: contains the actual code
- Data segment: contains the state
- Execution state: contains context of thread executing the object's code

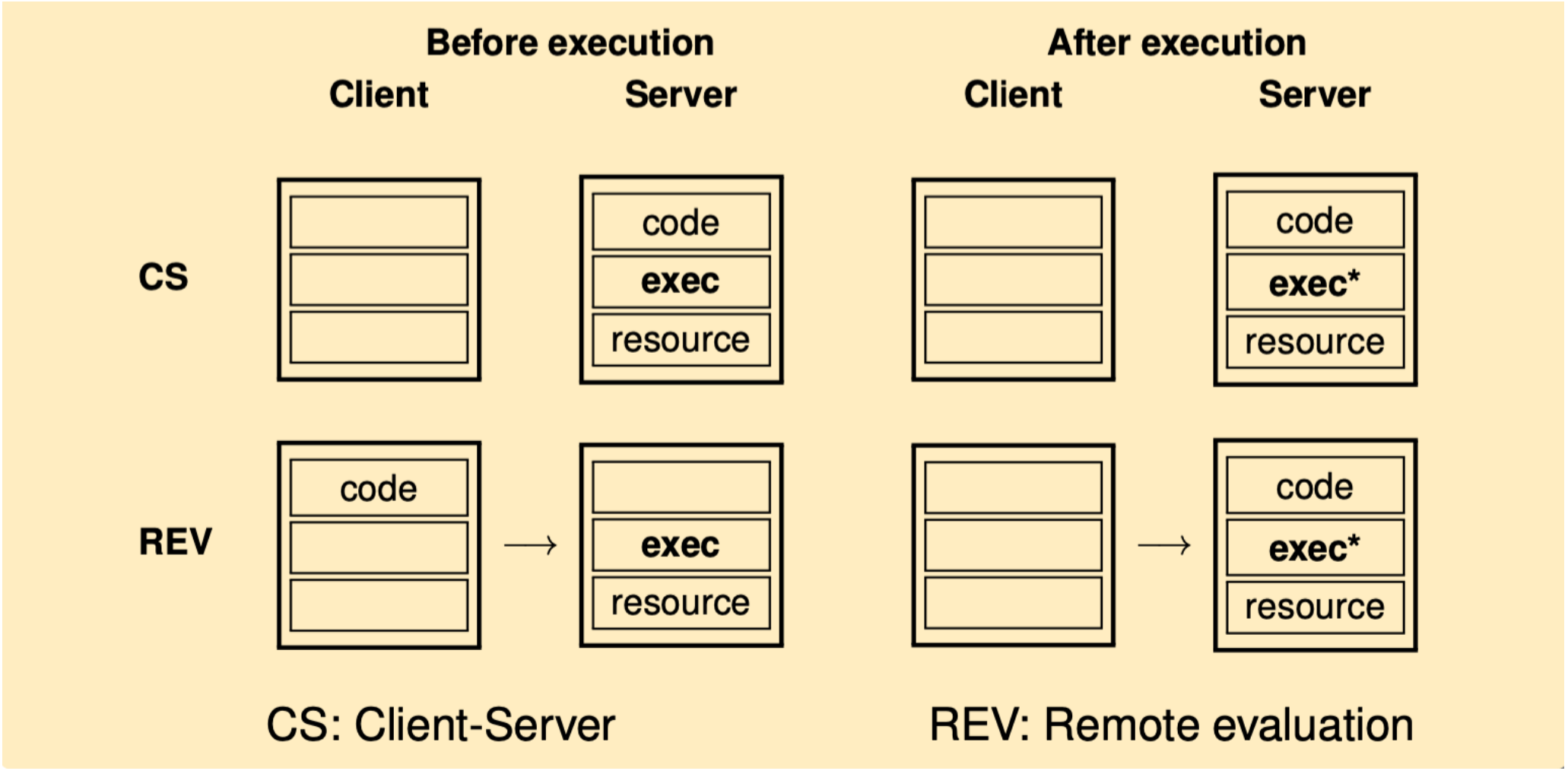
Weak mobility: Move only code and data segment (and reboot execution)

- Relatively simple, especially if code is portable
- Distinguish code shipping (push) from code fetching (pull)

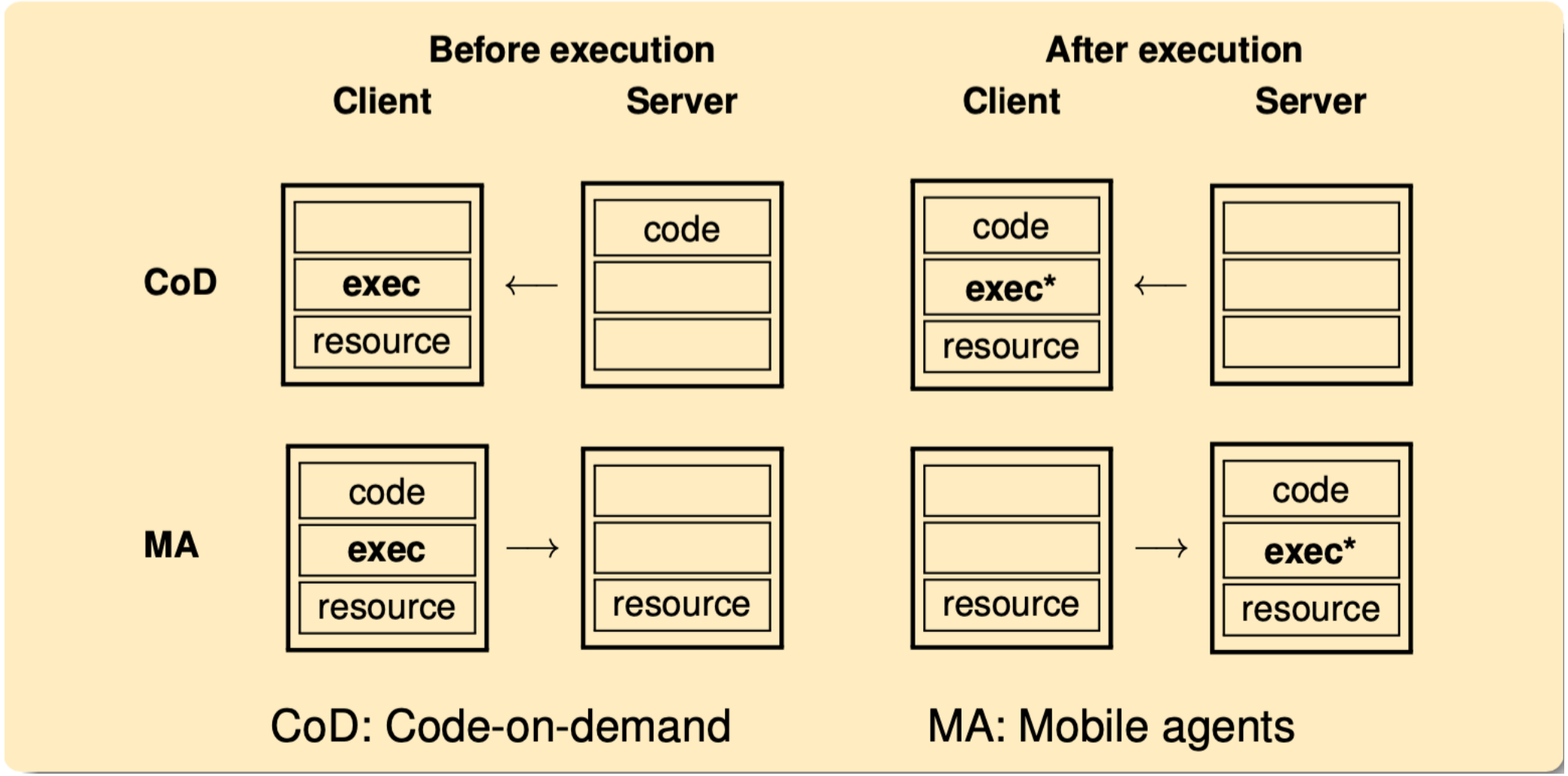
Strong mobility: Move component, including execution state

- Migration: move entire object from one machine to the other
- Cloning: start a clone, and set it in the same execution state

Models for code migration



Models for code migration



Migration in heterogeneous systems

Main problem

- The target machine may not be suitable to execute the migrated code
- The definition of process/thread/processor context is highly dependent on local hardware, operating system and runtime system

Only solution: abstract machine implemented on different platforms

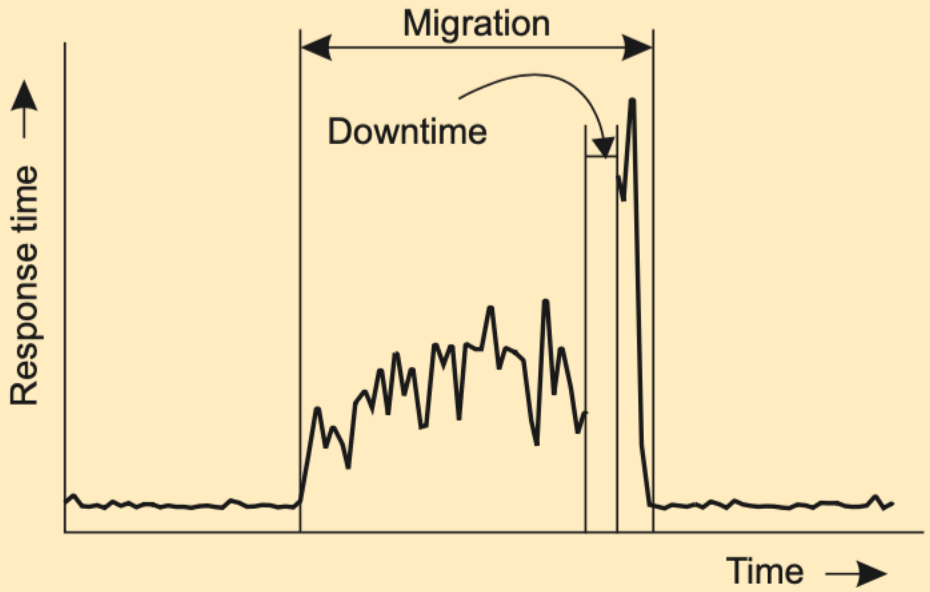
- Interpreted languages, effectively having their own
- VM Virtual machine monitors

Performance of migrating virtual machines

Main problem

A complete migration may actually take tens of seconds.
We also need to realize that during the migration, a service will be completely unavailable for multiple seconds.

Measurements regarding response times during VM migration



Scalability

- *Question:* How can you scale the server capacity?
- Buy bigger machine!
- Replicate
- Distribute data and/or algorithms
- Ship code instead of data
- Cache

A To-Do List

- Read Chapters 3.3, 3.4, 3.5
- Building your team