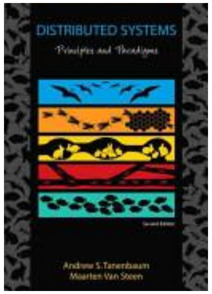
Chapter 3: PROCESSES THREADS

Processes and Threads in Distributed Systems



Thanks to the authors of the textbook [**TS**] for providing the base slides. I made several changes/additions.
These slides may incorporate materials kindly provided by Prof. Dakai Zhu.
So I would like to thank him, too.

Turgay Korkmaz
korkmaz@cs.utsa.edu

Chapter 3: PROCESSES THREADS

- THREADS
  - Introduction to Threads
  - Threads in Distributed Systems
- VIRTUALIZATION
  - The Role of Virtualization in Distributed Systems
  - Architectures of Virtual Machines
- CLIENTS and SERVERS
  - Client-Side Software for Distribution Transparency
  - Server Clusters and their Management
- CODE MIGRATION
  - Approaches to Code Migration
  - Migration and Local Resources
  - Migration in Heterogeneous Systems

Objectives

- To understand threads and related issues in DS
- To understand the role of virtualization in DS
- To learn general design issues for clients and servers in DS
- To understand code migration and its implications

Introduction

- We already studied processes in OS part, where the key issues were:
  - process management, scheduling, synchronization…
- We also studied threads (sgg-ch4)
  - User-level, kernel-level implementations, thread pool..
- Let us look at other equally important issues in the context of DS
  - Threads in DS
  - Client-server design
  - Code Migration

Thread Review

- Contrast Processes and Threads
  - Different address space vs. ….
  - CPU transparently switches processes vs. ….
  - Concurrency is costly (context switch) vs. …
- Explain the advantages of threads
  - In case of a blocking call, multithreaded application can execute another thread
  - Exploit parallelism when executed on multiprocessor
  - Processes can only cooperate using IPC, requiring expensive context switch, while threads…
  - Make software development easier (e.g., editor example)

Thread Review

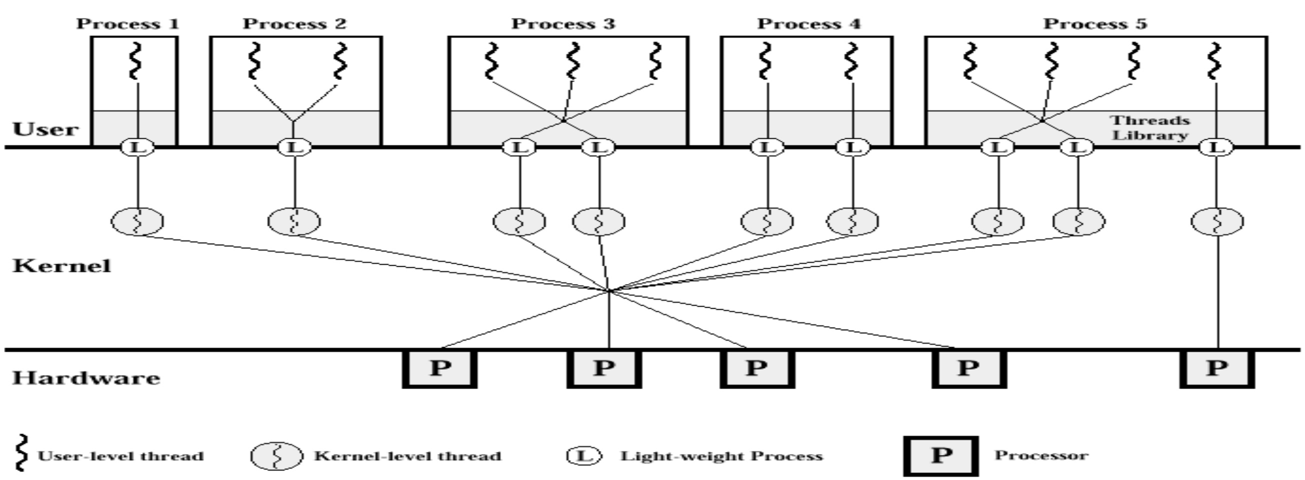
Contrast user-level and kernel-level threads

- +cheap, easy to create/destroy threads (memory allocation/release)
- +context switch is done in a few instruction (no need to change MMU, TLB, etc)
- - blocking call will block the entire process (so no benefit in editor :)

- + Circumvent problems in user-level threads
- - requires system call for every thread op
- -switching thread context is as expensive as switching processes
- Use hybrid form Lightweight process (LWP)

Thread Review
Light-Weight Process (LWP)

- **Lightweight process (LWP):** intermediate structure
  - **Virtual processor**: can execute user-level threads
  - **Each LWP attaches to a kernel thread**
- Multiple user-level threads → a single LWP
  - Normally from the same process
- **A process may be assigned multiple LWPs**
- OS schedules kernel threads (hence, LWPs) on the CPU



LWP: Advantages and Disadvantages

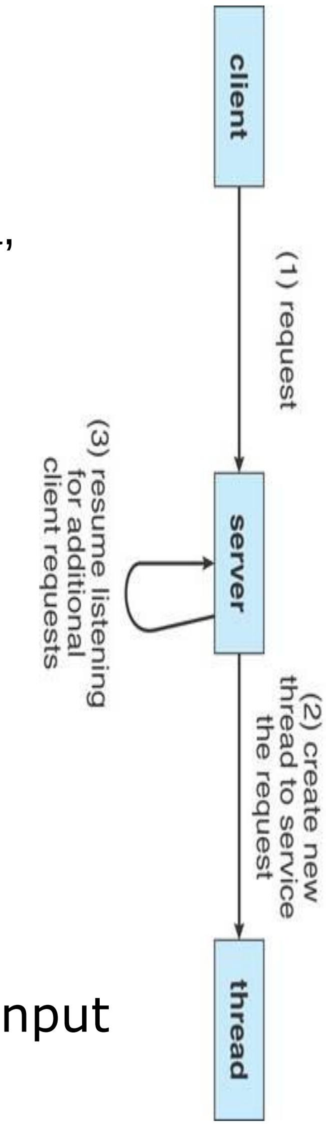
- + User level threads are easy to create/destroy/sync
- + A blocking call will not suspend the process if we have enough LWP
- + Application does not need to know about LWP
- +LWP can be executed on different CPUs, hiding multiprocessing
- Occasionally, we still need to create/destroy LWP (as expensive as kernel threads)
- Makes up calls (scheduler activation)
  - + simplifies LWP management
  - - Violates the layered structure

THREADS IN DISTRIBUTED SYSTEMS

Threads in Distributed Systems

Example: Web client and server

- **Client** (browser) starts communication in a thread. While it is waiting or getting the content, the other threads can do something else (e.g., display incoming data, allow users to click links, get different objects etc.)
  - Allow blocking systems calls without blocking the entire process
- **Server** creates a new thread to service a request.
  - Simplifies code (retains the idea of sequential process using blocking call)
  - Makes it easy to exploit parallelism




```
while()
{
   keyboard input
   socket1 input
}
```

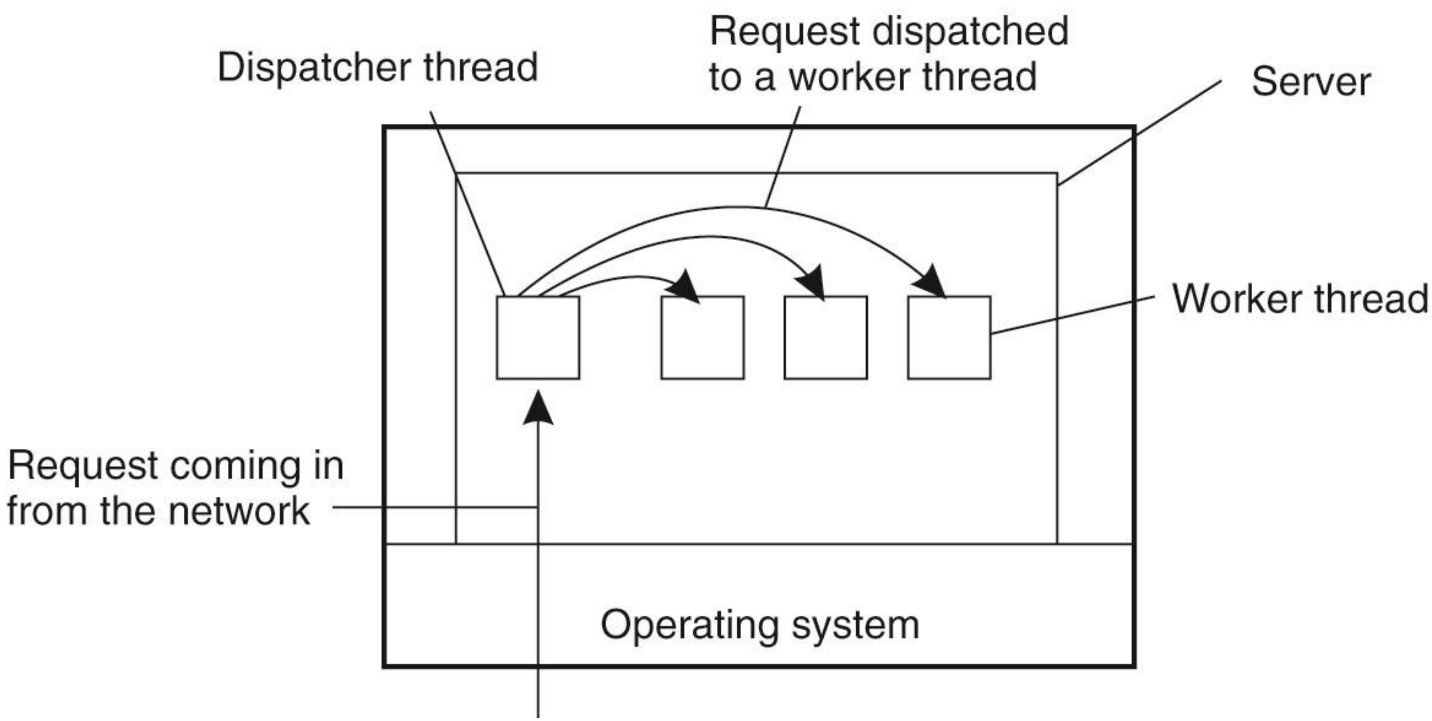
vs.

```
while()
{
   keyboard input
   ...
}
```

```
while()
{
   socket1 input
   ...
}
```

Threads in Distributed Systems

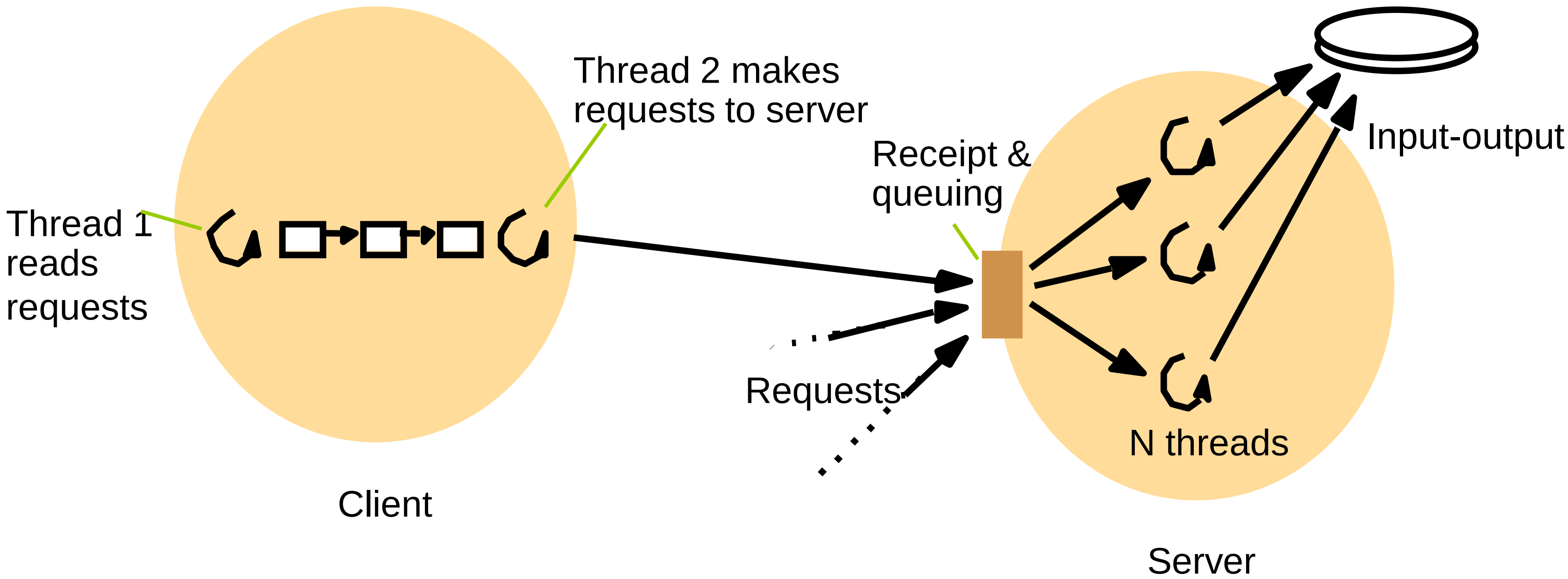
Another example: File server



Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls

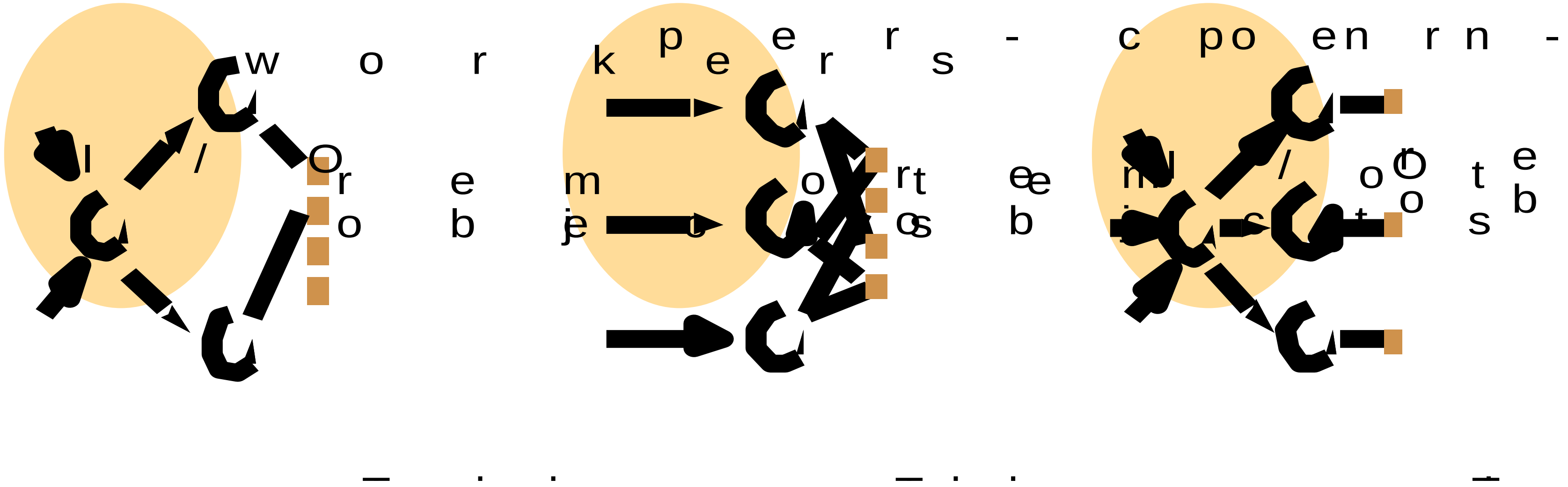
Threaded Implementations

- Use multiple threads to improve performance



How should the server handle the incoming requests?

Threaded Servers



Performance of Threaded Programs

- **Assumptions**
  - A single CPU & single disk system
  - CPU and disk can work concurrently
- Suppose that the processing of each request
  - Takes X seconds for computation; and
  - Takes Y seconds for reading data from I/O disk
- For single-thread program/process
  - What is the maximum throughput (i.e., the number of requests can be processed per second)?

Performance of Threaded Programs (cont'd)

- Suppose multi-thread implementation
  - Single CPU & single disk system
  - How many threads should be used?
    - Excessive number of threads → higher overhead
    - **Optimal** number of threads to be used
  - What is the maximum throughput (i.e., the number of requests can be processed per second)?
- Where is the bottleneck?
  - The slowest component determines the performance
- How to improve **without** extra hardware?
  - If I/O is slow → use main memory as data cache

Performance of Threaded Programs (cont)

- What about m-CPU and n-disk system
  - How many threads should be used?
  - What is the maximum throughput (i.e., the number of requests can be processed per second)?
- How to achieve a given throughput?
  - Balanced number of CPUs and I/O disks: m vs. n

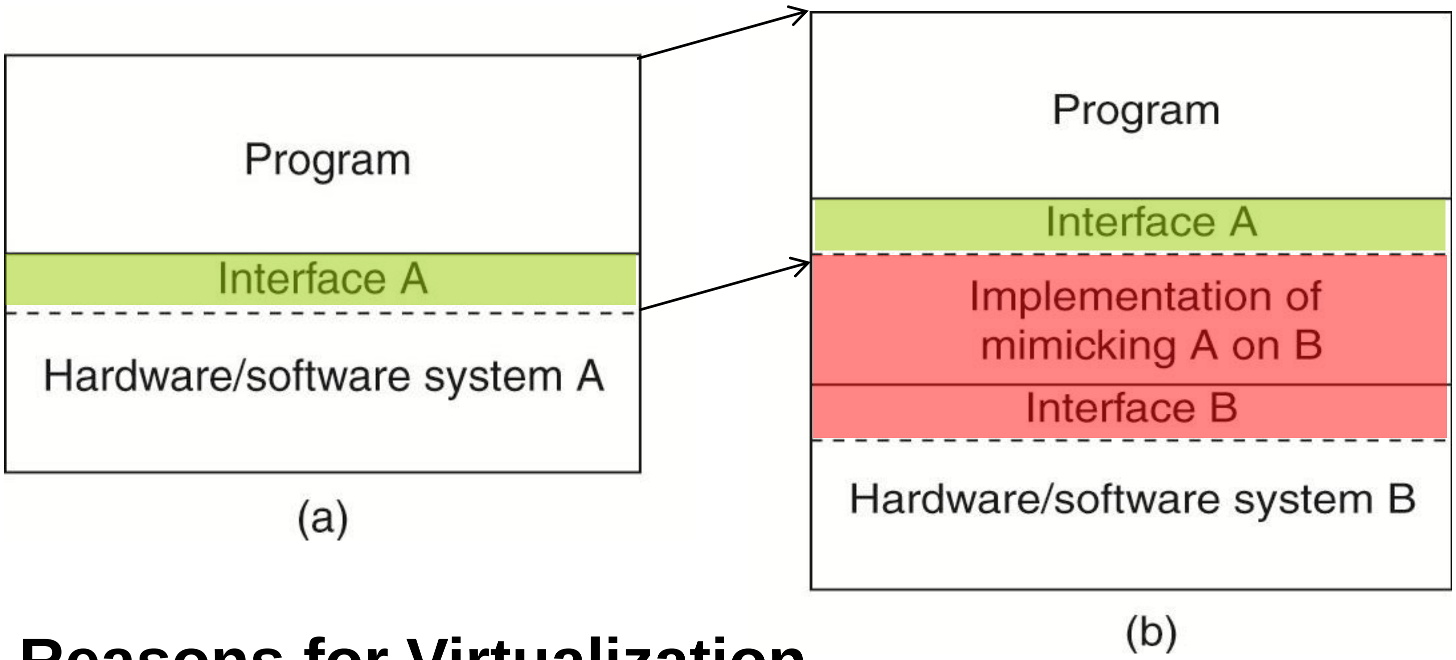
Parallelism among multiple threads on a single CPU is an illusion!

Generalization of this illusion to other resources is…

VIRTUALIZATION

The Role of Virtualization in DS

- Extend or replace an existing interface so as to mimic the behavior of another system

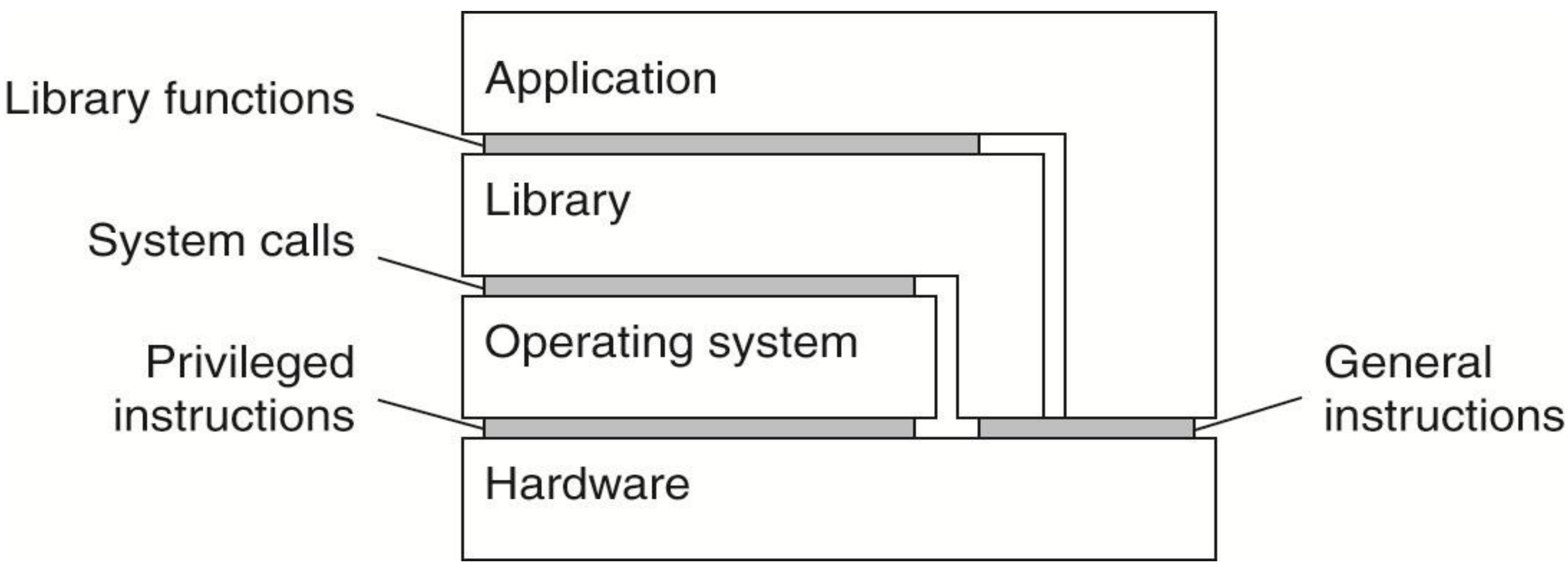


Reasons for Virtualization

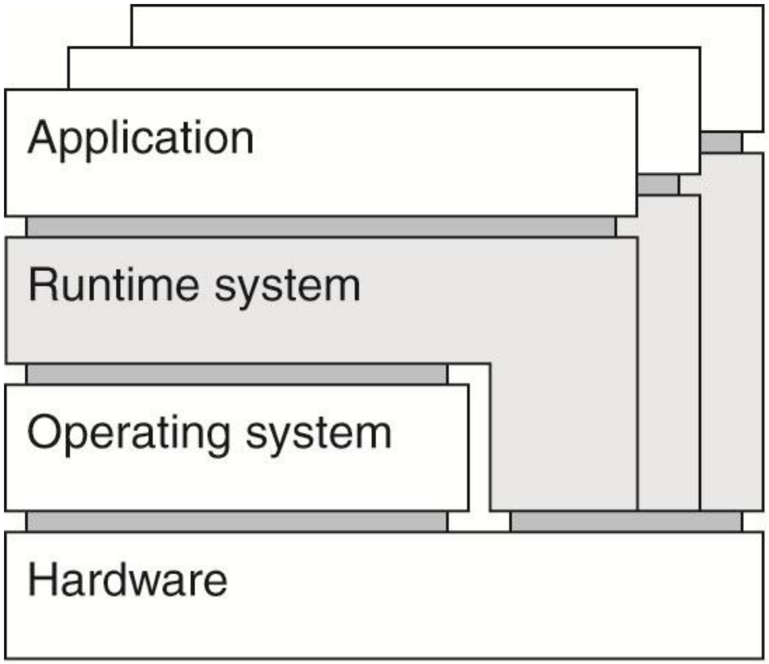
1. Hardware changes faster than software
2. Ease of portability and code migration
3. Isolation of failing or attacked components

Architecture of VMs

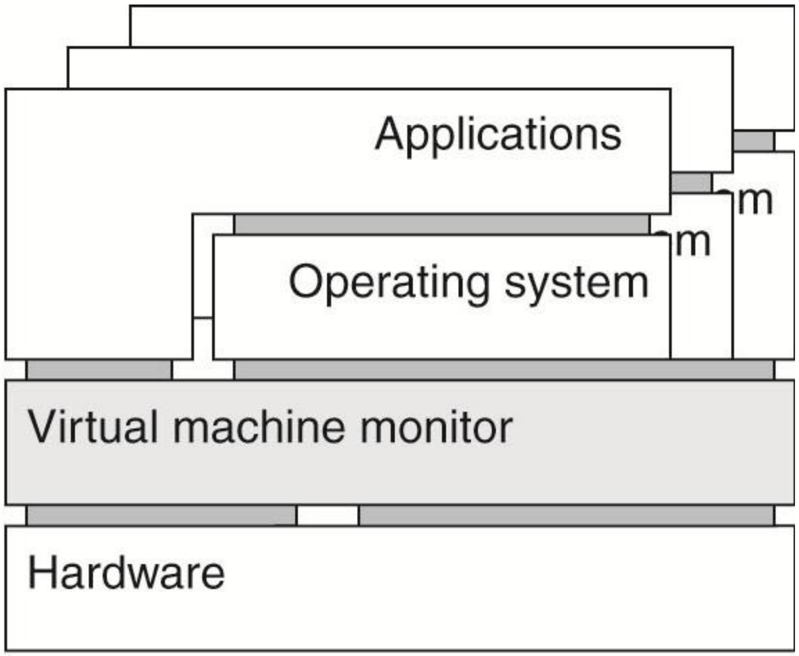
- Virtualization can take place at very different levels, strongly depending on the interfaces offered by computer systems
- The essence of virtualization is to mimic the behavior of these interfaces



Architecture of VMs (cont'd)



(a)



(b)

Process VM: A program is compiled to intermediate (portable) code, which is then executed by a runtime system (Example: Java VM).

VM Monitor (VMM): A separate software layer mimics the instruction set of hardware. So a complete operating system and its applications can be supported (Example: VMware, VirtualBox).

VM Monitors on operating systems

Practice

- We're seeing VMMs run on top of existing operating systems.
- Perform binary translation: while executing an application or operating system, translate instructions to that of the underlying machine.
- Distinguish sensitive instructions: traps to the original kernel (think of system calls, or privileged instructions).
- Sensitive instructions are replaced with calls to the VMM.

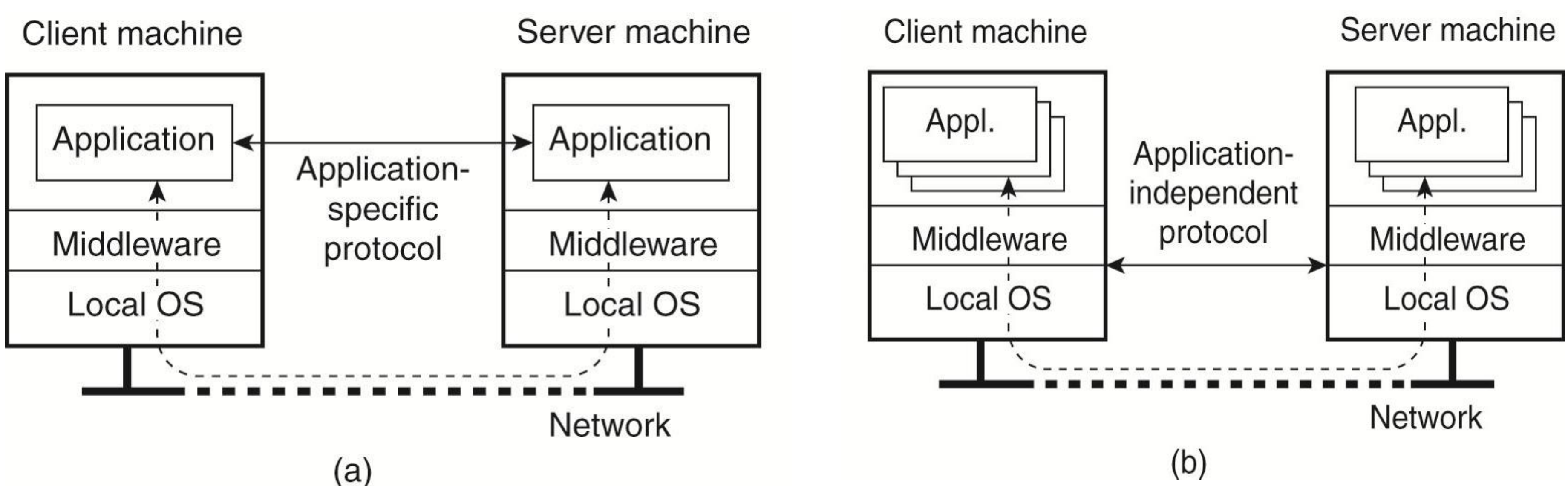
Very important for DS:

  - reliability, security, isolation, portability

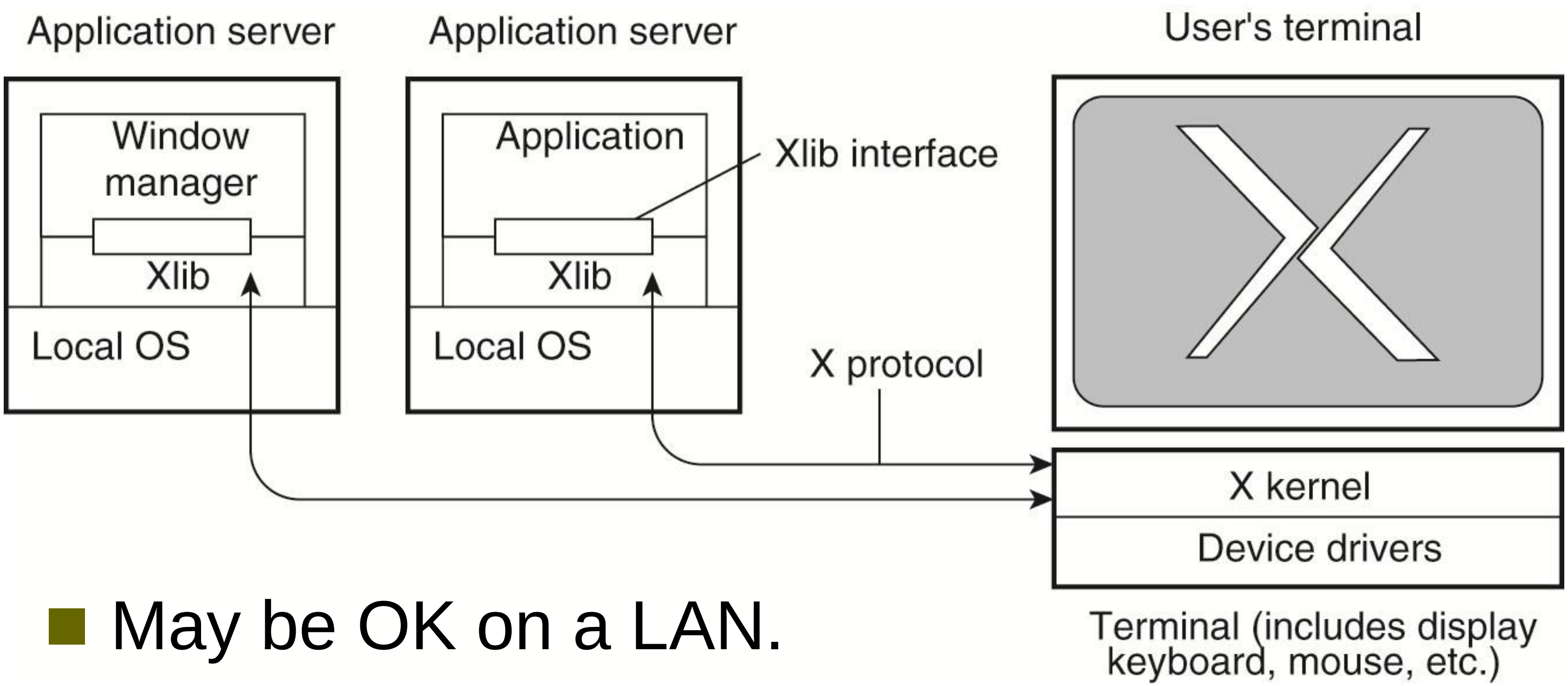
CLIENTS

Clients: User Interfaces and Communication Protocols

- A major part of client-side software is to develop a (graphical) user interfaces (software engineering)
- The other major part is communication protocols that make client to interact with the remote server
  - Application-specific protocols: -/+?
  - General (application-independent) solutions: -/+?



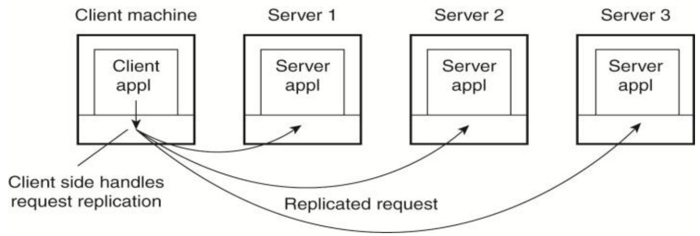
Example: The XWindow System



- May be OK on a LAN.
- How about WAN?
  - Re- engineer the protocol to avoid delay and need for excessive bandwidth for bitmaps
    - Cashing, (de)compression, consider app specific data

Clients: Distributed Transparency

- access transparency: client-side stubs for RPCs provides the same interface at server
- location/migration transparency: server let client-side software to know when it changes location, so client can hide it from user and keep track of actual location
- replication transparency:



  client stub sends multiple request to replicated servers and collect incoming responses
- failure transparency: client can try to re-transmit a request to mask server and communication failures

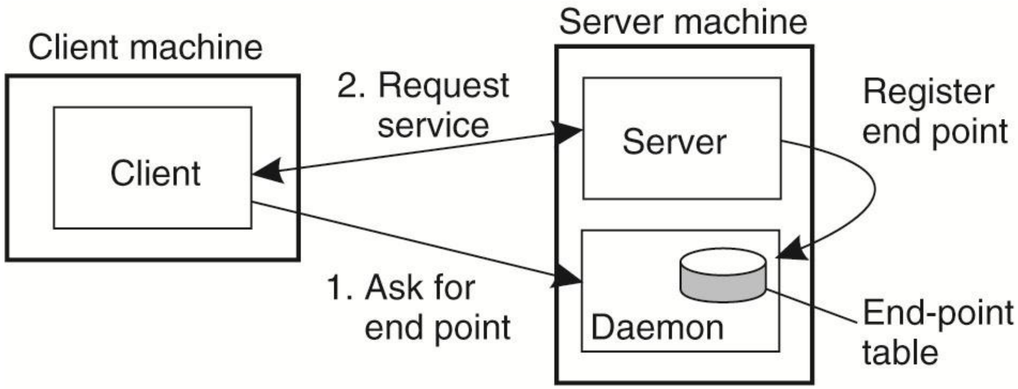
SERVERS

General Design issues

- A server is a process that
  - waits for incoming service requests from clients,
  - takes care of the requests, and
  - sends results back to clients
- Iterative vs. Concurrent servers
- Where/how clients connect servers
  - Each server listen to a specific transport address (e.g., IP address and port number)
  - Well-known services have a well known port number
  - What if the service is not offered on a well-known port

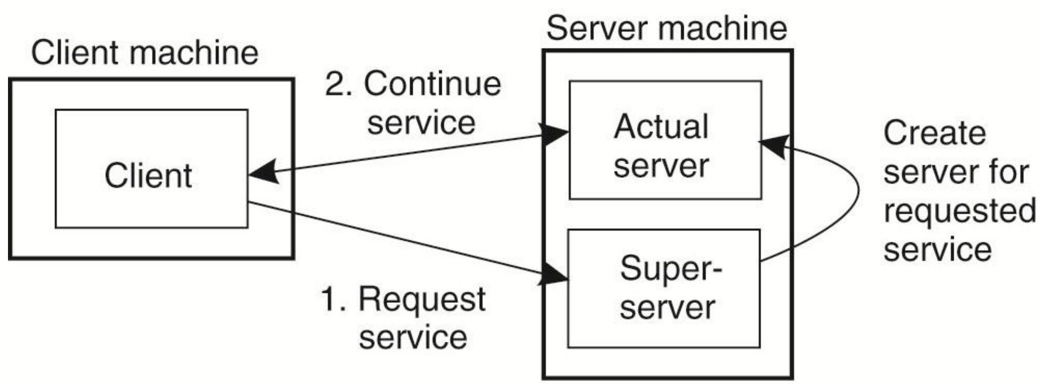
General Design issues (cont'd)

- Special daemons keep track of the port number of each service
  - If no client, waste of resources



(a)

- Superservers listen to several ports, i.e., provide several independent services (UNIX inetd)
  - + do not waste resources
  - - slow response time



(b)

General Design issues (cont'd)

How to interrupt a service? (e.g., downloading a webpage)

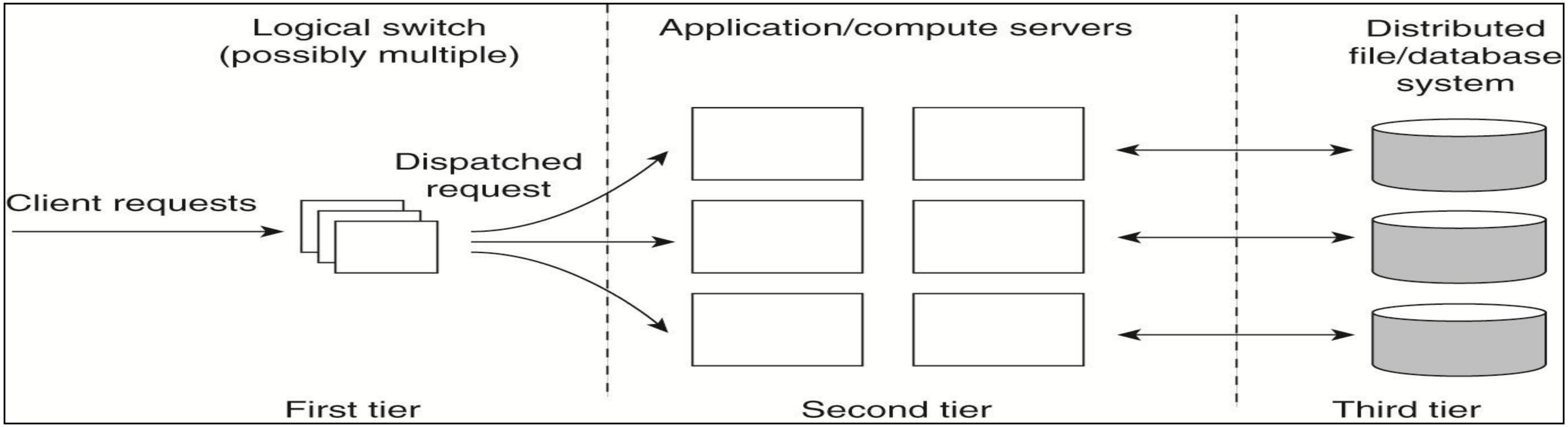
- User abruptly kills the client application
- Use separate port for urgent data
  - Server has a separate thread/process for urgent messages
  - Urgent message comes in → associated request is put on hold
  - Require OS supports priority-based scheduling
- Use out-of-band communication facilities of the transport layer:
  - Example: TCP allows for urgent messages in same connection
  - Urgent messages can be caught using OS signaling techniques

General Design issues (cont'd)

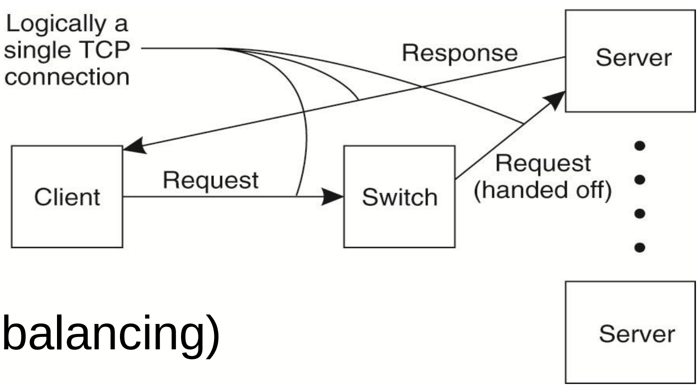
Should the server be stateless or stateful?

- **Stateless servers** *never* keep track of clients basic HTTP
  - Clients and servers are completely independent
  - State inconsistencies due to client or server crashes are reduced
  - Possible loss of performance, e.g., a server cannot anticipate client behavior (think of prefetching file blocks)
- **Stateful servers** keeps track of clients (e.g., file servers)
  - In case of crash, recovery is not an easy task
  - The performance of stateful servers can be extremely high, provided clients are allowed to keep local copies.
    - Record that a file has been opened, so that pre-fetching can be done
    - Knows which data a client has cached, and allows clients to keep local copies of shared data
- Soft state, temporary (session) states, cookies
- TCP, cookies in http?

Server Clusters

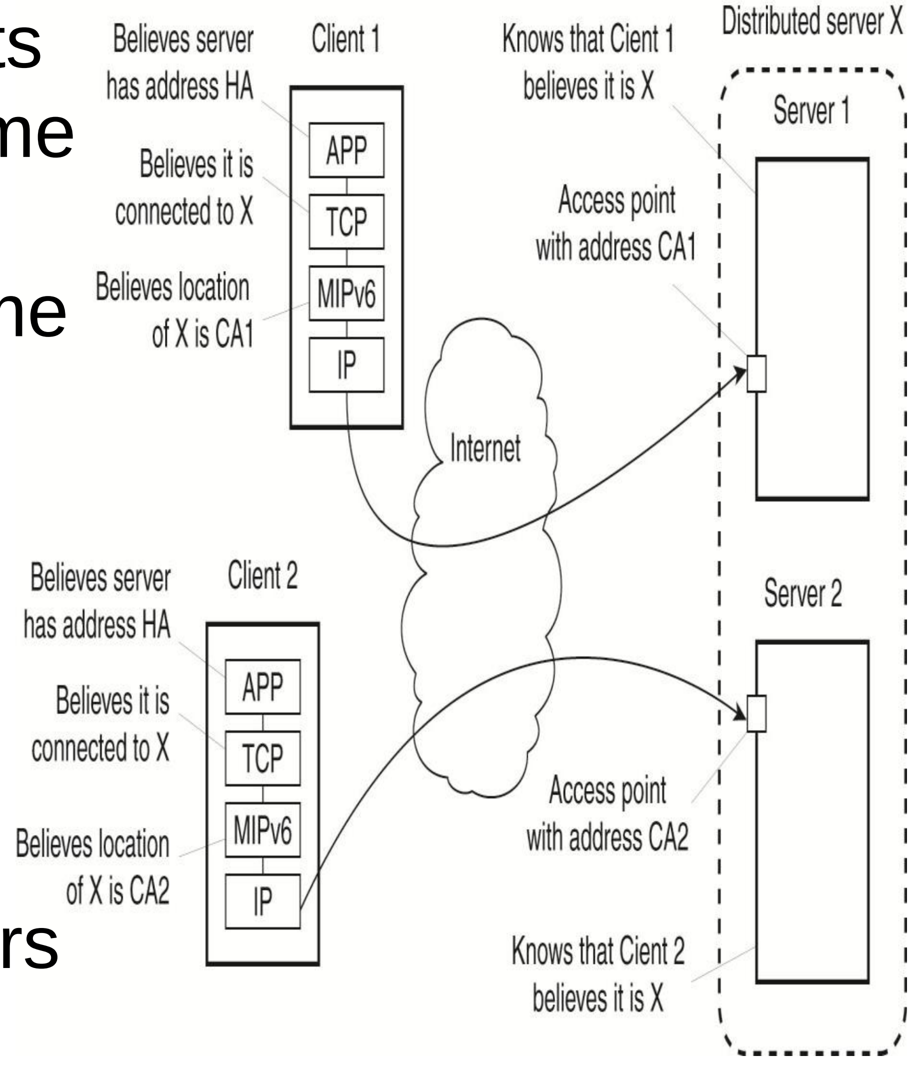


- The first tier hides the internal organization (e.g., TCP handoff)
- It passes requests to an appropriate server (important for load balancing)
- Could be the bottleneck
- **Challenge**: *how to replace this single point of failure by a fully distributed solution…*



Distributed Servers

- Add multiple access points having the same host name and DNS returns their address for the same name
- Clients can try different addresses if one fails
- - Still have static access points
- Stability and flexibility requires distributed servers
  - Mobile IP could be used

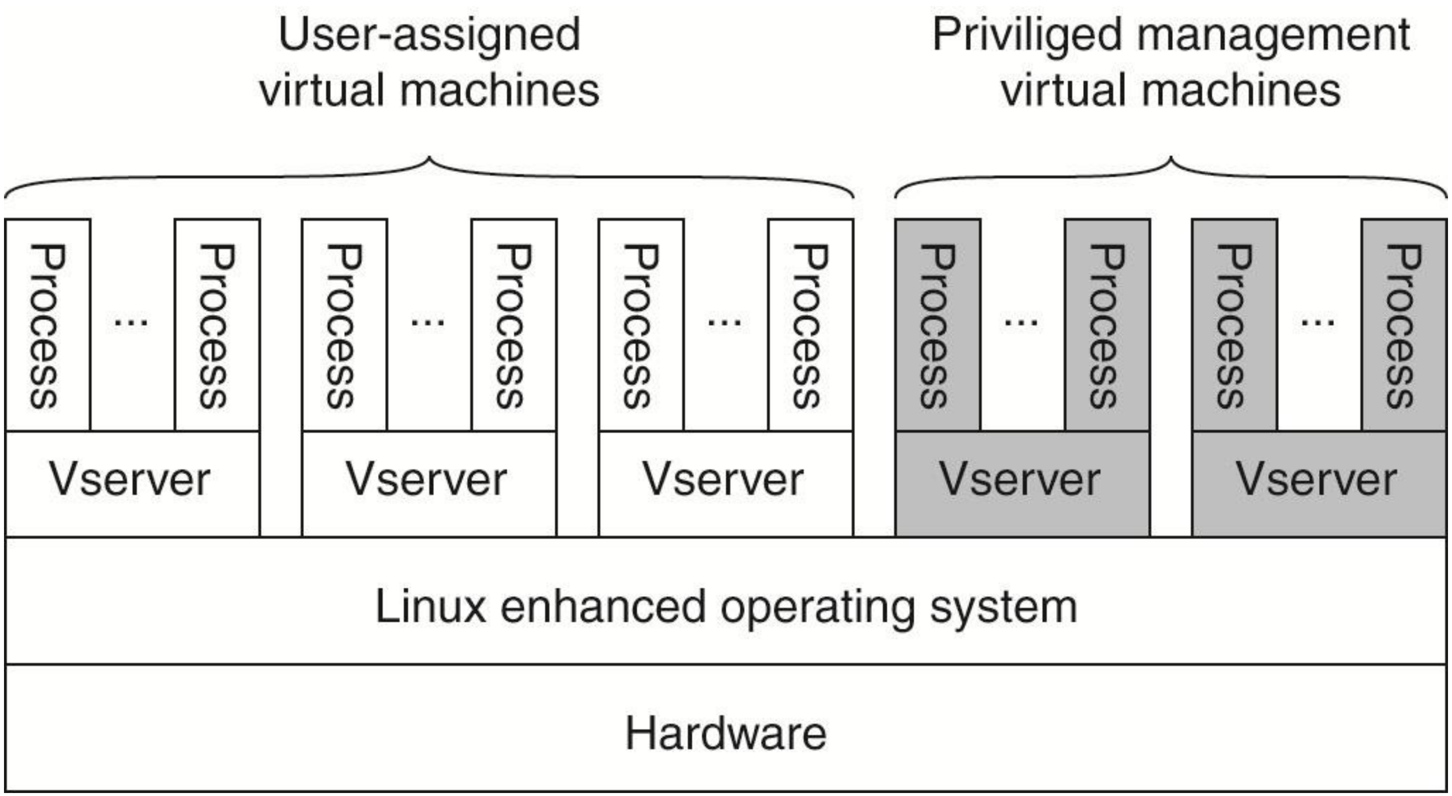


Managing Server Clusters

- Common approaches
  - Extend traditional management functions of a single machine so admin can log in and manage it
- Advanced forms
  - Centralized interface that hide the fact that admin needs to log into single machines
- Ad hoc
  - More works need to be done
  - Self-* solutions may help

Example: PlanetLab

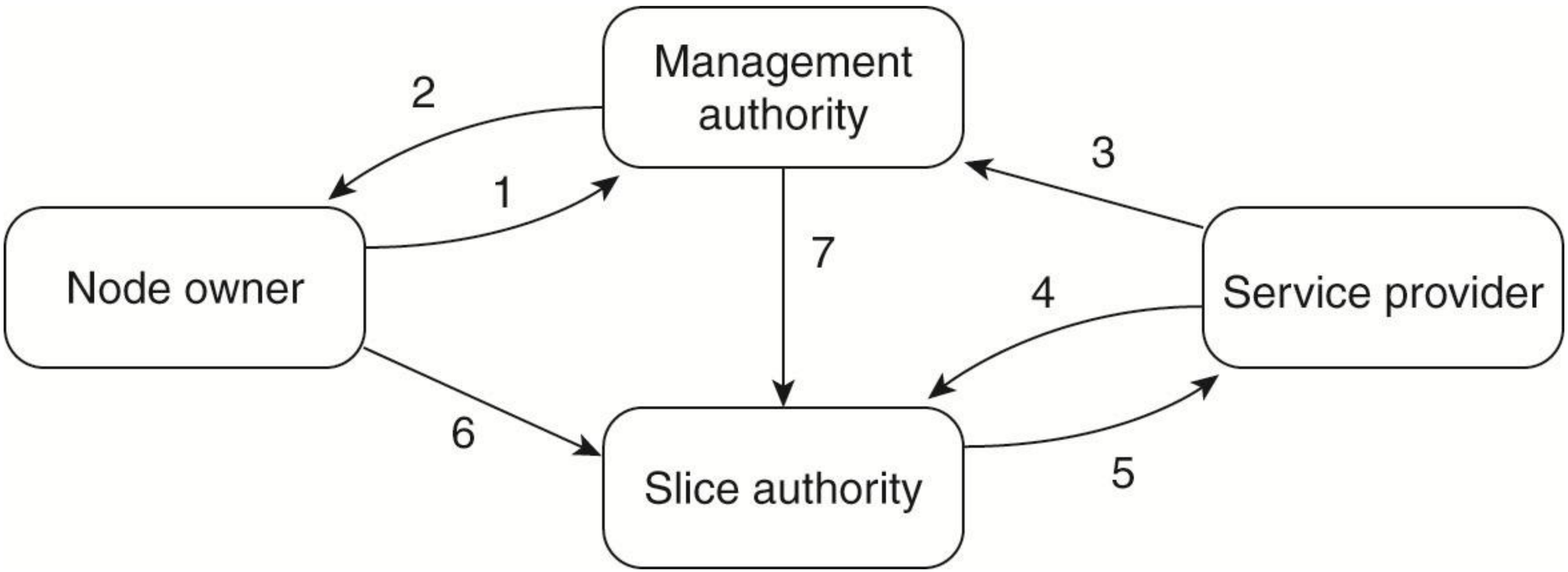
- The basic organization of a PlanetLab node.



PlanetLab (1)

- 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.

PlanetLab (2)



- Figure 3-16. The management relationships between various PlanetLab entities.

PlanetLab (3)

- Relationships between PlanetLab entities:
  - A node owner puts its node under the regime of a management authority, possibly restricting usage where appropriate.
  - A management authority provides the necessary software to add a node to PlanetLab.
  - A service provider registers itself with a management authority, trusting it to provide well-behaving nodes.

PlanetLab (4)

- 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.

So far we discussed passing data…

How about passing programs even when they are being executed…

CODE MIGRATION

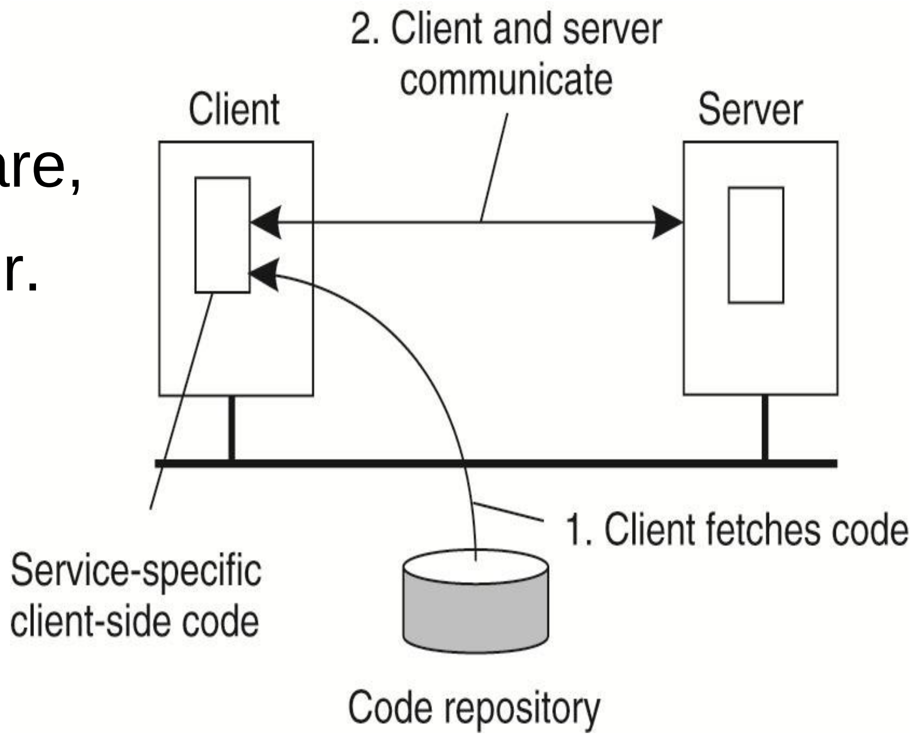
Reasons for Migrating Code

- Performance
  - Move processes from heavily-loaded to lightly-loaded
  - Minimize communication (e.g., JavaScript to check forms)
  - Exploit parallelism (e.g., mobile agent to search info)
- Flexibility
  - fetch the necessary software,
    and then invoke the server.

  + no need to pre-install sw

  + client-server protocols can
    be changed easily

  - Security (ch 9)

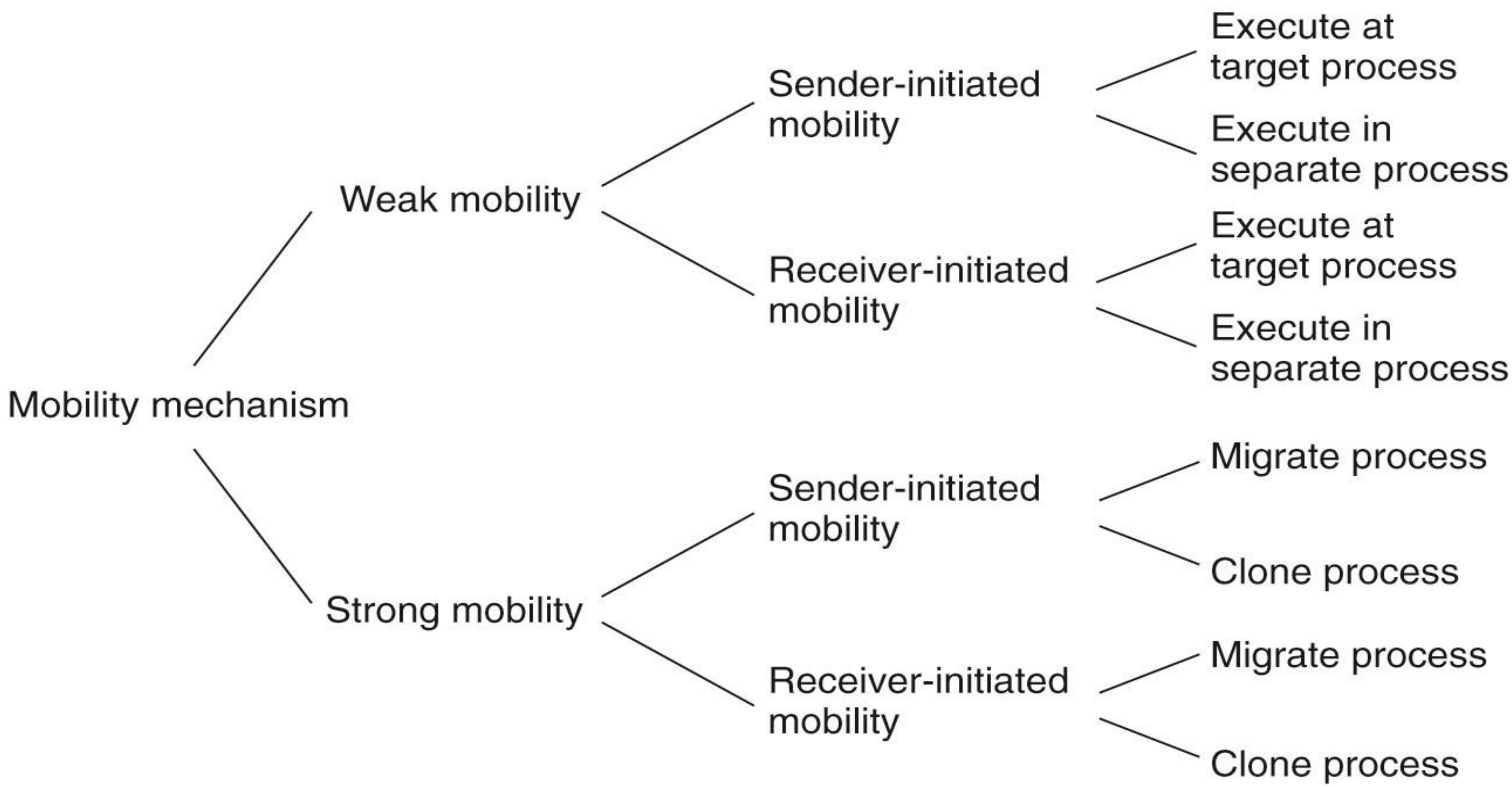


Models for Code Migration

- A process consists of three segments
  - Code (set of instructions, program)
  - Resource (external resources: files, printers, other processes)
  - Execution (private data, stack, program counter, registers)
- Weak vs. Strong mobility
  - Transfer only the code vs. transfer execution as well
  - Simple, easy vs. general, hard
- Sender-initiated vs. Receiver-initiated

  Code is at A and
  - A initiates migration vs. B initiates migration
  - Requires registration and authentication vs. simpler

Models for Code Migration (cont'd)



Migration and Local Resources

- Process to resource binding
  - The strongest form is by **identifier**
    - Requires a specific instance of a resource (URL, ftp server)
  - A weaker form is by **value**
    - Requires the value of a resource (cache entries, standard lib)
  - The Weakest form is by **type**
    - Requires a resource of specific type (monitor, printer)
- Resource types
  - **Un-attached** resource can be easily moves (data file)
  - **Fastened** resource can be moved but costly (local DB)
  - **Fixed** resource cannot be moved (local hard disk)
- Have nine combinations…

Migration and Local Resources (con’d)

Actions to be taken with respect to the references to local resources when migrating code to another machine.

		Resource-to-machine binding		
		Unattached	Fastened	Fixed
Process-to-resource binding	By identifier	MV (or GR)	GR (or MV)	GR
	By value	CP (or MV,GR)	GR (or CP)	GR
	By type	RB (or MV,CP)	RB (or GR,CP)	RB (or GR)

GR Establish a global systemwide reference
MV Move the resource
CP Copy the value of the resource
RB Rebind process to locally-available resource

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
  - Make use of an abstract machine that is implemented on different platforms
    - Interpreted languages, effectively having their own VM (Java)
  - Virtual machine migration

Migration in heterogeneous Systems cont'd

- Three ways to handle migration (which can be combined)
  - Pushing memory pages to the new machine and resending the ones that are later modified during the migration process.
  - Stopping the current virtual machine; migrate memory, and start the new virtual machine.
  - Letting the new virtual machine pull in new pages as needed, that is, let processes start on the new virtual machine immediately and copy memory pages on demand.