DISTRIBUTED SYSTEMS
Principles and Paradigms
Second Edition
ANDREW S. TANENBAUM
MAARTEN VAN STEEN

Chapter 3
Processes

Processes

OS support
Multithreading
Virtualization
Client-server organization
Process migration

Threads

an OS creates a number of virtual processors . . .

concurrency transparency . . .

thread context

Thread Usage in Nondistributed Systems

single threaded system
- blocking system call

multi threaded system
- blocking system call
- parallelism in multiprocessor systems

Thread Usage in Nondistributed Systems

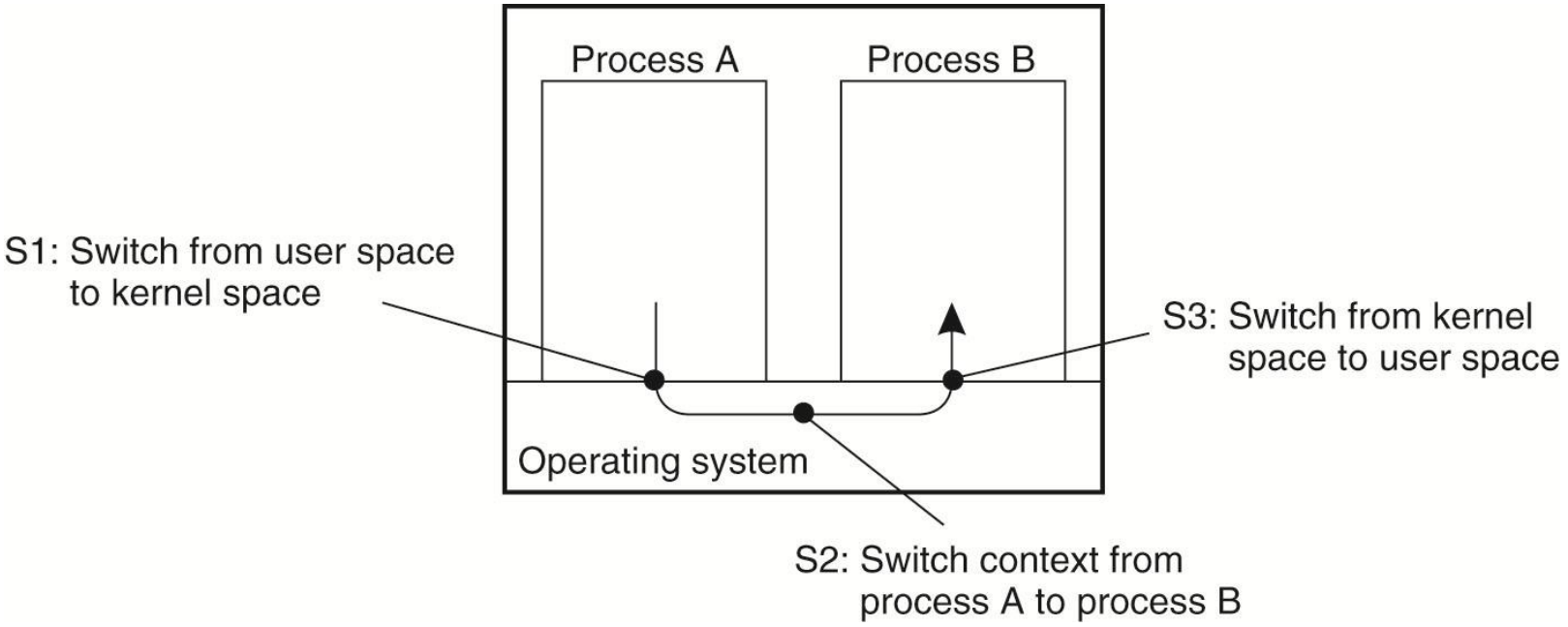

Figure 3-1. Context switching as the result of IPC.

Thread Implementation

thread package

two approaches to implementation

- thread library that is executed entirely in user mode
- implementing threads in the kernel

hybrid form of user-level and kernel-level threads

Thread Implementation

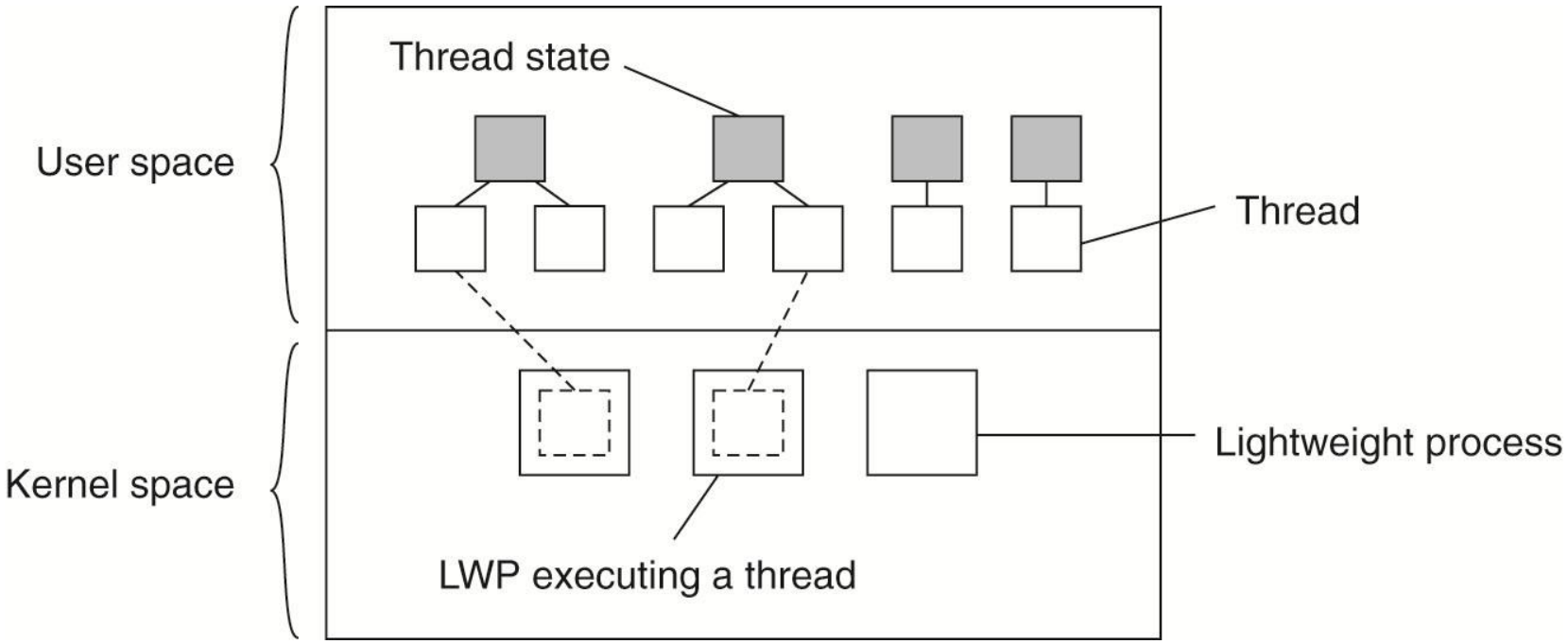

Figure 3-2. Combining kernel-level lightweight processes and user-level threads.

Threads in Distributed Systems

Multithreaded Clients

Multithreaded Servers

Multithreaded Servers

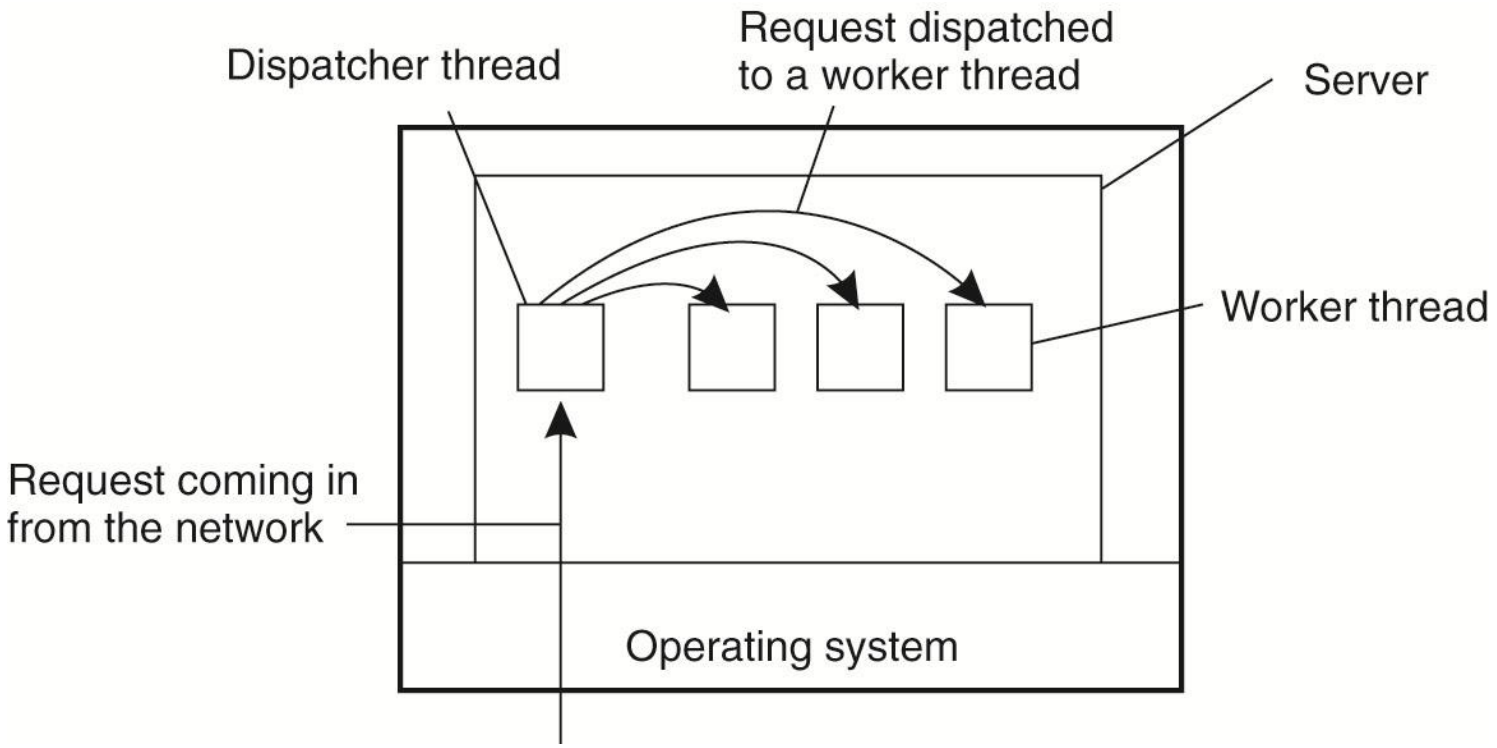

Figure 3-3. A multithreaded server organized in a dispatcher/worker model.

Multithreaded Servers

single threaded process

Multithreaded Servers

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

Figure 3-4. Three ways to construct a server.

Virtualization

resource virtualization

The Role of Virtualization in Distributed Systems

every computer systems offers a programming interface to higher level software

The Role of Virtualization in Distributed Systems

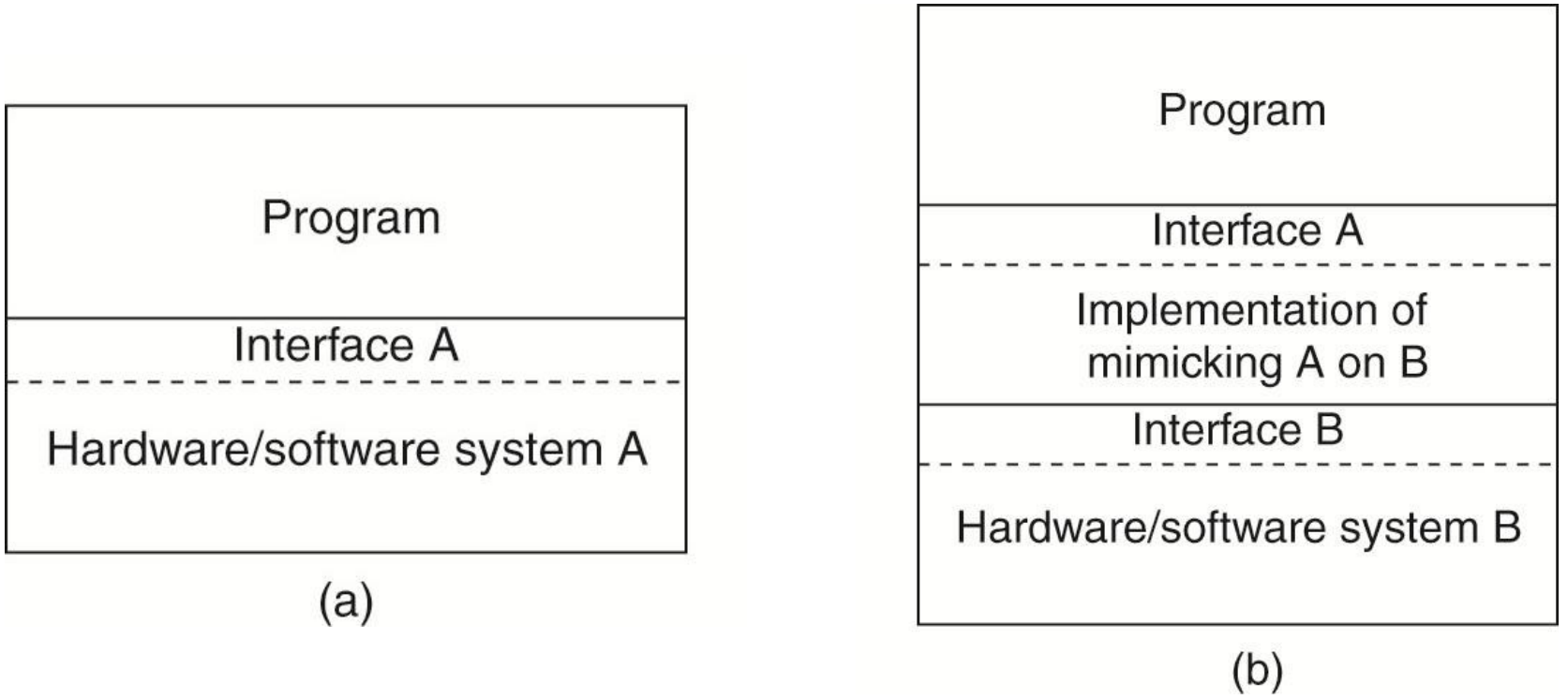

Figure 3-5. (a) General organization between a program, interface, and system. (b) General organization of virtualizing system A on top of system B.

The Role of Virtualization in Distributed Systems

supporting legacy software

Networking – the diversity of platforms and machines can be reduced

Architectures of Virtual Machines

Interfaces at different levels

- An interface between the hardware and software consisting of machine instructions
  - that can be invoked by any program.
- An interface between the hardware and software, consisting of machine instructions
  - that can be invoked only by privileged programs, such as an operating system.

Architectures of Virtual Machines

Interfaces at different levels

- An interface consisting of system calls as offered by an operating system.
- An interface consisting of library calls
  - generally forming what is known as an application programming interface (API).
  - In many cases, the aforementioned system calls are hidden by an API.

Architectures of Virtual Machines

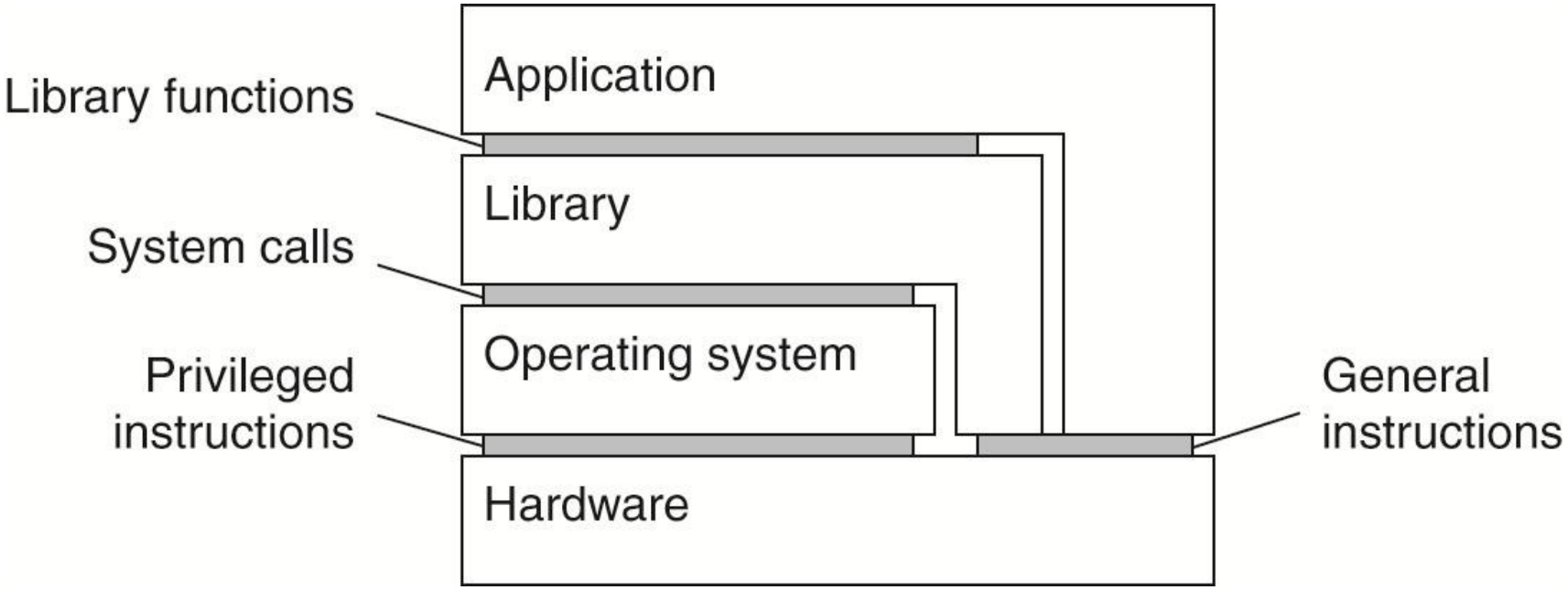

Figure 3-6. Various interfaces offered by computer systems.

Architectures of Virtual Machines

virtualization - two different ways

- process virtual machine
- virtual machine monitor (VMM)

Architectures of Virtual Machines

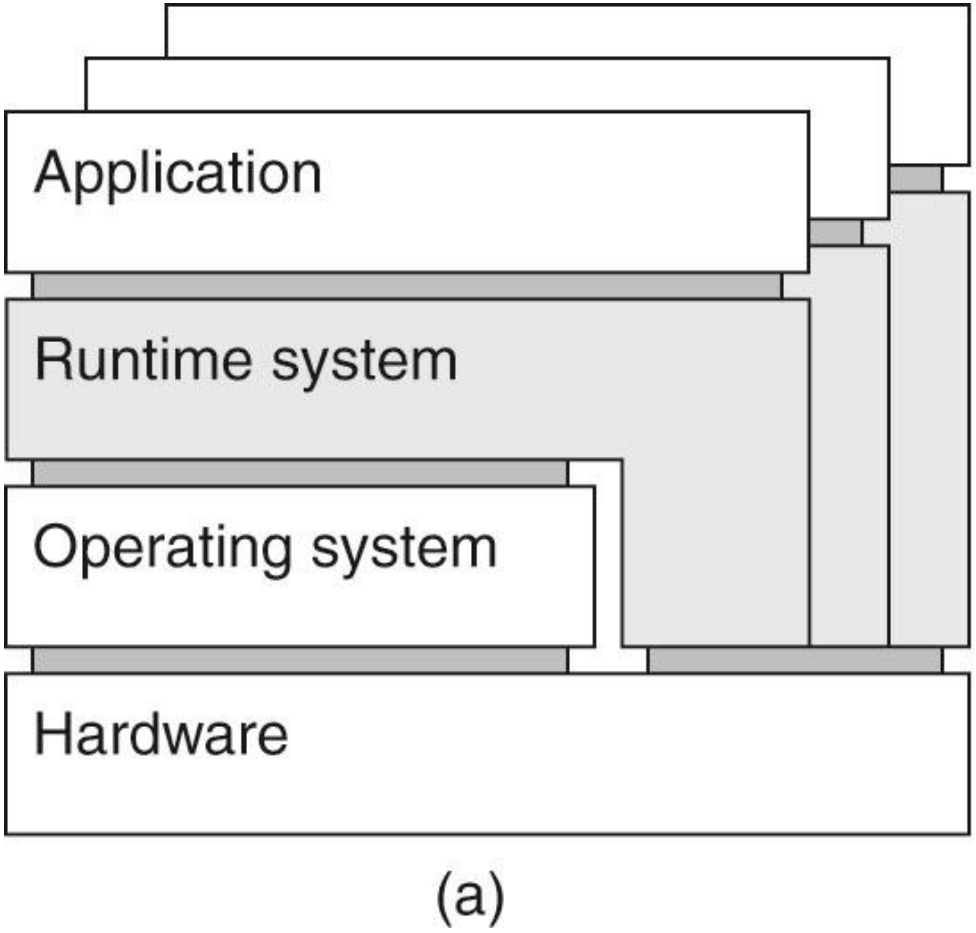

Figure 3-7. (a) A process virtual machine, with multiple instances of (application, runtime) combinations.

Architectures of Virtual Machines

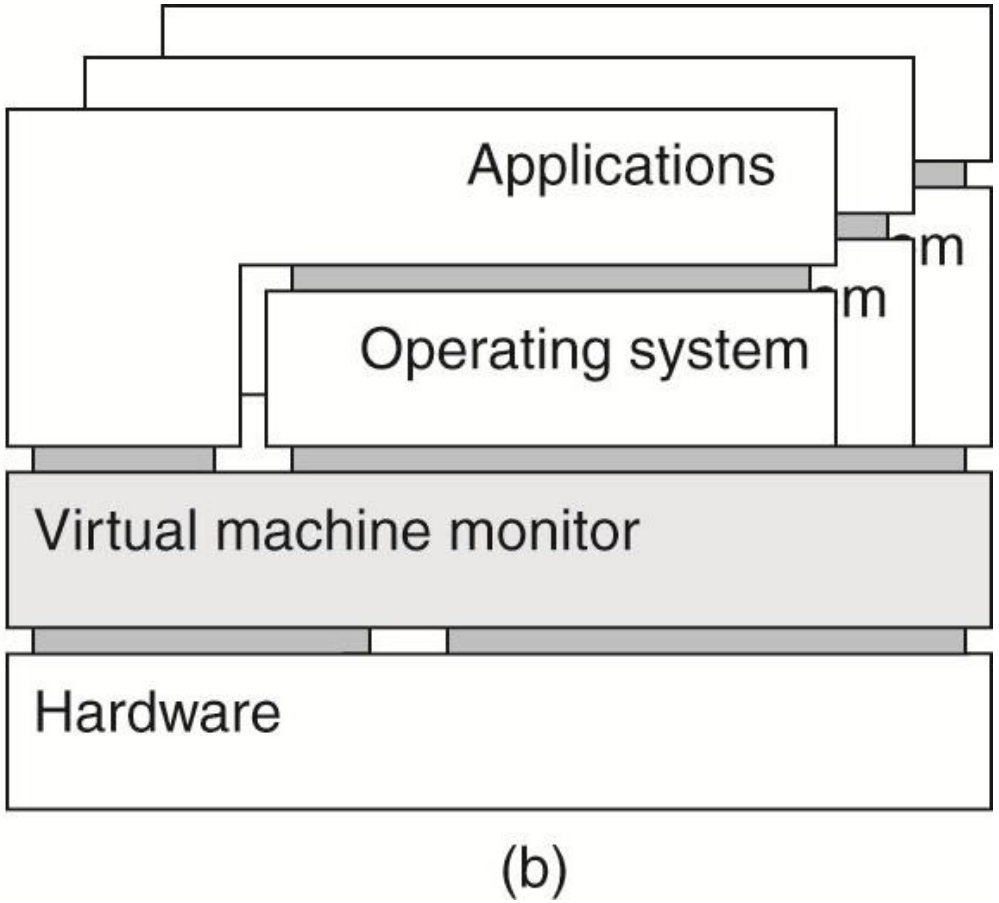

Figure 3-7. (b) A virtual machine monitor, with multiple instances of (applications, operating system) combinations.

Architectures of Virtual Machines

VMMs will become increasingly important in the context of reliability and security for (distributed) systems.

Clients

A major task of client machines is to provide the means for users to interact with remote servers.

Networked User Interfaces
- for each remote service the client machine will have a separate counterpart that can contact the service over the network.
- provide direct access to remote services by only offering a convenient user interface.
(thin-client approach)

Networked User Interfaces (1)

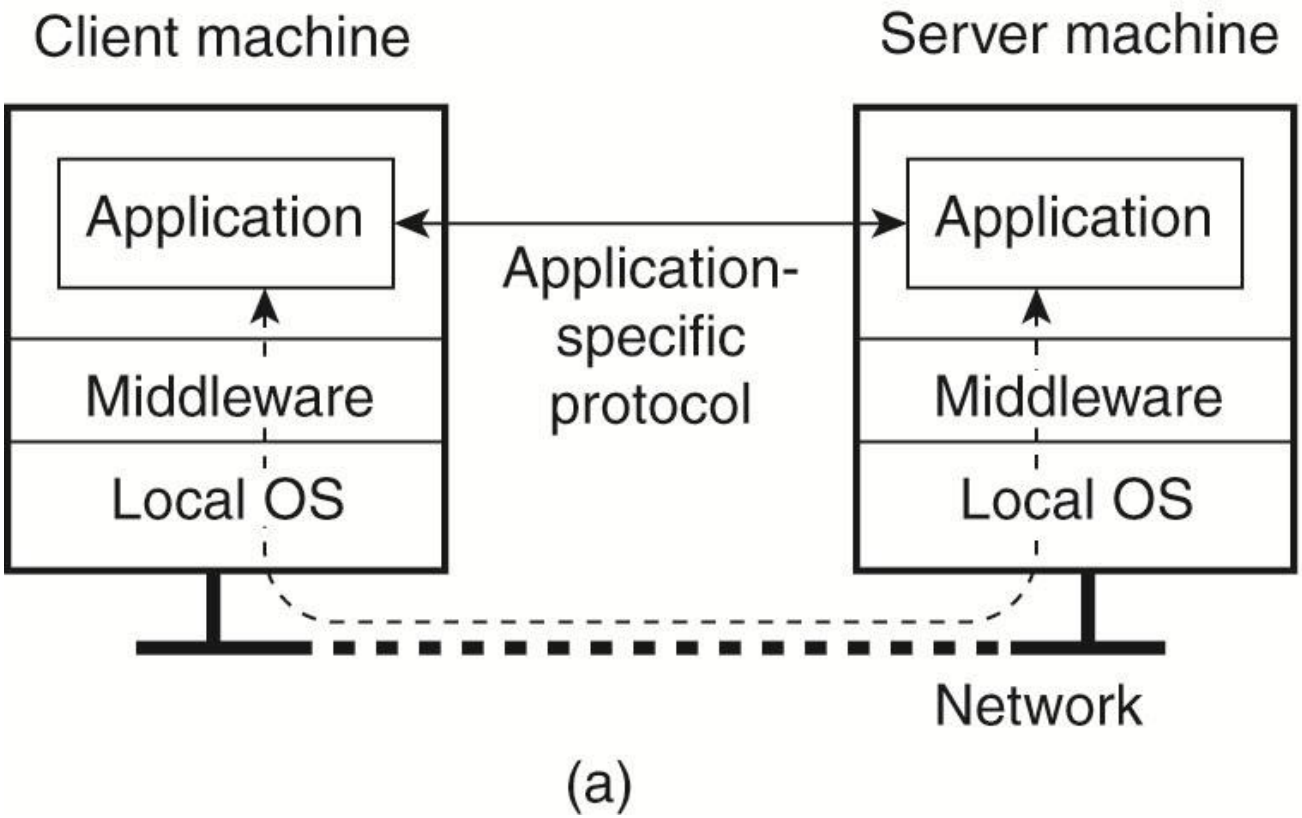

Figure 3-8. (a) A networked application with its own protocol.

Networked User Interfaces (2)

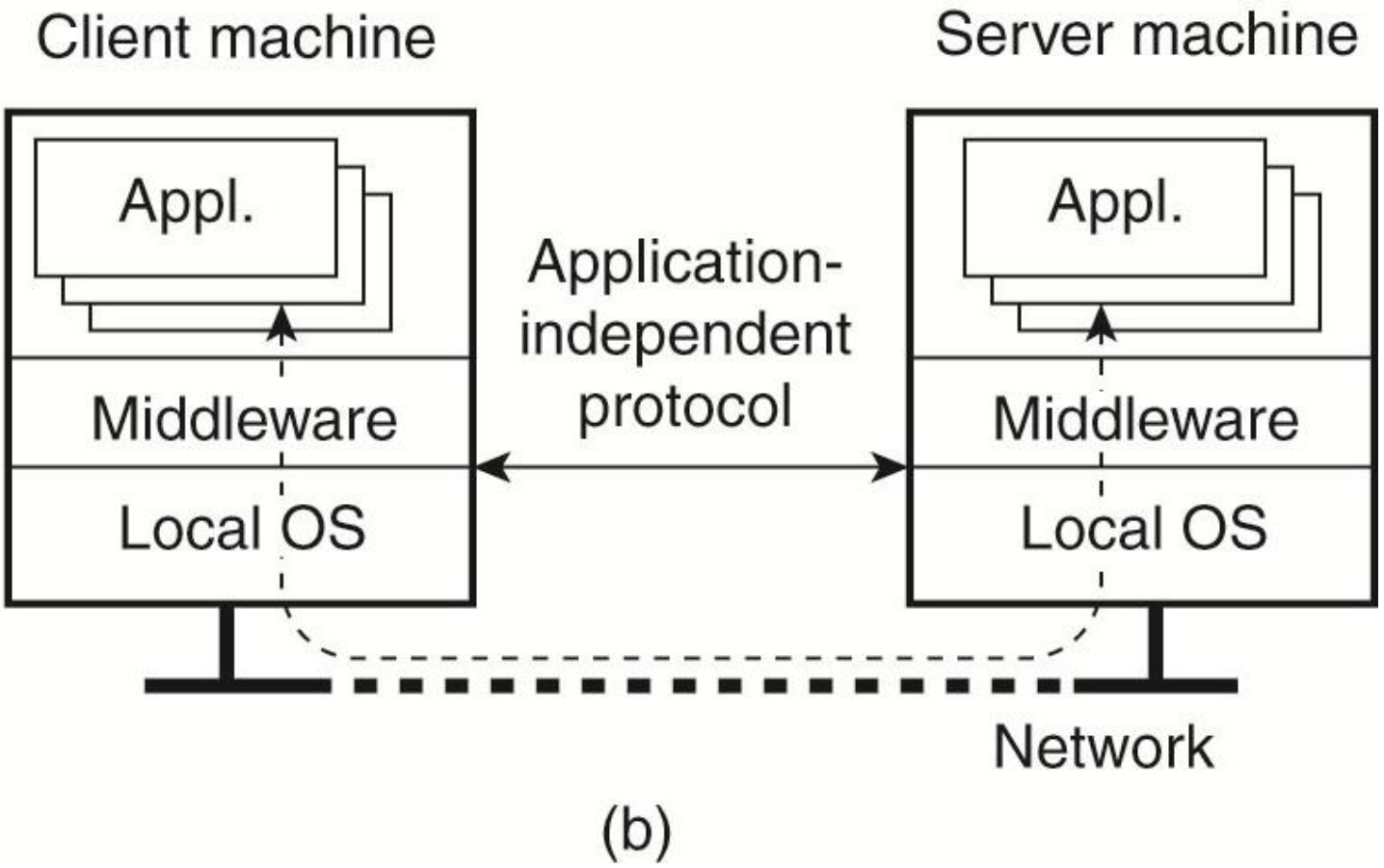

Figure 3-8. (b) A general solution to allow access to remote applications.

Client-Side Software for Distribution Transparency

A client should not be aware that it is communicating with remote processes.

- Access transparency - handled through the generation of a client stub.
- There are different ways to handle location, migration, and relocation transparency.

- Many distributed systems implement replication transparency by means of client-side solutions.

Client-Side Software for Distribution Transparency

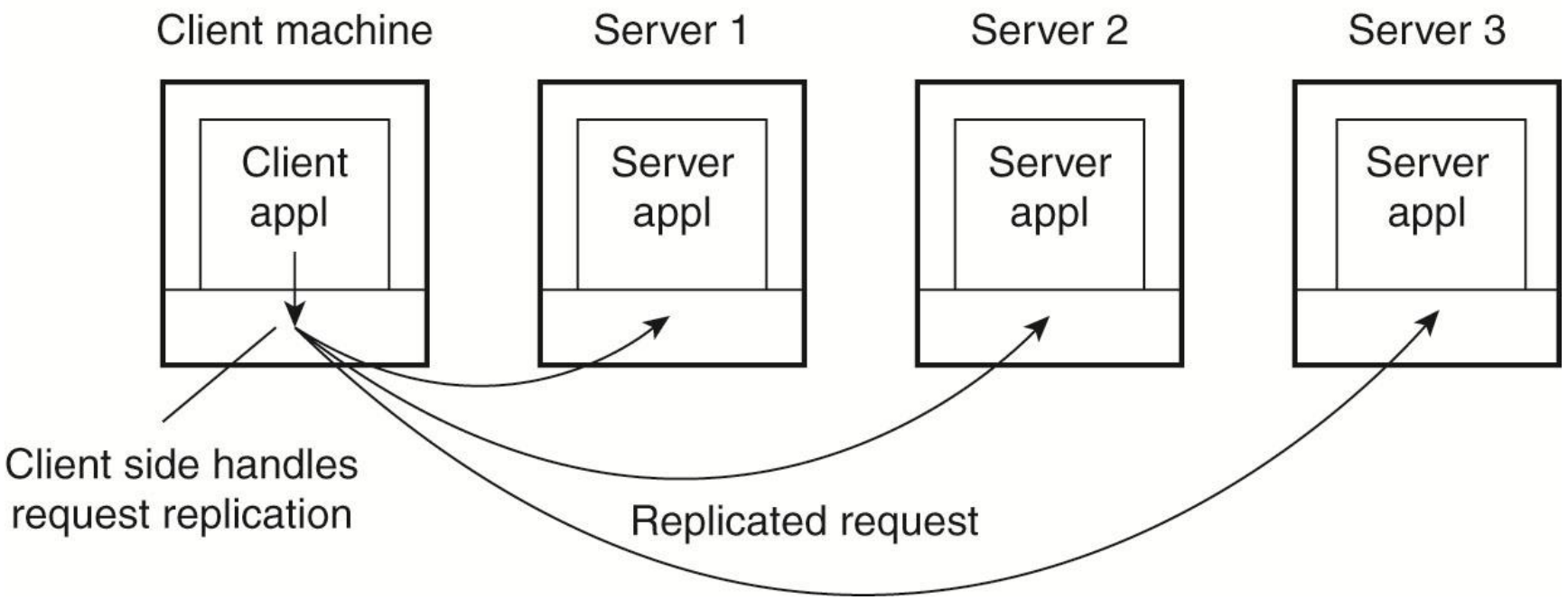

Figure 3-10. Transparent replication of a server using a client-side solution.

Servers

General Design Issues

There are several ways to organize servers

- iterative server
- concurrent server

Where clients contact a server? End point

There are many services that do not require a preassigned end point.

Servers - General Design Issues

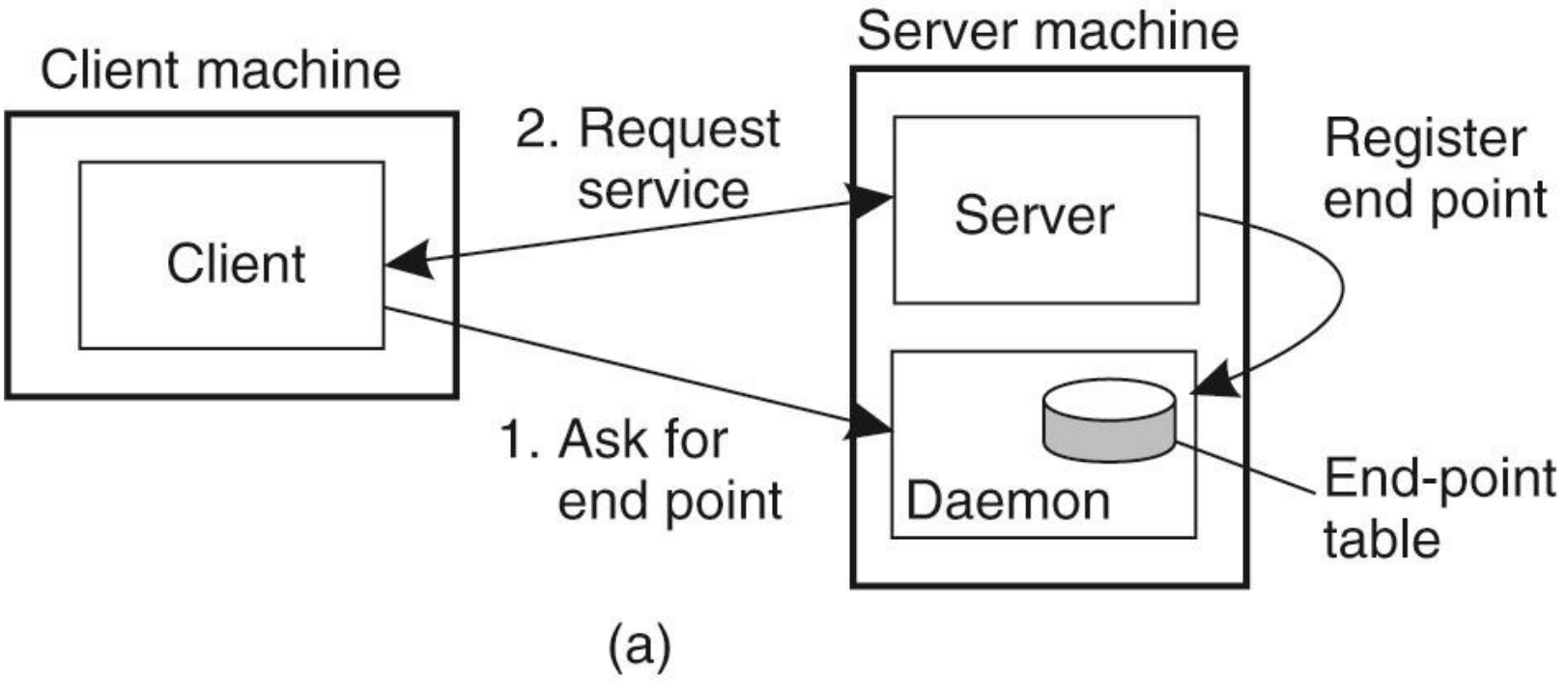

Figure 3-11. (a) Client-to-server binding using a daemon.

Servers - General Design Issues

It is common to associate an end point with a specific service.

However, actually implementing each service by means of a separate server may be a waste of resources.

Servers - General Design Issues

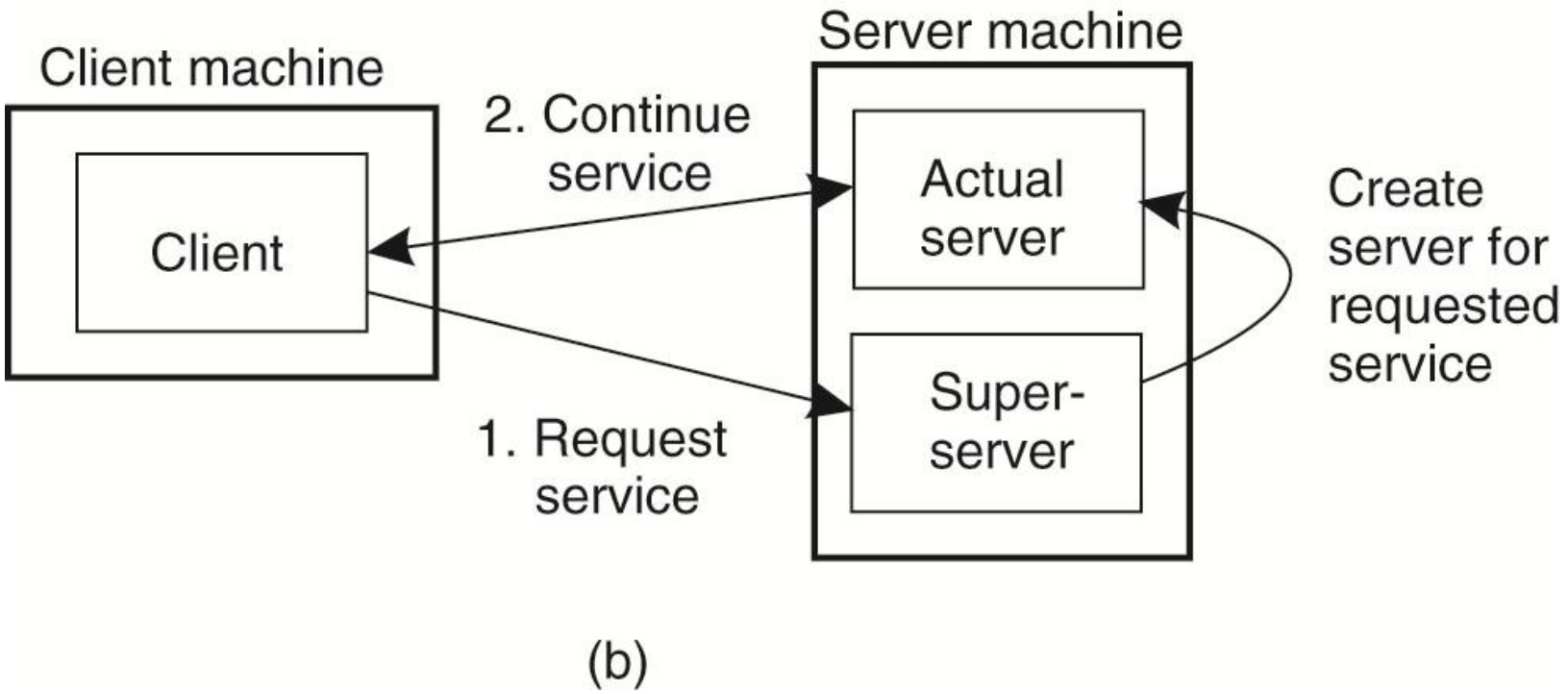

Figure 3-11. (b) Client-to-server binding using a superserver.

Servers - General Design Issues

Another issue is whether and how a server can be interrupted. Several ways:

- user abruptly exits the client application
- A much better approach - send **out-of-band** data (data that is to be processed by the server before any other data from that client)

A final, important design issue - whether or not the server is stateless.

- soft state
- a stateful server maintains persistent information on its clients.

Servers - General Design Issues

Distinction between session state and permanent state

Server Clusters

General Organization

A collection of machines connected through a network, where each machine runs one or more servers.

In most cases, a server cluster is logically organized into three tiers

Server Clusters

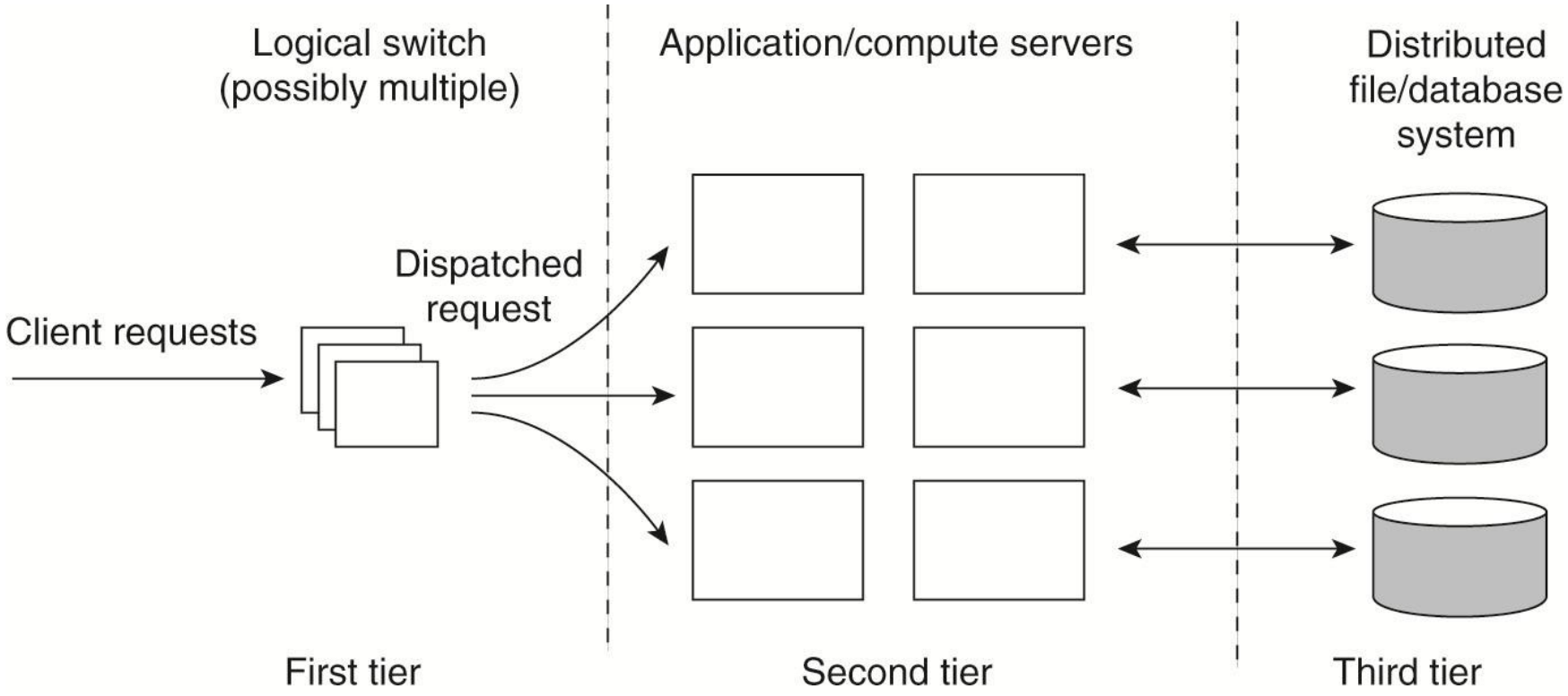

Figure 3-12. The general organization of a three-tiered server cluster.

Server Clusters

When a server cluster offers multiple services, it may happen that different machines run different application servers.

An important design goal for server clusters - hide the fact that there are multiple servers.

Server Clusters

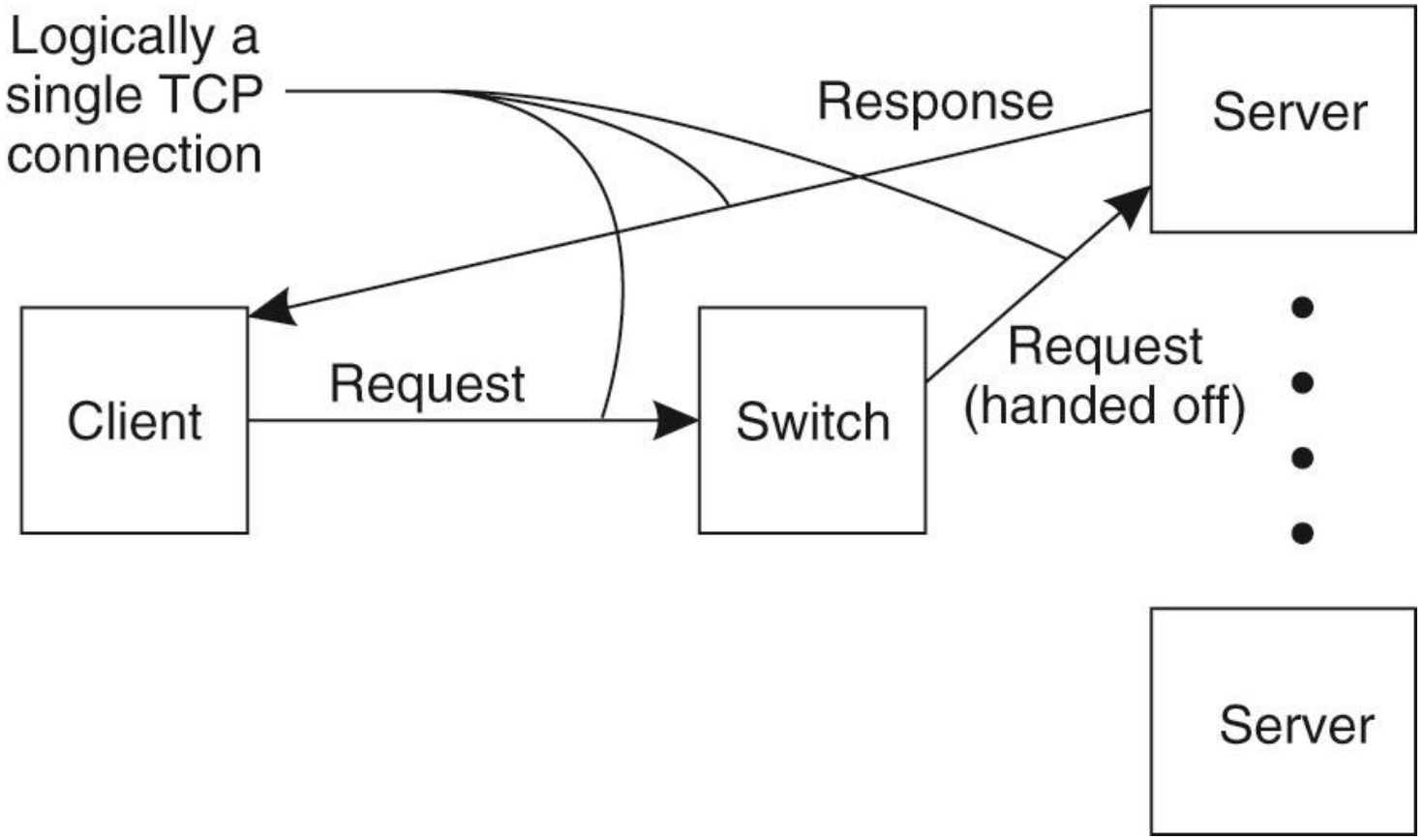

Figure 3-13. The principle of TCP handoff.

Distributed Servers

Most server clusters offer a single access point.

When that point fails, the cluster becomes unavailable.

- several access points can be provided

Distributed Servers

Having a long-living access point, is a desirable feature.

This observation has lead to a design of a distributed server

- dynamically changing set of machines
- with varying access points
- but appears to the outside world as a single powerful machine.

Distributed Servers

How can a stable access point can be achieved in such a system?

Main idea - make use of available networking services
- mobility support for IP version 6 (MIPv6).

MIPv6 - mobile node, home network, home address (HoA), home agent, care-of address (CoA)

This principle can be used to offer a stable address for a distributed server
- a single unique contact address is assigned to the server cluster.

Distributed Servers

This configuration makes the home agent and the access point a potential bottleneck.

This situation can be avoided

- by using an MIPv6 feature known as *route optimization.*

Route optimization can be used to make different clients believe they are communicating with a single server.

Distributed Servers

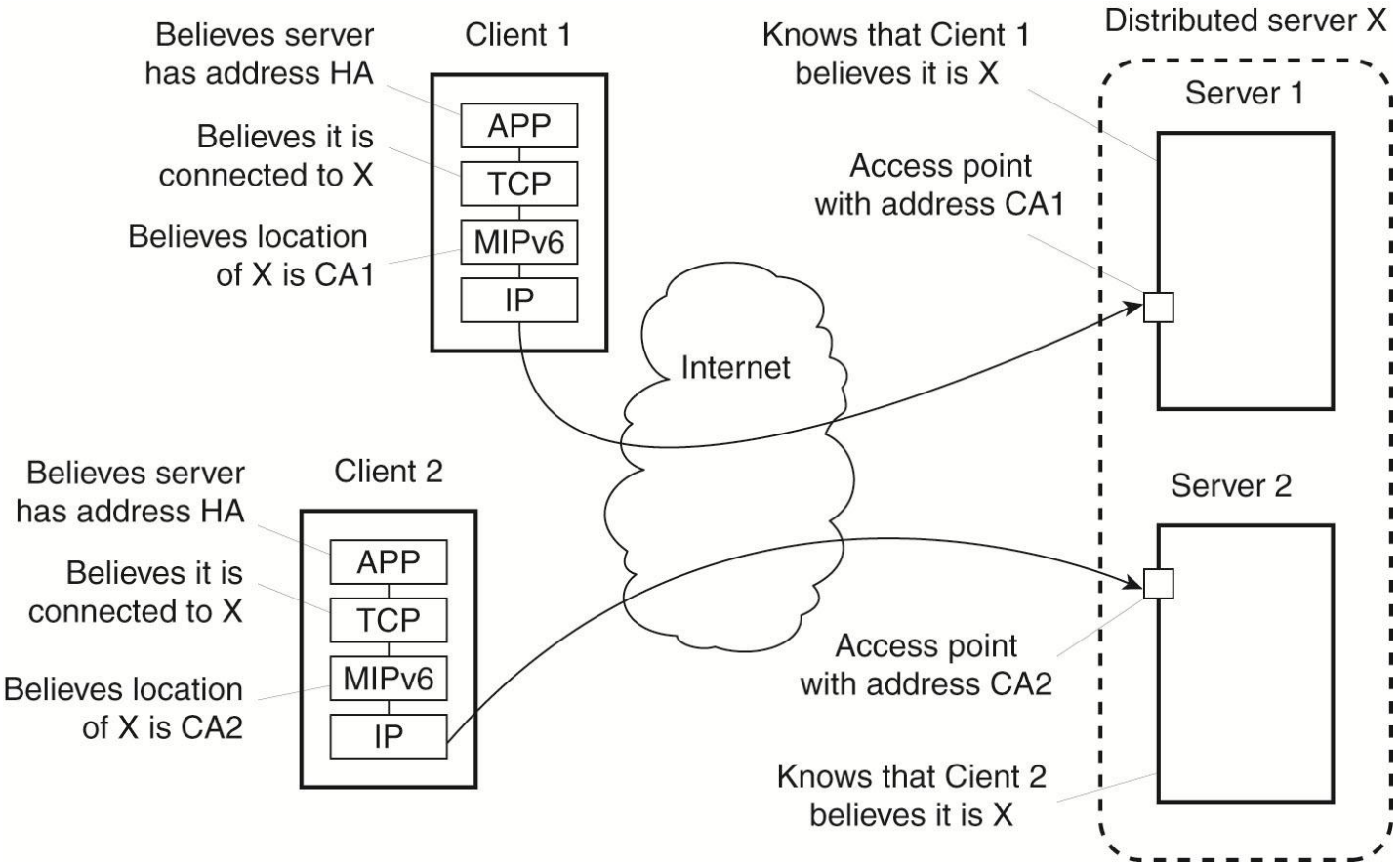

Figure 3-14. Route optimization in a distributed server.

Code Migration

Reasons for Migrating Code

- Load distribution algorithms
- minimize communication (modern distbd systems)
- exploiting parallelism (mobile agent)
- becomes possible to dynamically configure distributed systems

Reasons for Migrating Code

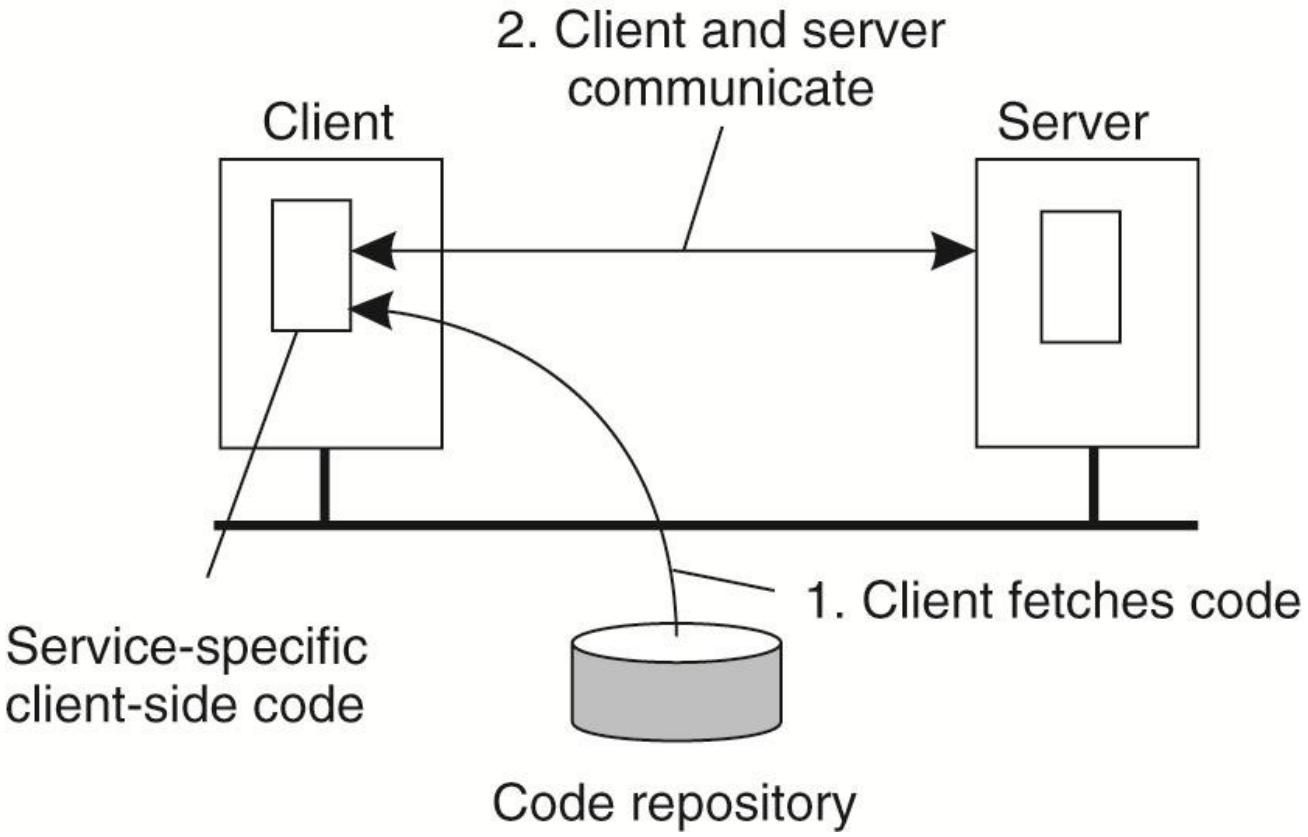

Figure 3-17. The principle of dynamically configuring a client to communicate to a server. The client first fetches the necessary software, and then invokes the server.

Code Migration

The important advantage of dynamically downloading clientside software

- clients need not have all the software preinstalled to talk to servers.

Many disadvantages

- security.

Models for Code Migration

A process consists of three segments:

- code segment
- resource segment
- execution segment

Weak mobility – transfer only code segment

Strong mobility – transfer code segment + execution segment

Models for Code Migration

Sender-initiated migration

Receiver-initiated migration
- simpler

Weak mobility - whether migrated code is executed by the target process, or whether a separate process is started?

Strong mobility can also be supported by remote cloning.

Models for Code Migration

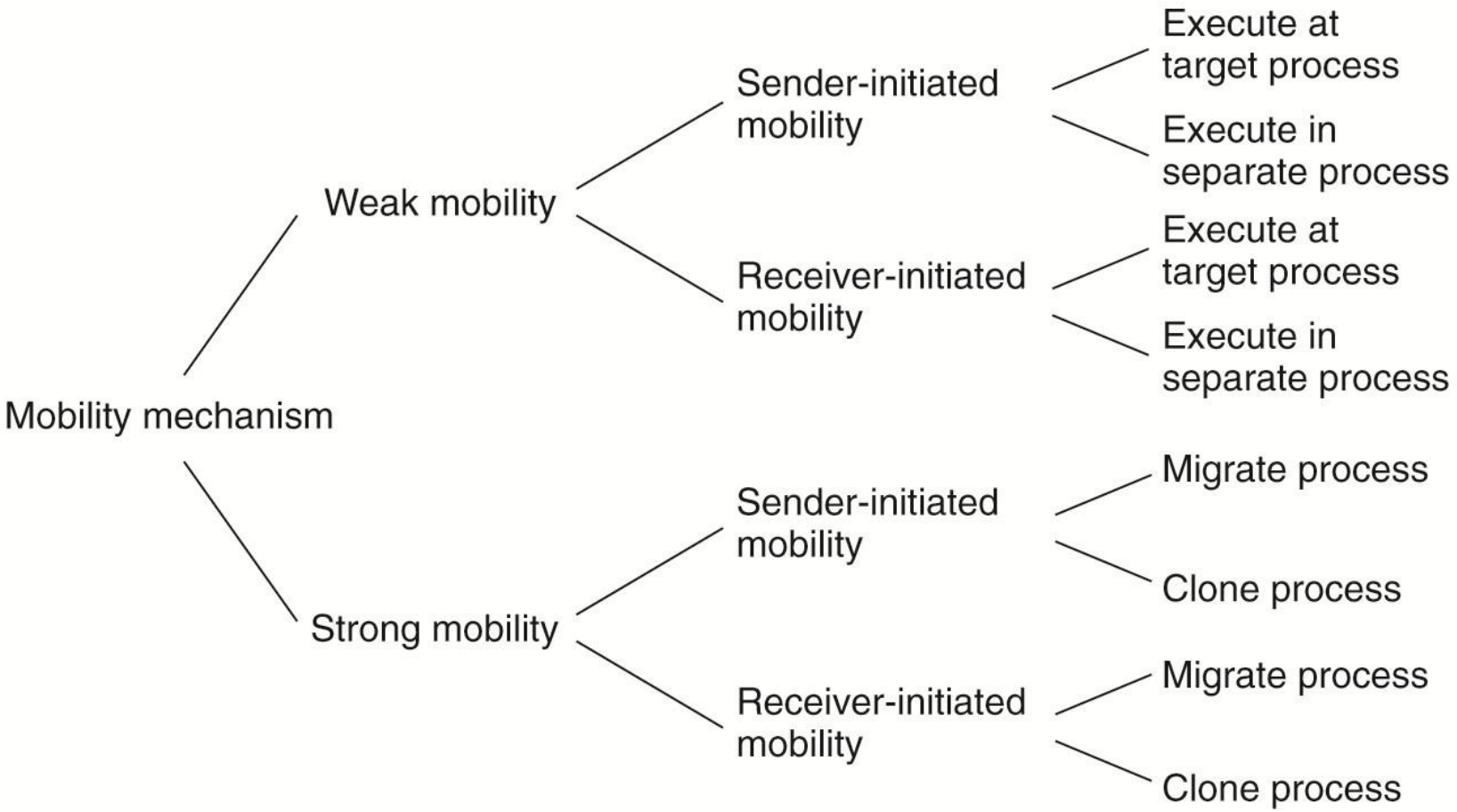

Figure 3-18. Alternatives for code migration.

Migration and Local Resources

Resource segment – cannot always be simply transferred without being changed

Process-to-resource bindings

- binding by identifier (strongest)
- binding by value
- binding by type (weakest)

Migration and Local Resources

When migrating code, we often need to change the references to resources, without affecting the kind of process-to-resource binding.

Resource-to-machine bindings

- unattached resource
- fastened resource
- fixed resource

Migration and Local Resources

Process-to-resource binding	Resource-to-machine binding: Unattached	Fastened	Fixed
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

Figure 3-19. Actions to be taken with respect to the references to local resources when migrating code to another machine.