DISTRIBUTED SYSTEMS
Principles and Paradigms

Second Edition

ANDREW S. TANENBAUM
MAARTEN VAN STEEN

Chapter 3
Processes

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Overview

- Multithreading (for higher performance)
- Virtualization (for portability and failure isolation)
- Clients
- Servers
- Code Migration

Advantages of Threads

- Improves application responsiveness by allowing them to not block
- Allows for parallel computation resulting in higher speed on many-core systems
- Easier to structure many applications as a collection of cooperating threads
- Higher performance compared to multiple processes since switching between threads takes less time

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Thread Usage in Nondistributed Systems

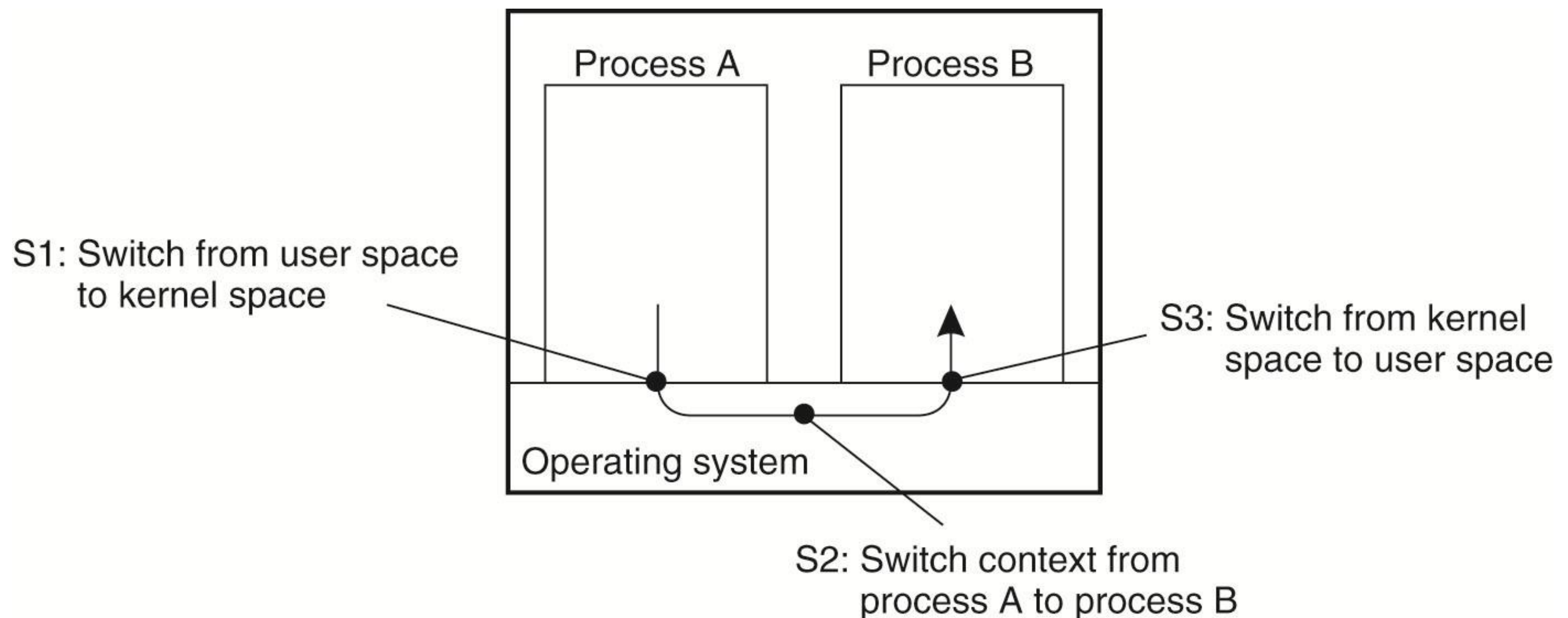


Figure 3-1. Context switching in processes as the result of IPC.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Thread Implementation

- *User-space* threads:
  - Creating/destroying threads is inexpensive
  - Switching context is very fast
  - But invocation of a blocking system call blocks all threads…
  - Cannot make use of multiple cores
- *Kernel-space* threads:
  - Overcomes the last two issues with user-space threads but loses performance
- *Hybrid model*: *Light-Weight Processes* (LWP). Multiple LWP/threads run inside a single (heavy-weight) process. In addition, the system offers a user-level threads package.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Thread Implementation

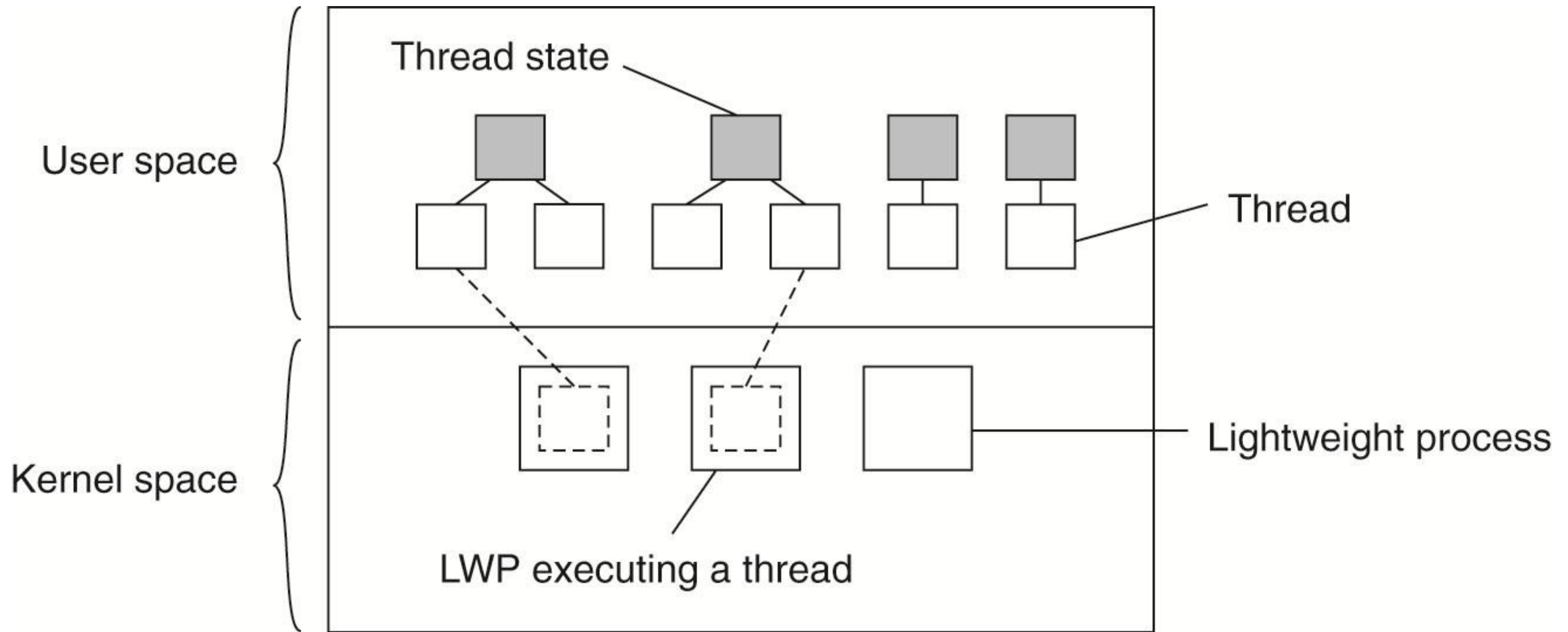


Figure 3-2. Combining kernel-level lightweight processes and user-level threads.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Multithreaded Clients

Multiple threads can be used to hide delays in network communications.

- For example, a web browser can start up several threads, one for downloading the HTML source of the page, one each for images on the page, one each for animations/applets etc
- Replicated web servers along with multi-threaded clients can result in shorter download times.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Multithreaded Servers (1)

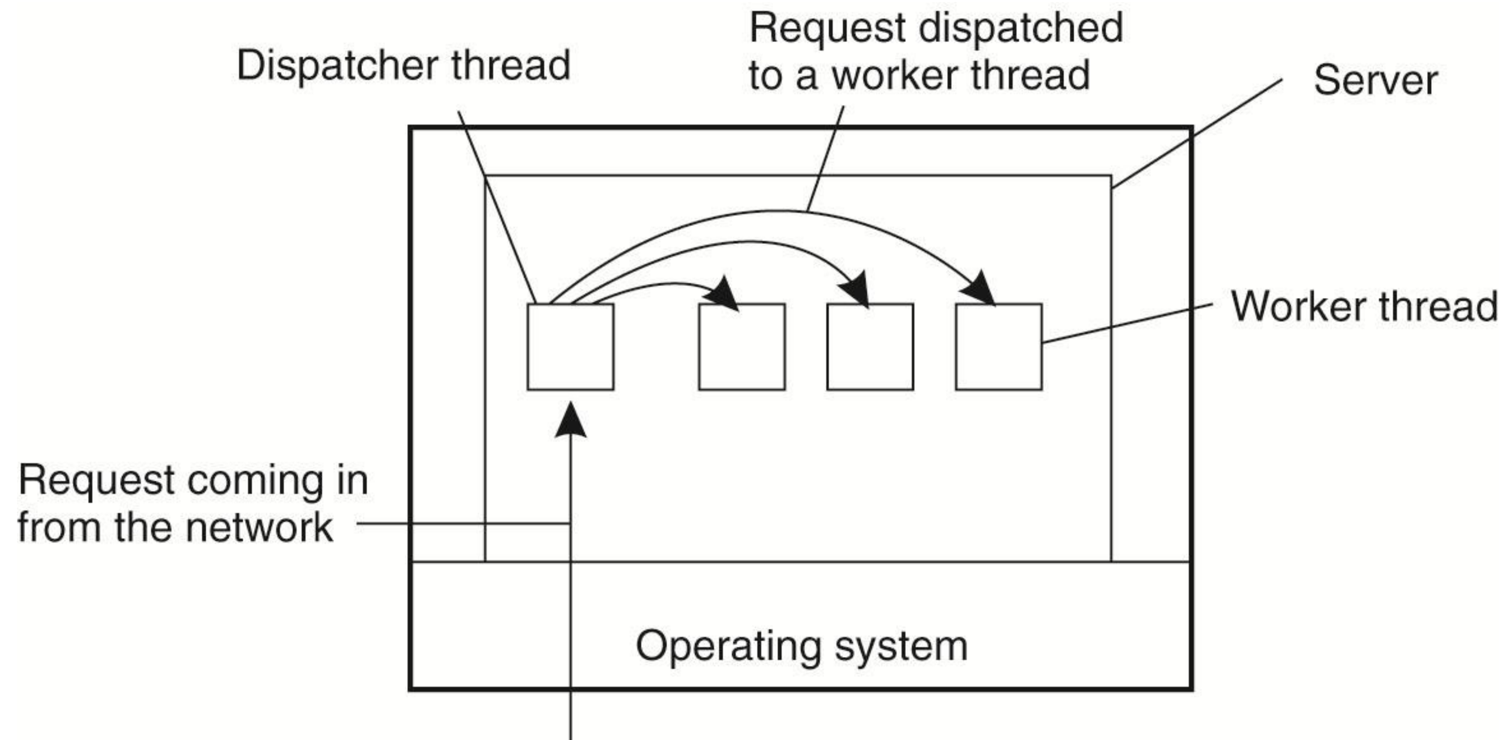


Figure 3-3. A multithreaded server organized in a *dispatcher/worker* model.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Multithreaded Servers (2)

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.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

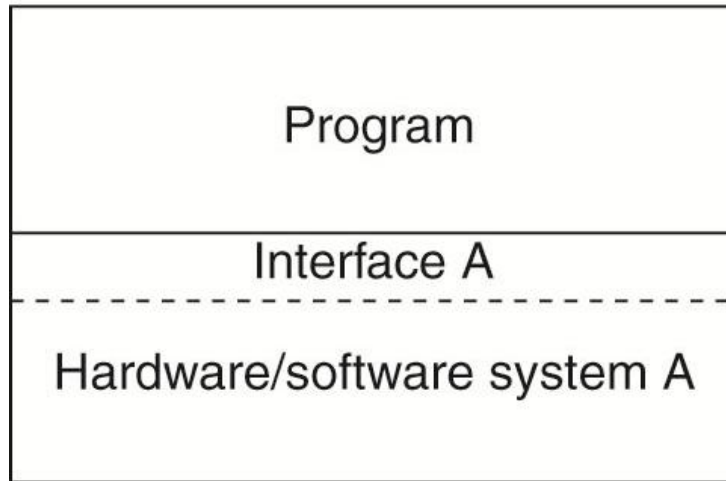
Virtualization

Virtualization is the creation of a virtual (rather than actual) version of something, such as a hardware platform, operating system, a storage device or network resources.

- Eases administration of large number of servers (or resources)
- Helps with scalability and better utilization of hardware resources
- The driver behind cloud computing and utility computing
- Has been around for decades. IBM mainframes have used this technique very successfully for a long time

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

The Role of Virtualization in Distributed Systems



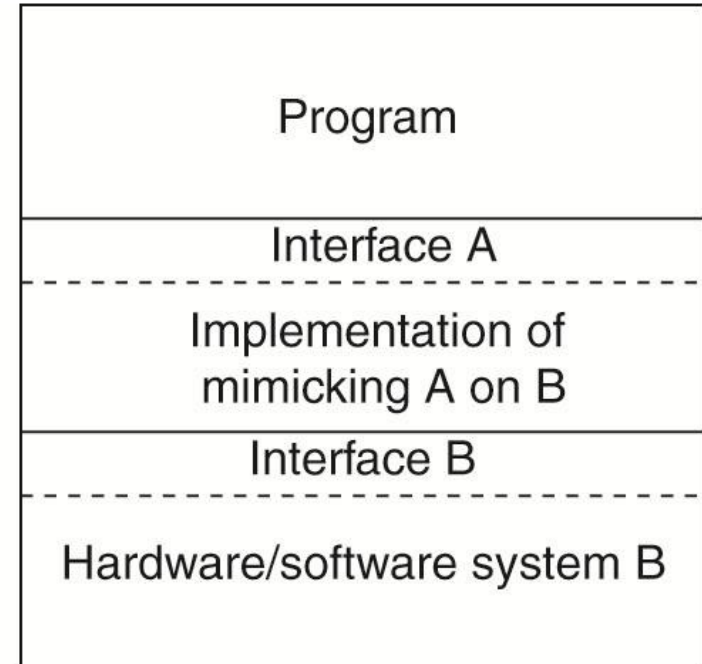


Figure 3-5. (a) General organization between a program, interface, and system. (b) General organization of virtualizing system A on top of system B.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Architectures of Virtual Machines (1)

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.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Architectures of Virtual Machines (2)

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.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Architectures of Virtual Machines (3)

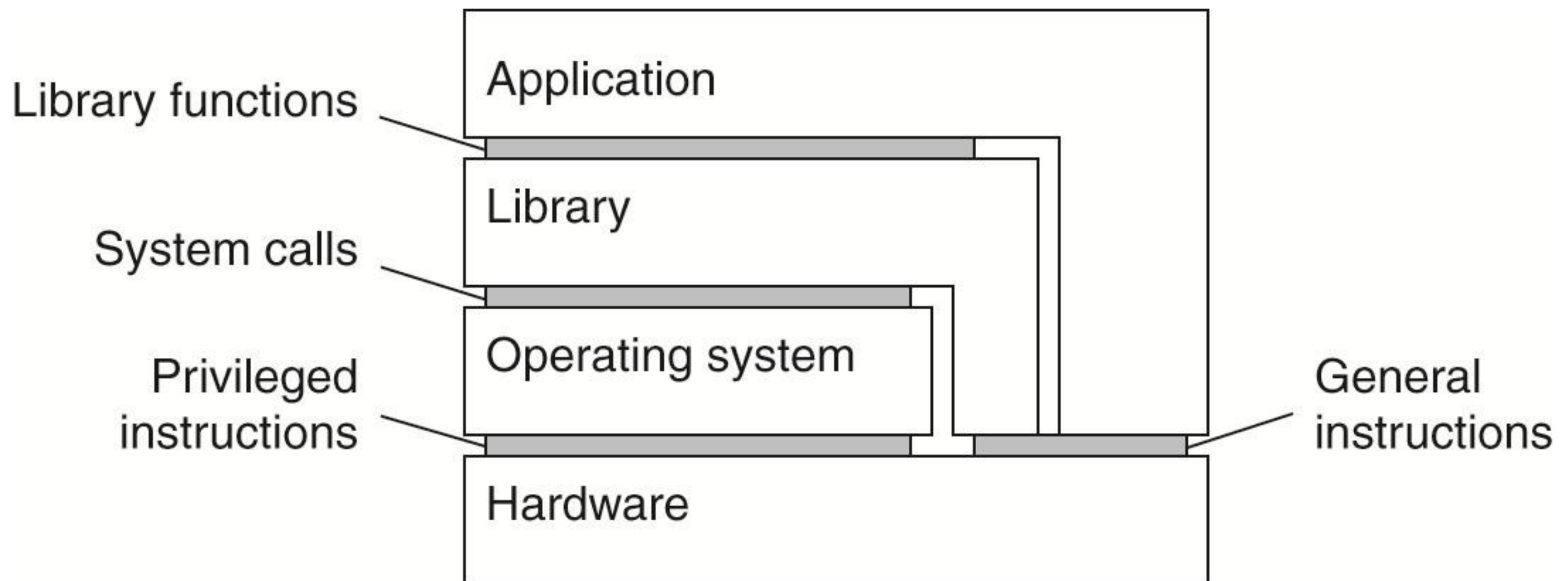


Figure 3-6. Various interfaces offered by computer systems.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Architectures of Virtual Machines (4)

Virtualization can be implemented at two levels.

- *Process Virtual Machine*: An abstract instruction set that is to be used for executing applications. For example: Java runtime, Windows emulation (Wine) on Unix/Linux/MacOS.
- *Virtual Machine Monitor*: A layer completely shielding the original hardware but offering the complete instruction set of that same (or other hardware) as an interface. Makes it possible to have multiple instances of different operating systems run simultabeously on the same platform. Examples: Vmware, VirtualBox, Xen, VirtualPC, Parallels etc

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Architectures of Virtual Machines (5)

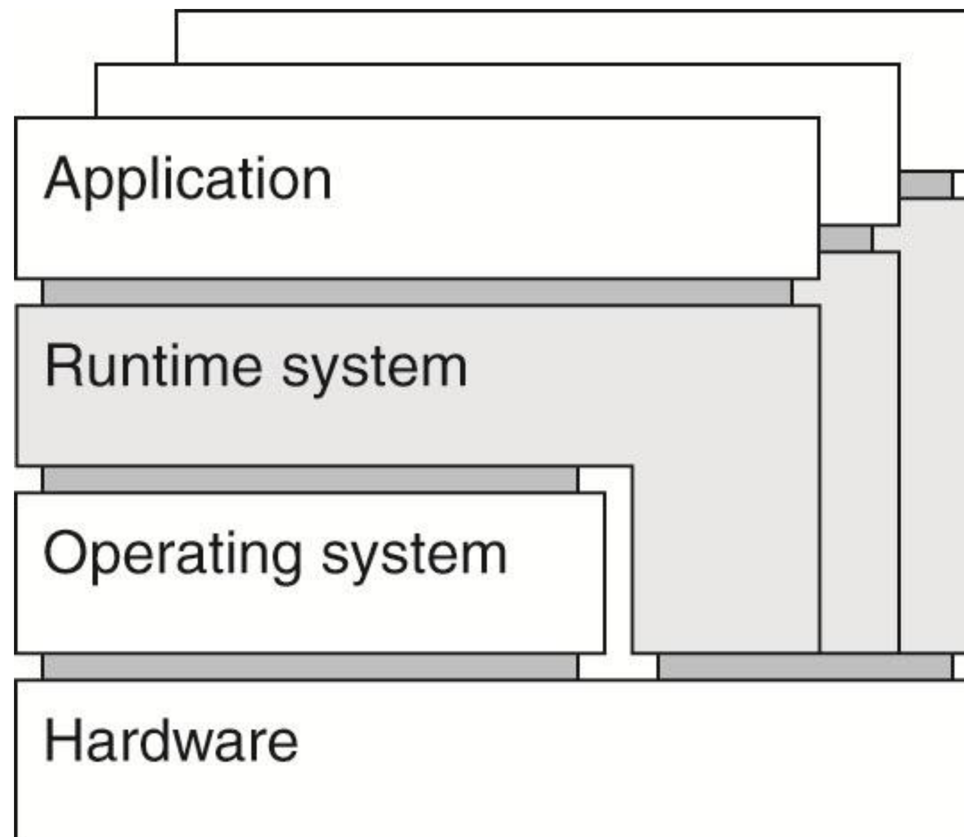


Figure 3-7. (a) A process virtual machine, with multiple instances of (application, runtime) combinations.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Architectures of Virtual Machines (6)

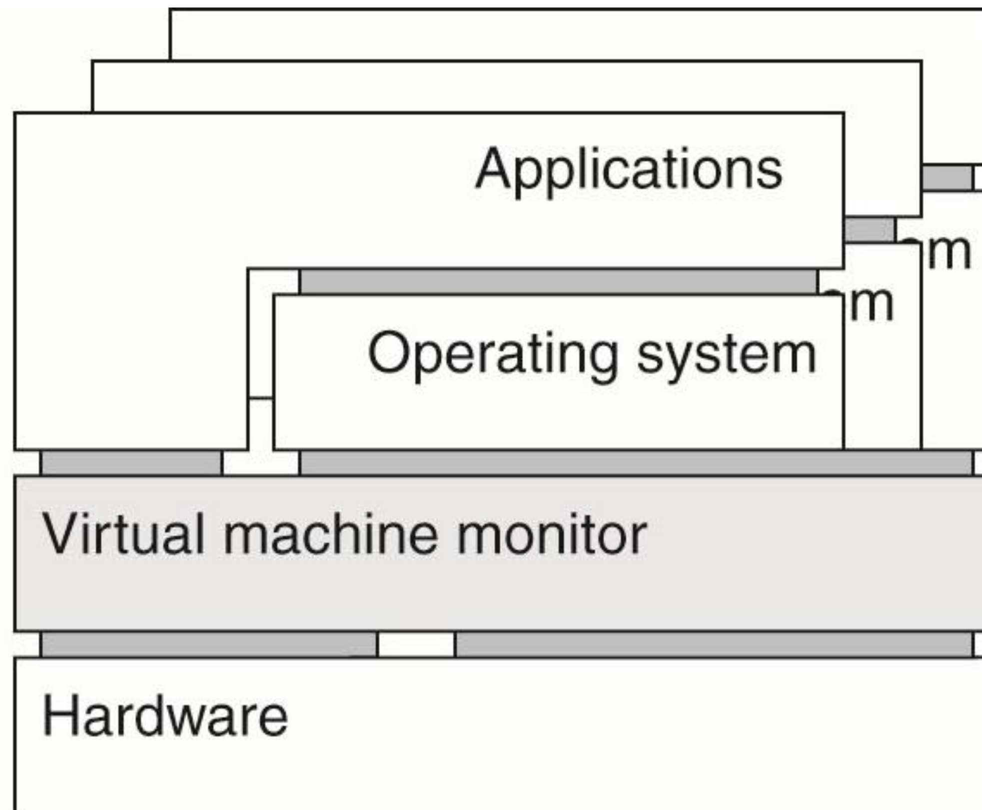


Figure 3-7. (b) A virtual machine monitor, with multiple instances of (applications, operating system) combinations.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Design Issues for Clients

- For each remote service the client machine can 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. The client machine is used only as a terminal with no need for local storage, leading to an application neutral solution. (*Thin-client* approach)
- **Transparency for clients**: Access (via stubs/interfaces), location, migration, relocation, replication, failure, concurrency (intermediate layer like a transaction monitor), persistence (server-side)

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Networked User Interfaces (1)

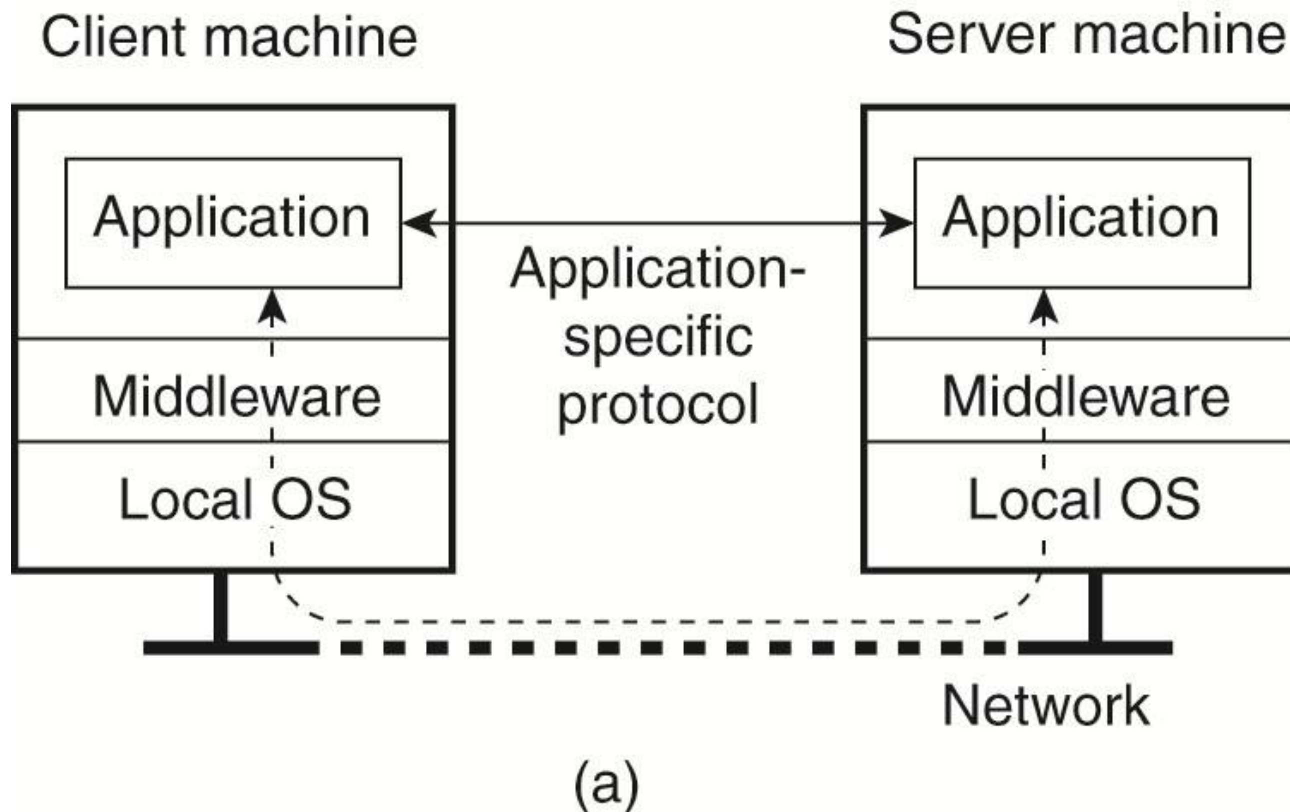


Figure 3-8. (a) A networked application with its own protocol.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Networked User Interfaces (2)

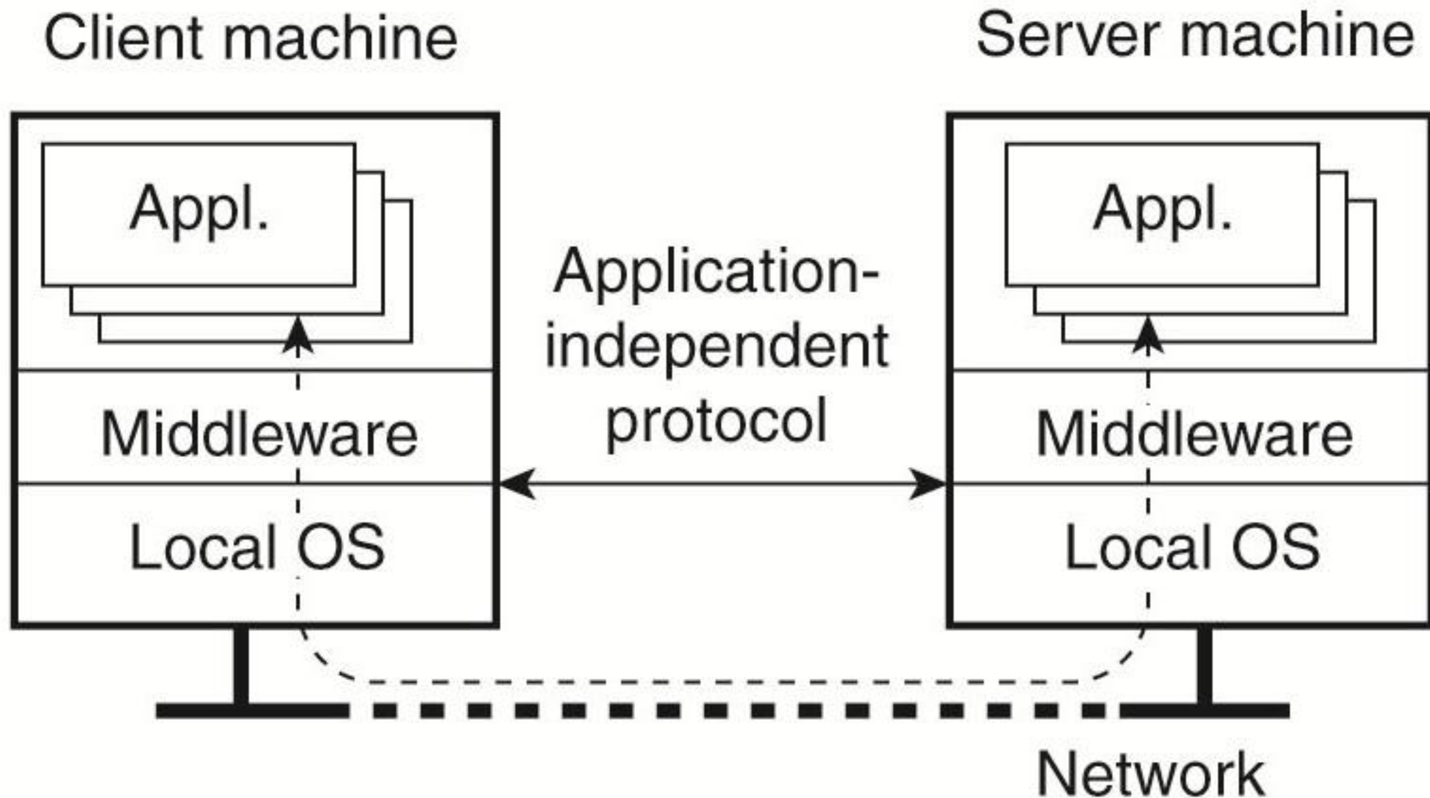


Figure 3-8. (b) A general solution to allow access to remote applications.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Example: The XWindow System

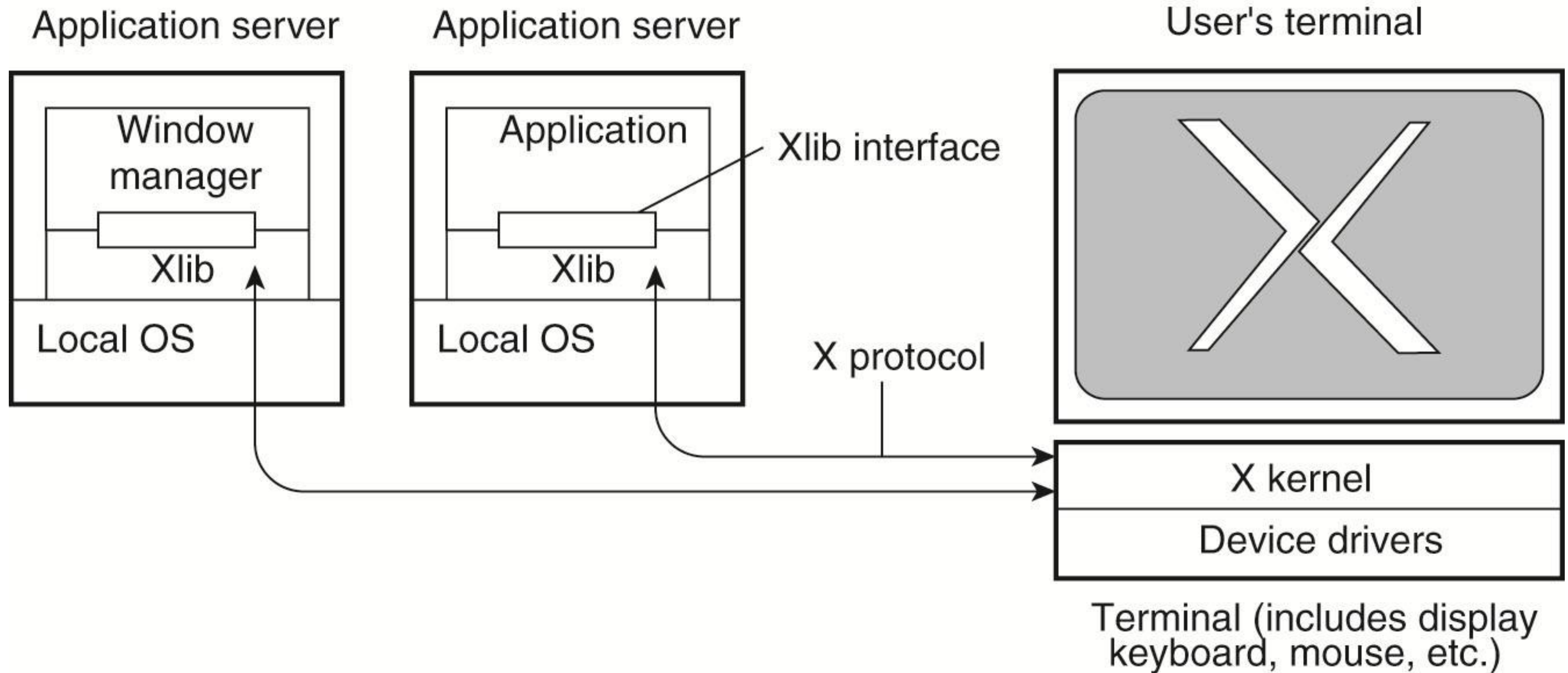


Figure 3-9. The basic organization of the XWindow System.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Client-Side Software for Distribution Transparency

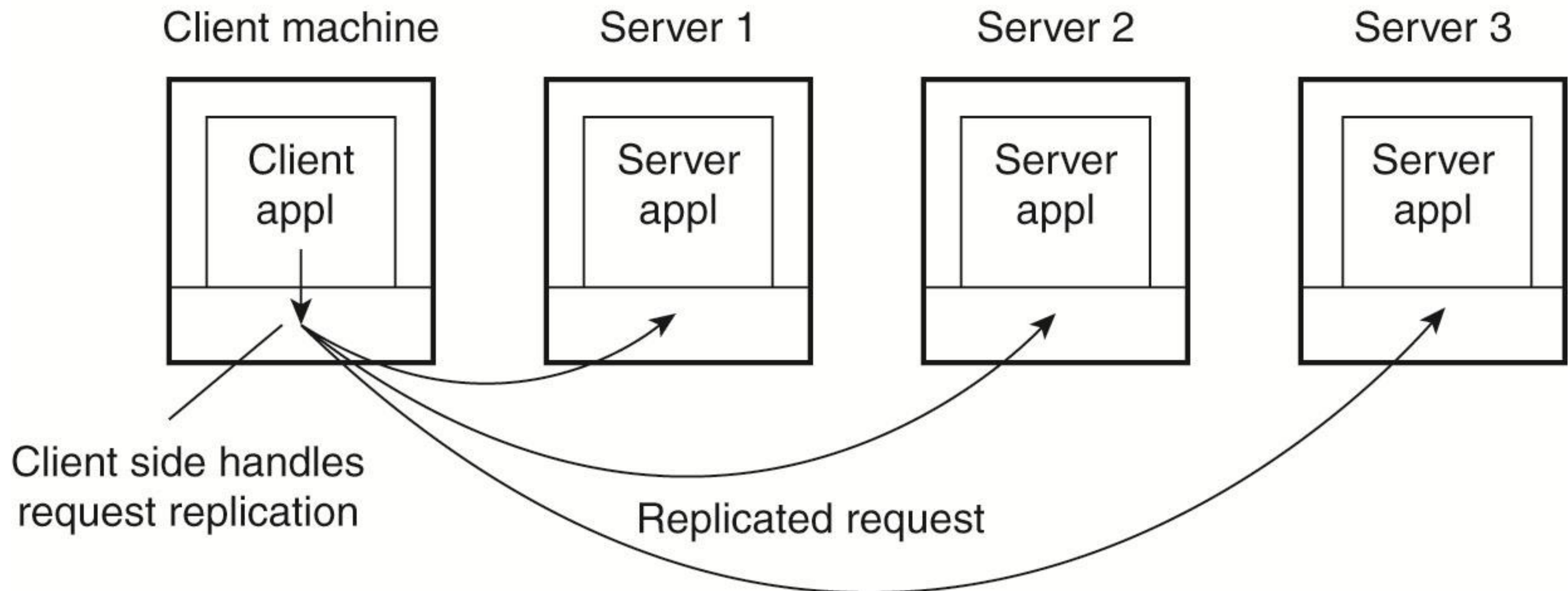


Figure 3-10. Transparent replication of a server using a client-side solution.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Design Issues for Servers

- A server is a process implementing a specific service on behalf of a collection of clients
- A server can be *iterative* or *concurrent*. Concurrent servers can be multi-threaded or multi-process
- How does a client find a server? Need to know the *end-point* or *port* and the host address
  - Statically assigned like well known servers like HTTP on port 80
  - Look-up service provided by a special *directory* server
  - Using a *superserver* that selects on multiple ports and forks off the appropriate server when a request comes in

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

General Design Issues (1)

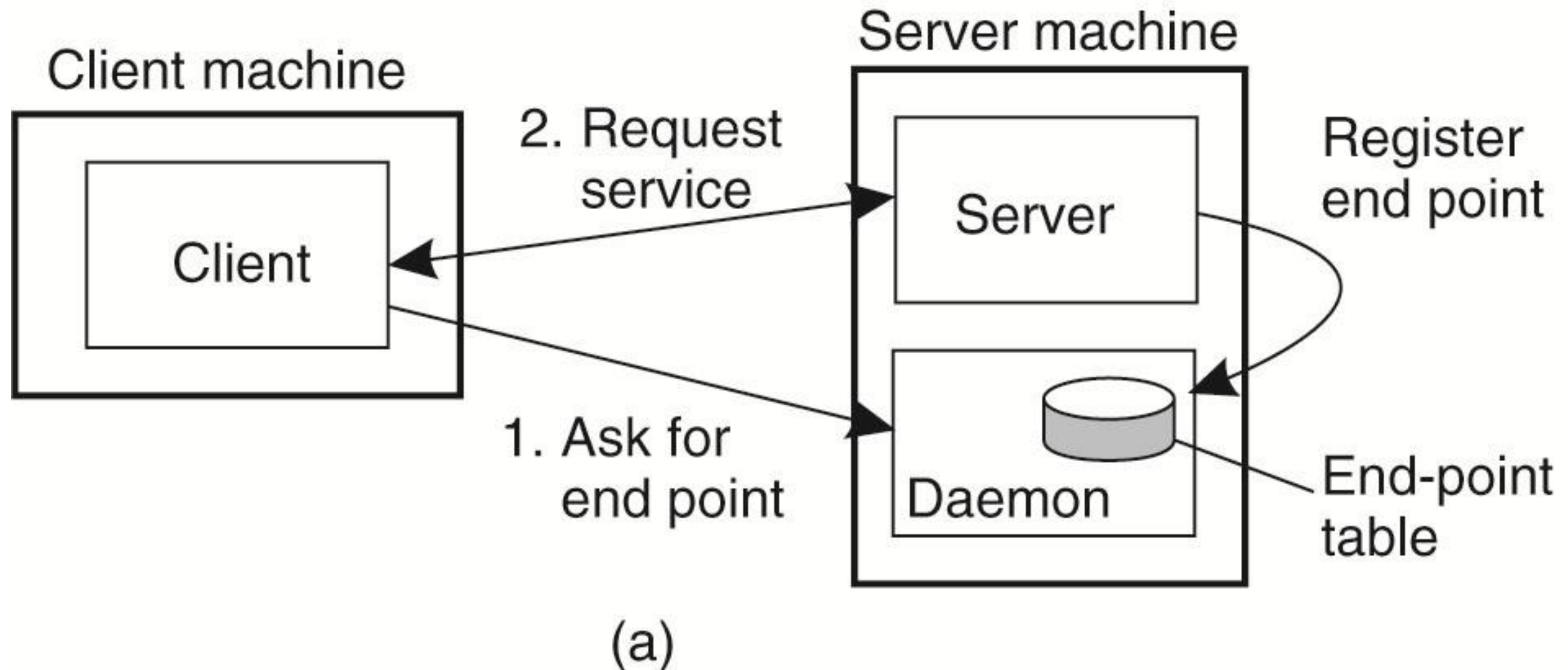


Figure 3-11. (a) Client-to-server binding using a daemon.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

General Design Issues (2)

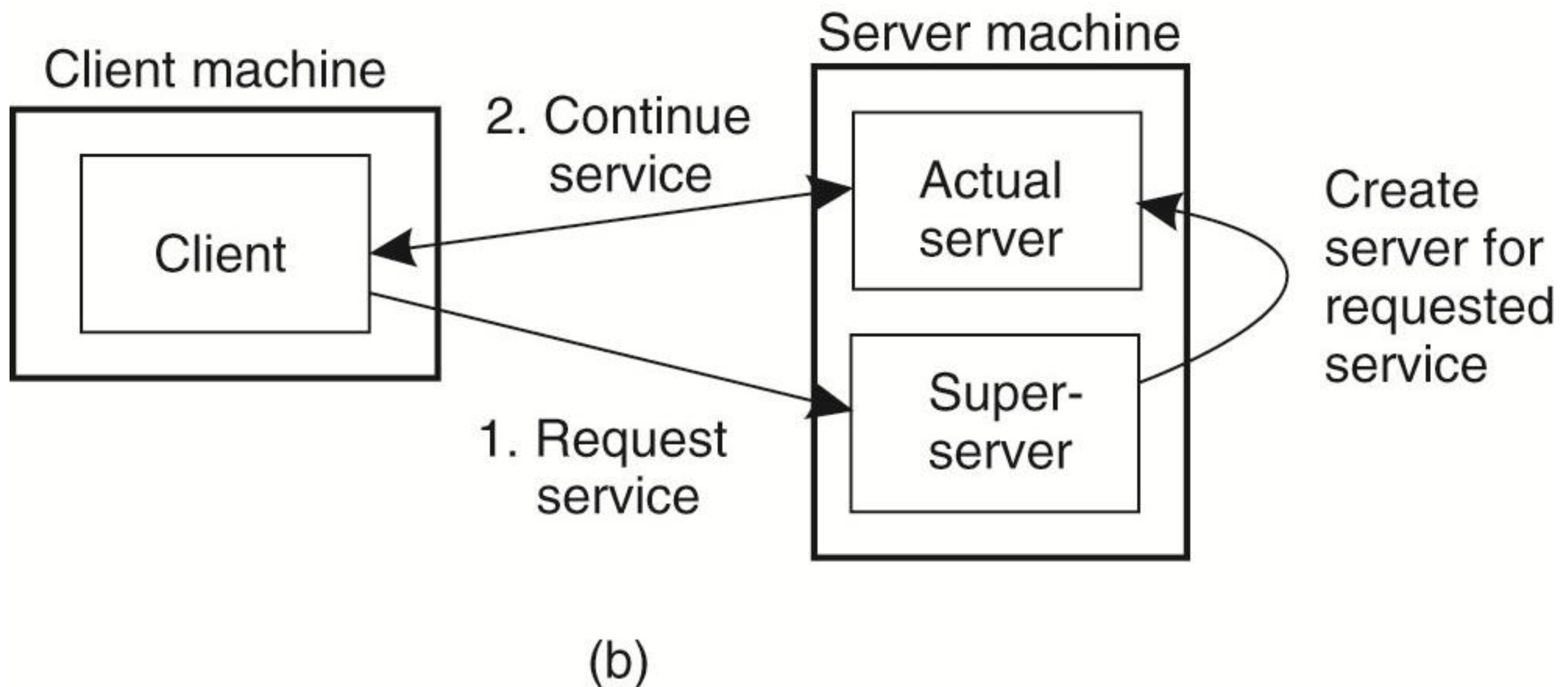


Figure 3-11. (b) Client-to-server binding using a superserver.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Servers: Further Design Issues

- How to handle communication interrupts? Use *out-of-band data*. Example: to cancel the upload of a huge file
  - Server listens to separate endpoint, which has higher priority, while also listening to the normal endpoint (with lower priority)
  - Send urgent data on the same connection. Can be done with TCP, where the server gets a signal (SIGURG) on receiving urgent data

- ***Stateless*** servers. A stateless server does not remember anything from one request to another. For example, a HTTP server is stateless. *Cookies* can be used to transmit information specific to a client with a stateless server. Easy to recover from a crash.
- ***Stateful*** servers. Maintains information about its clients. Performance improvement over stateless servers is often the reason for stateful servers. Needs to recover its entire state as it was just before crash. Can be quite complex for distributed servers.
- ***Soft state***: The server promises to maintain state on behalf of the client, but only for a limited time.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Server Clusters (1)

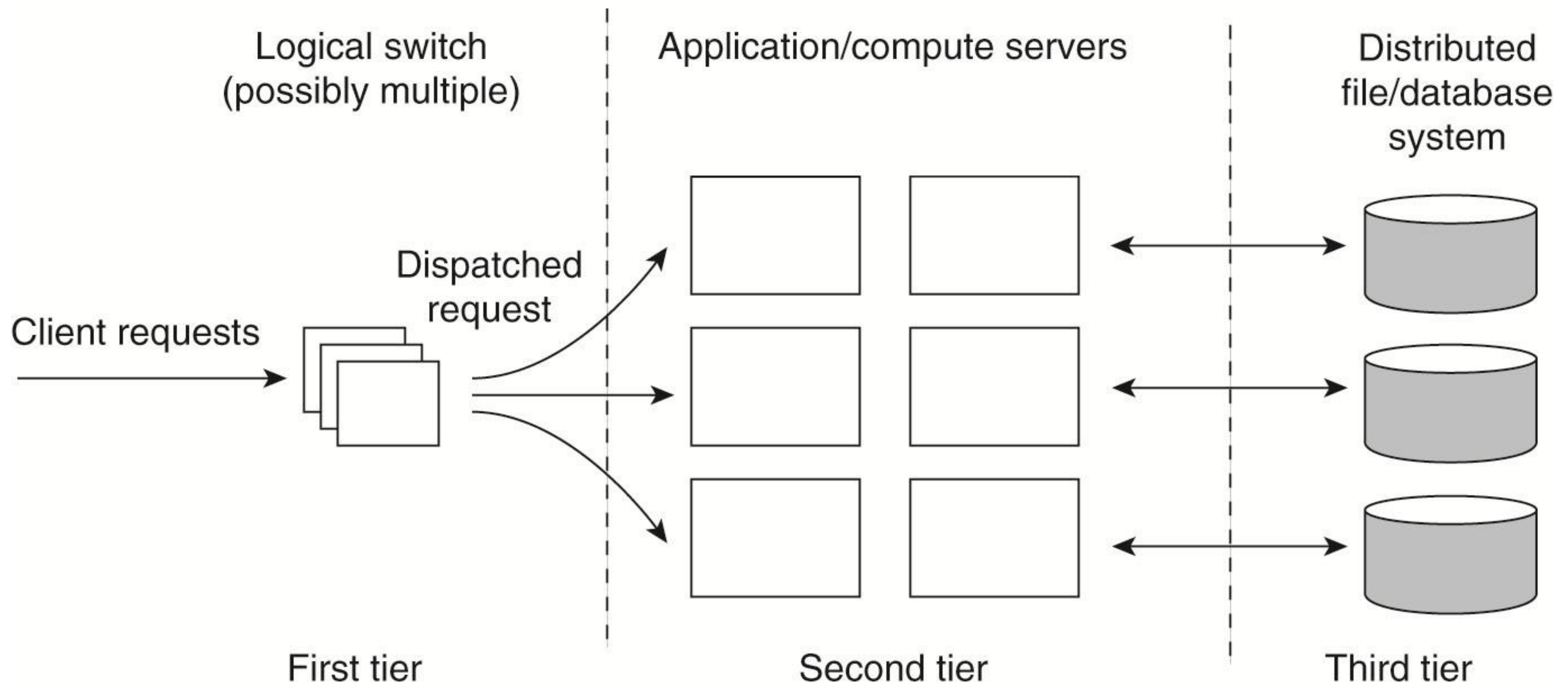


Figure 3-12. The general organization of a three-tiered server cluster.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Server Clusters (2)

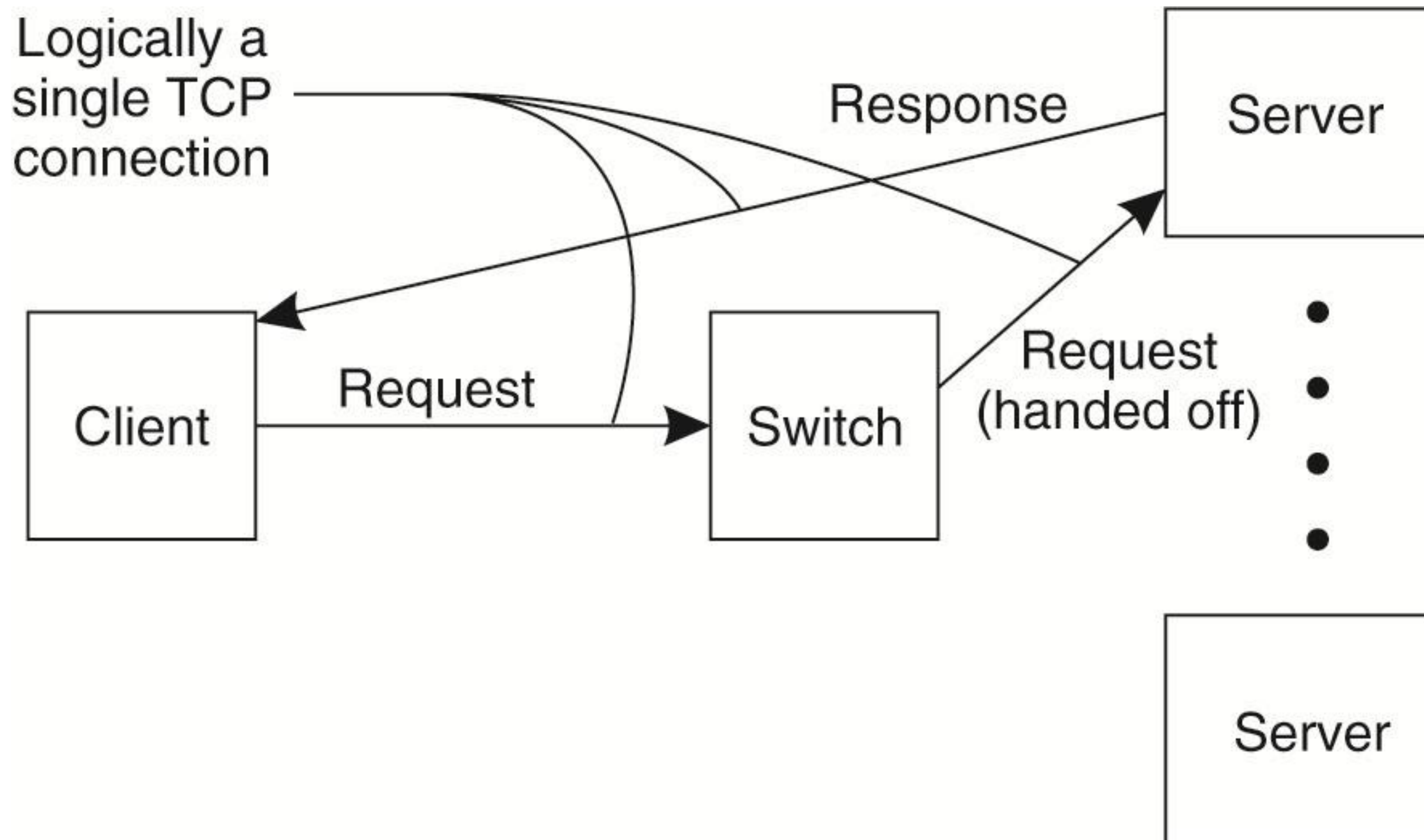


Figure 3-13. The principle of TCP handoff (requires IP forwarding and IP spoofing).

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Server Clusters (3)

- Switch can hand off connections in round-robin and thus be oblivious of the service being provided
- Switch can handoff request based on type of service requested to the appropriate server
- Switch can handoff request based on server loads
- Switch can handoff request by being aware of the content

- Single point of access can be made better using DNS to map one hostname to several servers. But the client still has to try multiple servers in case some are down.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Distributed Servers

- A *distributed server* is a possibly dynamically changing set of machines, with also possibly varying access points, but which nevertheless appears to the outside world as a single, powerful machine
- A *stable access point* across a distributed server can be implemented using mobility support for IPv6 (MIPv6)
  - *home network*: where a mobile node normally resides
  - *home address* (*HoA*): stable address for a node in its home network
  - *care of address* (*CoA*): when a mobile node attaches to a foreign network, its sets up a forwarding from its HoA to the CoA
  - *route optimization* from MPIv6 is used to make different clients believe they are communicating with a single server where, in fact, each is communicating with a different member node of the distributed server

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Distributed Servers

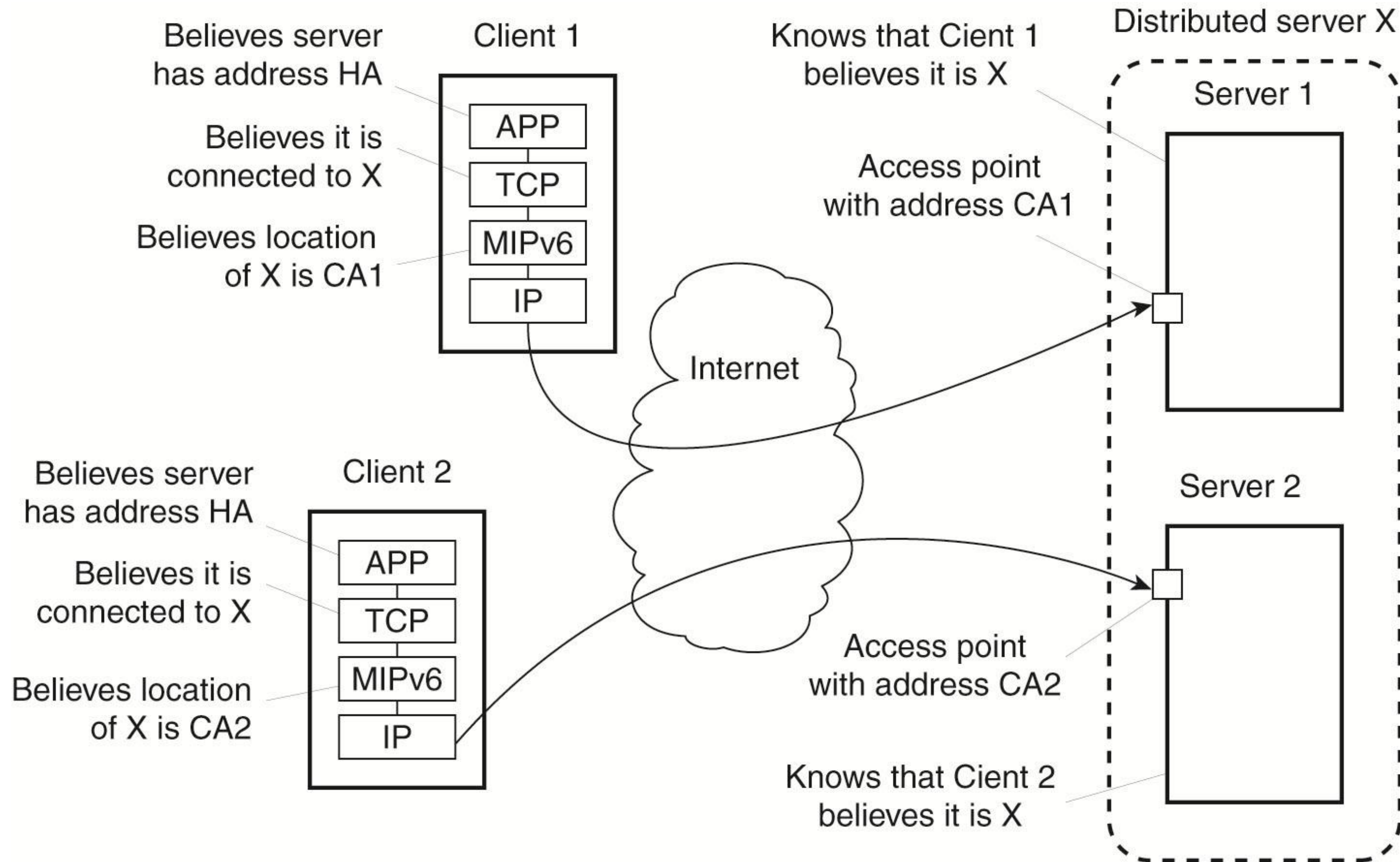


Figure 3-14. Route optimization in a distributed server.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Managing Server Clusters

- An administrator manually logs in to each server to manage them. Or they use shell scripts or simple tools to manage a collection of servers
- An management interface is provided at one system to collectively manage the cluster of servers
- However to go to the level of thousands or more of servers, most administration approaches are ad hoc and still an active area of research
- Large server cluster farms are increasing rapidly! Google is said to operate over a million servers in 12 farms

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Managing Server Clusters

Example: PlanetLab

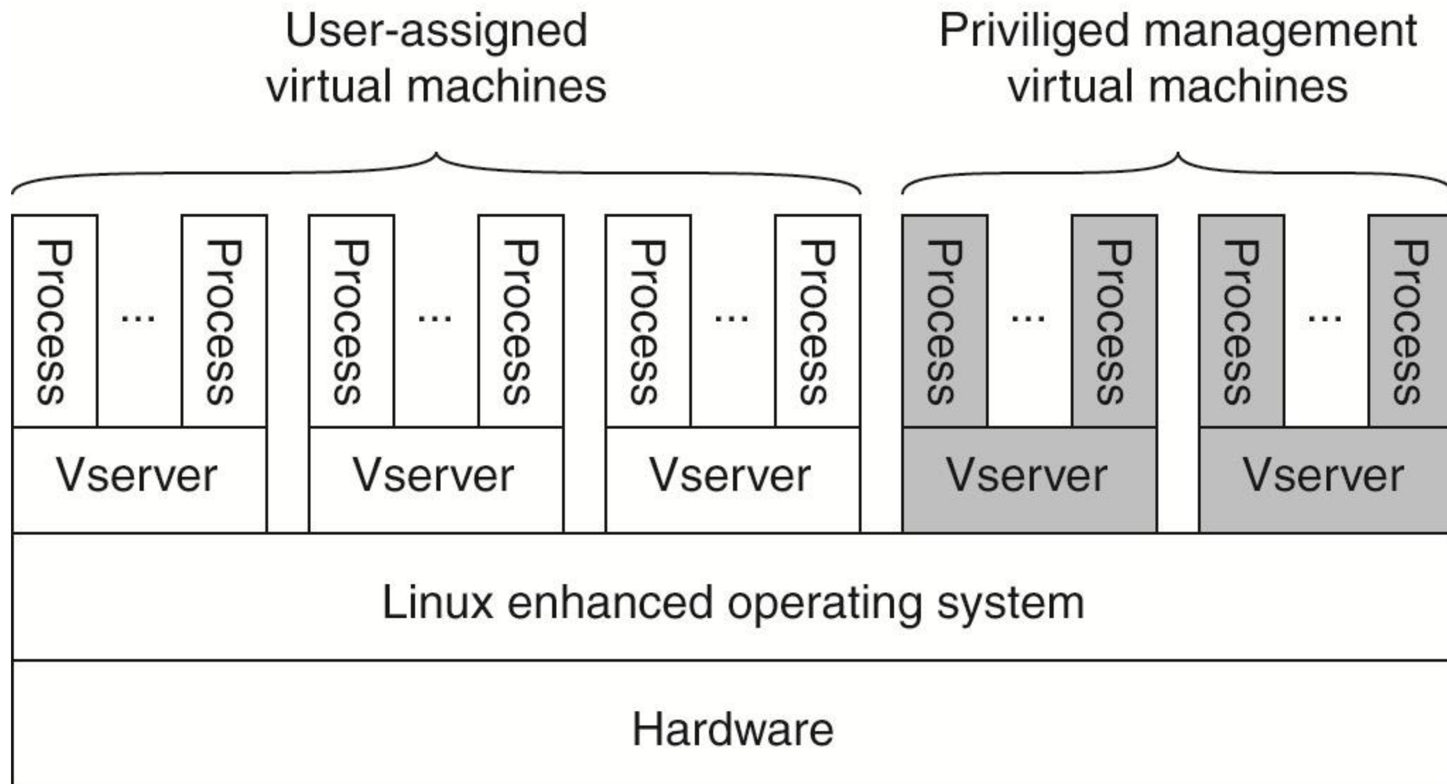


Figure 3-15. The basic organization of a PlanetLab node.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

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.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

PlanetLab (2)

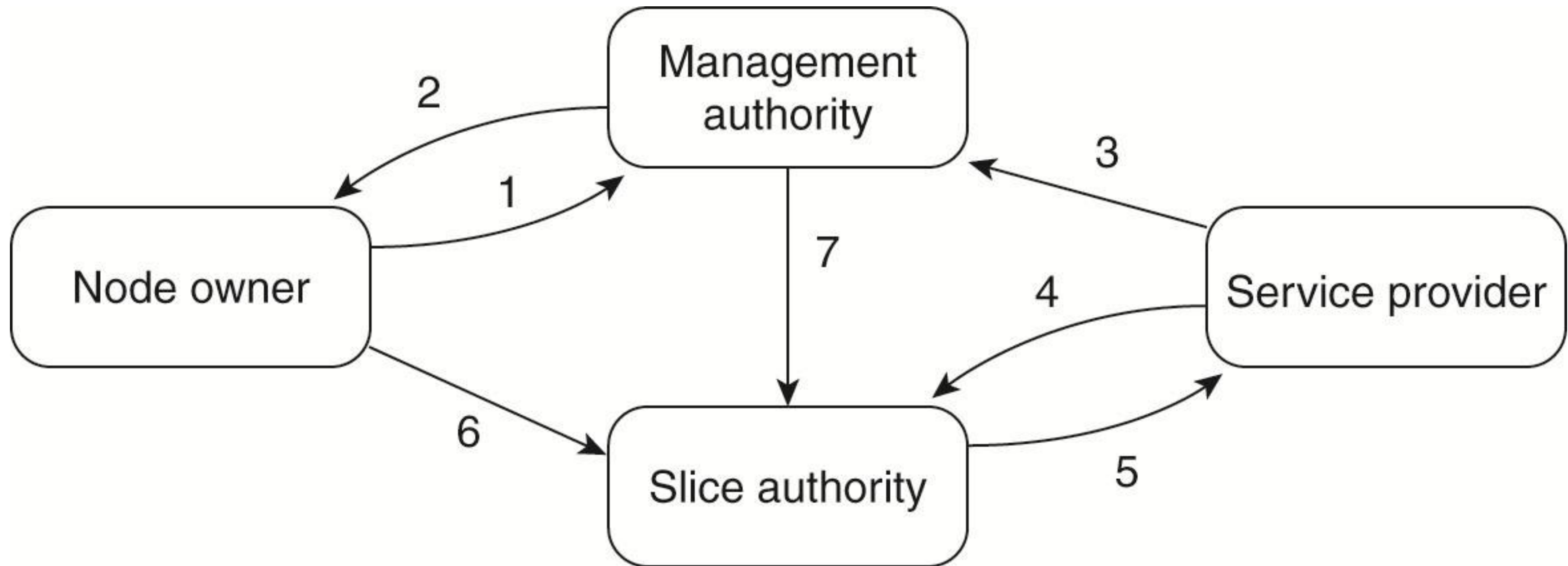


Figure 3-16. The management relationships between various PlanetLab entities.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

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

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Reasons for Migrating Code

- Improve computing performance by moving processes from heavily-loaded machines to lightly loaded machines.
- Improve communication times by shipping code to systems where large data sets reside. E.g. a client ships code to a database server or vice versa.
- *Mobile Agents*: small piece of code that moves from site to site for a web search. Several copies can be made to improve performance.
- Flexibility to dynamically configure distributed systems. E.g. a server can provide interface code to a client dynamically. This does the require that the protocol for downloading and initializing the code is standardized. Allows the interface to be changed as often as desired without having to rebuild applications or servers.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Reasons for Migrating Code

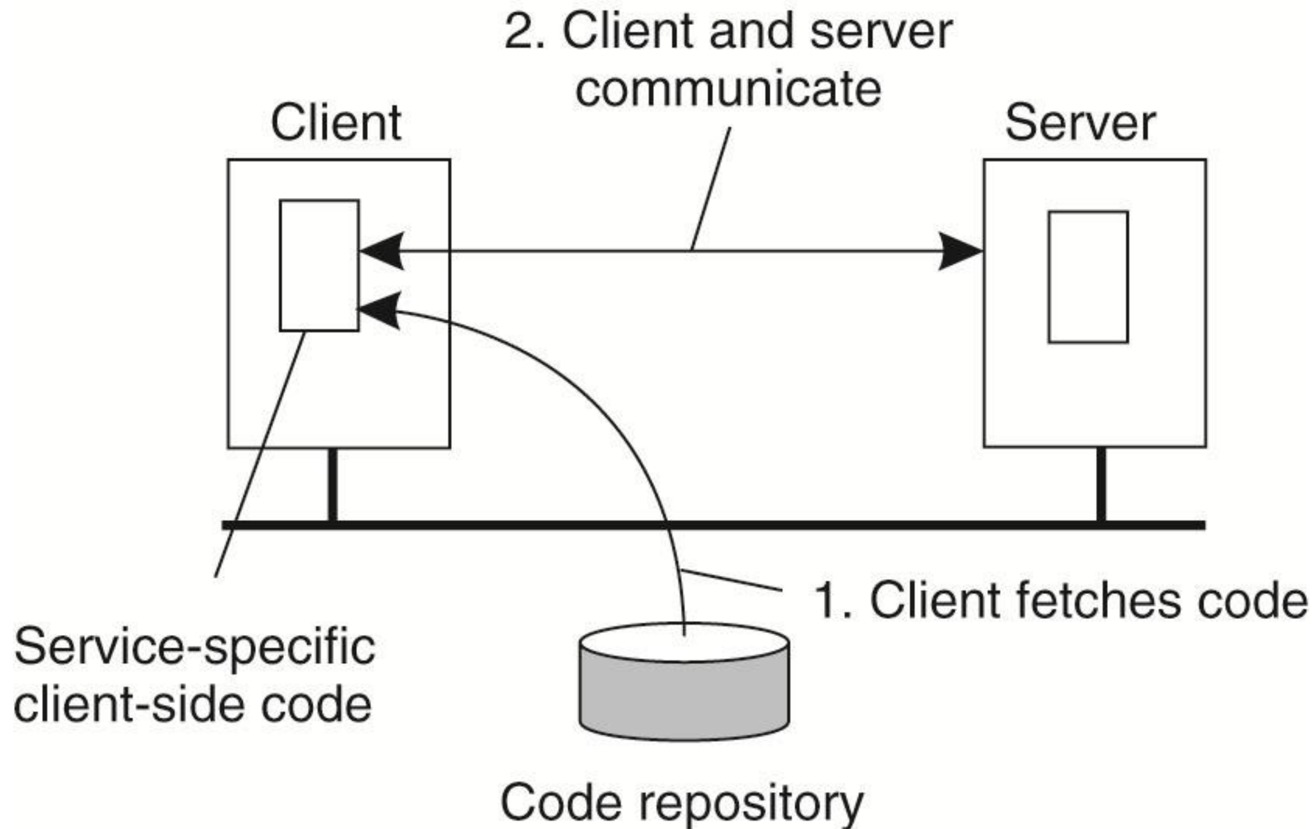


Figure 3-17. The principle of dynamically configuring a client to communicate to a server. The client first fetches the necessary software, and then invokes the server.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Code Migration Models

- A process consists of three segments: *code segment*, *resource segment*, *execution segment*.
- *Weak mobility*: Only the code segment (and some initialization data) can be transferred. Transferred program always starts at one of the predefined starting positions. E.g. java applets.
- *Strong mobility*: Code and execution segments can both be transferred.
- *Sender-initiated* versus *receiver-initiated*: Uploading code to a server versus downloading code from a server by a client.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Models for Code Migration

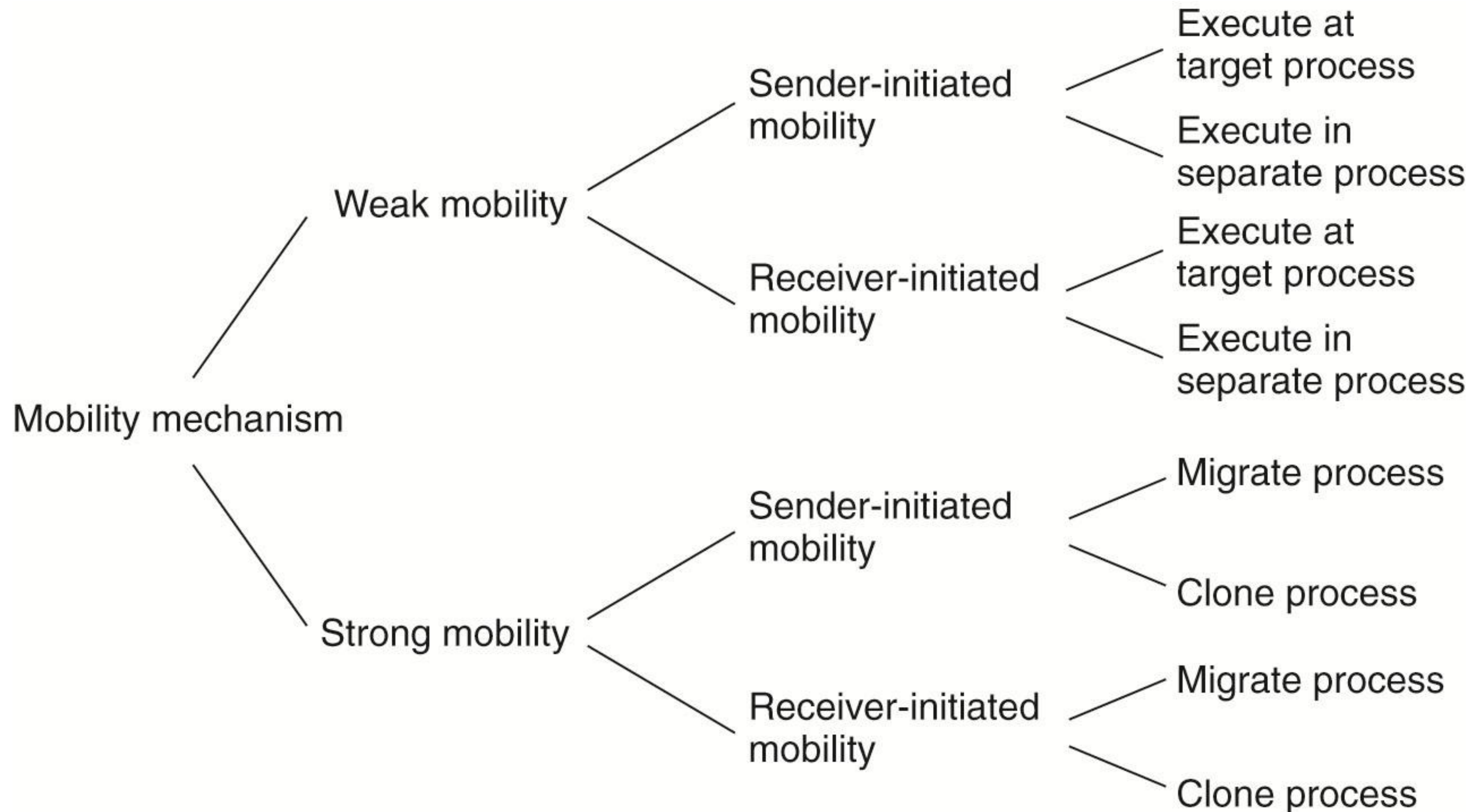


Figure 3-18. Alternatives for code migration.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Migration and Local Resources

		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

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

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Migration in Heterogeneous Systems

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.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5