CSE 5306 Distributed Systems

Processes

Jia Rao

http://ranger.uta.edu/~jrao/

Processes in Distributed Systems

- In traditional OS, management and scheduling of processes are the main issues.
  - ✓ Sharing the CPU, memory, I/O and other resources
- In distributed systems, other aspects needed to be considered:
  - ✓ Multi-threading for efficiency
  - ✓ Virtualization for isolation and elasticity
  - ✓ Process migration (in traditional OS and distributed systems)

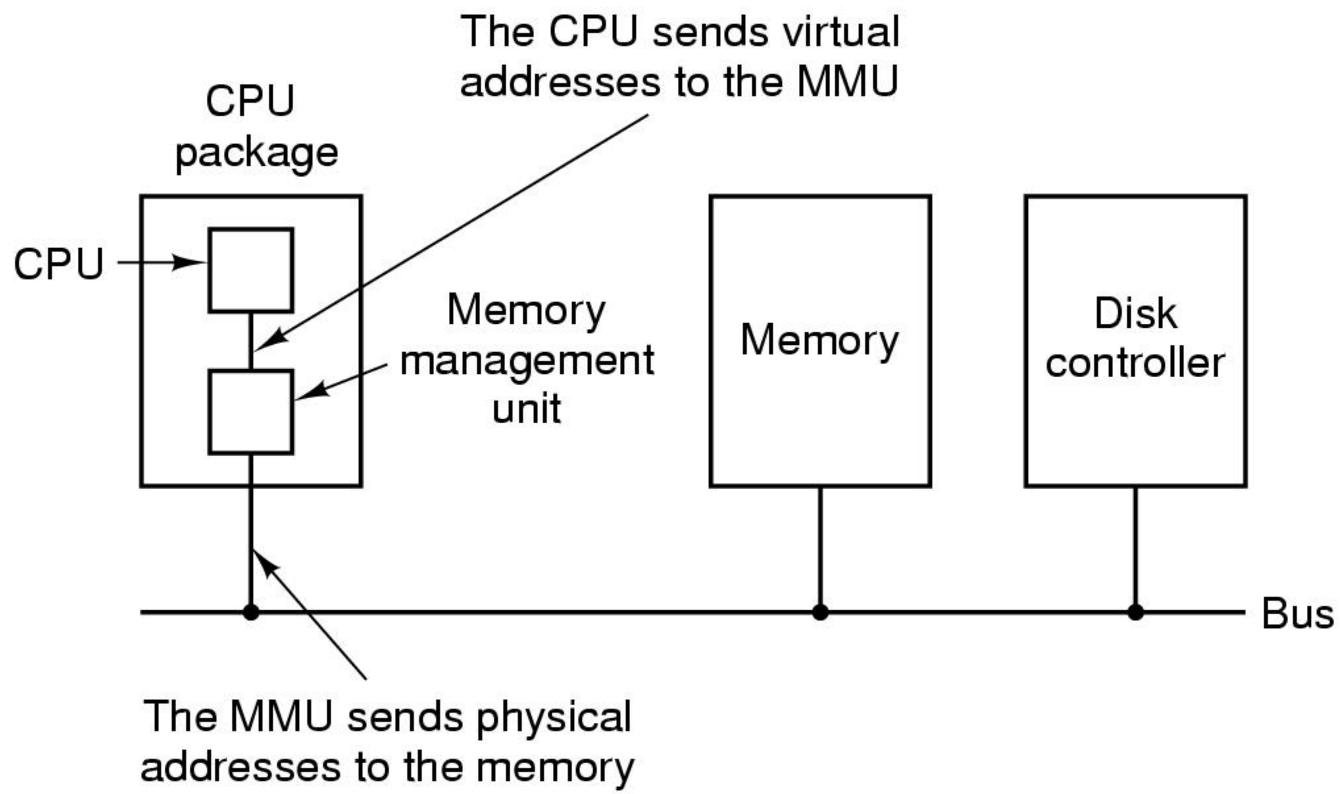
Multi-threaded Process

- Problems with process
  - ✓ Creating a new process is expensive
  - ✓ Context switch between processes is also expensive
- Benefits of multi-threaded processes
  - ✓ Blocking system call does not stop a process
  - ✓ Exploit the parallelism in multiprocessor system
  - ✓ Useful in cooperating programs: different parts of an application need to talk to each other (pipes, message queues, and shared memory segments)
  - ✓ Easier to develop a program using a collection of threads

Virtual Memory

Virtual memory: the combined size of the program, data, and stack may exceed the amount of physical memory available

Mapping of Virtual addresses to Physical addresses

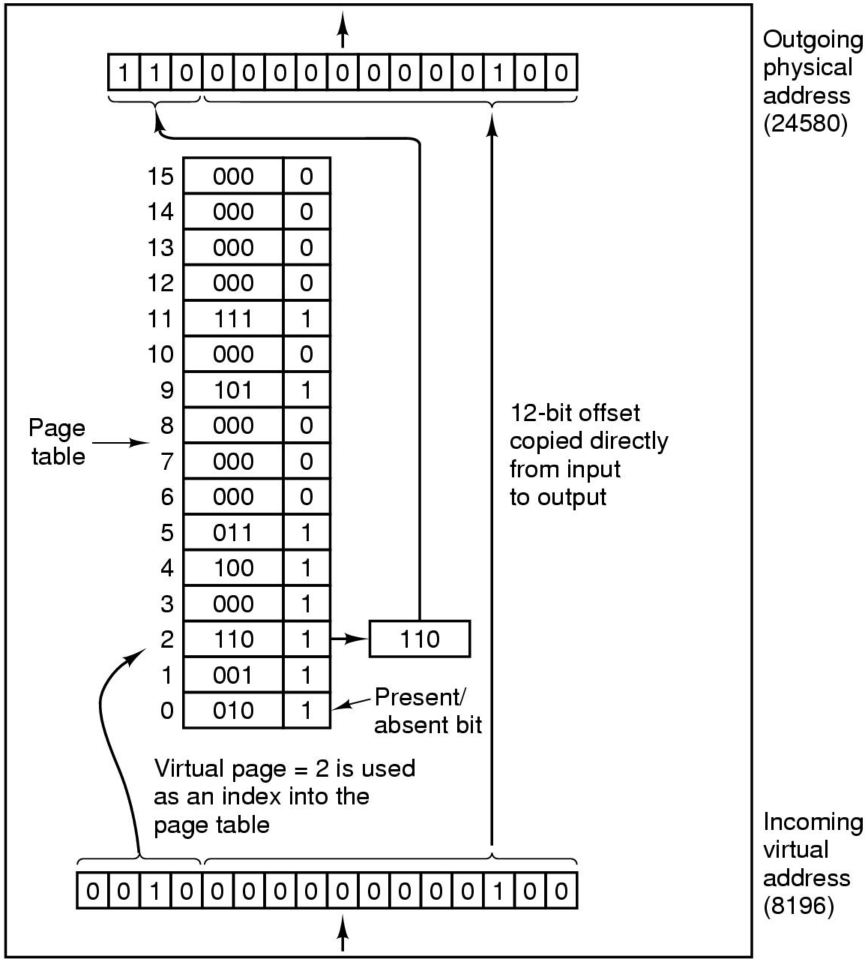


Logical program works in its contiguous virtual address space

Address translation done by MMU

Actual locations of the data in physical memory

Page Tables



Internal operation of MMU with 16 4 KB pages

Two issues:

1. **Mapping must be fast**
2. **Page table can be large**

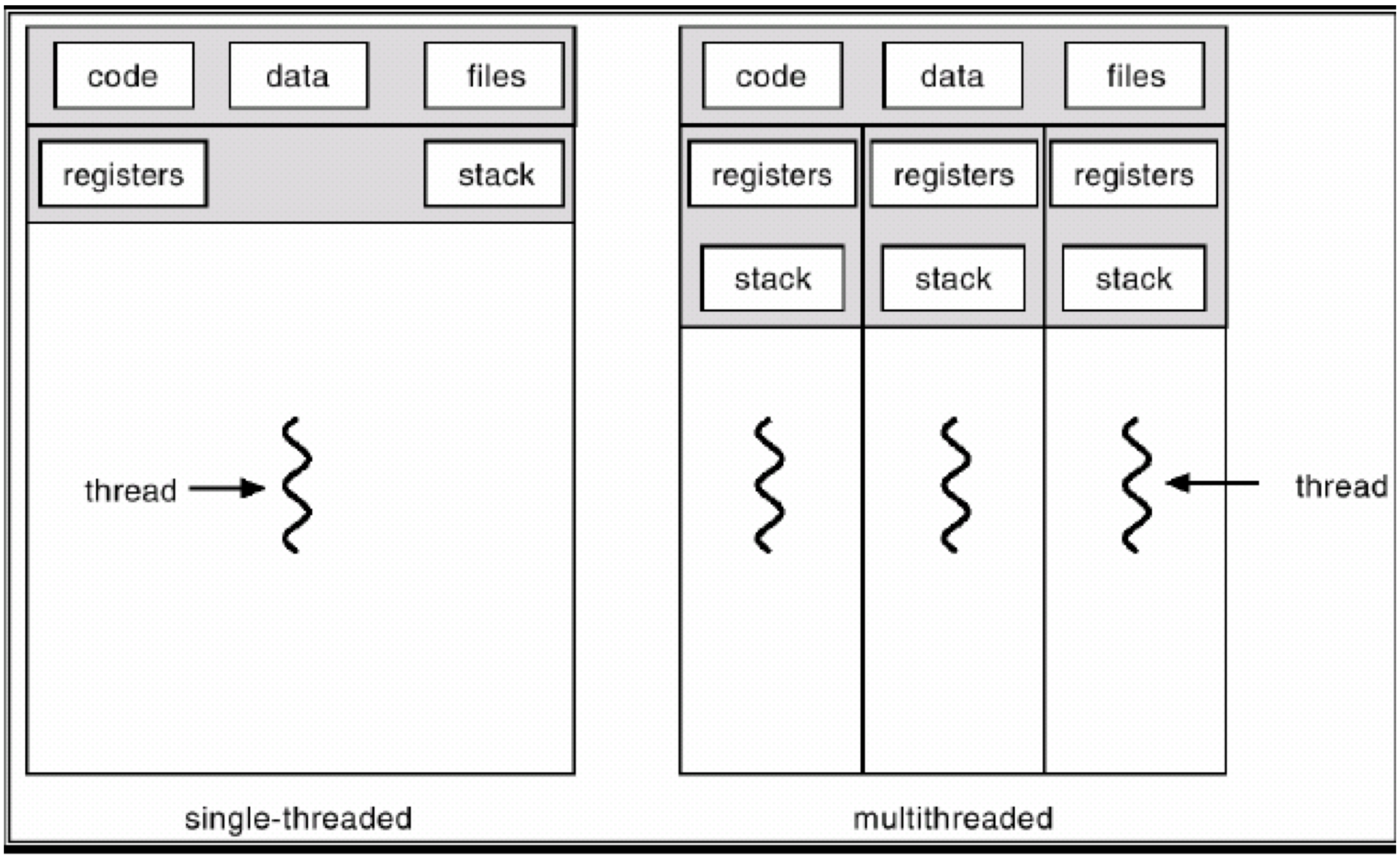
Processes v.s. Threads

- Process
  - ✓ Concurrency
    - Sequential execution stream of instructions
  - ✓ Protection
    - A dedicated address space
- Threads
  - ✓ Separate concurrency from protection
  - ✓ Maintain sequential execution stream of instructions
  - ✓ Share address space with other threads

A Closer Look

- Threads
  - ✓ No data segment or heap
  - ✓ Multiple can coexist in a process
  - ✓ Share code, data, heap, and I/O
  - ✓ Have own stack and registers
  - ✓ Inexpensive to create
  - ✓ Inexpensive context switching
  - ✓ Efficient communication
- Processes
  - ✓ Have data/code/heap
  - ✓ Include at lease one thread
  - ✓ Have own address space, isolated from other processes
  - ✓ Expensive to create
  - ✓ Expensive context switching
  - ✓ IPC can be expensive

An Illustration



IPC Mechanism

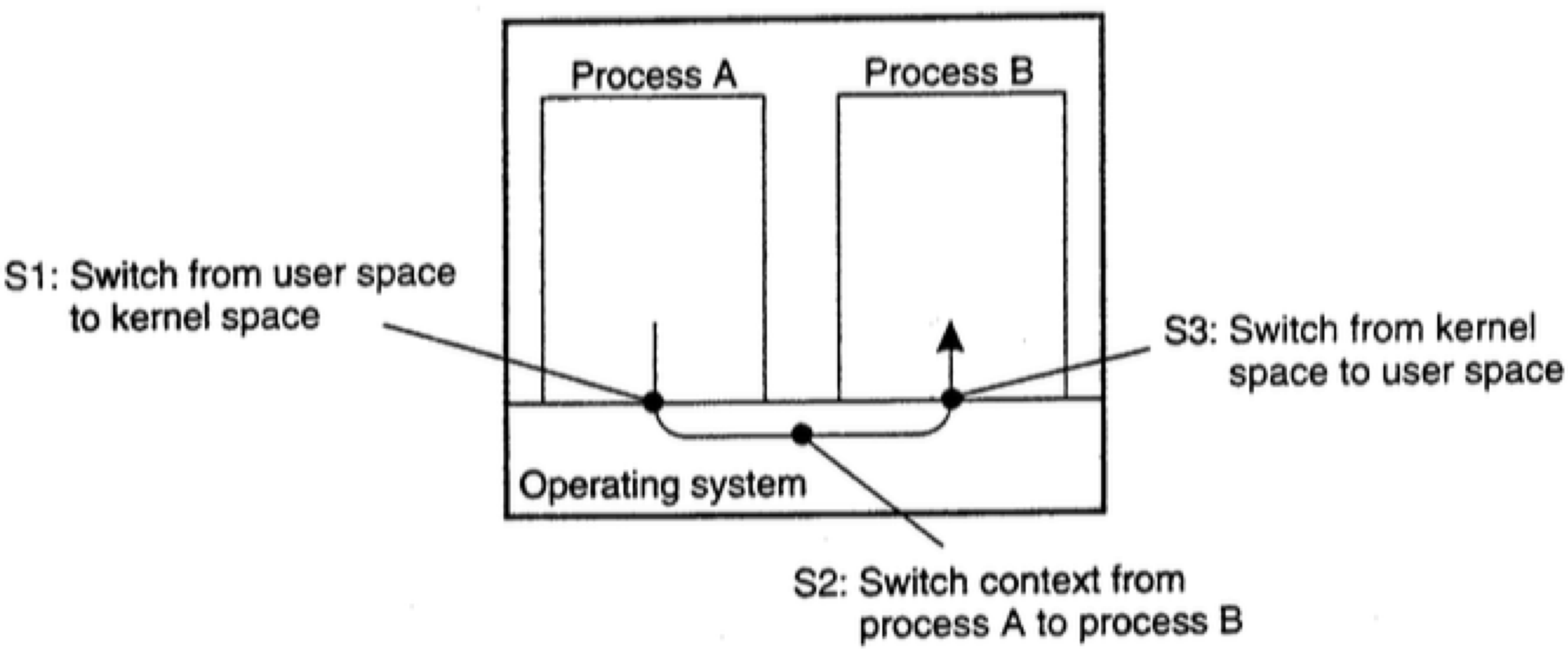
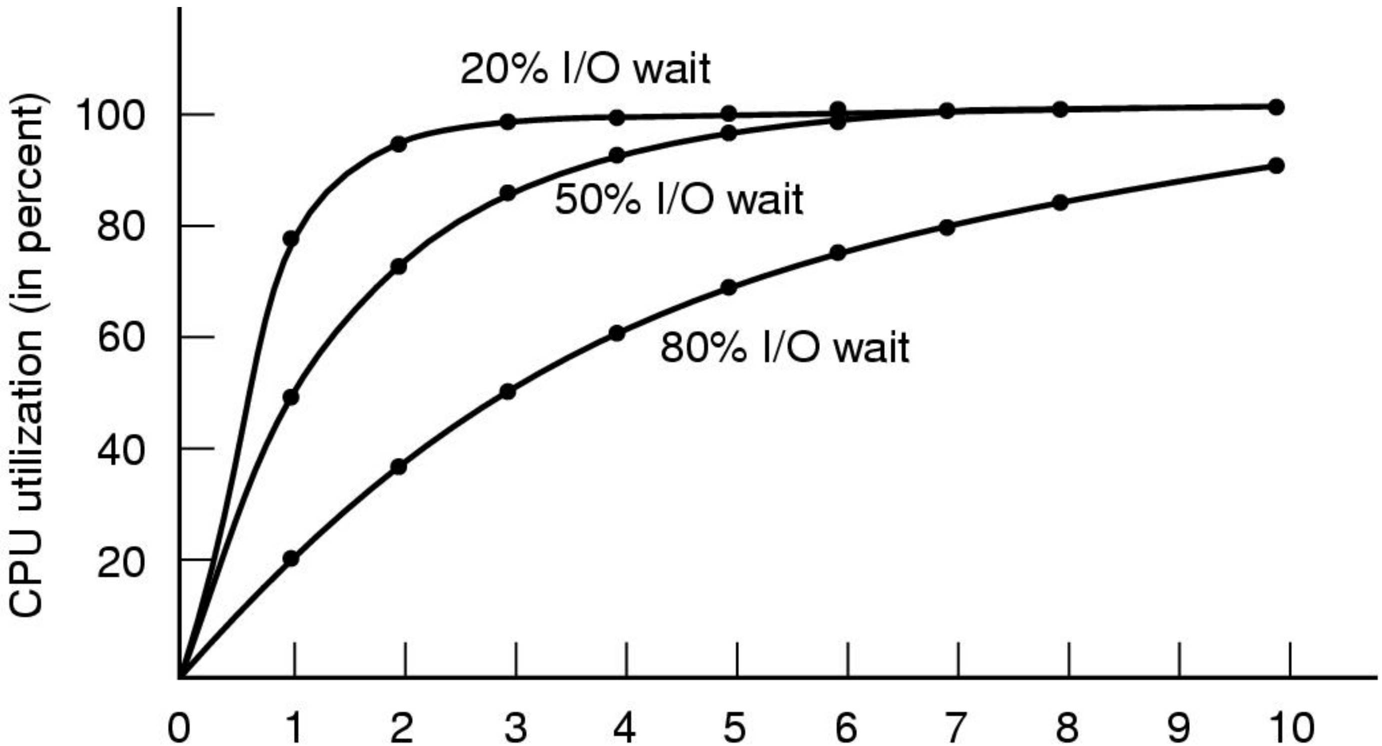


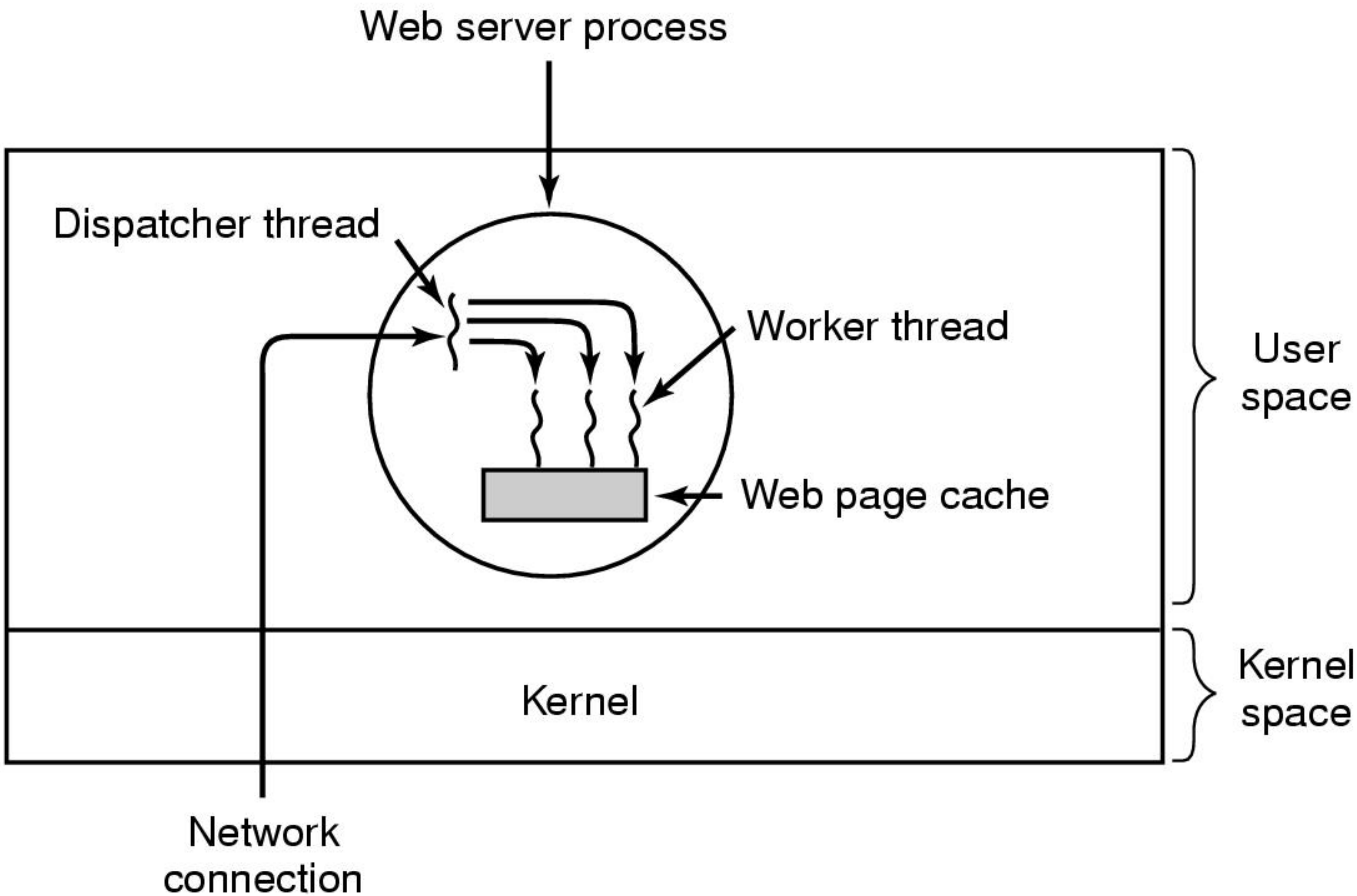
Figure 3-1. Context switching as the result of IPC.

Why Multiprogramming ?



CPU utilization as a function of the number of processes

Thread Usage



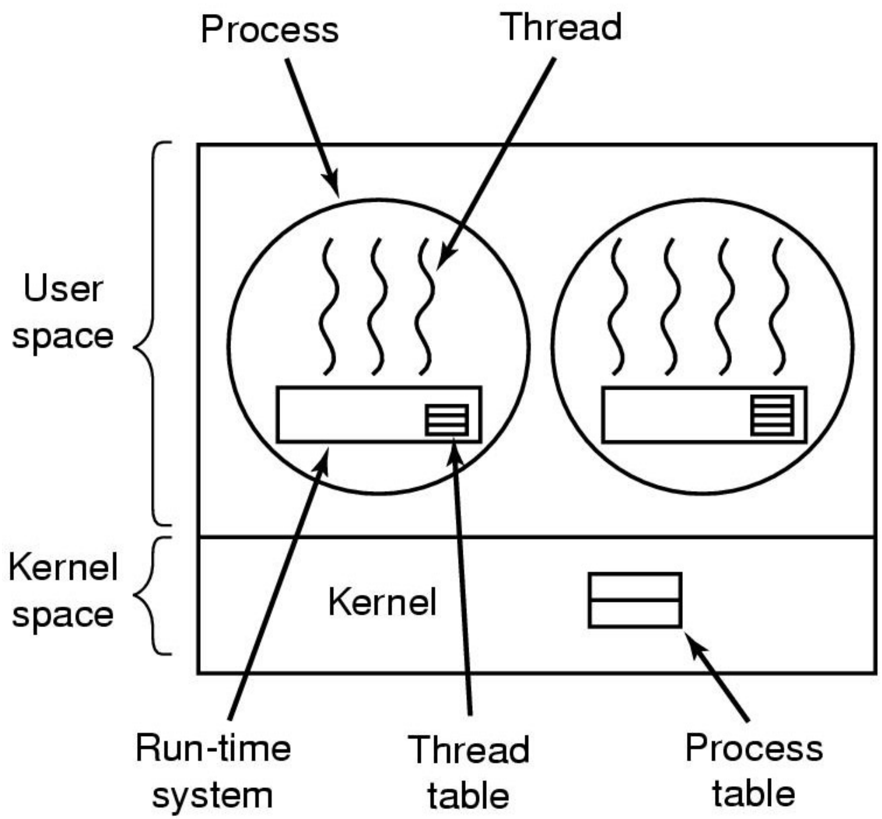
A multithreaded Web server.

A Simple Multi-threaded Webserver

```
void *worker(void *arg) // worker thread
{
          unsigned int socket;
          socket = *(unsigned in *)arg;
          process (socket);
          pthread_exit(0);
}
int main (void) // main thread, or dispatcher thread
{
         unsigned int server_s, client_s, i=0;
         pthread_t threads[200];
         server_s = socket(AF_INET, SOCK_STREAM, 0);
          ……
         listen(server_s, PEND_CONNECTIONS);
         while(1){
                  client_s = accept(server_s, …);
                  pthread_create(&threads[i++], &attr, worker, &client_s);
         }
}
```

Implementing Threads in User-Space

- User-level threads: the kernel knows nothing about them



A user-level threads package

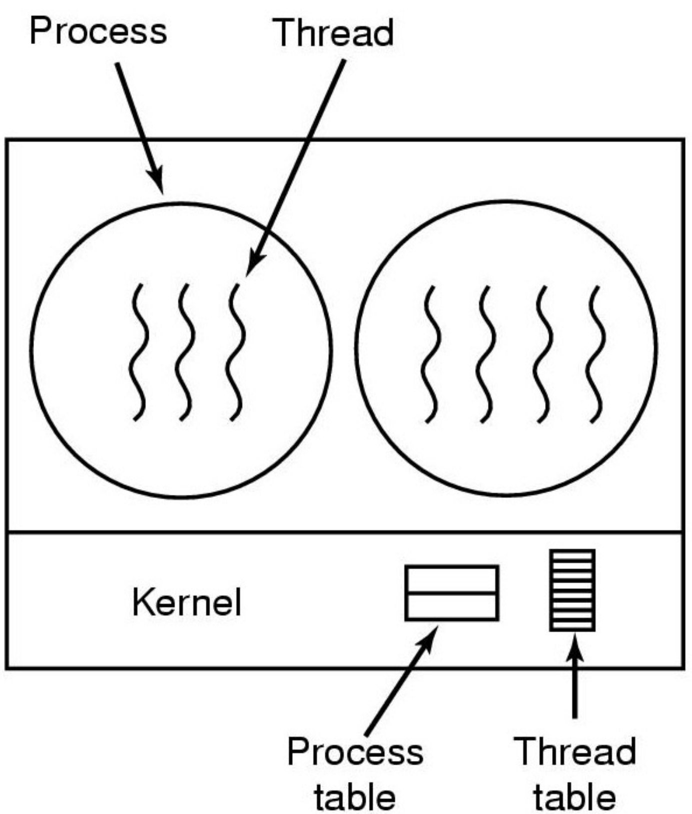
User-level Thread - Discussions

- Advantages
  - No OS thread-support needed
  - Lightweight: thread switching vs. process switching
    - Local procedure vs. system call (trap to kernel)
    - When we say a thread come-to-life? SP & PC switched
  - Each process has its own customized scheduling algorithms
    - thread_yield()

- Disadvantages
  - How blocking system calls implemented? Called by a thread?
    - Goal: to allow each thread to use blocking calls, but to prevent one blocked thread from affecting the others
  - How to change blocking system calls to non-blocking?
  - Jacket/wrapper: code to help check in advance if a call will block
  - How to deal with page faults?
  - How to stop a thread from running forever? No clock interrupts

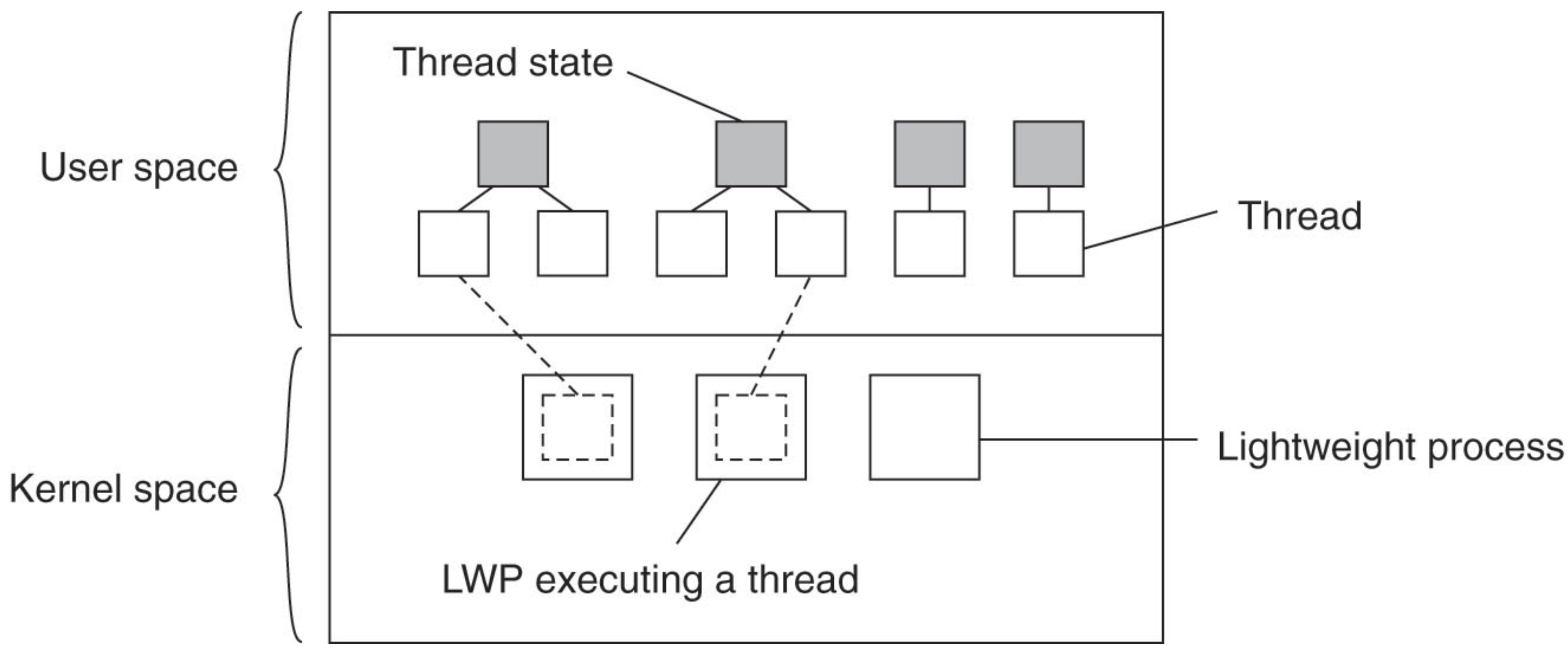
Implementing Threads in the Kernel

- Kernel-level threads: when a thread blocks, kernel re-schedules another thread
  - ✓ Threads known to OS
    - Scheduled by OS scheduler
  - ✓ Slow
    - Trap into the kernel mode
  - ✓ Expensive to create and switch



A threads package managed by the kernel

Hybrid Threading



Combining kernel-level lightweight processes and user-level threads.

Threading Models

- N:1 (User-level threading)
  - ✓ GNU Portable Threads
- 1:1 (Kernel-level threading)
  - ✓ Native POSIX Thread Library (NPTL)
- M:N (Hybrid threading)
  - ✓ Solaris

Three Ways to Construct a Server

- Single-threaded servers
  - ✓ No parallelism, blocking system call
  - ✓ Sequential process model
- Multi-threaded servers
  - ✓ Parallelism, blocking system call
  - ✓ Sequential process model
- Finite-state machine
  - ✓ Parallelism, must use non-blocking system call
  - ✓ Sequential process model lost

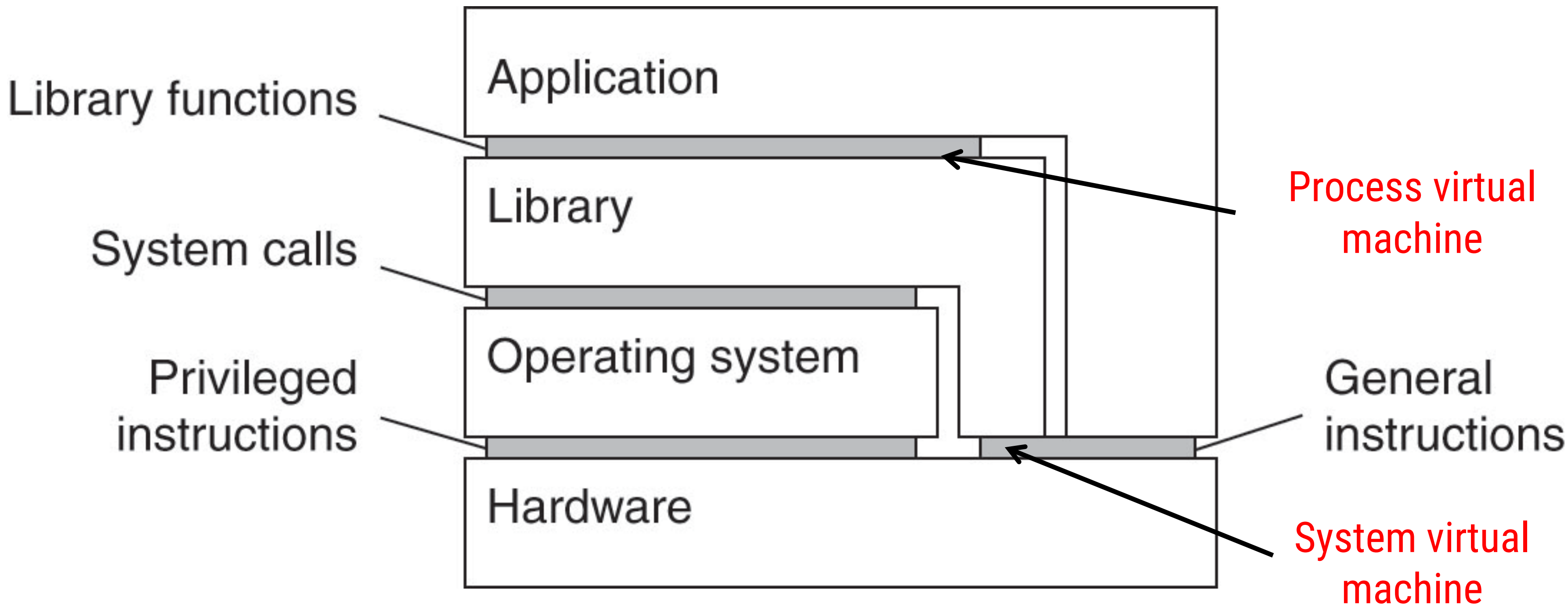
Virtualization

- Why virtualization?
  - ✓ In early days, to allow legacy software to run on expensive mainframe hardware
  - ✓ Hardware and low-level system software changes quickly but the software at high level remains stable
  - ✓ Portability and flexibility
  - ✓ Fault isolation

Architectures of Virtual Machines

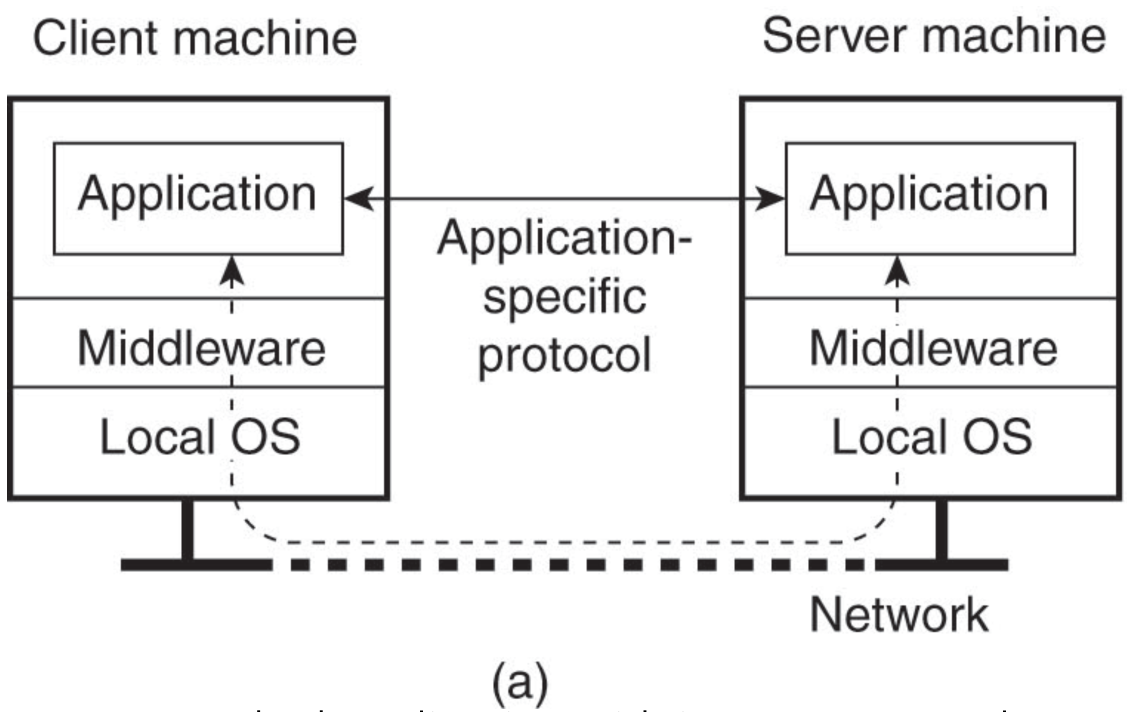
- Computer systems offer four types of interfaces
  - ✓ An interface between the hardware and software, consisting of machine instructions (non-privileged inst.)
  - ✓ An interface between the hardware and software, consisting of privileged instructions
  - ✓ An interface consisting of system calls offered by OS
  - ✓ An interface consisting of library calls

Logical View of Four Interfaces



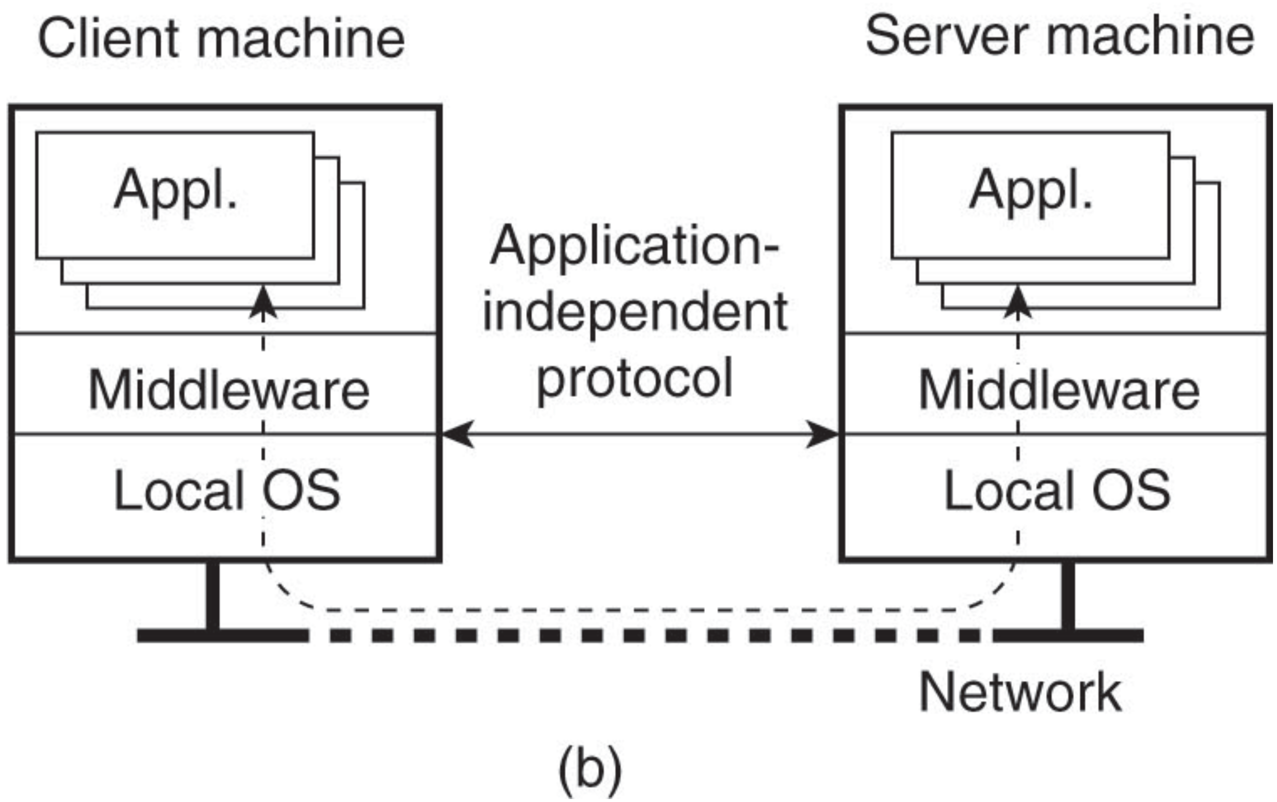
Client-side Processes

- The major task is to provide user interface to access remote servers



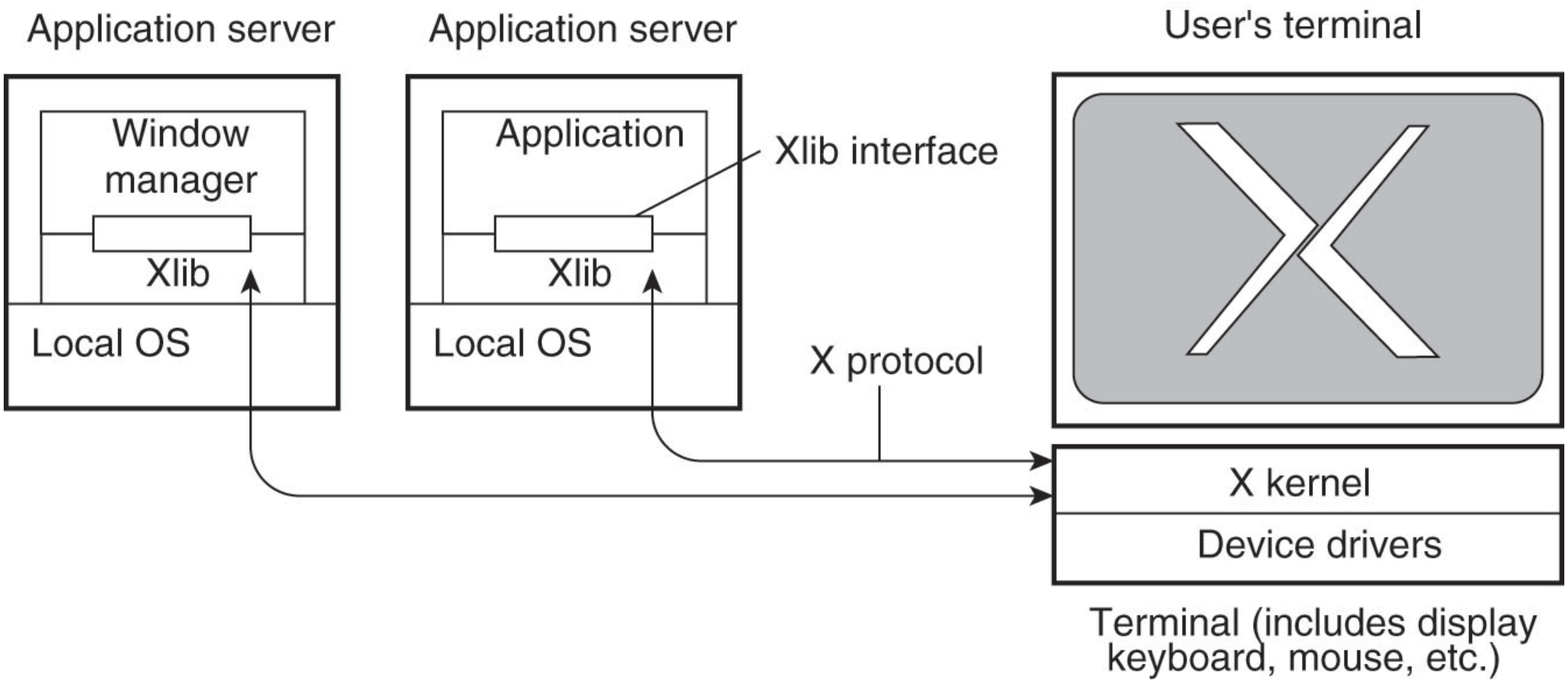
A networked application with its own protocol.

Thin-client Approach



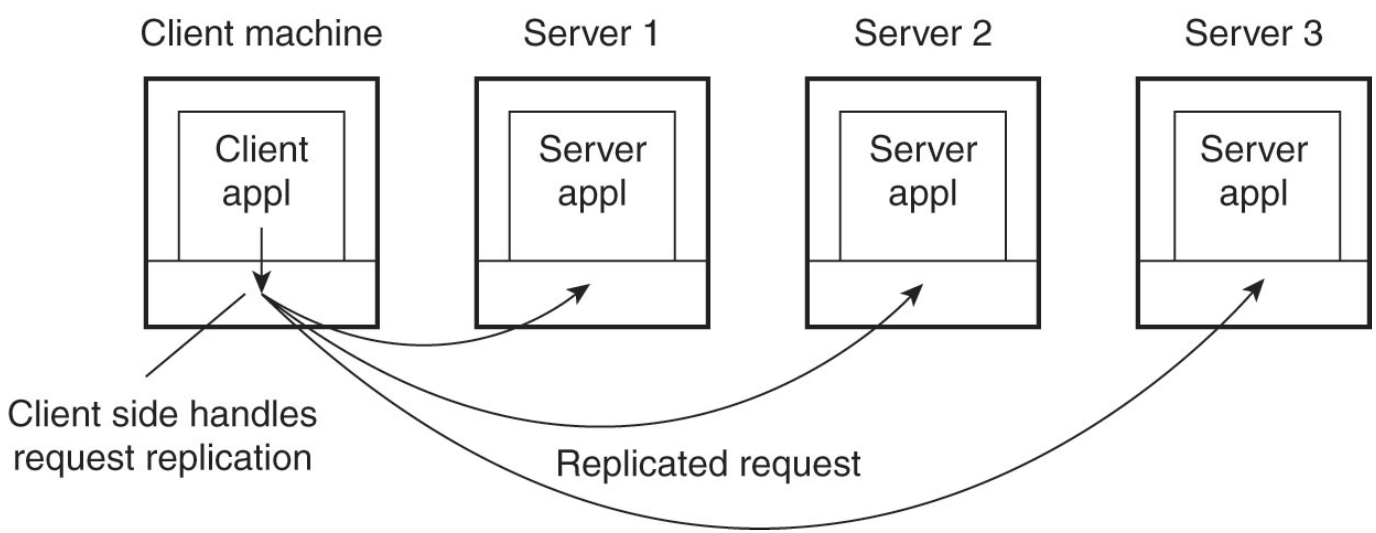
A general solution to allow access to remote applications.

Example: The XWindow System



Other Client-side Tasks

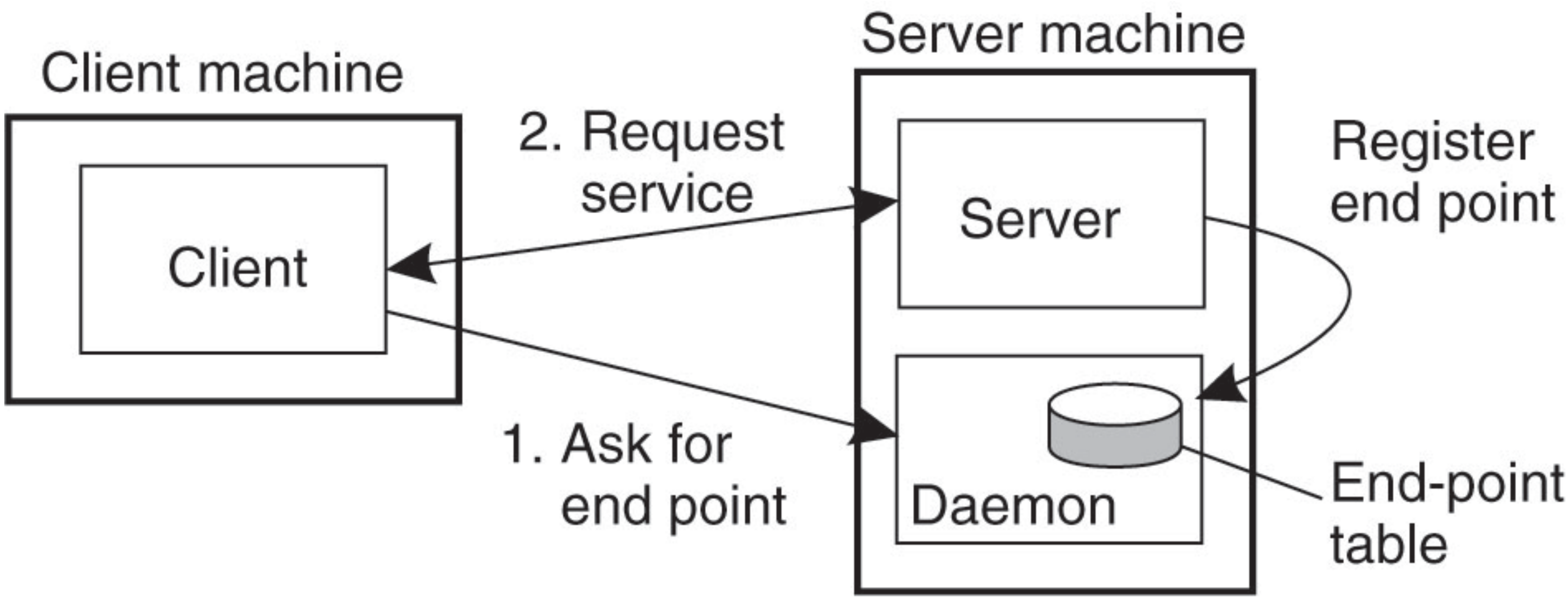
- In addition to network user interface, the client side may
  - ✓ Handle part of the processing level and data level
  - ✓ Have components to achieve distribution transparency
  - ✓ Have components to achieve failure transparency



Server-side Processes

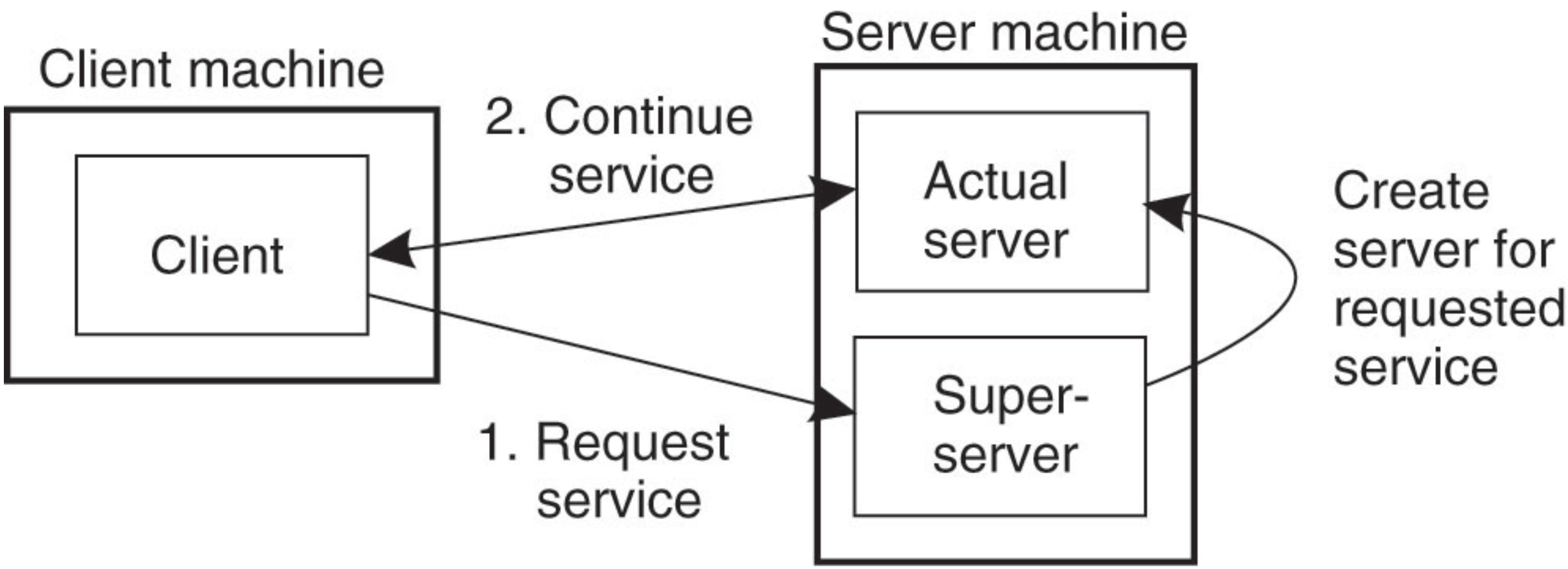
- Generally a server
  - ✓ Waits for an incoming request from a client
  - ✓ Ensures that the request has been taken care of
  - ✓ Waits for the next request
- General design issues
  - ✓ How to organize servers
  - ✓ How to locate the needed service
  - ✓ Where and how a server can be interrupted
  - ✓ Whether or not the server is stateless

Client-server Binding (Daemon)



(a)

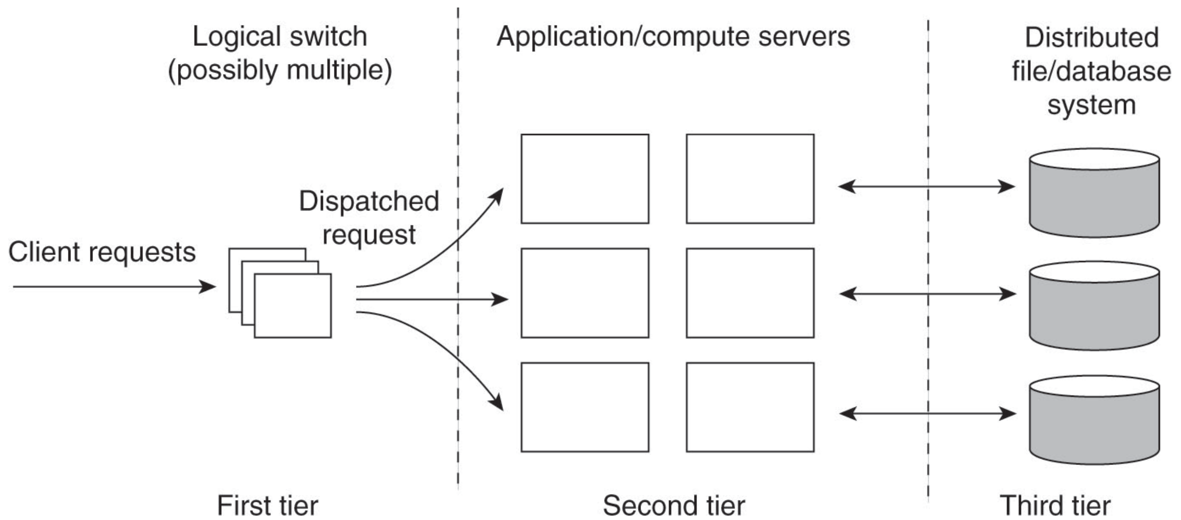
Client-server Binding (Superserver)



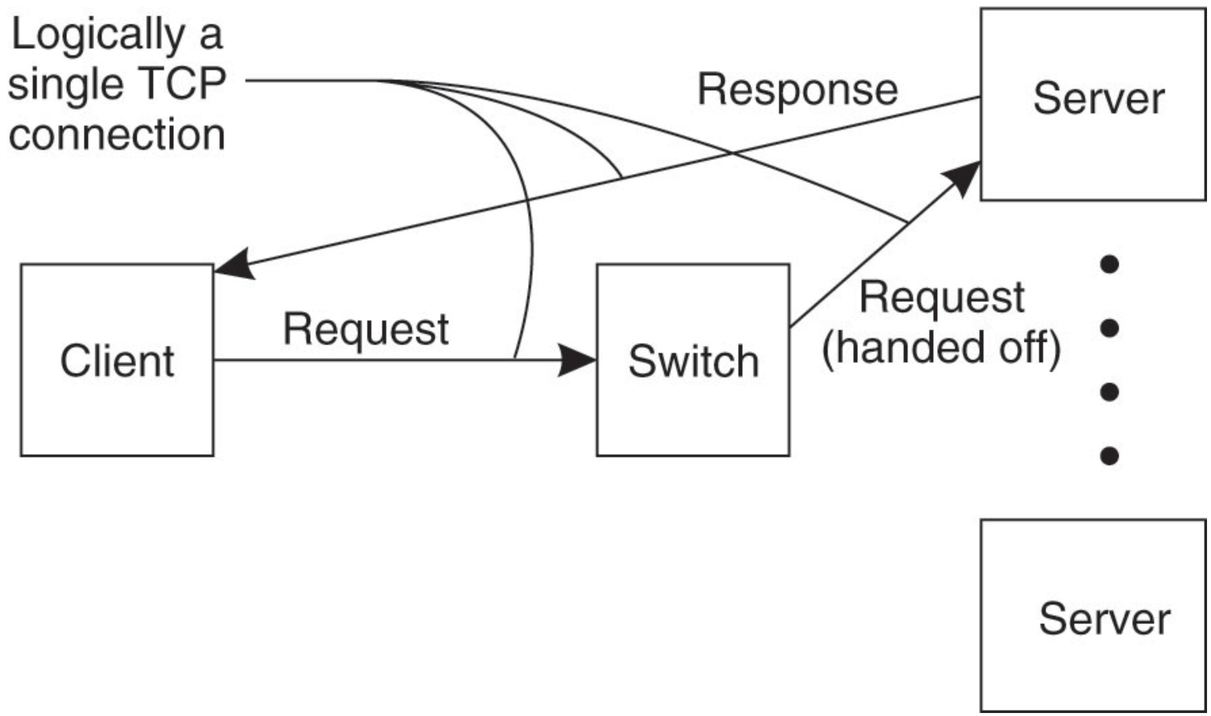
(b)

Server Cluster

- The need for a server cluster
  - ✓ A single computer cannot handle the needed bandwidth, computing, failure resistance, etc.
- The 3-tier architecture



Hiding the Cluster from Clients

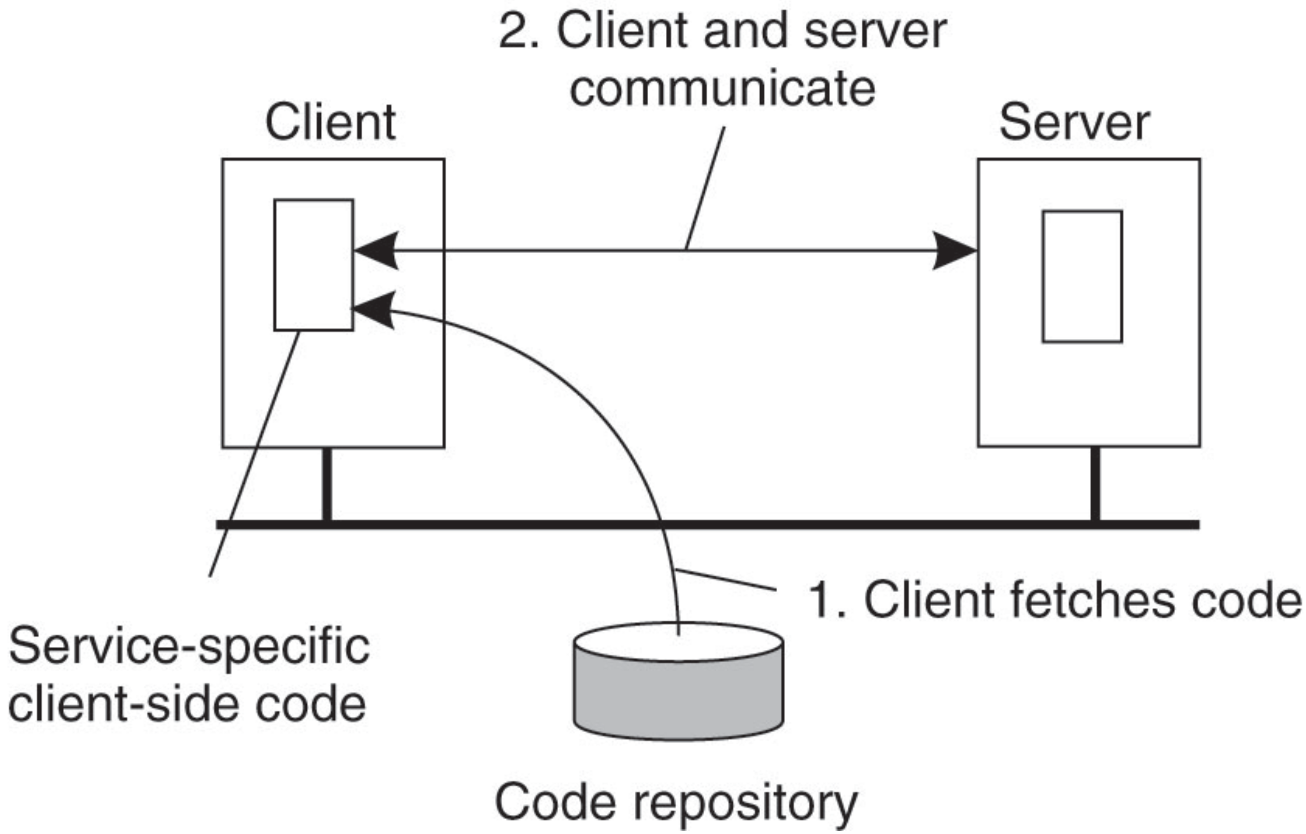


The principle of TCP handoff.

Code Migration

- The communication in the distributed systems discussed so far is limited to passing data
- Being able to pass code, even while in execution, can
  - ✓ Simplify distributed systems design
  - ✓ Improve performance by load balancing processes
  - ✓ Improve performance by exploiting parallelism
  - ✓ Provide flexibility, e.g., clients don’t need to install software

Reasons for Code Migration



Code Migration Examples (1/2)

- Example 1: (Send client code to server)
  - ✓ The server holds a huge database
  - ✓ It is better for a client to ship part of its application to the server and server sends only the results back
- Example 2: (Send server code to client)
  - ✓ In many DB applications, clients need to fill in forms that are translated into DB operations
  - ✓ The validation of the form can be moved to the client side to save the computation power of the server

Code Migration Examples (2/2)

- Example 3:
  - ✓ System administrator may be forced to shut down a server but does not want to stop the running process
- Example 4:
  - ✓ Temporarily freeze an environment, move to another machine and unfreeze (Live migration)

Models for Code Migration

- A process consists of
  - ✓ Code segment
  - ✓ Resource segment
  - ✓ Execution segment
- Weak mobility
  - ✓ Migrate only the code segment
- Strong mobility
  - ✓ Migrate all three segments
- Receiver-initiated: receiver requests code
  - ✓ Usually simple since receivers ask for info
- Sender-initiated: sender pushes code
  - ✓ Must make sure the sender is authenticated

Migration and Local Resource

- Resource migration examples:
  - ✓ What happens to a TCP port opened by a migrating process
  - ✓ URL reference to a file when the code is moved
- Resource types:
  - ✓ Fixed resources (e.g., local disks, NIC ports)
  - ✓ Unattached resources (e.g., data files)
  - ✓ Fastened resources (e.g., local databases)
- Binding strength:
  - ✓ (strongest) By identifier, e.g., URL
  - ✓ (weaker) By value, e.g., standard libraries
  - ✓ (weakest) By type, e.g., printer

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)

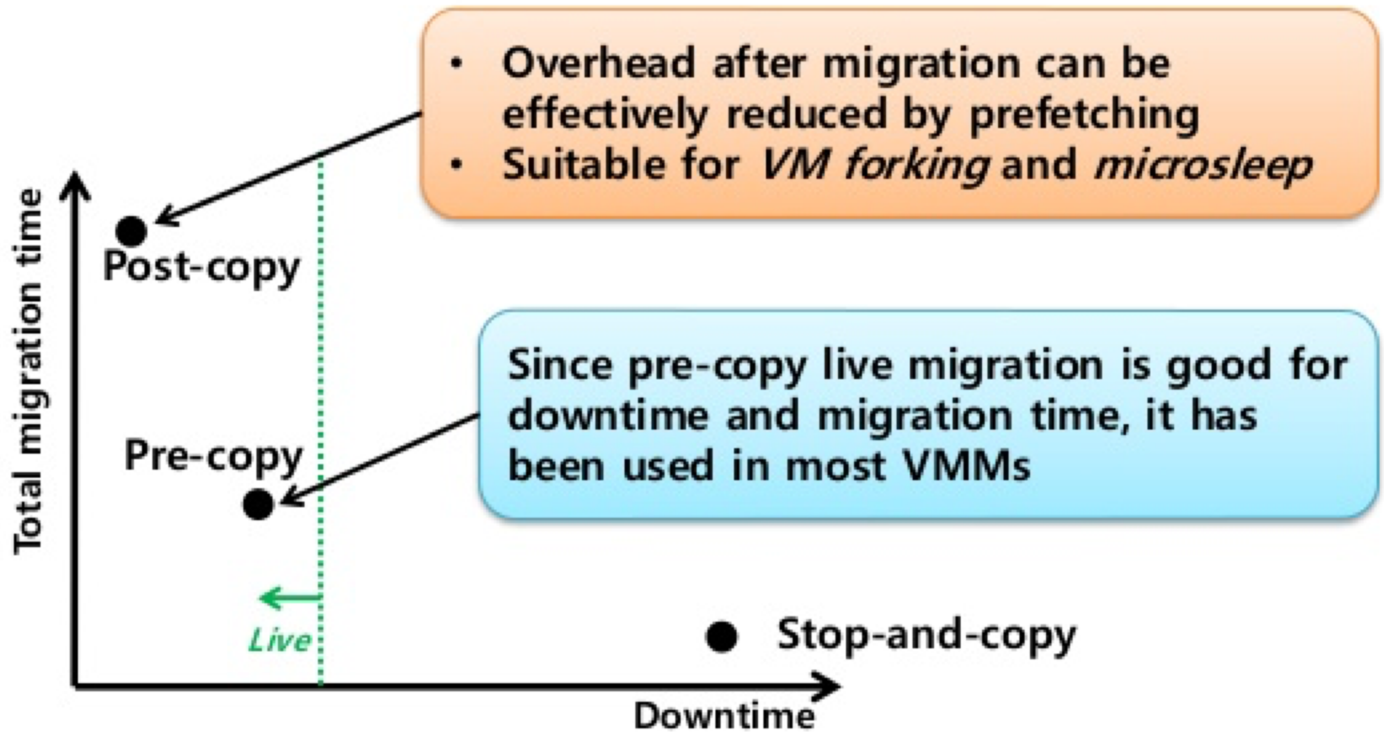
GR Establish a global systemwide reference
MV Move the resource
CP Copy the value of the resource
RB Rebind process to locally-available resource

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

Migration in Heterogeneous Systems

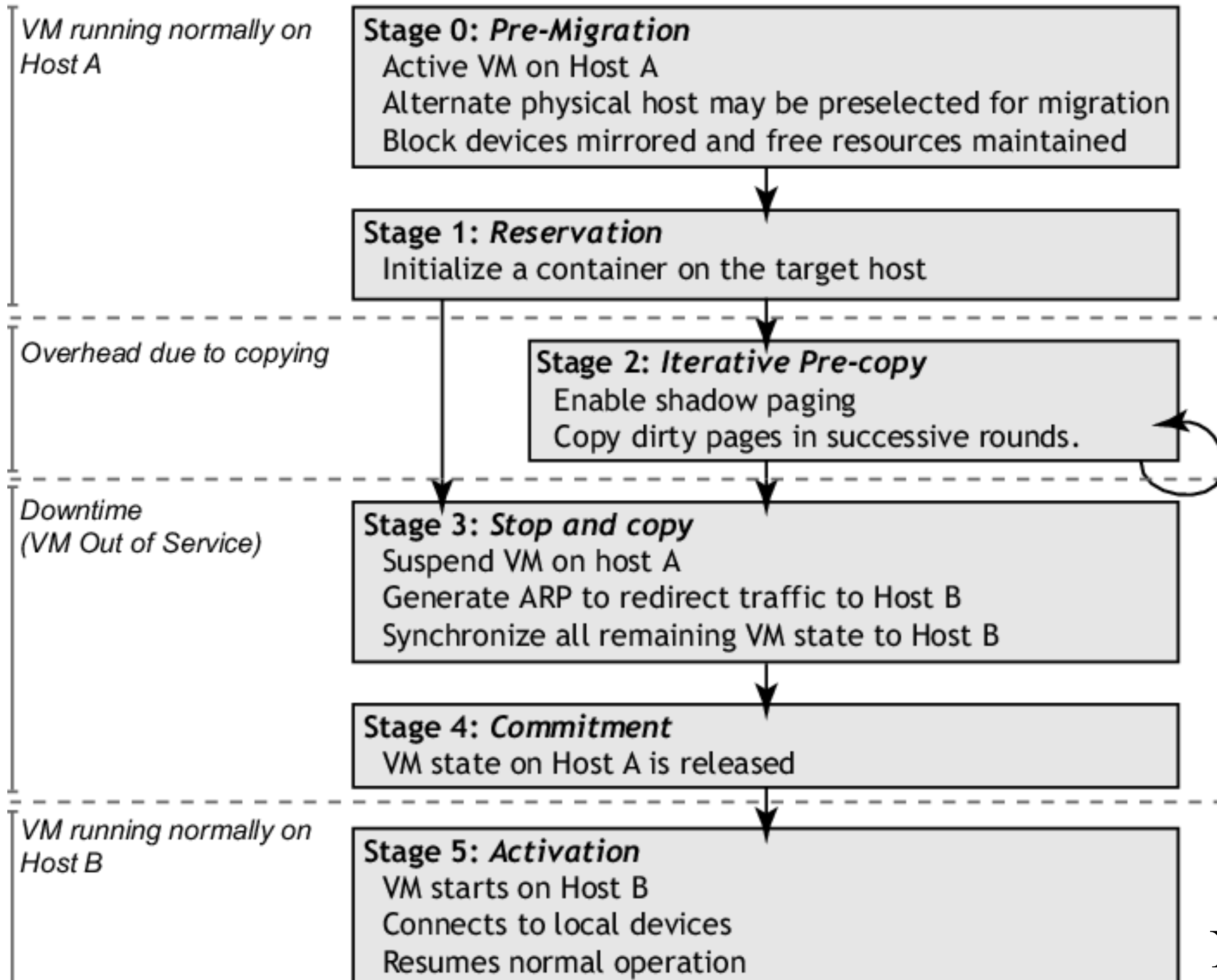
- Virtual machine migration
  - ✓ Pre-copy migration: pushing memory pages to the new VM and resending the ones that are later modified during the migration process
  - ✓ Stop-and copy migration: stopping the current VM; migrate memory, and start the new VM
  - ✓ Post-copy migration: letting the new VM pull in new pages as needed, that is, let processes start on the new VM immediately and copy memory pages on demand

Trade-off



• Overhead after migration can be effectively reduced by prefetching
• Suitable for *VM forking* and *microsleep*
Since pre-copy live migration is good for downtime and migration time, it has been used in most VMMs
Total migration time
Post-copy
Pre-copy
Live
Stop-and-copy
Downtime

Pre-Copy Migration



NSDI'05