The picture can't be displayed.

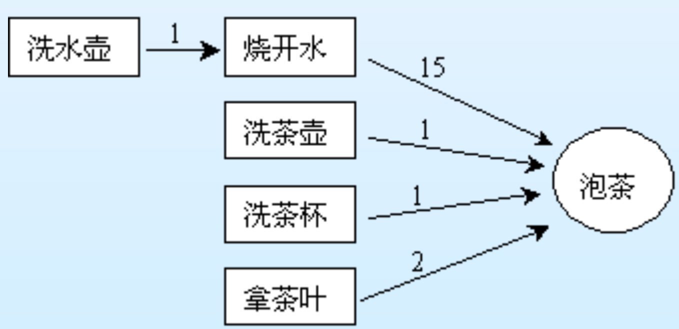
Chapter 3: Processes

A Lesson from High School

比如，想泡壶茶喝。当时的情况是：开水没有。开水壶要洗，茶壶茶杯要洗；火已升了，茶叶也有了。怎么办？

1. 办法甲：洗好开水壶，灌上凉水，放在火上；在等待水开的时候，洗茶壶、洗茶杯、拿茶叶；等水开了，泡茶喝。
2. 办法乙：先做好一些准备工作，洗开水壶，洗壶杯，拿茶叶；一切就绪，灌水烧水；坐待水开了，泡茶喝。
3. 办法丙：洗净开水壶，灌上凉水，放在火上；坐待水开，开了之后急急忙忙找茶叶，洗壶杯，泡茶喝。

哪一种办法省时间？谁都能一眼看出，第一种办法好，因为后二种办法都“窝了工”



Chapter 3: Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Cooperating Processes
- Interprocess Communication
- Communication in Client-Server Systems

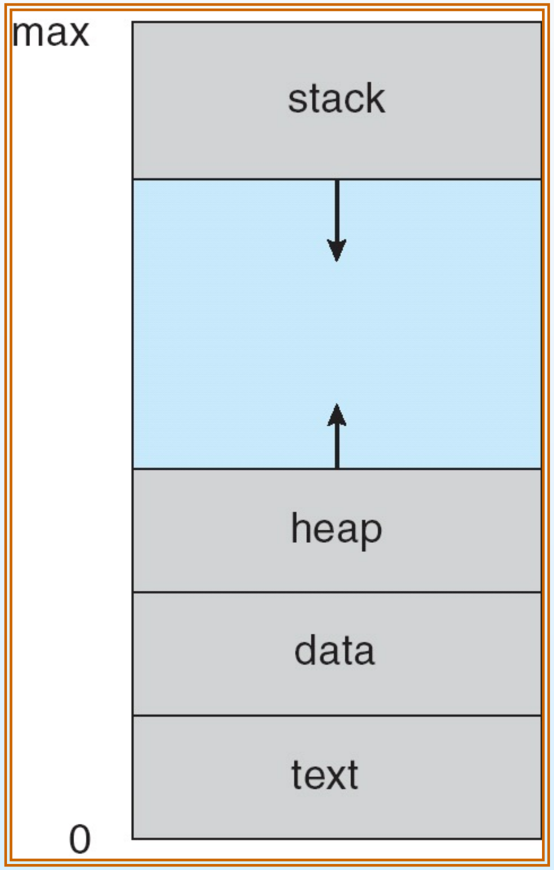
Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs

  - Time-shared systems – user programs or tasks
- In textbook, *job* = *process*
- Process – a program in execution using sequential fashion
- A process includes:
  - text section (code)
  - program counter
  - stack (function parameters, local vars, return addresses)
  - data section (global vars)
  - heap (dynamically allocated memory)

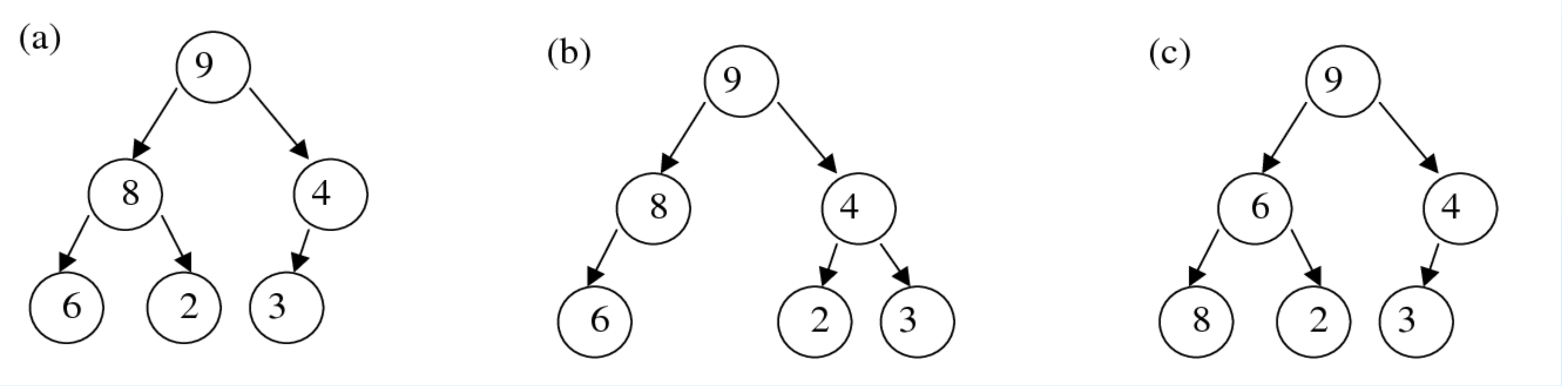
Process in Memory

```
class A{
   public:
              const int a = 0;
         ....
         int getValue(){
             int[] buffer = new int[1024];
             ....
             delete buffer;
             return 0;
         }
         ....
}
```



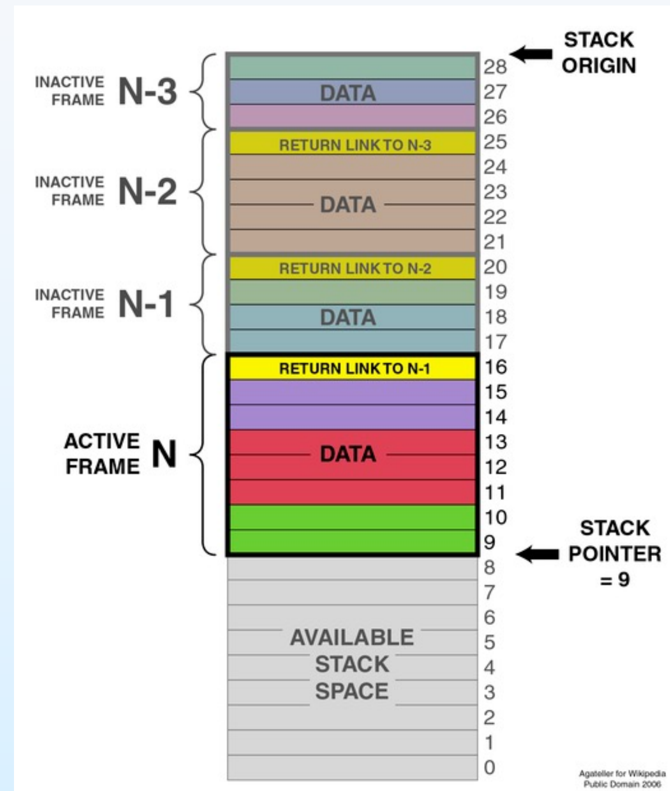
Data Structure: Heap

- Heap is a "complete" binary tree
  - Except the leaf level, all levels are full. The leaf level is filled from left to right
  - The node's value should be larger/smaller than its children
- (a) is a heap
- (b) is not for it is not complete,
- (c) is not because 6<8



Data Structure: Stack

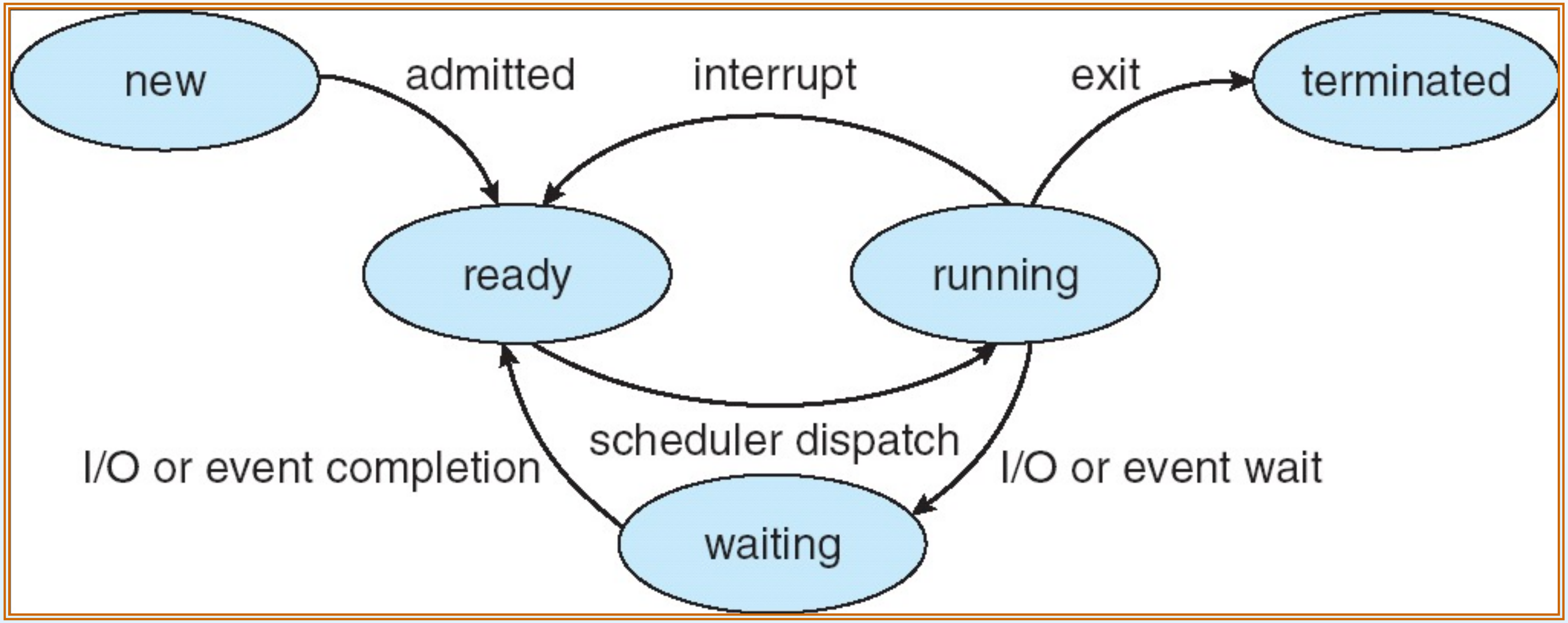
- A typical stack is an area of computer memory with a fixed origin and a variable size.
- Initially the size of the stack is zero.
- A stack pointer, usually in the form of a hardware register, points to the most recently referenced location on the stack
- a push operation, in which a data item is placed at the location pointed to by the stack pointer, and the address in the stack pointer is adjusted by the size of the data item;
- a pop or pull operation: a data item at the current location pointed to by the stack pointer is removed, and the stack pointer is adjusted by the size of the data item.



Process State

- As a process executes, it changes *state*
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting/blocked for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

Diagram of Process State

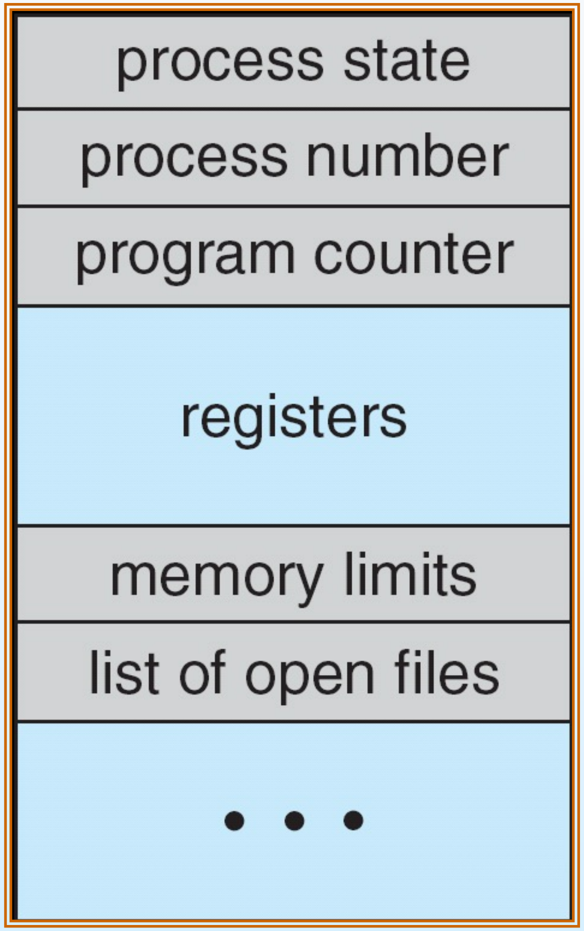


Process Control Block (PCB)

Information associated with each process

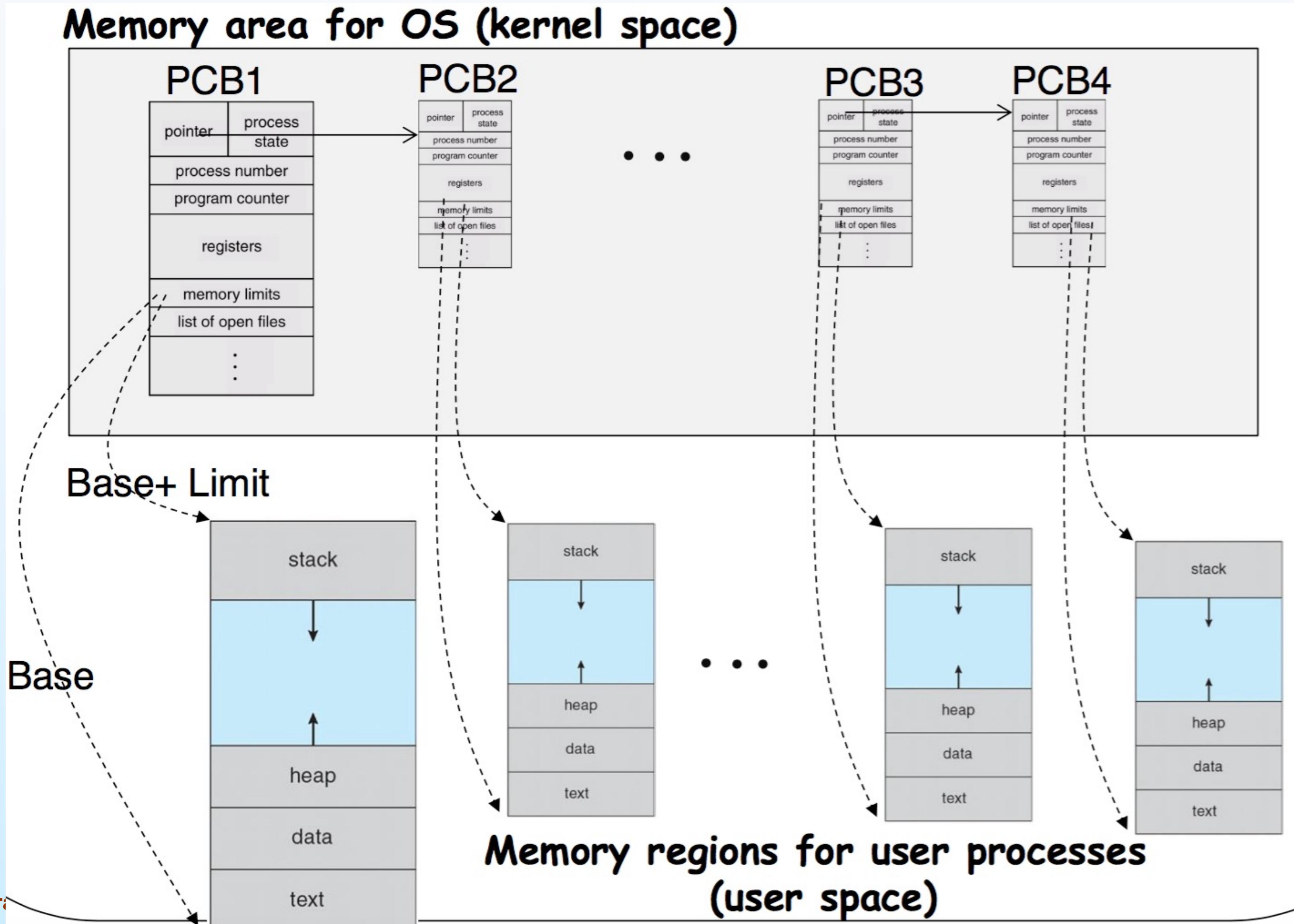
- Process state (new, ready, ...)
- Program counter (address of next instruction)
- Contents of CPU registers
- CPU scheduling information (priority)
- Memory-management information
- Accounting information (cpu/time used, time limits, process number)
- I/O status information (allocated devices and opened files)

Process Control Block (PCB)

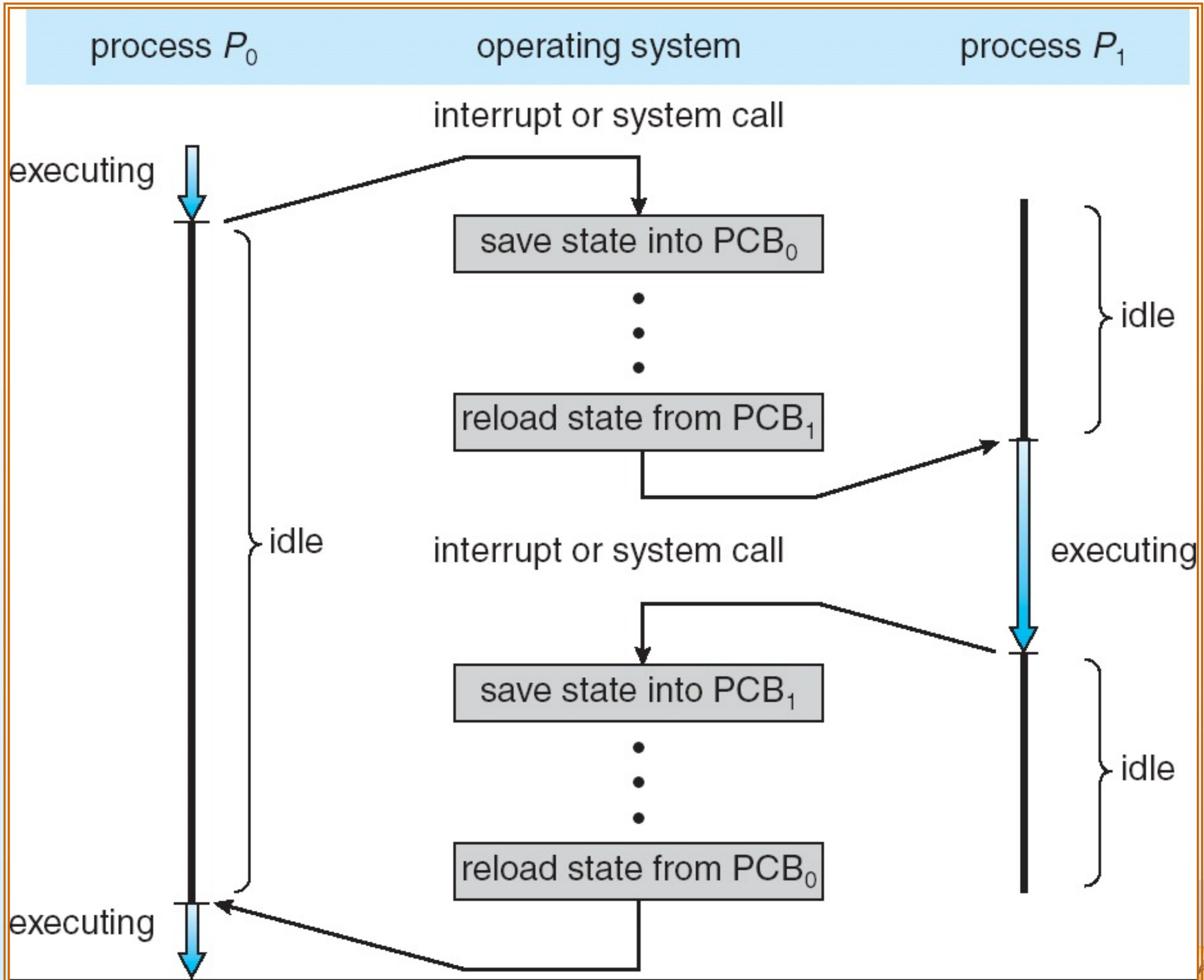



```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   512  2667 32203  0  80   0 - 26525 wait   pts/12   00:00:00 sh
0 D   512  2668  2667 29  80   0 - 50494499 sync_p pts/12 01:56:34 convert_imagese
0 S   504  4923 27549  0  80   0 - 39358 n_tty_ pts/21   00:00:00 mysql
0 S   529  6508  6355  0  80   0 -  3894 poll_s pts/6    00:00:16 top
0 R   501  6800  6773  0  80   0 - 27033 -      pts/32   00:00:00 ps
0 S   523  7485 13843  0  80   0 - 966037 futex_ pts/15  03:48:24 MATLAB
0 S   523 10629  1194  0  80   0 - 15047 n_tty_ pts/8    00:00:00 ssh
0 S   503 14152 14135  0  80   0 - 26524 wait   pts/5    00:00:00 nsight
0 S   503 14156 14152  0  80   0 - 26555 wait   pts/5    00:00:00 nsight
0 S   503 14157 14156  2  80   0 - 25208576 futex_ pts/5 00:48:03 java
4 S     0 14621 14605  0  80   0 - 40420 wait   pts/18   00:00:00 su
4 S     0 14632 14621  0  80   0 - 27117 n_tty_ pts/18   00:00:00 bash
0 S   501 16712 16692  0  80   0 - 15047 poll_s pts/10   00:00:00 ssh
4 S     0 17091 32191  0  80   0 - 40413 wait   pts/22   00:00:00 su
4 S     0 17109 17091  0  80   0 - 27117 n_tty_ pts/22   00:00:00 bash
4 S     0 22140     1  0  80   0 - 27050 wait   pts/22   00:00:00 mysqld_safe
4 S    27 22242 22140  1  80   0 - 144273 poll_s pts/22  01:16:10 mysqld
0 S   504 23075 17786  0  80   0 - 39382 n_tty_ pts/30   00:00:00 mysql
```

Process Control Block (PCB)



CPU Switch From Process to Process



Chapter 3: Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Cooperating Processes
- Interprocess Communication
- Communication in Client-Server Systems

New Teaching Assistant: zhifeipang@zju.edu.cn

Process Scheduling Queues

- **Job queue** – set of all processes in the system
- **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
- **Device queues** – set of processes waiting for an I/O device
- Processes migrate among the various queues

1001
1002
1003
1004
1005

job queue

1001
1003
1005

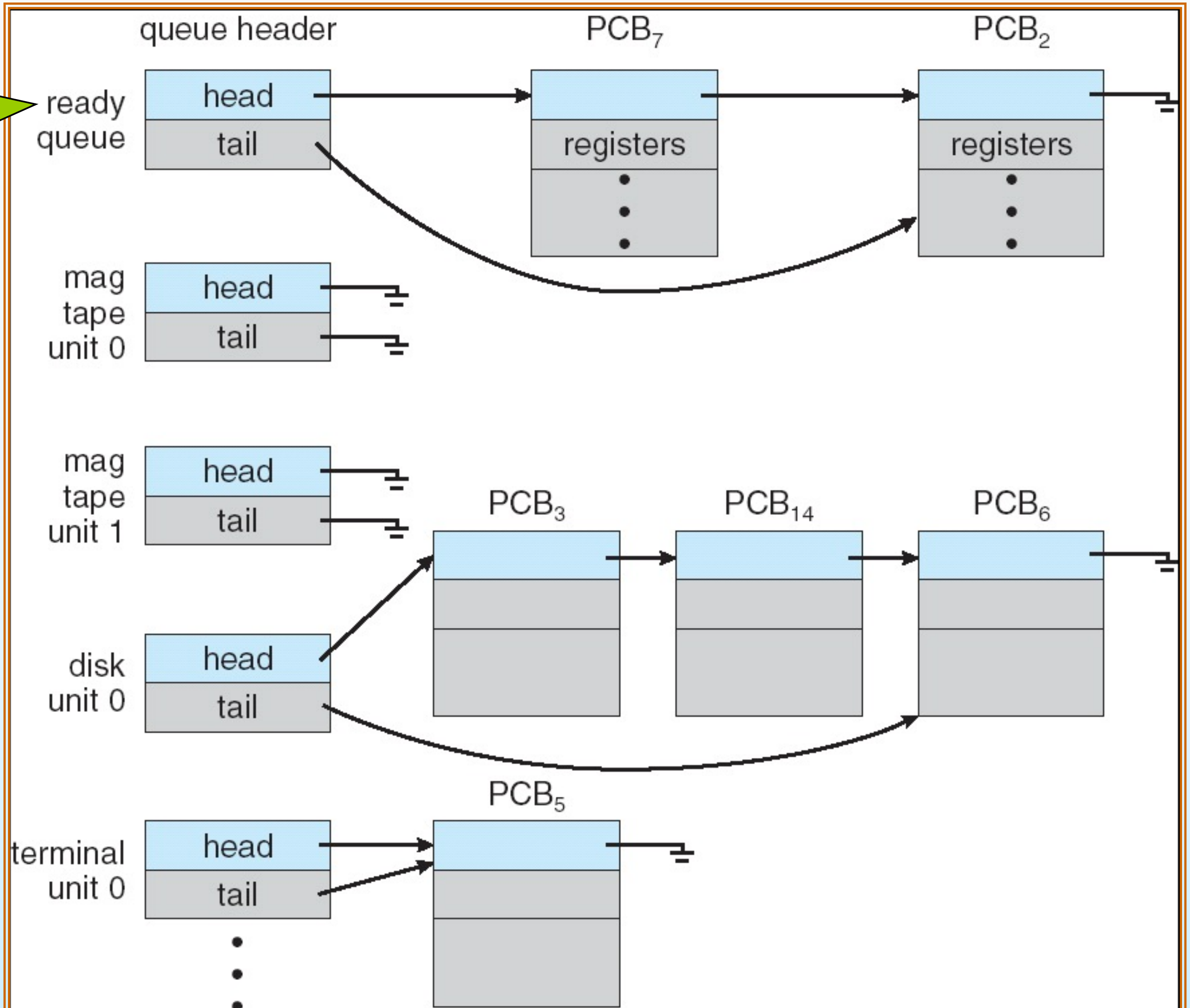
ready queue

Disk: 1002
Network Card: 1004

device queue

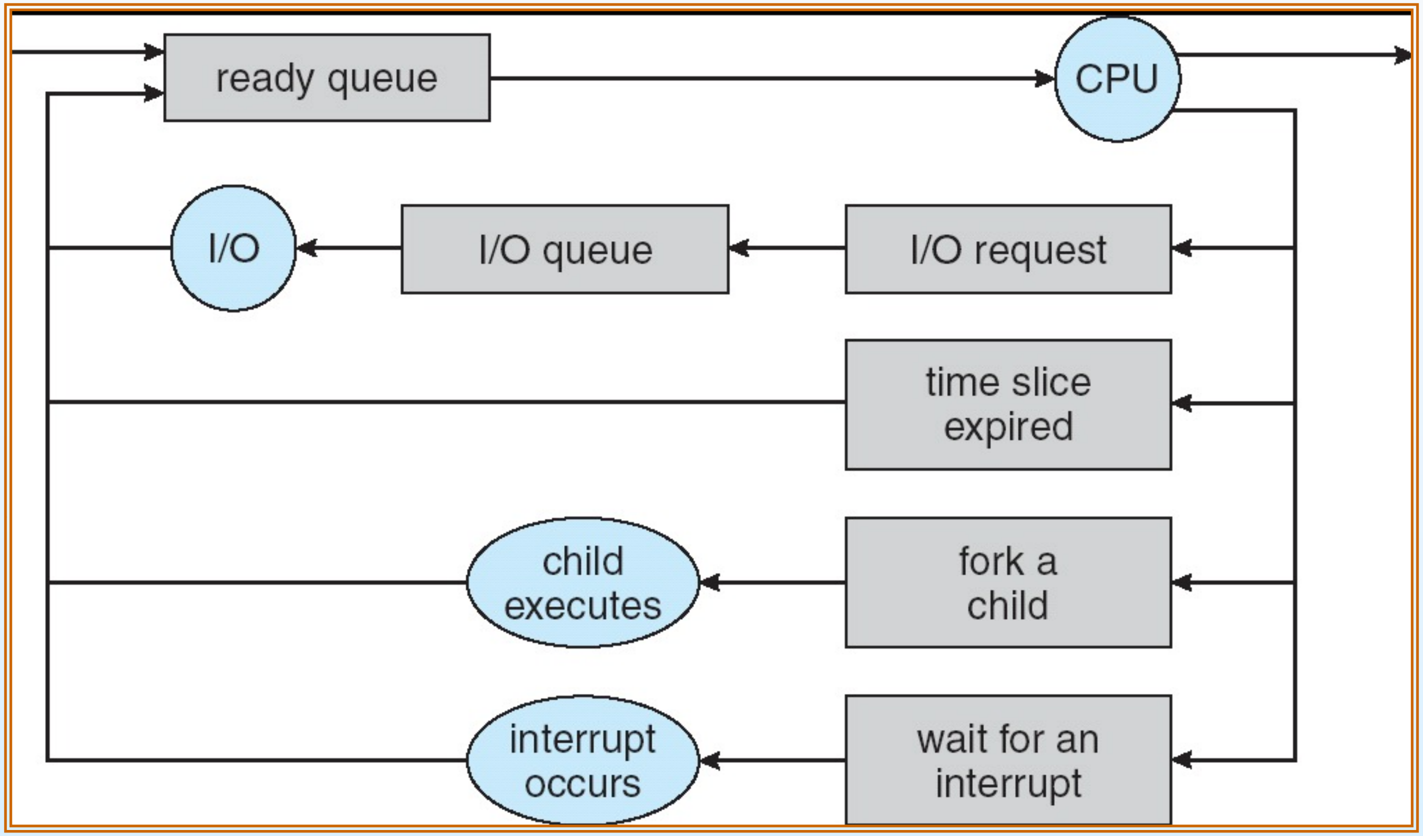
Ready Queue And Various I/O Device Queues

The CPU queue



Representation of Process Scheduling

A queueing-diagram



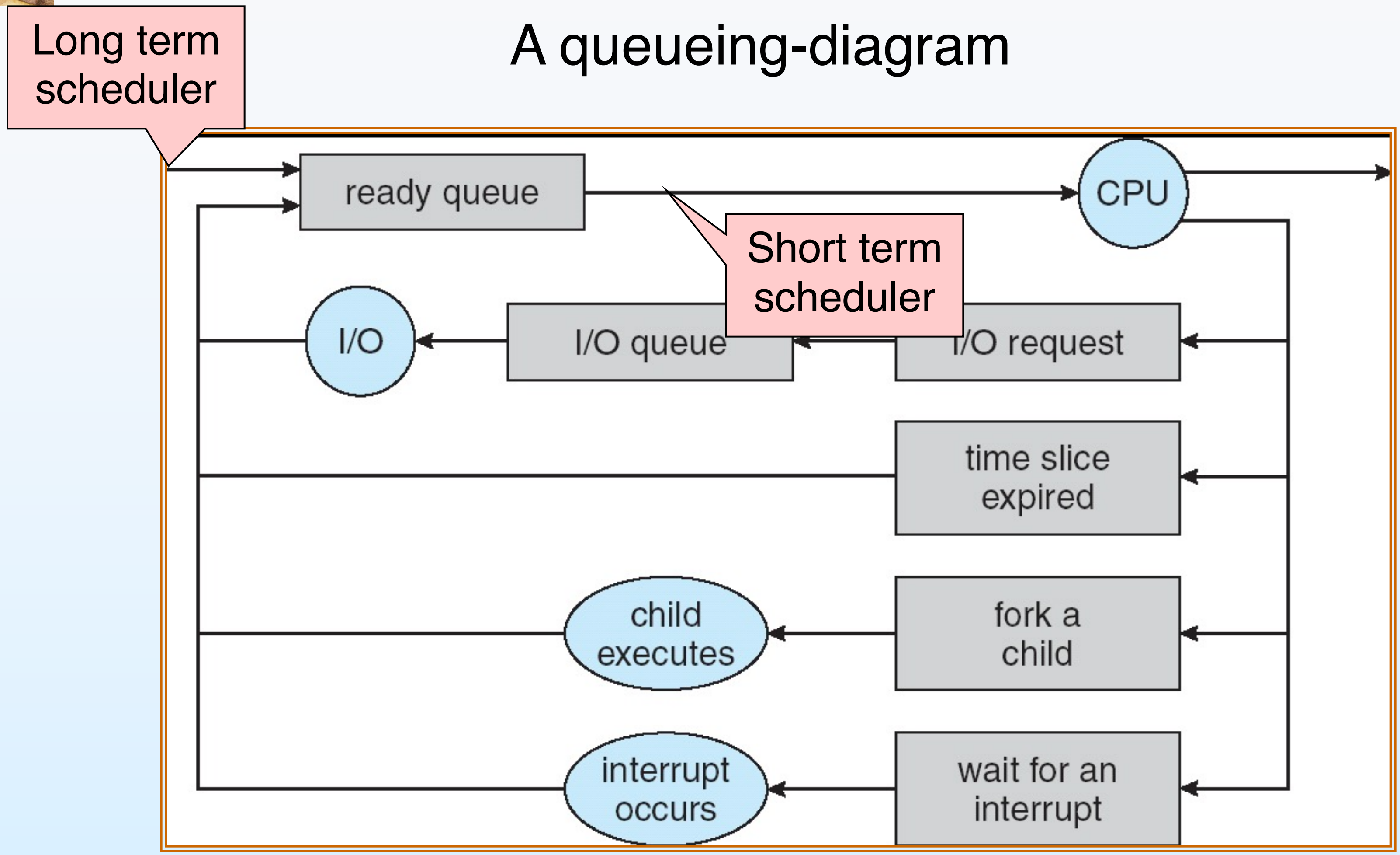
Schedulers

- What is a scheduler? A piece of program
- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into memory (the ready queue)
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU

UNIX and Windows do not use long-term scheduling

Representation of Process Scheduling

A queueing-diagram



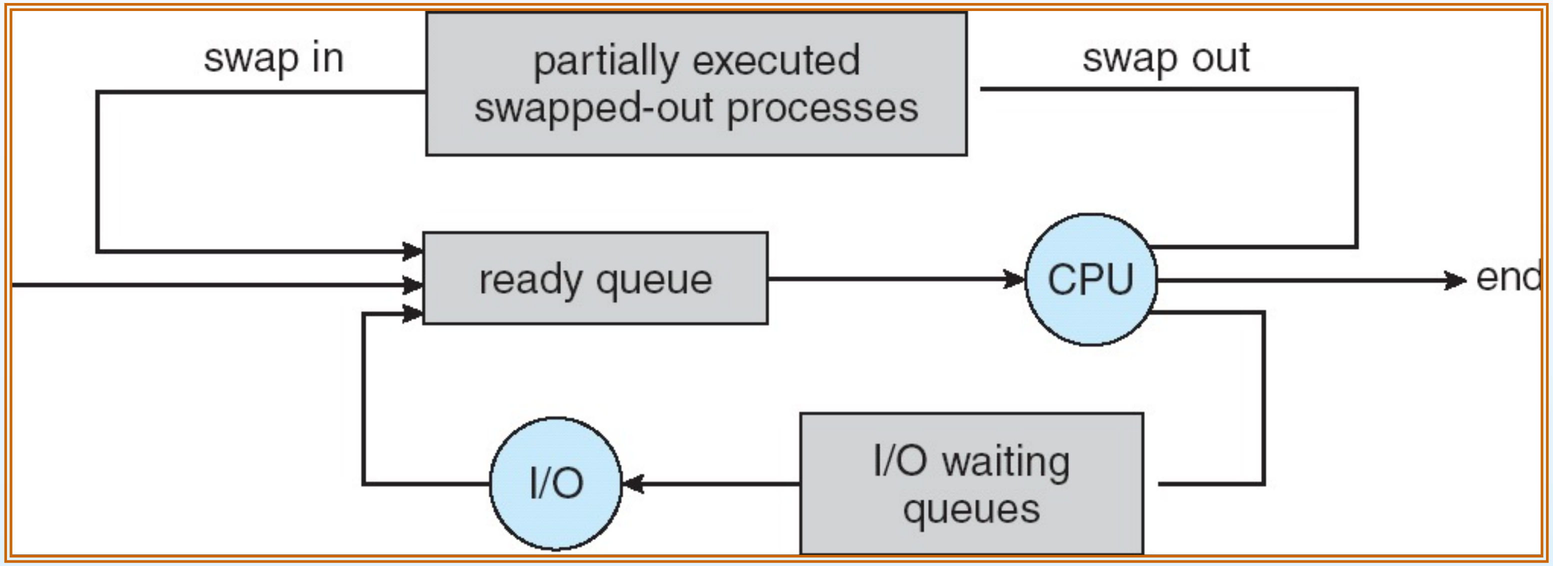
Representation of Process Scheduling

long term scheduling

short term scheduling

Addition of Medium Term Scheduling

Sometimes, it can be good to swap processes out.



Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
- The long-term scheduler controls the *degree of multiprogramming*
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts

Need to select good combination of CPU bound and I/O bound processes.

Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process
- Context-switch time is overhead; the system does no useful work while switching - typically takes milliseconds
- Time dependent on hardware support. In the SPARC architecture, groups of registers are provided.

Chapter 3: Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Cooperating Processes
- Interprocess Communication
- Communication in Client-Server Systems

Process Creation

- Parent process creates children processes, which, in turn create other processes, forming a tree of processes
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent’s resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate
- Example of Chrome

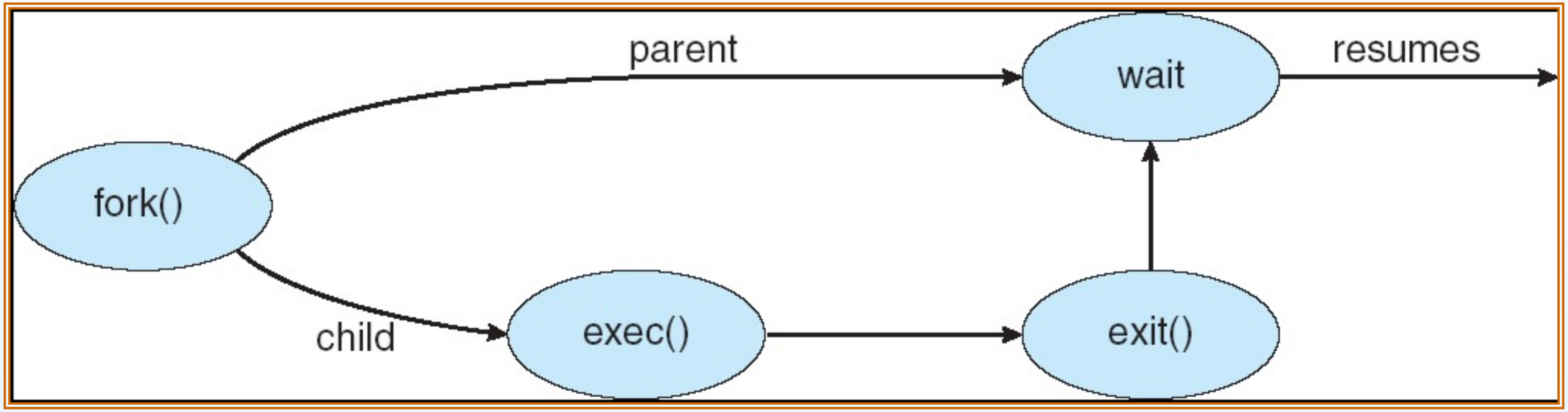
Process Creation (Cont.)

Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **fork** system call creates new process
  - **exec** system call used after a **fork** to replace the process' memory space with a new program

There are a lot “exec” APIs. For example: *execve() execv() execle() execvp() execlp()* etc.

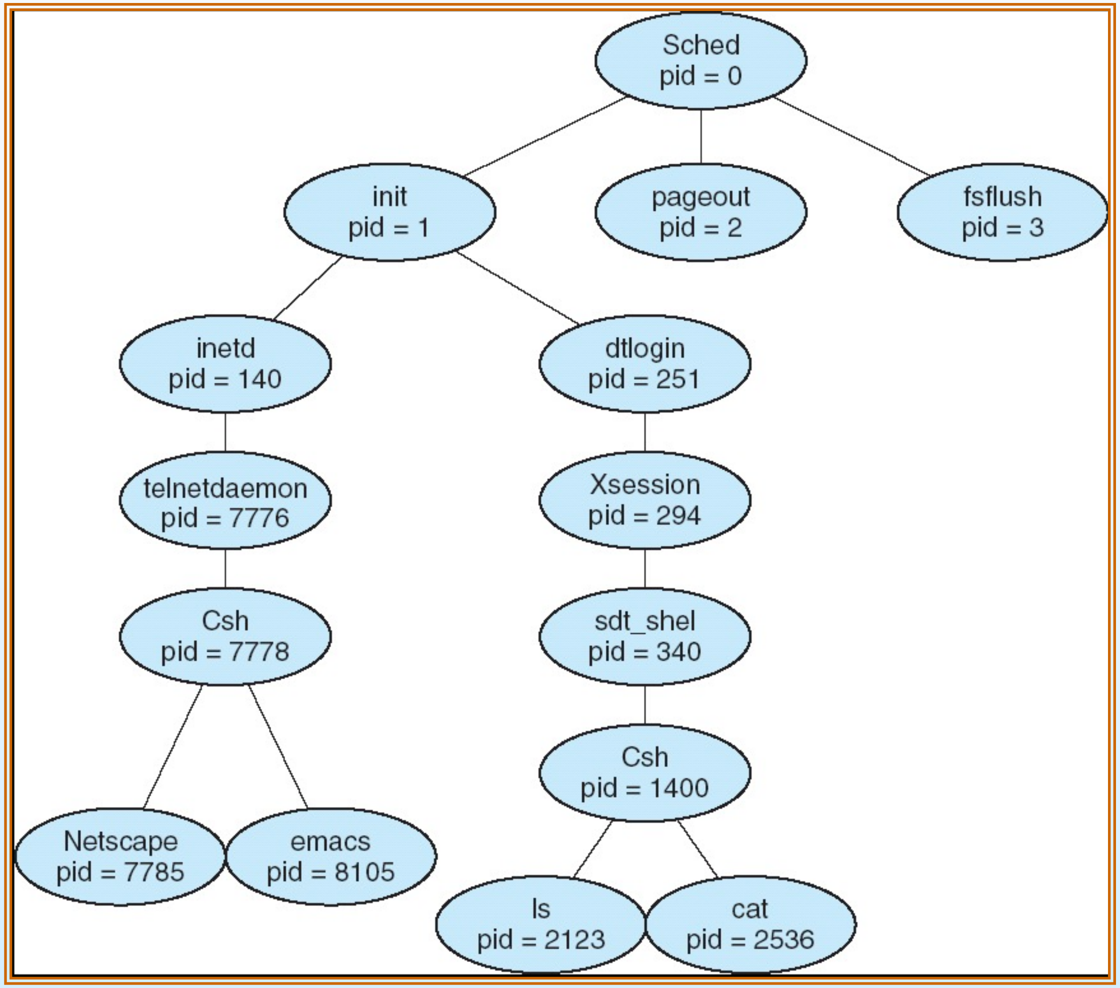
Process Creation



C Program Forking Separate Process

```
int main()
{
    pid_t  pid;
    /* fork another process */
    pid = fork();
    if (pid < 0) { /* error occurred */
          fprintf(stderr, "Fork Failed");
          exit(-1);
    }
    else if (pid == 0) { /* child process
    */
          execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
          /* parent will wait for the child
          to complete */
          wait (NULL);
          printf ("Child Complete");
          exit(0);
    }
}
```

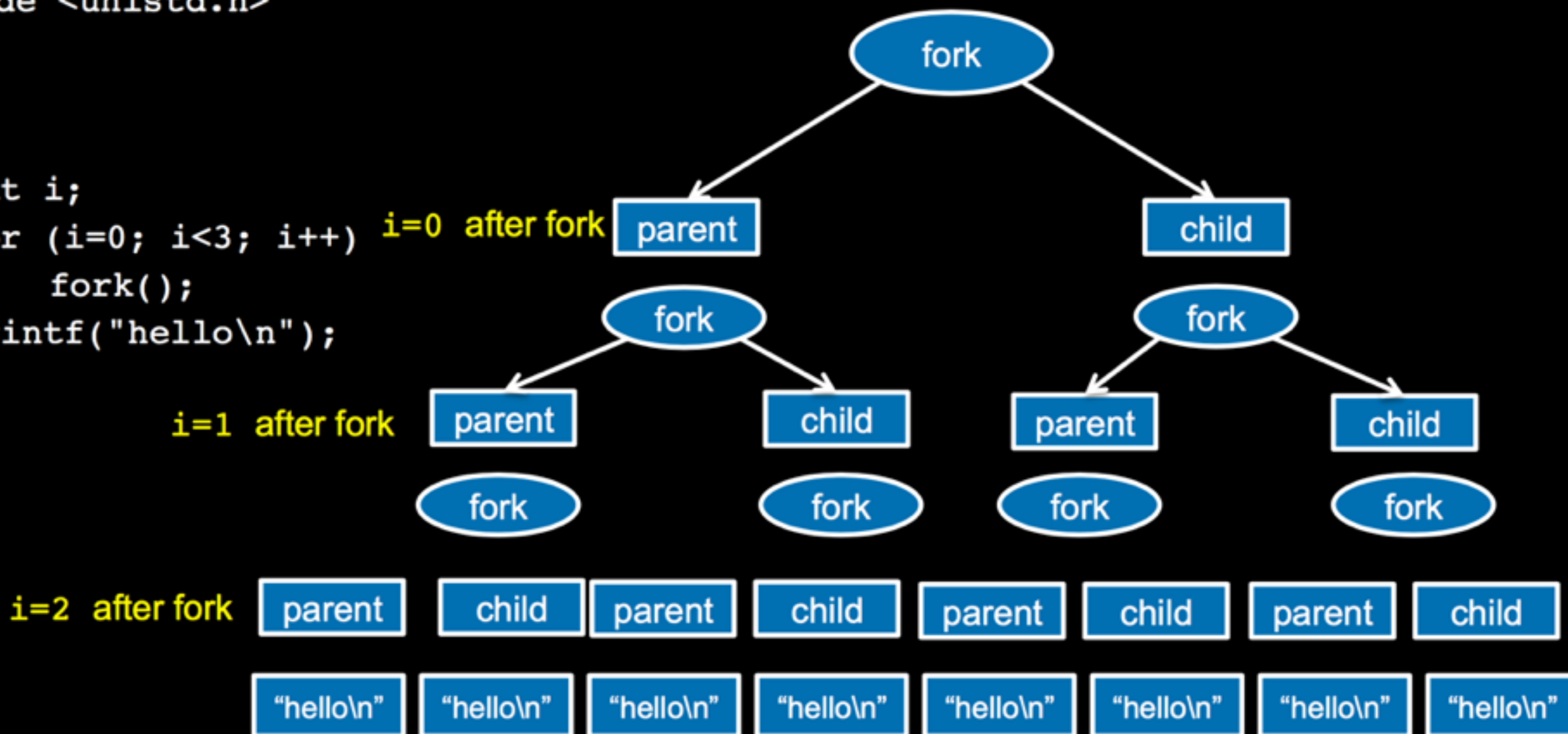
A tree of processes on a typical Solaris



C Program Forking Separate Process

```
#include <unistd.h>

main()
{
     int i;
     for (i=0; i<3; i++)
          fork();
     printf("hello\n");
}
```



Answer: 8

Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
  - Output data from child to parent (via **wait**)
  - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - ✓ Some operating system do not allow child to continue if its parent terminates
      - All children terminated - *cascading termination*
    - ✓ In some other operating systems, the child gets *orphaned*

Chapter 3: Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Cooperating Processes
- Interprocess Communication
- Communication in Client-Server Systems

Cooperating Processes

- **Independent** process cannot affect or be affected by the execution of another process
- **Cooperating** process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up (Multiple CPUs)
  - Modularity
  - Convenience

Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process
  - *unbounded-buffer* places no practical limit on the size of the buffer. Consumer has to wait if no new item.
  - *bounded-buffer* assumes that there is a fixed buffer size. Producer must wait if buffer full.

Bounded-Buffer – Shared-Memory Solution

- Shared data

```
#define BUFFER_SIZE 10
typedef struct {
    . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- Solution is correct, but ……

Bounded-Buffer – Insert() Method

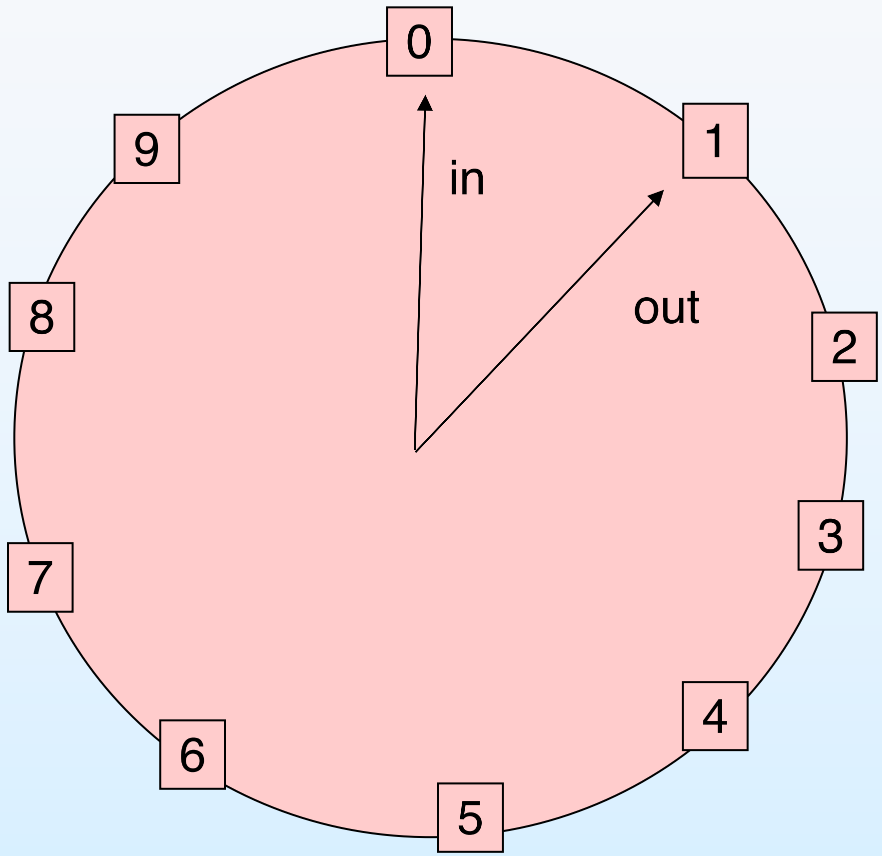
```
while (true) {
   /* Produce an item */
      while (((in + 1) % BUFFER_SIZE   ==
out)
     ;   /* do nothing -- no free buffers */
    buffer[in] = item;
    in = (in + 1) % BUFFER_SIZE;
   }
```

Bounded Buffer – Remove() Method

```
while (true) {
          while (in == out)
                   ; // do nothing --
nothing to consume

      // remove an item from the
buffer
      item = buffer[out];
      out = (out + 1) % BUFFER SIZE;
return item;
   }
```

Round Table



Chapter 3: Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Cooperating Processes
- Interprocess Communication
- Communication in Client-Server Systems

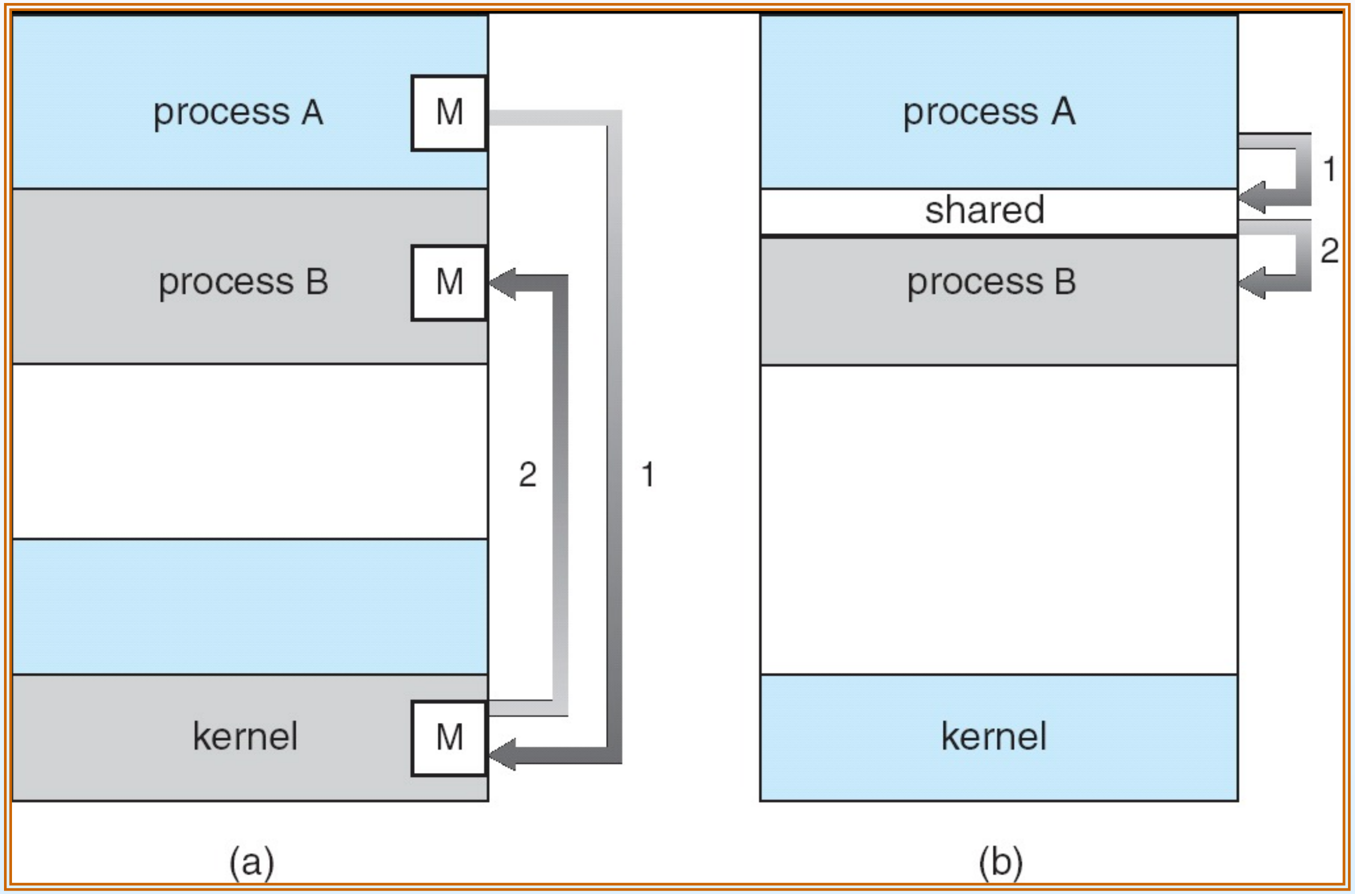
Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions
- Models for IPC: *message passing* and *shared memory*
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - **send**(*message*) – message size fixed or variable
  - **receive**(*message*)
- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

Communications Models



Message passing

Shared memory

Direct Communication

- Processes must name each other explicitly:
  - **send** (*P, message*) – send a message to process P
  - **receive**(*Q, message*) – receive a message from process Q
- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional

Indirect Communication

- Operations
  - create a new mailbox
  - send and receive messages through mailbox
  - destroy a mailbox
- Primitives are defined as:

  send(*A, message*) – send a message to mailbox A

  receive(*A, message*) – receive a message from mailbox A

Indirect Communication

- Mailbox sharing
  - P_1, P_2, and P_3 share mailbox A
  - P_1, sends; P_2 and P_3 receive
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most two processes
  - Allow only one process at a time to execute a receive operation
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

Synchronization

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
  - **Blocking send** has the sender block until the message is received
  - **Blocking receive** has the receiver block until a message is available
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking** send has the sender send the message and continue
  - **Non-blocking** receive has the receiver receive a valid message or null

Buffering

- Queue of messages attached to the link; implemented in one of three ways
  1. Zero capacity – 0 messages
     Sender must wait for receiver (rendezvous)
  2. Bounded capacity – finite length of *n* messages
     Sender must wait if link full
  3. Unbounded capacity – infinite length
     Sender never waits
- Control of Buffering

Chapter 3: Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Cooperating Processes
- Interprocess Communication
- Communication in Client-Server Systems

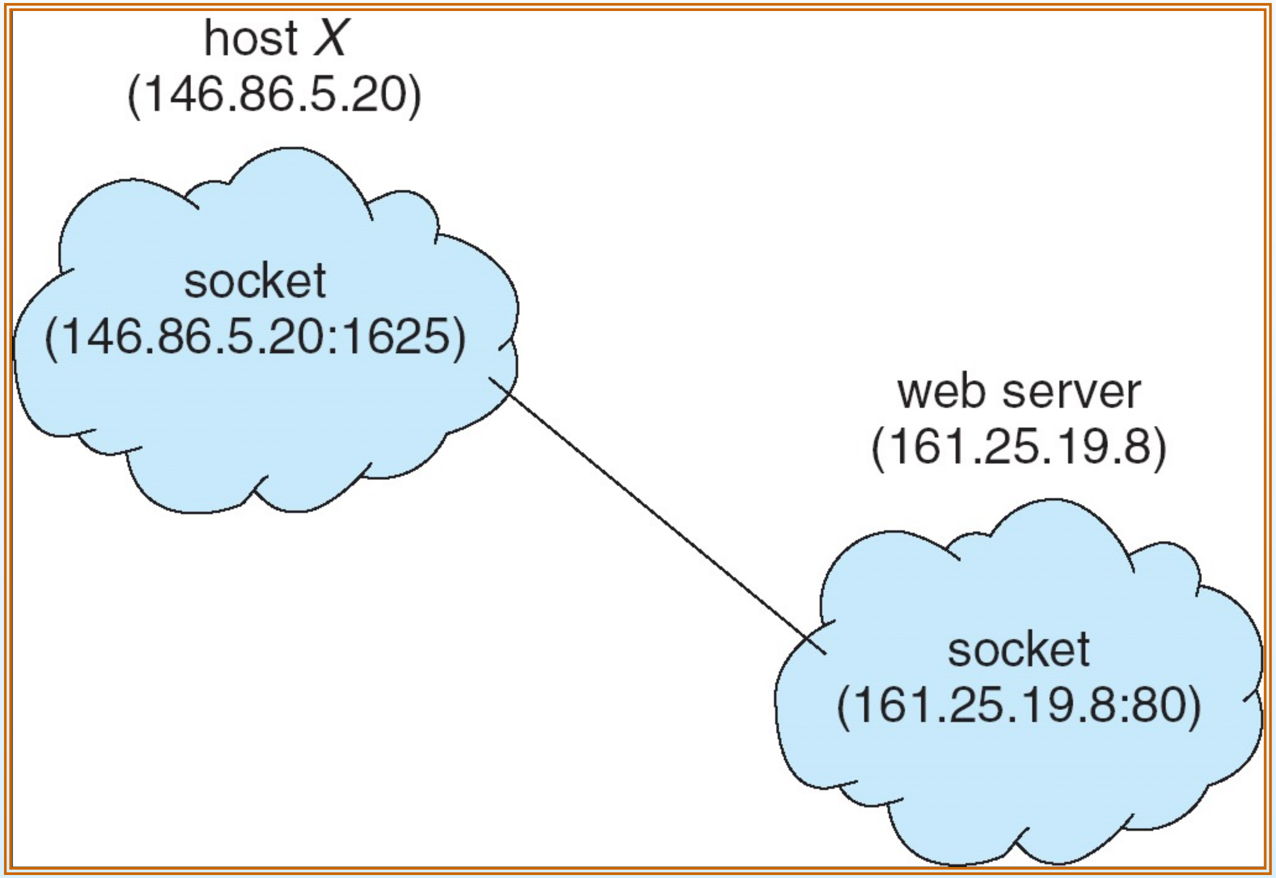
Client-Server Communication

- Sockets
- Remote Procedure Calls
- Remote Method Invocation (Java)

Sockets

- A socket is defined as an *endpoint for communication*
- Concatenation of IP address and port
- The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- Communication consists between a pair of sockets

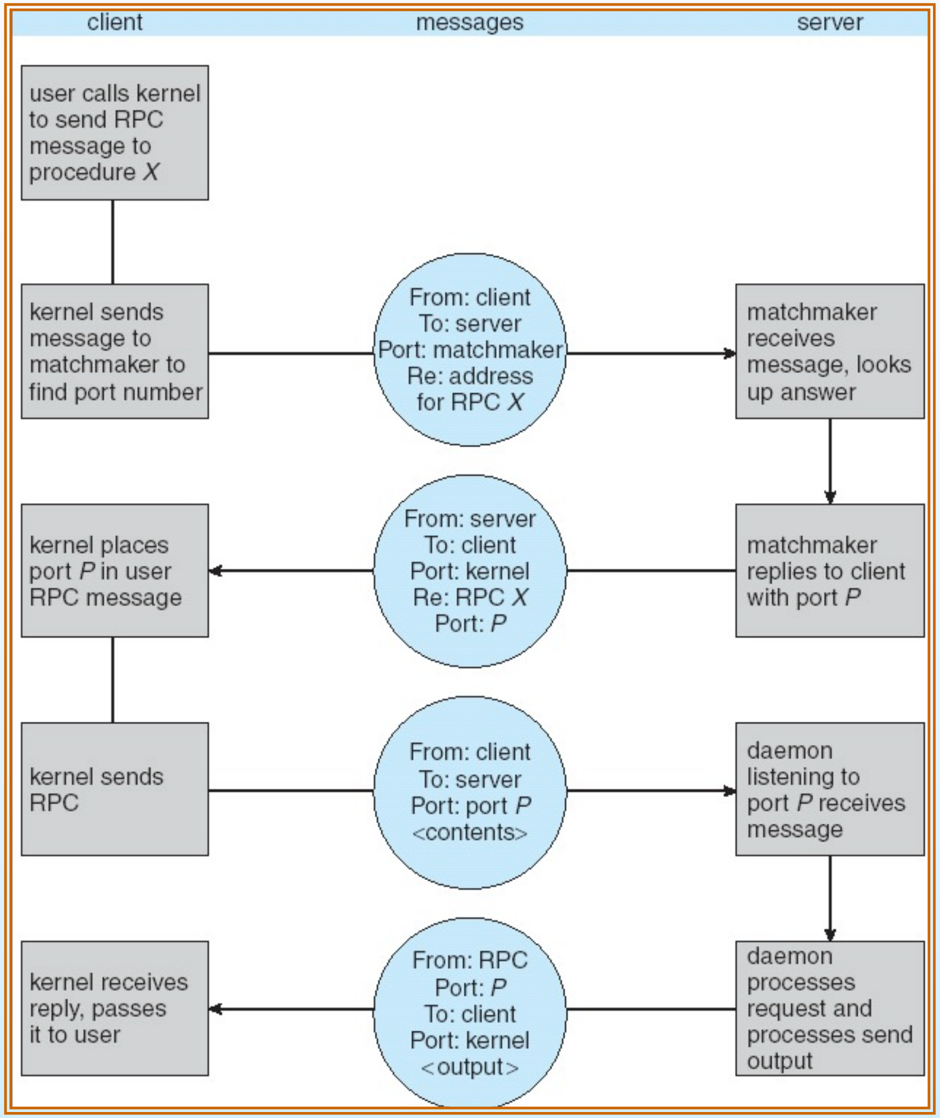
Socket Communication



Remote Procedure Calls

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems.
- **Stubs** – client-side proxy for the actual procedure on the server.
- The client-side stub locates the server and *marshals* the parameters.
- The server-side stub receives this message, unpacks the marshaled parameters, and performs the procedure on the server.

Execution of RPC



Apache Thrift

- Supported languages:
  - C GLib
  - C++ library
  - C# library
  - D library
  - Erlang library
  - Go library
  - Haskell library
  - Java library
  - JavaScript library
  - node.js library
  - OCaml library
  - Perl library
  - PHP library
  - Python library
  - Ruby library
  - SmallTalk library

Company	**Website**
Cloudera	http://www.cloudera.com
Evernote	http://evernote.com
Facebook	http://www.facebook.com
last.fm	http://www.last.fm
Mendeley	http://www.mendeley.com
OpenX	http://www.openx.org
RapLeaf	http://www.rapleaf.com
reCaptcha	http://www.recaptcha.com
Aereo	http://www.aereo.com
Siemens	http://www.siemens.com

Apache Thrift

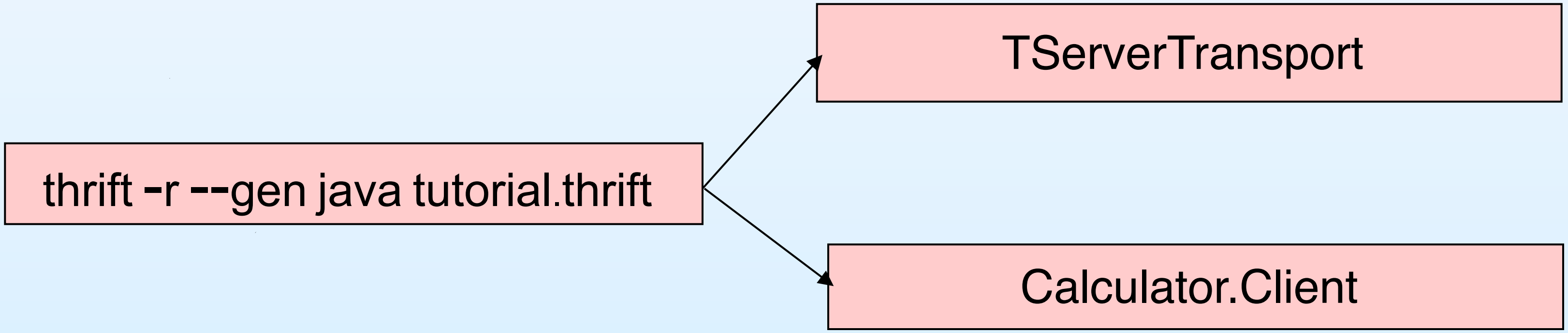
- service Calculator extends shared.SharedService {

  void ping(),

  i32 add(1:i32 num1, 2:i32 num2),

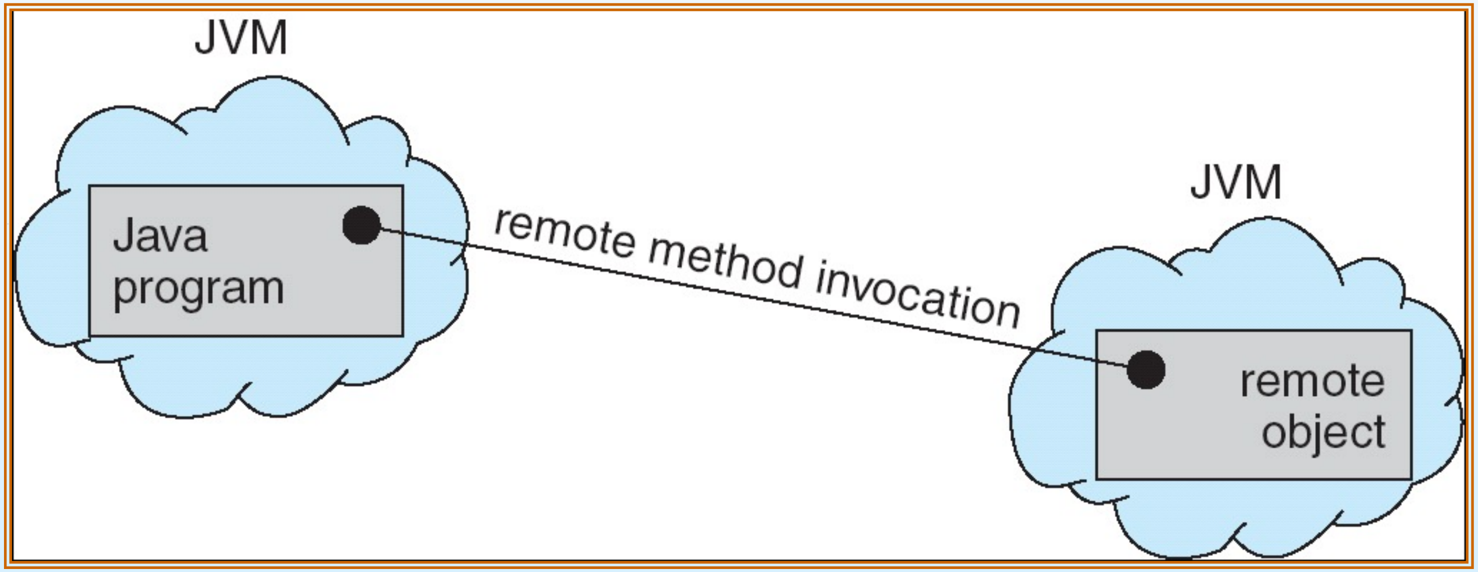
  i32 calculate(1:i32 logid, 2:Work w) throws (1:InvalidOperation ouch),

  void zip() }

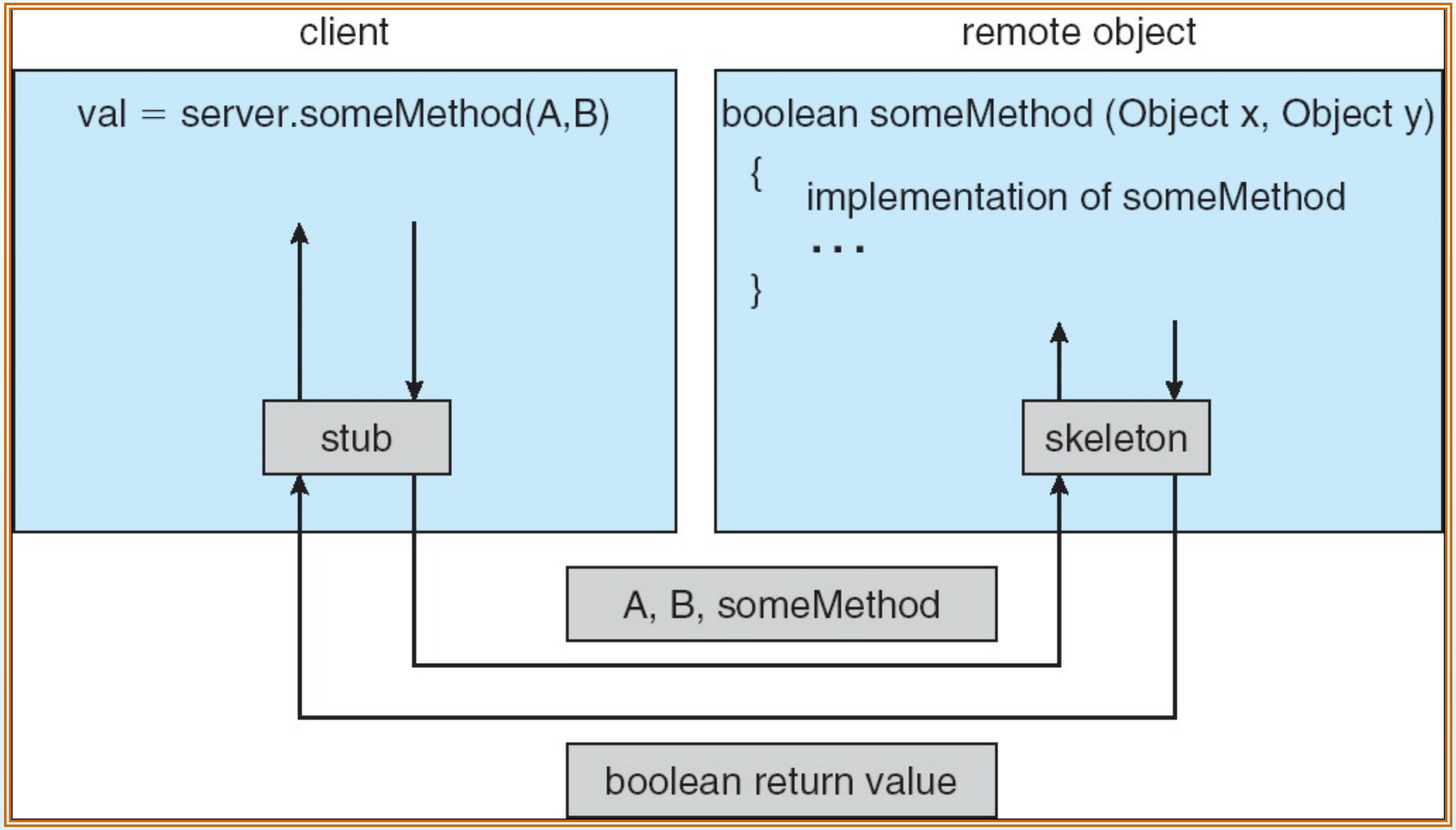


Remote Method Invocation

- Remote Method Invocation (RMI) is a Java mechanism similar to RPCs.
- RMI allows a Java program on one machine to invoke a method on a remote object.
- RMI is object-based



Marshalling Parameters



Since Java 2 v1.2, skeleton is not needed any more.

The picture can't be displayed.

End of Chapter 3