Chapter 3
Processes
we will completely ignore threads today

Images from Silberschatz

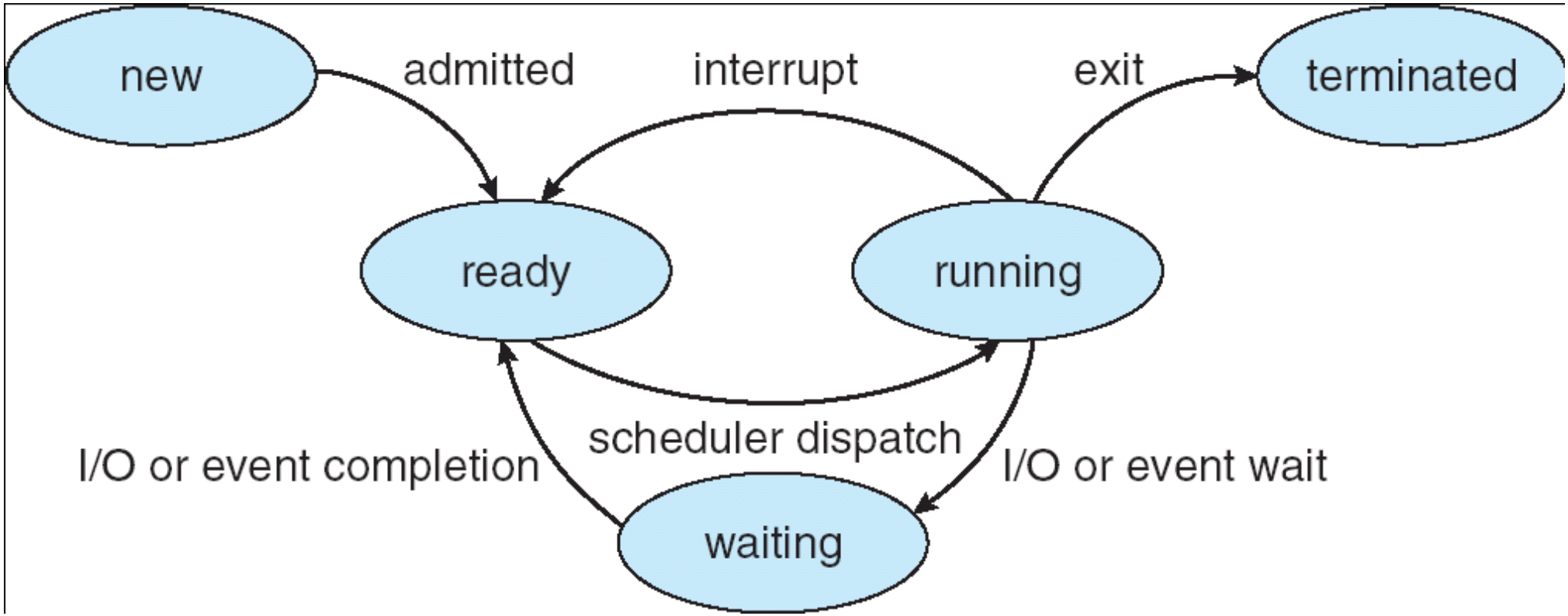
Process

- Define:

- Memory Regions:

- Loaded from executable file:
  - ELF: Executable and Linkable Format
    - Linux
    - What does this contain?

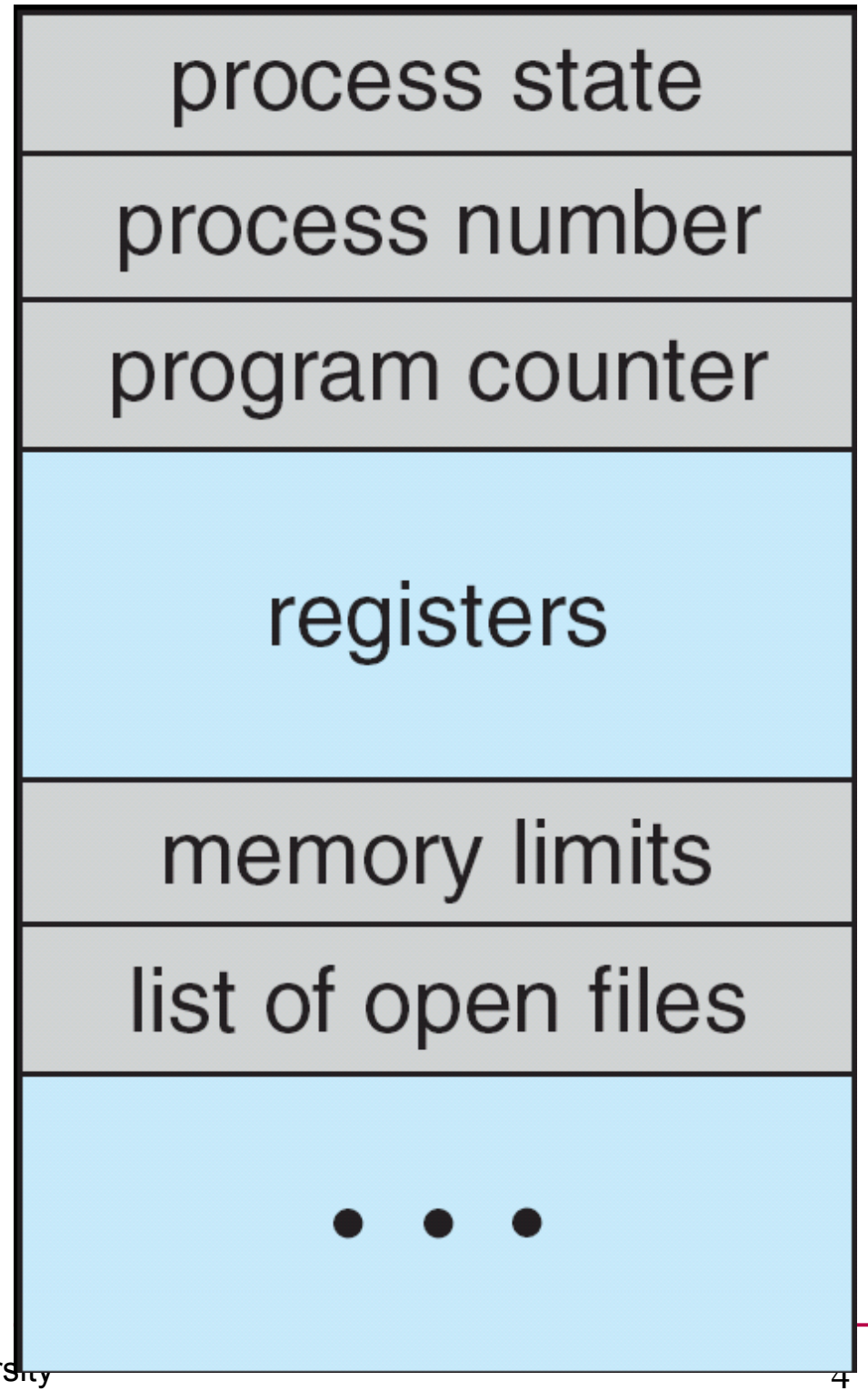
State Machine



- While a process is active it is in a particular state
- How many processes can be in each state?
- Data Structures? Where? Which kind? Why?

Process Control Block

- Who owns this data structure?
- CPU Scheduling data
- Memory Management data
- Accounting data



Types of Processes

- I/O Bound
- CPU Bound
- How does this affect the OS?

Process Scheduling

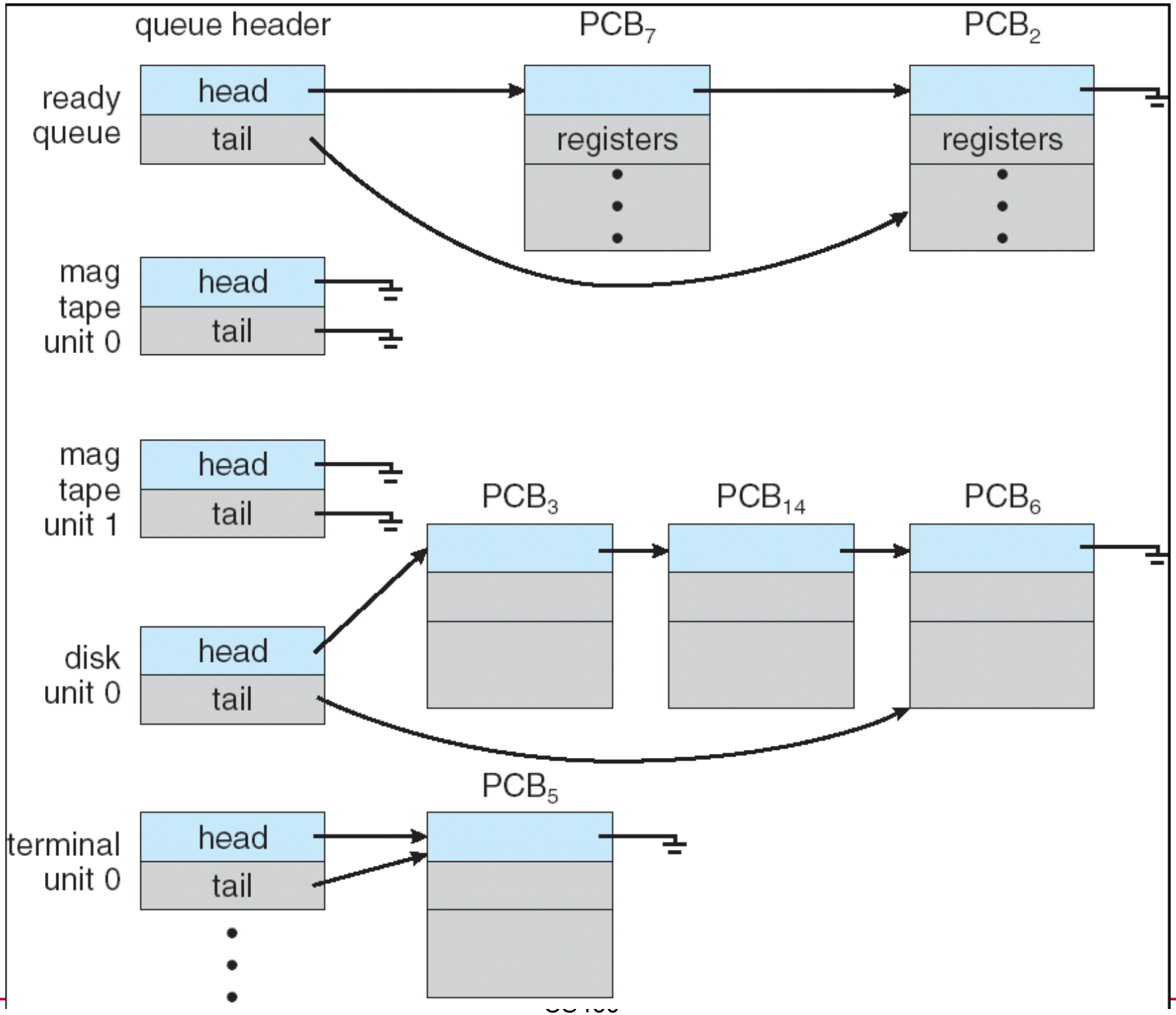
- Process Scheduler
  - Purpose:
  - Data structures:
  - Dispatched:

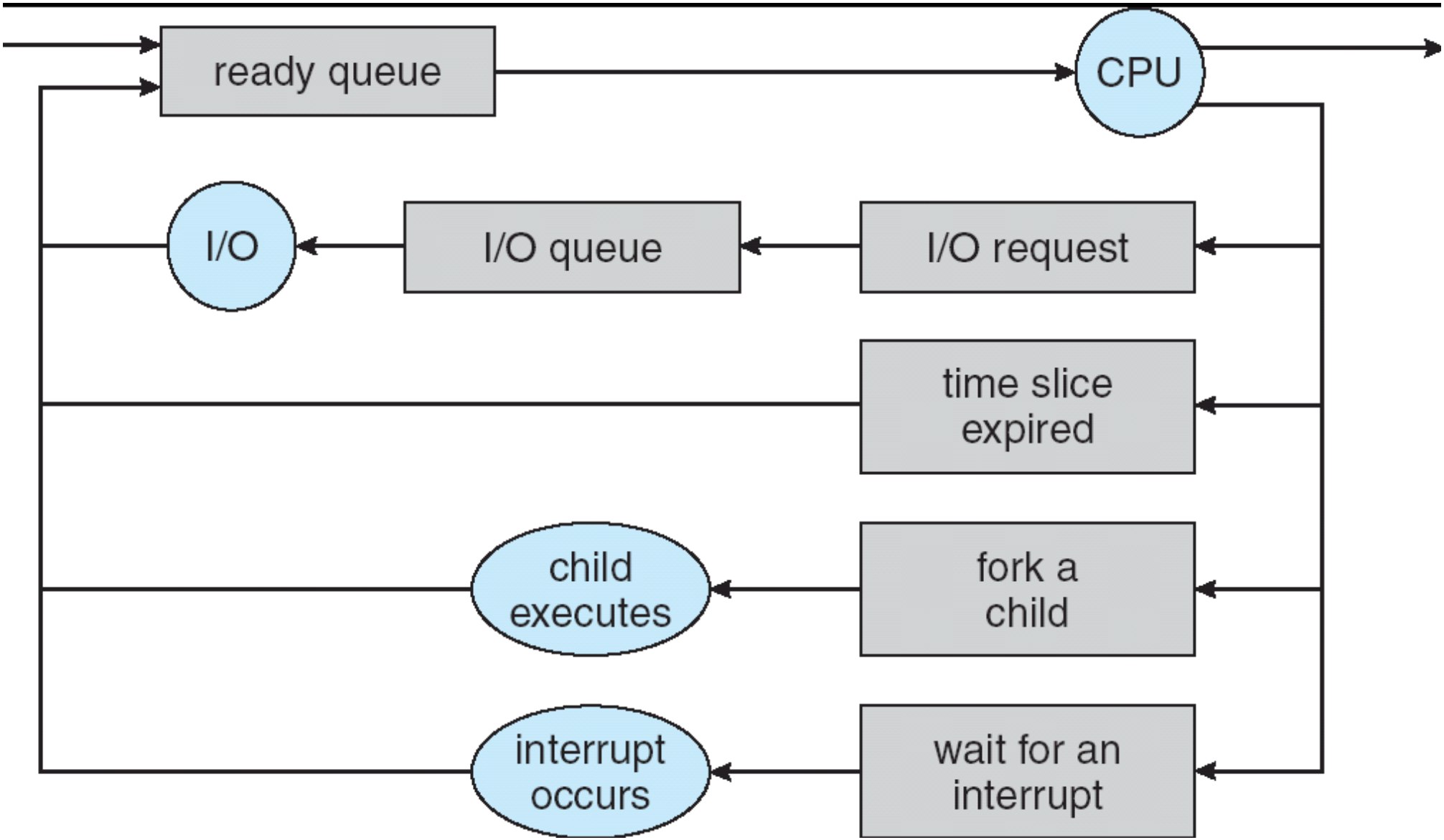
Schedulers

- Job Scheduler
  - Long term
  - Why is this important?

- CPU Scheduler
  - Short term
  - Constraints?

- Many OSes (Unix/Windows) don't really have a Job Scheduler


queue header
PCB7
PCB2
ready queue
head
tail
registers
registers
mag tape unit 0
head
tail
mag tape unit 1
head
tail
PCB3
PCB14
PCB6
disk unit 0
head
tail
PCB5
terminal unit 0
head
tail

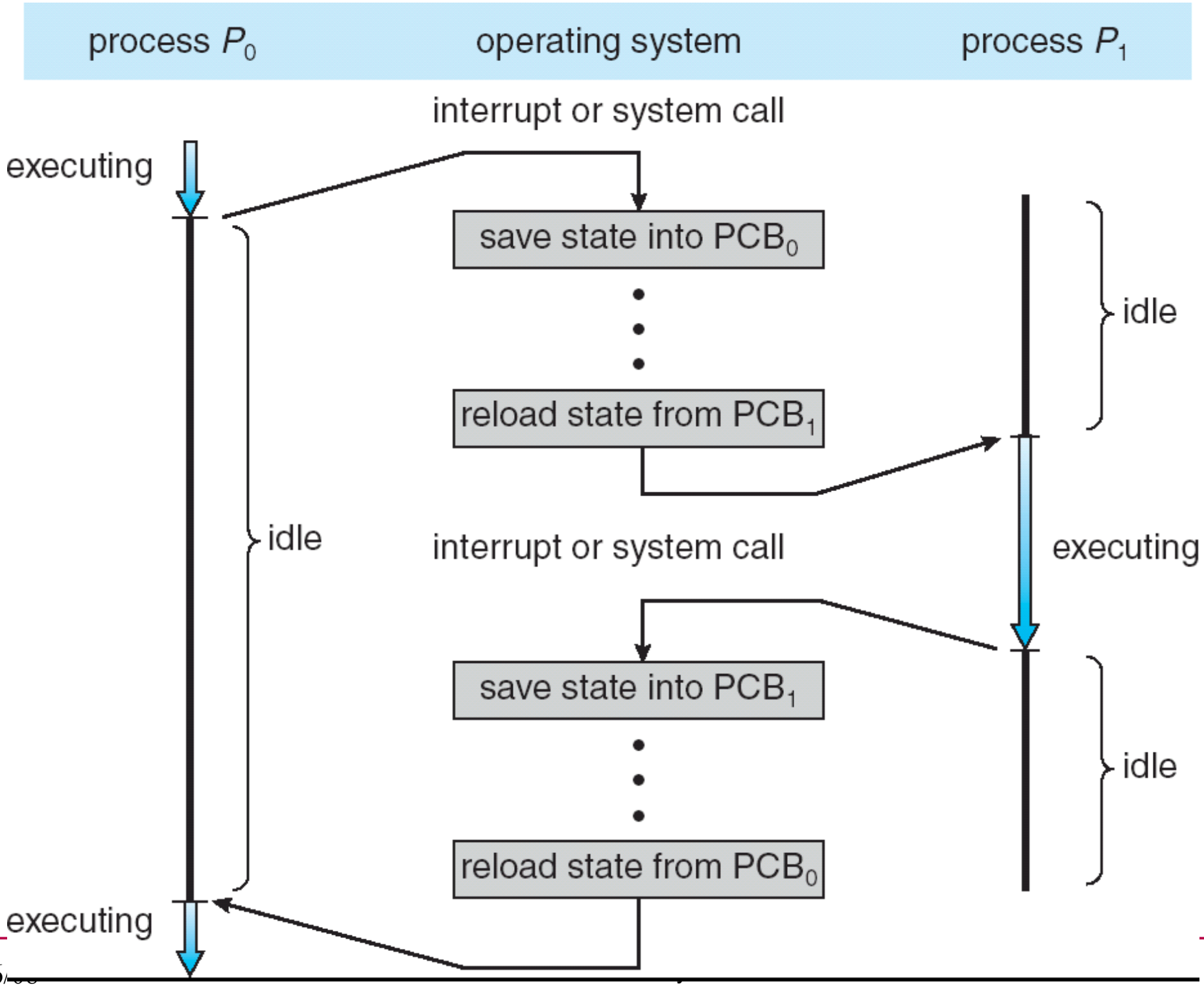

ready queue
CPU
I/O
I/O queue
I/O request
time slice expired
child executes
fork a child
interrupt occurs
wait for an interrupt

Context Switch

- Context:

- What happens during a Context Switch?

- Speed?


process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
reload state from PCB_1
idle
idle
interrupt or system call
executing
save state into PCB_1
reload state from PCB_0
idle
executing

Process Creation

```
/* This code works on Zeus! */
int main()
{
    pid_t  pid;
    int value = 0;
    value = 9;

    /* fork another process */
    pid = fork();
    fprintf(stderr,"The value: %d", value);

    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);
    }
}   /* page 92 of Silberschatz */
```

What happens if we put an fprintf() in side the block after the execlp()?

Process Termination

- kill(pid, signal)

  $ man kill

  $ ps u

  $ kill -9 pid

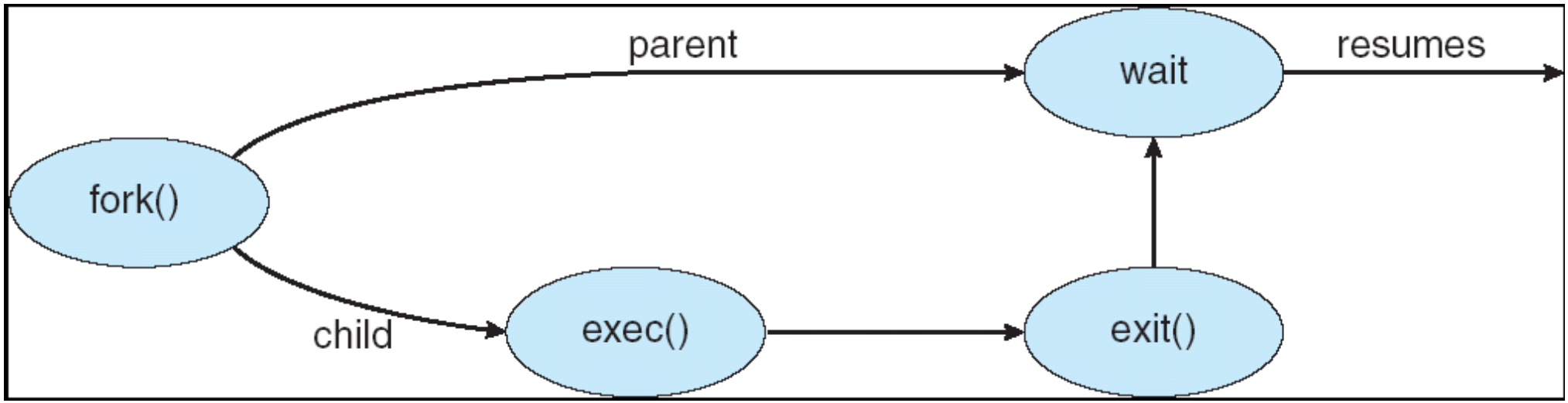
  $ man -s 2 kill

  $ man -s 7 signal

- Cascading termination:

Windows (Win32 API)

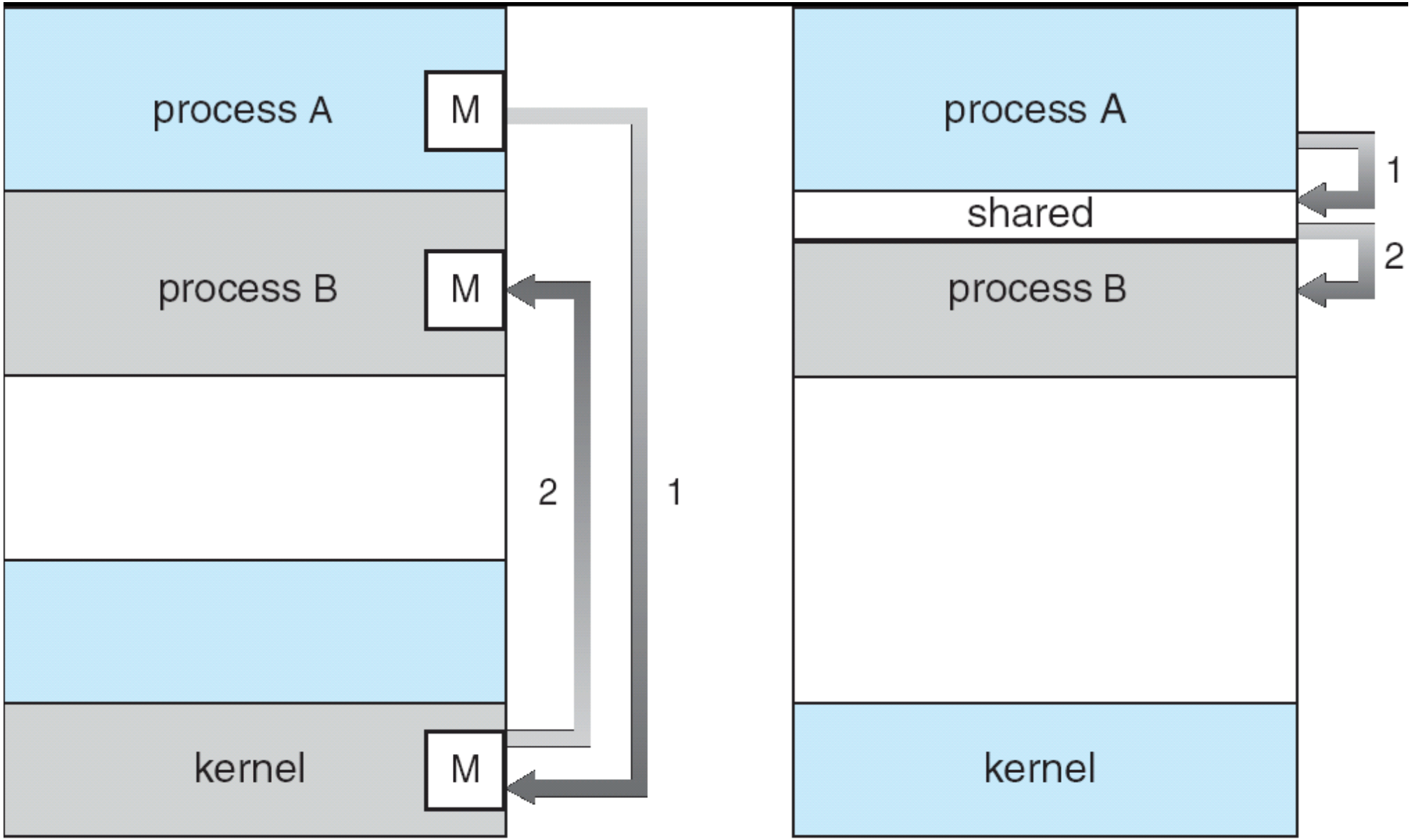
- CreateProcess()
  - fork() and exec() rolled into one
  - 10 parameters!
- WaitForSingleObject()
- TerminateProcess()


parent
wait
resumes
fork()
child
exec()
exit()

Interprocess Communication

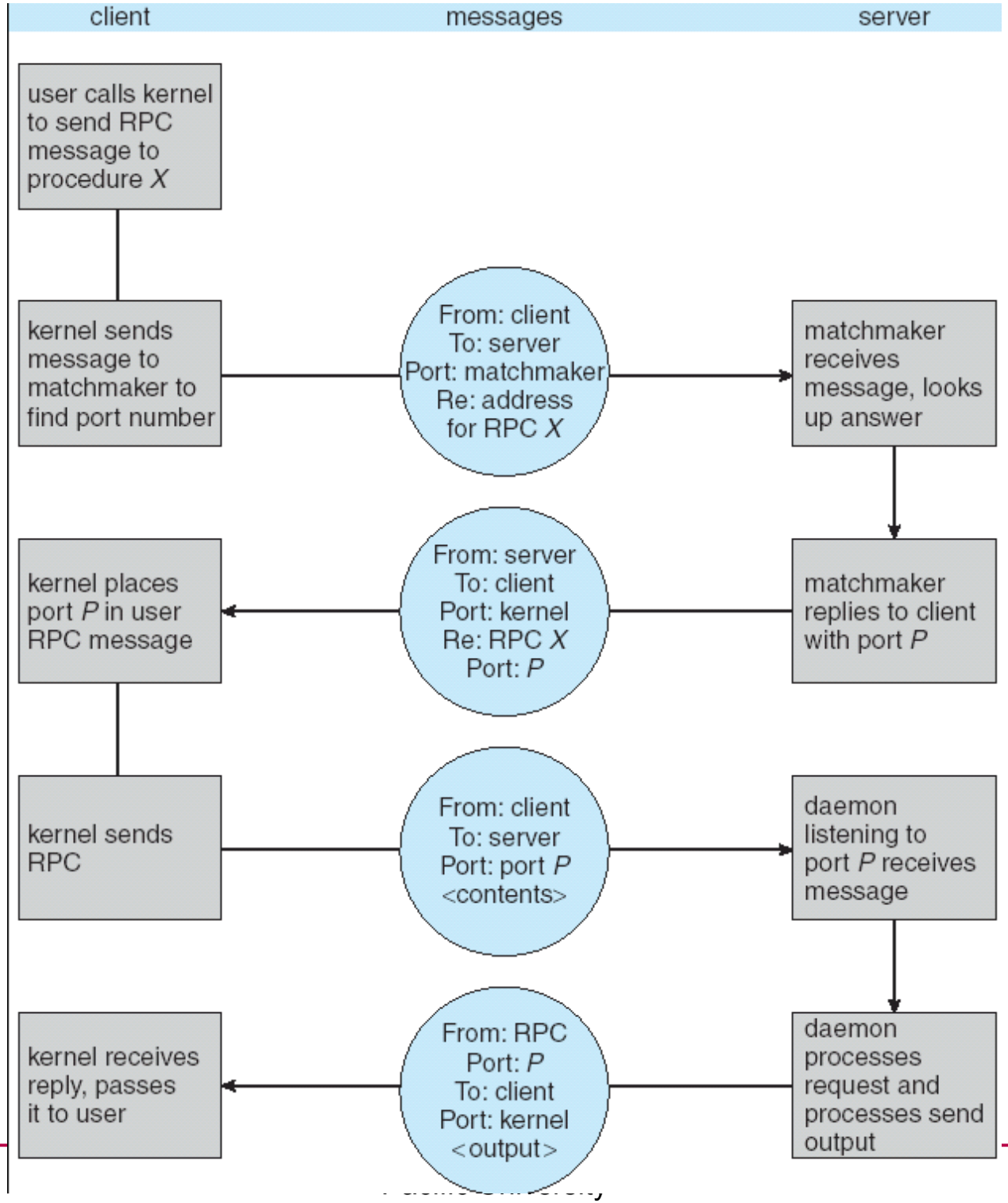
- Why do we want this?

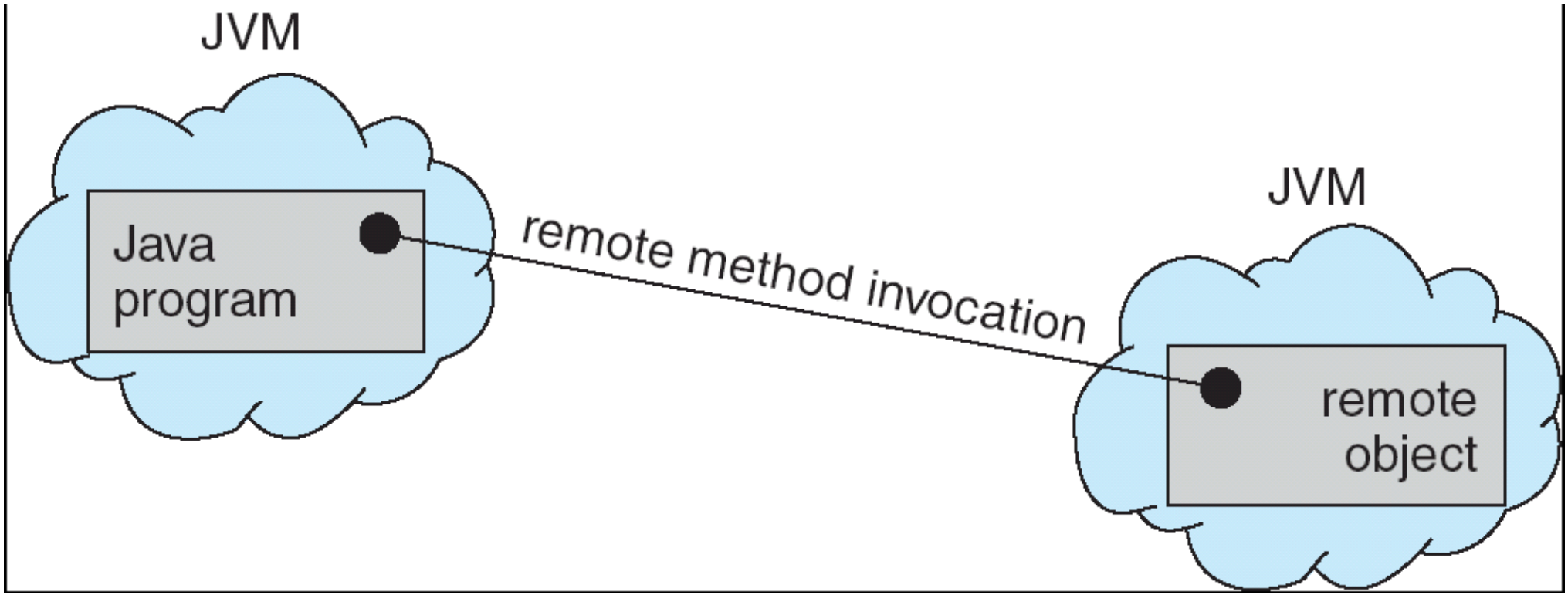
- Types:
  - Shared memory:
  - Message passing:


process A
M
process B
M
2
1
kernel
M
(a)
process A
1
shared
2
process B
kernel
(b)

Shared Memory

```
/* This code works on Zeus! */
int main()
{
   int segment_id;
   char *shared_memory;
   const int size = 4096;

   /* allocate shared memory segment */
   segment_id = shmget(IPC_PRIVATE, size, S_IRUSR | S_IWUSR);

   /* attach the shared memory segment */
   shared_memory = (char*) shmat (segment_id, NULL, 0);

   /* write a message to the shared memory segment */
   sprintf(shared_memory, "Hi there!");

   /* now print out the string from shared memory */
   printf("*%s\n", shared_memory);

   /* now detach the shared memory segment */
   shmdt(shared_memory);

   /* now remove the shared memory segment */
   shmctl(segment_id, IPC_RMID, NULL);
}
/* page 104 of Silberschatz */
```


client
messages
server
user calls kernel to send RPC message to procedure X
kernel sends message to matchmaker to find port number
From: client
To: server
Port: matchmaker
Re: address for RPC X
matchmaker receives message, looks up answer
kernel places port P in user RPC message
From: server
To: client
Port: kernel
Re: RPC X
Port: P
matchmaker replies to client with port P
kernel sends RPC
From: client
To: server
Port: port P
<contents>
daemon listening to port P receives message
kernel receives reply, passes it to user
From: RPC
Port: P
To: client
Port: kernel
<output>
daemon processes request and processes send output


JVM
Java
program
remote method invocation
JVM
remote
object

Functions

```
int execl(const char *path, const char *arg, ...)

int execlp(const char *file, const char *arg, ...)

int execle(const char *file, const char *arg, ...,
   char const* envp[])

int execv(const char *path, char *const argv[])

int execvp(const char *file, char *const argv[])

int dup2(int oldfd, int newfd)

int pipe(int filedes[2])

pid_t waitpid(pid_t pid, int *status, int options)

char* strtok_r(char *str, const char* delim, char **saveptr)
```

dup2()

```
#define MAXLEN 1024

/* This code works on Zeus! */
int main()
{
   /* dup2(oldfd, newfd)makes newfd be the copy of oldfd,
    * closing newfd first if necessary.
    */

   char data[MAXLEN];
   int fd;

   fd = open("test.txt", O_WRONLY | O_CREAT | O_TRUNC, S_IRWXU);
   dup2(fd, STDOUT_FILENO);

   fprintf(stderr,"> ");
   fgets(&(data[0]),MAXLEN,stdin);

   write(fd, &(data[0]), strlen(data));

   printf("%s\n", data); /* print to stdout */

   close(fd);

   printf("ONCE MORE: %s\n", data);/* print to stdout */
}
```

simple pipe()

```
#define MAXLEN 1024
#define READ 0
#define WRITE 1

/* This code works on Zeus! */
int main()
{
    char dataPipeWrite[MAXLEN];
    char dataPipeRead[MAXLEN];
    int thePipe[2];
    pid_t childPid;

    memset(&(dataPipeWrite[0]), '\0', MAXLEN);
    memset(&(dataPipeRead[0]), '\0', MAXLEN);

    pipe(thePipe);
                                        /* get data from user */
    readFromCommandLine(dataPipeWrite, MAXLEN);

    write(thePipe[WRITE], &(dataPipeWrite[0]), strlen(dataPipeWrite));
    read(thePipe[READ], &(dataPipeRead[0]), MAXLEN);

    fprintf(stderr,"READ FROM PIPE: %s\n",&(dataPipeRead[0]));

    close(thePipe[WRITE]);
    close(thePipe[READ]);
}
```

pipe()

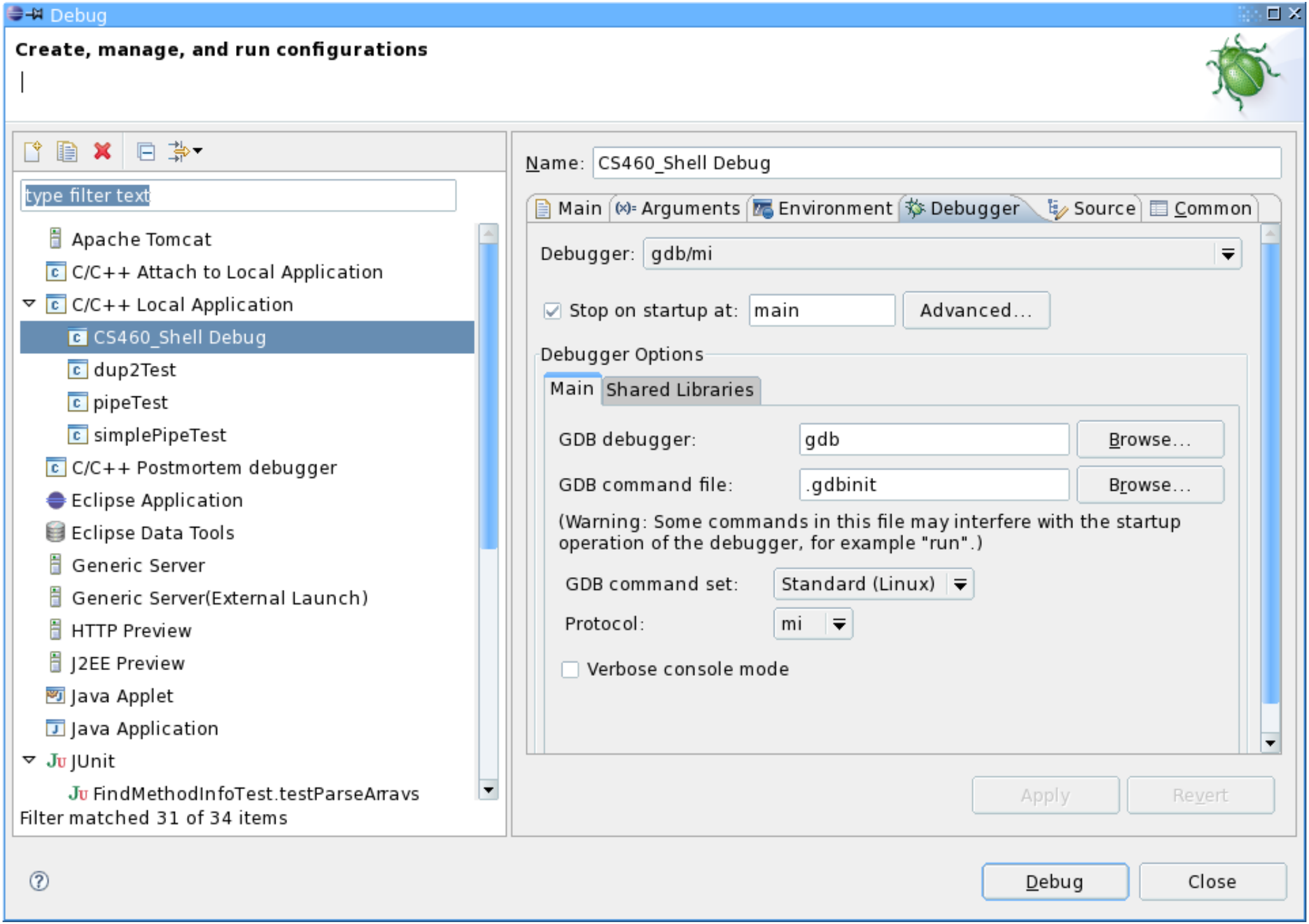
```
void readFromCommandLine(char * data, int maxSize)
{
    fprintf(stderr,"> ");
    fgets(data, maxSize, stdin);
}

/* This code works on Zeus! */
int main()
{
  /* pipe(int filedes[2])  creates  a  pair of file
   * descriptors, pointing to a pipe inode, and places
   * them in the array pointed to by filedes.
   * filedes[0] is for reading,
   * filedes[1] is for writing.
   */

    char data[MAXLEN];
    int thePipe[2];
    pid_t childPid;

    memset(&(data[0]), '\0', MAXLEN);

    pipe(thePipe);

    childPid = fork();
```

pipe()

```
if(childPid == 0)
{
    /* I AM A CHILD */
    close(thePipe[WRITE]);
    read(thePipe[READ], &(data[0]), MAXLEN);

    while(strncmp( &(data[0]), "STOP", 4) != 0 )
    {
        printf("CHILD> %s\n", &(data[0]));
        read(thePipe[READ], &(data[0]), MAXLEN);
    }
    close(thePipe[READ]);
}
else
{
    close(thePipe[READ]);

    readFromCommandLine(&(data[0]), MAXLEN);
    write(thePipe[WRITE], &(data[0]), strlen(data));

    while(strncmp(&(data[0]), "STOP", 4) != 0)
    {
        readFromCommandLine(&(data[0]), MAXLEN);
        write(thePipe[WRITE], &(data[0]), strlen(data));
    }
    close(thePipe[WRITE]);
}
```

Eclipse, fork(), and gdb

- Create a Debug Profile
- Set the debugger to gdb/mi
- Add the line: **set follow-fork-mode child** to .gdbinit
- Set the standard
- Set protocol mi


Debug
Create, manage, and run configurations
type filter text
Apache Tomcat
C/C++ Attach to Local Application
C/C++ Local Application
CS460_Shell Debug
dup2Test
pipeTest
simplePipeTest
C/C++ Postmortem debugger
Eclipse Application
Eclipse Data Tools
Generic Server
Generic Server(External Launch)
HTTP Preview
J2EE Preview
Java Applet
Java Application
JUnit
FindMethodInfoTest.testParseArravs
Filter matched 31 of 34 items
Name: CS460_Shell Debug
Main
Arguments
Environment
Debugger
Source
Common
Debugger: gdb/mi
Stop on startup at: main
Advanced...
Debugger Options
Main
Shared Libraries
GDB debugger: gdb
Browse...
GDB command file: .gdbinit
Browse...
(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)
GDB command set: Standard (Linux)
Protocol: mi
Verbose console mode
Apply
Revert
Debug
Close

valgrind

```
#include <stdio.h>

int main()
{
    char * myPtr;

    memcpy(myPtr, "HELLO", strlen("HELLO"));
}
```

valgrind

```
[zeus]$ valgrind ./valgrindTest
==17250== Use of uninitialised value of size 8
==17250==    at 0x400470: main (valgrindTest.c:7)


==17250== Invalid write of size 4
==17250==    at 0x400470: main (valgrindTest.c:7)
==17250==  Address 0x0 is not stack'd, malloc'd or (recently) free'd

==17250== Process terminating with default action of signal 11
==17250==  Access not within mapped region at address 0x0
==17250==    at 0x400470: main (valgrindTest.c:7)


==17250== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 2
from 1)
==17250== malloc/free: in use at exit: 0 bytes in 0 blocks.
==17250== malloc/free: 0 allocs, 0 frees, 0 bytes allocated.
==17250== For counts of detected errors, rerun with: -v
==17250== All heap blocks were freed -- no leaks are possible.
Segmentation fault
```