

---

# Processes

CSE 410, Spring 2004  
Computer Systems

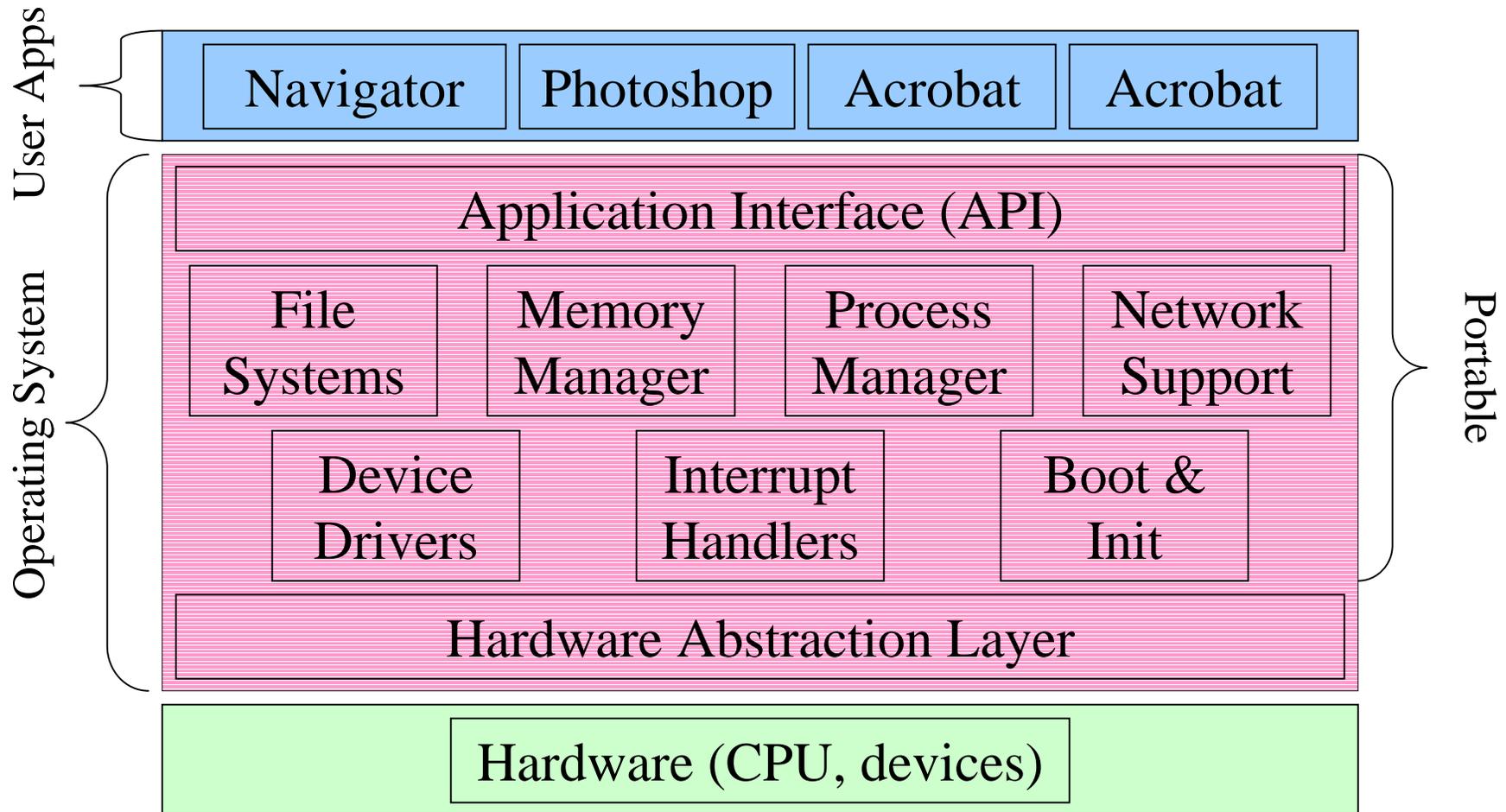
<http://www.cs.washington.edu/education/courses/410/04sp/>

# Reading and References

---

- Reading
  - » Chapter 4 through 4.5.4, *Operating System Concepts*, Silberschatz, Galvin, and Gagne
- Other References
  - » *Inside Microsoft Windows 2000*, Third Edition, Solomon and Russinovich

# Example OS in operation



# Programs and Processes

---

- A *program* is passive
  - » a file on disk with code that can be run
- A *process* is active
  - » an instance of a program in execution
  - » also called *job*, *task*, *sequential process*
- There are always many processes running
- Some may be running the same program
  - » but they are still separate and independent processes

# What are the parts of a process?

---

- code for the running program
- data for the running program
  - » heap, stack
- location of the next instruction (PC)
- current state of the general-purpose registers
- list of open resources
  - » files, network connections
- lots of OS management data

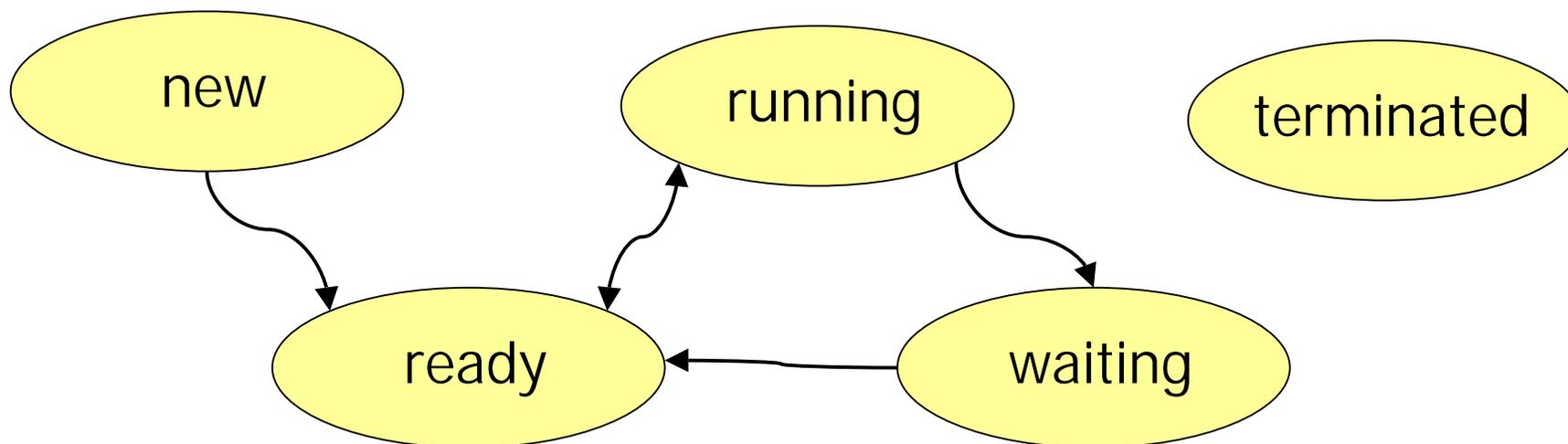
# Process State

---

- Each process has an execution state that indicates what it is currently doing:
  - » **ready**: waiting to be assigned to the CPU
  - » **running**: executing instructions on the CPU
  - » **waiting**: waiting for an event, e.g., I/O completion, so that it can be made ready
- As a program executes, the OS moves the process from state to state

# Process State Changing

---



Processes move from state to state as a result of actions they perform (e.g., system calls), OS actions (rescheduling) and external actions (interrupts)

# Process Data Structures

---

- At any time, there are many processes active in a system
- The OS has data structures representing each process
  - » primary structure is the Process Control Block (PCB)
- PCB contains info about a process
  - » including pointers to other related data blocks

# PCBs and Hardware State

---

- When a process runs, its PC, SP, and registers, are loaded on the CPU
- When the OS switches to a new process, it
  - » saves the current process's register values to its PCB
  - » loads the next process's register values from its PCB
- This is called a **context switch**. It occurs 100-1000 times per second
  - » why so often?
  - » why not more often?

# Context switch is pure overhead

---

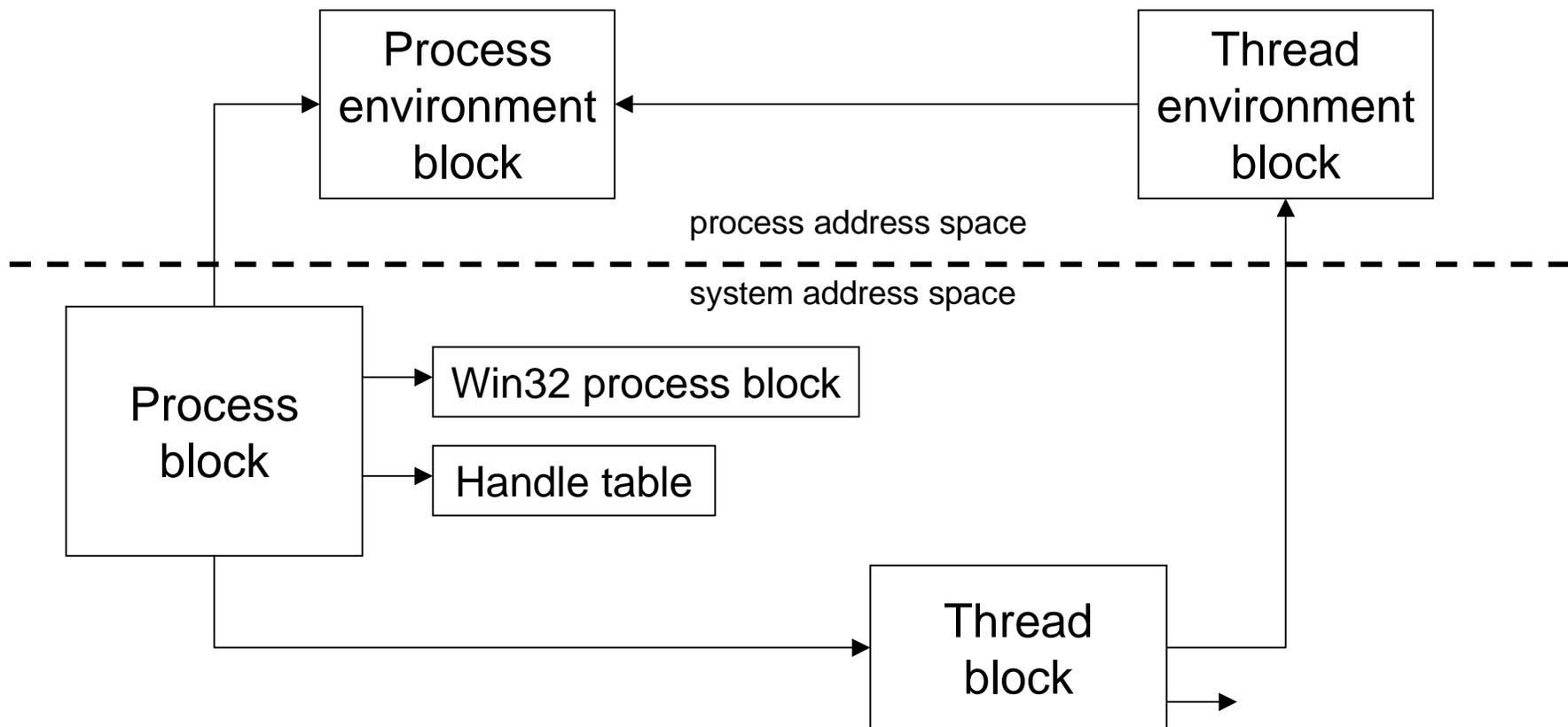
- Switching processes can be expensive
  - » register reload
  - » OS data structures
- Lightweight context reduces cost of switch
  - » threads
- Special hardware reduces cost of switch
  - » larger register files with register windows
  - » remember “load multiple register” instruction?

# Simple Process Control Block

---

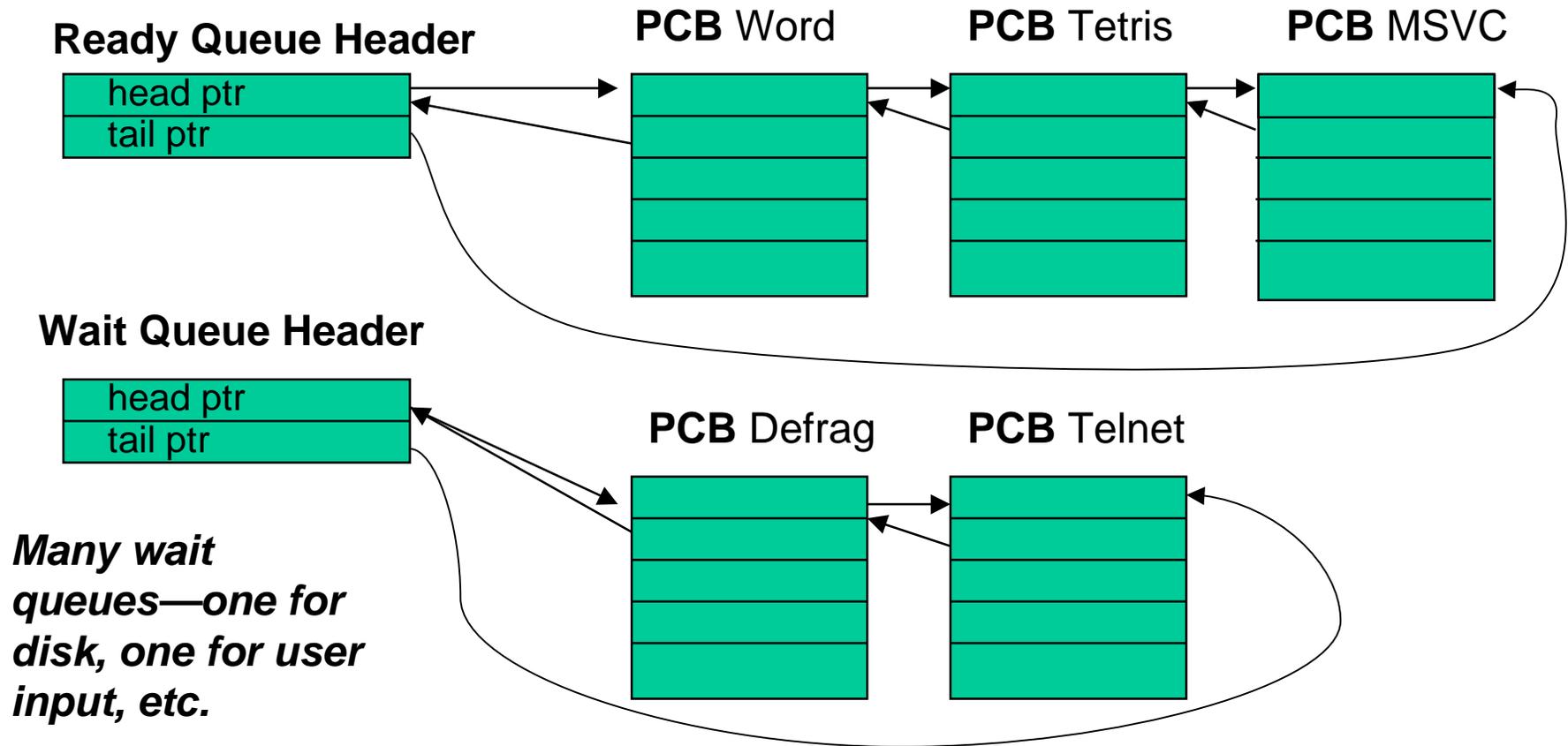
process state
process number
<b>program counter</b>
<b>stack pointer</b>
<b>32 general-purpose registers</b>
<b>memory management info</b>
username of owner
queue pointers for state queues
scheduling info (priority, etc.)
accounting info

# Simplified W2K Process Data



Copied from *Inside Windows 2000*

# Process State Queues



# PCBs and State Queues

---

- PCBs are data structures in OS memory
- A PCB is created for a process when it starts and put on the ready queue
- While the process is active, PCB is on one of the state queues
- When the process is terminated, its PCB is deallocated (*after a little while*)

# Getting control back

---

- How does the OS get control back from a running process?
  - » The process could explicitly return control to the OS (in many real-time systems)
  - » Generally, we can't trust the process to do this
- OS sets a timer on the CPU (privileged instruction) and starts a user process
- When the timer expires control passes to OS
  - » impact on “hard real-time” system?

# Scheduling a process

---

- Batch processes tend to be scheduled over a long period by a job scheduler
  - » explicit dollar value on priority
  - » longer time in CPU once loaded and started
- Interactive or soft real time processes are started as needed and compete for CPU
  - » dynamic priorities
  - » rapid context switching of many processes

# Creating a process

---

- The OS creates processes upon request
- The first few processes are all part of the operating system itself
  - » services, sessions, spoolers, network tools, ...
- Further processes created as response to login, user command, scheduled events
  - » winlogin, sshd, navigator, photoshop, ...

# create-process

---

- OS provides create-process system call
  - » parent process creates one or more children
  - » each child can create more children
  - » the result is a process tree
- Parent can wait or continue immediately
  - » create a process and block (synchronous)
  - » create a process and continue (asynchronous)

# “tlist -t” on my laptop

---

```
System Process (0)
System (8)
  smss.exe (140)
    csrss.exe (164)
      winlogon.exe (160) NetDDE Agent
        services.exe (212)
          svchost.exe (392)
            spoolsv.exe (420)
              Avsynmgr.exe (476)
                VsStat.exe (744) NAI_VS_STAT
                  Vshwin32.exe (760) VShieldWin_Class
                    Avconsol.exe (872)
                      svchost.exe (496)
                        HPConfig.exe (536) OleMainThreadWndName
                          regsvc.exe (580)
                            MSTask.exe (600) SYSTEM AGENT COM WINDOW
                              WinMgmt.exe (636)
                                mspmpsv.exe (724)
                                  Mcshield.exe (556)
                                    lsass.exe (224)
                                      Explorer.EXE (1076) Program Manager
                                        ESSD.exe (1132) ESS Daemon
                                          s3hotkey.exe (1160) S3HotKey
                                            S3trayhp.exe (1180) S3
                                              SynTPLpr.exe (1196) Touchpad driver helper
                                                SynTPEnh.exe (1228) Touchpad driver tray
                                                  motmon.exe (1220) motmon
                                                    mpbtn.exe (1256) hpisButton
                                                      CP32NBTN.EXE (1280) One-Touch
                                                        CDRomMnt.EXE (888) CD-Rom Monitor
                                                          KBOSDctl.EXE (1116) Dritek OSD Window
                                                            CP32NKCC.EXE (1264) Dritek HotKey
                                                              OSA.EXE (1308) Reminder
                                                                AcroTray.exe (1316) AcrobatTrayIcon
                                                                  CMD.EXE (984) Command Prompt - tlist -t
                                                                    tlist.exe (1112)
```

```

#include <stdio.h>
int main(int argc, char *argv[]) {
    int pid;
    int thisPid;
    thisPid = getpid();
    printf("Forking in (%i).\n", thisPid);
    pid = fork();
    if (pid < 0) {
        fprintf(stderr, "Fork Failed\n");
        exit(-1);
    }
    else if (pid == 0) {
        execlp("/bin/ls", "ls", NULL);
    }
    else {
        printf("Waiting in (%i) for (%i).\n", thisPid, pid);
        wait(NULL);
        printf("Child (%i) Complete.\n", pid);
        exit(0);
    }
}

```

# Fork Example

```

aspen $ gcc fork.c
aspen $ ./a.out
Forking in (20946).
Waiting in (20946) for (20947).
a.out fork.c fork.c~
Child (20947) Complete.

```

# W2K *CreateProcess* function

---

- Open the program file to be executed
- Create the W2K executive process object
- Create the initial thread (stack, context, ...)
- Notify Win32 subsystem about new process
- Start execution of the initial thread
- Complete initialization (eg, load dlls)
- Continue execution in both processes