Outline

- SHP1 : Assigned
- Course project info
- Chapters 1,2,3 & 4,5

Review

- What is an operating system?
  - Chapters 1, 2 and 3

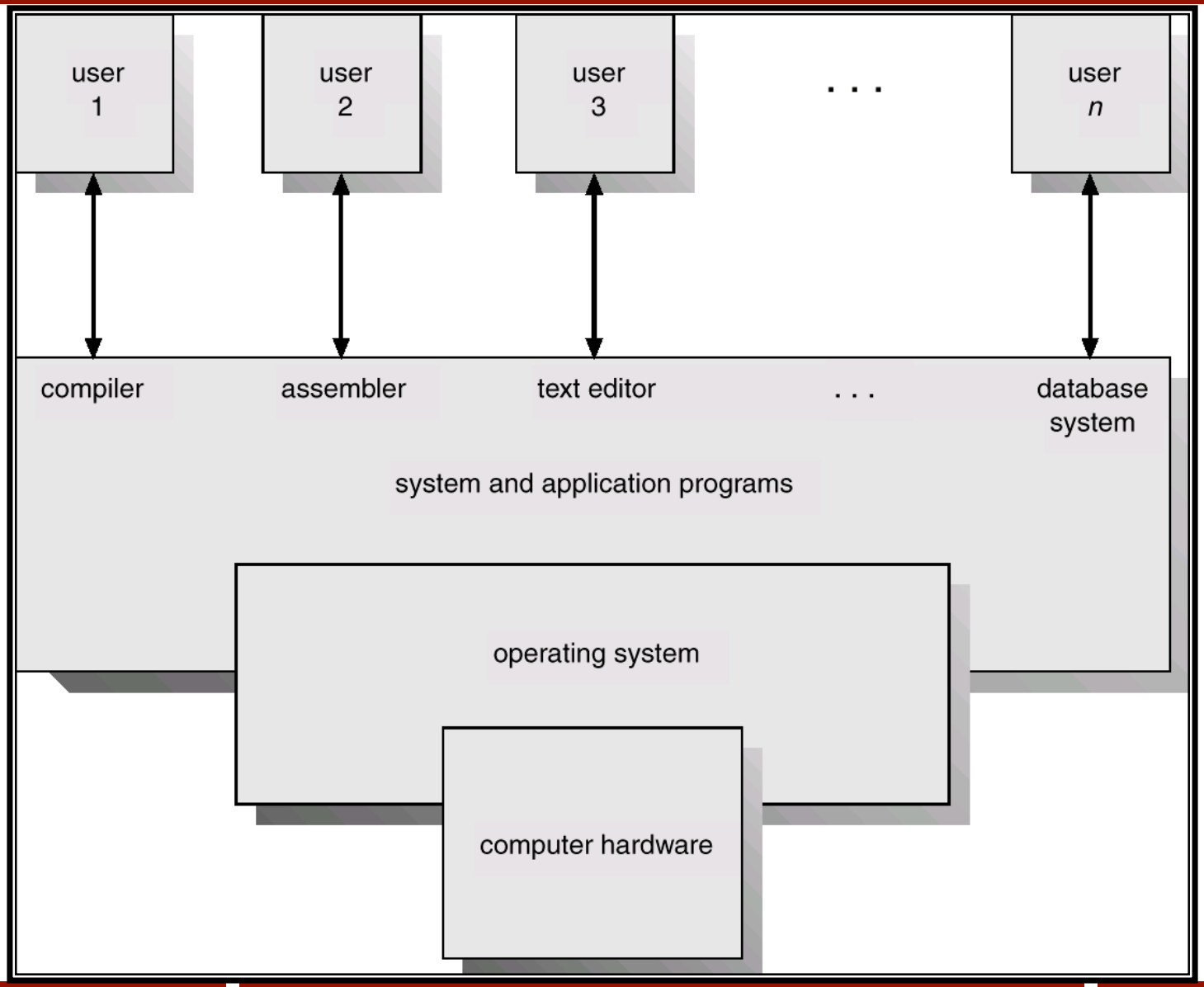
- Intermediary between hardware and applications.

- Hardware provides resources such as processing elements (CPU, coprocessors, graphics processors etc.), storage (memory, cache, disk, tape etc.) and I/O (terminal, screen, etc.). OS makes it easier for applications to access these resources.

Operating Systems

- "Fair" allocator of resources
  - Fairness depends on the system
    - Single user – Interactive performance
    - Multi user – improve utilization of costly resources
    - Real time – hard real time tasks
    - Mobile device – energy consumption
- Abstract lower level details from user
    - File systems vs disks
    - Windowing abstractions vs frame buffers
  - Common functionality across various hardware platforms
  - Should allow users to circumvent abstractions for performance

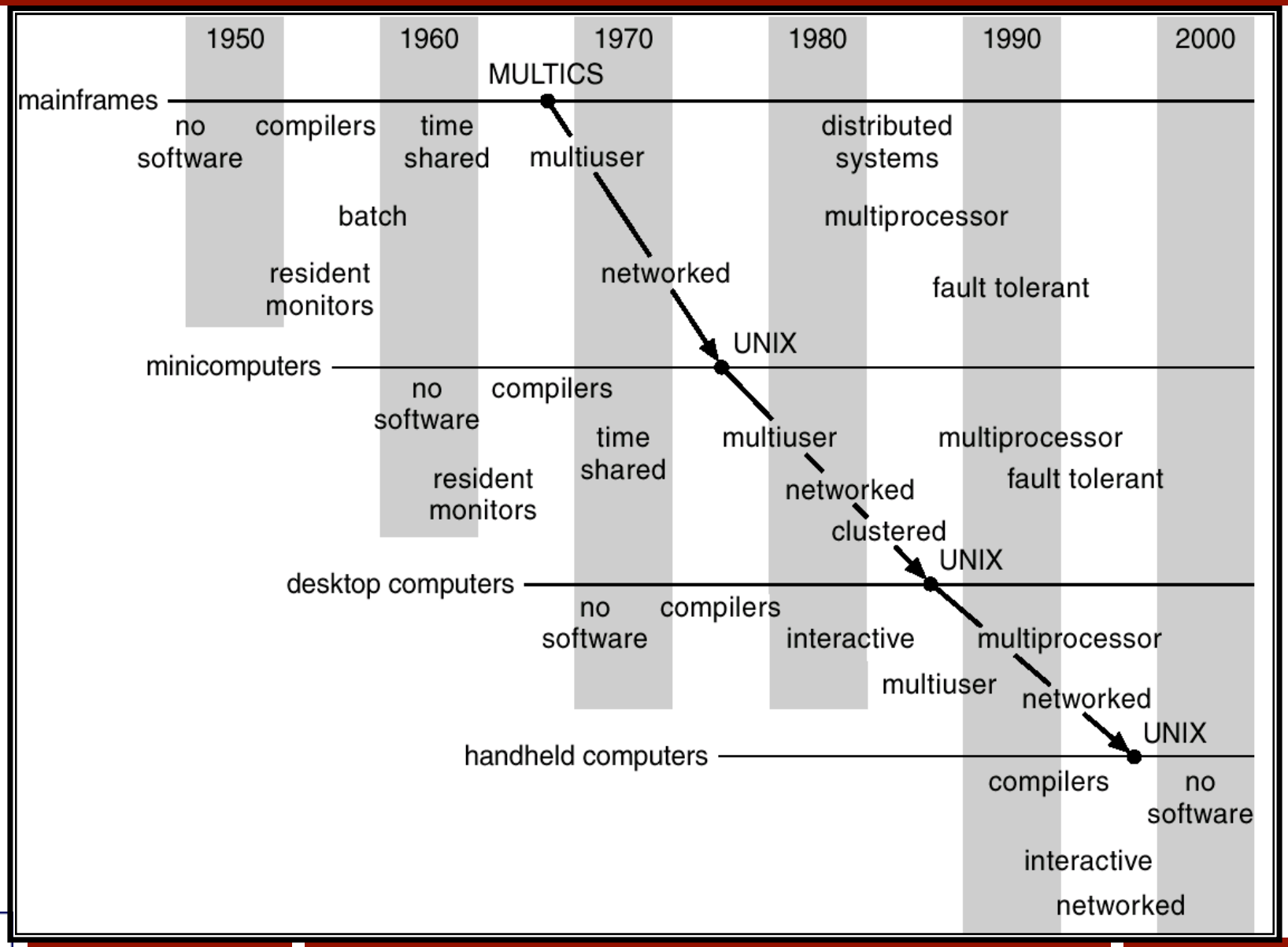
System Components



Historical evolution

- Main frames ($$$$):
  - Batch processing system
    - Machine was mostly idle waiting for I/O devices
    - Very little memory (a few KB)
  - Multiprogramming
    - Many jobs resident at the same time
- Time-sharing systems
    - Provides interactive
    - Virtual memory
- Desktops
    - Interactive performance critical
- PDAs (Personal digital assistant - Palm pilot, Linux watch)
- Parallel (SMPs), Distributed, Clustered
- Real time (Engine controller in your car)
- P2P

Migration of Operating-System Concepts and Features



Structure

- Modern OS are interrupt driven
  - Software generated interrupts via system calls
    - C libraries perform some work before requiring kernel help inside a system call - e.g. read the next character
  - Hardware interrupts
    - Synchronous and asynchronous I/O
      - Synchronous I/O: Wait till I/O operation finishes
        - » Read(), write()
      - Asynchrnous I/O: Returns before I/O is completed. Application is notified when I/O completes
    - Coordinate direct memory access (DMA)
      - I/O device directly reads into memory; once complete, the I/O device interrupts the OS

Structure

- Storage hierarchy
  - Fast to slower (or Expensive to cheaper)
    - Registers -> CPU Cache -> memory -> disks -> tape
  - As technology changes some hierarchies have less meaning
    - Cheaper to buy more memory than to buy extra swap (and pay the performance penalty)
- Caching improves performance
  - Faster to write into a buffer and asynchrnously flush to cache than to wait till data is written to disk
  - Coherency and consistency are the problems that need to be solved

Structure

- Protection
  - Prevent users from corrupting their own data, other's data or crash the machine
  - Desired protection different for single user vs multi user
  - Supervisor mode and user mode to achieve protection
  - Memory protection using hardware
  - CPU protection using context switch
    - After allocated time quanta expires, the CPU context switches and runs another application

Functional categorization

- Process management
  - Creating processes, synchronization, deadlocks etc
- Main memory management
  - Allocation and deallocation
- File management
  - Files, directories
- I/O system
  - Device drivers
- Networking
  - Communication abstractions
- Protection
- Command-Interpreter system

System structure

- Simple structure
  - MS Dos, PalmOS
- Layered approach
  - OS/2
- Microkernel
  - Design a simple efficient core
  - Build services on top of this abstraction
  - Mach (basis for Mac OS X)
- Virtual machine
  - IBM VM/CMS, Java

Application Interface

- Unstructured
  - MS Dos
- Event driven
  - PalmOS
- File system based
  - UNIX, Plan 9
- Object oriented
  - Hydra, OPAL
- Distributed OS
  - Amoeba
- Real time
  - QNX
- Single Address Space OS (SASOS)
  - OPAL

Discussion

- We will mostly discuss general purpose operating systems and their abstractions
- We would not focus on special purpose operating systems in microwaves, VCRs etc.
- Interesting Links:
  - http://www.digibarn.com/stories/desktop-history/bushytree.html
  - http://www.digibarn.com/

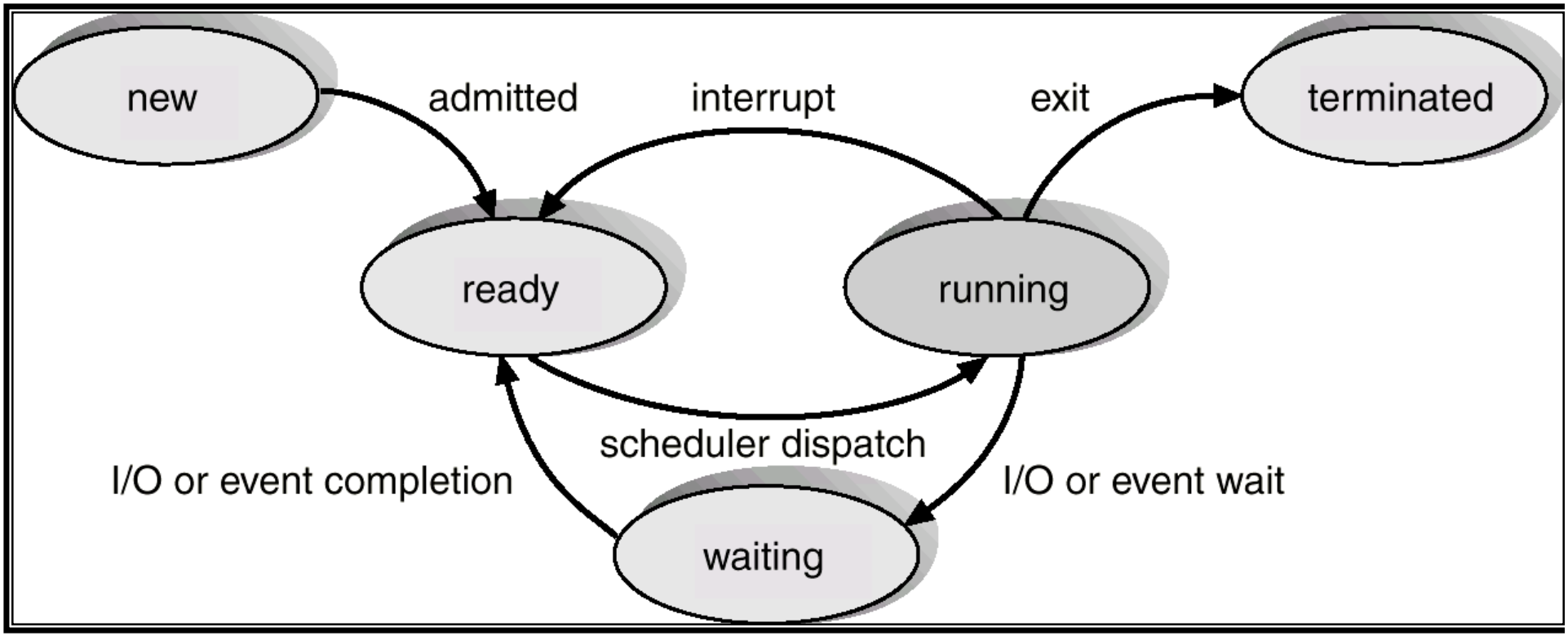
Review

- Chapter 4: Processes
- Chapter 5: Threads

Processes

- Process is a program in execution: It's the job of the OS to provide the environment required by the application to execute fruitfully
  - Program code
  - Data section
  - Execution context: Program counter, registers, stack
- Process has thread(s) of control
- Many processes "run" concurrently: Process scheduling
  - Fair allocation of CPU and I/O bound processes
    - Context switch
- Many users may run many tasks: Protect users from other users as well as from themselves.

Process States



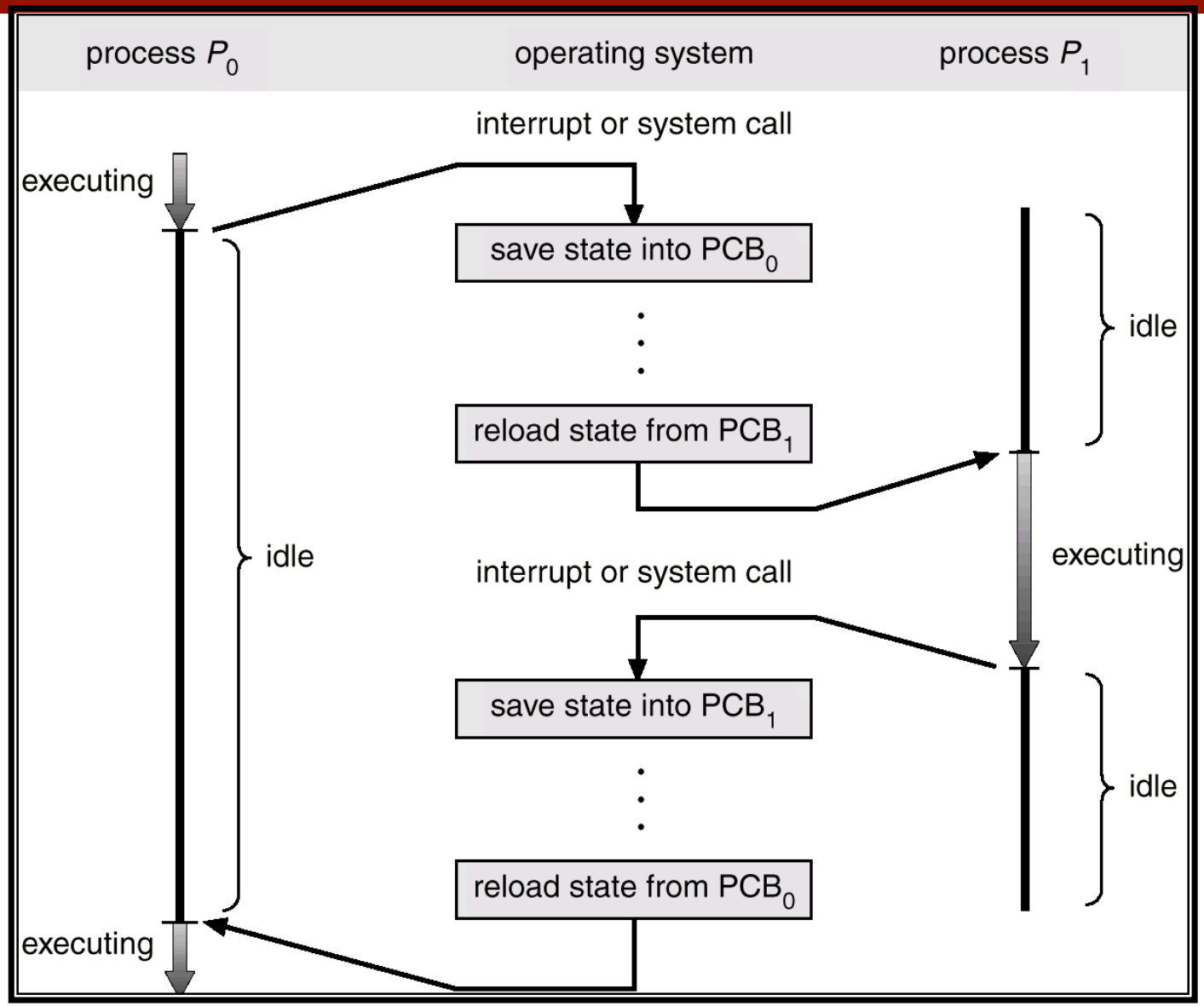
Process Control Block

pointer	process state
process number	
program counter	
registers	
memory limits	
list of open files	
⋮	

Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Process context switch



Process creation

- Creating new processes is expensive
  - Resource allocation issue
- Fork mechanism: UNIX, Windows NT
  - Duplicate the parent process
  - Shares file descriptors, memory is copied
  - Exec to create different process
  - Various optimizations to avoid copying the entire parent context (Copy on write (COW), etc..)
- Exec mechanism: VMS, Windows NT
  - New process is specifically loaded

Interprocess communication

- Processes need to communicate with each other
  - Naming
  - Message-passing
    - Direct (to process) or indirect (port, mailbox)
    - Symmetric or asymmetric (blocking, nonblocking)
    - Automatic or explicit buffering (capacity)
    - Send by copy or reference
    - Fixed size or variable size messages
  - Shared memory/mutexes
  - Remote Procedure Call (RPC/RMI)
- Bounded buffer problem
  - Producer buffers, consumer takes from buffer - finite buffer

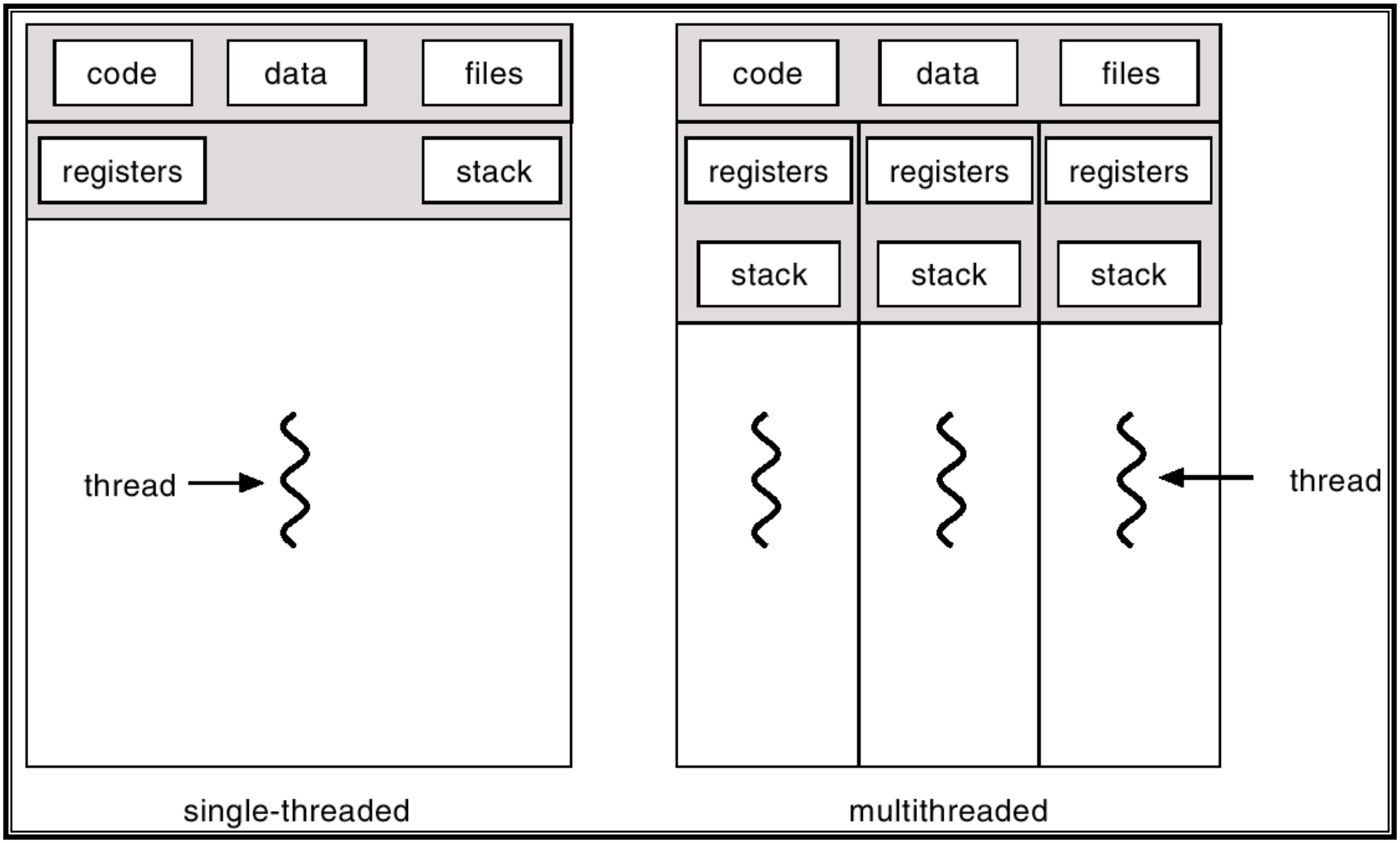
CPU scheduling

- Interleave processes so as to maximize utilization of CPU and I/O resources
- Scheduler should be fast as time spent in scheduler is wasted time
  - Switching context (h/w assists – register windows [sparc])
  - Switching to user mode
  - Jumping to proper location
- Preemptive scheduling:
  - Context switch without waiting for application to relinquish
  - Process could be in the middle of an operation
  - Especially bad for kernel structures
- Non-preemptive (cooperative) scheduling:
  - Can lead to Starvation

Threads

- Applications require concurrency. Threads provide a neat abstraction to specify concurrency
- E.g. word processor application
  - Needs to accept user input, display it on screen, spell check and grammar check
  - Implicit: Write code that reads user input, displays/formats it on screen, calls spell checked etc. while making sure that interactive response does not suffer. May or may not leverage multiple processors
  - Threads: Use threads to perform each task and communicate using queues and shared data structures
  - Processes: expensive to create and do not share data structures and so explicitly passed

Threaded application



Threads - Benefits

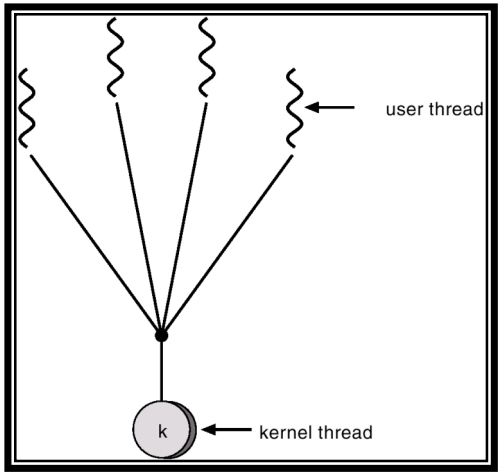
- Responsiveness
  - If one “task” takes too long, other “tasks” can still proceed
- Resource sharing: (No protection between threads)
  - Grammar checker can check the buffer as it is being typed
- Economy:
  - Process creation is expensive (spell checker)
- Utilization of multiprocessor architectures:
  - If we had four processors (say), the word processor can fully leverage them
- Pitfalls:
  - Shared data should be protected or results are undefined
    - Race conditions, dead locks, starvation (more later)

Thread types

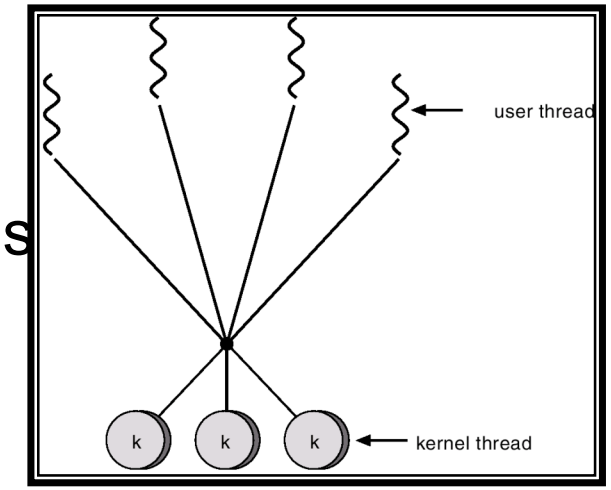
- Continuum: Cost to create and ease of management
- User level threads (e.g. pthreads)
  - Implemented as a library
  - Fast to create
  - Cannot have blocking system calls
  - Scheduling conflicts between kernel and threads. User level threads cannot do anything is kernel preempts the process
- Kernel level threads
  - Slower to create and manage
  - Blocking system calls are no problem
  - Most OS's support these threads

Threading models

- One to One model
  - Map each user thread to one kernel thread



- Many to one model
  - Map many user threads to a single kernel thread
  - Cannot exploit multiprocessors



- Many to many
  - Map *m* user threads to *n* kernel threads

Threading Issues:

- Cancellation:
  - Asynchronous or deferred cancellation
- Signal handling: which thread of a task should get it?
  - Relevant thread
  - Every thread
  - Certain threads
  - Specific thread
- Pooled threads (web server)
- Thread specific data

Wizard ‘ps -cfLeP’ output

```
UID       PID  PPID  LWP PSR   NLWP CLS PRI  STIME  TTY      LTIME CMD
root      0     0     1   -     1   SYS  96   Aug 03 ?         0:01 sched
root      1     0     1   -     1    TS  59   Aug 03 ?         7:12 /etc/init -
root      2     0     1   -     1   SYS  98   Aug 03 ?         0:00 pageout
root      3     0     1   -     1   SYS  60   Aug 03 ?       275:46 fsflush

root   477   352     1   -     1    IA  59   Aug 03 ??        0:0
                                                /usr/openwin/bin/fbconsole -d :0
root    62     1    14   -    14    TS  59   Aug 04 ?         0:00
                                                      /usr/lib/sysevent/syseventd
```

Discussion

- Constant tension between moving functionality to upper layers; involving the application programmer and performing automatically at the lower layers
- Automatically create/manage threads by compiler/ system? (open research question)