Operating system

Part III: Process [进程] & Thread [线程]

Know data structures to maintain the resources needed when executing your program

By KONG LingBo (孔令波)

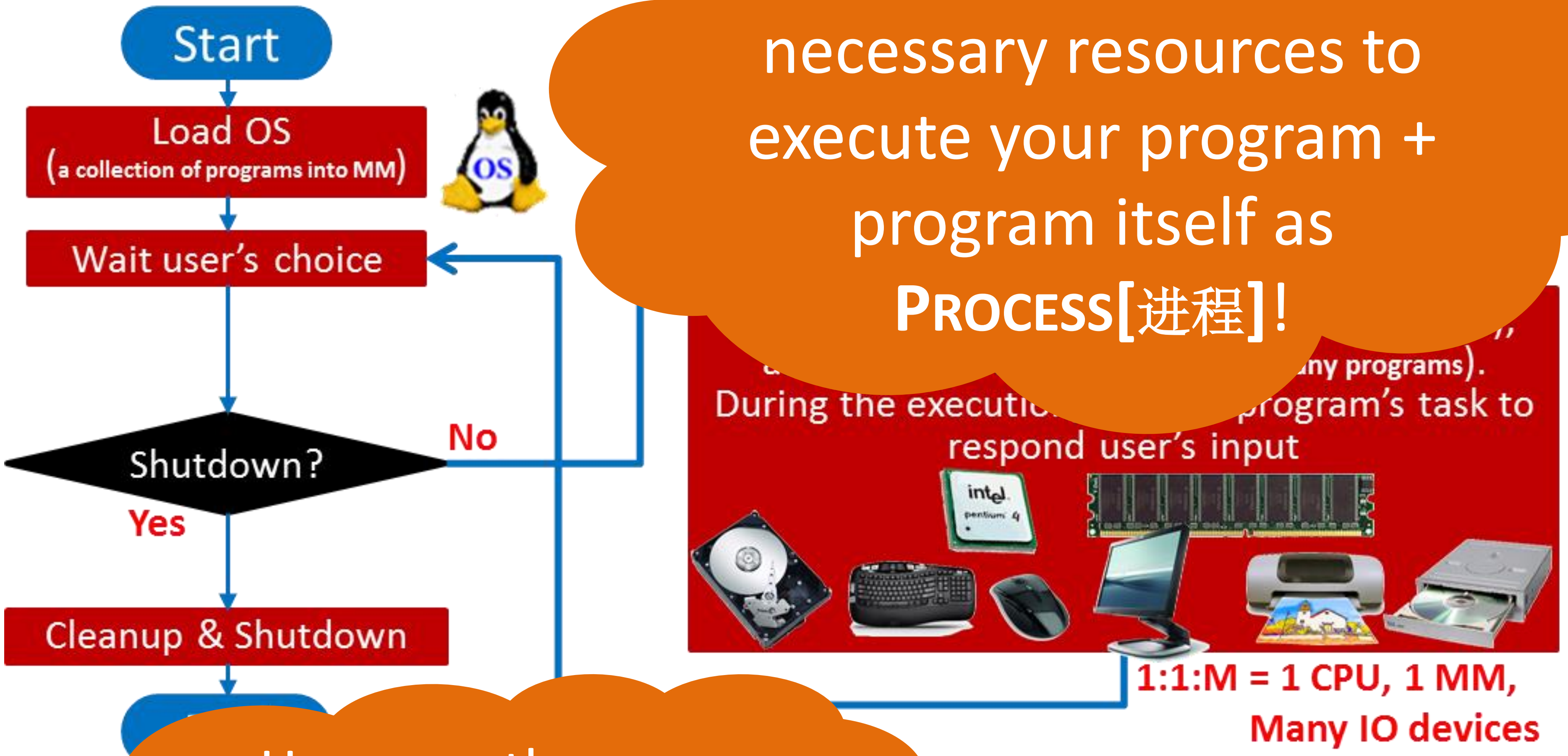
Outline of topics covered by this course

- **INTRODUCTION** – why should we learn OS?
- **OVERVIEW** – what problems are considered by modern OS in more details?
- **EXECUTION** – CPU management
  - Process and Thread
  - CPU scheduling
- **EXECUTION** – competition (synchronization problem)
  - Synchronization
  - Deadlock

Process

- To understand the execution of your program
  - Process [进程] is the traditional concept
    - The identification to manage the needed information to run one program (**OS's or user's**)
      - **PCB** is the data structure to record the necessary information: resources (MM, ownership, security, …), execution stages/states, …
    - Additional data structures and algorithms are needed to manage the concurrent execution of many programs
      - Queues, and schedulers
      - Inter-process communication (IPC)
  - Thread[线程] is the modern concept
    - The idea could be seen as **MULTIPLEX**ing **process,** *namely that your program is constructed to have more than one execution units*
      - CPU is occupied by the process, however the usage of the CPU is shared among the internal execution units (threads)
        - » The resources assigned to the process could be shared by those threads.

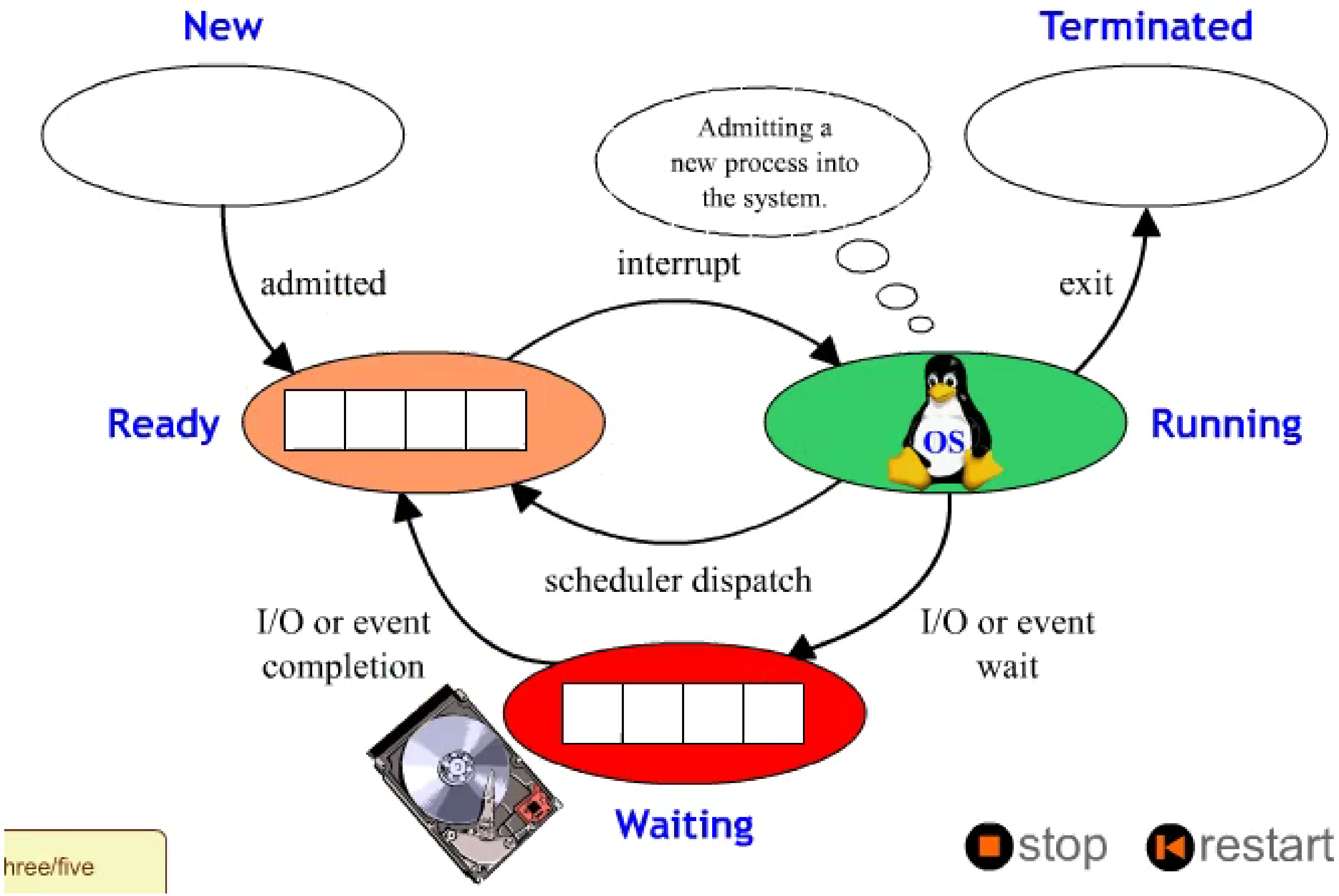
We have learned the



Inside OS, I prefer to call the collection of necessary resources to execute your program + program itself as **PROCESS[进程]!**

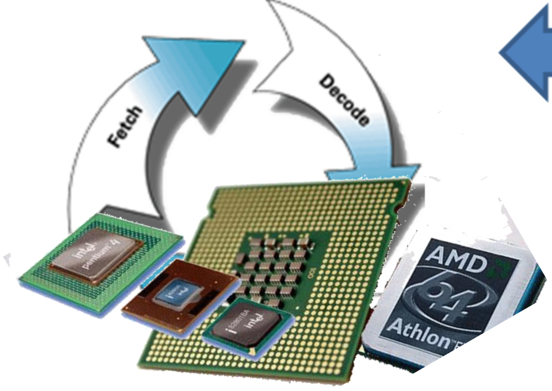
How exactly are those programs controlled by OS?

The execution of your program is alive



The execution of your program needs some information to support

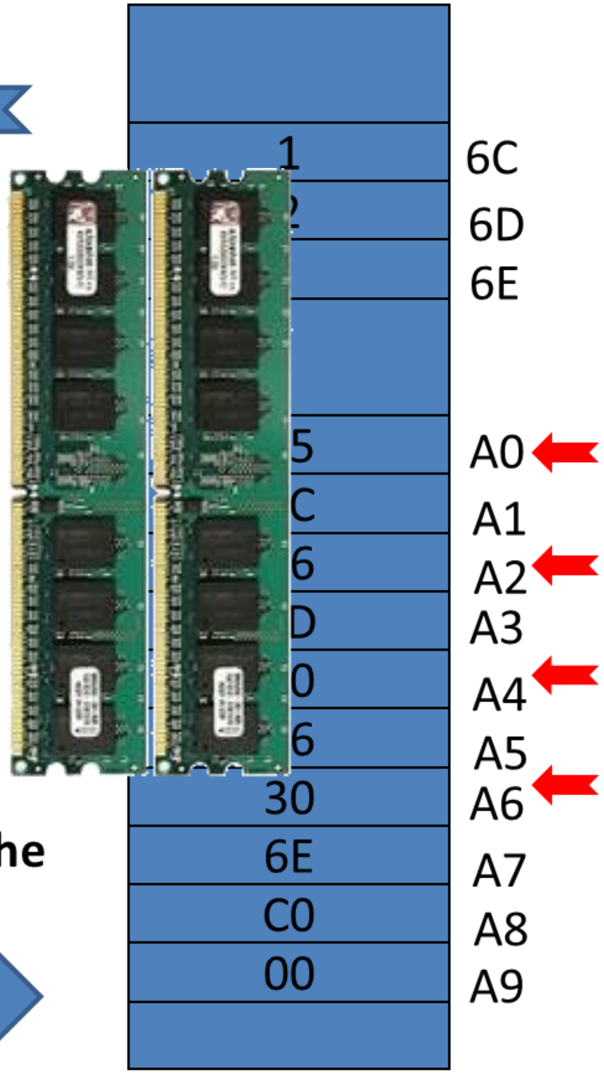
execute instructions serially in CPU



Programs are stored as Files in storage media

Program
X=1;
Y= 2;
Z= X+Y;

Machine code
156C→0001010101101100
166D→0001011001101101
5056→0101000001010110
306E→0011000001101110
C000→1100000000000000

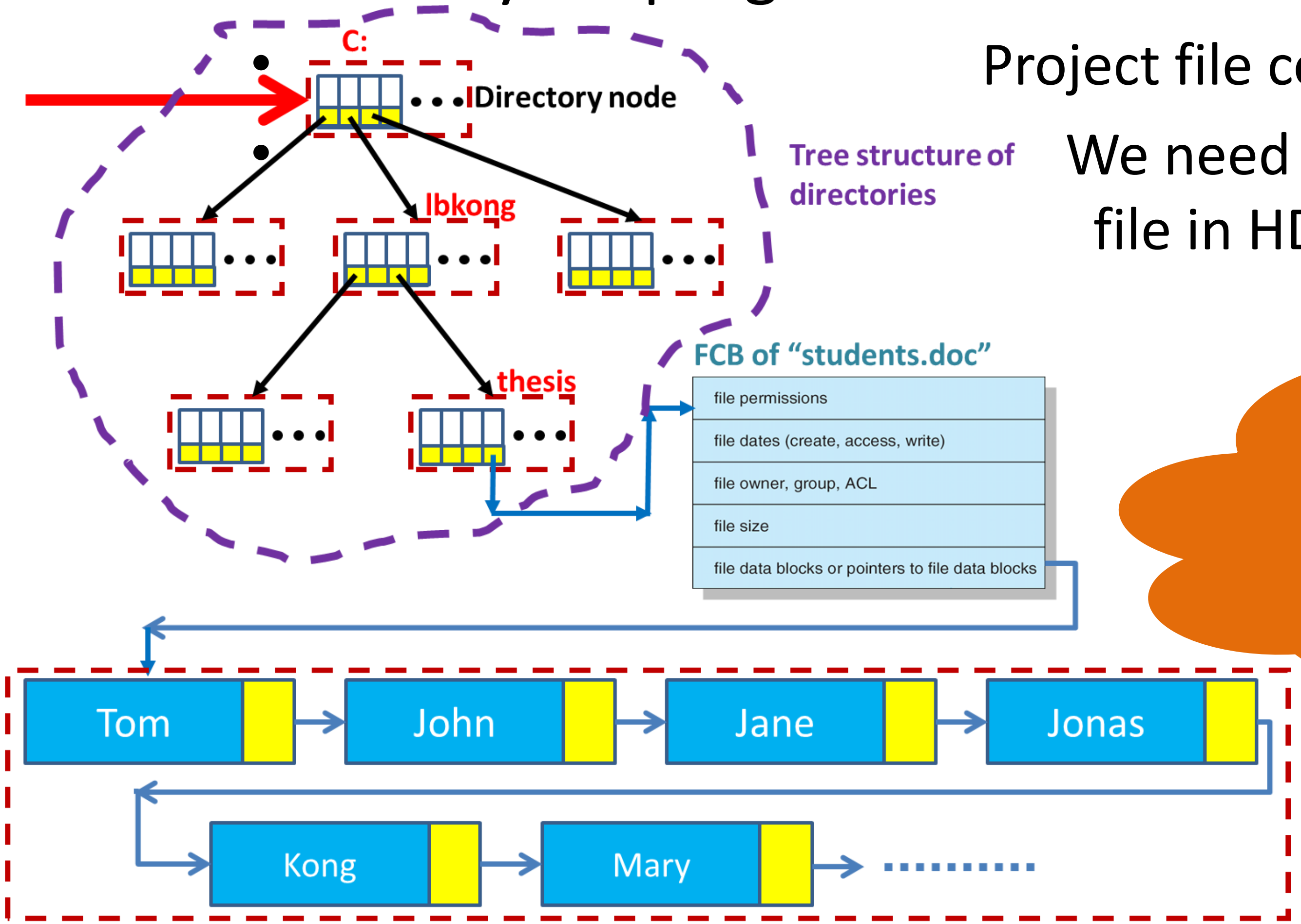


HDD

Put executable codes into the memory of the computer

Instructions are mapped into the address space of memory

- To support the execution of a program, we need to record some information (**define some data structures**) – besides the program itself
  - Where is your program stored in the HDD?



File space: collection of blocks used to store the data of the file

Project file contains the organization information.

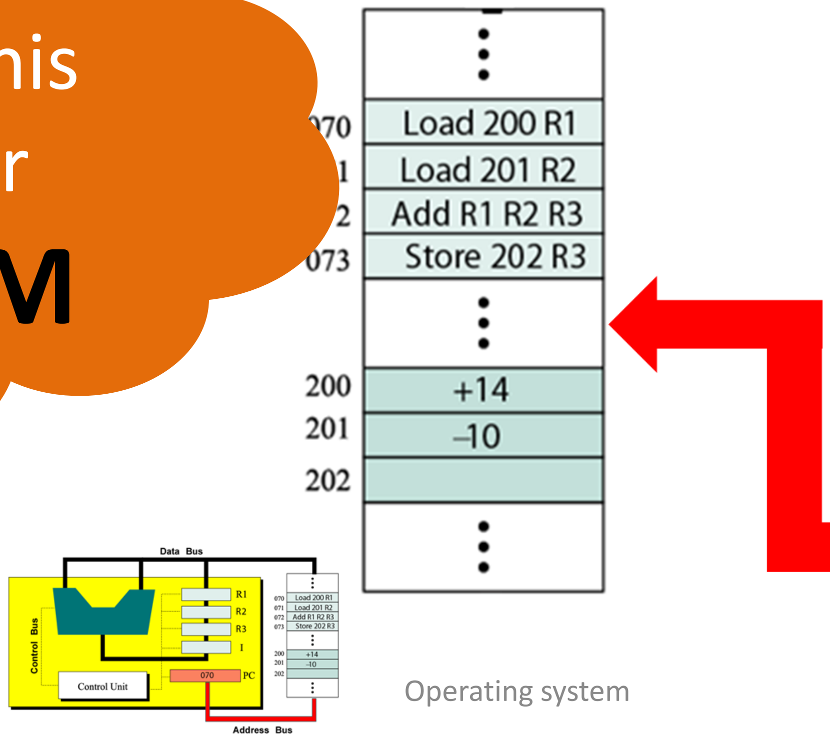
We need information to locate that project file in HDD

We'll learn this **how** in later chapter: **IO + File**

– Where could we copy the program in MM?

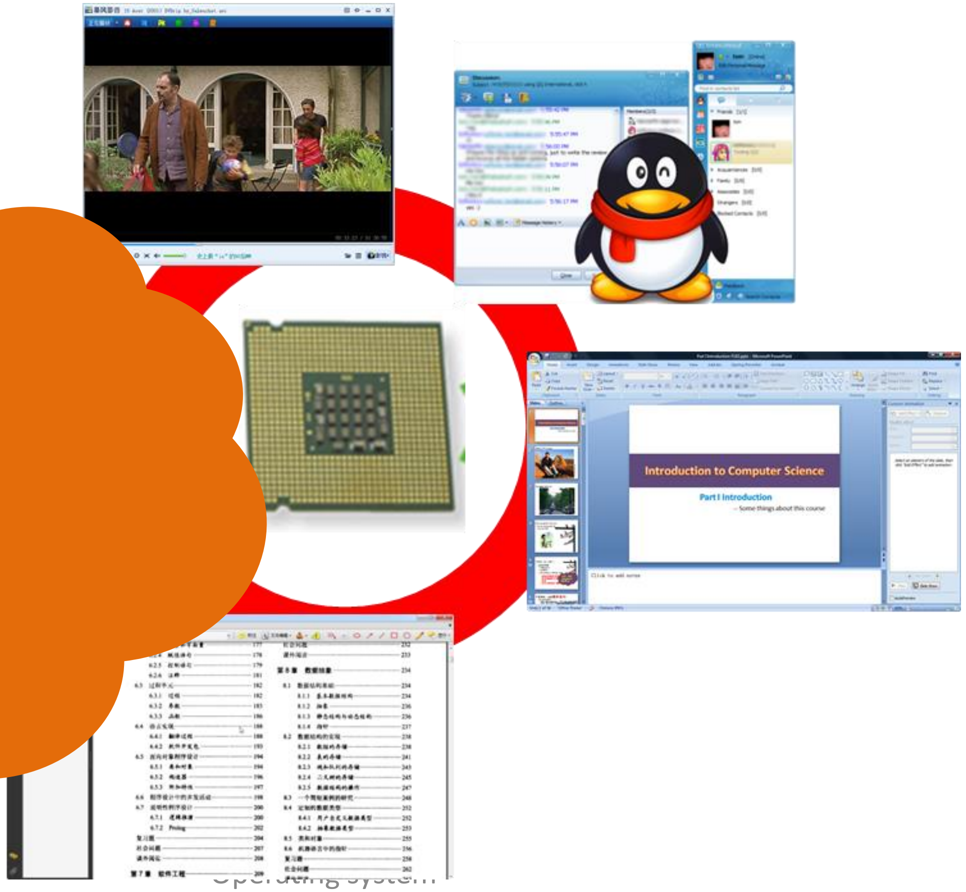
- We need information about the usage of MM first
  - Data structure (+algorithms) to manage MM space
- If available, the program is copied into MM
- We need information to locate the instructions in MM

We'll learn this **how** in later chapter: **MM**



Machine code
[00] 14 (should be bin)
[01] -10
[02] (used later)
[03] 0001 **00000000**
[04] 0010 **00000001**
[05] 0011
[06] 0100 **00000010**

- We also need to record the information of current instruction for CPU switching
  - The address of the instruction when CPU is switched, like 1031



We'll learn this **how** in later chapter: **CPU Scheduling**

PCB is the **very** concept/data structure

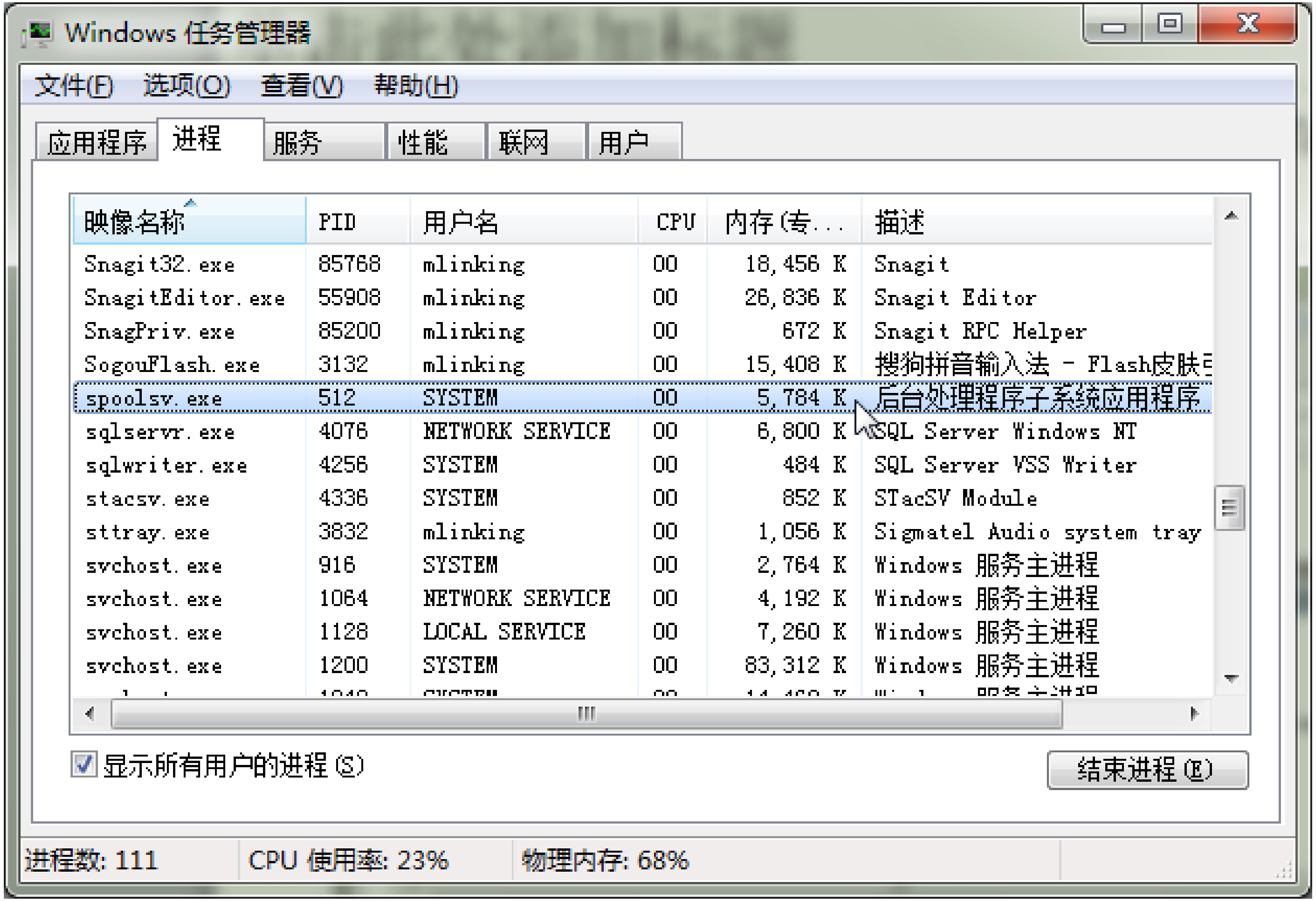
- You have learned how to record those kinds of information in **programming**.
  - Design the **data structure**!

- **PCB** (Process Control Block) is the one used/named data structure
  1. Process **location** information
  2. Process **identification** information
  3. Process **state** information
  4. Process **control** information

PCB: Process Location Information

- Process **Location** Information: Each process image in memory
  - may **not** occupy a contiguous range of addresses (depends on memory management scheme used, which will be discussed in later MM part).
    - both a private and shared memory address space can be used.
- Process **Identification** Information: A few numeric identifiers may be used
  - Unique process identifier (PID) –
    - indexes (directly or indirectly) into the process table.
  - User identifier (UID) –
    - the user who is responsible for the job.
  - Identifier of the process that created this process (PPID).

- The process!

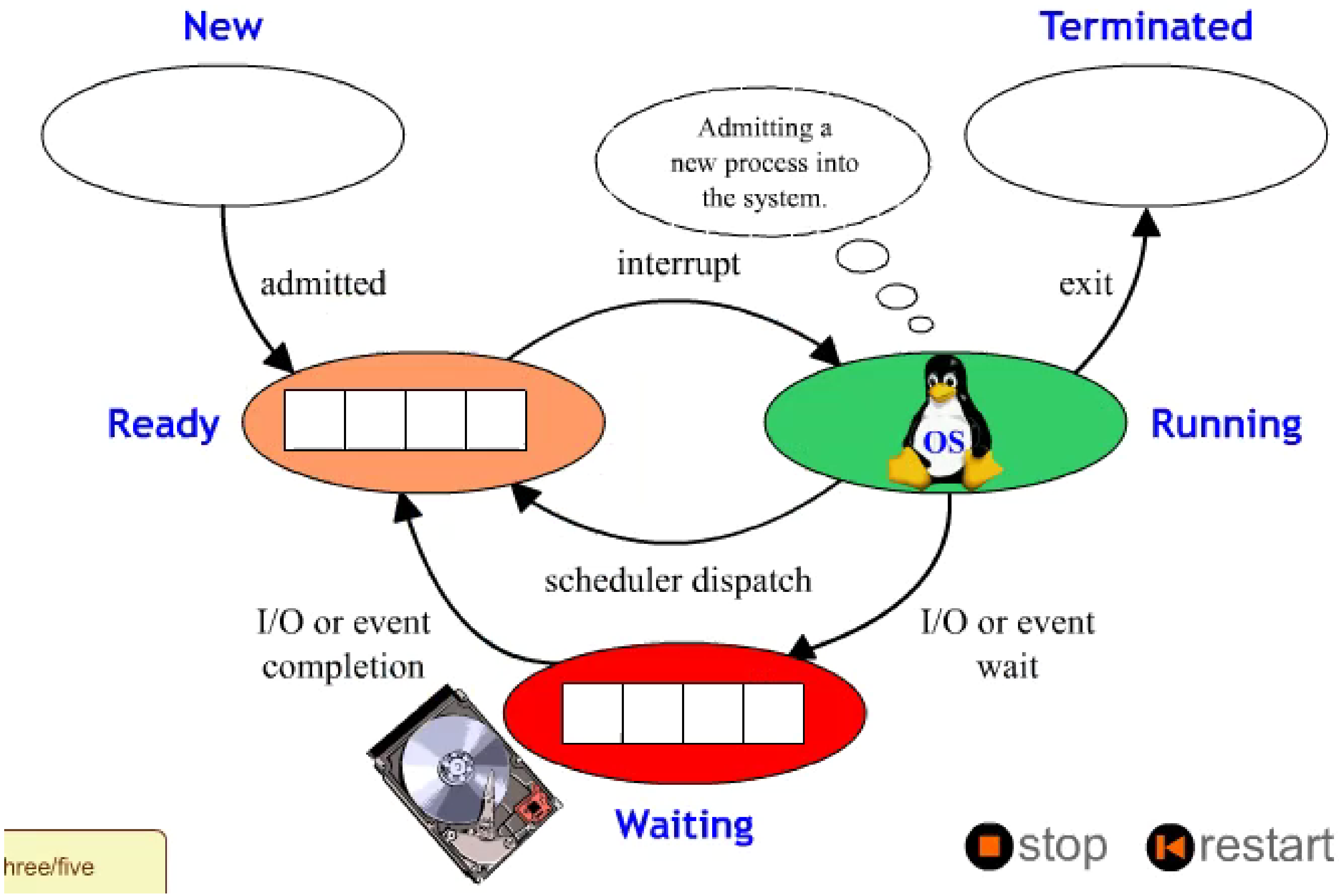

Windows 任务管理器
文件(F) 选项(O) 查看(V) 帮助(H)
应用程序 进程 服务 性能 联网 用户
映像名称 PID 用户名 CPU 内存(专... 描述
Snagit32.exe 85768 mlinking 00 18,456 K Snagit
SnagitEditor.exe 55908 mlinking 00 26,836 K Snagit Editor
SnagPriv.exe 85200 mlinking 00 672 K Snagit RPC Helper
SogouFlash.exe 3132 mlinking 00 15,408 K 搜狗拼音输入法 - Flash皮肤
spoolsv.exe 512 SYSTEM 00 5,784 K 后台处理程序子系统应用程序
sqlservr.exe 4076 NETWORK SERVICE 00 6,800 K SQL Server Windows NT
sqlwriter.exe 4256 SYSTEM 00 484 K SQL Server VSS Writer
stacsv.exe 4336 SYSTEM 00 852 K STacSV Module
sttray.exe 3832 mlinking 00 1,056 K Sigmatel Audio system tray
svchost.exe 916 SYSTEM 00 2,764 K Windows 服务主进程
svchost.exe 1064 NETWORK SERVICE 00 4,192 K Windows 服务主进程
svchost.exe 1128 LOCAL SERVICE 00 7,260 K Windows 服务主进程
svchost.exe 1200 SYSTEM 00 83,312 K Windows 服务主进程
显示所有用户的进程(S)
结束进程(E)
进程数: 111 CPU 使用率: 23% 物理内存: 68%

- Processor **State** Infor
registers
  - User-visible registe
  - Control and status
  - Stack pointers

Eh? What’s the meaning of “Control” information here? – Because process is alive in OS!

- Process **Control** Information: Scheduling and state information
    - Process state (i.e., running, ready, blocked...)
    - Priority of the process
    - Relationship with other processes
        - the process is waiting (if blocked).
        - other PCBs for process queues, parent-child relationships and other structures

The execution of your program is alive



Program execution is dynamic→ Process has **states**

- As a process executes, it chang
  - The state of a process is defin
    activity of that process.
- Each process may be in one o
  - **New**. The process is being created.
  - **Running**. Instructions are being executed ←Get **CPU**.
  - **Waiting**. The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
  - **Ready**. The process is waiting to be assigned to a processor.
  - **Terminated**. The process has finished execution.

We'll learn this <u>**how**</u> in later chapter: **CPU Scheduling**

Process Transitions (1)

- Ready → Running
  - When it is time, the dispatcher selects a new process to run.
- Running → Ready
  - the running process has expired his time slot.
  - the running process gets interrupted because a higher priority process is in the ready state.

Process Transitions (2)

- Running → Waiting
  - When a process requests something for which it must wait:
    - a service that the OS is not ready to perform.
    - an access to a resource not yet available.
    - initiates I/O and must wait for the result.
    - waiting for a process to provide input.
- Waiting → Ready
  - When the event for which it was waiting occurs.

PCB defined in Linux

PCB example

- PCB in Linux is defined using a struct **task_struct**
- There are many parameters in Linux's PCB
- The size of each PCB is usually a little larger than 1KB

```
struct task_struct{
    ...
    unsigned short uid;
    int pid;
    int processor;
    ...
    volatile long state;
    long priority;
    unsighed long rt_prority;
    long counter;
    unsigned long flags;
    unsigned long policy;
    ...
    Struct task_struct *next_task, *prev_task;
    Struct task_struct *next_run,*prev_run;
    Struct task_struct  *p_opptr, *p_pptr, *p_cptr, *pysptr, *p_ptr;
    ...
};
```

(2)int pid is the ID of the current process

(3)int processor: the CPU used by the current process. Support multi-processor

(4)volatile long state: corresponds to the states defined as follows：

Running (TASK-RUNING):可运行状态；

Interruptible state (TASK-IntERRUPTIBLE):可中断阻塞状态

Uninterruptible state (TASK-UNINTERRUPTIBLE):不可中断阻塞状态

Zombie (TASK-ZOMBIE):僵死状态

Stopped (TASK_STOPPED):暂停态

Swapping (TASK_SWAPPING):交换态

(5)long priority [进程的优先级]

(6)unsigned long rt_priority [实时进程的优先级，对于普通进程无效]

(7)long counter: a counter for counting the priority, used by Round Robin scheduling algorithm

(8)unsigned long policy: Strat f

SCHED_OTHER(=0) Round

SCHED_FIFO(=1) RT

SCHED_RR(=2) RT

I hope you could infer the related operations for PCB data structure. You've been trained in DSA course

(9)struct task_struct *next_ta ters for PCB's **Double linked lists** [进程PCB双向链表的前后项指针]

(10)struct task_struct *next_run,*prev_run: pointers for the PCBs in ready queue [就绪队列双向链表的前后项指针]

(11)struct task_struct *p_opptr,*p_pptr,*p_cptr,*p_ysptr,*p_ptr: relations among the process family [指明进程家族间的关系，分别为指向祖父进程、父进程、子进程以及新老进程的指针]

Process

- To understand the execution of your program
  - Process [进程] is the traditional concept
    - The identification to manage the needed information to run one program
      - PCB is the data structure to record the necessary information: resources (MM, ownership, security, …), execution stages/states, …
    - Additional data structures and algorithms are needed to manage the concurrent execution of many programs
      - Queues, and schedulers
      - Inter-process communication (IPC)
  - Thread[线程] is the modern concept
    - The idea could be seen as **MULTIPLEX**ing **process,** *namely that your program is constructed to have more than one execution units*
      - CPU is occupied by the process, however the usage of the CPU is shared among the internal execution units (threads)
        - » The resources assigned to the process could be shared by those threads.

Five-State Model

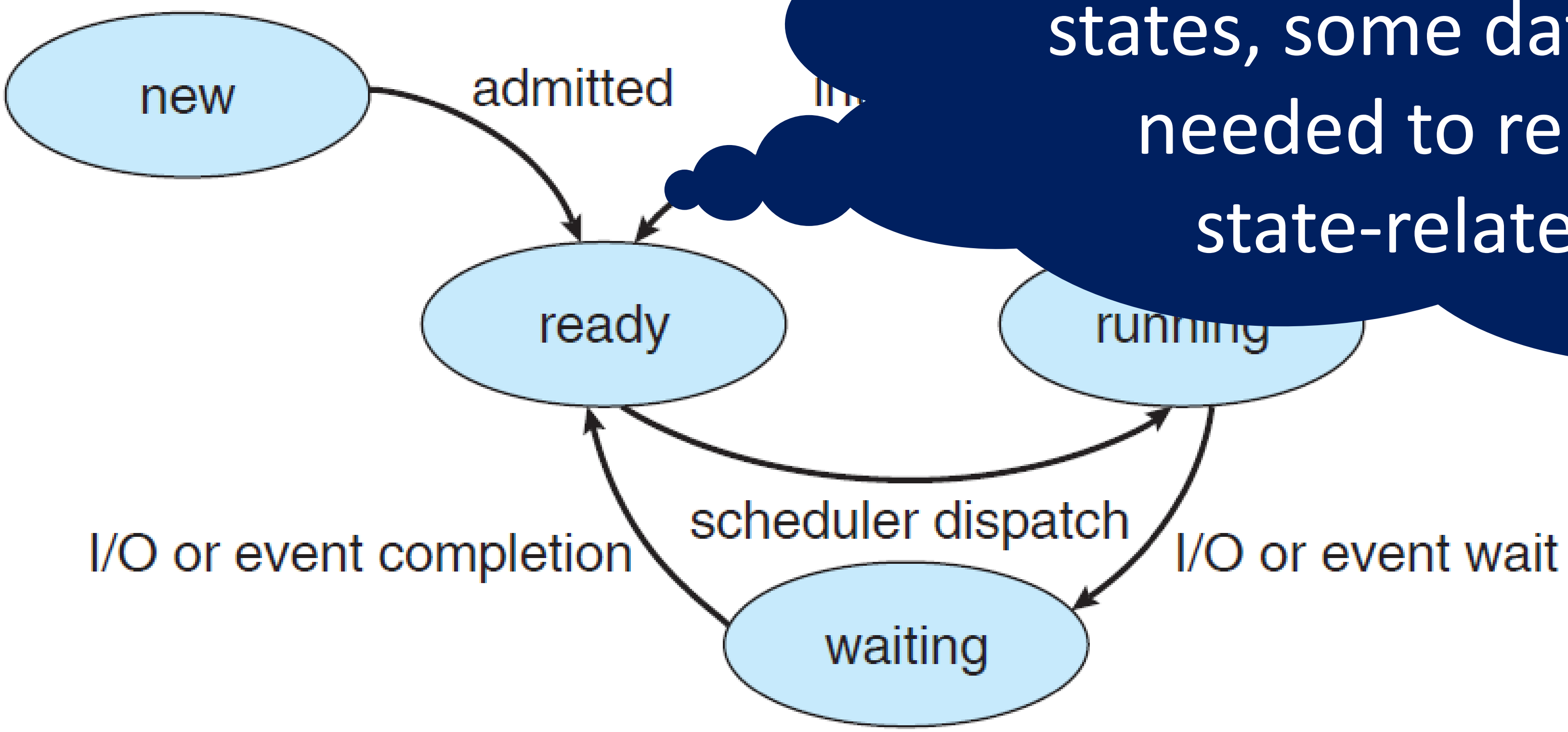


Figure 3.2 Diagram of process state.

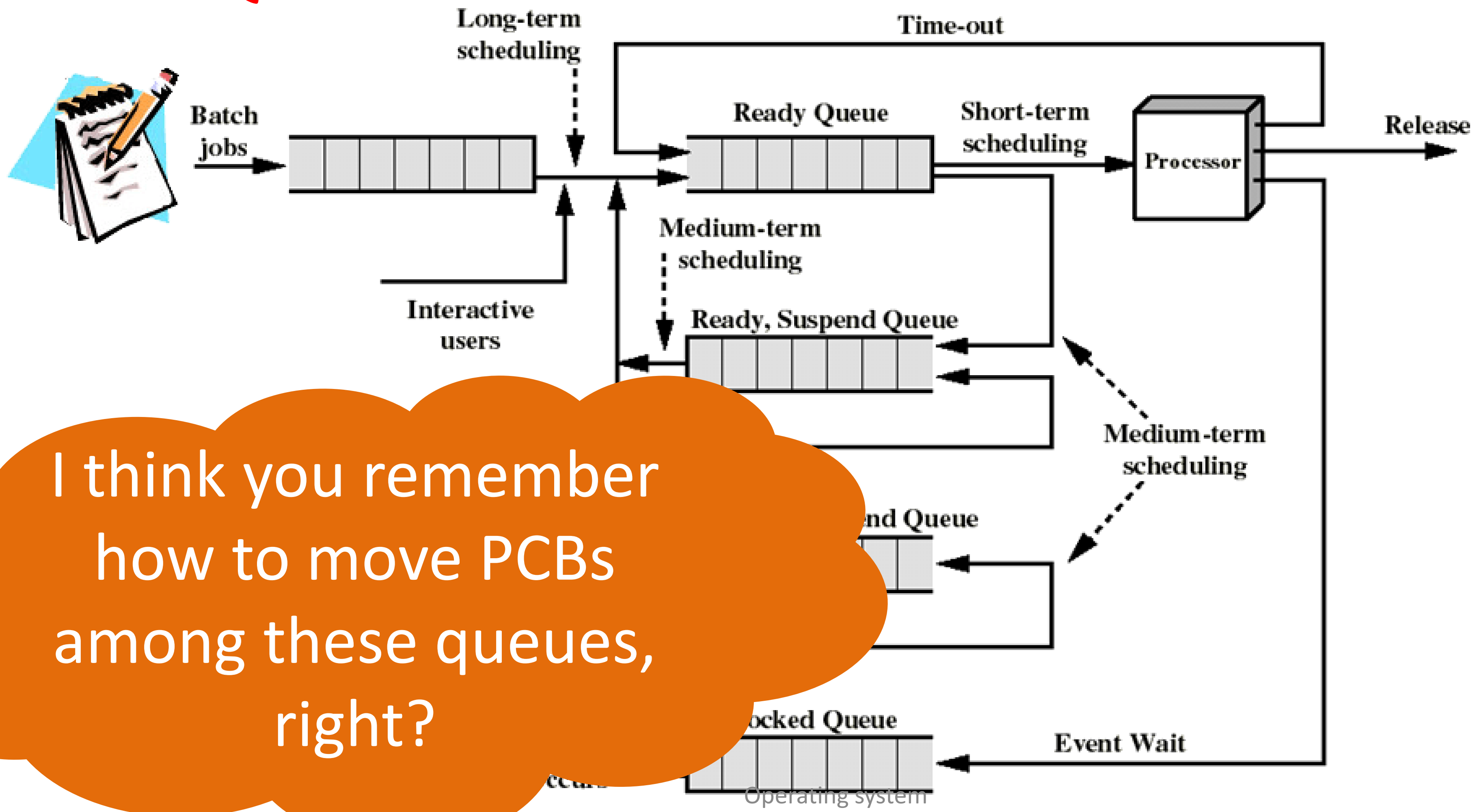
Since usually there may be several processes in these states, some data structures are needed to represent those state-related processes

It's also OK to use following 5 states: Running, Ready, **Blocked**, New, **Exit**

Supplement:

Data structures to manage those state-related processes

- **Queue**! ← You have learned it



I think you remember how to move PCBs among these queues, right?

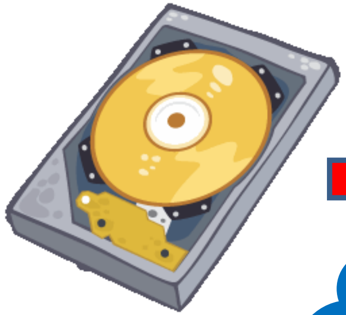
Three kinds of schedulers

1. Long-term scheduler (jobs scheduler) – selects which programs/processes should be brought into the **ready queue**.
2. Medium-term scheduler (emergency scheduler) – selects which job/process should be **swapped** out if system is loaded.
3. Short-term scheduler (CPU scheduler) – selects which process should be **executed** next and allocates CPU.

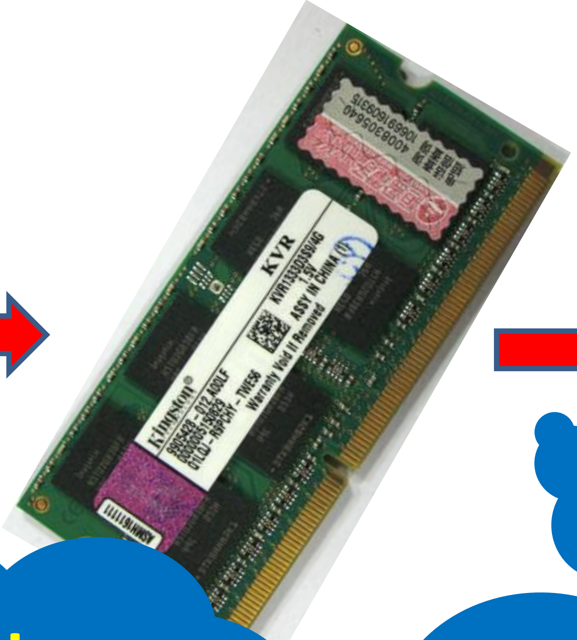
PPTs from others\From Ariel J. Frank\OS381\os3-2.ppt

Those queues are used by Three kinds of schedulers

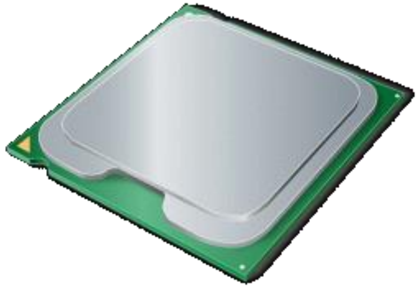
Storage media
(Magnetic disk)



Main Memory



CPU



Long-term scheduler determines which jobs could be transited into main memory

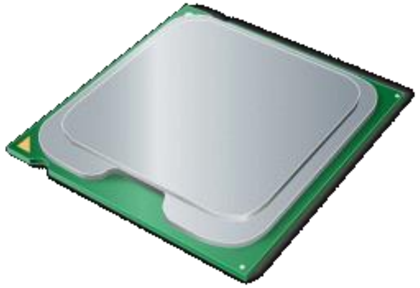
Short-term scheduler determines which processes could get the usage of CPU

Three kinds of schedulers

Storage media
(Magnetic disk)

Main Memory

CPU

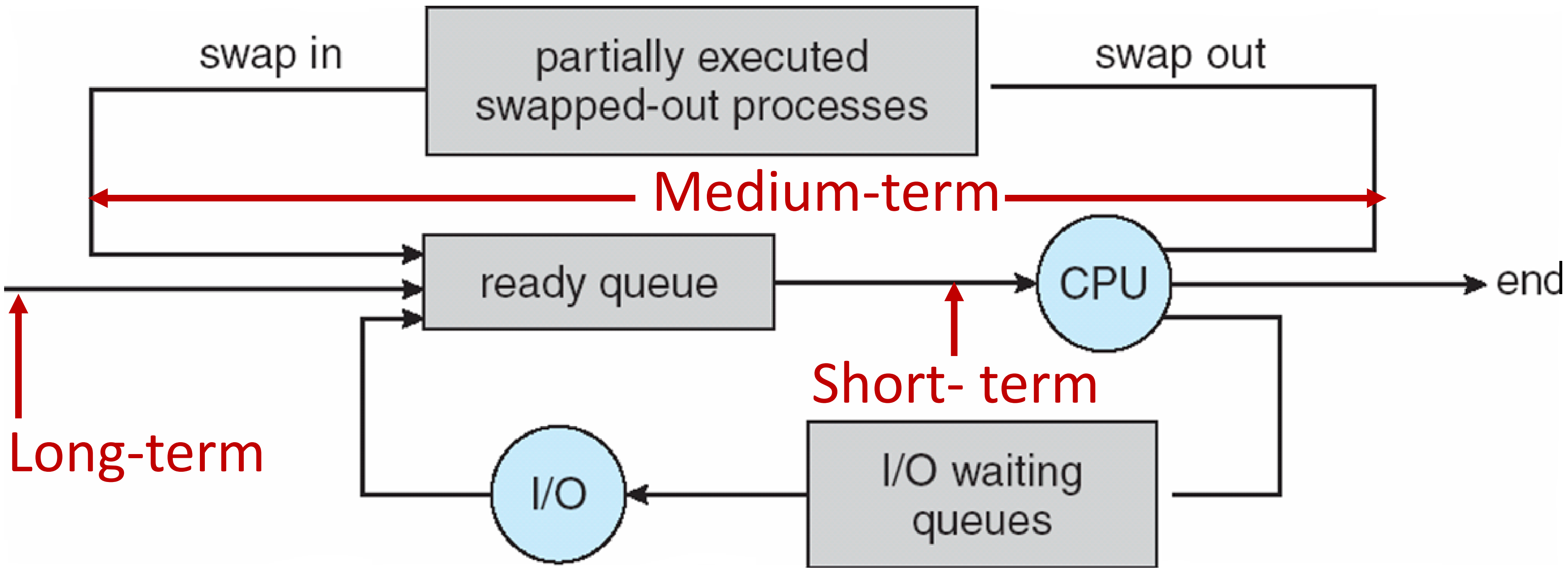


Mid-term scheduler is responsible to swap some processes out of memory or CPU usage

Data structures to store the swapped out processes

Medium-Term Scheduling

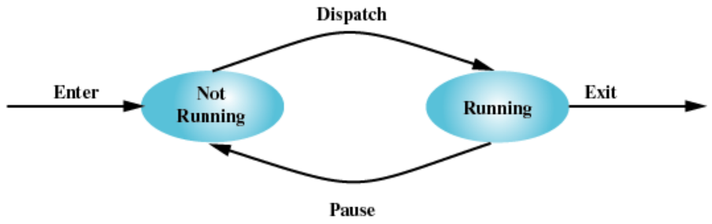
— Addition of Medium Term Scheduling



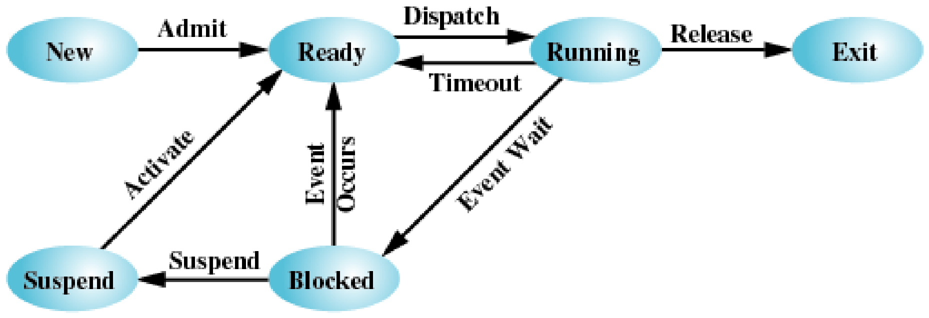
* **Queueing diagram** : A common representation for discussion for of process scheduling is a queueing diagram

Summary: Sketch of the control for processes

- The execution of a process goes through several states
  - New, Ready, Running, Waiting, Terminated [**Five state model**]
- There are many different models for state transitions
  - **<u>Two state model:</u>**
    - Running, Not-running
  - Three state model:
    - Ready, Running, blocked
  - Five state model
  - **<u>Six state model</u>**
  - Other models



(a) State transition diagram



(a) With One Suspend State

Suspended Processes

- Processor is faster than I/O so all processes could be waiting for I/O
- **Swap** these processes **to disk** to free up more memory
- Blocked state becomes suspend state when **swapped to disk**
- Two new states
  - Blocked/Suspend
  - Ready/Suspe

The strategy to select which process is swapped out, corresponds to the **midterm scheduling** [中期调度]

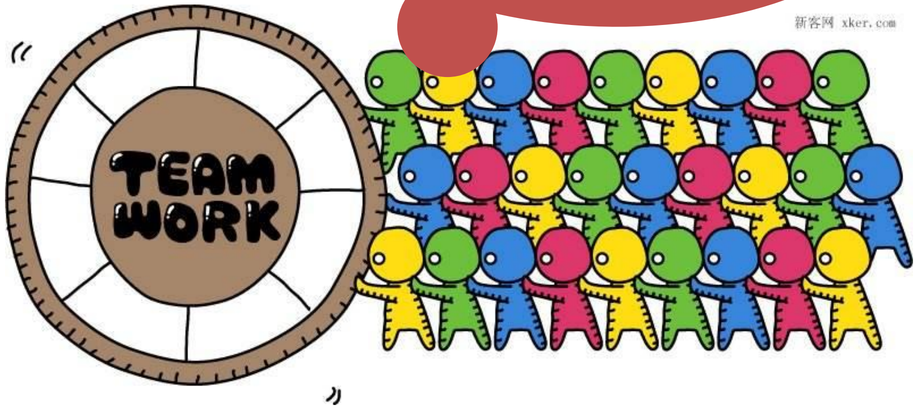
Process

- To understand the execution of your program
  - Process [进程] is the traditional concept
    - The identification to manage the needed information to run one program
      - **PCB** is the data structure to record the necessary information: resources (MM, ownership, security, …), execution stages/states, …
    - Additional data structures and algorithms are needed to manage the concurrent execution of many programs
      - Queues, and schedulers
      - Inter-process communication (IPC)
  - Thread[线程] is the modern concept
    - The idea could be seen as **MULTIPLEX**ing **process,** *namely that your program is constructed to have more than one execution units*
      - CPU is occupied by the process, however the usage of the CPU is shared among the internal execution units (threads)
        - » The resources assigned to the process could be shared by those threads.

Concurrency **OF COURSE** benefits users!
(?- only if well controlled)

- Concurrent processes executing in the operating system allows for the processes to cooperate (both mu… with other processes
  - The simplest example of how th… processes are using the sa…
- Reasons for cooperating pro…
  - Several processes may need to access th… stored in a file)
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

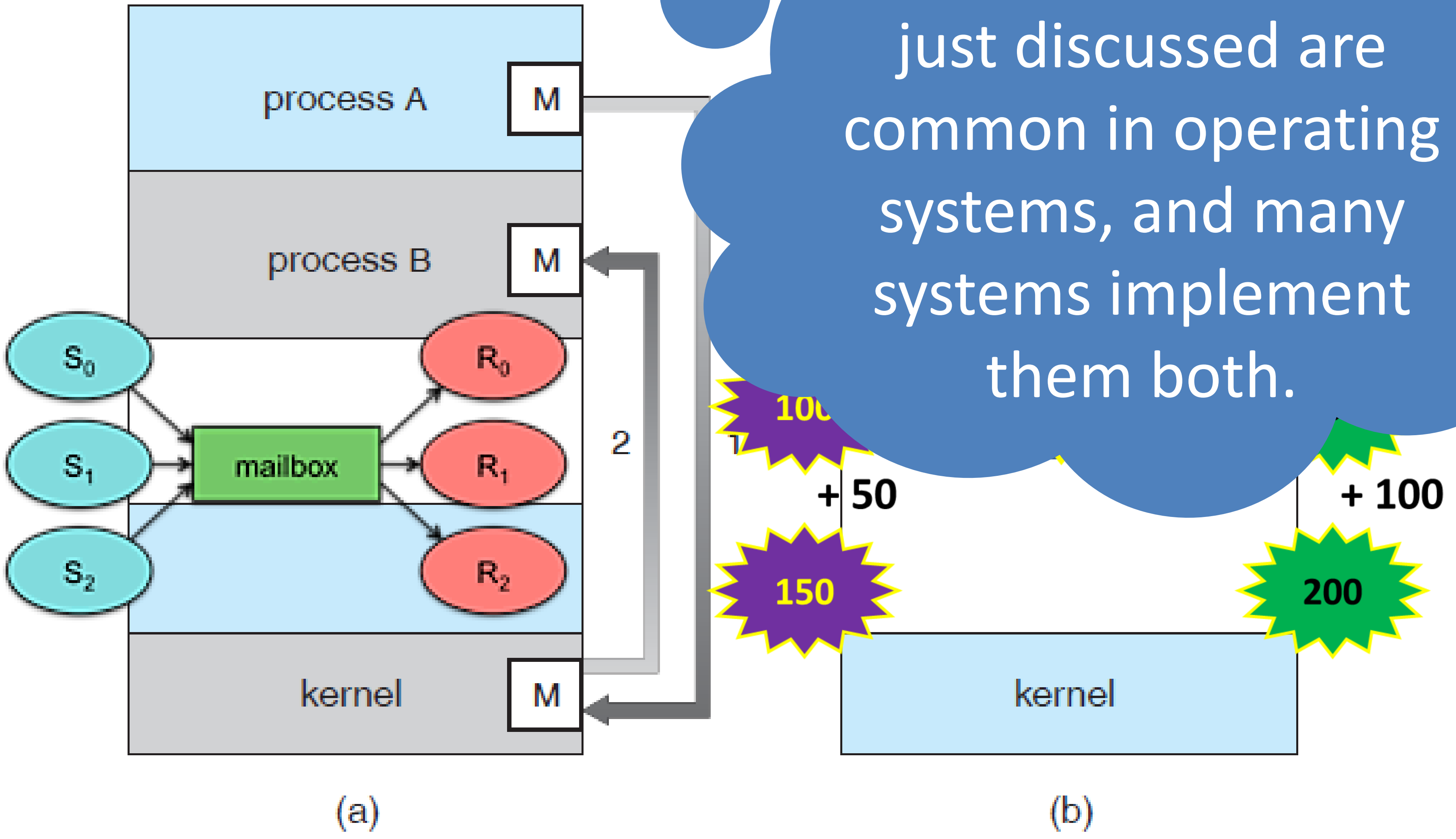
This is of course not FREE! The cooperation leads to complexity – deadlock and data inconsistency in later chapters



IPC: Inter-Process Communication

- Cooperating processes require an inter-process communication (IPC) mechanism that will allow them to exchange data and information.
- There are two fundamental models of inter-process communication:
  - **Shared memory**
  - **Message passing**
    - message passing interfaces, mailboxes and message queues
    - sockets, STREAMS, pipes

IPC: two fundamental types



Both of the models just discussed are common in operating systems, and many systems implement them both.

Figure 3.13 Communications models. (a) Message passing. (b) Shared memory.

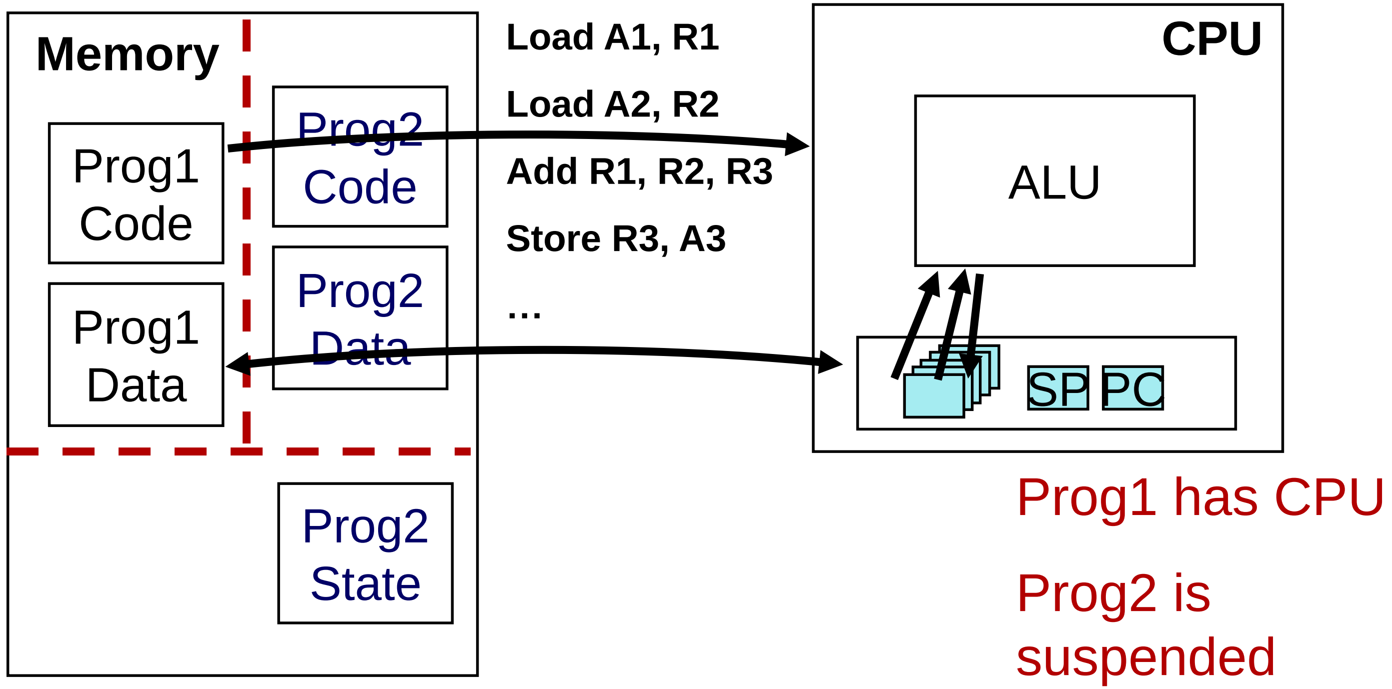
Process

- To understand the execution of your program
  - Process [进程] is the traditional concept
    - The identification to manage the needed information to run one program
      - **<u>PCB</u>** is the data structure to record the necessary information: resources (MM, ownership, security, …), execution stages/states, …
    - Additional data structures and algorithms are needed to manage the concurrent execution of many programs
      - Queues, and schedulers
      - Inter-process communication (IPC)
  - Thread[线程] is the modern concept
    - The idea could be seen as **<u>MULTIPLEX</u>**ing **process,** *namely that your program is constructed to have more than one execution units*
      - CPU is occupied by the process, however the usage of the CPU is shared among the internal execution units (threads)
        - » The resources assigned to the process could be shared by those threads.

Process switching is EXPENSIVE!

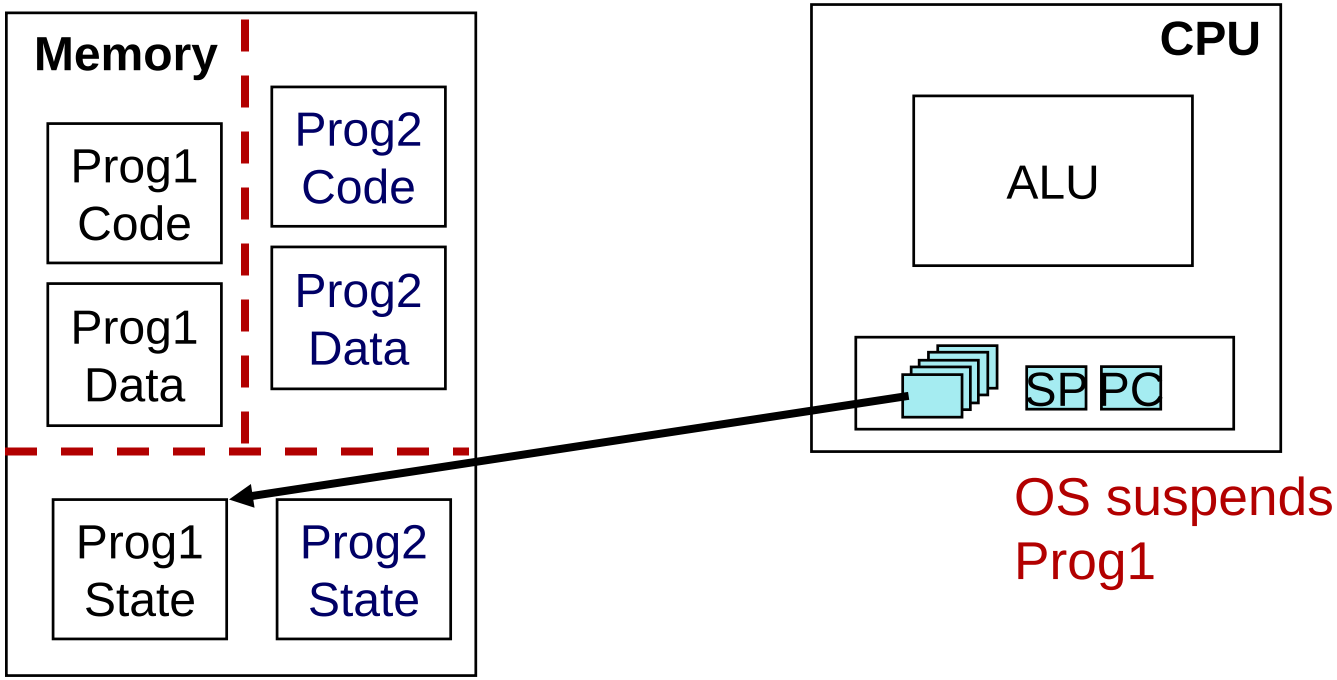
Context Switching

- Program instructions operate on operands in memory and (temporarily) in registers



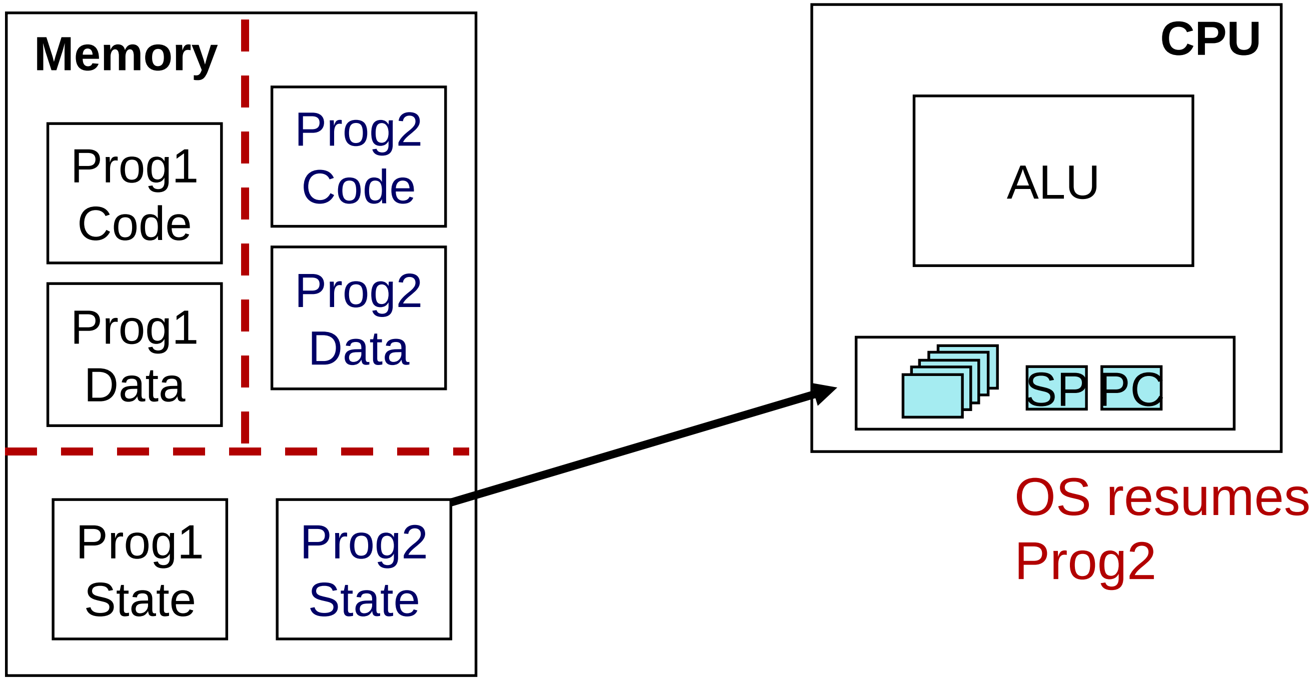
Context Switching

- Saving all the information about a process allows a process to be temporarily suspended and later resumed from the same point



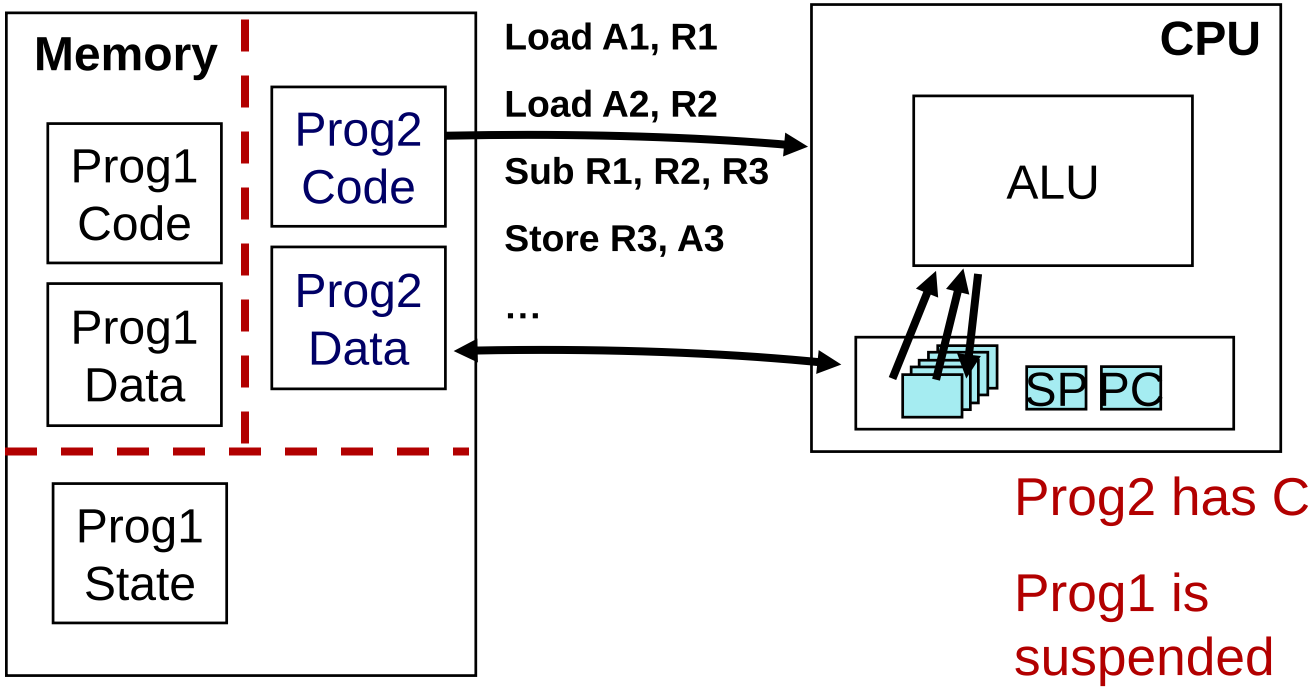
Context Switching

- Saving all the information about a process allows a process to be temporarily suspended and later resumed



Context Switching

- Program instructions operate on operands in memory and in registers



Prog2 has CPU

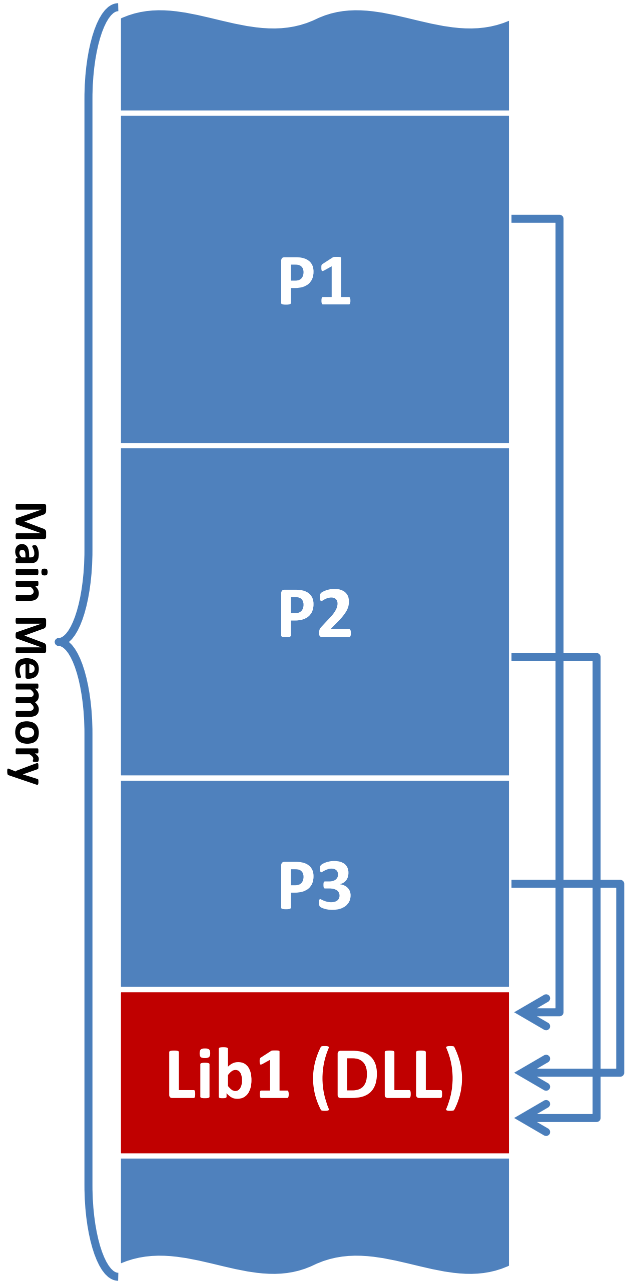
Prog1 is suspended

How to overcome the cost? SHARE

- Analogous with DLL

- Three programs (P1, P2, P3) share same DLL (lib1)
- Namely, there is only a copy of the functions defined in lib1 in the MM
- And, those programs may access different function in lib1

➔ So, each program should remember some information of the target functions (called CONTEXT[上下文]) in lib1



SHARE + CONTEXT: multiple services in one program

- We can use **1** CPU + **1** MM space + **1** HD to support MULTIPROGRAMMING – the basis of modern OSs, namely concurrently running many processes
- How about providing multiple services with one program? (MULTISERVING?)
  - If so, there is only one copy of the instructions of those server programs or MS Word in MM, which can provide services for many users/requests

← This is obviously **economical** and **efficient** way!

Here comes **thread**

- Concept of Process has two facets[ˈfasit/ˈfasət].
- A Process is:
  - A Unit of resource ownership – process is allocated:
    - a virtual address space for the process image
    - control of some resources (files, I/O devices...)
  - A Unit of execution/dispatching - process is an execution path through one or more programs (functions, code segments)
    - may be interleaved with other processes
    - execution state (Ready, Running, Blocked...) and dispatching priority

PPTs from others\flame.cs.dal.ca_~hawkey_3120\June30Threads.ppt

- These two characteristics are treated separately by some recent operating systems:
  - The unit of <u>**resource ownership**</u> is referred to as a **Task** or (for historical reasons) also as a **Process**.
  - <u>**The unit of dispatching**</u> is referred to a **Thread**.

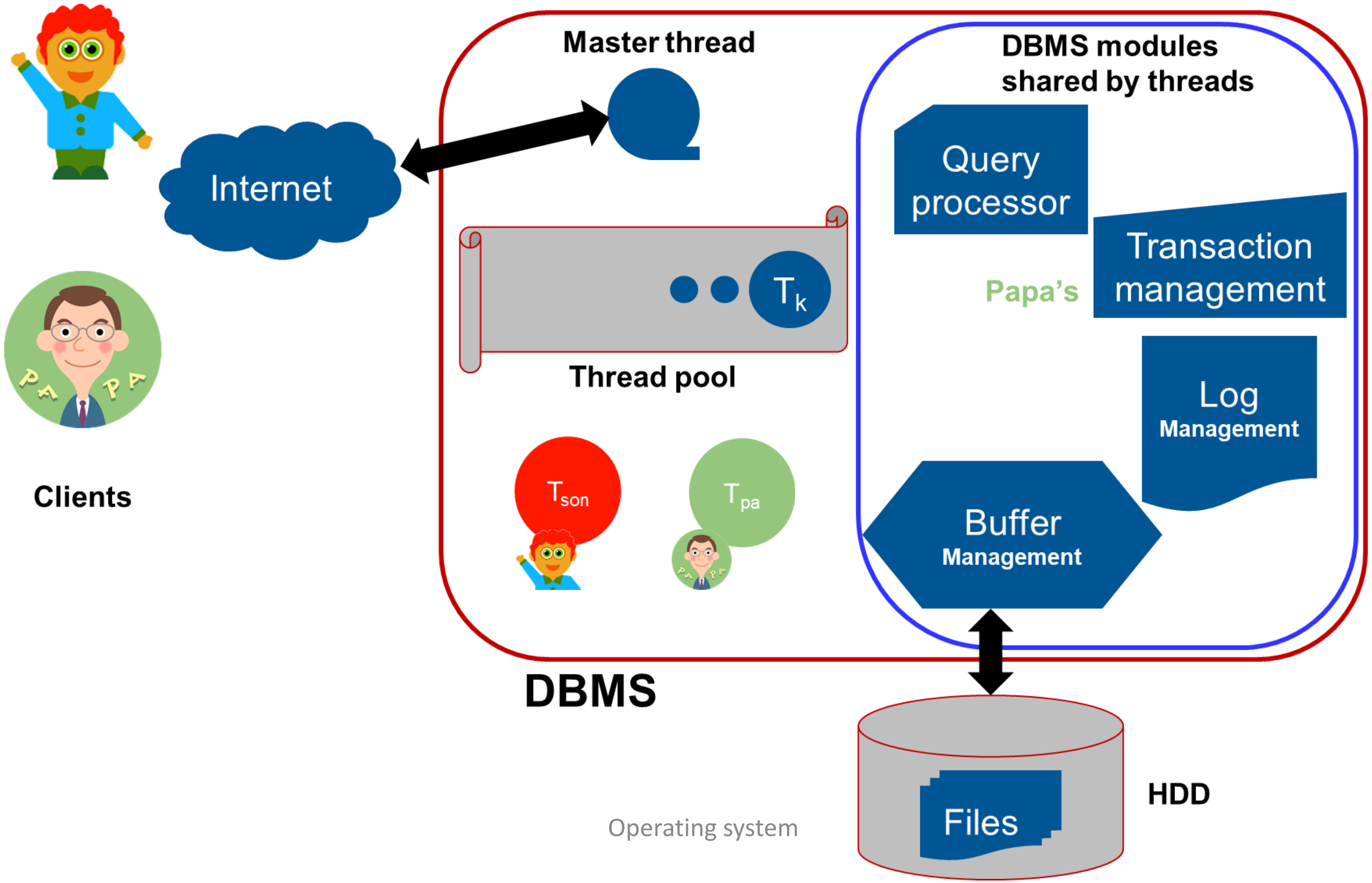
➔ A thread is an **execution unit** inside a process/program, which could be scheduled directly for CPU (this depends on the OS)

  - Several threads can exist as services in the same task/process.

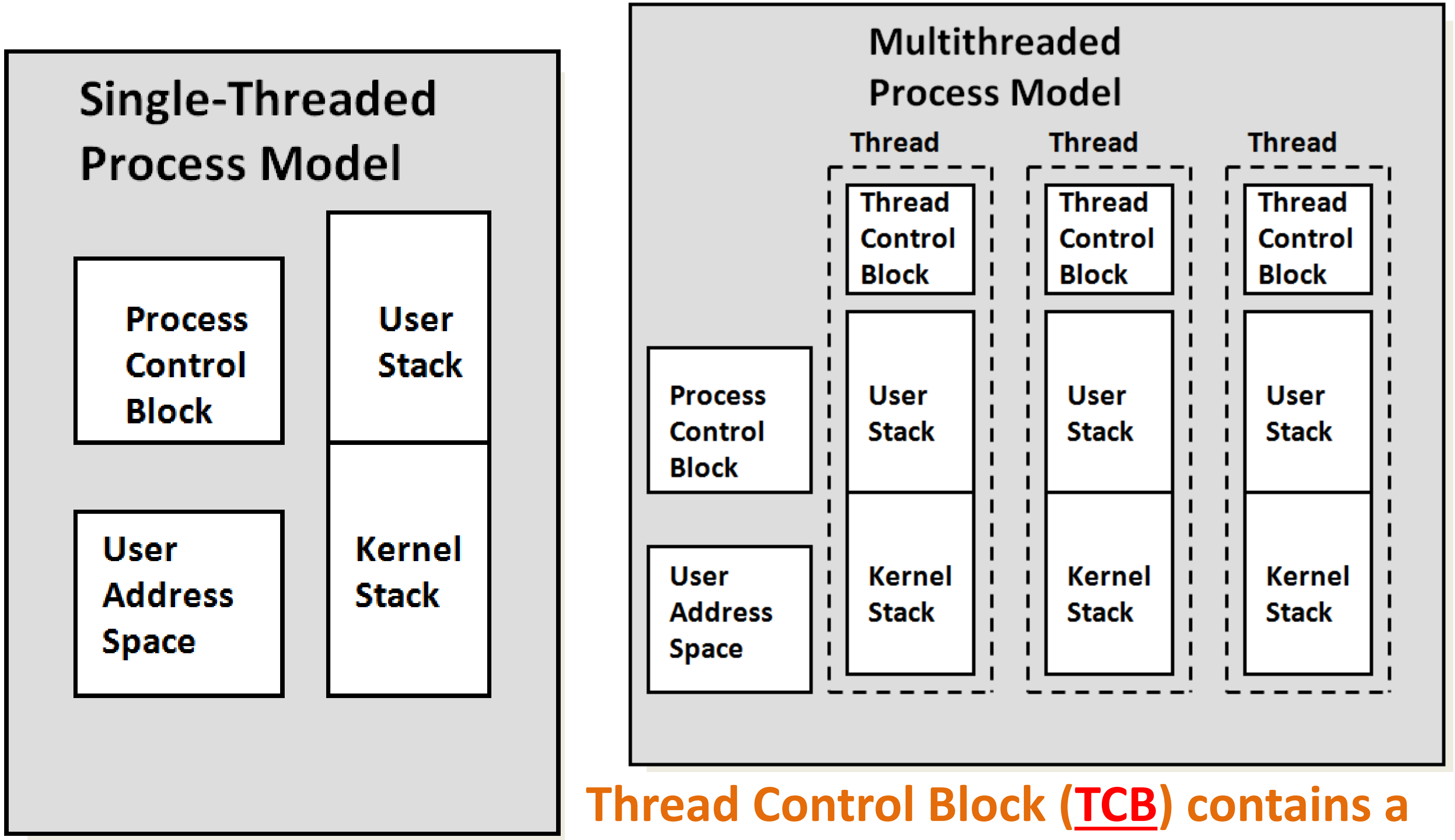
PPTs from others\flame.cs.dal.ca_~hawkey_3120\June30Threads.ppt
PPTs from others\From Ariel J. Frank\OS381\os3-3_pro.ppt

Thread now is popular, especially for high performance servers

- Modern DBMS – Java Thread (Pool) – **MySQL**

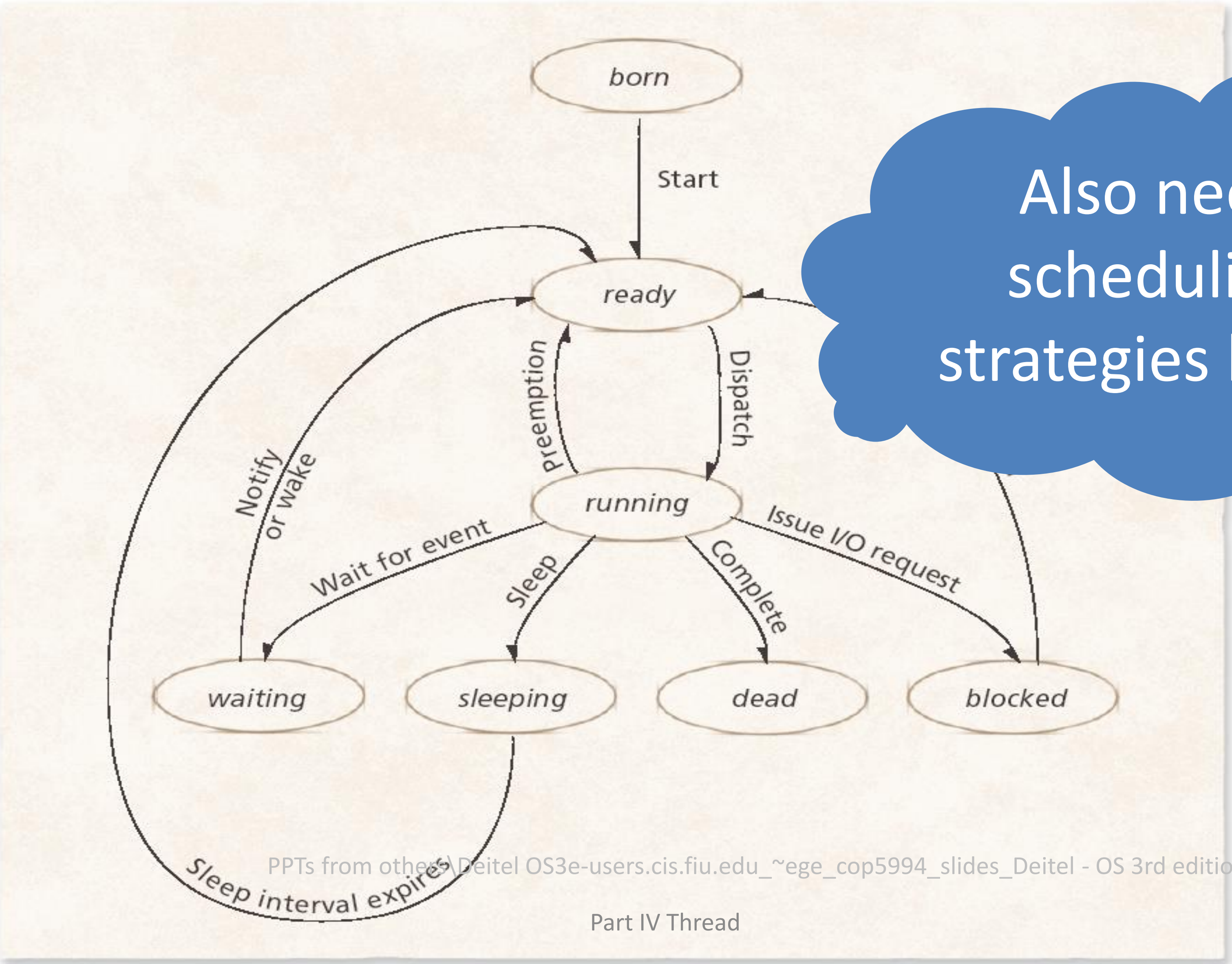


Single Threaded and Multithreaded Process Models



Thread Control Block (TCB) contains a register image, thread priority and thread state information

Thread States: Life Cycle of a Thread



Also need scheduling strategies here!

PPTs from others\Deitel OS3e-users.cis.fiu.edu_~ege_cop5994_slides_Deitel - OS 3rd edition\OS3e_04.ppt

Thread Operations

- Threads and processes have common operations
  - **Create**, **Exit** (terminate), **Suspend**, **Resume**, **Sleep**, **Wake**
- Thread operations do not correspond precisely to process operations
  - Cancel
    - Indicates that a thread should be terminated, but does not guarantee that the thread will be terminated
    - Threads can mask the cancellation signal
  - Join
    - A primary thread can wait for all other threads to exit by joining them
    - The joining thread blocks until the thread it joined exits

PPTs from others\Deitel OS3e-users.cis.fiu.edu_~ege_cop5994_slides_Deitel - OS 3rd edition\OS3e_04.ppt

Thread libraries

- To implement threads, a threading library (either in **user space** or **kernel space**) is responsible for handling the saving and switching of the execution context from one thread to another.
  – We have ULT (User-Level Thread) and KLT (Kernel-Level Thread).
- Multithreaded application

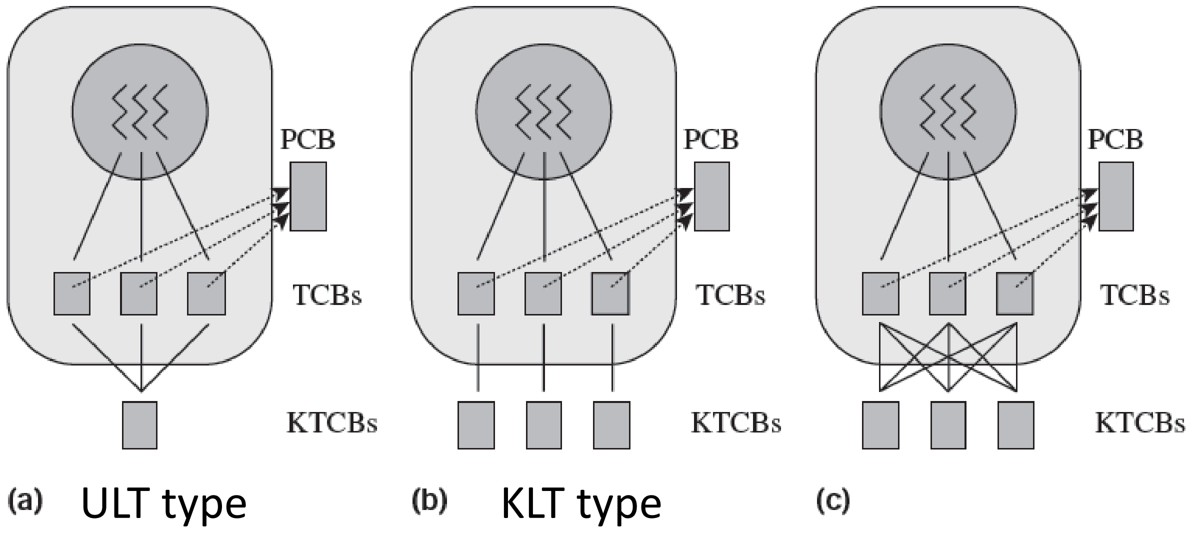
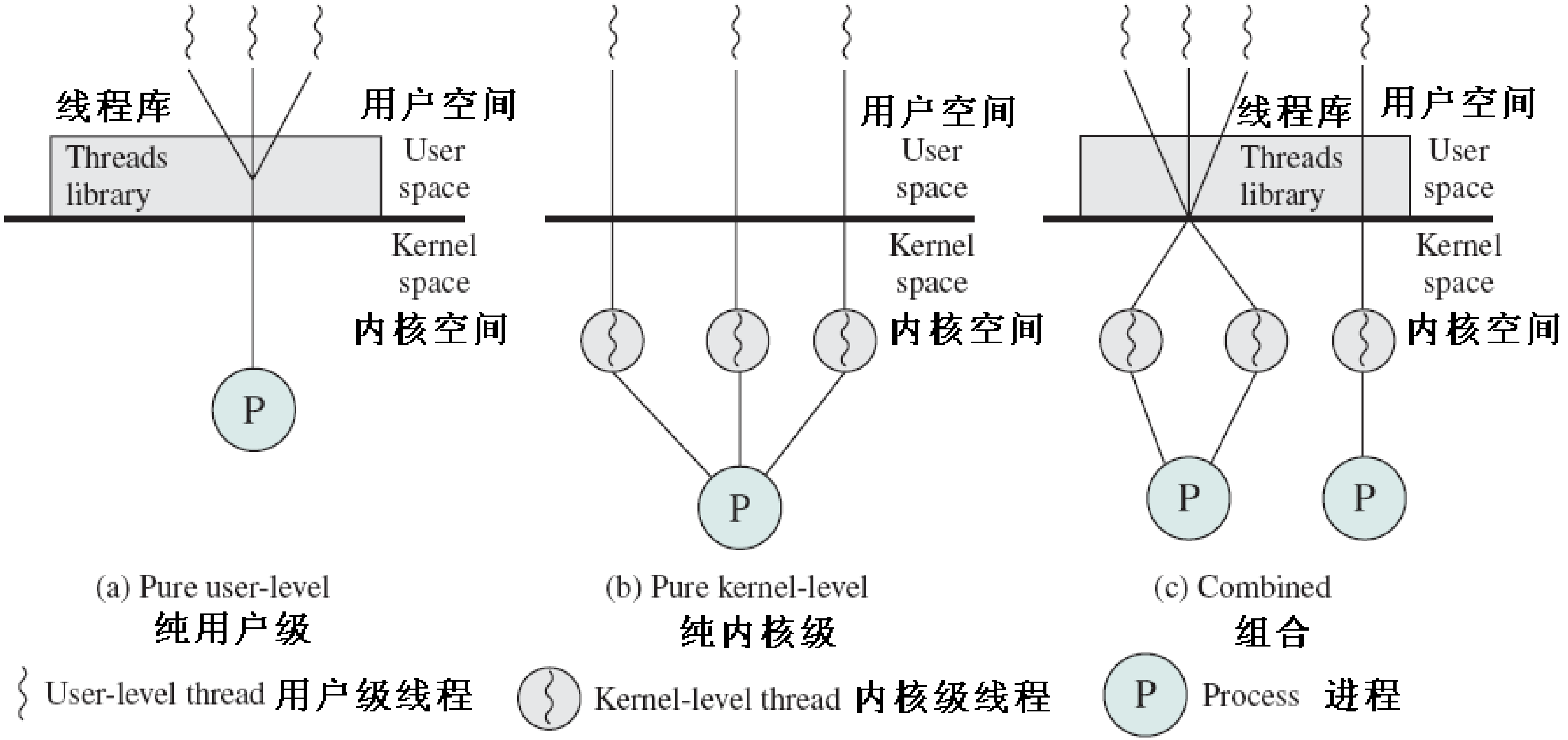


Figure 5.17 (a) Many-to-one; (b) one-to-one; (c) many-to-many associations in hybrid threads.



User-Level and Kernel-Level Threads

用户级与内核级线程

Threads library

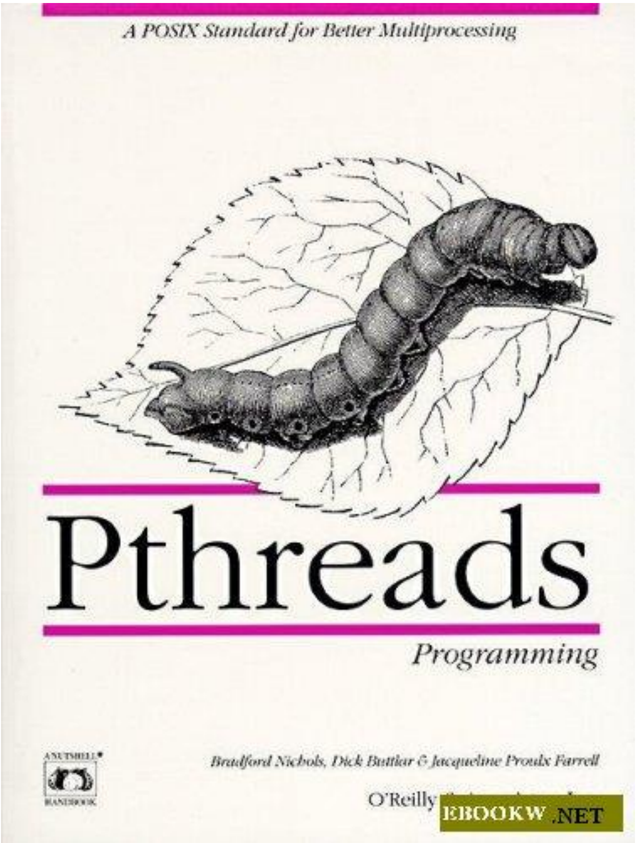
- Contains code for:
  – creating and destroying threads
  – passing messages and data between threads
  – scheduling thread execution
    - pass control from one thread to another
  – saving and restoring thread contexts

- ULT can be implemented on any Operating System, because no kernel services are required to support them
  – POSIX Pthreads, Mach C-threads, Solaris UI-threads

PPTs from others\flame.cs.dal.ca_~hawkey_3120\June30Threads.ppt

POSIX and Pthreads

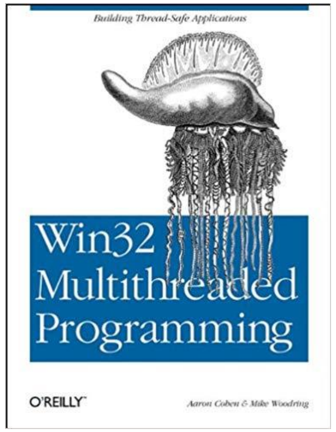
- **Pthreads**, the threads extension of the POSIX standard, may be provided as either a user- or kernel-level library
  - POSIX states that processor registers, stack and signal mask are maintained individually for each thread
  - POSIX specifies how operating systems should deliver signals to Pthreads in addition to specifying several thread-cancellation modes

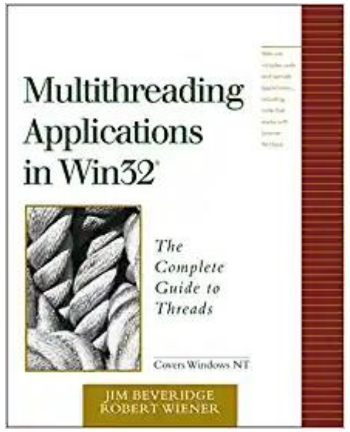


PPTs from others\Deitel OS3e-users.cis.fiu.edu_~ege_cop5994_slides_Deitel - OS 3rd edition\OS3e_04.ppt

Win32 Threads (Windows XP)

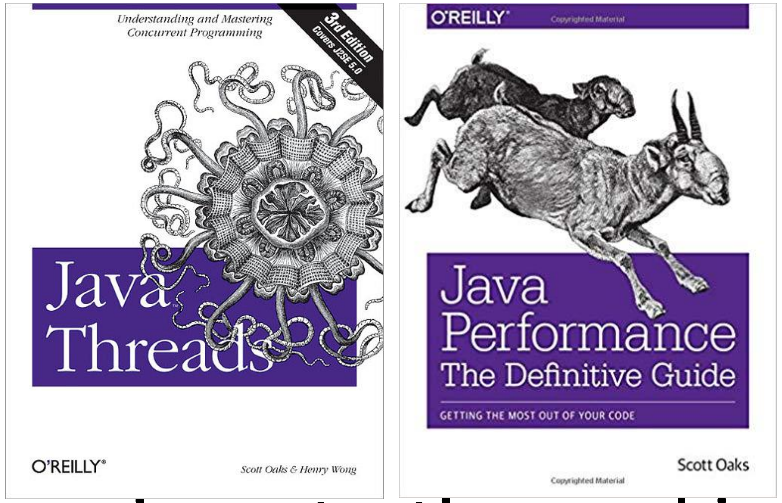
- The **Win32** thread library is a kernel-level library available on Windows systems
  - Actual unit of execution dispatched to a processor
  - Execute a piece of the process’s code in the process’s context, using the process’s resources
  - Execution context contains
    - Runtime stack
    - State of the machine’s registers
    - Several attributes





Do you remember Java Threads?

- Java allows the application programmer to create threads that can port to many computing platforms
- Threads
  - Created by class `Thread`
  - Execute code specified in a `Runnable` object's `run` method



- Java supports operations such as naming, starting and joining threads

PPTs from others\Deitel OS3e-users.cis.fiu.edu_~ege_cop5994_slides_Deitel - OS 3rd edition\OS3e_04.ppt

- The **Java** thread API allows threads to be created and managed directly in Java programs
  - However, because inmost instances the JVM is running on top of a host operating system, the Java thread API is generally implemented using a thread library available on the host system.

→This means that on Windows systems, Java threads are typically implemented using the Win32 API; UNIX and Linux systems often use Pthreads

Java Threads

- Example showing interleaved thread execution :

```
class SimpleThread extends Thread {
    public SimpleThread (String str) {          // superclass constructor
        super (str);
    }
    public void run () {
        for (int i=0; i<10; i++) {
            System.out.println (i + " " + getName() );
            try { sleep ((int) (Math.random () * 1000));
            catch (InterruptedException e) { }
        }
        System.out.println ("Finished " + getName () );
    }
}

class TwoThreadsTest {
    public static void main (String[ ] args) {
        new SimpleThread ("Edinburgh").start ();
        new SimpleThread ("Glasgow").start ();
    }
}
```

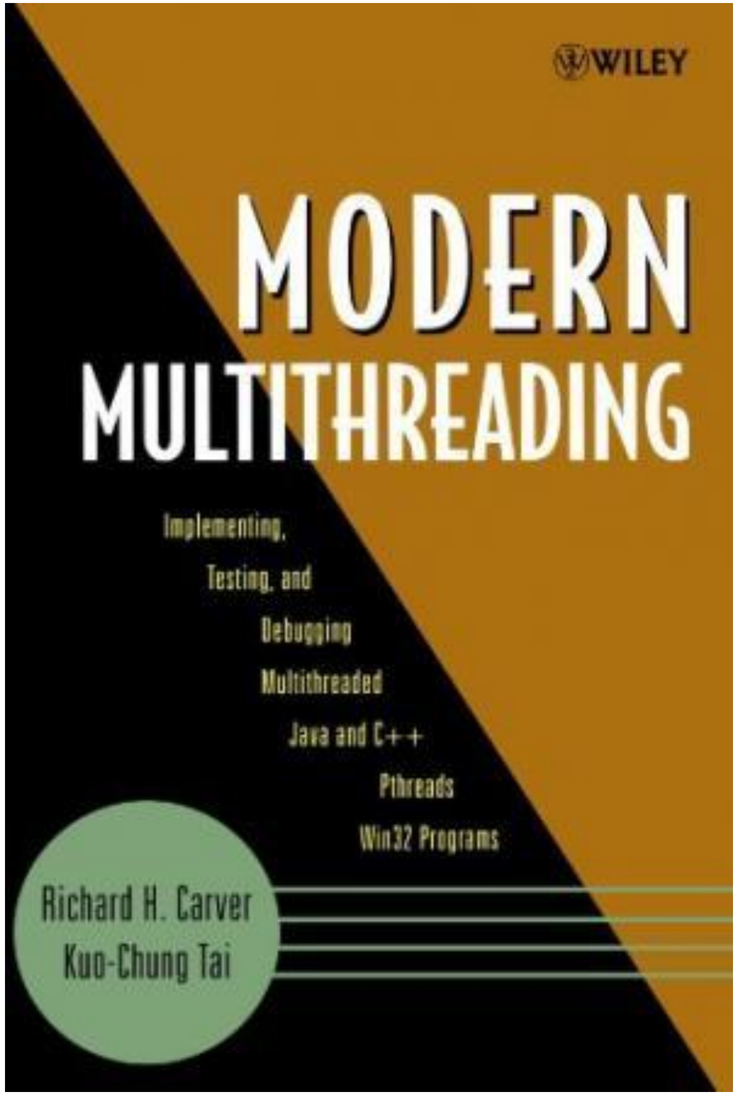
– main method starts two threads by calling the start method

- output something like :

0 Edinburgh
0 Glasgow
1 Glasgow
1 Edinburgh
2 Edinburgh
3 Edinburgh
2 Glasgow
3 Glasgow
4 Glasgow
4 Edinburgh
5 Glasgow
5 Edinburgh
6 Glasgow
7 Glasgow
8 Glasgow
6 Edinburgh
7 Edinburgh
8 Edinburgh
9 Edinburgh
Finished Edinburgh
9 Glasgow
Finished Glasgow

You can use the experiment framework of this course to simulate this , right?

PPTs from others\www.dcs.ed.ac.uk_teaching_cs3_osslides\thread.ppt



- Modern Multithreading: Implementing, Testing, and Debugging Multithreaded Java and C++/Pthreads/Win32
- *Richard H. Carver*, *Kuo-Chung Tai*
- 2005