Operating system

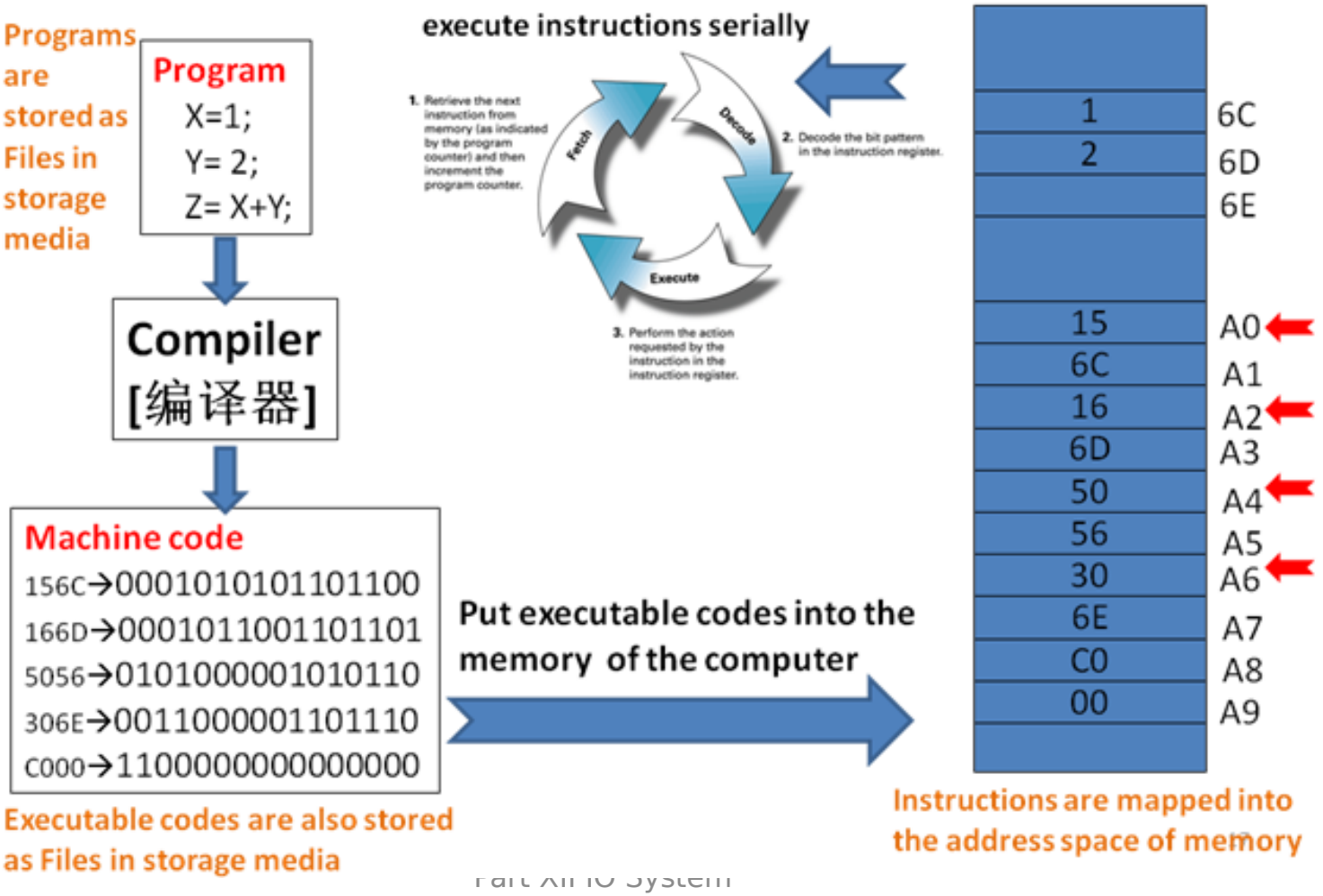
Review: OS's big picture now

Review

- Sketch of OS
  - Infinite repetition, 4 components, services for others
- Understand the **execution** first
  - CPU and controller as special chips which could understand and execute the instructions (together with other parameters)
- 2 **mappings** sharing similar scheme
  - from logic file space (a finite collection of bytes) into linear addressed space (frames, blocks)

The power of computer systems

- To execute programs!

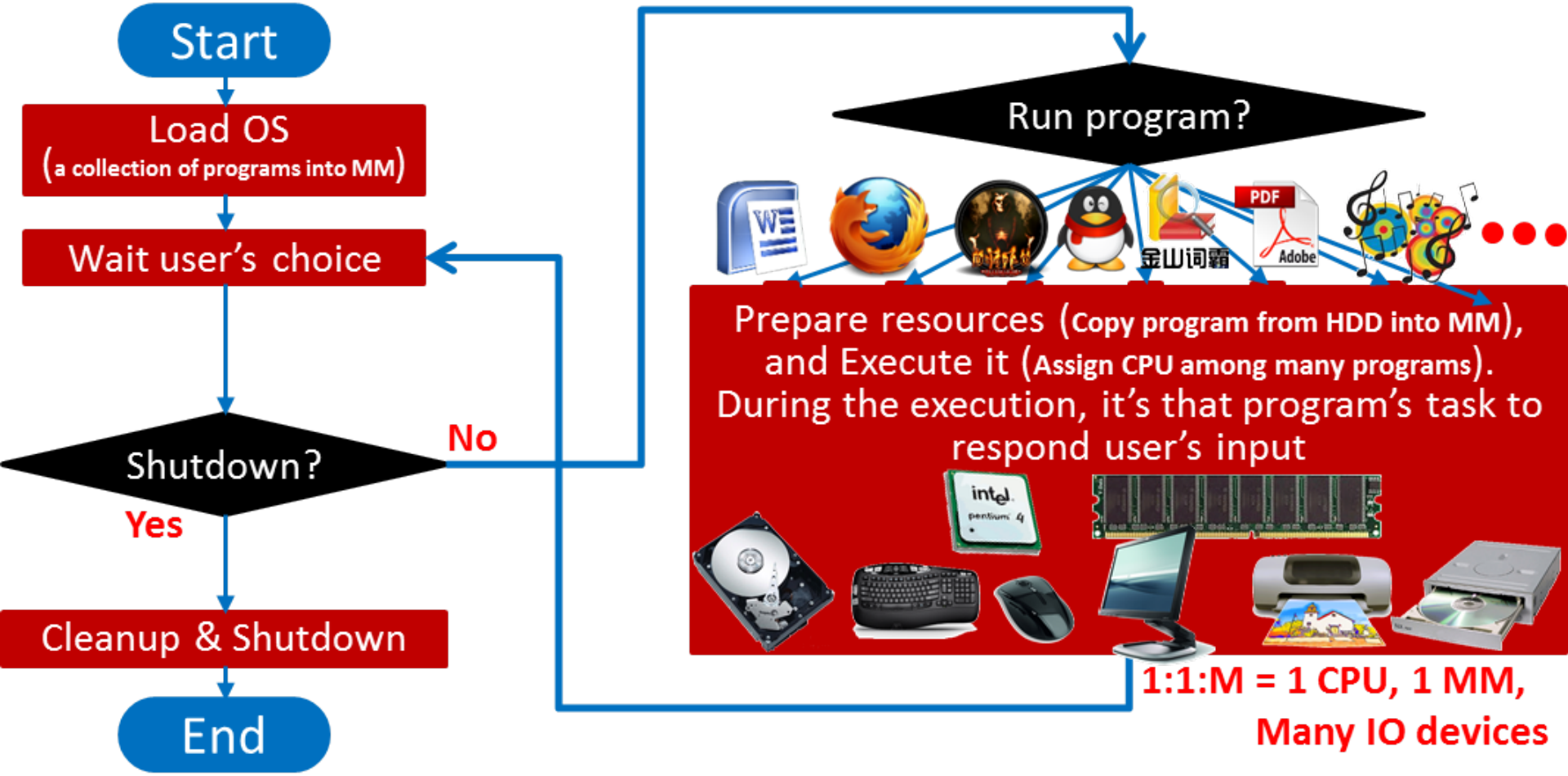


Definition of OS

- The software/program which contains a collection of many routines (functions, programs) to support the **automatic execution** of many **cooperated and concurrent programs**
- Many subtasks should be considered
  - **EMM**
    - **E**xecution: how is your program run?
    - **M**apping **2**: locate the program files (instructions and data) in Hdisk
    - **M**apping **1**: copy the selected program files (instructions and data) from Hdisk into appropriate regions in MM
  - **GSD**: **G**UI, **S**ecurity, **D**istributed

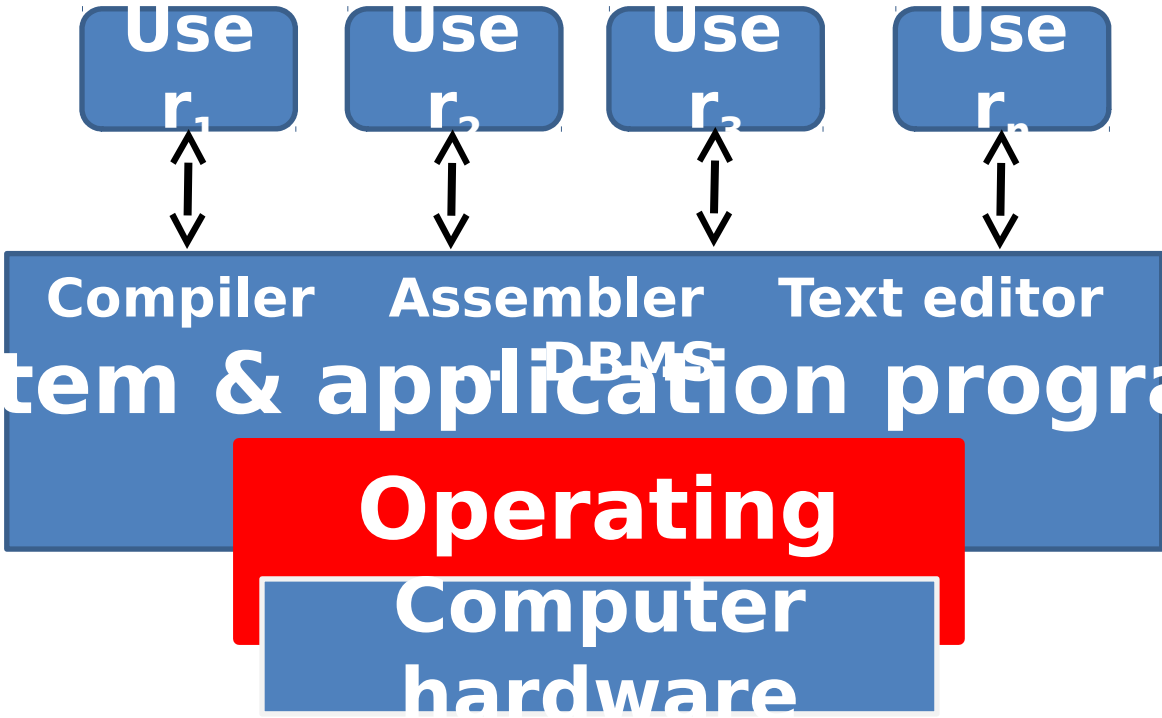
Helpful diagram to understand OS

- **<u>Infinite repetition</u>** controlled by users

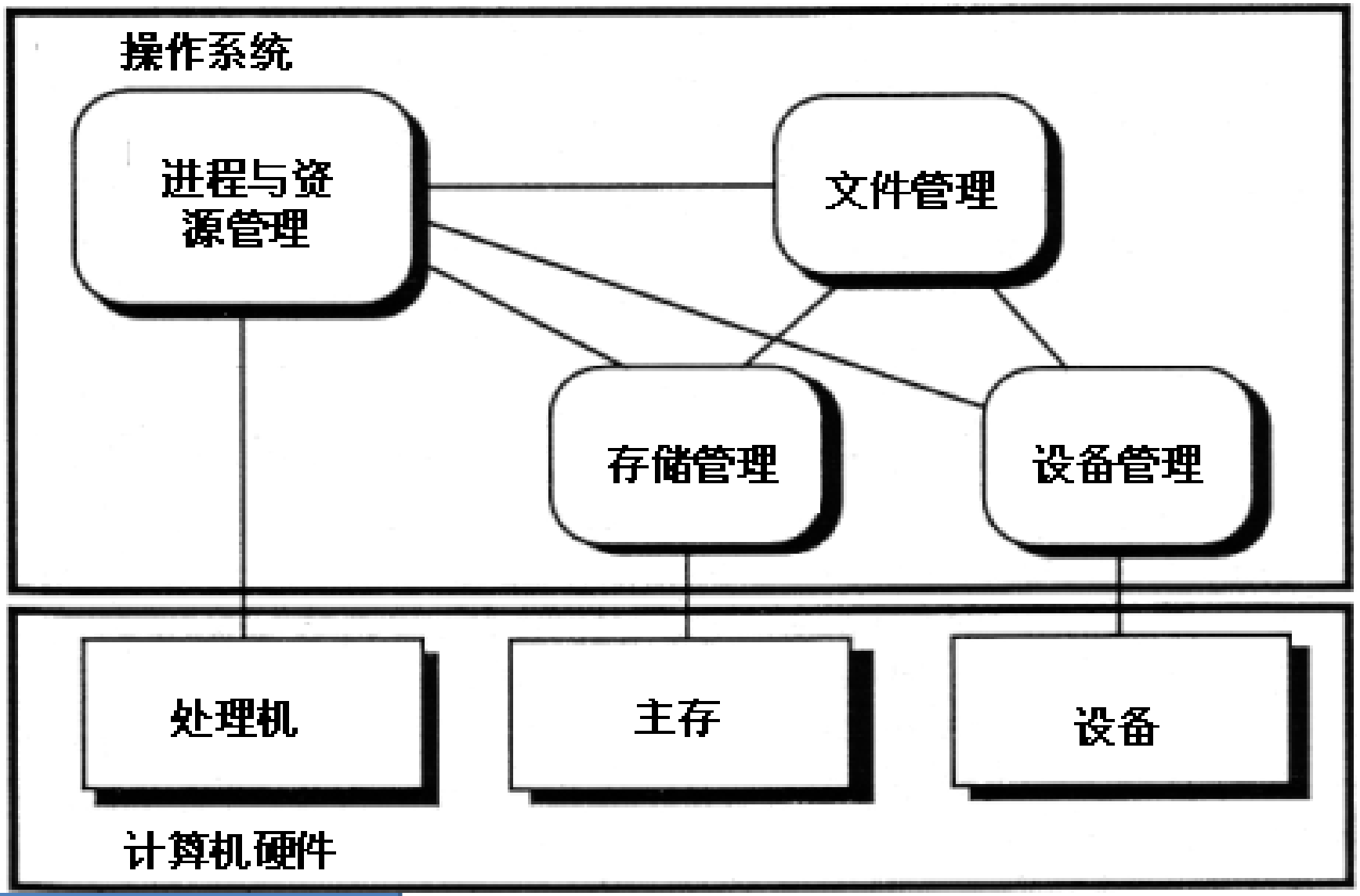


The roles of OS

- Friendly **interface** for the users
  - Files, GUI
- Efficient and safe **manager** for the resources
  - Storage media
    - I/O devices
  - Memory
  - CPU



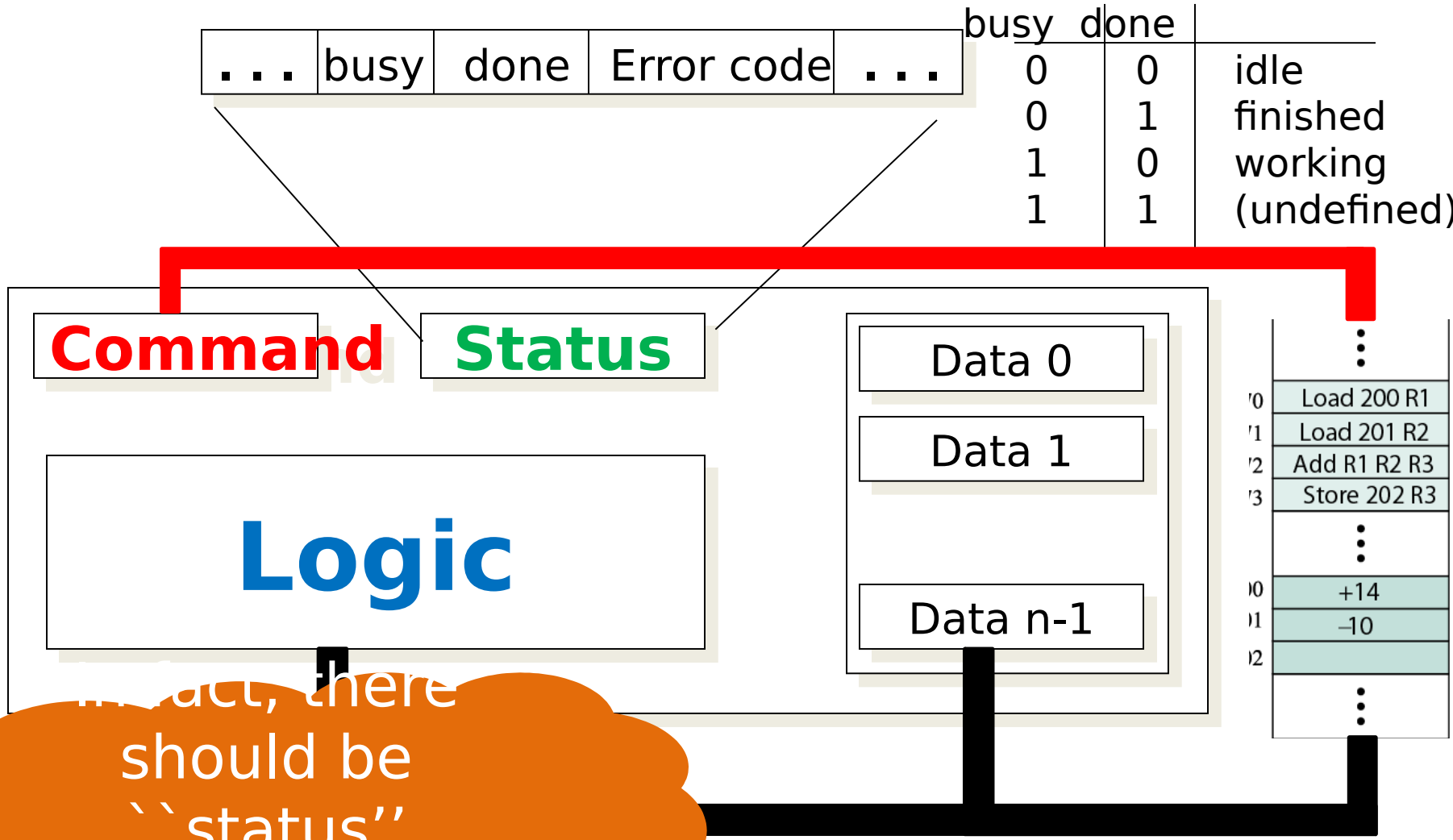
Four fundamental components of modern OS



Review

- Sketch of OS
  - Infinite repetition, 4 components, services for others
- Understand the **execution** first
  - CPU and controller as special chips which could understand and execute the instructions (together with other parameters)
- 2 **mappings** share similar scheme
  - from logic file space (a finite collection of bytes) into linear addressed space (frames, blocks)

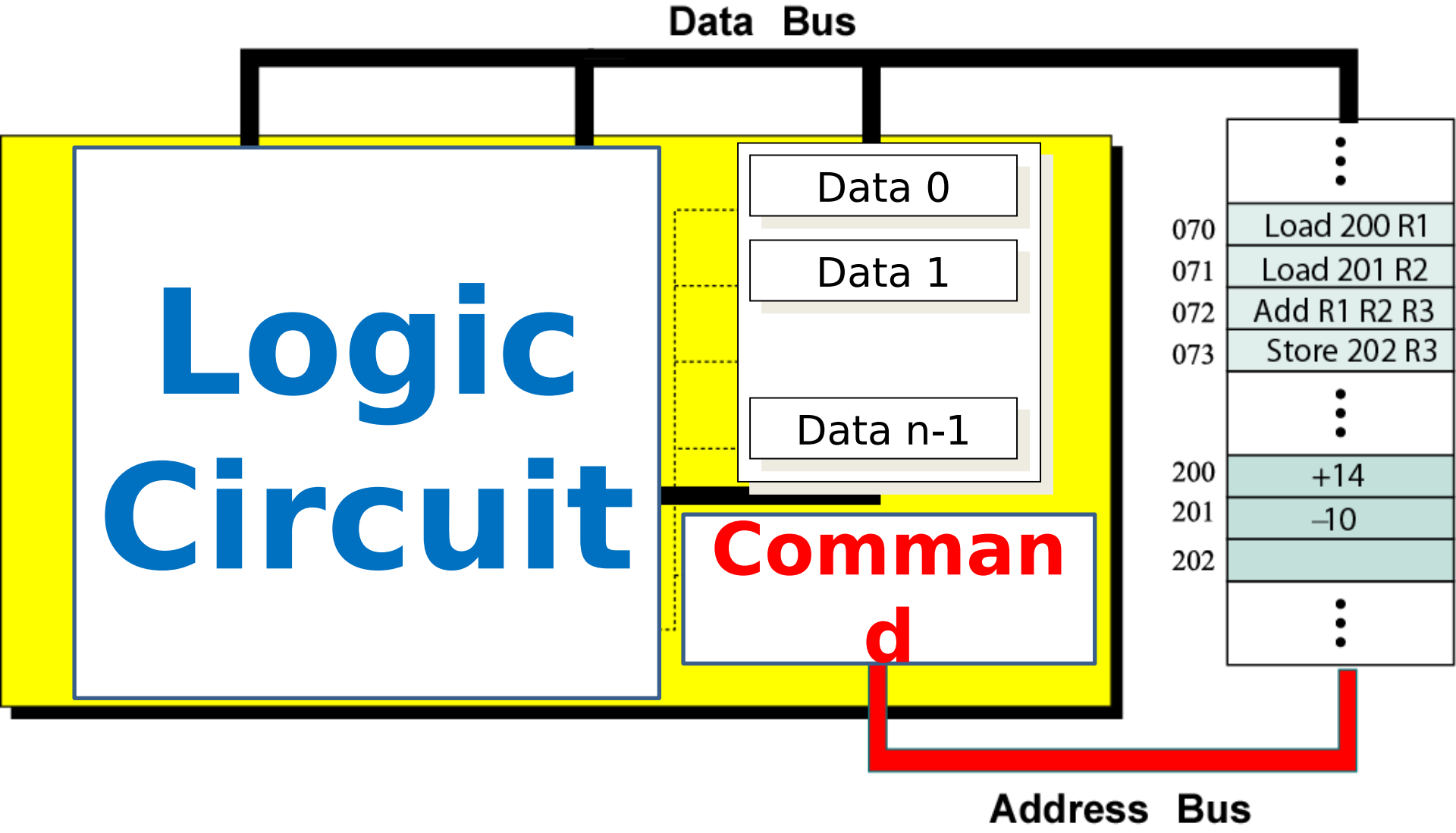
Device Controller Interface



For controller, Instructions and parameters are kept in some MM region – called ports

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

CPU


Data Bus
Logic Circuit
Data 0
Data 1
Data n-1
Command
070 Load 200 R1
071 Load 201 R2
072 Add R1 R2 R3
073 Store 202 R3
200 +14
201 –10
202
Address Bus

Executing instructions

- Instructions are transferred from MM into ` `command’ ’ register one after another
  - Parameters are also transferred into correspon ding registers when needed
- The predefined instruction could trigger rel ated electronic circuits to carry out corresp onding function
  - Like addition, multiplication, minus, etc
  - Also **read**/**write** data fro

I hope you remember these learned from CO course

Executing your program

- To run your program, executable file shoul d be gotten usually by compilation
  - Executable file contains the instructions and da ta you define when you do programming
- Those instructions are transferred to corres ponding registers in CPU/controller one aft er another, …
- And you’ ve learned h
  decoded and exe

your executable program is conveyed from hdisk into MM. – 2

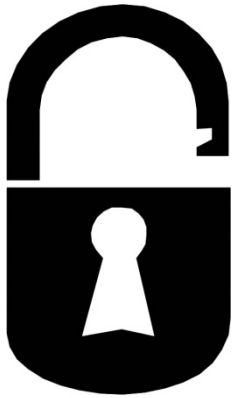
Executing cooperated & concurrent programs

- If there is **no controlled access** to shared data, execution of the processes on these data can interleave **Cooperation**.
  - **The results will then depend on the order in which data were modified ☾ Data Inconsistency Ⓟ Synchronization**
    - i.e. the results are non-deterministic.
- Concurrent processes (or threads) often need to **share data** (maintained either in shared memory or files) and **resources**
  - If there is no proper policy to assign resources among processes, it may result in that all the processes get blocked ☾ **Deadlock** [死锁]



Synchronization – All for **lock mechanism**

- The general layout is of **lock mechanism** is:



```
do {
    acquire lock
        critical section
    release lock
        remainder section
} while (TRUE);
```

General rules to cope with CS problem using semaphores

1. Find the types of actors
   – To determine the **processes**
2. Recognize the shared resources between actors
3. Infer the constraints based on the situations when actors use those shared resources
   – **ME or SCH**?
     - To determine semaphores and their initial values
     - To determine the code (nested for ME, and scattered for SCH)
4. Use semaphores to finish those processes

Deadlock

- Four necessary conditions
  - Mutual Exclusion [互斥]
  - Hold-and-Wait [占有并等待]
  - No preemption [非抢占]
  - Circular Wait [循环等待]
- Strategies to overcome the deadlock situation
  - **Providing enough resources**

$$\sum (P_{\max} - 1) + 1 \leq R_{Total}$$

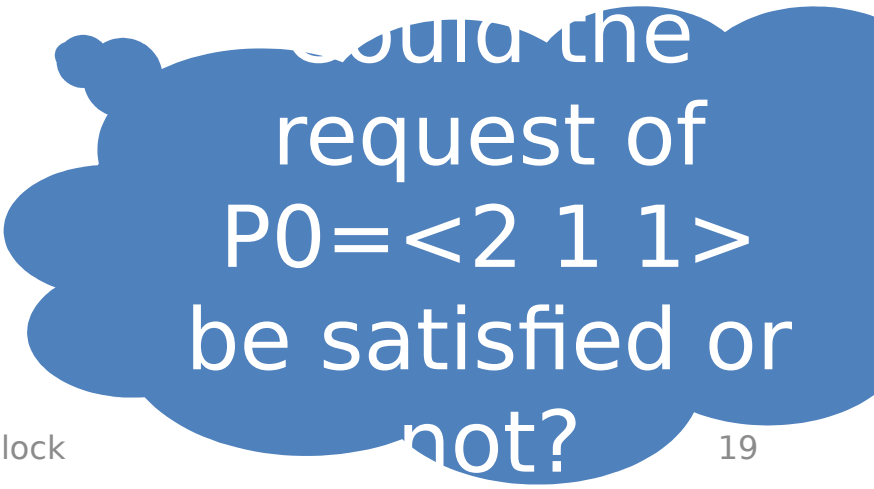
  - Staying Safe
    - **Preventing** Deadlocks
    - **Avoiding** Deadlocks ☾ **Banker’ s algorithm**!
  - Living Dangerously
    - **Keep blind** (Ostrich[鸵鸟] or **Head-in-the-Sand** algorithm)
    - **Detect** it and **Recover** from it.

Example 3:

- 5 processes P_0 through P_4;
  3 resource types:
  A (**10** instances), B (**5**instances), and C (**7** instances)

Snapshot at time T_0:

Allocation	*Max*	*Available*
A B C	*A B C*	*A B C*
P_0 0 1 0	7 5 3	**3 3 2**
P_1 2 0 0	3 2 2	
P_2 3 0 2	9 0 2	
P_3 2 1 1	2 2 2	
P_4 0 0 2	4 3 3	



Review

- Sketch of OS
  - Infinite repetition, 4 components, services for others
- Understand the **execution** first
  - CPU and controller as special chips which could understand and execute the instructions (together with other parameters)
- 2 **mappings** share similar scheme
  - from logic file space (a finite collection of bytes) into linear addressed space (frames, blocks)
    - Organize the basic storage units first
      - MM & Hdisk
    - keep necessary data structures (together with related operations) to carry out the mappings

2 spaces are similar

- Usually, your executable program is stored pe rmanently in Hdisk first
- To run, it should be copied into
- Hdisk and MM could be both conceived as tw o **linear addressed spaces**
  - Each primary storage unit is uniquely numbered.
    - Usually 1 byte for MM, and sector (512 bytes) for Hdisk
  - Those primary storage units are further organized into larger semantic regions
    - (Fixed or Variable) Partitions, Frames for MM
    - Partitions (logic drives), blocks for Hdisk

Basic tasks of 2 mappings

- 2 tasks when carrying out mappings (no m atter MM or Hdisk)
  - **Space Allocation (Relocation)**: find enough av ailable regions for your program
    - Enough available frames for MM, blocks for Hdisk
  - **Address translation (Relationship reserving)** : convert the logic relationship of records into physical relationship delegated by storing reco rds in connected physical regions
    - Through the physical addresses of those related reg ions

	Mapping 1: File to MM	Mapping 2: File to Hdisk
Space Allocation	**Fixed Partition** • Cut MM into partitions (equal or unequal) in advance • Place your program into the target partition **Variable partition** • Allocate MM according to your program **Overlay** **DLL (Dynamic Linking Library)** **Paging** • Cut MM and program into same sized regions (frame, page) • Paste needed pages into available frames **Segmenting** • Cut program into semantic regions • Allocate MM according to needed region	• Sector (C.H.S) ☾ Partitions (except MBR) ☾ Block space • Since file is usually also cut into pages whose size is same with the block, the allocation of the file into block space is similar as that of paging
Relationship	To compute physical address of an instruction is based on (**starting**	To compute physical address of a record

Review

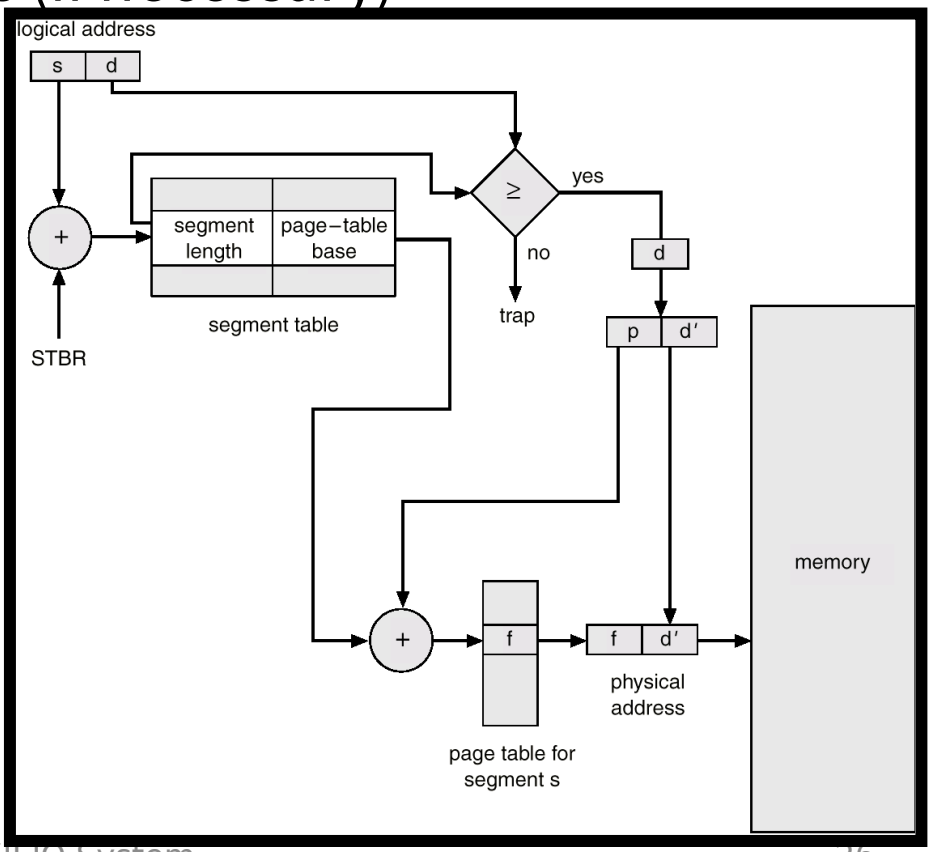
- Sketch of OS
  - Infinite repetition, 4 components, services for others
- Understand the **execution** first
  - CPU and controller as special chips which could understand and execute the instructions (together with other parameters)
- 2 **mappings** share similar scheme
  - from logic file space (a finite collection of bytes) into linear addressed space (frames, blocks)
    - Organize the basic storage units first
    - keep necessary data structures (together with related operations) to carry out the mappings

Data structures are needed to carry out those related tasks

- **Available regions**
  - Available partitions for partitioning, Available frames for paging/hybrid, available holes for segmenting, …
  - Organize sectors into semantic regions first, and Bit-vector, FAT for Hdisk, …
- **Mapping information** from file to target physical regions
  - Partition table, Page table, Segment table, …
  - Tree-structured directories + FCBs as File System for Hdisk, …

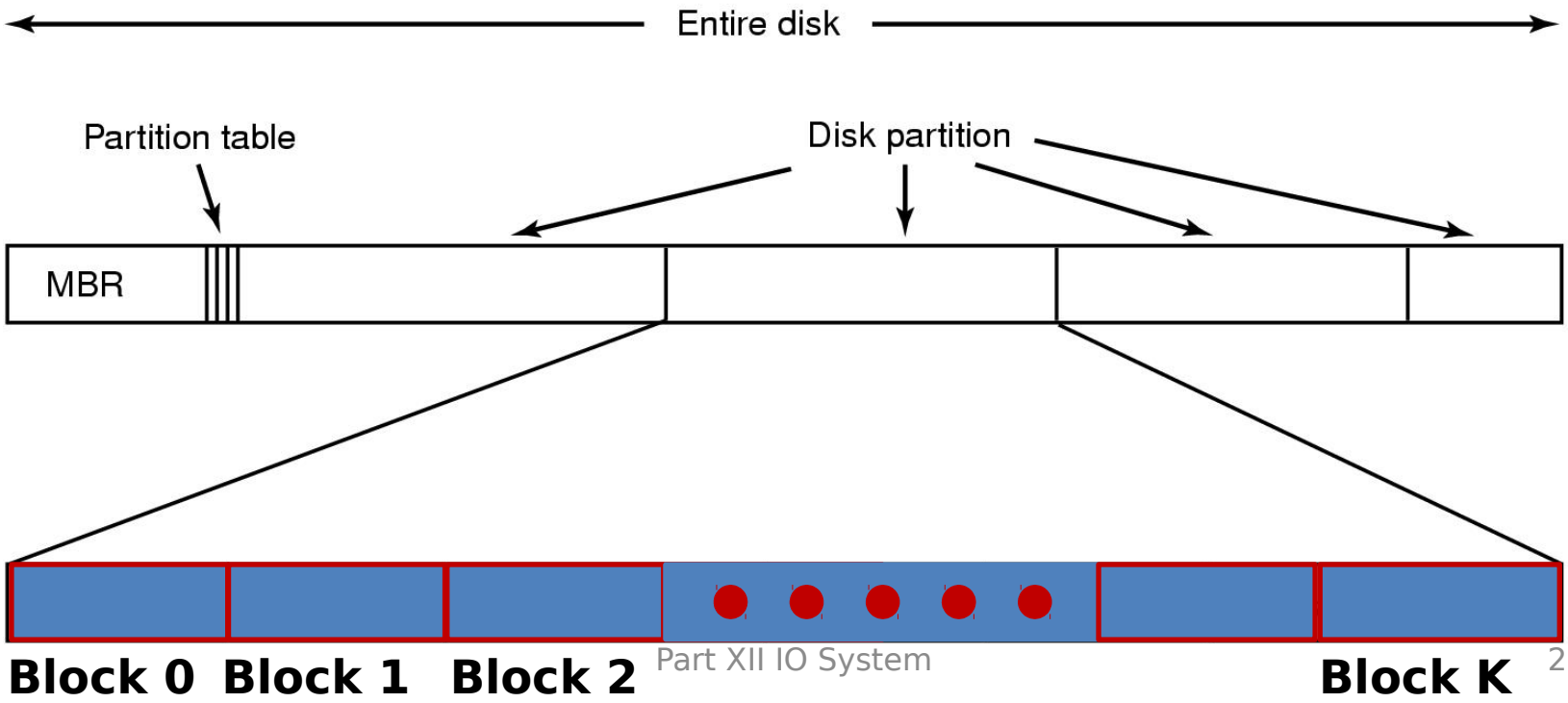
MM – hybrid

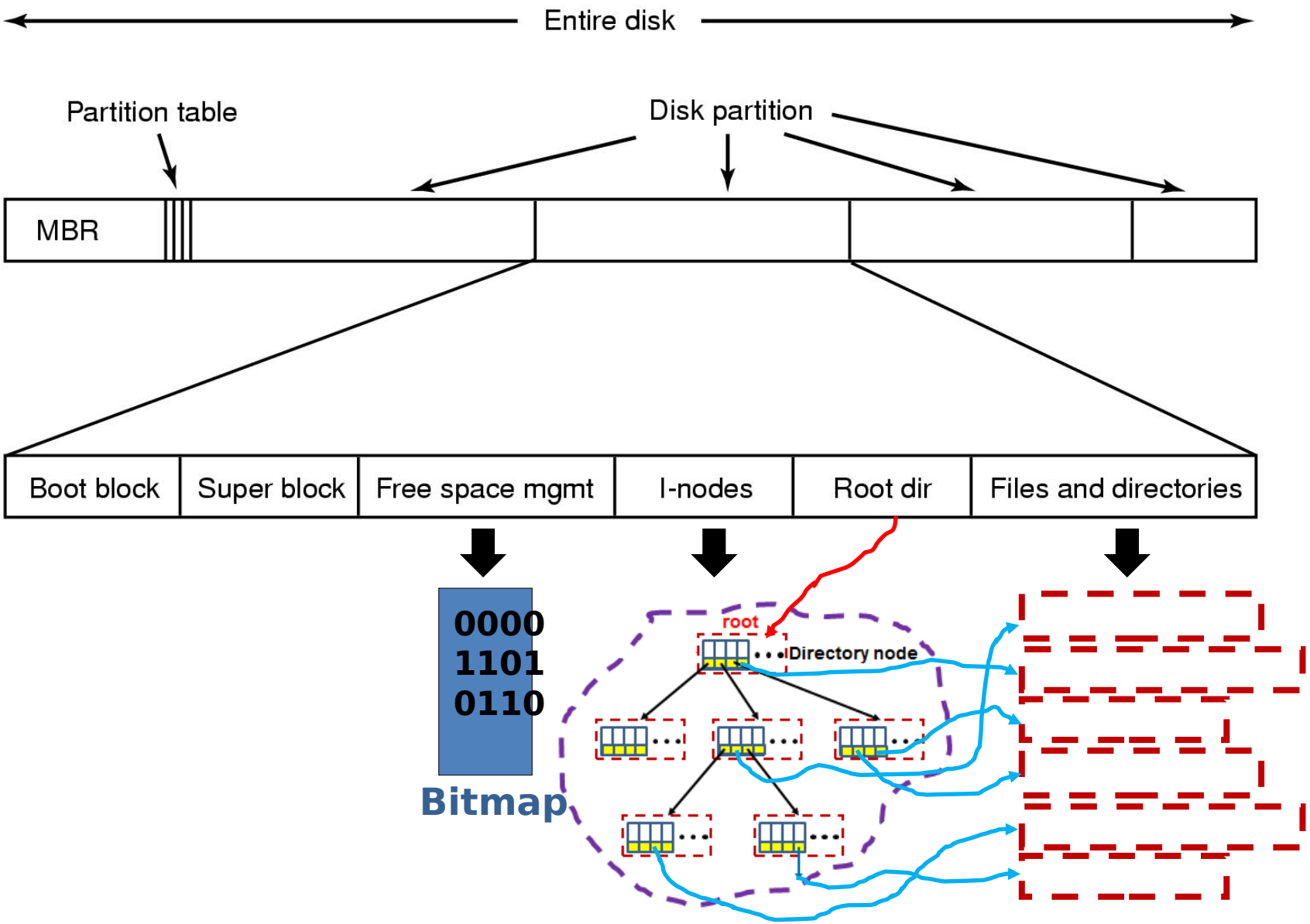
- Segmenting + Paging
  - Only segmenting may not practical: we should also consider the size of the segments
  - Cut MM into frames, program into segments first, and segm ent is further cut into pages (if necessary)
- Some data structures are needed
  - Available frames,
  - Mapping information between your and corresponding frames (segment/page table)
- Placement & Replacement algorithms
  - Based on those data structures



Sector ☾ Partition ☾ Block

- Sectors are further reorganized into blocks to convenience the mapping of your file
  - Usually, the size of a block is 4KB – same as a p age/frame





Disk ☾ Partitions ☾ Blocks ☾ FS

File System Implementation

- Major On-disk Structures (information):
  - Boot control block contains info needed by system to boot O S from that volume
  - Volume control block contains volume details
  - Directory structure organizes the files
  - Per-file File Control Block (FCB) contains many details about t he file