CS343: Operating System

Memory Management

Lect26 : 06th Oct 2023

Dr. A. Sahu

Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

Outline

- Motivations for Memory management
- Memory Management

Motivation for Memory Management

Motivation

- Memory size (RAM Size)
  - Server, Desktop, Mobile, my Old laptop, Smart Watch , smart atm card, RF ID
  - Upto 16TB, 32GB, 8GB, 2GB, 4MB, 1KB, Byte
- Page Mapping (Algorithmic)
  - Example 4GB RAM, page size 4KB
  - Number of Pages: 10^6 pages
    - O(n) or O (nlgn) Algorithm is OK
    - **O(n^2) is dangerous**
  - If Number of pages in Server is 10^9,
    - Linear algorithm have issues

Memory Management

Process Concept

- **Process** – a program in execution; process execution must progress in sequential fashion
- Multiple parts
  - The program code, also called **text section**
  - Current activity including **PC**, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time

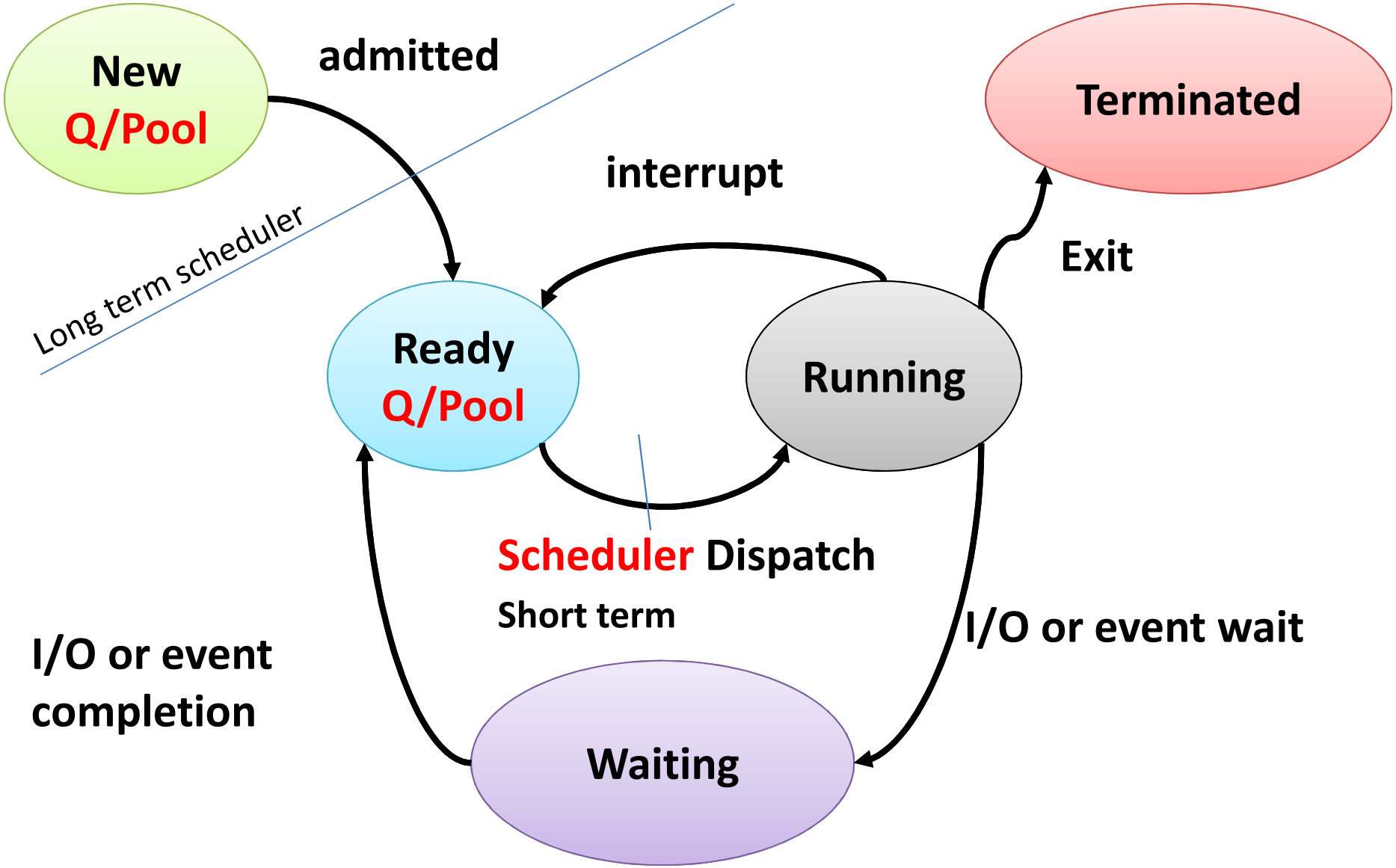
Process to be run..

- Program must be brought (from disk) into memory and placed within a process for it to be run
- CPU can access directly : Main memory and registers
- Memory unit only sees
  - A stream of addresses + read requests, or address + data and write requests

Process to be run..

- Register access in one CPU clock (or less)
- Main memory can take many cycles, causing a **stall**
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

Process State: State Diagram



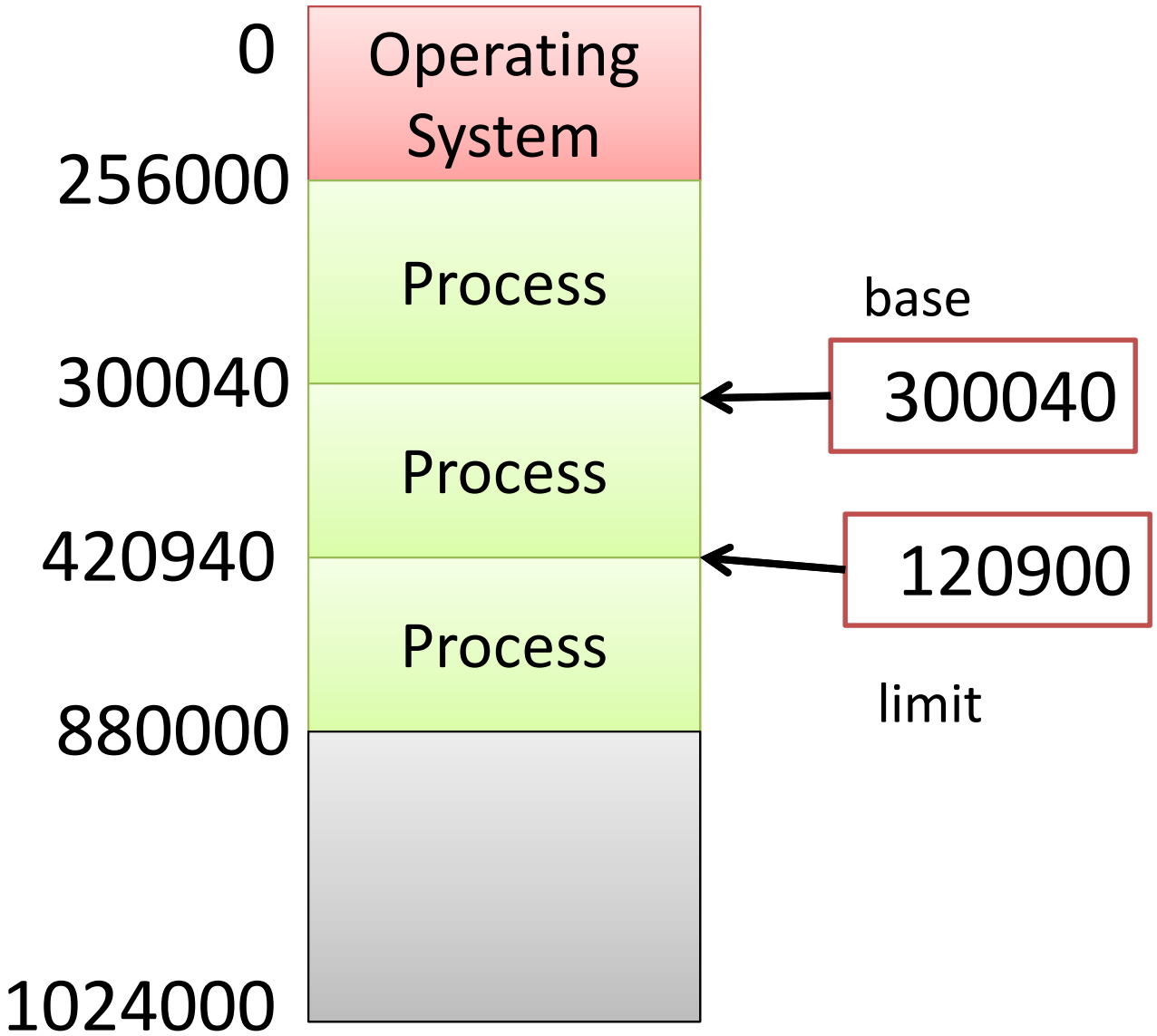
Protection Issue

- Long term scheduler may put many process in to Ready at a time
  - Where to put ?
  - How to access them ?
  - Is there any efficient way to put and access?
  - How ensure safety and protection?

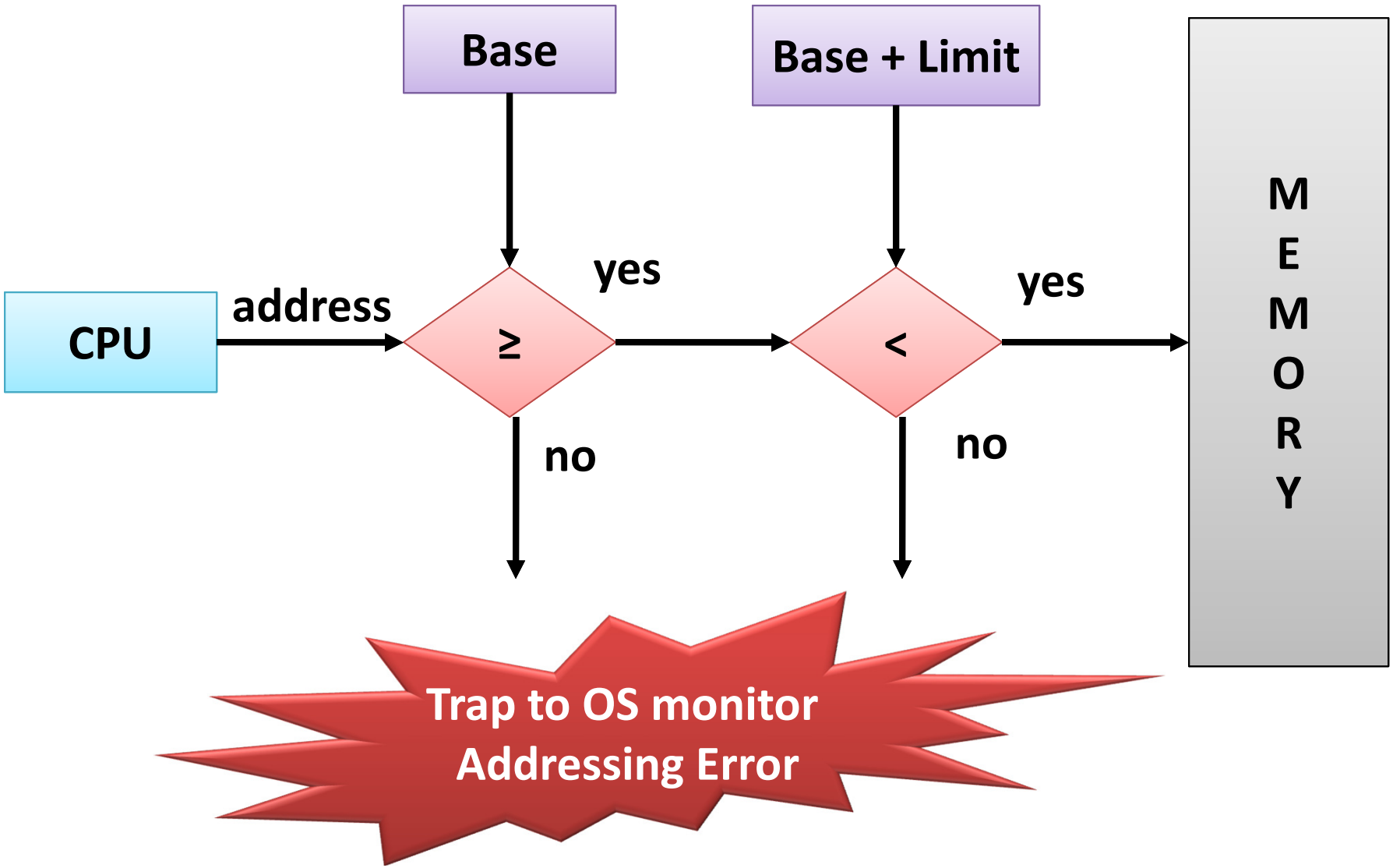
Base and Limit Registers

- A pair of **registers** define the logical address space
  - **base** and **limit**
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user

Base and Limit Registers



Hardware Address Protection



Address Binding

- Programs on disk, ready to be brought into memory to execute form an **input queue**
  - Without support, must be loaded into address 0000
- Inconvenient to have user process physical address always start at 0000
  - How can it not be?

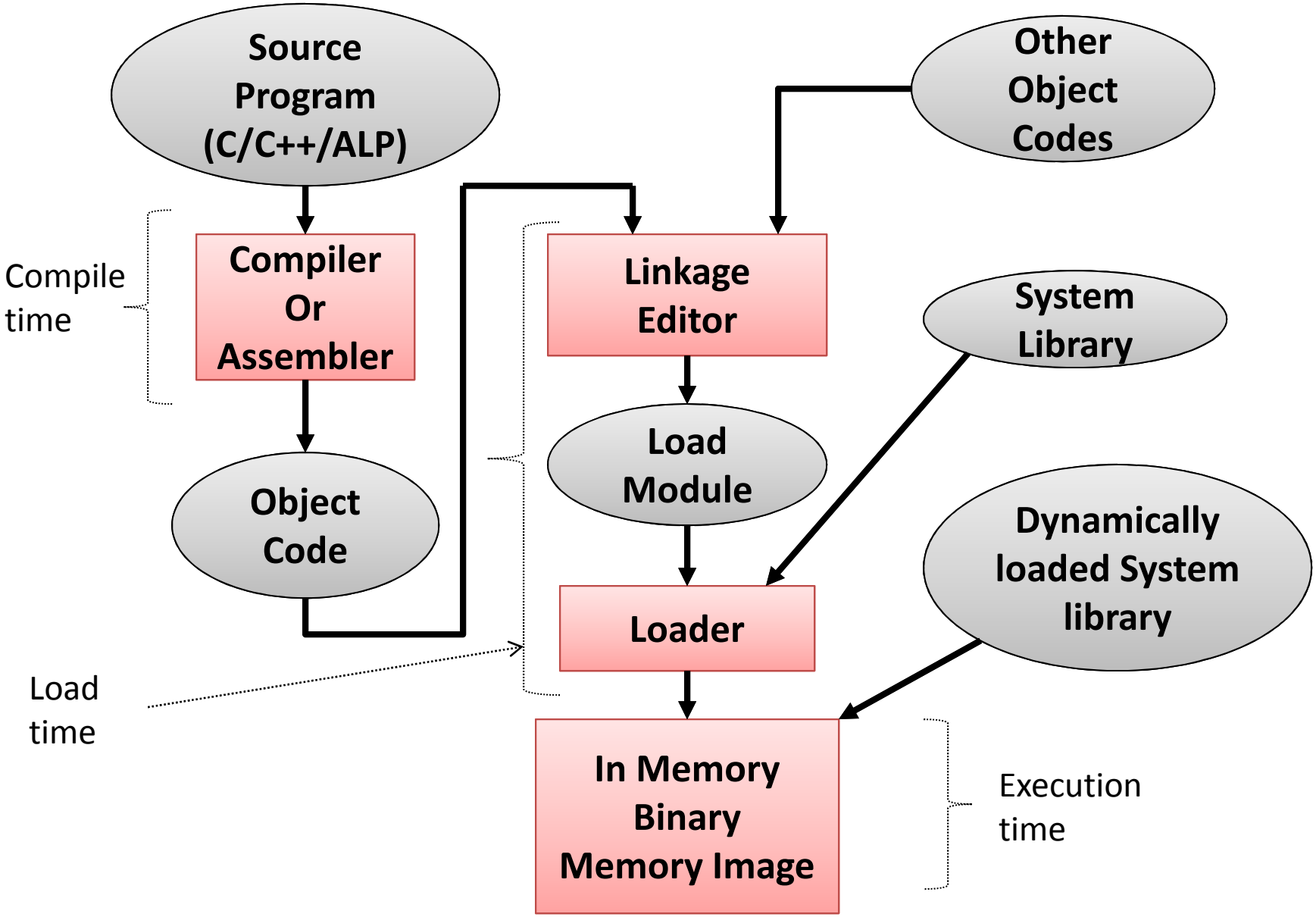
Address Binding

- Further, addresses represented in different ways at different stages of a program's life
  - Source code addresses usually symbolic
  - Compiled code addresses **bind** to relocatable addresses
    - i.e. "14 bytes from beginning of this module"
  - Linker or loader will bind relocatable addresses to absolute addresses
    - i.e. **740**14
  - Each binding maps one address space to another

Binding of Instructions and Data to Memory

- Address binding can happen at three different stages
  - **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
  - **Load time**: Must generate **relocatable code** if memory location is not known at compile time
  - **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
    - Need hardware support for address maps (e.g., base and limit registers)

Multistep Processing of a User Program



Example

- Program with Absolute Address
- Red marked are address
- Data are available at absolute address
  - LD R1 **10**, …
- Call & JMP goto to Absolute Address
  - Call **15** and JNZ **3**

```
0       LD   R1  10
1       LD   R2  11
2       LD   R3  12
3       CALL 15
4       CMP  R1  R3
5       JNZ  3
6       HLT
7
8
9
10   0
11   5
12   100
13
14
15      ADD  R1 R1 R2
16      RET
```

Example

- Program with Relative Address
  - With reloadable register
  - All address need to be add with RR
- Let program go relocated to 5000
  - RelReg1=5000

```
5000            LD      R1  RR+10
5001            LD      R2  RR+11
5002            LD      R3  RR+12
5003            CALL    RR+15
5004            CMP     R1  R3
5005            JNZ     RR+3
5006            HLT
5007
5008
5009
5010    0
5011    5
5012    100
5013
5014
5015            ADD  R1 R1 R2
5016            RET
```

Example

- Program with Relative Address
  - With reloadable register
  - All address need to be add with RR
- Call to Shared Lib
  - Shared Lib Register (Assumption)
  - Add at 2000
  - SR=2000
- Let program go relocated to 5000
  - RelReg1=5000

```
2000            ADD  R1 R1 R2
2001            RET
….
….
…

5000            LD     R1  RR+10
5001            LD     R2  RR+11
5002            LD     R3  RR+12
5003            CALL   SR+15
5004            CMP    R1  R3
5005            JNZ    RR+3
5006            HLT
5007
5008
5009
5010   0
5011   5
5012100
5013
```

Real Life Demo

```
main{
int A[10];
//int B[50];
for(i=0;i<20;i++) A[i]=20;
}
$./a.out
Memory violation
$//uncomment //in B[50]
$./a.out
Run without memory fault
```

A is accessing area of B

```
main{
int i;
printf("%p", &i)
}
$./a.out
0x7ffeff12e658
```

48 bit virtual address
Can address:
$2^8 x 2^{10} x 2^{30} B = 2^8 x 2^{10}$ GB=
2^8 TB= 256TB

Memory for Process

- Single Partition
  - OS + One more process can run at a time
- Same amount to all processes
  - Allow only N processes to be in Memory
  - Size of process is limited by MemSize/N
- Variable amount of memory
  - Allow some process to be fit in the memory at a time
  - Paging +Segmentation: Improve share and reduce fragmentation
- **Virtually : Allow a process to take infinite amount of memory**

Thanks