Operating systems

Lecture 6:

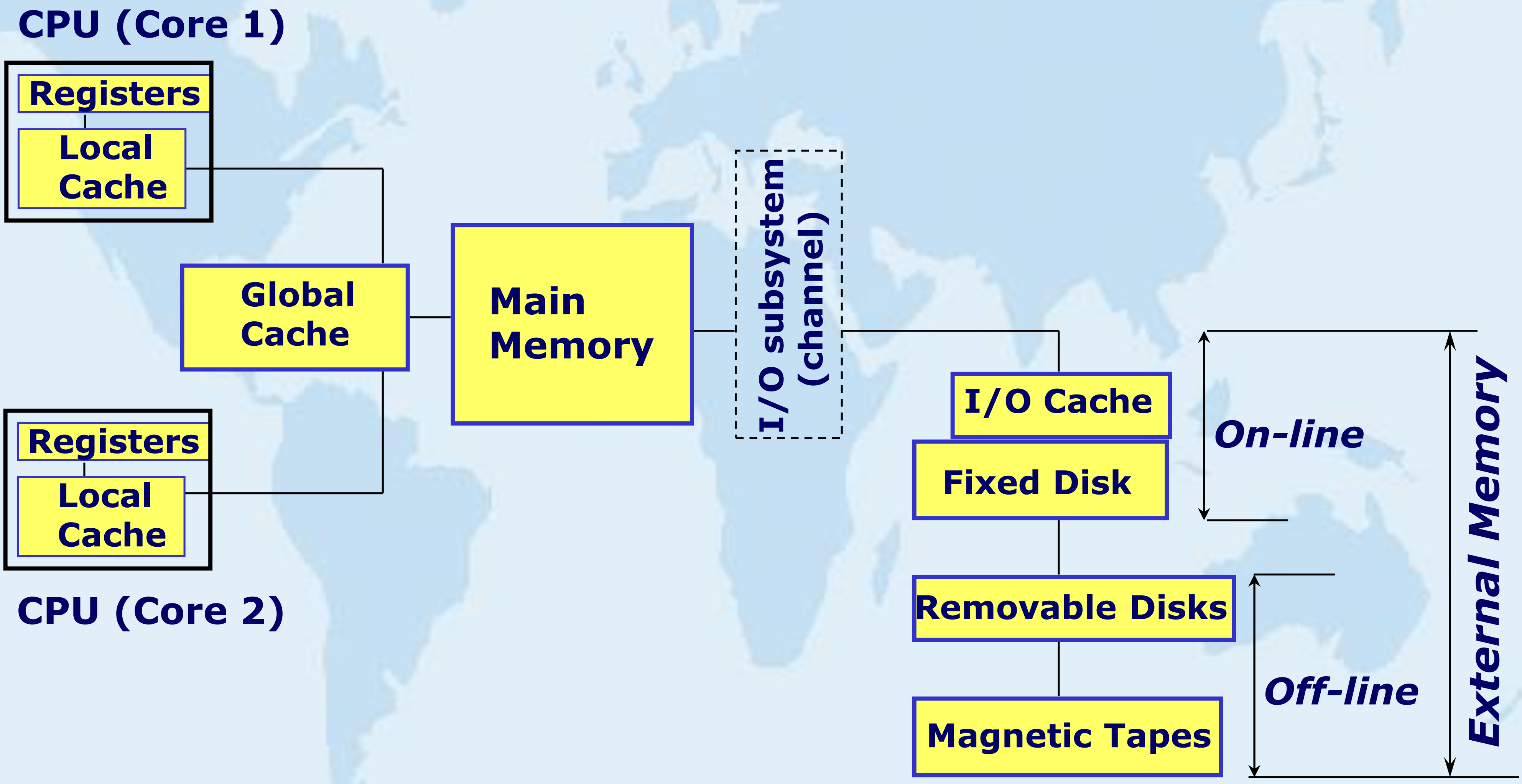
Memory management

Memory Management

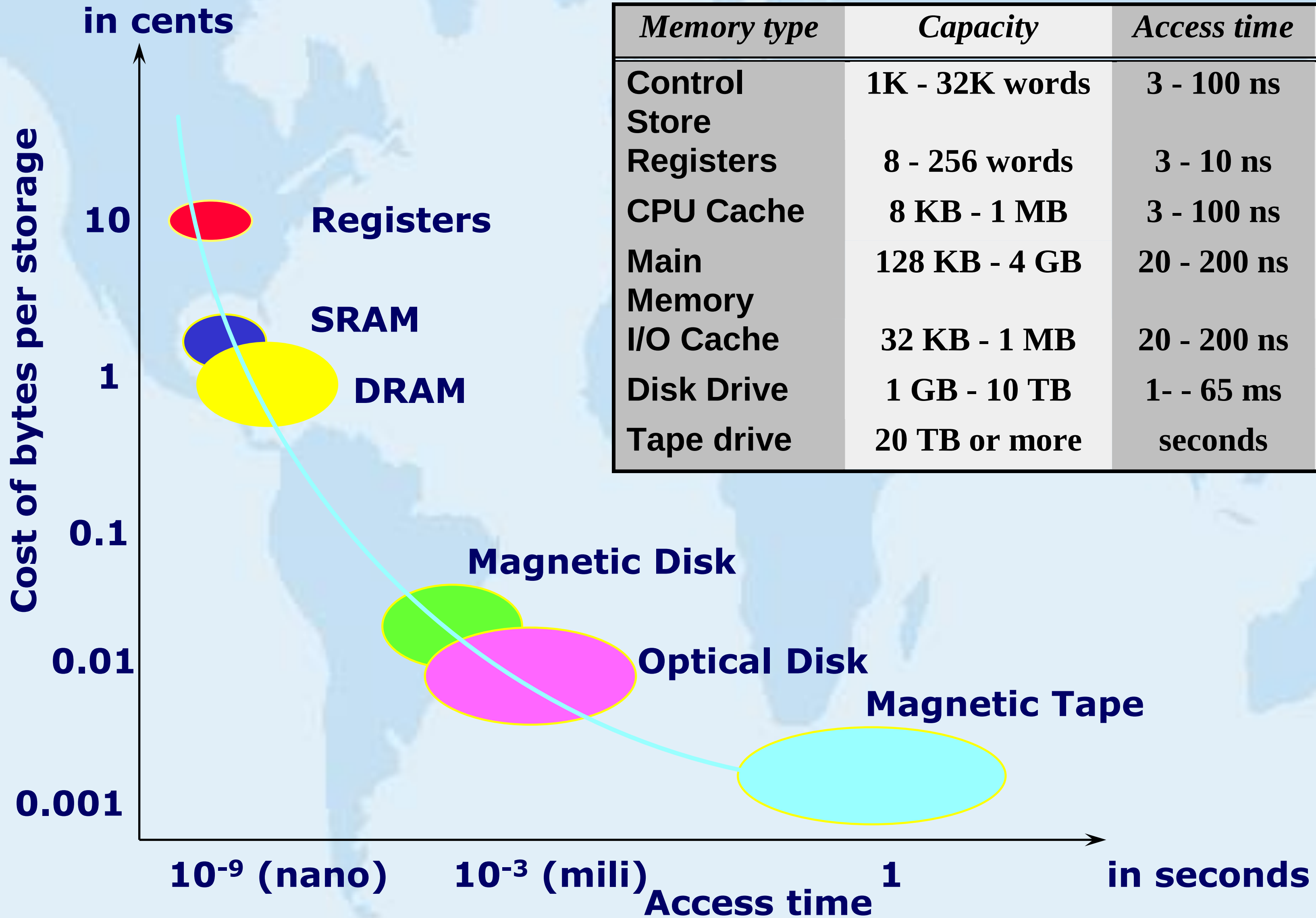
Lecture Outline

- Background
- Logical versus Physical Address Spaces
- Swapping
- Contiguous Allocation

Storage Hierarchy



Access time vs. Cost

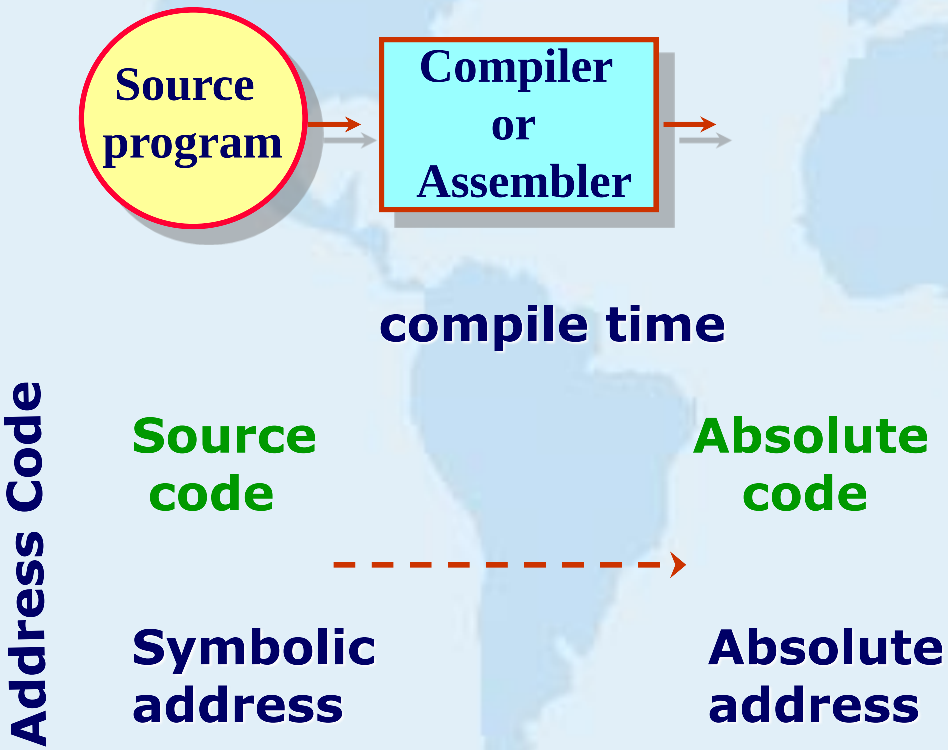


Memory type	*Capacity*	*Access time*	*Technology*
Control Store	1K - 32K words	3 - 100 ns	SRAM, ROM
Registers	8 - 256 words	3 - 10 ns	SRAM
CPU Cache	8 KB - 1 MB	3 - 100 ns	SRAM, DRAM
Main Memory	128 KB - 4 GB	20 - 200 ns	DRAM
I/O Cache	32 KB - 1 MB	20 - 200 ns	DRAM
Disk Drive	1 GB - 10 TB	1- - 65 ms	Magnetic Disks
Tape drive	20 TB or more	seconds	Magnetic Tape

Background

- Program must be brought into memory and placed within a **process** to be executed.
- **Input queue** - collection of processes on the disk that are waiting to be brought into memory for execution.
- User processes can reside in **any part** of the physical memory.
- Program addresses are mapped to a re-locatable or physical addresses – **address binding.**

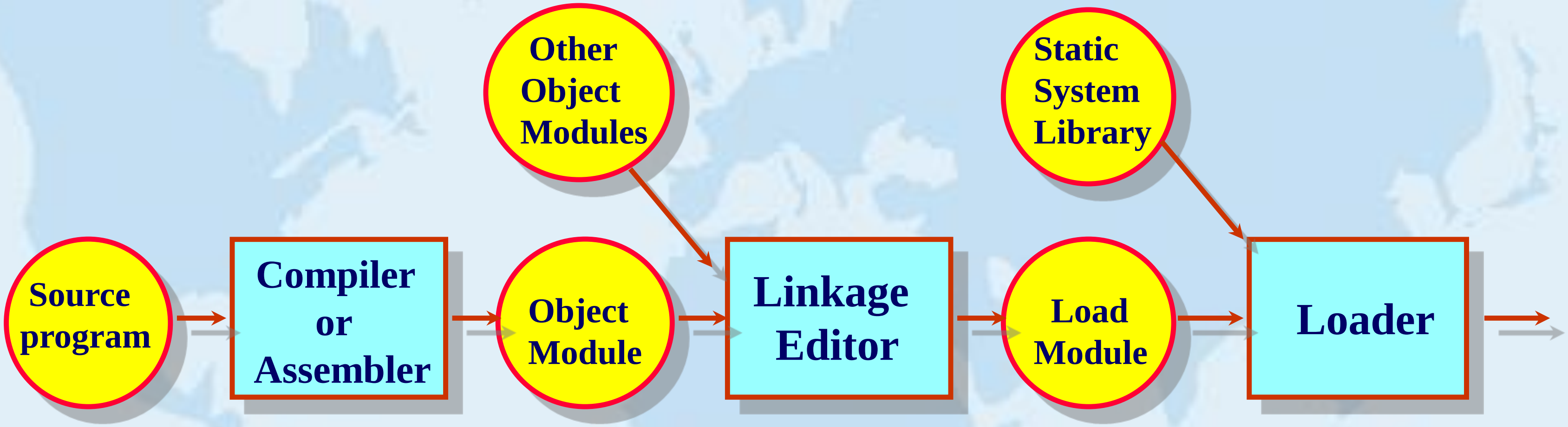
Address Binding: Compile Time



If it is known at compile time where the process will reside in memory, then **absolute** code can be generated.

- Must recompile code if starting location changes.

Address Binding: Load Time



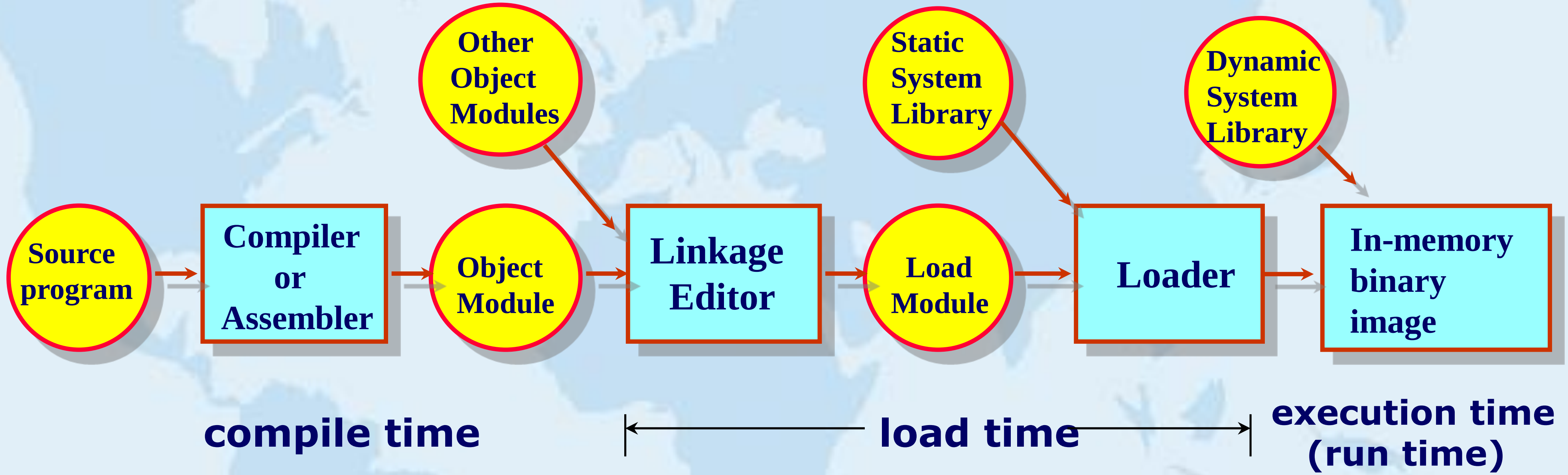
compile time | load time

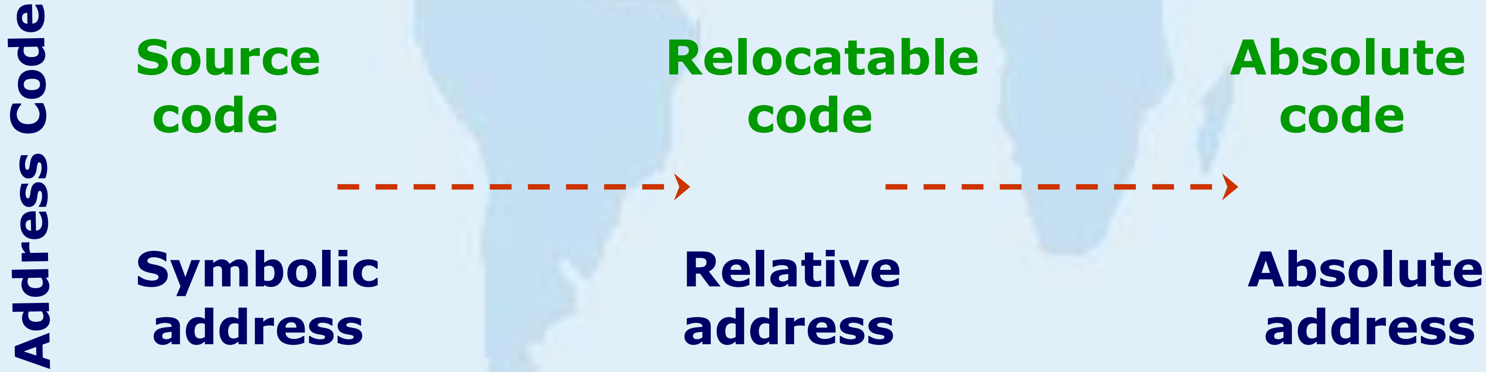
Address Code

Source code	Relocatable code	Absolute code
Symbolic address	Relative address	Absolute address

Compiler must generate **relocatable** code if memory location is not known at compile time.

Address Binding: Execution Time





Binding **delayed** until run time if the process can be moved during its execution from one memory segment to another.

Logical Address Spaces

- It is much **simpler** and **efficient** to assign addresses to variables at **load time**.
- However, this presents a **problem** for multi-programmed systems.
- **How** can linker know in advance which processes will be running at the same time in order to guarantee that they don't interfere with each other?
- Also, in the presence of OS revisions and reconfigurations, **how** can linker always know where/how big the OS's image is in the memory?

Logical Address Spaces

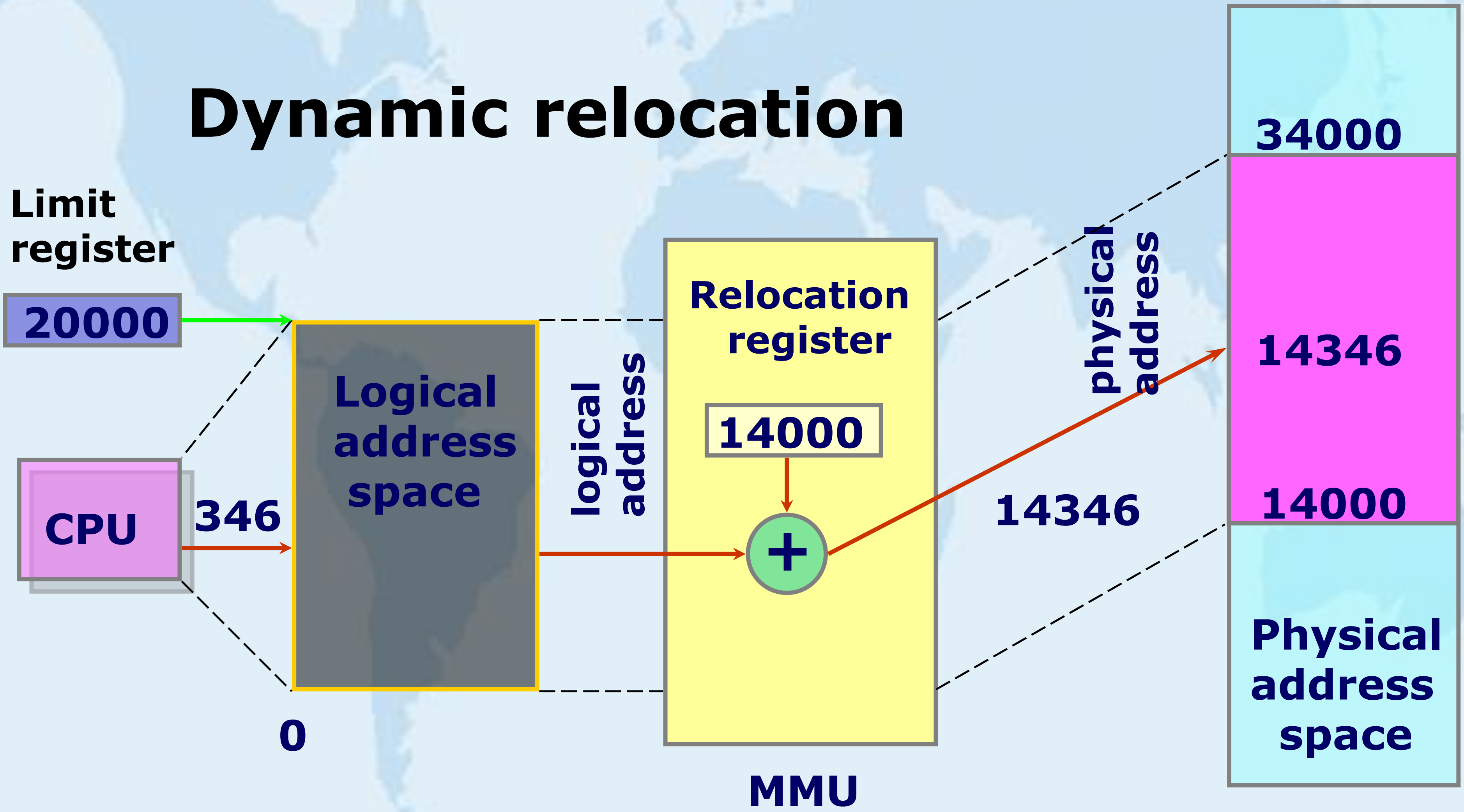
- The answer is to assign each process its own set of **logical** “absolute” addresses and delay the **binding** of these addresses to physical memory addresses until runtime by hardware assistance.
- At runtime, **Memory-Management Unit (MMU)** – an **address translation hardware** converts logical addresses to physical addresses on a per-process basis.

Logical vs. Physical Address Spaces

- The concept of a **logical address space** that is bound to a separate **physical address space** is central to proper memory management.
  - **Logical address** - generated by the CPU, also referred to as **virtual address**.
  - **Physical address** - address seen by the MMU.
- Logical and physical addresses are the **same** in compile-time and load-time address-binding schemes
- Logical and physical addresses **differ** in execution-time address-binding scheme.

Logical vs. Physical Address Spaces

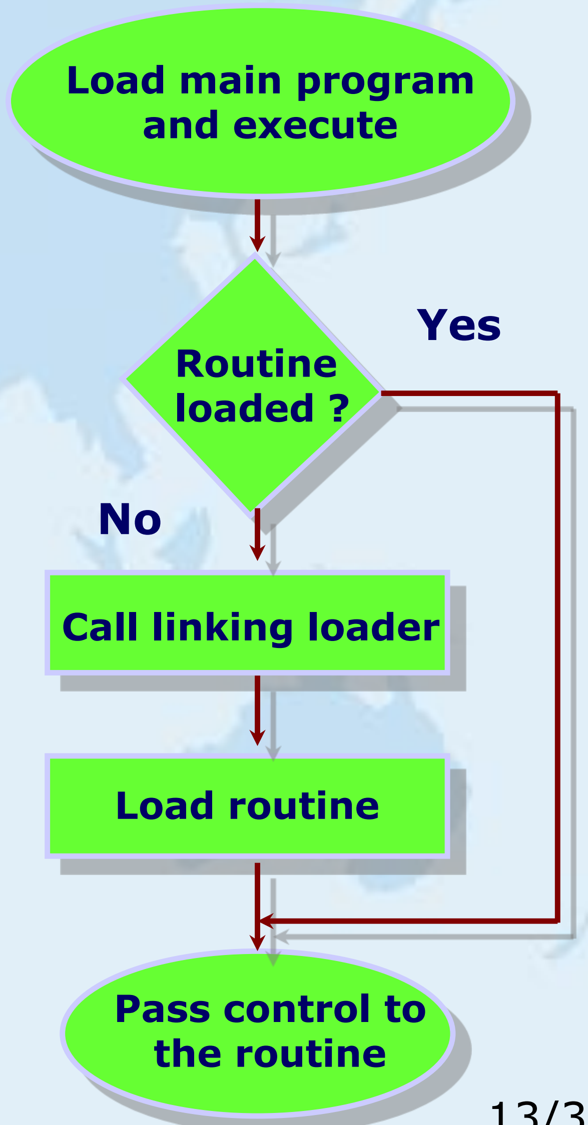
Dynamic relocation



Dynamic Loading

Dynamic Loading - routine is not loaded **until** it is called.

- Better memory-space utilization: unused routine is **never** loaded.
- Useful when large amounts of code are needed to handle infrequently occurring cases.
- No special support from the operating system is required; implemented through program design.



Dynamic Linking

Libraries – **Static** or **Dynamic.**

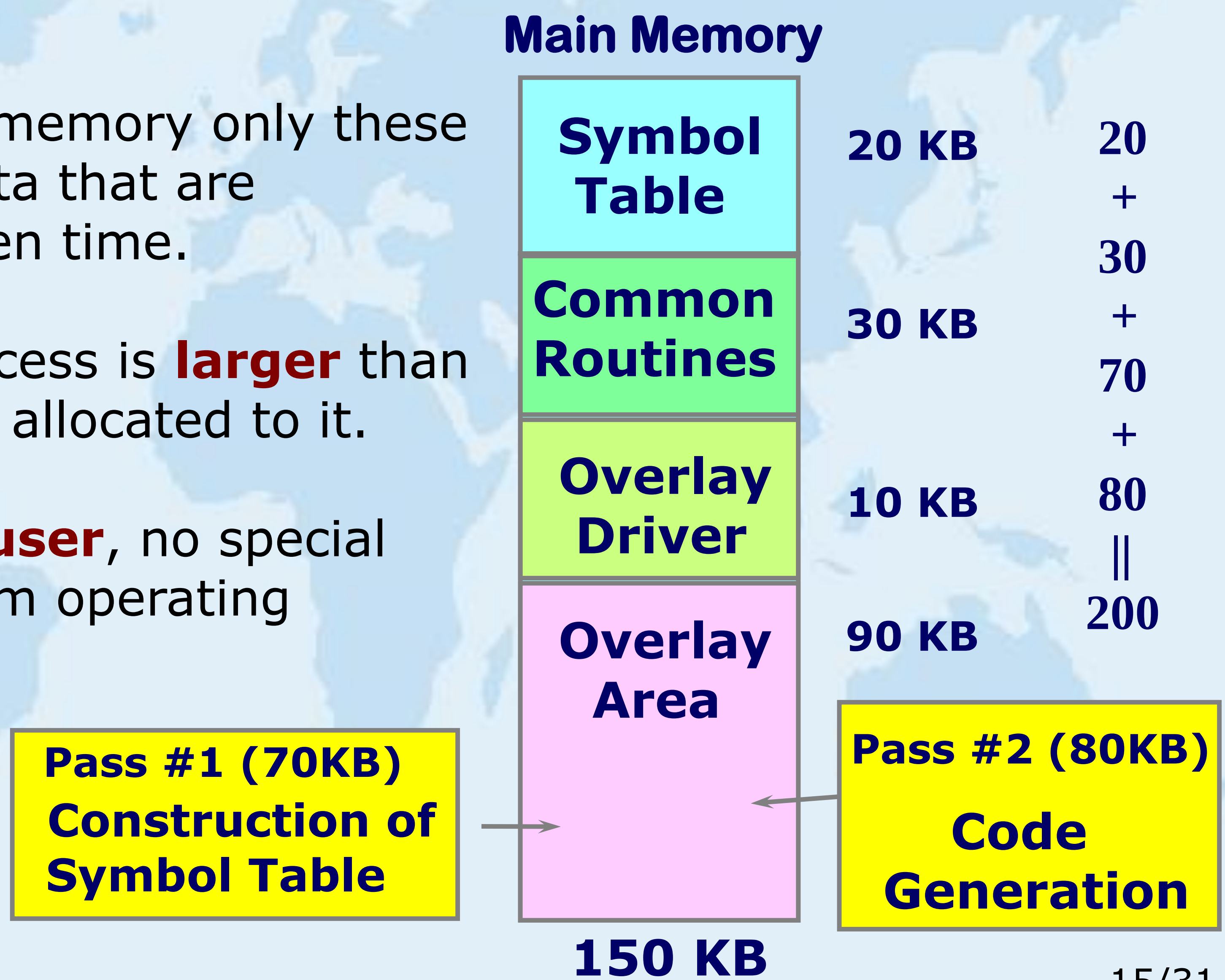
- Some systems support **only** static libraries.
- **Dynamic linking** is similar to dynamic loading – linking is postponed until execution time (rather than loading).
- Dynamically linked libraries: DLL (Windows), shared libraries (Unix, Linux)
- Advantages:
  - **Easy** to update libraries.
  - **Several** versions can be loaded in the memory.

Overlays

Overlays - keep in memory only these instructions and data that are **needed** at any given time.

- Needed when process is **larger** than amount of memory allocated to it.

- Implemented by **user**, no special support needed from operating system;

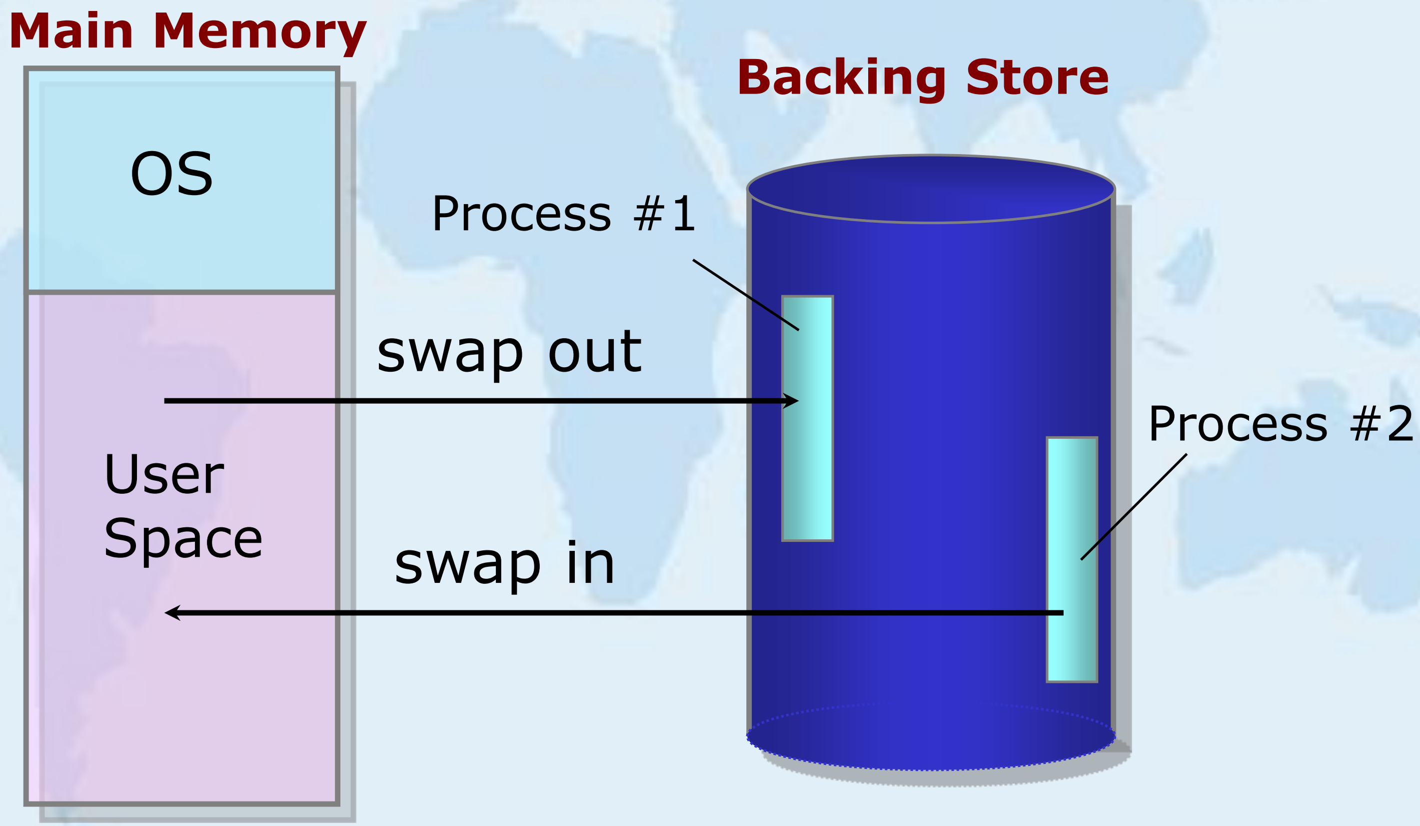


Swapping

- A process can be **swapped** temporarily out of memory to a **backing store**, and then brought back into memory for continued execution.
- **Backing store** - fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.
- **Roll out , roll in** - swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.

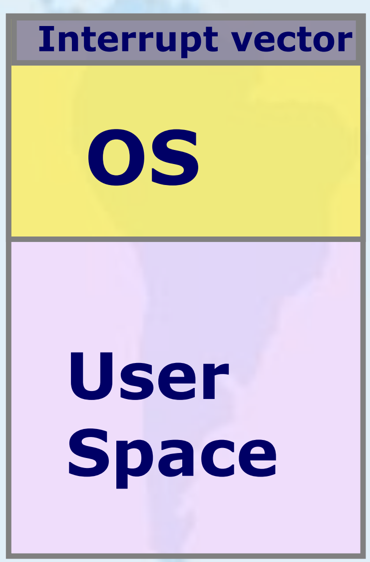
Swapping

Purpose of swapping: Release main memory for other processes while waiting for a significantly long period of time.



Contiguous Allocation

- Main Memory is usually split into **two** partitions:
  - Resident **operating system** is held in low memory with interrupt vector.
  - **User processes** are then held in high memory.

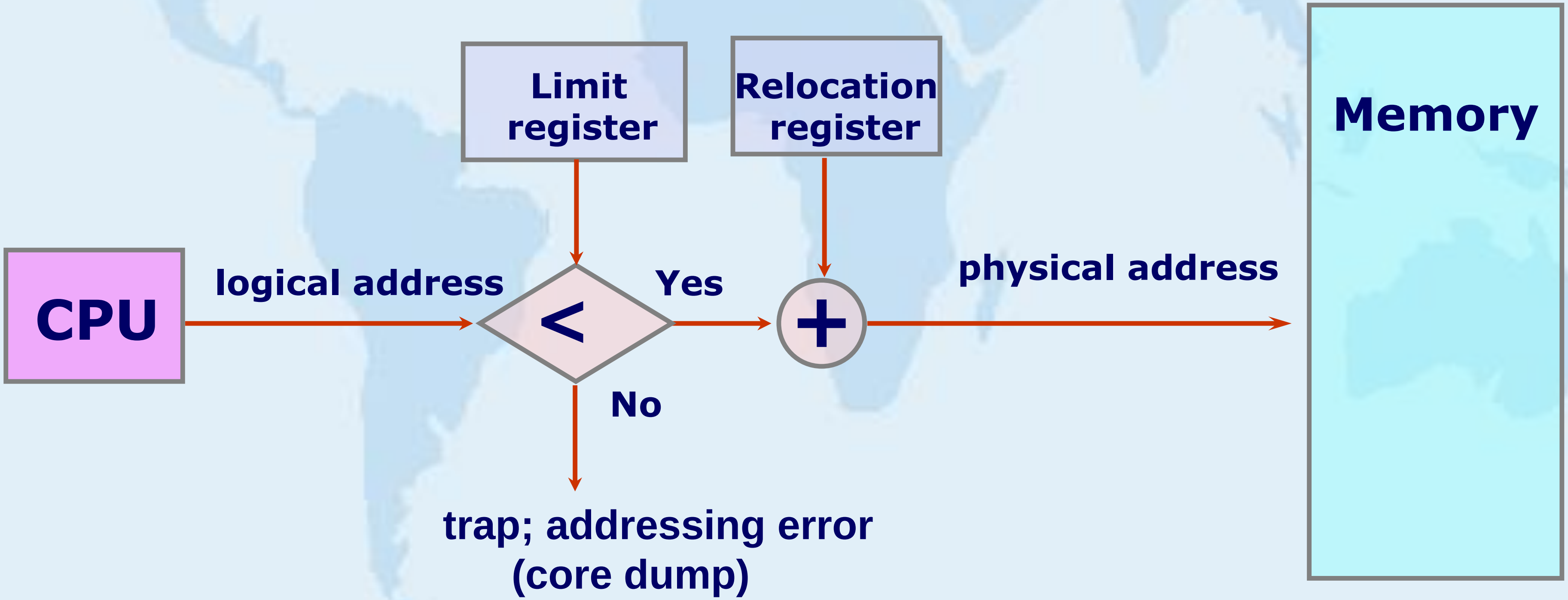


Memory Partition

- **Single-Partition Allocation**
- **Multiple-Partition Allocation**

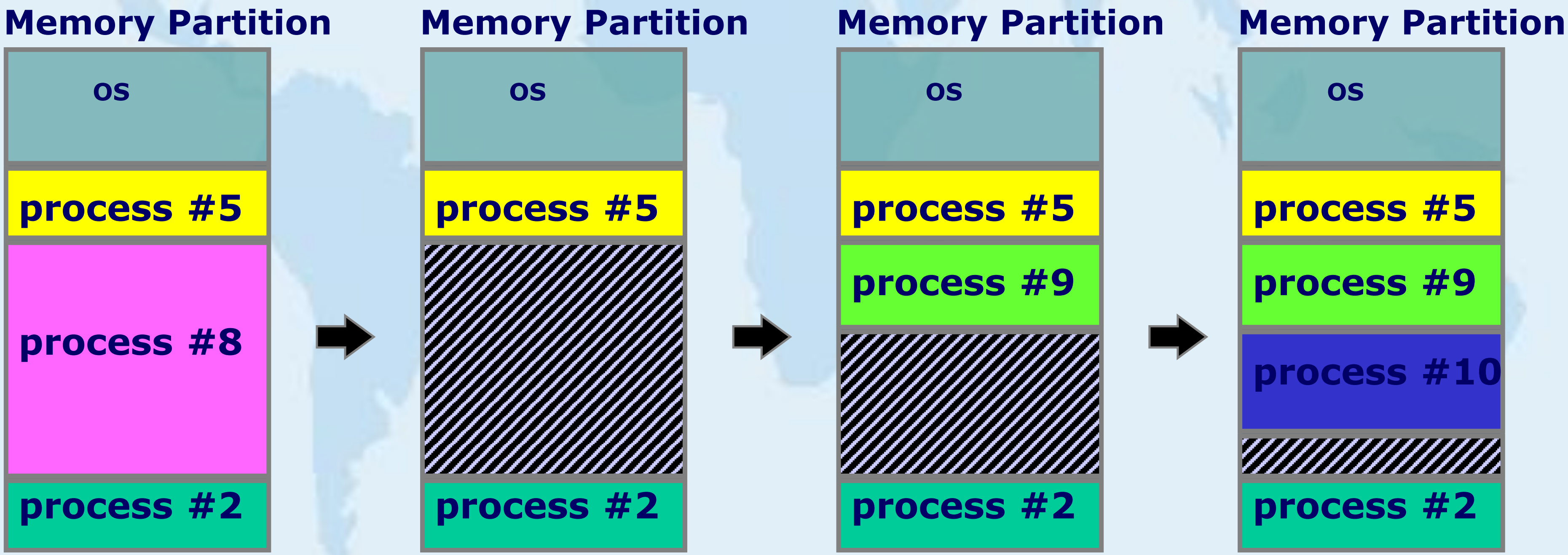
Contiguous Allocation

- **Relocation-register** scheme is used to protect user processes from each other, and from changing OS code and data.
- Relocation register contains value of smallest physical address; limit register contains **range** of logical addresses - each logical address must be **less** then the limit register.



Contiguous Allocation

- Multiple partition - required for multiprogramming.
- **Hole -** block of available memory; holes of various size are scattered throughout memory.
- When a process arrives, it is allocated memory from a hole large enough to accommodate it.



Contiguous Allocation

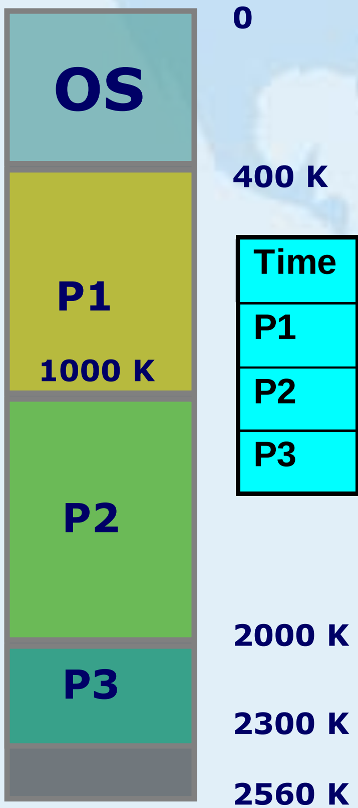
Example of memory allocation

Job queue:

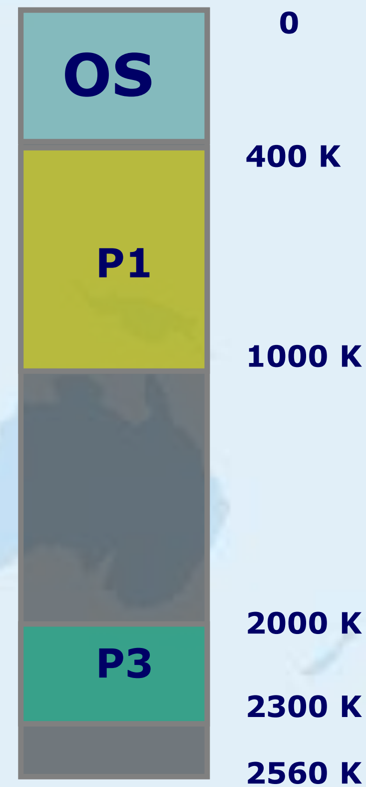
Process	Memory	Time
P5	500 K	15
P4	700 K	8
P3	300 K	20
P2	1000 K	5
P1	600 K	10

Scheduling parameters:

Job: FCFS
CPU: Round-robin
Quantum time = 1
Memory: 2560 KB



Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14
P1	1	1	1	2	2	2	3	3	3	4	4	4	5	5
P2	0	1	1	1	2	2	2	3	3	3	4	4	4	5
P3	0	0	1	1	1	2	2	2	3	3	3	4	4	4



Contiguous Allocation

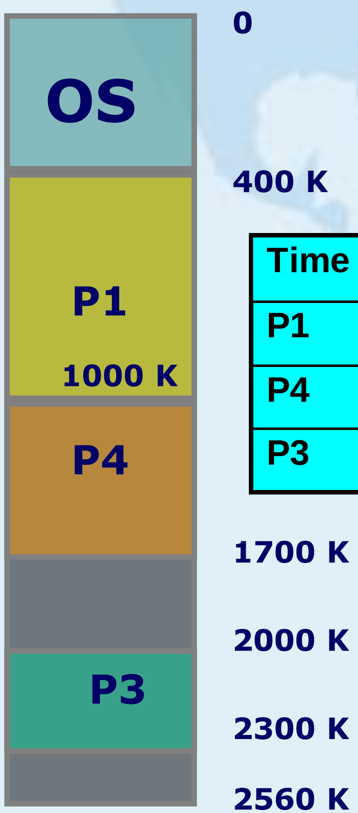
Example of memory allocation

Job queue:

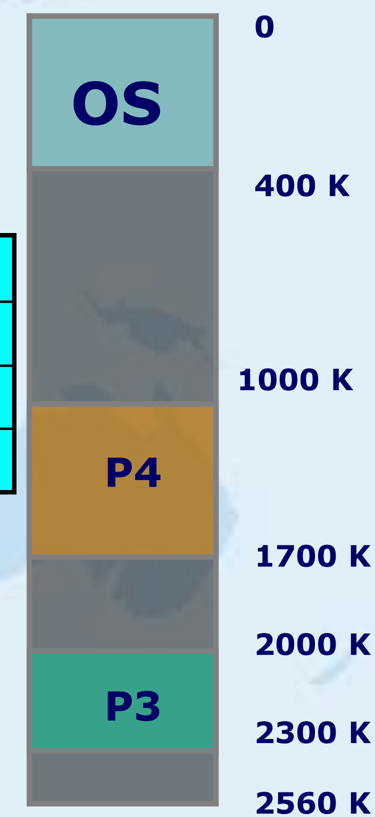
Process	Memory	Time
P5	500 K	15
P4	700 K	8
P3	300 K	20
P1	600 K	10

Scheduling parameters:

Job: FCFS
CPU: Round-robin
Quantum time = 1
Memory: 2560 KB



Time	15	16	17	18	19	20	21	22	23	24	25	26	27	28
P1	5	6	6	6	7	7	7	8	8	8	9	9	9	10
P4	0	0	1	1	1	2	2	2	3	3	3	4	4	4
P3	5	5	5	6	6	6	7	7	7	8	8	8	9	9



Contiguous Allocation

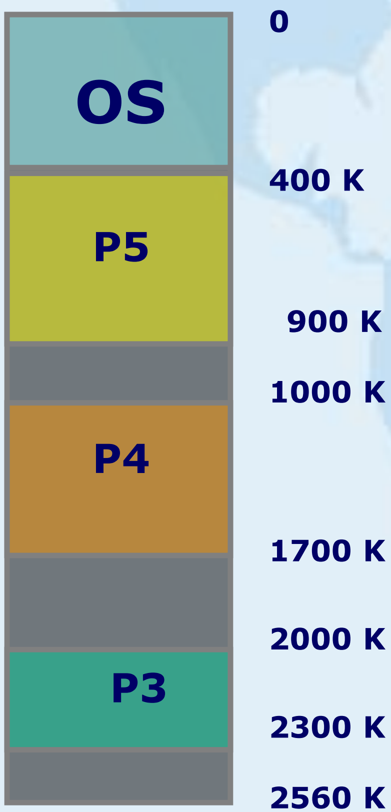
Example of memory allocation

Job queue:

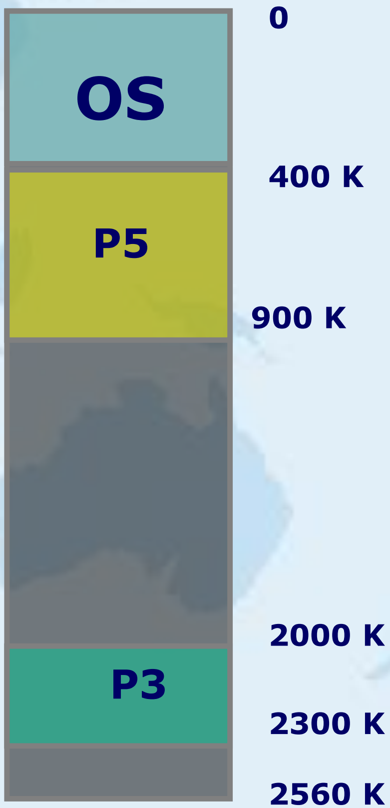
Process	Memory	Time
P5	500 K	15
P4	700 K	8
P3	300 K	20

Scheduling parameters:

Job: FCFS
CPU: Round-robin
Quantum time = 1
Memory: 2560 KB



Time	29	30	31	32	33	34	35	36	37	38
P5	0	0	1	1	1	2	2	2	3	3
P4	5	5	5	6	6	6	7	7	7	8
P3	9	10	10	10	11	11	11	12	12	12



Contiguous Allocation

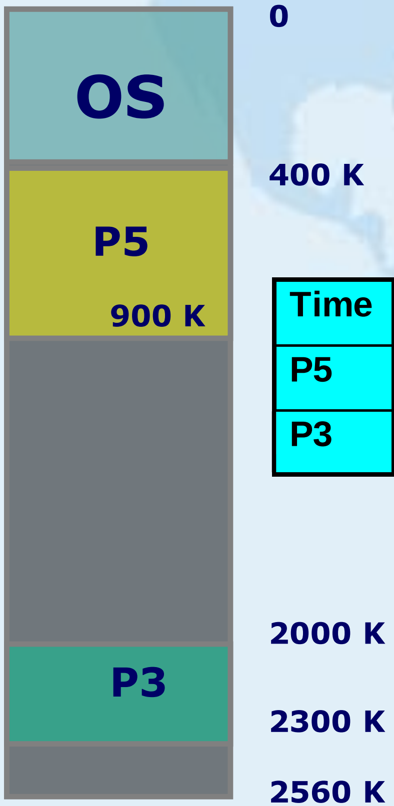
Example of memory allocation

Job queue:

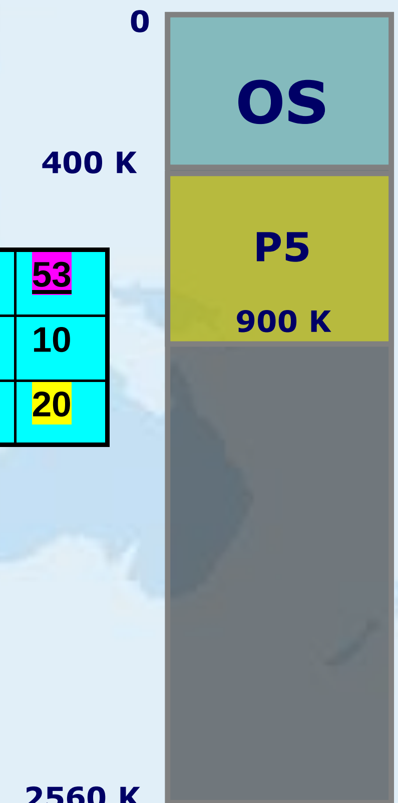
Process	Memory	Time
P5	500 K	15
P3	300 K	20

Scheduling parameters:

Job: FCFS
CPU: Round-robin
Quantum time = 1
Memory: 2560 KB

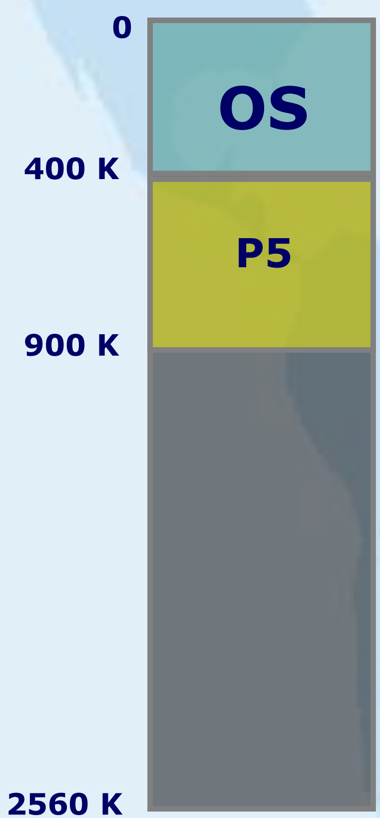


Time	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53
P5	3	4	4	5	5	6	6	7	7	8	8	9	9	10	10
P3	13	13	14	14	15	15	16	16	17	17	18	18	19	19	20



Contiguous Allocation

Example of memory allocation



Job queue:

Process	Memory	Time
P5	500 K	15

Time	54	55	56	57	58
P5	11	12	13	14	15

Scheduling parameters:

Job: FCFS
CPU: Round-robin
Quantum time = 1
Memory: 2560 KB

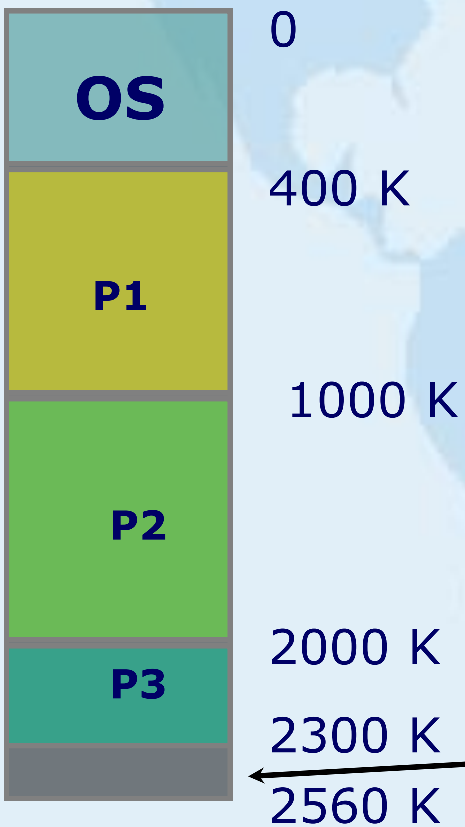
Dynamic Storage Allocation

- **Dynamic storage-allocation** problem - how to satisfy a request of size **n** from a list of free holes.
  - **First-fit**: Allocate the **first** hole that is big enough.
  - **Best-fit**: Allocate the **smallest** hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
  - **Worst-fit**: Allocate the **largest** hole; must also search list. Produces the largest leftover hole.

Strategy	Search Time	Memory Utilization
First-fit	Fast	Good
Best-fit	Slow	Good
Worst-fit	Slow	Bad

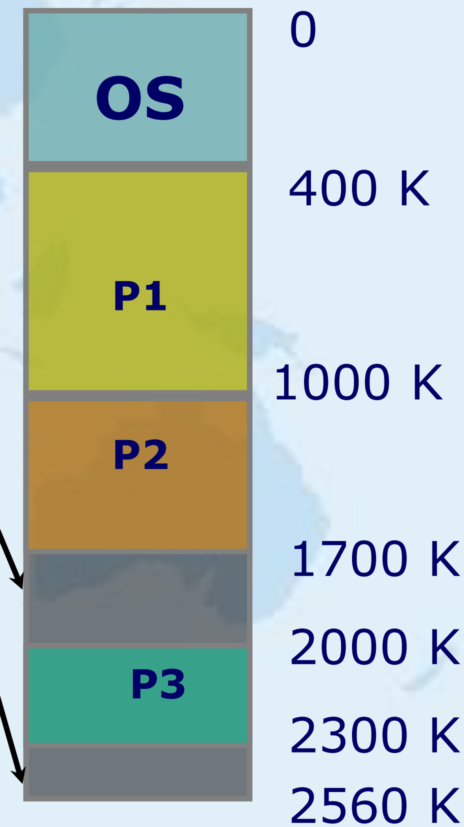
Memory Fragmentation

External fragmentation – when total memory space exists to satisfy a request, but it is not contiguous;



Total fragmentation of 560 K (=300K+260K), enough for P4 (500K), but not contiguous

Total fragmentation of 260 K, not enough for P4 (500K)



Memory Fragmentation

- Selection of first-fit versus best-fit can **affect** the amount of fragmentation.

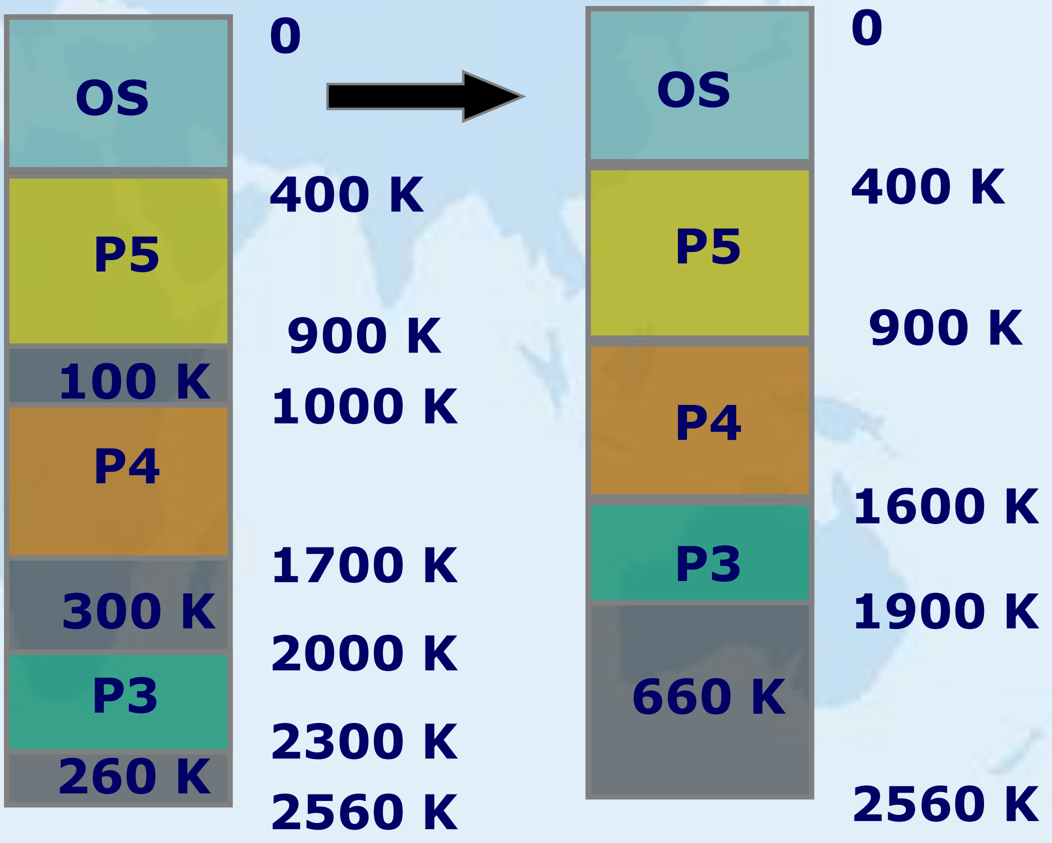
- Statistical analysis of first-fit reveals that, even with some optimization, given *N* allocated blocks, another *0.5N* blocks will be lost due to fragmentation. This property is known as the **50-percent rule.**

- Solutions:
  - (a) **Compaction**
  - (b) **Paging**

Memory Fragmentation

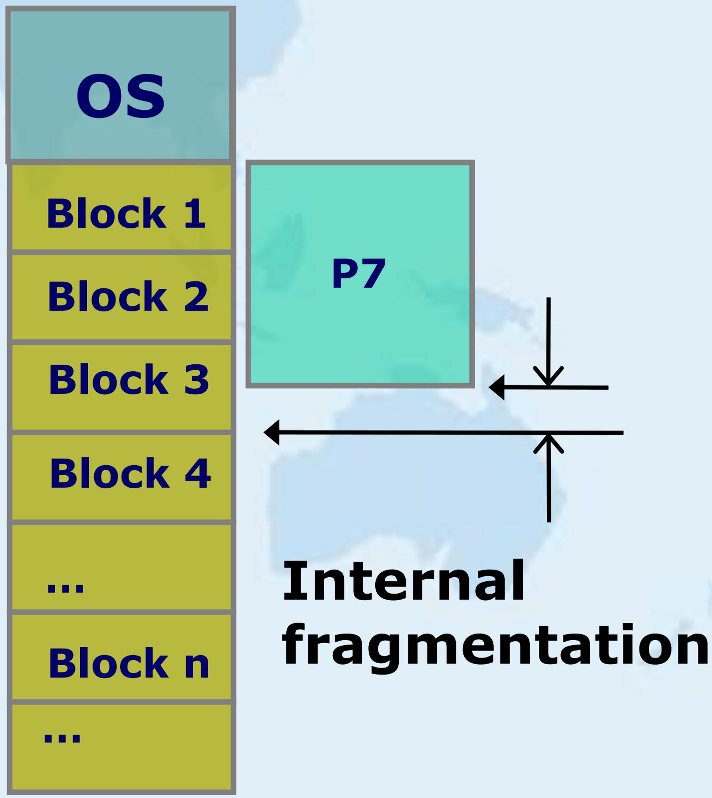
- Reduce external fragmentation by **compaction.**
  - Shuffle memory contents to place all free memory together in one large block.
  - Compaction is possible only if relocation is dynamic, and is done at execution time.

Compaction



Memory Fragmentation

- Memory is broken into fixed-size blocks – **pages**.
- Allocate memory in **unit** of block sizes.
- Allocated memory may be slightly **larger** than requested memory.
- **Internal fragmentation** - difference between requested and allocated memory.



That is all for today!