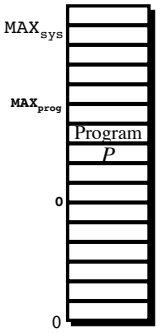
Memory Management Basics

Basic Memory Management Concepts
Address spaces

- *Physical address space* — The address space supported by the hardware
  - Starting at address 0, going to address MAX_{sys}
- *Logical/virtual address space* — A process's view of its own memory
  - Starting at address 0, going to address MAX_{prog}

But where do addresses come from?

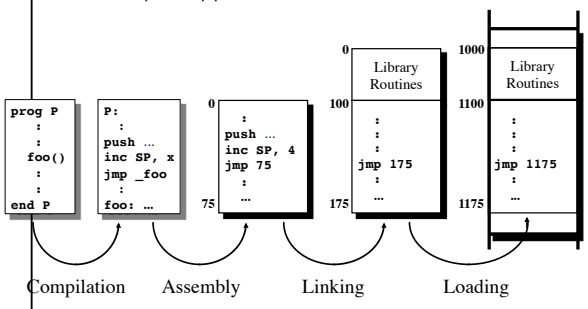
```
MOV r0, @0xfffa620e
```



- Which is bigger, physical or virtual address space?
  - A. Physical address space
  - B. Virtual address space
  - C. It depends on the system.

Basic Concepts
Address generation

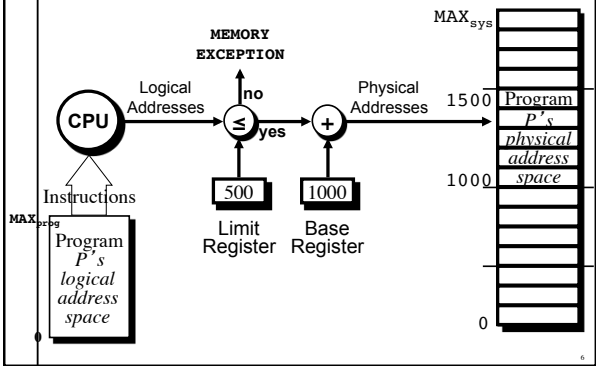
- The compilation pipeline



Program Relocation

- Program issues virtual addresses
- Machine has physical addresses.
- If virtual == physical, then how can we have multiple programs resident concurrently?
- Instead, relocate virtual addresses to physical at run time.
  - While we are relocating, also bounds check addresses for safety.
- I can relocate that program (safely) in two registers…

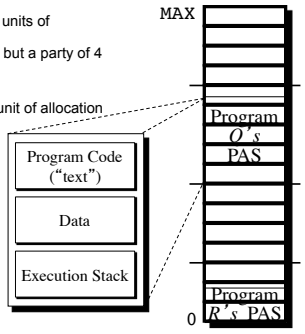
Basic Concepts (Cont'd.)
Address Translation



- With base and bounds registers, the OS needs a hole in physical memory at least as big as the process.
  - A. True
  - B. False

Evaluating Dynamic Allocation Techniques
The fragmentation problem

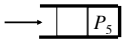
- External fragmentation
  - Unused memory between units of allocation
  - E.g, two fixed tables for 2, but a party of 4
- Internal fragmentation
  - Unused memory within a unit of allocation
  - E.g., a party of 3 at a table for 4



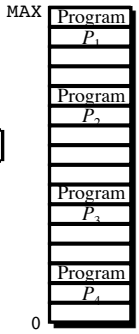
Simple Memory Management Schemes
Dynamic allocation of partitions

- Simple approach:
  - Allocate a partition when a process is admitted into the system
  - Allocate a contiguous memory partition to the process

OS keeps track of...
Full-blocks
Empty-blocks ("holes")



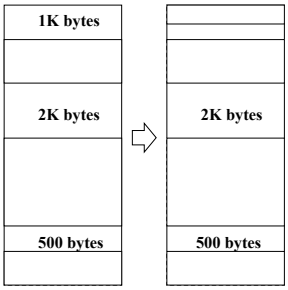
Allocation strategies
First-fit
Best-fit
Worst-fit



First Fit Allocation

To allocate *n* bytes, use the *first* available free block such that the block size is larger than *n*.

To allocate 400 bytes, we use the 1st free block available



Rationale & Implementation

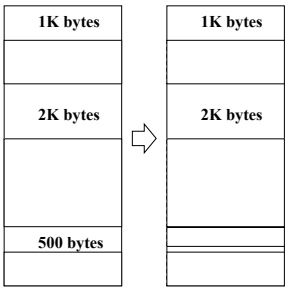
- Simplicity of implementation
- Requires:
  - Free block list sorted by address
  - Allocation requires a search for a suitable partition
  - De-allocation requires a check to see if the freed partition could be merged with adjacent free partitions (if any)

Advantages	Disadvantages
Simple	Slow allocation
Tends to produce larger free blocks toward the end of the address space	External fragmentation

Best Fit Allocation

To allocate *n* bytes, use the *smallest* available free block such that the block size is larger than *n*.

To allocate 400 bytes, we use the 3rd free block available (smallest)



Rationale & Implementation

- To avoid fragmenting big free blocks
- To minimize the size of external fragments produced
- Requires:
  - Free block list sorted by size
  - Allocation requires search for a suitable partition
  - De-allocation requires search + merge with adjacent free partitions, if any

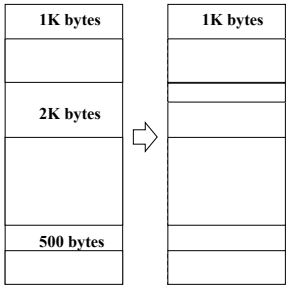
Advantages	Disadvantages
Works well when most allocations are of small size	External fragmentation
Relatively simple	Slow de-allocation
	Tends to produce many useless tiny fragments (not really great)

- Doug Lea's malloc "In most ways this malloc is a best-fit allocator"

Worst Fit Allocation

To allocate *n* bytes, use the *largest* available free block such that the block size is larger than *n*.

To allocate 400 bytes, we use the 2nd free block available (largest)



Rationale & Implementation

- To avoid having too many tiny fragments
- Requires:
  - Free block list sorted by size
  - Allocation is fast (get the largest partition)
  - De-allocation requires merge with adjacent free partitions, if any, and then adjusting the free block list

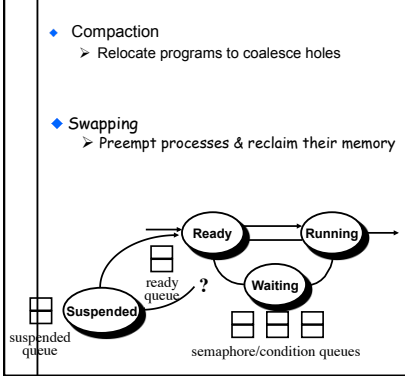
Advantages	Disadvantages
Works best if allocations are of medium sizes	Slow de-allocation
	External fragmentation
	Tends to break large free blocks such that large partitions cannot be allocated

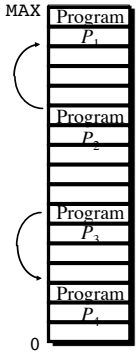
Allocation strategies

- First fit, best fit and worst fit all suffer from external fragmentation.
  - A. True
  - B. False

Dynamic Allocation of Partitions
Eliminating Fragmentation

- Compaction
  - Relocate programs to coalesce holes
- Swapping
  - Preempt processes & reclaim their memory

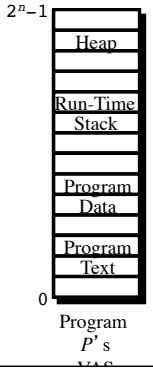




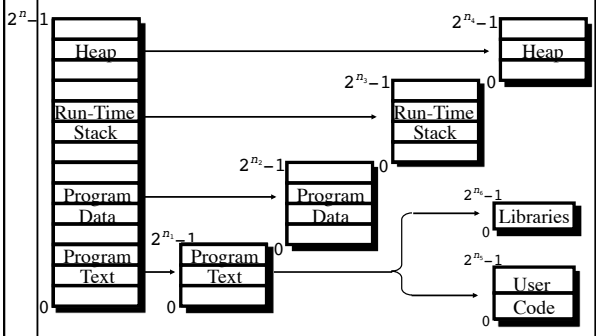
Memory Management
Sharing Between Processes

- Schemes so far have considered only a single address space per process
  - A single *name space* per process
  - No sharing

How can one share code and data between programs without paging?

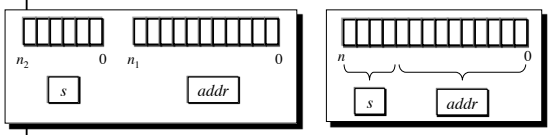


Multiple Name Spaces
Example — Protection/Fault isolation & sharing



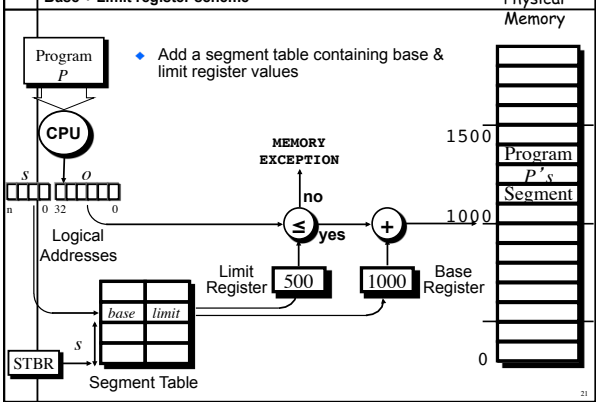
Supporting Multiple Name Spaces
Segmentation

- New concept: A *segment* — a memory "object"
  - A virtual address space
- A process now addresses objects —a pair (*s*, *addr*)
  - *s* — segment number
  - *addr* — an offset within an object
    - Don't know size of object, so 32 bits for offset?



Segment + Address register scheme Single address scheme

Implementing Segmentation
Base + Limit register scheme



Memory Management Basics
Are We Done?

- Segmentation allows sharing
- … but leads to poor memory utilization
  - We might not use much of a large segment, but we must keep the whole thing in memory (bad memory utilization).
  - Suffers from external fragmentation
  - Allocation/deallocation of arbitrary size segments is complex
- How can we improve memory management?
  - Paging