Virtual Memory

Mendel Rosenblum

Virtual Memory

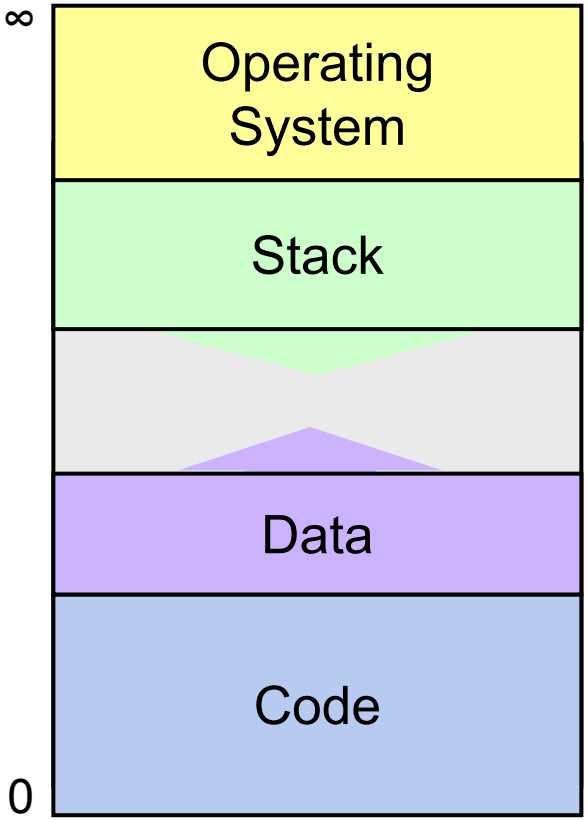
Optional readings: Operating Systems: Principles and Practice: Chapter 8

- Previously:
  - A single core shared by several concurrent threads
- Starting Today:
  - A single memory shared by several concurrent processes
  - Virtualize the memory

Take historical view: start with the earliest computers and show evolution

Explain why

Single-tasking: Early batch monitors



- Early PCs (MS-DOS) used this as well
- Limitations:
  - Programs can't share memory
  - Bad programs can trash OS/machine

Goals of Memory Sharing

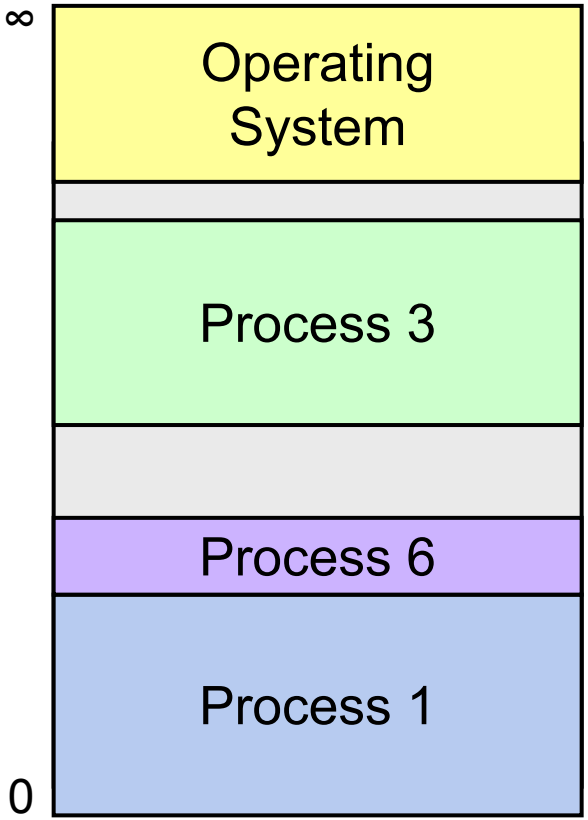
- Multitasking
  - Sharing means multiple processes to be memory-resident at once
- Transparency
  - Processes shouldn't be aware of the sharing (each process gets own memory)
- Isolation
  - Processes mustn't be able to corrupt each other or the OS
- Efficiency
  - CPU and memory efficiency shouldn't be degraded badly by sharing

Single-tasking evaluation

Goals:

- Multitasking: No
- Transparency: No
- Isolation: No
- Efficiency: Yes

Early sharing approach: Load-Time Relocation

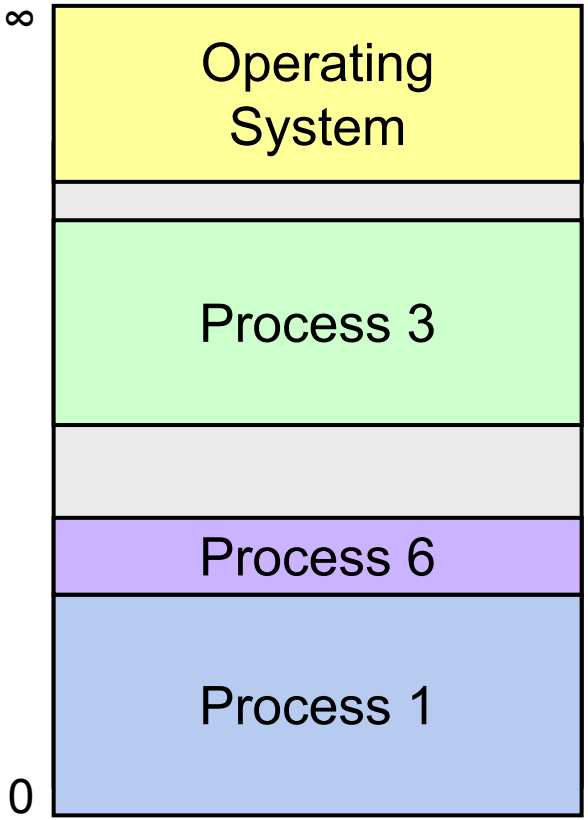


- Have loader relocate program when loading
  - Loader runs like linker to update addresses
  - Each process gets distinct memory region

Evaluation against goals:

- Multitasking: Yes
- Transparency?
- Isolation?
- Efficiency (CPU & Memory)?

Load-Time Relocation Limitations



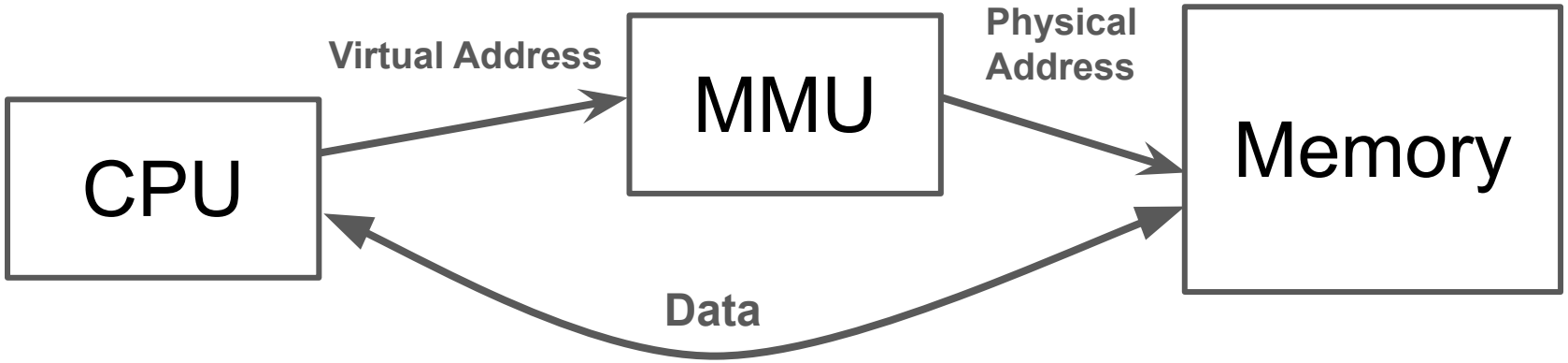
- Must declare process size statically
  - Can't grow process
- No isolation
  - Process can corrupt other processes & OS
- Poor efficiency: **Fragmentation**
  - Recall first-fit and best-fit
  - Can't move process

- Goals scoresheet

Multitasking:	Yes
Transparency:	No
Isolation:	No
Efficiency:	Yes

Key problem: Relocation of every instruction & data

Dynamic Address Translation



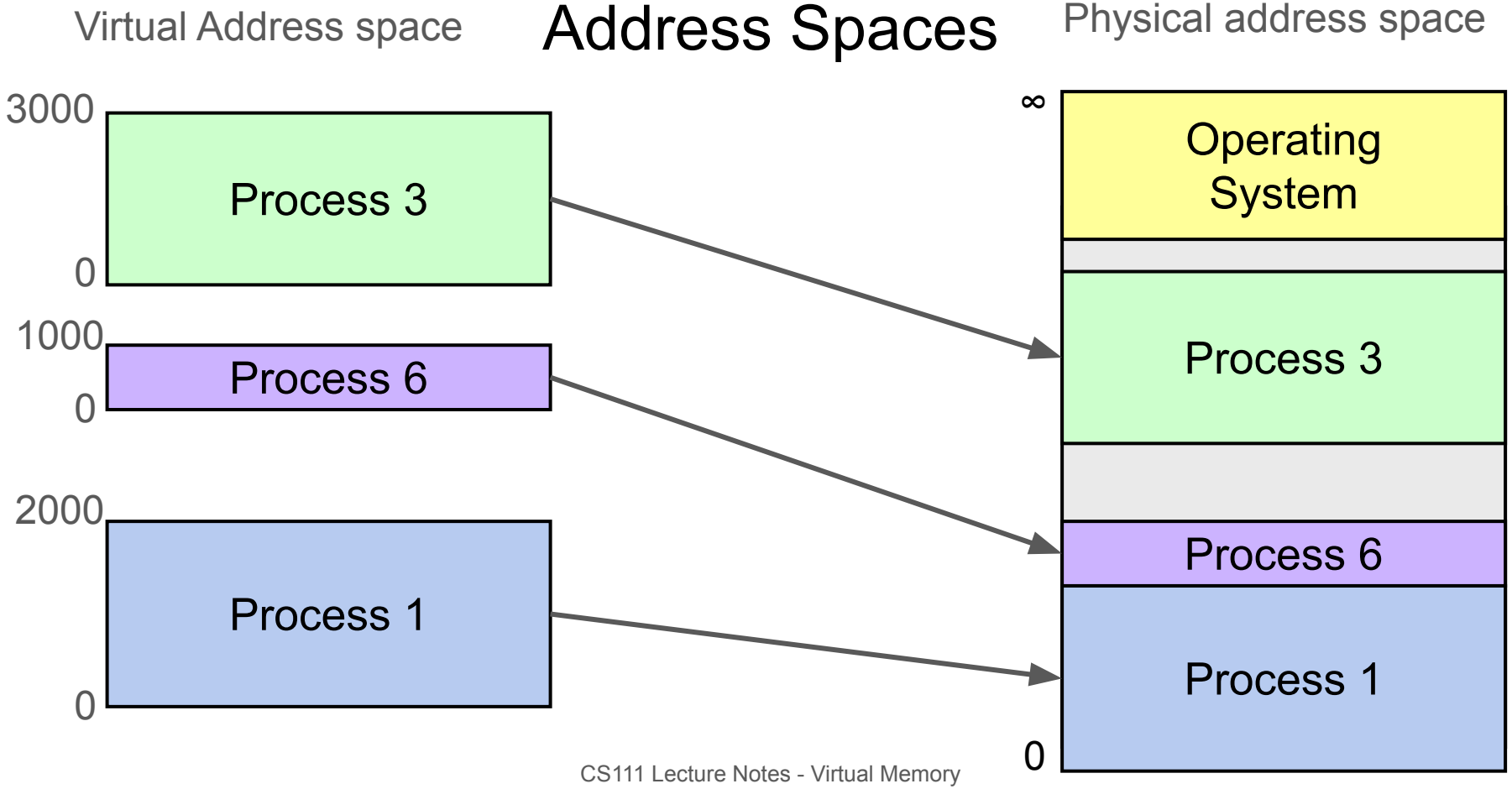
Memory **M**anagement **U**nit (MMU)

Hardware challenge: Every memory access translated

Two views of memory, called ***address spaces*:**

- **Virtual address space** is what the program sees
- **Physical address space** is the actual allocation of memory

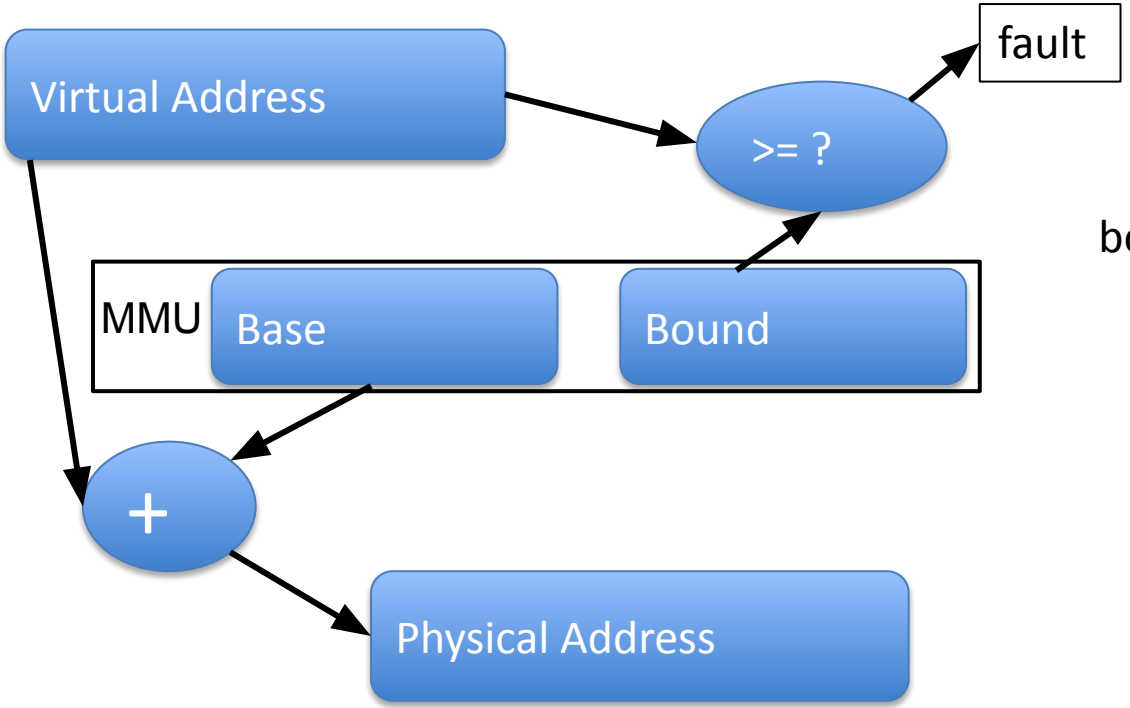
Address Spaces

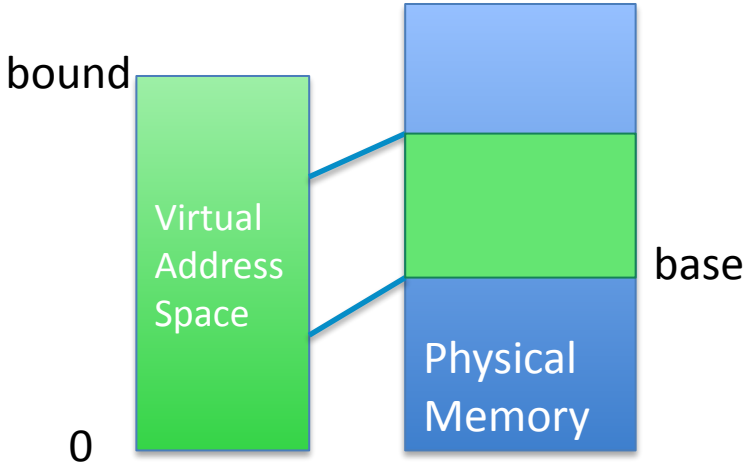


Early, simple hardware MMU

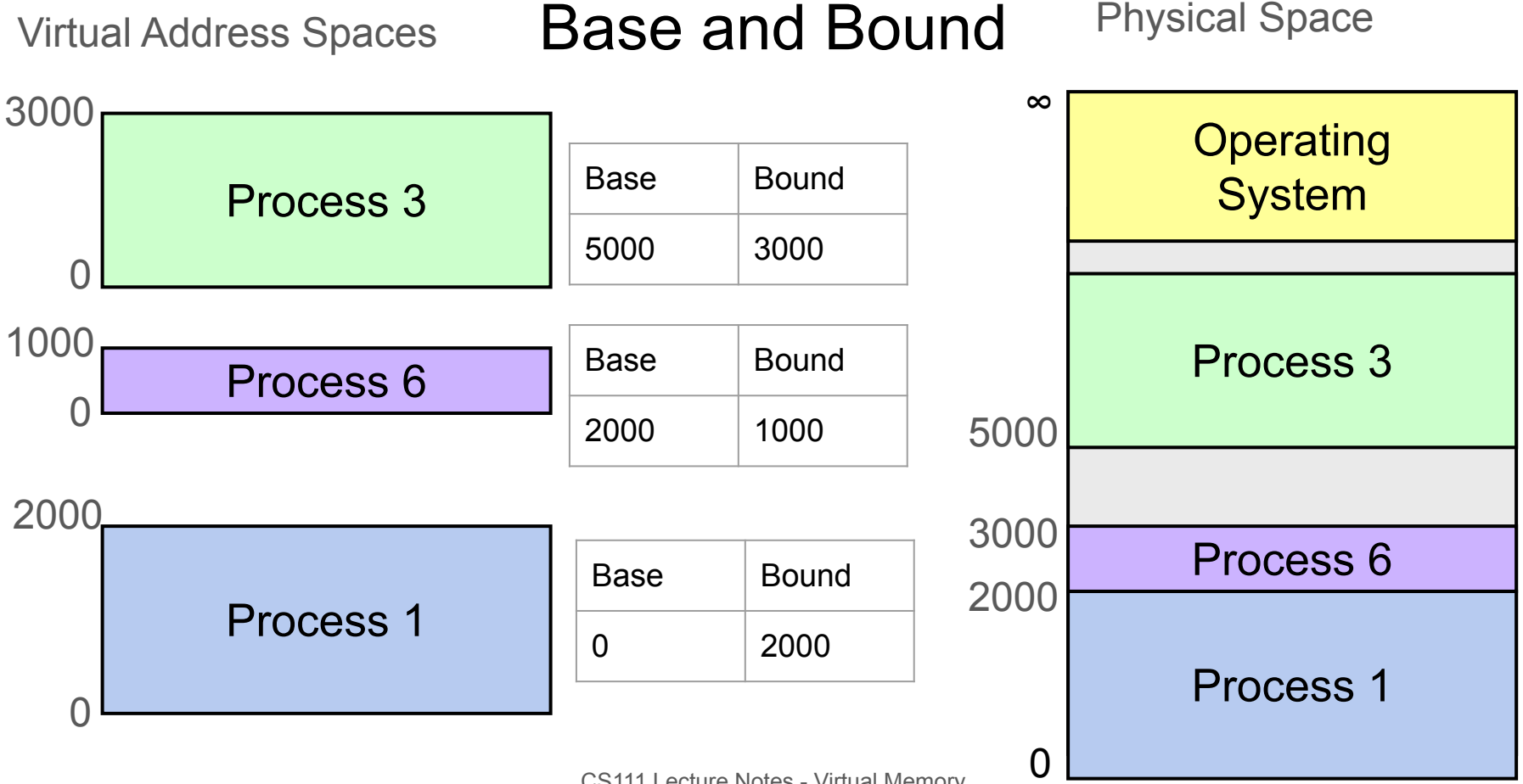
- Called **Base and Bound**
- Two hardware registers
  - **Base** - Location of process in physical memory
  - **Bound** - Size of the process virtual address space
- Operation done parallel:
  - Add Base to address
  - Compare address with bound - Error if out of bound

Base & Bound operation

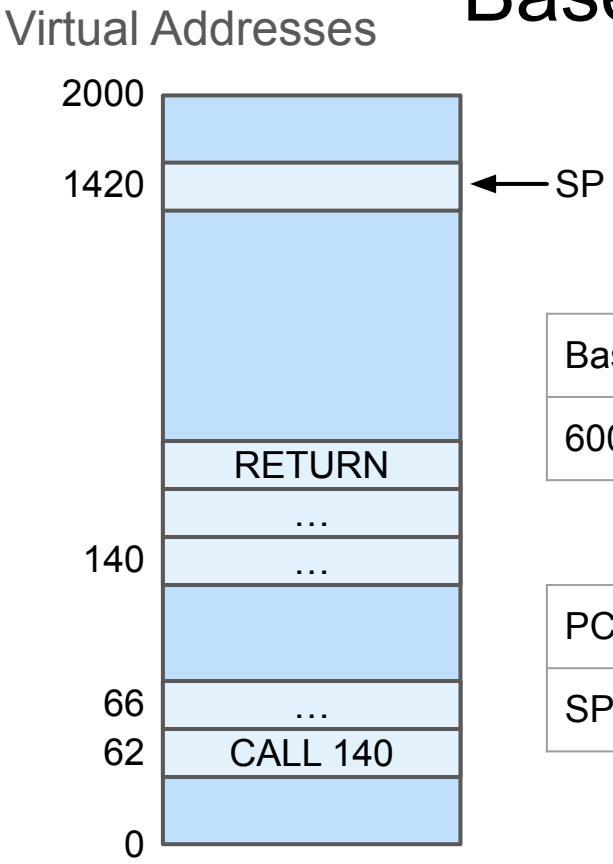




Base and Bound



Base & Bounds Example

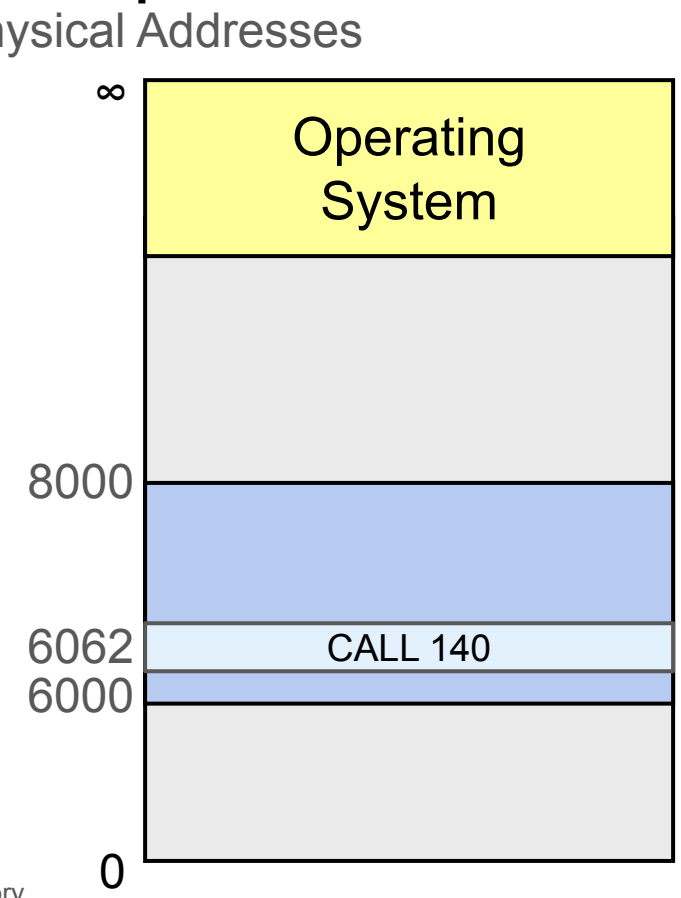


MMU

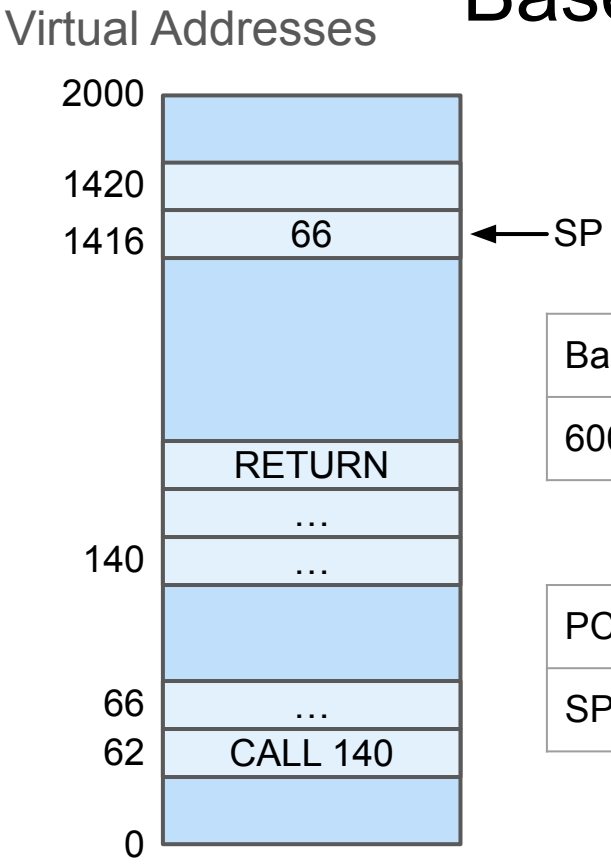
Base	Bound
6000	2000

CPU

PC	62
SP	1420



Base & Bounds Example

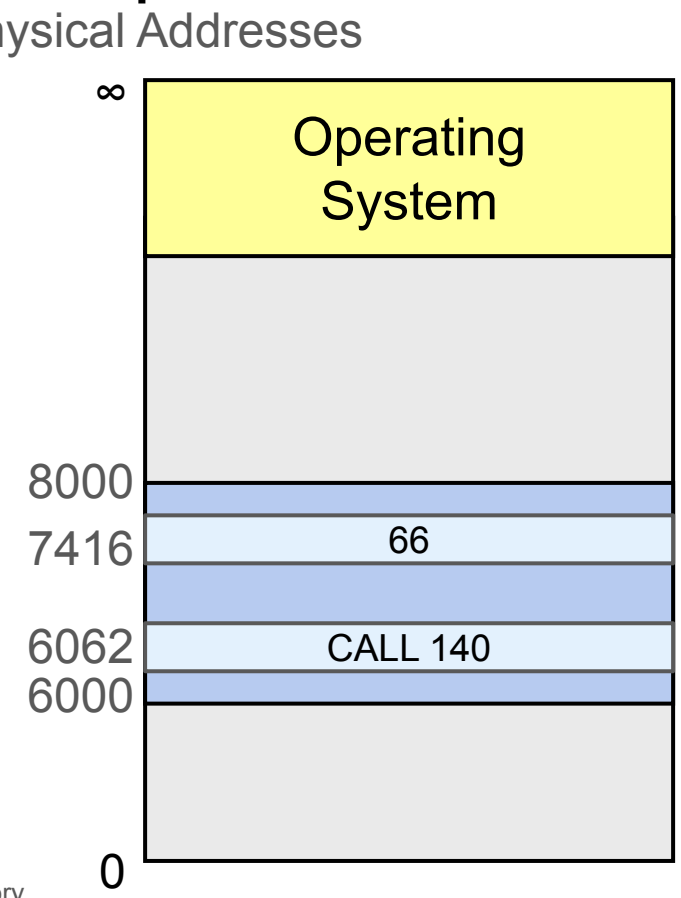


MMU

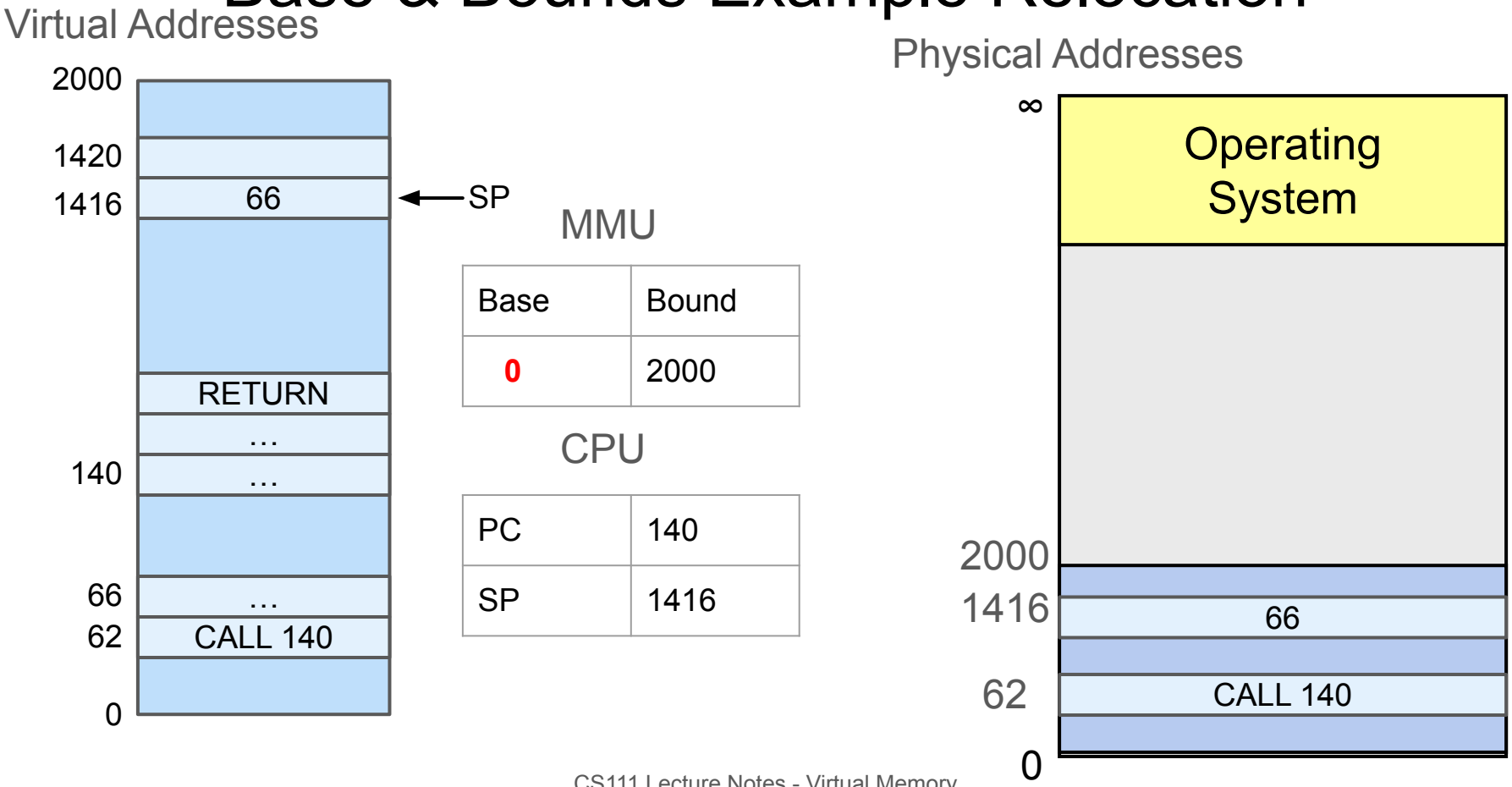
Base	Bound
6000	2000

CPU

PC	140
SP	1416



Base & Bounds Example Relocation



Process ⇔ OS transitions

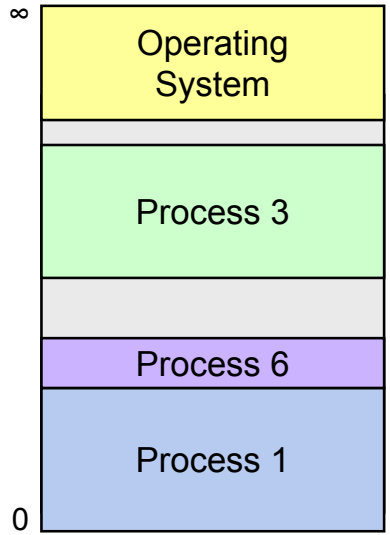
- OS runs with relocation/MMU disabled
  - Virtual address ⇒ Physical Address
  - Control in processor status register (PSR)
    - PSR bits: relocation on? User mode?

- Traps into OS - Atomically:
  - Save program counter (like procedure call)
  - Branch into OS (branch where? interrupt vectors)
  - Turn translation off & user mode off

- Return from trap - Atomically:
  - Turn translation and user mode back on
  - Restore saved program counter

Interrupt Vector Entry

Saved program counter
New program counter
New PSR value



Base and Bound evaluation

- Goals achieved:

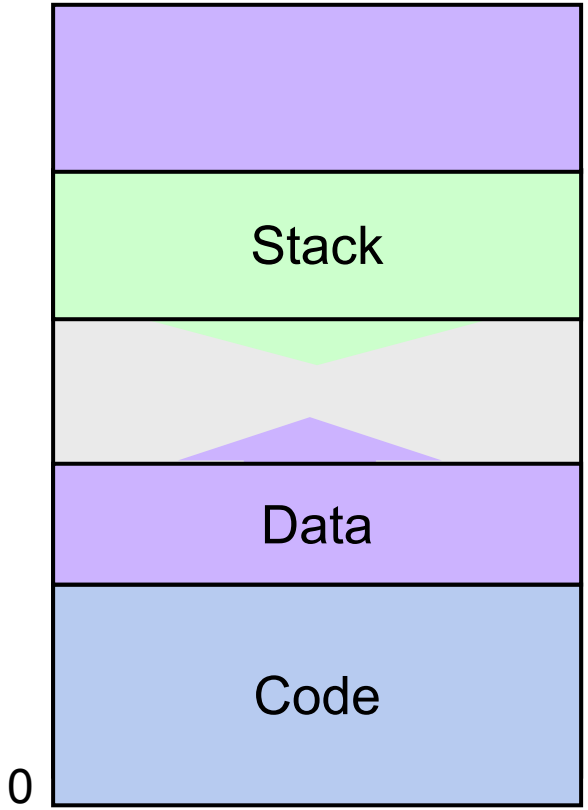
  Multitasking: Yes
  Transparency: Yes
  Isolation: Yes
  Efficiency: Yes

What are the problems with base and bound?

Base & Bound Disadvantages

- One contiguous region per program limiting
  - Doesn't match program (code, data, stack, etc.)
- Memory sharing not supported
- Can't support read-only information (e.g code)
- Memory fragmentation
- Can't grow processes (usefully)

Bottomline: one region too limiting



Segmentation: Getting more than one region

Process memory is split among several variable-size segments

MMU - Segment map:

Type	Base	Bound	Protection
Code	1000	1000	R/O
Data	3000	2000	R/W
Stack	8000	2000	R/W

Memory mapping logic consists of table lookup + add + compare

Like base and bound in efficiency (simple hardware, low overhead)

Determining segment of memory reference

For memory reference need segment number and offset

- Top bits of address select segment, low bits the offset
  - Example: PDP-10 with high and low segments selected by high-order address bit
- Segment can be selected implicitly by the instruction
  - PDP-11 example: Instruction fetch from code segment, Data fetched from data segment
  - Original x86: segment specified in the instruction (prefixes on instructions)

Advantages of segmentation

- Manage each segment separately:
  - Grow and shrink independently
  - Swap to disk
- Can move segments to compact memory and eliminate fragmentation
- Can share segments between processes (e.g., shared code).

Disadvantages of segmentation

- Fixed number of segments still creates limits:
  - Can't mmap files.
- Variable length results in memory fragmentation
- Address space is rigidly divided