CS444/544 Operating Systems II

Prof. Sibin Mohan

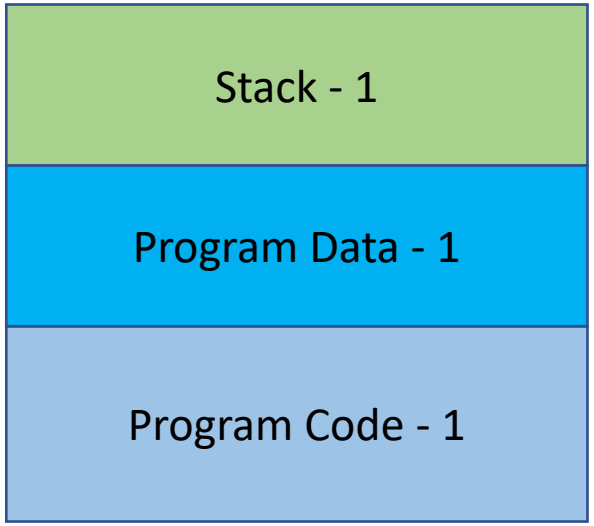
Spring 2022 | Lec4.1: Multiprogramming and Virtual Memory

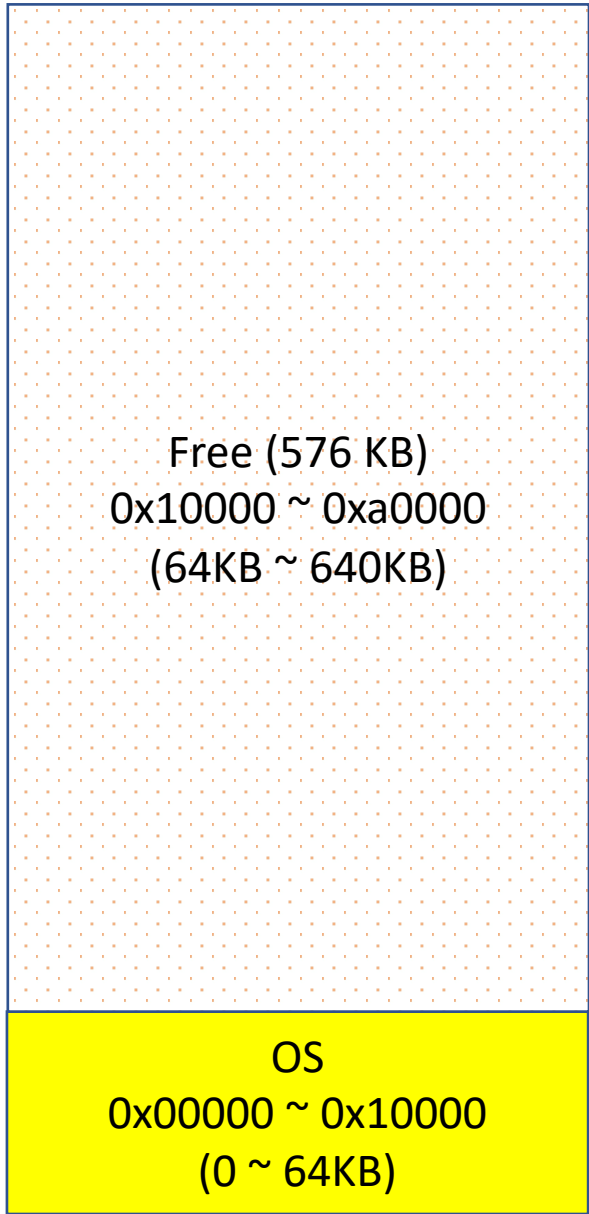
Adapted from content originally created by: Prof. Yeongjin Jang

Multiprogramming Challenges

Uniprogramming Environment

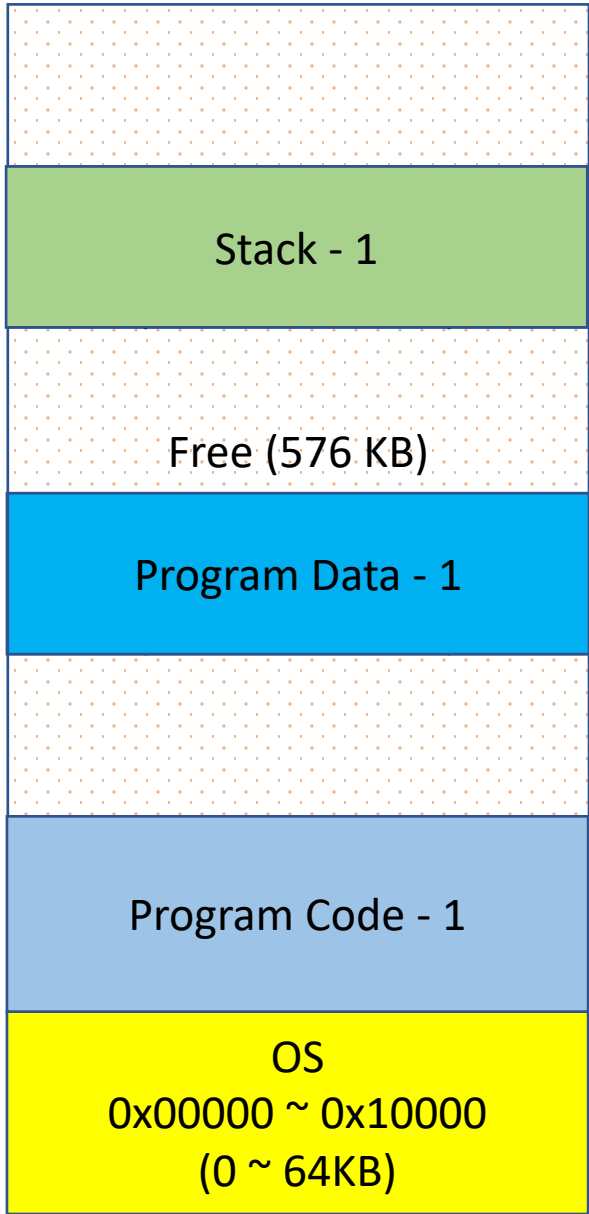
- Run one program
- It can use memory space freely





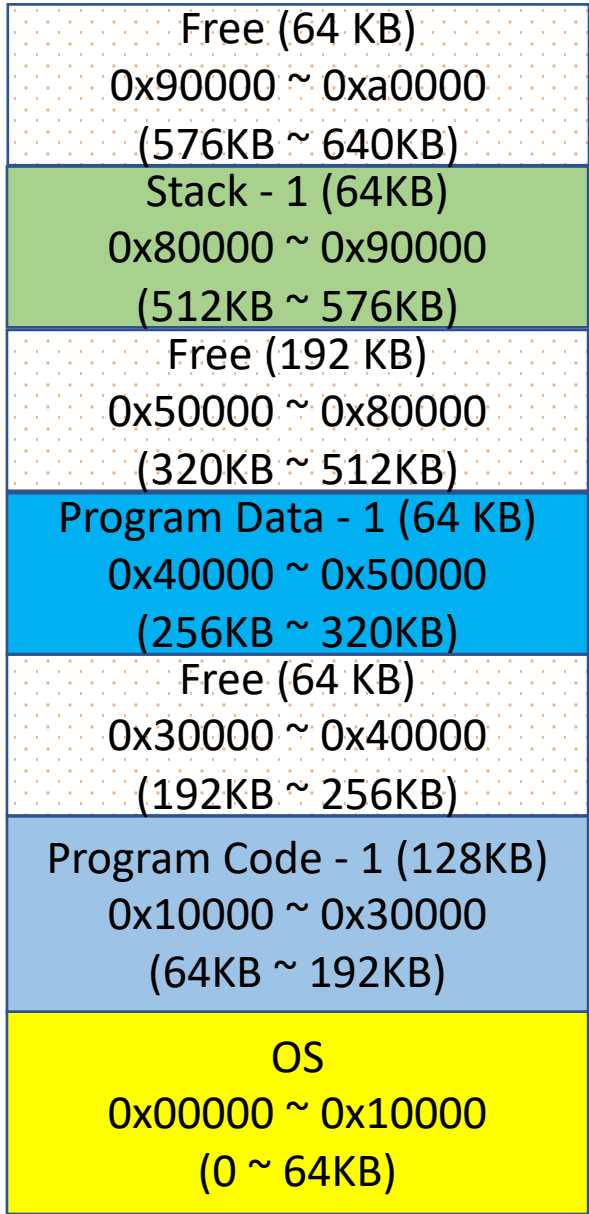
Uniprogramming Environment

- Run one program
- It can use memory space freely



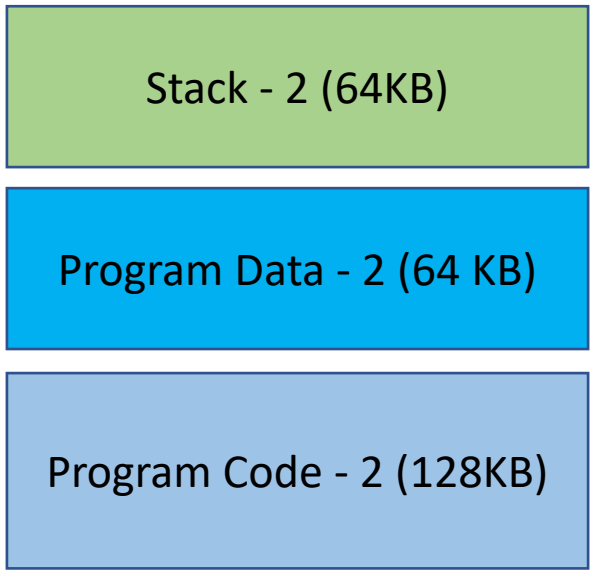
Uniprogramming Environment

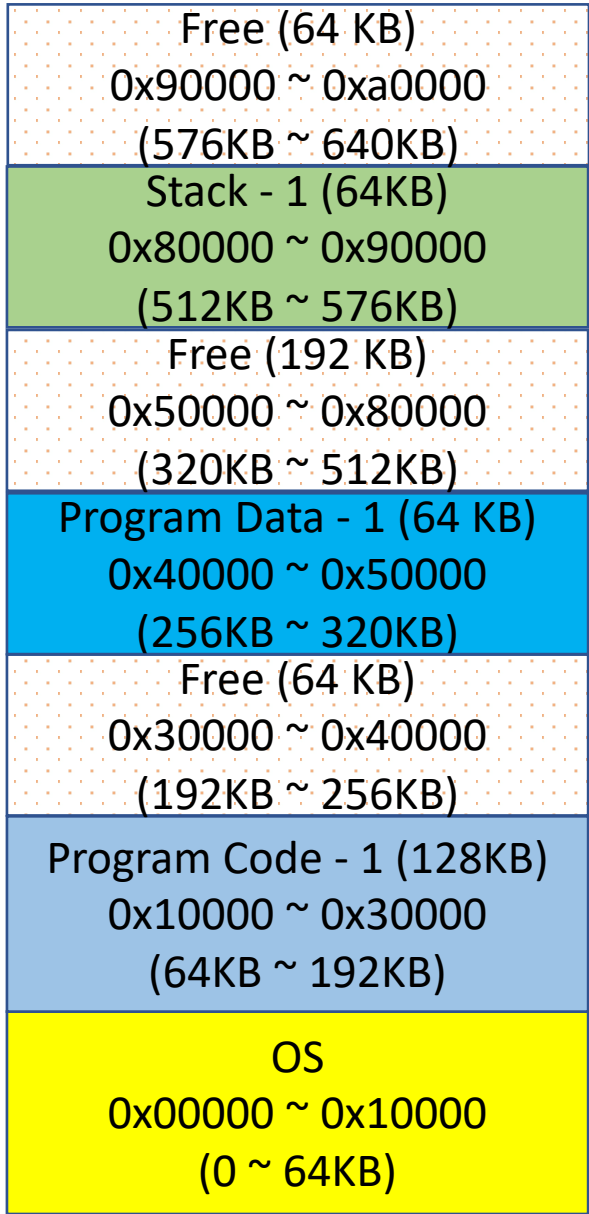
- Run one program
- It can use memory space freely



Multi-programming Environment

- Running **two** programs



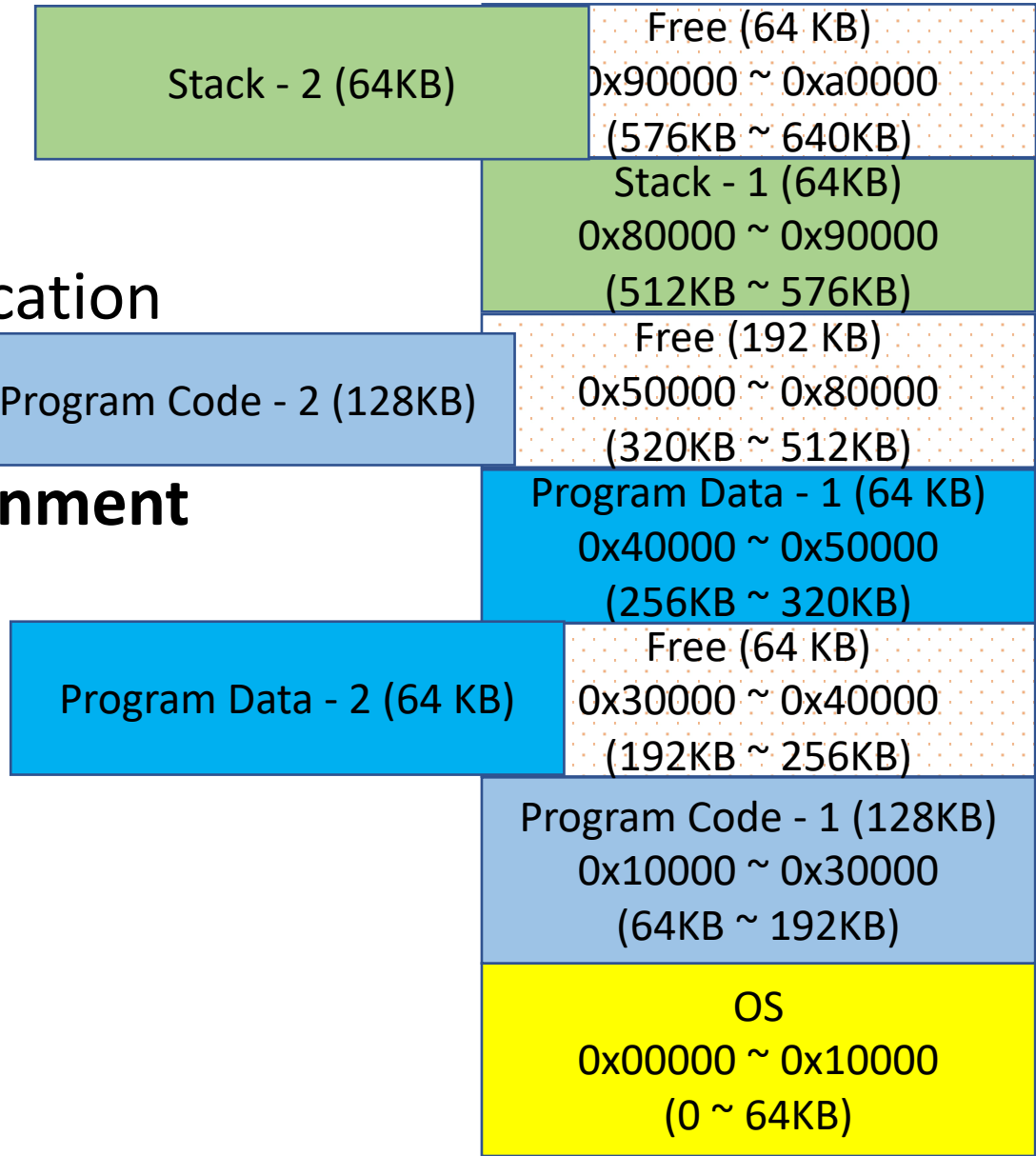


Multi-programming Environment

- Running **two** programs
- System's memory usage determines allocation

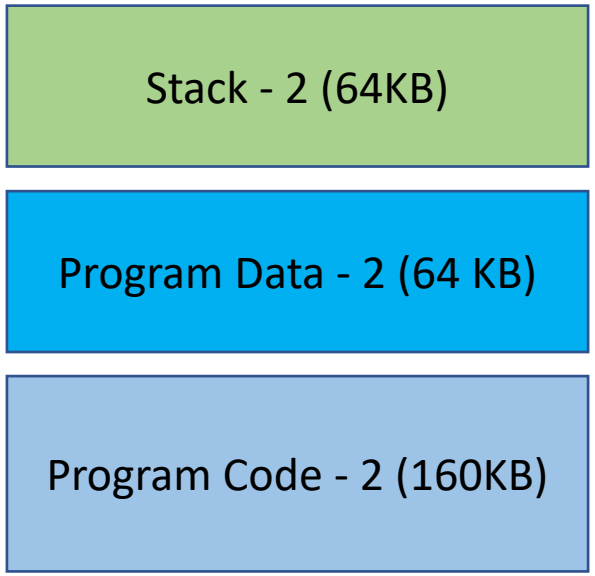
- Program need to be **aware of the environment**
  - Where does system load my code?
  - You can't determine, system does

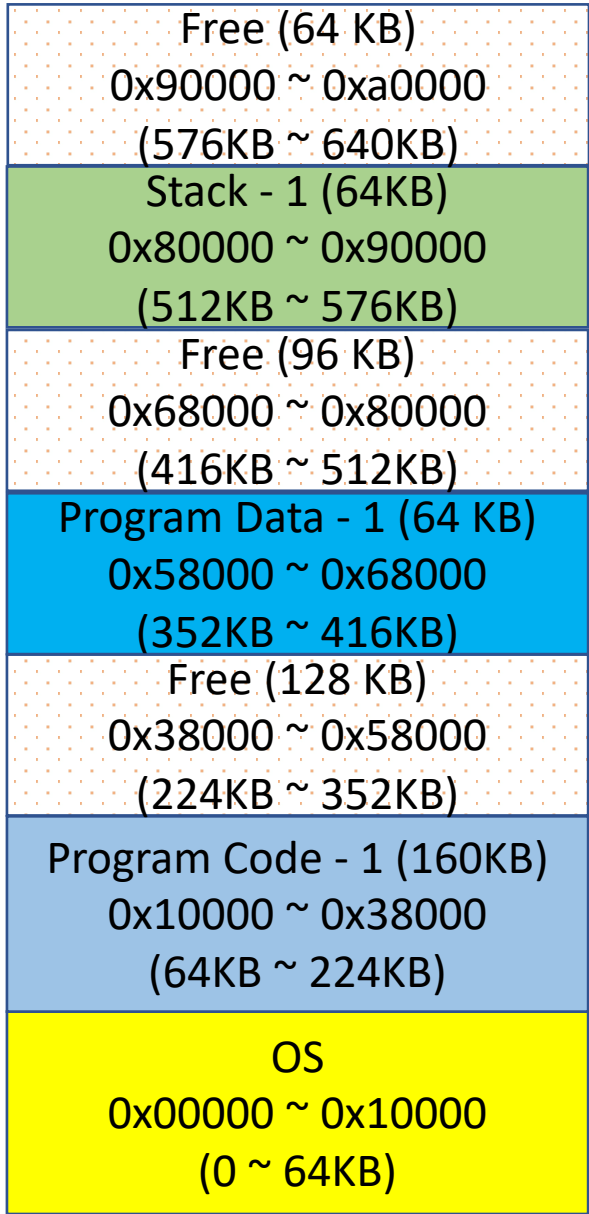
No Transparency!



Multi-programming Environment

- Running **two** programs

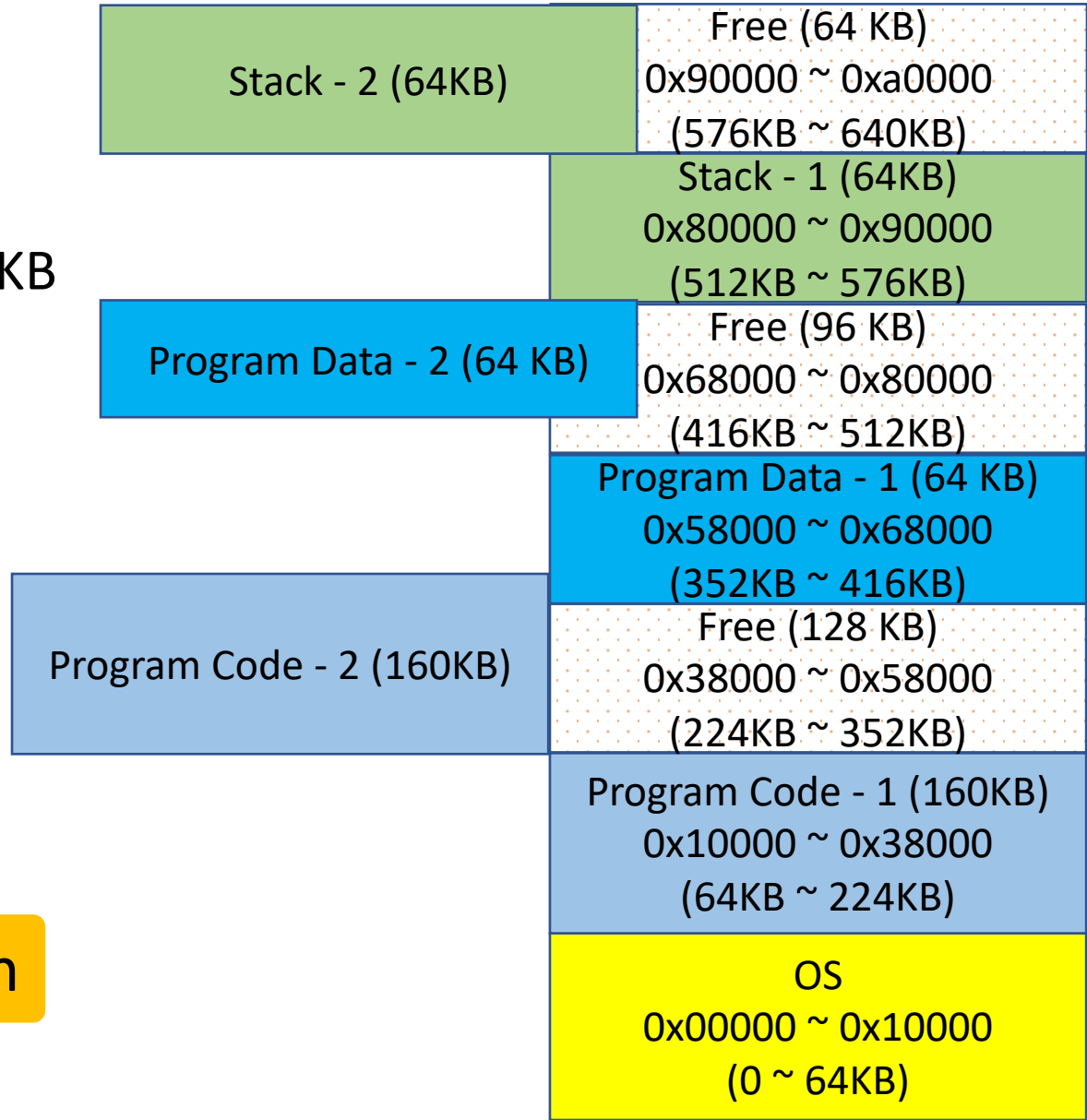




Multi-programming Environment

- Running **two** programs
  - Program size: 64KB + 64KB + 160K = 288KB
- Free memory
  - 64 + 96 + 128 = 288KB
- **Cannot run second Program**
  - Doesn't fit in memory

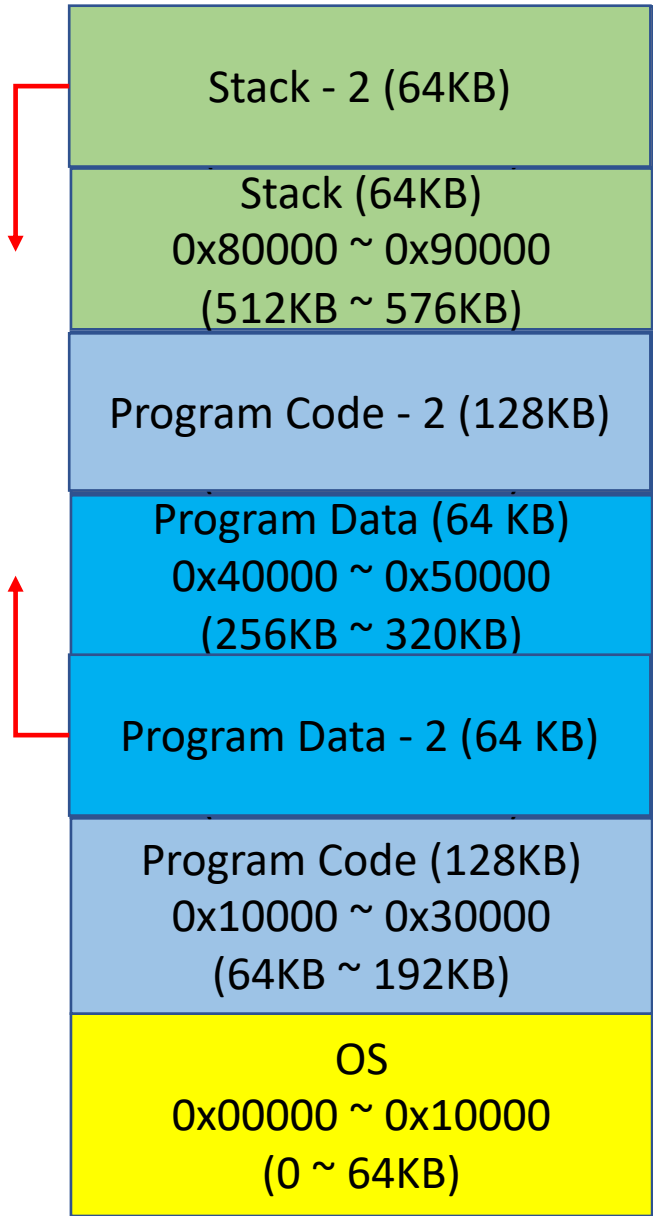
Not efficient → Memory fragmentation



Multi-programming Environment

- Running **two** programs
- What if Program-2's stack underflows?
- What if Program-2's data overflows?
- Programs can affect each other's execution

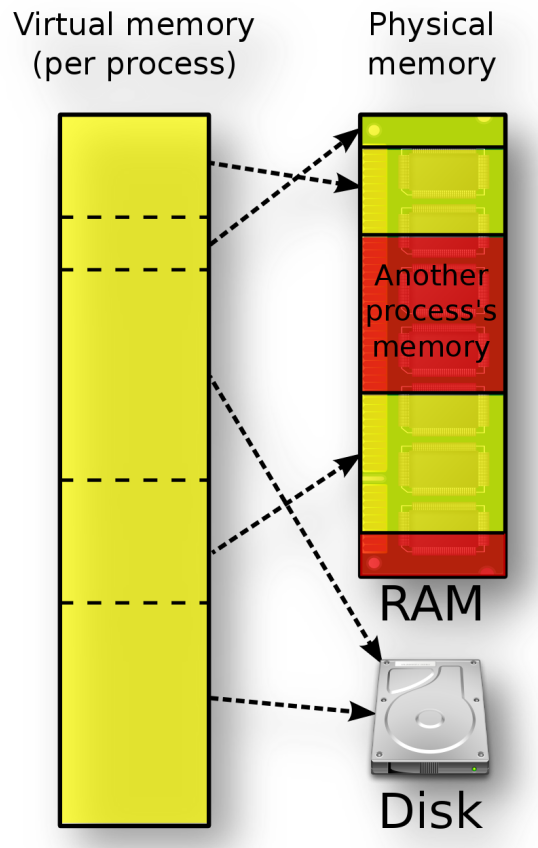
No isolation → Can lead to security problems



Virtual Memory to the Rescue!

So, what exactly is Virtual Memory?

- Memory management technique
- "idealized abstraction" of storage resources actually available
  - Creates and **illusion** of a very large main memory

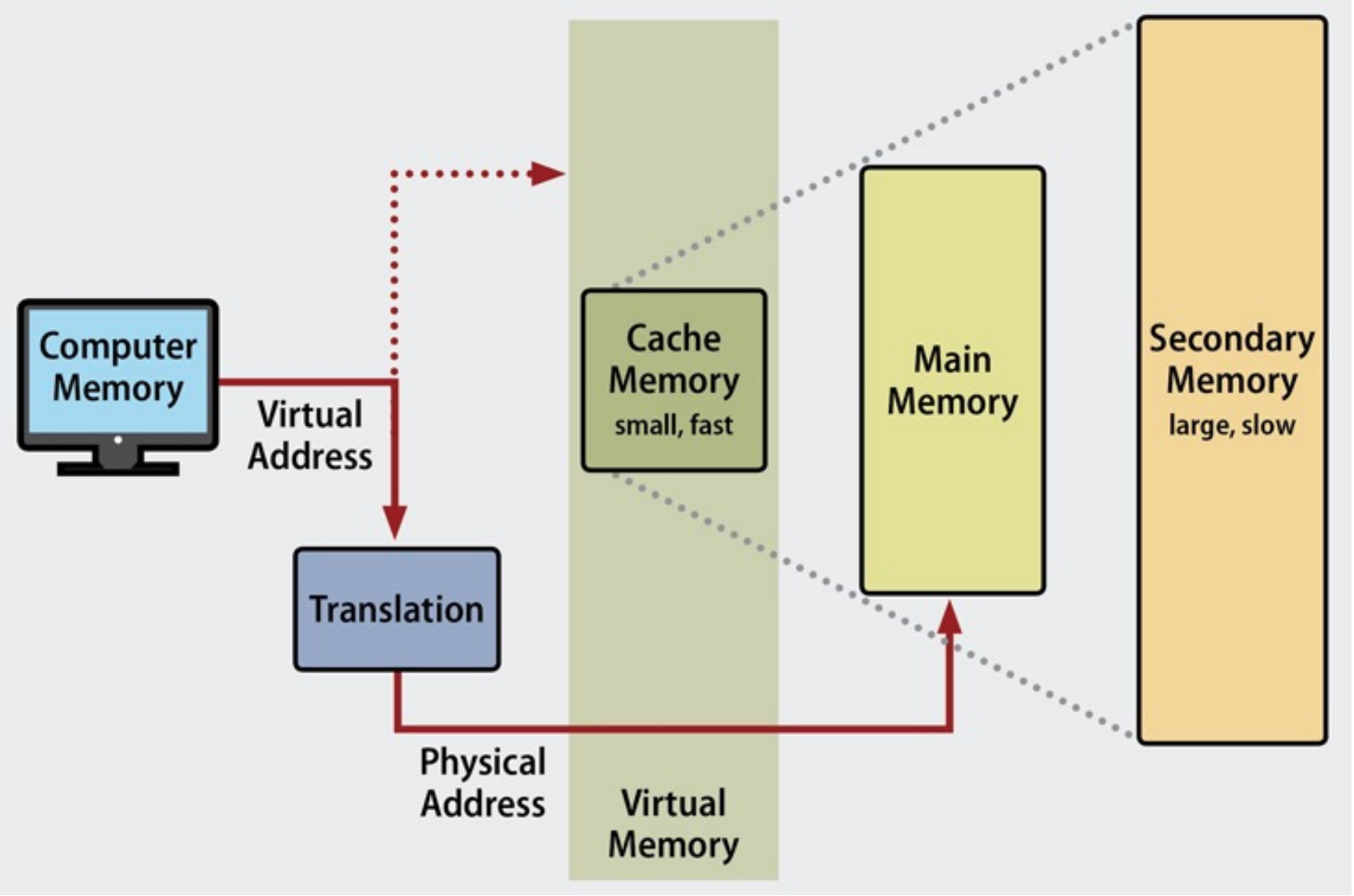


- **Programs can use more memory than is physically available**!

memory addresses used by a program [**virtual addresses**] → Mapping **OS** → actual addresses in computer memory [**physical addresses**]

- Main storage → **appears as contiguous address space**
- **Memory management unit [MMU]** → address translation hardware in CPU

Zoomed Out View of Virtual Memory



Virtual Memory

Three Goals!

TRANSPARENCY

EFFICIENCY

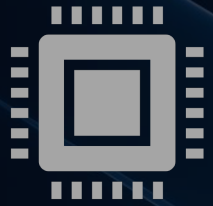
PROTECTION

Three Goals | Details

- **Transparency**: programs shouldn't need to know system's **internal state**
  - Program A is loaded at **0x8048000**. Can Program B be loaded at **0x8048000**?
- **Efficiency**: do not waste memory; avoid **memory fragmentation**
  - Can Program B (288KB) be loaded if 288 KB of memory is free, regardless of its allocation?
- **Protection**: **isolate** program's execution environment
  - Can we prevent an overflow from Program A from overwriting Program B's data?

Paging

A method to implement virtual memory



Split memory into multiple blocks [E.g. 4096 bytes, 12-bi])

Last 3 digits of page address are ZERO [hex]

E.g., 0x0, 0x1000, 0x2000, …, 0x8048000, 0x804a000, …, 0x7fffe000, etc.

An indirect map between virtual page and physical page

Set arbitrary virtual address for a page

e.g., 0x81815000

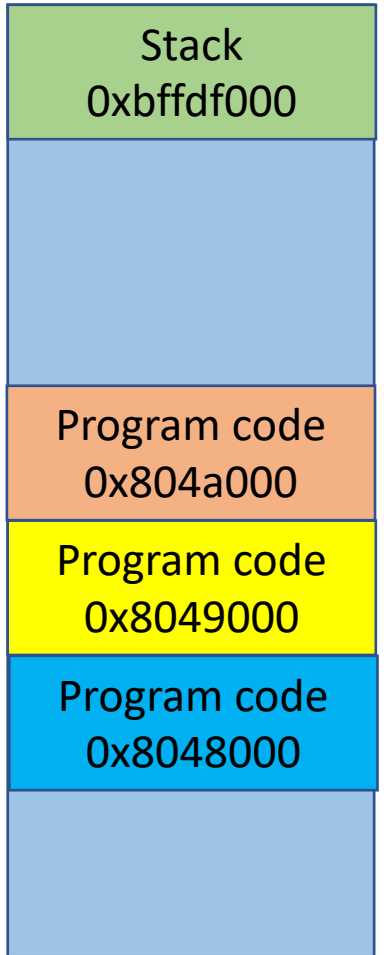
Set physical address to that page as a map

e.g., 0x32000

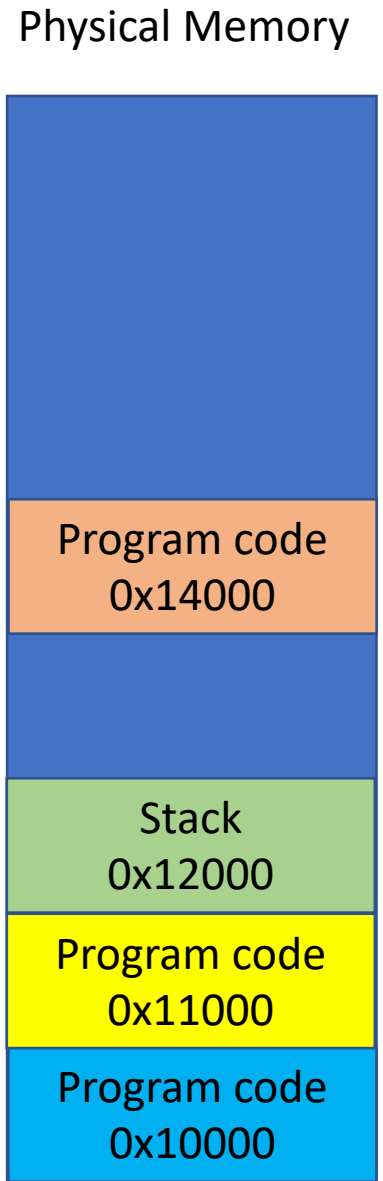
0x81815000 ~ 0x81815fff → 0x32000 ~ 0x32fff

Virtual Memory | Paging

- Uses an **indirect table** that **maps virt-addr → phys-addr**

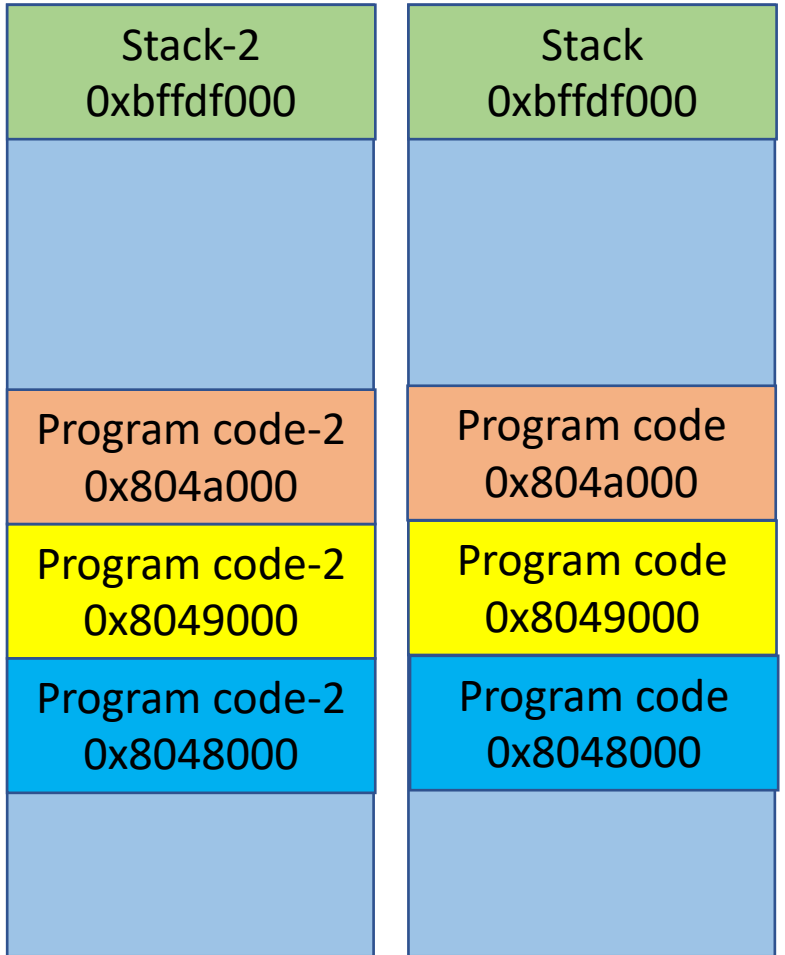


Virtual	Physical
0x8048000	0x10000
0x8049000	0x11000
0x804a000	0x14000
0xbffdf000	0x12000
…	…



Paging: Virtual Memory

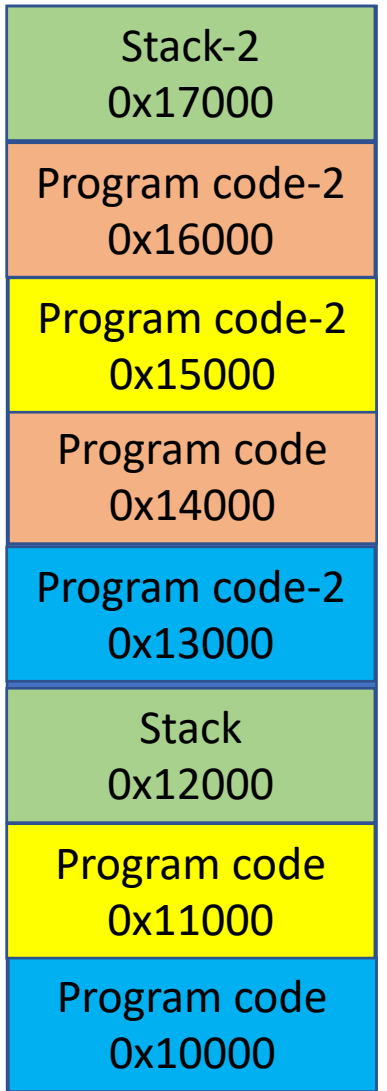
- Uses an **indirect table** that **maps virt-addr → phys-addr**



Virtual	Physical
0x8048000	0x10000
0x8049000	0x11000
0x804a000	0x14000
0xbffdf000	0x12000
…	…

Virtual-2	Physical-2
0x8048000	0x13000
0x8049000	0x15000
0x804a000	0x16000
0xbffdf000	0x17000
…	…

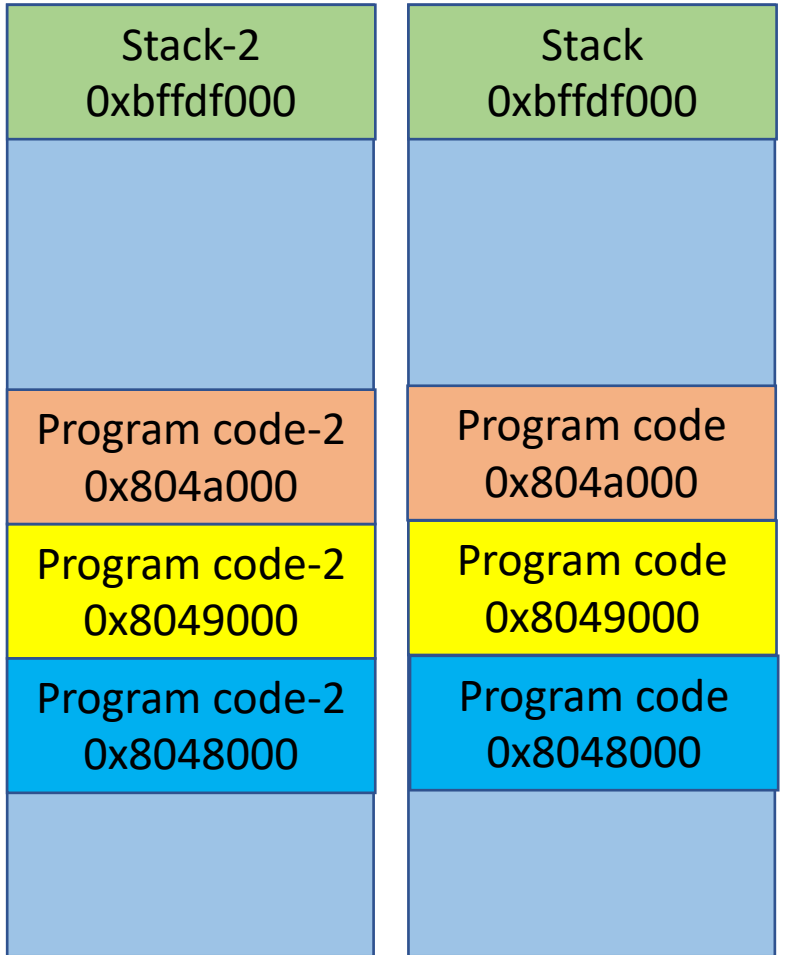
Physical Memory



Paging: Virtual Memory

Transparency: does not need to know system's internal state

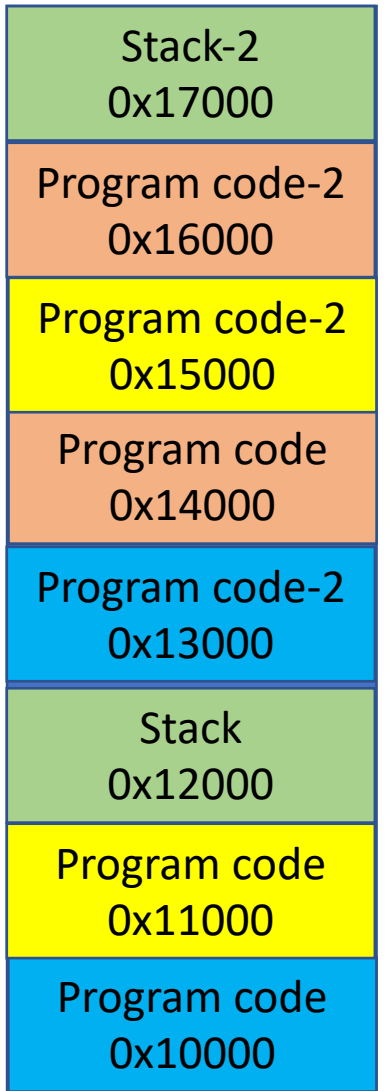
Program A & B loaded at **0x8048000?**



Virtual	Physical
0x8048000	0x10000
0x8049000	0x11000
0x804a000	0x14000
0xbffdf000	0x12000
…	…

Virtual-2	Physical-2
0x8048000	0x13000
0x8049000	0x15000
0x804a000	0x16000
0xbffdf000	0x17000
…	…

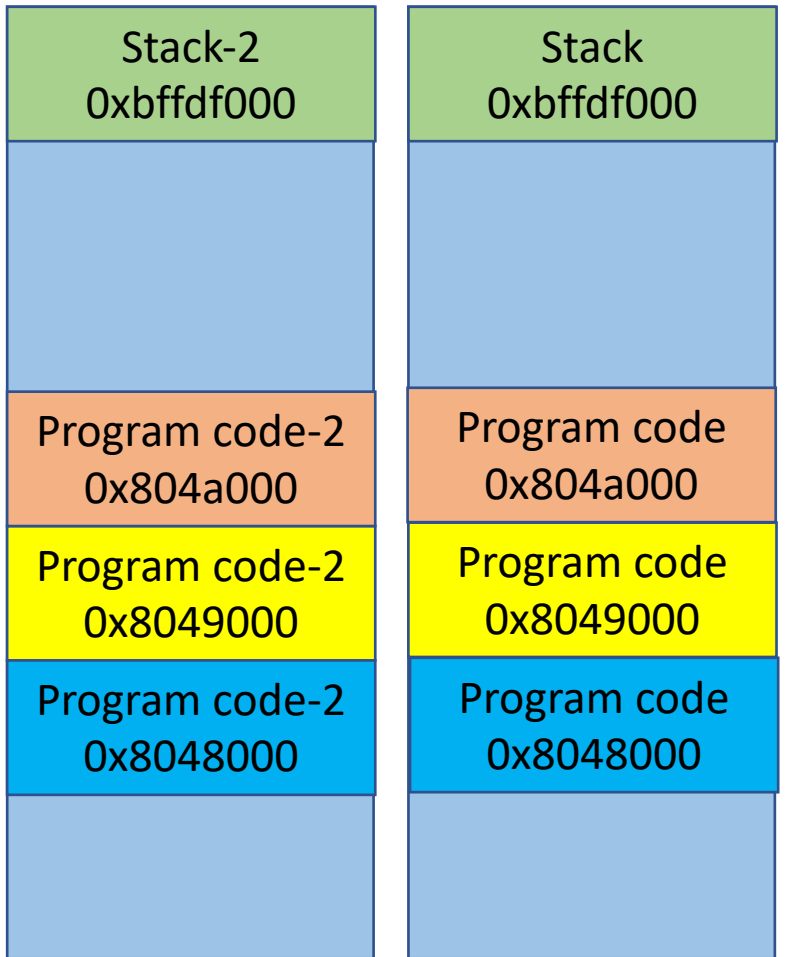
Physical Memory



Efficiency: do not waste memory

Can Program B (288KB) be loaded if only 288 KB of memory is free, regardless of its allocation?

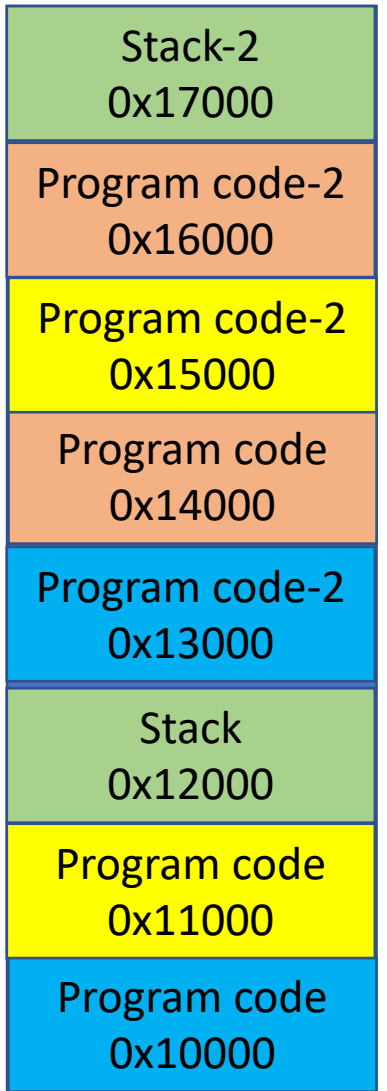
- Uses an **indirect table** that **maps virt-addr → phys-addr**



Virtual	Physical
0x8048000	0x10000
0x8049000	0x11000
0x804a000	0x14000
0xbffdf000	0x12000
…	…

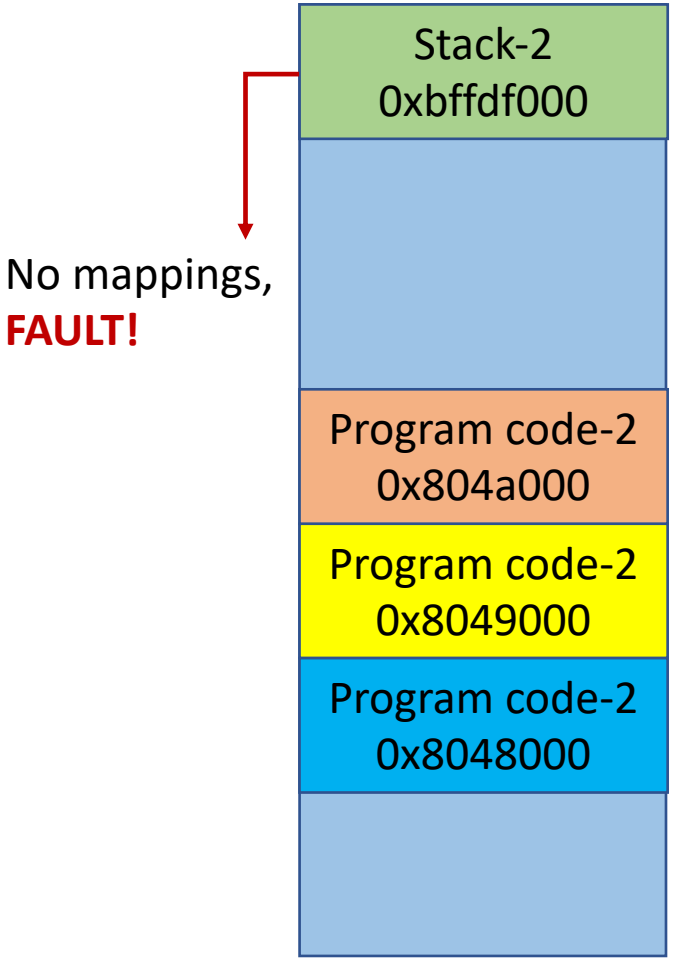
Virtual-2	Physical-2
0x8048000	0x13000
0x8049000	0x15000
0x804a000	0x16000
0xbffdf000	0x17000
…	…

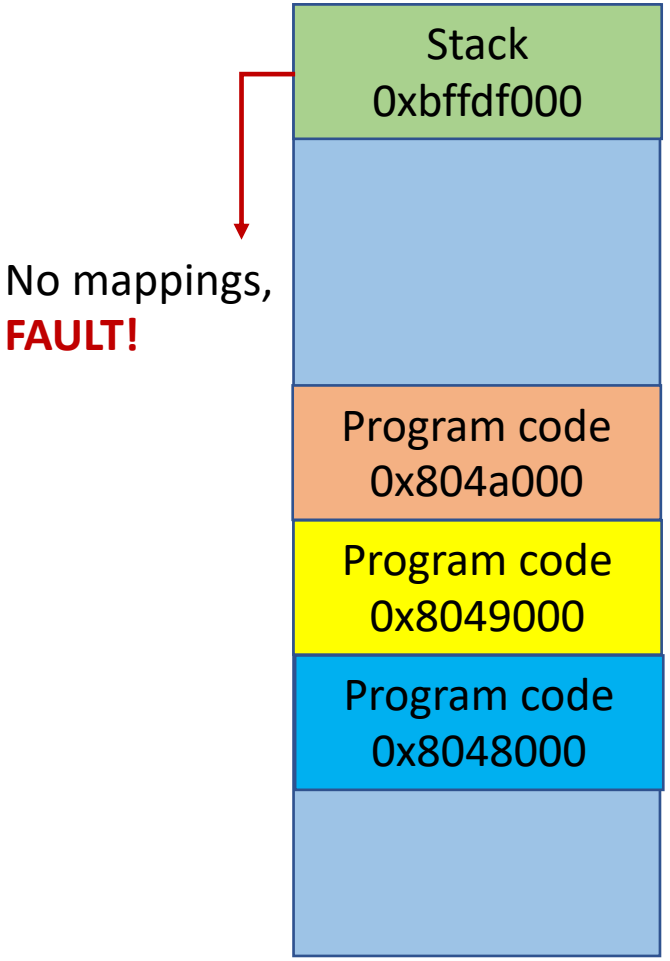
Physical Memory



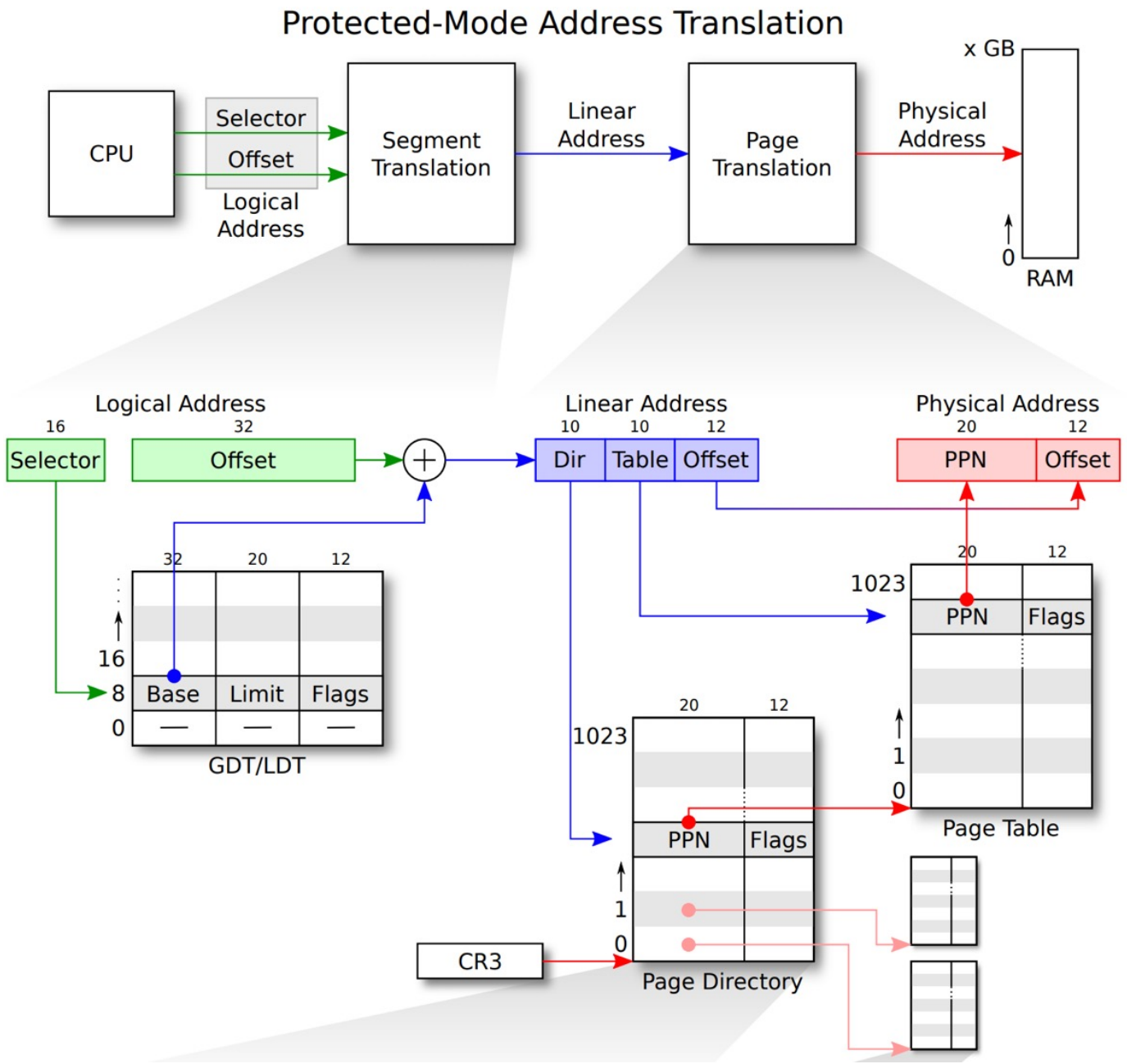
Paging: Virtual Memory

Protection: **isolate** program's execution environment
Prevent overflow/overwriting between multiple programs?





x86 Memory Access



Additional Reading

- ELF Format
  https://en.wikipedia.org/wiki/Executable_and_Linkable_Format