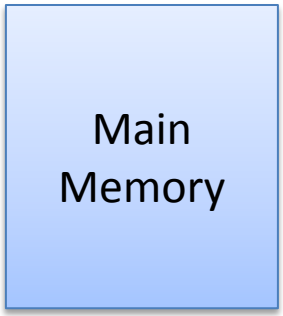
CS305
Computer Architecture

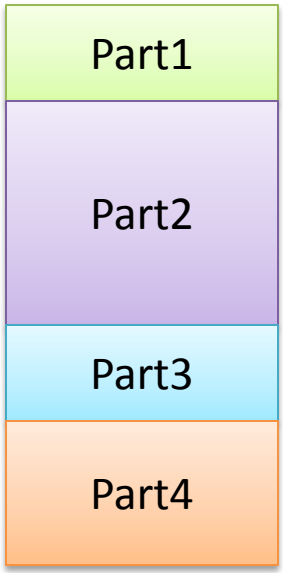
Virtual Memory

Bhaskaran Raman
Room 406, KR Building
Department of CSE, IIT Bombay

http://www.cse.iitb.ac.in/~br

Programmer's Burden: Program Size > Main Memory Size





Initially Part1, Part2

↓

Then Part1, Part3

↓

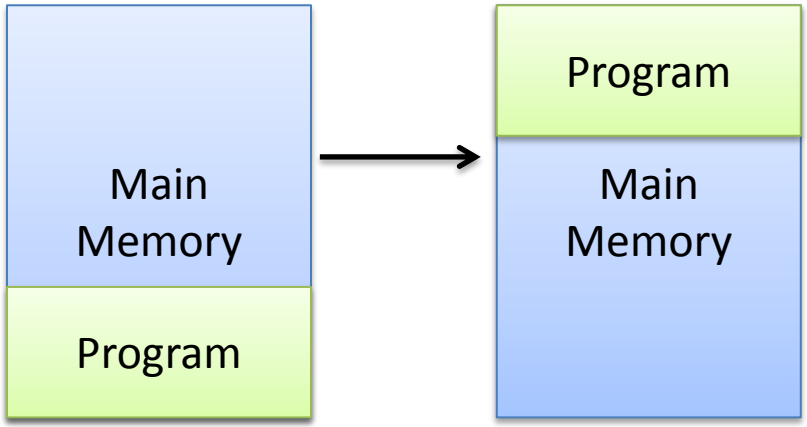
Then Part1, Part4

Program had to deal with bringing in relevant parts of the program, into main memory!

Not a problem today: most programs < memory size

Illusion required: memory larger than main memory

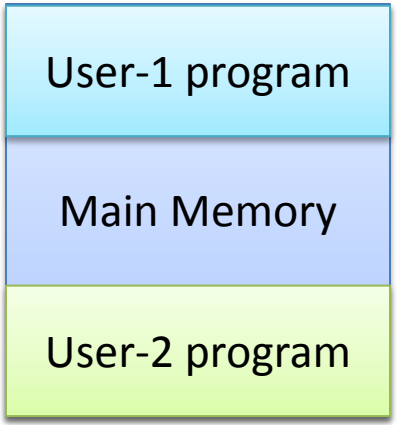
Loader's Burden: Program Relocation



Program, or parts of program need to be constantly rewritten, if loaded to different parts of memory

Illusion required: program at same memory location

OS's Burden: Program Safety



Different user's programs must be protected from one another: malicious or buggy behaviour

Illusion required: each program thinks it has the entire memory to itself, it is the only program

Virtual Memory: Illusions Galore

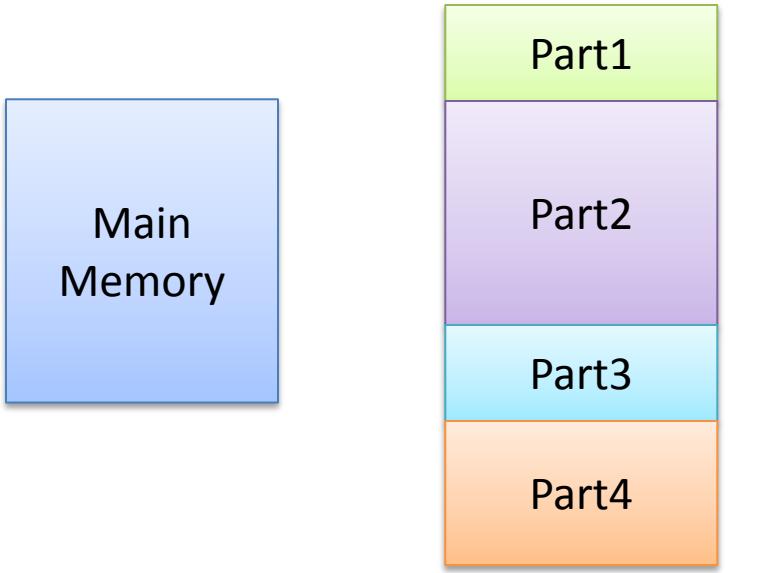
Illusion-1: Larger memory than main memory

Illusion-2: Program at same memory location

Illusion-3: The current program is the only program, with entire memory to itself

Illusion-N… E.g. with shared dynamic libraries, shared memory, etc.

Virtual Memory: The Cache Perspective

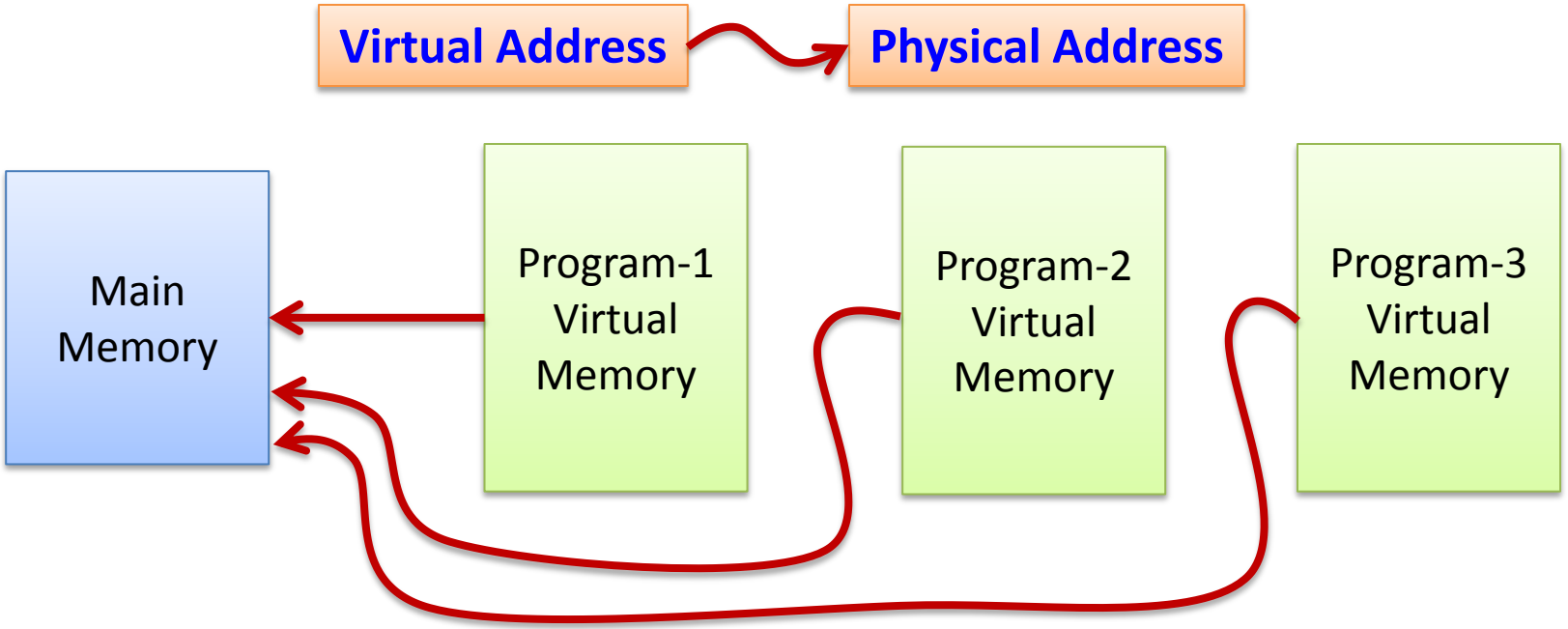


Main memory is a cache for virtual memory

Removes programmer's burden

Incomplete perspective: e.g. main memory can be 8GB, virtual memory 4GB

Virtual Memory:
The Level-of-Indirection Perspective



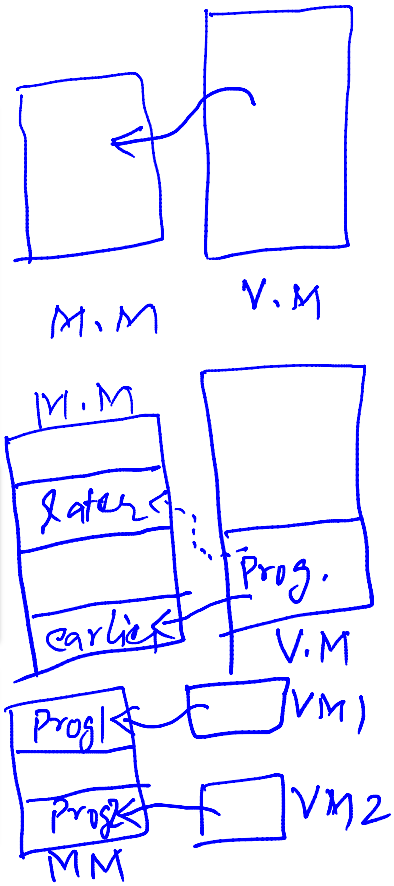
OS, with hardware support, maintains VA → PA mapping

Level-of-Indirection: The Basis for All Illusions

Virtual address (VA) → physical address (PA) mapping:
Like main memory address → cache location mapping
Can support much larger virtual address space than physical (main) memory

Program relocation: change VA → PA mapping!
Program (in terms of VA) remains unchanged

Program safety: separate VA space for each program
No chance of inadvertent access by another program



The Cache Perspective

L1 as Cache for MM	MM as Cache for VM
Unit of transfer = block	Unit of transfer = page
Not in cache ➔ miss	Not in cache ➔ page fault
Mapping: using MM addr, tags	Mapping: using page table, other schemes
Miss penalty: a few 100 cycles	Miss penalty: to disk ➔ a few ms! Millions or billions of cycles!

VM Design Decisions in the Cache Perspective

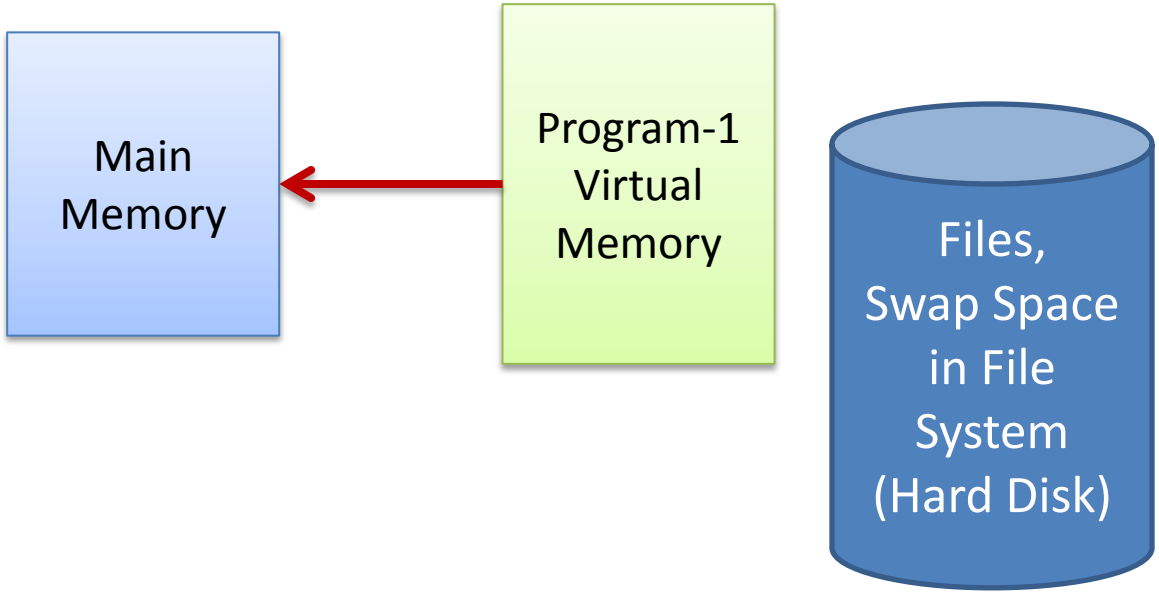
Strategy-1: Reduce miss rate

- Page size: large (e.g. 32KB)
- Associativity: full
- Write-back scheme
- Write-allocate scheme
- Page fault: handled in software (OS), sophisticated replacement policy

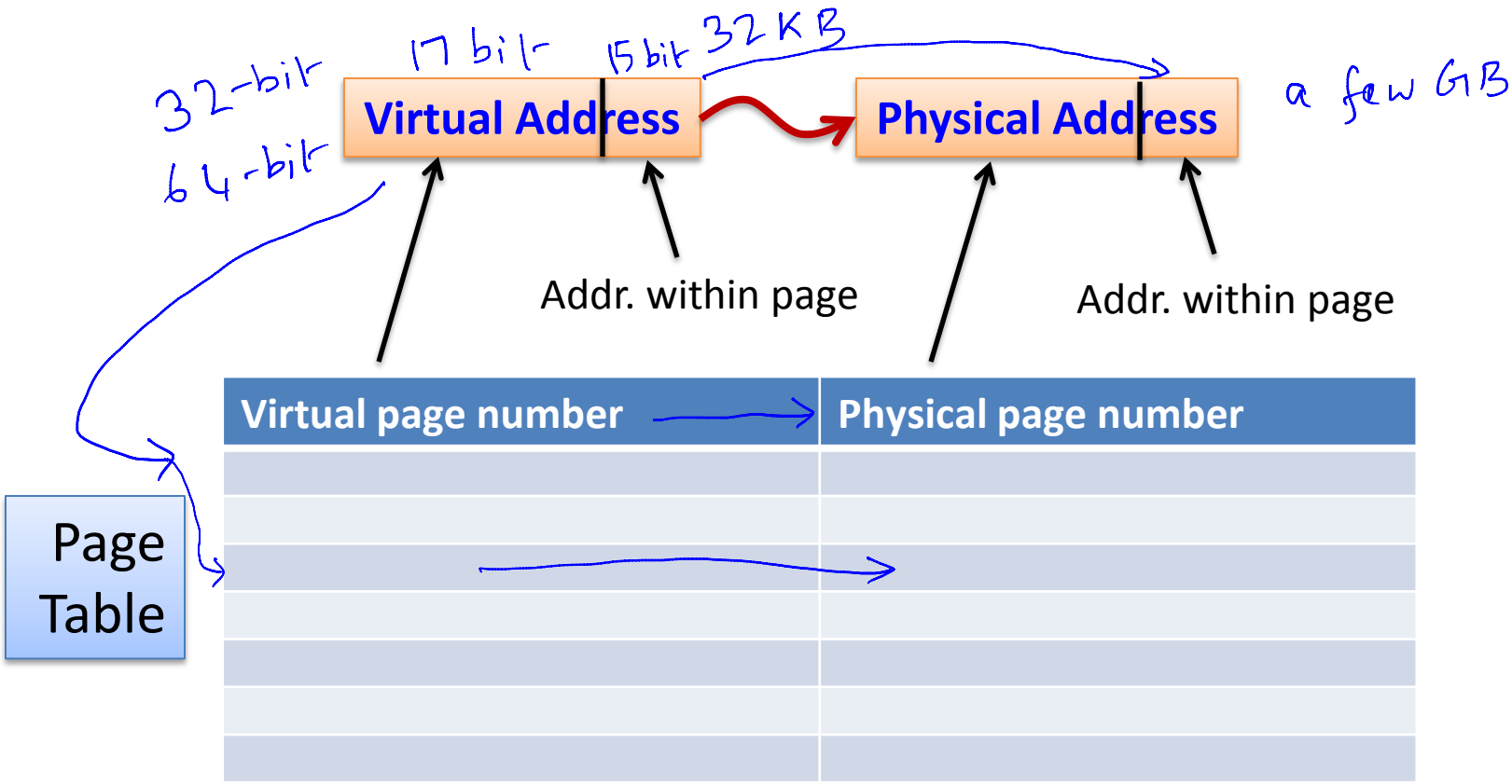
Strategy-2: Multi-tasking

- When a program has a miss (page fault), run some other program!

The Swap Space



The VA-to-PA Mapping: Page Table



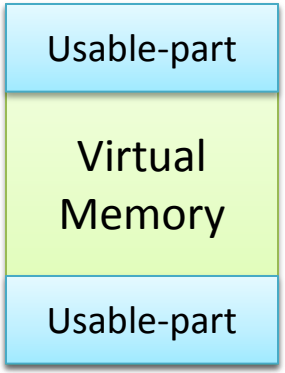
Page Table: A Numeric Example

32-bit VA space, 8KB pages
Say, 4 bytes per page table entry
Page table size = ?

pages in VM = $2^{32}/2^{13} = 2^{19}$
Page table size = 2^{21} bytes = 2MB !

This is per process !
64-bit VA space ➔ page table much larger!
What can we do?

Dealing with Large Mapping Size: Solution-1: Page Table Limits



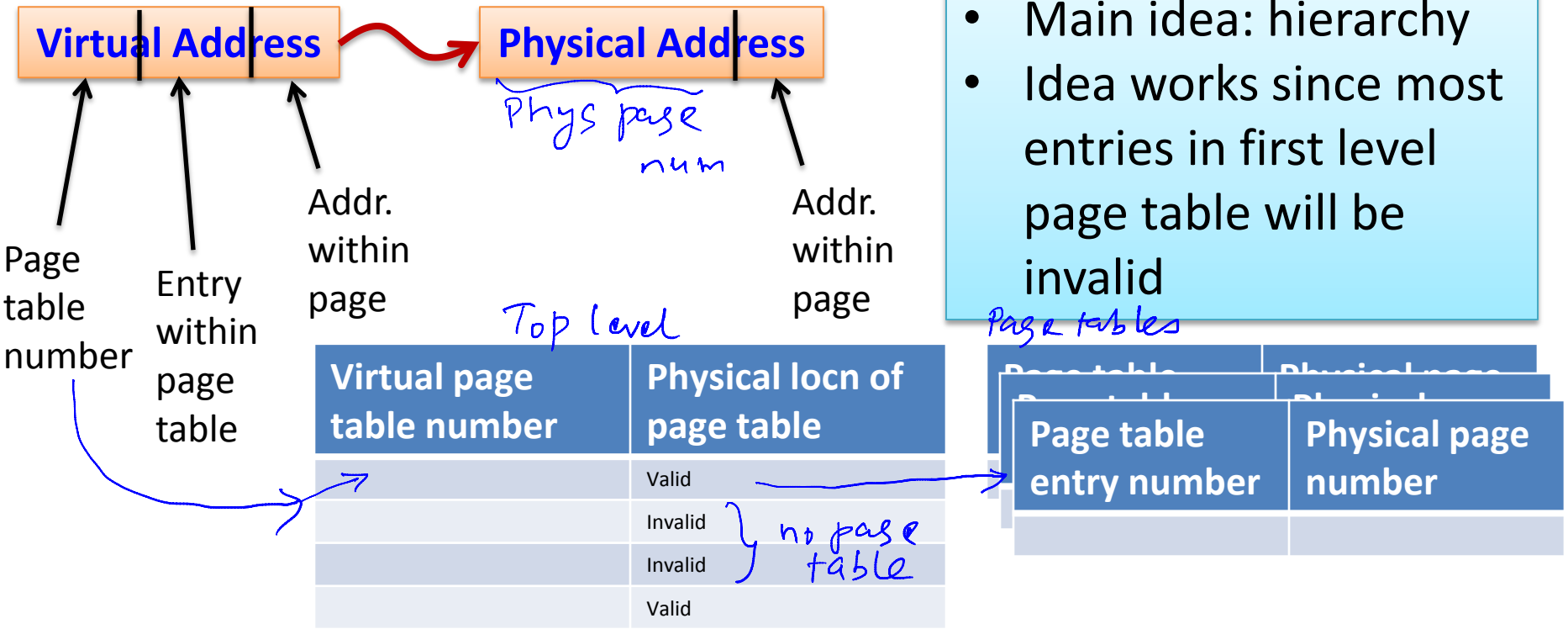
Maintain mapping only for usable parts of virtual memory

Dealing with Large Mapping Size: Solution-2: Inverted Page Table

Virtual page number	Physical page number

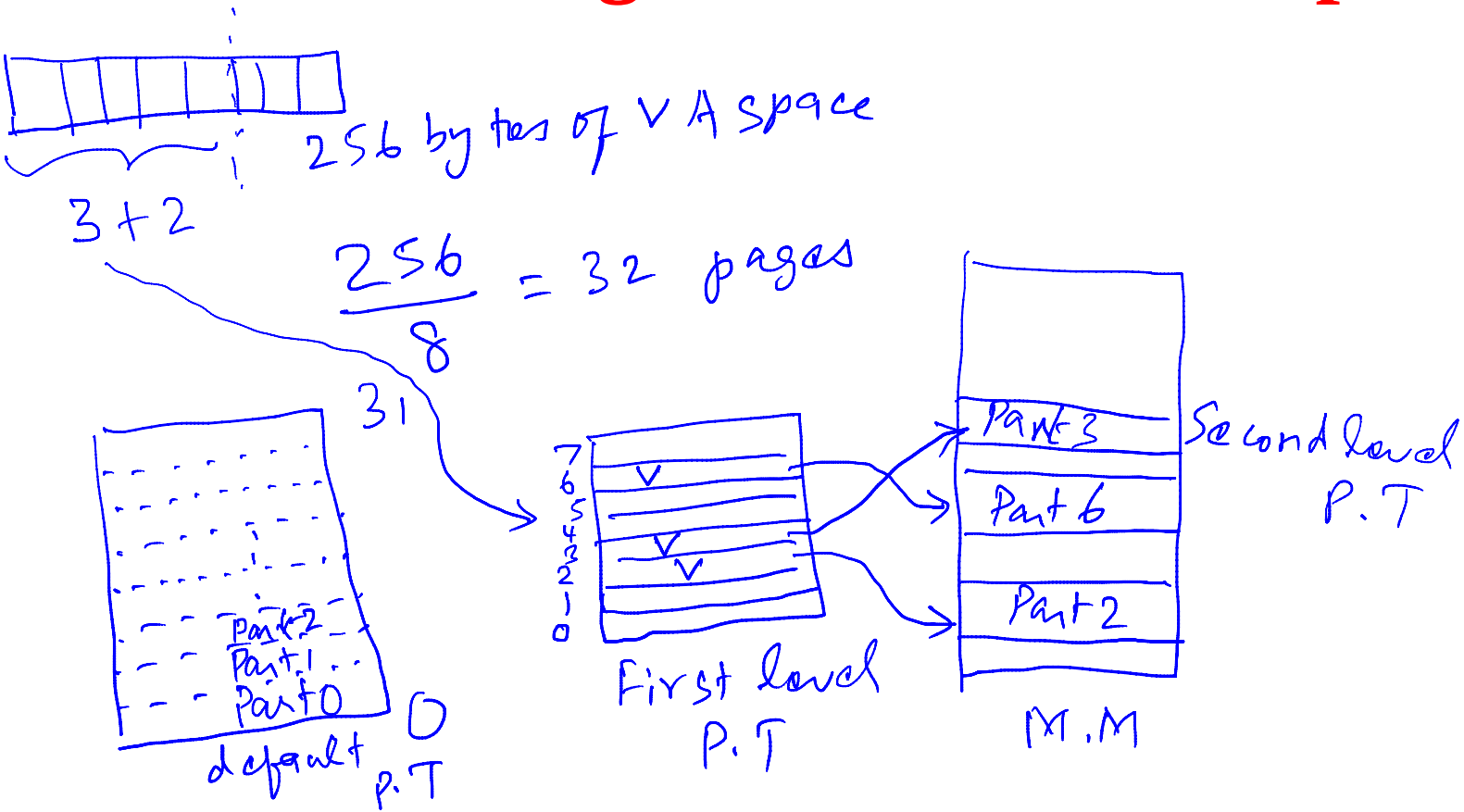
- One entry per physical page
- Entries hashed on virtual page number

Dealing with Large Mapping Size: Solution-3: Multi-level Page Table



- Main idea: hierarchy
- Idea works since most entries in first level page table will be invalid

Multi-Level Page Table: An Example

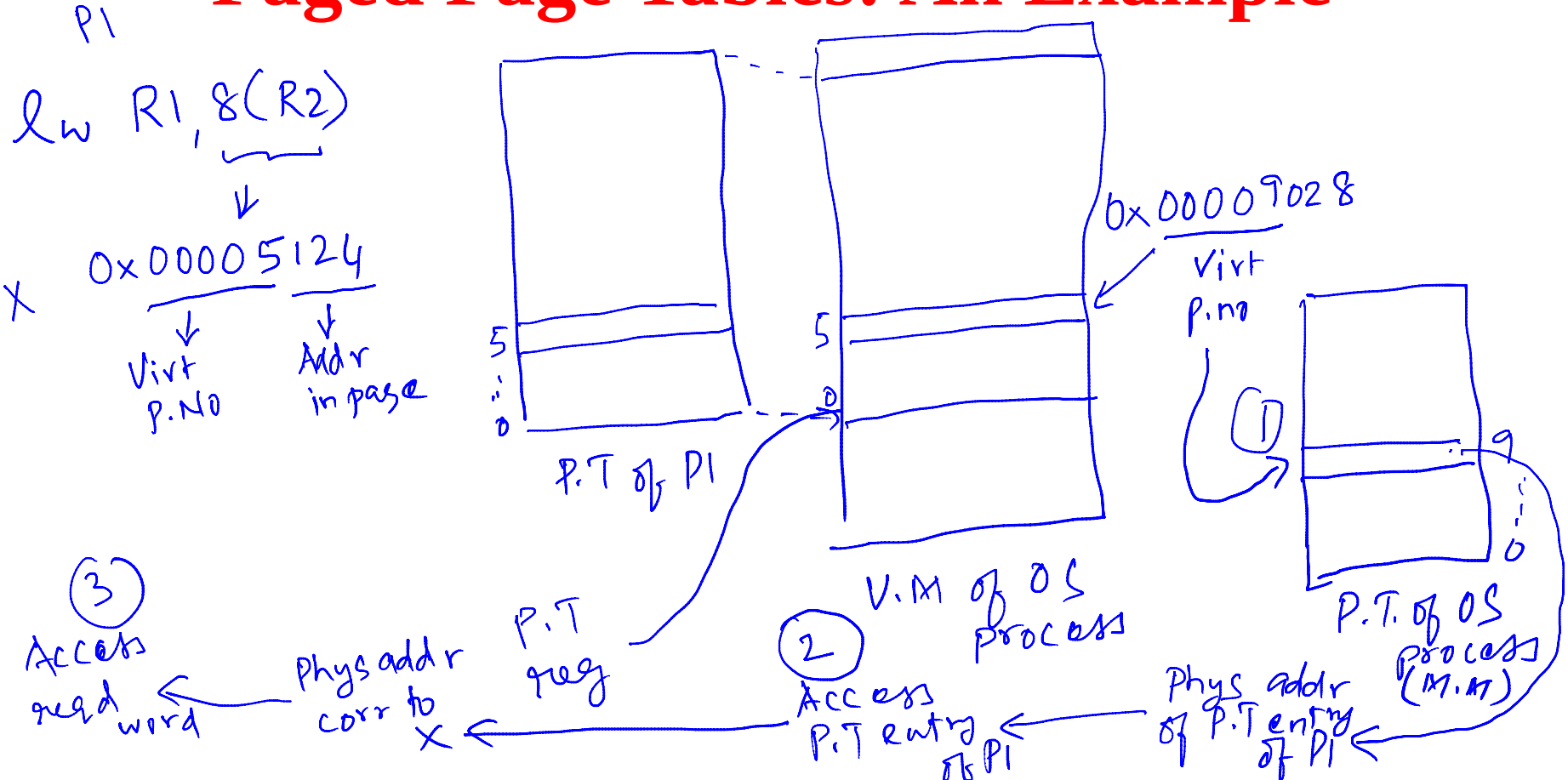


Dealing with Large Mapping Size: Solution-4: Paged Page Tables

Virtual page number	Physical page number

- Main ideas: level of indirection, caching
- Earlier: page table was in physical memory
- Now: page table in virtual memory (of a special OS process)
- Can manage even if page table size is large
- Page table of OS process pinned to physical memory

Paged Page Tables: An Example



Dealing with Large Mapping Size: Idea Behind the Solutions

- Page table limits: works since most of the page table is empty
- Inverted page table: works since most of the page table is empty
- Multi-level page table: works since most of the page table is empty
- Paged page table: works since there is locality of reference in virtual page references

Summary

- Virtual memory: VA-to-PA mapping is a level of indirection
  - Provides immense flexibility
- Segments: unequal sized blocks
- Managing the mapping: page table
- Ideas to deal with mapping size: page table limits, inverted page table, multi-level page table, paged page table
- Next: putting together VM, caches