CS 477:

Advanced Operating Systems

Virtual Memory & Super Pages

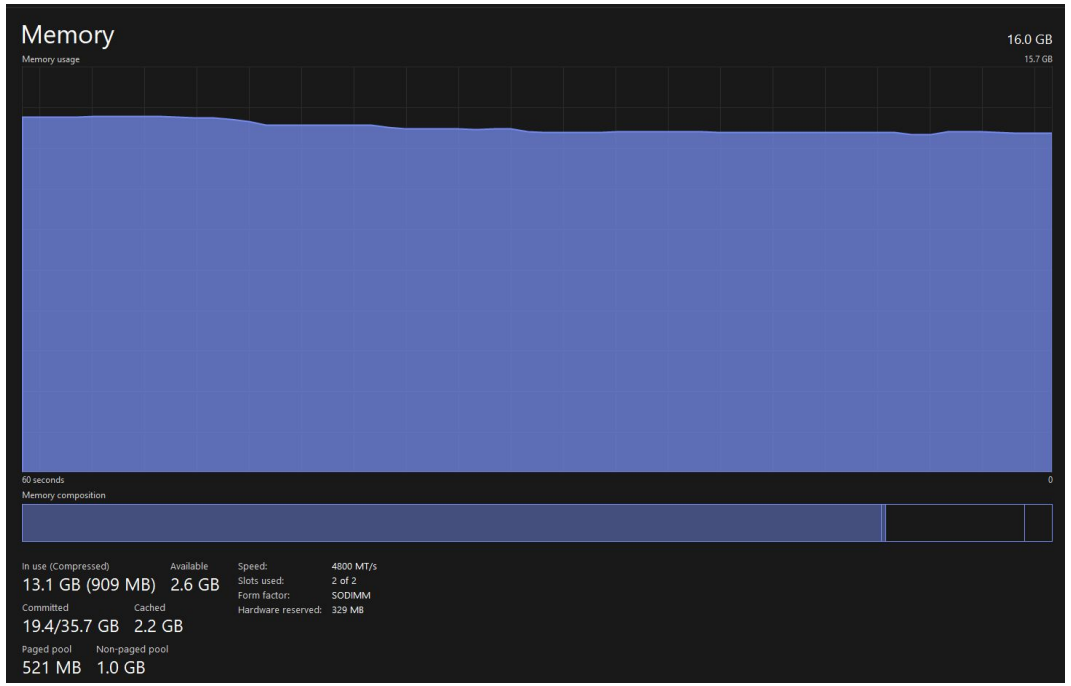
Administrivia

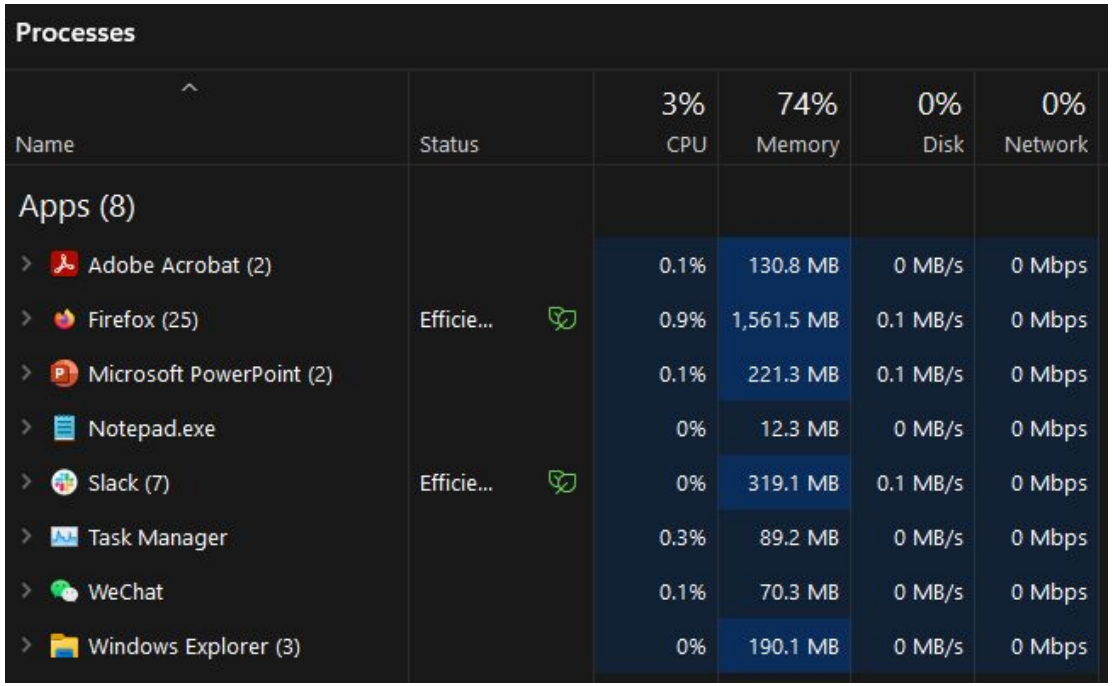
This week

- Virtual Memory and Paging
- TLB (Translation Lookaisde Buffer)
- Super Pages

Focus of today's lecture: Virtual Memory …

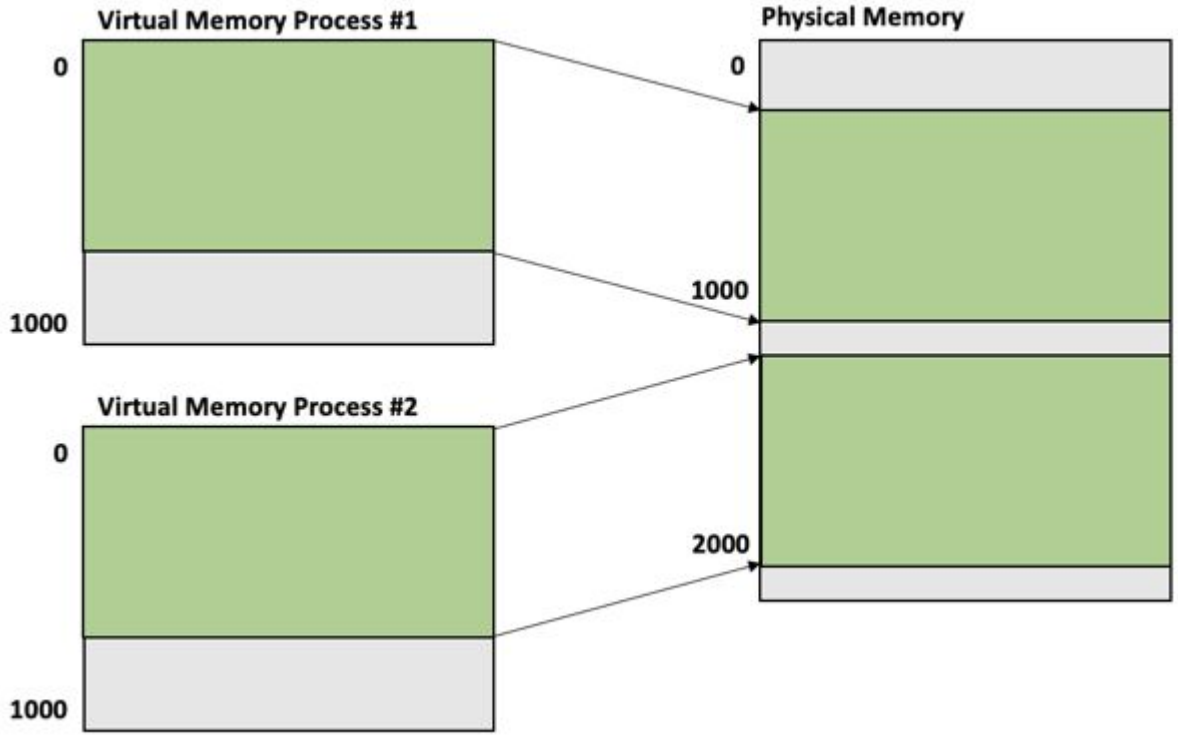
Applications share the physical memory as if they are the only owner





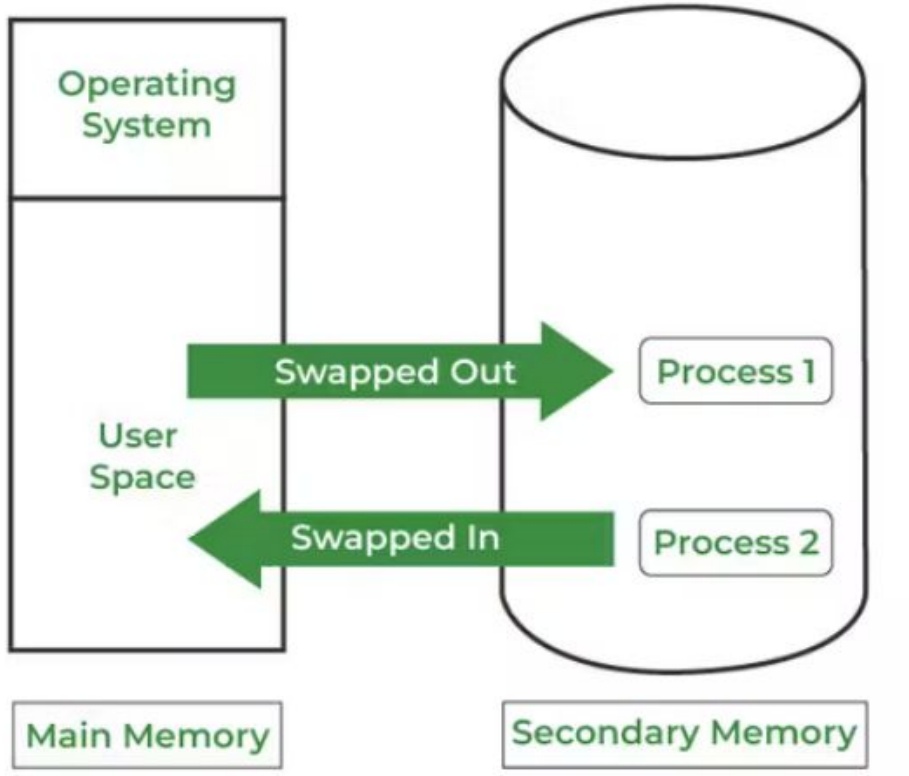
Why Virtual Memory

- Provide isolation and protection across processes
  - Each process has its own address space for the physical memory
  - Each process cannot directly access memory via physical address



Why Virtual Memory

- Enlarge the physical memory capacity with external storage in a transparent way
  - OS swaps unused data from physical memory to external storage (e.g., disk or remote memory)

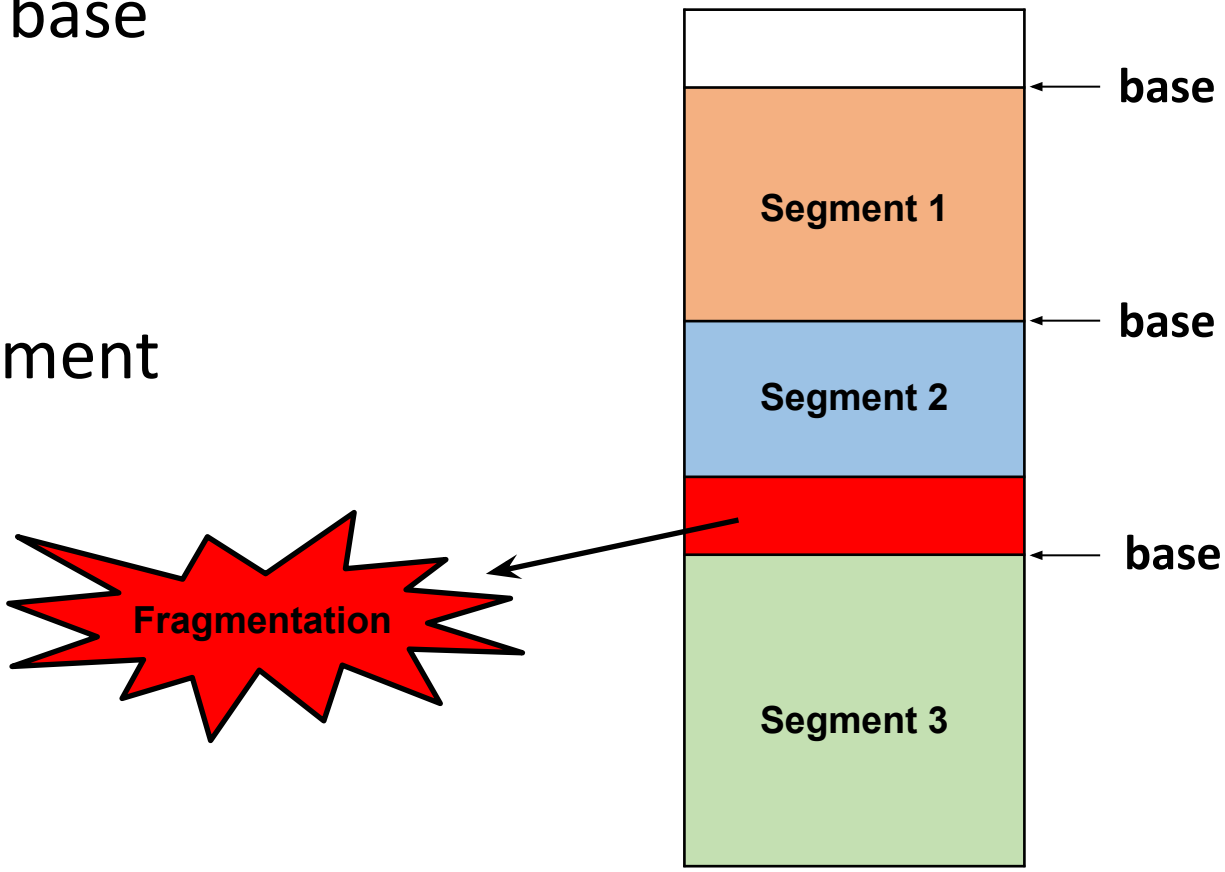


Virtual Memory – Hardware Mechanism

- Modern processors provide two mechanisms to separate virtual memory space from physical memory
  - **Segmentation:** Split physical memory into varied sized segments, and each processes can only access memory within its segments
  - **Paging:** Split physical memory into fixed size frames, and dynamically mapping physical page frame to virtual pages within each process

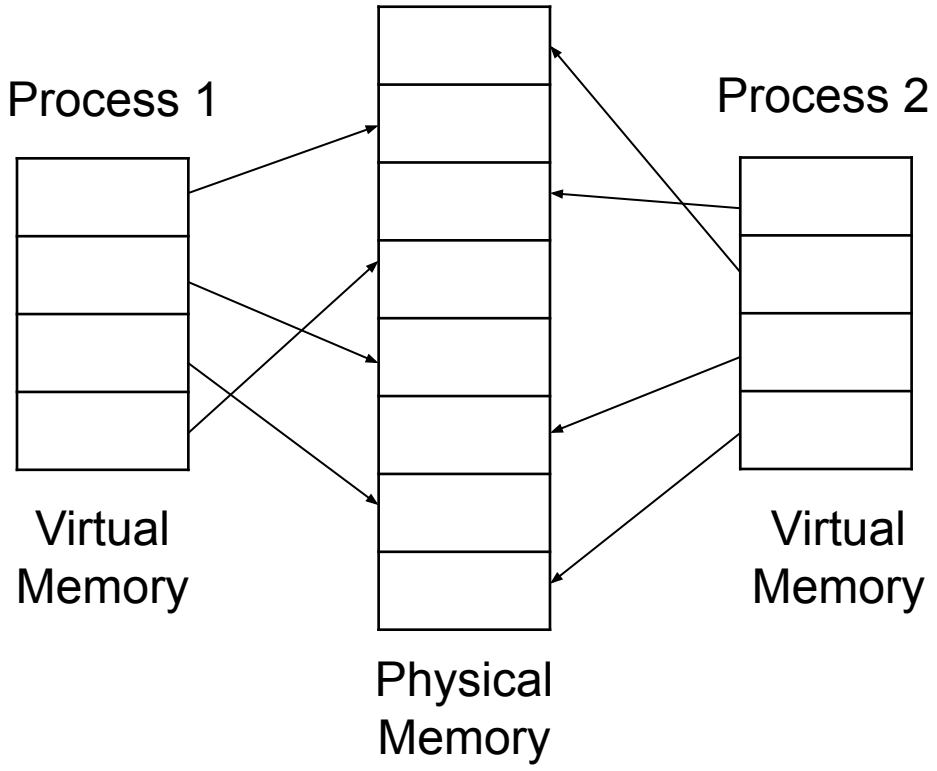
Segmentation – Hardware Mechanism

- Each Segment specifies a base address and size
- The memory within a segment needs to be contiguous
- Suffer from external fragmentation over time



Paging – Hardware Mechanism

- Physical memories are partitioned into chunks of fixed sizes (Physical Page Frames)
- Eliminates the need for contiguous physical memory
- Paging is used more often than segmentation in modern OSes



Paging – Hardware Mechanism

- OS maps applications' Virtual pages to the actual physical page frames by maintaining data structure called **Page Table**

- Simplest model: ***Array***

  Virtual Page number (VPN) as index

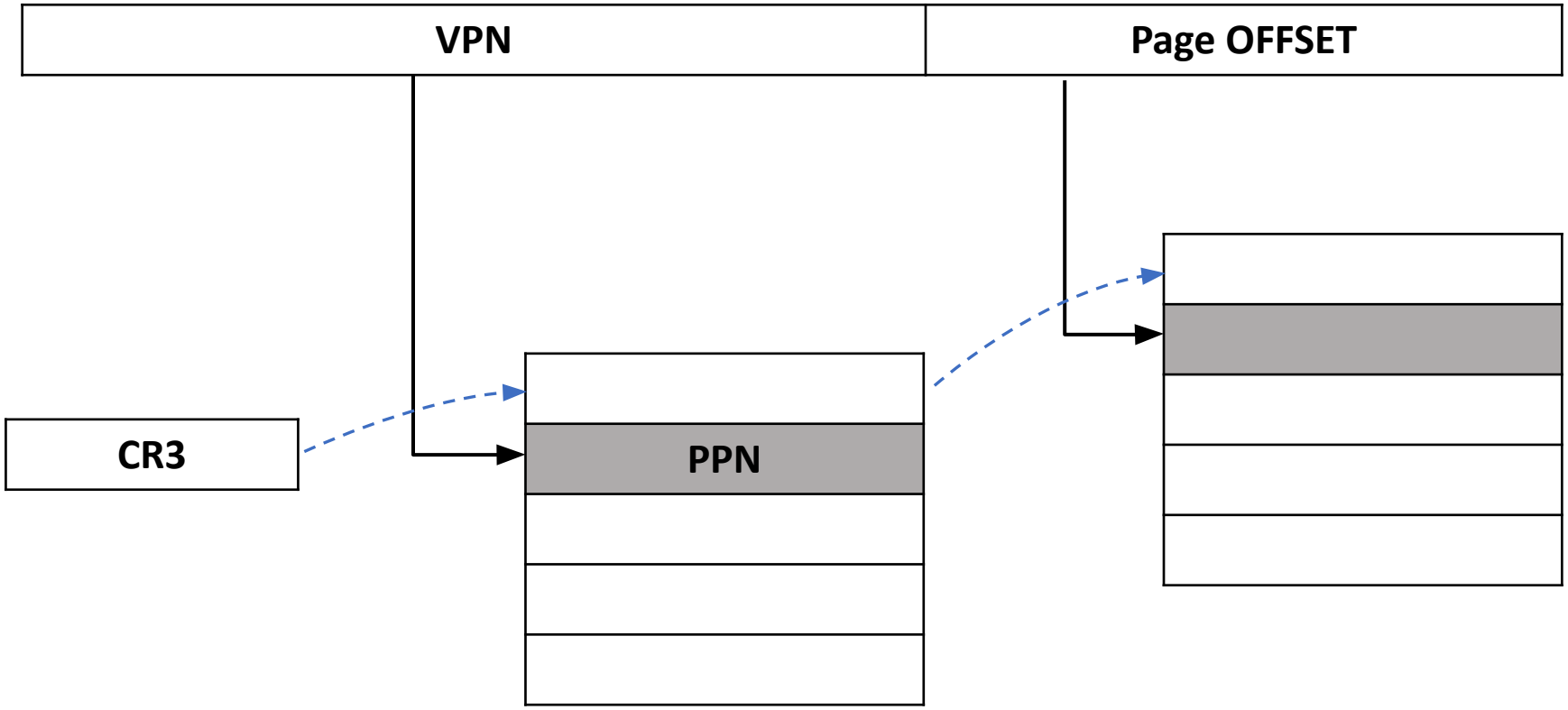
  Physical Page number (PPN) as value

- VPN -> PPN translation

VPN (index)	PPN	Valid?
0	1501	1
1	12	1
2	434	1
3	51	1
4	52	1
	...	
1048574	63463	0
1048575	376	1

Paging – Address Translation

LD [VA], R1



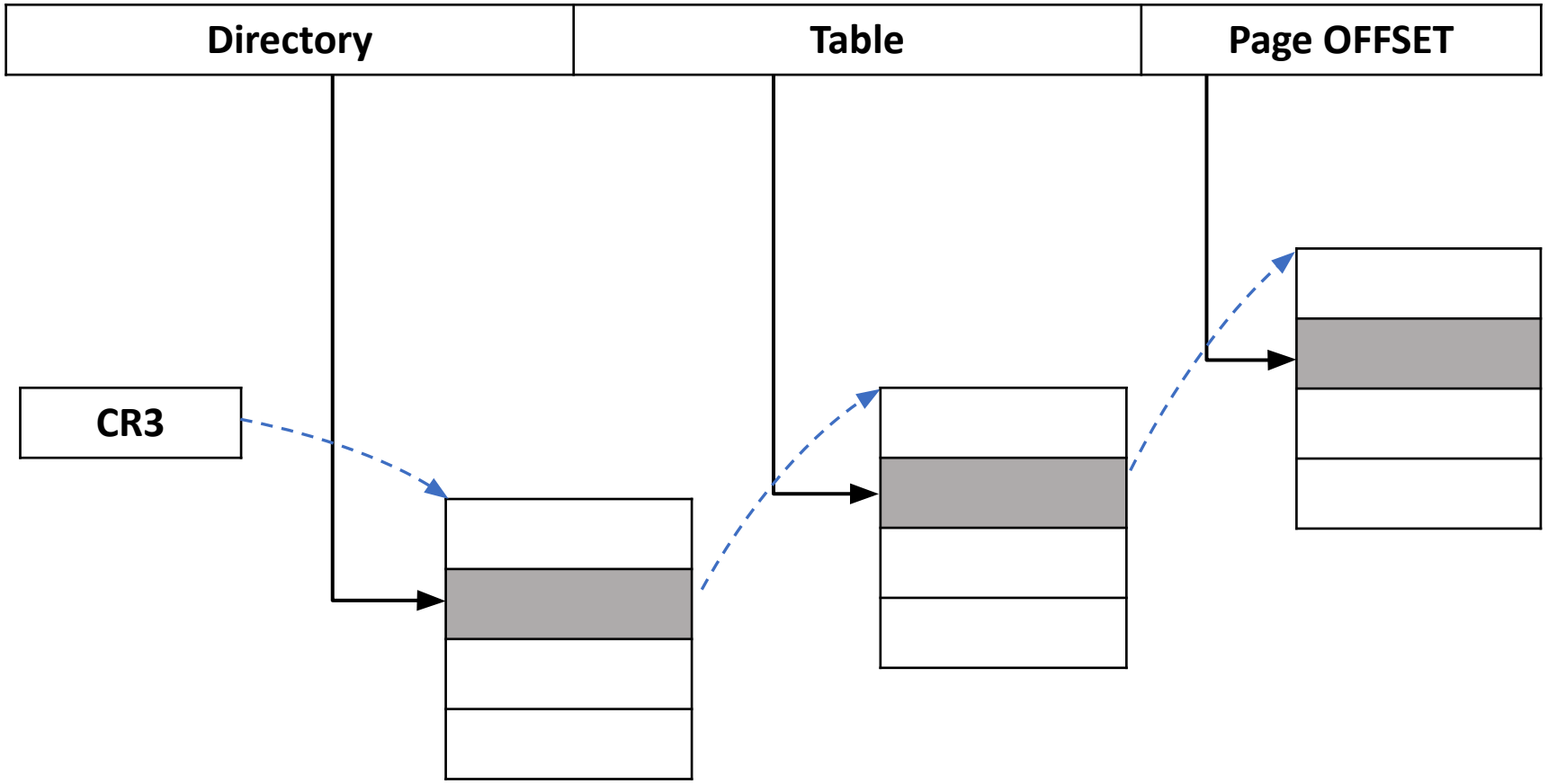
Paging – Limitation and Overhead

- Page Table size are fixed even a process uses very little memory
  - 32-bit system with 4KB page size,
  - We need $2^{32}/2^{12} = 2^{20} = 1048576$ entries for page table
  - 4MB required for page table only for each process

VPN (index)	PPN	Valid?
0	1501	1
1	12	1
2	434	1
3	51	1
4	52	1
…		
1048574	63463	0
1048575	376	1

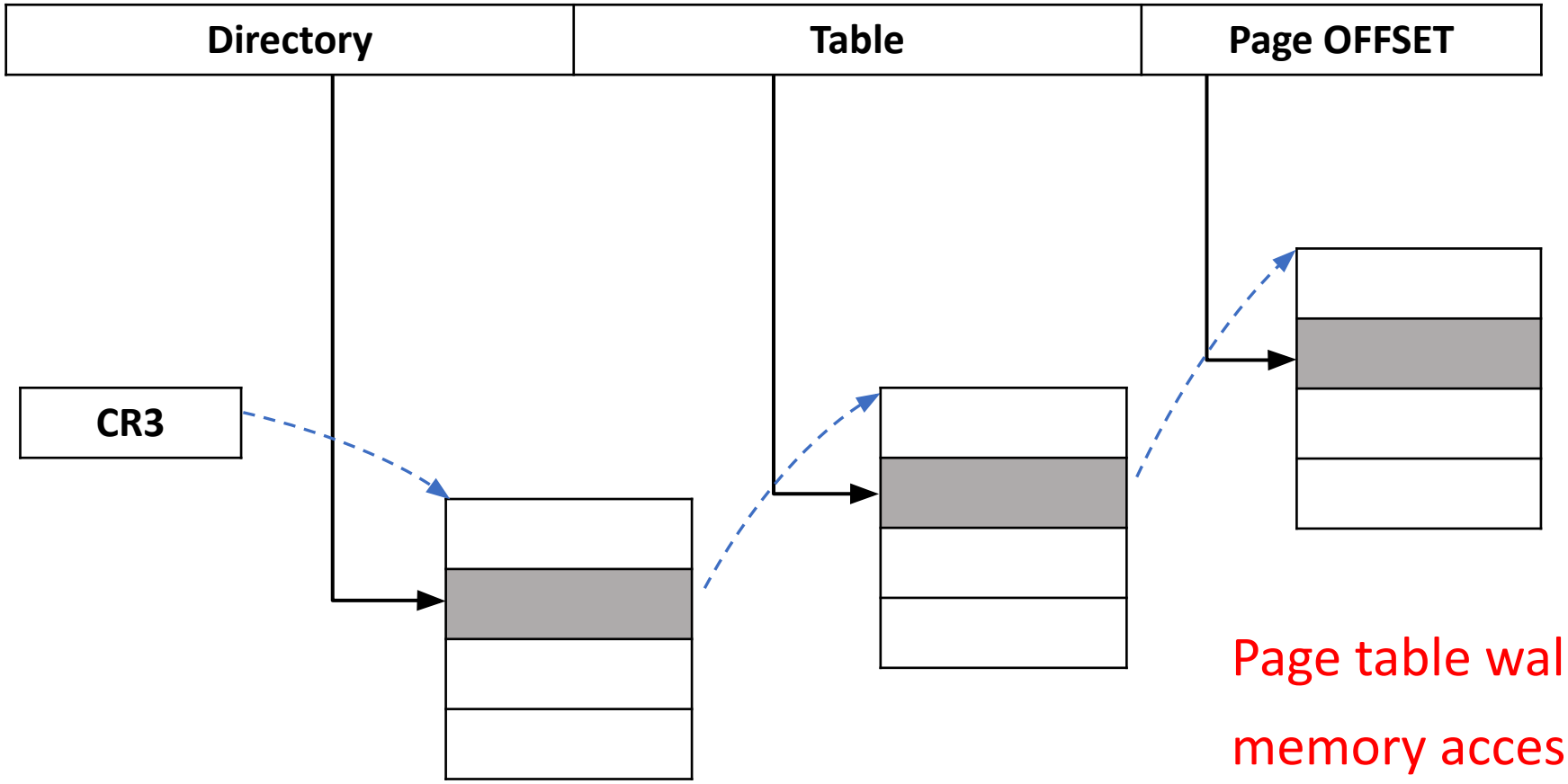
Paging – Multi-level Page Table

LD [VA], R1



Paging – Multi-level Page Table

LD [VA], R1

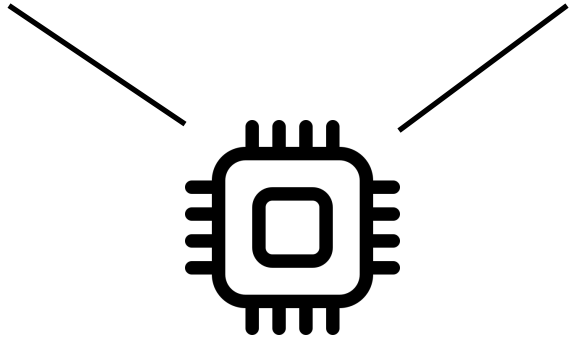


Page table walk needs multiple memory access for a load/store

Paging – TLB (Translation Lookaside Buffer)

- A hardware cache for page table entries
- VPN -> PPN translation
- Each CPU core has its own TLB
- OS maintain the TLB coherence

VPN	PPN
202	1501
76	12
5	434
67	51
13	52

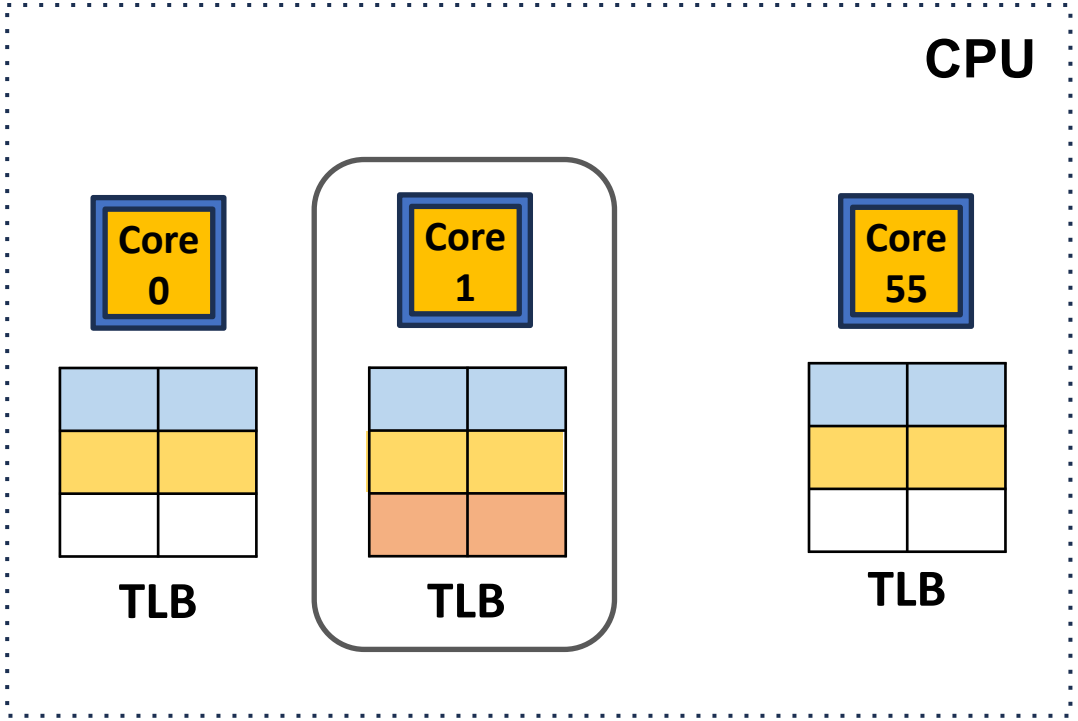


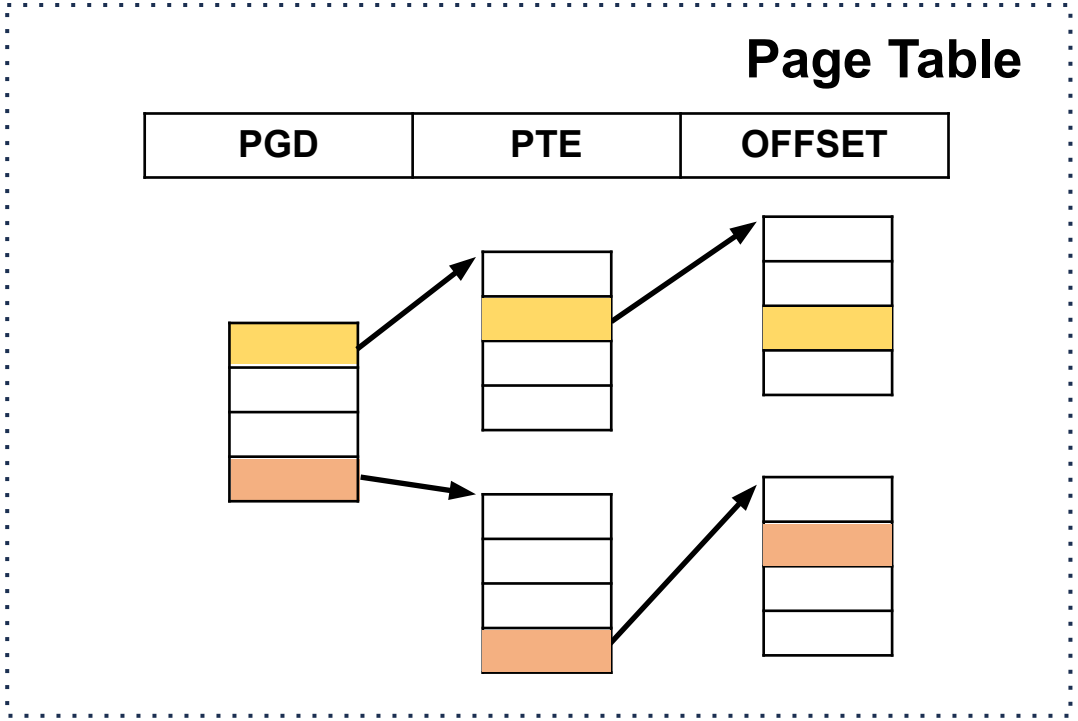
Paging –Example with TLB

LD [VA0], R1

ST [VA1], R2

LD [VA2], R3





Paging – TLB Coherence

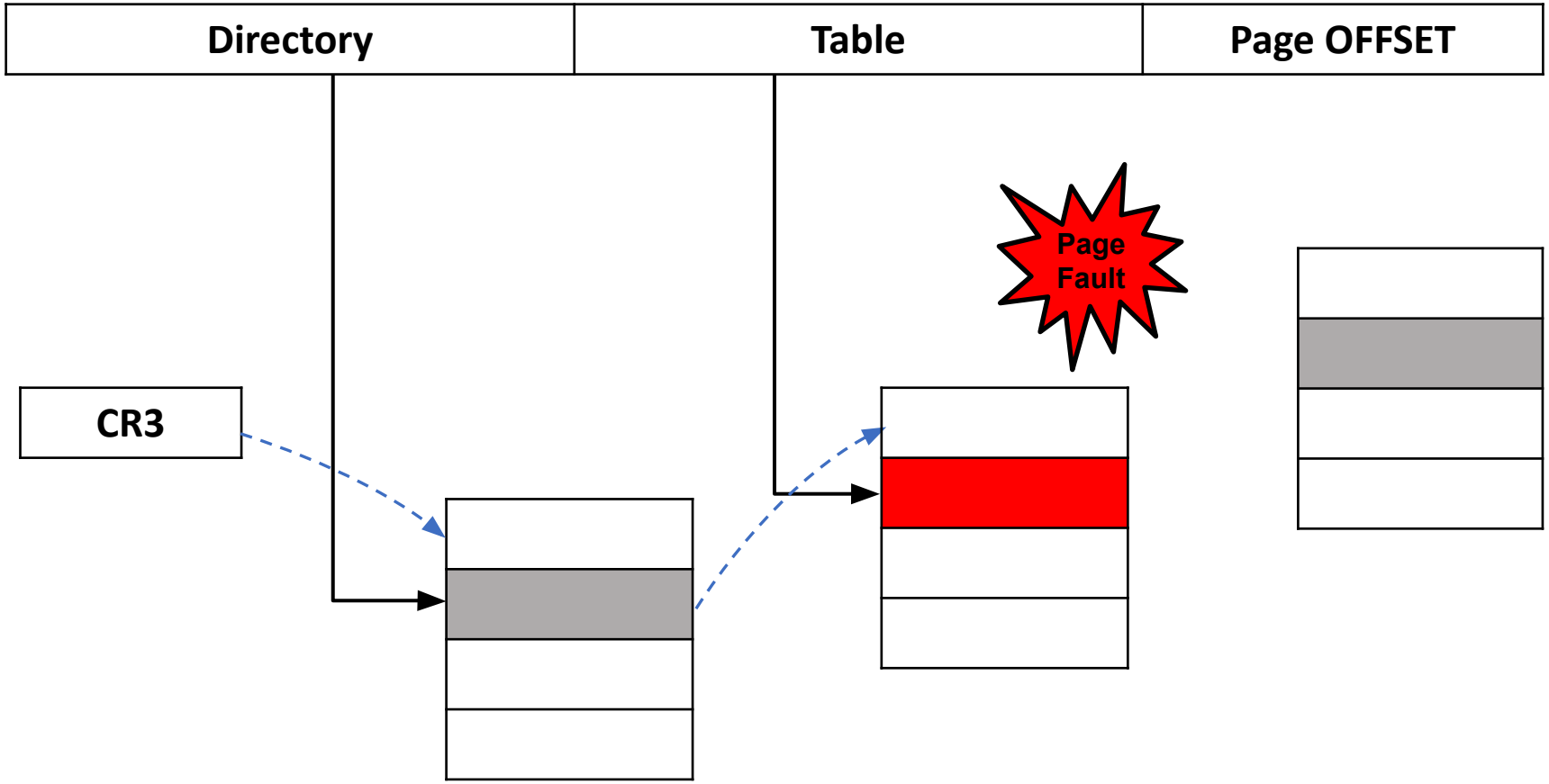
- OS maintain the TLB coherence across CPU cores

- TLB Flush
  - Clear TLB entries of its own CPU core

- TLB Shootdown
  - Clear TLB entries of other CPU cores who have the same mapping
  - Requires expensive point to point IPI operations to deliver TLB shootdown

Page Fault

- No corresponding PPN for a VPN



Page Fault

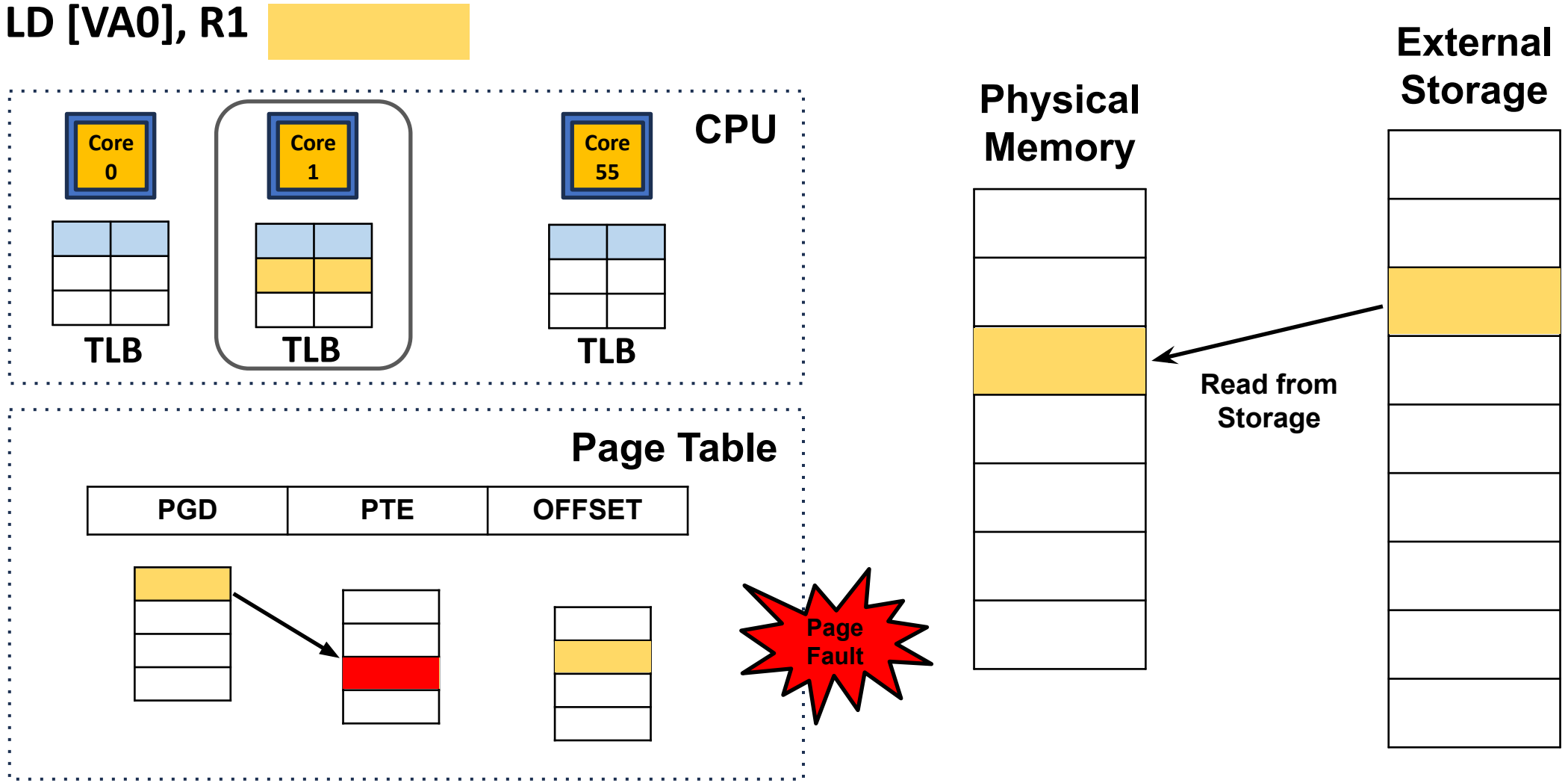
- Page Fault: no corresponding page table entry for a virtual address
- During load/store instruction when MMU walking the page table
- A fault signal was raised, and the control is taken over from user space process to the OS to handle the page fault

Major Page Fault

- Major Page Faults

  - The data was previously swapped to external memory (e.g., Storage)
  - Need slow I/O operations to swap the data to DRAM
  - Very slow because it involves I/O

Paging –Example with Major Page Fault

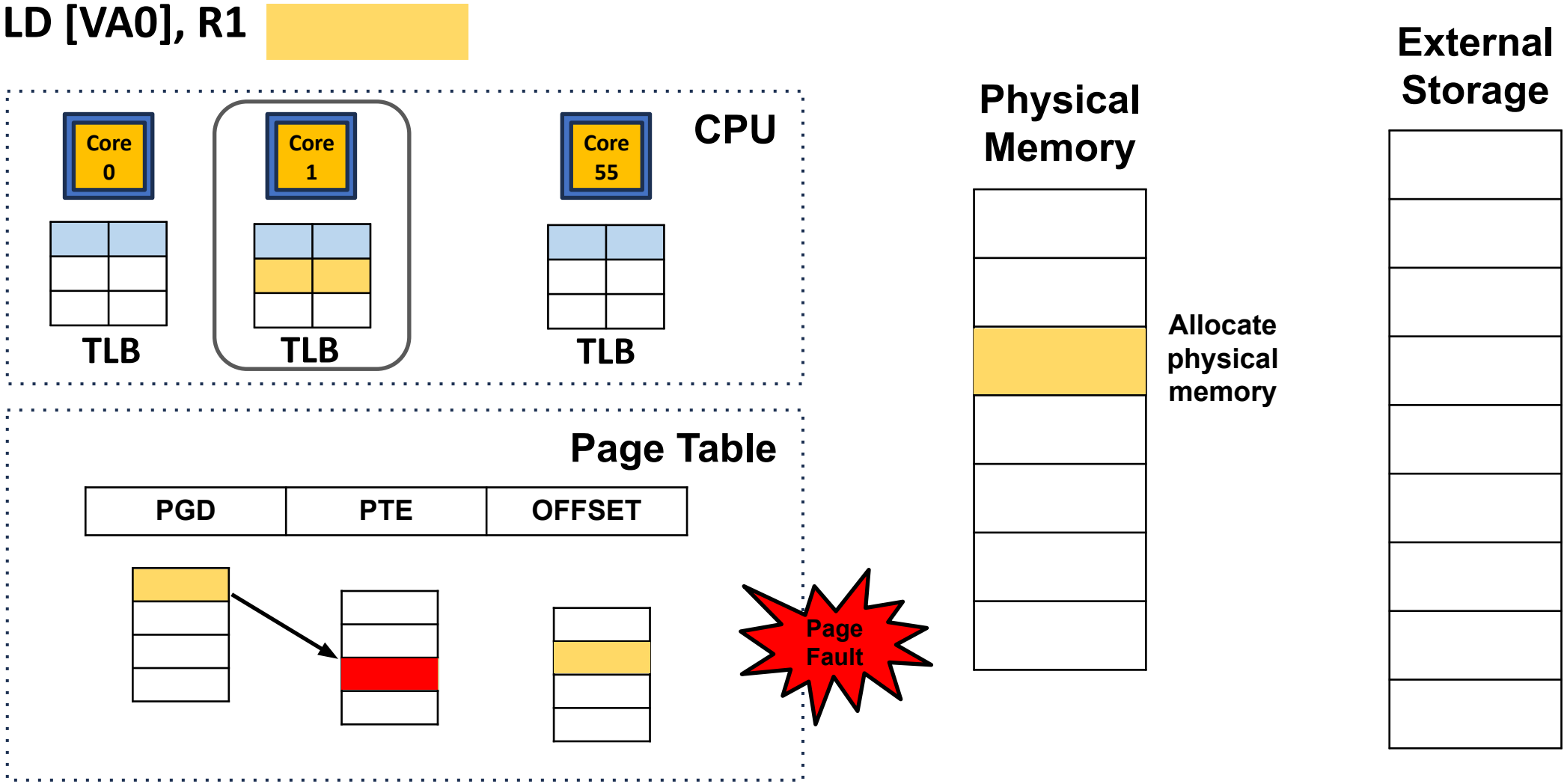


Minor Page Fault

- Minor Page Faults
  - No mapping in page table
  - No need to load data from external storage
- **Case 1:** data is in DRAM, but no mapping
  - Shared Memory (e.g., shared libraries)
  - Copy-on-Write (e.g., fork)
- **Case 2:** mapping is created, but physical memory is not allocated
  - Deferred allocation in mmap() with ANONYMOUS flag

Paging –Example with Minor Page Fault

LD [VA0], R1



Trade-offs of Page Sizes

- Smaller page size
  - Less internal fragmentation
  - Overhead in managing pages with larger page table sizes
  - Low TLB coverage

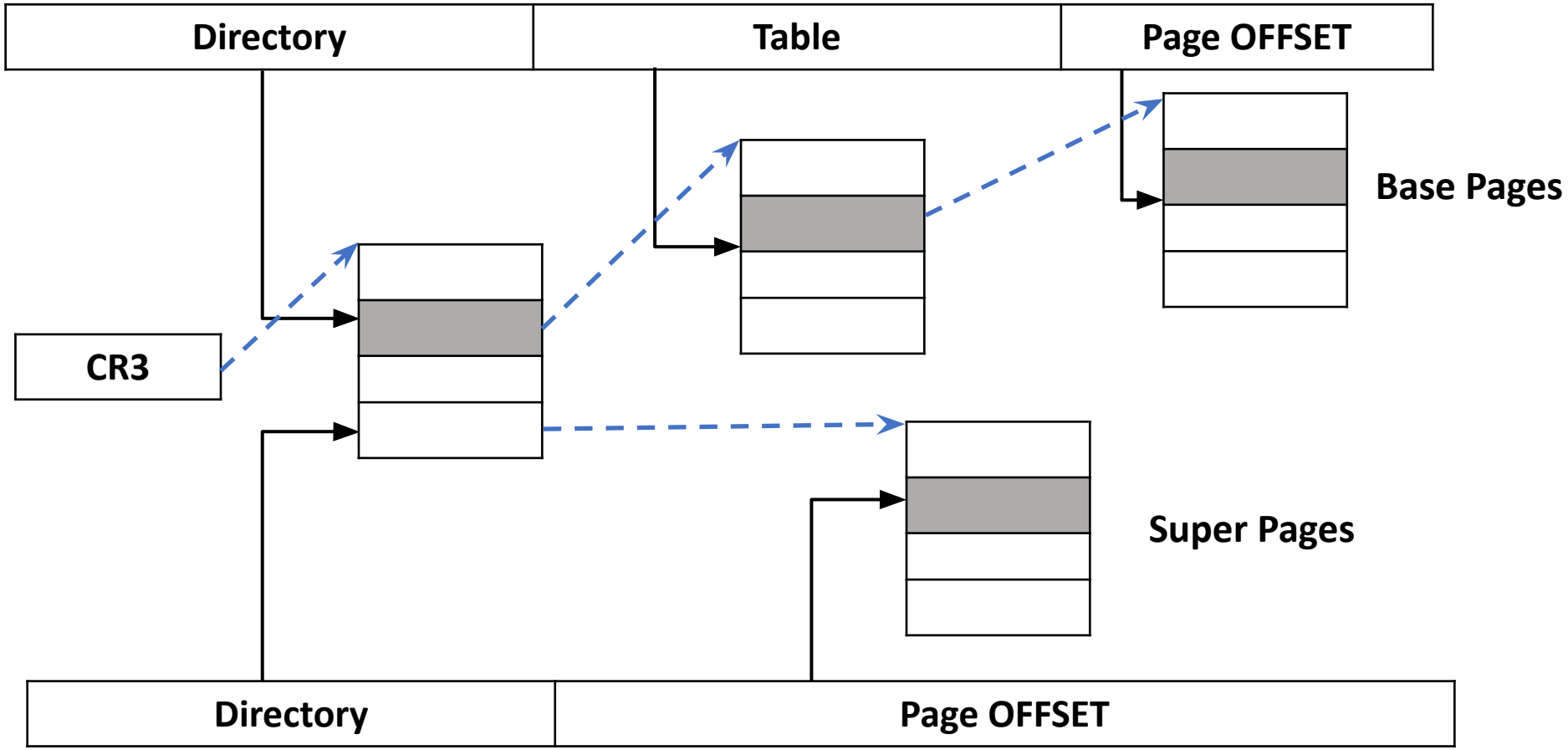
- Larger page size
  - more internal fragmentation
  - smaller page table sizes
  - High TLB coverage

TLB with 512 entries:

- cover 2MB physical memory with 4KB page
- cover 1GB physical memory with 2MB page

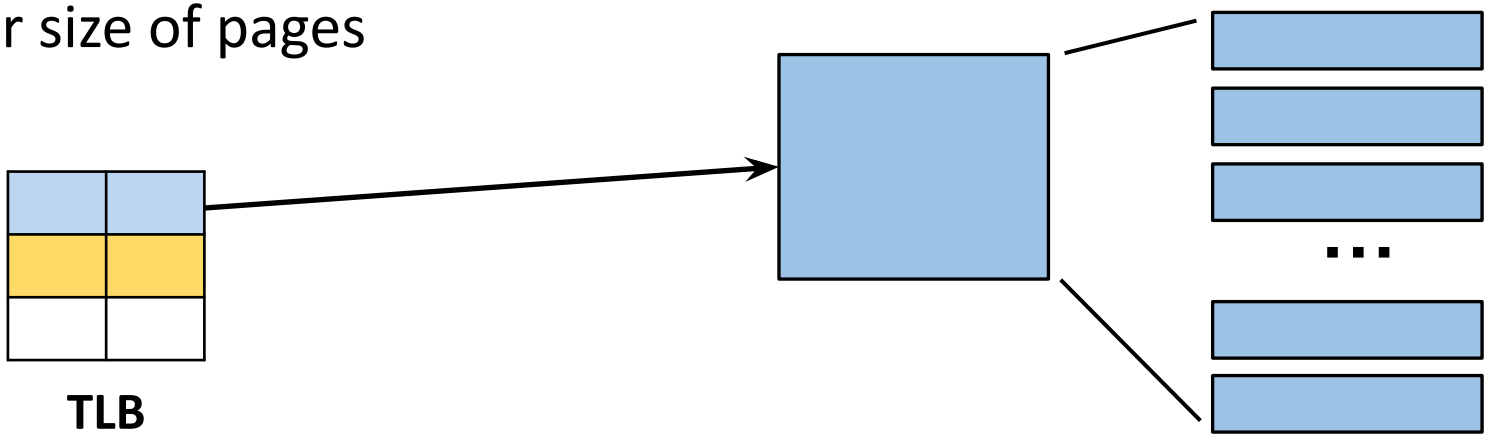
Superpage – The art of balancing

- Pages with varied sizes



Superpage – The art of balancing

- Goal 1: Increase TLB coverage
  - Larger size of pages



- Goal 2: Reduce fragmentation
  - Break large pages into smaller one if necessary

Superpage – Hardware Constraints

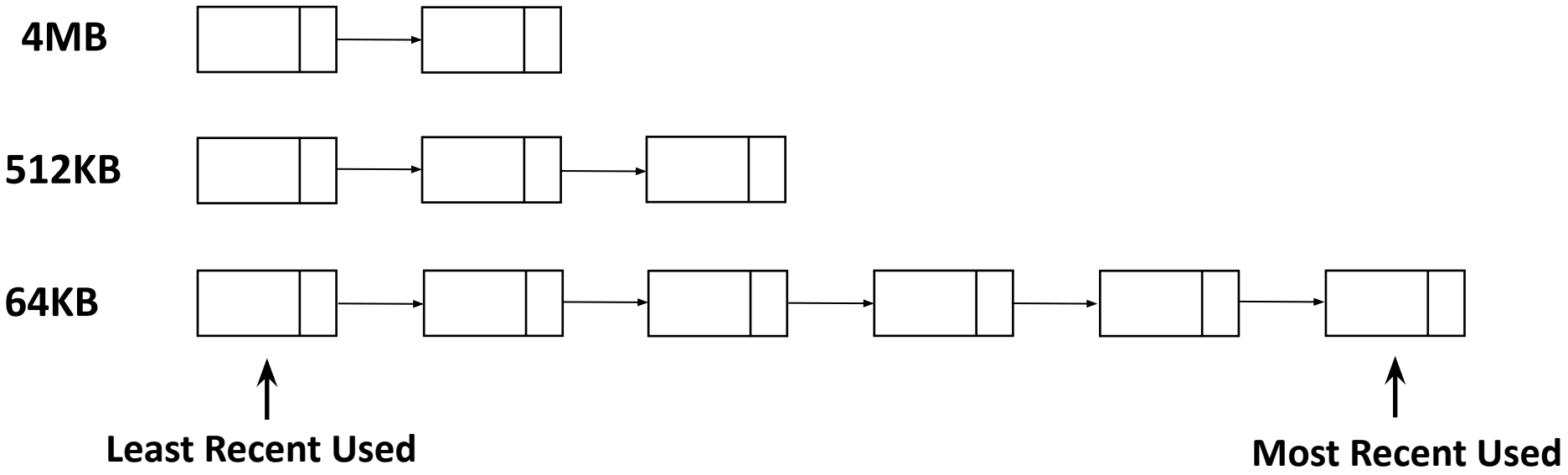
- **Constraint 1:** Only several options are available for page sizes
- **Constraint 2:** A superpage is required to be contiguous both virtually and physically
- **Constraint 3:** All base pages share the same attributes (protection, dirty bits) within a super page in the page table and TLB

Superpage – Reservation in Allocation

- Allocating a superpage instead of a base page
- All consequent allocations falling into the range covered by this superpage do not need to consume a dedicated page table entry
- Improves TLB coverage by extending the size of each page

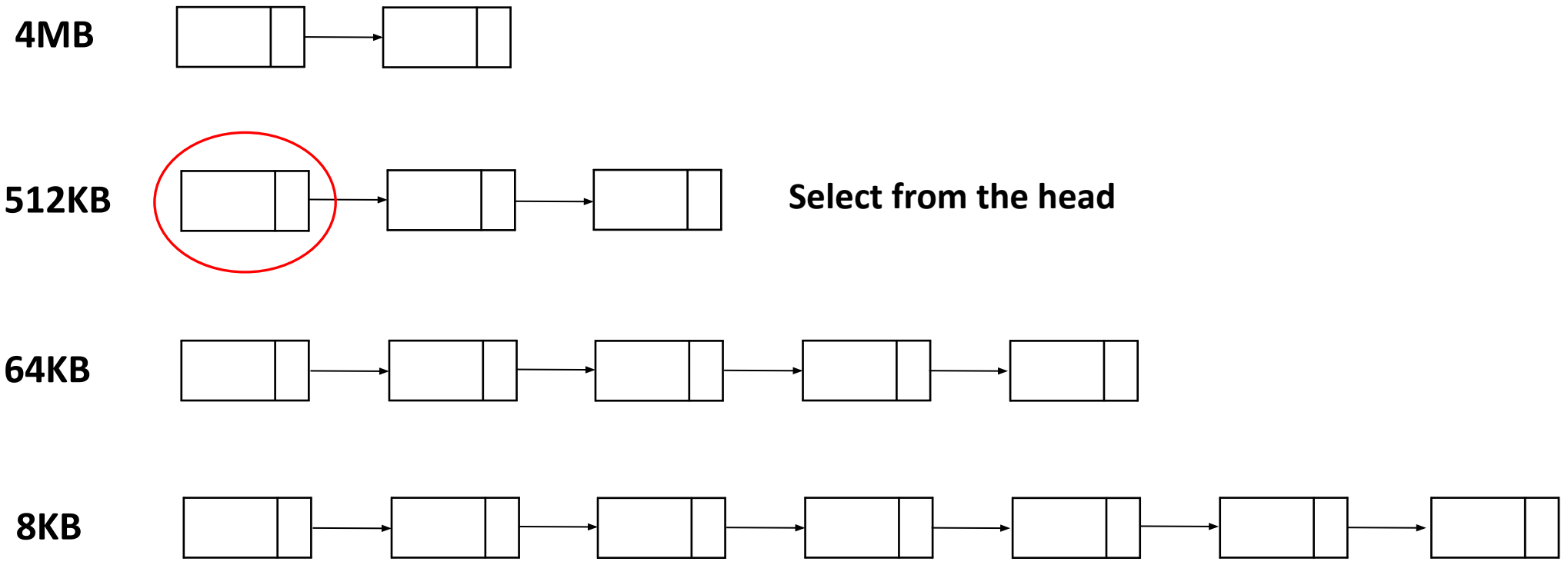
Superpage – Reservation List

- Tracking the reserved physical pages
- One reservation list for each page size
- Reservations in each list are sorted in LRU order



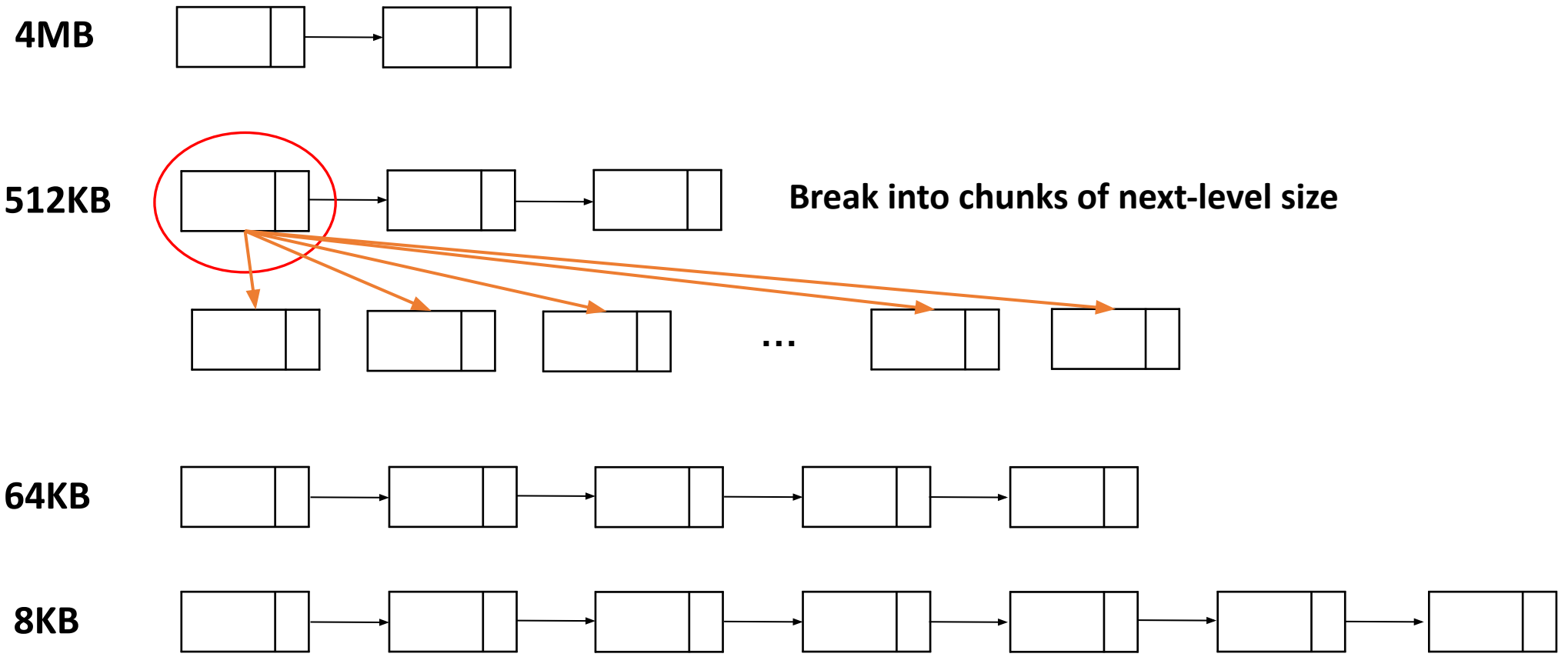
Superpage – Reservation List

- Preempting a reservation



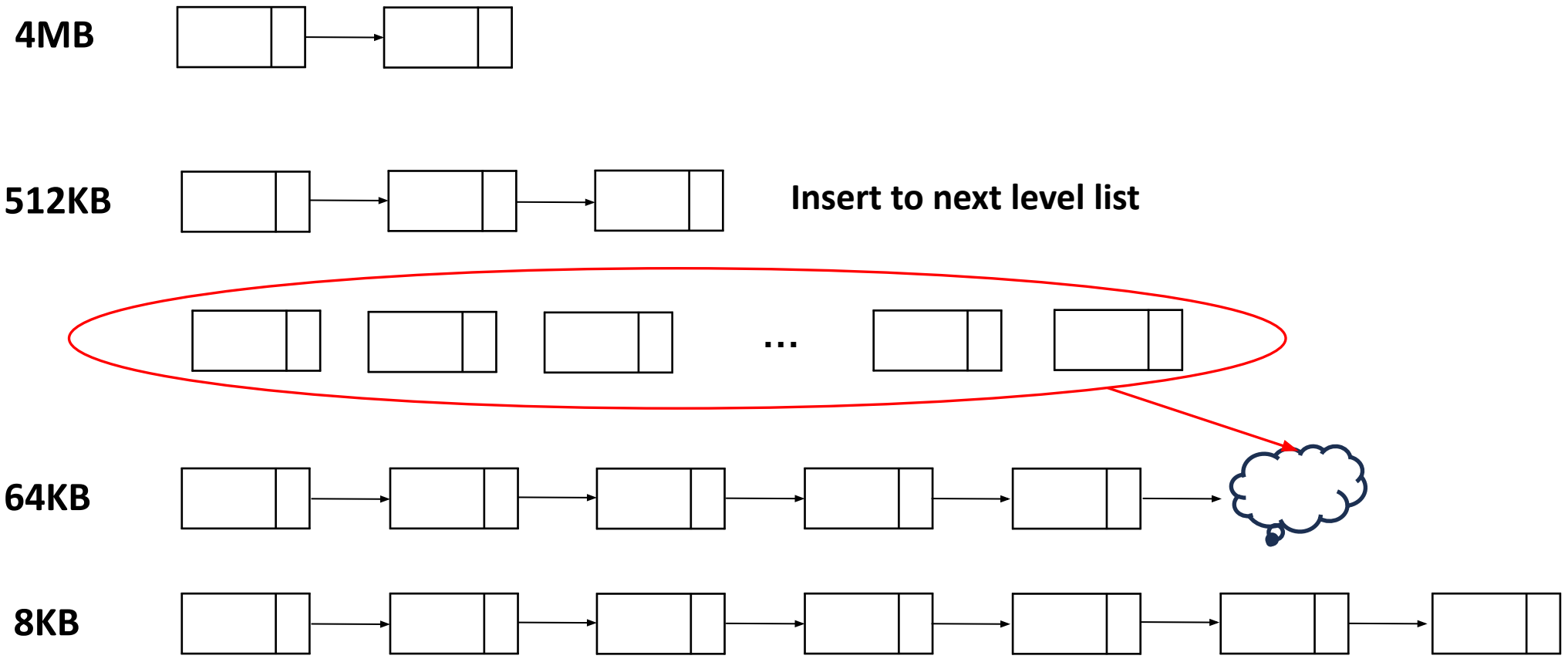
Superpage – Reservation List

- Preempting a reservation



Superpage – Reservation List

- Preempting a reservation



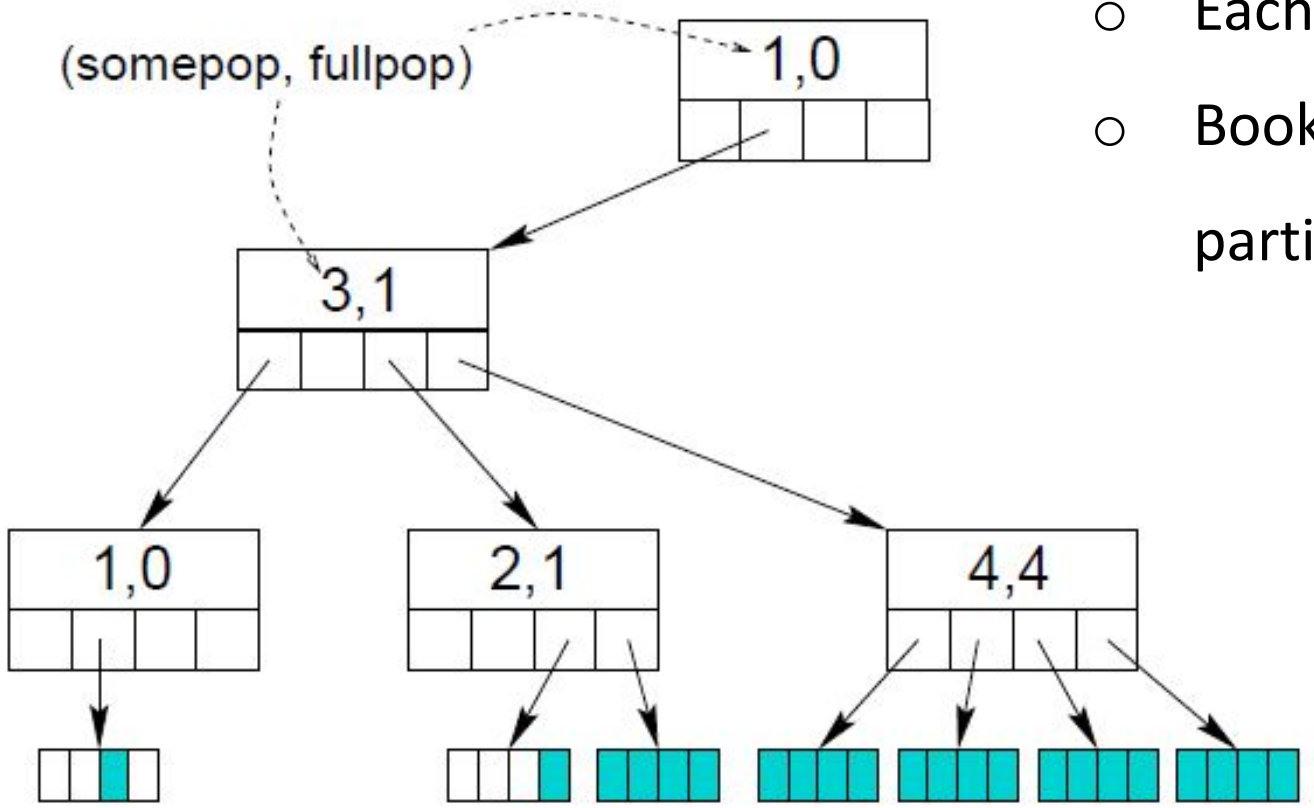
Superpage – Population Map

- Tracking the base page usage within each memory object
  - How many base pages are used
  - Helps promotion decision

- The population map is queried for a page fault
  - <memory object id, offset in object> to locate the population map
  - Round down the faulting address to align with the largest page size

Superpage – Population Map

- Radix Tree Data Structure



  - Each level is corresponding to a size
  - Bookkeeps number of children

    partially populated or fully populated

Superpage – Determine the Page Size

- Static approach for fixed-size memory objects
  - Code segments, memory-mapped files are of large and fixed sizes
  - Allocate the largest, aligned superpages for them

- Dynamic approach for varying-size memory objects
  - Dynamically allocated memory (on stack or heap)
  - Allocate the smallest size of super page to hold this object

Superpage – Fragmentation Control

- Preempting existing reservations
  - Preempting a reservation if the memory are not used within a superpage
- Buddy allocator
  - Coalescing available memory regions whenever possible
- Page replacement Daemon
  - Contiguity-aware page replacement

Superpage – Page Promotion

- Incremental Promotion
  - Promotion occurs to the smallest superpage size next to the current one
- When to promote
  - The population count within a superpage reaches a certain fraction of the size

Superpage – Page Demotion

- **Case 1:** During page replacement (eviction)
  - A superpage is demoted when a base page within it is selected as victim
- **Case 2:** Attributes of a base page changes (e.g., protection)
  - mprotect system call
- **Case 3:** Periodically performed in background
  - To determine if the superpage is still being actively used

Superpage – Swapping

- No individual dirty information for each base page
  - High I/O cost for swapping if only part of the superpage is dirty
- Tracking the dirty information while reserving the superpage
  - Compute the hashing value of each base page content
  - Recompute the new hash value before evicting
  - If the old hash value and new one are same, it means the base page is not dirty. Hence no I/O is required.

Superpage – Evaluation

- Best-case benefits without fragmentation

Bench-mark	Superpage usage 8 KB	64 KB	512 KB	4 MB	Miss reduc (%)	Speed-up
Web	30623	5	143	1	16.67	1.019
Image	163	1	17	7	75.00	1.228
Povray	136	6	17	14	97.44	1.042
Linker	6317	12	29	7	85.71	1.326
C4	76	2	9	0	95.65	1.360
Tree	207	6	14	1	97.14	1.503
SP	151	103	15	0	99.55	1.193
FFTW	160	5	7	60	99.59	1.549
Matrix	198	12	5	3	99.47	7.546

Superpage – Evaluation

- Sensitivity to different superpage sizes

Benchmark	64KB	512KB	4MB	All
CFP2000	1.02	1.08	1.06	1.12
galgel	1.28	1.28	1.01	1.29
lucas	1.04	1.28	1.24	1.28
apsi	1.04	1.79	1.83	1.83
Image	1.19	1.19	1.16	1.23
Linker	1.16	1.26	1.19	1.32
C4	1.30	1.34	0.98	1.36
SP	1.19	1.17	0.98	1.19
FFTW	1.01	1.00	1.55	1.55
Matrix	3.83	7.17	6.86	7.54

Performance Speedups

Benchmark	64KB	512KB	4MB	All
CFP2000				
galgel	98.51	98.71	0.00	99.80
lucas	12.79	96.98	87.61	99.90
apsi	9.69	98.70	99.98	99.98
Image	50.00	50.00	50.00	75.00
Linker	57.14	85.71	57.14	85.71
C4	95.65	95.65	0.00	95.65
SP	99.11	93.75	0.00	99.55
FFTW	7.41	7.41	99.59	99.59
Matrix	90.43	99.47	99.47	99.47

TLB miss reductions

Superpage – Summary

- Varied sizes of page to improve TLB coverage
- Promote base pages to a larger one to adapt to dynamic workload
- Demote a larger page to smaller ones to reduce fragmentation
- Hashing base page contents to identify the fine-grained dirty information for each base page within a super page

THP in Linux

- THP (Transparent Huge Page) is a feature in Linux MM
  - Share some similarities with the idea of superpages
  - Smaller pages can be promoted to a larger page
  - Larger pages can be demoted to smaller pages

- Promotion and Demotion in an automatic manner
  - Promotion is based on access patterns (e.g., sequential access)
  - Demotion is based on the memory fragmentation situation