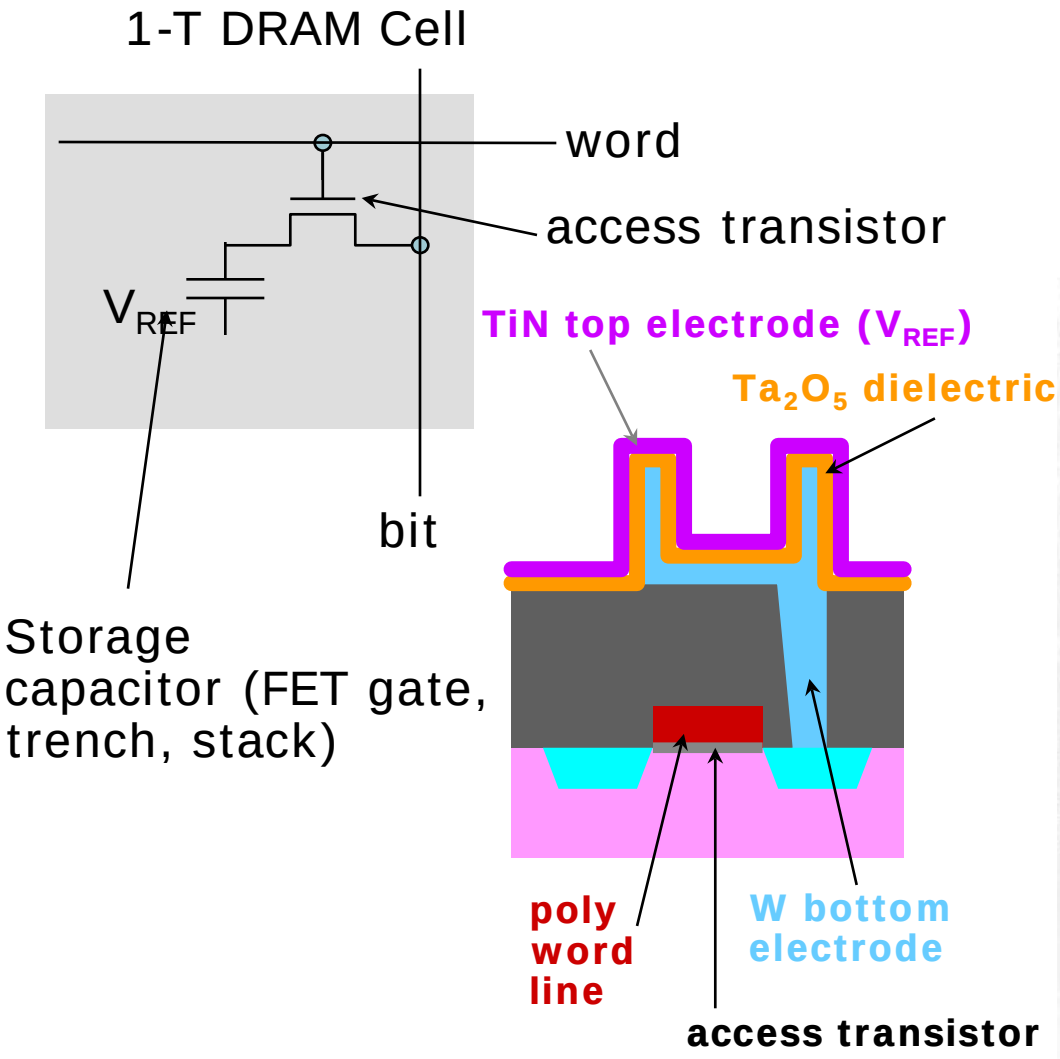
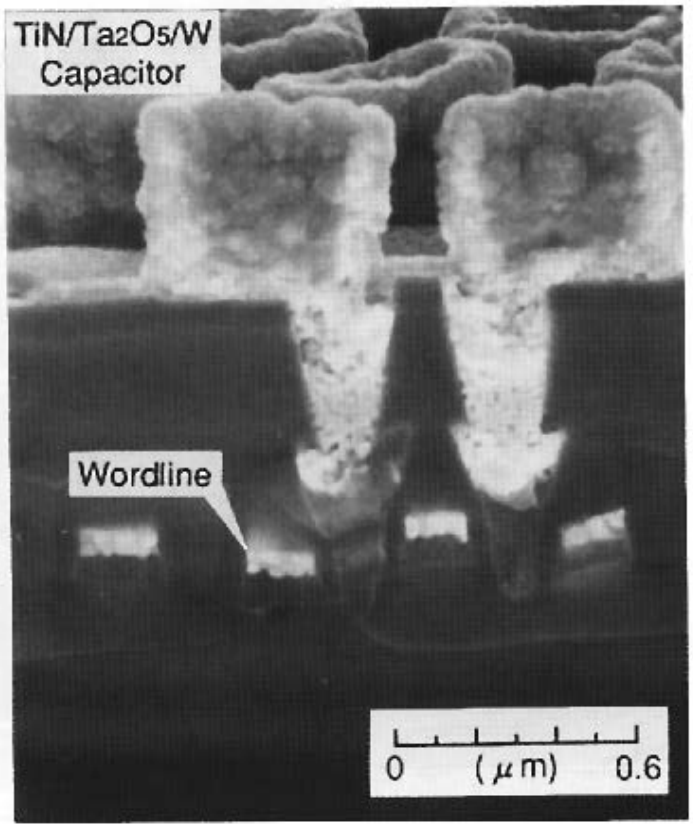
Chapter 5B

Large and Fast: Exploiting Memory Hierarchy

One Transistor Dynamic RAM





DRAM Cell Layout: $8F^2$

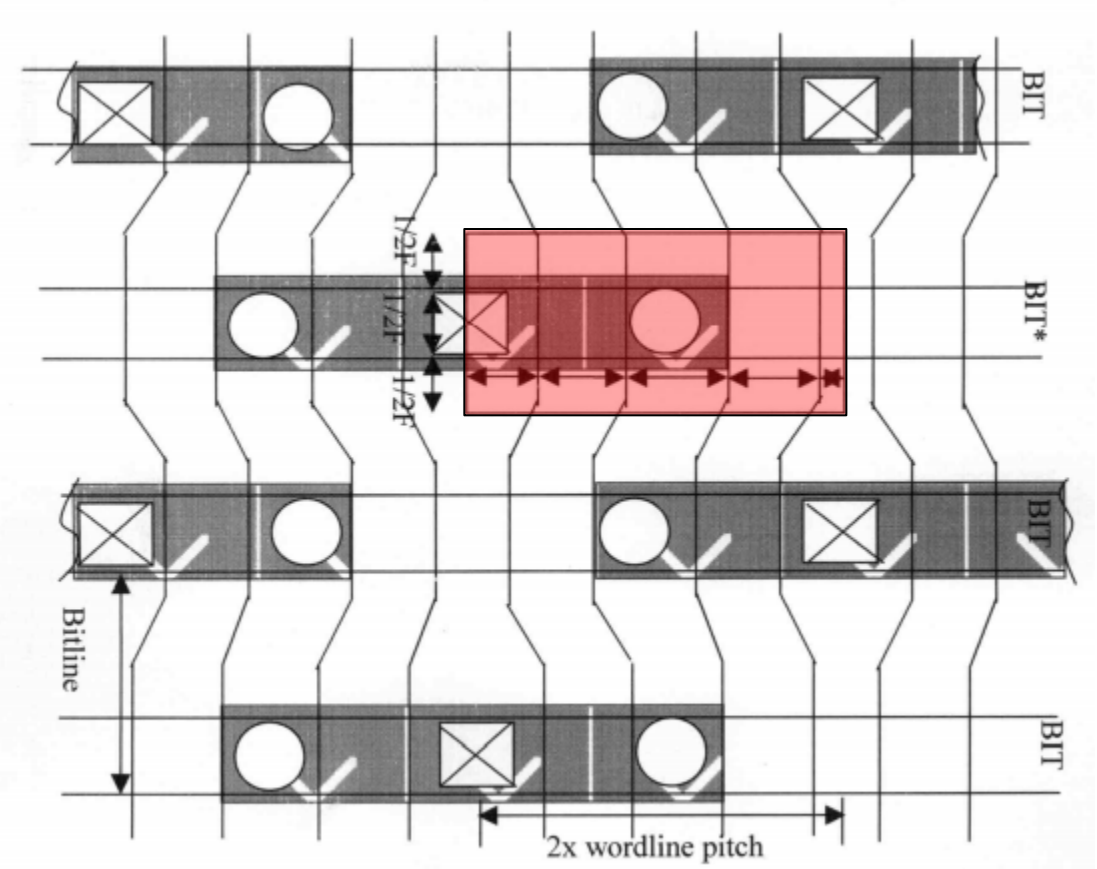
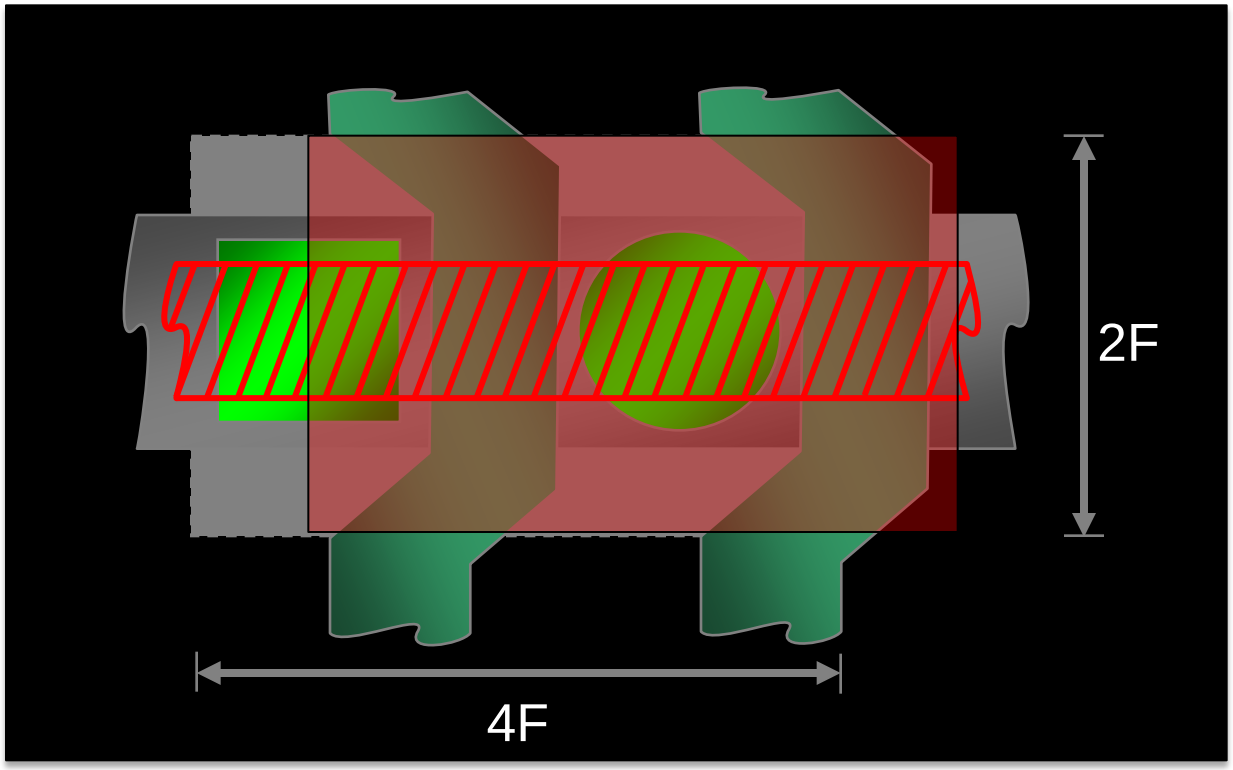


Fig. 5.2.1 Layout to show a $8F^2$ memory

DRAM Cell Layout: $8F^2$



Source: CMOS Circuit Design , Layout, and Simulation - Baker

DRAM Cell Layout: $6F^2$

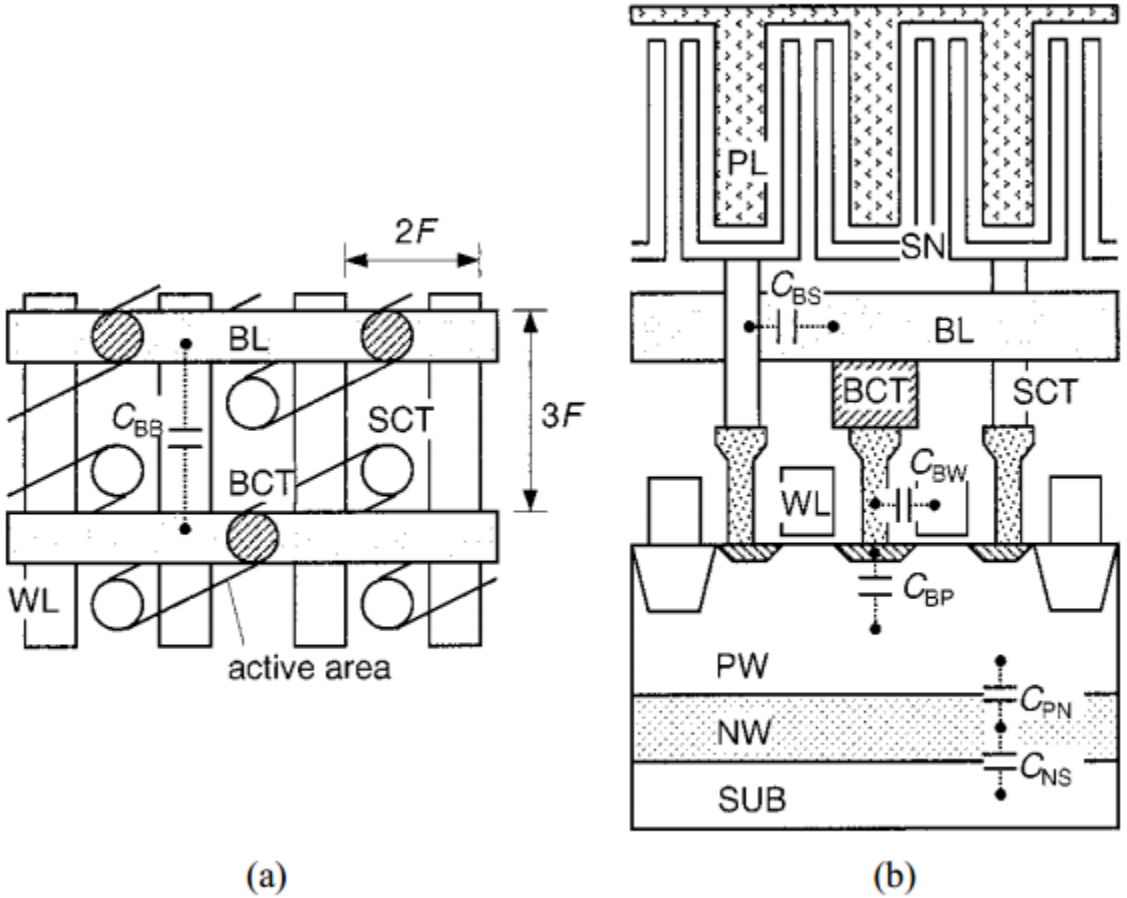
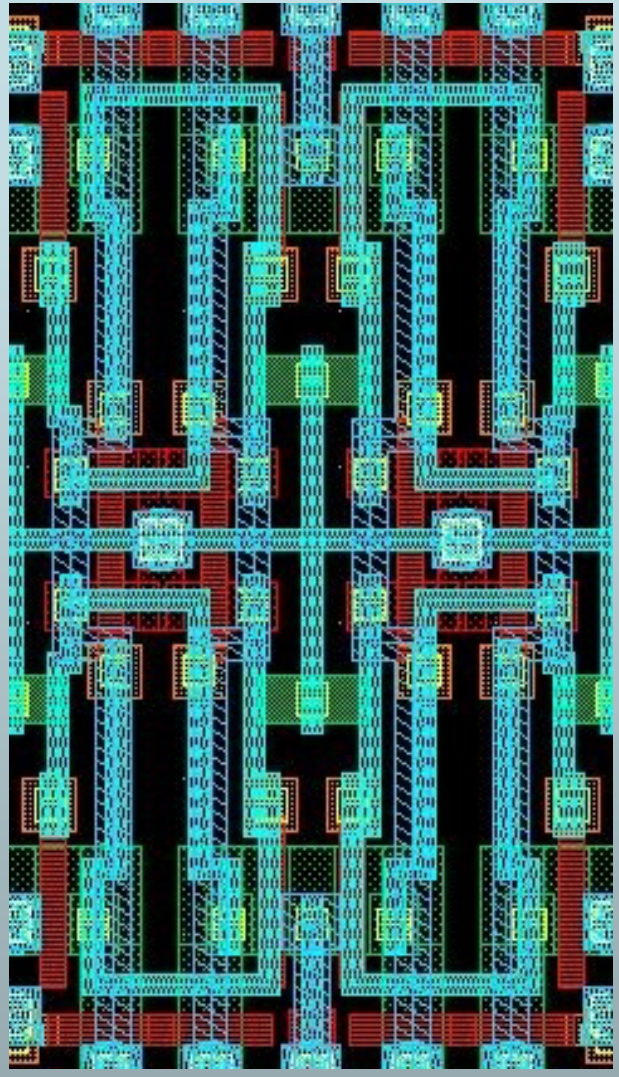


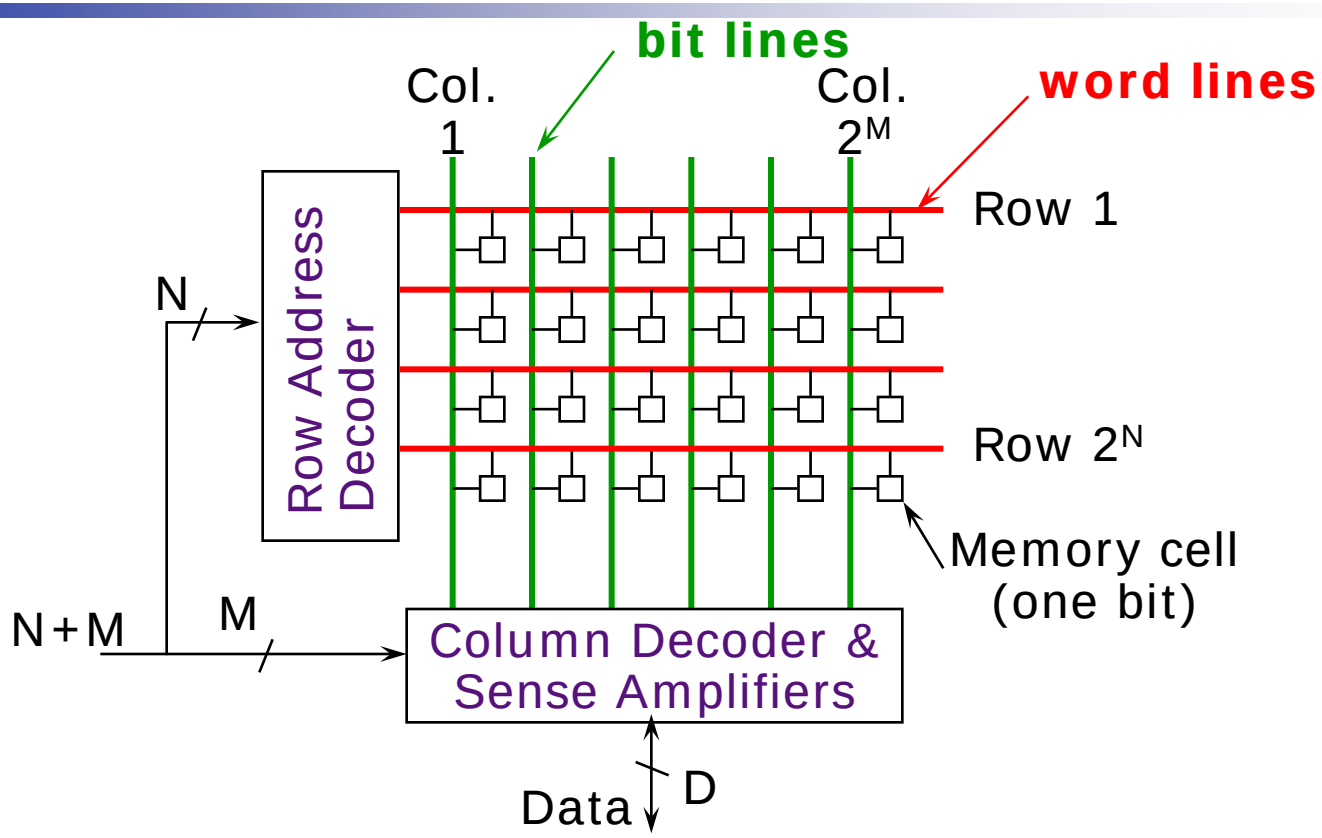
Fig. 1. $6F^2$ open-BL memory cell. (a) Top view. (b) Cross-sectional view.

SRAM Cell Layout: $140F^2$



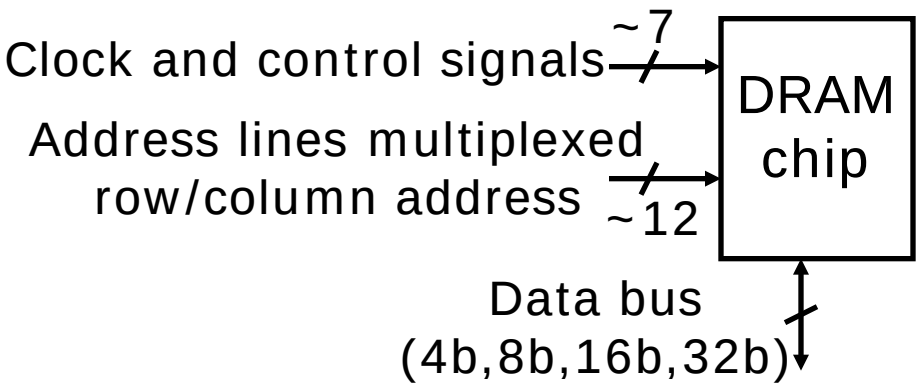
Source: CMOS Circuit Design , Layout, and Simulation - Baker

DRAM Architecture

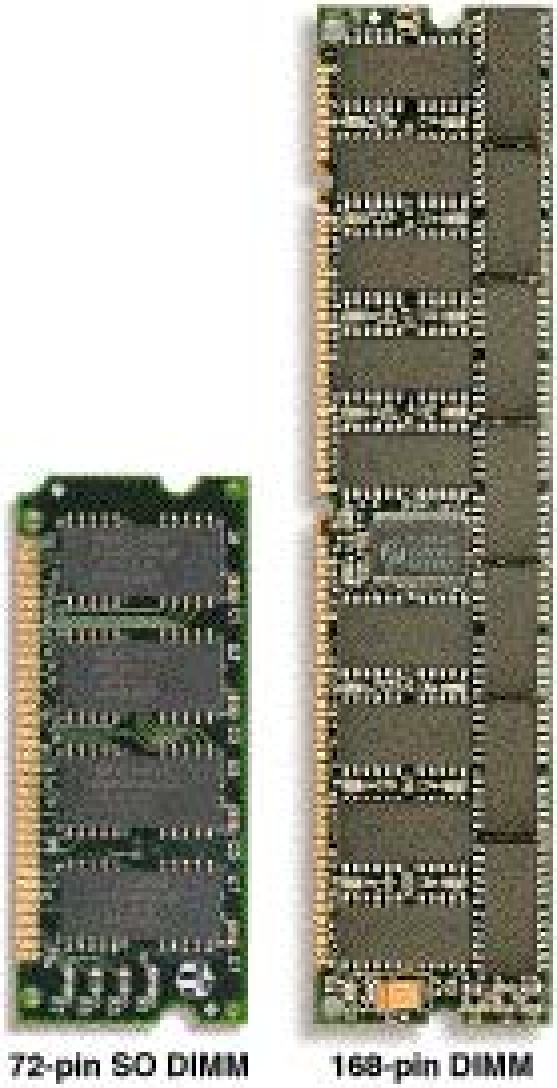


- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 4-8 logical banks on each chip
  - each logical bank physically implemented as many smaller arrays

DRAM Packaging

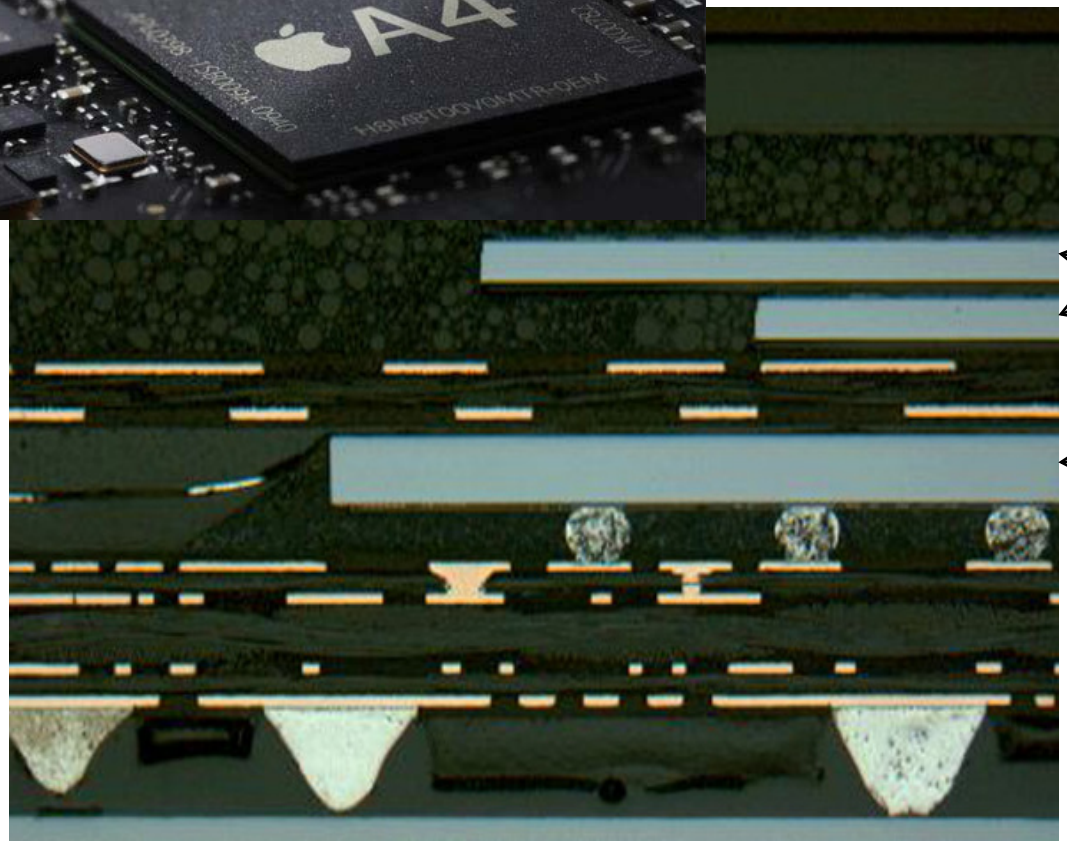


- DIMM (Dual Inline Memory Module) contains multiple chips with clock/control/address signals connected in parallel (sometimes need buffers to drive signals to all chips)
- Data pins work together to return wide word (e.g., 64-bit data bus using 16x4-bit parts)



DRAM Packaging, Mobile Devices

[Apple A4 package on circuit board]



[Apple A4 package cross-section, iFixit 2010]

DRAM Operation

- Three steps in read/write access to a given bank
  - Row access (RAS)
  - Column access (CAS)
  - Precharge: charges bit lines to known value, required before next row access
- Each step has a latency of around 15-20ns in modern DRAMs
- Various DRAM standards (DDR, RDRAM) have different ways of encoding the signals for transmission to the DRAM, but all share same core architecture

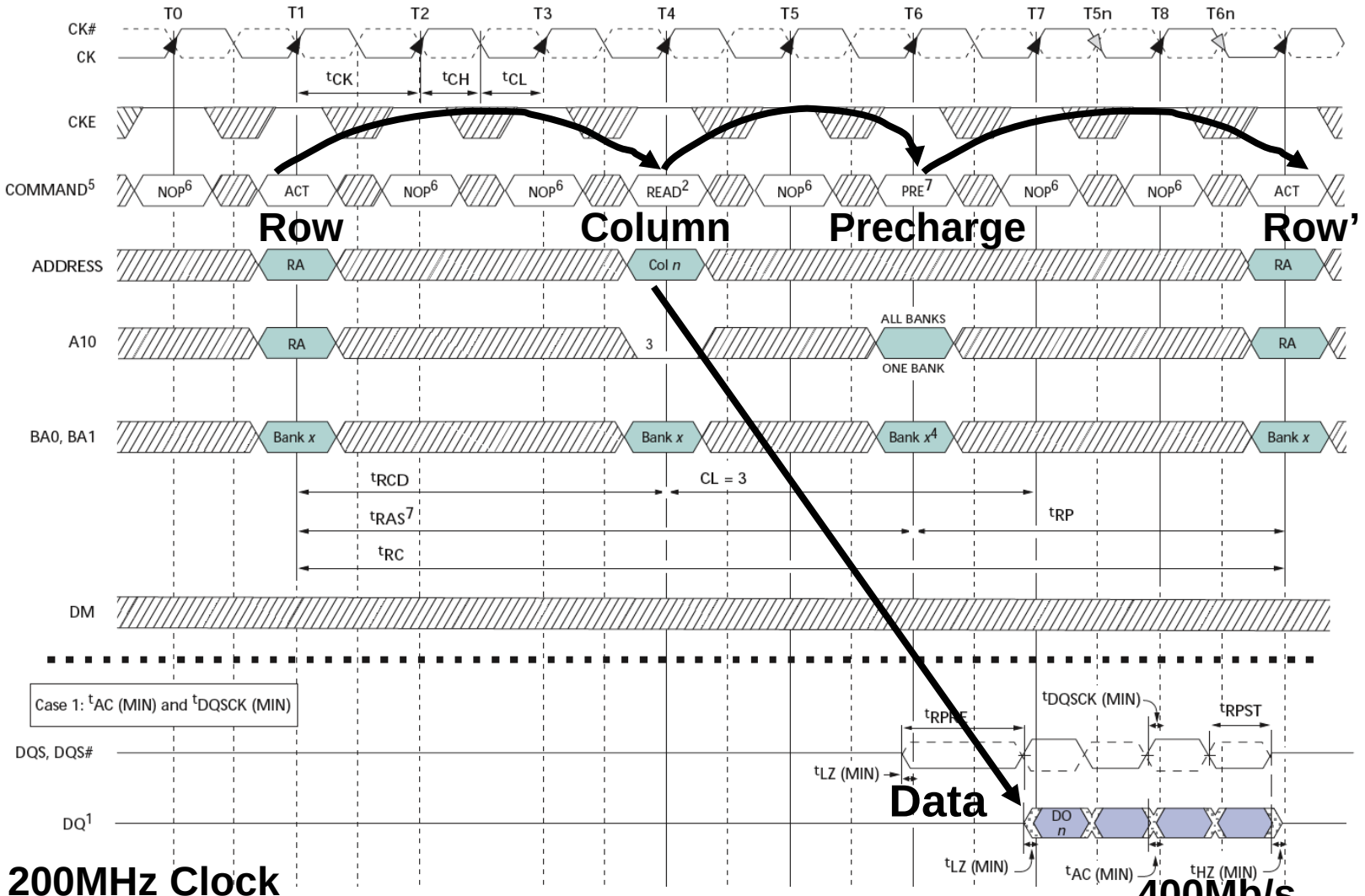
Row Access (RAS)

- decode row address, enable addressed row (often multiple Kb in row)
- bitlines share charge with storage cell
- small change in voltage detected by sense amplifiers which latch whole row of bits
- sense amplifiers drive bitlines full rail to recharge storage cells

Column Access (CAS)

- decode column address to select small number of sense amplifier latches (4, 8, 16, or 32 bits depending on DRAM package)
- on read, send latched bits out to chip pins
- on write, change sense amplifier latches which then charge storage cells to required value
- can perform sequentially multiple column accesses on same row without another row access (burst mode)

Double-Data Rate (DDR2) DRAM

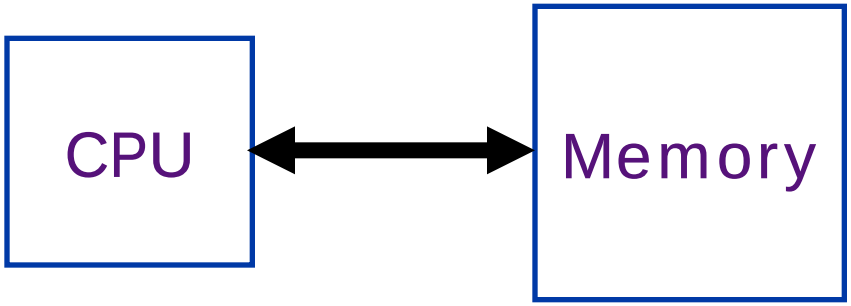


200MHz Clock

400Mb/s Data Rate

[Micron, 256Mb DDR2 SDRAM datasheet]

CPU-Memory Bottleneck



Performance of high-speed computers is usually limited by memory *bandwidth* & *latency*

- Latency (time for a single access)
  Memory access time >> Processor cycle time

- Bandwidth (number of accesses per unit time)
  if fraction *m* of instructions access memory,
  ⇒1+*m* memory references / instruction
  ⇒CPI = 1 requires 1+*m* memory refs / cycle
  (assuming MIPS RISC ISA)

Causes for Cache Misses

Compulsory miss*:* first-reference to a block *a.k.a.* cold start misses

- Occurs because all blocks are invalid at the beginning.
- misses that would occur even with infinite cache

- Capacity miss*:* cache is too small to hold all data needed by the program
  - Occurs if the number of different blocks in requests is more than the number of blocks in the cache

- Conflict miss*:* misses that occur because of collisions due to block-placement strategy
  - Occurs if the number of requests for a specific set is more than its way number.
  - misses that would not occur with full associativity

Effect of Cache Parameters

- Larger cache size
  + reduces capacity and conflict misses
  - hit time will increase

- Higher associativity
  + reduces conflict misses
  - may increase hit time

- Larger block size
  + reduces compulsory and capacity misses
  - increases conflict misses and miss penalty

Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
  - VM “block” is called a page
  - VM translation “miss” is called a page fault

Absolute Addresses

EDSAC, early 50's

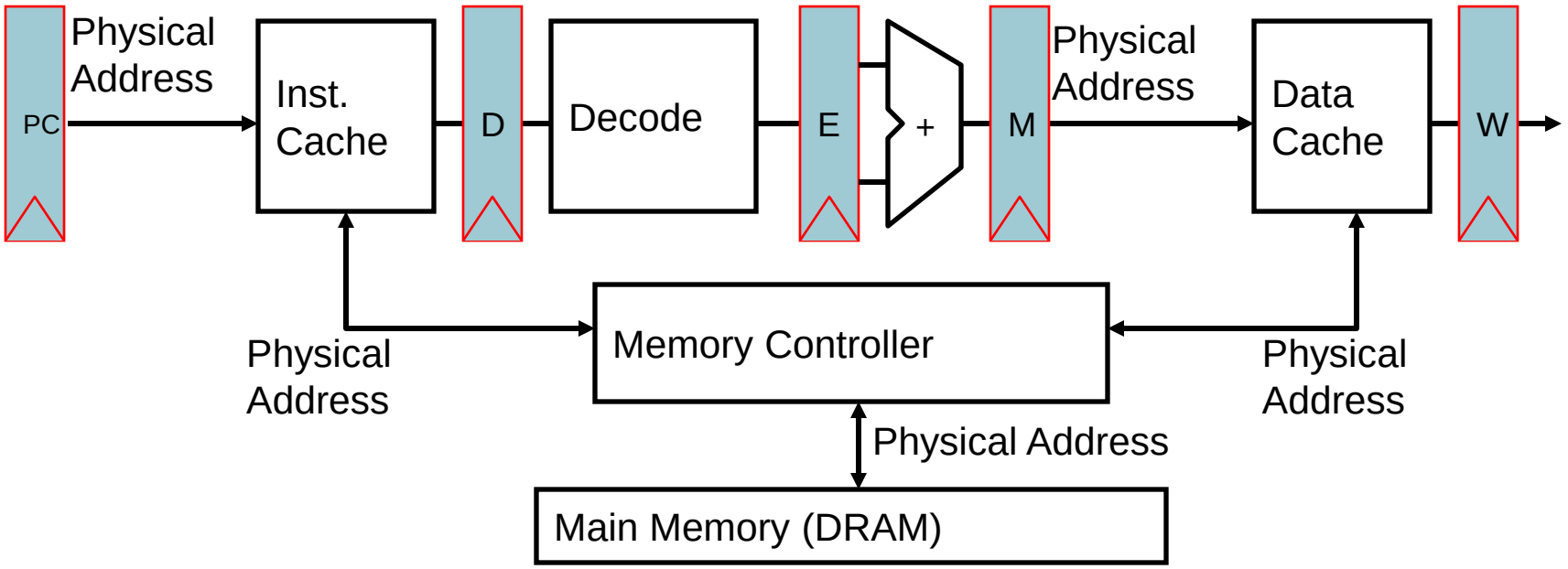
- Only one program ran at a time, with unrestricted access to entire machine (RAM + I/O devices)
- Addresses in a program depended upon where the program was to be loaded in memory
- *But* it was more convenient for programmers to write location-independent subroutines

How could location independence be achieved?

No absolute address in code; use only relative address

Linker and/or loader modify addresses of subroutines and callers when building a program memory image

Bare Machine



- In a bare machine, the only kind of address is a physical address: no address translation

Memory Management

- From early absolute addressing schemes, to modern virtual memory systems with support for virtual machine monitors (VMMs)
- Can separate into orthogonal functions:
  - Translation (mapping of virtual address to physical address)
  - Protection (permission to access word in memory)
  - Virtual memory (transparent extension of memory space using slower disk storage)
- But most modern systems provide support for all the above functions with a single page-based system

Protection

- The first motivation for virtual memory
- A set of mechanisms for ensuring that multiple processes sharing the processor, memory, or I/O devices cannot interfere with one another by reading or writing each other's data.
- These mechanisms also isolate the OS from a user process.

Overlay

- Programmers divided program into pieces and then identified the pieces that were mutually exclusive.
- These overlays were loaded or unloaded under user program control during execution, with the programmer ensuring that the program never exceeded the total size of the memory.
  - Overlays were traditionally organized as modules, each containing both code and data.
- Sharing was not the reason that virtual memory was invented!

Overlay

- The second motivation for virtual memory
  - To allow a single user program to exceed the size of primary memory
- Relocation: the same program can run any position in physical memory

Dynamic Address Translation

Motivation

In the early machines, I/O operations were slow and each word transferred involved the CPU

Higher throughput if CPU and I/O of 2 or more programs were overlapped.

How?⇒ multiprogramming with DMA I/O devices, interrupts

Location-independent programs

Programming and storage management ease

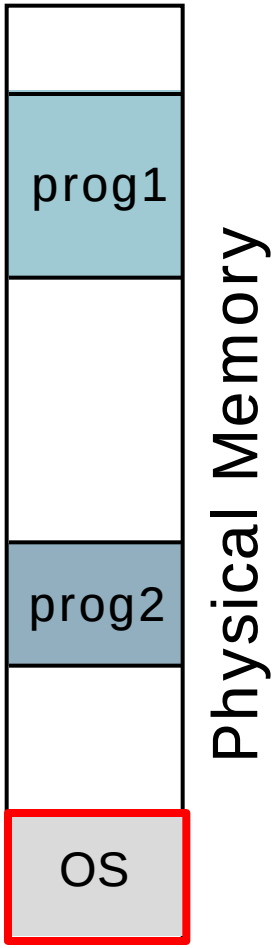
⇒ need for a *base register*

Protection

Independent programs should not affect each other inadvertently

⇒ need for a *bound register*

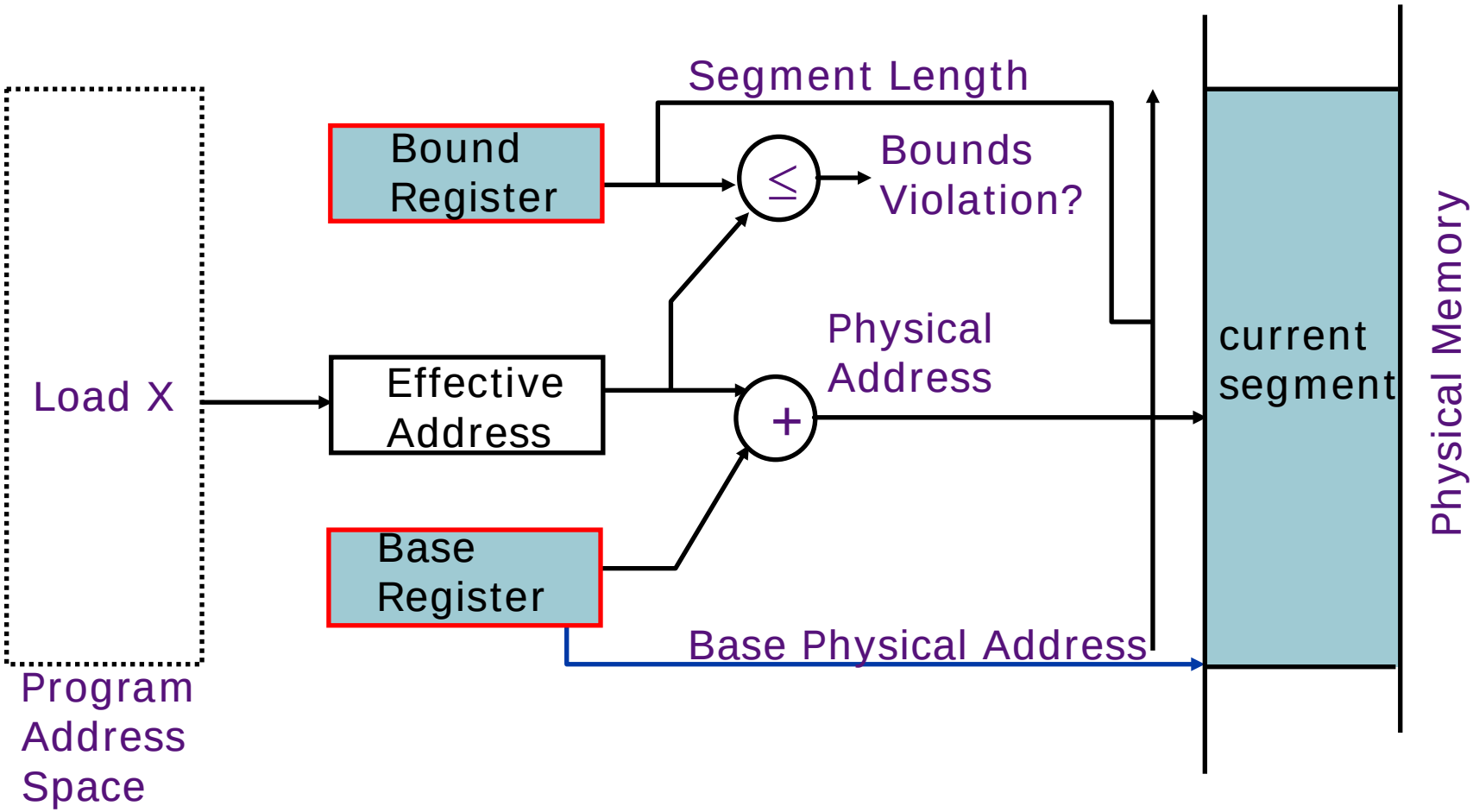
Multiprogramming drives requirement for resident *supervisor* software to manage context switches between multiple programs



Segmentation vs Paging

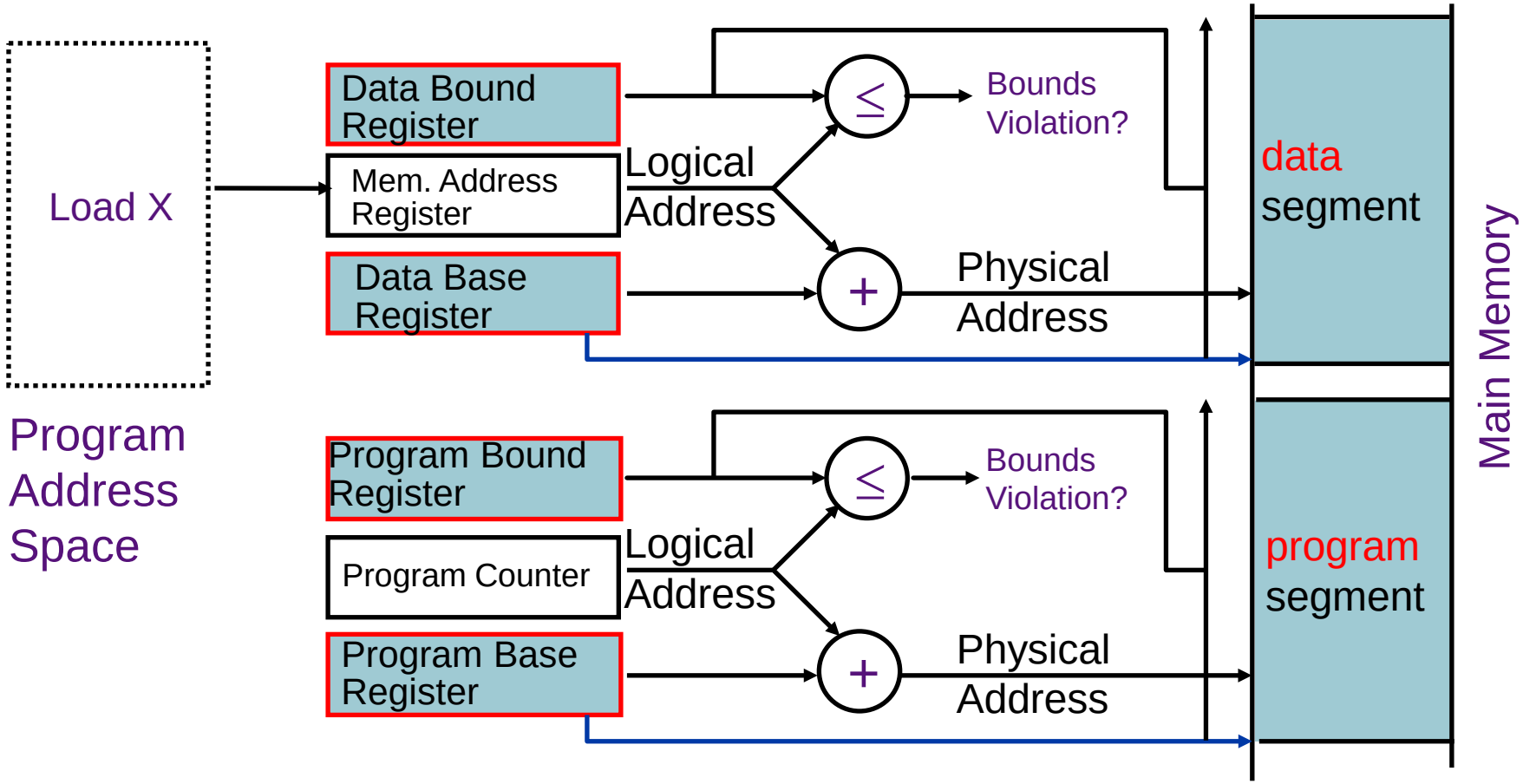
- Note that paging uses fixed-size blocks
- Segmentation: a variable-size block scheme
  - A segmentation number and a segmentation offset
  - A bound check is also needed to make sure that the offset is within the segment
- The major use of segmentation
  - to support more powerful methods of protection and sharing in an address space
- The major disadvantage of segmentation
  - It splits the address space into logically separate pieces (segments) that must be manipulated as a two-part address (base and bound addresses).

Base and Bound Registers



Base and bounds registers are visible/accessible only when processor is running in the *supervisor mode*

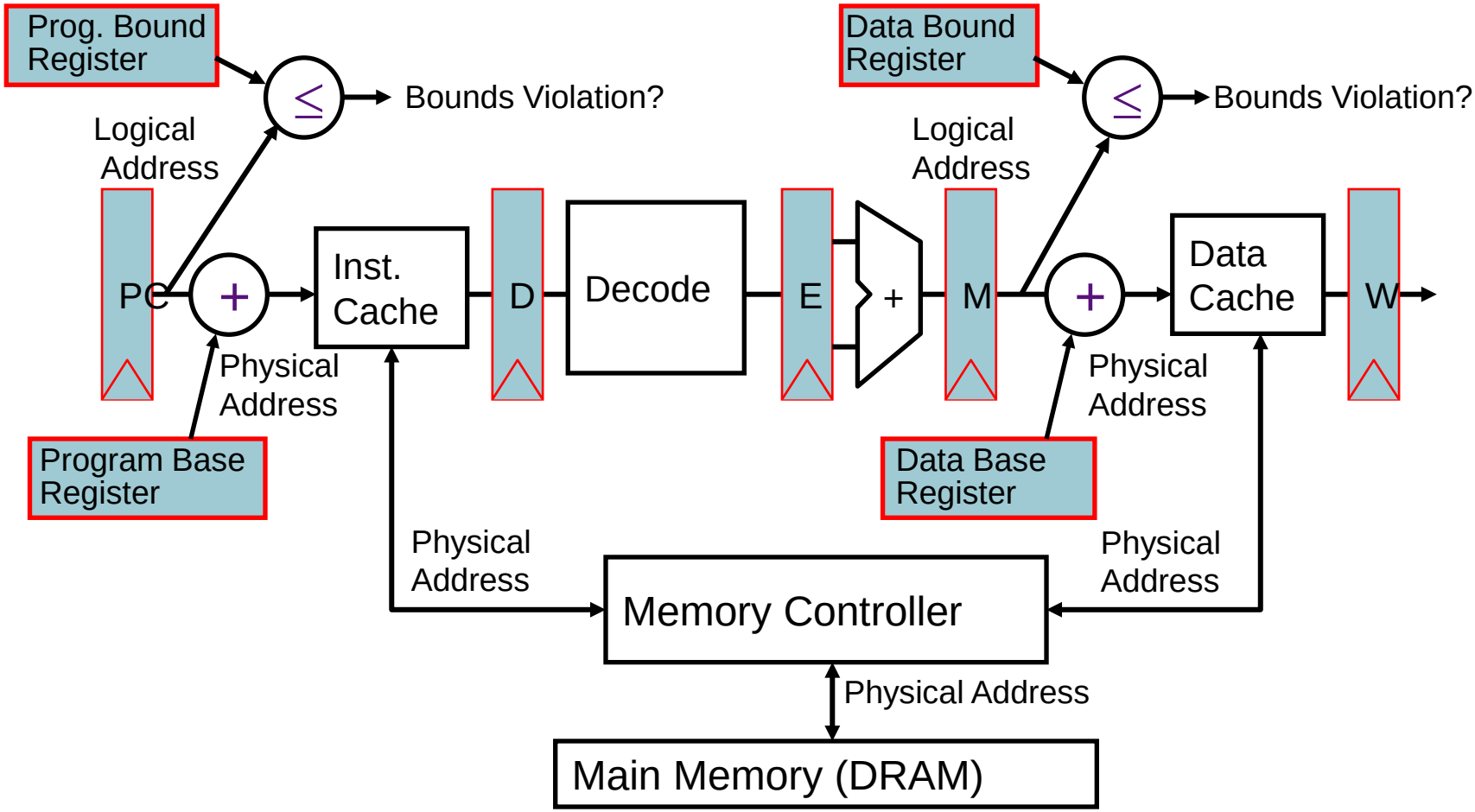
Separate Areas for Program and Data



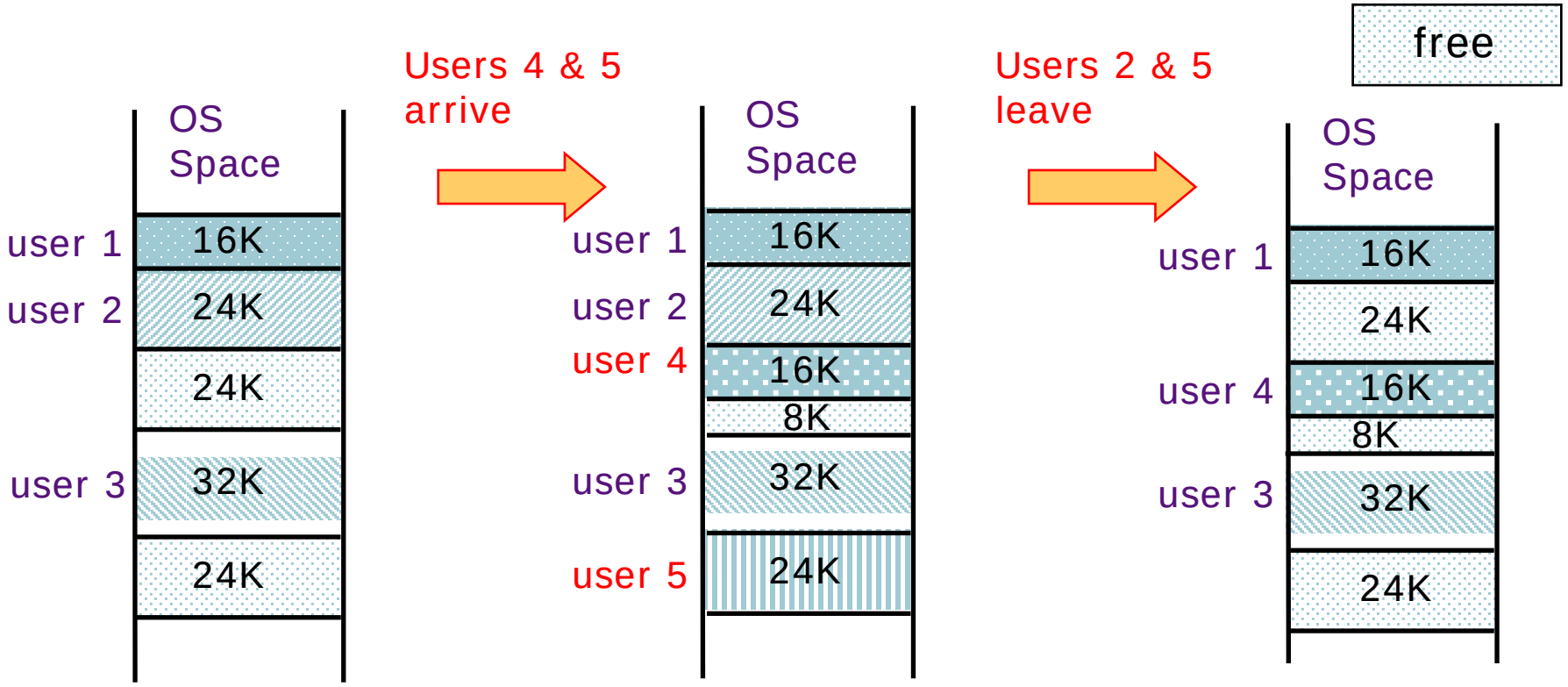
What is an advantage of this separation?

1. protection 2. permit sharing program segments

Base and Bound Machine



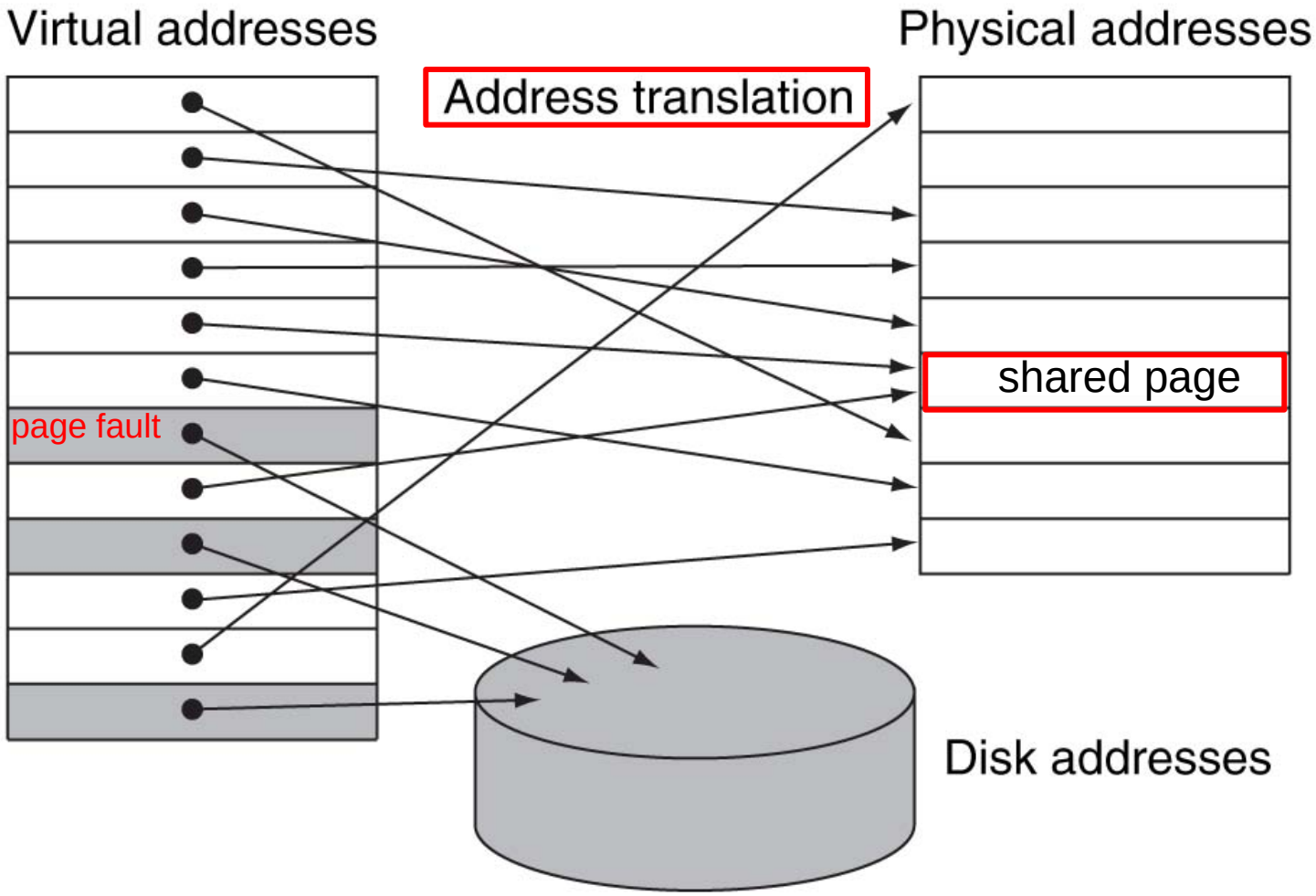
Memory Fragmentation



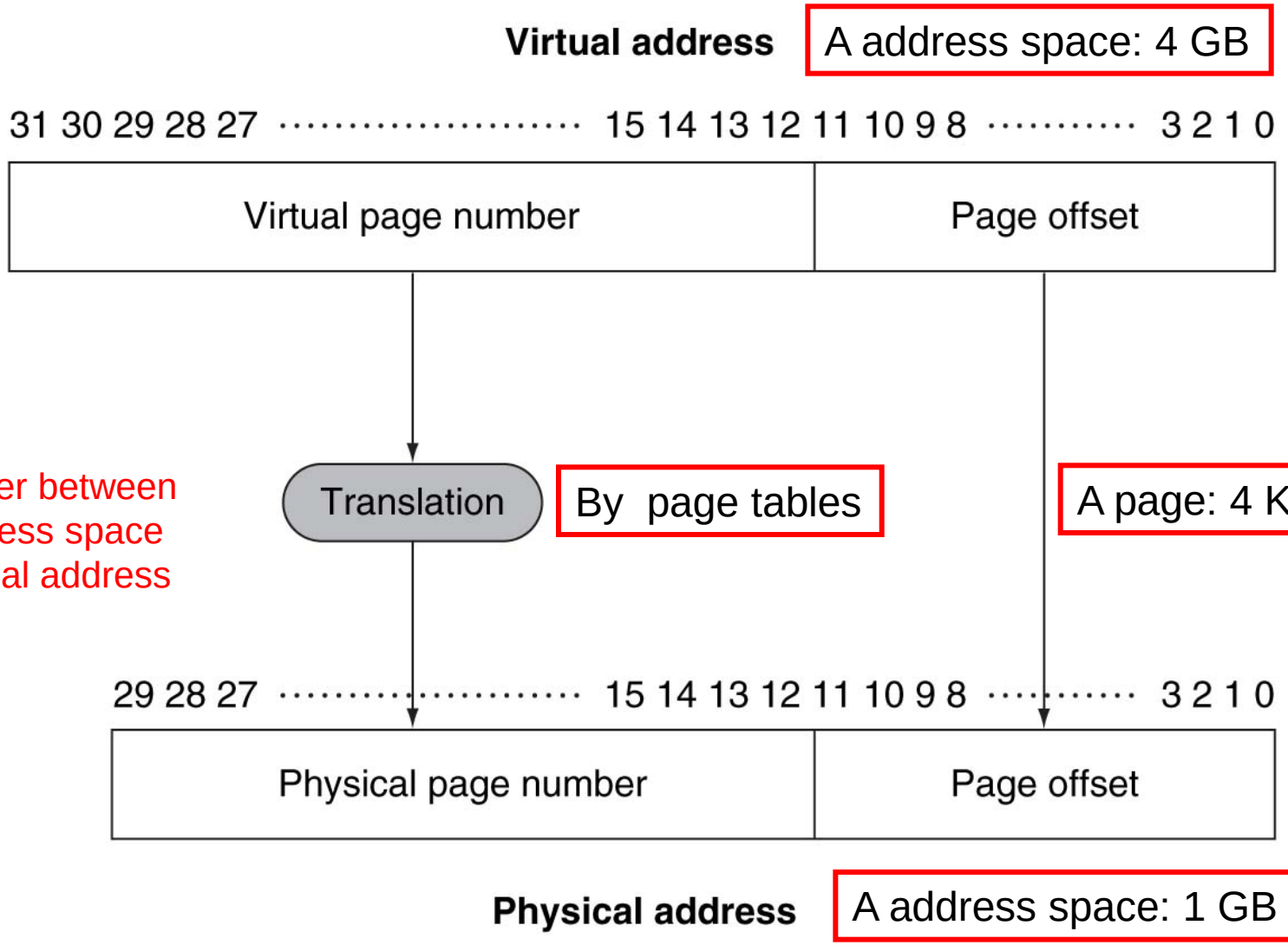
As users come and go, the storage is external "fragmented". Therefore, at some stage programs have to be moved around to compact the storage.

Address Translation

An event that occurs when an accessed page is not present in main memory



Address Translation

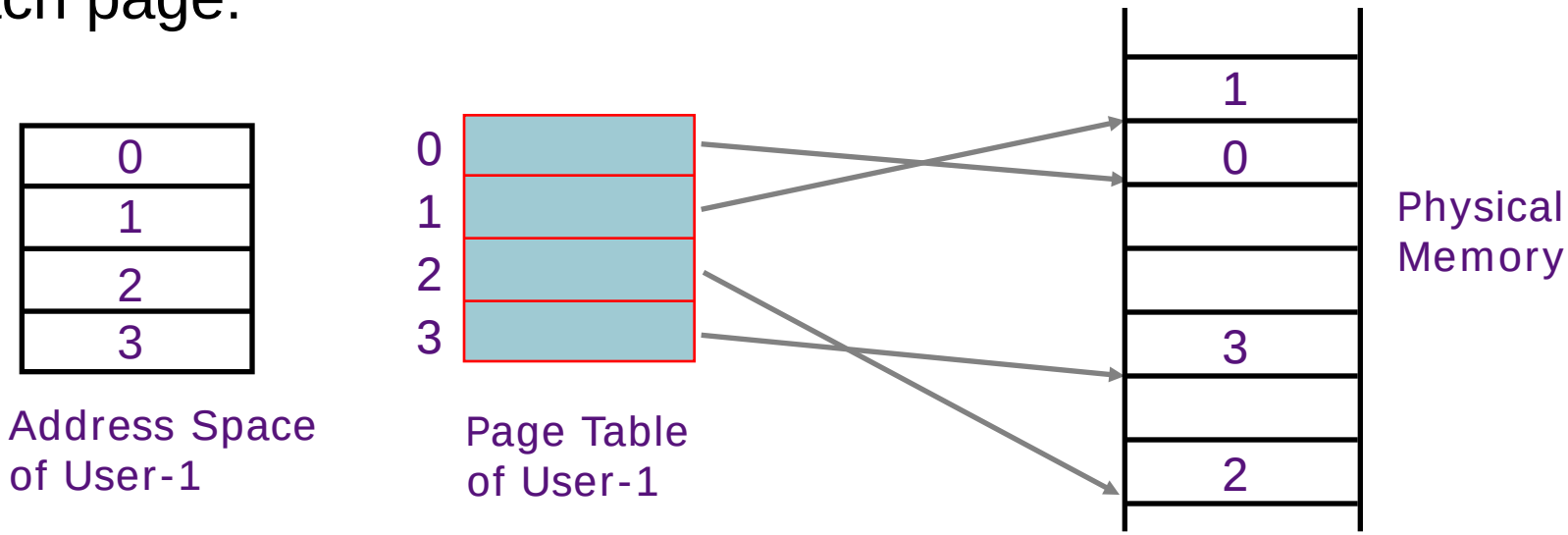


Which is larger between a virtual address space and a physical address space?

Paged Memory Systems

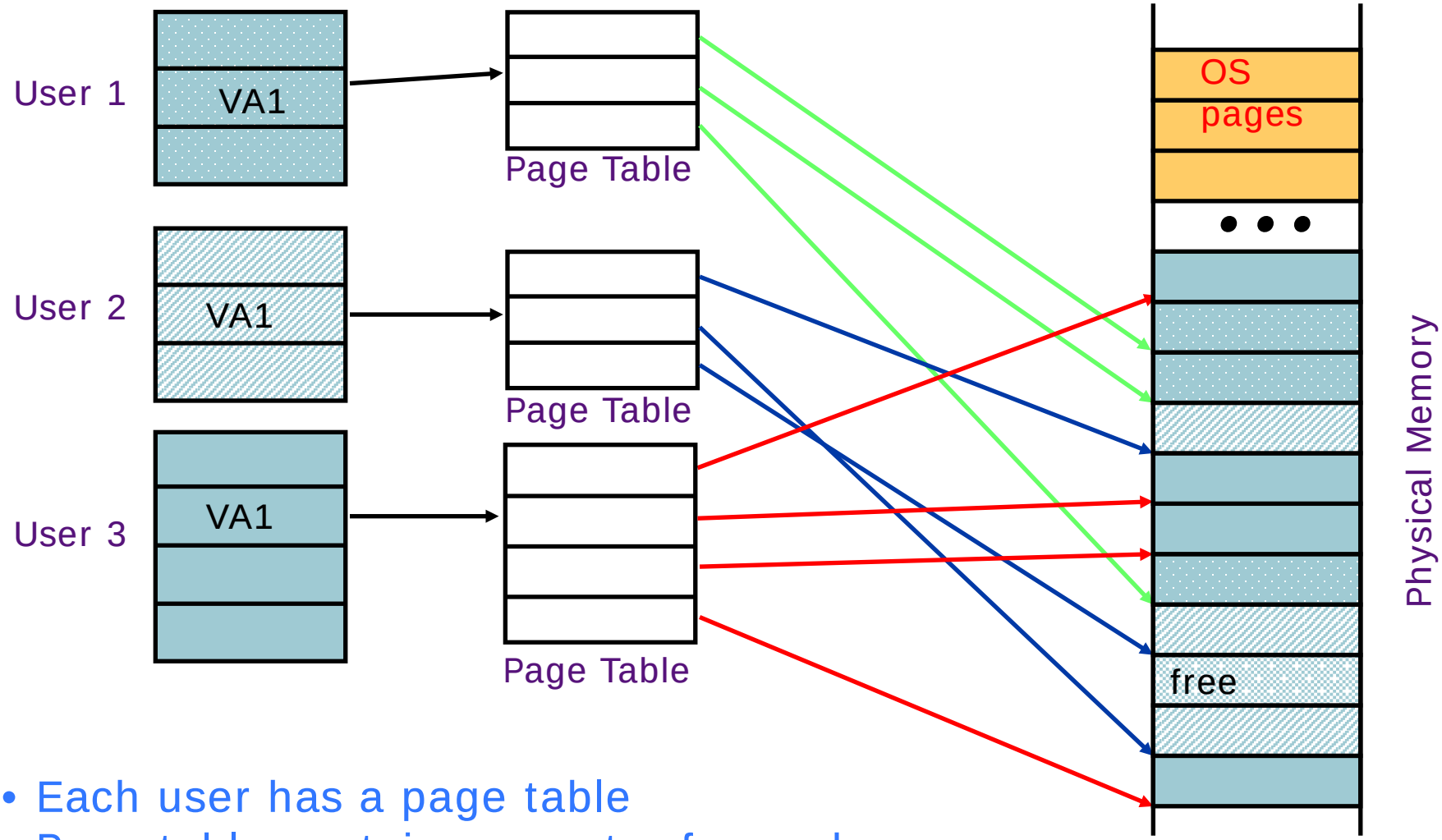
- Processor-generated address can be split into:

- A page table contains the physical address of the base of each page:



Page tables make it possible to store the pages of a program non-contiguously.

Private Address Space per User



- Each user has a page table
- Page table contains an entry for each user page

Where Do Page Tables Reside?

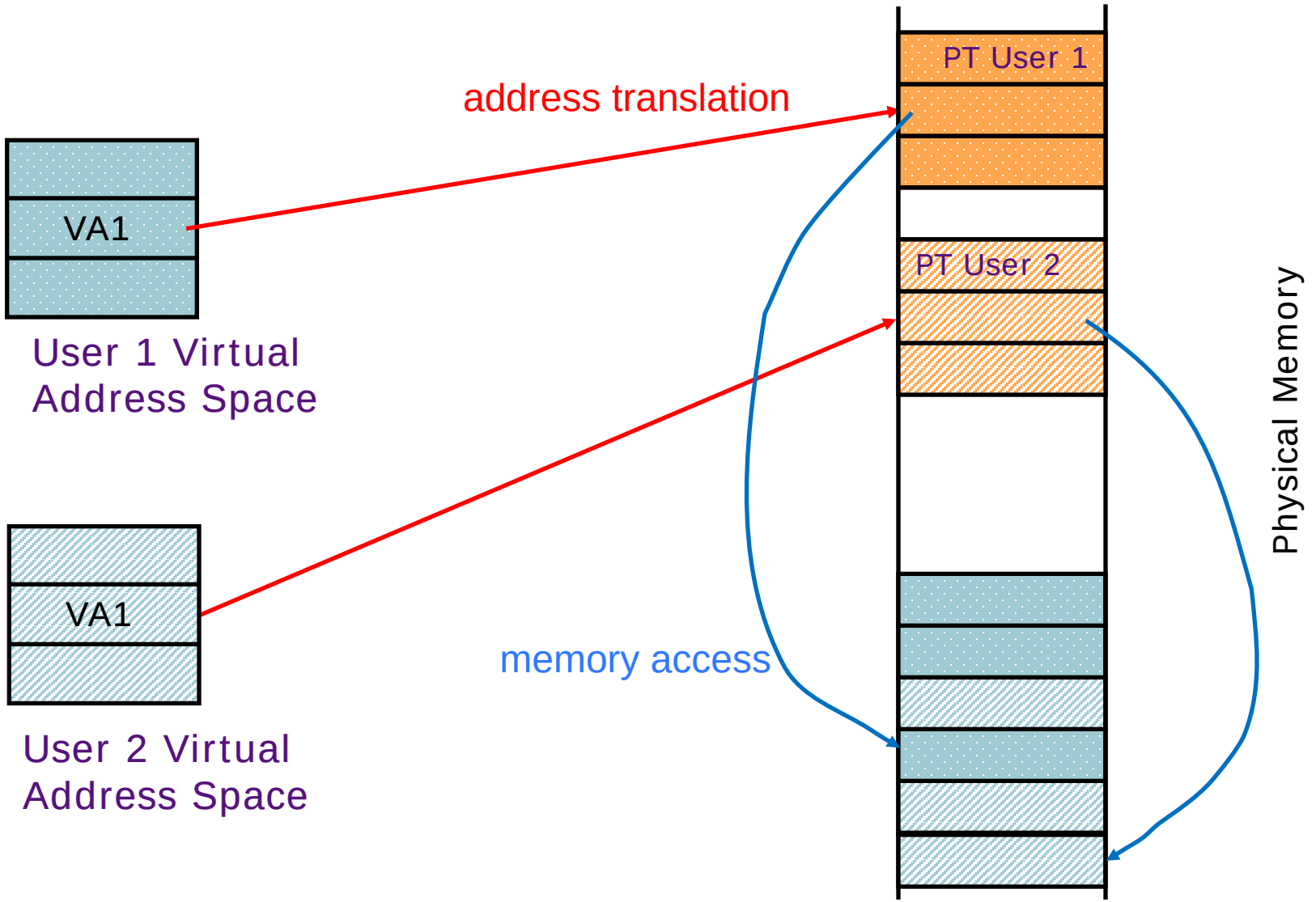
- Space required by the page tables (PT) is proportional to the address space, number of users, ...

  ⇒ Space requirement is huge

- Simple idea: Keep PTs in the main memory
  - needs one reference to retrieve the page base address and another to access the data word

    ⇒ *doubles the number of memory references!*

Page Tables in Physical Memory



Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code (software) because its overhead is small compared to the disk access time
- Try to minimize page fault rate
  - Fully associative placement of pages in memory
    - Can be placed anywhere in main memory
  - Smart replacement algorithms (e.g. LRU) can be used
  - Write-through will not work since writes take too long. Instead, write-back is used in virtual memory

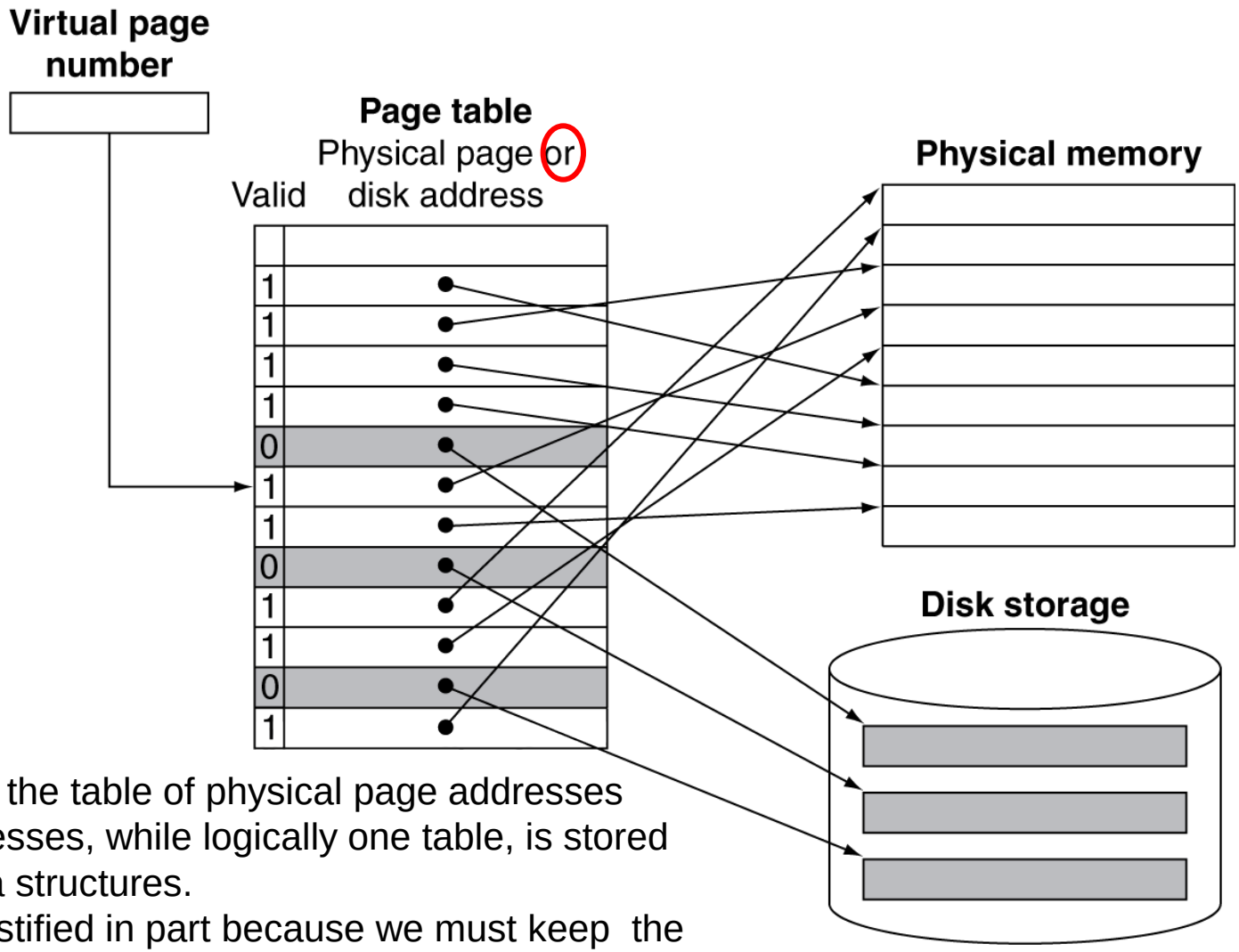
Fully associative replacement

- A page can be anywhere in the main memory
  - A full search by comparing a tag is impractical
- In virtual memory we locate pages by using a table that indexes pages in the memory
  - This structure is called a page table.
- The page table stores placement information
  - Each page table entry (PTE) is indexed by virtual page number
  - A page table base register in CPU points to the starting address of the page table in the main memory

PTE

- If the page is present in memory,
  - its PTE stores the physical page number and
  - status bits (referenced, dirty, …)
- If the page is not present in memory,
  - its PTE can refer to a location in the swap space on disk

Mapping Pages to Storage

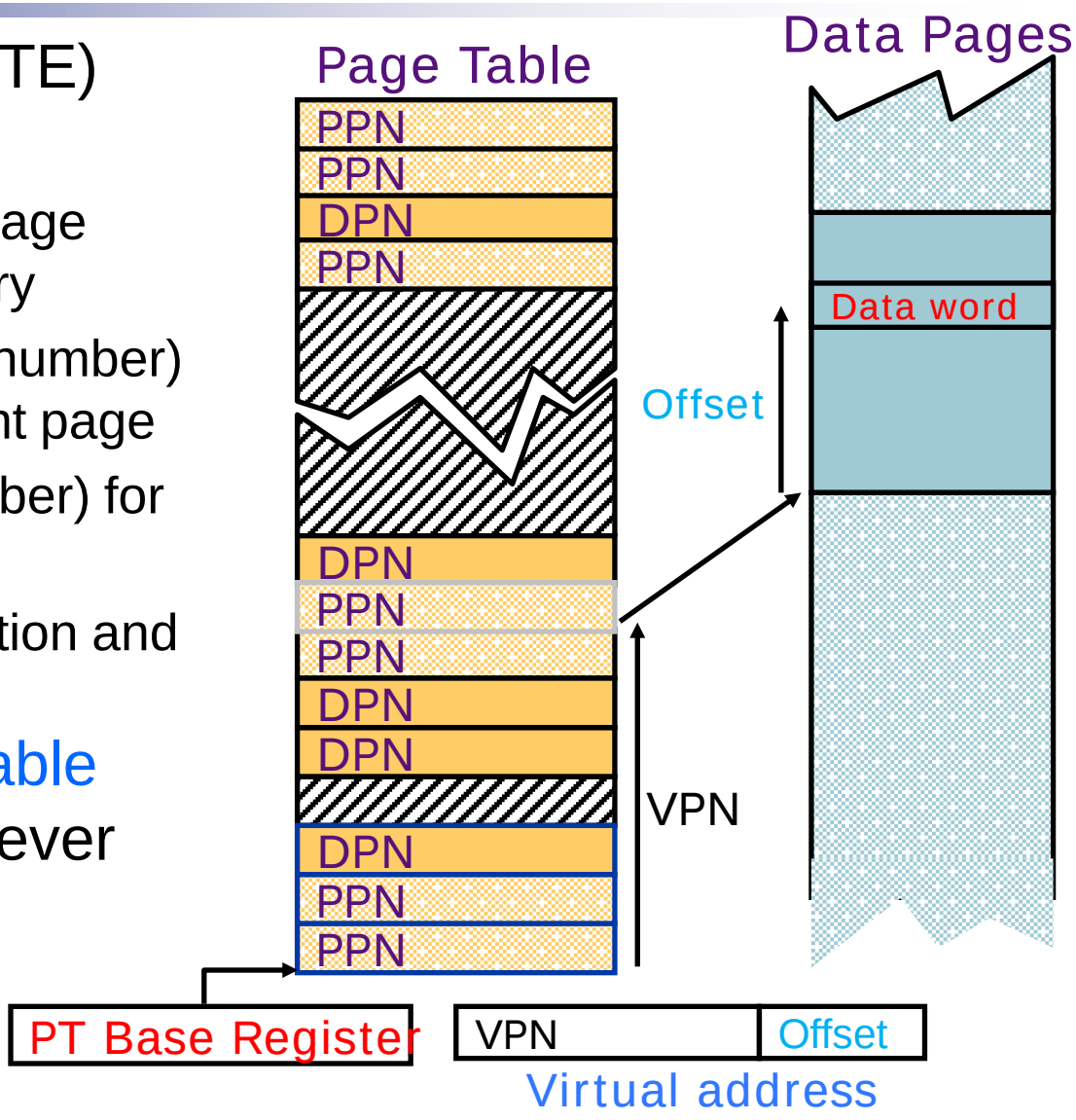


◆In many systems, the table of physical page addresses and disk page addresses, while logically one table, is stored in two separate data structures.

◆Dual tables are justified in part because we must keep the disk addresses of all the pages, even if they are currently in main memory.

Linear Page Table

- Page Table Entry (PTE) contains:
  - A bit to indicate if a page exists in main memory
  - PPN (physical page number) for a memory-resident page
  - DPN (disk page number) for a page on the disk
  - Status bits for protection and usage
- OS sets the Page Table Base Register whenever active user process changes



Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
  - Write in Block at once, not in location
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written

Size of Linear Page Table

With 32-bit addresses, 4-KB pages & 4-byte PTEs:

- ⇒ 2^{20} PTEs, i.e, 4 MB page table per user
- ⇒ 4 GB (2^{32}) of swap needed to back up full virtual address space

Larger pages?

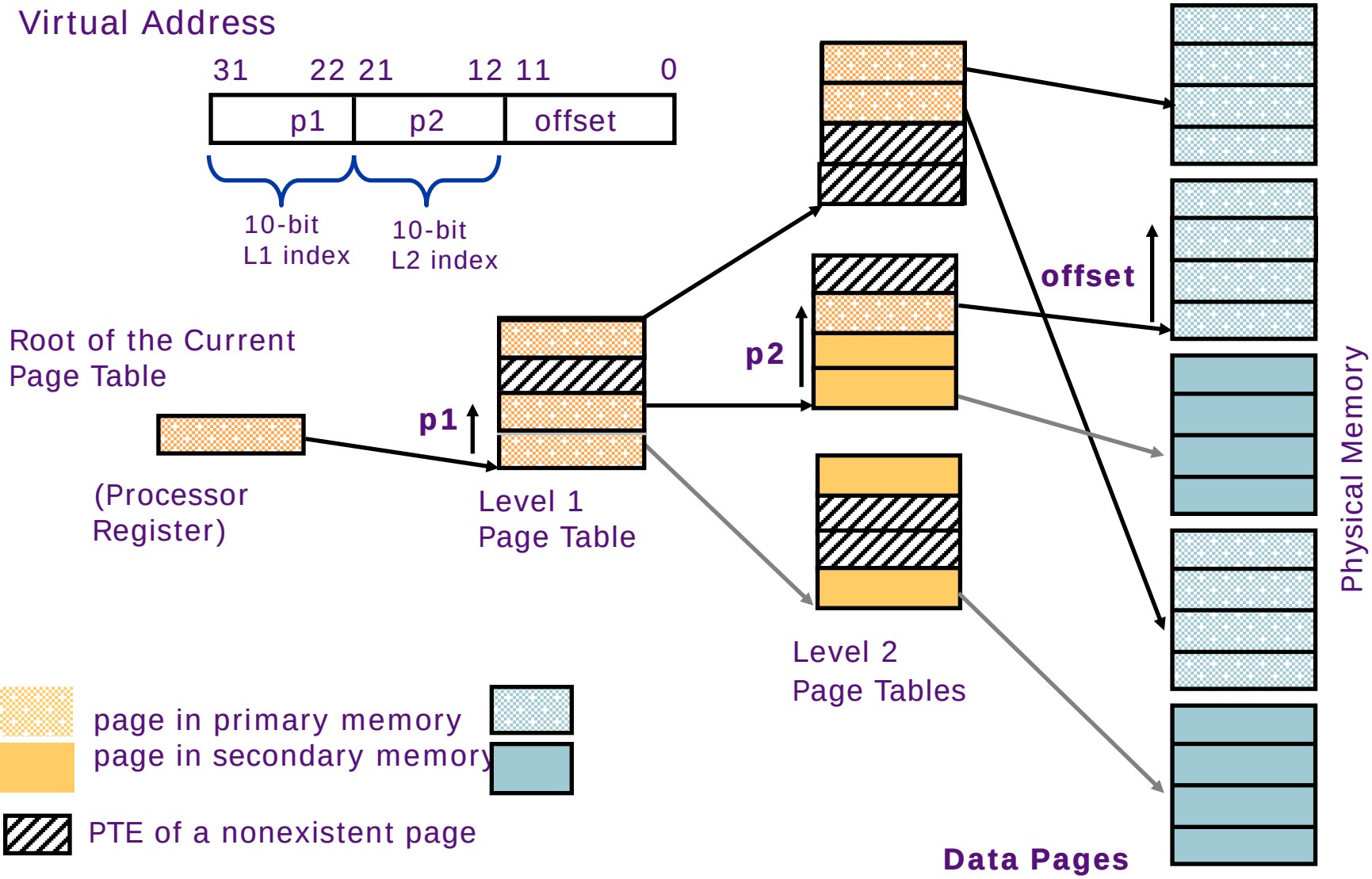
- Internal fragmentation (Not all memory in page is used)
- Larger page fault penalty (more time to read from disk)

What about 64-bit virtual address space???

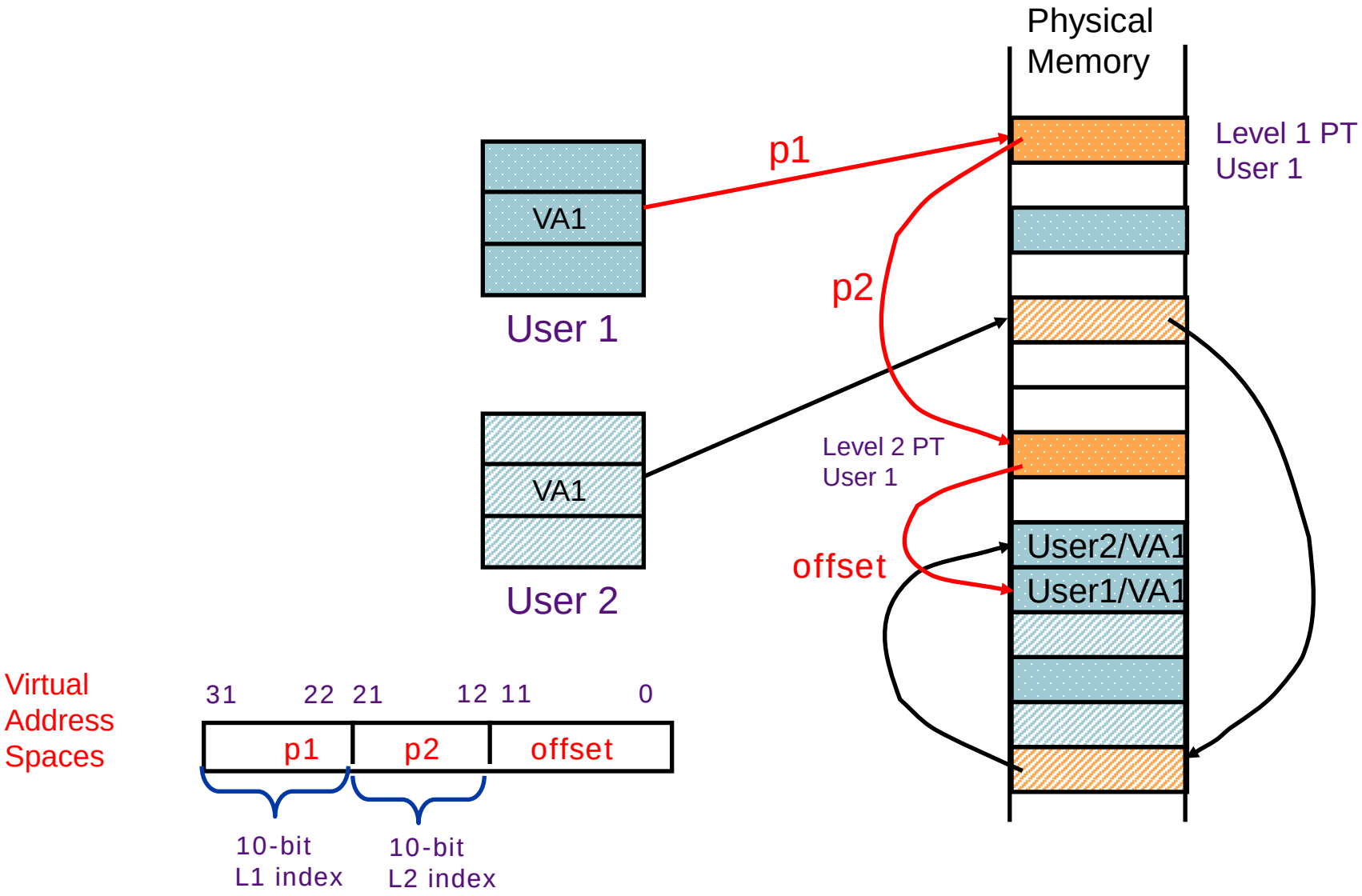
- Even 1MB pages would require 2^{44} 8-byte PTEs (35 TB!)

What is the "saving grace" ?

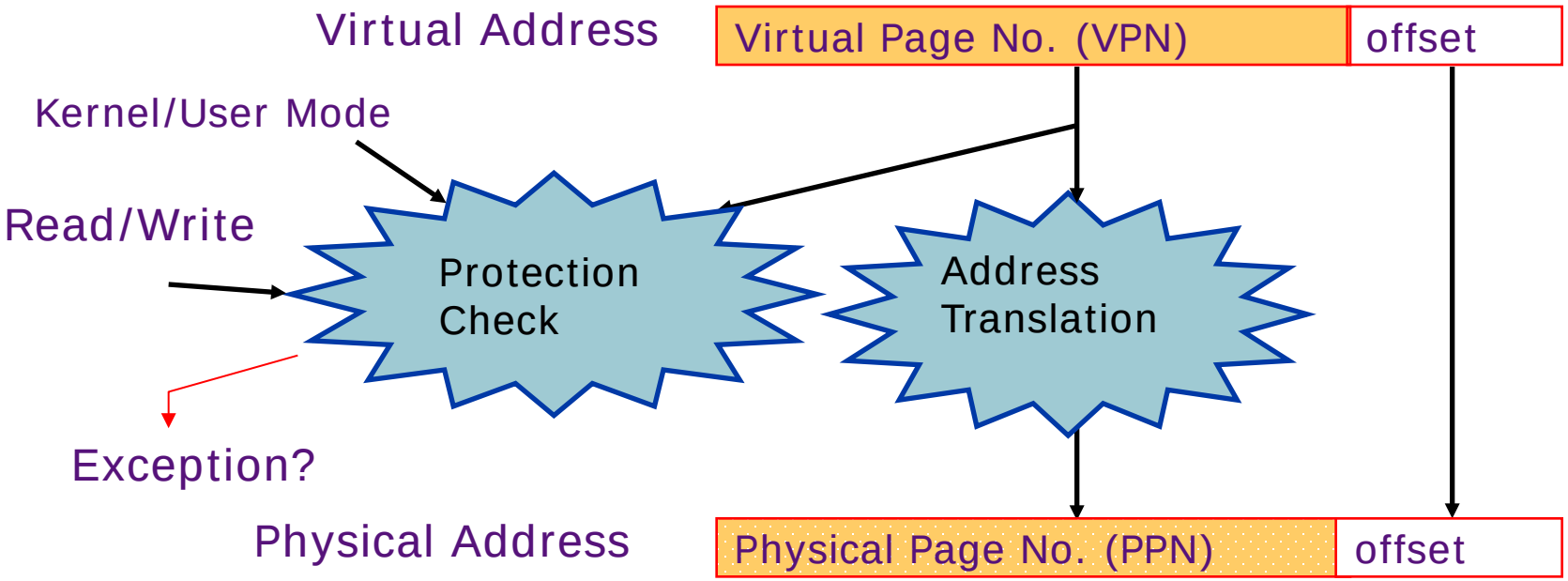
Hierarchical Page Table



Two-Level Page Tables



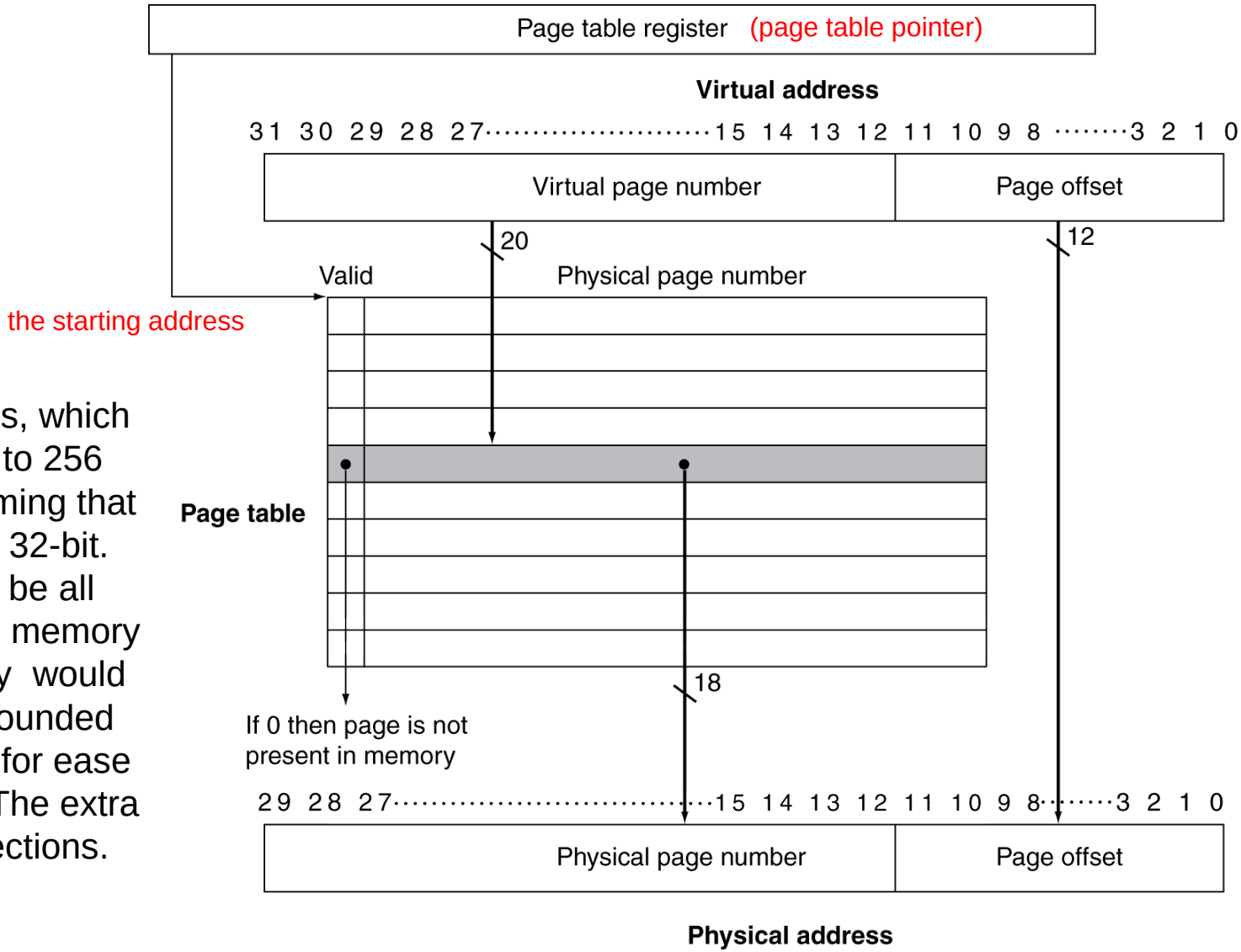
Address Translation & Protection



- Every instruction and data access needs address translation and protection checks

A good VM design needs to be fast (~ one cycle) and space efficient

Translation Using a Page Table



- 1 M entries, which corresponds to 256 pages, assuming that each entry is 32-bit.
- too big to be all entries in the memory
- Each entry would typically be rounded up to 32 bits for ease of indexing. The extra bits for protections.

Fast Translation Using a TLB

- Address translation would appear to require extra memory references
  - One to access the PTE
  - Then the actual memory access
- But access to page tables has good (?) locality
  - So use a translation cache (a fast cache of the page table) within the CPU
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - TLB Misses could be handled by hardware or software

Translation Lookaside Buffers

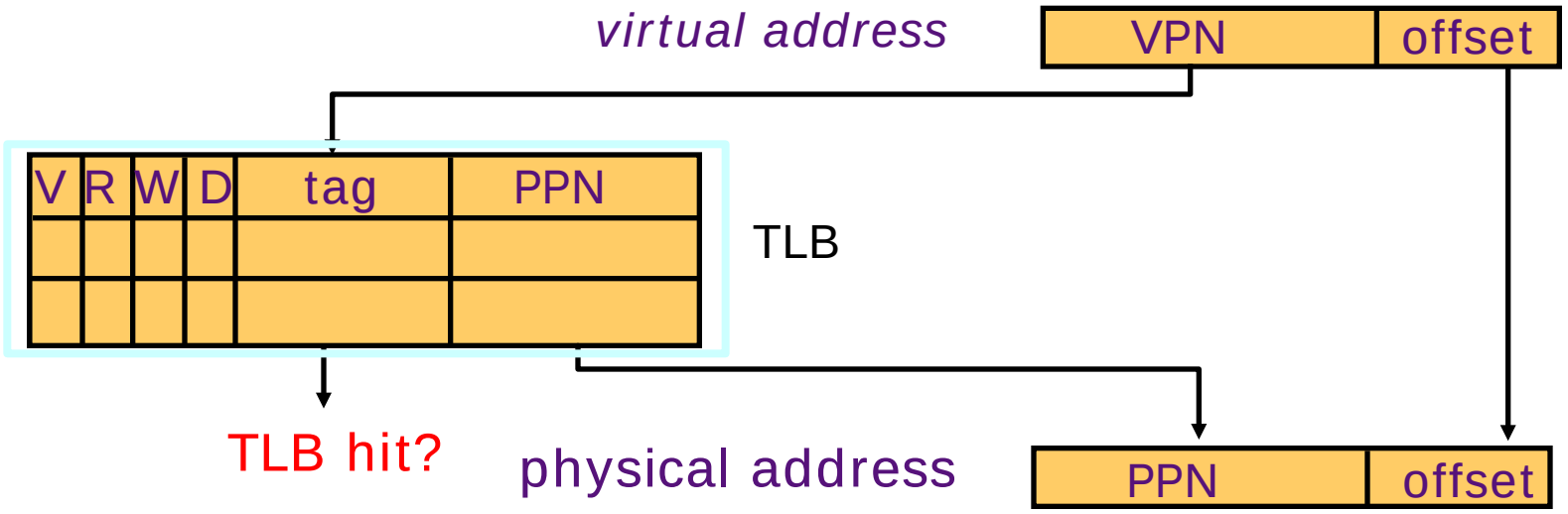
Address translation is very expensive!

In a two-level page table, each reference becomes several memory accesses

Solution: *Cache translations in TLB*

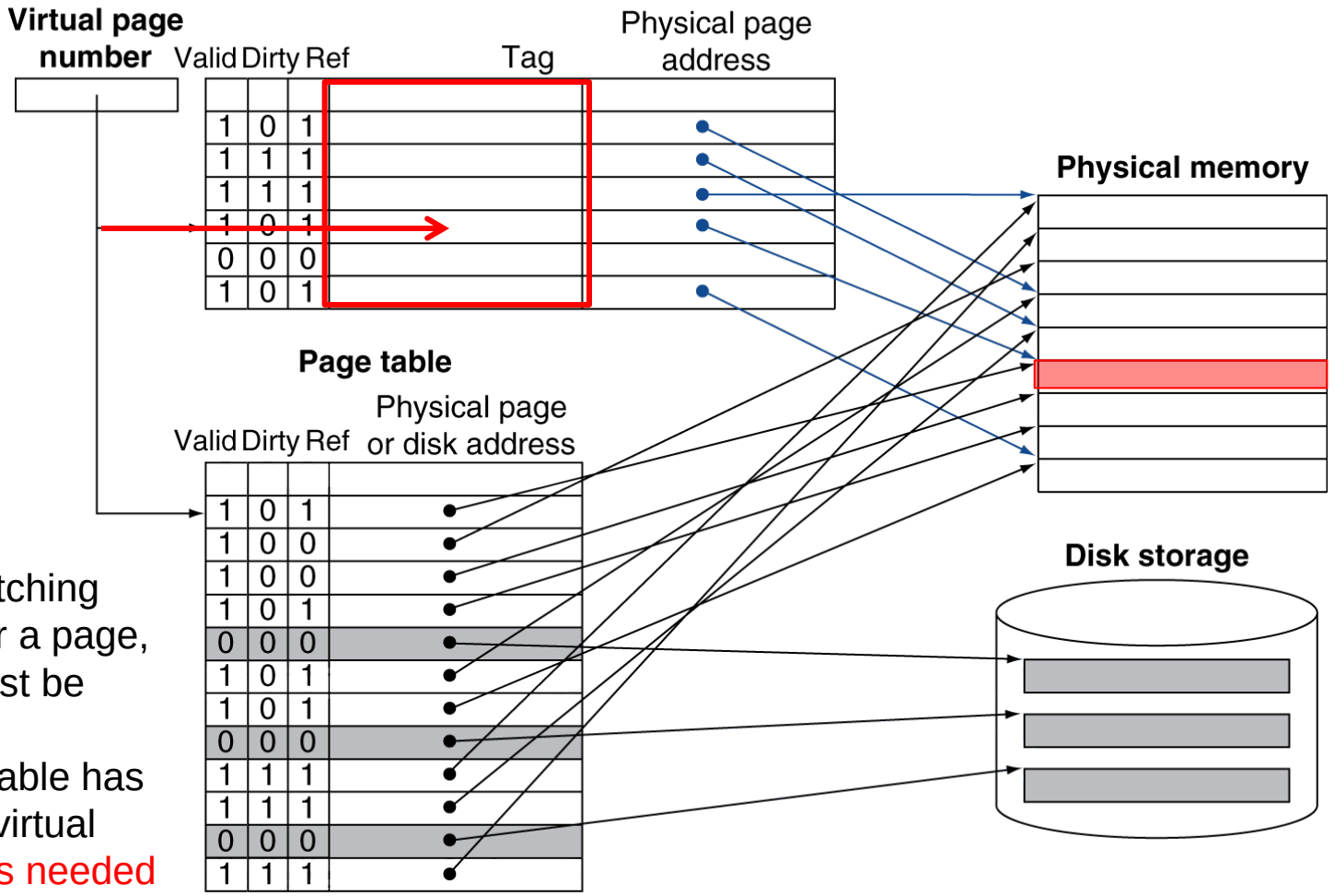
TLB hit ⇒ *Single-Cycle Translation*

TLB miss ⇒ *Page-Table Walk to refill TLB*



Fast Translation Using a TLB

TLB A cache that contains a subset of the page table



- If there is no matching entry in the TLB for a page, the page table must be examined.
- Since the page table has an entry for every virtual page, no tag field is needed