

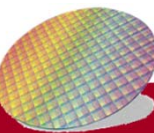


成功大學

National Cheng Kung University

Computer Organization

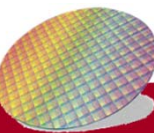
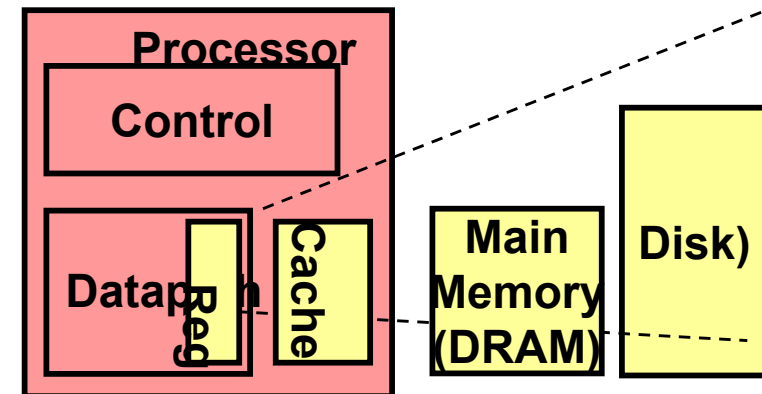
Virtual Memory



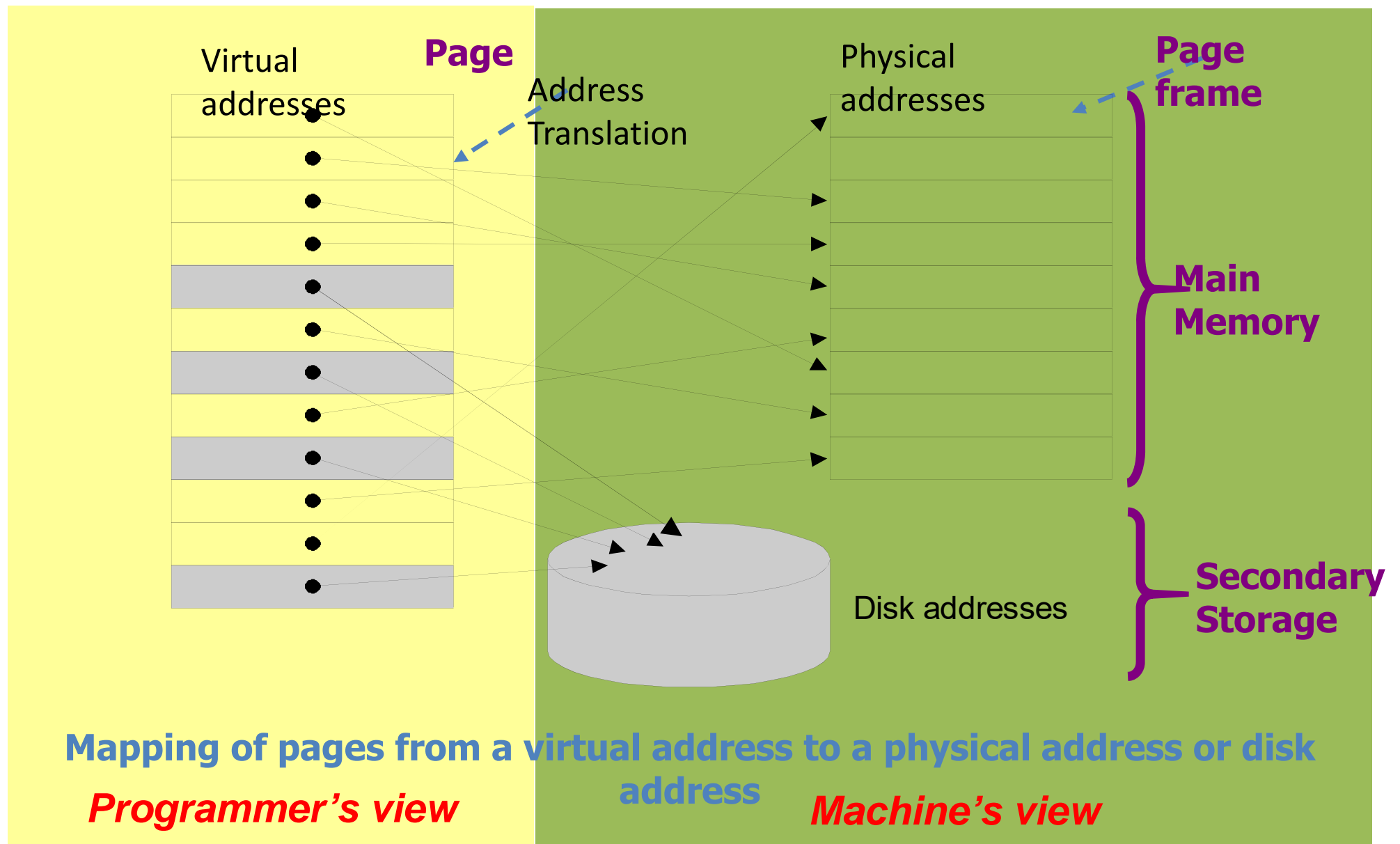


Virtual Memory (1/3)

- VM uses 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
 - **CPU** and **OS** translate virtual address to **physical** space
- VM can
 - allow **many programs** to run at once
 - **protected** from **other** programs



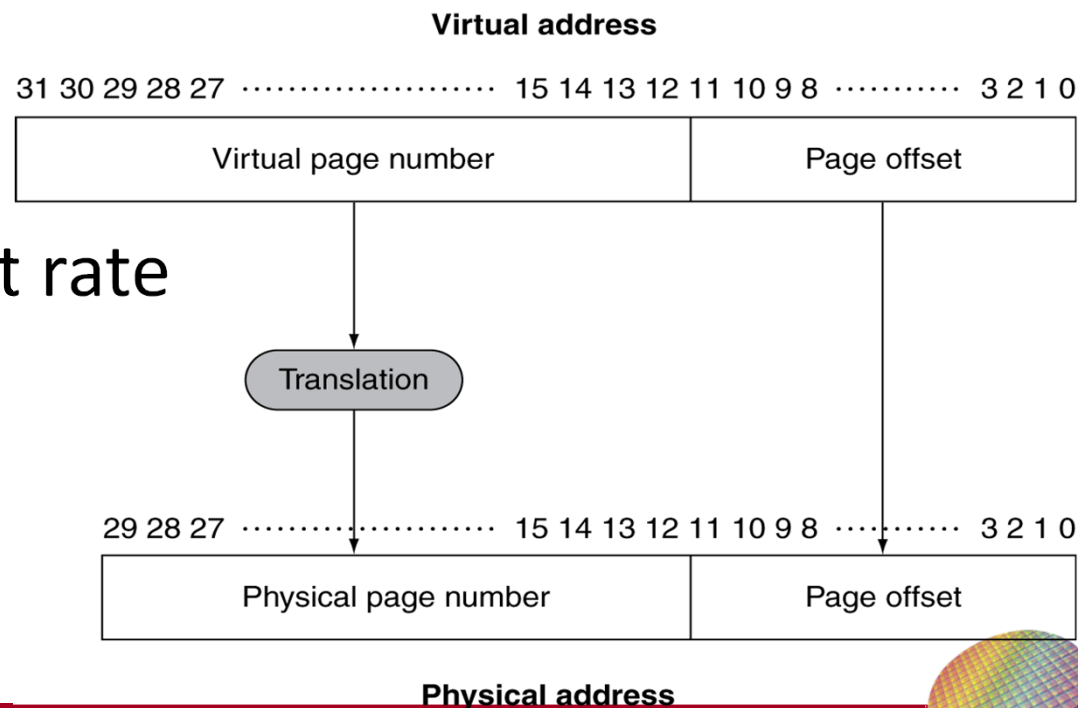
Virtual Memory (2/3)



Each program has its own memory space physical memory in system

Address Translation

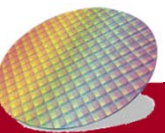
- A VM “**block**” is called a **page**, page size are normally fixed (e.g., 4K)
- **Virtual address** can be **larger** than physical address
 - Need to **translate** from virtual to physical address
- A VM translation “miss” is called a **page fault**
- On page fault, the page must be fetched from disk
 - Takes **millions** of clock cycles
 - Handled by **OS** code
- Methods to minimize page fault rate
 - **Fully associative** placement
 - **Smart replacement** algorithms





Page Tables

- Page Table stores **placement** information (see next slide)
 - Array of page table entries(PTE), **indexed** by virtual page number
 - **Page table register** in CPU points to a page table in **physical memory**
- If page is **present** in memory
 - PTE stores the **physical page** number and other **status** bits (**valid**, **referenced**, **dirty**, ...)
 - **Referenced bit** is used to indicate the page has been used before
- If page is not present
 - PTE can refer to location in swap space on disk



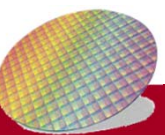
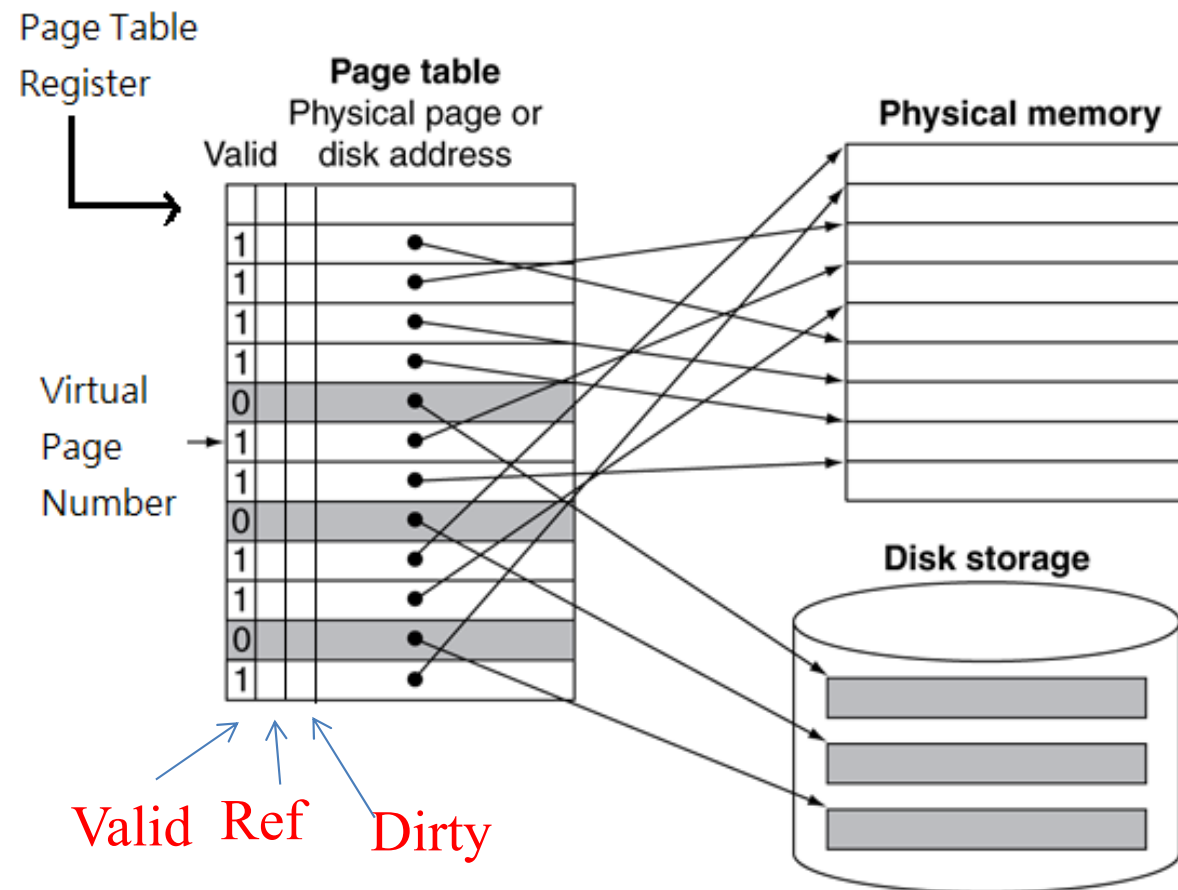


Mapping Pages to Storage

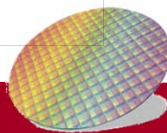
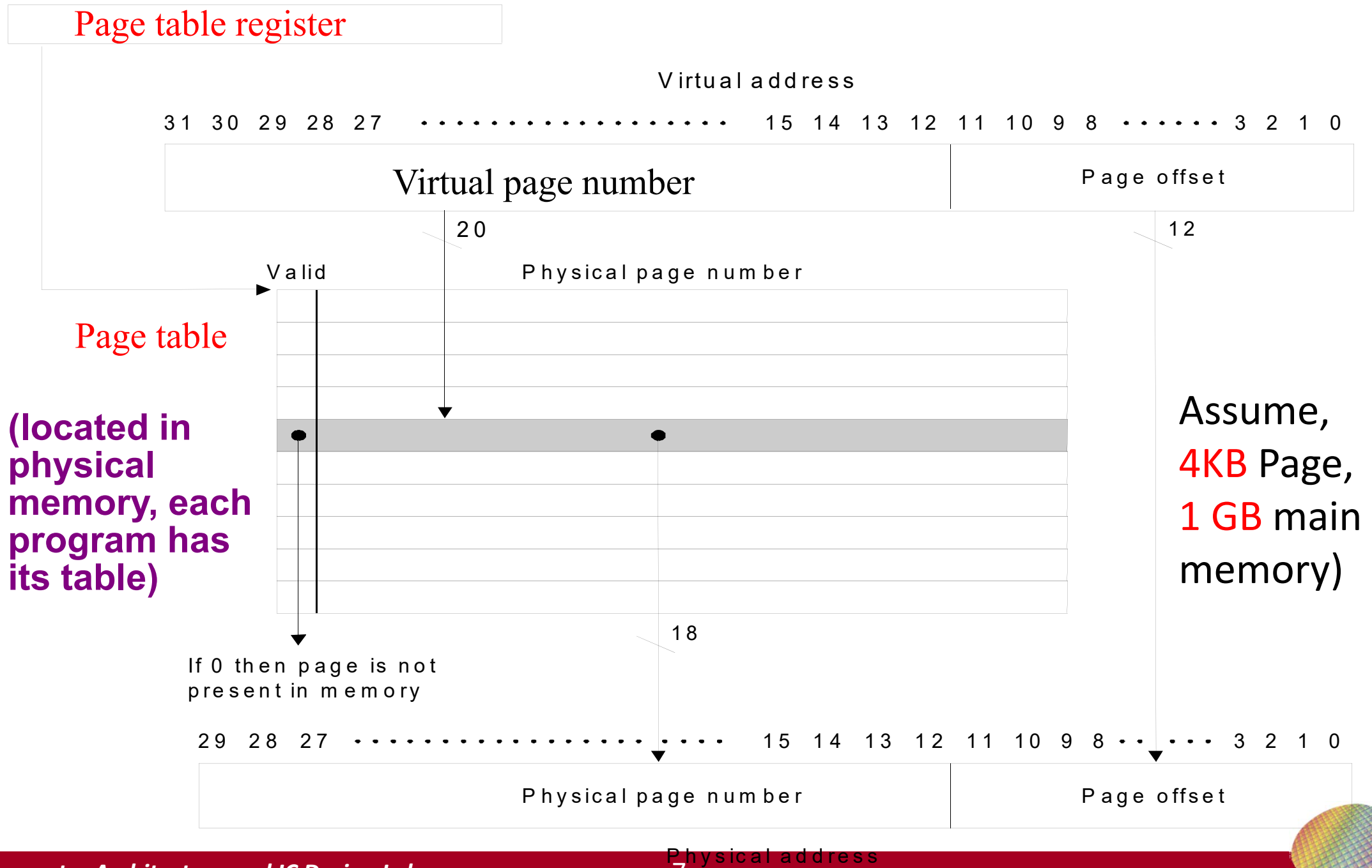
- Page table maps each page **in virtual memory** to either a page in **main memory** or a page stored on **disk**

Valid bit = 1
indicate the page
is Physical Mem

Ref bit = 1 indicate
the page is used
before

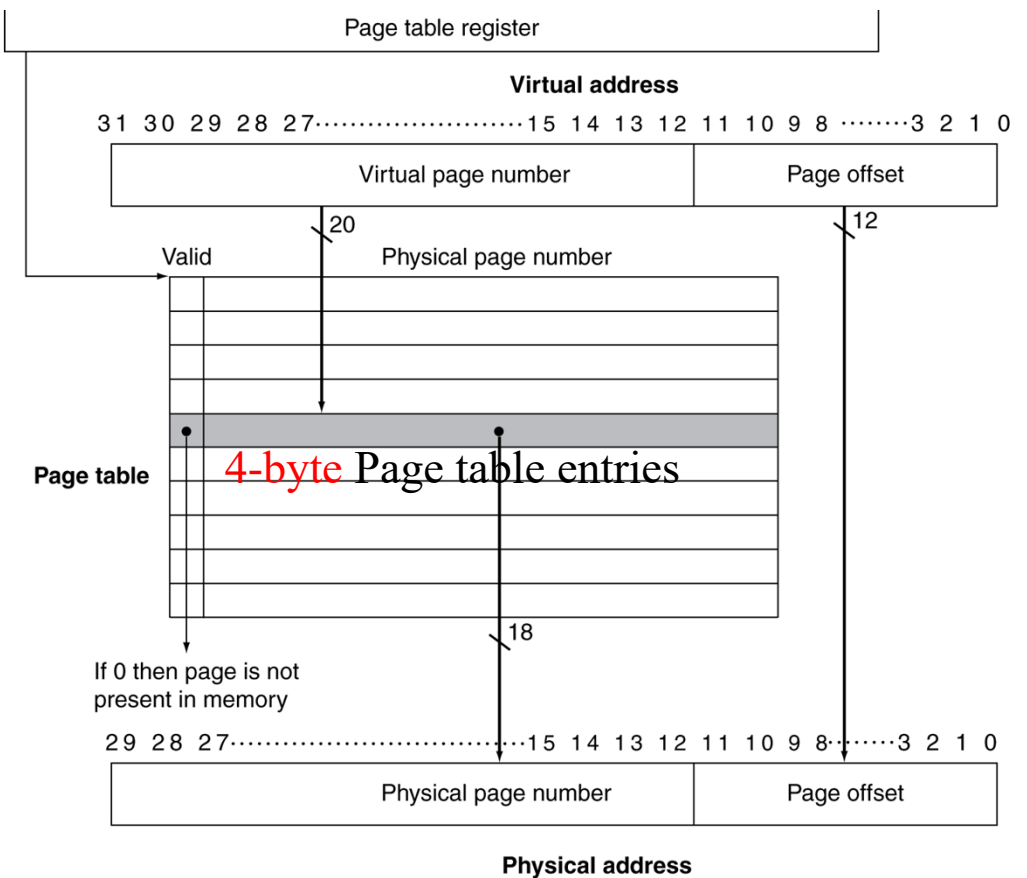


Translation Using a Page Table



Example 1

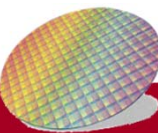
Assume 32-bit virtual address, each page is **4KB**, and each page table entry has **4 bytes**, what is the **total page table size**?



Each page has 4KB ($4\text{KB} = 2^{12}$) \Rightarrow Page offset has **12 bits**

Virtual page number has **20 bits** ($=32-12$) \Rightarrow No. of page table entries $= 2^{32}/2^{12}=2^{20}$

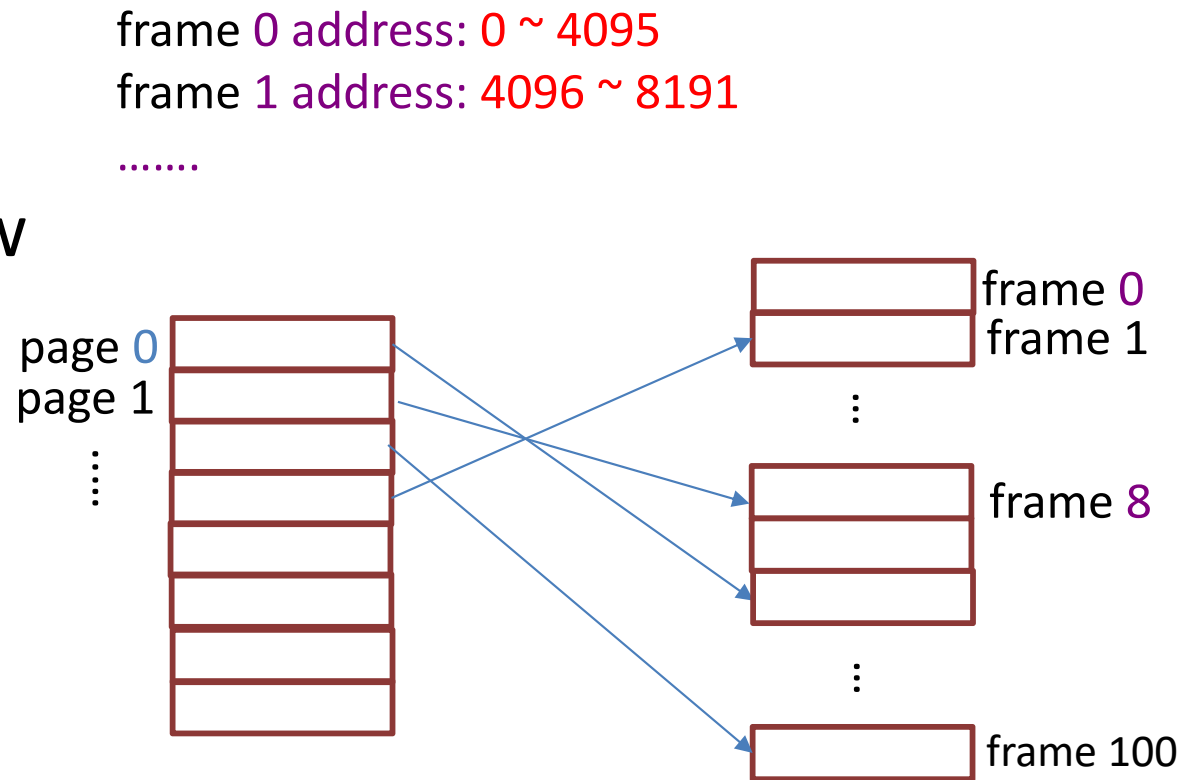
Size of page table = No. of entries \times entry size $= 2^{20} \times 4 \text{ bytes} = 4 \text{ MB}$



Example 2

Follow the above question, now the page table maps

page 0 to frame 10,
page 1 to frame 8,
page 2 to frame 100,
page 3 to frame 1..

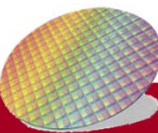


Q2: What is the **virtual page number** of the virtual address 8196?

$8196 / 4096 = 2 \dots 4$, therefore, address 8196 is at the 3rd page, virtual page number is **2**

Q3: What is the **physical address of** the virtual address 8196?

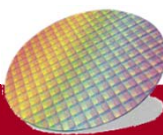
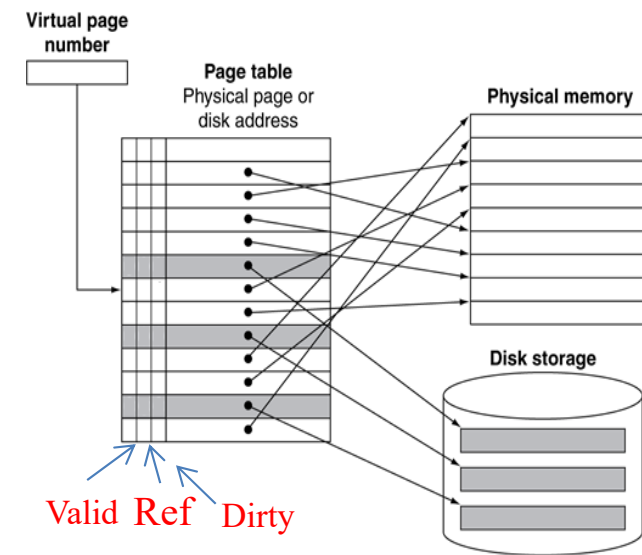
Virtual Page 2 mapped to physical address 100, Therefore, physical address is $100 * 4096 + 4$



What happens during a page fault?

Page fault means that page is **not** resident in memory, hardware must trap to the OS to obtain the required data

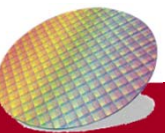
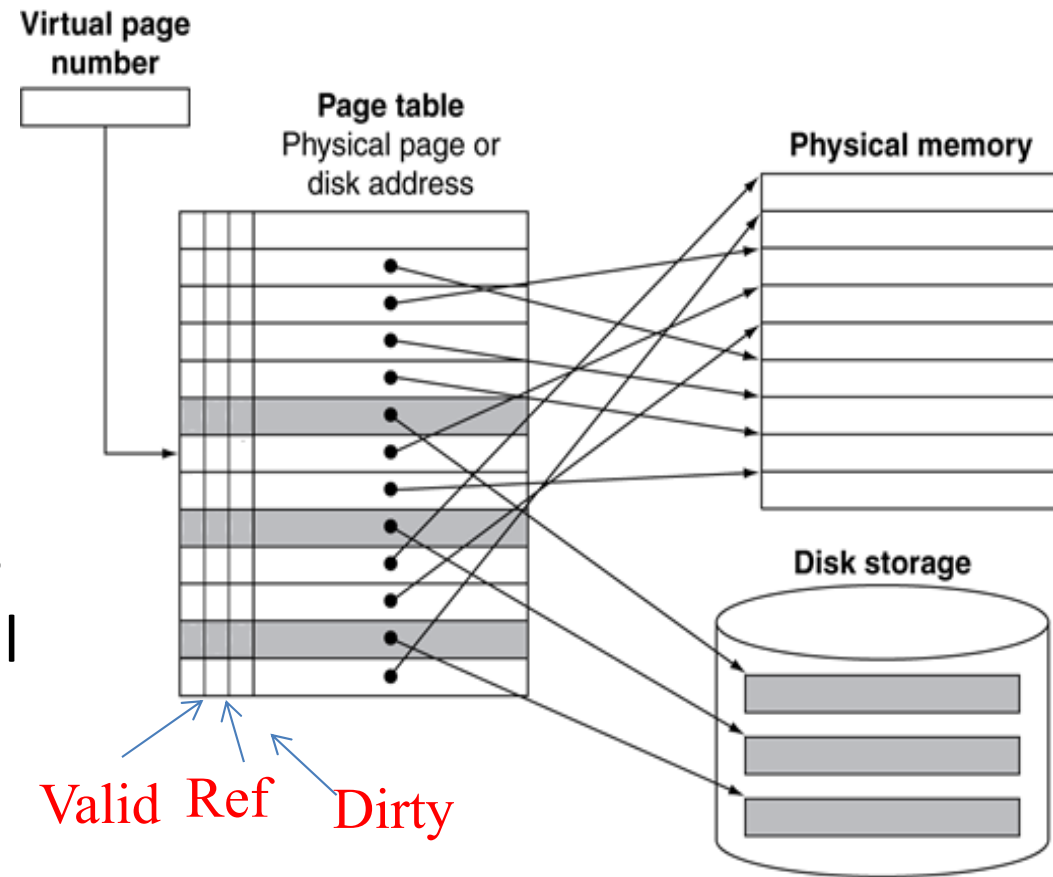
- “May” need to pick a page to discard if no free space in memory
 - i.e., **page replacement policy**
 - Note: if a page in “main memory” is updated, then it is “dirty”
- **Load** the requested page in **from disk**
 - i.e., **placement policy**
- **Update** the page table
- Resume to program so HW will retry and succeed!





Replacement and Writes

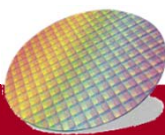
- To reduce page fault rate, prefer **least-recently used** (LRU) replacement
 - **Reference** bit (aka **use** bit) in page table set to 1 on access to page
 - Periodically cleared to **0** by OS
 - A page with **reference bit = 0** has not been used recently
- Disk writes take millions of cycles
 -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



Summary: Considerations for Virtual Memory

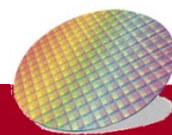
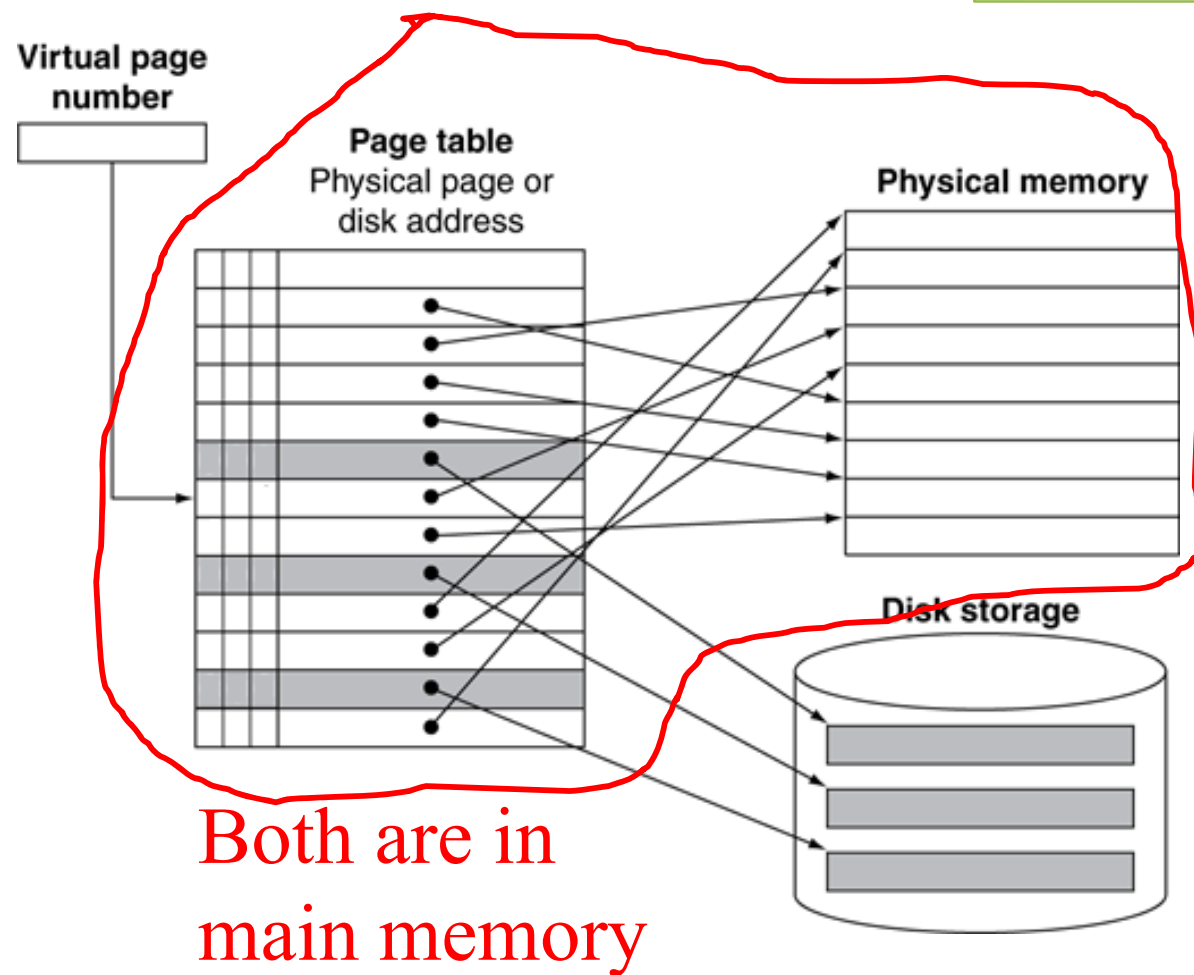


- Size of a page
Page: a basic unit of memory blocks transferring between main memory and secondary storage (e.g., disk)
- **Placement** policy
where to place a page in *main* memory (normally fully associative)
- **Replacement** policy
replace pages if *cannot find* any **free** space in main memory (normally LRU)
- When to fetch a page from disk?
fetch when page fault occurs



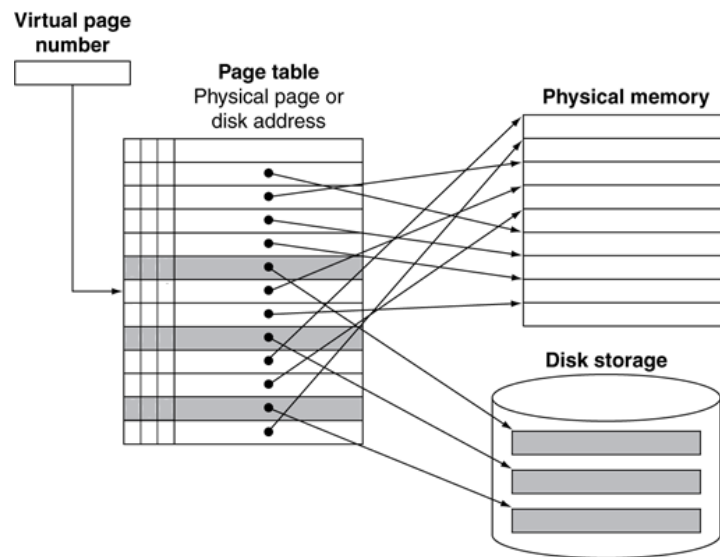
Problem of current technique

- Page table is also in the **main** memory
- Problem: every memory access by a program take at least **two memory access**
 - ⇒ One to obtain **physical address**
 - ⇒ One to get the data
 - ⇒ Time consuming
- Use **Translation Lookaside Buffer** (also called **translation cache**) to reduce memory accesses

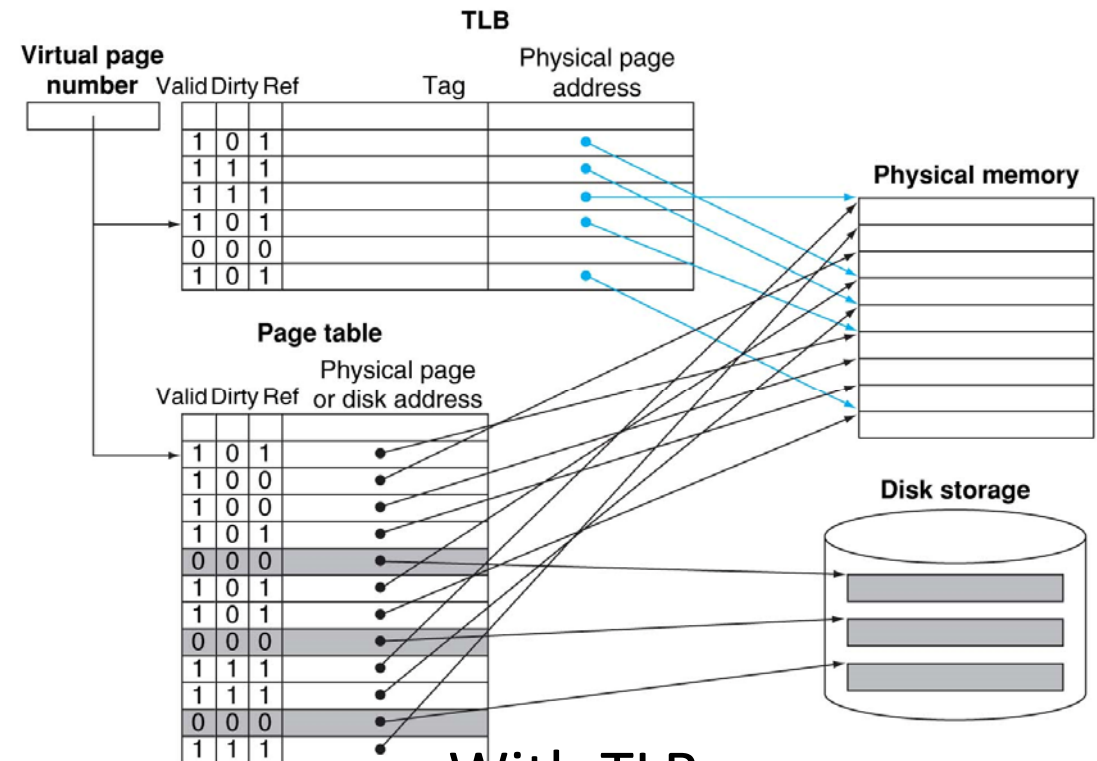


TLB to speedup the address translation

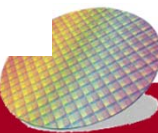
- TLB is “**cache**” of Page table : page table **entries** that are accessed frequently are put into TLB
- **Typically, it's hardware** inside a processor
- Either **fully** associative, **set** associative, or **direct** mapped
- TLBs are typically **small** in size, typically **< 128~256** entries



Without TLB



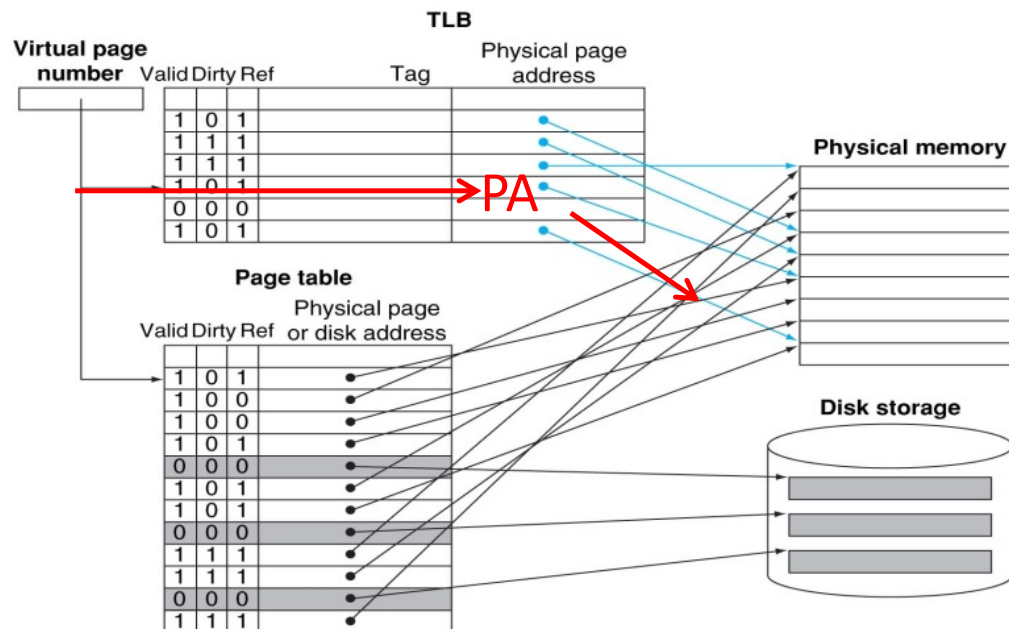
With TLB



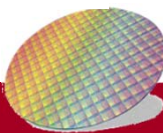
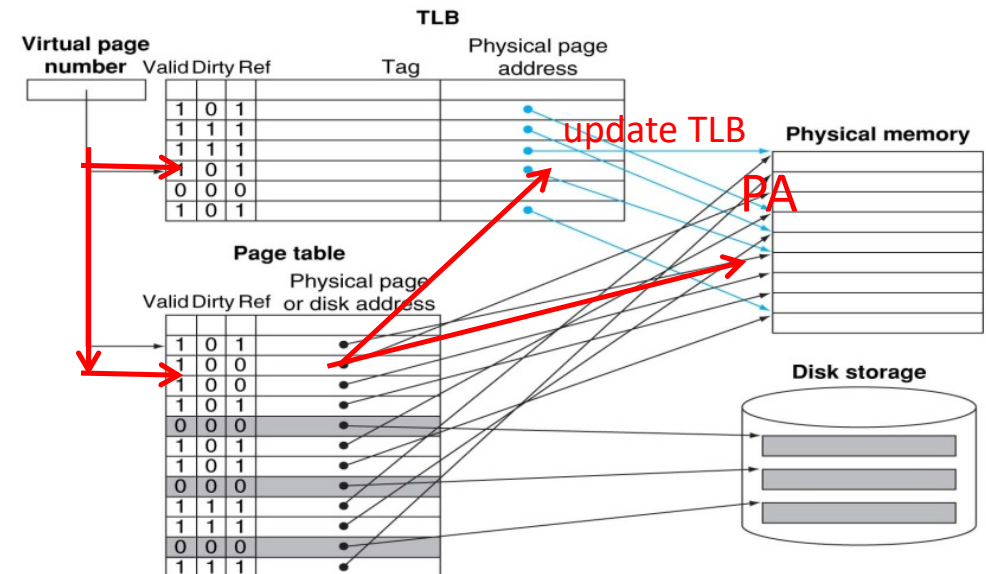
TLB hit and miss (1/2)

- TLB hit on read
 - Physical address is obtained
- TLB hit on write:
 - Toggle **dirty** bit (write **back** to page table on replacement)
- **TLB miss and Page table hit**
 - load “page table entry” into TLB,
 - Then restart the instruction

TLB Hit

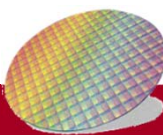
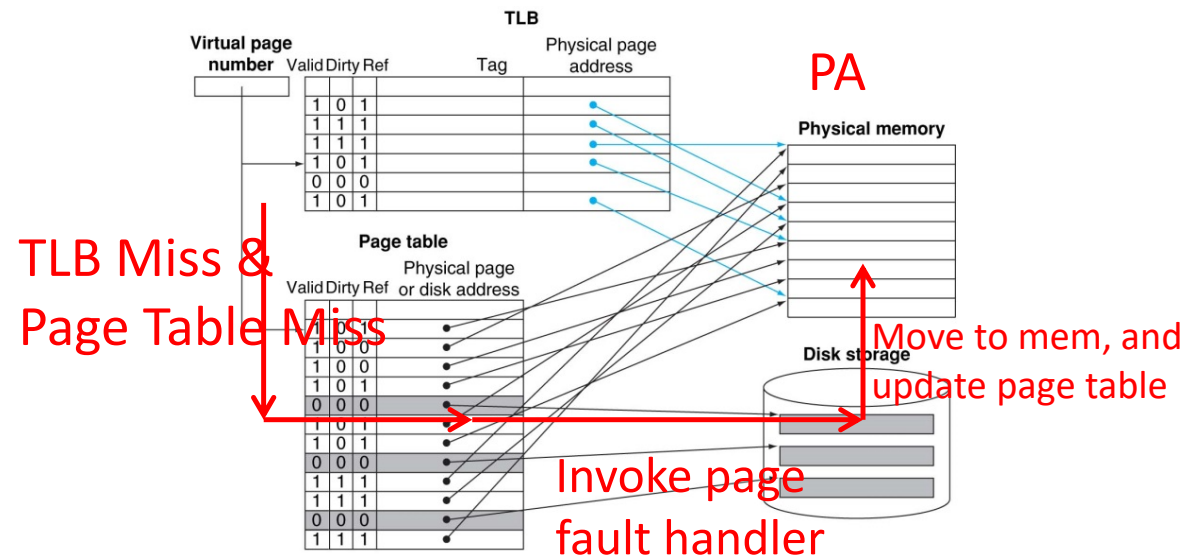


TLB Miss & Page Table Hit



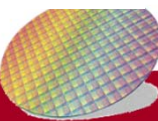
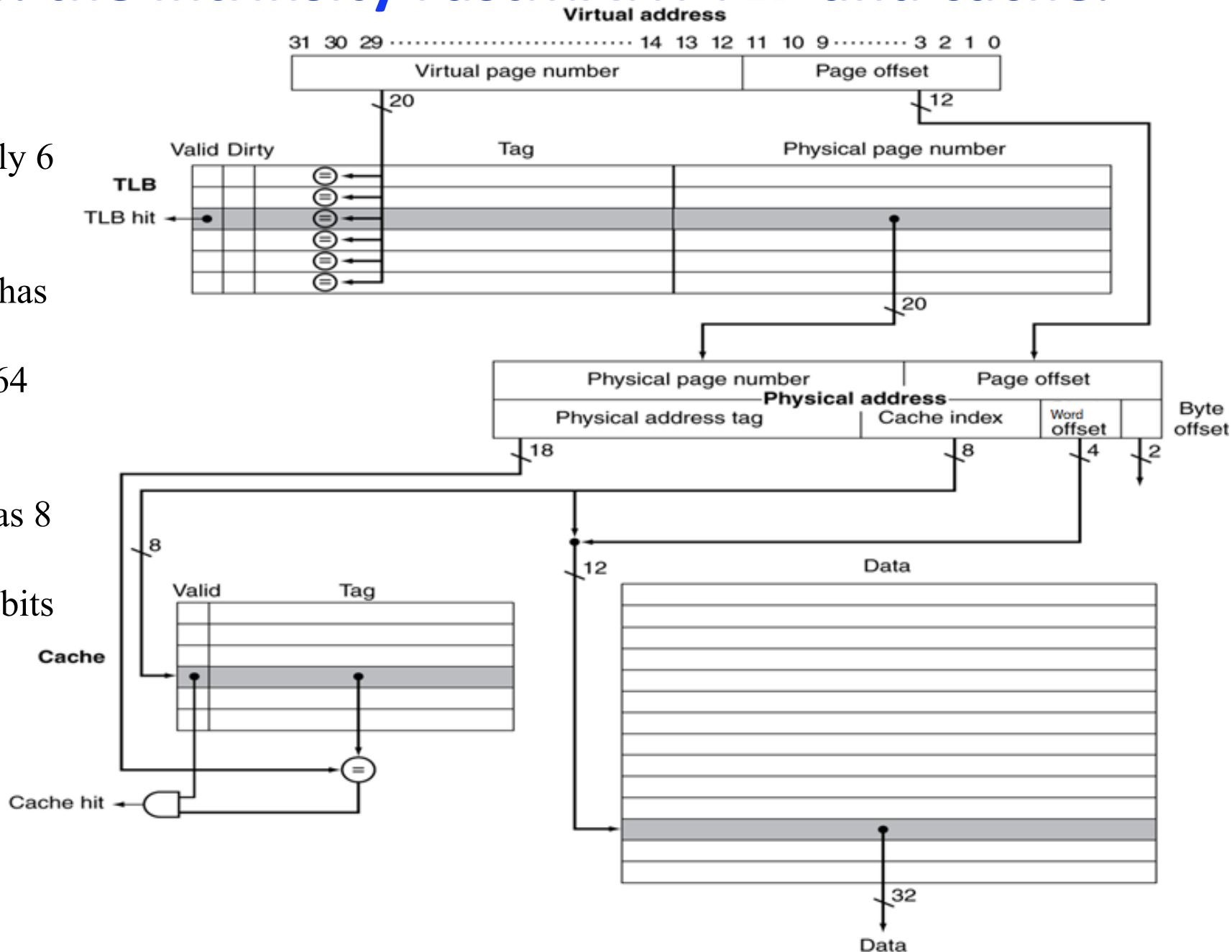
TLB hit and miss (2/2)

- TLB miss and page table miss (**page fault**)
 - OS handles fetching the page and updating the page table,
 - Then restart the faulting instruction

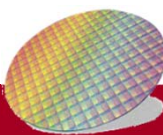
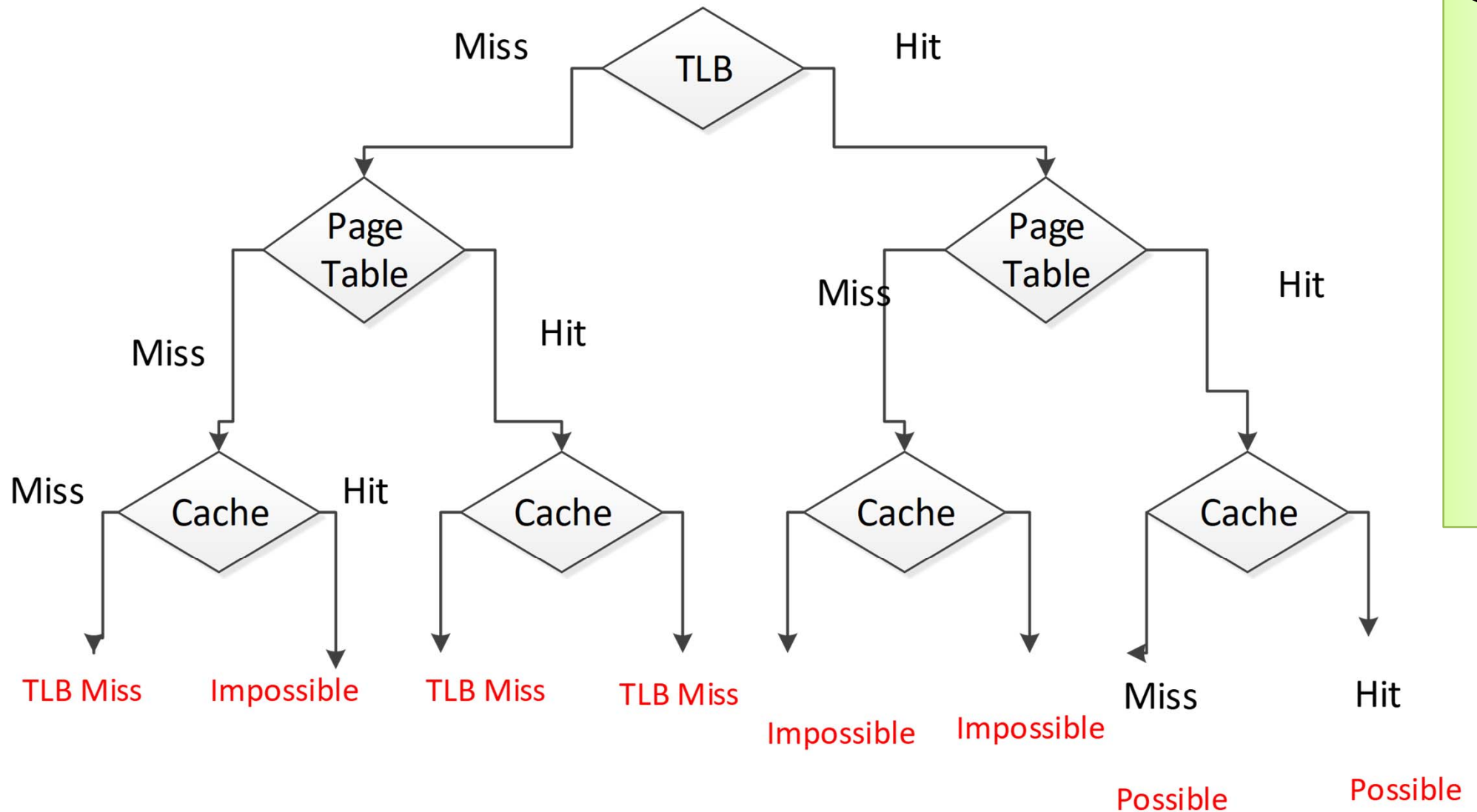


Example: the Intrinsity FastMATH TLB and cache.

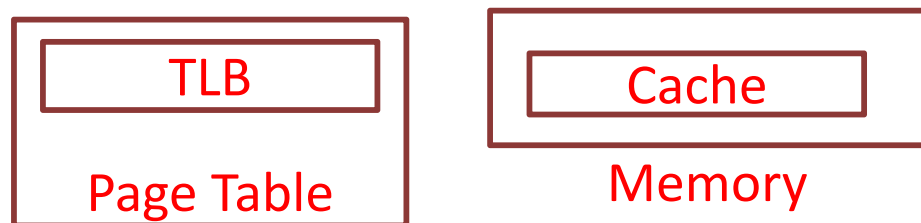
- 32-bit V.A. (4GB)
- 32-bit P.A. (4GB)
- TLB entries = **16** (only 6 entries are shown)
- Page size : **4KiB**
- Virtual page number has 20 bit (=32-12)
- 16 words per block (64 bytes)
- 256 Blocks (index has 8 bits)
- Tag has $32 - 8 - 6 = 18$ bits



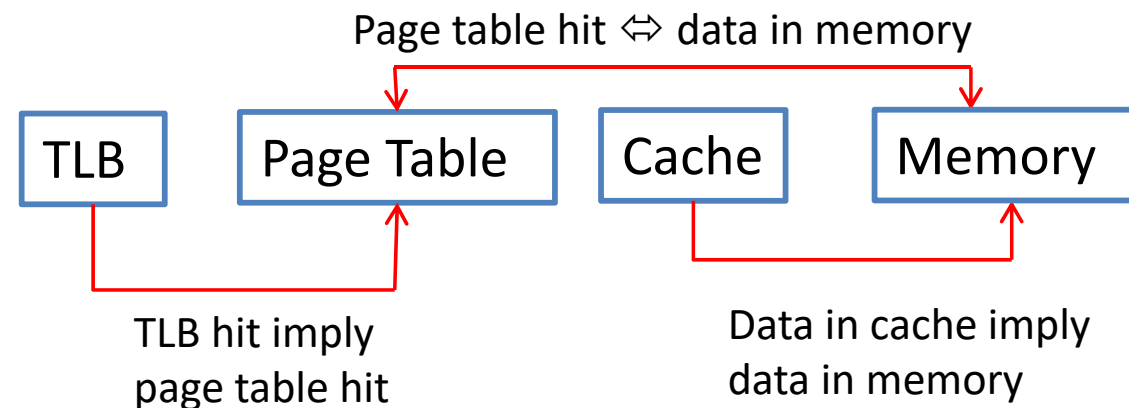
Overall operation of Memory Hierarchy



Overall operation of Memory Hierarchy



TLB data is subset of Page table,
Cache data is subset if memory

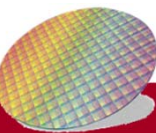


TLB	Page Table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Hit	Possible. Best case
Hit	Hit	Miss	Possible, but when TLB hits, the page table is not checked
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory (TLB is subset of page table)
Hit	Miss	Miss	
Miss	Hit	Hit	TLB misses, but entry found in the page table; after retry, data is found in cache
Miss	Hit	Miss	TLB misses, but entry found in the page table; after retry, data misses in cache
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if page is not in memory
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry data must miss in cache



The Memory Hierarchy

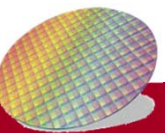
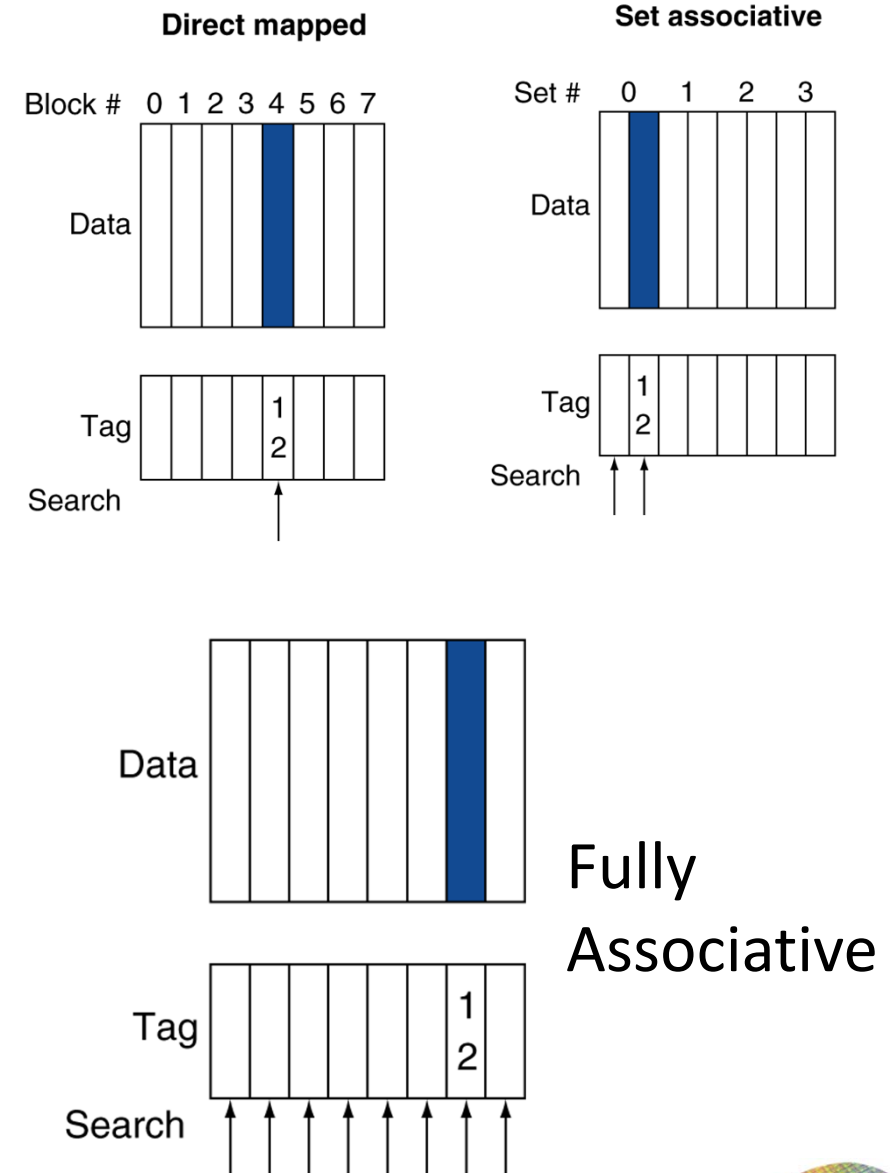
- Common **principles** apply at all levels of the memory hierarchy
- At each level in the hierarchy, we need to consider
 - Block **placement**
 - **Finding** a block
 - **Replacement** on a miss
 - **Write** policy



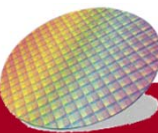
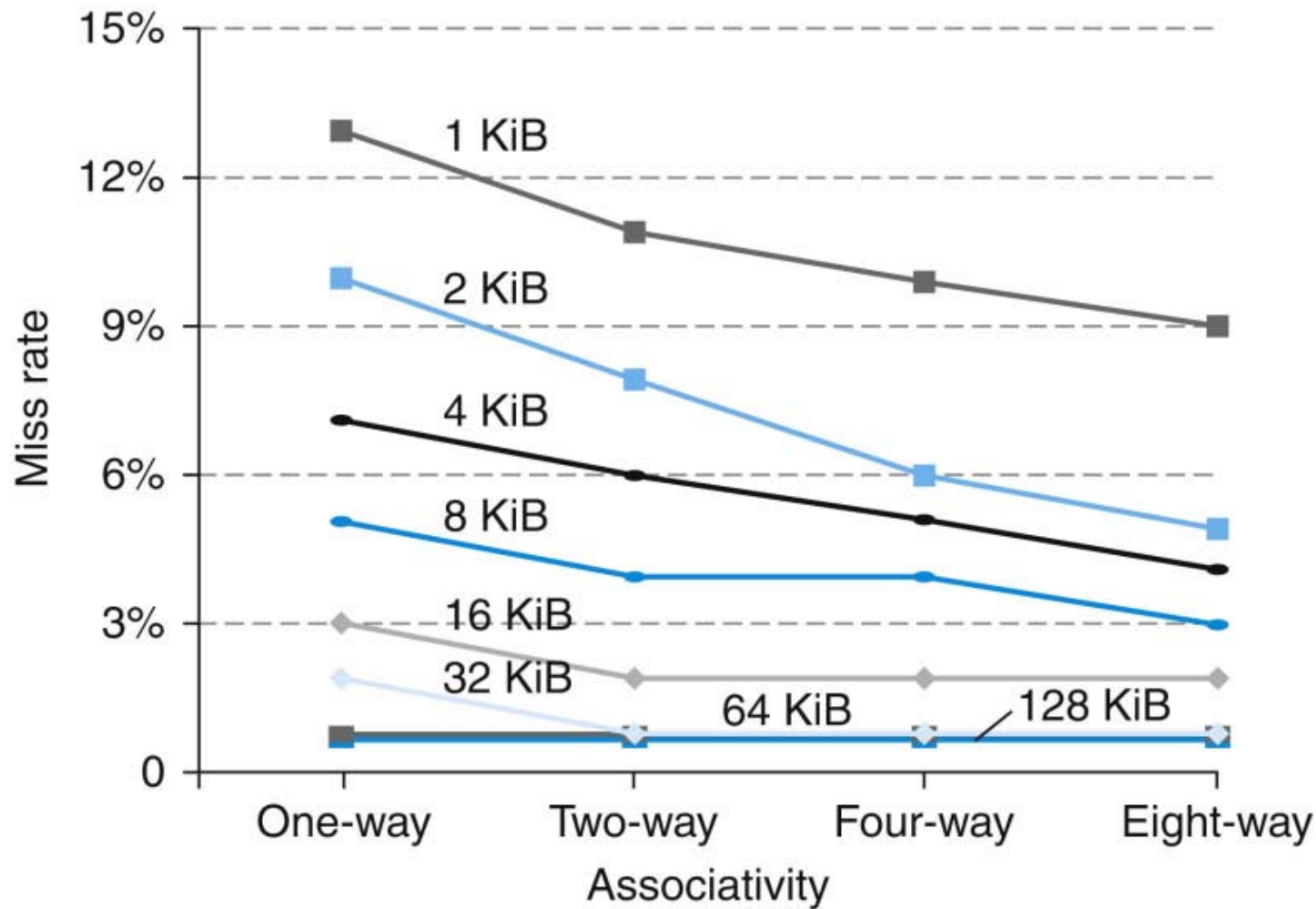


Block Placement

- Determined by associativity
 - Direct mapped (1-way associative)
 - One choice for placement
 - n-way set associative
 - n choices within a set
 - Fully associative
 - Any location
- Higher associativity reduces miss rate
 - Also Increases complexity, cost, and access time



Miss Rate vs. Associativity for different cache sizes

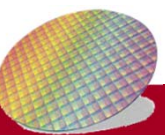




Finding a Block

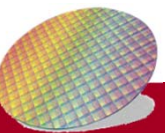
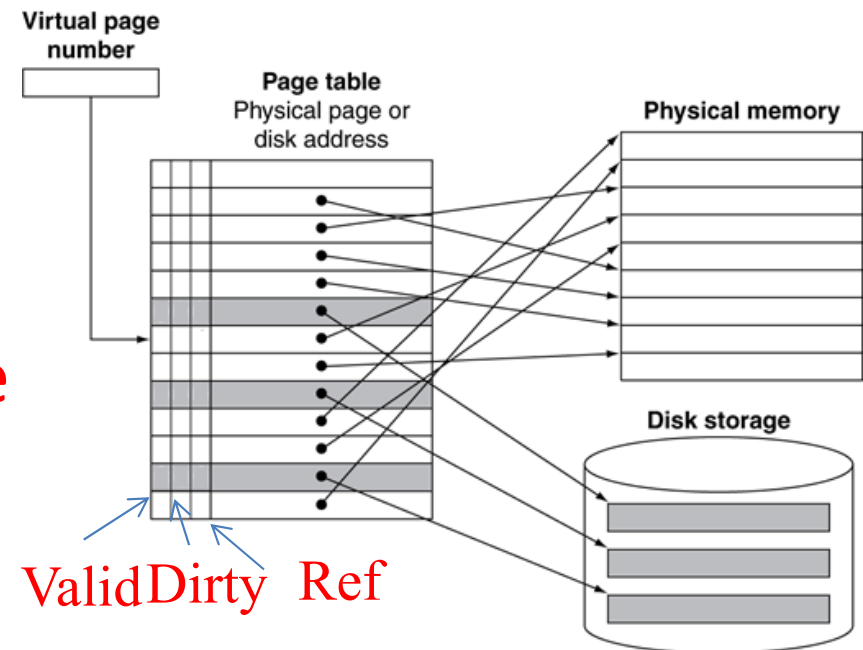
Associativity	Location method	Tag comparisons
Direct mapped	Index	1
n-way set associative	Set index, then search entries within the set	n
Fully associative	Search all entries	#entries
	Full lookup table	0

- Caches (using hardware)
 - Reduce **comparisons** to reduce **cost**
 - **Cache is normally set-associative or direct mapped**
- Virtual memory
 - **Reducing miss rate is first priority**
 - **Full table lookup (page table)** makes full associativity feasible
 - Therefore, VM is almost **fully associative**



Replacement

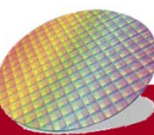
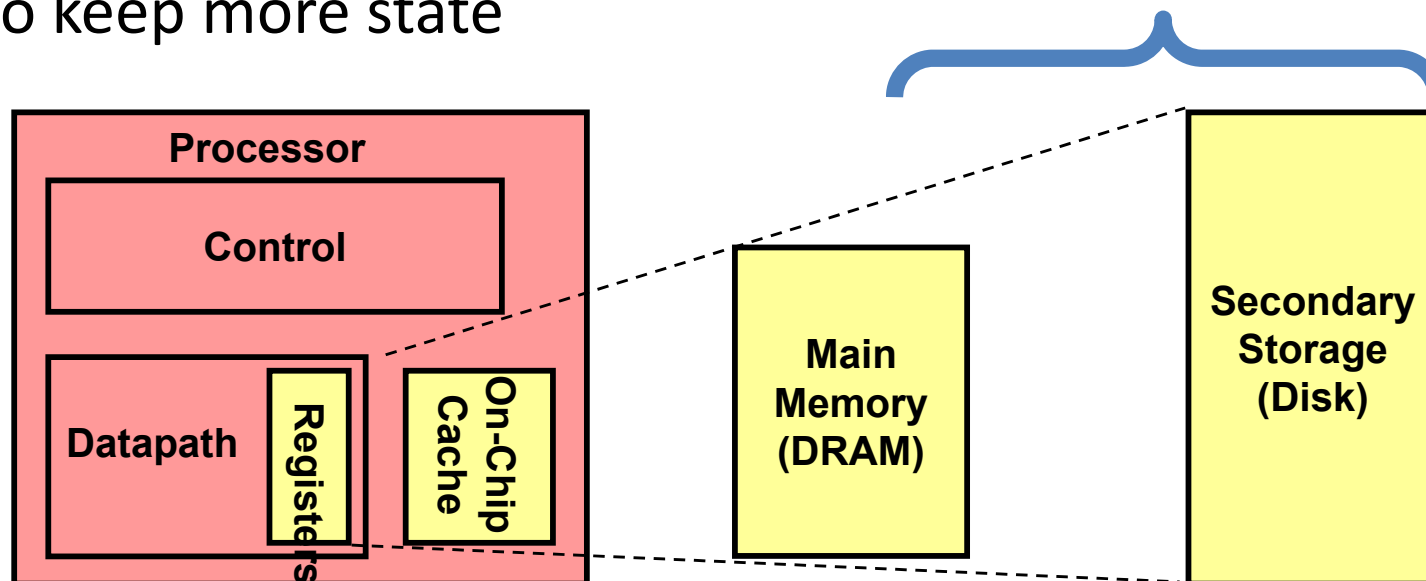
- Choice of entry to replace on a miss
 - Least recently used (LRU)
 - Complex and costly hardware for high associativity
 - Random
 - easier to implement
- Virtual memory
 - LRU approximation with hardware support





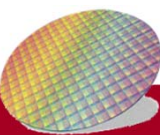
Write Policy

- **Write-through**
 - Update both upper and lower levels. It simplifies replacement, but may require **write** buffer
- **Write-back**
 - Update upper level only
 - Update lower level when block is replaced
 - Need to keep more state
- Between cache and main memory
 - **Either write-through and write-back are possible**
- **Virtual** memory
 - Only **write-back** is feasible, given disk write latency



Sources of Misses

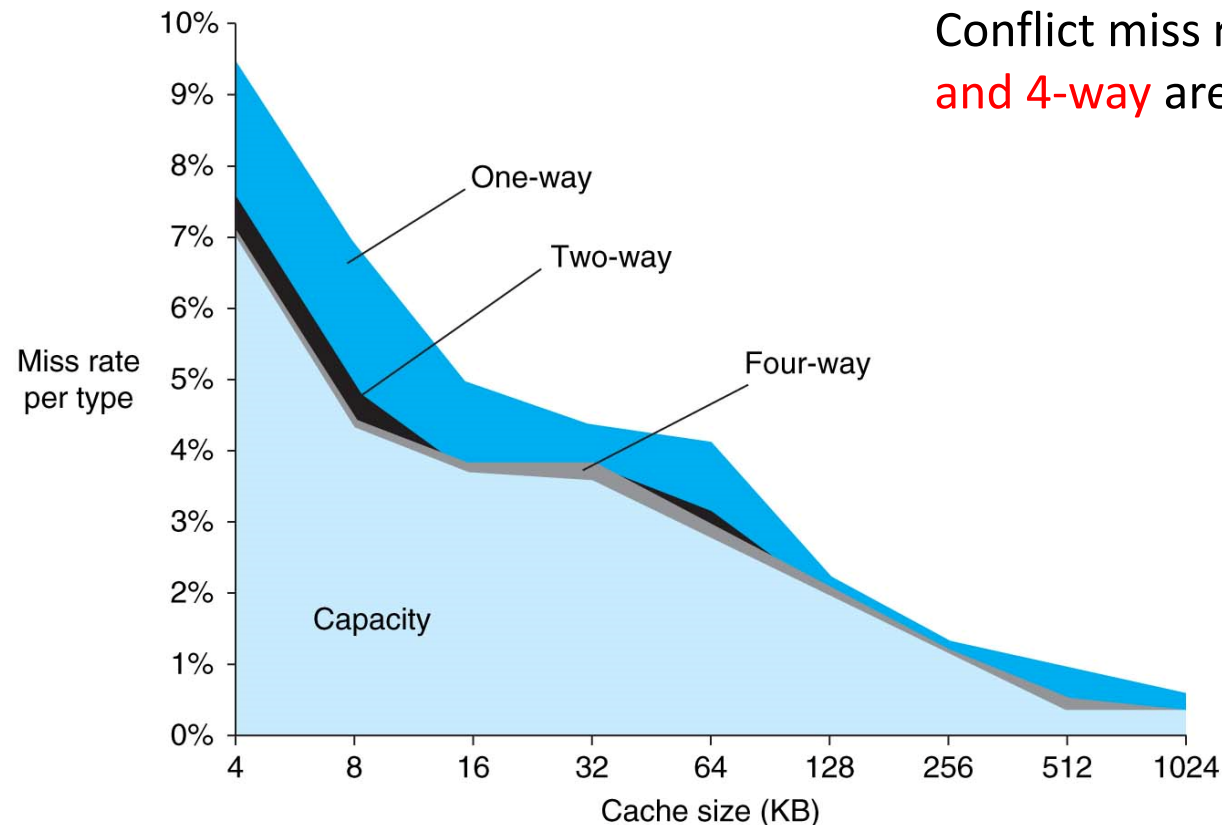
- **Compulsory** misses (aka cold start misses)
 - First access to a block
- **Capacity** misses
 - Due to finite cache size
 - A replaced block is later accessed again
- **Conflict** misses (aka collision misses)
 - In a **non-fully** associative cache
 - Due to competition for entries in a set
 - Would not occur in a fully associative cache of the same total size



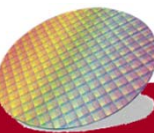


Miss rate

- **Compulsory** miss rate 0.006%, which is not shown
- Capacity miss is normally larger than conflict miss
- Capacity miss rate normally decrease when cache size increase

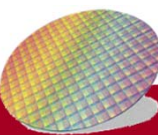


Conflict miss rates for **one-way, 2-way** and **4-way** are shown



Cache Design Trade-offs

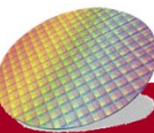
Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to larger block sizes and smaller number of blocks (a.k.a. pollution)





Concluding Remarks

- Fast memories are small, large memories are slow
 - We really want fast, large memories ☹
 - Caching gives this illusion ☺
- Principle of locality
 - Programs use a small part of their memory space frequently
- Memory hierarchy
 - L1 cache \leftrightarrow L2 cache \leftrightarrow ... \leftrightarrow DRAM memory \leftrightarrow disk
- Memory system design is critical for multiprocessors





成功大學

National Cheng Kung University

Backup

