DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Computer Architecture

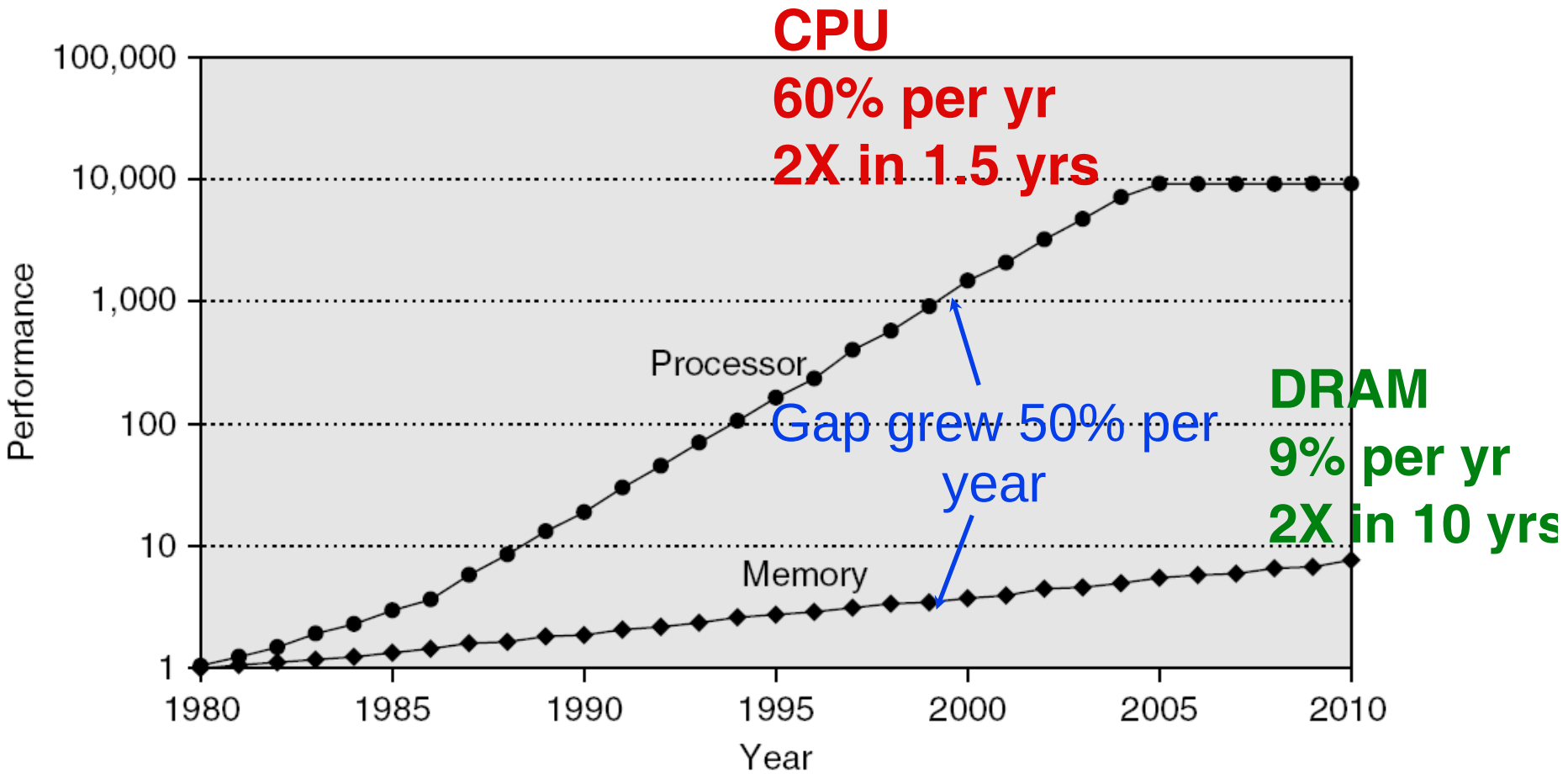
Lecture 3: Memory Hierarchy Design (Chapter 2, Appendix B)

Chih-Wei Liu 劉志尉

National Chiao Tung University

cwliu@twins.ee.nctu.edu.tw

Since 1980, CPU has outpaced DRAM…



Introduction

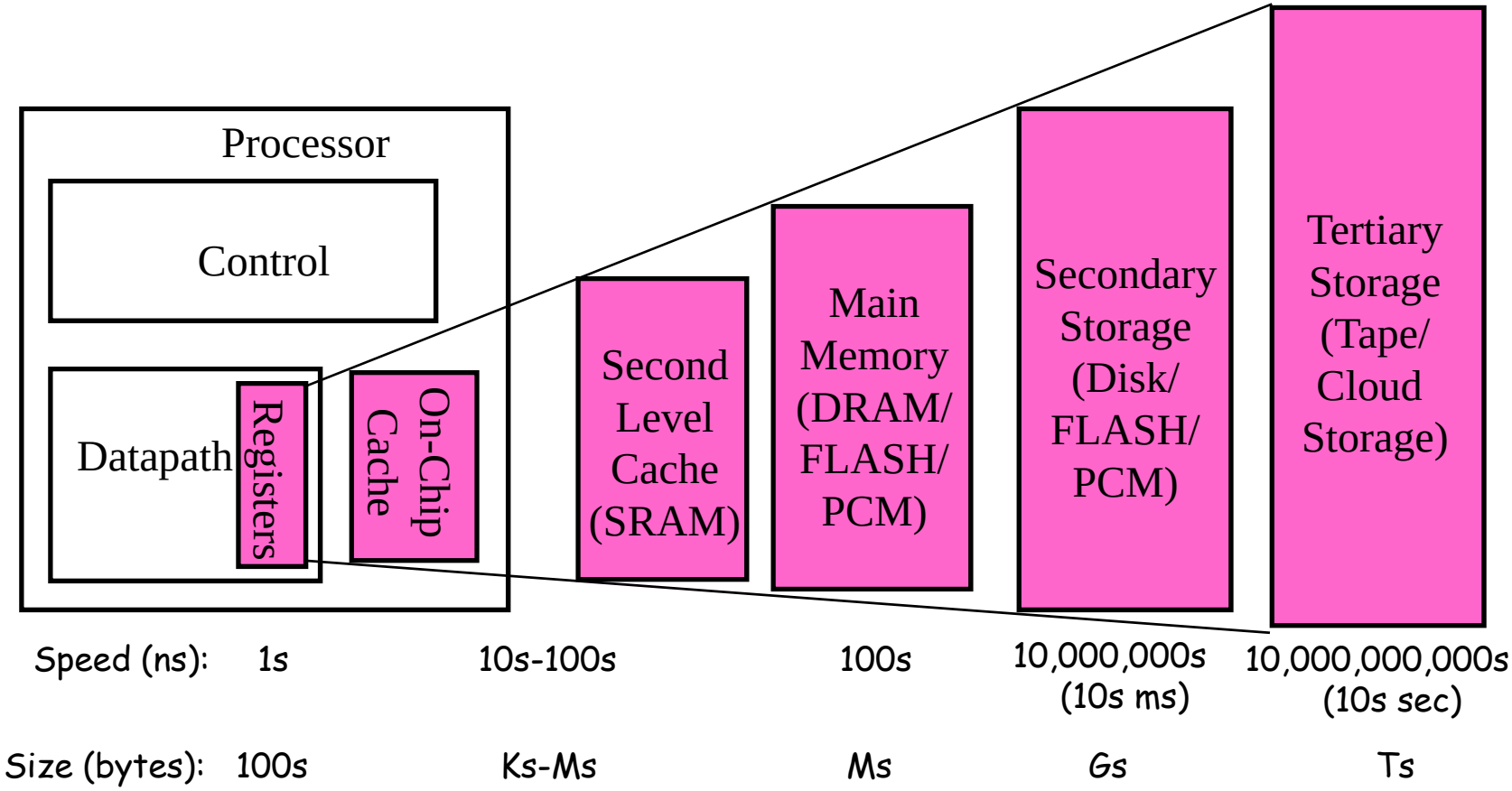
- Programmers want unlimited amounts of memory with low latency
- Fast memory technology is more expensive per bit than slower memory
- Solution: organize memory system into a hierarchy
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller and faster memories, each containing a subset of the memory below it, proceed in steps up toward the processor
- Temporal and spatial locality insures that nearly all references can be found in smaller memories
  - Gives the allusion of a large, fast memory being presented to the processor

Memory Hierarchy Design

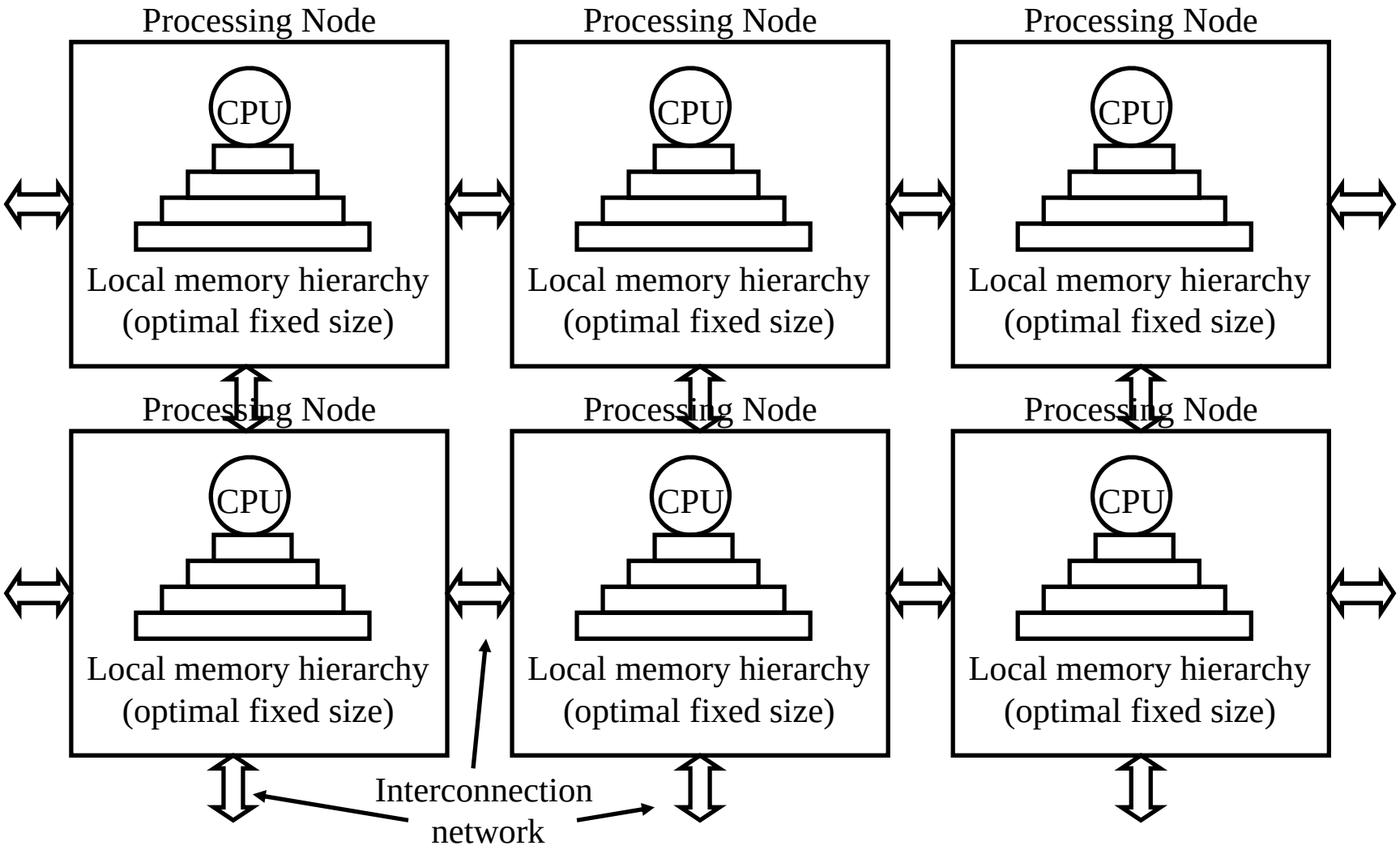
- Memory hierarchy design becomes more crucial with recent multi-core processors:
  - Aggregate peak bandwidth grows with # cores:
    - Intel Core i7 can generate two references per core per clock
    - Four cores and 3.2 GHz clock
      - 25.6 billion 64-bit data references/second +
        12.8 billion 128-bit instruction references
        = 409.6 GB/s!
    - DRAM bandwidth is only 6% of this (25 GB/s)
    - Requires:
      - Multi-port, pipelined caches
      - Two levels of cache per core
      - Shared third-level cache on chip

Memory Hierarchy

- Take advantage of the principle of locality to:
  - Present as much memory as in the cheapest technology
  - Provide access at speed offered by the fastest technology



Multi-core Architecture



The Principle of Locality

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)

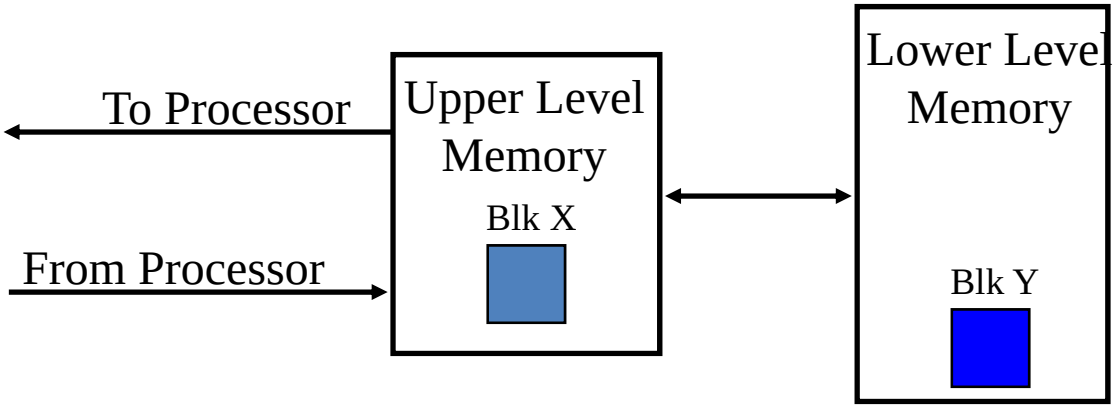
- HW relied on locality for speed

Memory Hierarchy Basics

- When a word is not found in the cache, a *miss* occurs:
  - Fetch word from lower level in hierarchy, requiring a higher latency reference
  - Lower level may be another cache or the main memory
  - Also fetch the other words contained within the *block*
    - Takes advantage of spatial locality
  - Place block into cache in any location within its *set*, determined by address
    - block address MOD number of sets

Hit and Miss

- Hit: data appears in some block in the upper level (e.g.: Block X)
  - Hit Rate: the fraction of memory access found in the upper level
  - Hit Time: Time to access the upper level which consists of
    RAM access time + Time to determine hit/miss
- Miss: data needs to be retrieve from a block in the lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level +
    Time to deliver the block the processor
- Hit Time << Miss Penalty (500 instructions on 21264!)

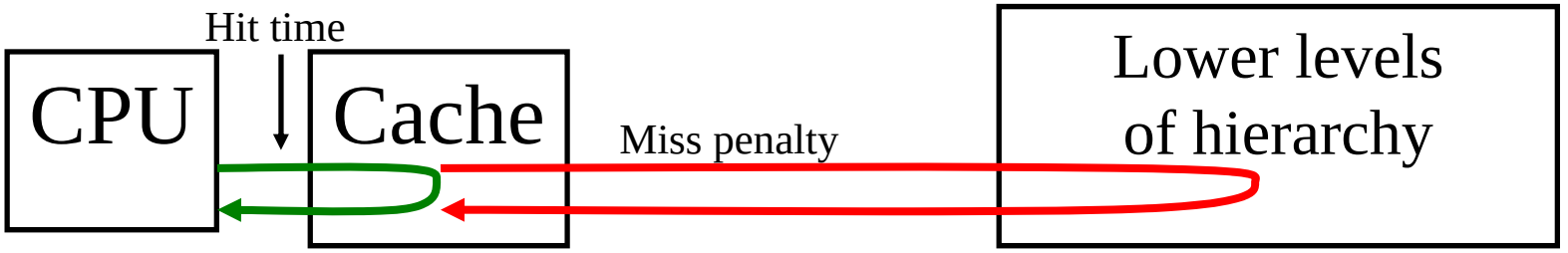


Cache Performance Formulas

(Average memory access time) =
(Hit time) + (Miss rate)×(Miss penalty)

$$\overline{T_{acc}} = T_{hit} + f_{miss} T_{+miss}$$

- The times T_{acc}, T_{hit}, and T_{+miss} can be all either:
  - Real time (*e.g.*, nanoseconds)
  - Or, number of clock cycles
    - In contexts where cycle time is known to be a constant
- Important:
  - T_{+miss} means the extra (not total) time for a miss
    - in *addition* to T_{hit}, which is incurred by all accesses

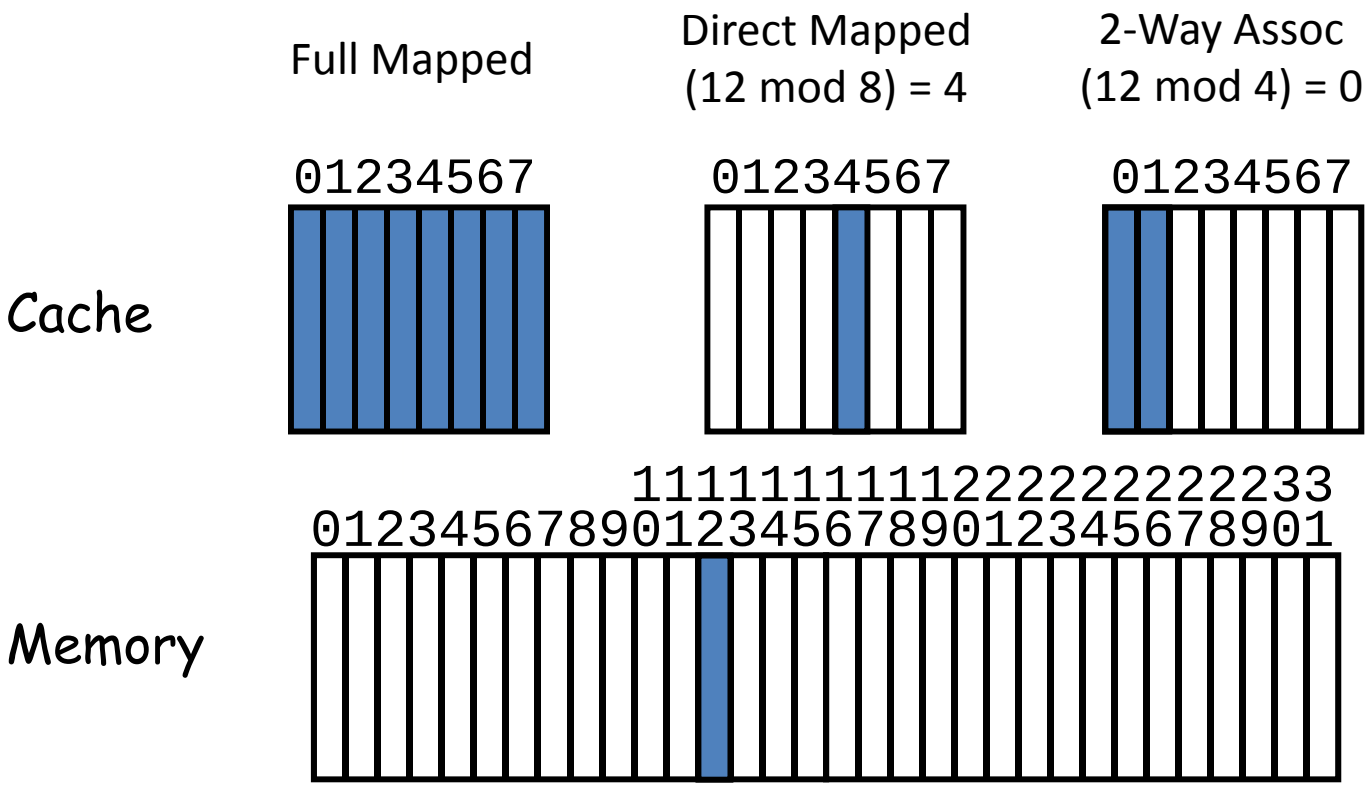


Four Questions for Memory Hierarchy

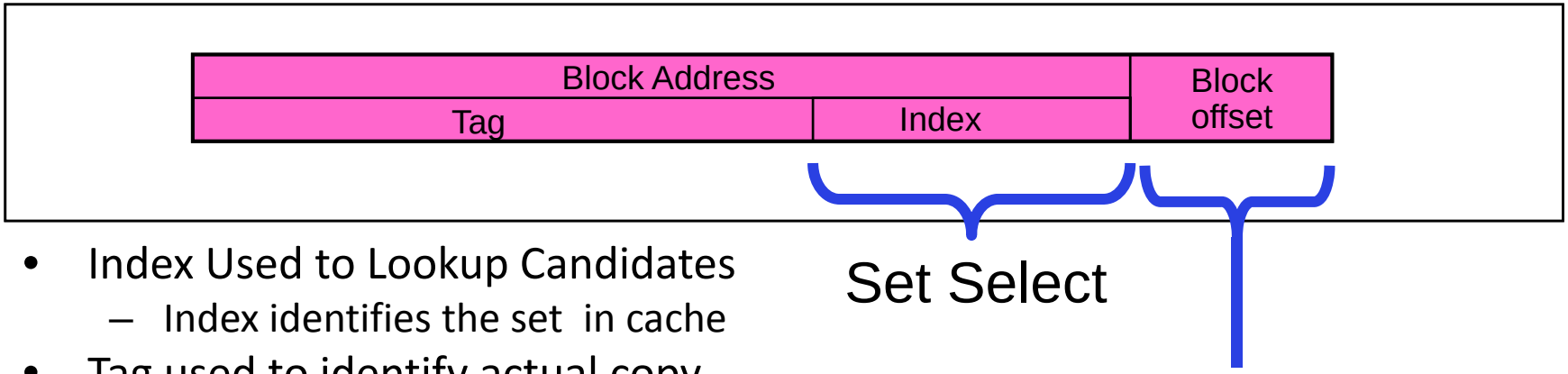
- Consider any level in a memory hierarchy.
  - Remember a block is the unit of data transfer.
    - Between the given level, and the levels below it
- The level design is described by four behaviors:
  - Block Placement*:*
    - Where could a new block be placed in the level?
  - Block Identification*:*
    - How is a block found if it is in the level?
  - Block Replacement*:*
    - Which existing block should be replaced if necessary?
  - Write Strategy*:*
    - How are writes to the block handled?

Q1: Where can a block be placed in the upper level?

- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets



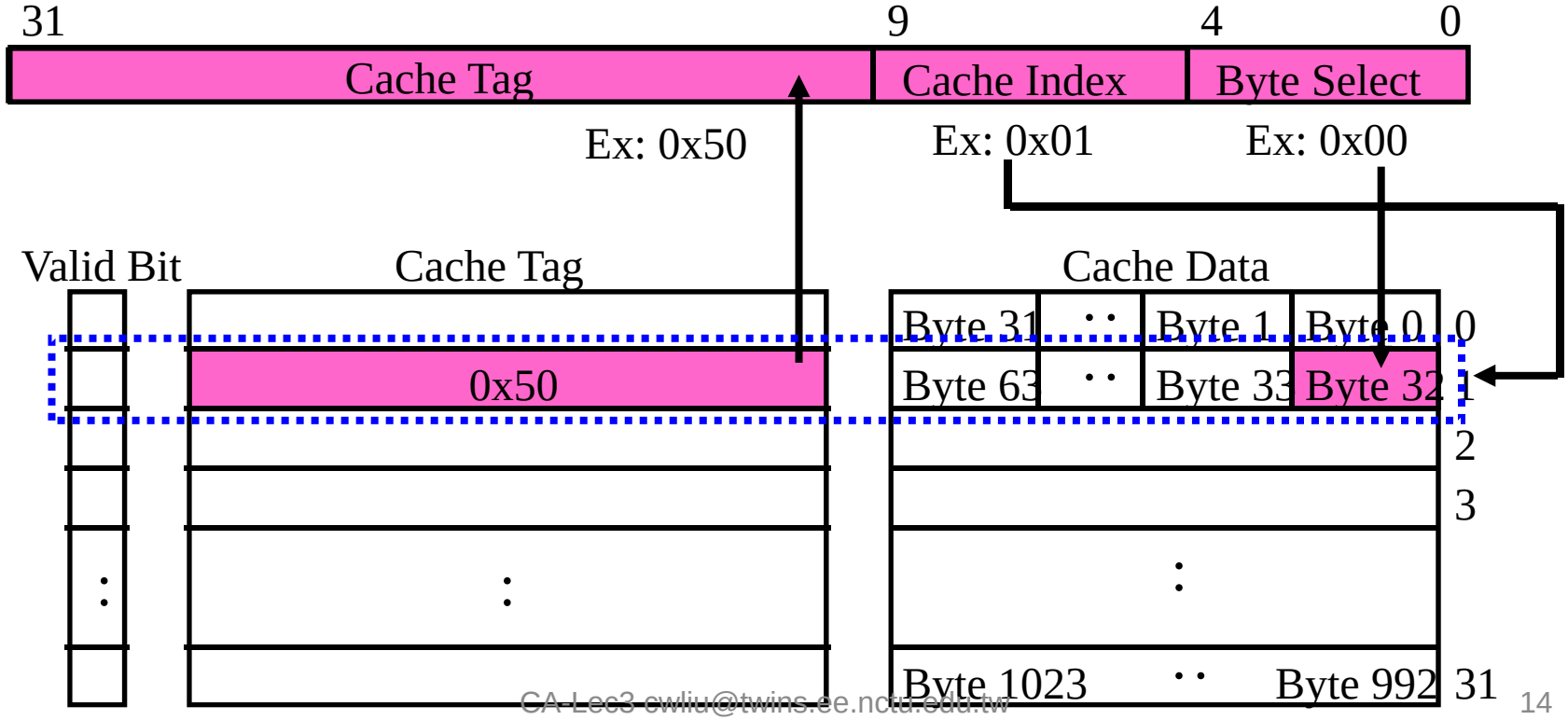
Q2: How is a block found if it is in the upper level?



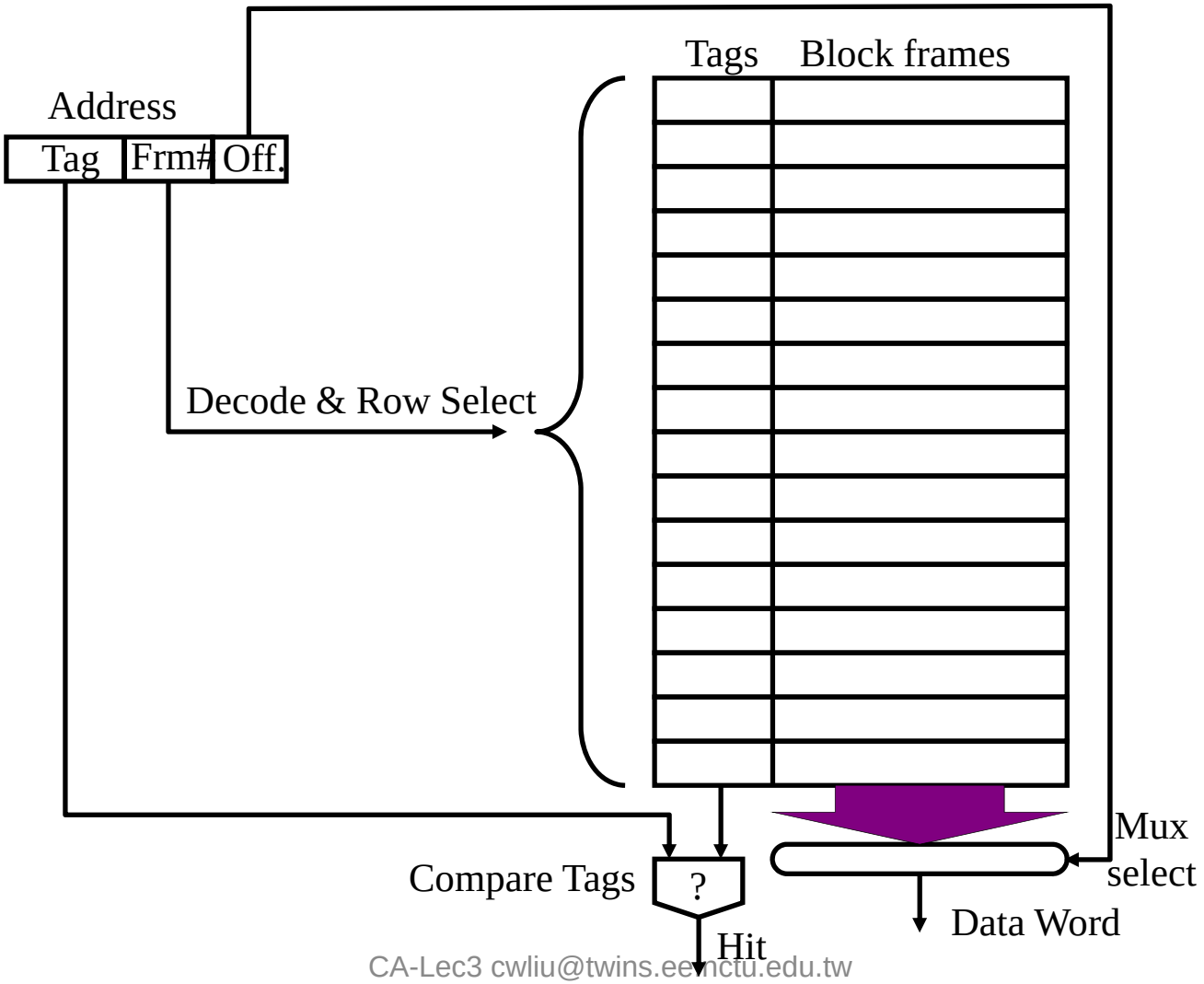
- Index Used to Lookup Candidates
  - Index identifies the set in cache
- Tag used to identify actual copy
  - If no candidates match, then declare cache miss
- Block is minimum quantum of caching
  - Data select field used to select data within block
  - Many caching applications don't have data select field
- Larger block size has distinct hardware advantages:
  - less tag overhead
  - exploit fast burst transfers from DRAM/over wide busses
- Disadvantages of larger block size?
  - Fewer blocks ⇒ more conflicts. Can waste bandwidth

Review: Direct Mapped Cache

- Direct Mapped 2^N byte cache:
  - The uppermost (32 - N) bits are always the Cache Tag
  - The lowest M bits are the Byte Select (Block Size = 2^M)
- Example: 1 KB Direct Mapped Cache with 32 B Blocks
  - Index chooses potential block
  - Tag checked to verify block
  - Byte select chooses byte within block

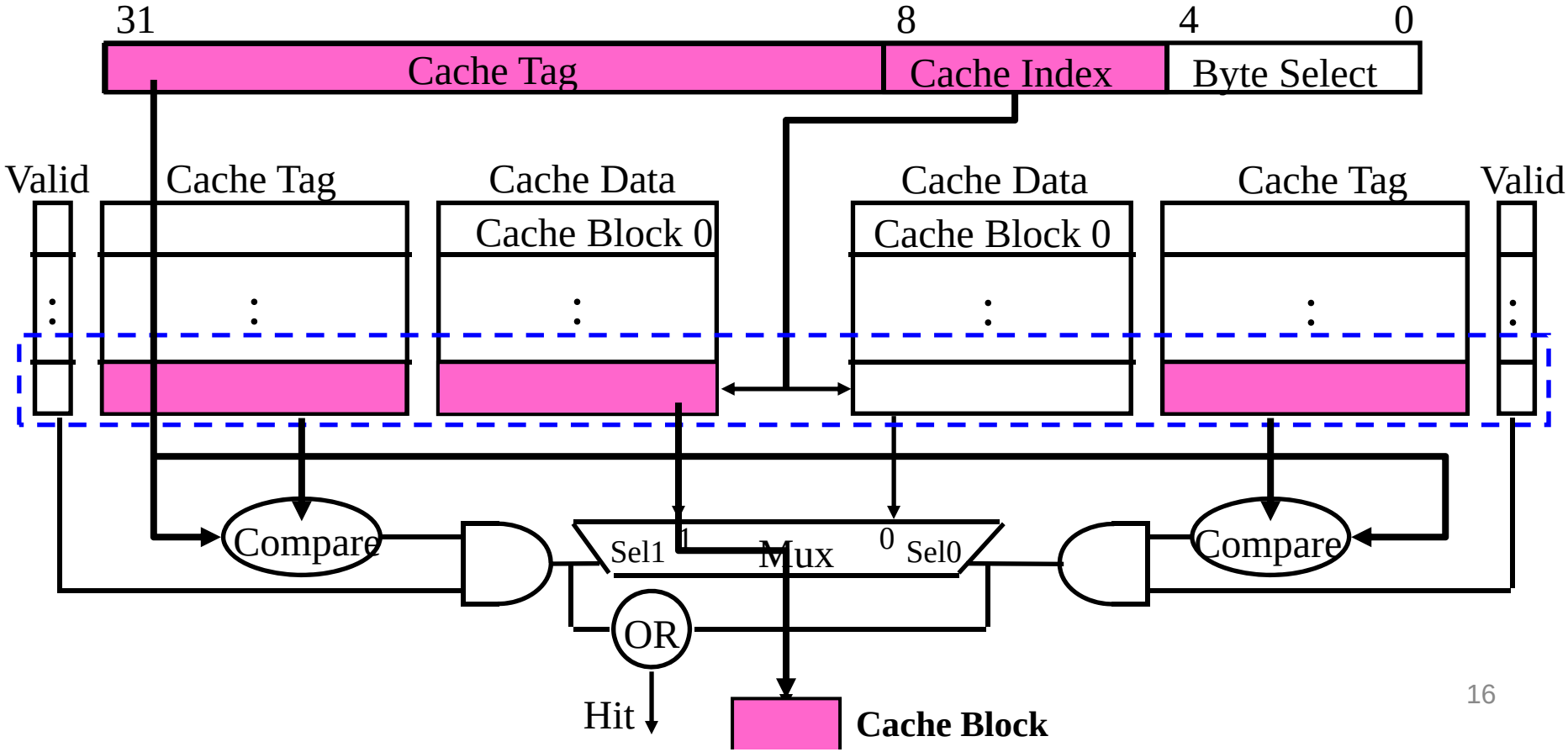


Direct-Mapped Cache Architecture



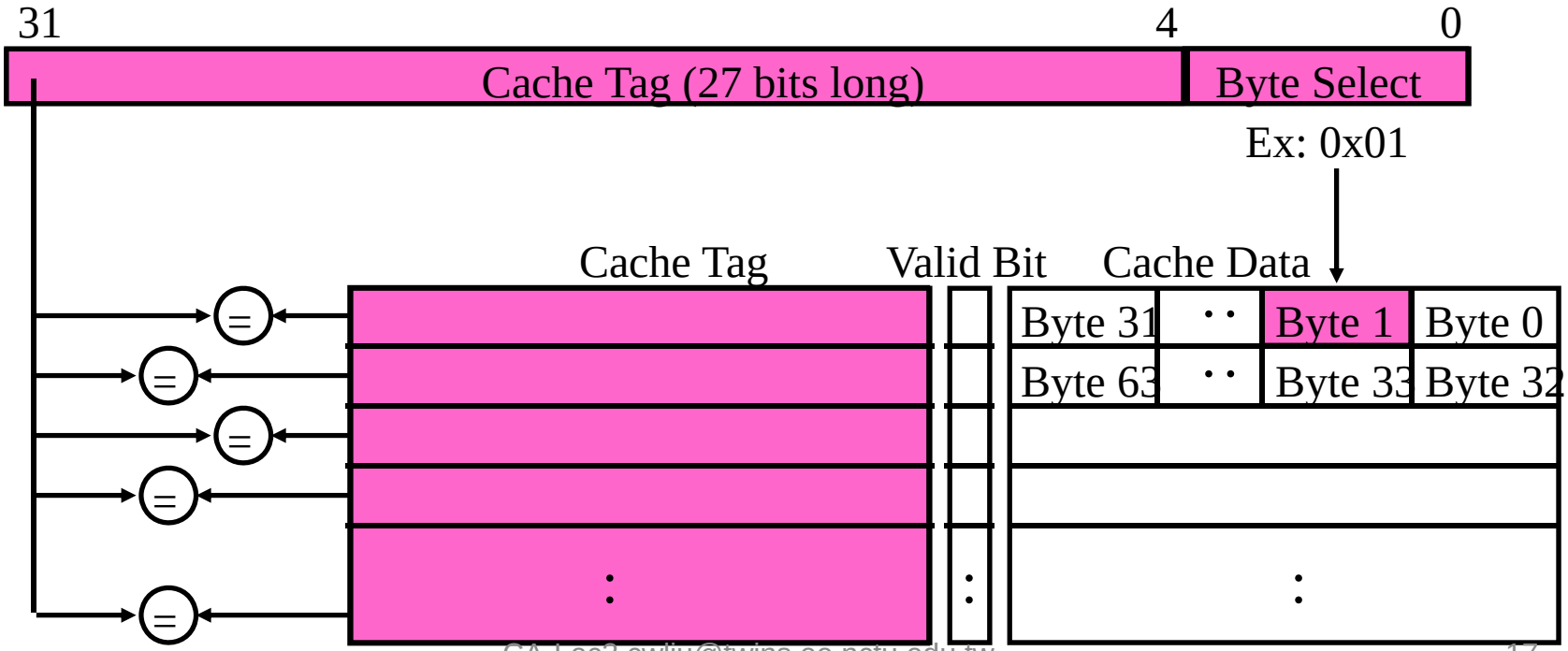
Review: Set Associative Cache

- N-way set associative: N entries per Cache Index
  - N direct mapped caches operates in parallel
- Example: Two-way set associative cache
  - Cache Index selects a “set” from the cache
  - Two tags in the set are compared to input in parallel
  - Data is selected based on the tag result



Review: Fully Associative Cache

- Fully Associative: Every block can hold any line
  - Address does not include a cache index
  - Compare Cache Tags of all Cache Entries in Parallel
- Example: Block Size=32B blocks
  - We need N 27-bit comparators
  - Still have byte select to choose from within block



Concluding Remarks

- Direct-mapped cache = 1-way set-associative cache
- Fully associative cache: there is only 1 set

Cache Size Equation

- Simple equation for the size of a cache:

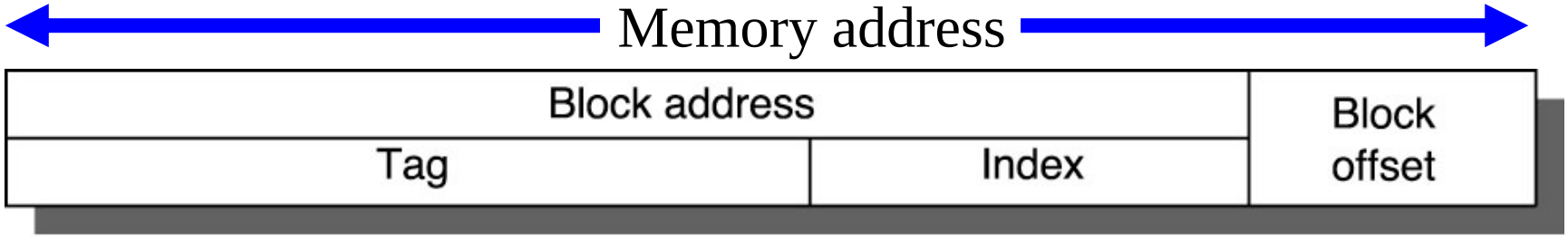
  (Cache size) = (Block size) × (Number of sets)
  × (Set Associativity)

- Can relate to the size of various address fields:

  (Block size) = $2^{(\#\text{ of offset bits})}$

  (Number of sets) = $2^{(\#\text{ of index bits})}$

  (# of tag bits) = (# of memory address bits)
  – (# of index bits) – (# of offset bits)



Q3: Which block should be replaced on a miss?

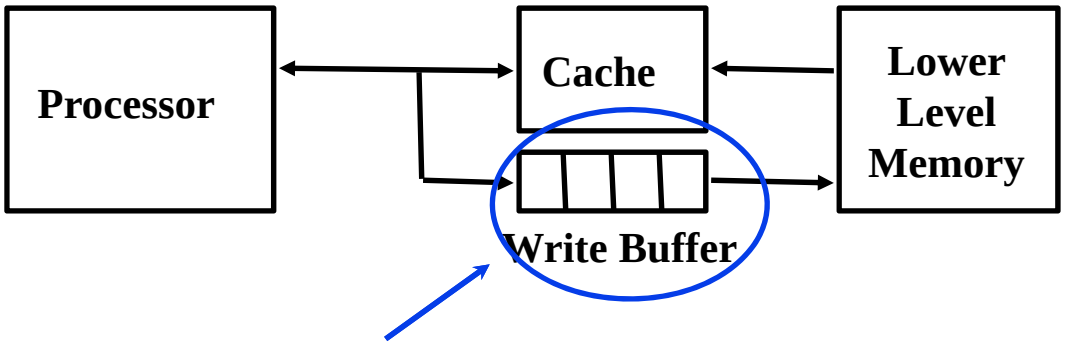
- Easy for direct-mapped cache
  - Only one choice
- Set associative or fully associative
  - LRU (least recently used)
    - Appealing, but hard to implement for high associativity
  - Random
    - Easy, but how well does it work?
  - First in, first out (FIFO)

Q4: What happens on a write?

	Write-Through	Write-Back
Policy	Data written to cache block also written to lower-level memory	Write data only to the cache Update lower level when a block falls out of the cache
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

Additional option -- let writes to an un-cached address allocate a new cache line ("write-allocate").

Write Buffers



Holds data awaiting write-through to lower level memory

Q. Why a write buffer ? **A. So CPU doesn't stall**

Q. Why a buffer, why not just one register ? **A. Bursts of writes are common.**

Q. Are Read After Write (RAW) hazards an issue for write buffer? **A. Yes! Drain buffer before next read, or check write buffers for match on reads**

More on Cache Performance Metrics

- Can split access time into instructions & data:

  Avg. mem. acc. time =
  (% instruction accesses) × (inst. mem. access time) +
  (% data accesses) × (data mem. access time)

- Another formula from chapter 1:

  CPU time = (CPU execution clock cycles + Memory stall clock cycles) × cycle time

  – Useful for exploring ISA changes

- Can break stalls into reads and writes:

  Memory stall cycles =
  (Reads × read miss rate × read miss penalty) +
  (Writes × write miss rate × write miss penalty)

Sources of Cache Misses

- Compulsory (cold start or process migration, first reference): first access to a block
  - "Cold" fact of life: not a whole lot you can do about it
  - Note: If you are going to run "billions" of instruction, Compulsory Misses are insignificant
- Capacity:
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size
- Conflict (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity
- Coherence (Invalidation): other process (e.g., I/O) updates memory

Memory Hierarchy Basics

- Six basic cache optimizations:
  - Larger block size
    - Reduces compulsory misses
    - Increases capacity and conflict misses, increases miss penalty
  - Larger total cache capacity to reduce miss rate
    - Increases hit time, increases power consumption
  - Higher associativity
    - Reduces conflict misses
    - Increases hit time, increases power consumption
  - Higher number of cache levels
    - Reduces overall memory access time
  - Giving priority to read misses over writes
    - Reduces miss penalty
  - Avoiding address translation in cache indexing
    - Reduces hit time

1. Larger Block Sizes

- Larger block size ➔ no. of blocks ↓
- Obvious advantages: reduce compulsory misses
  - Reason is due to spatial locality

- Obvious disadvantage
  - Higher miss penalty: larger block takes longer to move
  - May increase conflict misses and capacity miss if cache is small

- *Don't let increase in miss penalty outweigh the decrease in miss rate*

2. Large Caches

- Cache size↑ ➔ miss rate↓; hit time↑
- Help with both conflict and capacity misses

- May need longer hit time AND/OR higher HW cost
- Popular in off-chip caches

3. Higher Associativity

- Reduce conflict miss
- 2: 1 Cache rule of thumb on miss rate
  - 2 way set associative of size N/ 2 is about the same as a direct mapped cache of size N (held for cache size < 128 KB)

- Greater associativity comes at the cost of increased hit time
- Lengthen the clock cycle

4. Multi-Level Caches

- 2-level caches example
  - $AMAT_{L1} = Hit\text{-}time_{L1} + Miss\text{-}rate_{L1} \times Miss\text{-}penalty_{L1}$
  - $AMAT_{L2} = Hit\text{-}time_{L1} + Miss\text{-}rate_{L1} \times (Hit\text{-}time_{L2} + Miss\text{-}rate_{L2} \times Miss\text{-}penalty_{L2})$
- Probably the best miss-penalty reduction method
- Definitions:
  - Local miss rate: misses in this cache divided by the total number of memory accesses to this cache (Miss-rate-L2)
  - Global miss rate: misses in this cache divided by the total number of memory accesses generated by CPU (Miss-rate-L1 x Miss-rate-L2)
  - Global Miss Rate is what matters

Multi-Level Caches (Cont.)

- Advantages:
  - Capacity misses in L1 end up with a significant penalty reduction
  - Conflict misses in L1 similarly get supplied by L2

- Holding size of 1st level cache constant:
  - Decreases miss penalty of 1st-level cache.
  - Or, increases average global hit time a bit:
    - hit time-L1 + miss rate-L1 x hit time-L2
  - but decreases global miss rate
- Holding total cache size constant:
  - Global miss rate, miss penalty about the same
  - Decreases average global hit time significantly!
    - New L1 much smaller than old L1

Miss Rate Example

- Suppose that in 1000 memory references there are 40 misses in the first-level cache and 20 misses in the second-level cache
  - Miss rate for the first-level cache = 40/1000 (4%)
  - Local miss rate for the second-level cache = 20/40 (50%)
  - Global miss rate for the second-level cache = 20/1000 (2%)
- Assume miss-penalty-L2 is 200 CC, hit-time-L2 is 10 CC, hit-time-L1 is 1 CC, and 1.5 memory reference per instruction. What is average memory access time and average stall cycles per instructions? Ignore writes impact.
  - AMAT = Hit-time-L1 + Miss-rate-L1 × (Hit-time-L2 + Miss-rate-L2× Miss-penalty-L2) = 1 + 4% × (10 + 50% × 200) = 5.4 CC
  - Average memory stalls per instruction = Misses-per-instruction-L1 × Hit-time-L2 + Misses-per-instructions-L2×Miss-penalty-L2
    = (40×1.5/1000) × 10 + (20×1.5/1000) ×200 = 6.6 CC
  - Or (5.4 – 1.0) × 1.5 = 6.6 CC

5. Giving Priority to Read Misses Over Writes

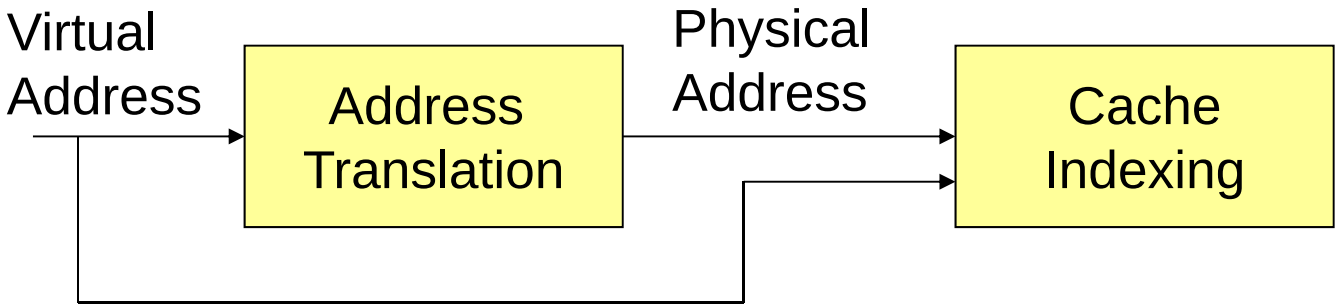
```
SW R3, 512(R0)     ;cache index 0
LW R1, 1024(R0)    ;cache index 0
LW R2, 512(R0)     ;cache index 0
```

R2=R3 ?

- In write through, write buffers complicate memory access in that they might hold the updated value of location needed on a read miss
  - **RAW** conflicts with main memory reads on cache misses
- Read miss waits until the write buffer empty → increase read miss penalty
- Check write buffer contents before read, and if no conflicts, let the memory access continue

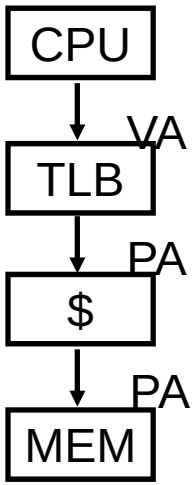
  read priority over write
- Write Back?
  - Read miss replacing dirty block
  - Normal: Write dirty block to memory, and then do the read
  - Instead, copy the dirty block to a write buffer, then do the read, and then do the write
  - CPU stall less since restarts as soon as do read

6. Avoiding Address Translation during Indexing of the Cache

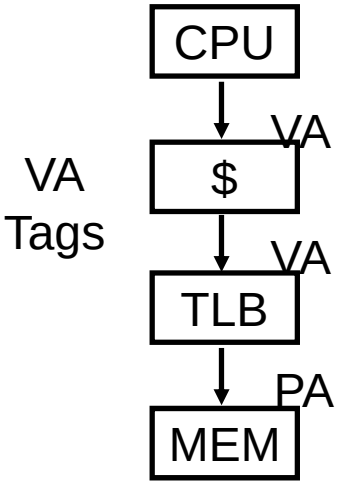


- Virtually addressed caches

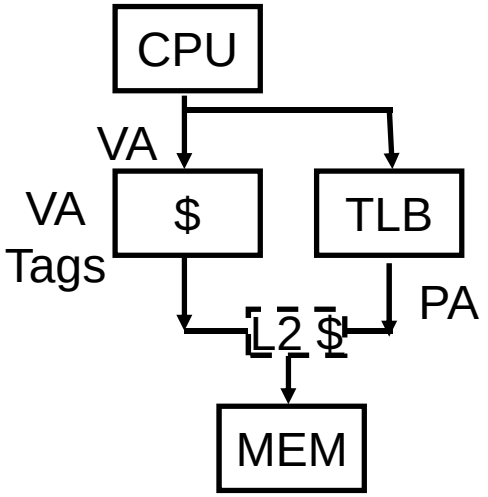
$ means cache



Conventional Organization



Virtually Addressed Cache
Translate only on miss
Synonym (Alias) Problem



Overlap $ access with VA translation: requires $ index to remain invariant across translation

Why not Virtual Cache?

- Task switch causes the same VA to refer to different PAs
  - Hence, cache must be flushed
    - Hugh task switch overhead
    - Also creates huge compulsory miss rates for new process
- Synonyms or Alias problem causes different VAs which map to the same PA
  - Two copies of the same data in a virtual cache
    - Anti-aliasing HW mechanism is required (complicated)
    - SW can help
- I/O (always uses PA)
  - Require mapping to VA to interact with a virtual cache

Advanced Cache Optimizations

- Reducing hit time

1. Small and simple caches
2. Way prediction

- Increasing cache bandwidth

3. Pipelined caches
4. Multibanked caches
5. Nonblocking caches

- Reducing Miss Penalty

6. Critical word first
7. Merging write buffers

- Reducing Miss Rate

8. Compiler optimizations

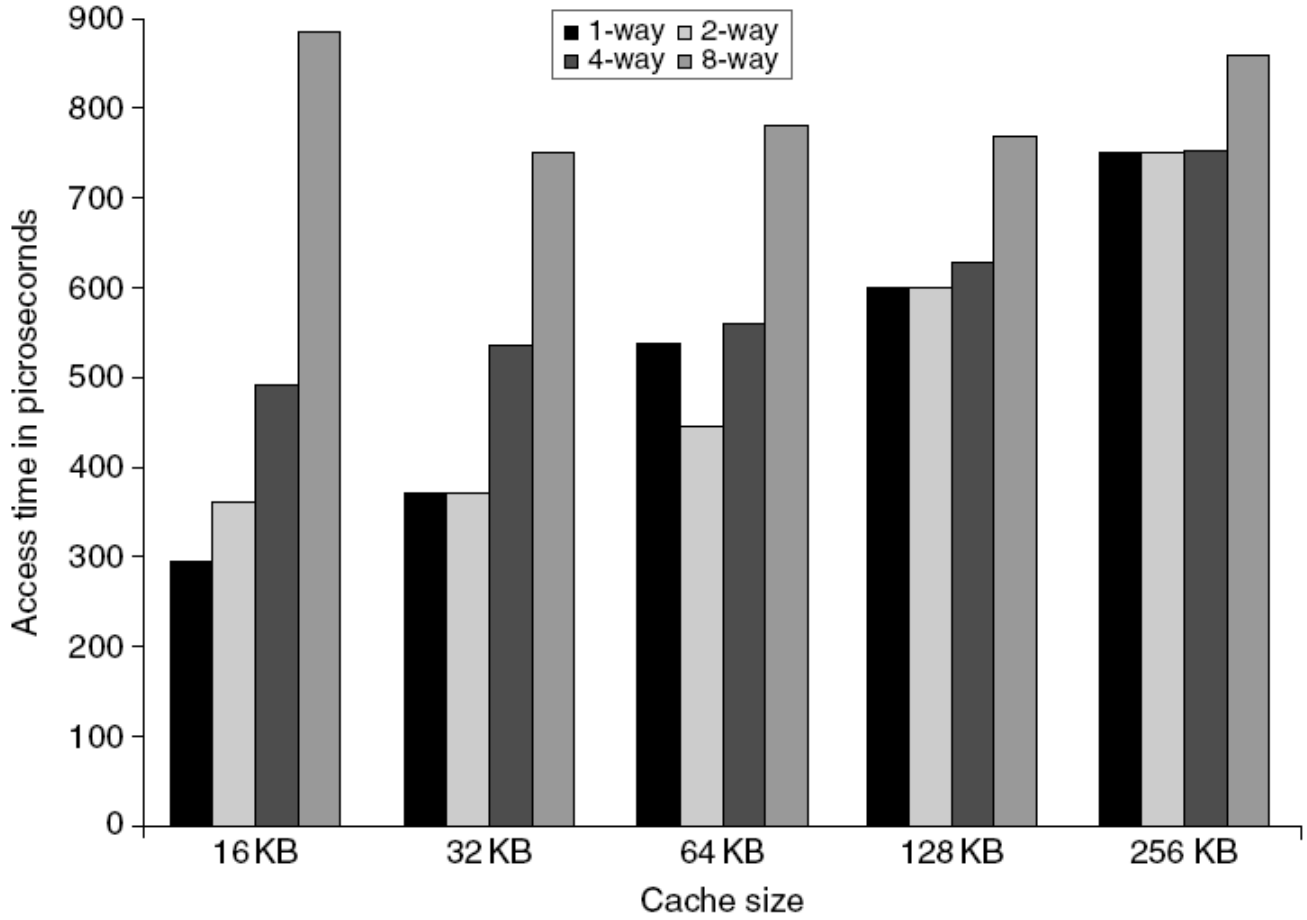
- Reducing miss penalty or miss rate via parallelism

9. Hardware prefetching
10. Compiler prefetching

1. Small and Simple L1 Cache

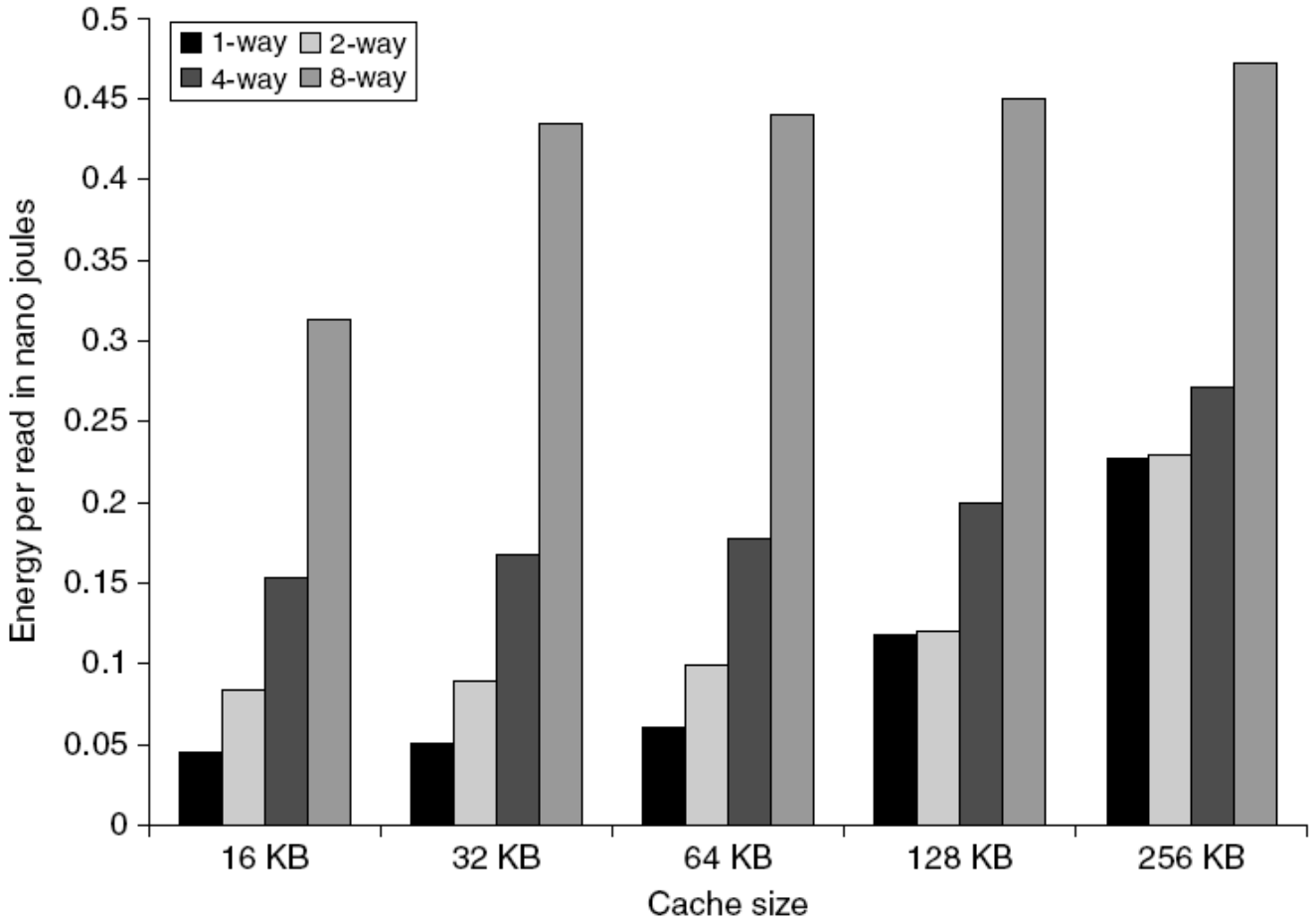
- Critical timing path in cache:
  - addressing tag memory, then comparing tags, then selecting correct set
  - Index tag memory and then compare takes time
- Direct-mapped caches can overlap tag compare and transmission of data
  - Since there is only one choice
- Lower associativity reduces power because fewer cache lines are accessed

L1 Size and Associativity



Access time vs. size and associativity

L1 Size and Associativity



Energy per read vs. size and associativity

2. Fast Hit times via Way Prediction

- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?
- Way prediction: keep extra bits in cache to predict the "way," or block within the set, of next cache access.
  - Multiplexor is set early to select desired block, only 1 tag comparison performed that clock cycle in parallel with reading the cache data
  - Miss ⇒ 1st check other blocks for matches in next clock cycle

- Accuracy ≈ 85%
- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles
  - Used for instruction caches vs. data caches

Way Prediction

- To improve hit time, predict the way to pre-set mux
  - Mis-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8
- Extend to predict block as well
  - “Way selection”
  - Increases mis-prediction penalty

3. Increasing Cache Bandwidth by Pipelining

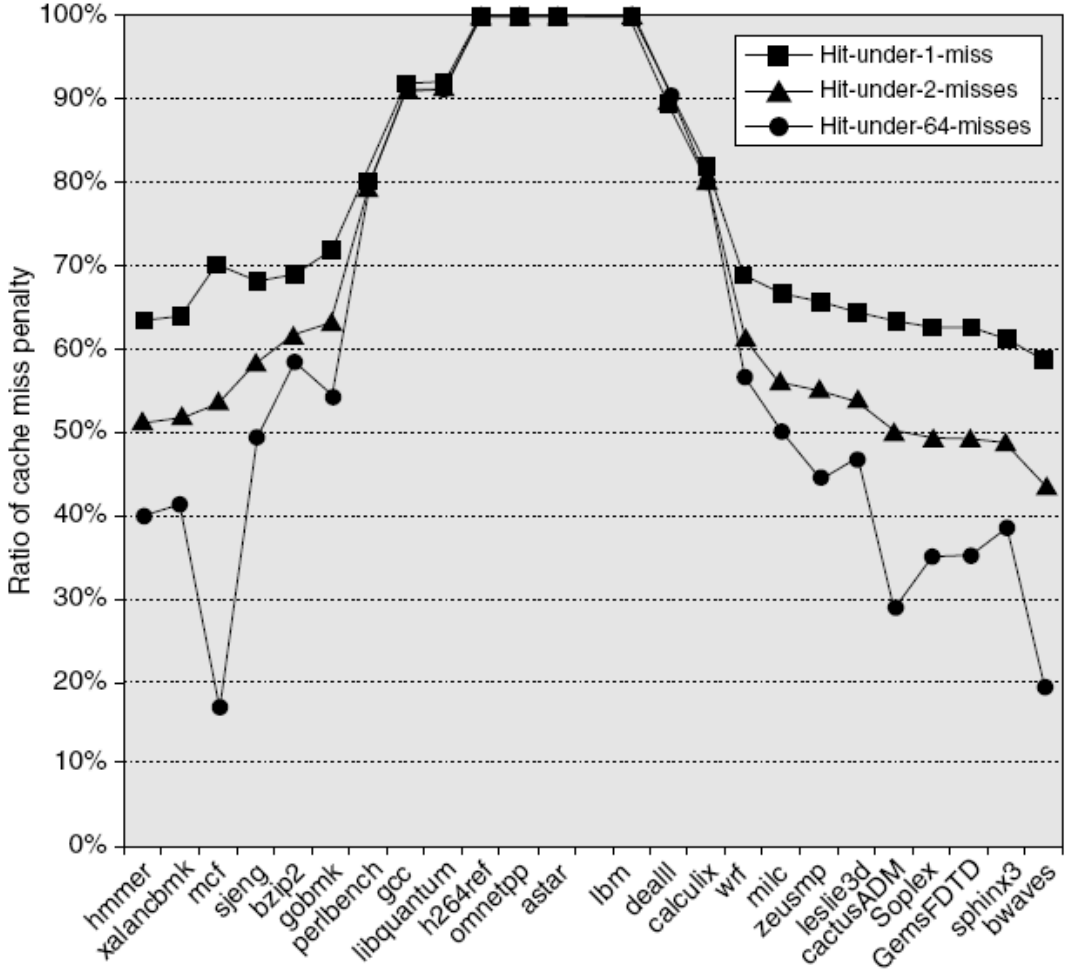
- Pipeline cache access to improve bandwidth
  - Examples:
    - Pentium: 1 cycle
    - Pentium Pro – Pentium III: 2 cycles
    - Pentium 4 – Core i7: 4 cycles
- Makes it easier to increase associativity

- But, pipeline cache increases the access latency
  - More clock cycles between the issue of the load and the use of the data
- Also Increases branch mis-prediction penalty

4. Increasing Cache Bandwidth: Non-Blocking Caches

- *Non-blocking cache* or *lockup-free cache* allow data cache to continue to supply cache hits during a miss
  - requires F/E bits on registers or out-of-order execution
  - requires multi-bank memories
- "*hit under miss*" reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- "*hit under multiple miss*" or "*miss under miss*" may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
  - Requires muliple memory banks (otherwise cannot support)
  - Penium Pro allows 4 outstanding memory misses

Nonblocking Cache Performances



- L2 must support this
- In general, processors can hide L1 miss penalty but not L2 miss penalty

6: Increasing Cache Bandwidth via Multiple Banks

- Rather than treat the cache as a single monolithic block, divide into independent banks that can support simultaneous accesses
  - E.g.,T1 ("Niagara") L2 has 4 banks
- Banking works best when accesses naturally spread themselves across banks ⇒ mapping of addresses to banks affects behavior of memory system
- Simple mapping that works well is "sequential interleaving"
  - Spread block addresses sequentially across banks
  - E,g, if there 4 banks, Bank 0 has all blocks whose address modulo 4 is 0; bank 1 has all blocks whose address modulo 4 is 1; …

5. Increasing Cache Bandwidth via Multibanked Caches

- Organize cache as independent banks to support simultaneous access (rather than a single monolithic block)
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2
- Banking works best when accesses naturally spread themselves across banks
  - Interleave banks according to block address

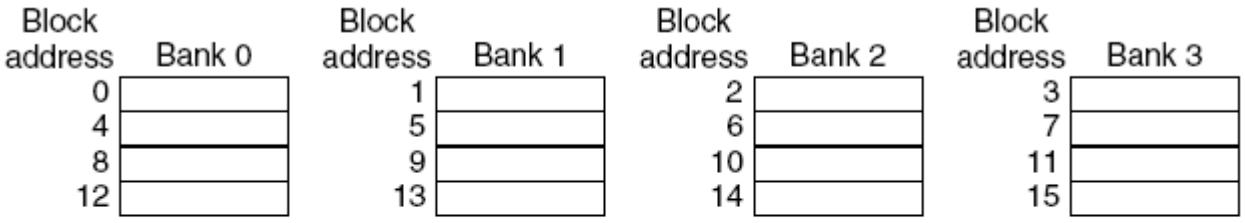


Figure 2.6 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

6. Reduce Miss Penalty: Critical Word First and Early Restart

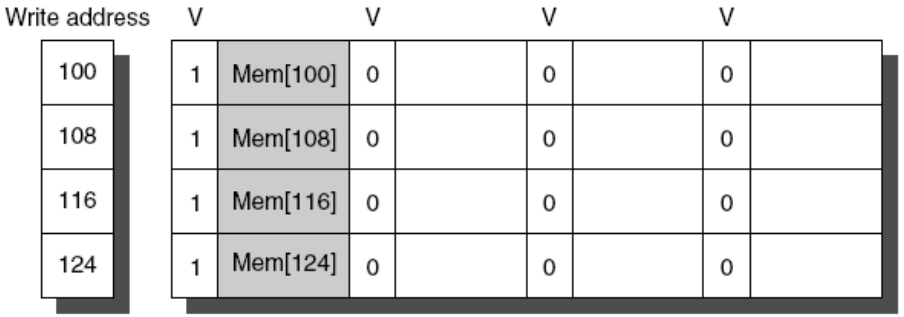
- Processor usually needs one word of the block at a time
- Do not wait for full block to be loaded before restarting processor
  - Critical Word First – request the missed word first from memory and send it to the processor as soon as it arrives; let the processor continue execution while filling the rest of the words in the block. Also called wrapped fetch and requested word first
  - Early restart -- as soon as the requested word of the block arrives, send it to the processor and let the processor continue execution
- Benefits of critical word first and early restart depend on
  - Block size: generally useful only in large blocks
  - Likelihood of another access to the portion of the block that has not yet been fetched
    - Spatial locality problem: tend to want next sequential word, so not clear if benefit

7. Merging Write Buffer to Reduce Miss Penalty

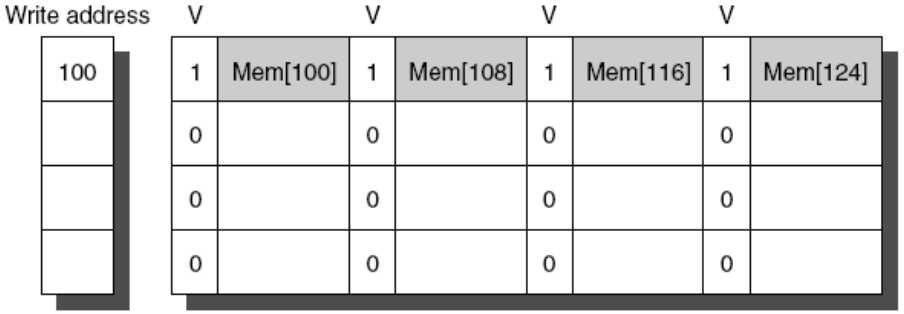
- Write buffer to allow processor to continue while waiting to write to memory
- If buffer contains modified blocks, the addresses can be checked to see if address of new data matches the address of a valid write buffer entry. If so, new data are combined with that entry
- Increases block size of write for write-through cache of writes to sequential words, bytes since multiword writes more efficient to memory

Merging Write Buffer

- When storing to a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer



No write buffering



Write buffering

8. Reducing Misses by Compiler Optimizations

- McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks in software
- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts(using tools they developed)
- Data
  - *Loop Interchange*: swap nested loops to access data in order stored in memory (in sequential order)
  - *Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap
  - *Blocking*: Improve temporal locality by accessing "blocks" of data repeatedly vs. going down whole columns or rows
    - Instead of accessing entire rows or columns, subdivide matrices into blocks
    - Requires more memory accesses but improves locality of accesses

Loop Interchange Example

```
/* Before */
for (k = 0; k < 100; k = k+1)
   for (j = 0; j < 100; j = j+1)
       for (i = 0; i < 5000; i = i+1)
              x[i][j] = 2 * x[i][j];
/* After */
for (k = 0; k < 100; k = k+1)
   for (i = 0; i < 5000; i = i+1)
       for (j = 0; j < 100; j = j+1)
              x[i][j] = 2 * x[i][j];
```

Sequential accesses instead of striding through memory every 100 words; improved spatial locality

Loop Fusion Example

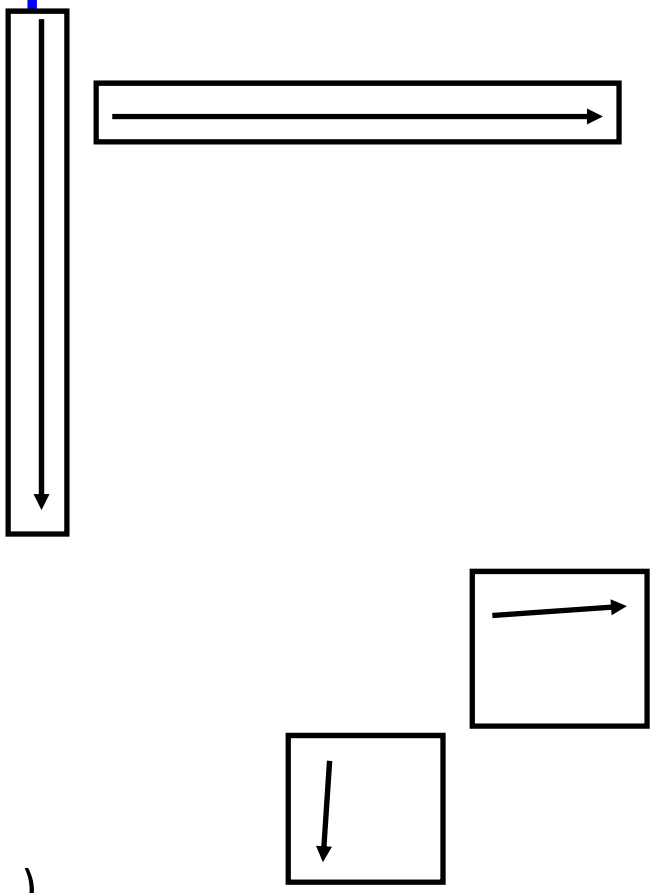
```
/* Before */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
       a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
       d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
   {   a[i][j] = 1/b[i][j] * c[i][j];
       d[i][j] = a[i][j] + c[i][j];}
```

Perform different computations on the common data in two loops
➔ fuse the two loops

2 misses per access to a & c vs. one miss per access; improve spatial locality

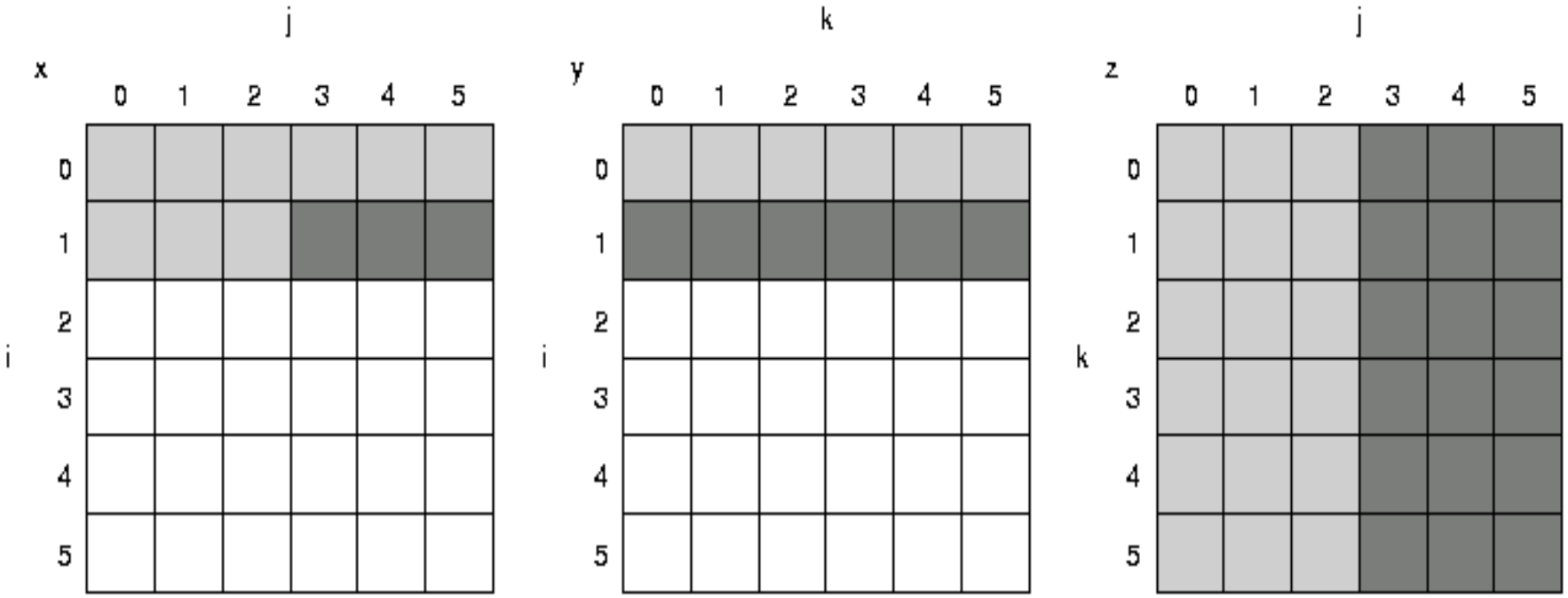
Blocking Example

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
     {r = 0;
      for (k = 0; k < N; k = k+1){
        r = r + y[i][k]*z[k][j];};
      x[i][j] = r;
     };
```



- Two Inner Loops:
  - Read all NxN elements of z[]
  - Read N elements of 1 row of y[] repeatedly
  - Write N elements of 1 row of x[]
- Capacity Misses a function of N & Cache Size:
  - $2N^3 + N^2$ => (assuming no conflict; otherwise …)
- Idea: compute on BxB submatrix that fits

Snapshot of x, y, z when N=6, i=1



White: not yet touched
Light: older access
Dark: newer access

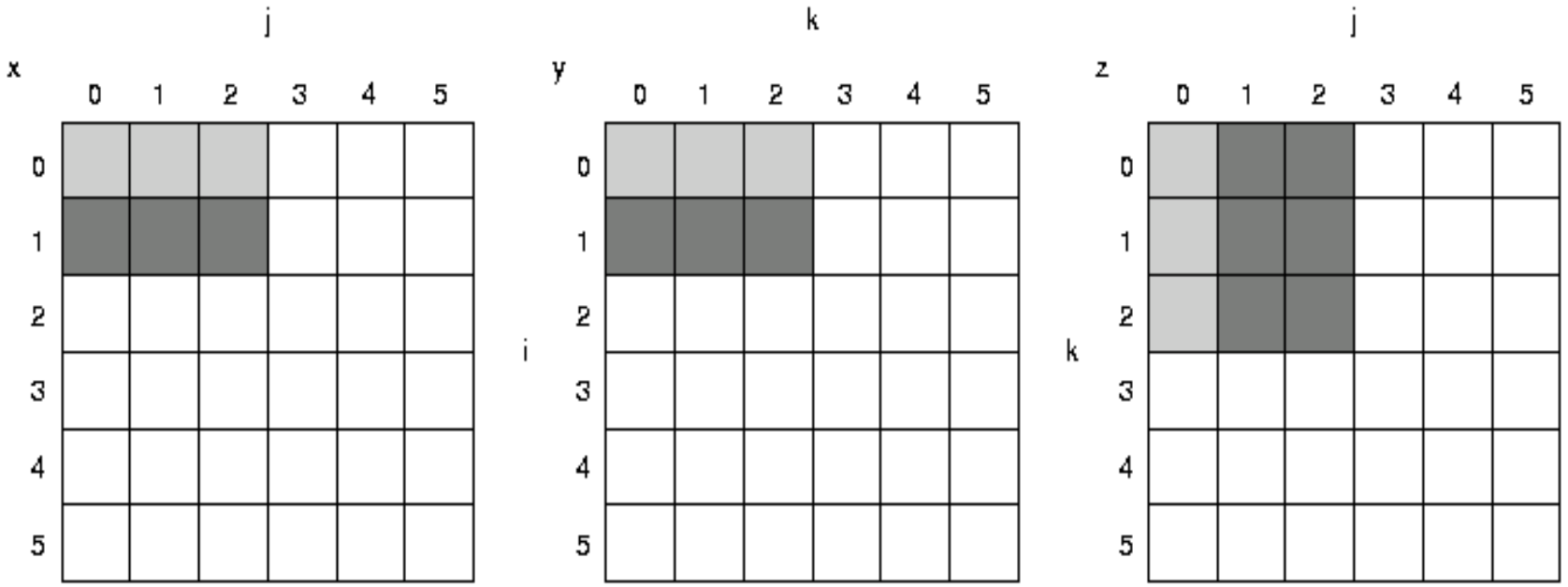
Before….

Blocking Example

```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
   for (j = jj; j < min(jj+B-1,N); j = j+1)
     {r = 0;
      for (k = kk; k < min(kk+B-1,N); k = k+1) {
        r = r + y[i][k]*z[k][j];};
      x[i][j] = x[i][j] + r;
     };
```

- B called *Blocking Factor*
- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
- Conflict Misses Too?

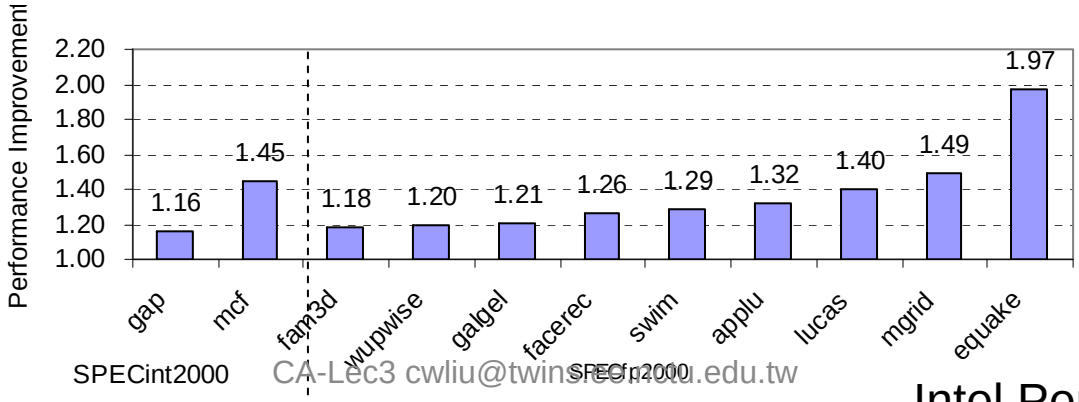
The Age of Accesses to x, y, z when B=3



Note in contrast to previous Figure, the smaller number of elements accessed

9. Reducing Miss Penalty or Miss Rate by Hardware Prefetching of Instructions & Data

- Prefetching relies on having extra memory bandwidth that can be used without penalty
- Instruction Prefetching
  - Typically, CPU fetches 2 blocks on a miss: the requested block and the next consecutive block.
  - Requested block is placed in instruction cache when it returns, and prefetched block is placed into instruction stream buffer
- Data Prefetching
  - Pentium 4 can prefetch data into L2 cache from up to 8 streams from 8 different 4 KB pages
  - Prefetching invoked if 2 successive L2 cache misses to a page, if distance between those cache blocks is < 256 bytes



Intel Pentium 4

10. Reducing Miss Penalty or Miss Rate by Compiler-Controlled Prefetching Data

- Prefetch instruction is inserted before data is needed

- Data Prefetch
  - Register prefetch: load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution

- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth
  - Combine with software pipelining and loop unrolling

Summary

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined cache access	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Banked caches		+			+	1	Used in L2 of both i7 and Cortex-A8
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware.
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs

Figure 2.11 Summary of 10 advanced cache optimizations showing impact on cache performance, power consumption, and complexity. Although generally a technique helps only one factor, prefetching can reduce misses if done sufficiently early; if not, it can reduce miss penalty. + means that the technique improves the factor, – means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.

Memory Technology

- Performance metrics
  - Latency is concern of cache
  - Bandwidth is concern of multiprocessors and I/O
  - Access time
    - Time between read request and when desired word arrives
  - Cycle time
    - Minimum time between unrelated requests to memory
- DRAM used for main memory, SRAM used for cache

Memory Technology

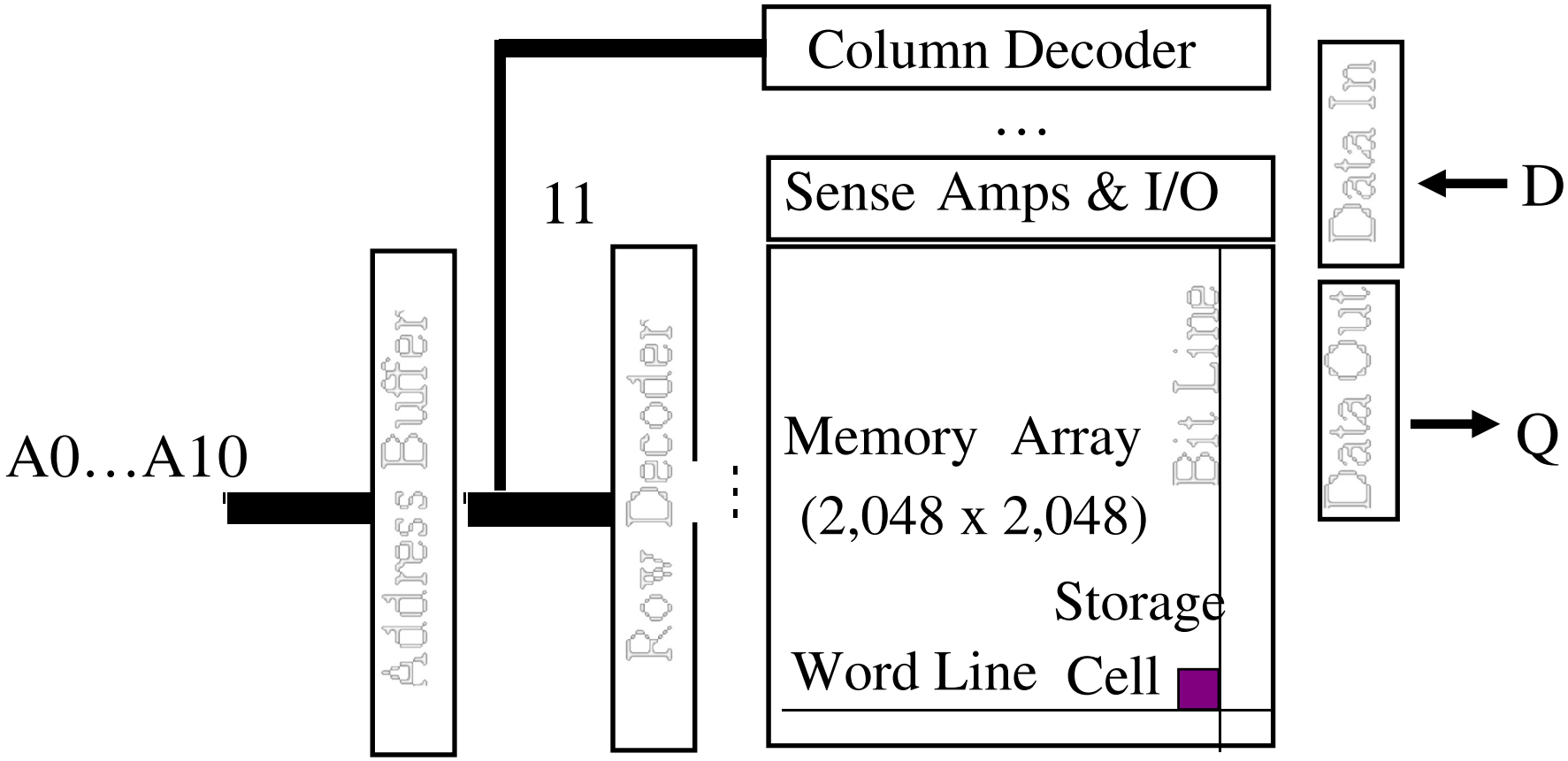
- SRAM: static random access memory
  - Requires low power to retain bit, since no refresh
  - But, requires 6 transistors/bit (vs. 1 transistor/bit)

- DRAM
  - One transistor/bit
  - Must be re-written after being read
  - Must also be periodically refreshed
    - Every ~ 8 ms
    - Each row can be refreshed simultaneously
  - Address lines are multiplexed:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)

DRAM Technology

- Emphasize on cost per bit and capacity
- Multiplex address lines ➔ cutting # of address pins in half
  - Row access strobe (RAS) first, then column access strobe (CAS)
  - Memory as a 2D matrix – rows go to a buffer
  - Subsequent CAS selects subrow
- Use only a single transistor to store a bit
  - Reading that bit can destroy the information
  - Refresh each bit periodically (ex. 8 milliseconds) by writing back
    - Keep refreshing time less than 5% of the total time
- DRAM capacity is 4 to 8 times that of SRAM

DRAM Logical Organization (4Mbit)



- Square root of bits per RAS/CAS

DRAM Technology (cont.)

- DIMM: Dual inline memory module
  - DRAM chips are commonly sold on small boards called DIMMs
  - DIMMs typically contain 4 to 16 DRAMs
- Slowing down in DRAM capacity growth
  - Four times the capacity every three years, for more than 20 years
  - New chips only double capacity every two year, since 1998
- DRAM performance is growing at a slower rate
  - RAS (related to latency): 5% per year
  - CAS (related to bandwidth): 10%+ per year

RAS Improvement

Production year	Chip size	DRAM Type	Row access strobe (RAS) Slowest DRAM (ns)	Row access strobe (RAS) Fastest DRAM (ns)	Column access strobe (CAS)/ data transfer time (ns)	Cycle time (ns)
1980	64K bit	DRAM	180	150	75	250
1983	256K bit	DRAM	150	120	50	220
1986	1M bit	DRAM	120	100	25	190
1989	4M bit	DRAM	100	80	20	165
1992	16M bit	DRAM	80	60	15	120
1996	64M bit	SDRAM	70	50	12	110
1998	128M bit	SDRAM	70	50	10	100
2000	256M bit	DDR1	65	45	7	90
2002	512M bit	DDR1	60	40	5	80
2004	1G bit	DDR2	55	35	5	70
2006	2G bit	DDR2	50	30	2.5	60
2010	4G bit	DDR3	36	28	1	37
2012	8G bit	DDR3	30	24	0.5	31

Figure 2.13 Times of fast and slow DRAMs vary with each generation. (Cycle time is defined on page 95.) Performance improvement of row access time is about 5% per year. The improvement by a factor of 2 in column access in 1986 accompanied the switch from NMOS DRAMs to CMOS DRAMs. The introduction of various burst transfer modes in the mid-1990s and SDRAMs in the late 1990s has significantly complicated the calculation of access time for blocks of data; we discuss this later in this section when we talk about SDRAM access time and power. The DDR4 designs are due for introduction in mid- to late 2012. We discuss these various forms of DRAMs in the next few pages.

Quest for DRAM Performance

1. Fast Page mode
   - Add timing signals that allow repeated accesses to row buffer without another row access time
   - Such a buffer comes naturally, as each array will buffer 1024 to 2048 bits for each access
2. Synchronous DRAM (SDRAM)
   - Add a clock signal to DRAM interface, so that the repeated transfers would not bear overhead to synchronize with DRAM controller
3. Double Data Rate (DDR SDRAM)
   - Transfer data on both the rising edge and falling edge of the DRAM clock signal $\Rightarrow$ doubling the peak data rate
   - DDR2 lowers power by dropping the voltage from 2.5 to 1.8 volts + offers higher clock rates: up to 400 MHz
   - DDR3 drops to 1.5 volts + higher clock rates: up to 800 MHz
   - DDR4 drops to 1.2 volts, clock rate up to 1600 MHz

- Improved Bandwidth, not Latency

DRAM name based on Peak Chip Transfers / Sec
DIMM name based on Peak DIMM MBytes / Sec

Fastest for sale 4/06 ($125/GB)

Stan-dard	Clock Rate (MHz)	M transfers / second	DRAM Name	Mbytes/s/ DIMM	DIMM Name
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10664	PC10700
DDR3	800	1600	DDR3-1600	12800	PC12800

x 2

x 8

DRAM Performance

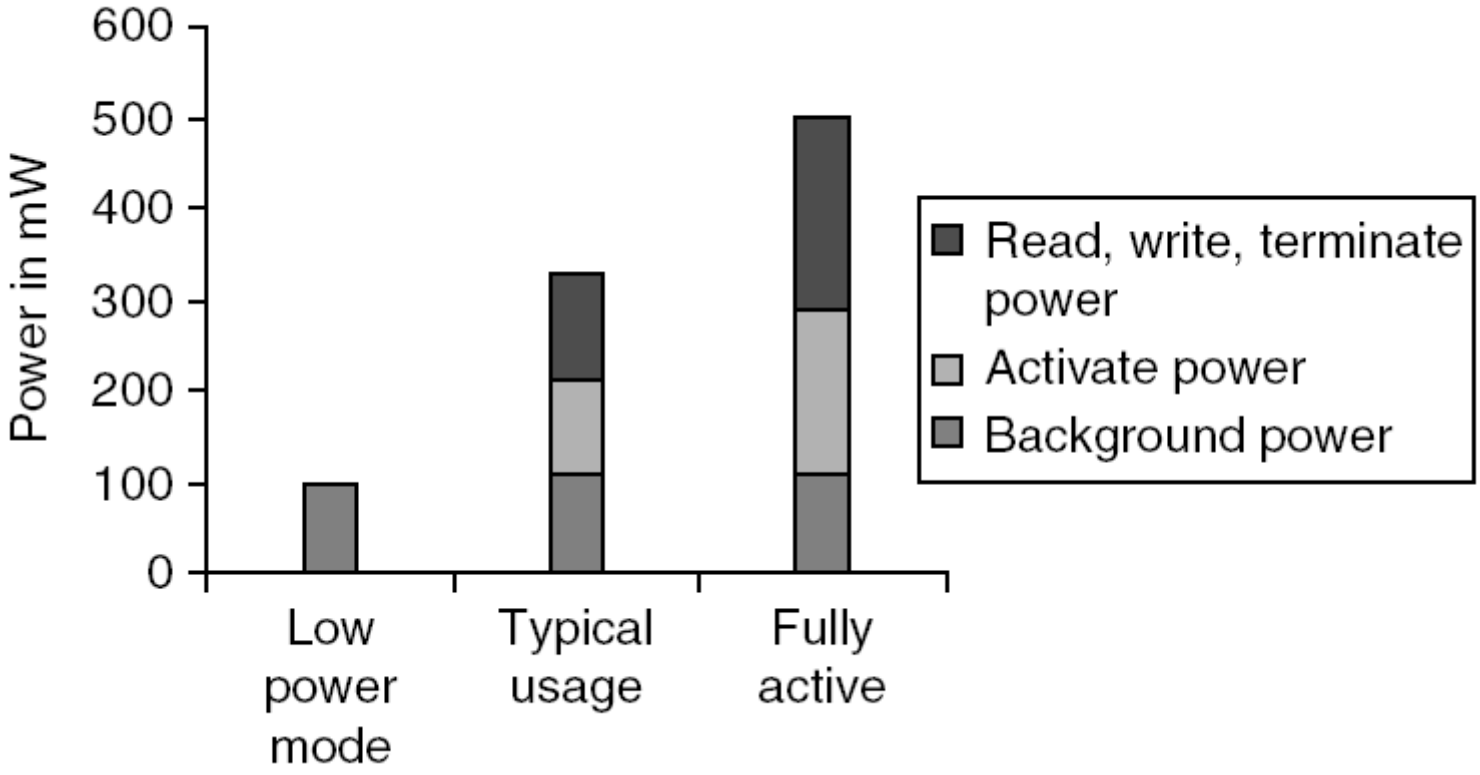
Standard	Clock rate (MHz)	M transfers per second	DRAM name	MB/sec /DIMM	DIMM name
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10,664	PC10700
DDR3	800	1600	DDR3-1600	12,800	PC12800
DDR4	1066–1600	2133–3200	DDR4-3200	17,056–25,600	PC25600

Figure 2.14 Clock rates, bandwidth, and names of DDR DRAMS and DIMMs in 2010. Note the numerical relationship between the columns. The third column is twice the second, and the fourth uses the number from the third column in the name of the DRAM chip. The fifth column is eight times the third column, and a rounded version of this number is used in the name of the DIMM. Although not shown in this figure, DDRs also specify latency in clock cycles as four numbers, which are specified by the DDR standard. For example, DDR3-2000 CL 9 has latencies of 9-9-9-28. What does this mean? With a 1 ns clock (clock cycle is one-half the transfer rate), this indicate 9 ns for row to columns address (RAS time), 9 ns for column access to data (CAS time), and a minimum read time of 28 ns. Closing the row takes 9 ns for precharge but happens only when the reads from that row are finished. In burst mode, transfers occur on every clock on both edges, when the first RAS and CAS times have elapsed. Furthermore, the precharge in not needed until the entire row is read. DDR4 will be produced in 2012 and is expected to reach clock rates of 1600 MHz in 2014, when DDR5 is expected to take over. The exercises explore these details further.

Graphics Memory

- GDDR5 is graphics memory based on DDR3
- Graphics memory:
  - Achieve 2-5 X bandwidth per DRAM vs. DDR3
    - Wider interfaces (32 vs. 16 bit)
    - Higher clock rate
      - Possible because they are attached via soldering instead of socketed DIMM modules

Memory Power Consumption



SRAM Technology

- Cache uses SRAM: Static Random Access Memory
- SRAM uses six transistors per bit to prevent the information from being disturbed when read

  ➔ no need to refresh
  - SRAM needs only minimal power to retain the charge in the standby mode ➔ good for embedded applications
  - No difference between access time and cycle time for SRAM
- Emphasize on speed and capacity
  - SRAM address lines are not multiplexed
- SRAM speed is 8 to 16x that of DRAM

ROM and Flash

- Embedded processor memory
- Read-only memory (ROM)
  - Programmed at the time of manufacture
  - Only a single transistor per bit to represent 1 or 0
  - Used for the embedded program and for constant
  - Nonvolatile and indestructible
- Flash memory:
  - Must be erased (in blocks) before being overwritten
  - Nonvolatile but allow the memory to be modified
  - Reads at almost DRAM speeds, but writes 10 to 100 times slower
  - DRAM capacity per chip and MB per dollar is about 4 to 8 times greater than flash
  - Cheaper than SDRAM, more expensive than disk
  - Slower than SRAM, faster than disk

Memory Dependability

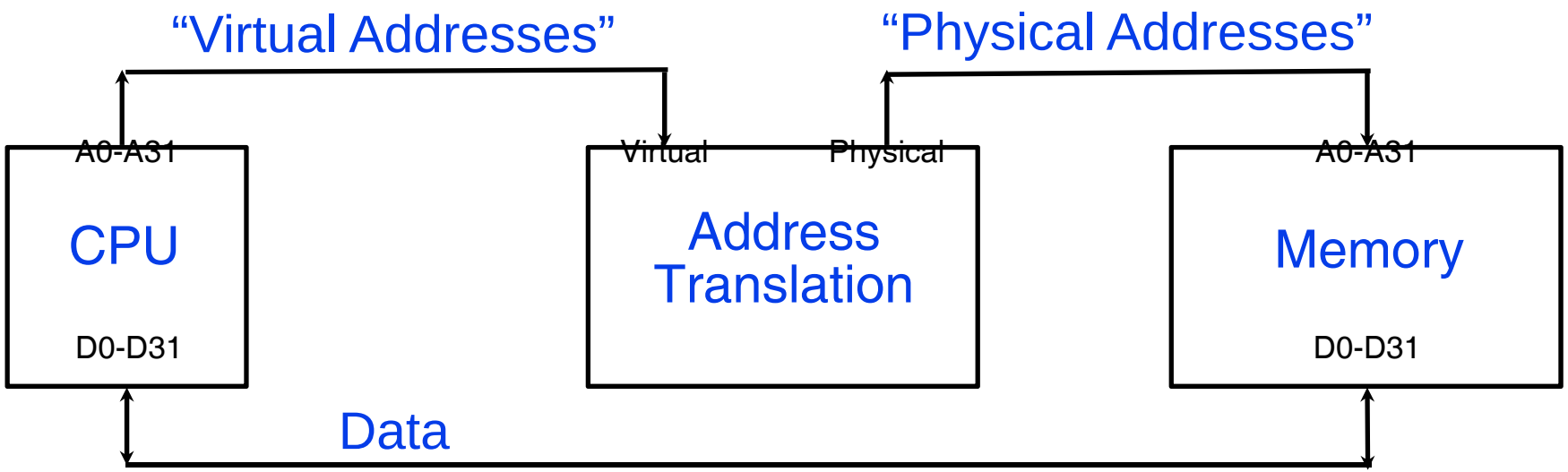
- Memory is susceptible to cosmic rays
- *Soft errors*: dynamic errors
  - Detected and fixed by error correcting codes (ECC)
- *Hard errors*: permanent errors
  - Use sparse rows to replace defective rows

- Chipkill: a RAID-like error recovery technique

Virtual Memory ?

- The limits of physical addressing
  - All programs share one physical address space
  - Machine language programs must be aware of the machine organization
  - No way to prevent a program from accessing any machine resource
- Recall: many processes use only a small portion of address space
- Virtual memory divides physical memory into blocks (called page or segment) and allocates them to different processes
- With virtual memory, the processor produces virtual address that are translated by a combination of HW and SW to physical addresses (called memory mapping or address translation).

Virtual Memory: Add a Layer of Indirection

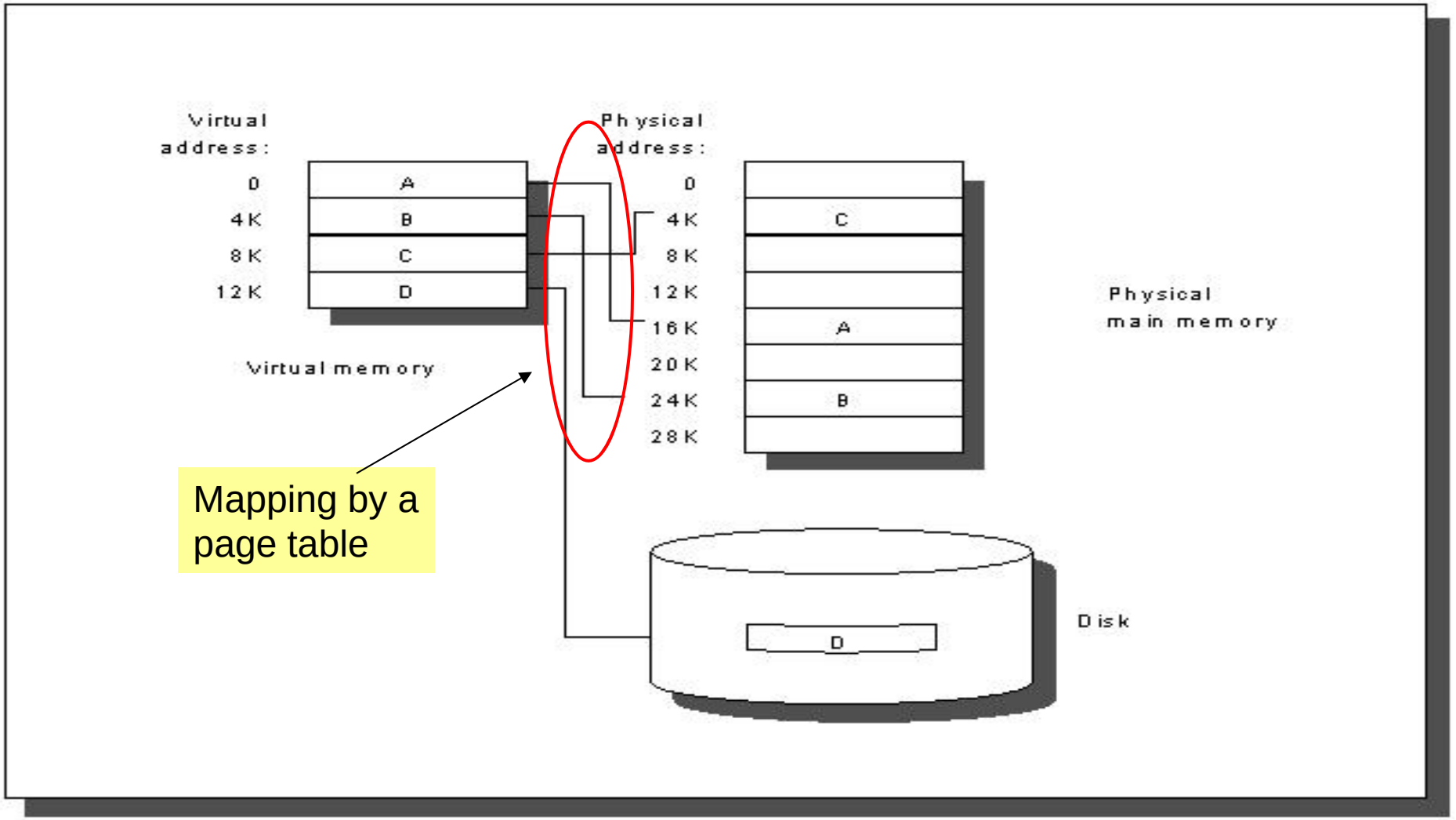


User programs run in an standardized
virtual address space

Address Translation hardware
managed by the operating system (OS)
maps virtual address to physical memory

Hardware supports "modern" OS features:
Protection, Translation, Sharing

Virtual Memory



Virtual Memory (cont.)

- Permits applications to grow bigger than main memory size
- Helps with multiple process management
  - Each process gets its own chunk of memory
  - Permits protection of 1 process' chunks from another
  - Mapping of multiple chunks onto shared physical memory
  - Mapping also facilitates relocation (a program can run in any memory location, and can be moved during execution)
  - Application and CPU run in virtual space (logical memory, 0 – max)
  - Mapping onto physical space is invisible to the application
- Cache vs. virtual memory
  - Block becomes a page or segment
  - Miss becomes a page or address fault

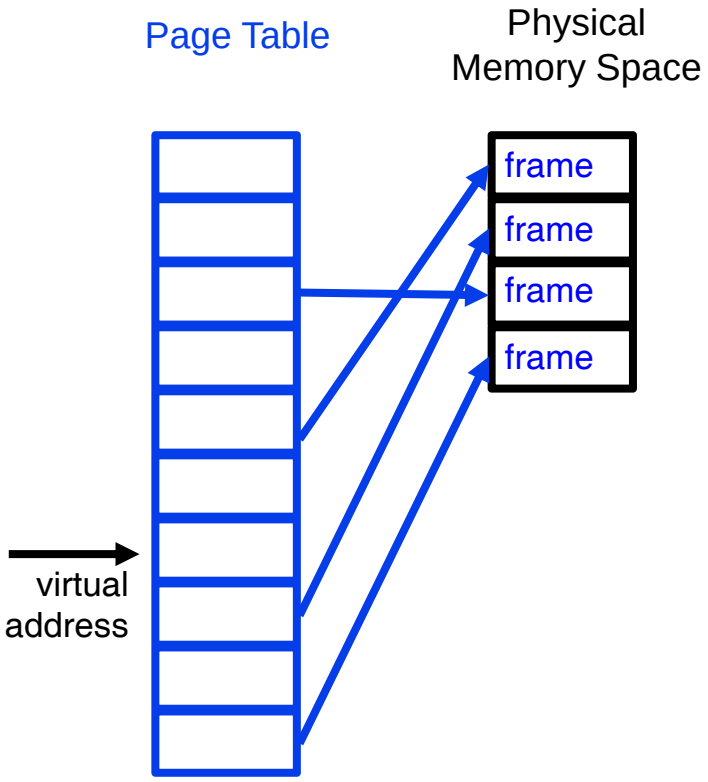
3 Advantages of VM

- Translation:
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - Makes multithreading reasonable (now used a lot!)
  - Only the most important part of program ("Working Set") must be in physical memory.
  - Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later.
- Protection:
  - Different threads (or processes) protected from each other.
  - Different pages can be given special behavior
    - (Read Only, Invisible to user programs, etc).
  - Kernel data protected from User programs
  - Very important for protection from malicious programs
- Sharing:
  - Can map same physical page to multiple users ("Shared memory")

Virtual Memory

- Protection via virtual memory
  - Keeps processes in their own memory space
- Role of architecture:
  - Provide user mode and supervisor mode
  - Protect certain aspects of CPU state
  - Provide mechanisms for switching between user mode and supervisor mode
  - Provide mechanisms to limit memory accesses
  - Provide TLB to translate addresses

Page Tables Encode Virtual Address Spaces



A virtual address space is divided into blocks of memory called pages

A machine usually supports pages of a few sizes (MIPS R4000):

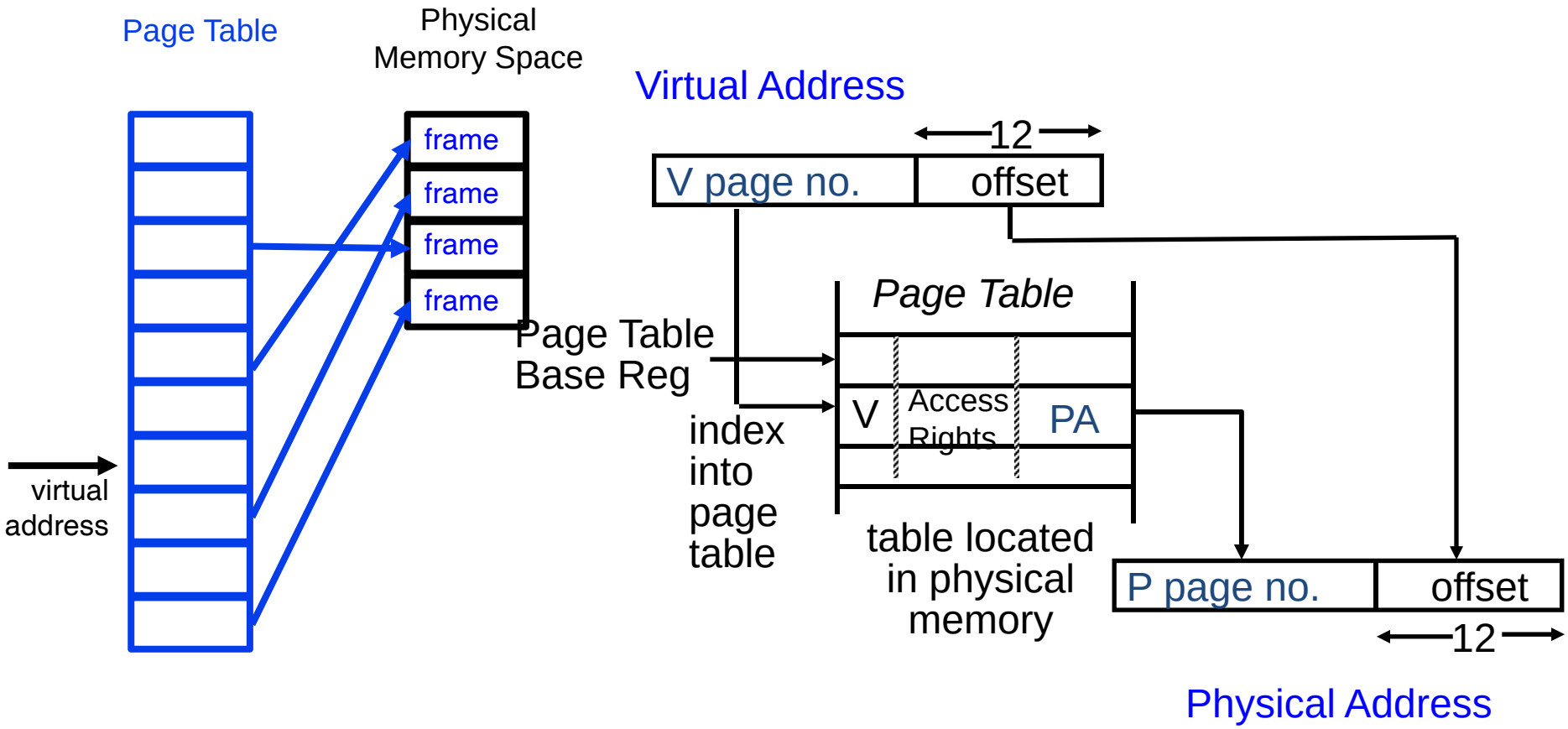
Page Size
4 Kbytes
16 Kbytes
64 Kbytes
256 Kbytes
1 Mbyte
4 Mbytes
16 Mbytes

OS manages the page table for each ASID

A page table is indexed by a virtual address

valid page table entry codes physical memory "frame" address for the page

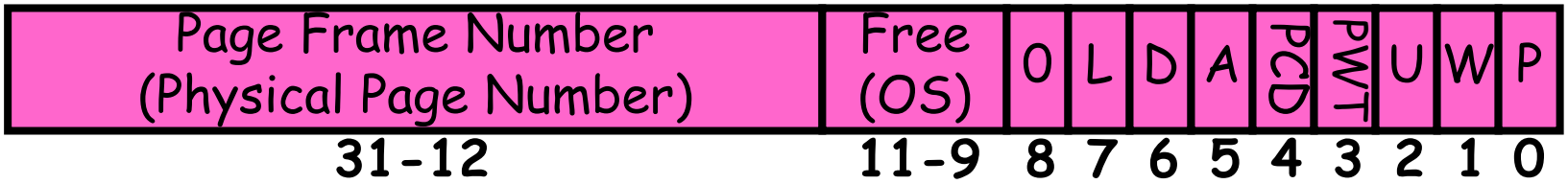
Details of Page Table



- **Page table maps virtual page numbers to physical frames ("PTE" = Page Table Entry)**
- **Virtual memory => treat memory ≈ cache for disk**

Page Table Entry (PTE)?

- What is in a Page Table Entry (or PTE)?
  - Pointer to next-level page table or to actual page
  - Permission bits: valid, read-only, read-write, write-only
- Example: Intel x86 architecture PTE:
  - Address same format previous slide (10, 10, 12-bit offset)
  - Intermediate page tables called “Directories”



P: Present (same as “valid” bit in other architectures)

W: Writeable

U: User accessible

PWT: Page write transparent: external cache write-through

PCD: Page cache disabled (page cannot be cached)

A: Accessed: page has been accessed recently

D: Dirty (PTE only): page has been modified recently

L: L=1⇒4MB page (directory only).
Bottom 22 bits of virtual address serve as offset

Cache vs. Virtual Memory

- Replacement
  - Cache miss handled by hardware
  - Page fault usually handled by OS
- Addresses
  - Virtual memory space is determined by the address size of the CPU
  - Cache space is independent of the CPU address size
- Lower level memory
  - For caches - the main memory is not shared by something else
  - For virtual memory - most of the disk contains the file system
    - File system addressed differently - usually in I/O space
    - Virtual memory lower level is usually called SWAP space

The same 4 questions for Virtual Memory

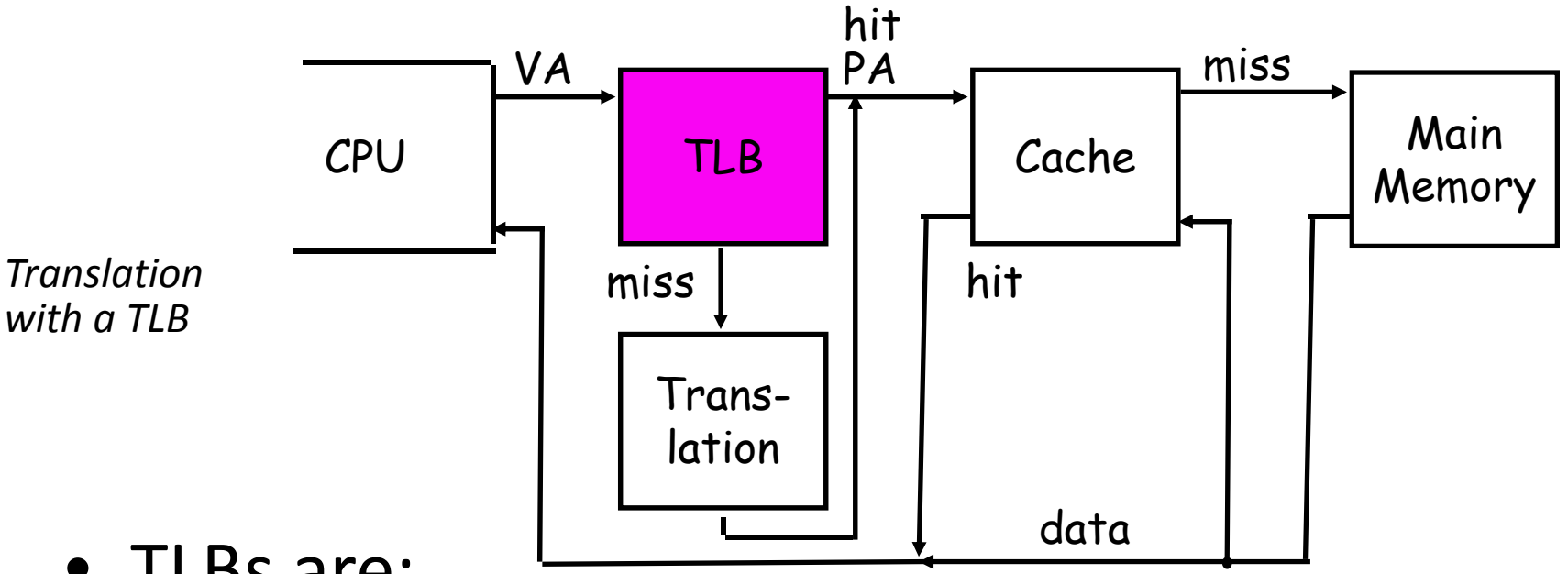
- Block Placement
  - Choice: lower miss rates and complex placement or vice versa
    - Miss penalty is huge, so choose low miss rate → place anywhere
    - Similar to fully associative cache model
- Block Identification - both use additional data structure
  - Fixed size pages - use a page table
  - Variable sized segments - segment table
- Block Replacement -- LRU is the best
  - However true LRU is a bit complex – so use approximation
    - Page table contains a use tag, and on access the use tag is set
    - OS checks them every so often - records what it sees in a data structure - then clears them all
    - On a miss the OS decides who has been used the least and replace that one
- Write Strategy -- always write back
  - Due to the access time to the disk, write through is silly
  - Use a dirty bit to only write back pages that have been modified

Techniques for Fast Address Translation

- Page table is kept in main memory (kernel memory)
  - Each process has a page table
- Every data/instruction access requires two memory accesses
  - One for the page table and one for the data/instruction
  - Can be solved by the use of a special fast-lookup hardware cache called associative registers or translation look-aside buffers (TLBs)
- If locality applies then cache the recent translation
  - TLB = translation look-aside buffer
  - TLB entry: virtual page no, physical page no, protection bit, use bit, dirty bit

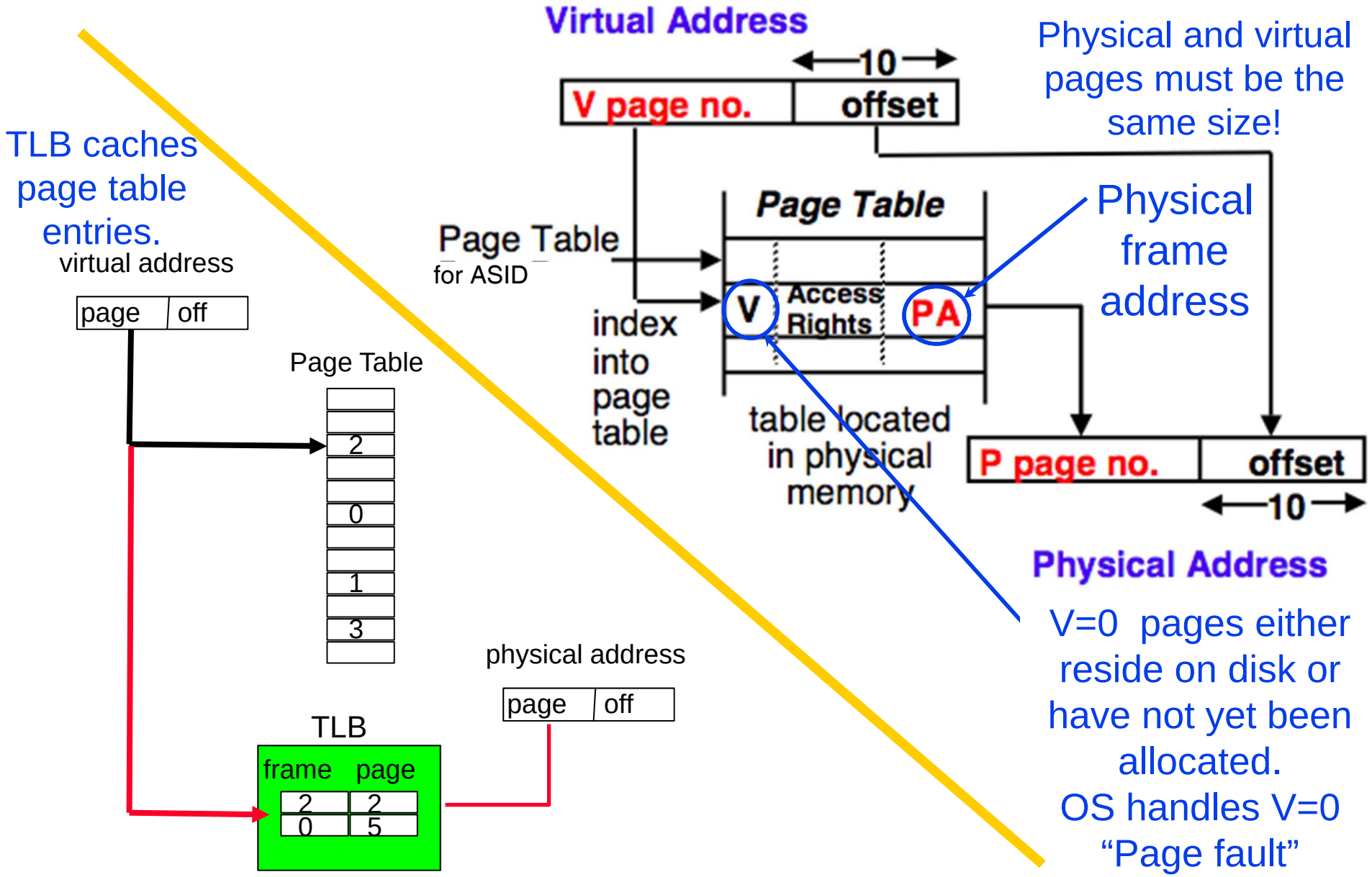
Translation Look-Aside Buffers

- Translation Look-Aside Buffers (TLB)
  - Cache on translations
  - Fully Associative, Set Associative, or Direct Mapped

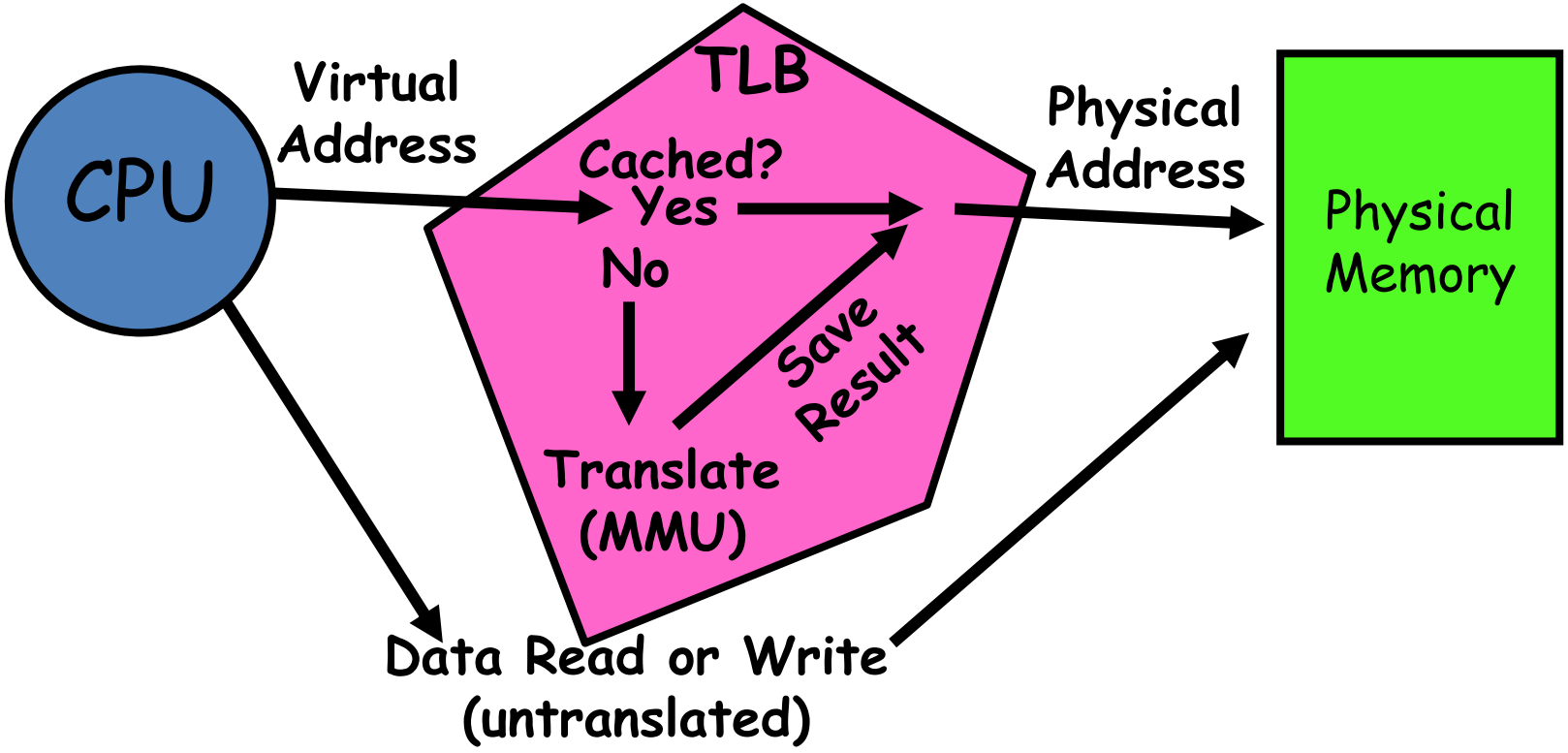


- TLBs are:
  - Small – typically not more than 128 – 256 entries
  - Fully Associative

The TLB Caches Page Table Entries



Caching Applied to Address Translation



Virtual Machines

- Supports isolation and security
- Sharing a computer among many unrelated users
- Enabled by raw speed of processors, making the overhead more acceptable

- Allows different ISAs and operating systems to be presented to user programs
  - "System Virtual Machines"
  - SVM software is called "virtual machine monitor" or "hypervisor"
  - Individual virtual machines run under the monitor are called "guest VMs"

Impact of VMs on Virtual Memory

- Each guest OS maintains its own set of page tables
  - VMM adds a level of memory between physical and virtual memory called “real memory”
  - VMM maintains shadow page table that maps guest virtual addresses to physical addresses
    - Requires VMM to detect guest’s changes to its own page table
    - Occurs naturally if accessing the page table pointer is a privileged operation