DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Computer Architecture

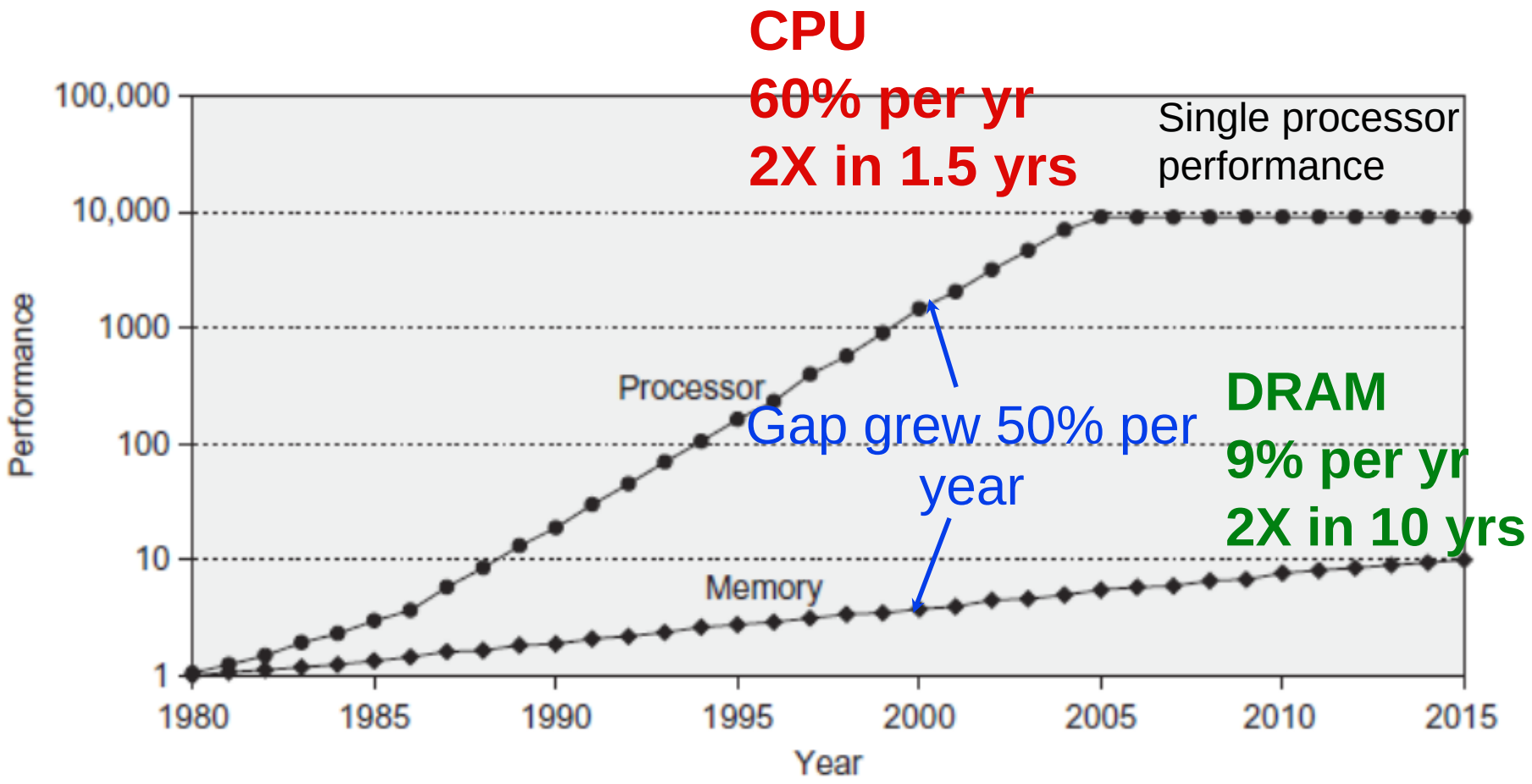
Lecture 2: Memory Hierarchy Design (Chapter 2, Appendix B)

Chih-Wei Liu 劉志尉

National Chiao Tung University

cwliu@twins.ee.nctu.edu.tw

Since 1980, CPU has outpaced DRAM…

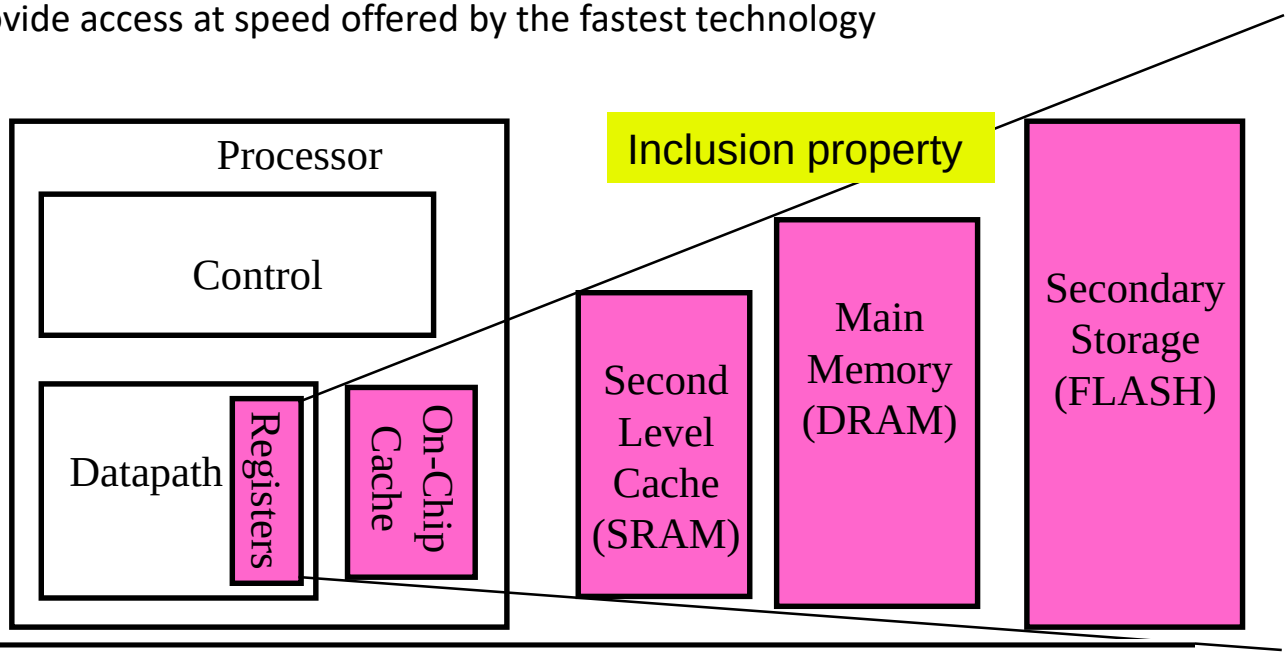


Introduction

- 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
- Cache: a safe place for hiding or storing things.

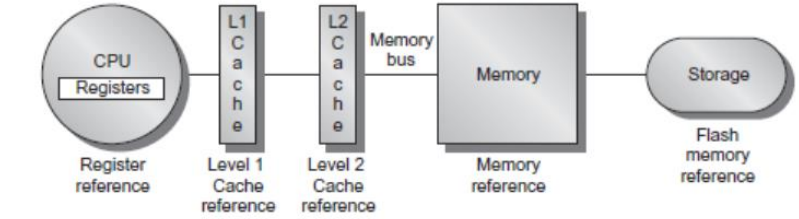
Memory Hierarchy

- 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

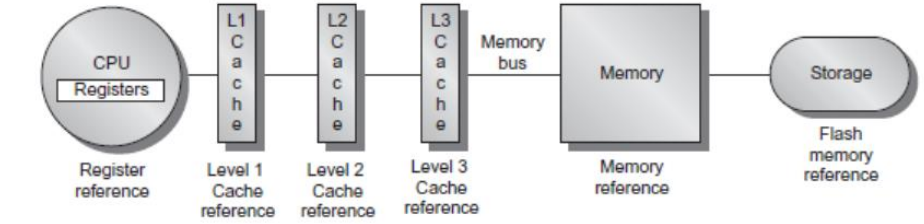


Level	1	2	3	4
Name	Registers	Cache	Main memory	Disk storage
Typical size	<4 KiB	32 KiB to 8 MiB	<1 TB	>1 TB
Implementation technology	Custom memory with multiple ports, CMOS	On-chip CMOS SRAM	CMOS DRAM	Magnetic disk or FLASH
Access time (ns)	0.1–0.2	0.5–10	30–150	5,000,000
Bandwidth (MiB/sec)	1,000,000–10,000,000	20,000–50,000	10,000–30,000	100–1000
Managed by	Compiler	Hardware	Operating system	Operating system
Backed by	Cache	Main memory	Disk or FLASH	Other disks and DVD

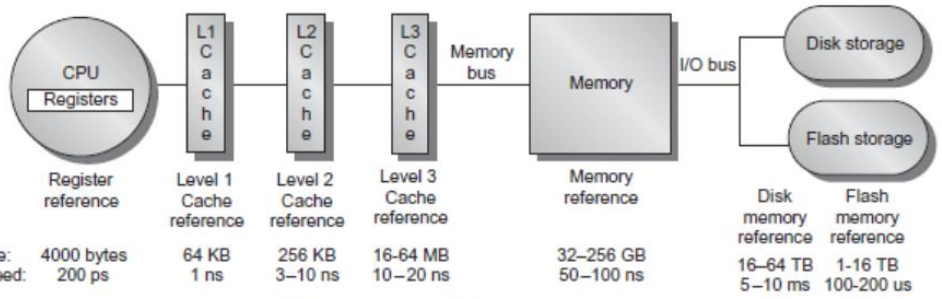
Memory Hierarchy Examples



(A) Memory hierarchy for a personal mobile device



(B) Memory hierarchy for a laptop or a desktop



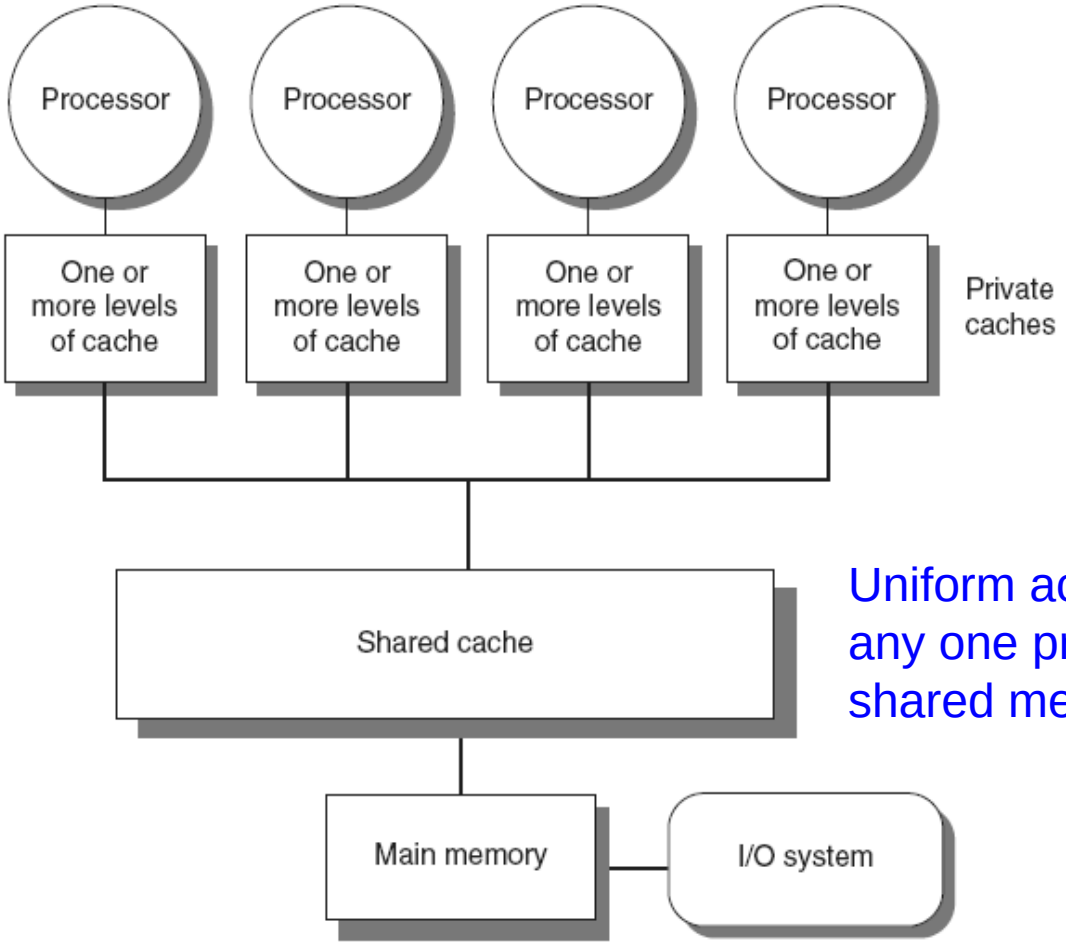
(C) Memory hierarchy for server

Memory Hierarchy Design

- Memory hierarchy design becomes more crucial with recent multicore processors:
  - Aggregate peak bandwidth grows with # cores:
    - Intel Core i7 6700 can generate two data references per core per clock
    - Four cores and 3.2 GHz clock
      - 25.6 billion 64-bit data references/s + 12.8 billion 128-bit instruction references/s = 409.6 GB/s!
    - DRAM bandwidth is approximately 8% of this (34.1 GB/s)
- Memory hierarchy for multicore processors?
  - More than optimizing AMAT (average memory access time)
  - Interconnection network
  - Coherent problem
  - Power issue

Possible Multi-core Architecture (1/2)

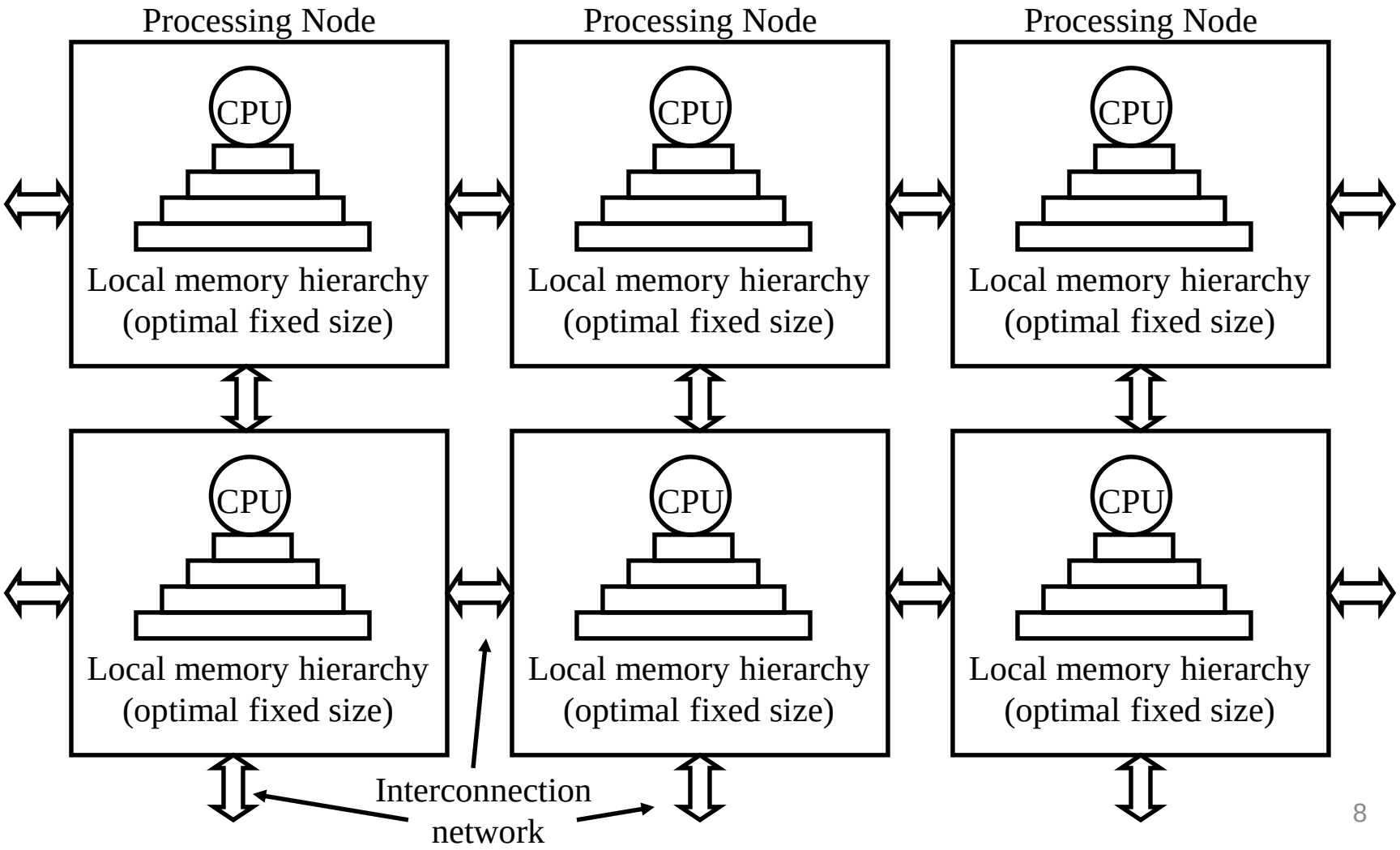
- Symmetric multiprocessors (SMP)



Uniform access time from any one processor to the shared memory

Possible Multi-core Architecture (2/2)

- Distributed shared-memory multiprocessors (DSM)

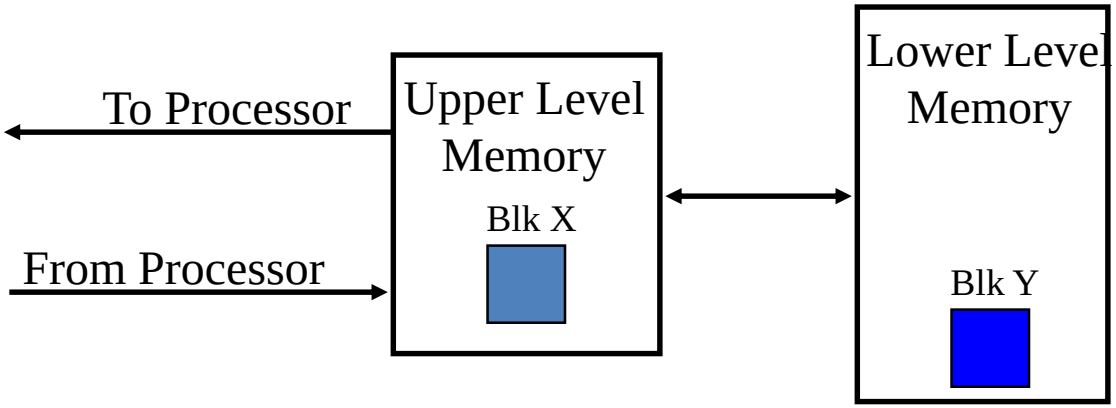


Memory Hierarchy Basics

- 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*
    - Multiple words per block
    - Takes advantage of spatial locality
  - Place block into cache in any location within its *set*, determined by address
    - A set is a group of blocks
    - (*block address*) MOD (*number of sets in a cache*)

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!)

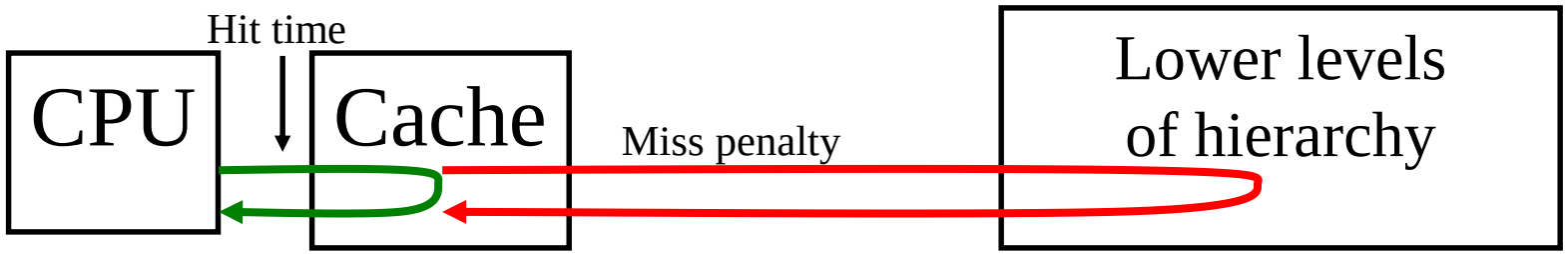


Cache Performance Formulas

(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



Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses

$$\text{Memory stall cycles} = \text{Number of misses} \times \text{Miss penalty}$$

$$= \text{IC} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

$$= \text{IC} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty}$$

    - More accurate cache misses:

$$\text{Memory stall clock cycles}$$

$$= \text{IC} \times \text{Reads per instruction} \times \text{Read miss rate} \times \text{Read miss penalty}$$

$$+ \text{IC} \times \text{Writes per instruction} \times \text{Write miss rate} \times \text{Write miss penalty}$$

Cache Performance Example

- Given
  - Cache miss rate = 1%
  - Miss penalty = 50 cycles
  - Base CPI (ideal cache) = 1
  - Load & stores are 50% of instructions

- Ideal case

$$\begin{aligned}\text{CPU execution time} &= (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle} \\ &= (\text{IC} \times \text{CPI} + 0) \times \text{Clock cycle} \\ &= \text{IC} \times 1.0 \times \text{Clock cycle}\end{aligned}$$

- Real case

$$\begin{aligned}\text{Memory stall cycles} &= \text{IC} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty} \\ &= \text{IC} \times (1 + 0.5) \times 0.01 \times 50 \\ &= \text{IC} \times 0.75\end{aligned}$$

- Performance ratio

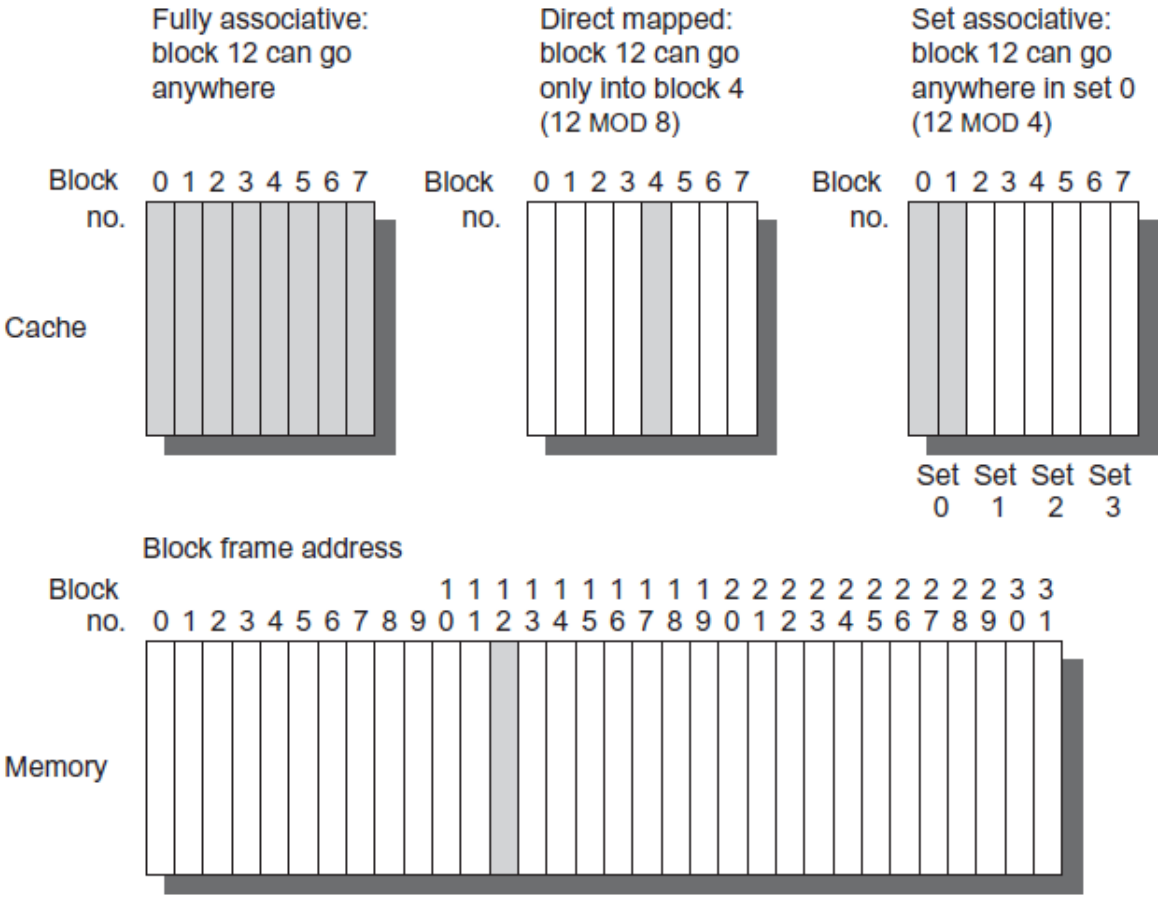
$$\begin{aligned}\frac{\text{CPU execution time}_{\text{cache}}}{\text{CPU execution time}} &= \frac{1.75 \times \text{IC} \times \text{Clock cycle}}{1.0 \times \text{IC} \times \text{Clock cycle}} \\ &= 1.75\end{aligned}$$

Four Questions for Memory Hierarchy

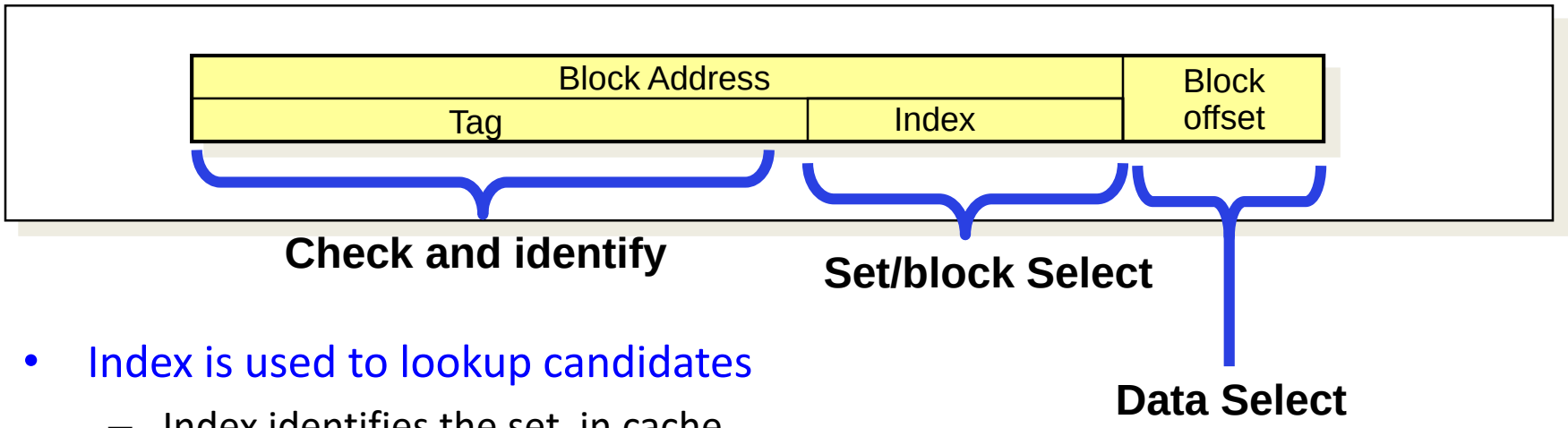
- Consider any level in a memory hierarchy.
  - Remember a block is the unit of data transfer.
    - Between the given level, and the levels below it
- The level design is described by four behaviors:
  - Block Placement*:*
    - Where can a block be placed in the upper level?
  - Block Identification*:*
    - How is a block found if it is in the upper level?
  - Block Replacement*:*
    - Which block should be replaced on a miss?
  - Write Strategy*:*
    - What happens on a write?

Q1: *Where can a block be placed in a Cache?*

- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. mapping = (Block address) MOD (Number of sets in a cache)
  - Direct mapping = (Block address) MOD (Number of blocks in a cache)



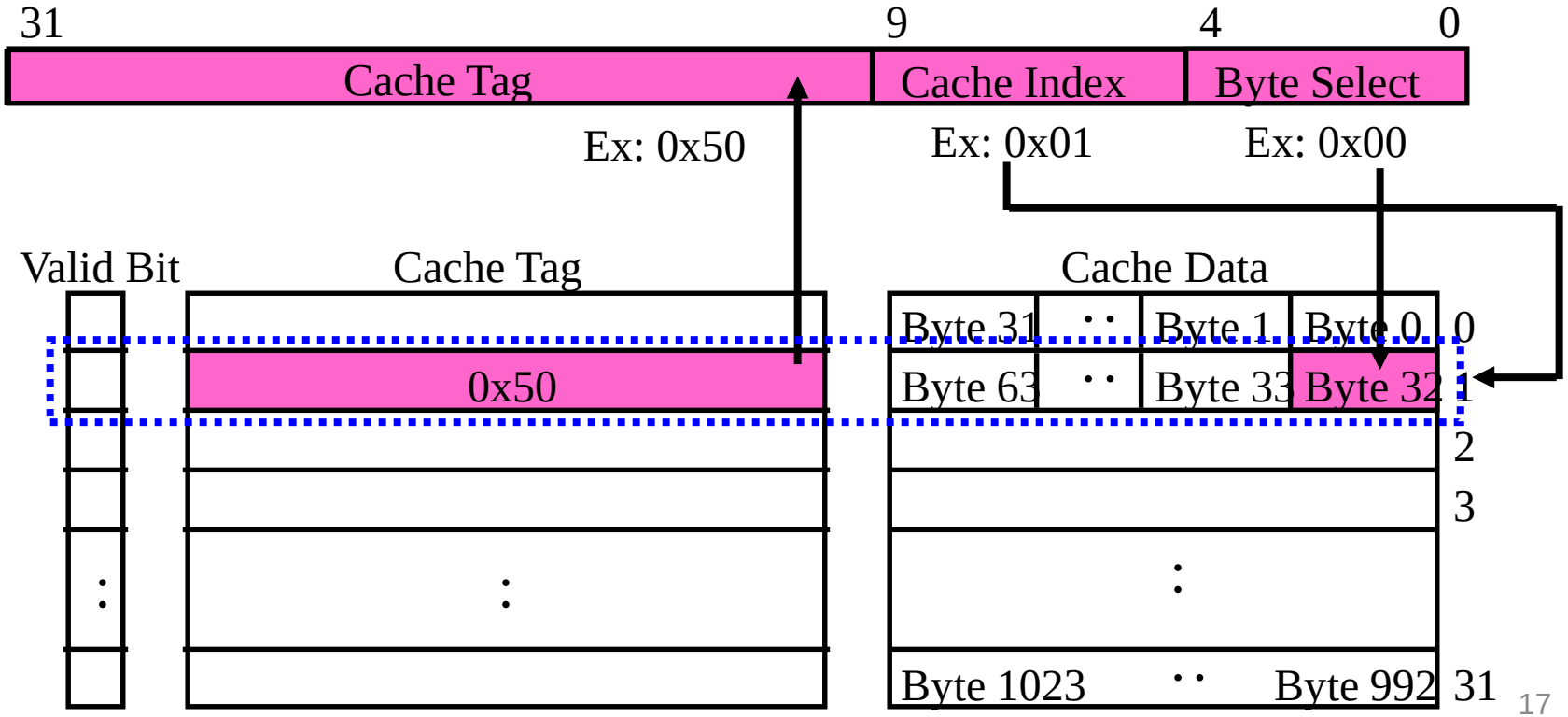
Q2: *How Is a Block Found If It Is in the Cache?*



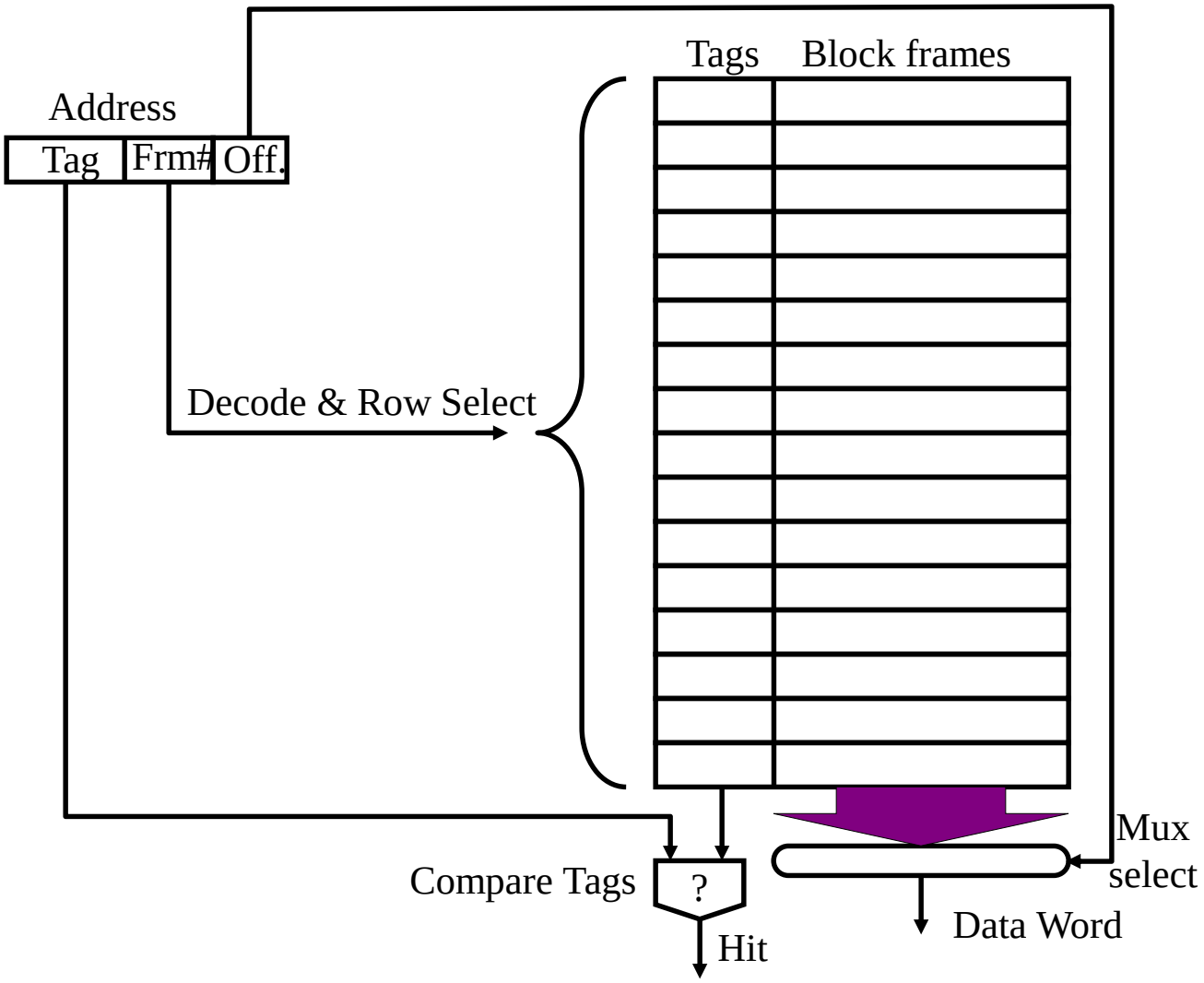
- Index is used to lookup candidates
  - Index identifies the set in cache
  - Fully associative cache has no index field
- Tag is used to identify actual copy
  - If no candidates match, then declare cache miss
- Block offset is used to select data within a block
  - Block is minimum quantum of caching
- Larger block size has distinct hardware advantages:
  - It exploits fast burst transfers from DRAM/over wide busses
- Disadvantages of larger block size?
  - Fewer blocks $\Rightarrow$ more conflicts. Can waste bandwidth

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

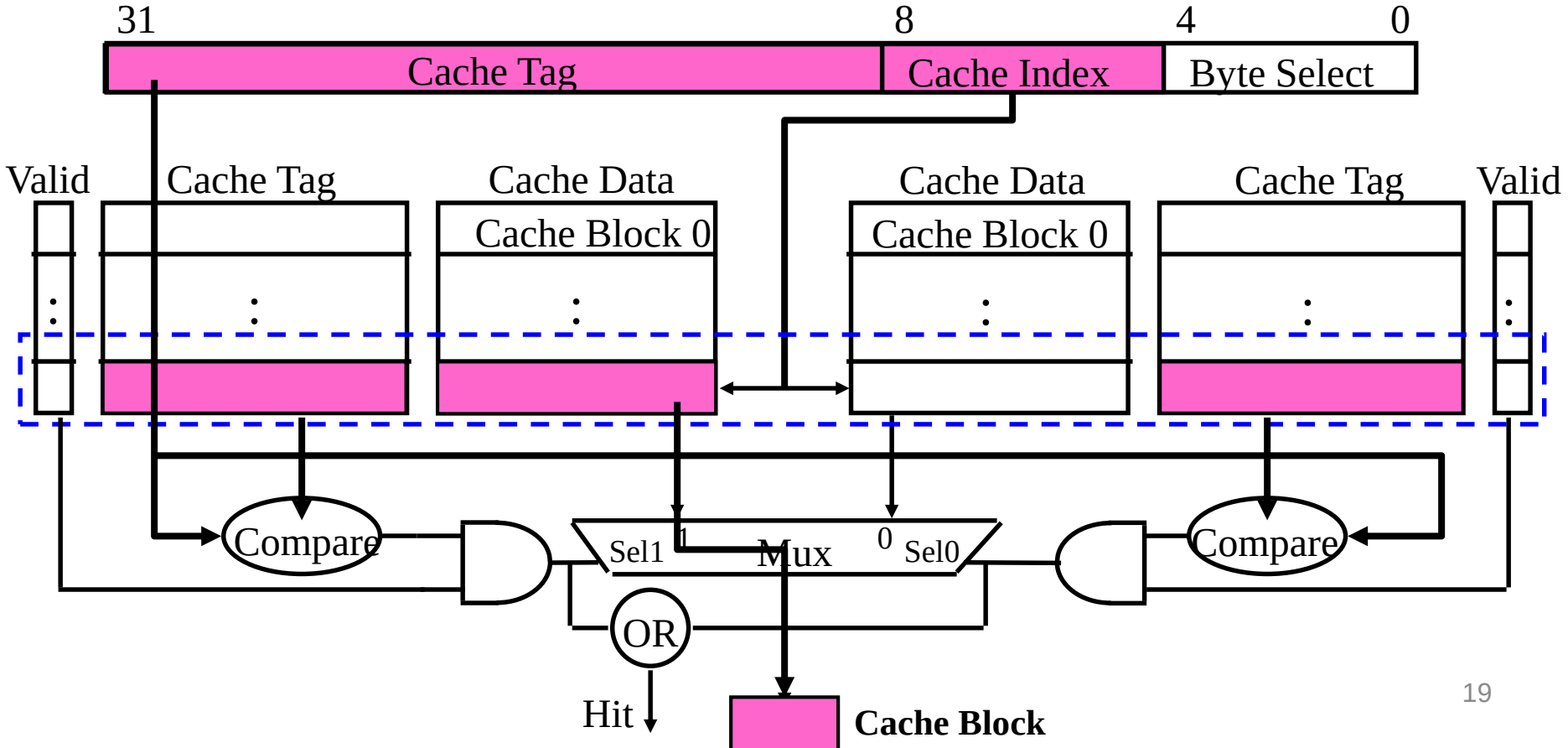


Direct-Mapped Cache Architecture



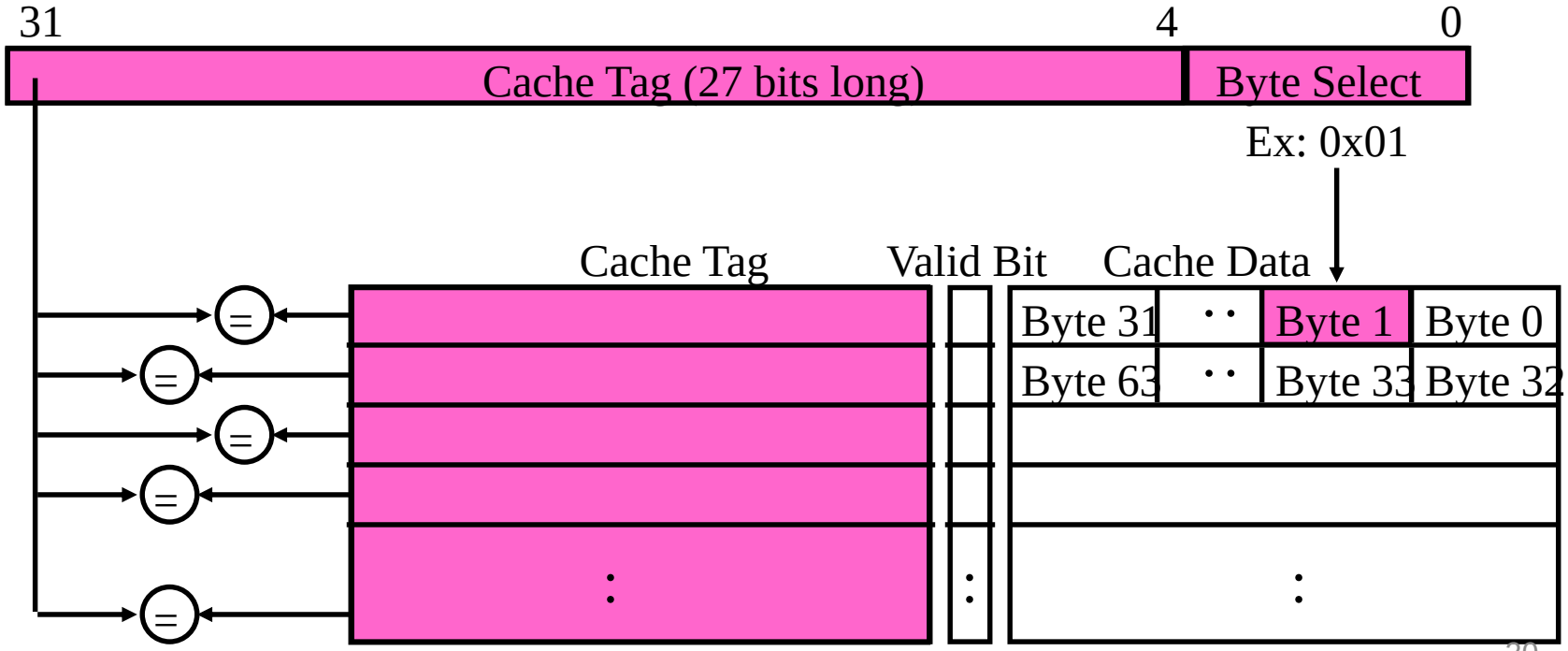
Set Associative Cache

- 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



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



Q3: *Which Block Should be Replaced on a Cache Miss?*

- Easy for direct-mapped cache
  - Only one block frame is checked for a hit, and only that block can be replaced. (There is no choice !!)
- Fully associative or set associative cache
  - Radom
    - Easy, but how well does it work?
  - Least recently used (LRU)
    - Relying on the locality property
    - Appealing, but hard to implement for high associativity
  - First in, first out (FIFO)
    - This approximates LRU by determining the oldest block.

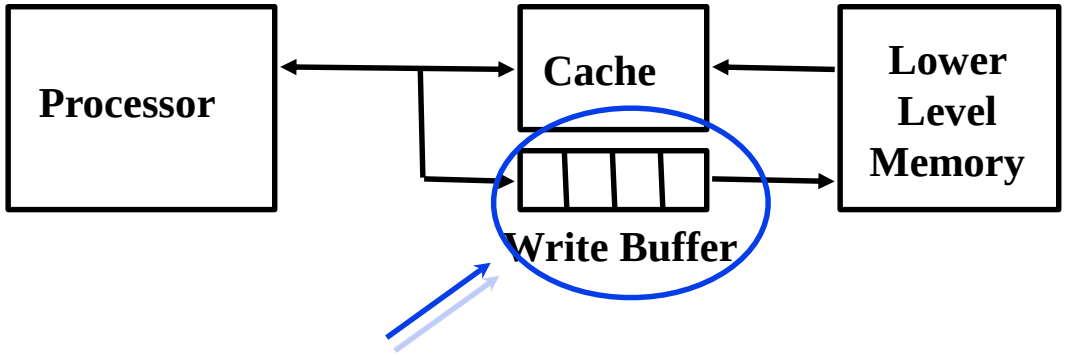
	Associativity								
	Two-way			Four-way			Eight-way		
Size	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16 KiB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KiB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KiB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

These data were collected for a block size of 64 bytes for the Alpha architecture using 10 SPEC2000 benchmarks.

Q4: *What Happens on a Write?*

- Processors traditionally wait for reads to complete but need not wait for writes
- Writes usually take longer than reads
  - Modifying a block cannot begin until the tag is checked to see if the address is a hit.
- Two strategies for writing to cache
  - *Write-through*
    - Immediately update lower levels of hierarchy.
  - *Write-back*
    - Only update lower levels of hierarchy when an updated (or dirty) block is replaced.
    - A dirty bit is commonly used.
  - Both strategies can use *write buffer* to make writes asynchronous

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

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").

Q: *What should happen on a write miss?*

- Two options on a write miss:
  - Write allocate
    - Write misses act like read misses
    - Allocate on miss: fetch the block (then overwrite it)
  - No-write allocate
    - Do not fetch the block, but update the portion of the block in the low-level memory.
    - Blocks stay out of the cache until the program tries to read the blocks (Since programs often write a whole block before reading it, e.g. initialization).
- Write-through caches often use no-write allocate.
- Write-back caches often use write allocate.

Example

- A fully associative write-back cache that starts empty. Following is a sequence of five memory operations (the address is in square brackets):

```
Write Mem[100];
Write Mem[100];
Read Mem[200];
Write Mem[200];
Write Mem[100].
```

- For no-write allocate (four misses and one hit)
  - The address 100 is not in the cache, so the first two writes will result in misses.
  - Address 200 is also not in the cache, so the read is also a miss.
  - The subsequent write to address 200 is a hit.
  - The last write to 100 is still a miss.
- For write allocate (two misses and three hits)
  - the first accesses to 100 and 200 are misses, and the rest are hits.

More on Cache Performance Metrics

- Can split access time into instructions & data:

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

- Another formula from execution time:

  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)

Cache Performance Example

Size (KiB)	Instruction cache	Data cache	Unified cache
8	8.16	44.0	63.0
16	3.82	40.9	51.0
32	1.36	38.4	43.3
64	0.61	36.9	39.4
128	0.30	35.3	36.2
256	0.02	32.6	32.9

Miss per 1000 instructions for instruction, data, and unified caches of different sizes.

- Which has the lower miss rate: a 16 KiB instruction cache with a 16 KiB data cache or a 32 KiB unified cache?
  - Assume write-through caches with a write buffer and ignore stalls due to the write buffer.
  - Assuming 36% of the instructions are data transfer instructions.
- Unified cache:
  - (43.3/(1000(1+0.36))) = 0.0318
- Separate I&D caches:
  - Instruction miss rate: 3.82/1000 = 0.004;
  - Data miss rate: (40.9/(1000*0.36)) = 0.114;
  - Overall miss rate:

$$\left(\frac{1000}{1000+360}\right)\times 0.004+\left(\frac{360}{1000+360}\right)\times 0.114\approx 0.0326$$

A 32 KiB unified cache has a slightly lower effective miss rate than two 16 KiB caches.

More examples can be found in the textbook (Appendix B) !!

Sources of Cache Misses

- Compulsory (cold-start miss, or first-reference miss):
  - "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
- Conflict (collision miss):
  - Multiple memory locations mapped to the same cache location
  - Increase associativity may be useful

- Coherence (Invalidation): other process (e.g., I/O) updates memory
  - Cache coherence protocol (will be discussed later).

Six Basic Cache Optimizations

$$\text{AMAT} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

- Six basics for reducing the miss rate, the miss penalty, and the hit time:
  - Larger block size to reduce miss rate
    - 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 to reduce miss rate
    - Reduces conflict misses
    - Increases hit time, increases power consumption
  - Multilevel caches to reduce miss penalty
    - 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 Size

- 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*

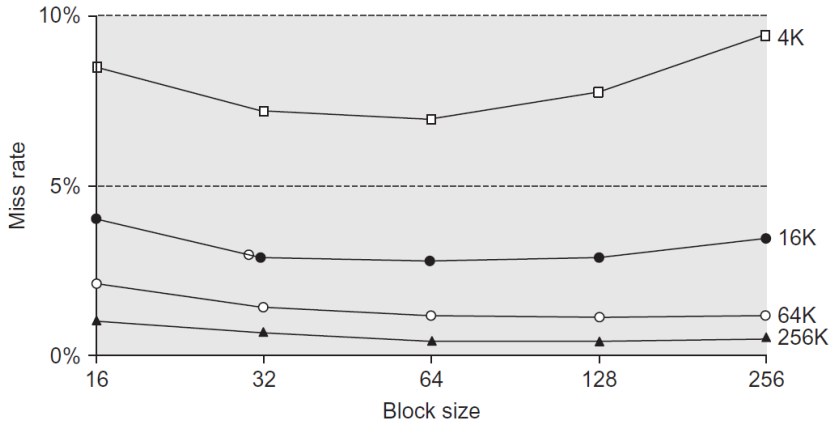


Figure B.10 Miss rate versus block size for five different-sized caches. Note that miss rate actually goes up if the block size is too large relative to the cache size. Each line represents a cache of different size. Figure B.11 shows the data used to plot these lines. Unfortunately, SPEC2000 traces would take too long if block size were included, so these data are based on SPEC92 on a DECstation 5000 (Gee et al. 1993).

	Cache size			
Block size	4K	16K	64K	256K
16	8.57%	3.94%	2.04%	1.09%
32	7.24%	2.87%	1.35%	0.70%
64	7.00%	2.64%	1.06%	0.51%
128	7.78%	2.77%	1.02%	0.49%
256	9.51%	3.29%	1.15%	0.49%

Figure B.11 Actual miss rate versus block size for the five different-sized caches in Figure B.10. Note that for a 4 KiB cache, 256-byte blocks have a higher miss rate than 32-byte blocks. In this example, the cache would have to be 256 KiB in order for a 256-byte block to decrease misses.

2. Large Caches

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

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

3. Higher Associativity

- 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 KiB)
- Higher associativity reduces conflict miss
  - 8-way set associative is for practical purposes as effective in reducing misses for these sized caches as fully associative.

- Greater associativity comes at the cost of increased hit time

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

- 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

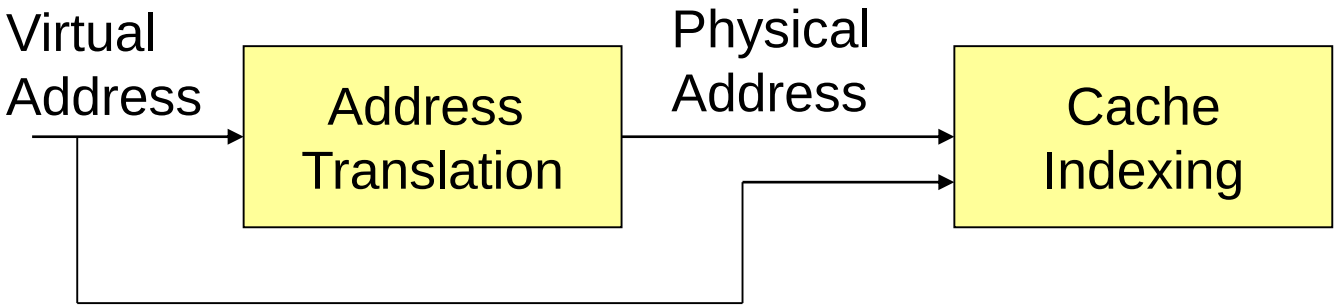
```
SW R3, 512(R0)     ;cache index 0
LW R1, 1024(R0)    ;cache index 0
LW R2, 512(R0)     ;cache index 0
```

The data in R3 are placed into the write buffer.

R2 = R3 ?

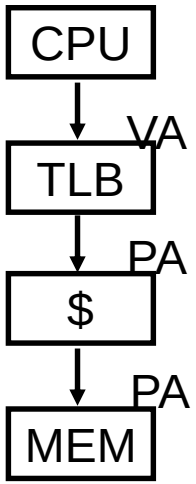
- Read miss waits until the write buffer empty → increase read miss penalty
- Check write buffer contents on a read miss, and if no conflicts, let the memory access continue read priority over write
- Write Back?
  - Read miss replacing dirty block
  - Normal: Write dirty block to memory, and then do 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

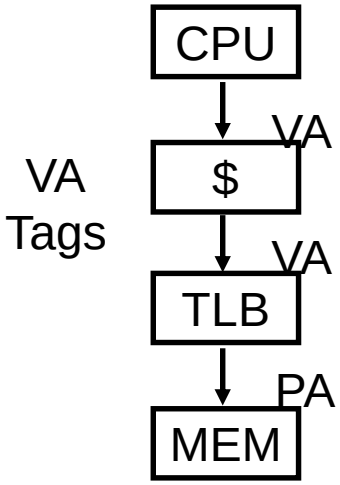


- Virtually addressed caches

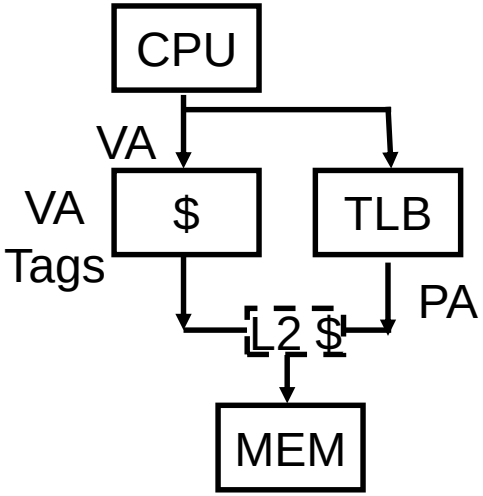
$ means cache



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

Concluding Remarks

Technique	Hit time	Miss penalty	Miss rate	Hardware complexity	Comment
Larger block size		–	+	0	Trivial; Pentium 4L2 uses 128 bytes
Larger cache size	–		+	1	Widely used, especially for L2 caches
Higher associativity	–		+	1	Widely used
Multilevel caches		+		2	Costly hardware; harder if L1 block size ≠ L2 block size; widely used
Read priority over writes		+		1	Widely used
Avoiding address translation during cache indexing	+			1	Widely used

Figure B.18 Summary of basic cache optimizations showing impact on cache performance and complexity for the techniques in this appendix. Generally a technique helps only one factor. + 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

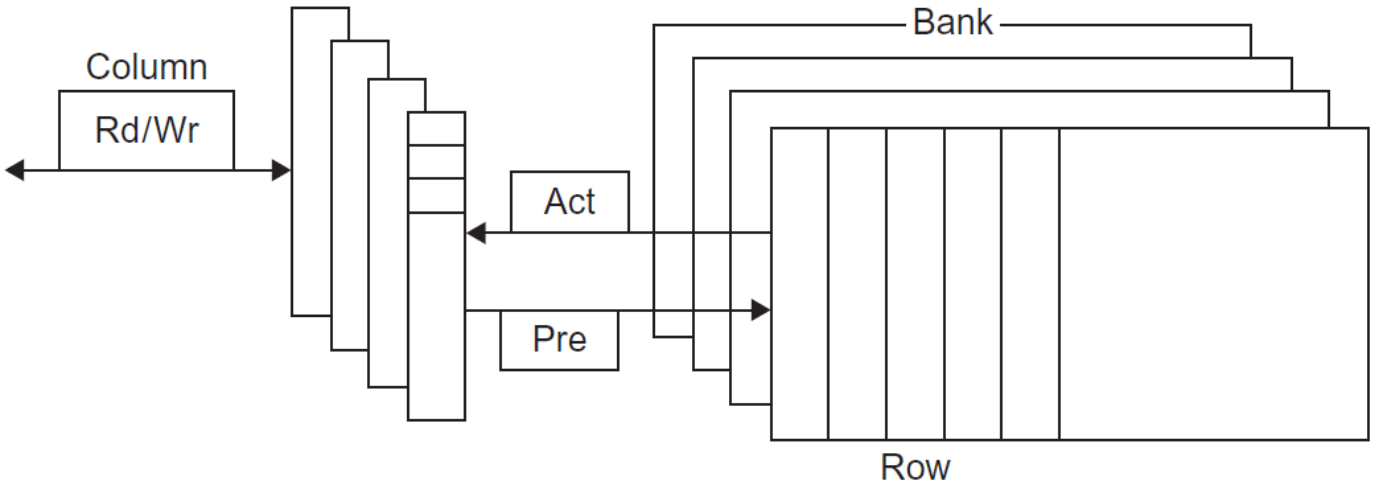
- Performance metrics
  - Latency is concern of cache
  - Bandwidth is concern of multiprocessors and I/O
  - Access time
    - Time between read request and when desired word arrives
  - Cycle time
    - Minimum time between unrelated requests to memory
- SRAM, a low latency memory, is used for cache
- DRAM, a high bandwidth memory with many banks, is used for main memory
- Flash is used as an alternative to hard disks.

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 for DRAM)

- 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
  - Multiplex address lines ➔ cutting # of address pins in half:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)
    - Row access strobe (RAS) first, then column access strobe (CAS)
      - Memory as a 2D matrix – rows go to a buffer；subsequent CAS selects subrow

Basic DRAM Organization



- Each bank consists of a series of rows.
- The ACT (activate) command opens a bank and a row. And, loads the row into a row buffer.
  - When the row is in the buffer, it can be transferred by successive column addresses at whatever the width of the DRAM is (typically 4, 8, or 16 bits in DDR4) or by specifying a block transfer and the starting address.
- The Pre (precharge) commend closes the bank and row and readies it for a new access.

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
   - Burst transfer mode (multiple transfers can occur without specifying a new column address) is allowed
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 1333 MHz
4. Multiple banks on each DRAM device
   - Improved bandwidth, not latency

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

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
DDR4	1333	2666	DDR4-2666	21300	PC21300

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

x 2

x 8

DIMMs: dual inline memory modules

Reducing Power Consumption in SDRAMs

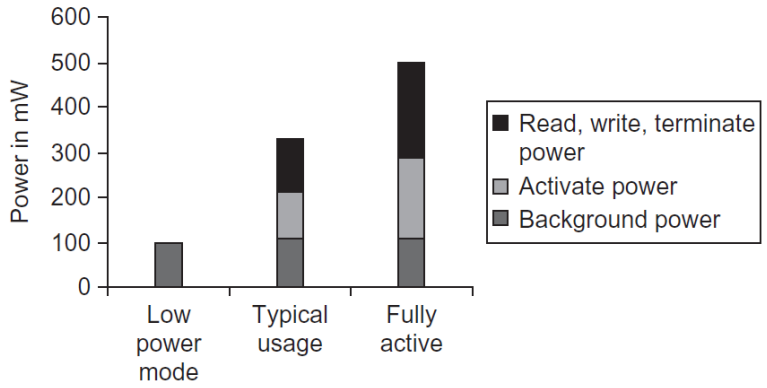


Figure 2.6 Power consumption for a DDR3 SDRAM operating under three conditions: low-power (shutdown) mode, typical system mode (DRAM is active 30% of the time for reads and 15% for writes), and fully active mode, where the DRAM is continuously reading or writing. Reads and writes assume bursts of eight transfers. These data are based on a Micron 1.5V 2GB DDR3-1066, although similar savings occur in DDR4 SDRAMs.

- Different operating modes for saving power
  - standby (or shutdown) mode, typical system mode, and fully active mode
- Lower voltage for reducing power
  - Both dynamic power used in a read or write and static or standby power depend on the operating voltage.
- Support power-down mode to disables the SDRAM, except for internal automatic refresh.

Graphics Memory (GDRAMs or GSDRAMs)

- GDDR5 is graphics memory based on DDR3
  - Earlier GDDRs is based on DDR2.
- Graphics memory:
  - Achieve 2-5X bandwidth per DRAM vs. DDR3
    - Wider interfaces (32 vs. 4, 8, or 16 bit)
    - Higher clock rate
      - Possible because they are attached via soldering (i.e. connecting directly to the GPU) instead of socketed DIMM modules on the board

Packaging Innovation: Stacked or Embedded DRAMs

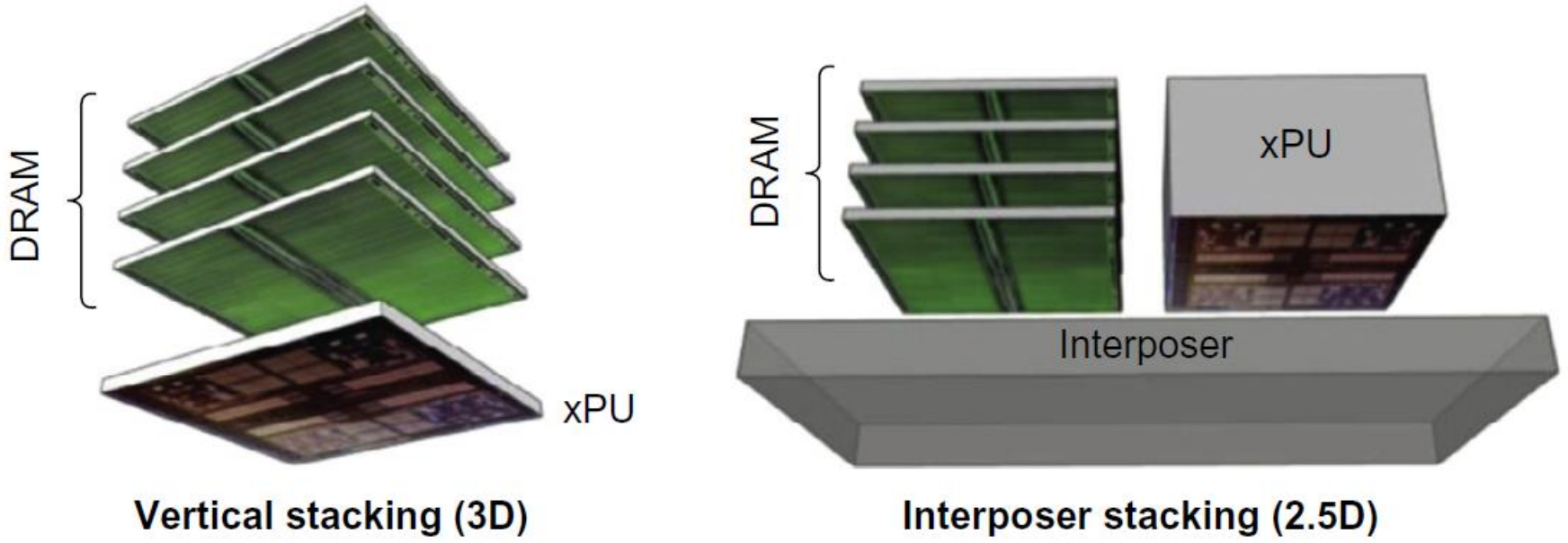


Figure 2.7 Two forms of die stacking. The 2.5D form is available now. 3D stacking is under development and faces heat management challenges due to the CPU.

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:
  - One type of EEPROM (electronically erasable programmable ROM)
    - NAND Flash (denser) vs. NOR Flash (faster)
    - Must be erased before it is overwritten
    - Nonvolatile, can use as little as zero power
    - Reads at almost DRAM speeds, but writes 10 to 100 times slower
  - NAND Flash:
    - Reads are sequential and read an entire page (0.5 to 4 KiB)
      - Long delay (~25 us) to access the first byte, but supporting at ~40 MiB/s for subsequent bytes for the rest of page
      - c.f. SDRAM: 40 ns for first byte, 4.8 GB/s for subsequent bytes for the rest of row
    - 2 KiB transfer: 75 us vs 500 ns for SDRAM, 150X slower
    - 300 to 500X faster than magnetic disk

New Memory Technology

- Phase-change/Memristor memory (PCM)
  - Xpoint memory, Micron (2017)
  - Possibly 10X improvement in write performance and 2-3X improvement in read performance than NAND Flash.
- Enhancing Dependability in Memory Systems
  - 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

10 Advanced Cache Optimizations

1. Reducing hit time
- Small and simple caches
- Way prediction

2. Increasing cache bandwidth
- Pipelined caches
- Multibanked caches
- Nonblocking caches

3. Reducing Miss Penalty
- Critical word first
- Merging write buffers

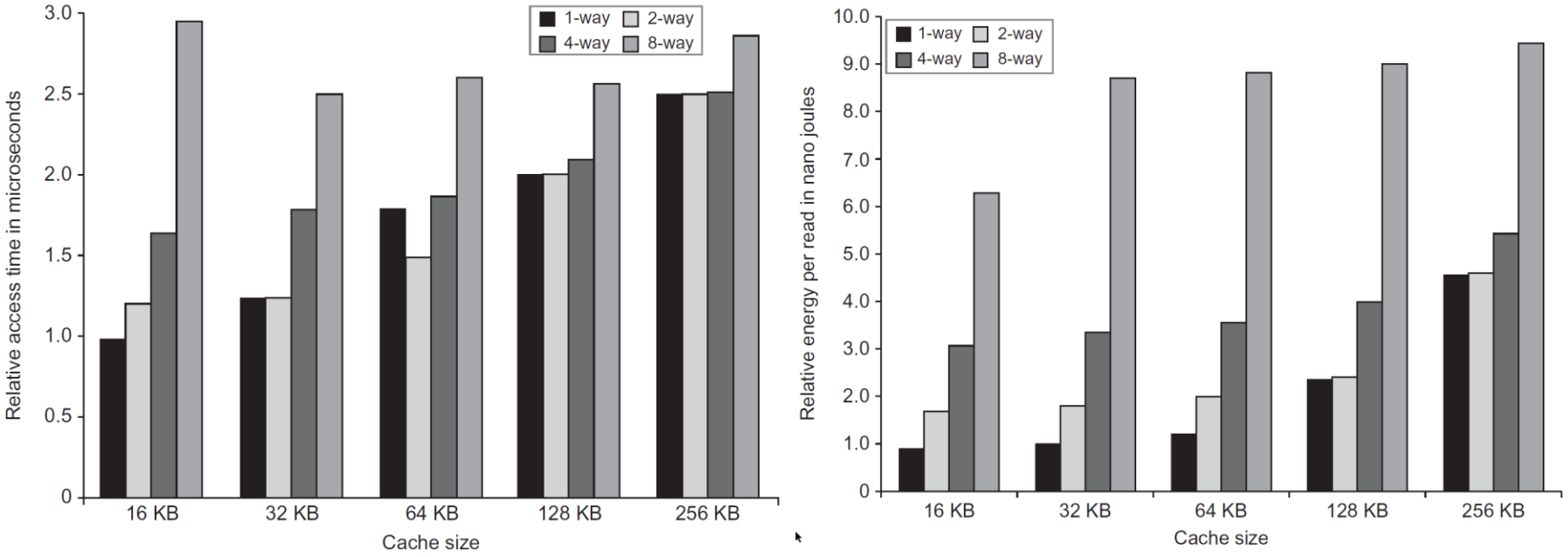
4. Reducing Miss Rate
- Compiler optimizations

5. Reducing miss penalty or miss rate via parallelism
- Hardware prefetching
- Compiler prefetching

1. *Small and Simple L1 Cache to Reduce Hit Time and Power*

- Critical timing path in a cache hit:
  - 3-step process: addressing tag memory (using index), then comparing tags, then selecting correct set
    - Index tag memory and then compare take time
- Direct-mapped caches can overlap the tag compare with the transmission of data
    - Since there is only on choice
    - Effectively reducing hit time.
- Lower associativity reduces power because fewer cache lines are accessed
- Limited size for the L1 cache
  - Fast clock rate and low power consumption

Access Time/Energy vs. Cache Size and Associativity



These data come from the CACTI model 6.5 by Tarjan et al. (2005).

CACTI is a program to estimate the access time and energy consumption of alternative cache structures on CMOS microprocessors.

2. *Way Prediction to Reduce Hit Time*

- How to combine fast hit time of direct mapped cache and have the lower conflict misses of 2-way SA cache?
- Way prediction: keep extra bits in cache to predict the "way" 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 ⇒ check the other blocks for matches in next clock cycle

- First used on MIPS R10000 in mid-90s and, was used in several ARM processors now.
- Accuracy: >90% for two-way；>80% for four-way；I-cache is better than D-cache.
- Can extend to predict block as well: way selection, but adds significant time on a way misprediction.
- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles

3. *Increasing Bandwidth by Pipeline Access and Multibanked Caches*

- These optimizations are the dual to the *superpipelined* and *superscalar* processors to increasing instruction throughput.
  - Pipelining L1 allows a higher clock cycle
    - Pentium: 1 cycle；Pentium Pro – Pentium III: 2 cycles；Pentium 4 – Core i7: 4 cycles.
  - Deeply pipelining leads to a greater penalty on control hazard
    - A high performance branch prediction scheme is necessary.
  - Organize cache as independent banks to support simultaneous accesses (rather than a single monolithic block)
    - Cortex-A8 supports 1-4 banks for L2； Core i7 supports 4 banks for L1 and 8 banks for L2
  - Banking works best when accesses naturally spread themselves across banks
    - *Sequential interleaving access*

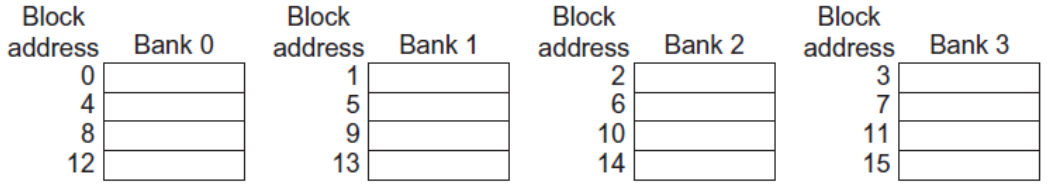
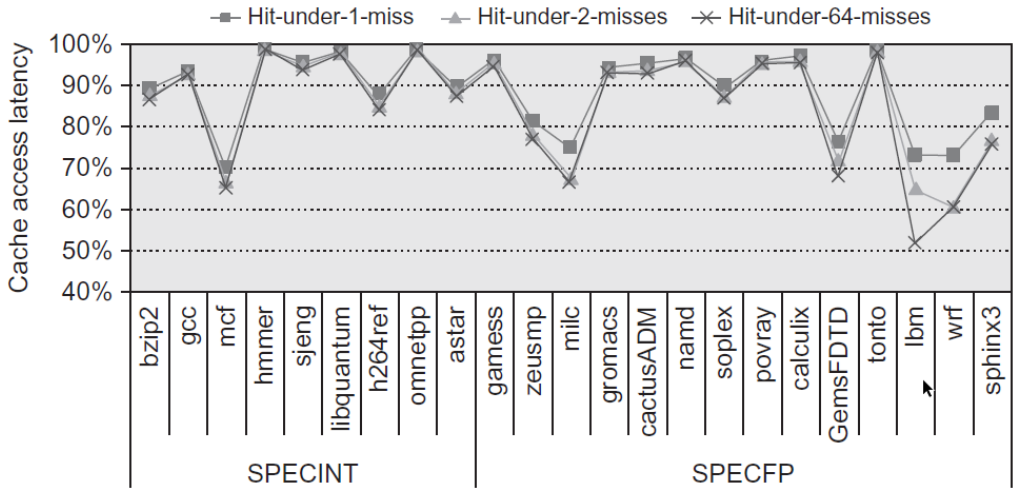


Figure 2.10 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per block, each of these addresses would be multiplied by 64 to get byte addressing.

Mapping of addresses to banks affects behavior of memory system

4. *Nonblocking Caches to Increase Bandwidth*

- *Non-blocking cache* or *lockup-free cache* allow data cache to continue to supply cache hits during a miss for *out-of-order superscalar* processor.
  - This requires multiple memory banks
  - The "*hit under miss*" reduces the effective miss penalty by working during miss vs. ignoring CPU requests
  - The "*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.



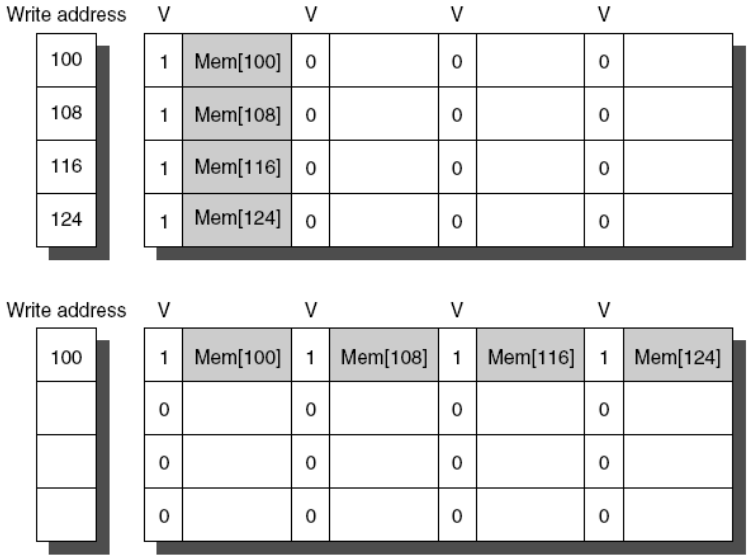
SPECCPU2006 @ Intel i7

5. *Critical Word First and Early Restart to Reduce Miss Penalty*

- Do not wait for full block to be loaded before restarting processor
  - *Critical word First* – Request the missed word first and let the processor continue execution while filling the rest of the words in the block. (Also called wrapped fetch)
  - *Early restart* – Fetch the words within the block in normal order, but as soon as the requested word 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
    - E.g., the next reference is the sequential word, so not clear if benefit

6. *Merging Write Buffer to Reduce Miss Penalty*

- 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 and update the write buffer.



No write merging

With write merging

- Might reduce stalls due to full write buffer
- Do not apply to I/O addresses

7. Compiler Optimizations to Reduce Miss Rate

- The hardware designer's favorite solution (without any hardware changes)
  - Profiling to look at conflicts (using tools they developed), and reorder procedures so as to reduce conflict misses in memory
- For data access
  - *Loop Fusion*: Improve spatial locality by combining 2 independent loops that have same looping and some variables overlap
  - *Loop Interchange*: Improve spatial locality by swapping nested loops to access data stored in memory in sequential order
  - *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 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

Loop Interchange Example

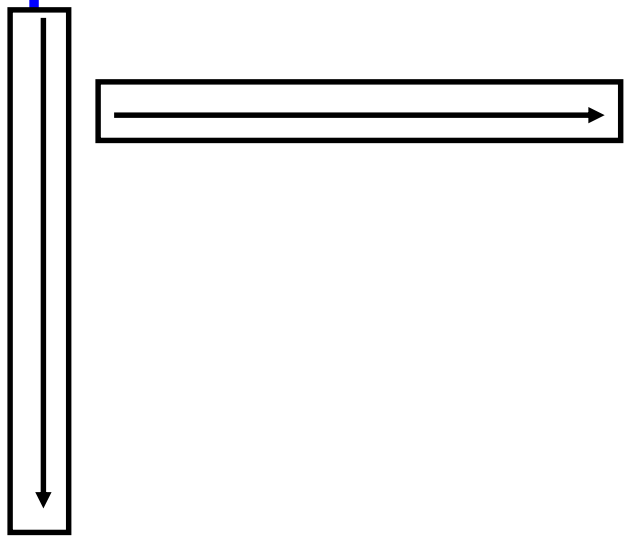
Assume x is a two-dimensional array of size [5000,100] stored in row-major order, i.e., x[i,j] and x[i,j+1] are adjacent.

```
/* 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];
```

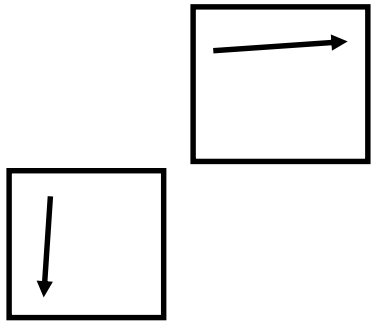
➔ Sequential accesses instead of striding access through memory every 100 words; improved spatial locality

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

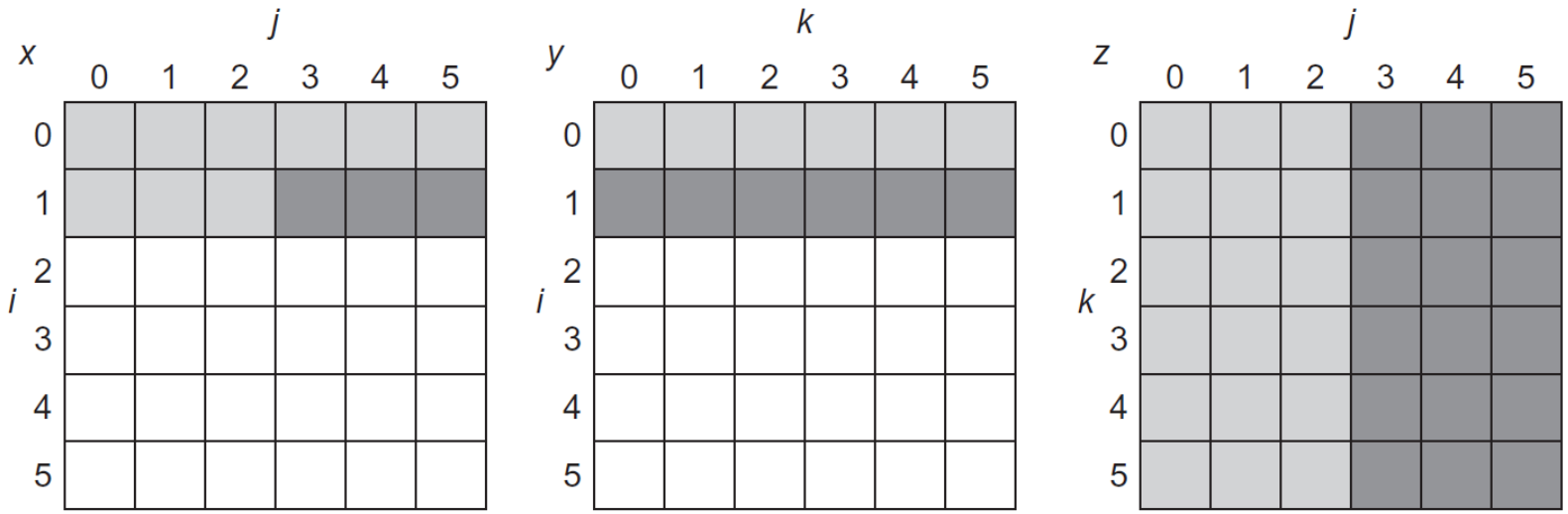


Figure 2.13 A snapshot of the three arrays `x`, `y`, **and** `z` **when** $N = 6$ **and** `i` $= 1$. The age of accesses to the array elements is indicated by shade: white means not yet touched, light means older accesses, and dark means newer accesses. The elements of `y` and `z` are read repeatedly to calculate new elements of `x`. The variables `i`, `j`, and `k` are shown along the rows or columns used to access the arrays.

Before….

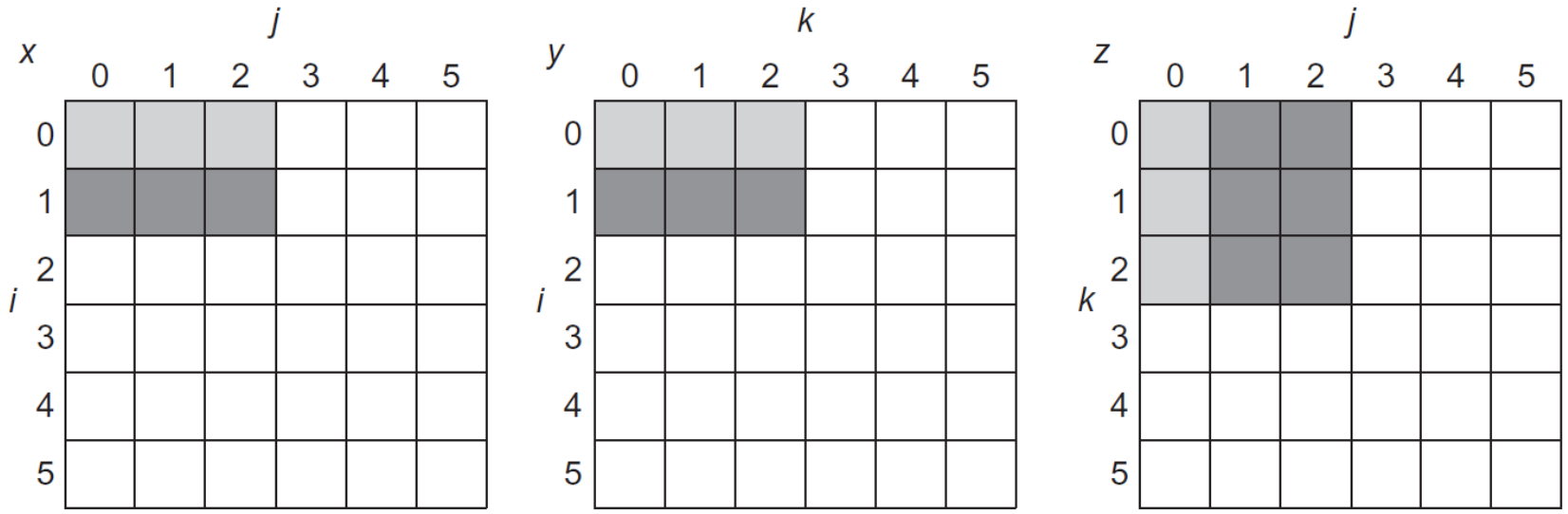
Blocking Example

```
/* 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?

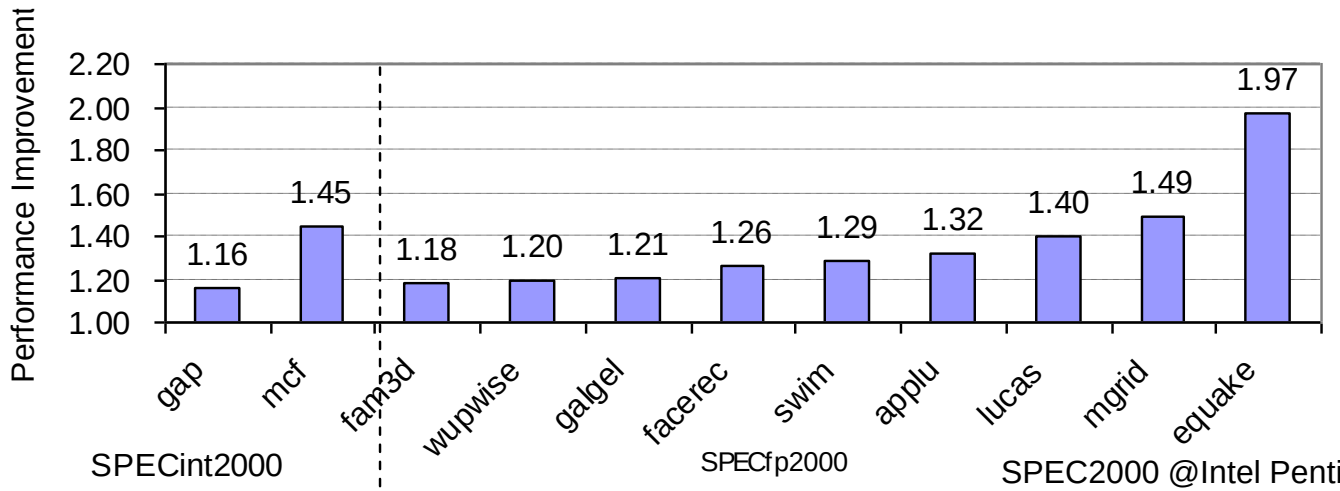
The Age of Accesses to x, y, z when B=3

After…. the smaller number of elements are accessed



8. *Hardware Prefetching of Instructions & Data to Reduce Miss Penalty or Miss Rate by*

- 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
  - Similar to instruction perfecting approach. Pentium 4 can prefetch data into L2 cache from up to 8 streams from 8 different 4 KB pages



9. *Compiler-Controlled Prefetching to Reduce Miss Penalty or Miss Rate*

- Prefetch instruction is inserted (in compiler time) before data is needed
- *Nonfaulting (or nonbinding) prefetch*: prefetch doesn't cause an exception

- 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? (see Example p 112-113).
  - Higher superscalar reduces difficulty of issue bandwidth
  - Combine with software pipelining and loop unrolling

10. *Using HBM to Extend the Memory Hierarchy*

- HBM (high bandwidth memory) packaging:
  - In-package DRAMs L4 cache: from 128 MiB to 1 GiB and mores
- Large DRAM-based cache suffers from an issue of “where do the tags reside?”
  - Smaller blocks require substantial tag storage
  - Larger blocks are potentially inefficient
    - Fragmentation problem
    - More conflict and consistency misses
- L-H cache (proposed by Loh and Hill in 2011)
- Alloy cache (proposed by Qureshi and Loh in 2012)

LH-Cache vs. Alloy Cache

- Long hit time for two accesses to L4 DRAM cache (one for the tags and one for the data itself)
- Two solutions of preventing from two DRAM accesses:
  - LH-Cache
    - Place the tags and the data in the same row in the HBM SDRAM.
      - One can access the tag first. If it is a hit, then use a column access to choose the correct word.
      - Hit requires a CAS
    - Each SDRAM row is a block index
    - Each row contains set of tags and 29 data segments
  - Alloy cache
    - Mold the tag and data together and use a direct mapped cache structure

- Unfortunately, in both schemes, misses require two full DRAM accesses:
  - one to get the initial tag and a follow-on access to the main memory

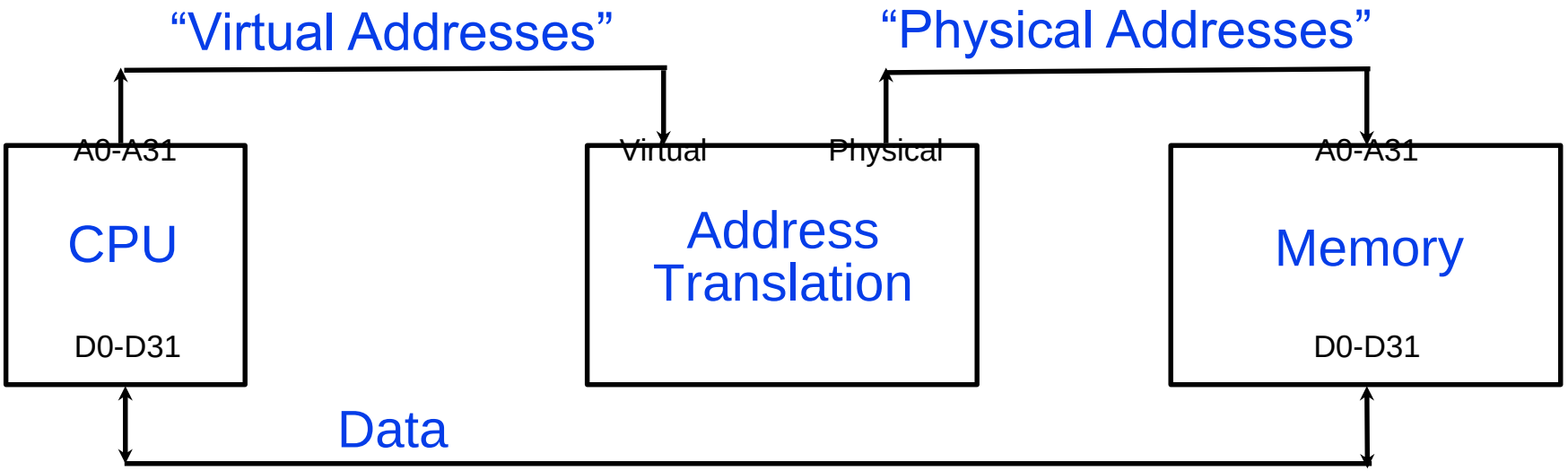
Cache Optimization Summary

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 & banked caches	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
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
HBM as additional level of cache		+/–	–	+	+	3	Depends on new packaging technology. Effects depend heavily on hit rate improvements

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

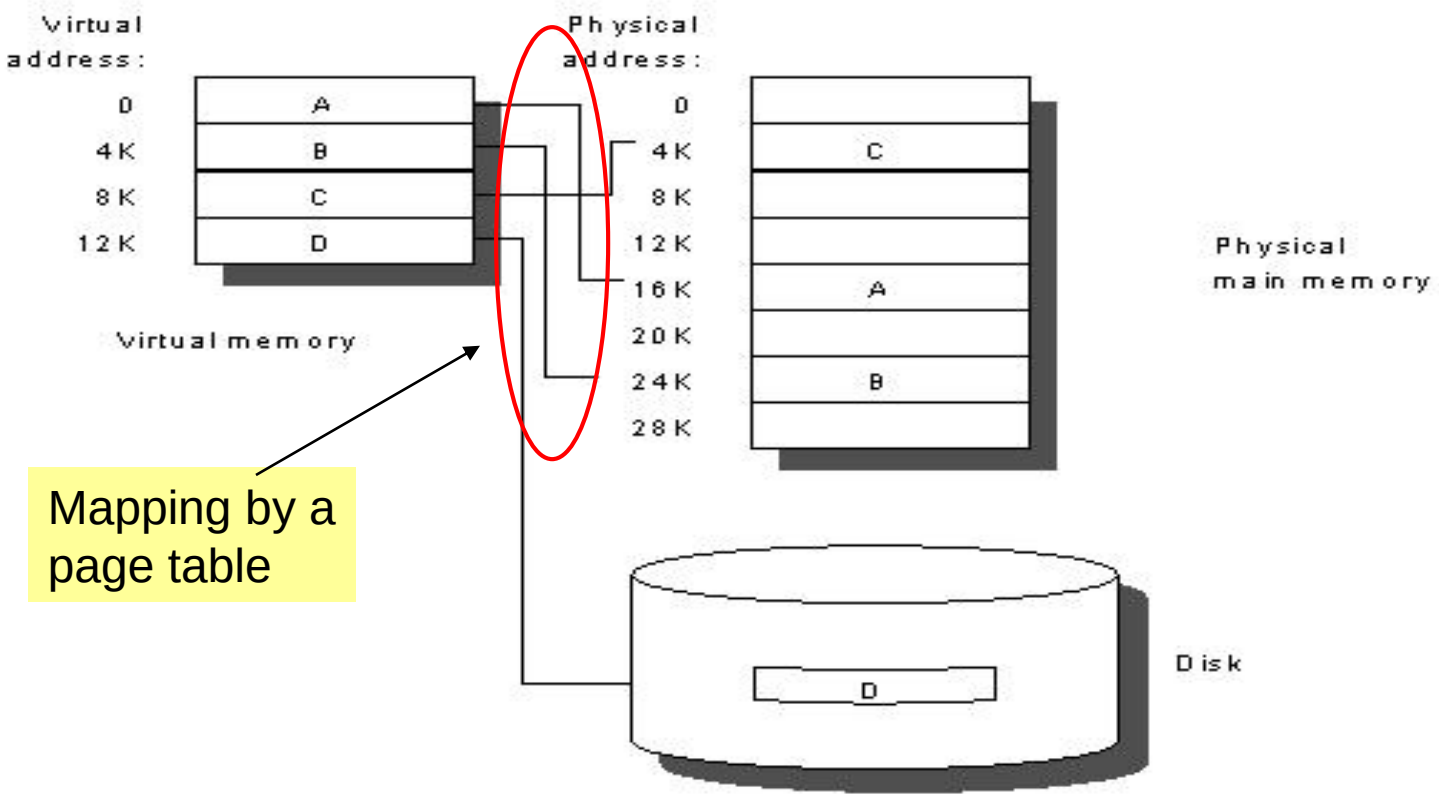


User programs run in an standardized virtual address space

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

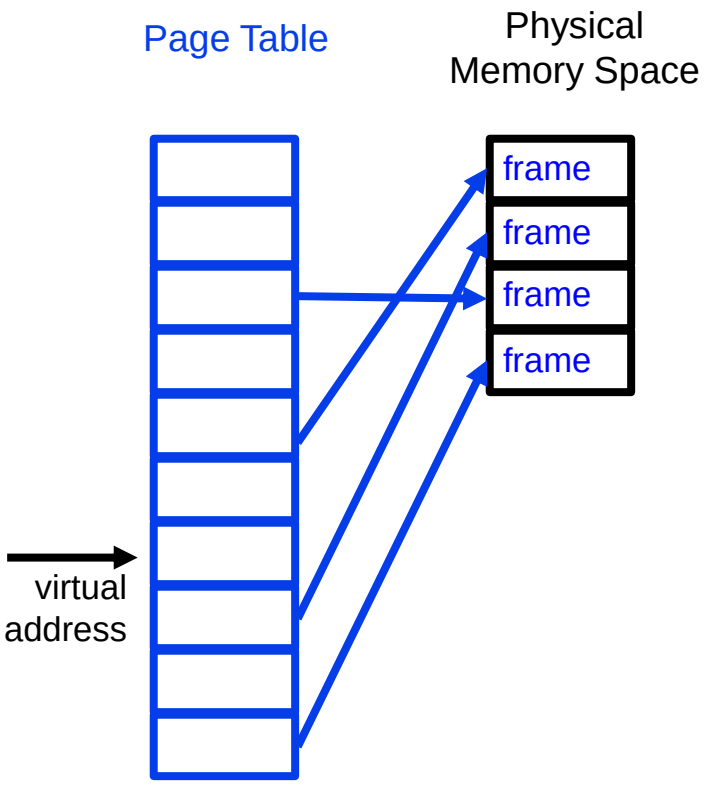
3 Advantages of VM

- 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):

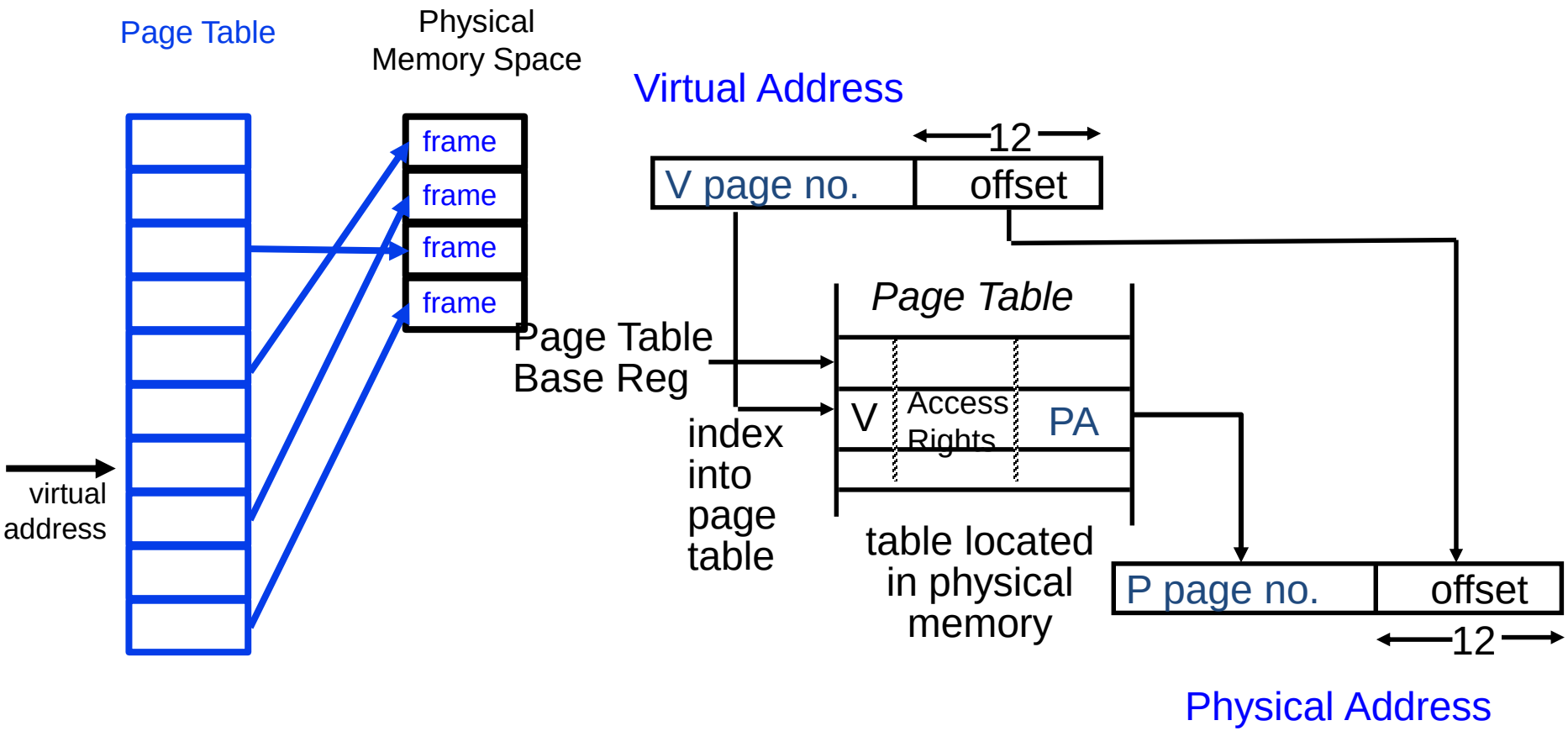
Page Size
4 Kbytes
16 Kbytes
64 Kbytes
256 Kbytes
1 Mbyte
4 Mbytes
16 Mbytes

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

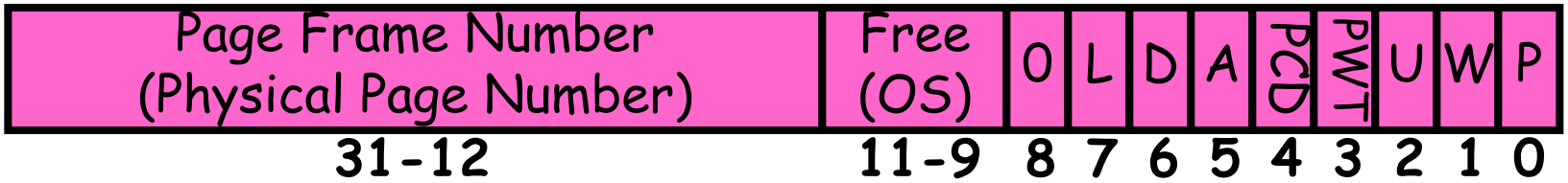
Details of Page Table



- **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

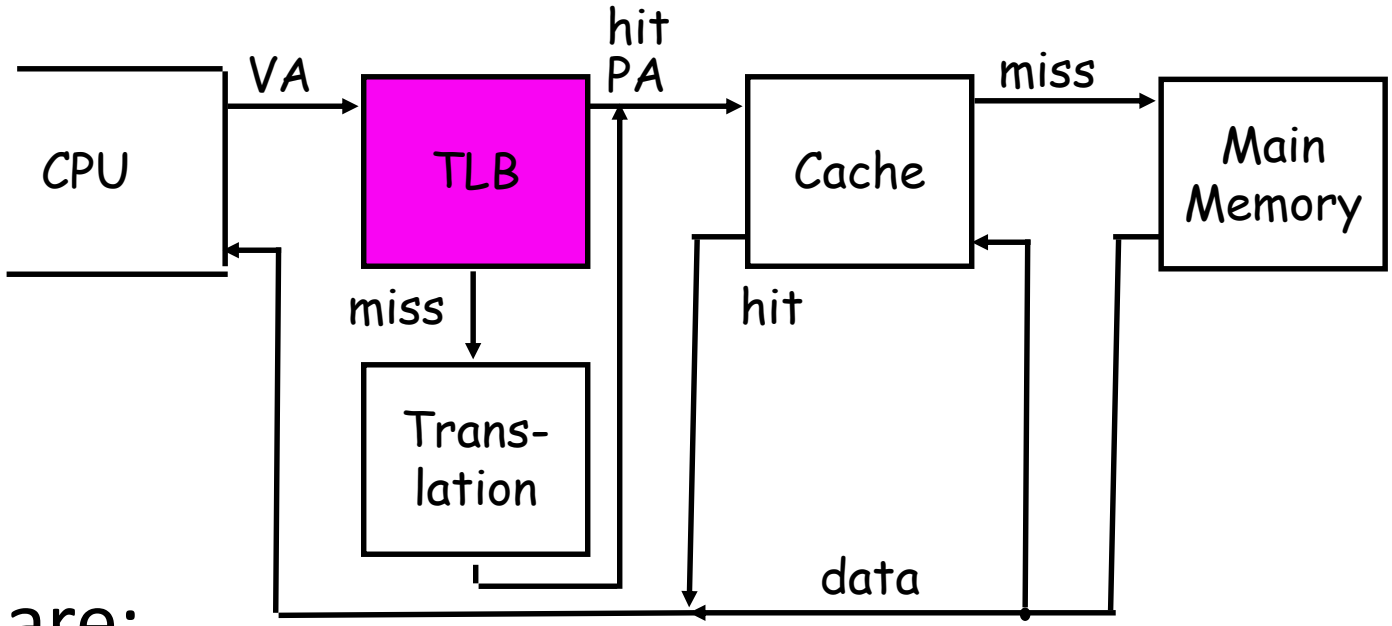
- Block Placement
  - Choice: lower miss rates and complex placement or vice versa
    - Miss penalty is huge, so choose low miss rate → place anywhere
    - Similar to fully associative cache model
- Block Identification - both use additional data structure
  - Fixed size pages - use a page table
  - 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

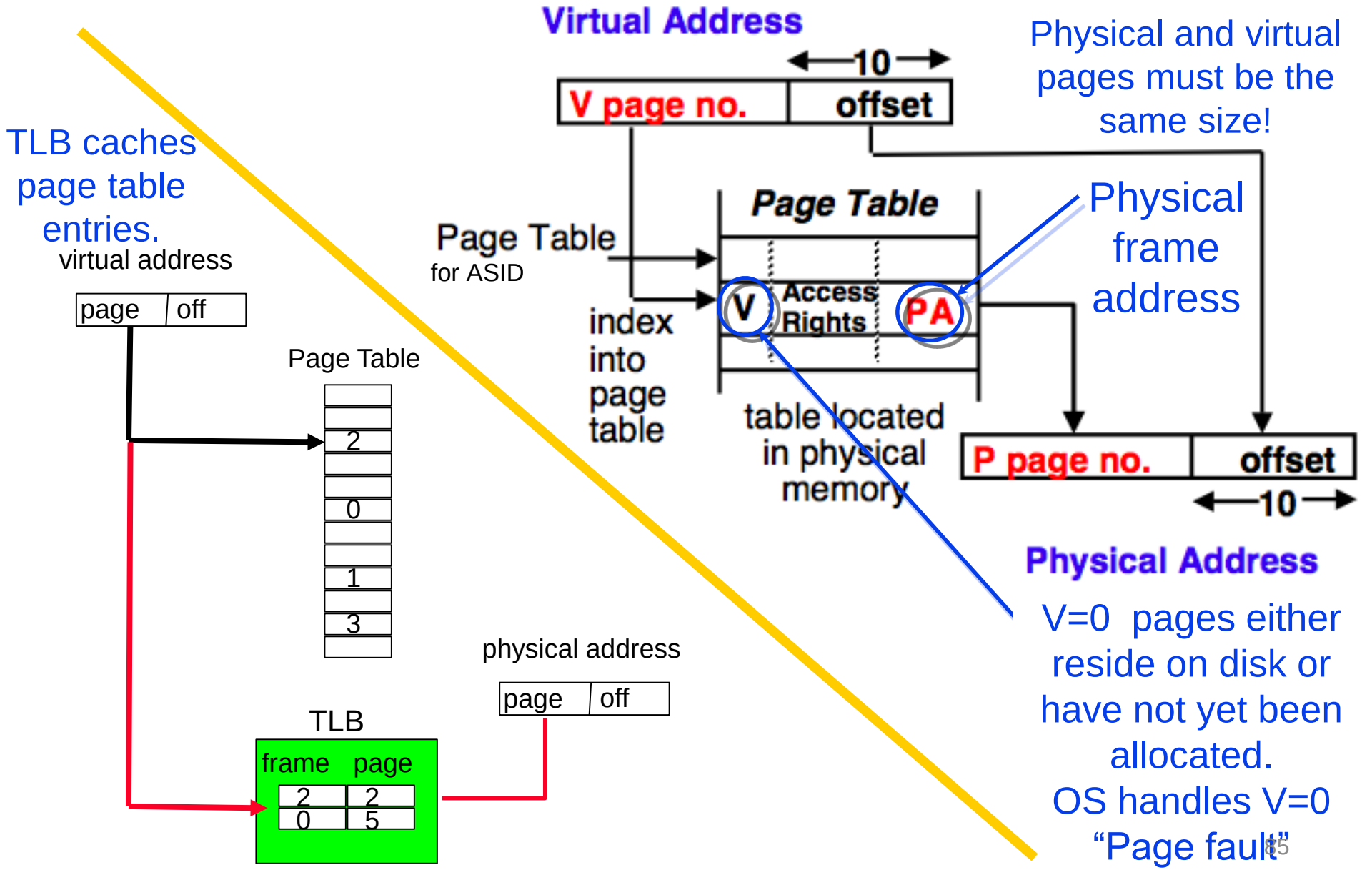
- Translation Look-Aside Buffers (TLB)
  - Cache on translations
  - Fully Associative, Set Associative, or Direct Mapped



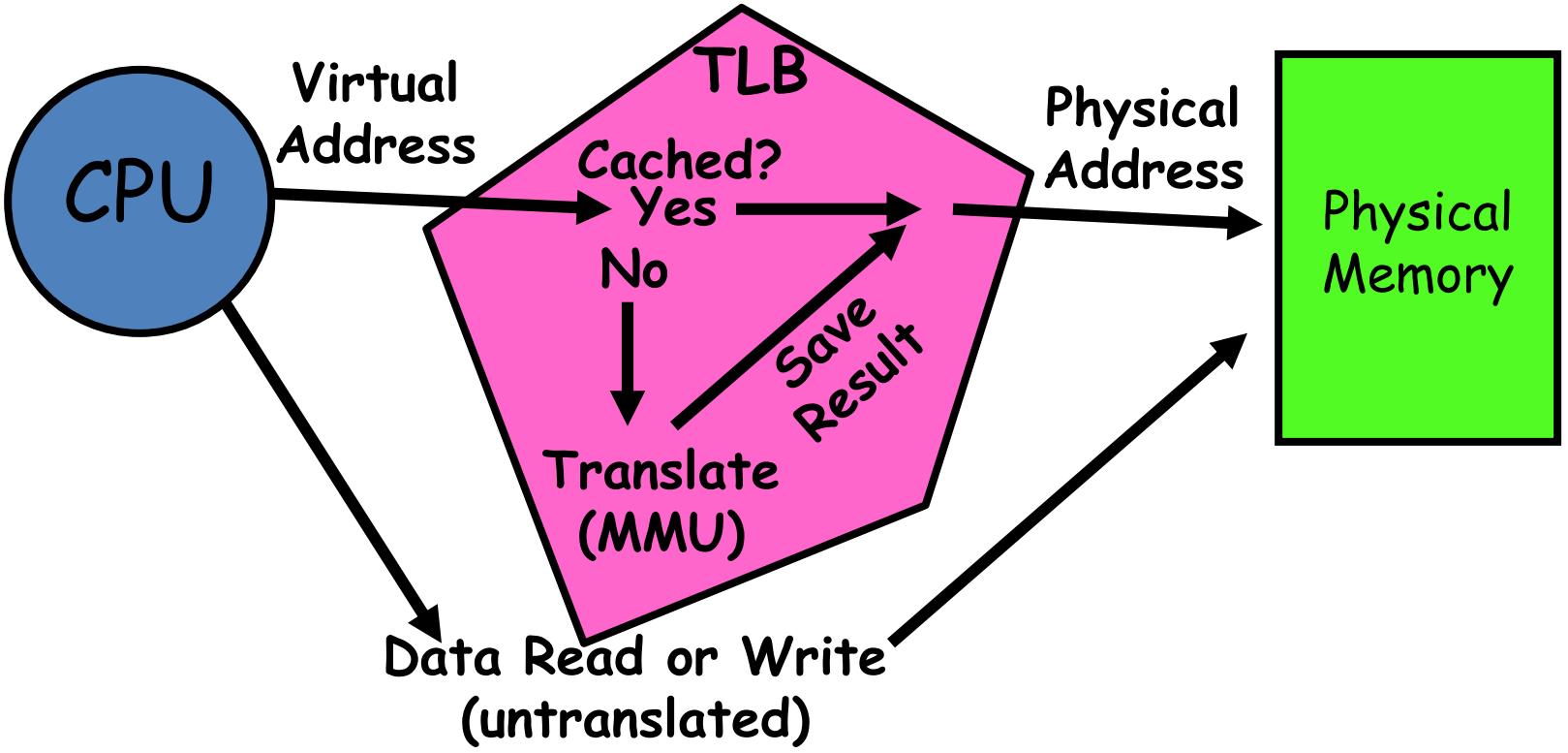
Translation with a TLB

- 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