Ch2-2
How to improve the performance of Memory hierarchy

2.2 Memory Technology and Optimizations

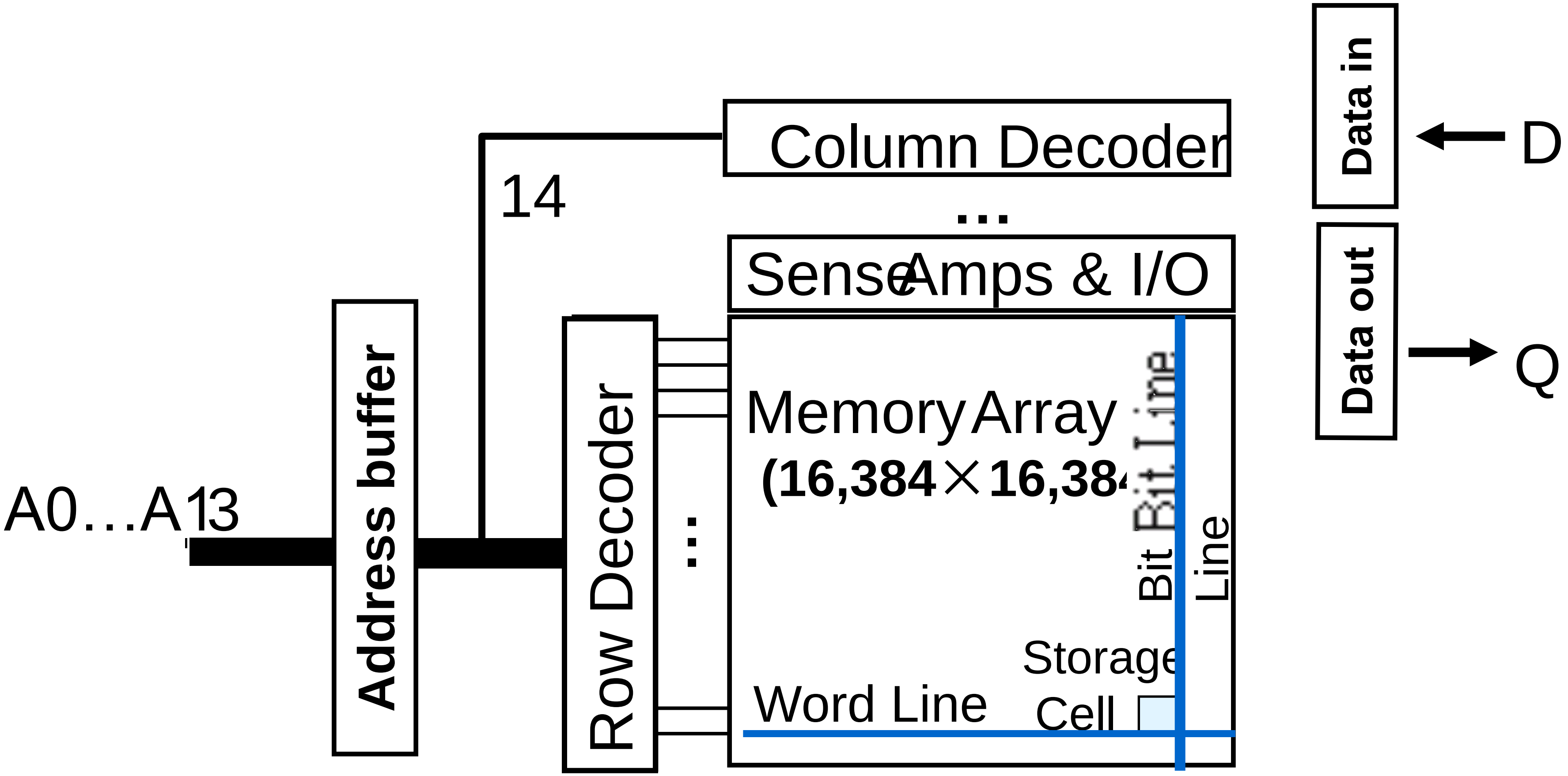
- Performance metrics
  - Latency is concern of cache
  - Bandwidth is concern of multiprocessors and I/O
  - Access time
    - Time between read request and when desired word arrives
  - Cycle time
    - Minimum time between unrelated requests to memory

- SRAM memory has low latency, use for cache
- Organize DRAM chips into many banks for high bandwidth, use for main memory

Memory Technology

- SRAM
  - Requires low power to retain bit
  - Requires 6 transistors/bit
- DRAM
  - Must be re-written after being read
  - Must also be periodically refeshed
    - Every ~ 8 ms (roughly 5% of time)
    - Each row can be refreshed simultaneously
  - One transistor/bit
  - Address lines are multiplexed:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)

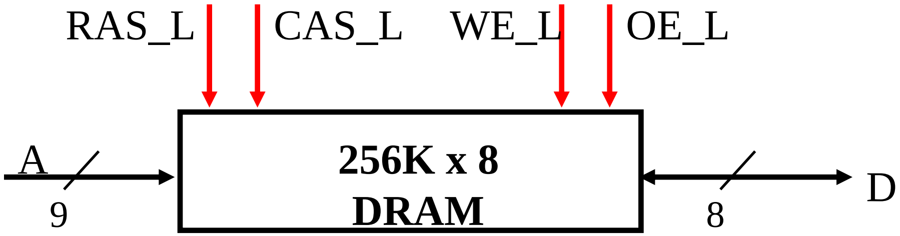
DRAM logical organization (64 Mbit) 这个不大对,改成256

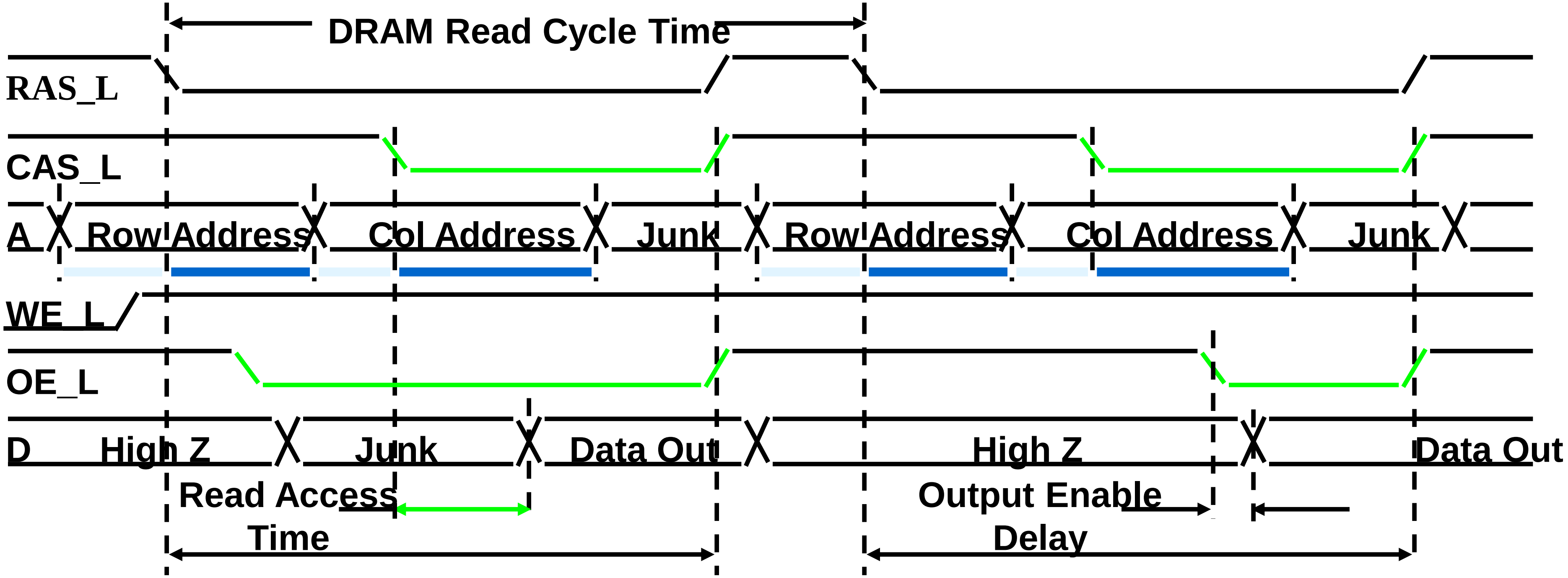


- ❑Square root of bits per RAS/CAS

DRAM Read Timing

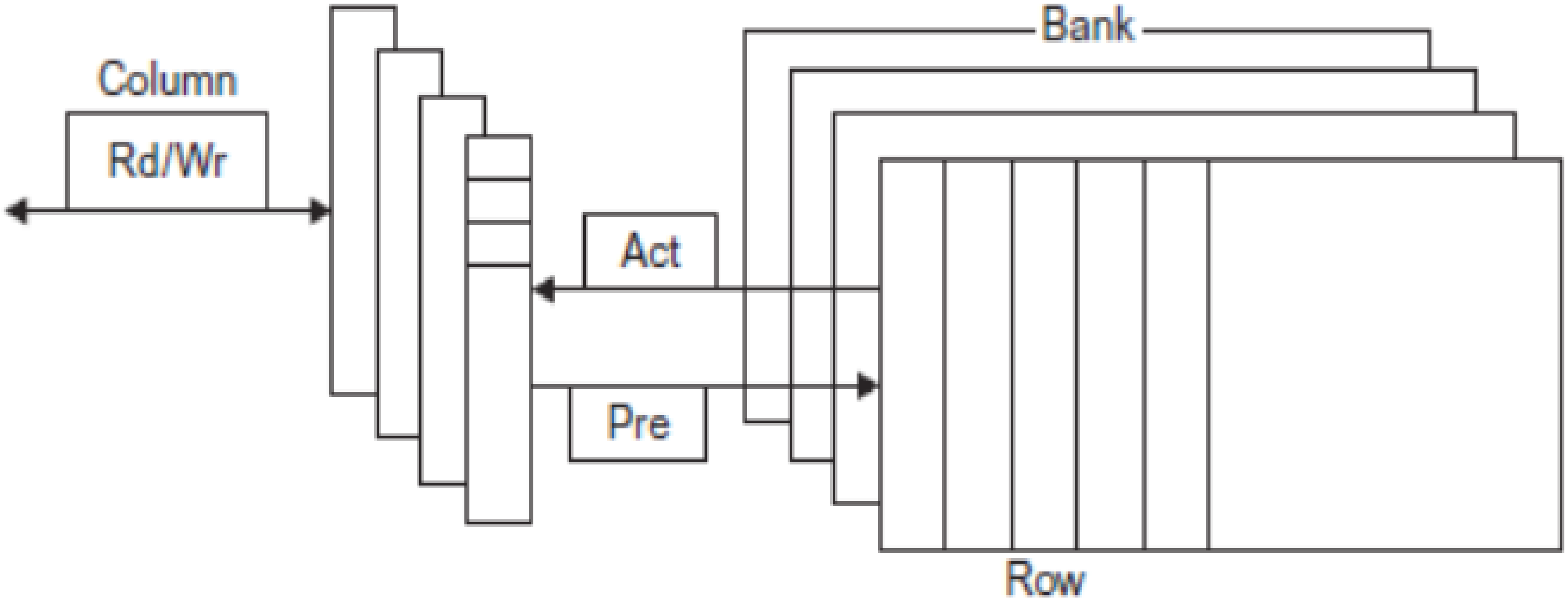
- Every DRAM access begins at:
  - The assertion of the RAS_L
  - 2 ways to read: early or late v. CAS



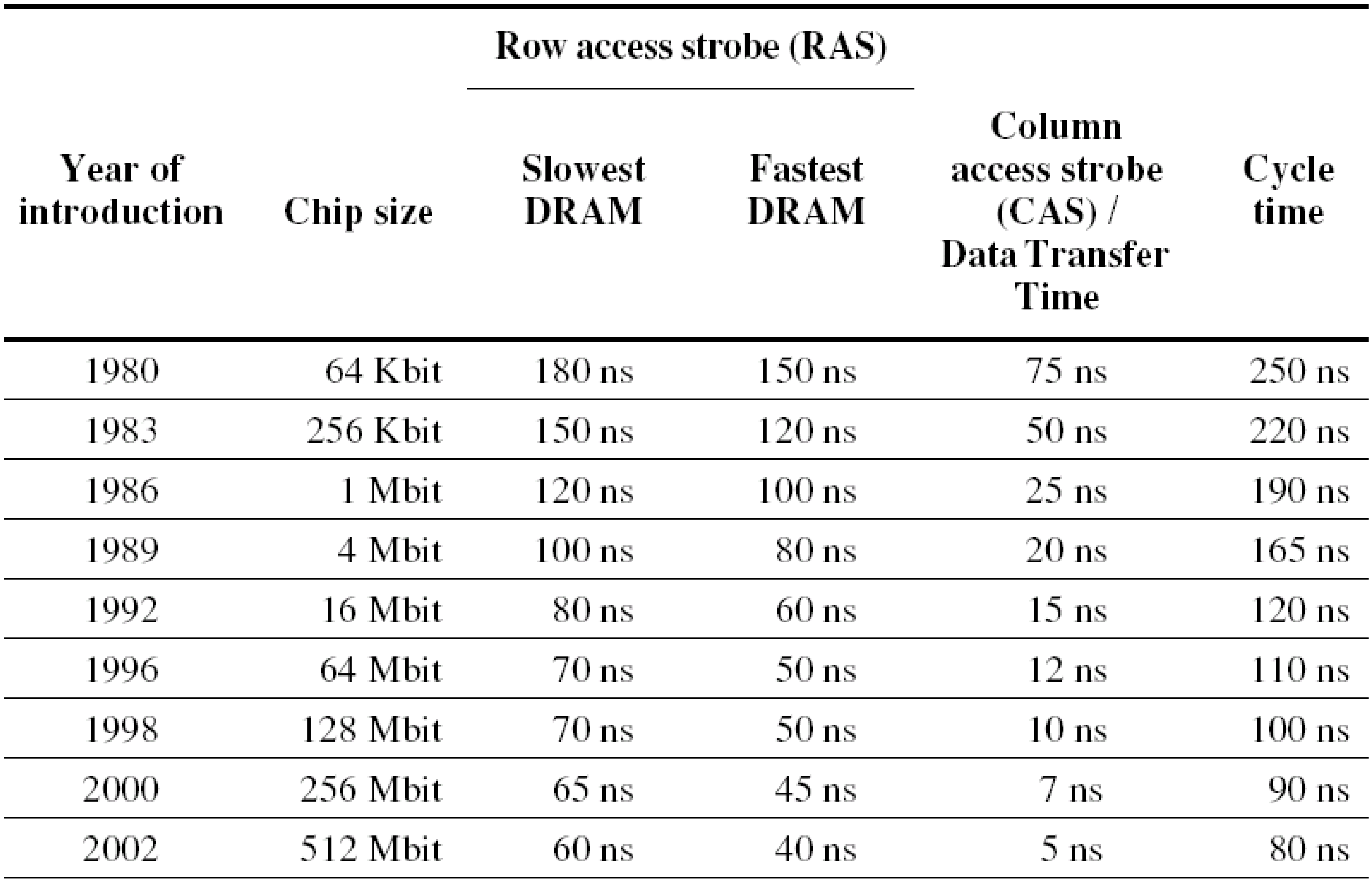


Early Read Cycle: OE_L asserted before CAS_L Late Read Cycle: OE_L asserted after CAS_L

Internal Organization of DRAM

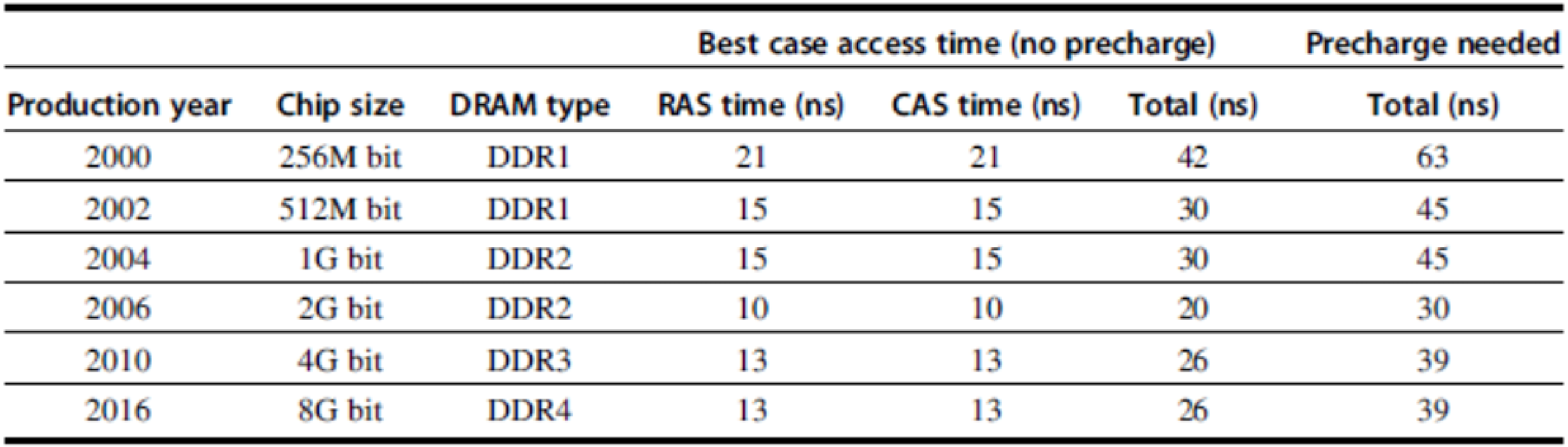


Times of fast and slow DRAMs with each generation.



		Row access strobe (RAS)			
Year of introduction	Chip size	Slowest DRAM	Fastest DRAM	Column access strobe (CAS) / Data Transfer Time	Cycle time
1980	64 Kbit	180 ns	150 ns	75 ns	250 ns
1983	256 Kbit	150 ns	120 ns	50 ns	220 ns
1986	1 Mbit	120 ns	100 ns	25 ns	190 ns
1989	4 Mbit	100 ns	80 ns	20 ns	165 ns
1992	16 Mbit	80 ns	60 ns	15 ns	120 ns
1996	64 Mbit	70 ns	50 ns	12 ns	110 ns
1998	128 Mbit	70 ns	50 ns	10 ns	100 ns
2000	256 Mbit	65 ns	45 ns	7 ns	90 ns
2002	512 Mbit	60 ns	40 ns	5 ns	80 ns

Memory Optimizations



			Best case access time (no precharge)			Precharge needed
Production year	Chip size	DRAM type	RAS time (ns)	CAS time (ns)	Total (ns)	Total (ns)
2000	256M bit	DDR1	21	21	42	63
2002	512M bit	DDR1	15	15	30	45
2004	1G bit	DDR2	15	15	30	45
2006	2G bit	DDR2	10	10	20	30
2010	4G bit	DDR3	13	13	26	39
2016	8G bit	DDR4	13	13	26	39

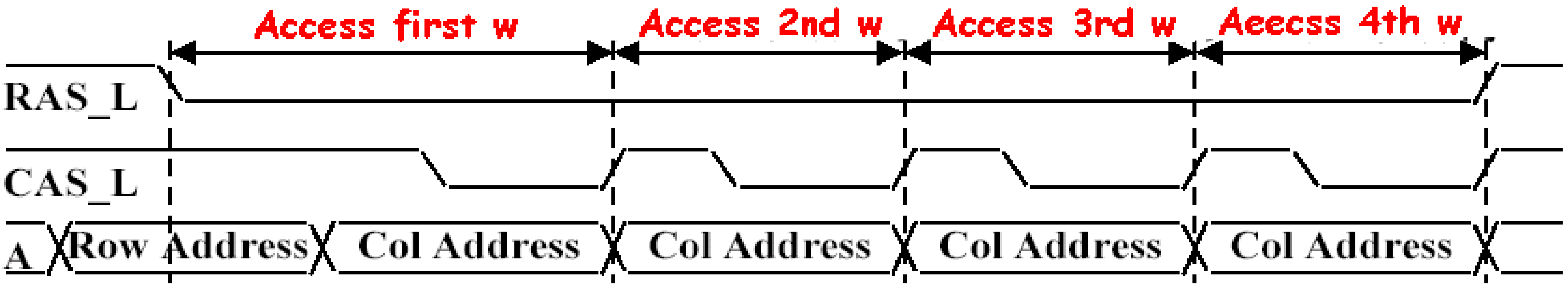
Memory Technology

- Amdahl:
  - Memory capacity should grow linearly with processor speed
  - Unfortunately, memory capacity and speed has not kept pace with processors

- Some optimizations:
  - Multiple accesses to same row
  - Synchronous DRAM 异步访问握手过程很耗时
    - Added clock to DRAM interface
    - Burst mode with critical word first
  - Wider interfaces 一次拿更多word
  - Double data rate (DDR) 工作频率提高
  - Multiple banks on each DRAM device

1st Improving DRAM Performance
Fast Page Mode DRAM (FPM)

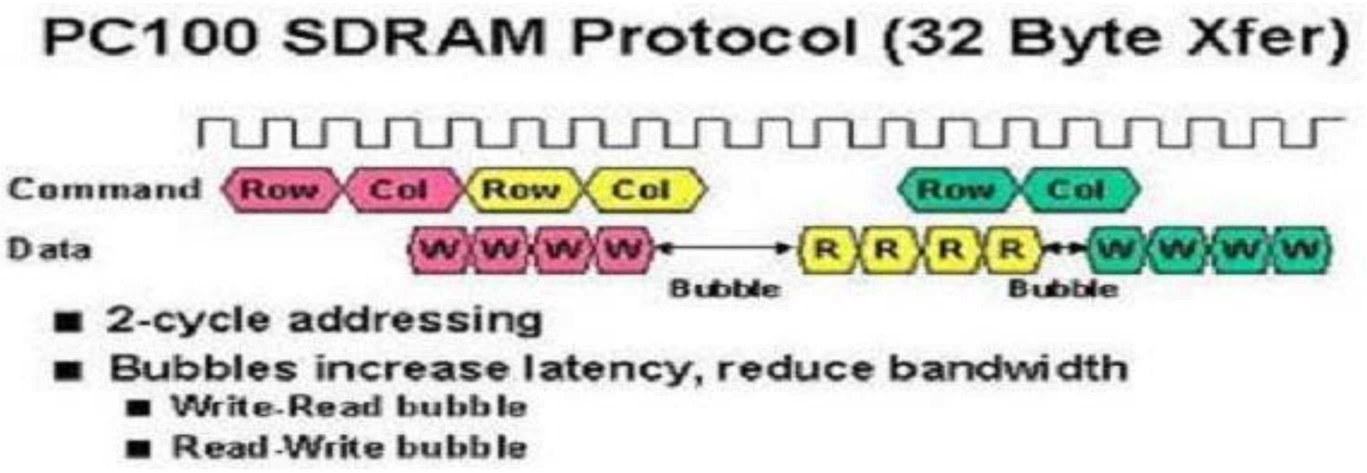
- ❑Timing signals that allow repeated accesses to the row buffer (page) without another row access time
- ❑ Such a buffer comes naturally, as each array will buffer 1024 to 2048 bits for each access.
- ❑Page: All bits on the same ROW (Spatial Locality)
  - ➢Don't need to wait for wordline to recharge
  - ➢Toggle CAS with new column address



2nd Improving DRAM Performance

Synchronous DRAM

- ❑conventional DRAMs have an asynchronous interface to the memory controller, and hence every transfer involves overhead to synchronize with the controller.
- ❑The solution was to **add a clock signal** to the DRAM interface, so that the repeated transfers would not bear that overhead.
  - ➢ Data output is in bursts w/ each element clocked



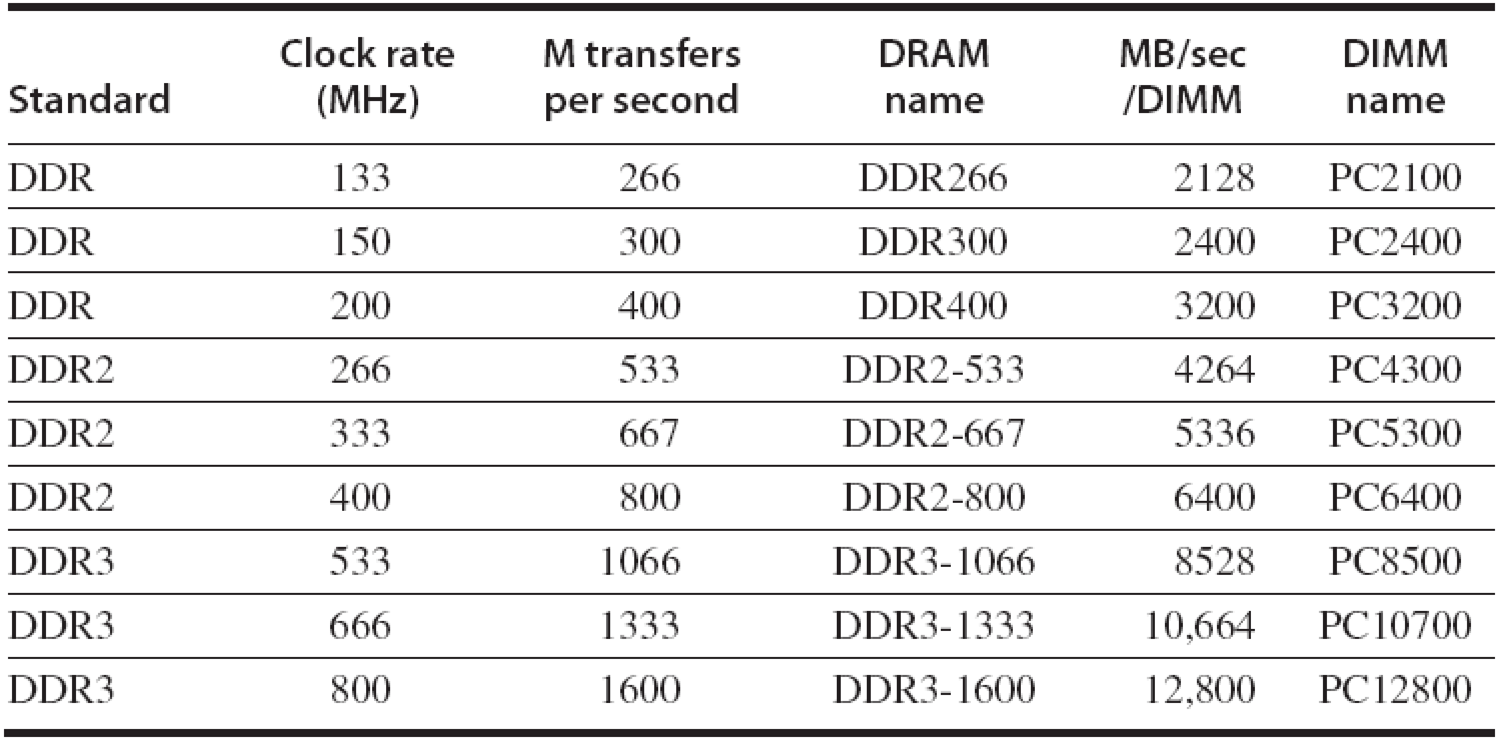
PC100 SDRAM Protocol (32 Byte Xfer)

- 2-cycle addressing
- Bubbles increase latency, reduce bandwidth
  - Write-Read bubble
  - Read-Write bubble

3rd Improving DRAM Performance

DDR--Double data rate

- **On both the** rising edge and falling edge **of the DRAM clock signal, DRAM innovation to increase bandwidth is to transfer data,**
  - **thereby doubling the peak data rate.**



	Standard	Clock rate (MHz)	M transfers per second	DRAM name	MB/sec /DIMM	DIMM name
2.5V	DDR	133	266	DDR266	2128	PC2100
	DDR	150	300	DDR300	2400	PC2400
	DDR	200	400	DDR400	3200	PC3200
1.8v	DDR2	266	533	DDR2-533	4264	PC4300
	DDR2	333	667	DDR2-667	5336	PC5300
	DDR2	400	800	DDR2-800	6400	PC6400
1.5v	DDR3	533	1066	DDR3-1066	8528	PC8500
	DDR3	666	1333	DDR3-1333	10,664	PC10700
	DDR3	800	1600	DDR3-1600	12,800	PC12800

DDR--Double data rate

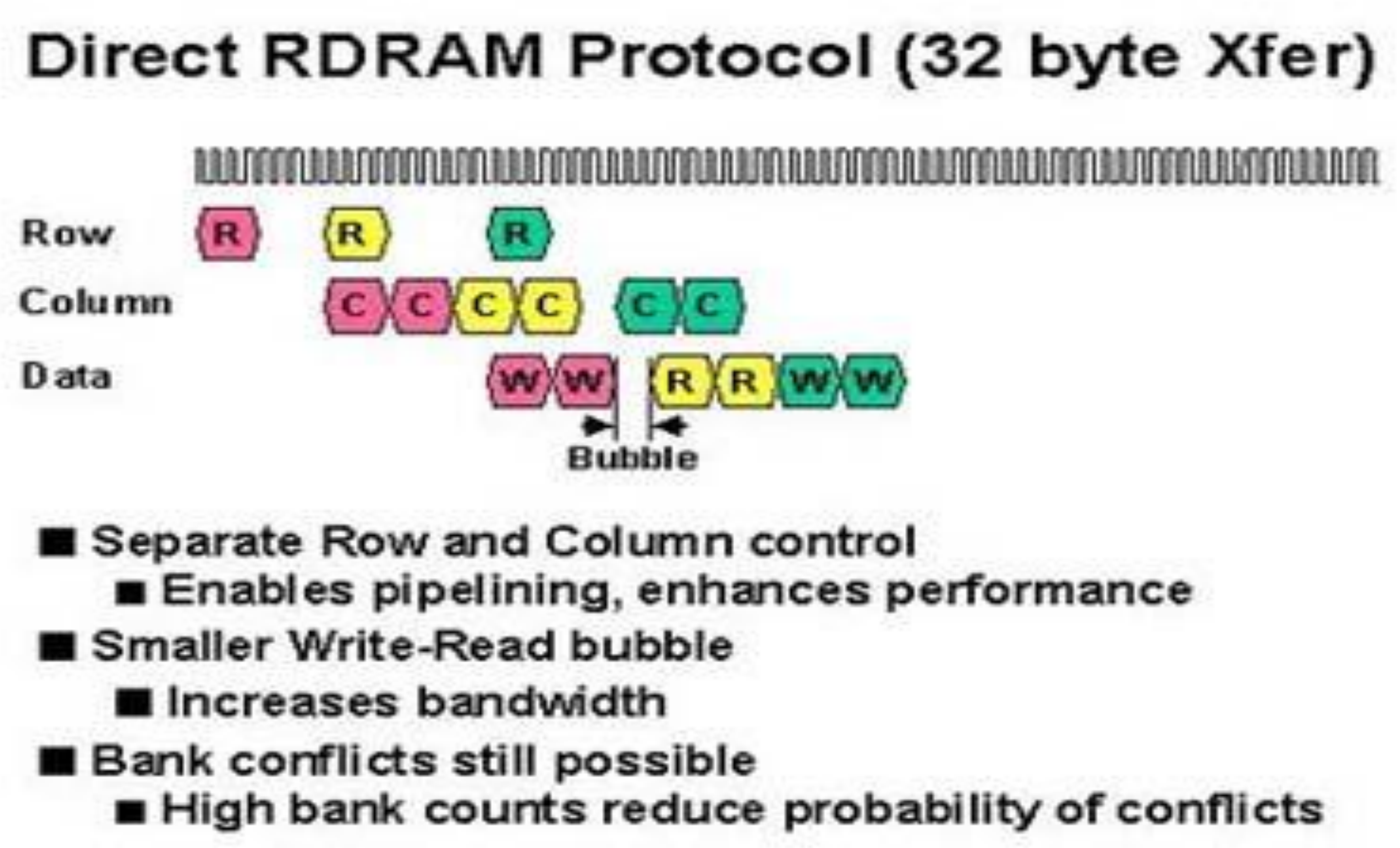
- DDR:
  - DDR2
    - Lower power (2.5 V -> 1.8 V)
    - Higher clock rates (266 MHz, 333 MHz, 400 MHz)
  - DDR3
    - 1.5 V
    - 800 MHz
  - DDR4
    - 1-1.2 V
    - 1333 MHz
- GDDR5 is graphics memory based on DDR3

4rd Improving DRAM Performance

New DRAM Interface: RAMBUS (RDRAM)

- ❑ a type of synchronous dynamic RAM, designed by the Rambus Corporation.
- ❑ Each chip has interleaved memory and a high speed interface.
- ❑ Protocol based RAM w/ narrow (16-bit) bus
  - ➢ High clock rate (400 Mhz), but long latency
  - ➢ Pipelined operation
- ❑ Multiple arrays w/ data transferred on both edges of clock
- ❑ The first generation RAMBUS interface dropped RAS/CAS, replacing it with a bus that allows other accesses over the bus between the sending of the address and return of the data. It is typically called *RDRAM*.
- ❑ The second generation RAMBUS interface include a separate row- and column-command buses instead of the conventional multiplexing; and a much more sophisticated controller on chip. Because of the separation of data, row, and column buses, three transactions can be performed simultaneously. called *Direct RDRAM* or *DRDRAM*

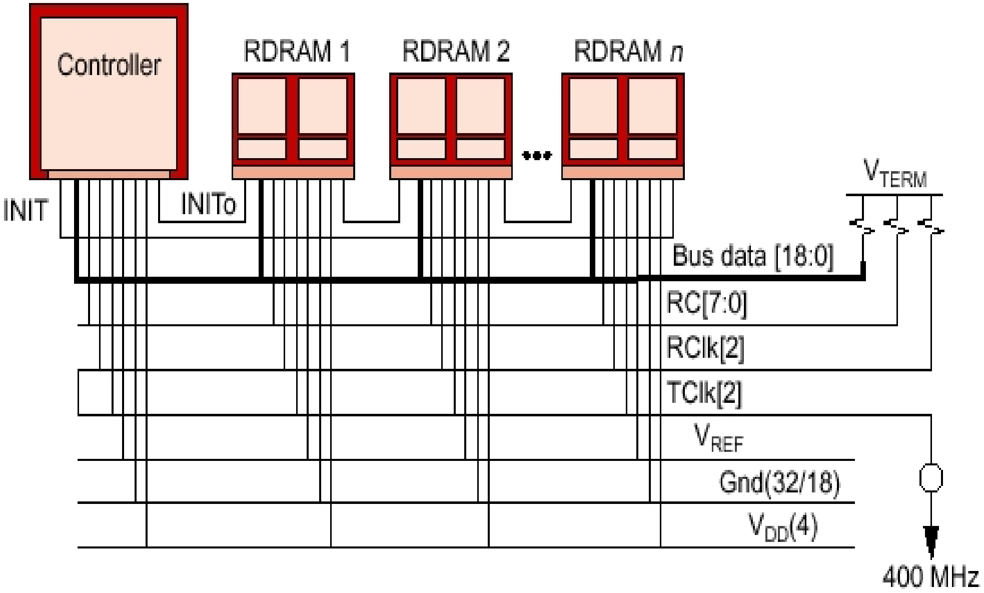
RDRAM Timing



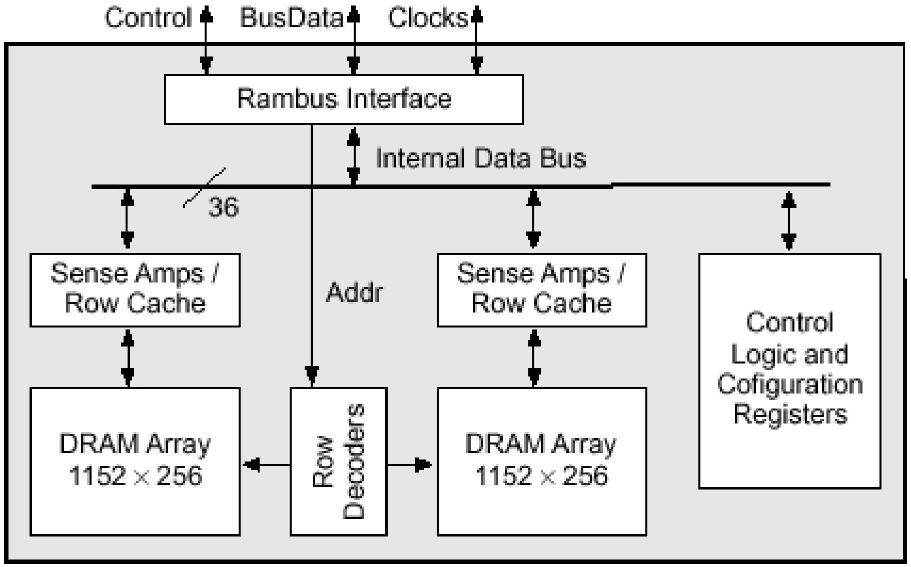
Direct RDRAM Protocol (32 byte Xfer)

- Separate Row and Column control
  - Enables pipelining, enhances performance
- Smaller Write-Read bubble
  - Increases bandwidth
- Bank conflicts still possible
  - High bank counts reduce probability of conflicts

RAMBUS (RDRAM)



RDRAM Memory System



RAMBUS Bank

Comparing RAMBUS and DDR SDRAM

- **Since the most computers use memory in DIMM packages, which are typically at least 64-bits wide, the DIMM memory bandwidth is** closer **to what RAMBUS provides than you might expect when just comparing DRAM chips.**
- Caution **that performance of cache are based in part on** latency **to the** first byte **and in part on** the bandwidth **to deliver the** rest of the bytes **in the block.**
  - **Although these innovations help with the latter case, none help with latency.**
  - **Amdahl's Law reminds us of the limits of accelerating one piece of the problem while ignoring another part.**

Memory Optimizations

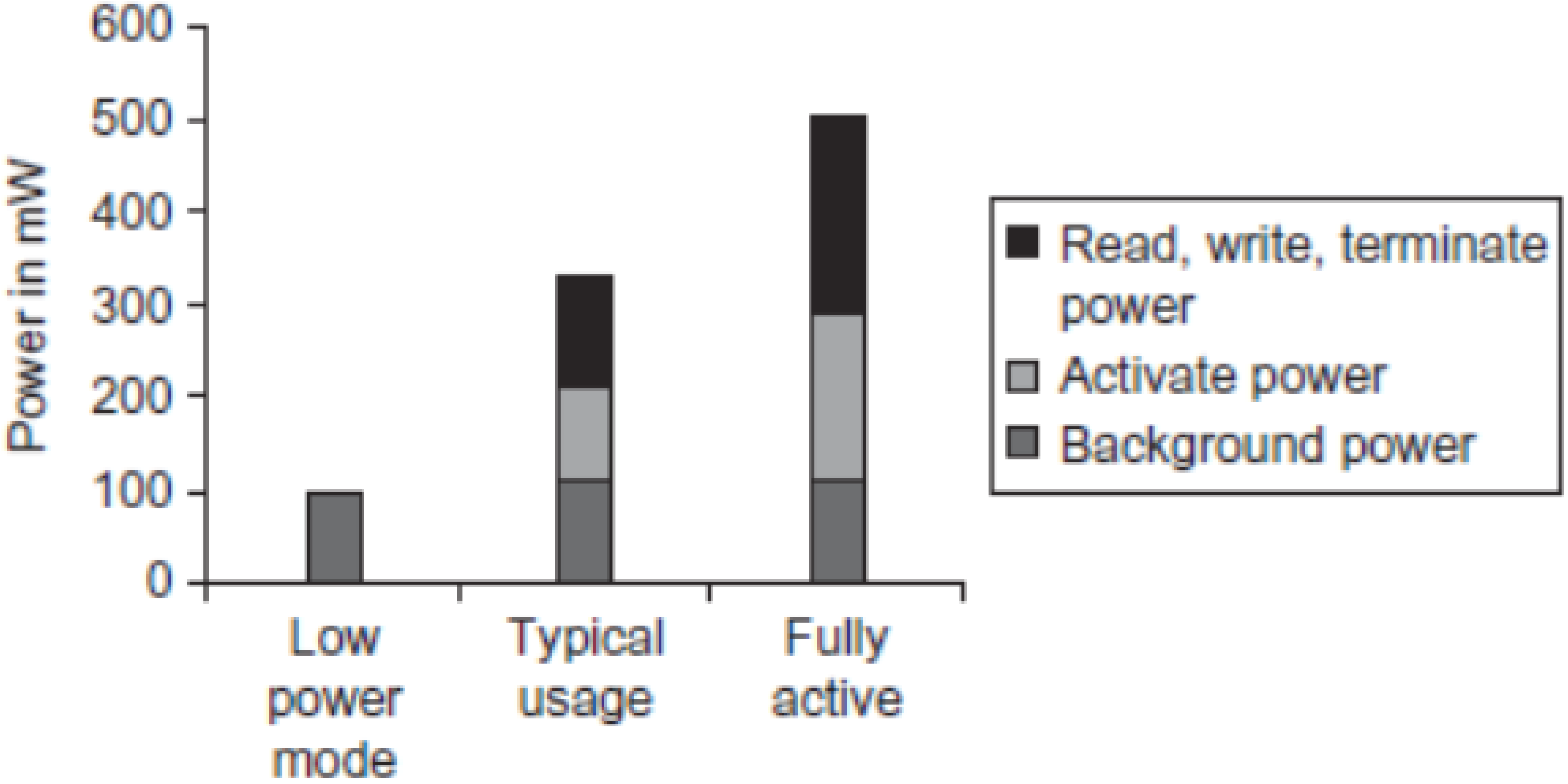
- Reducing power in SDRAMs:
  - Lower voltage
  - Low power mode (ignores clock, continues to refresh)

- 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 socketted DIMM modules

Summery

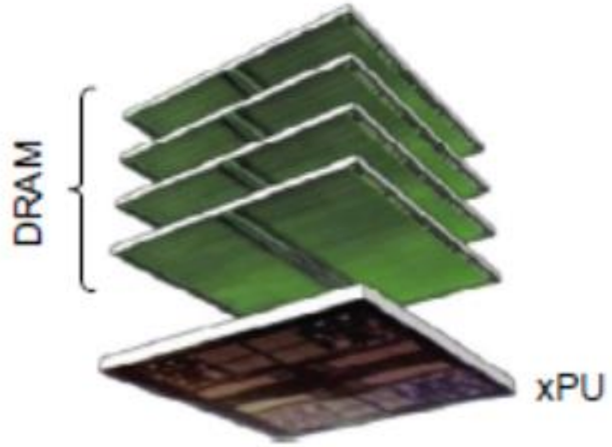
- Memory organization
  - Wider memory
  - Simple interleaved memory
  - Independent memory banks
  - Avoiding Memory Bank Conflicts
- Memory chip
  - Fast Page Mode DRAM
  - Synchronize DRAM
  - Double Date Rate
  - RDRAM

Memory Power Consumption

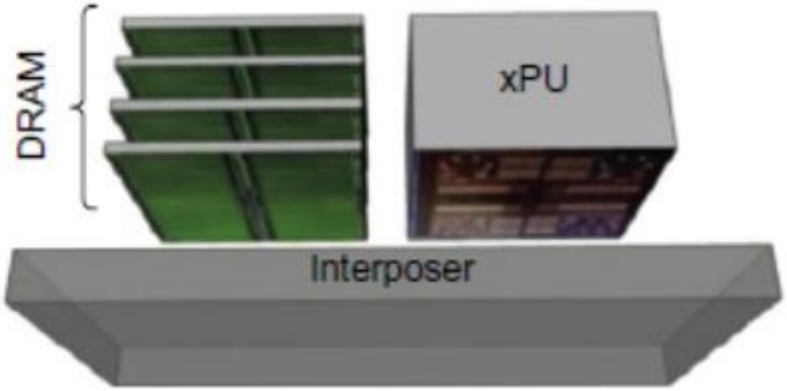


Stacked/Embedded DRAMs

- Stacked DRAMs in same package as processor
  - High Bandwidth Memory (HBM)



Vertical stacking (3D)



Interposer stacking (2.5D)

Flash Memory

- Type of EEPROM
- Types: NAND (denser) and NOR (faster)
- NAND Flash:
  - Reads are sequential, reads entire page (.5 to 4 KiB)
  - 25 us for first byte, 40 MiB/s for subsequent bytes
  - SDRAM: 40 ns for first byte, 4.8 GB/s for subsequent bytes
  - 2 KiB transfer: 75 uS vs 500 ns for SDRAM, 150X slower
  - 300 to 500X faster than magnetic disk

NAND Flash Memory

- Must be erased (in blocks) before being overwritten
- Nonvolatile, can use as little as zero power
- Limited number of write cycles (~100,000)
- $2/GiB, compared to $20-40/GiB for SDRAM and $0.09 GiB for magnetic disk

- Phase-Change（相变）/Memrister Memory
  - Possibly 10X improvement in write performance and 2X improvement in read performance

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 spare rows to replace defective rows

- Chipkill: a RAID-like error recovery technique

How to Improve Cache Performance?

$$\text{AMAT} = \text{HitTime} + \text{MissRate} \times \text{MissPenalty}$$

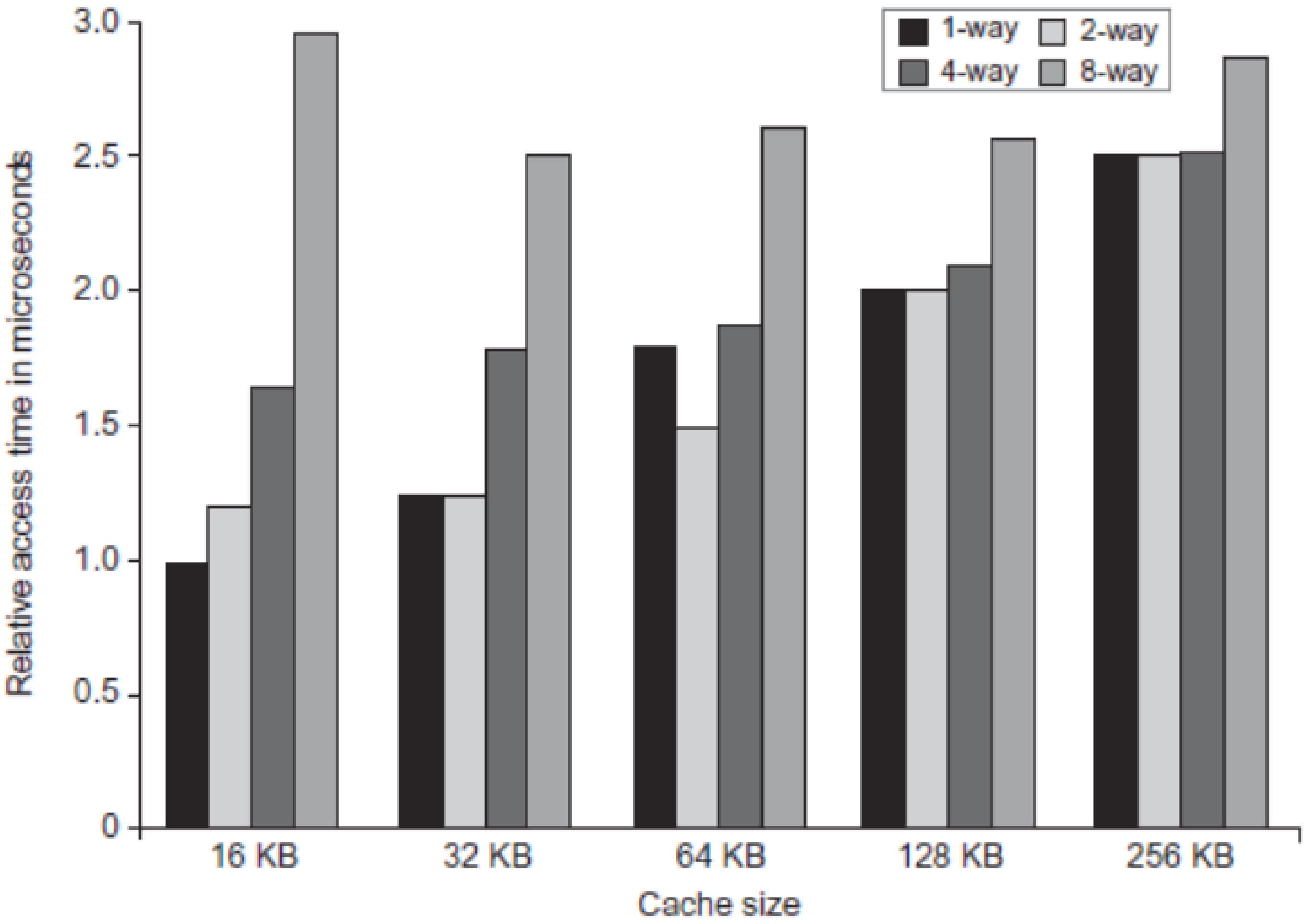
- ❑Reduce hit time(4)
  - ➢ Small and simple first-level caches, Way prediction
  - ➢ avoiding address translation, Trace cache
- ❑Increase bandwidth(3)
  - ➢ Pipelined caches, multibanked caches, non-blocking caches
- ❑Reduce miss penalty(5)
  - ➢ multilevel caches, read miss prior to writes,
  - ➢ Critical word first, merging write buffers, and victim caches
- ❑Reduce miss rate(4)
  - ➢ larger block size, large cache size, higher associativity
  - ➢ Compiler optimizations
- ❑Reduce miss penalty or miss rate via parallelization (1)
  - ➢ Hardware or compiler prefetching

1st Hit Time Reduction Technique: Small and Simple Caches

Using small and Direct-mapped cache

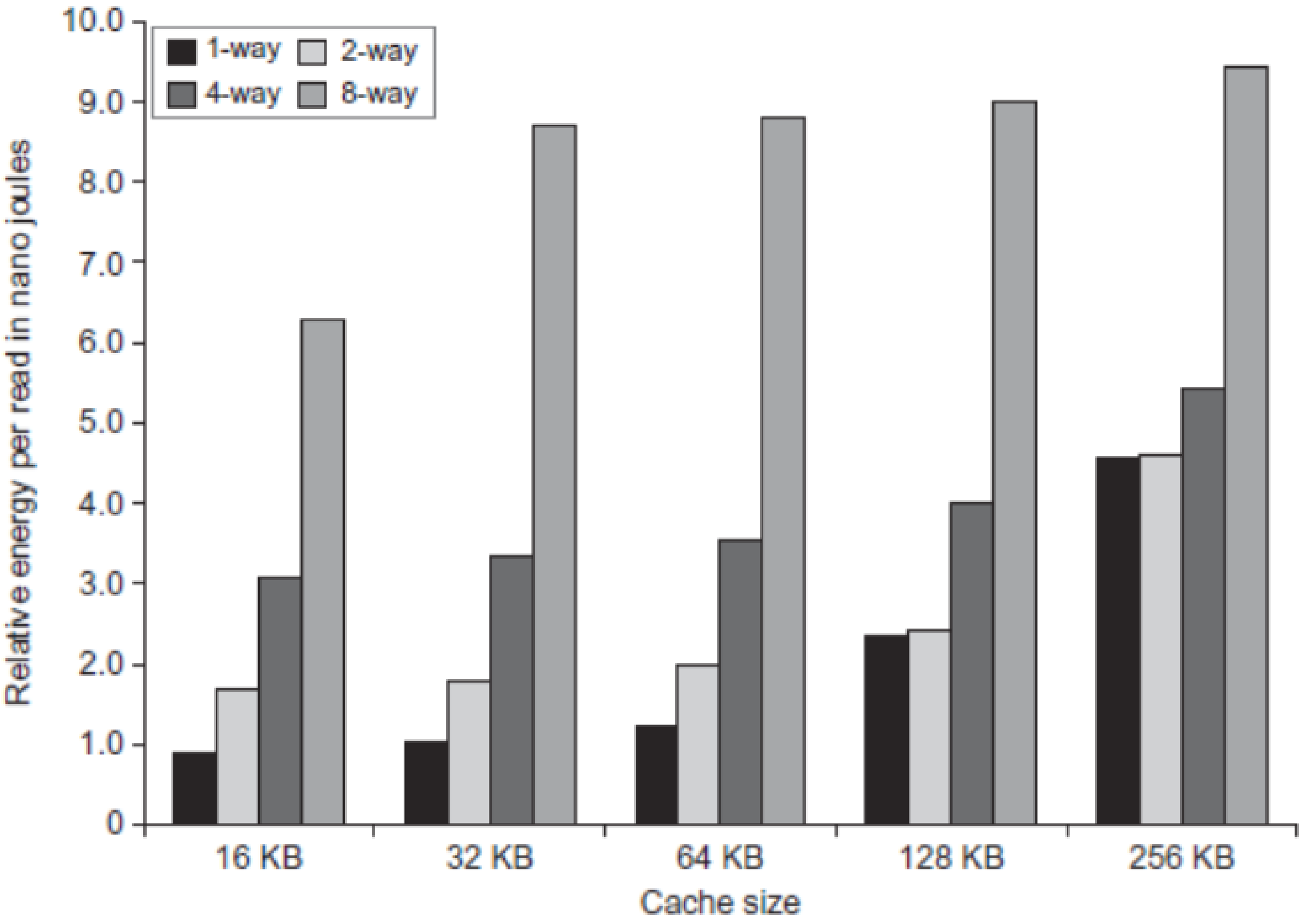
- ❑The less hardware that is necessary to implement a cache, the shorter the critical path through the hardware.
- ❑Direct-mapped is faster than set associative for both reads and writes.
- ❑Fitting the cache on the chip with the CPU is also very important for fast access times.

L1 Size and Associativity



Access time vs. size and associativity

L1 Size and Associativity



Energy per read vs. size and associativity

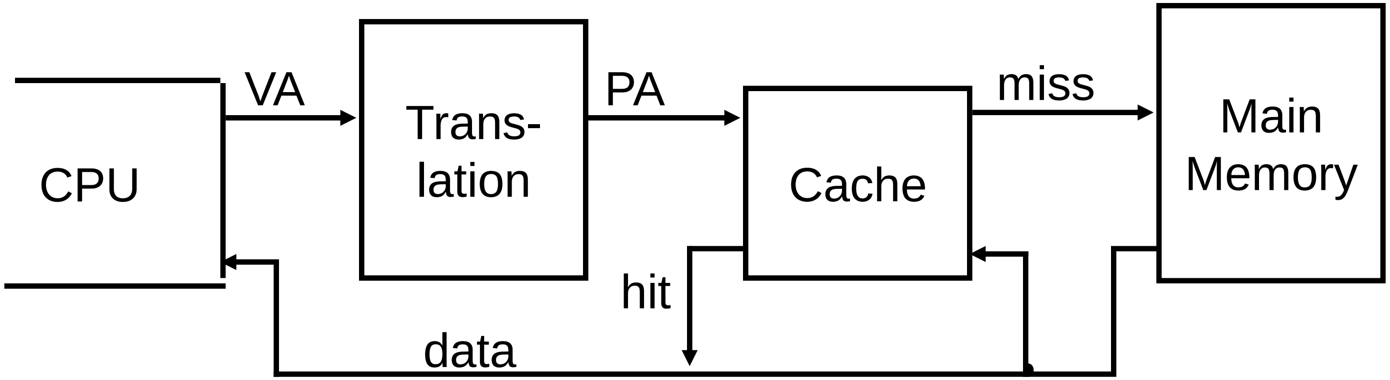
2nd Hit Time Reduction Technique: Way Prediction

- 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

2nd Hit Time Reduction Technique: Way Prediction

- Way Prediction (Pentium 4)
  - Extra bits are kept in the cache to predict the way,or block within set of the *next* cache access.
  - If the predictor is correct, the instruction cache latency is 1 clock clock cycle.
  - If not,it tries the other block, changes the way predictor, and has a latency of 1 extra clock cycles.
  - Simulation using SPEC95 suggested set prediction accuracy is excess of 85%, so way prediction saves pipeline stage in more than 85% of the instruction fetches.

3rd Hit Time Reduction Technique: Avoiding Address Translation during Indexing of the Cache



- Page table is a large data structure in memory
- TWO memory accesses for every load, store, or instruction fetch!!!
- Virtually addressed cache?
  - synonym problem
- Cache the address translations?

TLBs

A way to speed up translation is to use a special cache of recently used page table entries -- this has many names, but the most frequently used is *Translation Lookaside Buffer* or *TLB*

Virtual Address	Physical Address	Dirty	Ref	Valid	Access

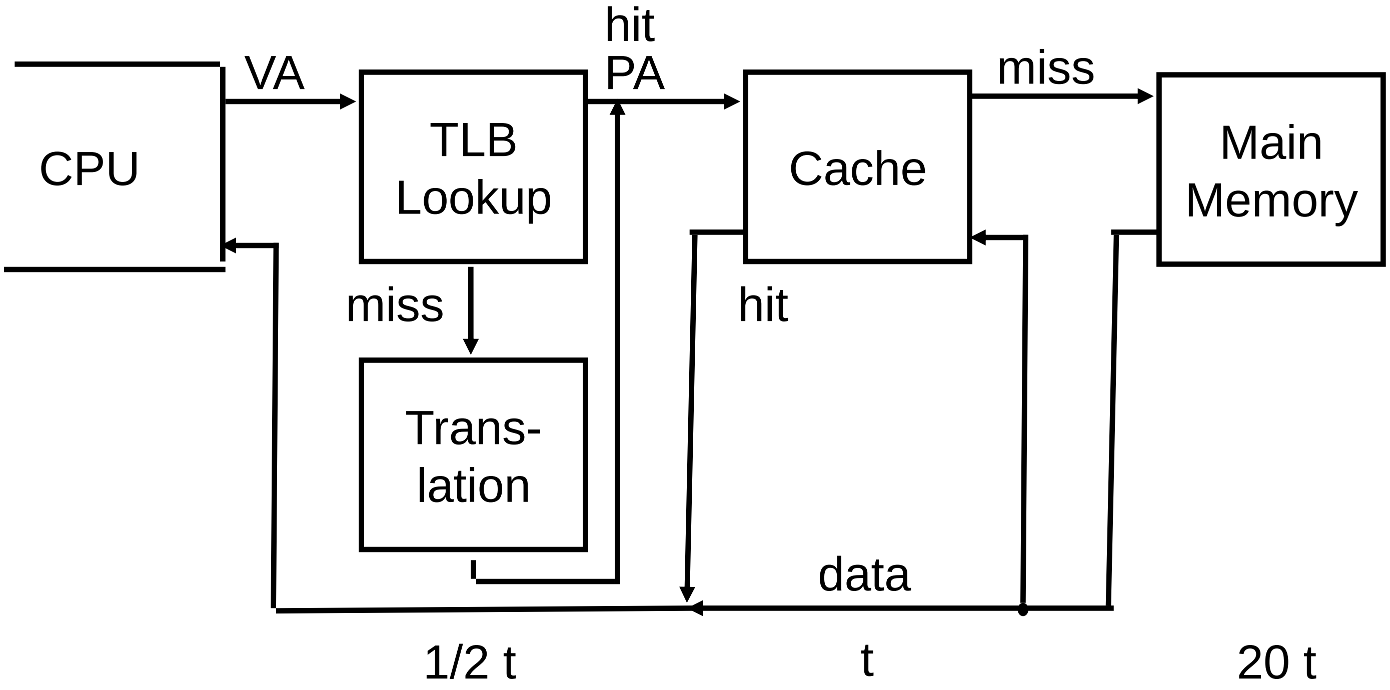
Really just a cache on the page table mappings

TLB access time comparable to cache access time
(much less than main memory access time)

Translation Look-Aside Buffers

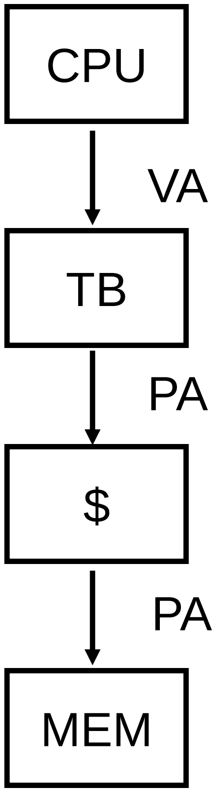
Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped

TLBs are usually small, typically not more than 128 - 256 entries even on high end machines. This permits fully associative lookup on these machines. Most mid-range machines use small n-way set associative organizations.

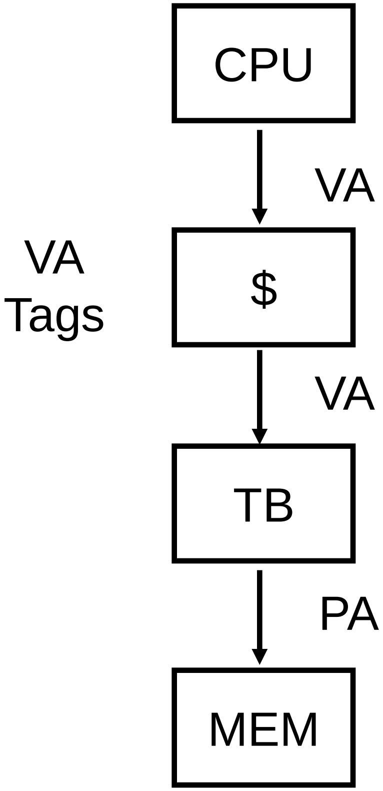


Translation with a TLB

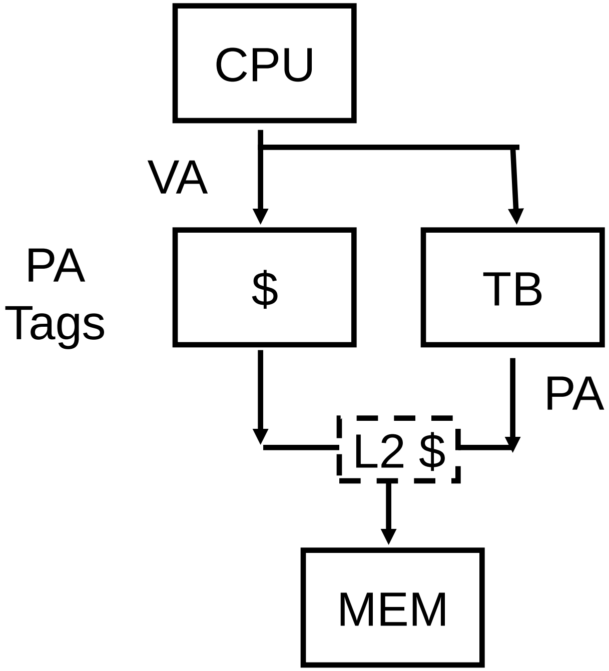
Fast hits by Avoiding Address Translation



Conventional
Organization



Virtually Addressed Cache
Translate only on miss
Synonym Problem



Virtual indexed, Physically tagged
Overlap $ access
with VA translation:
requires $ index to
remain invariant
across translation

Virtual Addressed Cache

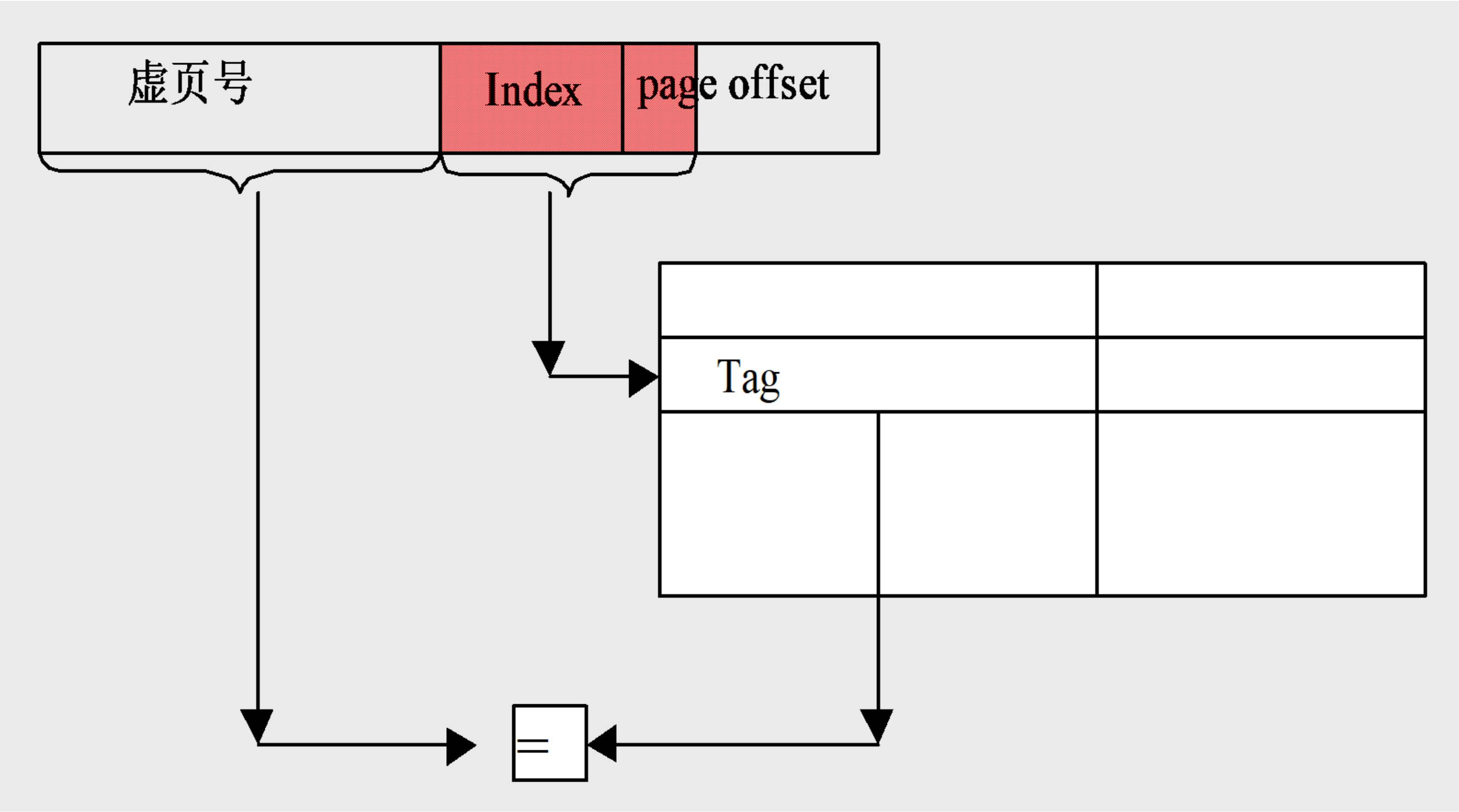
Any Problems ?

- ❑ Send virtual address to cache? Called *Virtu* *Cache* or

  just *Virtual Cache (*vs. *Physical Cache)*

- ❑ Every time process is switched logically must flush the cache; otherwise get false hits
  - ➢ Cost is time to flush + “compulsory” misses from empty cache
  - ➢ Add *process identifier tag* that identifies process as well as address within process: can’t get a hit if wrong process

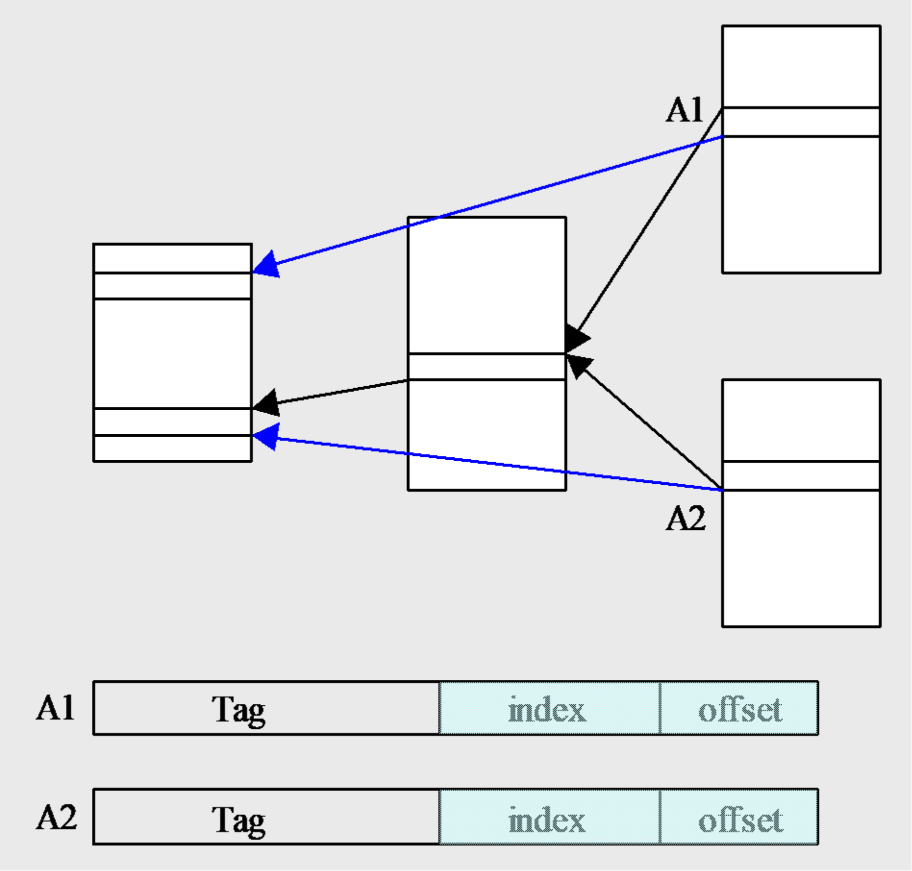
Virtual cache



Dealing with aliases

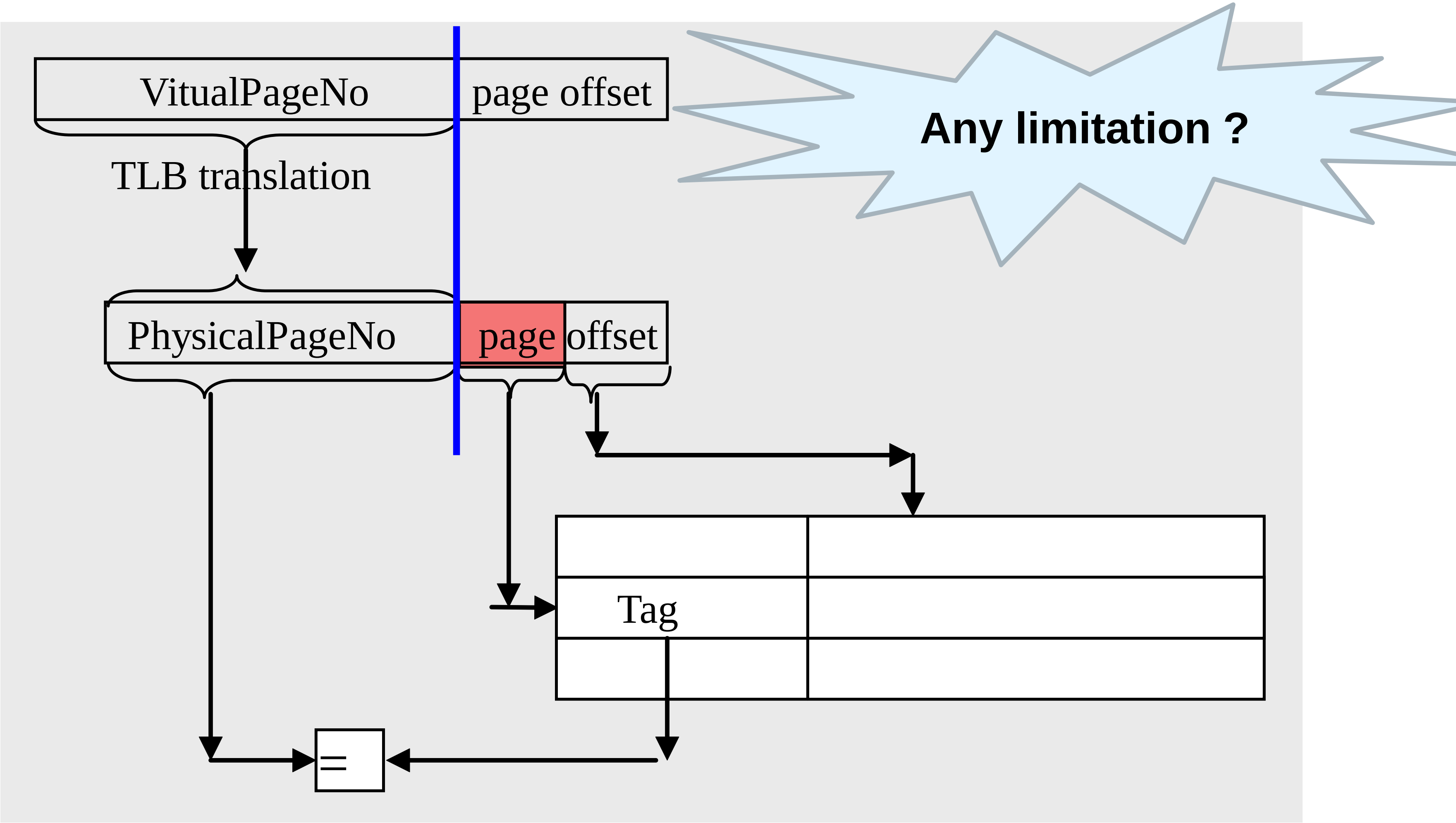
- Dealing with *aliases* (*synonyms*); Two different virtual addresses map to same physical address
- NO aliasing!   What are the implications?
- HW antialiasing: guarantees every cache block has unique address
  - verify on miss (rather than on every hit)
  - cache set size   <= page size ?
  - what if it gets larger?
- How can SW simplify the problem? (called *page coloring*)
  - I/O must interact with cache, so need virtual address

Aliases problem with Virtual cache



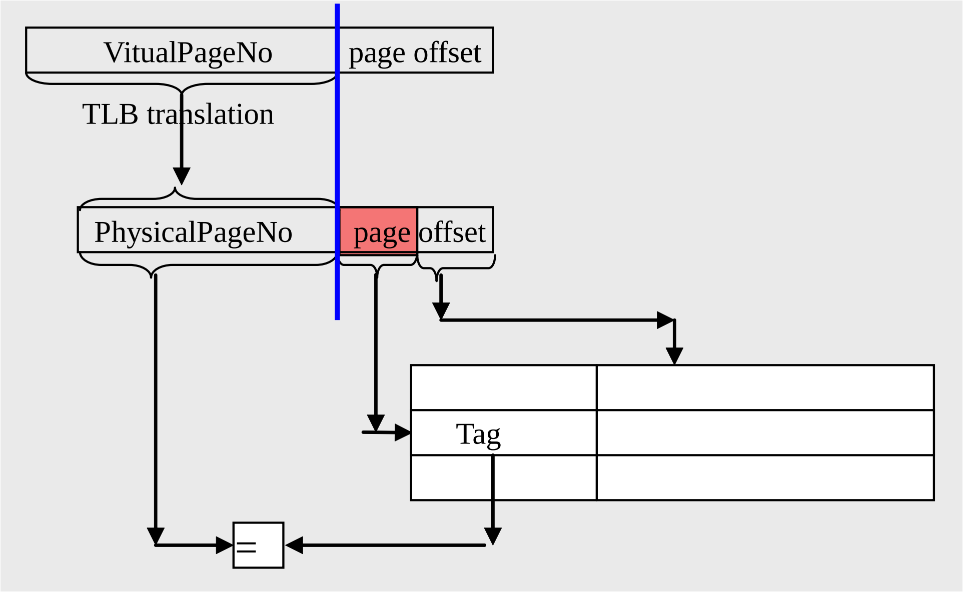
If the index and offset bits of two aliases are forced to be
the same, than the aliases address will map to the same block in cache.

Overlap address translation and cache access
(Virtual indexed, physically tagged)



Any limitation ?

What’s the limitation?



IF it’s direct map cache, then

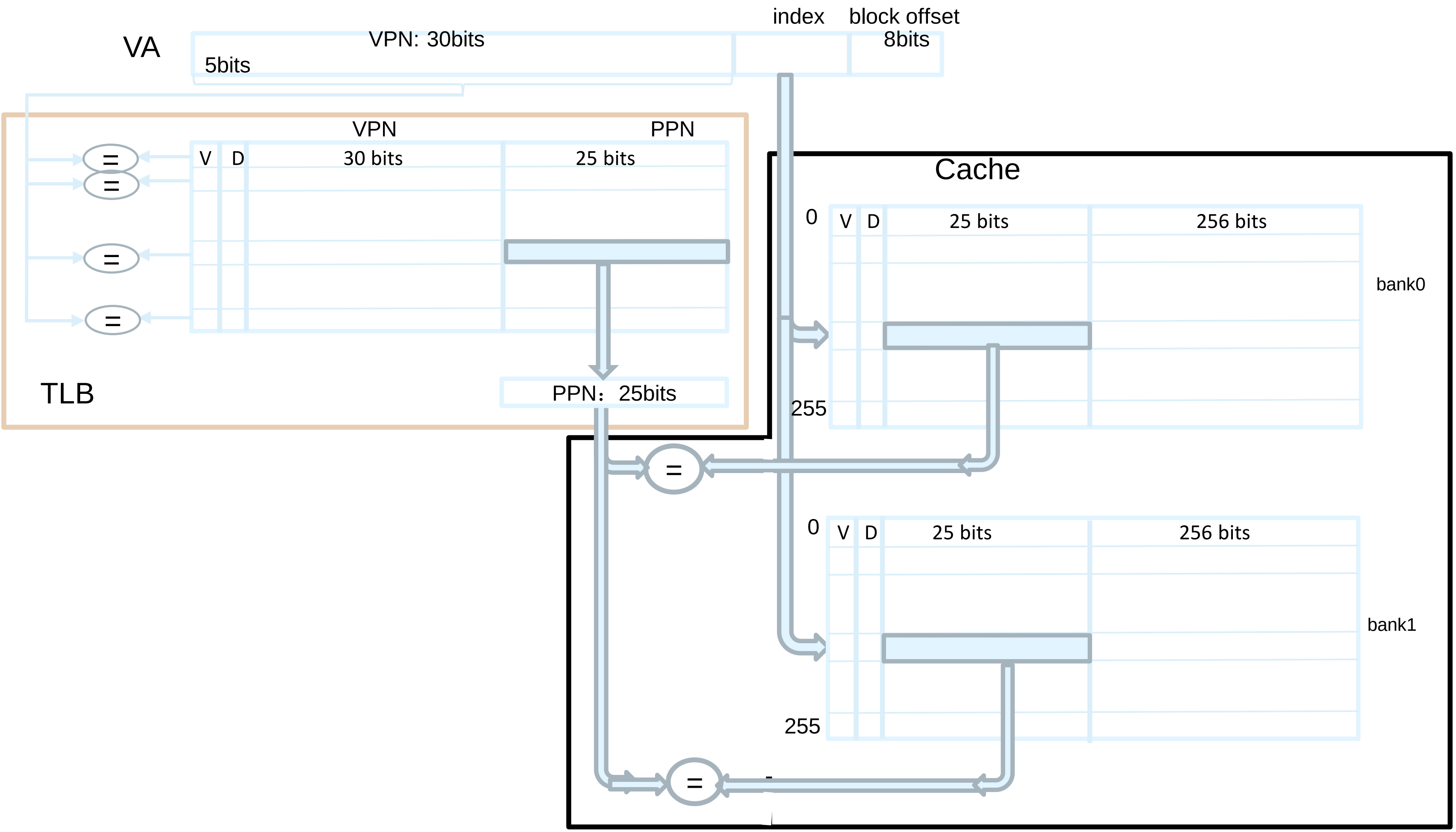
Cache size = $2^{\text{index}} * 2^{\text{blockoffset}} <= 2^{\text{pageoffset}}$

How to solve this problem?

Use higher association. Say it’s a 4 way, then cache size can reach 4 times $2^{\text{pageoffset}}$ with change nothing in index or tag or offset.

Example: Virtual indexed, physically tagged cache

Virtual address wide = 43 bits, Memory physical address wide = 38 bits, Page size = 8KB. Cache capacity =16KB. If a virtually indexed and physically tagged cache is used. And the cache is 2-way associative write back cache with 32 byte block size.



4th Hit Time Reduction Technique: Trace caches

- Find a dynamic sequence of instructions including taken branches to load into a cache block.
- The block determined by CPU instead of by memory layout.
- Complicated address mapping mechanism

Why Trace Cache ?

- Bring N instructions per cycle
  - No I-cache misses
  - No prediction miss
  - No packet breaks !

Because branch in each 5 instruction, so cache can only provide a packet in one cycle.

What’s Trace ?

❑ Trace: dynamic instruction sequence

❑ When instructions (operations) retire from the pipeline, pack the instruction segments into TRACE, and store them in the TRACE cache, including the branch instructions.

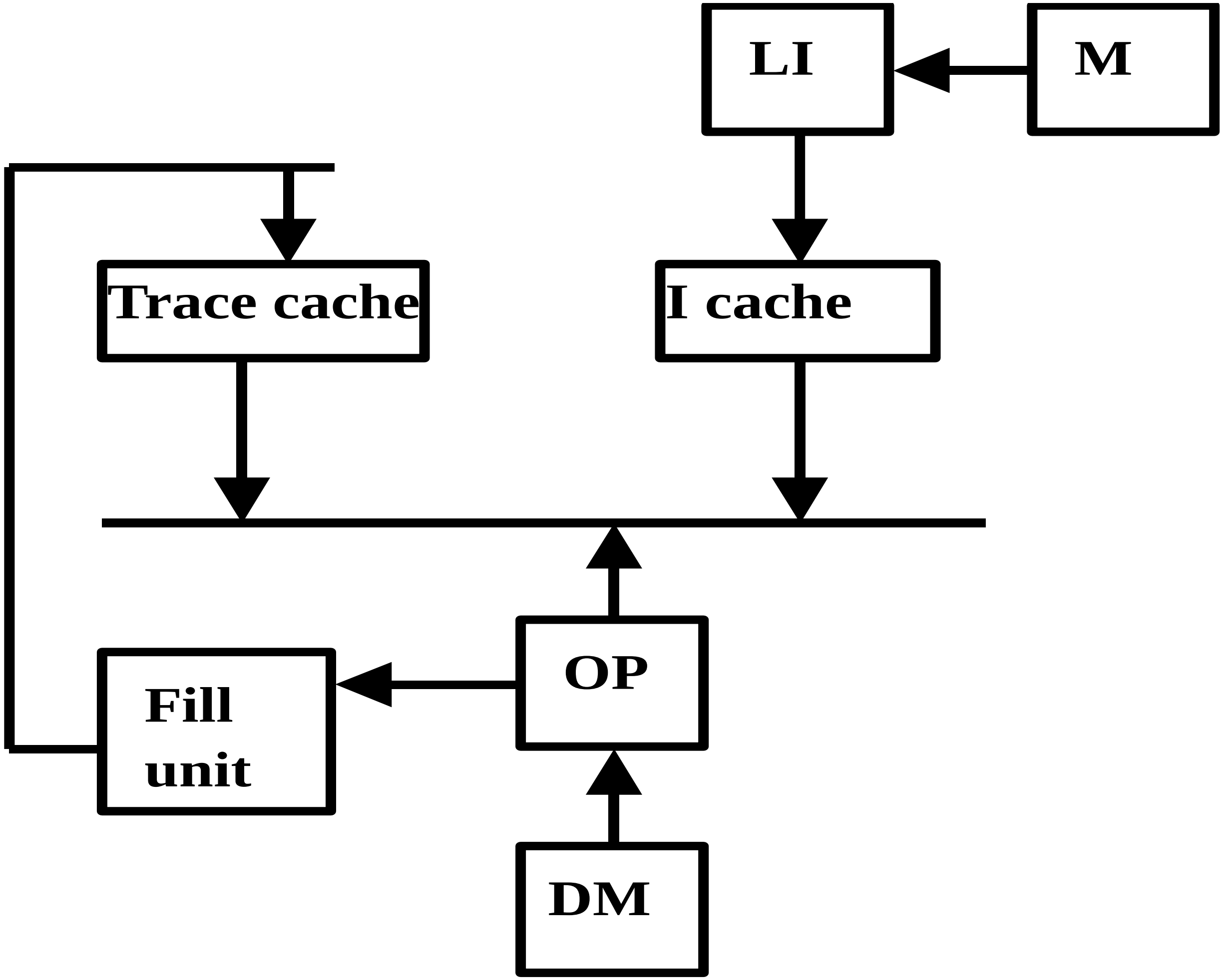
❑ Though branch instruction may go a different target, but **most times** the next operation sequential will just be the same as the last sequential. (locality)

Whose propose ?

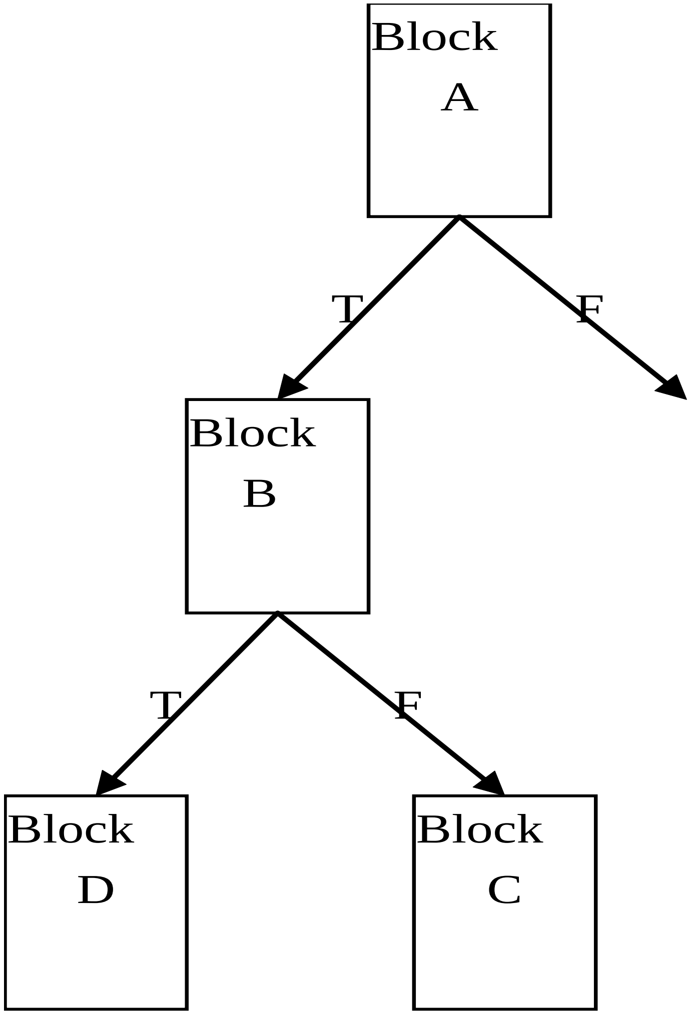
- ❑Peleg Weiser (1994) in Intel corporation
- ❑Patel / Patt (1996)
- ❑Rotenberg / J. Smith (1996)

- ❑Paper: ISCA’98

Trace in CPU



Instruction segment


Block A

T

F

Block B

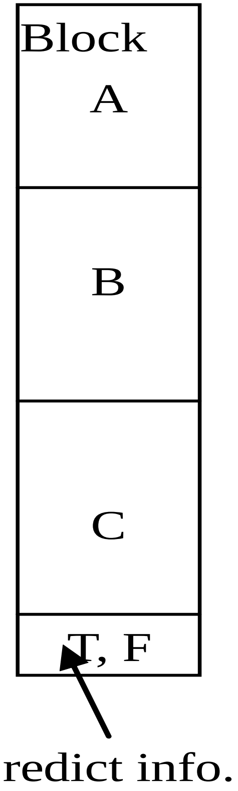
T

F

Block D

Block C

Fill unit pack them into :


Block A

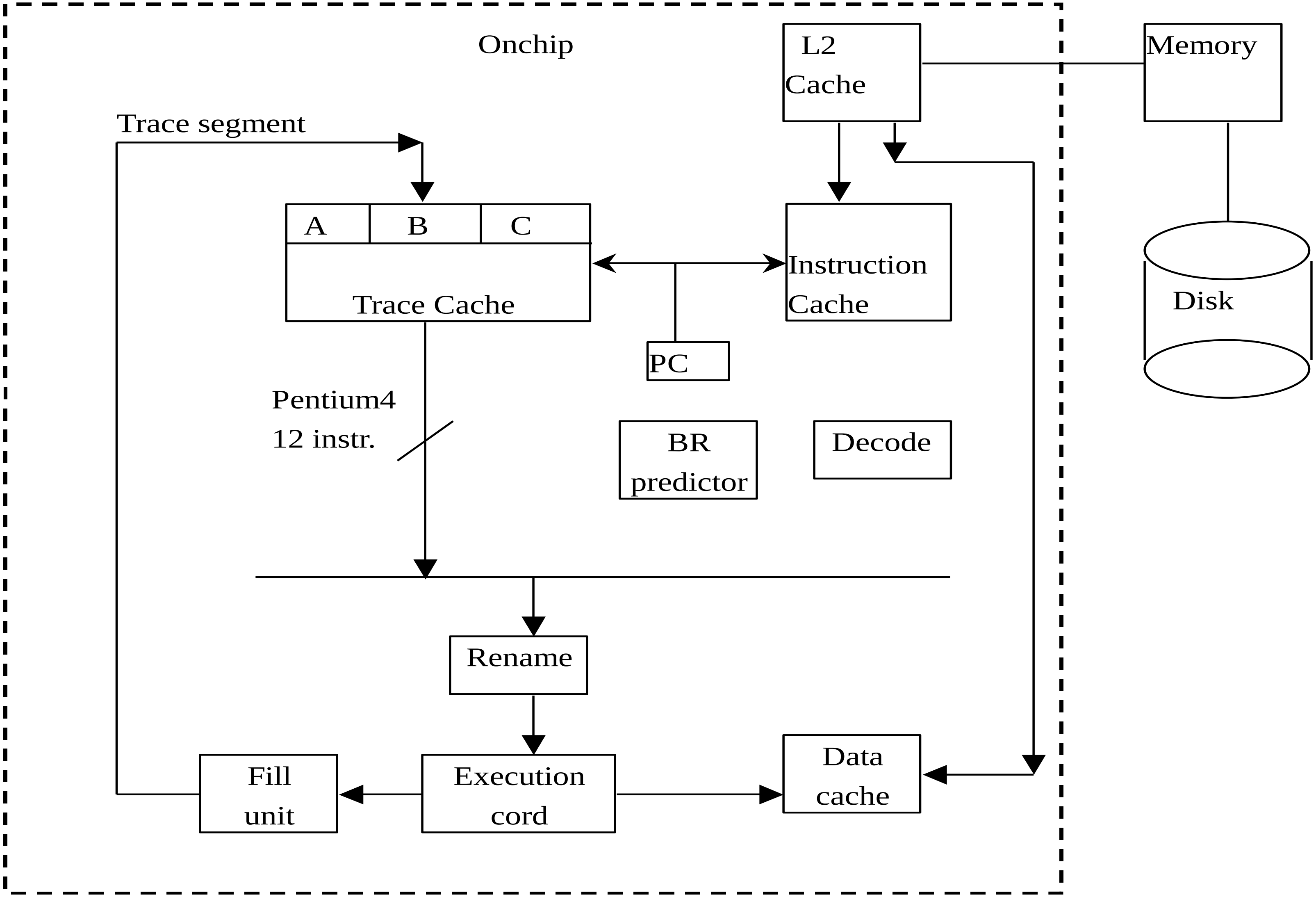
B

C

T, F

Predict info.

Pentium 4:
trace cache, 12 instr./per cycle



How to Improve Cache Performance?

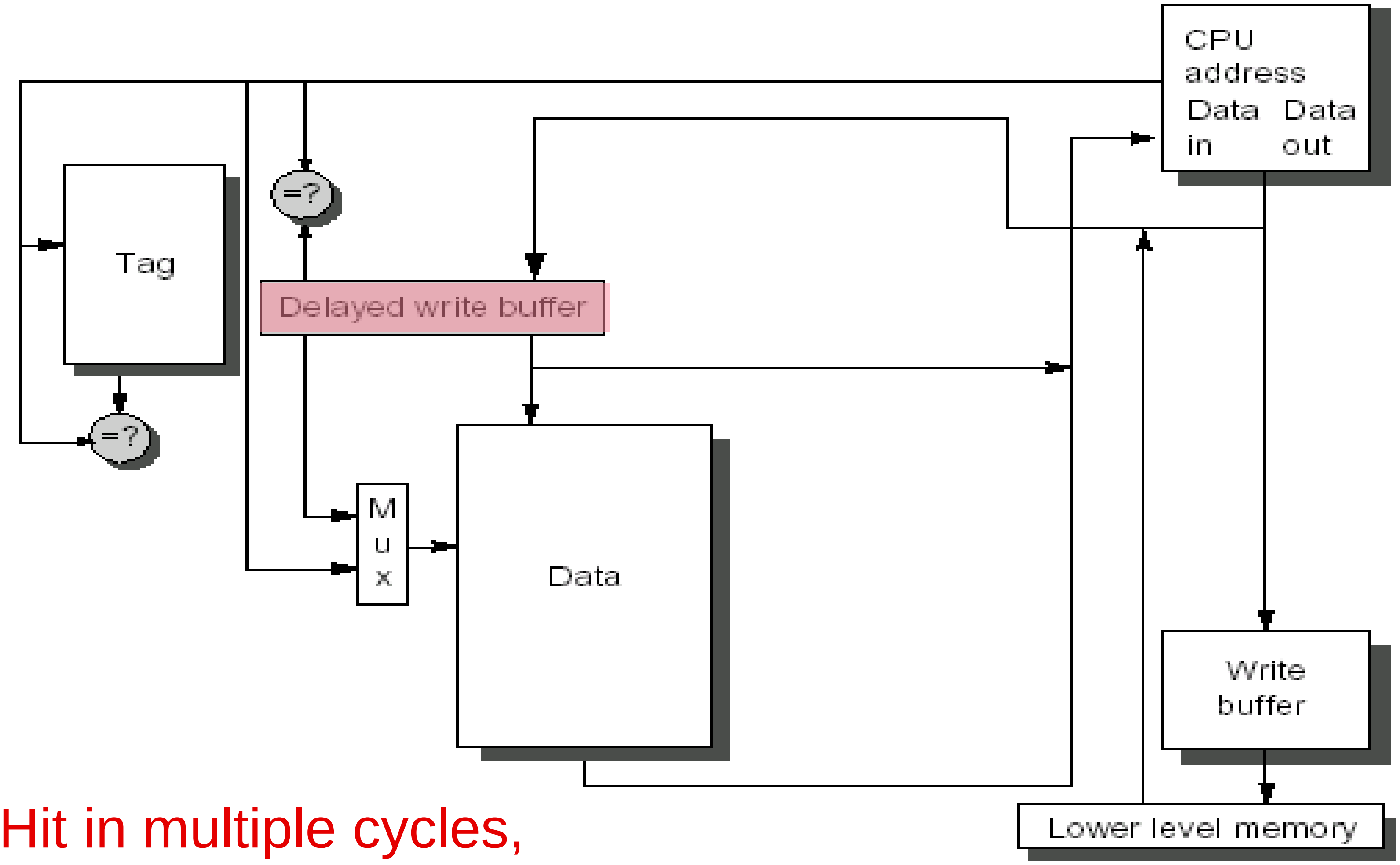
$$AMAT = HitTime + MissRate \times MissPenalty$$

- ❑Reduce hit time(4)
  - ➢ Small and simple first-level caches, Way prediction
  - ➢ avoiding address translation, Trace cache
- ❑Increase bandwidth(3)
  - ➢ Pipelined caches, multibanked caches, non-blocking caches
- ❑Reduce miss penalty(5)
  - ➢ multilevel caches, read miss prior to writes,
  - ➢ Critical word first, merging write buffers, and victim caches
- ❑Reduce miss rate(4)
  - ➢ larger block size,   large cache size, higher associativity
  - ➢ Compiler optimizations
- ❑Reduce miss penalty or miss rate via parallelization (1)
  - ➢ Hardware or compiler prefetching

Pipelined Caches

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

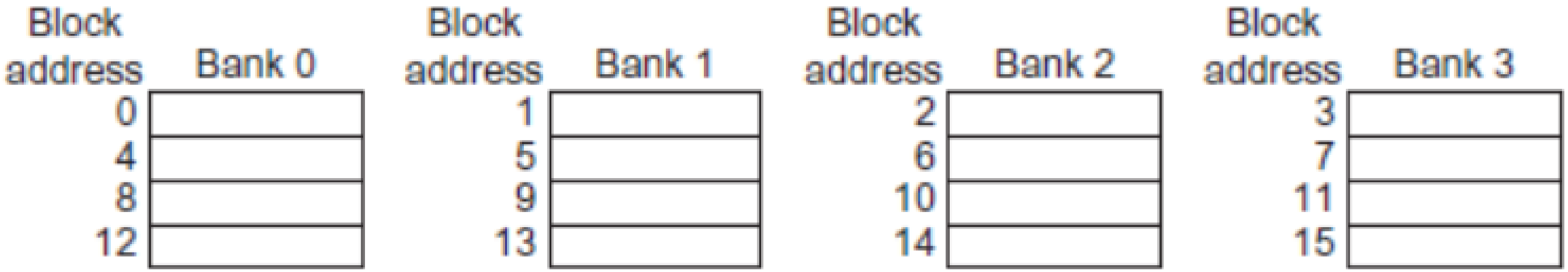
1st Increasing cache bandwidth: Pipelined Caches



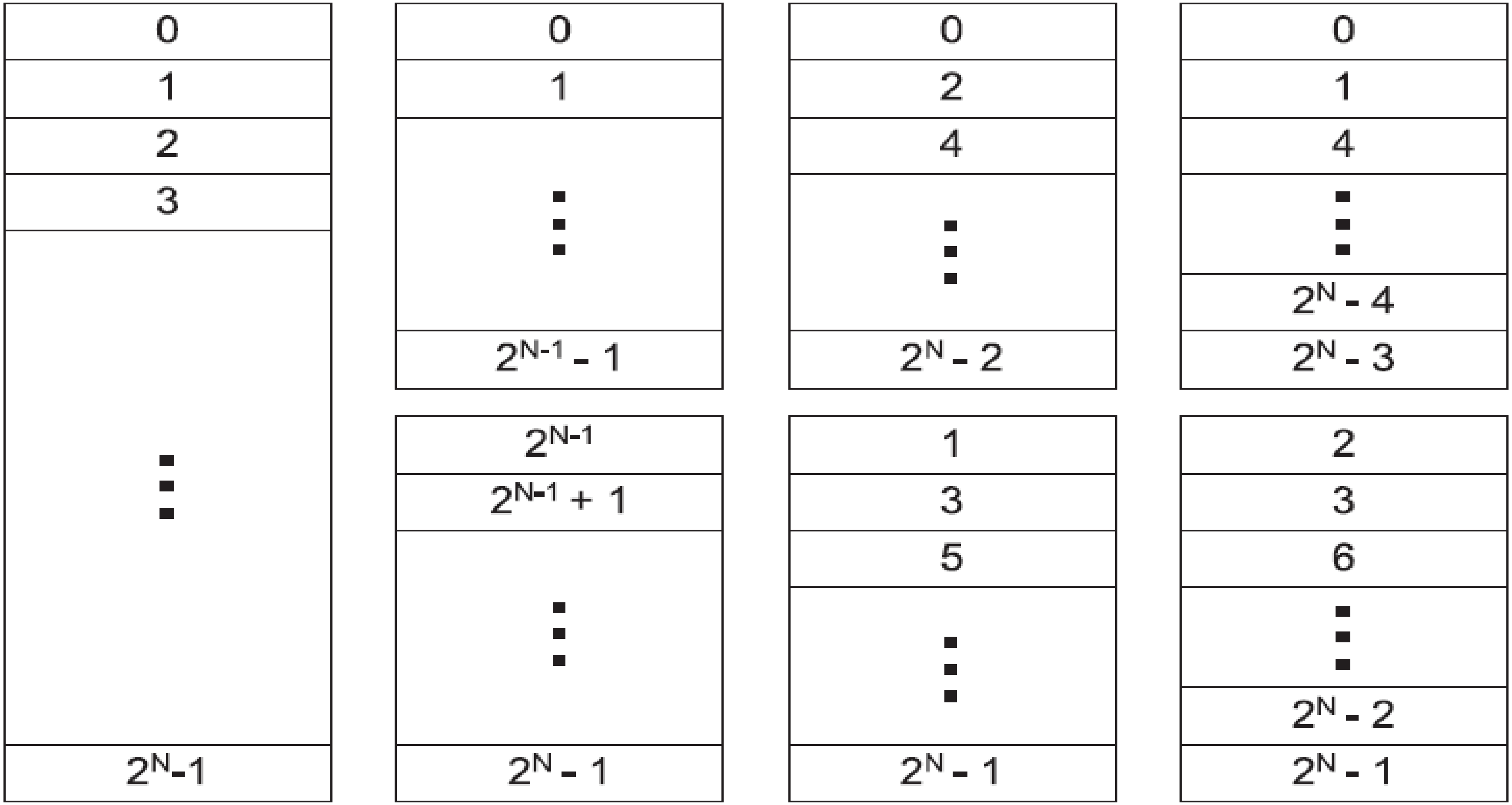
Hit in multiple cycles,
giving fast clock cycle time

2nd Increasing cache bandwidth: Multibanked Caches

- ❑Organize cache as independent banks to support simultaneous access
  - ➢ 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 ⇒ mapping of addresses to banks affects behavior of memory system
- ❑Interleave banks according to block address,Simple mapping that works well is “sequential interleaving”



Block address	Bank 0	Block address	Bank 1	Block address	Bank 2	Block address	Bank 3
0		1		2		3	
4		5		6		7	
8		9		10		11	
12		13		14		15	



(a) Single banked

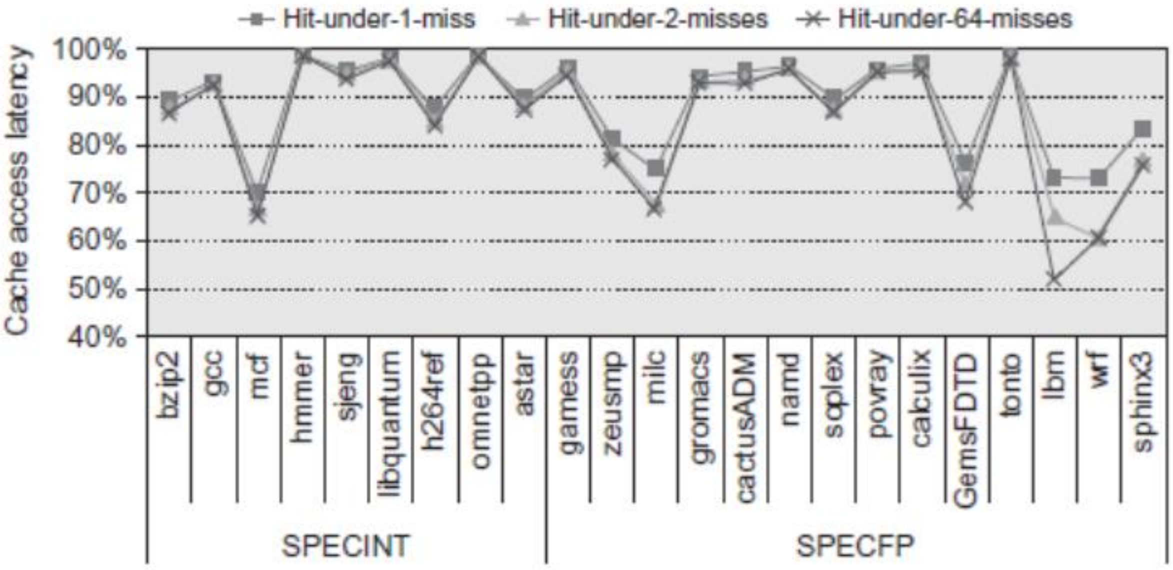
(b) two bank consecutive

(c) two bank interleaving

(d) two bank group interleaving

3rd Increasing cache bandwidth: Nonblocking Caches

- ❑ Allow hits before previous misses complete
  - ➢ "Hit under miss" ， "Hit under multiple miss"
- ❑ L2 must support this
- ❑ In general, processors can hide L1 miss penalty but not L2 miss penalty
- ❑ Nonblocking, in conjunction with out-of-order execution, can allow the CPU to continue executing instructions after a data cache miss.



Nonblocking cache

假设要执行下面一段程序：
1 Reg1:=LoadMem(A);
2 Reg2:=LoadMem(B);
3 Reg3:=Reg1 + Reg2;
当执行第一行时，cpu发现地址A不在cache中，就需要去内存读。但读内存的时间是很长的，此时CPU也不会闲着，就去执行了第二行。然后发现B也不在cache中。那么此时cache会怎么做呢？

- (a). cache阻塞，等着先把A读进来，然后再去读B。这种叫做Blocking Cache
- (b). cache同时去内存读B，最终B和A一起进入Cache。这种叫做Non-Blocking Cache

How to Improve Cache Performance?

$$AMAT = HitTime + MissRate \times MissPenalty$$

- ❑Reduce hit time(4)
  - ➢ Small and simple first-level caches, Way prediction
  - ➢ avoiding address translation, Trace cache
- ❑Increase bandwidth(3)
  - ➢ Pipelined caches, multibanked caches, non-blocking caches
- ❑Reduce miss penalty(5)
  - ➢ multilevel caches, read miss prior to writes,
  - ➢ Critical word first, merging write buffers, and victim caches
- ❑Reduce miss rate(4)
  - ➢ larger block size,   large cache size, higher associativity
  - ➢ Compiler optimizations
- ❑Reduce miss penalty or miss rate via parallelization (1)
  - ➢ Hardware or compiler prefetching

1st Miss Penalty Reduction Technique: Multilevel Caches

- This method focuses on the interface between the cache and main memory.
- Add an second-level cache between main memory and a small, fast first-level cache, **to make the cache fast and large.**
- The smaller first-level cache is fast enough to match the clock cycle time of the fast CPU and to fit on the chip with the CPU, thereby lessening the hits time.
- The second-level cache can be large enough to capture many memory accesses that would go to main memory, thereby lessening the effective miss penalty.

Parameter about Multilevel cache

- The performance of a two-level cache is calculated in a similar way to the performance for a single level cache.
- L2 Equations

So the miss penalty for level 1 is calculated using the hit time, miss rate, and miss penalty for the level 2 cache.

AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x Miss Penalty$_{L1}$

Miss Penalty$_{L1}$ = Hit Time$_{L2}$ + Miss Rate$_{L2}$ x Miss Penalty$_{L2}$

$$Miss\ rate_{L1} = \frac{Misses_{L1}}{M}$$

$$Miss\ rate_{L2} = \frac{Misses_{L2}}{M_{L2}} = \frac{Misses_{L2}}{M \times Miss\ rate_{L1}}$$

AMAT = Hit Time$_{L1}$ +
Miss Rate$_{L1}$ x (Hit Time$_{L2}$ + Miss Rate$_{L2}$ * Miss Penalty$_{L2}$)

Two conceptions for two-level cache

❑ 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 the CPU

Using the terms above, the global miss for the first-level cache is stall just Miss rate$_{L1}$, but for the second-level cache it is :

$$Miss\ rate_{Global-L2} = \frac{Misses_{L2}}{M} = \frac{M \times Miss\ rate_{L1}}{M} \times \frac{Misses_{L2}}{M \times Miss\ rate_{L1}}$$

$$= \frac{Misses_{L1}}{M} \times \frac{Misses_{L2}}{M \times Miss\ rate_{L1}} = Miss\ rate_{L1} \times Miss\ rate_{L2}$$

2nd Miss Penalty Reduction Technique:

Giving Priority to Read Misses over Writes

- If a system has a write buffer, writes can be delayed to come after reads.
- The system must, however, be careful to check the write buffer to see if the value being read is about to be written.

Write buffer

- Write-back want buffer to hold displaced blocks
  - –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

- Write-through want write buffers => RAW conflicts with main memory reads on cache misses
  - –If simply wait for write buffer to empty, might increase read miss penalty (old MIPS 1000 by 50%)
  - –Check write buffer contents before read;
    if no conflicts, let the memory access continue

3rd Miss Penalty Reduction Technique: Critical Word First & Early Restart

- Don't wait for full block to be loaded before restarting CPU
  - *Critical Word First*—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU 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 CPU and let the CPU continue execution
- Generally useful only in large blocks,
- Spatial locality => tend to want next sequential word, so not clear if benefit by early restart

Example: Critical Word First

Assume:

cache block＝64-byte

L2: take 11 CLK to get the critical 8 bytes,(AMD Athlon) and then 2 CLK per 8 byte to fetch the rest of the block

There will be no other accesses to rest of the block

Calculate the average miss penalty for critical word first.

Then assuming the following instructions read data sequentially 8 bytes at a time from the rest of the block

Compare the times with and without critical word first.

4th Miss Penalty Reduction Technique: Merging write Buffer

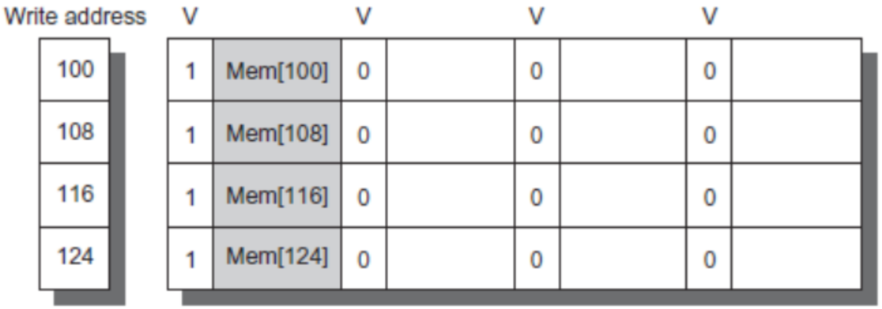
- ❑ One word writes replaces with multiword writes, and it improves buffers's efficiency.

- ❑ In write-through ,When write misses if the buffer contains other modified blocks,the addresses can be checked to see if the address of this new data matches the address of a valid write buffer entry.If so,the new data are combined with that entry.

- ❑ The optimization also reduces stalls due to the write buffer being full.

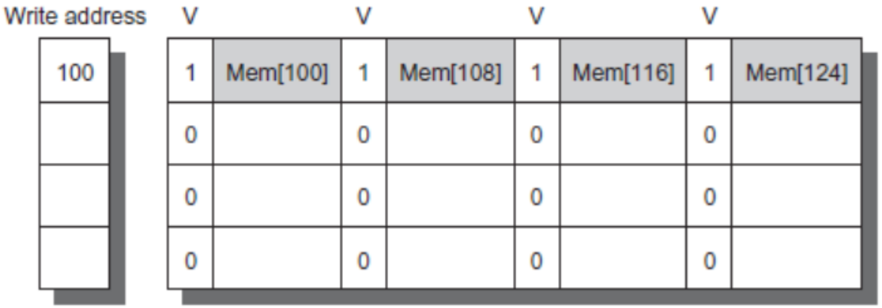
Merging Write Buffer

- ❑When storing to a block that is already pending in the write buffer, update write buffer
- ❑Reduces stalls due to full write buffer
- ❑Do not apply to I/O addresses



Write address	V		V		V		V	
100	1	Mem[100]	0		0		0	
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

No write buffering



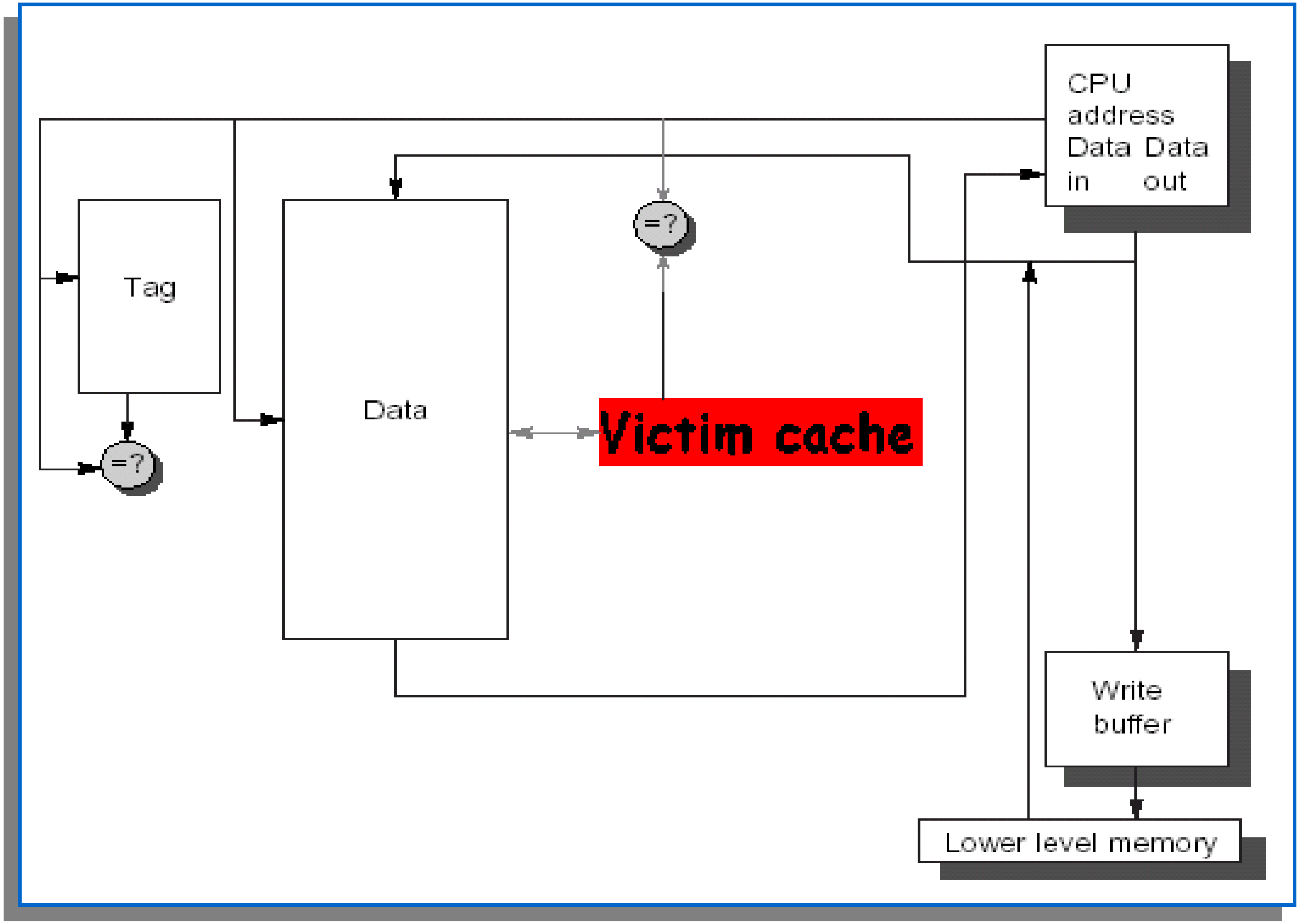
Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

Write buffering

5th Miss Penalty Reduction Technique: Victim Caches

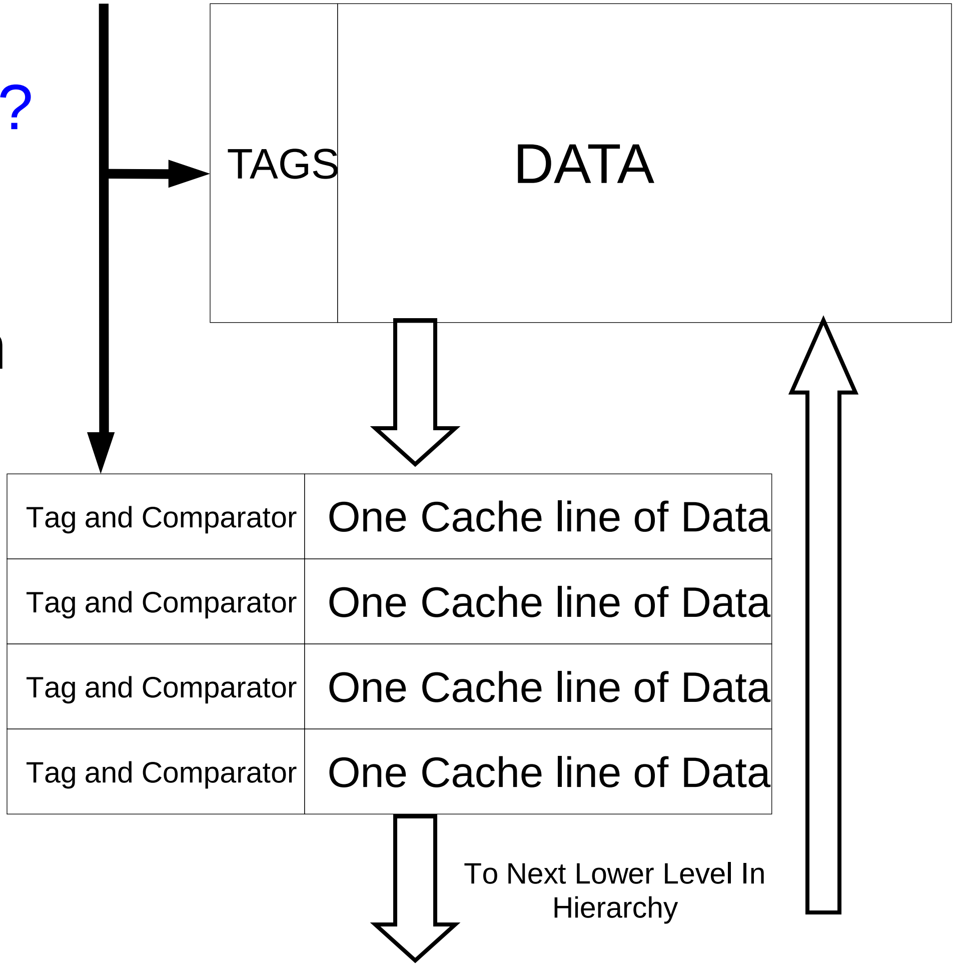
- A victim cache is a small (usually, but not necessarily) fully-associative cache that holds a few of the most recently replaced blocks or victims from the main cache.
- This cache is checked on a miss data before going to next lower-level memory(main memory).
  - to see if they have the desired item
  - If found, the victim block and the cache block are swapped.
  - The AMD Athlon has a victim caches (write buffer for write back blocks) with 8 entries.

The Victim Cache



How to combine victim Cache ?

- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP machines



Tag and Comparator	One Cache line of Data
Tag and Comparator	One Cache line of Data
Tag and Comparator	One Cache line of Data
Tag and Comparator	One Cache line of Data

How to Improve Cache Performance?

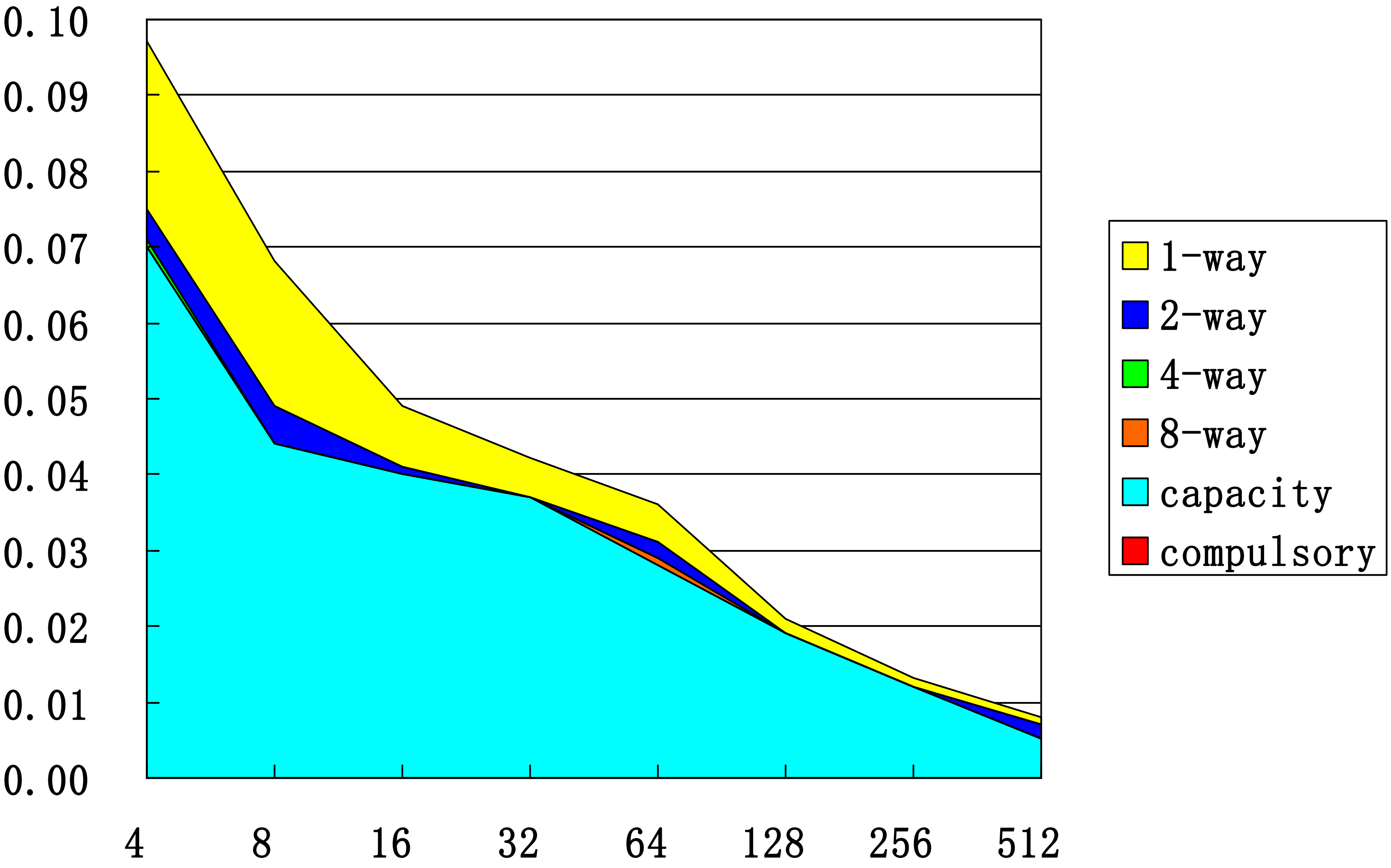
$$AMAT = HitTime + MissRate \times MissPenalty$$

- ❑Reduce hit time(4)
  - ➢ Small and simple first-level caches, Way prediction
  - ➢ avoiding address translation, Trace cache
- ❑Increase bandwidth(3)
  - ➢ Pipelined caches, multibanked caches, non-blocking caches
- ❑Reduce miss penalty(5)
  - ➢ multilevel caches, read miss prior to writes,
  - ➢ Critical word first, merging write buffers, and victim caches
- ❑Reduce miss rate(4)
  - ➢ larger block size, large cache size, higher associativity
  - ➢ Compiler optimizations
- ❑Reduce miss penalty or miss rate via parallelization (1)
  - ➢ Hardware or compiler prefetching

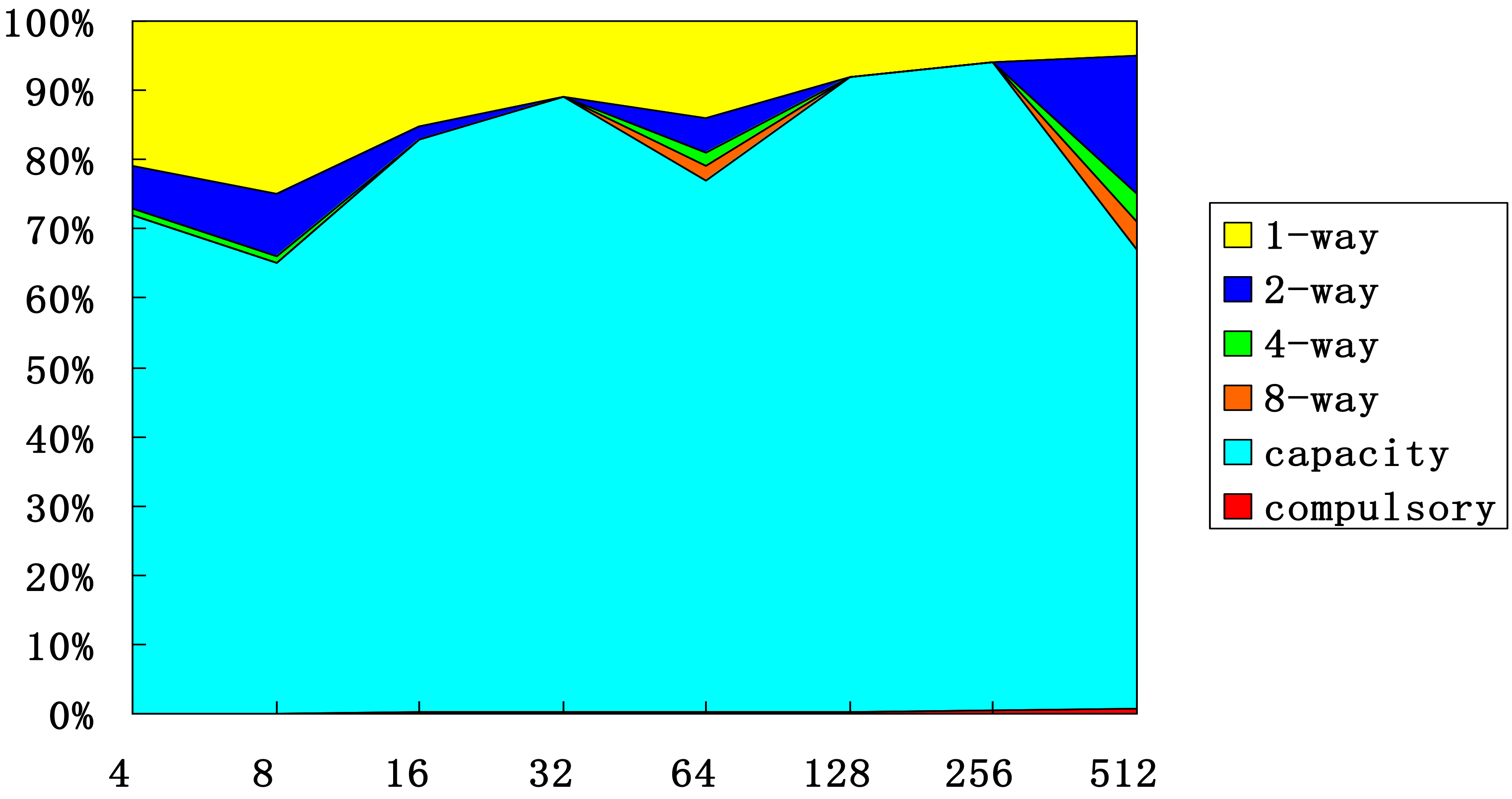
Where misses come from?

- Classifying Misses: 3 Cs
  - Compulsory—The first access to a block is not in the cache, so the block must be brought into the cache. Also called *cold start misses* or *first reference misses*.
    (Misses in even an Infinite Cache)
  - Capacity—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.
    (Misses in Fully Associative Size X Cache)
  - Conflict—If block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory & capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. Also called *collision misses* or *interference misses*.
    (Misses in N-way Associative, Size X Cache)
- 4th "C":
  - Coherence - Misses caused by cache coherence.

3Cs Absolute Miss Rate (SPEC92)



3Cs Relative Miss Rate



Flaws: for fixed block size

Good: insight => invention

Reducing Cache Miss Rate

- To reduce cache miss rate, we have to eliminate some of the misses due to the three C's.

- We cannot reduce capacity misses much except by making the cache larger.

- We can, however, reduce the conflict misses and compulsory misses in several ways:

Cache Organization?

- Assume total cache size not changed:
- What happens if:

1) Change Block Size:

2) Change Associativity:

3) Change Compiler:

Which of 3Cs is obviously affected?

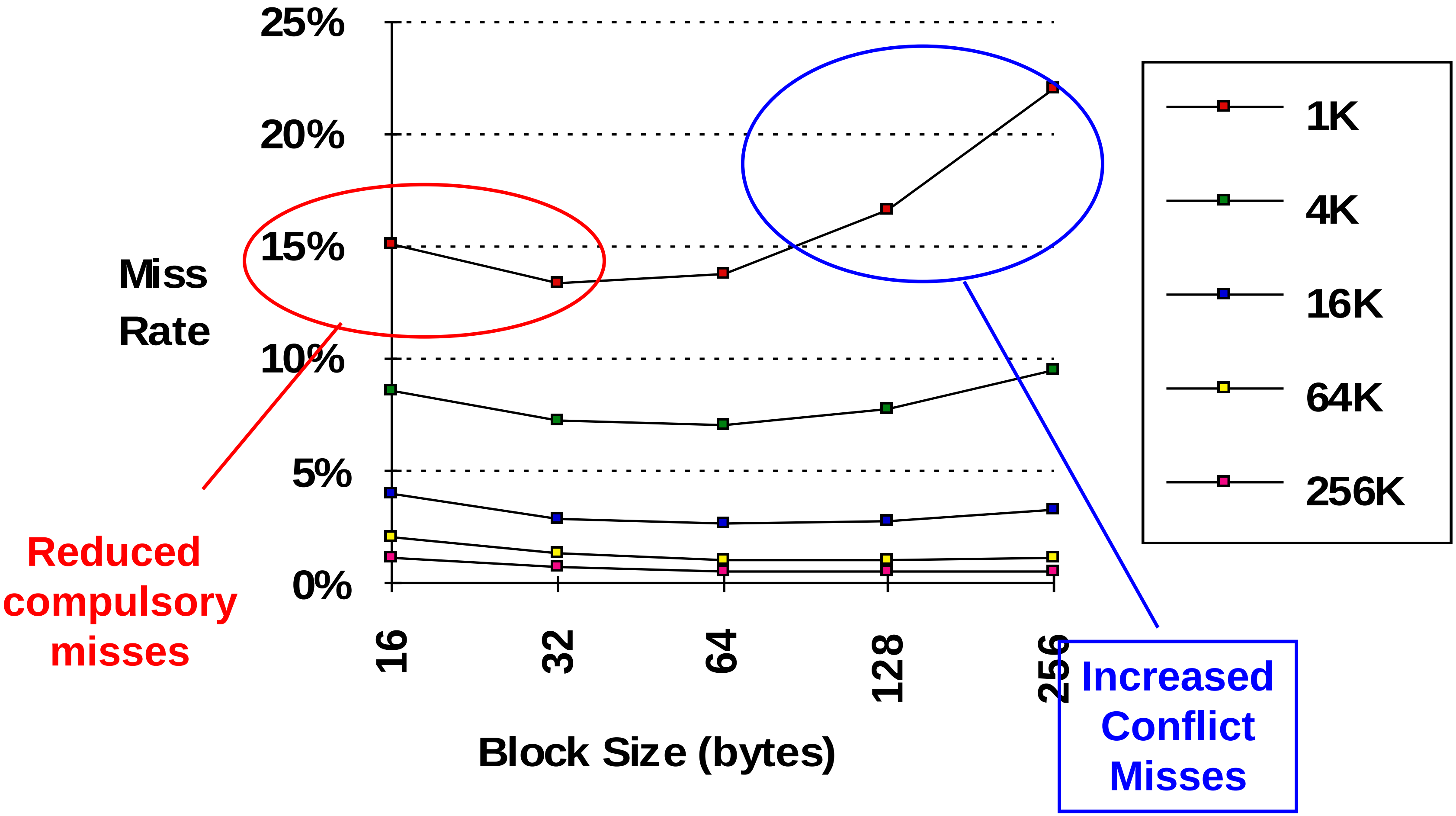
1st Miss Rate Reduction Technique: Larger Block Size (fixed size&assoc)

- Larger blocks decrease the compulsory miss rate by taking advantage of spatial locality.
- Drawback--curve is U-shaped
  - However, they may increase the miss penalty by requiring more data to be fetched per miss.
  - In addition, they will almost certainly increase conflict misses since fewer blocks can be stored in the cache.
  - And maybe even capacity misses in small caches
- Trade-off
  - Trying to minimize both the miss rate and the miss penalty.
  - The selection of block size depends on both the latency and bandwidth of lower-level memory

Miss Rate relates Block size

Block size	Cache 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%

Performance curve is U-shaped



Reduced compulsory misses

Increased Conflict Misses

What else drives up block size?

Example: Larger Block Size (C-26)

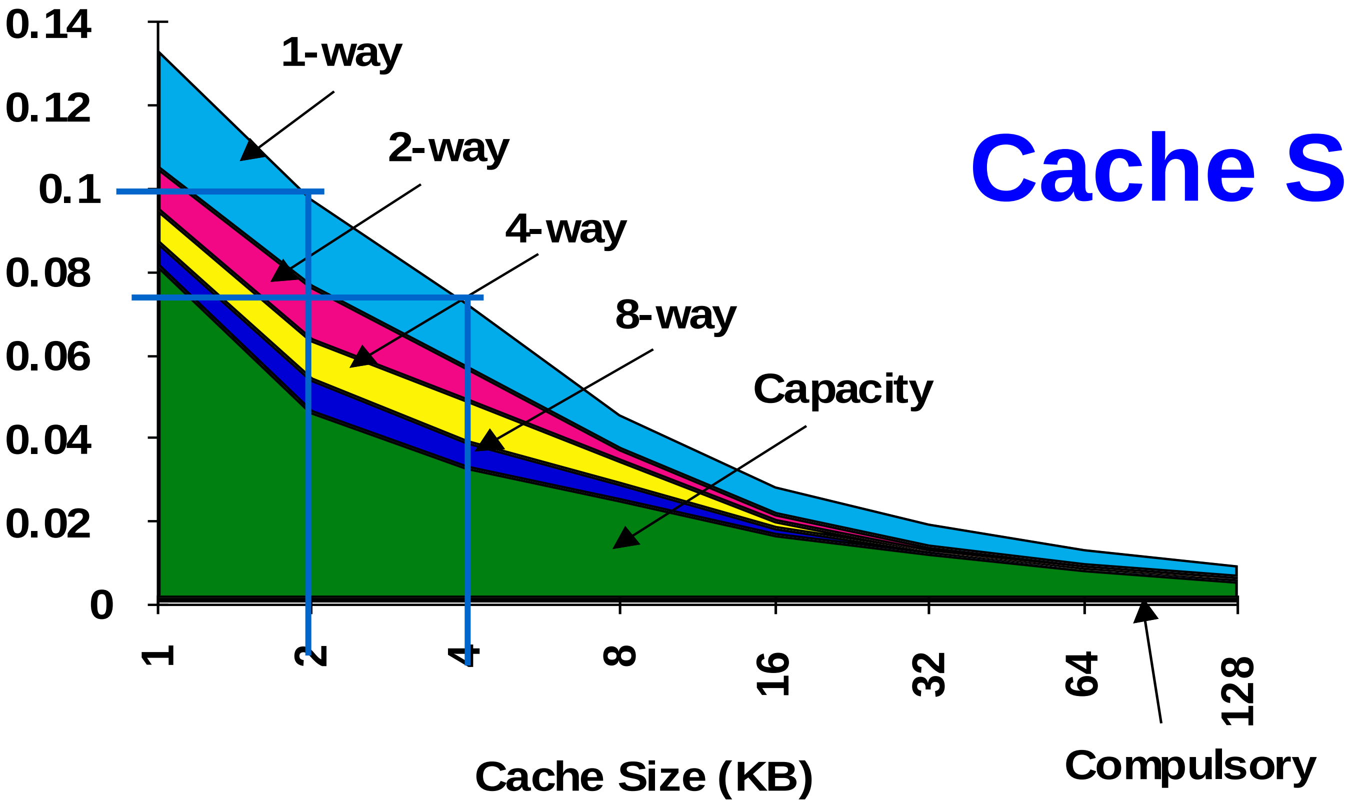
- ❑Assume: memory takes 80 clock cycles of overhead and then delivers 16 bytes every 2 cycles.
- ❑1 clock cycle hit time independent of block size.
- ❑Which block size has the smallest AMAT for each size in Fig.5.17 ?
- ❑Answer:

$\text{AMAT}_{\text{16-byte block, 4KB}} = 1+(8.57\%*82)=8.027$

$\text{AMAT}_{\text{256-byte block，256KB}}= 1+(0.49\%*112)=1.549$

2nd Miss Rate Reduction Technique:

Larger Caches



Cache Size

- rule of thumb: 2 x size => 25% cut in miss rate
- What does it reduce ?

Pro. Vs. cons for large caches

- Pro.
  - Reduce capacity misses
- Con.
  - Longer hit time, Higher cost, AMAT curve is U-shaped
- Popular in off-chip caches

Block size	Miss penalty	Cache size			
		4K	16K	64K	256K
16	82	8.027	4.231	2.673	1.894
32	84	7.082	3.411	2.134	1.588
64	88	7.160	3.323	1.933	1.449
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

3rd Miss Rate Reduction Technique: Higher Associativity

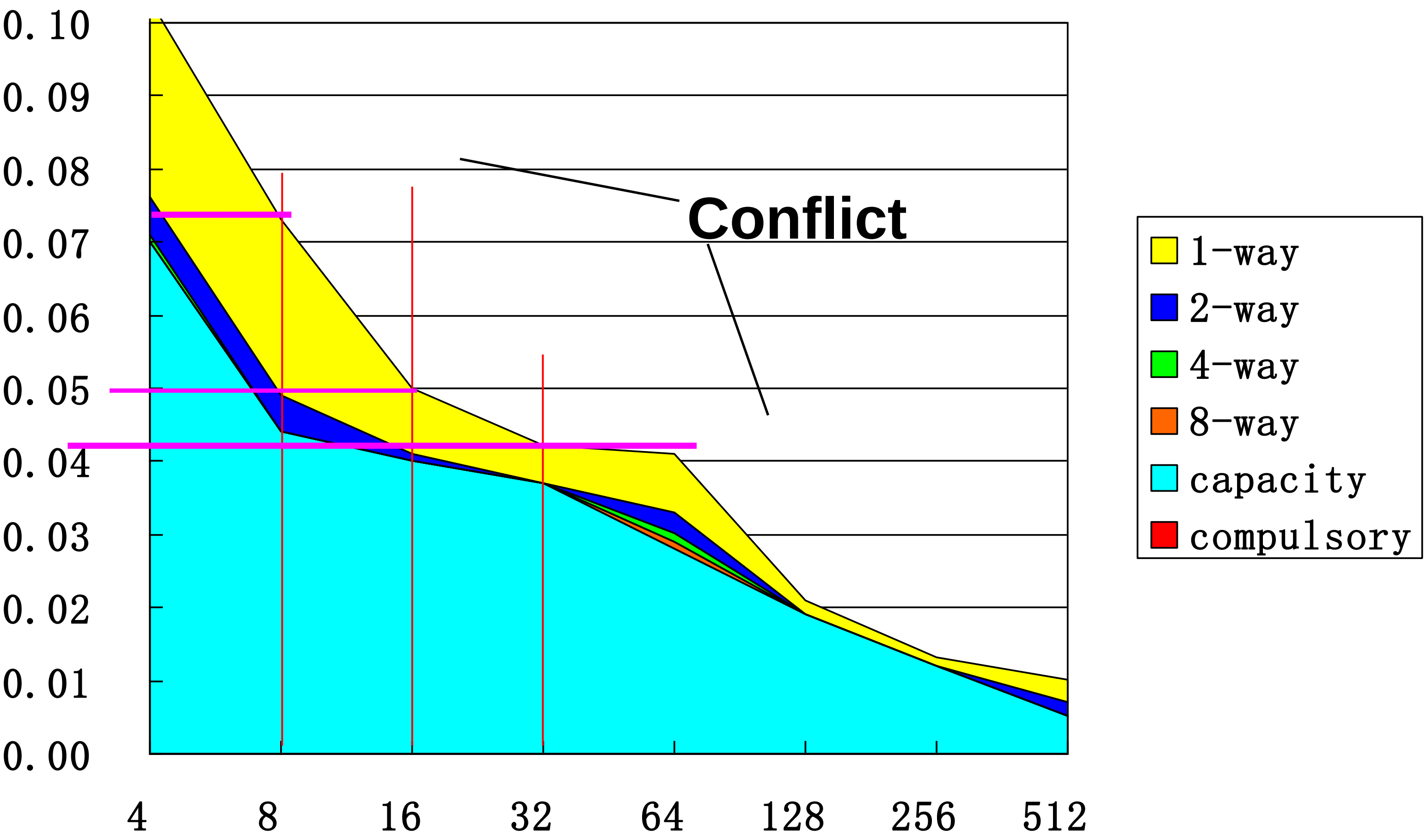
- ❑Conflict misses can be a problem for caches with low associativity (especially direct-mapped).
- ❑With higher associativity decreasing Conflict misses to improve miss rate

cache rule of thumb

- ❑2:1 rule of thumb *a direct-mapped cache of size N has the same miss rate as a 2-way set-associative cache of size N/2.*
- ❑*Eight-way set associative is for practical purposes as effective in reducing misses for these sized cache as fully associative.*

Associativity

2:1 rule of thumb



Associativity vs Cycle Time

- Beware: Execution time is only final measure!
- Why is cycle time tied to hit time?

- Will Clock Cycle time increase?
  - Hill [1988] suggested hit time for 2-way vs. 1-way external cache +10%,
    internal + 2%
  - suggested big and dumb caches

AMAT vs. Miss Rate (P430)

- Example: assume CCT = 1.36 for 2-way, 1.44 for 4-way, 1.52 for 8-way vs. CCT direct mapped

Cache Size	Associativity			
(KB)	1-way	2-way	4-way	8-way
4	3.44	3.25	3.22	3.28
8	2.69	2.58	2.55	2.62
16	2.33	2.40	2.46	2.53
32	2.06	2.30	2.37	2.45
64	1.92	2.24	2.18	2.25
128	1.52	1.84	1.92	2.00
256	1.32	1.66	1.74	1.82
512	1.20	1.55	1.59	1.66

(Red means A.M.A.T. not improved by more associativity)

4th Miss Rate Reduction Technique:
Compiler Optimizations

- The techniques reduces miss rates *without* any hardware changes and reorders instruction sequence with compiler.
- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts(using tools they developed)
- Data
  - *Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays
  - *Loop Interchange*: change nesting of loops to access data in order stored in memory
  - *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

a. Merging Arrays

- Combining independent matrices into a single compound array.
- Improving spatial locality
- Example

```
/*before*/
Int val[SIZE];
Int key[SIZE];

/*after*/
Struct merge{
     int val;
     int  key;
}
Struct merge merged_array[SIZE]
```

b. Loop Interchange

By switching the order in which loops execute, misses can be reduced due to improvements in spatial locality.

```
/* 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;

c. Loop fusion

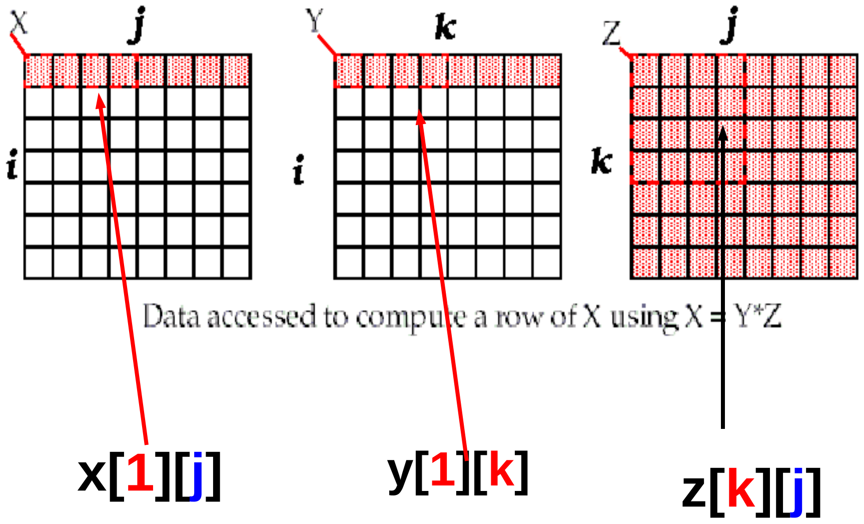
- ❑By fusion the code into a single loop, the data that are fetched into the cache can be used repeatedly before being swapped out.
- ❑Imporving the temporal locality
- ❑Example:

```
/*before*/
For (i=0; i<N; i=i+1)
For (i=0; j<N; j=i+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=i+1)
{

  a[i][j]=1/b[i][j]*c[i][j];
       d[i][j]=a[i][j]*c[i][j];
}
```

d. Unoptimized Matrix Multiplication

```
/* 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;
        };
```



Data accessed to compute a row of X using X = Y*Z

x[1][j]   y[1][k]   z[k][j]

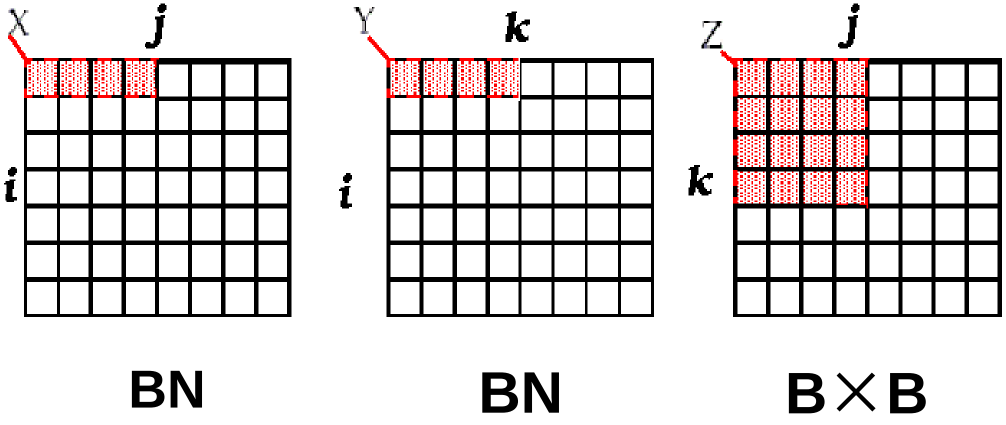
$((N+N)N+N)N=2N^3 + N^2$
Accessed For N^3 operations

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

Blocking optimized Matrix Multiplication

❑ *Matrix multiplication is performed by multiplying the submatrices first.*

```
/* 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*

$(BN+BN)+B^2)\times(N/B)^2=2N^3/B + N^2$

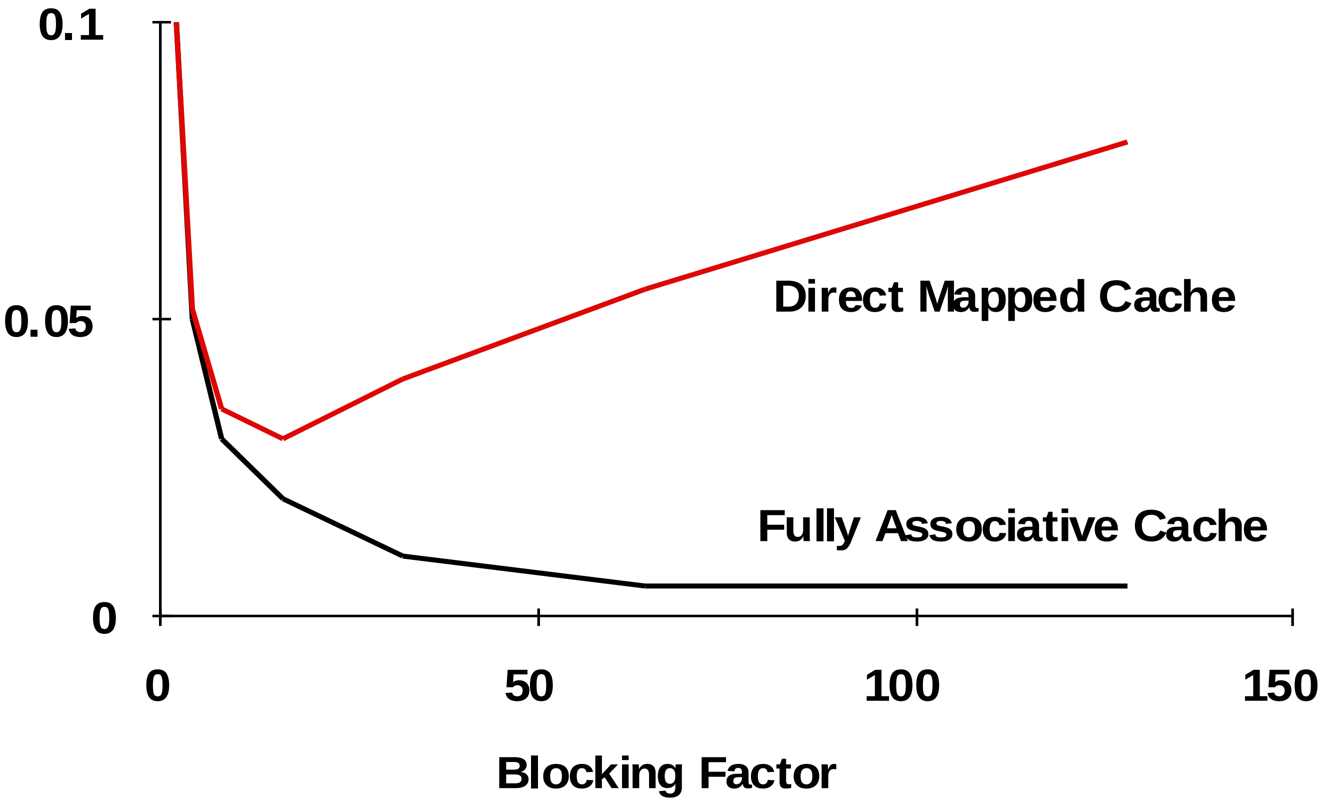
Accessed For N^3 operations

Y benefits from spatial locality

Z benefits from temporal locality

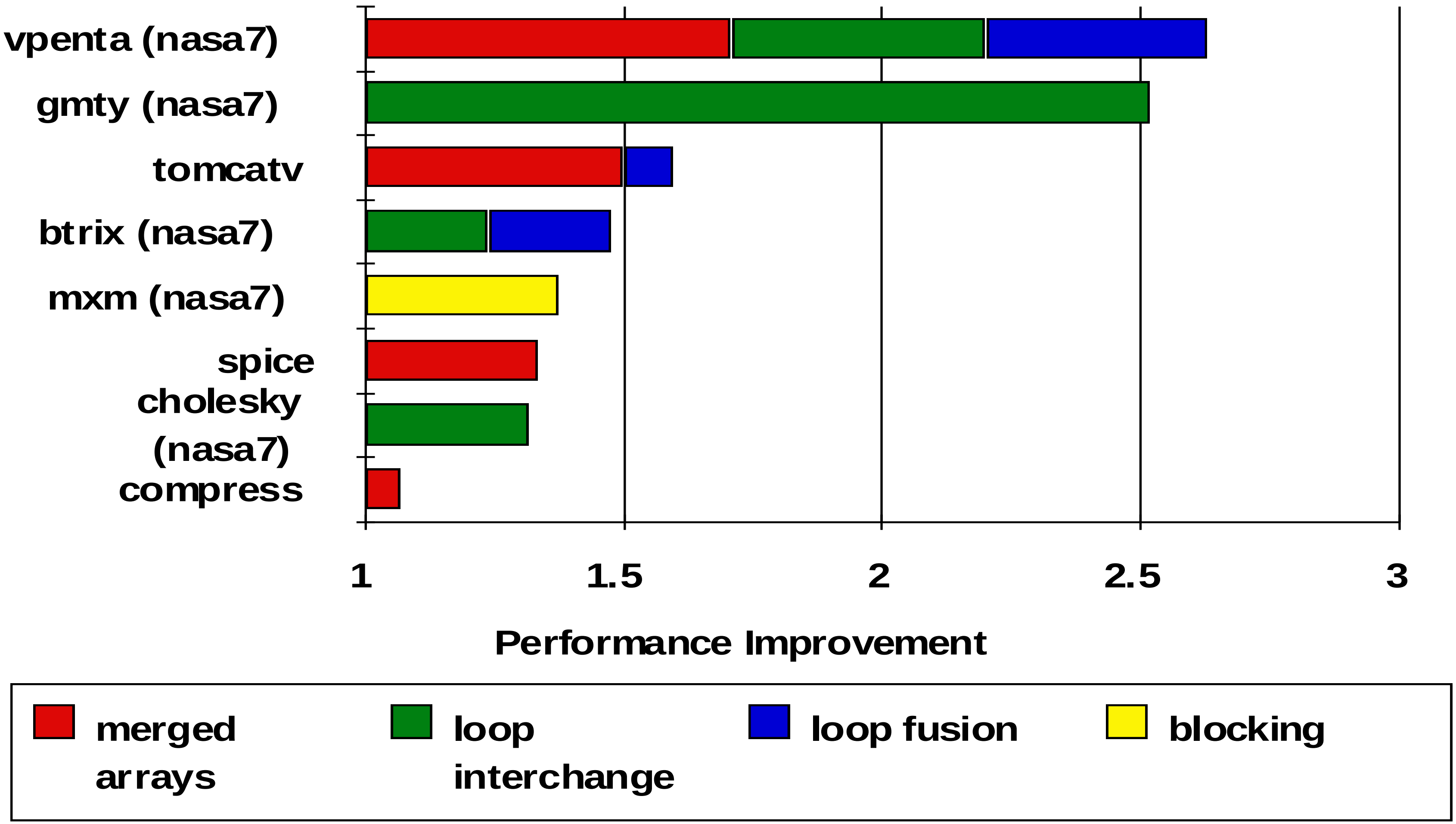
Capacity Misses from $2N^3 + N^2$ to $N^3/B+2N^2$

Reducing Conflict Misses by Blocking



- Conflict misses in caches not FA vs. Blocking size
  - Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache

Summary of Compiler Optimizations to Reduce Cache Misses (by hand)



5th Miss Rate Reduction Technique: Way Prediction and Pseudo-Associative Cache

Using two Technique reduces conflict misses and yet maintains hit speed of direct-mapped cache

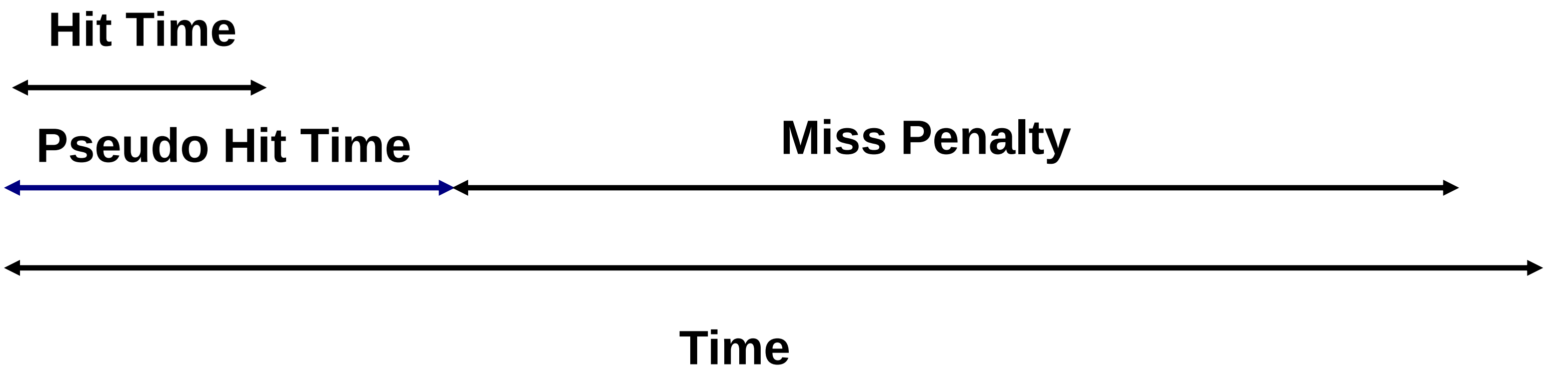
- Predictive bit - Pseudo-Associative

❑Way Prediction (Alpha 21264)

- Extra bits are kept in the cache to predict the way,or block within set of the *next* cache access.
- If the predictor is correct, the instruction cache latency is 1 clock clock cycle.
- If not,it tries the other block, changes the way predictor, and has a latency of 3 clock cycles.
- Simulation using SPEC95 suggested set prediction accuracy is excess of 85%, so way prediction saves pipeline stage in more than 85% of the instruction fetches.

Pseudo-Associative Cache (column associative)

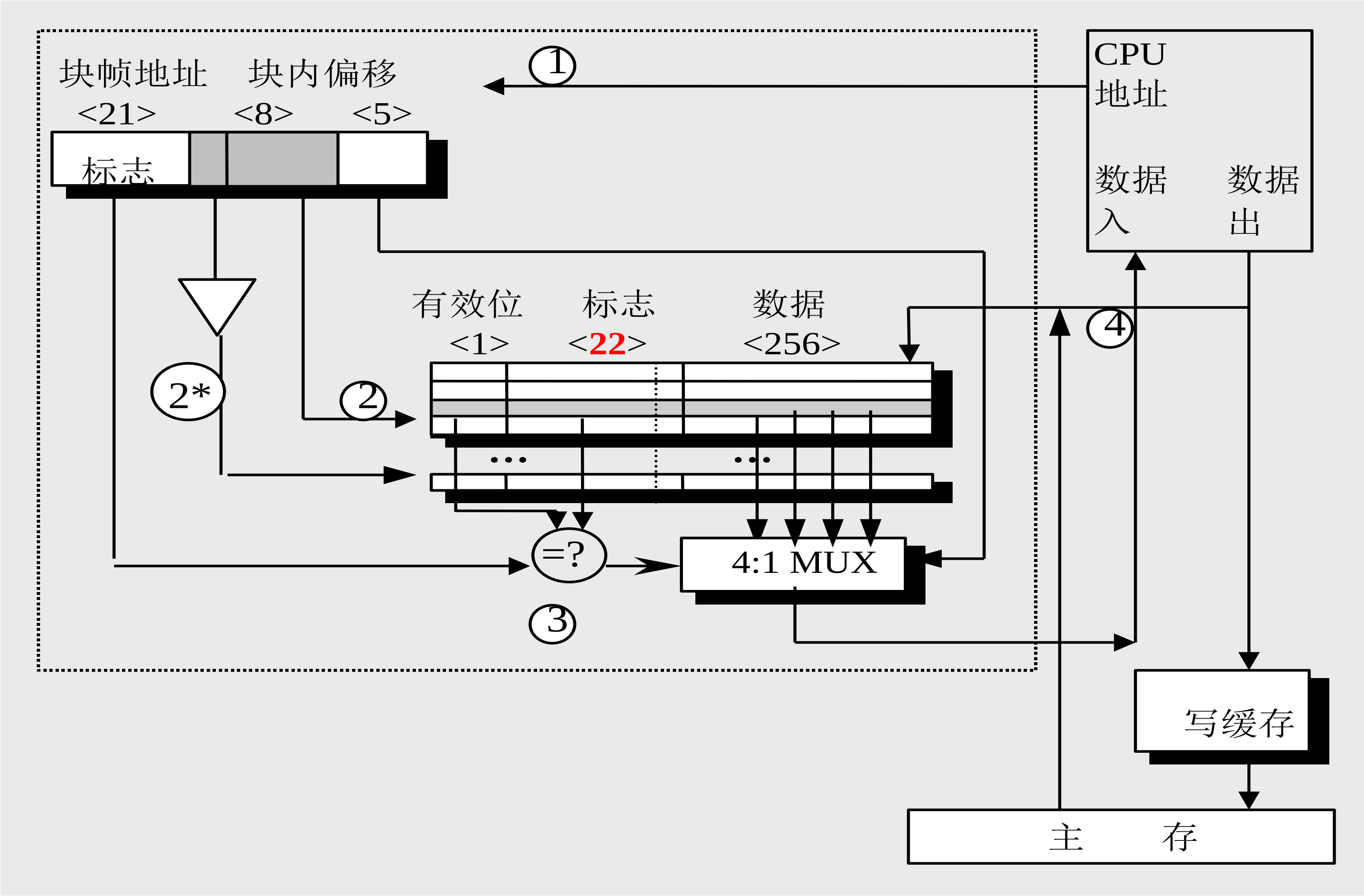
- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?
- Divide cache: on a miss, check other half of cache to see if there, if so have a pseudo-hit (slow hit)



Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles

- Better for caches not tied directly to processor (L2)
- Used in MIPS R1000 L2 cache, similar in UltraSPARC

Pseudo-Associative Cache



How to Improve Cache Performance?

$$AMAT = HitTime + MissRate \times MissPenalty$$

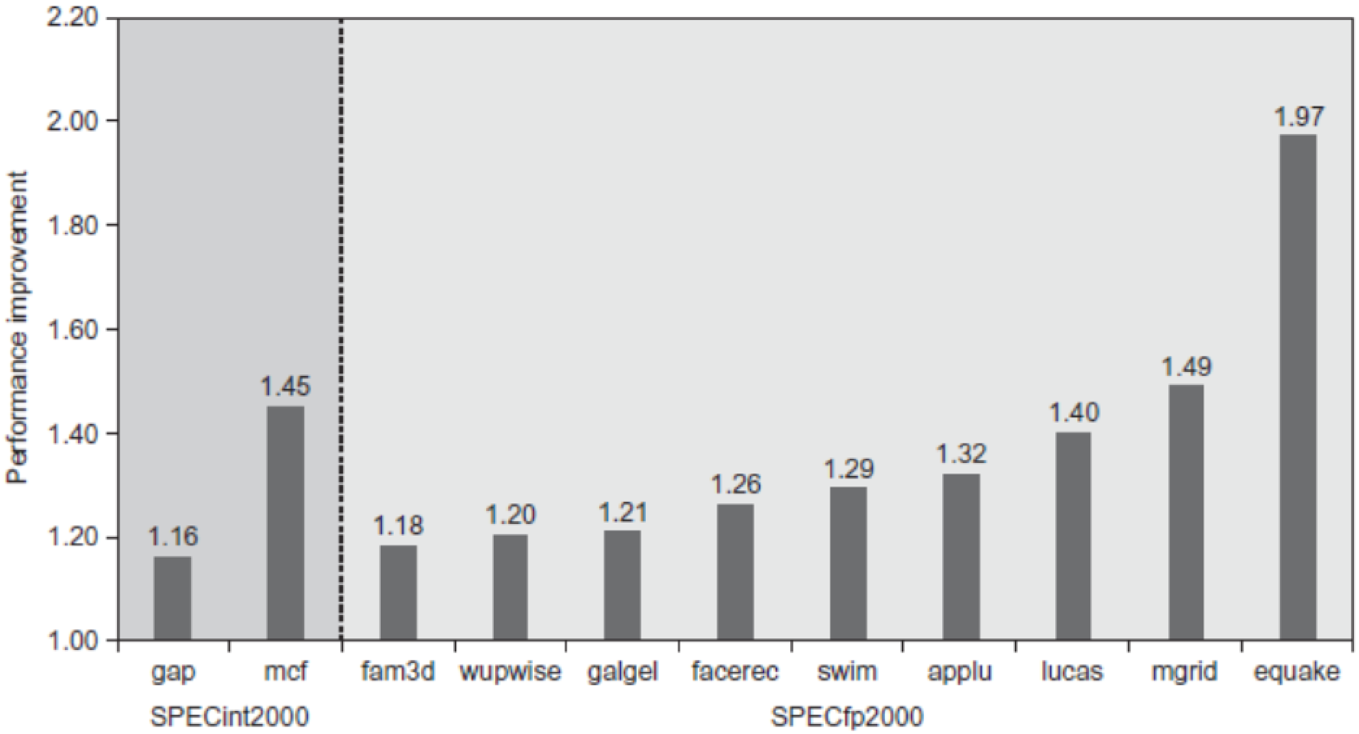
- Reduce hit time(4)
  - Small and simple first-level caches, Way prediction
  - avoiding address translation, Trace cache
- Increase bandwidth(3)
  - Pipelined caches, multibanked caches, non-blocking caches
- Reduce miss penalty(5)
  - multilevel caches, read miss prior to writes,
  - Critical word first, merging write buffers, and victim caches
- Reduce miss rate(4)
  - larger block size,   large cache size, higher associativity
  - Compiler optimizations
- Reduce miss penalty or miss rate via parallelization (1)
  - Hardware or compiler prefetching
- Using HBM to extend the memory hierarchy

1st Miss Penalty/Rate Reduction Technique: Hardware Prefetching of Inst.and data

- *The act of getting data from memory before it is actually needed by the CPU.*
- *This reduces compulsory misses by retrieving the data before it is requested.*
- *Of course, this may increase other misses by removing useful blocks from the cache.*
  - *Thus, many caches hold prefetched blocks in a* special buffer *until they are actually needed.*
- E.g., Instruction Prefetching
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in "stream buffer"
  - On miss check stream buffer
- Prefetching relies on having extra memory bandwidth that can be used without penalty

Hardware Prefetching

- Fetch two blocks on miss (include next sequential block)



Pentium 4 Pre-fetching

2nd Miss Penalty/Rate Reduction Technique: Compiler-controlled prefetch

- *The compiler inserts prefetch instructions before the data is needed*
- *Non-faulting: prefetch doesn't cause exceptions,* a form of speculative execution
- *Combine with loop unrolling and software pipelining*

- Register Prefetch (HP PA-RISC loads)
  - Binding prefetch: Requests load directly into register.
    - Must be correct address and register!
- Cache Prefetch (MIPS IV, PowerPC, SPARC v. 9)
  - Non-Binding prefetch: Load into cache.
    - Can be incorrect. Faults?

- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses ?
  - Higher superscalar reduces difficulty of issue bandwidth

Example: Compiler-controlled prefetch

```
for( i=0; i <3; i = i +1)
for( j=0; j<100; j=j+1)                          7 cycles/iteration
        a[i][j] = b[j][0] * b[j+1][0];
```

16B/block， 8B/element， 2elements/block

A[i][j]: 3*100 the even value of j will miss,
the odd values will hit, total 150 misses

B[i][j]：101*3 the same elements are accessed for each iteration of i

j=0	B[0][0]、B[1][0]	2
j=1	B[1][0]、B[2][0]	1

total 2+99=101 misses

Performance: 300 * 7 + 251 *100 = 27200

Example cont.:
Compiler-controlled prefetch

```
For (j=0; j<100; j=j+1){                    9 cycles / iteration
        Prefetch(b[j+7][0]);
        prefetch(a[0][j+7]);
        a[0][j]=b[j][0]*b [j+1][0];};          7 misses for b
                                                     4 misses for a
For (I=1; I<3; I=I+1)                        8 cycles / iteration
For (j=0; j<100; j=j+1){
        prefetch(a[i][j+7]);
        a[i][j]=b[j][0]*b [j+1][0];};4 misses for a[1][j]
                                    4 misses for a[2][j]
```

Total: 19 misses

save 232 cache misses at the price of 400 prefetch instructions.

Performance: 100 * 9 + 200* 8 + 19 *100 = 4400 cycles

Use HBM to Extend Hierarchy

- HBM： High Bandwidth Memory
- Build massive L4 cache (128 MiB to 1GiB) using DRAMs
- Smaller blocks require substantial huge tag storage

block size: 64B/4KB; Block number:$2^{24}/2^{18}$; 16M/256K

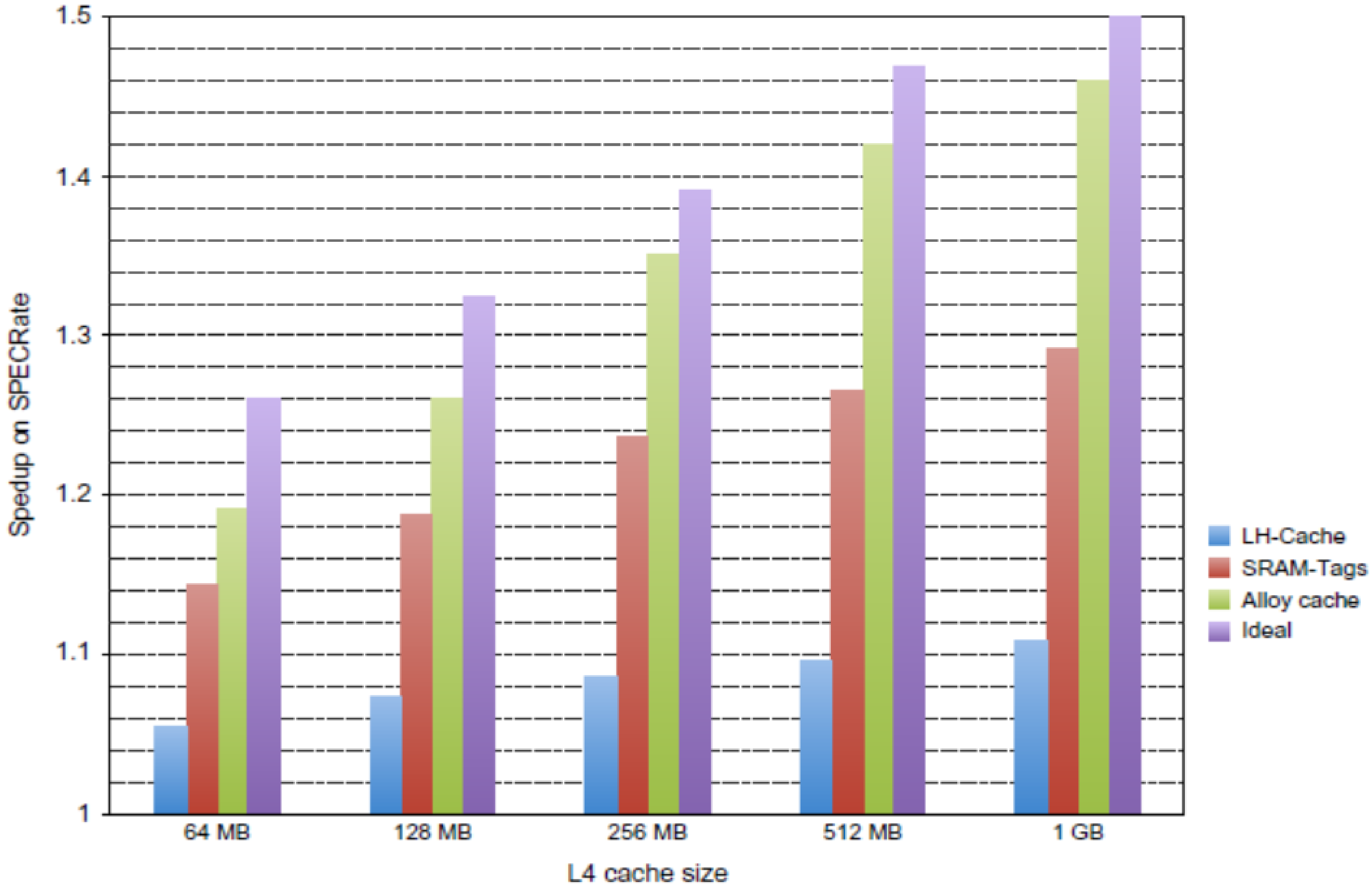
- Problems for Large block size
  - unneed contents, inefficient transferring
  - More conflict and consistency misses
- One approach (L-H):
  - Each SDRAM row is a block index
  - Each row contains set of tags and 29 data segments
  - 29-set associative
  - Hit requires a CAS

Use HBM to Extend Hierarchy

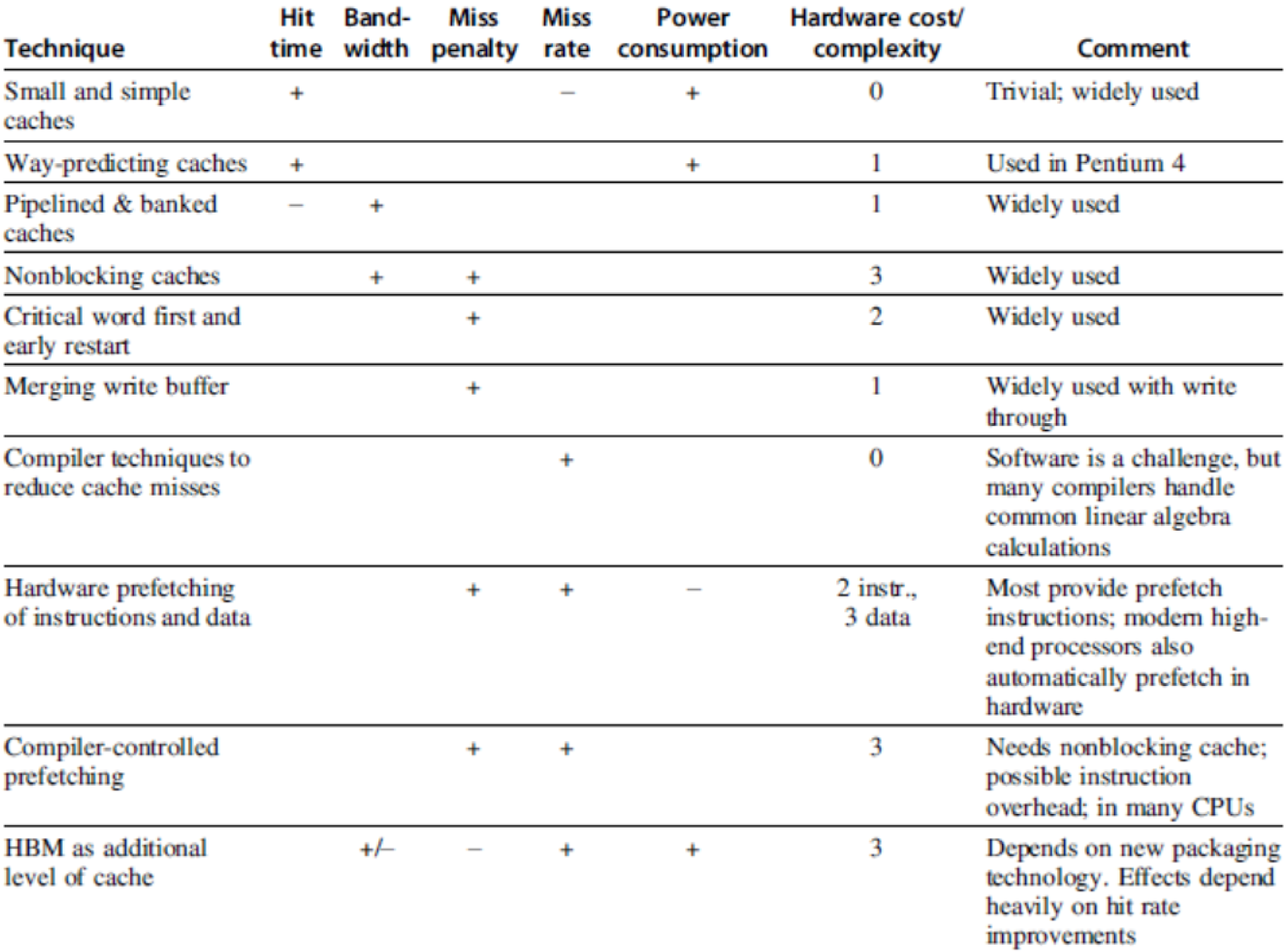
- Another approach (Alloy cache):
  - Mold tag and data together
  - Use direct mapped

- Both schemes require two DRAM accesses for misses
  - Two solutions:
    - Use map to keep track of blocks
    - Predict likely misses

Use HBM to Extend Hierarchy



Summary

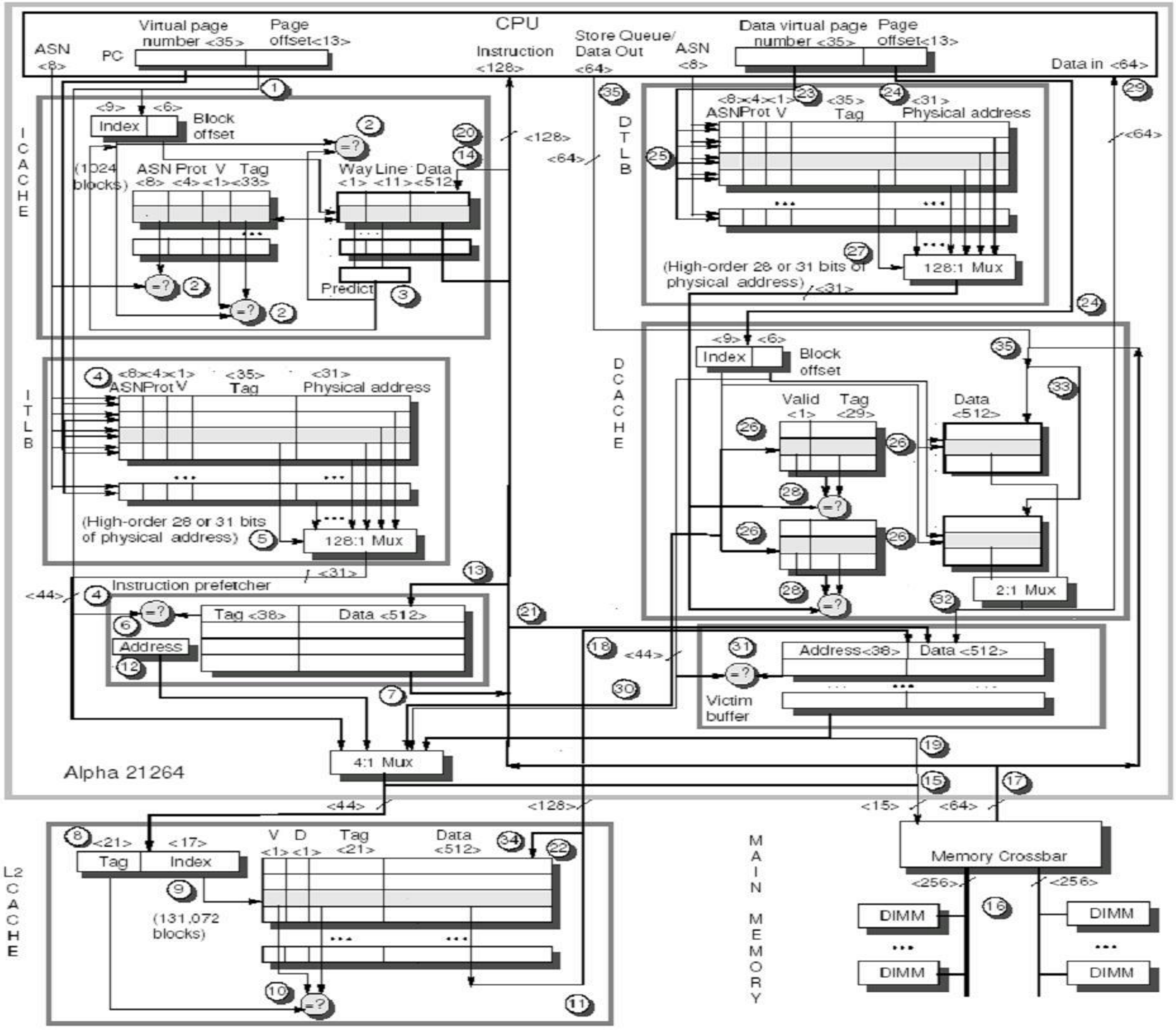


Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined & banked caches	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Critical 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

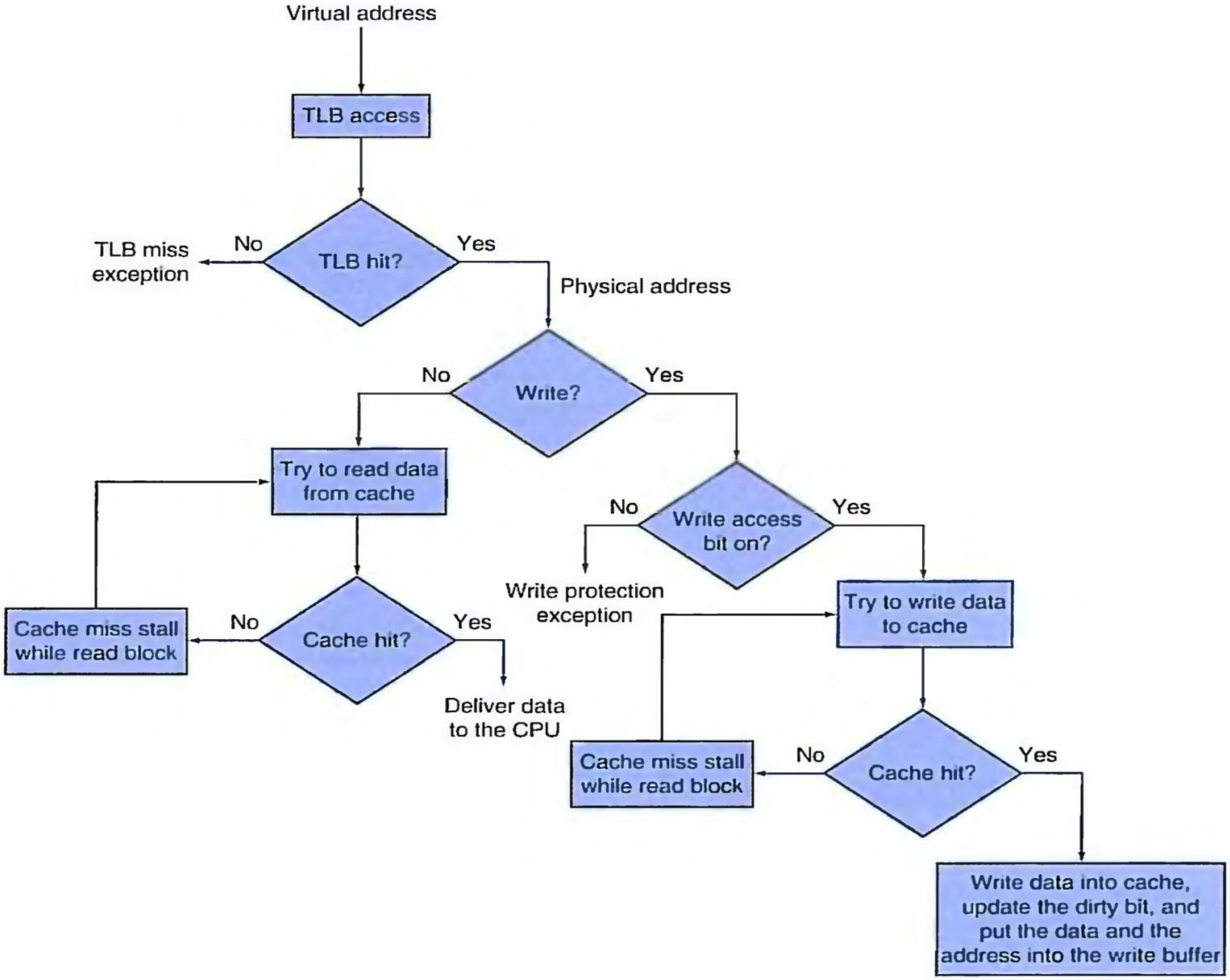
Example: Alpha 21264 Memory Hierarchy

- **The 21264 is an out-of-order execution processor**
  - fetches up to 4 instructions per clock cycle and executes up to 6 instructions per clock cycle.
- **virtual address**
  - 48-bit virtual address and a 44-bit physical address

    / 43-bit virtual address and 41-bit physical;
  - In either case, Alpha halves the physical address space, with the lower half for memory addresses and the upper half for I/O addresses.
- **when the Alpha is turned on.**
  - Hardware on the chip loads the instruction cache serially from an external PROM. (16K instructions)
  - The same serial interface (and PROM) also loads configuration information that specifies L2 cache speed/timing, system port speed/timing, and much other information necessary


CPU
ASN <8>
PC
Virtual page number <35>
Page offset<13>
Instruction <128>
Store Queue/ Data Out <64>
ASN <8>
Data virtual page number <35>
Page offset<13>
Data in <64>
1
I CACHE
<9>
<6>
Index
Block offset
2
20
14
<128>
(1024 blocks)
ASN Prot V Tag
<8> <4> <1> <33>
Way Line Data
<1> <11> <512>
=?
2
=?
2
Predict
3
I TLB
4
<8><4><1>
ASNProt V
<35>
Tag
<31>
Physical address
(High-order 28 or 31 bits of physical address)
5
128:1 Mux
<31>
<44>
4
Instruction prefetcher
13
=?
6
Tag <38>
Data <512>
Address
12
7
4:1 Mux
Alpha 21264
<64>
35
D TLB
23
24
<8><4><1>
ASNProt V
<35>
Tag
<31>
Physical address
25
27
(High-order 28 or 31 bits of physical address)
128:1 Mux
<31>
24
29
<64>
D CACHE
<9>
<6>
Index
Block offset
35
33
Valid <1>
Tag <29>
Data <512>
26
26
28
=?
26
26
28
=?
2:1 Mux
32
21
18
<44>
31
=?
Address<38>
Data <512>
30
Victim buffer
19
15
17
<44>
<128>
<15>
<64>
L2 CACHE
8
<21>
<17>
Tag
Index
V D Tag Data
<1> <1> <21> <512>
34
22
9
(131,072 blocks)
10
=?
11
MAIN MEMORY
Memory Crossbar
<256>
<256>
16
DIMM
DIMM
DIMM
DIMM

Important concept： Cache


Virtual address
TLB access
TLB hit?
No
TLB miss exception
Yes
Physical address
Write?
No
Yes
Try to read data from cache
Write access bit on?
No
Write protection exception
Yes
Try to write data to cache
Cache hit?
No
Cache miss stall while read block
Yes
Deliver data to the CPU
Cache hit?
No
Cache miss stall while read block
Yes
Write data into cache, update the dirty bit, and put the data and the address into the write buffer

Virtual Memory and Virtual Machines

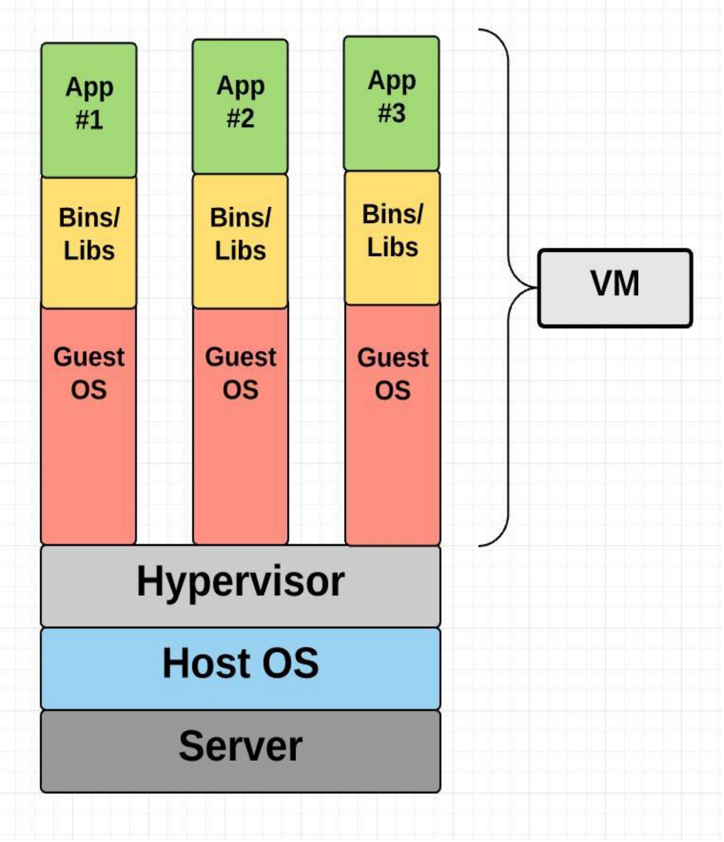
- ❑ Protection via virtual memory
  - ➢ Keeps processes in their own memory space

- ❑ 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

Virtual machine

- An efficient, isolated duplicate of the real machine.

- VMM （virtual machine monitor）
  - Provide an environment for programs，which is essentially identical with the original machine.
  - Programs run in this environment show at worst only minor decrease in speed.
  - VMM is in complete control of system resources.



Protection via 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" (at binary instruction set architecture)
  - SVM software is called "virtual machine monitor" or "hypervisor"
  - Individual virtual machines run under the monitor are called "guest VMs"

Requirements of VMM

- Guest software should:
  - Behave on as if running on native hardware
  - Not be able to change allocation of real system resources

- VMM should be able to "context switch" guests

- Hardware must allow:
  - System and use processor modes
  - Privileged subset of instructions for allocating system resources

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

Extending the ISA for Virtualization

- Objectives:
  - Avoid flushing TLB
  - Use nested page tables instead of shadow page tables
  - Allow devices to use DMA to move data
  - Allow guest OS's to handle device interrupts
  - For security: allow programs to manage encrypted portions of code and data

Fallacies and Pitfalls

- Predicting cache performance of one program from another
- Simulating enough instructions to get accurate performance measures of the memory hierarchy
- Not deliverying high memory bandwidth in a cache-based system

End.