CS252
Graduate Computer Architecture
Lecture 4

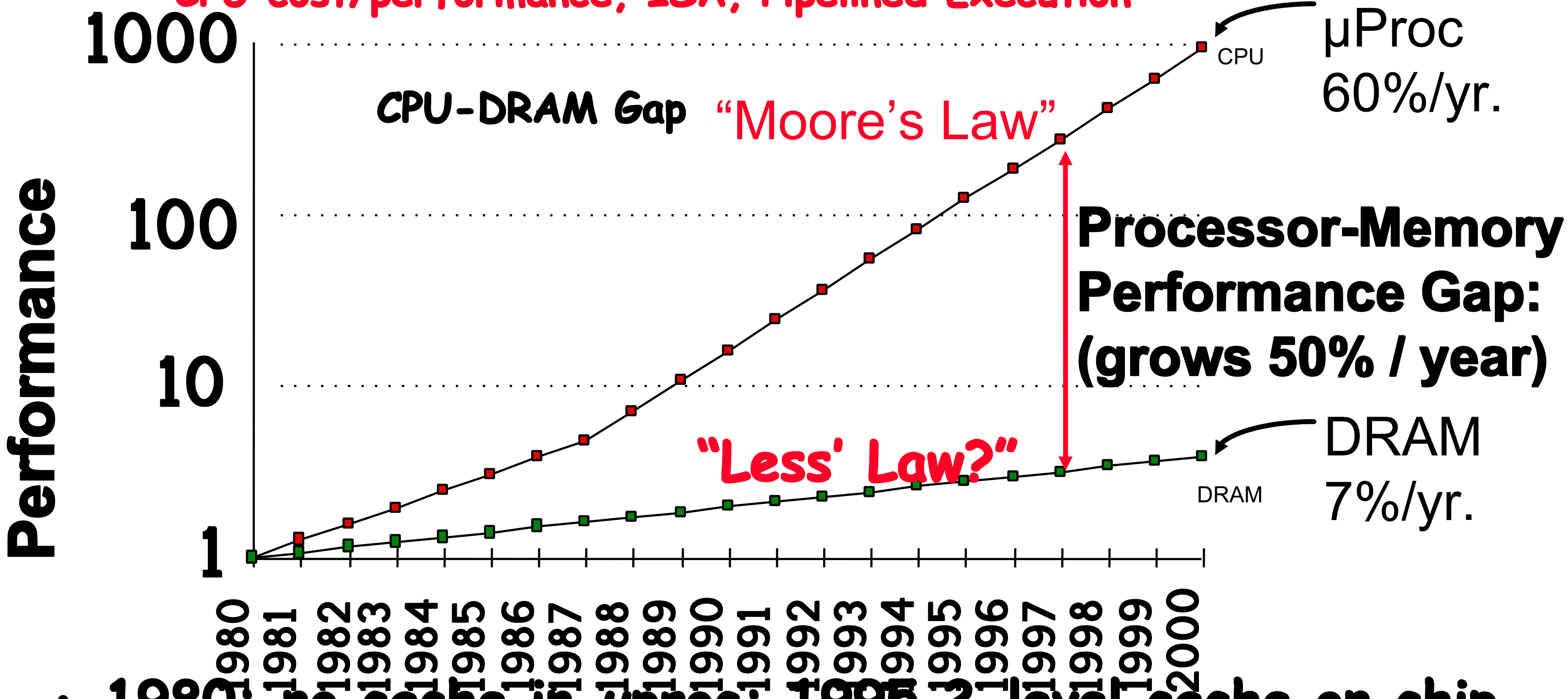
Caches and Memory Systems

January 26, 2001

Prof. John Kubiatowicz

Review: Who Cares About the Memory Hierarchy?

- **Processor Only Thus Far in Course:**
  - CPU cost/performance, ISA, Pipelined Execution



- **1980: no cache in µproc; 1995 2-level cache on chip (1989 first Intel µproc with a cache on chip)**

Review: Cache performance

- **Miss-oriented Approach to Memory Access:**

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemAccess}{Inst} \times MissRate \times MissPenalty \right) \times CycleTime$$

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemMisses}{Inst} \times MissPenalty \right) \times CycleTime$$

  - $CPI_{Execution}$ includes ALU and Memory instructions

- **Separating out Memory component entirely**
  - AMAT = Average Memory Access Time
  - CPI_{ALUOps} does not include memory instructions

$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT \right) \times CycleTime$$

$$\begin{aligned} AMAT &= HitTime + MissRate \times MissPenalty \\ &= \left(HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst} \right) + \\ &\quad \left(HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data} \right) \end{aligned}$$

Review: Reducing Misses

- Classifying Misses: 3 Cs
  - Compulsory—Misses in even an Infinite Cache
  - Capacity—Misses in Fully Associative Size X Cache
  - Conflict—Misses in N-way Associative, Size X Cache
- More recent, 4th "C":
  - *Coherence* - Misses caused by cache coherence.

Review: Miss Rate Reduction

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

- 3 Cs: Compulsory, Capacity, Conflict
  1. Reduce Misses via Larger Block Size
  2. Reduce Misses via Higher Associativity
  3. Reducing Misses via Victim Cache
  4. Reducing Misses via Pseudo-Associativity
  5. Reducing Misses by HW Prefetching Instr, Data
  6. Reducing Misses by SW Prefetching Data
  7. Reducing Misses by Compiler Optimizations
- Prefetching comes in two flavors:
  - Binding prefetch: Requests load directly into register.
    - » Must be correct address and register!
  - Non-Binding prefetch: Load into cache.
    - » Can be incorrect. Frees HW/SW to guess!

Improving Cache Performance Continued

1. Reduce the miss rate,
2. *Reduce the miss penalty,* or
3. Reduce the time to hit in the cache.

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

What happens on a Cache miss?

- For in-order pipeline, 2 options:
  - Freeze pipeline in Mem stage (popular early on: Sparc, R4000)

```
IF   ID   EX   Mem stall stall stall … stall Mem   Wr
     IF   ID   EX  stall stall stall … stall stall Ex Wr
```

  - Use Full/Empty bits in registers + MSHR queue
    » MSHR = "Miss Status/Handler Registers" (Kroft)
    Each entry in this queue keeps track of status of outstanding memory requests to one complete memory line.
      - Per cache-line: keep info about memory address.
      - For each word: register (if any) that is waiting for result.
      - Used to "merge" multiple requests to one memory line
    » New load creates MSHR entry and sets destination register to "Empty". Load is "released" from pipeline.
    » Attempt to use register before result returns causes instruction to block in decode stage.
    » Limited "out-of-order" execution with respect to loads.
    Popular with in-order superscalar architectures.
- Out-of-order pipelines already have this functionality built in… (load queues, etc).

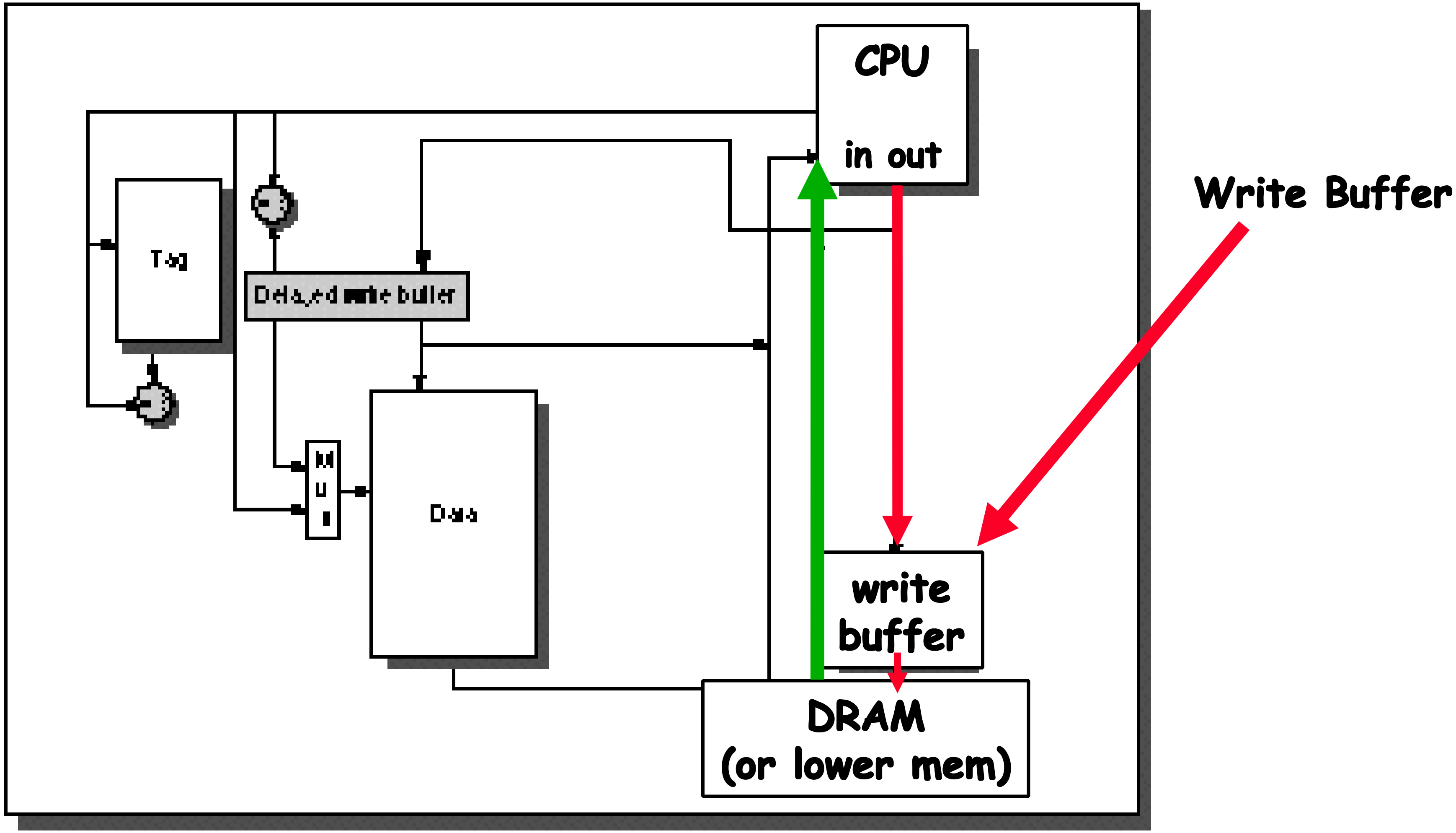
Write Policy: Write-Through vs Write-Back

- **Write-through: all writes update cache and underlying memory/cache**
  - Can always discard cached data - most up-to-date data is in memory
  - Cache control bit: only a *valid* bit
- **Write-back: all writes simply update cache**
  - Can't just discard cached data - may have to write it back to memory
  - Cache control bits: both *valid* and *dirty* bits
- **Other Advantages:**
  - Write-through:
    - » memory (or other processors) always have latest data
    - » Simpler management of cache
  - Write-back:
    - » much lower bandwidth, since data often overwritten multiple times
    - » Better tolerance to long-latency memory?

Write Policy 2: Write Allocate vs Non-Allocate (What happens on write-miss)

- Write allocate: allocate new cache line in cache
  - Usually means that you have to do a "read miss" to fill in rest of the cache-line!
  - Alternative: per/word valid bits
- Write non-allocate (or "write-around"):
  - Simply send write data through to underlying memory/cache - don't allocate new cache line!

1. Reducing Miss Penalty: Read Priority over Write on Miss

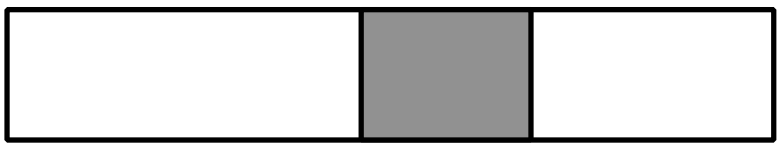


1. Reducing Miss Penalty: Read Priority over Write on Miss

- **Write-through with write buffers offer 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
- **Write-back also want buffer to hold misplaced 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

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

- **Don't wait for full block to be loaded before restarting CPU**
  - ***Early restart***—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - ***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*
- **Generally useful only in large blocks,**
- **Spatial locality a problem; tend to want next sequential word, so not clear if benefit by early restart**

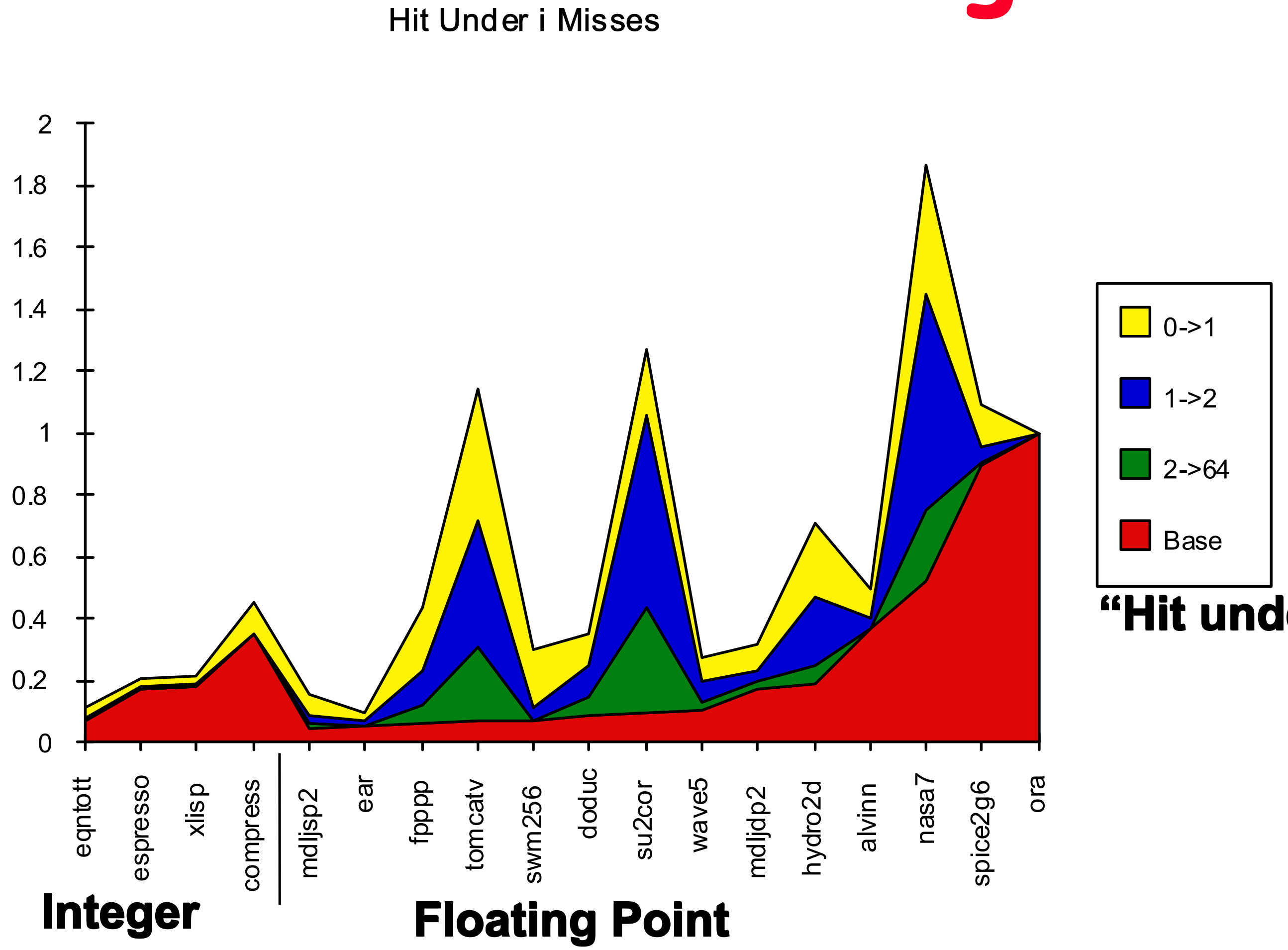


block

3. Reduce Miss Penalty: Non-blocking Caches to reduce stalls on misses

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

Value of Hit Under Miss for SPEC (Normalized to blocking cache)



0->1
1->2
2->64
Base
"Hit under n Misses"

- FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26
- Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss

4. Second level cache

- **L2 Equations**

  $AMAT = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}$

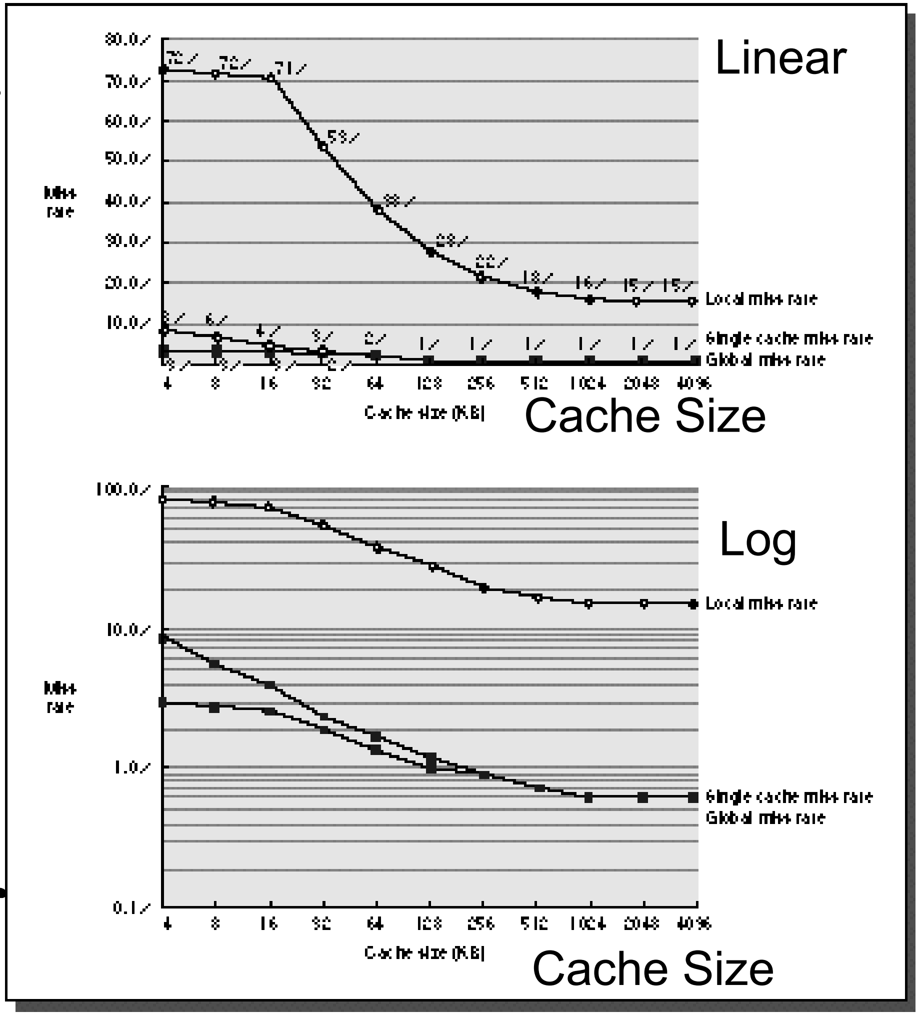
  $\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$

  $AMAT = \text{Hit Time}_{L1} +$
  $\underline{\text{Miss Rate}_{L1}} \times (\text{Hit Time}_{L2} + \underline{\text{Miss Rate}_{L2}} + \text{Miss Penalty}_{L2})$

- **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* ($\text{Miss Rate}_{L1} \times \text{Miss Rate}_{L2}$)
  - Global Miss Rate is what matters

Comparing Local and Global Miss Rates

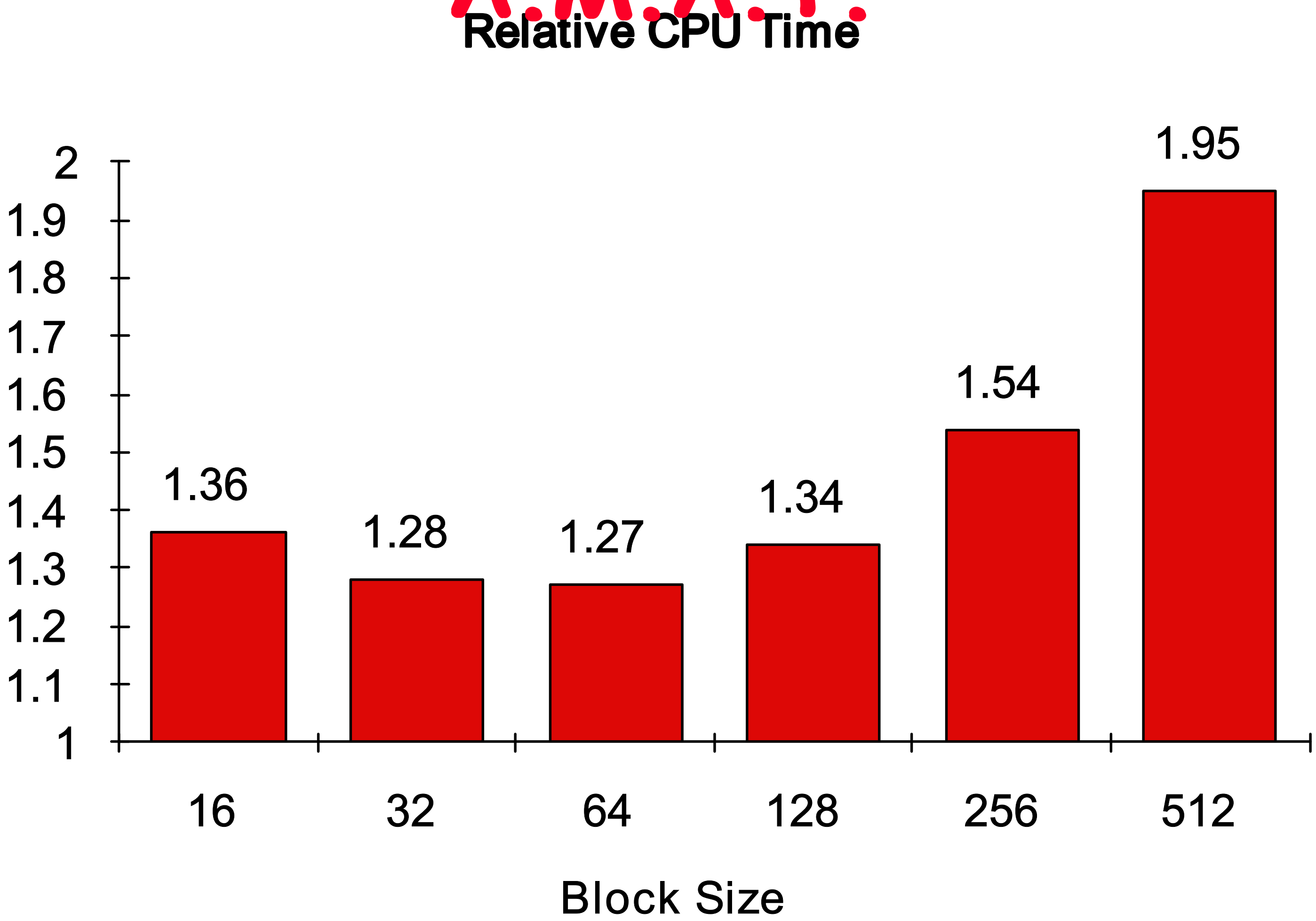
- 32 KByte 1st level cache; Increasing 2nd level cache
- Global miss rate close to single level cache rate provided L2 >> L1
- Don't use local miss rate
- L2 not tied to CPU clock cycle!
- Cost & A.M.A.T.
- Generally Fast Hit Times and fewer misses
- Since hits are few, target miss reduction



Reducing Misses: Which apply to L2 Cache?

- **Reducing Miss Rate**
  1. Reduce Misses via Larger Block Size
  2. Reduce Conflict Misses via Higher Associativity
  3. Reducing Conflict Misses via Victim Cache
  4. Reducing Conflict Misses via Pseudo-Associativity
  5. Reducing Misses by HW Prefetching Instr, Data
  6. Reducing Misses by SW Prefetching Data
  7. Reducing Capacity/Conf. Misses by Compiler Optimizations

L2 cache block size & A.M.A.T.



- 32KB L1, 8 byte path to memory

Reducing Miss Penalty Summary

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

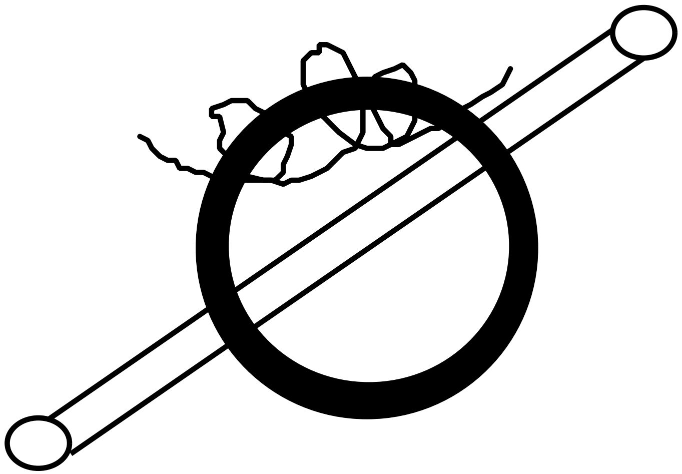
- Four techniques
  - Read priority over write on miss
  - Early Restart and Critical Word First on miss
  - Non-blocking Caches (Hit under Miss, Miss under Miss)
  - Second Level Cache
- Can be applied recursively to Multilevel Caches
  - Danger is that time to DRAM will grow with multiple levels in between
  - First attempts at L2 caches can make things worse, since increased worst case is worse

Main Memory Background

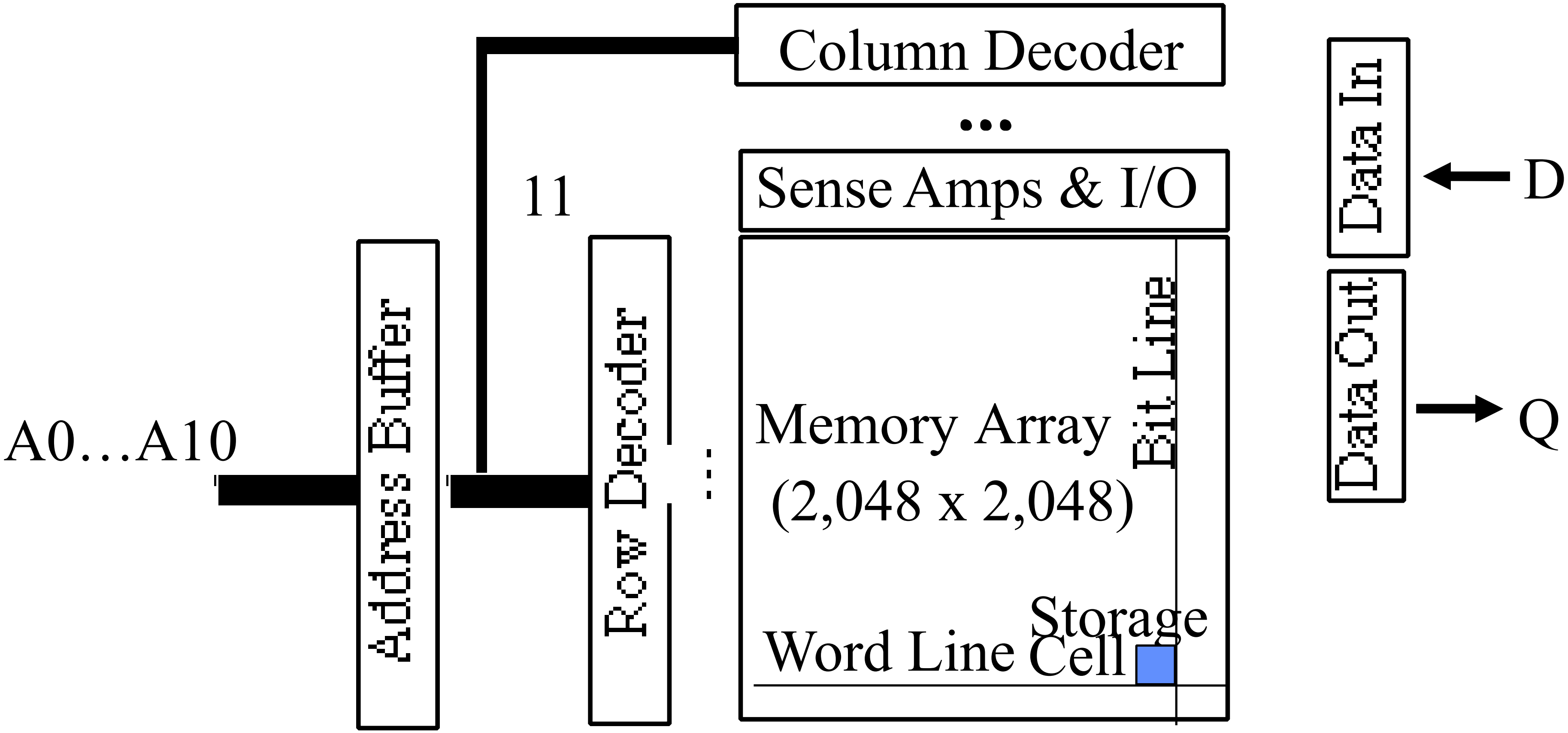
- **Performance of Main Memory:**
  - Latency: Cache Miss Penalty
    - » *Access Time*: time between request and word arrives
    - » *Cycle Time*: time between requests
  - Bandwidth: I/O & Large Block Miss Penalty (L2)
- **Main Memory is *DRAM*: Dynamic Random Access Memory**
  - Dynamic since needs to be refreshed periodically (8 ms, 1% time)
  - Addresses divided into 2 halves (Memory as a 2D matrix):
    - » *RAS* or *Row Access Strobe*
    - » *CAS* or *Column Access Strobe*
- **Cache uses *SRAM*: Static Random Access Memory**
  - No refresh (6 transistors/bit vs. 1 transistor
    Size: DRAM/SRAM - *4-8*,
    Cost/Cycle time: SRAM/DRAM - *8-16*

Main Memory Deep Background

- "Out-of-Core", "In-Core," "Core Dump"?
- "Core memory"?
- Non-volatile, magnetic
- Lost to 4 Kbit DRAM (today using 64Kbit DRAM)
- Access time 750 ns, cycle time 1500-3000 ns



DRAM logical organization (4 Mbit)



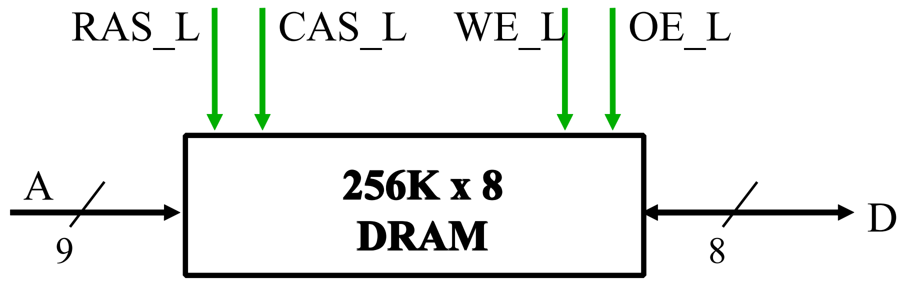
- Square root of bits per RAS/CAS

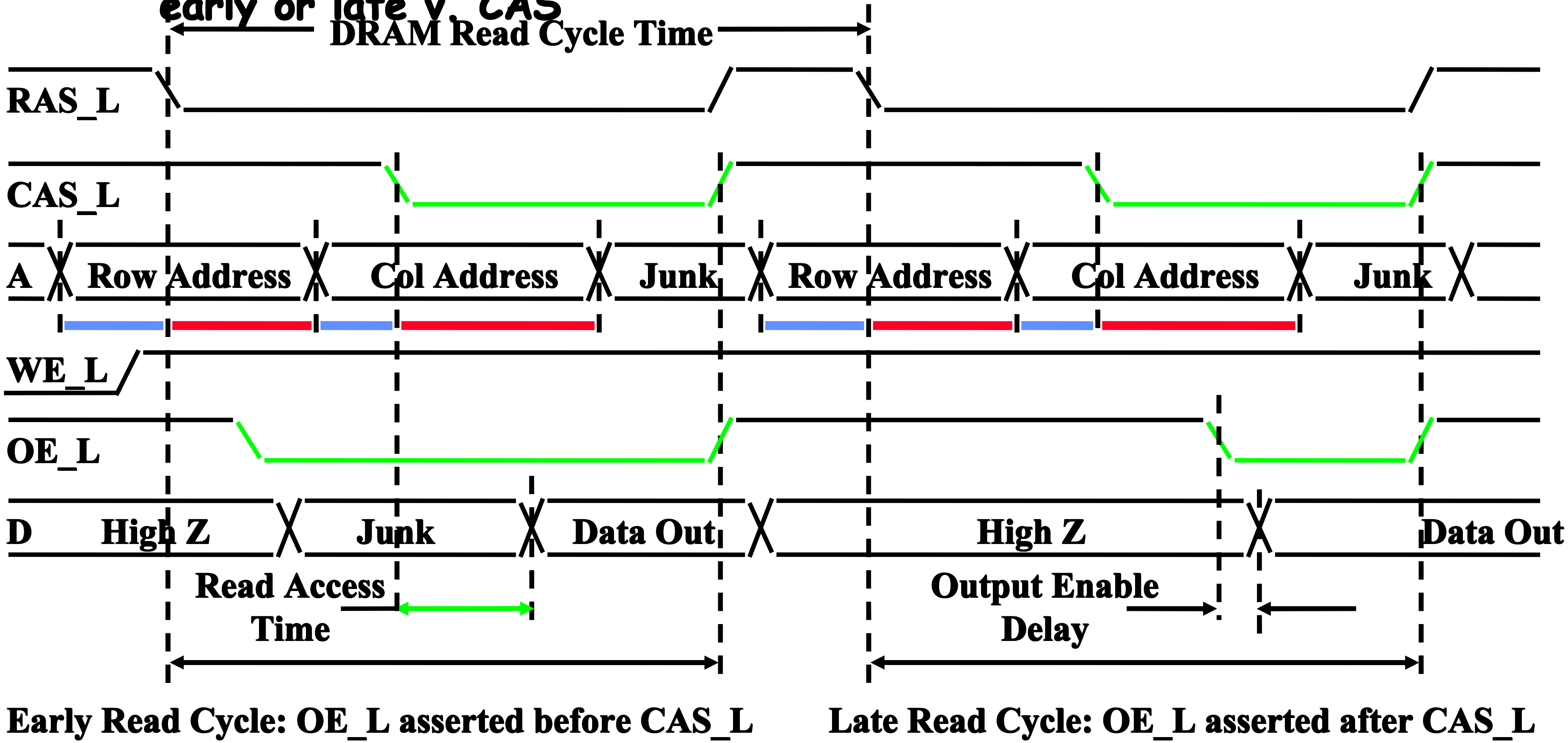
4 Key DRAM Timing Parameters

- t_{RAC}: minimum time from RAS line falling to the valid data output.
  - Quoted as the speed of a DRAM when buy
  - A typical 4Mb DRAM t_{RAC} = 60 ns
  - Speed of DRAM since on purchase sheet?
- t_{RC}: minimum time from the start of one row access to the start of the next.
  - t_{RC} = 110 ns for a 4Mbit DRAM with a t_{RAC} of 60 ns
- t_{CAC}: minimum time from CAS line falling to valid data output.
  - 15 ns for a 4Mbit DRAM with a t_{RAC} of 60 ns
- t_{PC}: minimum time from the start of one column access to the start of the next.
  - 35 ns for a 4Mbit DRAM with a t_{RAC} of 60 ns

DRAM Read Timing

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





DRAM Performance

- A 60 ns (t_{RAC}) DRAM can
  - perform a row access only every 110 ns (t_{RC})
  - perform column access (t_{CAC}) in 15 ns, but time between column accesses is at least 35 ns (t_{PC}).
    - » In practice, external address delays and turning around buses make it 40 to 50 ns
- These times do not include the time to drive the addresses off the microprocessor nor the memory controller overhead!

DRAM History

- DRAMs: capacity +60%/yr, cost –30%/yr
  - 2.5X cells/area, 1.5X die size in -3 years
- '98 DRAM fab line costs $2B
  - DRAM only: density, leakage v. speed
- Rely on increasing no. of computers & memory per computer (60% market)
  - SIMM or DIMM is replaceable unit
    => computers use any generation DRAM
- Commodity, second source industry
  => high volume, low profit, conservative
  - Little organization innovation in 20 years
- Order of importance: 1) Cost/bit 2) Capacity
  - First RAMBUS: 10X BW, +30% cost => little impact

DRAM Future: 1 Gbit DRAM (ISSCC '96; production '02?)

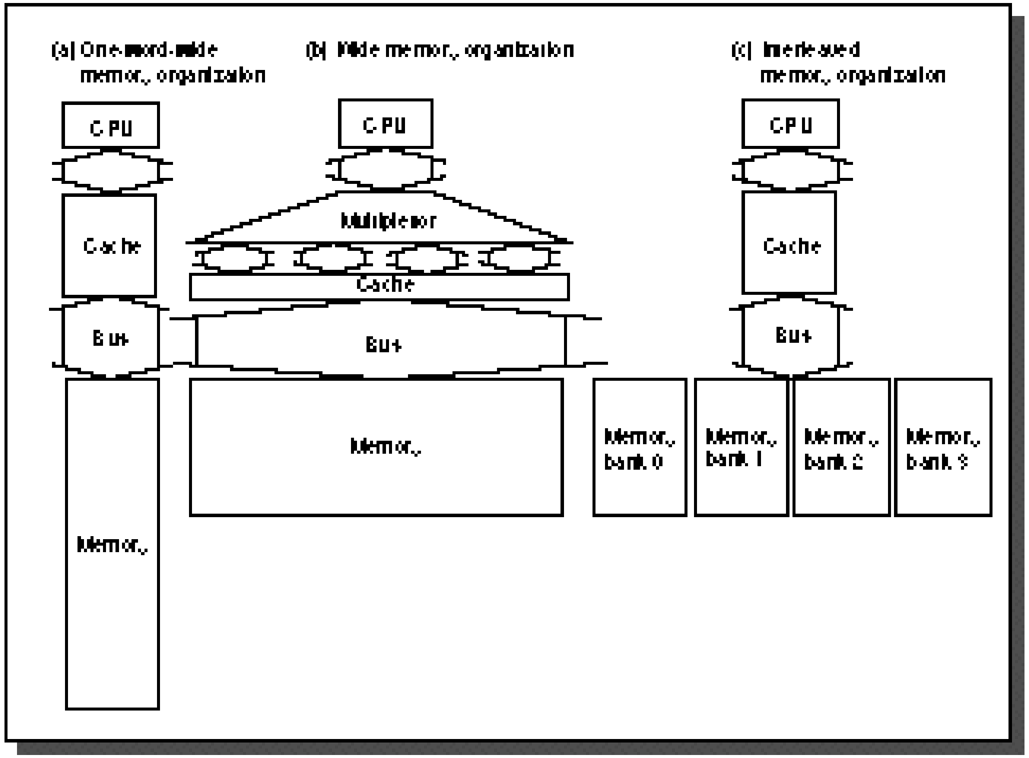
	Mitsubishi	Samsung
• Blocks	512 x 2 Mbit	1024 x 1 Mbit
• Clock	200 MHz	250 MHz
• Data Pins	64	16
• Die Size	24 x 24 mm	31 x 21 mm
– Sizes will be much smaller in production		
• Metal Layers	3	4
• Technology	0.15 micron	0.16 micron

Fast Memory Systems: DRAM specific

- Multiple CAS accesses: several names (page mode)
  - *Extended Data Out (EDO)*: 30% faster in page mode
- New DRAMs to address gap; what will they cost, will they survive?
  - *RAMBUS*: startup company; reinvent DRAM interface
    - » Each Chip a module vs. slice of memory
    - » Short bus between CPU and chips
    - » Does own refresh
    - » Variable amount of data returned
    - » 1 byte / 2 ns (500 MB/s per chip)
    - » 20% increase in DRAM area
  - *Synchronous DRAM*: 2 banks on chip, a clock signal to DRAM, transfer synchronous to system clock (66 - 150 MHz)
  - Intel claims RAMBUS Direct (16 b wide) is future PC memory?
    - » Possibly not true! Intel to drop RAMBUS?
- Niche memory or main memory?
  - e.g., Video RAM for frame buffers, DRAM + fast serial output

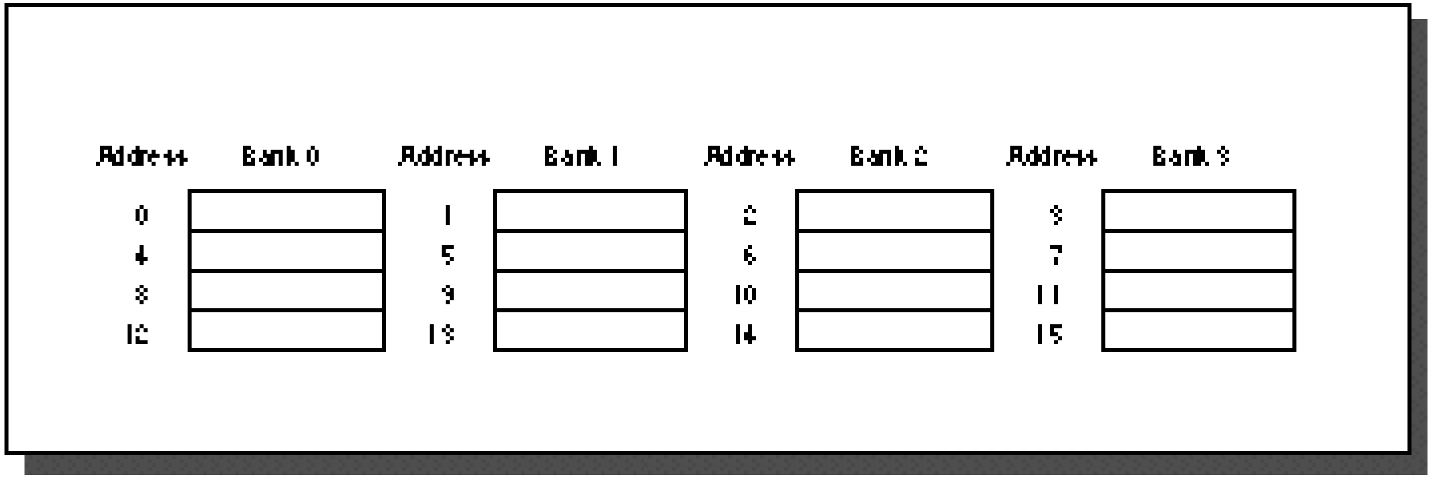
Main Memory Organizations

- ***Simple***:
  - CPU, Cache, Bus, Memory same width (32 or 64 bits)
- ***Wide***:
  - CPU/Mux 1 word; Mux/Cache, Bus, Memory N words (Alpha: 64 bits & 256 bits; UtraSPARC 512)
- ***Interleaved***:
  - CPU, Cache, Bus 1 word: Memory N Modules (4 Modules); example is *word interleaved*



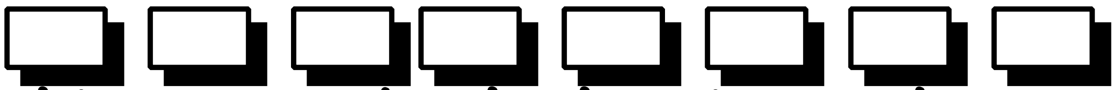
Main Memory Performance

- Timing model (word size is 32 bits)
  - 1 to send address,
  - 6 access time, 1 to send data
  - Cache Block is 4 words
- *Simple M.P.* = 4 x (1+6+1) = 32
- *Wide M.P.* = 1 + 6 + 1 = 8
- *Interleaved M.P.* = 1 + 6 + 4x1 = 11

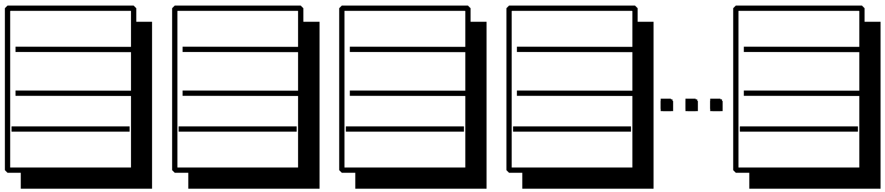


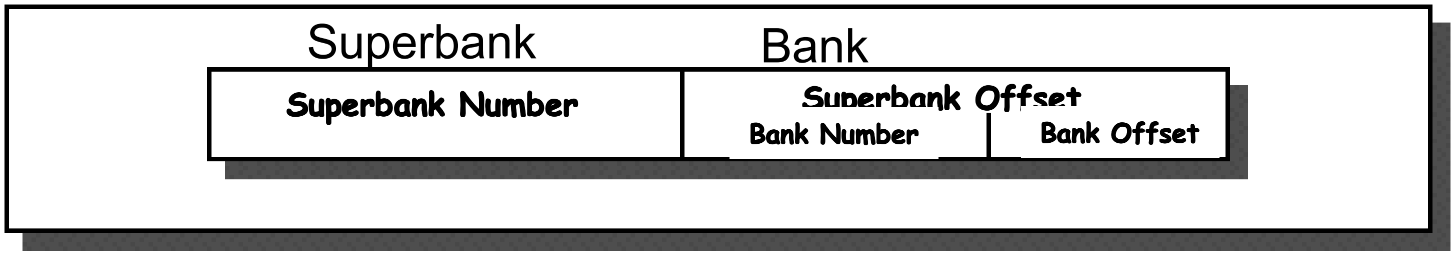
Independent Memory Banks

- Memory banks for independent accesses vs. faster sequential accesses
  - Multiprocessor
  - I/O
  - CPU with Hit under n Misses, Non-blocking Cache
- *Superbank*: all memory active on one block transfer (or *Bank*)



- *Bank*: portion within a superbank that is word interleaved (or *Subbank*)





Independent Memory Banks

- How many banks?

  number banks $\leq$ number clocks to access word in bank

  - For sequential accesses, otherwise will return to original bank before it has next word ready
  - (like in vector case)

- Increasing DRAM => fewer chips => harder to have banks

Avoiding Bank Conflicts

- Lots of banks

```
int x[256][512];
    for (j = 0; j < 512; j = j+1)
        for (i = 0; i < 256; i = i+1)
            x[i][j] = 2 * x[i][j];
```

- Even with 128 banks, since 512 is multiple of 128, conflict on word accesses
- SW: loop interchange or declaring array not power of 2 (“array padding”)
- HW: Prime number of banks
  - bank number = address mod number of banks
  - address within bank = address / number of words in bank
  - modulo & divide per memory access with prime no. banks?
  - address within bank = address mod number words in bank
  - bank number? easy if 2^N words per bank

Fast Bank Number

- **Chinese Remainder Theorem**

  As long as two sets of integers ai and bi follow these rules

  $$b_i = x \bmod a_i, 0 \le b_i < a_i, 0 \le x < a_0 \times a_1 \times a_2 \times \ldots$$

  and that ai and aj are co-prime if i ≠ j, then the integer x has only one solution (unambiguous mapping):

  - bank number = b_0, number of banks = a_0 (= 3 in example)
  - address within bank = b_1, number of words in bank = a_1 (= 8 in example)
  - N word address 0 to N-1, prime no. banks, words power of 2

	Seq. Interleaved			Modulo Interleaved		
Bank Number:	0	1	2	0	1	2
Address within Bank: *0*	0	1	2	0	16	8
1	3	4	5	9	1	17
2	6	7	8	18	10	2
3	9	10	11	3	19	11
4	12	13	14	12	4	20
5	15	16	17	21	13	5
6	18	19	20	6	22	14
7	21	22	23	15	7	23

DRAMs per PC over Time

Minimum Memory Size	DRAM Generation: '86 1 Mb	'89 4 Mb	'92 16 Mb	'96 64 Mb	'99 256 Mb	'02 1 Gb
4 MB	32 →	8				
8 MB		16 →	4			
16 MB			8 →	2		
32 MB				4 →	1	
64 MB				8 →	2	
128 MB					4 →	1
256 MB					8 →	2

Need for Error Correction!

- **Motivation:**
  - Failures/time *proportional* to number of bits!
  - As DRAM cells shrink, more vulnerable
- **Went through period in which failure rate was low enough without error correction that people didn't do correction**
  - DRAM banks too large now
  - Servers always corrected memory systems
- **Basic idea: add redundancy through parity bits**
  - Simple but wastful version:
    - » Keep three copies of everything, vote to find right value
    - » 200% overhead, so not good!
  - Common configuration: Random error correction
    - » SEC-DED (single error correct, double error detect)
    - » One example: 64 data bits + 8 parity bits (11% overhead)
    - » Papers up on reading list from last term tell you how to do these types of codes
  - Really want to handle failures of physical components as well
    - » Organization is multiple DRAMs/SIMM, multiple SIMMs
    - » Want to recover from failed DRAM and failed SIMM!
    - » Requires more redundancy to do this
    - » All major vendors thinking about this in high-end machines

Architecture in practice

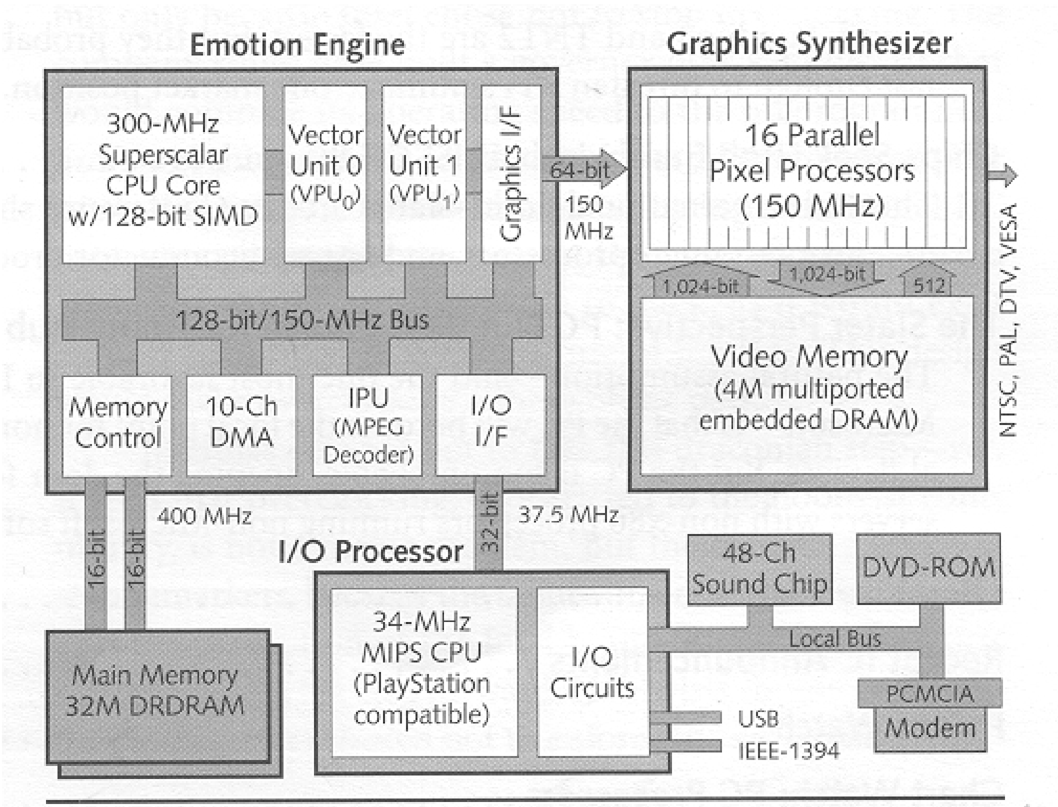


Figure 1. PlayStation 2000 employs an unprecedented level of parallelism to achieve workstation-class 3D performance.

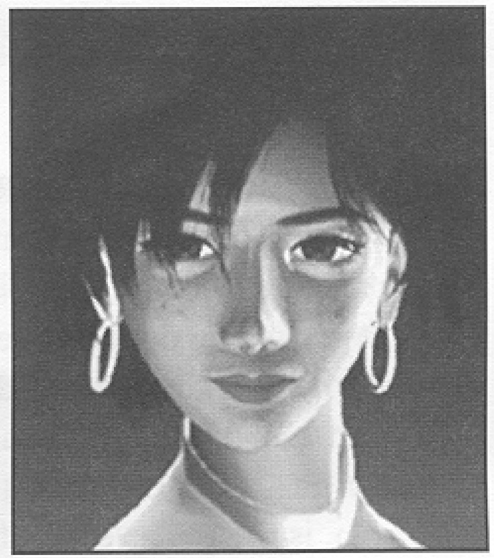


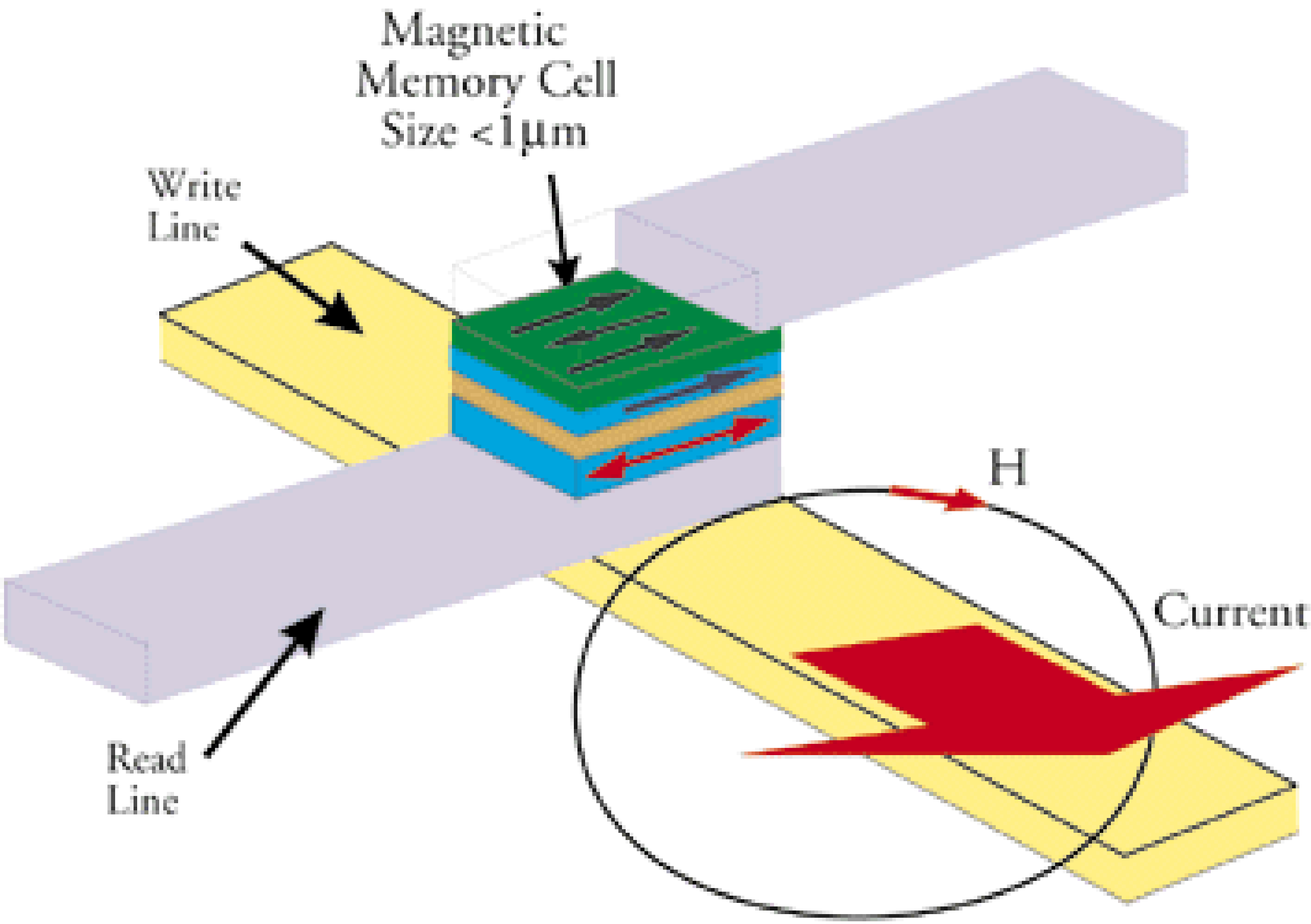
Figure 2. PlayStation 2000 screenshot. (Source: Namco)

- **(as reported in Microprocessor Report, Vol 13, No. 5)**
  - **Emotion Engine: 6.2 GFLOPS, 75 million polygons per second**
  - **Graphics Synthesizer: 2.4 Billion pixels per second**
  - **Claim: *Toy Story* realism brought to games!**

More esoteric Storage Technologies?

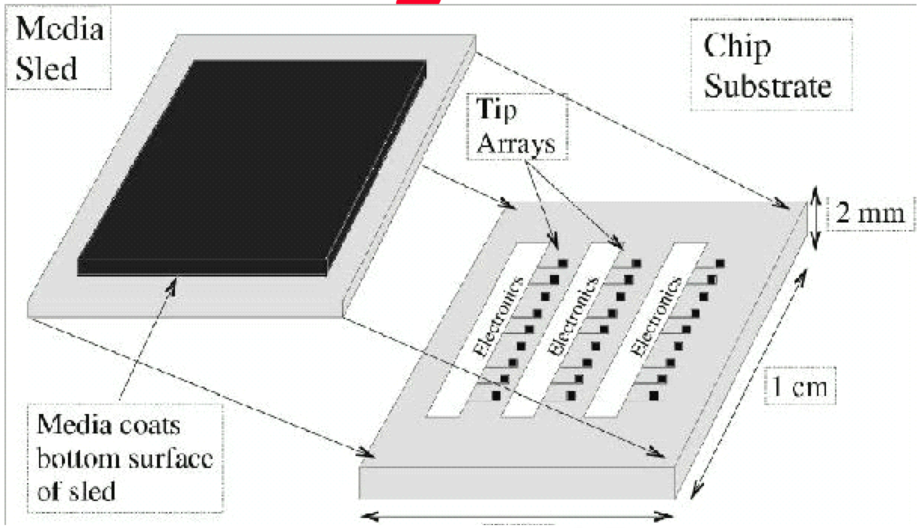
- Tunneling Magnetic Junction RAM (TMJ-RAM):
  - Speed of SRAM, density of DRAM, non-volatile (no refresh)
  - New field called "Spintronics": combination of quantum spin and electronics
  - Same technology used in high-density disk-drives
- MEMs storage devices:
  - Large magnetic "sled" floating on top of lots of little read/write heads
  - Micromechanical actuators move the sled back and forth over the heads

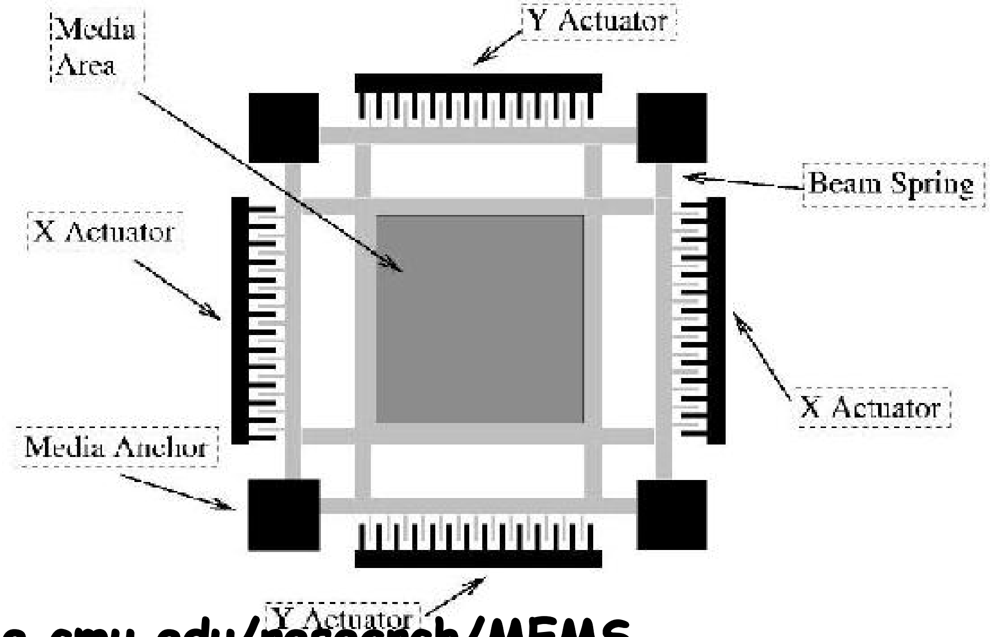
Tunneling Magnetic Junction



MEMS-based Storage

- Magnetic "sled" floats on array of read/write heads
  - Approx 250 Gbit/in^2
  - Data rates:
    IBM: 250 MB/s w 1000 heads
    CMU: 3.1 MB/s w 400 heads
- Electrostatic actuators move media around to align it with heads
  - Sweep sled $\pm 50\mu m$ in $< 0.5\mu s$
- Capacity estimated to be in the 1-10GB in 10cm^2





See Ganger et all: http://www.lcs.ece.cmu.edu/research/MEMS

Main Memory Summary

- Wider Memory
- Interleaved Memory: for sequential or independent accesses
- Avoiding bank conflicts: SW & HW
- DRAM specific optimizations: page mode & Specialty DRAM
- Need Error correction

Review: Improving Cache Performance

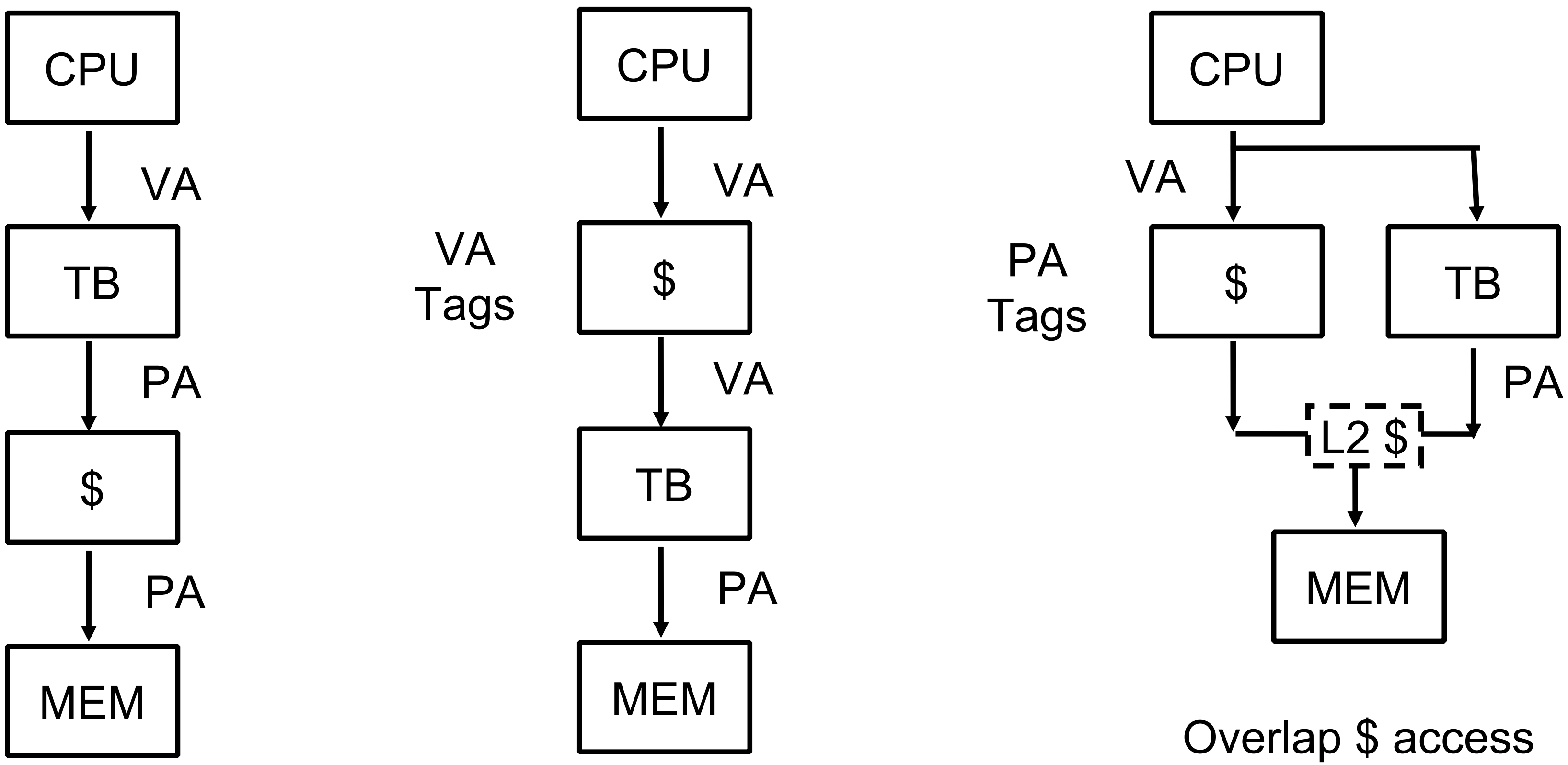
1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. *Reduce the time to hit in the cache*.

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

1. Fast Hit times via Small and Simple Caches

- Why Alpha 21164 has 8KB Instruction and 8KB data cache + 96KB second level cache?
  - Small data cache and clock rate
- Direct Mapped, on chip

2. Fast hits by Avoiding Address Translation



Conventional
Organization

Virtually Addressed Cache
Translate only on miss
Synonym Problem

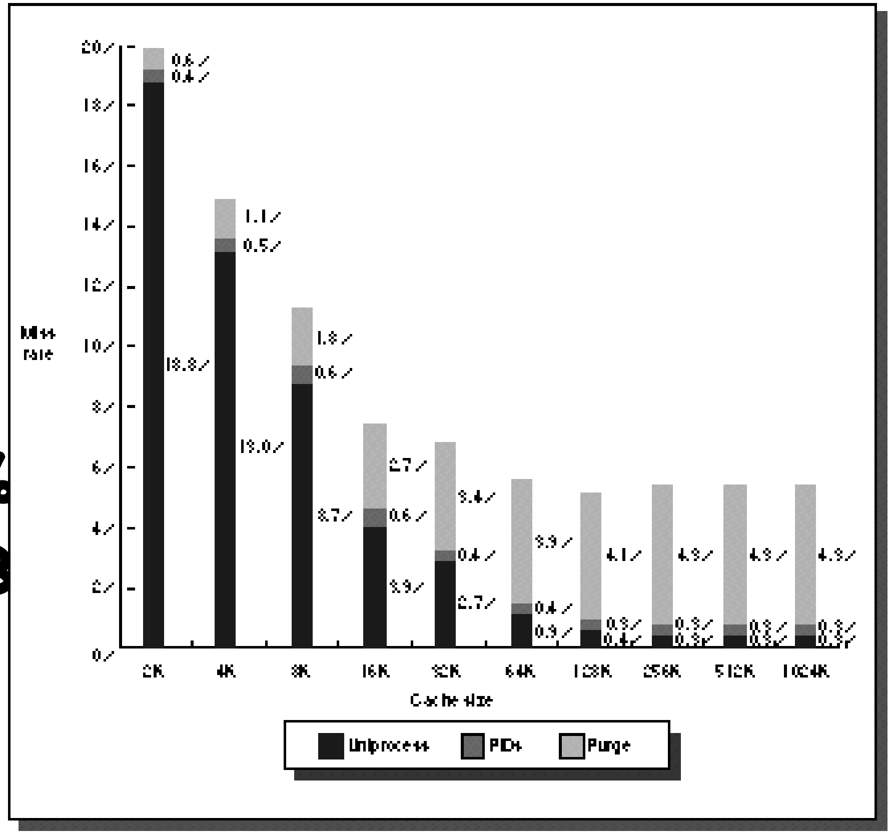
Overlap $ access
with VA translation:
requires $ index to
remain invariant
across translation

2. Fast hits by Avoiding Address Translation

- Send virtual address to cache? Called *Virtually Addressed 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
  - Dealing with *aliases* (sometimes called *synonyms*);
    Two different virtual addresses map to same physical address
  - I/O must interact with cache, so need virtual address
- Solution to aliases
  - HW guaranteess covers index field & direct mapped, they must be unique;
    called *page coloring*
- Solution to cache flush
  - Add *process identifier tag* that identifies process as well as address within process: can't get a hit if wrong process

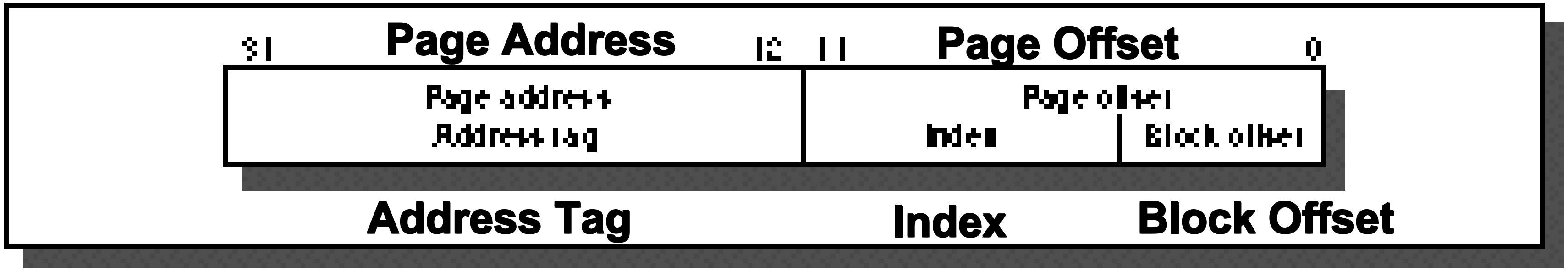
2. Fast Cache Hits by Avoiding Translation: Process ID impact

- Black is uniprocess
- Light Gray is multiprocess when flush cache
- Dark Gray is multiprocess when use Process ID tag
- Y axis: Miss Rates up to 20%
- X axis: Cache size from 2 KB to 1024 KB



2. Fast Cache Hits by Avoiding Translation: Index with Physical Portion of Address

- If index is physical part of address, can start tag access in parallel with translation so that can compare to physical tag



- Limits cache to page size: what if want bigger caches and uses same trick?
  - Higher associativity moves barrier to right
  - Page coloring

3: Fast Hits by pipelining Cache Case Study: MIPS R4000

- 8 Stage Pipeline:
  - IF–first half of fetching of instruction; PC selection happens here as well as initiation of instruction cache access.
  - IS–second half of access to instruction cache.
  - RF–instruction decode and register fetch, hazard checking and also instruction cache hit detection.
  - EX–execution, which includes effective address calculation, ALU operation, and branch target computation and condition evaluation.
  - DF–data fetch, first half of access to data cache.
  - DS–second half of access to data cache.
  - TC–tag check, determine whether the data cache access hit.
  - WB–write back for loads and register-register operations.
- What is impact on Load delay?
  - Need 2 instructions between a load and its use!

Case Study: MIPS R4000

TWO Cycle Load Latency

IF	IS	RF	EX	DF	$\boxed{\text{DS}}$	TC	WB
	IF	IS	RF	EX	DF	DS	TC
		IF	IS	RF	EX	DF	DS
			IF	IS	RF	$\boxed{\text{EX}}$	DF
				IF	IS	RF	EX
					IF	IS	RF
						IF	IS
							IF

THREE Cycle Branch Latency

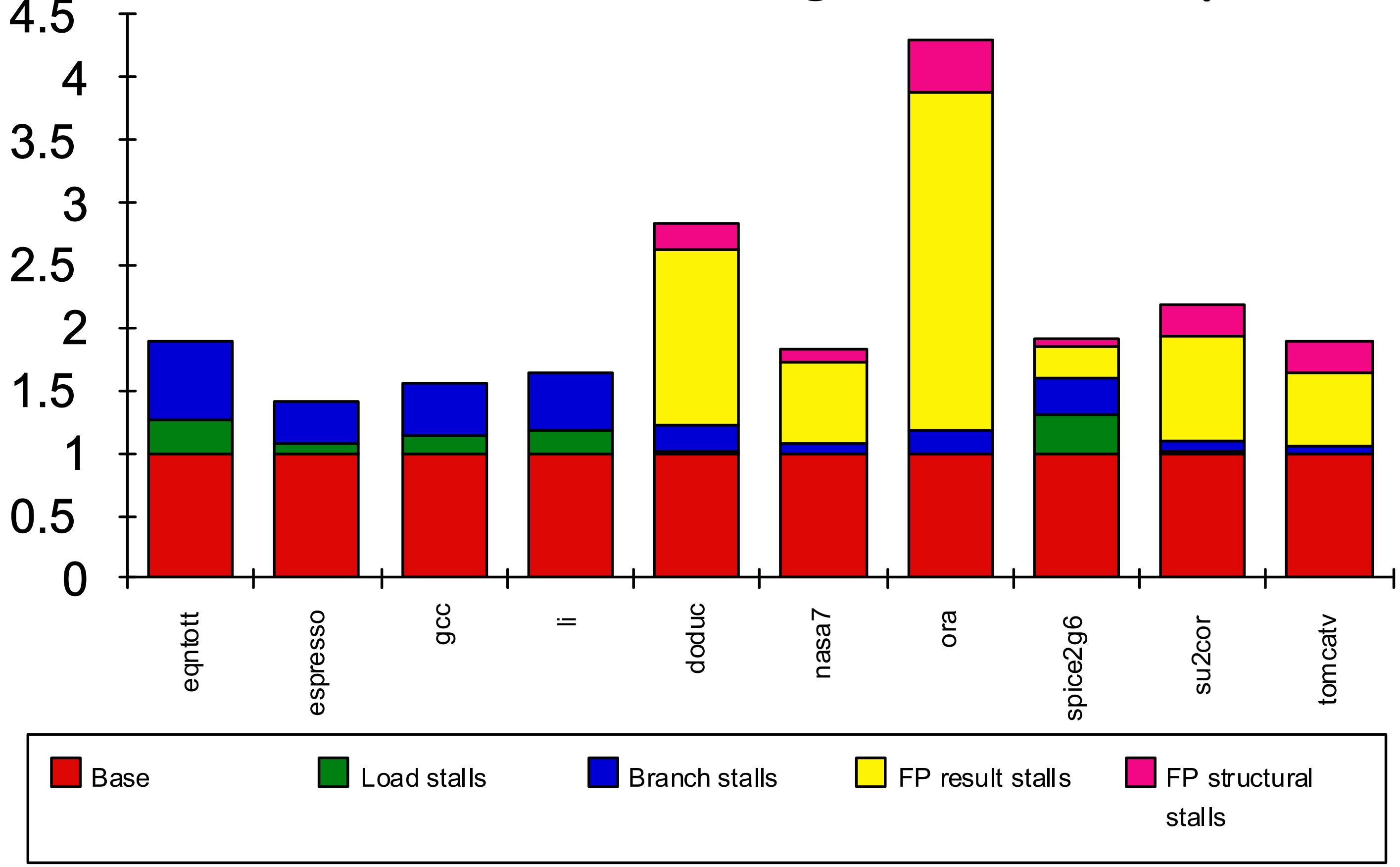
(conditions evaluated during EX phase)

IF	IS	RF	$\boxed{\text{EX}}$	DF	DS	TC	WB
	IF	IS	RF	EX	DF	DS	TC
		IF	IS	RF	EX	DF	DS
			IF	IS	RF	EX	DF
				$\boxed{\text{IF}}$	IS	RF	EX
					IF	IS	RF
						IF	IS
							IF

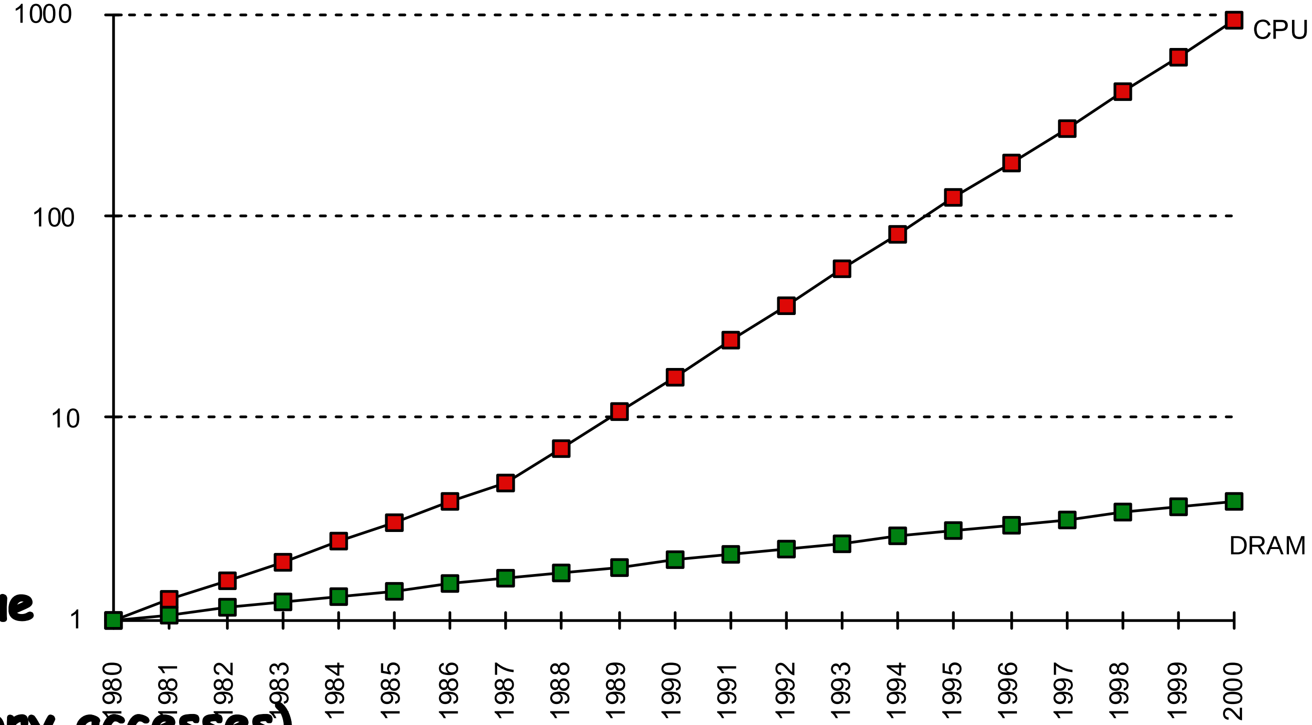
Delay slot plus two stalls
Branch likely cancels delay slot if not taken

R4000 Performance

- Not ideal CPI of 1:
  - Load stalls (1 or 2 clock cycles)
  - Branch stalls (2 cycles + unfilled slots)
  - FP result stalls: RAW data hazard (latency)
  - FP structural stalls: Not enough FP hardware (parallelism)



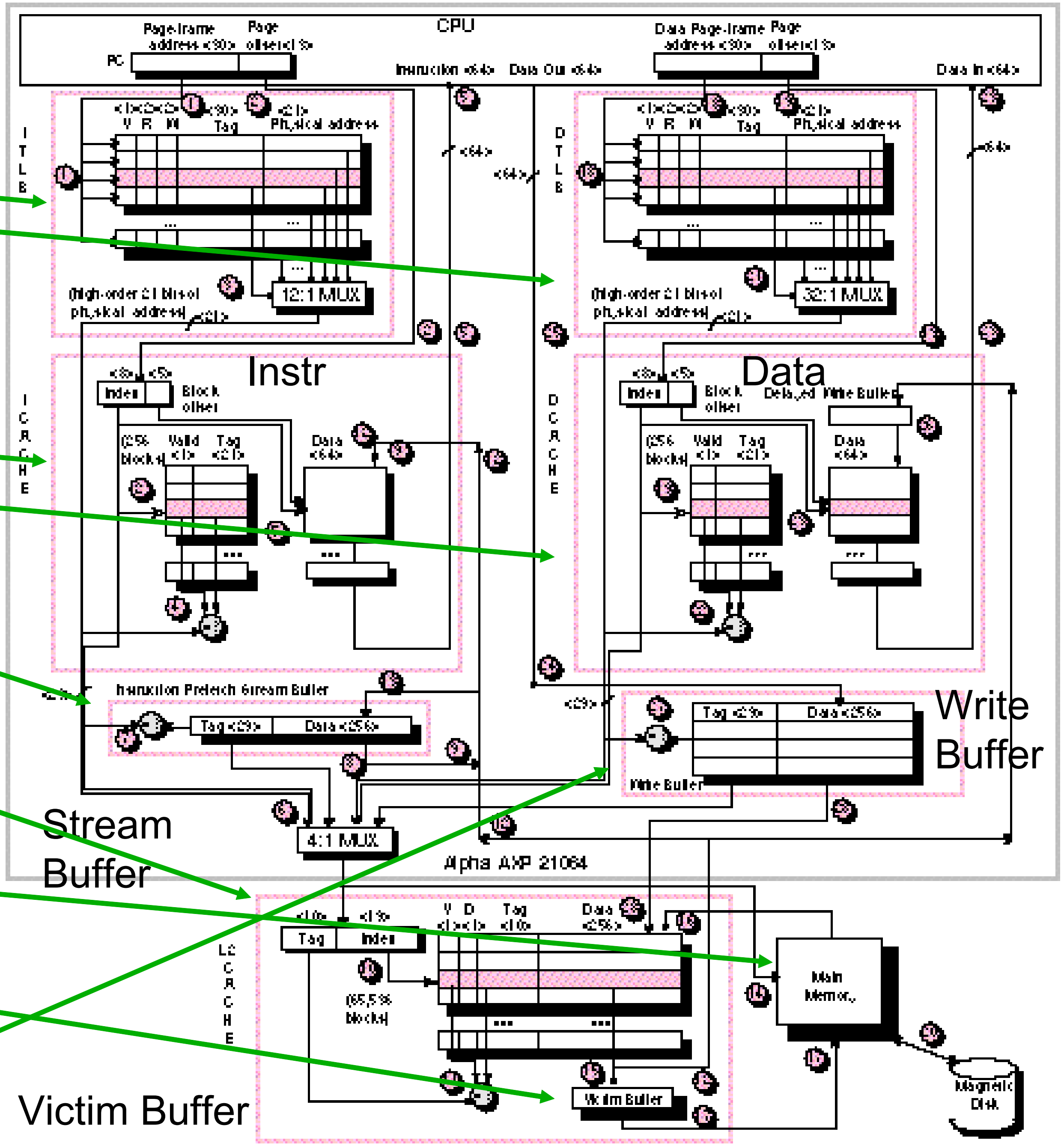
What is the Impact of What You've Learned About Caches?



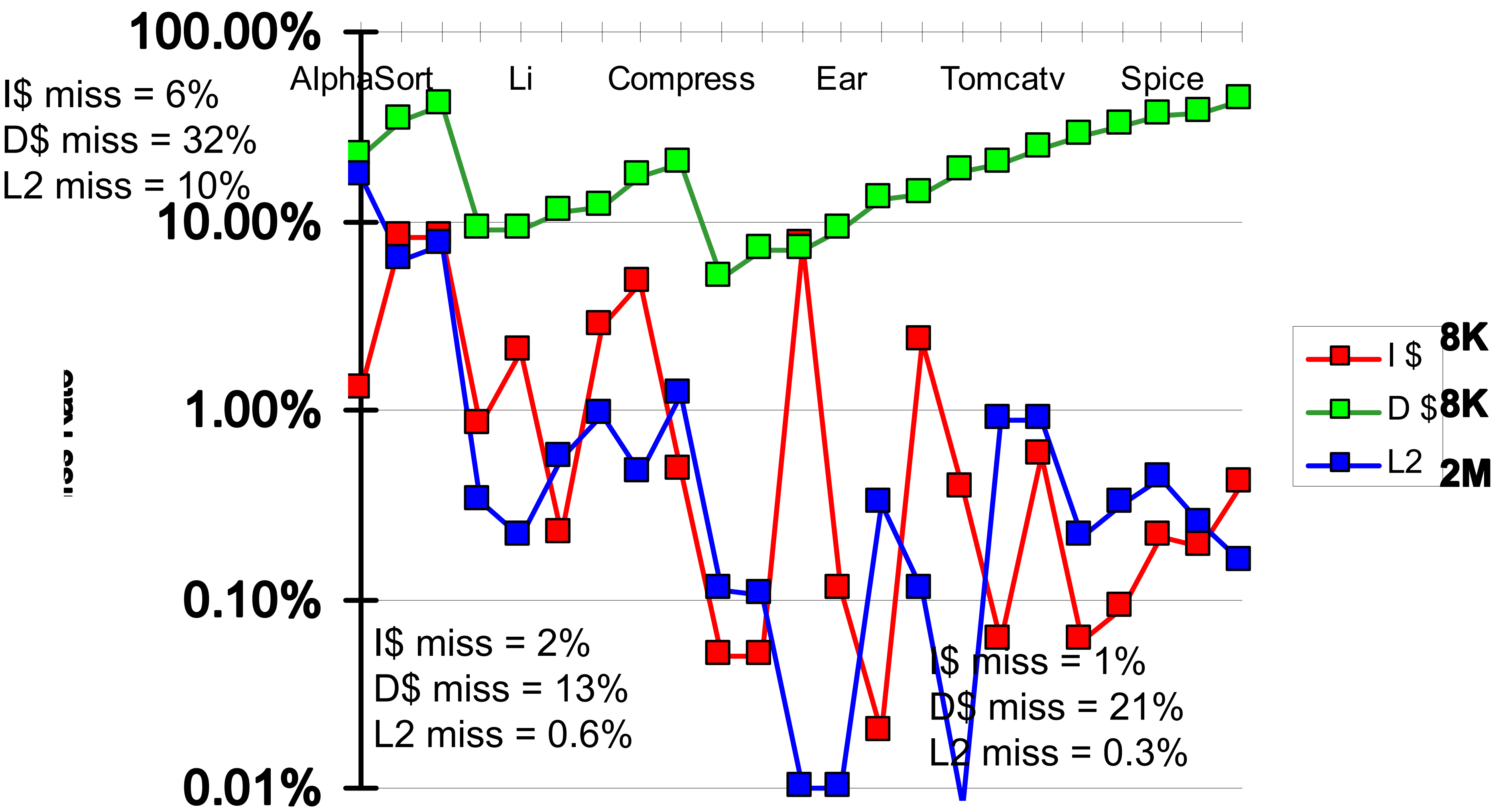
- **1960-1985: Speed = *f*(no. operations)**
- **1990**
  - **Pipelined Execution & Fast Clock Rate**
  - **Out-of-Order execution**
  - **Superscalar Instruction Issue**
- **1998: Speed = *f*(non-cached memory accesses)**
- **What does this mean for**
  - **Compilers?,Operating Systems?, Algorithms? Data Structures?**

Alpha 21064

- Separate Instr & Data TLB & Caches
- TLBs fully associative
- TLB updates in SW ("Priv Arch Libr")
- Caches 8KB direct mapped, write thru
- Critical 8 bytes first
- Prefetch instr. stream buffer
- 2 MB L2 cache, direct mapped, WB (off-chip)
- 256 bit path to main memory, 4 x 64-bit modules
- Victim Buffer: to give read priority over write
- 4 entry write buffer between D$ & L2$

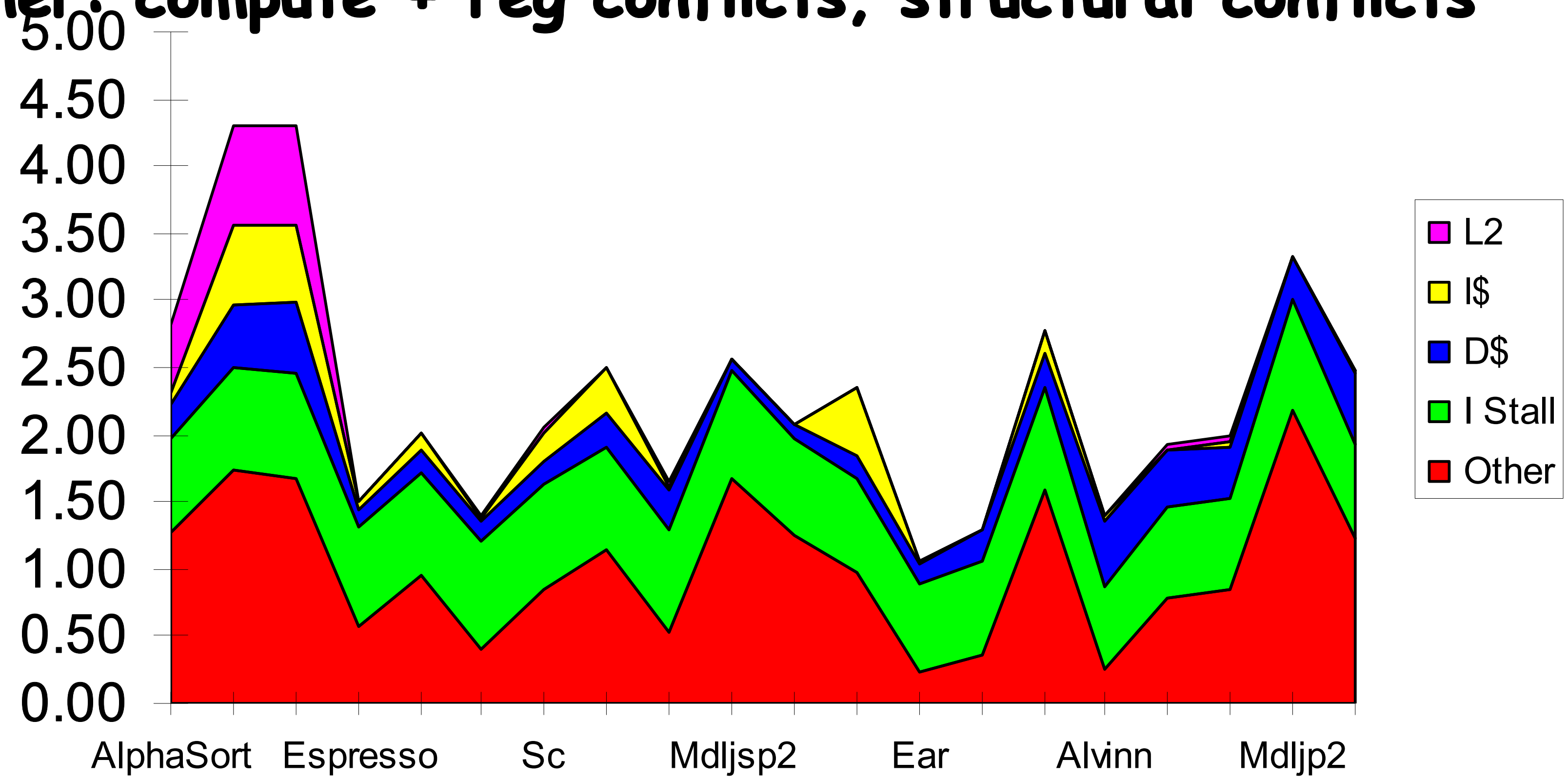


Alpha Memory Performance: Miss Rates of SPEC92



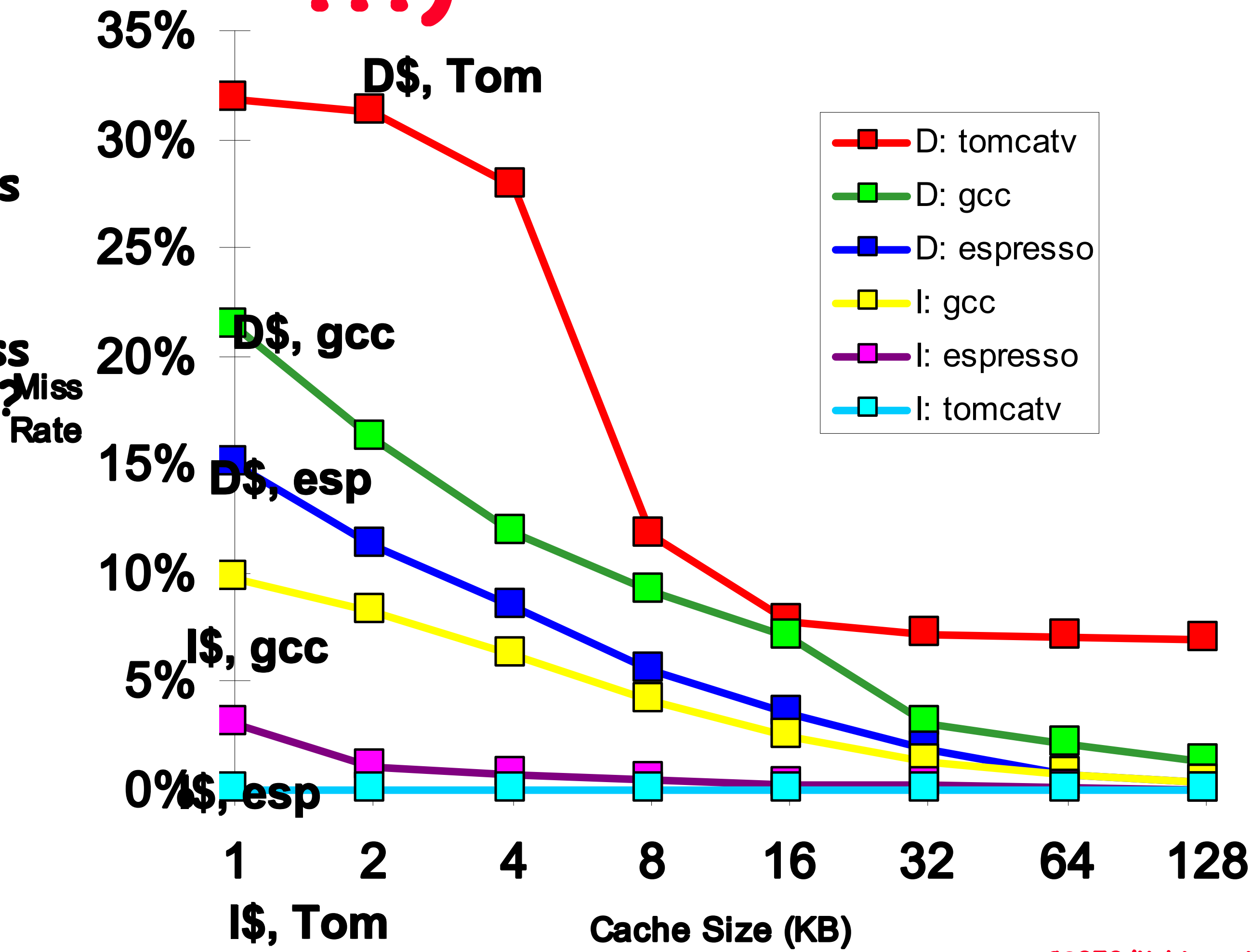
Alpha CPI Components

- Instruction stall: branch mispredict (green);
- Data cache (blue); Instruction cache (yellow); L2$ (pink)
  Other: compute + reg conflicts, structural conflicts



Pitfall: Predicting Cache Performance from Different Prog. (ISA, compiler, ...)

- 4KB Data cache miss rate 8%,12%, or 28%?
- 1KB Instr cache miss rate 0%,3%,or 10%?
- Alpha vs. MIPS for 8KB Data $: 17% vs. 10%
- Why 2X Alpha v. MIPS?



Cache Optimization Summary

	Technique	*MR*	*MP*	*HT*	*Complexity*
miss rate	Larger Block Size	+	–		0
	Higher Associativity	+		–	1
	Victim Caches	+			2
	Pseudo-Associative Caches	+			2
	HW Prefetching of Instr/Data	+			2
	Compiler Controlled Prefetching	+			3
	Compiler Reduce Misses	+			0
miss penalty	Priority to Read Misses		+		1
	Early Restart & Critical Word 1st		+		2
	Non-Blocking Caches		+		3
	Second Level Caches		+		2
	Better memory system		+		3
hit time	Small & Simple Caches	–		+	0
	Avoiding Address Translation			+	2
	Pipelining Caches			+	2