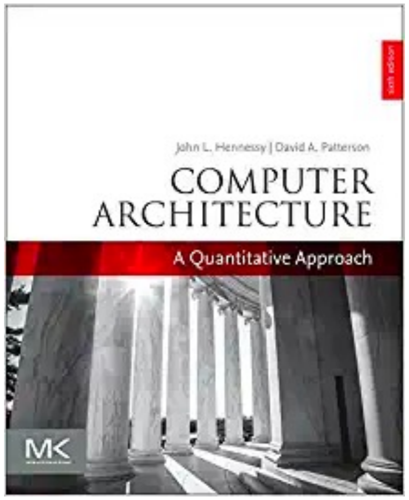


Chapter 2

Memory Hierarchy Design

Part 2: Beyond the Basics

> "Ideally one would desire an indefinitely large memory capacity such that any particular … word would be immediately available. … We are … forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible."
>
> – A. W. Burks, H. H. Goldstine, and J. von Neumann, *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument (1946)*

Acknowledgements

- Thanks to many sources for slide material

© 1990 Morgan Kaufmann Publishers, © 2001-present Elsevier
Computer Architecture: A Quantitative Approach by J. Hennessy & D. Patterson

© 1994 Morgan Kaufmann Publishers, © 2001-present Elsevier
Computer Organization and Design by D. Patterson & J. Hennessy

© 2002 K. Asinovic & Arvind, MIT

© 2002 J. Kubiatowicz, University of California at Berkeley

© 2006, © 2010 No Starch Press for Inside the Machine by J. Stokes

© 2007 W.-M. Hwu & D. Kirk, University of Illinois & NVIDIA

© 2007-2010 J. Owens, University of California at Davis

© 2010 CRC Press for Introduction to Concurrency in Programming Languages by M. Sottile, T. Mattson, and C. Rasmussen

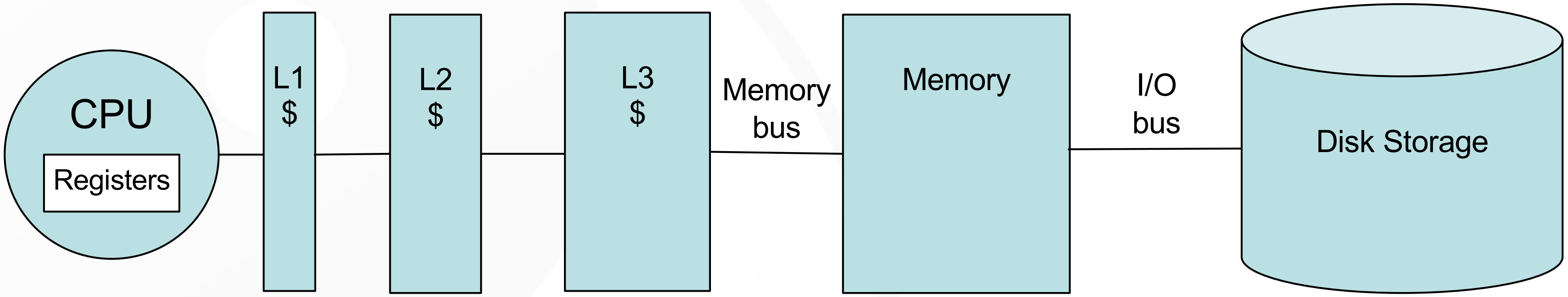
© 2017, IBM POWER9 Processor Architecture by Sadasivam et al., IBM

© 2016, © 2019 POWER9 Processor User’s Manual, IBM

© The OpenPOWER Foundation

Memory Hierarchy

Key: $ = cache



CPU

Registers

L1 $

L2 $

L3 $

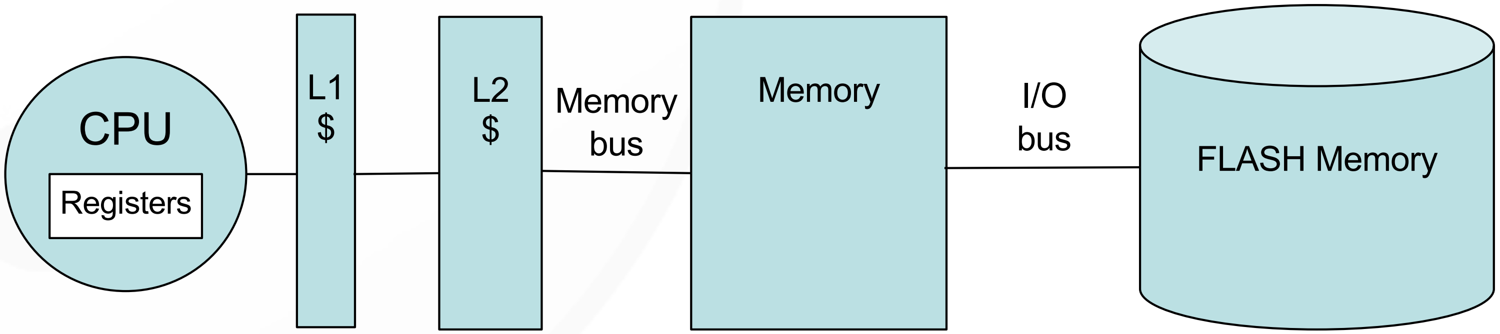
Memory bus

Memory

I/O bus

Disk Storage

	Register reference	Level 1 $ reference	Level 2 $ reference	Level 3 $ reference	Memory reference	Disk memory reference
Size:	1000 bytes	64 kB	256 kB	8-16 MB	8-16 GB	1-8 TB
Speed:	0.3 ns	1 ns	3-10 ns	20-30 ns	80-100 ns	5,000,000-10,000,000 ns



CPU

Registers

L1 $

L2 $

Memory bus

Memory

I/O bus

FLASH Memory

	Register reference	Level 1 $ reference	Level 2 $ reference	Memory reference	FLASH memory reference
Size:	500 bytes	64 kB	256 kB	4-6 GB	256-512 GB
Speed:	0.3 ns	2 ns	10-20 ns	80-100 ns	25,000-50,000 ns

Basic Cache Optimizations

Review Appendix B,
as needed.

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

Recall: Avg. Memory Access Time

$$\frac{\text{Misses}}{\text{Instruction}} = \frac{\text{Miss rate} \times \text{Memory accesses}}{\text{Instruction count}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

$$\boxed{\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}}$$

- How to reduce the *average memory access time?*
  - Reduce hit time
  - Reduce miss rate
  - Reduce miss penalty

Advanced Optimizations for Caching

Key
$ = cache

- Reduce Hit Time
  (1) Small & simple first-level $ and (2) way prediction
    - Side effect: Reduce power consumption
- Increase Cache Bandwidth
  (3) Pipelined $, (4) non-blocking $, and (5) multi-banked $
    - Side effect: Varying impacts on power consumption
- Reduce Miss Penalty
  (6) Critical word first and (7) merging write buffers
    - Side effect: Little impact on power
- Reduce Miss Rate
  (8) Compiler optimizations. Side effect: Reduces power consumption
- Reduce Miss Penalty or Miss Rate via Parallelism
  (9) Hardware pre-fetching and (10) compiler pre-fetching

Advanced Optimizations for Caching

Key

$ = cache

- Reduce Hit Time

  (1) Small & simple first-level $ and (2) way prediction

  - Side effect: Reduce power consumption

- Increase Cache Bandwidth

  (3) Pipelined $, (4) non-blocking $, and (5) multi-banked $

  - Side effect: Varying impacts on power consumption

- Reduce Miss Penalty

  (6) Critical word first and (7) merging write buffers

  - Side effect: Little impact on power

- Reduce Miss Rate

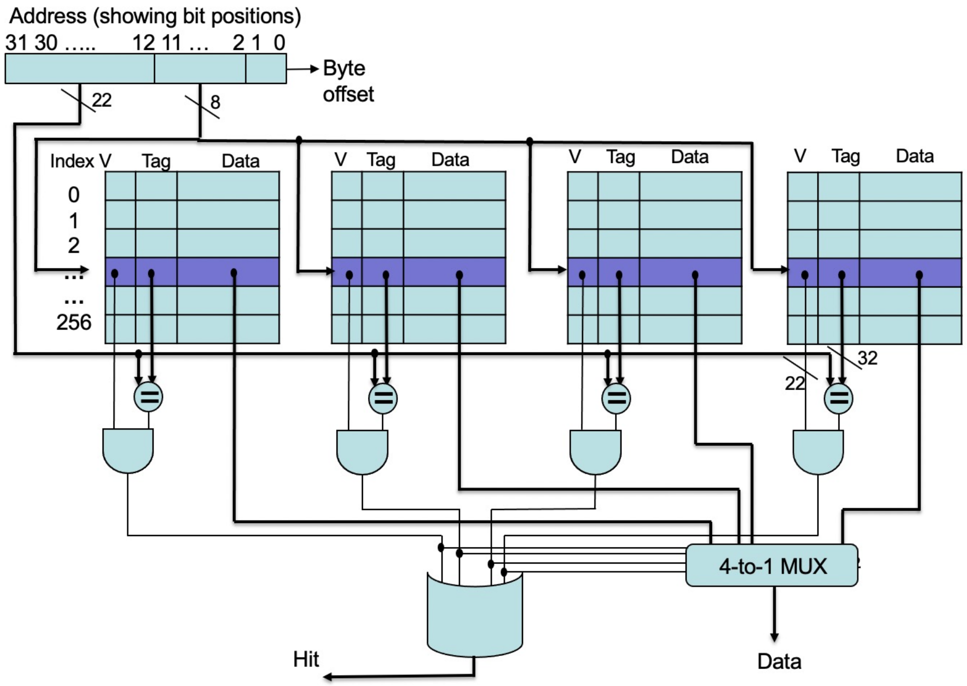
  (8) Compiler optimizations. Side effect: Reduces power consumption

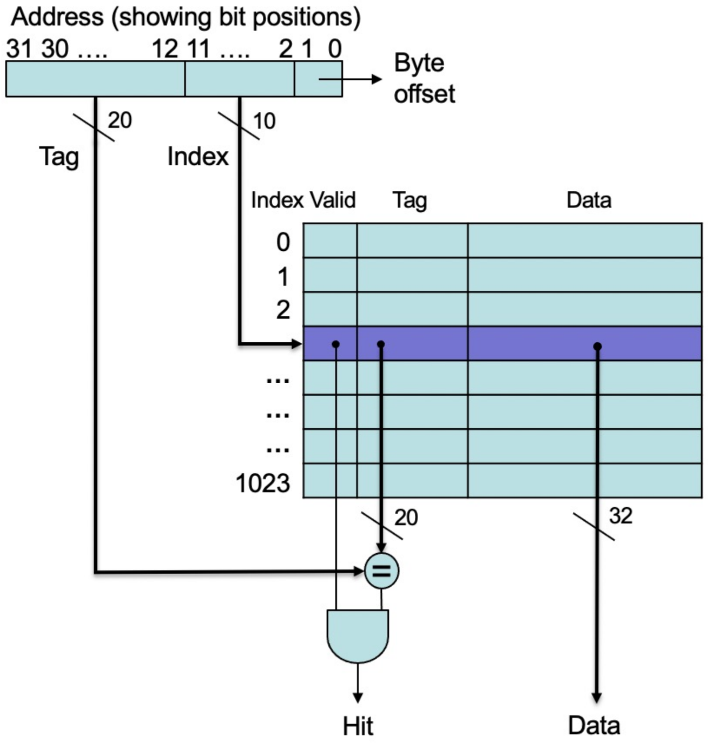
- Reduce Miss Penalty or Miss Rate via Parallelism

  (9) Hardware pre-fetching and (10) compiler pre-fetching

Small & Simple First-Level Caches

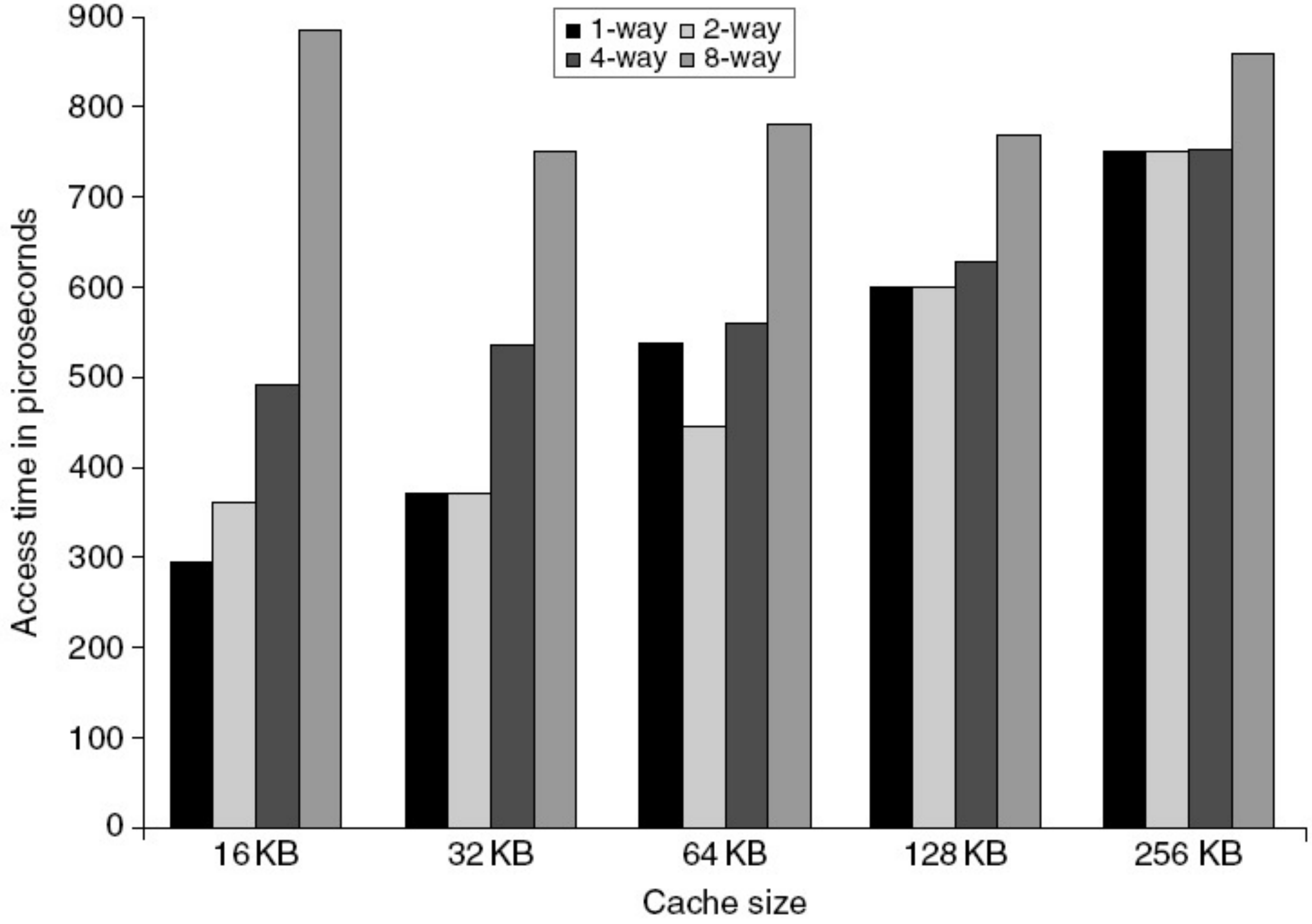
- *Critical timing path*
    - addressing tag memory, then
    - comparing tags, then
    - selecting correct set





- *Direct-mapped caches* can overlap tag compare and transmission of data
- *Lower associativity* reduces power because fewer cache lines are accessed

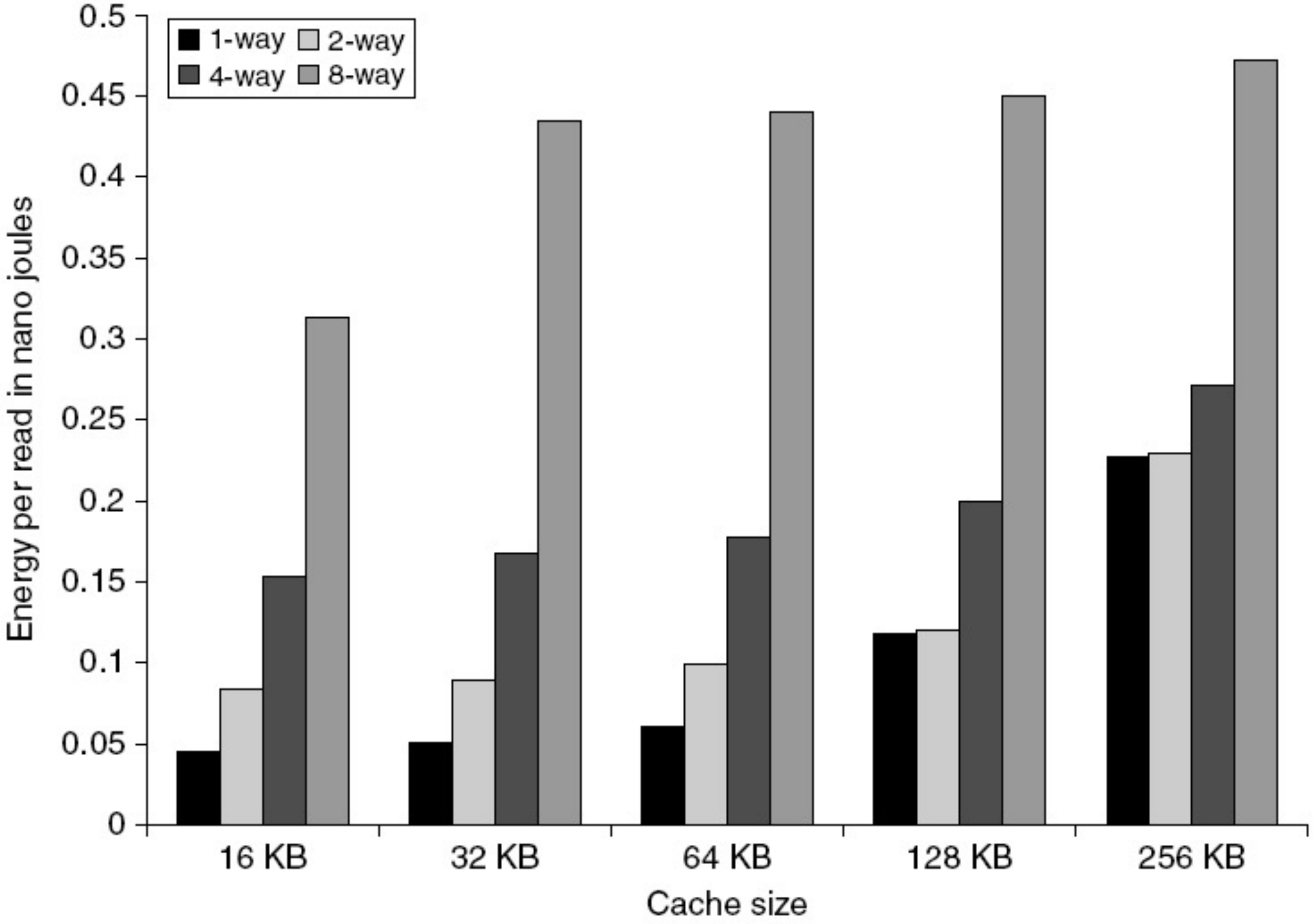
L1 Size and Associativity



Access time vs. size and associativity

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

L1 Size and Associativity



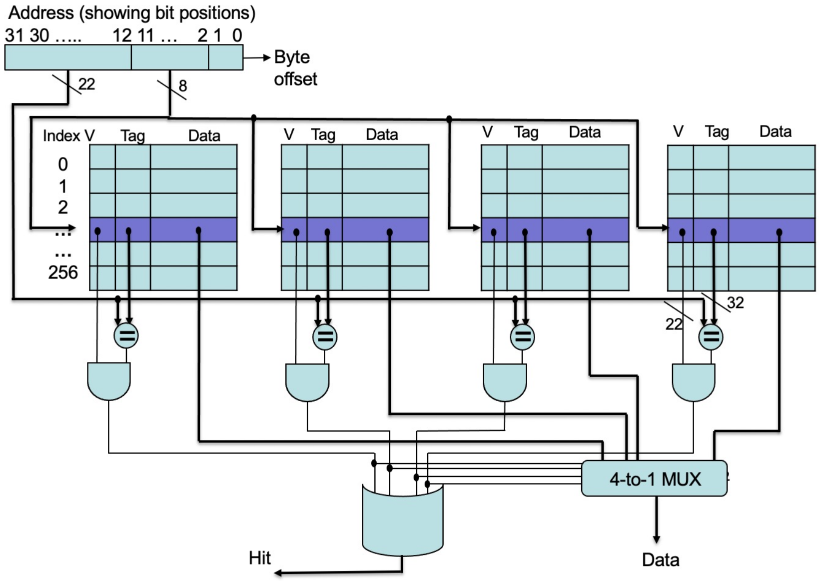
Energy per read vs. size and associativity

Way Prediction:

Predict the “Way” or Block within the Set

- To improve hit time, predict the “way” to pre-set mux
  - Misprediction 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 misprediction penalty

How to combine fast hit time of direct-mapped $ and have the lower conflict miss rate of of 2-way set associative $?



Advanced Optimizations for Caching

Key

$ = cache

- Reduce Hit Time

  (1) Small & simple first-level $ and (2) way prediction

  - Side effect: Reduce power consumption

- Increase Cache Bandwidth

  (3) Pipelined $, (4) non-blocking $, and (5) multi-banked $

  - Side effect: Varying impacts on power consumption

- Reduce Miss Penalty

  (6) Critical word first and (7) merging write buffers

  - Side effect: Little impact on power

- Reduce Miss Rate

  (8) Compiler optimizations. Side effect: Reduces power consumption

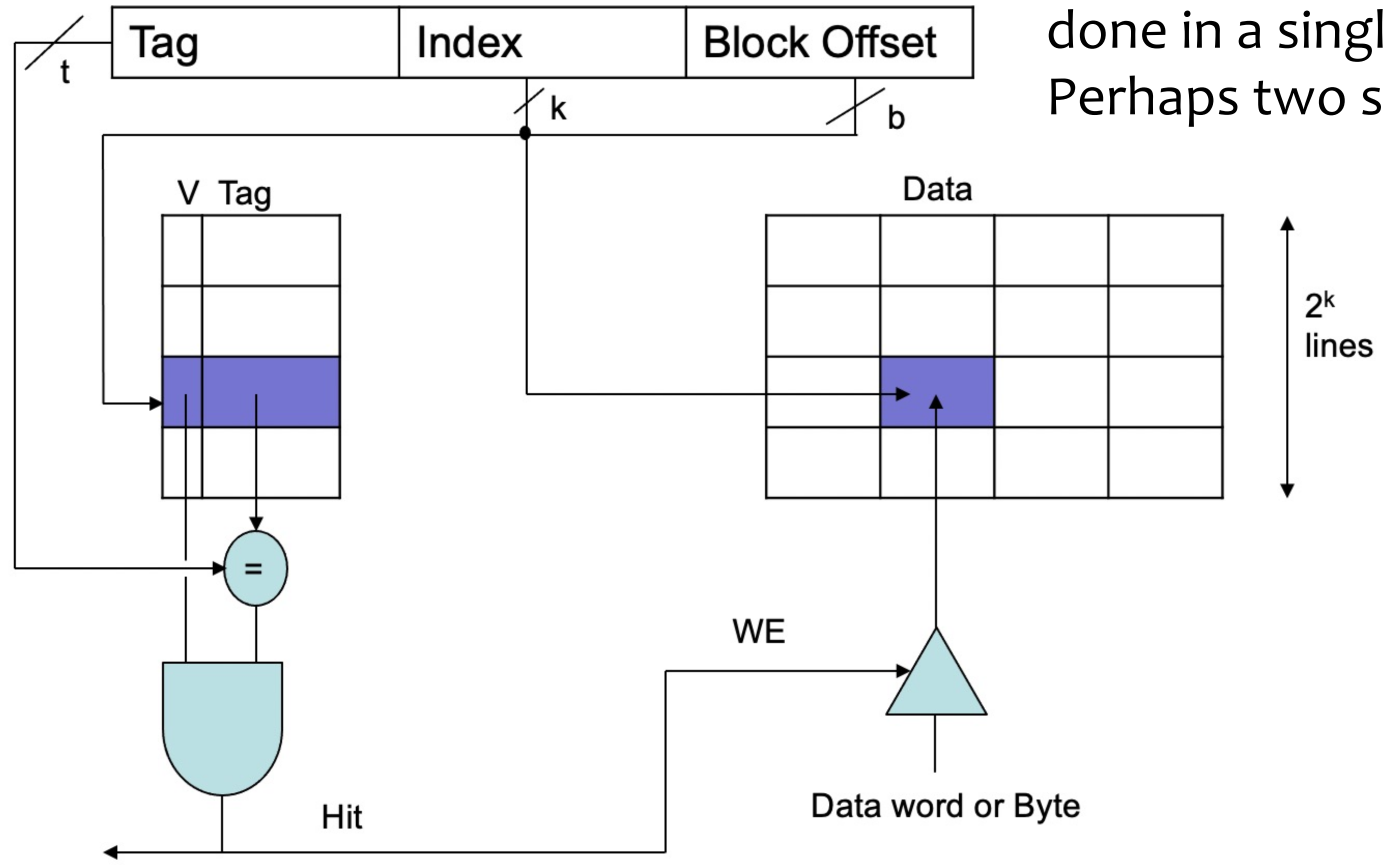
- Reduce Miss Penalty or Miss Rate via Parallelism

  (9) Hardware pre-fetching and (10) compiler pre-fetching

Write Performance

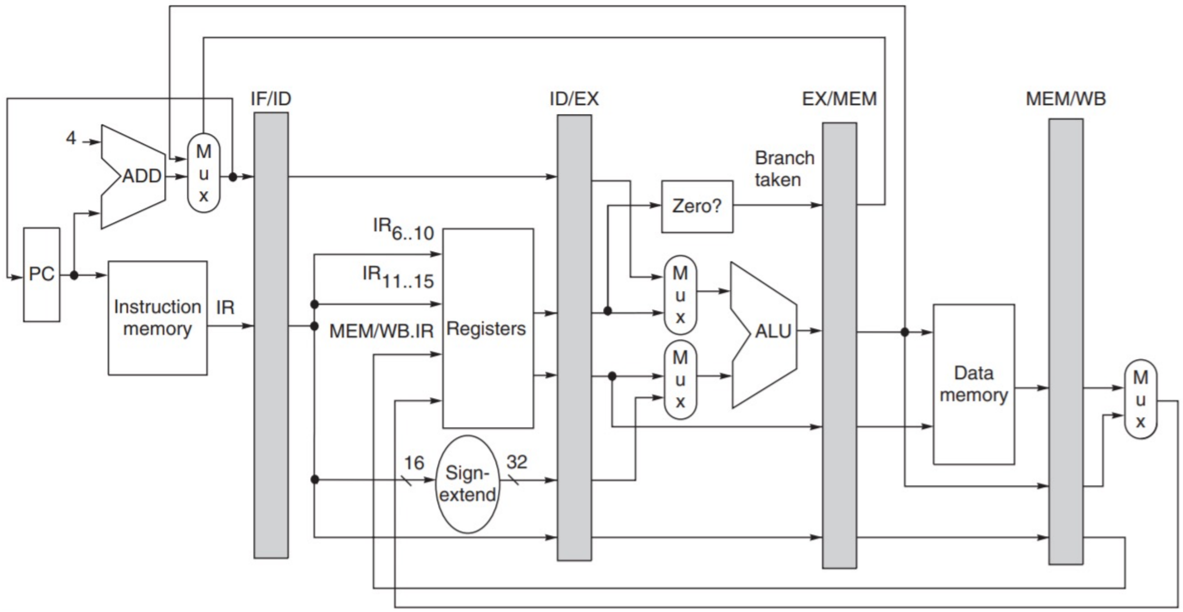
- Steps: (1) Index tag array. (2) Compare tags. (3) Check valid bit. (4) If valid, enable write to memory location.

Serial set of steps that *can* be done in a single (long) cycle. Perhaps two shorter cycles?

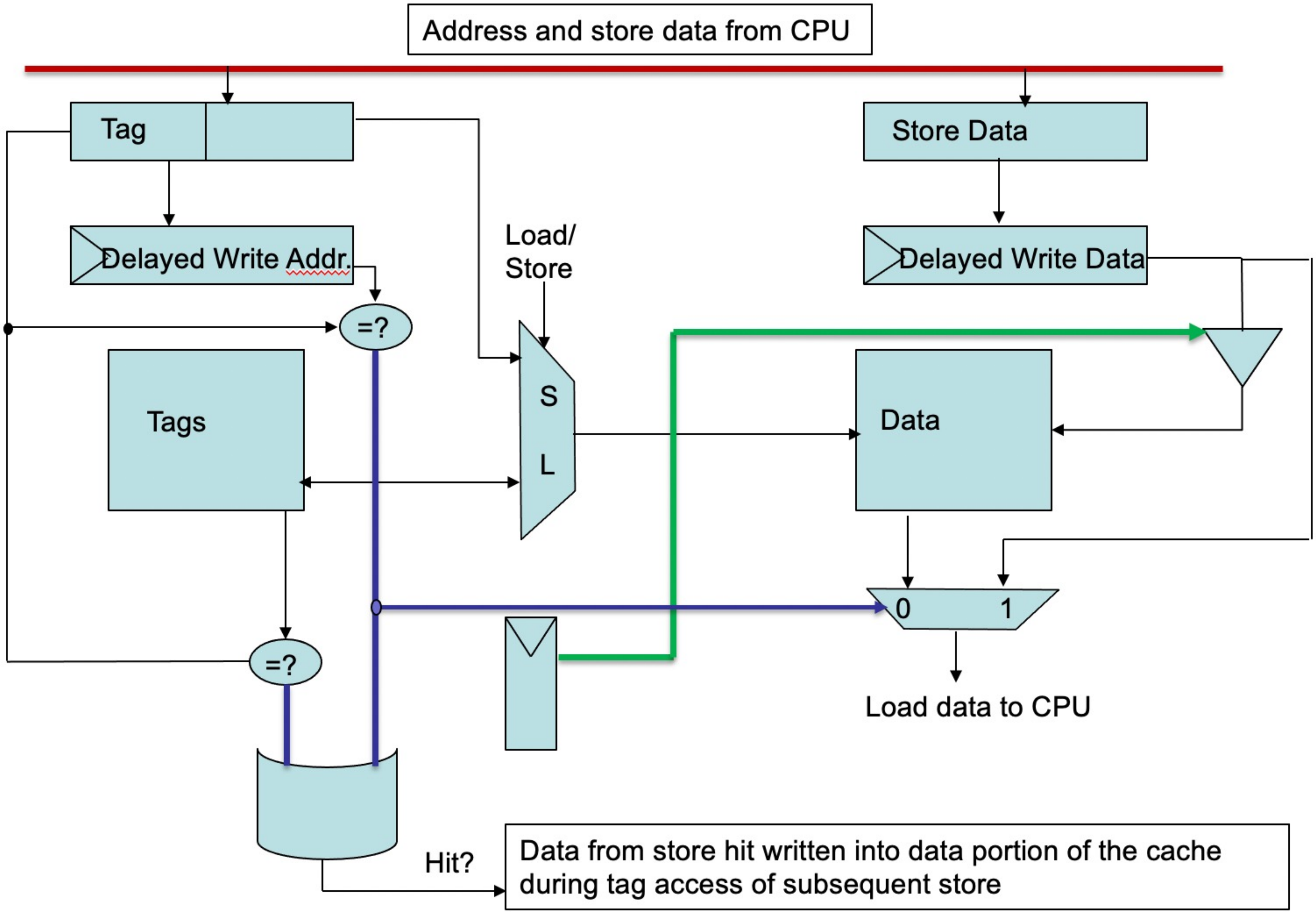


Write Performance

- Problem: Writes take two cycles in the memory stage, one cycle for tag check plus one cycle for data write if hit
- Solutions
  - Design data RAM that can perform read and write concurrently, restore old value after tag miss
  - Hold write data for store in single buffer ahead of the cache; write cache data during the next store’s tag check.



Pipelining Cache Writes



Data from store hit written into data portion of the cache during tag access of subsequent store

Pipelining Cache

- Pipeline cache access to *improve bandwidth*
  - Examples
    - Pentium: 1 cycle
    - Pentium Pro – Pentium III: 2 cycles
    - Pentium 4 – Core i7: 4 cycles

  but the increased number of pipeline stages leads to …
- Increases in branch misprediction penalty
  - Is it easier to pipeline the instruction cache or data cache?
- Makes it easier to increase associativity

In practice?

- All CPUs pipeline L1 cache access simply to separate the access and the hit detection stages.
- When does banking work best?

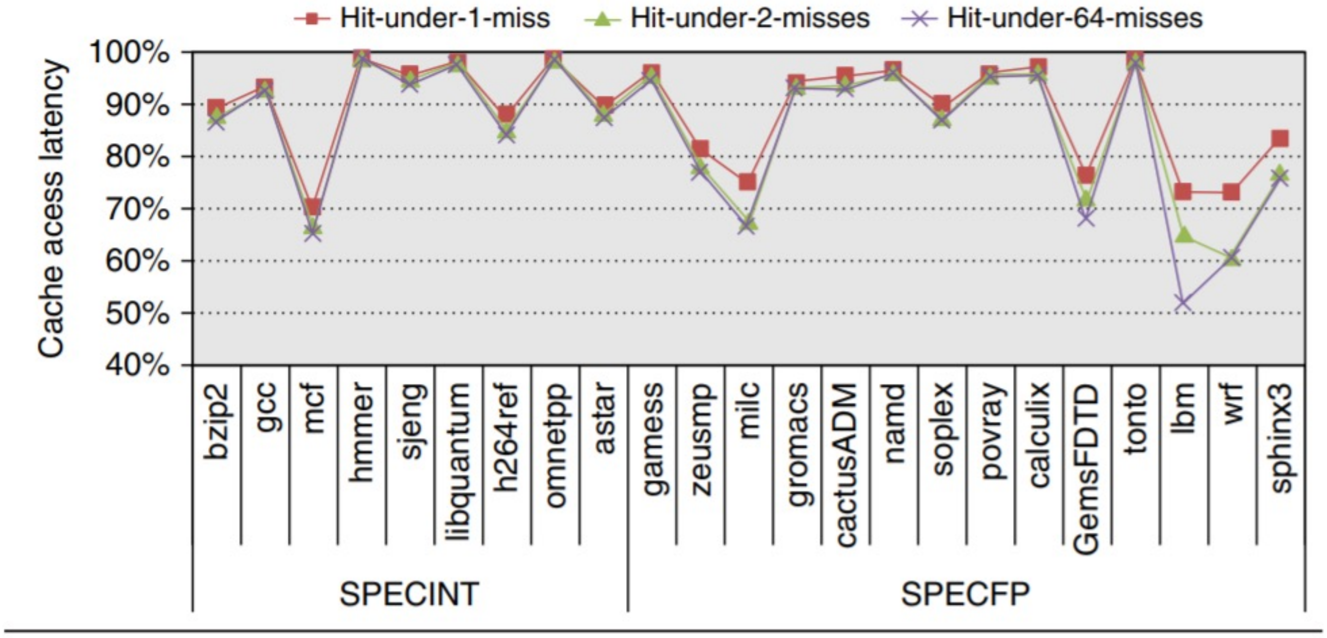
Non-blocking Caches

Stall CPU on $ Miss

Hit under $ Miss

Multiple Outstanding $ Misses

Non-blocking Caches: Basic Idea



- 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

Basic MIPS Architecture

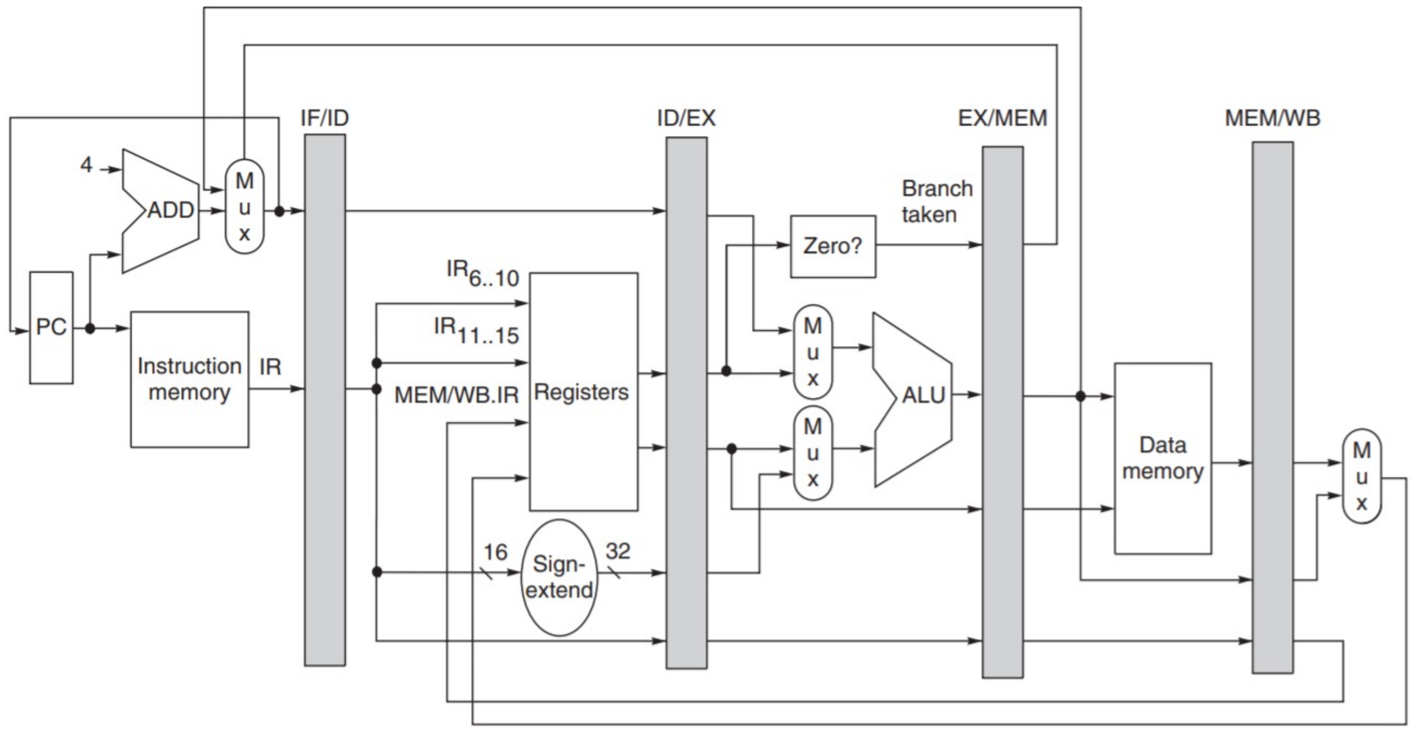


Figure C.22 The data path is pipelined by adding a set of registers, one between each pair of pipe stages. The registers serve to convey values and control information from one stage to the next. We can also think of the PC as a pipeline register, which sits before the IF stage of the pipeline, leading to one pipeline register for each pipe stage. Recall that the PC is an edge-triggered register written at the end of the clock cycle; hence, there is no race condition in writing the PC. The selection multiplexer for the PC has been moved so that the PC is written in exactly one stage (IF). If we didn't move it, there would be a conflict when a branch occurred, since two instructions would try to write different values into the PC. Most of the data paths flow from left to right, which is from earlier in time to later. The paths flowing from right to left (which carry the register write-back information and PC information on a branch) introduce complications into our pipeline.

Non-blocking Caches: Details

- *Non-blocking cache* or *lockup-free cache*
  - Allows data $ to continue to supply $ hits during a miss
- *"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 effective miss penalty by overlapping multiple misses
  - Examples
    - Pentium Pro allows 4 outstanding memory misses
    - (Cray X1E vector supercomputer allows 2,048 outstanding memory misses)
- Issues?

Non-blocking Cache: Example

- Assume the following information
  - Sustained transfer rate: 16 GB/s
  - Memory-access time: 36 ns
  - Block size: 64 bytes

  What is the maximum number of outstanding references to maintain peak bandwidth for a system?

- Answer: $(16*10^9) / 64 * (36*10^{-9}) = 9$

Multi-banked Caches

- Organize cache as independent banks to support simultaneous access
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel Core i7 supports 4 banks for L1 and 8 banks for L2
- Interleave banks according to block address
  - Simple mapping that works well? "Sequential Interleaving"
    - Spread block addresses sequentially across banks
    - Example: If 4 banks, Bank 0 has all blocks whose *address mod 4* is 0; bank 1 has all blocks whose *address mod 4* is 1; …

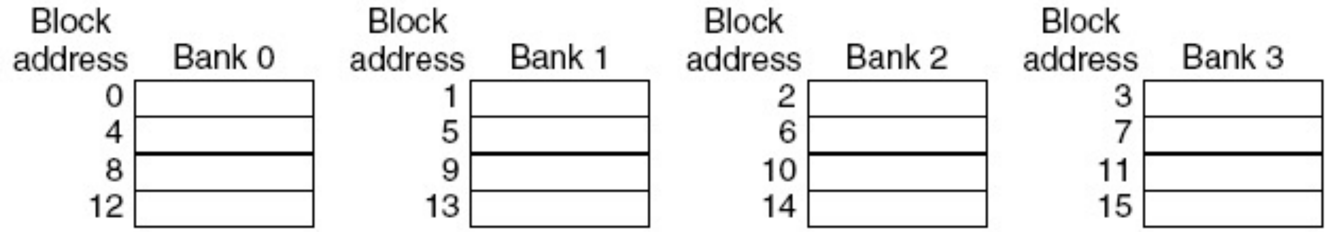


Figure 2.6 **Four-way interleaved cache banks using block addressing.** Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

Advanced Optimizations for Caching

- Reduce Hit Time

  (1) Small & simple first-level \$ and (2) way prediction

  - Side effect: Reduce power consumption

- Increase Cache Bandwidth

  (3) Pipelined \$, (4) non-blocking \$, and (5) multi-banked \$

  - Side effect: Varying impacts on power consumption

- Reduce Miss Penalty

  (6) Critical word first and (7) merging write buffers

  - Side effect: Little impact on power

- Reduce Miss Rate

  (8) Compiler optimizations. Side effect: Reduces power consumption

- Reduce Miss Penalty or Miss Rate via Parallelism

  (9) Hardware pre-fetching and (10) compiler pre-fetching

Key

\$ = cache

Critical Word First or Early Restart

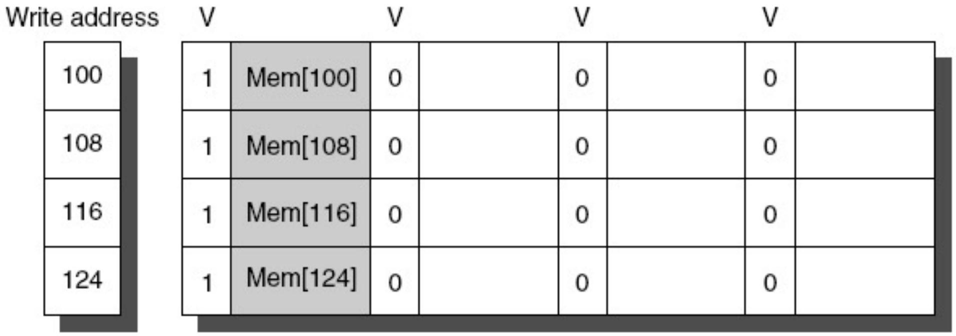
Don’t wait for full block before restarting CPU

- Critical Word First
  - Request missed word from memory first
  - Send it to the processor as soon as it arrives
  - Let processor continue execution while filling the rest of the words in the block
- Early Restart
  - Request words in normal order
  - Send missed work to the processor as soon as it arrives
  - Let the CPU continue execution
- Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched

Which one more widely used?
Why?

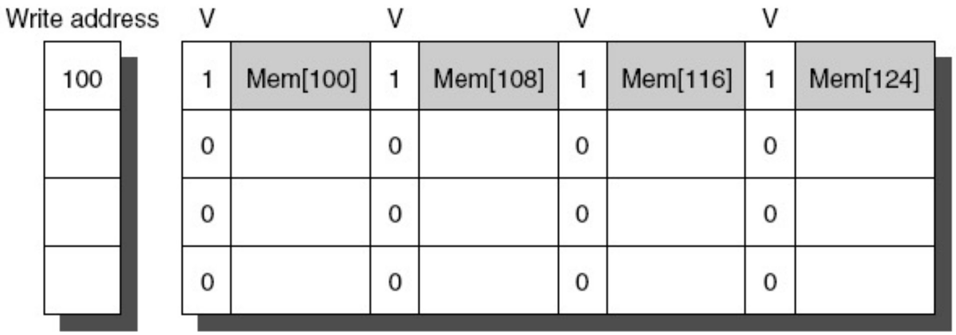
Merging Write Buffer

- 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

Advanced Optimizations for Caching

- Reduce Hit Time

  (1) Small & simple first-level $ and (2) way prediction

  - Side effect: Reduce power consumption

- Increase Cache Bandwidth

  (3) Pipelined $, (4) non-blocking $, and (5) multi-banked $

  - Side effect: Varying impacts on power consumption

- Reduce Miss Penalty

  (6) Critical word first and (7) merging write buffers

  - Side effect: Little impact on power

- Reduce Miss Rate

  (8) Compiler optimizations. Side effect: Reduces power consumption

- Reduce Miss Penalty or Miss Rate via Parallelism

  (9) Hardware pre-fetching and (10) compiler pre-fetching

Key

$ = cache

Compiler Optimizations

- Restructuring code affects the data block access sequence
  - Group data accesses together to improve spatial locality
  - Re-order data accesses to improve temporal locality
- Prevent data from entering the cache
  - Useful for variables that will only be accessed once before being replaced
  - Needs mechanism for software to tell hardware *not* to cache data (i.e., instruction hints or page table bits)
- Kill data that will never be used again
  - Streaming data exploits spatial locality but not temporal locality
  - Replace into dead cache locations

Compiler Optimizations

- Loop Interchange
  - Swap nested loops to access memory in sequential order

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

- How does the above change the memory access pattern?

- What locality is improved?

Compiler Optimizations

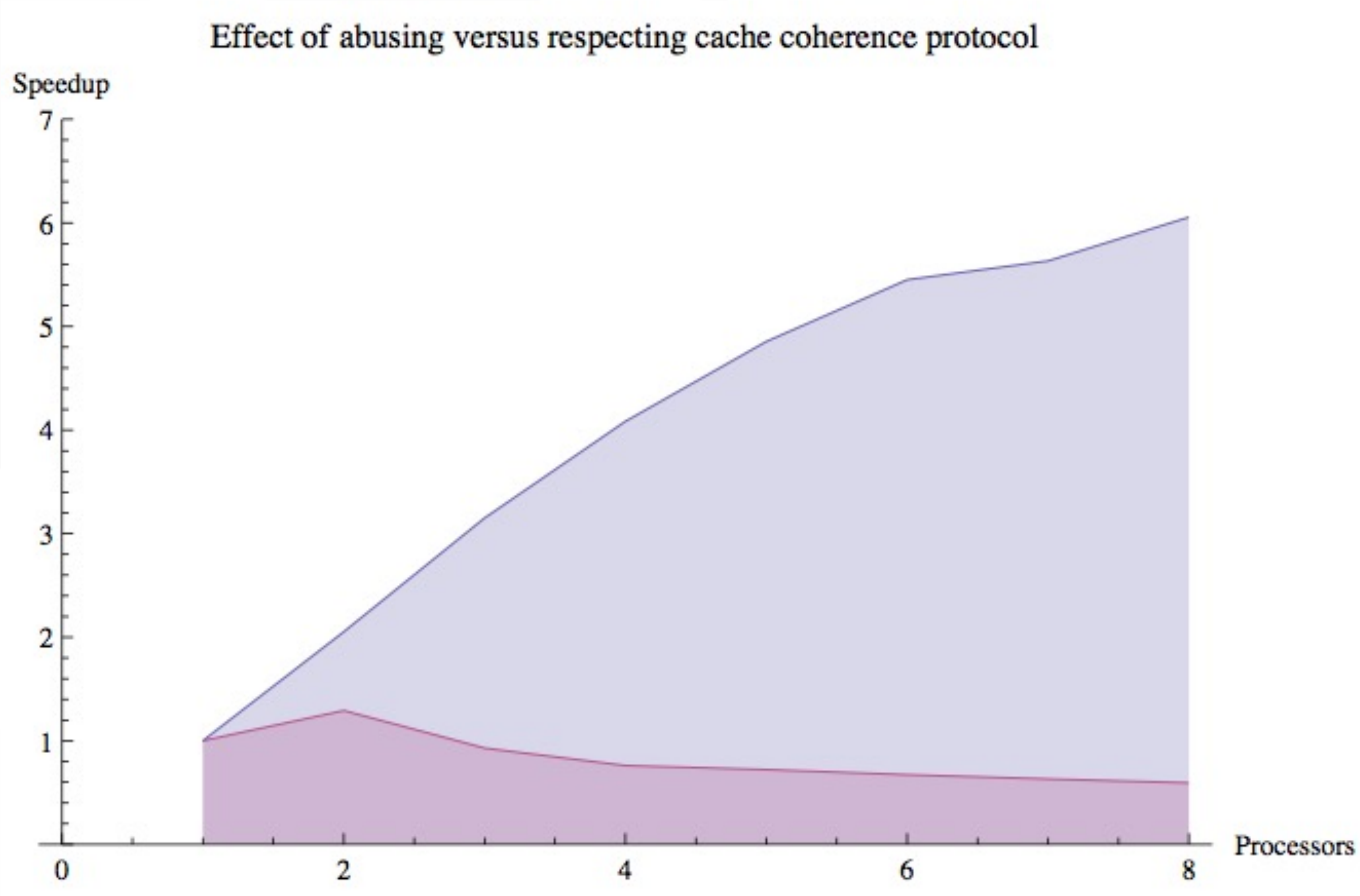
- What optimization(s) do you see?
- How does the optimization(s) improve locality?
- What locality is improved?

```
for (i=0; i< N; i++)
  for (j=0; j < M; j++)
    a[i][j] = b[i][j] * c[i][j];

for (i=0; i< N; i++)
  for (j=0; j < M; j++)
    d[i][j] = a[i][j] * c[i][j];
```

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

Impact of Cache Coherence in Multicore CPUs



Compiler Optimizations

- Blocking
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
  - Requires more memory accesses but improves locality of accesses

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

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

Advanced Optimizations for Caching

Key
$ = cache

- Reduce Hit Time

  (1) Small & simple first-level $ and (2) way prediction

    - Side effect: Reduce power consumption

- Increase Cache Bandwidth

  (3) Pipelined $, (4) non-blocking $, and (5) multi-banked $

    - Side effect: Varying impacts on power consumption

- Reduce Miss Penalty

  (6) Critical word first and (7) merging write buffers

    - Side effect: Little impact on power

- Reduce Miss Rate

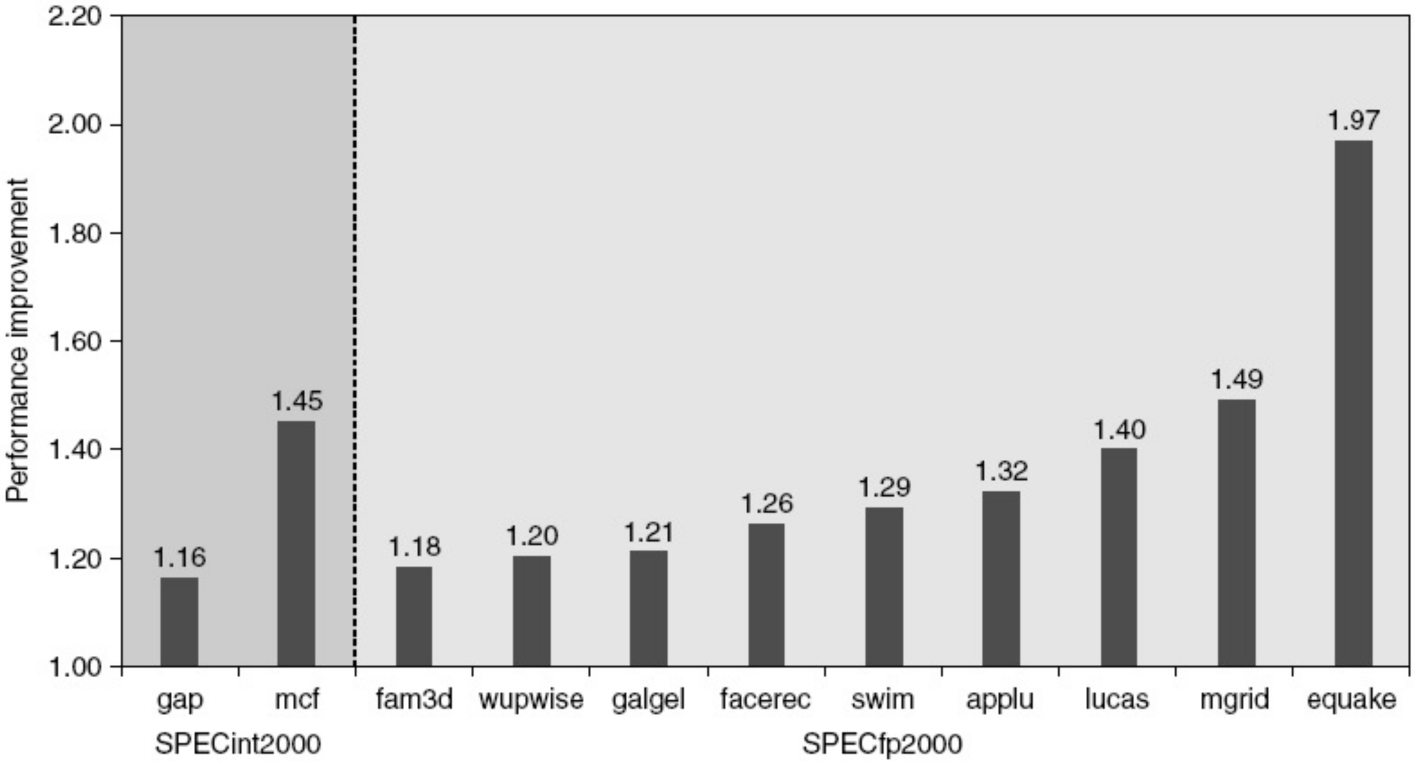
  (8) Compiler optimizations. Side effect: Reduces power consumption

- Reduce Miss Penalty or Miss Rate via Parallelism

  (9) Hardware pre-fetching and (10) compiler pre-fetching

Hardware Prefetching

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



Pentium 4 Pre-fetching

Compiler Prefetching

- Insert prefetch instructions before data is needed
- Non-faulting: prefetch doesn't cause exceptions

- Register prefetch
  - Loads data into register
- Cache prefetch
  - Loads data into cache

- Combine with loop unrolling and software pipelining

Summary of Advanced $ Optimizations

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

Memory Technology

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

- DRAM used for main memory, SRAM used for cache

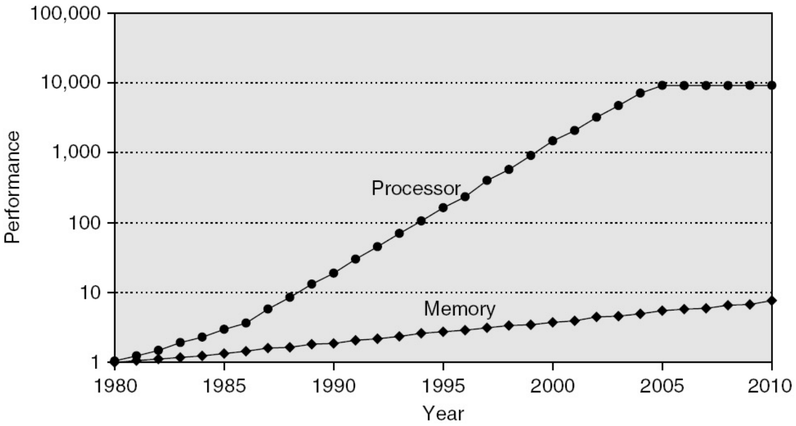
Memory Technology

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

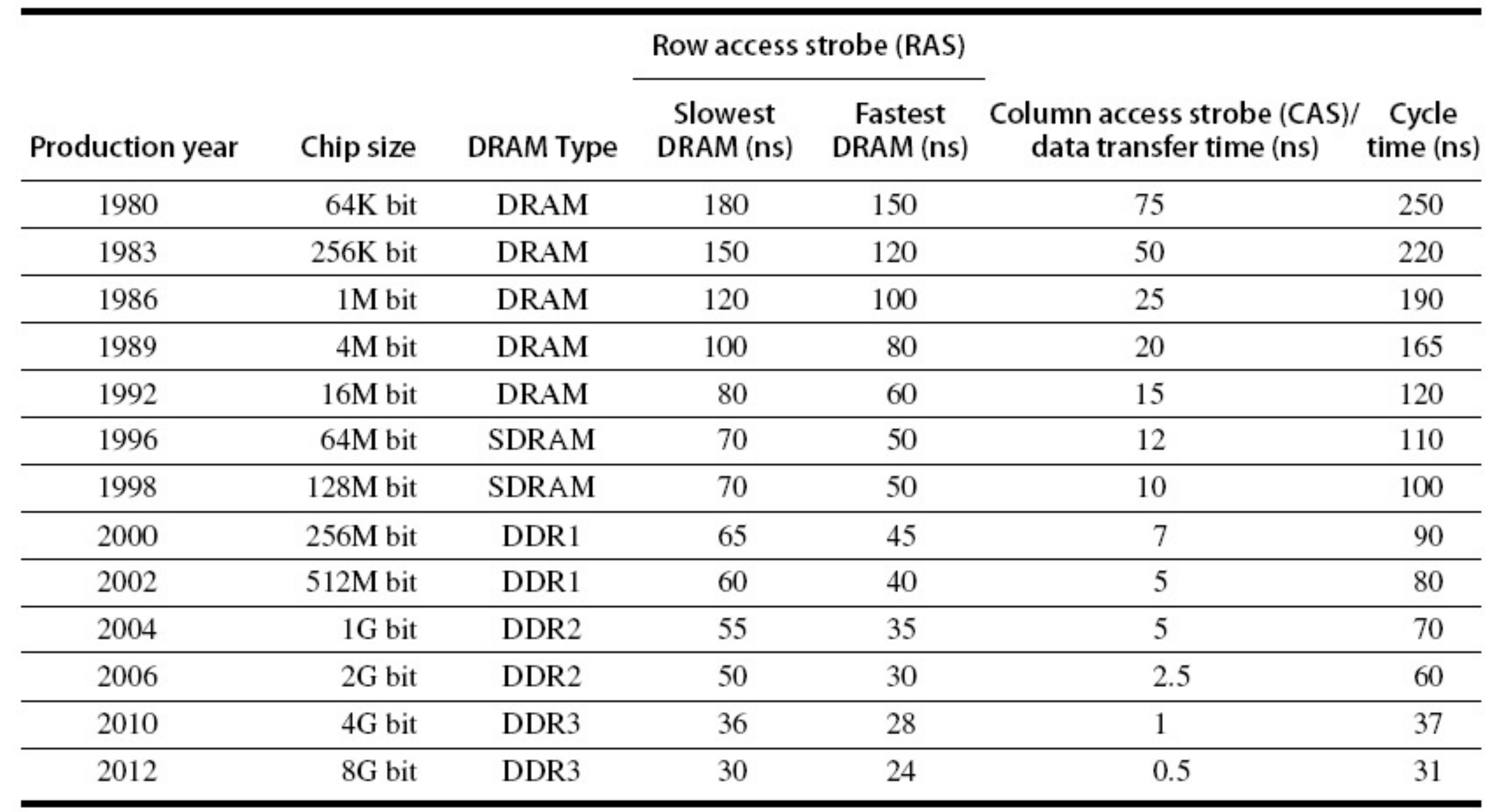
Memory Technology

- Amdahl
  - Memory speed should grow linearly with processor speed
  - Unfortunately, memory speed has not kept pace with processors
- Optimizations
  - Multiple accesses to same row
  - Synchronous DRAM
    - Added clock to DRAM interface
    - Burst mode with critical word first
  - Wider interfaces
  - Double data rate (DDR)
  - Multiple banks on each DRAM device

Recall

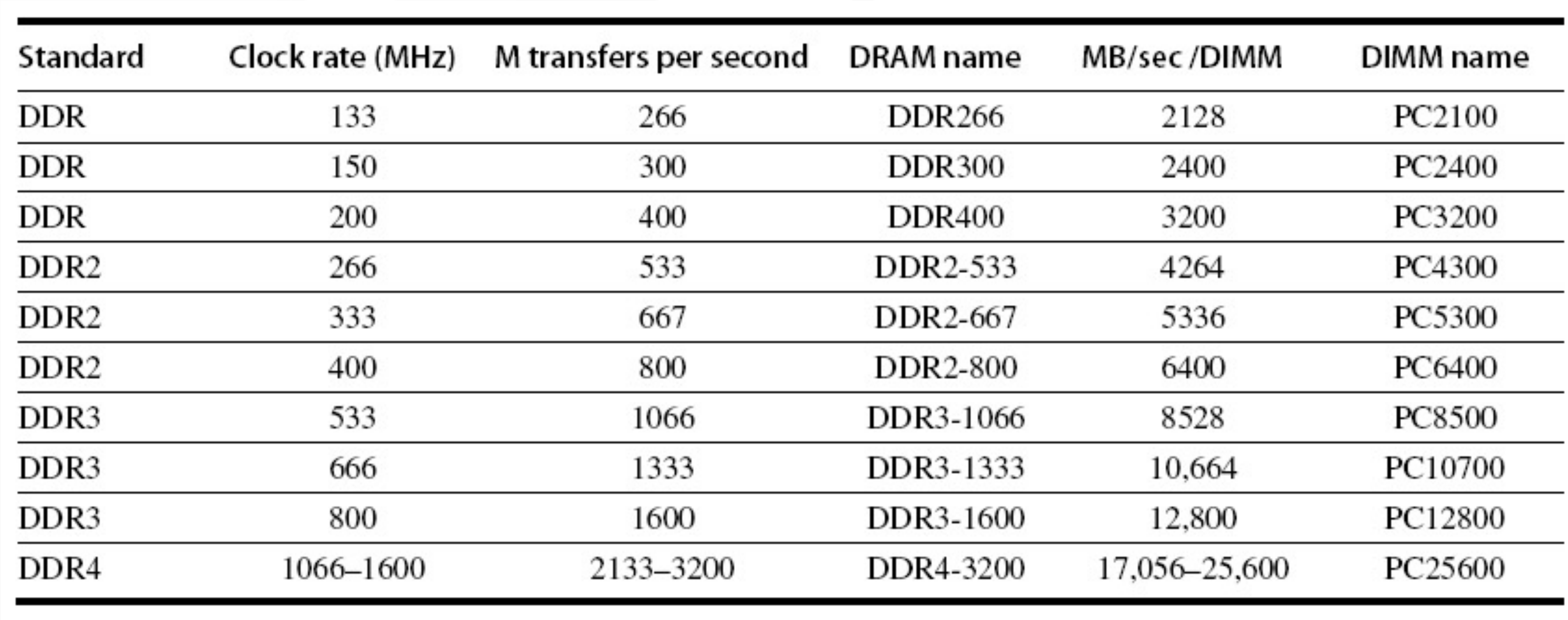


Memory Technology + Optimize



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

Memory Optimizations



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

Memory Optimizations

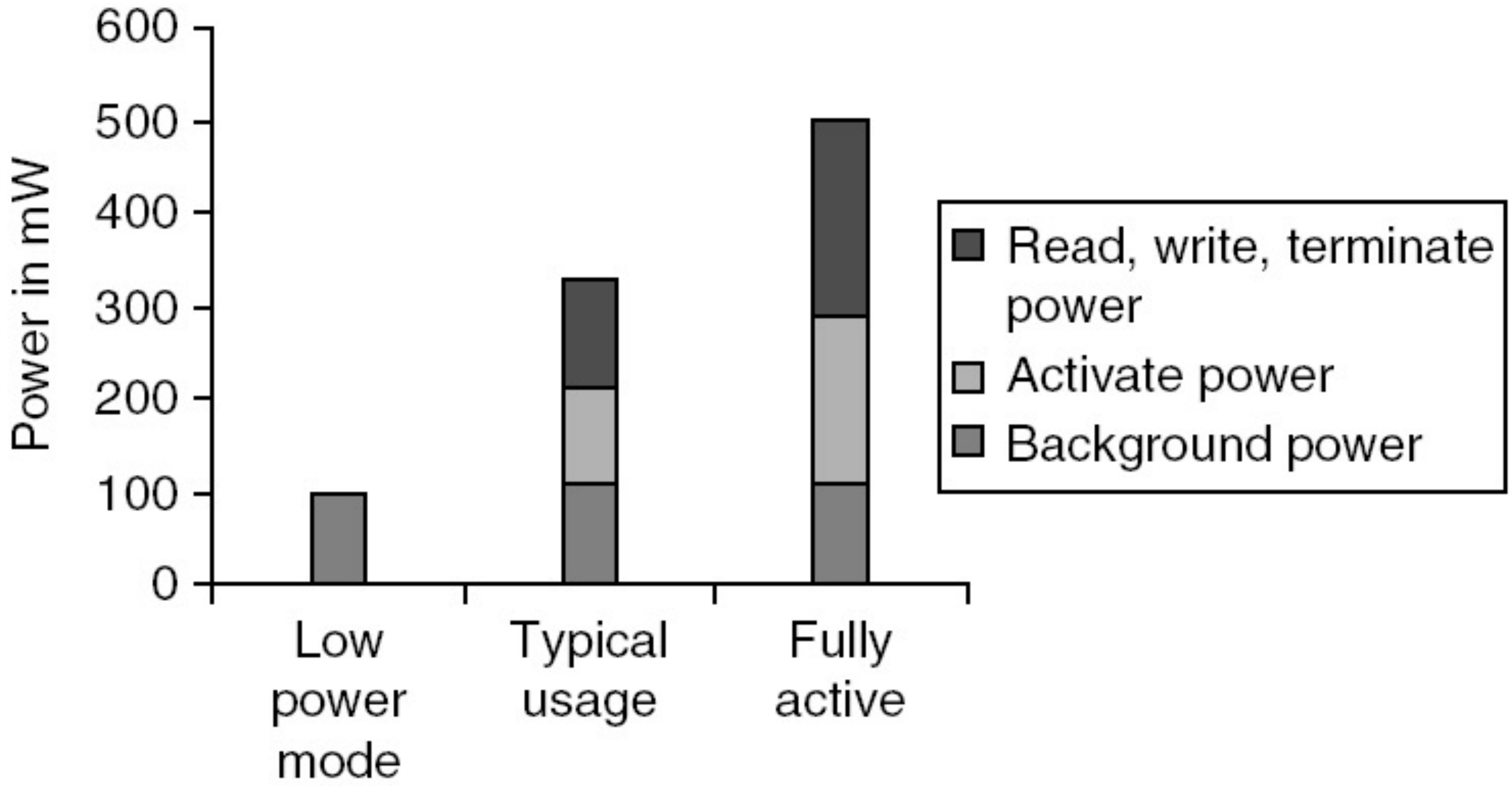
- DDR
  - DDR2
    - Lower power (2.5 V → 1.8 V)
    - Higher clock rates (266 MHz, 333 MHz, 400 MHz)
  - DDR3
    - 1.5 V
    - 800 MHz
  - DDR4
    - 1-1.2 V
    - 1600 MHz
- GDDR5 is graphics memory, based on DDR3
- GDDR6 successor offers increased per-pin bandwidth (up to 16 Gbit/s[3]) and lower operating voltages (1.35 V[4]).

Memory Optimizations

- Graphics Memory
  - Achieve 2x-5x bandwidth per DRAM vs. DDR3
    - Wider interfaces (32 vs. 16 bit)
    - Higher clock rate
      - Possible because they are attached via soldering instead of socketed DIMM modules

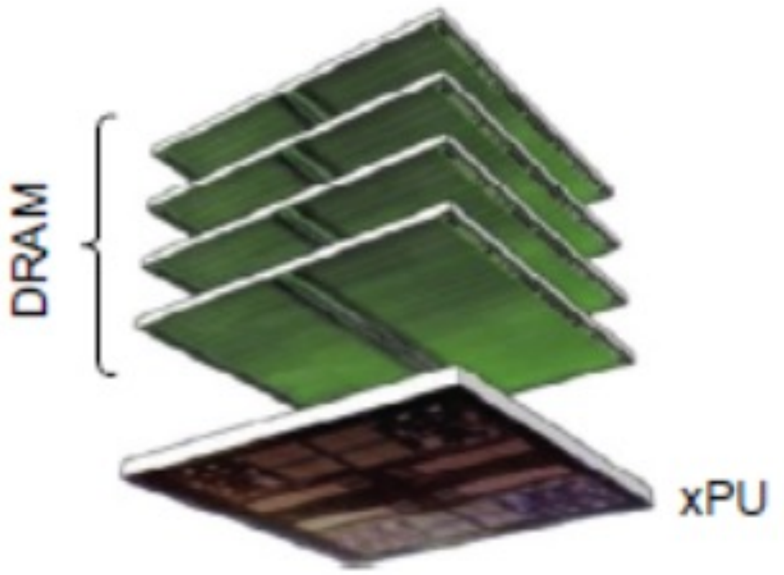
- Reducing Power in SDRAMs
  - Lower voltage
  - Low power mode (ignores clock, continues to refresh)

Memory Power Consumption

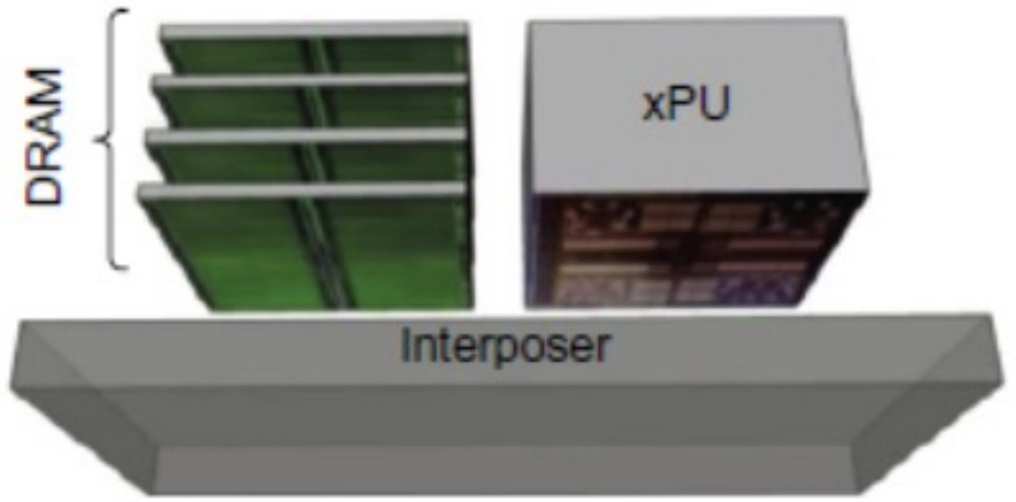


Stacked/Embedded DRAMs

- 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 (0.5 to 4 KiB)
  - 25 µs 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 µs 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/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 sparse rows to replace defective rows

- Chipkill: A RAID-like error-recovery technique

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

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” (VMM) 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

Cache Coherence & Performance

Summary

- Unlike details with pipelining (e.g., ILP) that only concern compiler writers, you the programmer need to acknowledge that cache coherence is going on “under the covers.” Why?

  The coherence protocol can DRAMATICALLY impact your performance!

Impact of Cache Coherence in Multicore CPUs