CPT_S 260 Intro to Computer Architecture
Lecture 40

Memory
April 22, 2022

Ganapati Bhat

School of Electrical Engineering and Computer Science

Washington State University

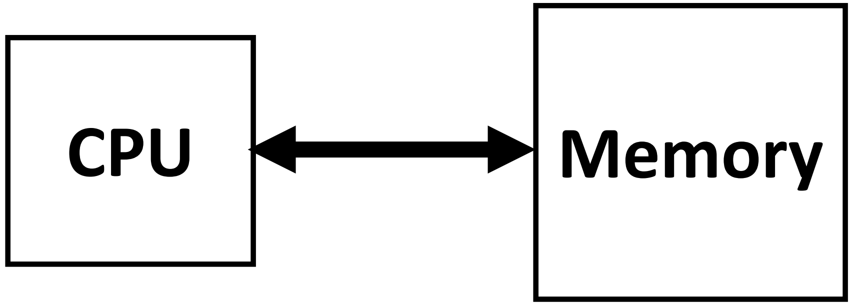
Announcements

- **Final exam**
  - May 4th 2022
  - Comprehensive with all topics
  - Review on Wednesday and Friday
- **Class evaluations are open**
  - Please complete the class review
  - Feedback will help in improving the course in the future
  - Included as part of class participation

Eight Great Ideas in Computer Architecture

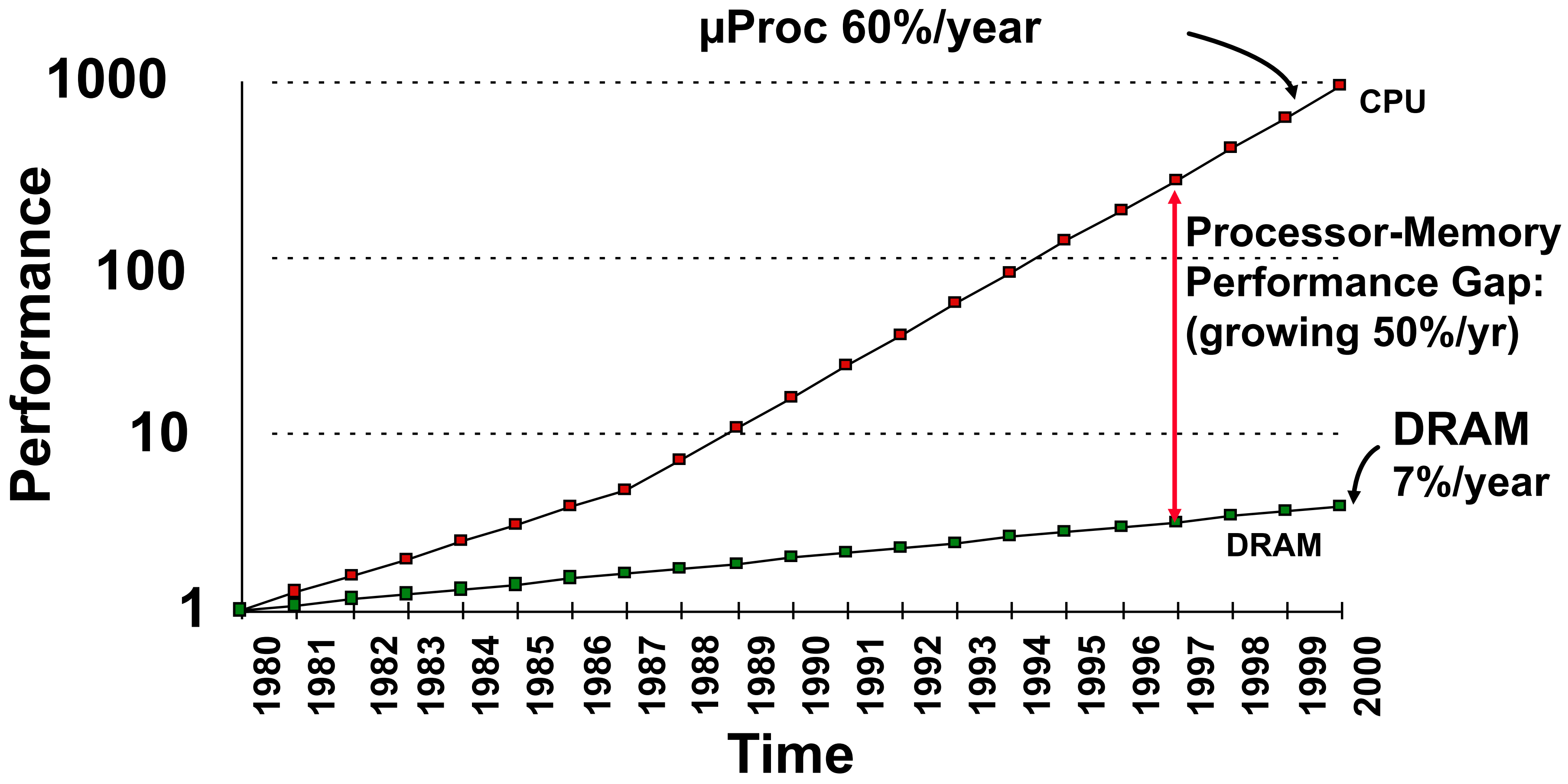
- Design for *Moore's Law*
- Use *abstraction* to simplify design
- Make the *common case fast*
- Performance *via parallelism*
- Performance *via pipelining*
- Performance *via prediction*
- *Hierarchy* of memories
- *Dependability via* redundancy

CPU-Memory Bottleneck



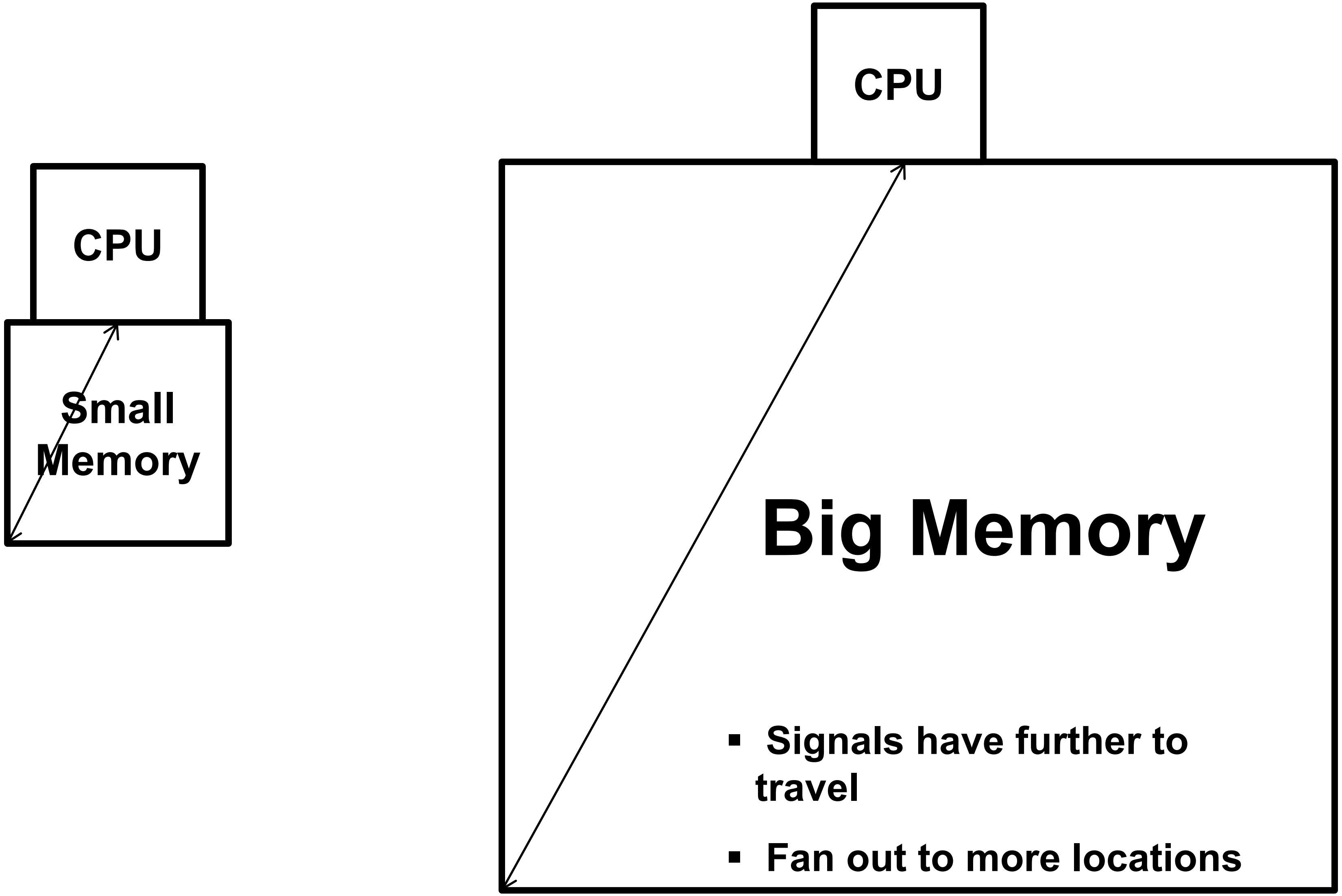
- **Performance of high-speed computers is usually limited by memory bandwidth & latency**
- **Latency (time for a single access)**
  - Memory access time >> Processor cycle time
- **Bandwidth (number of accesses per unit time)**
- **If fraction m of instructions access memory**
  - 1+m memory references / instruction
  - CPI = 1 requires 1+m memory refs / cycle (assuming RISC ISA)
- ***Also, Occupancy (time a memory bank is busy with one request)***

Processor-DRAM Gap (latency)



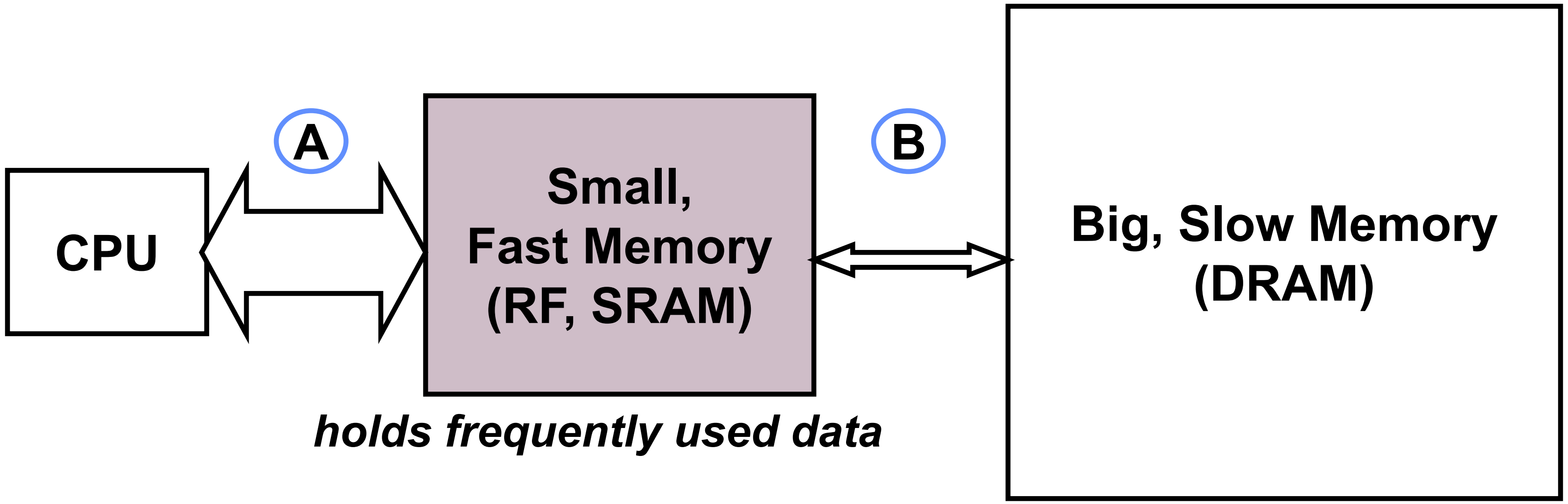
Four-issue 3GHz superscalar accessing 100ns DRAM could execute 1,200 instructions during time for one memory access!

Physical Size Affects Latency



- Signals have further to travel
- Fan out to more locations

Memory Hierarchy



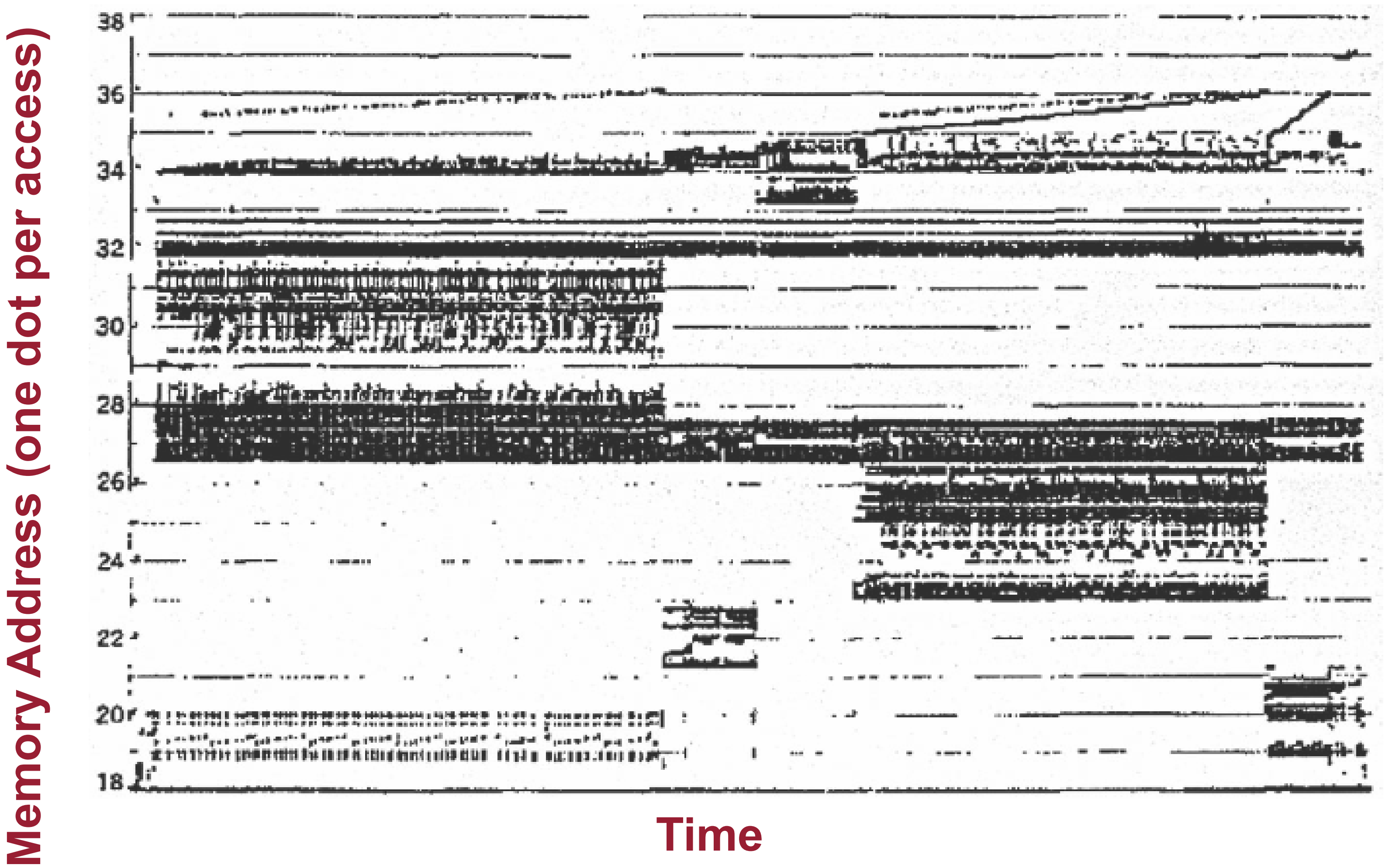
- **Capacity:** **Register << SRAM << DRAM**
- **Latency:** **Register << SRAM << DRAM**
- **Bandwidth:** **on-chip >> off-chip**
- **On a data access**
  - if data $\in$ fast memory $\Longrightarrow$ low latency access (SRAM)
  - if data $\notin$ fast memory $\Longrightarrow$ high latency access (DRAM)

Management of Memory Hierarchy

- **Small/fast storage, e.g., registers**
  - Address usually specified in instruction
  - Generally implemented directly as a register file
    - » *But hardware might do things behind software's back, e.g., stack management, register renaming*

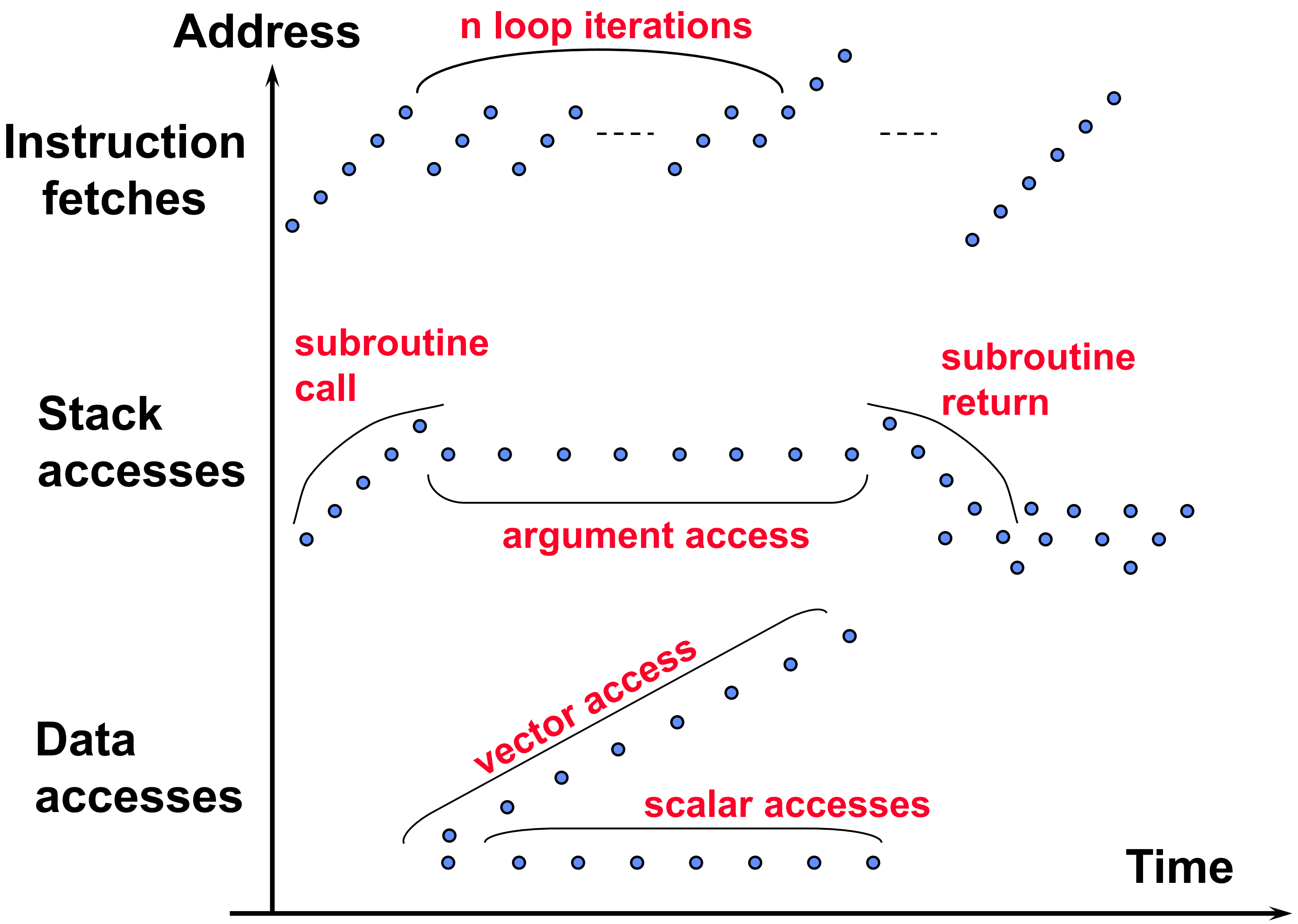
- **Larger/slower storage, e.g., main memory**
  - Address usually computed from values in register
  - Generally implemented as a hardware-managed cache hierarchy (hardware decides what is kept in fast memory)
    - » *But software may provide "hints", e.g., don't cache or prefetch*

Real Memory Reference Patterns



Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Typical Memory Reference Patterns

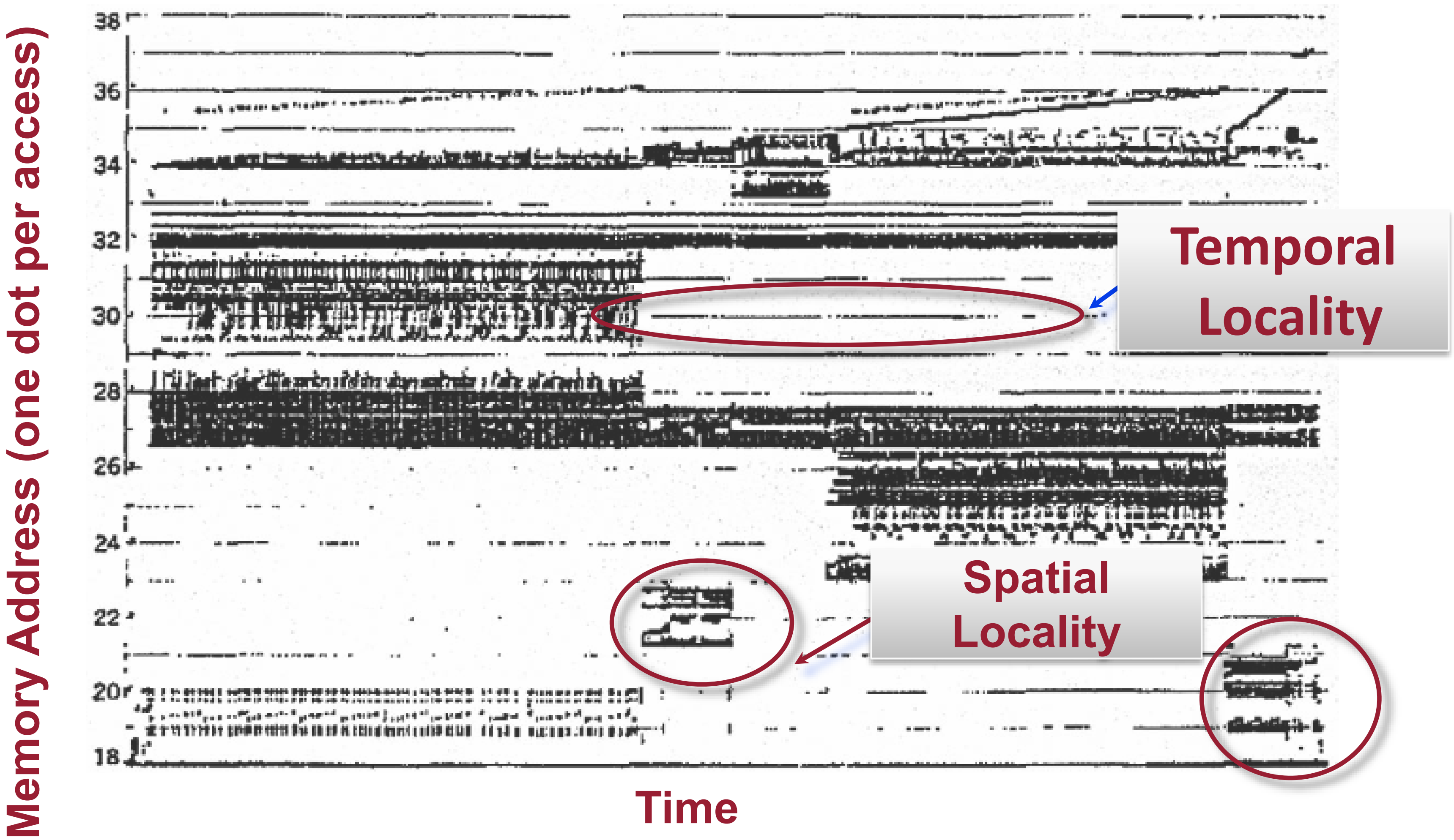


Two Predictable Properties of Memory References

- **Temporal Locality: If a location is referenced it is likely to be referenced again in the near future.**

- **Spatial Locality: If a location is referenced it is likely that locations near it will be referenced in the near future.**

Memory Reference Patterns



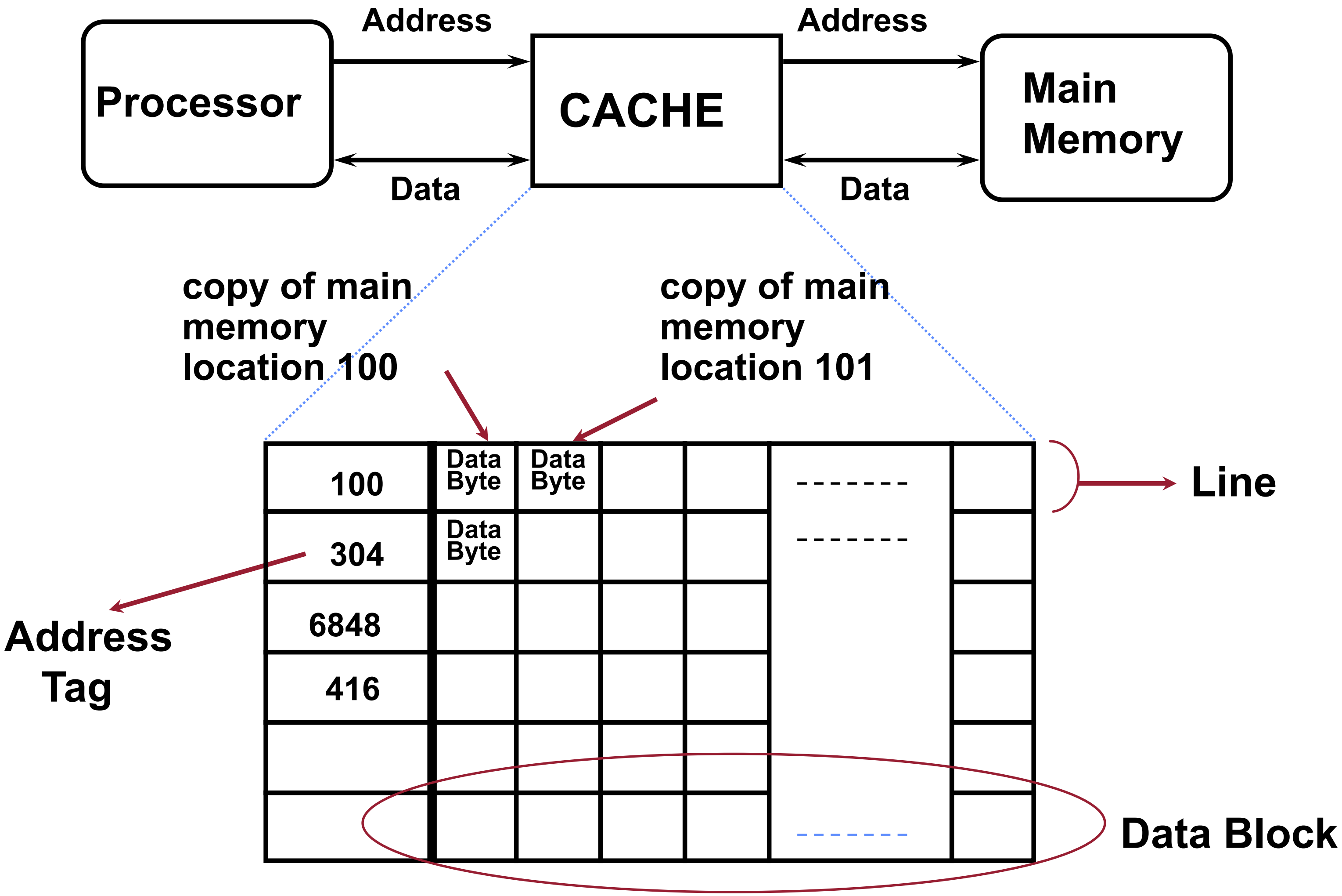
Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Caches Exploit both Types of Predictability

- **Exploit temporal locality by remembering the contents of recently accessed locations.**

- **Exploit spatial locality by fetching blocks of data around recently accessed locations.**

Inside a Cache



Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either

Found in cache
a.k.a. HIT

Return copy of data from cache

Not in cache
a.k.a. MISS

Read block of data from Main Memory

Wait …

Return data to processor and update cache

Q: Which line do we replace?

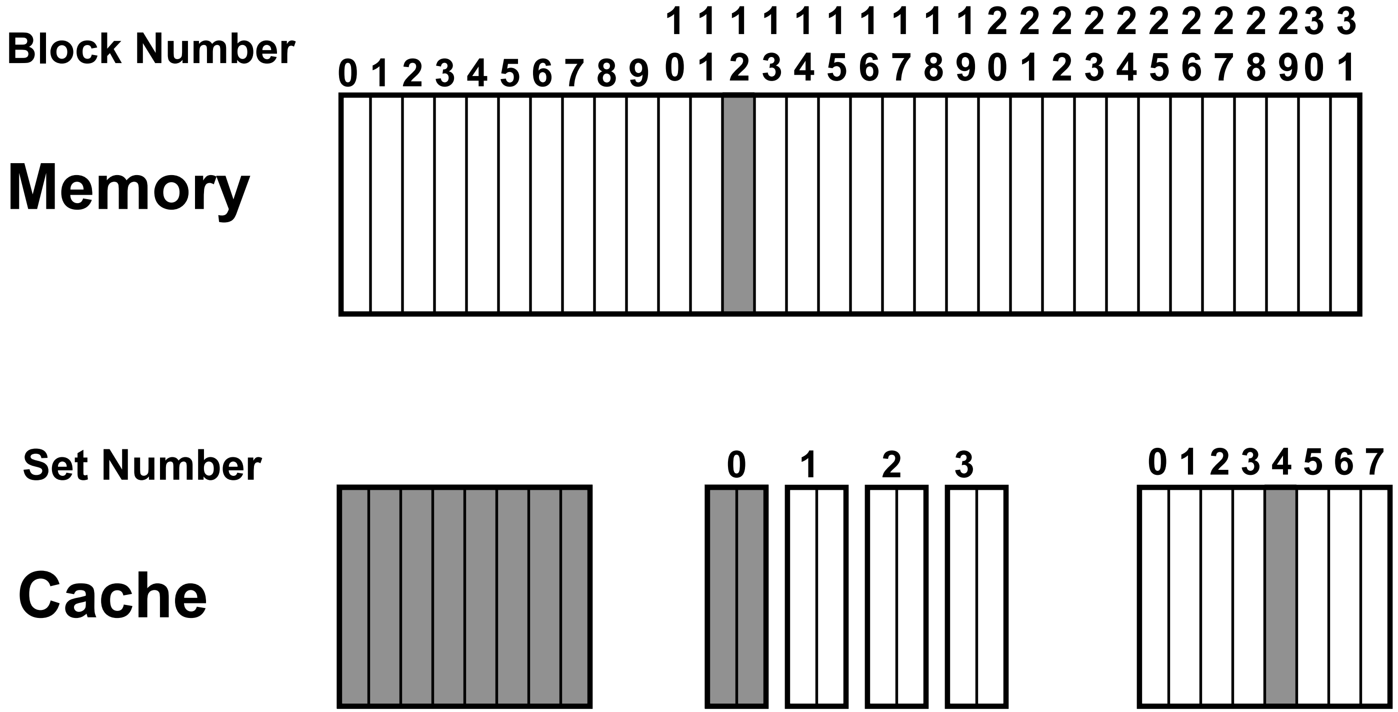
Cache Parameters

- **Cache line**
  - A single entry in the cache that includes the valid bit, tag, and data
- **Block size**
  - The number of bytes of data present in each cache line
- **Sets**
  - A group of blocks in the cache is called a set (applicable for set-associative caches)
- **Associativity**
  - Number of cache blocks in each set of a cache e.g. 2-way set associative
- **Tag**
  - Address identifier used to locate data in a cache

Division of Address in a Cache

- **Block offset**
  - Identifies the byte in a block e.g. 4th byte in a 64 byte block
  - Size is $\log_2 B$ (n)
- **Index**
  - Used to identify the set in a set-associative caches
  - Size is $\log_2 S$ (k)
- **Tag**
  - Size is m – k – n for m-bit memory addresses

Placement Policy



block 12 can be placed	**Fully Associative**	**(2-way) Set Associative**	**Direct Mapped**
	anywhere	anywhere in set 0 *(12 mod 4)*	only into block 4 *(12 mod 8)*

Replacement Policy

- **In an associative cache, which block from a set should be evicted when the set becomes full?**
- **Random**
- **Least-Recently Used (LRU)**
  - LRU cache state must be updated on every access
  - True implementation only feasible for small sets (2-way)
  - Pseudo-LRU binary tree often used for 4-8 way
- **First-In, First-Out (FIFO) a.k.a. Round-Robin**
  - Used in highly associative caches
- **Not-Most-Recently Used (NMRU)**
  - FIFO with exception for most-recently used block or blocks

Replacement only happens on misses

Cache Performance

- **Recall CPU Execution time**

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) * \text{Clock cycle time}$$

- **We need to know memory stall cycles**

$$\begin{aligned}\text{Memory Stall cycles} &= \text{IC} * \frac{\text{Misses}}{\text{Instruction}} * \text{Miss penalty} \\ &= \text{IC} * \frac{\text{Memory acesses}}{\text{Instruction}} * \text{Miss rate} * \text{Miss penalty}\end{aligned}$$

- **We need to measure each of these components to get stall cycles**
- **An approximation since penalty of reads and writes is different**
- **Average memory access time (AMAT) = Hit time + Miss rate x Miss penalty**

Example of Cache Performance

- **Assume CPI = 1 when all memory accesses are hits**
- **50% instructions are loads and stores**
- **Miss rate = 2%, Miss penalty = 25 cycles. Calculate slowdown**

All hits

Execution time = (IC*CPI + 0)* Clock cycle

= IC*1*Clock cycle

With misses

Memory stall cycles = IC*(1+0.5)*0.02*25

= IC * 0.75

Execution time = (IC+IC*0.75)*Clock cycle

= 1.75*IC*Clock cycle

Slowdown = 1.75

Improving Cache Performance

- **AMAT = Hit time + Miss rate * Miss penalty**
- **To improve performance**
  - Reduce the hit time
  - Reduce the miss rate
  - Reduce the miss penalty
- **What is best cache design for 5-stage pipeline?**

Biggest cache that doesn't increase hit time past 1 cycle (approx 8-32KB in modern technology)

[design issues more complex with deeper pipelines and/or out-of-order superscalar processors]

Causes of Cache Misses: The 3 C's

- **Compulsory: First reference to a line (a.k.a. cold start misses)**
  – Misses that would occur even with infinite cache
- **Capacity: Cache is too small to hold all data needed by the program**
  – Misses that would occur even under perfect replacement policy
- **Conflict: Misses that occur because of collisions due to line-placement strategy**
  – Misses that would not occur with ideal full associativity

Effect of Cache Parameters on Performance

- **Larger cache size**
  - \+ Reduces capacity and conflict misses
  - \- Hit time will increase

- **Higher associativity**
  - \+ Reduces conflict misses
  - \- May increase hit time

- **Larger line size**
  - \+ Reduces compulsory and capacity (reload) misses
  - \- Increases conflict misses and miss penalty

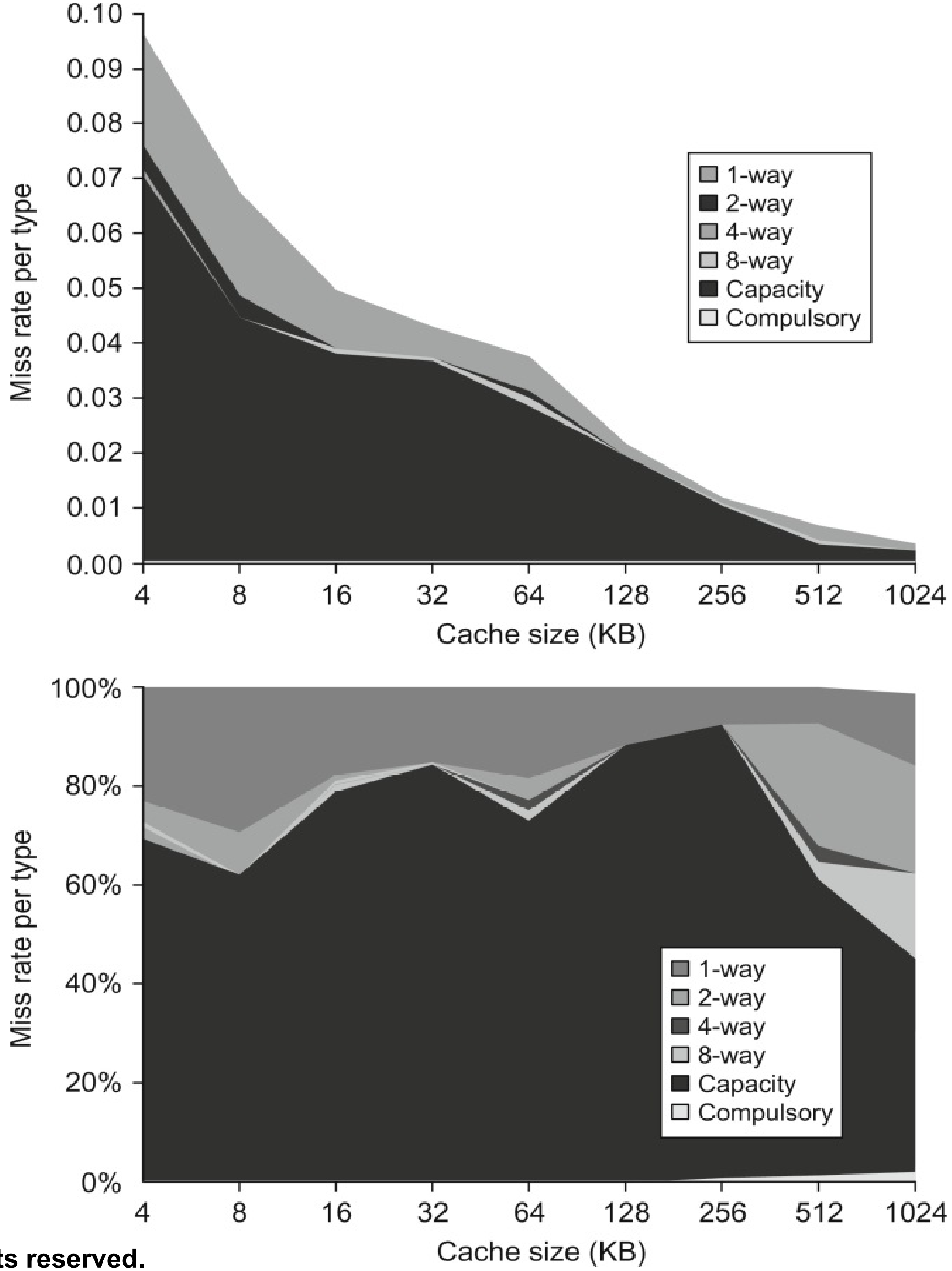
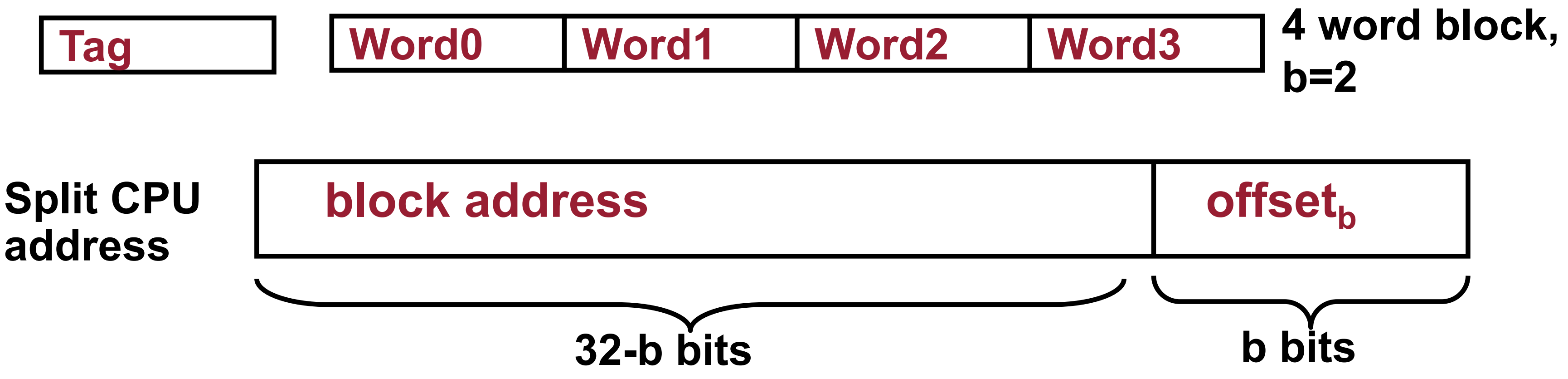


Figure B.9 Total miss rate (top) and distribution of miss rate (bottom) for each size cache according to the three C's for the data in Figure B.8. The top diagram shows the actual data cache miss rates, while the bottom diagram shows the percentage in each category. (*Space allows* the graphs to show one extra cache size than can fit in Figure B.8.)

Block Size and Spatial Locality

- **Recall: Block is unit of transfer between the cache and memory**



2^b = block size *a.k.a* line size (in bytes)

- **Larger block size has distinct hardware advantages**
  - Less tag overhead
  - Exploit fast burst transfers from DRAM
  - Exploit fast burst transfers over wide busses
- **What are the disadvantages of increasing block size?**
  - *Fewer blocks => more conflicts. Can waste bandwidth.*

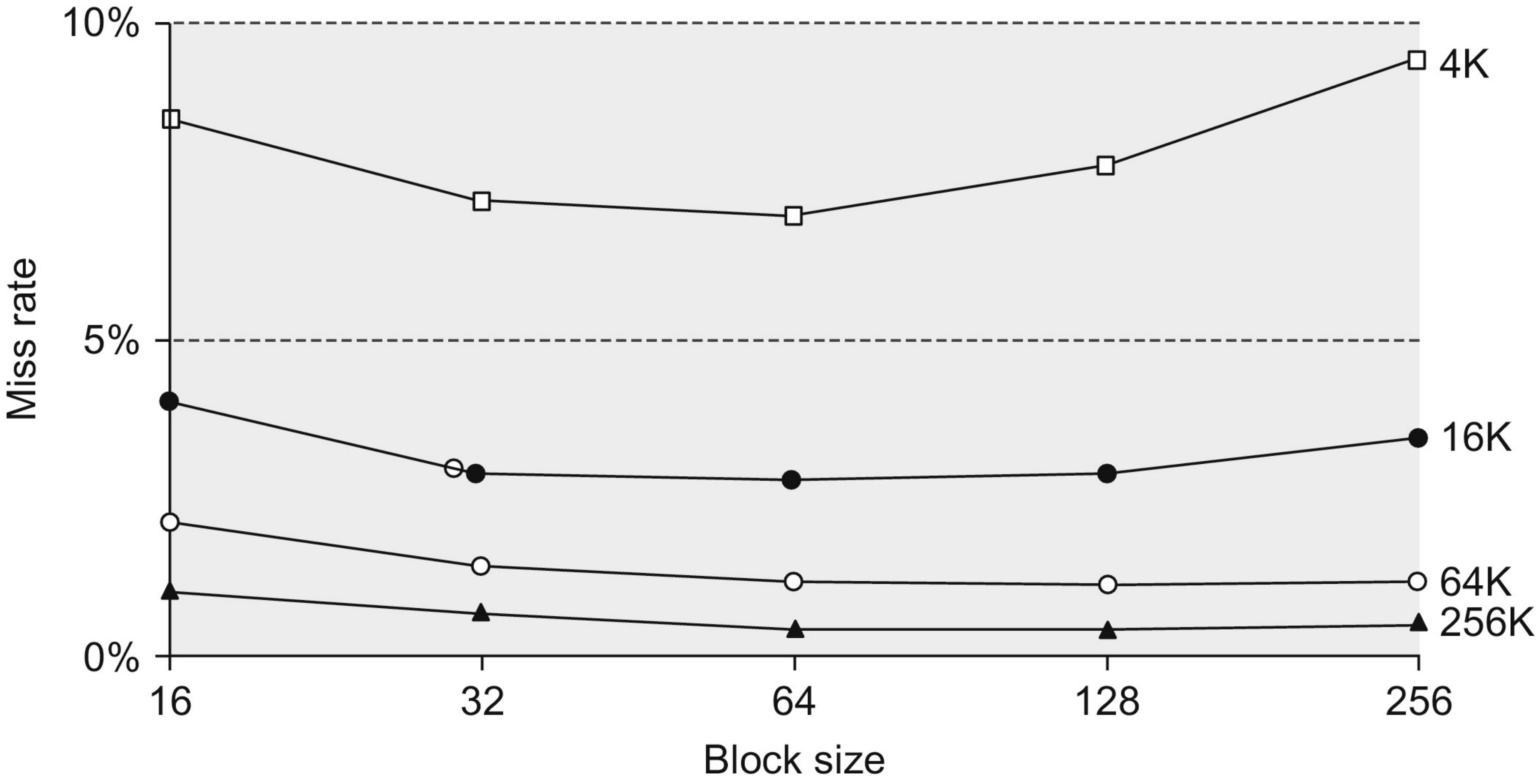
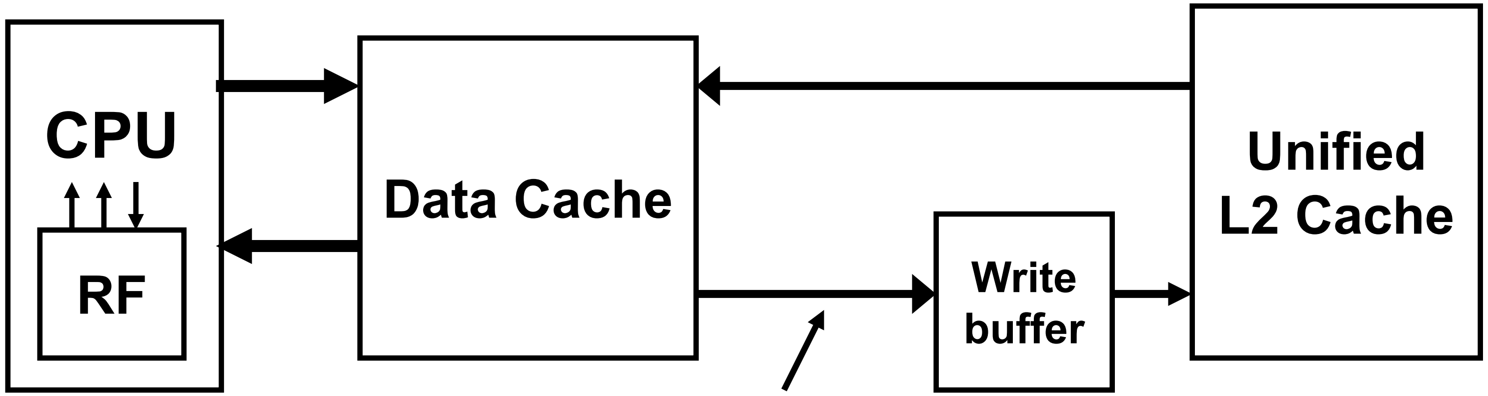


Figure B.10 Miss rate versus block size for five different-sized caches. Note that miss rate actually goes up if the block size is too large relative to the cache size. Each line represents a cache of different size. Figure B.11 shows the data used to plot these lines. Unfortunately, SPEC2000 traces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

Write Policy Choices

- **Cache hit:**
  - ***Write-through***: Write both cache & memory
    - » Generally higher traffic but simpler pipeline & cache design
  - ***Write-back***: Write cache only, memory is written only when the entry is evicted
    - » A dirty bit per line further reduces write-back traffic
    - » Must handle 0, 1, or 2 accesses to memory for each load/store
- **Cache miss:**
  - ***Wo-write-allocate***: Only write to main memory
  - ***Write-allocate*** (aka fetch-on-write): Fetch into cache
- **Common combinations:**
  - Write-through and no-write-allocate
  - Write-back with write-allocate

Write Buffer to Reduce Read Miss Penalty



Evicted dirty lines for write-back cache
OR
All writes in write-through cache

- **Processor is not stalled on writes, and read misses can go ahead of write to main memory**
- **Problem: Write buffer may hold updated value of location needed by a read miss**
- **Simple solution: on a read miss, wait for the write buffer to go empty**
- **Faster solution: Check write buffer addresses against read miss addresses, if no match, allow read miss to go ahead of writes, else, return value in write buffer**