Lecture 10:

Memory Hierarchy:

- **reduce miss-penalty**
- **Reduce hit time**
- **Main memory**

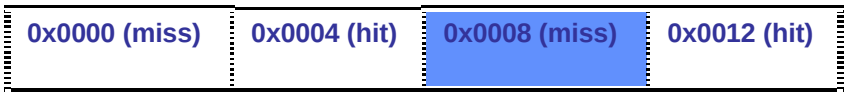
Review: Summary

- 3 Cs: Compulsory, Capacity, Conflict Misses
- Improving cache performance
  - Average memory access time = hit time + miss-rate x miss-penalty
- 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

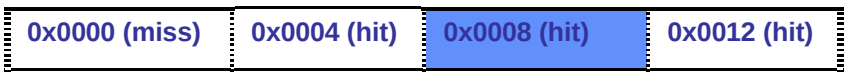
1. Reduce Misses via Larger Block Size

- **Block size ↑ Compulsory misses ↓**
  - **Spatial locality**
  - **Example: access patter 0x0000,0x0004,0x0008,0x0012,..**

Block size = 2 Word



Block size = 4 Word

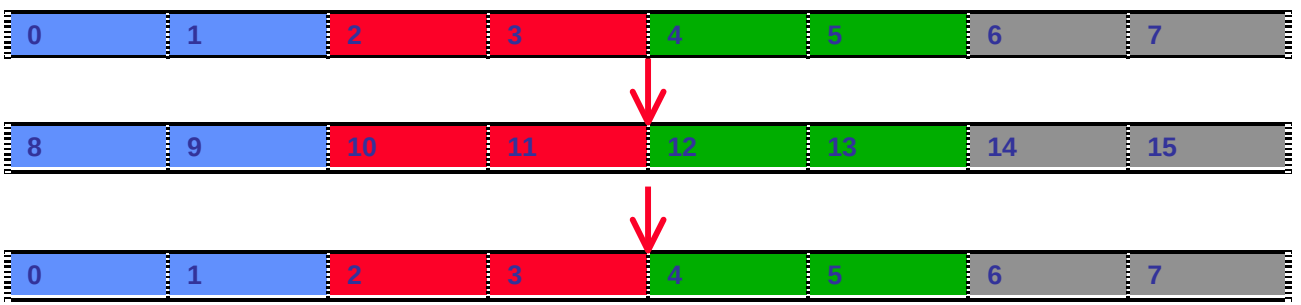


1. Reduce Misses via Larger Block Size

- **Block size ↑ Miss penalty ↑**
- **Larger block size may increase capacity misses & conflict misses**
  - **Example1 (conflict misses):**
    - » **access pattern 0,2,4,6,9,11,13,15,0,2,4,6,…**

Block size = 1B, direct-mapped cache

Block size = 2B, direct--mapped cache

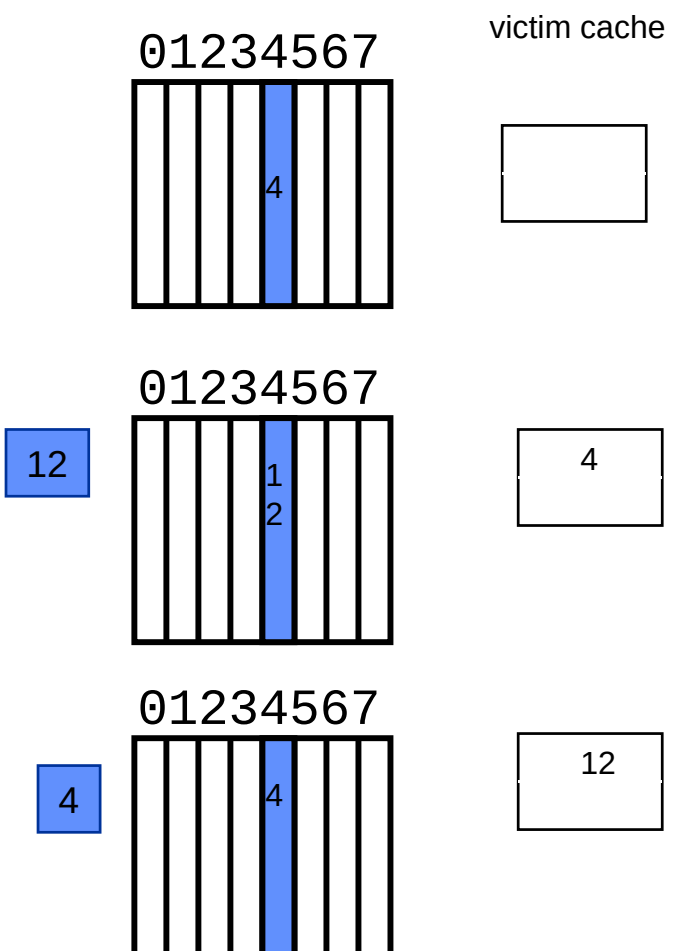


2. Reduce Misses via Higher Associativity

- **2:1 Cache Rule:**
  - **Miss Rate DM cache size N - Miss Rate 2-way cache size N/2**
- **Beware: Execution time is only final measure!**
  - **Will Clock Cycle time increase?**
  - **Hill [1988] suggested hit time external cache +10%, internal + 2% for 2-way vs. 1-way**

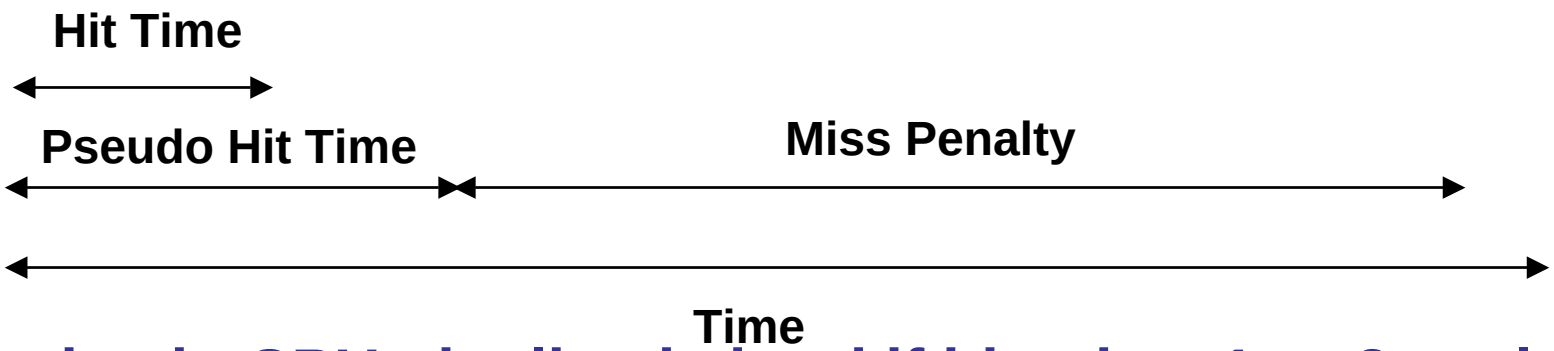
3. Reducing Misses via Victim Cache

- **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**



4. Reducing Misses via Pseudo-Associativity

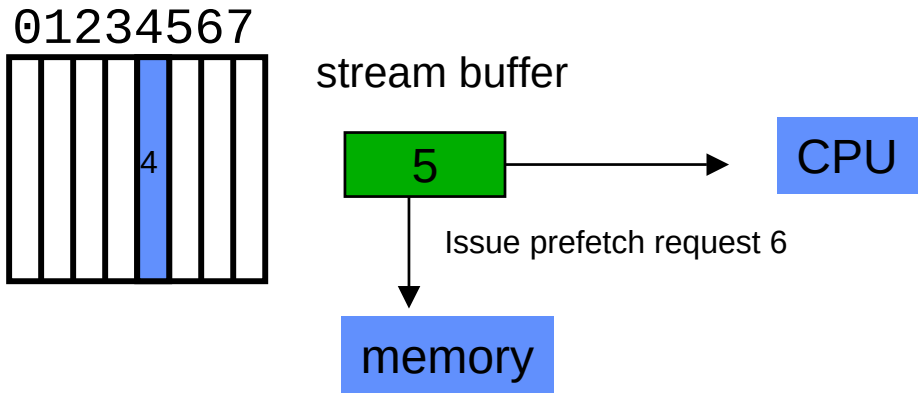
- **How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?**
- **Divide cache: on a miss, check other half of cache to see if there, if so have a pseudo-hit (slow hit)**



- **Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles**
  - **Better for caches not tied directly to processor**

5. Reducing Misses by HW Prefetching of Instruction & Data

- **Bring a cache block up the memory hierarchy before it is requested by the processor**
- **Example: Stream Buffer for instruction prefetching (alpha 21064)**
  - **Alpha 21064 fetches 2 blocks on a miss**
  - **Extra block placed in stream buffer**
  - **On miss check stream buffer**



6. Reducing Misses by SW Prefetching Data

- **Data Prefetch**
  - Compiler insert prefetch instructions to the request the data before they are needed
  - Example:

```
For (i = 0; i< 100; i++)
    prefetch (a[i+4]);
    a[i] = a[i]+ 8
```

  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution
  - Need a non-blocking cache
- **Issuing Prefetch Instructions takes time**
  - Is cost of prefetch issues < savings in reduced misses?

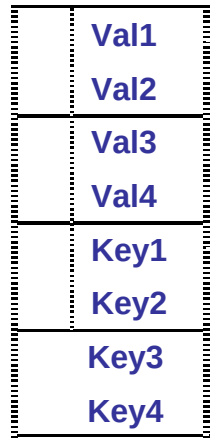
7. Reducing Misses by Compiler Optimizations

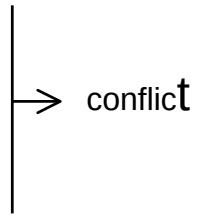
- **Instructions**
  - Reorder procedures in memory so as to reduce misses
  - Profiling to look at conflicts
  - McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache with 4 byte blocks
- **Data**
  - ***Merging Arrays***: improve spatial locality by single array of compound elements 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

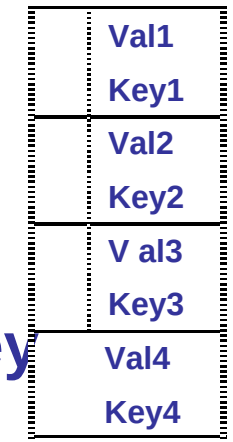
Merging Arrays Example

```
/* Before */
int val[SIZE];
int key[SIZE];

/* After */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];
```



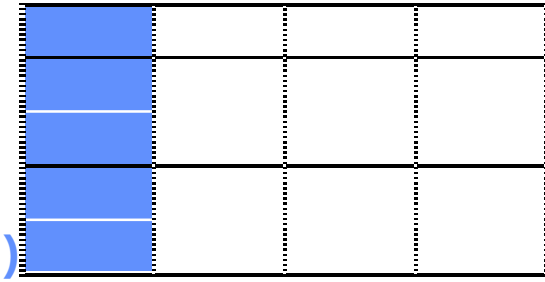


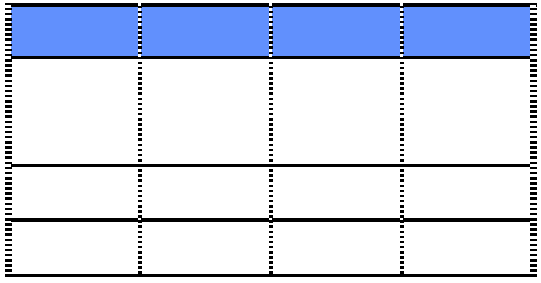


Reducing conflicts between val & key

Loop Interchange Example

```
/* 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 (improve spatial locality)

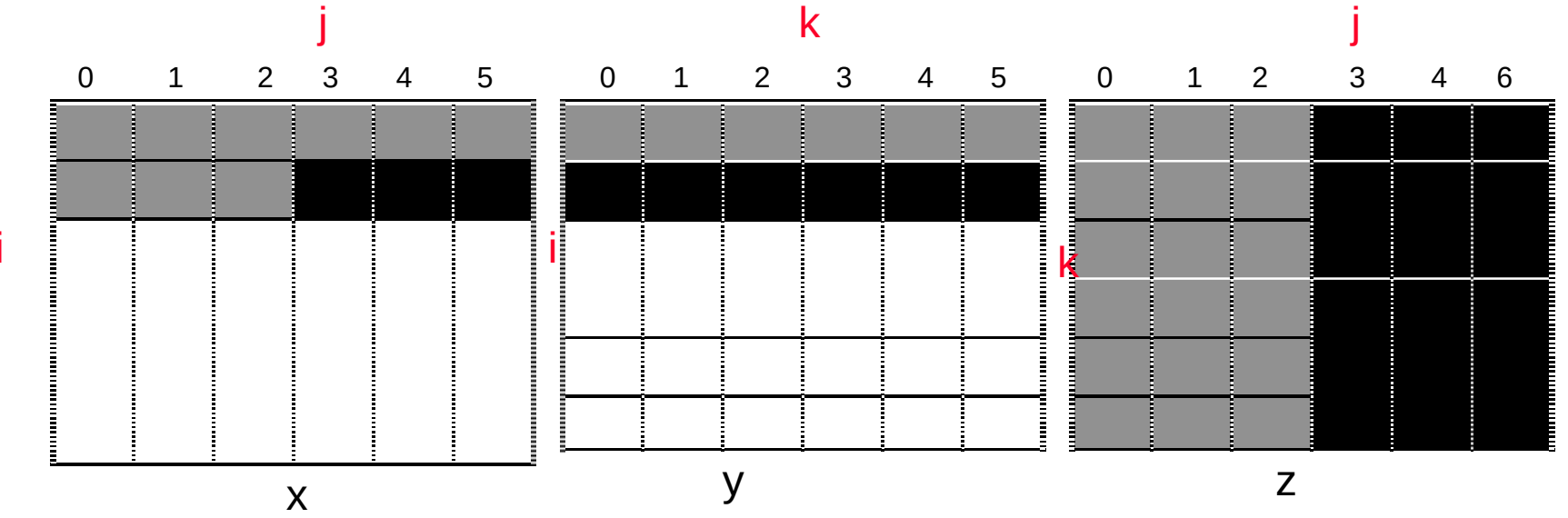
Loop Fusion Example

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
       a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
       d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {    a[i][j] = 1/b[i][j] * c[i][j];
       d[i][j] = a[i][j] + c[i][j];}
```

2 misses per access to a & c vs. one miss per access (improve temporal locality)

Blocking Example

```
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;
       };
```



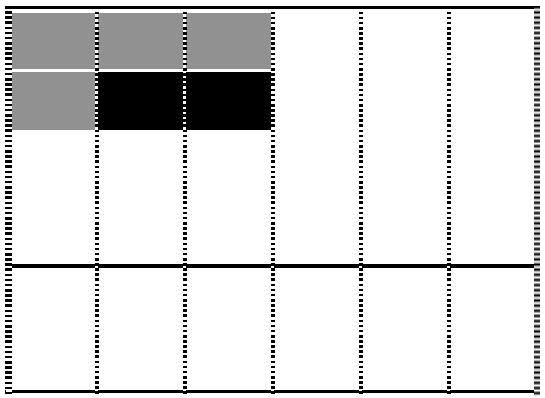
Blocking Example

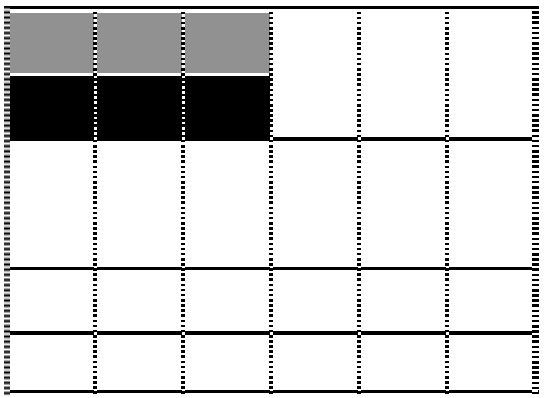
```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
     {r = 0;
      for (k = 0; k < N; k = k+1){
        r = r + y[i][k]*z[k][j];};
      x[i][j] = r;
     };
```

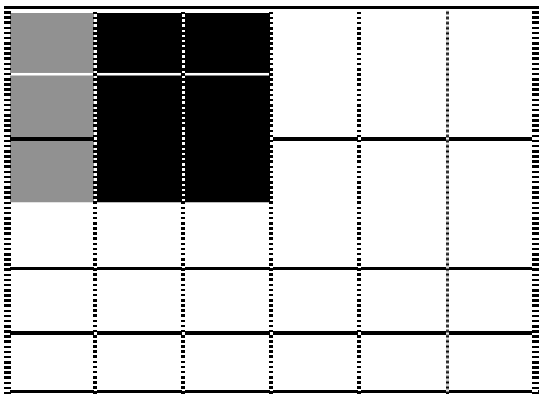
- **Capacity Misses a function of N & Cache Size:**
  - **3 NxN => no capacity misses; otherwise $2N^3 + N^2$**
- **Idea: compute on BxB submatrix that fits**

Blocking Example

```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
   for (j = jj; j < min(jj+B-1,N); j = j+1)
       {r = 0;
        for (k = kk; k < min(kk+B-1,N); k = k+1) {
              r = r + y[i][k]*z[k][j];};
        x[i][j] = x[i][j] + r;
       };
```





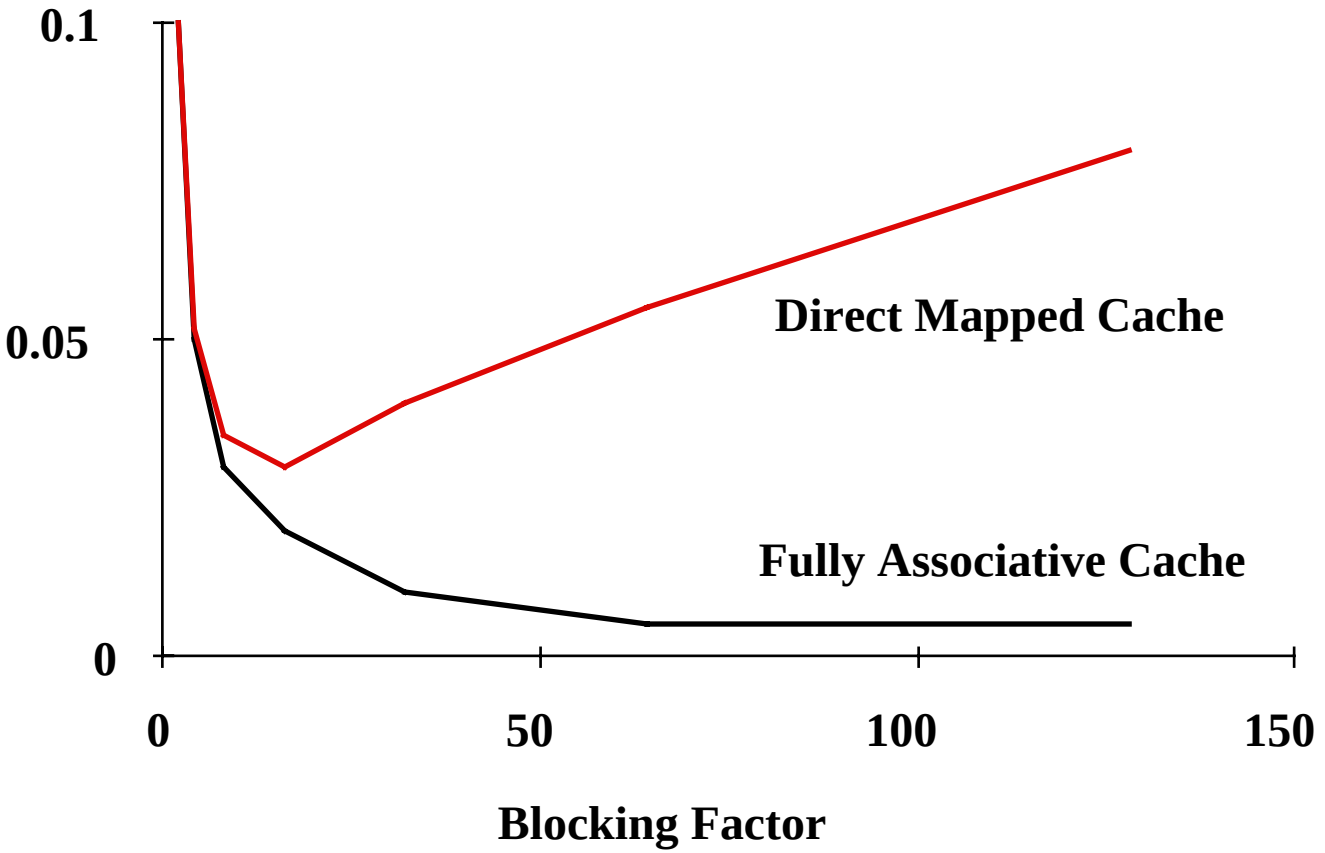


Blocking Example

```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
   for (j = jj; j < min(jj+B-1,N); j = j+1)
     {r = 0;
      for (k = kk; k < min(kk+B-1,N); k = k+1) {
        r = r + y[i][k]*z[k][j];};
      x[i][j] = x[i][j] + r;
     };
```

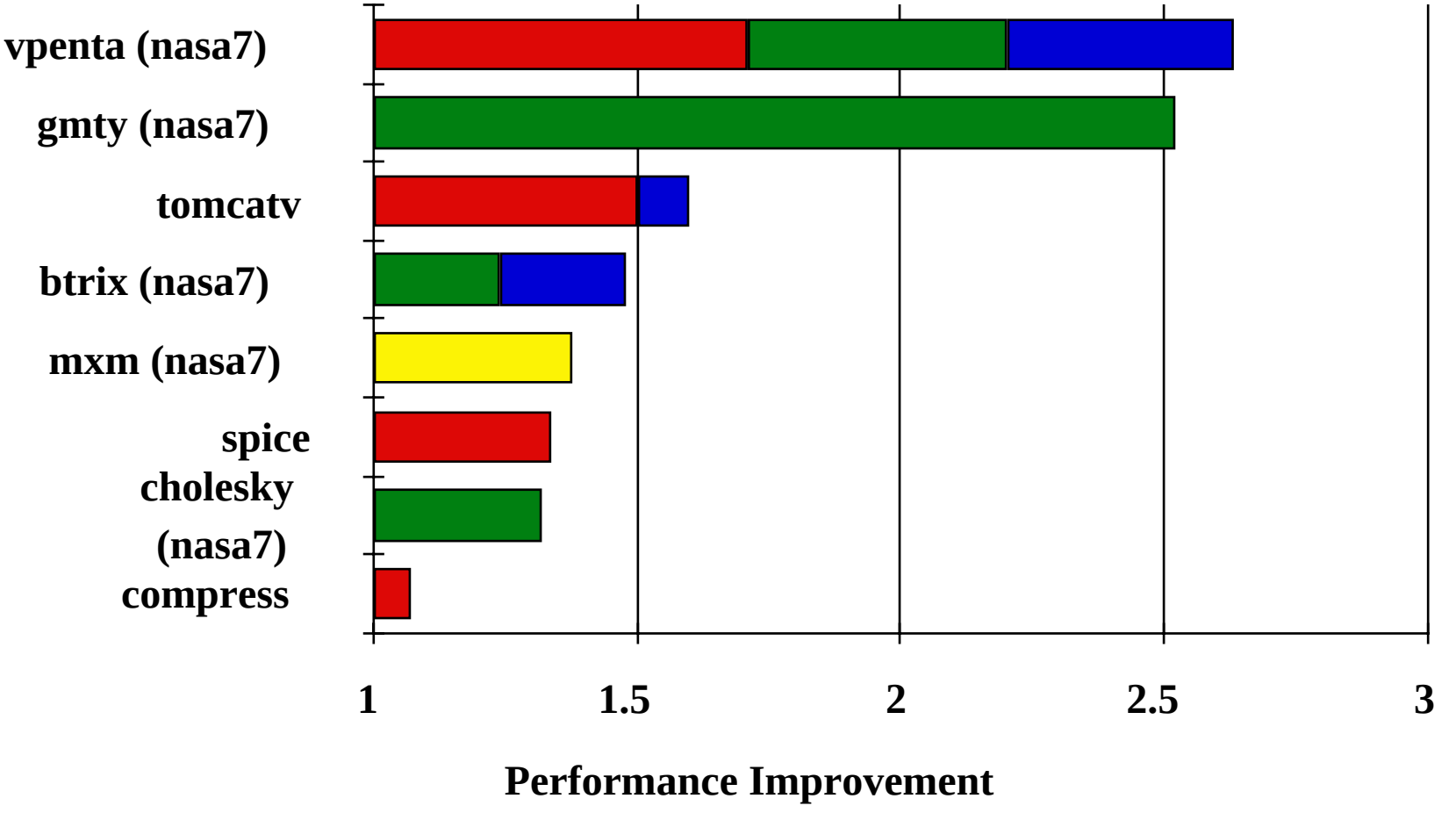
- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
- B called *Blocking Factor*
- Conflict Misses Too?

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

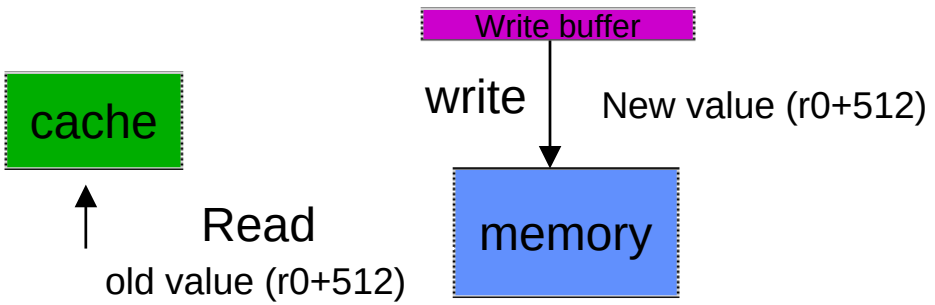


1. Reducing Miss Penalty: Read Priority over Write on Miss

- **Write buffer :**
  - **Decrease write stall time (+)**
  - **RAW Hazard (-)**

```
SW   512(R0), R3        ;  M[512]<- R3          (cache index 0)
LW   R1, 1024(R0)       ; R1 <- M[1024]         (cache index 0)
LW   R2, 512(R0)        ; R2 <- M[512]          (cache index 0)
```

RAW Hazard (assume direct-mapped, write-through cache)

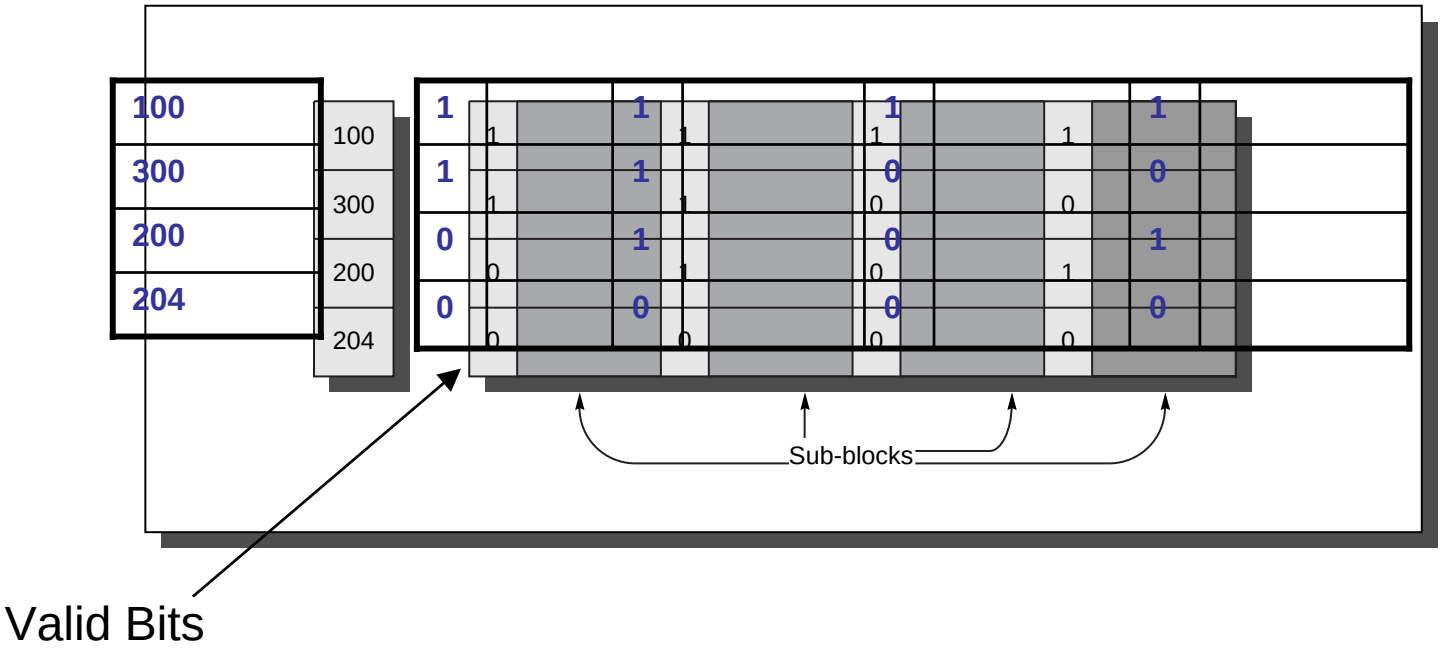


1. Reducing Miss Penalty: Read Priority over Write on Miss

- If simply wait for write buffer to empty might increase read miss penalty by 50% (old MIPS 1000)
- Check write buffer contents before read;
  if no conflicts, let the memory access continue
- How to reduce read-miss penalty for a write-back cache?
  - 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. Subblock Placement to Reduce Miss Penalty

- Don’t have to load full block on a miss
- Have bits per subblock to indicate valid
- (Originally invented to reduce tag storage)



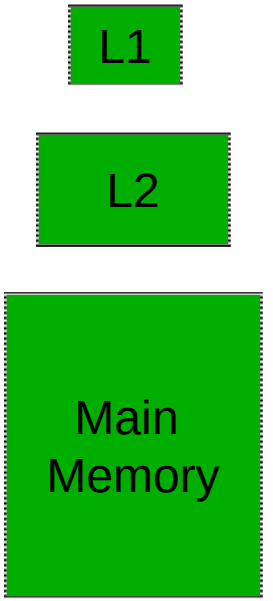
3. Early Restart and Critical Word First

- **Don’t wait for full block to be loaded before restarting CPU**
  - ***Early restart*—**

4. Non-blocking Caches to reduce stalls on misses

- ***Non-blocking cache*** **or** ***lockup-free cache*** **allowing the data cache to continue to supply cache hits during a miss**
- **"*hit under miss*" reduces the effective miss penalty by being helpful during a miss instead of ignoring the requests of the CPU**
- **"*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**

5th Miss Penalty Reduction: Second Level Cache



- **L2 Equations**

  $AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times Miss\ Penalty_{L1}$

  $Miss\ Penalty_{L1} = Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2}$

  $AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times (Hit\ Time_{L2} + Miss\ Rate_{L2} + Miss\ Penalty_{L2})$

- **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*** ($Miss\ Rate_{L1} \times Miss\ Rate_{L2}$)

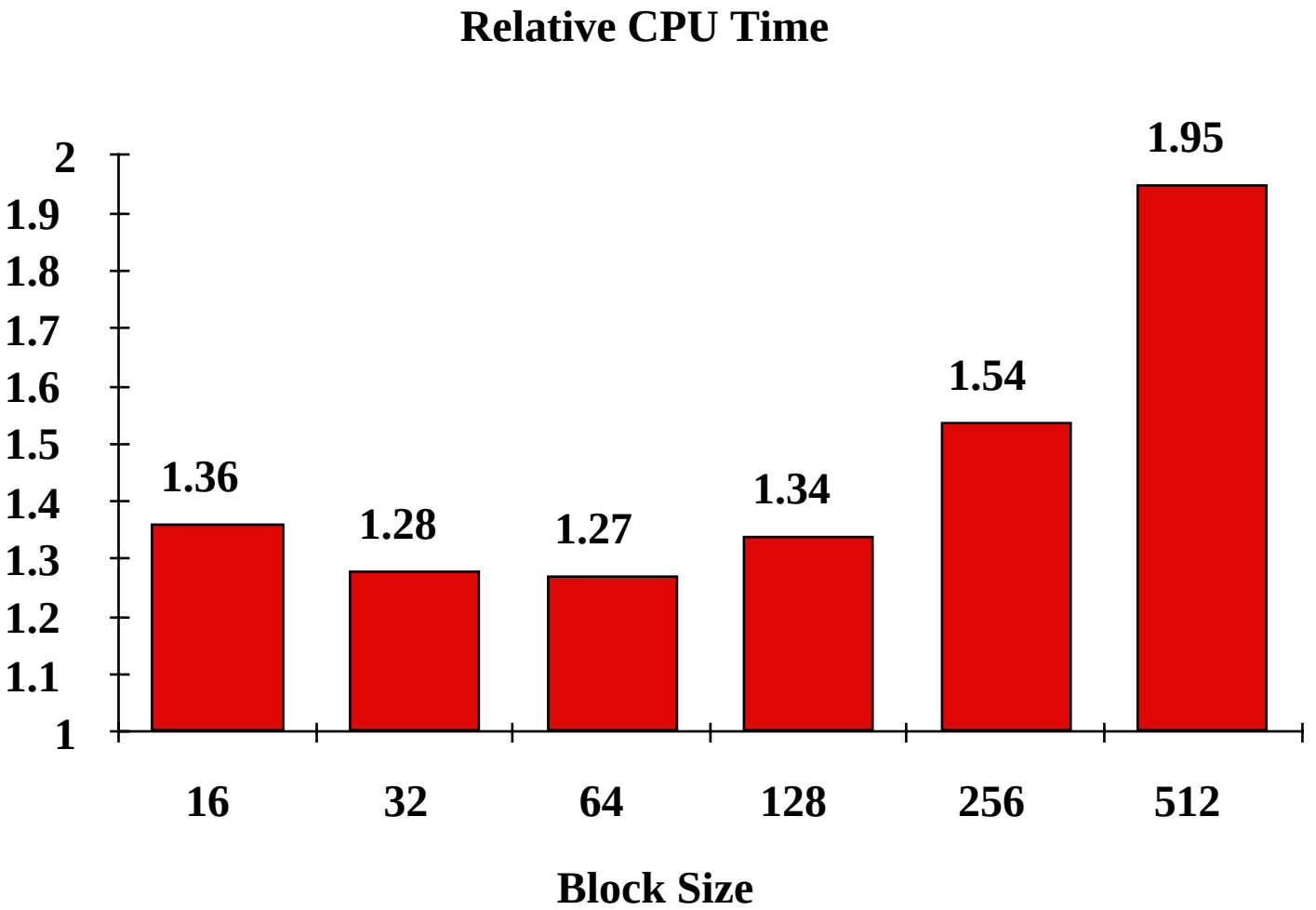
Comparing Local and Global Miss Rates Figure 5.23

- 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 Cache Design Principle

- **L2 not tied to CPU clock cycle**
- **Consider Cost & A.M.A.T.**
- **Generally Fast Hit Times and fewer misses**
- **Since hits are few, target miss reduction**
  - **Larger cache, higher associativity and larger blocks**

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



- 32KB L1, 8 byte path to memory

Reducing Miss Penalty Summary

- **Five techniques**
  - **Read priority over write on miss**
  - **Subblock placement**
  - **Early Restart and Critical Word First on miss**
  - **Non-blocking Caches (Hit 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**

Review: Improving Cache Performance

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

1. Fast Hit times via Small and Simple Caches

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

2. Fast hits by Avoiding Address Translation

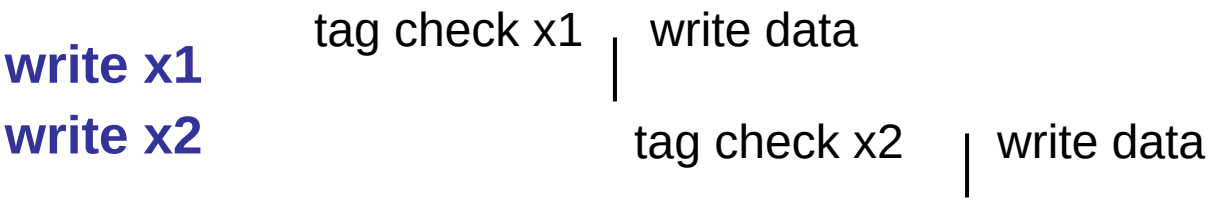
- **Address translation – from virtual to physical addresses**
- **Physical cache:**
  - **1. Virtual -> physical**
  - **2. Use physical address to index cache => longer hit time**
- **Virtual cache:**
  - **Use virtual cache to index cache => shorter hit time**
  - **problem: aliasing**

More on this after covering virtual memory issues !

3. Fast Hit Times Via Pipelined Writes

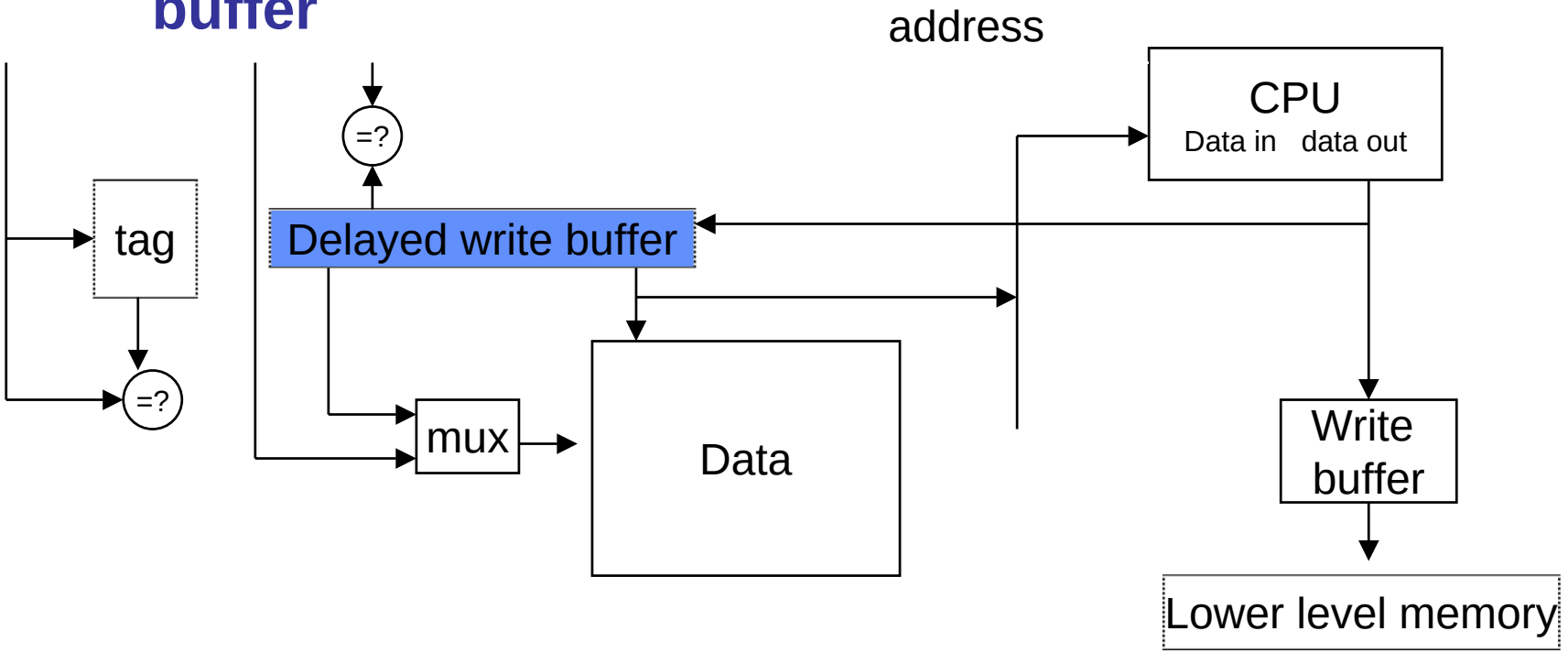
Pipeline Tag Check and Update Cache as separate stages; current write tag check & previous write cache update

Only Write in the pipeline; empty during a miss



3. Fast Hit Times Via Pipelined Writes

- **Delayed Write Buffer: must be checked on reads; either complete write or read from buffer**



4. Fast Writes on Misses Via Small Subblocks

- **If most writes are 1 word, subblock size is 1 word, & write through then always write subblock & valid bit immediately**
  - ***Tag match and valid bit already set*: Writing the block was proper, & nothing lost by setting valid bit on again.**
  - ***Tag match and valid bit not set*: The tag match means that this is the proper block; writing the data into the subblock makes it appropriate to turn the valid bit on.**
  - ***Tag mismatch*: This is a miss and will modify the data portion of the block. As this is a write-through cache, however, no harm was done; memory still has an up-to-date copy of the old value. Only the tag to the address of the write and the valid bits of the other subblock need be changed because the valid bit for this subblock has already been set**
- **Doesn't work with write back due to last case**

Cache Optimization Summary

Technique	*MR*	*MP*	*HT*	*Complexity*
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
Priority to Read Misses		+		1
Subblock Placement		+		1
Early Restart & Critical Word 1st		+		2
Non-Blocking Caches		+		3
Second Level Caches		+		2
Small & Simple Caches	–		+	0
Avoiding Address Translation			+	2
Pipelining Writes			+	1

Main Memory

Main Memory Background

Performance of Main Memory:

- Latency: time to finish a request
  - *Access Time*: time between request and word arrives
  - *Cycle Time*: time between requests
  - Cycle time > Access Time
- Bandwidth: Bytes/per second

Main Memory is *DRAM*: Dynamic Random Access Memory

- Dynamic since needs to be refreshed periodically (8 ms)
- Addresses divided into 2 halves (Memory as a 2D matrix):
  - *RAS* or *Row Access Strobe*
  - *CAS* or *Column Access Strobe*

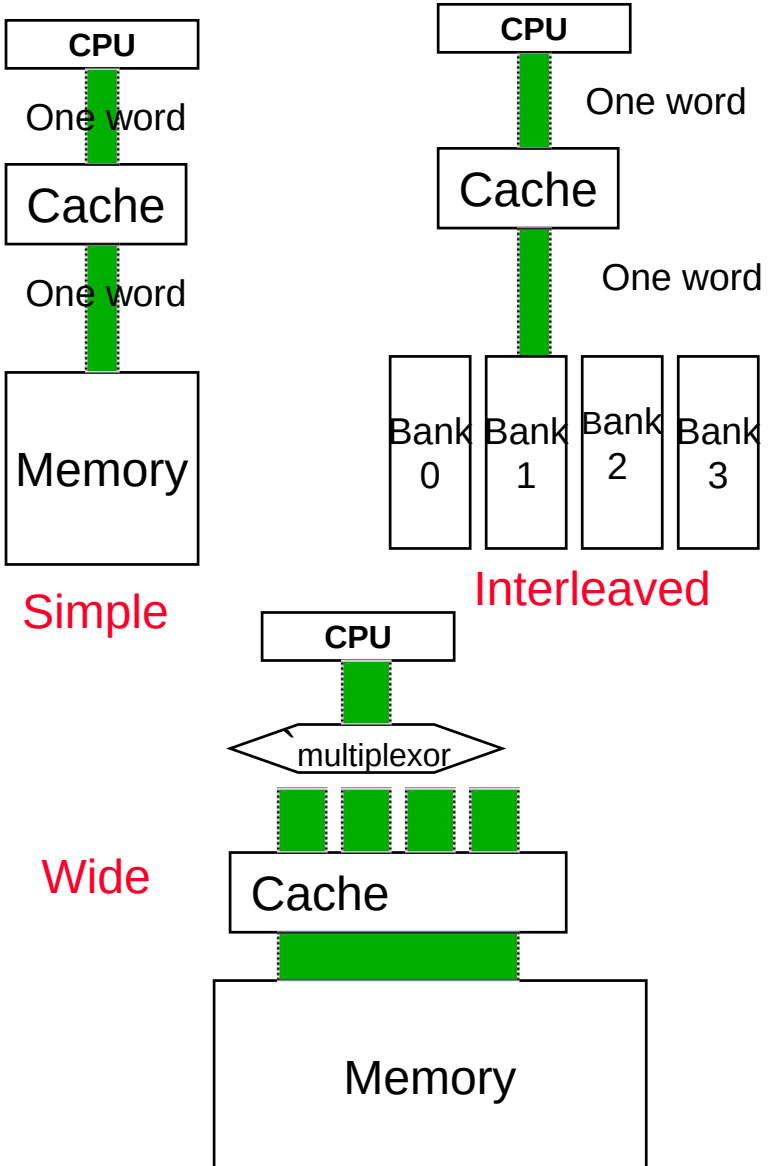
Cache uses *SRAM*: Static Random Access Memory

- No refresh (6 transistors/bit vs. 1 transistor /bit)
- Address not divided

Size: DRAM/SRAM - ***4-8***,
Cost/Cycle time: SRAM/DRAM - ***8-16***

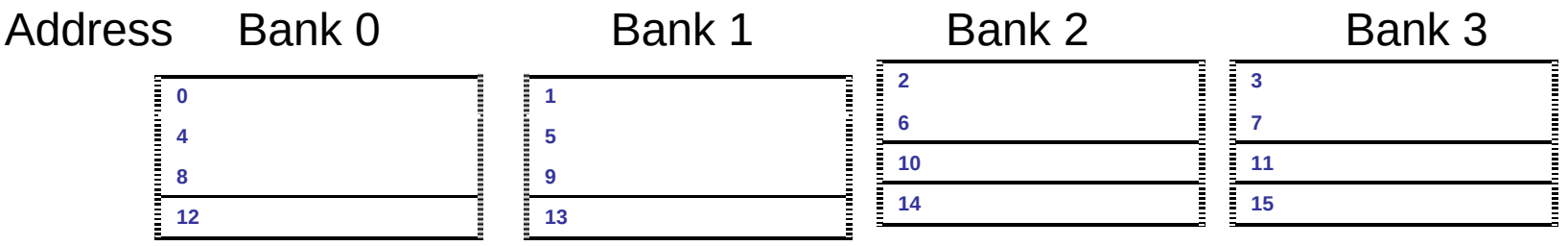
Main Memory Performance

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



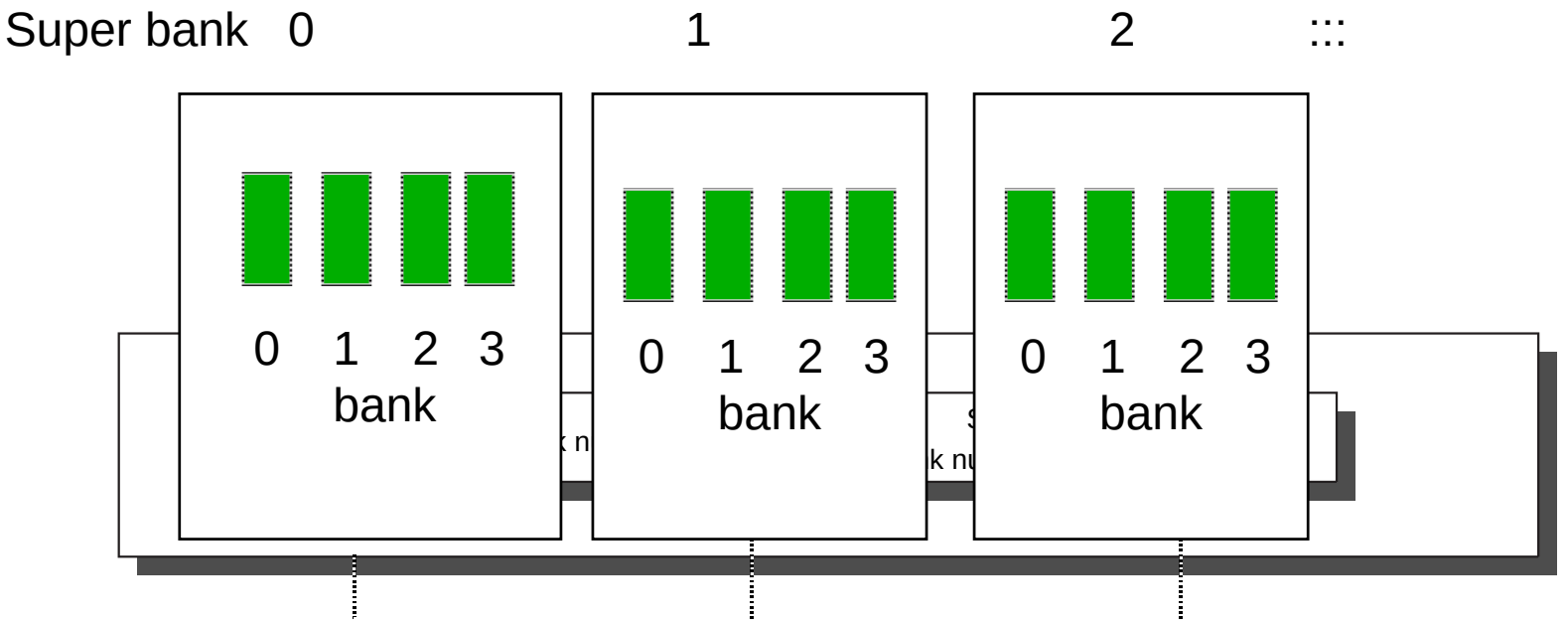
Main Memory Performance

- **Timing model**
  - **4 clock cycles to send address,**
  - **24 clock cycles for access time per word**
  - **4 clock cycles to send x (bus bandwidth) words of data**
  - **Cache block size = 4 words**
- ***Simple M.P.*** **= 4 x (4+24+4) = 128**
- ***Wide M.P.*** **= 4 + 24 + 4 = 32**
- ***Interleaved M.P.*** **= 4+ 24 + 4x4 = 44**



Independent Memory Banks

- **Memory banks for independent accesses vs. faster sequential accesses**
  - **Multiprocessor**
  - **I/O**
  - **Miss under Miss, Non-blocking Cache**



Avoiding Bank Conflicts

- **Bank Conflicts**
  - Memory references map to the same bank
  - Problem: cannot take advantage of multiple banks (supporting multiple independent request)
- **Example: all elements of a column are in the same memory bank with 128 memory banks, interleaved on a word basis**

```
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];
```

- **SW: loop interchange or declaring array not power of 2**
- **HW: Prime number of banks**
  - **Problem: more complex calculation per memory access**
  - **Memory address = (bank number, address within bank)**
  - **bank number = address mod number of banks**
  - **address within bank = address / number of banks**
  - **modulo & divide per memory access**

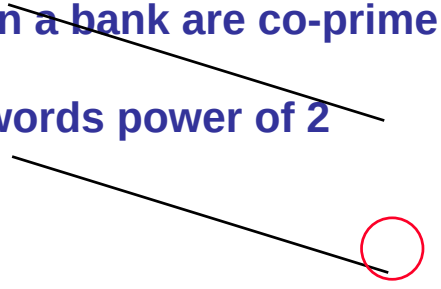
Fast Bank Number

- **Bank number = address mod number of banks**
- **Address within bank = address mod number words in bank**
  => prove that there is no ambiguity using this mapping method
- **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 (b0 = x mod a0)**
- **address within bank = b_1, number of words in bank = a_1 (b1 = x mod a1)**
- **Bank number < Number of banks (0≤b0≤a0)**
- **Address within a bank < Number of words in bank (0≤b1≤ a1)**
- **Address < Number of banks x Number of words in a bank (0 ≤x<a0xa1)**
- **The number of banks and the number of words in a bank are co-prime (a0 and a1 are co-prime)**
  - **N word address 0 to N-1, prime no. banks, words power of 2**



Fast Bank Number

		Seq. Interleaved			Modulo Interleaved		
BankNumber:	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	

Fast Memory Systems: DRAM specific

- **Multiple RAS accesses: several names (page mode)**
  - **64 Mbit DRAM: cycle time = 100 ns, page mode = 20 ns**
- **New DRAMs to address gap; what will they cost, will they survive?**
  - ***Synchronous DRAM*: Provide a clock signal to DRAM, transfer synchronous to system clock**
  - ***RAMBUS*: 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)**
- **Niche memory**
  - **e.g., Video RAM for frame buffers, DRAM + fast serial output**

Main Memory Summary

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