CS 152 Computer Architecture and Engineering
CS252 Graduate Computer Architecture

Lecture 7 – Memory III

Krste Asanovic
Electrical Engineering and Computer Sciences
University of California at Berkeley

http://www.eecs.berkeley.edu/~krste
http://inst.eecs.berkeley.edu/~cs152

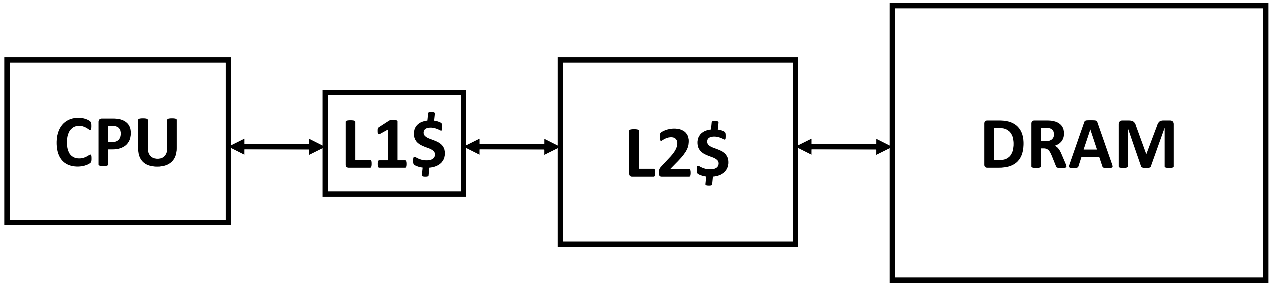
Last time in Lecture 6

- 3 C’s of cache misses
  - Compulsory, Capacity, Conflict
- Write policies
  - Write back, write-through, write-allocate, no write allocate
- Pipelining write hits
- Multi-level cache hierarchies reduce miss penalty
  - 3 levels common in modern systems (some have 4!)
  - Can change design tradeoffs of L1 cache if known to have L2
  - Inclusive versus exclusive cache hierarchies

Recap: Multilevel Caches

Problem: A memory cannot be large and fast

Solution: Increasing sizes of cache at each level

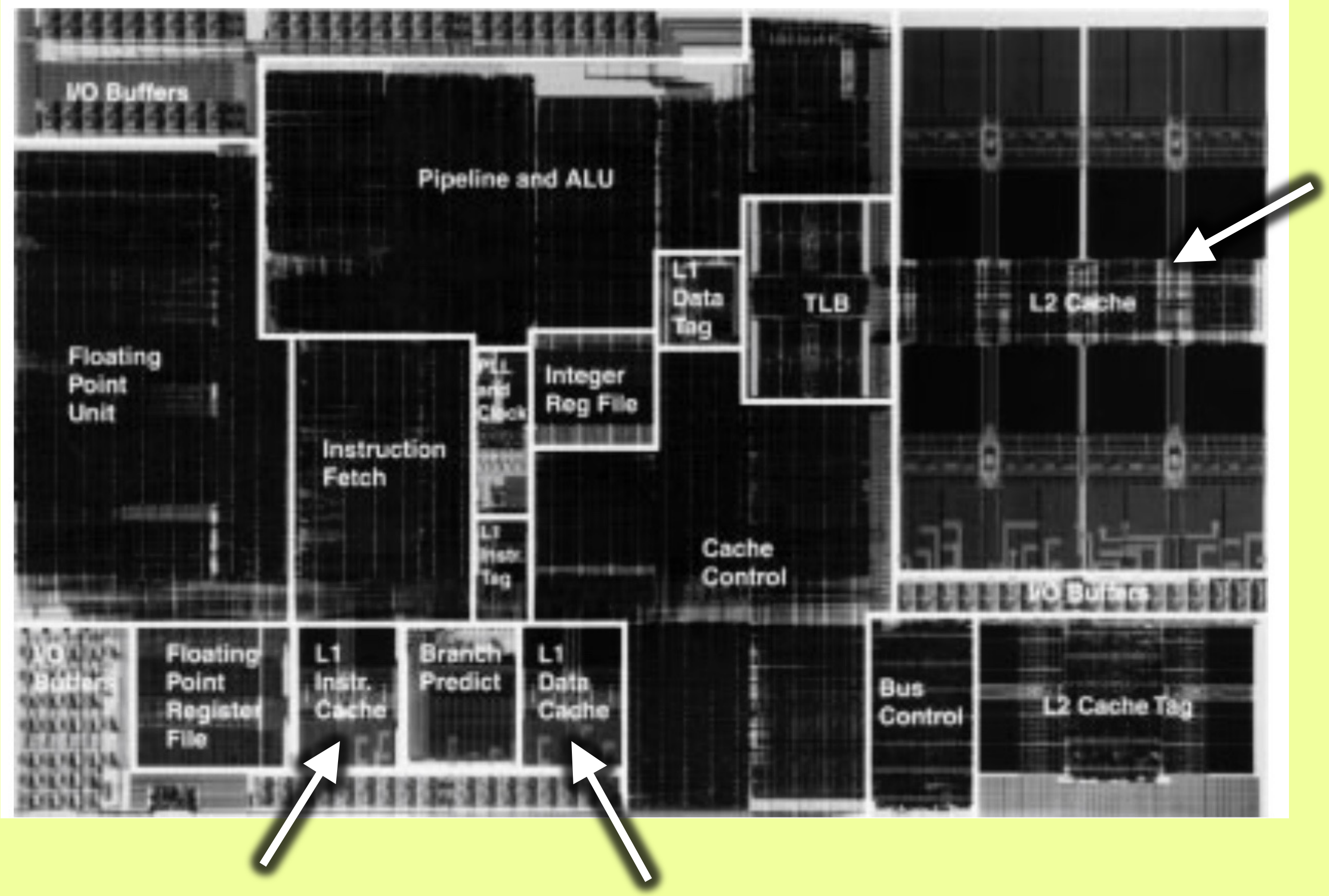


Local miss rate = misses in cache / accesses to cache

Global miss rate = misses in cache / CPU memory accesses

Misses per instruction = misses in cache / number of instructions

Exponential X704 PowerPC Processor (1997)



32KB L2 8-way Set-Associative Write-Back Unified Cache

2KB L1 Direct-Mapped Instruction Cache

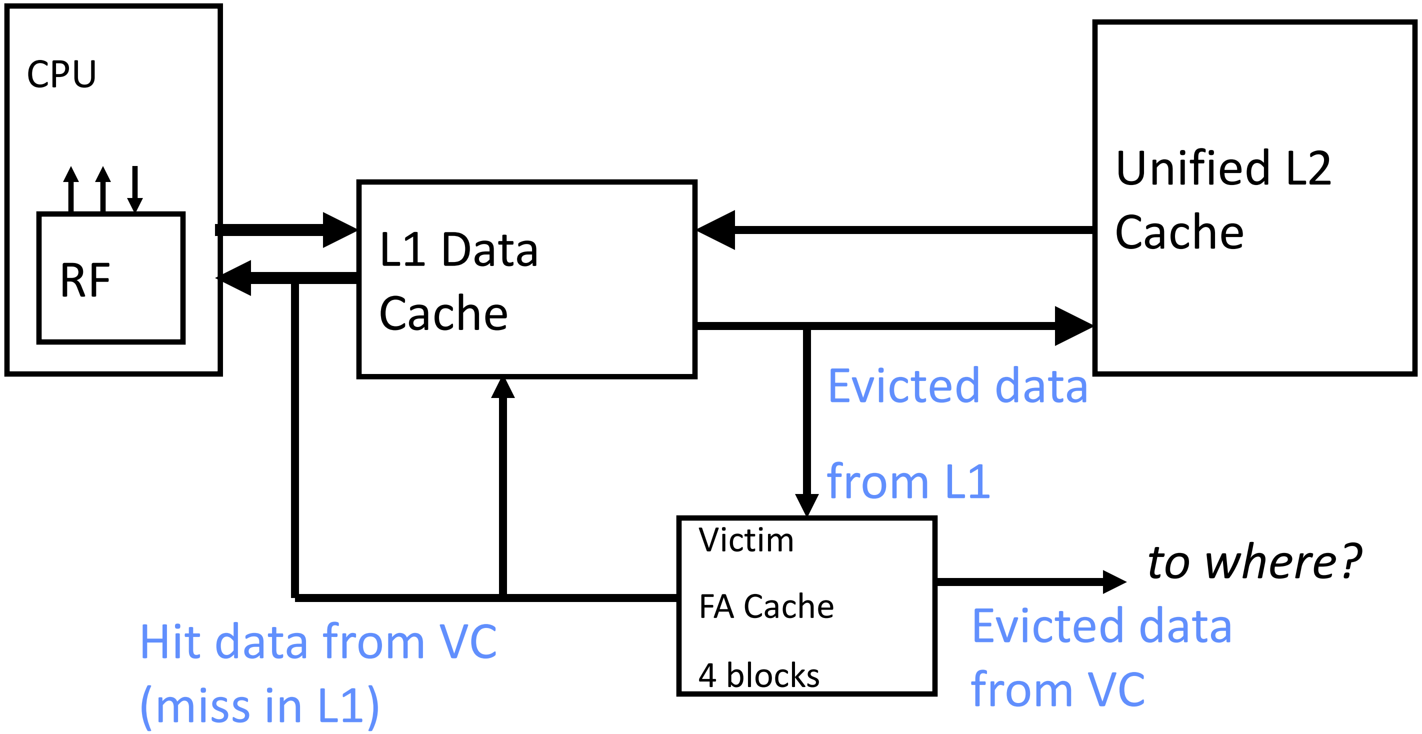
2KB L1 Direct-Mapped Write-Through Data Cache

0.5µm BiCMOS

Ran at 410-533MHz when other PC processors were much lower clock rate

Project delayed – missed market window for Apple

Victim Caches (HP 7200)



Victim cache is a small associative backup cache, added to a direct-mapped cache, which holds recently evicted lines

- First look up in direct-mapped cache
- If miss, look in victim cache
- If hit in victim cache, swap hit line with line now evicted from L1
- If miss in victim cache, L1 victim -> VC, VC victim->?

Fast hit time of direct mapped but with reduced conflict misses

MIPS R10000 Off-Chip L2 Cache
(Yeager, IEEE Micro 1996)

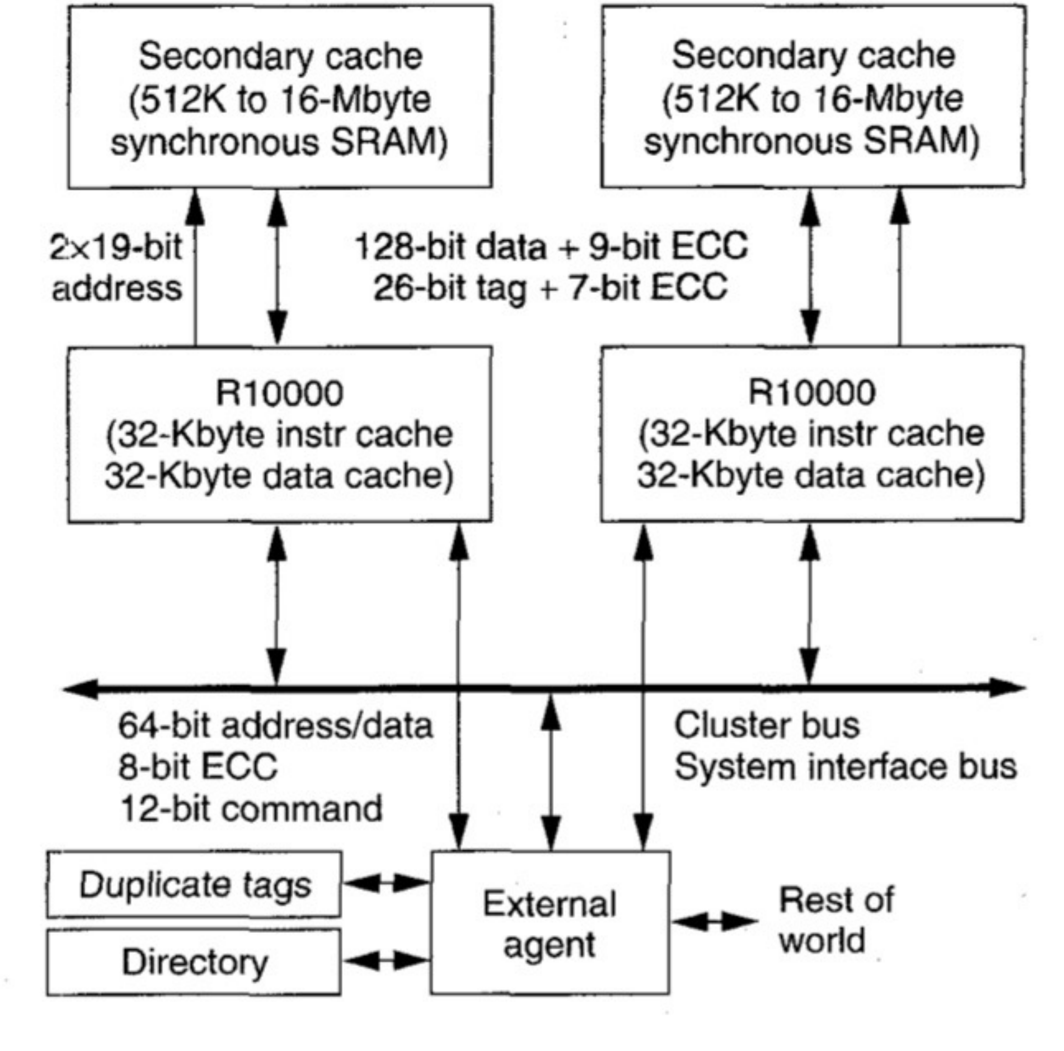
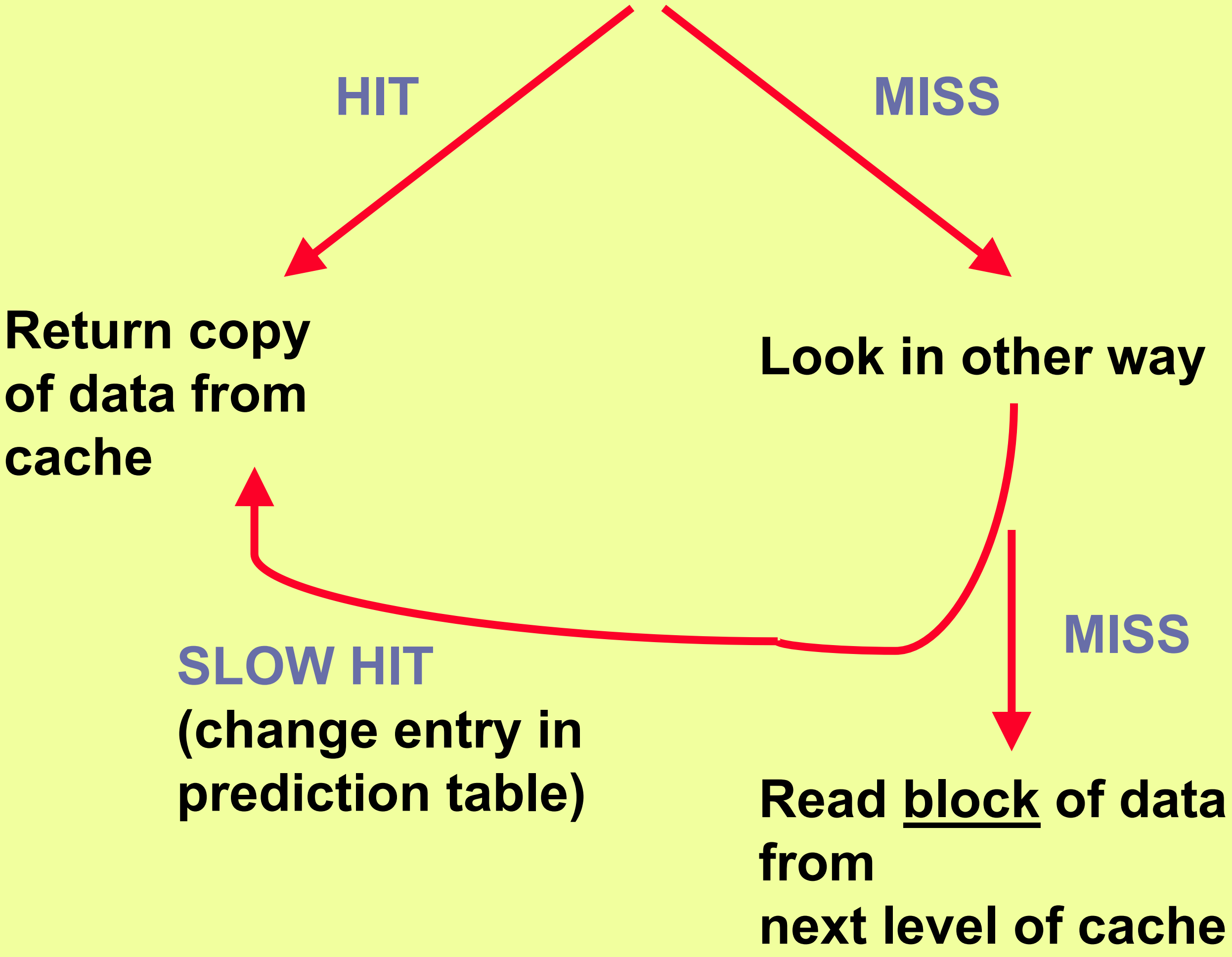

Secondary cache (512K to 16-Mbyte synchronous SRAM)
Secondary cache (512K to 16-Mbyte synchronous SRAM)
2×19-bit address
128-bit data + 9-bit ECC
26-bit tag + 7-bit ECC
R10000 (32-Kbyte instr cache 32-Kbyte data cache)
R10000 (32-Kbyte instr cache 32-Kbyte data cache)
64-bit address/data
8-bit ECC
12-bit command
Cluster bus
System interface bus
Duplicate tags
Directory
External agent
Rest of world

Figure 1. System configuration. The cluster bus directly connects as many as four chips.

Way-Predicting Caches (MIPS R10000 L2 cache)

- Use processor address to index into way-prediction table
- Look in predicted way at given index, then:



HIT

MISS

Return copy
of data from
cache

Look in other way

SLOW HIT
(change entry in
prediction table)

MISS

Read block of data
from
next level of cache

R10000 L2 Cache Timing Diagram

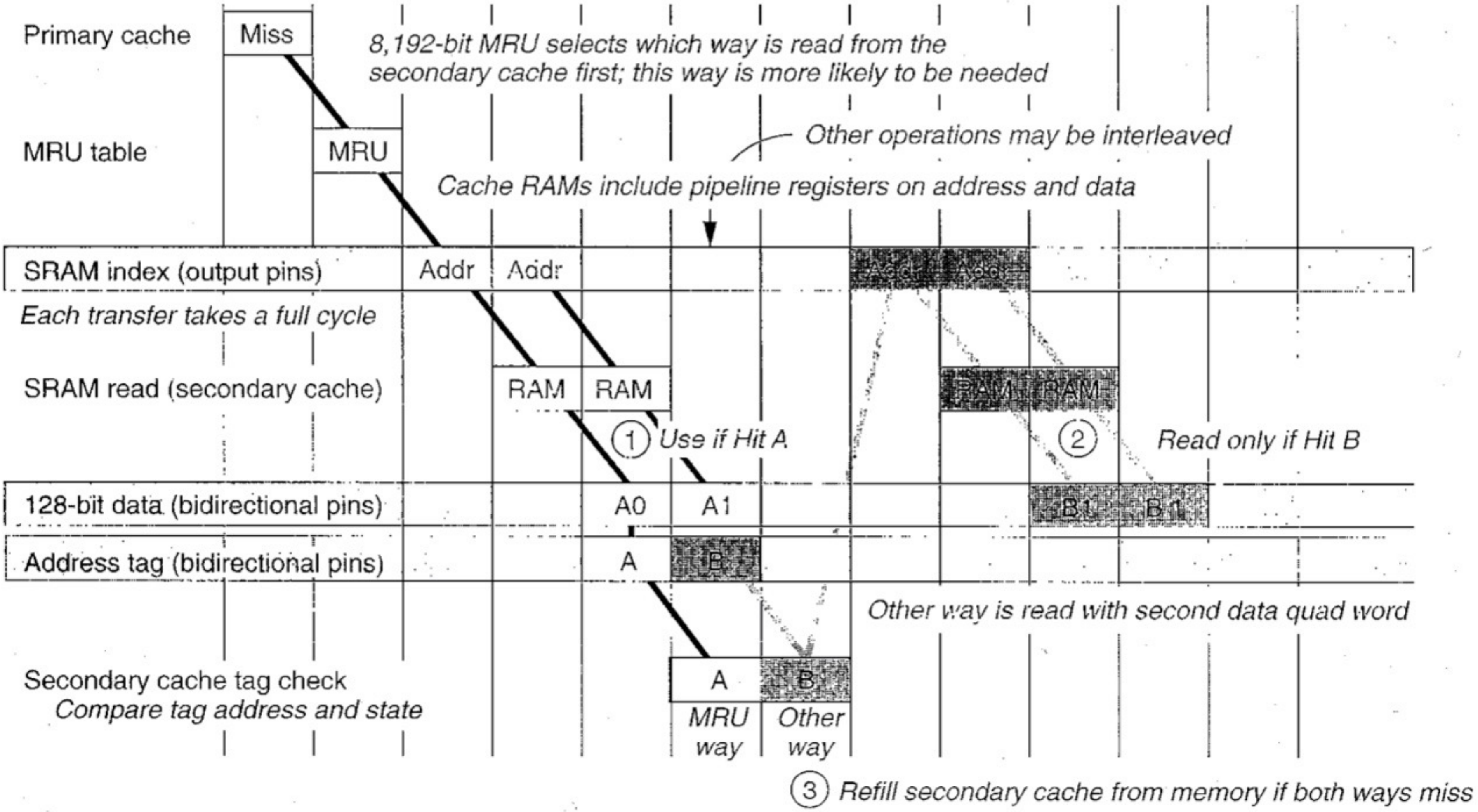
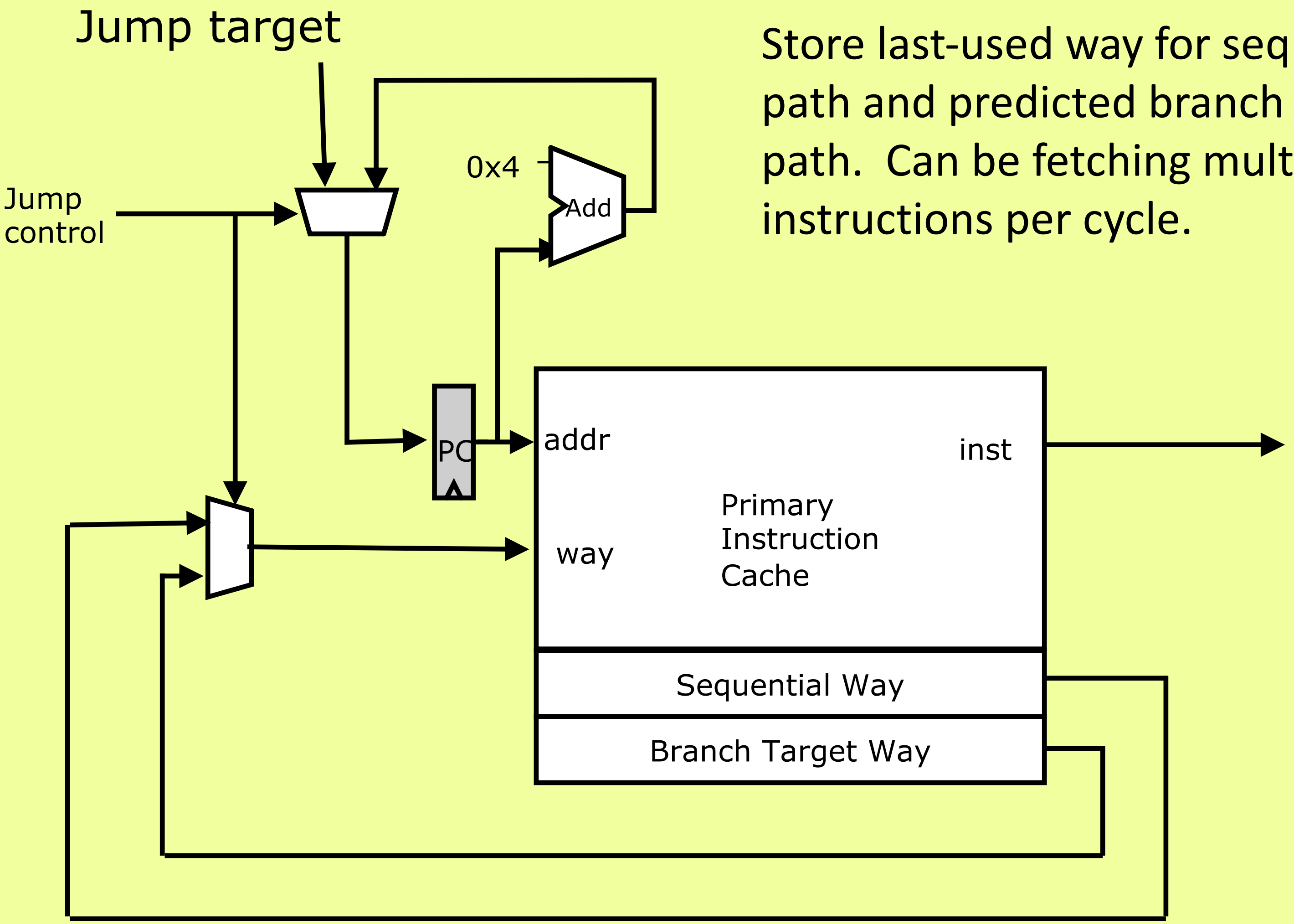

Primary cache
Miss
8,192-bit MRU selects which way is read from the secondary cache first; this way is more likely to be needed
MRU table
MRU
Other operations may be interleaved
Cache RAMs include pipeline registers on address and data
SRAM index (output pins)
Addr
Addr
Addr
Addr
Each transfer takes a full cycle
SRAM read (secondary cache)
RAM
RAM
RAM
RAM
1 Use if Hit A
2 Read only if Hit B
128-bit data (bidirectional pins)
A0
A1
B1
B1
Address tag (bidirectional pins)
A
B
Other way is read with second data quad word
Secondary cache tag check
Compare tag address and state
A
B
MRU way
Other way
3 Refill secondary cache from memory if both ways miss

Figure 12. Refill from the set-associative secondary cache. In this example, the secondary clock equals the processor's internal pipeline clock. It may be slower.

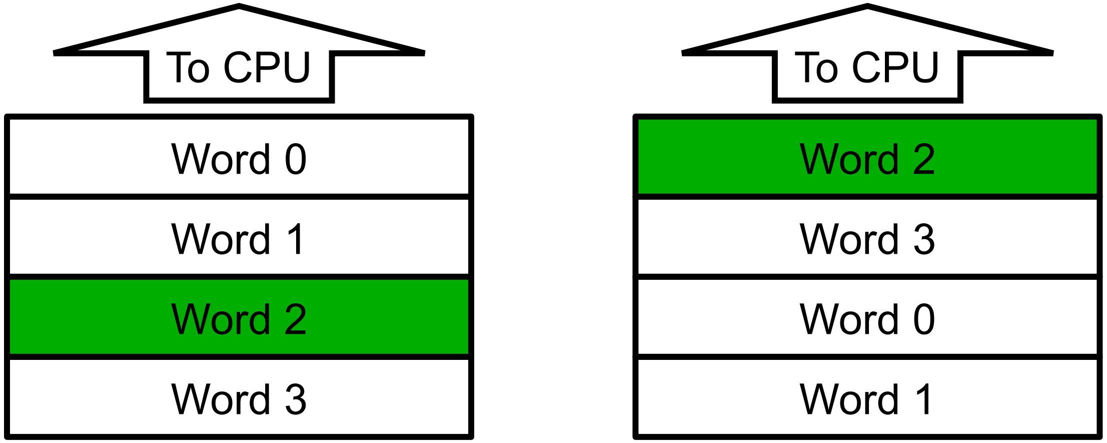
Way-Predicting Instruction Cache (Alpha 21264-like)



Store last-used way for sequential path and predicted branch taken path. Can be fetching multiple instructions per cycle.

Reduce Miss Penalty of Long Blocks: Early Restart and Critical Word First

- Don’t wait for full block 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
  - Long blocks more popular today ⇒ Critical Word 1st Widely used

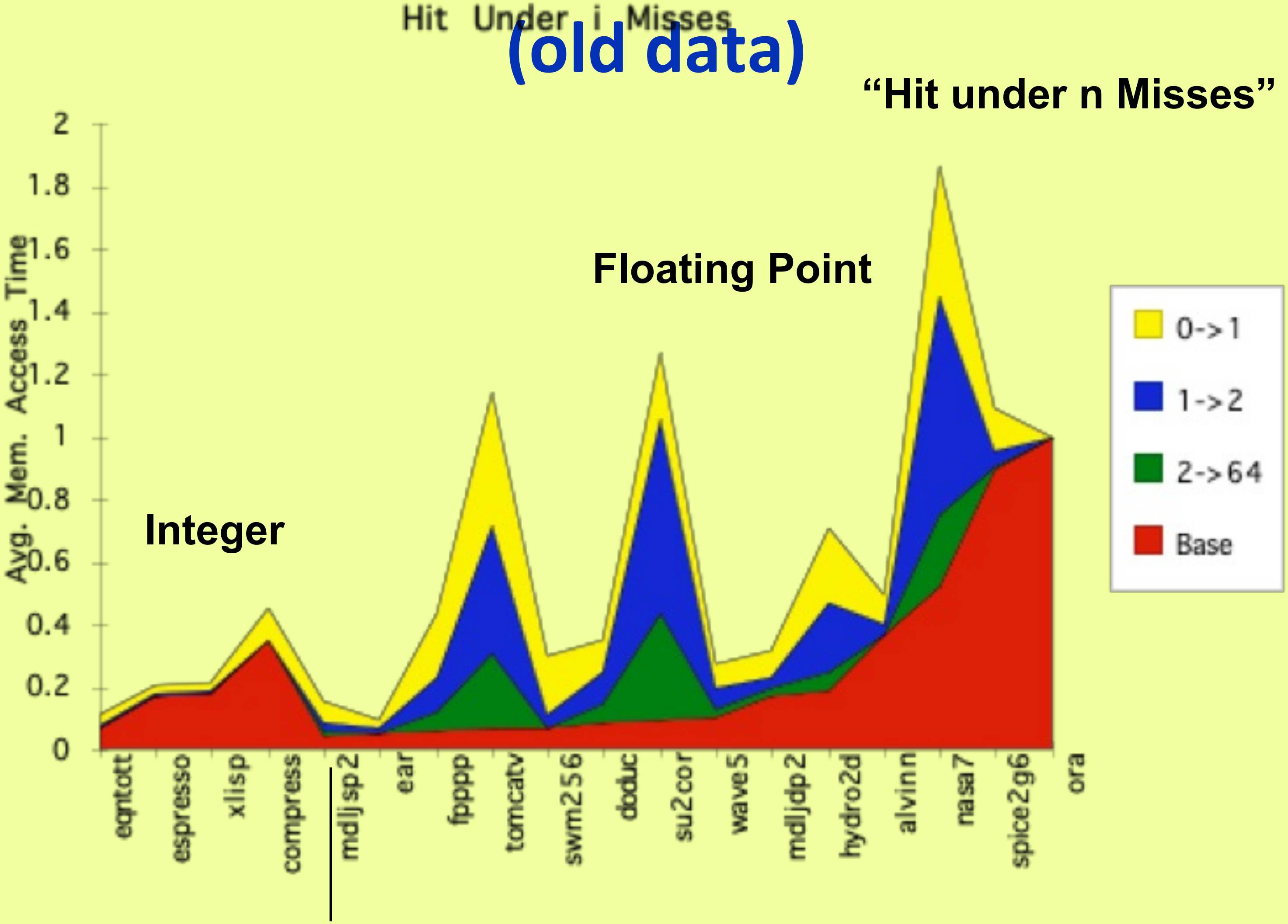


Rest of line filled in with wrap-around on cache line

Increasing Cache Bandwidth with Non-Blocking Caches

- *Non-blocking cache* or *lockup-free cache* allow data cache to continue to supply cache hits during a miss
  - requires Full/Empty bits on registers or out-of-order execution
- "*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, and can get miss to line with outstanding miss (secondary miss)
  - Requires pipelined or banked memory system (otherwise cannot support multiple misses)
  - Pentium Pro allows 4 outstanding memory misses
  - Cray X1E vector supercomputer allows 2,048 outstanding memory misses

Value of Hit Under Miss for SPEC (old data)



"Hit under n Misses"

Floating Point

Integer

0->1
1->2
2->64
Base

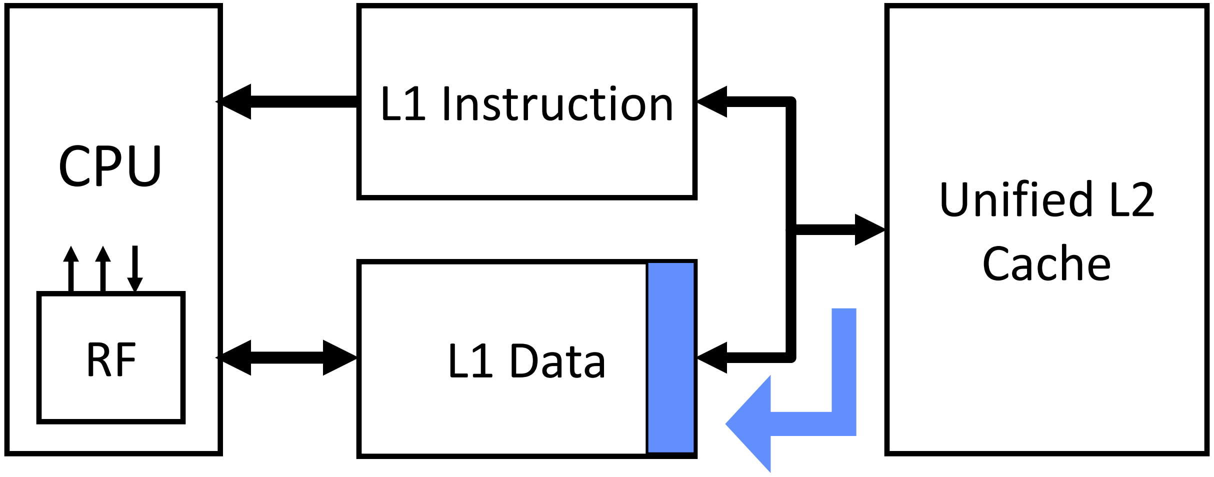
- FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26
- Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss, SPEC 92

Prefetching

- Speculate on future instruction and data accesses and fetch them into cache(s)
  - Instruction accesses easier to predict than data accesses
- Varieties of prefetching
  - Hardware prefetching
  - Software prefetching
  - Mixed schemes
- What types of misses does prefetching affect?

Issues in Prefetching

- Usefulness – should produce hits
- Timeliness – not late and not too early
- Cache and bandwidth pollution

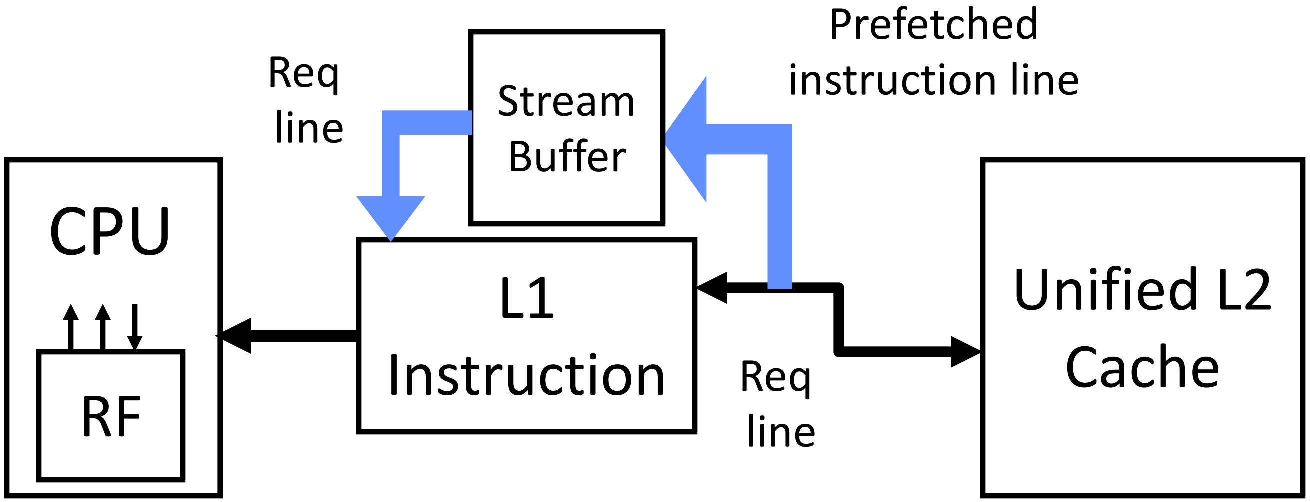


Prefetched data

Hardware Instruction Prefetching

Instruction prefetch in Alpha AXP 21064

- Fetch two lines on a miss; the requested line (i) and the next consecutive line (i+1)
- Requested line placed in cache, and next line in instruction stream buffer
- If miss in cache but hit in stream buffer, move stream buffer line into cache and prefetch next line (i+2)



Hardware Data Prefetching

- Prefetch-on-miss:
  - Prefetch b + 1 upon miss on b
- One-Block Lookahead (OBL) scheme
  - Initiate prefetch for block b + 1 when block b is accessed
  - Why is this different from doubling block size?
  - Can extend to N-block lookahead
- Strided prefetch
  - If observe sequence of accesses to line b, b+N, b+2N, then prefetch b+3N etc.

- Example: IBM Power 5 [2003] supports eight independent streams of strided prefetch per processor, prefetching 12 lines ahead of current access

Software Prefetching

```
for(i=0; i < N; i++) {
    prefetch( &a[i + 1] );
    prefetch( &b[i + 1] );
    SUM = SUM + a[i] * b[i];
 }
```

Software Prefetching Issues

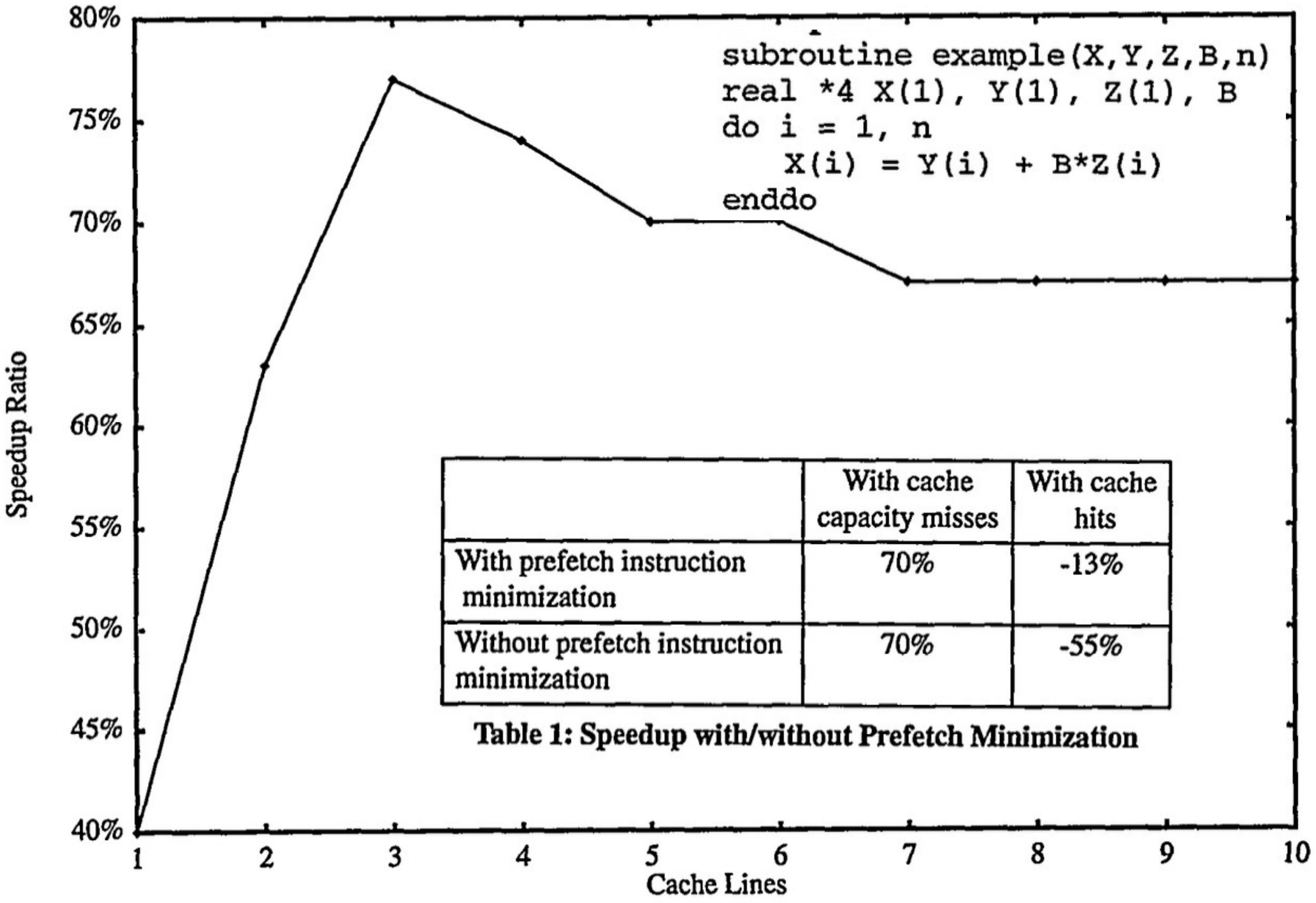
- Timing is the biggest issue, not predictability
  - If you prefetch very close to when the data is required, you might be too late
  - Prefetch too early, cause pollution
  - Estimate how long it will take for the data to come into L1, so we can set P appropriately
  - *Why is this hard to do?*

```
for(i=0; i < N; i++) {
      prefetch( &a[i + P] );
      prefetch( &b[i + P] );
      SUM = SUM + a[i] * b[i];
  }
```

Must consider cost of prefetch instructions

Software Prefetching Example

[“Data prefetching on the HP PA8000”, Santhanam et al., 1997]



```
subroutine example(X,Y,Z,B,n)
real *4 X(1), Y(1), Z(1), B
do i = 1, n
   X(i) = Y(i) + B*Z(i)
enddo
```

	With cache capacity misses	With cache hits
With prefetch instruction minimization	70%	-13%
Without prefetch instruction minimization	70%	-55%

Table 1: Speedup with/without Prefetch Minimization

Figure 2: Speedup Ratio for Different Prefetch Distances

Compiler Optimizations

- Restructuring code affects the data 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 ("no-allocate" 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

Loop Interchange

```
for(j=0; j < N; j++) {
    for(i=0; i < M; i++) {
        x[i][j] = 2 * x[i][j];
    }
}
```

↓

```
for(i=0; i < M; i++) {
    for(j=0; j < N; j++) {
        x[i][j] = 2 * x[i][j];
    }
}
```

What type of locality does this improve?

Loop Fusion

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

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

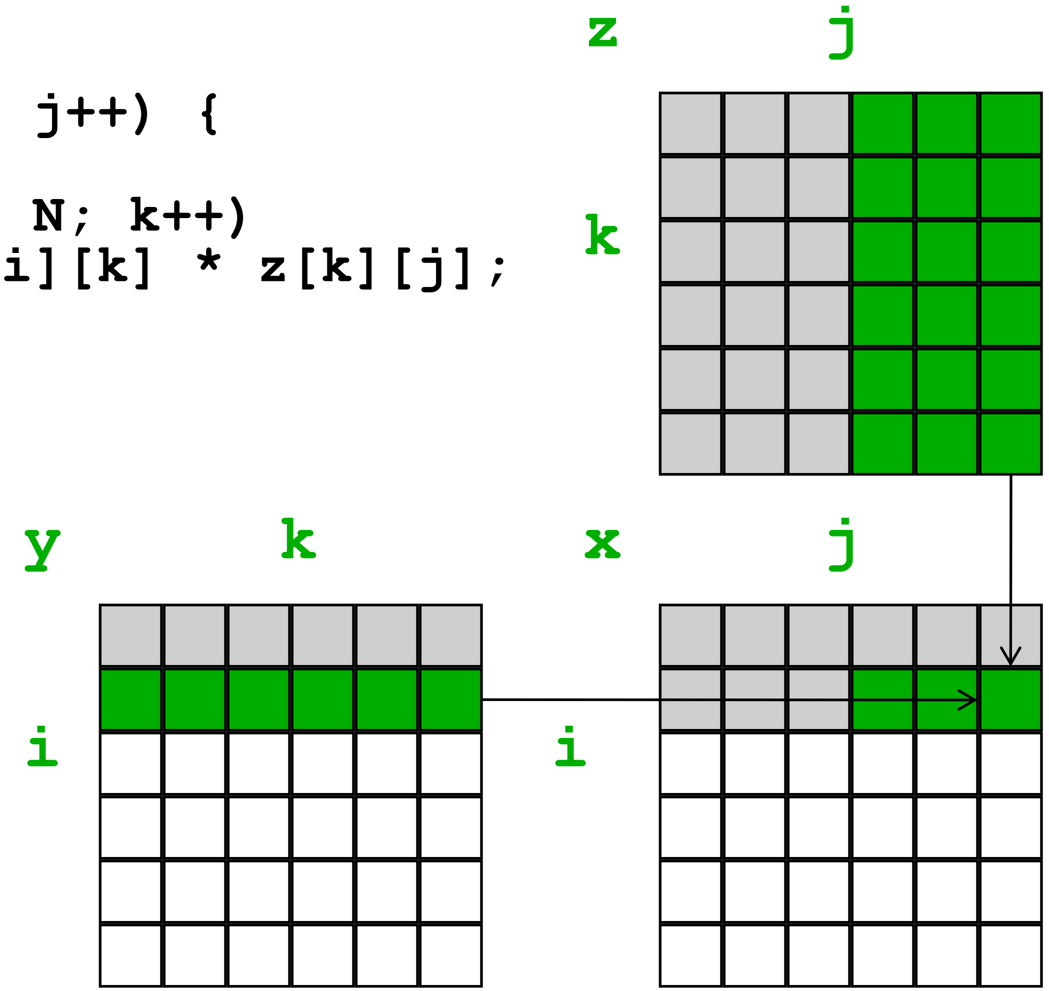
⇩

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

What type of locality does this improve?

Matrix Multiply, Naïve Code

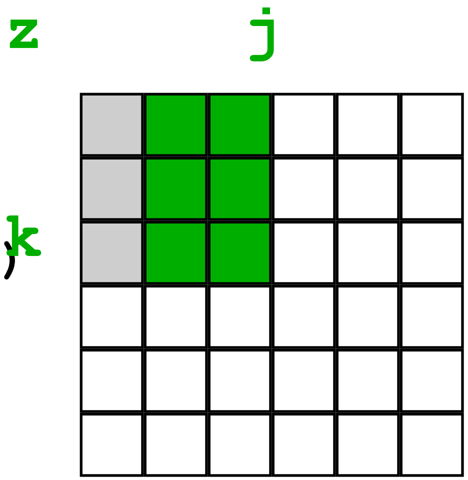
```
for(i=0; i < N; i++)
     for(j=0; j < N; j++) {
        r = 0;
        for(k=0; k < N; k++)
          r = r + y[i][k] * z[k][j];
        x[i][j] = r;
     }
```

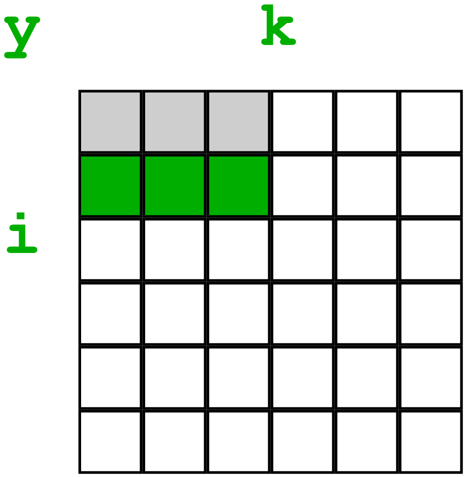


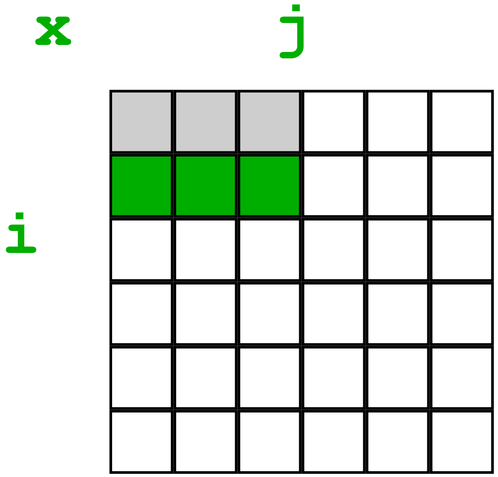
□ *Not touched* ▒ *Old access* 🟩 *New access*

Matrix Multiply with Cache Tiling

```
for(jj=0; jj < N; jj=jj+B)
    for(kk=0; kk < N; kk=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] = x[i][j] + r;
           }
```







What type of locality does this improve?

Acknowledgements

- This course is partly inspired by previous MIT 6.823 and Berkeley CS252 computer architecture courses created by my collaborators and colleagues:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)