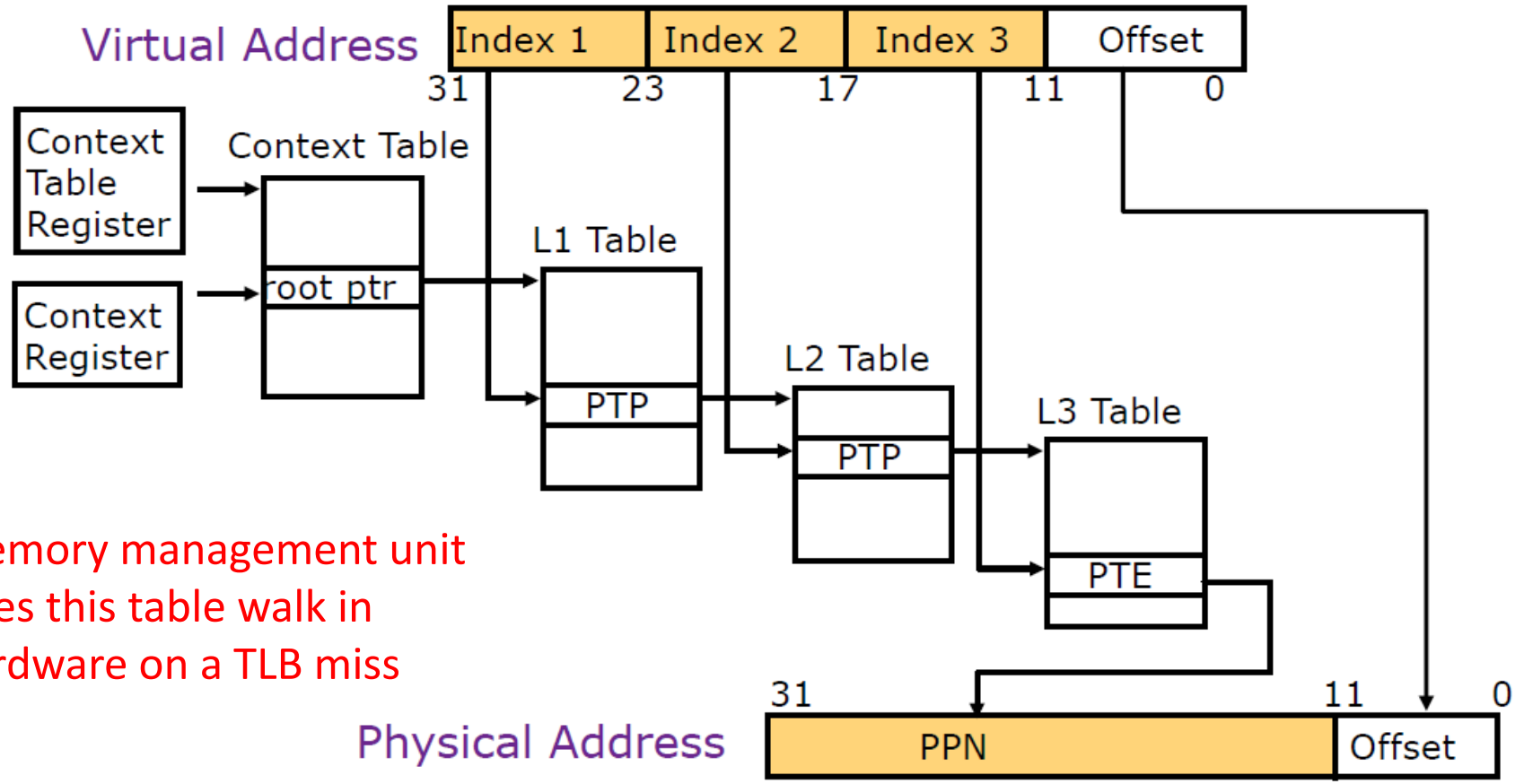
CSE 140
Computer Architecture

Lecture 11 – Memory Architecture (2)

Announcement

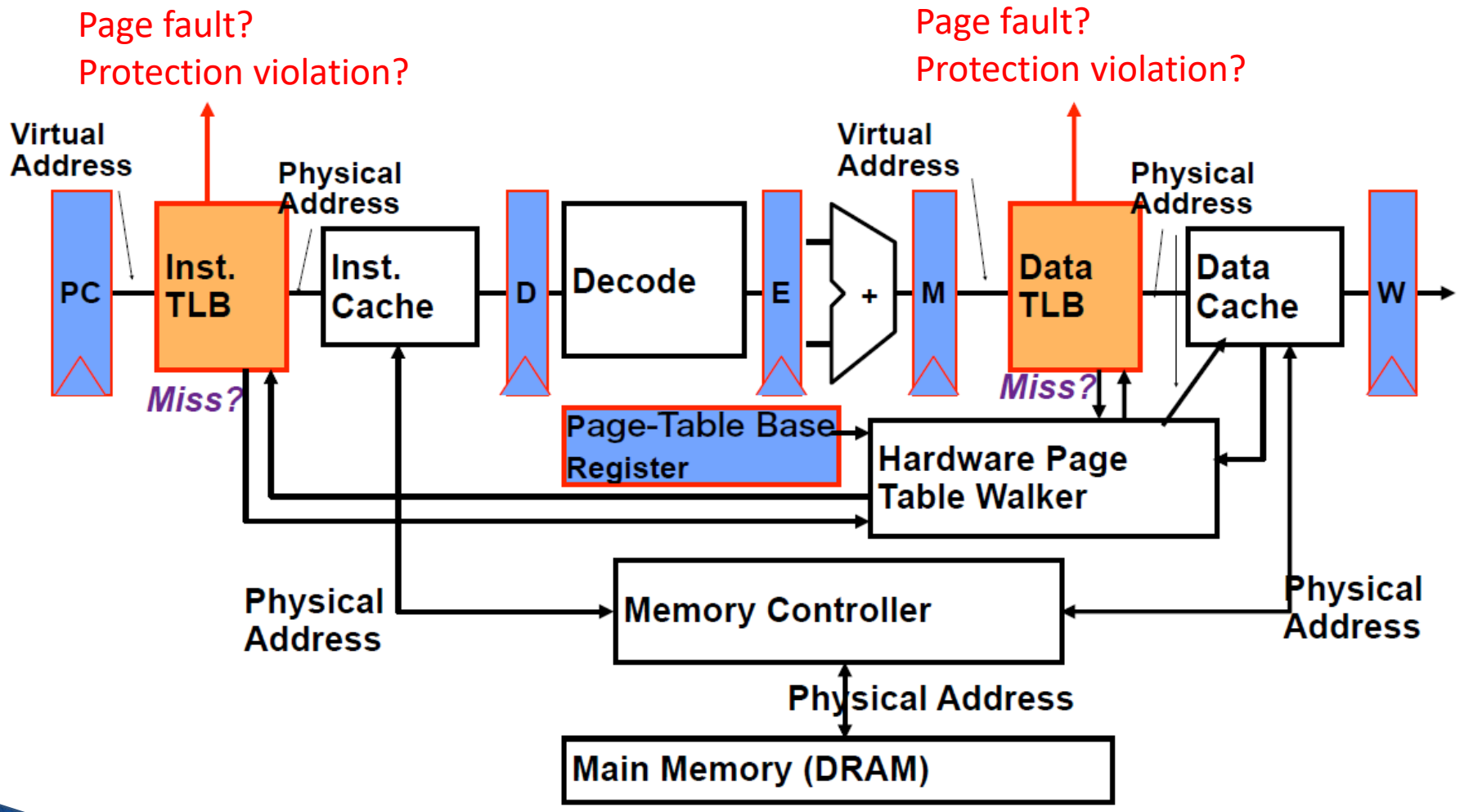
- HW #3
  - Due in 1 week (same as HW #3a)
- Project #1
  - Due 10/11 (Friday) at 11:59pm
- Midterm #1
  - 10/17, during lecture (not 10/10)
  - Lectures #1 - #8 (No virtual memory)
  - HW #1 - #3
  - Review on 10/15
- Reading assignment
  - Chapter 4.9, 4.10, 4.12
    - Do all Participation Activities in each section
    - Due Thursday (10/10) at 11:59pm

Hierarchical Page Table Walk: SPARC v8



Memory management unit does this table walk in hardware on a TLB miss

Page-Based Virtual-Memory Machine

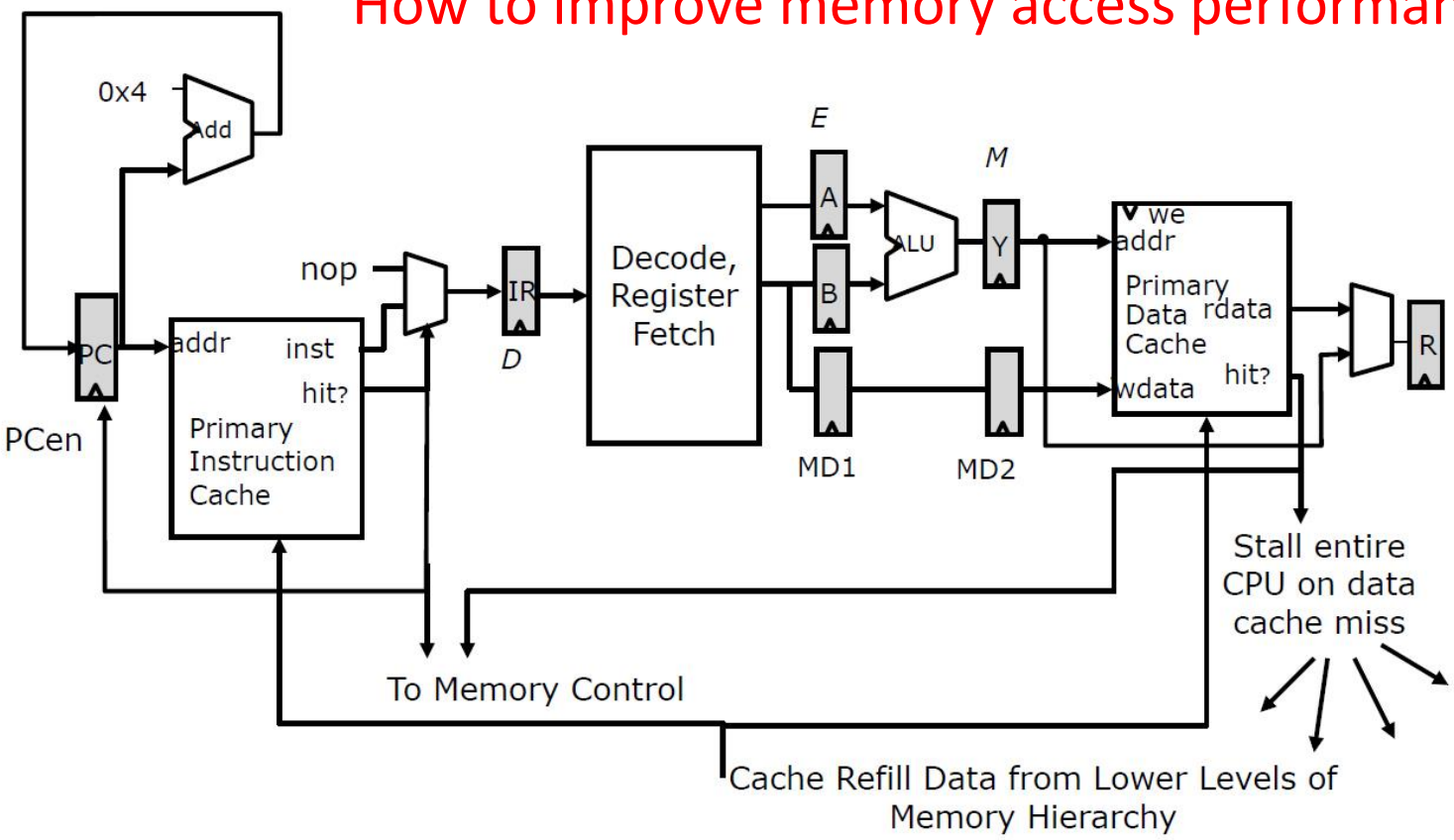


Assumes page tables held in untranslated physical memory

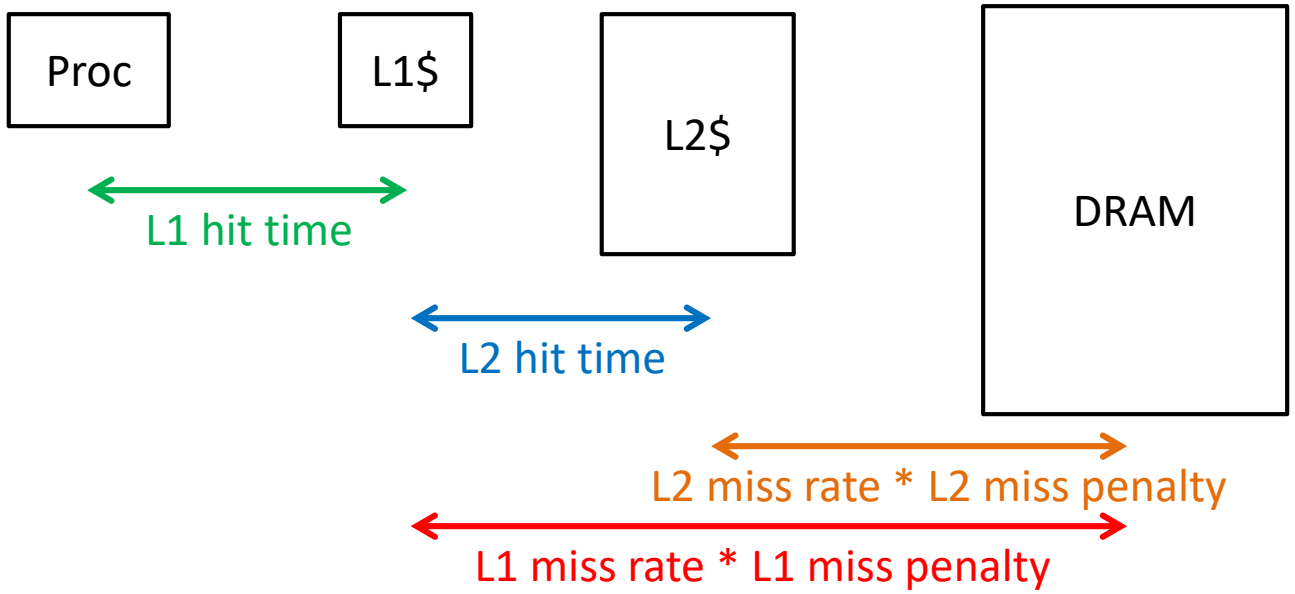
CPU-Cache Interaction

5-stage pipeline

How to improve memory access performance?



Analyzing Multi-level cache hierarchy



Avg Mem Access Time (AMAT)

= L1 hit time + L1 miss rate * L1 miss penalty

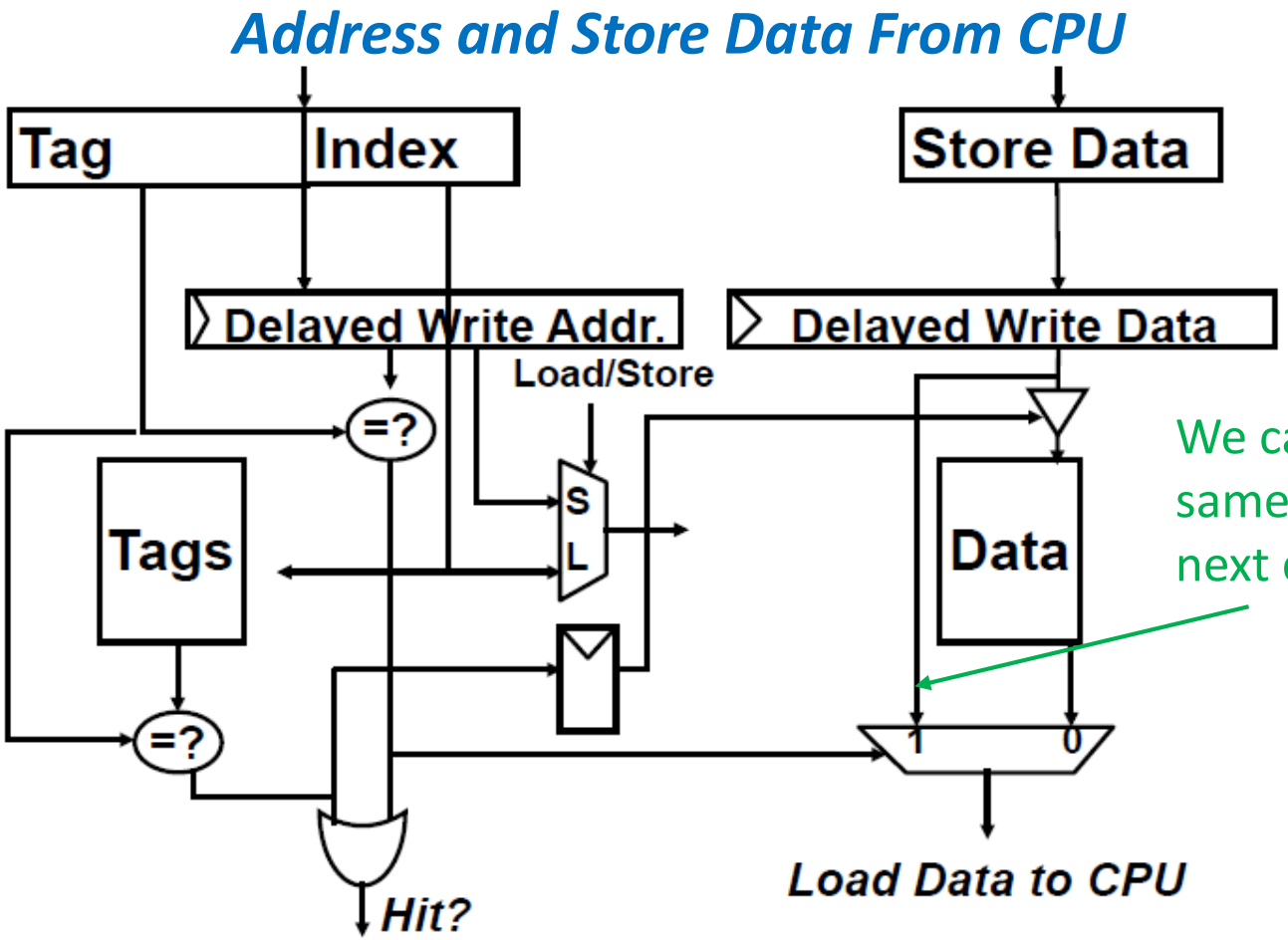
L1 miss penalty = L2 hit time + L2 miss rate * L2 miss penalty

AMAT = L1 hit time + L1 miss rate * (L2 hit time + L2 miss rate * L2 miss penalty)

Reducing Write Hit Time

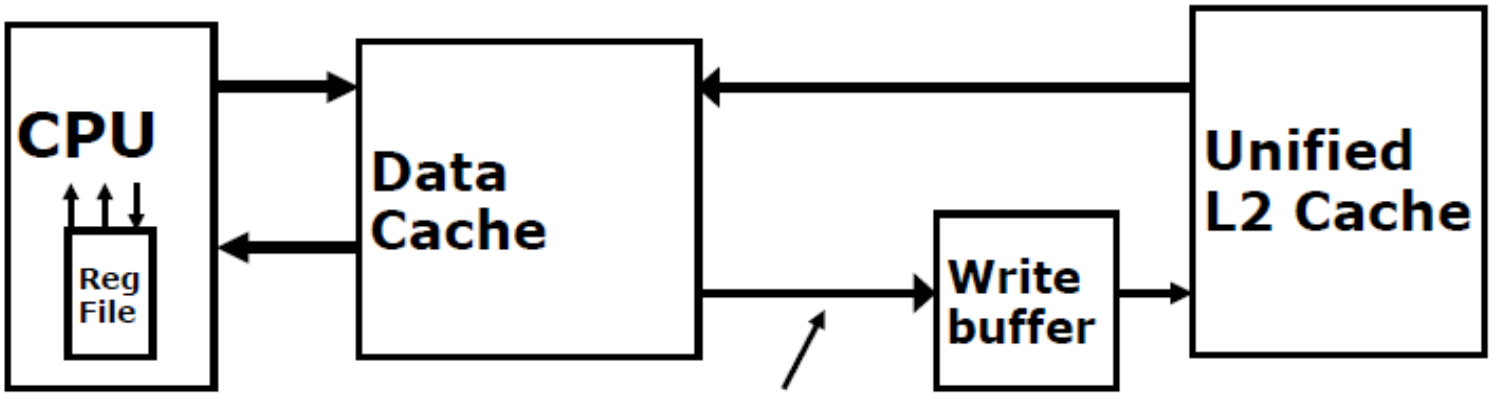
- Problem: Writes take two sub-cycles in memory stage: one for tag check (read) plus one for data write if hit
- Solutions:
  - Design data cache that can perform read and write in one cycle, restore old value after tag miss (wrong block)
  - CAM-Tag caches (Fully-associative): Data block only enabled if hit
  - Pipelined writes: Hold write data for store in single buffer ahead of cache, write cache data during next store's tag check

(1) Pipelining Cache Writes



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

(2) Write Buffer to Enable Reads



Replace dirty lines for **writeback** cache / All writes in **writethrough** cache

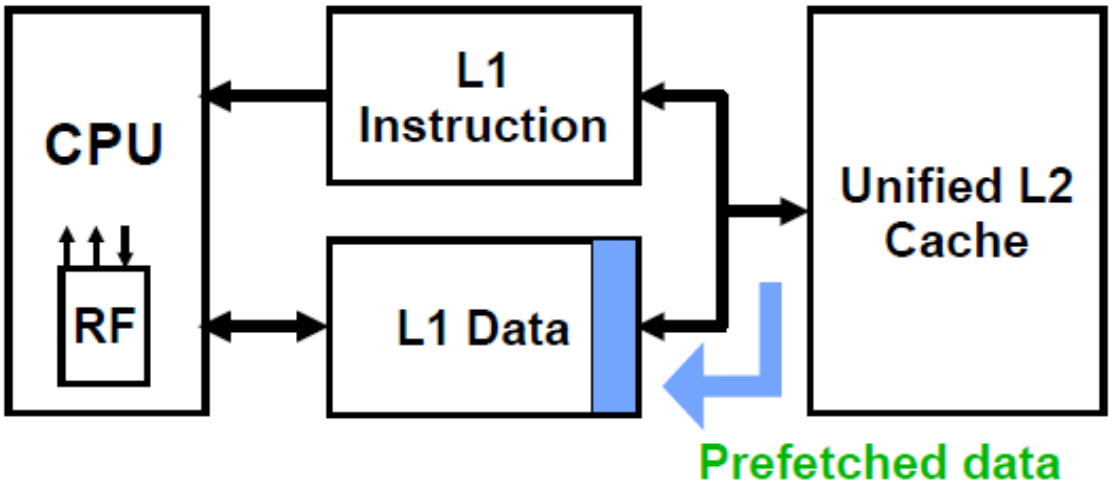
- Processor is not stalled on writes, and any following read misses may go ahead of write to main memory
  - Problem: Write buffer may hold updated value of location needed by a read miss
  - Simple scheme: on a read miss, wait for the write buffer to go empty
  - Faster scheme: Check write buffer addresses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

(3) Prefetching

- Speculate on future instruction or 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?

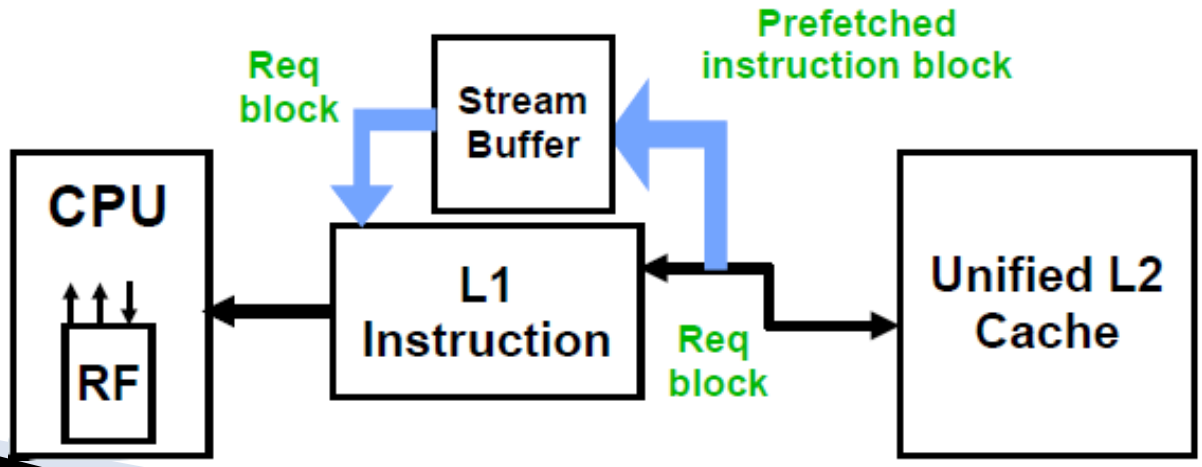
Issues in Prefetching

- Usefulness – should produce hits (higher hit rate)
- Timeliness – not late and not too early
  - Cache and bandwidth pollution



Hardware Instruction Prefetching

- Instruction prefetch in Alpha AXP 21064
  - Fetch two blocks on a miss; the requested block (i) and the next consecutive block (i+1) (which locality?)
  - Requested block placed in cache, and next block in instruction stream buffer
  - If miss in cache but hit in stream buffer, move stream buffer block into cache and prefetch next block (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 block b, b+N, b+2N, then prefetch b+3N etc. (in what occasion?)
    - 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 in cache
  - Estimate how long it will take for the data to come into L1, so we can set prefetch appropriately
  - Why is this hard to do?
    - Must consider cost of prefetch instructions
  - Software prefetching may reduce morden CPU performance

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

(4) 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
    - "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

Loop Interchange

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

Column by column

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

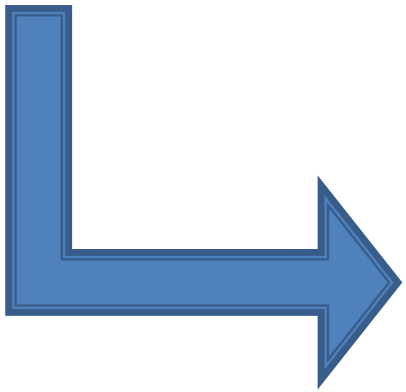
Row by row

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];
```

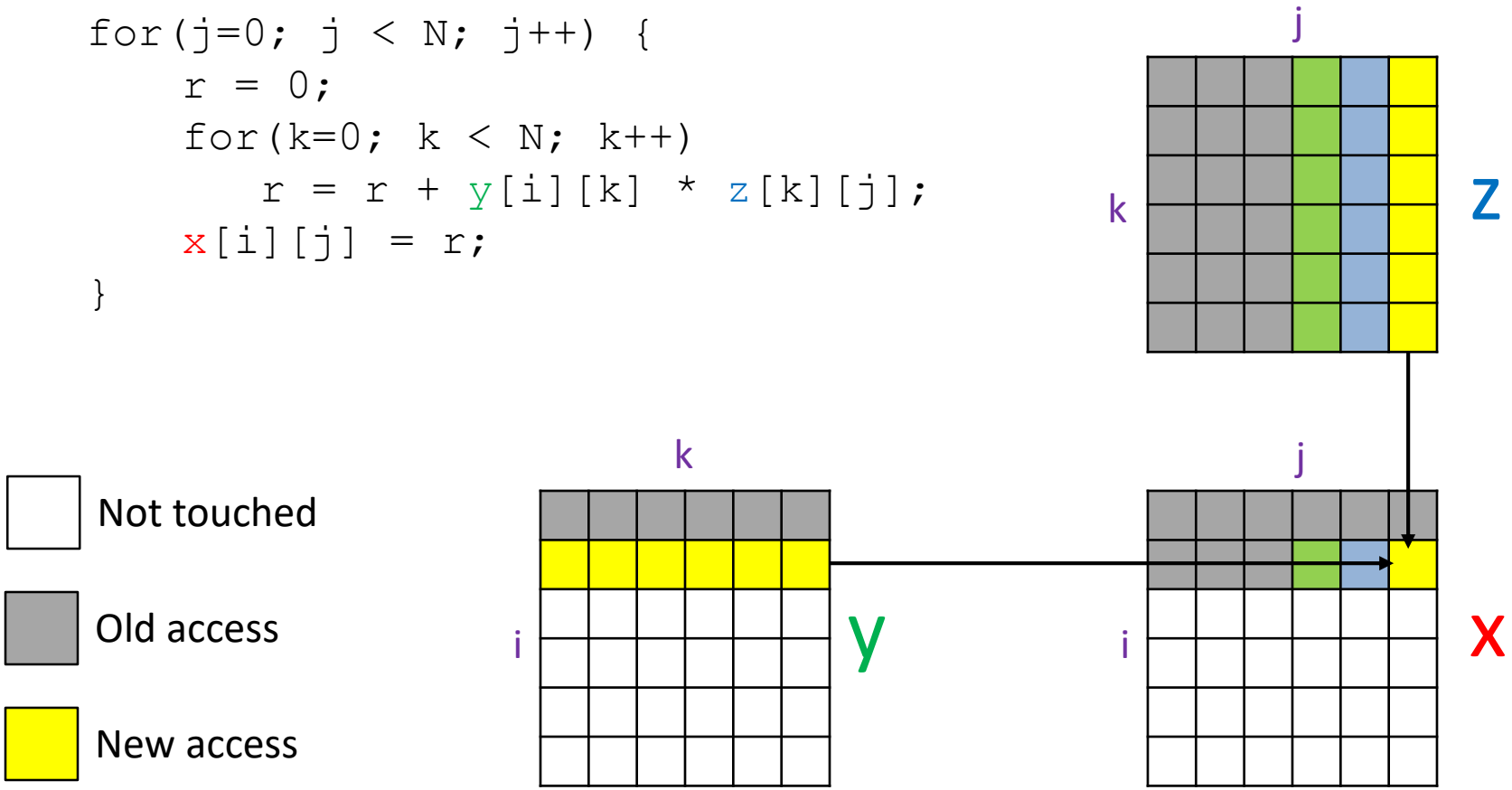


```
for(i=0; i < N; i++)
{
   a[i] = b[i] * c[i];
   d[i] = a[i] * c[i];
}
```

What type of locality does this improve?

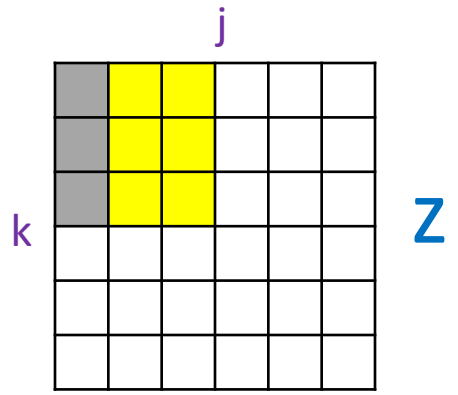
Matrix Multiply, Naïve Code

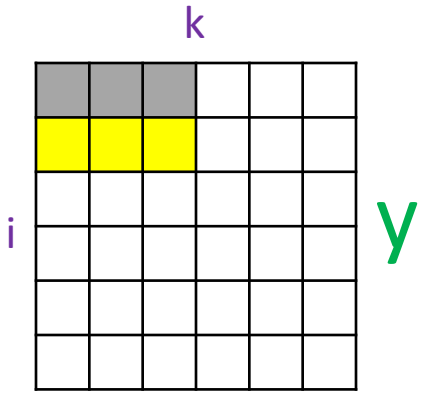
```
for(i=0; i < N; i++)
    for(j=0; j < N; j++) {
        r = 0;
        for(k=0; k < N; k++)
            r = r + y[i][k] * z[k][j];
        x[i][j] = r;
    }
```

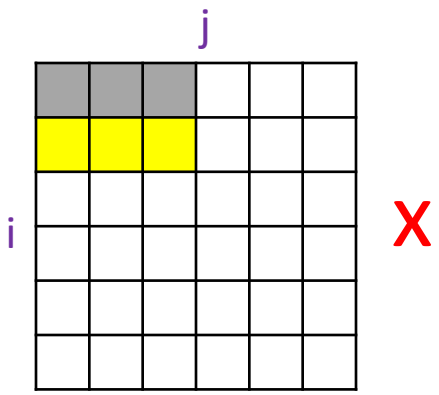


Matrix Multiply with Cache Tiling

```
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?

Summary

- To improve memory performance
  - Pipelined cache writes
  - Write buffer
  - Prefetching
    - Hardware/Software
  - Compiler optimizations