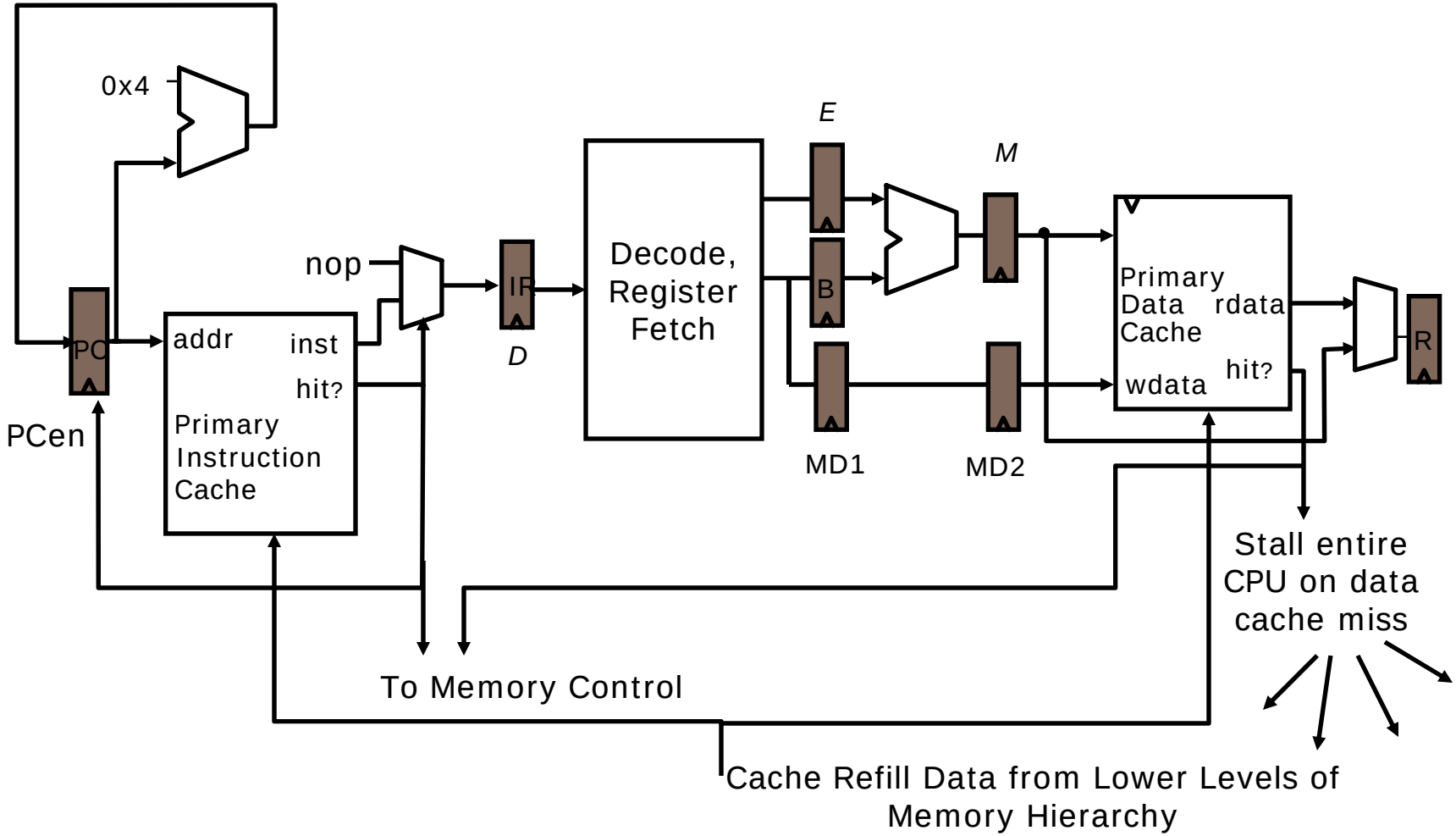
Cache Optimizations

Joel Emer
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

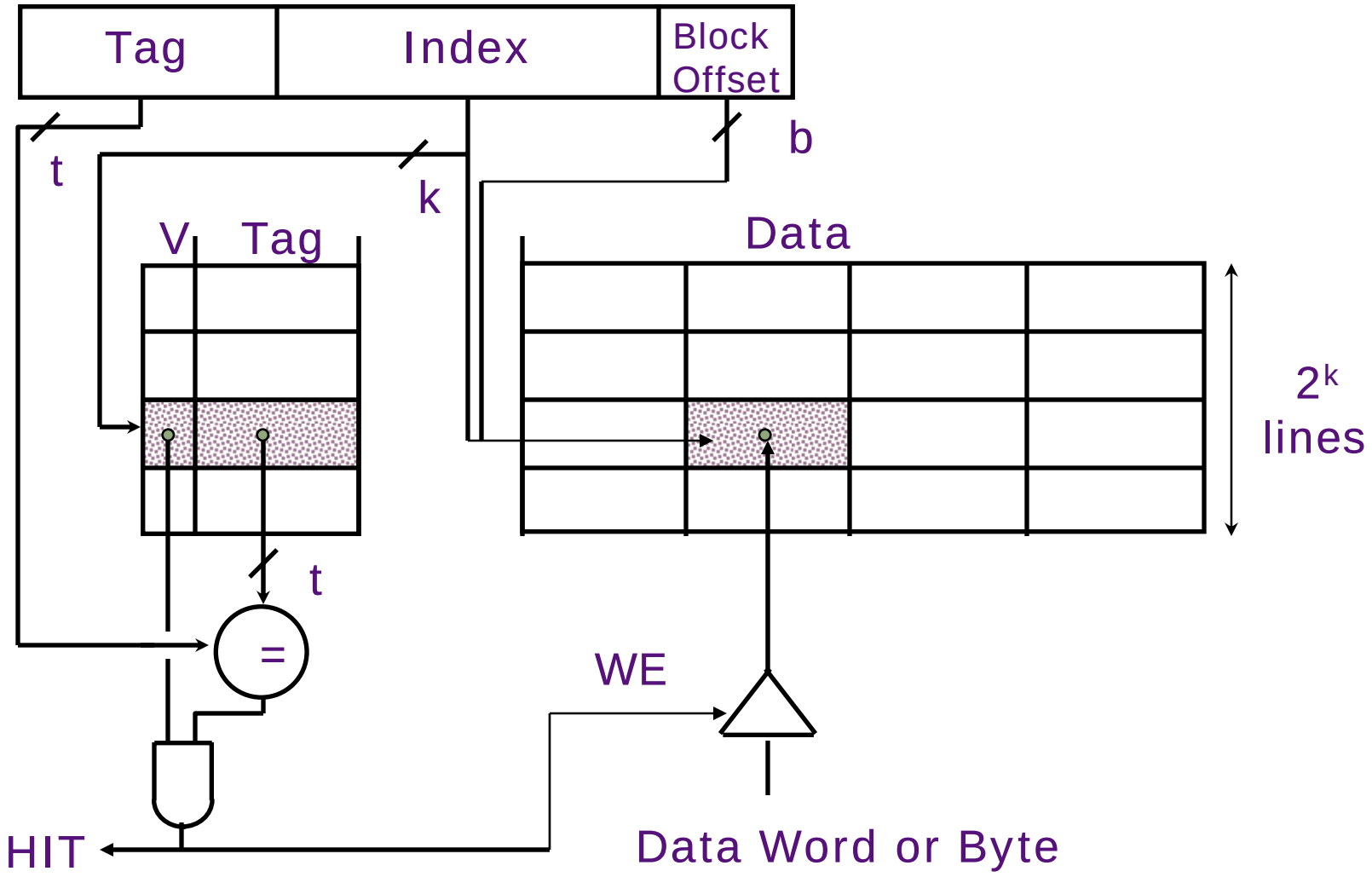
Based on the material prepared by
Krste Asanovic and Arvind

CPU-Cache Interaction
(5-stage pipeline)



What about Instruction miss or writes to i-stream ?

Write Performance



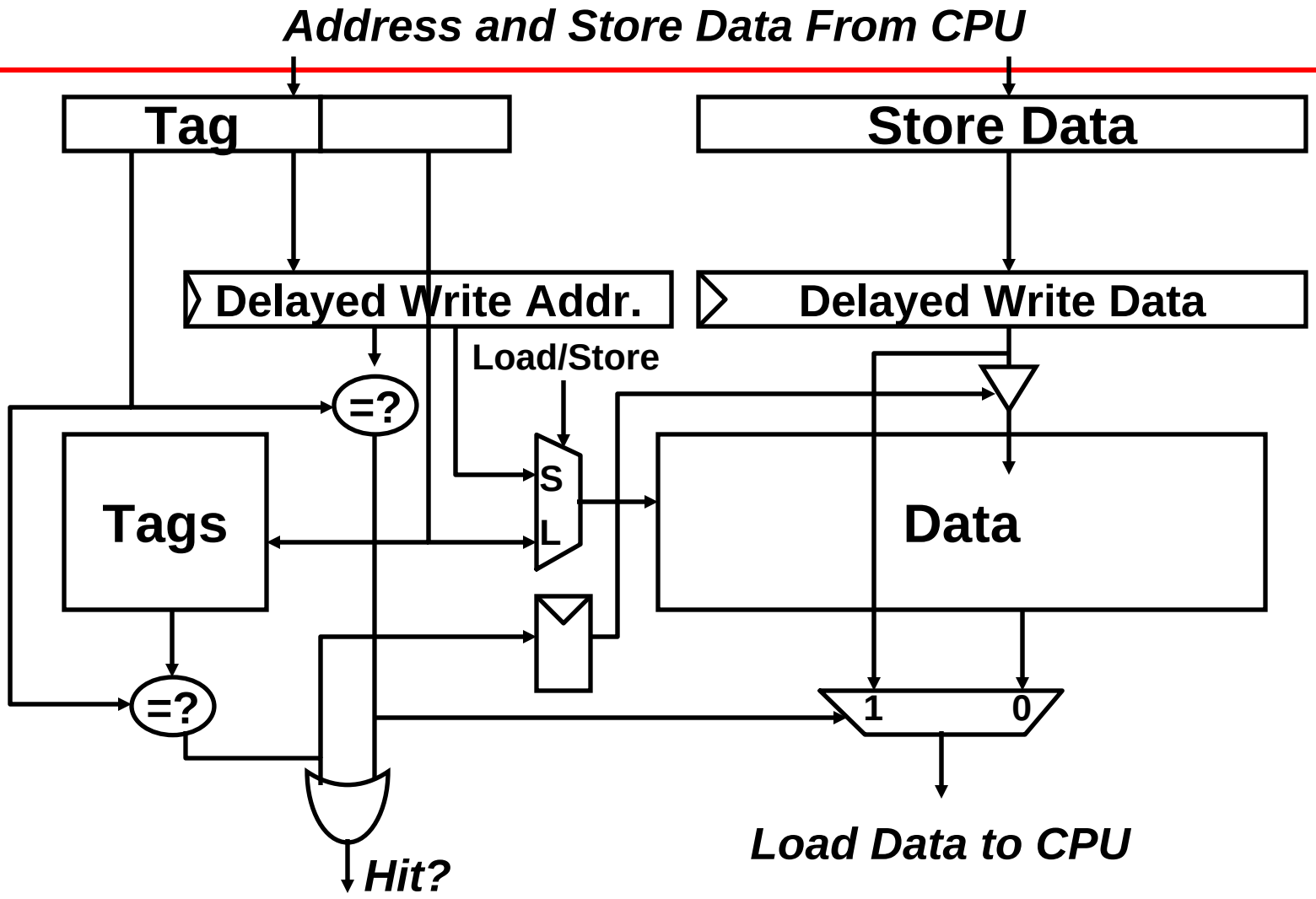
Reducing Write Hit Time

Problem: Writes take two cycles in memory stage, one cycle for tag check plus one cycle for data write if hit

Solutions:

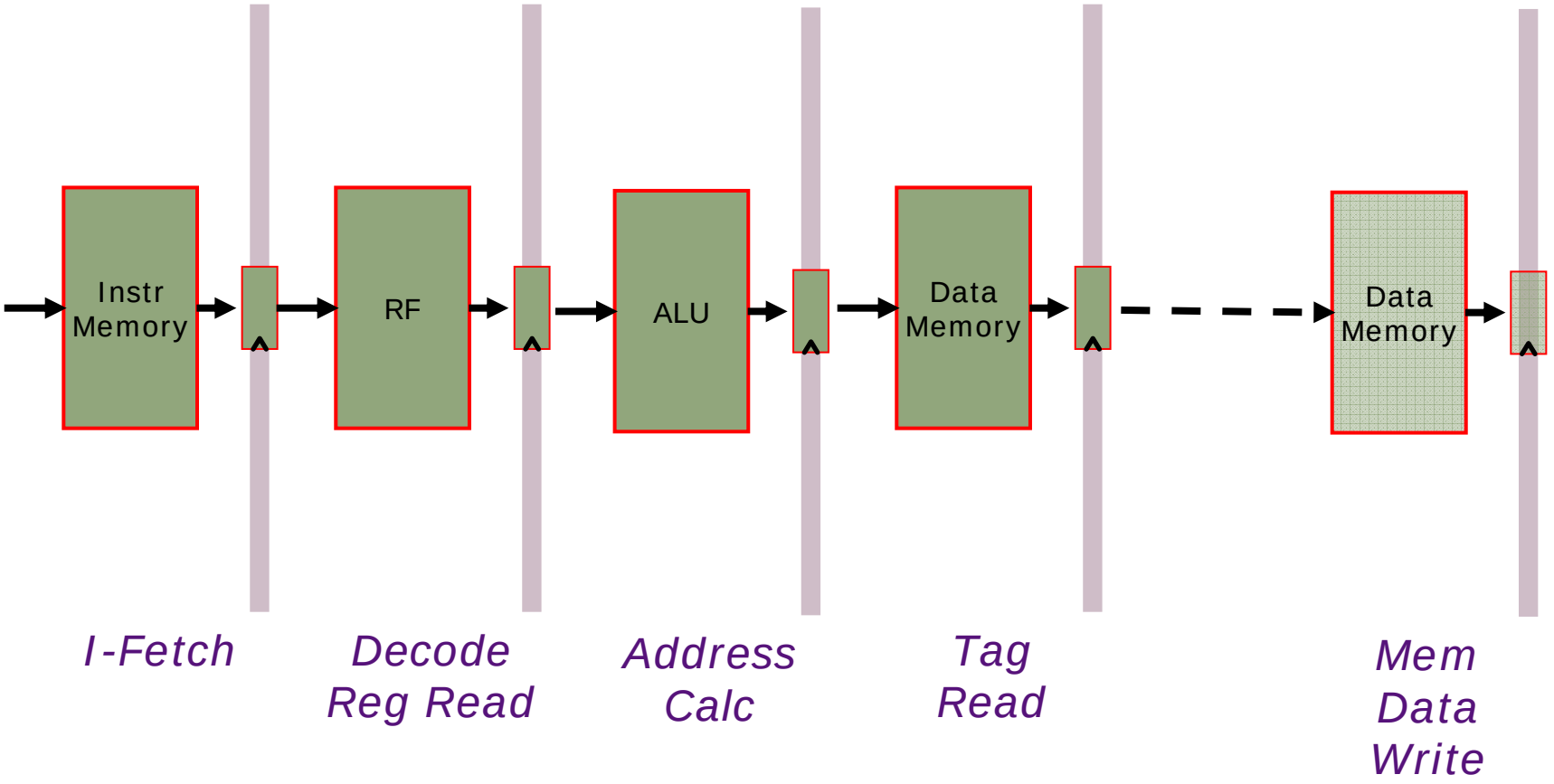
- Design data RAM that can perform read and write in one cycle, restore old value after tag miss
- CAM-Tag caches: Word line only enabled if hit
- Pipelined writes: Hold write data for store in single buffer ahead of cache, write cache data during next store's tag check

Pipelining Cache Writes



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

Write pipeline



What hazard has been introduced in this pipeline?

Write Policy

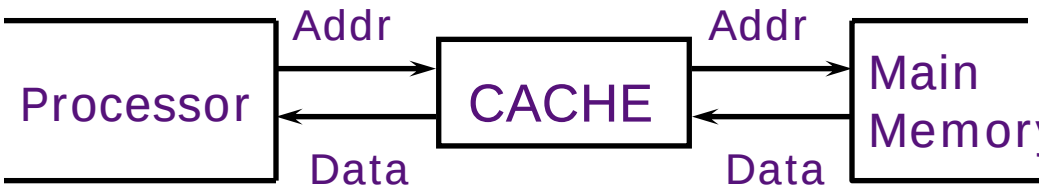
- Cache hit:
  - write through: write both cache & memory
    - generally higher traffic but simplifies cache coherence
  - write back: write cache only
    (memory is written only when the entry is evicted)
    - a dirty bit per block can further reduce the traffic
- Cache miss:
  - no write allocate: 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

Average Cache Read Latency

α is HIT RATIO: Fraction of references in cache

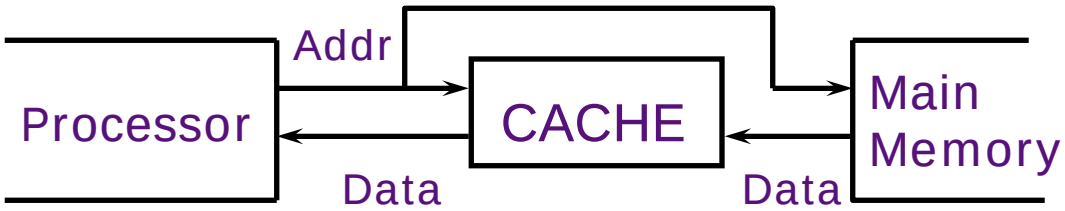
1 - α is MISS RATIO: Remaining references

Average access time for serial search:



$t_c + (1 - \alpha)\, t_m$

Average access time for parallel search:



$\alpha\, t_c + (1 - \alpha)\, t_m$

t_c is smallest for which type of cache?

Improving Cache Performance

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

To improve performance:

- reduce the miss rate (e.g., larger cache)
- reduce the miss penalty (e.g., L2 cache)
- reduce the hit time

What is the simplest design strategy?

Improving Cache Performance

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

To improve performance:

- reduce the miss rate (e.g., larger cache)
- reduce the miss penalty (e.g., L2 cache)
- reduce the hit time

The simplest design strategy is to design the largest primary cache without slowing down the clock or adding pipeline stages

(but design decisions are more complex with out-of-order or highly pipelined CPUs)

Causes for Cache Misses

- *Compulsory:* first-reference to a block *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 placement & replacement policy

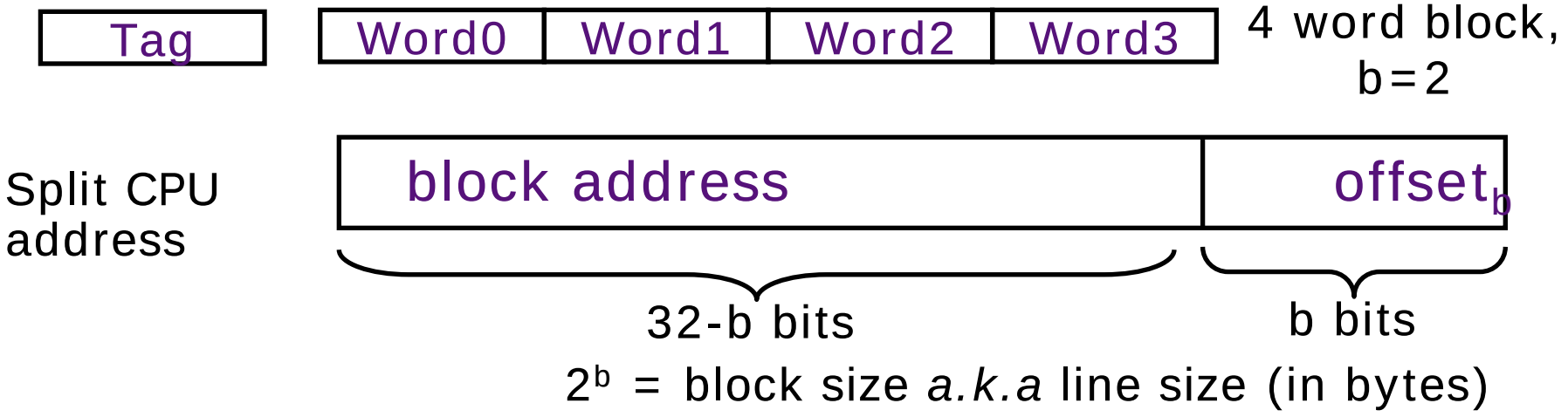
- *Conflict:* misses that occur because of collisions due to block-placement strategy
  - misses that would not occur with full associativity

Effect of Cache Parameters on Performance

- Larger cache size
  - + reduces capacity and conflict misses
  - - hit time will increase
- Higher associativity
  - + reduces conflict misses (up to around 4-8 way)
  - - may increase access time
- Larger block size

Block Size and Spatial Locality

Block is unit of transfer between the cache and memory



Larger block size has distinct hardware advantages

- less tag overhead
- exploit fast burst transfers from DRAM
- exploit fast burst transfers over wide busses

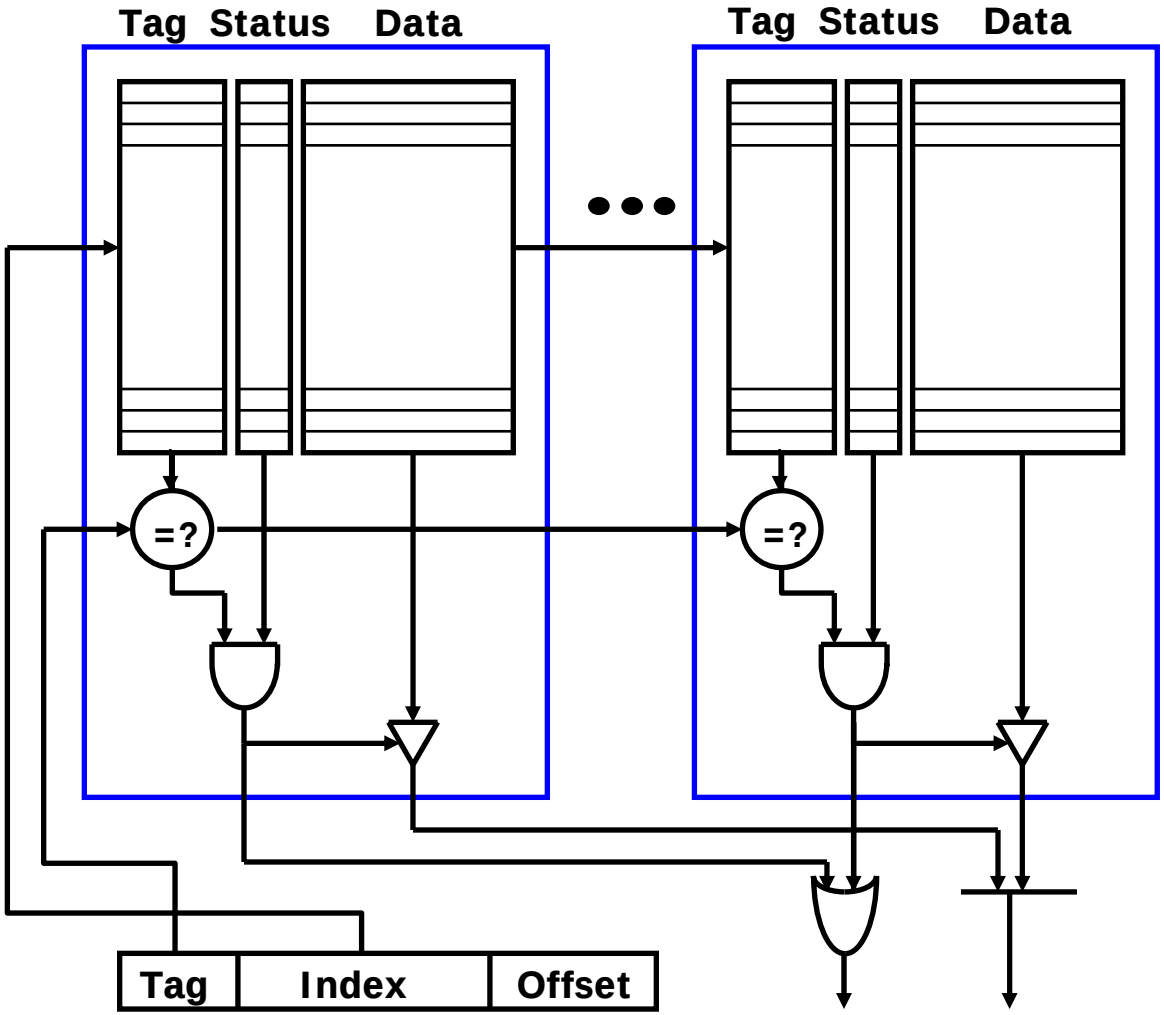
What are the disadvantages of increasing block size?

Block-level Optimizations

- Tags are too large, i.e., too much overhead
  - Simple solution: Larger blocks, but miss penalty could be large.
- Sub-block placement (aka sector cache)
  - A valid bit added to units smaller than the full block, called sub-blocks
  - Only read a sub-block on a miss
  - *If a tag matches, is the word in the cache?*

100	***1***		***1***		***1***		***1***	
300	***1***		***1***		***0***		***0***	
204	***0***		***1***		***0***		***1***	

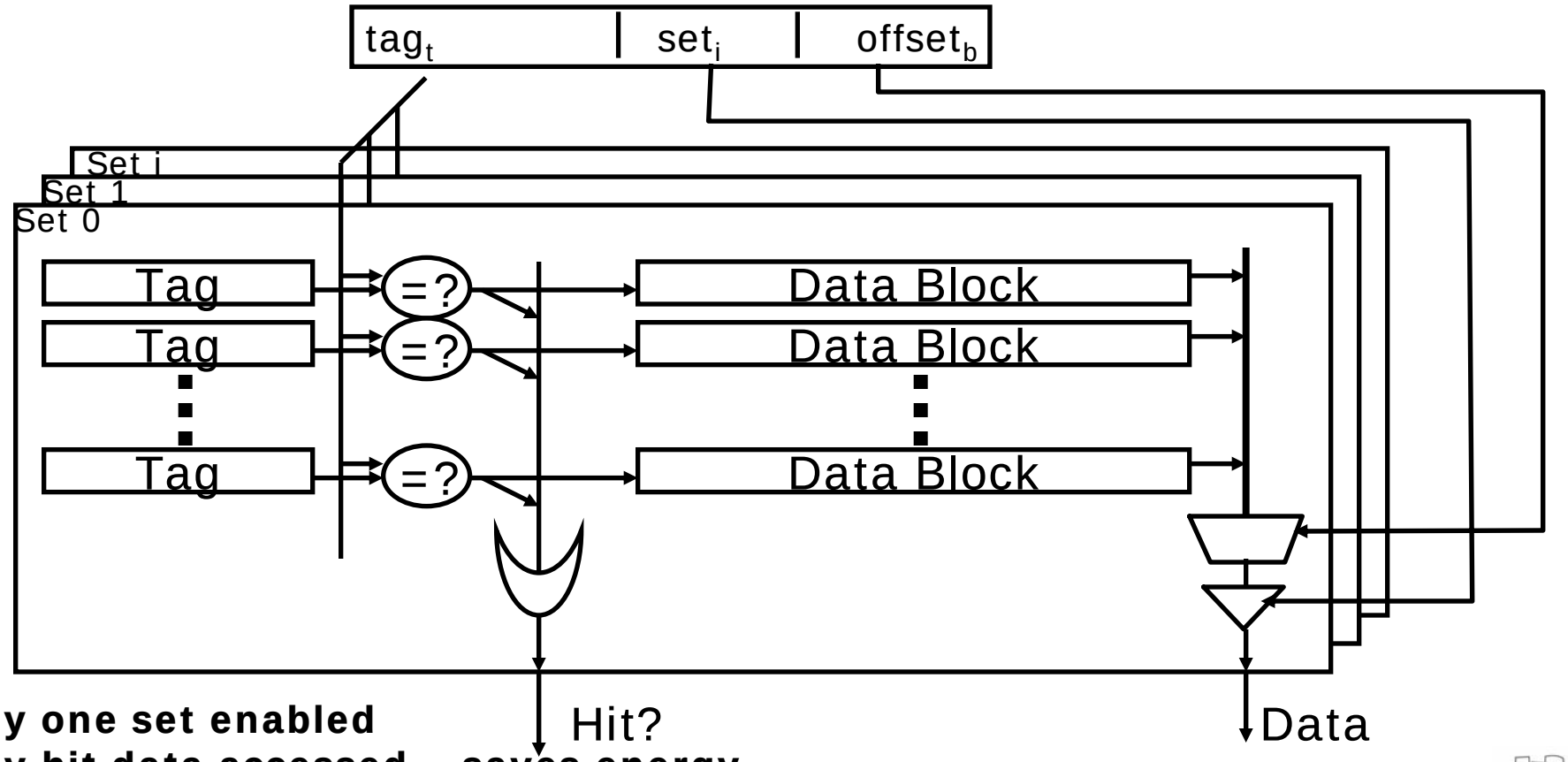
Set-Associative RAM-Tag Cache



- Not energy-efficient
  - A tag and data word is read from every way
- Two-phase approach
  - First read tags, then just read data from selected way
  - More energy-efficient
  - Doubles latency in L1
  - OK, for L2 and above, why?

Highly-Associative CAM-Tag Caches

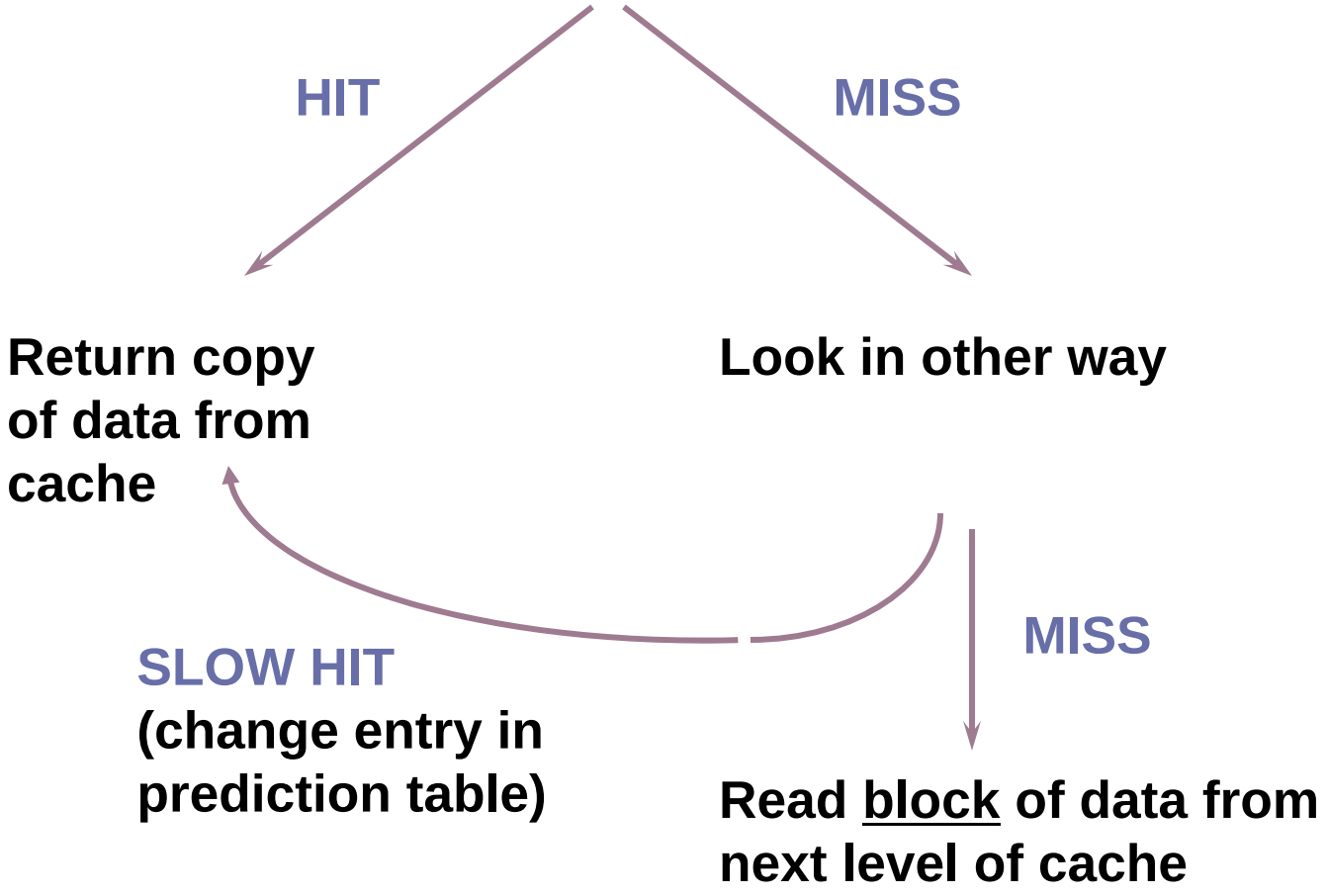
- For high associativity (e.g., 32-way), use content-addressable memory (CAM) for tags *(Intel XScale)*
- *Overhead: Tag+comparator bit 2-4x area of plain RAM-tag bit*



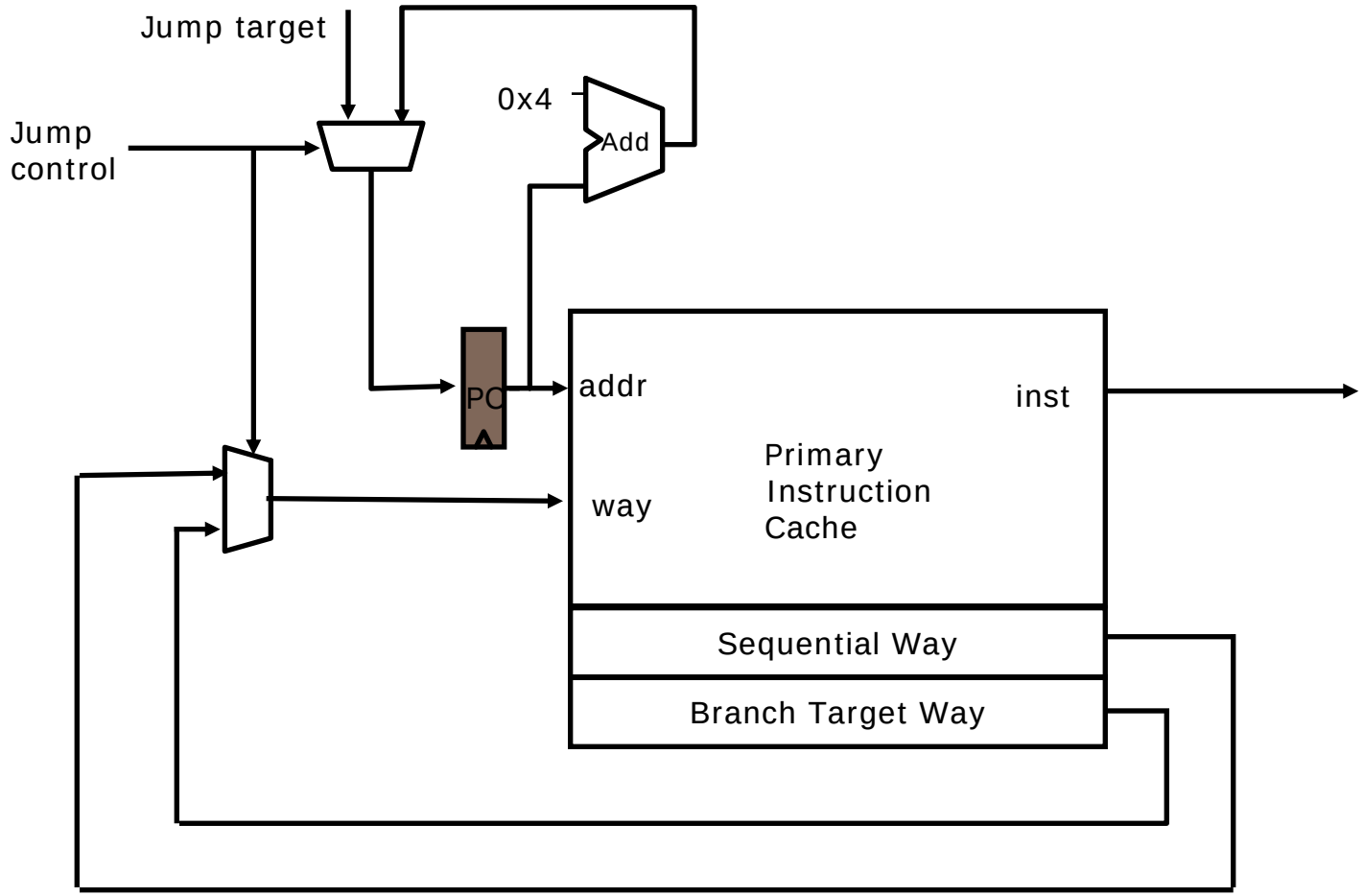
Only one set enabled
Only hit data accessed – saves energy

Way Predicting Caches *(MIPS R10000 L2 cache)*

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



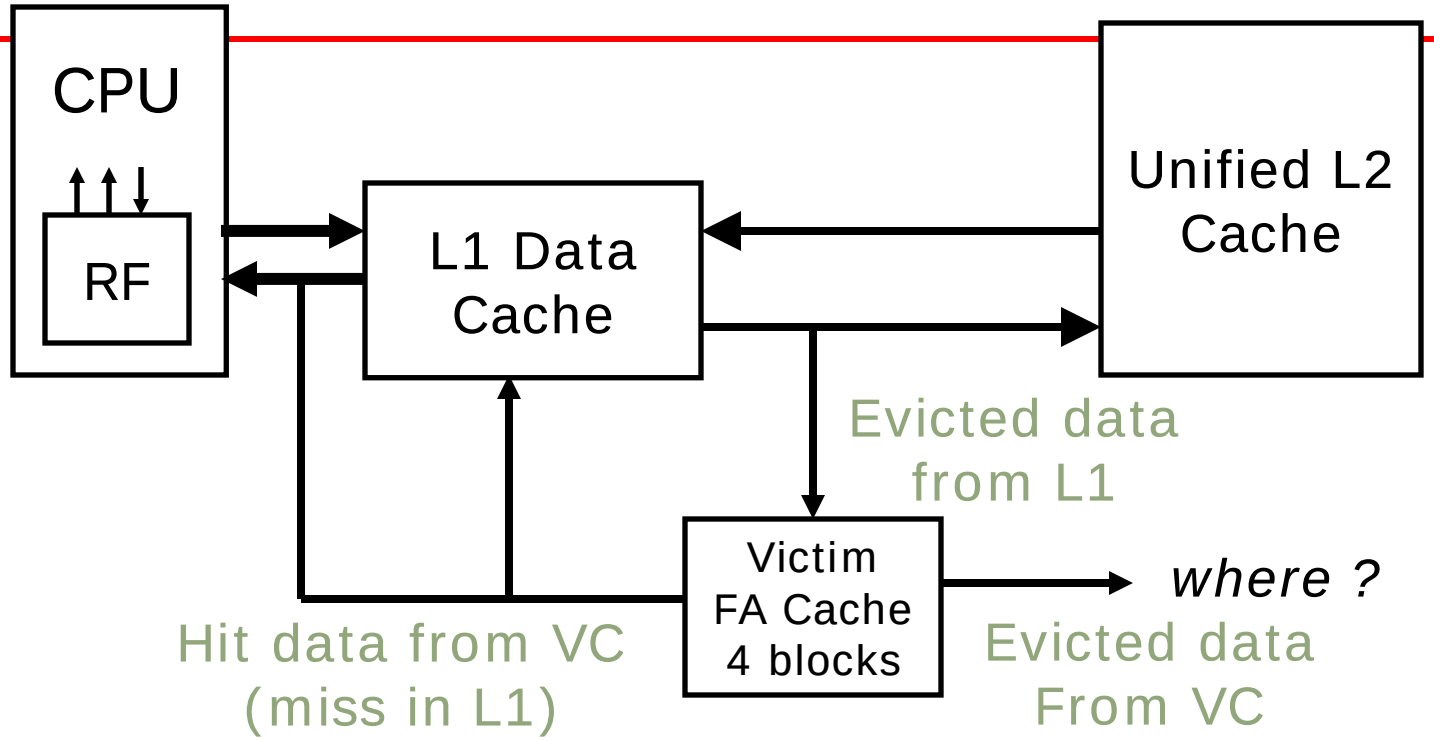
Way Predicting Instruction Cache (Alpha 21264-like)



CSAIL

Five-minute break to stretch your legs

Victim Caches (HP 7200)



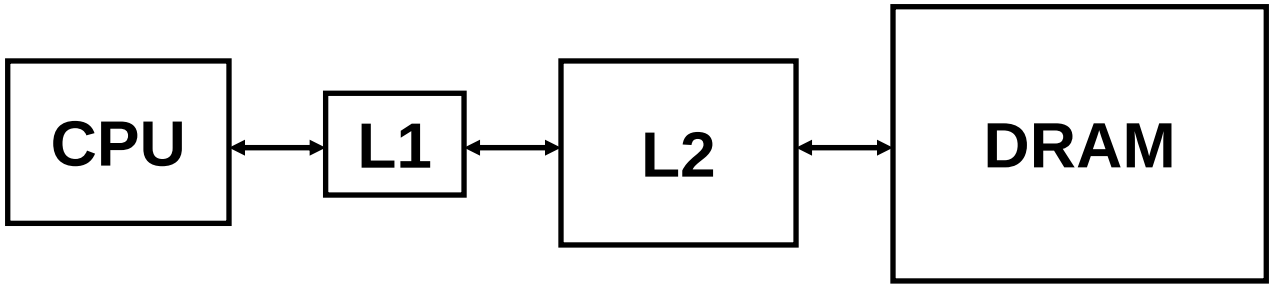
Victim cache is a small associative back up cache, added to a direct mapped cache, which holds recently evicted lines

- 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

Multilevel Caches

- A memory cannot be large and fast
- 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

Inclusion Policy

- Inclusive multilevel cache:
  - Inner cache holds copies of data in outer cache
  - Extra-CPU access needs only check outer cache
  - Most common case
- *Exclusive* multilevel caches:
  - Inner cache may hold data not in outer cache
  - Swap lines between inner/outer caches on miss
  - Used in Athlon with 64KB primary and 256KB secondary cache

Why choose one type of the other?

Itanium-2 On-Chip Caches (Intel/HP, 2002)

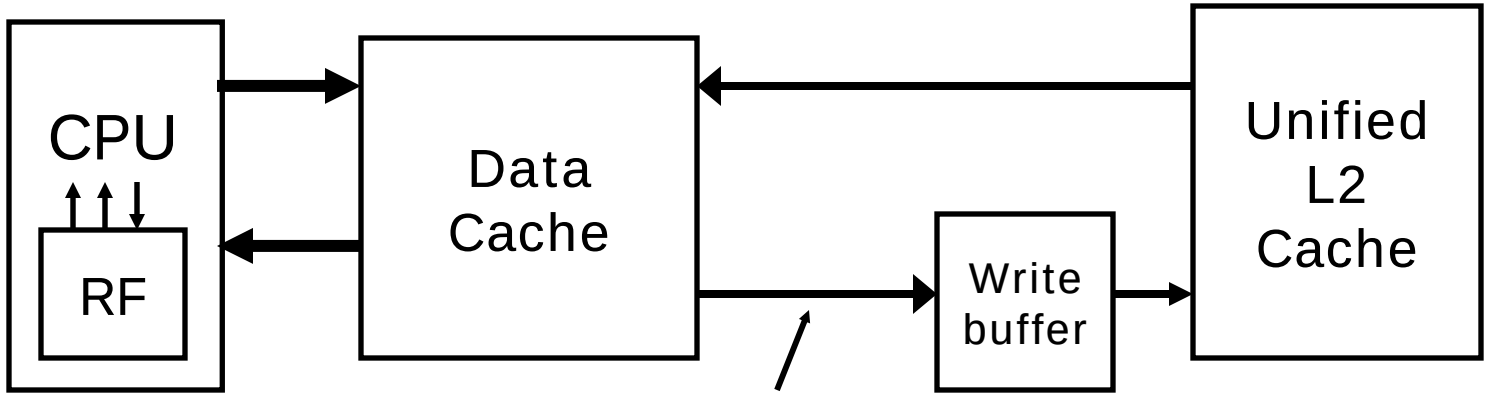
Image removed due to copyright restrictions.
To view image, visit http://www-vlsi.stanford.edu/group/chips_micropro_body.html

Level 1, 16KB, 4-way s.a., 64B line, quad-port (2 load+2 store), single cycle latency

Level 2, 256KB, 4-way s.a, 128B line, quad-port (4 load or 4 store), five cycle latency

Level 3, 3MB, 12-way s.a., 128B line, single 32B port, twelve cycle latency

Reducing Read Miss Penalty



Evicted dirty lines for writeback cache
OR
All writes in writethru cache

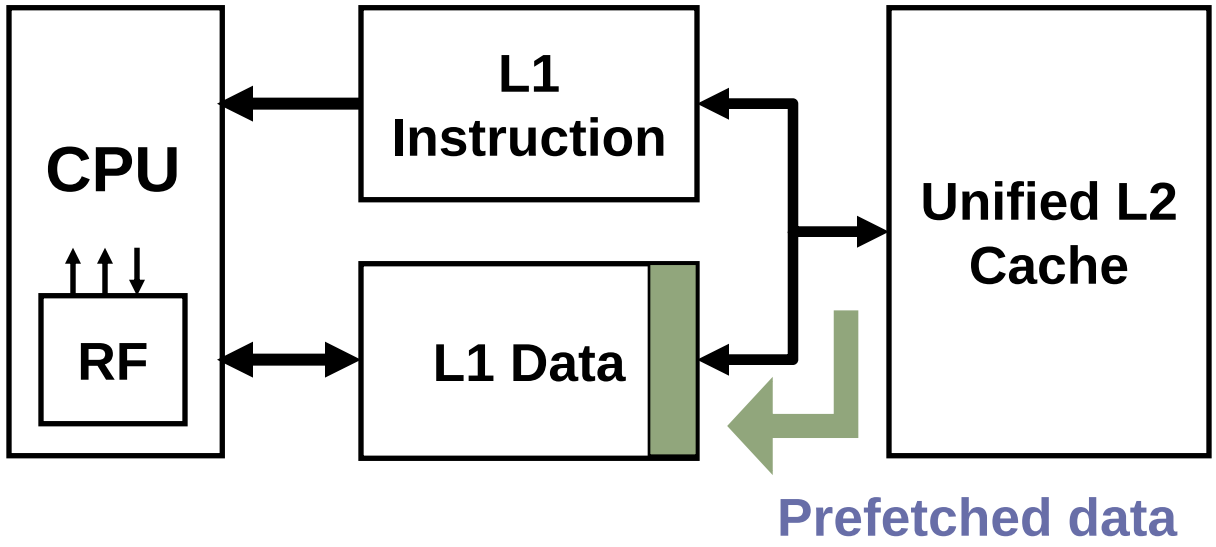
- Write buffer may hold updated value of location needed by a read miss
- Simple scheme: on a read miss, wait for the write buffer to go empty
- Faster scheme: Check write buffer addresses against read miss addresses, if no match, allow read miss to go ahead of writes, else, return value in write buffer

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?*

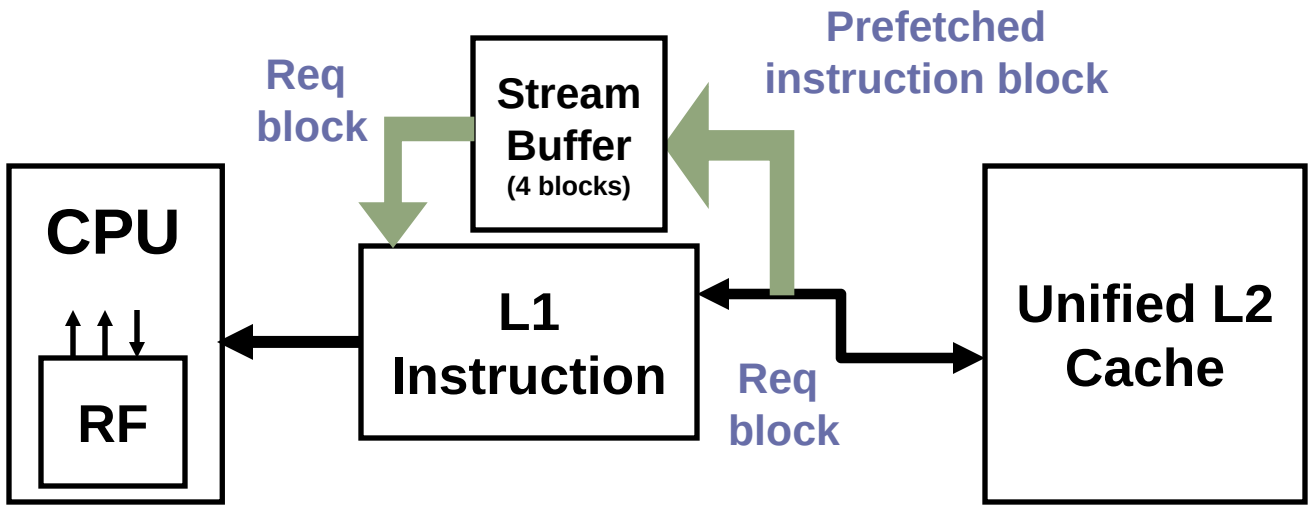
Issues in Prefetching

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



Hardware Instruction Prefetching

- Instruction prefetch in Alpha AXP 21064
  - Fetch two blocks on a miss; the requested block and the next consecutive block
  - Requested block placed in cache, and next block in instruction stream buffer



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 sequence of accesses to block b, b+N, b+2N, then prefetch b+3N etc.

Software Prefetching

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

- *What property do we require of the cache for prefetching to work ?*

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

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 (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

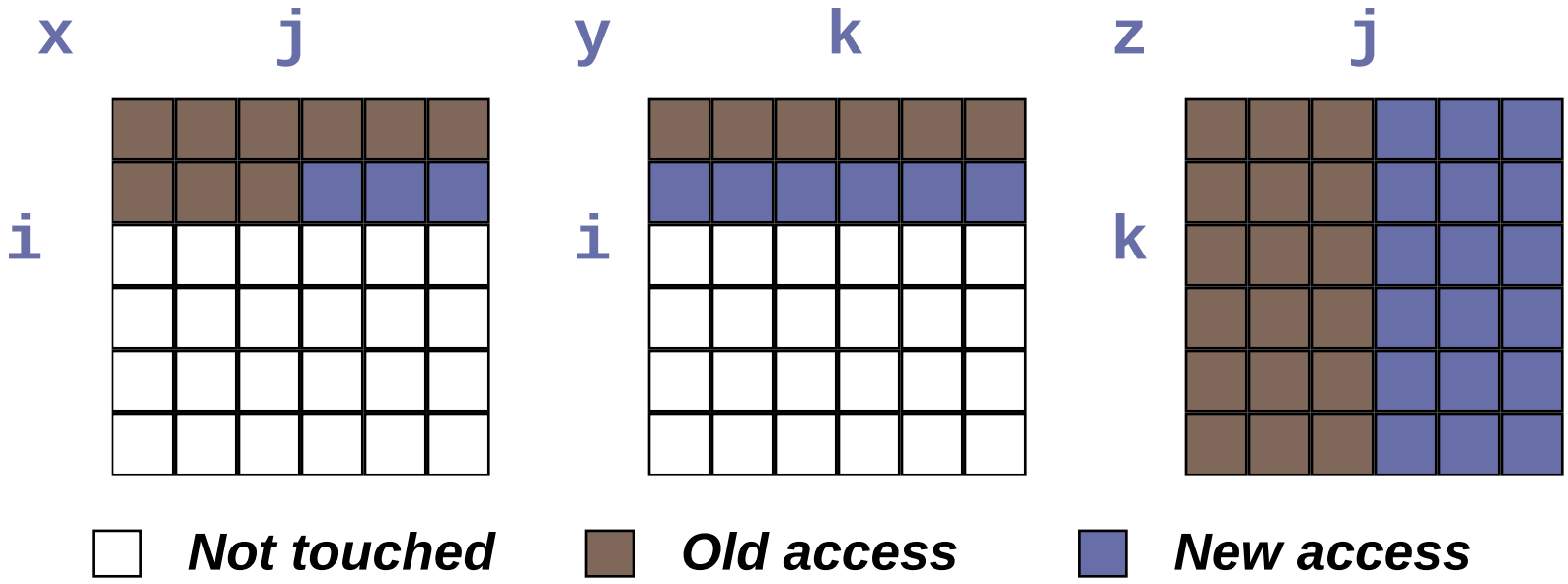
```
for(i=0; i < N; i++)
    for(j=0; j < M; j++)
       a[i][j] = b[i][j] * c[i][j];

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

What type of locality does this improve?

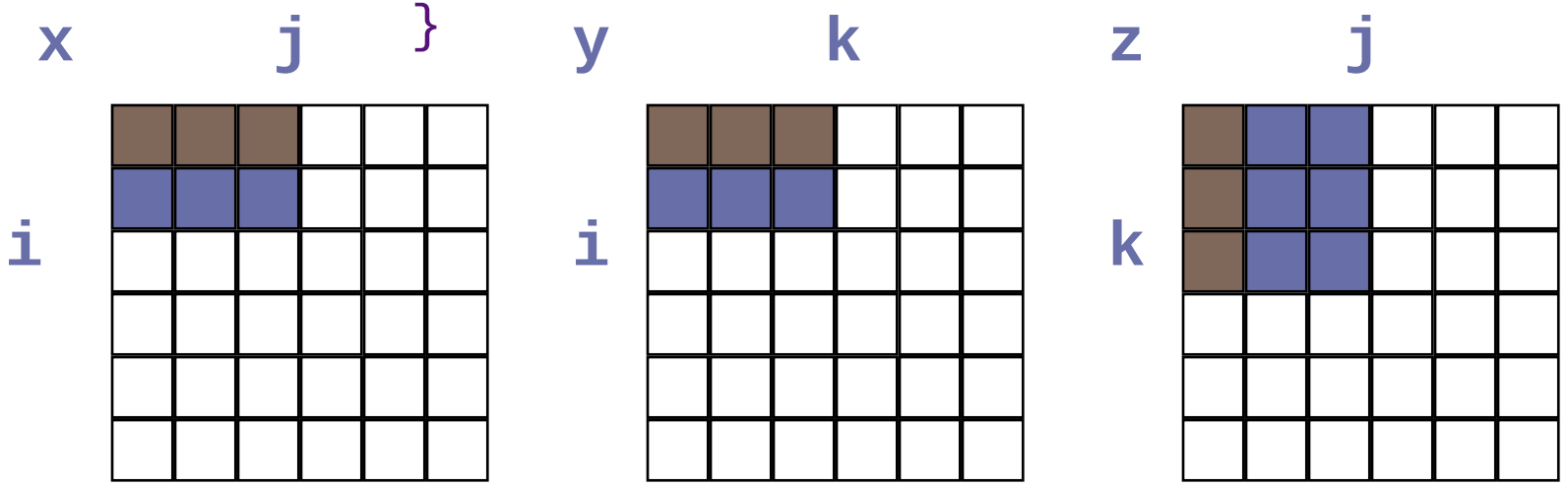
Blocking

```
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;
     }
```



Blocking

```
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?

Thank you !

Extras

Memory Hierarchy Example

- AlphaStation 600/5 desktop workstation
  - Alpha 21164 @ 333 MHz
  - On-chip L1 and L2 caches
  - L1 instruction cache, 8KB direct-mapped, 32B lines, fetch four instructions/cycle (16B)
  - Instruction stream prefetches up to 4 cache lines ahead
  - L1 data cache, 8KB direct-mapped, 32B lines, write-through, load two 8B words or store one 8B word/cycle (2 cycle latency)
  - up to 21 outstanding loads, 6x32B lines of outstanding writes
  - L2 unified cache, 96KB 3-way set-associative, 64B blocks/32B sub-blocks, write-back, 16B/cycle bandwidth (7 cycle latency)
  - Off-chip L3 unified cache, 8MB direct-mapped, 64B blocks, peak bandwidth is 16B every 7 cycles (15 cycle latency)
  - DRAM, peak bandwidth 16B every 10 cycles (60 cycle latency)

Further Issues

There are several other factors that are intimately connected with cache design:

- Virtual memory and associated address translation

- Multiprocessor and associated memory model issues - *cache coherence*

stay tuned