Lectures:15-17 (Caches Continued)
CS422-Spring 2020

Biswa@CSE-IITK

Design Issues: Unified vs Split

- Unified:
  - + Dynamic sharing of cache space: no overprovisioning that might happen with static partitioning (i.e., split I and D caches)
  - -- Instructions and data can thrash each other (i.e., no guaranteed space for either)
  - -- I and D are accessed in different places in the pipeline. Where do we place the unified cache for fast access?

- First level caches are almost always split
  - for the last reason above
- Second and higher levels are almost always unified

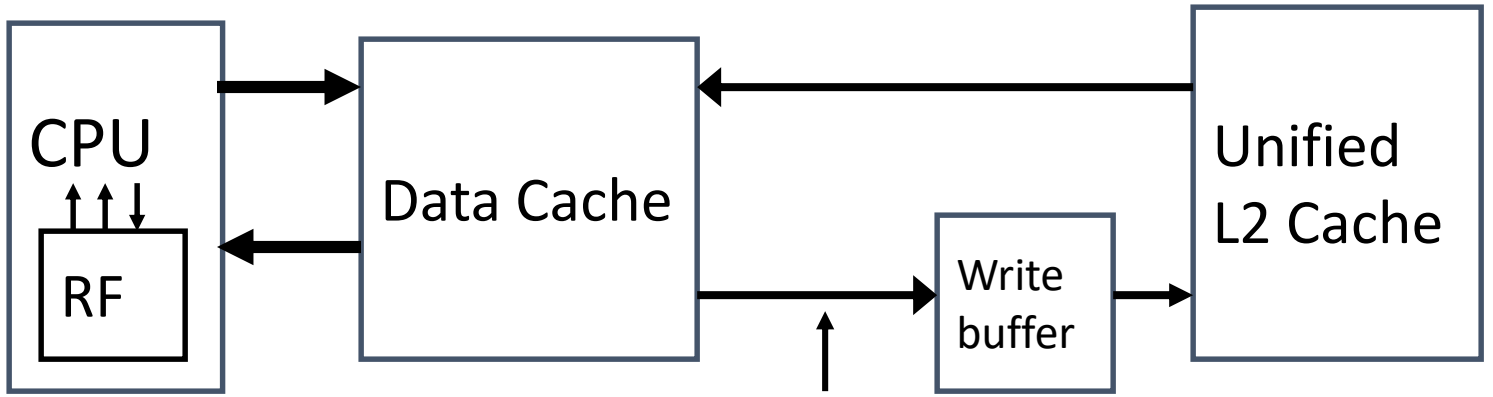
Reads are not Writes

- If a write enters the cache, what happens if
  - There is a cache miss
    - Does the cache need to bring in the cache line?
  - There is a cache hit
    - Does the cache need to write back to memory?

Write Policies

- 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:
  - ***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

Write Buffers



Evicted dirty lines for writeback cache
OR
All writes in writethrough 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

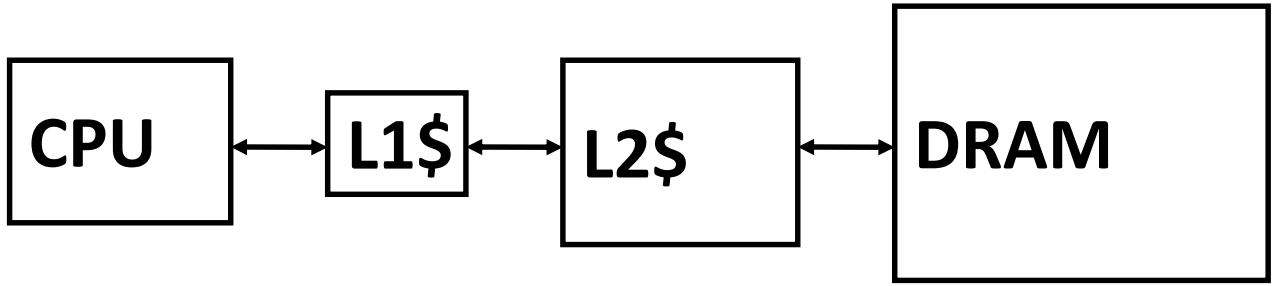
Simple solution: on a read miss, wait for the write buffer to go empty

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

Multi-level 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

Performance: AMAT

Average memory access time (AMAT) = Hit time + Miss rate x Miss penalty

Average memory access time (AMAT) = Hit time + Miss $rate_1$ x Miss $penalty_1$
+ Miss $rate_2$ x Miss $penalty_2$

Improving Cache Performance

Average memory access time (AMAT) =

Hit time + Miss rate x Miss penalty

To improve performance:

- reduce the hit time
- reduce the miss rate
- reduce the miss penalty

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]

The 3Cs

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*

Cache Knobs and Performance

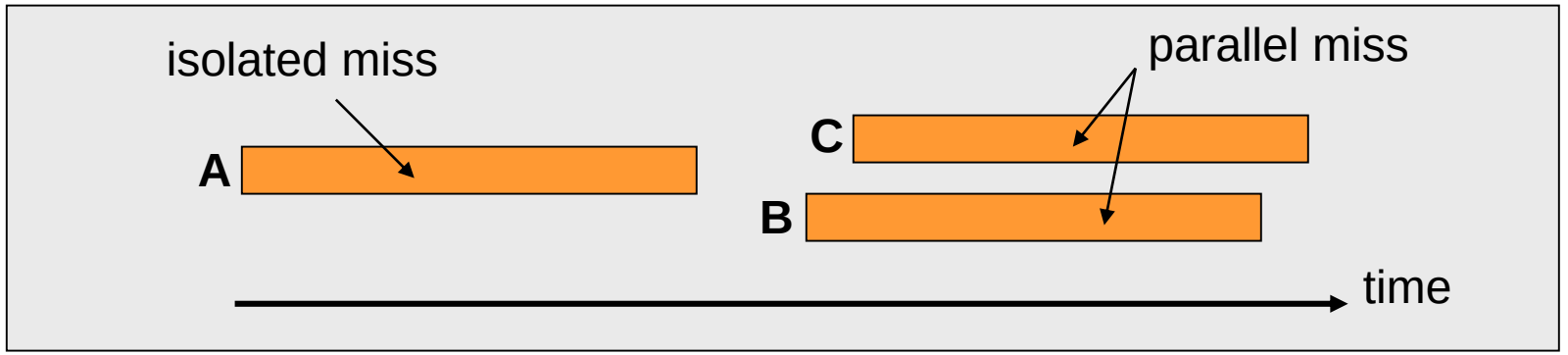
- Larger cache size
  + reduces capacity and conflict misses
  - hit time will increase

- Higher associativity
  + reduces conflict misses
  - may increase hit time

- Larger line size
  + reduces compulsory and capacity (reload) misses
  - increases conflict misses and miss penalty

Non-blocking Cache

- Enable cache access when there is a pending miss
- Enable multiple misses in parallel
  - Memory-level parallelism (MLP)
    - generating and servicing multiple memory accesses in parallel
  - Why generate multiple misses?

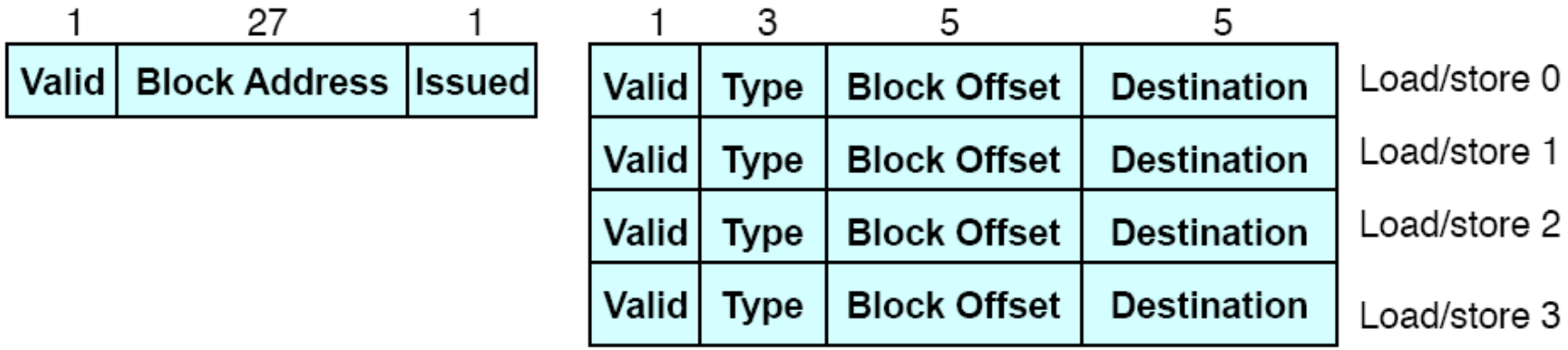


    - Enables latency tolerance: overlaps latency of different misses
  - How to generate multiple misses?
    - Out-of-order execution, multithreading, prefetching

Miss-Status Holding Registers

- Also called “miss buffer”
- Keeps track of
  - Outstanding cache misses
  - Pending load/store accesses that refer to the missing cache block
- Fields of a single MSHR
  - Valid bit
  - Cache block address (to match incoming accesses)
  - Control/status bits (prefetch, issued to memory, which subblocks have arrived, etc)
  - Data for each subblock
  - For each pending load/store
    - Valid, type, data size, byte in block, destination register or store buffer entry address

MSHRs


1
27
1
Valid
Block Address
Issued
1
3
5
5
Valid
Type
Block Offset
Destination
Load/store 0
Valid
Type
Block Offset
Destination
Load/store 1
Valid
Type
Block Offset
Destination
Load/store 2
Valid
Type
Block Offset
Destination
Load/store 3

MSHR in Action

- On a cache miss:
  - Search MSHR for a pending access to the same block
    - Found: Allocate a load/store entry in the same MSHR entry
    - Not found: Allocate a new MSHR
    - No free entry: stall

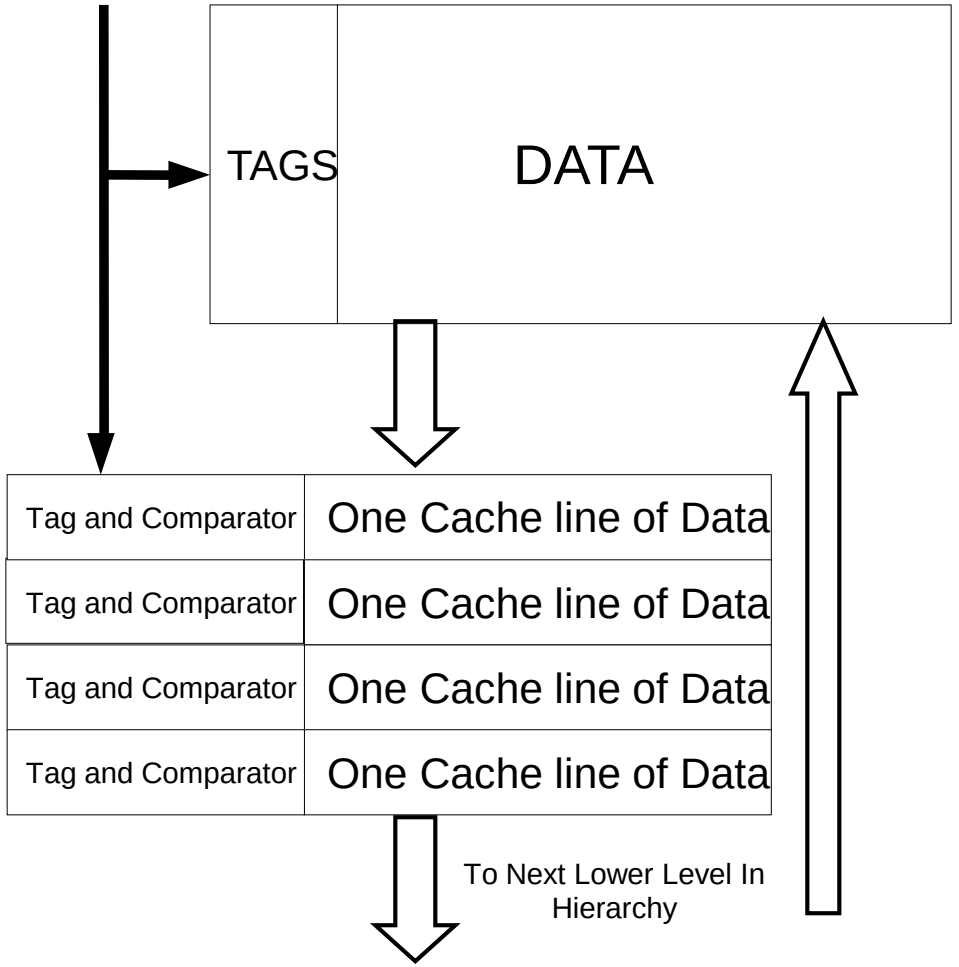
- When a subblock returns from the next level in memory
  - Check which loads/stores waiting for it
    - Forward data to the load/store unit
    - Deallocate load/store entry in the MSHR entry
  - Write subblock in cache or MSHR
  - If last subblock, dellaocate MSHR (after writing the block in cache)

Early Restart and CWF

- **Don’t wait for full block to be loaded 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. Also called *wrapped fetch* and *requested word first***
- **Generally useful only in large blocks,**
- **Spatial locality a problem; tend to want next sequential word, so not clear if benefit by early restart**

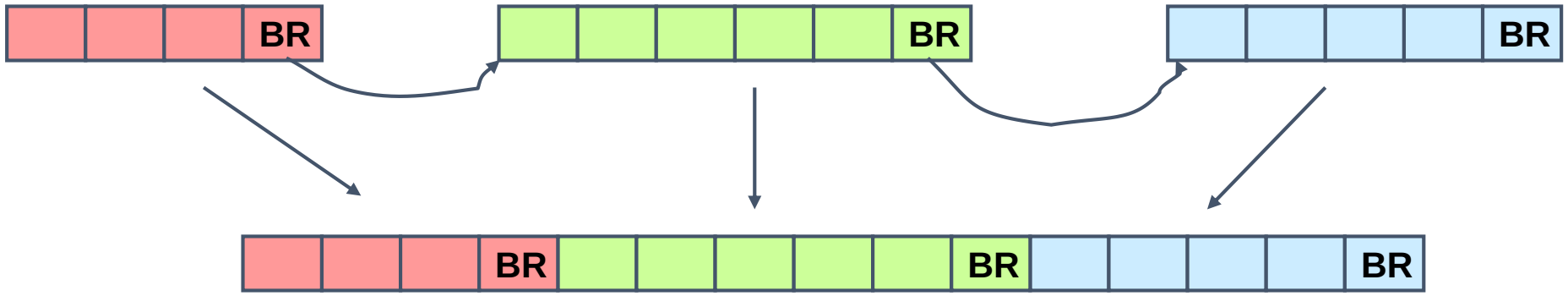
Victim Cache

- 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
- Used in Alpha, HP machines



Trace Cache

Key Idea: Pack multiple non-contiguous basic blocks into one contiguous trace cache line



- **Single fetch brings in multiple basic blocks**
- **Trace cache indexed by start address *and* next *n* branch predictions**

Multi-Banked Cache

- **Rather than treat the cache as a single monolithic block, divide into independent banks that can support simultaneous accesses**
  - **E.g.,T1 ("Niagara") L2 has 4 banks**
- **Banking works best when accesses naturally spread themselves across banks ⇒ mapping of addresses to banks affects behavior of memory system**
- **Simple mapping that works well is "sequential interleaving"**
  - **Spread block addresses sequentially across banks**
  - **E,g, if there 4 banks, Bank 0 has all blocks whose address modulo 4 is 0; bank 1 has all blocks whose address modulo 4 is 1; …**

Way Prediction

- ***Way prediction*** helps select one block among those in a set, thus requiring only one tag comparison (if hit).
  - Preserves advantages of direct-mapping (why?);
  - In case of a miss, other block(s) are checked.
- ***Pseudoassociative*** (also called column associative) caches
  - Operate exactly as direct-mapping caches when hit, thus again preserving advantages of the direct-mapping;
  - In case of a miss, another block is checked (as if in set-associative caches), by simply inverting the most significant bit of the index field to find the other block in the "pseudoset".
  - real hit time < pseudo-hit time

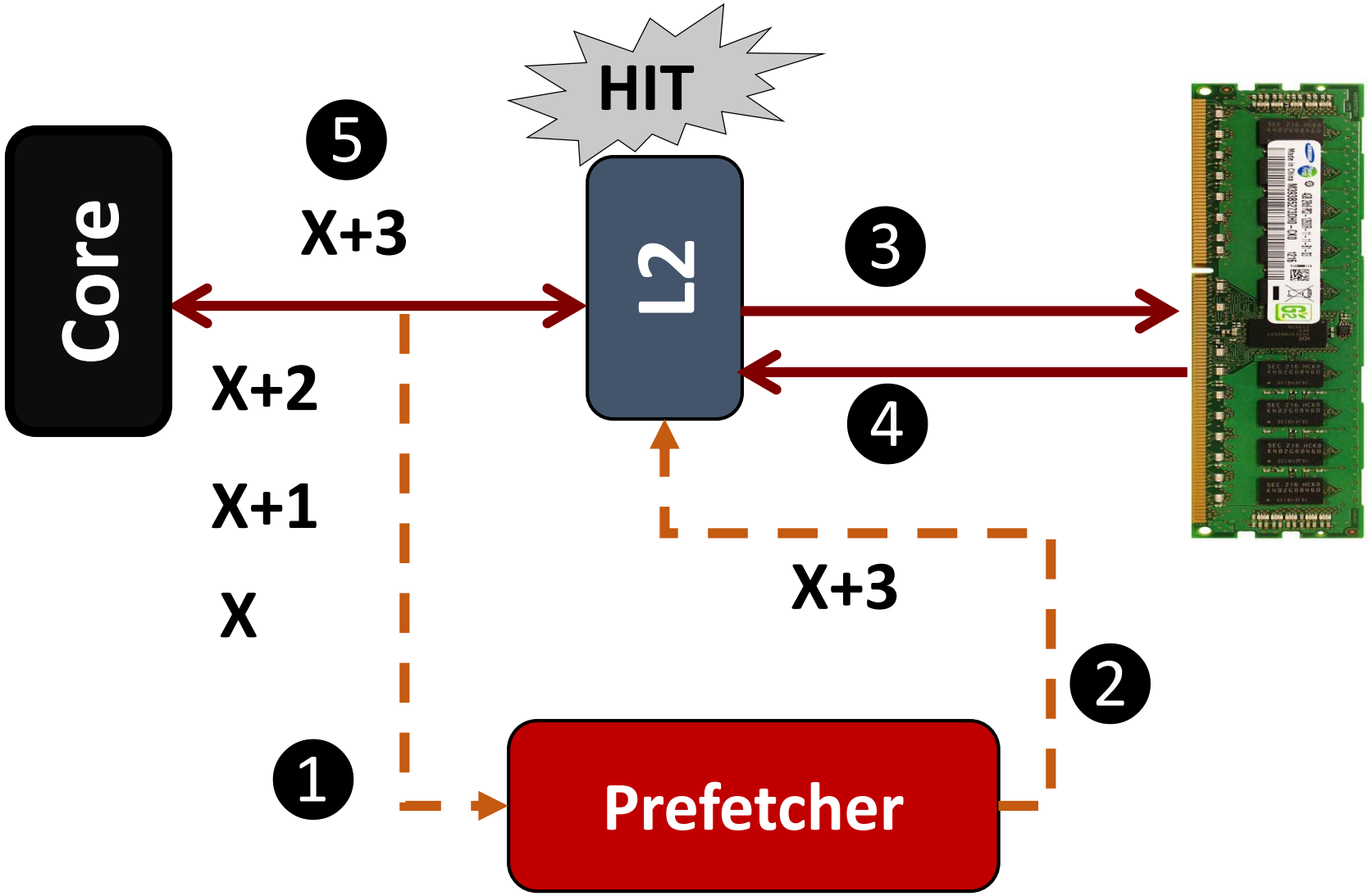
Hardware Prefetching

What?
Latency-hiding technique - Fetches data before the core demands.

Why?
Off-chip DRAM latency has grown up to 400 to 800 cycles.

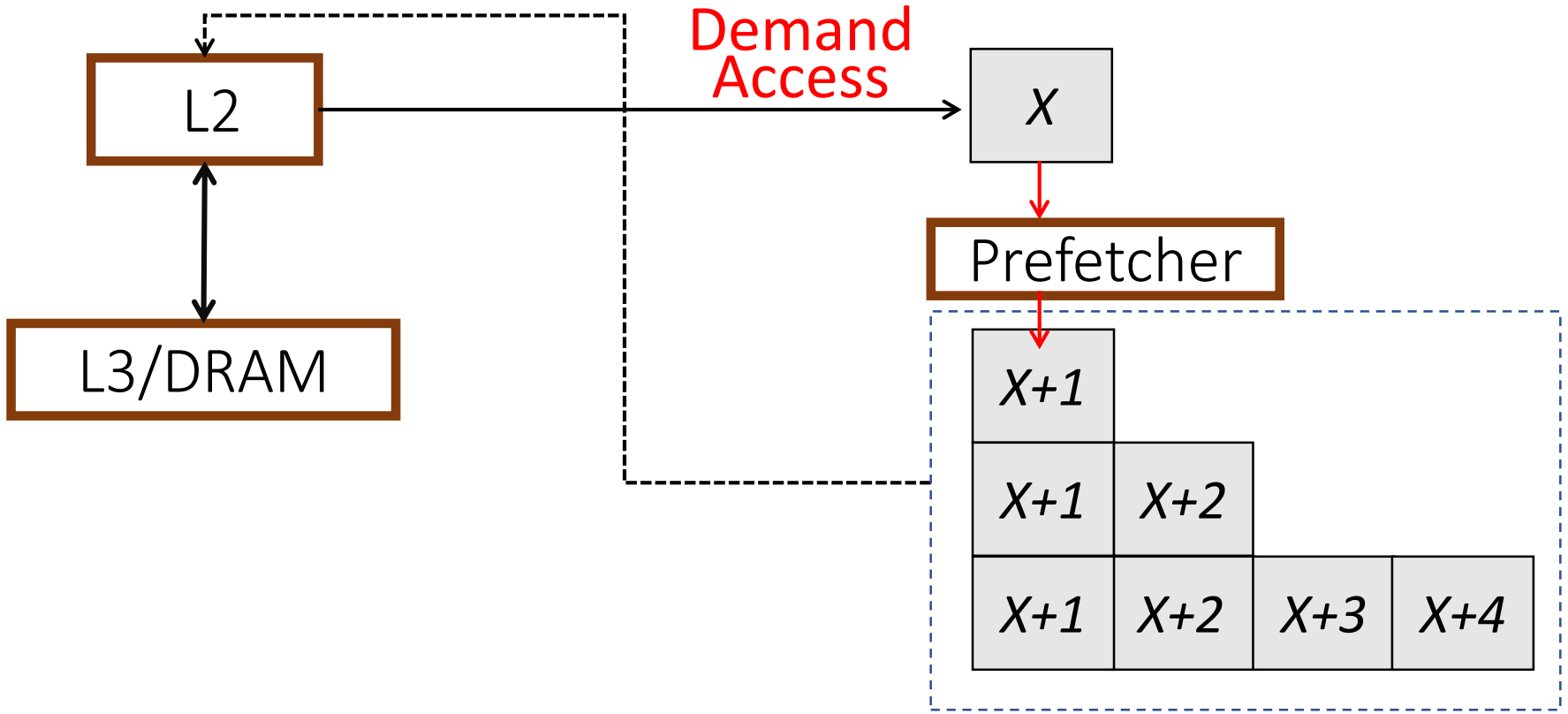
How?
By observing/predicting the demand access (LOAD/STORE) patterns.

Prefetch Engine



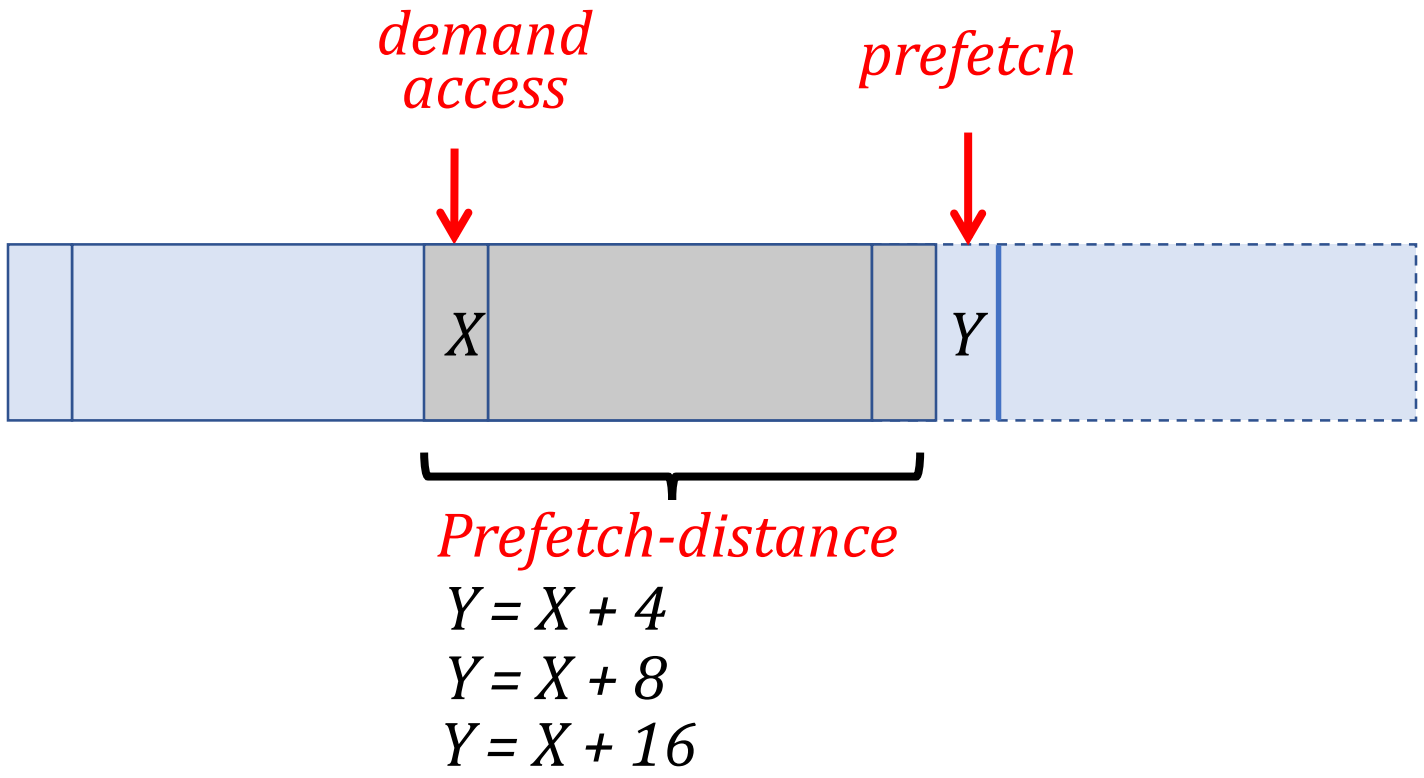
Prefetch Degree

Prefetch Degree: Number of prefetch requests to issue at a given time.



Prefetch Distance

Prefetch Distance: How far ahead of the demand access stream are the prefetch requests issued?

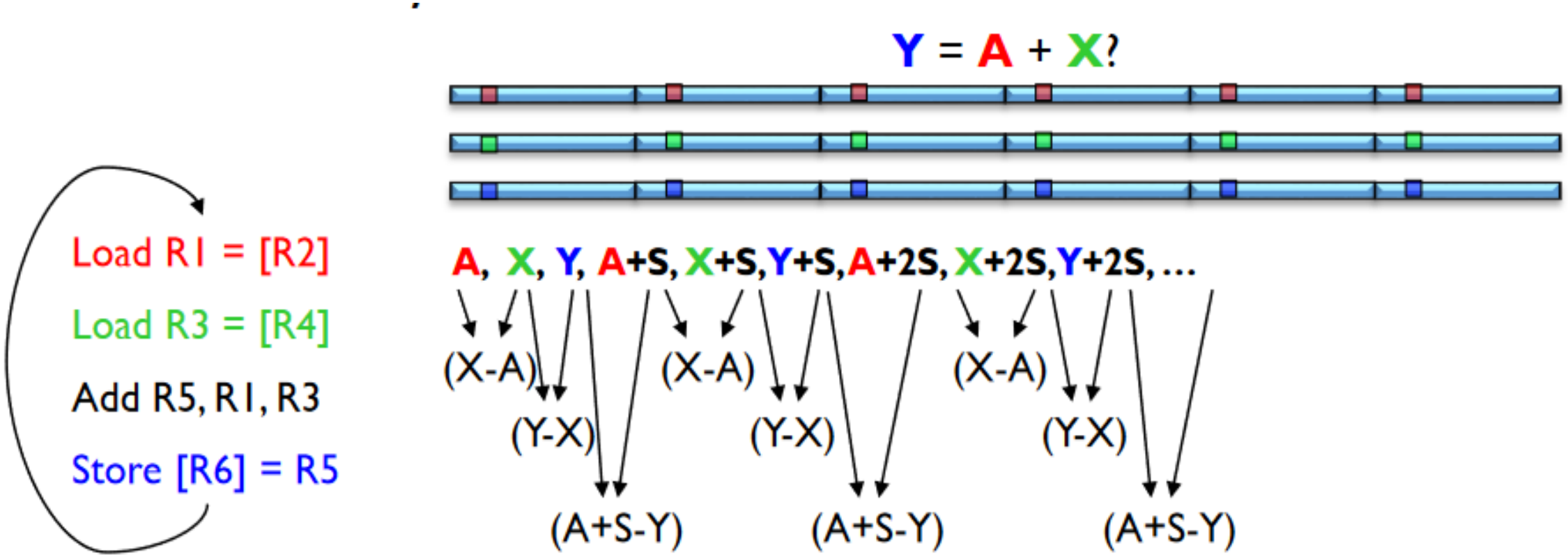


Next-line Prefetcher

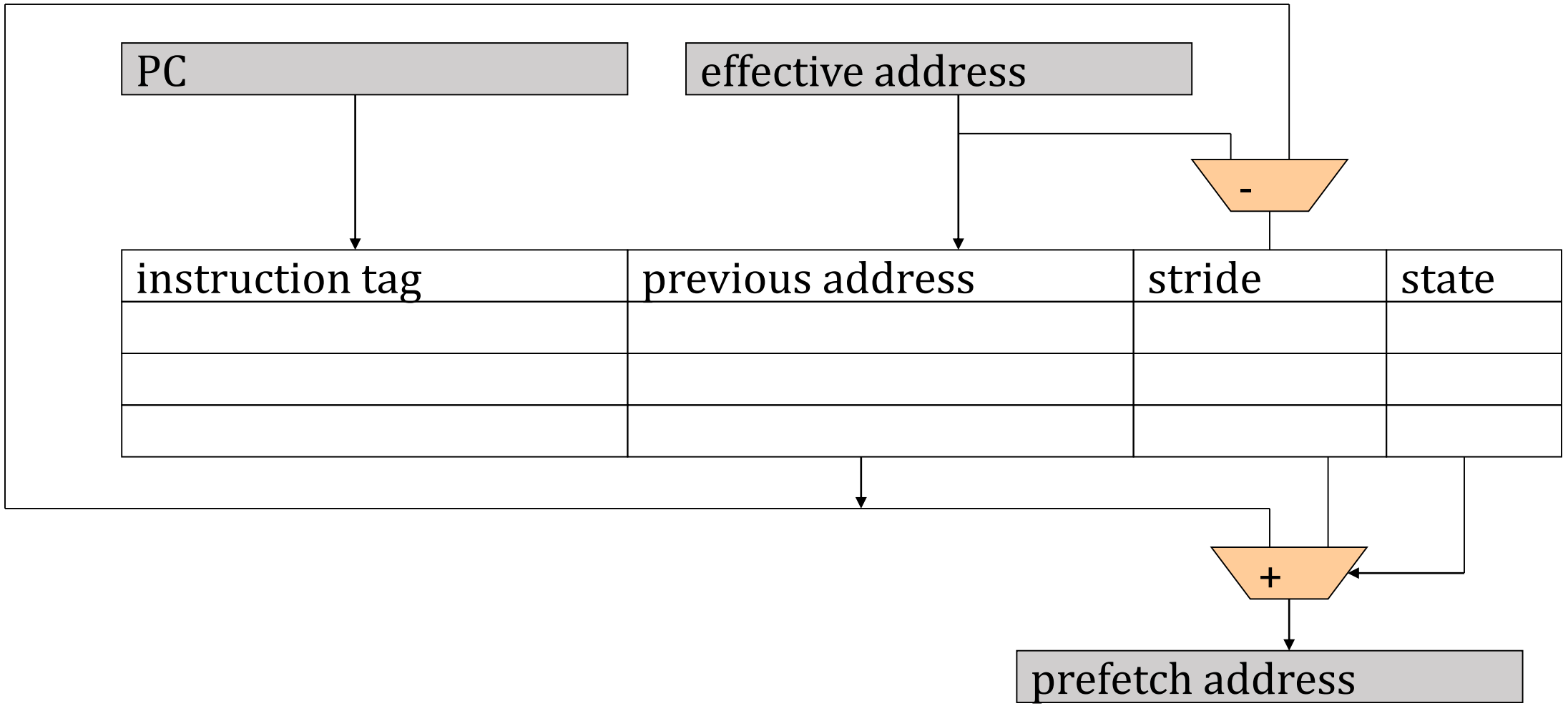
Next Line: Miss to cache block X , prefetch X+1. Degree=1, Distance=1

Works well for L1 Icache and L1 Dcache.

What About This?



Stride Prefetching



Quantifying Prefetchers

$$\text{PrefetchAccuracy}(i) = \frac{\text{Prefetch}_{\text{hits}}(i)}{\text{Prefetch}_{\text{issued}}(i)}$$

Prefetched Block in the Cache.

$$\text{Lateness}(i) = \frac{\text{Prefetch}_{\text{late}}(i)}{\text{Prefetch}_{\text{hits}}(i)}$$

Prefetched Block Still on its way

$$\text{Pollution}(i) = \frac{\text{LLC Poll}(i)}{\text{Demand}_{\text{misses}}(i)}$$

Prefetched Block evicted a demand block that will be reused

$$\text{Coverage}(i) = \frac{\text{Prefetch Hits}(i)}{\text{Prefetch Hits}\,(i) + \text{Demand}_{\text{misses}}(i)}$$

Fraction of misses avoided