

CS622: Advanced Computer Architecture

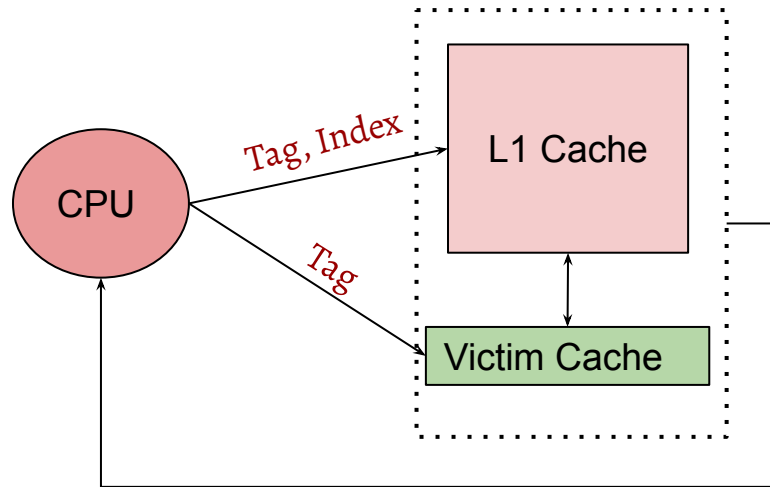
MLP Improvements

Reducing hit time

- Small caches with lower associativity $\Rightarrow$ Reduced lookup time
- How to achieve lookup performance of direct mapped caches while reducing conflict misses?

Reducing hit time

- Small caches with lower associativity $\Rightarrow$ Reduced lookup time
- How to achieve lookup performance of direct mapped caches while reducing conflict misses?



Victim cache (proposed by Jouppi in 1990)

- Direct mapped L1 cache with a fully associative small cache to hold evicted blocks
- Shown to be effective in reducing conflicts

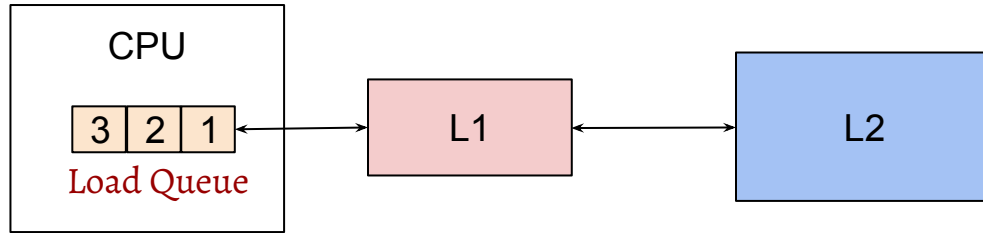
Reducing hit time

- Small caches with lower associativity $\Rightarrow$ Reduced lookup time
- How to achieve lookup performance of direct mapped caches while reducing conflict misses?

Way prediction

- Extra bits (e.g., address-bits) to predict the way in a set-associative cache for the access
- Start parallel tag comparison and data read from the predicted cache line
- If mispredicted, check other blocks (extra cycles needed)
- Used in MIPS R10000 and ARM Cortex-A8

Boosting Cache Bandwidth: Non-blocking caches



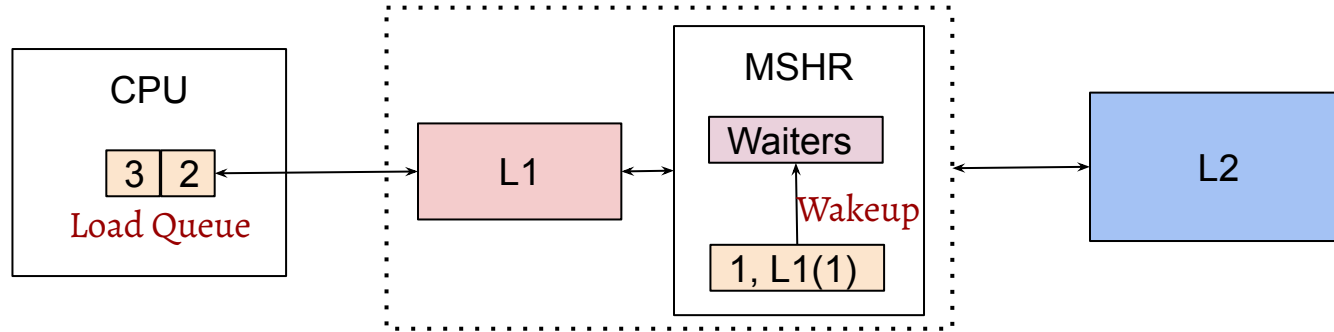
If '1' is a miss at L1, '2' can not be issued till the miss is handled at lower layers

Even worse, if '2' and '3' are cache hits at L1, may result in unnecessary stalls

- Pipelined caches can help. Support for reordering needed, limited by speed of the mem-side
- How to allow cache lookup operations to proceed while there are outstanding cache misses?

Non-blocking caches with MSHR

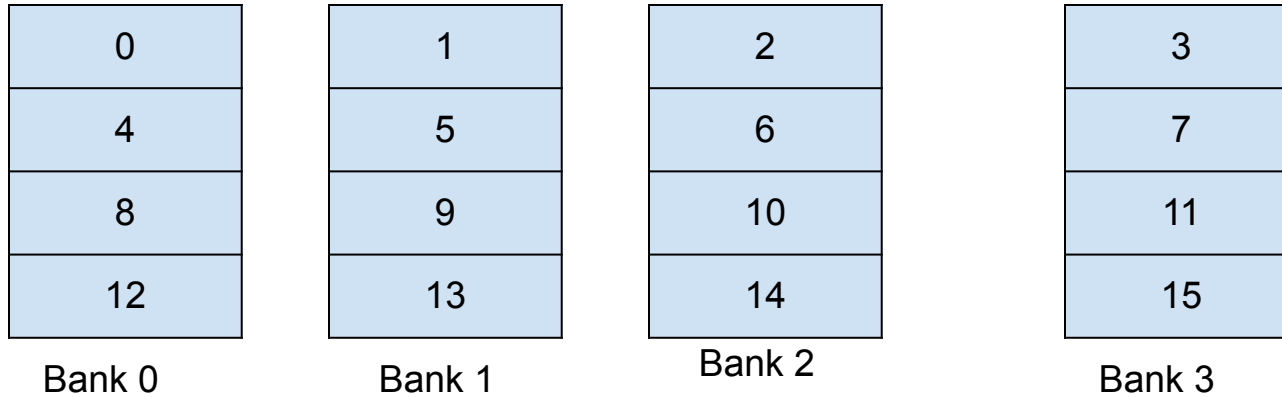
- How to allow cache lookup operations to proceed while there are outstanding cache misses?



- Miss status holding register (MSHR) holds information for outstanding cache misses
- On L1 miss, allocate a MSHR entry or merge with an existing entry
- MSHR allows lookup and merge (reduced bandwidth usage)
- Wake up waiting requests on arrival of required sub blocks from down the hierarchy
- Stall only when MSHR is full

Increase cache bandwidth with multiple banks

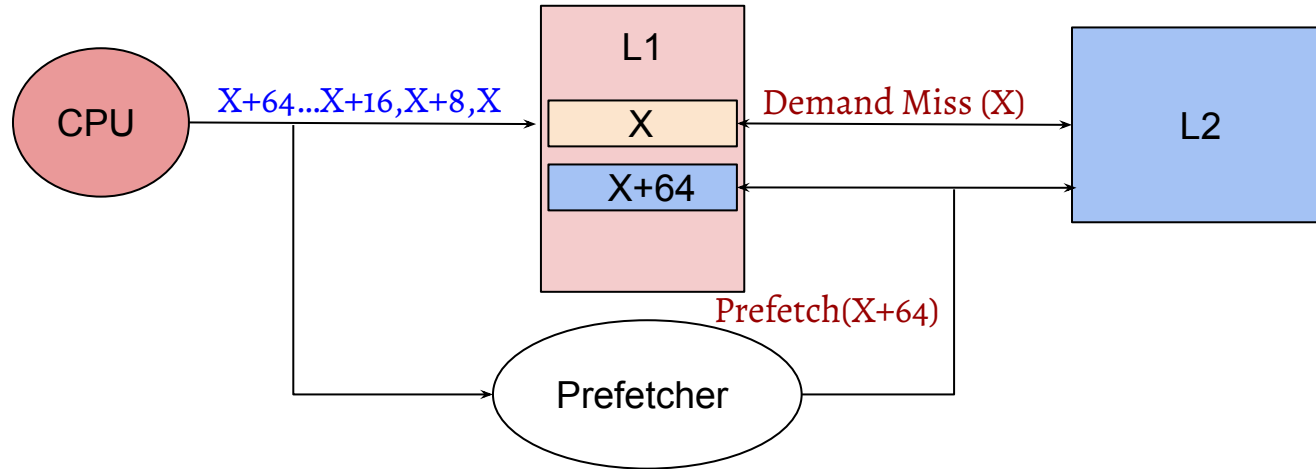
- Can we support more than one concurrent cache operations?



- Split the monolithic cache into banks (used in ARM and Intel)
- Maximum throughput when accesses spread across banks
- Sequential interleaving works well in many cases

Reducing Miss Penalty: Hardware prefetching

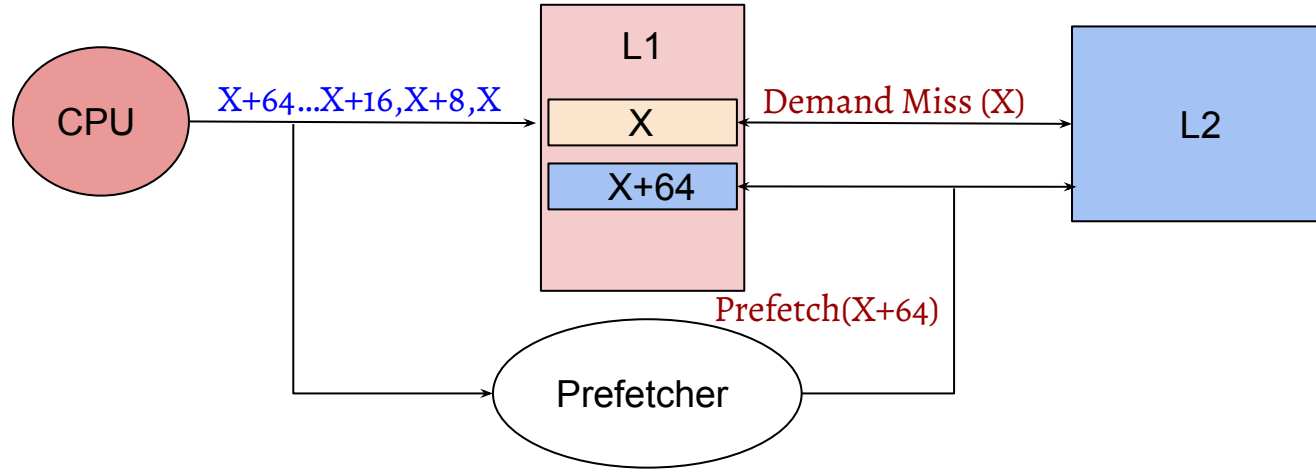
- Prefetch blocks before they are required by the CPU by predicting future access



- Design parameters: Degree of prefetch, prefetch distance
- Policy: which address to prefetch? Next line prefetcher, Stride prefetcher
- How good is a prefetcher?

Hardware prefetching

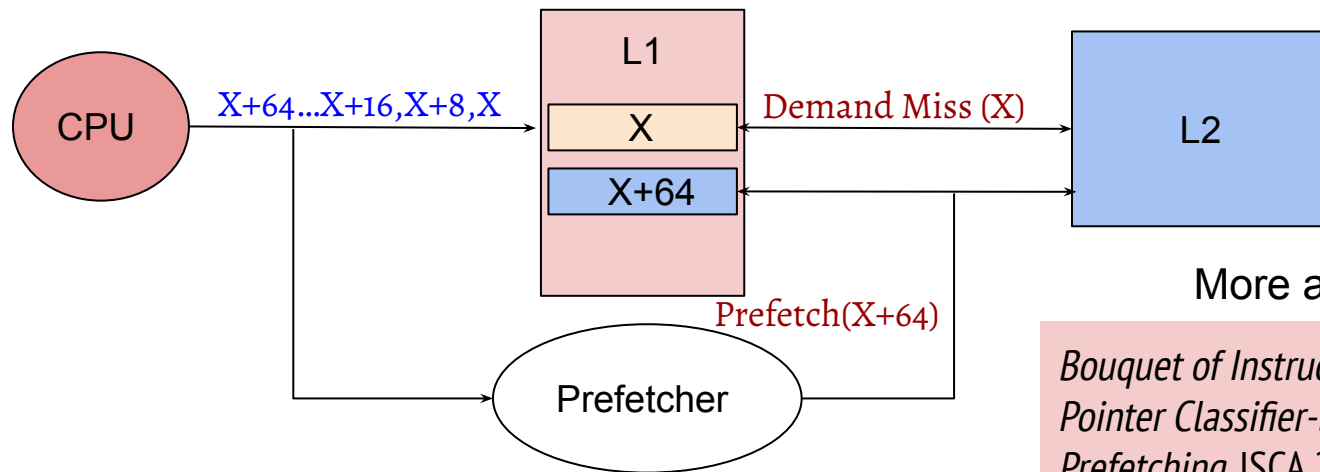
- Prefetch blocks before they are required by the CPU by predicting future access



- How good is a prefetcher?
 - Prefetch accuracy (Hits due to prefetch / Total prefetches)
 - Pollution (Misses due to eviction caused by the prefetcher)
 - Timeliness (Prefetched block status: in cache or in flight)

Hardware prefetching

- Prefetch blocks before they are required by the CPU by predicting future access

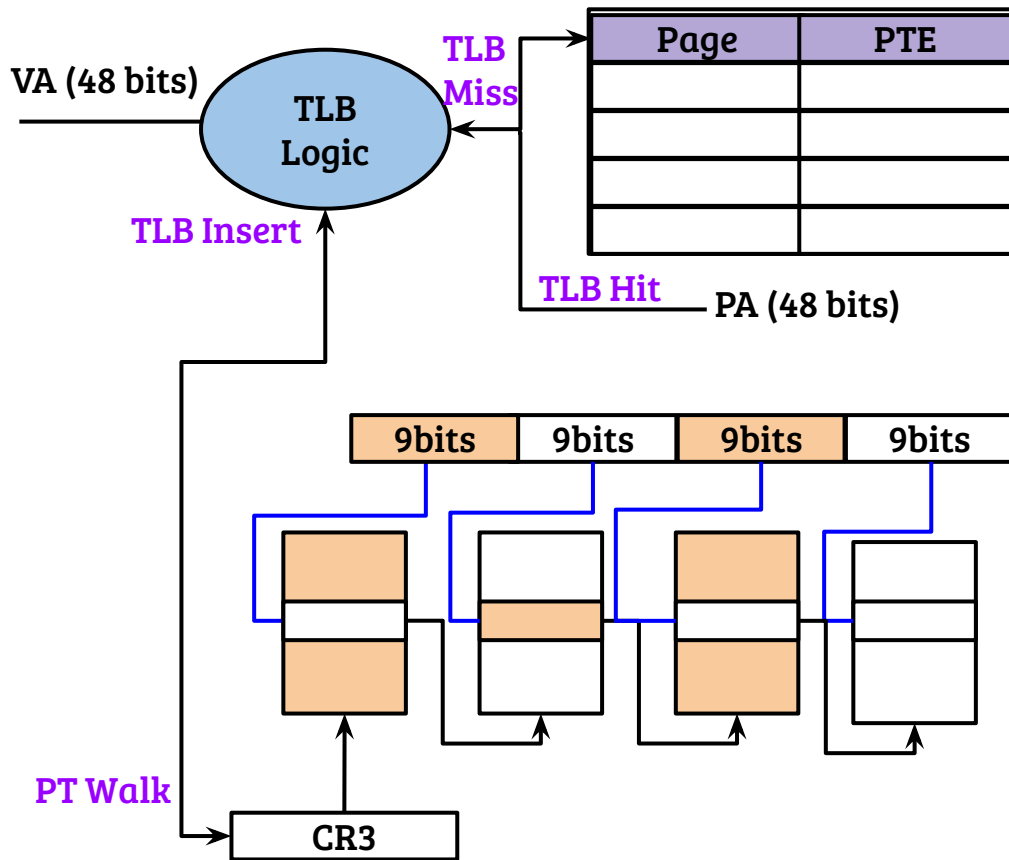


More about prefetchers

Bouquet of Instruction Pointers: Instruction Pointer Classifier-based Spatial Hardware Prefetching, ISCA 2020

- How good is a prefetcher?
 - Prefetch accuracy (Hits due to prefetch / Total prefetches)
 - Pollution (Misses due to eviction caused by the prefetcher)
 - Timeliness (Prefetched block status: in cache or in flight)

Address translation (TLB + PTW)



- TLB in the path of address translation
- Separate TLBs for instruction and data, multi-level TLBs
- In X86, OS can not make entries into the TLB directly, it can flush entries

Memory access (simplified serial view)

- Are there any differences in loading the data into L1 in two cases?
 - TLB Hit?
 - TLB Miss?

```
Addr Translate(V){  
    PageAddress P = V >> 12;  
    TLBEntry entry = lookup(P);  
    if (entry.valid) return entry.pte;  
    entry = PageTableWalk(V);  
    MakeEntry(entry);  
    return entry.pte;  
}  
Load(V){  
    Addr PA = Translate( V);  
    Load_L1(PA);  
    ...  
}
```

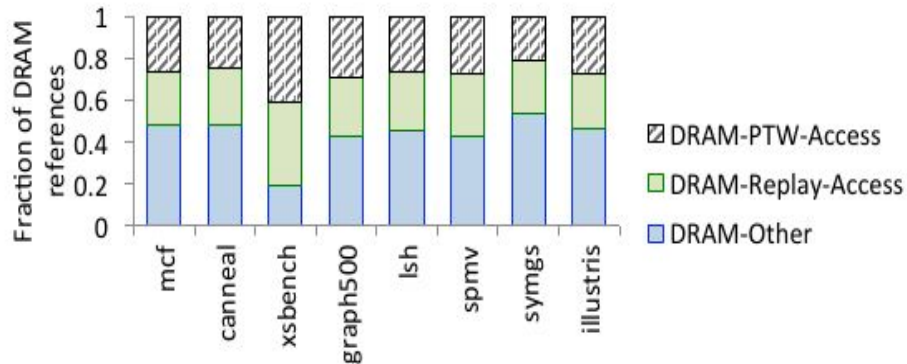
Memory access (simplified serial view)

- Are there any differences in loading the data into L1 in two cases?
 - TLB Hit? (i) No access to page table entries required (ii) Accessed data is as recent as TLB entry
 - TLB Miss? (i) DRAM accesses for PTW (pte entries) (ii) Most likely the accessed data is not in the cache

```
Addr Translate(V){
    PageAddress P = V >> 12;
    TLBEntry entry = lookup(P);
    if (entry.valid) return entry.pte;
    entry = PageTableWalk(V);
    MakeEntry(entry);
    return entry.pte;
}

Load(V){
    Addr PA = Translate( V);
    Load_L1(PA);
    ...
}
```

Translation Triggered Prefetching ¹



- Replay accesses: DRAM accesses after a TLB miss followed by translation using PTW
- What is the expected replay access latency?

Figure 4. Fraction of total DRAM references devoted to page table walk accesses to DRAM (DRAM-PTW-access), replay accesses to DRAM (DRAM-Replay-Access), and other non-page table DRAM accesses (DRAM-Other).

Translation Triggered Prefetching ¹

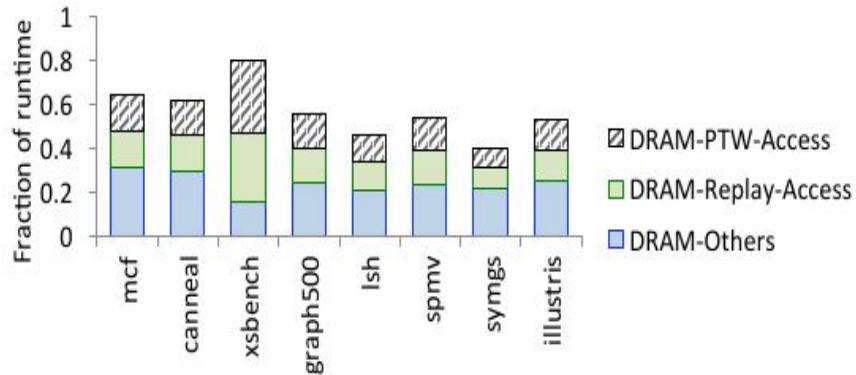
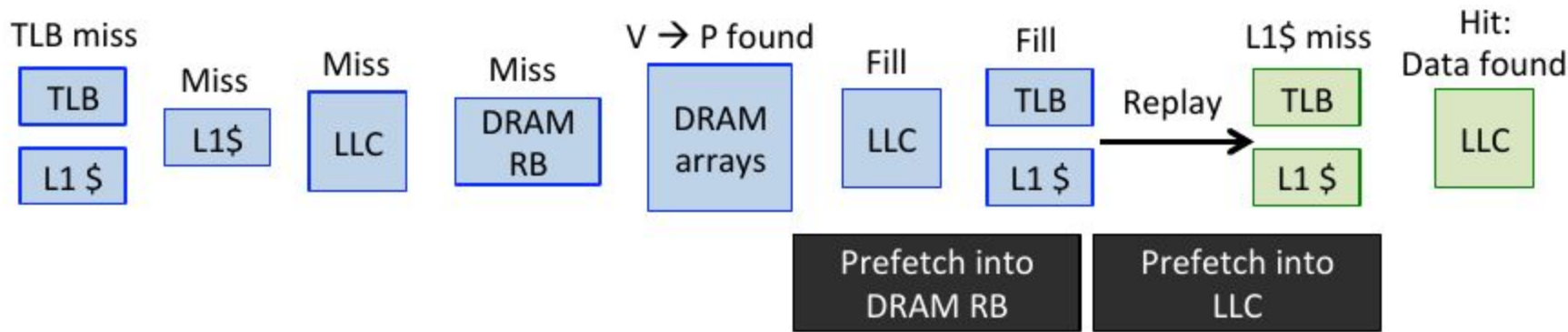


Figure 1. Fraction of total application runtime for page table accesses to DRAM (DRAM-PTW-Access), replayed accesses to DRAM (DRAM-Replay-Access), and other non-page table DRAM accesses (DRAM-Other).

- Replay accesses: DRAM accesses after a TLB miss followed by translation using PTW
- What is the expected replay access latency? If the translation is a miss, mostly the data access is a miss at all cache levels
- How to improve replay access latency?

Translation Triggered Prefetching ¹



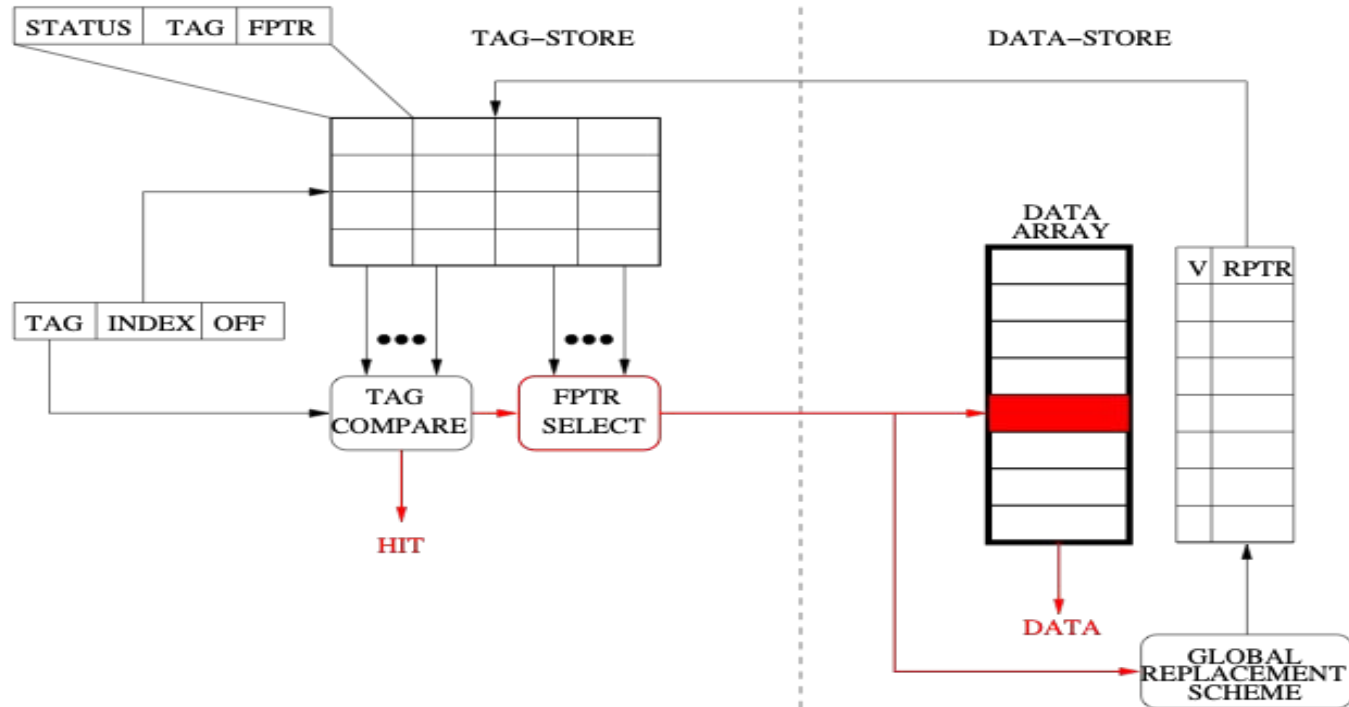
- Main idea: Prefetch the data block into LLC as soon as the target physical address is known at the DRAM controller

Reducing Miss Penalty: V-Way cache²

- Ideal cache: FA with Belady replacement policy
 - Remove conflict misses, benefits from global eviction decisions
- Challenges?
 - Increased # of tag comparisons → increased access latency, power
- Idea: Separate the data and tag stores with additional cross-reference
 - Tag store is SA and data store is accessed by 'PTR' like mechanism, global eviction from data store

Reducing Miss Penalty: V-Way cache²

- Idea: Separate the data and tag stores with additional cross-reference



Cache Replacement Policies

D1			D2			D3			D4		
Tag	V D	Ev	Tag	V D	Ev	Tag	V D	Ev	Tag	V D	Tag

Eviction
Meta-data

- Problem of selecting a victim block
 - Set-associative cache: Victim selection in the set
 - Fully-associative cache: Global victim selection
- Policies
 - LRU, LFU, NMRU, BIP (DIP), RRIP (D-RRIP), Pseudo-Random

LRU Replacement: How to implement?

- Implementation of a classical LRU requires insertion in MRU and eviction from LRU position
- Not easy to implement (especially in hardware)
 - What if we use a sorted list? Challenge?
 - What if we use timestamps? Challenge?

LRU Replacement: How to implement?

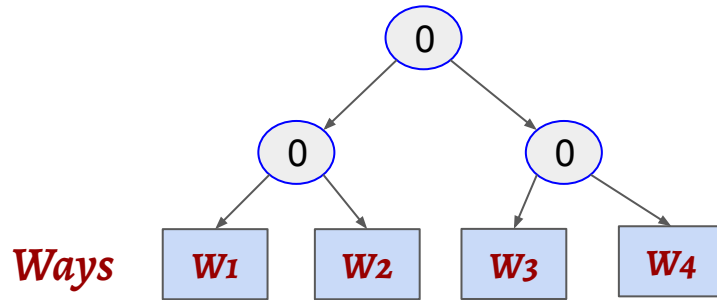
- Implementation of a classical LRU requires insertion in MRU and eviction from LRU position
- Not easy to implement (especially in hardware)
 - What if we use a sorted list? Challenge? Maintaining sorted order on every access, would require $\log_2(\text{Assoc})$ bits **per block**
 - What if we use timestamps? Challenge? Selection of victim require linear scan, would require some bits to maintain LSB of timestamp

LRU Replacement: Approximate

- Implementation of a classical LRU requires insertion in MRU and eviction from LRU position
- Not easy to implement (especially in hardware)
 - What if we use a sorted list? Challenge? Maintaining sorted order on every access, would require $\log_2(\text{Assoc})$ bits **per block**
 - What if we use timestamps? Challenge? Selection of victim require linear scan, would require some bits to maintain LSB of timestamp
- Approximate LRU using not most recently used (NMRU)
 - Never evict the most recently used
 - Choose any other block using a pseudo random generator
 - Approximates LRU for low associativity (equivalent to LRU for 2-way SA cache)

Pseudo LRU using binary trees

- Approximate LRU using trees
 - Encode age in the path by setting bits in the access path based on next level



- On access, set path nodes '0' if traversing left or '1' if traversing right
- While eviction, follow in the reverse order (i.e., right if '0' and '1' if left)
 - Example: If tree level-order encoding is "001" replace W3

Homework: Workout an examples to observe when P-LRU deviate from True LRU

Homework: What is the storage overhead of PLRU?

When true LRU is not good?

- Let us analyze some access patterns for a cache size (associativity) M

$$(A_1, A_2, A_3 \dots A_k, A_k, A_{k-1} \dots A_1)^N$$

LRU friendly

$$(A_1, A_2, A_3 \dots A_k)^N$$

LRU is good if $K < M$, else results in thrashing

$$(A_1, A_2, A_3 \dots A_k \dots \dots A_\infty)^N$$

Streaming access pattern (LRU not good)

- There can be combinations of access patterns in reality
 - Objective: Scan resistance and get some hits for thrashing

Techniques (Bi-modal and RRIP)

- Formulate a different question: Where to insert?
- Idea 1: Insert into LRU position not MRU
- Idea 2: Insert into somewhere in the middle
- Both ideas should encompass the benefits of LRU
 - Probabilistically choose between LRU or MRU insertion (Bi-modal)
 - Use set dueling to decide which is suitable for any workload

Software techniques

- Allocation
 - Alignment of structures/objects, placement of members, inlining vs. redirection
- Program logic restructuring
 - Loop interchange
 - Blocking
- Software prefetching
 - Use software prefetch instructions
 - X86 prefetch instruction
 - GCC: `__builtin_prefetch`