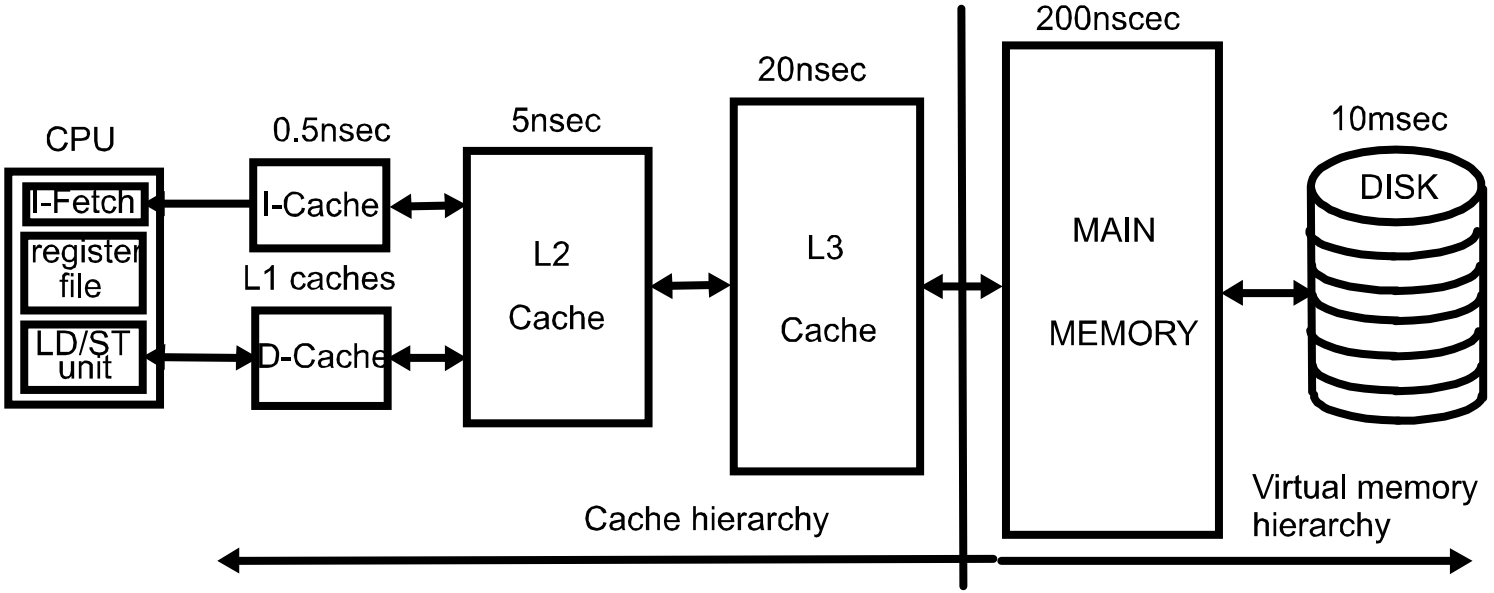
CHAPTER 4

MEMORY HIERARCHIES

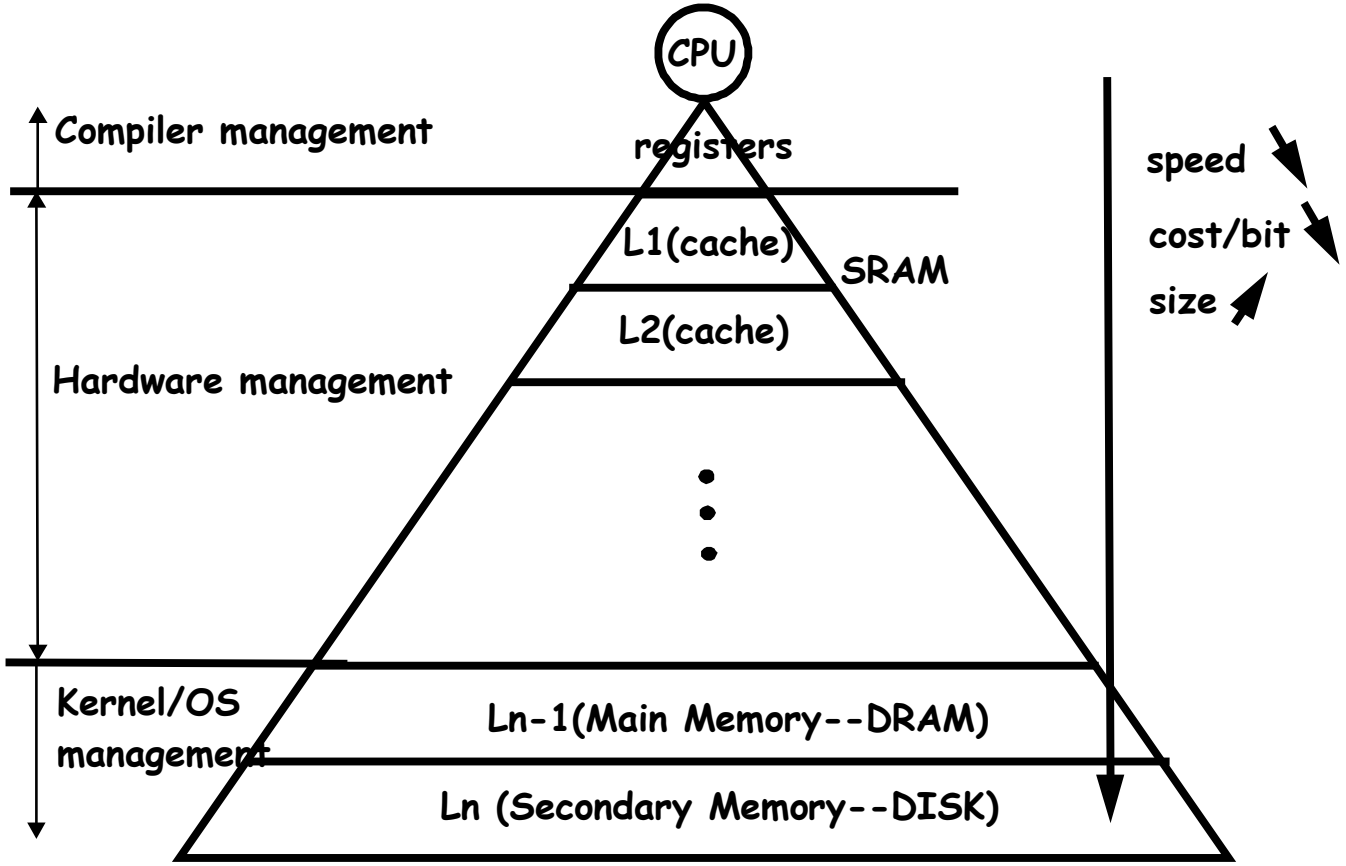
- MEMORY HIERARCHIES
- CACHE DESIGN
- TECHNIQUES TO IMPROVE CACHE PERFORMANCE
- VIRTUAL MEMORY SUPPORT

TYPICAL MEMORY HIERARCHY



- **PRINCIPLE OF LOCALITY:**
- **A PROGRAM ACCESSES A RELATIVELY SMALL PORTION OF THE ADDRESS SPACE AT A TIME**
- **TWO DIFFERENT TYPES OF LOCALITY:**
  - TEMPORAL LOCALITY: IF AN ITEM IS REFERENCED, IT WILL TEND TO BE REFERENCED AGAIN SOON
  - SPATIAL LOCALITY: IF AN ITEM IS REFERENCED, ITEMS WHOSE ADDRESSES ARE CLOSE TEND TO BE REFERENCED SOON
  - SPATIAL LOCALITY TURNS INTO TEMPORAL LOCALITY IN BLOCKS/PAGES

TYPICAL MEMORY HIERARCHY: THE PYRAMID



GOALS: HIGH SPEED, LOW COST, HIGH CAPACITY
INCLUSION
COHERENCE

CACHE PERFORMANCE

- **AVERAGE MEMORY ACCESS TIME (AMAT)**
  AMAT= hit time + miss rate x miss penalty

- **MISS RATE: FRACTION OF ACCESSES NOT SATISFIED AT THE HIGHEST LEVEL**
  - NUMBER OF MISSES IN L1 DIVIDED BY THE NUMBER OF PROCESSOR REFERENCES
  - ALSO HIT RATE = 1 - MISS RATE

- **MISSES PER INSTRUCTIONS (MPI)**
  - NUMBER OF MISSES IN L1 DIVIDED BY NUMBER OF INSTRUCTIONS
  - EASIER TO USE THAN MISS RATE: $CPI = CPI_0 + MPI * MISS\ PENALTY$

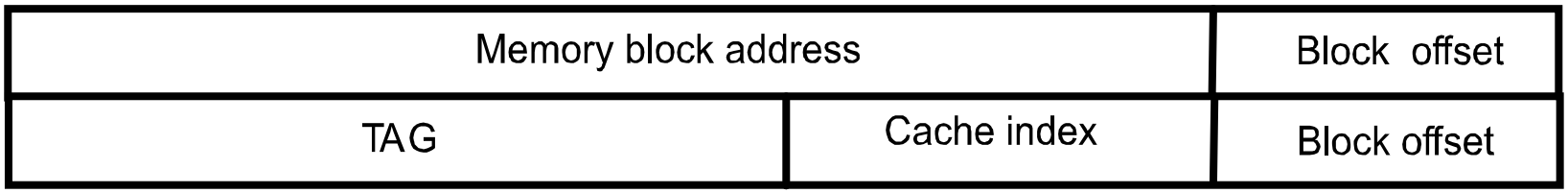
- **MISS PENALTY: AVERAGE DELAY PER MISS CAUSED IN THE PROCESSOR**
  - IF PROCESSOR BLOCKS ON MISSES, THEN THIS IS SIMPLY THE NUMBER OF CLOCK CYCLES TO BRING A BLOCK FROM MEMORY OR MISS LATENCY
  - IN A OoO PROCESSOR, THE PENALTY OF A MISS CANNOT BE MEASURED DIRECTLY
  - DIFFERENT FROM MISS LATENCY

- **MISS RATE AND PENALTY CAN BE DEFINED AT EVERY CACHE LEVELS**
  - USUALLY NORMALIZED TO THE NUMBER OF PROCESSOR REFERENCES
  - OR TO THE NUMBER OF ACCESSES FROM THE LOWER LEVEL

CACHE MAPPING

- MEMORY BLOCKS ARE MAPPED TO CACHE LINES
- MAPPING CAN BE DIRECT, SET-ASSOCIATIVE OR FULLY ASSOCIATIVE

Physical Address



- DIRECT-MAPPED: EACH MEMORY BLOCK CAN BE MAPPED TO ONLY ONE CACHE LINE: BLOCK ADDRESS MODULO THE NUMBER OF LINES IN CACHE
- SET-ASSOCIATIVE: EACH MEMORY BLOCK CAN BE MAPPED TO A SET OF LINES IN CACHE; SET NUMBER IS BLOCK ADDRESS MODULO THE NUMBER OF CACHE SETS
- FULLY ASSOCIATIVE: EACH MEMORY BLOCK CAN BE IN ANY CACHE LINE

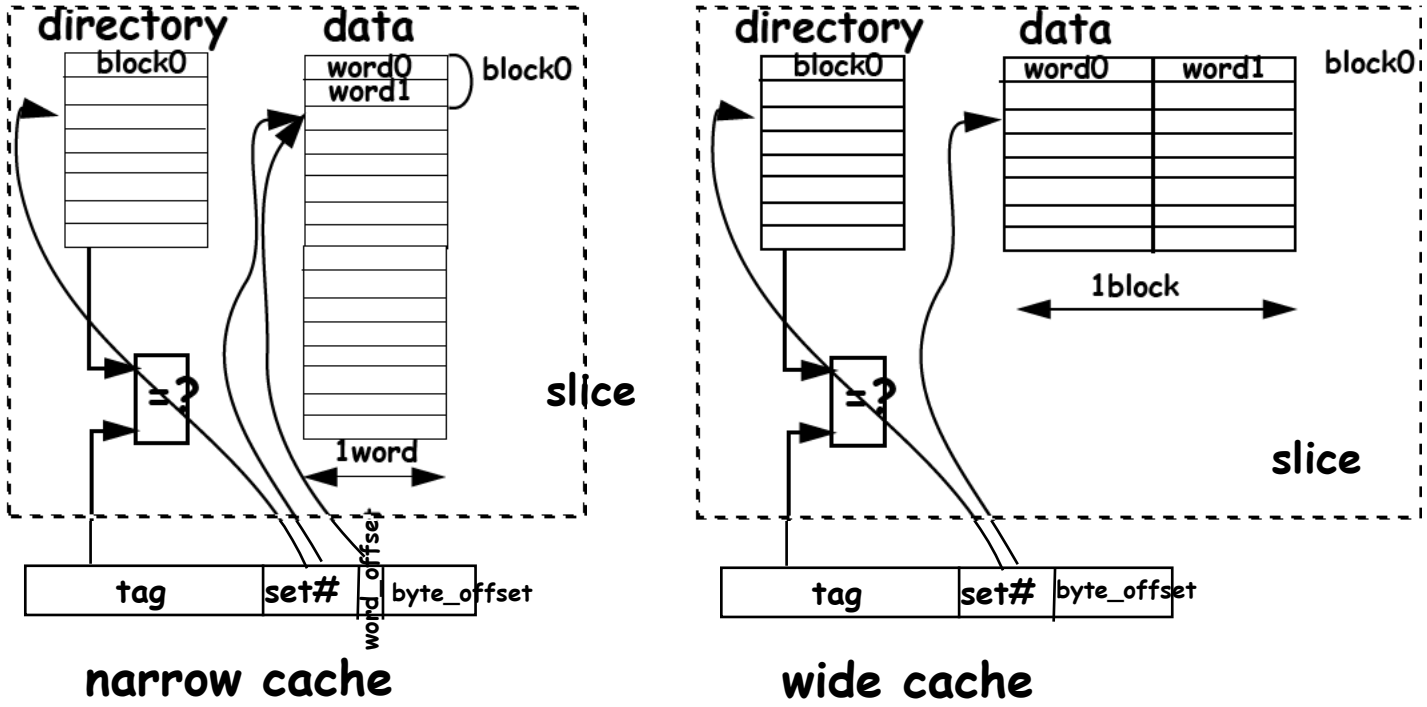
- CACHE IS MADE OF DIRECTORY+ DATA MEMORY, ONE ENTRY PER CACHE LINE
  - DIRECTORY: STATUS (STATE) BITS: VALID, DIRTY, REFERENCE, CACHE COHERENCE

- CACHE ACCESS HAS TWO PHASES
- USE INDEX BITS TO FETCH THE TAGS AND DATA FROM THE SET (CACHE INDEX)
- CHECK TAGS TO DETECT HIT/MISS

CACHE ACCESS

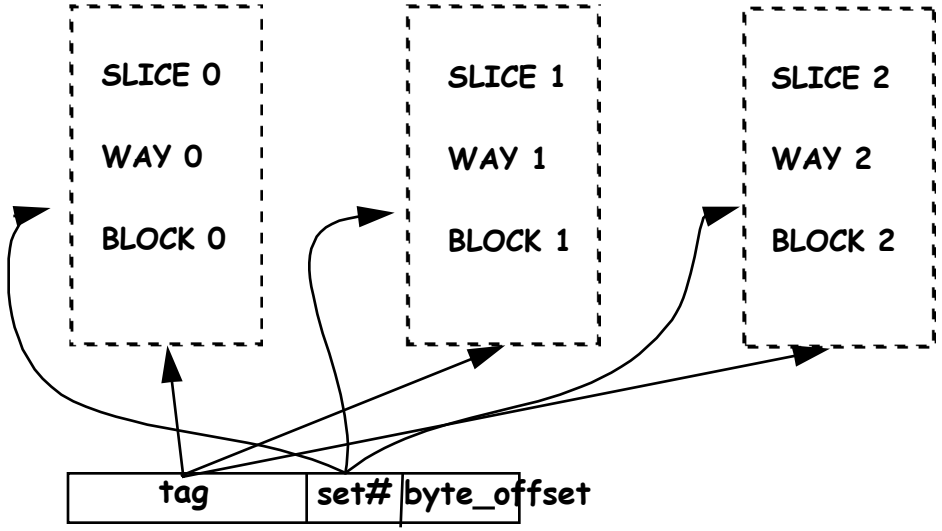
- TWO PHASES: INDEX + TAG CHECK

- DIRECT-MAPPED CACHES: CACHE SLICE
- EXAMPLE OF A DIRECT-MAPPED CAHE WITH TWO WORDS PER LINE.



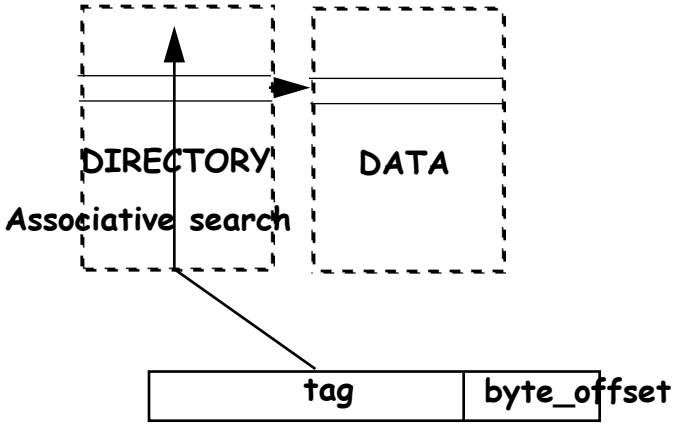
CACHE ACCESS

- SET-ASSOCIATIVE CACHE



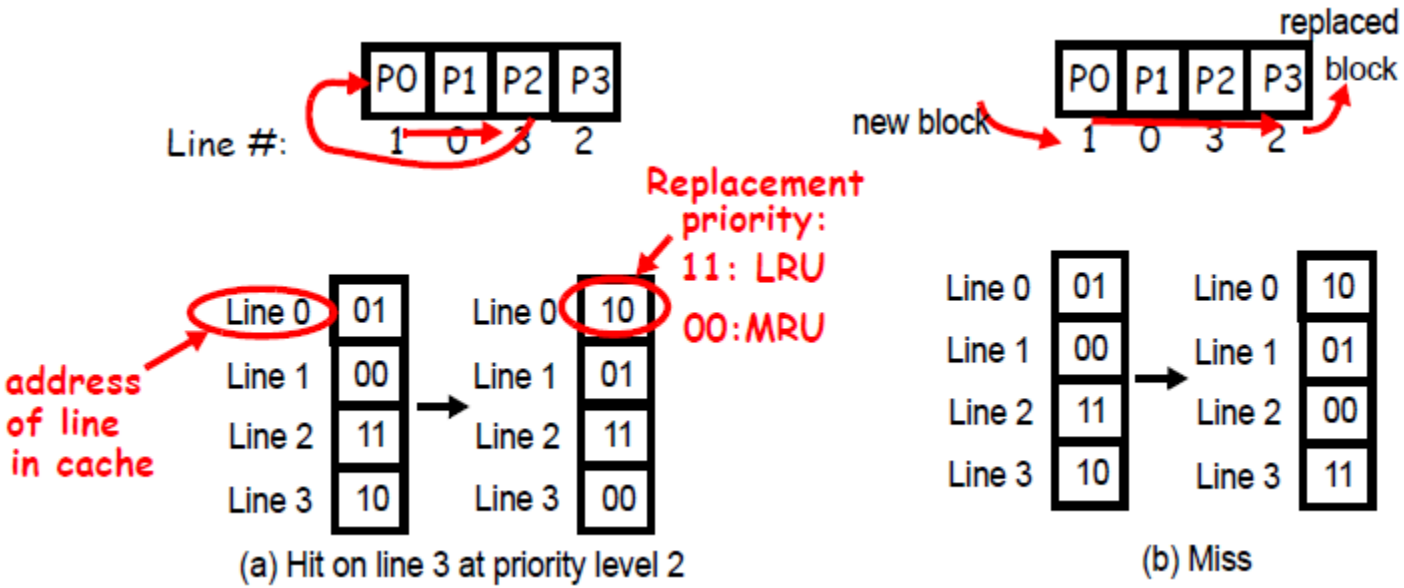
3-WAY SET-ASSOCIATIVE CACHE

- FULLY ASSOCIATIVE



REPLACEMENT POLICIES

- **RANDOM, LRU, FIFO, PSEUDO-LRU**
  - MAINTAINS REPLACEMENT BITS
- **EXAMPLE: LEAST-RECENTLY USED (LRU)**



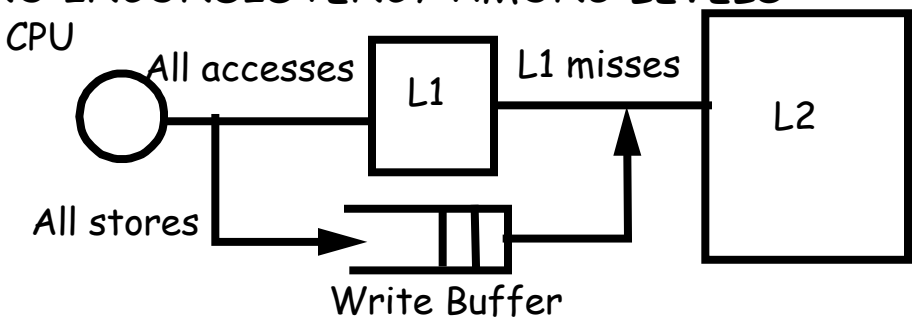
DIRECT-MAPPED: NO NEED

SET-ASSOCIATIVE: PER-SET REPLACEMENT

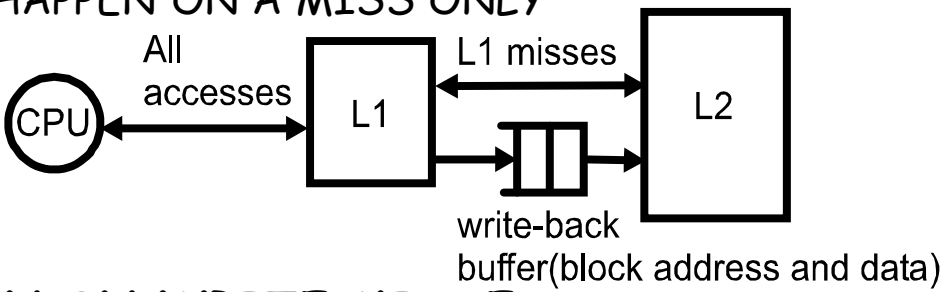
FULLY ASSOCIATIVE: CACHE REPLACEMENT

WRITE POLICIES

- **WRITE THROUGH: WRITE TO NEXT LEVEL ON ALL WRITES**
  - COMBINED WITH WRITE BUFFER TO AVOID CPU STALLS
  - SIMPLE, NO INCONSISTENCY AMONG LEVELS



- **WRITE-BACK: WRITE TO NEXT LEVEL ON REPLACEMENT**
  - WITH THE HELP OF A DIRTY BIT AND A WRITE-BACK BUFFER
  - WRITES HAPPEN ON A MISS ONLY



- **ALLOCATION ON WRITE MISSES**
  - ALWAYS ALLOCATE IN WRITE-BACK; DESIGN CHOICE IN WRITE-THROUGH

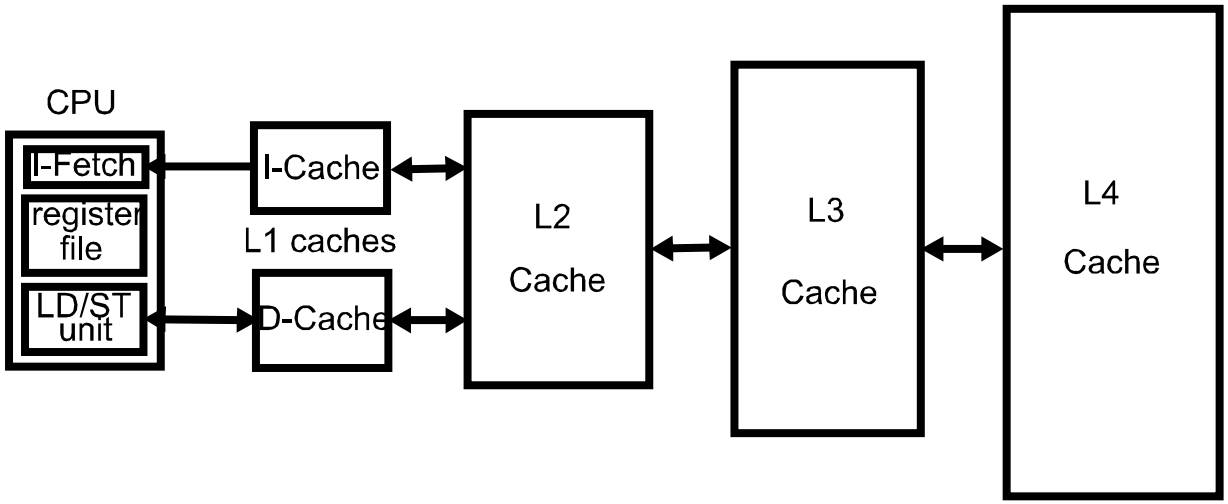
CLASSIFICATION OF CACHE MISSES

- **THE 3 C's**
  - COMPULSORY (COLD) MISSES: ON THE 1ST REFERENCE TO A BLOCK
  - CAPACITY MISSES: SPACE IS NOT SUFFICIENT TO HOST DATA OR CODE
  - CONFLICT MISSES: HAPPEN WHEN TWO MEMORY BLOCKS MAP ON THE SAME CACHE BLOCK IN DIRECT-MAPPED OR SET-ASSOCIATIVE CACHES
    - LATER ON: COHERENCE MISSES ➔ 4C's CLASSIFICATION
- **HOW TO FIND OUT?**
  - COLD MISSES: SIMULATE INFINITE CACHE SIZE
  - CAPACITY MISSES: SIMULATE FULLY ASSOCIATIVE CACHE THEN DEDUCT COLD MISSES
  - CONFLICT MISSES: SIMULATE CACHE THEN DEDUCT COLD AND CAPACITY MISSES

CLASSIFICATION IS USEFUL TO UNDERSTAND HOW TO ELIMINATE MISSES

PROBLEM: WHICH REPLACEMENT POLICY SHOULD WE USE IN THE FULLY ASSOCIATIVE CACHE?

MULTI-LEVEL CACHE HIERARCHIES



- 1ST-LEVEL AND 2ND LEVEL ARE ON-CHIP;3RD AND 4TH LEVELS ARE MOSTLY OFF-CHIP
- USUALLY, CACHE INCLUSION IS MAINTAINED
  - WHEN A BLOCK MISSES IN L1 THEN IT MUST BE BROUGHT INTO ALL Li.
  - WHEN A BLOCK IS REPLACED IN Li, THEN IT MUST BE REMOVED FROM ALL Lj, j<i
- ALSO: EXCLUSION
  - IF A BLOCK IS IN Li THEN IT IS NOT IN ANY OTHER CACHE LEVEL
  - IF A BLOCK MISSES IN L1 THEN ALL COPIES ARE REMOVED FROM ALL Li's, i>1
  - IF A BLOCK IS REPLACED IN Li THEN IT IS ALLOCATED IN Li+1
- OR NO POLICY

WE WILL ASSUME INCLUSION

EFFECT OF CACHE PARAMETERS

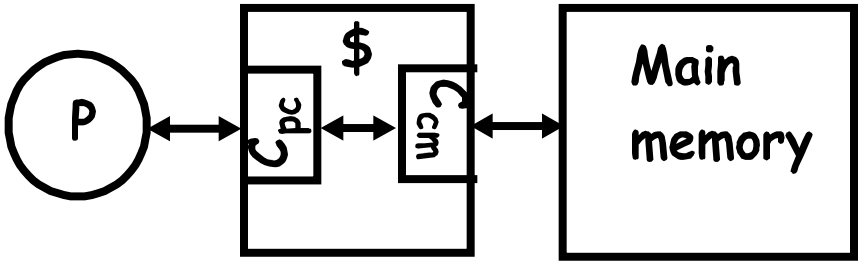
- **LARGER CACHES**
  - SLOWER,
  - MORE COMPLEX,
  - LESS CAPACITY MISSES
- **LARGER BLOCK SIZE**
  - EXPLOIT SPATIAL LOCALITY
  - TOO BIG A BLOCK INCREASES CAPACITY MISSES
  - BIG BLOCKS ALSO INCREASE MISS PENALTY
- **HIGHER ASSOCIATIVITY**
  - ADDRESSES CONFLICT MISSES
  - 8-16 WAY SA IS AS GOOD AS FULLY ASSOCIATIVE
  - A 2-WAY SA CACHE OF SIZE N HAS A SIMILAR MISS RATE AS A DIRECT MAPPED CACHE OF SIZE 2N
  - HIGHER HIT TIME

LOCKUP-FREE (NON-BLOCKING CACHES)

- **CACHE IS A 2-PORTED DEVICE: MEMORY & PROCESSOR**

C_{cm}: cache to memory interface

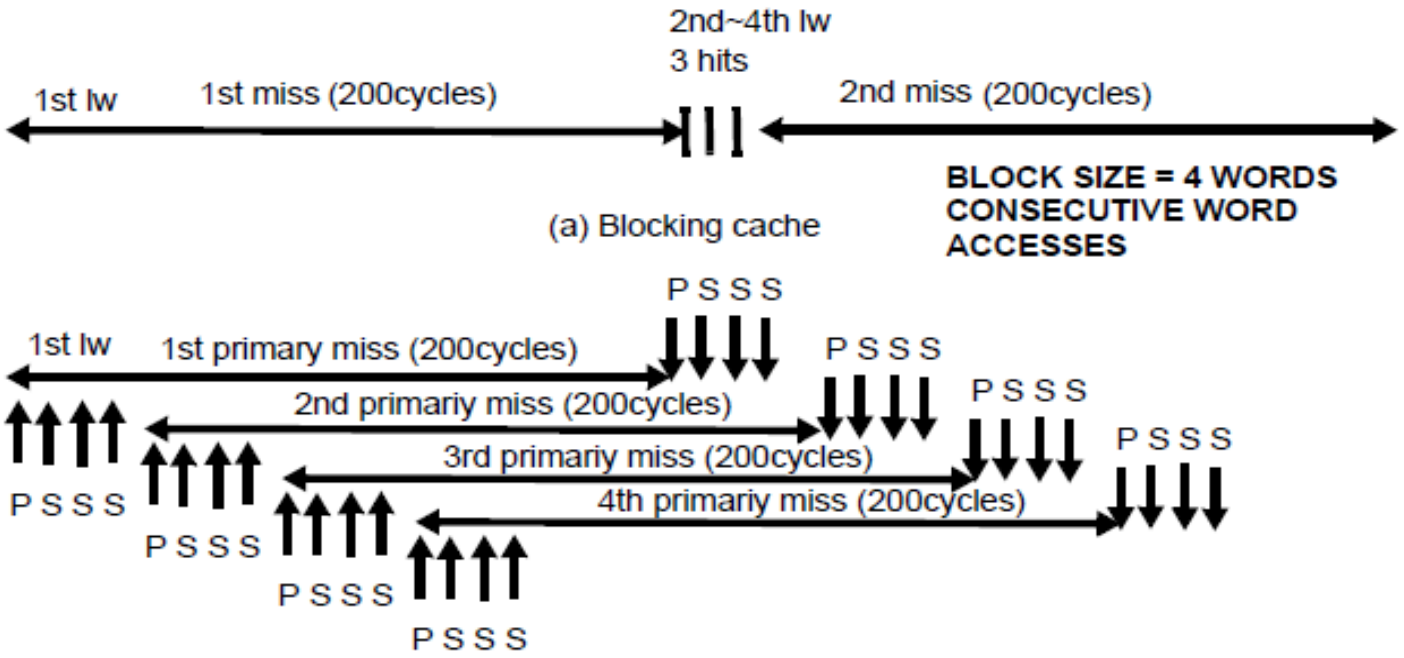
C_{pc}: processor to cache interface



- IF A LOCKUP-FREE CACHE MISSES, IT DOES NOT BLOCK
- RATHER, IT HANDLES THE MISS AND KEEPS ACCEPTING ACCESSES FROM THE PROCESSOR
  - ALLOWS FOR THE CONCURRENT PROCESSING OF MULTIPLE MISSES AND HITS
- CACHE HAS TO BOOKKEEP ALL PENDING MISSES
  - MSHRs (Miss Status Handling Registers) CONTAIN THE ADDRESS OF PENDING MISS, THE DESTINATION BLOCK IN CACHE, AND THE DESTINATION REGISTER
  - NUMBER OF MSHRs LIMITS THE NUMBER OF PENDING MISSES
- DATA DEPENDENCIES EVENTUALLY BLOCK THE PROCESSOR
- NON-BLOCKING CACHES ARE REQUIRED IN DYNAMICALLY SCHEDULED PROCESSOR AND TO SUPPORT PREFETCHING

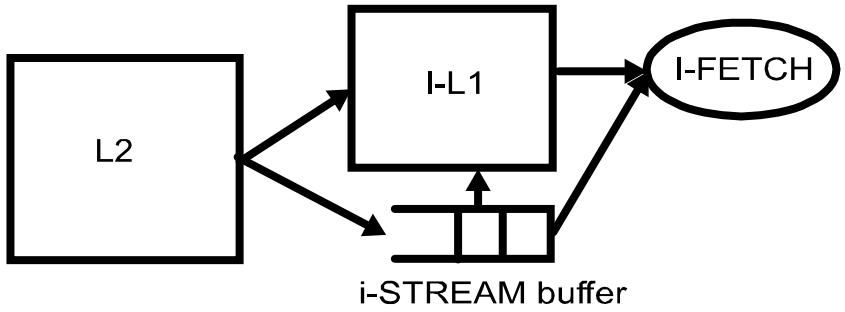
LOCKUP-FREE CACHES:PRIMARY/SECONDARY MISSES

- **PRIMARY: THE FIRST MISS TO A BLOCK**
- **SECONDARY: FOLLOWING ACCESSES TO BLOCKS PENDING DUE TO PRIMARY MISS**
  - LOT MORE MISSES (BLOCKING CACHE ONLY HAS PRIMARY MISSES)
  - NEEDS MSHRs FOR BOTH PRIMARY AND SECONDARY MISSES
  - MISSES ARE OVERLAPPED WITH COMPUTATION AND OTHER MISSES

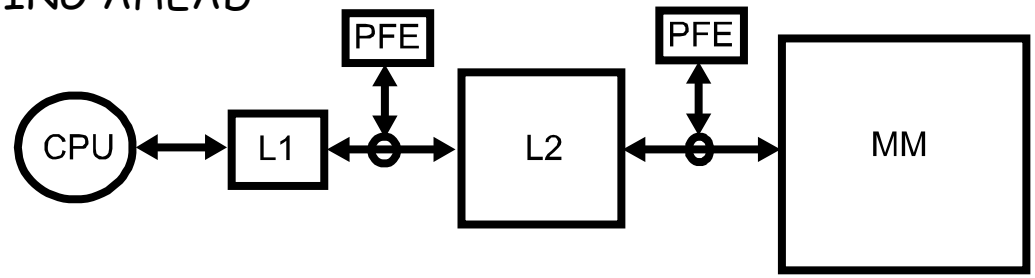


HARDWARE PREFETCHING OF INSTRUCTIONS AND DATA

- **SEQUENTIAL PREFETCHING OF INSTRUCTIONS**



  - ON AN I-FETCH MISS, FETCH TWO BLOCKS INSTEAD OF ONE
  - SECOND BLOCK IS STORED IN AN I-STREAM BUFFER
  - IF I-STREAM-BUFFER HITS, BLOCK IS MOVED TO L1
  - I-STREAM BUFFER BLOCKS ARE OVERLAID IF NOT ACCESSED
  - ALSO APPLICABLE TO DATA, BUT LESS EFFECTIVE
- **HARDWARE PREFETCH ENGINES**
  - DETECT STRIDES IN STREAM OF MISSING ADDRESSES, THEN START FETCHING AHEAD



COMPILER-CONTROLLED (aka SOFTWARE) PREFETCHING

- INSERT PREFETCH INSTRUCTIONS IN CODE

- THESE ARE NON-BINDING, LOAD IN CACHE ONLY
- IN A LOOP, WE MAY INSERT PREFETCH INSTRUCTIONS IN THE BODY OF THE LOOP TO PREFETCH DATA NEEDED IN FUTURE LOOP ITERATIONS.

```
LOOP    L.D F2,0(R1)
        PREF-24(R1)
        ADD.D F4,F2,F0
        S.D F4,0(R1)
        SUBI R1,R1,#8
        BNEZ R1, LOOP
```

  - CAN WORK FOR BOTH LOADS AND STORES
  - REQUIRES A NON-BLOCKING CACHE
  - INSTRUCTION OVERHEAD
  - DATA MUST BE PREFETCHED ON TIME, SO THAT THEY ARE PRESENT IN CACHE AT THE TIME OF ACCESS
  - DATA MAY NOT BE PREFETCHED TOO EARLY SO THAT THEY ARE STILL IN CACHE AT THE TIME OF THE ACCESS

CAN EASILY BE DONE FOR ARRAYS, BUT CAN BE APPLIED TO POINTER ACCESSES

FASTER HIT TIMES

- **PRINCETON vs HARVARD CACHE:**
  - PRINCETON: UNIFIED INSTRUCTIONS/DATA CACHE
  - HARVARD: SPLIT INSTRUCTION AND DATA CACHE
  - PRINCETON MEANS THAT INSTRUCTIONS AND DATA CAN USE THE WHOLE CACHE, AS THEY NEED
  - HARVARD MEANS THAT BOTH CACHES CAN BE OPTIMIZED FOR THEIR ACCESS TYPE
  - IN A PIPELINED MACHINE, FLC IS HARVARD AND SLC IS PRINCETON

- **PIPELINE CACHE ACCESSES**
  - ESPECIALLY USEFUL FOR STOREs
  - PIPELINE TAG CHECK AND DATA STORE
  - SEPARATE READ AND WRITE PORTS TO CACHE OPTIMIZED FOR EACH
  - ALSO USEFUL FOR I-CACHES AND LW IN D-CACHES
  - INCREASES THE PIPELINE LENGTH
  - MUST FIND WAYS OF SPLITTING CACHE ACCESSES INTO STAGES

FASTER HIT TIMES

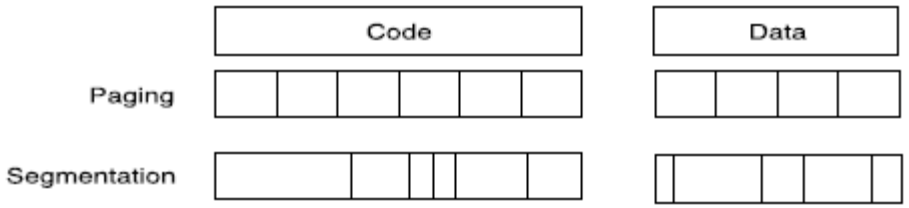
- KEEP THE CACHE SIMPLE AND FAST
  - THIS FAVORS DIRECT-MAPPED CACHES
    - LESS MULTIPLEXING
    - OVERLAP OF TAG CHECK AND USE OF DATA
  - INTERESTINGLY, THE SIZE OF FLC TENDS TO DECREASE AND ASSOCIATIVITY GOES UP AS FLCs TRY TO KEEP UP WITH CPU

Processor	L1 data cache
Alpha 21164	8KB, direct mapped
Alpha 21364	64KB, 2-way
MPC750	32KB, 8-way, PLRU
PA-8500	1MB, 4-way, PLRU
Classic Pentium	16K, 4-way, LRU
Pentium-II	16KB, 4 -way, PLRU
Pentium-III	16K, 4-way, PLRU
Pentium-IV	8KB, 4-way, PLRU
Mips R10K/12K	32KB, 2-way, LRU
UltraSPARC-II*i*	16KB, direct mapped
UltraSPARC-III	64KB, 4-way, Random

- AVOID ADDRESS TRANSLATION OVERHEAD (ToBeCoveredLater)

VIRTUAL MEMORY—WHY?

- **ALLOWS APPLICATIONS TO BE BIGGER THAN MAIN MEMORY SIZE**
  - PREVIOUSLY: MEMORY OVERLAYS
- **HELPS WITH MULTIPLE PROCESS MANAGEMENT**
  - EACH PROCESS GETS ITS OWN CHUNK OF MEMORY
  - PROTECTION OF PROCESSES AGAINST EACH OTHER
  - PROTECTION OF PROCESSES AGAINST THEMSELVES
  - MAPPING OF MULTIPLE PROCESSES TO MEMORY
  - RELOCATION
  - APPLICATION AND CPU RUN IN VIRTUAL SPACE
  - MAPPING OF VIRTUAL TO PHYSICAL SPACE IS INVISIBLE TO THE APPLICATION
- **MANAGEMENT BETWEEN MM AND DISK**
  - MISS IN MM IS A PAGE FAULT OR ADDRESS FAULT
  - BLOCK IS A PAGE OR SEGMENT

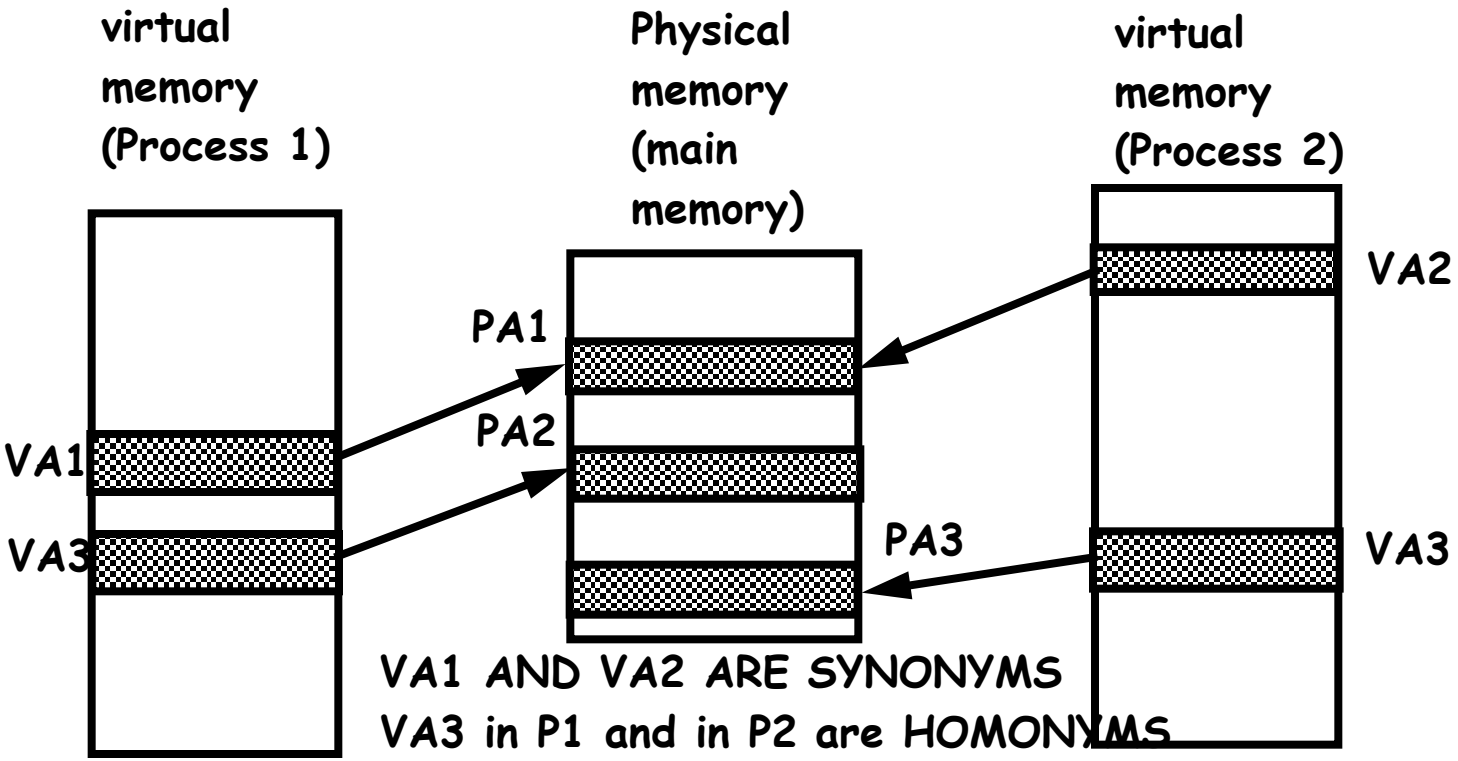


PAGES vs SEGMENTS

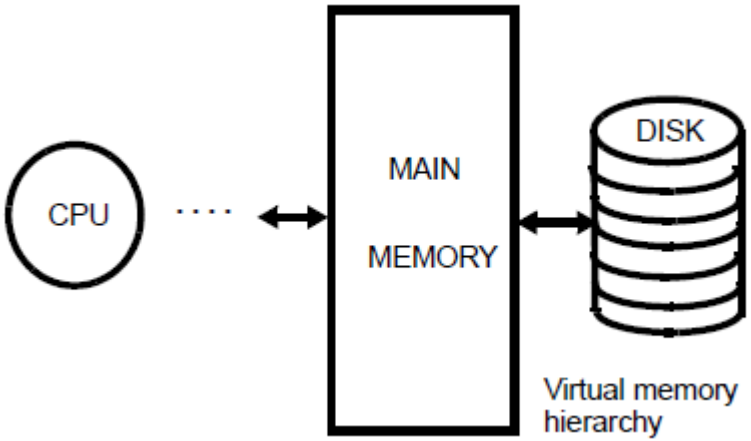
- MOST SYSTEMS TODAY USE PAGING
- SOME SYSTEMS USE PAGED SEGMENTS
- SOME SYSTEMS USE MULTIPLE PAGE SIZES
  - SUPERPAGES (TO BE COVERED LATER)

	PAGE	SEGMENT
Addressing	one	two (segment and offset)
Programmer visible?	Invisible	May be visible
Replacing a block	Trivial	Hard
Memory use efficiency	Internal fragmentation	External fragmentation
Efficient disk traffic	Yes	Not always

VIRTUAL ADDRESS MAPPING



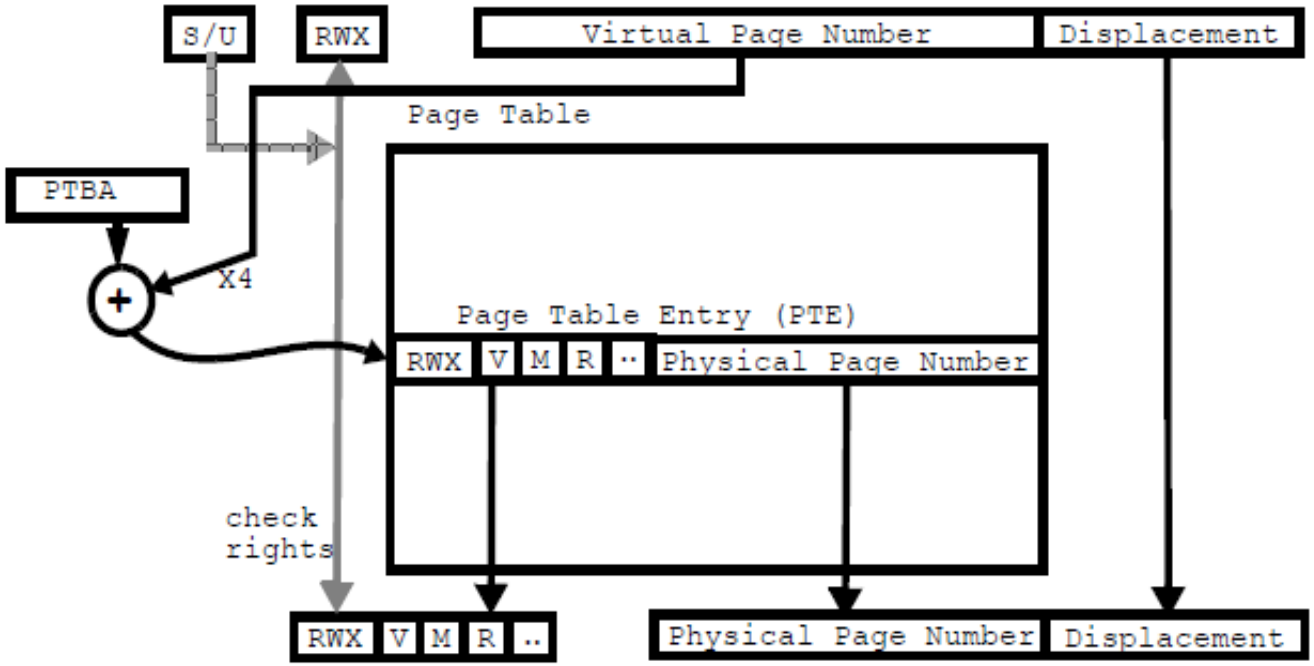
PAGED VIRTUAL MEMORY



- **VIRTUAL ADDRESS SPACE DIVIDED INTO PAGES**
- **PHYSICAL ADDRESS SPACE DIVIDED INTO PAGEFRAMES**
- **PAGE MISSING IN MM = PAGE FAULT**
  - Pages not in MM are on disk: swap-in/swap-out
  - Or have never been allocated
  - New page may be placed anywhere in MM (fully associative map)
- **DYNAMIC ADDRESS TRANSLATION**
  - Effective address is virtual
  - Must be translated to physical for every access
  - Virtual to physical translation through page table in MM

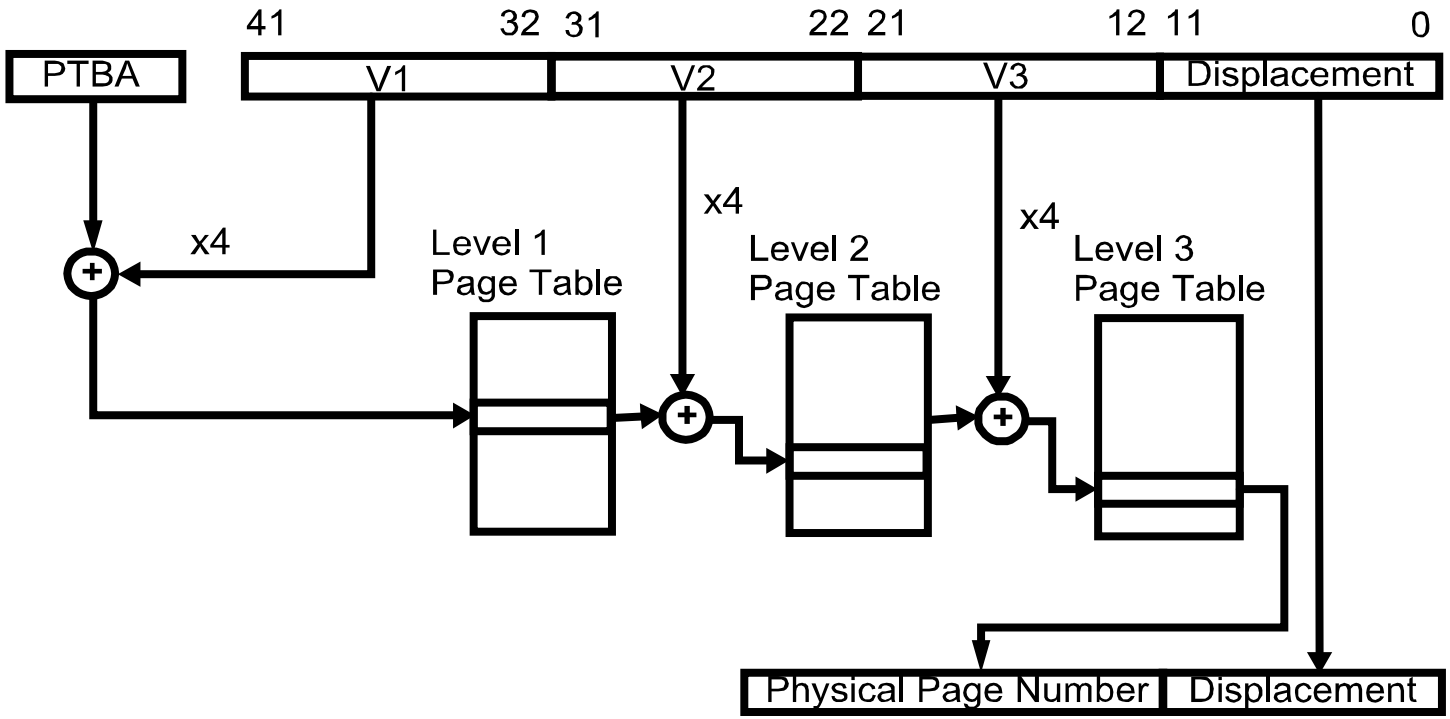
PAGE TABLE

- PAGE TABLE TRANSLATES ADDRESS, ENFORCES PROTECTION



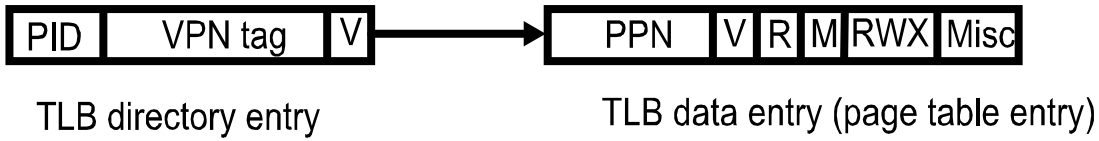
- PAGE REPLACEMENT
  - FIFO—LRU--MFU
  - APPROXIMATE LRU (working set)
    - REFERENCE BIT (R) PER PAGE IS PERIODICALLY RESET BY O/S
    - PAGE CACHE: HARD vs SOFT PAGE FAULTS
- WRITE STRATEGY IS WRITE BACK USING MODIFY (M) BIT
  - M AND R BITS ARE EASILY MAINTAINED BY SOFTWARE USINGTRAPS

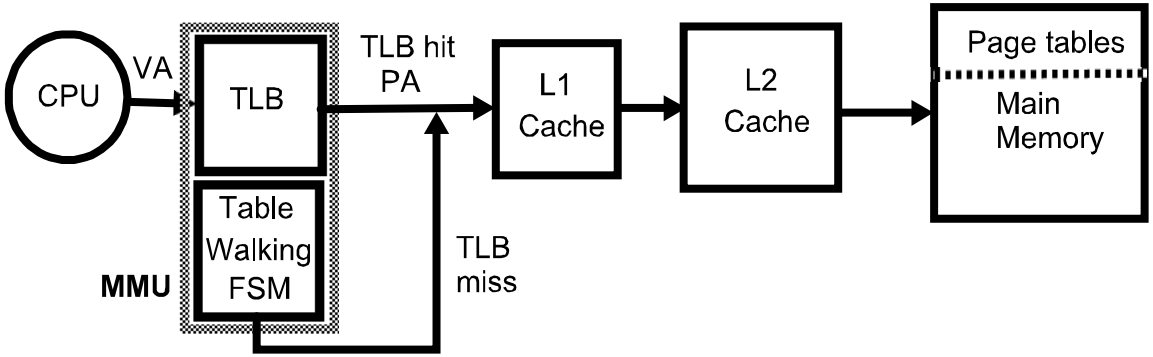
HIERARCHICAL PAGE TABLE



- HIERARCHICAL PAGE TABLE SUPPORTS SUPERPAGES (HOW?)
- MULTIPLE DRAM ACCESSES PER MEMORY TRANSLATION
  - TRANSLATION ACCESSES CAN BE CACHED, BUT STILL REQUIRE MULTIPLE CACHE ACCESSES
  - SOLUTION: SPECIAL CACHE DEDICATED TO TRANSLATIONS

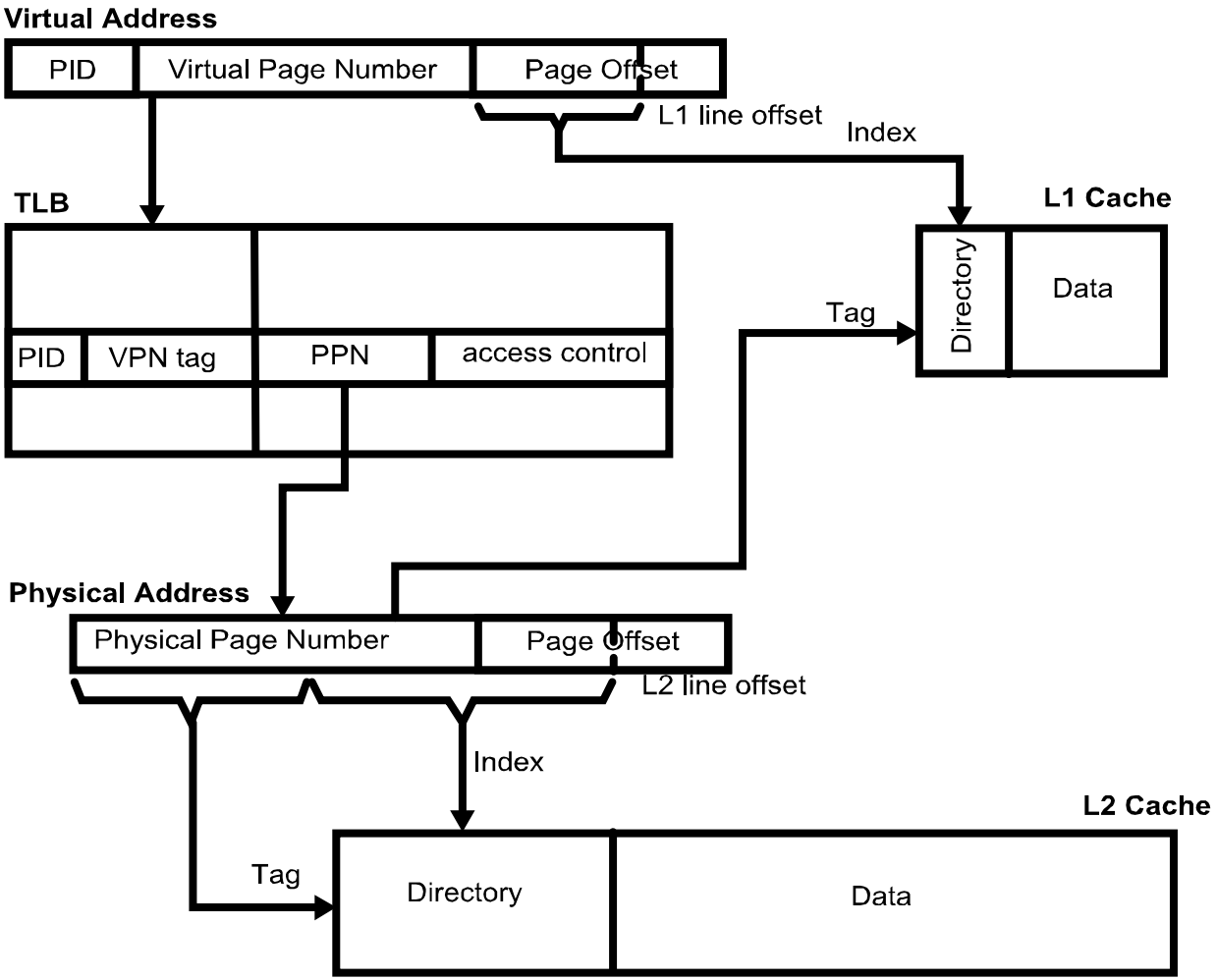
TRANSLATION LOOKASIDE BUFFER





- **PAGE TABLE ENTRY IS CACHED IN TLB**
  - TLB IS ORGANIZED AS A CACHE (DM,SA,or FA) ACCESSED WITH THE VPN
    - PID ADDED TO DEAL WITH HOMONYMS
  - TLBs ARE MUCH SMALLER THAN CACHES BECAUSE OF COVERAGE
  - USUALLY TWO TLB: i-TLB AND d-TLB
  - TLB MISS CAN BE HANDLED IN A HARDWARED MMU OR BY A SOFTWARE TRAP HANDLER
    - "TABLE WALKING"

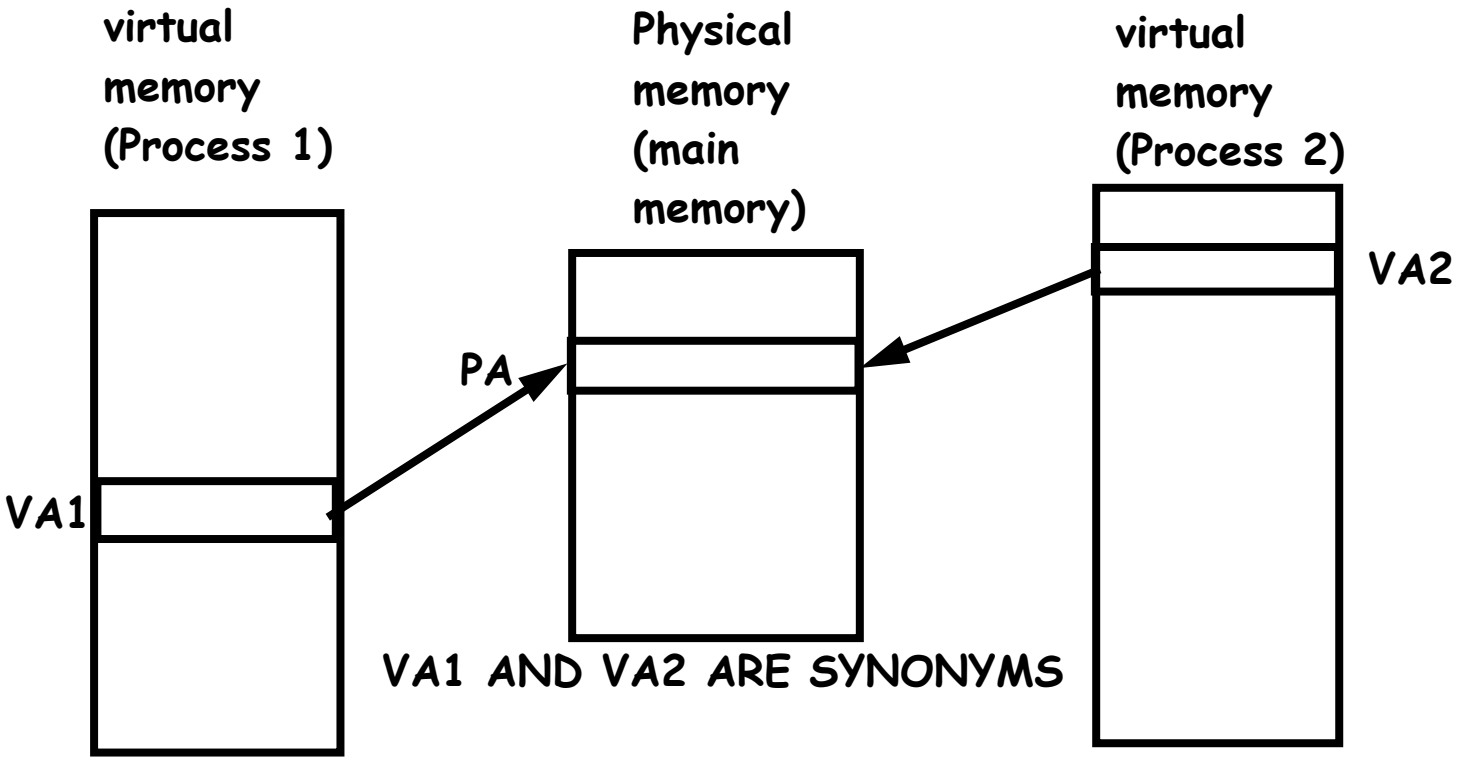
TYPICAL MEMORY HIERARCHY



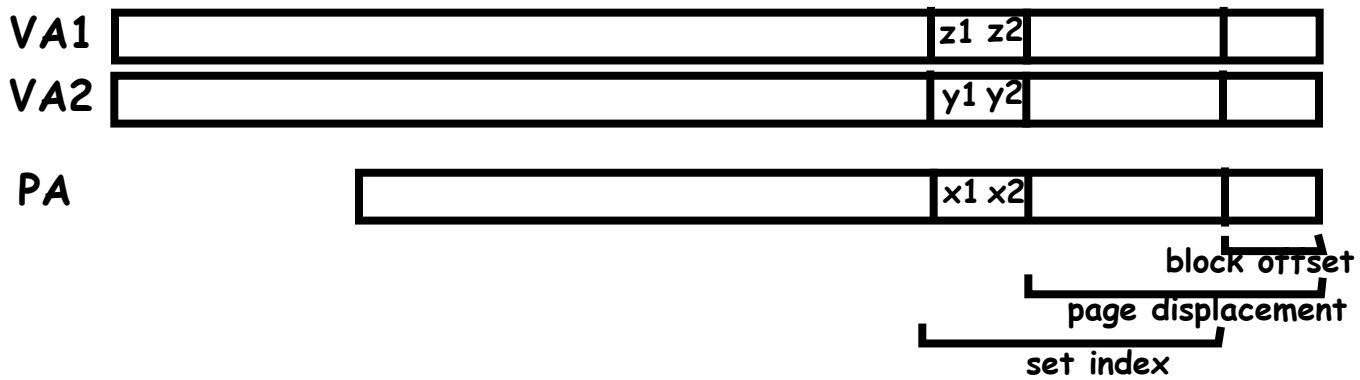
CACHE SIZE LIMITED TO 1 PAGE PER WAY OF ASSOCIATIVITY

INDEXING CACHE WITH VIRTUAL ADDRESS BITS

- WHEN THE L1 CACHE SIZE IS LARGER THAN 1 PAGE PER WAY OF ASSOCIATIVITY, THEN THE BLOCK MIGHT END UP IN TWO DIFFERENT SETS
- THIS IS DUE TO SYNONYMS A.K.A. ALIASES



ALIASING PROBLEM

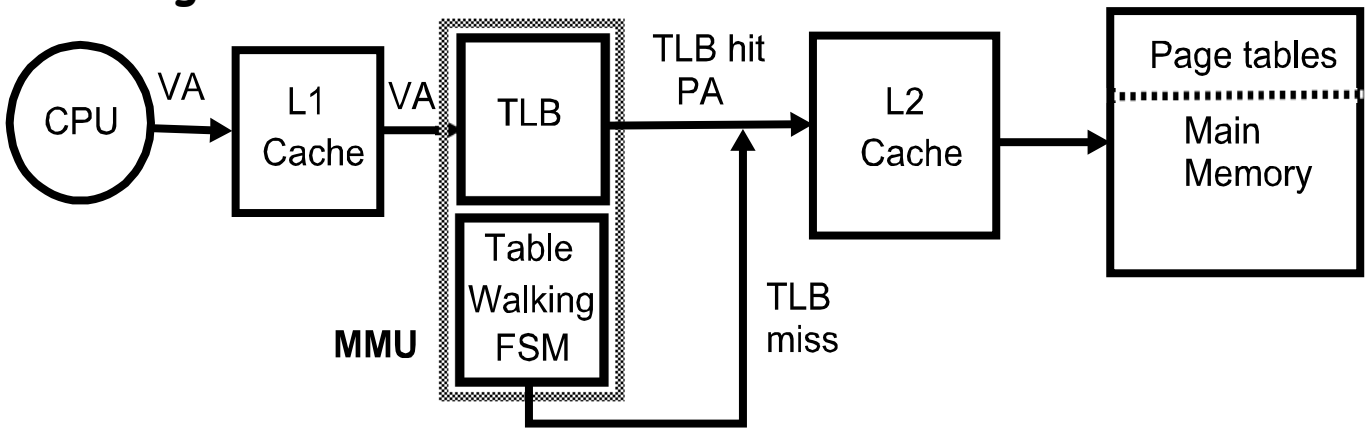


- **If the cache is indexed with the 2 extra bits of VA, then, a block may end up in different sets if accessed with two synonyms, causing consistency problems. How to avoid this??**
  - On a miss, search all the possible sets (in this case 4 sets), move block if needed (SHORT miss), and settle on a (LONG) miss only if all 4 sets miss, OR
  - Since SLC is physically addressed and inclusion is enforced, make sure that SLC entry contains a pointer to the latest block copy in cache, OR
  - Page coloring: make sure that all aliases have the same two extra bits(z1=y1=x1 and z2=y2=x2)
- **No two synonyms can reside in the same page**

Aliasing problem is more acute in caches that are both indexed and tagged with virtual address bits.

VIRTUAL ADDRESS CACHES

- Index and tag are virtual and TLB is accessed on cache misses only



- To search all the sets, we need to check each tag one by one in the TLB (very inefficient unless L1 cache is direct mapped)
- Aliasing can be solved with anti-aliasing hardware, usually in the form of a reverse map in the SLC or some other table
- Protection: Access right bits must be migrated to the L1 cache and managed
- In multiprocessor systems virtual address caches aggravate the coherence problem

The main advantages of VA in L1 caches are 1) fast hit with large cache (no TLB translation in the critical path) and 2) very large TLB with extremely low miss rates