SELECTED TOPICS IN COHERENCE AND CONSISTENCY

Michel Dubois

Ming-Hsieh Department of Electrical Engineering

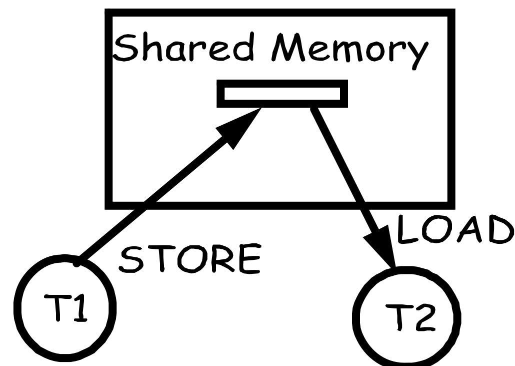
University of Southern California
Los Angeles, CA90089-2562
dubois@usc.edu

INTRODUCTION

- IN CHIP MULTIPROCESSORS (CMPs) CPUs (cores) SHARE SOME MEMORY

- COMMUNICATION IS IMPLICIT THROUGH LOADS AND STORES

SHARED MEMORY IS A COMMUNICATION MECHANISM



NO HYPOTHESIS ON THE RELATIVE SPEED OF PROCESSORS

INTRODUCTION

CACHING

- CACHES ARE NECESSARY TO MAINTAIN THE HIGH EXECUTION RATE OF CORES
- MANY COPIES OF THE SAME ADDRESS (IN CACHES OR IN OTHER BUFFERS) MAY BE PRESENT AT ANY TIME

THESE COPIES MUST GIVE THE ILLUSION OF BEING COHERENT (I.E., EQUIVALENT TO A SINGLE COPY)

- COHERENCE IS TRIVIALLY ENFORCED WHEN THERE IS A SINGLE COPY

- COHERENCE IS ALSO TRIVIALLY ENFORCED WHEN MULTIPLE COPIES OF THE SAME ADDRESS ARE ALWAYS THE SAME

INTRODUCTION

MEMORY CONSISTENCY MODEL(MCM)

"THE ORDER IN WHICH ACCESSES (TO ALL ADDRESSES) BY ONE THREAD ARE OBSERVED BY OTHER THREADS"

MUCH TOUGHER THAN COHERENCE

- CONSIDER THE FOLLOWING CODE:
  ASSUME A AND flag ARE BOTH 0 INITIALLY

```
P1                     P2
...                    ...
A:=1;                  while(flag==0)do nothing;
flag:=1;               print A;
...                    ...
```

- ONE WOULD ASSUME THAT THE VALUE 1 IS PRINTED BY P2
- HOWEVER THIS MAY NOT ALWAYS BE THE CASE

COHERENCE DOES NOT DO THAT

MEMORY CONSISTENCY MODEL

PROCESSOR ARCHITECTURE +

MEMORY ARCHITECTURE

Design each independently

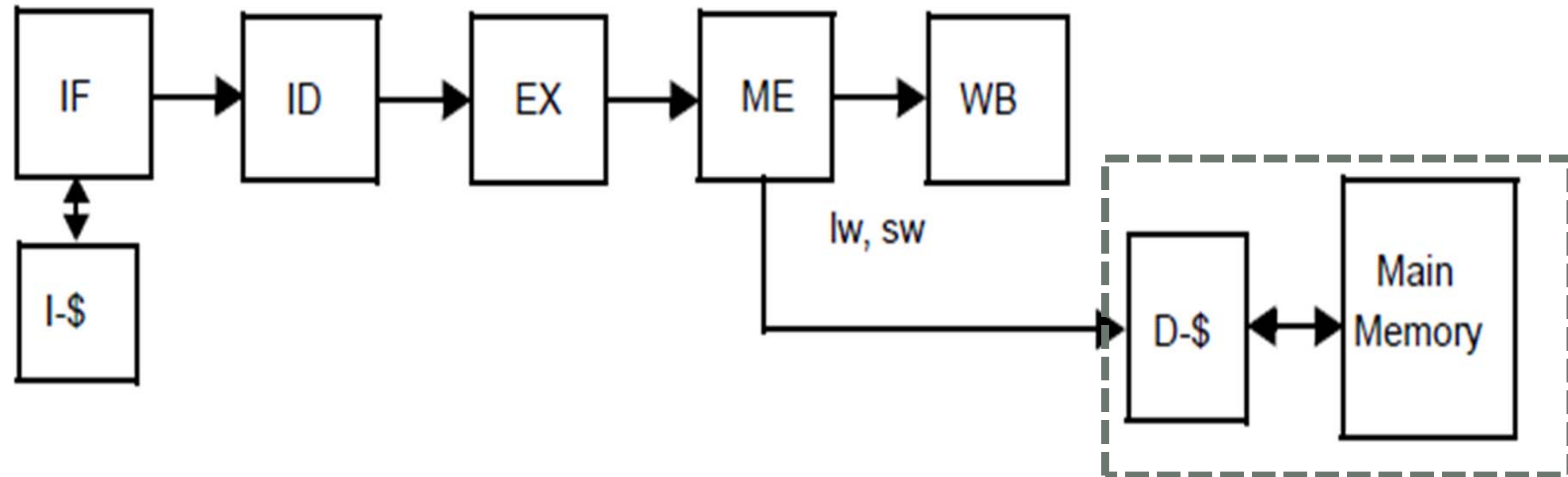
INTRODUCTION

CONTENT

1. PIPELINES
2. SYNCHRONIZATION
3. STORE ATOMICITY
4. SEQUENTIAL CONSISTENCY
5. RELAXED MEMORY CONSISTENCY MODELS
6. SPECULATIVE VIOLATIONS OF MEMORY CONSISTENCY MODELS
7. QUESTION: DO WE REALLY NEED COHERENCE?
8. OTHER TOPICS

1. PIPELINES

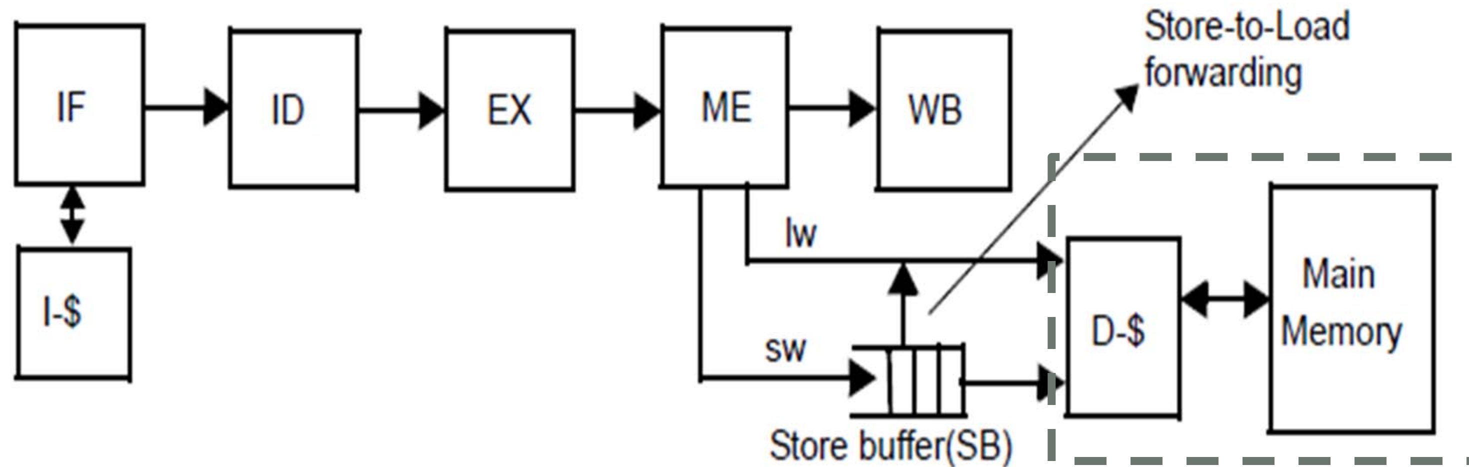
5-STAGE PIPELINE



- MEMORY ACCESSES (LOADS AND STORES) ARE BLOCKING AND EXECUTE ONE BY ONE IN PROCESS ORDER IN THE MEMORY STAGE
- NO MEMORY HAZARDS
- NO PROBLEM FROM THE PROCESSOR SIDE

NO MEMORY HAZARD

5-STAGE PIPELINE WITH STORE BUFFER



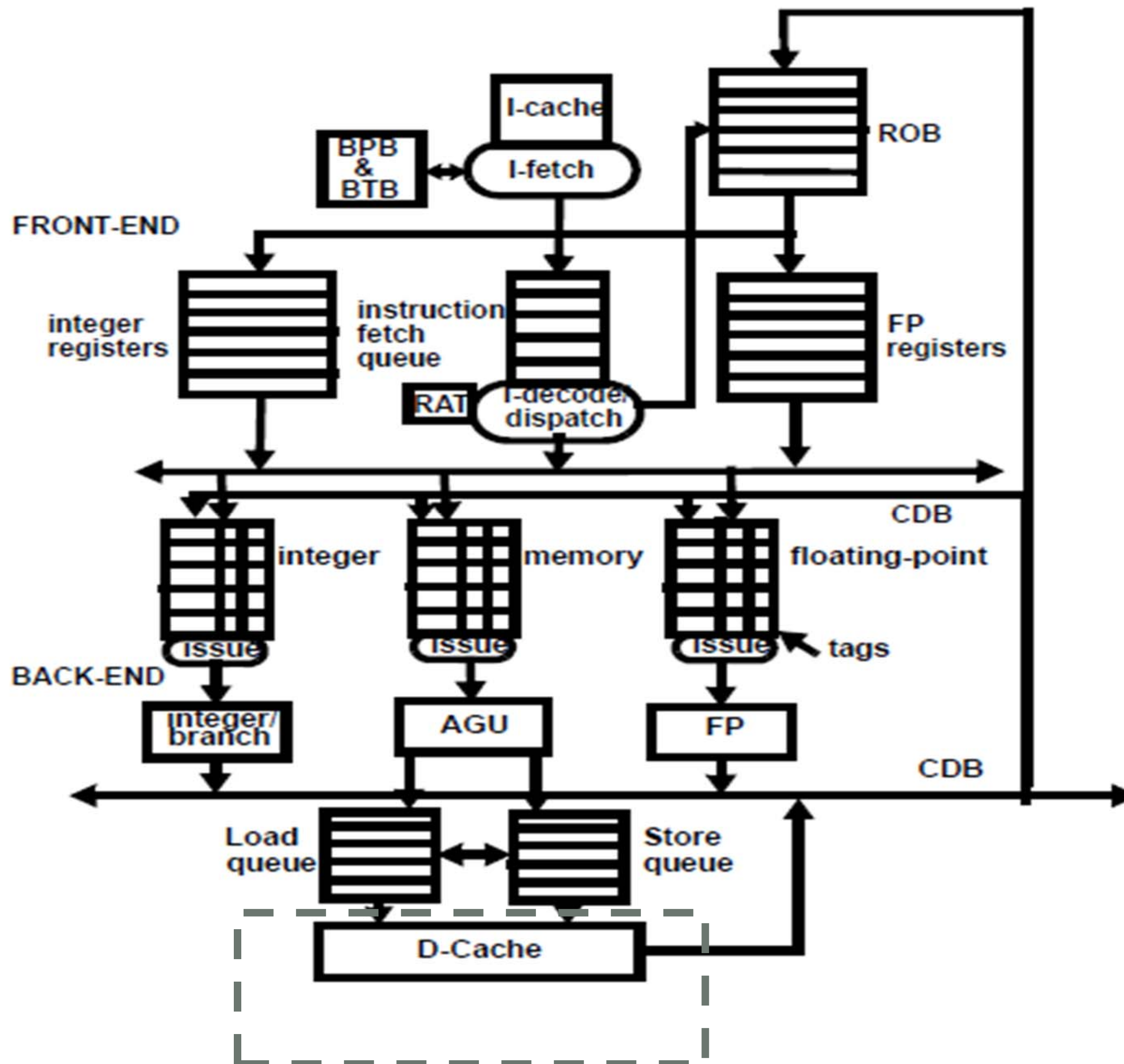
HERE, LOADS ARE BLOCKING BUT STORES ARE NOT

LOADS AND STORES REACH THE CACHE OUT OF PROCESS ORDER

$\Rightarrow$ MEMORY HAZARDS ARE POSSIBLE

VALUES MAY OR MAY NOT BE FORWARDED FROM SB

Out-of-Order EXECUTION



INSTRUCTIONS ARE DISPATCHED IN ORDER

THEY EXECUTE OUT OF ORDER

THEY COMMIT THEIR RESULTS IN ORDER

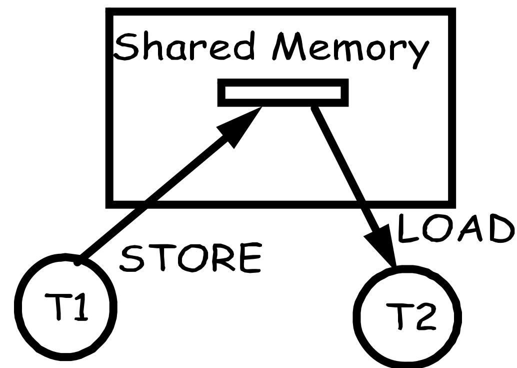
STORES EXECUTE WHEN THEY ARE AT THE TOP OF ROB

LOADS EXECUTE SPECULATIVELY

2. SYNCHRONIZATION

SHARED-MEMORY COMMUNICATION

- IMPLICITELY VIA MEMORY



  - PROCESSORS SHARE SOME MEMORY
  - COMMUNICATION IS **IMPLICIT** THROUGH LOADS AND STORES

SYNCHRONIZATION

THE THREE FORMS OF SYNCHRONIZATION:

1. MUTUAL EXCLUSION
2. BARRIER SYNCHRONIZATION
3. POINT-TO-POINT SYNCHRONIZATION

MUTUAL EXCLUSION

- NEED FOR “MUTUAL EXCLUSION”
- ASSUME THE FOLLOWING STATEMENTS ARE EXECUTED BY 2 THREADS, T1 AND T2

T1	T2
A<- A+1	A<- A+1

- THE PROGRAMMER’S EXPECTATION IS THAT, WHATEVER THE ORDER OF EXECUTION OF THE TWO STATEMENTS, THE FINAL RESULT WILL BE THAT A IS INCREMENTED BY 2
- HOWEVER PROGRAM STATEMENTS ARE NOT EXECUTED IN AN ATOMIC FASHION.
- COMPILED CODE ON A RISC MACHINE WILL INCLUDE SEVERAL INSTRUCTIONS
- A POSSIBLE TEMPORAL INTERLEAVING IS:

T1	T2
r1 <- A	
	r1 <- A
r1 <- r1 + 1	
	r1 <- r1 + 1
A <- r1	
	A <- r1

- IN THE END A IS INCREMENTED BY 1 (NOT 2 AS EXPECTED)

MUTUAL EXCLUSION

MAKE PROGRAM STATEMENT EXECUTION ATOMIC

- CRITICAL SECTIONS
  - PROVIDED BY LOCK (ACQUIRE) AND UNLOCK (RELEASE) PRIMITIVES FRAMING THE STATEMENT(S)
  - MODIFICATIONS ARE “RELEASED” ATOMICALLY AT THE END OF THE CRITICAL SECTION
- SO THE CODE SHOULD BE:

```
T1                  T2
lock(La)            lock(La)            /acquire
A<- A+1             A<- A+1
unlock(La)          unlock(La)          /release
```

HONOR SYSTEM

DEKKER’S ALGORITHMS FOR LOCKING

- ASSUME A AND B ARE BOTH 0 INITIALLY

```
T1                                      T2
A:=1                                    B:=1
while(B==1);          /acquire          while(A==1);
<critical section>                      <critical section>
A:=0                  release           B:=0
```

- AT MOST ONE PROCESS CAN BE IN THE CRITICAL SECTION AT ANY ONE TIME.
- DEADLOCK IS POSSIBLE
- COMPLEX (TO SOLVE DEADLOCK AND TO SYNCHRONIZE MORE THAN 2 THREADS)

BARRIER SYNCHRONIZATION

GLOBAL SYNCHRONIZATION AMONG ALL THREADS

ALL THREADS MUST REACH THE BARRIER BEFORE ANY THREAD IS ALLOWED TO EXECUTE BEYOND THE BARRIER

P1

```
...
BAR := BAR+1;

while (BAR < 2);
```

P2

```
...
BAR := BAR +1;

while (BAR < 2);
```

NOTE: NEED A CRITICAL SECTION TO INCREMENT BAR

POINT-TO-POINT SYNCHRONIZATION

SIGNALS THAT A VALUE IS AVAILABLE

T1	T2
	A = 1;
	FLAG = 1;
while (FLAG==0);	
print A	

SIGNAL SENT BY T1 TO T2 THROUGH FLAG (PRODUCER/CONSUMER SYNCHRONIZATION)

SYNCHRONIZATION

- USE NEW MEMORY INSTRUCTIONS
  - TEST AND SET
  - RESET

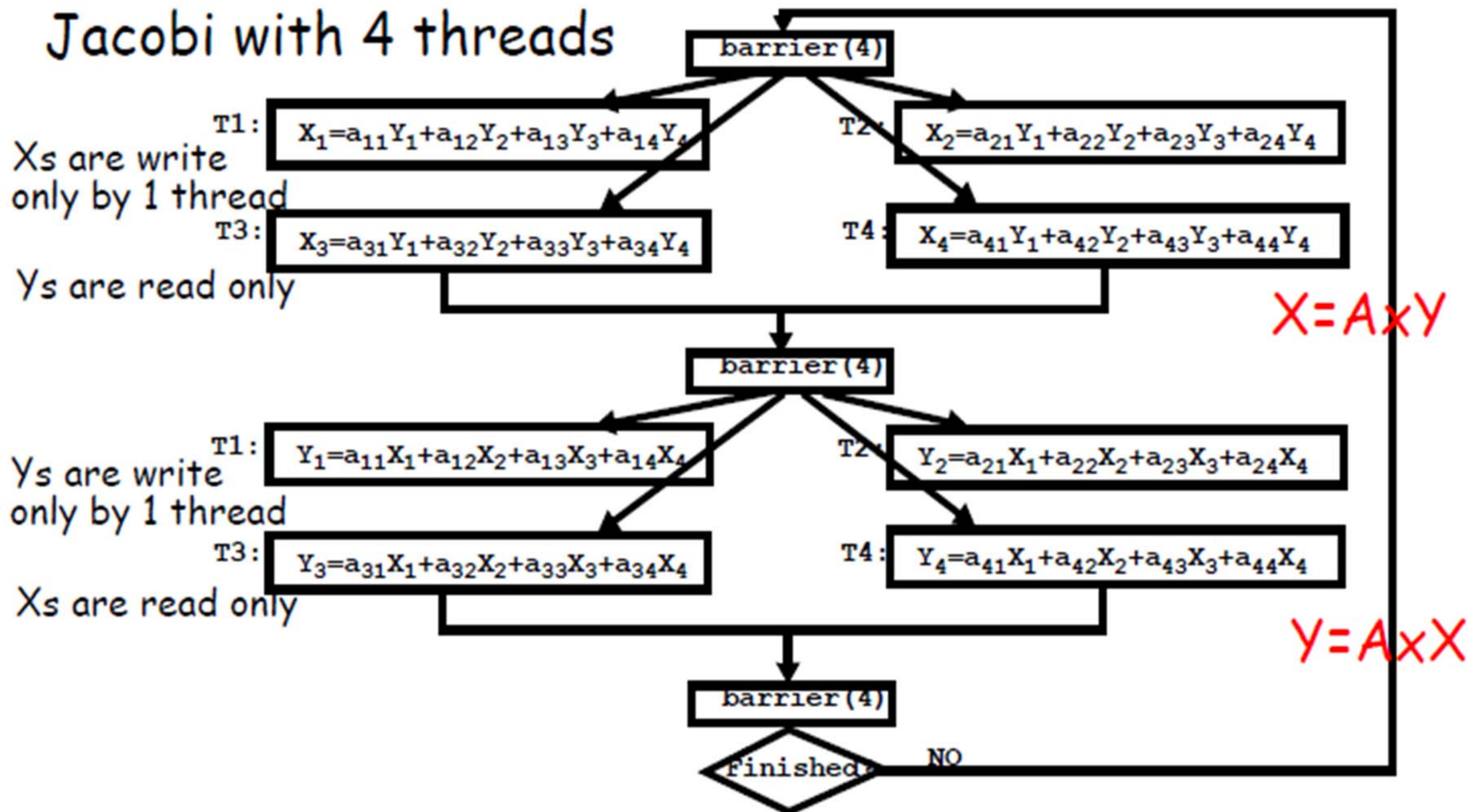
```
Lock:     T&S R1, lock
          BNEZ R1, Lock
          RET

Unlock:   SW R0, lock
          RET
```

  - LOAD LINK
  - STORE CONDITIONAL

```
T&S:      ADDI R1,R0,1
          LL Rx,lock
          SC R1,lock
          BEQZ R1, T&S
          RET
```

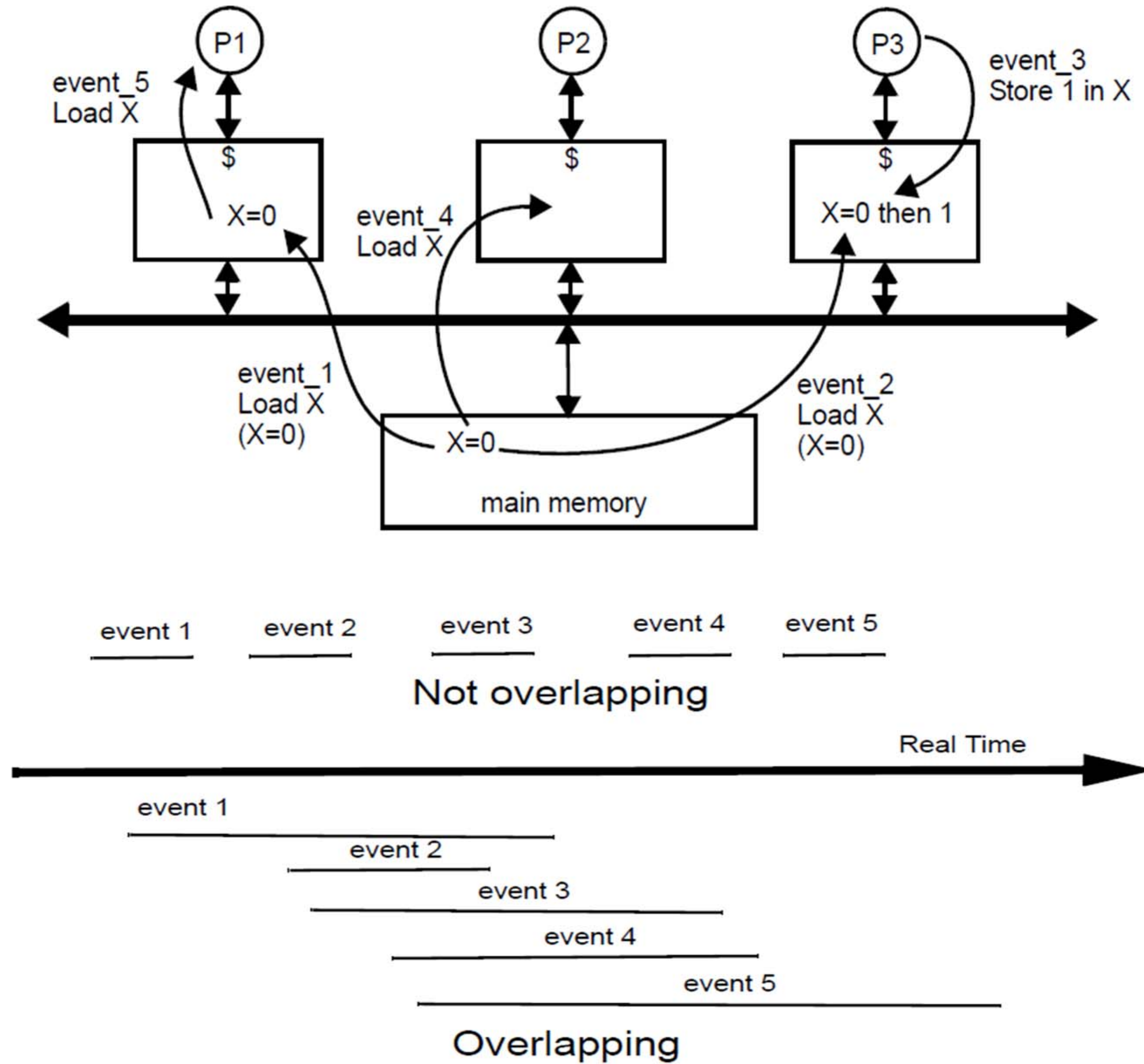
EXAMPLE



- IN 1ST PHASE ACCESSES TO Xs ARE MUTUALLY EXCLUSIVE
- IN 2ND PHASE, MULTIPLE ACCESSES TO Xs (READ-ONLY)
- OPPOSITE IS TRUE FOR Ys

3. STORE ATOMICITY

MEMORY COHERENCE: WHAT'S THE PROBLEM



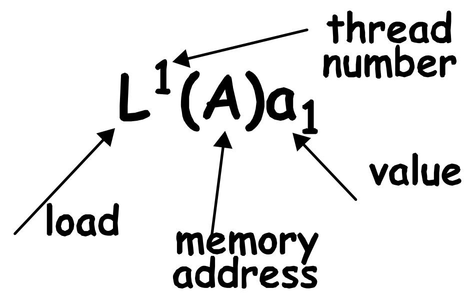
COHERENCE: WHY IT IS SO HARD

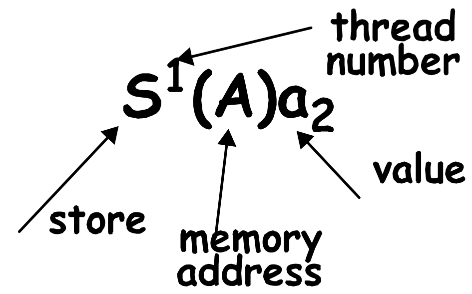
- AFTER EVENT 3, THE CACHES CONTAIN DIFFERENT COPIES. IS THIS STILL COHERENT?
  - THE ANSWER IS “YES”, FOR AS LONG AS P1 OR P2 DO NOT EXECUTE THEIR LOADs
  - A SYSTEM REMAINS COHERENT FOR AS LONG AS INCOHERENCE IS NOT DETECTED. OTHERWISE SAME RESULT.
- CAN THE LOADs IN EVENTS 4 AND 5 RETURN 0 OR 1? COHERENT?
  - YES, EITHER COULD STILL BE “COHERENT” (DEPENDING ON THE DEFINITION)
    - BECAUSE SOFTWARE CANNOT DETECT THE DIFFERENCE
      - P1 OR P2 COULD HAVE RUN SLIGHTLY FASTER, SO THAT EVENTS 4 OR 5 OCCUR BEFORE EVENT 3 IN TIME.
- IN PRACTICE, MUCH MORE COMPLEX BECAUSE MEMORY EVENTS ARE NOT ATOMIC AND DO SOMETIMES OVERLAP IN TIME
  - FOR EXAMPLE, EVENTS 3, 4, AND 5 MAY BE TRIGGERED IN THE SAME CLOCK CYCLE
  - THEY CONFLICT IN SOME PARTS OF THE HARDWARE WHERE THEY ARE SERIALIZED
  - HOWEVER, THEY CAN PROCEED INDEPENDENTLY IN PARALLEL (TEMPORAL OVERLAP) ON THEIR OWN HARDWARE PATHS

STRICT COHERENCE

- "A memory system is coherent if the value returned on a Load is always the value given by the latest Store with the same address"
  - SAME DEFINITION AS FOR UNIPROCESSORS
- DIFFICULT TO EXTEND AS SUCH TO MULTIPROCESSORS
  - ORDER WITHIN EACH THREAD. NO ORDER BETWEEN THREADS.
- CAN BE APPLIED IF
  - MEMORY ACCESSES DO NOT OVERLAP IN TIME
    - TIME ORDER
    - WON'T HAPPEN ALL THE TIME
  - STORES ARE ATOMIC SO THAT ALL COPIES ARE UPDATED INSTANTANEOUSLY (STORE ATOMICITY)
  - STORE/LOAD ORDERS ARE ENFORCED BY ACCESSES TO OTHER MEMORY LOCATIONS (E.G., SYNCHRONIZATION PRIMITIVES)

NOTATIONS

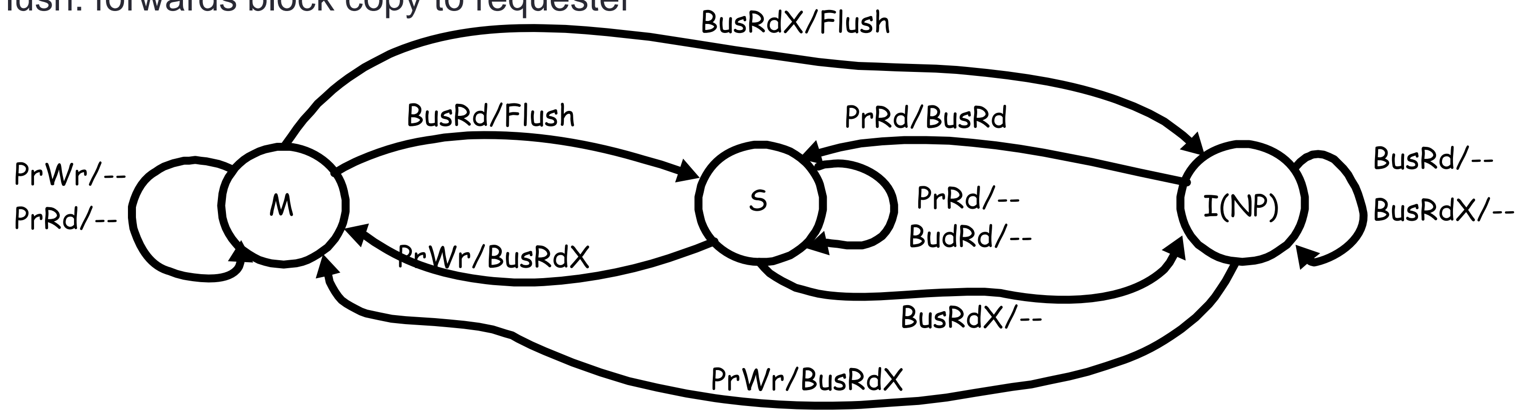




PROTOCOL MACHINE: MSI INVALIDATE PROTOCOL

THE PROTOCOL MACHINES FOR PROTOCOL SPECIFICATION ASSUMES THAT EVENTS ARE ATOMIC (HAPPEN INSTANTANEOUSLY)

- BLOCK STATES:
  - Invalid (I); same as Not Present (NP)
  - Shared (S): one copy or more, memory is clean;
  - Dirty (D) or Modified (M): one copy, memory is stale
- PROCESSOR REQUESTS
  - PrRd or PrWr
- BUS TRANSACTIONS
  - BusRd: requests copy with no intent to modify
  - BusRdX: requests copy with intent to modify
  - Flush: forwards block copy to requester

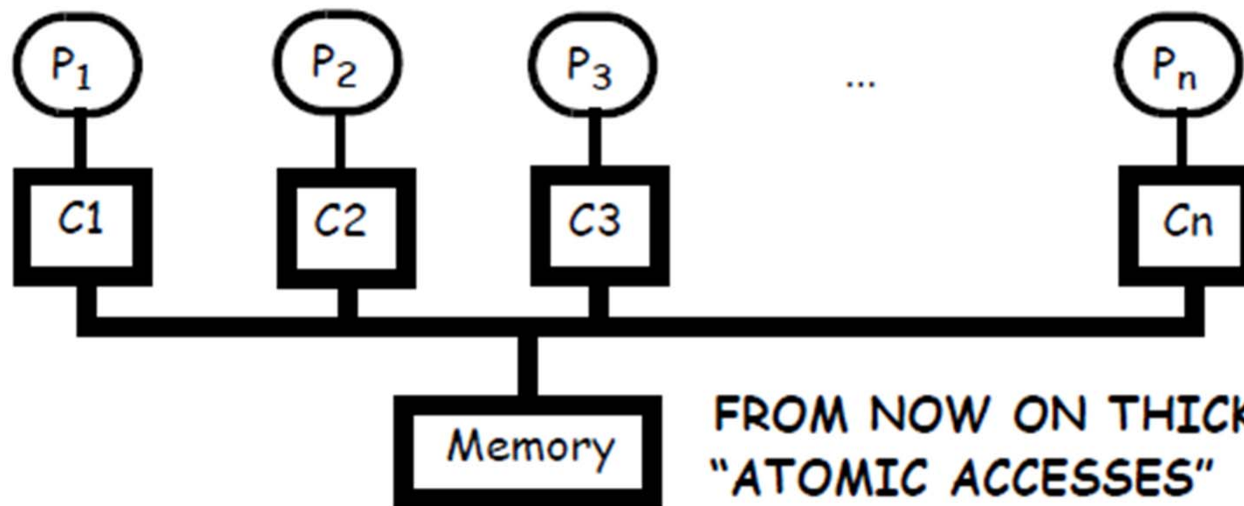


ATOMIC MEMORY ACCESSES

- Strict coherence is applicable if all memory accesses are atomic or do not overlap in time

CLK	T1:	T2:	Comments
t1:	$S^1(A)a_1$		Data not in C2 and dirty in C1
t2:	$L^1(A)a_1$		
t3:	$S^1(A)a_2$		
t4:	$L^1(A)a_2$		
t5:	----------------------------	$L^2(A)a_2$-----------	APT: Read Miss in C2;
t6:	$L^1(A)a_2$		A becomes Shared in C1 and C2; both threads can read A= a_2; No one can write
t7:		$L^2(A)a_2$	
t8:	$L^1(A)a_2$		
t9:	----------------------------	$S^2(A)a_3$--------	APT:C1 is invalidated
t10:		$L^2(A)a_3$	and C2 becomes Dirty
t11:	$S^1(A)a_4$--------------------------------		APT:Store miss in C1; C2 is invalidated
t13:	$S^1(A)a_5$		

MEMORY ACCESS ATOMICITY



- On a coherence transaction, processor blocks, cache gets access to the bus and complete the transaction in remote caches and memory atomically
  - COHERENCE TRANSACTIONS DO NOT OVERLAP IN TIME
  - THUS THE PROTOCOL WORKS EXACTLY AS ITS PROTOCOL MACHINE

THE COHERENCE TRANSACTION IS PERFORMED ATOMICALLY WHEN THE BUS IS RELEASED

Today we must deal with non-atomic transactions

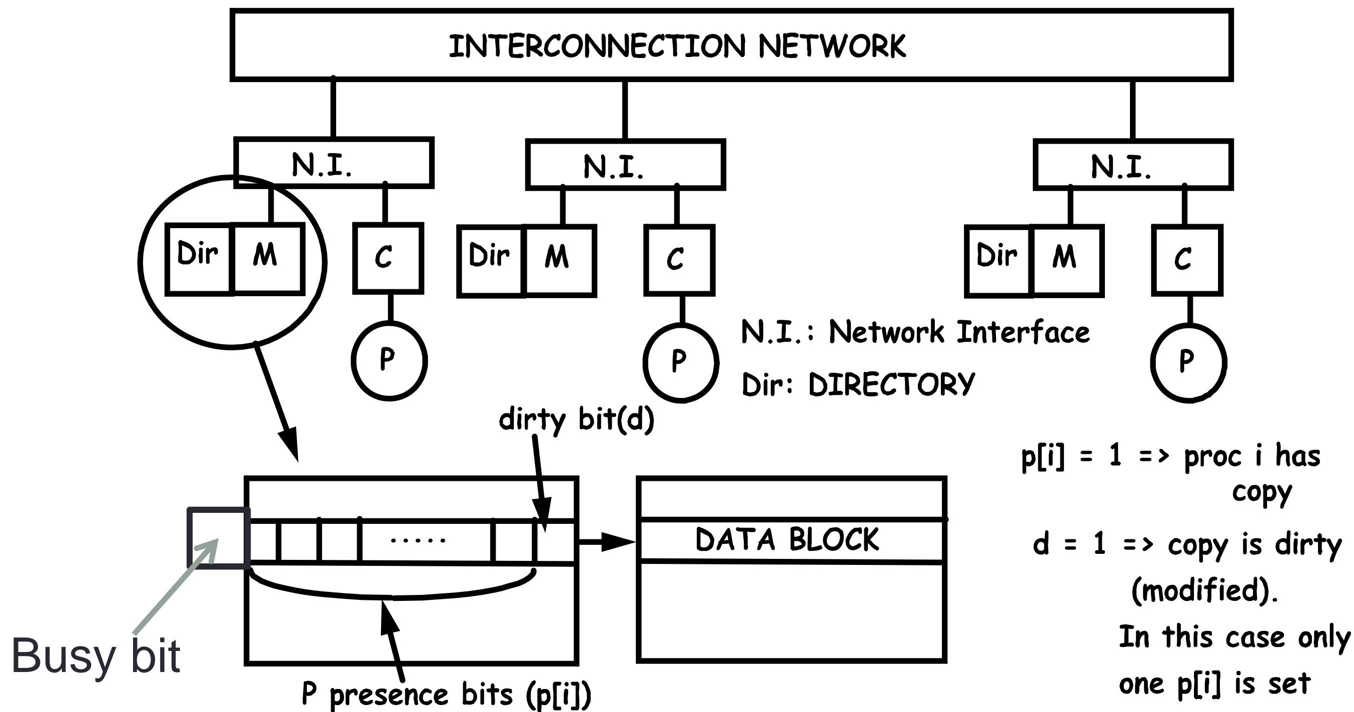
MEMORY ACCESS ATOMICITY--SUFFICIENT CONDITION

- COHERENCE TRANSACTIONS CANNOT HAPPEN INSTANTANEOUSLY
  - MUST MAKE THEM LOOK ATOMIC TO THE SOFTWARE
  - WELL-KNOWN PROBLEM: DATABASE SYSTEMS OR CRITICAL SECTIONS

- CONDITION: MAKE SURE THAT ONLY ONE VALUE IS ACCESSIBLE AT ANY ONE TIME

No thread can observe a new value while any thread can still observe the old value

DIRECTORY-BASED PROTOCOLS



p[i] = 1 => proc i has copy

d = 1 => copy is dirty (modified).
In this case only one p[i] is set

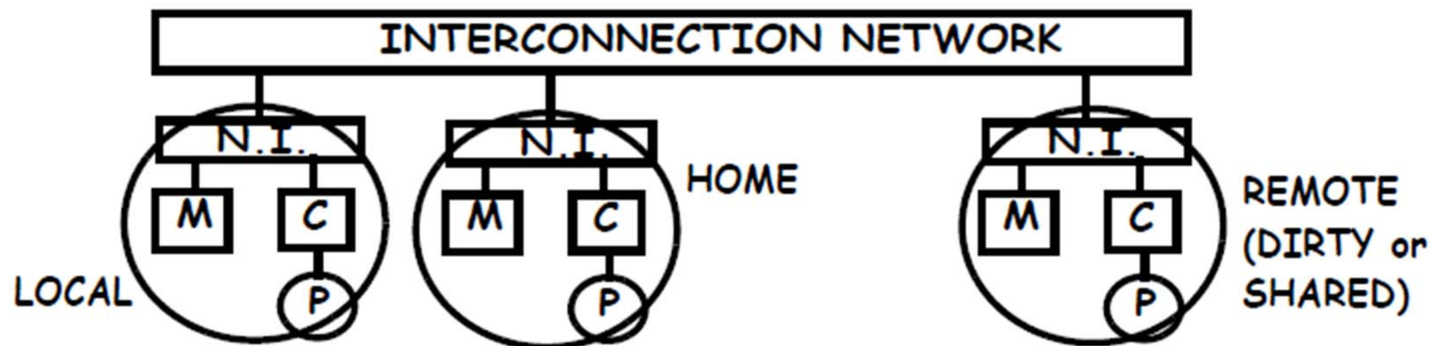
BUSY BIT IS ADDED TO EACH DIRECTORY ENTRY

THE FUNCTION OF THE BUSY BIT IS TO LOCK THE DIRECTORY ENTRY SO THAT THE DIRECTORY CAN MANAGE MULTIPLE TRANSACTIONS ON DIFFERENT ADDRESSES AT THE SAME TIME

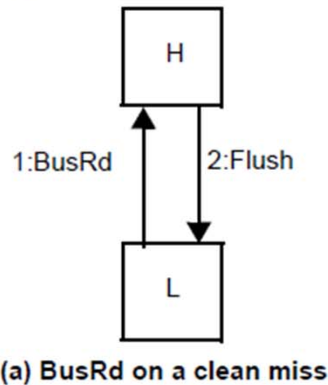
cc-NUMA PROTOCOLS

PROTOCOL AGENTS:

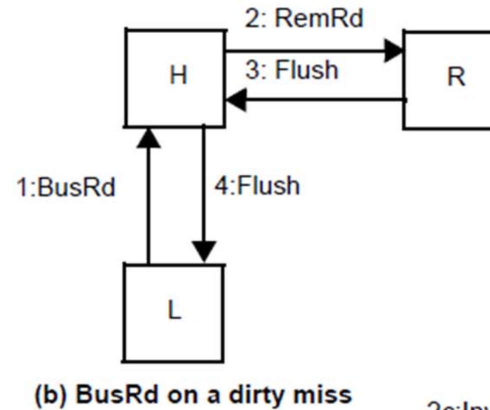
- HOME NODE (H): NODE WHERE THE MEMORY BLOCK AND ITS DIRECTORY ENTRY RESIDE
- LOCAL NODE (R): NODE MAKING THE REQUEST (REQUESTER)
- REMOTE NODE
  - DIRTY (D): NODE HOLDING THE LATEST, MODIFIED COPY
  - SHARED (S) : NODE HOLDING A SHARED COPY



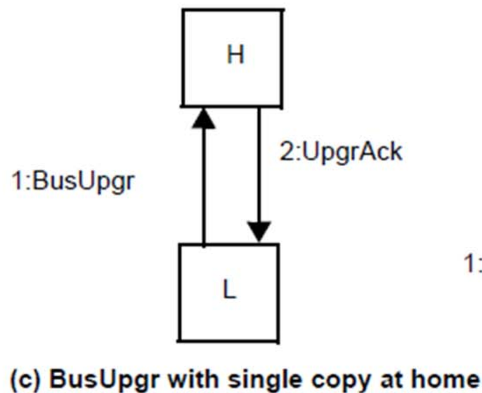
MSI INVALIDATE PROTOCOL IN cc-NUMAs (4-LEG PROTOCOL)



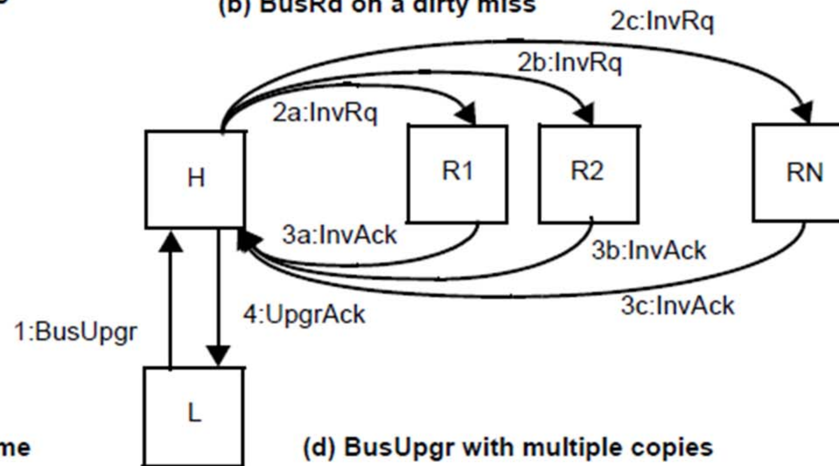
(a) BusRd on a clean miss



(b) BusRd on a dirty miss



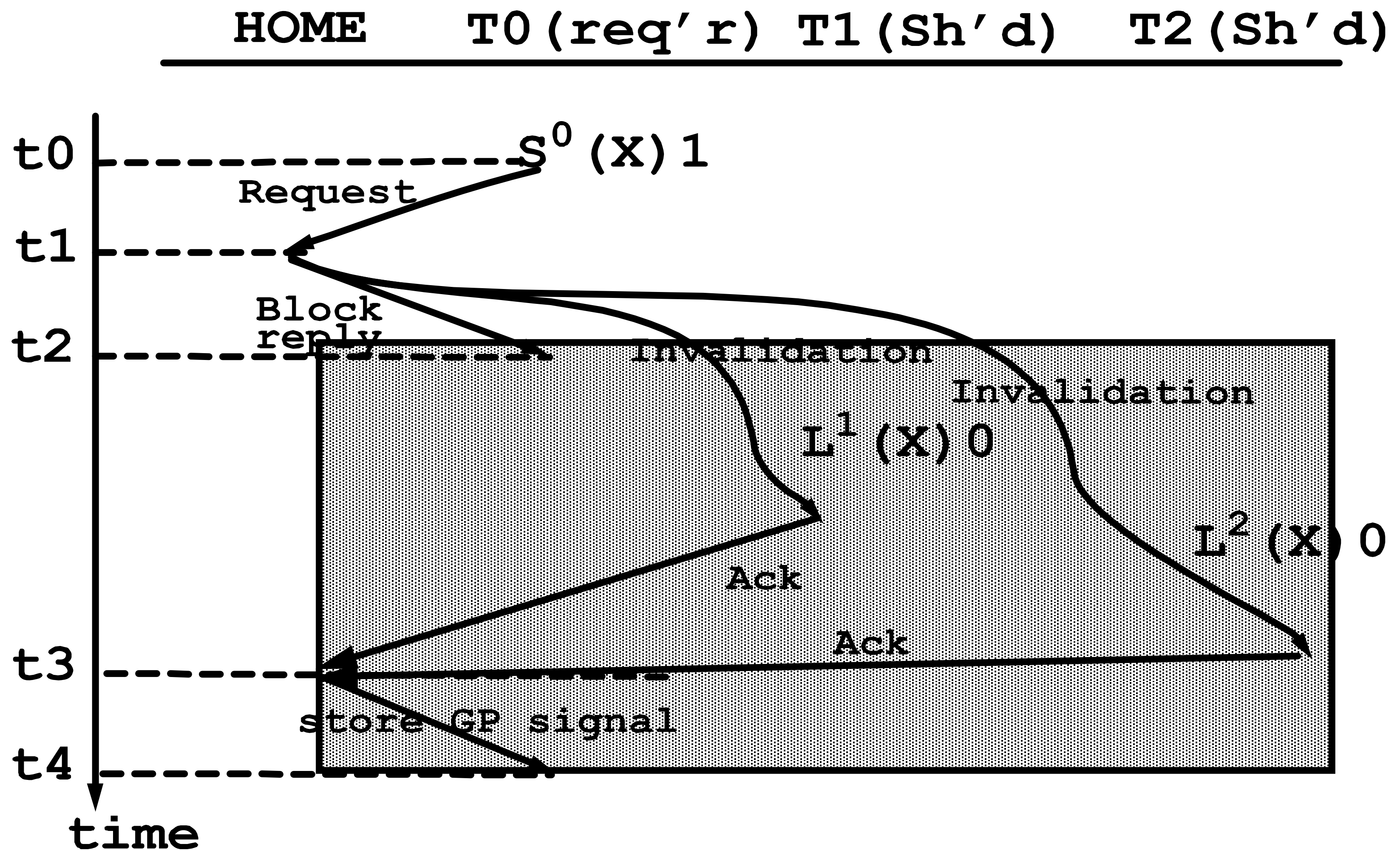
(c) BusUpgr with single copy at home



(d) BusUpgr with multiple copies

BASELINE DIRECTORY PROTOCOL HAS FOUR HOPS ON A REMOTE MISS.

STORE ATOMICITY IN cc-NUMAs



WRITE MISS IN THREAD T0 IN A cc-NUMA WITH MSI INVALIDATE

STORE ATOMICITY IN cc-NUMAs

- Threads must all observe the same value
  - While a new value is propagated, no thread can read the new value (including the writing thread) if other threads can still read the old value

STORE ATOMICITY AND MEMORY INTERLEAVING

- IN A STORE-ATOMIC MEMORY SYSTEM, ONLY ONE VALUE OF EACH ADDRESS IS ACCESSIBLE BY THREADS AT ANY ONE TIME
  - THIS IS THE SAME SITUATION AS INTERLEAVED MEMORIES

➔A STORE-ATOMIC MEMORY SYSTEM BEHAVES LIKE AN INTERLEAVED MEMORY SYSTEM (TYPICAL SYSTEM WITH NO CACHE)

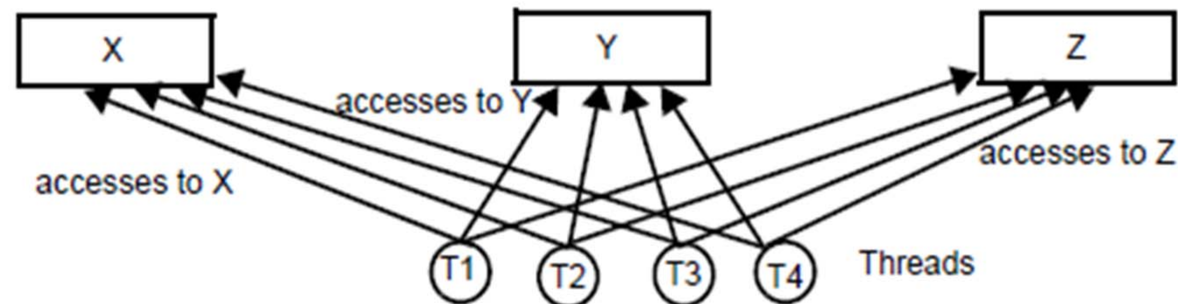


Figure 7.21. Model for Store atomicity and memory interleaving

IT IS WELL-KNOWN THAT INTERLEAVED MEMORY SYSTEMS ARE EQUIVALENT TO A MONOLITHIC (CENTRALIZED) MEMORY

COHERENCE IS NOT SUFFICIENT

Point-to-point Synchronization

ASSUME A AND flag ARE BOTH 0 INITIALLY

P1	P2
...	...
A:=1;	while (flag==0) do nothing;
flag:=1;	print A;
...	...

Communication

ASSUME A AND B ARE BOTH 0 INITIALLY

P1	P2
...	...
A:=1;	print B;
B:=2;	print A;
...	...

Critical section

ASSUME A AND B ARE BOTH 0 INITIALLY

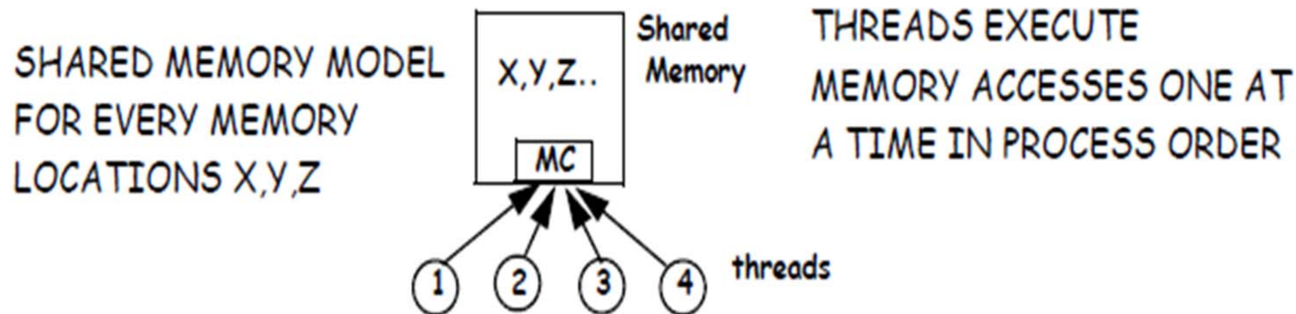
P1	P2
...	...
A:=1	B:=1
while(B==1) do nothing;	while(A==1) do nothing;
<critical section>	<critical section>
A:=0	B:=0

- PROGRAMMER’S INTUITION HERE IS THAT ACCESSES FROM DIFFERENT PROCESSORS ARE “INTERLEAVED” IN PROCESS ORDER

SEQUENTIAL CONSISTENCY

4. SEQUENTIAL CONSISTENCY

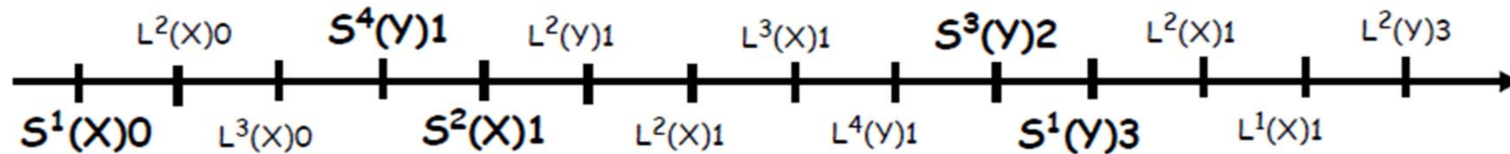
FORMAL MODEL FOR SEQUENTIAL CONSISTENCY



- PROGRAM ORDER OF ALL THREADs IS RESPECTED AND ALL MEMORY ACCESSES ARE ATOMIC BECAUSE OF THE MEMORY CONTROLLER
  - “A MULTIPROCESSOR IS SEQUENTIALLY CONSISTENT IF THE RESULT OF ANY EXECUTION IS THE SAME AS IF THE MEMORY OPERATIONS OF ALL THE PROCESSORS WERE EXECUTED IN SOME SEQUENTIAL ORDER, AND THE OPERATIONS OF EACH INDIVIDUAL PROCESSOR APPEAR IN THE SEQUENCE IN THE ORDER SPECIFIED BY ITS PROGRAM”

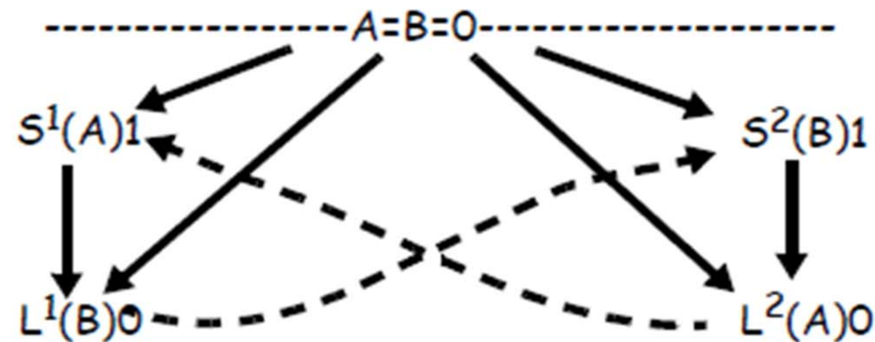
CONDITIONS FOR SC

- WE NEED TO FIND A COHERENT ORDER OF ALL MEMORY ACCESSES IN WHICH ACCESSES OF EACH THREAD ARE IN THREAD ORDER FOR EVERY EXECUTION



- .

ACCESS ORDERING RULES FOR SEQUENTIAL CONSISTENCY



- EXECUTION GRAPH HAS CYCLES ➔ EXECUTION CANNOT BE ORDERED

THIS CAN HAPPEN EVEN IF MEMORY IS STORE ATOMIC– THE BEST IT CAN BE. HOW?

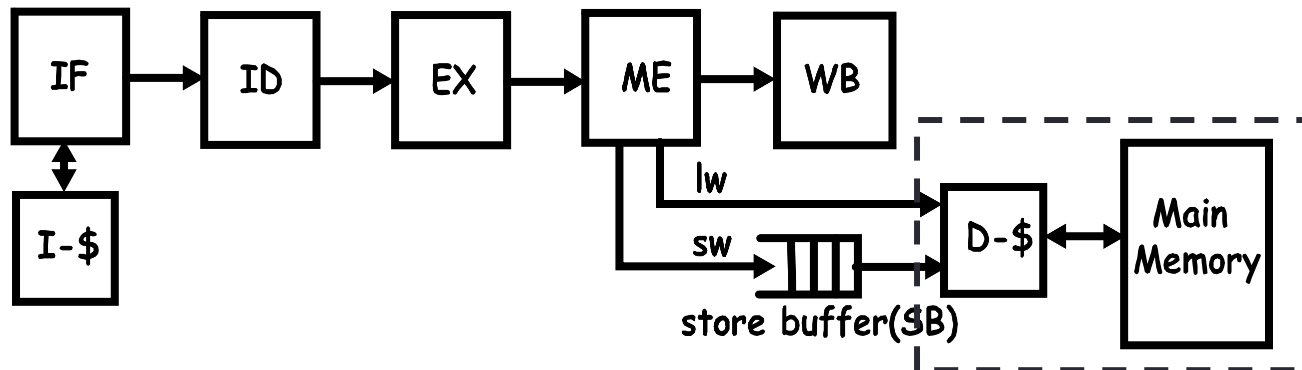
- IN A 5-STAGE PIPELINE WITH STORE BUFFERS
  - THE TWO STORES ARE INSERTED IN THE STORE BUFFER OF EACH PROCESSOR
  - THE TWO LOADS BYPASS THE TWO STORES AND BOTH RETURN 0

CONCLUSION: WE MUST ALSO ADDRESS THE BEHAVIOR OF THE PROCESSOR

SEQUENTIAL CONSISTENCY

- LOOK AT MEMORY ACCESSES ORDERS THAT MUST BE OBSERVED (PERFORMED) GLOBALLY IN THE WHOLE SYSTEM

- IN SC ALL ORDERS MUST BE ENFORCED
- ASSUME MEMORY STORES ARE ATOMIC
- WHAT DOES THIS MEAN FOR IN-ORDER PROCESSORS WITH STORE BUFFERS?



- LOADs ARE BLOCKING, SO LOAD-LOAD AND LOAD-STORE ORDERS ARE ENFORCED
- STOREs ARE NON-BLOCKING (THEY MOVE TO SB)
- STORE-LOAD: LOADs MUST BLOCK IN ME UNTIL SB IS EMPTY
- STORE-STORE: STOREs MUST BE EXECUTED ONE BY ONE FROM SB IN THE STORE ATOMIC MEMORY SYSTEM

CONCLUSION ON SC

- PROBLEM WITH STORE BUFFERS
  - SC CANNOT REALLY TAKE ADVANTAGE OF STORE BUFFERS
- PROBLEM WITH COMPILERS
  - TAKE AGAIN PT-TO-PT SYNCHRONIZATION

P1	P2
...	...
A:=1;	while(flag==0)do nothing;
flag:=1;	print A;
...	...

  - SINCE THERE IS NO DEPENDENCY BETWEEN flag AND A, THE COMPILER MAY REORDER THE TWO INSTRUCTIONS IN P1
  - MIGHT ALSO REMOVE THE LOOP ON flag IN P2

NEXT IDEA: CHANGE THE RULES
MEMORY CONSISTENCY MODELS
MAY RELY OR NOT ON SYNCHRONIZATION

5. RELAXED MEMORY CONSISTENCY MODELS

MEMORY CONSISTENCY MODELS

- FIRST: WHY SHOULD WE BOTHER ABOUT THE PROGRAMMER ANYWAY?
  - WHY NOT DESIGN MEMORY ACCESS RULES THAT ARE HARDWARE FRIENDLY AND THEN ASK THE PROGRAMMER TO ABIDE BY THEM??
- SECOND: ARE THERE OTHER PROGRAMMER'S INTUITIONS BESIDES SC?
  - SUCH AS PROVIDED BY SYNCHRONIZATION?
  - WHENEVER SHARED VARIABLES ARE READ/WRITE ACCESSES TO THEM SHOULD BE PROTECTED BY LOCKS

BAR IS 0 INITIALLY

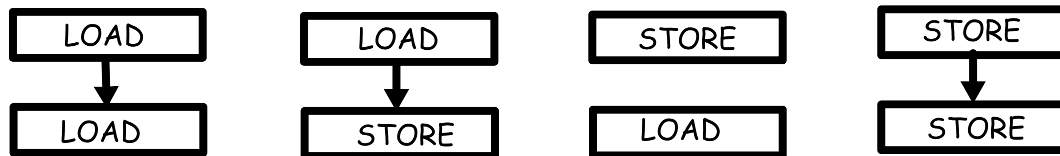
P1	P2
A:=1;	BARRIER(BAR,2);
B:=2;	R1:=A;
BARRIER(BAR,2);	R2:=B;

BECAUSE OF THE BARRIER, THE RESULT IS ALWAYS SC AND P2 RETURNS (A=1 AND B=2)

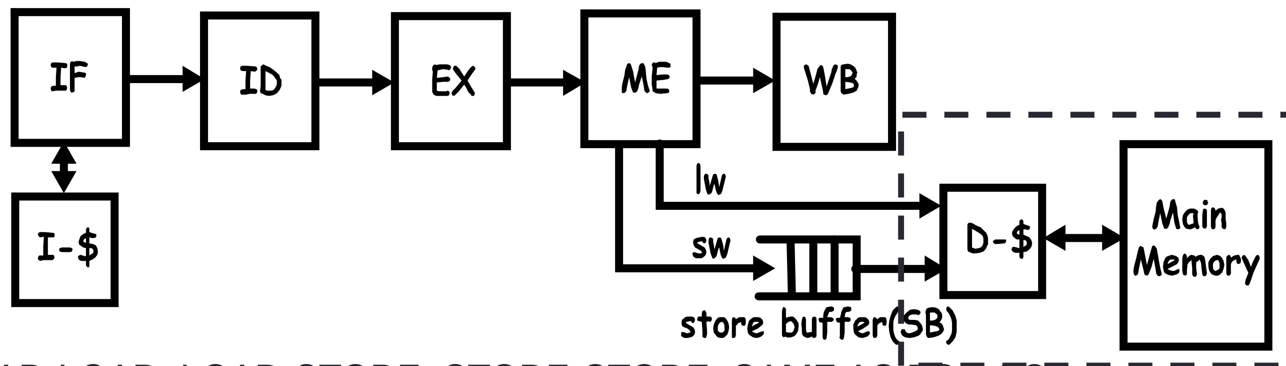
- IN ALL CASES WE NEED A MEMORY ACCESS ORDERING MODEL ON WHICH PROGRAMMERS AND MACHINE ARCHITECTS CAN AGREE
  - IT'S CALLED THE MEMORY CONSISTENCY MODEL
- THIS MODEL MUST BE PART OF THE ISA DEFINITION, SINCE IT IS AT THE INTERFACE BETWEEN SOFTWARE AND HARDWARE

RELAXED MEMORY CONSISTENCY MODELS

- WE CAN RELAX SOME OF THE ACCESS ORDERS OF EACH THREAD
- THE MAJOR RELAXATION IS THE STORE-TO-LOAD ORDER: LOADS CAN BYPASS PRIOR STORES



  - STORES FROM THE SAME PROCESSOR MUST BE OBSERVED BY ALL OTHER PROCESSORS IN THREAD ORDER (BECAUSE OF STORE-STORE AND LOAD-LOAD ORDERS)
- WHAT DOES THIS MEAN FOR IO PROCESSORS?
  - ASSUME MEMORY IS ATOMIC: LOADS RETURN VALUES FROM MEMORY ONLY WHEN THE VALUES ARE GPed



  - LOAD-LOAD, LOAD-STORE, STORE-STORE: SAME AS FOR SC
  - STORE-LOAD: LOADs DON'T WAIT FOR SB EMPTY

RELAXING STORE-TO-LOAD ORDERS

- EXAMPLE: SUN MICRO TOTAL STORE ORDER (TSO)
  - DEKKER'S ALGORITHM DOES NOT WORK
  - POINT-TO-POINT COMMUNICATION STILL WORKS

- VALUES MAY OR MAY NOT BE FORWARDED FROM SB TO LOADS
  - NO FORWARDING FROM SB → STORE ATOMIC
  - FORWARDING FROM SB → NOT STORE ATOMIC (as in TSO)
    - THIS MEANS THAT SOME LOADS IN TSO RETURN VALUES EVEN IF THE VALUES ARE NOT VISIBLE TO THE REST OF THE MEMORY SYSTEM

IMPLICATIONS OF RELAXING STORE-TO-LOAD ORDER

- BECAUSE MEMORY ACCESS ORDERS ARE WEAKENED, MORE EXECUTIONS ARE VALID UNDER RELAXED MEMORY MODELS
  - EXAMPLE:

INIT: A=B=0

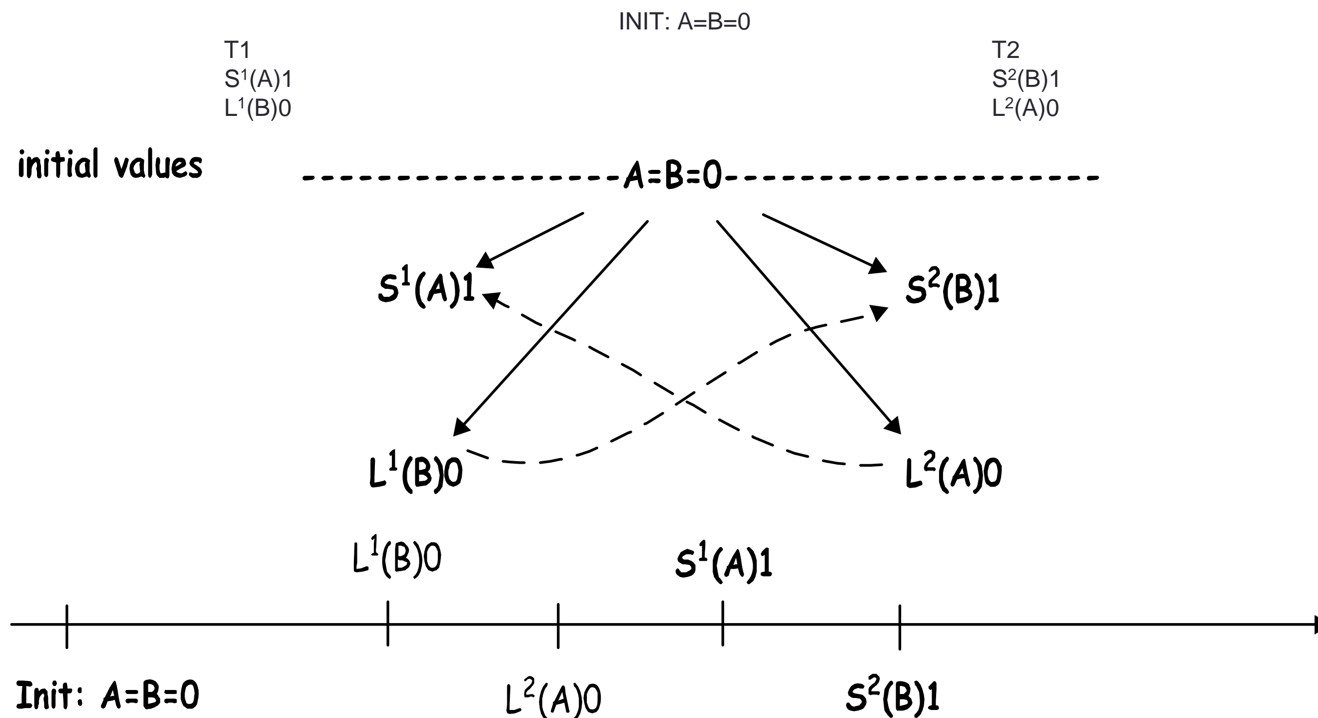
T1	T2
$S^1(A)1$	$S^2(B)1$
$L^1(B)0$	$L^2(A)0$

    - IN SC A LOADS CANNOT BE EXECUTED BEFORE A PREVIOUS STORE IS GLOBALLY PERFORMED. THUS, THIS IS **NOT** A VALID EXECUTION.
    - NOT SO IN RELAXED MEMORY MODELS!!!!!THIS IS A VALID EXECUTION WITH STORE-TO-LOAD RELAXATION

- AS A RESULT, SOME PROGRAMS WRITTEN FOR A STRONGER MODEL WON'T RUN CORRECTLY ON WEAKER MEMORY MODELS
    - IN THE CASE OF STORE-TO-LOAD RELAXATION
      - DEKKER'S ALGORITHM DOES NOT WORK
      - POINT-TO-POINT COMMUNICATION STILL WORKS
        - BECAUSE STORES FROM THE SAME PROCESSOR MUST BE OBSERVED BY ALL OTHER PROCESSORS IN THEIR THREAD ORDER (BECAUSE STORE-STORE AND LOAD-LOAD ORDERS ARE ENFORCED)

NO FORWARDING FROM SB

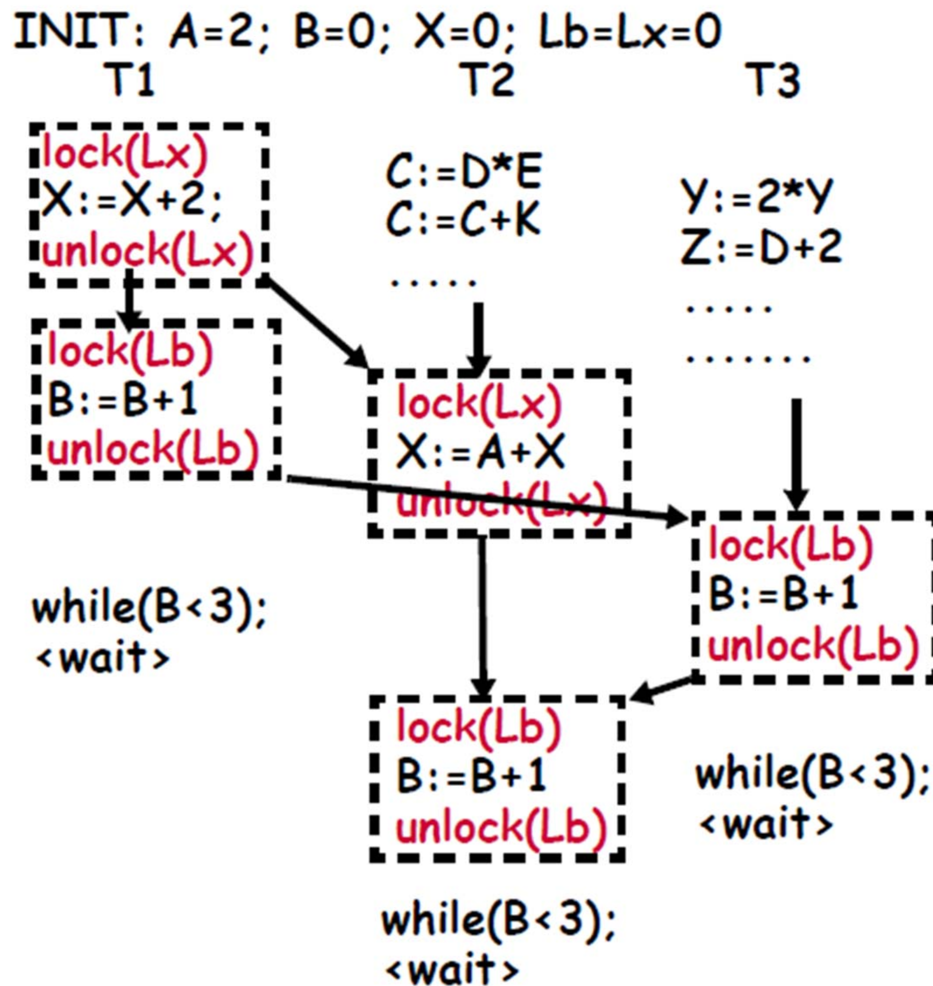
- VALID EXECUTION



- NO ARROW BETWEEN STORE AND LOAD IN EACH THREAD
- NO GLOBAL ORDER BETWEEN THE STORE AND LOAD IN EACH THREAD

STILL GLOBAL MEMORY ORDER IS STORE ATOMIC. THE REASON THIS HAPPENS HAS TO DO WITH THE BEHAVIOR OF THE PROCESSOR

MCM USING SYNCHRONIZATION



Locks use special instructions

- RMW (for lock)
- special stores (for unlocks)

Treat locks as FENCES for all memory accesses

Also called "SYNC" in PowerPC's memory model

- lock and unlock act as fences
- perform all preceding accesses in T.O. before attempting a SYNC
- perform SYNC before attempting any following accesses in T.O.

In between two SYNC Ops, the order of memory Ops is arbitrary

WEAK ORDERING

- MULTITHREADED EXECUTION USES LOCKING MECHANISMS TO AVOID RACE CONDITIONS.
- EXECUTIONS INCLUDE VARIOUS PHASES:
  - ACCESSES TO PRIVATE OR READ-ONLY SHARED DATA, OR
  - ACCESSES TO SHARED MODIFIABLE DATA, PROTECTED BY LOCKS AND BARRIERS.
- IN EACH PHASE THE THREAD HAS EXCLUSIVE ACCESS TO ALL ITS DATA, MEANING
  - NO OTHER THREAD CAN WRITE TO THEM, OR
  - NO OTHER THREAD CAN READ THE DATA IT MODIFIES
- ACCESSES TO SYNCHRONIZATION DATA (INCLUDING ALL LOCKS AND SHARED DATA IN SYNCHRONIZATION PROTOCOLS) ARE TREATED DIFFERENTLY BY THE HARDWARE FROM ACCESSES TO OTHER SHARED AND PRIVATE DATA.
  - THEY ACT AS FENCES ON ALL ACCESSES
  - MUST GLOBALLY PERFORM ALL ACCESS PRECEDING SYNC ACCESS IN T.O.
  - MUST GLOBALLY PERFORM SYNC ACCESS BEFORE ALL FOLLOWING ACCESSES IN T.O.

WEAK ORDERING

- WEAK ORDERING RELIES ON (HOPEFULLY) CORRECTLY WRITTEN PROGRAMS IN WHICH RACES ON SHARED DATA ARE PREVENTED THROUGH SYNCHRONIZATION
  - A RACE OCCURS WHEN TWO THREADS ACCESS THE SAME ADDRESS AND AT LEAST ONE OF THE ACCESSES IS A WRITE
  - WHICH MEANS THAT THERE WILL BE HAZARDS
  - MULTITHREADED EXECUTION RELIES ON LOCKING MECHANISMS TO AVOID DATA RACE CONDITIONS.

"DATA-RACE FREE PROGRAMS"

- EXECUTIONS OF DATA-RACE FREE PROGRAMS INCLUDE VARIOUS PHASES:
  - ACCESSES TO PRIVATE OR READ-ONLY SHARED DATA, OR
  - ACCESSES TO SHARED MODIFIABLE DATA, PROTECTED BY LOCKS AND BARRIERS.
  - IN EACH PHASE THE THREAD HAS EXCLUSIVE ACCESS TO ALL ITS DATA, MEANING
    - NO OTHER THREAD CAN WRITE TO THEM, OR
    - NO OTHER THREAD CAN READ THE DATA IT MODIFIES

WEAK ORDERING

- ACCESSES TO OTHER (NON-SYNC) SHARED AND PRIVATE DATA MUST ENFORCE UNIPROCESSOR DEPENDENCIES ON SAME ADDRESS
  - OTHERWISE, NO REQUIREMENT
  - EVEN COHERENCE IS NOT A REQUIREMENT

- VARIABLES THAT ARE USED FOR SYNCHRONIZATION MUST BE DECLARED AS SUCH (e.g., flag, A and B below)
  - SO THAT EXECUTION ON THESE VARIABLES ARE "SAFE".

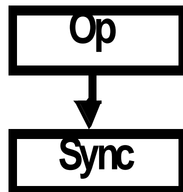
A=flag=0 initially

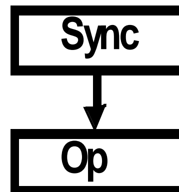
T1	T2
A:=1;	while(flag==0)do nothing;
flag:=1;	print A;
...	...

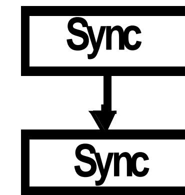
Flag MUST BE DECLARED AS SYNC VARIABLE

WEAK ORDERING

- A RMW ACCESS TO A MEMORY LOCATION IS GLOBALLY PERFORMED ONCE BOTH THE LOAD AND THE STORE IN THE RMW ACCESS ARE GLOBALLY PERFORMED.
  - ATOMICITY MUST ALSO BE ENFORCED BETWEEN THE TWO ACCESSES (EASIER IN WRITE-INVALIDATE PROTOCOLS--TREAT THE T&S AS A STORE IN THE PROTOCOL)
- SYNC OPERATION MUST BE RECOGNIZABLE BY THE HARDWARE AT THE ISA LEVEL
  - RMW (T&S)
  - SPECIAL LOADS AND STORES FOR SYNC VARIABLE ACCESSES
- ORDERS TO ENFORCE:



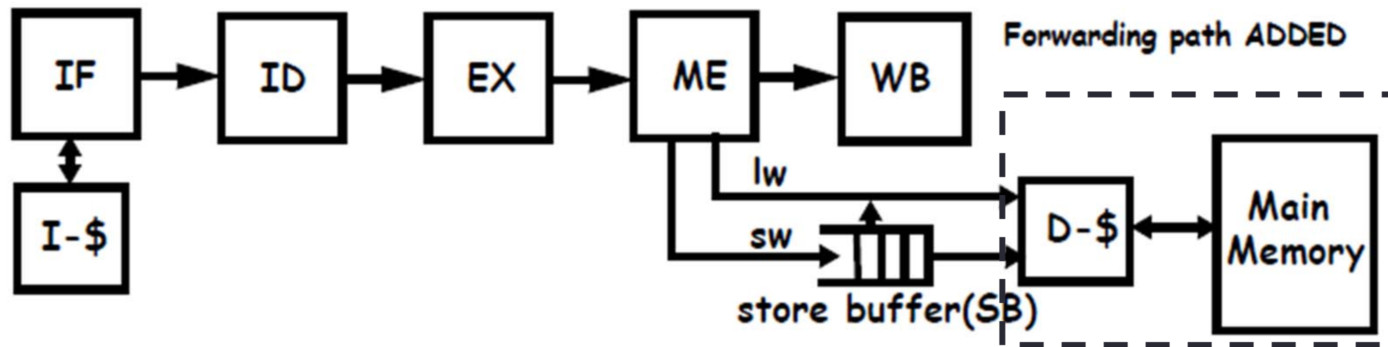




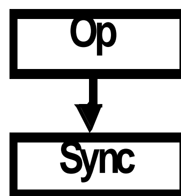
- Op = regular load or store
- Sync = any synchronization access, e.g., swap, T&S, special load/store

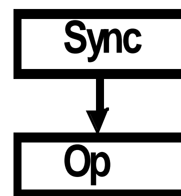
WEAK ORDERING

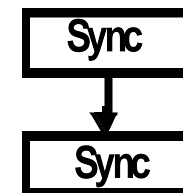
- WHAT DOES IT MEAN FOR IO PROCESSORS?
  - NOTE: HERE REGULAR LOADS CAN RETURN VALUES EVEN IF THEY ARE NOT GPed



- REGULAR STORES IN THE STORE BUFFER CAN BE EXECUTED IN ANY ORDER, IN PARALLEL
- REGULAR LOADS NEVER WAIT FOR STORES AND CAN BE FORWARDED TO
- WHEN A SYNC ACCESS IS EXECUTED, IT IS TREATED DIFFERENTLY:
  - IT BLOCKS IN THE MEMORY STAGE UNTIL ALL STORES IN THE STORE BUFFER ARE GLOBALLY PERFORMED WHICH ENFORCES OP-TO-SYNC.
- SYNC-TO-OP AND SYNC-TO-SYNC ORDERS ARE AUTOMATICALLY ENFORCED BY IO PROCESSOR

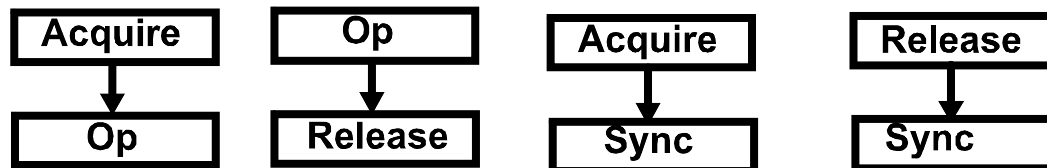






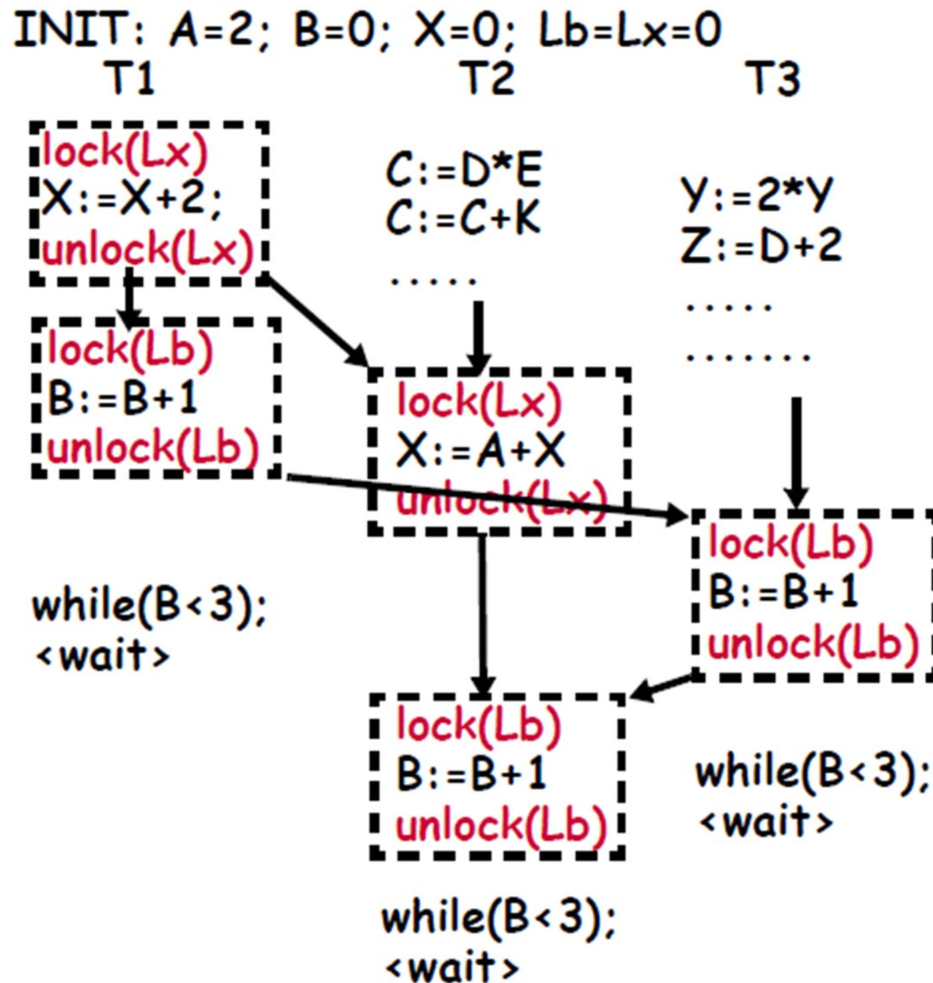
RELEASE CONSISTENCY

- A REFINEMENT OF WEAK ORDERING
  - DISTINGUISHES BETWEEN ACQUIRES AND RELEASES OF LOCK
  - ACQUIRES MUST BE GLOBALLY PERFORMED BEFORE STARTING ANY FOLLOWING MEMORY OPS
  - ALL MEMORY OPS MUST BE GLOBALLY PERFORMED BEFORE A RELEASE CAN START
  - MORE “RELAXED” THAN WEAK ORDERING
- PROGRAMMERS MUST MARK (LABEL) SYNCHRONIZATION ACCESSES AS “ACQUIRES” OR “RELEASES”
  - ACQUIRES AND RELEASES MUST BE SEQUENTIALLY CONSISTENT
- ORDERS TO ENFORCE GLOBALLY



  - Op = REGULAR LOAD OR STORE
  - Sync = ACQUIRE OR RELEASE

MCM USING SYNCHRONIZATION



Locks use special instructions
RMW (for lock)
special stores (for unlocks)

Treat locks as FENCES for all memory accesses
Also called “SYNC” in PowerPC’s memory model

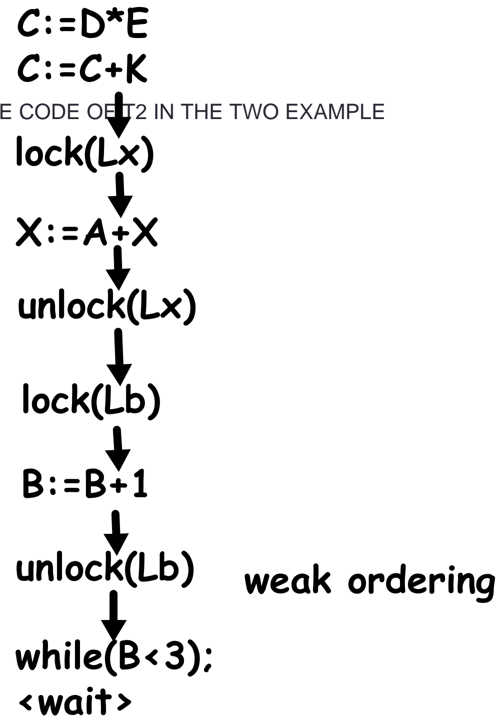
lock and unlock act as fences, i.e.

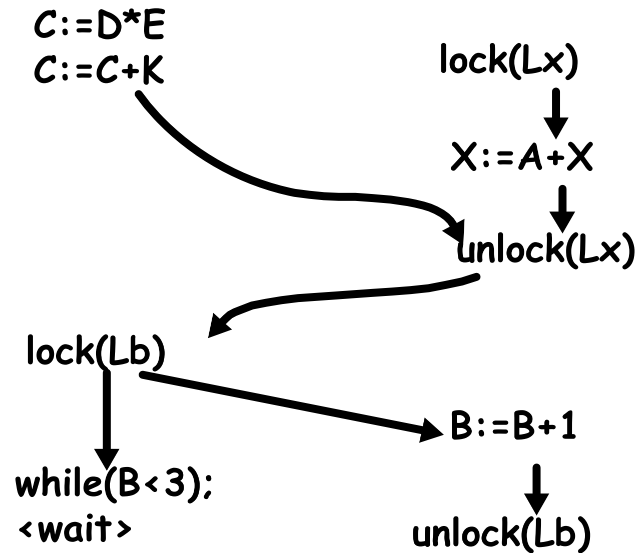
- perform all preceding accesses in T.O. before attempting a SYNC
- perform SYNC before attempting any following accesses in T.O.

In between two SYNC Ops, the order of memory Ops is arbitrary

RELEASE CONSISTENCY

- CONSIDER THE CODE OF T2 IN THE TWO EXAMPLE





WEAK ORDERING

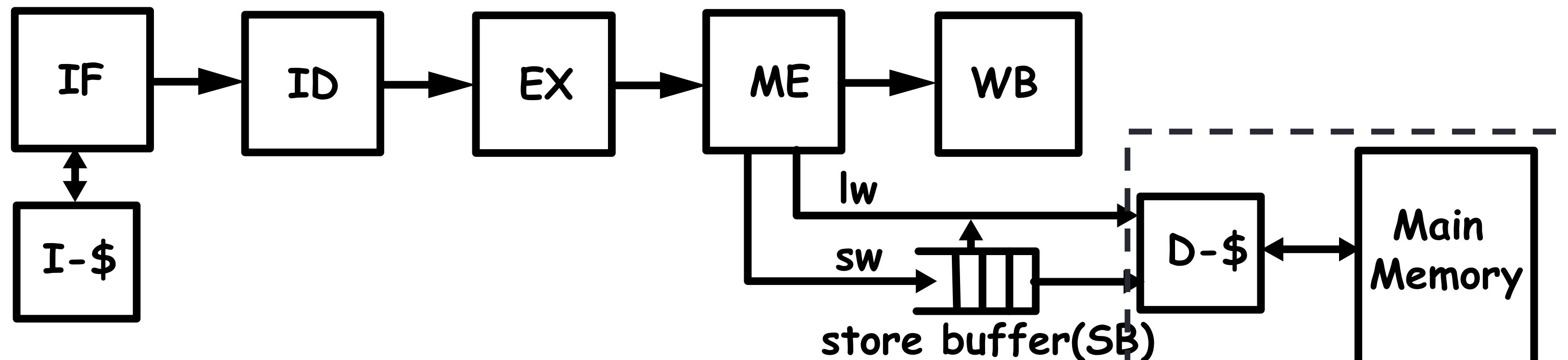
RELEASE CONSISTENCY

- A LOT MORE CONCURRENCY INSIDE PROCESSOR IN RELEASE CONSISTENCY vs WEAK ORDERING
  - THE CODE BEFORE UNLOCK(Lx) IN T2 CAN BE EXECUTED IN PARALLEL
  - THE CODE AFTER LOCK(Lb) IN T2 CAN BE EXECUTED IN PARALLEL

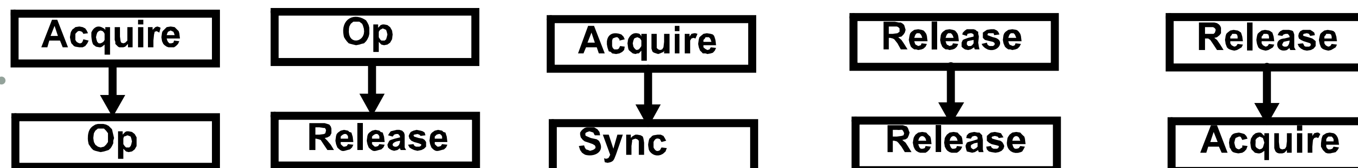
THE PROBLEM IS TO BE ABLE TO TAKE ADVANTAGE OF THIS ADDITIONAL CONCURRENCY WITHIN EACH THREAD IN A REGULAR PROCESSOR PLUS PROGRAMMING DIFFICULITES

RELEASE CONSISTENCY

- WHAT DOES IT MEAN FOR IO PROCESSOR?

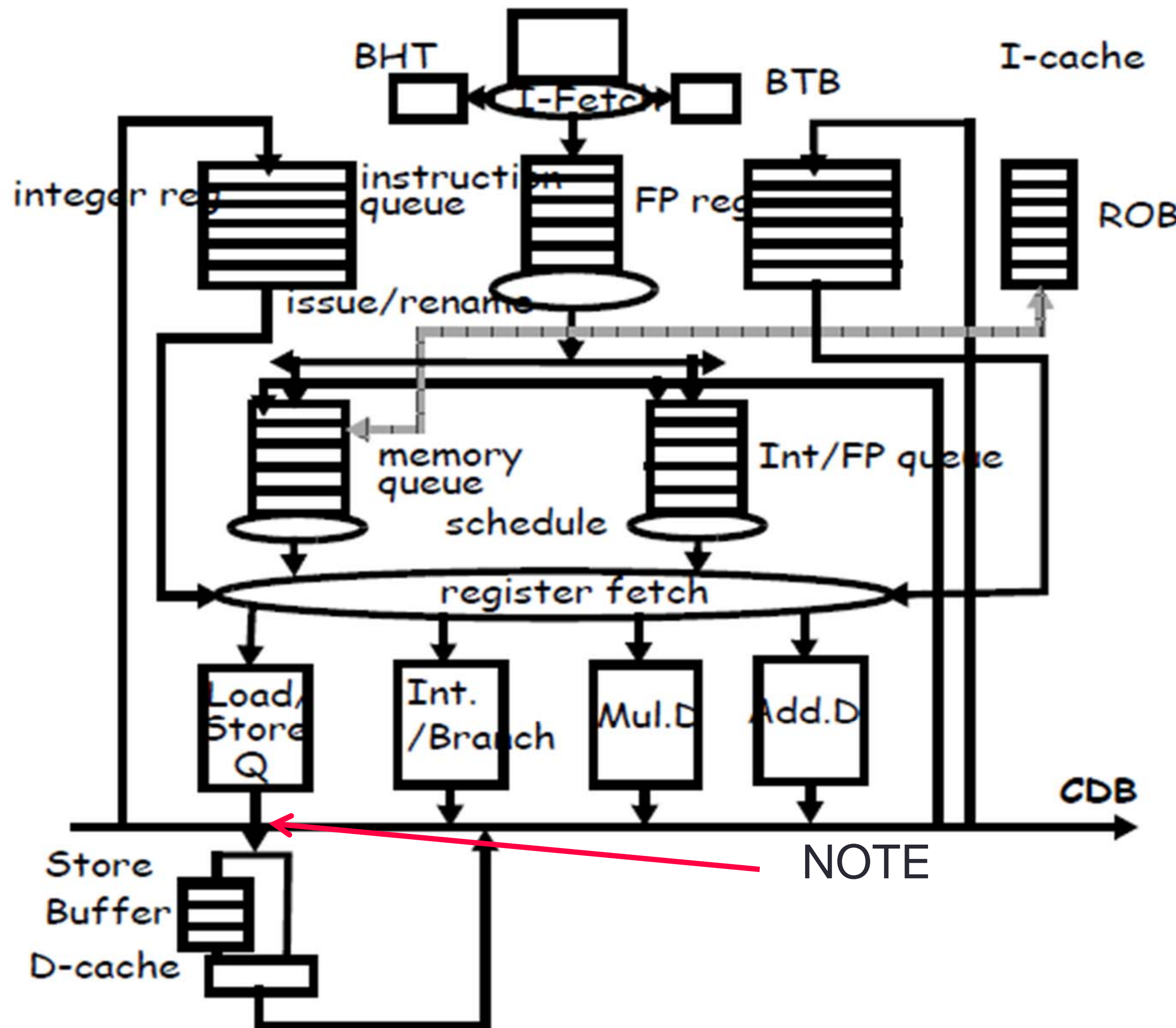


- NO ORDERING RULES AMONG REGULAR LOADs/STOREs
  - REGULAR STORES IN THE STORE BUFFER CAN BE EXECUTED IN ANY ORDER, IN PARALLEL
  - REGULAR LOADS NEVER WAIT FOR REGULAR STORES IN SB AND CAN BE FORWARDED TO
- ACQUIRES ARE BLOCKING: THUS ACQUIRE-OP AND ACQUIRE-SYNC ORDERS ARE AUTOMATICALLY ENFORCED
- A RELEASE IS INSERTED IN THE STORE BUFFER WITH REGULAR STORES.
  - OP-RELEASE AND RELEASE-RELEASE ORDERS: RELEASES MUST WAIT IN SB UNTIL ALL PRIOR STORES AND RELEASES ARE GP (LOADS PRIOR TO THE RELEASE HAVE RETIRED)
  - RELEASE-ACQUIRE ORDER: AN ACQUIRE WAITS UNTIL ALL PRIOR RELEASES IN SB HAVE BEEN GPed



6. SPECULATIVE VIOLATIONS OF MEMORY CONSISTENCY MODELS

OoO PROCESSORS



ALL HAZARDS ON SAME MEMORY ADDRESS (RAW, WAW and WAR) ARE SOLVED

- IN LOAD/STORE QUEUE
- HELPS COHERENCE;
- DOES NOTHING FOR THE MCM --DIFFERENT ADDRESSES

LOADs/STOREs HANDLING IN SPECULATIVE OoO PROCESSORS

- WHEN THE LOAD ADDRESS IS KNOWN, THE LOAD CAN ISSUE TO CACHE
  - PROVIDED NO STORE WITH THE SAME OR AN UNKNOWN ADDRESS IS AHEAD IN THE L/S QUEUE (CONSERVATIVE MEMORY DISAMBIGUATION)
  - CAN EVEN BE ISSUED BEFORE ADDRESSES OF PREVIOUS STORES ARE KNOWN (SPECULATIVE MEMORY DISAMBIGUATION)
- IN BOTH CASES THE VALUE OF A LOAD ISSUED TO CACHE IS RETURNED AND USED SPECULATIVELY BEFORE THE LOAD RETIRES
  - THE LOAD VALUE IS NOT BOUND UNTIL THE LOAD RETIRES
  - THE VALUE CAN BE RECALLED FOR ALL KINDS OF REASONS (MISSPREDICTED BRANCH, EXCEPTION, ETC...)
- STORES CANNOT UPDATE CACHE UNTIL THEY REACH THE TOP OF THE ROB
  - AS STORES REACH THE TOP OF THE REORDER BUFFER AND OF THE L/S QUEUE THEY COMMIT BY MOVING TO THE STORE BUFFER

CONSERVATIVE MCM ENFORCEMENT

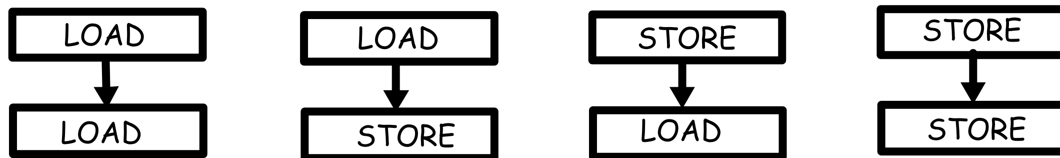
- THE ONLY TIME THAT IT IS KNOWN FOR SURE THAT A MEMORY ACCESS CAN BE PERFORMED IS WHEN IT REACHES THE TOP OF ROB
  - BEFORE THAT AN ACCESS (LOAD OR STORE) IS SPECULATIVE
  - STORES MUST WAIT UNTIL THEN ANYWAY
- COULD WAIT TO EXECUTE AND PERFORM A LOAD IN CACHE UNTIL IT REACHES THE TOP OF ROB
  - DOWNSIDE: NO MEMORY LATENCY TOLERANCE DUE TO OoO EXECUTION
- IN THIS CONSERVATIVE SCHEME, LOADs AND STOREs CAN STILL BE *PREFETCHED* IN CACHE (NON-BINDING PREFETCHES)
  - IF THE LOAD OR STORE DATA IS IN THE RIGHT STATE IN CACHE AT THE TOP OF THE ROB, PERFORMING THE ACCESS TAKES ONE CACHE CYCLE.
  - STILL, LOAD VALUES CAN NOT BE USED SPECULATIVELY

NEXT IDEA: EXPLOIT SPECULATIVE EXECUTION TO SPECULATIVELY VIOLATE MEMORY ORDERS

WORKS BECAUSE ORDER VIOLATIONS ARE RARE

SPECULATIVE VIOLATIONS OF SEQUENTIAL CONSISTENCY

- LET'S LOOK AGAIN AT ORDERS TO ENFORCE GLOBALLY



- LOAD-STORE AND STORE-STORE ORDERS ARE ENFORCED
  - BECAUSE STOREs ARE SENT TO THE STORE BUFFER ONLY WHEN THEY RETIRE FROM THE PROCESSOR, AND
  - PROVIDED STOREs ARE GLOBALLY PERFORMED ONE BY ONE IN ORDER FROM THE STORE BUFFER
- REMAINING ORDERS ARE LOAD-LOAD AND STORE-LOAD
  - THE SECOND ACCESS IN THESE ORDERS IS A LOAD

EXPLOIT SPECULATION

SPECULATIVE VIOLATIONS OF SEQUENTIAL CONSISTENCY

- BASIC IDEA:
  - PERFORM ALL LOADs AND USE THE VALUE RETURNED BY EACH LOAD SPECULATIVELY, AS IN UNIPROCESSOR
  - LATER, WHEN THE LOAD RETIRES AT THE TOP OF ROB, CHECK WHETHER THE VALUE OF THE LOAD HAS CHANGED
  - ROLL BACK IF IT HAS. OTHERWISE CONTINUE.

- TWO MECHANISMS ARE NEEDED: VALIDATION AND ROLLBACK
  - VALIDATION OF LOAD VALUES AT THE TOP OF ROB
    - CHECK THE VALUE OF THE LOAD BY RE-EXECUTING THE LOAD IN CACHE (Pb: TWO CACHE ACCESSES FOR EACH LOAD)
    - OR: MONITOR EVENTS THAT COULD CHANGE THE SPECULATIVE VALUE OF A LOAD BETWEEN THE TIME THE LOAD RETURNS THE VALUE SPECULATIVELY UNTIL THE LOAD COMMITS
      - “”EVENTS” INCLUDE INVALIDATES, UPDATES OR CACHE REPLACEMENTS IN THE NODE’S CACHE
        - THESE EVENTS SNOOP THE LOAD Q
  - ROB ROLLBACK USES SAME MECHANISM AS MISPREDICTED BRANCHES OR EXCEPTIONS

LOAD VALUE RECALL

SPECULATIVE VIOLATIONS OF SEQUENTIAL CONSISTENCY

- LOAD-LOAD ORDER: USE LOAD VALUE RECALL
- STORE-LOAD ORDER: USE LOAD VALUE RECALL + STALL LOADS AT TOP OF ROB UNTIL ALL PREVIOUS STOREs HAVE BEEN GLOBALLY PERFORMED
- LOAD-STORE ORDER: STOREs RETIRE AT TOP OF ROB, BY THAT TIME ALL PREVIOUS LOADS HAVE RETIRED
- STORE-STORE ORDER: STOREs ARE RETIRED AT TOP OF ROB + STOREs ARE GLOBALLY PERFORMED FROM STORE BUFFER ONE BY ONE IN T.O.
- PERFORMANCE ISSUES
  - A LOAD MAY REACH THE TOP OF THE ROB BUT CANNOT PERFORM AND IS STALLED BECAUSE OF PRIOR STORES IN THE STORE BUFFER
  - THIS BACKS UP THE ROB (AND OTHER INTERNAL BUFFERS) AND EVENTUALLY STALLS THE PROCESSOR

RELAX STORE-LOAD ORDER (TSO)

SPECULATIVE TSO VIOLATIONS

- LOAD-LOAD ORDER: USE LOAD VALUE RECALL
- STORE-LOAD ORDER: NO NEED TO ENFORCE. THUS VALUE OF LOAD SHOULD NOT BE RECALLED IF ALL PRECEDING PENDING ACCESSES IN THE LOAD/STORE Q ARE STOREs (POSSIBLE OPTIMIZATION) + LOADS DON'T WAIT ON STORES IN SB
- LOAD-STORE ORDER: STOREs RETIRE AT TOP OF ROB
- STORE-STORE ORDER: STOREs RETIRE AT TOP OF ROB + STOREs ARE GLOBALLY PERFORMED FROM STORE BUFFER IN T.O.
- PERFORMANCE ISSUES:
  - IF A LONG LATENCY STORE BACKS UP THE STORE BUFFER THEN STORES CANNOT RETIRE, WHICH MAY BACK UP THE ROB AND OTHER Q's AND STALL DISPATCH

RELAX ORDERS FURTHER WITH WEAK ORDERING OR RELEASE CONSISTENCY

SPECULATIVE EXECUTION OF RMW ACCESSES

- RMW INSTRUCTIONS ARE MADE OF A LOAD FOLLOWED BY A STORE
  - MUST EXECUTE AS A GROUP AND BE ATOMIC

- BECAUSE OF THE LOAD, THE VALUE OF THE LOCK IS RETURNED SPECULATIVELY
  - BASED ON THE SPECULATIVE LOCK VALUE THE CRITICAL SECTION IS ENTERED SPECULATIVELY OR THE LOCK IS RETRIED SPECULATIVELY
  - THIS ACTIVITY IS ALL SPECULATIVE IN THE ROB. NO MEMORY UPDATES.
  - LOADs IN RMW ACCESSES MUST REMAIN SUBJECT TO LOAD VALUE RECALL UNTIL THE RMW IS RETIRED
    - IF VIOLATION IS DETECTED LOAD IS ROLLED BACK

- BECAUSE OF THE STORE, THE RMW ACCESS DOES NOT UPDATE MEMORY (AND CANNOT PERFORM) UNTIL THE TOP OF THE ROB
  - THE LOAD AND THE STORE CAN BE PERFORMED ATOMICALLY IN CACHE WITH A WRITE-INVALIDATE PROTOCOL
  - IF THE STORE CAN EXECUTE AT THE TOP OF THE ROB, THEN NO OTHER THREAD GOT THE LOCK DURING THE TIME THE RMW WAS SPECULATIVE

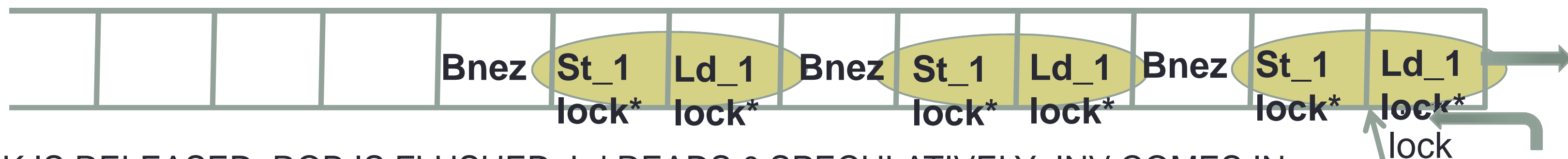
SPECULATIVE EXECUTION OF RMW ACCESSES

EXAMPLE: BUSY WAITING ON HIGHLY CONTENDED LOCK

```
Lock:    T&S R1, lock          Unlock:   SW R0, lock
         BNEZ R1, Lock                   RET
         RET
```

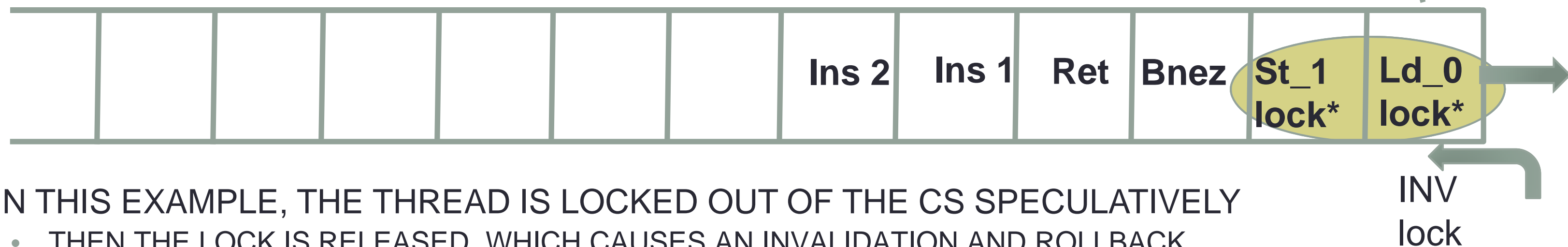
ROB

ASSUME LOCK BLOCKS Ld RETURNS 1; INVALIDATION COMES IN FOR LOCK



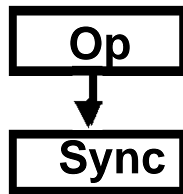
LOCK IS RELEASED; ROB IS FLUSHED; Ld READS 0 SPECULATIVELY; INV COMES IN

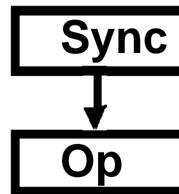
MUST BE ATOMIC

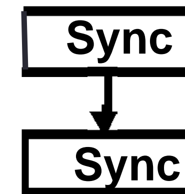


- IN THIS EXAMPLE, THE THREAD IS LOCKED OUT OF THE CS SPECULATIVELY
  - THEN THE LOCK IS RELEASED, WHICH CAUSES AN INVALIDATION AND ROLLBACK
  - THEN THE Ld IS RE-EXECUTED SPECULATIVELY AND LOCK IS FREE
  - BUT AN INVALIDATION COMES IN BEFORE THE T&S HAS BEEN ABLE TO COMMIT
  - SO THE EXECUTION MUST ROLLBACK AGAIN

SPECULATIVE VIOLATIONS OF WEAK ORDERING





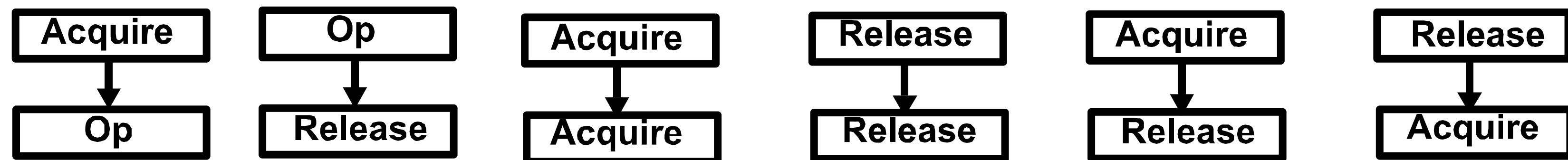


- OP-TO-SYNC:
  - ANY ACCESS TO A SYNC VARIABLE MUST BE GLOBALLY PERFORMED ATOMICALLY AT THE TOP OF THE ROB
    - PREVIOUS LOADS HAVE RETIRED
    - PREVIOUS STORES HAVE RETIRED FROM PROCESSOR
      - THE STORES IN THE SB MUST EXECUTE IN CACHE BEFORE THE SYNC CAN START ITS EXECUTION IN CACHE
- SYNC-TO-OP
  - REGULAR STORES FOLLOWING AN ACCESS TO A SYNC VARIABLE CANNOT PERFORM IN CACHE UNTIL THEY REACH THE TOP OF ROB.
  - REGULAR LOADs RETURN THEIR VALUE SPECULATIVELY AND ARE NOT SUBJECT TO LOAD VALUE RECALL
- SYNC-TO-SYNC
  - SYNC CAN ONLY EXECUTE IN CACHE AT THE TOP OF ROB
    - PRIOR SYNCs ARE RETIRED BY THEN
  - A LOAD OF A SYNC VARIABLE (INCLUDING THE LOAD IN A RMW ACCESS) CAN RETURN ITS VALUE SPECULATIVELY PROVIDED IT IS SUBJECT TO LOAD VALUE RECALL.

- EXECUTION IS NOT ROLLED BACK

SPECULATIVE VIOLATIONS OF RELEASE CONSISTENCY

- IN RC, A RELEASE MAY NOT PERFORM UNTIL ALL PREVIOUS ACCESSES ARE GLOBALLY PERFORMED AND ACCESSES FOLLOWING AN ACQUIRE MUST BE DELAYED UNTIL THE ACQUIRE IS GLOBALLY PERFORMED.

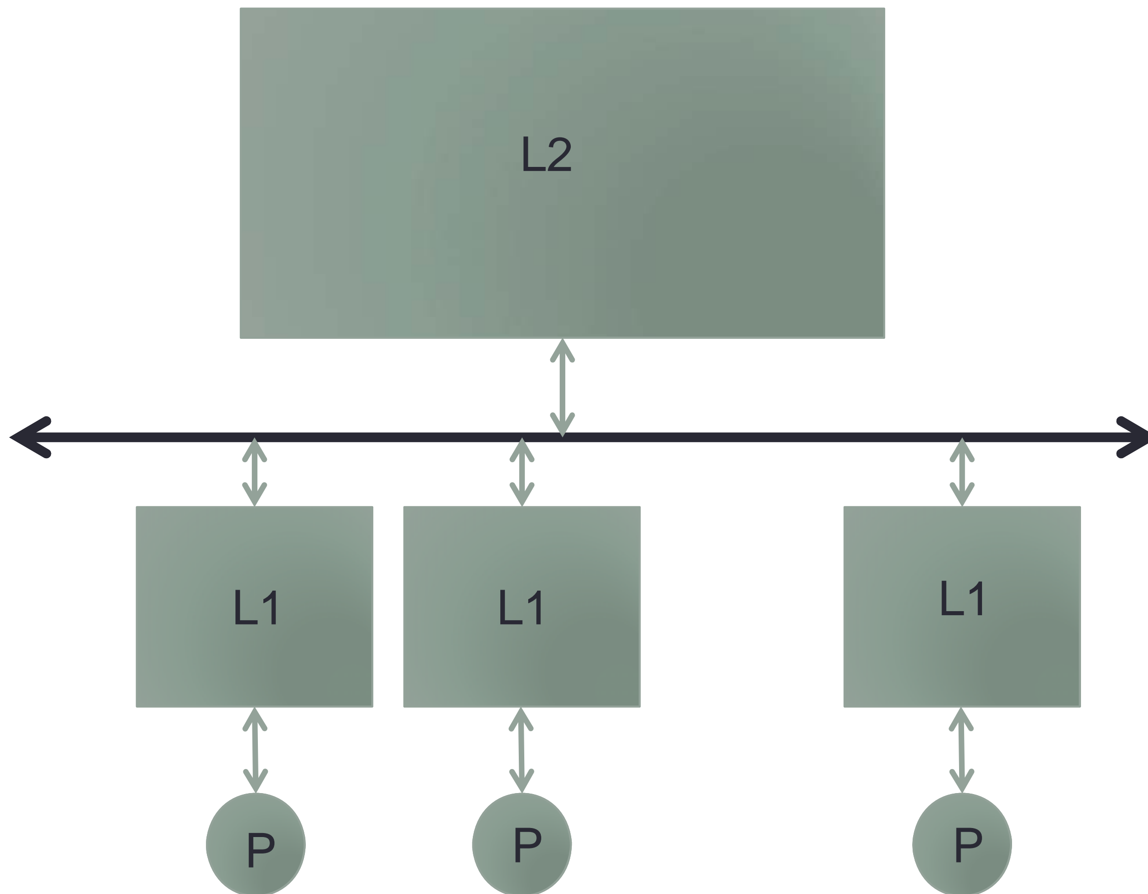


- Acquire-Op: RC REQUIRES THAT ACQUIRES MUST BE PERFORMED BEFORE ANY FOLLOWING REGULAR ACCESS CAN PERFORM.
  - HOWEVER THE VALUE OF REGULAR LOADS MAY BE RETURNED AND USED **SPECULATIVELY**.
    - REGULAR LOADS ARE NOT SUBJECT TO VALUE RECALL,
- Op-Release: RELEASES ARE PUT IN THE STORE BUFFER AS “SPECIAL” STORES
  - BY THAT TIME ALL PREVIOUS LOADS HAVE BEEN EXECUTED
  - RELEASE MUST WAIT IN SB UNTIL ALL PREVIOUS STORES HAVE BEEN PERFORMED
- Acquire-Acquire & Acquire-Release: ACQUIRES EXECUTE AT THE TOP OF ROB, BEFORE ANY ACQUIRE OR RELEASE HAS A CHANCE TO EXECUTE
  - SYNC LOADS IN ACQUIRES CAN RETURN THEIR VALUE SPECULATIVELY BUT ARE SUBJECT TO VALUE RECALL
- Release-Release: RELEASES JOIN THE SB ONE BY ONE IN THREAD ORDER
  - RELEASES MUST WAIT IN SB UNTIL ALL PREVIOUS RELEASES HAVE EXECUTED IN CACHE
  - STORES IN SB CAN EXECUTE IN CACHE IN ANY ORDER, EVEN BYPASSING PREVIOUS RELEASES
- Release-Acquire: ACQUIRES MUST WAIT UNTIL ALL PRIOR RELEASES IN SB ARE PERFORMED IN CACHE

STORE BUFFER

8. ADVANCED TOPICS

DO WE NEED COHERENCE UNDER WEAK ORDERING?



Processors execute from their L1 cache
Before a processor can execute a SYNC, it flushes its L1 cache, writing back all dirty block and invalidating
Then it executes the SYNC in shared L2

It is coherent if the program is correct
It is not if the program is incorrect

SPECULATIVE RETIREMENT OF NON-STORE INSTRUCTIONS

HOW TO MAKE SEQUENTIAL CONSISTENCY COMPETITIVE WITH RELAXED MODELS

- IN SPECULATIVE SC, A LOAD AT THE TOP OF THE ROB MUST WAIT UNTIL ALL PRECEDING STORES IN THE STORE BUFFER ARE GLOBALLY PERFORMED BEFORE RETIRING
- NEW (SCARY!!) IDEA: SPECULATIVE RETIREMENT OF NON-STORE INSTRUCTIONS
  - SPECULATIVELY RETIRE A LOAD AND ALL DEPENDENT NON-STORE INSTRUCTIONS
  - THE LIKELIHOOD THAT THE LOAD WILL BE RECALLED IS VERY LOW

THIS DOES NOT HELP TSO, BUT IT HELPS SC

SPECULATIVE RETIREMENT OF NON-STORE INSTRUCTIONS

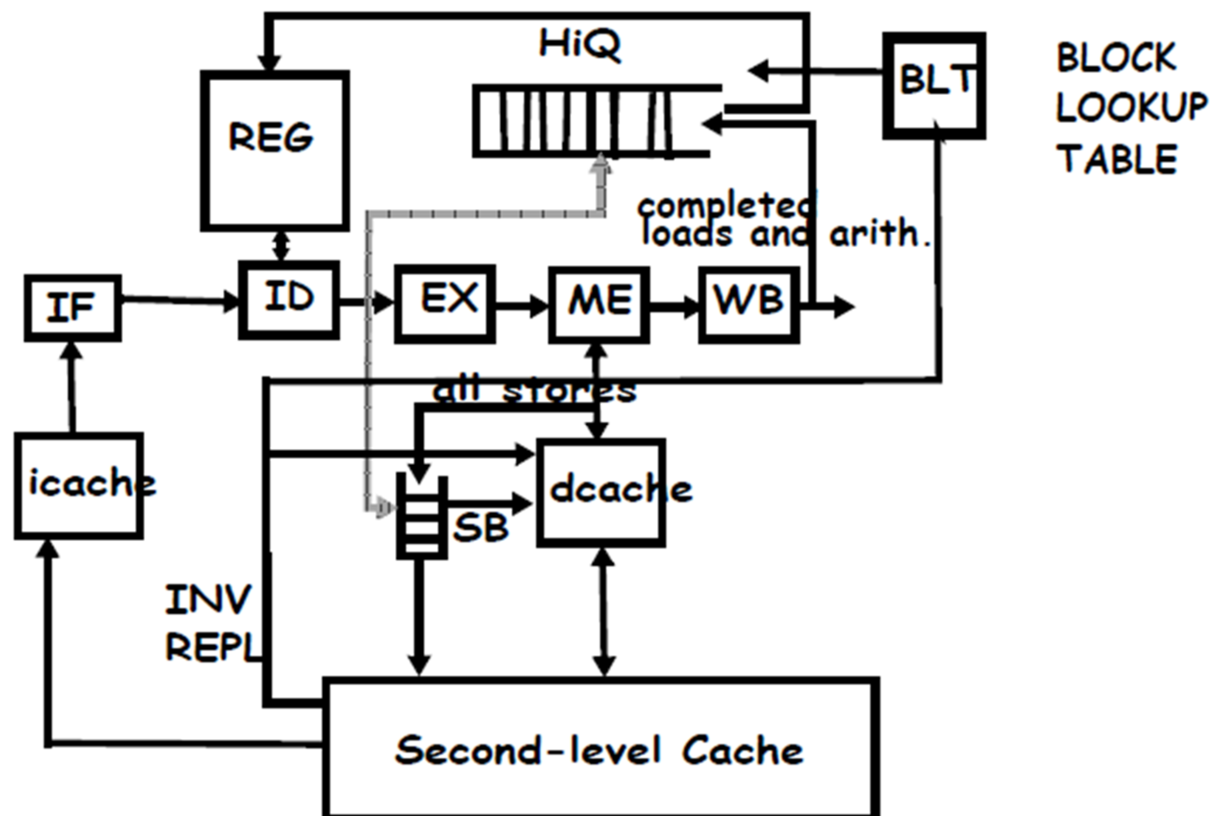
HOW TO MAKE SEQUENTIAL CONSISTENCY COMPETITIVE WITH RELAXED MODELS

WHEN A LOAD INSTRUCTION REACHES THE TOP OF THE ROB, IS COMPLETE, AND IS STALLED BECAUSE OF PRIOR STORES AND THE PROCESSOR IS BACKED UP BECAUSE OF THAT:

- THE LOAD IS REMOVED FROM THE ROB AND SPECULATIVELY RETIRES—MEANING IT MODIFIES THE ARCHITECTURAL REGISTER AND FREES QUEUES
- A RECORD FOR THE LOAD IS INSERTED IN A FIFO HISTORY BUFFER
- FOLLOWING INSTRUCTIONS ARE ALSO RETIRED FROM THE ROB SPECULATIVELY AND RECORDS ARE ALSO INSERTED
- STORE INSTRUCTIONS ARE PUT IN THE STORE BUFFER
- HISTORY BUFFER ENTRY CONTAINS THE PC, THE OLD REGISTER VALUE AND THE OLD MAPPING FOR RECOVERY
- STORES AND BRANCHES ARE NOT INSERTED IN THE HISTORY BUFFER SINCE THEY DO NOT UPDATE ARCHITECTURAL REGISTERS
- HISTORY BUFFER ENTRIES ARE RECLAIMED AS SOON AS ALL STORES PRIOR TO THEM HAVE RETIRED FROM THE STORE BUFFER

SPECULATIVE RETIREMENT OF NON-STORE INSTRUCTIONS

HOW TO MAKE SEQUENTIAL CONSISTENCY COMPETITIVE WITH RELAXED MODELS



SPECULATIVE RETIREMENT OF NON-STORE INSTRUCTIONS

- VIOLATION DETECTION
  - INVALIDATION OR REPLACEMENT OF BLOCK WITH SPECULATIVE LOAD
- RECOVERY
  - IF LOAD IS STILL IN THE ROB, PROCEED AS BEFORE FOR SPECULATIVE LOADS
  - IF LOAD IS SPECULATIVELY RETIRED IN THE HISTORY BUFFER, STATE MUST BE RESTORED BY PROCESSING HISTORY BUFFER ENTRIES IN REVERSE ORDER
  - REMOVE STORES FOLLOWING THE VIOLATING LOAD FROM THE STORE BUFFER

SPECULATIVE RETIREMENT OF LOADS CLOSES THE GAP BETWEEN SC AND TSO BECAUSE IT ENABLES THE USE OF STORE BUFFERS IN SC

AN ALTERNATIVE IS TO USE A LARGER WINDOW AND SUFFER THE CONSEQUENCES (LONGER CYCLE TIME)

SPECULATIVE RETIREMENT OF ALL INSTRUCTIONS

- A RECORD IS LOGGED IN THE HiQ FOR ALL SPECULATIVELY RETIRED INSTRUCTIONS MODIFYING MACHINE STATE
- RECORD FOR STORES MUST INCLUDE THE PREVIOUS MEMORY VALUE SO THAT IT CAN BE RESTORED
  - THUS SPECULATIVELY RETIRED STORE MUST EXECUTE AS RMW
- AS PENDING STORES IN SB PERFORM, THE ENTRIES AT THE TOP OF HiQ ARE DEALLOCATED UP UNTIL THE NEXT PENDING STORE IN SB
- VIOLATION DETECTION
  - EVERY INVALIDATION/UPDATE, EVERY REMOTE LOAD AND EVERY NODE REPLACEMENT OF A SPECULATIVELY LOADED OR STORED CACHE BLOCK
- VIOLATION RECOVERY
  - ROLLBACK TO THE FIRST INSTRUCTION IN THREAD ORDER ACCESSING THE FAULTING BLOCK
  - ALL THE MEMORY LOCATIONS ACCESSED IN THE ROLLBACK ARE GUARANTEED TO BE IN THE NODE’S CACHE
  - UPON ENDING ROLLBACK FURTHER SPECULATIVE EXECUTION IS INHIBITED WHILE WAITING FOR THE PENDING STORES TO FINISH