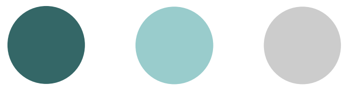
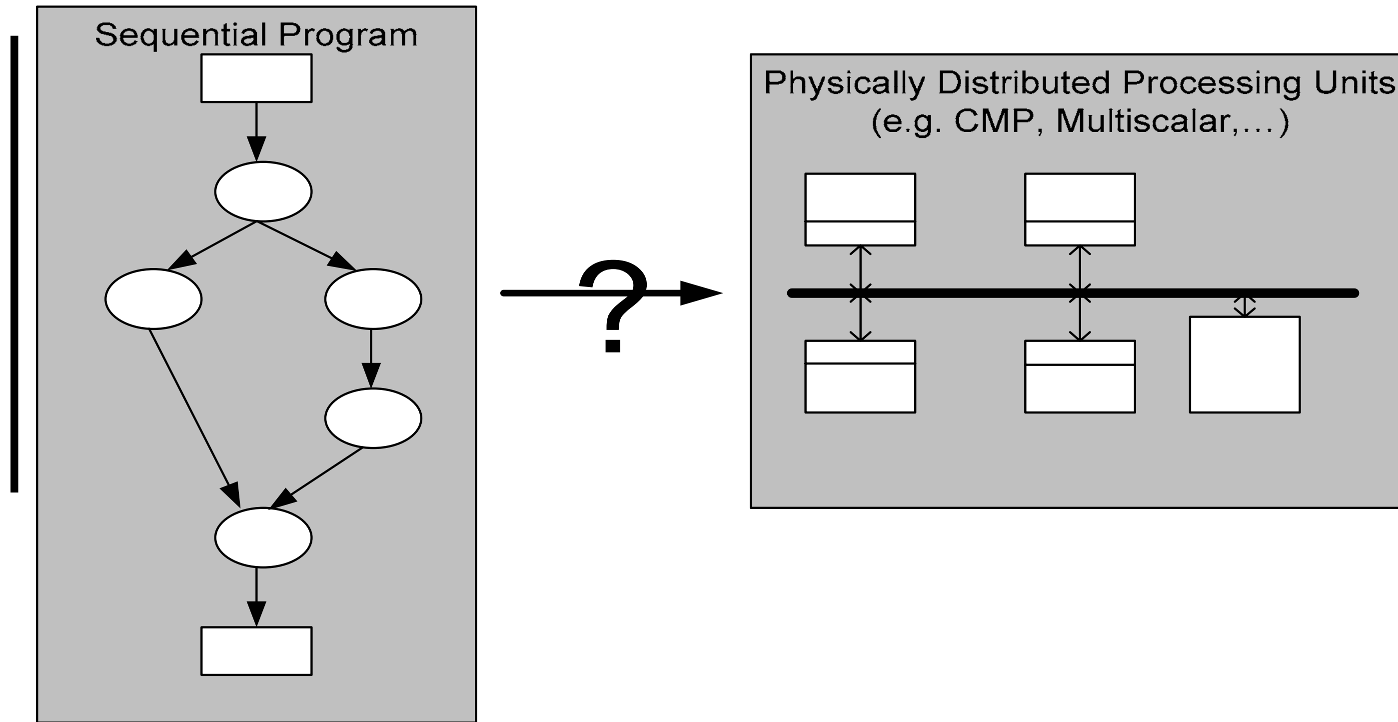




Xiaobin Li

xiaobinL@uci.edu

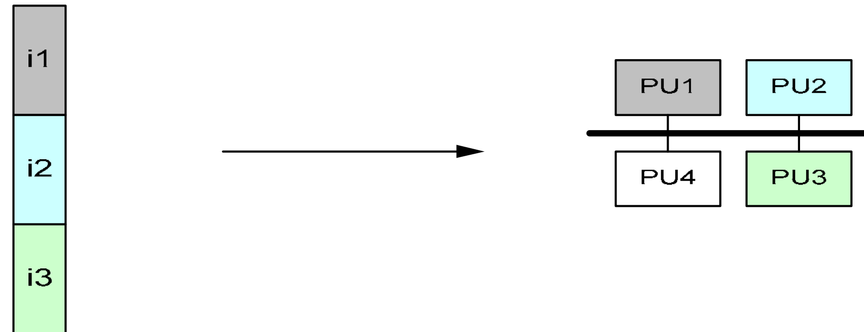
Sequential Program Semantics

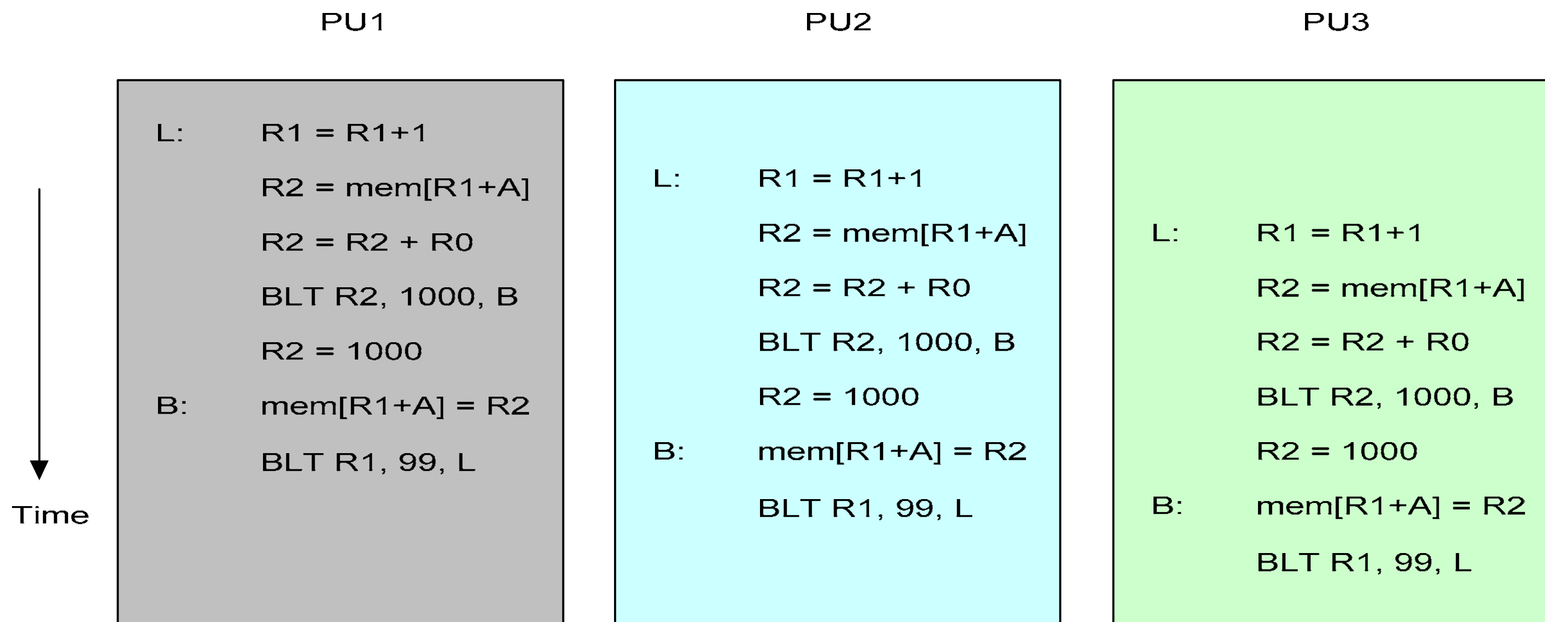
```
… …
for (i=0;i<100;i++)
{
      A[i] += 10;
      if(A[i] > 1000)
            A[i] = 1000;
}
……
```

↓

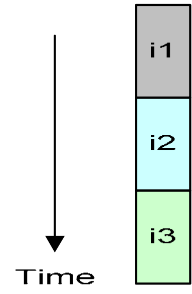
```
      R1 = -1                 ;initial induction variable i
      R0 = b                  ;assign loop-invariable b(=10) to R0
L:    R1 = R1 +1              ;increment i
      R2 = mem[R1+A]          ;load A[i]
      R2 = R2 + R0            ;add b
      BLT   R2, 1000, B       ;branch to B if A[i] < 1000
      R2 = 1000               ;set A[i] to 1000
B:    mem[R1+A] = R2          ;store A[i]
      BLT   R1, 99,L          ;branch to L if i<99
```

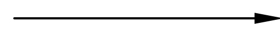
Speculative Multithreading

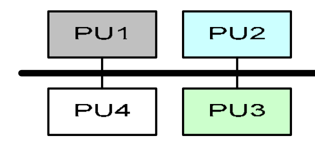




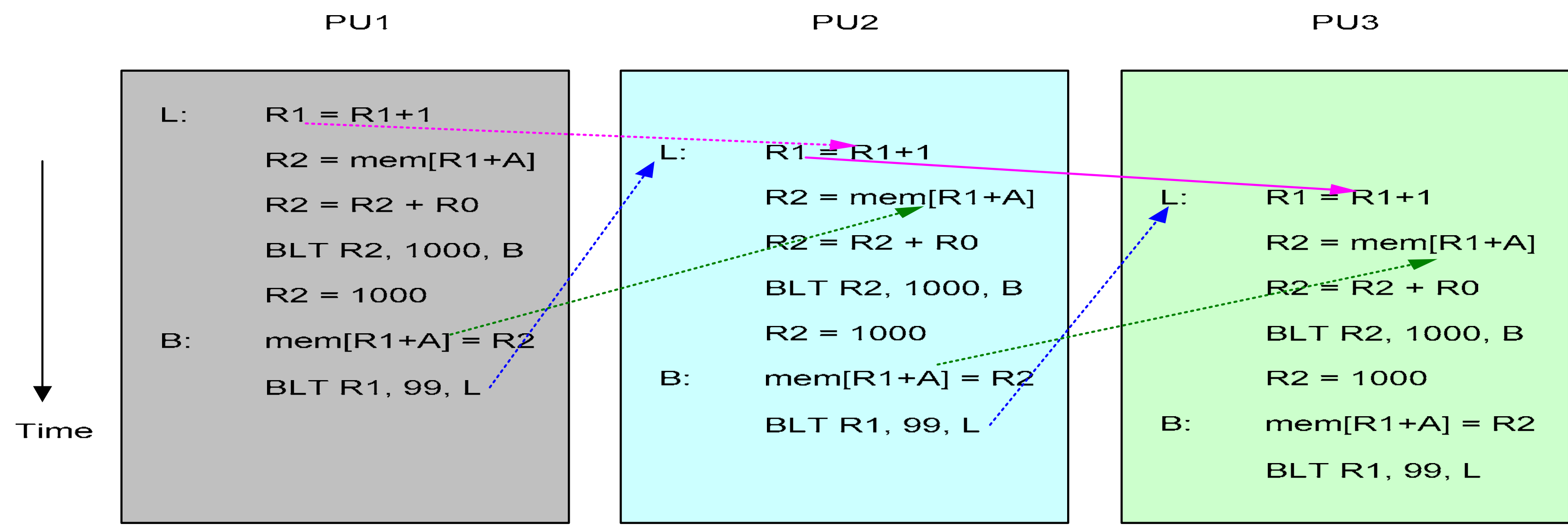
Inter -Thread Dependencies







PU1 PU2 PU3



Non-speculative thread

Speculative thread 1

Speculative thread 2

Inter-Thread Control Dependency

Inter-Thread Register Dependency

Inter -Thread Potential Memory Dependency

Proving Sequential Program Semantics

- Inter-thread dependencies
  - Control dependency
  - Register-level data dependency
  - Memory-level data dependency

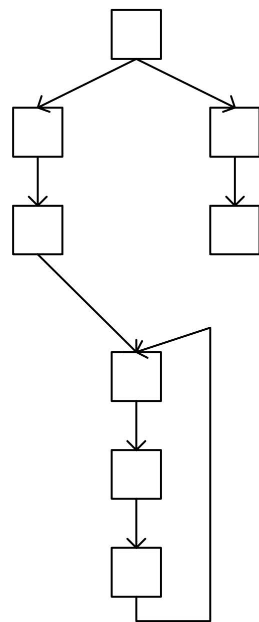
- "Latest data version" requirement

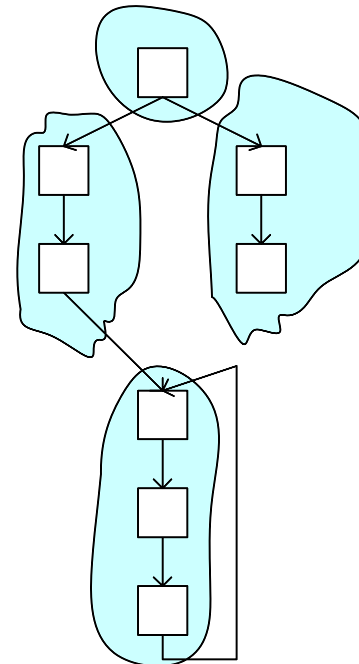
About the Presentation

- Problem statement
- Handling threads
- Register-Level
- Memory-Level
- Evaluation
- Conclusion

Thread Identifying

Forming from a control-flow-graph (CFG)





Loop Iteration Threads

- “Currently, we limit the threads to loop iterations.”
- “At present, our current approach of analyzing sequential binaries is restricted to loop iterations.”

► Post-compiler step: “ … …, we mark the entry and exit points of each loop.”

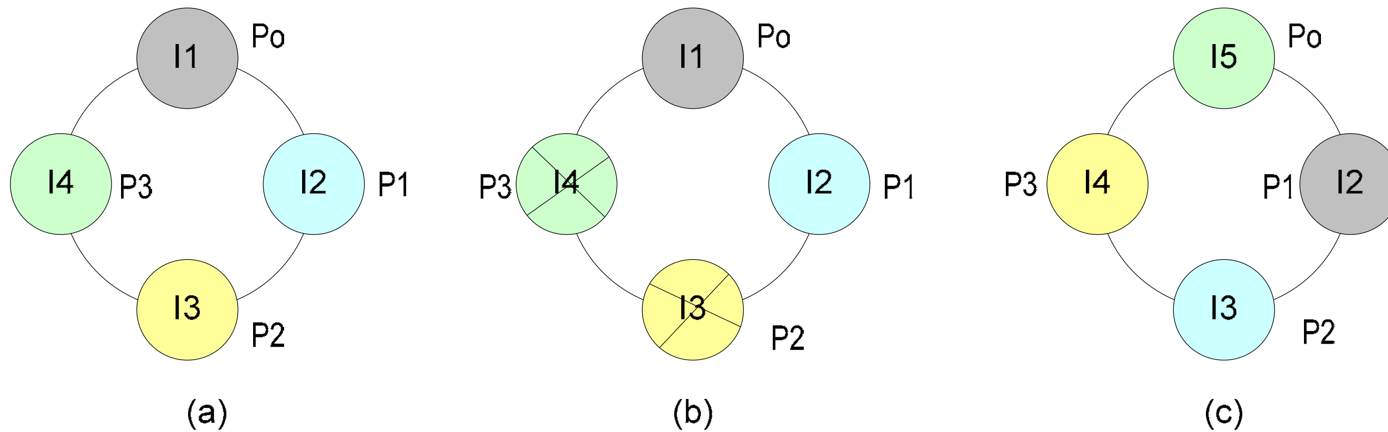
► At runtime: “… …, when a loop entry point is reached, multiple threads are spawned to begin execution of successive iterations speculatively.”

☺ Already found one example from previous discussion

Analysis Assumptions

- 4-processor CMP
- Each processor guarantees itself sequential program semantics → thread-level sequential semantics
- 4-processor configuration as circular queue (explicitly at Multiscalar)
- Sequential thread termination: until thread becomes non-speculative (for example, its predecessor resolved conditional branch so its execution approved and got non-speculative state), it can commit and new thread can initiate on same processor.

Thread Commits and Squashes



(a) Thread assigns (b)Thread squashes (c) Thread commits

About the Presentation

- Problem Statement
- Handling threads
- Register-Level
- Memory-Level
- Evaluation
- Conclusion

Register - Level

- SW+HW
- Requirements:
  - Communication mechanism
  - Data dependency
  - Latest data version
  - Last copy problem
  - Complexity

Register-Level SW Support (1)

- Looplive register identifying
  → Specify which registers will be communicated between loop iteration threads, (loop-carried) and, for a thread “Me”, am I a producer? A consumer?
  → Live at loop entry/exits and may also be redefined in the loop

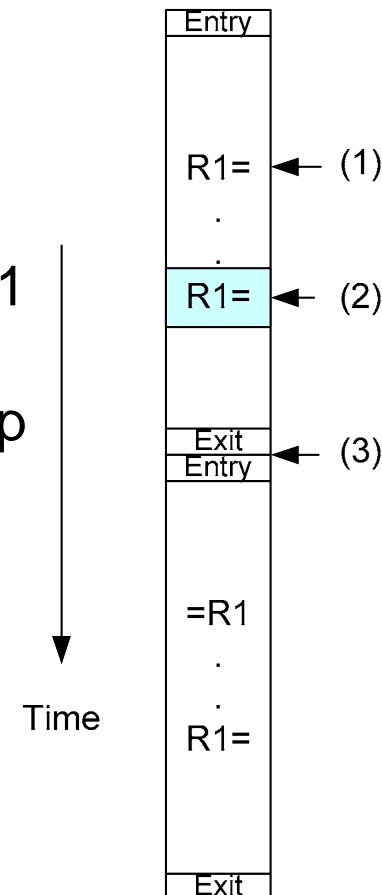
- Early validation of prediction

Register-Level SW Support (2)

Safe definitions + release points

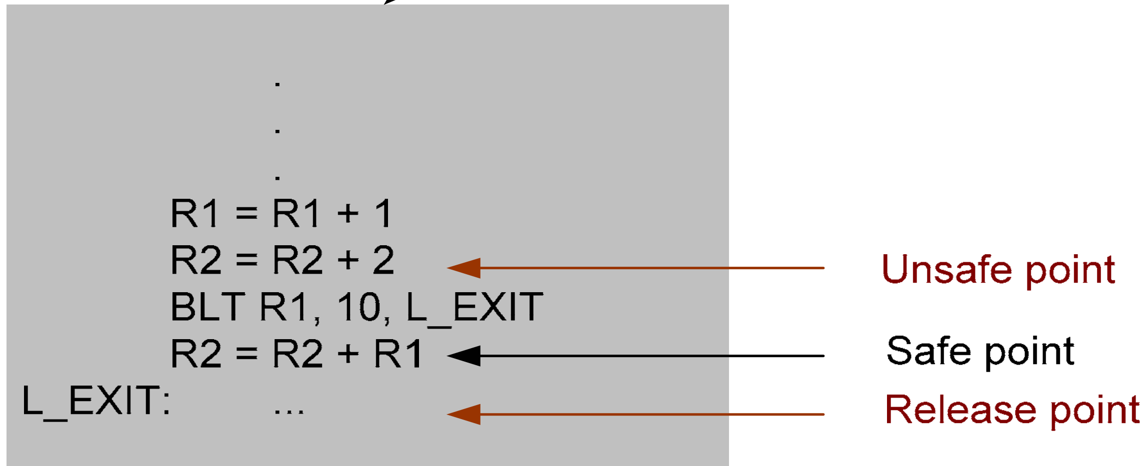
→ Guarantee “latest data version”

EX. For a given looplive register, say, R1 as shown at the figure, SW tells the time point “no more update it until loop exit”, which is time point (2) provided between (2) and (3) no update R1 instruction exist.



Send R1 after point (2), so that its successors get “latest version of R1”.

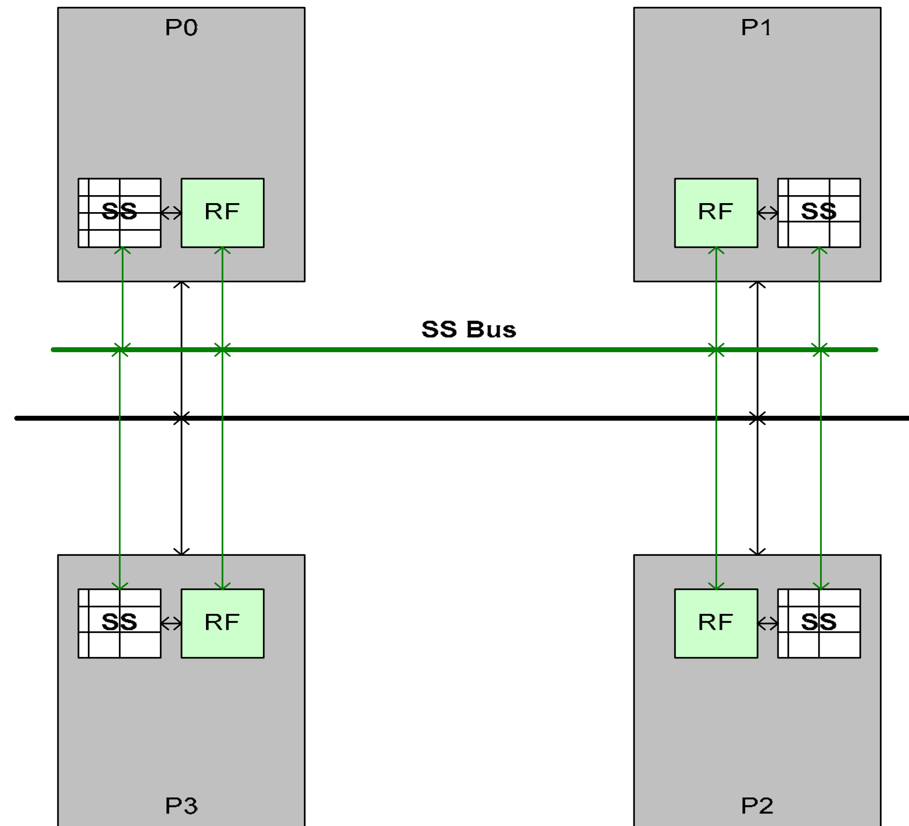
Register-Level SW Support (3)

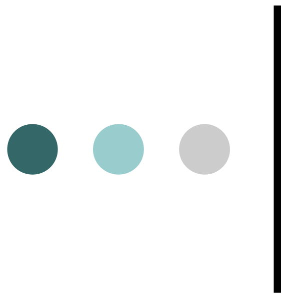


Register-Level HW Support

1. Road – Synchronizing scoreboard bus (SS Bus)
2. Signal - Synchronizing scoreboard (SS)
3. Roadmap – Register-level Read/Write protocol

Synchronizing Scoreboard





Entry of SS

Each register has an entry

- S-bit (global):

Si = 1 →the thread running on Pi, will write this register, but not ready

Si = 0 → data available

- F-bit (global):

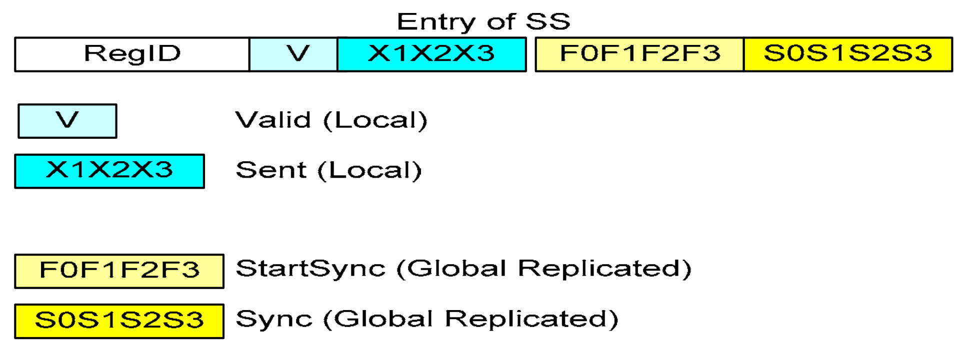
Fi = 1 →the thread running on Pi is a producer

Fi = 0 → not a producer

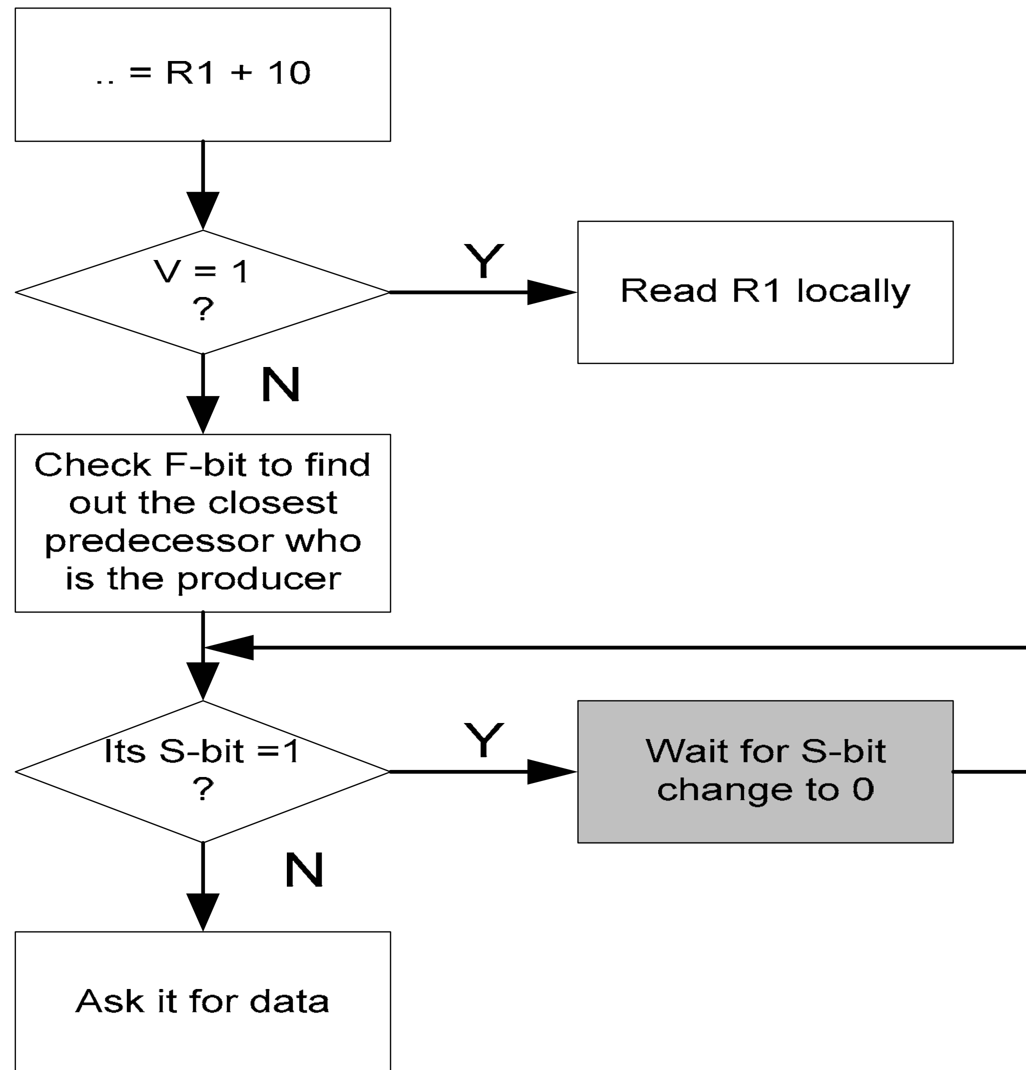
- V-bit (local):

V = 1 →data available

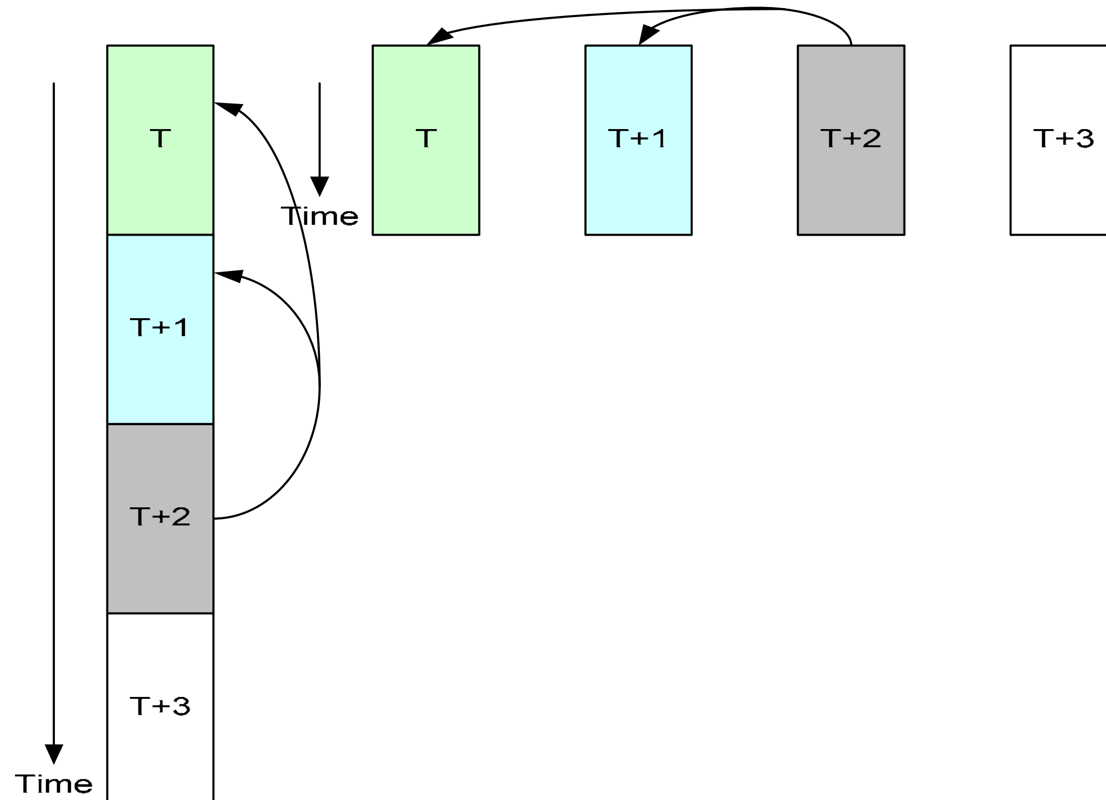
V = 0 →data not yet available



Consumer-Initiated Approach - Read



Why the Closest Predecessor?



How to Find Out the Closest Predecessor ?

"Me" Thread on P2 (ThreadMask = 1111)

→ It's speculative thread #3

→ Its predecessors are speculative thread #2, speculative thread #1 and non-speculative thread.

→ The locations of those can be found out from ThreadMask associated with processors.

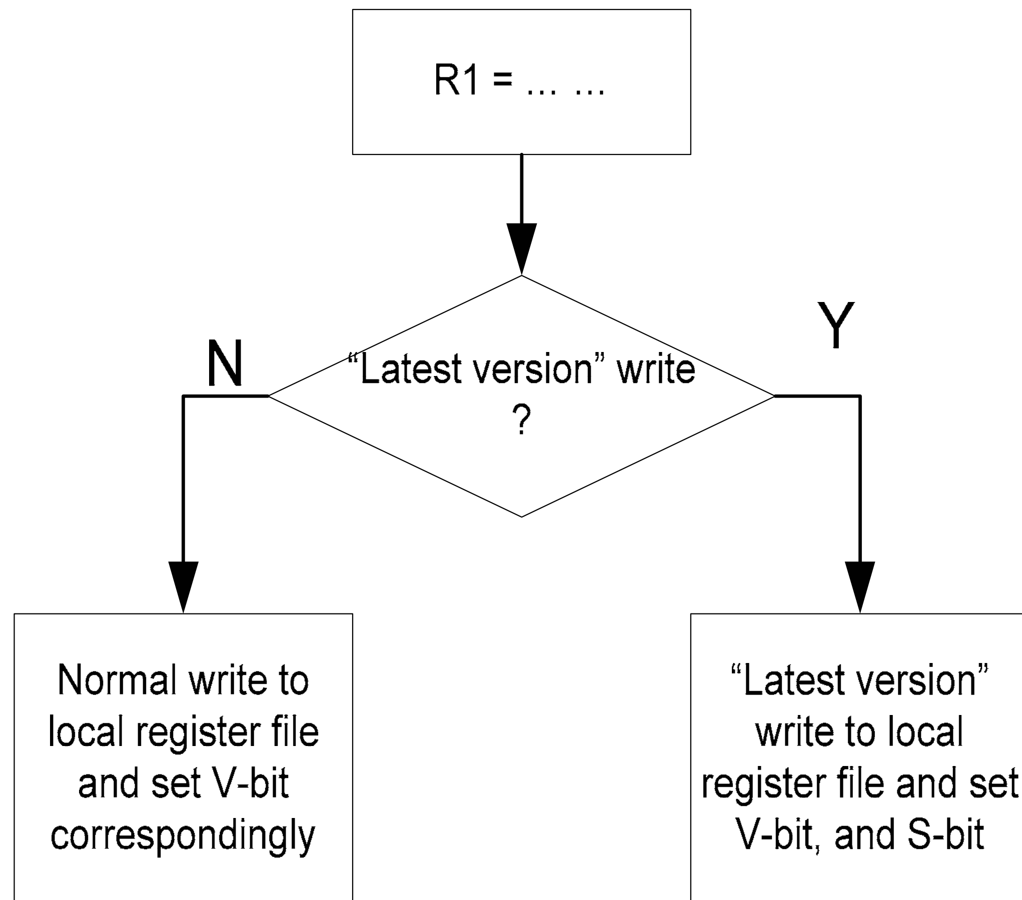
→ Read register R1 from P3

Entry of SS

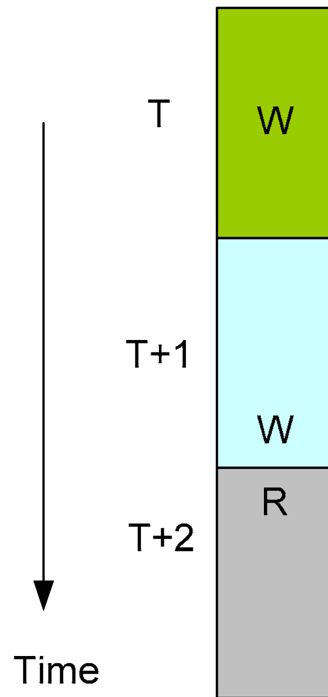
	V	X1X2X3	F0 F1 F2 F3	S0 S1 S2 S3
R1	0		0 0 0 1	0 0 0 1

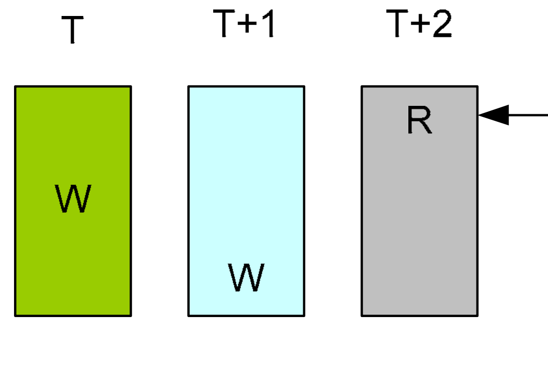
P#	ThreadMask	Thread Status	
P0	0011	Specu1	
P1	0111	Specu2	
P2	1111	Specu3	"Me" Thread
P3	0001	Non-Specu	

Consumer-Initiated Approach - Write



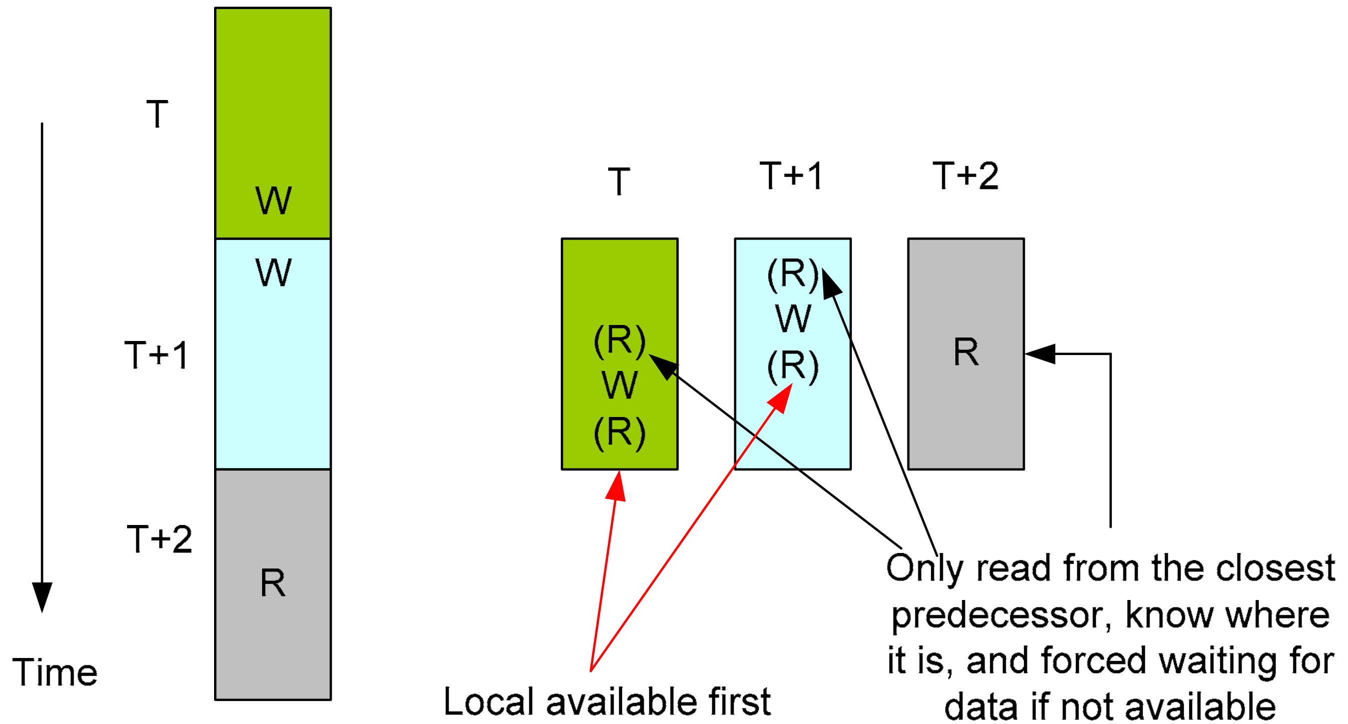
RAW Proving



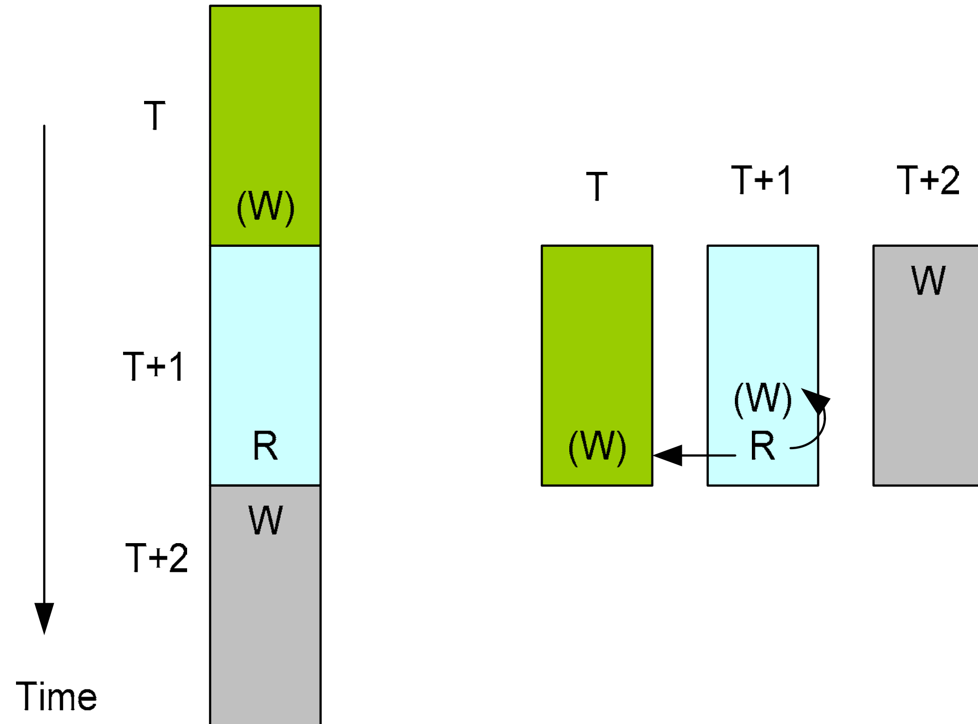


Only read from the closest predecessor, know where it is, and forced waiting for data if not available

WAW Proving



WAR Proving



The Last Copy Problem

Non-Specu | Specu#1 | Specu#2 | Specu#3
=R3 R3= (T0) | =R3 R3= (T1) | =R3 R3= (T2) | =R3 R3= (T3)

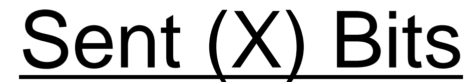
T0, T1, T2 all committed

Non-Specu: =R3 R3= (T3)

New threads T4, T5 T6 come
But T3 is about to commit

Specu#1 | Specu#2 | Specu#3 | Non-Specu
=R3 R3= (T4) | =R3 R3= (T5) | =R3 R3= (T6) | =R3 R3= (T3)

=R3 R3= (T7)



For each looplive register:

i = processor ID

Xi = 0 → when thread initials and hold until updated if any

Xi = 1 → sent out to Pi in the past or Pi asked for data before

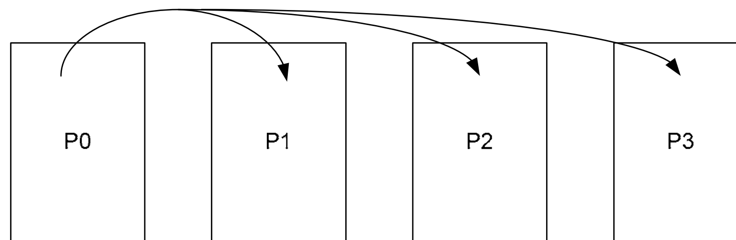
When a thread ready to commit, checks X-bit.

EX. P0 checks, whose thread is non-speculative one:

```
if (X1(F1 + X2(F2 + X3)) == 0)
{
        re-sent data || wait for retrieving and then commit;

}
```

$X1(F1+X2(F2+X3)) == 0$



- For P1, X1 = 0, not OK!
- For P2, F1=0 means P1 is not a producer, so if X2 = 0 at same time, not OK!
- For P3, F1 = 0 and F2=0 means P1 and P2 are not producers, so if X3 = 0 at same time, not OK!

Speculative Issues at Register -Level

- Only read latest version register
- No speculative read within a thread
- Thread is speculative and incorrect thread prediction

- Once non-speculative thread committed, new thread may be initialed on the same processor. Since dealing with loop iteration thread, some register value can be reused, for performance.

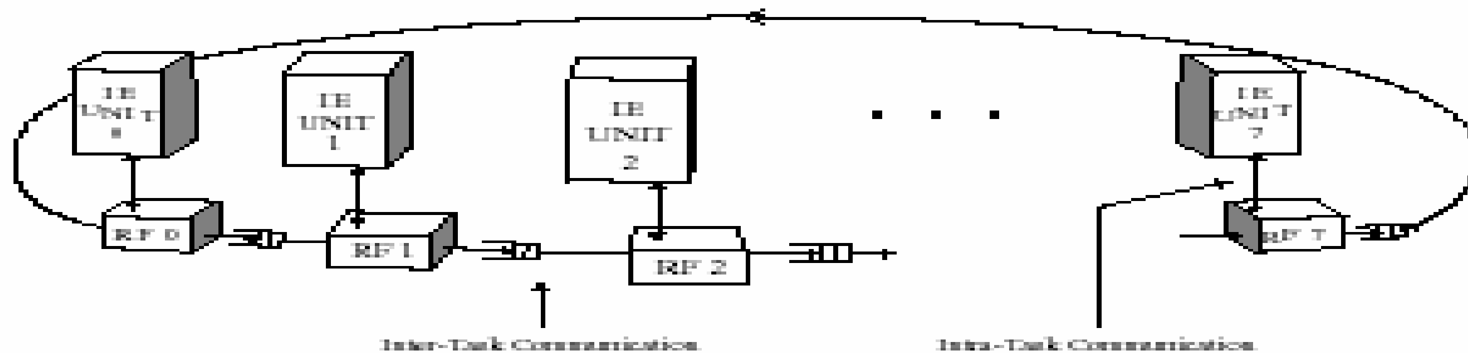
EX.

```
    R1 = -1              ;initial induction variable i
    R0 = b               ;assign loop-invariable b to R0
L:  R1 = R1 +1           ;increment i
    R2 = mem[R1+A]       ;load A[i]
    R2 = R2 + R0         ;add b
    BLT  R2, 1000, B     ;branch to B if A[i] < 1000
    R2 = 1000            ;set A[i] to 1000
B:  mem[R1+A] = R2       ;store A[i]
    BLT  R1, 99,L        ;branch to L if i<99
```

Need Crossbar-like Connection between Register Files?

- Broadcast-like bus → Bus contention
- Register traffic analysis from the paper:

  M. Franklin and G. S. Sohi:," Register traffic analysis for streamlining inter-operation communication in fine-grain parallel processors", MICRO-25, 1992



About the Presentation

- Problem Statement
- Handling threads
- Register-Level
- Memory-Level
- Evaluation
- Conclusion

Memory-Level

- HW
- Memory disambiguation
- Load/Store operations
- “Latest version” vs. “many-write”
- RAW/WAR/WAW proving
- Speculative issues

Potential Memory Dependency

- Without address calculation, we don’t know if two memory accesses go to same location. So don’t know if any memory-level data dependency.
- Memory disambiguation: “the process of determining if two memory referencing instructions access the same memory location.”

Flush(F)	Invalid(I)	Dirty(D)	SafeWrite(SW)	SafeRead(SR)	Forward(FD)

- For each cached word **(not cache line!)**
- SW-bit:

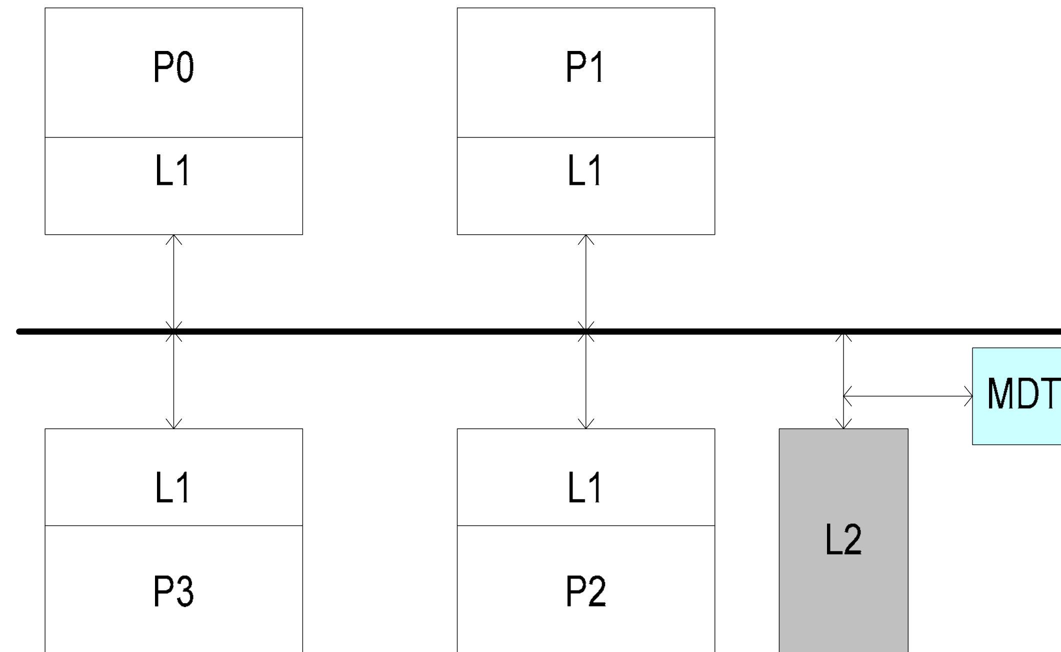
  SW = 1 → OK write a word without informing MDT

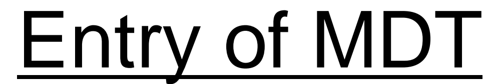
  SW = 0 → Write need inform MDT
- SR-bit:

  SR = 1 → OK read a word without informing MDT

  SR = 0 → Read need inform MDT

Memory Disambiguation Table (MDT)





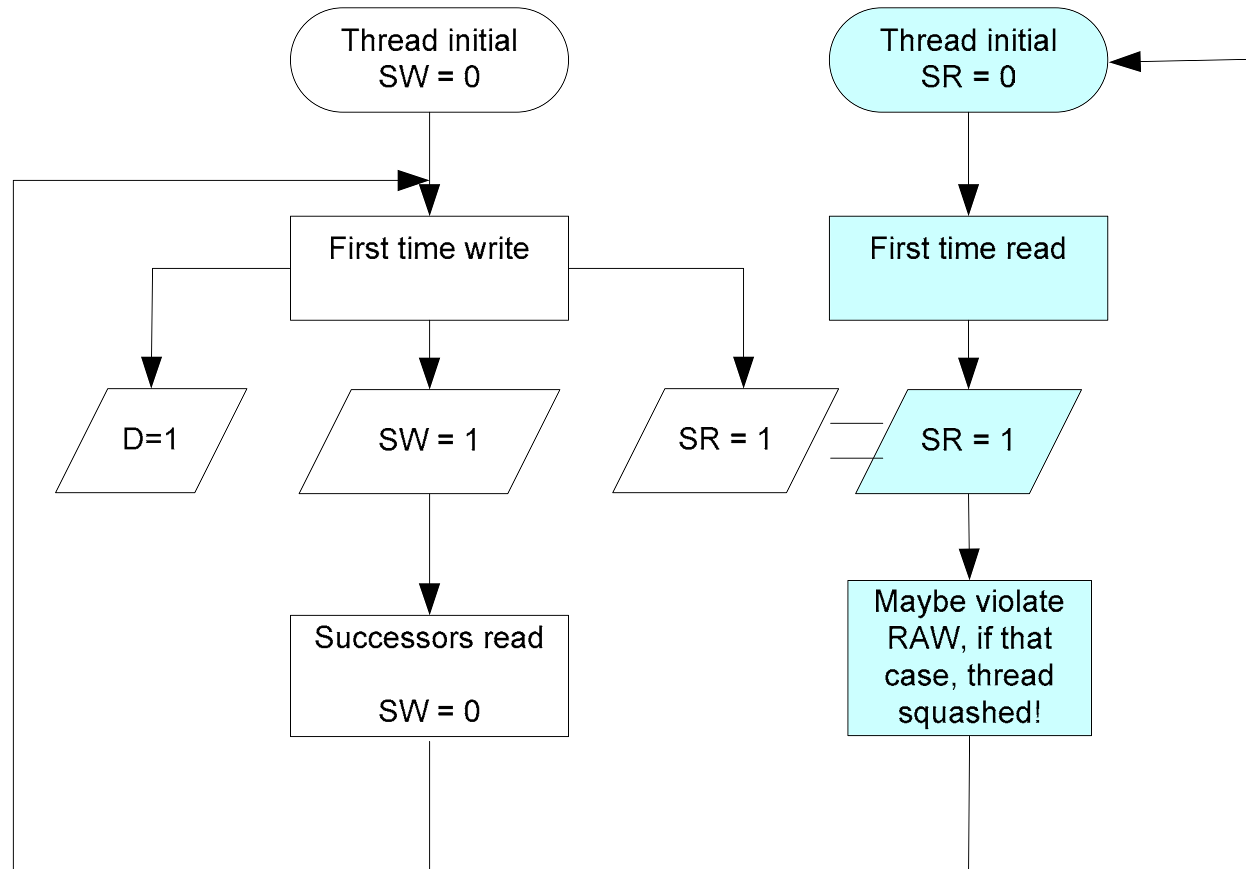
V	Address Tag	Load-Bit	Store-Bit
		L0L1L2L3	S0S1S2S3

i = processor ID

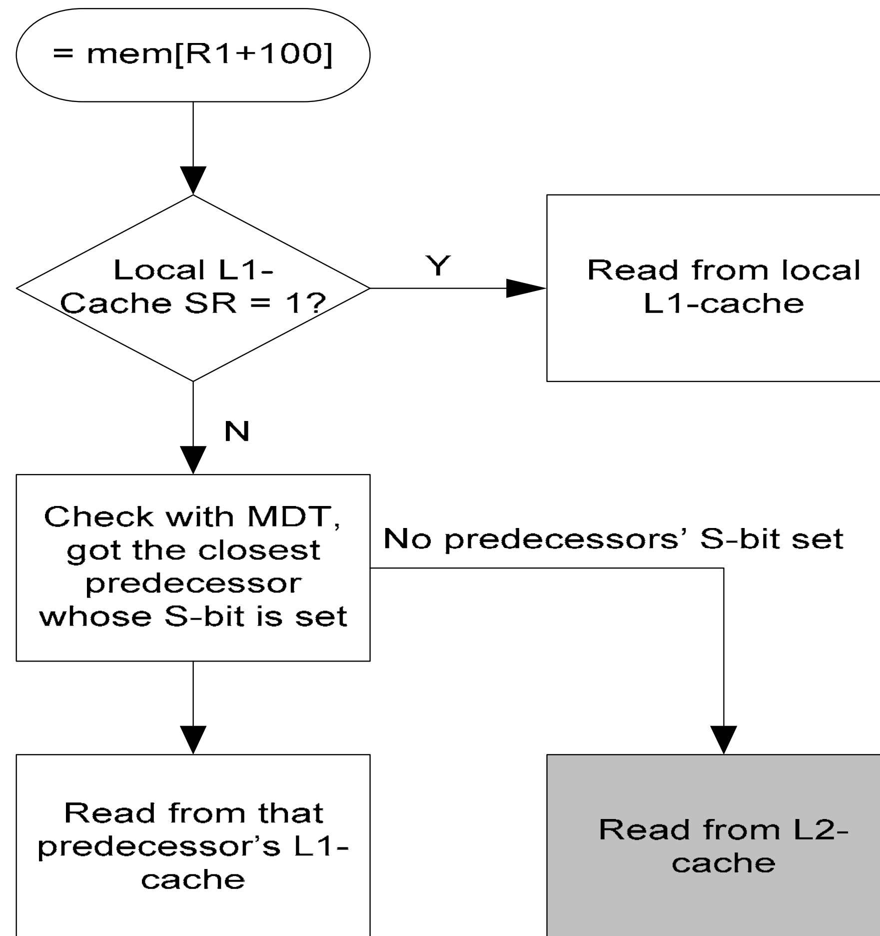
Li = 1 → thread on Pi read this word before

Si = 1 → thread on Pi write this word before

How SW-bit and SR-bit modified?



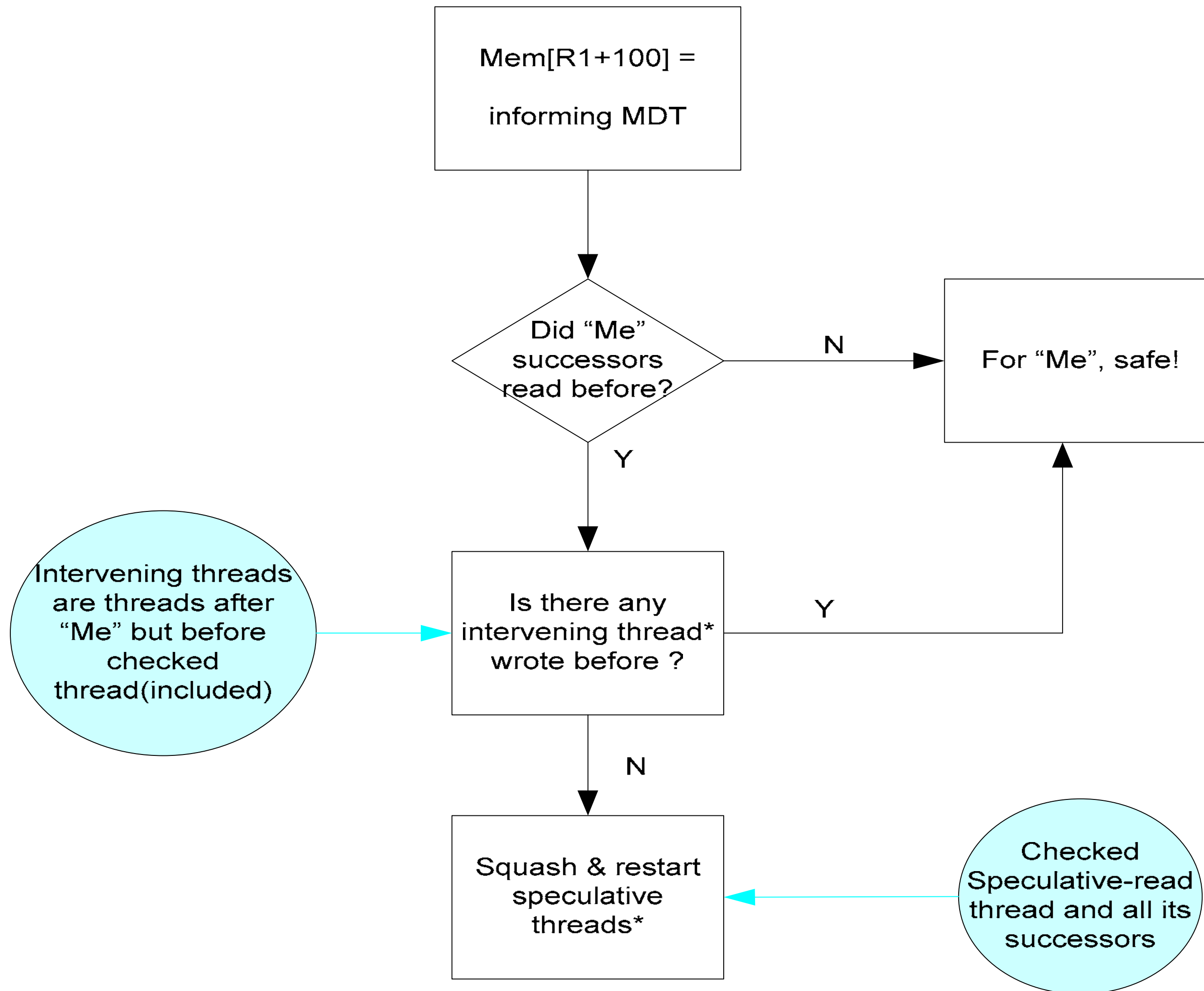
Memory Read Operation



Memory Write Operation

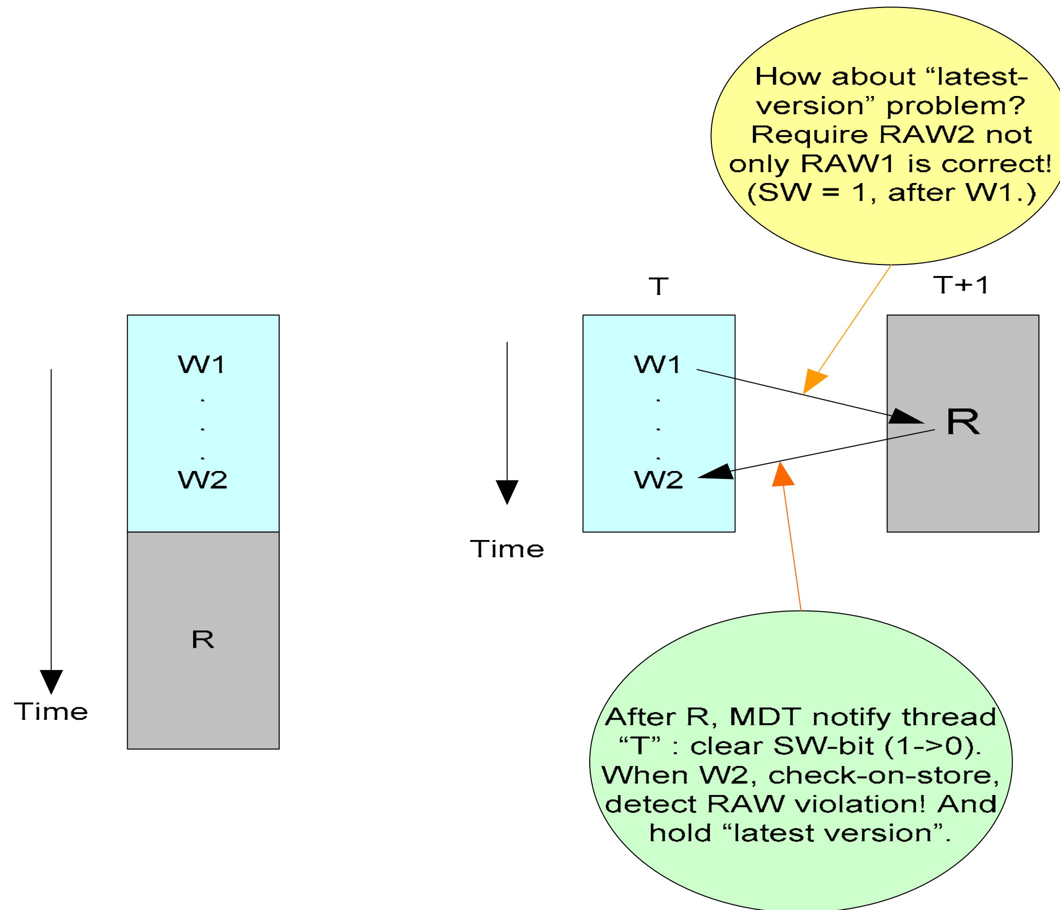
- Non-speculative thread
  - → L1-cache write-through
    - → all operation check with MDT
- Speculative threads
  - → L1-cache write-back
    - → (miss in L1-cache) or (hit but SW = 0)
      - → check with MDT

Unidirectional check-on-store

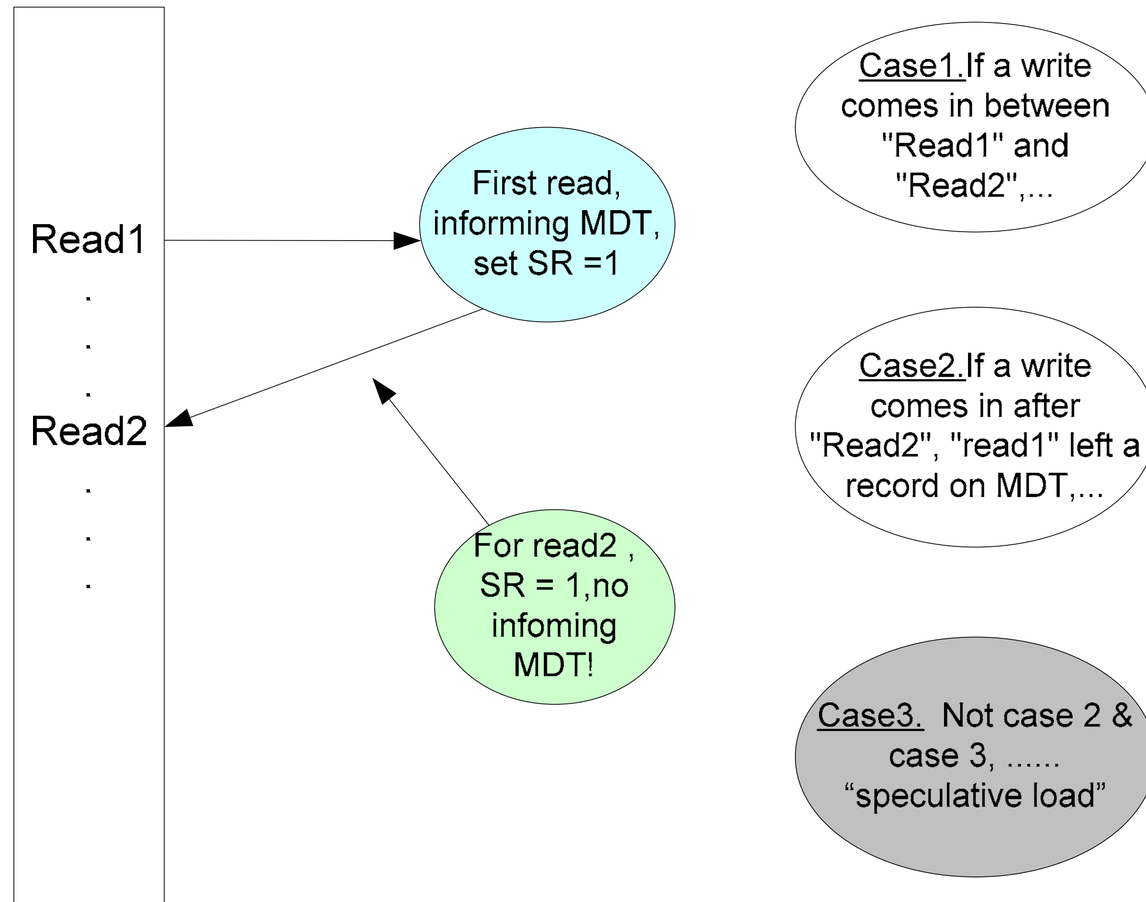


Is SafeWrite-bit safe?

"Many-write" and "latest version"



Is SafeRead-bit Safe (1) ?



Is SafeRead-bit Safe (2)?

- To exploit spatial locality, load a cache line while loading, so have the case “hits a word with SR = 0”, recall SR-bit is word-based.
- When write, “invalidation message is sent to all successors up to, but not including, the one whose S bit is set.”

  → “hits a word with SR = 0” becomes “misses a word with SR = 0”.

RAW Proving

T0 T1 T2 T3

W

R

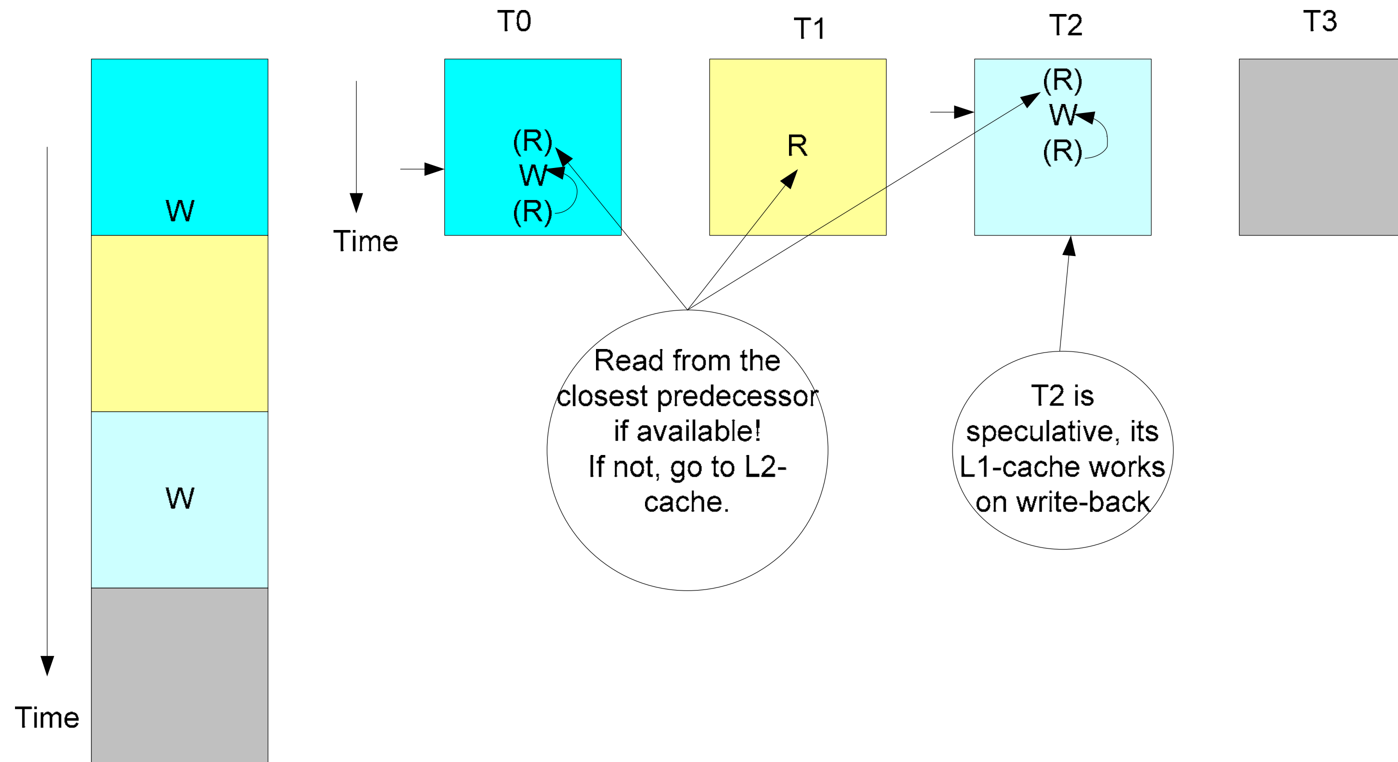
Time

At time point of R, fine!

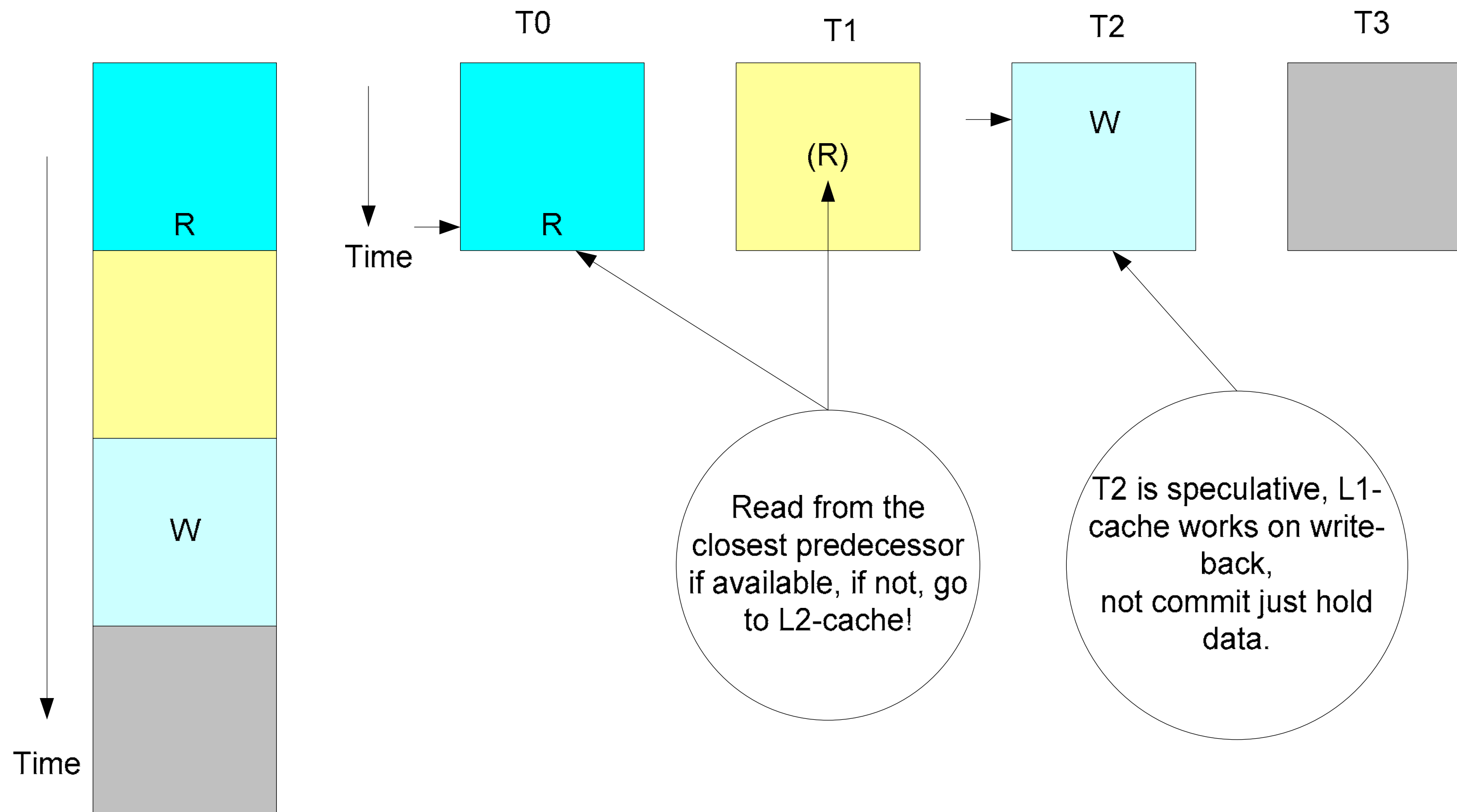
No Write between.

At time point of W, check-on-store detect T2's R violated RAW, so T2 and T3 squash.

WAW Proving



WAR Proving



- Thread Initial

  All words whose F-bit is set are invalidated;

  then F, SW, SR bits are cleared.

- Thread Squash

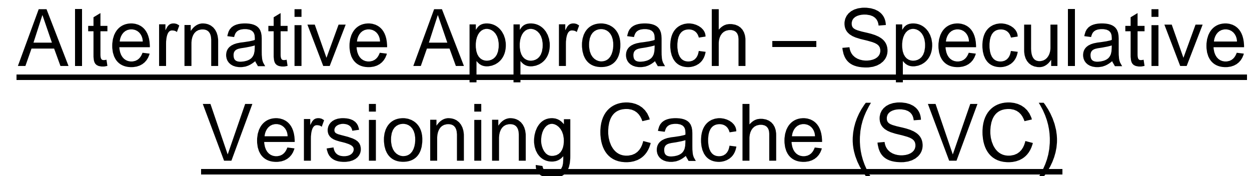
  → Speculative load data from memory

  → Thread is speculative and incorrect prediction

  When a thread is squashed, some additional words need to be invalidated:

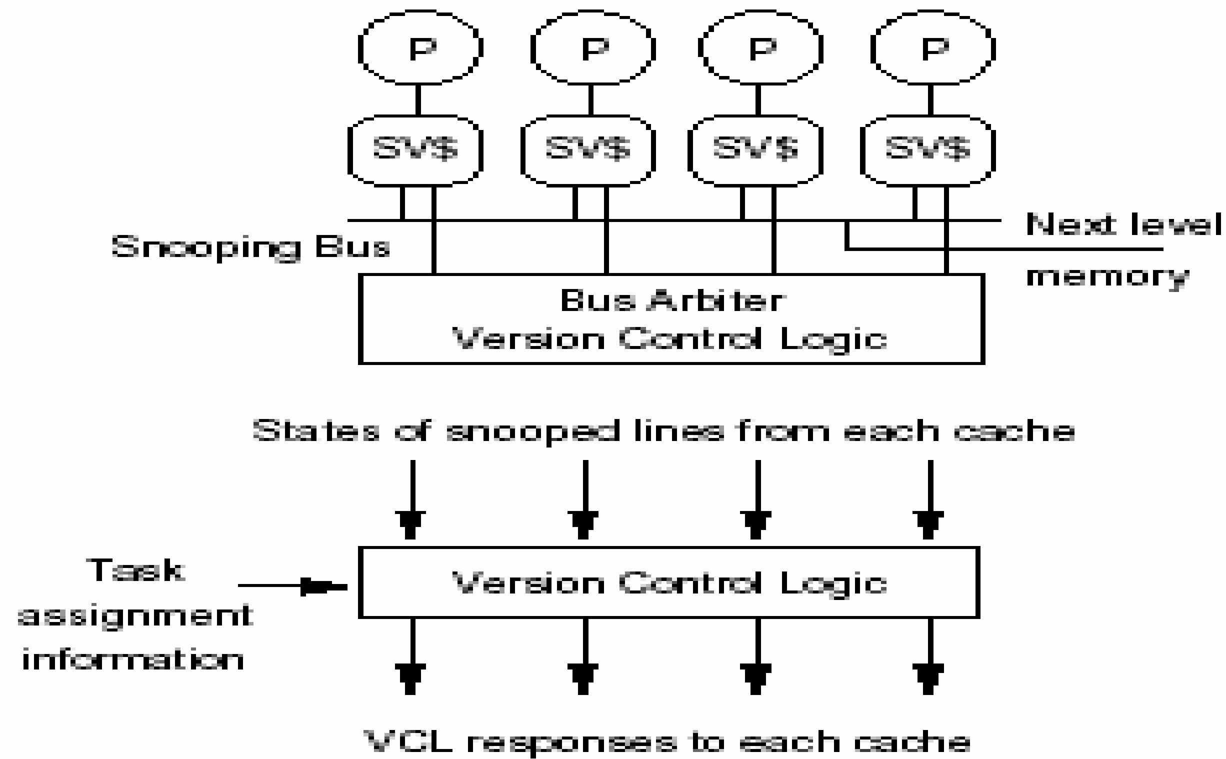
  →those are dirty;

  →those were forward from a squashed predecessor thread, with the help from FD-bit.

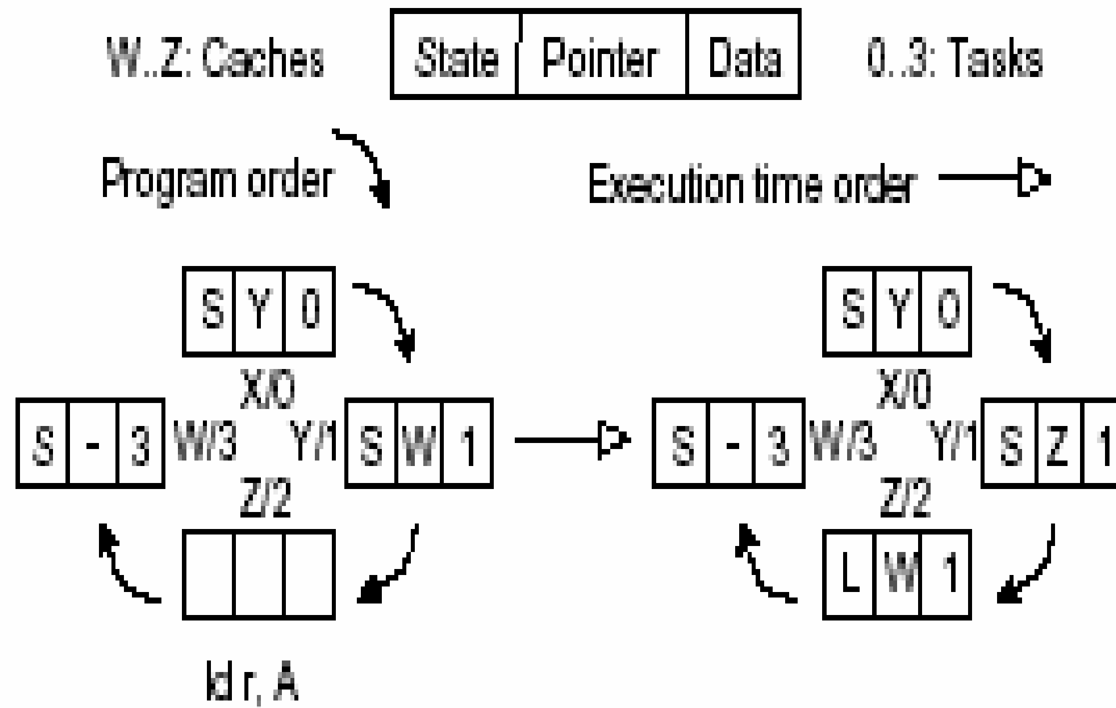


Operation	Actions
Load	Record use before ; supply the closest previous version
Store	Communicate store to later task; later tasks look for memory dependence violations
Commit	Writeback buffered versions created by the task to main memory
Squash	Invalidate buffered versions created by the task

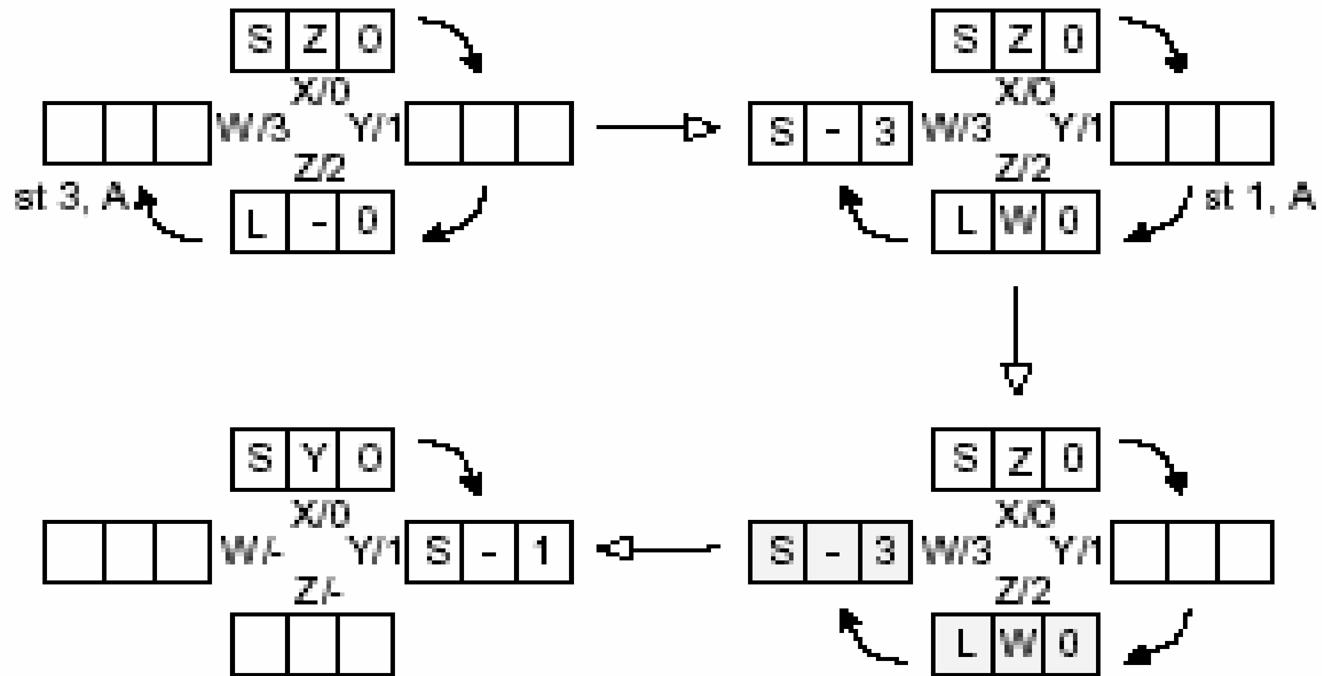
Base SVC Design



SVC Load



SVC Store



About the Presentation

- Problem Statement
- Handling threads
- Register-Level
- Memory-Level
- Evaluation
- Conclusion

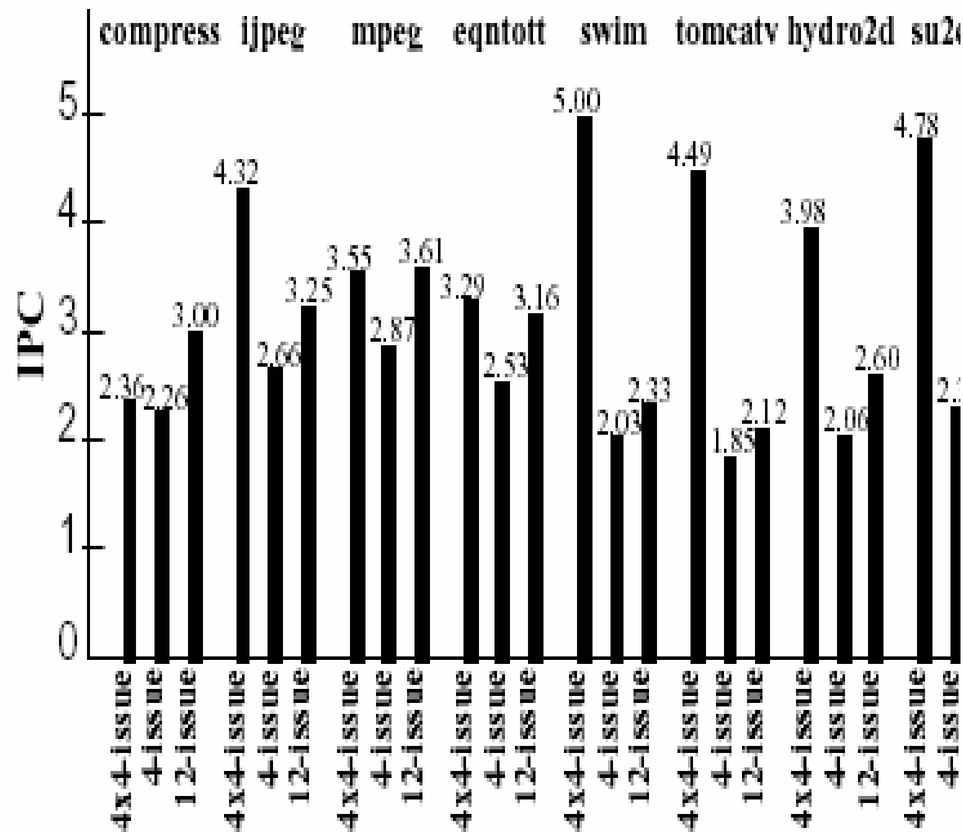
- 4-processor CMP, each core processor is modeled as MIPS R10000
- Compare with:

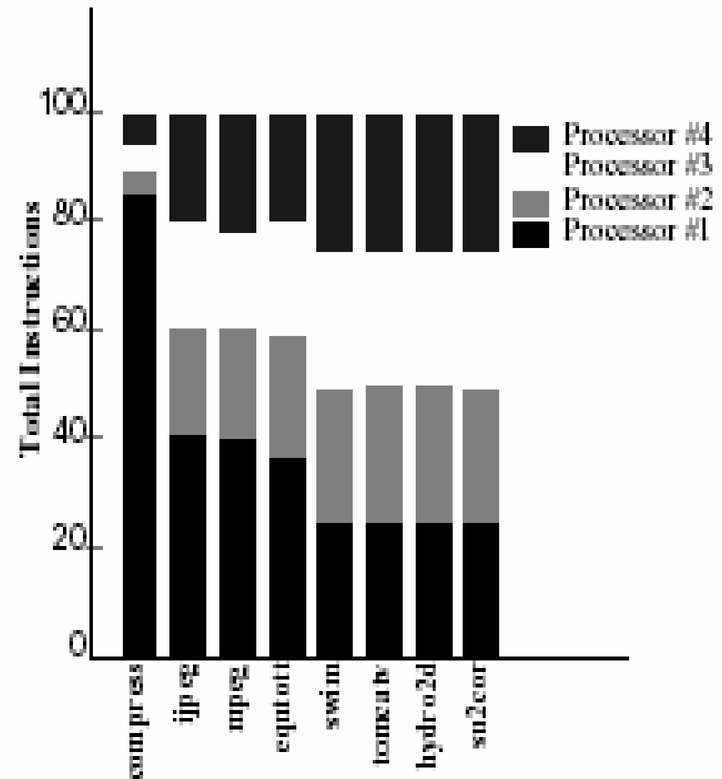
Issue Width	# of F. U. (int/ld-st/fp)	Entries in Instr. Window	# of Renaming Reg. (int/fp)
4	4/2/2	64	64/64
12	12/6/6	200	200/200

Memory Hierarchy Parameters

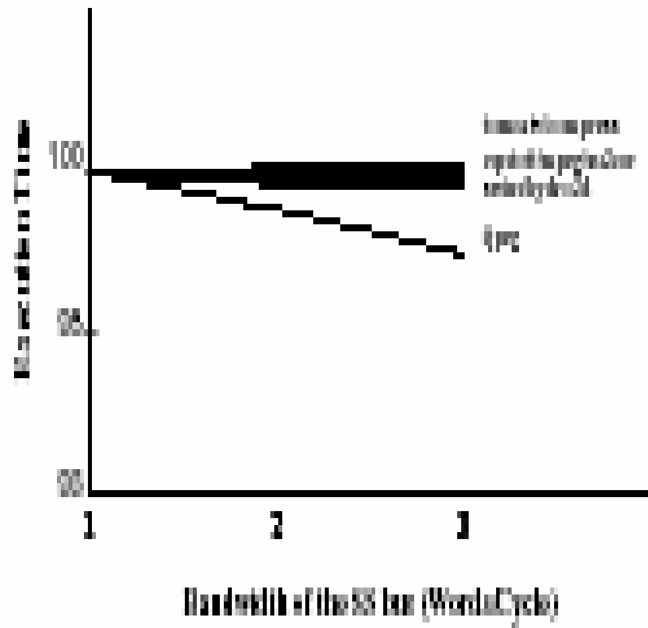
Parameter	CMP	4-Issue Superscalar	12-Issue Superscalar
[L1,L2] size (KB)	[4X16,1024]	[64,1024]	[64,1024]
[L1,L2] line size (B)	[32,64]	[32,64]	[32,64]
[L1,L2] associativity	[2,4]	[2,4]	[2,4]
L1 banks	3	7	7
L1 latency (cycle)	1	1	2
L2 latency (cycle)	6	6	6
Mem. Latency (cycle)	26	26	26

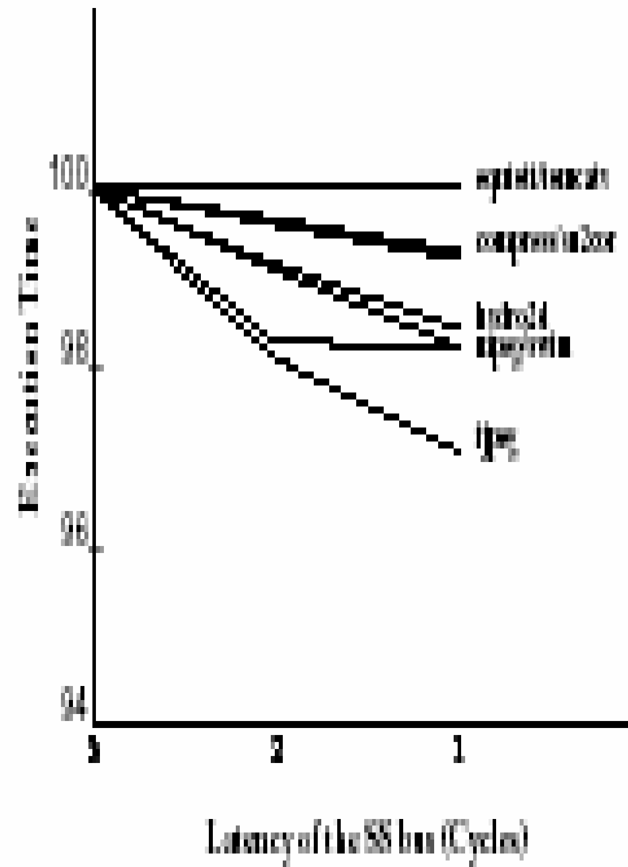
Performance Evaluation



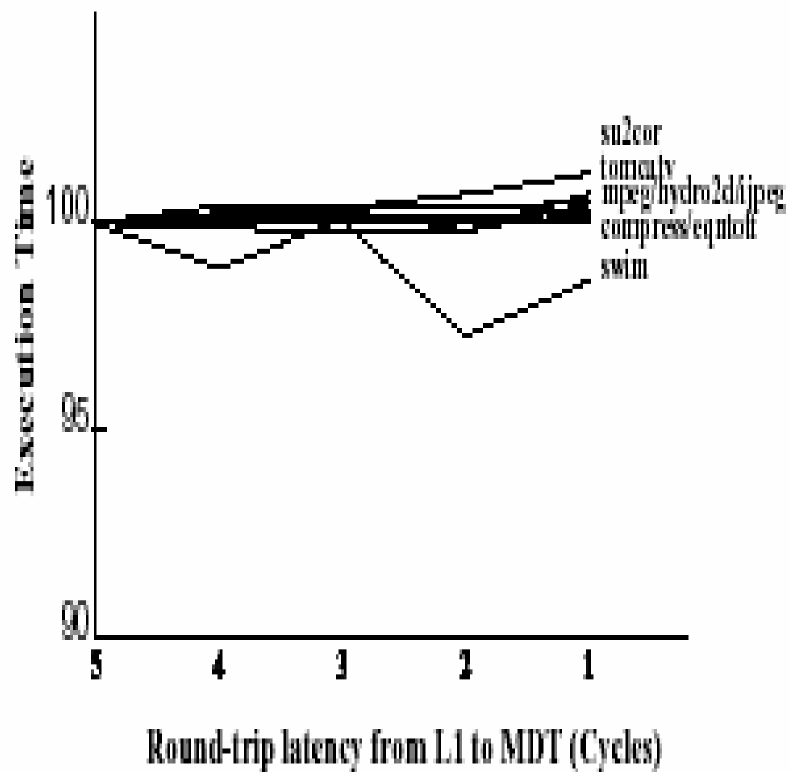


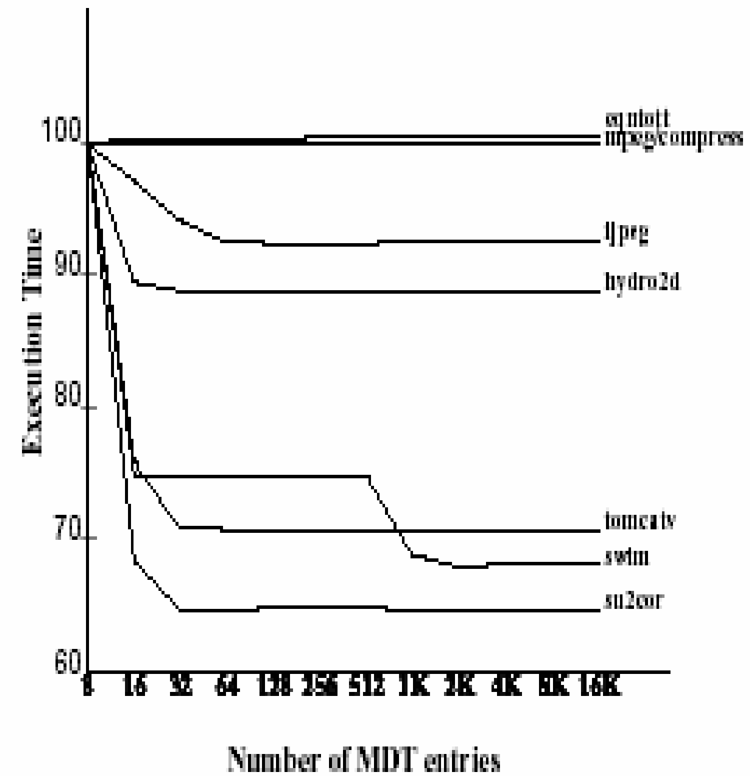
SS Bus BW Evaluation





MDT Evaluation





About the Presentation

- Problem Statement
- Handling threads
- Register-Level
- Memory-Level
- Evaluation
- Conclusion

Conclusion

- Fundamental requirement - sequential program semantics
- Complicity variation between register-level and memory-level – crack differently
- Holding sequential program semantics, go to high-performance – more thread identifying

- Helpful on-line papers available:
  1. Dr.Sohi's Multiscalar: http://www.cs.wisc.edu/~mscalar
  2. Stanford Hydra CMP: http://www-hydra.stanford.edu
  3. IEEE digital library