Vulcan: Hardware Support for Detecting Sequential Consistency Violations Dynamically

Abdullah Muzahid, **Shanxiang Qi**, and **Josep Torrellas**
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

MICRO
December 2012

Sequential Consistency (SC)

PA	PB
A0: x =1	
A1: y =1	
	B0: p = y
	B1: t = x

A0	A0
A1	B0
B0	A1
B1	B1

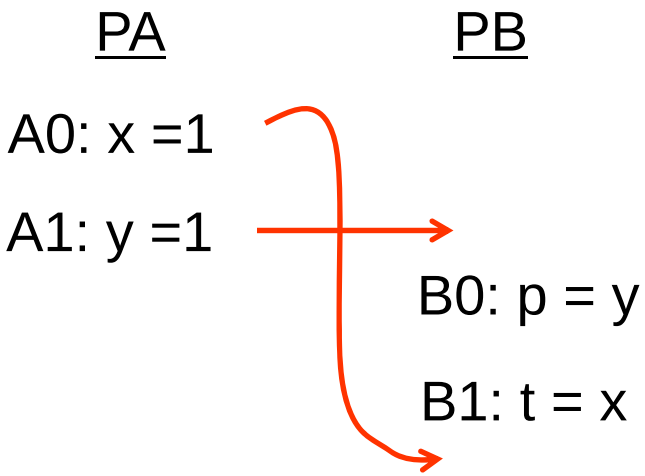
- In SC, memory accesses:
  - Appear atomic
  - Have a total global order
  - For each thread, follow program order

Sequential Consistency Violation (SCV)

- SCV: access reorder that does not conform to SC
- Machines support relaxed models, not SC
- Machines may induce SC violations (SCV)

initially x=y=0

In SC, if p=1 then t=1

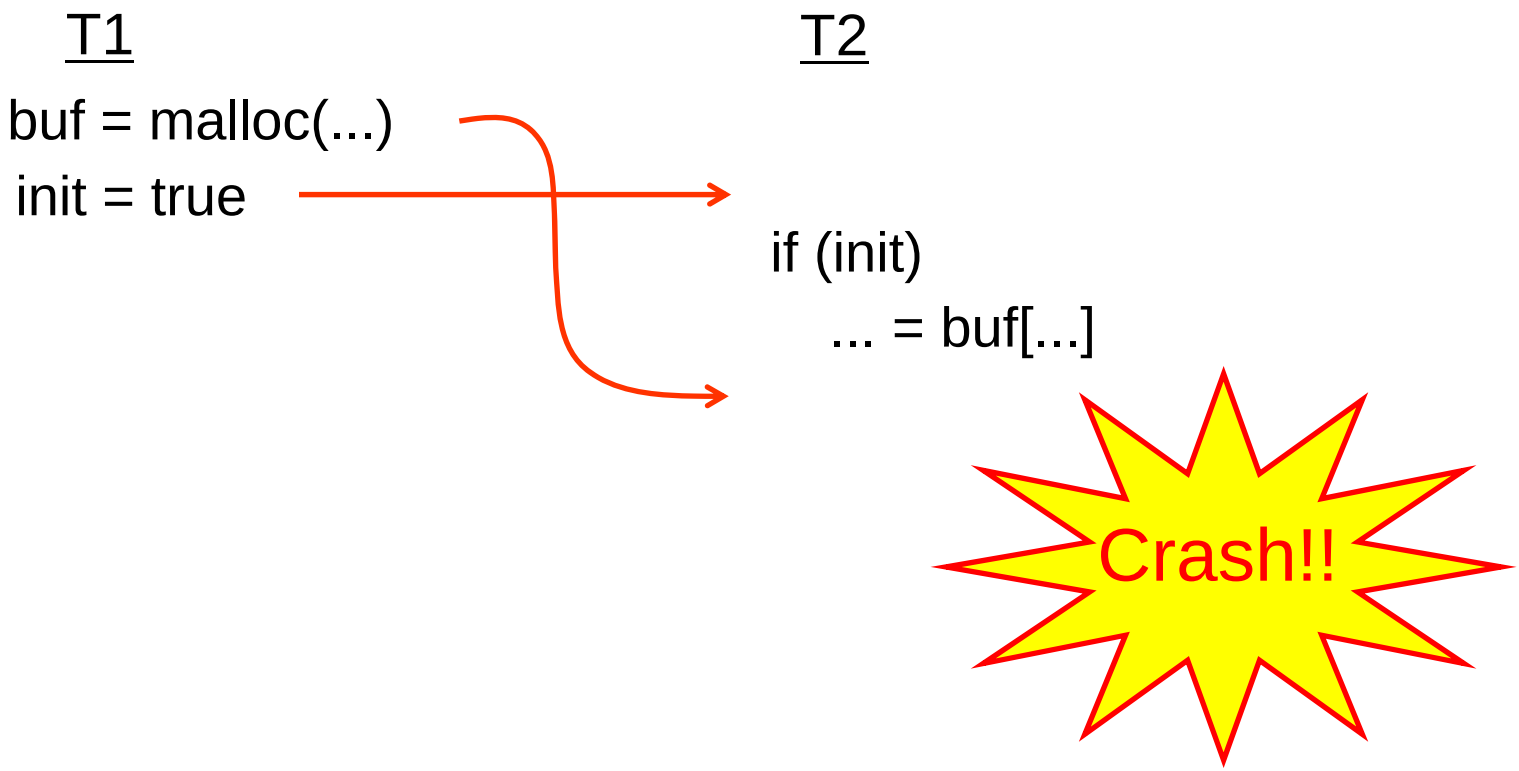


A1
B0
B1
A0

p is 1
t is 0

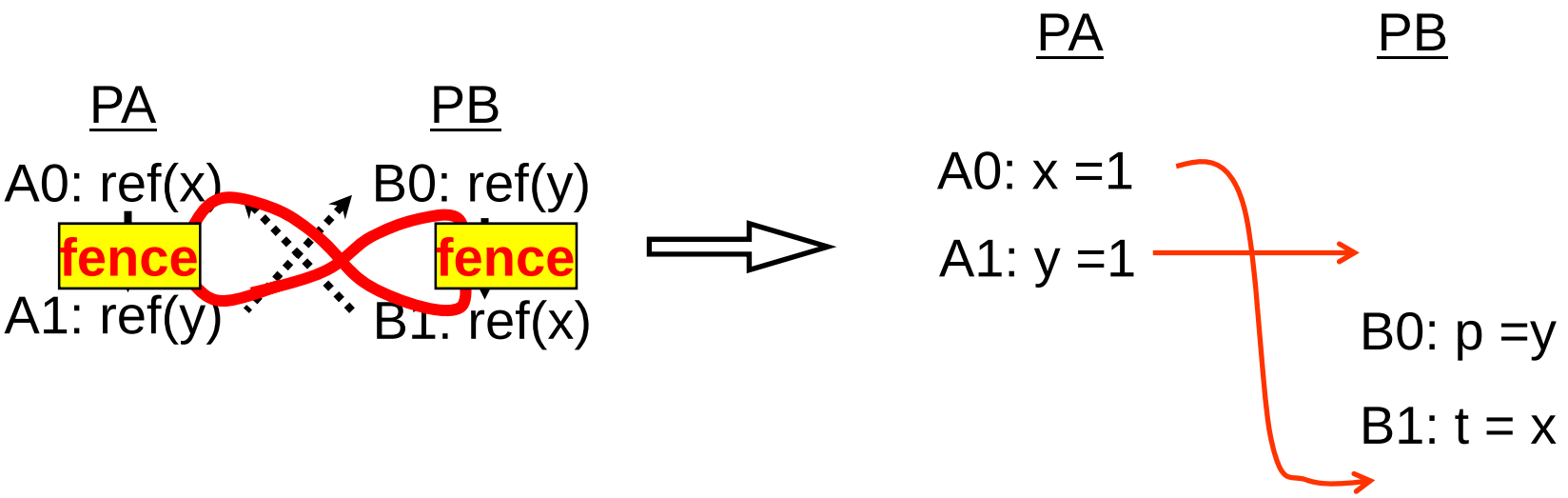
Very unintuitive bug

Example of SCV



When Can an SCV Occur?

- Two or more data races overlap
- They create a cycle



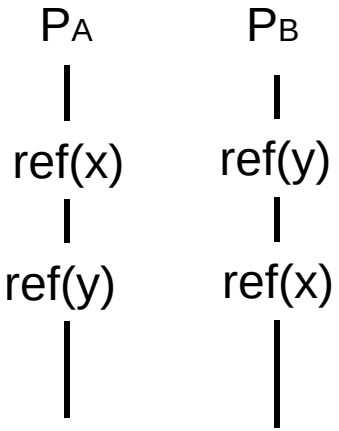
Why Detecting SCVs is Important?

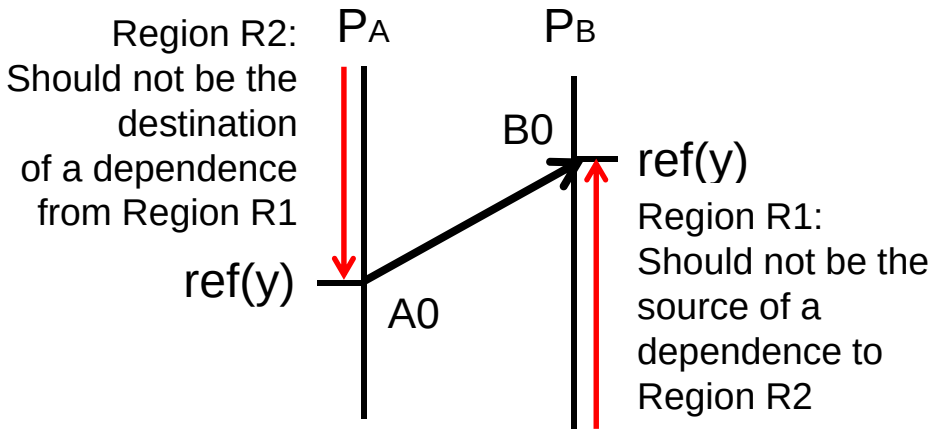
- Programmers assume SC
  - SCV is almost always a bug: unexpected interleaving
  - Single-stepping debuggers cannot reproduce the bug
- Causes portability problems (e.g. Intel TBB)
  - Code may not work across machines
- Lock-free data structures sometimes explicitly use races but rely on SC
  - Traditional data race detectors won't work
- Around 18% of reported races can cause SCV (see paper)

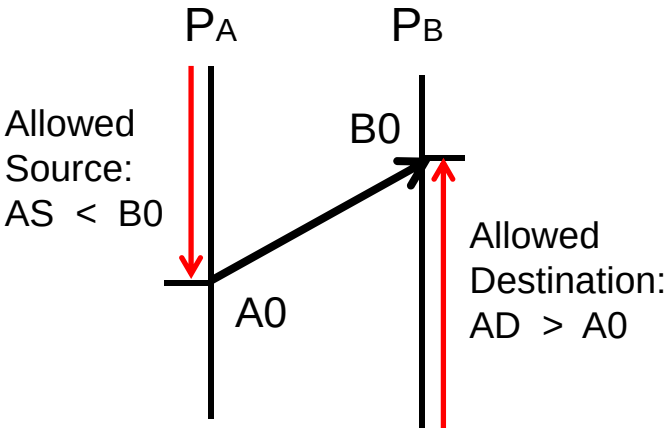
Proposal: Vulcan

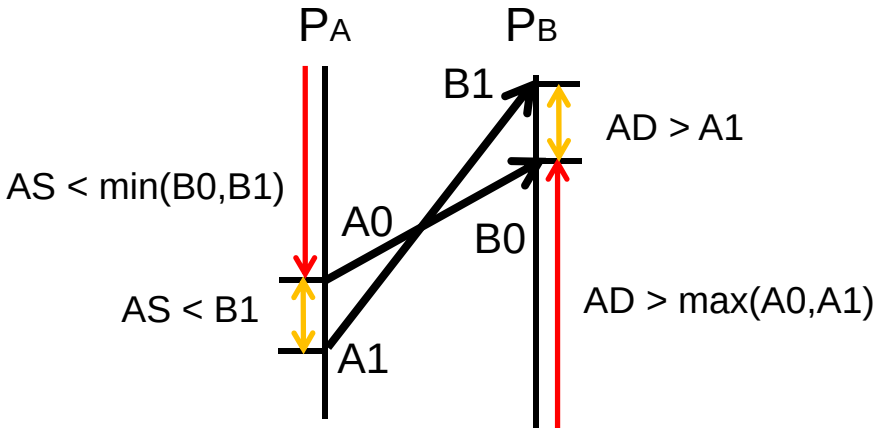
- Detects SCVs in relaxed consistency machines in highly precise manner
  - No false positives; no false negatives
- Provides info to debugger to debug SCV
- No SW changes; only use executable; negligible execution overhead
- Idea: Use HW to detect cycles of inter-thread dependences at runtime
- Approach:
  - Use the cache coherence protocol to dynamically record dependences
  - Interrupt the processor when a cycle is about to occur

Basic Algorithm

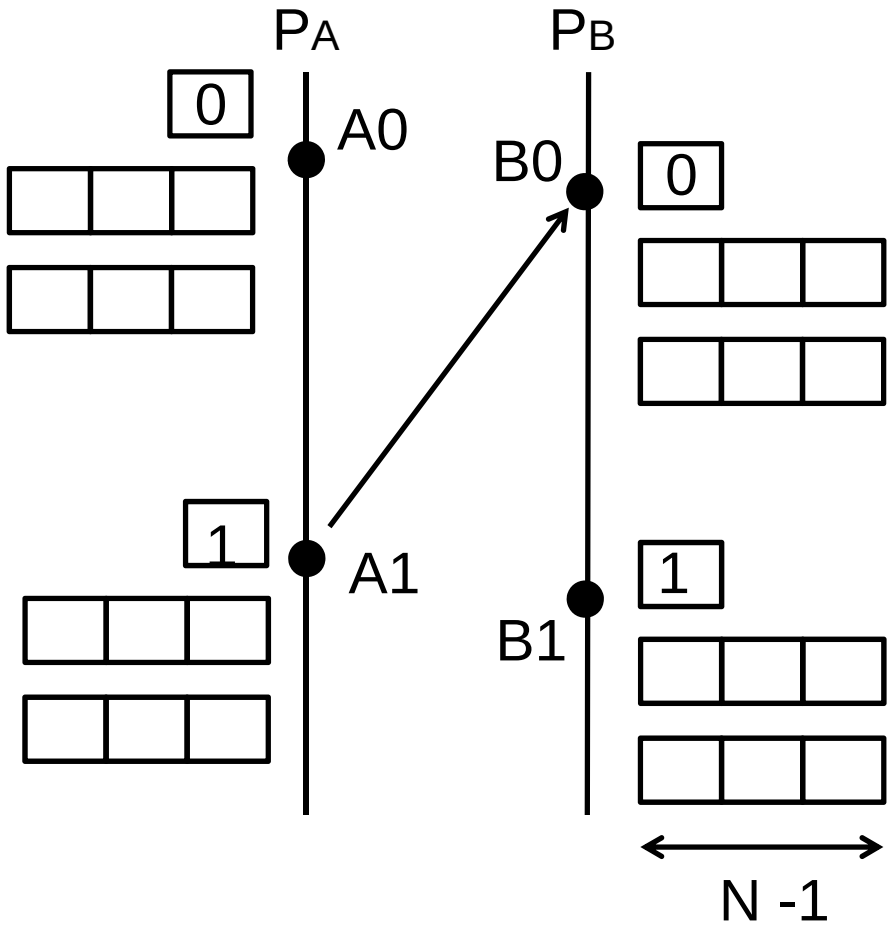








Hardware Structures

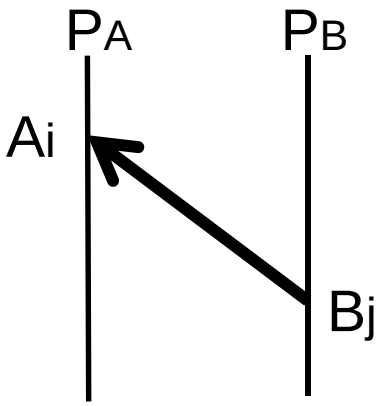


SN: Sequence Number

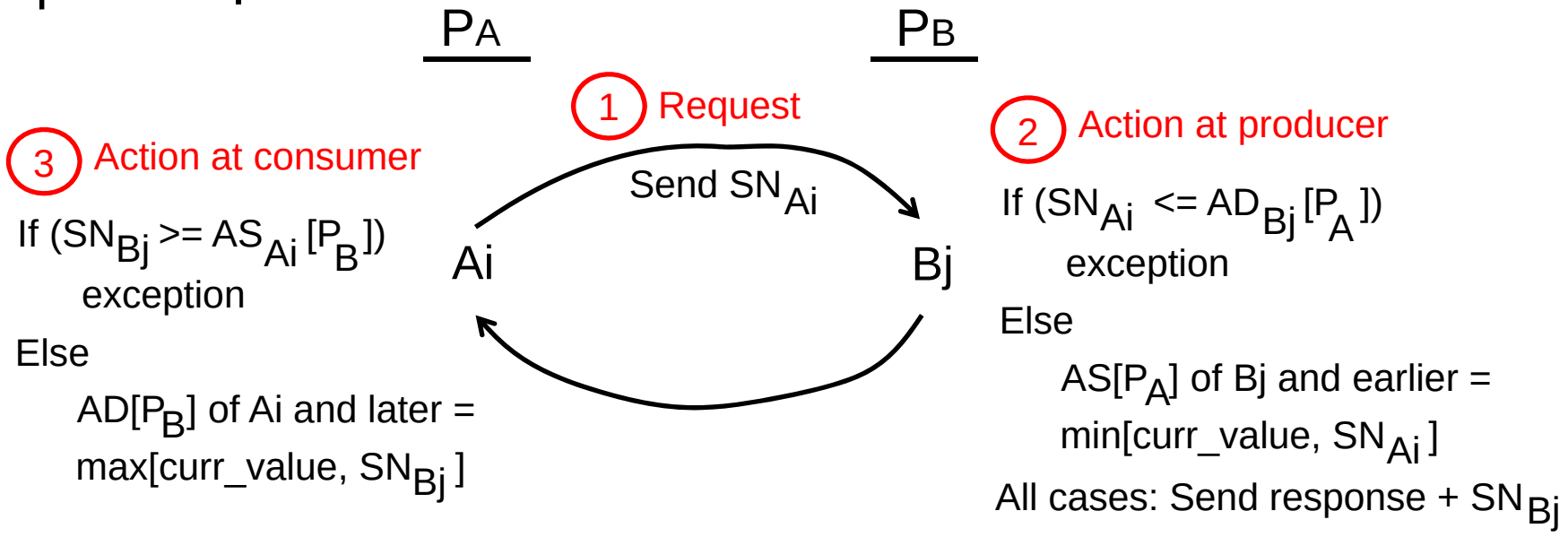
AS: Allowed Source

AD: Allowed Destination

N: # of processors

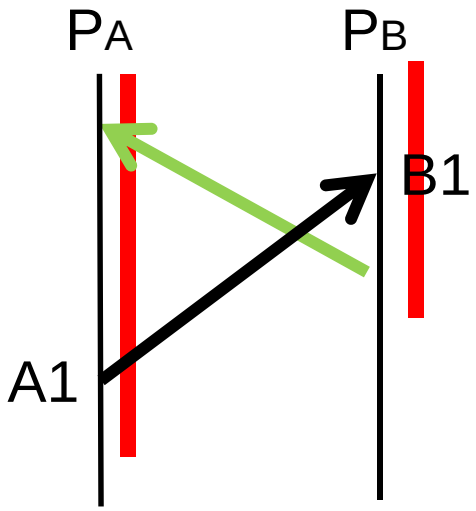


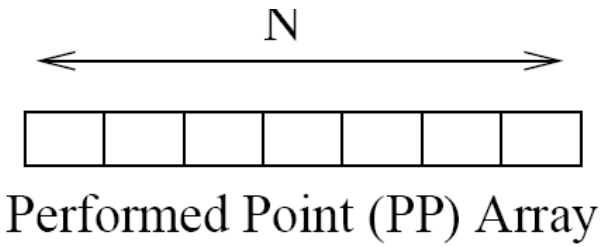
Hardware Checks



Safe Accesses

- An access is Safe when it cannot cause an SCV anymore
  - The access and all its predecessors are performed and
  - All of disallowed destinations (in all the other procs) are performed





SC Violations and Safe Accesses

- When an SCV occurs the following must be true:
  - In the two arrows that form the cycle, the source reference is Unsafe wrt the destination processor

How Long to Keep Metadata?

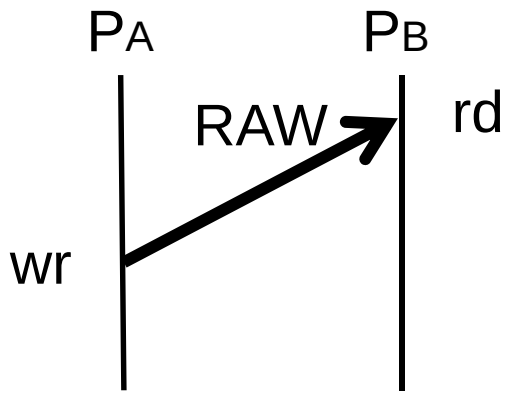
- Keep metadata as long as the access can participate in an SCV
  - Keep metadata for Unsafe accesses only

Unsafe accesses = Pending + Disallowed_destinations_not_perf

- Store metadata in a per-processor SC Violation Queue (SCVQ)
  - Contains address + SN + AD[] + AS[], not data

Detecting Dependences and Cycles: Single Word Cache Line

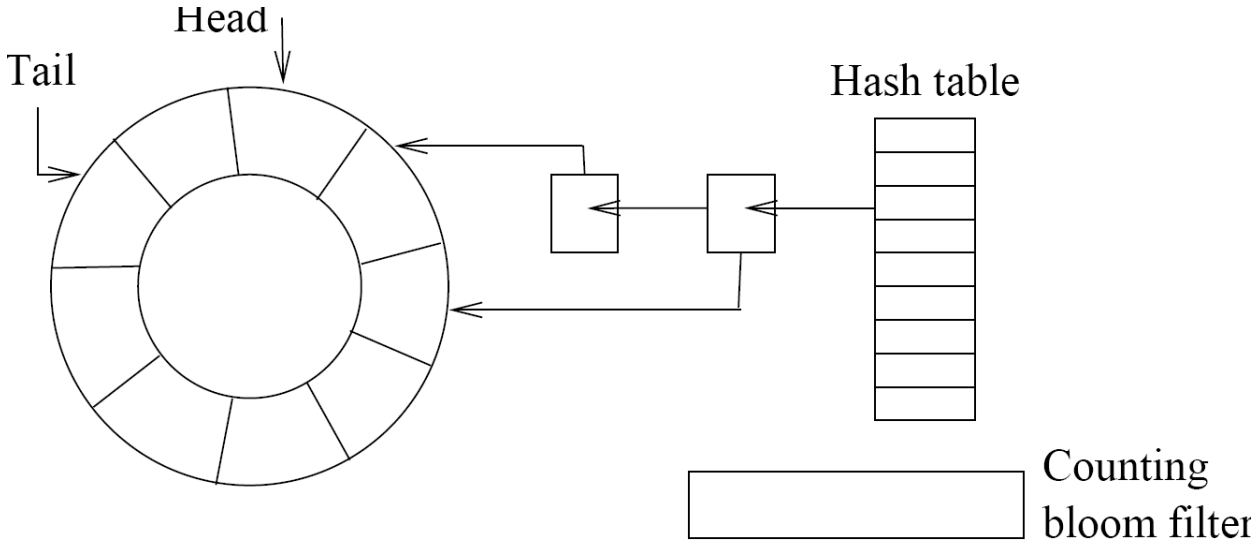
- Inter-thread dependence induces a coherence bus transaction
  - Bus transaction searches the SCVQs of other processors
  - If hit, src and dst references exchange SN and run the Vulcan algorithm



Same for WAW, WAR

SC Violation Queue (SCVQ)

- Keeps Vulcan metadata for Unsafe local load/stores



- Need efficient search; cannot rely on cache snooper
- Counting bloom filter to minimize useless SCVQ lookups

Detecting Dependences and Cycles: Multiword Cache Line

See Paper for Details

- When the destination reference of an inter-thread dependence occurs…
  - Either coherence protocol triggers a coherence bus transaction
  - Or Vulcan forces a metadata update bus access
- Implementation: Vulcan adds V-State per byte in each line
  - Tracks whether the latest dependence on that word has already been recorded
  - If not recorded when processor accesses the word, even if no coherence action, force a metadata bus access

Issues

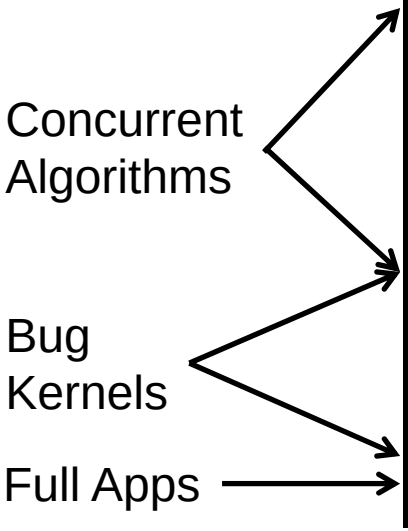
With these constraints
→ No false positives, no false negatives

- Advantages:
  - Detects actual SC violations, not data races
  - Works for any memory model
  - Low overhead
- Limitations:
  - Race cycles involving only two processors (very large majority)
  - Not concerned with impact of compiler transformations on SCV

Multicore Modeled

Modeled a multicore chip with 8 processors

- Core: Out of order, 2-issue width
- RC memory model
- Private L1, Shared L2
- Cache line size: 32 bytes
- Byte-level V-State bits
- SCVQ size: 256 entries



PROGRAM	DESCRIPTION
Dekker	Mutual exclusion
Lazylist	List-based concurrent set
Snark	Non-blocking double-ended queue
Harris	Non-blocking set
Pthread from glibc	Unwind code after canceling a thread
Crypt from glibc	Small table initialization code
DCL bug	Kernel using double-checked locking
SPLASH-2	8 programs from SPLASH-2

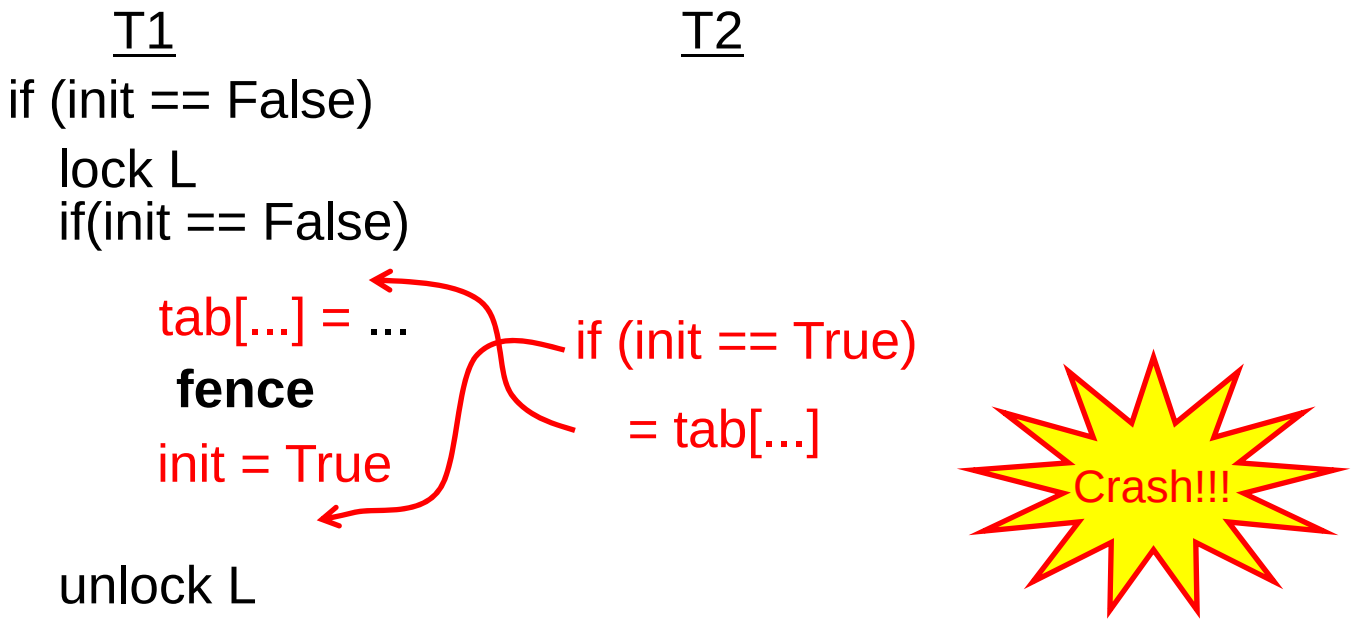
Vulcan Effectively Detects SCVs

Program	SC Violations Found		
	Unique	Total	New?
Dekker	1	224	
Lazylist	1	150	
Snark	1	1467	
Harris	1	18	
Pthread	**2**	**142**	**Y**
Crypt	**2**	**130**	**Y**
DCL	1	2	
fmm (SPLASH2)	**3**	**18**	**Y**

- Vulcan detects 3 new bugs in important codes (libraries)

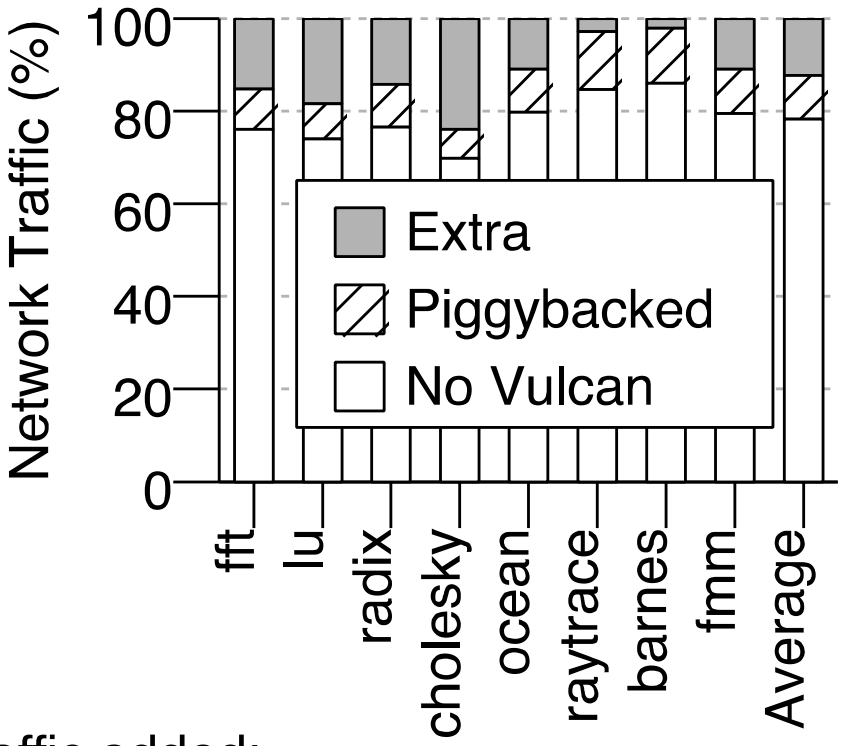
Example New Bug: Crypt Library Bug

- Found a new SCV in a bug fix



- Branch condition predicted TRUE although not TRUE
- THEN code uses the old tab[] (wrong one)
- Tab[] is updated
- Branch prediction is later confirmed correct

Overhead



Traffic added:

- 9% due to piggybacked
- 12% due to extra bus accesses

App	Exec Overhead(%)
fft	9.5
lu	3.6
radix	1.4
cholesky	9.0
ocean	12.3
raytrace	6.9
barnes	2.7
fmm	2.8
avg	**6.0**

Low overhead: OK for on-the-fly

Also in the Paper

- Full description of the protocol for multi-word cache lines
- Information that a debugger would get after the exception
- HW structure sizes and cost
- Comparison to related work

Conclusions

- SCV bugs are arguably the hardest type of concurrency bugs
- Vulcan is the first HW scheme to detect these bugs with high precision
  - No false positives; no false negatives
- It has low execution overhead for on-the-fly deployment
  - 6% for 8-proc runs; 4.4% for 4-proc runs
- It detects 3 previously unknown bugs in popular libraries

Lots of work to do!

Vulcan: Hardware Support for Detecting Sequential Consistency Violations Dynamically

Abdullah Muzahid, **Shanxiang Qi**, and **Josep Torrellas**
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

MICRO
December 2012