

# Effective Data-Race Detection for the Kernel

John Erickson, Madanlal Musuvathi, Sebastian  
Burckhardt, Kirk Olynyk

Microsoft Research

Presented by Thaddeus Czauski

06 Aug 2011

CS 5204

A problem has been detected and windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE\_FAULT\_IN\_NONPAGED\_AREA

If this is the first time you've seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

\*\*\* STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

\*\*\* SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c



# How do we prevent data races in programs?

- User Mode
- 'Synchronized' code blocks
- Locks on clearly defined objects/code blocks
- Kernel Mode
- Interrupt services
- Deferred procedure calls
- Kernel to Hardware Sync
  - Direct Memory Accesses

The kernel employs a variety of non-trivial synchronization mechanisms



# Traditional methods of detecting data races

- Happens-Before
- Monitor thread execution
- Observe all shared accesses are sequential
- Lock-Set
- Monitor memory accesses
- Observe that at least one synchronization primitive is in use

Need to know how synchronization primitives work



# How does DataCollider work?

- What is a data race?
- Looking for races w/o knowing what a lock is
- DataCollider overview
  - Static Analysis before application launches
  - Dynamic Analysis as application executes
  - Heuristically filtering benign races when the application finishes
- Testing DataCollider
- What we found out about DataCollider



# What is a data race?

- Conflicting memory access by different threads to the same location
  - The memory being accessed is
    - Not disjoint, or is the at the same location
    - At least one access is a write
    - The data being access is not related to a synchronization structure like a lock.
- When an application can be executed on a multiprocessor in such a way that two conflicting memory accesses are performed simultaneously (by processors or any other device).



# Hybrid Analysis: Looking for races without knowing how locks work

1. Monitor memory accesses via code breakpoint
2. Pause the current thread when a breakpoint is hit
3. See if any other thread writes to the shared memory via data breakpoint

Don't need to be aware of synchronization primitives or program ordering to identify when a conflict occurs



# Static analysis: just before the application executes



Disassemble the program



Scan for memory accesses



Prune out local stack accesses & synchronization accesses

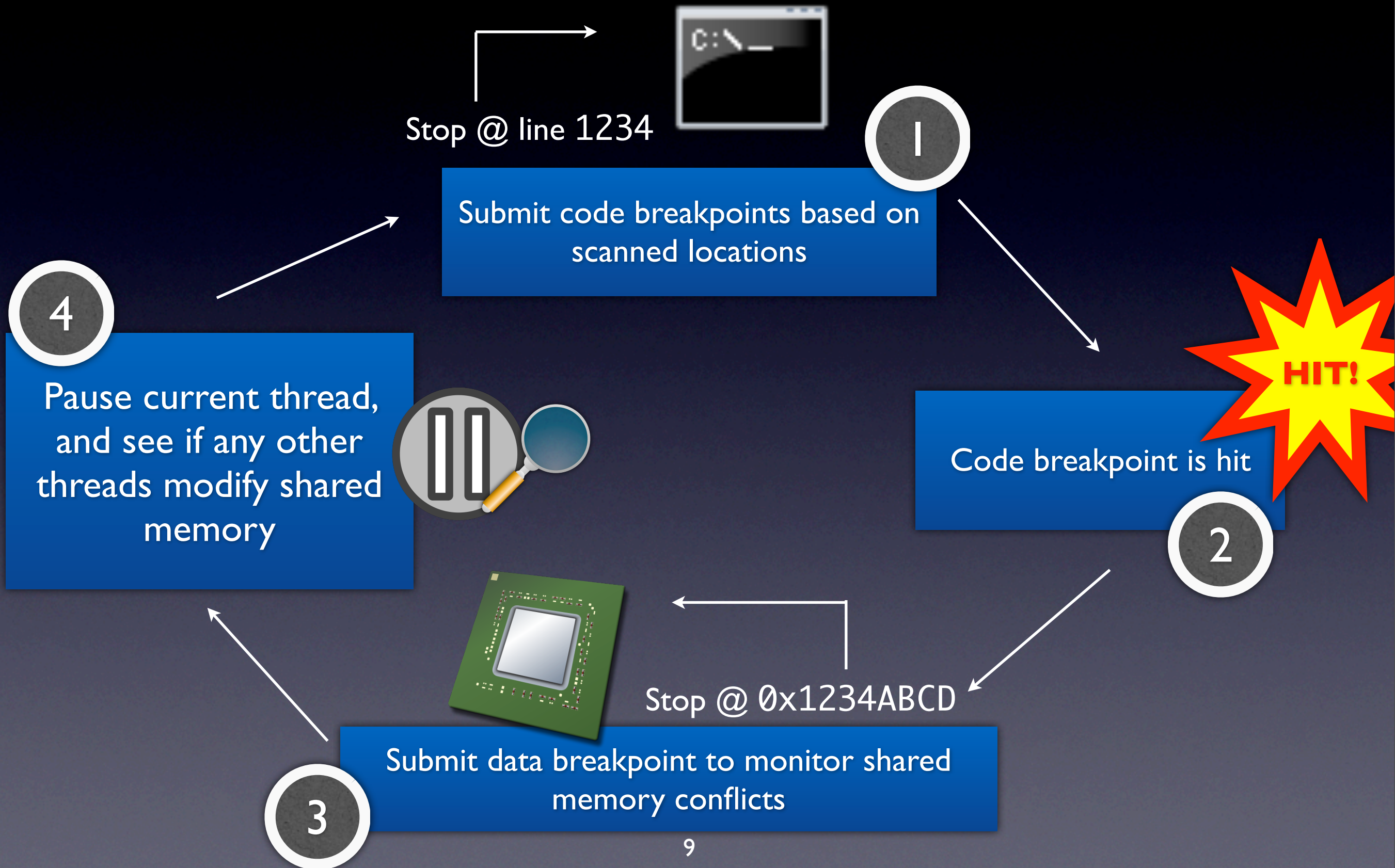


`#explicit`

Add in any memory accesses explicitly annotated in code



# Dynamic analysis: monitoring running programs





# Pausing threads and looking for racing threads

- Ensure that other threads are still executing by monitoring other breakpoints being hit
  - Resume execution if no other code breakpoints are firing
- Pause the current thread for a max of 15 ms
  - Reduce this time to  $< 1$  ms for threads with higher IRQs, like those running at DISPATCH



# Have the CPU find the racing thread

- Submitting data breakpoints to the CPU
  - Modern CPUs can break when a particular address is accessed
  - With paging enabled, the CPU uses virtual addressing. This can cause problems:
    - Kernel threads accessing user space can map to different physical addresses
    - Session memory can have one virtual address which maps to different physical addresses
    - Direct Memory Accesses don't involve another thread, so thread to break on

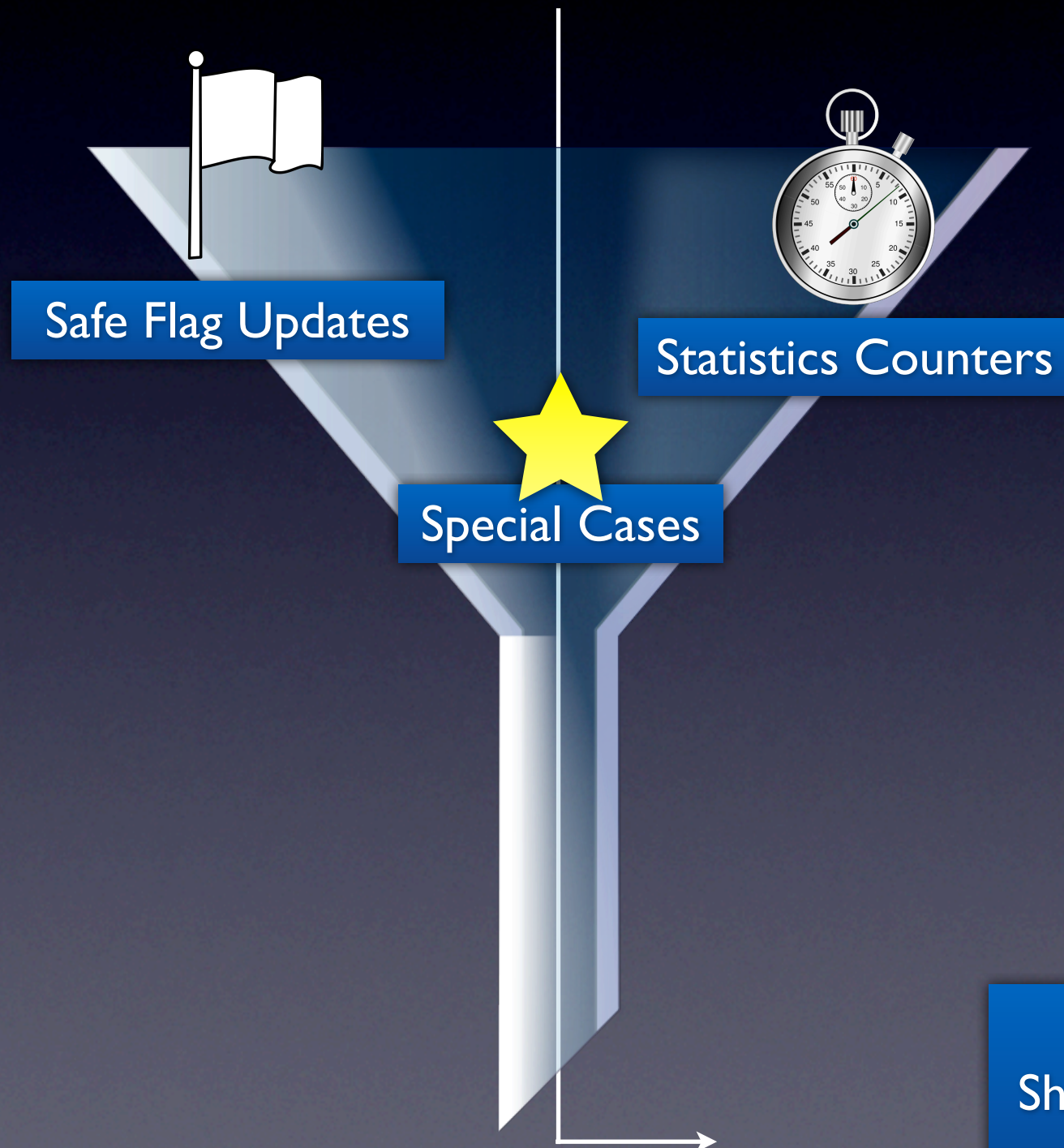


# When data breakpoints don't work

- Repeatedly reading a particular address, and looking for modifications. Has a few limitations:
  - Cannot detect multiple reads of an address
  - Cannot detect multiple writes, which set the same value
  - Only catches one thread in the conflict. Need to manually identify the racing thread.



# Using heuristics to remove false-positives when the program finishes

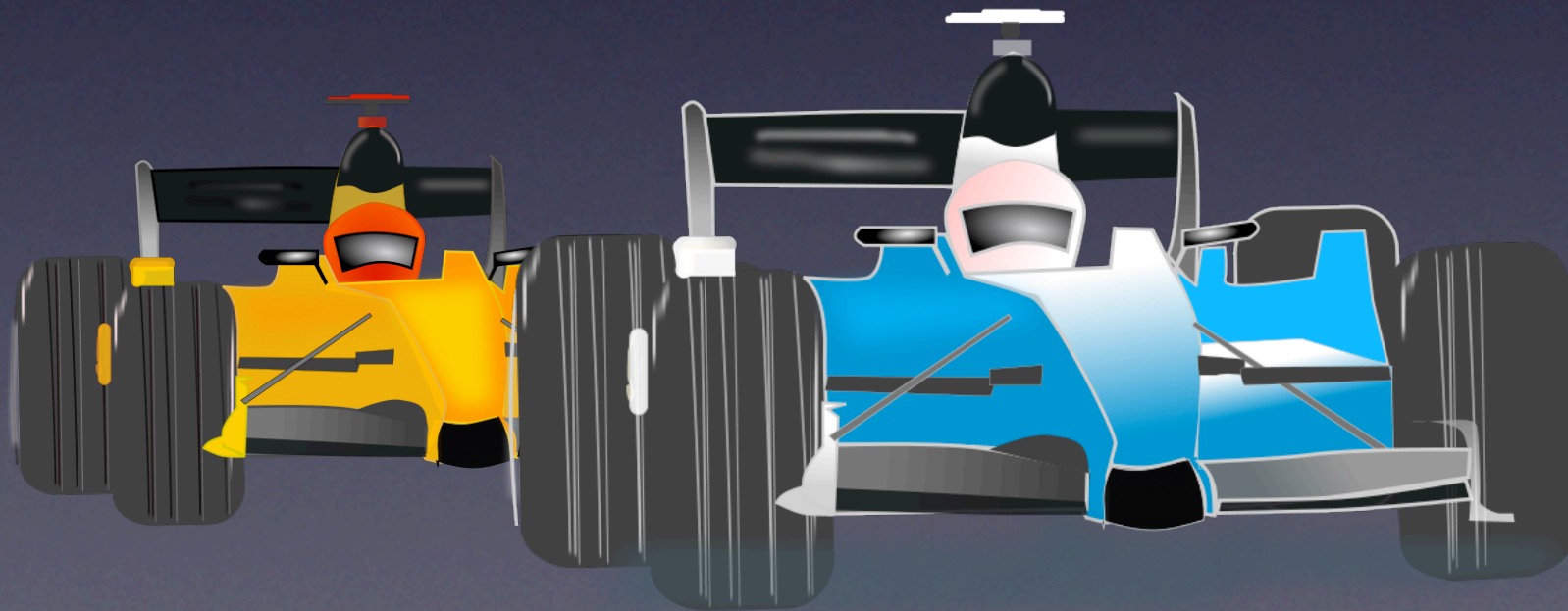


Should be left with actual data races



# Experimental setup: testing DataCollider

- Ran DataCollider as a part of Windows 7 kernel (x86) stress tests to look for data races.





# More on testing DataCollider

- Tested Windows 7 running within a VM
- Host Machine
  - Windows Server 2008
  - 2x Intel Core2 (Quad Core @ 2.4 GHz)
  - 4GB RAM
- Virtual Machine
  - Windows 7 (x86)
  - Limited to 50% of host CPU
  - 512MB RAM
- Measured time elapsed to complete boot-shutdown sequence with DataCollider versus non-DataCollider executions within the VM



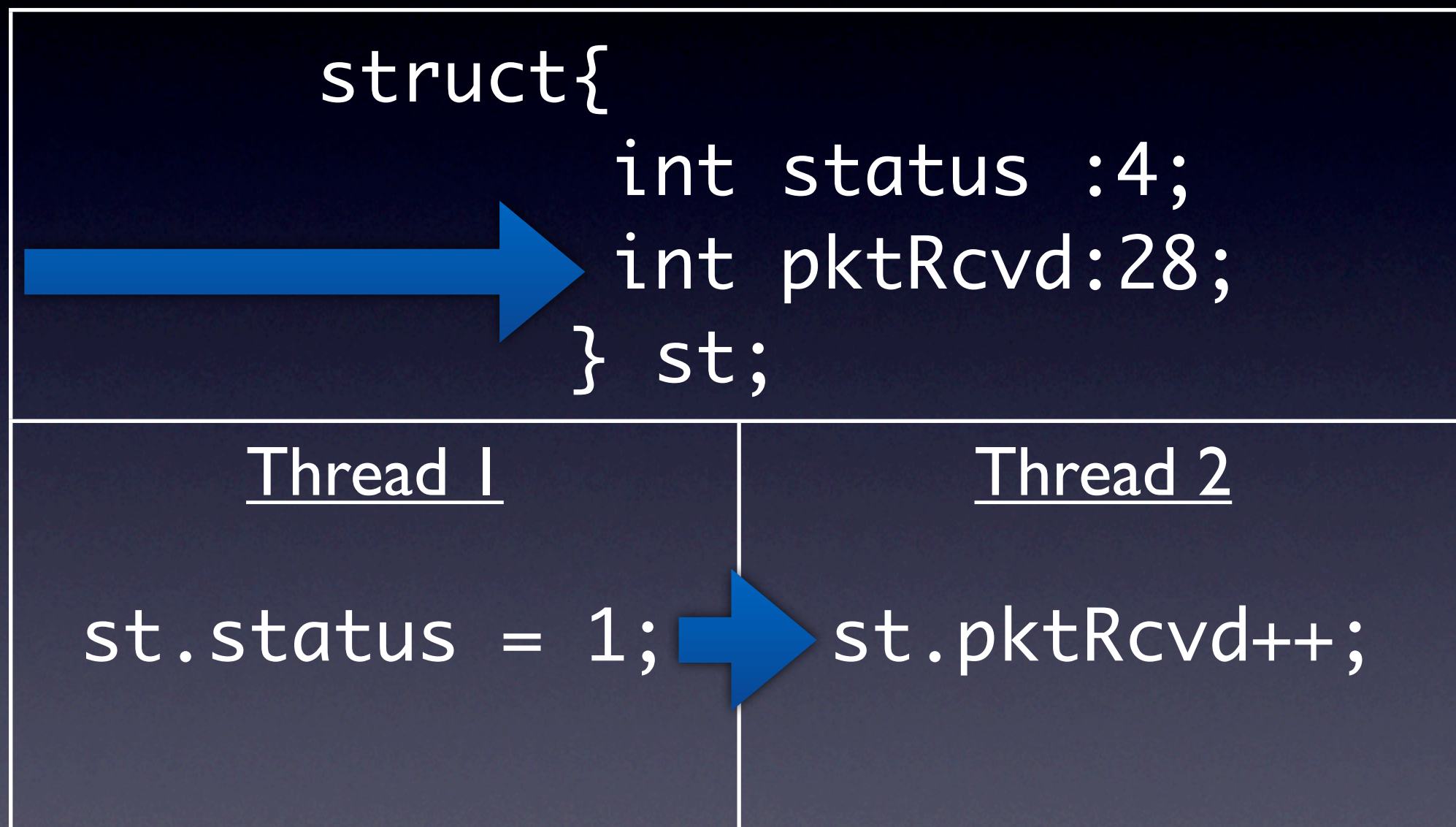
# Here's what we found during our execution tests

Data Races Reported	Count
Fixed	12
Confirmed as being fixed	13
Under investigation	8
Harmless	5
<b>Total</b>	<b>38</b>

25 issues confirmed as races, and 12 have already been fixed



# Bit Field Accesses

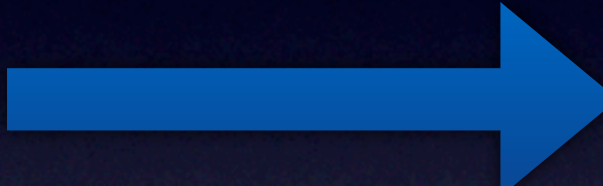


Updates to pktRcvd can hide updates to status



# Flag Accesses

```
void AddToCache() {  
    // ...  
    A: x &= ~(FLAG_NOT_DELETED);  
    B: x |= FLAG_CACHED;  
    MemoryBarrier();  
    // ...  
}
```



If preempted before reaching B, then flag assertion may fail in concurrent threads which

are past the barrier

```
AddToCache();  
assert( x & FLAG_CACHED );
```



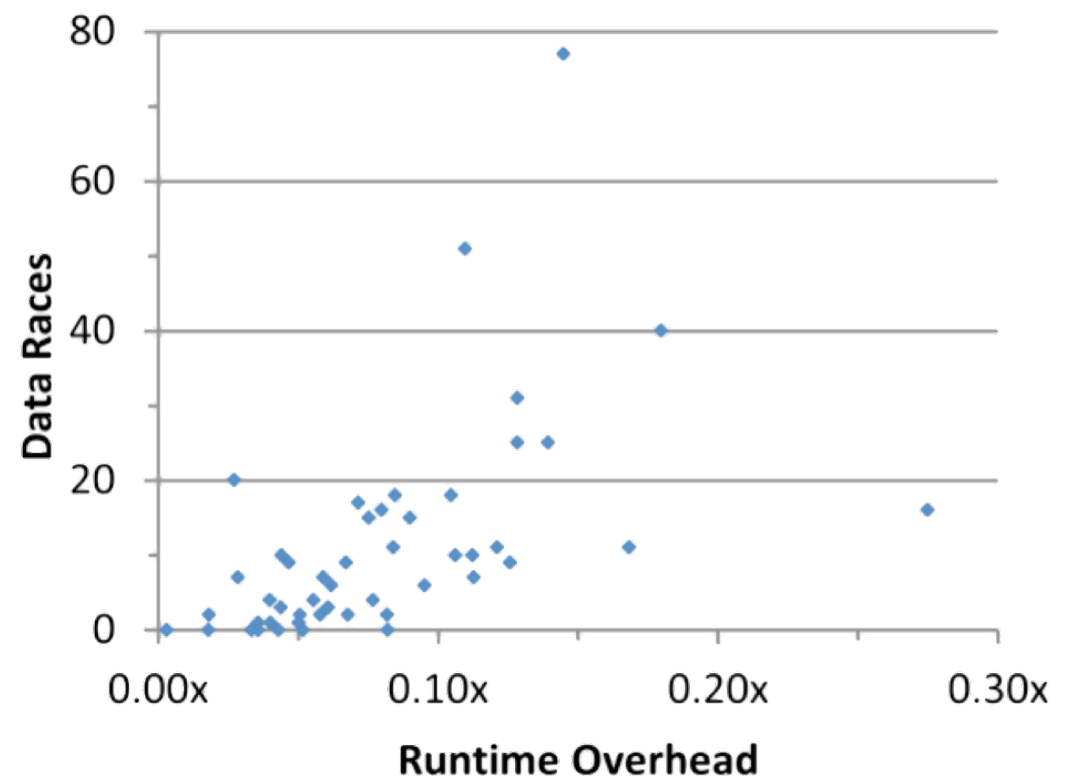
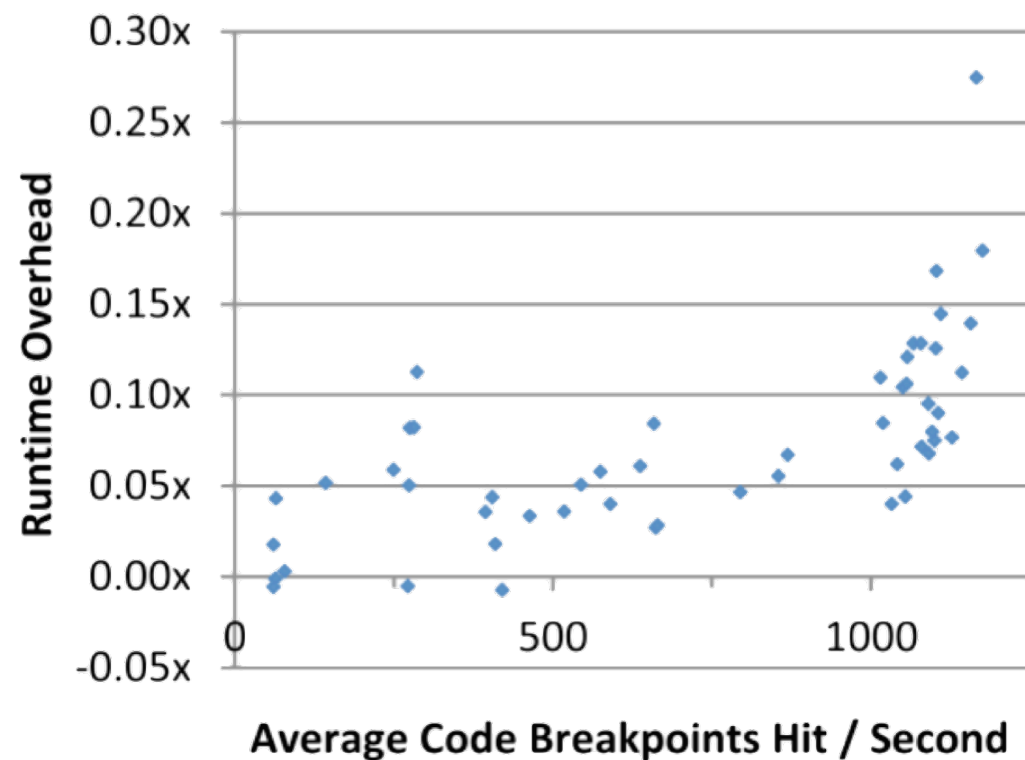
# Boot Time Hangs

- Vendor was seeing a hang at boot
- Could not replicate in the lab without vendor hardware
- Developer manually had to find the issue
  - Driver data corruption (non-atomic status update)
- DataCollider found the same issue within one hour of stress testing

DataCollider did not need a hang condition to find this data race



# Performance when running DataCollider with Windows 7



- Was able to run DataCollider with  $<5\%$  overhead
- Overhead required is fairly linear as the average number of code breakpoints is increased



# Conclusion

- **DataCollider is a great application for finding data races in low-level code**
  - Light-weight and simple application
    - Identifies data races with low overhead
  - Effective at identifying race conditions
    - Does not need to know how locks work
    - Can identify which threads are racing



# Questions?



Thank You!