

2007 JavaOne Conference

Debugging Data Races

Dr. Cliff Click

Chief JVM Architect & Distinguished Engineer

blogs.azulsystems.com/cliff

Azul Systems

May 8, 2007



A Short Debugging Tale



```
if (method.hasCode() != true )
    return false;
code = method.getCode();
...setup;
code.execute(); // Throws NPE rarely!!!
return true;
```

- Example is real; simplified for slide
 - Many more wrapper layers removed
 - Shown “as if” aggressive inlining
- I've debugged dozens of slight variations

Hash Tables



- Constant-Time Key-Value Mapping
- Fast arbitrary function
- Extendable, defined at runtime
- Used for symbol tables, DB caching, network access, url caching, web content, etc
- Crucial for Large Business Applications
 - > 1MLOC
- Used in Very heavily multi-threaded apps
 - > 1000 threads

Agenda



www.azulsystems.com

- **(not very) Formal Stuff**
- Reordering Memory – A peek under the hood
- Common Data Races
- Finally(!) Some Debugging Techniques
- Summary

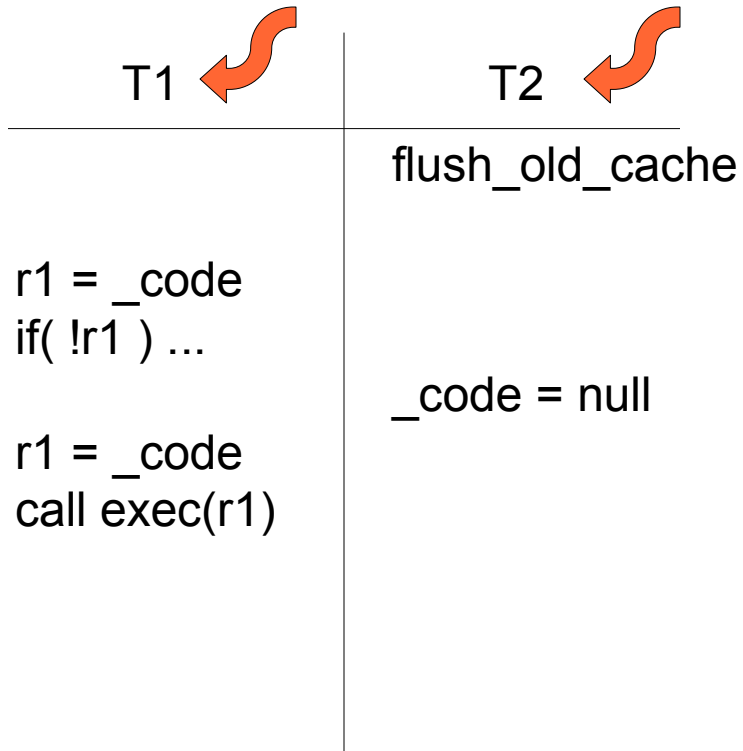
What IS a Data Race?



www.azulsystems.com

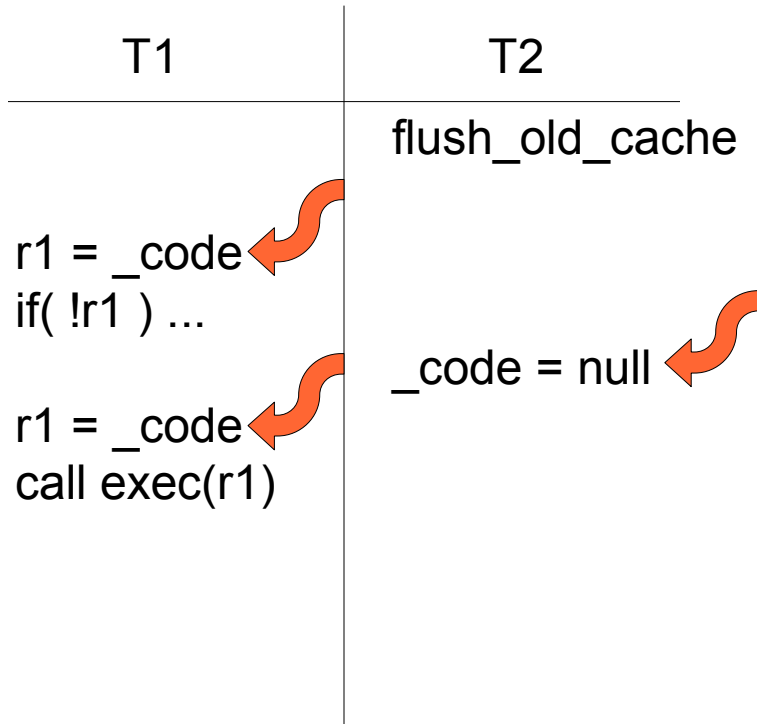
- Formally:
 - Two threads accessing the same memory
 - At least one is writing
 - **And no language-level ordering**
- Informally:
 - Broken attempt to use more CPUs
 - (but can happen with 1 CPU)
- Generally because 1 CPU is too slow
 - End of frequency scaling :-)
- Multi-core, dual-socket, big server, etc

Timeline of a Data Race



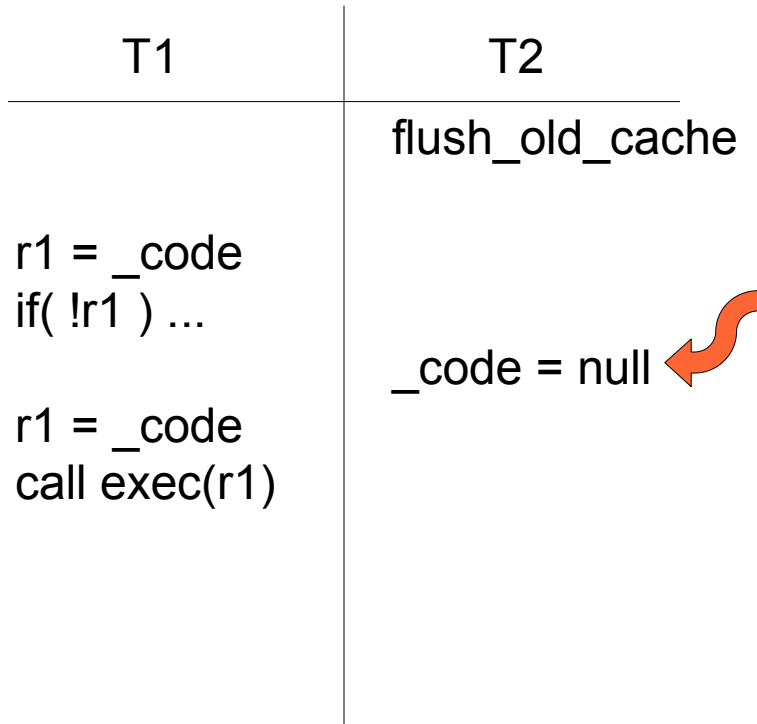
- **Two threads**

Timeline of a Data Race



- Two threads
- **Accessing same memory**

Timeline of a Data Race

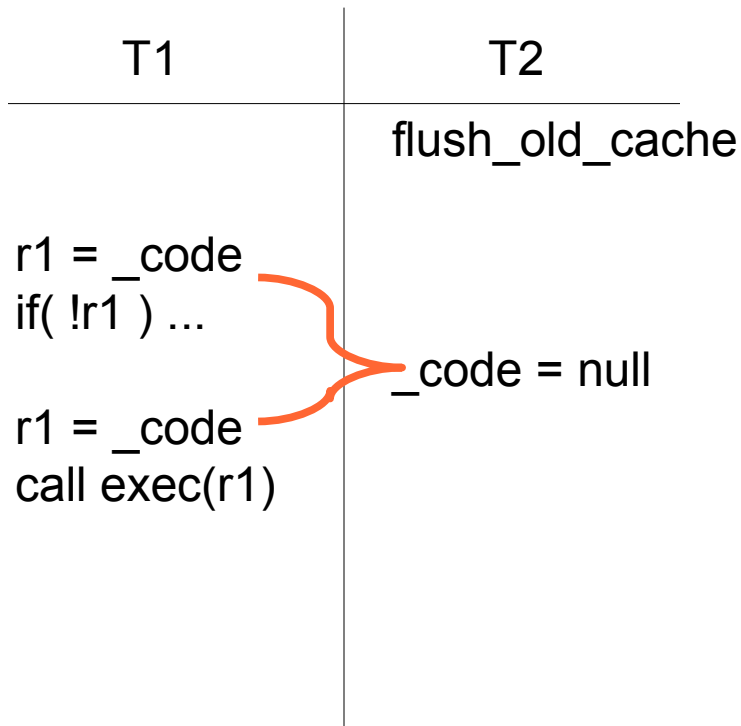


- Two threads
- Accessing same memory
- **At least one is writing**

Timeline of a Data Race



www.azulsystems.com



- Two threads
- Accessing same memory
- At least one is writing
- **No language-level ordering**

Timeline of a Data Race



T1	T2
	flush_old_cache
r1 = _code if(!r1) ...	
r1 = _code call exec(r1)	_code = null

- OK if:
 - Write before 1st Read OR
 - Write after 2nd Read
- Broken if in-between
- Pot-Luck based on OS thread schedule
- Crash rarely in testing
- More context switches under heavy load
- Crash routine in production

Agenda



www.azulsystems.com

- (not very) Formal Stuff
- **Reordering Memory – A peek under the hood**
- Common Data Races
- Finally(!) Some Debugging Techniques
- Summary

Reordering Memory Ops




T1	T2
<code>_datum = stuff</code>	
<code>_flag = true</code>	
	<code>r1 = _flag</code> <code>if(!r1) ...</code>
	<code>r2 = _datum</code>

- Writing 2 fields
- Can T2 see stale `_datum`?
- Yes!

Reordering Memory Ops




T1	T2
<code>_datum = stuff</code>	<code>r2 = _datum</code> 
<code>_flag = true</code>	<code>r1 = _flag</code> <code>if(!r1) ...</code>

- Writing 2 fields
- Can T2 see stale `_datum`?
- Yes!
- **Compiler can reorder**
 - Standard faire for -O
- Java: make `_flag` volatile
- C/C++: dicier
 - no C memory model *yet*

Reordering Memory Ops



T1	T2
	r1 = _flag 
	if(!r1) ... // predict
_datum = stuff	r2 = _datum
_flag = true	
	...r1 true
	...so keep r2


- Writing 2 fields
- Can T2 see stale _datum?
- Yes!
- **Hardware can reorder**
- Load _flag misses cache
- Predict r1==true
- Speculatively load _datum early, hits cache
- _flag comes back true
- Keep speculative _datum


Reordering Memory Ops



T1

T2

 `_datum = stuff`
`membar st-st`
`_flag = true`

`r1 = _flag`
`if(!r1) ...`
`membar ld-ld` 
`r2 = _datum`

- Writing 2 fields
- Can T2 see stale `_datum`?
- No!
- **Need store-side ordering**
- **Need load-side ordering**
- Included in Java **volatile**

Reordering Memory Ops





T1	T2
<pre>_datum = stuff membar st-st _flag = true</pre>	<pre>r1 = _flag if(!r1) ... // predict r2 = _datum ...r1 true ...so keep r2</pre>



- Writing 2 fields
- Can T2 see stale _datum?
- Yes!
- **Missing load-ordering**
- Read of _flag misses
- Predict branch
- Fetch _datum early
- Confirm good branch

Reordering Memory Ops

T1	T2
<code>_datum = stuff</code>	
<code>_flag = true</code>	
 ...stuff actually ...visible	<code>r1 = _flag</code> <code>if(!r1) ...</code> <code>membar ld-ld</code> <code>r2 = _datum</code> 

- Writing 2 fields
- Can T2 see stale `_datum`?
- Yes!
- **Missing store ordering**
- Write of `_datum` misses
- Write of `_flag` hits cache
- T2 reads `_flag`
- T2 reads stale `_datum`

Agenda



www.azulsystems.com

- (not very) Formal Stuff
- Reordering Memory – A peek under the hood
- **Common Data Races**
- Finally(!) Some Debugging Techniques
- Summary

Common Data Races



- My experiences only
- Double-read with write in the middle:

```
if( _p != null )           _p = null;
    ... _p._fld...
```

- ...and it's usually a null write
- Two writes with reads in the middle:

```
_size++;
_array=new[_size];           ..._array[_size-1]...
```


- Double Checked Locking:

```
if( _global == null )
    synchronized(x)
        if( _global == null )
            _global = new;
```

Double Checked Locking



T1	T2
<pre>r1 = new _global = r1</pre>	<pre>r1 = _global if(!r1) ... r2 = r1._fld</pre>

 `r1._fld = stuff`
`membar st-st`

- Initializing global singleton
- Can T2 see stale `_fld`?
- Yes!
- **Misplaced store-ordering**
- Unlock puts barrier AFTER both writes
 - Not between
- Actually missing load-ordering as well
 - True data dependency
 - Only IA64 might reorder?

More On Double-Read



www.azulsystems.com

- `if(_p != null) ..._p._fld...`
- Compiler likes to CSE together
 - C: Crashes in debug, not product
 - Java: Crashes before high-opt JIT kicks in
- Crashes when context-switch between reads
- i.e., just as heavy load hits system
- If you survive startup, **might** last a long time
- Bug can persist for years
 - Plenty of personal experience here...

Getting Clever w/HashMap



www.azulsystems.com

- Using a collection unsafely, and catching NPE
 - But not catching rarer AIOOB
 - Bit a customer AND in-house engineer
- Idea: HashMap w/single writer, many readers
 - No locking needed since 1 writer
 - Readers sometimes see 1/2 of 'put'
 - Throw NPE occasionally; catch & retry
- Writer is mid-resize, reader hashes to larger table
 - But does lookup on smaller table, Bang!
- Reader calls size(), size() calls resize, Bang!

Agenda



www.azulsystems.com

- (not very) Formal Stuff
- Reordering Memory – A peek under the hood
- Common Data Races
- **Finally(!) Some Debugging Techniques**
- Summary

Visual Inspection



www.azulsystems.com

- Easy to get started with
- Works on core files; after the fact
- Very slow per LOC
- Just obtaining the code is often a problem
- Sometimes can make a more directed search
 - e.g., Stack trace points out where somebody failed
 - Play mental Sherlock Holmes w/self
- Does not scale
- Requires Memory Model expert, domain expert

Visual Inspection



www.azulsystems.com

- Biggest Flaw: Not Knowing The Players
- Maintainer cannot name shared variables
 - Or which threads can access them
 - Or when they are allowed to touch
- Sometimes suffices to Make Access Explicit
- Avoiding Double-Read:
 - Often requires changing accessor patterns
 - Cache in a local variable
 - Return flag & value in 1 shot

Visual Inspection



www.azulsystems.com

- 2nd Biggest Flaw: forgetting “The Cycle”
- Concurrent code does: {start, do, end}
- Inspect: two threads both doing {start, do, end}
- But code in a cycle!
 - ...start, do, end, stuff, start, do, end, stuff...
- Inspect: two threads both doing {end, stuff, start}
- Inspect: {start, do, end} vs {end, stuff, start}
 - (chasing each others' tail)

- Printing / Logging / Events
 - “Make noise” at each read/write of shared variable
 - Inspect trace after crash
 - Serialized results...
 - ...but blocks; changes timing, hides data-race bugs
 - Also OS can buffer per-thread; WYSI **not** WYG
- Per-Thread Printing
 - Write “event tokens” (enums) to per-thread ring-buffer
 - No complex String creation, no object creation
 - Much less overhead & no blocking (ring buffer!)
 - Less likely to hide bug

It Takes Two to Tangle...



www.azulsystems.com

- Want: Who Dun It and When Dey Did It
- Per-Thread Printing w/TimeStamp
 - Slap a `System.nanoTime` in the per-thread event buffer
- Post-process the crash
 - Sort all ring-buffers by `.nanoTime`
 - Print a time-line just before the crash
 - AND after the crash
 - 99% chance the “guilty thread” stands out
- Rather heavy-weight technique
 - Need to know where to target it

Agenda



www.azulsystems.com

- (not very) Formal Stuff
- Reordering Memory – A peek under the hood
- Common Data Races
- Finally(!) Some Debugging Techniques
- **Summary**

Writing Data Races



www.azulsystems.com

- Often hidden by Good Programming Practice
- Already solving a Large, Complex Problem
- Using abstraction, accessors
 - Giving meaning to memory access
 - In context of L.C.P.
- Need more speed
- So introduce Concurrency, Threads
- Patterns like Double-Checked Locking
 - Or double-read

The Pitfall



www.azulsystems.com

- End up adding Concurrency to L.C.P.
- Fail to recognize Concurrency as a (subtle) Complex Problem
- Needs its own wrappers, access control
- Interviews w/Data-Race Victims:
 - Often they cannot name the shared vars
 - Don't know which thread can touch what
 - ... or when
- Fix starts with Gathering this info and asserting Access Control over shared vars

Summary



www.azulsystems.com

- Other techniques
 - Formal proofs? Still not ready for prime-time
 - Although hardware designers make it work
- Statistical
 - I get X fails/month; what happens that often?
- No good answers yet
- Best answers so far:
 - Don't Get Clever w/Concurrency
 - Document, Document, Document

<http://www.azulsystems.com/blogs/cliff>

***#1 Platform for
Business Critical Java™***

WWW.AZULSYSTEMS.COM

.....THE ERA OF UNBOUND COMPUTE IS NOW.....

Thank You

