# A dozen years of Memcheck: Looking backwards and looking forwards

Julian Seward, jseward@acm.org

4 February 2017. FOSDEM. Brussels.

# Overview

What it does

Some history

A sketch of the design

Challenges

Looking forwards: relevance

Looking forwards: opportunities

# What it does

A process-level virtual machine:

- Observes all memory accesses AND all malloc/free calls
- Verifies each access is allowable
- Verifies that undefinedness will not cause observable behaviour
- As a side effect, checks for memory leaks

```
char* p = malloc(10);   ... p[10] ...          error: out of bounds read

char* p = malloc(10);   free(p);   ... p[5] ...   error: reading freed

char p[10];   ... if (p[5] == 'x') {..}          error: branch on undefined

char* p = malloc(10);   p = NULL;                error: lost block
```

# History 1

Motivation

- mid 90s -- data compression hacking in C
- late 90s -- early versions of KDE on sparc-solaris

I wrote a machine-code interpreter

- “Heimdall”, 1999
- Infeasibly slow, inflexible, and uncool

“Borrowed” a bunch of interesting ideas

- from the Glasgow Haskell Compiler
  - simple, typed, checkable intermediate representation
- from LCC, the Fraser-Hanson C compiler book
  - simple code generators
- JITs -- the cool new thing

# History 2

2001: initial development work

- Hacked up Valgrind 1.0
- Just in time for KDE 3.0
- Excellent interaction with KDE3 developers

2001-2004

- Early key developers joined in
- Much rework, redesign
- New JIT framework, VEX
- Proper tool framework interface
- New tools (cachegrind, massif, callgrind)
- First new port, x86_64-linux

# History 3

Initial design goals

- Must not be a toy.  Must scale.
  to KDE, Gnome, OpenOffice, later Firefox
- Low false positive rate
  preferably zero
- Be easy to use
  recompile “-g -O” and go
- To have influence
  change expectations of C/C++ developers

How?

- Build the worst thing that actually works
  improve it on demand
  daily interactions with end users (C++ devs)
- Assertions are always enabled
  budget 5% for assertions
- Threading was a terrible hack -- our own `libpthread.so`
  Nowadays it's a slightly less terrible hack

# History 4

Some experiments on the way

- Tracking definedness
  1 bit per byte or 8 bits per byte?
- Shadow memory
  9 bits per byte or 2.epsilon bits per byte?
- When to report undefinedness
  early (on read) or later (on branch, addr use) ?
- Copy-annotate or disassemble-resynthesise?
  is not memcheck-specific
- Can we make it multithreaded?
  not really

What did we learn?

- Porting to new architectures is surprisingly easy (still a lot of work)
- Porting to new OSs is surprisingly difficult (OSX, AIX)
- Users will do all manner of crazy stuff

# History 5

What did the users think?

- Used it widely (I assume)
- Integrated it into their test systems
- Reported many many bugs
- Provided loads of good suggestions, patches, support
- Stressed it in numerous bizarre ways
  - User-Mode Linux
  - Big (BIG!) processes
  - 10s of millions of lines of C++

They also

- didn't complain about the slowness
- didn't believe what it was telling them
- had difficulty finding undefinedness sources

  hence `--track-origins=yes`
- expected zero complaints on **all** valid programs

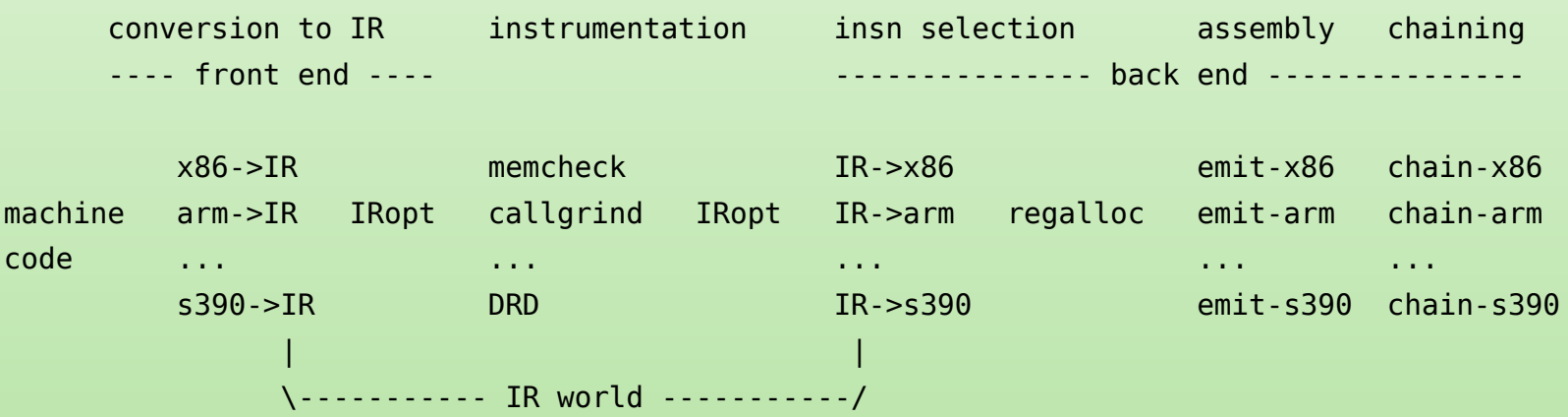
# A sketch of the design 1

Memcheck, a tool built on the Valgrind framework

Framework provides

- Kernel interface virtualisation
- Debuginfo reading
- Error management
- Code JITting and management
- a GDB server

JIT pipeline:



# A sketch of the design 2

| Original code | In IR | Instrumentation IR |
|---|---|---|
| `subq %rax, %rdi` | `tL = GET(328)`<br>`tR = GET(416)`<br>`tRes = Sub64(tL, tR)`<br>`PUT(416) = tRes` | `qL = GET(1328)`<br>`qR = GET(1416)`<br>`qRes = Left64(UifU64(qL, qR))`<br>`PUT(1416) = qR` |
| `jz 0x1234` | `ExitIf CmpEQ64(tL, tR)`<br>`0x1234` | `CallIf (CmpNEZ64(qRes))`<br>`report_error()` |
| `movq (%rcx), %rdx` | `tA = GET(360)`<br>`tD = LOAD64le(tA)`<br><br><br>`PUT(368) = tD` | `qA = GET(1360)`<br>`CallIf (CmpNEZ64(qA))`<br>`report_error()`<br>`qD = Call helper_LOAD64le(tA)`<br>`PUT(1368) = qD` |

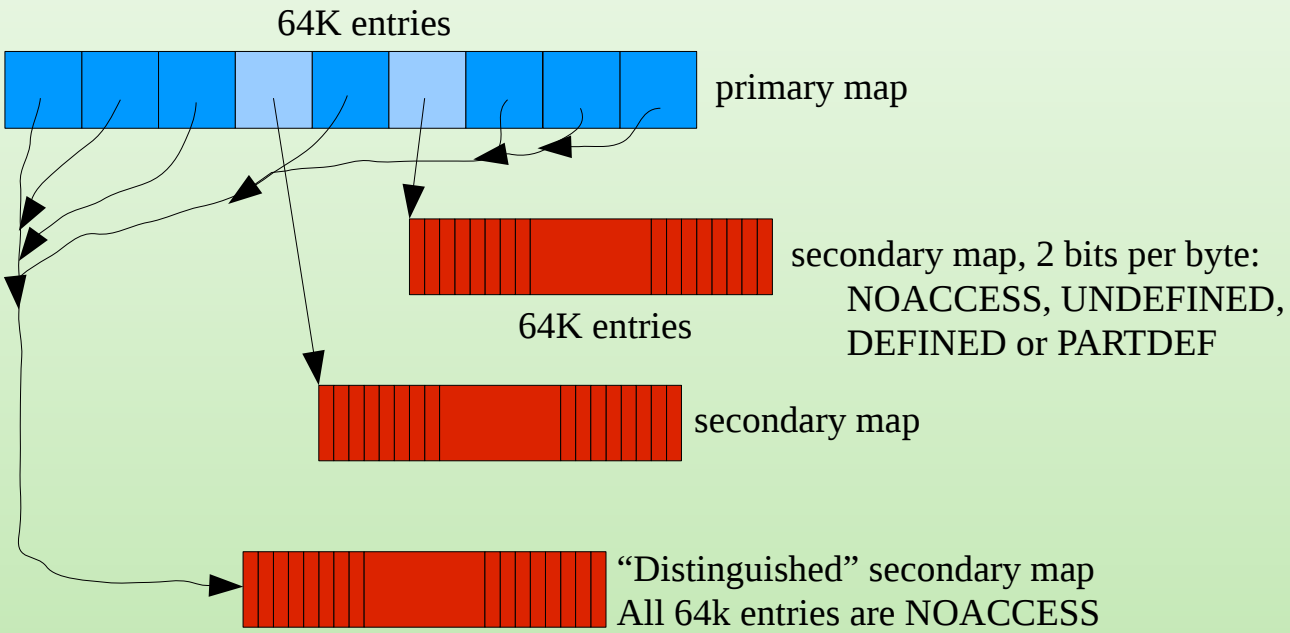
tXX: 64 bit IR temps holding original values.

qXX: 64 bit IR temps holding definedness bits. 0 = defined, 1 = undefined.

# A sketch of the design 3

For a 32-bit address space:

- Divide address space into 64KB chunks -- Secondary Maps
- Have a 64K entry Primary Map




- Distinguished secondary map makes reads faster
- No need for a NULL check

- 64 bit is a horrible kludge (but it's fast!)

# Challenges that have emerged

Problems with existing functionality

- Compiler optimisations (gcc 5.x, clang 3.7.x)

  `if (A && B) ==> if (B && A)` if A is false whenever B is undefined

  => correct program branches on undef'd
- Load-Linked / Store-Conditional -- can cause looping
- Hardware transactional memory -- how to handle?
- 64 bit shadow memory -- current scheme a complex kludge
- Threading -- currently sequentialises threads
- Floating point simulation is inaccurate and incomplete

Missing functionality

- No stack bounds checks
- No help for GPU programming

Expected but non-problems

- Autovectorisation
- Kernel interface changes

# Looking forwards: relevance 1

2000s, 2010s: rise of the Undefined Behaviour Checkers

Now we have:

- Memcheck: heap invalid access, uninit value uses
- ASan: heap and stack invalid access
- UBSan: other C++ undef checking, incl some definedness
- TSan: invalid cross-thread accesses
- and many others

These convert an undefined behavior detection problem into a code coverage problem
to paraphrase John Regehr, Niko Matsakis

Loads of hassle!  And still incomplete ..

# Looking forwards: relevance 2

Dynamic analysis is forever incomplete.  Can't we do better?

Static analysis of C/C++

- Some checkers exist as FOSS, but nothing comparable to the commercial state of the art
- C++ still basically sucks for safety, and always will

  Immense complexity / effort and look where we are now ..

Use a systems programming language designed for safety

- Rust!  Yay!  Rust!
- A modern imperative language with control of mutability
- “Safety, speed, concurrency: choose all 3”
- Interworks with C/C++
- http://www.rust-lang.org

# Meanwhile, back at the ranch ..

Memcheck continues to have a unique niche

- Run anything, anywhere, on Linux
- High precision definedness tracking
- Zero effort to use

What can we do?

- Parallelise: new shadow memory scheme
  - loss of perf?
  - initial studies (Julian) and initial hacking (Philippe)
- Increase single thread performance
  - new JIT framework
  - Check multiple accesses together
  - come back at 15.30 today!
- False positives
  - may be helped by a new JIT
  - more accurate definedness instrumentation

# So, in conclusion ..

A big Thank You to all our users, developers, supporters over the years who have made the tool suite so successful

We need your engineering support for the next 12 years!

Questions?

Shameless ad: VEX talk at 15.30!