Detecting Heisenbugs

the case for dynamic program analysis

Andrei Terechko

Vector Fabrics

Andrei Terechko

andrei@vectorfabrics.com

+31 40 8200960

Eindhoven, The Netherlands

Outline

2. Dynamic analysis tools to find them

1. Software bugs that cost billions

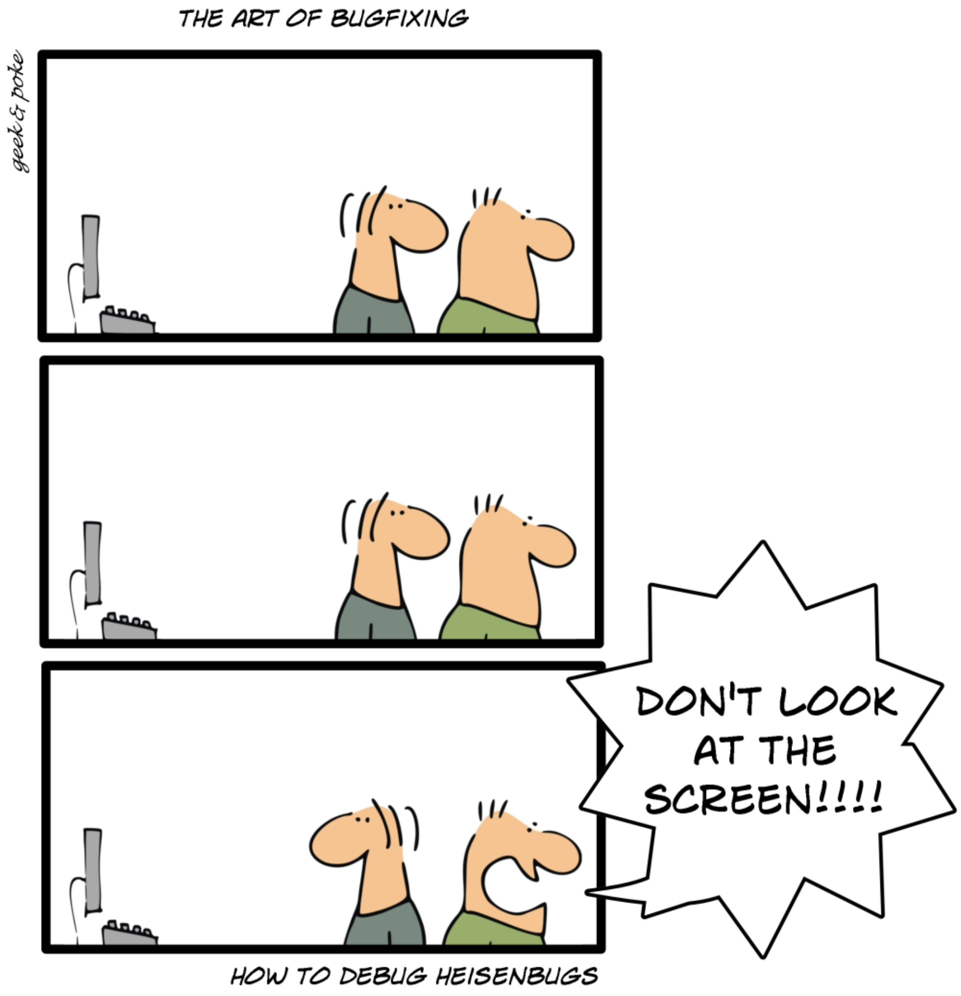
Anything wrong here?

```
int main()
{
  // create an empty bucket
  std::set<int> bucket;

  // Insert '3' in the bucket from background thread
  std::thread t([&](){bucket.insert(3);});

  // Check if '5' is in the bucket
  bool has5 = bucket.find(5) != bucket.cend();

  // Wait for background task
  t.join();

  return has5;
}
```

Vector Fabrics

Heisenbugs are critical

- ***Wikipedia on Heisenbug**:*
  a software bug
  that seems to disappear
  or alter its behavior
  when one attempts to study it
- hard to catch in development
- crash systems in production



Vector Fabrics

How about Heisenbugs in real-life?

Heartbleed OpenSSL bug

Costs $1B+
Due to an
uninitialized memory read

Toyota Prius recall of 8M cars

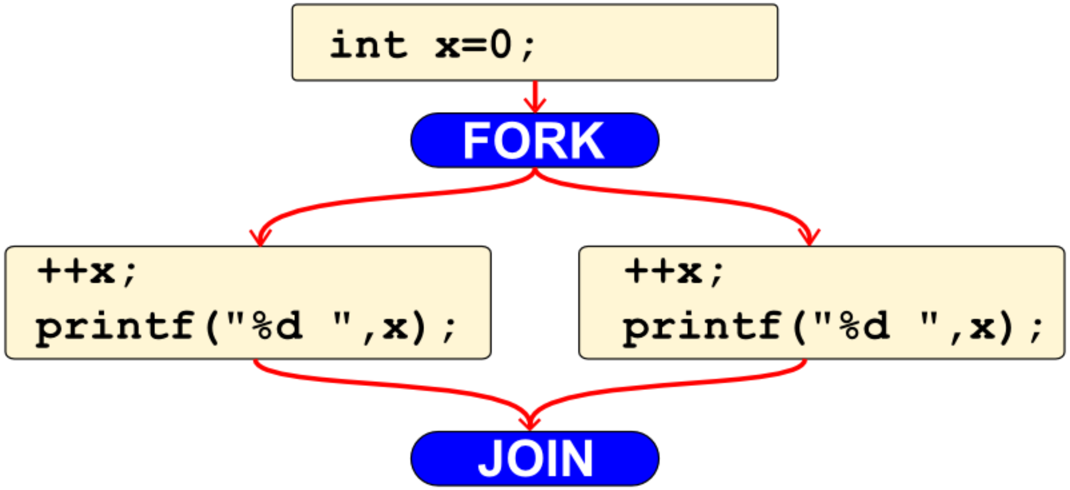
Costs: $2B
due to a data race

Google Chrome: 8.3 million lines of code

Joint Strike Fighter: 24+ million lines of code

Radio & navigation in Mercedes S-class: 20 million lines of code

Multi-threading – rise of the Heisenbugs

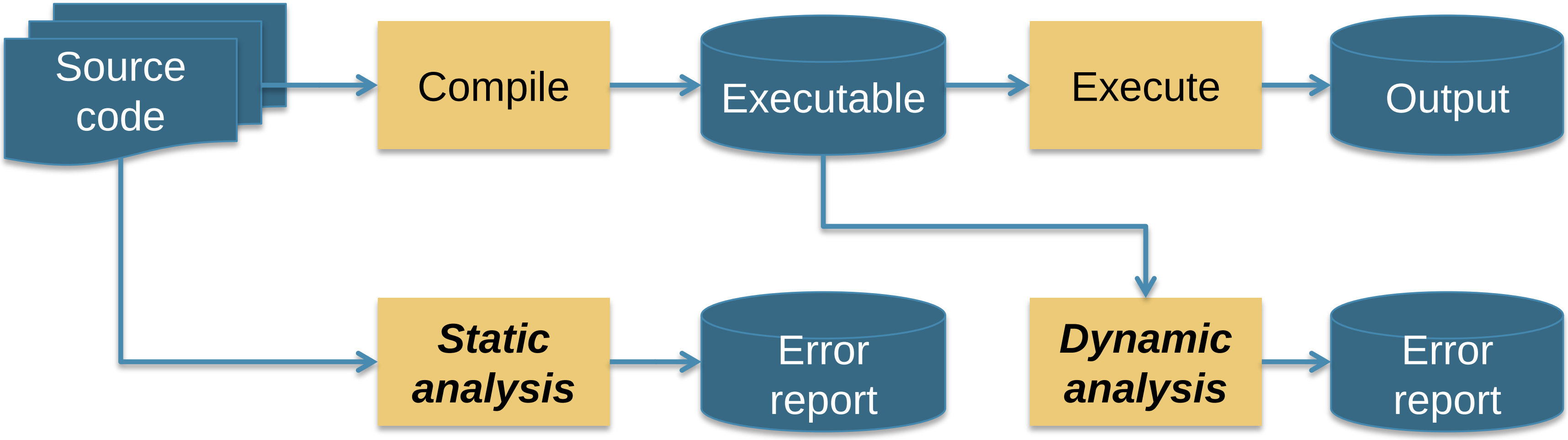


Quiz: without further synchronization, which are valid print-outs according to C (and Java) semantics?

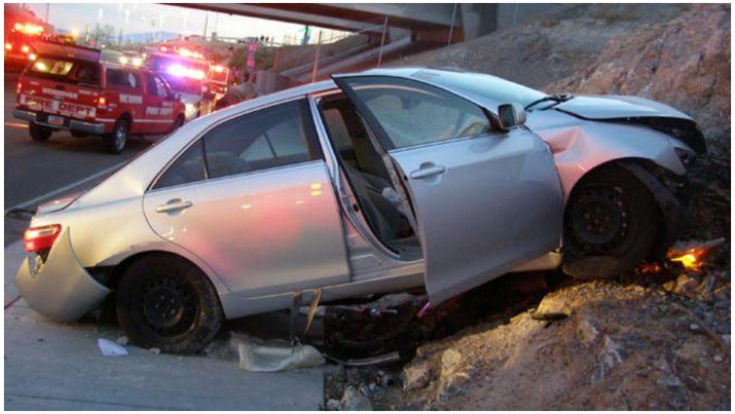
a) 1 1
b) 1 2
c) 2 1
d) 2 2

Vector Fabrics

Program analysis tools to the rescue



What about static analysis tools?



"The team's experience is **that there is no single analysis technique today that can reliably intercept all vulnerabilities**, but that it is strongly recommended to deploy a range of different leading tools. Each tool used can excel at a different aspect of **static analysis**, which results in remarkably little overlap in the set of warnings that is produced."

– *NASA assessment report on Toyota's unintended acceleration failure*

www.nhtsa.gov/staticfiles/nvs/pdf/NASA_FR_Appendix_A_Software.pdf

Vector Fabrics

Statically analyzable?

```
/* obfuscated and simplified yet production code */

char array[3][52];

/* function argument from another compilation unit */
int func_a(int indexB)
{
  for (i = 0; i < n; i++) {          /* complex control */
    int indexA = func_a(d->g);       /* recursive functions */
    int filter = *e++ + *f++;        /* pointer arithmetic */
    if (filter < h && indexB < g)    /* dynamic control */
        z = array[indexA][indexB];   /* ERROR! Out-of-bound. */
...
```


Vector Fabrics

Vector Fabrics' activities

PRODUCTS

Pareon makes your C/C++ code run faster

PAREON OVERVIEW >

Multicore programming tools based on dynamic analysis

CONSULTANCY

Software optimization

CHECK OUT OUR EXPERTISE >

Consultancy services

TRAINING

Multicore programming training

SEE OUR TRAINING PROGRAM >

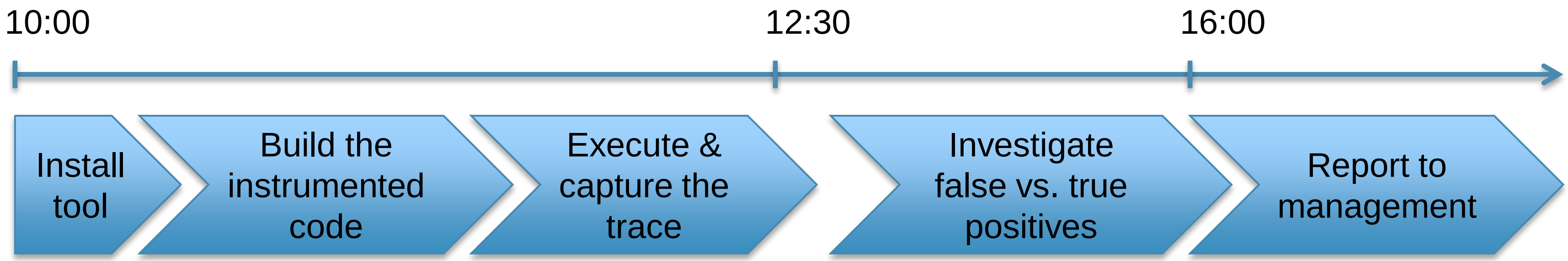
Training in-house and on-site

Vector Fabrics

Dynamic analysis – opportunities

- Many true positives
  - Analyze code that is inherently dynamic: dynamic loop bounds, (C++) function pointers, virtual functions, recursion, etc.
- Analyze code with full-program scope
  - Find many more true positives: across functions, files, (shared) libraries
  - False negatives when code paths are not executed
- Find errors in code that is timing-dependent
  - Data races from interrupts, threads, signals – even if these do not occur in the test!
  - Potential deadlocks
- Capture code coverage & hotspot profiles

Dynamic analysis – sales challenge



- Just one day to prove you can find critical bugs in customer code
- Challenges in install, build, execute
  - No longer supported operating system and tools
  - Highly custom build system not known by the customer
  - Customer does not know all the required libraries
  - Illegal code constructs accepted by just one specific compiler
  - Highly custom target platform with custom compiler extensions
  - Custom way to get trace out of the target platform (RS-232, JTAG, ??)
- Challenges in interpreting the results
  - Even belt-stop bugs are regarded as a “false positive”

Vector Fabrics

Dynamic analysis – technology challenge

```
/* uninitialized read that is not used */
typedef struct {
  int flag;
  int data;
} struct_s;

void fun(int flag) {
  struct_s s1;
  /* defines if data is properly set */
  s1.flag = flag;
  /* read of uninitialized data */
  struct_s s2 = s1;
```

```
/* buffer overflow in adjacent array */
int a[2] = {0, 1};
int b[2] = {2, 3};

/* what if i is 2? */
int foo(int i) {
  int *ptr = a;
  /* out-of-bound access in b[0] */
  return *(ptr + i);
```

Simple messages, easy to understand?

```
int *a;

int main(void)
{
  int b[2], i;

  for (i=0; i<=2; ++i)
    b[i] = i+1;

  a = malloc(4*sizeof(short));
  a[b[2]] = b[b[0]-2];

  return 0;
}
```

```
==6043==ERROR: AddressSanitizer: stack-buffer-overflow
on address 0x7fff135ca5a8 at pc 0x42d59b bp
0x7fff135ca4b0 sp 0x7fff135ca4a8
WRITE of size 4 at 0x7fff135ca5a8 thread T0
    #0 0x42d59a
(/home1/stefan/vf/code/demo/b.out+0x42d59a)
    #1 0x7f65a84ce76c (/lib/x86_64-linux-gnu/libc-
2.15.so+0x2176c)
    #2 0x42d25c
(/home1/stefan/vf/code/demo/b.out+0x42d25c)
Address 0x7fff135ca5a8 is located in stack of thread
T0 at offset 104 in frame
    #0 0x42d32f
(/home1/stefan/vf/code/demo/b.out+0x42d32f)
  This frame has 3 object(s):
    [32, 36) ''
    [96, 104) 'b'
    [160, 164) 'i'
HINT: this may be a false positive if your program
uses some custom stack unwind mechanism or swapcontext
      (longjmp and C++ exceptions *are* supported)
Shadow bytes around the buggy address:
  0x1000626b1460: 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000626b1470: 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000626b1480: 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000626b1490: 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000626b14a0: 00 00 00 00 00 00 00 00 f1 f1 f1 f1
=>0x1000626b14b0: f2 f2 f2 f2 00[f4]f4 f4 f2 f2 f2 f2
```

Tool development

Real-life environment

- Embedded targets
- Huge C/C++ code bases
- Memory leaks

- Real-world code
- Predictive data races & taint analysis
- Zero false positives

Sorry, not interested

- Small Java examples
- Memory leaks

- Small Java examples
- Predictive data races & taint analysis
- Zero false negatives?

Errors found w/o false positives

Vector Fabrics

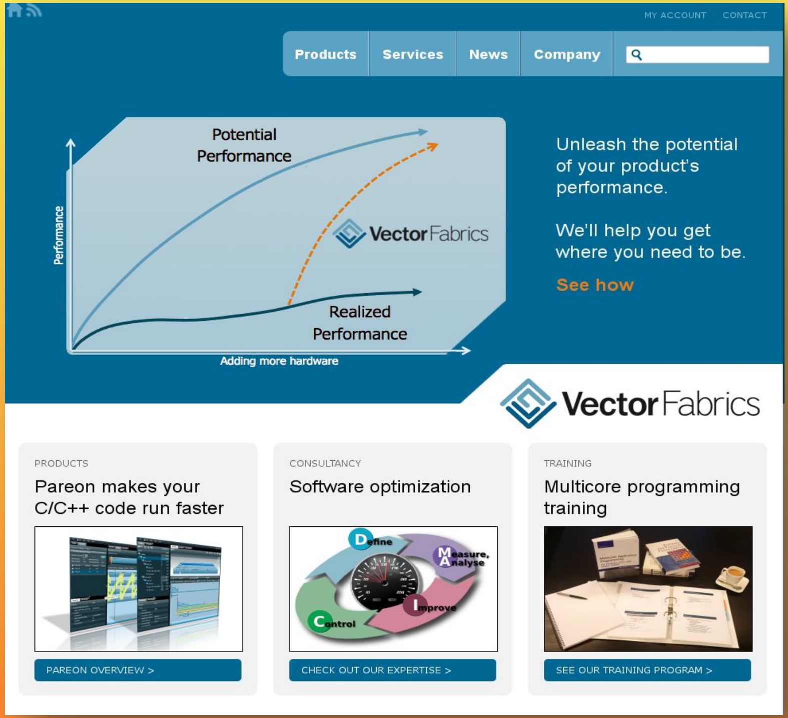
Take-away

- Software complexity grows to **huge systems** nobody can oversee
- **Multicore** will cause more recalls, power outages, and heartbleed
- **Dynamic analysis** is key to find errors in these dynamic systems

Dynamic analysis for C/C++ requires extensive “**plumbing**” to work on real-life code

and needs innovation in **scalable**, **pragmatic** tooling to detect real-life errors

Vector Fabrics



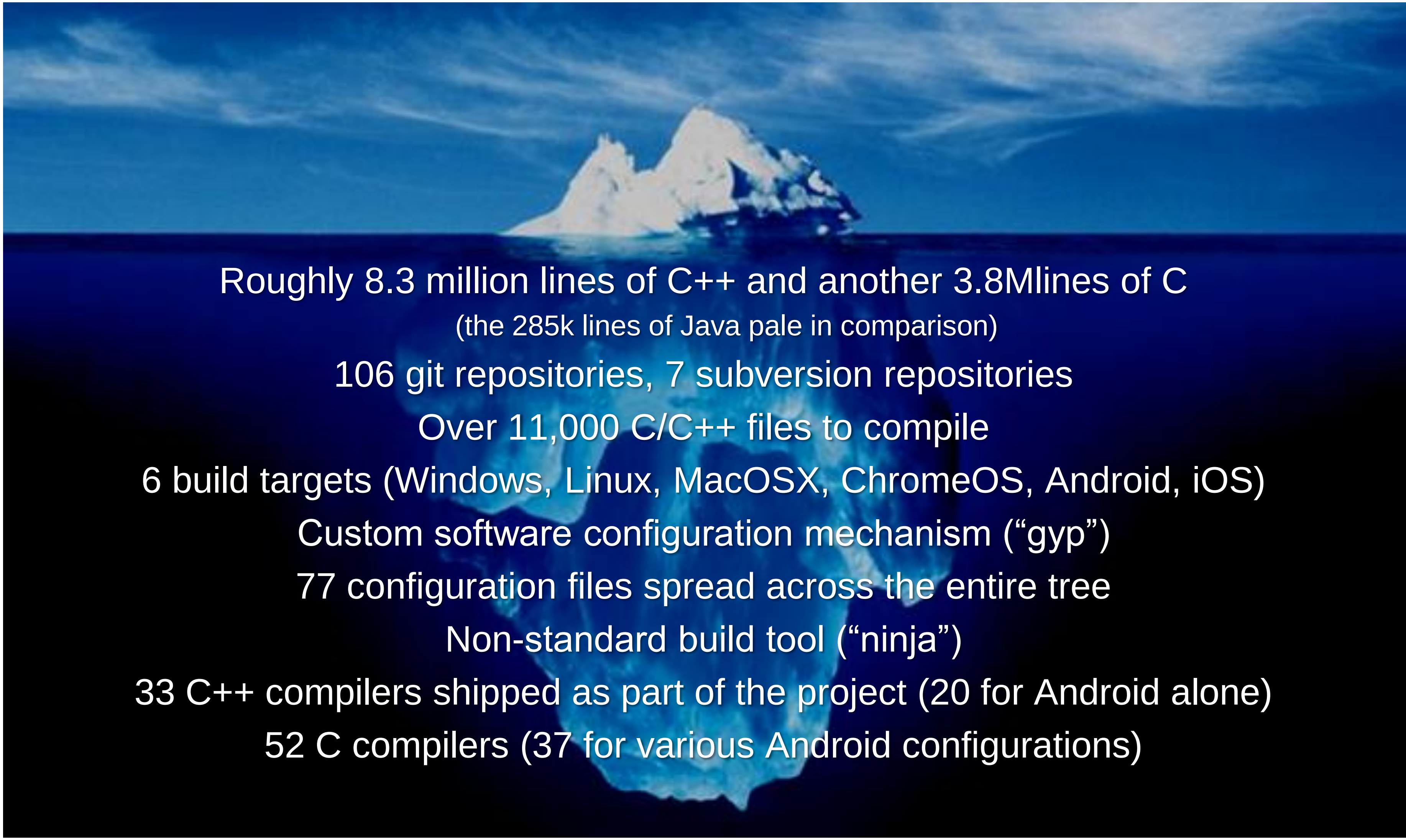
Optimize your software and find critical bugs

http://vectorfabrics.com

Vector Fabrics

Andrei Terechko
andrei@vectorfabrics.com
+31 40 8200960
Eindhoven, The Netherlands

Chromium – real-life build system



Heartbleed – typical dynamic problem

Uninitialized memory read in OpenSSL:

`p = s->s3->rrec.data[0]`

- to identify exactly to which object **`p`** is pointing,
- to see how the data flowed through the application to that object

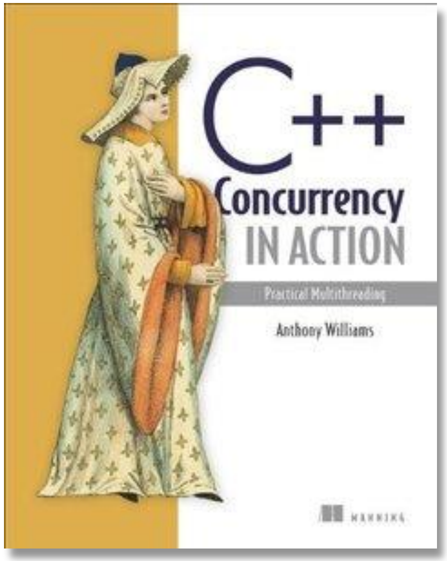
➔ Understand **`data`** originally came from a tainted source

The industry "solution" – admit defeat and apply a heuristic:

> "*We noticed that the tainted data was being converted via* **`n2s`***, a macro that performs byte swapping. […] This heuristic bypasses the complex control-flow and data-flow path that reaches this point in the program, and instead infers and tracks tainted data near the point where it is used.*"

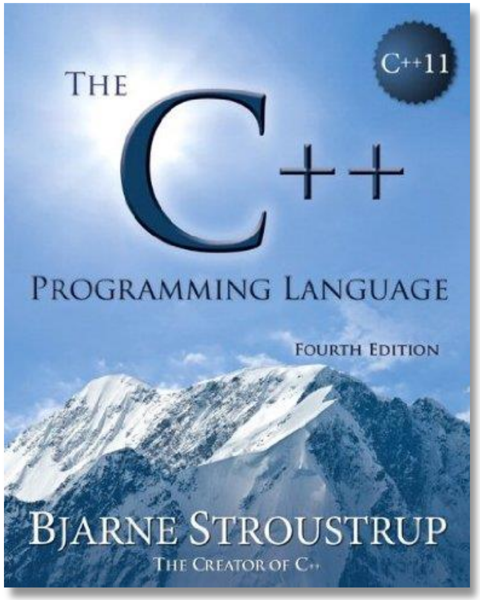
http://security.coverity.com/blog/2014/Apr/on-detecting-heartbleed-with-static-analysis.html

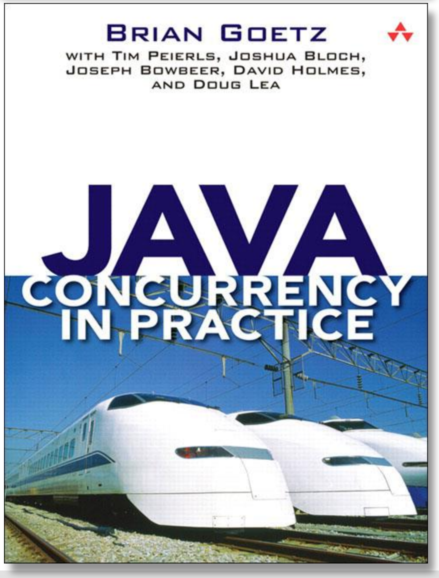
1300 pages just for C++?



- Provides good insight in C++ concurrency
- C++11 standardizes concurrency primitives
- Warns for ***many many*** subtle problems

  - The authorative description (4th edition)
  - Apparently requires 1300+ pages...





- Safe concurrency by defensive design
- Shows that Java shares many concurrency issues with C++

Vector Fabrics

No silver bullet

"A wide range of commercial static source-code-analysis tools is on the market, **each with slightly different strengths**. We found that running multiple analyzers over the same code can be very effective; there is **surprisingly little overlap** in the output from the various tools. This observation prompted us to run not just one but four different analyzers over all code as part of the nightly integration builds for the MSL mission."

Gerard J. Holzmann: Mars code. Commun. ACM 57(2):64–73 (2014)

Trivial program?

Global `int x = 0, y = 0;`

Launch four threads, namely:

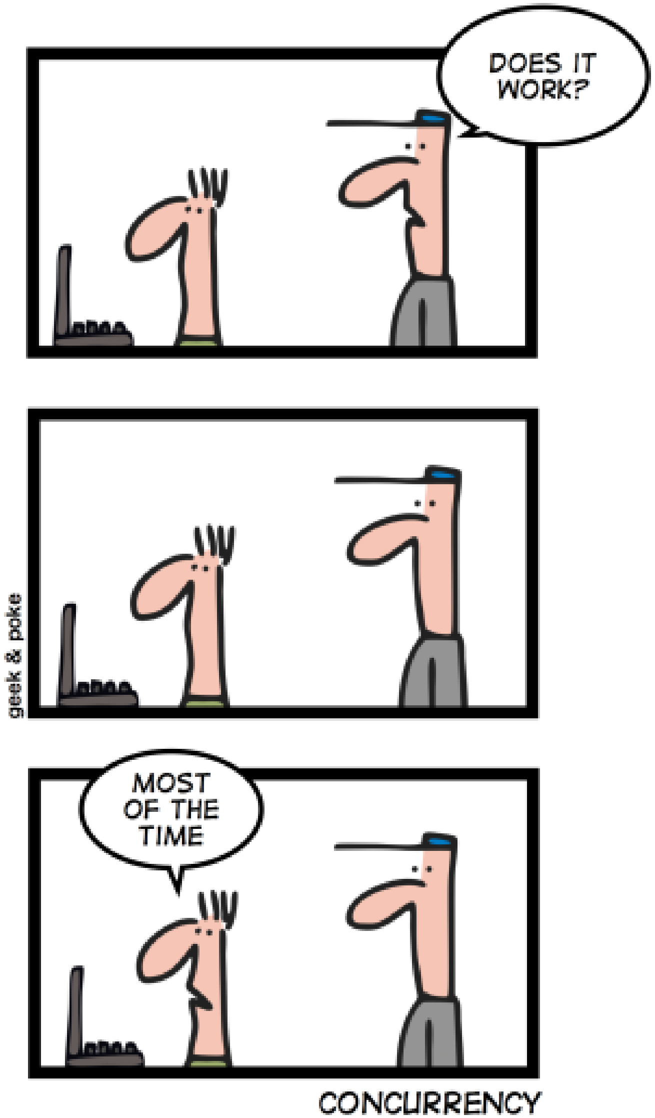
- Thread 1: `x = 1;`
- Thread 2: `y = 1;`
- Thread 3:

  ```
  if (x && !y) print("X first");
  ```

- Thread 4:

  ```
  if (y && !x) print("Y first");
  ```

Can this program execute both prints?



Vector Fabrics

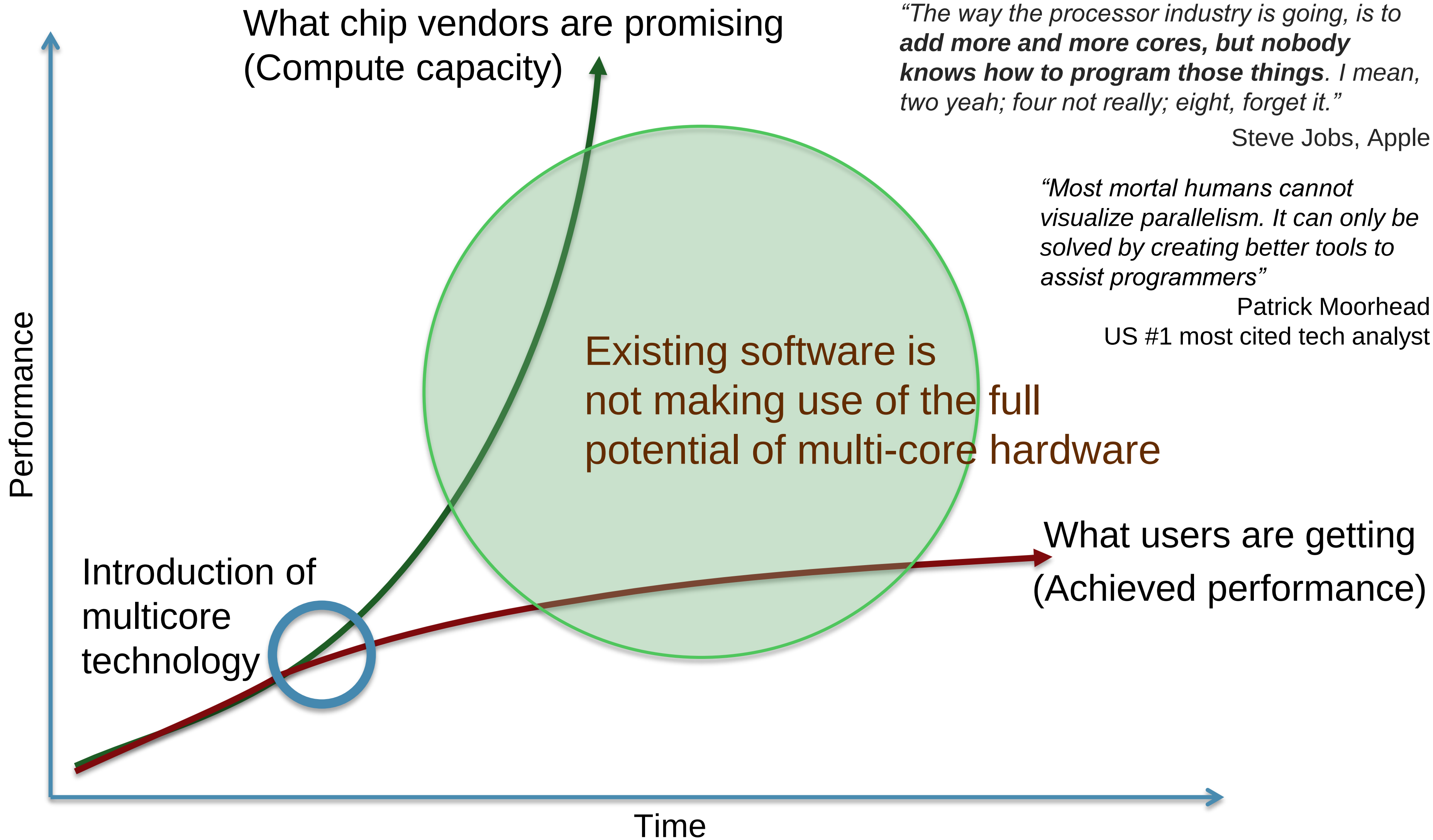
Multicore – Moore’s law vs. Amdahl’s law

“The way the processor industry is going, is to ***add more and more cores, but nobody knows how to program those things****. I mean, two yeah; four not really; eight, forget it.”*

Steve Jobs, Apple

“Most mortal humans cannot visualize parallelism. It can only be solved by creating better tools to assist programmers”

Patrick Moorhead
US #1 most cited tech analyst



Vector Fabrics

Mobile web browsers – browsermark

TOP 5 WEB PHONES

BROWSERMARK

SHARE:

#	OEM	Device	GPU CPU	Score
1	Apple	iPhone 5s	Apple A7 GPU Apple A7 Dual-core 1.3 GHz Cyclone	3621.12
2	Samsung	SM-G900V Galaxy S5 (Verizon)	Unknown Unknown	3546.70
3	Samsung	SM-G900F Galaxy S5	Adreno 330 Qualcomm Snapdragon 801 Quad-core 2.5 GHz Krait 400	3303.88
4	Samsung	Galaxy Note 3 SM-N900A (AT&T)	Adreno 330 Qualcomm Snapdragon 800 Quad-core 2.3 GHz Krait 400	3296.91
5	LG	Nexus 5	Adreno 330 Qualcomm Snapdragon 800 Quad-core 2.26 GHz Krait 400	3296.34

How can a dual-core iPhone outperform a quad-core Samsung S5?!

Vector Fabrics

Anything wrong here?

```
const string& pass( const string& s )
{
  return s;
}

int main()
{
  const string& s = pass("foo");
  return s == "bar";
}
```

The world goes multicore!

Galaxy S (2010)
single core
1 processor

Galaxy S2 (2011)
dual-core
2 processors

Galaxy S3 (2012)
quad-core
4 processors

Galaxy S4 (2013)
octa-core
8 processors

Intel processor with 61 processors
for servers

Cisco manycore (336 processors)
for software-defined networking

Vector Fabrics

Creating parallel programs is hard

Herb Sutter, ISO C++ standards committee, Microsoft:

“Everybody who learns concurrency thinks they understand it, ends up finding mysterious races they thought weren’t possible, and discovers that they didn’t actually understand it yet after all”

Edward A. Lee, EECS professor at U.C. Berkeley:

“Although threads seem to be a small step from sequential computation, in fact, they represent a huge step. They discard the most essential and appealing properties of sequential computation: understandability, predictability, and determinism.”

Vector Fabrics