Fault Localization and Profiling

The Story So Far …

- Quality assurance is critical to software engineering.
  - Static and dynamic QA approaches are common
- Defect reports are tracked from creation to resolution
- Some are assigned to developers for resolution (triage)
- How do we know which part of a program to change to repair a bug or improve a program?

One-Slide Summary

- A **debugger** helps to detect the source of a program error by single-stepping through the program and inspecting variable values.
- **Fault localization** is the task of identifying lines implicated in a bug. Humans are better at localizing some types of bugs than others.
- Automatic **tools** can help with the dynamic analyses of fault localization and profiling.
- Care must be taken when evaluating such tools (and their assumptions) for real-world use.

Outline

- Software Scales
- Manual Debuggers
- Human Study Results
- Automatic Tools
- Profilers
- Are Tools Helping?

A lot of code. A lot of defects.

Microsoft: 70 percent of all security bugs are memory safety issues

Percentage of memory safety issues has been hovering at 70 percent for the past 12 years.

By Catalin Cimpanu for Zero Day | February 11, 2019 -- 15:48 GMT (07:48 PST) | Topic: Security

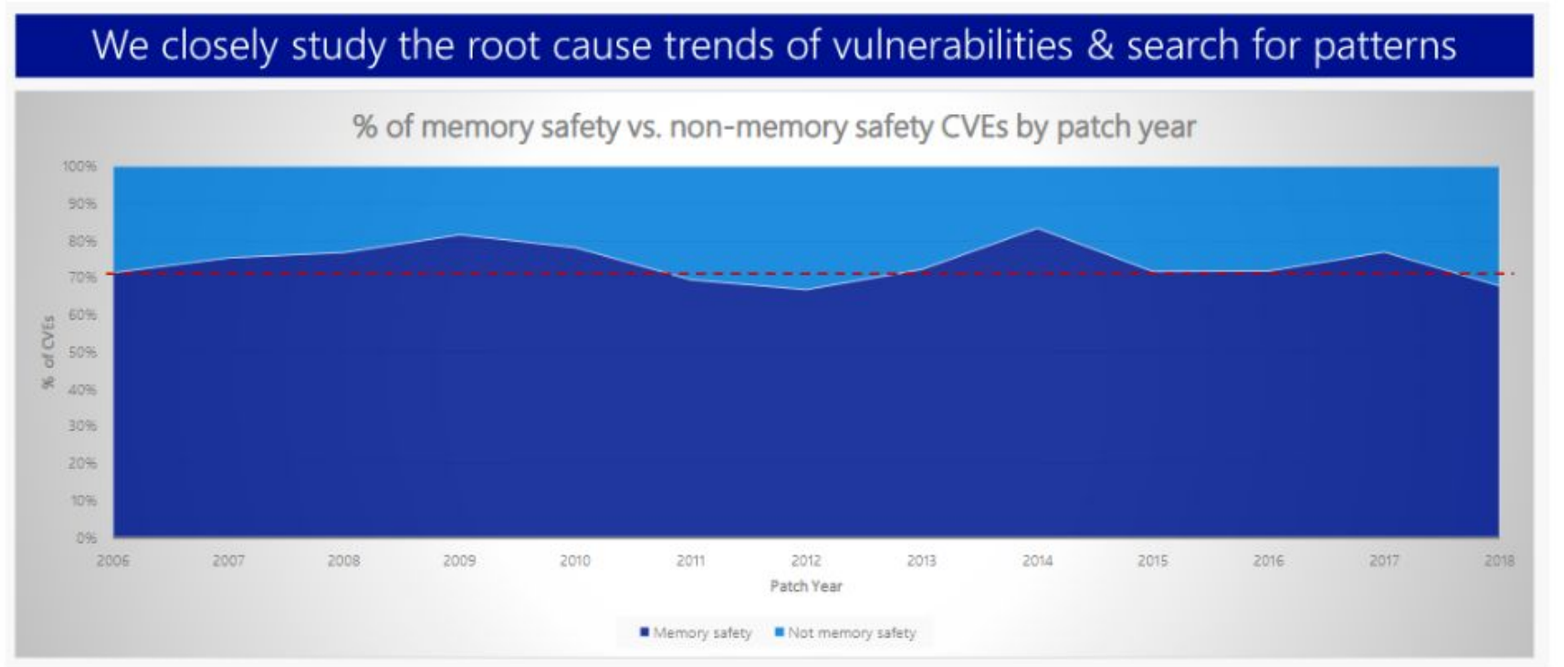


Image: Matt Miller

Around 70 percent of all the vulnerabilities in Microsoft products addressed through a security update each year are memory safety issues; a Microsoft engineer revealed last week at a security conference.

Which of these is photoshopped?

Bucket-Wheel Excavators

- Heaviest land vehicles
  - ~14,000 tons
    - (avg USA car: 2 tons)
  - Mobile strip-mining

Modern Software Is Huge

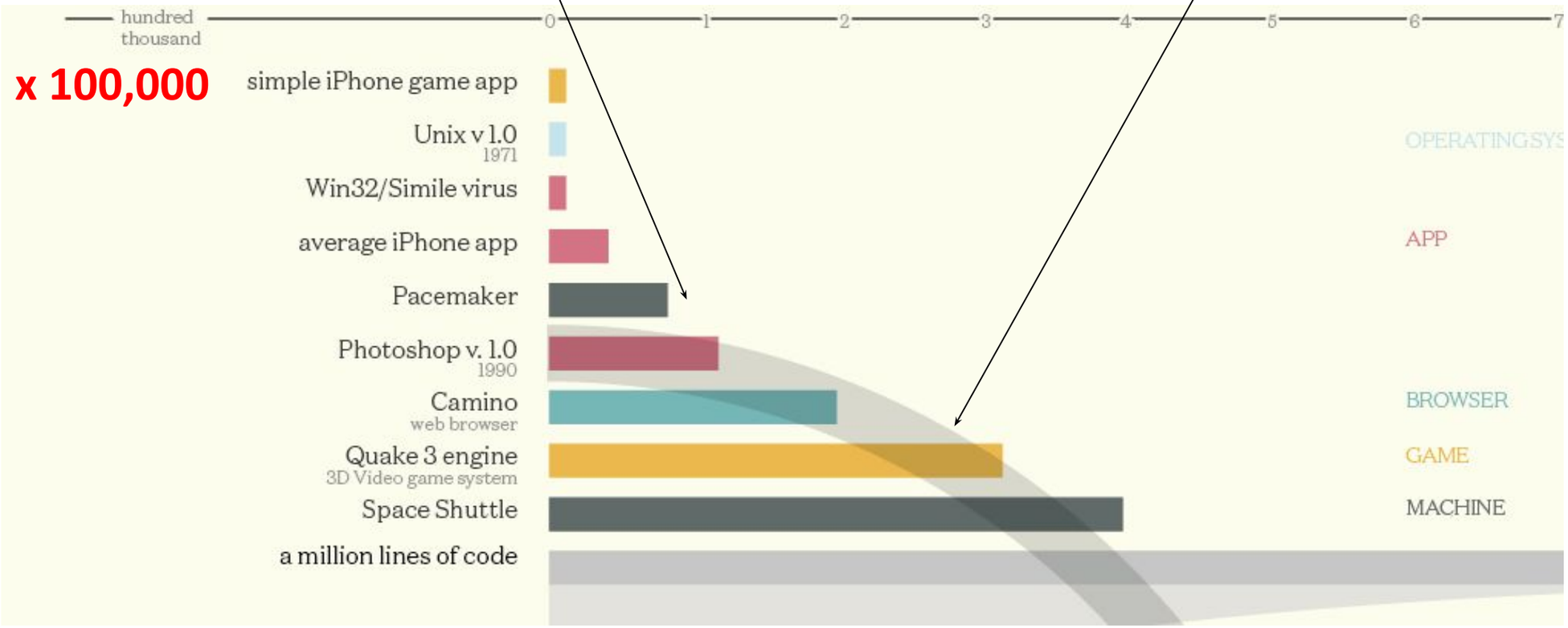
- “Space is big. Really big. You just won't believe how vastly, hugely, mind-bogglingly big it is. I mean, you may think it's a long way down the road to the chemist, but that's just peanuts to space.” – Douglas Adams
- Who cares?
  - Techniques developed based on smaller code bases simply do *not* apply or scale to larger code bases
    - Techniques from the 1980s or your habits from classes

- How many lines of code? Guess??
  - iPhone app
  - Facebook
  - Chrome/Firefox
  - Microsoft Office
  - Car Software
  - Space Shuttle

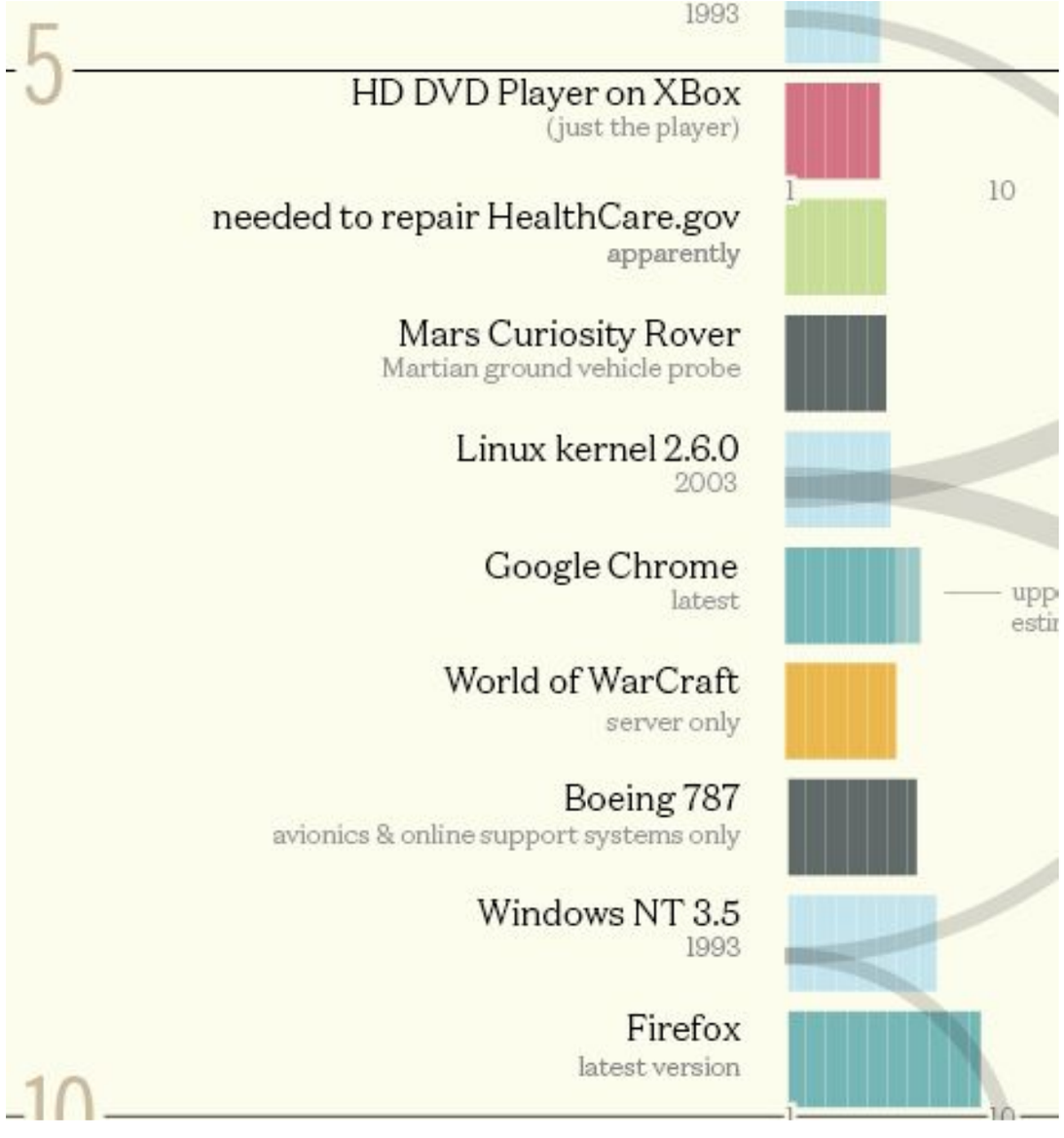
Example Programs: < 1MLOC

- libpng: 85,000

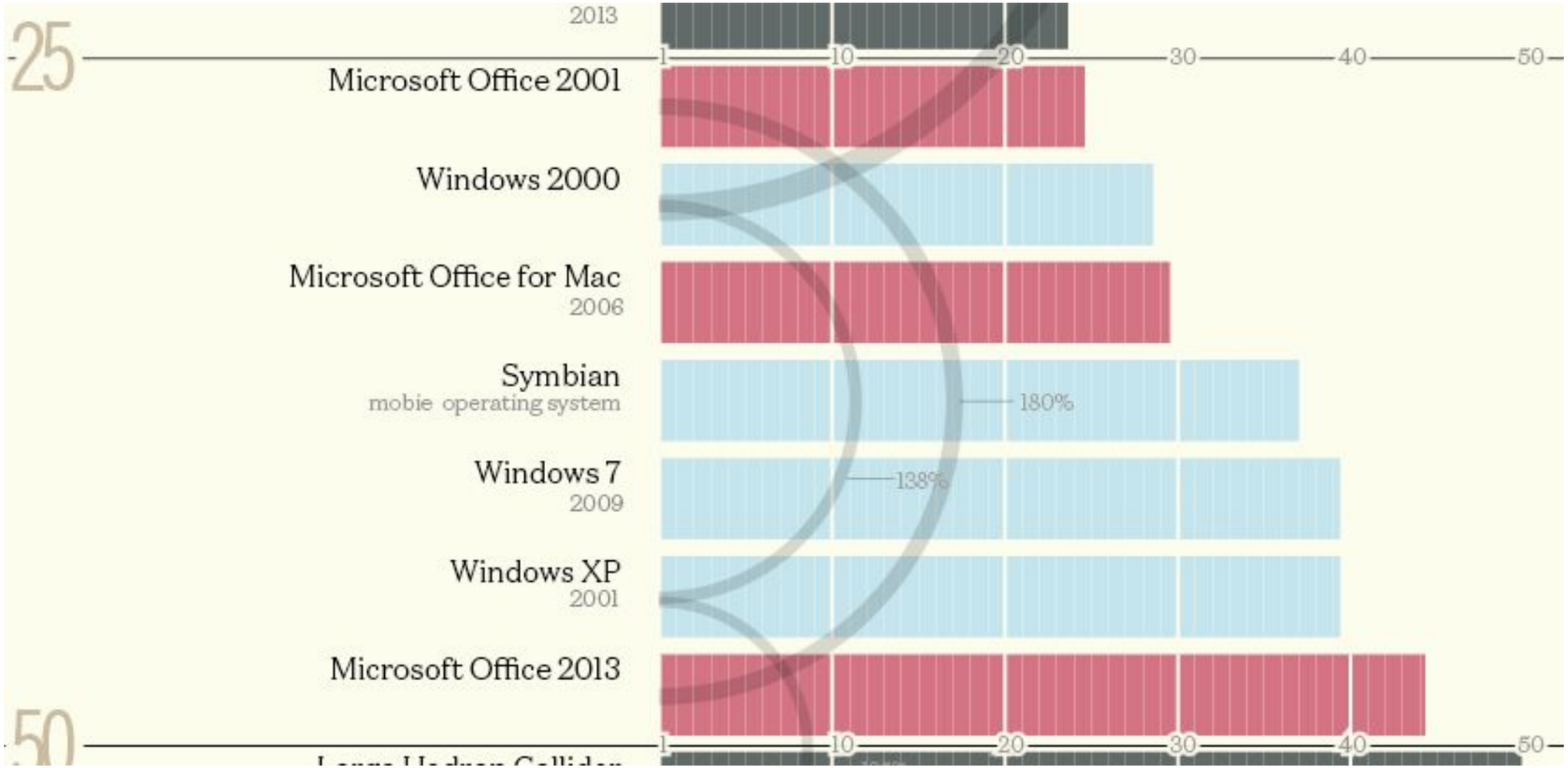
jfreechart: 300,000



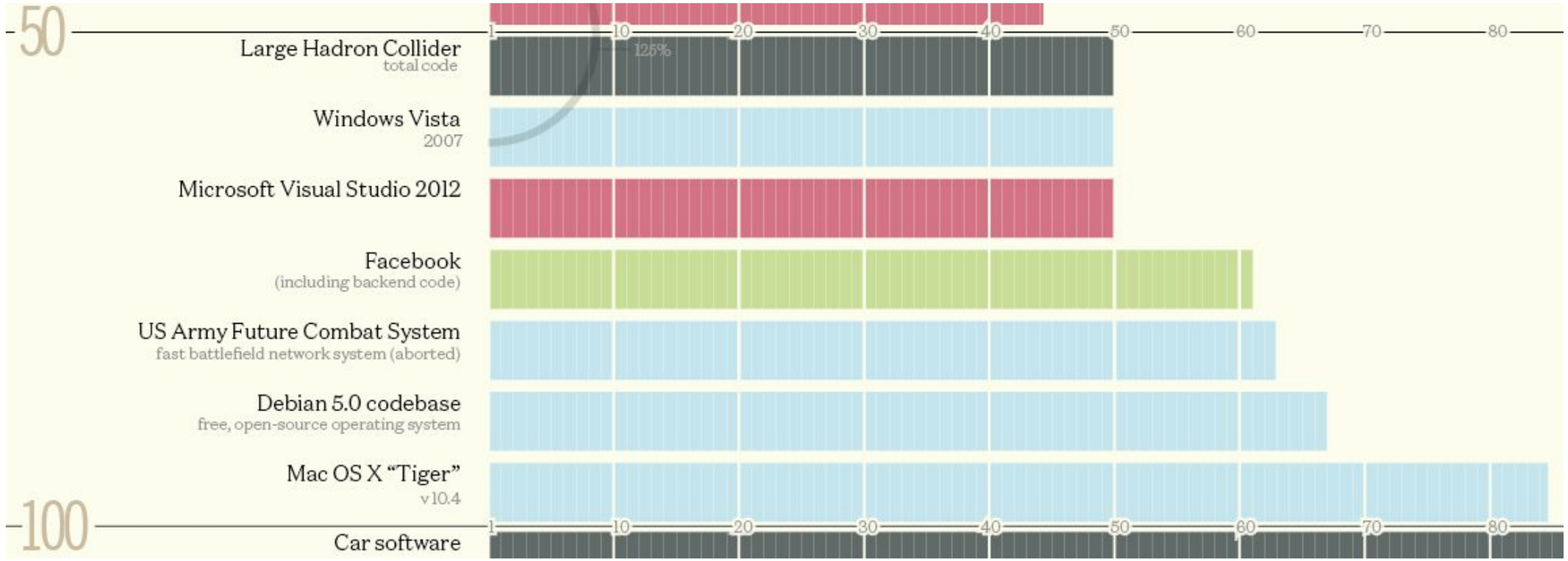
Example Programs: 1-10 MLOC



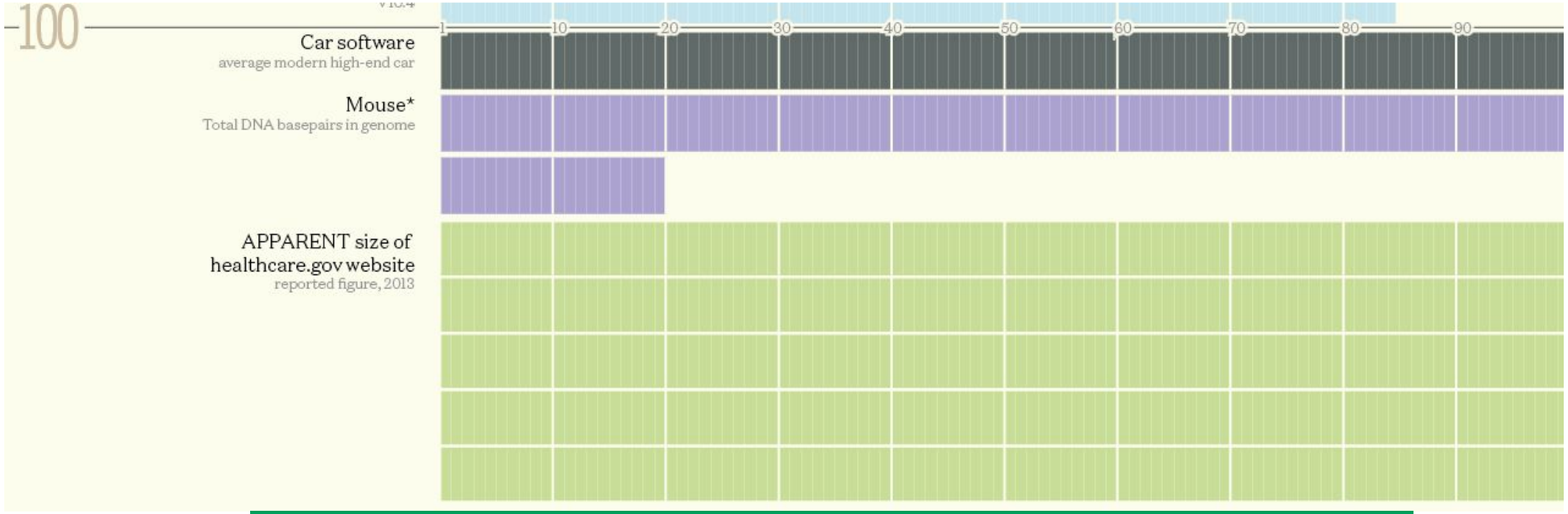
Example Programs: 25 – 50 MLOC



Example Programs: 50 – 100 MLOC



Example Programs: 0.1 – 2.0BLOC



WIRED

Google Is 2 Billion Lines of Code—And It's All in One Place

BUSINESS | CULTURE | GEAR | IDEAS | SCIENCE

SHARE

SHARE

TWEET

CADE METZ BUSINESS 09.16.15 10:00 AM

GOOGLE IS 2 BILLION LINES OF CODE—AND IT'S ALL IN ONE PLACE

Humans Are Poor At Comprehending Large Scales

- libpng 85 000
- google 2 000 000 000
- Imagine that there is a bug somewhere, anywhere, in libpng
- You can find it in a minute!
- At that same rate, it will take you *more than two weeks* to find it in all of google
  - A one-hour bug on libpng is three years on google
  - Unless we do things differently …

Fault Localization

- **Fault localization** is the task of identifying source code regions implicated in a bug
  - "This regression test is failing. Which lines should we change to fix things?"
- Answer is not unique: there are often many places to fix a bug
  - Example: check for null at caller or callee?
- Debugging includes fault localization
- Answer may take the form of a list (e.g., of lines) ranked by suspiciousness

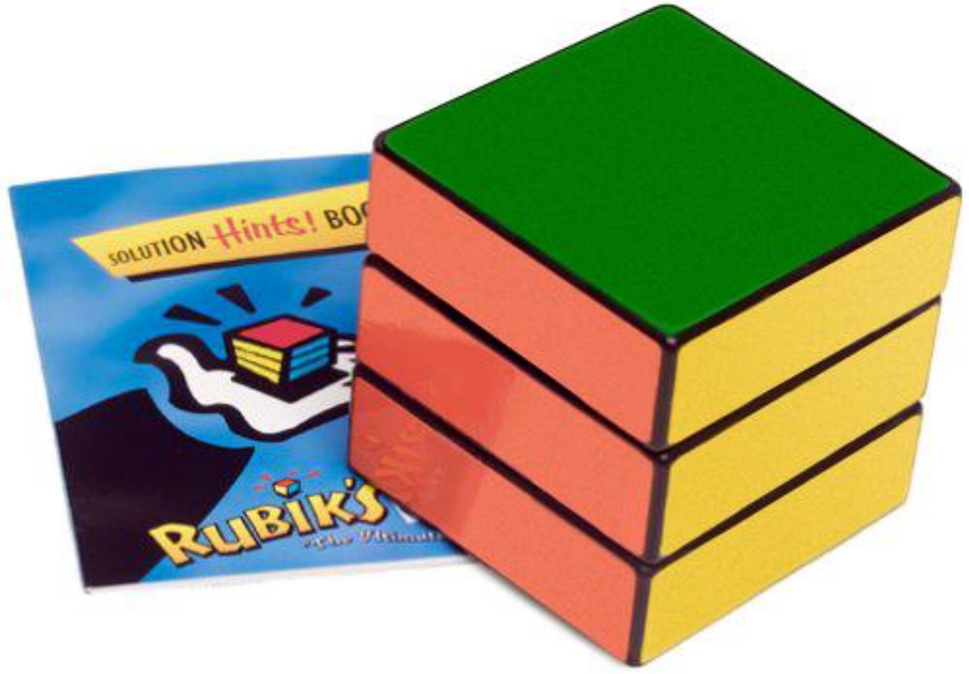
What is a Debugger?

- “A software tool that is used to detect the source of program or script errors, by performing step-by-step execution of application code and viewing the content of code variables.”

- Microsoft Developer Network

Debuggers

- Can operate on source code or assembly code
- Inspect the values of registers, memory
- Key Features (we'll explain all of them)
  - Attach to process
  - Single-stepping
  - Breakpoints
  - Conditional Breakpoints
  - Watchpoints



Signals

- A **signal** is a notification sent to a process about an event:
  - User pressed Ctrl-C (or did **kill %pid**)
    - Or asked the Windows Task Manager to terminate it
  - Exceptions (divide by zero, null pointer)
  - From the OS (**SIGPIPE**)

- You can install a **signal handler** – a procedure that will be executed when the signal occurs.

Signal Example

- What does this program print?


```c
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

int global = 11;

int my_handler() {
  printf("In signal handler, global = %d\n",
    global);
  exit(1);
}

void main() {
  int * pointer = NULL;

  signal(SIGSEGV, my_handler) ;

  global = 33;

  * pointer = 0;

  global = 55;

  printf("Outside, global = %d\n", global);
}
```

Attaching A Debugger

- Requires operating system support
- There is a special **system call** that allows one process to act as a debugger for a target
  - What are the security concerns?
- Once this is done, the debugger can basically "catch signals" delivered to the target
  - This isn't exactly what happens, but it's a good explanation …

Building a Debugger

- We can then get breakpoints and interactive debugging
  - Attach to target
  - Set up signal handler
  - Add in exception-causing instructions
  - Inspect globals, etc.

```
#include <stdio.h>
#include <signal.h>

#define BREAKPOINT *(0)=0

int global = 11;

int debugger_signal_handler() {
  printf(“debugger prompt: \n”);
  // debugger code goes here!
}

void main() {
  signal(SIGSEGV, debugger_signal_handler) ;

  global = 33;

  BREAKPOINT;

  global = 55;

  printf("Outside, global = %d\n", global);
}
```

Advanced Breakpoints

- Optimization: **hardware breakpoints**
  - Special registers (a few of them): if PC value = HBP register value, signal an exception
  - Faster than software, works on ROMs, only limited number of breakpoints, etc.
- Feature: **conditional breakpoint**: “break at instruction X if some_variable = some_value”
- As before, but signal handler checks to see if some_variable = some_value
  - If so, present interactive debugging prompt
  - If not, return to program immediately
  - Is this fast or slow?

Single-Stepping

- Debuggers also allow you to advance through code one instruction at a time
- To implement this, put a breakpoint at the first instruction (= at program start)
- The "**single step**" or "next" interactive command is equal to:
  - Put a breakpoint at the next instruction
  - Resume execution
  - (No, really.)

Watchpoints

- You want to know when a variable changes
- A **watchpoint** is like a breakpoint, but it stops execution after any instruction changes the value at location L
- How could we implement this?

Watchpoint Implementation

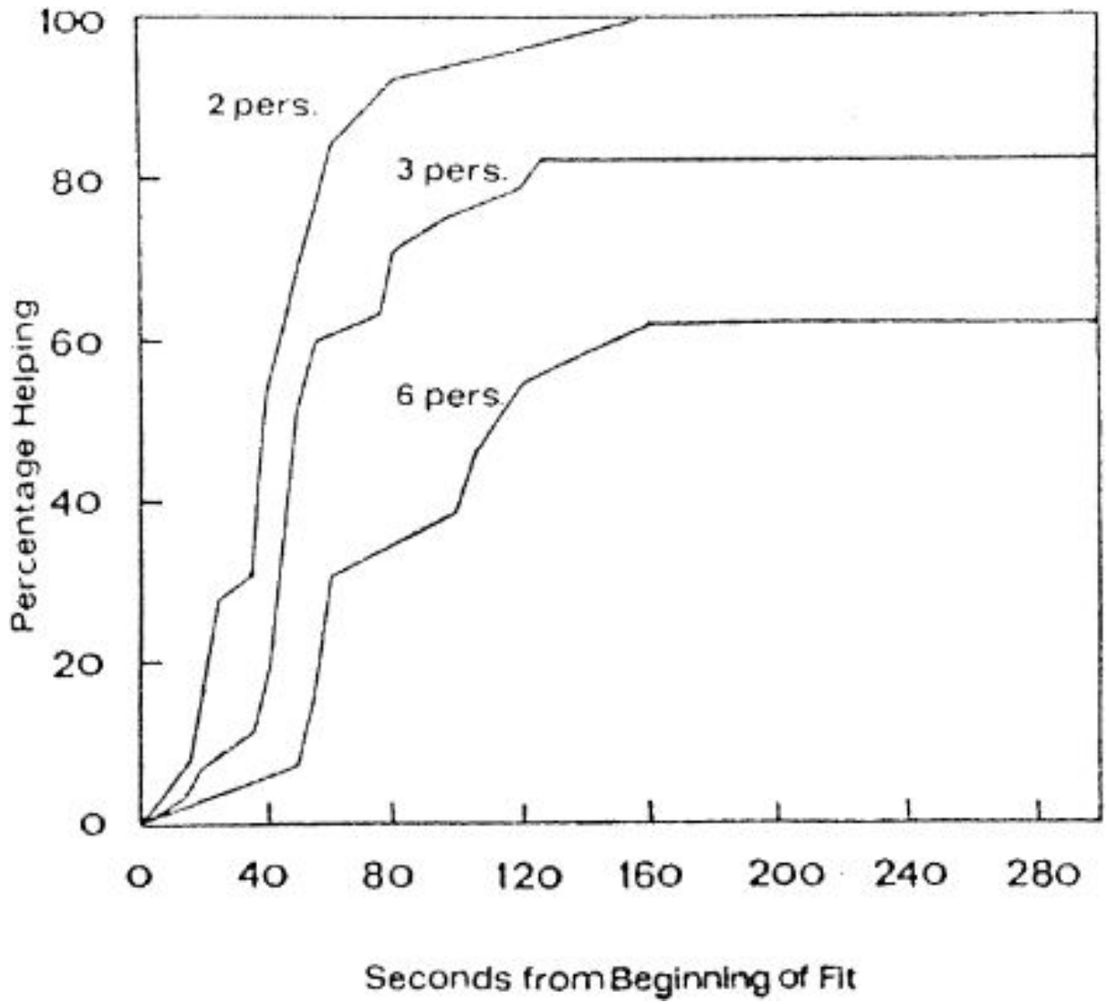
- Software Watchpoints
  - Put a breakpoint at *every instruction* (ouch!)
  - Check the current value of L against a stored value
  - If different, give interactive debugging prompt
  - If not, set next breakpoint and continue (single-step)
- Hardware Watchpoints
  - Special register holds L: if the value at address L ever changes, the CPU raises an exception

Psychology: Reactions

- You are invited to participate in a group discussion of “personal problems”. Because of the sensitive nature of the discussion, it takes place over an intercom. During the discussion, you hear:
  - “I-er-um-I think I-I need-er-if-if could-er-er-somebody er-er-er-er-er-er-er give me a little-er-give me a little help here because-er-I-er-I’m-er-erh-h-having a-a-a real problem-er-right now and I-er-if somebody could help me out it would-it would-er-er s-s-sure be-sure be good . . . because-there-er-er-a cause I-er-I-uh-I’ve got a-a one of the-er-sei er-er-things coming on and-and-and I could really-er-use some help so if somebody would-er-give me a little h-help-uh-er-er-er-er-er c-could somebody-er-er-help-er-uh-uh-uh (choking sounds). . . . I’m gonna die-er-er-I’m . . . gonna die-er-help-er-er-seizure-er-[chokes, then quiet].”

Psychology: Reactions

- The more people in the discussion, the longer it takes anyone to take action
- Gender (of you or others) had no effect



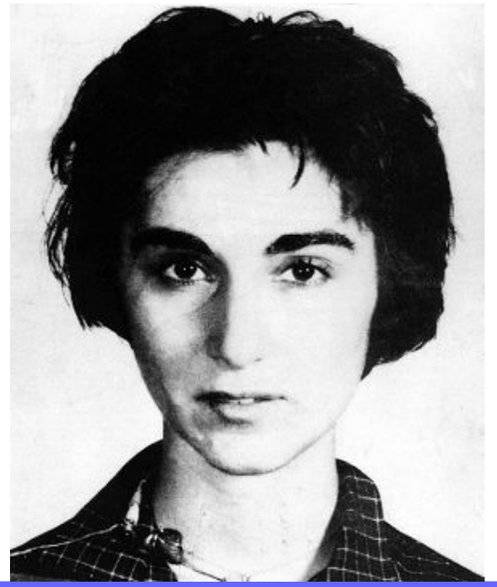
Bystander Effect

- “It is our impression that nonintervening subjects not decided *not* to respond. Rather they were still in a state of indecision and conflict concerning whether to respond or not. The emotional behavior of these nonresponding subjects was a sign of their continuing conflict ...”
- Implications for SE: Team sizing considerations. Who will volunteer to be assigned this bug?
- [Darley and Latane. Bystander Intervention in Emergencies: Diffusion of Responsibility. J. Personality and Social Psych. 8(4) 1968.]

Bystander Effect

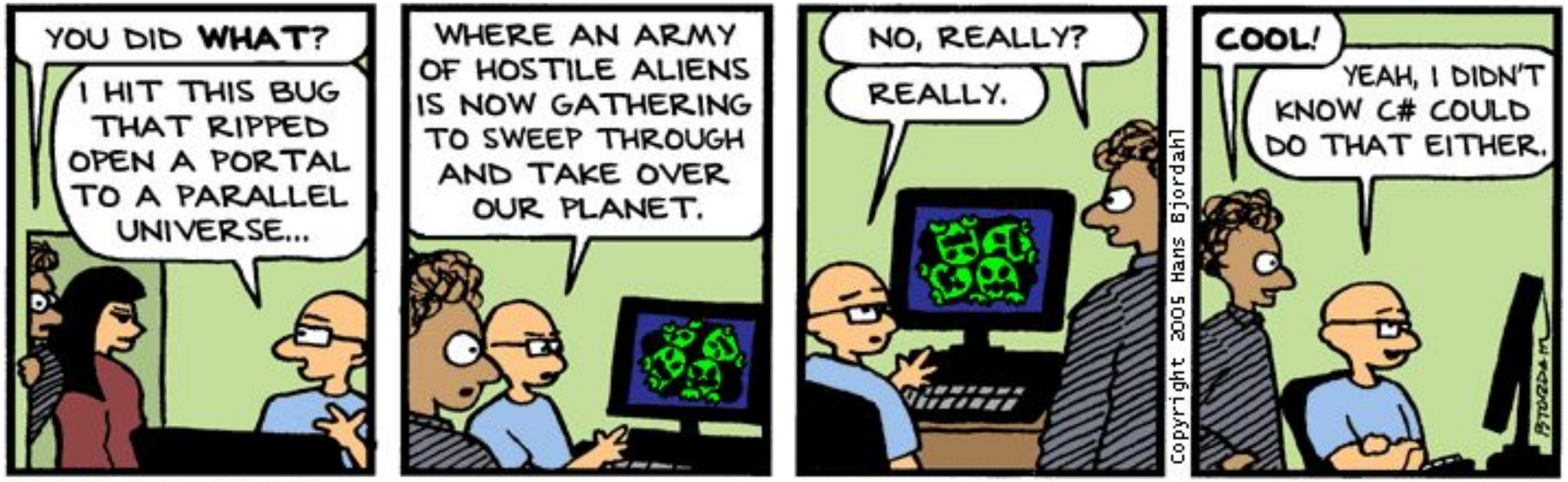
- [Darley and Latane. Bystander Intervention in Emergencies: Diffusion of Responsibility. J. Personality and Social Psych. 8(4) 1968.]

- Implications for SE: Team sizing considerations. Who will volunteer to be assigned this bug?



Human Fault Localization

- OK, so humans have debugg*ers*
- Are humans any good at debugg*ing*?
- Not all bugs are equally easy to find
- Not all programs are equally easy to debug



Find The Bug (Towers of Hanoi)

- Over 53% of participants (seniors) could find the bug in about 3 minutes
- Note: conditional branches, recursive calls, rich comments, variable names

```
1  /*****************************************
2    Performs the initial call to moveTower
3    to solve the puzzle.  Moves the disks
4    from tower 1 to tower 3 using tower 2.
5  *****************************************/
6  public void solve () {
7    moveTower (totalDisks, 1, 3, 2);
8  }
9
10 /*****************************************
11   Moves the specified number of disks
12   from one tower to another by moving a
13   subtower of n-1 disks out of the way,
14   moving one disk, then moving the
15   subtower back. Base case of 1 disk.
16 *****************************************/
17 private void moveTower (int numDisks,
18                         int start, int end, int temp) {
19 if (numDisks == 1)
20     moveTower(numDisks-1, temp, end, start);
21   else {
22     moveTower (numDisks-1, start, temp, end);
23     moveOneDisk (start, end);
24     moveTower (numDisks-1, temp, end, start);
25   }
26 }
27 /*****************************************
28   Prints instructions to move one disk
29   from the specified start tower to the
30   specified end tower.
31 *****************************************/
32 private void moveOneDisk (int start, int end) {
33   System.out.println ("Move one disk from "
34       + start + " to " + end);
35 }
```

Find The Bug 2

- Only 33% could locate the bug
- Note: shorter, simpler identifiers, simpler control flow, not as abstract

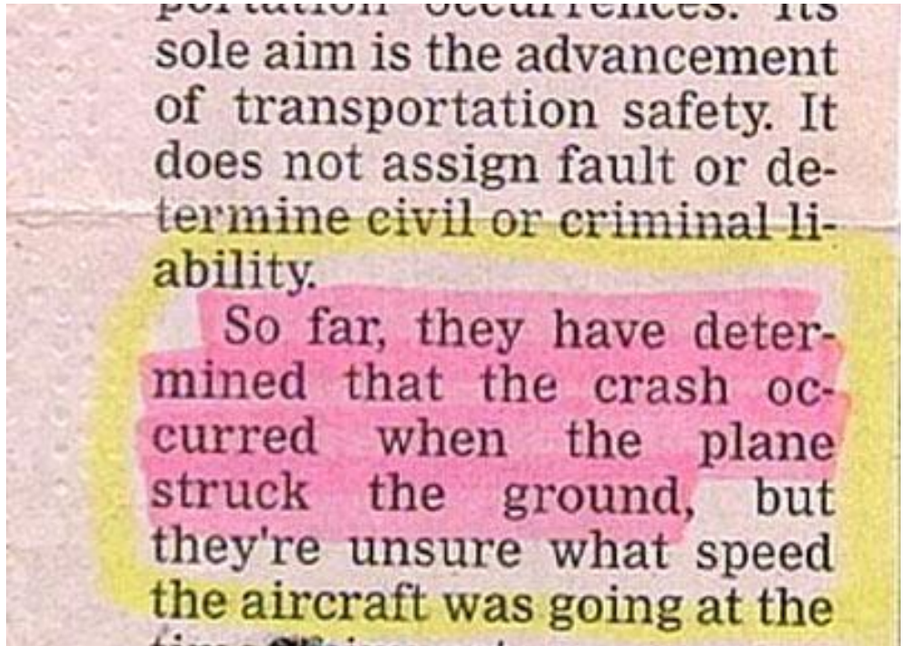
```
1  /** Move a single disk from src to dest. */
2  public static void hanoi1(int src, int dest){
3    System.out.println(src + " => " + dest);
4  }
5  /** Move two disks from src to dest,
6      making use of a spare peg. */
7  public static void hanoi2(int src,
8                            int dest, int spare) {
9    hanoi1(src, dest);
10   System.out.println(src + " => " + dest);
11   hanoi1(spare, dest);
12 }
13 /** Move three disks from src to dest,
14     making use of a spare peg. */
15 public static void hanoi3(int src,
16                            int dest, int spare) {
17   hanoi2(src, spare, dest);
18   System.out.println(src + " => " + dest);
19   hanoi2(spare, dest, src);
20 }
```

Human Study

- Participants (n=65, half with >4 years of experience) were shown snippets of textbook
  - Defects seeded based on 100 consecutive bug fixes from the Mozilla bug repository
- Double experimental control
  - Quicksort in Textbook A vs. Textbook B has the same complexity (differs only in style)
  - Bubblesort in Textbook A vs. AVL Tree in Textbook A differ in complexity (have same presentation style)
- [Z. Fry et al.: A Human Study of Fault Localization Accuracy. International Conference on Software Maintenance (ICSM) 2010]

What Do You Think?

- Rank these: which of these bugs is easiest for humans to find?
  - Extra Assignment
  - Missing Statement
  - Extra Conditional
  - Calling Wrong Method
  - Extra Statement


sole aim is the advancement of transportation safety. It does not assign fault or determine civil or criminal liability.

So far, they have determined that the crash occurred when the plane struck the ground, but they're unsure what speed the aircraft was going at the

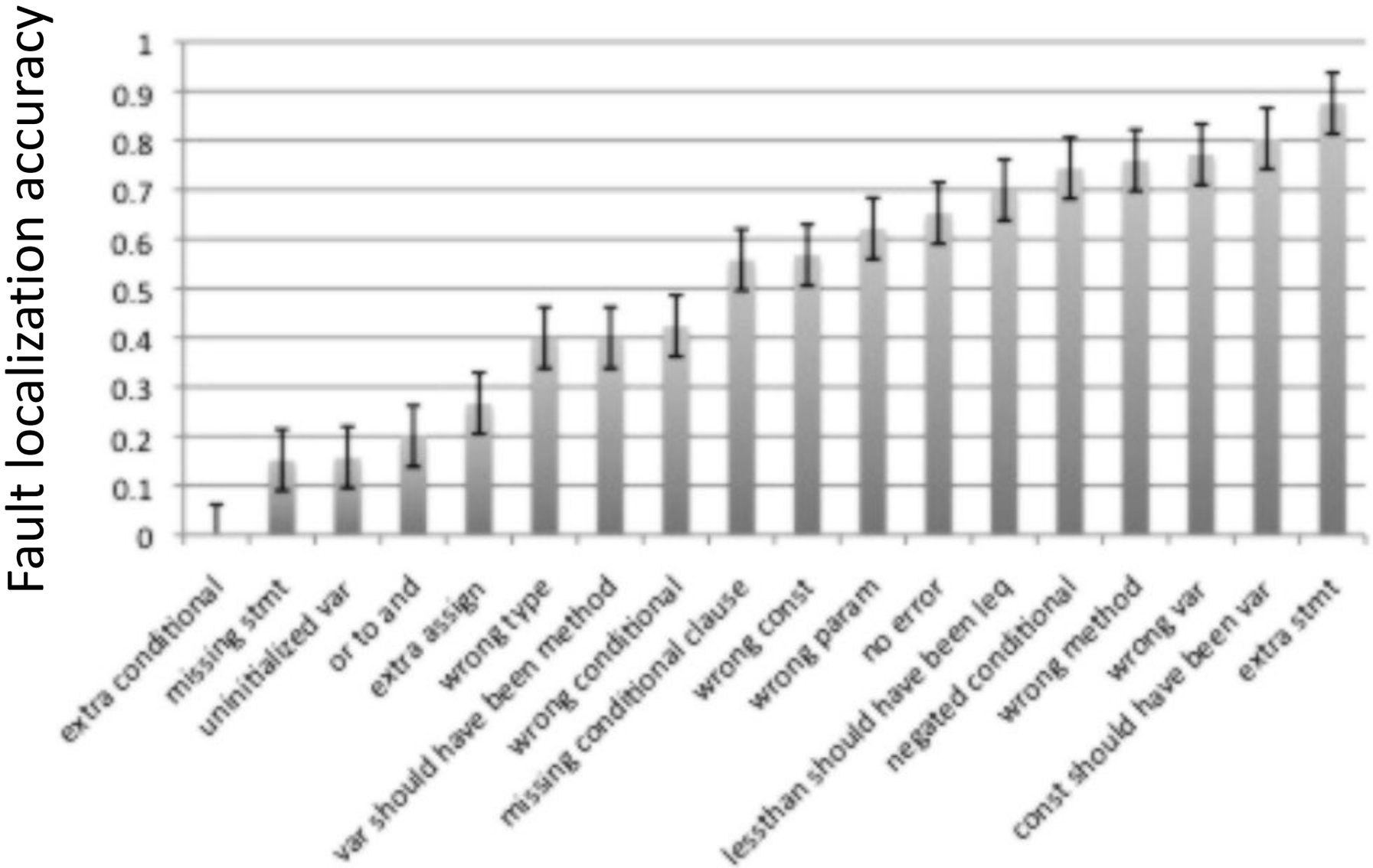


Fig. 3. Human fault localization accuracy as a function of defect type.

Tool Support for Fault Localization

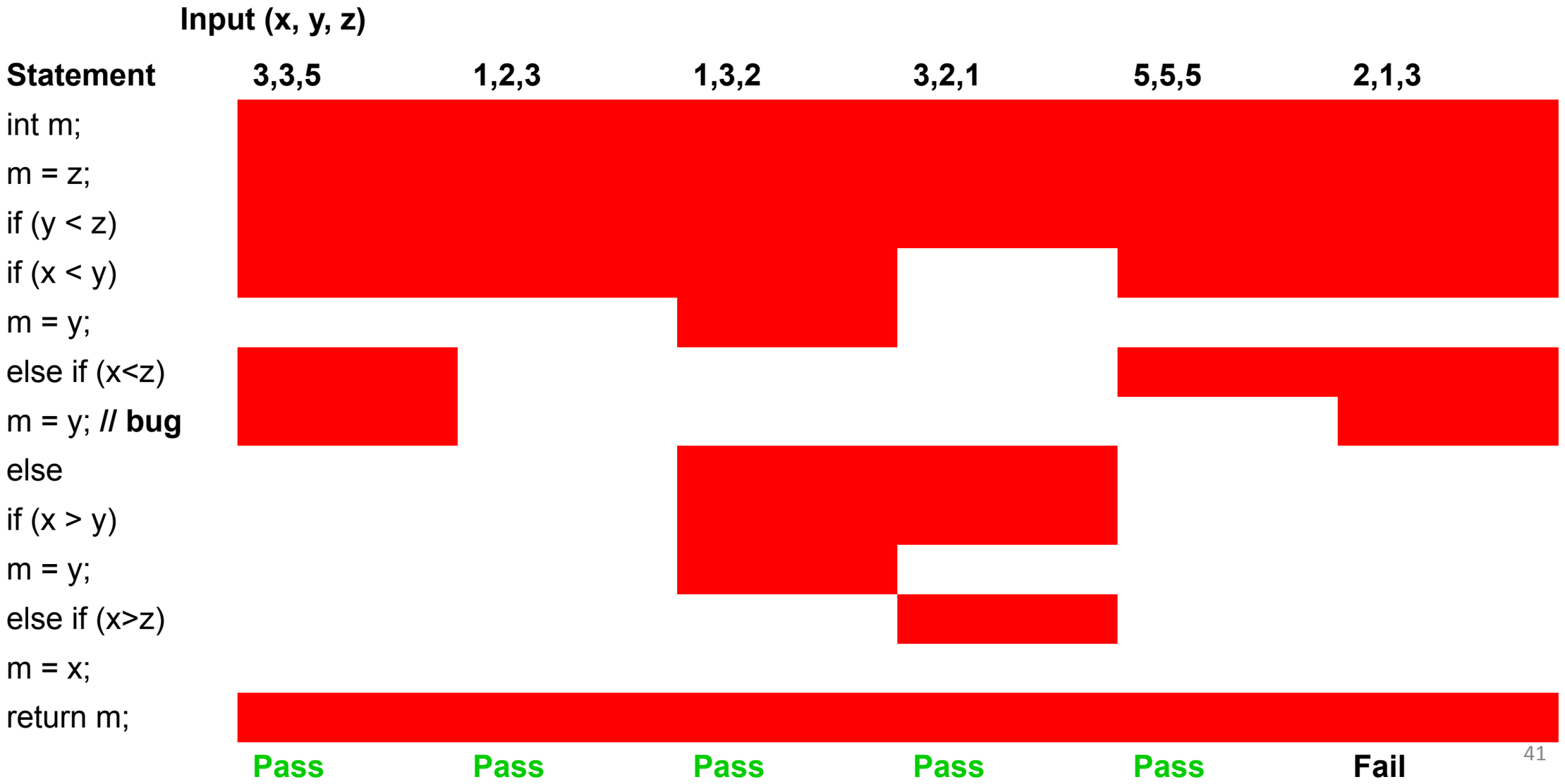
- A **spectrum-based fault localization** tool uses a dynamic analysis to rank suspicious statements implicated in a fault by comparing the statements covered on failing tests to the statements covered on passing tests
- Basic idea:
  - Instrument the program for coverage (put print statements everywhere)
  - Run separately on normal inputs and bug-inducing inputs
  - Compute the set difference!

Fault Localization Example

- Consider this simple buggy program:

```
int mid(int x, int y, int z) {
  int m;
  m = z;
  if (y < z) {
    if (x < y) m = y;
    else if (x < z) m = y; /* BUG: m=x; */
  } else {
    if (x > y) m = y;
    else if ( x > z) m = x;
  }
  return m;
}
```

Coverage-Based Fault Localization



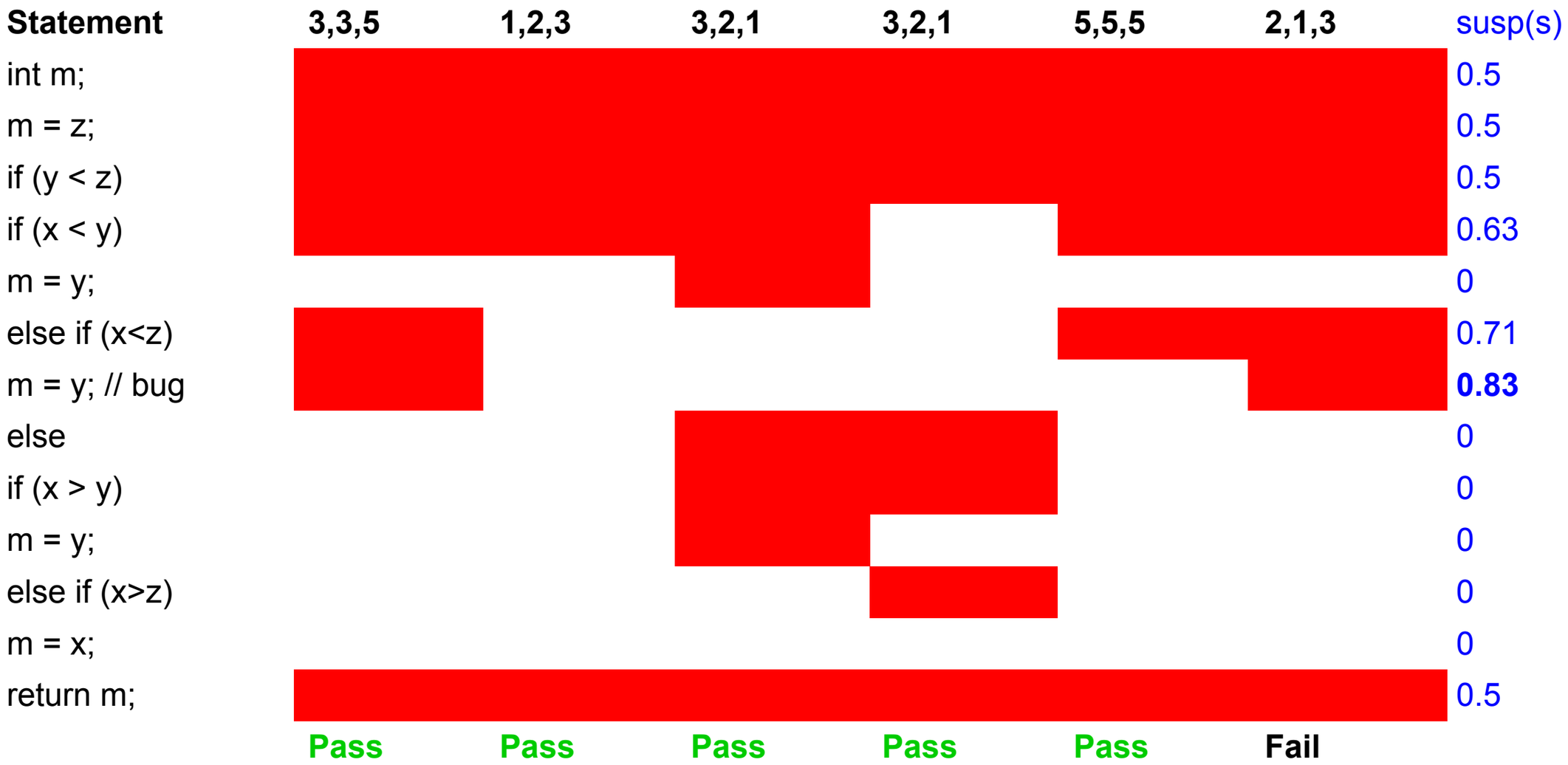
Insight: Print-Statement Debugging

- If you do not execute X but you do observe the bug, X cannot be related to that bug
- If Y is primarily executed when you observe the bug, it is more likely to be implicated than Z which is primarily executed when you do not observe the bug
- **Suspiciousness Ranking**

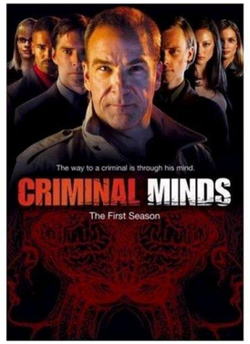
$$susp(s) = \frac{fail(s) / total\_fail}{fail(s)/total\_fail + pass(s)/total\_pass}$$

[Jones and Harrold. Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique. ASE 2005.]

Fault Localization Ranking



Profiling



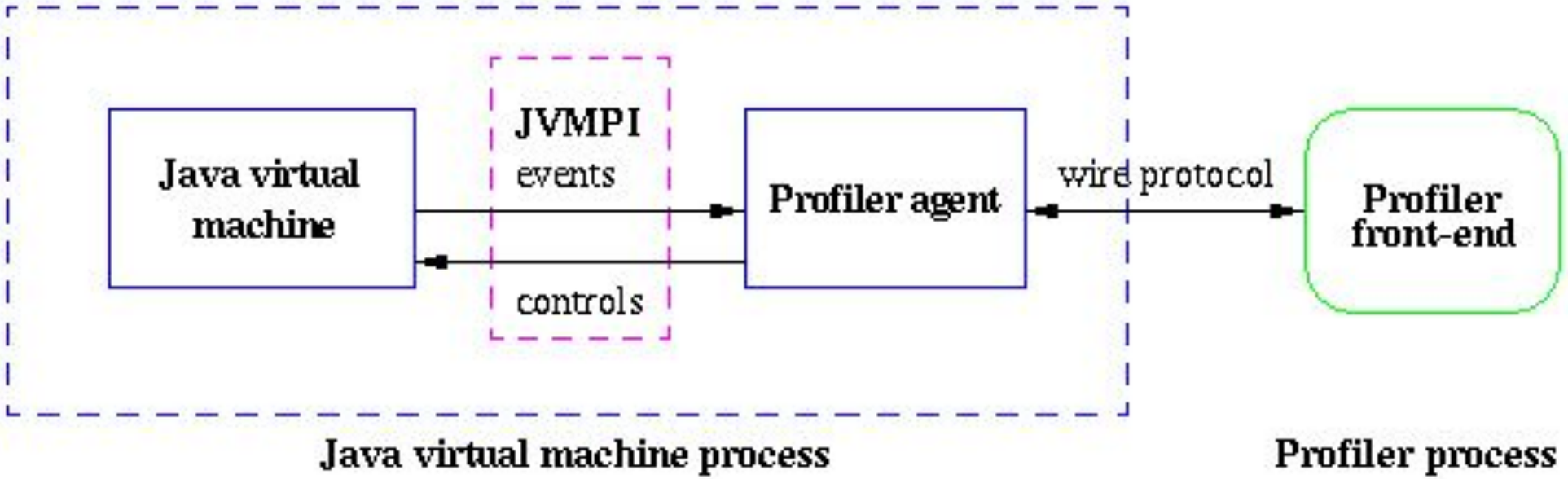
- A **profiler** is a performance analysis tool that measures the frequency and duration of function calls as a program runs.
  - A **flat profile** computes the average call times for functions but does not break times down based on context
  - A **call-graph profile** computes call times for functions and also the call-chains involved

Event-Based Profiling

- Interpreted languages provide special hooks for profiling
  - Java: JVM-Profile Interface, JVM API
  - Python: sys.set_profile() module
  - Ruby: profile.rb, etc.
- You register a function that will get called whenever the target program calls a method, loads a class, allocates an object, etc.
  - cf. “signal handler”

JVM Profiling Interface

- VM notifies profiler agent of various **events** (heap allocation, thread start, method invocation, etc.)
- Profiler agent issues control commands to the JVM and communicates with a GUI



Statistical Profiling

- You can arrange for the operating system to send you a **signal** (just like before) every X seconds (see **alarm(2)**)
- In the **signal handler** you determine the value of the target program counter
  - And append it to a growing list file
  - This is **sampling**
- Later, you use debug information from the compiler to map the PC values to procedure names
  - Sum up to get amount of time in each procedure

Sampling Analysis

- Advantages
  - Simple and cheap – the **instrumentation** is unlikely to disturb the program
  - No big slowdown
- Disadvantages
  - Can completely miss periodic behavior (e.g., you sample every *k* seconds but do a network send at times 0.5 + n*k* seconds)
  - High error rate: if a value is n times the sampling period, the expected error in it is sqrt(n) sampling periods
- Read the **gprof** paper

Real-World Tool Utility

- Human study of 34 graduate students
- Given Tarantula (as a friendly plugin for Eclipse) and asked to complete two debugging tasks
  - Tetris: square block rotation bug
  - NanoXML: parsing library exception
- Hypotheses:
  - Tools will help us debug faster
  - Tools help more on harder problems

[Parnin and Orso. Are Automated Debugging Techniques Actually Helping Programmers? ISSTA '11.]

Results

- Experts Are Faster When Using Tools
  - Over all participants, tools did not help
  - Top-third of participants went from 14m:28s to 8:51 with tool support (for Tetris, $p < 0.05$)
- Tools Did Not Help With Harder Tasks
- Changes In Rank Did Not Matter
  - (Rank) 7 → 35 in Tetris, 83 → 16 in NanoXML
  - Why is this so crucial here?

Explanations

- "Based on this data, we have determined that programmers do not visit each statement in a linear fashion."
- "If the ***faulty nature*** of a statement were apparent to the developers by ***just looking at it***, tool usage ***should stop*** as soon as they get to ***that statement*** in the list."
  - "participants, on average, spent another ten minutes using the tool after they first examined the faulty statement. That is, participants spent (or wasted) on average 61% of their time continuing to inspect statements with the tool after they had already encountered the fault."

Implications

- You are a Software Engineering manager
- Making a process decision: do we purchase, train on, and deploy Tarantula?
- Tarantula claims: this tool will correctly rank buggy statements near the top of the list
  - This is almost a red herring!
  - You must examine the “end-to-end” performance
- So fault localization tools are worthless?

Nuanced Example

- Suppose you have three devs: A, B and C
  - Expert, Medium, Novice
- Tarantula makes A, the expert, 39% faster
  - But makes everything 13% slower (training, overhead, whatever)
- If everything is equal, net gain = 0 (as in study)
- But suppose A is 25x faster than C (*productivity* later)
  - A=25, B=13, C=1 → in this world your team, overall, is 8.7% faster with Tarantula

Questions?

- Exam 1 (Friday)
- Enjoy the break!
- HW4 (due after break): CodeSonar