Software Foundations of Security and Privacy (15-316, spring 2016)
Lecture 2: Building Safe Systems

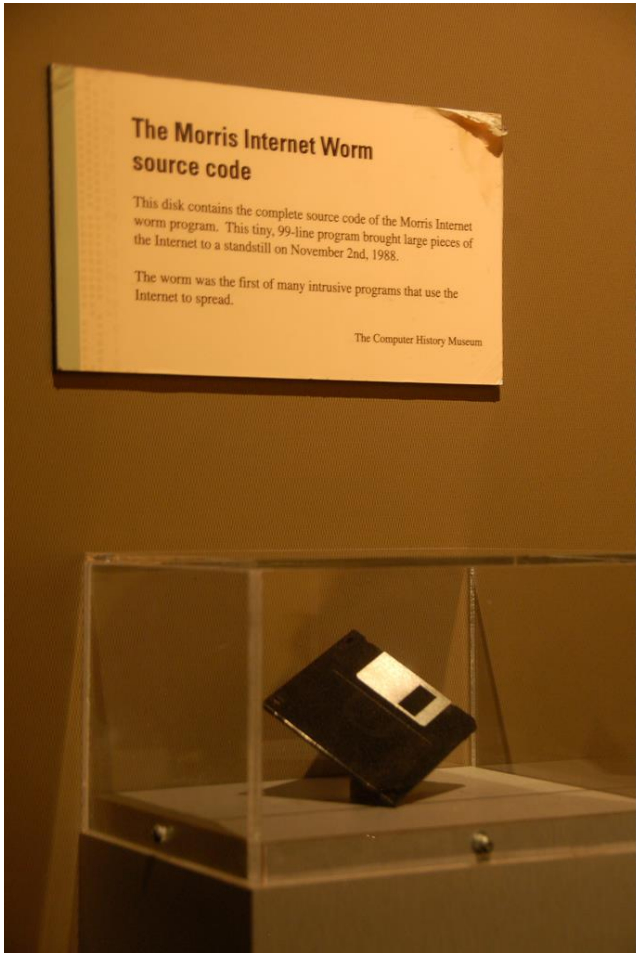
Jean Yang

jyang2@andrew.cmu.edu

This lecture: celebrities and the Internet

Selfie from the 2014 Oscars that was retweeted over one million times and caused a Twitter outage.

Before *photos* broke the Internet…




THE NATION'S NEWSPAPER

10 CENTS

USA TODAY

NO. 1 IN THE USA...5.3 MILLION READERS EVERY DAY

CAMPAIGN DIARIES

Computer alarms set since 'infection'

VIRUSES 'INEVITABLE,' 1B

The first "celebrity bug:" Morris Worm, November 1988.

Bug: Morris Worm

- Launched **November 2, 1988** from MIT.
- One of the **first worms** distributed on the internet, and the first celebrity worm.
- Resulted in first felony conviction, under the 1986 **Computer Fraud and Abuse Act**.
- Named after its creator, Cornell University graduate student **Robert Morris**.

Innocent intentions

Photo of Robert Morris, now a tenured professor at MIT (and my good friend's PhD thesis advisor)

- Supposedly intended to gauge the size of the Internet.
- Exploited known vulnerabilities in Unix commands.
- Flaw (or was it?) in design caused program to spread very rapidly.

Fallout

- The estimated damage was $100k to $10,000k.
- People estimated the worm affected 10% of the 60k computers on the internet.
- Internet was partitioned for several days while people cleaned up their networks.
- DARPA funded founding of CERT/CC at CMU.
- Robert Morris was sentenced to three years probation, 400 hours of community service, and fined over $10k.

Bug: Heartbleed

- Vulnerability in the OpenSSL cryptographic library, introduced in 2012 and announced April 2014.
- Allows anyone from the Internet to access the protected memory.

What Heartbleed did

- Allows attackers to eavesdrop on communications, steal data, and impersonate services and users—all without a trace.
- Users' session cookies, passwords, and more became vulnerable.
- Solutions included patching the vulnerability, changing your password, and staying away from the Internet.

Fallout

“Some might argue that Heartbleed is the worst vulnerability found (at least in terms of its potential impact) since commercial traffic began to flow on the Internet.” –Joseph Steinberg, *Forbes*

“If you need strong anonymity or privacy on the Internet, you might want to stay away from the Internet entirely for the next few days while things settle.” –Tor project

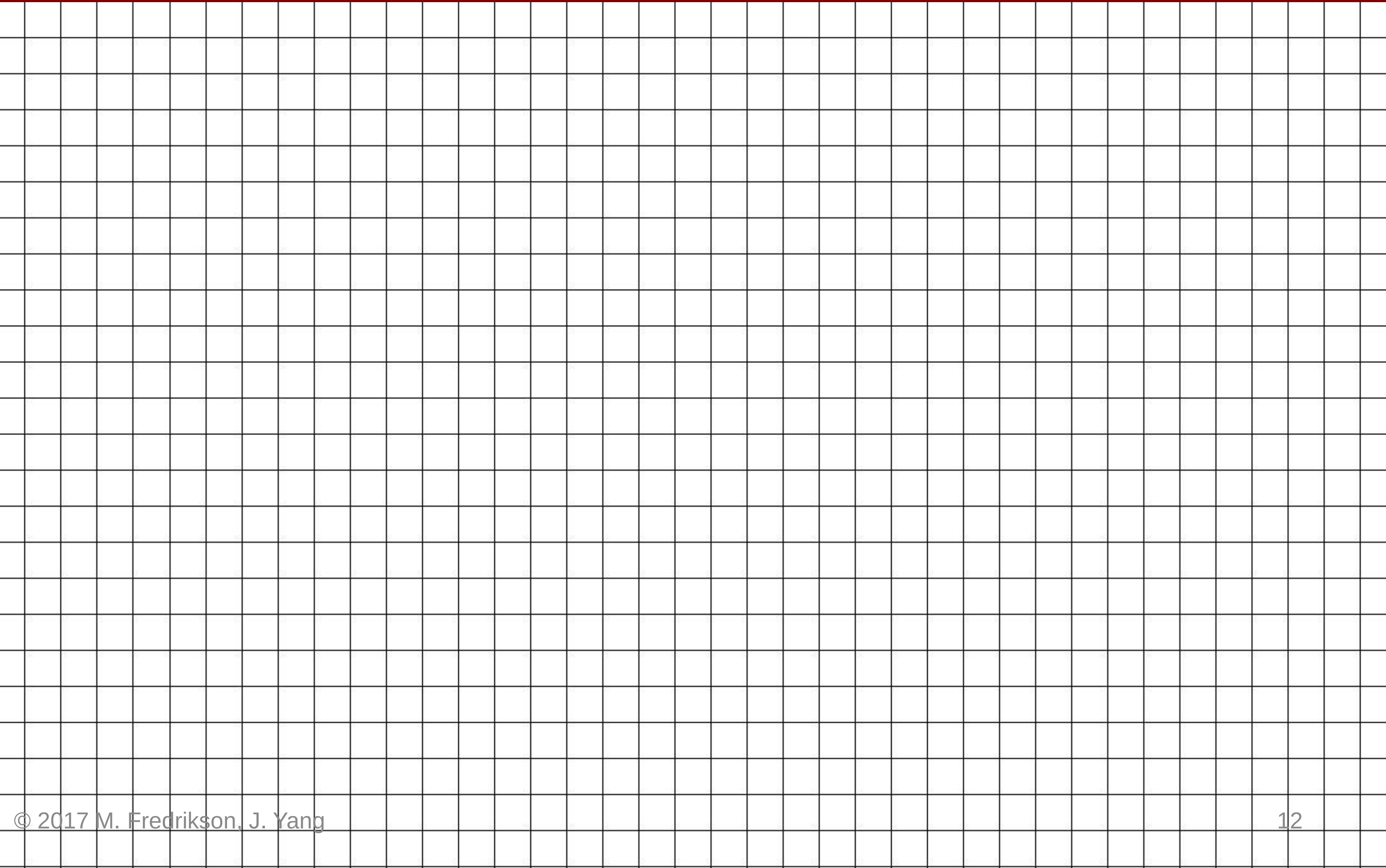
Fallout

- At the time of disclosure, **17%** (around half a million) of the Internet's secure web servers were believed to be vulnerable.
- As of May 20, 2014, **1.5%** of the 800,000 most popular TLS-enabled websites were still vulnerable.
- eWEEK estimates **$500 million** as a starting point for the cost. For example, Heartbleed enabled hackers to steal security keys from Community Health Systems, the second-biggest for-profit hospital chain in the United States, compromising the confidentiality of **4.5 million patient** records.

What went wrong?

We want memory to be everything

But memory is just memory



Managing memory requires care

Mixing memories: buffer overflows

- Programs expect inputs (often strings) to be of a certain size.
- Giving programs inputs of larger size usually causes them to crash.
- A buffer overflow exploit involves “taking over” programs by giving programs inputs of larger size.

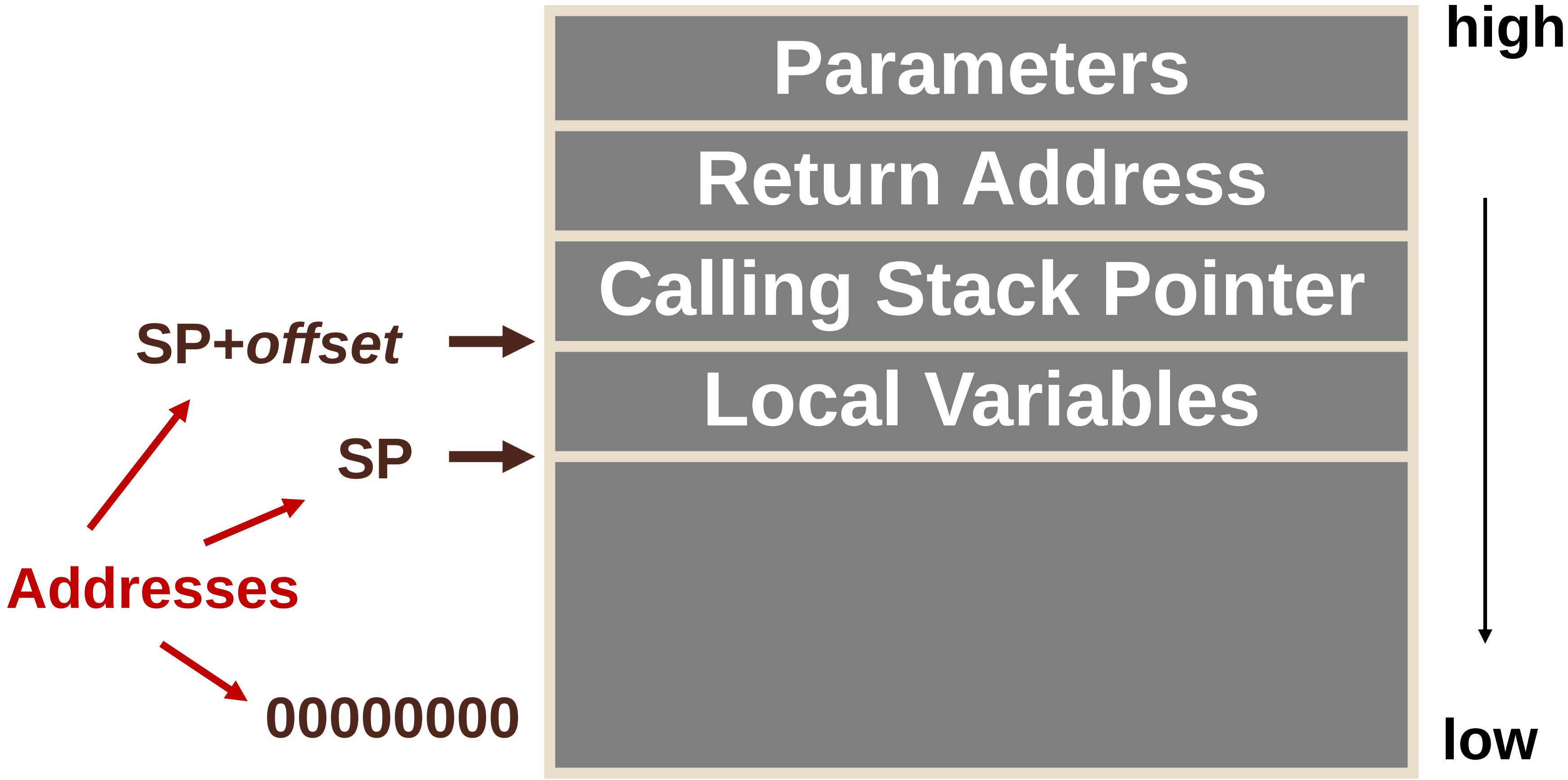
Borrowed heavily from
https://courses.cs.washington.edu/courses/cse451/05sp/.../**overflow**1.ppt

Background: C call stacks

- When a function call is made, the return address is put on the stack.
- Often the values of parameters are put on the stack.
- Usually the function saves the stack frame pointer (on the stack).
- Local variables are on the stack.

Borrowed heavily from
https://courses.cs.washington.edu/courses/cse451/05sp/.../**overflow**1.ppt

Background: stack frames



Borrowed heavily from
https://courses.cs.washington.edu/courses/cse451/05sp/.../**overflow**1.ppt

Background: example stack

18

addressof(y=3) ***return address***

saved stack pointer

y

x

buf

```
x=2;
foo(18);
y=3;
```

```
void foo(int j) {
    int x,y;
     char buf[100];
    x=j;
      …
}
```

Buffer overflows, more specifically…

- General idea is to overflow a buffer so that it overwrites the return address.
- When the function is done it will jump to whatever address is on the stack.
- We put some code in the buffer and set the return address to point to it!

Borrowed heavily from
https://courses.cs.washington.edu/courses/cse451/05sp/.../**overflow**1.ppt

Example of buffer overflow

```
void foo(char *s) {
  char buf[10];
  strcpy(buf,s);
  printf("buf is %s\n",s);
}
…
foo("thisstringistoolongforfoo");
```

Borrowed heavily from
https://courses.cs.washington.edu/courses/cse451/05sp/.../**overflow**1.ppt

Morris Worm, a buffer overflow

- Buffer overflow attacking gets() in fingerd.
- fingerd declares a 512-byte buffer to be used by gets() without bounds checking. The buffer is the first local variable.
- Attack inserted a 400-byte **NOP sled** that ended with a call to execve("/bin/sh",0,0), opening a shell and receiving instructions across the network.

Heartbleed: the opposite

- In TLS, *heartbeat* requests keep a connection open by exchanging information back and forth. The requester specifies the size of the “payload,” and the server sends it back.
- In vulnerable implementations, setting payload size to something bigger than it is makes the server send back *arbitrary* information!
- Instead of *writing* out-of-bounds code to a buffer, Heartbleed *reads* nearby out-of-bounds memory.

What we're not going to talk about

- The Morris Worm only harmed the computers it infected, and also made people start taking computer security seriously. Should hacks of this nature be considered criminal?
- The Morris Worm did not steal documents, distribute spam, or aid terrorism. Why did people get so upset? What *are* the risks of doing this kind of hacking?

What we *are* going to talk about

- Tools and frameworks for reasoning about these vast seas of memory.
- How to *specify and verify correctness* to rule out memory bugs.
- How using *safe languages* to produce programs that are safe by construction.

Why security is a correctness problem

What we ~~want~~ need*

- Buffers have their bounds checked!
- Data is data, and not contain random code.
- People can't just randomly insert code when they are supposed to be giving us data values.

Secret theme of class: we need to demand more of our software!

What is safety?

Nothing **bad** happens. ← All or nothing!

- The toaster does not burn down the house.
- Only people within the house can operate the toaster.
- Uber does not leak information about locations and credit card numbers.
- Uber does not allow data scientists to infer individual locations.

Safety and security

- **Safety:** protecting a system from accidental failures.
- **Security:** protecting a system from active attackers

Safety is required for security!

Memory safety

Memory access errors do not occur:

- buffer overflow
- null pointer dereference
- use after free
- use of uninitialized memory
- illegal free (of an already-freed pointer, or a non-malloced pointer)

Type safety

The programming language ensures that there are no discrepancies between the stated type of a value and the actual type.

- Example of something unsafe: Booleans are integers in C.
- Often strictly stronger than memory safety.

Liveness is a different thing

A **liveness property** states that something good **eventually** happens.

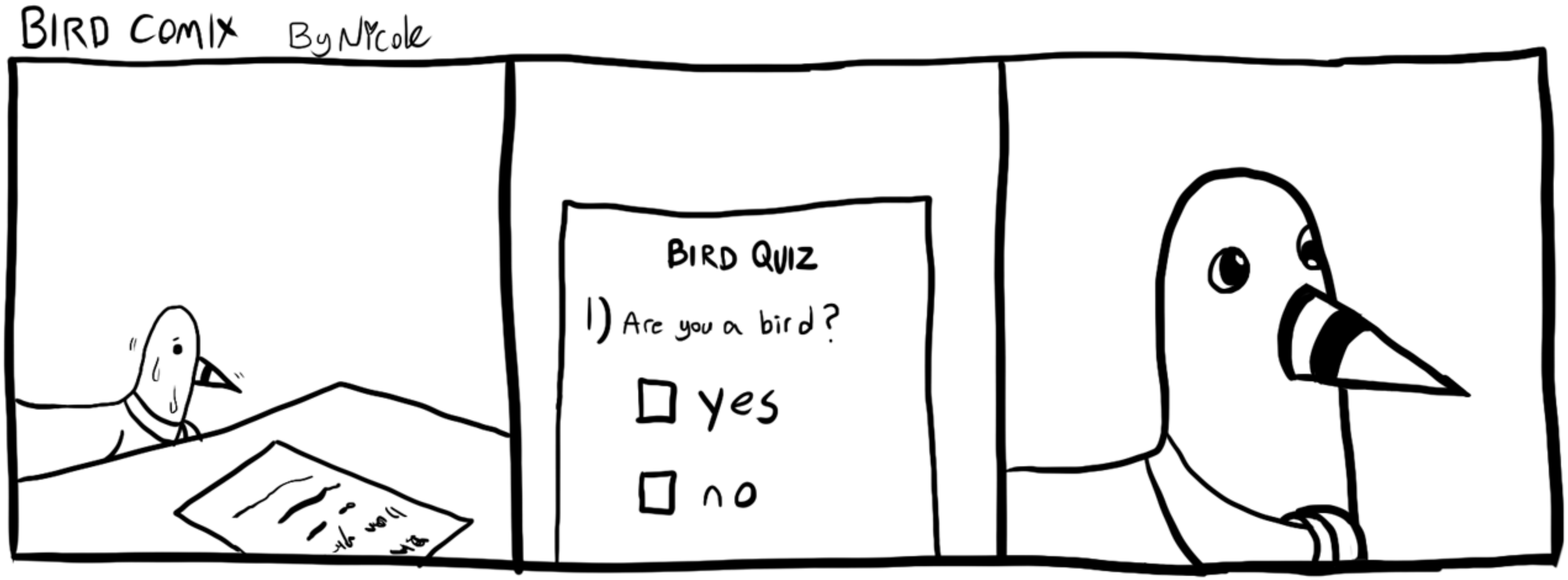
- Toaster eventually produces sufficiently cooked toast.
- Uber eventually gets you where you need to go.
- This course eventually teaches you everything about constructive security.

Need both safety and liveness

Only safety

Only liveness

Quiz time!



How safe languages can help

Cowboy programming looks cool

But often, in the Wild West…

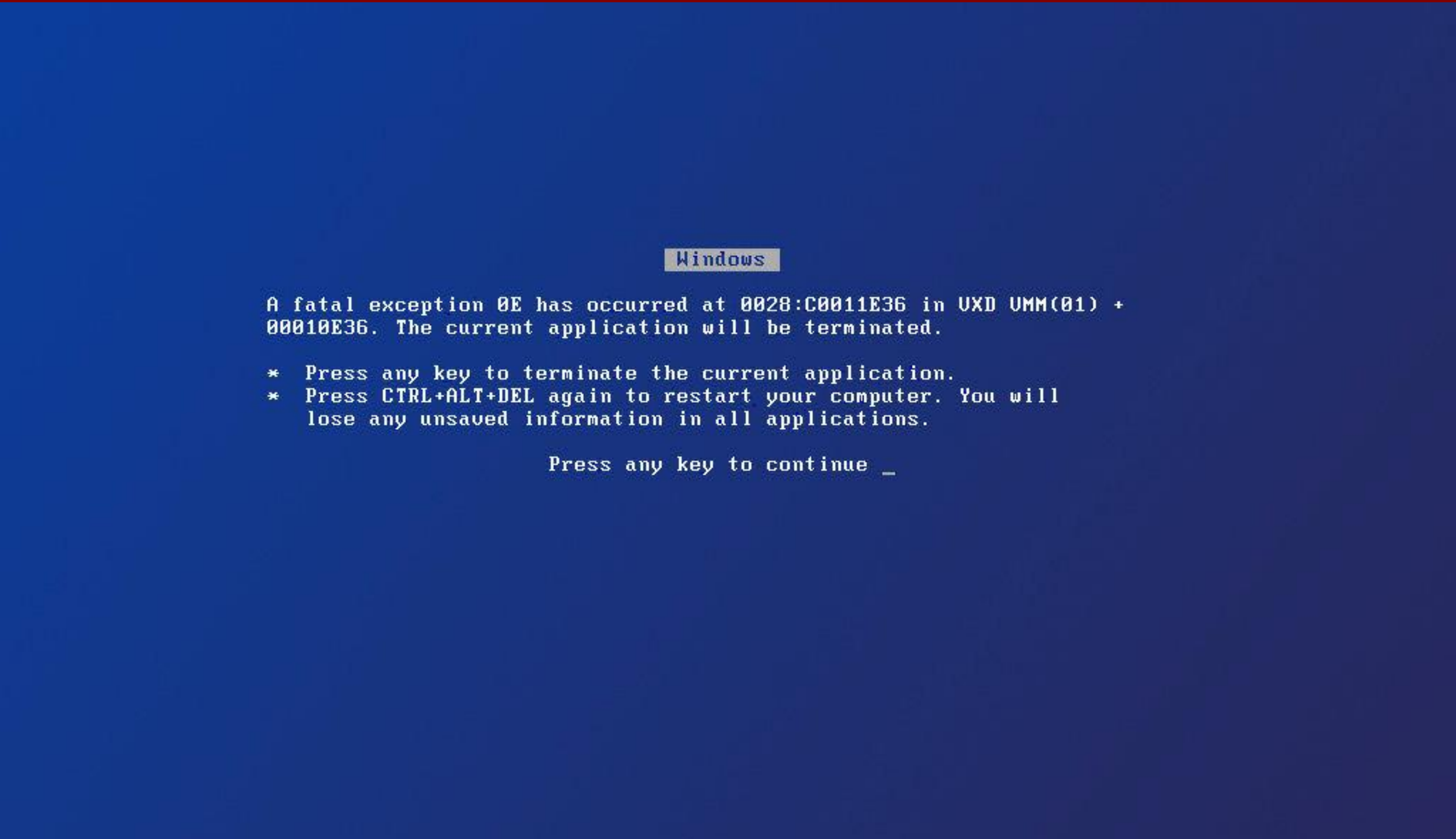
Getting these guarantees in unsafe languages

- Programming very, very carefully.
- Writing myriad tests with good coverage.
- Verifying the correctness of the code.
  - Fully automated techniques.
  - Interactive techniques.

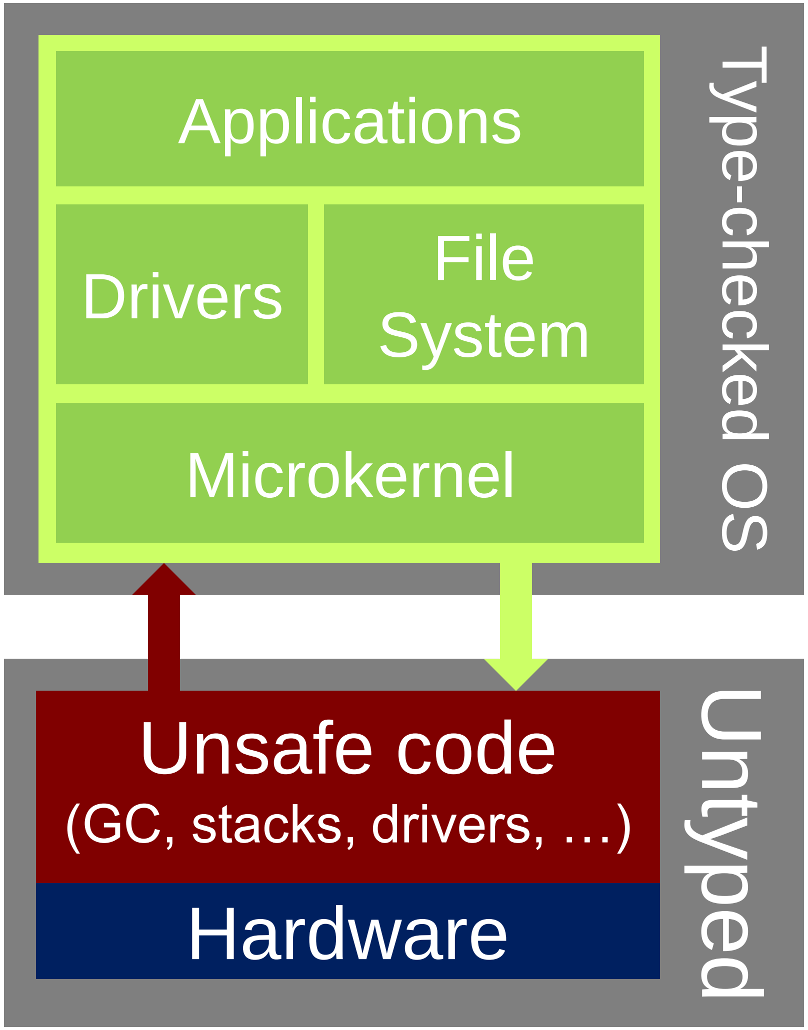
On the other hand…

Safe language provide guarantees about **every** program written in the language.

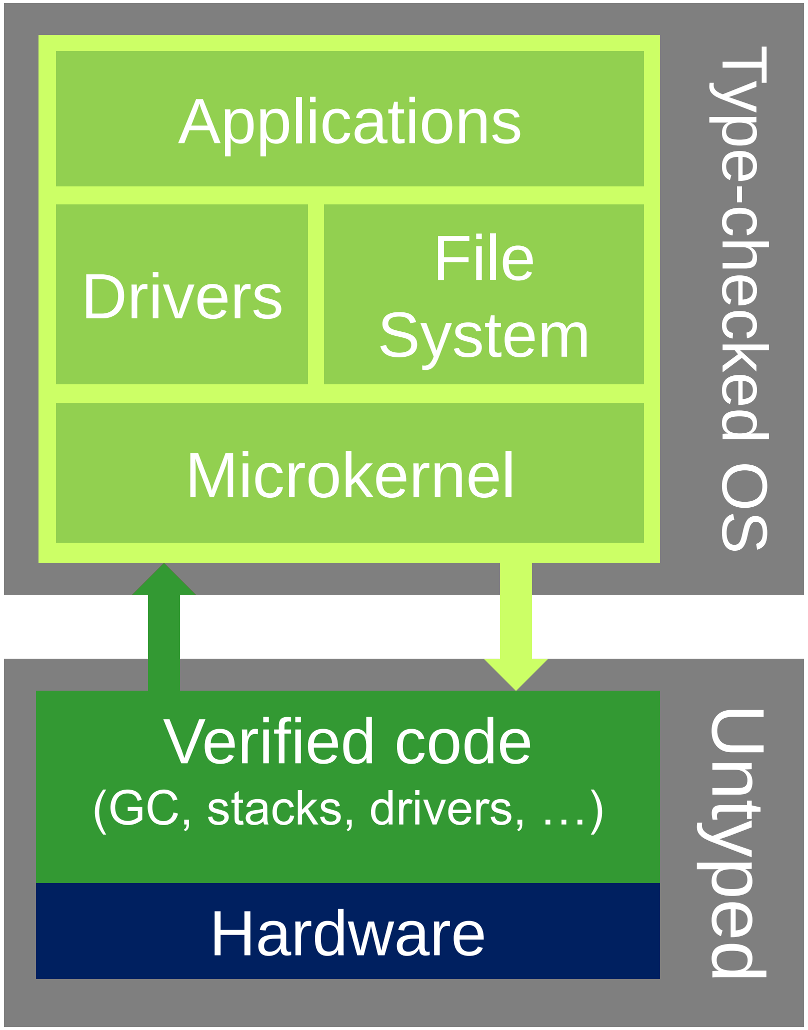
Case study: type-safe OSes



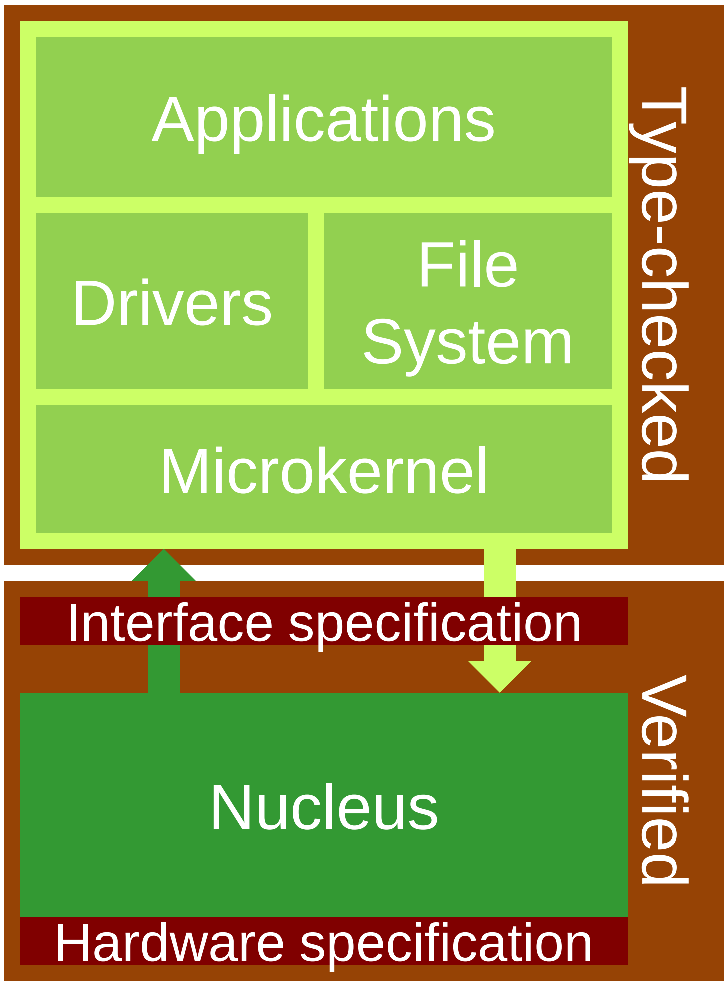
What if we used a safe language to build an OS?



Wanted: end-to-end type safety

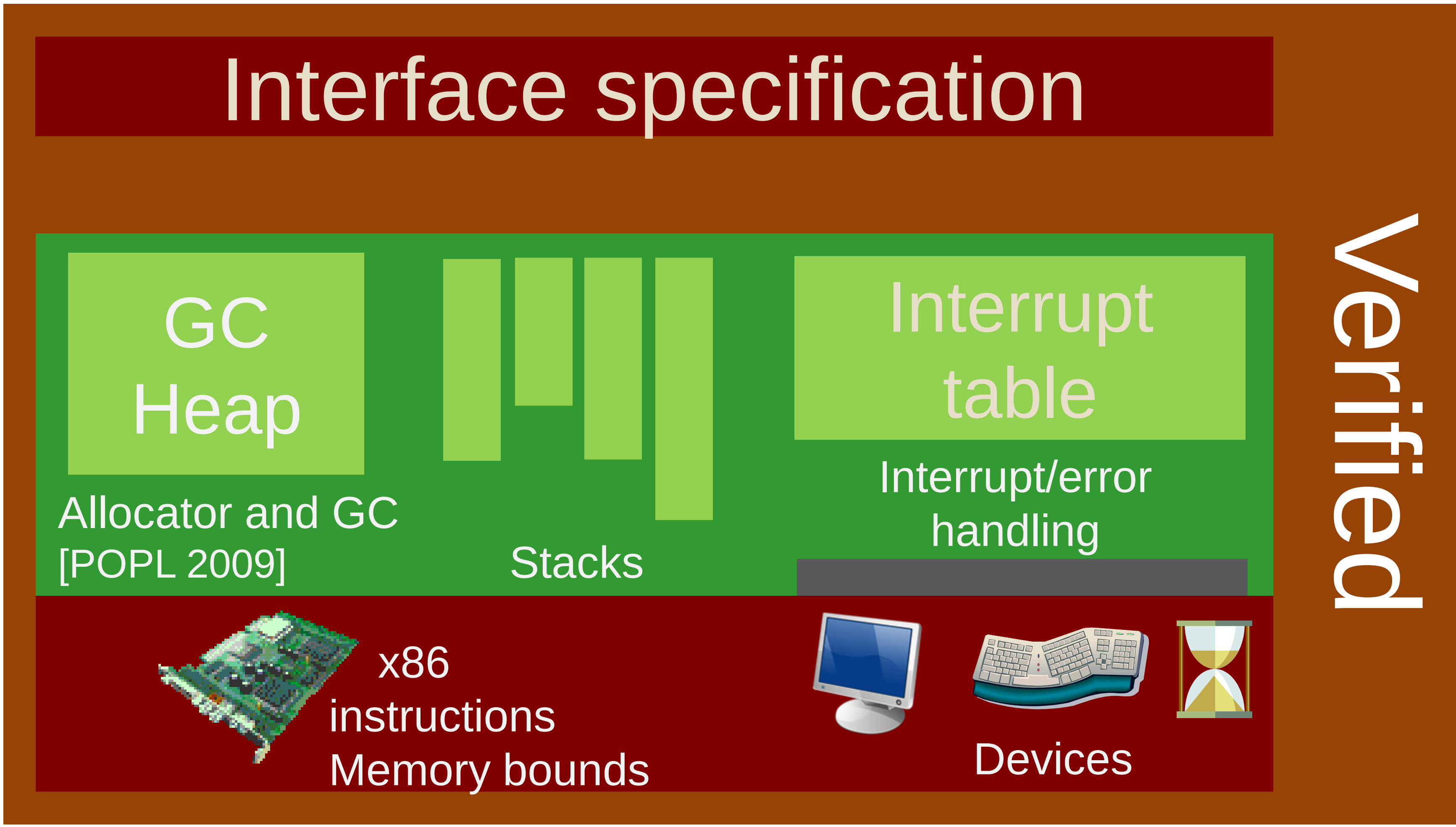


Verve, a type-safe OS
[Yang & Hawblitzel, 2009]



- Verify **partial correctness** of low-level **Nucleus** using Hoare logic based on a **hardware spec**.
- Verify an **interface to typed assembly** for end-to-end safety.

The Verve Nucleus



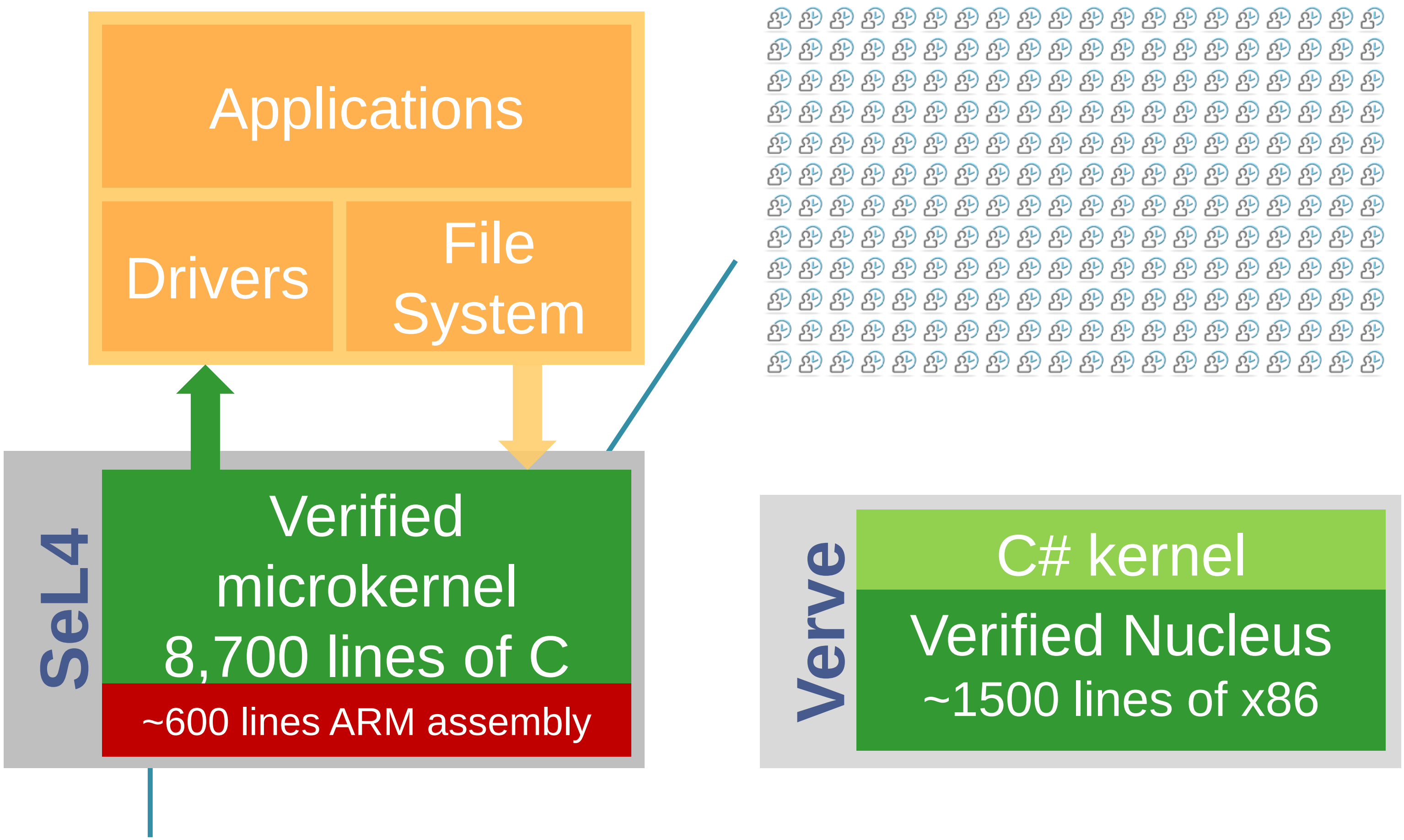
Thread Context Invariant

```
function StateInv
   (s:StackID, state:StackState, …)
    returns(bool) {
      (!IsEmpty(state) → …
  && (IsInterrupted(state) → …
  && (IsYielded(state) → …
    && state == StackYielded(
          StackEbp(s, tMems)
        , StackEsp(s, tMems) + 4
        , StackRA(s, tMems, fMems)) && …
}
```

“Load” Specification

```
procedure Load(ptr:int)
  returns (val:int);
  requires memAddr(ptr);
  requires Aligned(ptr);
  modifies Eip;
  ensures word(val);
  ensures val == Mem[ptr];
```

Verve vs. SeL4?



200,000 lines of Isabelle **20x code**

Takeaways

- **Safety** is an important property for security. Low-level languages like C are unsafe, so we have to be careful!
- Using **types** gives us lightweight way of getting safety.
- We can even get type safety in systems with low-level code, but this requires us to do some more heavyweight **verification**.

Assignment 1 is coming out!

- Will go out later today. We'll announce over Piazza.
- Task: implement an OCaml data server with a basic security policy.
- Develop a test suite to check compliance with the spec.
- Due in 2 weeks: 2/2/17.

Selfie from beginning of class, and answer to quiz question #1