

CSE 331

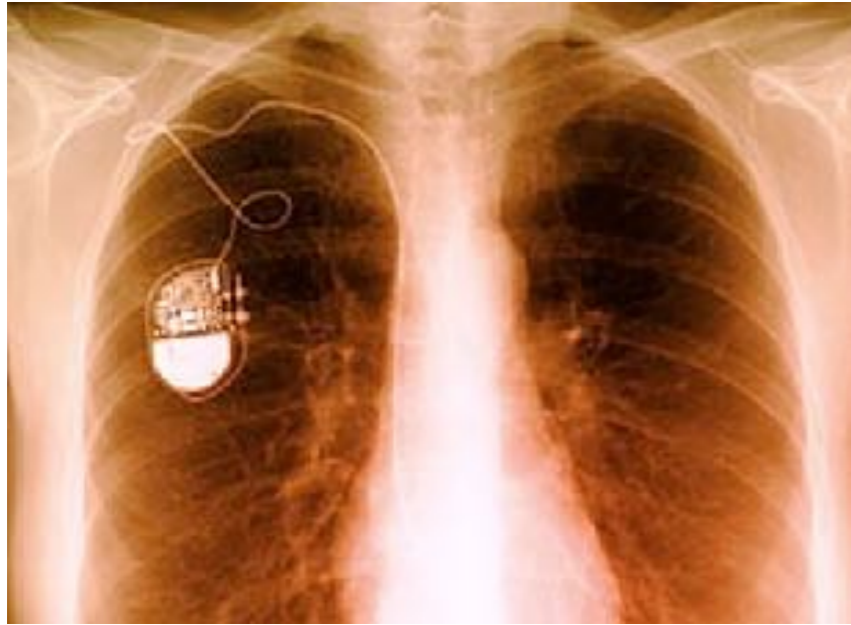
Software Design and Implementation

# Lecture 23

## *Verified Systems*

Zach Tatlock / Spring 2018

# Software Infrastructure



# Software Infrastructure is Shaky

## The New York Times

### Cars' Computer Systems Called at Risk to Hackers

By JOHN MARKOFF  
Published May 14, 2010

Automobiles, which will be increasingly in the near future, could be vulnerable to hackers, now, two teams of computer scientists at MIT presented next week.

Connect With Us on Social Media

@nytimescience on Twitter

Science Reporters and Editors on Twitter

Like the science desk on Facebook.



The scientists to remote functions was run security r

"We demand adversari automoti

including disabling the brakes, selectively the engine, and so on," they wrote in the [Modern Automobile.](#)

In the paper, which will be presented at Oakland, Calif., computer security specialist [University of California, San Diego](#), reported engineering in the design of their computers to the potential threat of hackers who may increasingly control modern cars.



### Medical Devices

Home Medical Devices Medical Device Safety

#### Medical Device Safety

##### Medical Device Recalls

- 2012 Medical Device Recalls
- 2011 Medical Device Recalls
- 2010 Medical Device Recalls
- 2009 Medical Device Recalls
- 2008 Medical Device Recalls
- 2007 Medical Device Recalls
- 2006 Medical Device Recalls
- 2001 - 2005 Medical Device Recalls

#### List of Device

FDA posts consumer the list because the

#### Recent Me

Listed by date p

##### Device No

##### Vascular S

##### Spacelabs

##### Service Kit

##### Symbios M

##### Pump/Kit, P

##### Ad-Tech M

##### Electrodes

##### Lumenis LI

##### DePuy Orth

##### GE Healthcare, LLC, Giraffe and Panda T-Piece Resuscitation

##### Systems, and the Giraffe and Panda Bag and Mask Resuscitation

##### Systems

##### St. Jude Medical, AMPLATZER TorqVue FX Delivery System

##### Hamilton Medical, Inc., HAMILTON-T1 Ventilators with Software

##### Vycor Medical, Inc., Vycor ViewSite Brain Access System (VBAS)

##### Bausch and Lomb 27G Sterile Cannula Packed in Bausch and Lomb

##### Anvisic 1.2% Sodium Hyaluronate (Model 59051, 59001, 59051L,

##### 59001L) and Anvisic Plus 1.6% Sodium Hyaluronate (Model 59004,

## Bloomberg Businessweek Markets & Finance

### Software Bug Made Swedish Exchange Go Bork, Bork, Bork

By Karen Weise on November 29, 2012

A computer error stalks the markets—again. An order on a relatively obscure derivatives index in Stockholm yesterday was asking to buy futures contracts on Swedish stocks valued at **131 times the country's entire GDP**. The order made the exchange go “bananas” and caused Nasdaq OMX to stop trading in Swedish derivatives for four hours.

This was no “fat finger” incident, where a trader accidentally types an extra few digits or the wrong numbers in an order. Instead, a software glitch magnified an order, Nasdaq OMX spokesman Carl Norell told Bloomberg News. “Our system misinterpreted a certain order category and communicated a value that was way too high into the book,” he said.

The interruption was in a small corner of the market, but it's just the latest in a string of technical problems that have halted trading. As more trading is driven by the algorithms of high-frequency traders, one glitch or bad order can spark major disruptions. The 2010 flash crash caused \$862 billion in stock values to vanish from the market temporarily, and technical problems have limited other

# Software Infrastructure is Shaky

## The New York Times

### Cars' Computer Systems

By JOHN MARKOFF  
Published May 14, 2013

Automobiles, which will be increasingly in the near future, could be vulnerable to hackers. Now, two teams of computer scientists at MIT and Stanford presented next week.

#### Connect With Us on Social Media

@nytimescience on Twitter

Science Reporters and Editors on Twitter

Like the science desk on Facebook.



The scientists to remote functions was run security r

"We demonstrate adversarial automoti

including disabling the brakes, selectively the engine, and so on," they wrote in the [Modern Automobile.](#)

In the paper, which will be presented at Oakland, Calif., computer security specialist [University of California, San Diego](#), report engineering in the design of their computers to the potential threat of hackers who may increasingly control modern cars.

## Bloomberg Businessweek Markets & Finance

### Made Swedish Bork, Bork, Bork

markets—again. An order on a relatively obscure market yesterday was asking to buy futures contracts on [times the country's entire GDP](#). The order made the Nasdaq OMX to stop trading in Swedish

ent, where a trader accidentally types an extra few in an order. Instead, a software glitch magnified an order. Carl Norell told Bloomberg News. "Our system category and communicated a value that was way id.

all corner of the market, but it's just the latest in a series of events that have halted trading. As more trading is driven by high-frequency traders, one glitch or bad order can spark a flash crash caused \$862 billion in stock values to drop temporarily, and technical problems have halted other



Systems, and the Oracle and Pines bug and other vulnerabilities	02/11/13
St. Jude Medical, AMPLATZER TorqVue FX Delivery System	02/12/13
Hamilton Medical, Inc., HAMILTON-T1 Ventilators with Software Versions 1.1.2 and Lower	02/07/13
Vycor Medical, Inc., Vycor Viewsite Brain Access System (VBAS)	01/30/13
Bausch and Lomb 27G Sterile Cannula Packed in Bausch and Lomb Amvisc 1.2% Sodium Hyaluronate (Model 59051, 59081, 59051L, 59081L) and Amvisc Plus 1.6% Sodium Hyaluronate	01/23/13

# Software Infrastructure is Shaky

The New York Times

Bloomberg Businessweek  
Markets & Finance

Cars' Co

By JOHN MARKO  
Published May 14

Automobiles  
the near future  
now, two teams  
presented ne

Connect With  
Us on Social  
Media

@nytimescience  
Twitter.

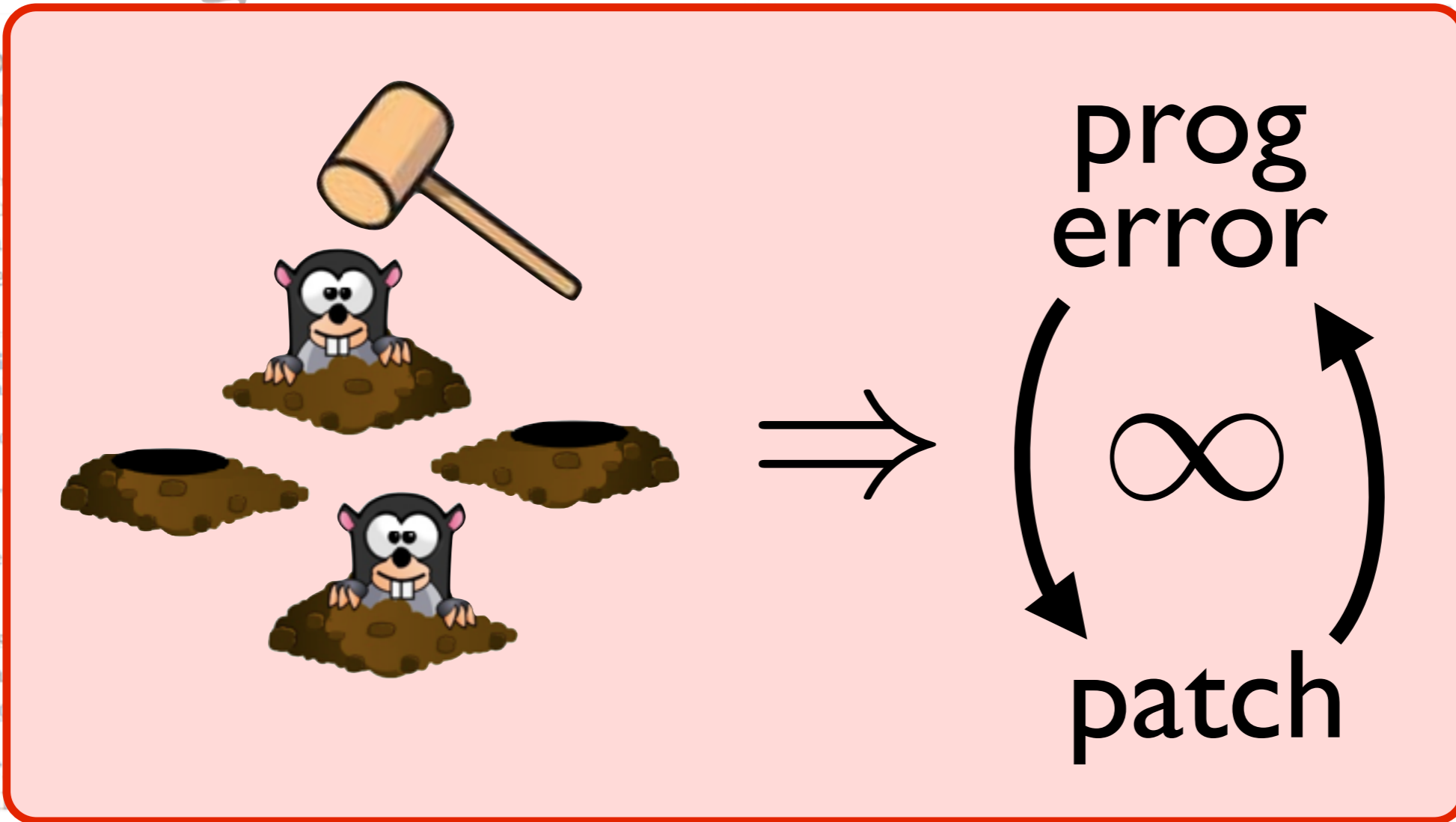
Science Report  
Twitter

Like the science

including dis  
the engine, a  
Modern Auto

In the paper  
Oakland, Ca

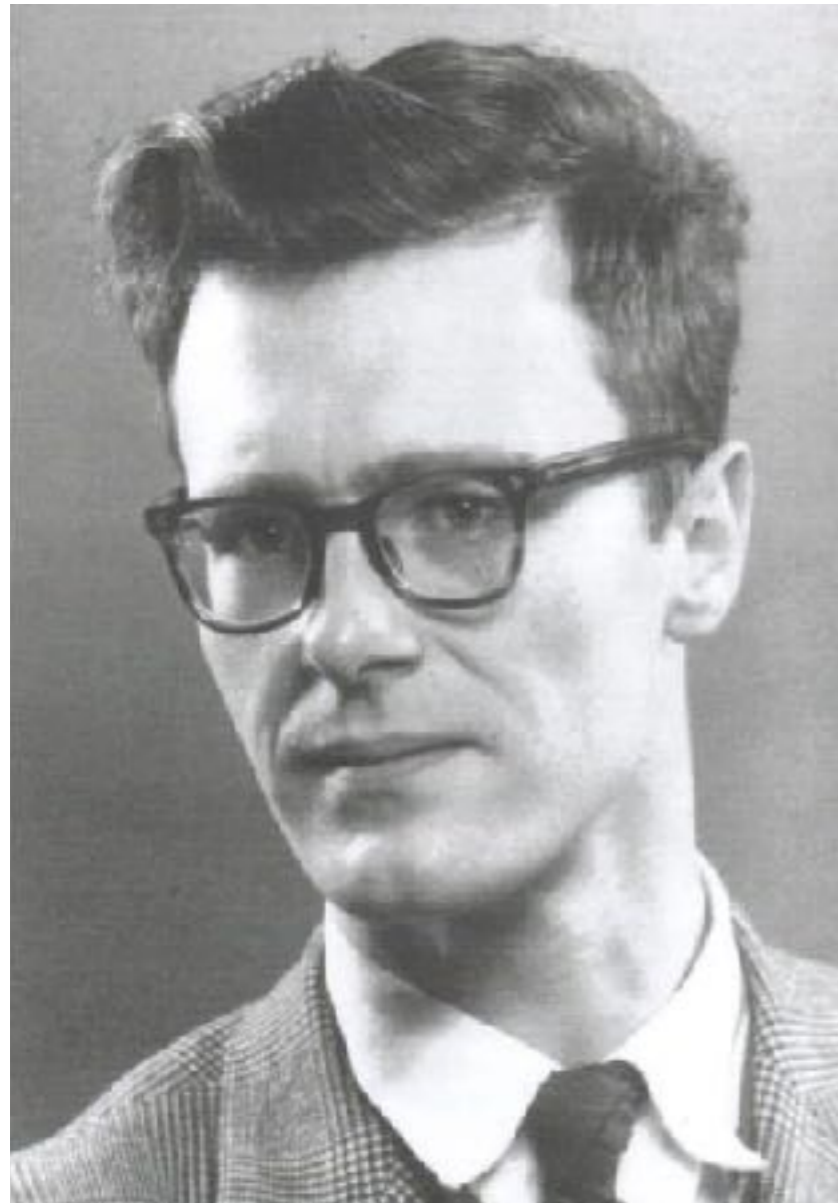
University of California, San Diego, reported  
engineering in the design of their computer  
to the potential threat of hackers who may  
increasingly control modern cars.



rk  
relatively obscure  
futures contracts on  
. The order made the  
ing in Swedish  
types an extra few  
glitch magnified an  
News. "Our system  
a value that was way  
s just the latest in a  
re trading is driven  
d order can spark  
in stock values to

Systems, and the Oracle and Puma Bug and IBM Red Hat Linux Systems	
St. Jude Medical, AMPLATZER TorqVue FX Delivery System	02/12/13
Hamilton Medical, Inc., HAMILTON-T1 Ventilators with Software Versions 1.1.2 and Lower	02/07/13
Vycor Medical, Inc., Vycor Viewsite Brain Access System (VBAS)	01/30/13
Bausch and Lomb 27G Sterile Cannula Packed in Bausch and Lomb Amvisc 1.2% Sodium Hyaluronate (Model 59051, 59081, 59051L, 59081L) and Amvisc Plus 1.6% Sodium Hyaluronate	01/23/13

*When exhaustive testing is impossible,  
our trust can only be based on proof.*



## **Edsger W. Dijkstra**

Under the Spell of Leibniz's Dream

Reports and Articles

### **Social Processes and Proofs of Theorems and Programs**

Richard A. De Millo  
Georgia Institute of Technology

Richard J. Lipton and Alan J. Perlis  
Yale University

proofs won't happen

**... not just a dream!**

# Proof Assistant Based Verification

Code in language suited for reasoning

Develop correctness proof in synch

Fully formal, *machine checkable* proof

# Proof Assistant Based Verification

Verified Compiler: **CompCert** [*Leroy POPL 06*]

<i>Compiler</i>	<i>Bugs Found</i>
GCC	122
LLVM	181
CompCert	?

[*Yang et al. PLDI 11*]



# Proof Assistant Based Verification

Verified Compiler: **CompCert** [Leroy POPL 06]

<i>Compiler</i>	<i>Bugs Found</i>
GCC	122
LLVM	181
CompCert	0

[Yang et al. PLDI 11]

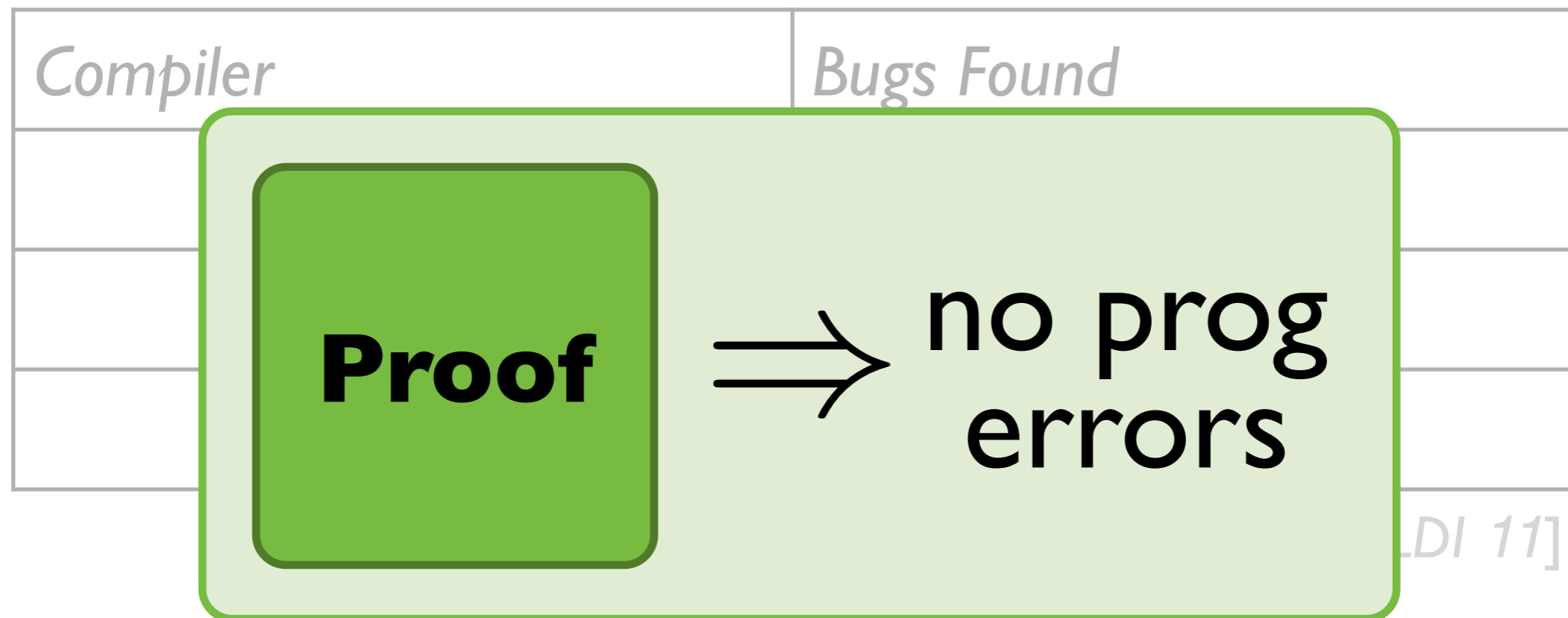
[Vu et al. PLDI 14]

Verified OS kernel: **seL4** [Klein et al. SOSP 09]

*realistic implementation guaranteed bug free*

# Proof Assistant Based Verification

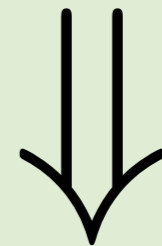
Verified Compiler: **CompCert** [Leroy POPL 06]



Verified OS kernel: **seL4** [Klein et al. SOSPP 09]

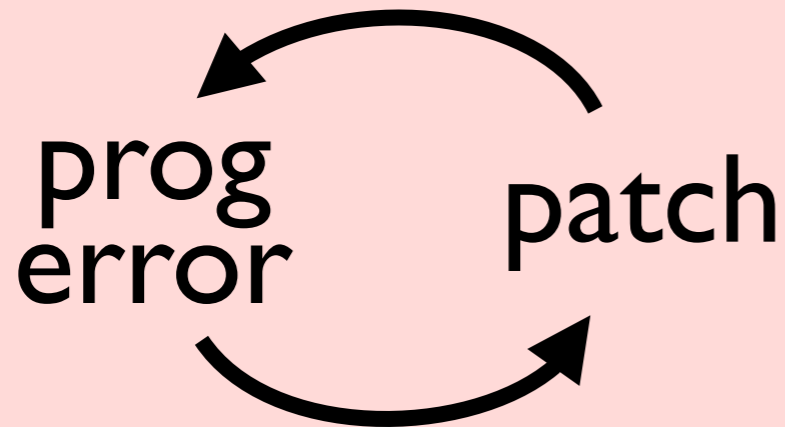
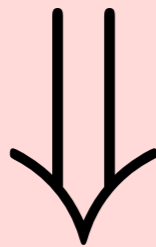
*realistic implementation guaranteed bug free*

# Promise

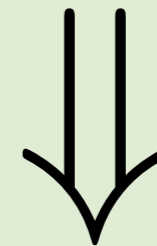


no prog  
errors

# Today

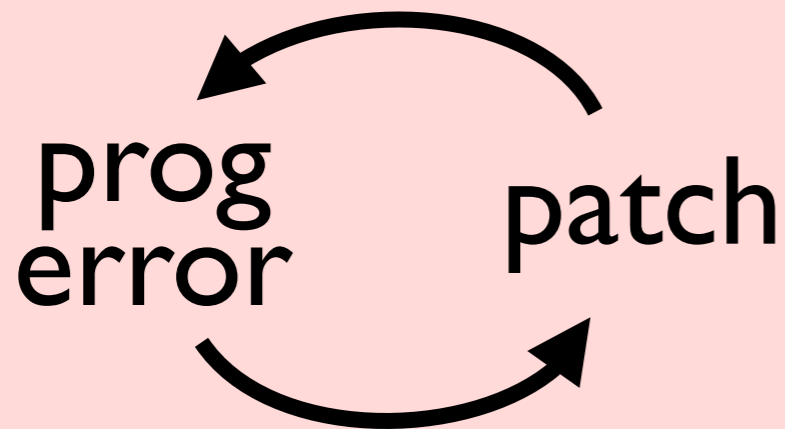
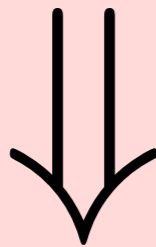


# Promise

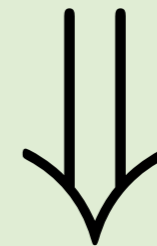


no prog  
errors

# Today

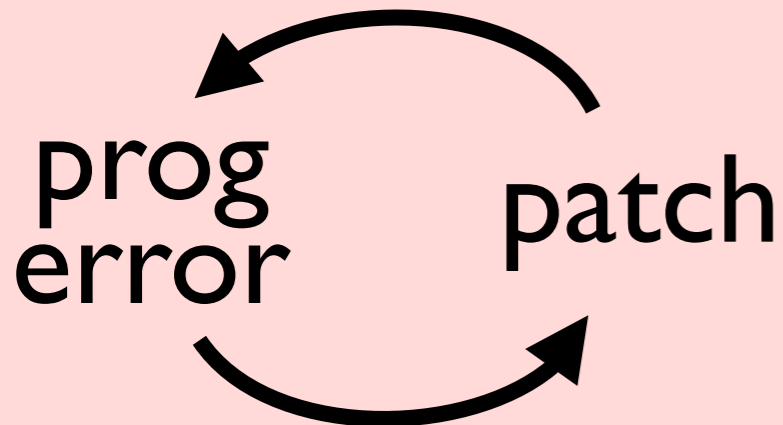
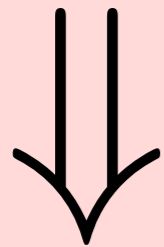


# Promise



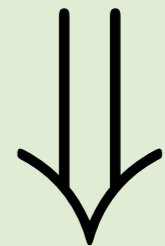
no prog  
errors

# Today



# Proof Burden

# Promise



no prog errors

# The Burden of Proof

## 1. Initial proofs require heroic effort

*CompCert: 70% proof, vast majority of effort*

*seL4: 200,000 line proof for 9,000 lines of C*

## 2. Code updates require re-proving

*CompCert: adding opts [Tristan POPL 08, PLDI 09, POPL 10]*

*seL4: changing RPC took 17% of proof effort*

# Mitigating the Burden of Proof

1: Scaling proofs to critical infrastructure

➔ *Formal shim verification for large apps*

*QUARK: browser with security guarantees*

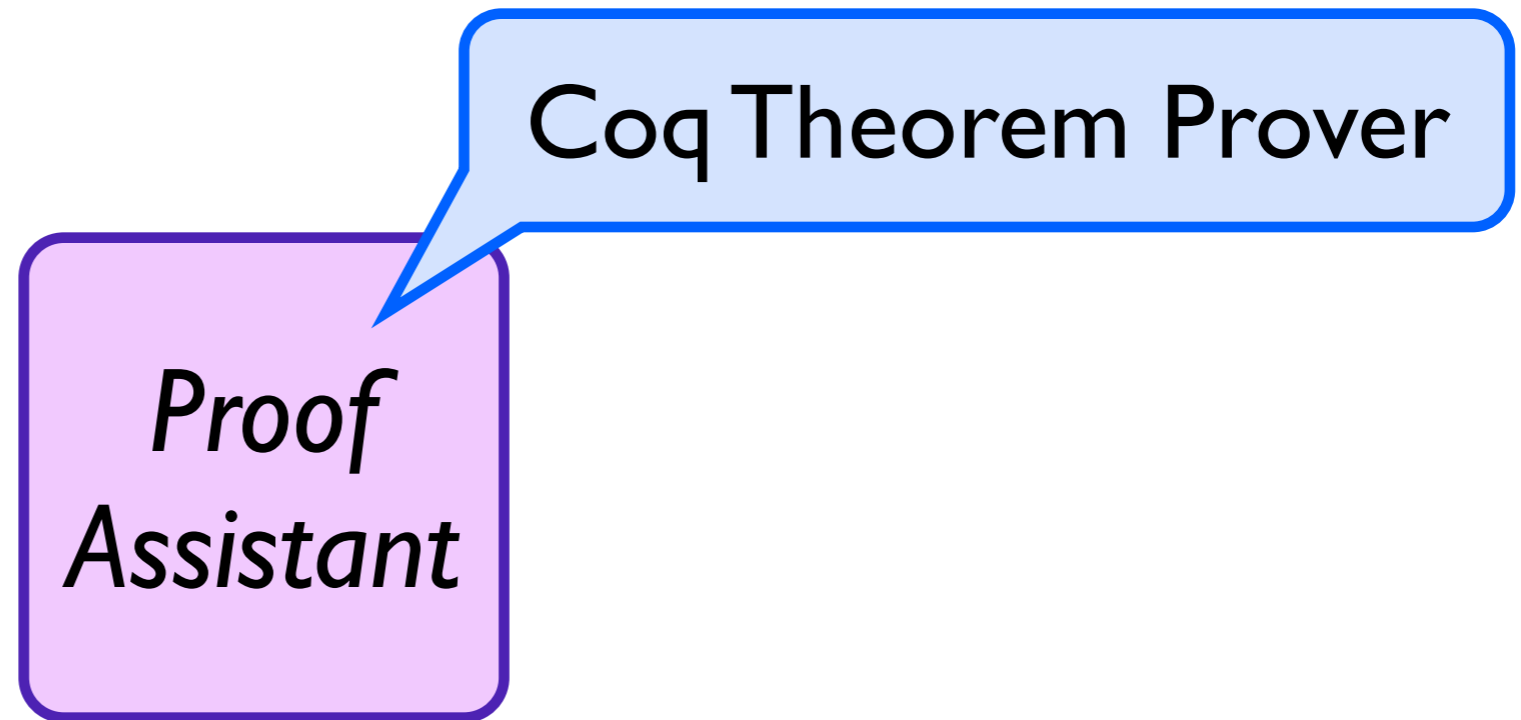
2: Evolving formally verified systems

*Reflex DSL exploits domain for proof auto*

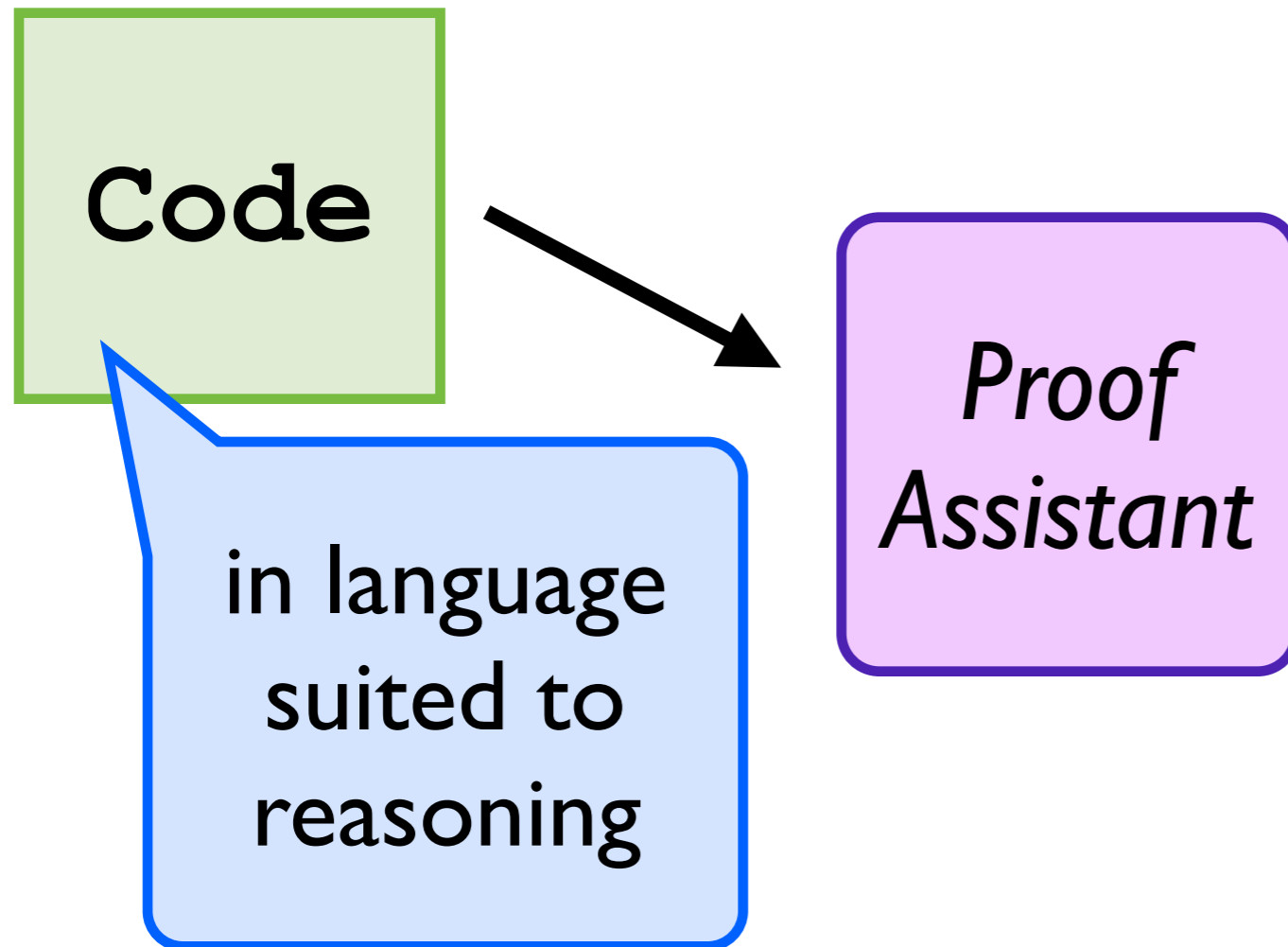


# Fully Formal Verification

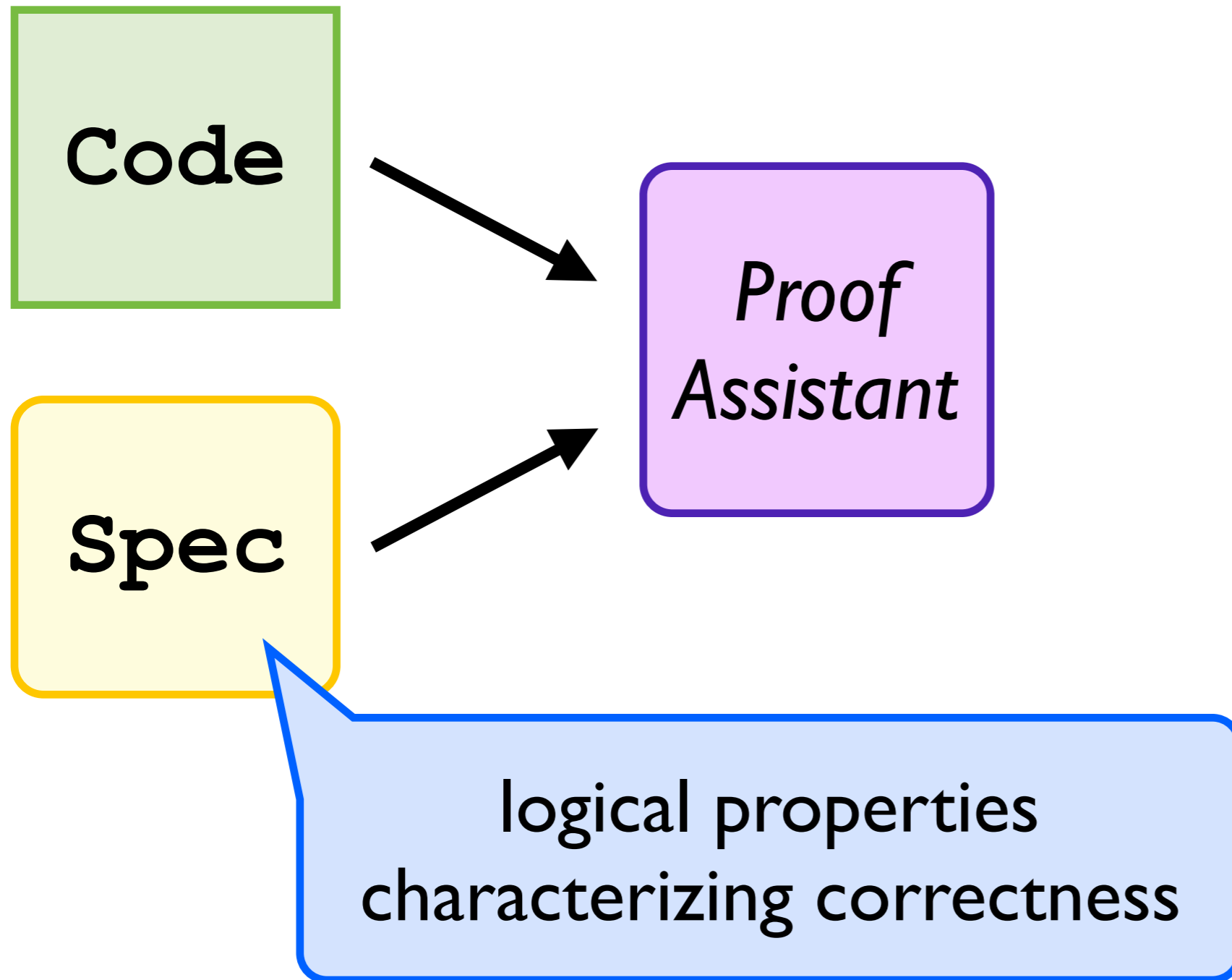
# Fully Formal Verification



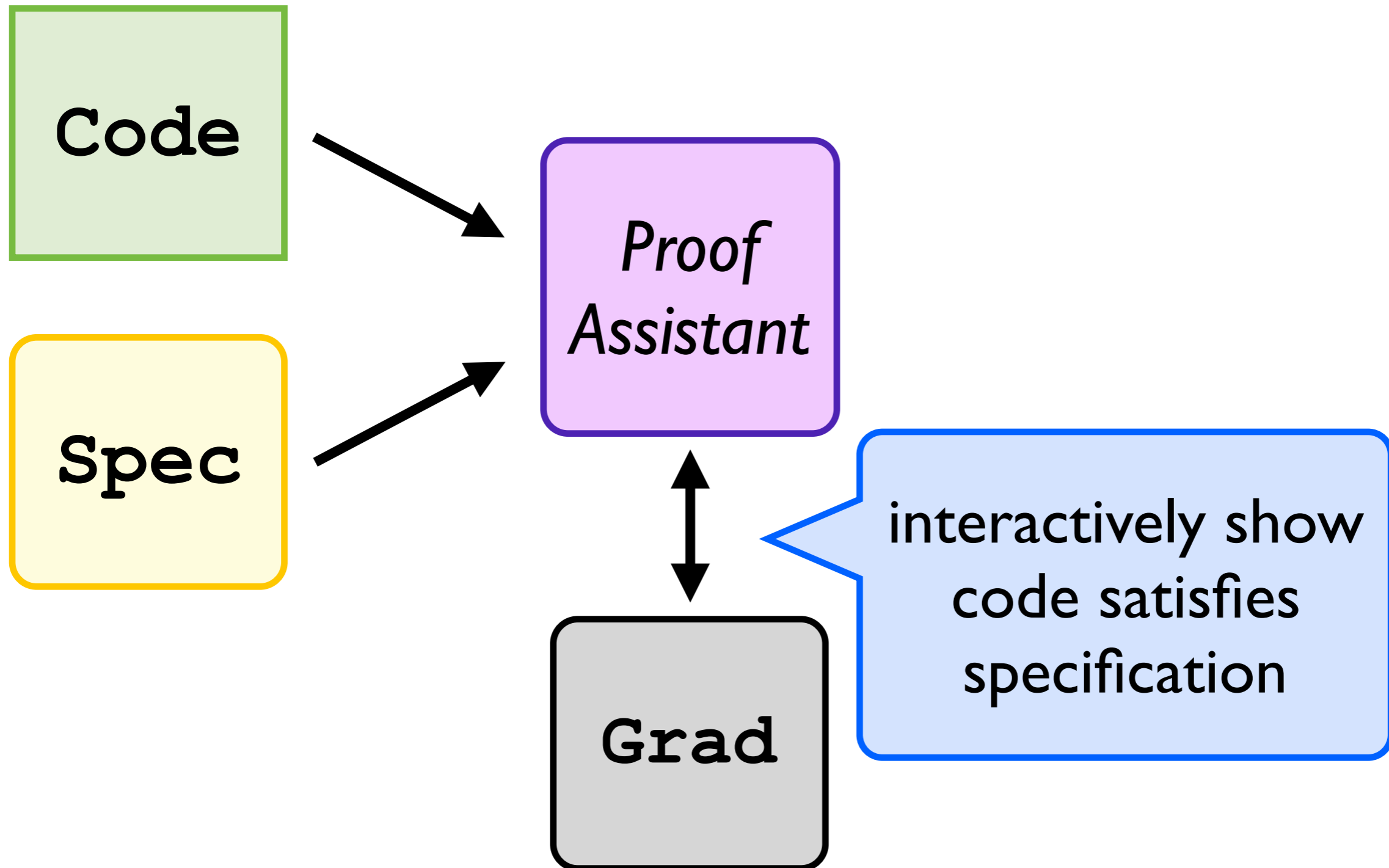
# Fully Formal Verification



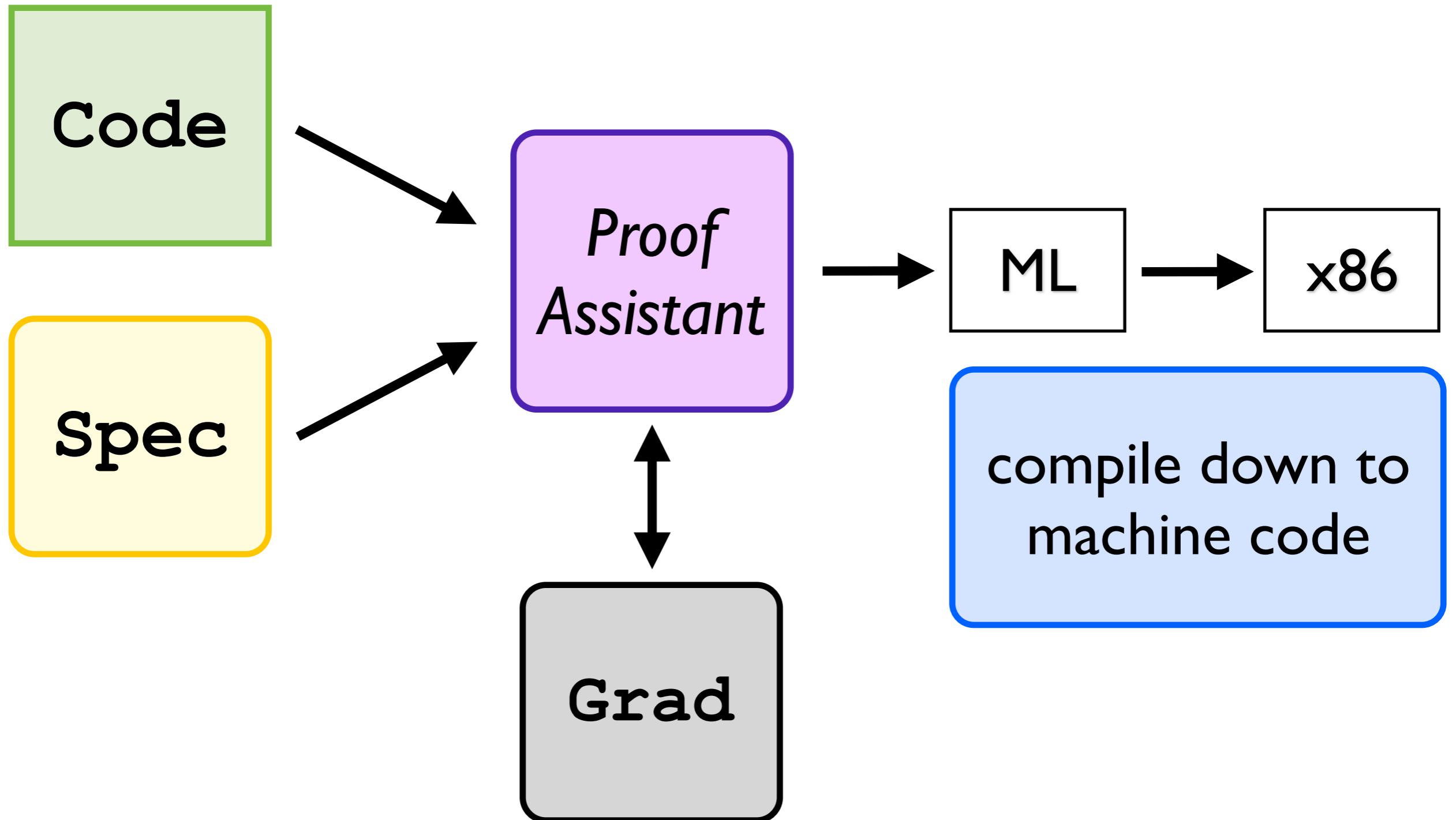
# Fully Formal Verification



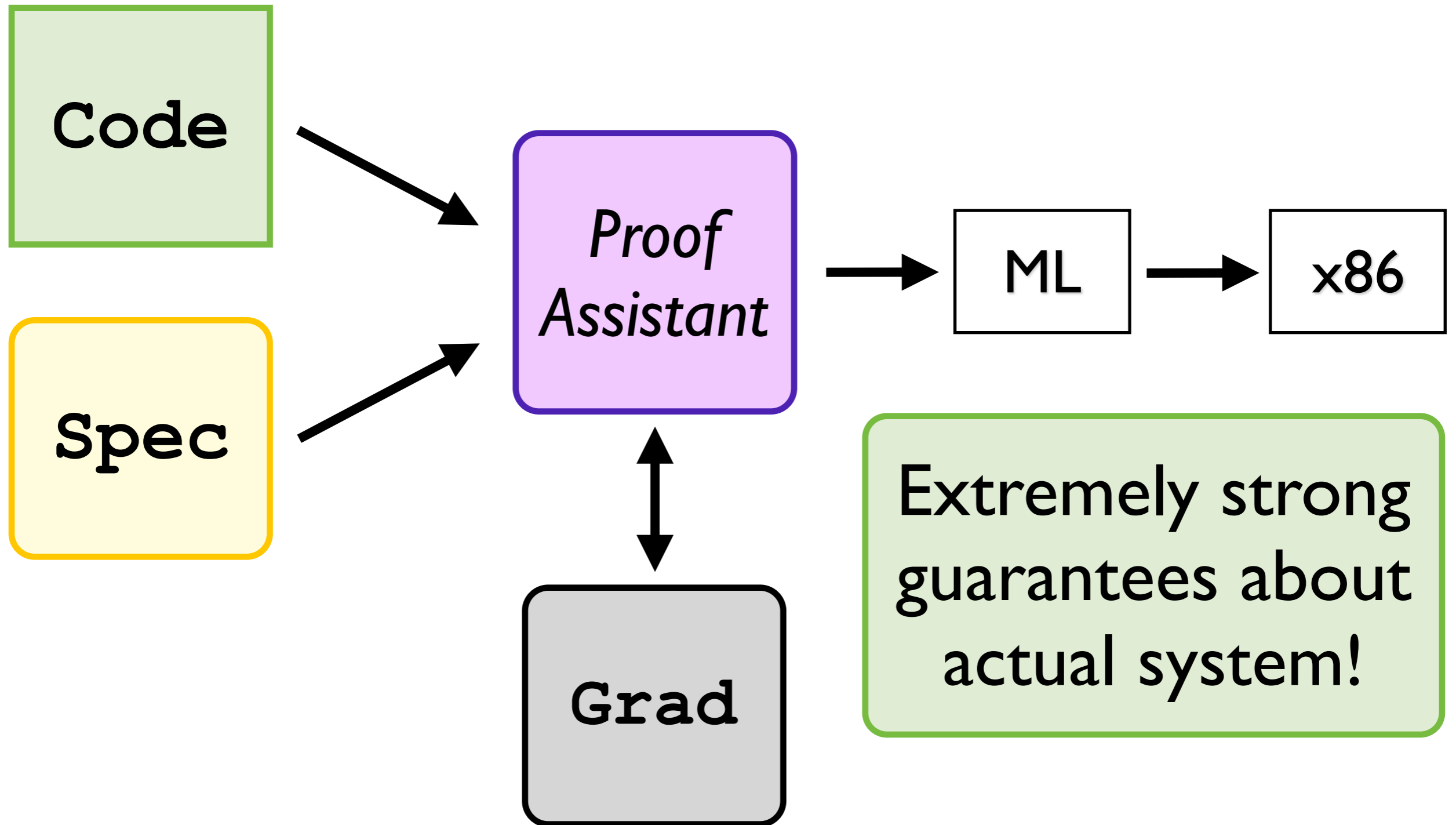
# Fully Formal Verification



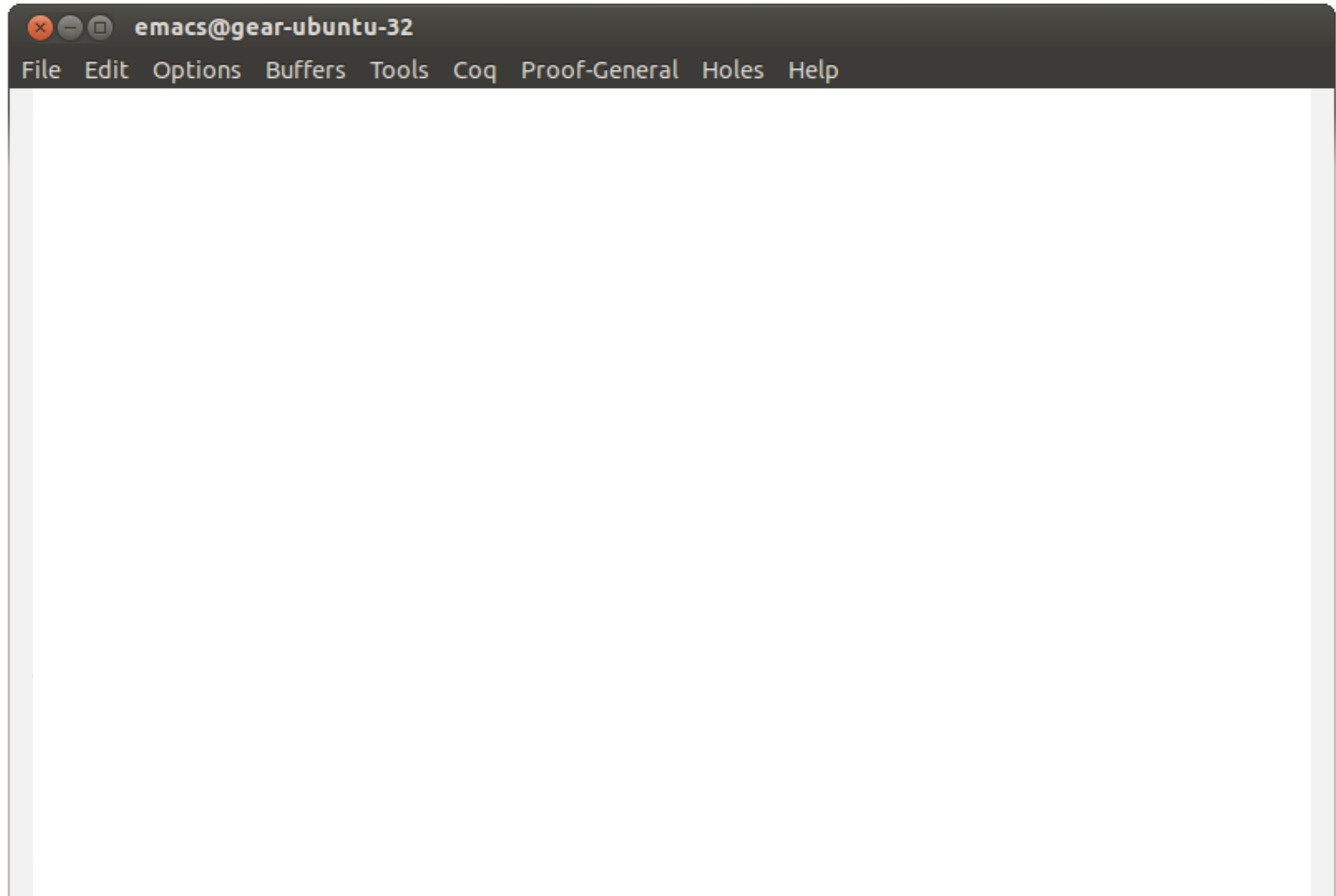
# Fully Formal Verification



# Fully Formal Verification



# Fully Formal Verification





# Fully Formal Verification

```
emacs@gear-ubuntu-32
File Edit Options Buffers Tools Coq Proof-General

Fixpoint factorial n :=
  match n with
  | 0    => 1
  | S m => n * factorial m
end.
```

program in a purely functional language

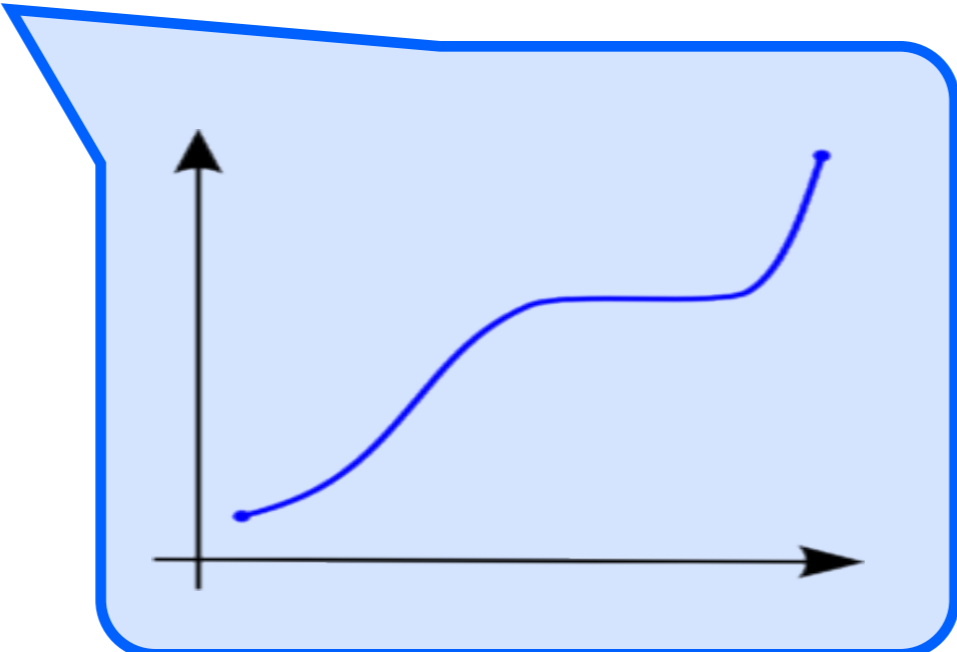
# Fully Formal Verification

```
emacs@gear-ubuntu-32
File Edit Options Buffers Tools Coq Proof-General Holes Help

Fixpoint factorial n :=
  match n with
  | 0   => 1
  | S m => n * factorial m
end.

Definition monotonic f :=
  forall a b,
  a <= b ->
  f a <= f b.
```

specification characterizes desired behavior



# Fully Formal Verification

```
emacs@gear-ubuntu-32
File Edit Options Buffers Tools Coq Proof-General Holes Help

Fixpoint factorial n :=
  match n with
  | 0    => 1
  | S m => n * factorial m
  end.

Definition monotonic f :=
  forall a b,
  a <= b ->
  f a <= f b.

Theorem example :
  monotonic factorial.
Proof.
  ...
```

claim program satisfies spec

construct proof *interactively*

# Fully Formal Verification

```
emacs@gear-ubuntu-32
File Edit Options Buffers Tools Coq Proof-General Holes Help
► Fixpoint factorial n :=
  match n with
  | 0   => 1
  | S m => n * factorial m
  end.

Definition monotonic f :=
  forall a b,
  a <= b ->
  f a <= f b.

Theorem example :
  monotonic factorial.
Proof.
  unfold monotonic. intros n1 n2 H.
  induction H. apply le_refl. simpl.
  apply le_trans with (m := factorial m); auto.
  destruct (mult_0_le (factorial m) m).
  rewrite H0; simpl. apply le_refl.
  apply le_trans with (m := m * factorial m); auto.
  rewrite plus_n_0 at 1. rewrite plus_comm.
  apply plus_le_compat. apply le_0_n. apply le_refl.
Qed.
```

# Fully Formal Verification

browsers don't look like factorial

```
Fixpoint factorial n :=  
  match n with  
  | 0   => 1  
  | S m => n * factorial m  
  end.
```

browsers don't have simple specs

```
Definition monotonic f :=  
  forall a b,  
  a <= b ->  
  f a <= f b.
```

even easy proofs grow quickly and become opaque

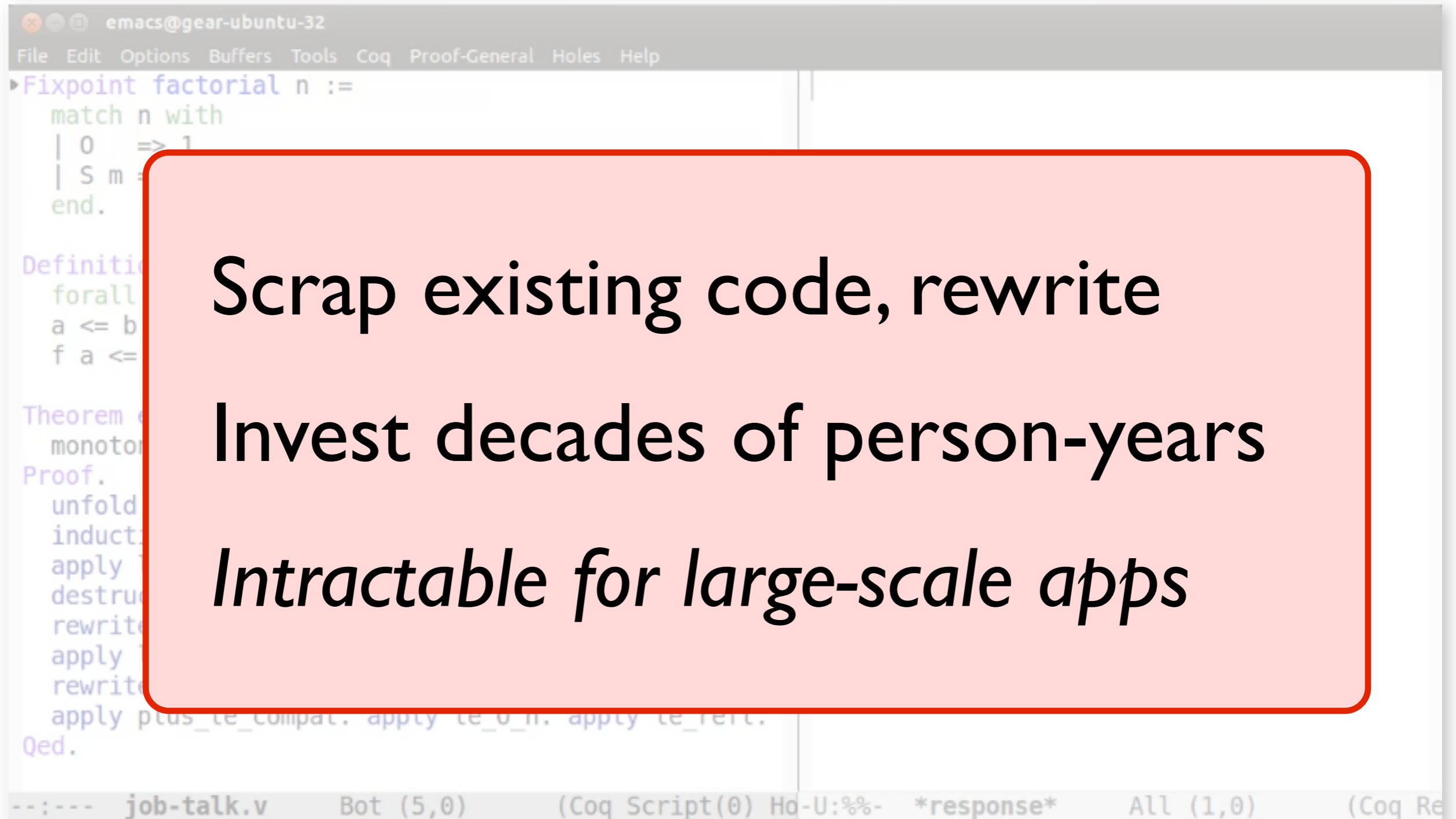
```
Theorem example :  
  monotonic factorial.
```

```
Proof.
```

```
unfold monotonic. intros n1 n2 H.  
induction H. apply le_refl. simpl.  
apply le_trans with (m := factorial  
destruct (mult_0_le (factorial m)  
rewrite H0; simpl. apply le_refl.  
apply le_trans with (m := m * factor  
rewrite plus_n_0 at 1. rewrite plus  
apply plus_le_compat. apply le_0_n.
```

```
Qed.
```

# Fully Formal Verification



emacs@gear-ubuntu-32  
File Edit Options Buffers Tools Coq Proof-General Holes Help

```
► Fixpoint factorial n :=  
  match n with  
  | 0 => 1  
  | S m => m * factorial m  
  end.  
  
Definition  
forall  
a <= b  
f a <=
```

**Scrap existing code, rewrite**  
**Invest decades of person-years**  
***Intractable for large-scale apps***

Theorem  
monoton  
Proof.  
unfold  
induct  
apply  
destru  
rewrit  
apply  
rewrit  
apply plus\_te\_compat. apply te\_0\_n. apply te\_refl.  
Qed.

--:-- job-talk.v Bot (5,0) (Coq Script(0) Ho-U:%%- \*response\* All (1,0) (Coq Re

# Formally Verify a Browser?!

# Formally Verify a Browser?!

Millions of LOC



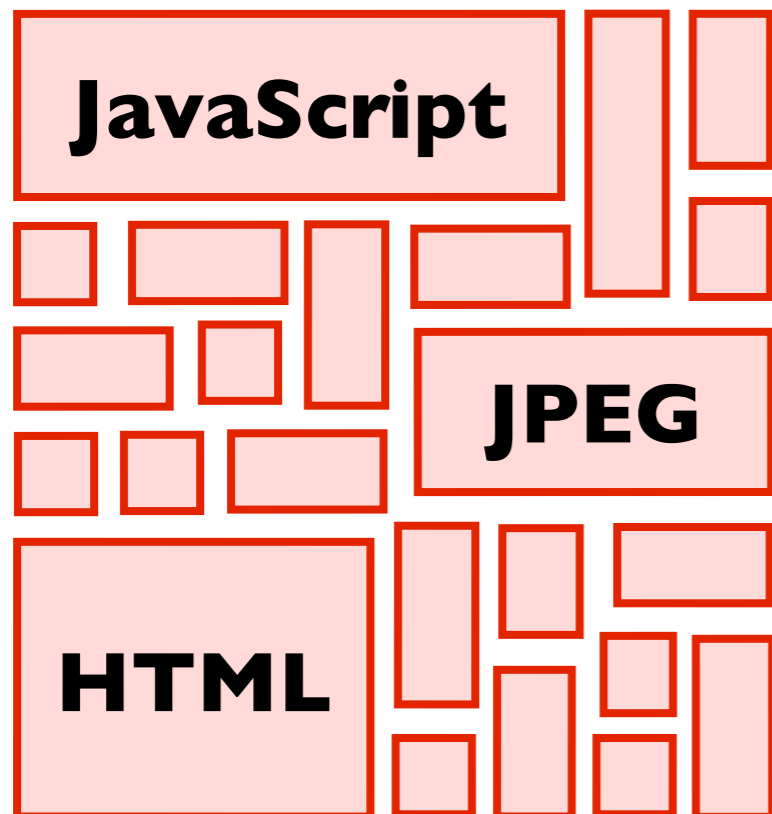
**Web  
Browser**



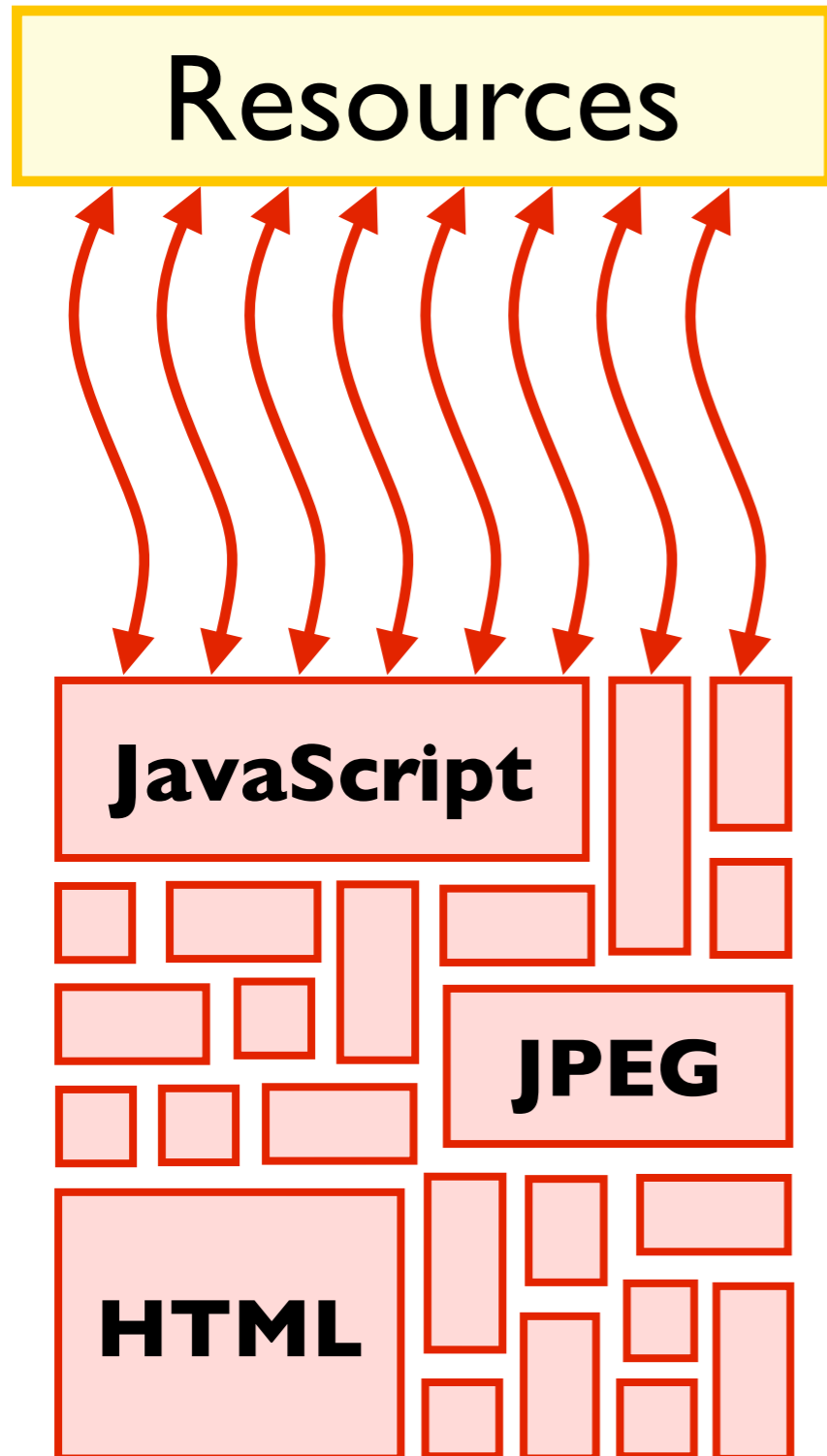
# Formally Verify a Browser?!

Millions of LOC

High performance



# Formally Verify a Browser?!

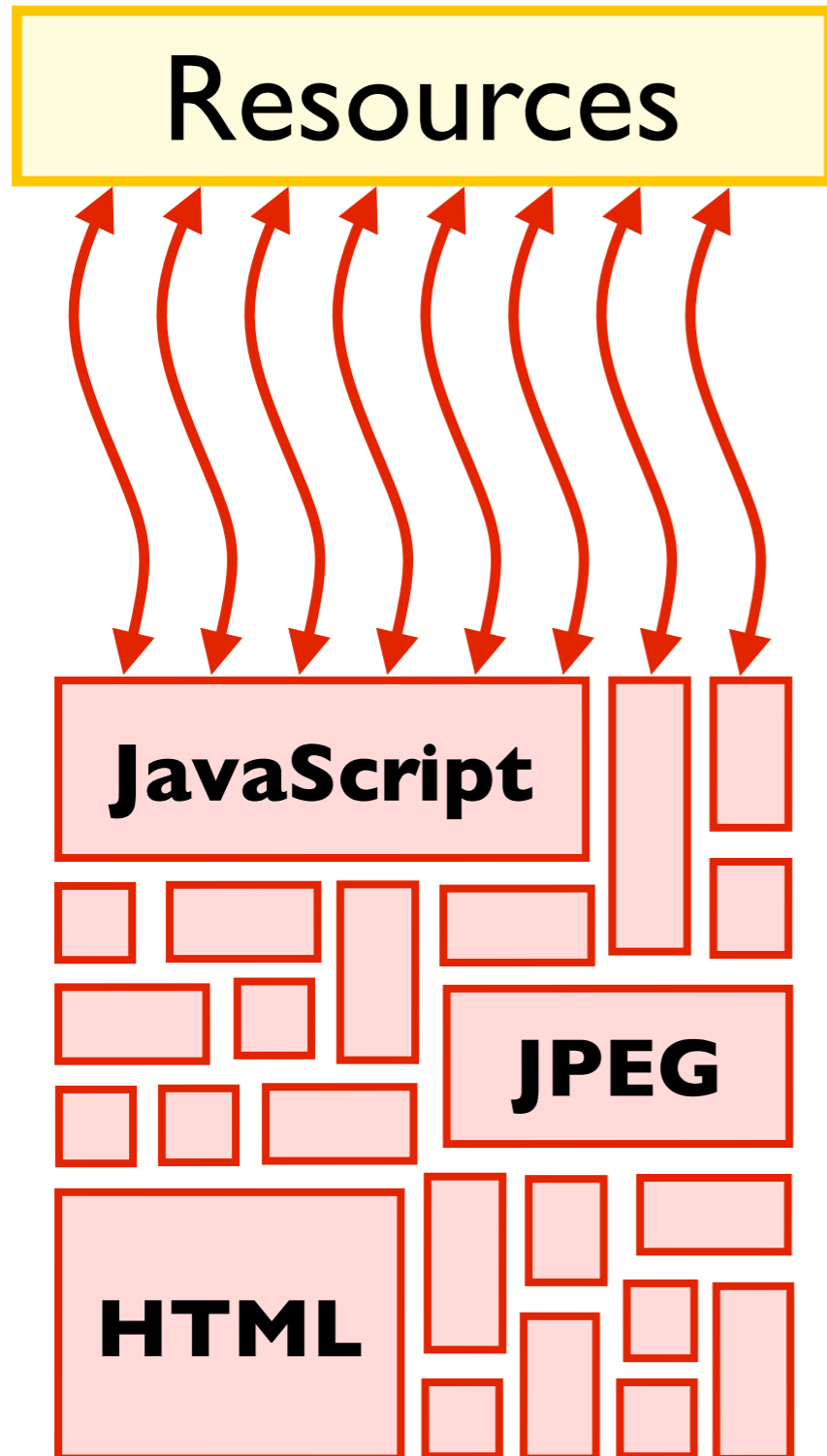


Millions of LOC

High performance

Loose access policy

# Formally Verify a Browser?!



Millions of LOC

High performance

Loose access policy

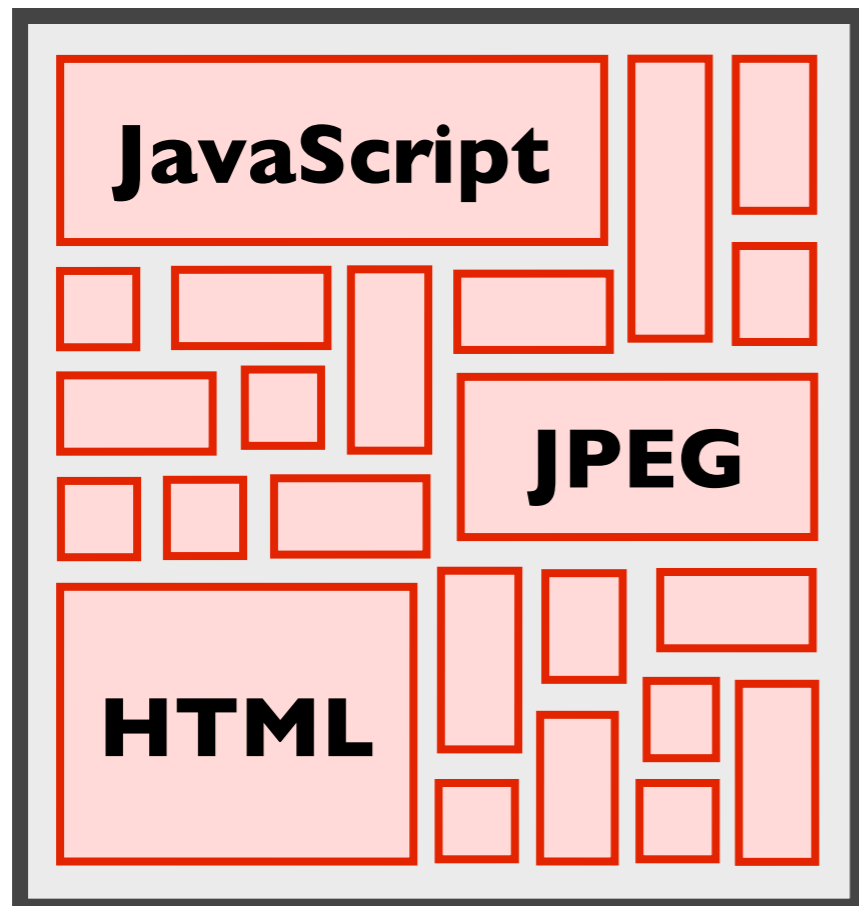
Constant evolution

# Formally Verify a Browser?!

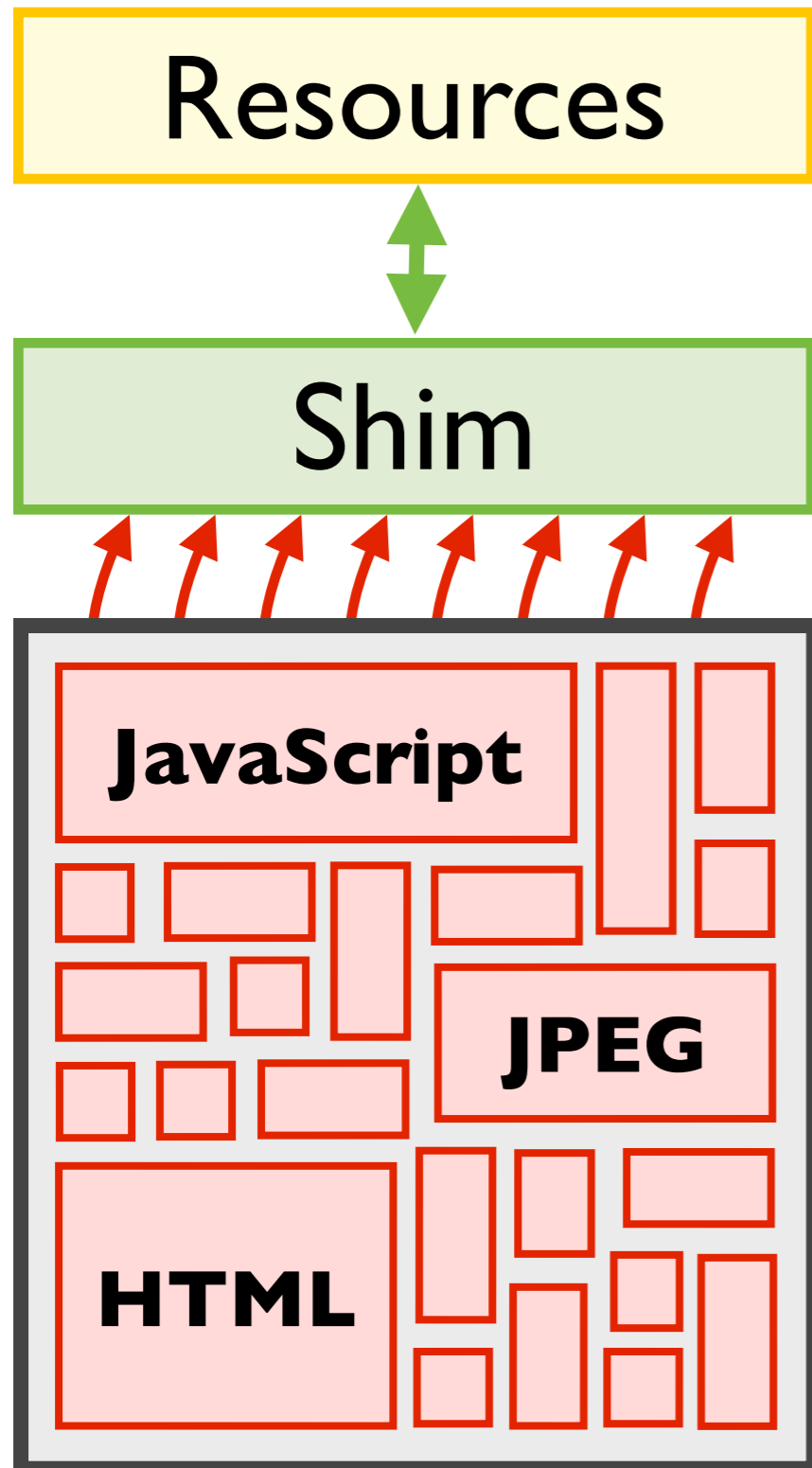
Resources

Isolate

*sandbox untrusted code*



# Formally Verify a Browser?!



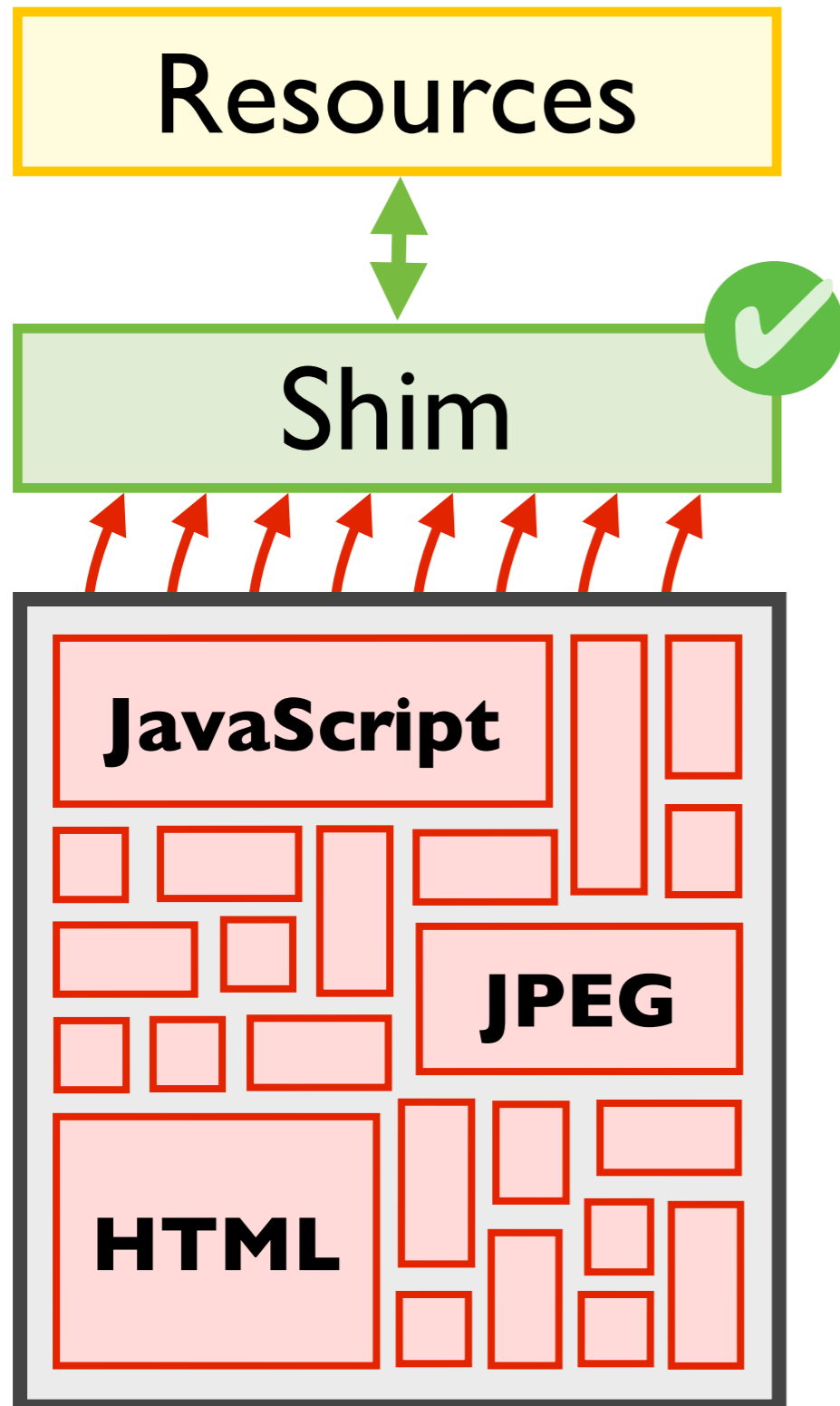
Isolate

*sandbox untrusted code*

Implement shim

*guards resource access*

# Formally Verify a Browser?!



Isolate

*sandbox untrusted code*

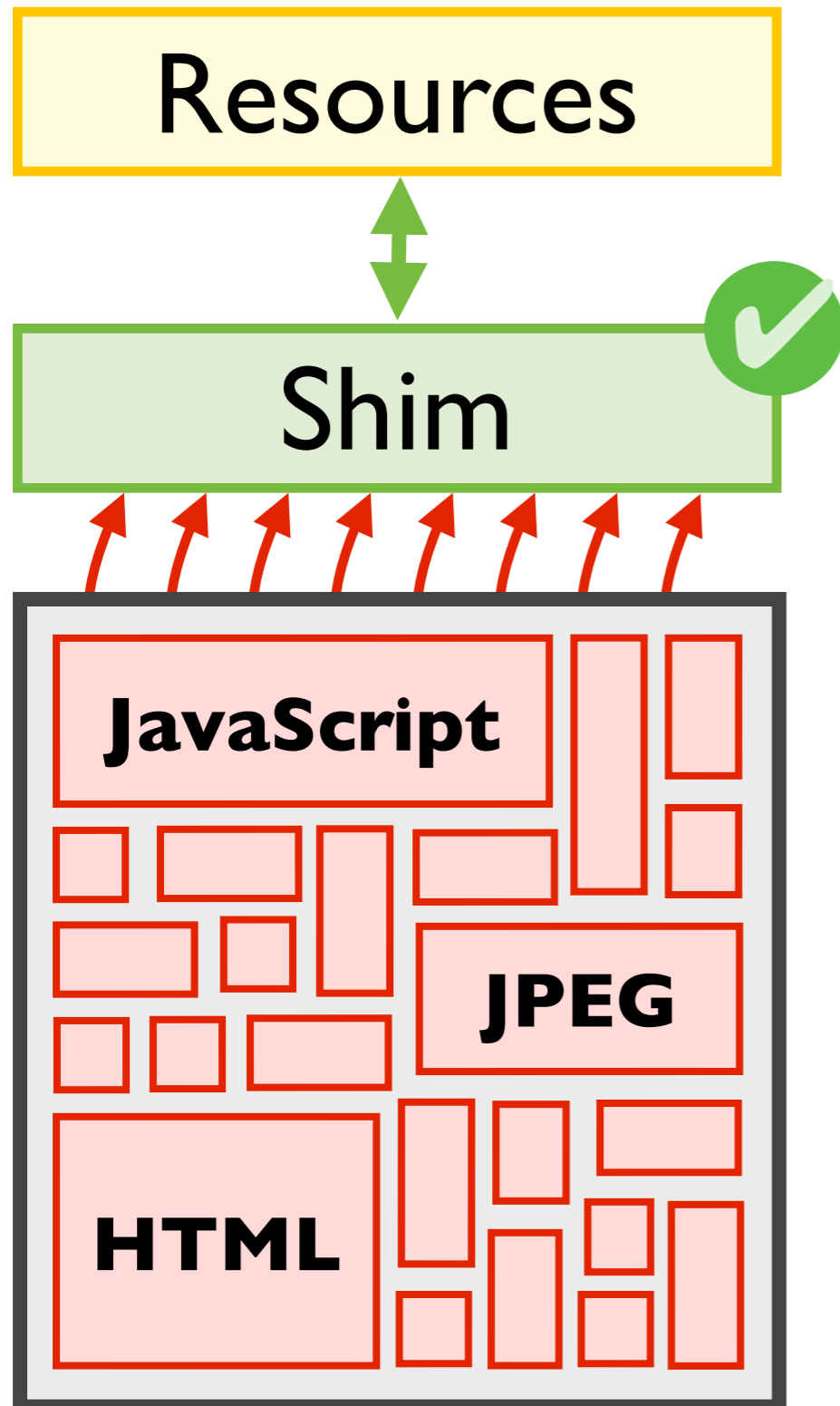
Implement shim

*guards resource access*

Verify shim

*prove security policy*

# Formal Shim Verification

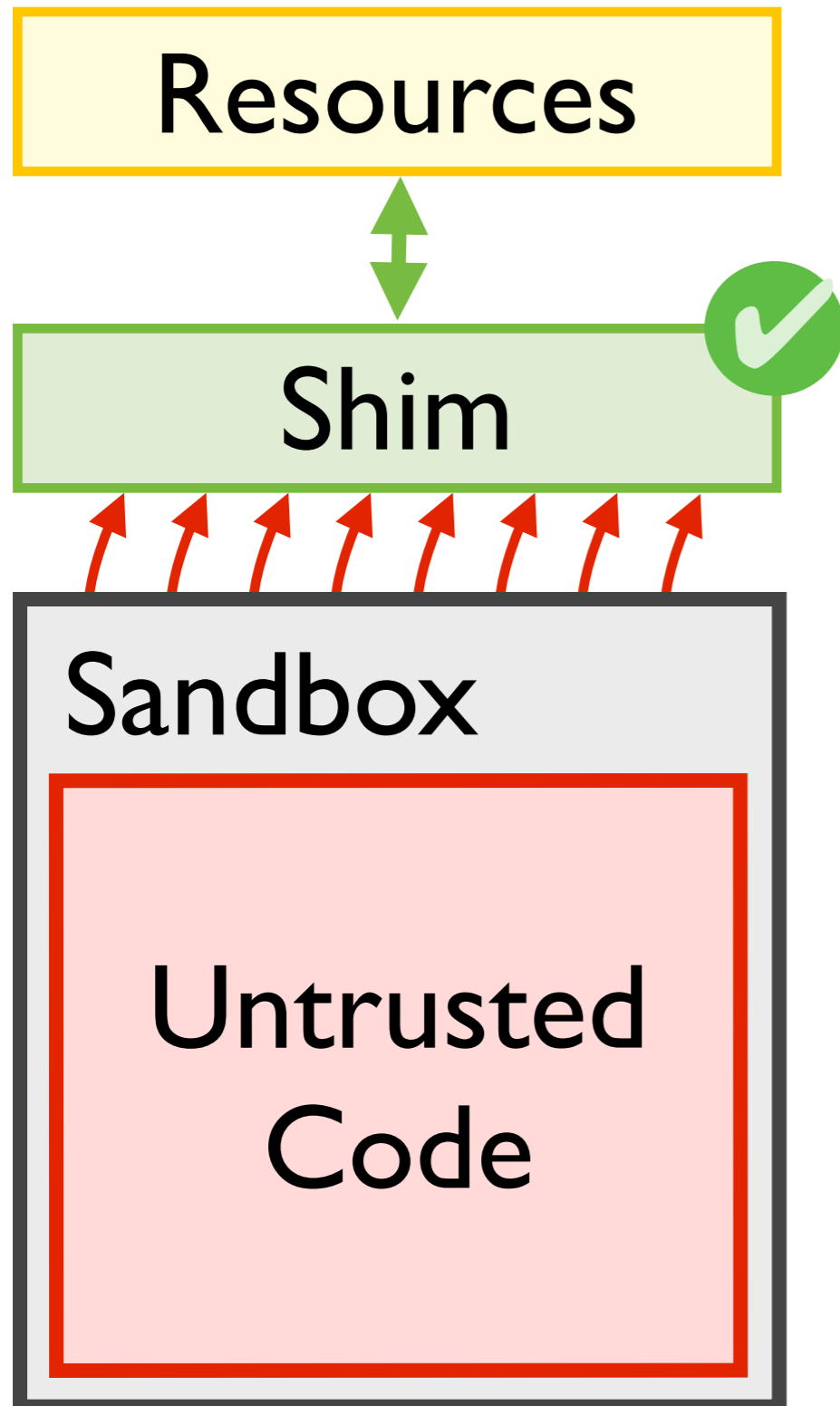


**Isolate**  
*sandbox untrusted code*

**Implement shim**  
*guards resource access*

**Verify shim**  
*prove security policy*

# Formal Shim Verification



Isolate  
Implement shim  
Verify shim

Applies when:

- 1. sys fits architecture*
  - 2. policy over resources*
- browser, httpd, sshd, ...*



# Formal Shim Verification

Key Insight: *Focus Effort*

Guarantee sec props for entire system

Only implement and prove small shim

Radically ease verification burden

Prove *actual code* correct

# Mitigating the Burden of Proof

1: Scaling proofs to critical infrastructure

➔ *Formal shim verification for large apps*

*QUARK: browser with security guarantees*

2: Evolving formally verified systems

*Reflex DSL exploits domain for proof auto*

# Mitigating the Burden of Proof

1: Scaling proofs to critical infrastructure

*Formal shim verification for large apps*

➔ *QUARK: browser with security guarantees*

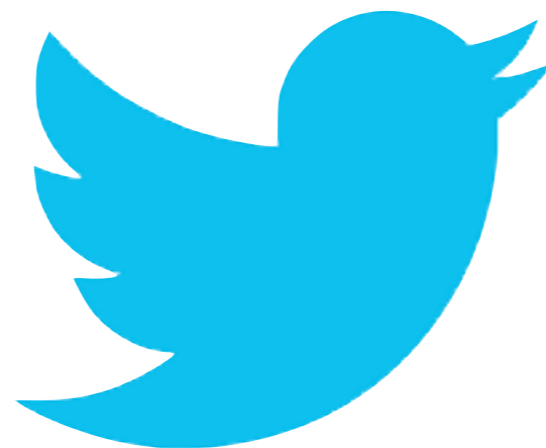
2: Evolving formally verified systems

*Reflex DSL exploits domain for proof auto*

# Browsers: Critical Infrastructure



Hotmail



# Browsers: Vulnerable

News

## Pwn2Own hacking contest puts record \$560K on the line

Google back as co-sponsor after organizer changes rules

By Gregg Keizer

January 18, 2013 10:57 AM ET 1 Comment



Computerworld - HP TippingPoint, the long-time organizer of the annual Pwn2Own hacking contest, has revamped the challenge for the second year running and will offer cash awards exceeding half a million dollars, more than five times the amount paid out last year, the company said yesterday.

The 2013 edition of the contest will offer \$560,000 in potential prize money to hackers who demonstrate exploits of previously-unknown vulnerabilities in Chrome, Firefox, Internet Explorer (IE) or Safari, or the Adobe Reader, Adobe Flash or Oracle Java browser plug-ins.

Prizes will be awarded on a sliding schedule, with \$100,000 for the first to hack Chrome on Windows 7 or IE10 on Windows 8. From there, payments will fall to \$75,000 for IE9 and slide through a number of targets before ending at \$20,000 for Java. Prizes will also be given for exploiting Adobe Flash and Adobe Reader (\$70,000 each), Safari (\$65,000) and Firefox (\$60,000).

About the Java award, Kostya Kortchinsky, a researcher who now works for Microsoft, quickly [tweeted](#), "ZDI giving out \$20k for free," referring to the Oracle software's recent vulnerabilities.

Pwn2Own will run March 6-8 at the CanSecWest security conference in Vancouver, British Columbia.

## Defenses / Policies:

[Jang et al. W2SP]

[Stamm et al. WWW]

[Jackson et al. W2SP]

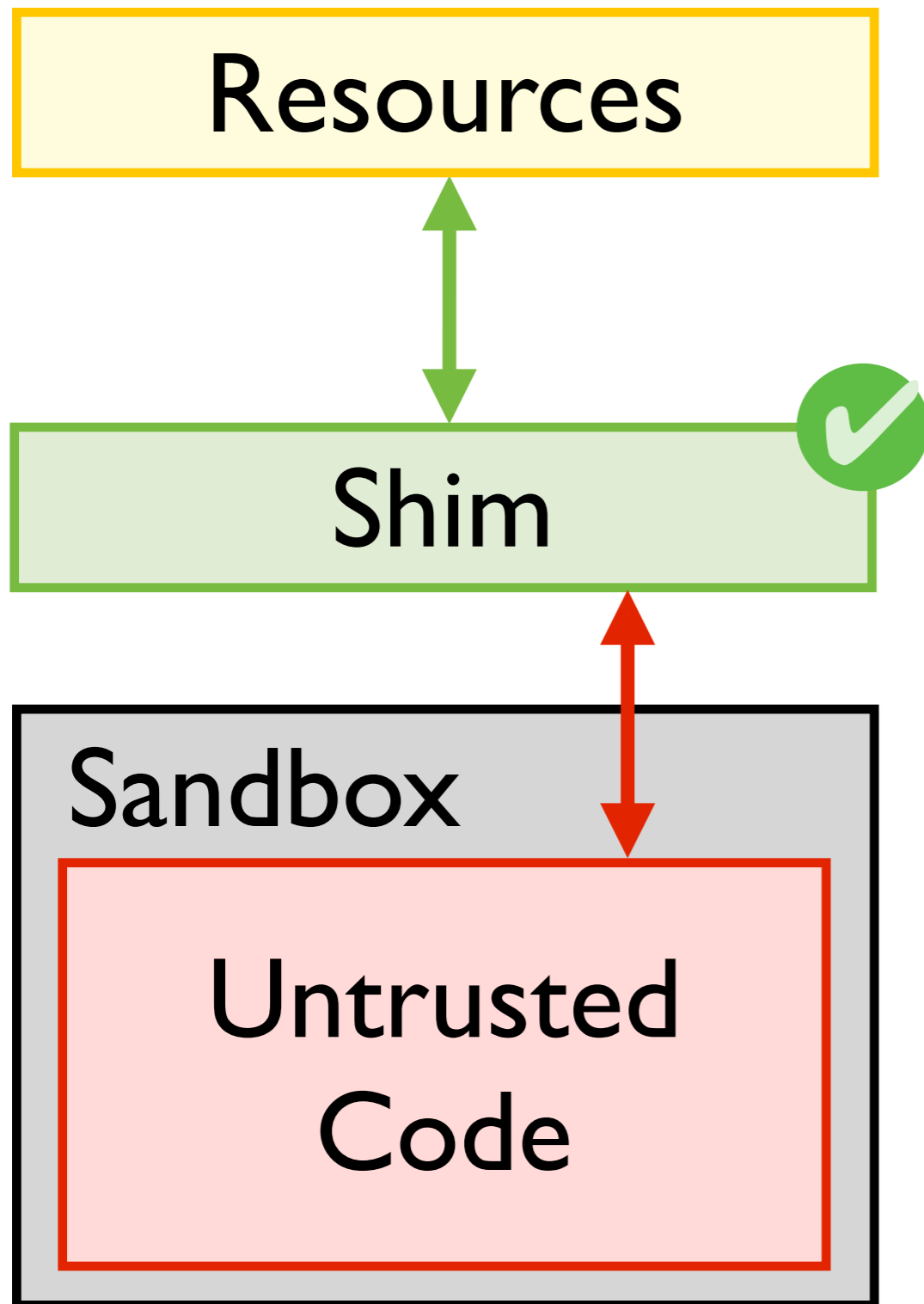
[Barth et al. CCS]

[Singh et al. OAKLAND]

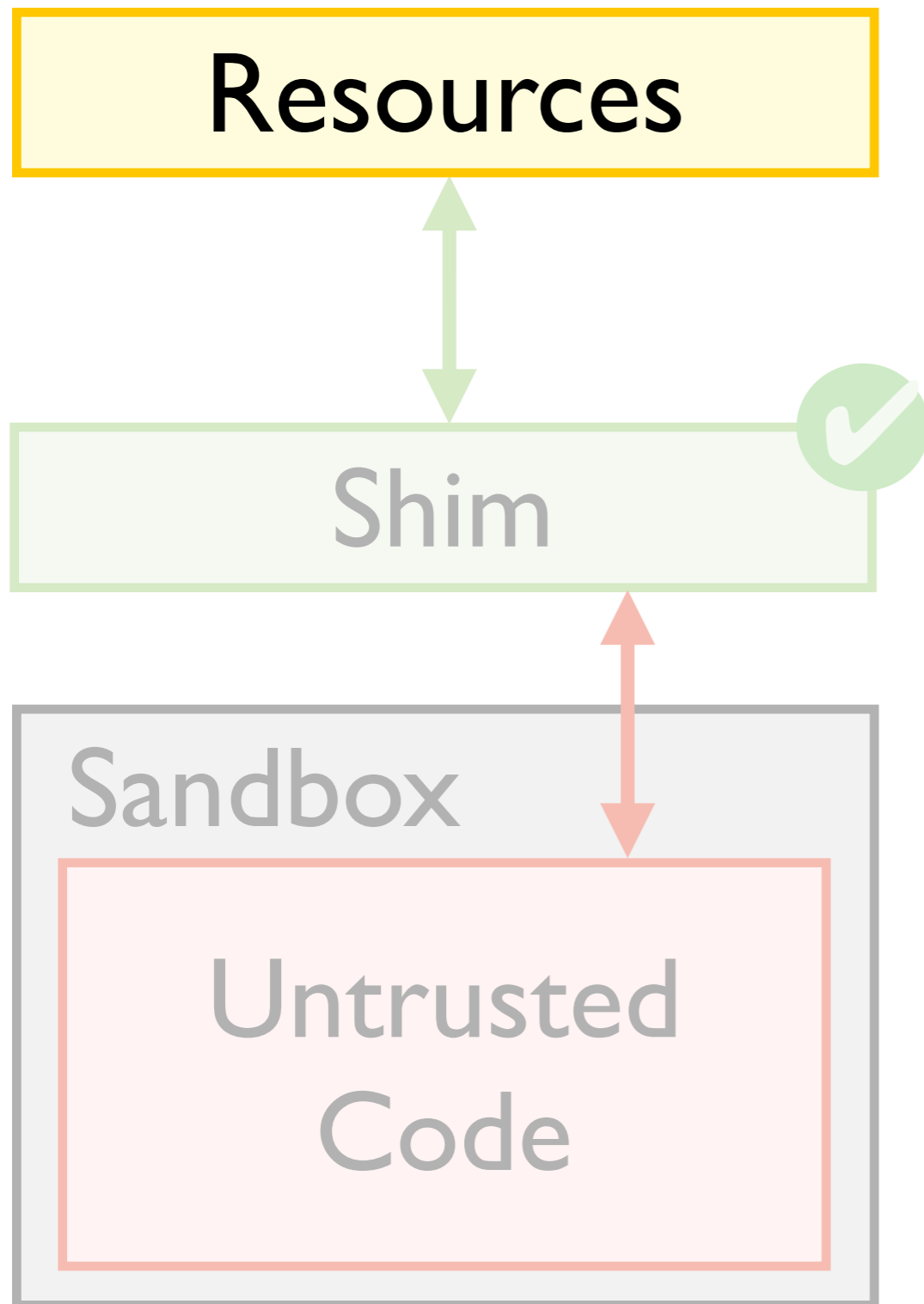
...

**Complex +  
Implementation Bugs**

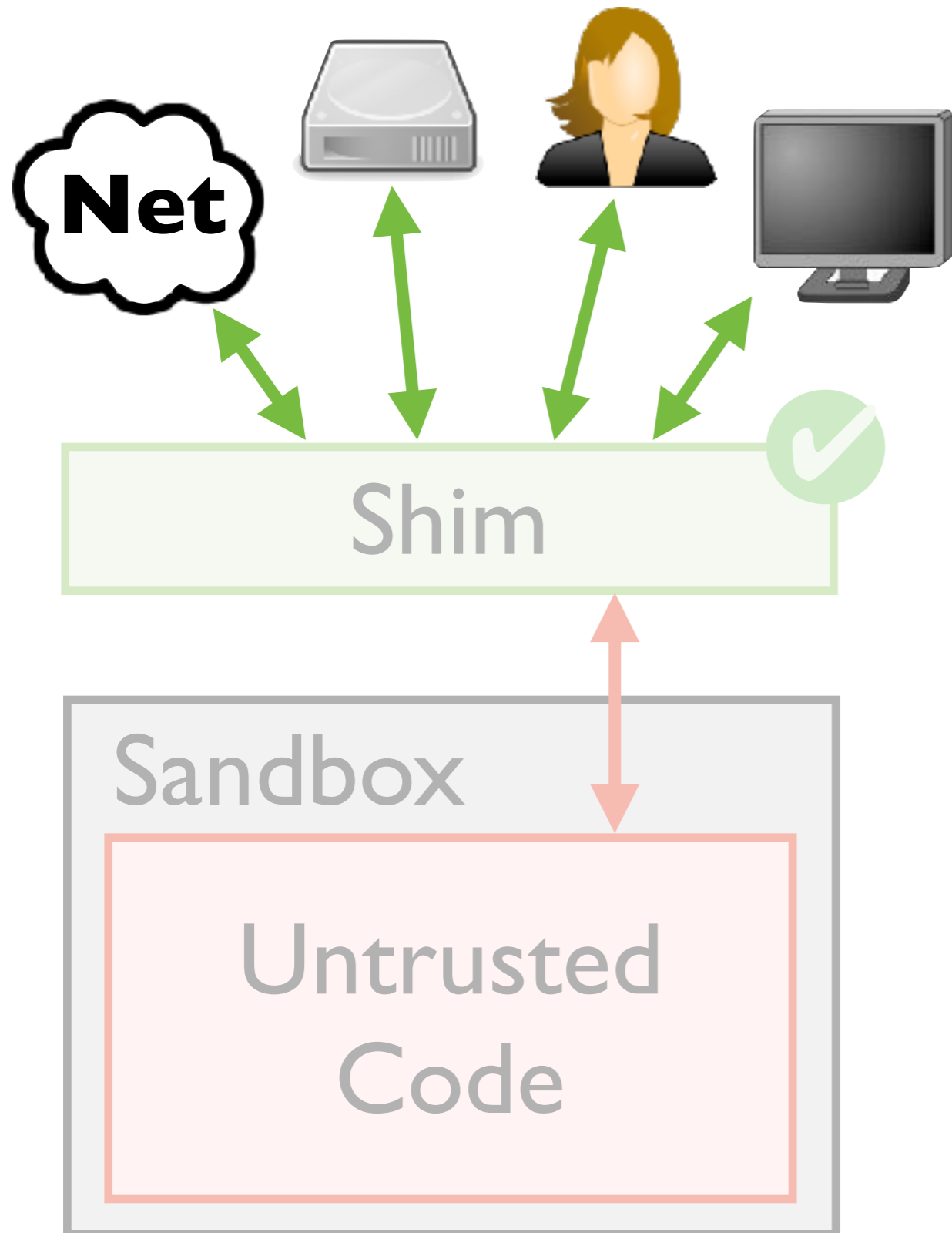
# Quark: Verified Browser



# Quark: Verified Browser



# Quark: Verified Browser



## Resources

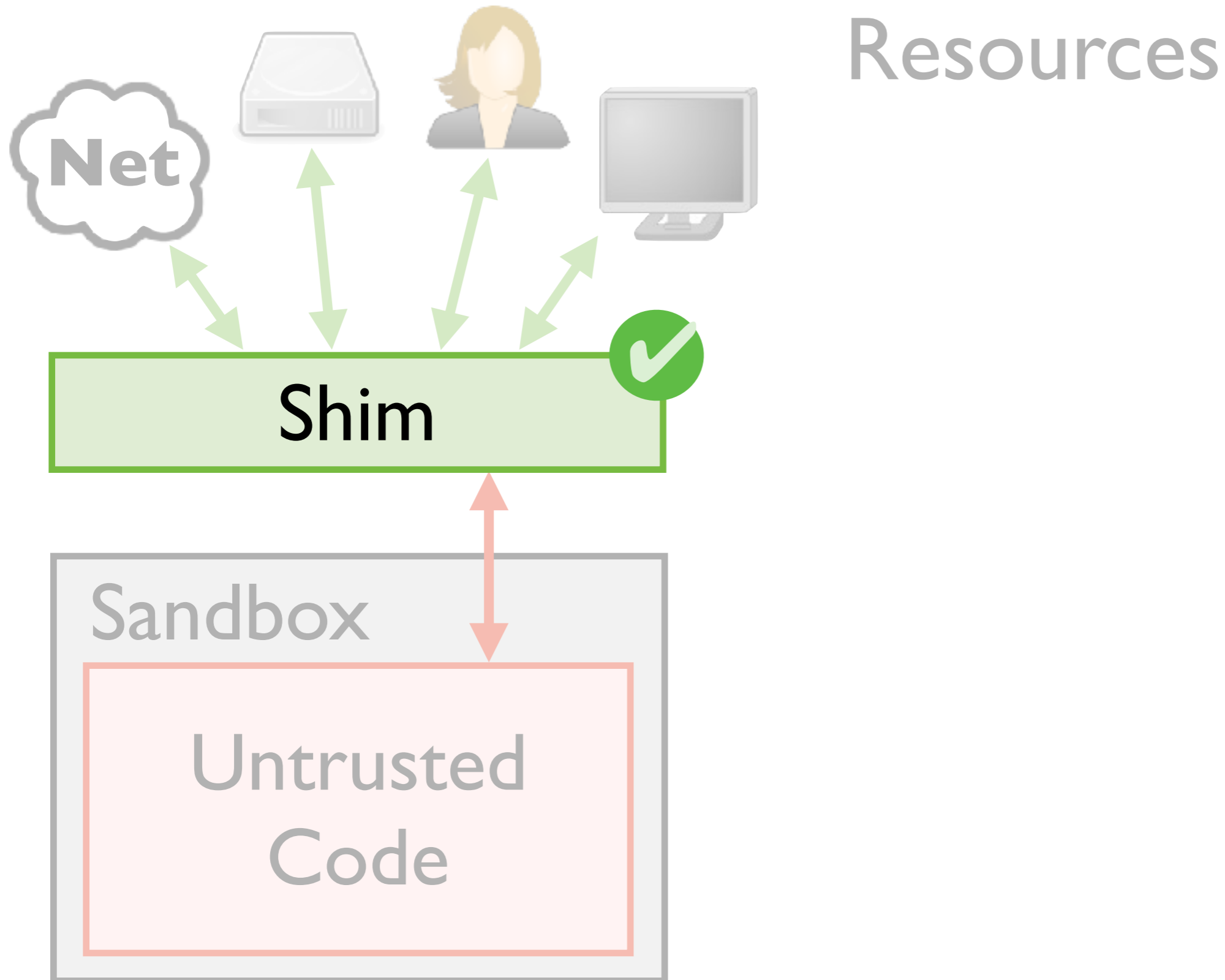
*network*

*persistent storage*

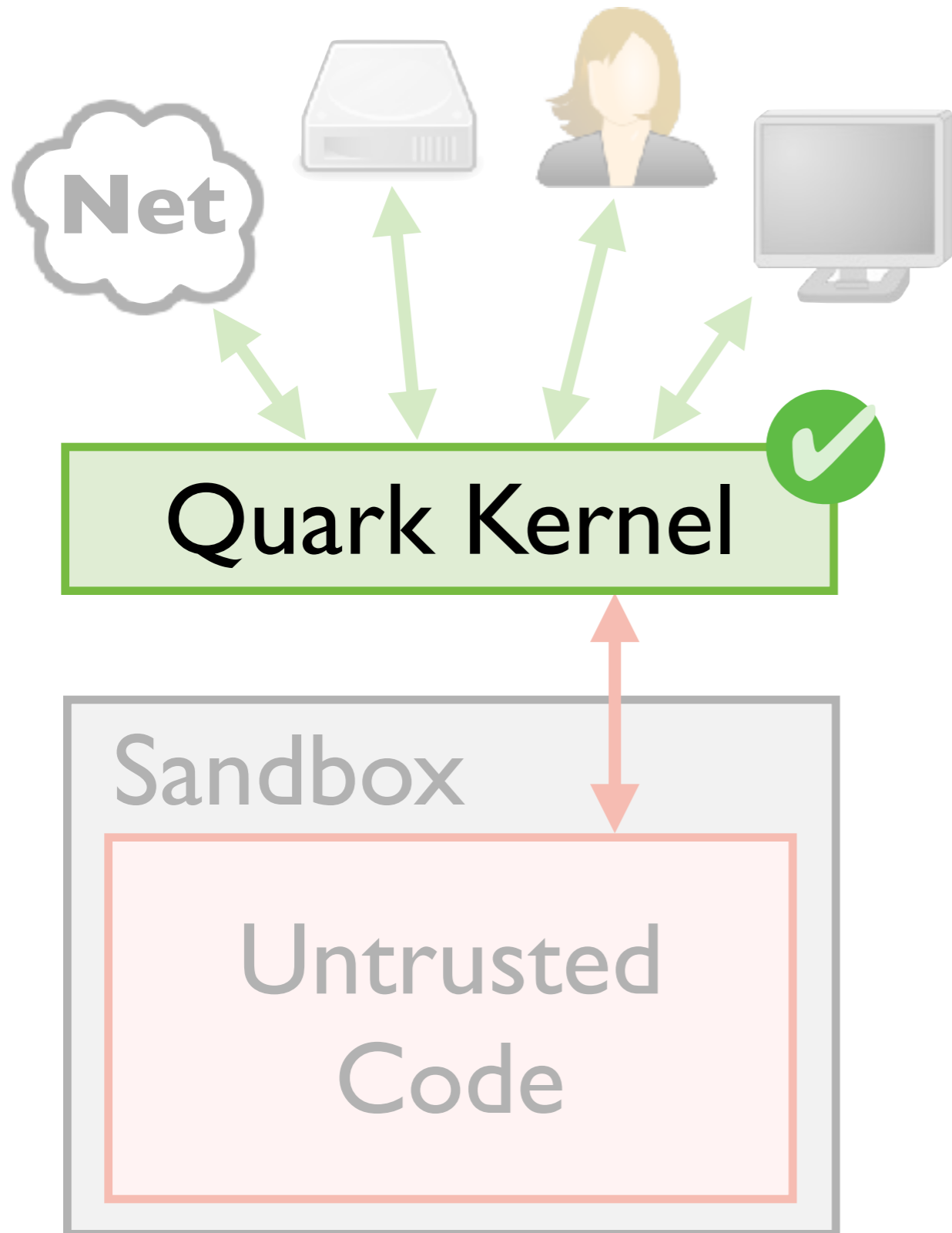
*user interface*



# Quark: Verified Browser



# Quark: Verified Browser

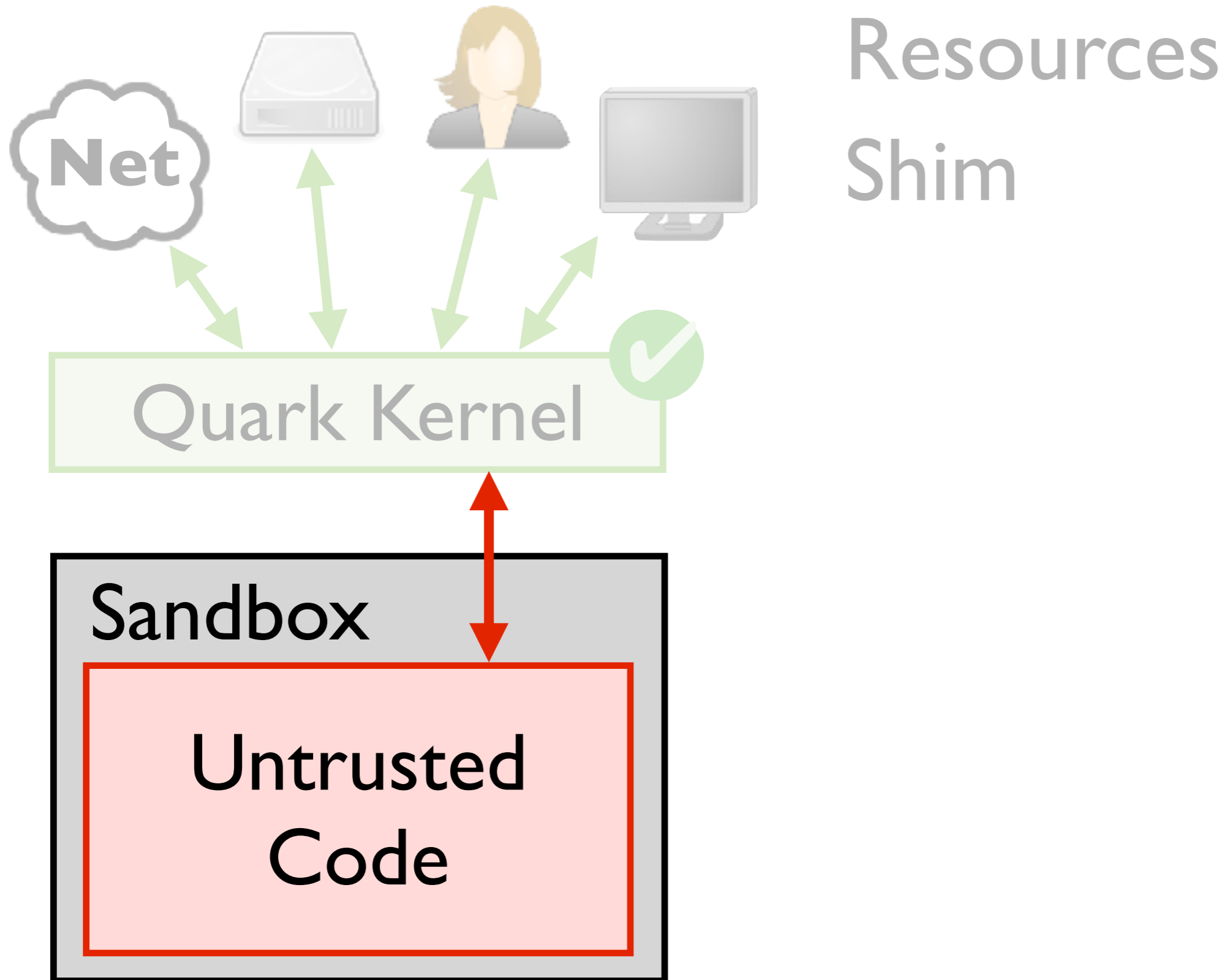


Resources

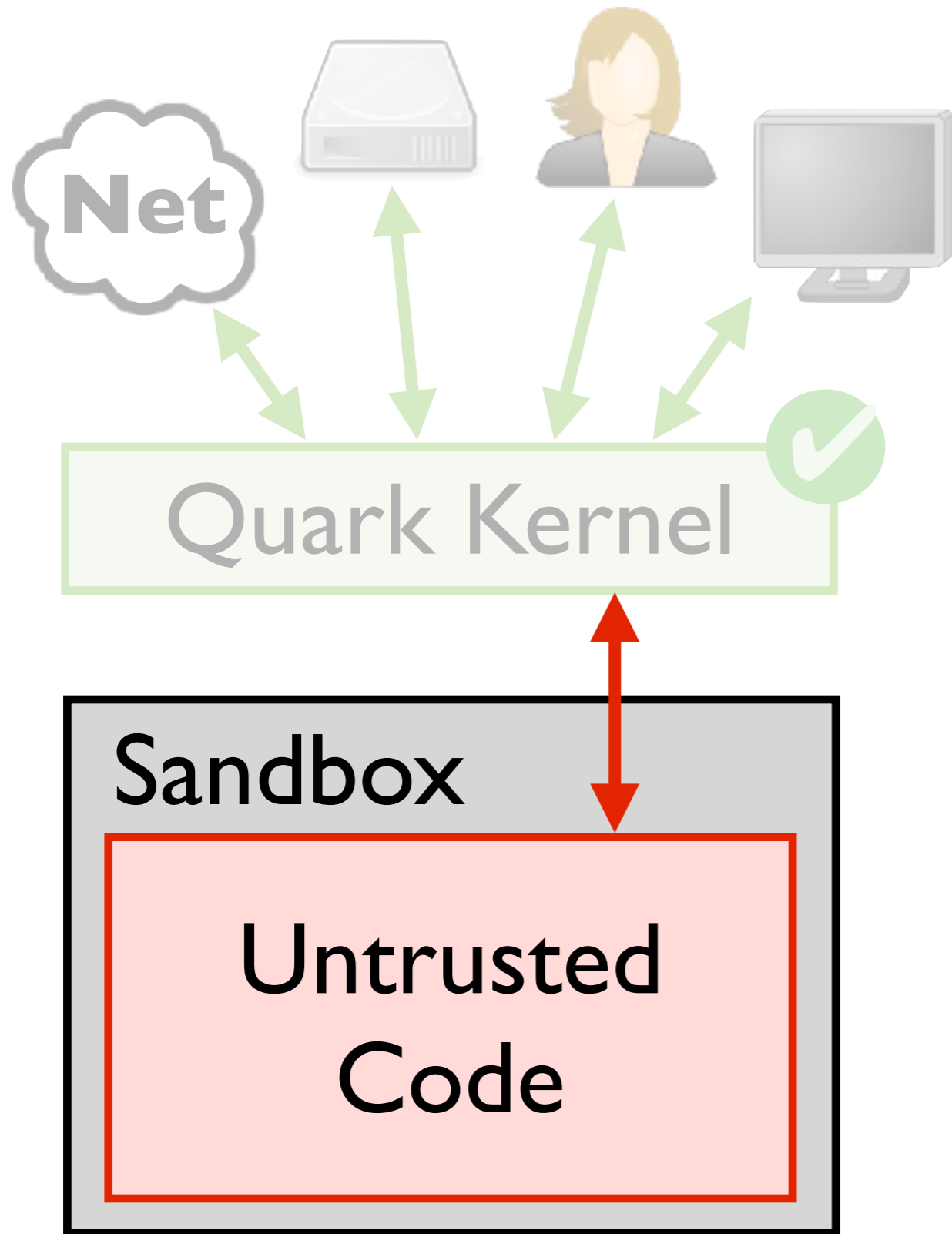
Shim

*Quark browser kernel  
code, spec, proof in Coq*

# Quark: Verified Browser



# Quark: Verified Browser



Resources

Shim

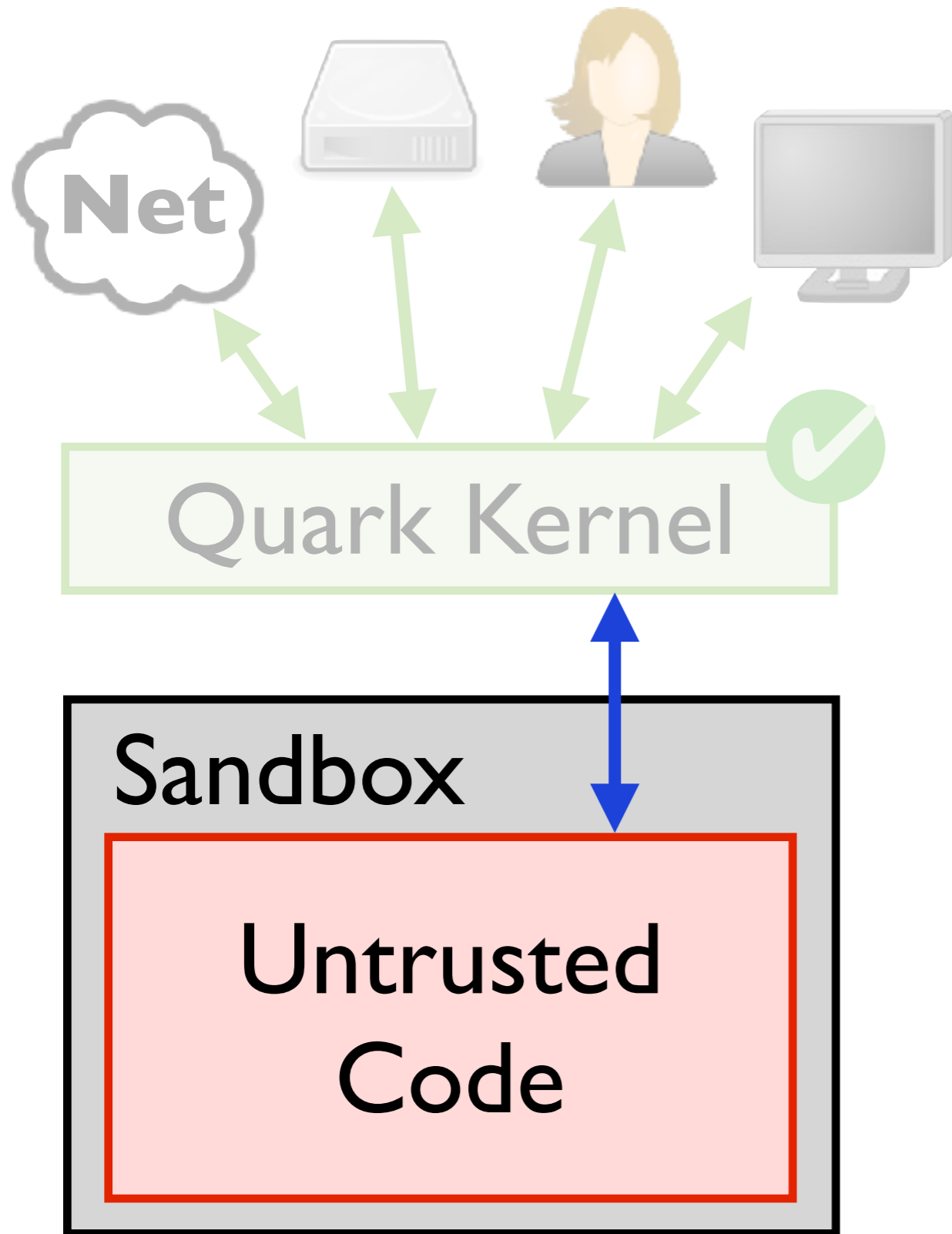
Untrusted Code

*browser components*

*run as separate procs*

*strictly sandboxed*

# Quark: Verified Browser



Resources

Shim

Untrusted Code

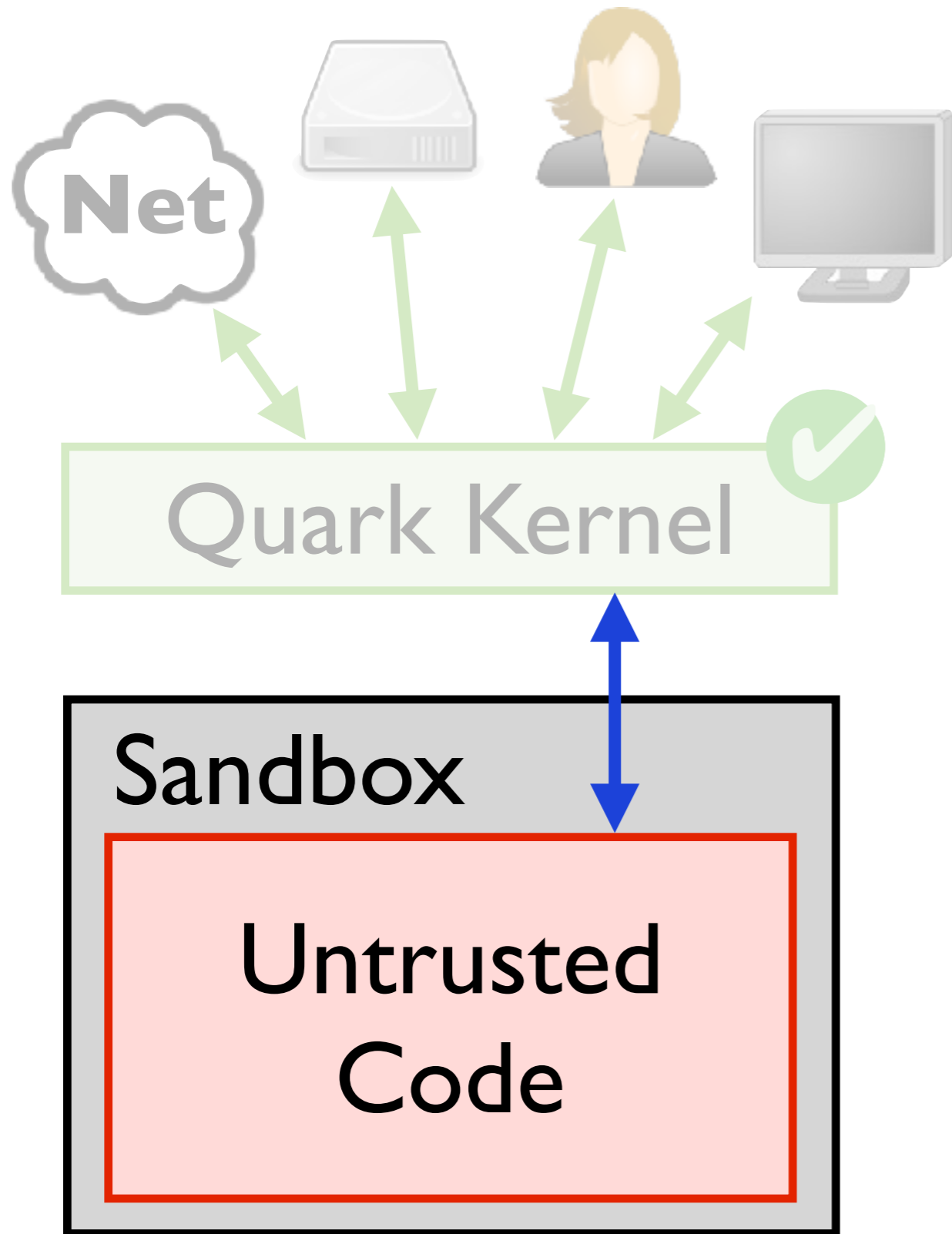
*browser components*

*run as separate procs*

*strictly sandboxed*

*talk to kernel over **pipe***

# Quark: Verified Browser



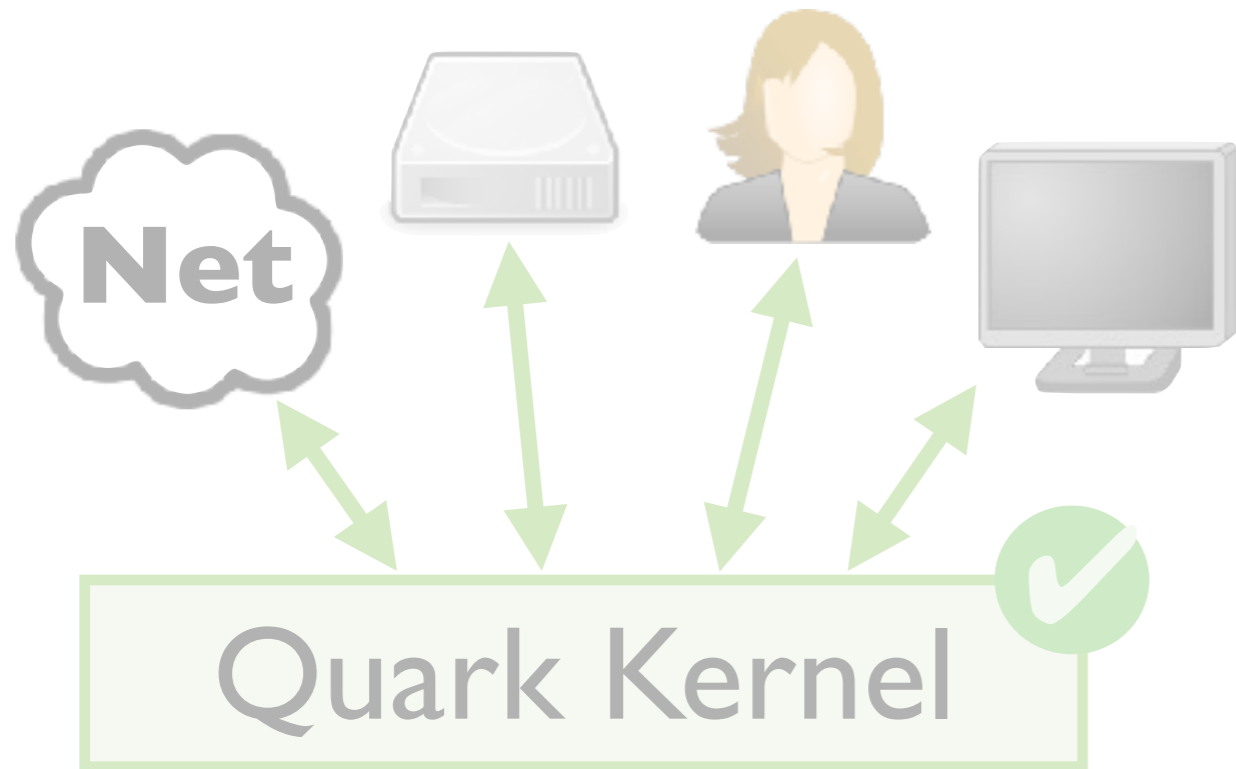
Resources

Shim

Untrusted Code

*two component types*

# Quark: Verified Browser

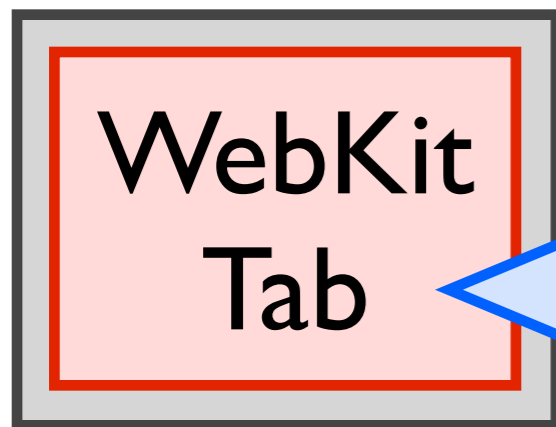


Resources

Shim

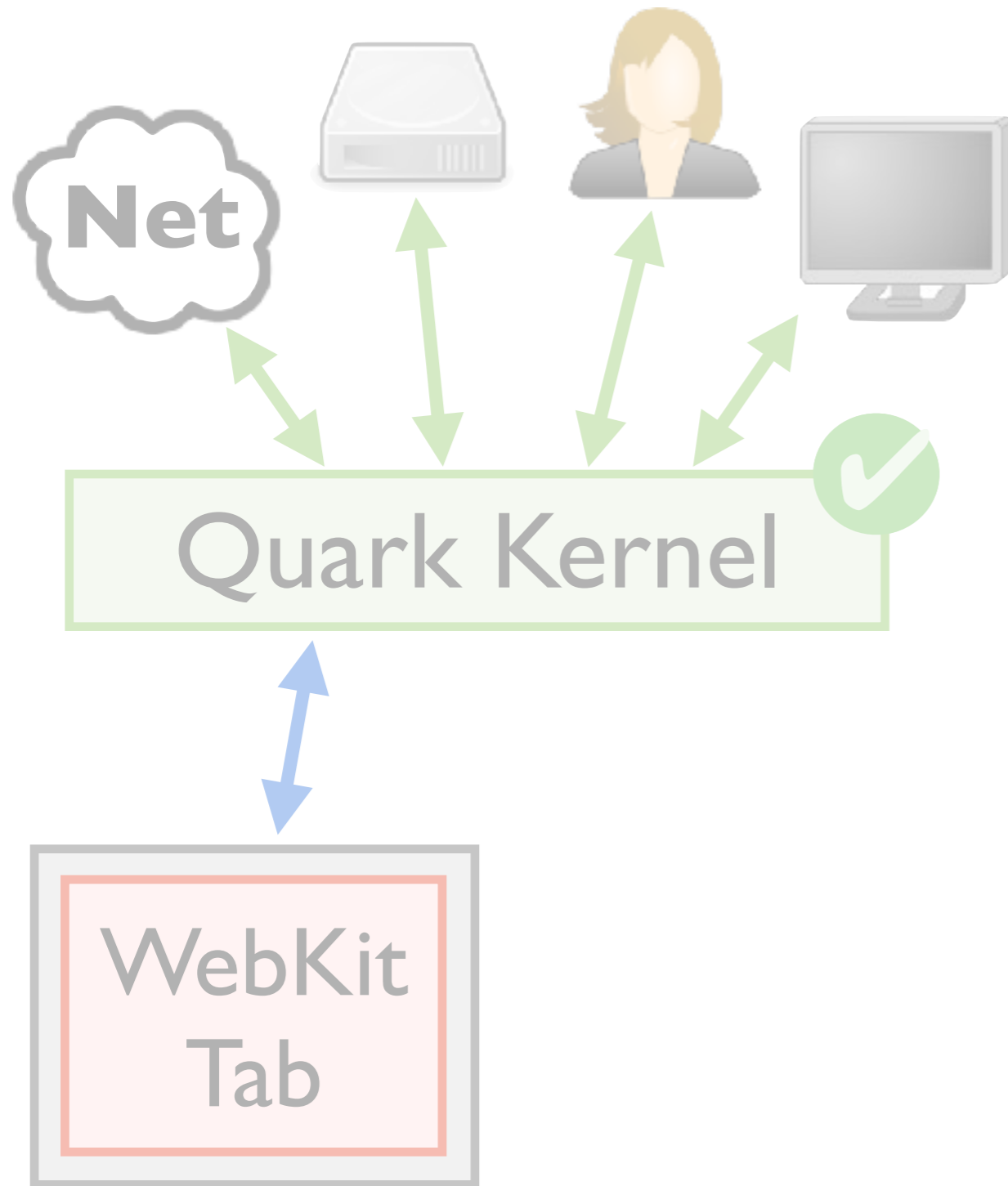
**Untrusted Code**

*two component types*



modified WebKit,  
intercept accesses

# Quark: Verified Browser



Resources

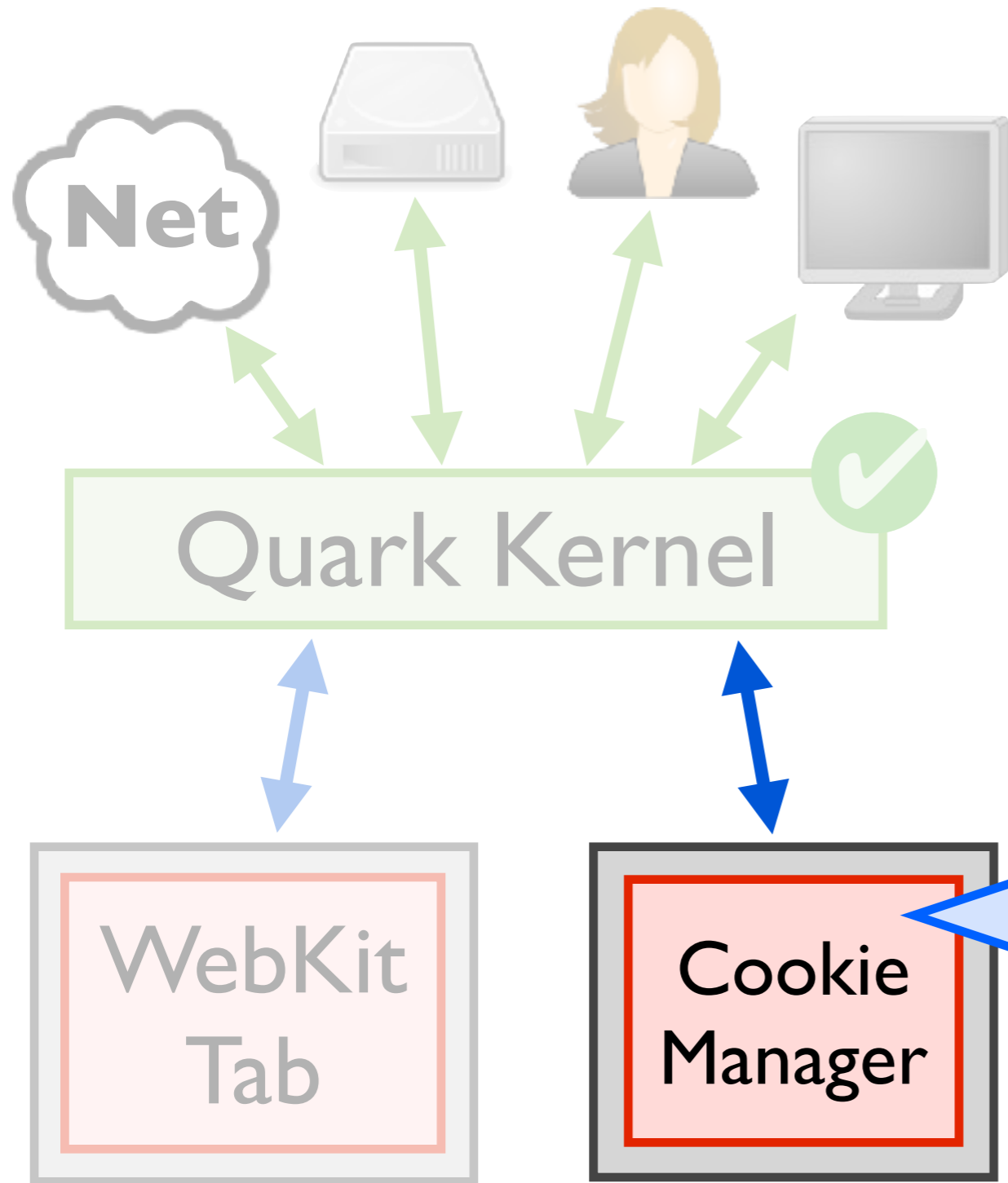
Shim

Untrusted Code

*two component types*



# Quark: Verified Browser



Resources

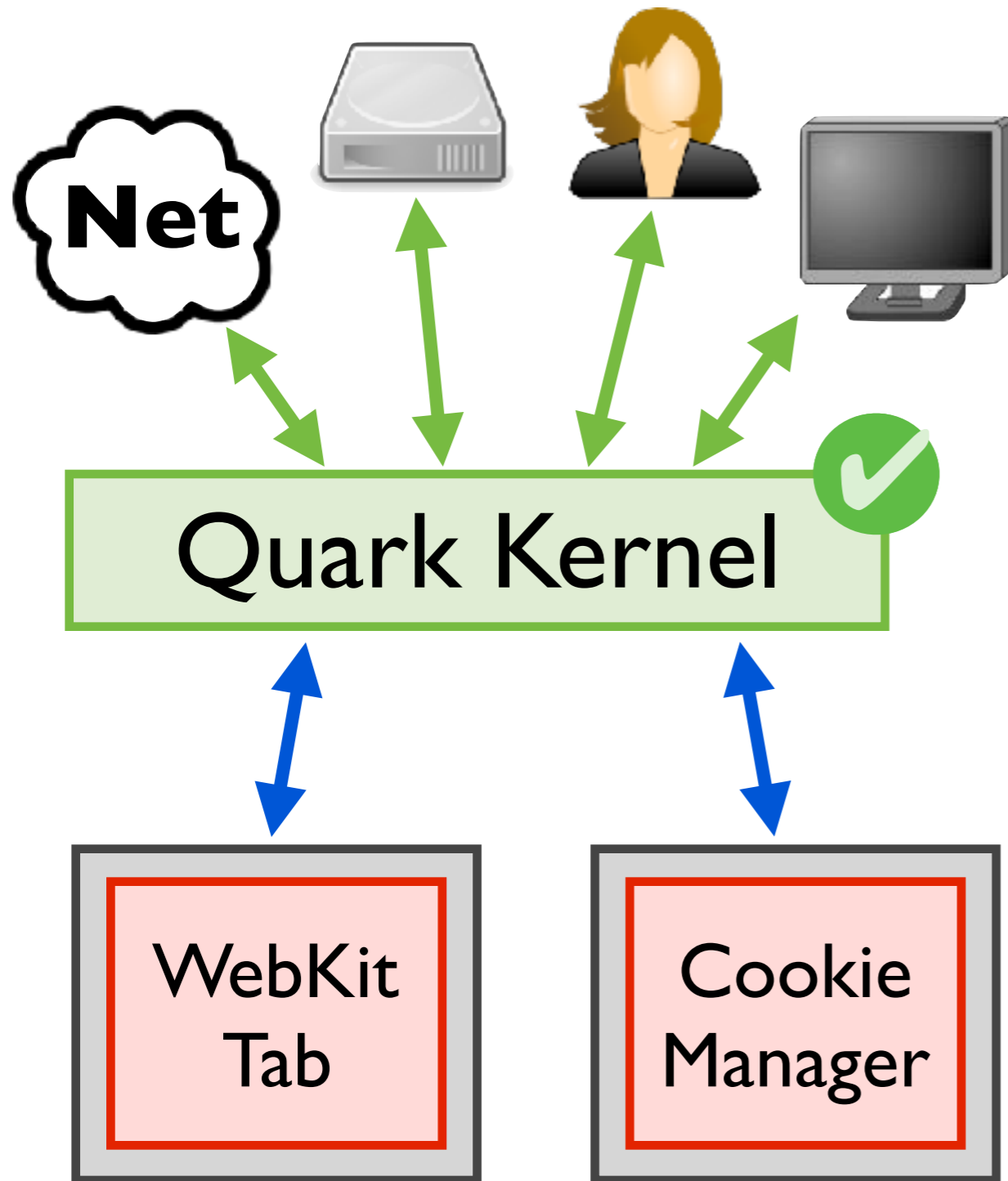
Shim

Untrusted Code

*two component types*

written in Python,  
manages single domain

# Quark: Verified Browser



Resources

Shim

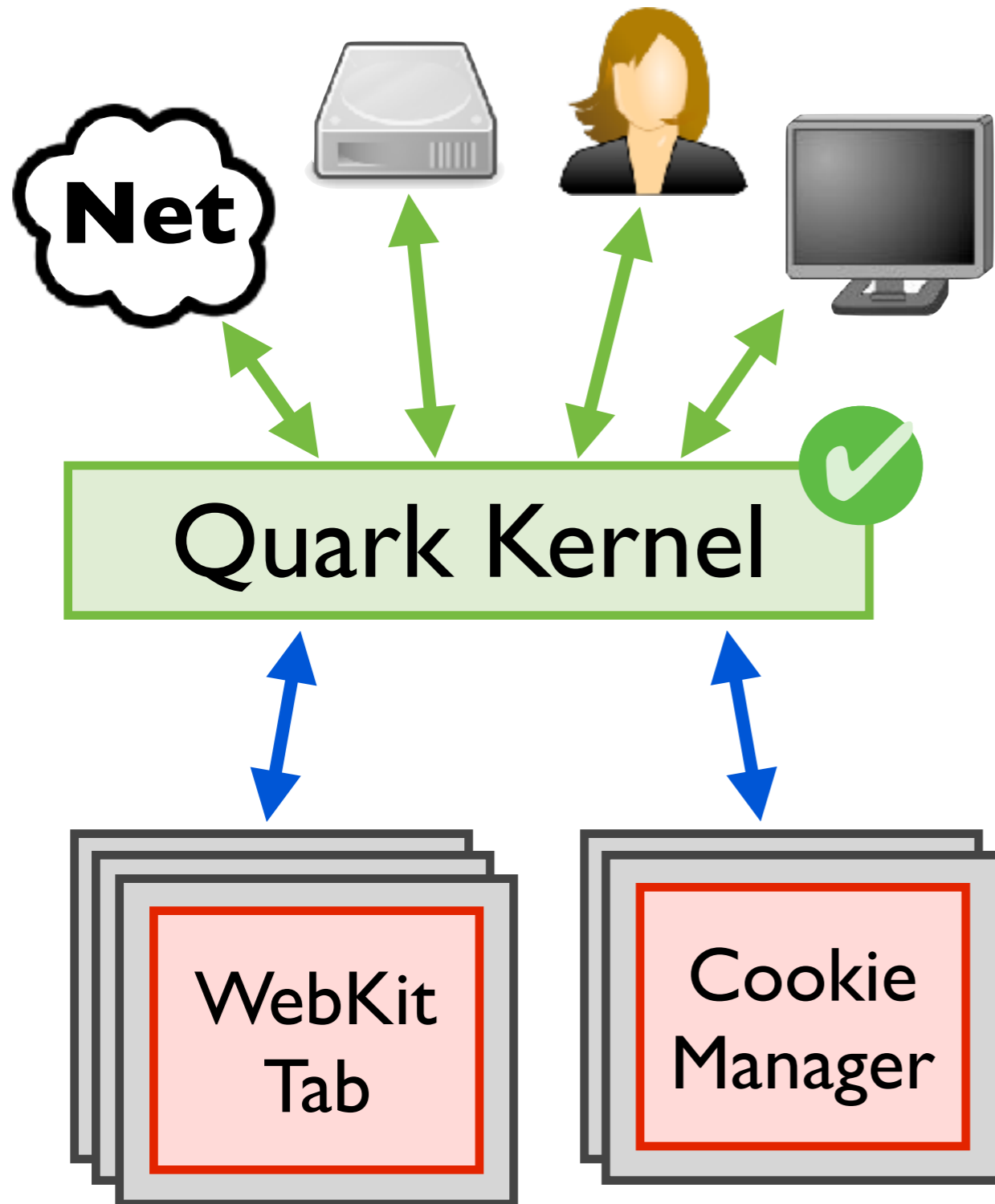
Untrusted Code

*two component types*

*WebKit tabs*

*cookie managers*

# Quark: Verified Browser



Resources

Shim

Untrusted Code

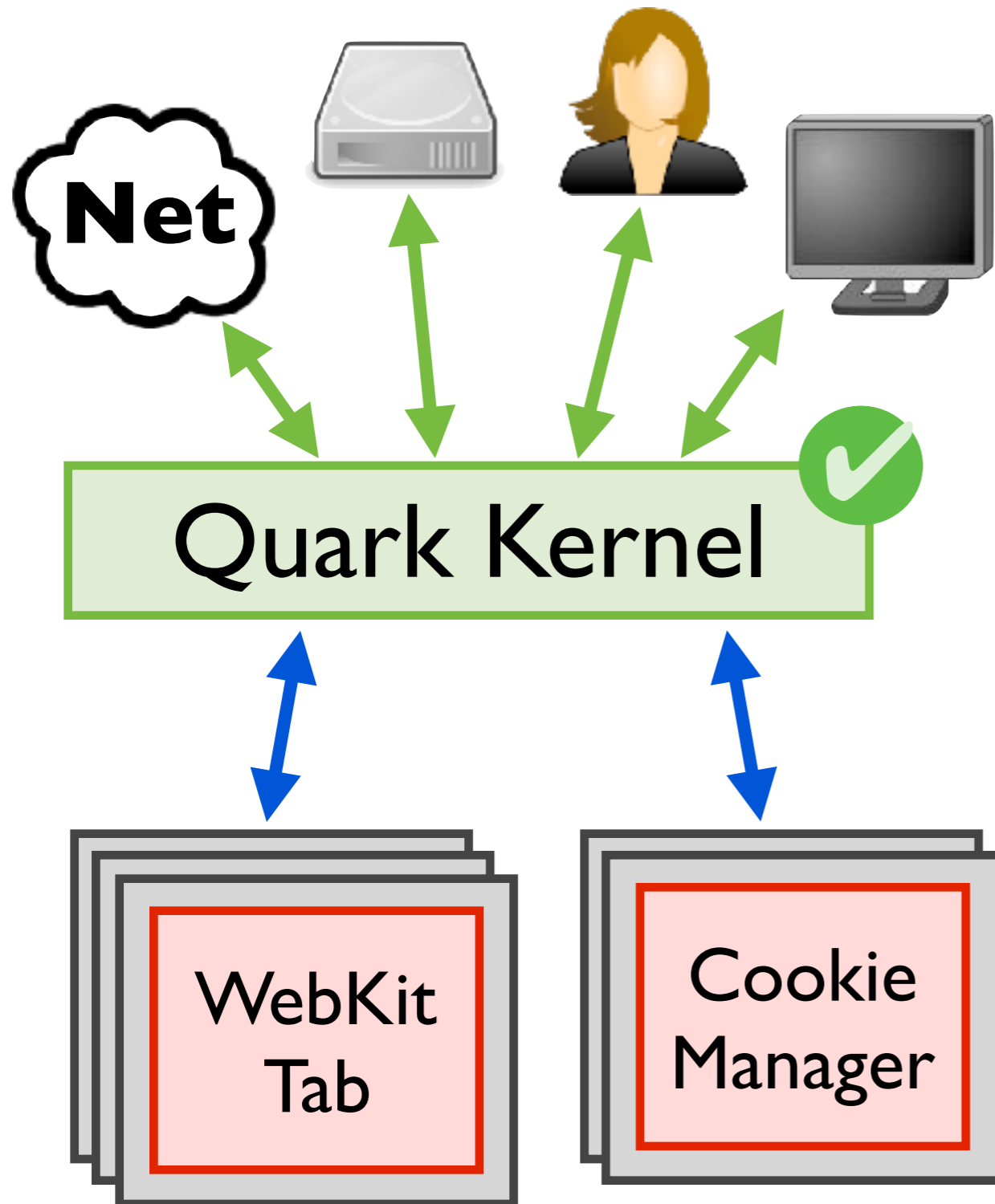
*two component types*

*WebKit tabs*

*cookie managers*

*several instances each*

# Quark: Verified Browser



# Quark: Verified Browser

Quark Kernel



# Quark Kernel

Quark Kernel



# Quark Kernel: Code, Spec, Proof

Quark Kernel



# Quark Kernel: *Code*, Spec, Proof

Quark Kernel





# Quark Kernel: *Code*, Spec, Proof



# Quark Kernel: *Code*, Spec, Proof

```
Definition kstep ...
```

# Quark Kernel: *Code*, Spec, Proof

```
Definition kstep(focused_tab, tabs) :=
```

```
...
```



kernel state

# Quark Kernel: *Code*, Spec, Proof

```
Definition kstep(focused_tab, tabs) :=  
  f <- select(stdin, tabs);  
  ...
```

Unix-style select to  
find a component  
pipe ready to read

# Quark Kernel: *Code*, Spec, Proof

```
Definition kstep(focused_tab, tabs) :=  
  f <- select(stdin, tabs);  
  match f with  
  | Stdin => case: f is user input  
    ...  
  | Tab t => case: f is tab pipe  
    ...
```

# Quark Kernel: *Code*, Spec, Proof

```
Definition kstep(focused_tab, tabs) :=  
  f <- select(stdin, tabs);  
  match f with  
  | Stdin =>  
    cmd <- read_cmd(stdin);  
    ...  
  
  | Tab t =>  
    ...
```

read command from  
user over **stdin**

# Quark Kernel: *Code*, Spec, Proof

```
Definition kstep(focused_tab, tabs) :=  
  f <- select(stdin, tabs);  
  match f with  
  | Stdin =>  
    cmd <- read_cmd(stdin);  
    match cmd with  
    | AddTab =>  
      ...  
    | ...  
  | Tab t =>  
    ...
```

user wants to create  
and focus a new tab

# Quark Kernel: *Code*, Spec, Proof

```
Definition kstep(focused_tab, tabs) :=  
  f <- select(stdin, tabs);  
  match f with  
  | Stdin =>  
    cmd <- read_cmd(stdin);  
    match cmd with  
    | AddTab =>  
      t <- mk_tab();  
      ...  
    | ...  
  | Tab t =>  
    ...
```

create a new tab



# Quark Kernel: *Code*, Spec, Proof

```
Definition kstep(focused_tab, tabs) :=  
  f <- select(stdin, tabs);  
  match f with  
  | Stdin =>  
    cmd <- read_cmd(stdin);  
    match cmd with  
    | AddTab =>  
      t <- mk_tab();  
      write_msg(t, Render);  
      ...  
    | ...  
  | Tab t =>  
    ...
```

tell new tab to  
render itself

# Quark Kernel: *Code*, Spec, Proof

```
Definition kstep(focused_tab, tabs) :=  
  f <- select(stdin, tabs);  
  match f with  
  | Stdin =>  
    cmd <- read_cmd(stdin);  
    match cmd with  
    | AddTab =>  
      t <- mk_tab();  
      write_msg(t, Render);  
      return (t, t::tabs)  
    | ...  
  | Tab t =>  
    ...
```

return updated state

# Quark Kernel: *Code*, Spec, Proof

```
Definition kstep(focused_tab, tabs) :=
  f <- select(stdin, tabs);
  match f with
  | Stdin =>
    cmd <- read_cmd(stdin);
    match cmd with
    | AddTab =>
      t <- mk_tab();
      write_msg(t, Render);
      return (t, t::tabs)
    | ...
  | Tab t =>
    ...
```

# Quark Kernel: *Code*, Spec, Proof

```
Definition kstep(focused_tab, tabs) :=  
  f <- select(stdin, tabs);  
  match f with  
  | Stdin =>  
    cmd <- read_cmd(stdin);  
    match cmd with  
    | AddTab =>  
      t <- mk_tab();  
      write_msg(t, Render);  
      return (t, t::tabs)  
    | ...  
  | Tab t =>  
    ...
```

handle other  
user commands

# Quark Kernel: *Code*, Spec, Proof

```
Definition kstep(focused_tab, tabs) :=  
  f <- select(stdin, tabs);  
  match f with  
  | Stdin =>  
    cmd <- read_cmd(stdin);  
    match cmd with  
    | AddTab =>  
      t <- mk_tab();  
      write_msg(t, Render);  
      return t;  
    | ...  
  | Tab t =>  
    ...
```

handle requests  
from tabs

# Quark Kernel: *Code*, Spec, Proof

```
Definition kstep(focused_tab, tabs) :=
  f <- select(stdin, tabs);
  match f with
  | Stdin =>
    cmd <- read_cmd(stdin);
    match cmd with
    | AddTab =>
      t <- mk_tab();
      write_msg(t, Render);
      return (t, t::tabs)
    | ...
  | Tab t =>
    ...
```

# Quark Kernel: *Code*, Spec, Proof

# Quark Kernel: Code, *Spec*, Proof



# Quark Kernel: Code, *Spec*, Proof

Safety properties to mitigate attacks

*restrict kernel behavior to only safe executions*

Example: mitigate phishing attacks

*prevent tricks that get users to divulge secrets*

seems legit



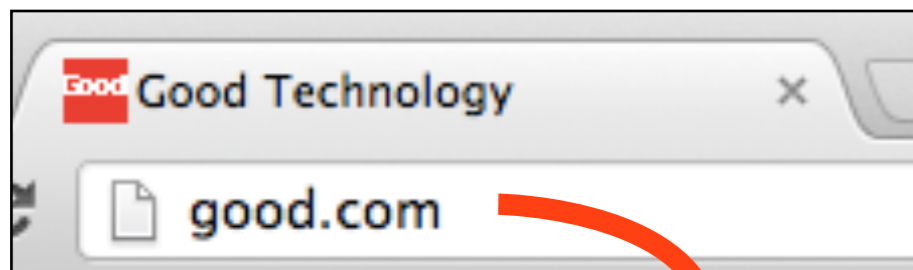
# Quark Kernel: Code, *Spec*, Proof

Safety properties to mitigate attacks

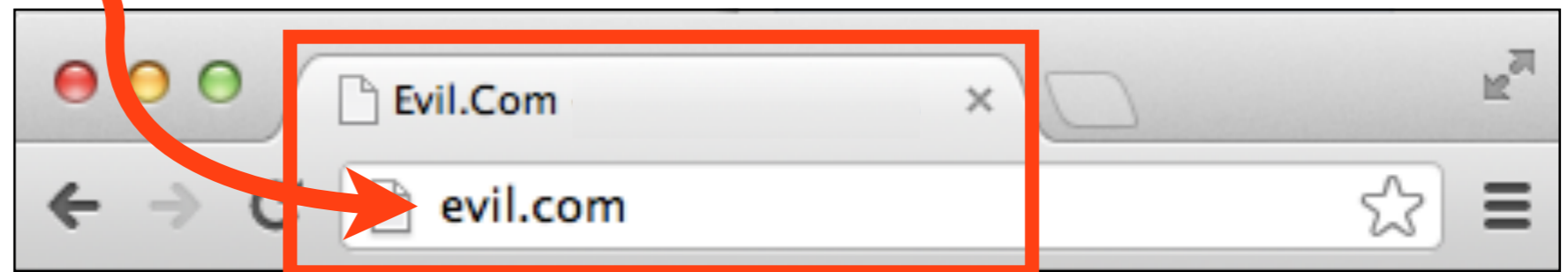
*restrict kernel behavior to only safe executions*

Example: mitigate phishing attacks

*prevent tricks that get users to divulge secrets*



**spoofed!**



# Quark Kernel: Code, *Spec*, Proof

Specify correct behavior wrt syscall seqs

`read() , write() , open() , write() , ...`

# Quark Kernel: Code, *Spec*, Proof

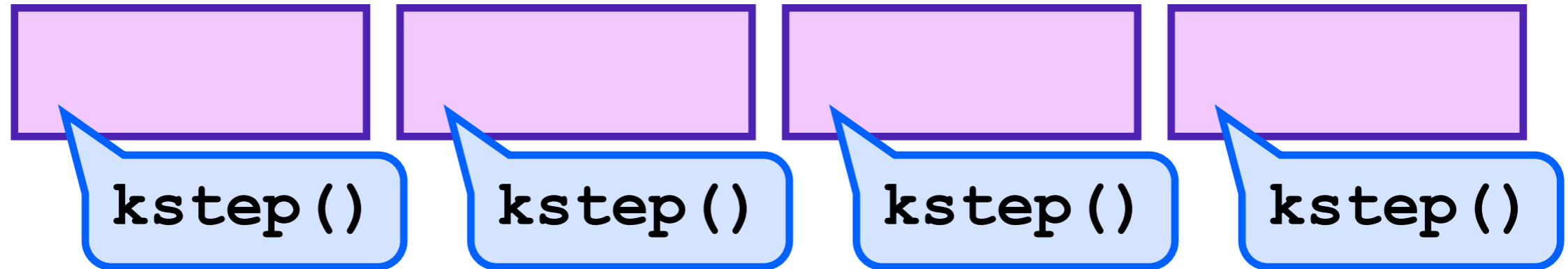
Specify correct behavior wrt syscall seqs



*trace*: all syscalls made  
by Quark kernel  
during execution

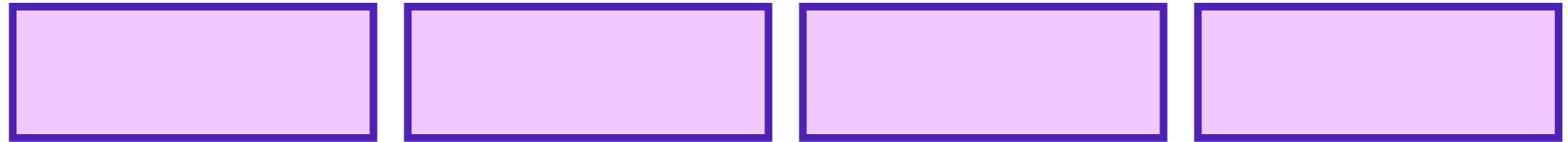
# Quark Kernel: Code, *Spec*, Proof

Specify correct behavior wrt syscall seqs



# Quark Kernel: Code, *Spec*, Proof

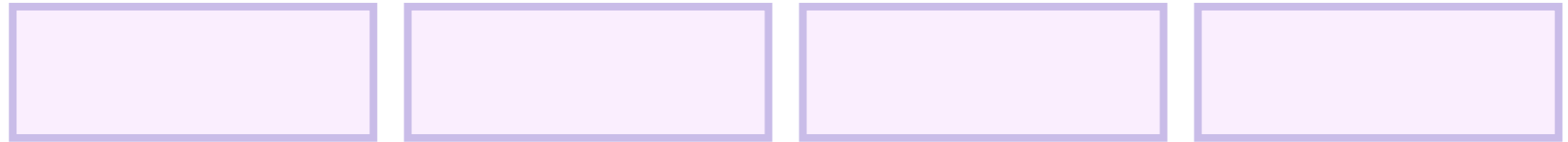
Specify correct behavior wrt syscall seqs



*structure of produceable traces supports spec & proof*

# Quark Kernel: Code, *Spec*, Proof

Specify correct behavior wrt syscall seqs

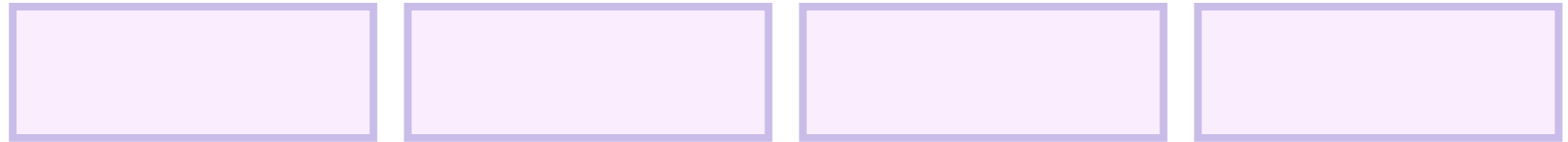


*structure of produceable traces supports spec & proof*

**Example: address bar correctness**

# Quark Kernel: Code, *Spec*, Proof

Specify correct behavior wrt syscall seqs



*structure of produceable traces supports spec & proof*

## Example: address bar correctness

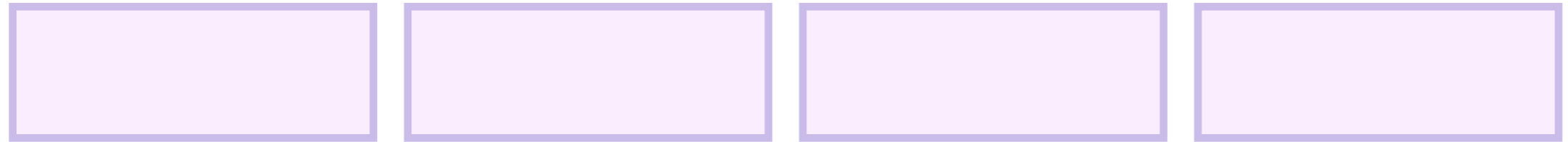
```
forall trace tab domain,
```

for *any* trace, tab,  
and domain



# Quark Kernel: Code, *Spec*, Proof

Specify correct behavior wrt syscall seqs



*structure of produceable traces supports spec & proof*

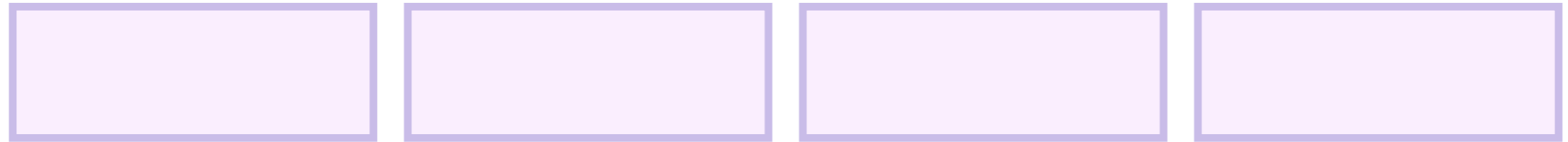
## Example: address bar correctness

```
forall trace tab domain,  
  quark_produced(trace)  $\wedge$   
  ...
```

*if Quark could have produced this trace*

# Quark Kernel: Code, *Spec*, Proof

Specify correct behavior wrt syscall seqs



*structure of produceable traces supports spec & proof*

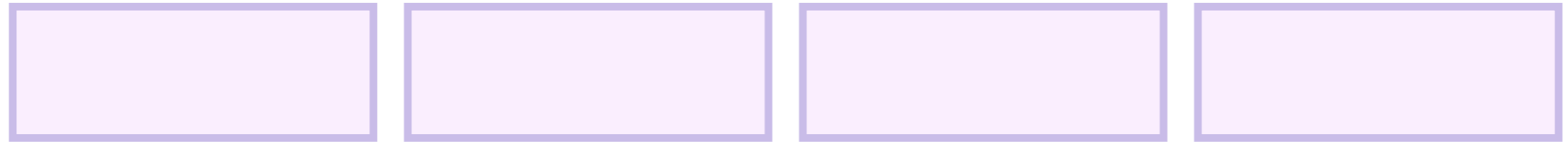
## Example: address bar correctness

```
forall trace tab domain,  
  quark_produced(trace)     $\wedge$   
  tab = cur_tab(trace)      $\wedge$   
  ...
```

*and tab* is the selected  
tab in this trace

# Quark Kernel: Code, *Spec*, Proof

Specify correct behavior wrt syscall seqs



*structure of produceable traces supports spec & proof*

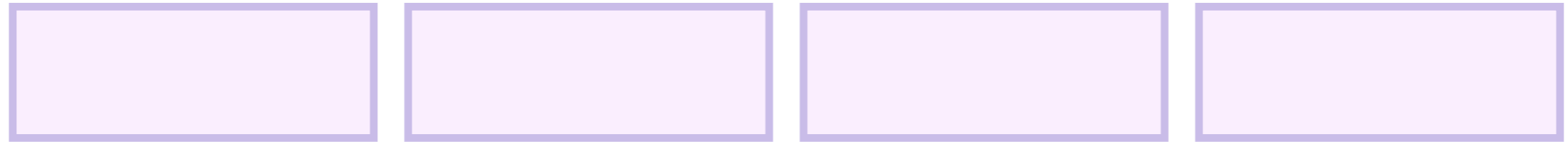
## Example: address bar correctness

```
forall trace t  
  quark_produce  
  tab = cur_tab(trace) ^  
  domain = addr_bar(trace) ->  
  ...
```

*and domain displayed in  
address bar for this trace*

# Quark Kernel: Code, *Spec*, Proof

Specify correct behavior wrt syscall seqs



*structure of produceable traces supports spec & proof*

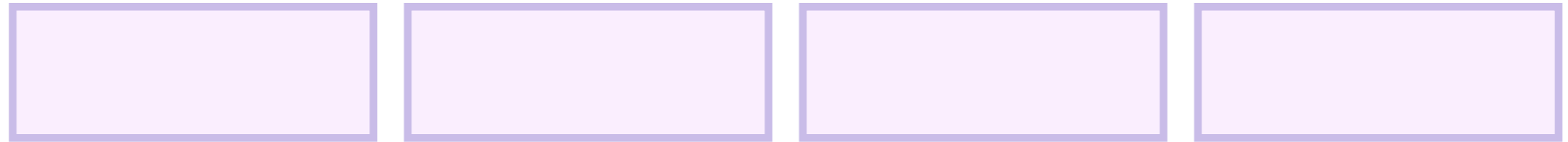
## Example: address bar correctness

```
forall trace tab do
  quark_produced(tr
  tab = cur_tab(trace)
  domain = addr_bar(trace)
  domain = tab_domain(tab)
```

*then domain is the  
domain of the  
focused tab*

# Quark Kernel: Code, *Spec*, Proof

Specify correct behavior wrt syscall seqs



*structure of produceable traces supports spec & proof*

## Example: address bar correctness

```
forall trace tab domain,  
  quark_produced(trace)      ^  
  tab = cur_tab(trace)       ^  
  domain = addr_bar(trace)   ->  
  domain = tab_domain(tab)
```

# Quark Kernel: Code, *Spec*, Proof

## Formal Security Properties

### Tab Non-Interference

*no tab affects kernel interaction with another tab*

### Cookie Confidentiality and Integrity

*cookies only accessed by tabs of same domain*

### Address Bar Integrity and Correctness

*address bar accurate, only modified by user action*

# Quark Kernel: Code, *Spec*, Proof

# Quark Kernel: Code, Spec, *Proof*



# Quark Kernel: Code, Spec, *Proof*

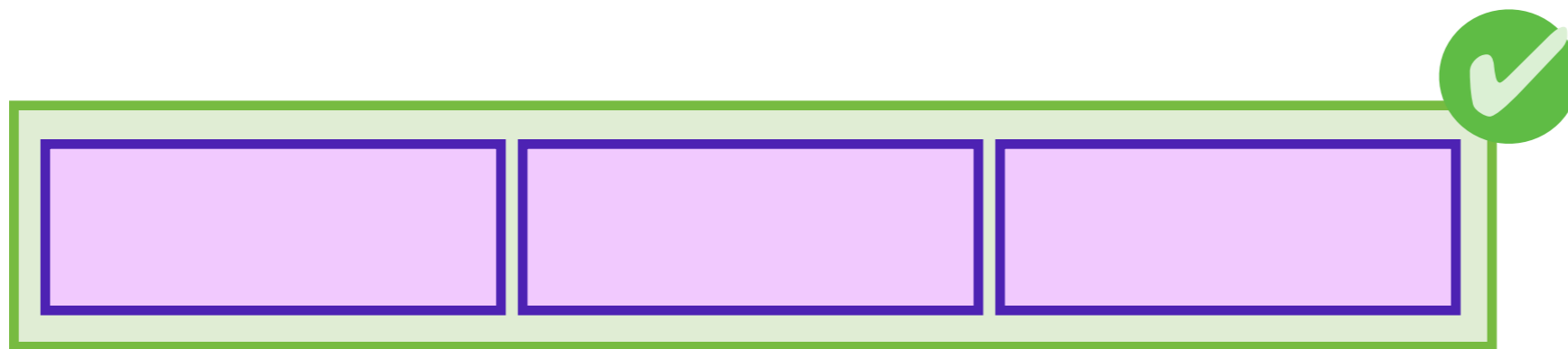
Prove kernel code satisfies sec props

*by induction on traces Quark can produce*

# Quark Kernel: Code, Spec, *Proof*

Prove kernel code satisfies sec props

*by induction on traces Quark can produce*



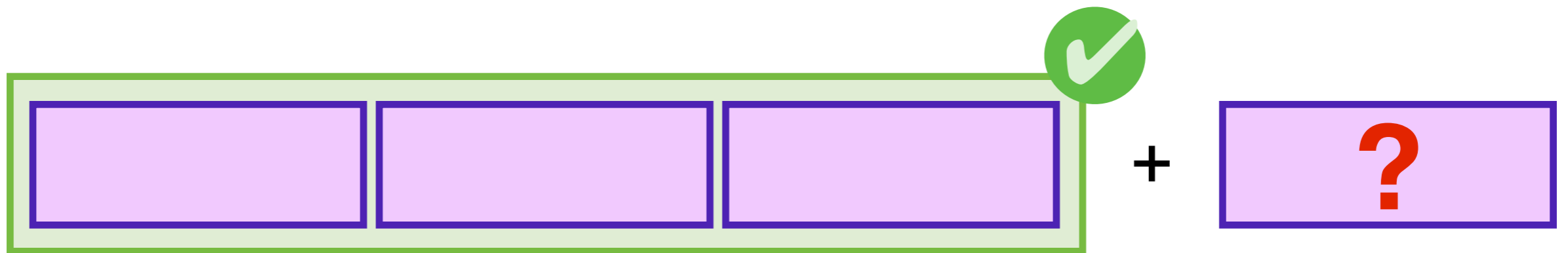
*induction hypothesis:*

*trace valid up to this point*

# Quark Kernel: Code, Spec, *Proof*

Prove kernel code satisfies sec props

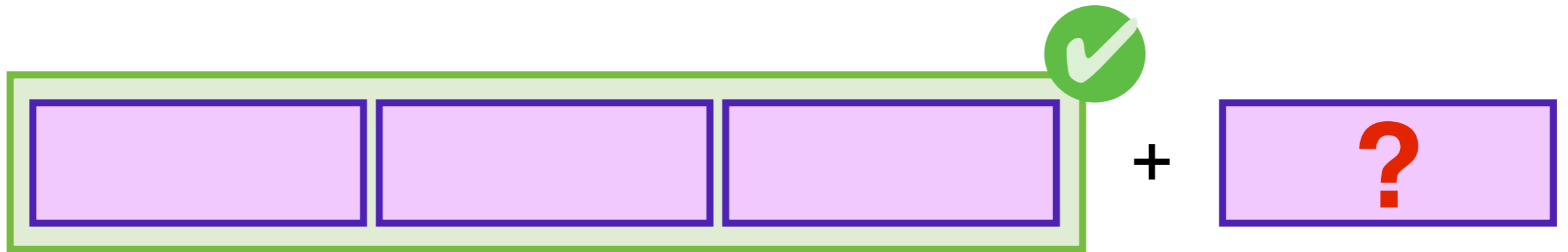
*by induction on traces Quark can produce*



*induction hypothesis:  
trace valid up to this point*

*proof obligation:  
still valid after step?*

# Quark Kernel: Code, Spec, *Proof*



*induction hypothesis:  
trace valid up to this point*

*proof obligation:  
still valid after step?*

Proceed by case analysis on `kstep()`

*what syscalls can be appended to trace?*

*will they still satisfy all security properties?*

*prove each case interactively in proof assistant*

# Quark Kernel: Code, Spec, *Proof*

Proving required diverse range of tools

*monads* encoding I/O in functional language

*Hoare logic* reasoning about imperative programs

*op. semantics* defining correctness of Quark kernel

*linear logic* proving resources created / destroyed

**YNot**

[Naneveski et al. ICFP 08]

# Quark Kernel: Code, Spec, Proof

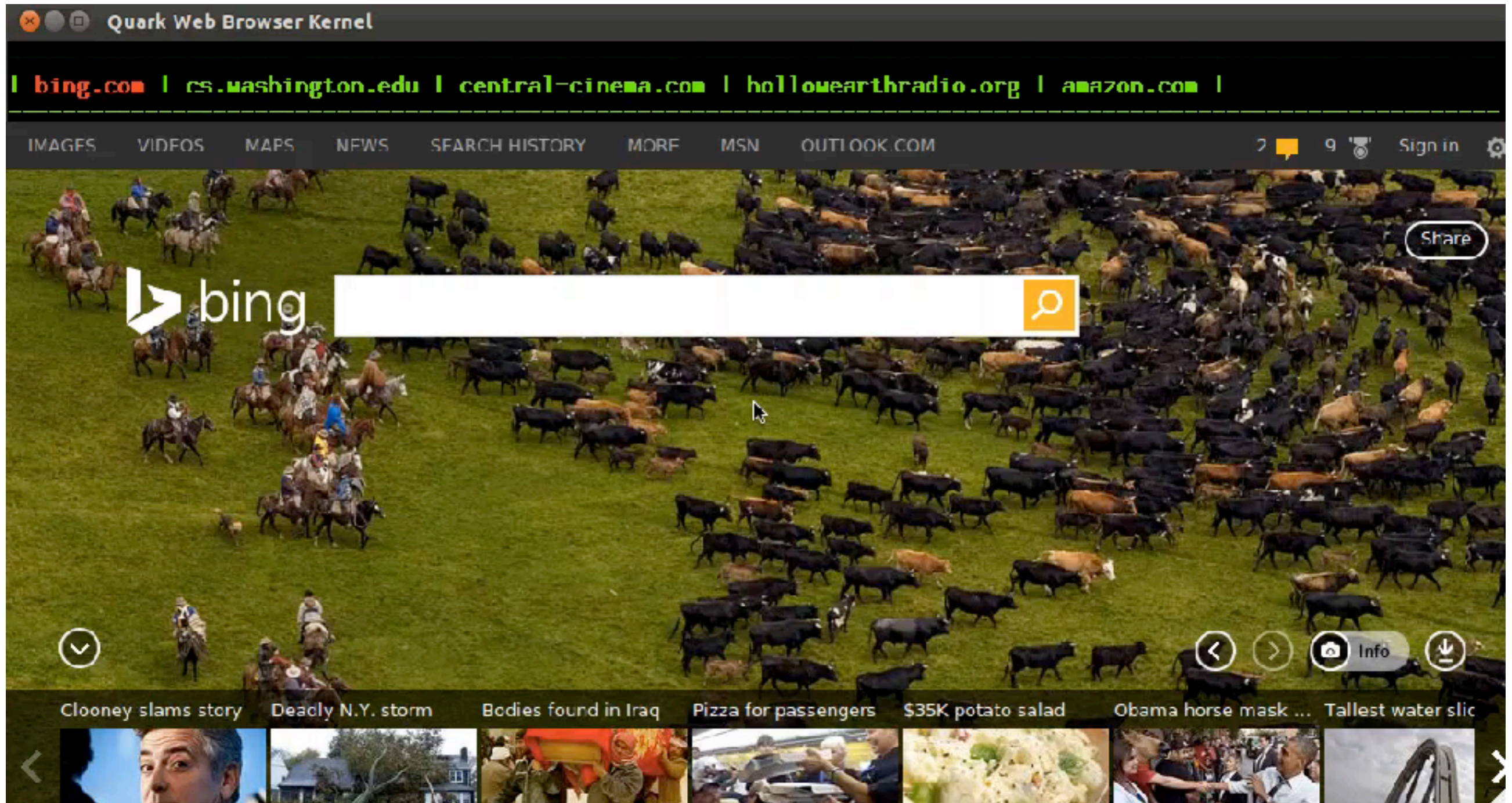
Key Insight: *FSV Effective*

Guarantee sec props for browser

Use state-of-the-art components

Only prove simple browser kernel

# Formally Verified Browser!



# Extending Quark

Filesystem access, sound, history

*could be implemented w/out major redesign*

Finer grained resource accesses

*support mashups and plugins*

Liveness properties

*no blocking, kernel eventually services all requests*



# Trusted Computing Base

Infrastructure we assume correct

*bugs here can invalidate our formal guarantees*

Fundamental

Statement of security properties  
Coq (soundness, proof checker)

---

Eventually  
Verified  
[active research]

OCaml [VeriML]  
Tab Sandbox [RockSalt]  
Operating System [seL4]

...

# Quark Development Effort

150 lines of security props

900 lines of kernel code

4,500 lines of proofs

1,000,000 lines of WebKit

# Quark Development Effort

150 lines of security

week

900 lines of kernel code

4,500 lines of proofs

1,000,000 lines of WebAssembly

months

# Mitigating the Burden of Proof

1: Scaling proofs to critical infrastructure

*Formal shim verification for large apps*

➔ *QUARK: browser with security guarantees*

2: Evolving formally verified systems

*Reflex DSL exploits domain for proof auto*

# Mitigating the Burden of Proof

1: Scaling proofs to critical infrastructure

*Formal shim verification for large apps*

*QUARK: browser with security guarantees*

2: Evolving formally verified systems

➔ *Reflex DSL exploits domain for proof auto*

# Struggle Against Formality Inertia

Adding cookies to Quark quite difficult  
*all the pieces already there, still took over a month*

Proof updates repetitive and shallow  
*sensitive proof scripts, changes not mechanical*

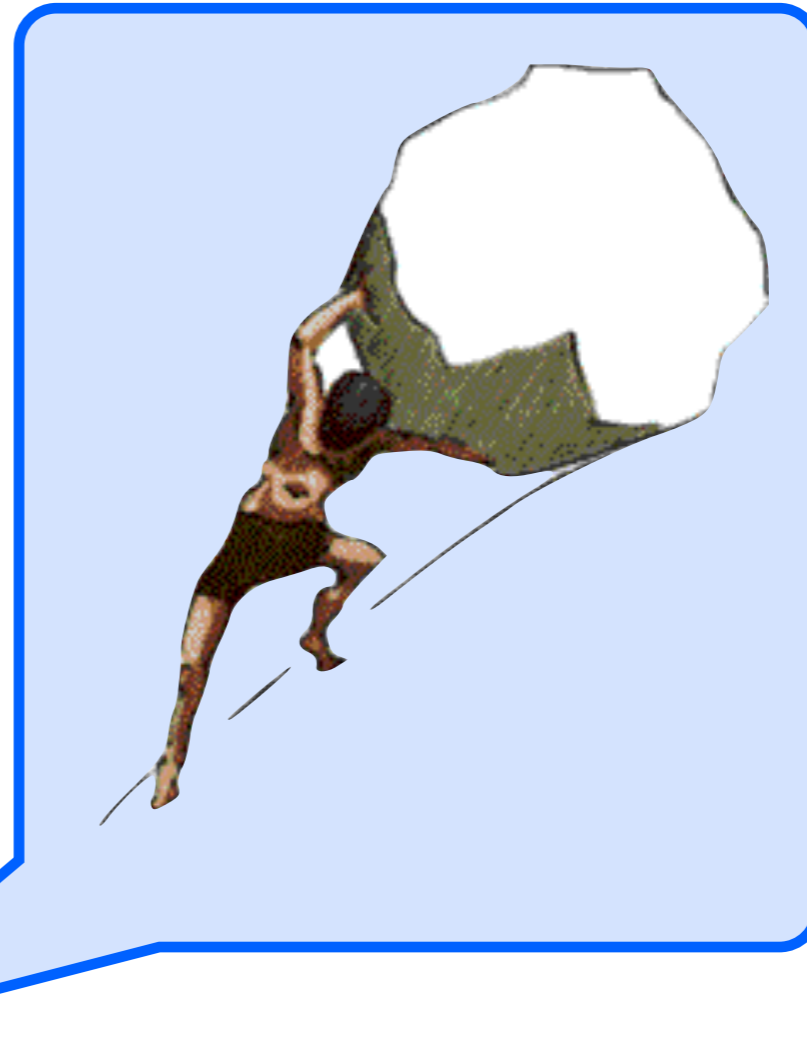
```
match svec_ith PAYREST i as _vi return
  forall (EQ: (svec_ith (projT2 (existT vdesc' ENVD_SIZE PAYREST)) i) = _vi),
  match _vi as __d return (base_term (existT vdesc' ENVD_SIZE PAYREST) __d -> Prop)
  with
  | Desc d => fun _ => True
  | Comp c => fun b=> FdSet.In
    (comp_fd (projT1 (eval_base_term (envd:=existT _ ENVD_SIZE PAYREST) erest b))) fds end
  match EQ in _ = __vi return base_term _ __vi with Logic.eq_refl =>
    Var (existT vdesc' ENVD_SIZE PAYREST) i end
->
match _vi as __d return (base_term (existT vdesc' (S ENVD_SIZE) (PAY0, PAYREST)) __d -> Prop) with
| Desc d => fun _ => True
| Comp c => fun b =>
  FdSet.In (comp_fd (projT1 (eval_base_term (envd:=existT _ (S ENVD_SIZE) (PAY0, PAYREST)) (e0, erest) b))) fds end
  match EQ in _ = __vi return base_term _ __vi with Logic.eq_refl =>
    Var (existT vdesc' (S ENVD_SIZE) (PAY0, PAYREST)) (Some i) end
with
| Desc d => _ | Comp c => _ end (Logic.eq_refl _)
```

# Division of Labor *(to scale)*

Spec

Code

Proof



# Division of Labor

Ideal?

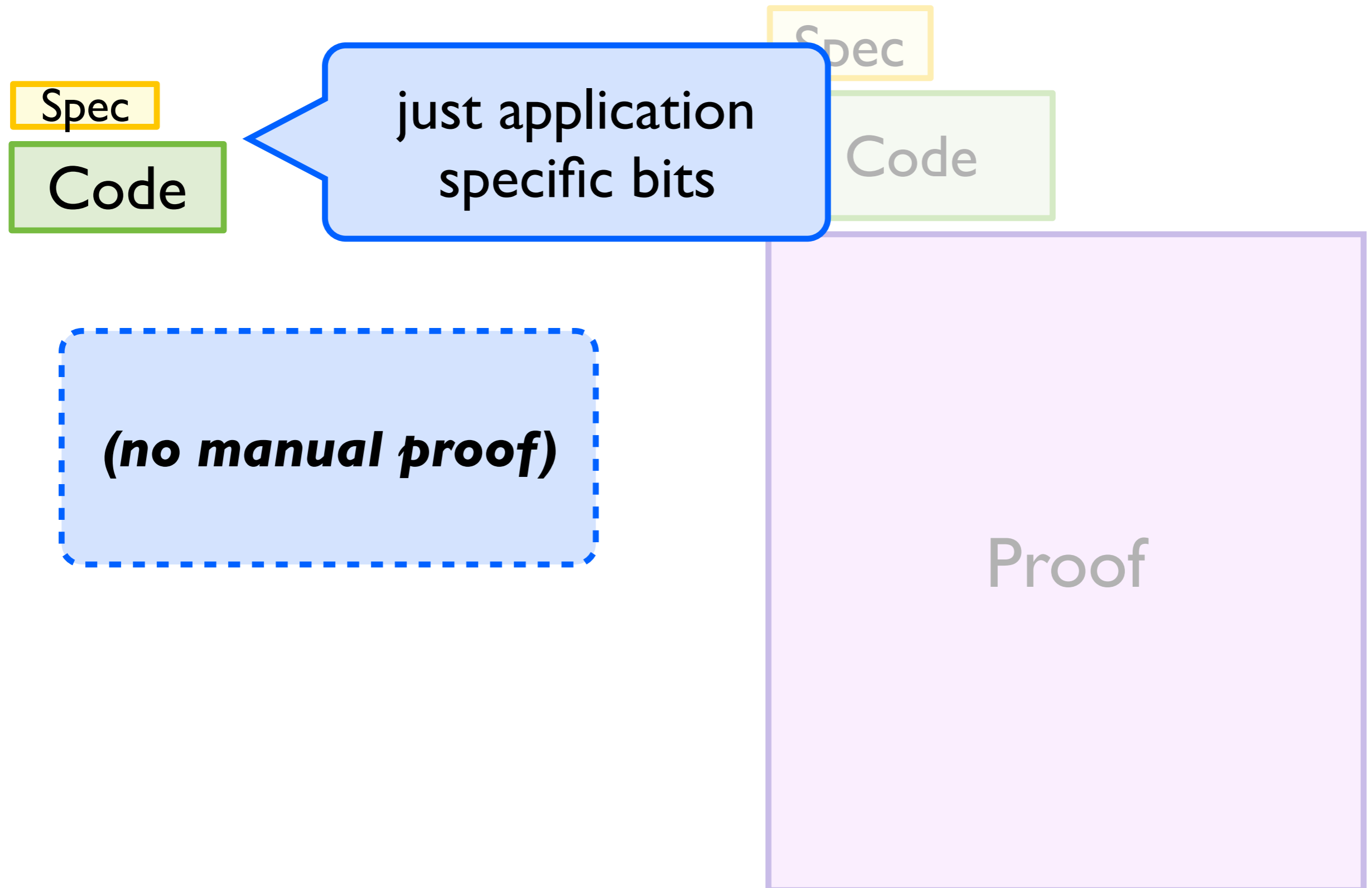
Spec

Code

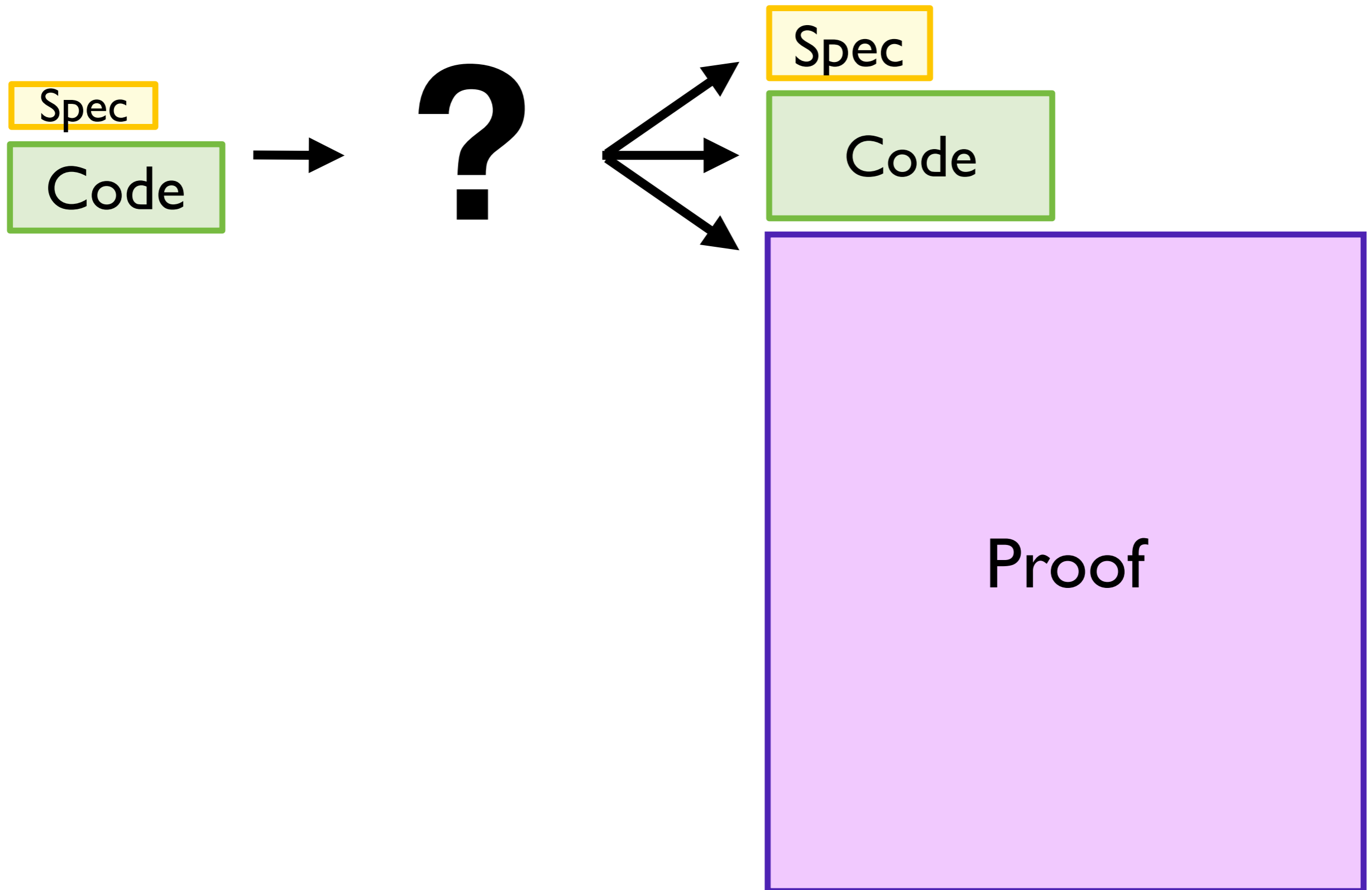
Proof



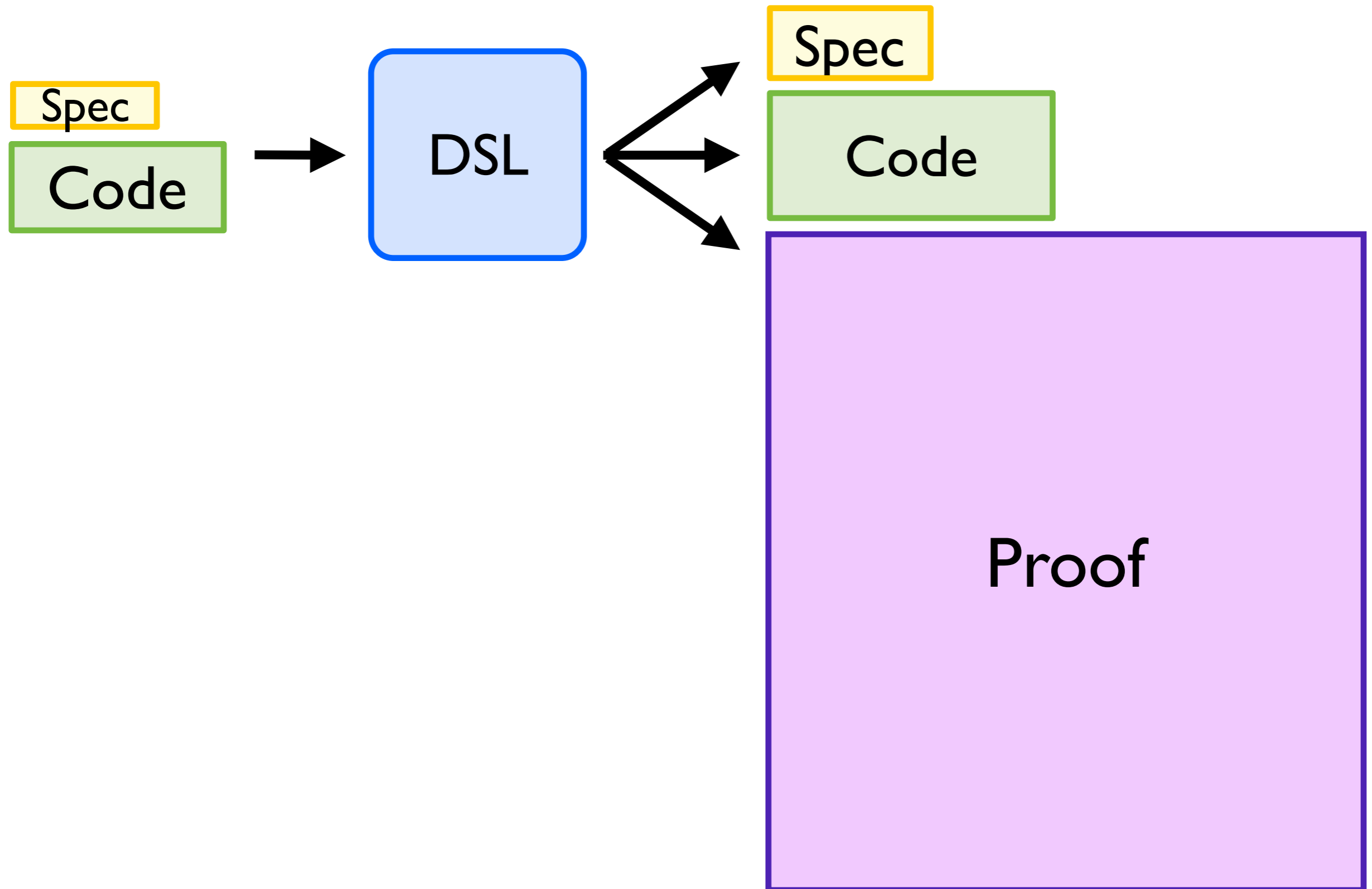
# Division of Labor



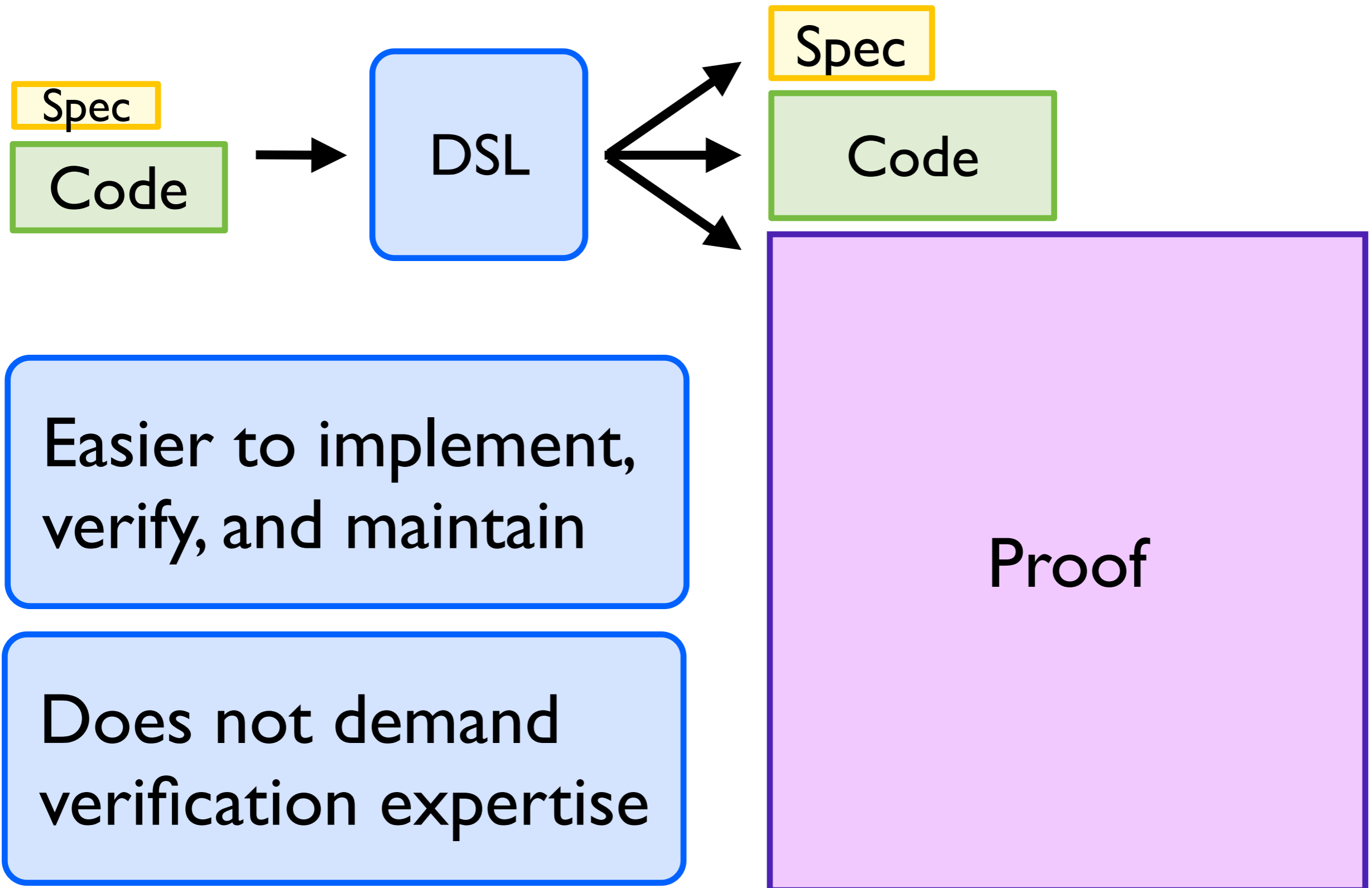
# Division of Labor



# Division of Labor



# Division of Labor



Easier to implement,  
verify, and maintain

Does not demand  
verification expertise

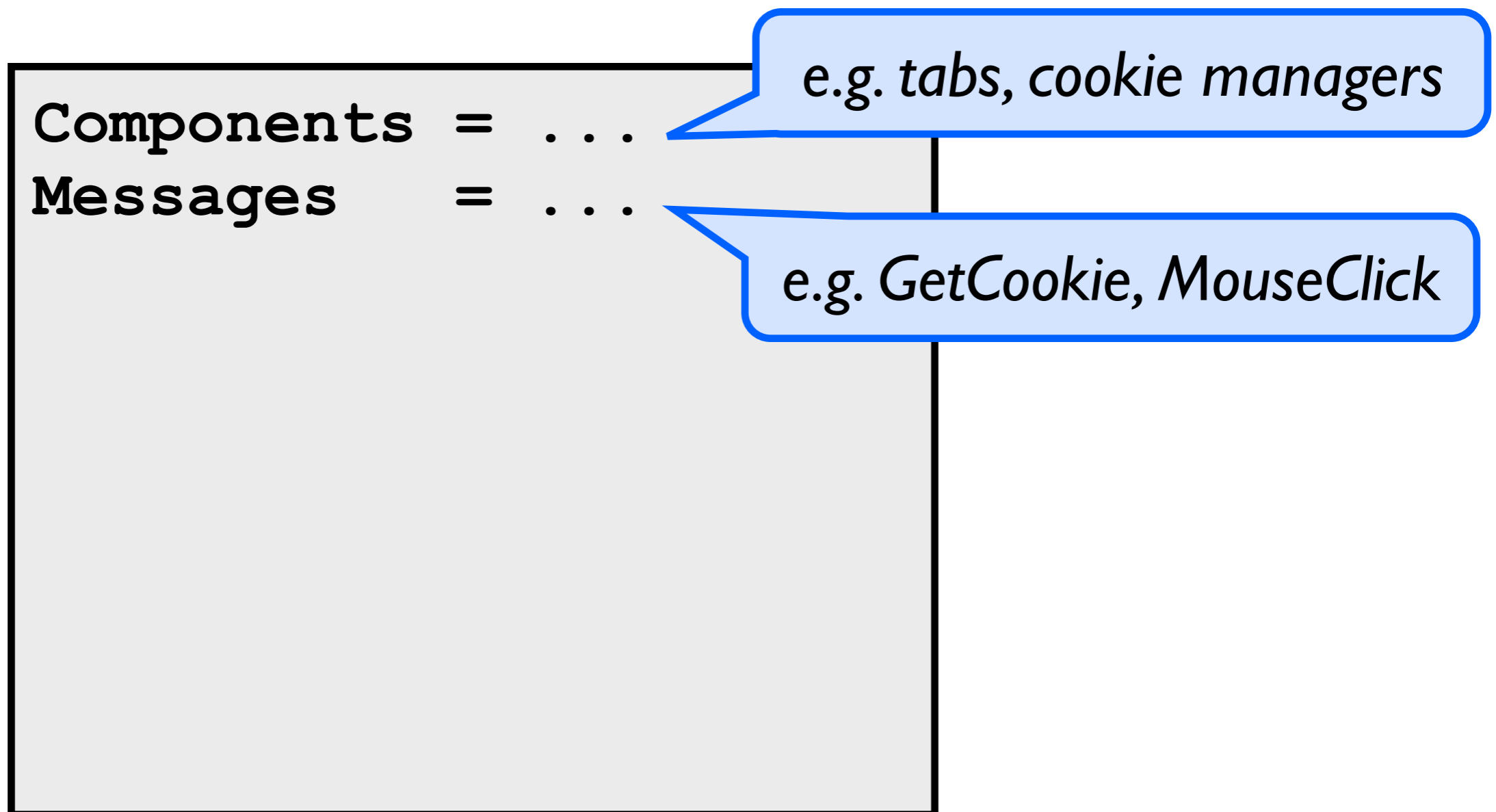
Proof

# Reflex: a DSL for Reactive Systems

[PLDI 14]

Exploit structure of app domain

*kernel based archs, well suited to FSV design*

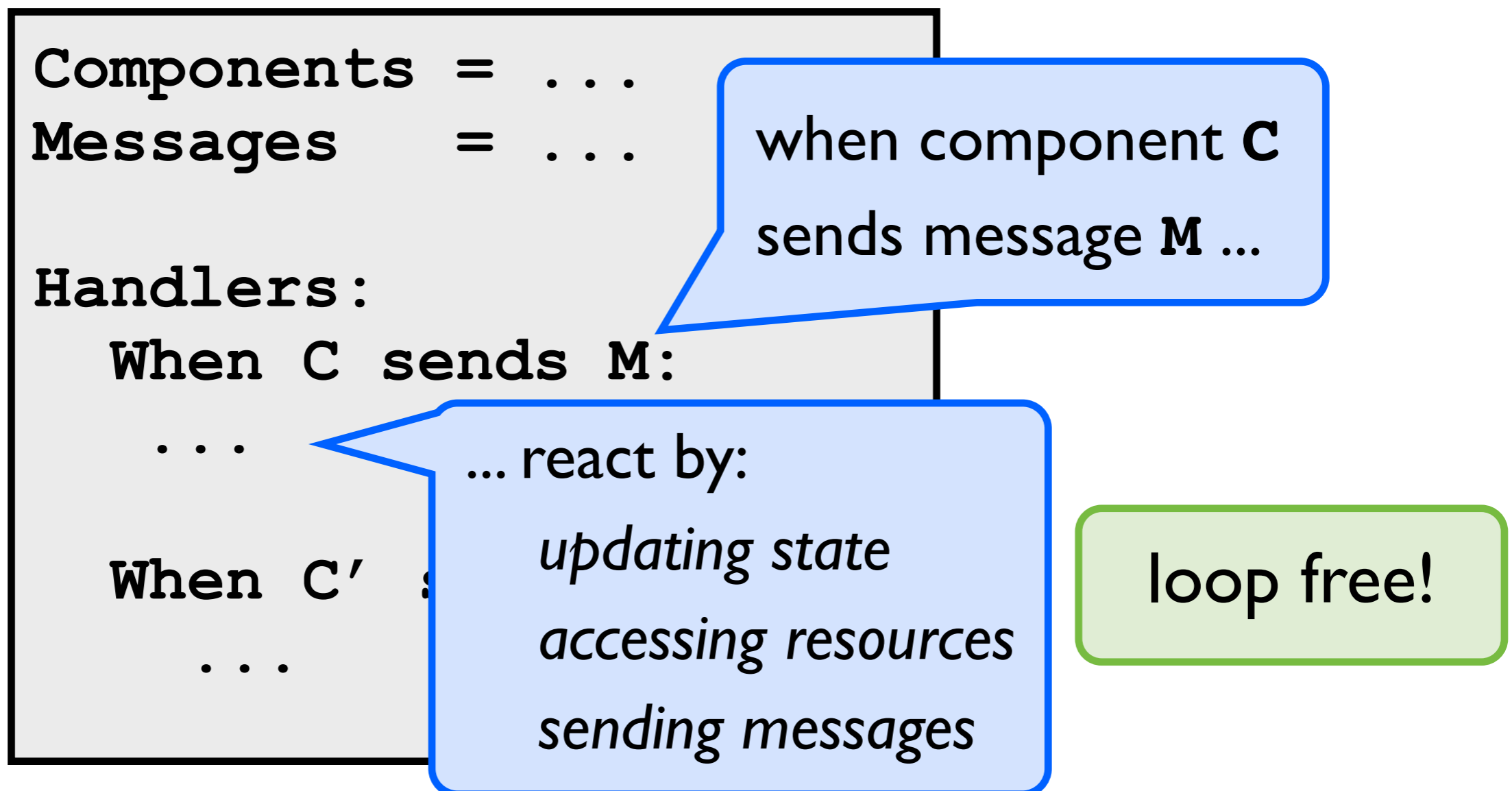


# Reflex: a DSL for Reactive Systems

[PLDI 14]

Exploit structure of app domain

*kernel based archs, well suited to FSV design*



# Reflex: a DSL for Reactive Systems

[PLDI 14]

Exploit structure of app domain

*kernel based archs, well suited to FSV design*

Provide expressive spec language

*subset of LTL and non-interference properties*

```
forall d c,  
  [Recv(Tab(d), CookieSet(c))]  
  Enables  
  [Send(CookieMgr(d), CookieSet(c))]
```

cookie  
integrity

# Reflex: a DSL for Reactive Systems

[PLDI 14]

Exploit structure of app domain

*kernel based archs, well suited to FSV design*

Provide expressive spec language

*subset of LTL and non-interference properties*

Auto prove user-provided specs

*exploit domain, ensure all traces match spec*

Counterexample-driven search discovers invariants.



# Reflex: a DSL for Reactive Systems

[PLDI 14]

## *Reflex Effective:*

Prototype sshd, browser, httpd

Specify basic access controls

Auto prove user-provided specs

# Reflex: Evaluation

<b>Web browser</b>	Domains do not interfere, Cookie integrity, ... <i>auto prove non-interference</i>
<b>SSH server</b>	No PTY access before authentication, At most 3 authentication attempts, ...
<b>Web server</b>	Clients only spawned after successful login, File requests guarded by access control, ... <i>auto prove non-local props</i>

*Auto verified 33 properties (80% in < 2 minutes)*

# Reflex: Development Effort

Reflex :

*Many reactive systems*

*7500 lines of Coq*



Quark Web browser :

*5500 lines of Coq*

*Single reactive system*

# Mitigating the Burden of Proof

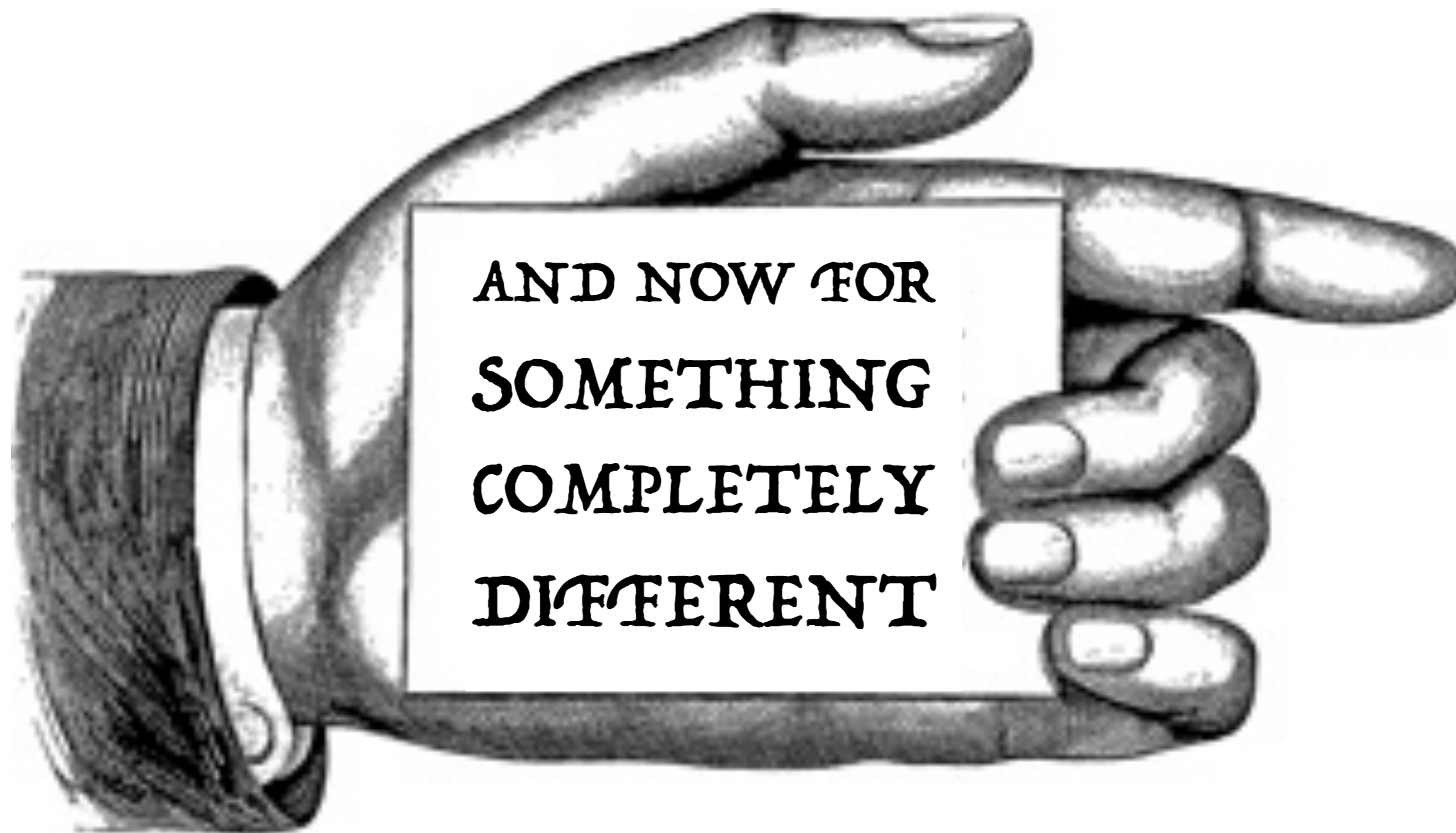
1: Scaling proofs to critical infrastructure

*Formal shim verification for large apps*

*QUARK: browser with security guarantees*

2: Evolving formally verified systems

 *Reflex DSL exploits domain for proof auto*



AND NOW FOR  
SOMETHING  
COMPLETELY  
DIFFERENT

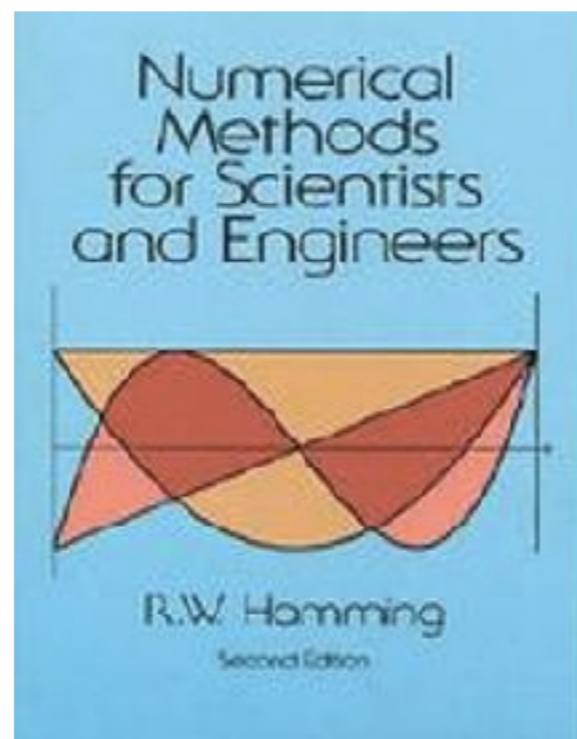
# Double Trouble

```
x = 0.1 + 0.2;  
if (x != 0.3)  
    printf("wat.\n");
```

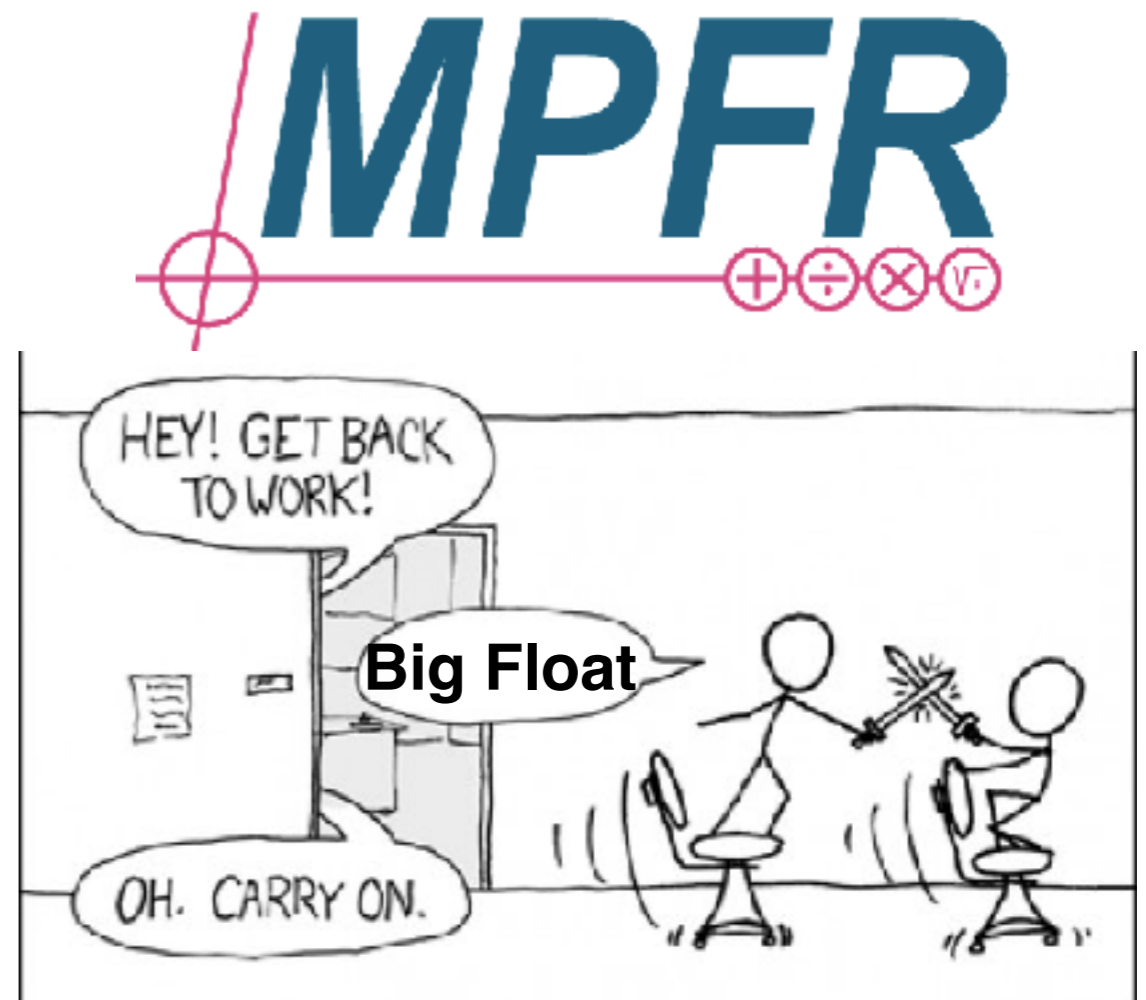
$$\frac{(-b) - \sqrt{b^2 - 4 \cdot (a \cdot c)}}{2 \cdot a}$$



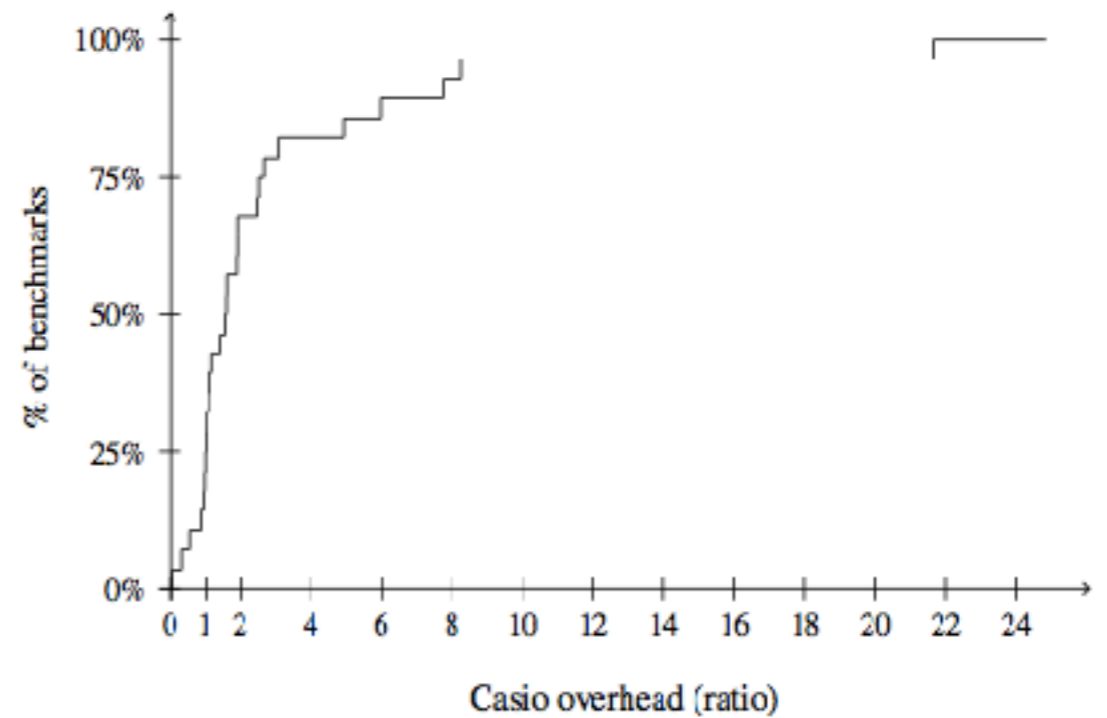
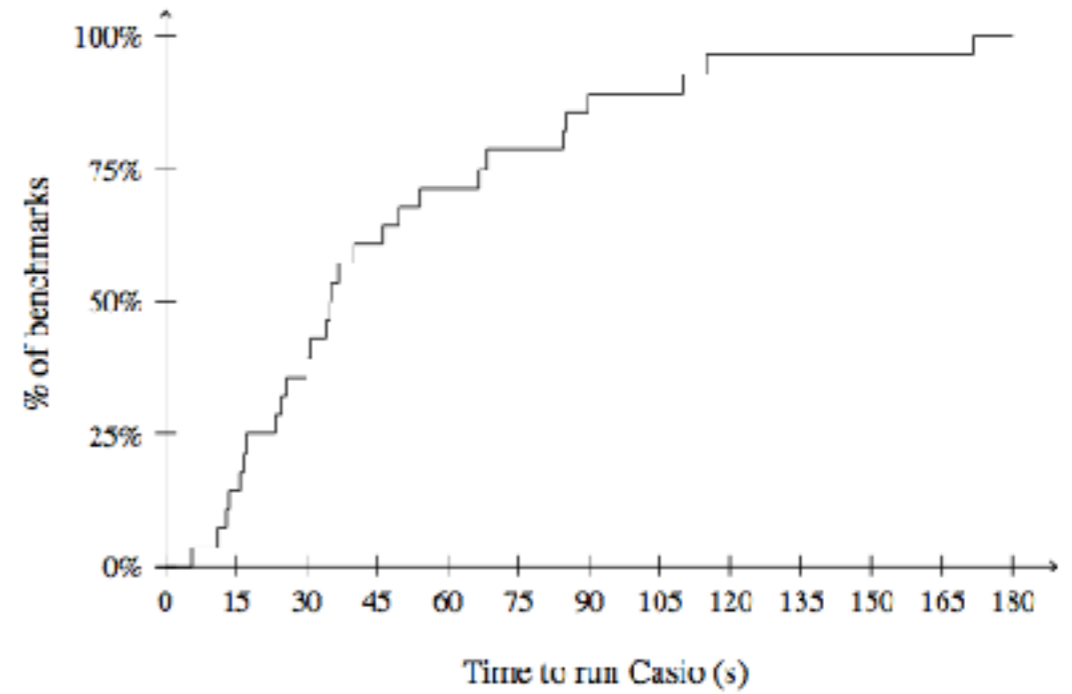
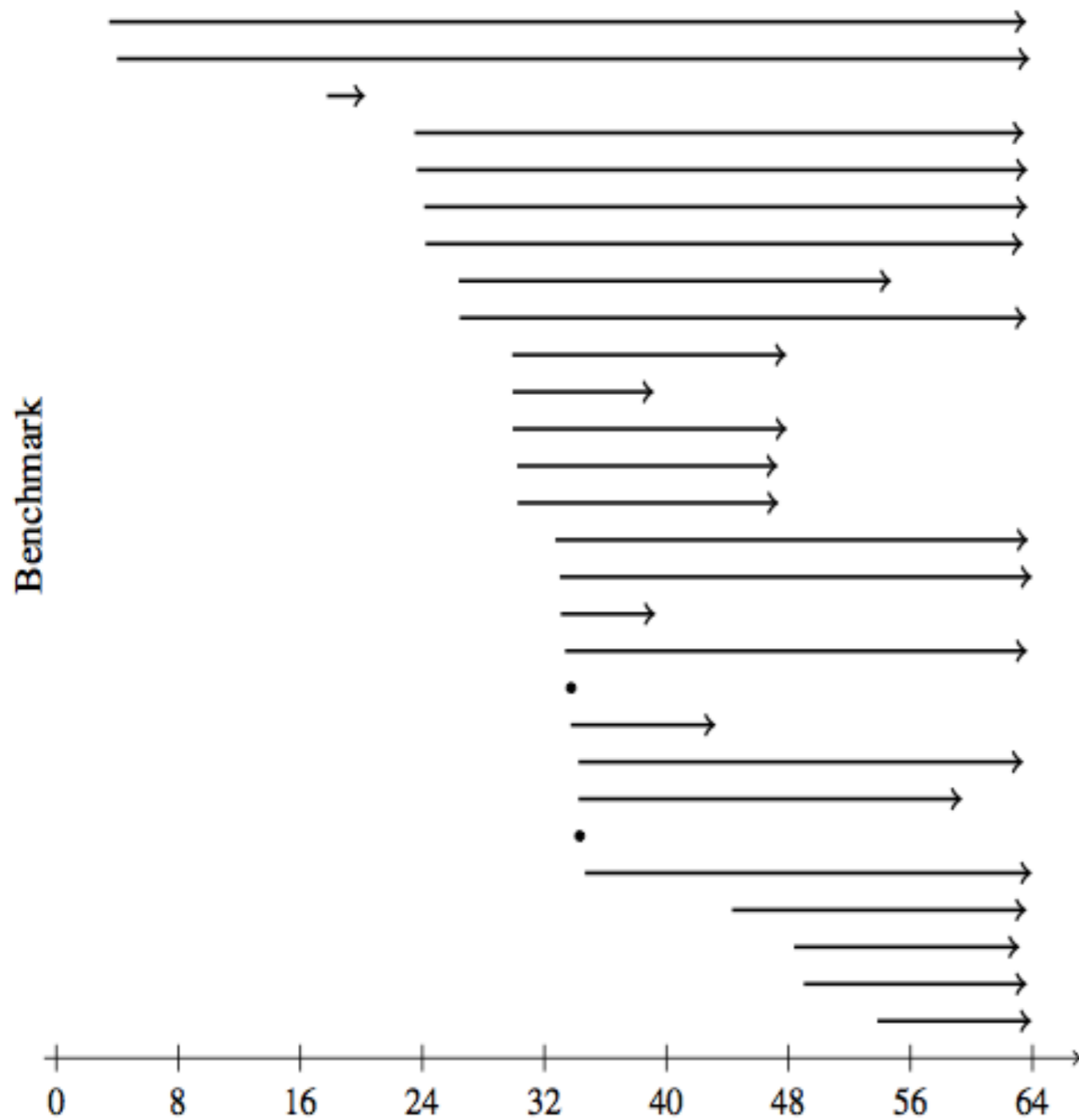
Futz



Analyze

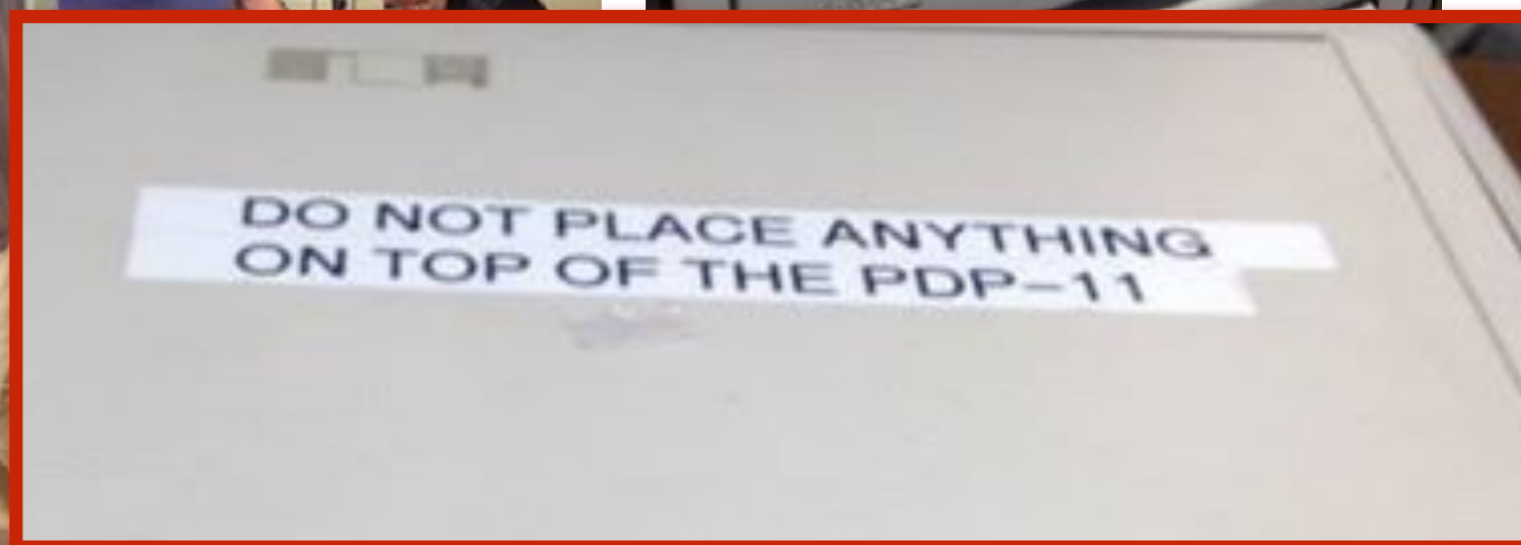
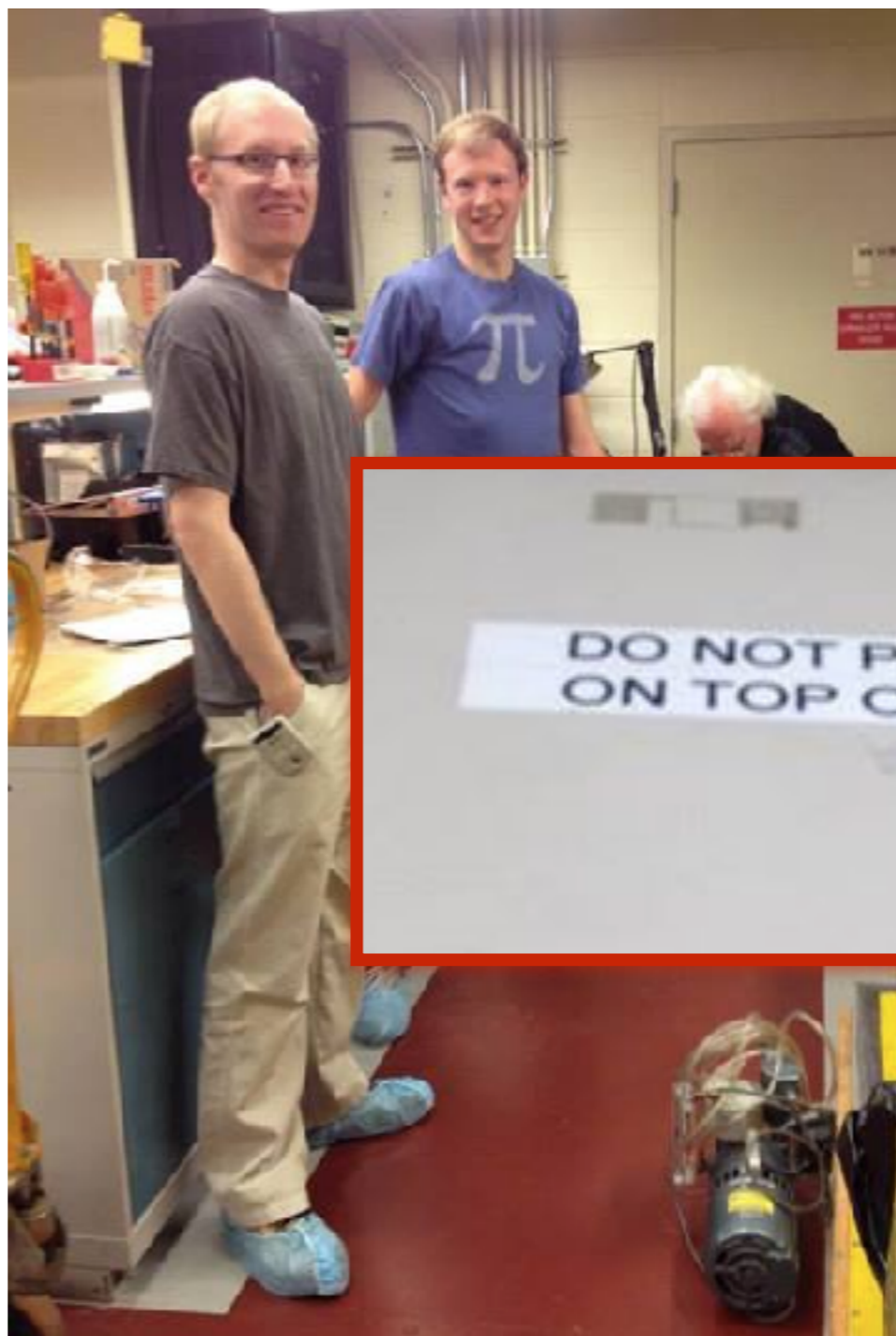


# Less Double Trouble



# Neutron Beams

**UW Medicine**  
SCHOOL OF MEDICINE



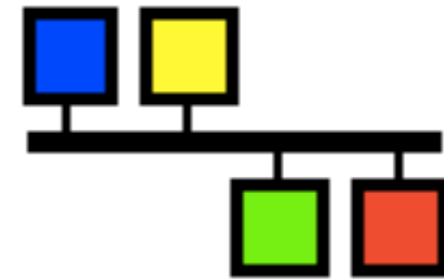


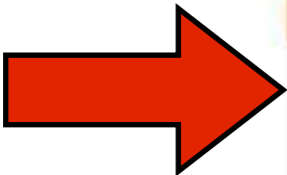
# Neutron Beams

**UW Medicine**  
SCHOOL OF MEDICINE



**EPICS**





**Hacker News** [new](#) | [comments](#) | [ask](#) | [jobs](#) | [submit](#) [login](#)

1. ▲ **Quark : A secure Web Browser with a Formally Verified Kernel** (ucsd.edu)  
141 points by herшел 4 hours ago | 38 comments
2. ▲ **Writing an nginx authentication module in Lua and Go** (stavros.io)  
41 points by StavrosK 2 hours ago | 7 comments
3. ▲ **Code & Conquer: A War Game for Coders** (codeandconquer.co)  
60 points by eliottcarlson 2 hours ago | 13 comments
4. ▲ **Proposal to Change the Default TLS Ciphersuites Offered by Browsers** (briansmith.org)  
78 points by fejr 5 hours ago | 44 comments
5. ▲ **The backlash against running firms like progressive schools has begun** (economist.com)  
26 points by alexfarran 2 hours ago | 16 comments
6. ▲ **Darkness** (wegnerdesign.com)  
41 points by yesplorer 4 hours ago | 14 comments
7. ▲ **Big Data and the Soviet Ghosts** (mempko.wordpress.com)  
36 points by mempko 4 hours ago | 8 comments
8. ▲ **Startup Ideas Every Nerd Has (That Never Work)** (swombat.com)  
5 points by lou 23 minutes ago | discuss
9. ▲ **GCP - cp with a progress bar** (hecticgeek.com)  
18 points by dannyrosen 2 hours ago | 14 comments
10. ▲ **Doing Good in the Addiction Economy** (kajsotala.fi)  
58 points by kaj\_sotala 6 hours ago | 7 comments
11. ▲ **Yahoo says U.S. sought data on 40,322 user accounts in 2013** (washingtonpost.com)  
5 points by tareqak 35 minutes ago
12. ▲ **Setup a Docker Compose** (medium.com)  
14 points by milka21 2 hours ago

**Achievement unlocked**



# Thank You!

**Goal: mitigate formality inertia**

*address scaling and evolving formally verified systems*

**1. Extend verification frontier**

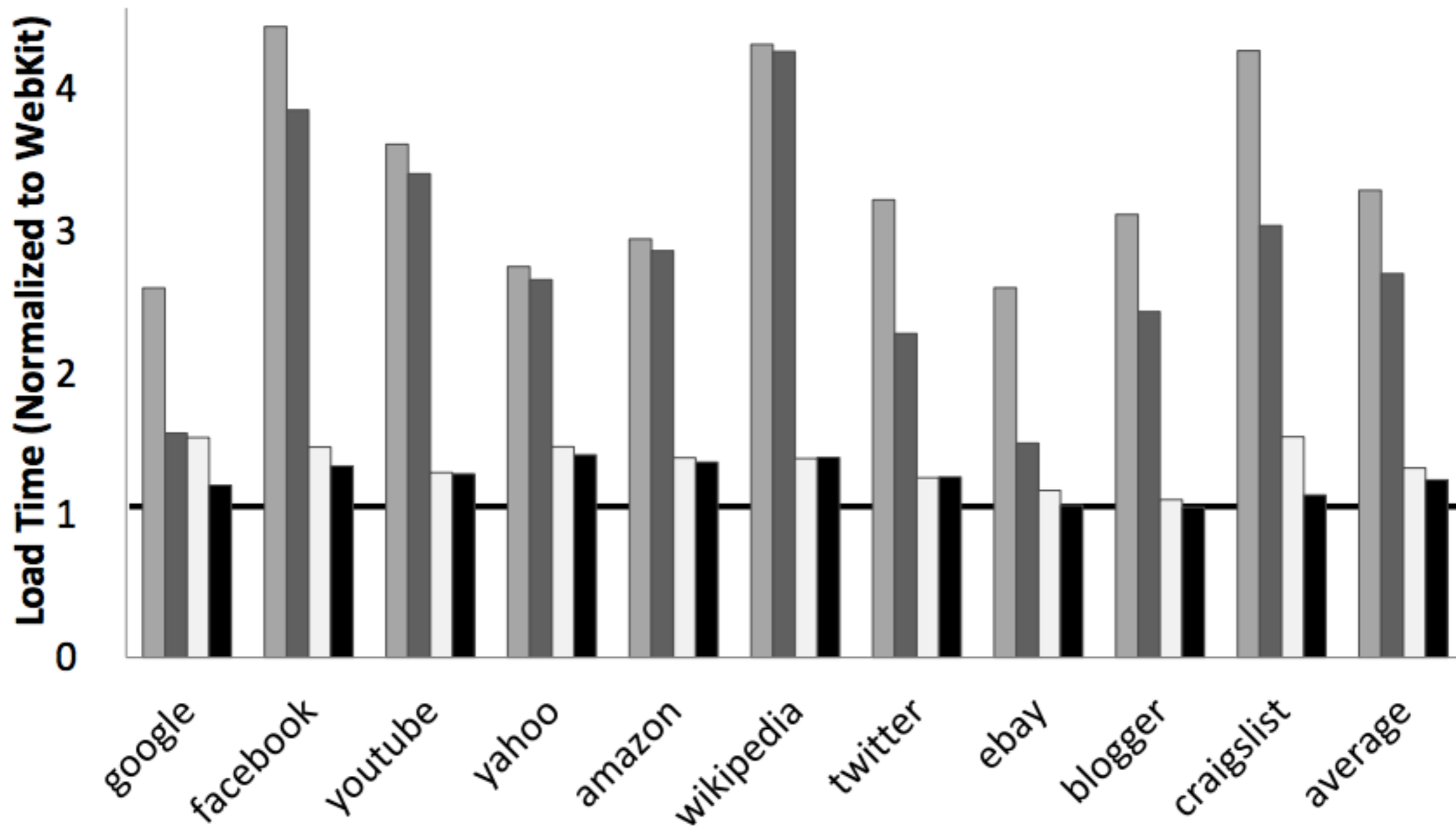
*develop techniques to verify critical “pinch points”*

**2. Make verification accessible**

*equip domain experts with effective tools*



■ not optimized ■ + socket (same origin) □ + socket (whitelist) ■ + cookie cache



# Verifying Optimizations

Rich compiler correctness history:

*McCarthy 67, Samet 75, Cousot 77, ...*

Already solved?

<i>Compiler</i>	<i>Bugs Found</i>
GCC	122
LLVM	181
CompCert	0

many  
optimization  
bugs

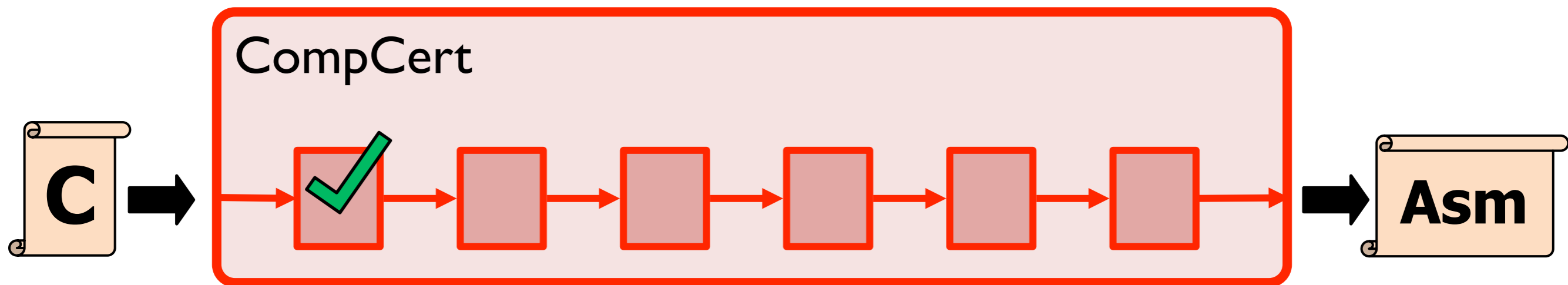
lacks many  
optimizations

[Yang et al. PLDI 11]

# Verifying Optimizations

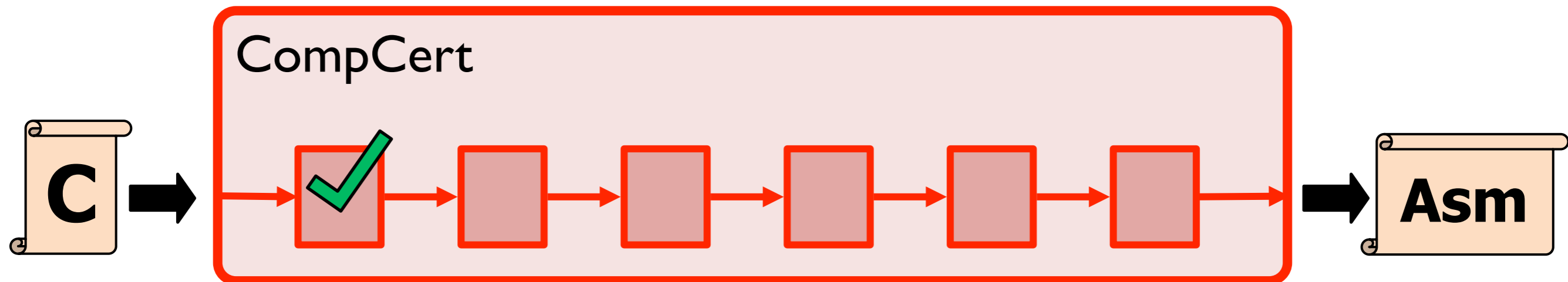
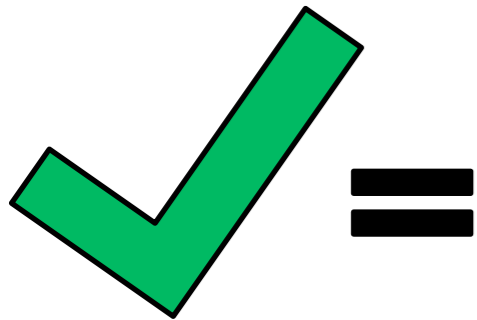


# Verifying Optimizations



# Verifying Optimizations

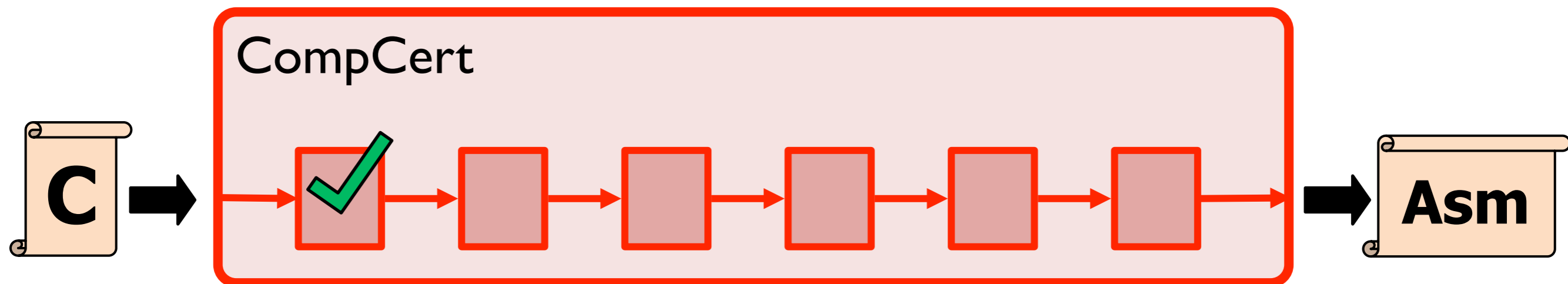
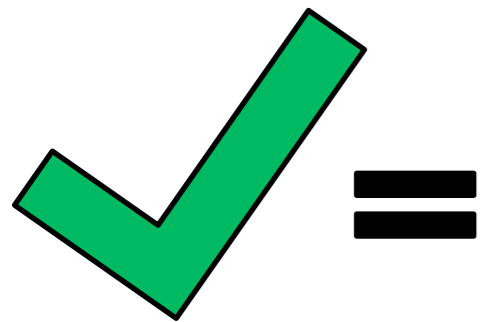
Proof original and opt code equivalent.



# Verifying Optimizations

Proof original and opt code equivalent.

Construct *bisimulation relation*:



# Verifying Optimizations

Proof original and opt code equivalent.

Construct if orig and opt in *ation*:

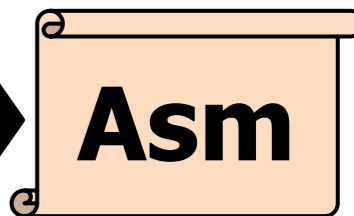
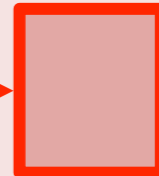
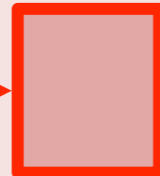
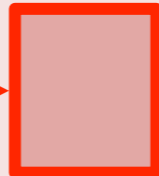
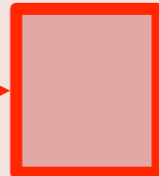
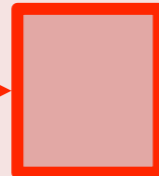
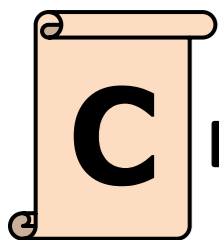
equal states



=



CompCert



# Verifying Optimizations

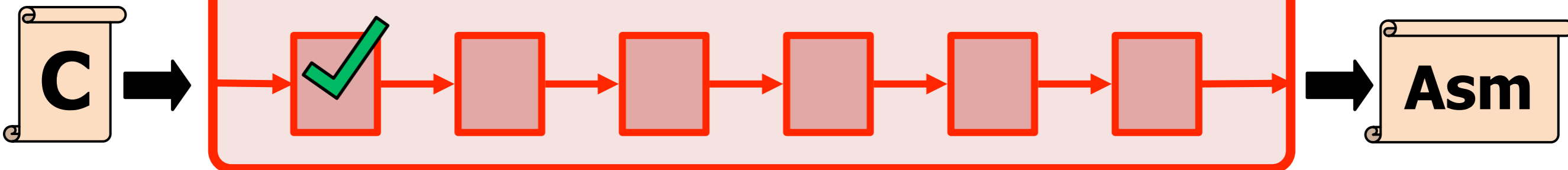
Proof original and opt code equivalent.

Construct if orig and opt in *equivalent* states:



and orig prog can take some action

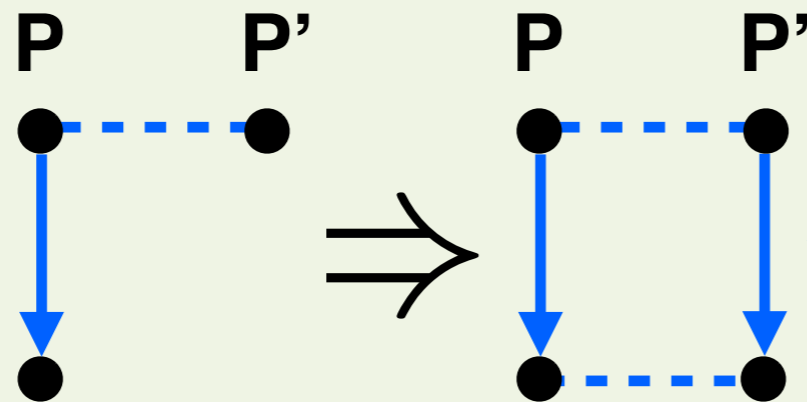
CompCert



# Verifying Optimizations

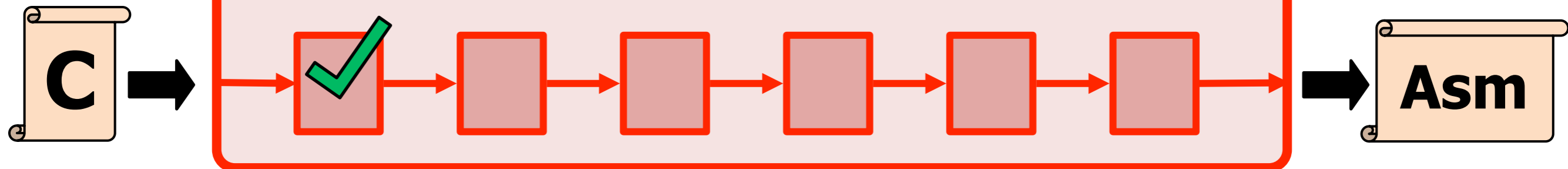
Proof original and opt code equivalent.

Construct *bisimulation relation*:



*then opt prog can take same action to another equal state*

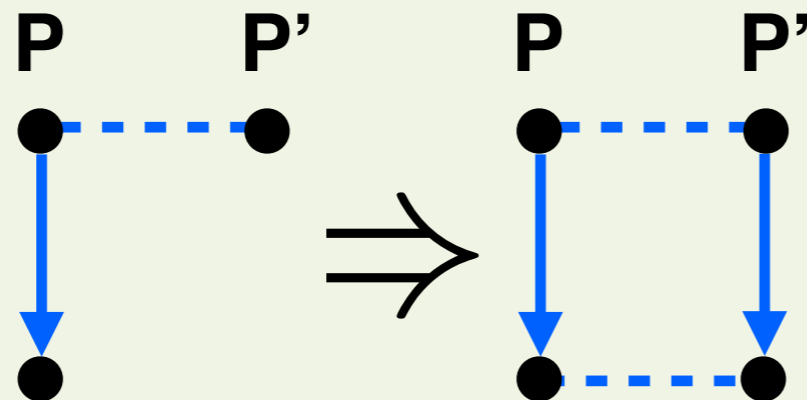
CompCert



# Verifying Optimizations

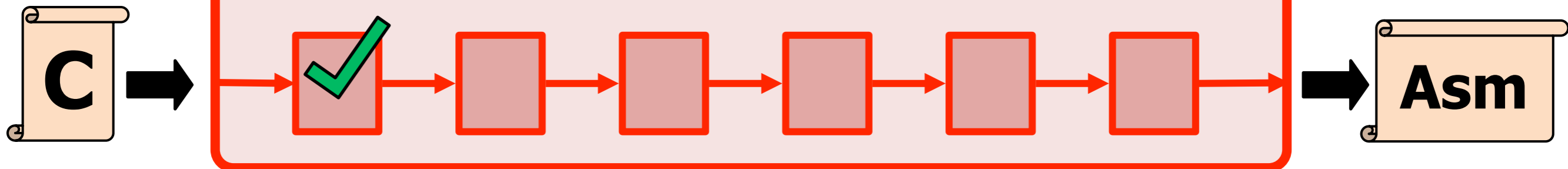
Proof original and opt code equivalent.

Construct *bisimulation relation*:



implies: *anything orig can do, opt can do too*

CompCert



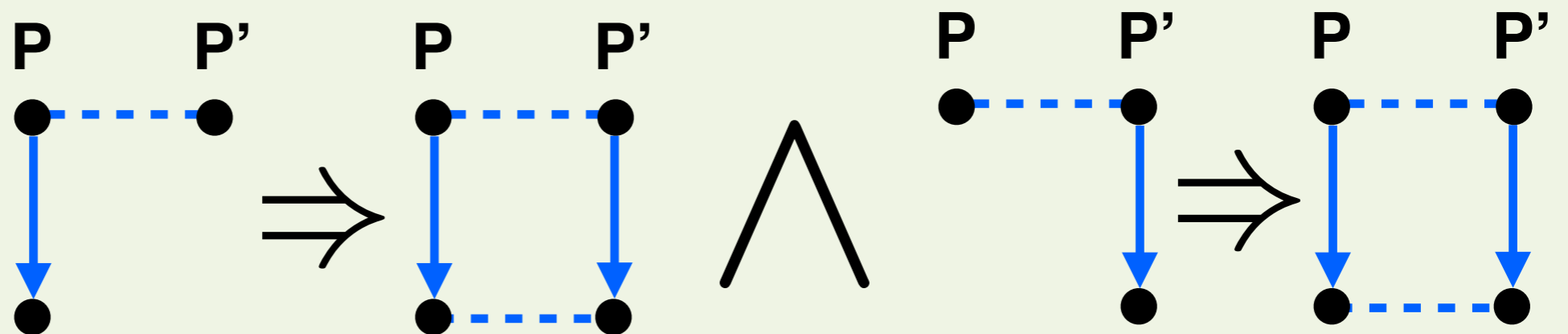




# Verifying Optimizations

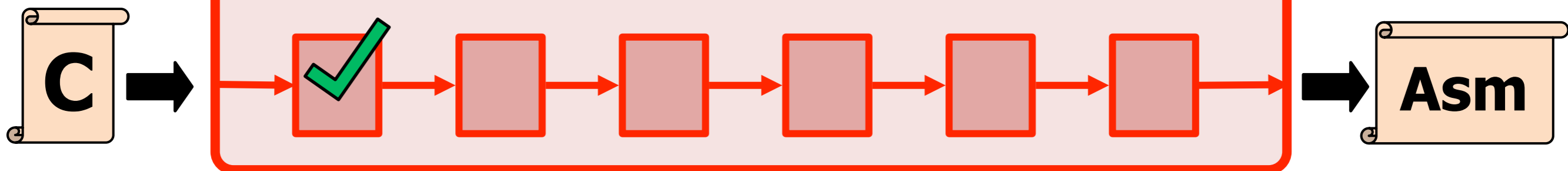
Proof original and opt code equivalent.

Construct *bisimulation relation*:



together, implies *indistinguishability*

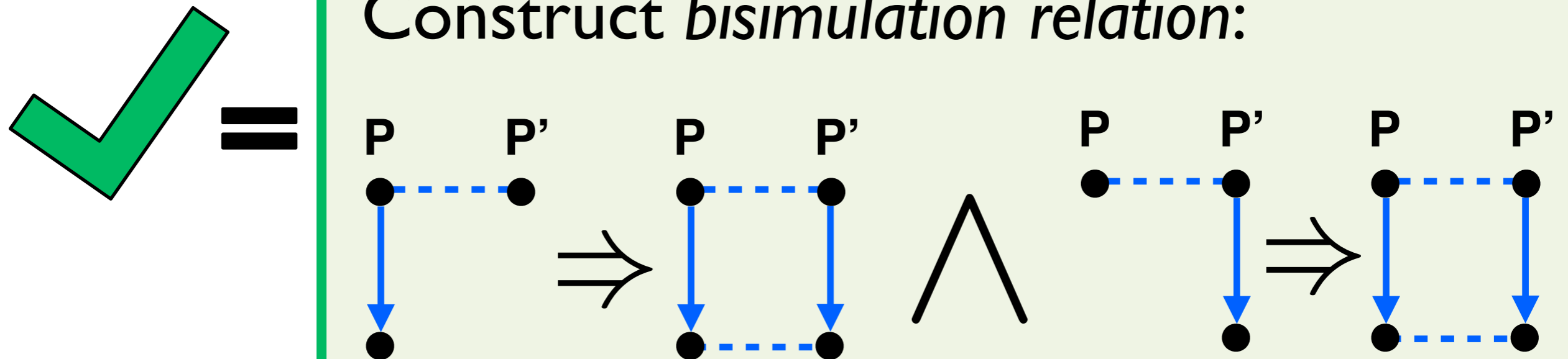
CompCert



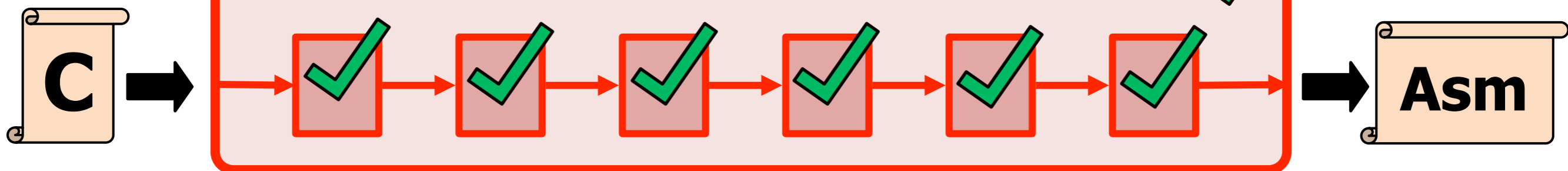
# Verifying Optimizations

Proof original and opt code equivalent.

Construct *bisimulation relation*:



CompCert

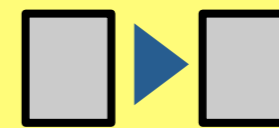


# Verifying Optimizations

Formally Proved:

Rewrites locally correct

$\Rightarrow \exists$  bisimulation relation



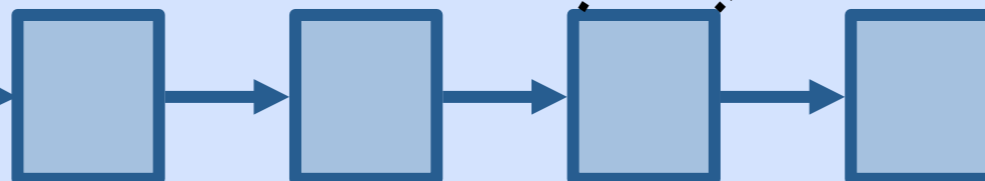
**Rewrite**



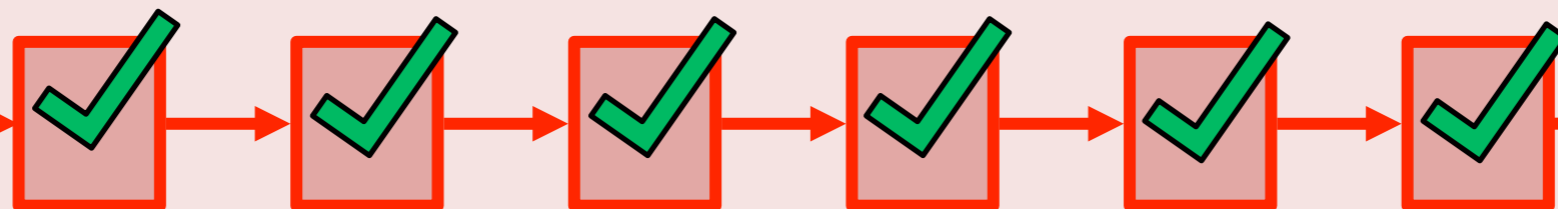
**Local Proofs**

CompCert

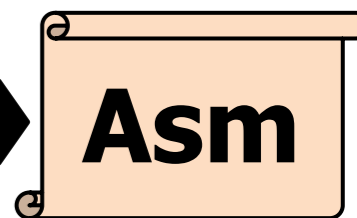
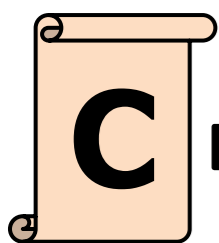
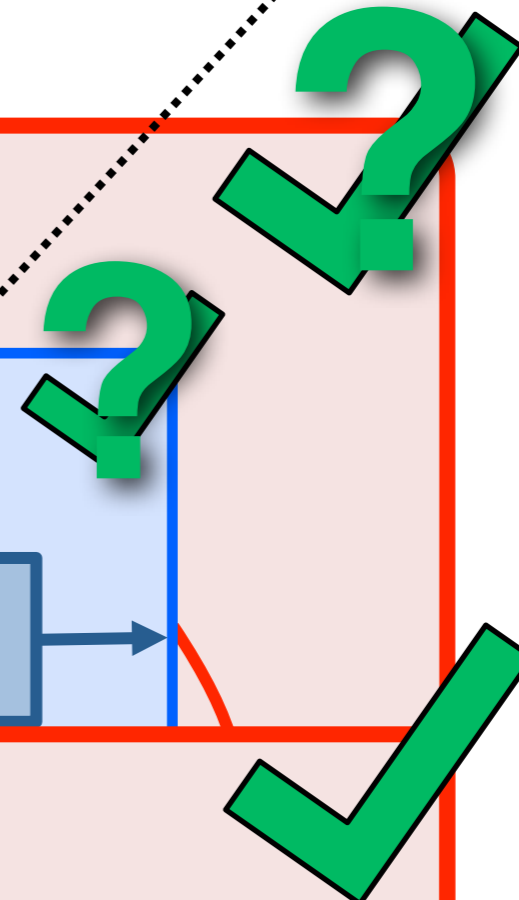
XCert



CompCert

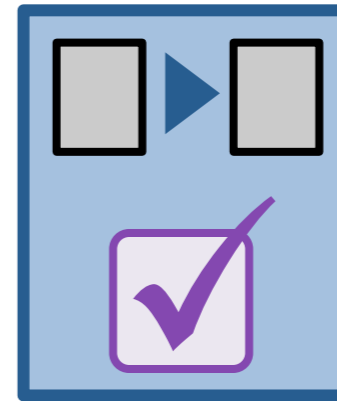
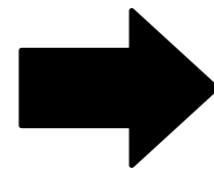
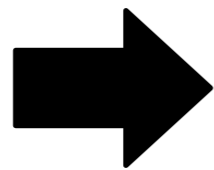
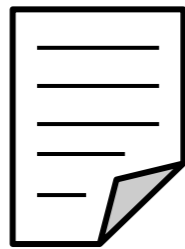


**Asm**



# Verifying Optimizations

Rewrite Rule

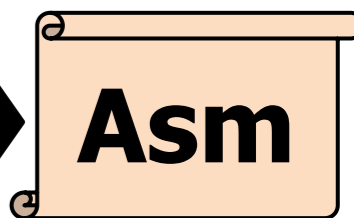
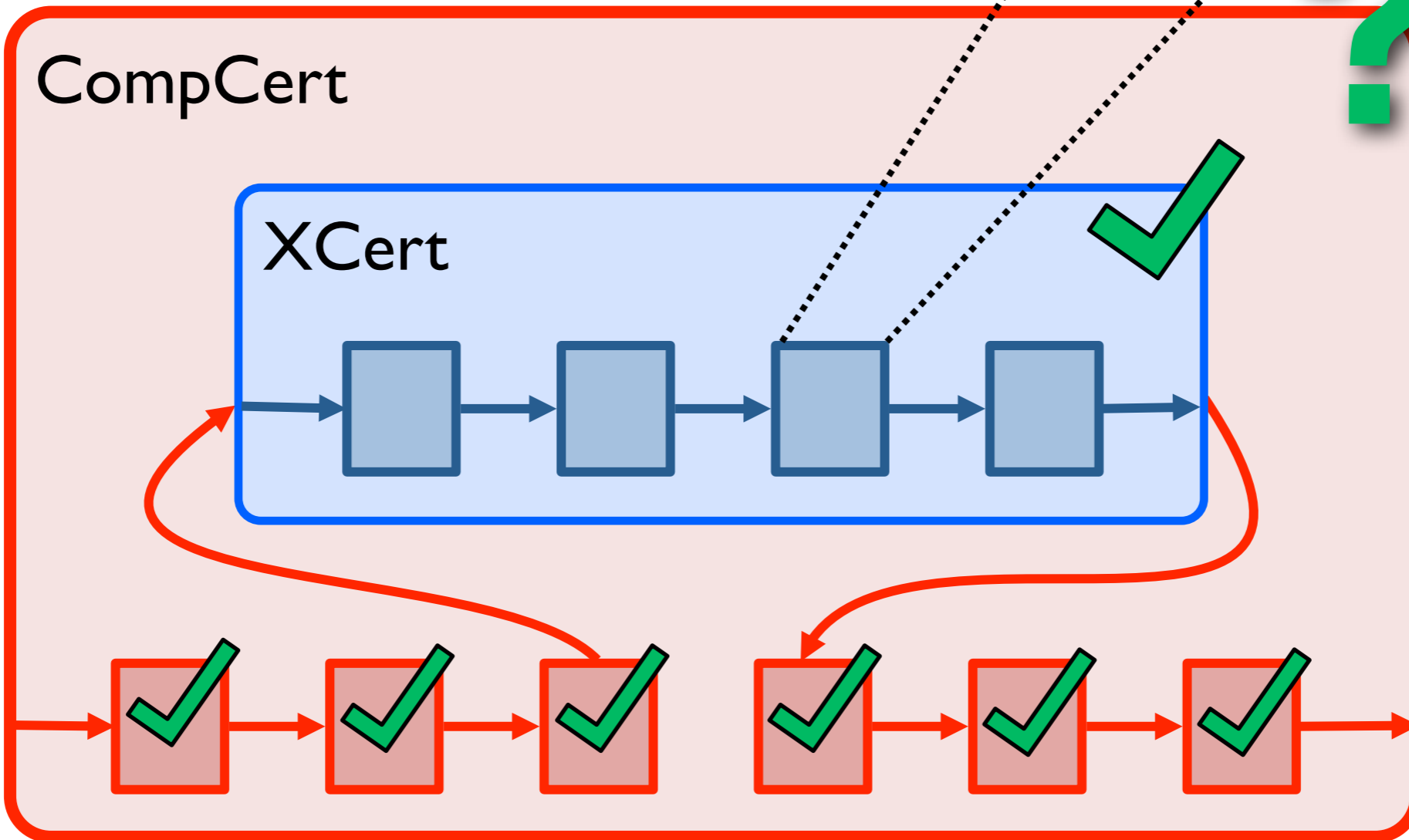
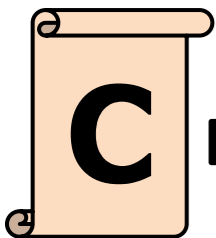
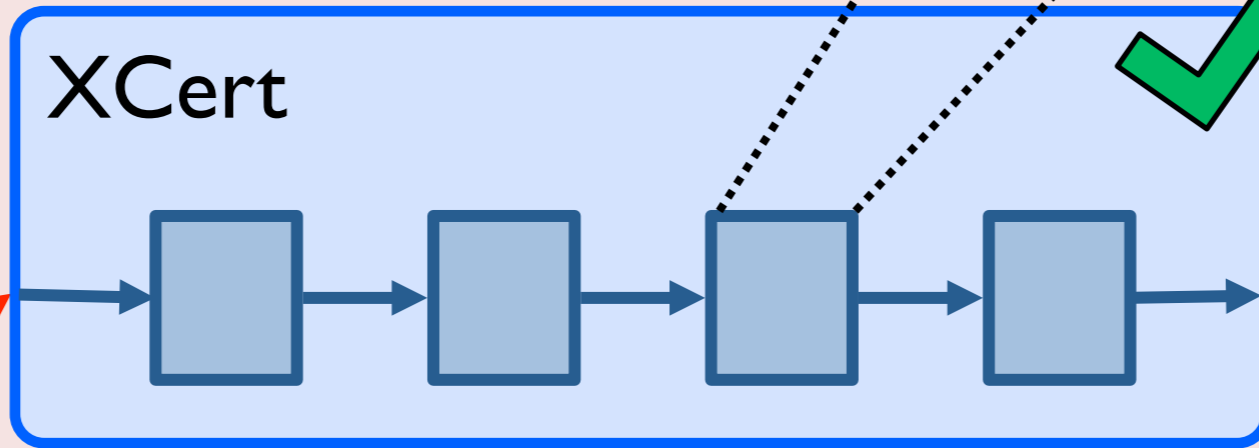


Rewrite

Local Proofs

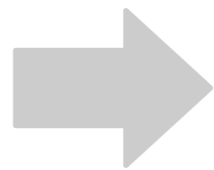
CompCert

XCert

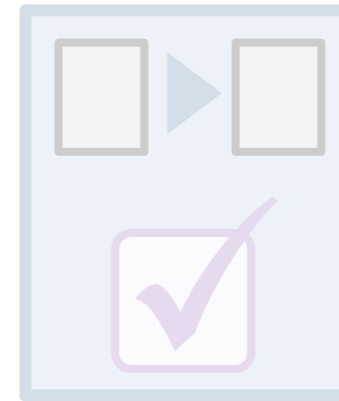
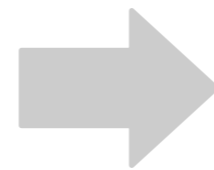


# Verifying Optimizations

Rewrite Rule



**PEC**

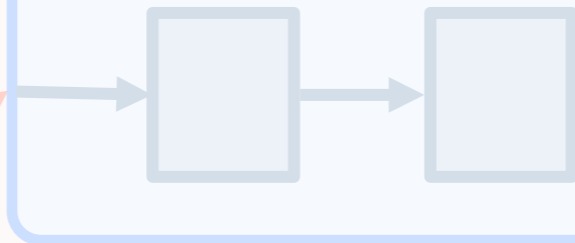


Rewrite

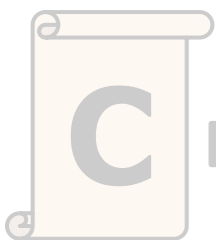
Local Proofs

CompCert

XCert

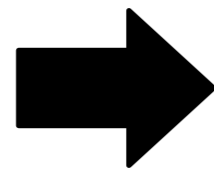
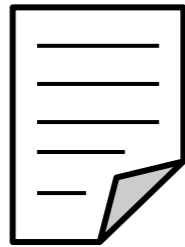


Auto prove complex opts:  
*software pipelining*  
*loop fusion / distribution*  
*loop unswitching*  
...

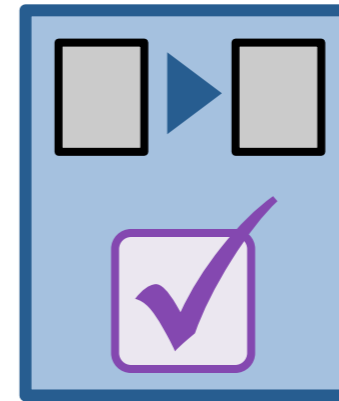
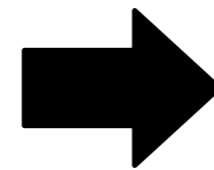


# Verifying Optimizations

Rewrite Rule



**PEC**

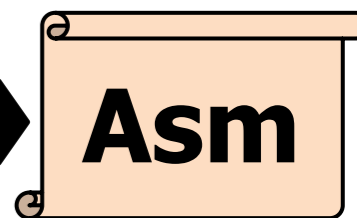
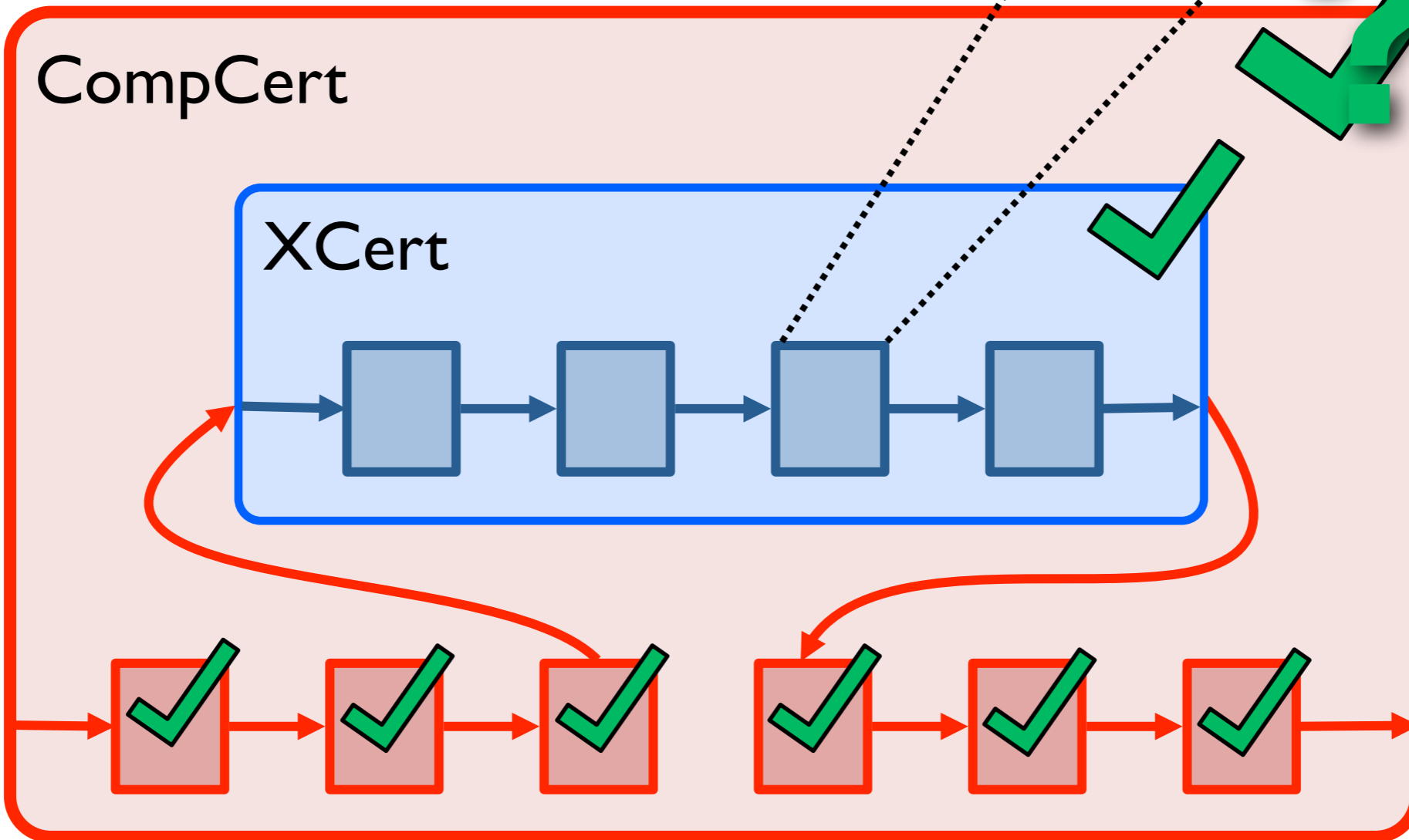
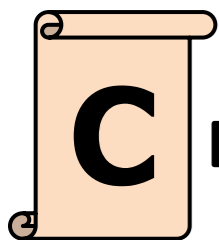
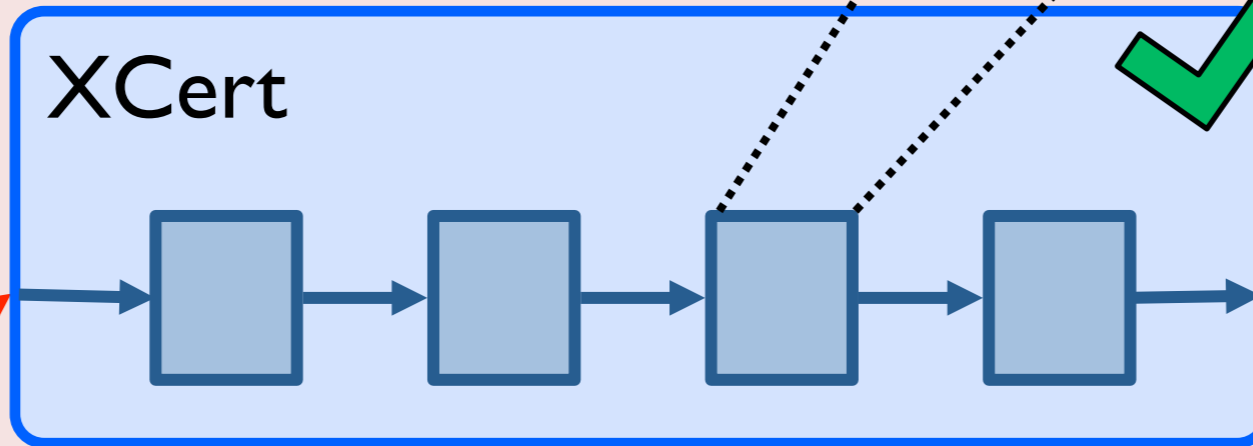


**Rewrite**

**Local Proofs**

CompCert

XCert



# Future Work

## Generating and evaluating specs

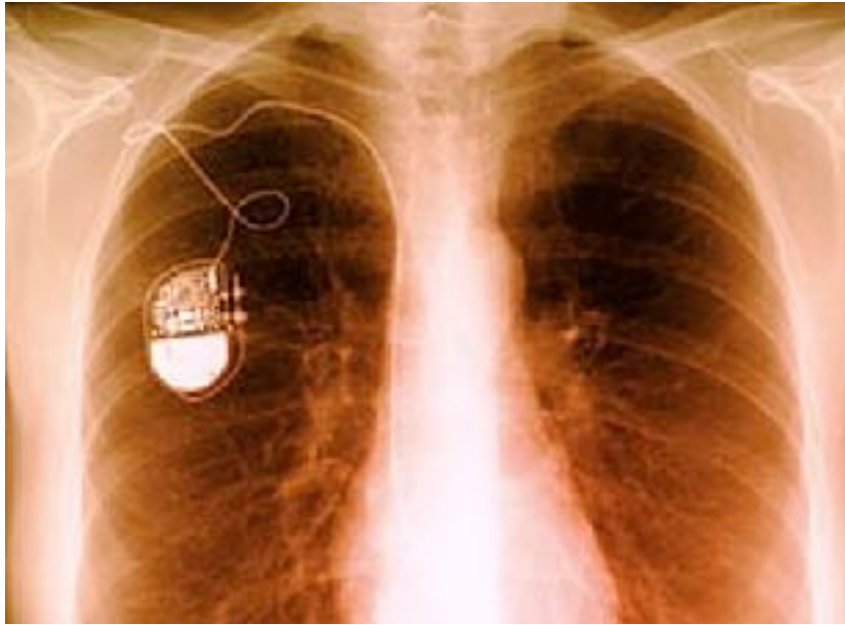
*techniques to ensure spec matches intuition*

*Even perfect program verification can only establish that a program meets its specification... Much of the essence of building a program is in fact the debugging of the specification.*

**Frederick P. Brooks, Jr.**  
*No Silver Bullet*



# Software Infrastructure







# Quark Usability

! google.com | maps.google.com | amazon.com | facebook.com |

+Quark Search Images Maps Play YouTube News Gmail Documents Calendar More ▾

Quark Tester  + Share 

 A faster way to browse the web  
[Install Google Chrome](#)

[Google Search](#) [I'm Feeling Lucky](#)

And we have lift off! Celebrate 50 years of the Kennedy Space Center with [Google Maps](#)

[Change background image](#) [Advertising Programs](#) [Business Solutions](#) [Privacy & Terms](#) [+Google](#) [About Google](#)

# Browsers: Critical Infrastructure

# Browsers: Critical Infrastructure



CHASE



ING



E\*TRADE

# Browsers: Critical Infrastructure



# Browsers: Critical Infrastructure



CHASE



ING



EXTRA



Gmail



Hotmail



# Browsers: Critical Infrastructure



CHASE



ING



E\*TRADE



Gmail



Hotmail



# Browsers: Critical Infrastructure



CHASE



ING



E→TRADE



Gmail



Conference Submissions

