

Browser Security Guarantees through Formal Shim Verification

A dramatic illustration of a dark, horned demon-like creature breathing fire, with a figure hanging from a rope in the background.

Dongseok Jang

Zachary Tatlock
UC San Diego

Sorin Lerner

Browsers: Critical Infrastructure

Ubiquitous:

many platforms, sensitive apps

Vulnerable:

Pwn2Own, just a click to exploit

Reactive Defenses:

many ad hoc, bug triage, regressions

Fully Formal Verification

Code in language that eases reasoning

Develop correctness proof in synch

Fully formal, *machine checkable* proof

Fully Formal Verification

Success story: CompCert C compiler

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

[Yang et al. PLDI 11]

OS (seL4), RDBMS & HTTPD (YNot)

realistic implementations guaranteed bug free

Fully Formal Verification

The Catch

Throw away all your code

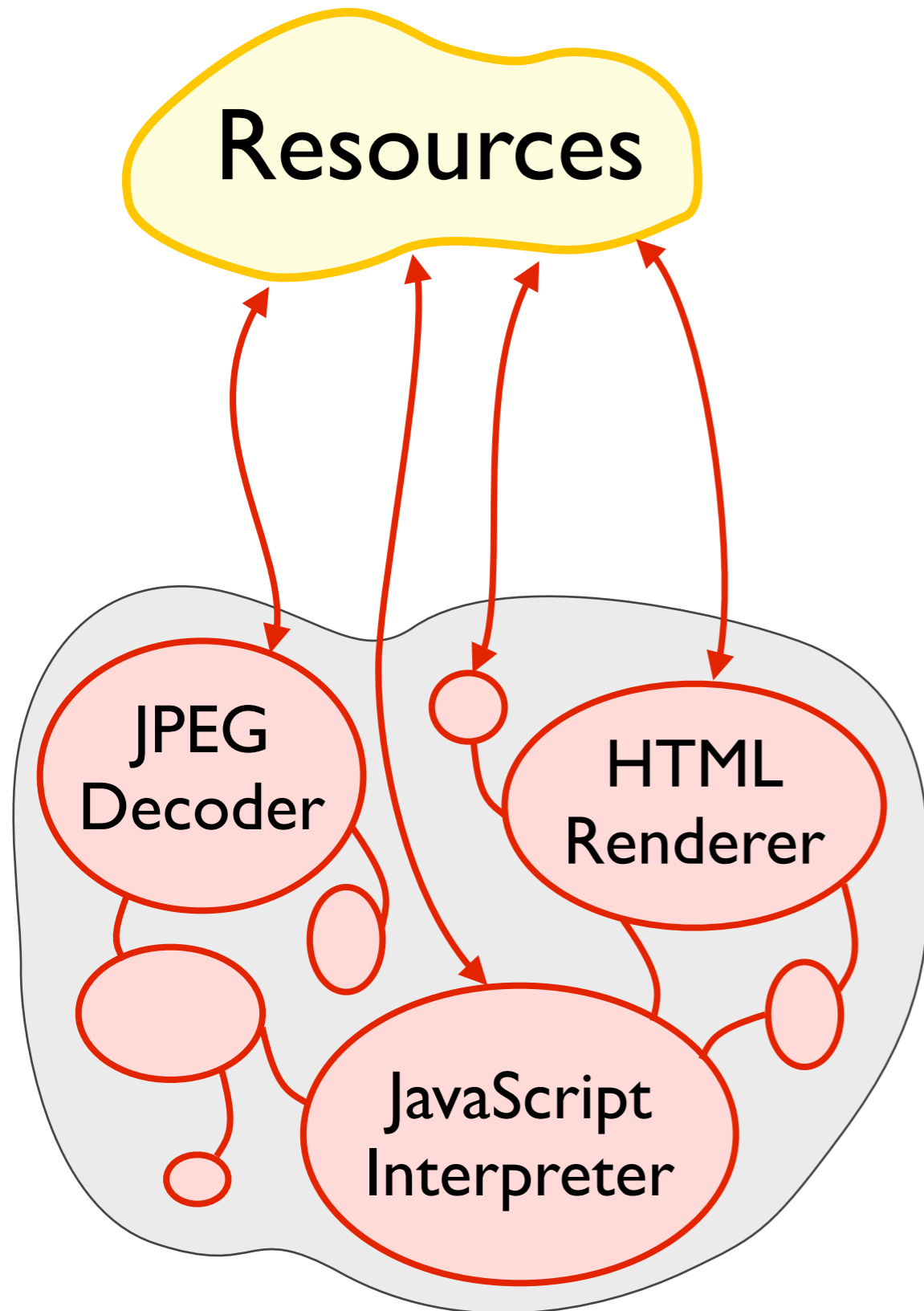
Rewrite in unfamiliar language

Formally specify correctness

Prove every detail correct

Heroic effort

Formally Verify a Browser?!



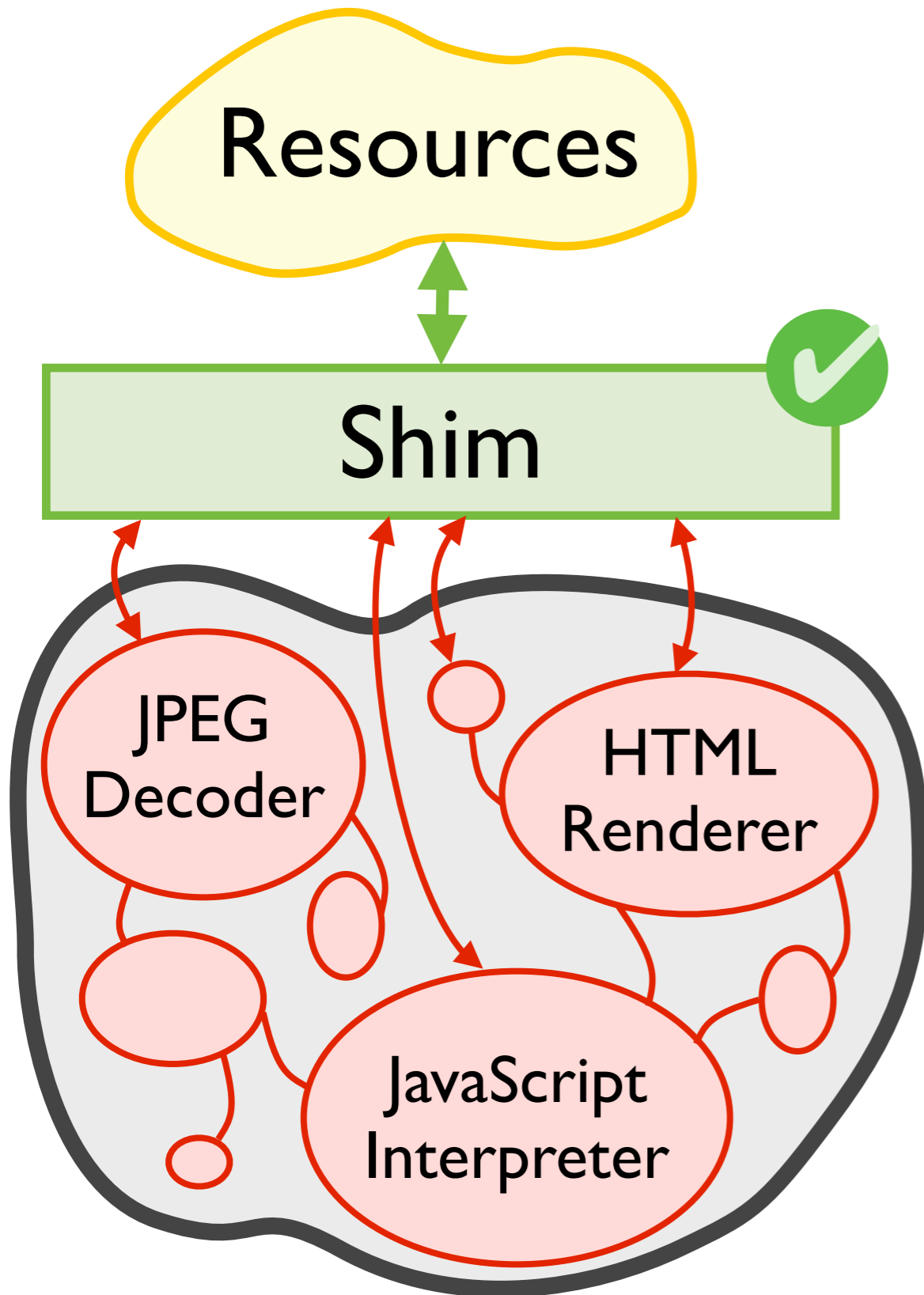
Complex parts

Subtle interactions

Loose access policy

Constant evolution

Formal Shim Verification



Isolate

sandbox untrusted code

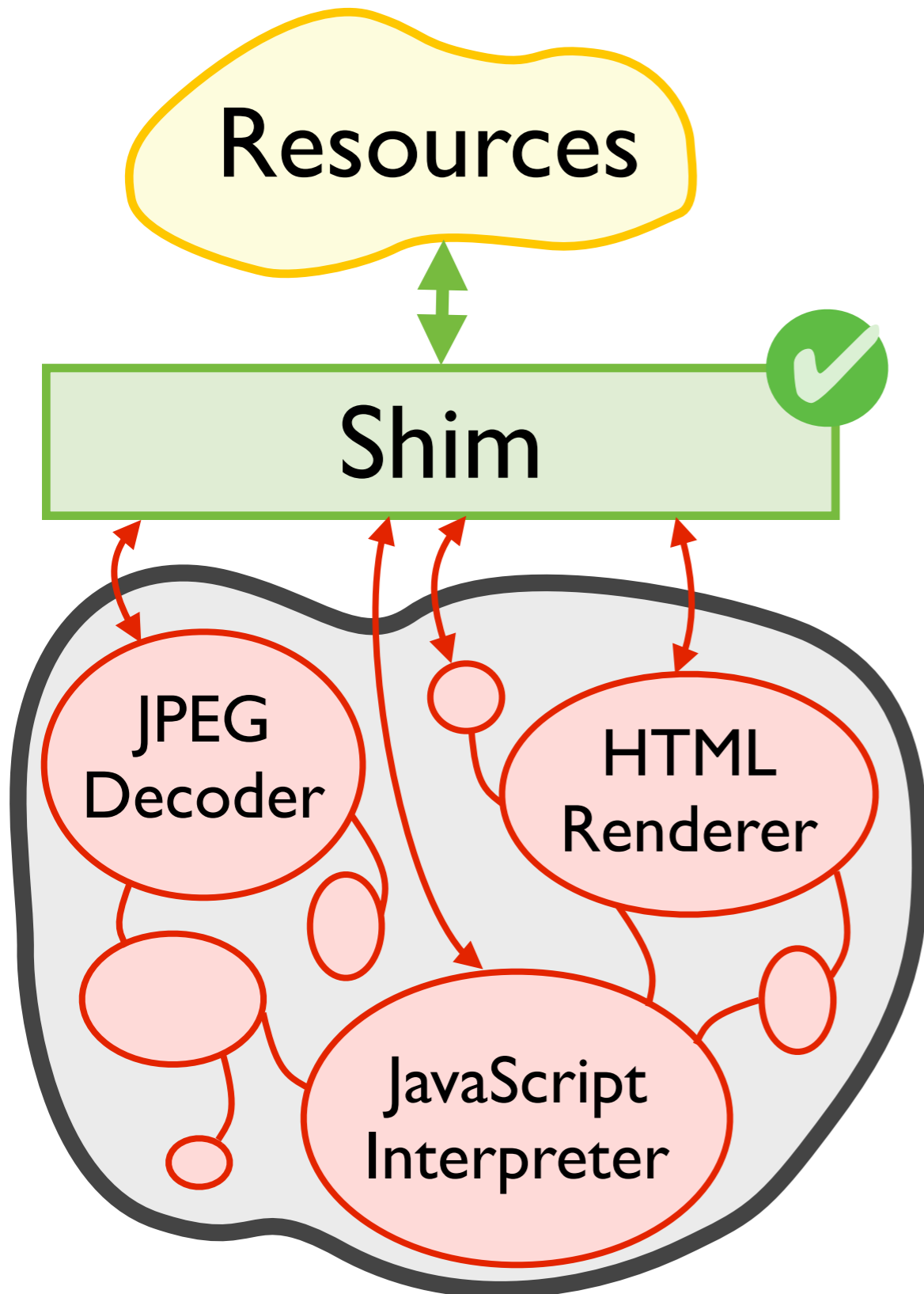
Insert shim

guards resource access

Verify shim

prove security props

Formal Shim Verification



QUARK

formally verified browser

Security Props

1. Tab isolation

2. Cookie integrity

3. Addr bar correctness

Prove code correct

machine checkable proof

Fully Formal Verification

Fully Formal Verification



Code

in language
supporting
reasoning

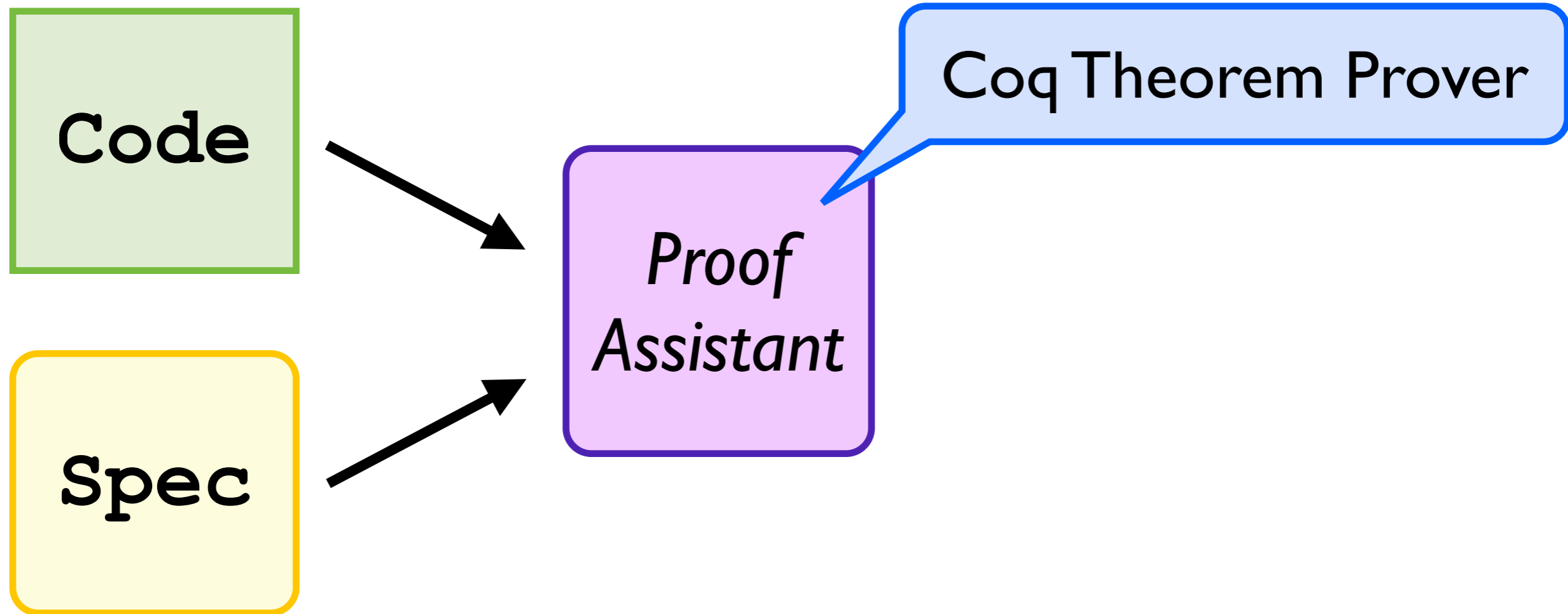
Fully Formal Verification

Code

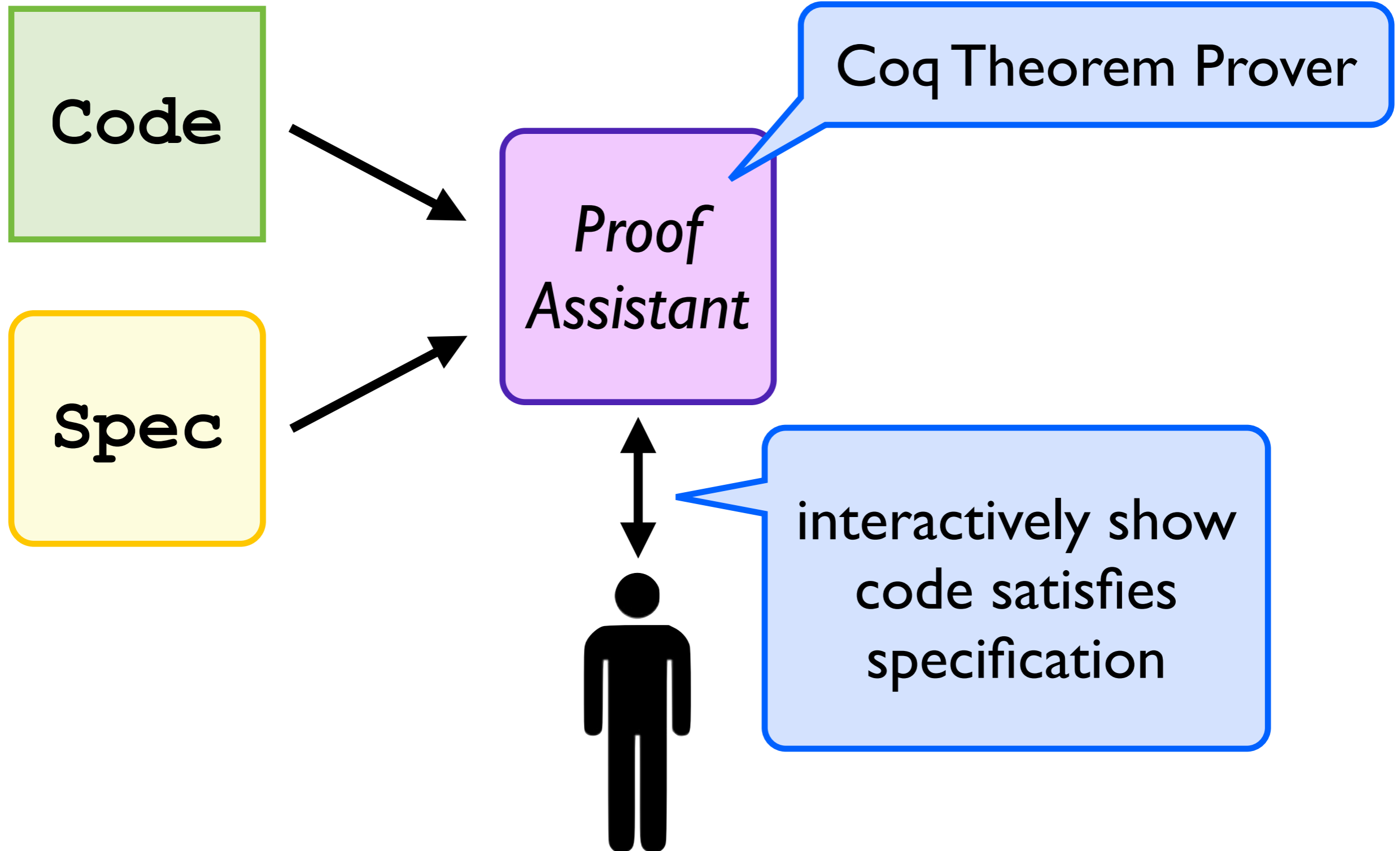
Spec

logical properties
characterizing correctness

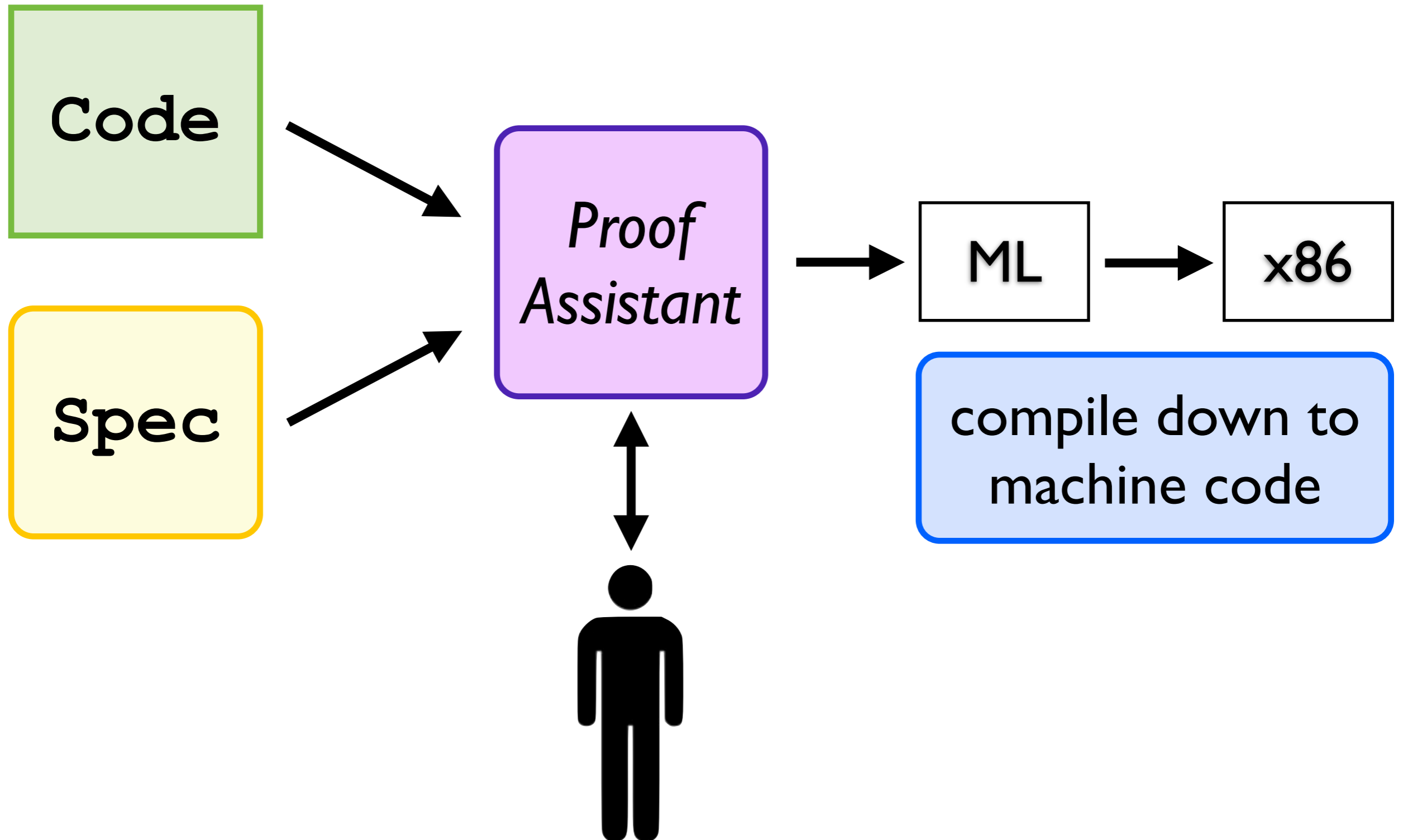
Fully Formal Verification



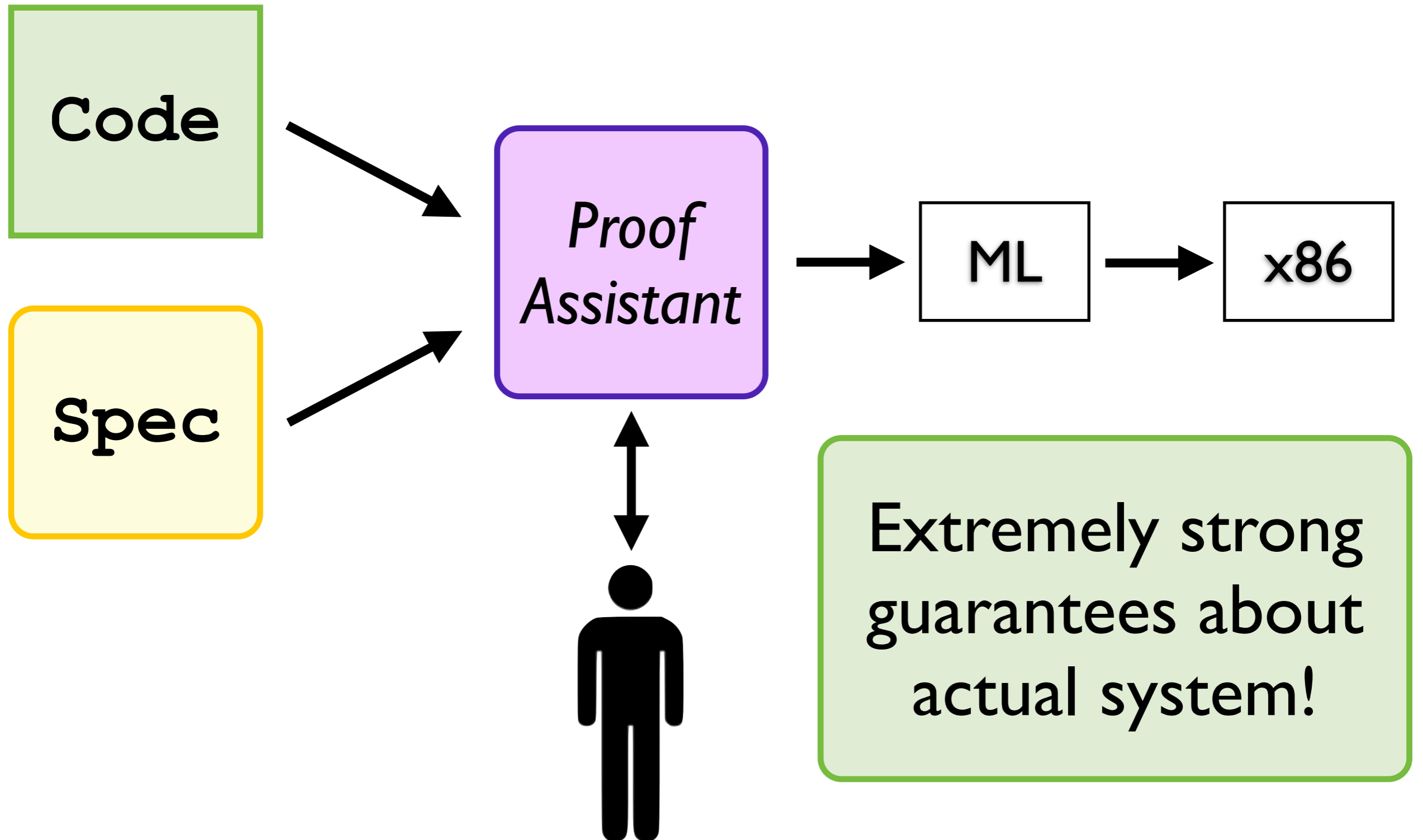
Fully Formal Verification



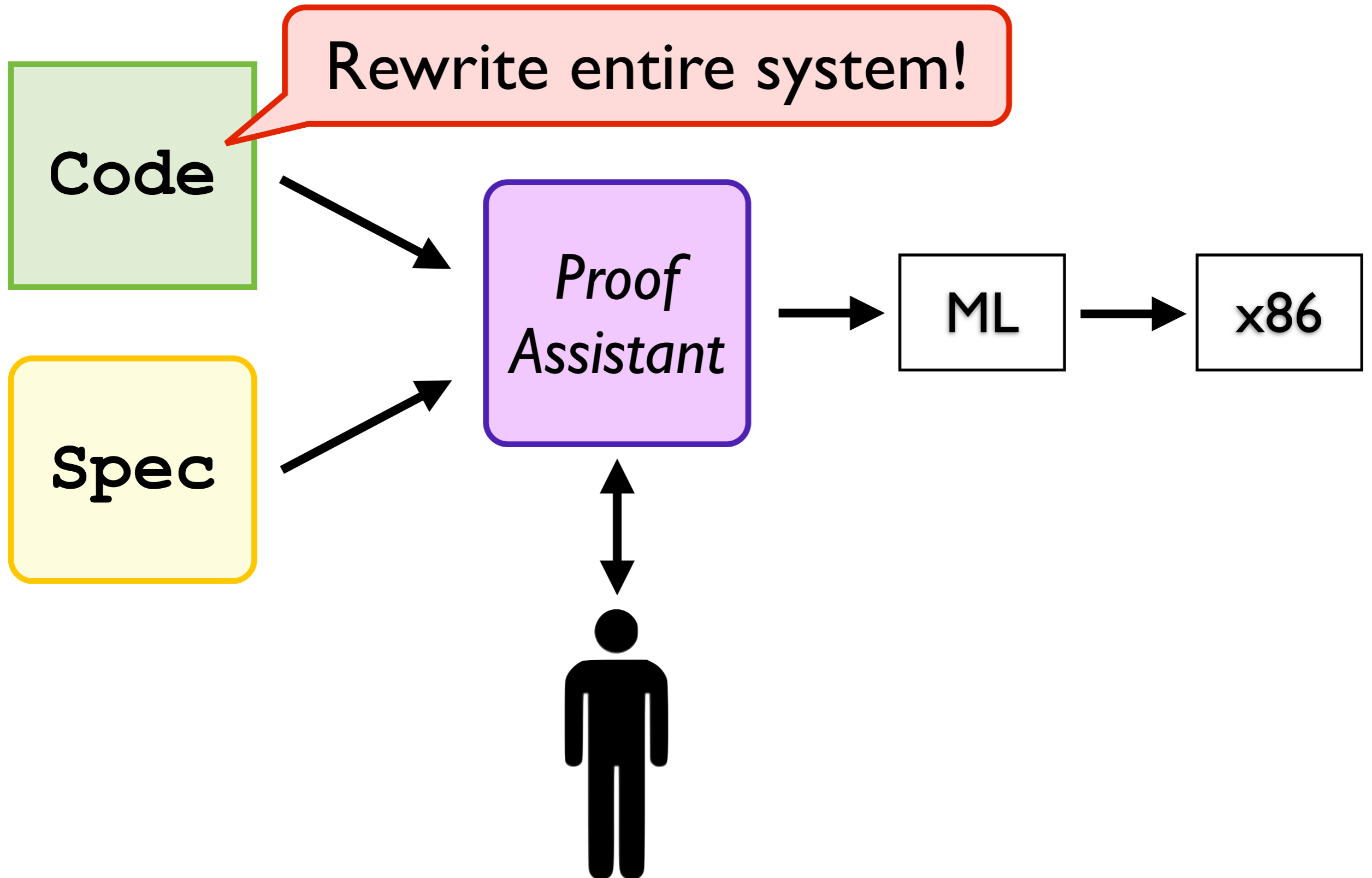
Fully Formal Verification



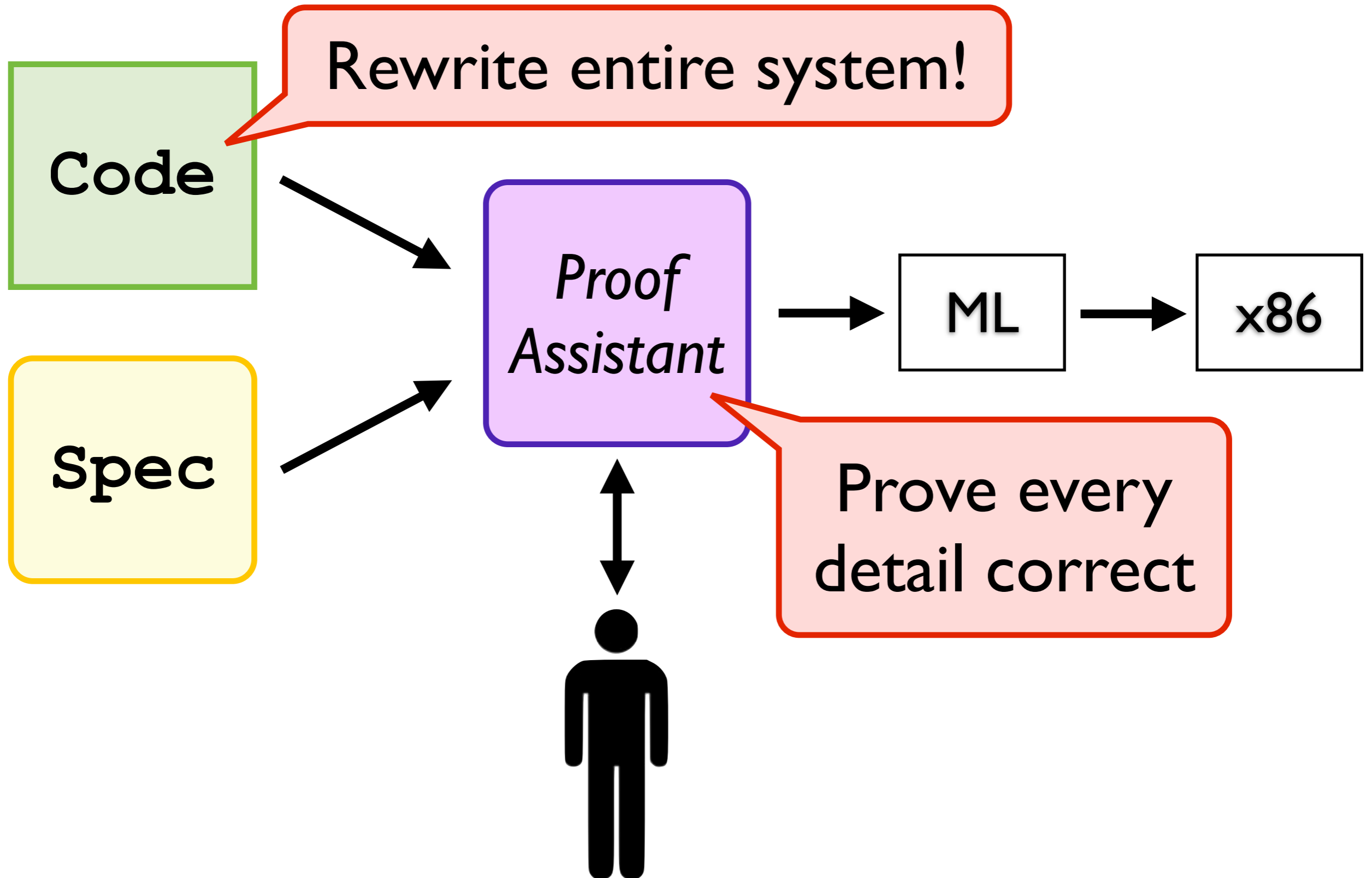
Fully Formal Verification



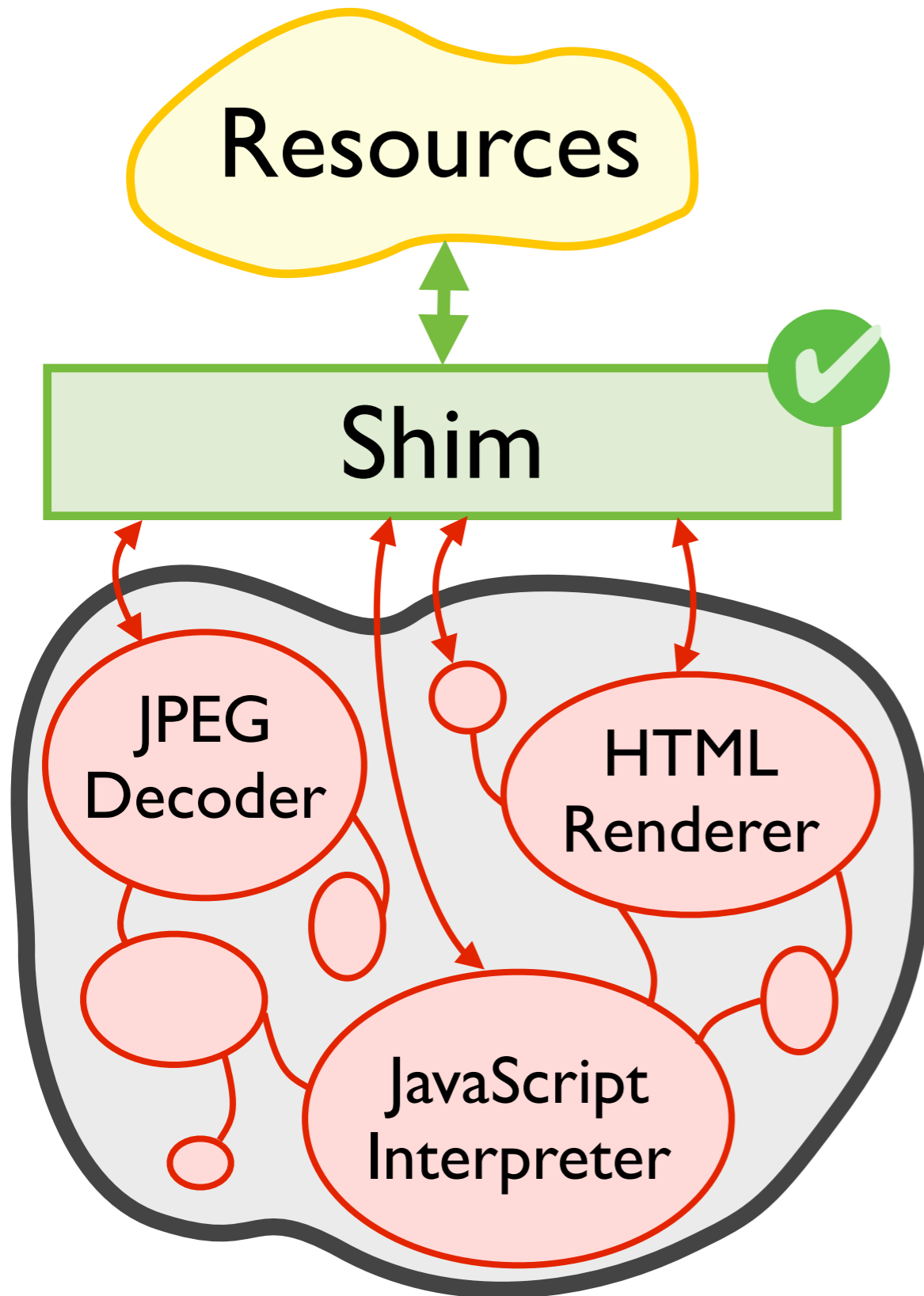
Fully Formal Verification



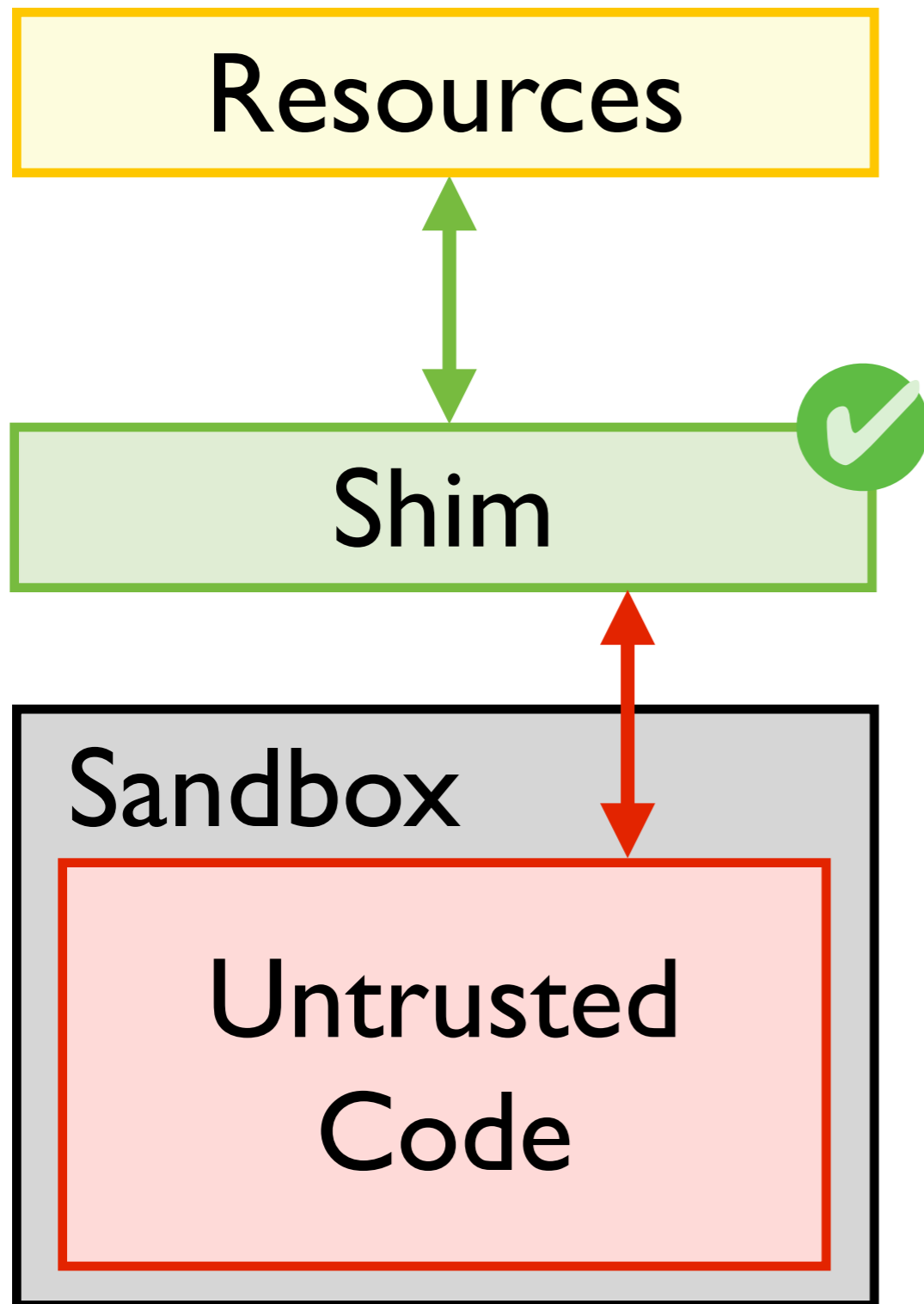
Fully Formal Verification



Formal Shim Verification



Formal Shim Verification



Adapt to sandbox
request access via shim

Write shim
design effective interface

Formally verify shim
ensure accesses secure

Formal Shim Verification

Adapt to sandbox

Key Insight

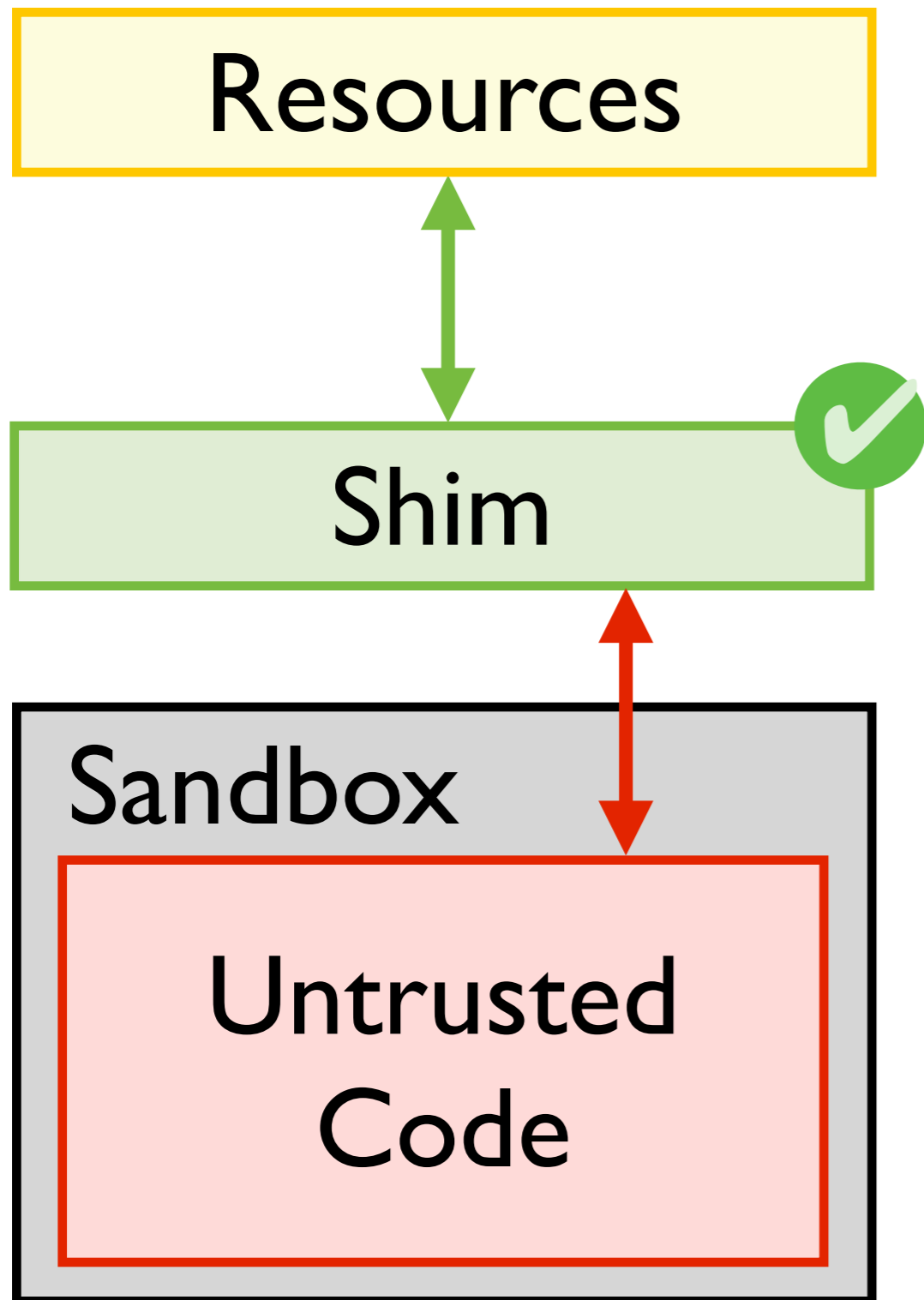
Guarantee sec props for entire system

Only reason about small shim

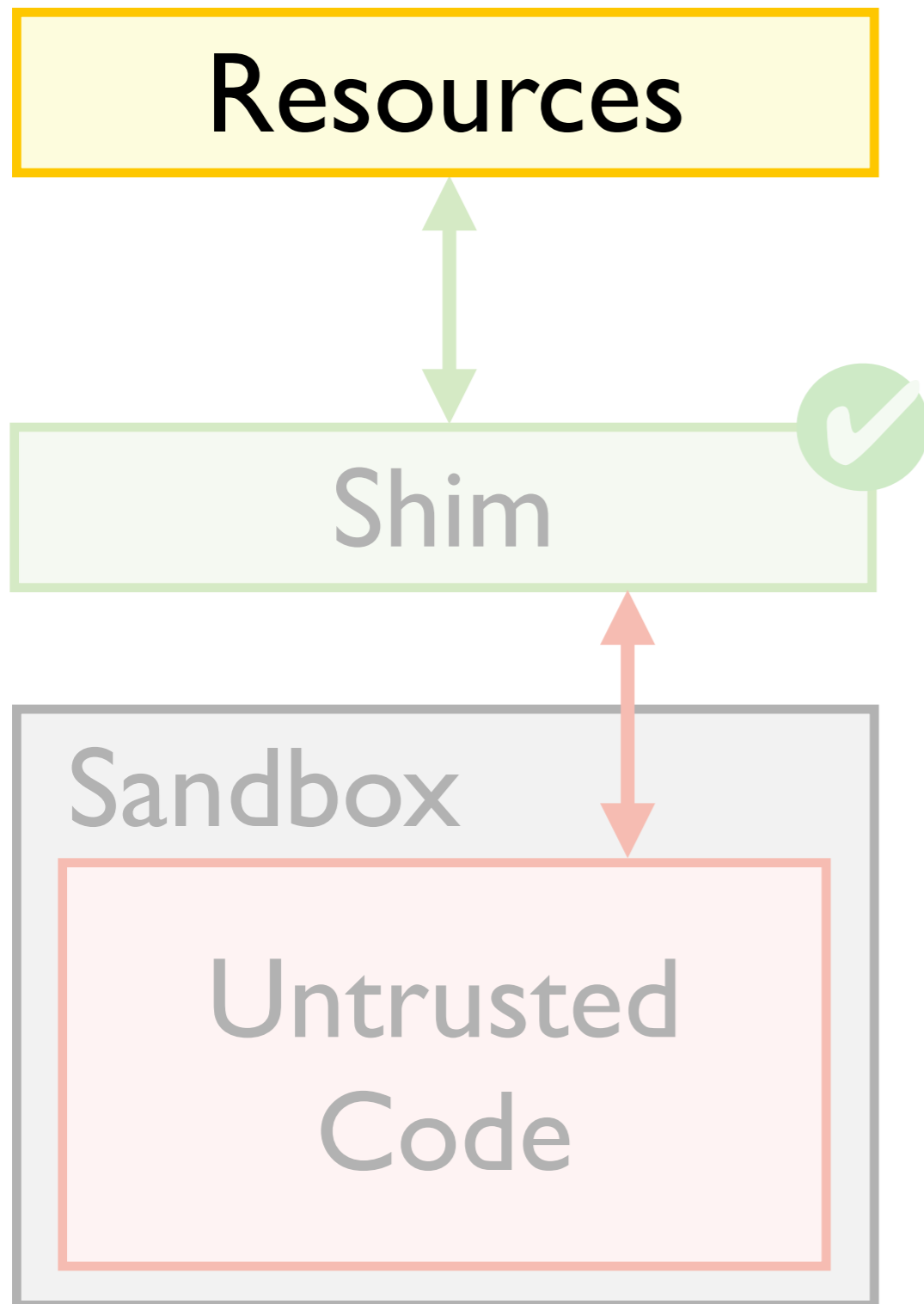
Radically ease verification burden

Prove *actual code* correct

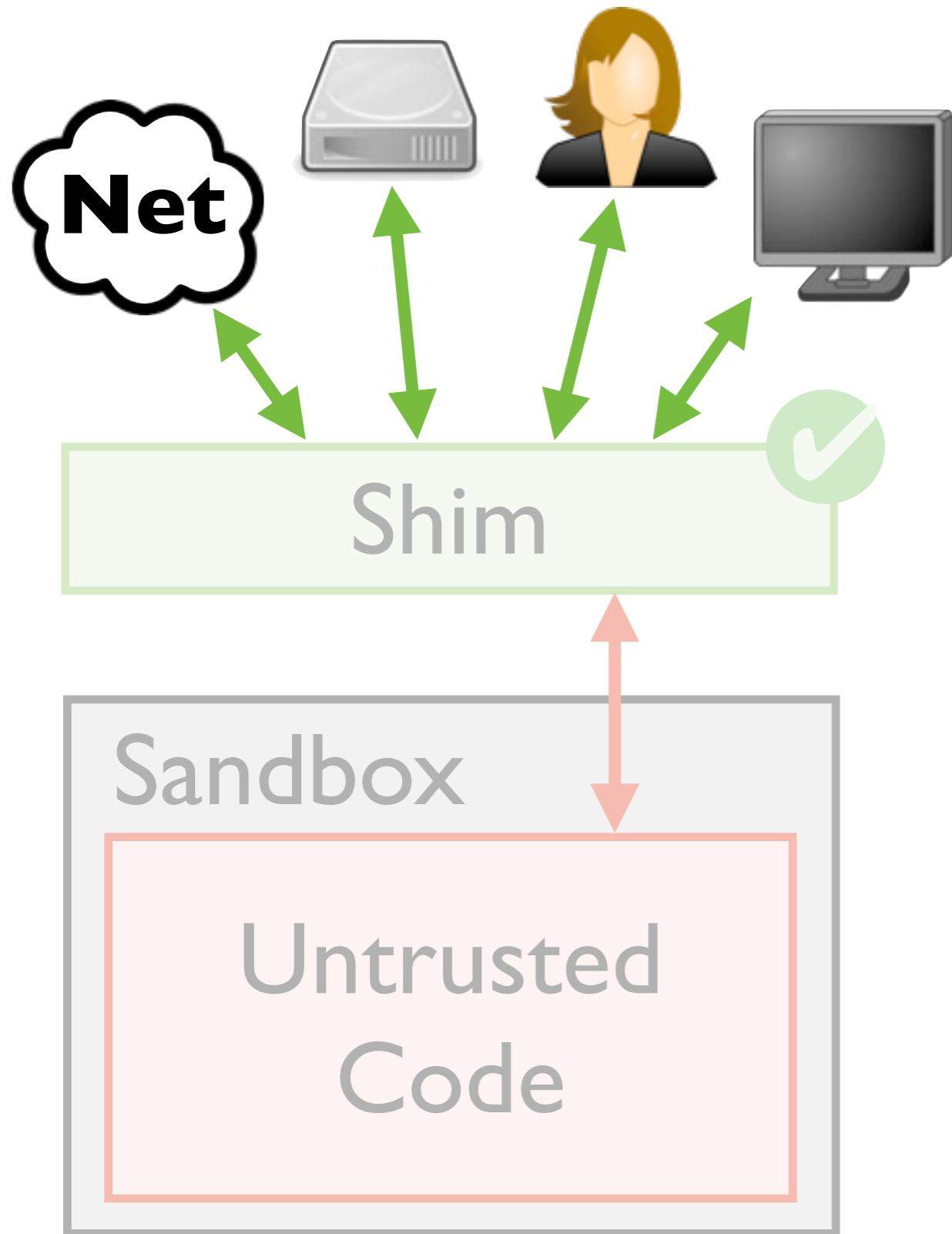
Quark: Verified Browser



Quark: Verified Browser



Quark: Verified Browser



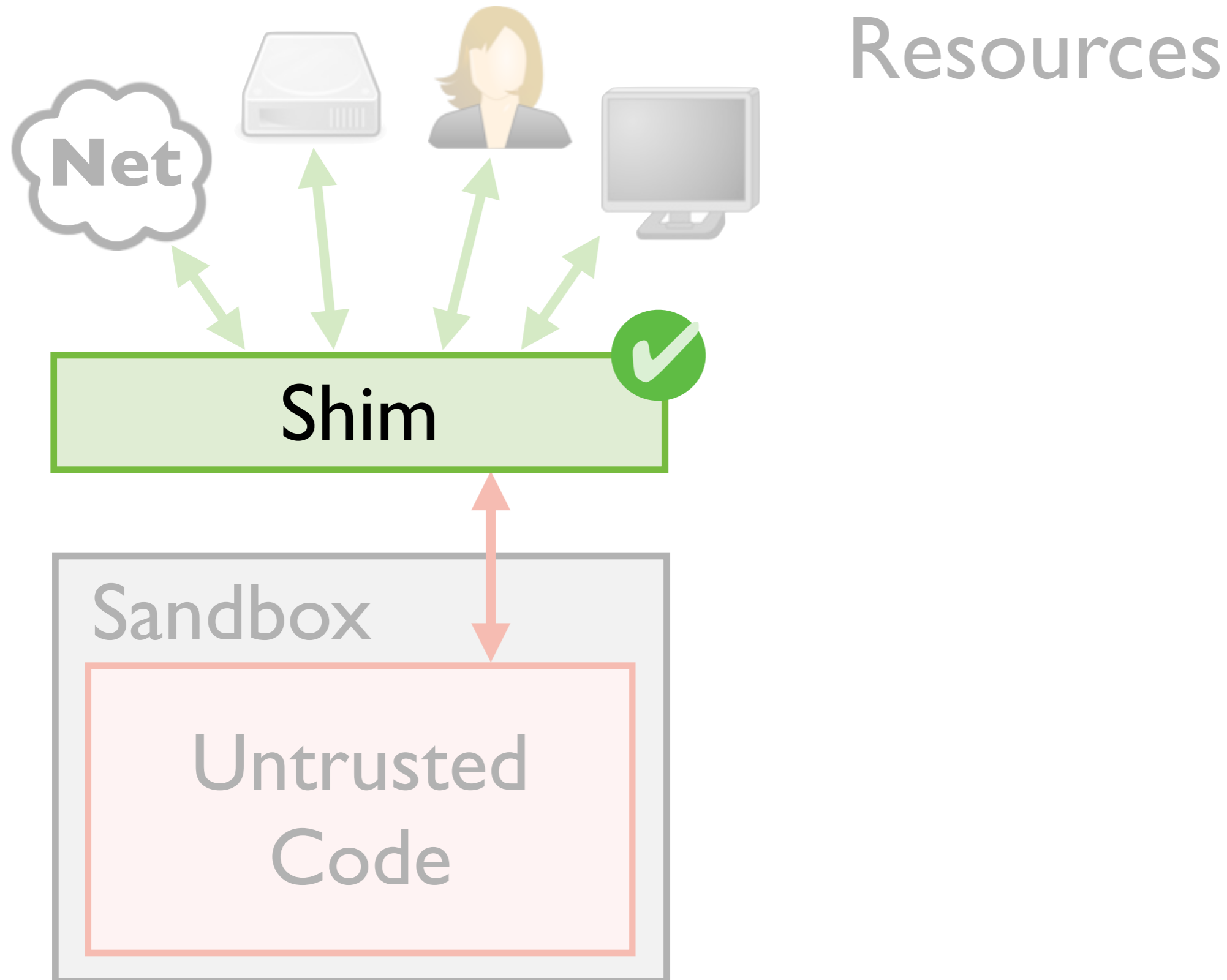
Resources

network

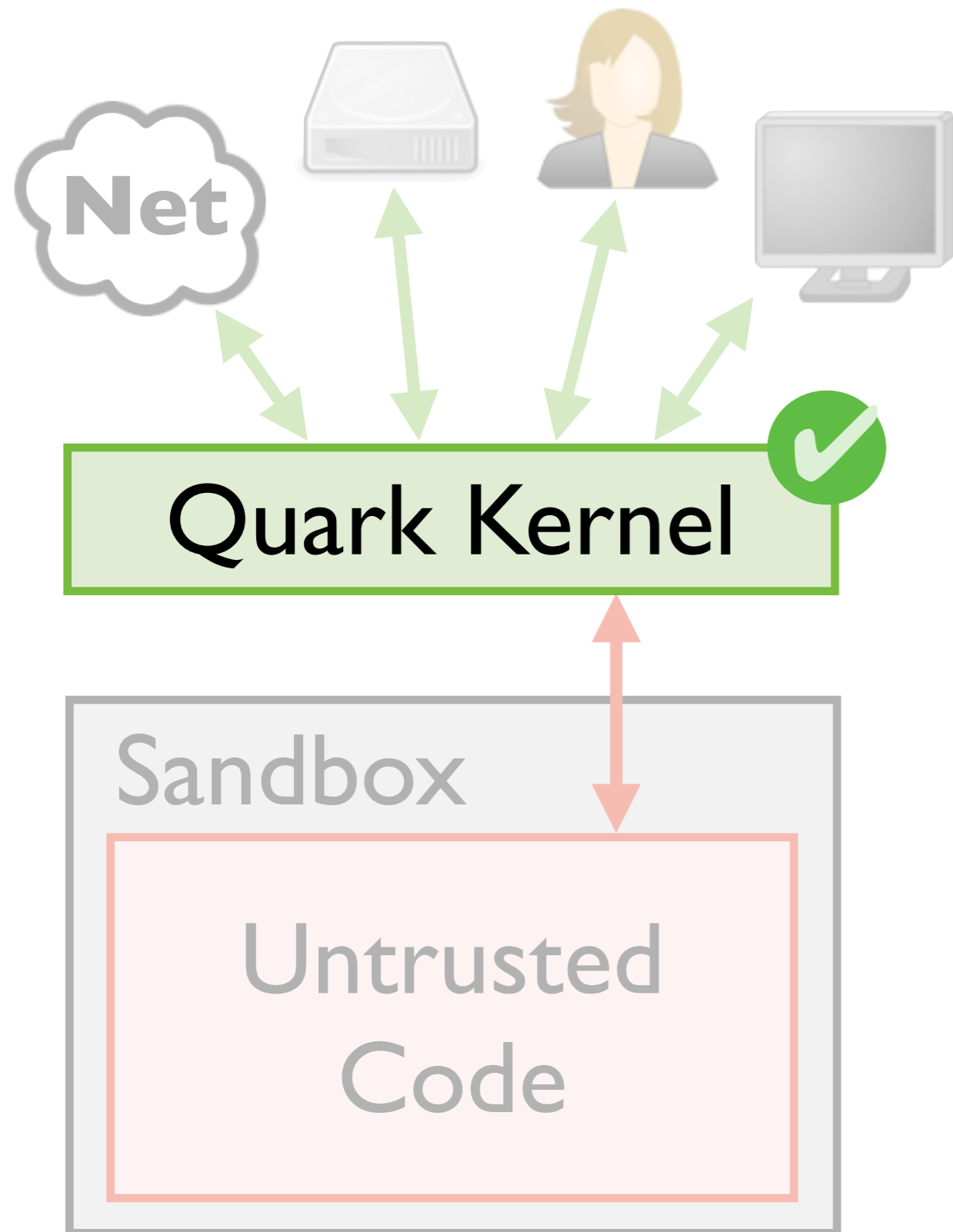
persistent storage

user interface

Quark: Verified Browser



Quark: Verified Browser

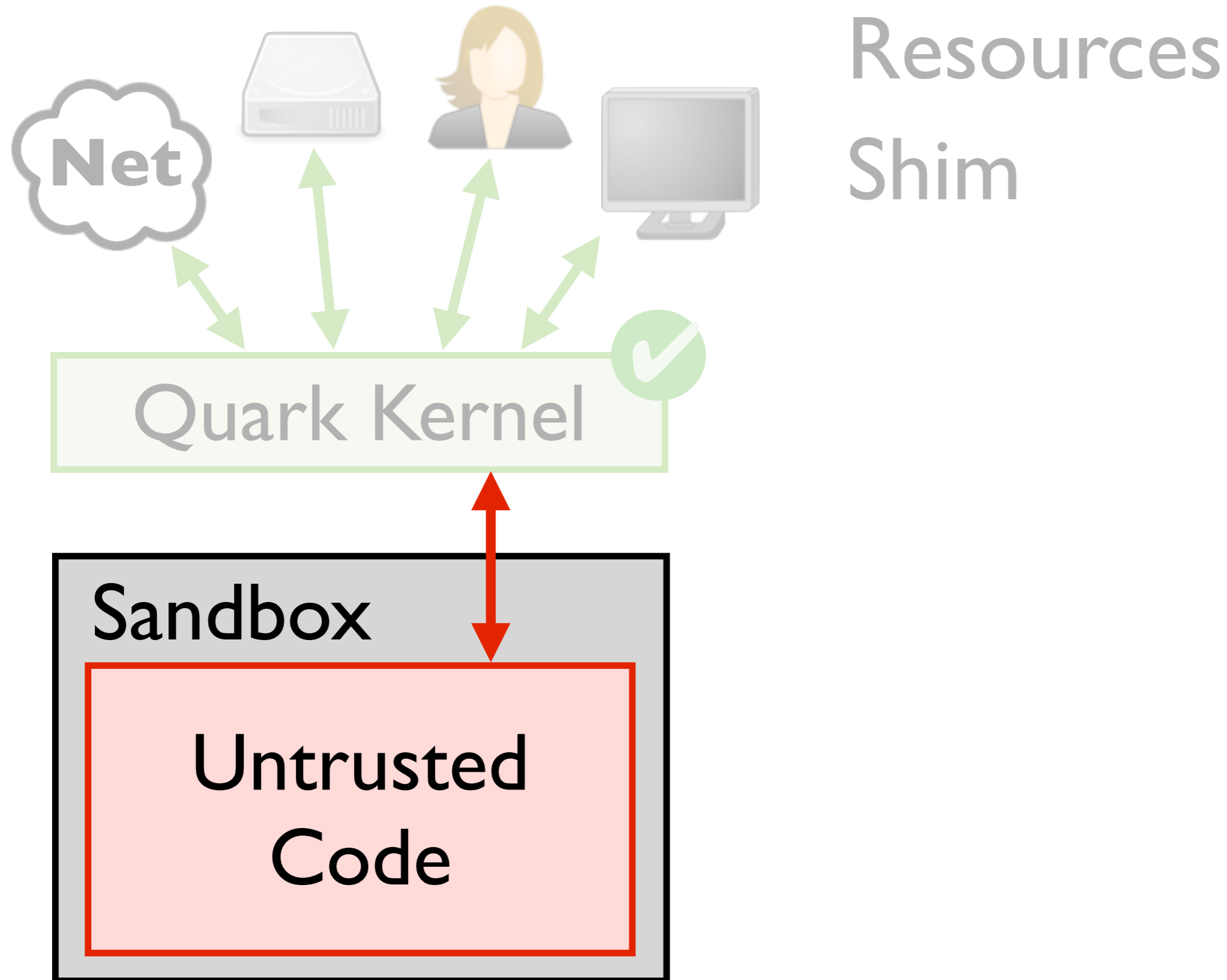


Resources

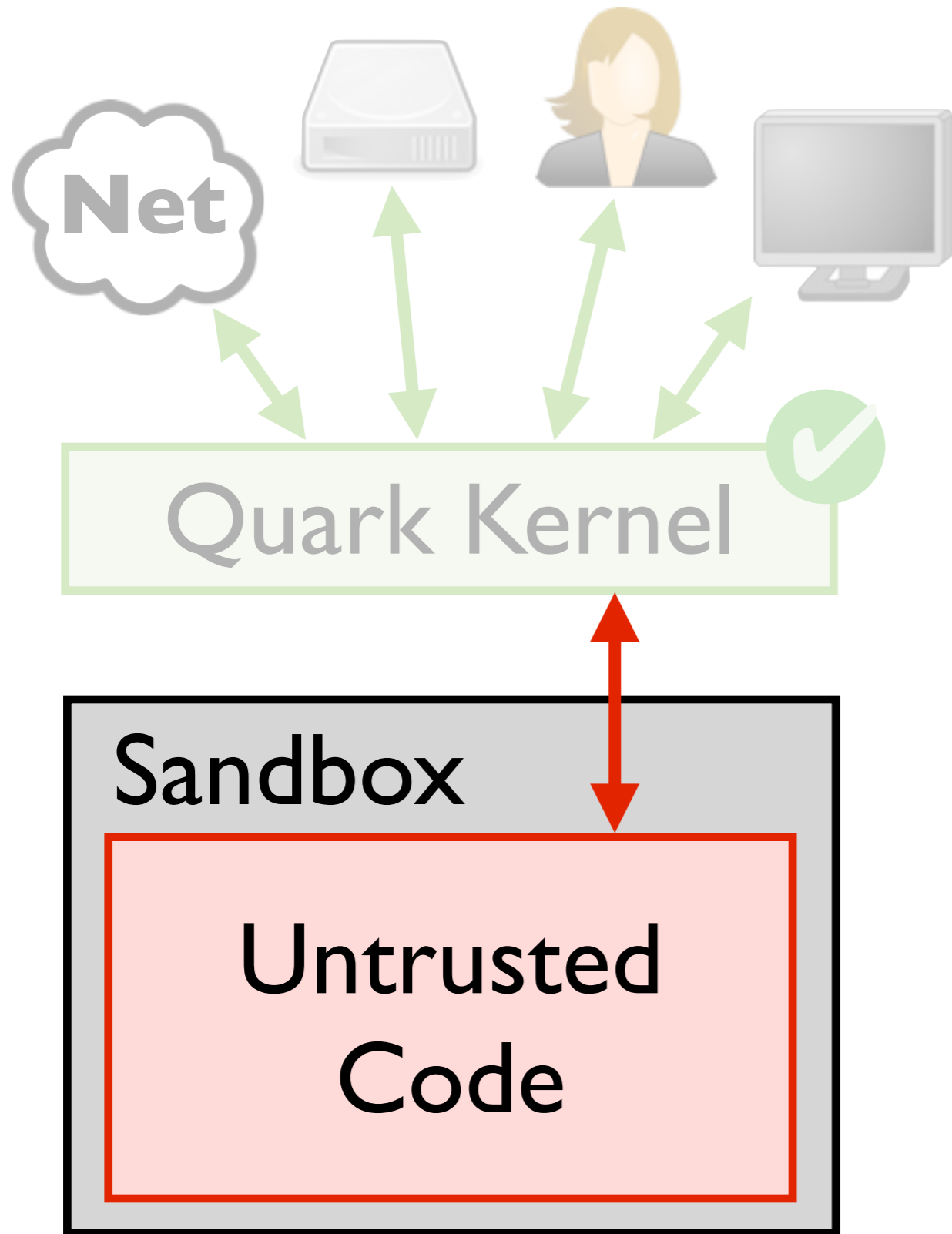
Shim

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

Quark: Verified Browser



Quark: Verified Browser



Resources

Shim

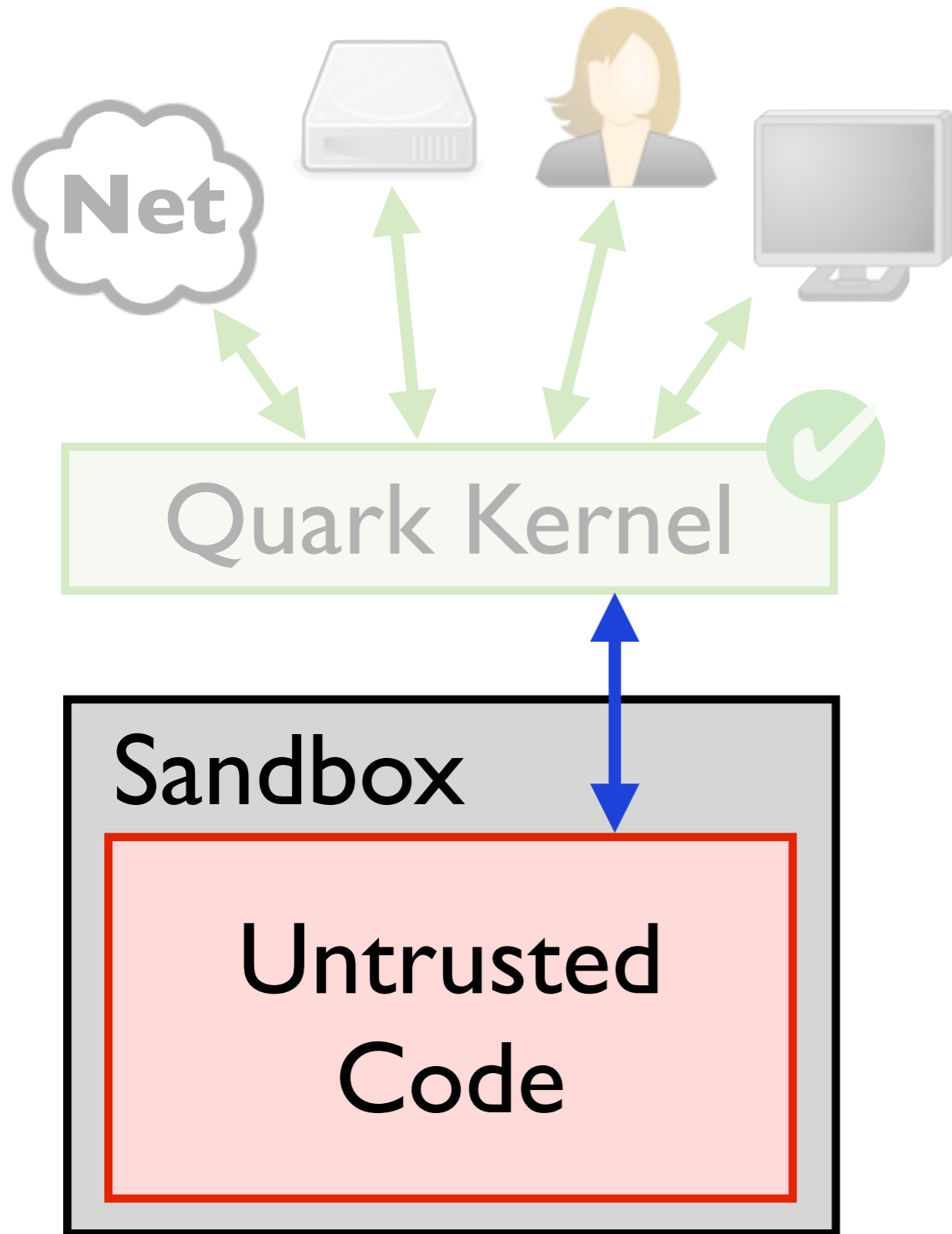
Untrusted Code

browser components

run as separate procs

strictly sandboxed

Quark: Verified Browser



Resources

Shim

Untrusted Code

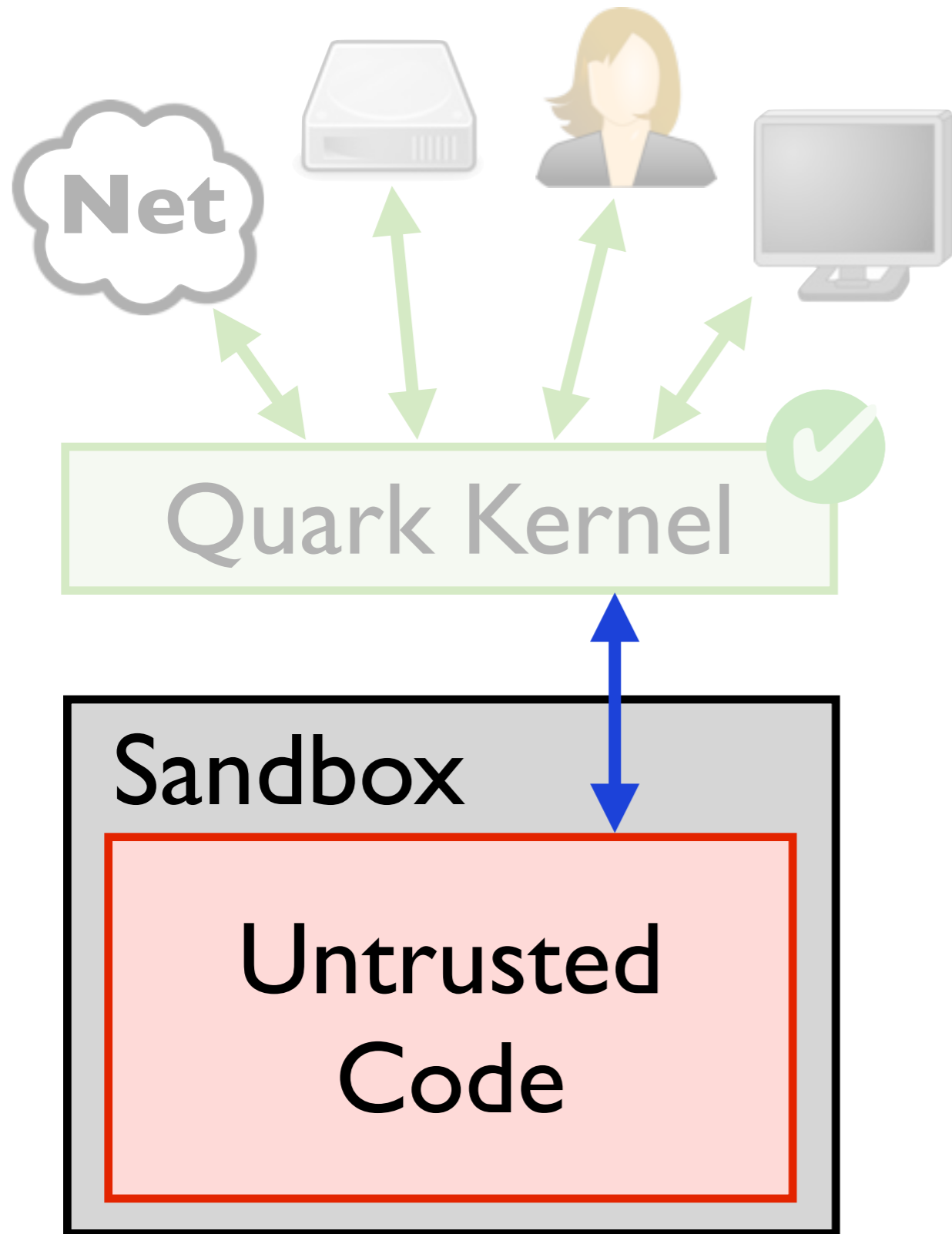
browser components

run as separate procs

strictly sandboxed

*talk to kernel over **pipe***

Quark: Verified Browser



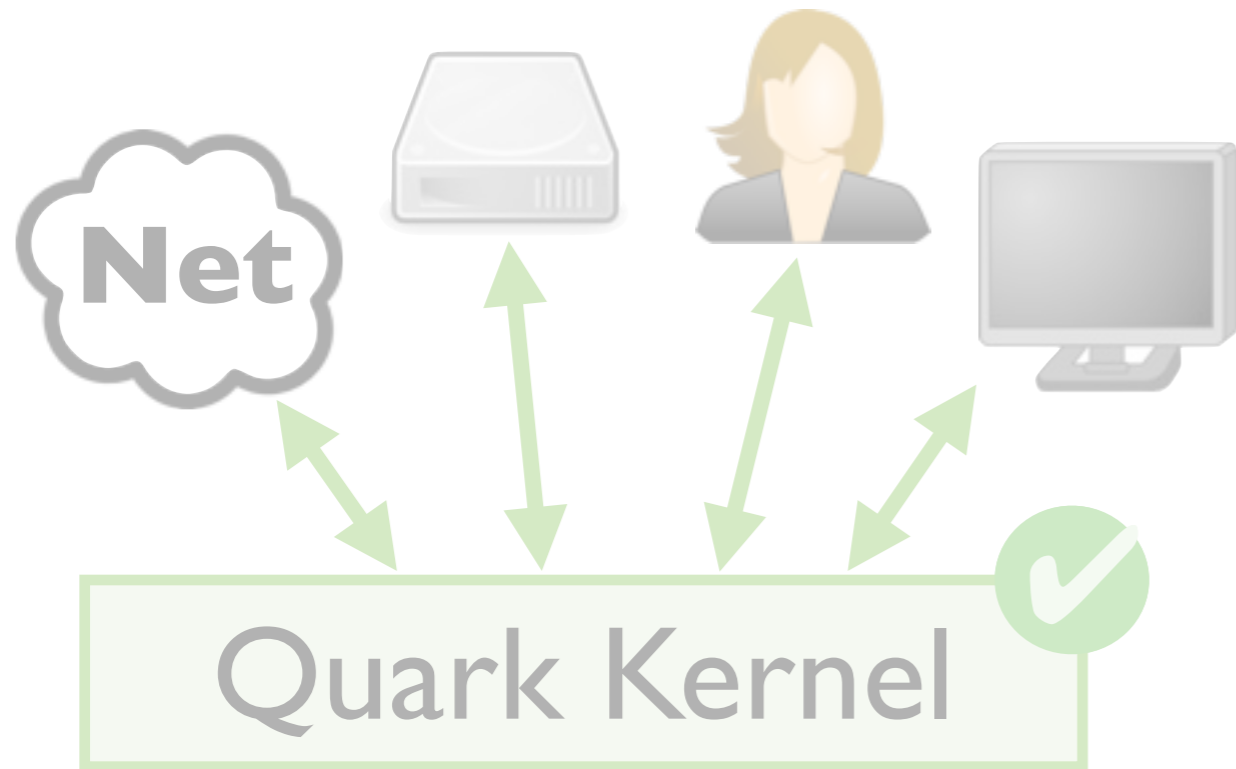
Resources

Shim

Untrusted Code

two component types

Quark: Verified Browser

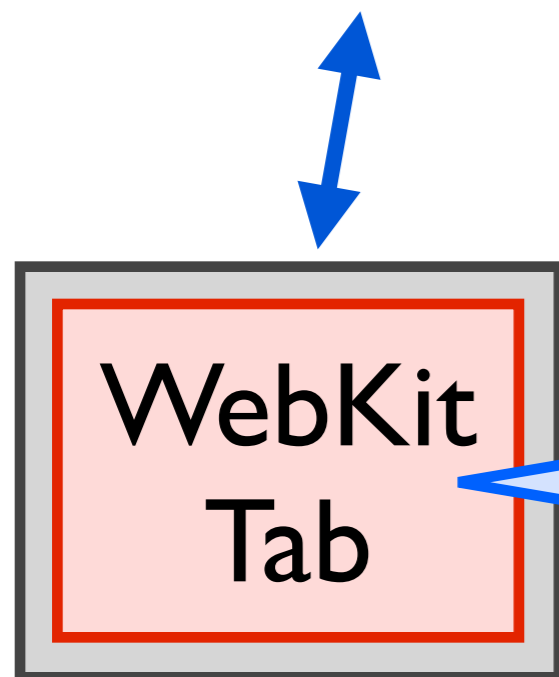


Resources

Shim

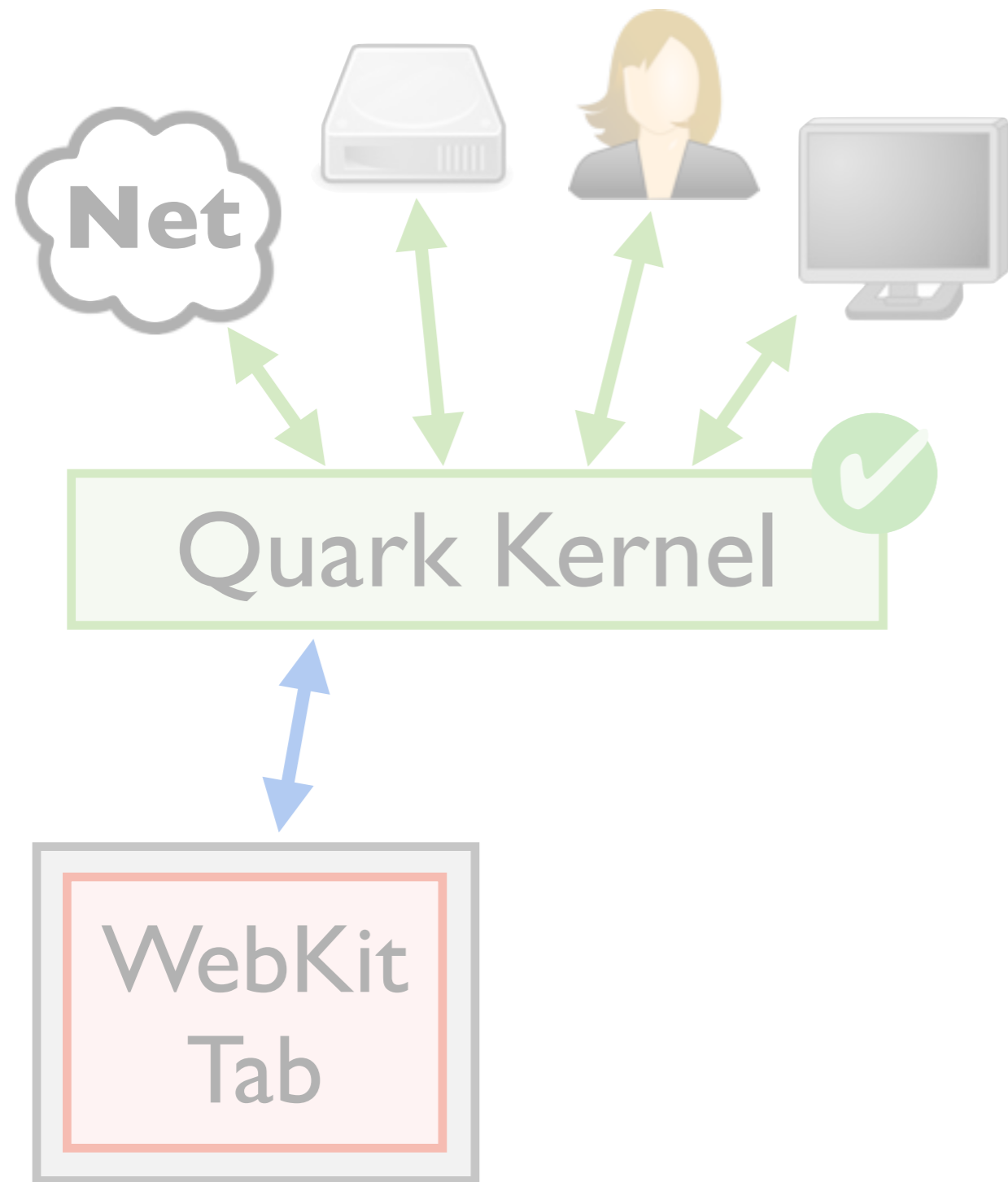
Untrusted Code

two component types



modified WebKit,
intercept accesses

Quark: Verified Browser



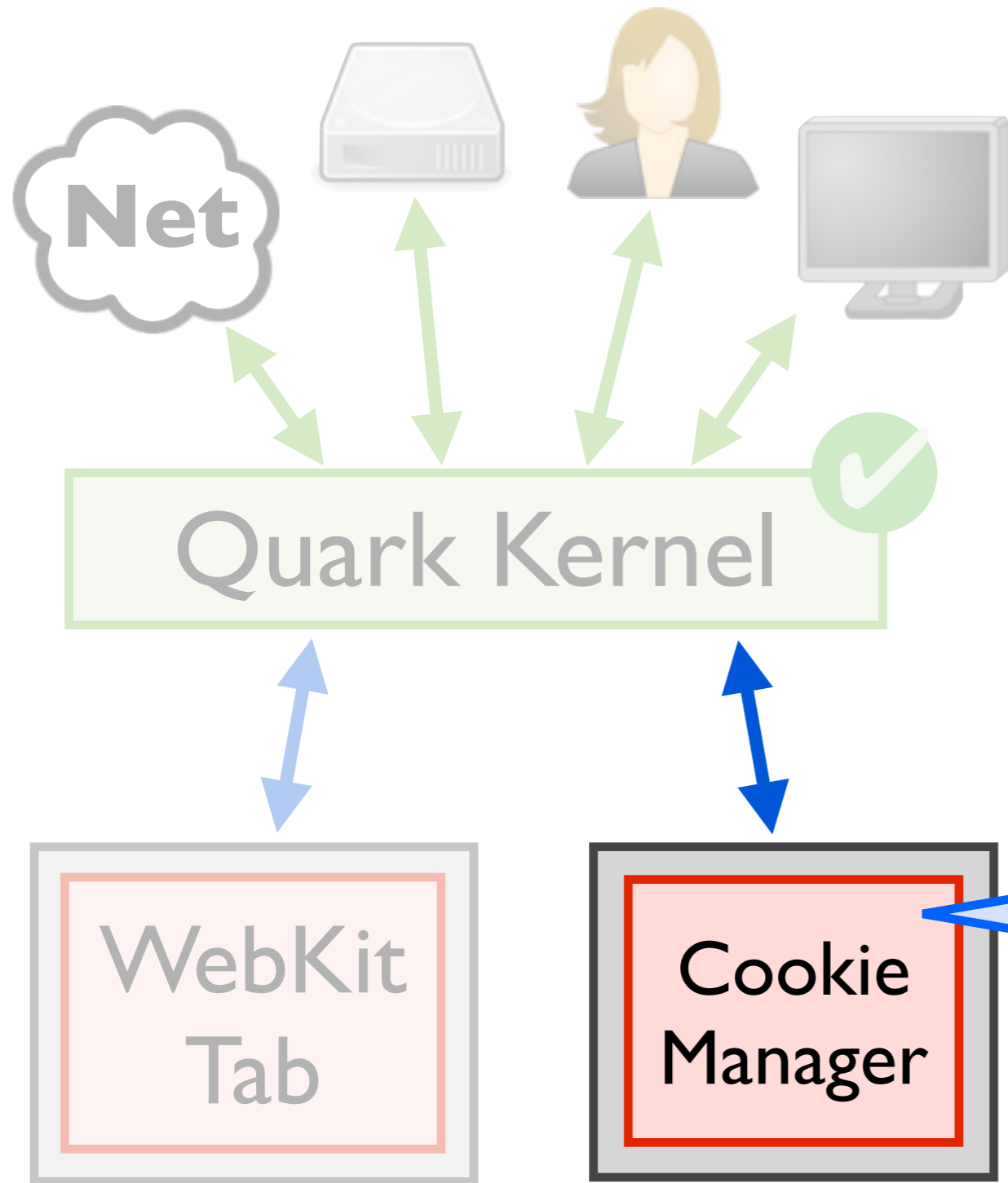
Resources

Shim

Untrusted Code

two component types

Quark: Verified Browser



Resources

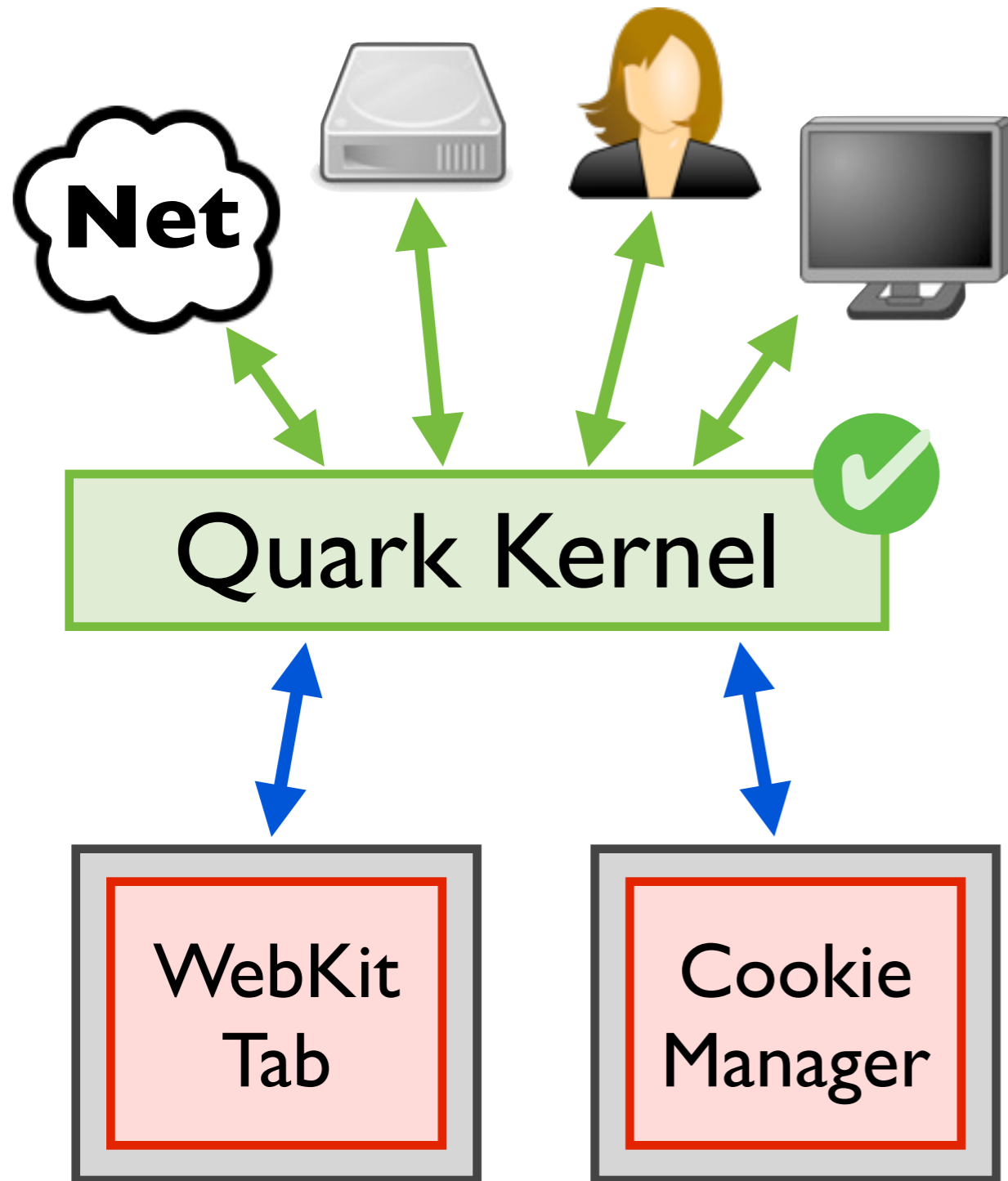
Shim

Untrusted Code

two component types

written in Python,
manages single domain

Quark: Verified Browser



Resources

Shim

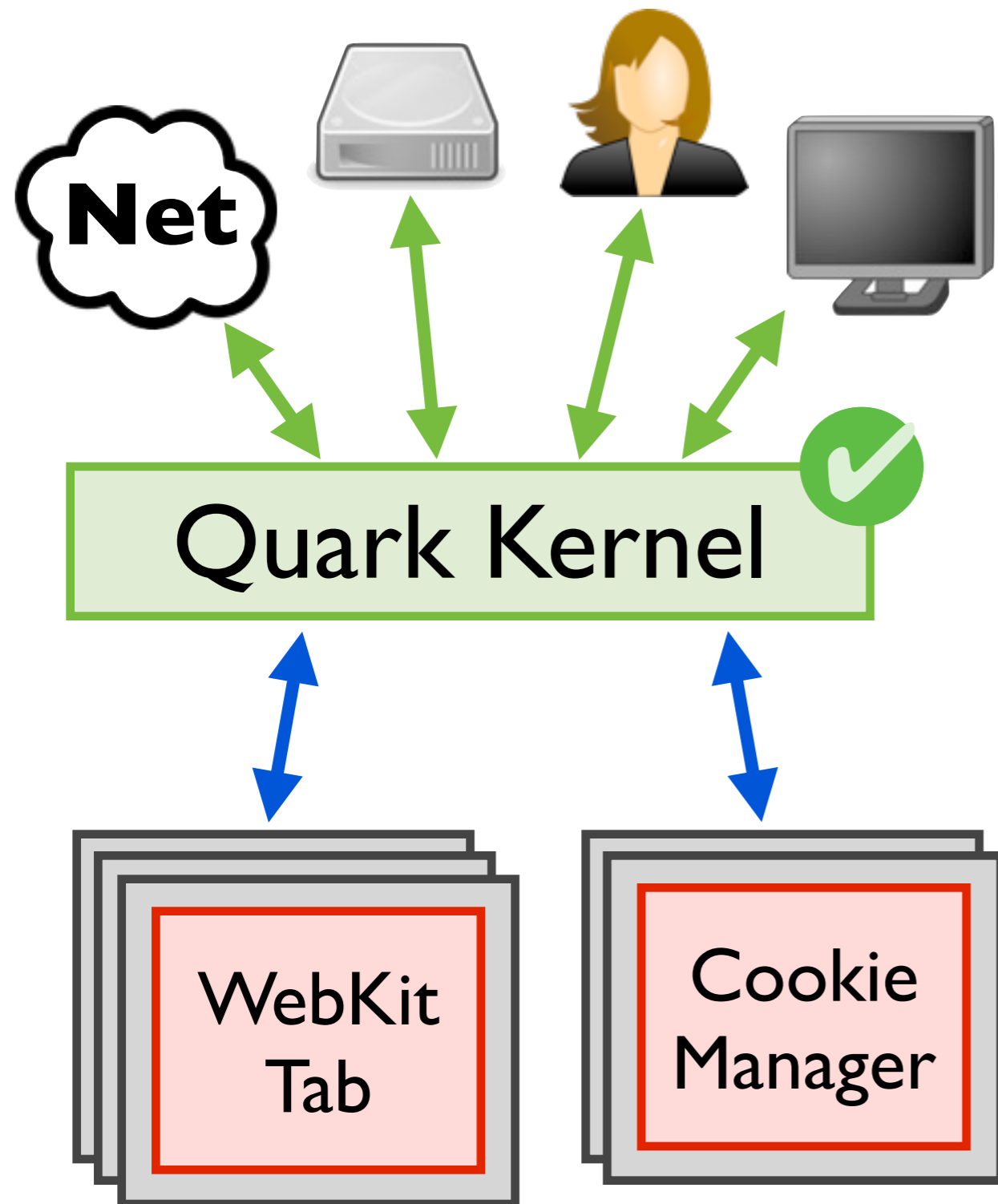
Untrusted Code

two component types

WebKit tabs

cookie managers

Quark: Verified Browser



Resources

Shim

Untrusted Code

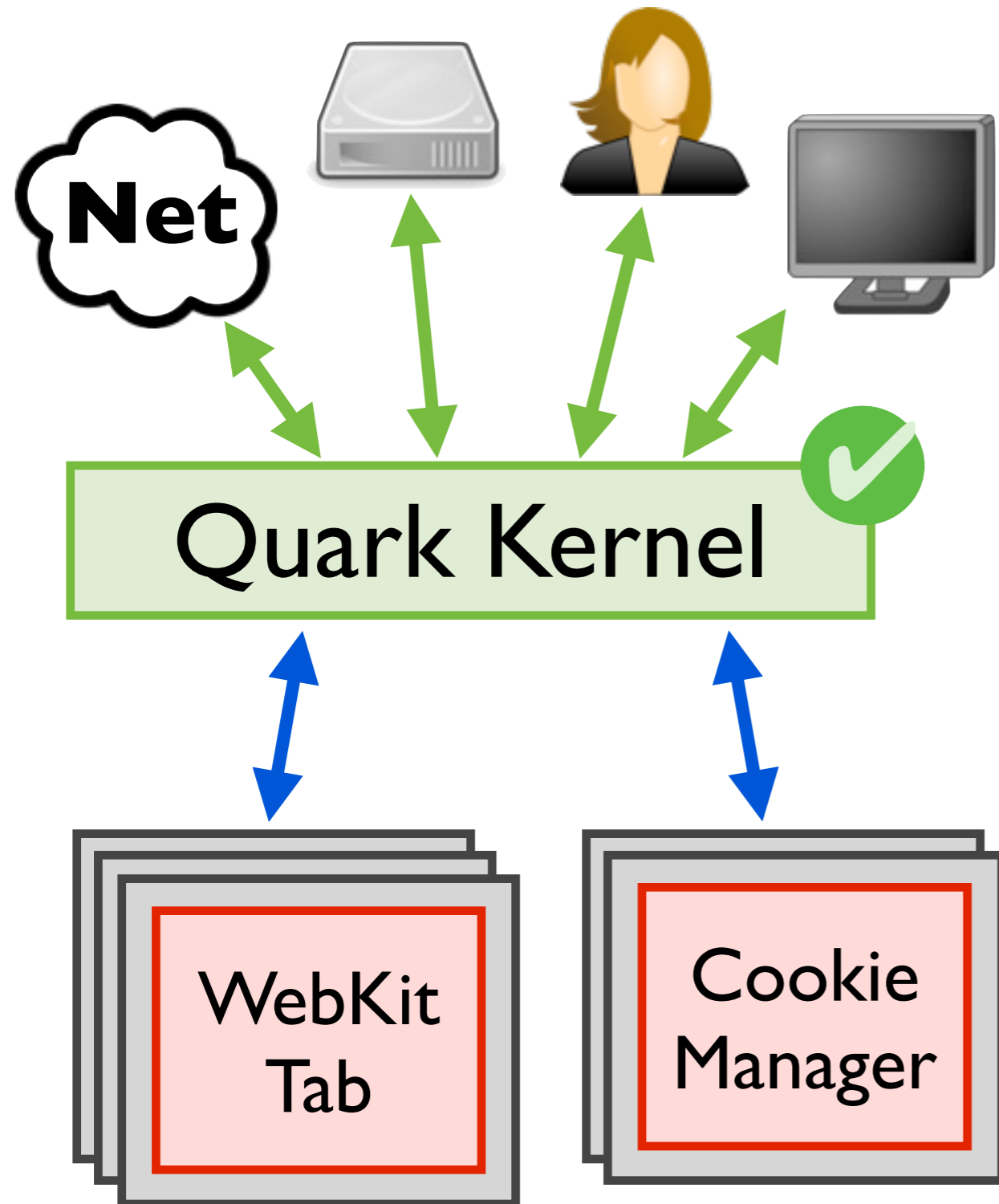
two component types

WebKit tabs

cookie managers

several instances each

Quark: Verified Browser



Quark 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 =>  
    ...
```

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

Specify correct behavior wrt syscall seqs

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

Quark Kernel: Code, *Spec*, Proof

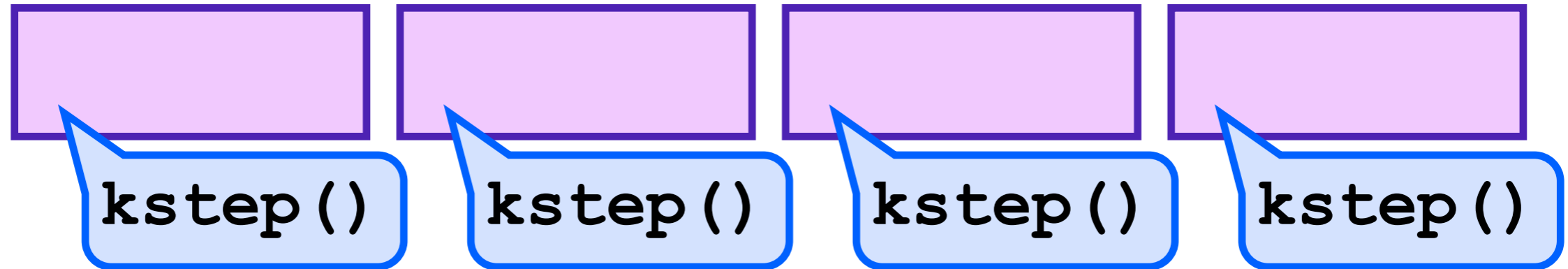
Specify correct behavior wrt syscall seqs



trace: all syscalls made
by Quark kernel
during execution

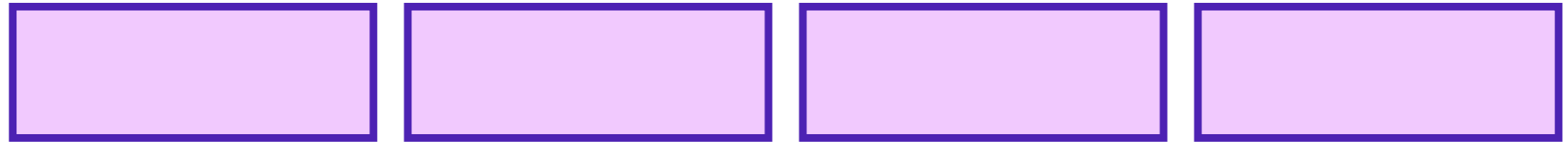
Quark Kernel: Code, *Spec*, Proof

Specify correct behavior wrt syscall seqs



Quark Kernel: Code, *Spec*, Proof

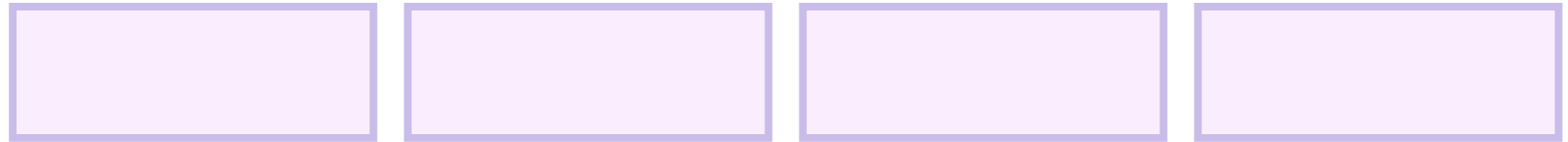
Specify correct behavior wrt syscall seqs



structure of produceable traces supports spec & proof

Quark Kernel: Code, *Spec*, Proof

Specify correct behavior wrt syscall seqs

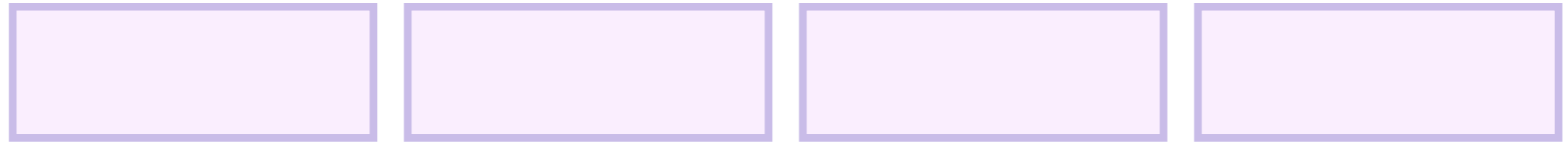


structure of produceable traces supports spec & proof

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

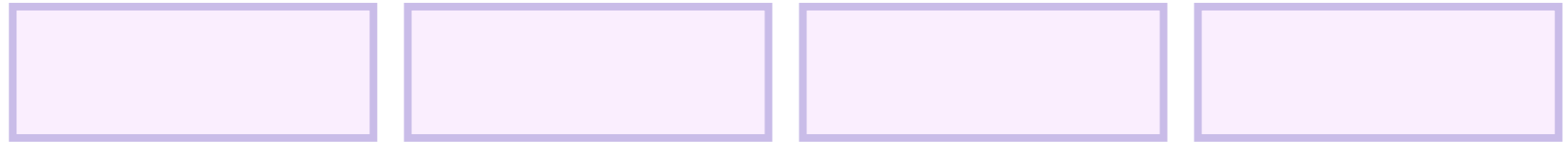
`forall trace tab domain,`

for *any* trace, tab,
and domain

where trace is a
sequence of syscalls

Quark Kernel: Code, *Spec*, Proof

Specify correct behavior wrt syscall seqs



structure of produceable traces supports spec & proof

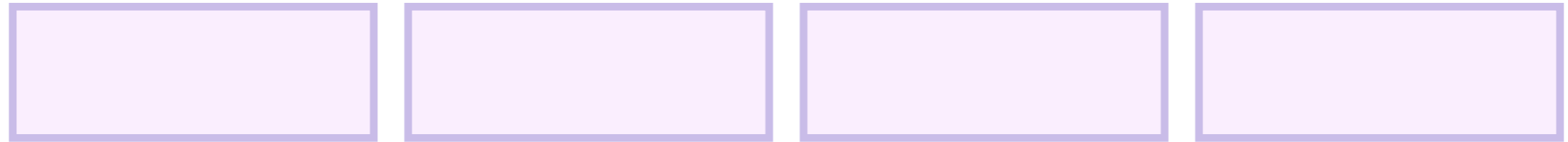
Example: address bar correctness

```
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

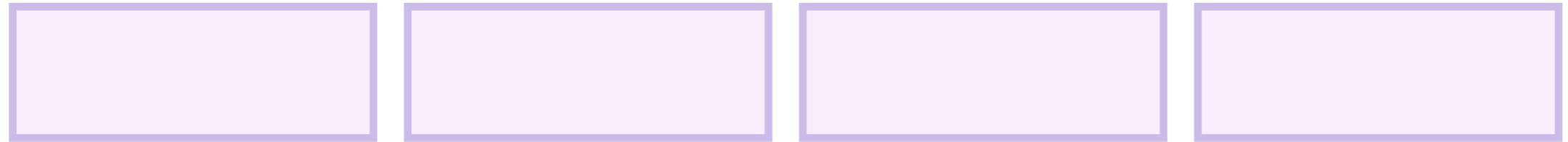
Example: address bar correctness

```
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

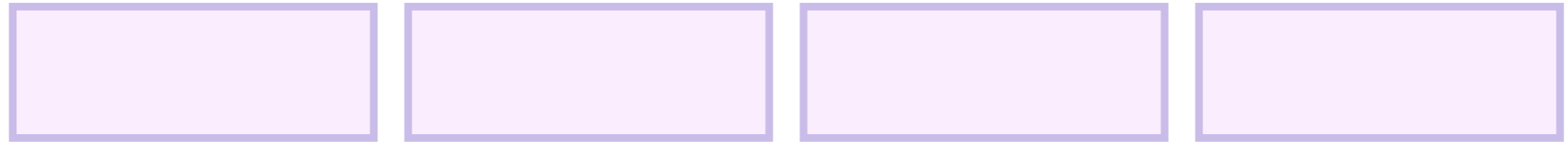
Example: address bar correctness

```
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

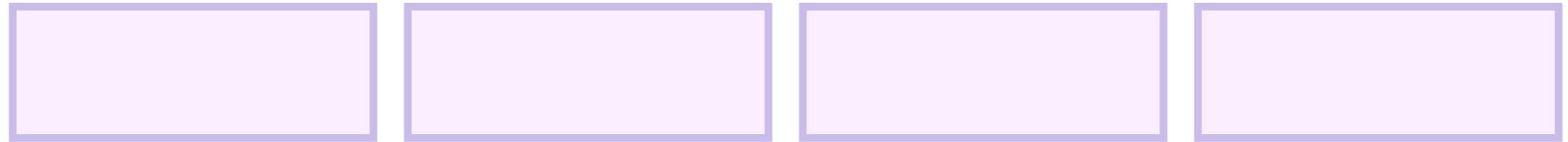
Example: address bar correctness

```
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*

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



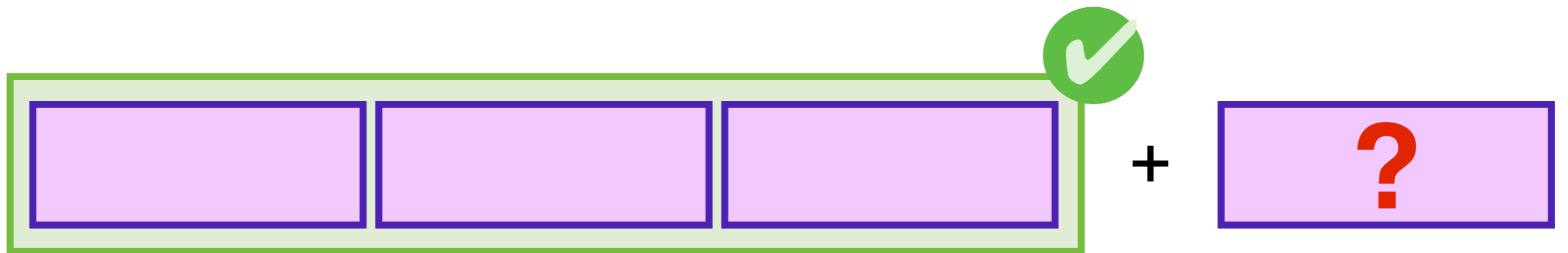
induction hypothesis:

trace valid up to this point

Quark Kernel: Code, Spec, *Proof*

Prove kernel code satisfies sec props

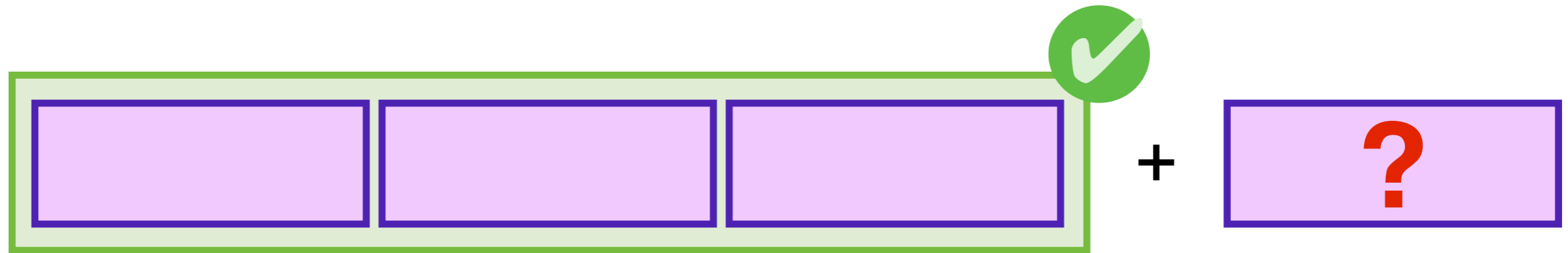
by induction on traces Quark can produce



*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 using interactive proof assistant

Quark Kernel: Code, Spec, Proof

Key Insight

Guarantee sec props for browser

Use state-of-the-art components

Only prove simple browser kernel

Usability Demo Video

The image shows a screenshot of the Google homepage with a usability demo overlay. The browser's address bar at the top contains the following text: `| google.com | maps.google.com | amazon.com | facebook.com |`. Below the address bar is a navigation menu with links for [+Quark](#), [Search](#), [Images](#), [Maps](#), [Play](#), [YouTube](#), [News](#), [Gmail](#), [Documents](#), [Calendar](#), and [More](#). In the top right corner, there is a user profile for "Quark Tester" with a "Share" button and a dropdown arrow. A notification box on the right side of the page reads "A faster way to browse the web" and includes a blue button labeled "Install Google Chrome". The main content area features a large illustration of a cartoon astronaut in a yellow tank top and orange shorts, holding a silver pole and waving, standing on a platform with the word "Google" in the background. Below the illustration is a search input field, followed by "Google Search" and "I'm Feeling Lucky" buttons. At the bottom of the page, there is a promotional message: "And we have lift off! Celebrate 50 years of the Kennedy Space Center with [Google Maps](#)". The footer contains links for "Change background image", "Advertising Programs", "Business Solutions", "Privacy & Terms", "+Google", and "About Google".

Trusted Computing Base

Infrastructure we assume correct

any 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]

...

Security Analysis

Formally prove important sec props

WebKit defenses remain in effect

Other desirable security policies

Future Work

Filesystem access, sound, history
could be implemented w/out major redesign

Finer grained resource accesses
support mashups and plugins

Liveness properties
formally prove that kernel never blocks

Conclusion

Formal Shim Verification

Guarantee sec props for entire system

Only reason about small shim

Radically ease verification burden

Quark: Verified Browser

Guarantee sec props for browser

Only prove simple browser kernel

Use state-of-the-art components

<http://goto.ucsd.edu/quark>