# Formal Verification of Cloud Services

Bryan Parno

https://www.chase.com/

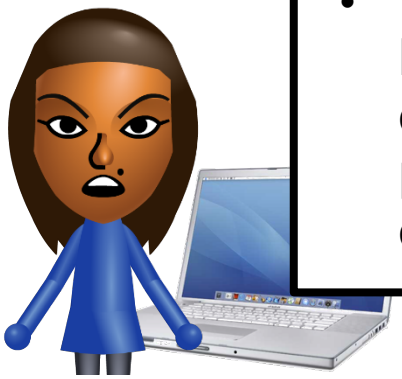
Personal | Business | Commercial

CHASE

CHASE

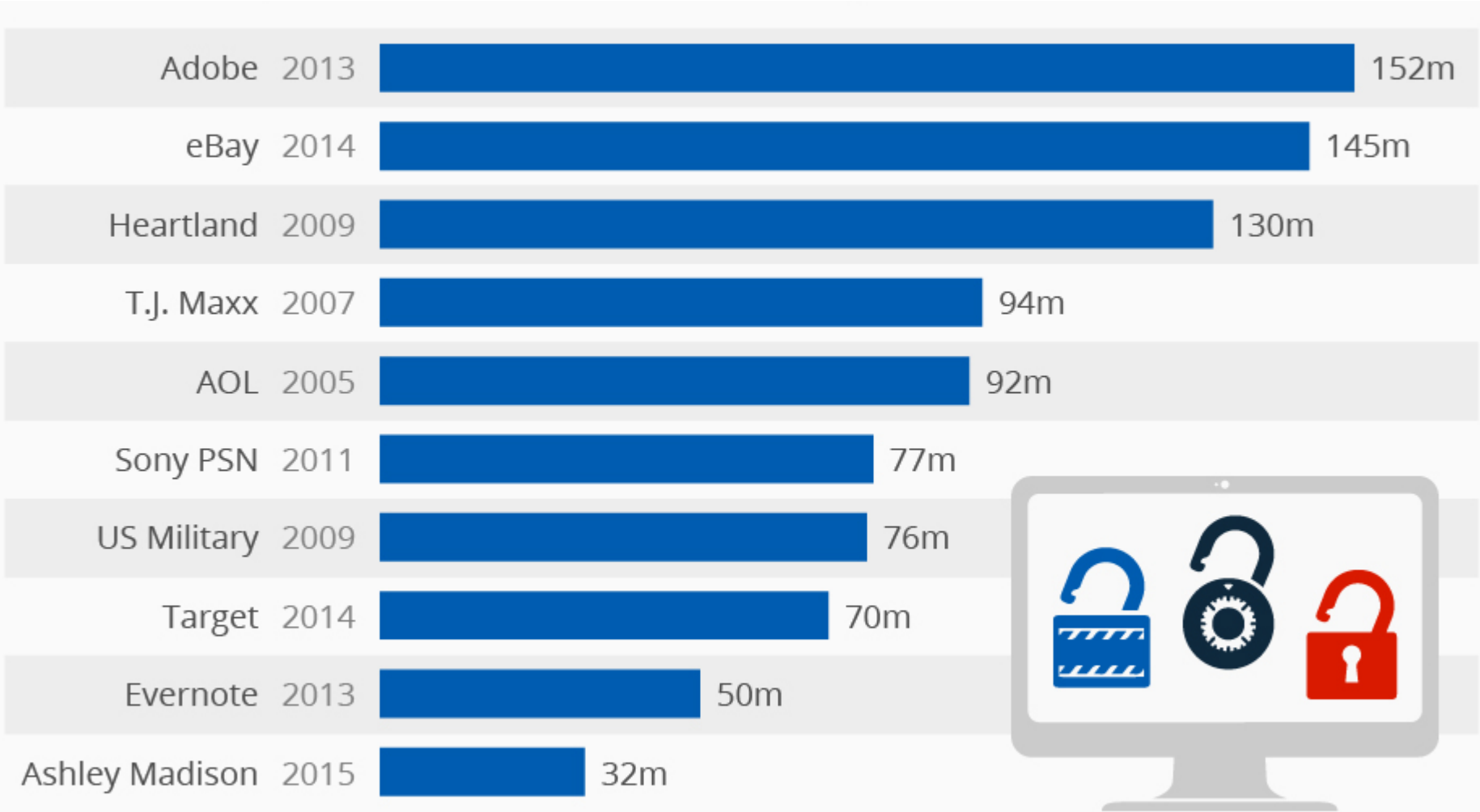
**Online and Mobile Security**

- We periodically review our operations and business practices to make sure they comply with the corporate policies and procedures we follow to protect confidential information



# Large-Scale Data Breaches Affect Millions of Users

Number of compromised records in recent large-scale data breaches




 Source: Media Reports

# The Ironclad Project

Chris Hawblitzel

Jon Howell

Manos Kapritsos

Rustan Leino

Jay Lorch

Arjun Narayan

**Bryan Parno**

Michael Lowell Roberts

Srinath Setty

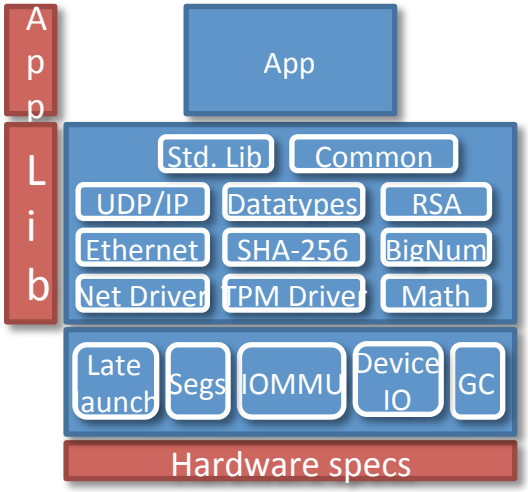
Danfeng Zhang

Brian Zill

# The Ironclad Project

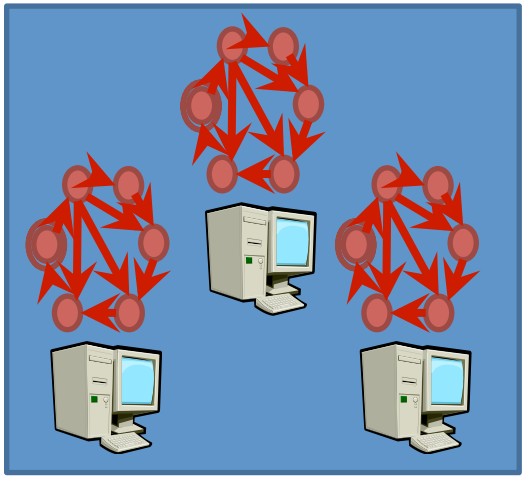
## Ironclad Apps

[OSDI 2014]

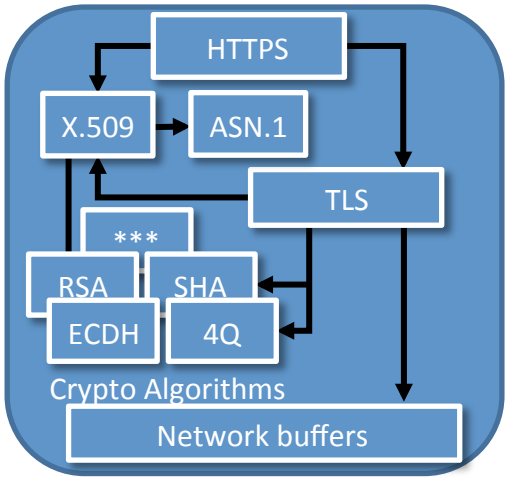



## IronFleet

[SOSP 2015]



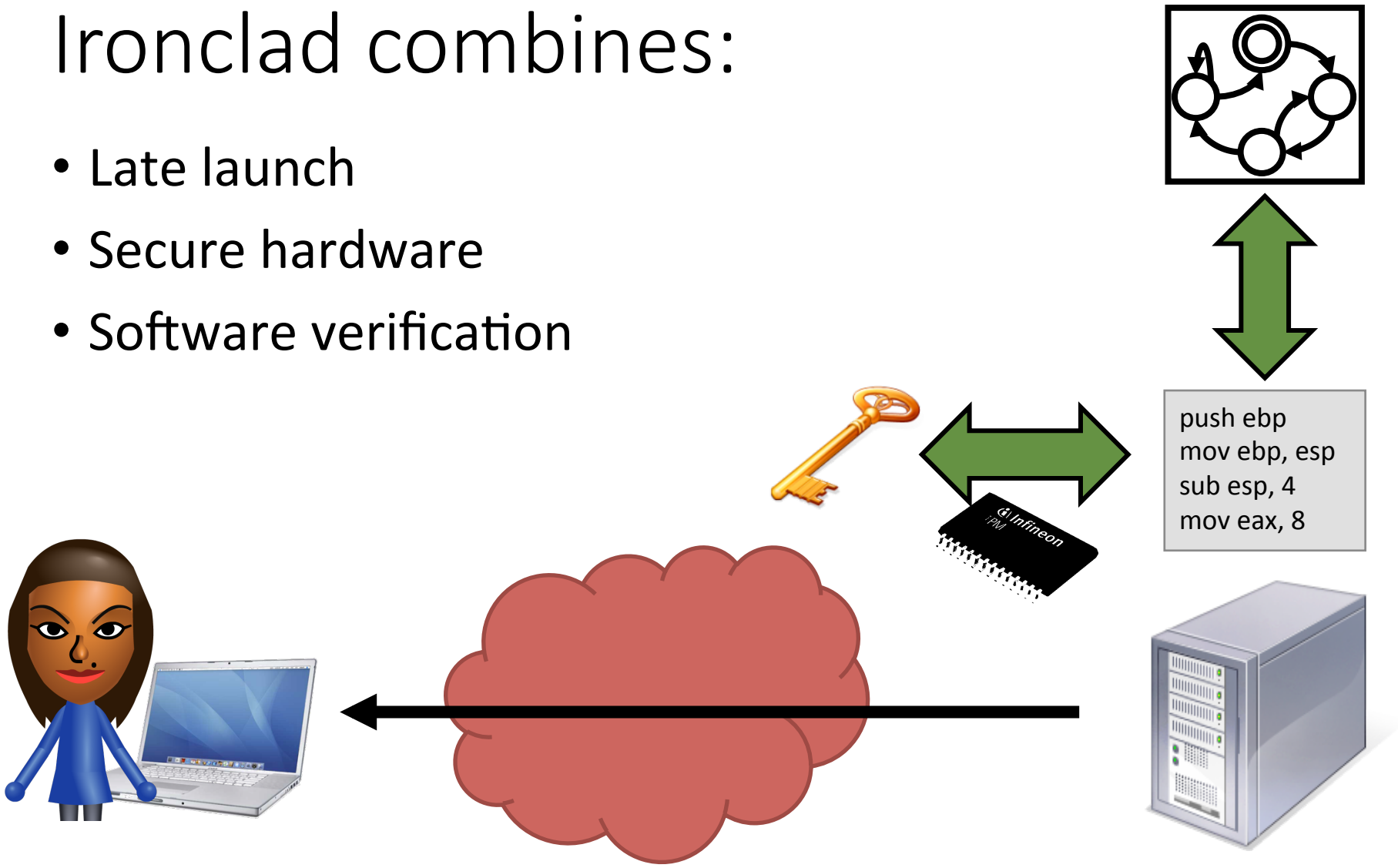
## Everest HTTPS



An *Ironclad app* guarantees to remote parties that every instruction it executes adheres to a high-level security spec.

# Ironclad combines:

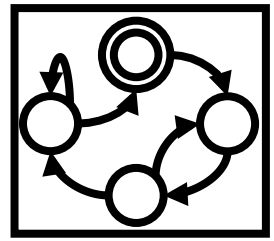
- Late launch
- Secure hardware
- Software verification



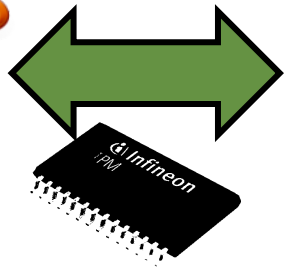
# Ironclad combines:

- Late launch
- Secure hardware
- Software verification

Secure Remote Equivalence

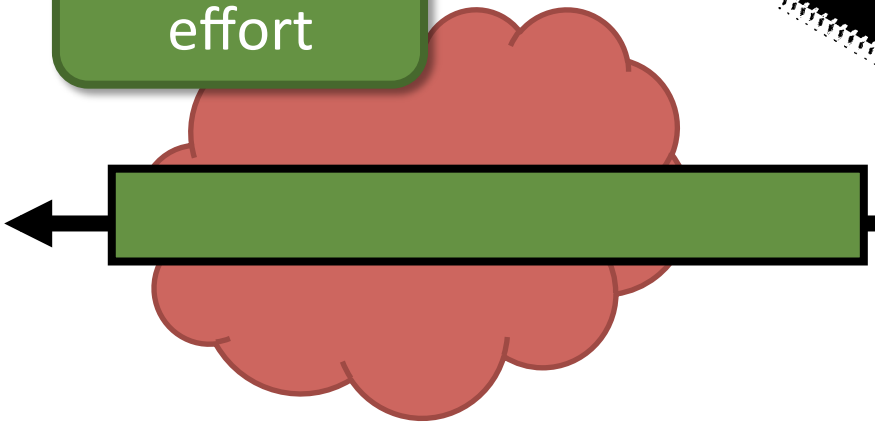


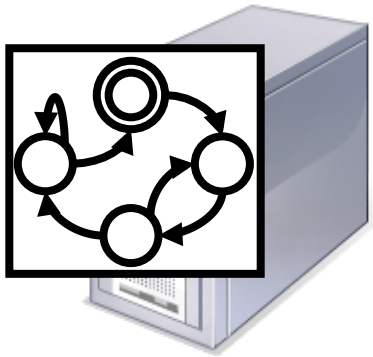
push ebp
mov ebp, esp
sub esp, 4
mov eax, 8




Entire software stack

Reasonable effort



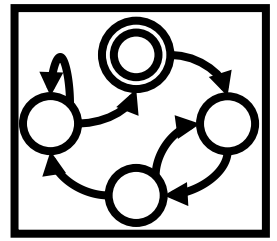


# Verification goals

- End-to-end security
  - Complete
  - Low-level
- Rapid development

- Non-goal: Verify existing code

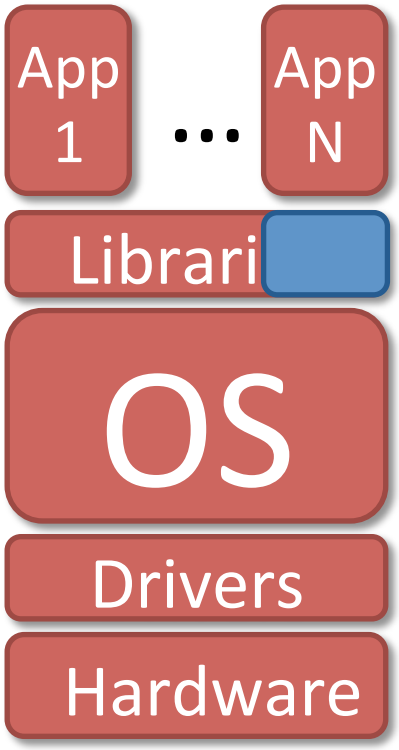
- Long-term: Performance matches unsafe code



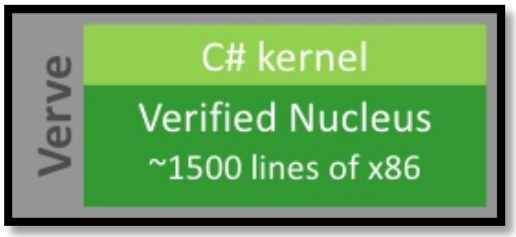
```
push ebp
mov ebp, esp
sub esp, 4
mov eax, 8
```

# Prior work

- Small components
  - E.g., RSA-OAEP
  - Localized guarantee
- Single layers
  - No end-to-end guarantee

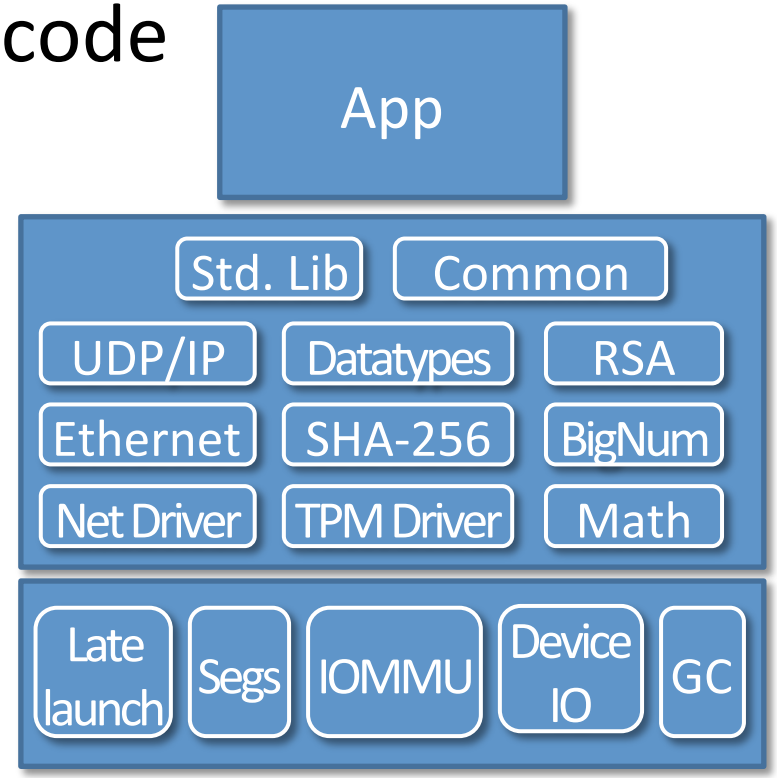




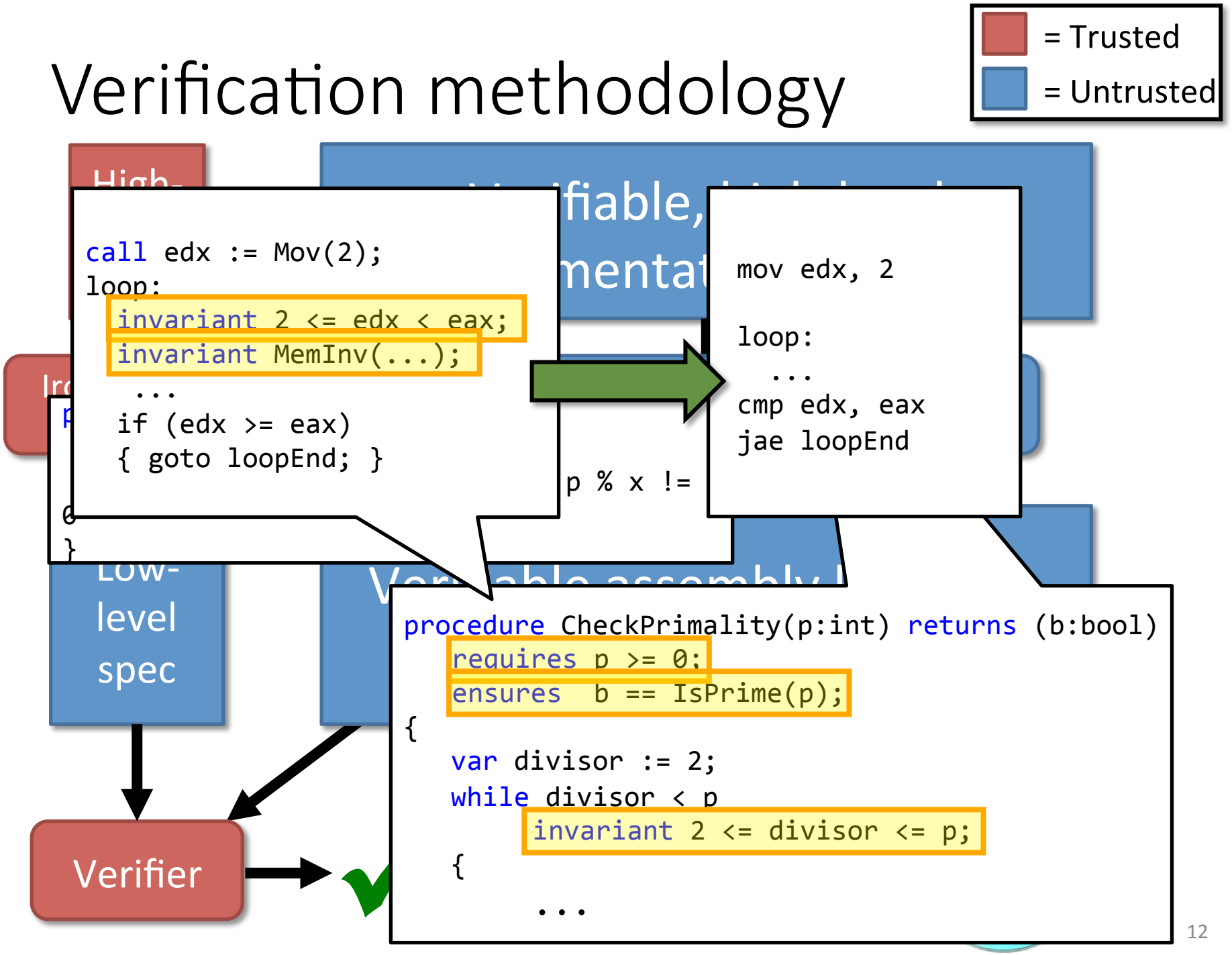



# Our formal, end-to-end guarantee

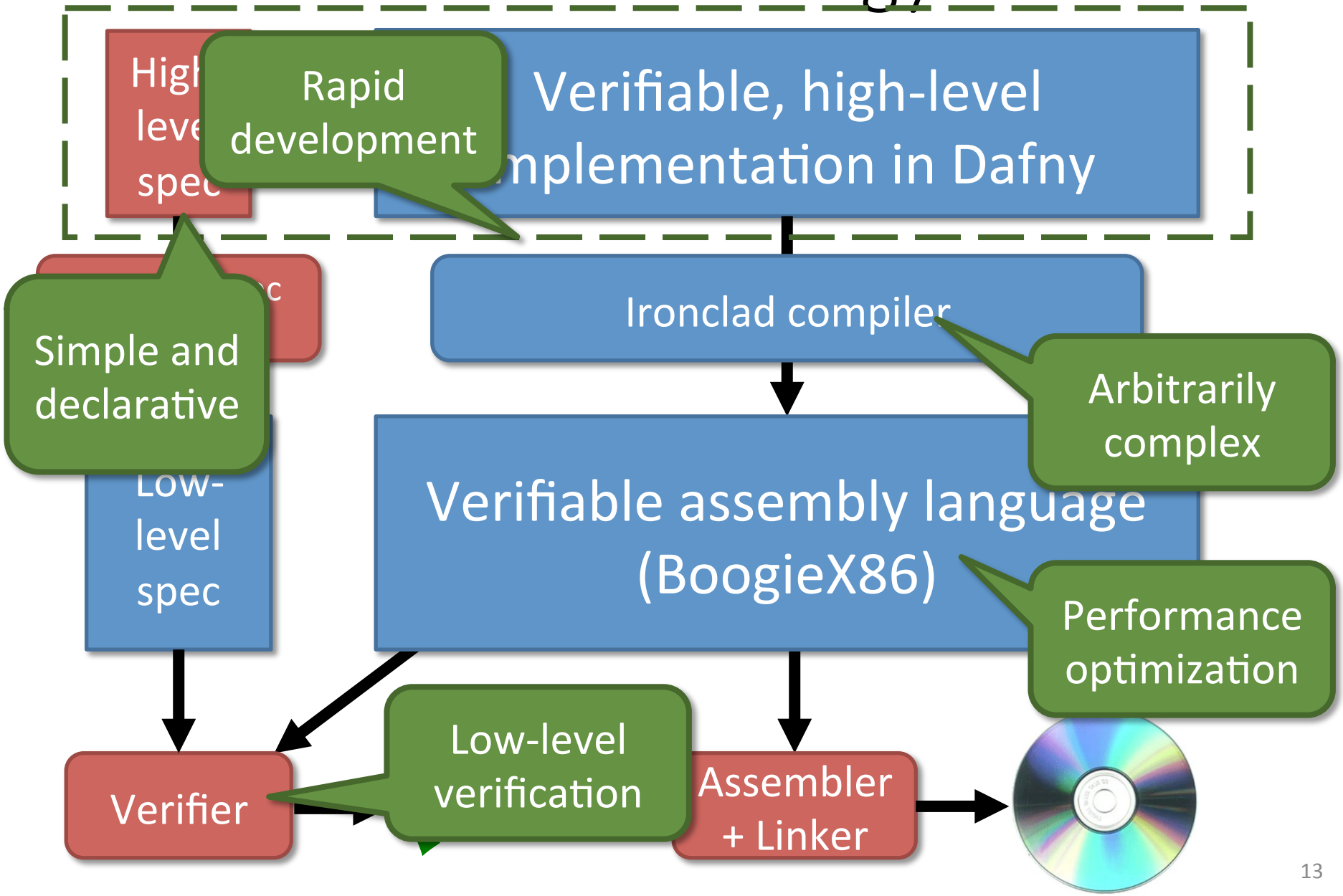
- End-to-end secure communication with provably secure assembly code
- Implies:
  - No buffer overflows
  - No code injection
  - No type-safety flaws
  - No information disclosures
  - No crypto impl flaws
  - …



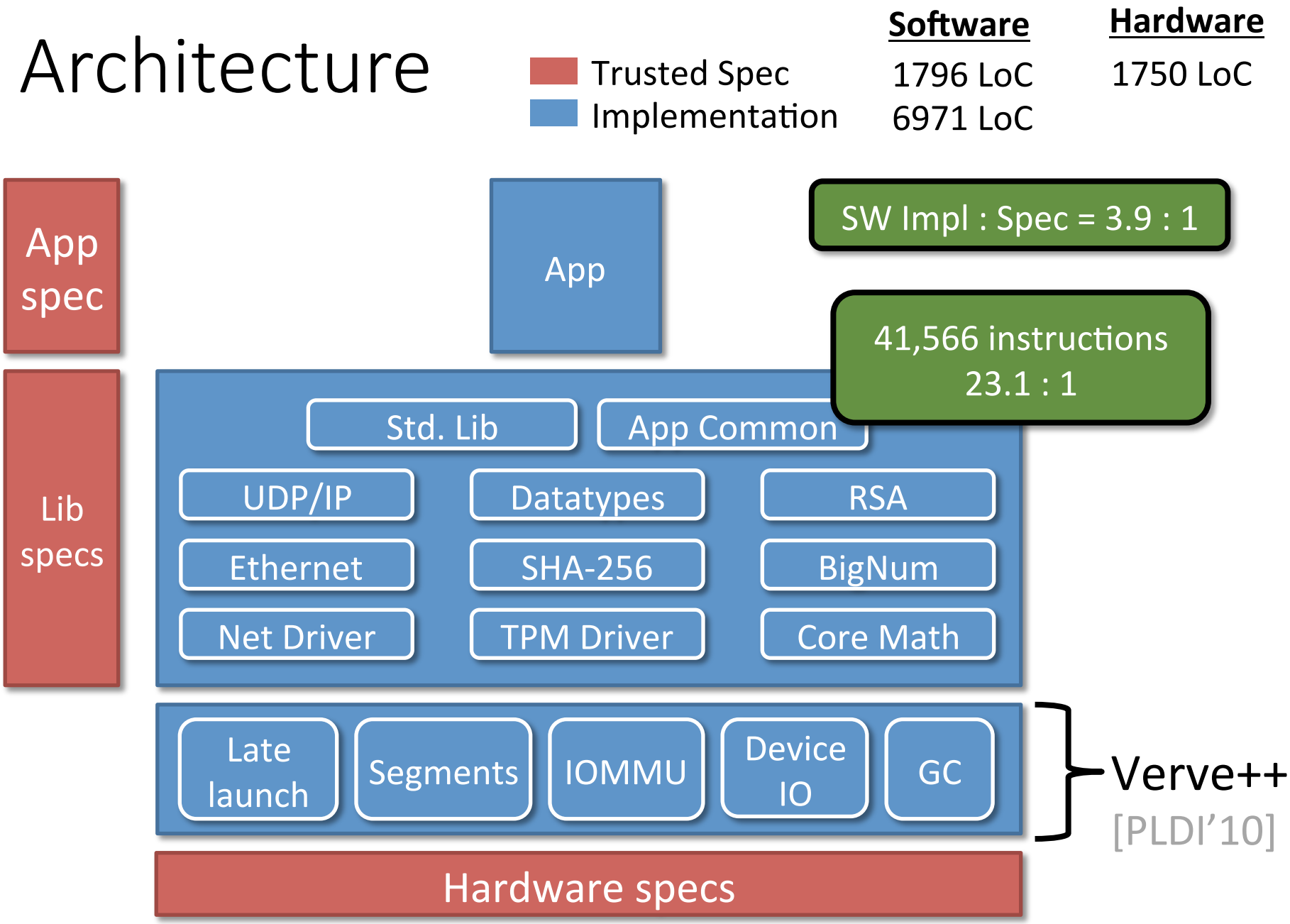
# Verification methodology



# Verification methodology: Benefits



# Architecture



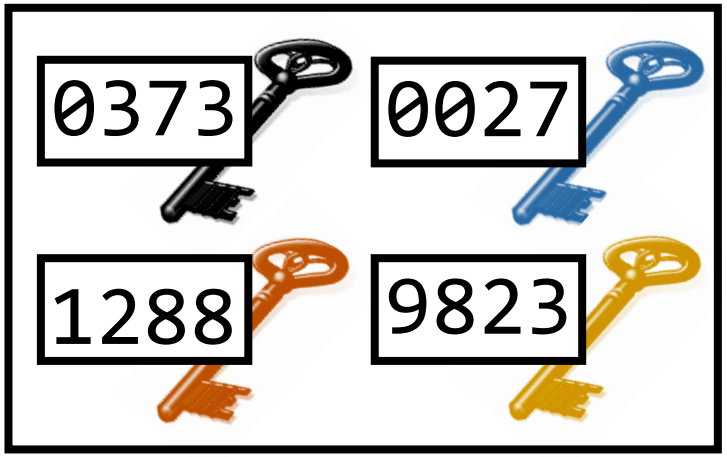
# Ironclad Apps

## Password Protector

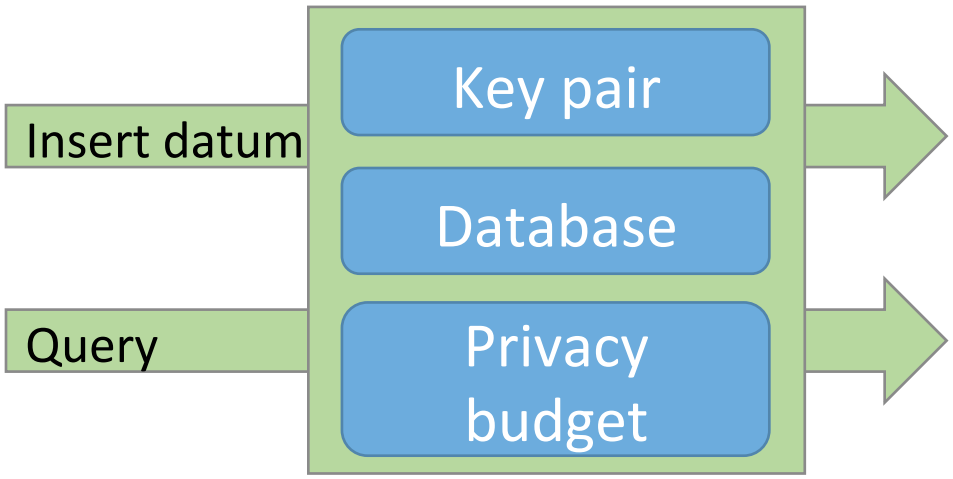
password letmein
123456 dragon
123 1111
abc baseball
monkey loveyou
qwerty trustno1

## Notary

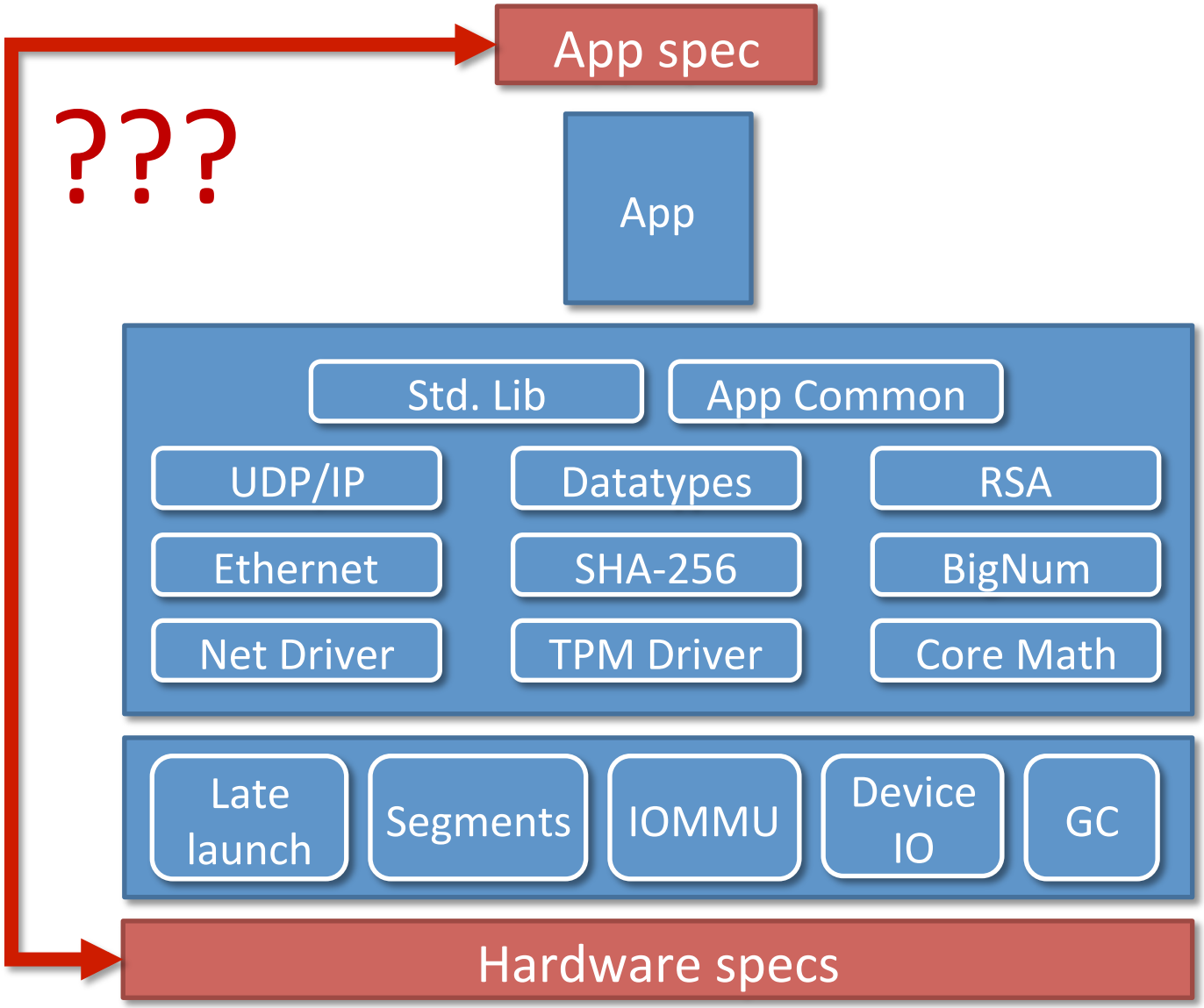
## Trusted Incrementer




## Differentially Private DB

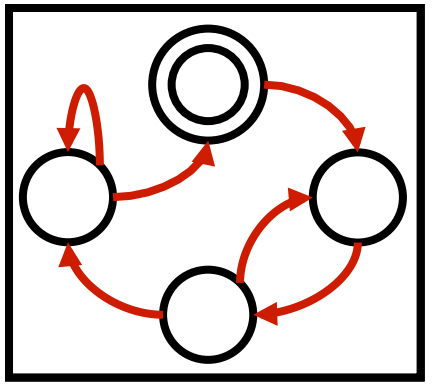


# Key Challenge: Specifying security



# Specifying the DiffPriv DB

App spec



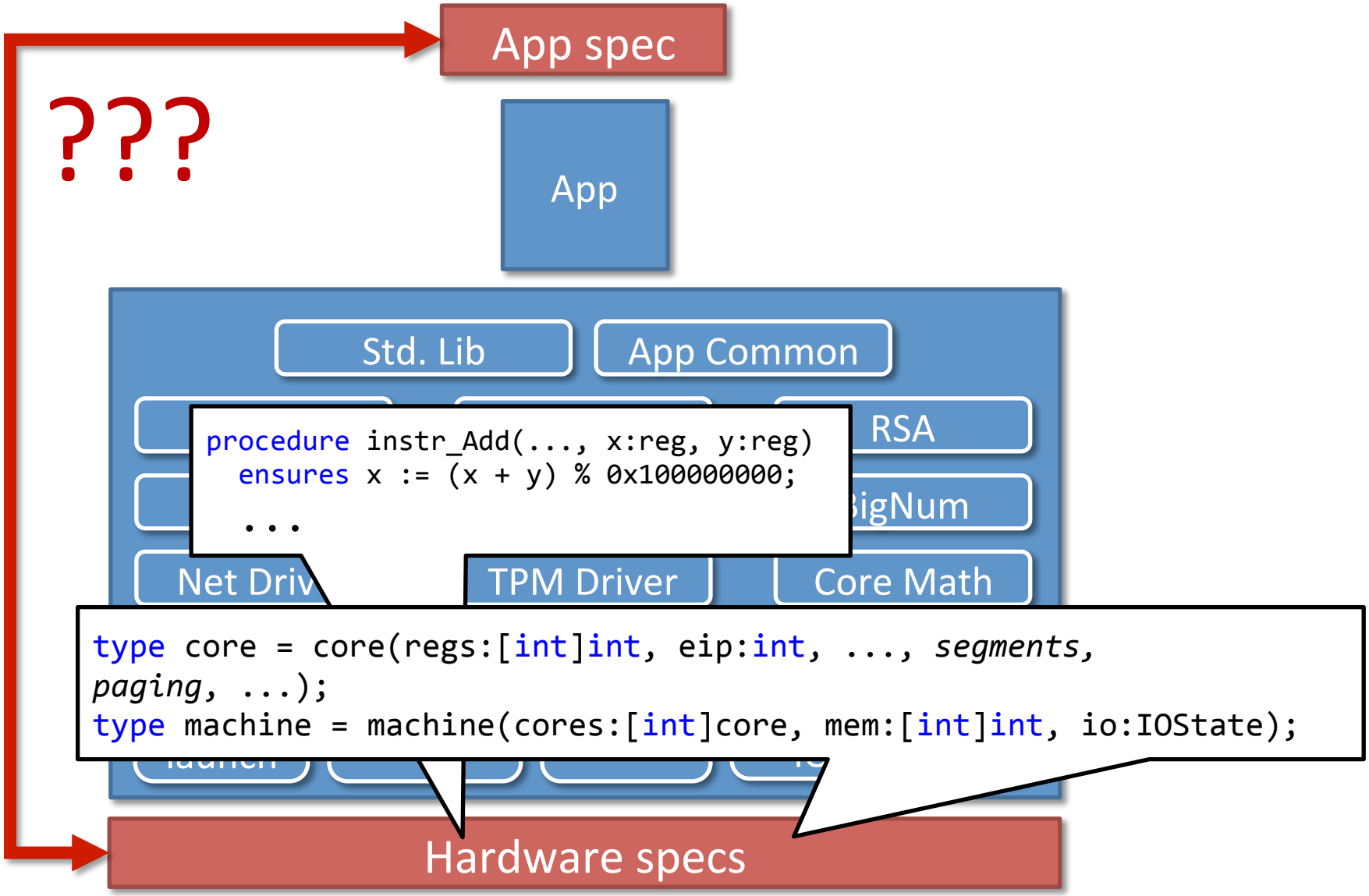
```
datatype DiffPrivState(
        keys:RSA,
        budget:real,
        db:seq<Row>
)

predicate Transition(old_state:DiffPrivState,
                     new_state:DiffPrivState,
                     request:Request,
                     reply:Reply)
{
  match request
      ...
      case Query =>
            request.spend < old_state.budget

         && new_state.budget == old_state.budget - request.spend

         && reply.answer == RunQuery(request.query, old_state.db)
                          + ComputeNoise(request.spend)
         && ...
```

# Key Challenge: Specifying security



# Connecting the app to the hardware

```
procedure instr_inb(..., x:reg)
```

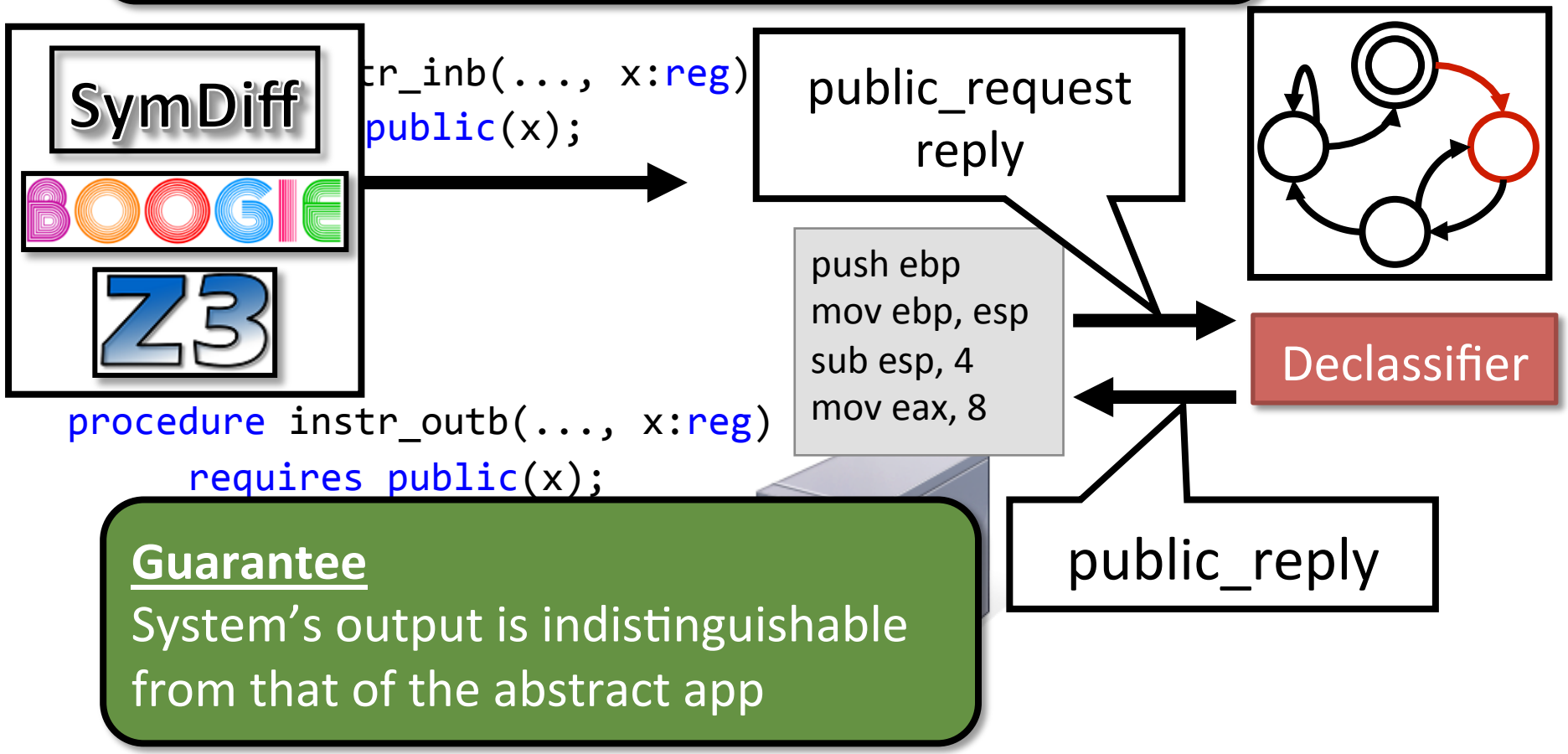
F( all possible inputs ) → all permitted output words

```
procedure instr_outb(..., x:reg)
    requires ????
```

```
push ebp
mov ebp, esp
sub esp, 4
mov eax, 8
```

# Connecting the app to the hardware

**Key idea**
State-machine-based relational verification




**Guarantee**
System's output is indistinguishable from that of the abstract app

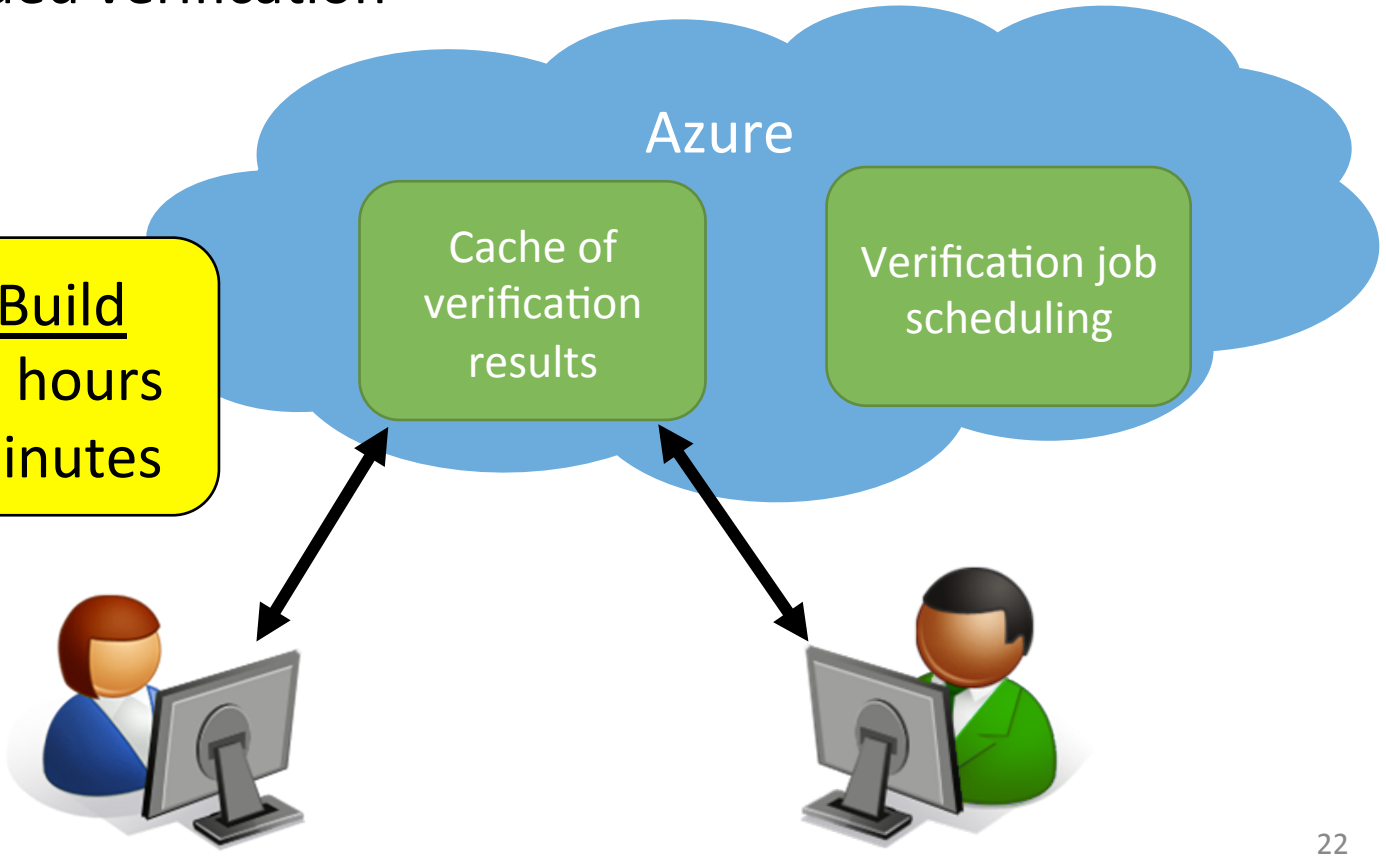
# Evaluation questions

- How does verification affect development?
- How does verification affect performance?

# Eval: Developer experience

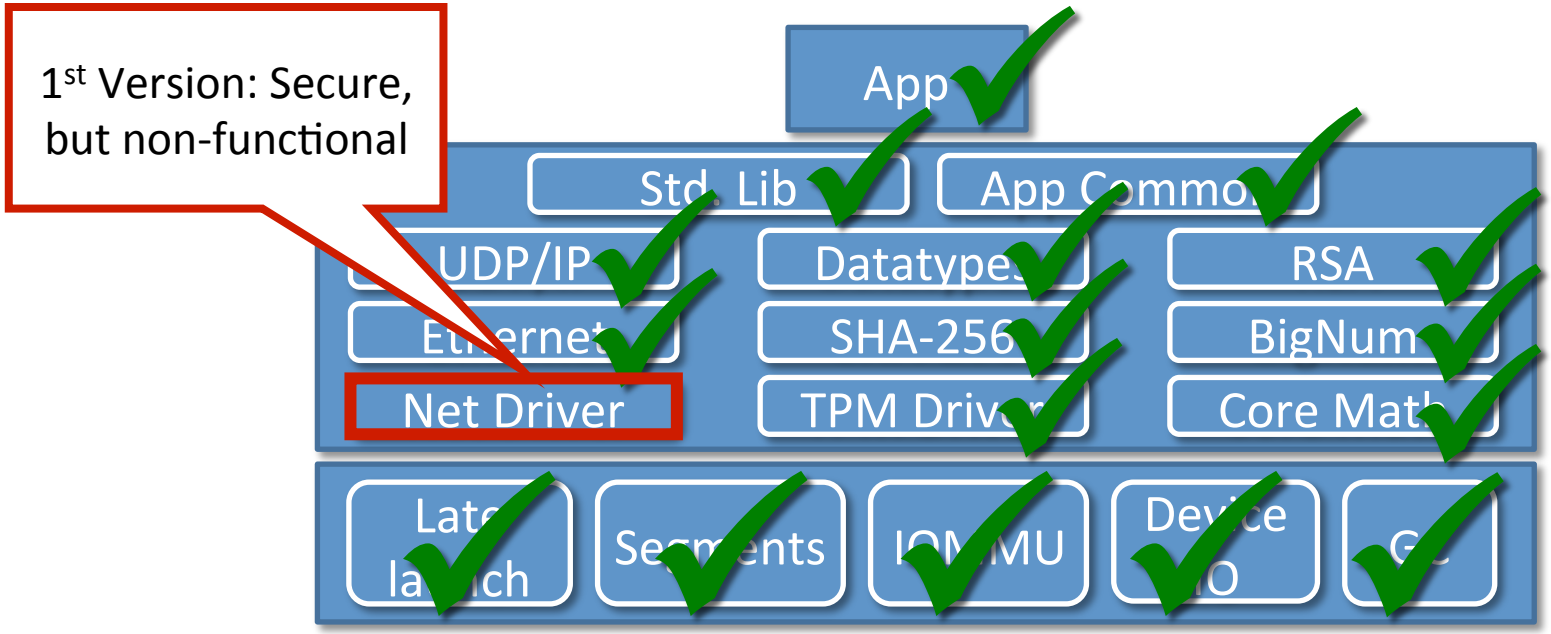
- Rapid feedback via aggressive caching:
  - Method-level via Dafny's IDE plugins
  - File-level via new Dafny `include` feature
  - Cloud-aided verification

| | Full Build |
|---|---|
| Sequential: | 30+ hours |
| Cloud: | 8 minutes |



# Eval: Developer experience

Verified software works!

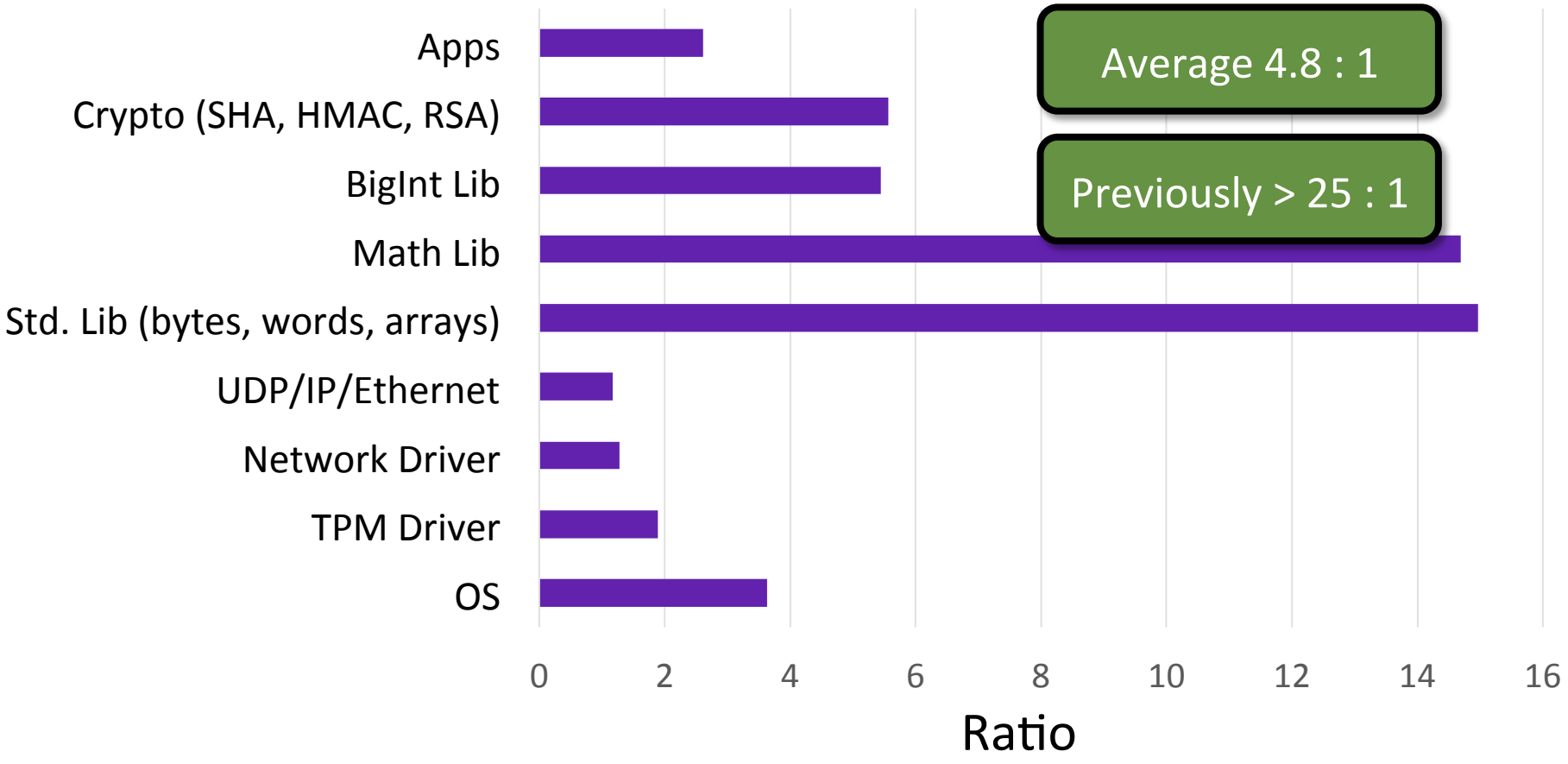


# Eval: Proof burden

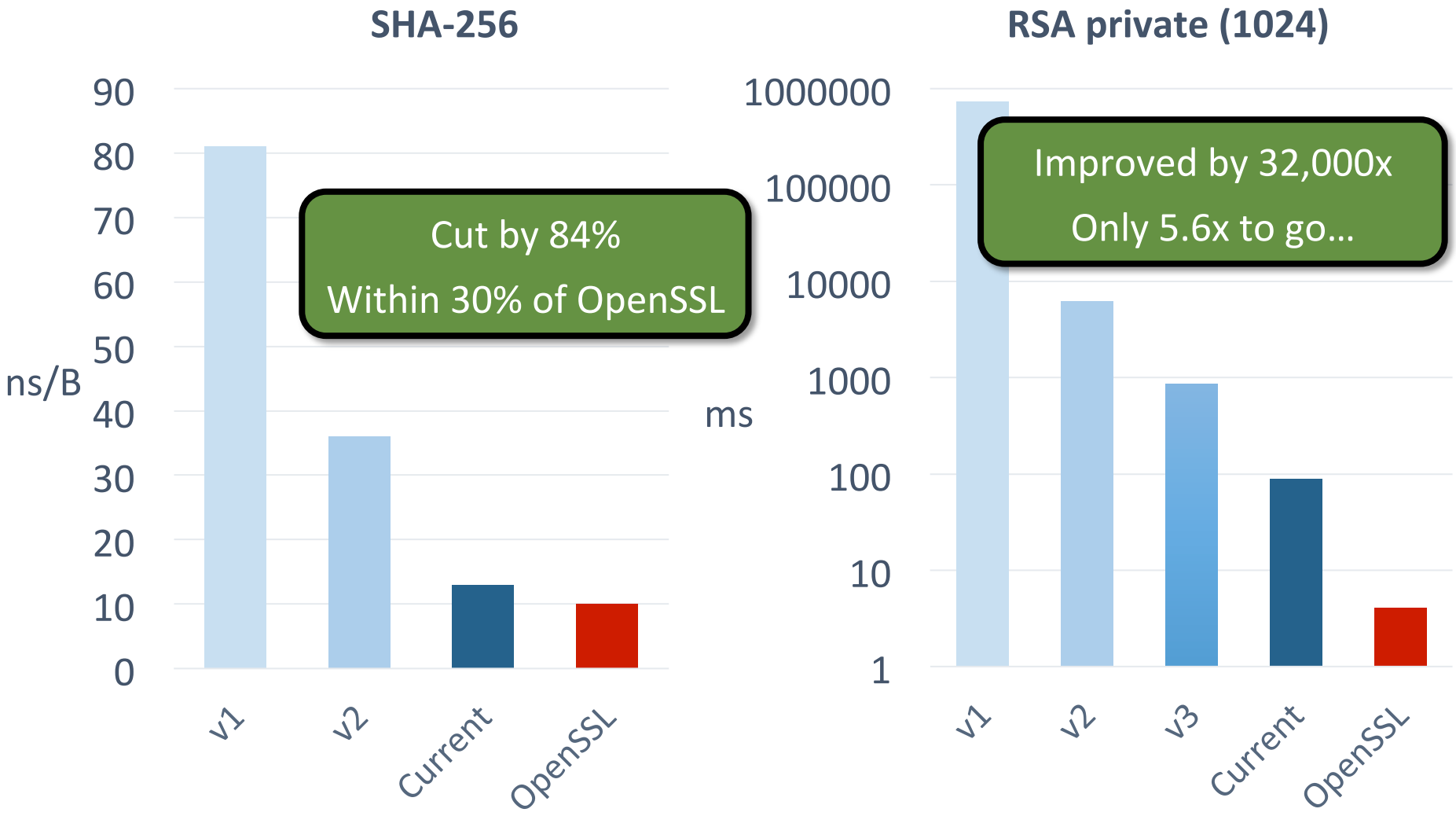
~3 person-years

Previously 22+ pys

## Proof hints : Implementation LoC



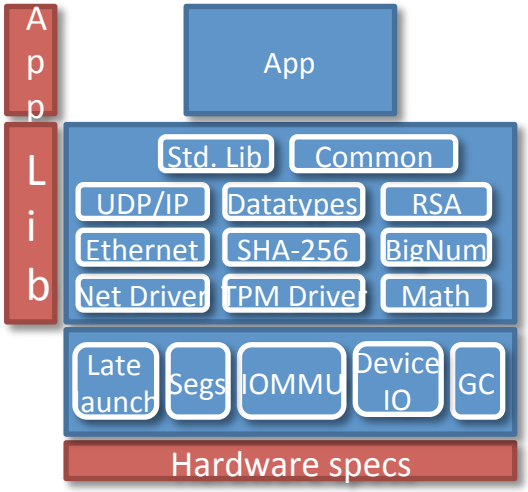
# Eval: Performance



# The Ironclad Project

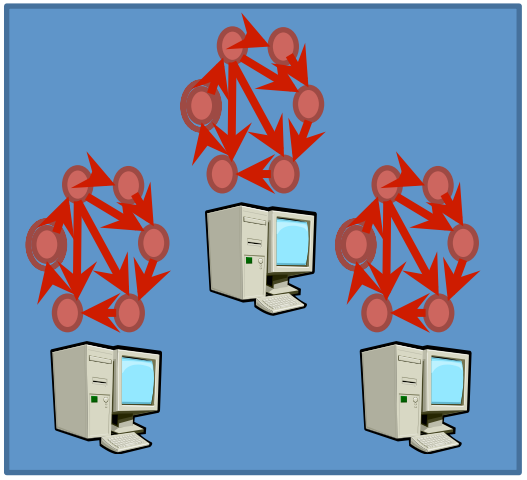
## Ironclad Apps

[OSDI 2014]

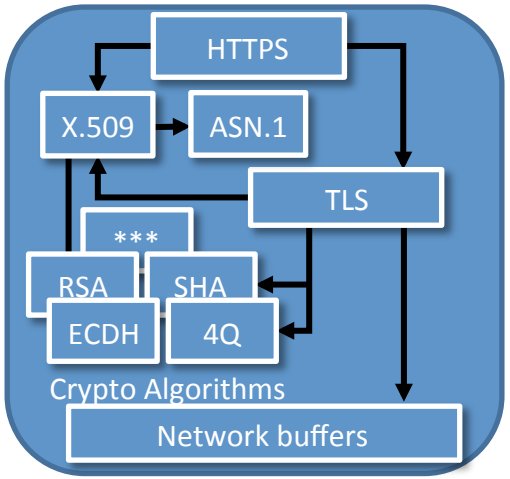



## IronFleet

[SOSP 2015]



## Everest HTTPS



# IronFleet: Proving Practical Distributed Systems Correct

[SOSP 2015]

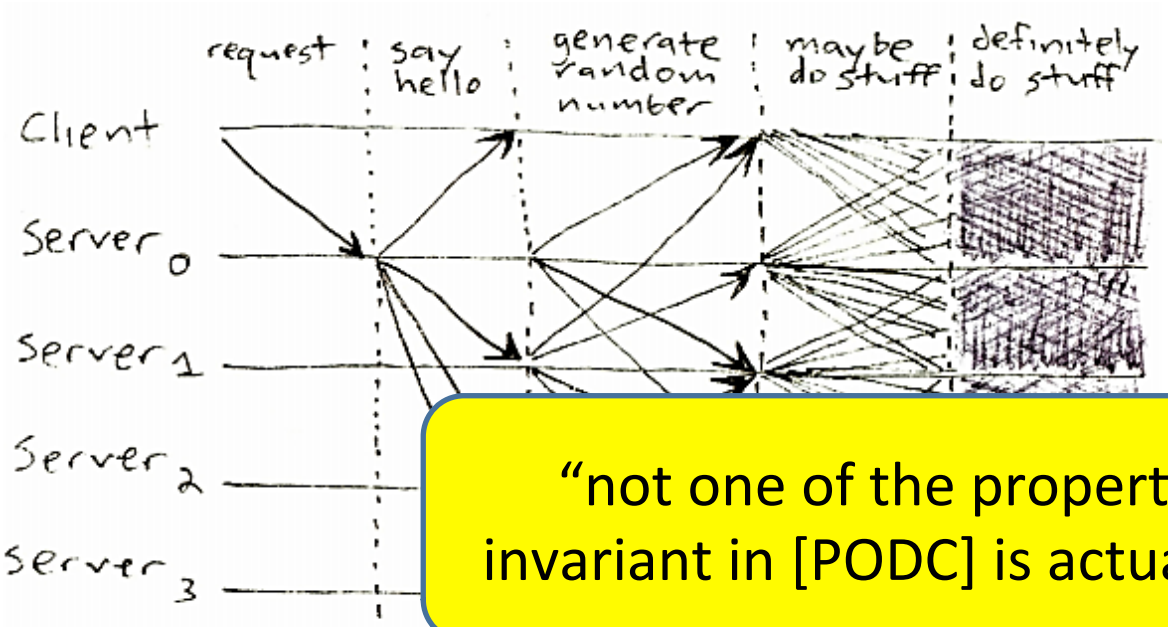
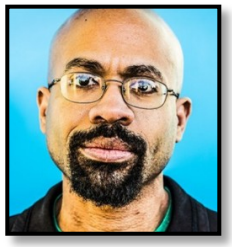



**Figure 1:** Typical Figure 2 from Byzantine fault paper: Our network protocol



[Mickens 2013]

"not one of the properties claimed invariant in [PODC] is actually true of it."

[Zave 2015]

Under the same assumptions ma[illegible]n the Chord papers, the [SIGCOMM] version of the protocol is not correct, and not one of the properties claimed invariant in [PODC] is actually invariantly true of it. The [PODC] version satisfies one invariant, but is still not correct. The results are presented by means of counterexamples to the invariants in Section 4. In preparation for the results, Section 2 gives a brief summary of the protocol and failure assumptions, and Section 3 introduces the invariants.

# IronFleet

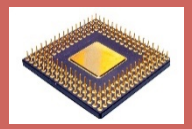
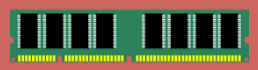
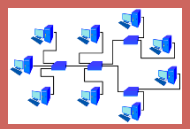
[SOSP

Implementations are correct, not just abstract protocols

*We show how to build com[illegible]ex, efficient distributed systems whose implementations are provably safe and live.*

System will not take incorrect actions

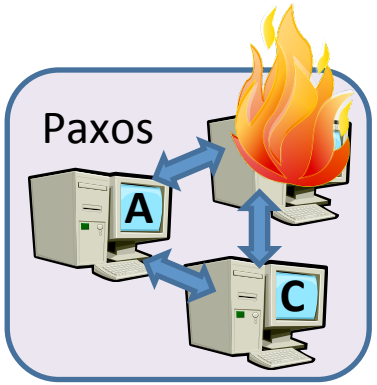
[illegible] absolute


# Our verified distributed systems

IronRSL
Replicated state library

IronKV
Sharded key-value store

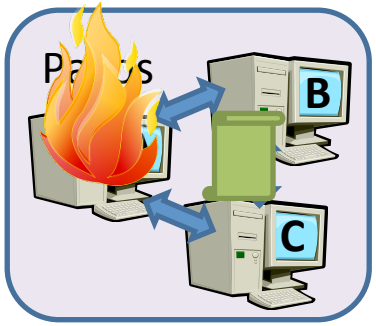


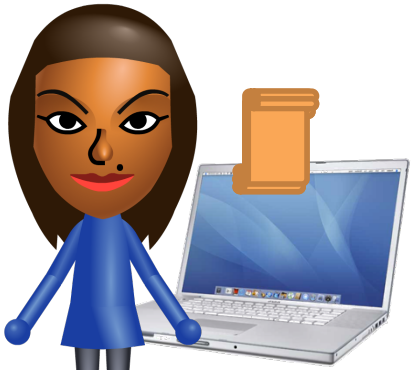

=

**Complex with many features:**

- state transfer
- log truncation
- dynamic view-change timeouts
- batching
- reply cache

# Running example: IronRSL replicated state library



Safety property:
Equivalence to single machine

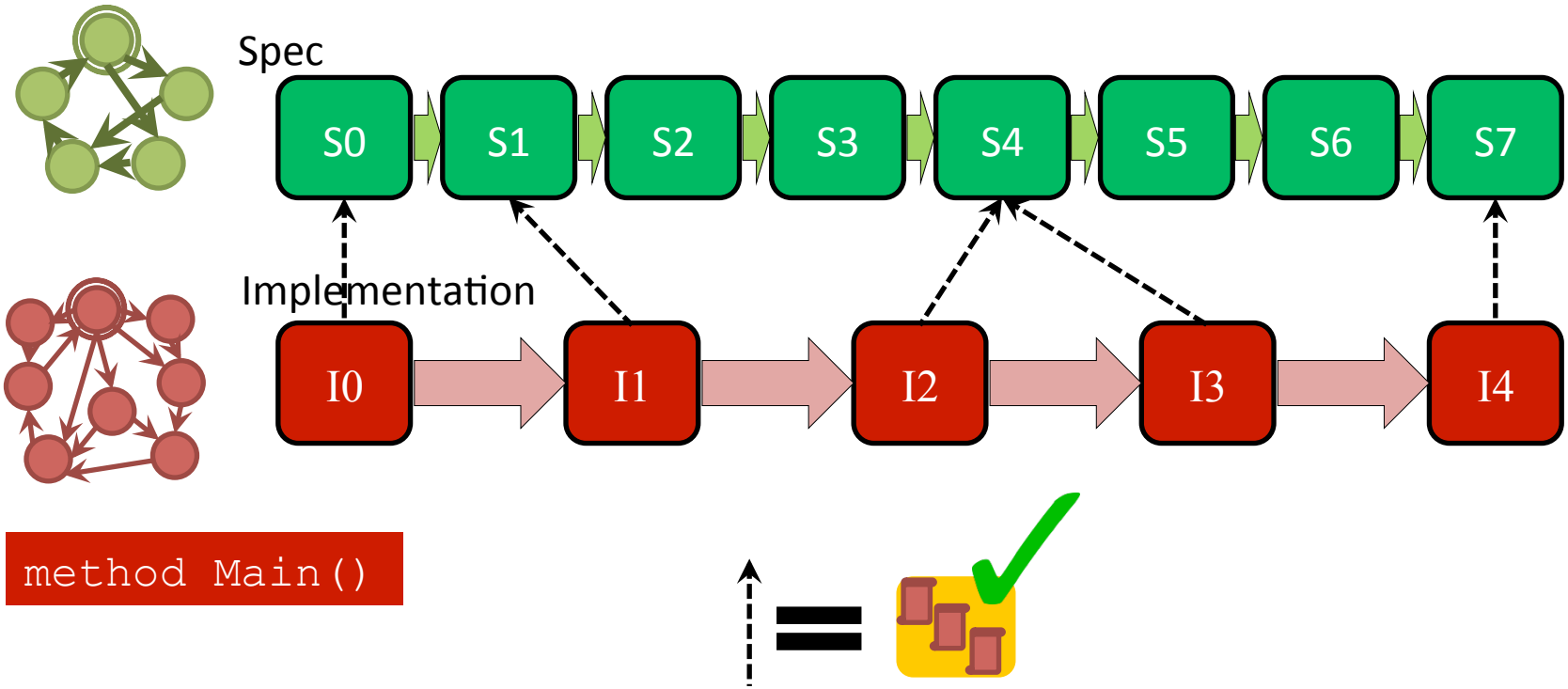
Liveness property:
Clients eventually get replies

# Verification rules out *all* bugs by construction

| | |
|---|---|
| Invariant violations | Parsing errors |
| Race conditions | Marshalling errors |
| Integer overflow | Deadlock |
| Buffer overflow | Livelock |

# Background: Refinement

[Milner 1971, Park 1981, Lamport 1983, Lynch 1983, ...]

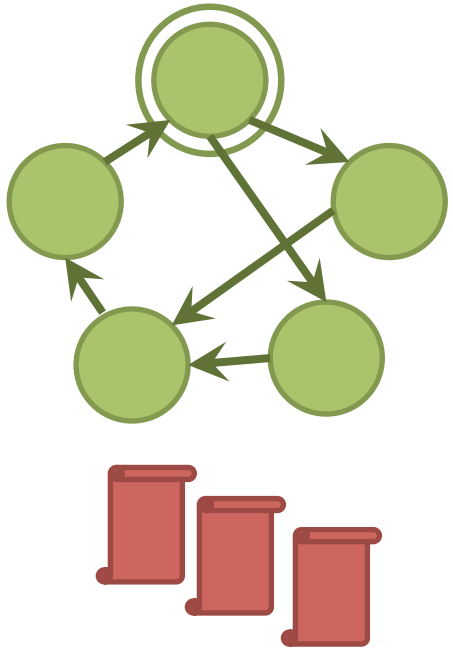


# Specification is a simple, logically centralized state machine

```
type SpecState = int
```

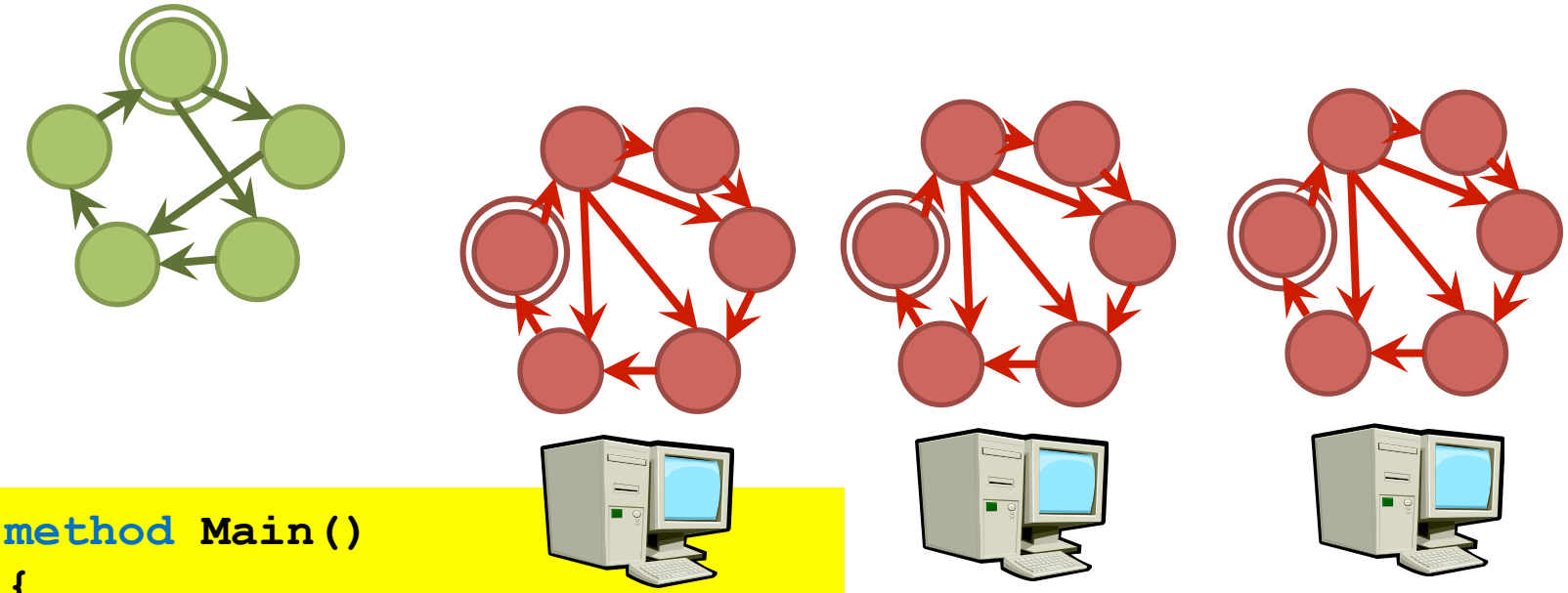
```
function SpecInit(s:SpecState) : bool
{
    s == 0
}
```

```
function SpecNext(s:SpecState,
                  s':SpecState) : bool
{
    s' == s + 1
}
```



```
function SpecRelation(realPackets:set<Packet>,
                      s:SpecState) : bool
{
    forall p, i :: p in realPackets &&
                   p.msg == MarshallCounterVal(i) ==> i <= s
}
```

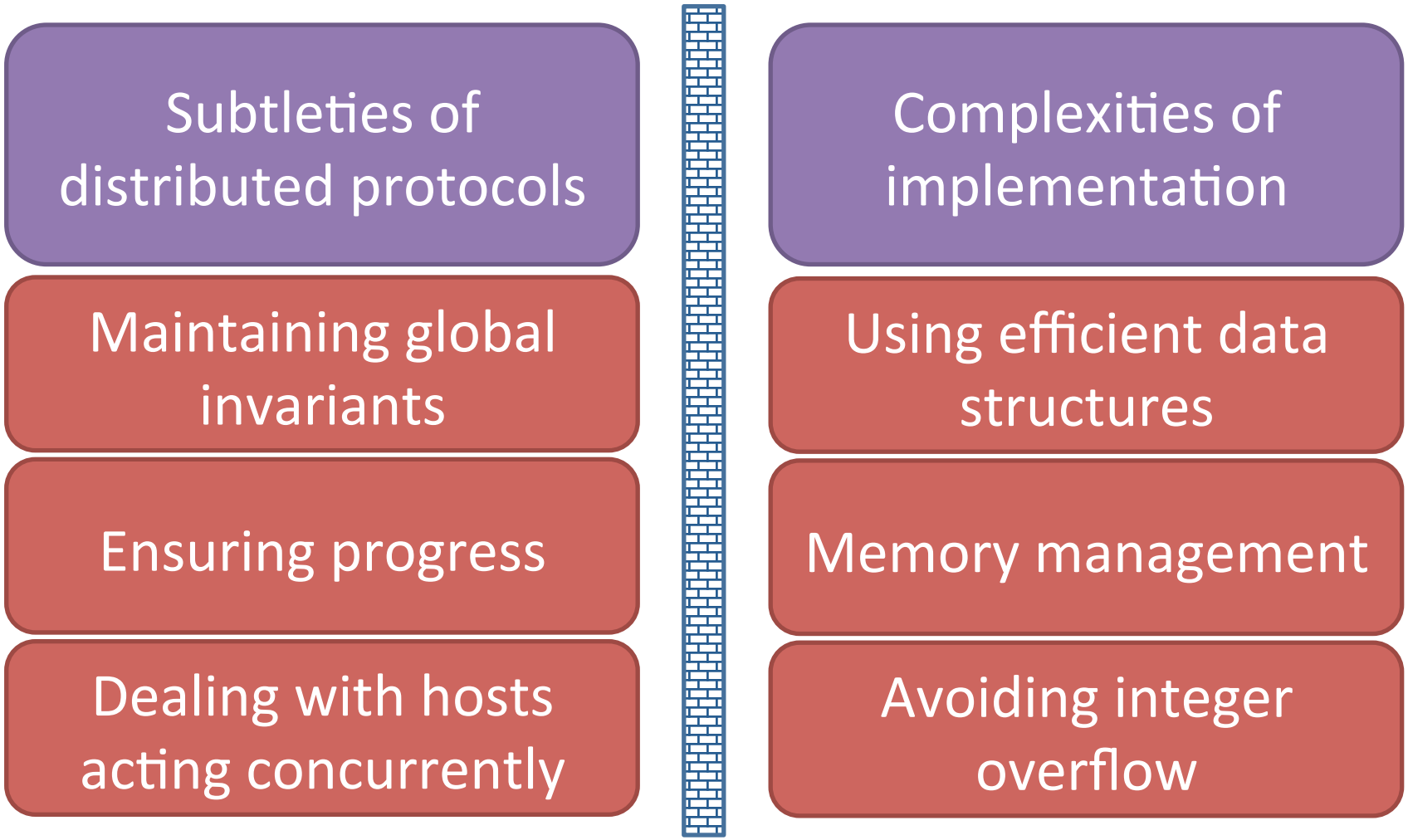
# Implementation



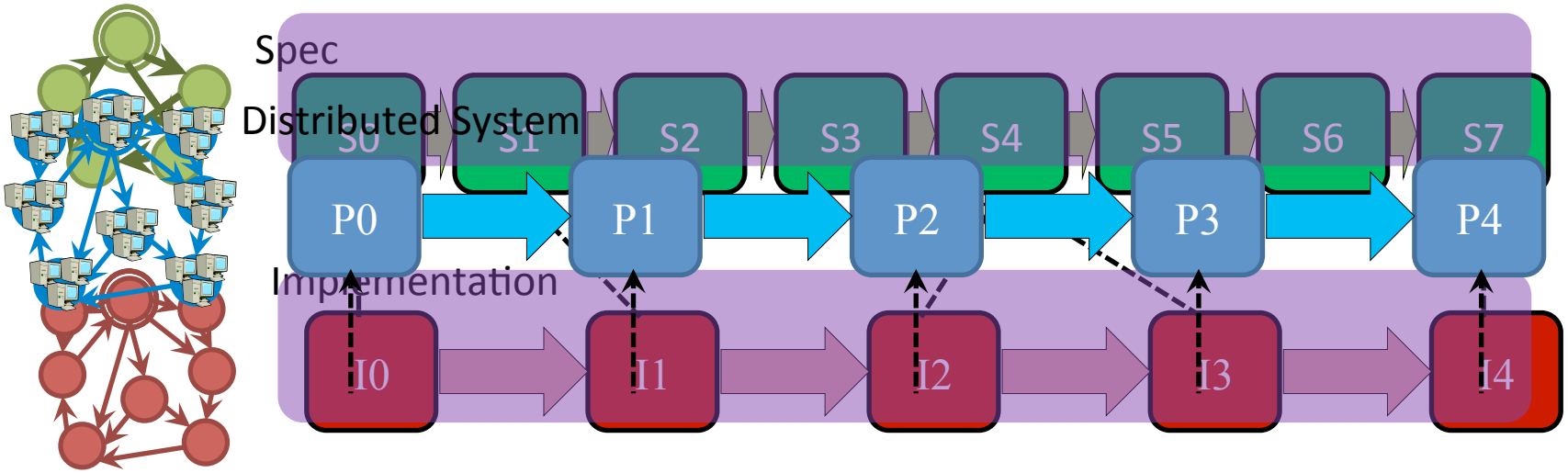
```
method Main()
{
    var s:ImplState;
    s := ImplInit();
    while (true) {
        s := EventHandler(s);
    }
}
```

Host implementation is a single-threaded event-handler loop

# Proving correctness is hard



# Two-level refinement

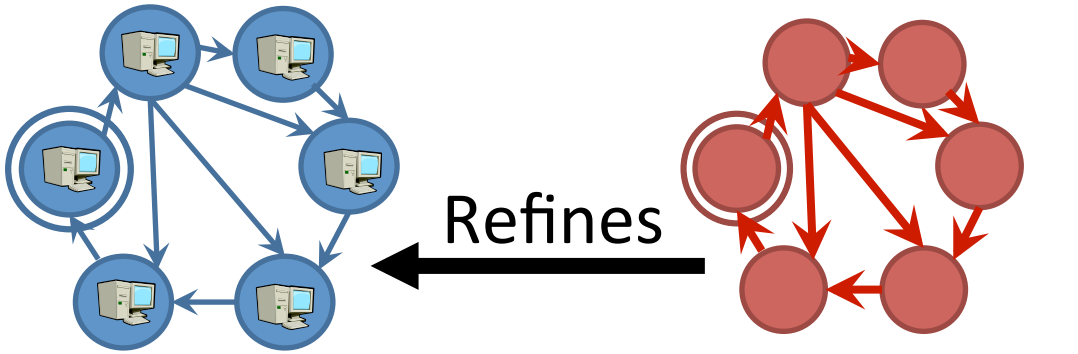


# Distributed System Layer

```
seq<int>
array<uint64>
```

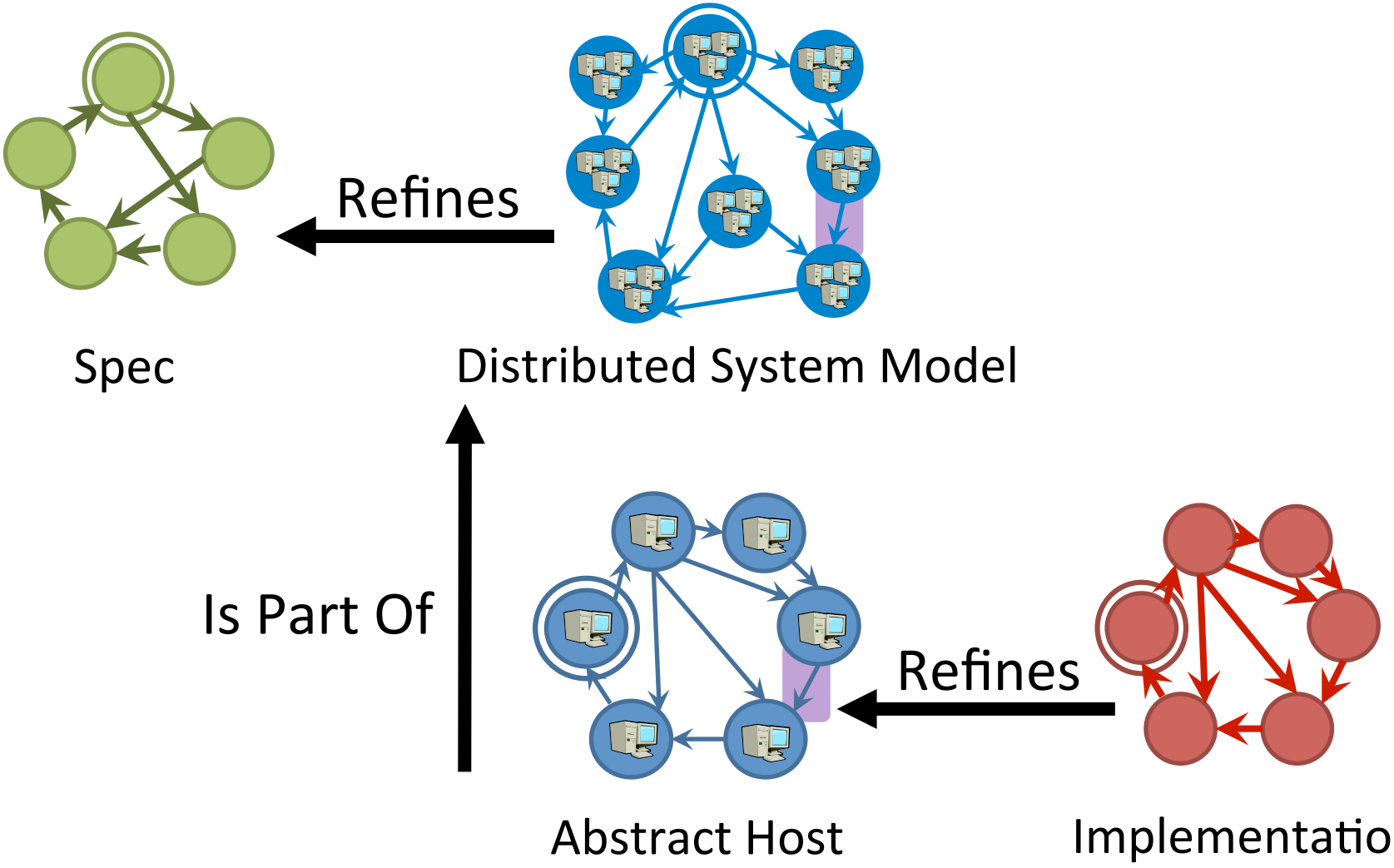
```
function ProtocolNext(s:HostState, s':HostState) : bool
method EventHandler(s:HostState) returns (s':HostState)
```

```
type Message = MessageRequest() | MessageReply() | ...
type Packet = array<byte>
```




Abstract Host Implementation

# Distributed System Layer



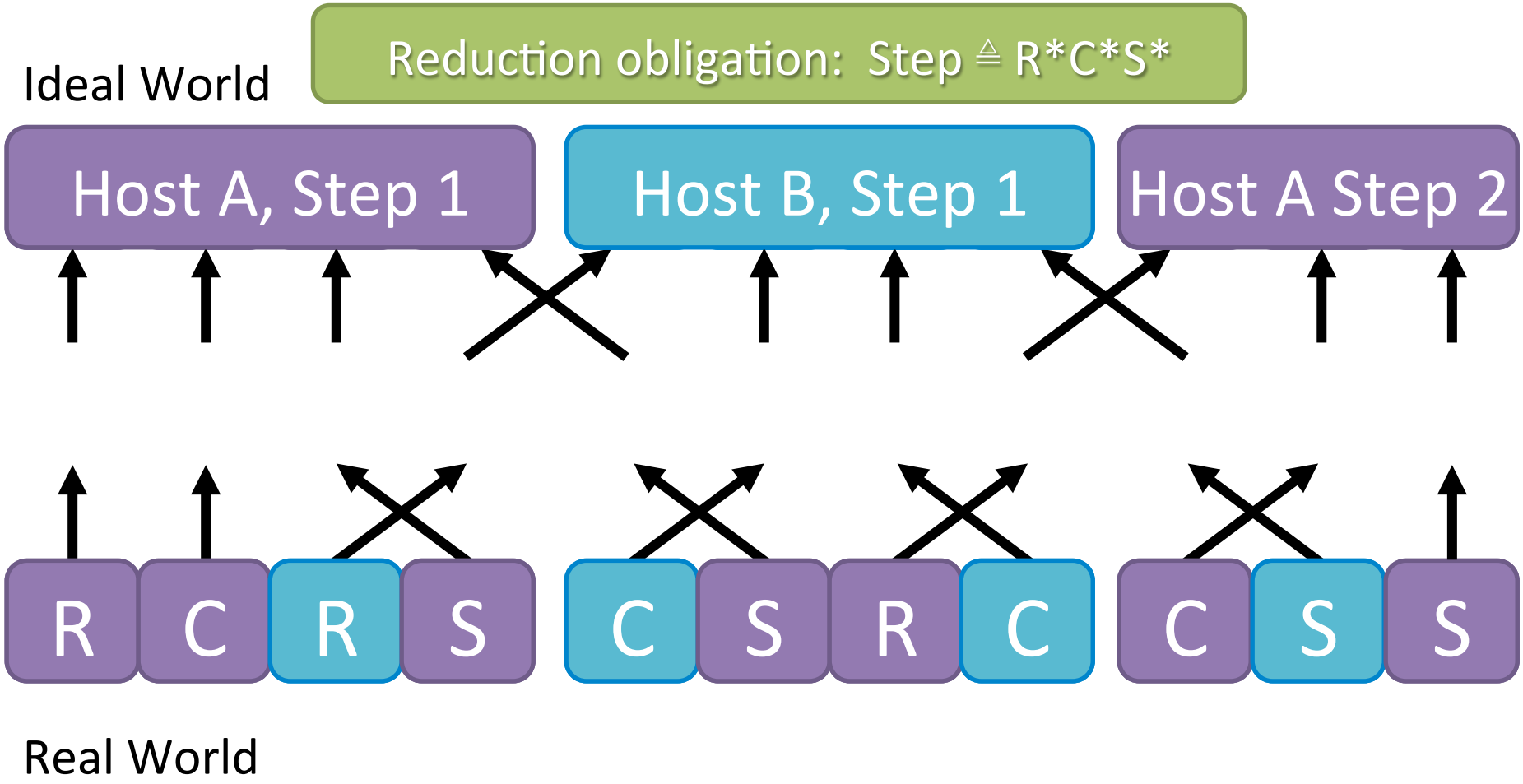
# Concurrency containment strategy

Ideal World



Real World

# Concurrency containment strategy

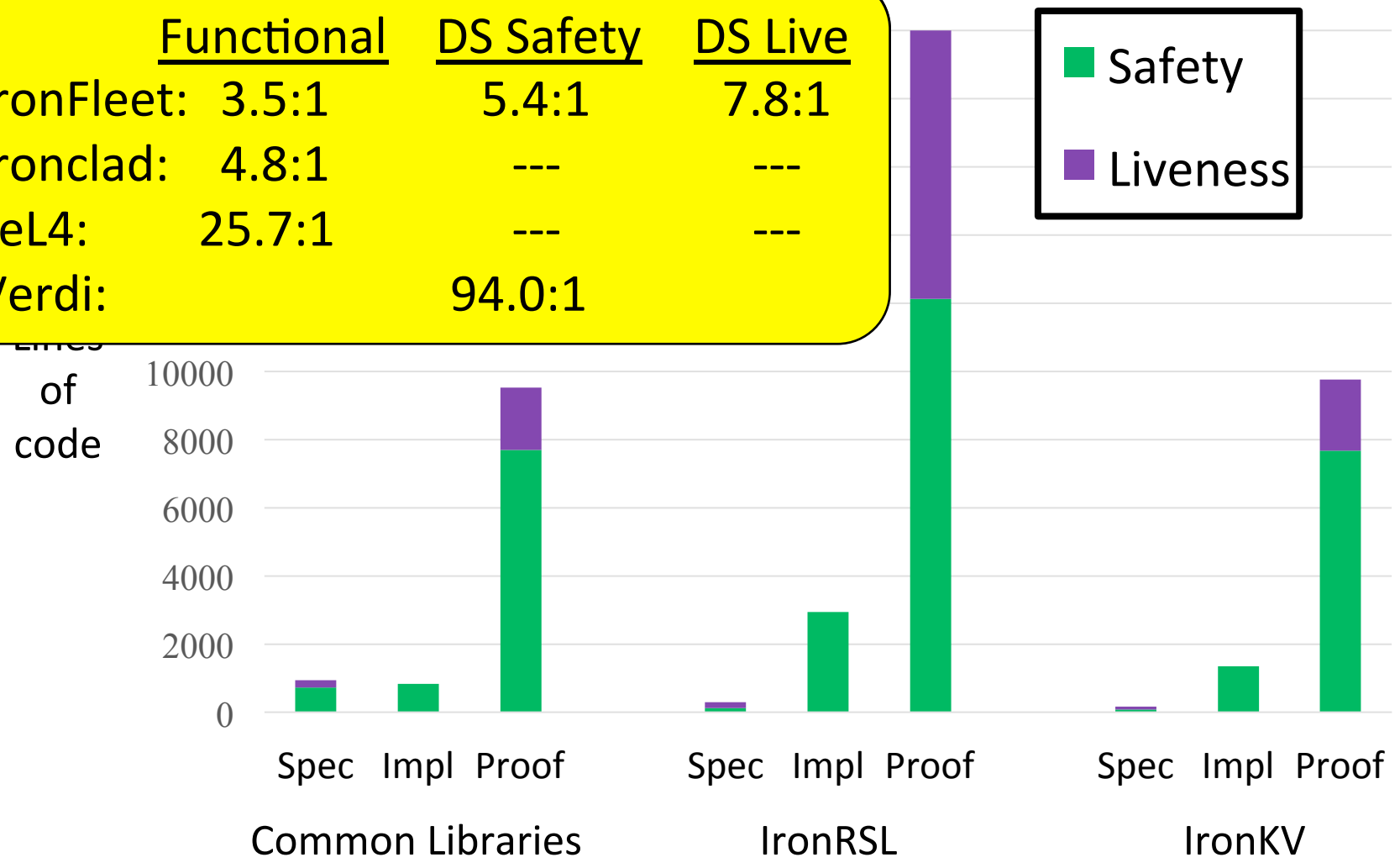


# Evaluation questions

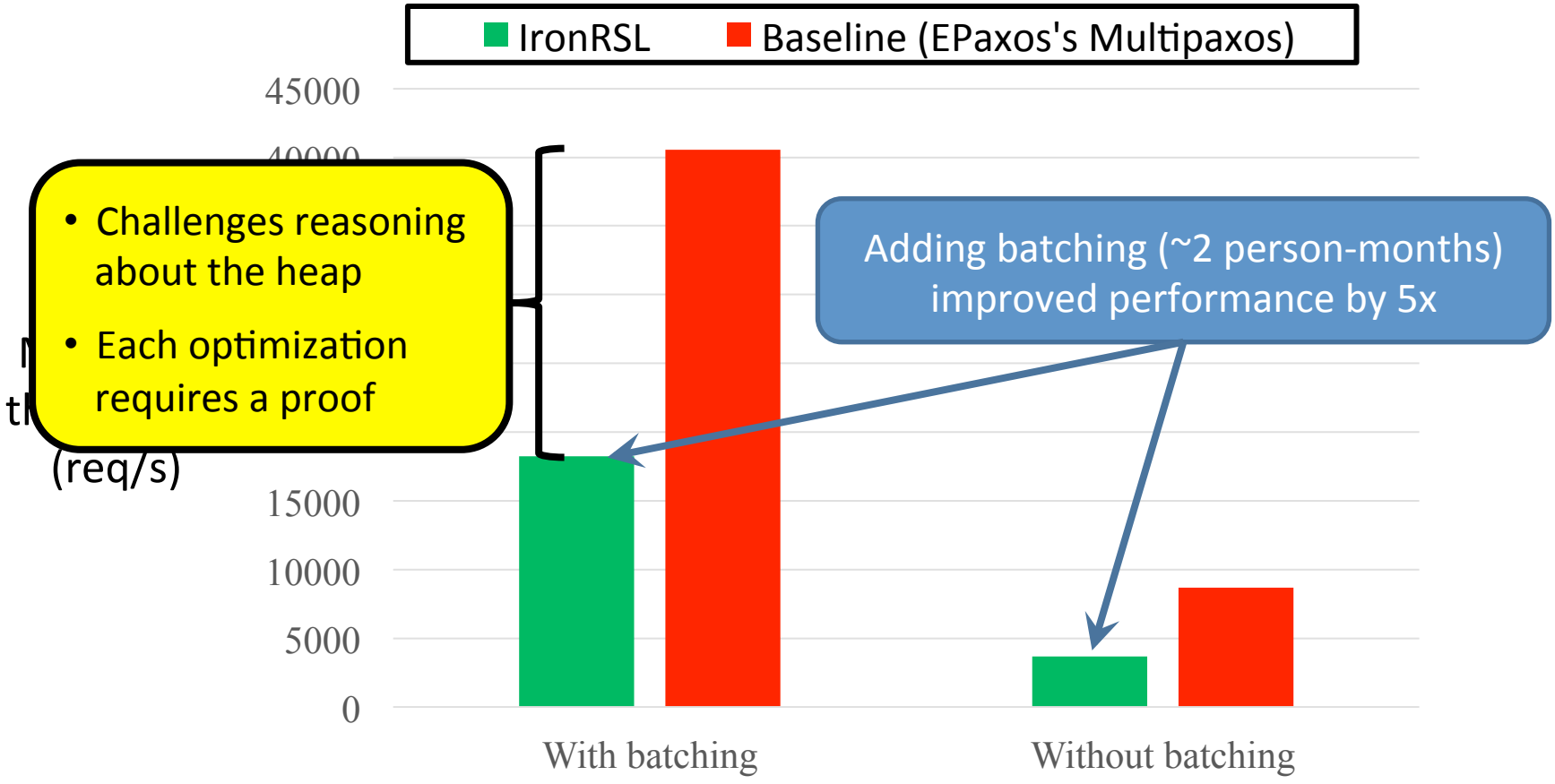
- How does verification affect development?
- How does verification affect performance?

# Developer experience: Proof burden

| | Functional | DS Safety | DS Live |
|---|---|---|---|
| IronFleet: | 3.5:1 | 5.4:1 | 7.8:1 |
| Ironclad: | 4.8:1 | --- | --- |
| seL4: | 25.7:1 | --- | --- |
| Verdi: | | 94.0:1 | |

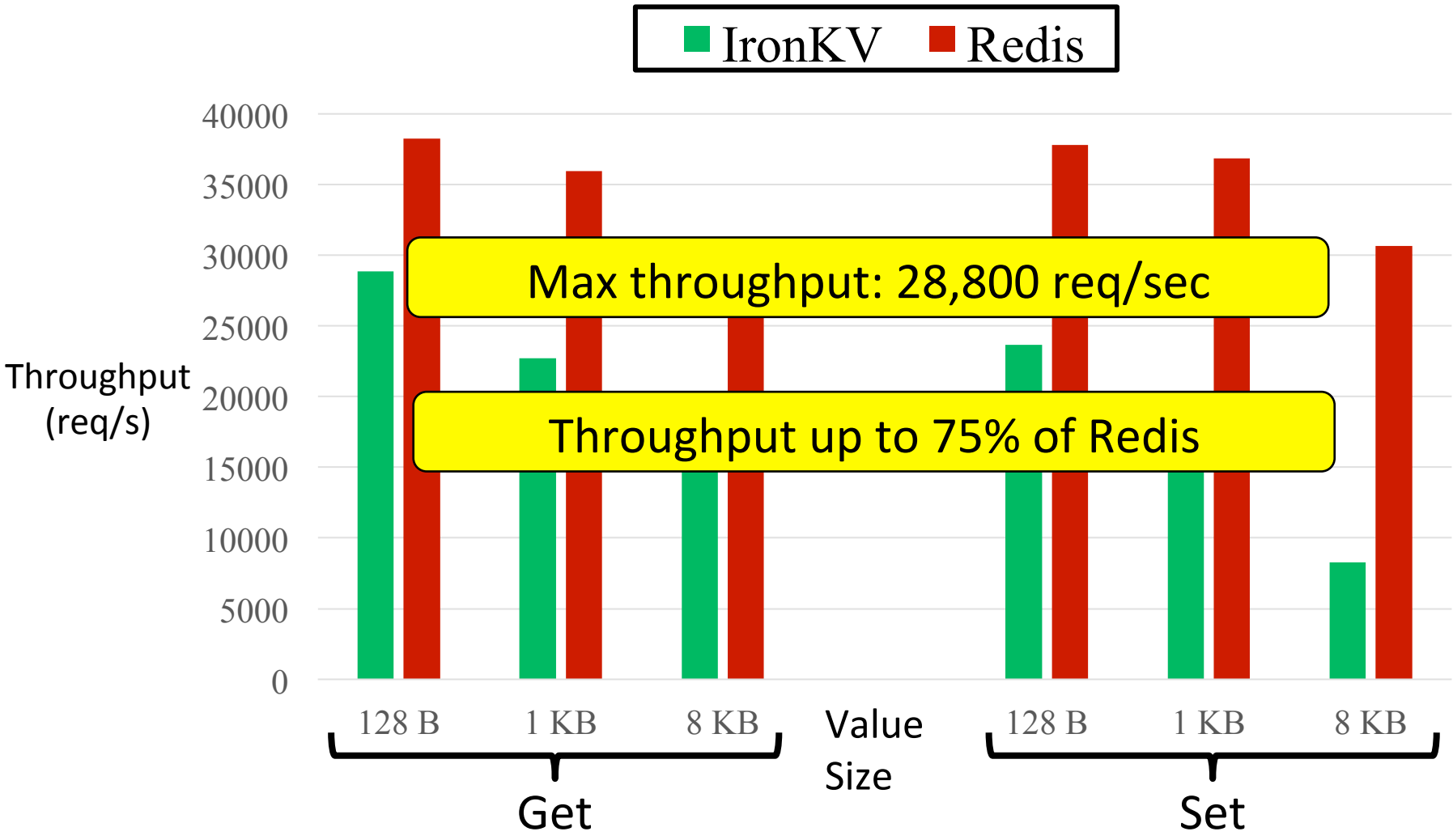


# IronRSL performance




We trade some performance for strong correctness, but no fundamental reason verified code should be slower

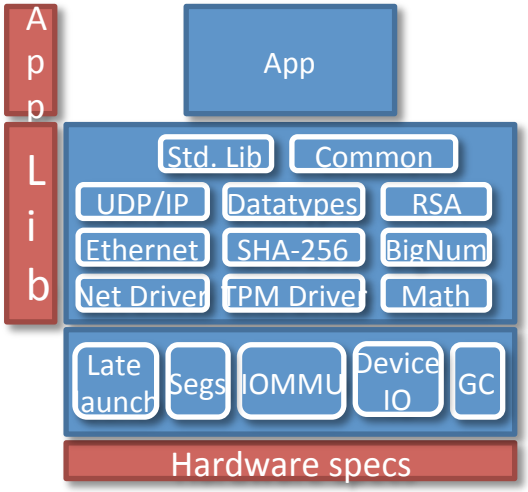
# IronKV performance



# The Ironclad Project

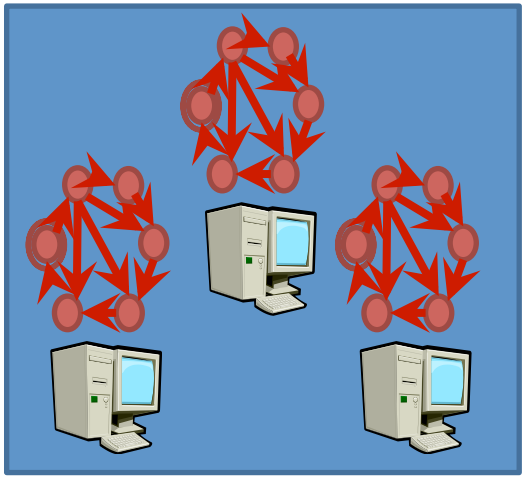
## Ironclad Apps

[OSDI 2014]

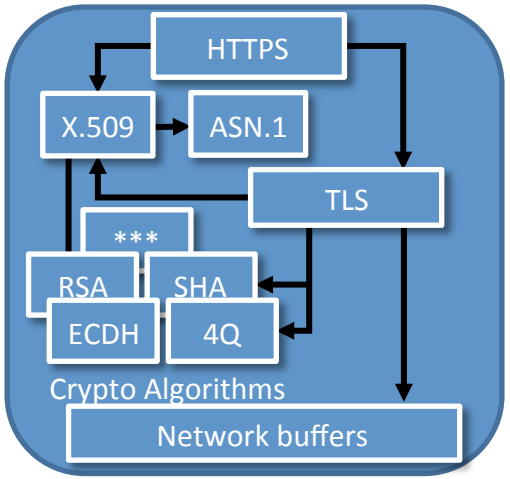



## IronFleet

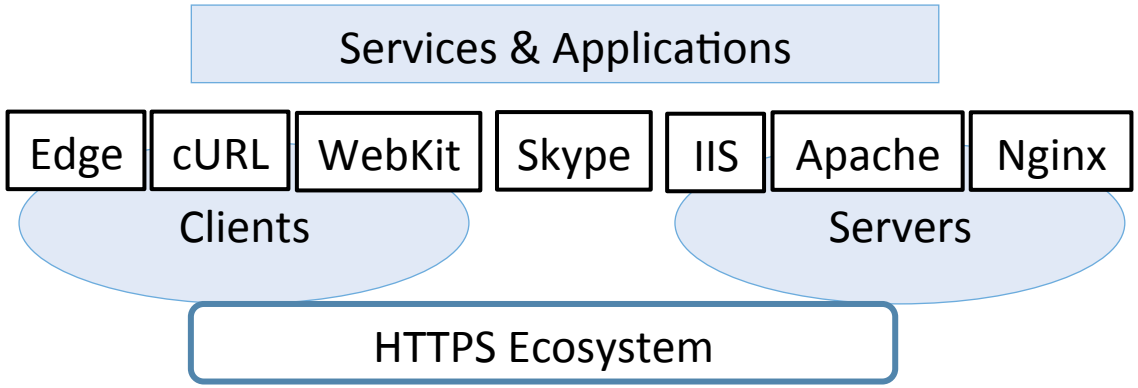
[SOSP 2015]



## Everest HTTPS

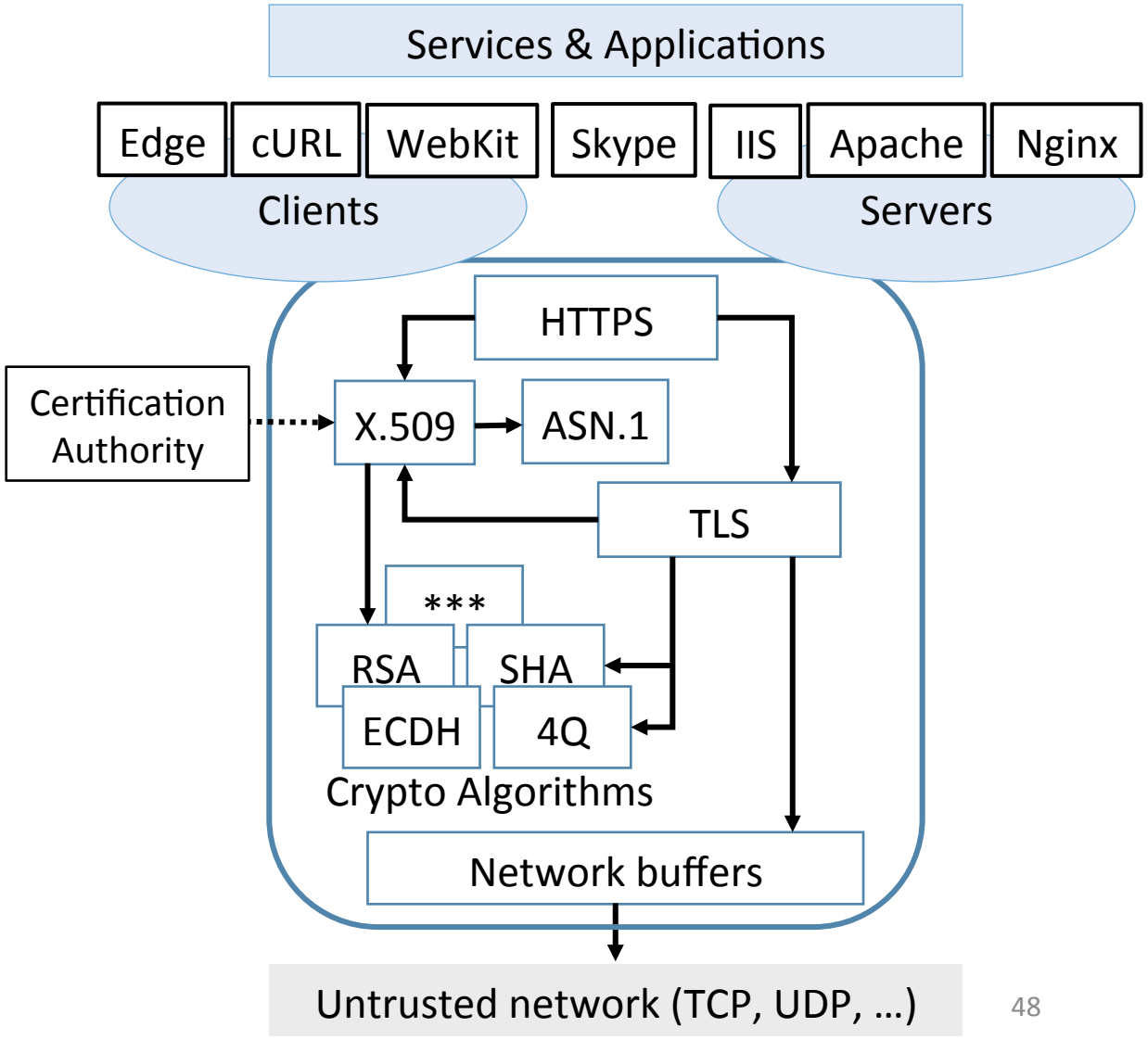


# The HTTPS Ecosystem is critical




- Most widely deployed security protocol?
  - 40% all Internet traffic (+40%/year)
- Web, cloud, email, VoIP, 802.1x, VPNs, …

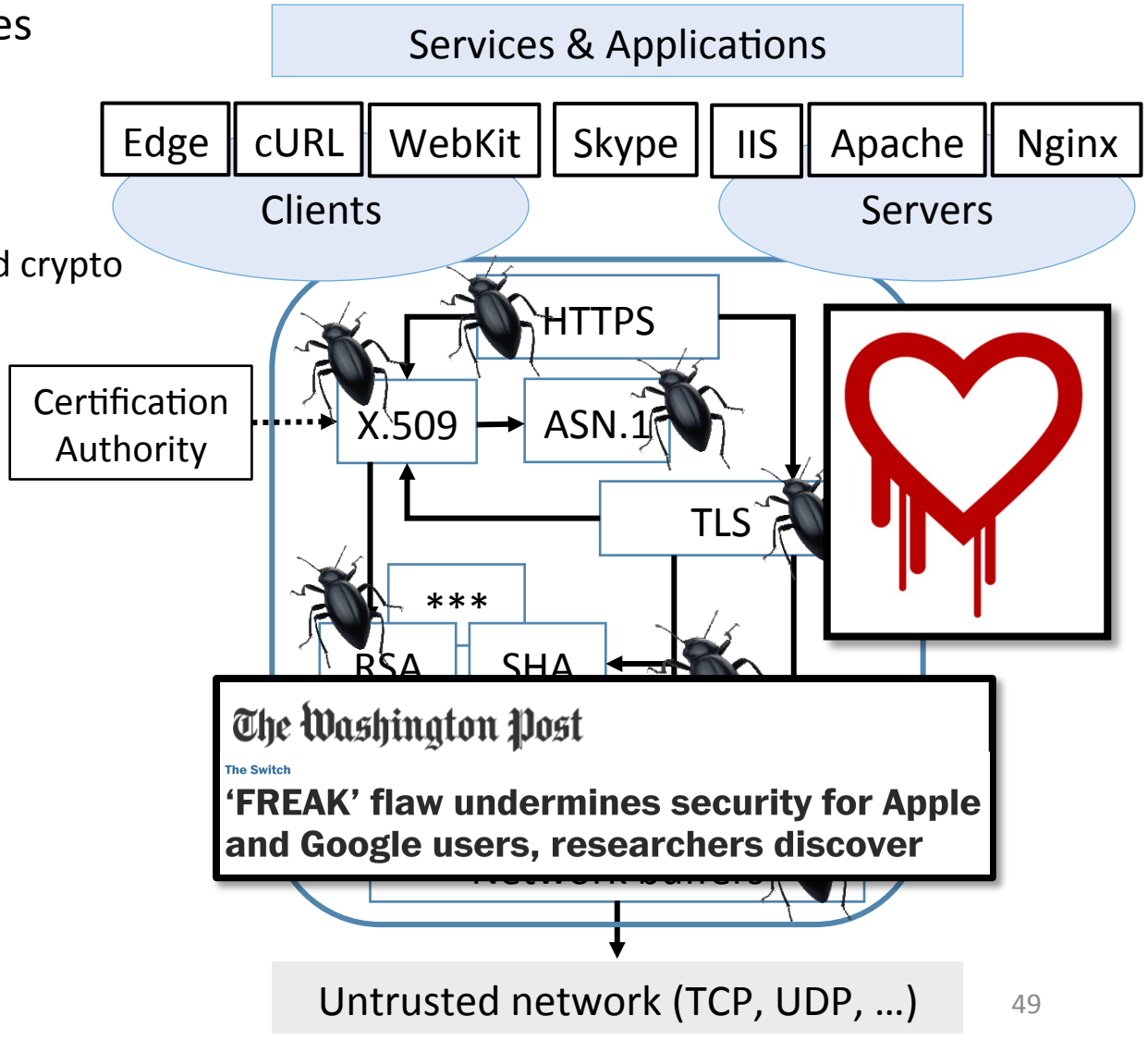
# The HTTPS Ecosystem is complex



# The HTTPS Ecosystem is buggy

- 20 years of attacks & fixes
  - Buffer overflows
  - Memory management
  - Incorrect state machines
  - Lax certificate parsing
  - Weakly or badly implemented crypto
  - Side channels
  - Error-inducing APIs
  - Flawed standards
  - …
- Many implementations
  - OpenSSL, Schannel, NSS, …

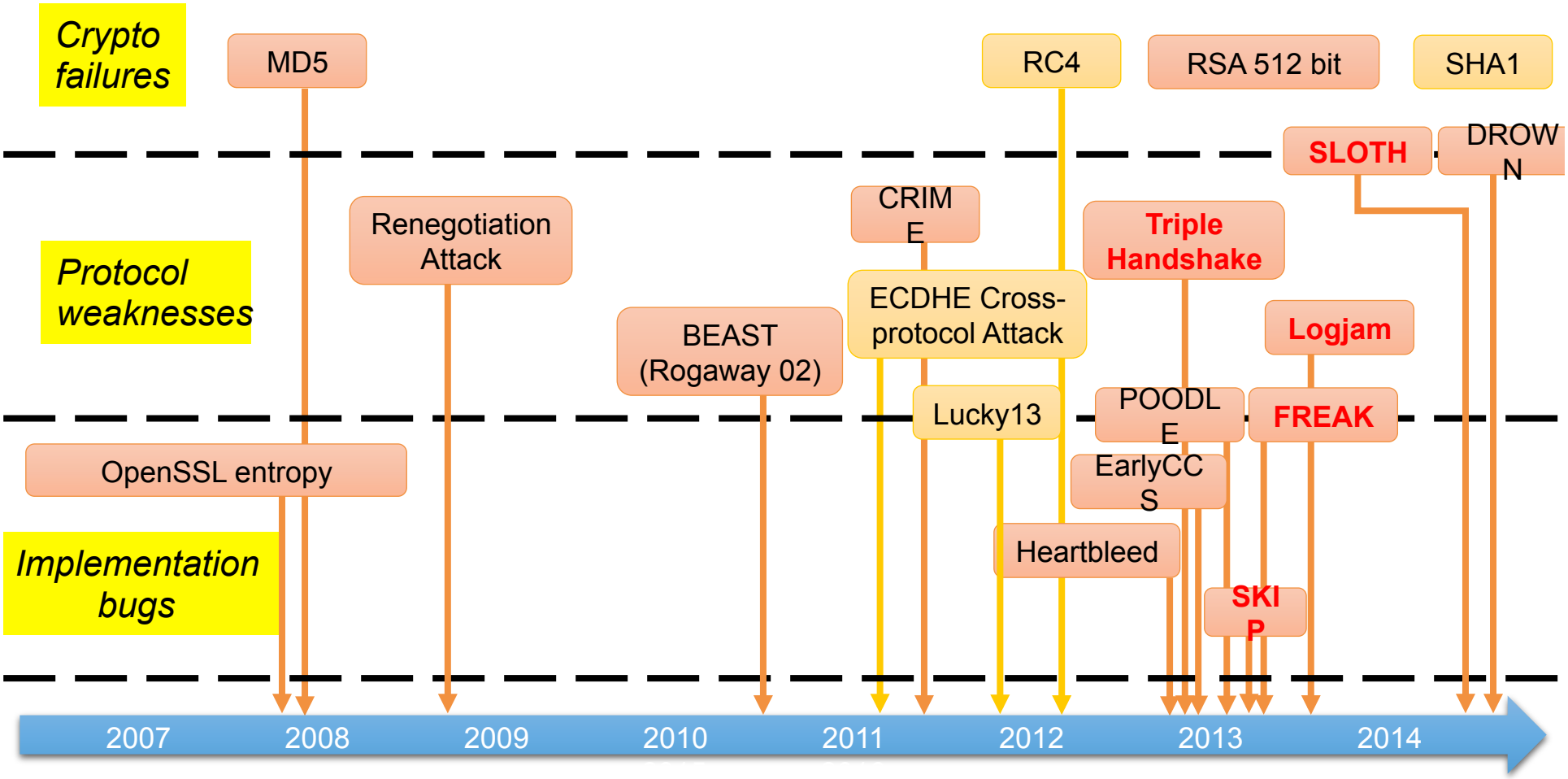
*Still patched every month!*



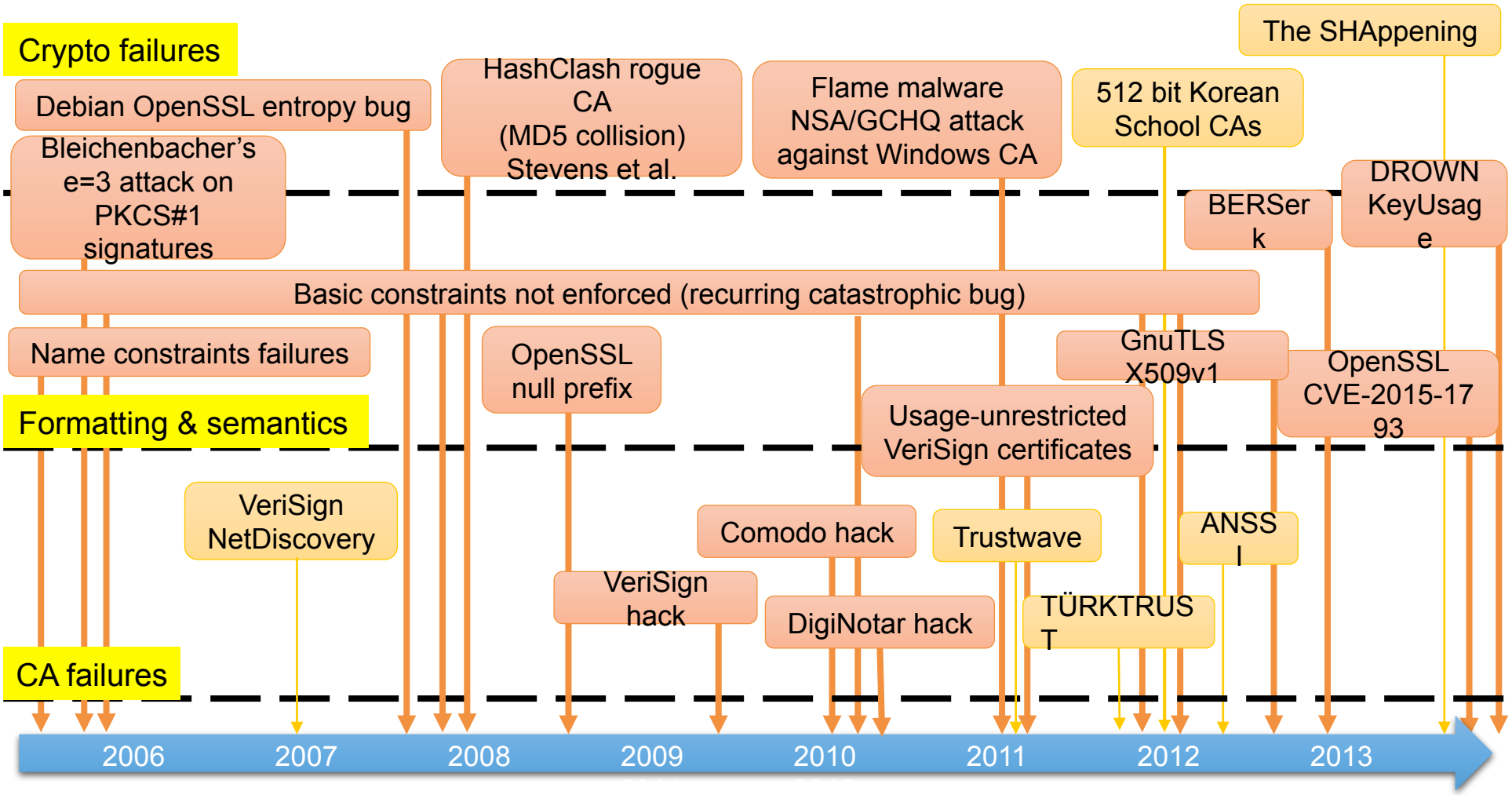
# Unsolved Attacks against HTTPS

| SSL Stripping (Marlinspike) | Cookie-based Attacks (various variants) | CRIME / BREACH (Rizzo, Duong et al.) | Virtual Host Confusion (Delignat-Lavaud) |
|---|---|---|---|
| TLS is optional in HTTP and can be disabled by an active attacker | Shared cookie database for HTTP and HTTPS can be used to mount various session fixation and login CSRF attacks. | Attackers can easily mount adaptive chosen-plaintext attacks. Encryption after compression can leak secrets through length. | HTTPS servers do not correlate transport-layer and HTTP identities, leading to origin confusion |
| Mitigated by correct use of HTTP Strict Transport Security (HSTS) | Mitigated by new binding proposals (ChannelID, Token Binding). Mitigation is not widely implemented. | Mitigated by refreshing secrets (e.g. CSRF tokens). Some protocol-specific mitigations (QUICK, HTTP2) | Mitigated by configuration of HTTPS servers with strict host rules |
| Mitigation is not widely used and vulnerability is still widespread in practice. | Extremely difficult to mitigate in all browsers with current technologies. Can be used to attack almost every website. | Ad-hoc mitigation; attack is still widespread in practice as HTTP compression remains popular. | Ad-hoc mitigation; attack is still widespread in practice. |

# High-Profile TLS Attacks

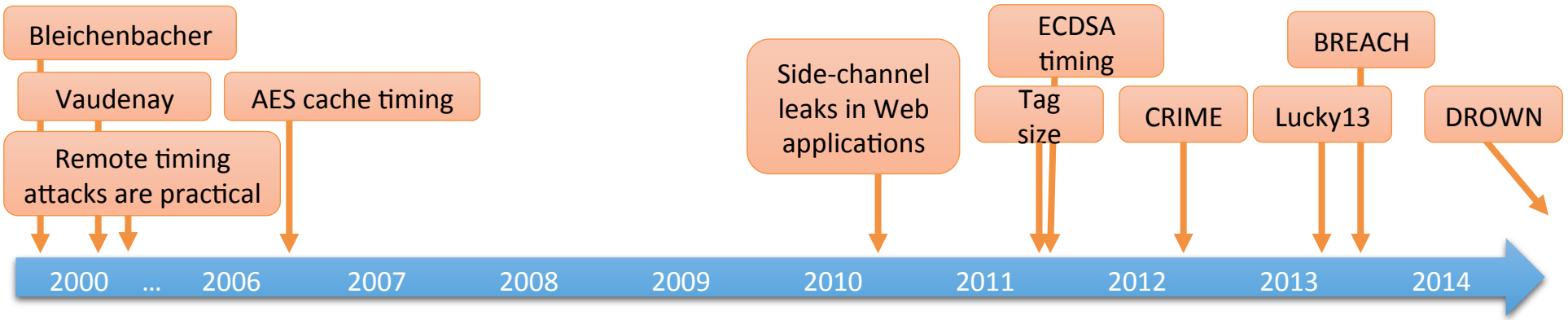


# A Timeline of Recent PKI Failures



# Side Channel Challenge (Attacks)

| Protocol-level side channels | Traffic analysis | Timing attacks against cryptographic primitives | Memory & Cache |
|---|---|---|---|
| TLS messages may reveal information about the internal protocol state or the application data | Combined analysis of the time and length distributions of packets leaks information about the application | A remote attacker may learn information about crypto secrets by timing execution time for various inputs | Memory access patterns may expose secrets, in particular because caching may expose sensitive data (e.g. by timing) |
| • Hello message contents (e.g. time in nonces, SNI)<br>• Alerts (e.g. decryption vs. padding alerts)<br>• Record headers | • CRIME/BREACH (adaptive chosen plaintext attack)<br>• User tracking<br>• Auto-complete input theft | • Bleichenbacher attacks against PKCS#1 decryption and signatures<br>• Timing attacks against RC4 (Lucky 13) | • OpenSSL key recovery in virtual machines<br>• Cache timing attacks against AES |



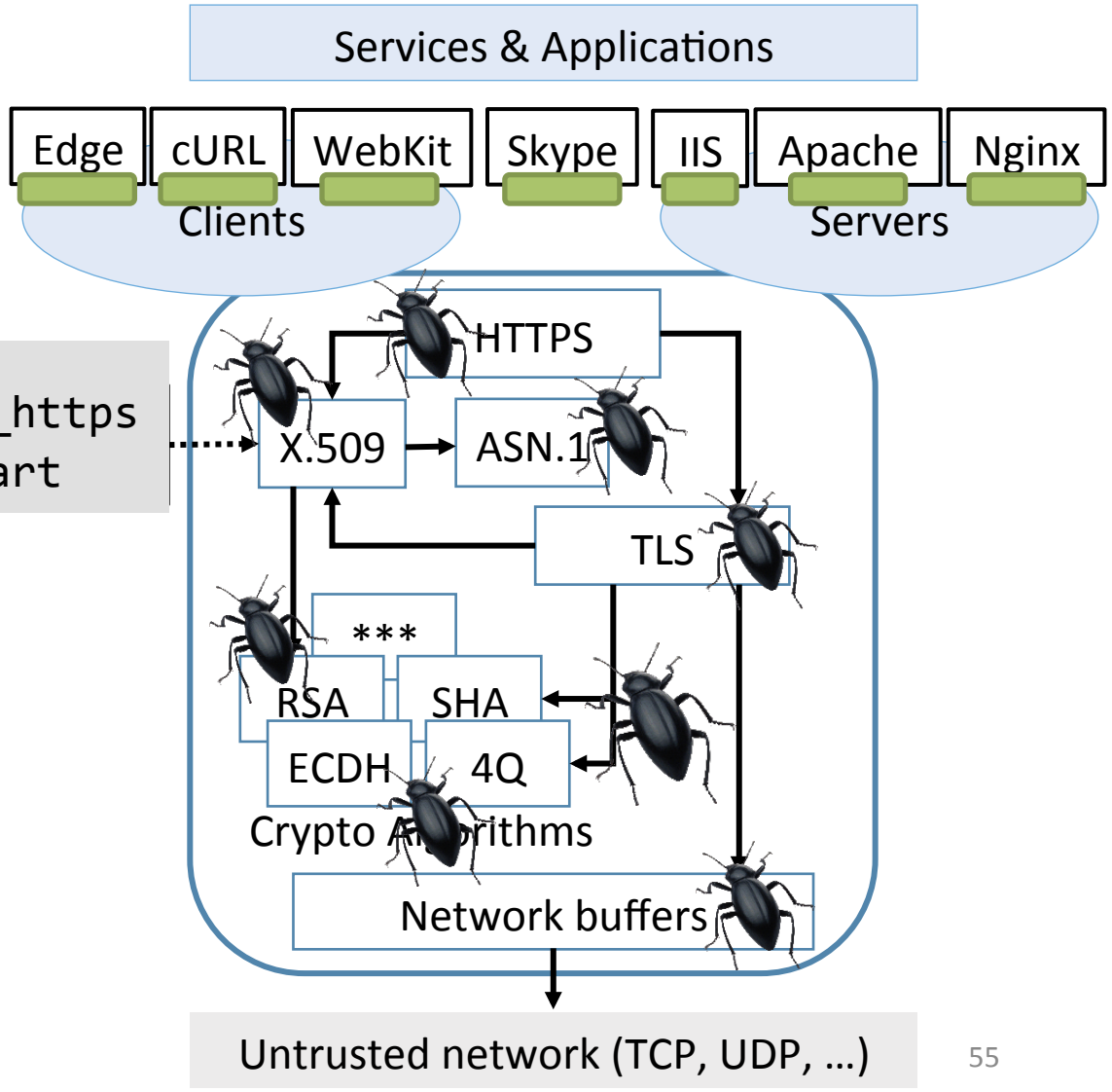
# *Everest:*

## *Deploying Verified-Secure Implementations in the HTTPS Ecosystem*

# Everest Goals

- Fully verified replacement
- Widespread deployment
- Trustworthy, usable tools

```
$ apt-get install verified_https
$ /                              tart
```

# Research Questions

- How do we decide whether new protocols are secure?
  - Especially when interoperating with insecure protocols
- Can we make verified systems as fast as unverified?
- How do we handle advanced threats?
  - Ex: Side channels
- Why should we trust automated verification tools?
- How can verification be more accessible?
  - Especially to non-experts in verification

# Summary

- Ironclad Apps guarantee end-to-end security to remote parties: Every instruction meets the app's security spec
- IronFleet extends these techniques to prove the safety and liveness of distributed systems
- Everest will showcase the power of verification and its applicability to real-world security problems
- Verification of systems code is possible, and we're scaling it to even larger more complex systems

https://github.com/Microsoft/Ironclad

Thank you!

parno@cmu.edu