Verifying Security Protocols End-to-End with Owl

Joshua Gancher
Carnegie Mellon University

Owl

new language for building security protocols

- TLS, WireGuard, Signal, …
- Supports **verified implementations**

Owl

new language for building security protocols

- TLS, WireGuard, Signal, …
- Supports **verified implementations**

Our vision:
formally verified, drop-in
replacements of protocol implementations

Owl

new language for building security protocols

- TLS, WireGuard, Signal, …
- Supports **verified implementations**

Our vision:
formally verified, drop-in
replacements of protocol implementations

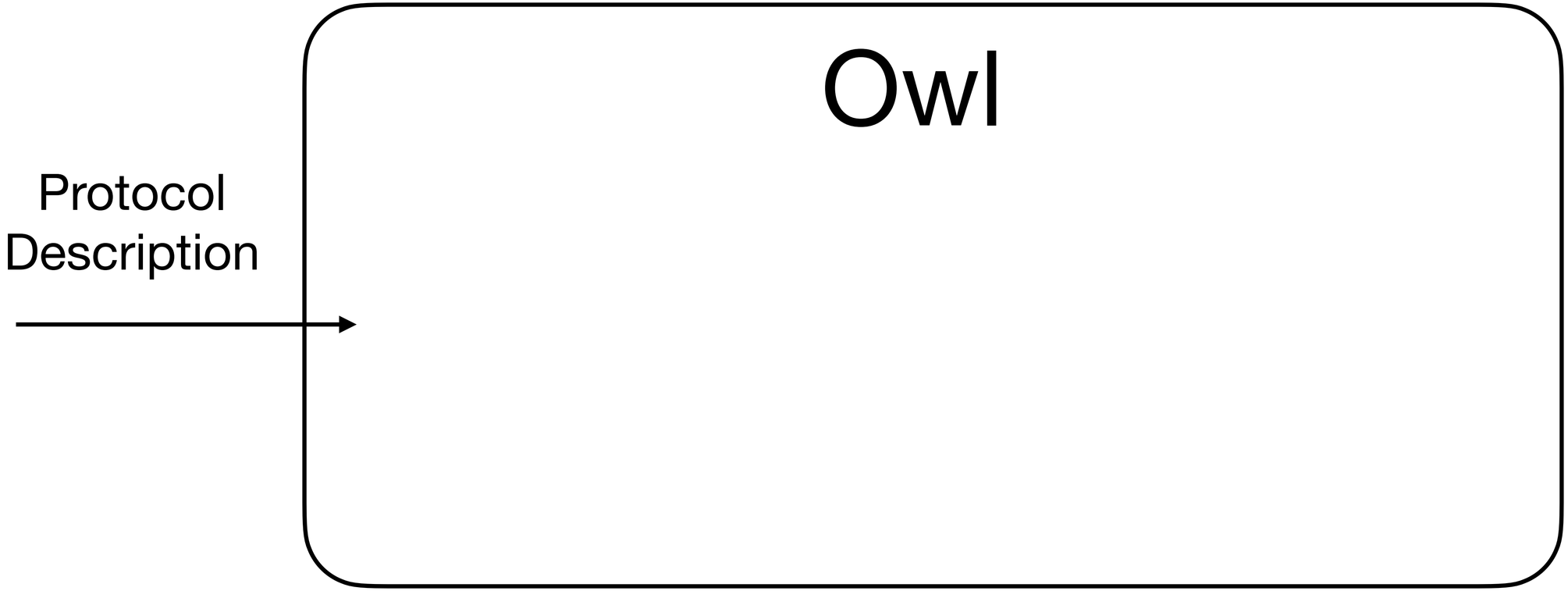
Owl

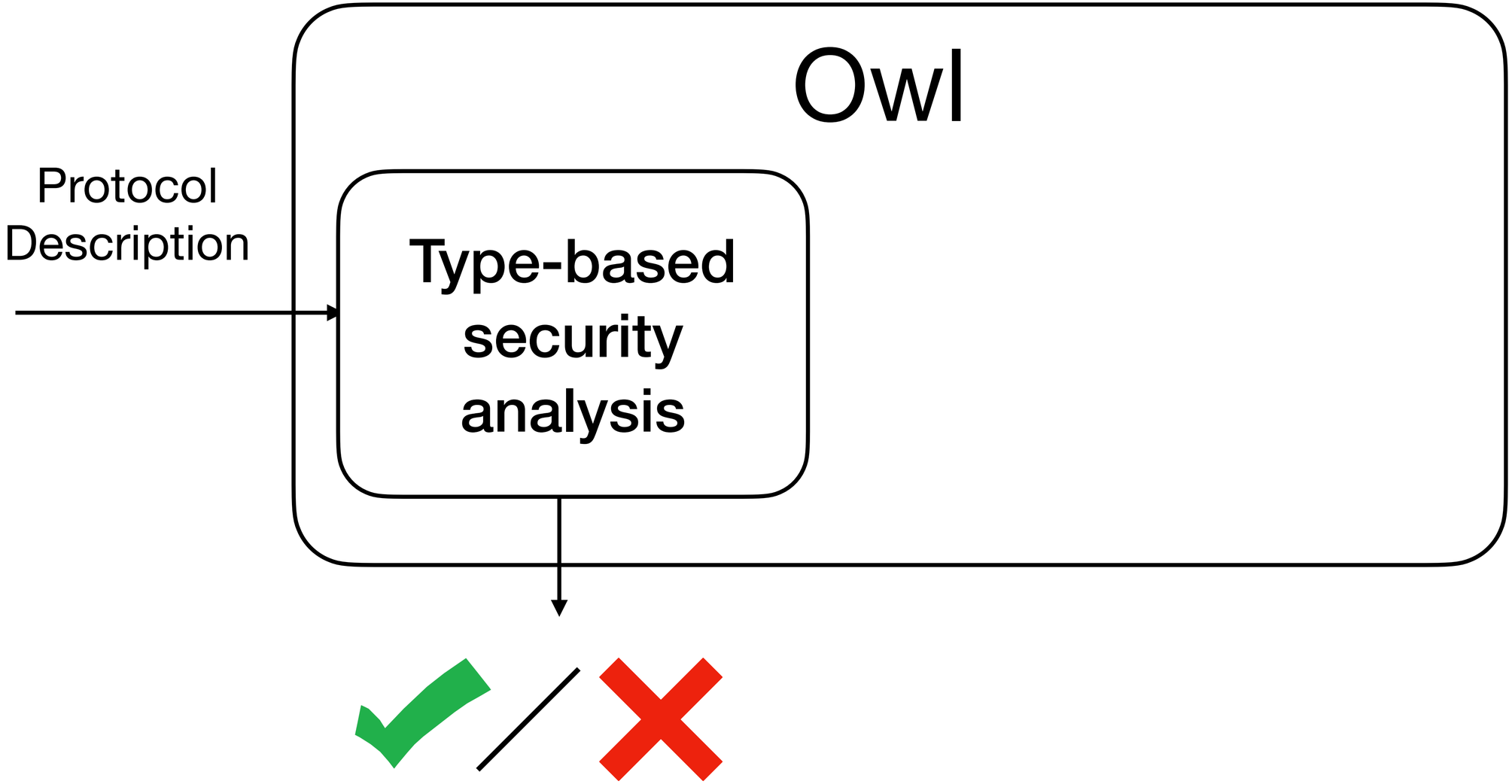
new language for building security protocols

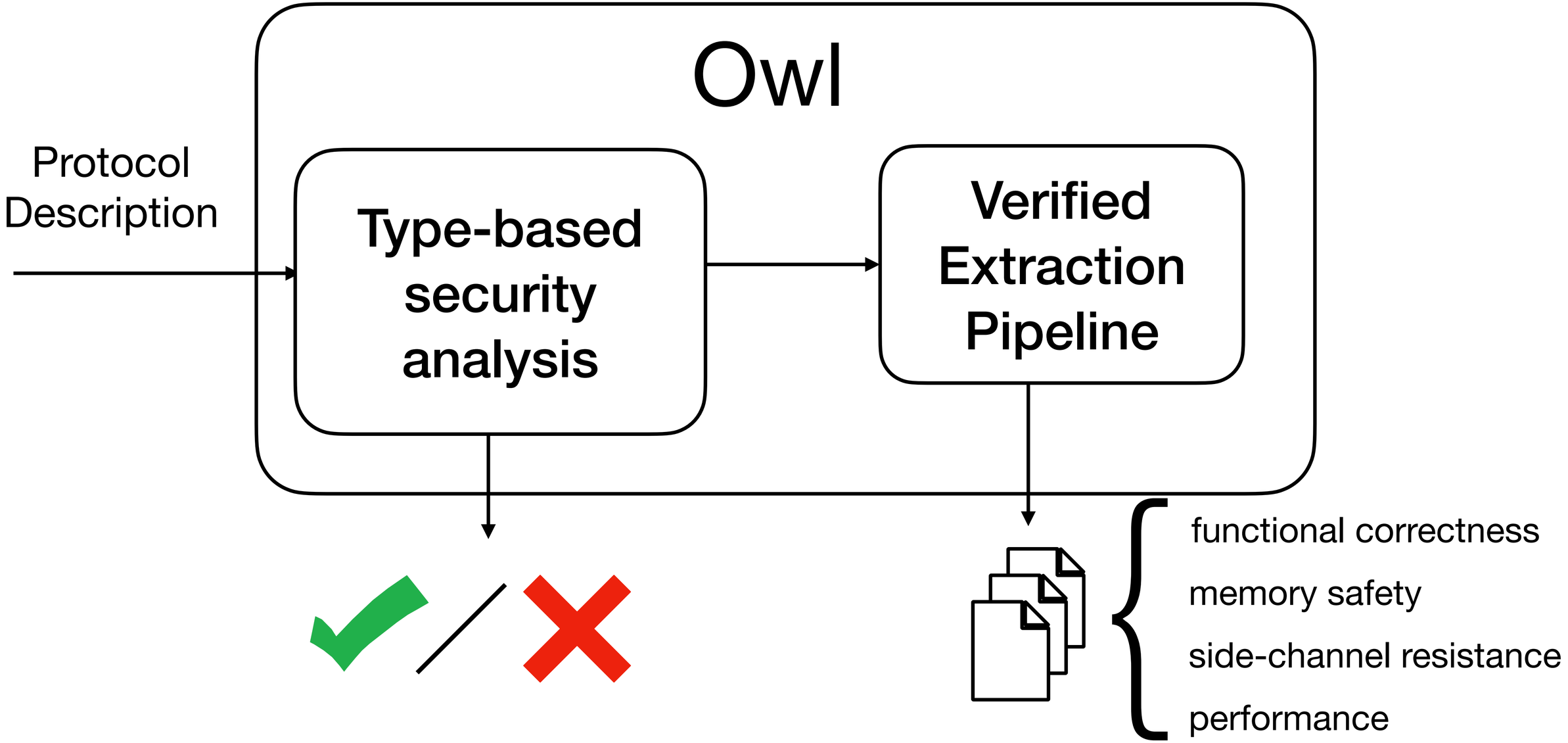
- TLS, WireGuard, Signal, …
- Supports **verified implementations**

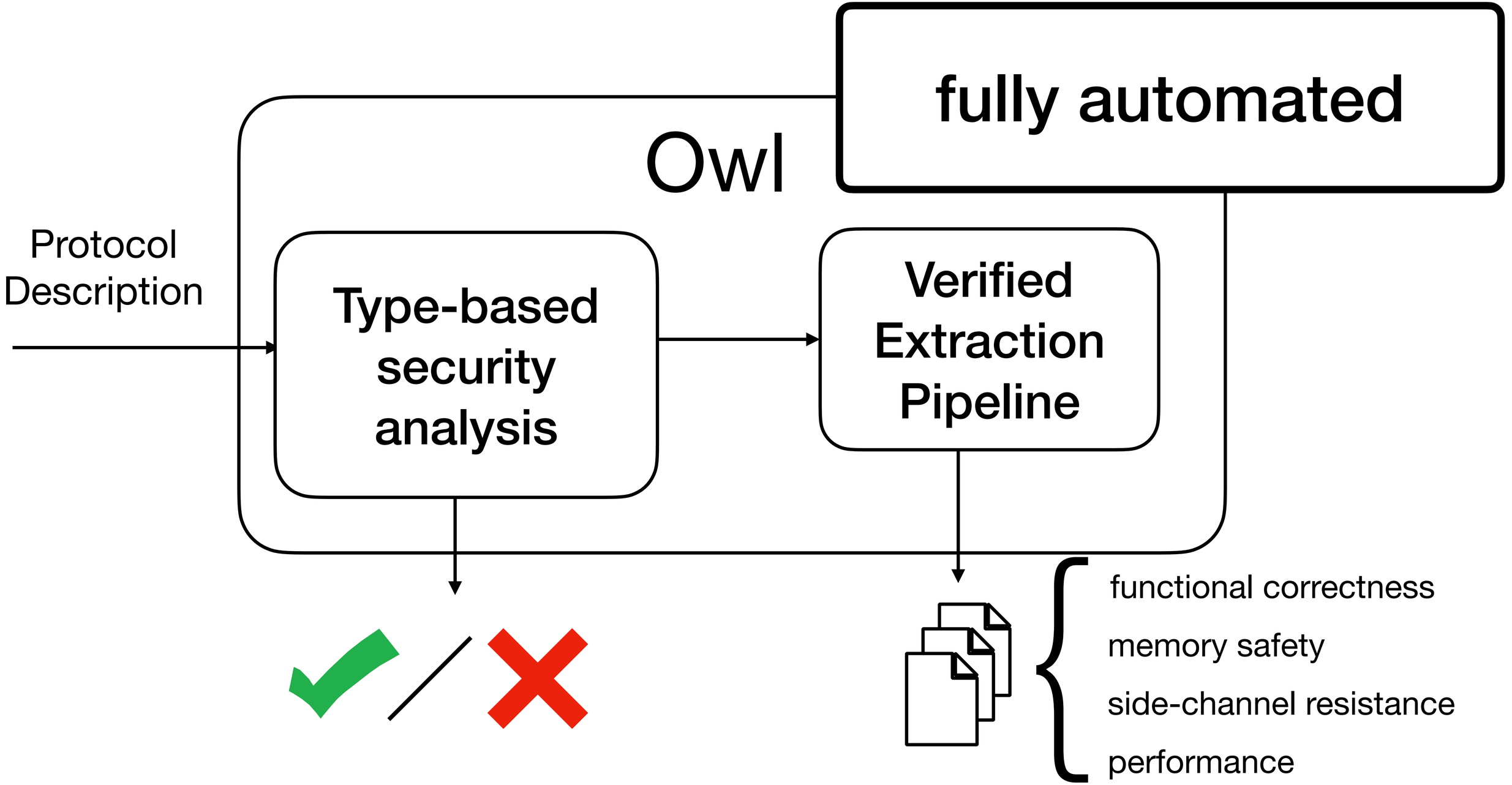
Our vision:
formally verified, drop-in
replacements of protocol implementations

- functional correctness
- memory safety
- secure designs


Owl
Protocol
Description

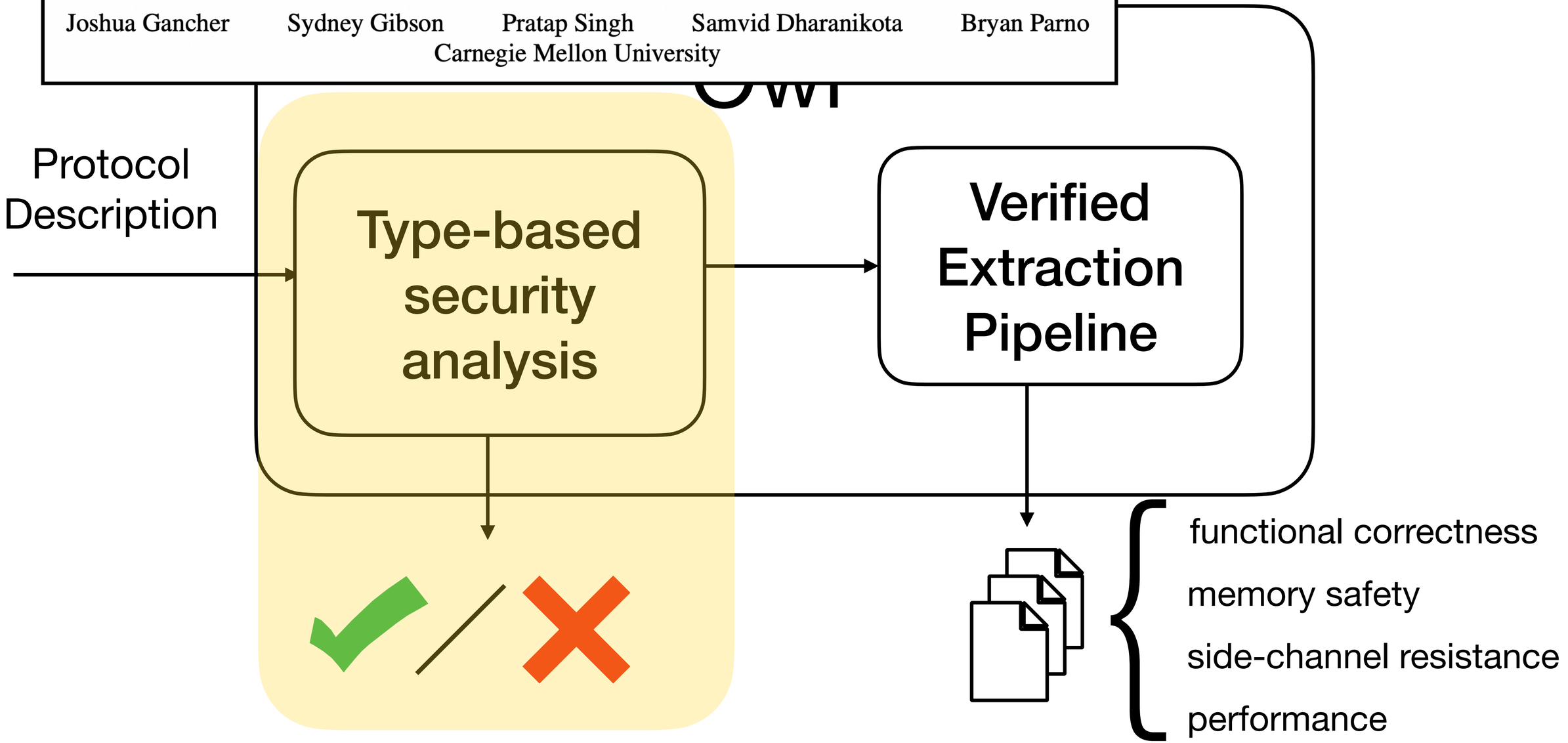

Owl
Protocol Description
Type-based security analysis


Owl
Protocol Description
Type-based security analysis
Verified Extraction Pipeline
functional correctness
memory safety
side-channel resistance
performance


fully automated
Owl
Protocol Description
Type-based security analysis
Verified Extraction Pipeline
functional correctness
memory safety
side-channel resistance
performance

OWL: Compositional Verification of Security Protocols via an Information-Flow Type System

Joshua Gancher Sydney Gibson Pratap Singh Samvid Dharanikota Bryan Parno
Carnegie Mellon University



Symbolic Security

Computational Security

Symbolic Security

Crypto modeled by
abstract terms,
equations on functions

Computational Security

Symbolic Security

Crypto modeled by
abstract terms,
equations on functions

```
functions: enc/2, dec/2
equations: dec(enc(m, k), k) = m
```

Computational Security

Symbolic Security

Crypto modeled by
abstract terms,
equations on functions

```
functions: enc/2, dec/2
equations: dec(enc(m, k), k) = m
```

attacker can only use
specified functions,
equations

Computational Security

Symbolic Security

Crypto modeled by
abstract terms,
equations on functions

```
functions: enc/2, dec/2
equations: dec(enc(m, k), k) = m
```

attacker can only use
specified functions,
equations

Computational Security

Crypto is given by algorithms on bytes

Cryptography specified by
security properties:
secrecy of messages,
unforgeability of ciphertexts,
…

Symbolic Security

Crypto modeled by
abstract terms,
equations on functions

```
functions: enc/2, dec/2
equations: dec(enc(m, k), k) = m
```

attacker can only use
specified functions,
equations

Computational Security

Crypto is given by algorithms on bytes

Cryptography specified by
security properties:
secrecy of messages,
unforgeability of ciphertexts,
…

Strong attacker model;
closer to implementations

Types for Security Protocols

Types for Security Protocols

K : enckey for T
$m : T$

$\Longrightarrow$ $\mathrm{enc}(K, m)$: public

Types for Security Protocols

K : enckey for T
$m : T$

$\Longrightarrow$ enc(K, m) : public

security via type checking: $\vdash P \Longrightarrow P$ secure

Types for Security Protocols

K : enckey for T
$m : T$

$\Longrightarrow$ enc(K, m) : public

security via type checking: $\vdash P \Longrightarrow P$ secure

one time proof effort

Types for Security Protocols

K : enckey for T
$m : T$

$\Longrightarrow$ $\text{enc}(K, m)$: public

security via type checking: $\vdash P \Longrightarrow P$ secure

one time proof effort

developer only
uses type system!

Types for Security Protocols

K : enckey for T
$m : T$

$\Longrightarrow$ enc(K, m) : public

security via type checking: $\vdash P \Longrightarrow P$ secure

one time proof effort

developer only **uses** type system!

$\longrightarrow$

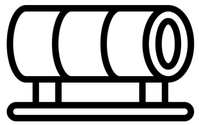
automatic, modular proof effort

Protocol-Level Modularity

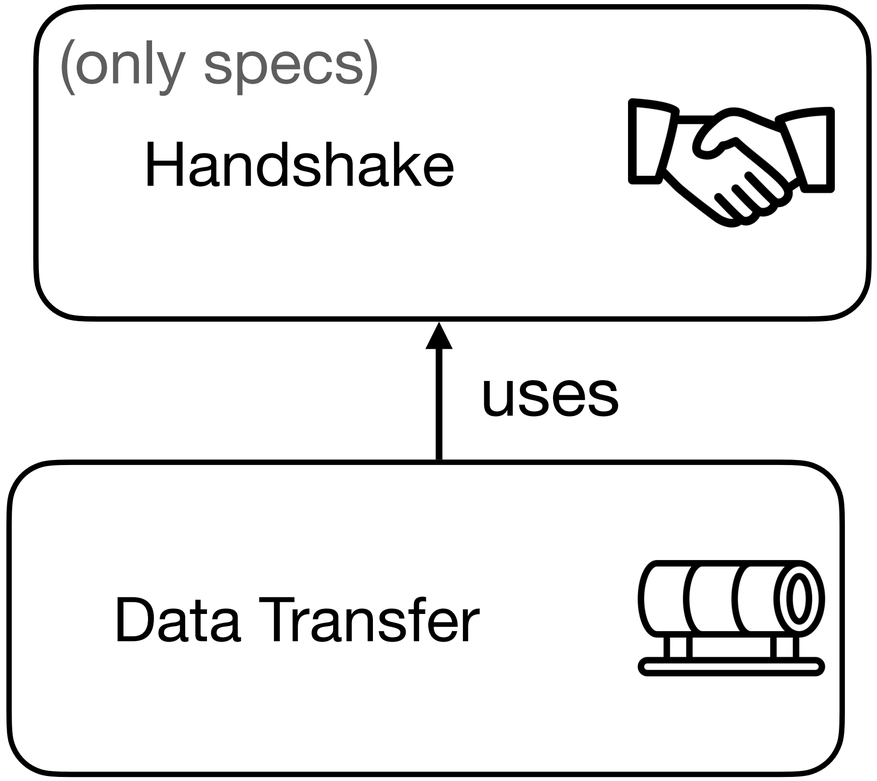
Handshake

+

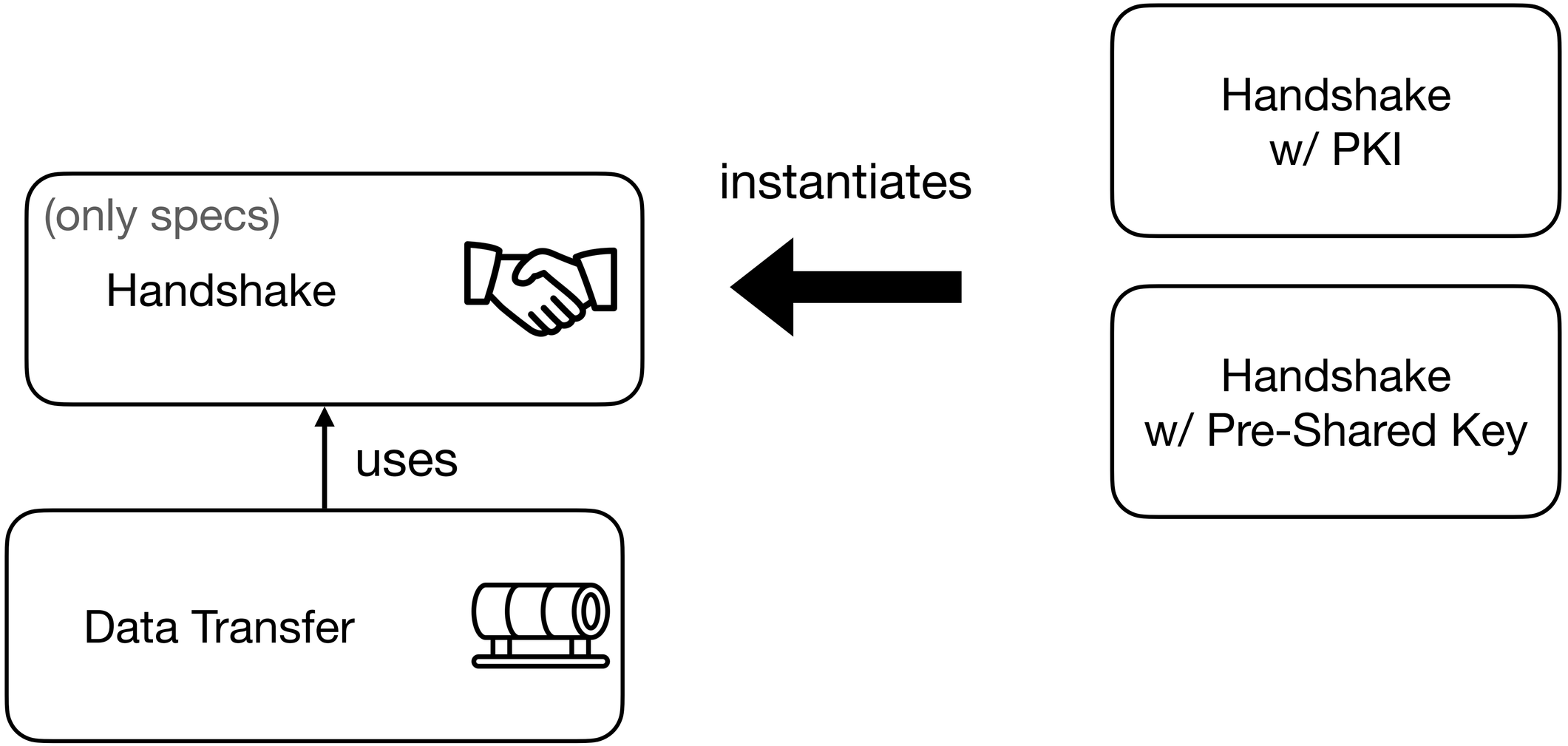
Data Transfer

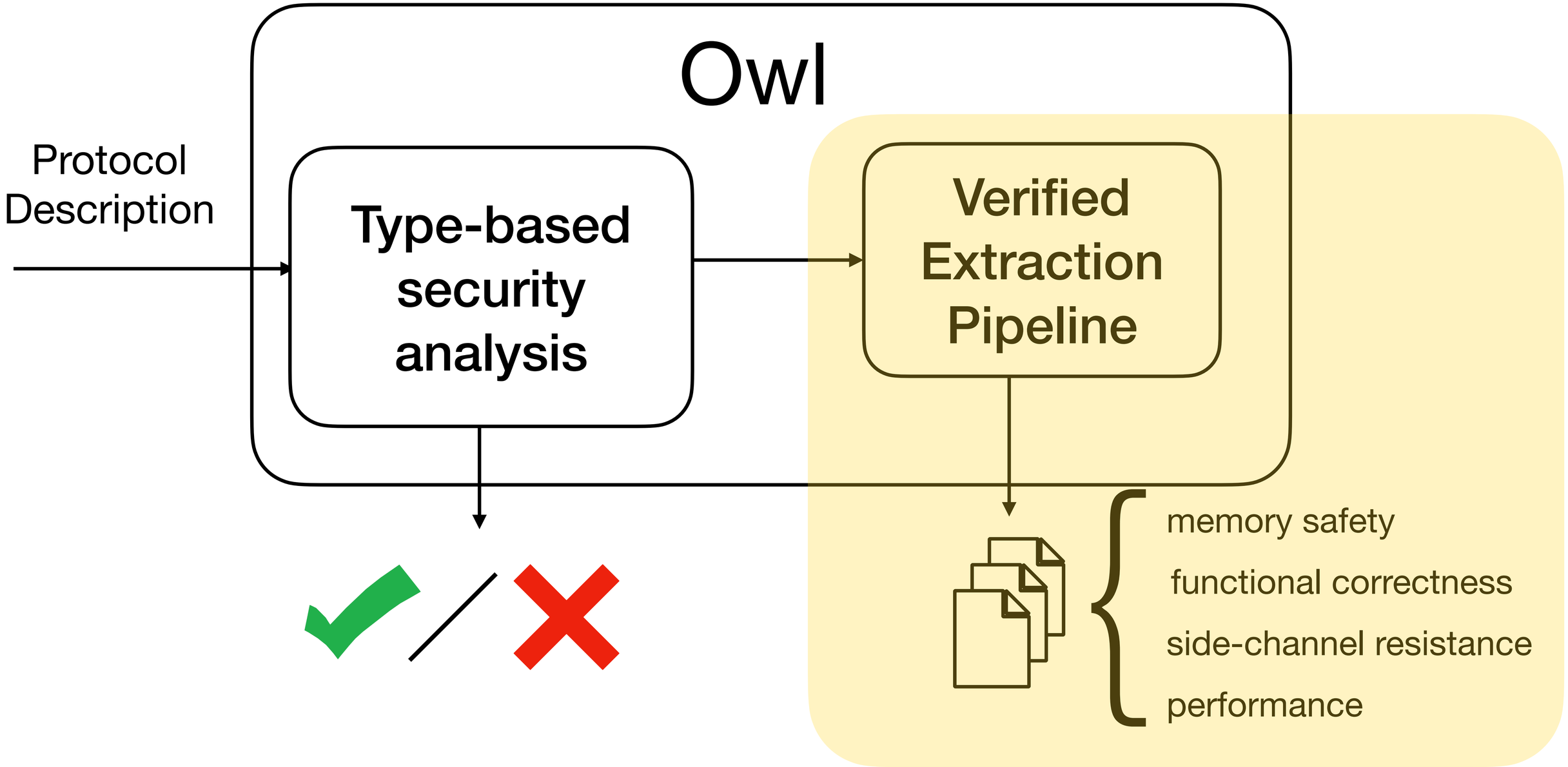


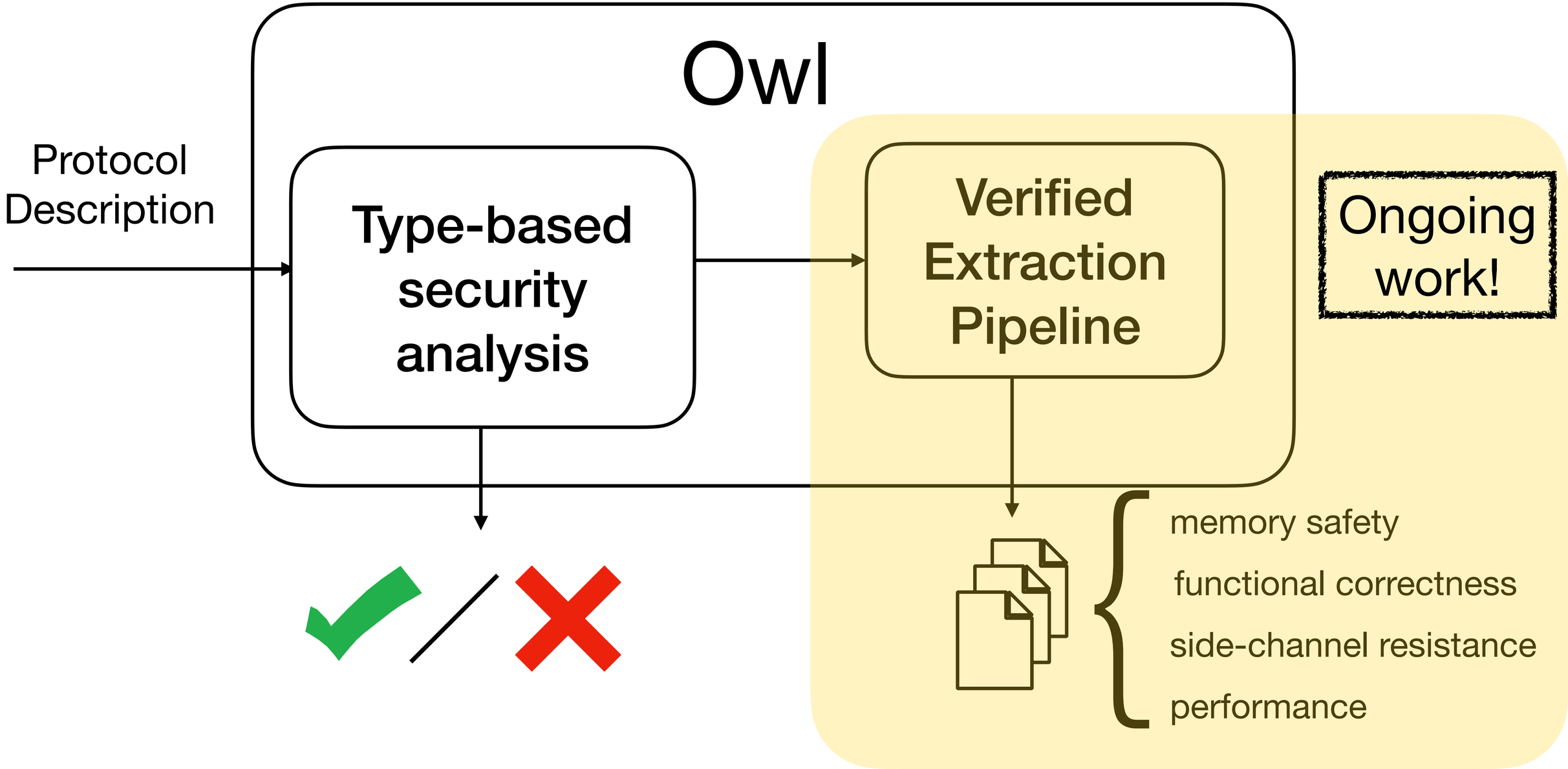
Protocol-Level Modularity



Protocol-Level Modularity

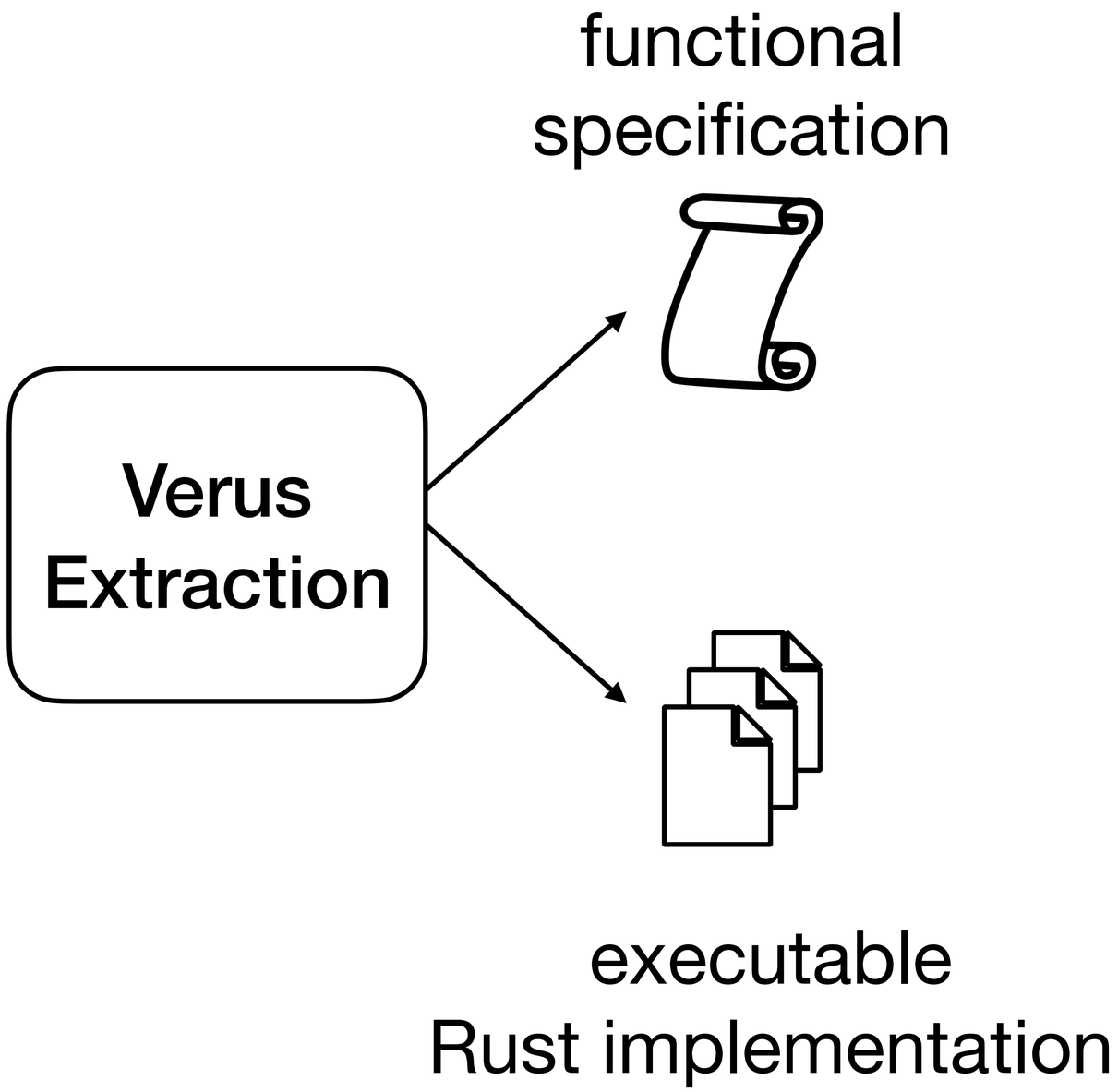



Owl
Protocol Description
Type-based security analysis
Verified Extraction Pipeline
memory safety
functional correctness
side-channel resistance
performance

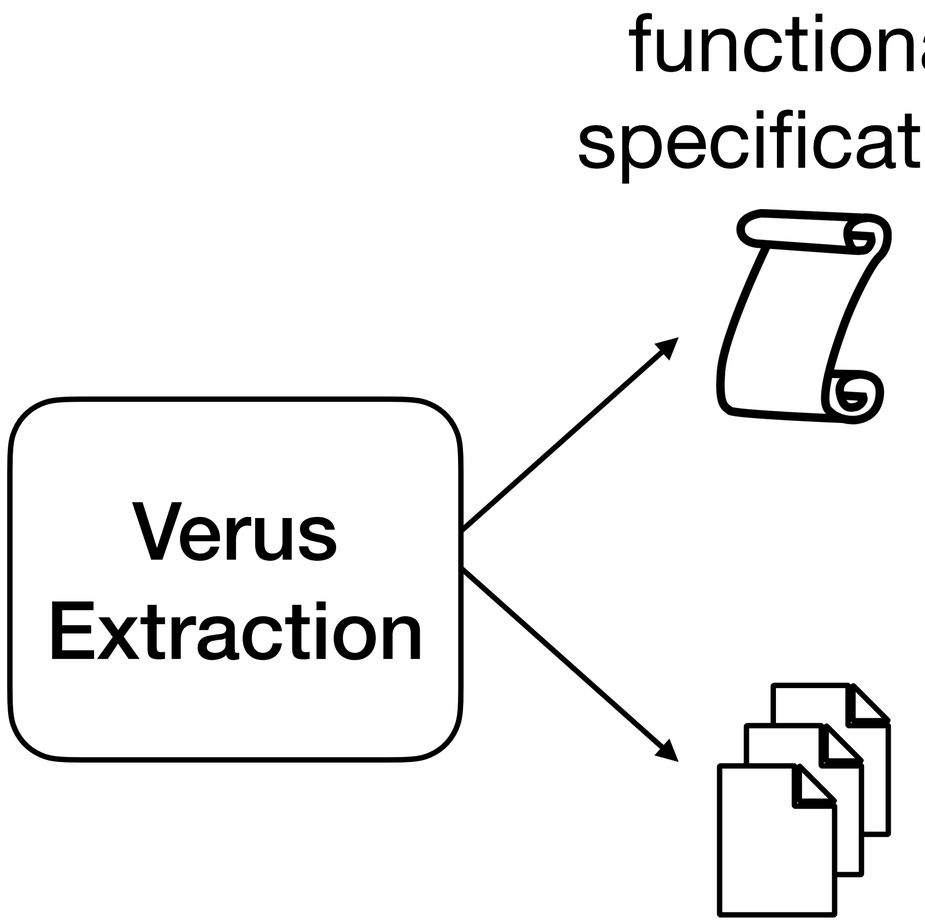

Owl
Protocol Description
Type-based security analysis
Verified Extraction Pipeline
Ongoing work!
memory safety
functional correctness
side-channel resistance
performance

Verus: Verifying Rust Programs using Linear Ghost Types

Verus: Verifying Rust Programs using Linear Ghost Types



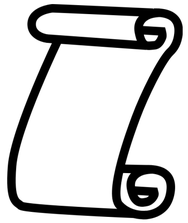
Verus: Verifying Rust Programs using Linear Ghost Types



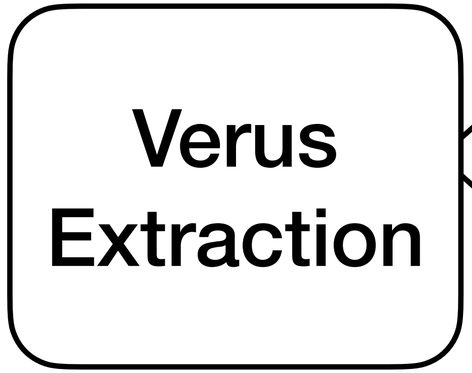
(almost) **directly equal** to input
Owl code

Verus: Verifying Rust Programs using Linear Ghost Types

functional specification



(almost) **directly equal** to input Owl code

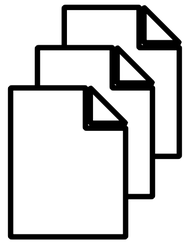


gory low-level details

verified parser library,

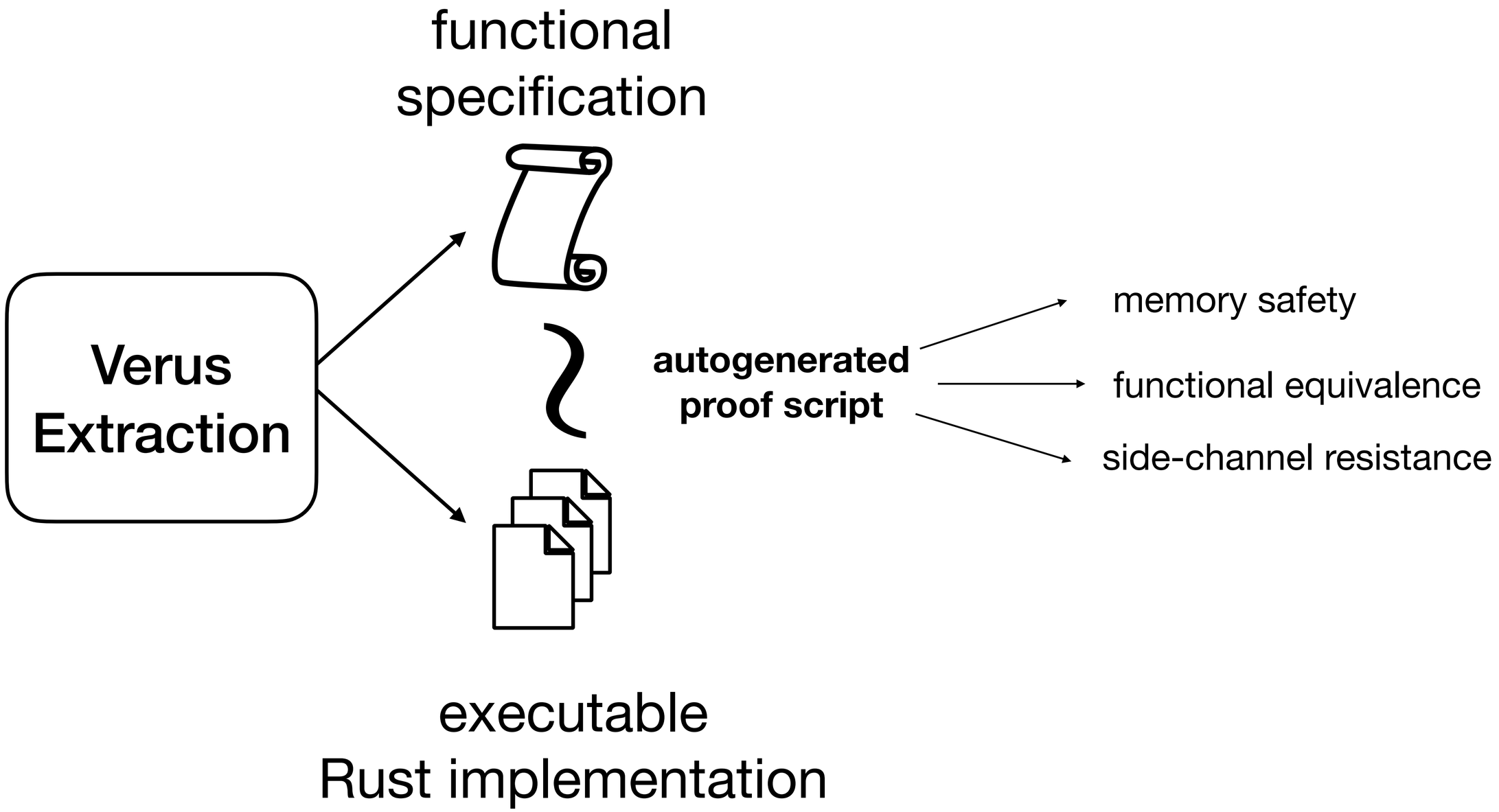
zero-copy ciphers,

...



executable Rust implementation

Verus: Verifying Rust Programs using Linear Ghost Types



Ongoing Work: a verified VPN

Ongoing Work: a verified VPN

widely used:

inside Linux kernel

Ongoing Work: a verified VPN

widely used:

inside Linux kernel

very lean:

implementable in 4K LoC

Ongoing Work: a verified VPN

widely used:

inside Linux kernel

very lean:

implementable in 4K LoC

Goal: verified, drop-in replacement

Owl:
End-to-End Verification of Security Protocols via a Secure Type System

New tool for **modular**, **automated** proofs of security protocols

- Novel use of **type systems** for constructing secure cryptographic protocols
  - Security is proved once-and-for-all;
  - Protocols checked **via type checker**
- Ongoing work: **verified extraction** and drop-in implementation of WireGuard

owl-lang.org

jgancher@andrew.cmu.edu