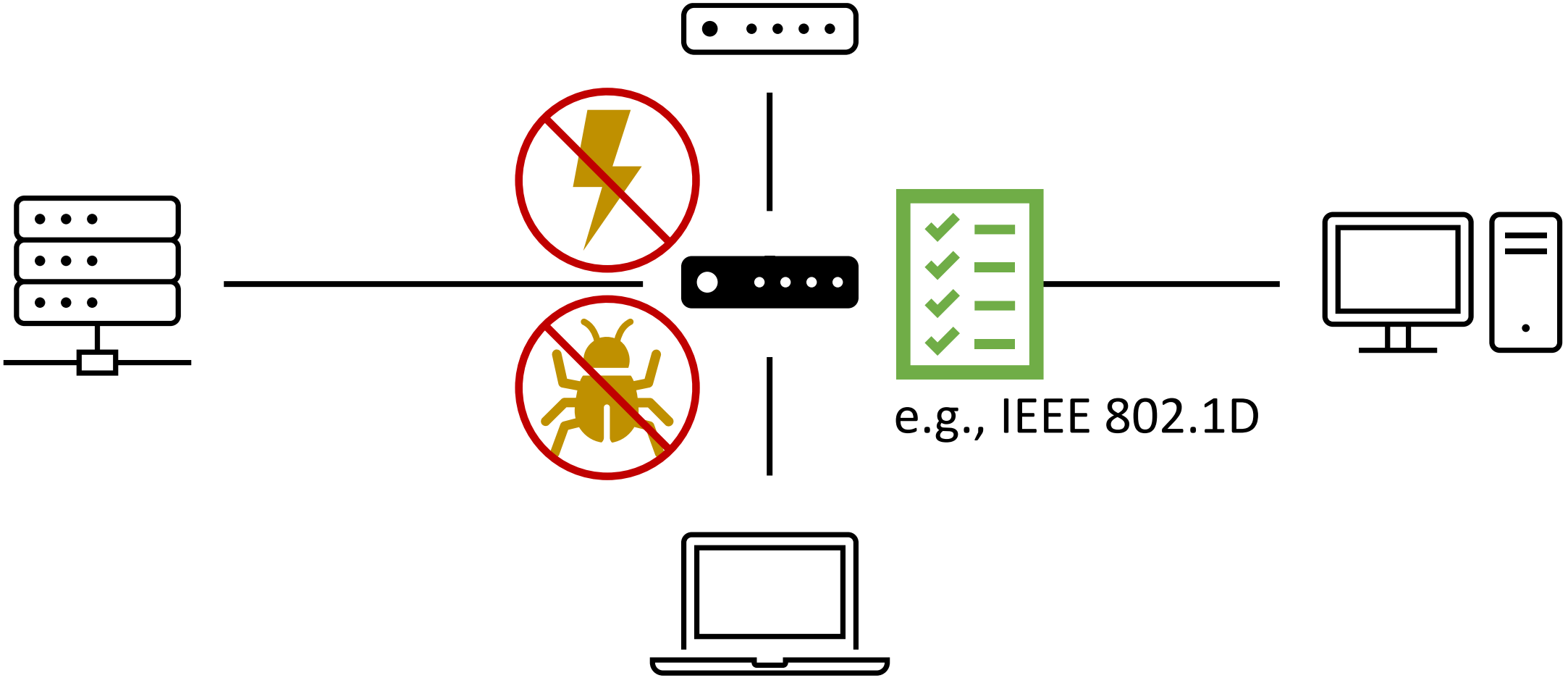
Automated Verification of Network Function Binaries

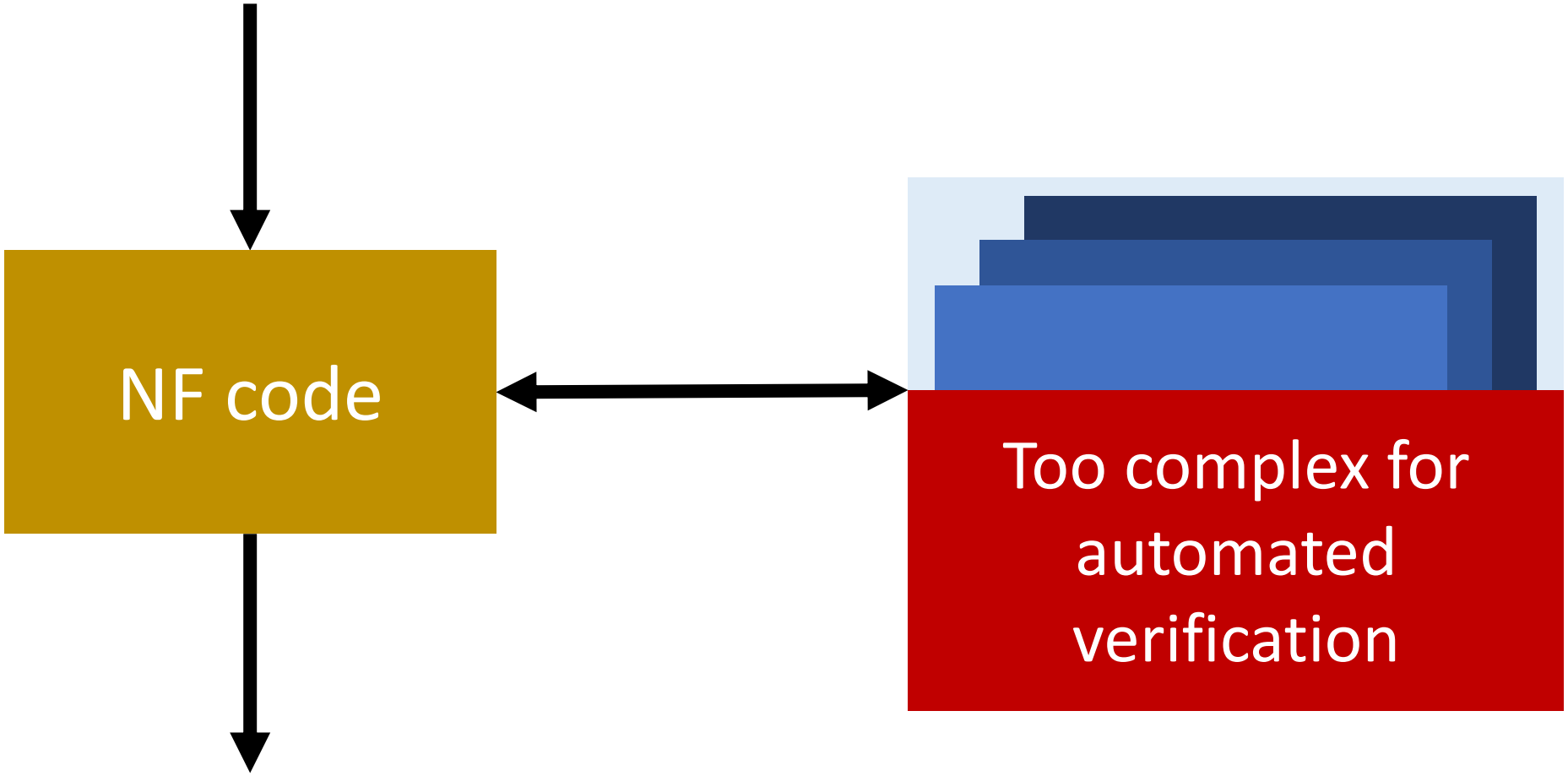
Solal Pirelli,
Akvilė Valentukonytė, Katerina Argyraki, George Candea

EPFL

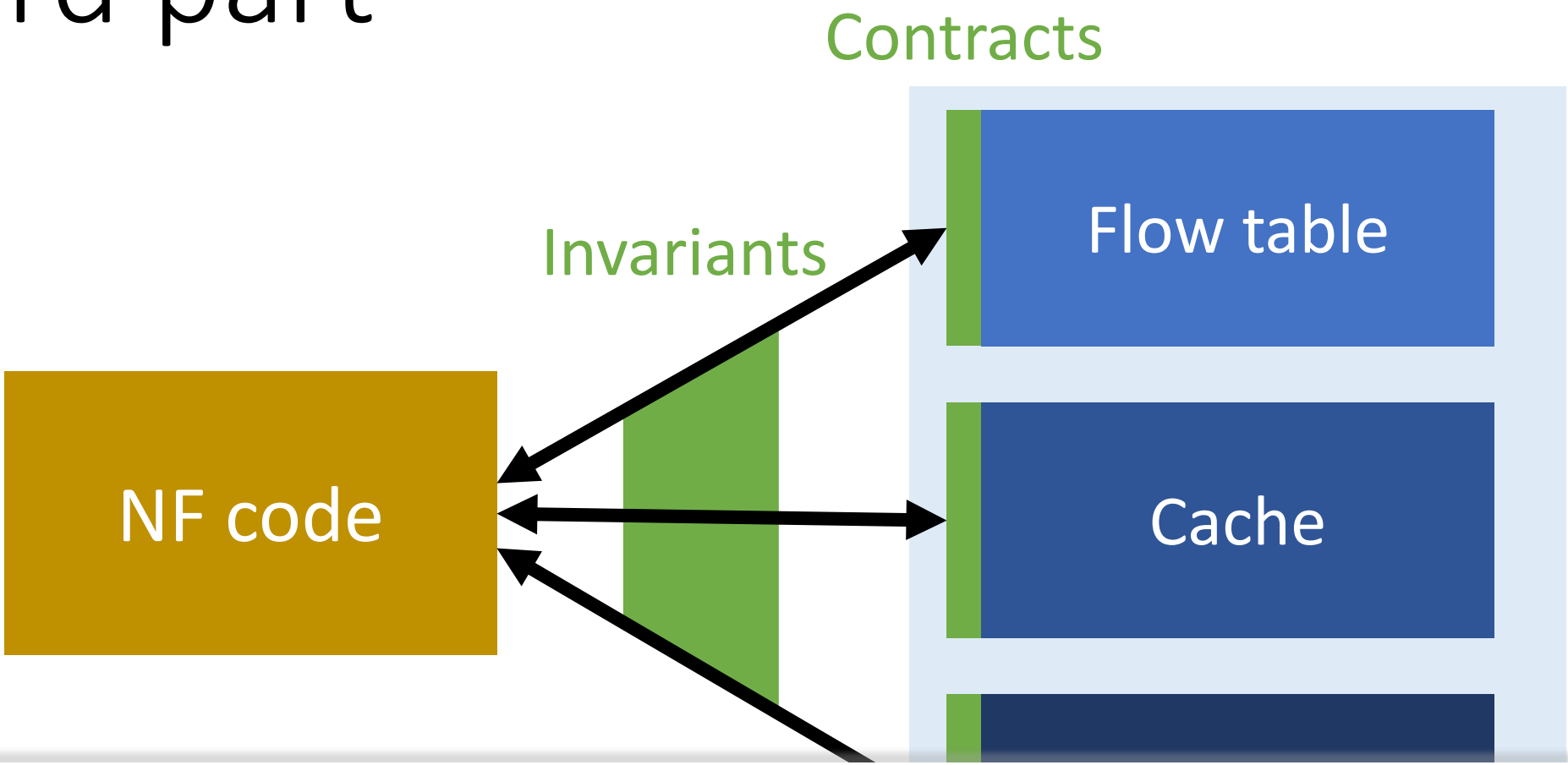
Context



Network Functions



Hard part



Automatically prove the code obeys contracts by inferring the necessary invariants

Requirements

NFs need **automated verification**, but:

1. Developers use **many data structures**
2. Operators deploy **binaries**

Vigor *SOSP'19*	**Gravel** *NSDI'20*	**This work**
◐	◐	●
○	○	●

Describing data structures with
maps
enables the automated verification
of network function binaries

Outline

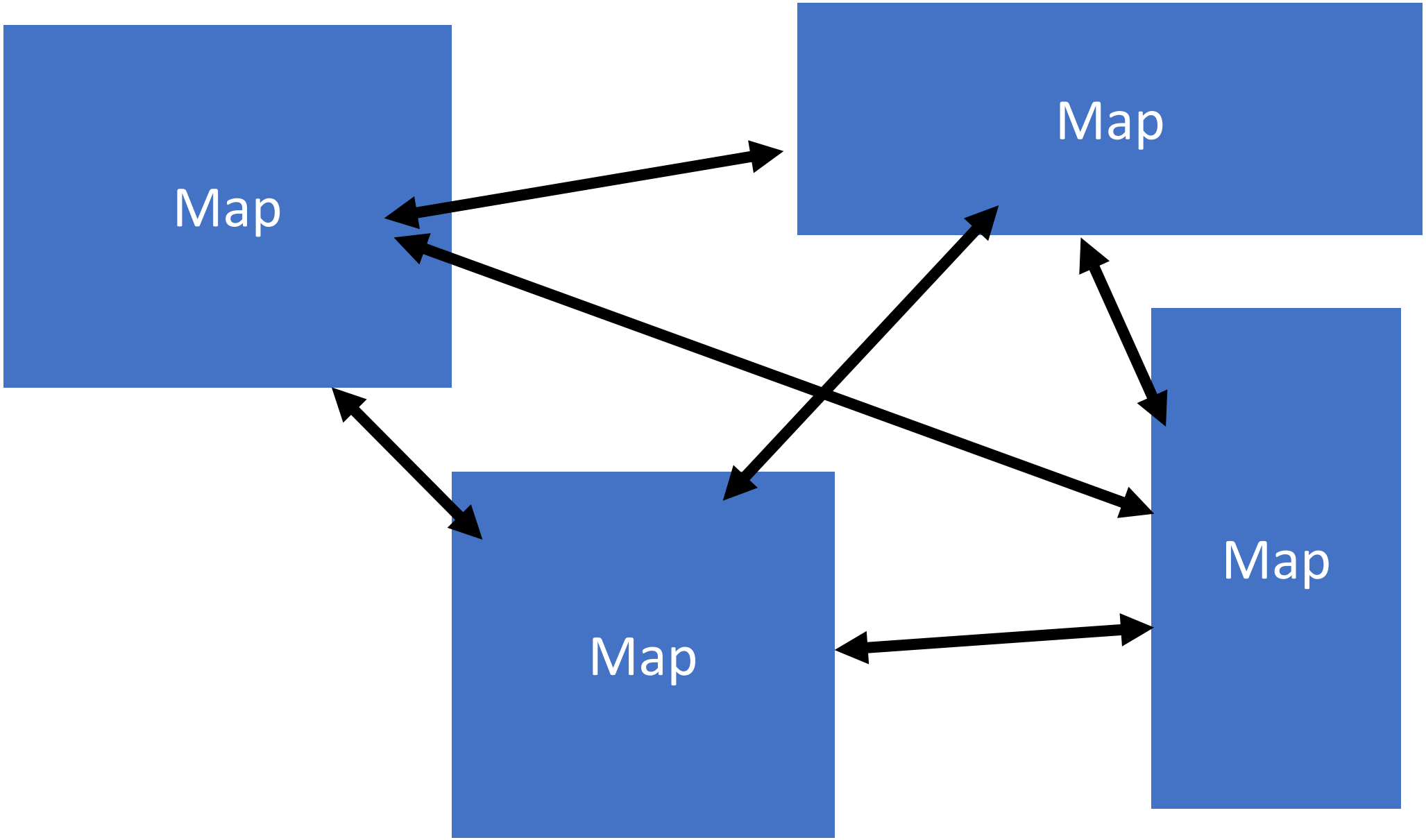
Intro

Abstracting data structures

Handling binaries

Implementation

Evaluation & Limitations


Map
Map
Map
Map

Contract example: `LRU.evict()`

State: map M (value → age)

Pre: M.**length**() > 0

Post: M.**contain**

M' = M.**re**

∀ (v,a) ∈ M

```
void* LRU_expire(struct LRU* lru)
// assert len(lru.items) > 0
// age = lru.items[result]
// assume(lru.items.forall(lambda v, a: a <= age))
// del lru.items[result]
```

Translation

Map operations → decidable solver queries

Problem: map size is unknown

Insight: few operations per packet

Translation

Track **known** items explicitly,
and **unknown** items as an invariant

$\mathrm{V}_1 = \mathrm{get}(\mathrm{M}, \mathrm{K}_1)$ $\quad (\ \mathrm{K}_1, \mathrm{V}_1, \mathit{present}\ )$

$\mathrm{remove}(\mathrm{M}, \mathrm{K}_2)$ $\quad (\ \mathrm{K}_2, \_, \mathit{absent}\ )$

others: λk,v,p. ...

Finding invariants

Use **known** items as templates:

"Values in M are > X"

"Values in M_1 are keys in M_2"

…

Outline

Intro

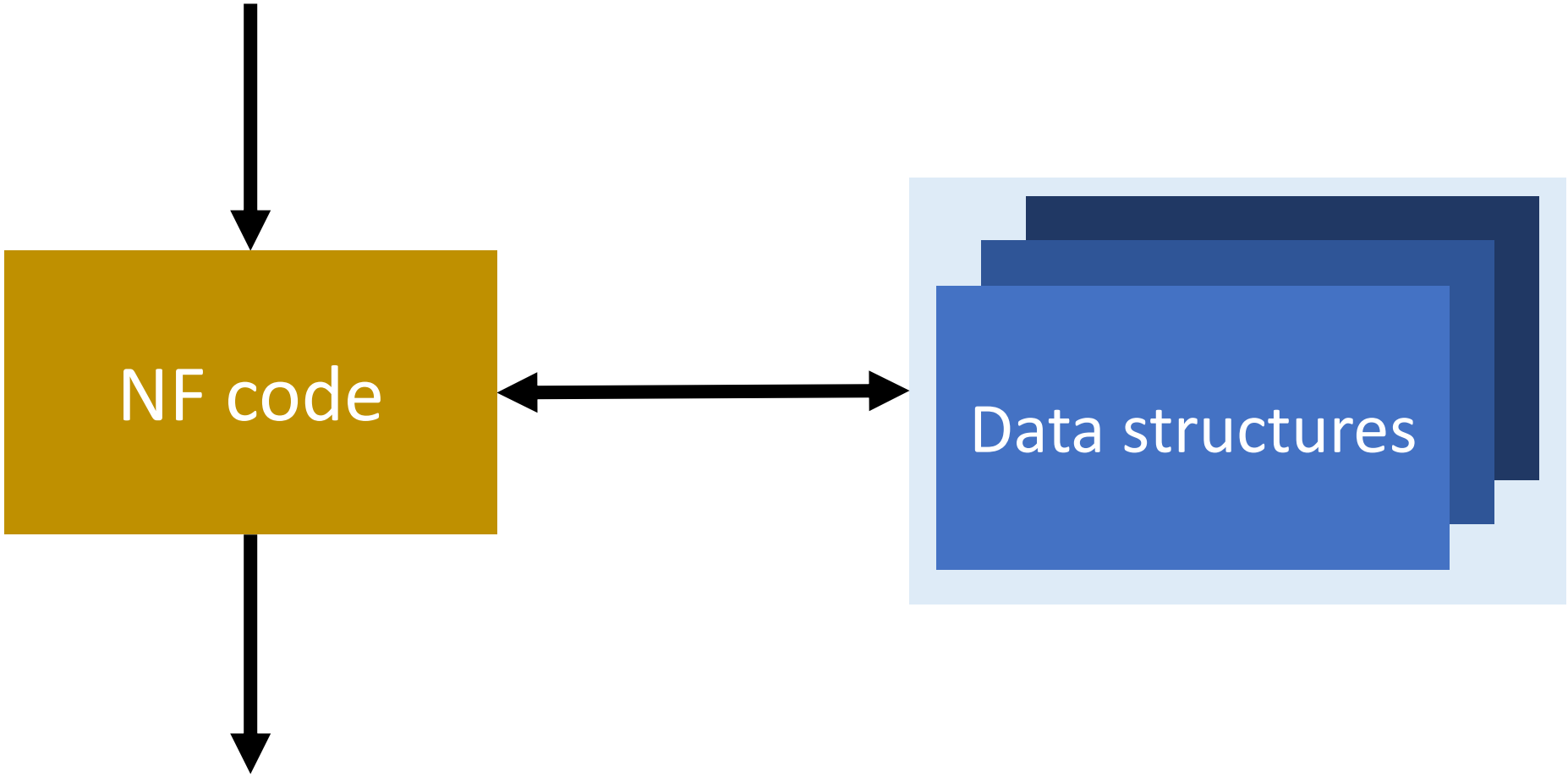
Abstracting data structures

Handling binaries

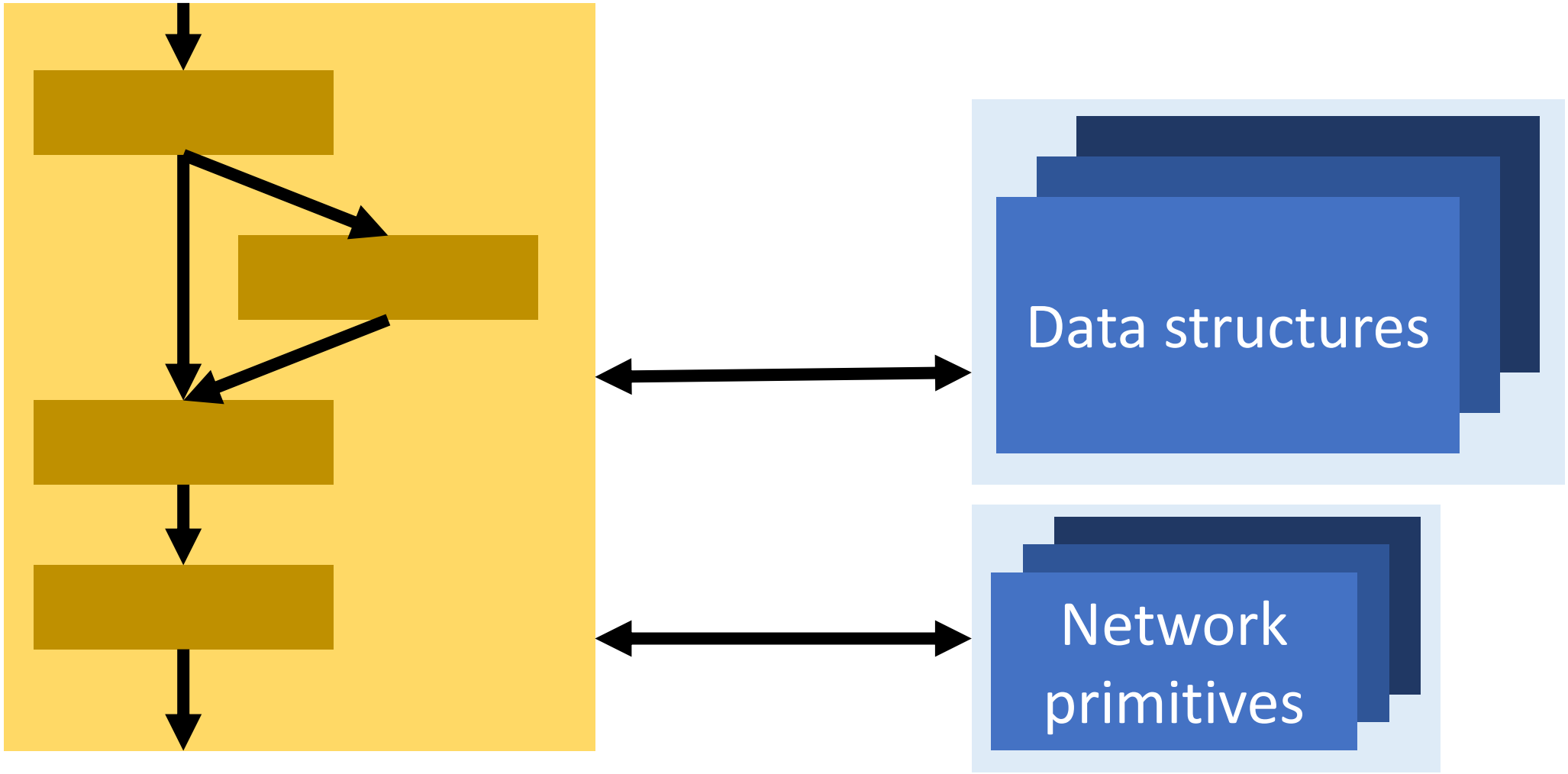
Implementation

Evaluation & Limitations

Network Functions

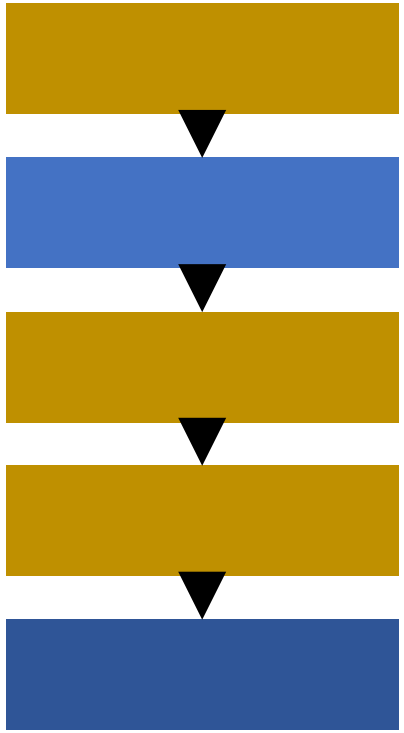


Source code

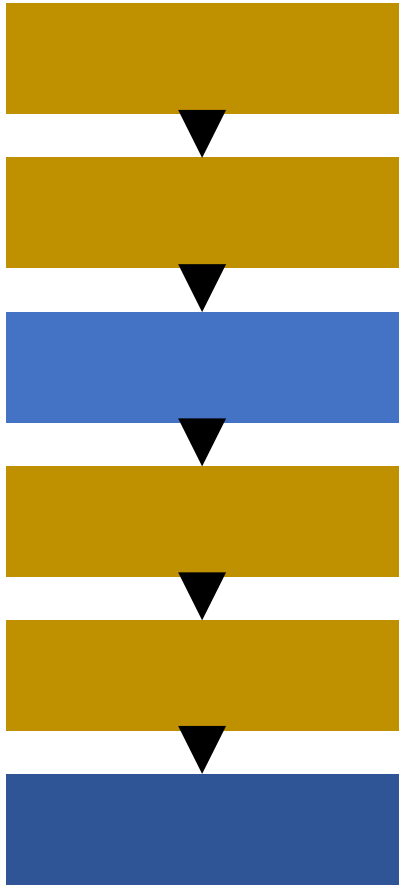


Exhaustive symbolic execution

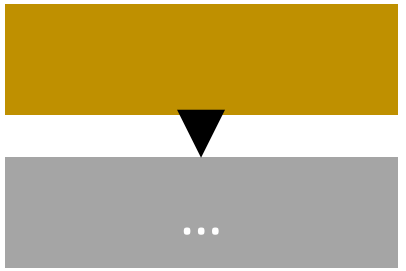
pkt_is_ipv4 ∧ …



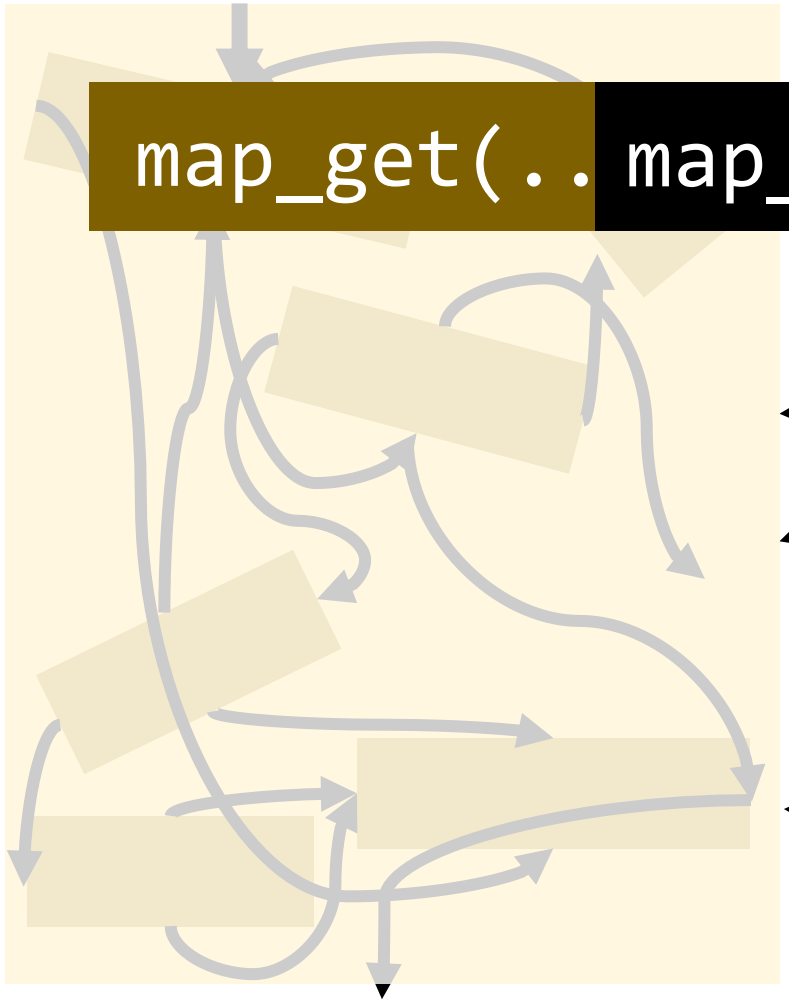
!pkt_is_ipv4 ∧ …



…



Binaries



Data structures

Network primitives

Dynamic linking

Outline

Intro

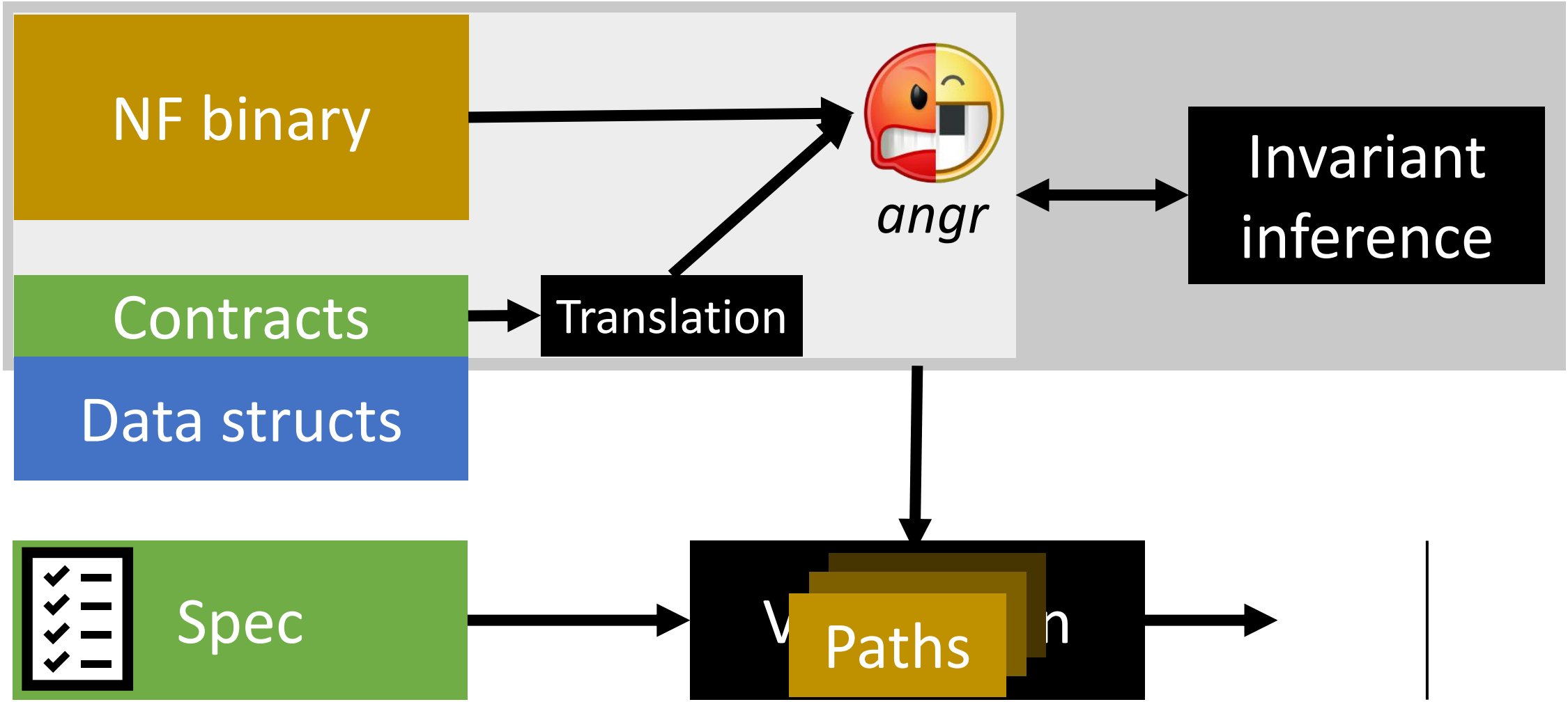
Abstracting data structures

Handling binaries

Implementation

Evaluation & Limitations

Klint


NF binary
Contracts
Data structs
Spec
angr
Translation
Invariant inference
Paths

Outline

Intro

Abstracting data structures

Handling binaries

Implementation

Evaluation & Limitations

Verified NFs

C: Bridge
Firewall
Maglev
NAT
Policer
Router

Rust: Policer

Verification performance

< 2 min / NF

Single-threaded prototype

Verified NF performance

Prototyping new data structures is now quick
(then verify manually, as previous work)

Faster than previous verified NFs

Why only NFs?

Few and shallow “networking” operations

Explicitly modeled by our tool

Not feasible for e.g. Linux

Map limitations

Good fit for most data structures

Not applicable to all complex code: regex, ...

Summary

We verify network function binaries
that use any data structure with map-based contracts

We translate maps by separating known/unknown items,
which scales and enables invariant inference

dslab.epfl.ch/research/klint

solal.pirelli@epfl.ch