verifiedSCION: Verified Secure Routing

Marco Eilers

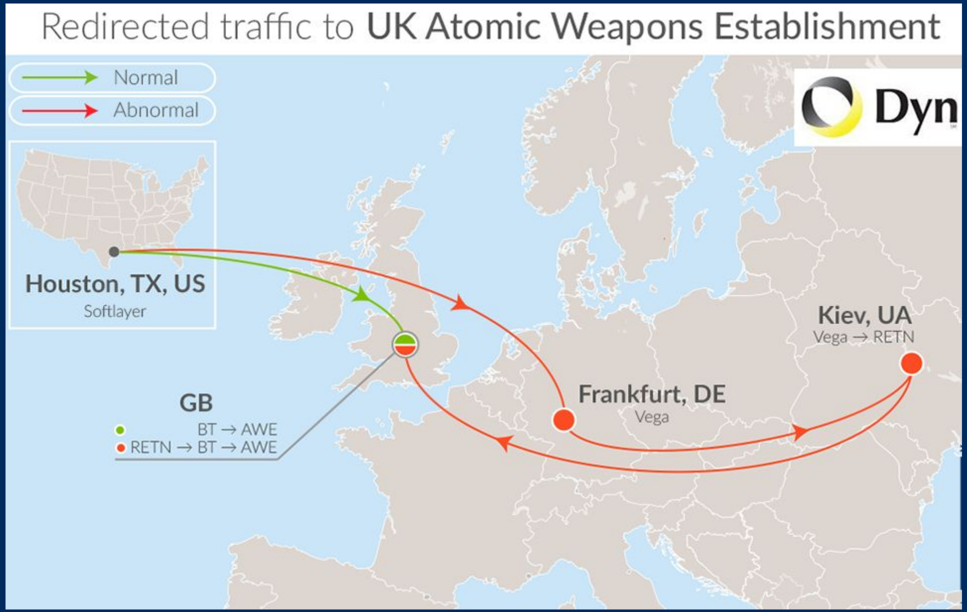
Joint work with

João C. Pereira, Tobias Klenze, Sofia Giampietro,
Markus Limbeck, Dionysios Spiliopoulos, Felix A. Wolf,
Christoph Sprenger, David Basin, Peter Müller, Adrian Perrig

ETH zürich

- Internet is a network of Autonomous Systems (AS)
- Each AS is a network of routers run by an institution
- Routes between AS are discovered using Border Gateway Protocol (BGP)
- Based on trust, for instance, any AS can announce any IP address range

- There are numerous ways to attack Internet routing



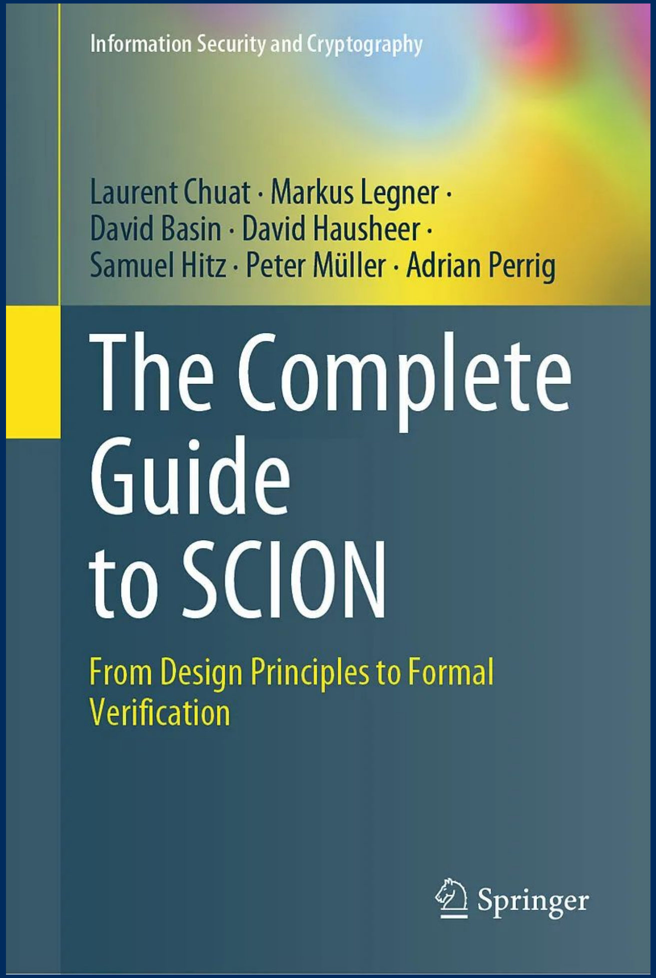
- In 2013, Ukrainian ISP announced route prefixes to British Telecom AS
- Traffic of some UK customers was redirected to Ukraine, including UK's Atomic Weapons Establishment

- Senders have no control over the taken routes
- Routers on path can read and modify data

Scion Internet Architecture

- Scion is a new architecture for inter-domain routing
    - Path control, e.g., geofencing
    - Multipath communication
    - DDoS protection

- Research and commercial deployments

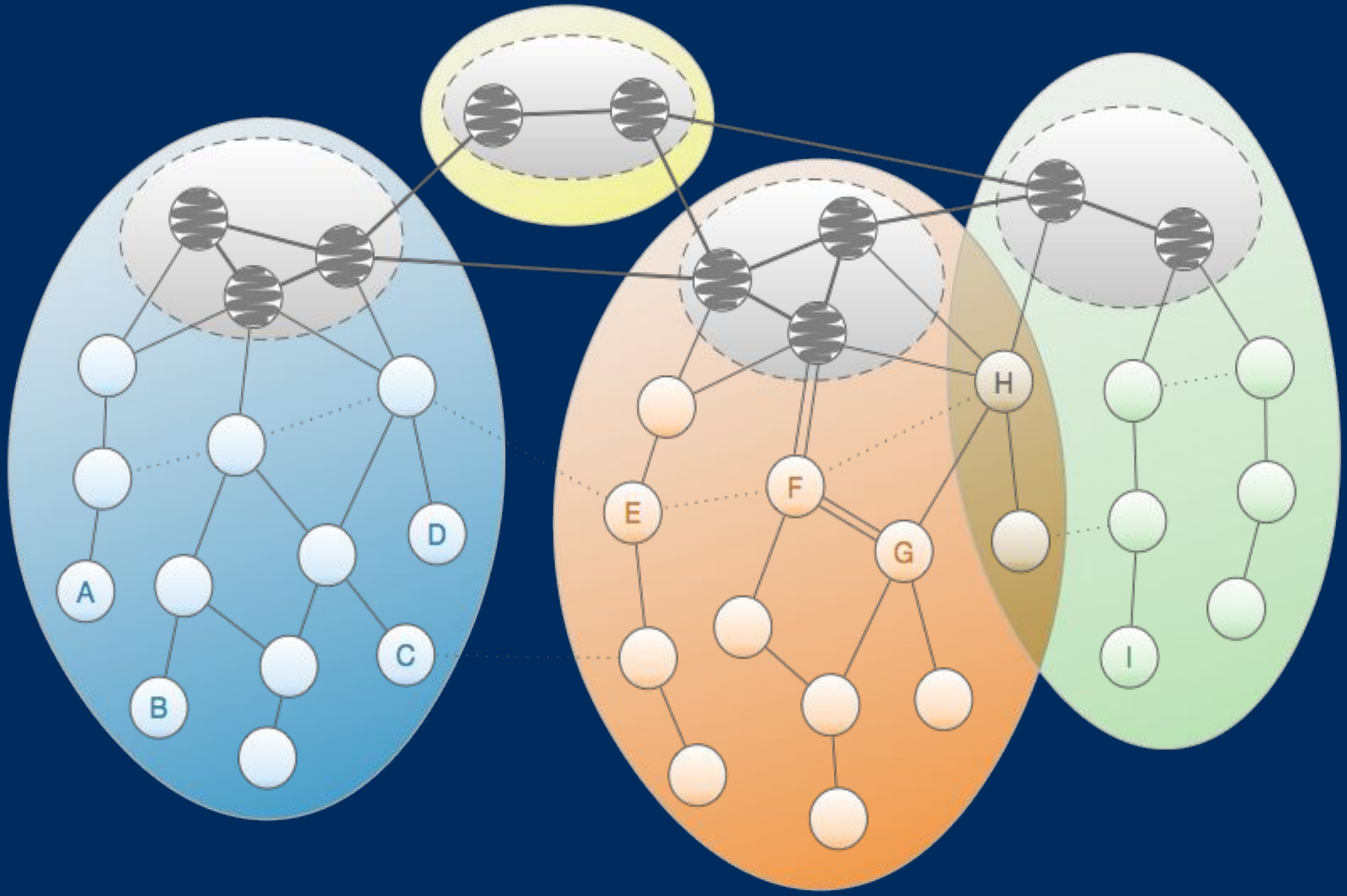


VerifiedSCION

Formal end-to-end verification of security and correctness

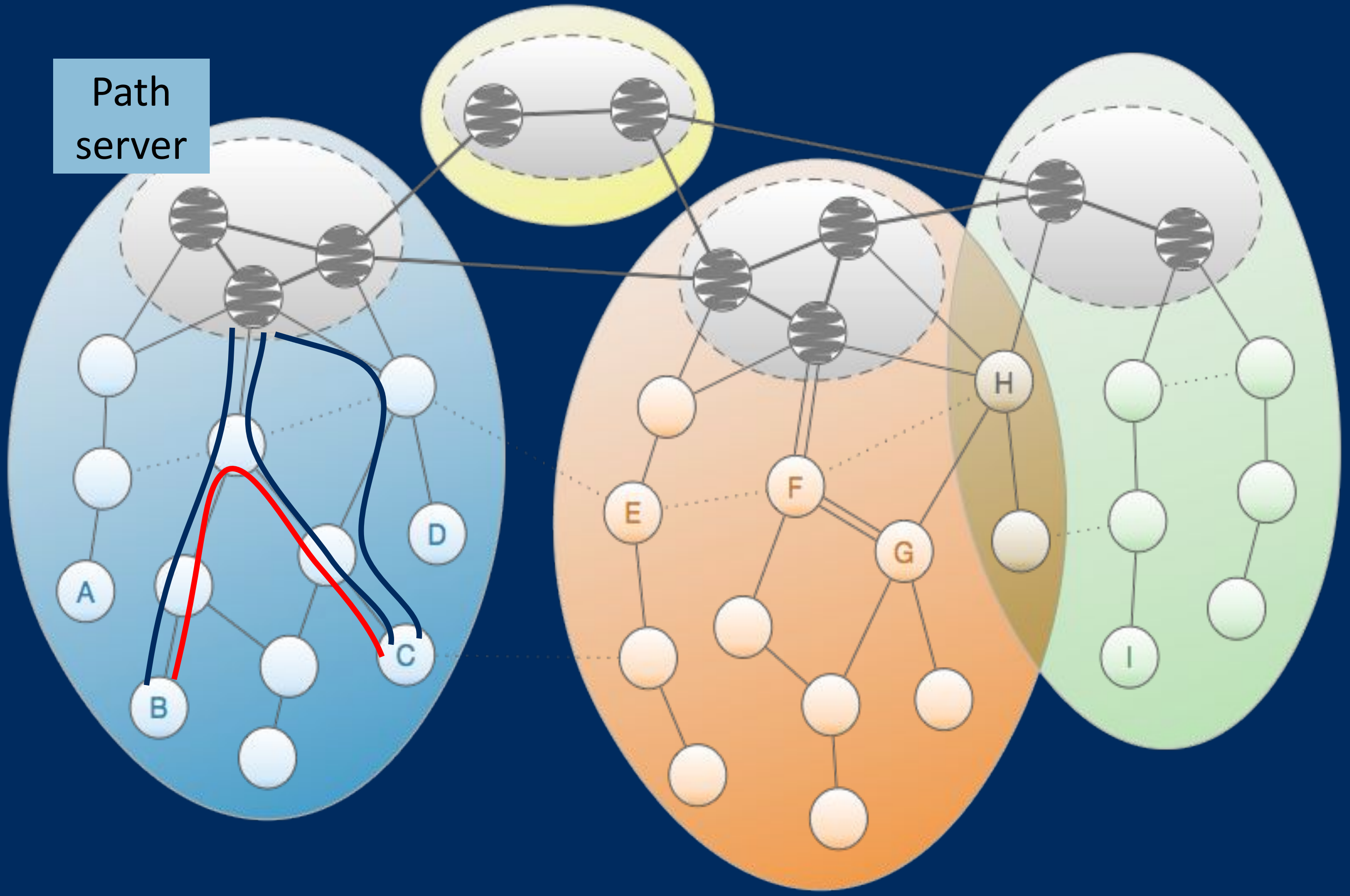
Isolation Domains

ASes are organized into isolation domains with independent control planes and root of trust



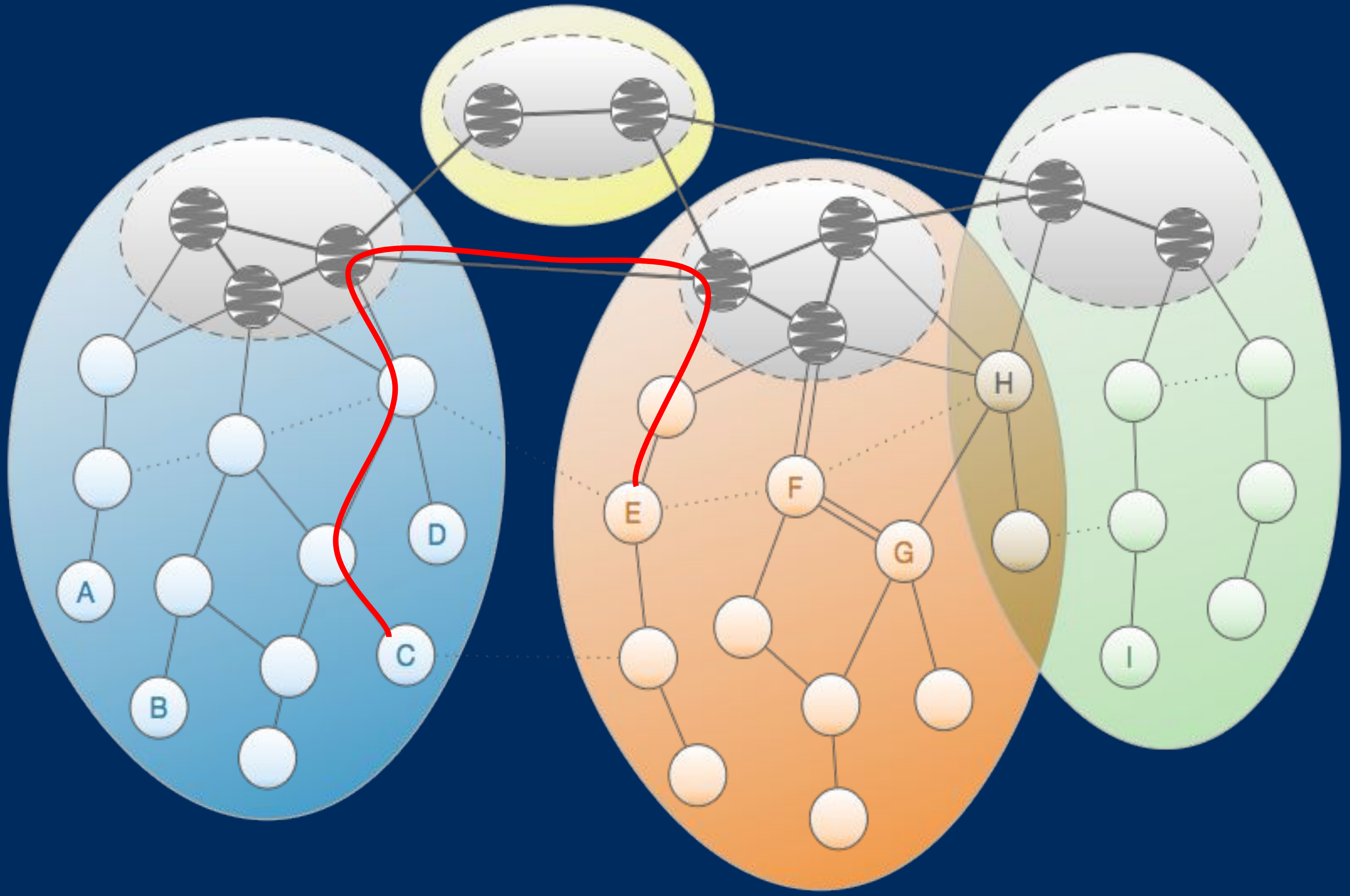
Scion Routing

- Path exploration
  - Paths are sequences of signed hop fields
  - Each hop field carries routing information for one AS (input and output ports)
- Path registration with path server
- Path selection
  - Path is stored in packet header



Scion Forwarding

- Path is stored in packet header
- Consisting of up segment, core segment, and down segment



Security and Correctness

- Protocol-level properties

  Path validity: Constructed paths are valid and reflect the routing decisions by on-path ASes.

  Path authorization: Packets travel only along previously authorized paths

  Detectability: An active attacker cannot hide their presence on the path

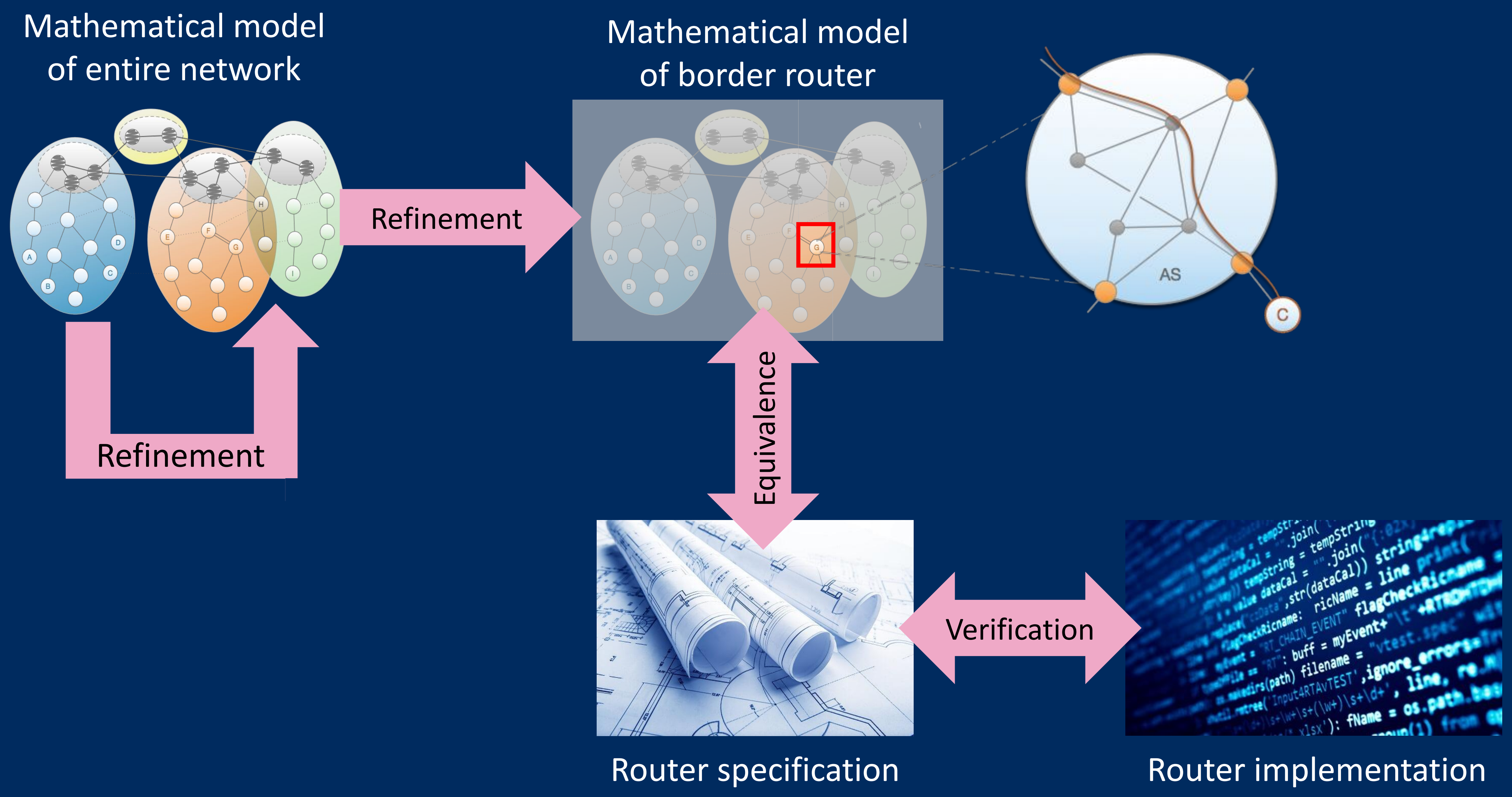
- Code-level properties

  Safety: No run-time errors

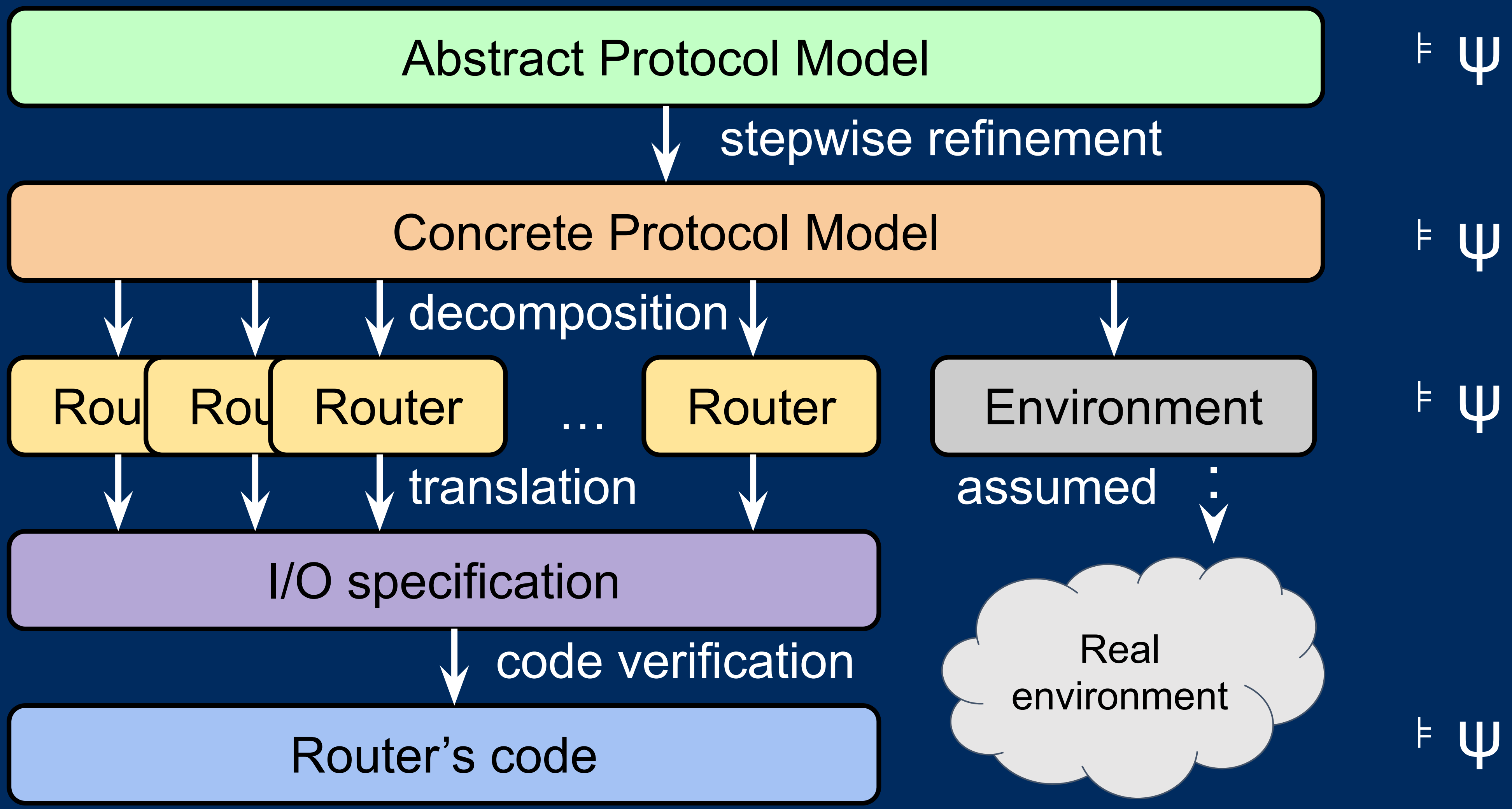
  Correctness: Routers and servers implement protocol correctly

  Progress: Required I/O happens eventually

  Secure information flow: Code does not leak information about crypto keys


Mathematical model of entire network
Refinement
Refinement
Mathematical model of border router
AS
C
Equivalence
Verification
Router specification
Router implementation

Igloo Framework (OOPSLA 20)



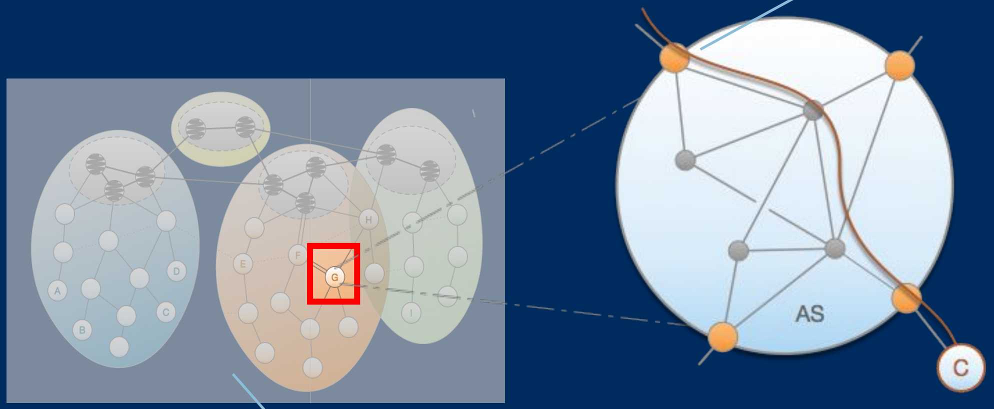
Protocol Models

- Formalize the design model as transition system

```
var in: Multiset<Msg>
var out: Multiset<Msg>

initially
  in = { }
  out = { }

event process(M, M')
  guard  M ∈ in ∧ valid(M) ∧
       reply(M, M')
  action
    in := in \ { M }
    out := out ∪ {M'}
```

- Model describes

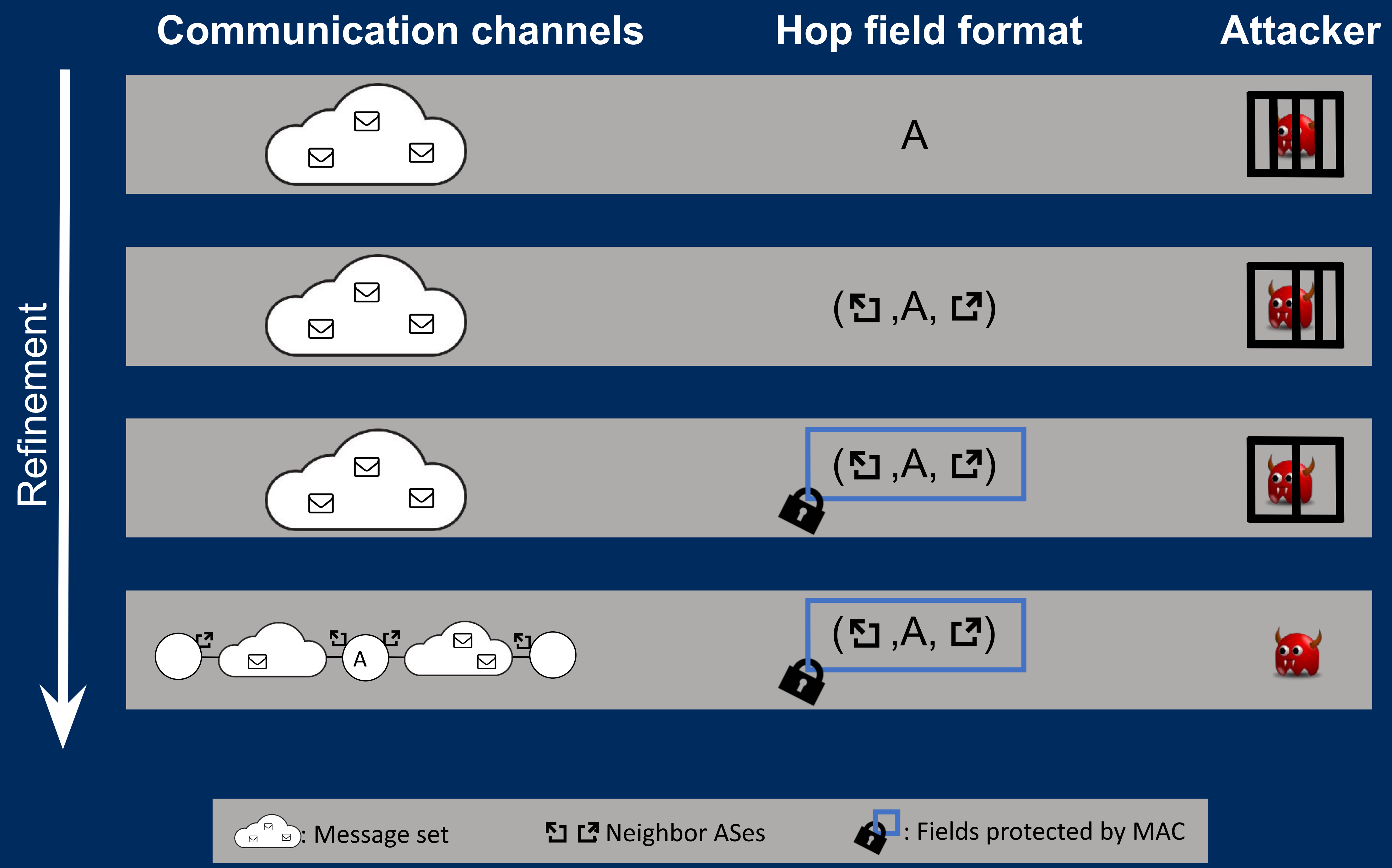
System:
Border router



Environment:
Network
End hosts
Attacker

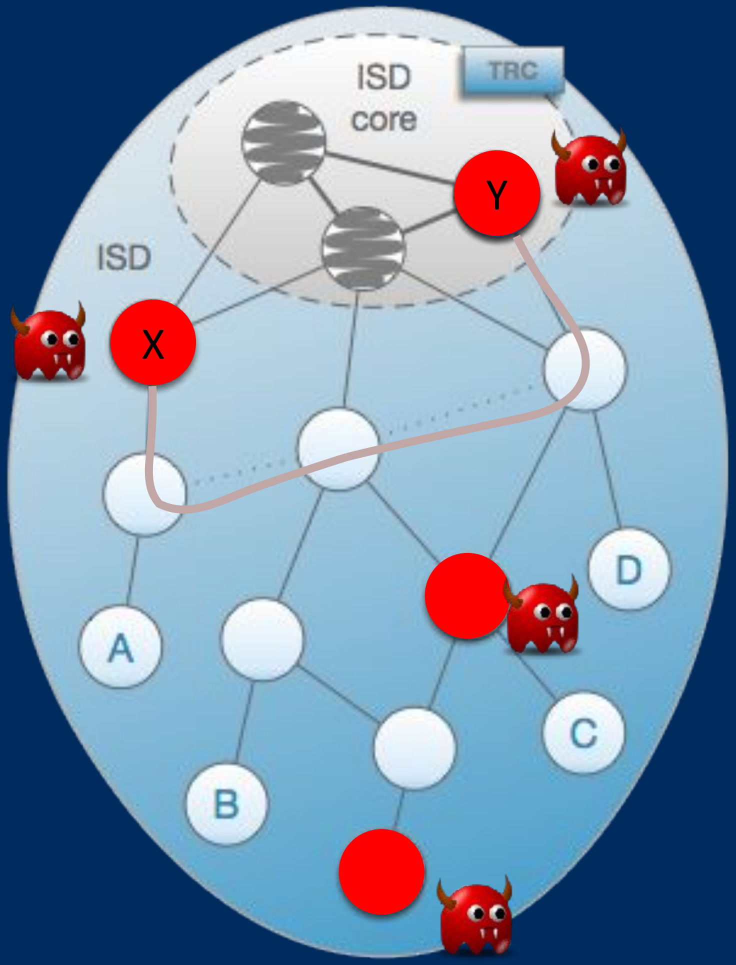
Stepwise Refinement

- Protocol models are developed by stepwise refinement
- Prove properties of most abstract model
- Each refinement
  - Incorporates additional system requirements or environment assumptions
  - Preserves properties of more-abstract system
  - Is tool-checked in Isabelle
- Strategy: strengthen attacker while increasing protection of paths


Communication channels
Hop field format
Attacker
Refinement
A
(⮤ ,A, ⮥)
(⮤ ,A, ⮥)
(⮤ ,A, ⮥)
A
: Message set
Neighbor ASes
: Fields protected by MAC

Attacker Model

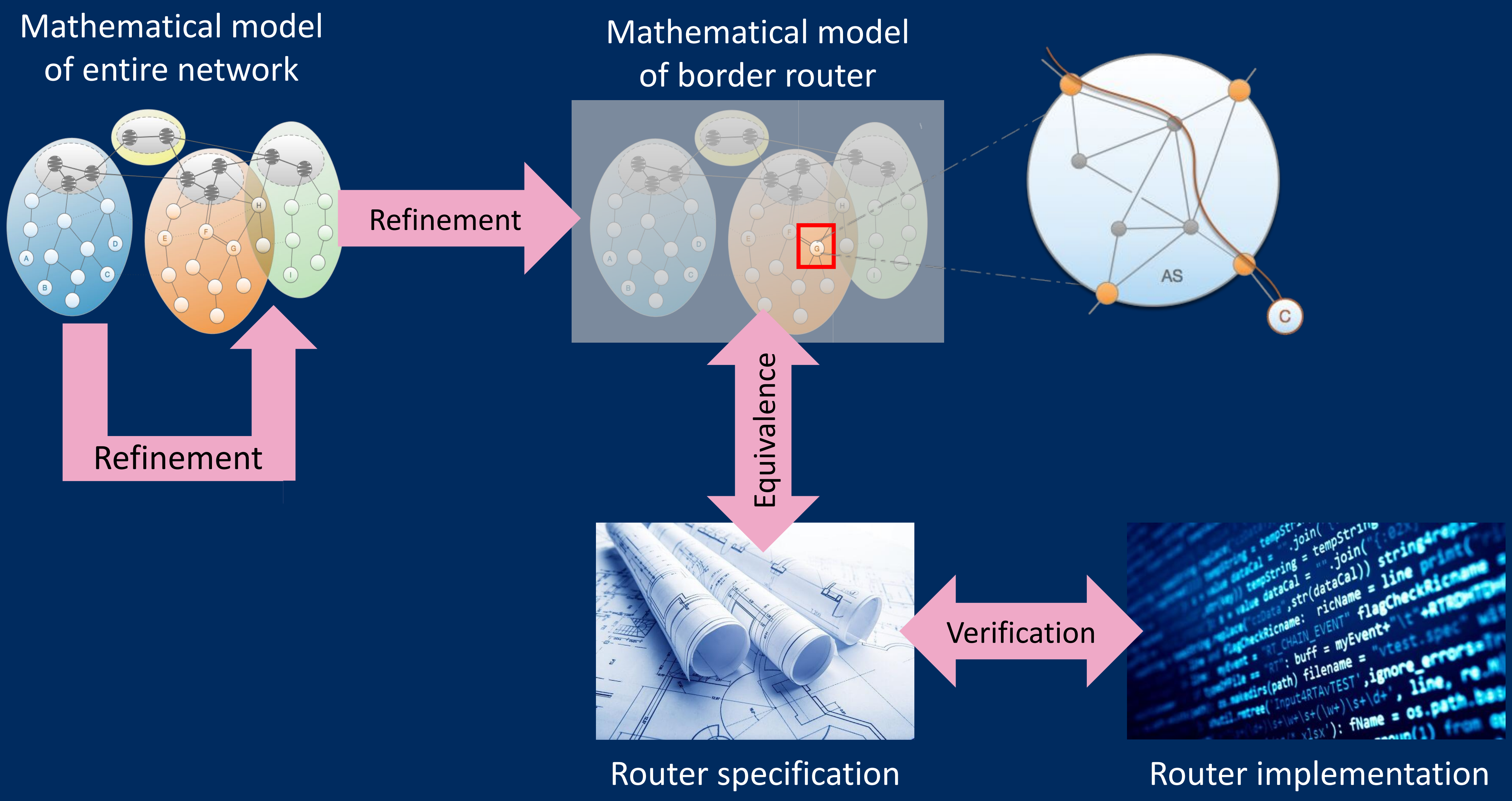
- Localized, colluding Dolev-Yao attacker model
- Attacker:
  - Actively controls some ASes
  - Can observe, block, and inject messages
  - Can eavesdrop globally
- Cryptography is assumed to be perfect



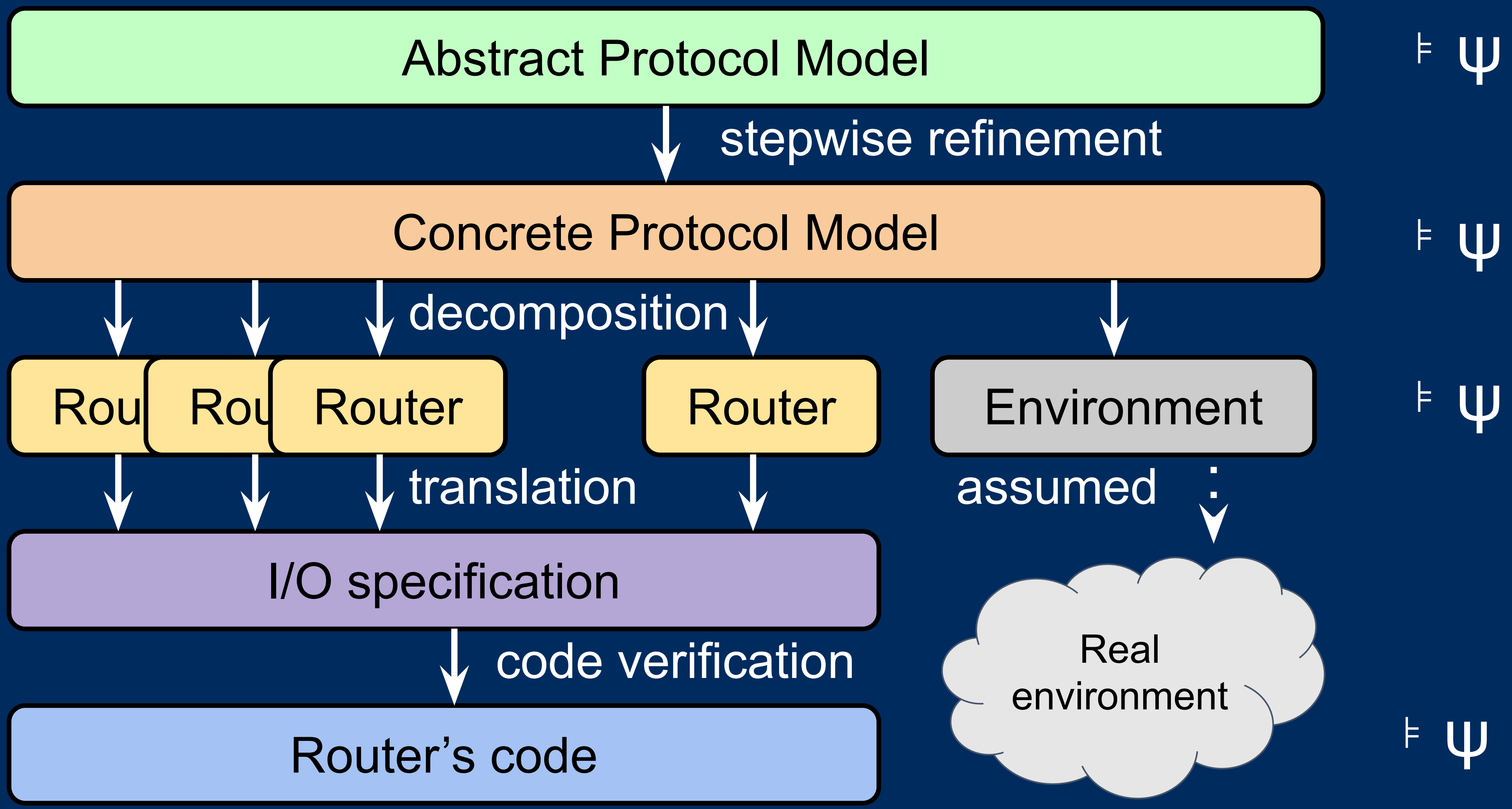
Results of Protocol Verification

- A formal model of the network components and their environment
- Model serves as formal specification for the implementation
- Proofs of the desired properties under the assumption that each component satisfies its specification
- 16,100 lines (models and proofs)

- Improved understanding of protocols and properties
- Revealed design flaws that enabled five different security attacks
- Issues were found during modeling and formalization


Mathematical model of entire network
Mathematical model of border router
Refinement
Refinement
AS
C
Equivalence
Verification
Router specification
Router implementation

Igloo Framework (OOPSLA 20)



Decomposition

- Last refinement step must include explicit I/O events
  - components and environment interact *only* via I/O events
  - I/O event represents atomic I/O operation in implementation

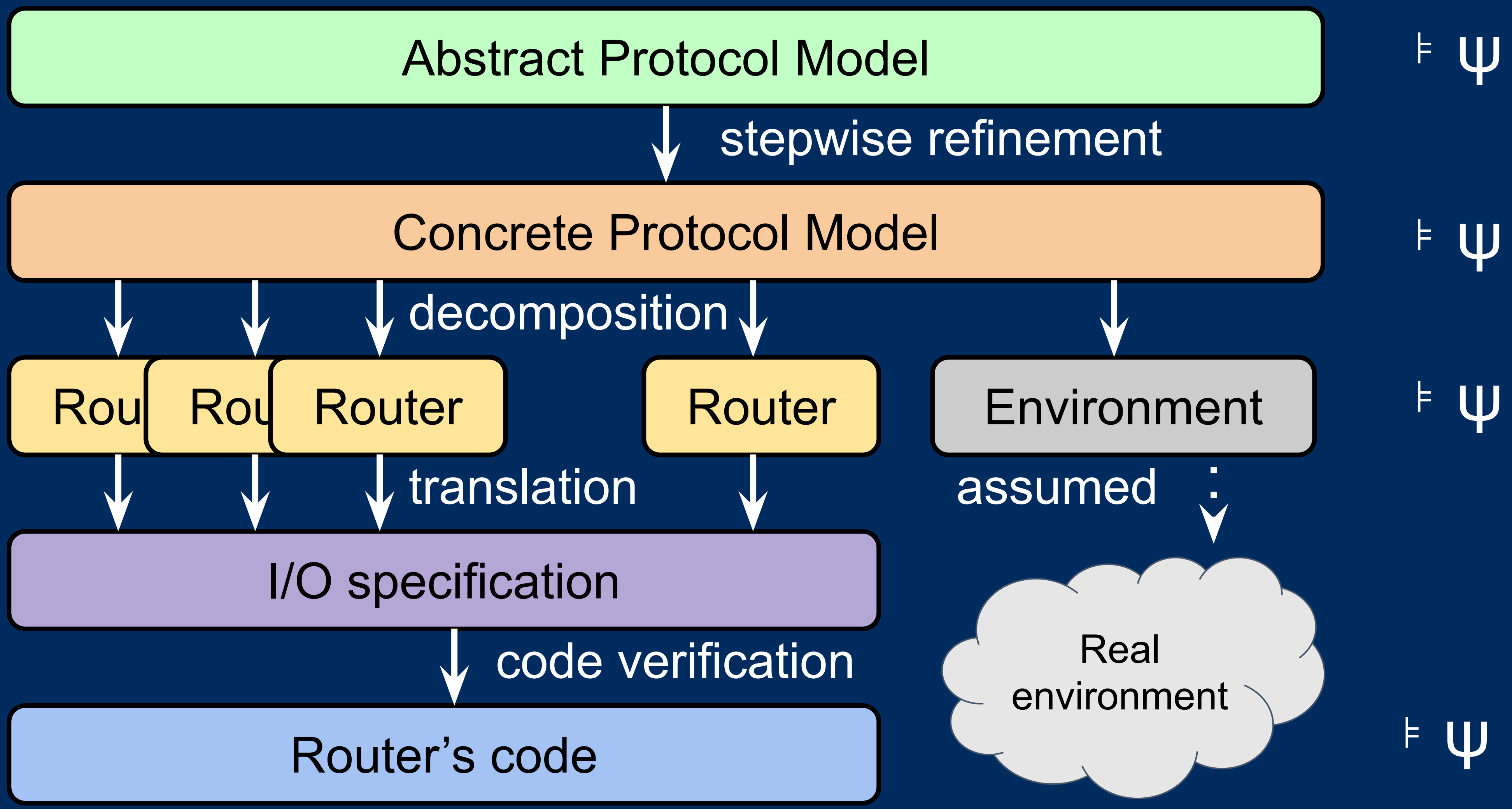
```
var in: Multiset<Msg>
var out: Multiset<Msg>

event process(M, M’)
  guard  M ∈ in ∧
      valid(M) ∧
       reply(M, M’)
  action
   in := in \ { M }
   out := out ∪ {M’}
```

Refinement

```
…
var i_buf : Multiset<Msg>
var o_buf : Multiset<Msg>

event receive(M)
  guard  M ∈ in
  action
   in := in \ { M }
   i_buf := i_buf ∪ {M}
```

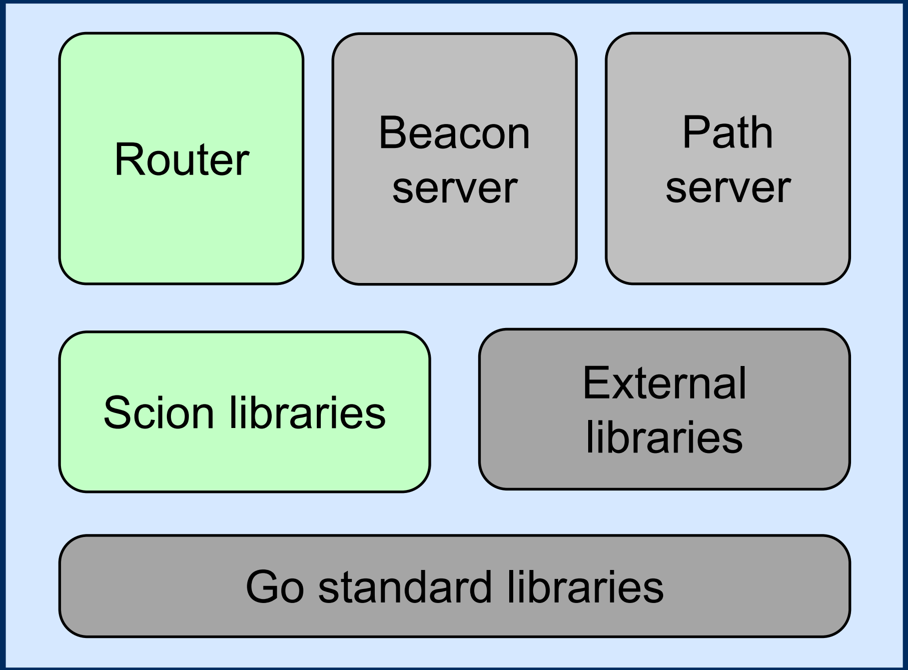
```
event process(M, M’)
  guard  M ∈ i_buf ∧
      valid(M) ∧
       reply(M, M’)
  action
   i_buf := i_buf \ { M }
   o_buf := o_buf ∪ {M’}

event send(M, Addr)
  …
```

Igloo Framework (OOPSLA 20)



Scion Implementation

- Open-source implementation
  - 35kloc of Go (Router: 4.7kloc)
  - Uses concurrency, async, globals
- Verify safety, functional correctness, progress, secure information flow
- Assume correctness of external libraries, Go compiler, OS, hardware



Gobra: Verification for Go (CAV 21)

```
func indexOf(l [ ]int, i, val int) (res int)
  requires 0 <= i && i < len(l)
  requires forall j int :: i <= j && j < len(l) ==> acc(&l[ j ])

  decreases len(l) - i

  ensures forall j int :: i <= j && j < len(l) ==> acc(&l[ j ])
  ensures res != -1 ==> i <= res && res < len(l) &&
    (forall j int :: i <= j && j < res ==> l[ j ] != val) && l[ res ] == val
{
  if l[ i ] == val             { return i }
  else if i >= len(l) - 1      { return -1 }
  else                         { return indexOf(l, i+1, val) }
}
```

- No run-time errors
- No data races
- Functional properties
- Termination
- I/O behavior
- Secure information flow

Gobra Toolchain

GO

```
func main(a [ ]int) int {
    var b int = a[3]
    …
}
```

Error: Slice access a[3] might be out of bounds

gobra

Program translation

Error translation

VIPER

```
method main(a: Ref)
        returns (res: Int)
  requires slice(a) ...
{
  var b: Int
  b := lookup(a, 3)
  …
}
```

Error: Insufficient permission to access loc(a, 3).val

Z3

✔

Separation Logic

- Associate each heap location with a permission
- Permissions are held by method executions
- Access to a memory location requires permission

```
func indexOf(l [ ]int, i, val int) (res int)
  requires forall j int :: i <= j && j < len(l) ==> acc(&l[ j ])
```

- Permissions can be transferred, but not duplicated or forged
- Guarantees memory safety, data race freedom, enables local reasoning

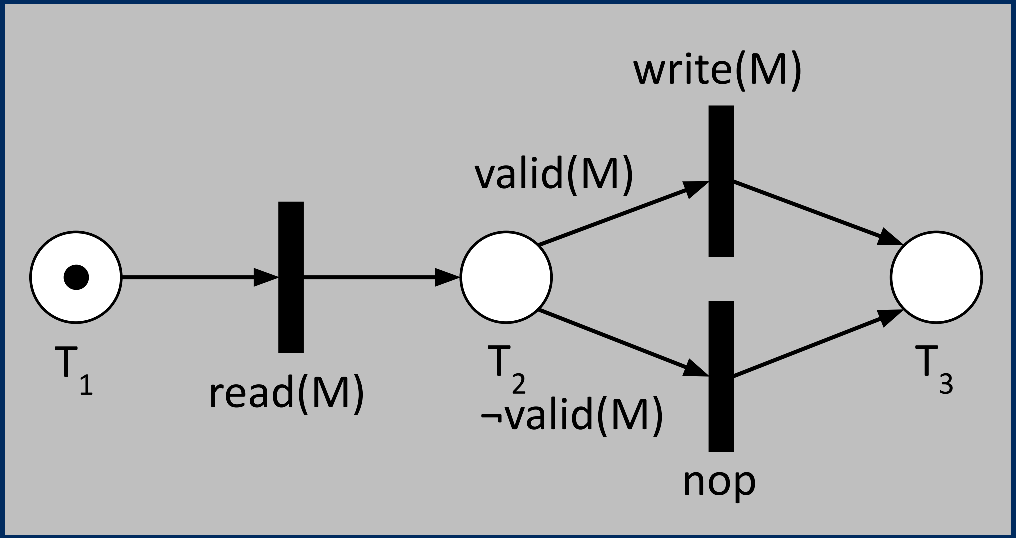
I/O Permissions

- Permissions can be used to reason about resources
- Here: permission to perform an I/O operation

```
func write(value string)
  requires writeIO(value)
```

I/O Behavior as Petri Nets

- Adaptation of work by Penninckx et al., ESOP 15
- Petri nets specify permitted I/O behavior
  - Traces of basic I/O operations
  - Sequences, parallelism, non-determinism
- Petri nets are encoded as (recursive) predicates



predicate router(T_1) {
  $\forall M\ \exists T_2, T_3 \cdot$
    readIO(T_1,M,T_2) *
    (valid(M) ⇒ writeIO(T_2,M,T_3)) *
    (¬ valid(M) ⇒ nop(T_2,T_3)) *
    router(T_3)
}

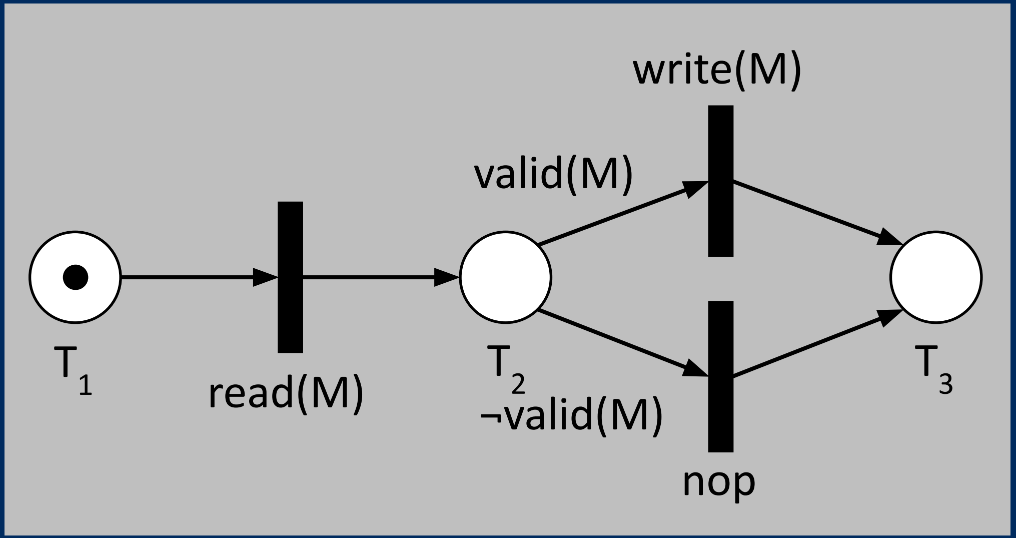
Specification of I/O Behavior

- Basic I/O operations
  - Require I/O permission
  - Require token in appropriate place
  - Advance token

```
func write(value string)
  requires token(T) * writeIO(T, value, T')
  ensures token(T')
```

- Method precondition characterizes permitted I/O behavior

```
func main()
  requires token(T) * router(T)
```




```
predicate router(T₁) {
  ∀M ∃T₂, T₃ ·
    readIO(T₁,M,T₂) *
    (valid(M) ⇒ writeIO(T₂,M,T₃)) *
    (¬ valid(M) ⇒ nop(T₂,T₃)) *
    router(T₃)
}
```

From Model to I/O Specification

- Refine component model to have an event for each basic I/O operation

I/O event write(val)

func write(value **string**)

- Encode entire event system as recursive predicate

event drop(M)
guard ¬valid(M)
action
  buf := buf \ { M }

predicate system(T_1, state) {
  (∀args · guard(args, state) ⇒
      ∃T_2 · opIO(T_1,args,T_2) * system(T_2, state')) *
  …

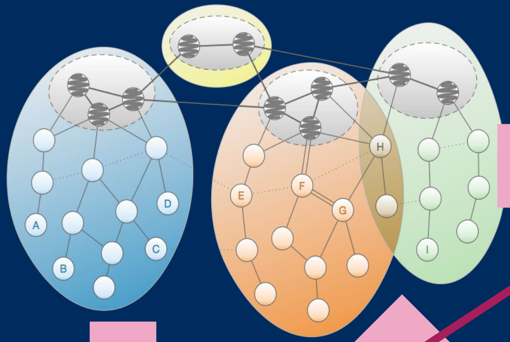
predicate router(T_1, buf) {
  (∀M · ¬valid(M) ⇒ router(T_1, buf \ { M })) *
  …

Status of Code Verification

- Completed verification of SCION router (4,700 LoC)
    - Memory safety
    - Functional correctness
    - I/O behavior
    - Termination
- 13,400 lines of annotations (2.8 LoS per LoC)
- Required only three code changes

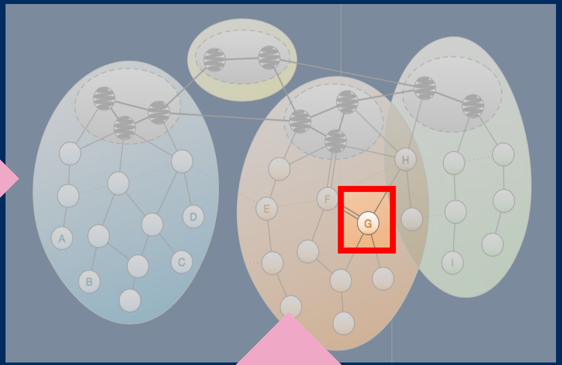
- Identified 13 confirmed issues related to memory safety, functional correctness, and I/O behavior
(plus 2 performance issues)
- Despite extensive code reviews, testing, and fuzzing

Mathematical model of entire network



Refinement

Mathematical model of border router

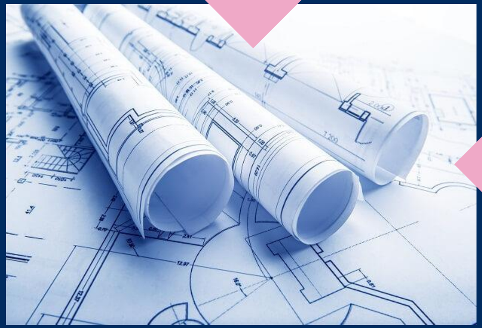


Equivalence

- Developed by stepwise refinement
- Verified in Isabelle

- Properties encoded in permission logic
- Verified in Gobra/Viper

- Transition system encoded as I/O specification
- Verified in Isabelle



Router specification

Verification

Router implementation