Scaling Security Protocol Verification

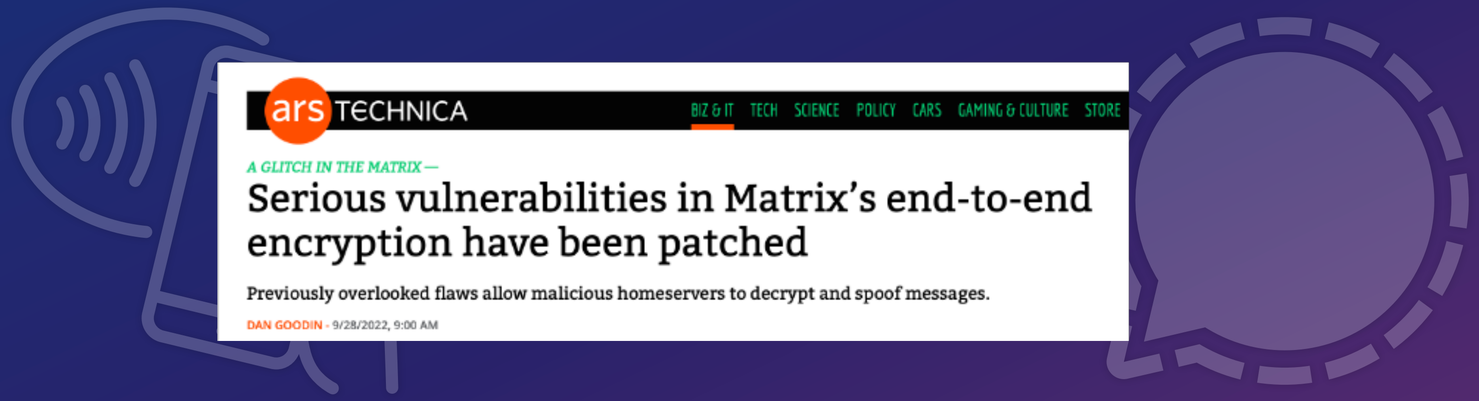
Linard Arquint

Centre for Cyber Trust — 27. Nov. ‘25

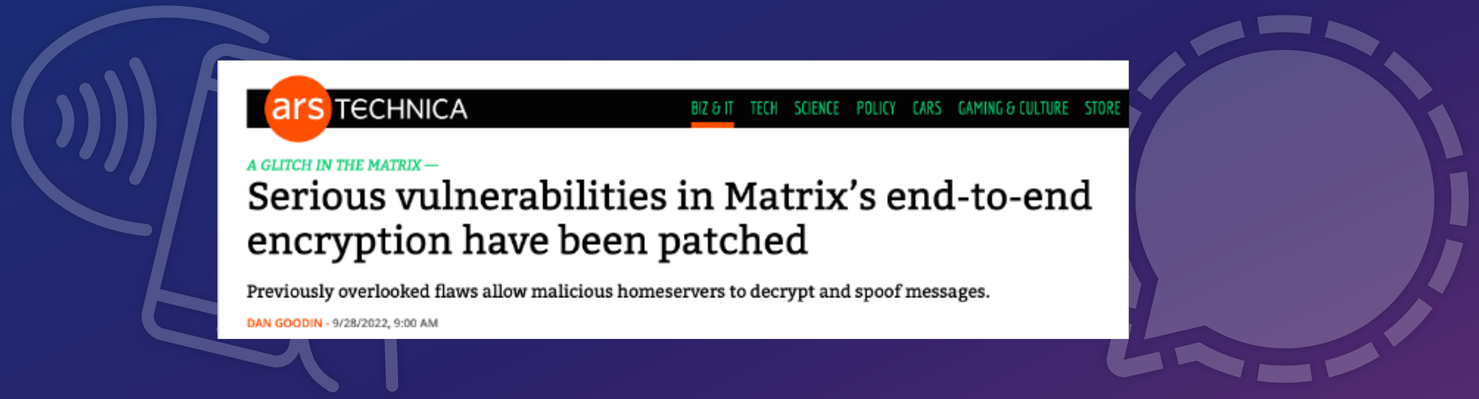
ETH zürich

Motivation

Motivation


ars TECHNICA
BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE STORE
A GLITCH IN THE MATRIX —
Serious vulnerabilities in Matrix's end-to-end encryption have been patched
Previously overlooked flaws allow malicious homeservers to decrypt and spoof messages.
DAN GOODIN - 9/28/2022, 9:00 AM

Motivation



```
if ((err = SSLHashSHA1.update(…)) != 0)
  goto fail;
if ((err = SSLHashSHA1.update(…)) != 0)
  goto fail;
  goto fail;
if ((err = SSLHashSHA1.final(…)) != 0)
  goto fail;
```

State of the Art

Protocol model

- Formal protocol description
- Pen and paper proofs
- Automatic tools: Tamarin, ProVerif, …
- Successfully applied to many protocols

State of the Art

Protocol model

- Formal protocol description
- Pen and paper proofs
- Automatic tools: Tamarin, ProVerif, …
- Successfully applied to many protocols

Code generation

- Generated code cannot be optimized
- Integration into a codebase is a soundness threat

Gap

Existing implementation

Gap

Correct protocol implementation?

Existing implementation

Gap

Correct protocol implementation?

Free of buffer overflows?

Existing implementation

Gap

Existing implementation

Correct protocol implementation?

Free of buffer overflows?

Is it safe, i.e., no panics, divisions by zero, …?

Earlier Work

Model refinement

S&P ‘23

Earlier Work

Model refinement

S&P ‘23

Trace invariant

CCS ‘23

This Talk

Scaling to large codebases

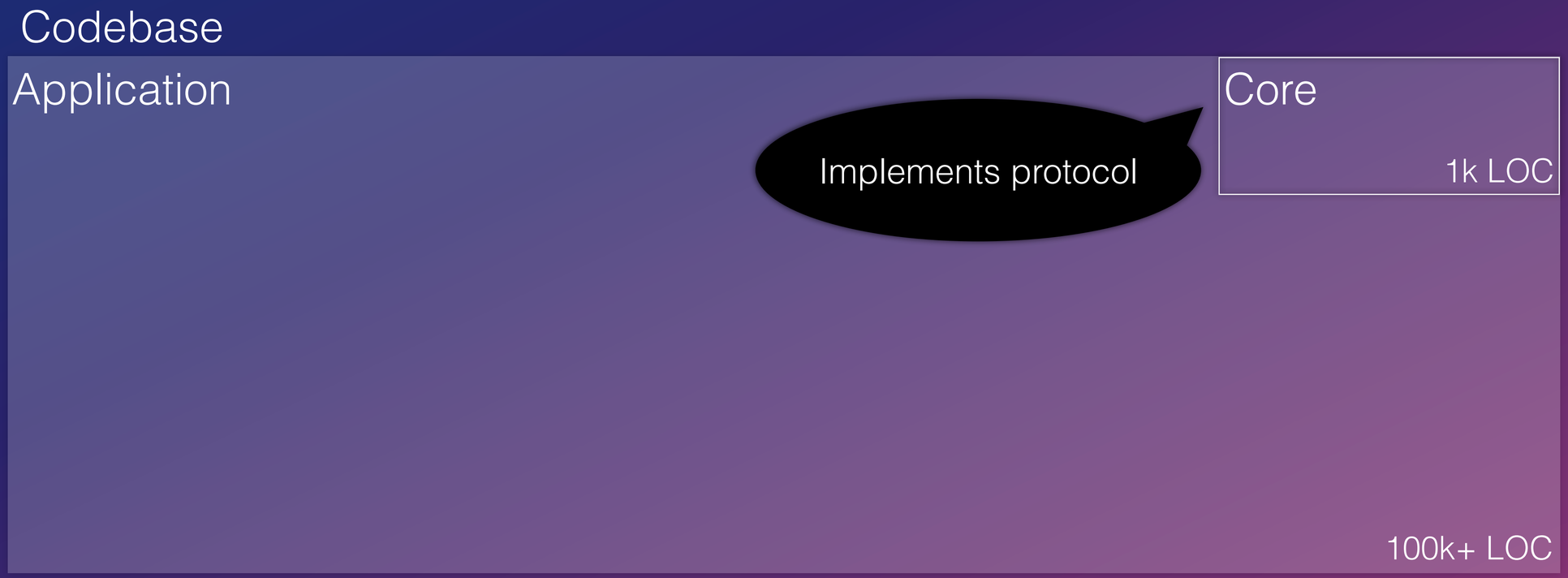
S&P ‘26

Problem

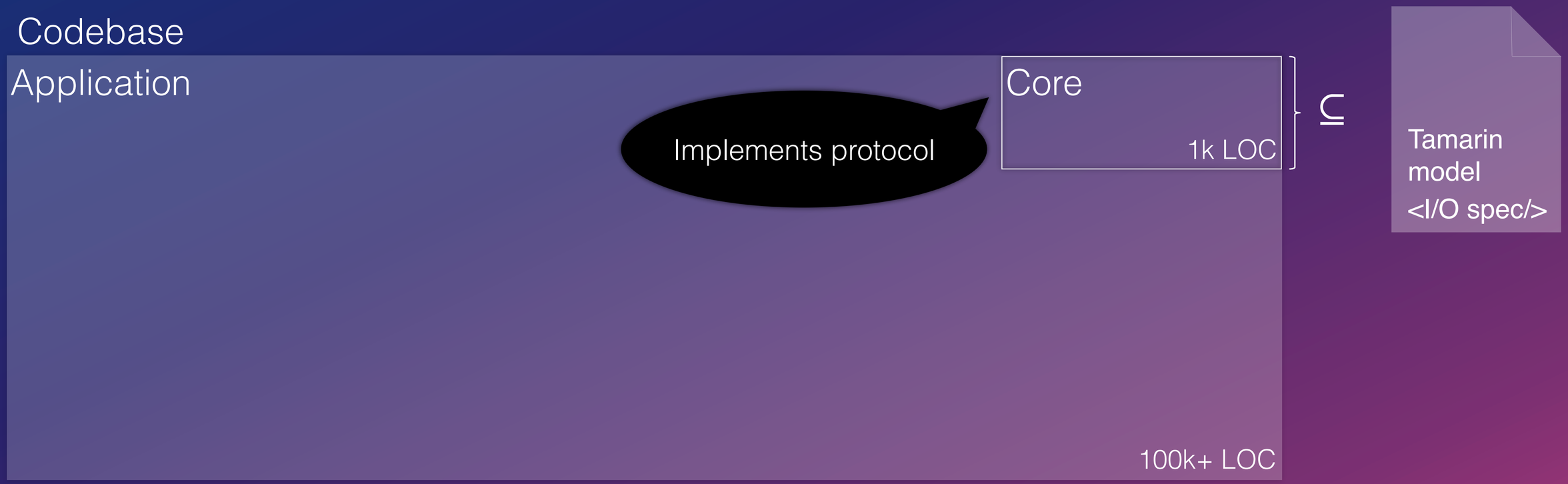
Codebase

100k+ LOC

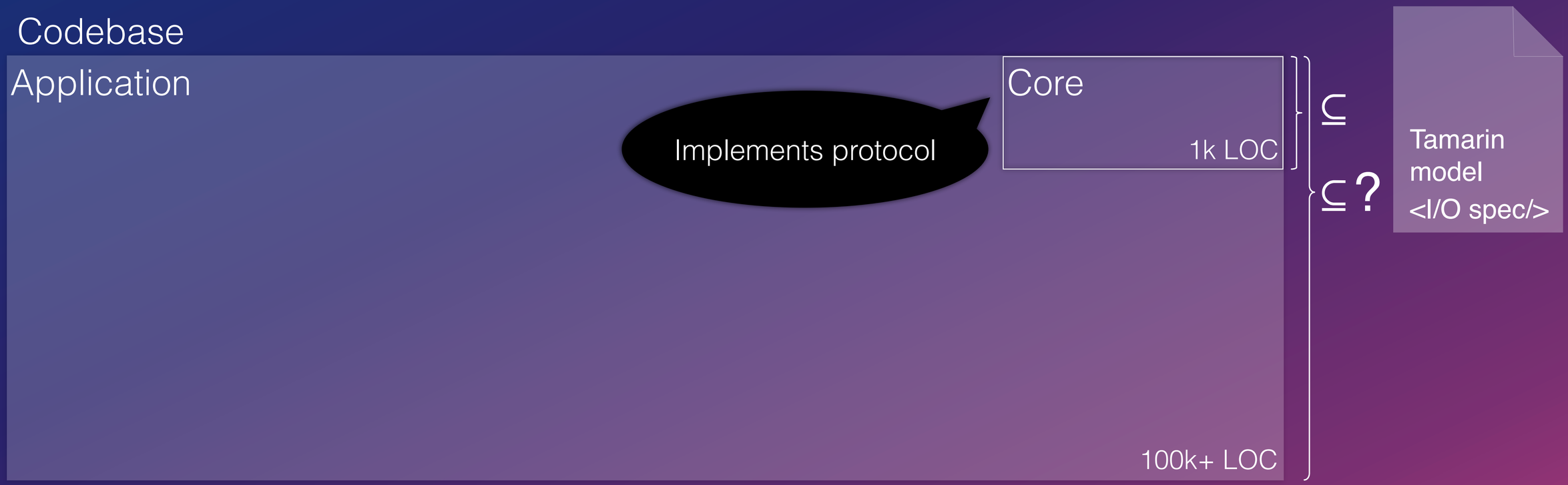
Problem


Codebase
Application
Implements protocol
Core
1k LOC
100k+ LOC

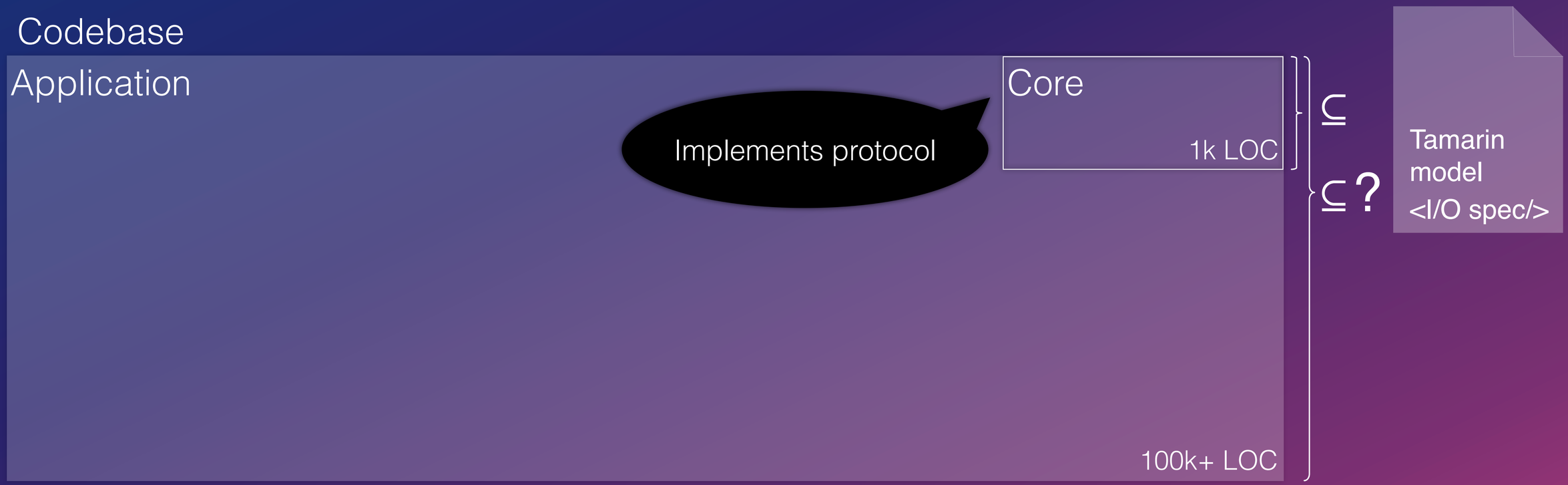
Problem


Codebase
Application
Implements protocol
Core
1k LOC
100k+ LOC
⊆
Tamarin
model
<I/O spec/>

Problem

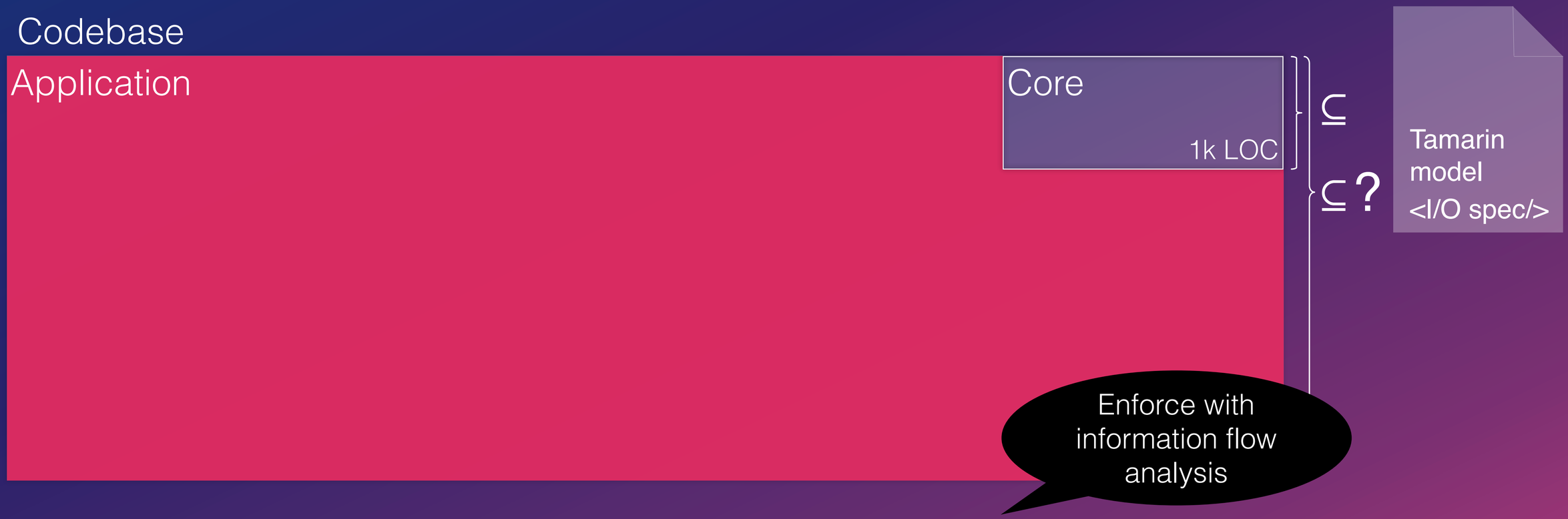

Codebase
Application
Implements protocol
Core
1k LOC
100k+ LOC
⊆
⊆ ?
Tamarin
model
<I/O spec/>

Problem



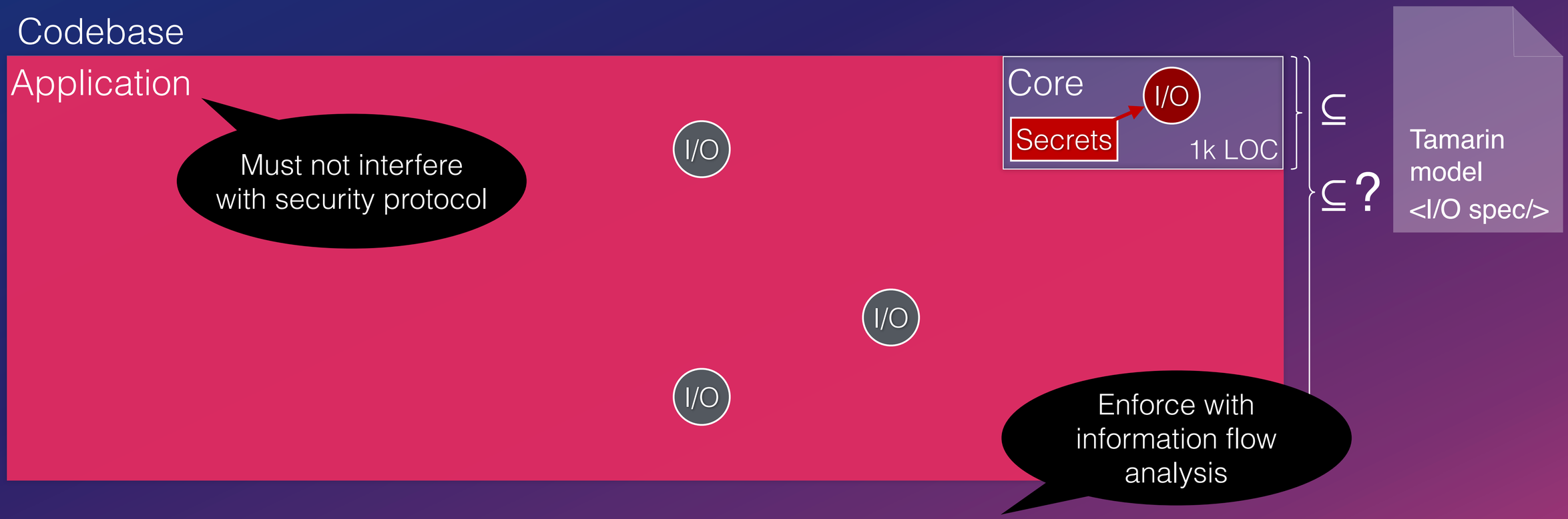
Sources of unsoundness

1. I/O Independence



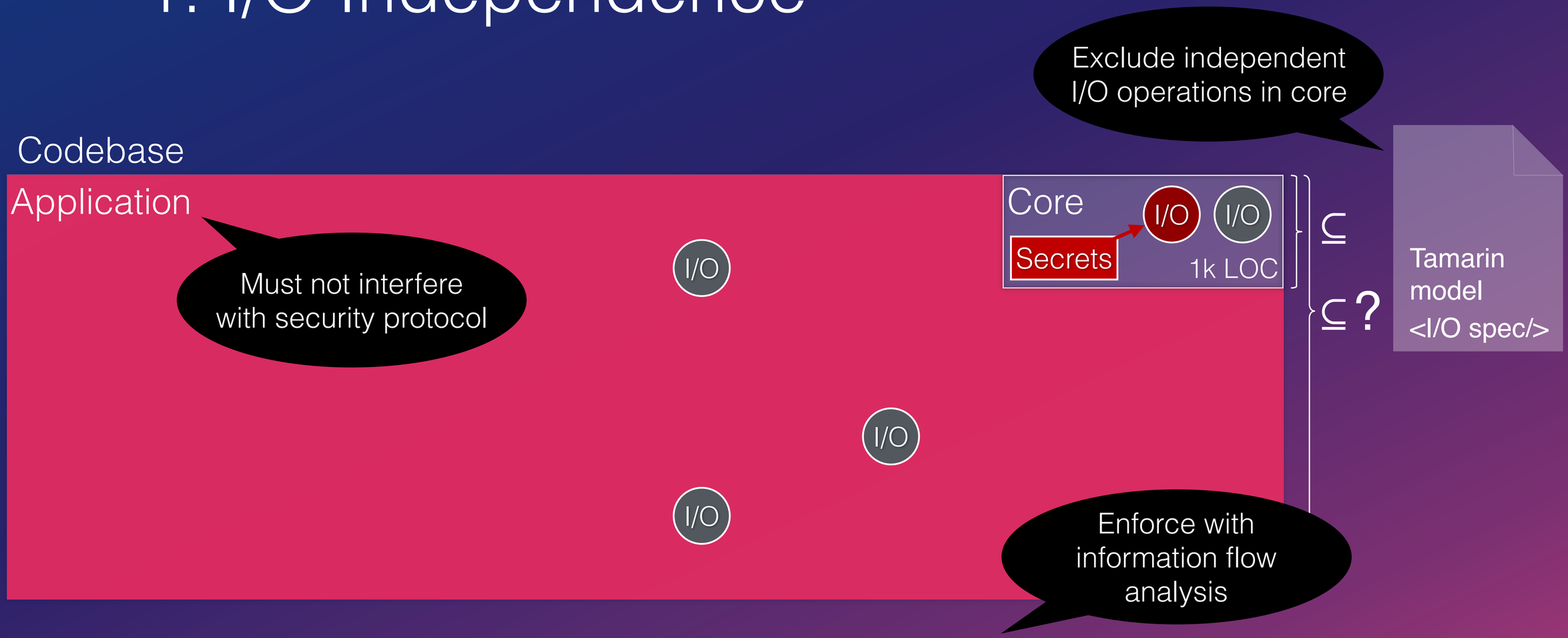
➔ I/O operations in application must be independent of core secrets

1. I/O Independence



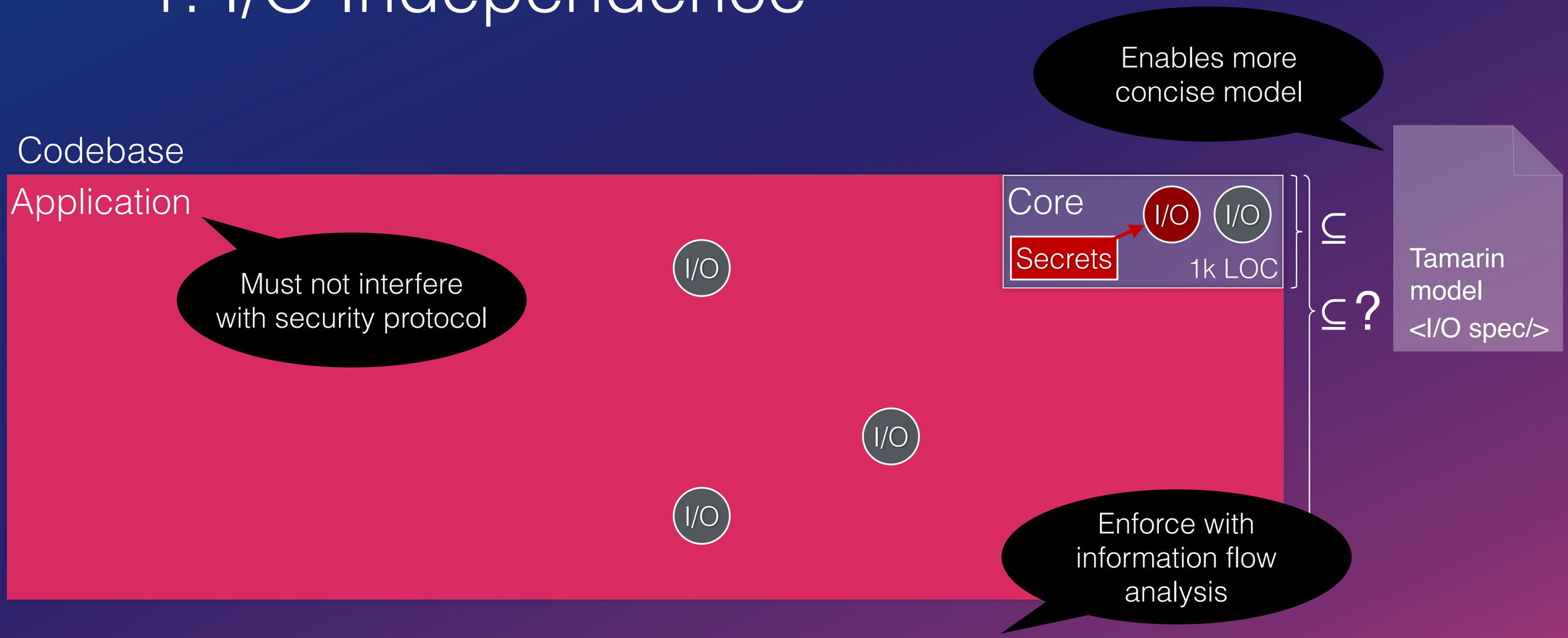
➔ I/O operations in application must be independent of core secrets

1. I/O Independence



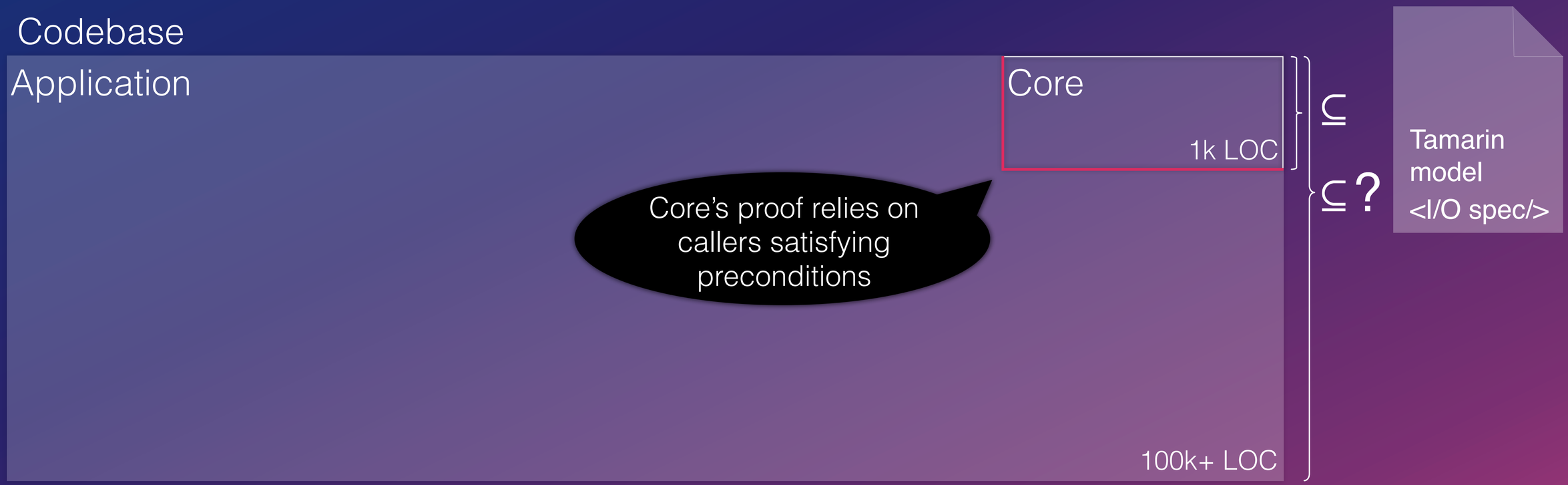
➔ I/O operations in application must be independent of core secrets

1. I/O Independence



➔ I/O operations in application must be independent of core secrets

2. Discharging Core's Assumptions



➔ Syntactically restrict specifications to make them amenable to static analyses

2. Discharging Core's Assumptions

```
//@ requires p != nil ==> acc(p)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
func NewCore(p *int) (c *Core)
```

```
//@ requires p != nil ==> acc(p)
//@ requires c != nil ==> Inv(c)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
//@ ensures  r != nil ==> acc(r)
func (c *Core) APIFn(p *int) (r *int)
```

➔ Syntactically restrict specifications to make them amenable to static analyses

Soundness

```
func main() {

  c := NewCore()

  a := 42



  c.APIFn(&a)

}
```

```
//@ requires p != nil ==> acc(p)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
func NewCore(p *int) (c *Core)
```

```
//@ requires p != nil ==> acc(p)
//@ requires c != nil ==> Inv(c)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
//@ ensures  r != nil ==> acc(r)
func (c *Core) APIFn(p *int) (r *int)
```

Soundness

```
func main() {
  //@ L := {}; I := {}

  c := NewCore()
  //@ I = I ∪ {c}

  a := 42
  //@ L = L ∪ {&a}

  //@ L = L \ {&a}
  //@ I = I \ {c}
  c.APIFn(&a)
  //@ L = L ∪ {&a}
  //@ I = I ∪ {c}
}
```

```
//@ requires p != nil ==> acc(p)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
func NewCore(p *int) (c *Core)
```

```
//@ requires p != nil ==> acc(p)
//@ requires c != nil ==> Inv(c)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
//@ ensures  r != nil ==> acc(r)
func (c *Core) APIFn(p *int) (r *int)
```

Soundness

```
func main() {
  //@ L := {}; I := {}

  c := NewCore()
  //@ I = I ∪ {c}

  a := 42
  //@ L = L ∪ {&a}

  //@ L = L \ {&a}
  //@ I = I \ {c}
  c.APIFn(&a)
  //@ L = L ∪ {&a}
  //@ I = I \ {c}
}
```

```
//@ requires p != nil ==> acc(p)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
func NewCore(p *int) (c *Core)
```

```
//@ requires p != nil ==> acc(p)
//@ requires c != nil ==> Inv(c)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
//@ ensures  r != nil ==> acc(r)
func (c *Core) APIFn(p *int) (r *int)
```

Program invariant $\texttt{pinv} \stackrel{\text{def}}{=} (\forall a \in L.\ \texttt{acc}(a)) \bigstar (\forall c \in I.\ \texttt{Inv}(c))$

Program invariant $\texttt{pinv} \stackrel{\text{def}}{=} (\forall a \in L.\ \texttt{acc}(a)) \bigstar (\forall c \in I.\ \texttt{Inv}(c))$

```
func main() {
  //@ L := {}; I := {}

  c := NewCore()
  //@ I = I ∪ {c}

  a := 
  //@ 

  //@ L = L \ {&a}
  //@ I = I \ {c}
  c.APIFn(&a)
  //@ L = L ∪ {&a}
  //@ I = I \ {c}
}
```

$$\frac{x \in L \cup G}{\vdash \{\texttt{pinv}\}\, \mathbb{G}(\texttt{*x := e})\, \{\texttt{pinv}\}}$$

```
//@ requires p != nil ==> acc(p)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
func NewCore(p *int) (c *Core)
```

```
//@ requires p != nil ==> acc(p)
//@ requires c != nil ==> Inv(c)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
//@ ensures  r != nil ==> acc(r)
func (c *Core) APIFn(p *int) (r *int)
```

Program invariant $\texttt{pinv} \triangleq (\forall a \in L.\ \texttt{acc}(a)) \bigstar (\forall c \in I.\ \texttt{Inv}(c))$

```
func main() {
  //@ L :=
  c := NewCore()
  //@ I = I ∪ {c}
  a :=
  //@ L = L ∪ {&a}
  //@ L = L \ {&a}
  //@ I = I \ {c}
  c.APIFn(&a)
  //@ L = L ∪ {&a}
  //@ I = I \ {c}
}
```

x is may not alias core memory

$$\frac{x \in L \cup G}{\vdash \{\texttt{pinv}\}\, \mathbb{G}(*x := e)\, \{\texttt{pinv}\}}$$

```
//@ requires p != nil ==> acc(p)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
func NewCore(p *int) (c *Core)
```

```
//@ requires p != nil ==> acc(p)
//@ requires c != nil ==> Inv(c)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
//@ ensures  r != nil ==> acc(r)
func (c *Core) APIFn(p *int) (r *int)
```

Program invariant $\texttt{pinv} \triangleq (\forall a \in L.\ \texttt{acc}(a)) \bigstar (\forall c \in I.\ \texttt{Inv}(c))$

```
func main() {
  //@ L :=
  c := NewCore()
  //@ I = I ∪ {c}
  a :=
  //@ L = L ∪ {&a}

  //@ L = L \ {&a}
  //@ I = I \ {c}
  c.APIFn(&a)
  //@ L = L ∪ {&a}
  //@ I = I \ {c}
}
```

x is may not alias core memory

$$\frac{x \in L \cup G}{\vdash \{\texttt{pinv}\}\, \mathbb{G}(*x := e)\, \{\texttt{pinv}\}}$$

```
//@ requires p
//@ ensures  p
//@ ensures  c
func NewCore(p *int) (c *Core)
```

Pointer Analysis
Make sure there are no writes to memory covered by `Inv()`

```
//@ requires p != nil ==> acc(p)
//@ requires c != nil ==> Inv(c)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
//@ ensures  r != nil ==> acc(r)
func (c *Core) APIFn(p *int) (r *int)
```

Program invariant $\texttt{pinv} \triangleq (\forall a \in L.\ \texttt{acc}(a)) \bigstar (\forall c \in I.\ \texttt{Inv}(c))$

```
func main() {
  //@ L :=
  c := NewCore()
  //@ I = I ∪ {c}
  a :=
  //@ L = L ∪ {&a}
  //@ L = L \ {&a}
  //@ I = I \ {c}
  c.APIFn(&a)
  //@ L = L ∪ {&a}
  //@ I = I \ {c}
}
```

x is may not alias core memory

$$\frac{x \in L \cup G}{\vdash \{\texttt{pinv}\}\, \mathbb{G}(\texttt{*x := e})\, \{\texttt{pinv}\}}$$

$$\frac{\&a \in L \qquad c \in I}{\vdash \{\texttt{pinv}\}\, \mathbb{G}(\texttt{c.APIFn(\&a)})\, \{\texttt{pinv}\}}$$

```
//@ requires p
//@ ensures  p
//@ ensures  c
func NewCore(p *int) (c *Core)
```

```
//@ requires p != nil ==> acc(p)
//@ requires c != nil ==> Inv(c)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
//@ ensures  r != nil ==> acc(r)
func (c *Core) APIFn(p *int) (r *int)
```

Pointer Analysis
Make sure there are no writes to memory covered by `Inv()`

Program invariant $\texttt{pinv} \triangleq (\forall a \in L.\ \texttt{acc}(a)) \bigstar (\forall c \in I.\ \texttt{Inv}(c))$

```
func main() {
  //@ L :=
  c := NewCore()
  //@ I = I ∪ {c}
  a :=
  //@ L = L \ {&a}
  //@ L = L \ {c}
}
```

x is may not alias core memory

$$\frac{x \in L \cup G}{\vdash \{\texttt{pinv}\}\,\mathbb{G}(\texttt{*x := e})\,\{\texttt{pinv}\}}$$

&a is may not alias core memory & must not escape current thread

c is must not escape current thread

$$\frac{\&a \in L \qquad c \in I}{\vdash \{\texttt{pinv}\}\,\mathbb{G}(\texttt{c.APIFn(\&a)})\,\{\texttt{pinv}\}}$$

Pointer Analysis
Make sure there are no writes to memory covered by `Inv()`

```
//@ requires p
//@ ensures  p
//@ ensures  c != nil ==> Inv(c)
func NewCore(p *int) (c *Core)
```

```
//@ requires p != nil ==> acc(p)
//@ requires c != nil ==> Inv(c)
//@ ensures  p != nil ==> acc(p)
//@ ensures  c != nil ==> Inv(c)
//@ ensures  r != nil ==> acc(r)
func (c *Core) APIFn(p *int) (r *int)
```

Program invariant $\texttt{pinv} \triangleq (\forall a \in L.\ \texttt{acc}(a)) \mathbin{\bigstar} (\forall c \in I.\ \texttt{Inv}(c))$

```
func main() {
  //@ L :=
  c := NewCore()
  //@ I = I ∪ {c}
  a :=
  //@ L = L \ {&a}
  //@ L = L \ {c}
}
```

x is may not alias core memory

$$\frac{x \in L \cup G}{\vdash \{\texttt{pinv}\}\,\mathbb{G}(\texttt{*x := e})\,\{\texttt{pinv}\}}$$

&a is may not alias core memory & must not escape current thread

c is must not escape current thread

$$\frac{\&a \in L \qquad c \in I}{\vdash \{\texttt{pinv}\}\,\mathbb{G}(\texttt{c.APIFn(\&a)})\,\{\texttt{pinv}\}}$$

```
//@ requires p
//@ ensures  p
//@ ensures  c != nil ==> Inv(c)
func NewCore(p *int) (c *Core)

//@ requires p
//@ requires
//@ ensures  p
//@ ensures  c !=
//@ ensures  r != nil ==> acc(r)
func (c *Core) APIFn(p *int) (r *int)
```

Pointer Analysis
Make sure there are no writes to memory covered by `Inv()`

Escape Analysis
Make sure heap locations remain thread-local

Program invariant $\texttt{pinv} \triangleq (\forall a \in L.\ \texttt{acc}(a)) \mathbin{\bigstar} (\forall c \in I.\ \texttt{Inv}(c))$

```
func main() {
  //@ L :=
  c := NewCore()
  //@ I = I ∪ {c}
  a :=
  //@ L = L \ {&a}
  //@ L = L \ {c}
}
```

x is may not alias core memory

$$\frac{x \in L \cup G}{\vdash \{\texttt{pinv}\}\, \mathbb{G}(\texttt{*x := e})\, \{\texttt{pinv}\}}$$

&a is may not alias core memory & must not escape current thread

c is must not escape current thread

$$\frac{\&a \in L \qquad c \in I}{\vdash \{\texttt{pinv}\}\, \mathbb{G}(\texttt{c.APIFn(\&a)})\, \{\texttt{pinv}\}}$$

```
//@ requires p
//@ ensures  p
//@ ensures  c !=
func NewCore(p *int) (c *Core)

//@ requires p
//@ requires
//@ ensures  p
//@ ensures  c
//@ ensures  r !=
func (c *Core) APIFn(p *int) (r *int)
```

Pointer Analysis
Make sure there are no writes to memory covered by `Inv()`

Escape Analysis
Make sure heap locations remain thread-local

Data Race Freedom
Provides linearizability of heap accesses outside core

Program invariant `pinv` $\stackrel{\text{def}}{=} (\forall a \in L.\ acc(a)) \bigstar (\forall c \in I.\ Inv(c))$

```
func main() {
  //@ L :=
  c := NewCore()
  //@ I = I ∪ {c}
  a :=
  //@ L
  //@ L = L \ {&a}
  //@ I = I \ {c}
}
```

x is may not alias core memory

$$\frac{x \in L \cup G}{\vdash \{pinv\}\ \mathbb{G}(*x := e)\ \{pinv\}}$$

&a is may not alias core memory & must not escape current thread

c is must not escape current thread

$$\frac{\&a \in L \qquad c \in I}{\vdash \{pinv\}\ \mathbb{G}(c.APIFn(\&a))\ \{pinv\}}$$

```
//@ requires p
//@ ensures  p
//@ ensures  c != nil ==> Inv(c)
func NewCore(p *int) (c *Core)

//@ requires p
//@ requires
//@ ensures  p
//@ ensures  c
//@ ensures  r != nil ==> acc(r)
func (c *Core) APIFn(p *int) (r *int)
```

Pointer Analysis
Make sure there are no writes to memory covered by `Inv()`

Escape Analysis
Make sure heap locations remain thread-local

Data Race Freedom
Provides linearizability of heap accesses outside core

Soundness

Whole codebase proof

Separation logic proof for the whole codebase with side-conditions dischargeable by static analyses

$$\frac{\text{side-conditions}}{\vdash \{\,\texttt{pinv}\,\}\,\mathbb{G}(\texttt{prog})\,\{\,\texttt{pinv}\,\}}$$

ghost code tracking heap locations, core invariant, and I/O specification

I/O independence

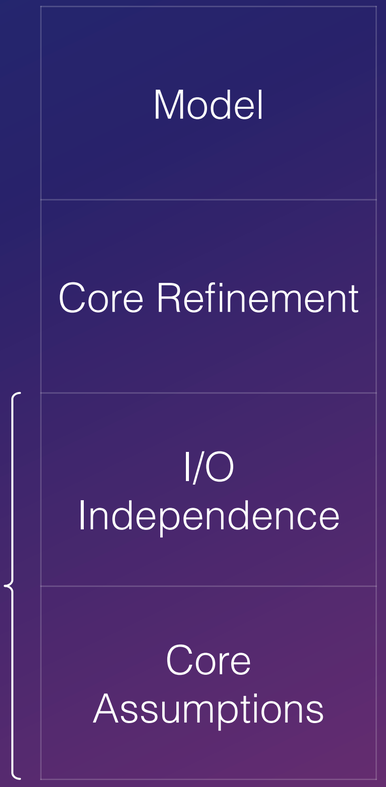
Secret-independent I/O operations refine the symbolic attacker

Evaluation — AWS SSM Agent

Tamarin

gobra

Argot



Evaluation — AWS SSM Agent

Tamarin

gobra

Argot

		LoC	Proof annotations	Verification time [min]	Effort
Tamarin	Model	319	75	3.3	< 2 pms
gobra	Core Refinement	749	2761 + 1064 (generated)	1.2	< 3 pms
Argot	I/O Independence	~105k		0.5	< 0.5 pms
Argot	Core Assumptions	~105k		2.2	< 1.5 pms

Conclusions

Implements protocol

Codebase

Application

Core

1k LOC

100k+ LOC

Static analyses enforce that Application uses Core correctly

Codebase may perform additional *independent* I/O operations

Tamarin model

<I/O spec/>

Tamarin

gobra

Argot