Incorrectness Logic & Under-Approximation: Foundations of Bug Detection

Azalea Raad

Imperial College London

Principles of Programming Languages

16 January 2023

azalea@imperial.ac.uk SoundAndComplete.org @azalearaad

State of the Art: ***Correctness***

- Lots of work on ***reasoning*** for proving ***correctness***
  - ➡ Prove the ***absence of bugs***
  - ➡ ***Over-approximate*** reasoning
  - ➡ ***Compositionality***

    in ***code*** ⇒ reasoning about ***incomplete components***

    in ***resources*** accessed ⇒ spatial locality
  - ➡ ***Scalability*** to large teams and codebases

Hoare Logic (HL)

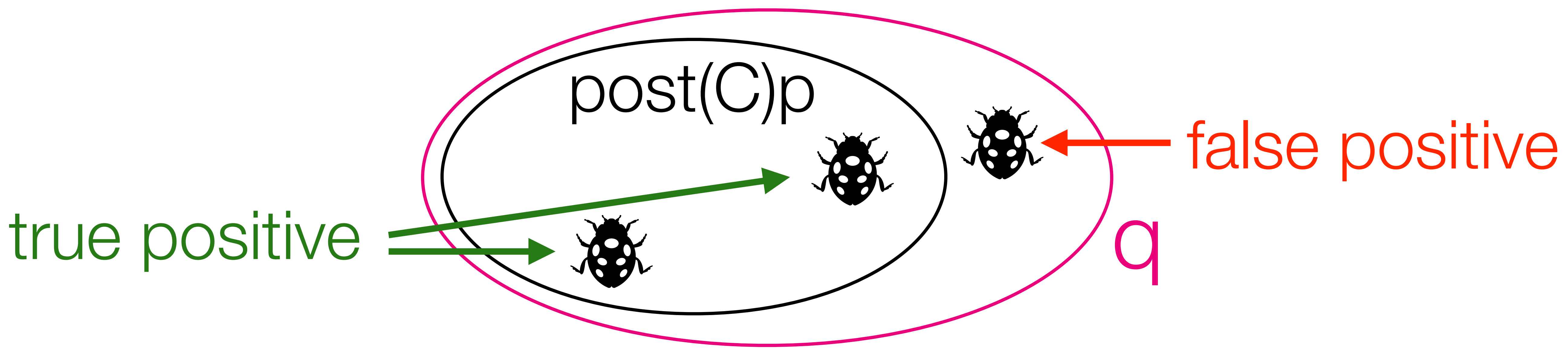
Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

For all states s *in* p
if running C *on* s *terminates in* s', *then* s' *is in* q

Hoare Logic (HL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

q *over-approximates* post(C)p



“Don’t spam the developers!”

Incorrectness Logic:

A Formal Foundation

for

Bug Catching

Part I.
Incorrectness Logic (IL)
&
Incorrectness Separation Logic (ISL)

Incorrectness Logic (IL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

For all states s in p
if running C *on* s *terminates in* s', *then* s' *is in* q

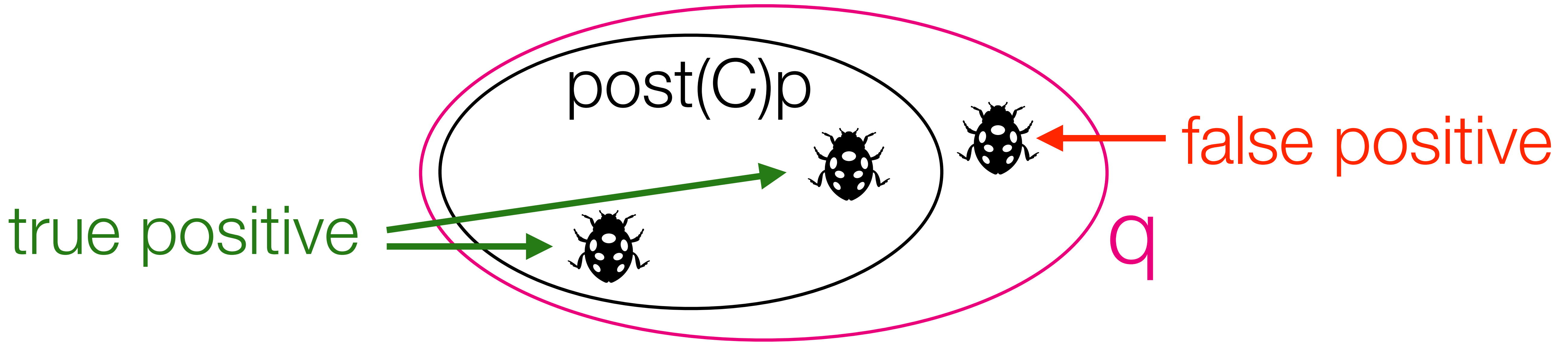
Incorrectness triples $[p]\ C\ [q]$ *iff* $post(C)p \supseteq q$

For all states s in q
s *can be reached by running* C *on some* s' *in* p

Incorrectness Logic (IL)

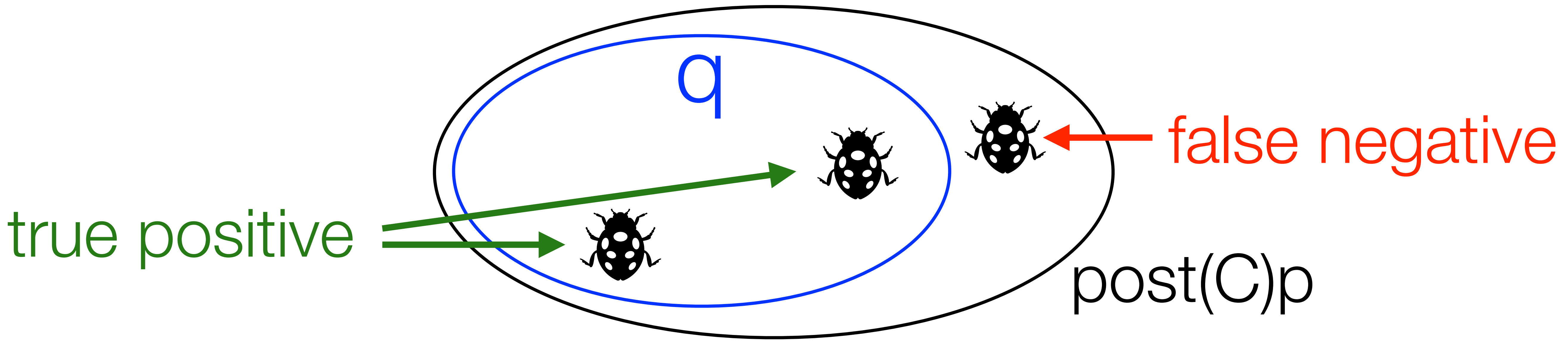
Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

q *over-approximates* post(C)p



Incorrectness triples $[p]\ C\ [q]$ *iff* $post(C)p \supseteq q$

q *under-approximates* post(C)p



Incorrectness Logic (IL)

$[p]\ C\ [\varepsilon\!:\ q]$

ε: exit condition

ok: normal execution

er : erroneous execution

$[y{=}v]\ x{:=}y\ [ok\!:\ x{=}y{=}v]$

$[p]$ error( ) $[er\!:\ p]$

Incorrectness Logic (IL)

$[p]\ C\ [\varepsilon: q]$ *iff* $\mathrm{post}(C, \varepsilon)p \supseteq q$

Equivalent Definition (reachability)

$[p]\ C\ [\varepsilon: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

IL Proof Rules and Principles (Sequencing)

$$\frac{[p]\ C_1\ [er: q]}{[p]\ C_1; C_2\ [er: q]}$$

$$\frac{[p]\ C_1\ [ok: r] \qquad [r]\ C_2\ [\varepsilon: q]}{[p]\ C_1; C_2\ [\varepsilon: q]}$$

- **Short-circuiting** semantics for errors

IL Proof Rules and Principles (Branches)

$$\frac{[p]\ C_i\ [\varepsilon\!: q] \qquad \textbf{some}\ i \in \{1, 2\}}{[p]\ C_1 + C_2\ [\varepsilon\!: q]}$$

- **Drop paths/branches** (this is a **sound** under-approximation)
- **Scalable** bug detection!

$[p]\ C\ [\varepsilon\!: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]_\varepsilon$

Example

$[\, y=0 \,]$ **if** (`is-even(x)`) $[ok\!: y=42\,]$?

y:= 42

$[p]\ C\ [\varepsilon\!: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s',s) \in [C]\varepsilon$

Example

$$\overset{p}{[\boxed{y=0}]}\ \textbf{if}\ (\texttt{is-even(x)})\ \overset{q}{[ok{:}\ \boxed{y=42}]}$$

$$y := 42$$

?

$q = \{(x=0, y=42), (x=1, y=42), (x=2, y=42), \dots\}$

$[p]\ C\ [\varepsilon{:}\ q]$ *iff* $\forall s \in q.\ \exists s' \in p.\ (s',s) \in [C]\varepsilon$

Example

$$\overset{p}{[\boxed{y=0}]}\ \textbf{if}\ (\texttt{is-even(x)})\ \ \overset{q}{[ok\!:\boxed{y=42}]}$$

y:= 42

✘

q={(x=0, y=42), (x=1, y=42), (x=2, y=42), …}

✔ ✘ ✔

[p] C [ε: q] *iff* $\forall$ s $\in$ q. $\exists$ s' $\in$ p. (s',s) $\in$ [C]ε

Example

$$\overset{p}{[\boxed{y=0}]}\ \textbf{if}\ (\texttt{is-even}(x))\ \ y:=42\ \ \overset{q}{[ok{:}\ \boxed{y=42 \wedge even(x)}]}$$ ✔

$$q=\{(x=0,\ y=42),\ (x=2,\ y=42),\ (x=4,\ y=42),\ \ldots\}$$

✔ ✔ ✔

$[p]\ C\ [\varepsilon{:}\ q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s',s) \in [C]\varepsilon$

IL Proof Rules and Principles (Loops)

$$\frac{}{[p]\ C^*\ [ok: p]}\ (\text{Unroll-Zero})$$

$$\frac{[p]\ C^*; C\ [\varepsilon: q]}{[p]\ C^*\ [\varepsilon: q]}\ (\text{Unroll-Many})$$

- **Bounded unrolling of loops** (this is a **sound** under-approximation)
- **Scalable** bug detection!

$[p]\ C\ [\varepsilon: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

IL Proof Rules and Principles (Loops continued)

$$\frac{\forall n \in \mathbb{N}.\ [p(n)]\ C\ [ok: p(n+1)] \qquad k \in \mathbb{N}}{[p(0)]\ C^*\ [ok: p(k)]}\ \text{(Backwards-Variant)}$$

- Loop **invariants** are inherently **over-approximate**
- Reason about loops **under-approximately** via **sub-variants**

$[p]\ C\ [\varepsilon: q]$ *iff* $\forall\ s \in q.\ \exists\ s' \in p.\ (s', s) \in [C]\varepsilon$

Example

[x=0]   (x++)*; **if** (`x==2,000,000`) error;   [er: x=2,000,000]

Example

[x=0] (x++)*; **if** (`x==2,000,000`) error; [er: x=2,000,000]

[x=0]

(x++)*; // Backwards-Variant

[ok: x=2,000,000]

if (`x==2,000,000`)

error;

p(n): x=n

$$\frac{\forall n \in \mathbb{N}.\ [p(n)]\ C\ [ok: p(n+1)] \qquad k \in \mathbb{N}}{[p(0)]\ C^*\ [ok: p(k)]}\ \text{(Backwards-Variant)}$$

Example

$[\, x=0 \,]$ $\quad$ (x++)*; **if** (x==2,000,000) error; $\quad$ $[er: x=2,000,000 \,]$ ✔

$[\, x=0 \,]$

(x++)*; // Backwards-Variant

$[ok: x=2,000,000 \,]$

if (x==2,000,000)

error;

$[er: x=2,000,000 \,]$

$\boxed{p(n): x=n}$

$$\frac{\forall n \in \mathbb{N}.\ [p(n)]\ C\ [ok: p(n+1)] \qquad k \in \mathbb{N}}{[p(0)]\ C^*\ [ok: p(k)]}\ \text{(Backwards-Variant)}$$

IL Proof Rules and Principles (Consequence)

$$\frac{p' \subseteq p \quad [p']\ C\ [\varepsilon: q'] \quad q' \supseteq q}{[p]\ C\ [\varepsilon: q]}\ \text{(Cons)}$$

- **Shrink** the **post** (e.g. drop disjuncts)
- **Scalable** bug detection!

$[p]\ C\ [\varepsilon: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

IL Proof Rules and Principles (Consequence)

$$\frac{p' \subseteq p \quad [p']\ C\ [\varepsilon: q'] \quad q' \supseteq q}{[p]\ C\ [\varepsilon: q]}\ (\text{Cons})$$

$$\frac{[p]\ C\ [\varepsilon: q_1 \vee q_2]}{[p]\ C\ [\varepsilon: q_1]}$$

- **Shrink** the **post** (e.g. drop disjuncts)
- **Scalable** bug detection!

$[p]\ C\ [\varepsilon: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

IL Proof Rules and Principles (Consequence)

$$\frac{p' \subseteq p \quad [p']\ C\ [\varepsilon: q'] \quad q' \supseteq q}{[p]\ C\ [\varepsilon: q]}\ \text{(Cons)} \qquad \frac{p' \supseteq p \quad \{p'\}\ C\ \{q'\} \quad q' \subseteq q}{\{p\}\ C\ \{q\}}\ \text{(HL-Cons)}$$

$$\frac{[p]\ C\ [\varepsilon: q_1 \vee q_2]}{[p]\ C\ [\varepsilon: q_1]}$$

- **Shrink** the **post** (e.g. drop disjuncts)
- **Scalable** bug detection!

$$[p]\ C\ [\varepsilon: q] \quad \textit{iff} \quad \forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$$

Incorrectness Logic: Summary

+ ***Under-approximate*** analogue of Hoare Logic

+ Formal foundation for ***bug catching***

– Global reasoning: ***non-compositional*** (as in original Hoare Logic)

– Cannot target ***memory safety bugs*** (e.g. use-after-free)

Incorrectness Logic: Summary

+ ***Under-approximate*** analogue of Hoare Logic

+ Formal foundation for ***bug catching***

– Global reasonin

– Cannot target *m*

<u>*Our Solution*</u>

Incorrectness Separation Logic

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

☛ ideal for heap-manipulating programs with ***aliasing***

```
[x]:= 1;
[y]:= 2;
[z]:= 3;
```

post: $\{x = 1 \wedge y = 2 \wedge z = 3\}$

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

☛ ideal for heap-manipulating programs with ***aliasing***

pre: $\{x \neq y \wedge x \neq z \wedge y \neq z\}$

```
[x]:= 1;
[y]:= 2;
[z]:= 3;
```

post: $\{x = 1 \wedge y = 2 \wedge z = 3\}$

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

☛ ideal for heap-manipulating programs with ***aliasing***

pre: $\{\boxed{x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \ldots}\}$

n!/2 conjuncts !

```
[x1]:= 1;
[x2]:= 2;
…
[xn]:= n;
```

post: $\{ x_1 = 1 \wedge \ldots \wedge x_n = n \}$

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

☛ ideal for heap-manipulating programs with ***aliasing***

pre: $\{ x \mapsto - * y \mapsto - * z \mapsto - \}$

```
[x]:= 1;
[y]:= 2;
[z]:= 3;
```

post: $\{ x \mapsto 1 * y \mapsto 2 * z \mapsto 3 \}$

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

☛ ideal for heap-manipulating programs with ***aliasing***

pre: $\{ \boxed{x \mapsto -} * y \mapsto - \; \boxed{*} \; z \mapsto - \}$

ownership
of heap cell at x

```
[x]:= 1;
[y]:= 2;
[z]:= 3;
```

'and ***separately***'

post: $\{ x \mapsto 1 * y \mapsto 2 * z \mapsto 3 \}$

$$\forall x, v, v'.\ x \mapsto v * x \mapsto v' \Rightarrow \text{false}$$

The Essence of Separation Logic (SL)

Frame Rule

$$\frac{\{p\}\ C\ \{q\}}{\{p * r\}\ C\ \{q * r\}}$$

$x \mapsto v \ * \ x \mapsto v' \Leftrightarrow \text{false}$ $\qquad$ $p \ * \ \text{emp} \Leftrightarrow p$

The Essence of Separation Logic (SL)

Frame Rule

$$\frac{\{p\}\ C\ \{q\}}{\{p * r\}\ C\ \{q * r\}}$$

$x \mapsto v \ * \ x \mapsto v' \Leftrightarrow \text{false}$ $\qquad$ $p \ * \ \text{emp} \Leftrightarrow p$

Local Axioms

WRITE $\{x \mapsto -\}\ [x] := v\ \{x \mapsto v\}$

READ $\{x \mapsto v\}\ y := [x]\ \{x \mapsto v \wedge y = v\}$

ALLOC $\{\text{emp}\}\ x := \text{alloc}()\ \{\exists l.\ l \mapsto - \wedge x = l\}$

FREE $\{x \mapsto -\}\ \text{free}(x)\ \{\ \text{emp}\ \}$

Incorrectness Separation Logic (ISL)

IL

$[p]\ C\ [\varepsilon: q]$

SL

$$\frac{\{p\}\ C\ \{q\}}{\{p * r\}\ C\ \{q * r\}}$$

$x \mapsto - \ * \ x \mapsto - \Leftrightarrow \text{false}$

$x \mapsto v \ * \ \text{emp} \Leftrightarrow x \mapsto v$

ISL

$$\frac{[p]\ C\ [\varepsilon: q]}{[p * r]\ C\ [\varepsilon: q * r]}$$

$x \mapsto v \ * \ x \mapsto v' \Leftrightarrow \text{false}$

$x \mapsto v \ * \ \text{emp} \Leftrightarrow x \mapsto v$

ISL: Local Axioms (First Attempt)

WRITE

$[x \mapsto v']$ $[x] := v$ $[ok: x \mapsto v]$

$[x = null]$ $[x] := v$ $[er: x = null]$

null-pointer dereference error

ISL: Local Axioms (First Attempt)

WRITE

$[x \mapsto v'] \; [x] := v \; [ok: x \mapsto v]$

$[x = null] \; [x] := v \; [er: x = null]$

null-pointer dereference error

READ

$[x \mapsto v] \; y := [x] \; [ok: x \mapsto v \land y = v]$

$[x = null] \; y := [x] \; [er: x = null]$

ALLOC

$[emp] \; x := alloc() \; [ok: \exists l.\ l \mapsto v \land x = l]$

ISL: Local Axioms (First Attempt)

WRITE

$[x \mapsto v'] \; [x]{:=}\, v \; [ok{:}\; x \mapsto v]$ $\qquad$ $[x{=}null] \; [x]{:=}\, v \; [er{:}\; x{=}null]$

null-pointer dereference error

READ

$[x \mapsto v] \; y{:=}\, [x] \; [ok{:}\; x \mapsto v \wedge y{=}v]$ $\qquad$ $[x{=}null] \; y{:=}\, [x] \; [er{:}\; x{=}null]$

ALLOC

$[emp] \; x{:=}\, alloc() \; [ok{:} \exists l.\; l \mapsto v \wedge x{=}l]$

FREE

$[x \mapsto v] \; free(x) \; [ok{:}\; emp]$ $\qquad$ $[x{=}null] \; free(x) \; [er{:}\; x{=}null]$

ISL: Local Axioms (First Attempt)

WRITE

$[x \mapsto v']\ [x] := v\ [ok: x \mapsto v]$

$[x = null]\ [x] := v\ [er: x = null]$

null-pointer dereference error

READ

$[x \mapsto v]\ y := [x]\ [ok: x \mapsto v \land y = v]$

$[x = null]\ y := [x]\ [er: x = null]$

ALLOC

$[emp]\ x := alloc()\ [ok: \exists l.\ l \mapsto v \land x = l]$

FREE

$[x \mapsto v]\ free(x)\ [ok: emp]$

ll] $free(x)\ [er: x = null]$

ISL: Local Axioms (First Attempt)

ISL $\dfrac{[p]\ C\ [\varepsilon{:}\ q]}{[p * r]\ C\ [\varepsilon{:}\ q * r]}$

$x \mapsto v * x \mapsto v' \Leftrightarrow \text{false}$

$\text{emp} * p \Leftrightarrow p$

$[x \mapsto v]\ \text{free}(x)\ [ok{:}\ \text{emp}]$

$[p]\ C\ [\varepsilon{:}\ q]$ *iff* $\forall s \in q.\ \exists s' \in p.\ (s', s) \in [C]\varepsilon$

ISL: Local Axioms (First Attempt)

ISL
$$\frac{[p]\ C\ [\varepsilon : q]}{[p * r]\ C\ [\varepsilon : q * r]}$$

$x \mapsto v \ * \ x \mapsto v' \Leftrightarrow \text{false}$

$\text{emp} \ * \ p \Leftrightarrow p$

$$\frac{[x \mapsto v]\ \text{free}(x)\ [\text{ok}: \text{emp}]}{[x \mapsto v * x \mapsto v]\ \text{free}(x)\ [\text{ok}: \text{emp} * x \mapsto v]}\ \text{(Frame)}$$

$[p]\ C\ [\varepsilon : q]$ *iff* $\forall s \in q.\ \exists s' \in p.\ (s',s) \in [C]\varepsilon$

ISL: Local Axioms (First Attempt)

ISL

$$\frac{[p]\ C\ [\varepsilon: q]}{[p * r]\ C\ [\varepsilon: q * r]}$$

$$x \mapsto v \ * \ x \mapsto v' \Leftrightarrow \text{false}$$
$$\text{emp} \ * \ p \Leftrightarrow p$$

$$\dfrac{\dfrac{[x \mapsto v]\ \text{free}(x)\ [ok: \text{emp}]}{[x \mapsto v * x \mapsto v]\ \text{free}(x)\ [ok: \text{emp} * x \mapsto v]}\ \text{(Frame)}}{[\text{false}]\ \text{free}(x)\ [ok: x \mapsto v]}\ \text{(Cons)}$$

$[p]\ C\ [\varepsilon: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

ISL: Local Axioms (First Attempt)

ISL

$$\frac{[p]\ C\ [\varepsilon: q]}{[p * r]\ C\ [\varepsilon: q * r]}$$

$x \mapsto v \ * \ x \mapsto v' \Leftrightarrow \text{false}$

$\text{emp} \ * \ p \Leftrightarrow p$

$$\dfrac{\dfrac{[x \mapsto v]\ \text{free(x)}\ [\text{ok: emp}]}{[x \mapsto v * x \mapsto v]\ \text{free(x)}\ [\text{ok: emp} * x \mapsto v]}\ \text{(Frame)}}{[\text{false}]\ \text{free(x)}\ [\text{ok: } x \mapsto v]}\ \text{(Cons)}$$

$[p]\ C\ [\varepsilon: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

$[\text{false}]\ C\ [\varepsilon: q]$ ✗ (unless $q \Rightarrow$ false)

ISL: Local Axioms (First Attempt)

ISL

$$\frac{[p]\ C\ [\varepsilon: q]}{[p * r]\ C\ [\varepsilon: q * r]}$$

$x \mapsto v \ * \ x \mapsto v' \Leftrightarrow \text{false}$

$\text{emp} \ * \ p \Leftrightarrow p$

Solution:

Track Deallocated Locations!

$[x \mapsto v$ … $\mapsto v]$ (Frame)

(Cons)

$[p]\ C\ [\varepsilon: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

$[\text{false}]\ C\ [\varepsilon: q]$ ✗ (unless $q \Rightarrow \text{false}$)

Solution: Track Deallocated Locations!

$[x \mapsto v]$ free(x) $[ok: \boxed{x \not\mapsto}]$

x is ***deallocated***

$x \mapsto v \ * \ x \mapsto v' \Leftrightarrow \text{false}$

$p \ * \ \text{emp} \Leftrightarrow p$

$x \mapsto v \ * \ x \not\mapsto \ \Leftrightarrow \text{false}$

$x \not\mapsto \ * \ x \not\mapsto \ \Leftrightarrow \text{false}$

Solution: Track Deallocated Locations!

$[x \mapsto v]$ free(x) $[ok: x \not\mapsto\ ]$

Solution: Track Deallocated Locations!

$$\frac{[x \mapsto v]\ \text{free}(x)\ [ok\!:\ x \not\mapsto\ ]}{[x \mapsto v \ast x \mapsto v]\ \text{free}(x)\ [ok\!:\ x \not\mapsto \ast\, x \mapsto v]}$$

Solution: Track Deallocated Locations!

$$\begin{array}{c}
[x \mapsto v]\ \mathrm{free}(x)\ [ok\!: x \not\mapsto\ ] \\
\hline
[x \mapsto v * x \mapsto v]\ \mathrm{free}(x)\ [ok\!: x \not\mapsto * x \mapsto v] \\
\hline
[\mathit{false}]\ \mathrm{free}(x)\ [ok\!: \mathit{false}]
\end{array} \quad ✔$$

$$\boxed{[p]\ C\ [\varepsilon\!: q] \quad \textit{iff} \quad \forall s \in q.\ \exists s' \in p.\ (s', s) \in [C]\varepsilon}$$

$$\boxed{[p]\ C\ [\varepsilon\!: \mathit{false}]\ ✔\ \text{(vacuous)}}$$

ISL: Local Axioms

$[x \mapsto v]\ \text{free}(x)\ [ok: x \not\mapsto]$ $[x{=}null]\ \text{free}(x)\ [er: x{=}null]$

FREE

$\boxed{[x \not\mapsto]\ \text{free}(x)\ [er: x \not\mapsto]}$

double-free error

ISL: Local Axioms

$[x \mapsto v]$ free(x) $[ok: x \not\mapsto]$

$[x{=}null]$ free(x) $[er: x{=}null]$

$\boxed{[x \not\mapsto]\ \text{free(x)}\ [er: x \not\mapsto]}$

FREE

double-free error

$[x \mapsto v']$ [x]:= v $[ok: x \mapsto v]$

$[x{=}null]$ [x]:= v $[er: x{=}null]$

$[x \not\mapsto]$ [x]:= v $[er: x \not\mapsto]$

WRITE

$[x \mapsto v]$ y:= [x] $[ok: x \mapsto v \wedge y{=}v]$

$[x{=}null]$ y:= [x] $[er: x{=}null]$

$[x \not\mapsto]$ y:= [x] $[er: x \not\mapsto]$

READ

$[emp]$ x:= alloc() $[ok: \exists l.\ l \mapsto v \wedge x{=}l]$

ALLOC

ISL: Local Axioms

$[x \mapsto v]$ free(x) $[ok: x \not\mapsto]$

$[x{=}null]$ free(x) $[er: x{=}null]$

$[x \not\mapsto]$ free(x) $[er: x \not\mapsto]$

FREE

double-free error

$[x \mapsto v']$ [x]:= v $[ok: x \mapsto v]$

$[x{=}null]$ [x]:= v $[er: x{=}null]$

$[x \not\mapsto]$ [x]:= v $[er: x \not\mapsto]$

WRITE

$[x \mapsto v]$ y:= [x] $[ok: x \mapsto v \wedge y{=}v]$

$[x{=}null]$ y:= [x] $[er: x{=}null]$

$[x \not\mapsto]$ y:= [x] $[er: x \not\mapsto]$

READ

$[emp]$ x:= alloc() $[ok: \exists l.\ l \mapsto v \wedge x{=}l]$

$[y \not\mapsto]$ x:= alloc() $[ok: y \mapsto v \wedge x{=}y]$

ALLOC

ISL Summary

- Incorrectness ***Separation*** Logic (ISL)
  - ➡ IL + SL for ***compositional bug catching***
  - ➡ ***Under-approximate*** analogue of SL
  - ➡ Targets ***memory safety bugs*** (e.g. use-after-free)
- Combining IL+SL: not straightforward
  - ➡ ***invalid frame*** rule!
- Fix: a ***monotonic model*** for frame preservation
- Recovering the ***footprint property*** for completeness
- ISL-based ***analysis***
  - ➡ ***No-false-positives theorem:***

    All bugs found are true bugs

Part II.

Pulse-X: ISL for Scalable Bug Detection

Pulse-X at a Glance

- **Automated** program analysis for **memory safety errors** (NPEs, UAFs) and **leaks**
- Underpinned by ISL (under-approximate) — **no false positives***
- **Inter-procedural** and **bi-abductive** — under-approximate analogue of Infer
- **Compositional** (begin-anywhere analysis) — important for CI
- Deployed at Meta
- **Performance**: comparable to Infer, though merely an academic tool!
- **Fix rate**: comparable or better than Infer!
- Three dimensional scalability
  - code size (large codebases)
  - people (large teams, CI)
  - speed (high frequency of code changes)

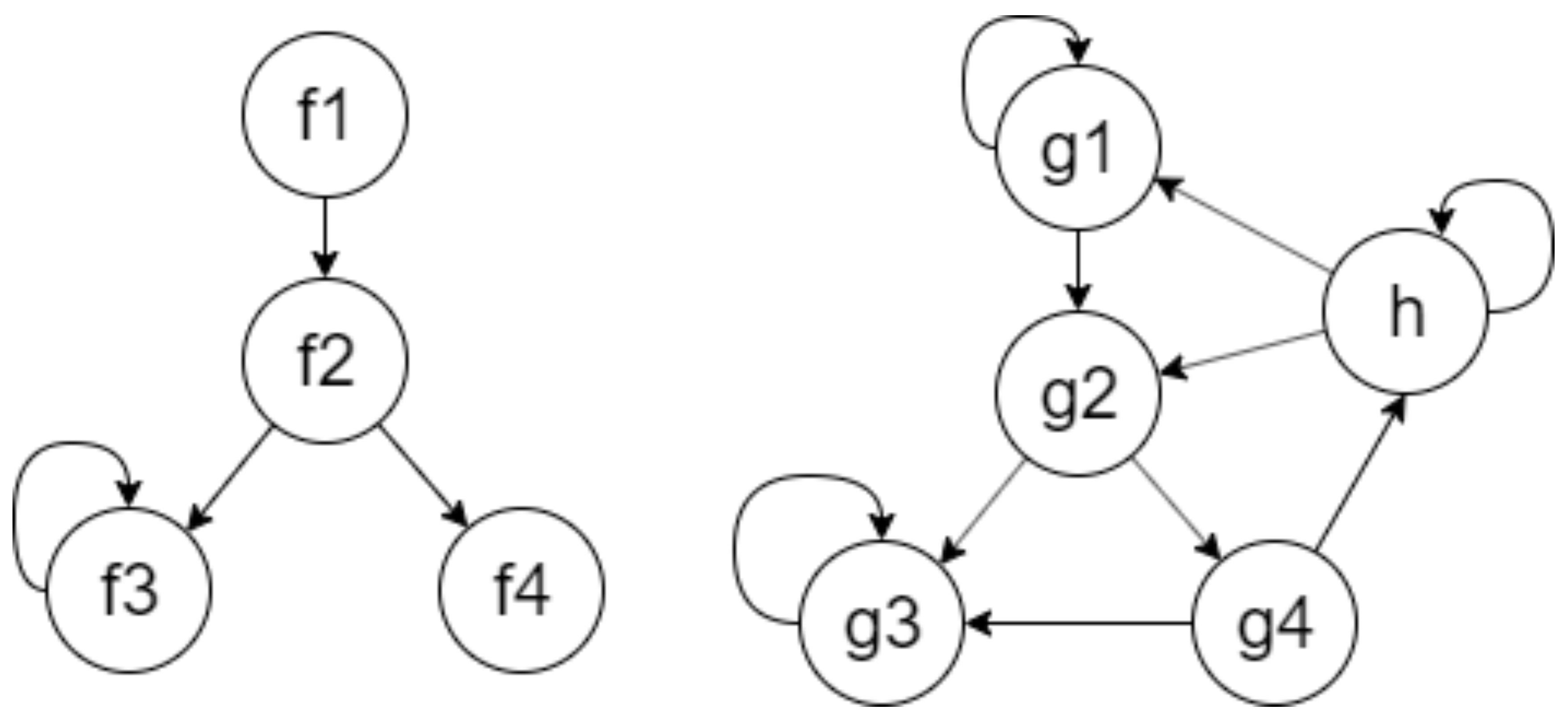
Compositional, Begin-Anywhere Analysis

- Analysis result of a program = analysis results of its parts

  +

  a method of combining them

  - ➡ Parts: Procedures



  - ➡ Method: under-approximate bi-abduction
  - ➡ Analysis result: incorrectness triples (under-approximate specs)

Pulse-X Algorithm: Proof Search in ISL

- Analyse each procedure f in isolation, find its **summary** (collection of ISL triples)
  - A **summary table** T, initially populated only with local (pre-defined) axioms
  - Use bi-abduction and T to find the summary of f
  - Recursion: bounded unrolling
  - Extend T with the summary of f
- Similar bi-abductive mechanism to Infer, but:
  - Can **soundly** drop execution paths/branches
  - Can **soundly** bound loop unrolling

Pulse-X: Null Pointer Dereference in OpenSSL

```
int ssl_excert_prepend(...){
    SSL_EXCERT *exc= app_malloc(sizeof(*exc), "prepend cert”);
    memset(exc, 0, sizeof(*exc));
    …
}
```

Pulse-X: Null Pointer Dereference in OpenSSL

```
int ssl_excert_prepend(...){
    SSL_EXCERT *exc= app_malloc(sizeof(*exc), "prepend cert”);
    memset(exc, 0, sizeof(*exc));
    …
}
```

calls CRYPTO_malloc (a malloc wrapper)

Pulse-X: Null Pointer Dereference in OpenSSL

```
1.int ssl_excert_prepend(...){
2.    SSL_EXCERT *exc= app_malloc(sizeof(*exc), "prepend cert”);
3.    memset(exc, 0, sizeof(*exc));
      …
}
```

calls CRYPTO_malloc (a malloc wrapper)

null pointer dereference

CRYPTO_malloc may return null!

Pulse-X: Null Pointer Dereference in OpenSSL

```c
int ssl_excert_prepend(...){
    SSL_EXCERT *exc= app_malloc(sizeof(*exc), "prepend cert");
    memset(exc, 0, sizeof(*exc));

    …
}
```

calls CRYPTO_malloc (a malloc wrapper)

null pointer dereference

CRYPTO_malloc may return null!

[emp] `*exc= app_malloc(sz, …)` [ok: exc = null]

+

[exc = null] `memset(exc,-,-)` [er: exc = null]

[emp] `ssl_excert_prepend(…)` [er: exc = null]

Pulse-X: Null Pointer Dereference in OpenSSL

apps/lib/s_cb.c Outdated

Hide resolved

```
...    ...    @@ -956,6 +956,9 @@ static int ssl_excert_prepend(SSL_EXCERT **pexc)
956    956      {
957    957          SSL_EXCERT *exc = app_malloc(sizeof(*exc), "prepend cert");
958    958
       959    +      if (!exc) {
```

paulidale 13 days ago Contributor

False positive, `app_malloc()` doesn't return if the allocation fails.

lequangloc 13 days ago Author

Our tool recognizes app_malloc() in test/testutil/apps_mem.c rather than the one in apps/lib/apps.c. While the former doesn't return if the allocation fails, the latter does. How do we know which one is actually called?

paulidale 13 days ago Contributor

It would need to look at the link lines or build dependencies to figure out which sources were used.

We should fix the one in `test/testutil/apps_mem.c`.

Pulse-X: Null Pointer Dereference in OpenSSL

apps/lib/s_cb.c Outdated Hide resolved

```
@@ -956,6 +956,9 @@ static int ssl_excert_prepend(SSL_EXCERT **pexc)
956 956  {
957 957      SSL_EXCERT *exc = app_malloc(sizeof(*exc), "prepend cert");
958 958
    959 +       if (!exc) {
```

paulidale 13 days ago Contributor

False positive, `app_malloc()` doesn't return if the allocation fails.

lequangloc 13 days ago Author

Our tool recognizes app_malloc() in test/testutil/apps_mem.c rather than the one in apps/lib/apps.c. While the former doesn't return if the allocation fails, the latter does. How do we know which one is actually called?

paulidale 13 days ago Contributor

It would need to look at the link lines or build dependencies to figure out which sources were used.

We should fix the one in `test/testutil/apps_mem.c`.

Created pull request #15836 to commit the fix.

Pulse-X: Bug Reporting

No False Positives: Report ***All*** Bugs Found?

Not quite…

Pulse-X: Bug Reporting

```
1.void foo(int *x){
2.    *x = 42;
}
```

WRITE [x=null] *x = v [er: x=null]

[x=null] foo(x) [er: x=null]

Pulse-X: Bug Reporting

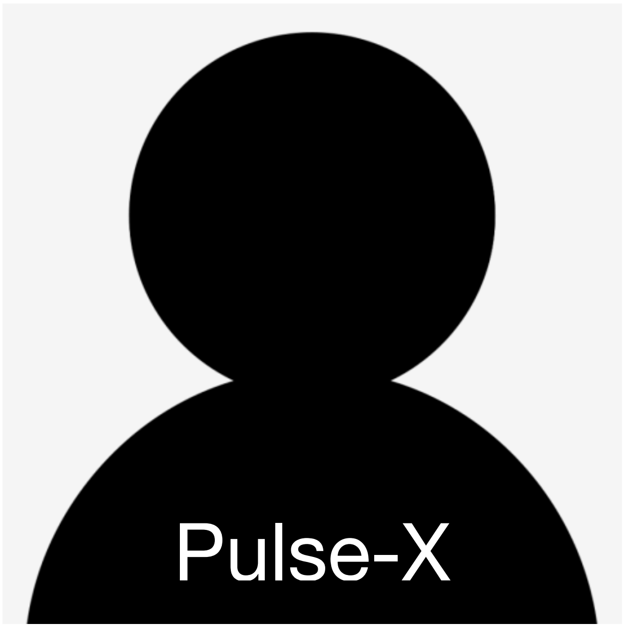
```
void foo(int *x){
    *x = 42;
}
```

WRITE [x=null] *x = v [er: x=null]

[x=null] foo(x) [er: x=null]

Should we report this NPD?

yes **no**



"But I never call foo with null!"

"Which bugs shall I report then?"

Pulse-X: Bug Reporting

```
1.void foo(int *x){
2.    *x = 42;
}
```

WRITE [x=null] *x = v [er: x=null]

x=null]

Problem

Must consider the **whole program** to decide whether to report

Solution

Manifest Errors

Developer

Pulse-X

"But I never call foo with null!"

"Which bugs shall I report then?"

Pulse-X: ***Manifest*** Errors

- Intuitively: the error occurs for **all input states**
- Formally: [p] C [er: q] is manifest iff:

$$\forall\, s.\ \exists\, s'.\ \ (s,s') \in [C]_{er}\ \ \wedge\ \ s' \in (q * \text{true})$$

- Algorithmically: [p] C [er: q] is manifest if when: $q = \exists \vec{X}.\ h_q \wedge \boldsymbol{\pi}_q$:
  - ➡ $p = \text{emp} \wedge \text{true}$
  - ➡ $\text{sat}(q)$ and $\text{locs}(q) \subseteq \vec{X}$
  - ➡ for all $\vec{v}$: $\text{sat}\ (\,\vec{v}/\ \text{flv}(q) \cup \vec{X}\ )$

Pulse-X: ***Manifest*** Errors

- Intuitively: the error occurs for **all input states**
- Formally: [p] C [er: q] is manifest iff:

Theorem 3.5 (Manifest errors). *An error triple* $\models [p]\ \mathrm{C}\ [er: q]$ *with* $q \triangleq \exists \overrightarrow{X_q}.\ \kappa_q \land \pi_q$ *denotes a* manifest error *if:*

(1) $p \equiv \mathsf{emp} \land \mathsf{true}$*;*

(2) $\mathsf{sat}(q)$ *holds;*

(3) $\mathsf{locs}(\kappa_q) \subseteq \overrightarrow{X_q}$*, where* $\mathsf{locs}(.)$ *is as defined below; and*

(4) for all $\overrightarrow{v}$, $\mathsf{sat}(\pi_q[\overrightarrow{v}/\overrightarrow{Y} \cup \mathsf{locs}(\kappa_q)])$ *holds, where* $\overrightarrow{Y} = \mathsf{flv}(q)$*.*

$\mathsf{locs}(\mathsf{emp}) \triangleq \emptyset \quad \mathsf{locs}(x \mapsto X) \triangleq \{x\} \quad \mathsf{locs}(X \mapsto V) = \mathsf{locs}(X \not\mapsto) \triangleq \{X\} \quad \mathsf{locs}(\kappa_1 * \kappa_2) \triangleq \mathsf{locs}(\kappa_1) \cup \mathsf{locs}(\kappa_2)$

Pulse-X: Null Pointer Dereference in OpenSSL

```
1.int ssl_excert_prepend(...){
2.    SSL_EXCERT *exc= app_malloc(sizeof(*exc), "prepend cert”);
3.    memset(exc, 0, sizeof(*exc));
      …
}
```

calls CRYPTO_malloc (a malloc wrapper)

null pointer dereference

CRYPTO_malloc may return null!

[emp] `ssl_excert_prepend(…)` [er: exc = null]

Manifest Error (all calls to `ssl_excert_prepend` can trigger the error)!

Pulse-X: ***Latent*** Errors

An error triple [p] C [er: q] is <u>latent</u> iff it is not manifest

Pulse-X: Latent Error

```
int chopup_args(ARGS *args,…){
    …
    if (args->count == 0 ) {
      args->count=20;
      args->data= (char**)ssl_excert_prepend(…);
    }
    for (i=0; i<args->count; i++) {
      args->data[i]=NULL;
      …
    }
```

Pulse-X: Latent Error

```
int chopup_args(ARGS *args,…){
      …
      if (args->count == 0 ) {
         args->count=20;
         args->data= (char**)ssl_excert_prepend(…);
      }
      for (i=0; i<args->count; i++) {
         args->data[i]=NULL;
         …
      }
```

null pointer
dereference

Pulse-X: Latent Error

```
int chopup_args(ARGS *args,…){
    …
    if (args->count == 0 ) {
      args->count=20;
      args->data= (char**)ssl_excert_prepend(…);
    }
    for (i=0; i<args->count; i++) {
      args->data[i]=NULL;
      …
    }
```

null pointer dereference

Latent Error:

only calls with `args->count==0` can trigger the error

Pulse-X: Memory Leak in OpenSSL

```
static int www_body(…){
  ...
  io = BIO_new(BIO_f_buffer());
  ssl_bio BIO_new(BIO_f_ssl());
  ...
  BIO_push(io, ssl_bio);
  ...
  BIO_free_all(io);
  ...
  return ret;
}
```

Pulse-X: Memory Leak in OpenSSL

```
static int www_body(…){
  ...
  io = BIO_new(BIO_f_buffer());
  ssl_bio BIO_new(BIO_f_ssl());
  ...
  BIO_push(io, ssl_bio);
  ...
  BIO_free_all(io);
  ...
  return ret;
}
```

does nothing when `io` is null

Pulse-X: Memory Leak in OpenSSL

```
static int www_body(…){
  ...
  io = BIO_new(BIO_f_buffer());
  ssl_bio BIO_new(BIO_f_ssl());
  ...
  BIO_push(io, ssl_bio);
  ...
  BIO_free_all(io);
  ...
  return ret;
}
```

does nothing when `io` is null

leaks `ssl_bio`

Pulse-X: Memory Leak in OpenSSL

```
static int www_body(…){
  ...
  io = BIO_new(BIO_f_buffer());
  ssl_bio BIO_new(BIO_f_ssl());
  ...
  BIO_push(io, ssl_bio);
  ...
  BIO_free_all(io);
  ...
  return ret;
}
```

does nothing when `io` is null

leaks `ssl_bio`

426 lines of complex code:
`io` manipulated by several procedures and multiple loops

Pulse-X performs under-approximation with bounded loop unrolling

No-False Positives: Caveat

- Unknown procedures (e.g. where the code is unavailable) are treated as `skip`
- Incomplete arithmetic solver

Speed (fast but simplistic)	vs	Precision (slow but accurate)

No-False Positives: Caveat

- Unknown procedures (e.g. where the code is unavailable) are treated as `skip`
- Incomplete arithmetic solver

Speed (fast but simplistic) vs Precision (slow but accurate)

"Scientists seek perfection and are idealists. ... An engineer's task is to not be idealistic. You need to be realistic as you have to compromise between conflicting interests."

Conclusions

- Incorrectness Separation Logic (ISL)
  - ➡ Combining IL and SL for ***compositional bug catching*** (in sequential programs)
  - ➡ ***no-false-positives*** theorem

- Pulse-X
  - ➡ Automated program analysis for detecting memory safety errors and leaks
  - ➡ Manifest errors (underpinned by ISL): no false positives*
  - ➡ compositional, scalable, begin-anywhere

Thank You for Listening!

azalea@imperial.ac.uk SoundAndComplete.org @azalearaad

Incorrectness Logic & Under-Approximation: Foundations of Bug Detection

Azalea Raad
Imperial College London

Principles of Programming Languages
16 January 2023

azalea@imperial.ac.uk SoundAndComplete.org @azalearaad

Part III.

ISL Extension:

Concurrent Incorrectness Separation Logic (CISL)

&

Concurrent Adversarial Separation Logic (CASL)

&

Incorrectness Non-Termination Logic (INTL)

Extension 1: Concurrent Incorrectness Separation Logic (CISL)

ISL

$$\frac{[p]\ C\ [\varepsilon: q]}{[p * r]\ C\ [\varepsilon: q * r]}$$

CSL

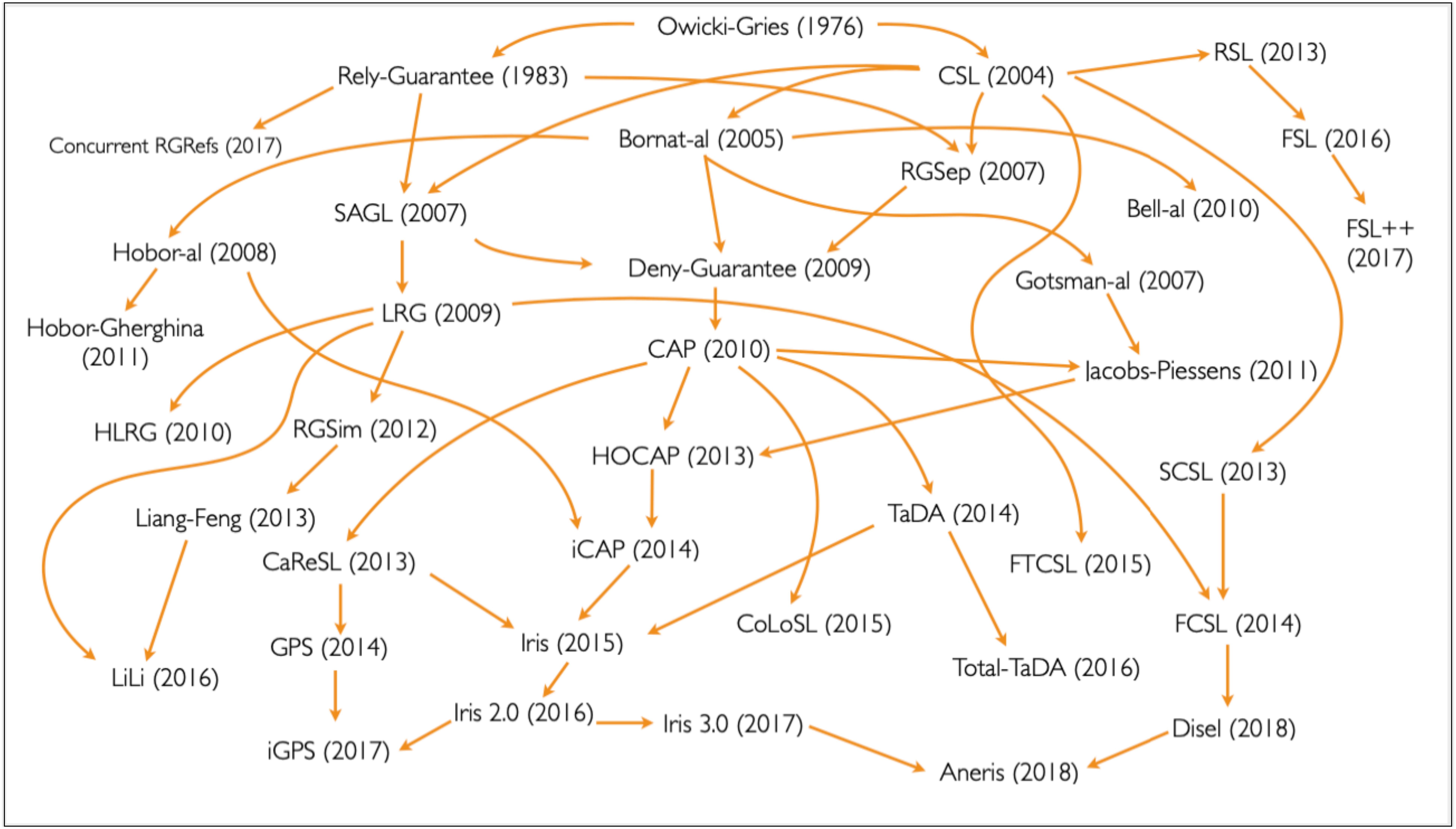
$$\frac{\{p_1\}\ C_1\ \{q_1\} \qquad \{p_2\}\ C_2\ \{q_2\}}{\{p_1 * p_2\}\ C_1 \,||\, C_2\ \{q_1 * q_2\}}$$

CISL

$$\frac{[p_1]\ C_1\ [\varepsilon: q_1] \qquad [p_2]\ C_2\ [\varepsilon: q_2]}{[p_1 * p_2]\ C_1 \,||\, C_2\ [\varepsilon: q_1 * q_2]}$$

Which CISL?

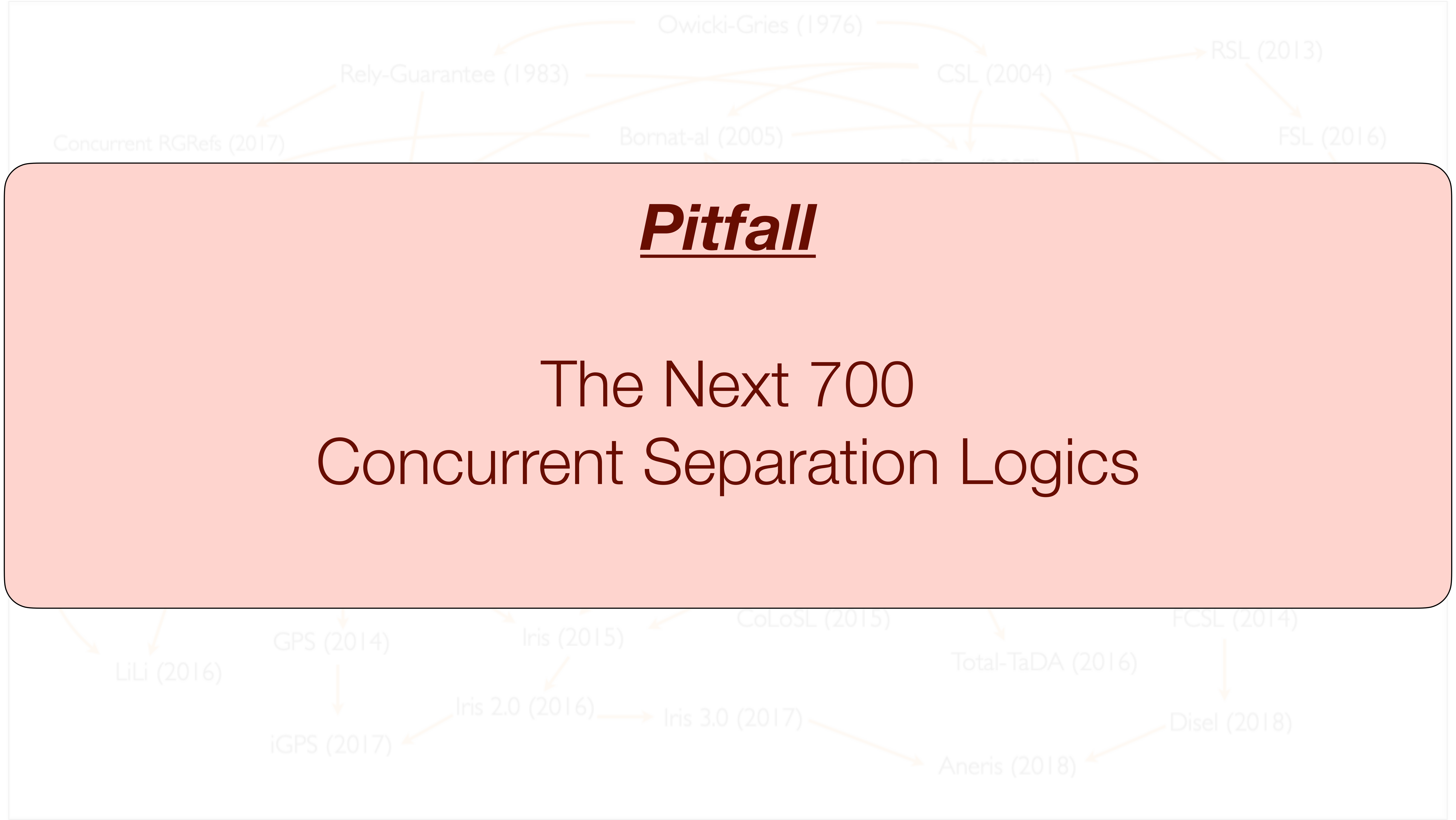
CSL (Correctness) Family Tree…



Graph courtesy of Ilya Sergey

Which CISL?

CSL (Correctness) Family Tree…



Pitfall

The Next 700
Concurrent Separation Logics

Graph courtesy of Ilya Sergey

Which CISL?

CSL (Correctness) Family Tree…

Pitfall

The Next 700

Concurrent ***Incorrectness*** Separation Logics

Graph courtesy of Ilya Sergey

Which CISL?

CSL (Correctness) Family Tree…

Solution

CISL: ***general, parametric*** framework
that can be ***instantiated***
for different use cases
à la Views [Dinsdale-Young et al., 2013]

Graph courtesy of Ilya Sergey

CISL Framework

- ***First*** unifying framework for ***concurrent under-approximate*** reasoning
- ***General*** framework for multiple bug catching analyses
  - ➡ Memory safety errors (e.g. null-pointer exception, use-after-free errors): CISL_{SV}
  - ➡ Races: CISL_{RD}
  - ➡ Deadlocks: CISL_{DD}
- Sound: ***no false positives*** (NFP) guaranteed
- Underpins ***scalable*** bug-catching tools (NFP for free)
  - ➡ CISL_{RD}: analogous to ***RacerD*** @Meta
  - ➡ CISL_{DD}: analogous to ***DLTool*** @Meta
- <u>Caveat</u>: cannot detect bugs where there are <u>control flow dependencies</u> between threads

Three Faces of Concurrency Bugs:
1. **Local** Bugs

What are they?

➡ They are due to one thread

$$\begin{array}{l} \texttt{free}(x); \\ \text{L: } [x] := 1 \end{array} \;\Big\|\; \mathrm{C}$$

local use-after-free (memory safety) bug at L

Three Faces of Concurrency Bugs: 1. **Local** Bugs

What are they?

➡ They are <u>due to one thread</u>

$$\begin{array}{l} \texttt{free}(x); \\ \text{L}: [x] := 1 \end{array} \;\Bigg\|\; \text{C}$$

local use-after-free (memory safety) bug at L

Thread-local analysis tools?

➡ <u>Existing</u> (sequential) tools out of the box
e.g. PulseX @Meta (based on ISL)

CISL

$$\frac{[p]\ C_1\ [er: q]}{[p]\ C_1 \;||\; C_2\ [er: q]}\ \text{ParEr}$$

Short-circuiting on errors

Three Faces of Concurrency Bugs:
2, 3. **Global** Bugs

Bug is <u>due to two or more threads</u>, under <u>certain interleavings</u>

2. ***data-agnostic***: threads do not affect one another's control flow

$$\mathrm{L}\colon \mathtt{free}(x) \;\|\; \mathrm{L}'\colon \mathtt{free}(x)$$

(global) data-agnostic use-after-free bug at L (L')

$$\begin{array}{l} \mathtt{free}(x); \\ [z] := 1; \end{array} \;\Bigg\|\; \begin{array}{l} a := [z]; \\ \mathtt{if}\,(*)\; \mathrm{L}\colon [x] := 1 \end{array}$$

(global) data-agnostic use-after-free bug at L

Three Faces of Concurrency Bugs:
2, 3. **Global** Bugs

Bug is due to two or more threads, under certain interleavings

2. ***data-agnostic***: threads do not affect one another's control flow

$$\text{L}: \mathtt{free}(x) \;\|\; \text{L}': \mathtt{free}(x)$$

(global) data-agnostic use-after-free bug at L (L')

$$\begin{array}{l} \mathtt{free}(x); \\ [z] := 1; \end{array} \;\Bigg\|\; \begin{array}{l} a := [z]; \\ \mathtt{if}\,(*)\ \text{L}: [x] := 1 \end{array}$$

(global) data-agnostic use-after-free bug at L

3. ***data-dependent*** bugs: threads do affect one another's control flow

$$\begin{array}{l} \mathtt{free}(x); \\ [z] := 1; \end{array} \;\Bigg\|\; \begin{array}{l} a := [z]; \\ \mathtt{if}\,(a{=}1)\ \text{L}: [x] := 1 \end{array}$$

(global) data-dependent use-after-free bug at L

Three Faces of Concurrency Bugs:
2, 3. **Global** Bugs

Thread-local analysis tools?

2. ***data-agnostic***: threads do not affect one another's control flow

- ➡ encode errors as ok (no short-circuiting)
- ➡ assumed by existing tools: RacerD, DLTool @Meta

CISL

$$\frac{[p_1]\ C_1\ [ok{:}q_1] \qquad [p_2]\ C_2\ [ok{:}q_2]}{[p_1 * p_2]\ C_1 \,||\, C_2\ [ok{:}q_1 * q_2]}\ \text{Par}$$

3. ***data-dependent*** bugs: threads do affect one another's control flow

- ➡ not handled compositionally in CISL theory

Three Faces of Concurrency Bugs:
2, 3. **Global** Bugs

Thread-local analysis tools?

2. ***data-agnostic***: threads do not affect one another's control flow

- ➡ encode errors as ok (no short-circuiting)
- ➡ assumed by existing tools: RacerD, DLTool @Meta

CISL

$$\frac{[p_1]\ C_1\ [ok{:}q_1] \qquad [p_2]\ C_2\ [ok{:}q_2]}{[p_1 * p_2]\ C_1 \parallel C_2\ [ok{:}q_1 * q_2]}\ \text{Par}$$

CISL

3. ***data-dependent*** bugs: threads do affect one another's control flow

- ➡ not handled compositionally in CISL theory

CISL$_{RD}$: Data-Agnostic Races

Two memory accesses (reads/writes), a and b, in program C ***race*** iff

1. a and b are ***conflicting***:
   - ➡ they are by distinct threads
   - ➡ on the same location
   - ➡ at least one of them is a write
2. they appear ***next to each other in an interleaving*** (history) of C

CISL$_{RD}$: Data-Agnostic Races

Two memory accesses (reads/writes), a and b, in program C ***race*** iff

1. a and b are ***conflicting***:
   - ➡ they are by distinct threads
   - ➡ on the same location
   - ➡ at least one of them is a write
2. they appear ***next to each other in an interleaving*** (history) of C

1. lock l;	4. lock l;
2. unlock l;	5. $[x] := 2$;
3. $[x] := 1$;	6. unlock l;

Race between lines 3, 5witnessed by:
H = [1, 2, 4, **3**, **5**, 6]

CISL$_{RD}$: Data-Agnostic Races

Two memory accesses (reads/writes), a and b, in program C ***race*** iff

1. a and b are ***conflicting***:
   - ➡ they are by distinct threads
   - ➡ on the same location
   - ➡ at least one of them is a write
2. they appear ***next to each other in an interleaving*** (history) of C

1. $\texttt{lock}\ l$;	4. $\texttt{lock}\ l$;
2. $\texttt{unlock}\ l$;	5. $[x] := 2$;
3. $[x] := 1$;	6. $\texttt{unlock}\ l$;

Race between lines 3, 5witnessed by:
H = [1, 2, 4, **3**, **5**, 6]

1. $\texttt{lock}\ l$;	4. $\texttt{lock}\ l$;
2. $[x] := 1$;	5. $[x] := 2$;
3. $\texttt{unlock}\ l$;	6. $\texttt{unlock}\ l$;

No races

CISL$_{RD}$

$[\ \tau_1 \mapsto [\,] * \tau_2 \mapsto [\,]\ ]$

$[\ \tau_1 \mapsto [\,]\ ]$	$\Vert$	$[\ \tau_2 \mapsto [\,]\ ]$
1. lock l;		4. lock l;
2. unlock l;		5. $[x]$:= 2;
3. $[x]$:= 1;		6. unlock l;

Methodology:

➡ construct sequential histories

➡ analyse them for races

CISL

$$\frac{[p_1]\ C_1\ [ok{:}q_1] \qquad [p_2]\ C_2\ [ok{:}q_2]}{[p_1 * p_2]\ C_1 \mid\mid C_2\ [ok{:}q_1 * q_2]}\ \text{Par}$$

CISL$_{RD}$

$[\ \tau_1 \mapsto [] * \tau_2 \mapsto []\ ]$

$[\ \tau_1 \mapsto []\ ]$

1. lock l;
2. unlock l;
3. $[x]$:= 1;

$||$

$[\ \tau_2 \mapsto []\ ]$

4. lock l;
5. $[x]$:= 2;
6. unlock l;

Methodology:

➡ construct sequential histories

➡ analyse them for races

CISL

$$\frac{[p_1]\ C_1\ [ok{:}q_1] \qquad [p_2]\ C_2\ [ok{:}q_2]}{[p_1 * p_2]\ C_1 \parallel C_2\ [ok{:}q_1 * q_2]}\ \text{Par}$$

CISL$_{RD}$

$[\ \tau_1 \mapsto [] * \tau_2 \mapsto []\ ]$

$[\ \tau_1 \mapsto []\ ]$

1. lock l;

$[ok\!: \tau_1 \mapsto [L(\tau_1, l)]\ ]$

2. unlock l;

3. $[x]$:= 1;

||

$[\ \tau_2 \mapsto []\ ]$

4. lock l;

5. $[x]$:= 2;

6. unlock l;

Methodology:

➡ construct sequential histories

➡ analyse them for races

CISL

$$\frac{[p_1]\ C_1\ [ok\!:q_1] \qquad [p_2]\ C_2\ [ok\!:q_2]}{[p_1 * p_2]\ C_1 \,||\, C_2\ [ok\!:q_1 * q_2]}\ \text{Par}$$

CISL$_{RD}$: Lock Axiom

CISL$_{RD}$

$$\frac{H' = H \mathbin{++} [L(\tau, l)] \qquad H' \text{ is well-formed}}{[\tau \mapsto H]\ \mathrm{lock}_\tau\ l\ [ok: \tau \mapsto H']}\ \text{RD-Lock}$$

H is well-formed iff it respects the lock semantics:

➡ lock l is acquired only if it is not already held

➡ lock l is released by τ only if it is already held by τ

CISL$_{RD}$

$[\ \tau_1 \mapsto [] * \tau_2 \mapsto []\ ]$

$[\ \tau_1 \mapsto []\ ]$	$[\ \tau_2 \mapsto []\ ]$
1. lock l;	4. lock l;
$[ok\!:\ \tau_1 \mapsto [L(\tau_1, l)]\ ]$	
2. unlock l;	5. $[x]$:= 2;
$[ok\!:\ \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l)]\ ]$	
3. $[x]$:= 1;	6. unlock l;

Methodology:

➡ construct sequential histories

➡ analyse them for races

CISL

$$\frac{[p_1]\ C_1\ [ok\!:q_1] \qquad [p_2]\ C_2\ [ok\!:q_2]}{[p_1 * p_2]\ C_1 \parallel C_2\ [ok\!:q_1 * q_2]}\ \text{Par}$$

CISL$_{RD}$: Unlock Axiom

CISL$_{RD}$

$$\frac{H' = H +\!\!+ [U(\tau, l)] \qquad H' \text{ is well-formed}}{[\tau \mapsto H] \text{ unlock}_\tau\ l\ [ok\!: \tau \mapsto H']} \text{ RD-Unlock}$$

A history H is well-formed iff it respects the lock semantics:

➡ lock l is acquired only if it is not already held

➡ lock l is released by τ only if it is already held by τ

CISL$_{RD}$

$$[\ \tau_1 \mapsto [\,] * \tau_2 \mapsto [\,]\ ]$$

$[\ \tau_1 \mapsto [\,]\ ]$	$\Vert$	$[\ \tau_2 \mapsto [\,]\ ]$
1. lock l;		4. lock l;
$[ok: \tau_1 \mapsto [L(\tau_1, l)]\ ]$		
2. unlock l;		5. $[x]$:= 2;
$[ok: \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l)]\ ]$		
3. $[x]$:= 1;		6. unlock l;
$[ok: \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)]\ ]$		

Methodology:

- ➡ construct sequential histories
- ➡ analyse them for races

CISL

$$\frac{[p_1]\ C_1\ [ok{:}q_1] \qquad [p_2]\ C_2\ [ok{:}q_2]}{[p_1 * p_2]\ C_1 \,||\, C_2\ [ok{:}q_1 * q_2]}\ \text{Par}$$

CISL$_{RD}$: Memory Access Axioms

CISL$_{RD}$

$$\frac{H' = H{+}{+}[\boxed{R(\tau, L, x)}]}{[\tau \mapsto H]\ L\colon a :=_\tau [x]\ [ok\colon \tau \mapsto H']}\ \text{RD-Read}$$

$$\frac{H' = H{+}{+}[\boxed{W(\tau, L, x)}]}{[\tau \mapsto H]\ L\colon [x] :=_\tau a\ [ok\colon \tau \mapsto H']}\ \text{RD-Write}$$

We ***do not record the values*** read/written

CISL$_{RD}$

$[\ \tau_1 \mapsto [] * \tau_2 \mapsto []\ ]$

$[\ \tau_1 \mapsto []\ ]$	$[\ \tau_2 \mapsto []\ ]$
1. lock l;	4. lock l;
$[ok\!: \tau_1 \mapsto [L(\tau_1, l)]\ ]$	$[ok\!: \tau_2 \mapsto [L(\tau_2, l)]\ ]$
2. unlock l;	5. $[x]$:= 2;
$[ok\!: \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l)]\ ]$	
3. $[x]$:= 1;	6. unlock l;
$[ok\!: \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)]\ ]$	

Methodology:

- ➡ construct sequential histories
- ➡ analyse them for races

CISL

$$\frac{[p_1]\ C_1\ [ok\!:q_1] \qquad [p_2]\ C_2\ [ok\!:q_2]}{[p_1 * p_2]\ C_1 \mid\mid C_2\ [ok\!:q_1 * q_2]}\ \text{Par}$$

CISL$_{RD}$

$[\ \tau_1 \mapsto [] * \tau_2 \mapsto []\ ]$

$[\ \tau_1 \mapsto []\ ]$

1. lock l;

$[ok{:}\ \tau_1 \mapsto [L(\tau_1, l)]\ ]$

2. unlock l;

$[ok{:}\ \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l)]\ ]$

3. $[x]$:= 1;

$[ok{:}\ \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)]\ ]$

||

$[\ \tau_2 \mapsto []\ ]$

4. lock l;

$[ok{:}\ \tau_2 \mapsto [L(\tau_2, l)]\ ]$

5. $[x]$:= 2;

$[ok{:}\ \tau_2 \mapsto [L(\tau_2, l), W(\tau_2, 5, x)]\ ]$

6. unlock l;

$[ok{:}\ \tau_2 \mapsto [L(\tau_2, l), W(\tau_2, 5, x), U(\tau_2, l)]\ ]$

Methodology:

➡ construct sequential histories

➡ analyse them for races

CISL

$$\frac{[p_1]\ C_1\ [ok{:}q_1] \qquad [p_2]\ C_2\ [ok{:}q_2]}{[p_1 * p_2]\ C_1 \parallel C_2\ [ok{:}q_1 * q_2]}\ \text{Par}$$

CISL$_{RD}$

$[\ \tau_1 \mapsto [\,] * \tau_2 \mapsto [\,]\ ]$

$[\ \tau_1 \mapsto [\,]\ ]$
1. lock l;

$[ok: \tau_1 \mapsto [L(\tau_1, l)]\ ]$
2. unlock l;

$[ok: \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l)]\ ]$
3. $[x]$:= 1;

$[ok: \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)]\ ]$

||

$[\ \tau_2 \mapsto [\,]\ ]$
4. lock l;

$[ok: \tau_2 \mapsto [L(\tau_2, l)]\ ]$
5. $[x]$:= 2;

$[ok: \tau_2 \mapsto [L(\tau_2, l), W(\tau_2, 5, x)]\ ]$
6. unlock l;

$[ok: \tau_2 \mapsto [L(\tau_2, l), W(\tau_2, 5, x), U(\tau_2, l)]\ ]$

$[ok: \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)] * \tau_2 \mapsto [L(\tau_2, l), W(\tau_2, 5, x), U(\tau_2, l)]\ ]$

Methodology:
- ➡ construct sequential histories
- ➡ analyse them for races

CISL

$$\frac{[p_1]\ C_1\ [ok:q_1] \qquad [p_2]\ C_2\ [ok:q_2]}{[p_1 * p_2]\ C_1 \mid\mid C_2\ [ok:q_1 * q_2]}\ \text{Par}$$

CISL$_{RD}$

$[\ \tau_1 \mapsto [] * \tau_2 \mapsto []\ ]$

$[\ \tau_1 \mapsto []\ ]$	$[\ \tau_2 \mapsto []\ ]$
1. lock l;	4. lock l;
$[\mathrm{ok}\!:\ \tau_1 \mapsto [\mathrm{L}(\tau_1, l)]\ ]$	$[\mathrm{ok}\!:\ \tau_2 \mapsto [\mathrm{L}(\tau_2, l)]\ ]$
2. unlock l;	5. $[x]$:= 2;
$[\mathrm{ok}\!:\ \tau_1 \mapsto [\mathrm{L}(\tau_1, l), \mathrm{U}(\tau_1, l)]\ ]$	$[\mathrm{ok}\!:\ \tau_2 \mapsto [\mathrm{L}(\tau_2, l), \mathrm{W}(\tau_2, 5, x)]\ ]$
3. $[x]$:= 1;	6. unlock l;
$[\mathrm{ok}\!:\ \tau_1 \mapsto [\mathrm{L}(\tau_1, l), \mathrm{U}(\tau_1, l), \mathrm{W}(\tau_1, 3, x)]\ ]$	$[\mathrm{ok}\!:\ \tau_2 \mapsto [\mathrm{L}(\tau_2, l), \mathrm{W}(\tau_2, 5, x), \mathrm{U}(\tau_2, l)]\ ]$

$[\mathrm{ok}\!:\ \tau_1 \mapsto [\mathrm{L}(\tau_1, l), \mathrm{U}(\tau_1, l), \mathrm{W}(\tau_1, 3, x)] * \tau_2 \mapsto [\mathrm{L}(\tau_2, l), \mathrm{W}(\tau_2, 5, x), \mathrm{U}(\tau_2, l)]\ ]$

Methodology:

- ➡ construct sequential histories
- ➡ analyse them for races

CISL

$$\frac{[p_1]\ C_1\ [\mathrm{ok}\!:q_1] \qquad [p_2]\ C_2\ [\mathrm{ok}\!:q_2]}{[p_1 * p_2]\ C_1 \parallel C_2\ [\mathrm{ok}\!:q_1 * q_2]}\ \text{Par}$$

CISL_{RD}: `race` Predicate

$\tau_1 \mapsto H_1 * \tau_2 \mapsto H_2 \;\Rightarrow\; \texttt{race}(L_1, L_2, H)$ iff:

there exist H'_1, H'_2, H', a, b such that:

- ➡ a and b are conflicting accesses
- ➡ $H_1 = H'_1 \mathbin{++} [a] \mathbin{++} \_$ and $H_2 = H'_2 \mathbin{++} [b] \mathbin{++} \_$
- ➡ $H = H' \mathbin{++} [a, b]$
- ➡ H' is a permutation of $H'_1 \mathbin{++} H'_2$
- ➡ H is well-formed

CISL$_{RD}$

$$[\ \tau_1 \mapsto [] * \tau_2 \mapsto []\ ]$$

$[\ \tau_1 \mapsto []\ ]$	||	$[\ \tau_2 \mapsto []\ ]$
1. lock l;		4. lock l;
$[ok: \tau_1 \mapsto [L(\tau_1, l)]\ ]$		$[ok: \tau_2 \mapsto [L(\tau_2, l)]\ ]$
2. unlock l;		5. $[x]$:= 2;
$[ok: \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l)]\ ]$		$[ok: \tau_2 \mapsto [L(\tau_2, l), W(\tau_2, 5, x)]\ ]$
3. $[x]$:= 1;		6. unlock l;
$[ok: \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)]\ ]$		$[ok: \tau_2 \mapsto [L(\tau_2, l), W(\tau_2, 5, x), U(\tau_2, l)]\ ]$

$$[ok: \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)] * \tau_2 \mapsto [L(\tau_2, l), W(\tau_2, 5, x), U(\tau_2, l)]\ ]$$

Methodology:

- ➡ construct sequential histories
- ➡ analyse them for races

CISL

$$\frac{[p_1]\ C_1\ [ok{:}q_1] \qquad [p_2]\ C_2\ [ok{:}q_2]}{[p_1 * p_2]\ C_1 \mid\mid C_2\ [ok{:}q_1 * q_2]}\ \text{Par}$$

CISL_{RD}

$$[\ \tau_1 \mapsto [\,] * \tau_2 \mapsto [\,]\ ]$$

$[\ \tau_1 \mapsto [\,]\ ]$	$[\ \tau_2 \mapsto [\,]\ ]$
1. lock l;	4. lock l;
$[ok\!: \tau_1 \mapsto [L(\tau_1, l)]\ ]$	$[ok\!: \tau_2 \mapsto [L(\tau_2, l)]\ ]$
2. unlock l;	5. $[x]$:= 2;
$[ok\!: \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l)]\ ]$	$[ok\!: \tau_2 \mapsto [L(\tau_2, l), W(\tau_2, 5, x)]\ ]$
3. $[x]$:= 1;	6. unlock l;
$[ok\!: \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)]\ ]$	$[ok\!: \tau_2 \mapsto [L(\tau_2, l), W(\tau_2, 5, x), U(\tau_2, l)]\ ]$

$$[ok\!: \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)] * \tau_2 \mapsto [L(\tau_2, l), W(\tau_2, 5, x), U(\tau_2, l)]\ ]$$

$$[ok\!: \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)] * \tau_2 \mapsto [L(\tau_2, l), W(\tau_2, 5, x), U(\tau_2, l)]$$
$$\wedge\ \mathtt{race}(3, 5, [L(\tau_1, l), U(\tau_1, l), L(\tau_2, l), W(\tau_1, 3, x), W(\tau_2, 5, x)])\ ]$$

Methodology:

➡ construct sequential histories

➡ analyse them for races

CISL

$$\frac{[p_1]\ C_1\ [ok\!:q_1] \qquad [p_2]\ C_2\ [ok\!:q_2]}{[p_1 * p_2]\ C_1 \,||\, C_2\ [ok\!:q_1 * q_2]}\ \text{Par}$$

CISL$_{RD}$

$[\ \tau_1 \mapsto [] * \tau_2 \mapsto []\ ]$

$[\ \tau_1 \mapsto []\ ]$ || $[\ \tau_2 \mapsto []\ ]$

Simple

yet

Effective in Practice

à la RacerD

Methodology:

➡ construct sequential histories

➡ analyse them for races

CISL

$$\frac{[p_1]\ C_1\ [ok{:}q_1] \qquad [p_2]\ C_2\ [ok{:}q_2]}{[p_1 * p_2]\ C_1 \parallel C_2\ [ok{:}q_1 * q_2]}\ \text{Par}$$

CISL Framework

- ***First*** unifying framework for ***concurrent under-approximate*** reasoning
- ***General*** framework for multiple bug catching analyses
  - ➡ Memory safety errors (e.g. null-pointer exception, use-after-free errors): CISL_{SV}
  - ➡ Races: CISL_{RD}
  - ➡ Deadlocks: CISL_{DD}
- Sound: ***no false positives*** (NFP) guaranteed
- Underpins ***scalable*** bug-catching tools (NFP for free)
  - ➡ CISL_{RD}: analogous to ***RacerD*** @Meta
  - ➡ CISL_{DD}: analogous to ***DLTool*** @Meta
- <u>Caveat</u>: cannot detect bugs where there are <u>control flow dependencies</u> between threads

Extension 2: Concurrent Adversarial Separation Logic (CASL)

- *A* ***general*** framework for ***concurrent under-approximate*** reasoning
- It **subsumes** CISL
- Can handle ***both*** data-agnostic and data-dependent bugs
- Instantiated to detect ***software exploits/attacks***
  - ➡ Information disclosure attacks (over stacks & heaps)
  - ➡ Buffer overflow attacks (over stacks & heaps)
  - ➡ Memory safety attacks (e.g. zero allocation)

CASL for Exploit Detection: Information Disclosure Attacks

$$
\begin{array}{l} \mathrm{send}(c, 8); \\ \mathrm{recv}(c, y); \end{array}
\quad \Bigg\| \quad
\begin{array}{l}
\mathrm{local}\ sec := *; \\
\mathrm{local}\ w[8] := \{0\}; \\
\mathrm{recv}(c, x); \\
\mathrm{if}\ (x \leq 8) \\
\quad z := w[x]; \\
\quad \mathrm{send}(c, z);
\end{array}
$$

adversary

vulnerable program

CASL for Exploit Detection: Information Disclosure Attacks

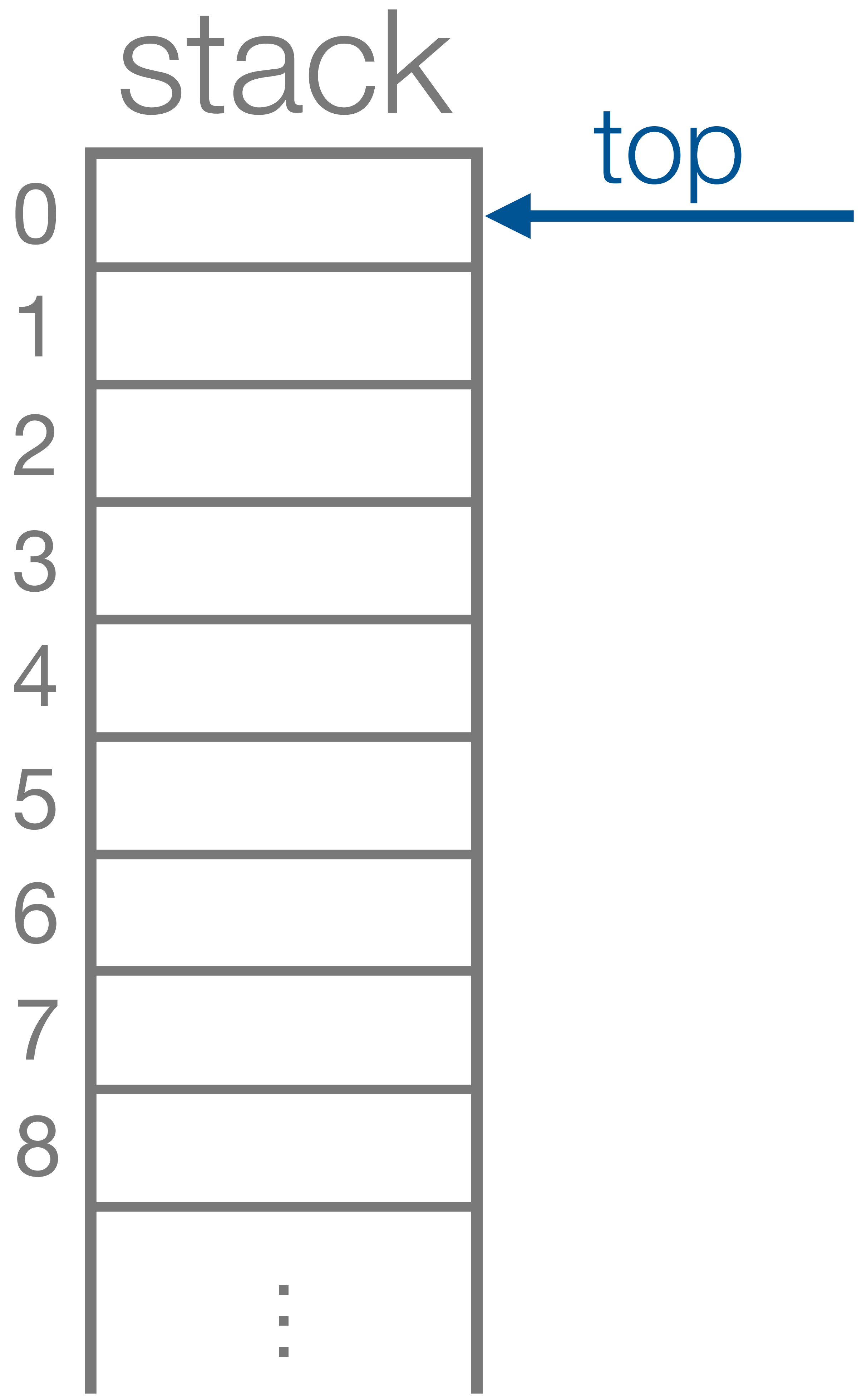
$\text{send}(c, 8);$
$\text{recv}(c, y);$

adversary

$\|$

$\text{local } sec := *;$
$\text{local } w[8] := \{0\};$
$\text{recv}(c, x);$
$\text{if } (x \leq 8)$
$\quad z := w[x];$
$\quad \text{send}(c, z);$

vulnerable program



CASL for Exploit Detection: Information Disclosure Attacks

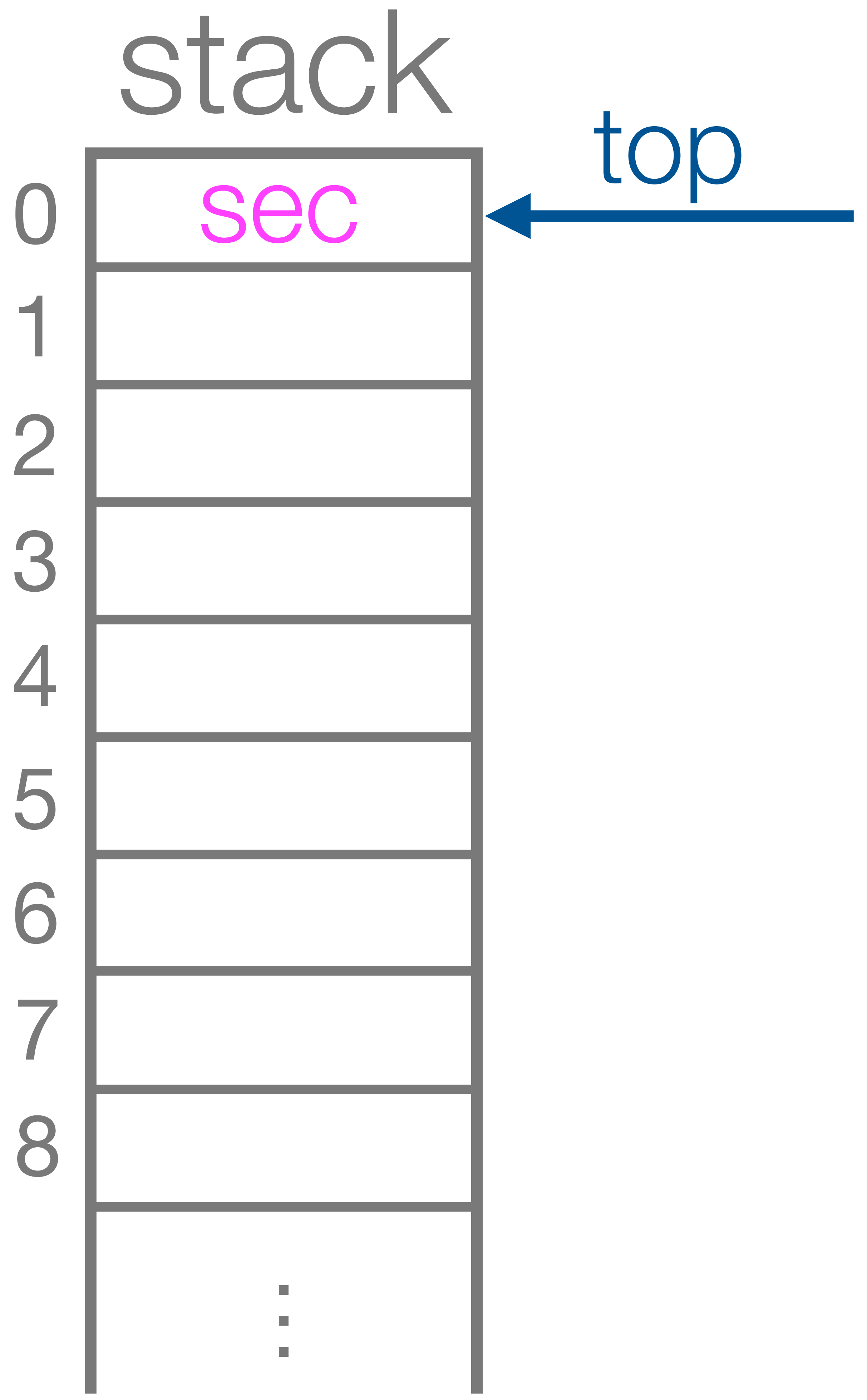
$\text{send}(c, 8);$
$\text{recv}(c, y);$

adversary

$\parallel$

$\text{local } sec := *;$
$\text{local } w[8] := \{0\};$
$\text{recv}(c, x);$
$\text{if } (x \leq 8)$
$\quad z := w[x];$
$\quad \text{send}(c, z);$

vulnerable program



CASL for Exploit Detection: Information Disclosure Attacks

$\text{send}(c, 8);$
$\text{recv}(c, y);$

adversary

$\text{local } sec := *;$
$\text{local } w[8] := \{0\};$
$\text{recv}(c, x);$
$\text{if } (x \leq 8)$
$\quad z := w[x];$
$\quad \text{send}(c, z);$

vulnerable program

CASL for Exploit Detection: Information Disclosure Attacks

$\boxed{\mathrm{send}(c, 8);}$
$\mathrm{recv}(c, y);$

adversary

$\|$

$\mathrm{local}\ sec := *;$
$\mathrm{local}\ w[8] := \{0\};$
$\boxed{\mathrm{recv}(c, x);}$ // 8
$\mathrm{if}\ (x \leq 8)$
$\quad z := w[x];$
$\quad \mathrm{send}(c, z);$

vulnerable program

CASL for Exploit Detection: Information Disclosure Attacks

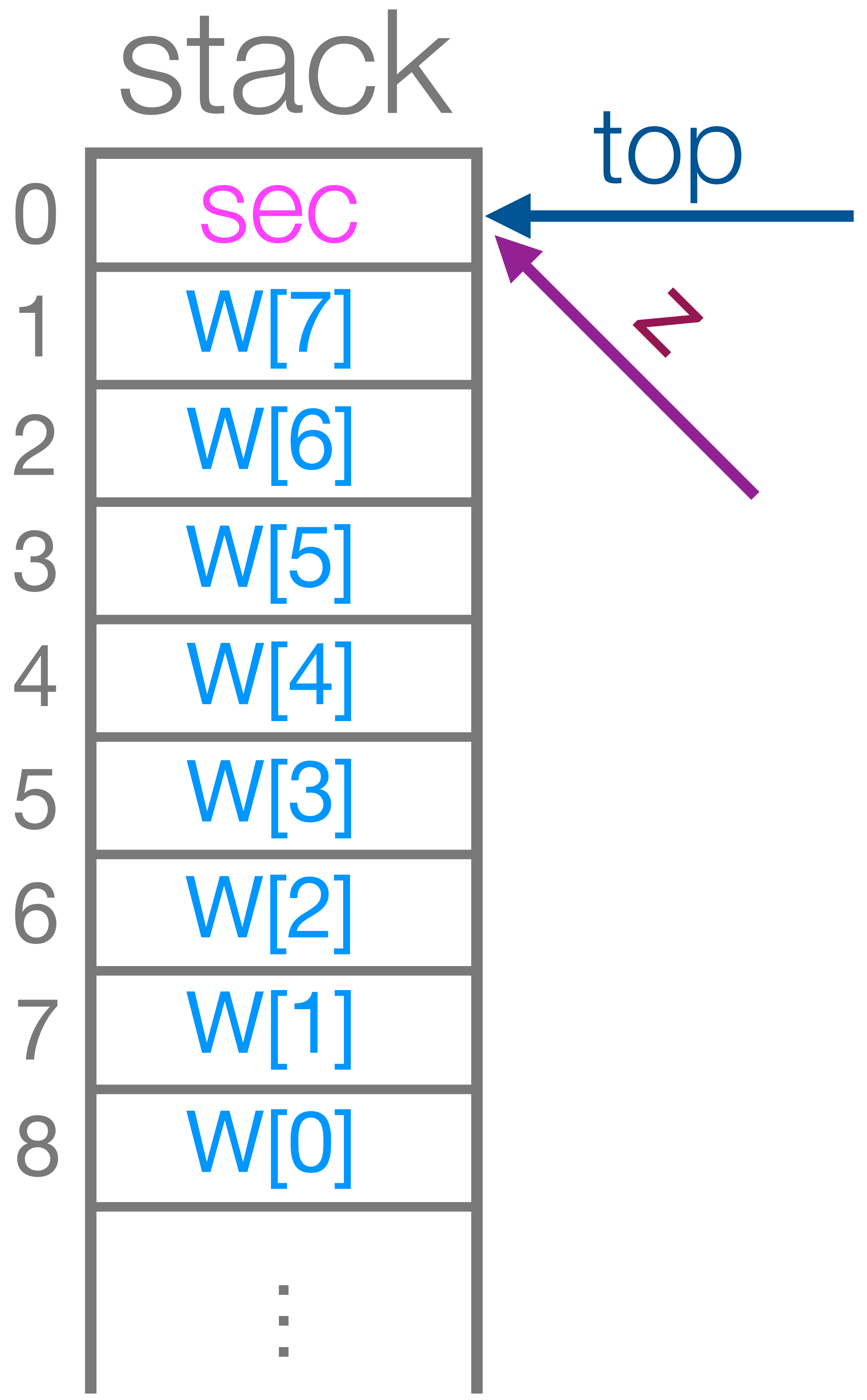
$\mathsf{send}(c, 8);$
$\mathsf{recv}(c, y);$

adversary

$\parallel$

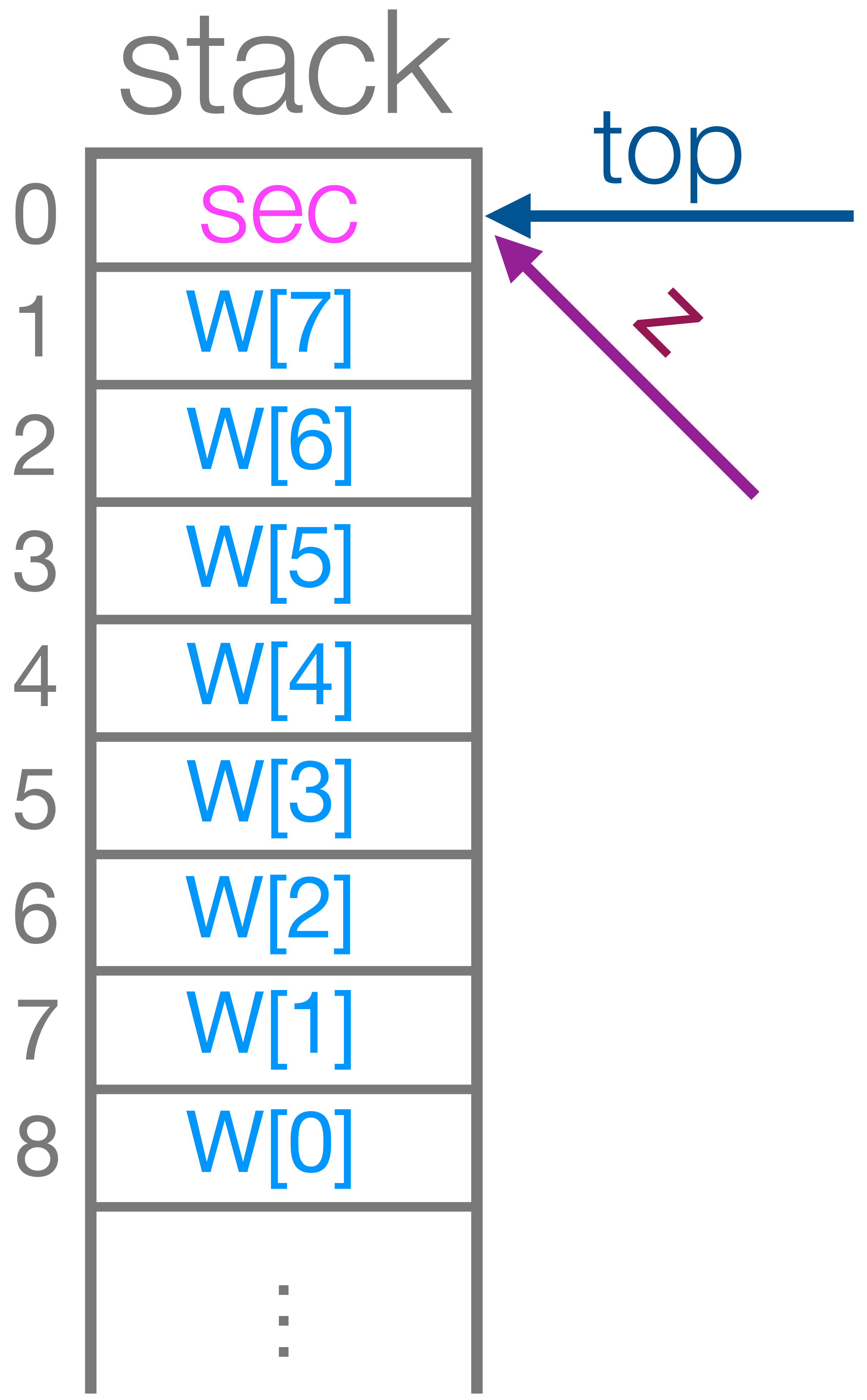
$\mathsf{local}\ sec := *;$
$\mathsf{local}\ w[8] := \{0\};$
$\mathsf{recv}(c, x);$ // 8
$\mathsf{if}\ (x \leq 8)$
$\quad z := w[x];$ // sec
$\quad \mathsf{send}(c, z);$

vulnerable program



CASL for Exploit Detection: Information Disclosure Attacks

adversary		vulnerable program
$\mathrm{send}(c, 8);$	$\Vert$	$\mathrm{local}\ sec := *;$
$\boxed{\mathrm{recv}(c, y);}$ // sec		$\mathrm{local}\ w[8] := \{0\};$
		$\mathrm{recv}(c, x);$ // 8
		$\mathrm{if}\ (x \leq 8)$
		$\quad z := w[x];$ // sec
		$\quad \boxed{\mathrm{send}(c, z);}$



CASL for Exploit Detection: Information Disclosure Attacks

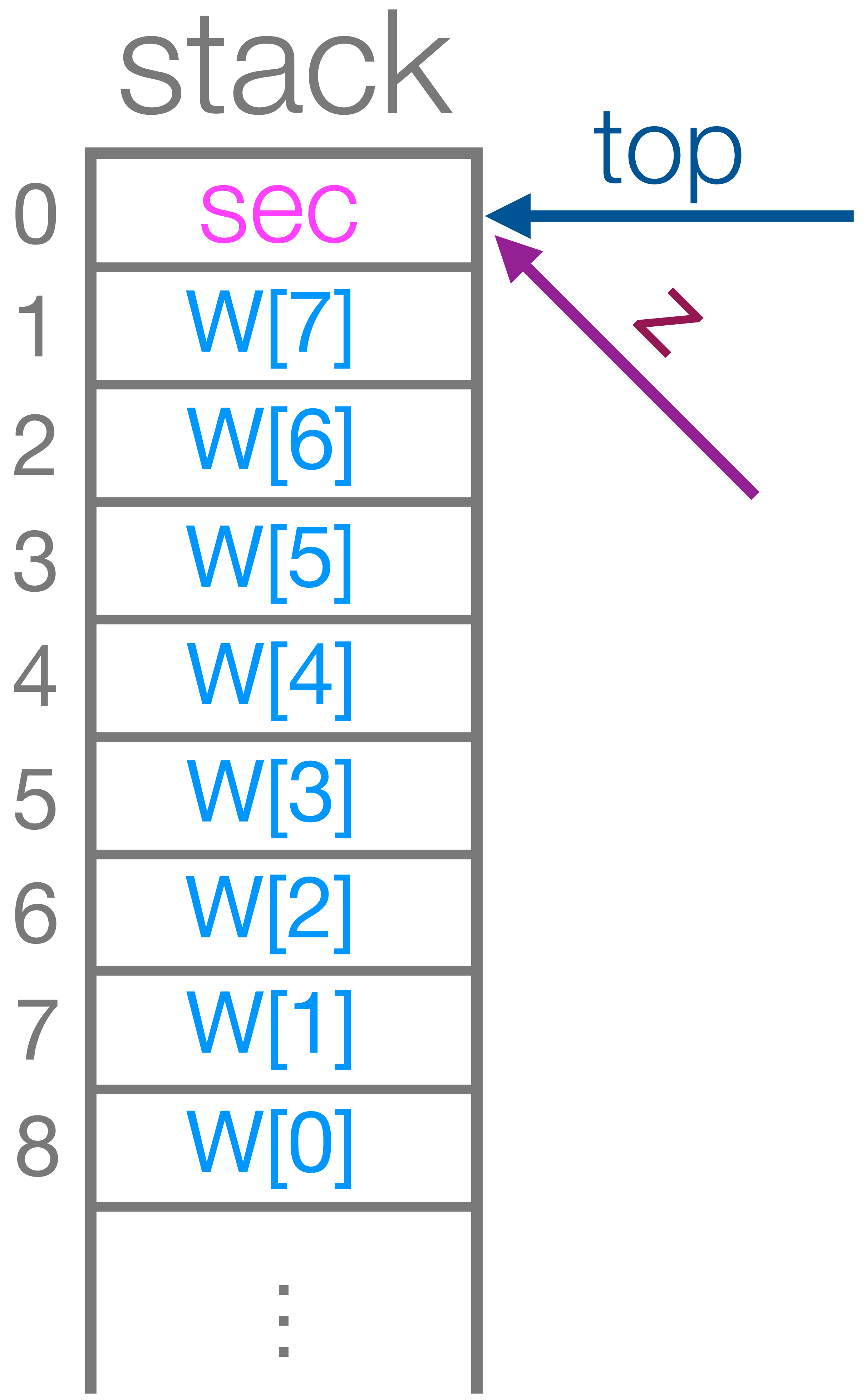
$\mathsf{send}(c, 8);$
$\mathsf{recv}(c, y);$ // sec

adversary

$\parallel$

$\mathsf{local}\ sec := *;$
$\mathsf{local}\ w[8] := \{0\};$
$\mathsf{recv}(c, x);$ // 8
$\mathsf{if}\ (x \leq 8)$
$\quad z := w[x];$ // sec
$\quad \mathsf{send}(c, z);$

vulnerable program



information disclosure!

CASL for Exploit Detection: Memory Safety Attacks

$$\mathrm{send}(c, \mathit{maxInt}); \;\Bigg\|\; \begin{array}{l} \mathrm{recv}(c, s); \\ \mathrm{if}\ (s \leq \mathit{maxInt}) \\ \quad y := s{+}1; \\ \quad x := \mathrm{alloc}(y); \\ \quad \textsc{l}\text{: } [x{+}s] := 0; \end{array}$$

adversary — vulnerable program

CASL for Exploit Detection: Memory Safety Attacks

$$\begin{array}{l|l}
 & \boxed{\mathrm{recv}(c, s);} \quad \text{// maxInt} \\
 & \text{if } (s \leq \mathit{maxInt}) \\
\boxed{\mathrm{send}(c, \mathit{maxInt});} & \quad y := s{+}1; \\
 & \quad x := \mathrm{alloc}(y); \\
 & \quad \textsc{l}\text{: } [x{+}s] := 0;
\end{array}$$

adversary vulnerable program

CASL for Exploit Detection: Memory Safety Attacks

$$
\textsf{send}(c, \mathit{maxInt}); \;\Big\|\; \begin{array}{l} \textsf{recv}(c, s); \quad \text{// maxInt} \\ \textsf{if}\ (s \le \mathit{maxInt}) \\ \quad \boxed{y := s{+}1;} \quad \text{// y=0} \\ \quad x := \textsf{alloc}(y); \\ \quad \textsc{l}\text{: } [x{+}s] := 0; \end{array}
$$

adversary — vulnerable program

CASL for Exploit Detection: Memory Safety Attacks

$$\mathsf{send}(c, \mathit{maxInt}); \;\Big\|\; \begin{array}{l} \mathsf{recv}(c, s); \quad \text{// maxInt} \\ \mathsf{if}\ (s \leq \mathit{maxInt}) \\ \quad y := s{+}1; \quad \text{// y=0} \\ \quad \boxed{x := \mathsf{alloc}(y);} \quad \text{// x=null} \\ \quad \textsc{l}\text{: } [x{+}s] := 0; \end{array}$$

adversary — vulnerable program

CASL for Exploit Detection: Memory Safety Attacks

$$\text{send}(c, \mathit{maxInt}); \quad \Bigg\| \quad \begin{array}{l} \text{recv}(c, s); \quad \text{// maxInt} \\ \text{if } (s \leq \mathit{maxInt}) \\ \quad y := s{+}1; \quad \text{// y=0} \\ \quad x := \text{alloc}(y); \quad \text{// x=null} \\ \quad \boxed{\text{L: } [x{+}s] := 0;} \end{array}$$

adversary

vulnerable program

null pointer dereference!

Termination vs Non-Termination

- Showing **termination** is compatible with **correctness** frameworks:
  - ➡ **Every** trace of a given program must terminate
  - ➡ Inherently **over-approximate**

```
skip + x:=1
```

Termination vs Non-Termination

- Showing **termination** is compatible with **correctness** frameworks:
  - ➡ **Every** trace of a given program must terminate
  - ➡ Inherently **over-approximate**

```
skip + x:=1
```

- Showing **non-termination** compatible with **incorrectness** frameworks:
  - ➡ **Some** trace of a given program must not-terminate
  - ➡ Inherently **under-approximate**

```
skip + while(true)skip
```

Extension 3: Incorrectness Non-Termination Logic (INTL)

- A framework for **detecting non-termination bugs**
- Supports **unstructured** constructs (goto), as well exceptions and breaks
- Reasons for non-termination:
  - Infinite loops
  - Infinite recursion
  - Cyclic `goto` soups

INTL Proof Rules and Principles

INTL Proof Rules

=

ISL Proof Rules

+

Divergence (Non-Termination) Rules

INTL Divergence Proof Rules

$$[p]\ C\ [\infty]$$

Starting from some state in p, C has a divergent trace

INTL Divergence Proof Rules (Sequencing)

$$\frac{[p]\ C_1\ [\infty]}{[p]\ C_1;\ C_2\ [\infty]}$$

$$\frac{[p]\ C_1\ [ok\!:\ q] \qquad [q]\ C_2\ [\infty]}{[p]\ C_1;\ C_2\ [\infty]}$$

INTL Divergence Proof Rules (Branches)

$$\frac{[p]\ C_i\ [\infty] \qquad\qquad \textbf{some}\ i \in \{1, 2\}}{[p]\ C_1 + C_2\ [\varepsilon : q]}$$

- **Drop paths/branches** (this is a **sound** under-approximation)
- **Scalable** bug detection!

INTL Divergence Proof Rules (Loops)

$$\frac{[p]\ C\ [\infty]}{[p]\ C^*\ [\infty]}$$

$$\frac{[p]\ C\ [ok: p] \qquad \text{[extra condition omitted]}}{[p]\ C^*\ [\infty]}$$

Conclusions

- CISL and CASL
  - ➡ Combining ISL and CSL for ***concurrent bug catching***
  - ➡ ***no-false-positives*** theorem
  - ➡ Race detection, deadlock detection, exploit detection
- INTL
  - ➡ ISL for detecting **non-termination bugs**
  - ➡ **Unstructured** language (`goto` construct)
  - ➡ ***no-false-positives*** theorem
  - ➡ Infinite loop detection
  - ➡ Infinite recursion detection — ongoing work
  - ➡ Cyclic goto-soup — ongoing work

azalea@imperial.ac.uk SoundAndComplete.org @azalearaad

Announcement

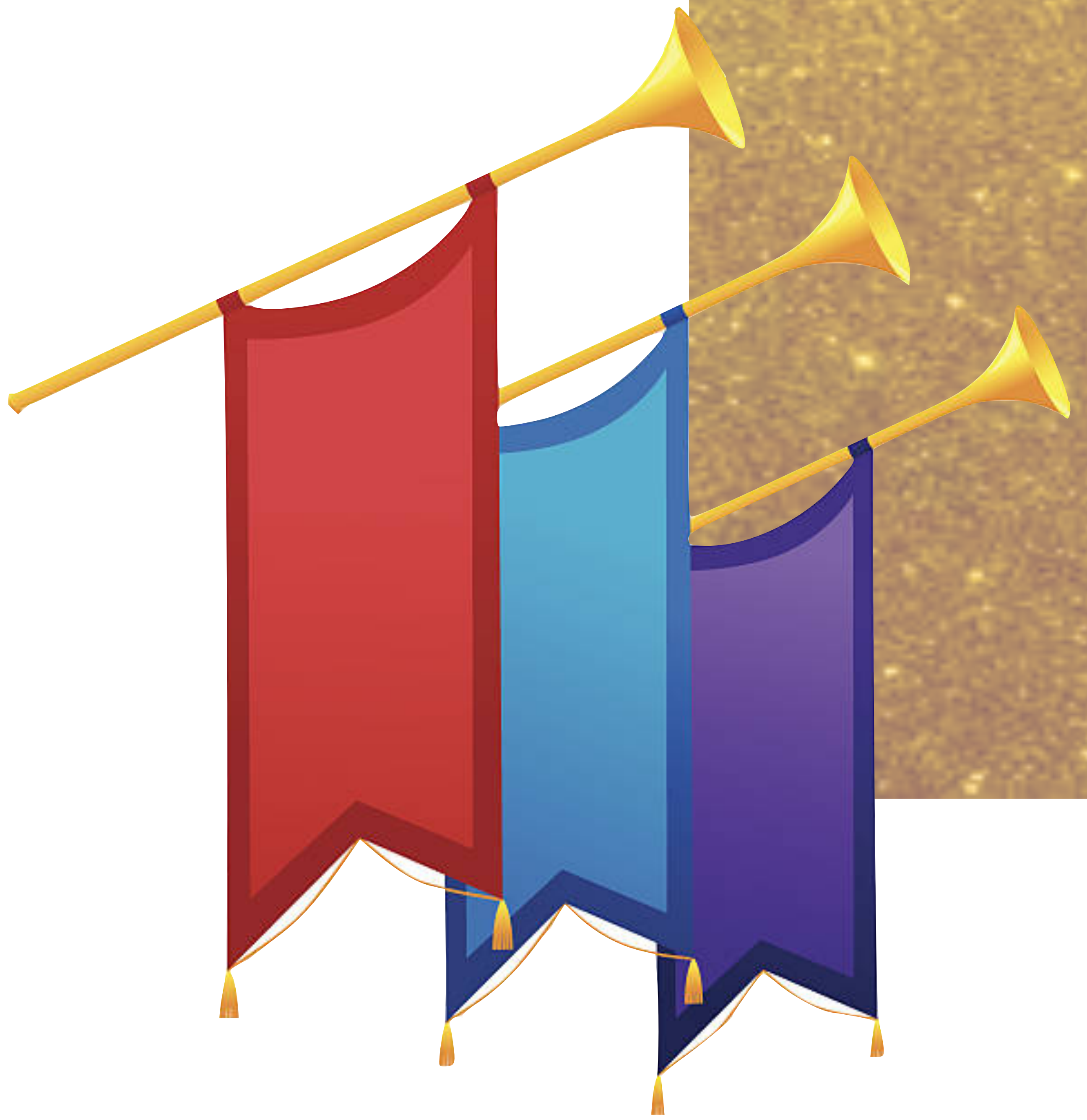
<u>I'm hiring!</u>

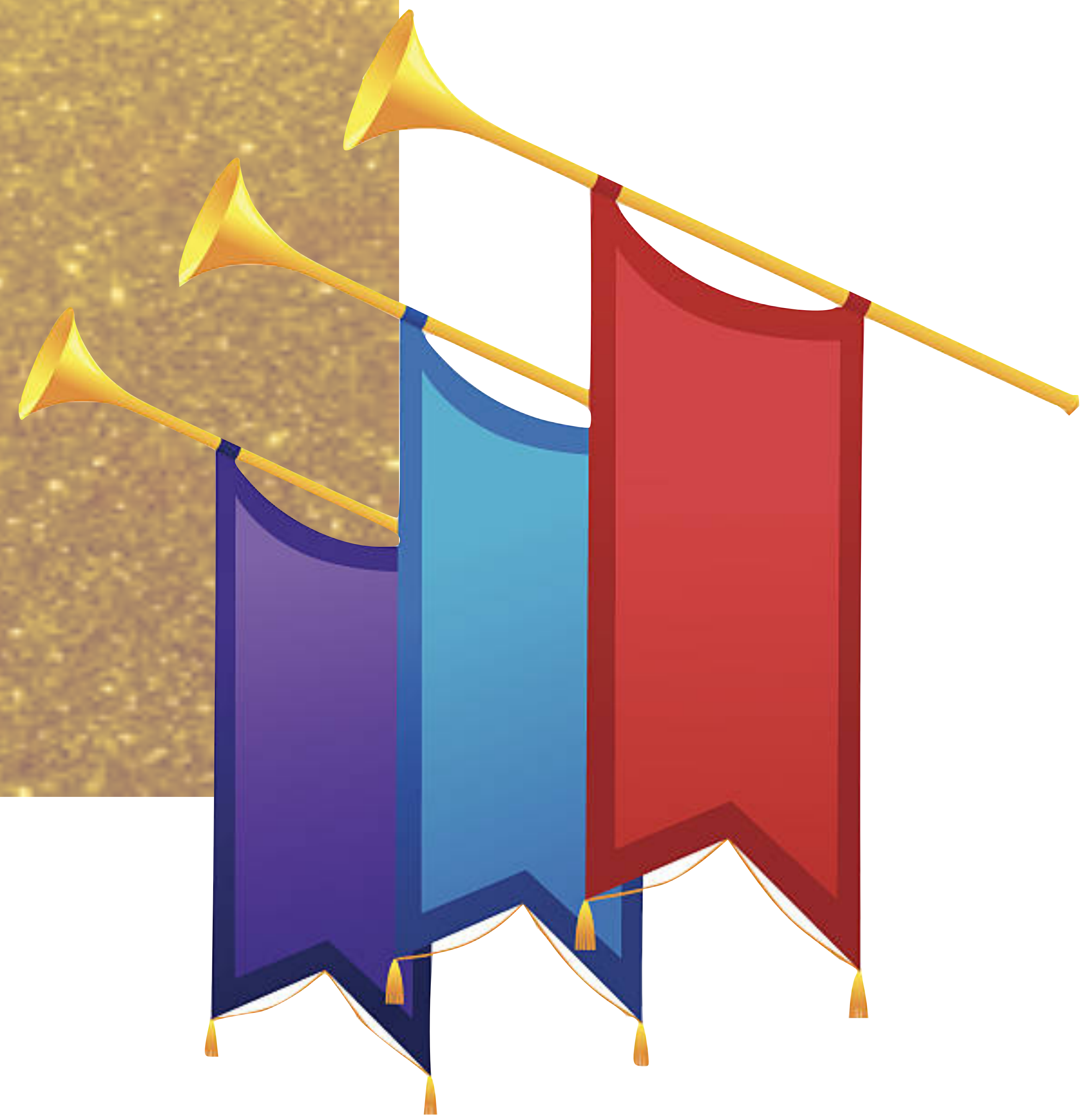
Join my team (Veritas) @Imperial College

★ **Internship** opportunities

★ **PhD** studentships

★ **Post-doc** positions





Thank You for Listening!

azalea@imperial.ac.uk SoundAndComplete.org @azalearaad