Concurrent Incorrectness Separation Logic

Azalea Raad[1,2] Josh Berdine[2] Derek Dreyer[3] Peter O'Hearn[2]

[1] Imperial College London

[2] Meta

[3] MPI-SWS

POPL, 2022

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

CISL

=

Incorrectness Separation Logic (ISL)

+

Concurrency

for

Concurrent Bug Detection & Analysis

Incorrectness Logic

- Prove the ***presence*** of bugs — bug catching
- ***Under-approximate*** reasoning

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

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

Incorrectness Logic

- Prove the ***presence*** of bugs — bug catching
- ***Under-approximate*** reasoning

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

q *under-approximates* post(C)p $\nearrow$

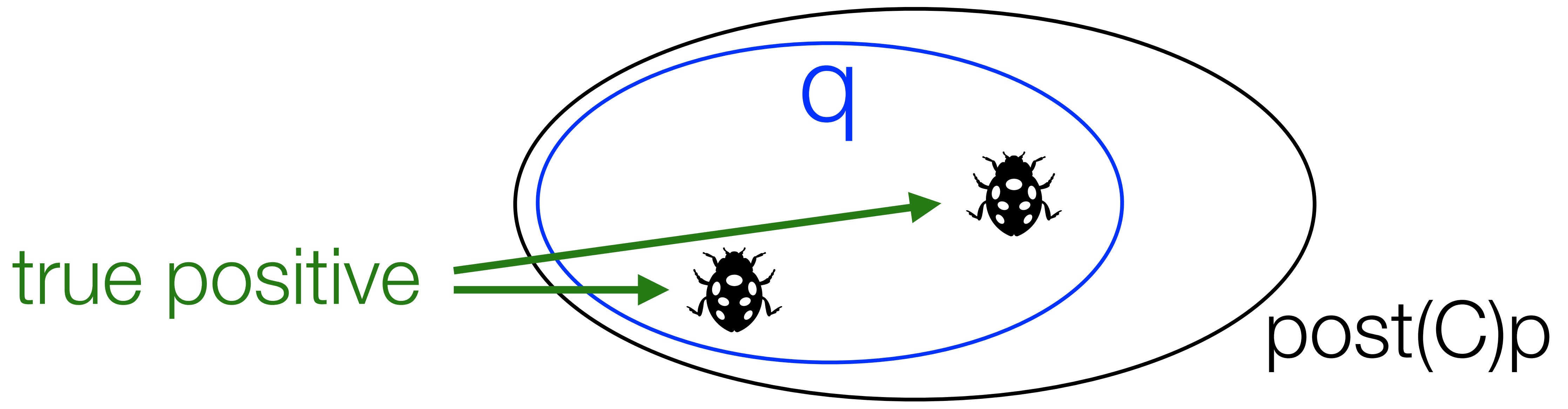
Incorrectness Logic

❖ Prove the ***presence*** of bugs — bug catching

❖ ***Under-approximate*** reasoning

IL $[p]\ C\ [q]$ *iff* $\text{post}(C)p \supseteq q$

q *under-approximates* post(C)p

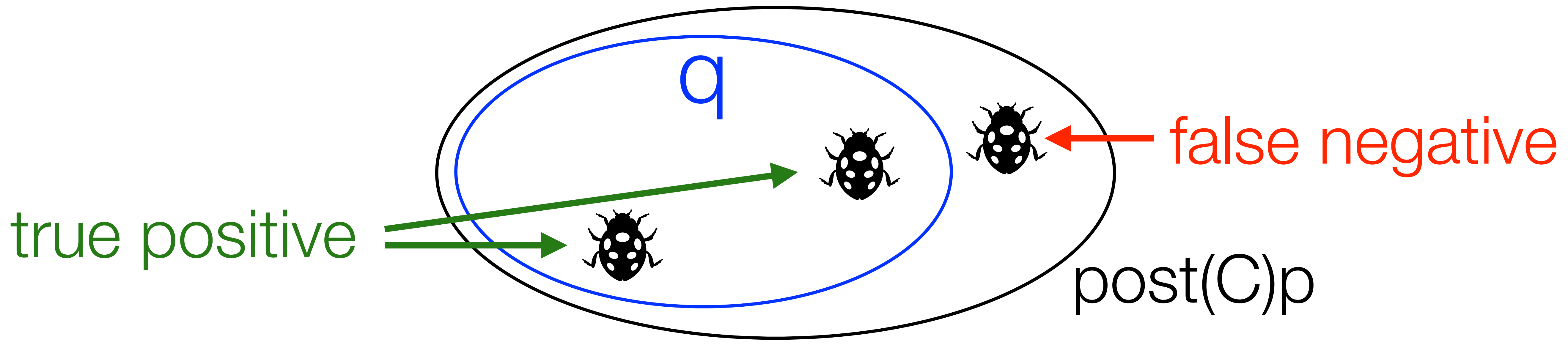


Incorrectness Logic

- Prove the ***presence*** of bugs — bug catching
- ***Under-approximate*** reasoning

IL $[p]\ C\ [q]$ *iff* $\text{post}(C)p \supseteq q$

q *under-approximates* post(C)p

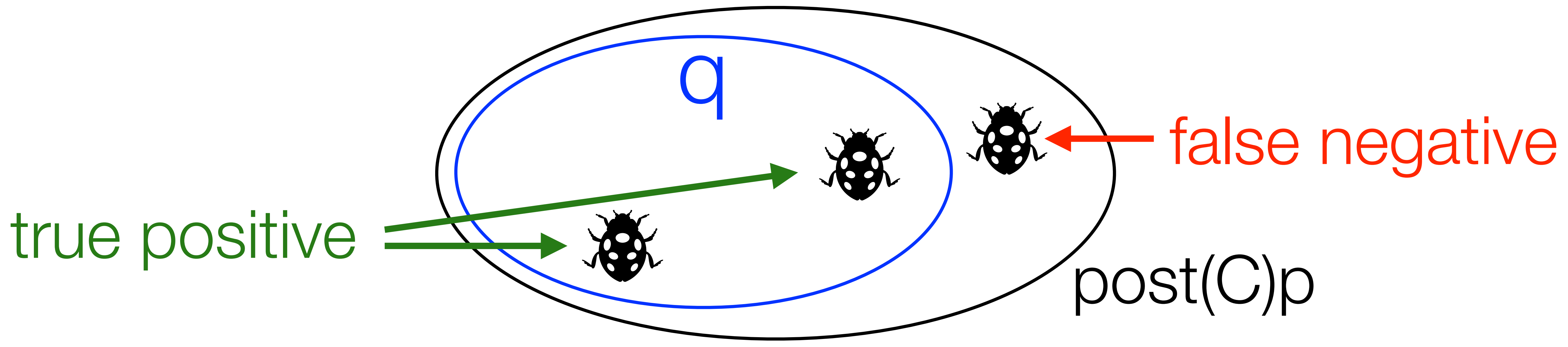


Incorrectness ***Separation*** Logic

- ❖ Prove the ***presence*** of bugs — bug catching
- ❖ ***Under-approximate*** reasoning

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

q *under-approximates* post(C)p



ISL

$$\frac{[p]\ C\ [q]}{[p * r]\ C\ [q * r]}\ \text{Frame}$$

Incorrectness ***Separation*** Logic

- ❖ Prove the ***presence*** of bugs — bug catching
- ❖ ***Under-approximate*** reasoning

ISL $[p]\ C\ [q]$ *iff* $\text{post}(C)p \supseteq q$

Problem 1

No support for ***concurrency***

ISL

$$\frac{[p]\ C\ [q]}{[p * r]\ C\ [q * r]}\ \text{Frame}$$

Concurrent Bug Detection

- Several **bug catching** tools for **concurrency** based on **under-approximation**
  - RacerD [Blackshear et al., 2018]: **race detection** @Meta
  - ToolDL [Brotherston et al., 2021]: **deadlock detection** @Meta

- Each prove a **no-false-positives (NFP) theorem**: bugs found are true bugs

Concurrent Bug Detection

- Several **bug catching** tools for **concurrency** based on **under-approximation**
  - ➡ RacerD [Blackshear et al., 2018]: **race detection** @Meta
  - ➡ ToolD
- Each pr

Problem 2

Each analysis must prove NFP ***independently***

Concurrent Bug Detection

❖ Several **bug catching** tools for **concurrency** based on **under-approximation**

➡ Racer

➡ ToolD

❖ Each pr

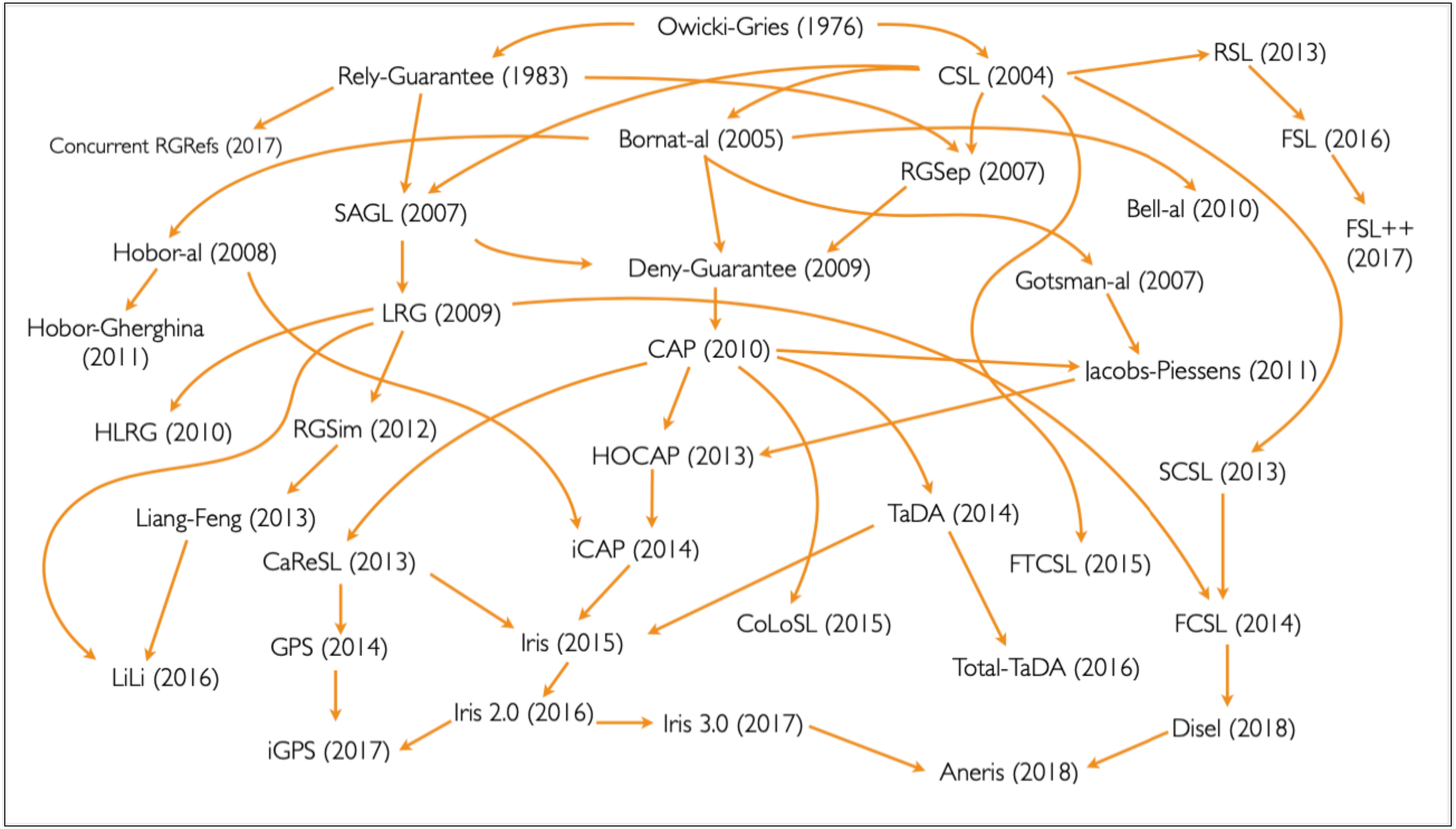
Solution

CISL:

Concurrent Incorrectness Separation Logic

Which CISL?

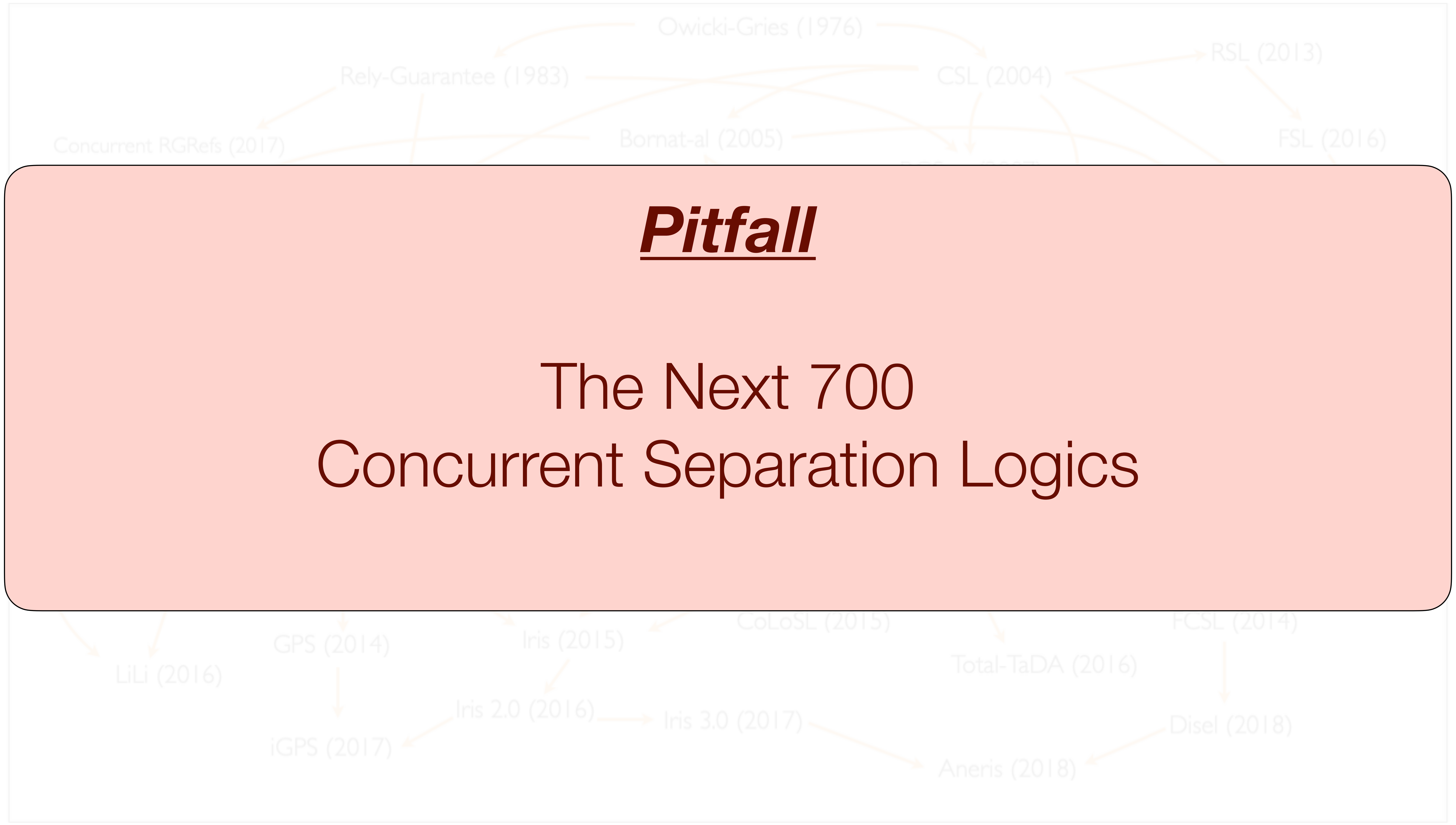
CSL (<u>Correctness</u>) Family Tree…



Graph courtesy of Ilya Sergey

Which CISL?

CSL (Correctness) Family Tree…



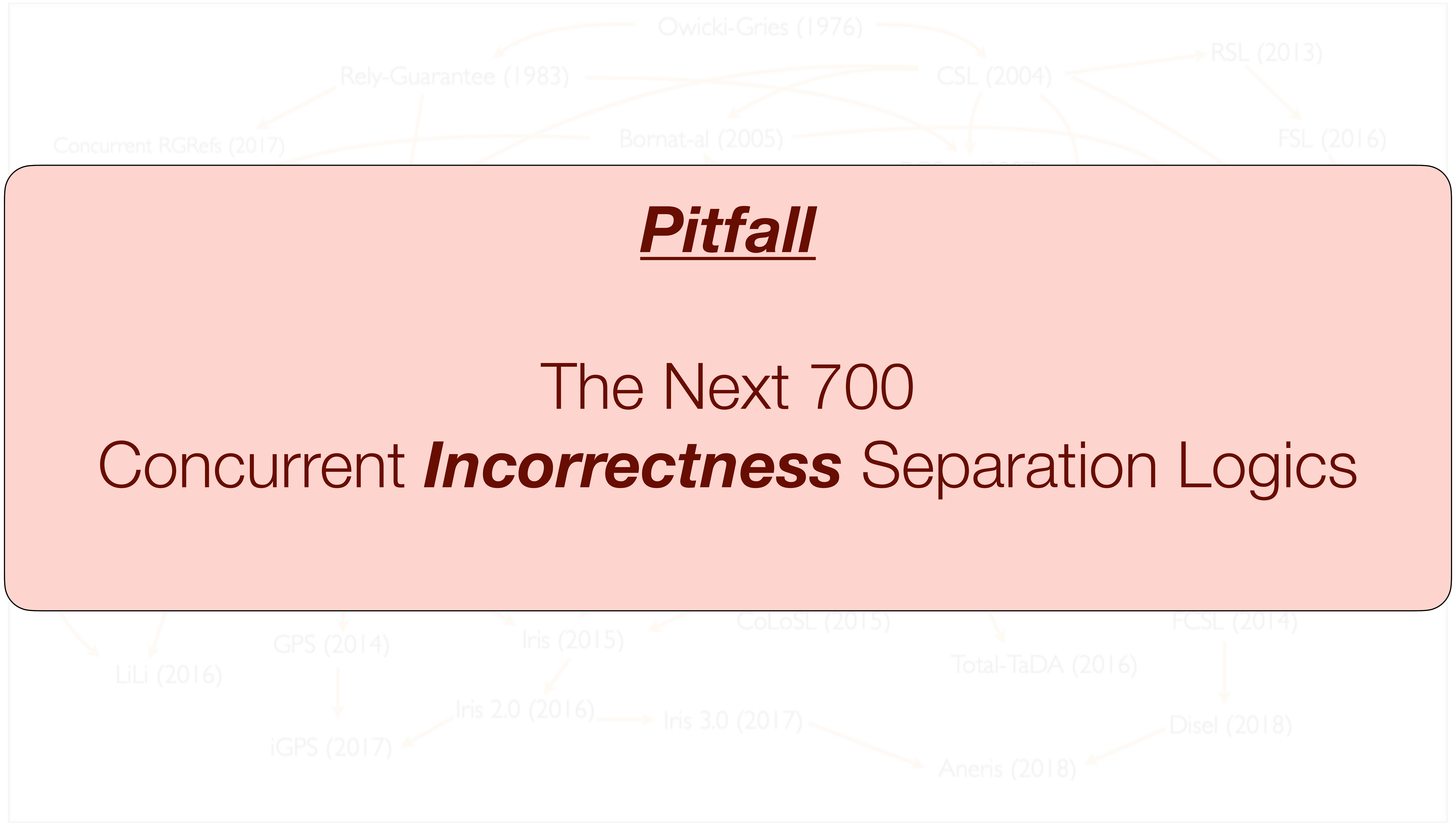
Pitfall

The Next 700
Concurrent Separation Logics

Graph courtesy of Ilya Sergey

Which CISL?

CSL (Correctness) Family Tree…



Pitfall

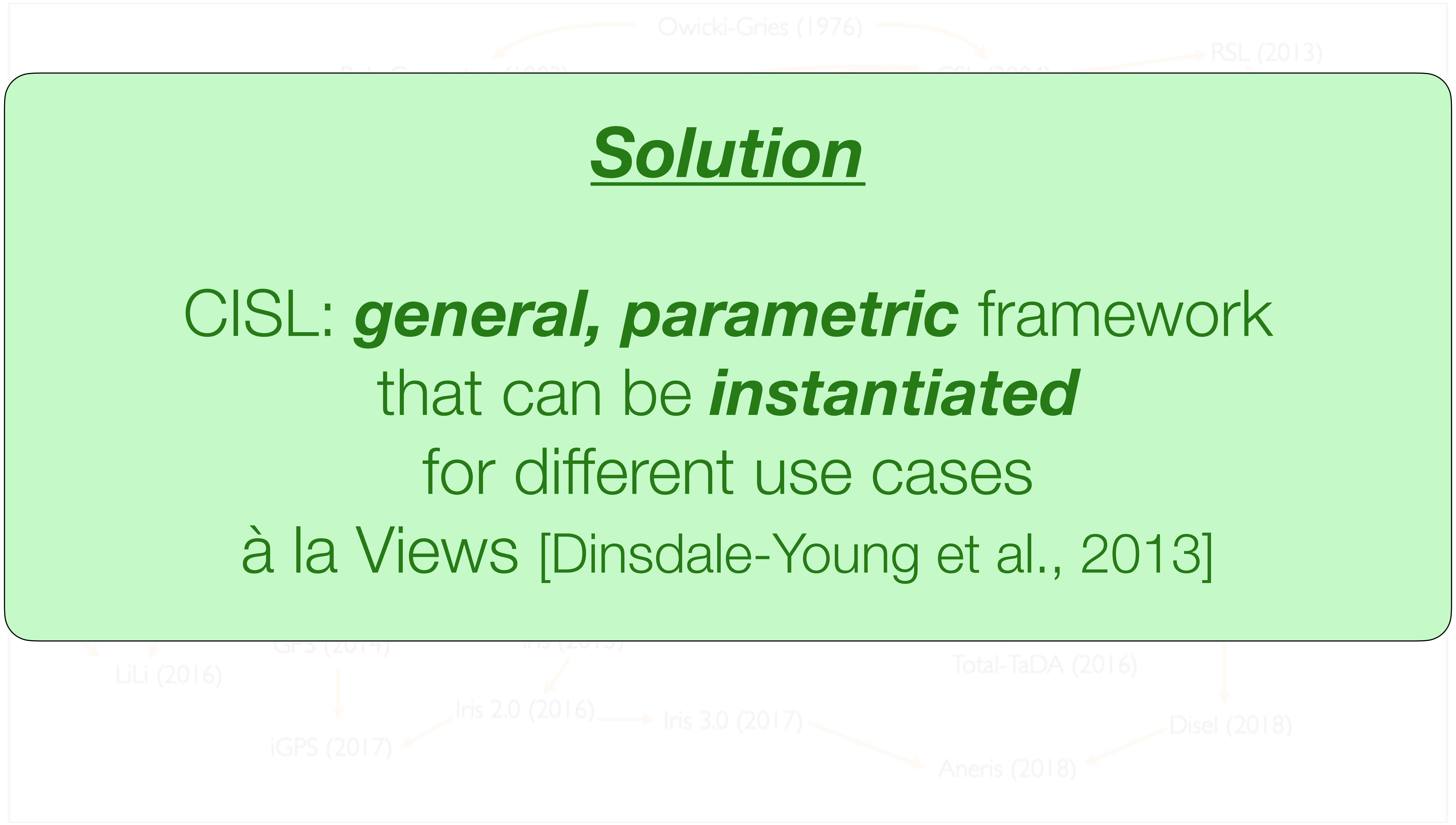
The Next 700

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

(Concurrent) Incorrectness (Separation) Logic

[p] C [ε: q]

ε: exit condition

ok: normal execution

er : erroneous execution

[p] skip [ok: p]

[p] error( ) [er: p]

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

What are they?

➡ They are due to one thread

$$\left.\begin{array}{l}\texttt{free}(x);\\ \textsc{l}\text{: } [x] := 1\end{array}\right\| \;\mathrm{C}$$

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

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

What are they?

➡ They are due to one thread

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

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

Thread-local analysis tools?

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

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

What are they?

➡ They are due to one thread

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

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

Thread-local analysis tools?

➡ Existing (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')

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

(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

$\texttt{L}: \texttt{free}(x) \,\|\, \texttt{L}': \texttt{free}(x)$

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

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

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

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

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

(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

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

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

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

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

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

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

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

   - ➡ possible in CISL theory
   - ➡ no existing analysis tools: ongoing work with Meta

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

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

This talk

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

- ➡ possible in CISL theory
- ➡ no existing analysis tools: ongoing work with Meta

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

- Races are ***global*** bugs by definition:

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

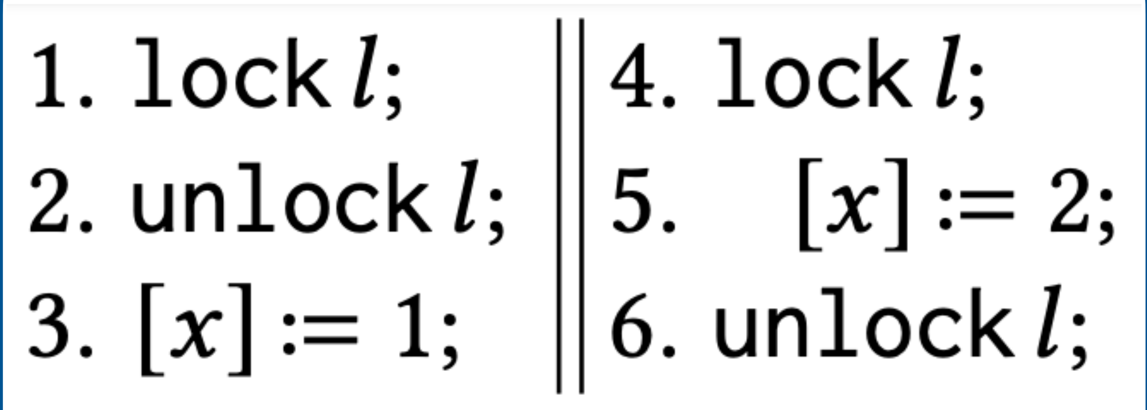
  1. a and b are ***conflicting***:
     - ➡ they are by distinct threads
     - ➡ on the same location
     - ➡ at least one of them is a write
  2. they appear ***next to each other in an interleaving*** (history) of C

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

❖ Races are ***global*** bugs by definition:

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

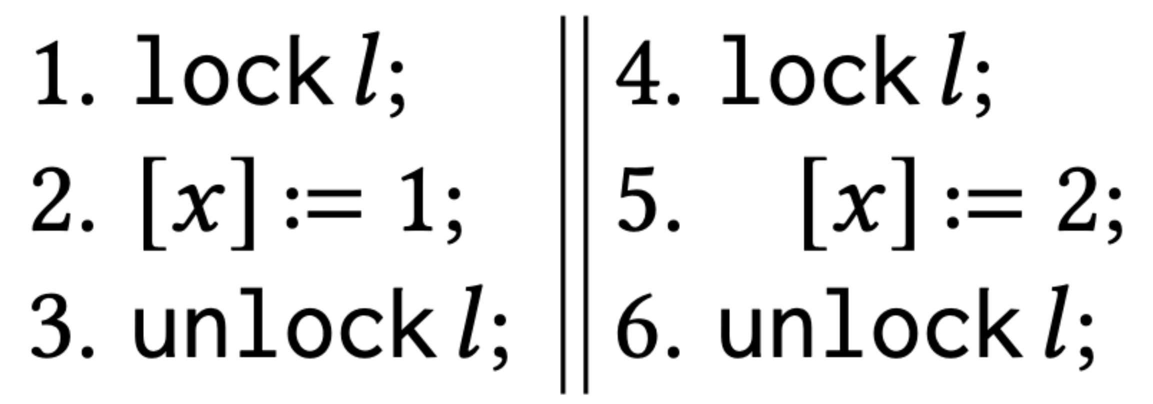
1. a and b are ***conflicting***:
   - ➡ they are by <u>distinct threads</u>
   - ➡ on the <u>same location</u>
   - ➡ at least <u>one of them is a write</u>
2. they appear ***next to each other in an interleaving*** (history) of C



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

Race between lines 3, 5 witnessed by:

H = [1, 2, 4, **3**, **5**, 6]



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

No races

CISL$_{RD}$

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

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

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

CISL_{RD}

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

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

Methodology:

➡ construct sequential histories

➡ analyse them for races

CISL

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

CISL_{RD}

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

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

Methodology:
➡ construct sequential histories
➡ analyse them for races

CISL

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

CISL$_{RD}$: Lock Axiom

CISL$_{RD}$

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

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

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

CISL$_{\mathrm{RD}}$: Unlock Axiom

CISL$_{\mathrm{RD}}$

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

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

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

CISL

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

CISL$_{RD}$: Memory Access Axioms

CISL$_{RD}$

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

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

CISL$_{\text{RD}}$: Memory Access Axioms

CISL$_{\text{RD}}$

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

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

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

CISL_{RD}

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

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

1. lock l;

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

2. unlock l;

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

3. $[x]$:= 1;

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

$||$

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

4. lock l;

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

5. $[x]$:= 2;

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

6. unlock l;

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

Methodology:

➡ construct sequential histories

➡ analyse them for races

CISL

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

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

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

CISL$_{RD}$

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

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

1. lock l;

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

2. unlock l;

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

3. $[x]$:= 1;

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

$||$

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

4. lock l;

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

5. $[x]$:= 2;

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

6. unlock l;

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

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

Methodology:

- ➡ construct sequential histories
- ➡ analyse them for races

CISL

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

CISL$_{RD}$: `race` Predicate

$\tau_1 \mapsto H_1 * \tau_2 \mapsto H_2 \Rightarrow$ `race`(L_1, L_2, H) iff:

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

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

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

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

CISL_{RD}

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

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

1. lock l;

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

2. unlock l;

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

3. $[x]$:= 1;

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

$\|$

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

4. lock l;

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

5. $[x]$:= 2;

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

6. unlock l;

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

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

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

$\wedge\ \texttt{race}(3, 5, [\text{L}(\tau_1, l), \text{U}(\tau_1, l), \text{L}(\tau_2, l), \text{W}(\tau_1, 3, x), \text{W}(\tau_2, 5, x)]) \quad]$

Methodology:

- ➡ construct sequential histories
- ➡ analyse them for races

CISL

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

CISL$_{RD}$

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

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

Simple

yet

Effective in Practice

à la RacerD

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

CISL

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

Conclusions

- ***First*** work to adapt ***under-approximate*** reasoning for ***concurrent bug detection***
- ***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}: à la ***RacerD*** @Meta; CISL_{DD}: à la ***DLTool*** @Meta

Conclusions

- ***First*** work to adapt ***under-approximate*** reasoning for ***concurrent bug detection***
- ***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}: à la ***RacerD*** @Meta; CISL_{DD}: à la ***DLTool*** @Meta
- Future work:
  - CISL for data-dependent bugs
  - automated tools based on CISL, e.g. data-dependent races, deadlocks, memory safety errors
  - mechanisation

Conclusions

- ***First*** work to adapt ***under-approximate*** reasoning for ***concurrent bug detection***
- ***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}: à la ***RacerD*** @Meta; CISL_{DD}: à la ***DLTool*** @Meta
- Future work:
  - ➡ CISL for data-dependent bugs
  - ➡ automated tools based on CISL, e.g. data-dependent races, deadlocks, memory safety errors
  - ➡ mechanisation

Thank You for Listening!

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