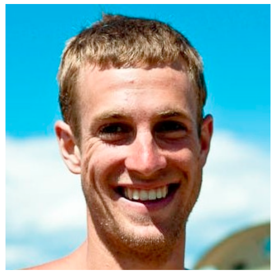
Goal-Directed Program Analysis with Jumping



Sam Blackshear
University of Colorado Boulder

Bor-Yuh Evan Chang
University of Colorado Boulder

Manu Sridharan
Samsung Research America

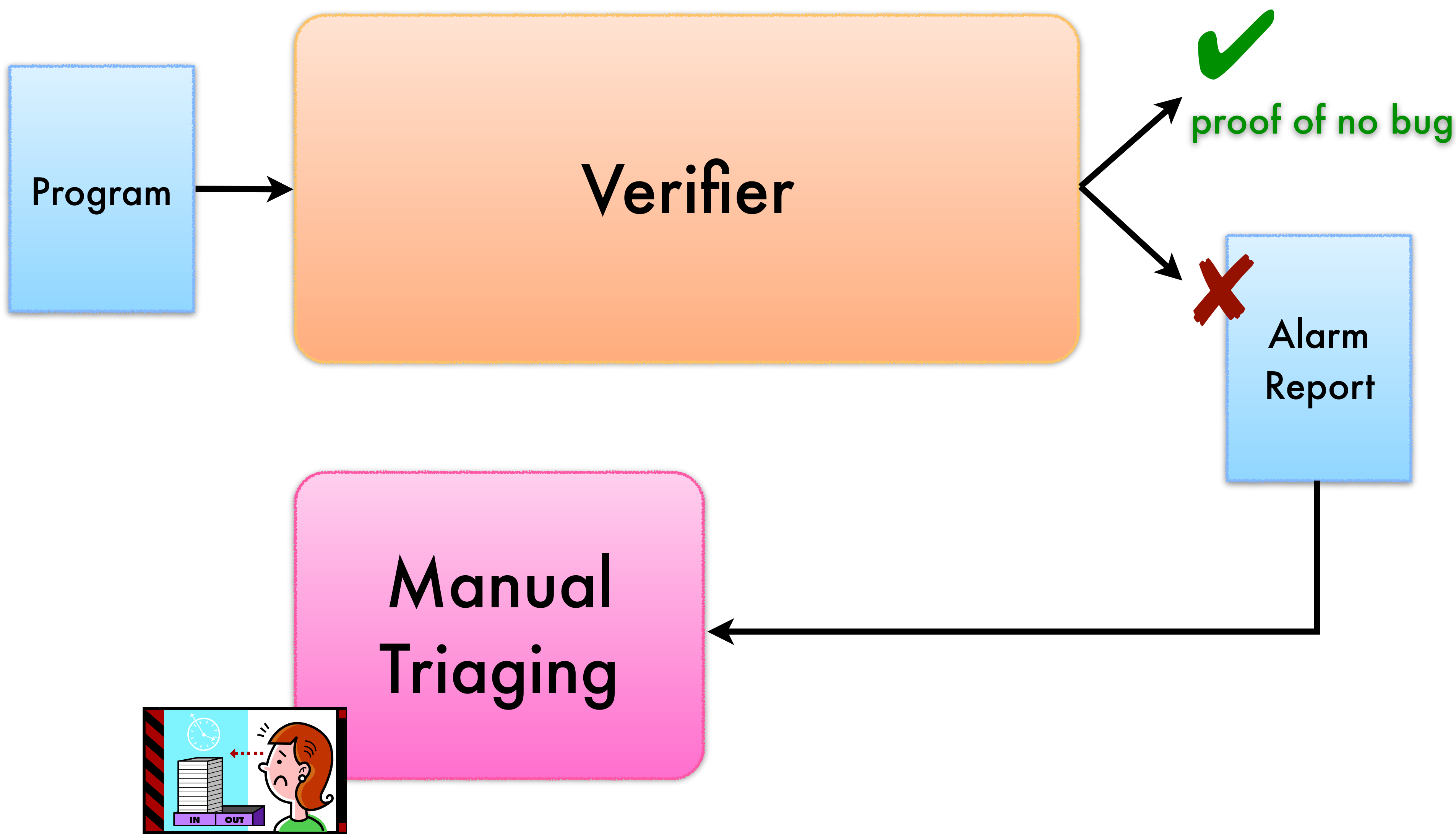
École normale supérieure
1 October 2015

PLV

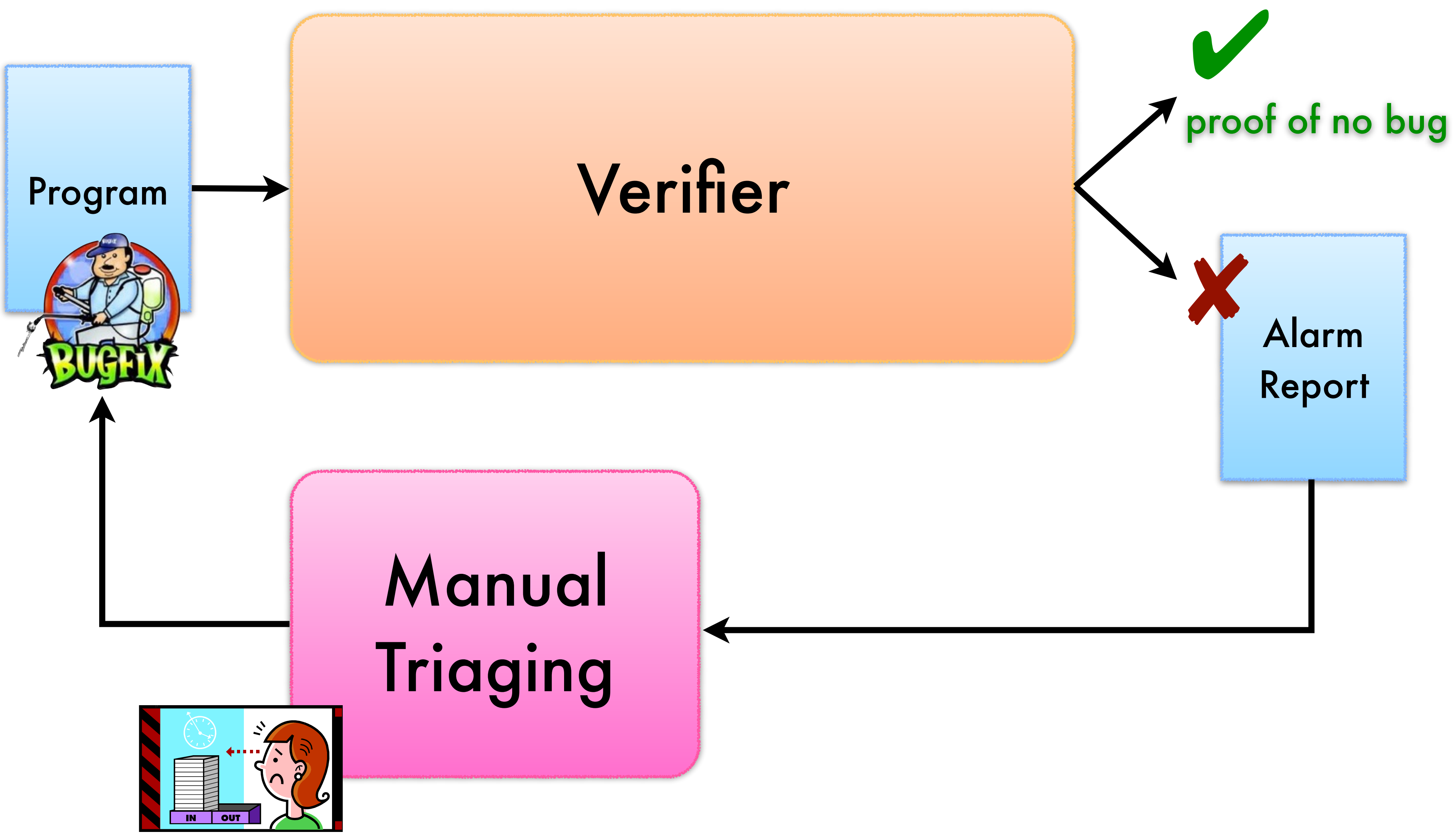
Lab: **Program analysis** in the **whole** bug mitigation **process**

Lab: **Program analysis** in the **whole** bug mitigation **process**

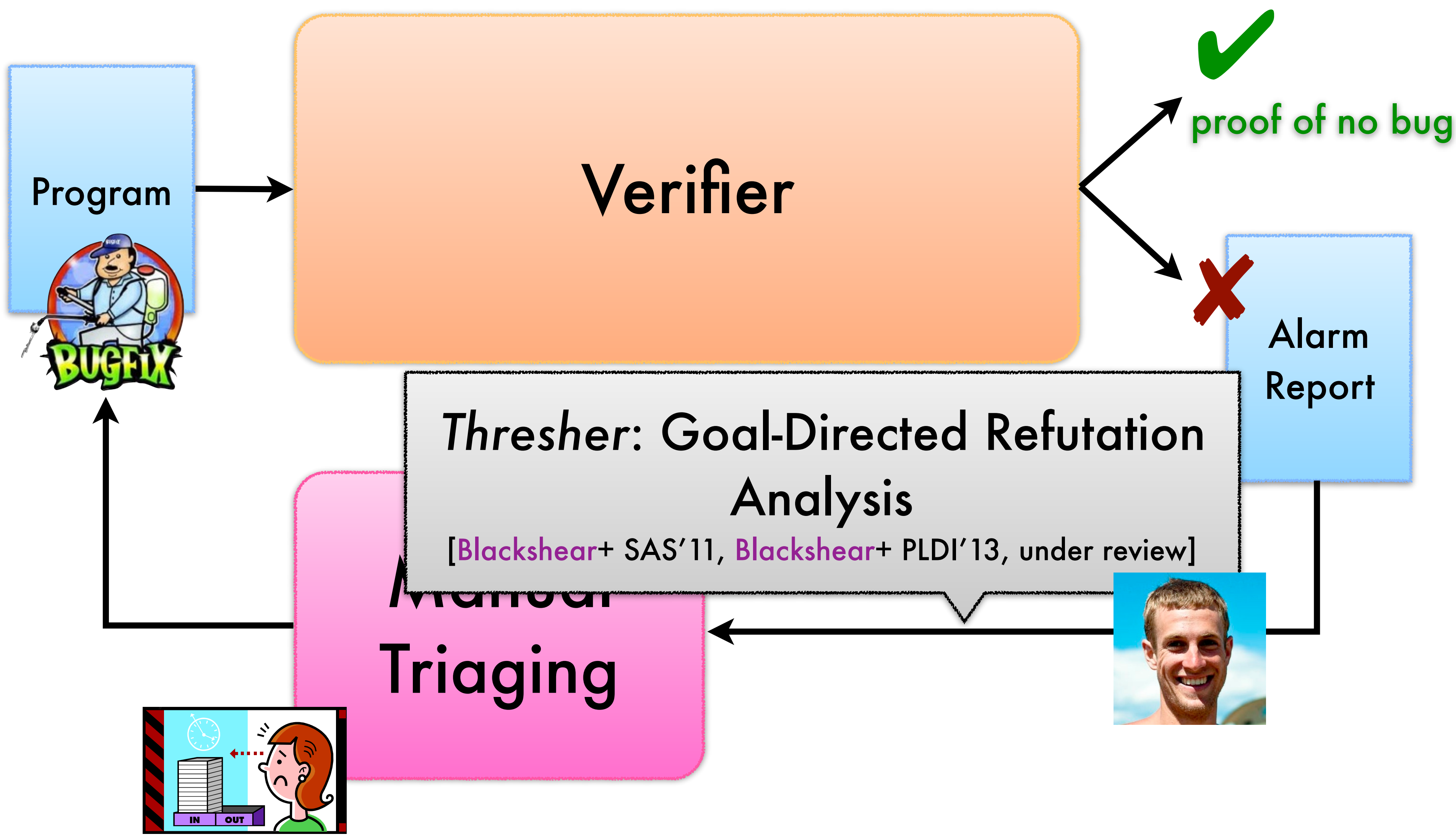
Lab: **Program analysis** in the **whole** bug mitigation **process**



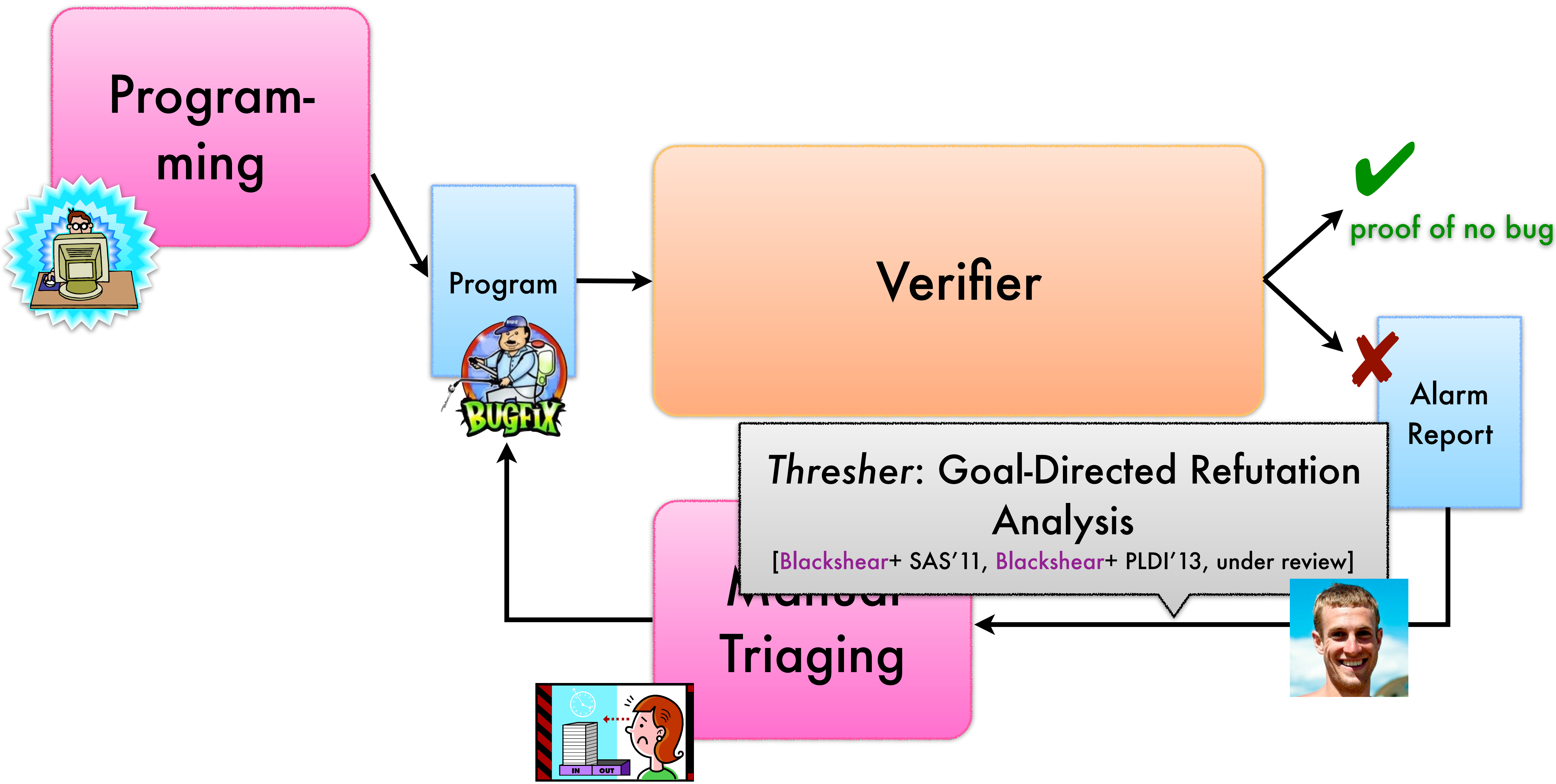
Lab: **Program analysis** in the **whole** bug mitigation **process**



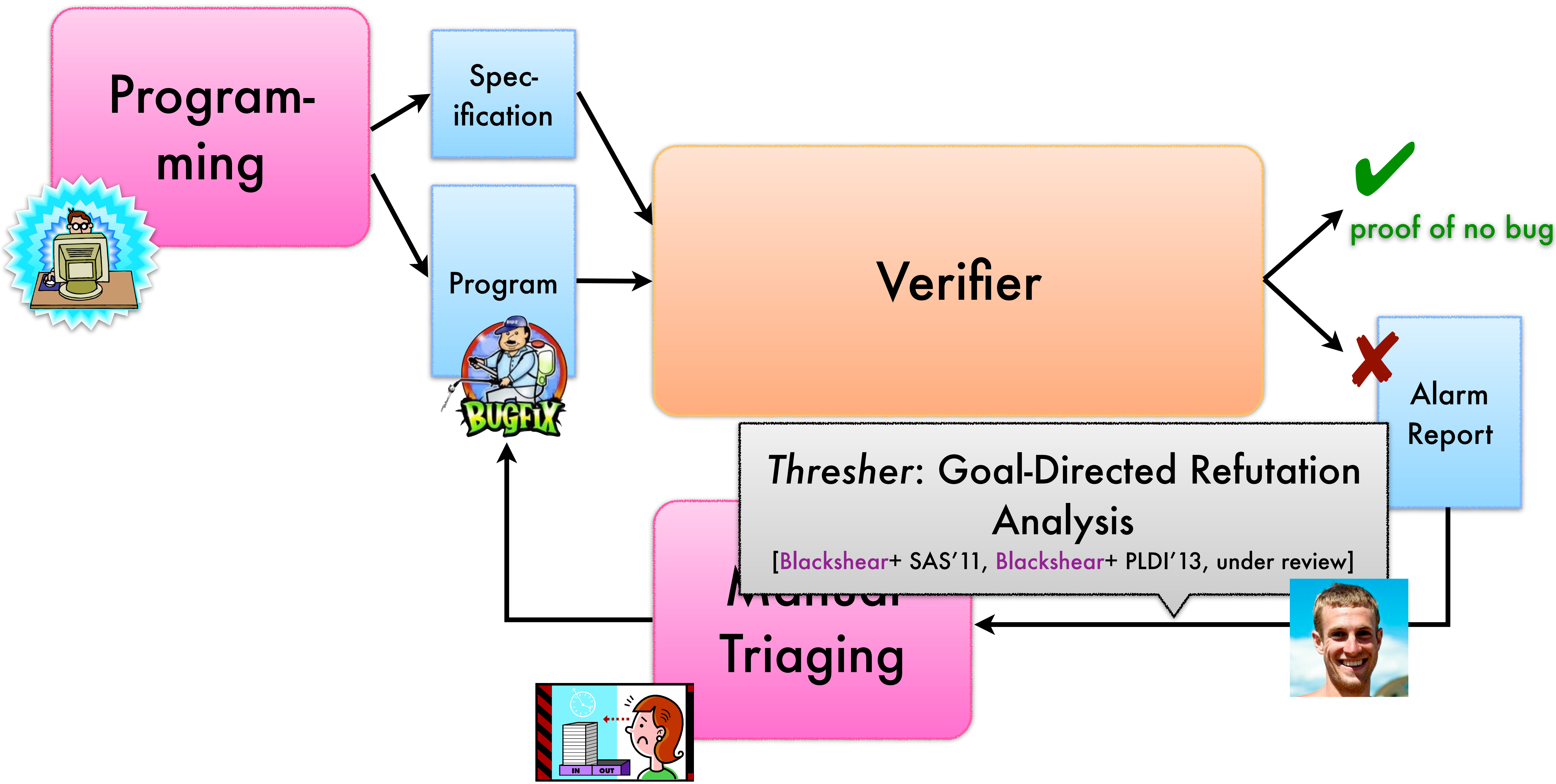
Lab: **Program analysis** in the **whole** bug mitigation **process**



Lab: **Program analysis** in the **whole** bug mitigation **process**



Lab: **Program analysis** in the **whole** bug mitigation **process**



lysis in the **whole** bug mitigation **process**

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14, in prep]

Program-ming

Spec-ification

Program

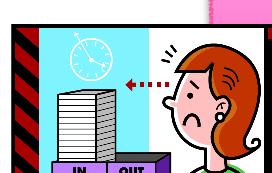
Verifier

proof of no bug

Alarm Report

Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, under review]

Manual Triaging



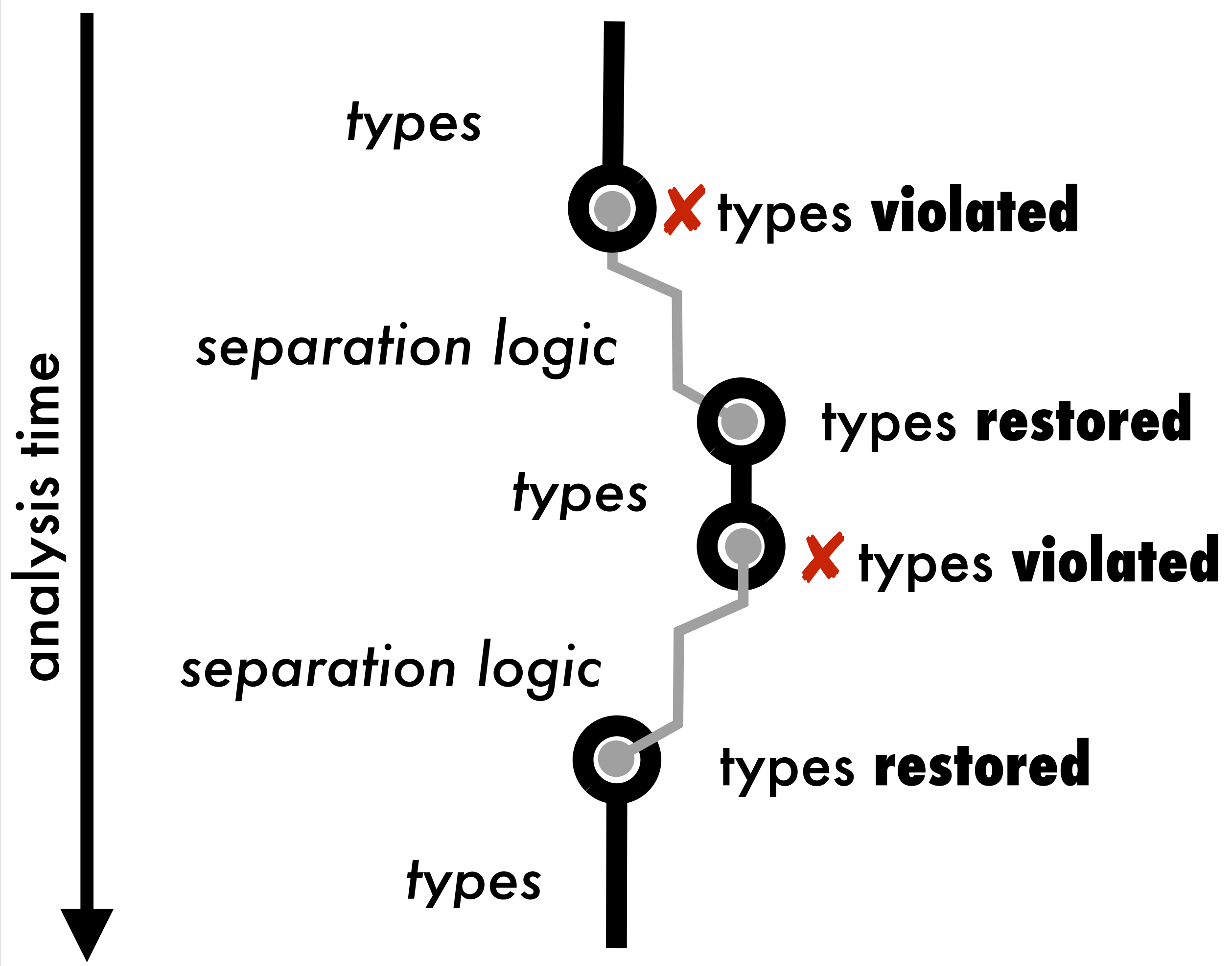
…lysis in the **whole** bug mitigation **process**

Fissile Types:
Checking Almost Everywhere Invariants
[Coughlin+ POPL'14, in prep]

Prog
m

no bug

m
rt

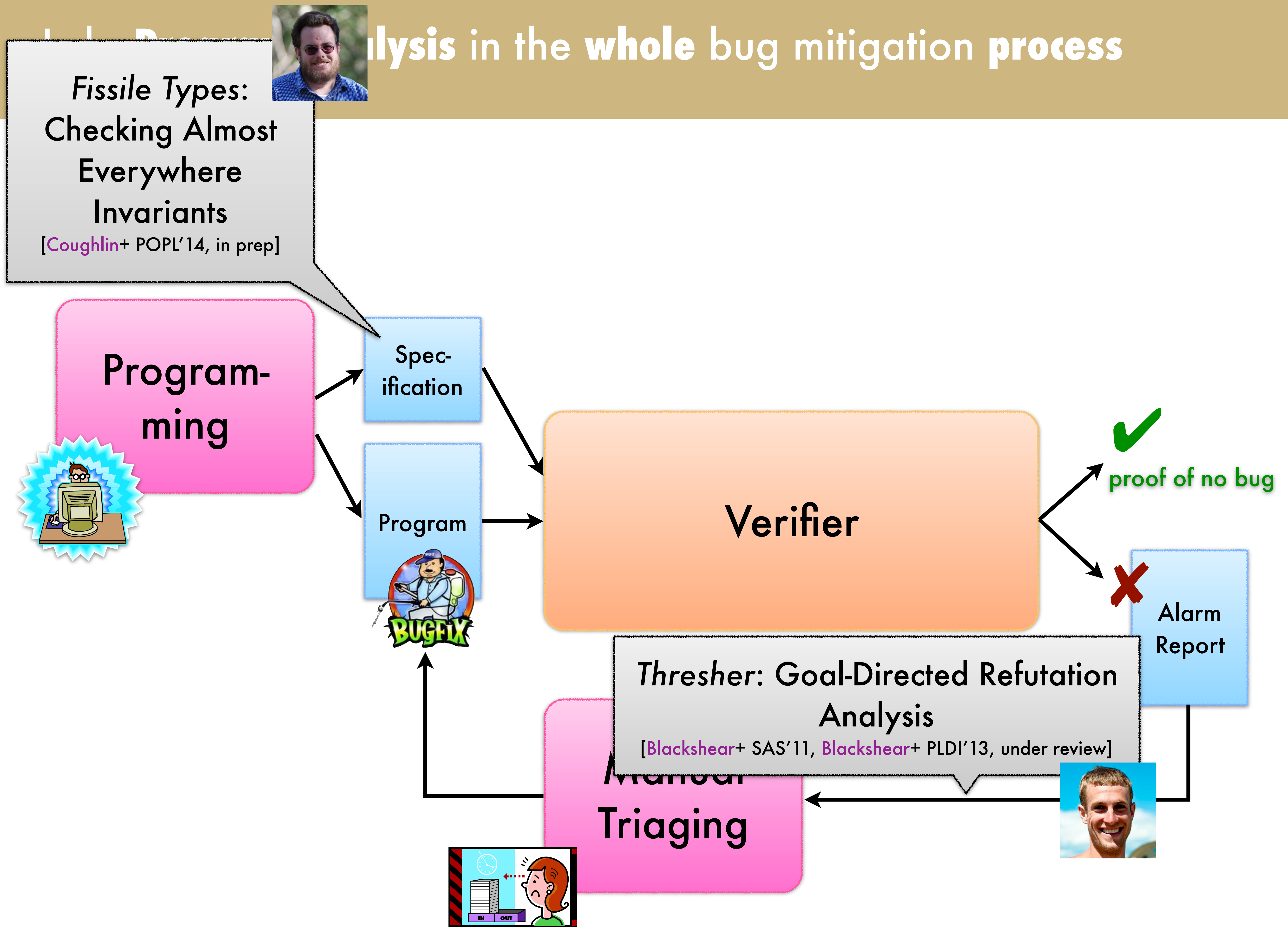


type-intertwined separation logic

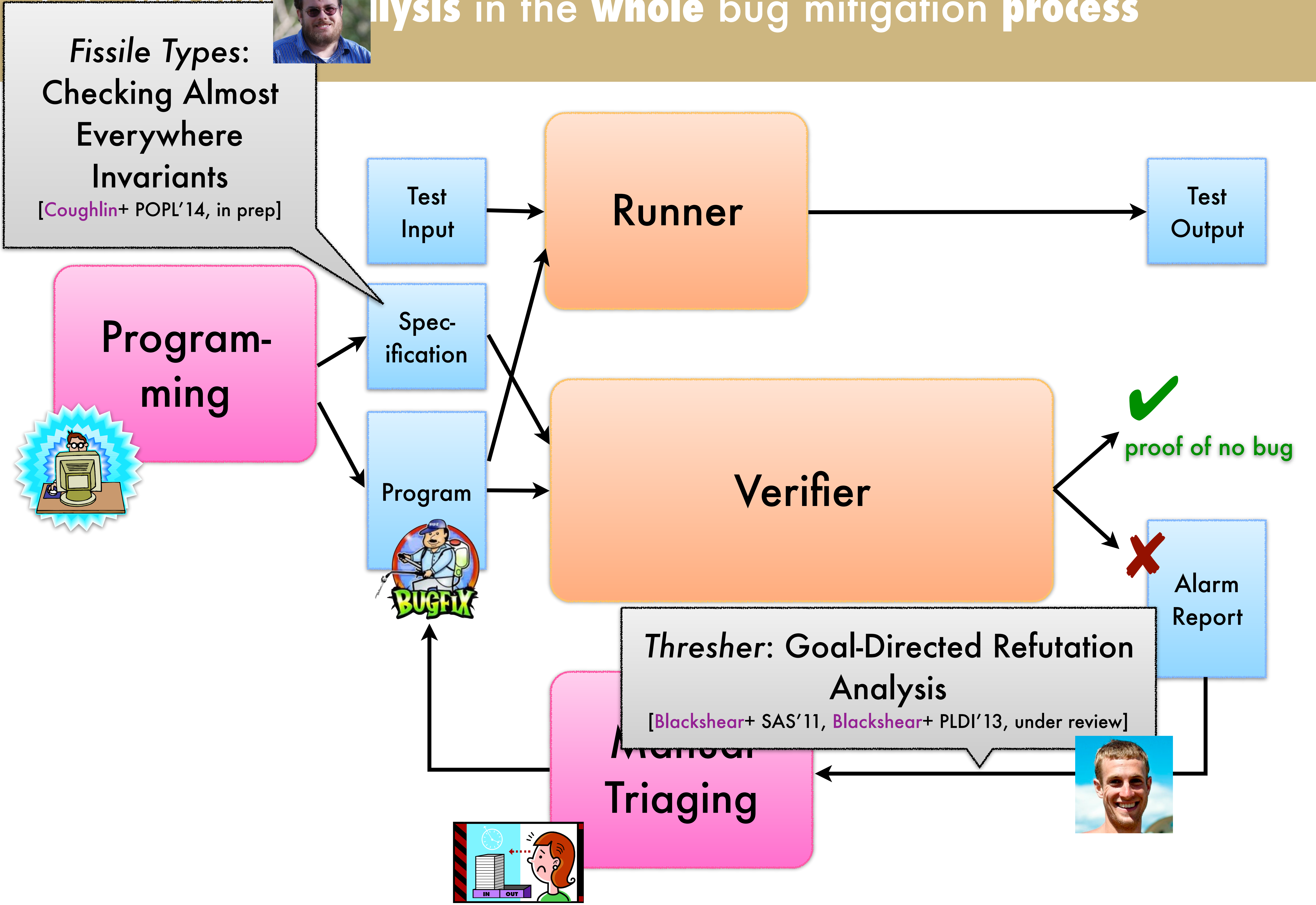
Intertwine type-based and separation-based analysis to verify **almost-everywhere** invariants

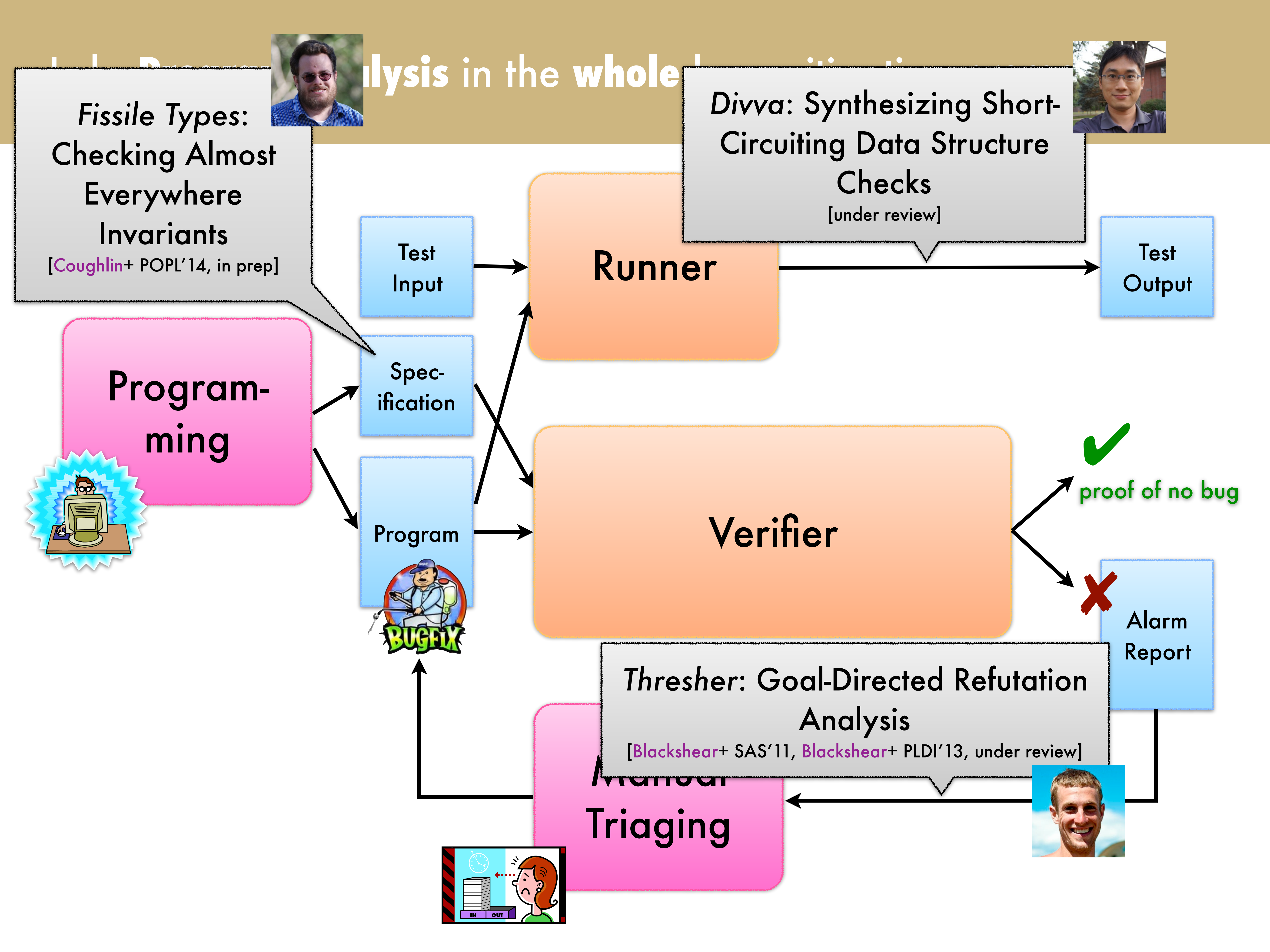
Objective-C

lysis in the **whole** bug mitigation **process**



…lysis in the **whole** bug mitigation **process**




lysis in the whole
Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14, in prep]
Divva: Synthesizing Short-Circuiting Data Structure Checks
[under review]
Test Input
Runner
Test Output
Program-ming
Spec-ification
Program
BUGFIX
Verifier
proof of no bug
Alarm Report
Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, under review]
Triaging
IN
OUT

lysis in the whole

Fissile Types: Checking Almost Everywhere Invariants

[Coughlin+ POPL'14, in prep]

Divva: Synthesizing Short-Circuiting Data Structure Checks

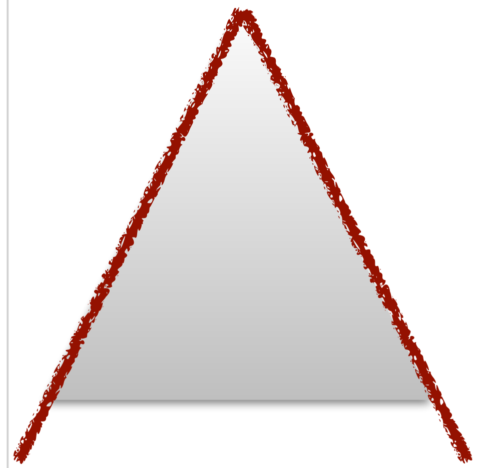
[under review]

Program-ming

Test Input

Spec-ification

Program



Use **static shape analysis** to **synthesize short-circuiting dynamic validation** of data structure invariants

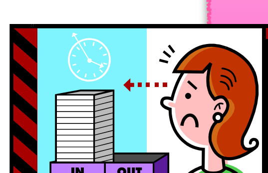
uninterpreted inductive separation logic predicates

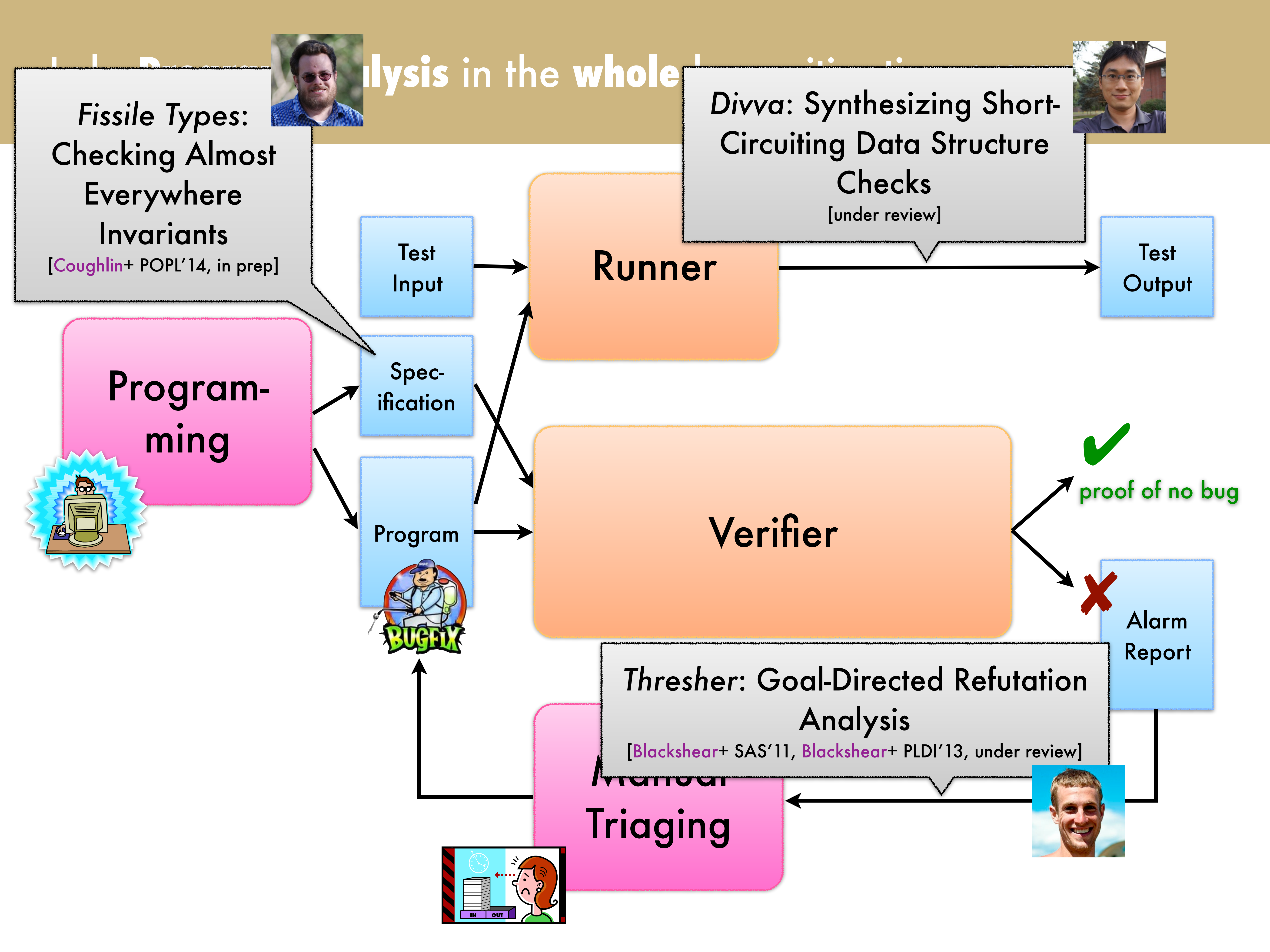
Alarm Report

Thresher: Goal-Directed Refutation Analysis

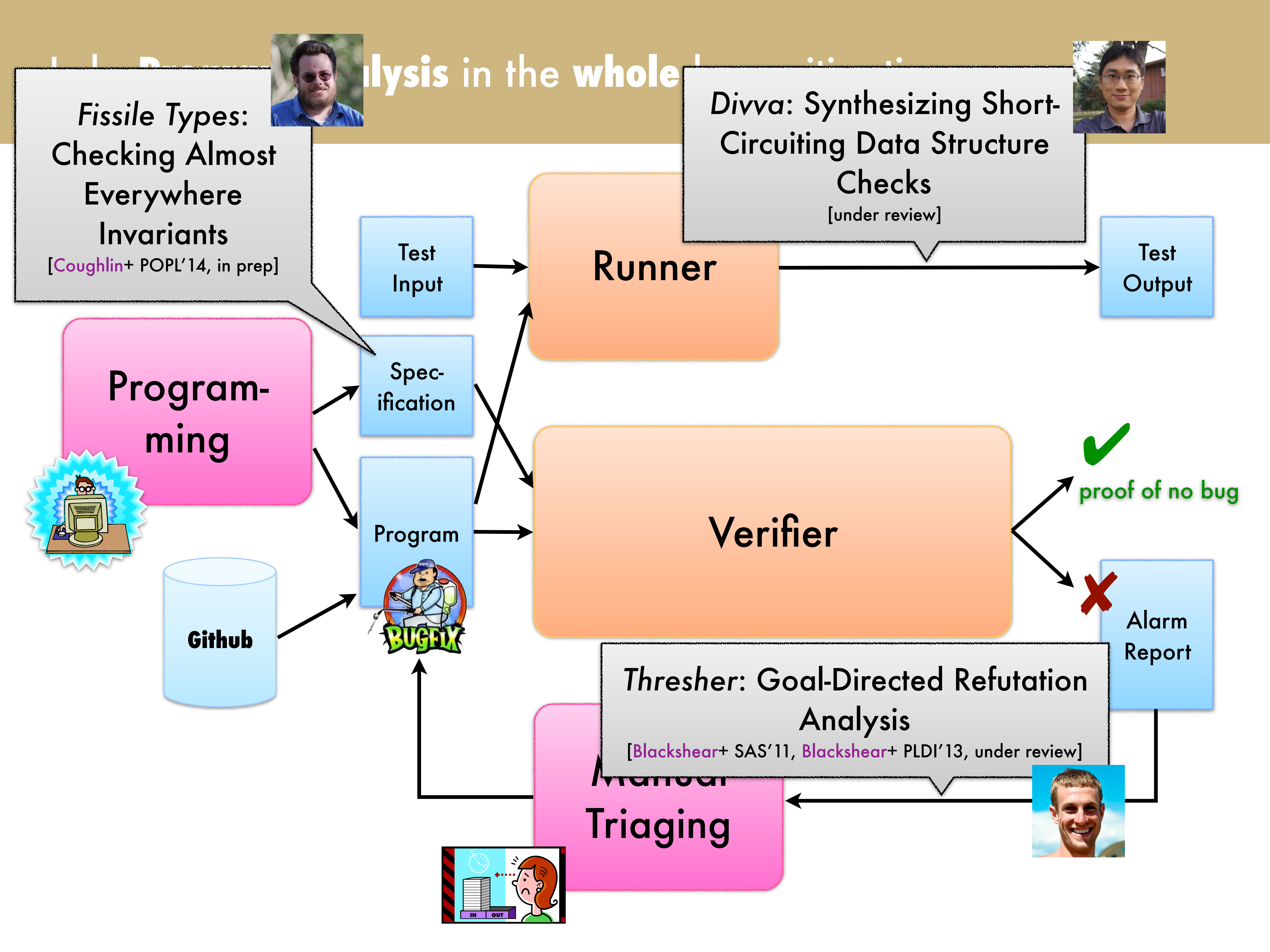
[Blackshear+ SAS'11, Blackshear+ PLDI'13, under review]

Triaging

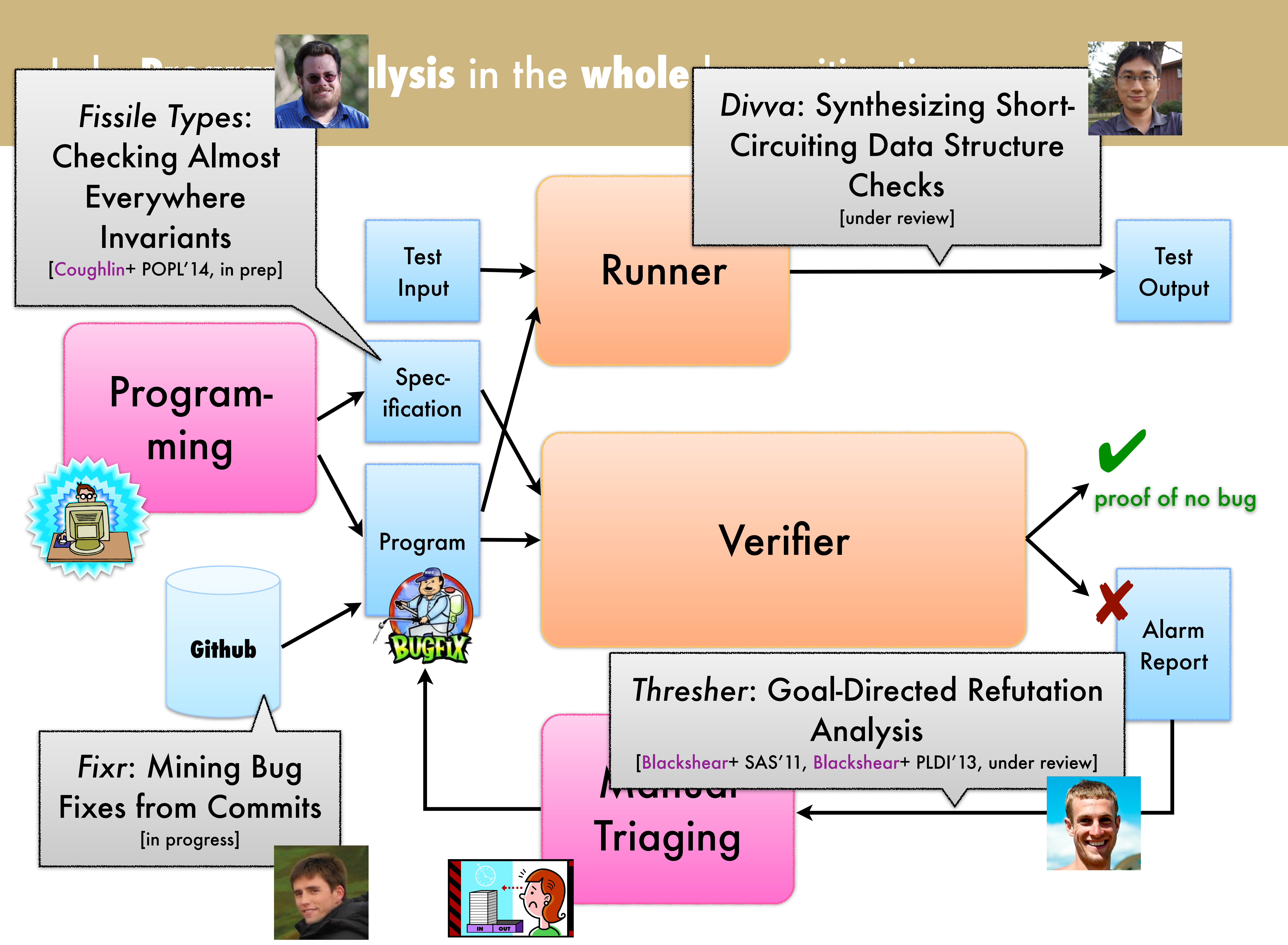


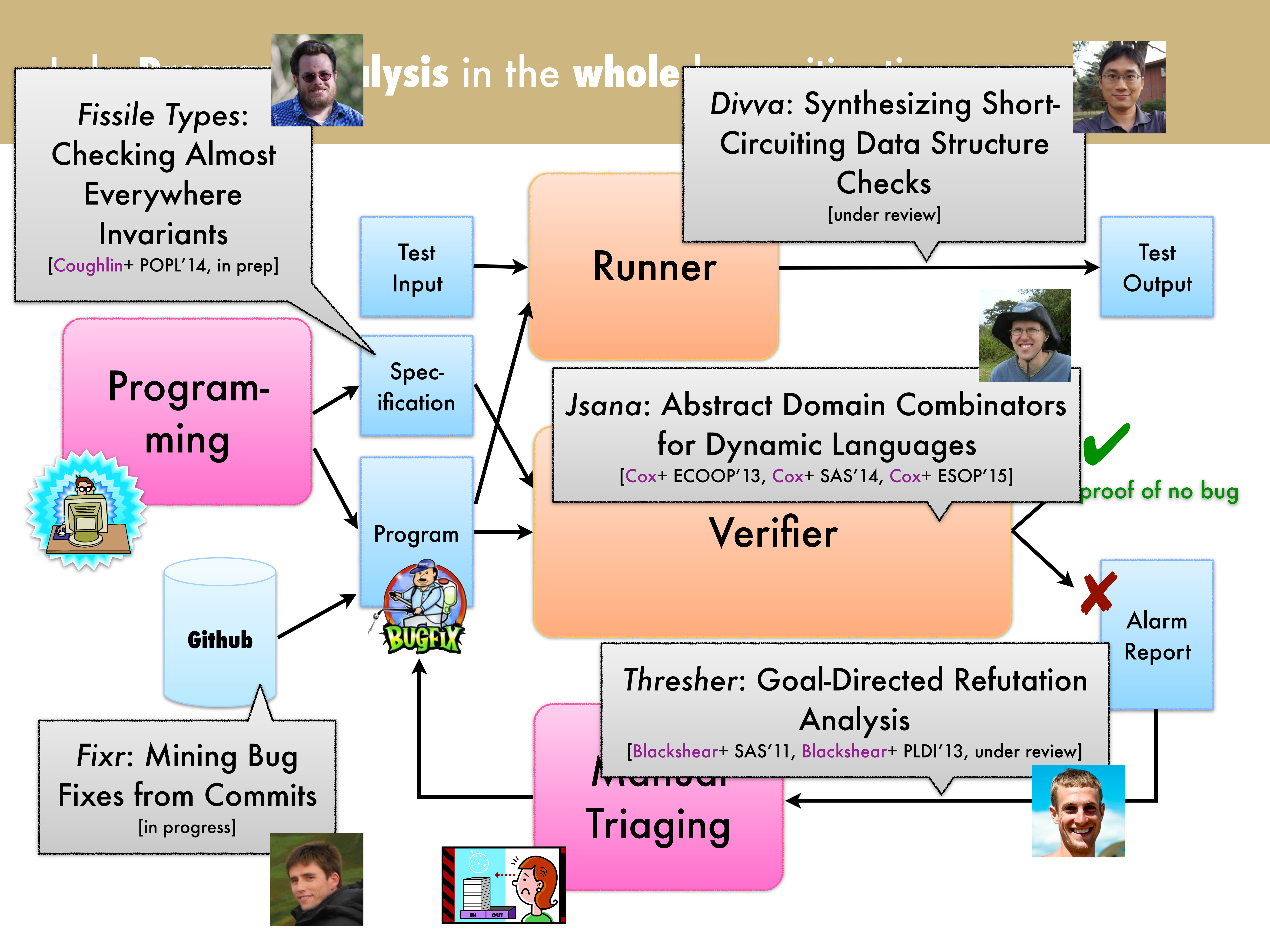


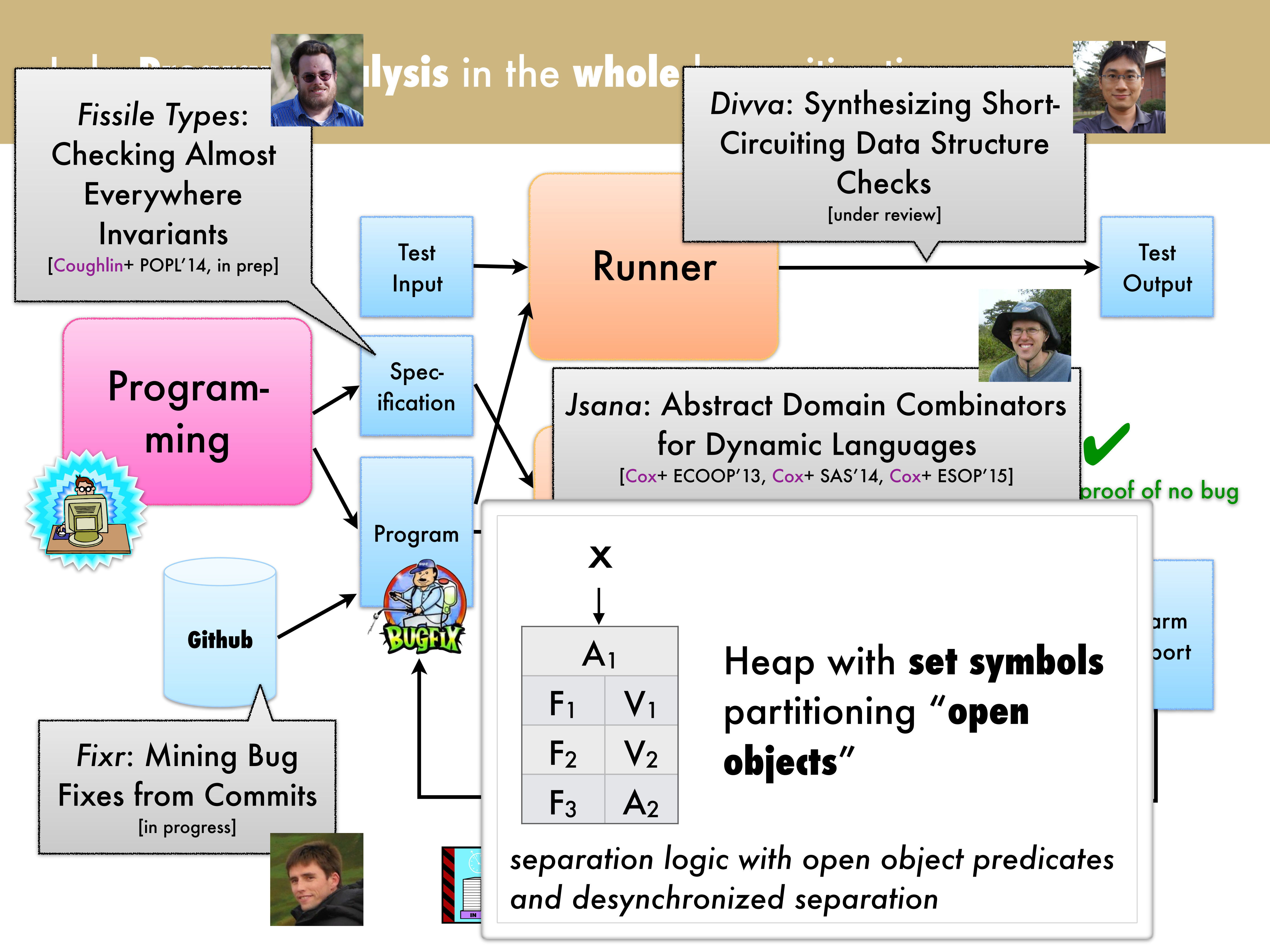
lysis in the whole
Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14, in prep]
Divva: Synthesizing Short-Circuiting Data Structure Checks
[under review]
Test Input
Runner
Test Output
Program-ming
Spec-ification
Program
BUGFIX
Verifier
✔
proof of no bug
✘
Alarm Report
Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, under review]
Triaging
IN
OUT



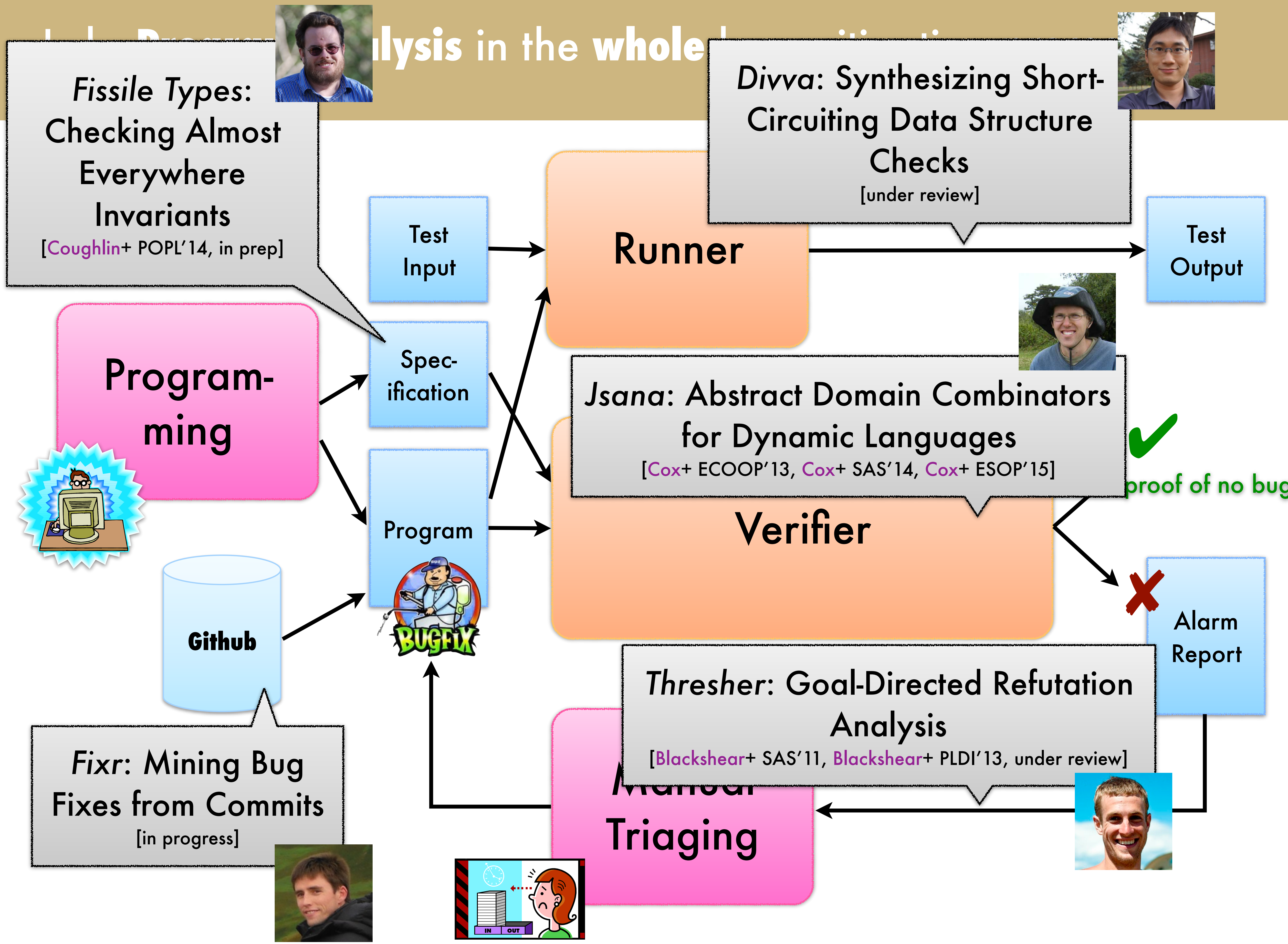
lysis in the whole
Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14, in prep]
Divva: Synthesizing Short-Circuiting Data Structure Checks
[under review]
Test Input
Runner
Test Output
Program-ming
Spec-ification
Program
BUGFIX
Github
Verifier
✔
proof of no bug
✘
Alarm Report
Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, under review]
Triaging
IN
OUT


lysis in the whole
Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14, in prep]
Divva: Synthesizing Short-Circuiting Data Structure Checks
[under review]
Test Input
Runner
Test Output
Program-ming
Spec-ification
Program
Verifier
proof of no bug
Alarm Report
Github
BUGFIX
Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, under review]
Triaging
Fixr: Mining Bug Fixes from Commits
[in progress]
IN
OUT

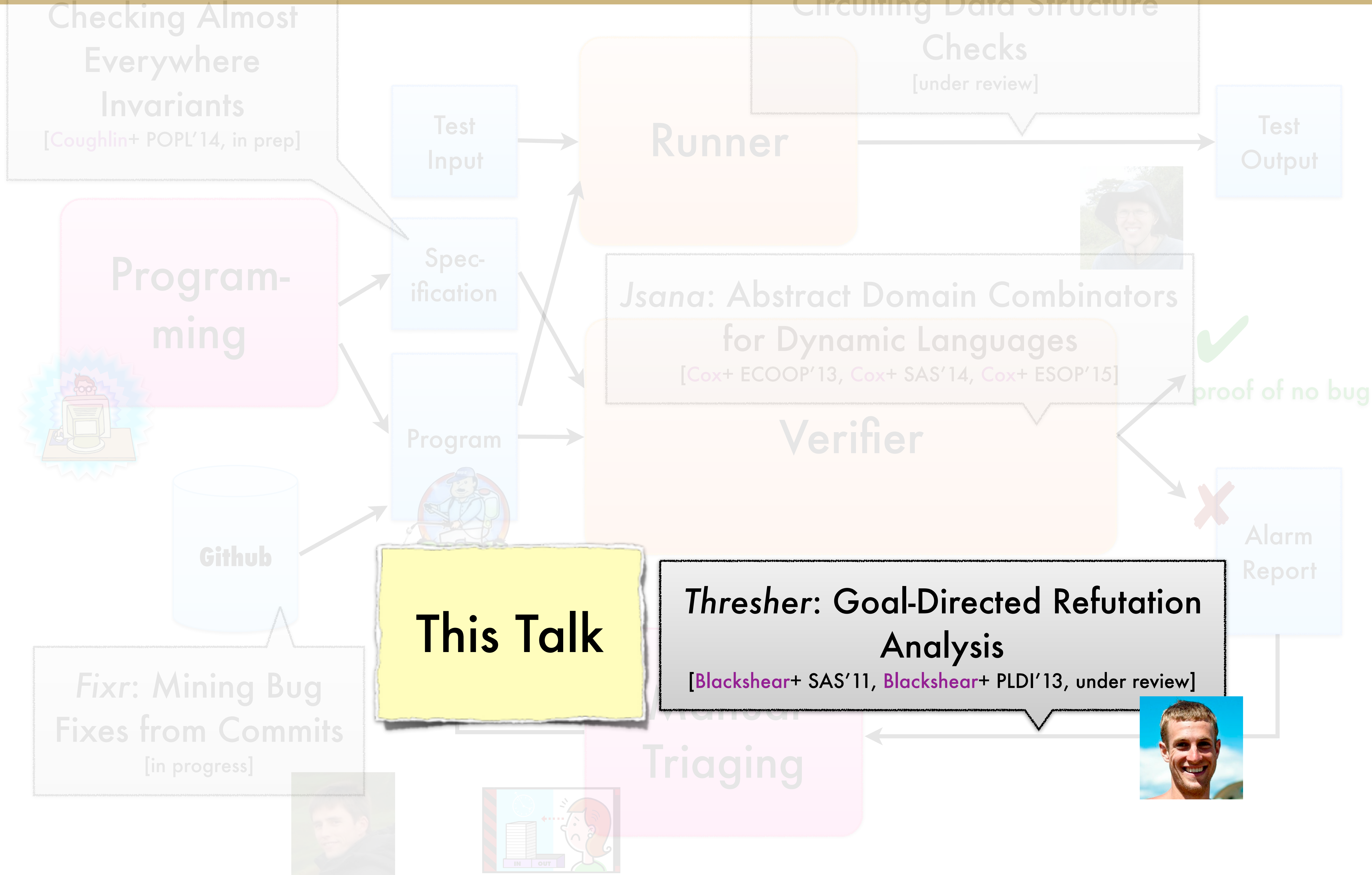

lysis in the whole
Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14, in prep]
Divva: Synthesizing Short-Circuiting Data Structure Checks
[under review]
Test Input
Runner
Test Output
Program-ming
Spec-ification
Program
Jsana: Abstract Domain Combinators for Dynamic Languages
[Cox+ ECOOP'13, Cox+ SAS'14, Cox+ ESOP'15]
proof of no bug
Verifier
Alarm Report
Github
BUGFIX
Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS'11, Blackshear+ PLDI'13, under review]
Triaging
Fixr: Mining Bug Fixes from Commits
[in progress]
IN
OUT


in the whole
Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL'14, in prep]
Divva: Synthesizing Short-Circuiting Data Structure Checks
[under review]
Test Input
Runner
Test Output
Program-ming
Spec-ification
Program
BUGFIX
Jsana: Abstract Domain Combinators for Dynamic Languages
[Cox+ ECOOP'13, Cox+ SAS'14, Cox+ ESOP'15]
proof of no bug
Github
Fixr: Mining Bug Fixes from Commits
[in progress]
x
A1
F1 V1
F2 V2
F3 A2
Heap with set symbols partitioning "open objects"
separation logic with open object predicates and desynchronized separation

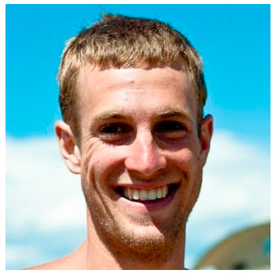
lysis in the whole



Lab: **Program analysis** in the **whole** bug mitigation **process**



Goal-Directed Program Analysis with Jumping



Sam Blackshear
University of Colorado Boulder

Bor-Yuh Evan Chang
University of Colorado Boulder

Manu Sridharan
Samsung Research America

École normale supérieure
1 October 2015

PLV


Crash

Roughly, 3% of all commits fix NullPointerExceptions

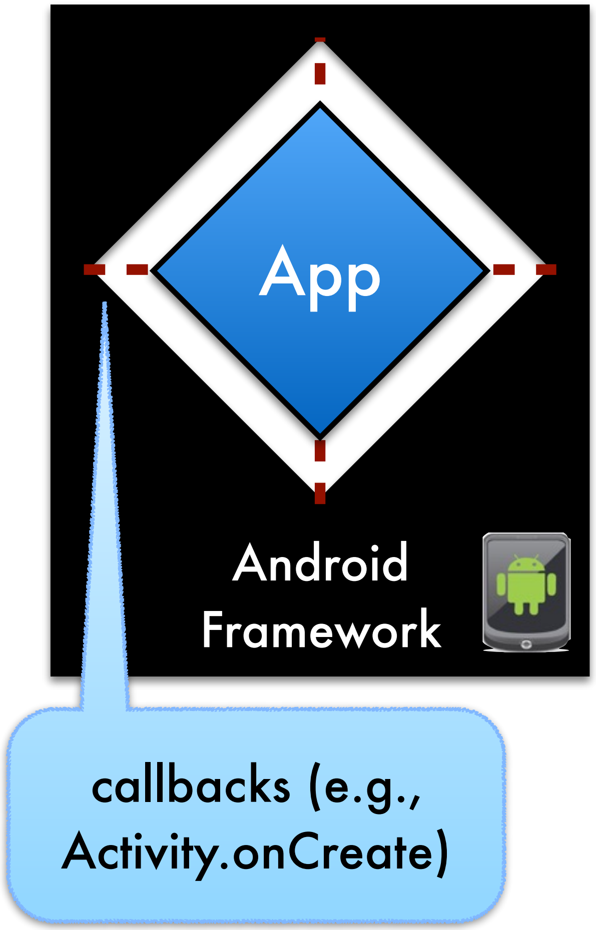
Callback-oriented programming

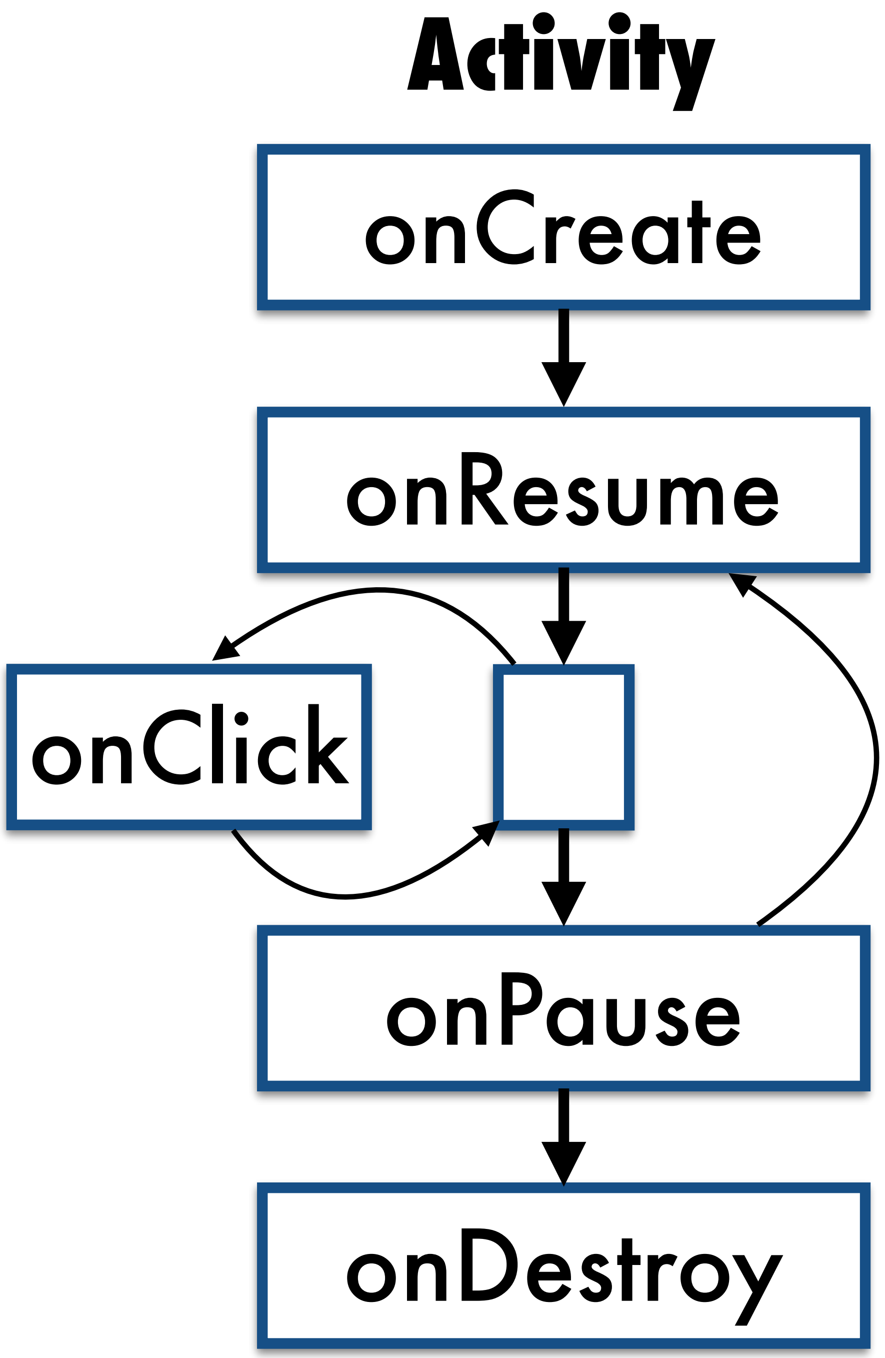
Callback-oriented programming

Callback-oriented programming

callbacks (e.g.,
Activity.onCreate)

Callback-oriented programming

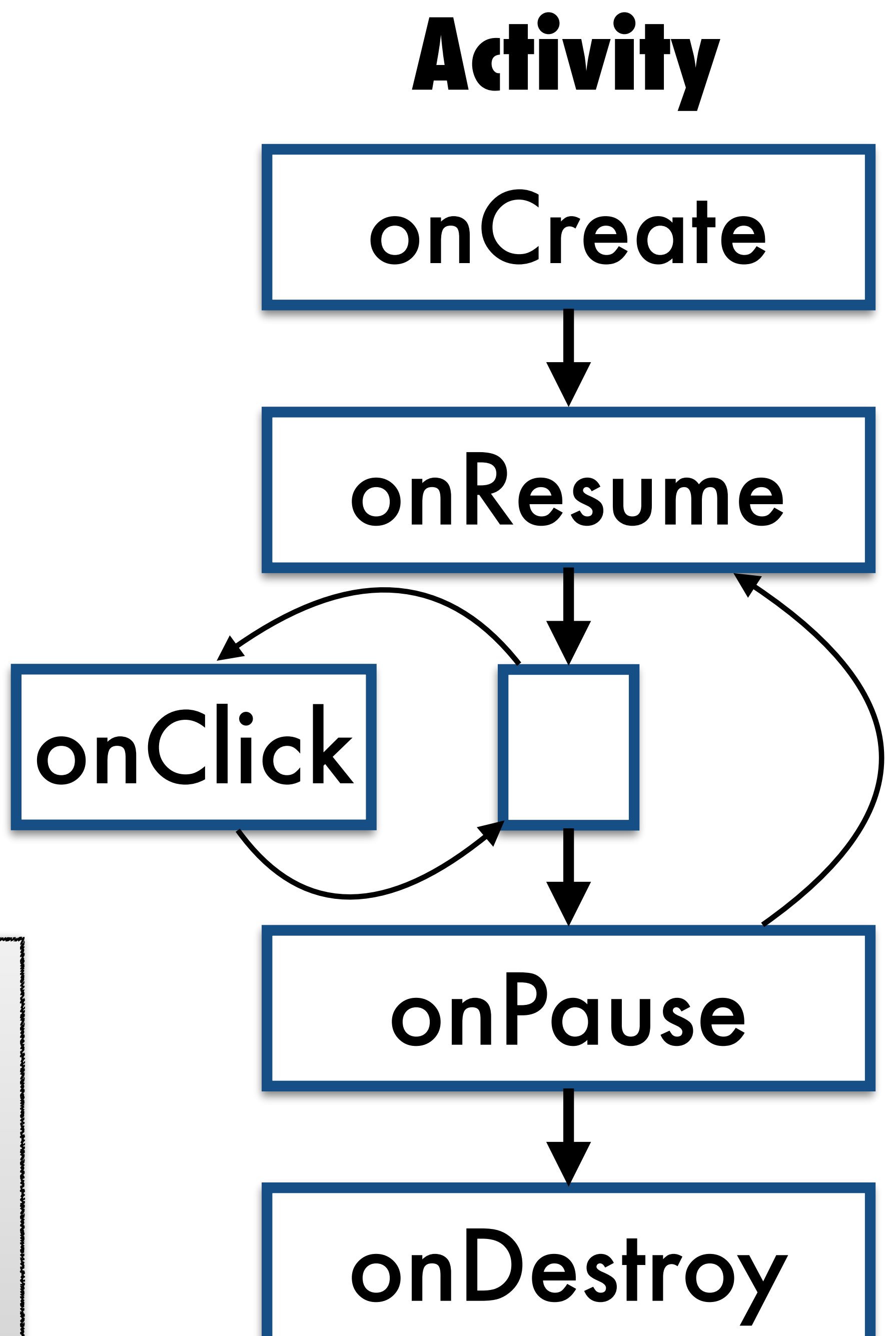




Callback-oriented programming

callbacks (e.g., Activity.onCreate)

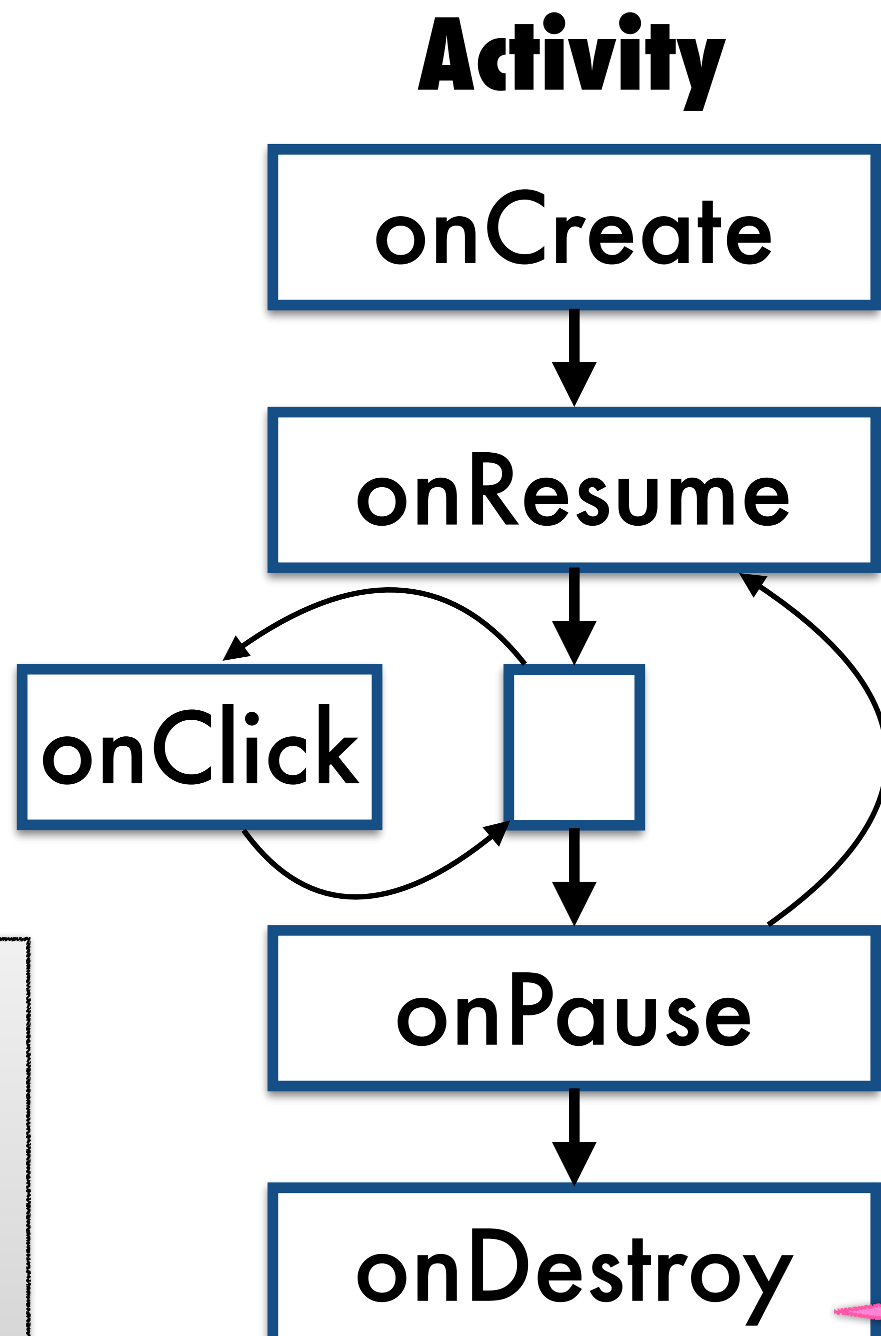
Android components have an ordered, event-driven lifecycle



Callback-oriented programming

callbacks (e.g., Activity.onCreate)

Android components have an ordered, event-driven lifecycle

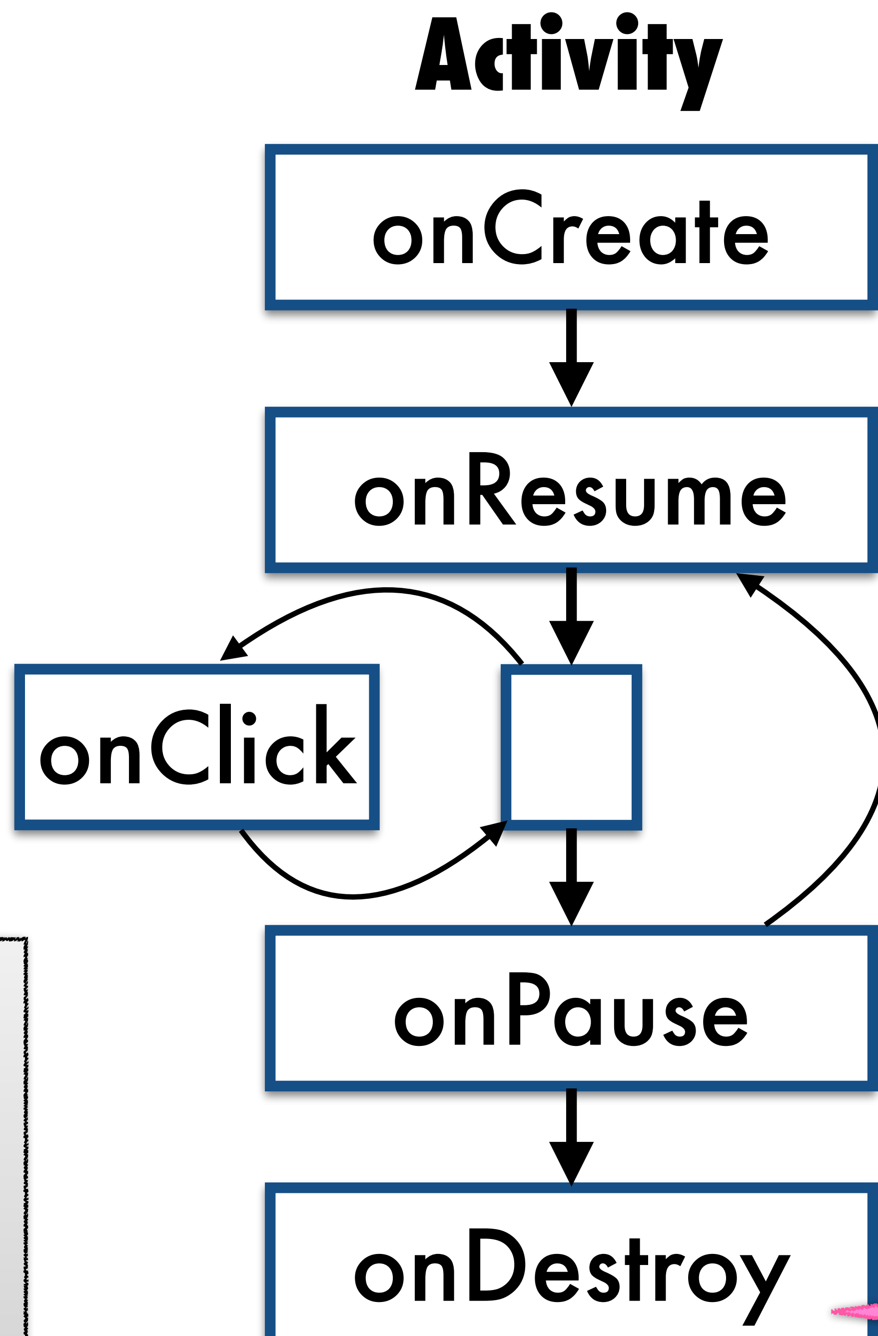



```
mHostDb = null;
mService = null;
```

Callback-oriented programming

callbacks (e.g., Activity.onCreate)

Android components have an ordered, event-driven lifecycle




```
mHostDb = null;
mService = null;
```

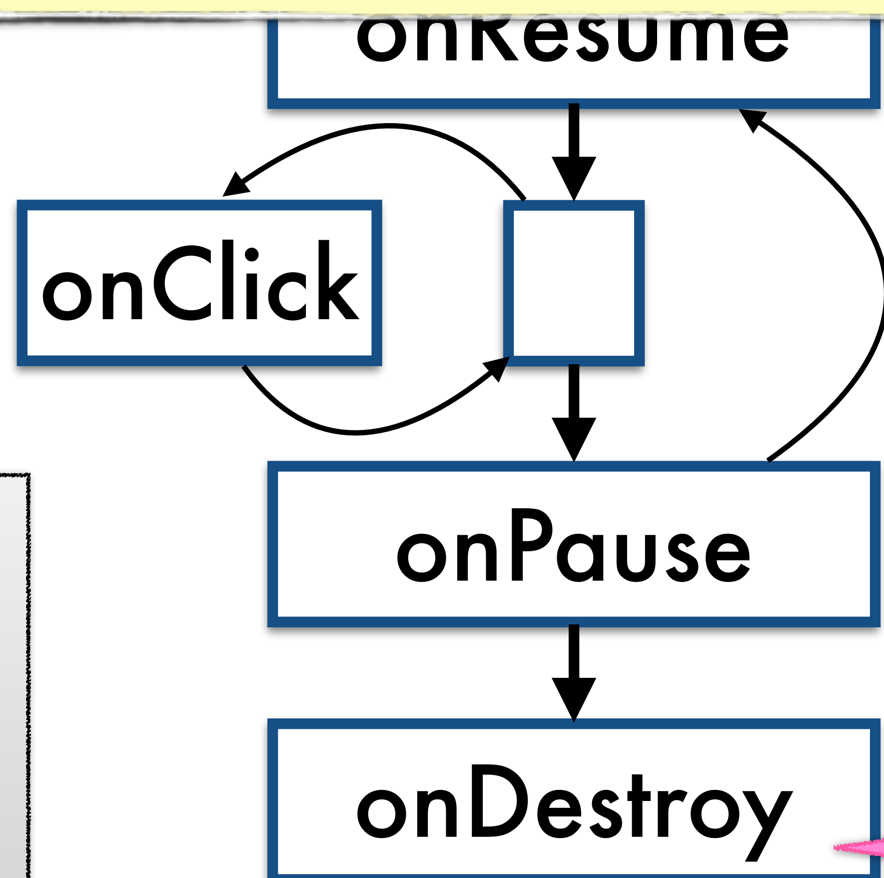
But, lifecycles of different components and other callbacks can interleave …

Callback-oriented programming

Challenge: Verifying safety (of dereferences) depends on callback interleaving

Android Framework

callbacks (e.g., Activity.onCreate)

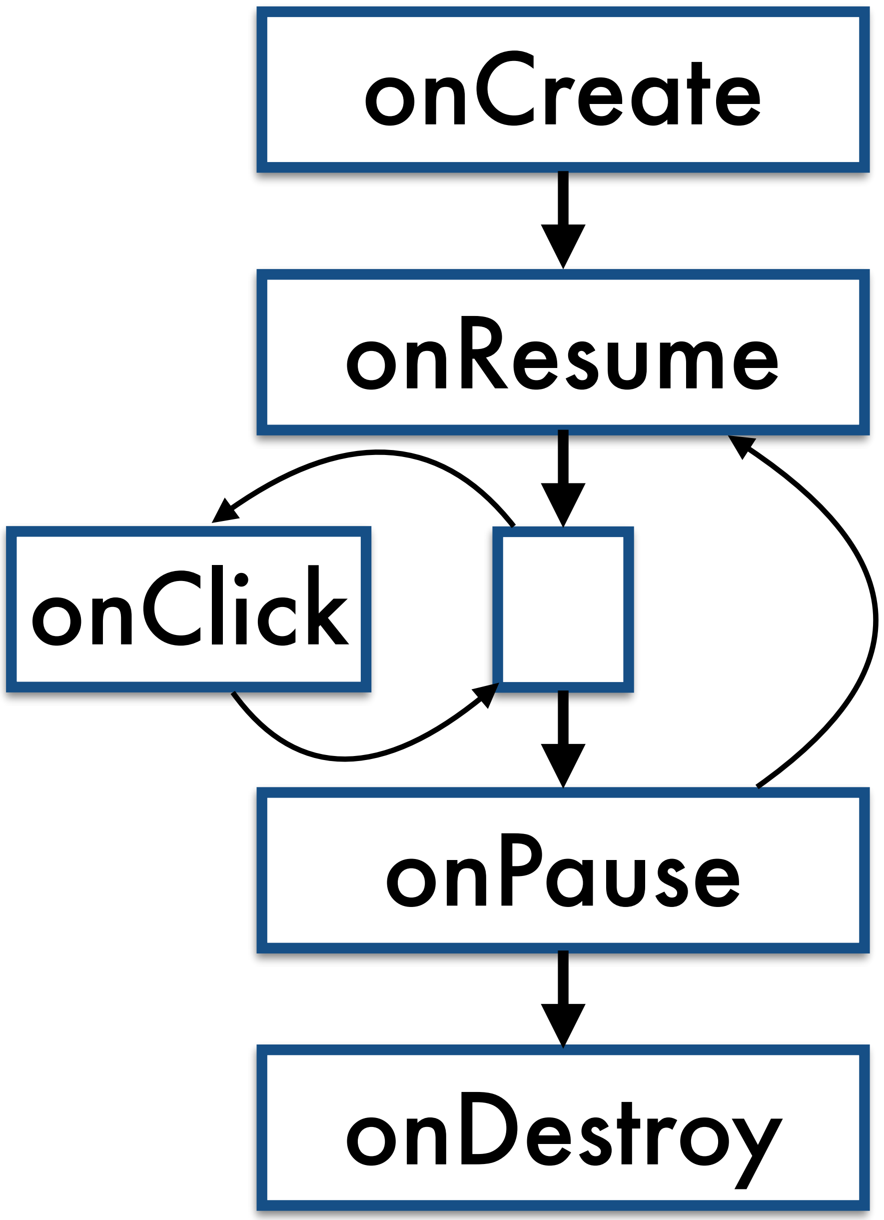


Android components have an ordered, event-driven lifecycle

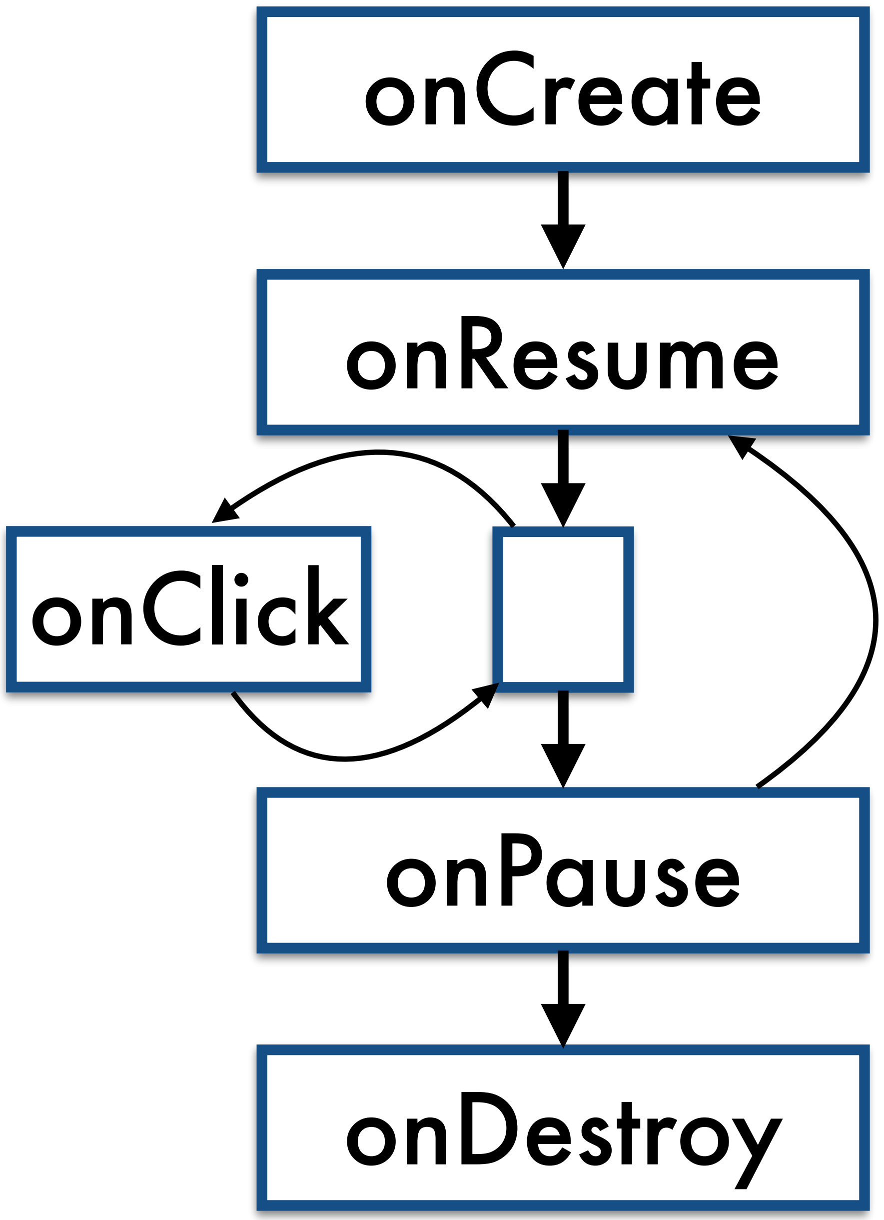
But, lifecycles of different components and other callbacks can interleave …

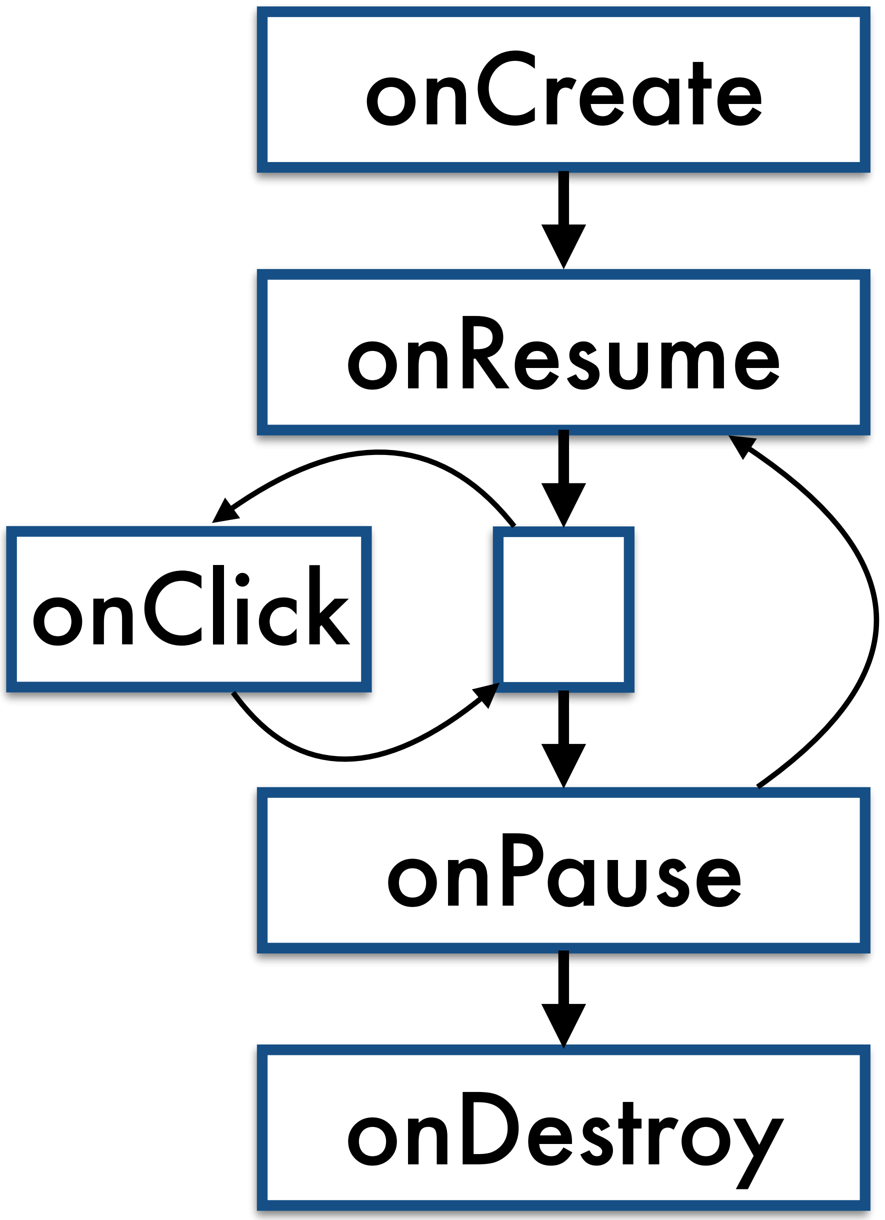
```
mHostDb = null;
mService = null;
```

Callback-oriented programming

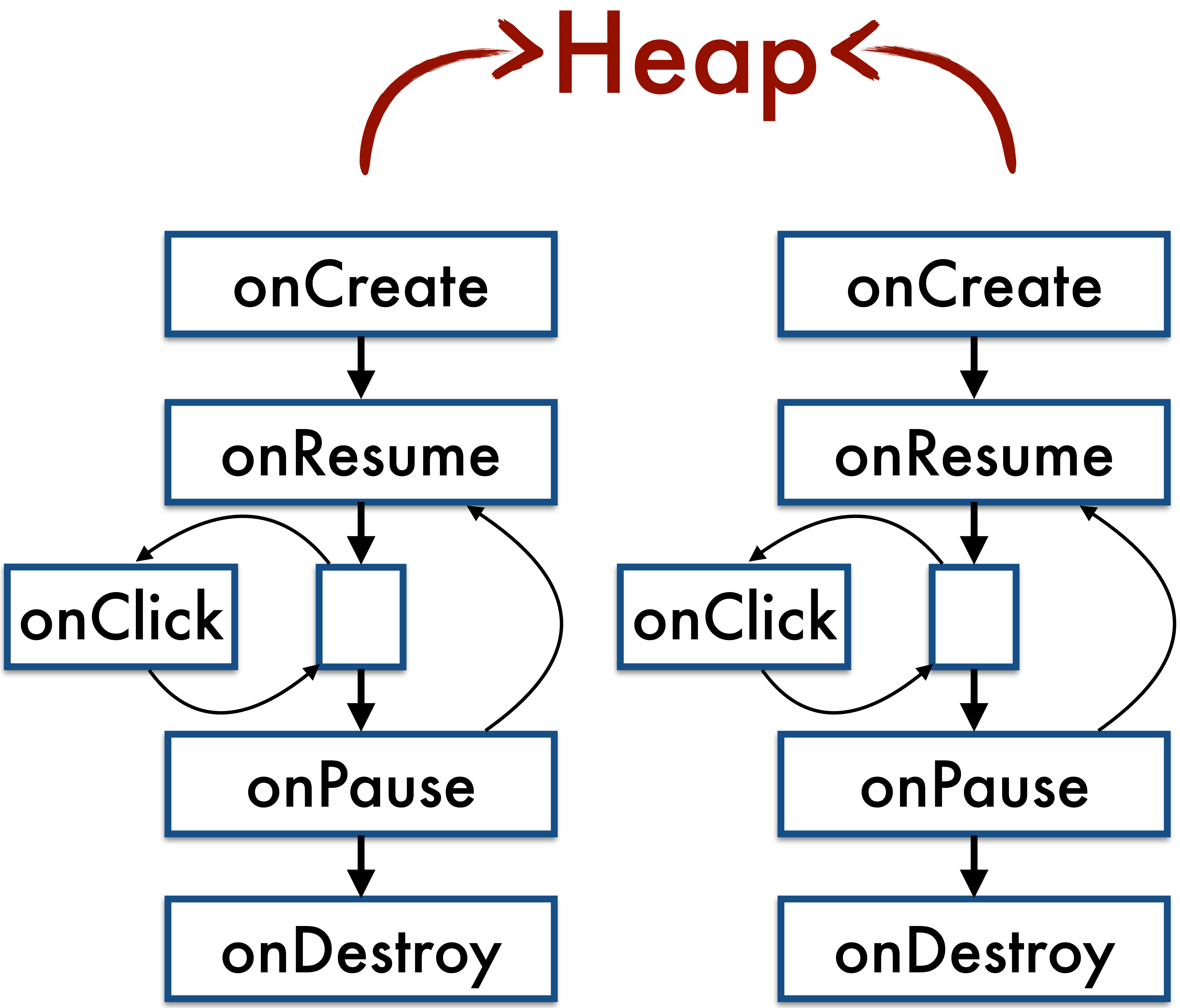


Callback-oriented programming

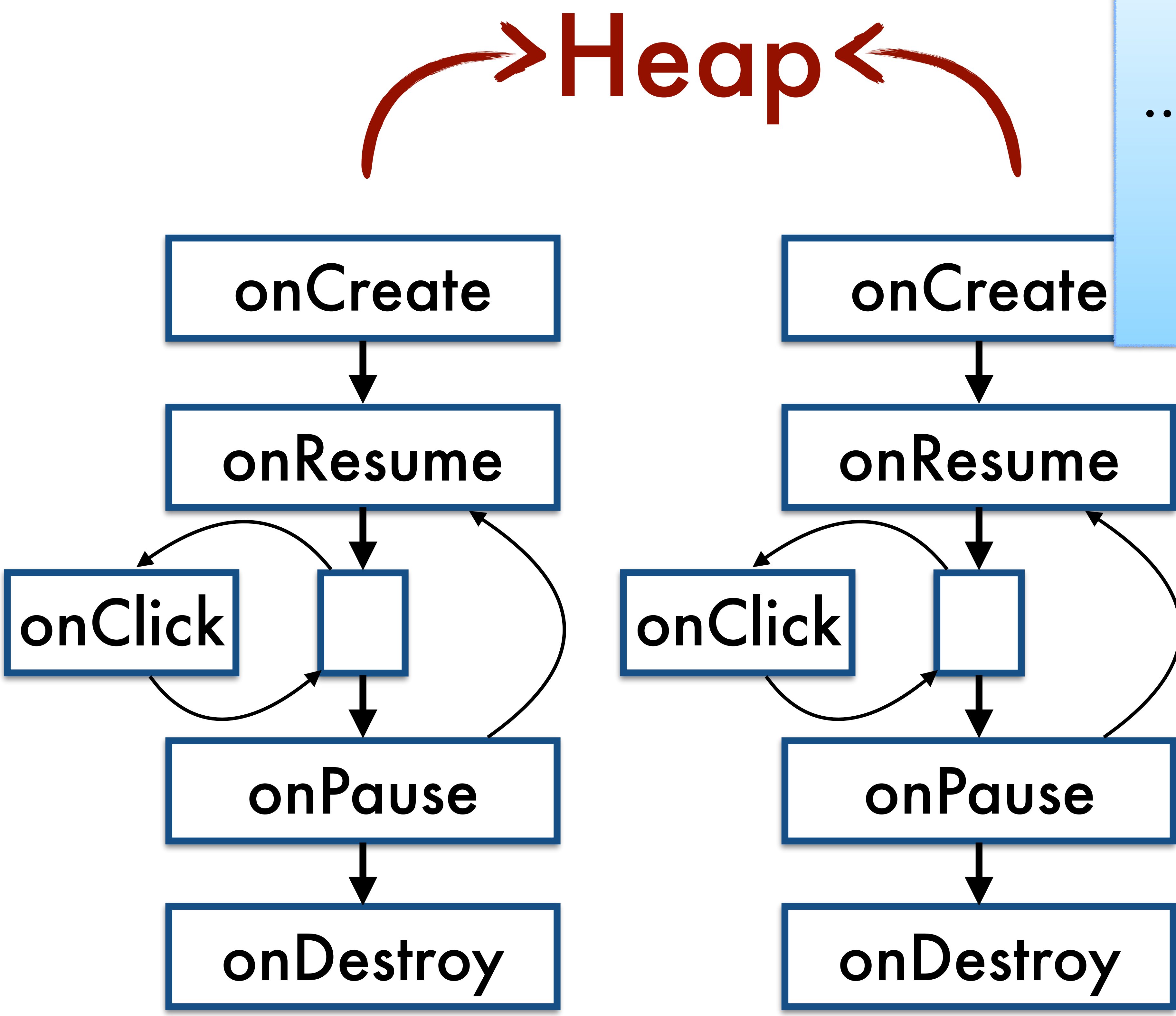




Callback-oriented programming

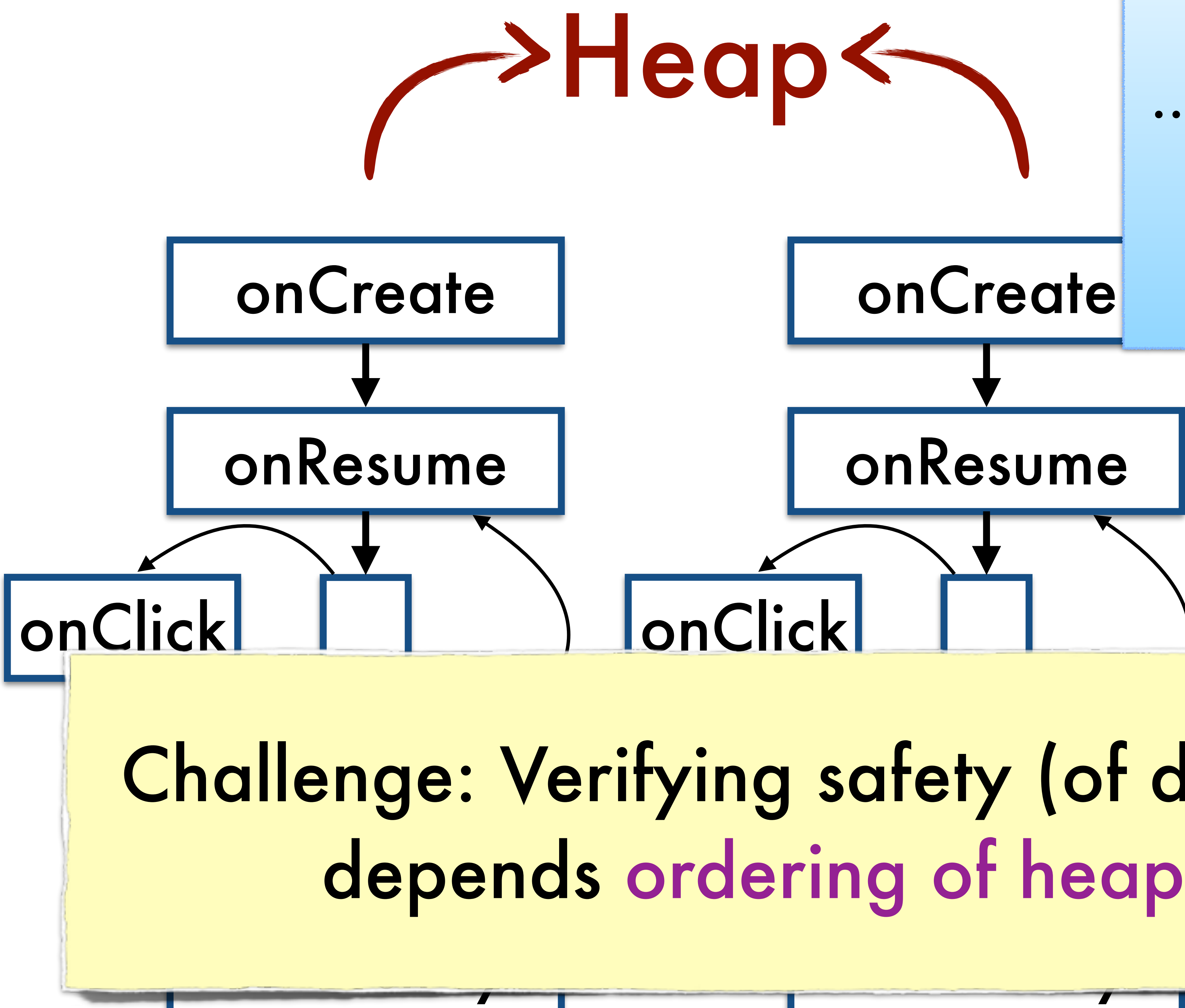


Callback-oriented programming



… and operates over a shared, global heap

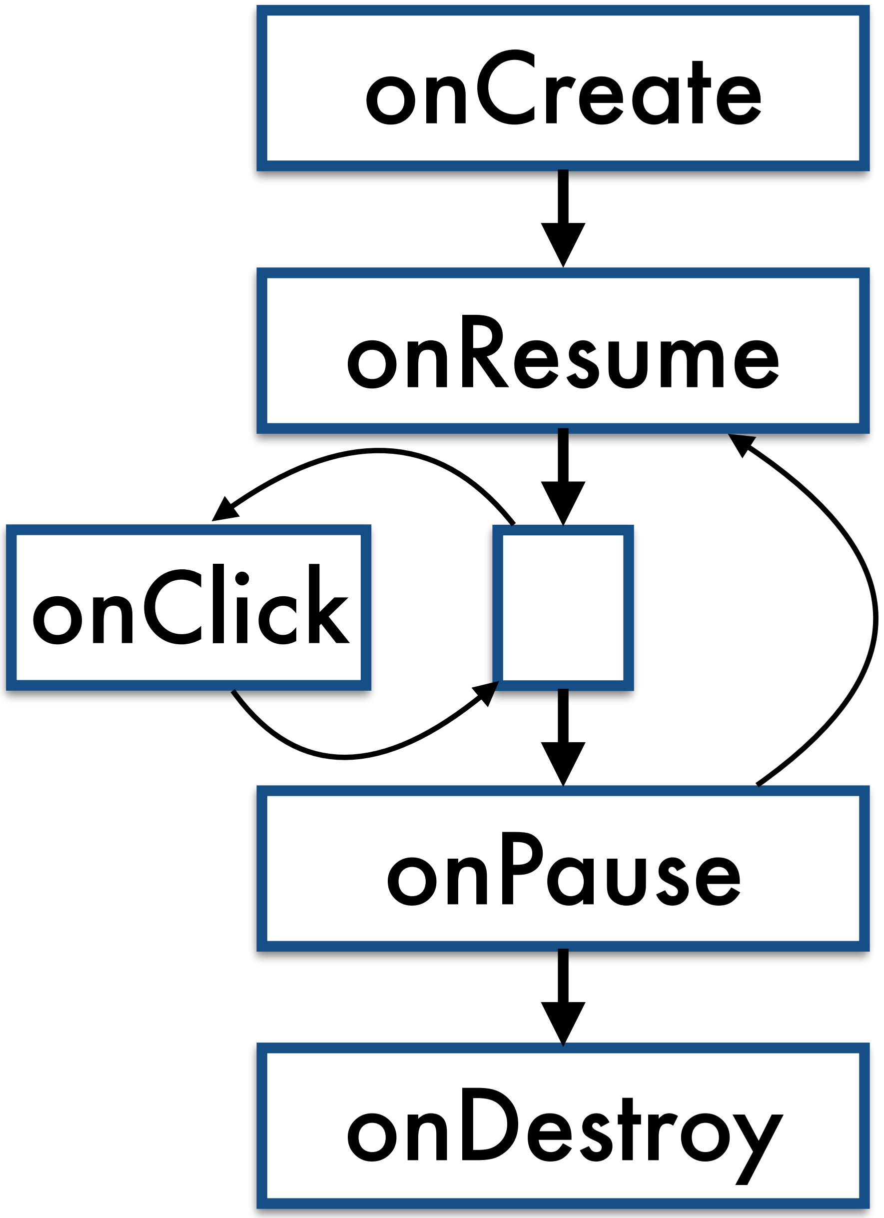
Callback-oriented programming



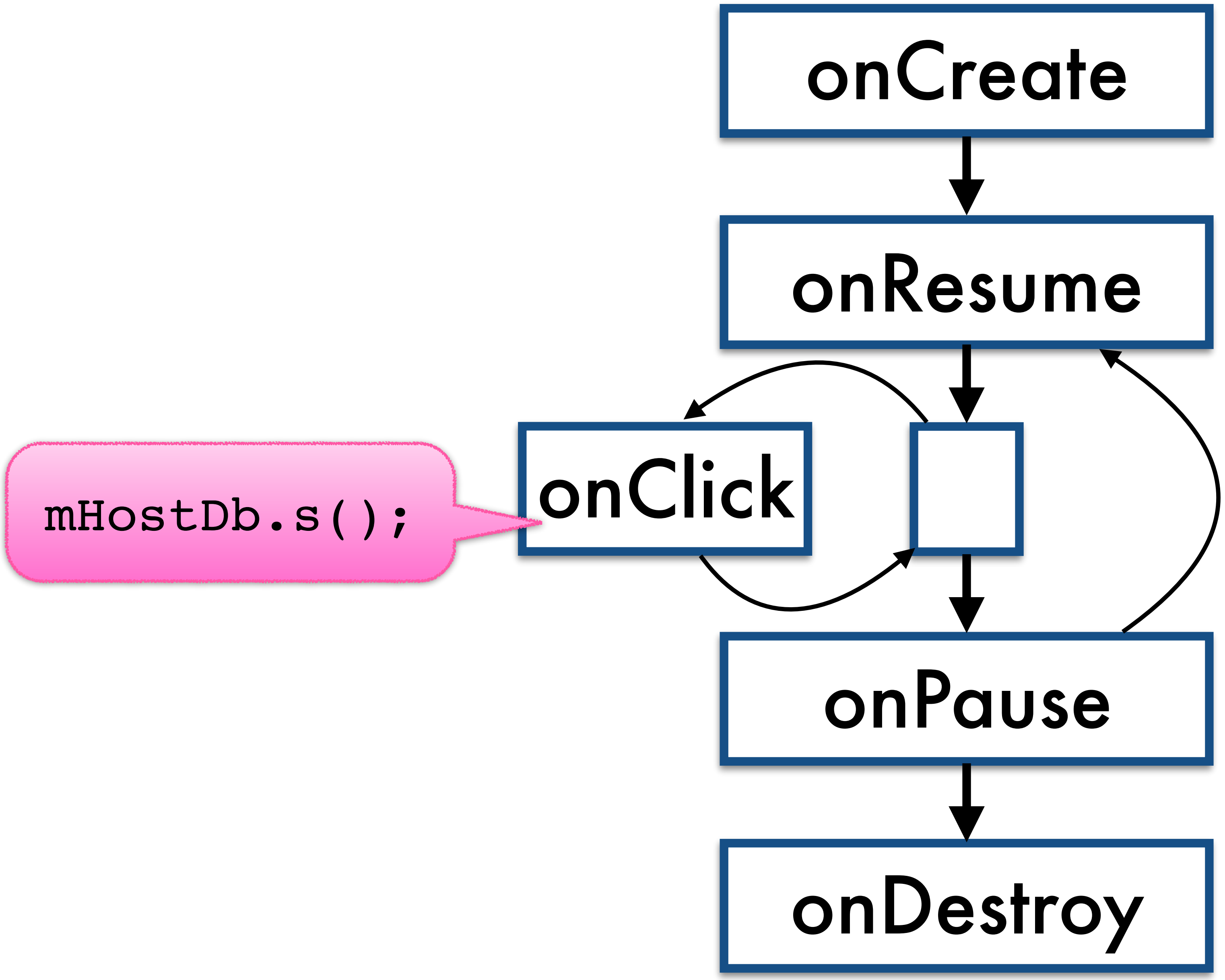
… and operates over a shared, global heap

Challenge: Verifying safety (of dereferences) depends ordering of heap writes

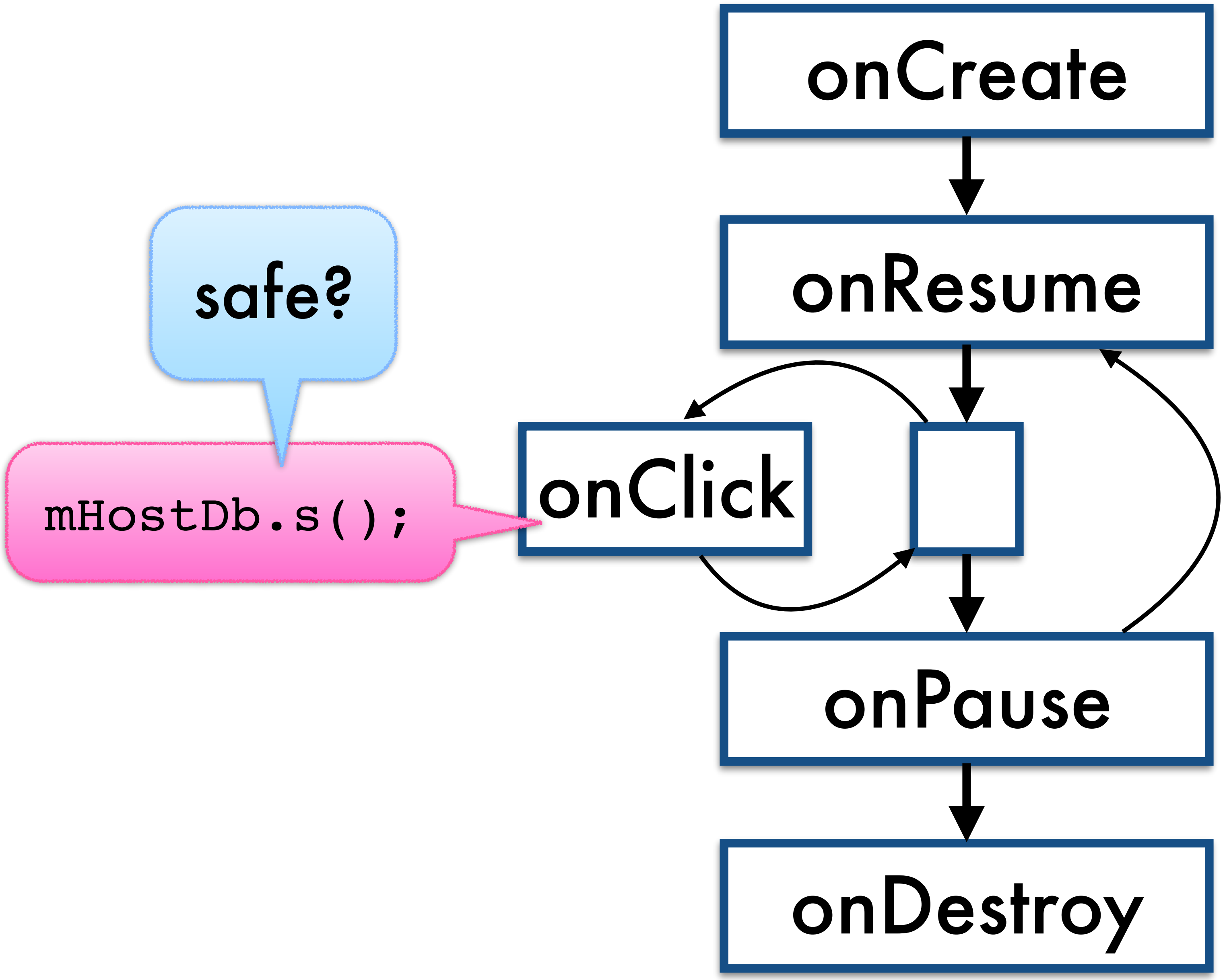
But it shouldn't be so hard ...



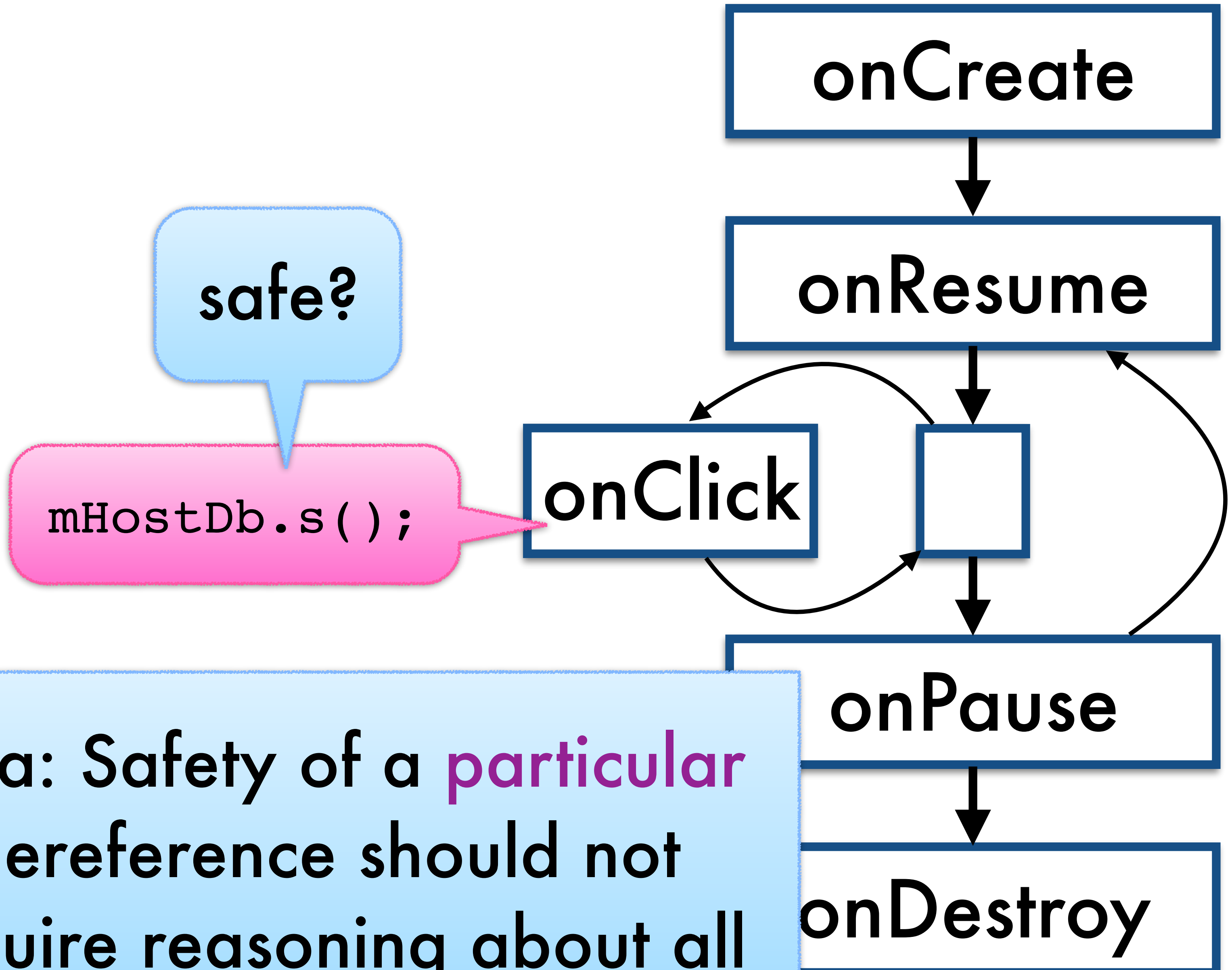
But it shouldn't be so hard ...



But it shouldn't be so hard ...

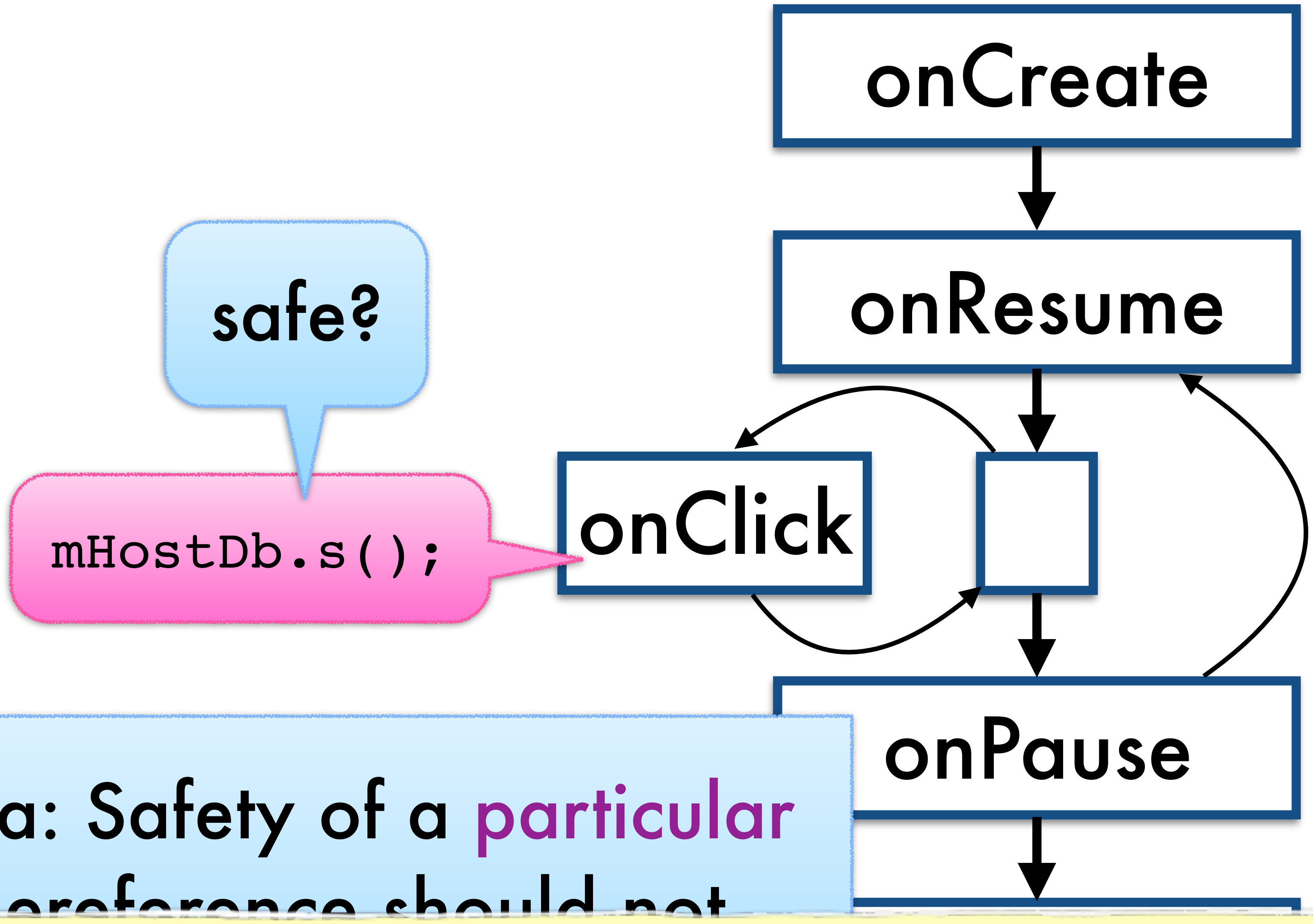


But it shouldn't be so hard …



Idea: Safety of a particular dereference should not require reasoning about all callback interleavings

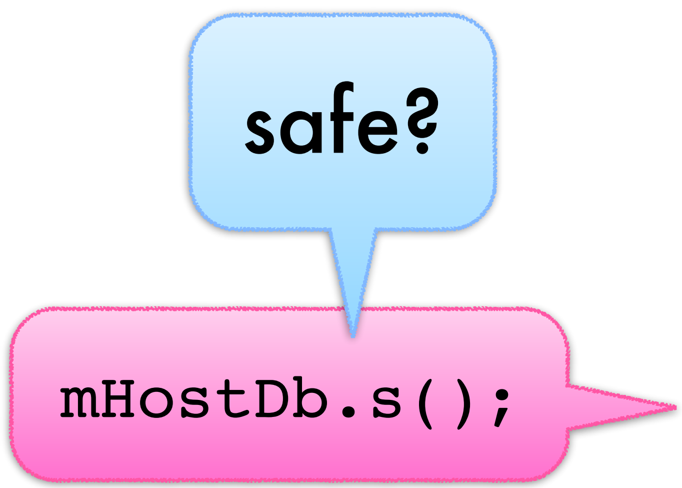
But it shouldn’t be so hard …



Idea: Safety of a particular

A “smart” goal-directed analysis could consider relevant callback orderings without considering all of them

Goal-directed program analysis



Goal-directed program analysis

Given a program configuration **goal**, derive a **contradiction** w.r.t. its reachability

safe?

```
mHostDb.s();
```

Goal-directed program analysis

Given a program configuration **goal**, derive a **contradiction** w.r.t. its reachability

safe?

`mHostDb.s();`

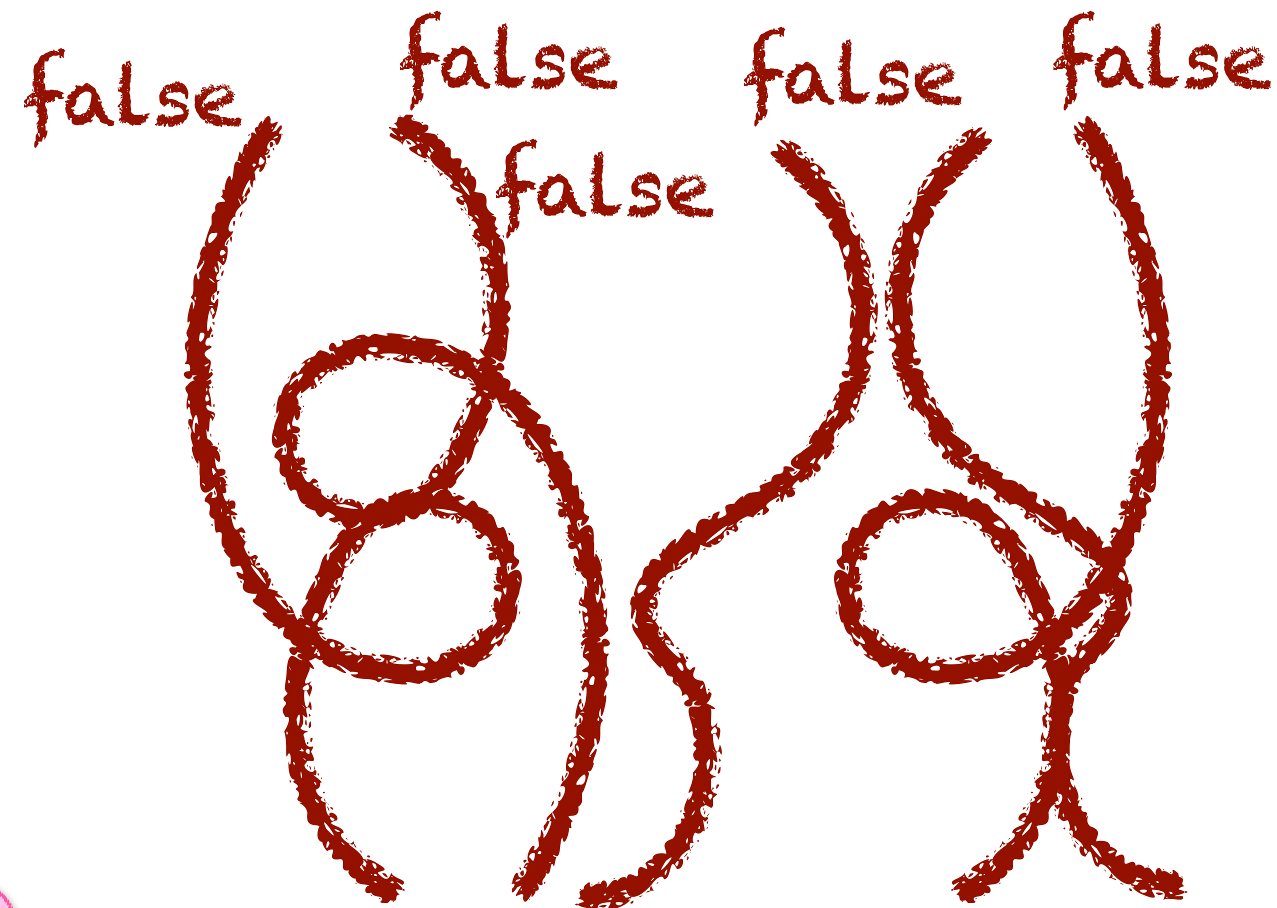
`mHostDb == null`

Goal-directed program analysis

Given a program configuration **goal**, derive a **contradiction** w.r.t. its reachability

safe?

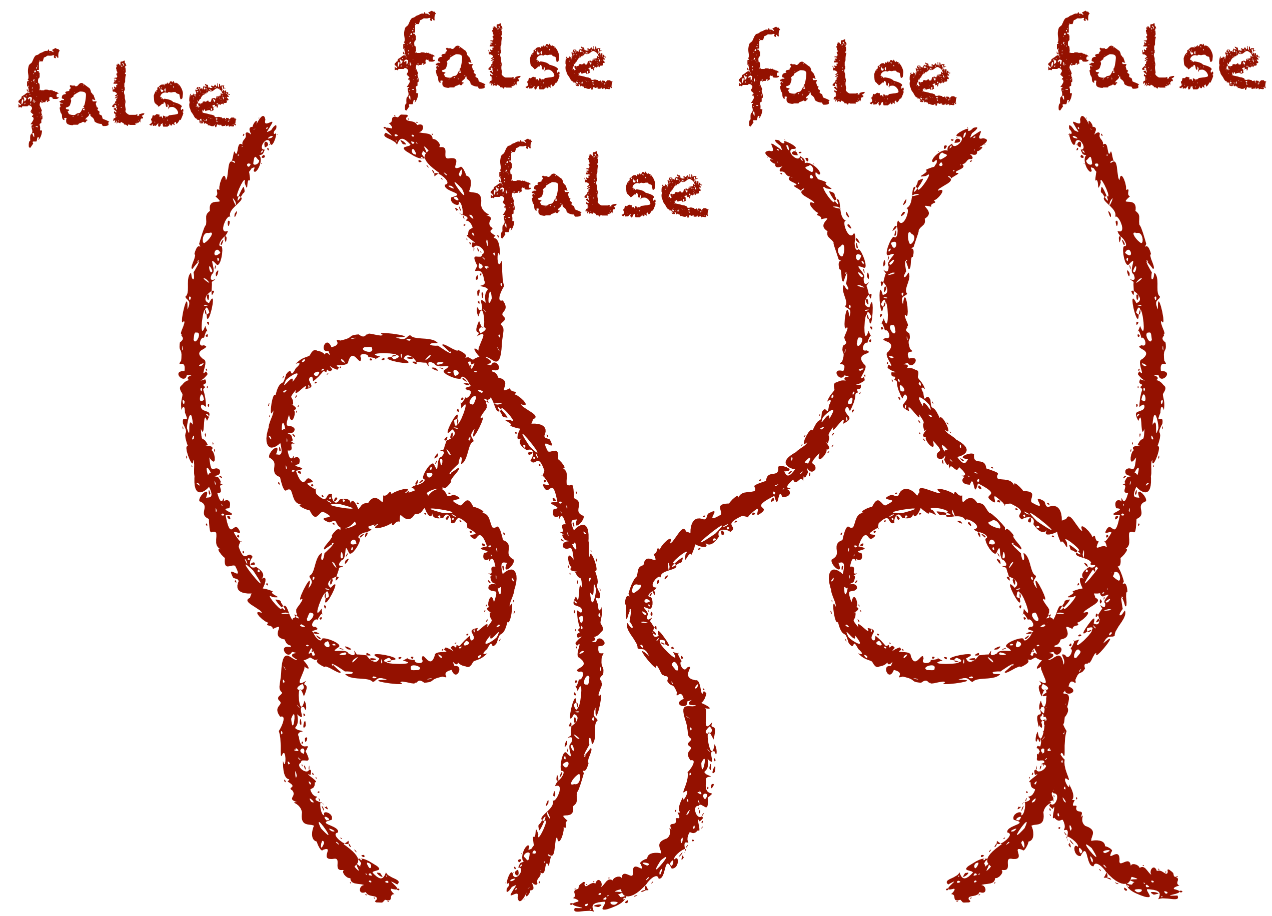
`mHostDb.s();`



`mHostDb == null`

Goal-directed program analysis

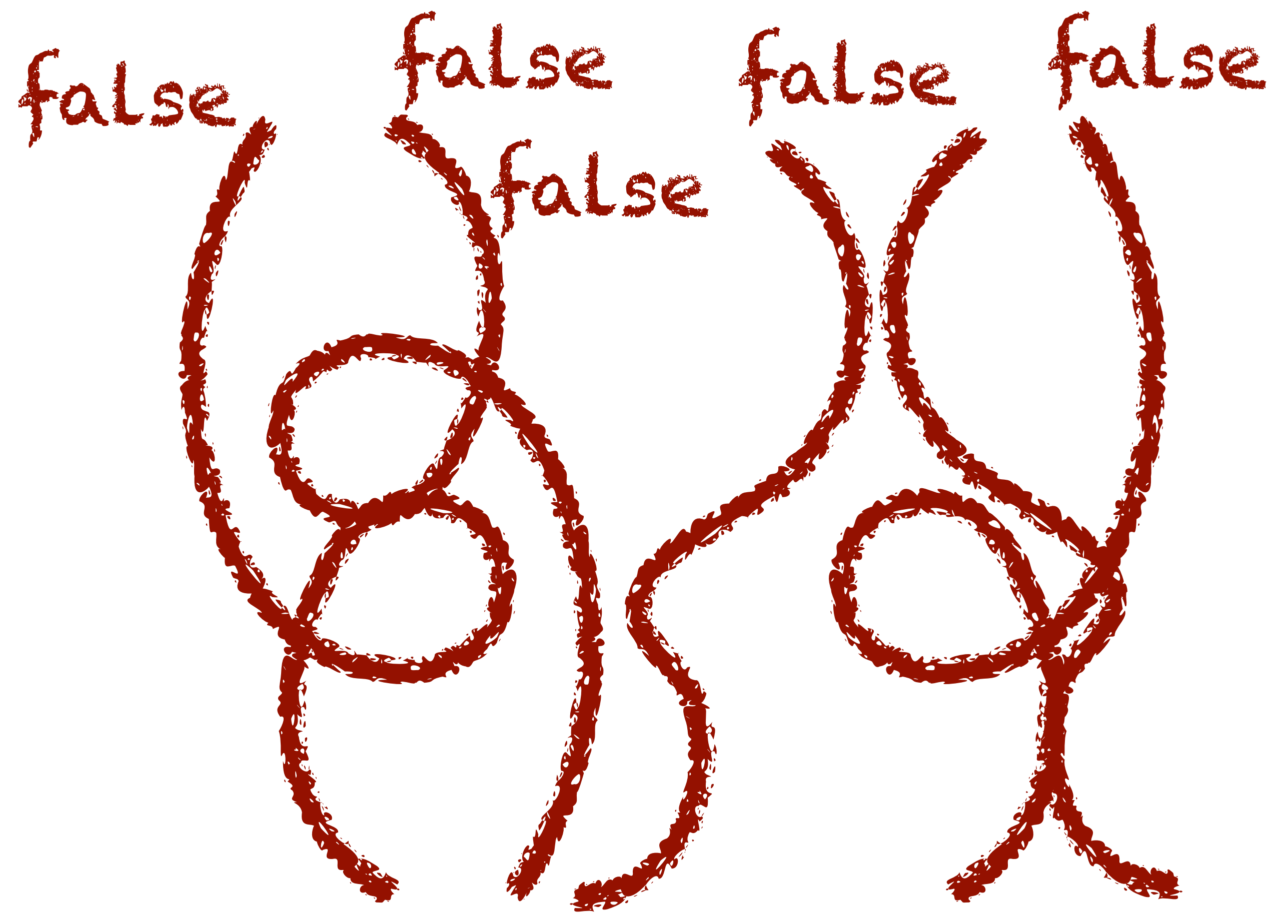
Given a program configuration **goal**, derive a **contradiction** w.r.t. its reachability



$$(\mathbf{this} \mapsto \widehat{t} * \widehat{t} \cdot \texttt{mHostDb} \mapsto \widehat{a} * \mathsf{true}) \wedge \widehat{a} = \mathsf{null}$$

Goal-directed program analysis

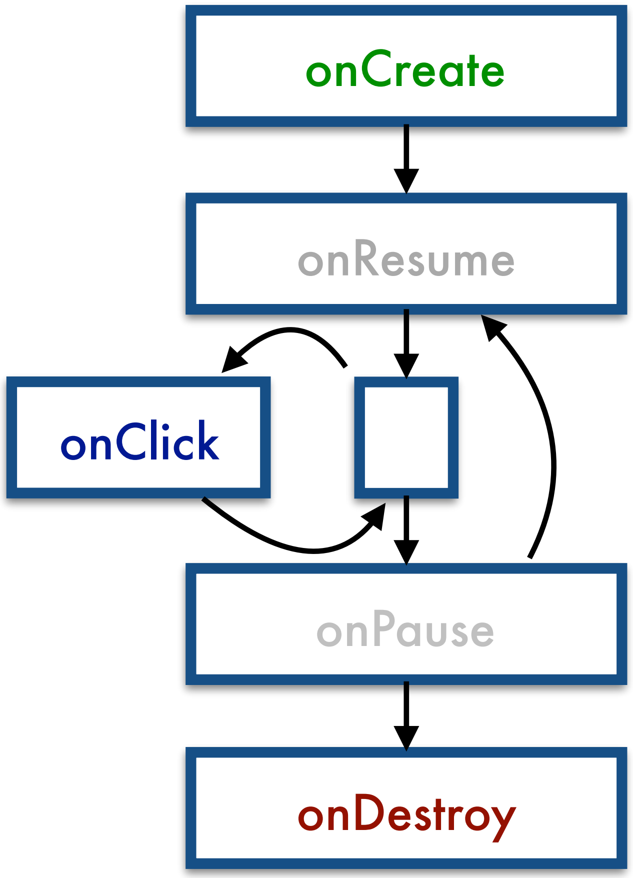
Given a program configuration **goal**, derive a **contradiction** w.r.t. its reachability



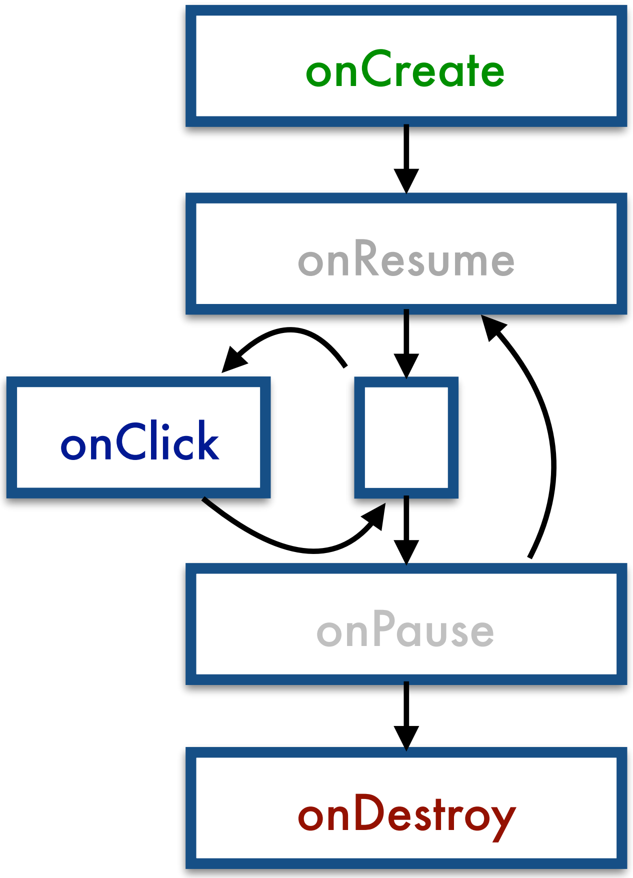
$$(\mathbf{this} \mapsto \widehat{t} * \widehat{t} \cdot \mathtt{mHostDb} \mapsto \widehat{a} * \mathsf{true}) \wedge \widehat{a} = \mathsf{null}$$

A precise backwards abstract interpretation with separation logic constraints to refute error conditions [PLDI'13]

Two dereferences: one safe and one buggy

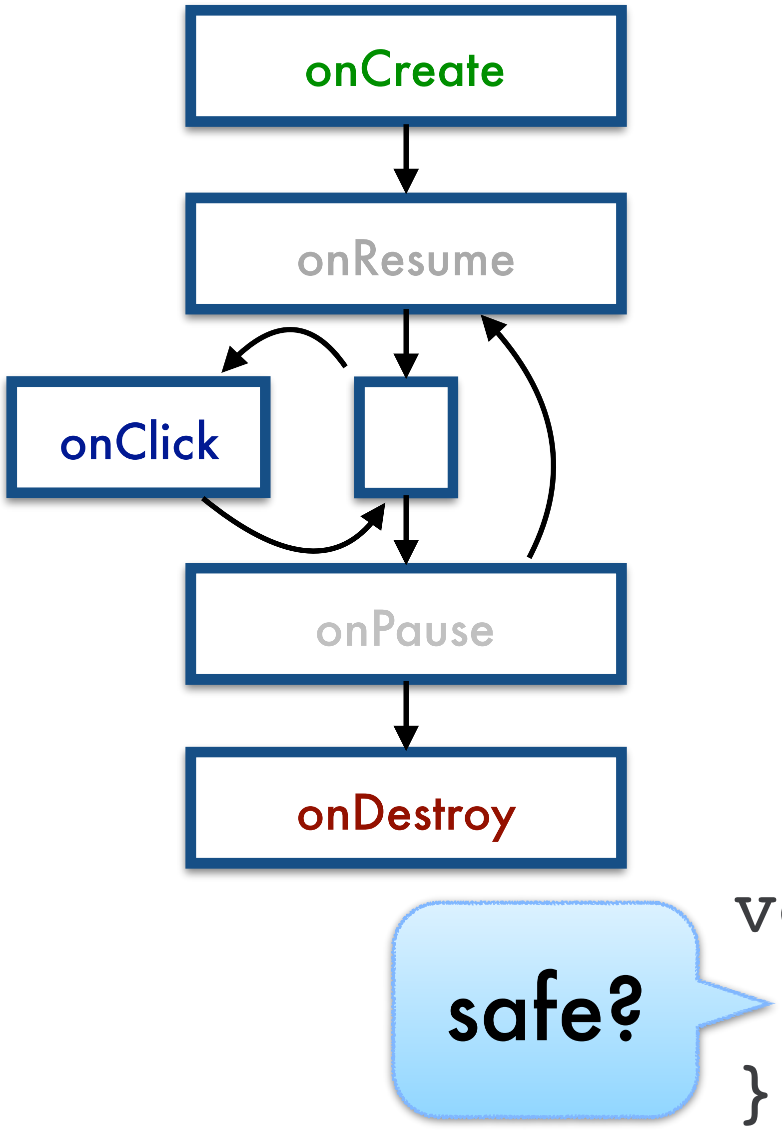


Two dereferences: one safe and one buggy



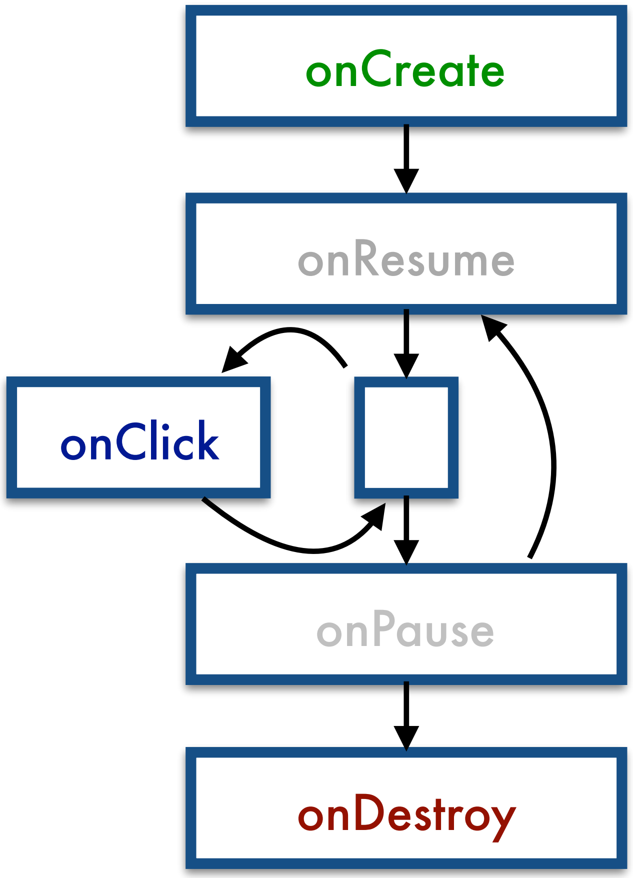

```
void onClick(…) {
  mHostDb.s(mService.g());
}
```

Two dereferences: one safe and one buggy




```
void onClick(…) {
  mHostDb.s(mService.g());
}
```

Two dereferences: one safe and one buggy

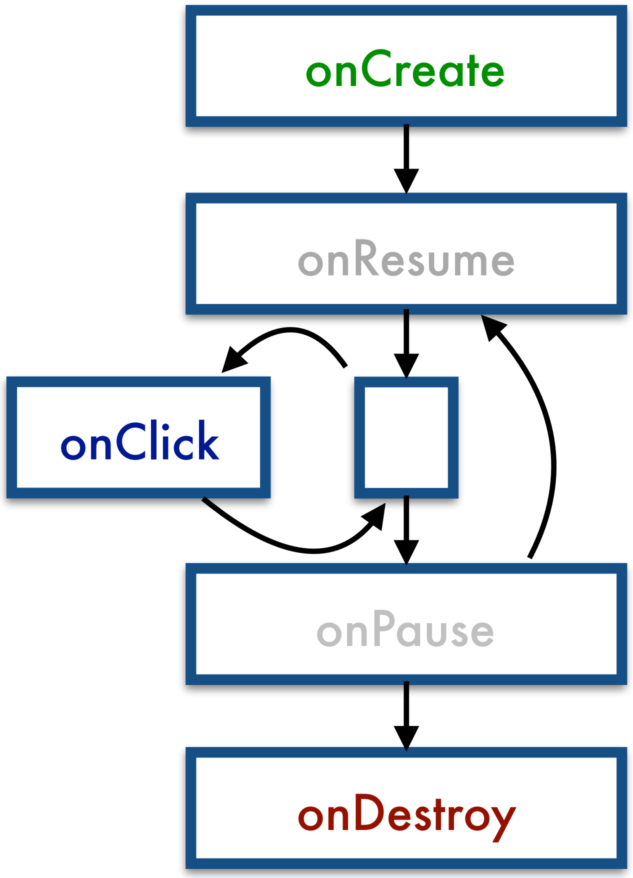


safe?

```
void onClick(…) {
  mHostDb.s(mService.g());
}
```

```
void onDestroy(…) {
  mHostDb = null;
  mService = null;
}
```

Two dereferences: one safe and one buggy

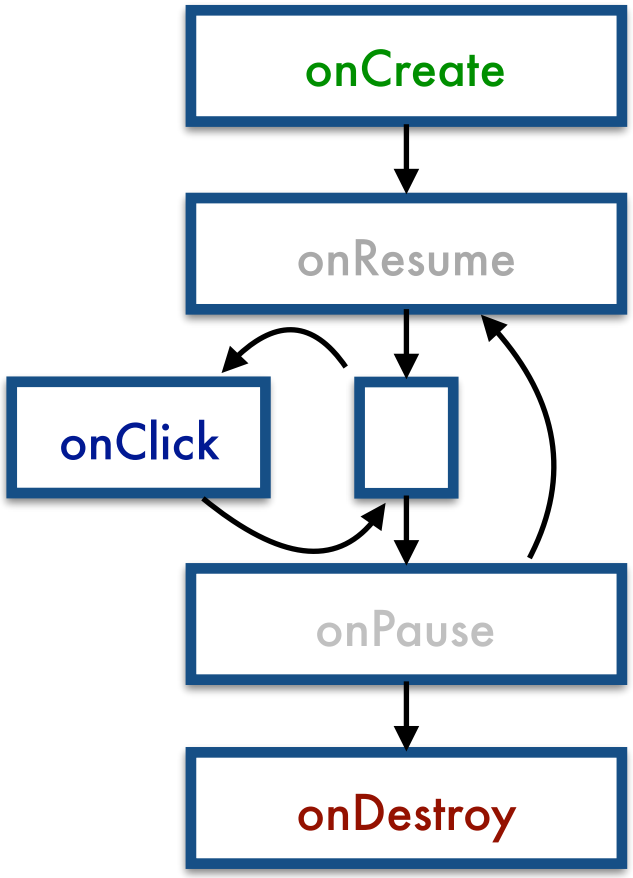


safe?

lifecycle constraints relevant

```
void onClick(…) {
  mHostDb.s(mService.g());
}

void onDestroy(…) {
  mHostDb = null;
  mService = null;
}
```

Two dereferences: one safe and one buggy




```
void onCreate() {
  bindService(…, new ServiceConn {
    void onConnected(@Nonnull Service s) {
      mService = s;
    }
  });
  mHostDb = new Db();
}
```

safe?

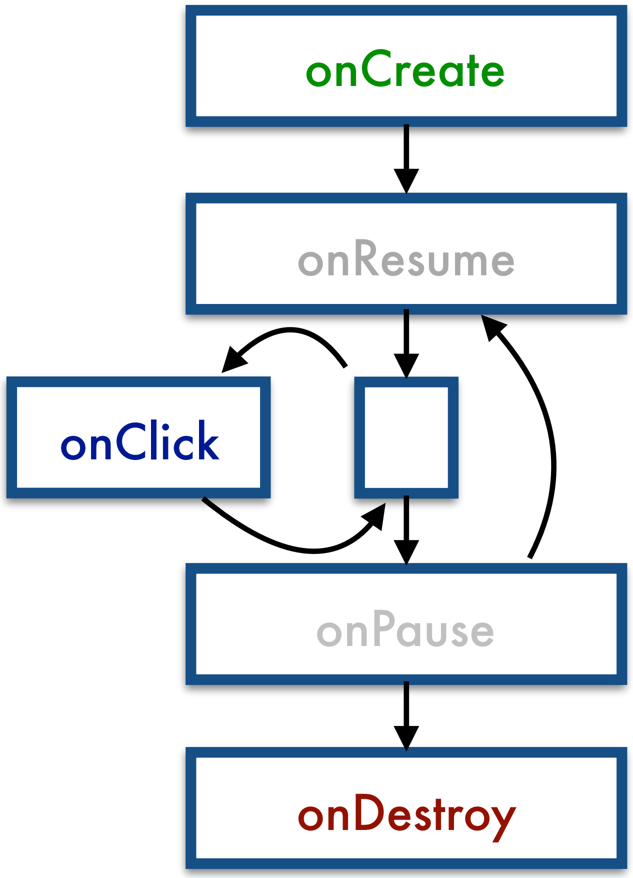
```
void onClick(…) {
  mHostDb.s(mService.g());
}
```

1 2

lifecycle constraints relevant

```
void onDestroy(…) {
  mHostDb = null;
  mService = null;
}
```

Two dereferences: one safe and one buggy




```
void onCreate() {
  bindService(…, new ServiceConn {
    void onConnected(@Nonnull Service s) {
      mService = s;
    }
  });
  mHostDb = new Db();
}
```

safe?

```
void onClick(…) {
  mHostDb.s(mService.g());
}
```

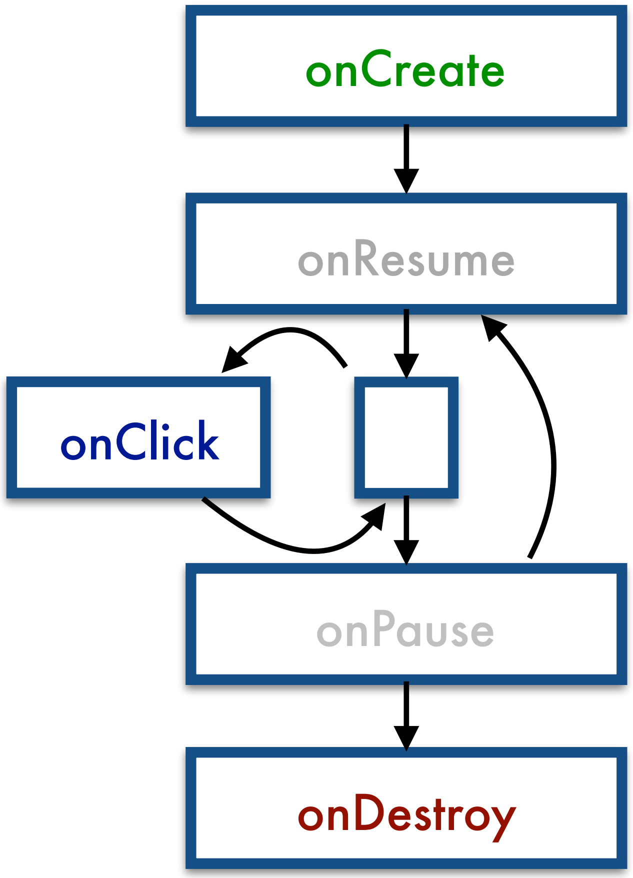
1

2

lifecycle constraints relevant

```
void onDestroy(…) {
  mHostDb = null;
  mService = null;
}
```

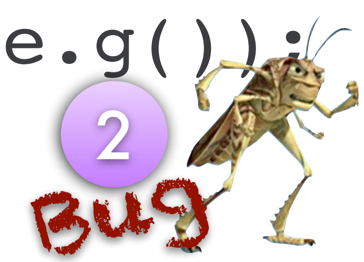
Two dereferences: one safe and one buggy




```
void onCreate() {
  bindService(…, new ServiceConn {
    void onConnected(@Nonnull Service s) {
      mService = s;
    }
  });
  mHostDb = new Db();
}

void onClick(…) {
  mHostDb.s(mService.g());
}

void onDestroy(…) {
  mHostDb = null;
  mService = null;
}
```

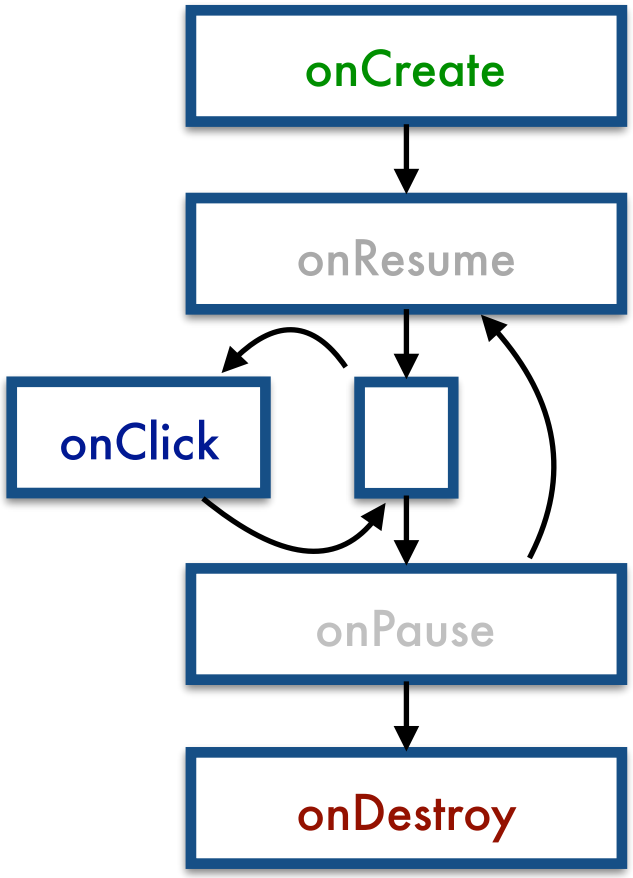
safe?

1 ✔ 2



lifecycle constraints relevant

Two dereferences: one safe and one buggy



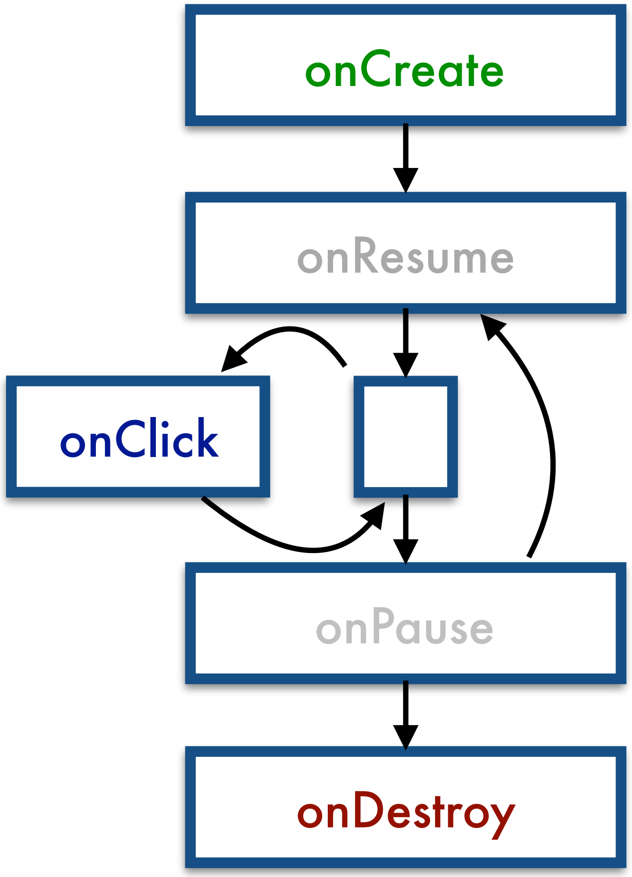

```
void onCreate() {
  bindService(…, new ServiceConn {
    void onConnected(@Nonnull Service s) {
      mService = s;
    }
  });
  mHostDb = new Db();
}

void onClick(…) {
  mHostDb.s(mService.g());
```

safe?

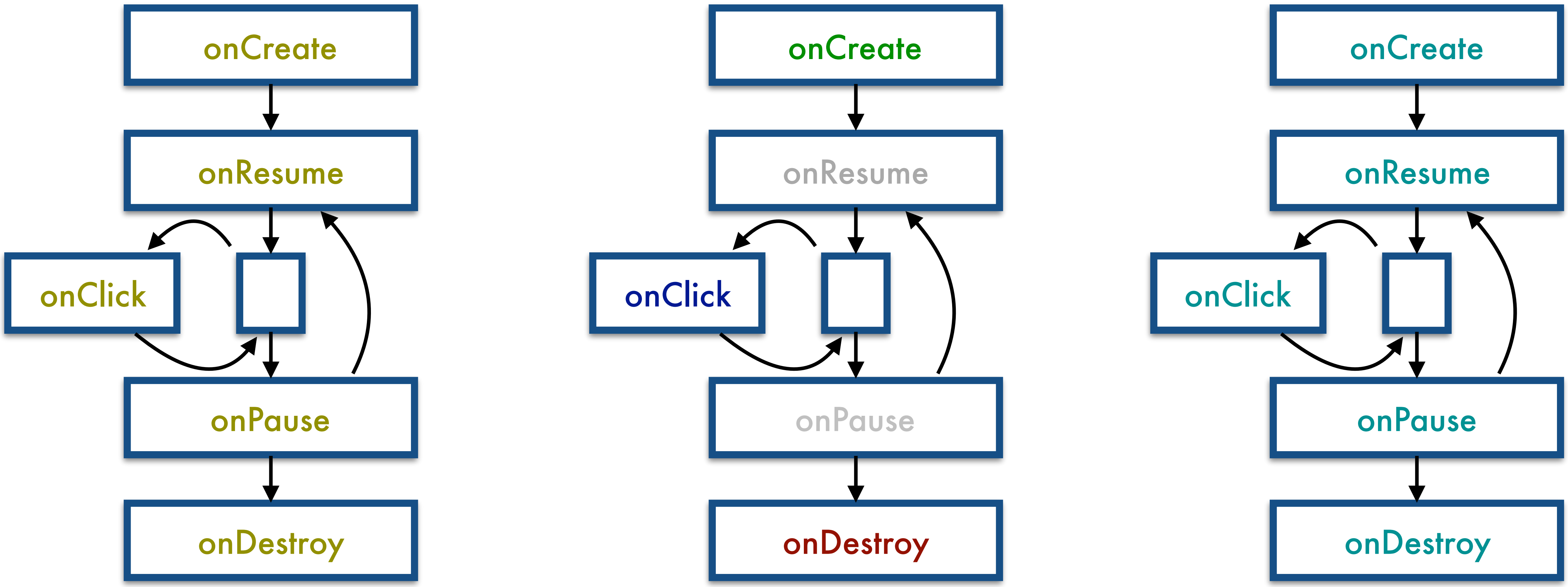
```
}
```

Need to consider some but not all callback ordering constraints

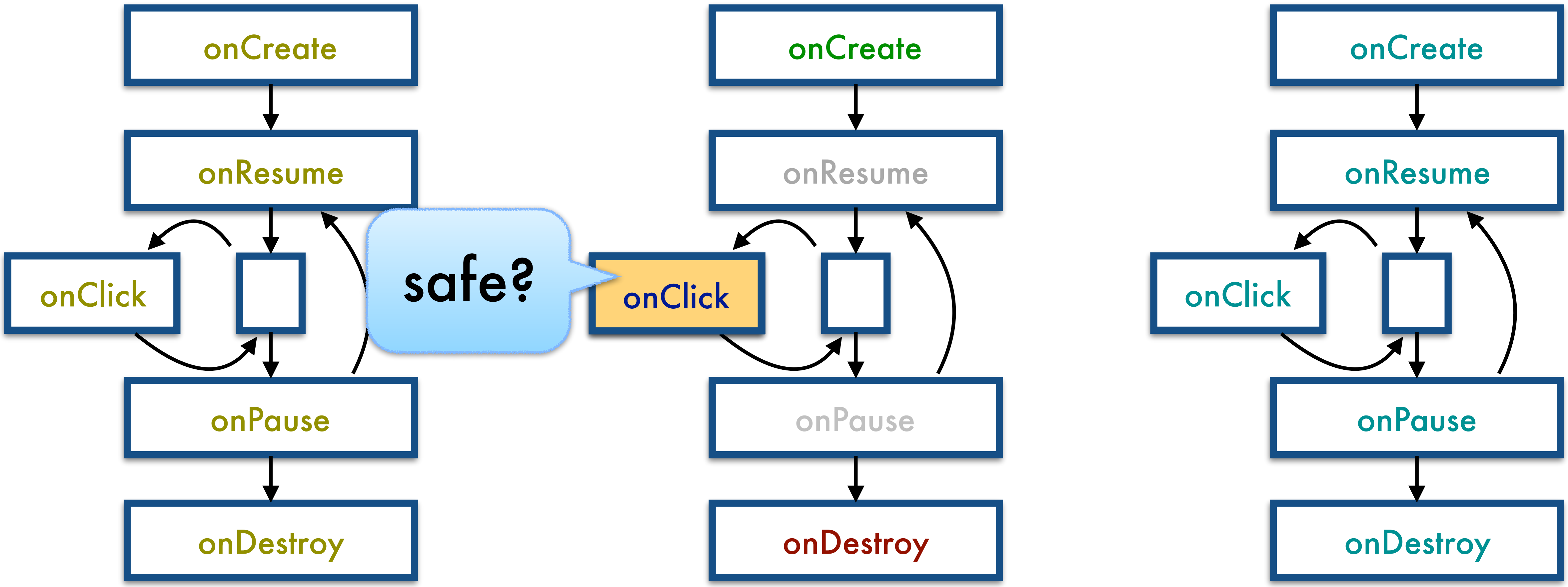
Idea: Jumping to relevant callbacks



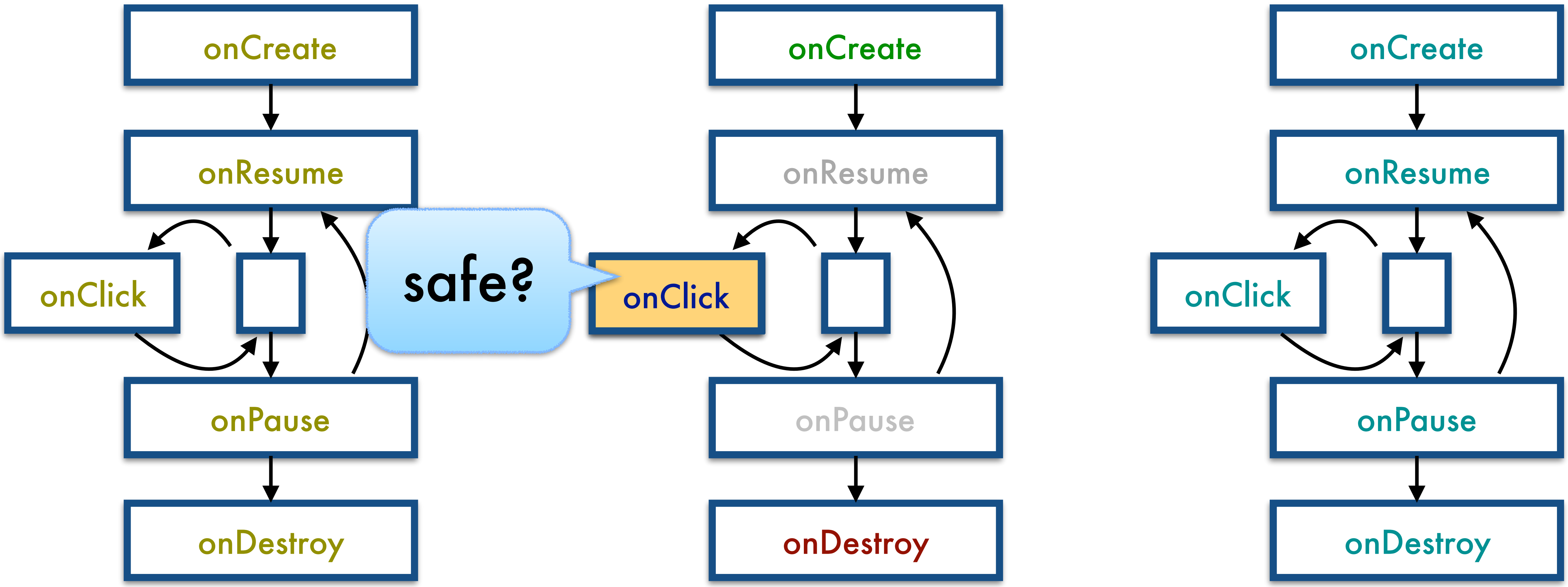
Idea: Jumping to relevant callbacks



Idea: Jumping to relevant callbacks

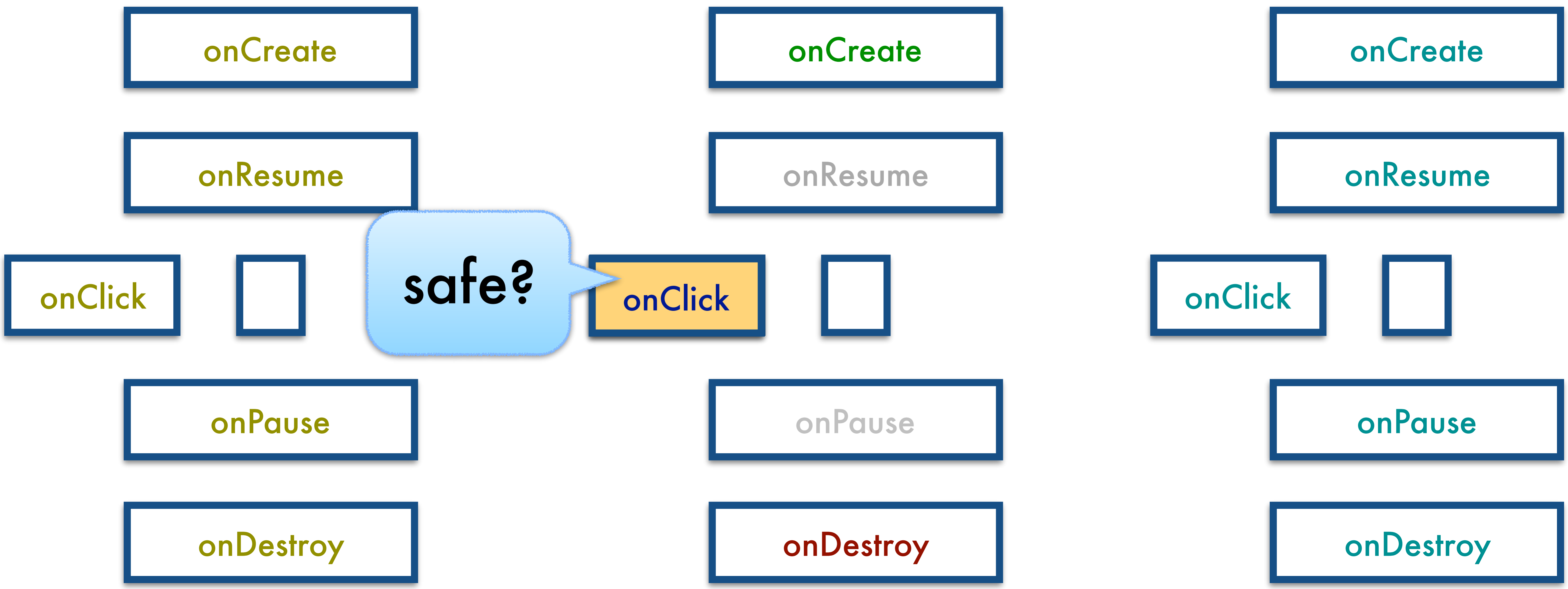


Idea: Jumping to relevant callbacks



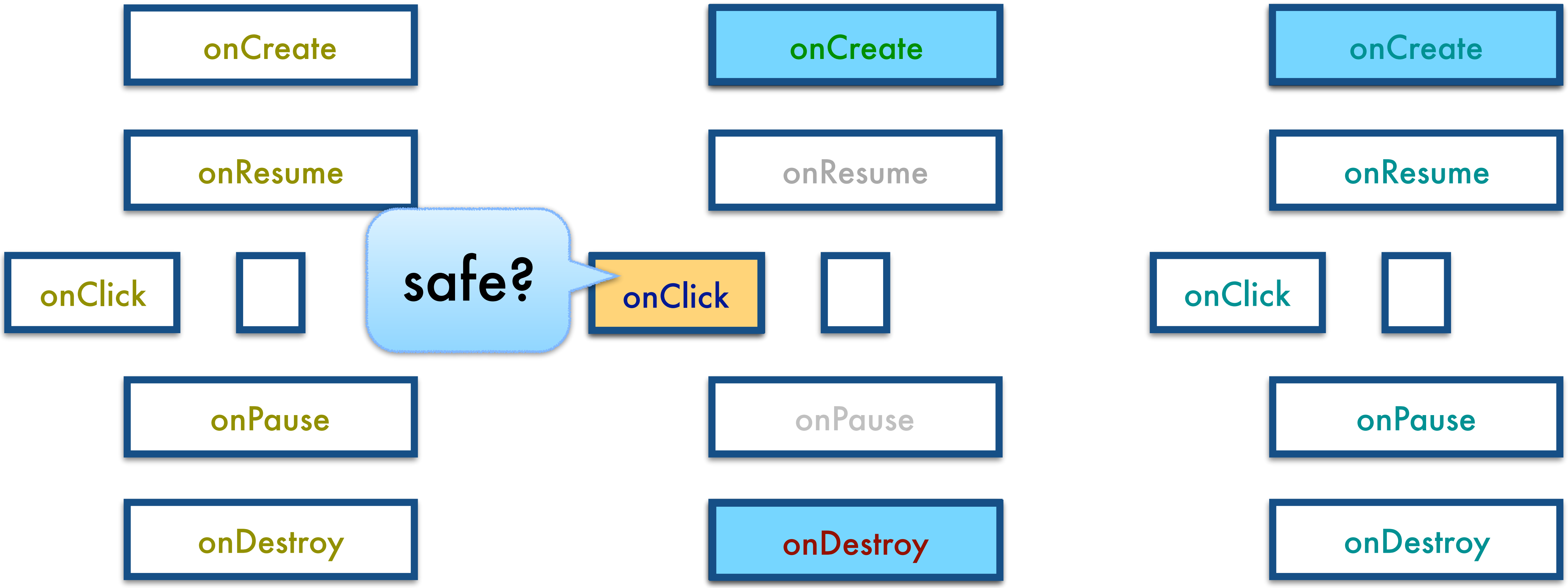
Find data relevant callbacks

Idea: Jumping to relevant callbacks



Find data relevant callbacks

Idea: Jumping to relevant callbacks



Find data relevant callbacks

Idea: Jumping to relevant callbacks

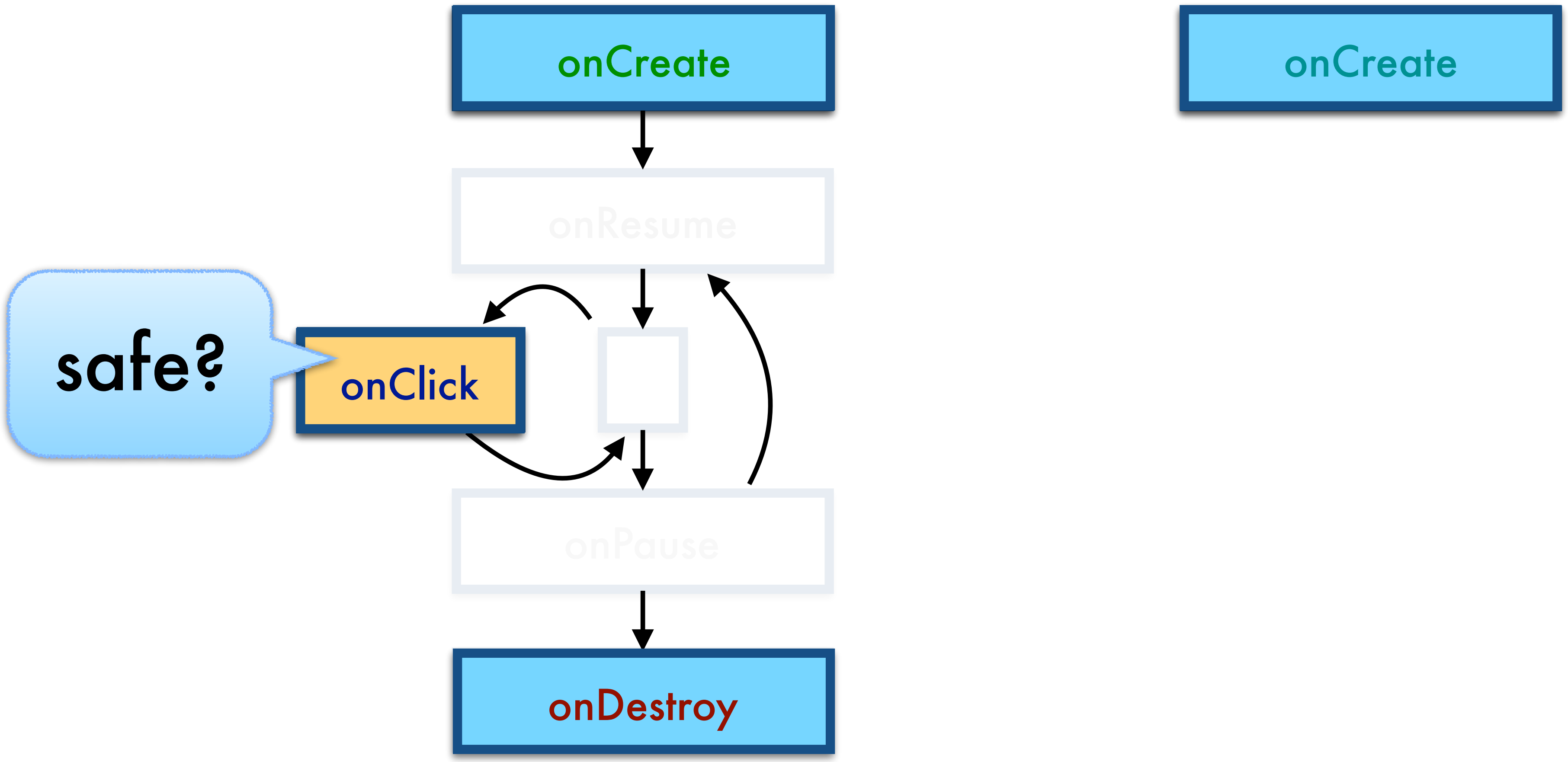
Find data relevant callbacks

Idea: Jumping to relevant callbacks

Find data relevant callbacks

Filter using control-feasibility

Idea: Jumping to relevant callbacks



Find data relevant callbacks

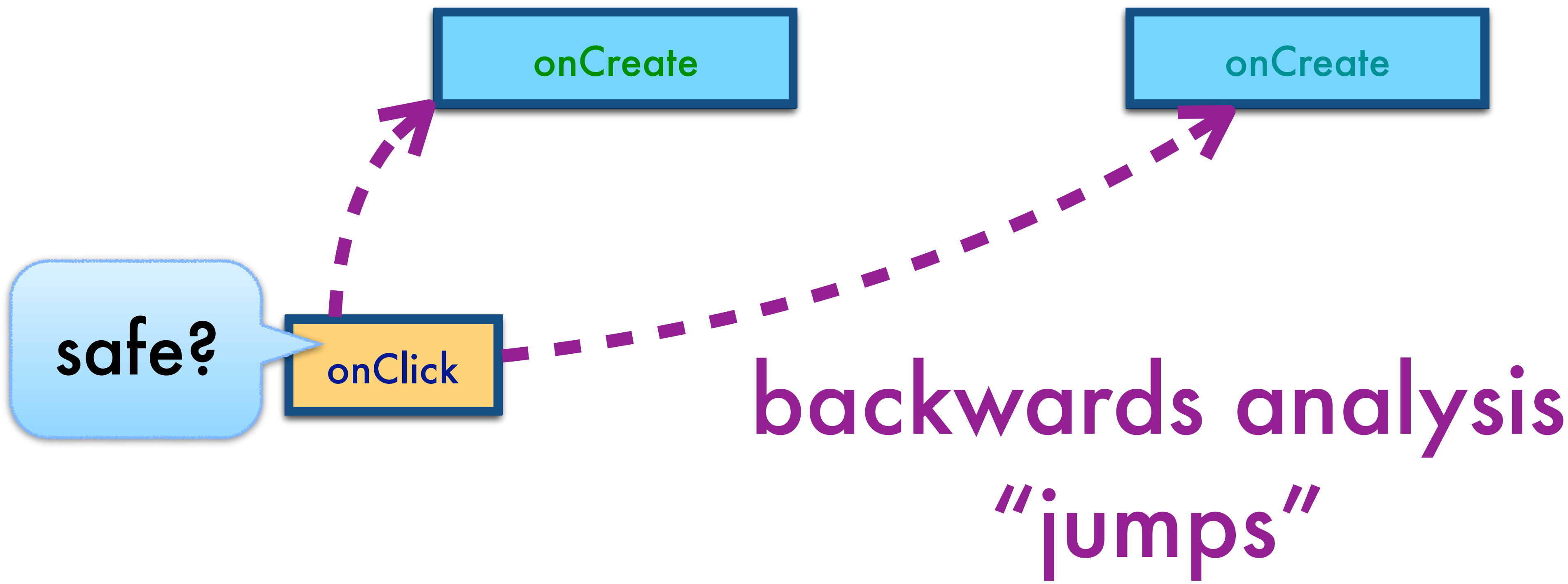
Filter using control-feasibility

Idea: Jumping to relevant callbacks

Find data relevant callbacks

Filter using control-feasibility

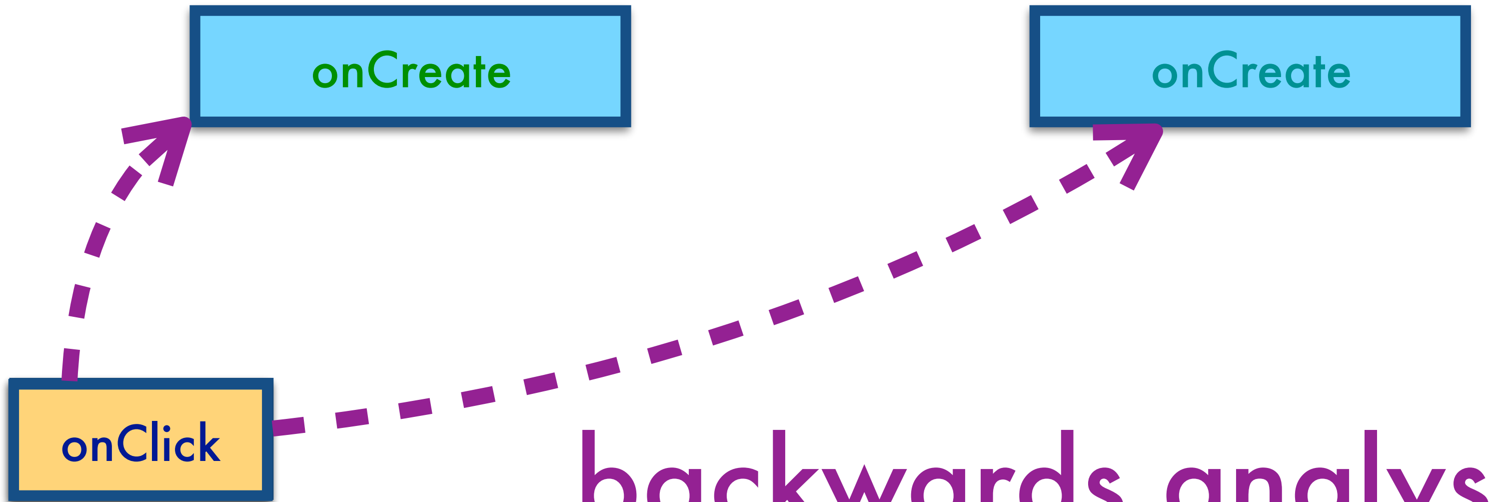
Idea: Jumping to relevant callbacks



Find data relevant callbacks

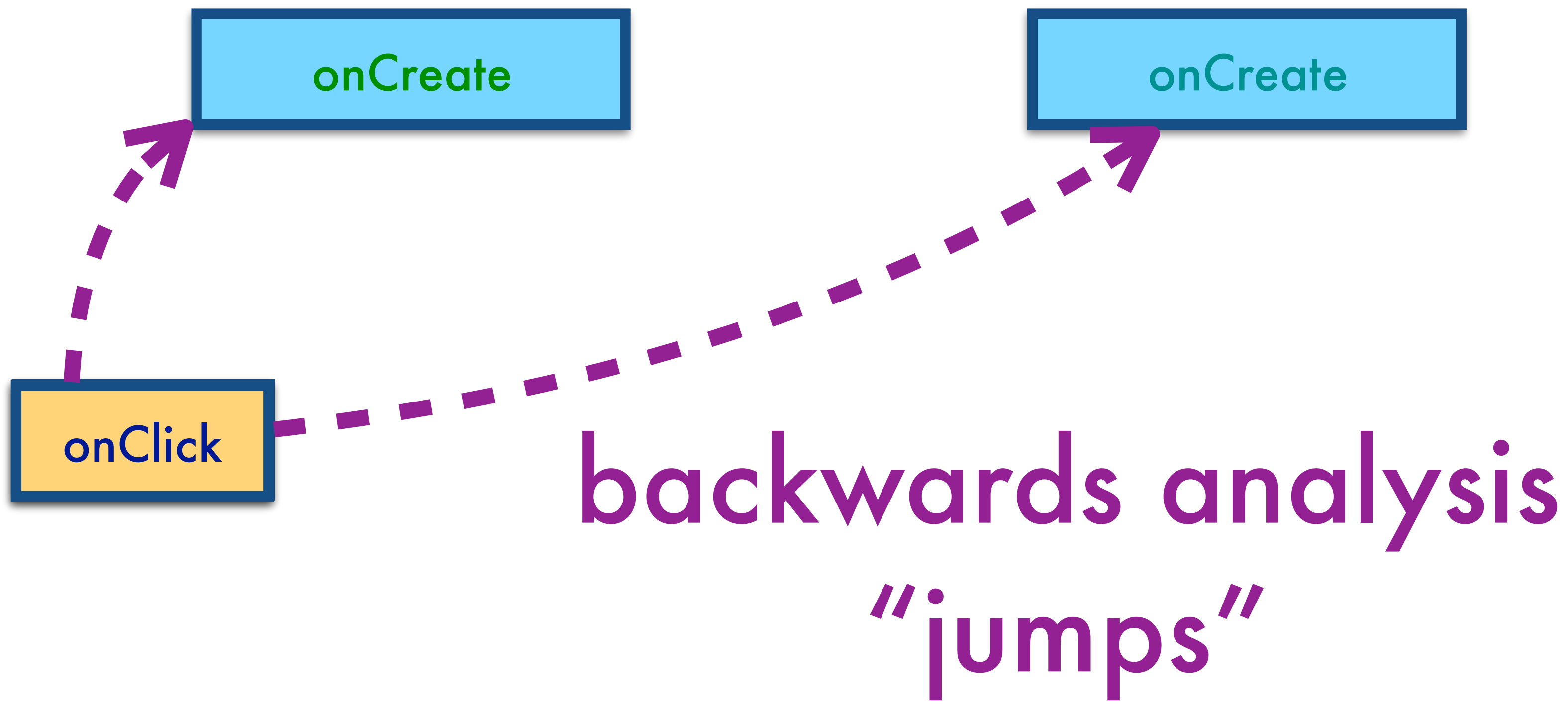
Filter using control-feasibility

Contributions



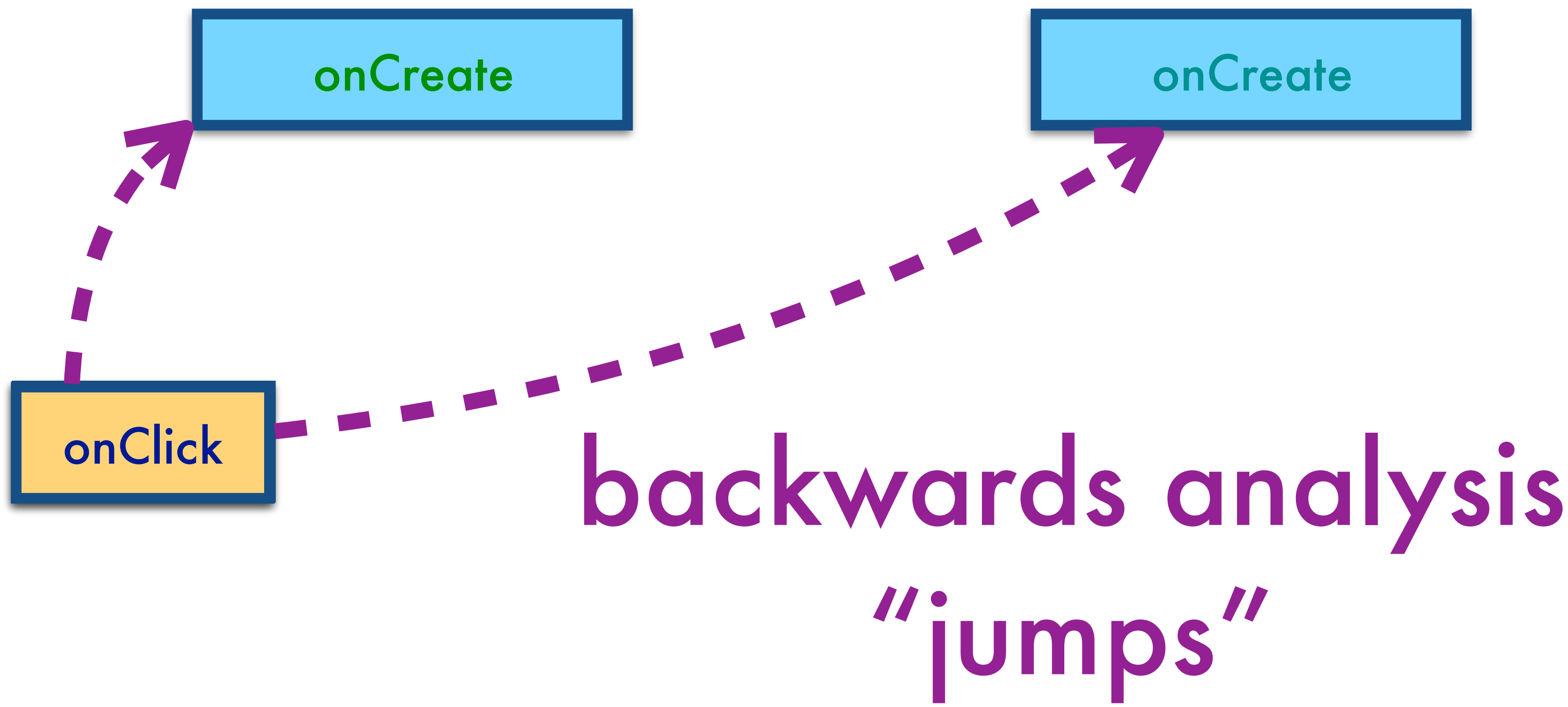
backwards analysis
"jumps"

Contributions



1 $\ell_1 \xrightarrow{[c]} \ell_2$ **Framework for jumping analyses**

Contributions



1 $\ell_1 \;\dashv\!\![c]\!\!\vdash\!\!\rightarrow\; \ell_2$ Framework for jumping analyses

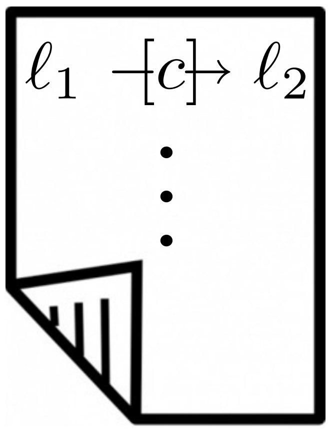
2  Applied to Android events

A program model for a jumping analysis

1 $\ell_1 \;\text{-}\!\!\left[c\right]\!\!\rightarrow \ell_2$

A program model for a jumping analysis

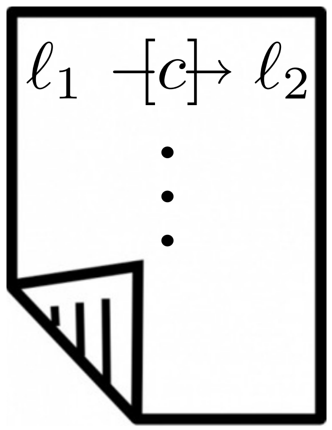
1 $\ell_1 \ \text{-}[c]\!\!\rightarrow\ \ell_2$



Program

A program model for a jumping analysis

1 $\ell_1 \;\text{-}\!\![c]\!\!\rightarrow\; \ell_2$

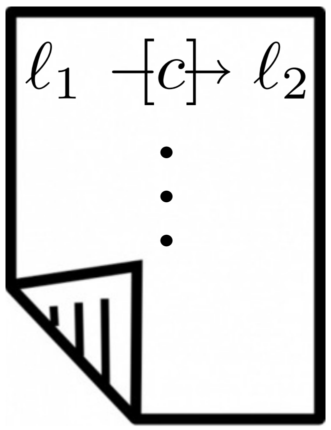


Program

Unstructured control-flow: atomic commands connected by pre/post labels

A program model for a jumping analysis

1 $\ell_1 \;\text{-}\![c]\!\!\rightarrow\; \ell_2$



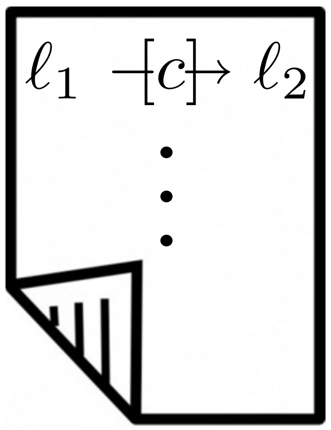
Program

Unstructured control-flow: atomic commands connected by pre/post labels

Represent conditionals with `assume`, loops with `assume` and a back edge.

A program model for a jumping analysis

1 $\ell_1 \mathrel{-\!\![c]\!\!\to} \ell_2$



Program

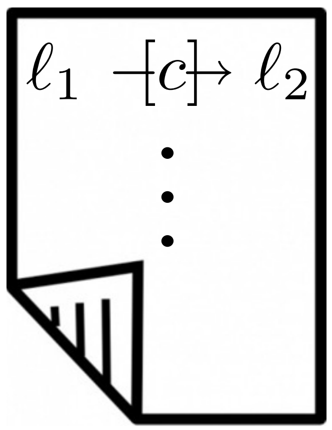
Unstructured control-flow: atomic commands connected by pre/post labels

Represent conditionals with `assume`, loops with `assume` and a back edge.

Analysis parameters

A program model for a jumping analysis

1 $\ell_1 \;{-}[c]{\rightarrow}\; \ell_2$



Program

Unstructured control-flow: atomic commands connected by pre/post labels

Represent conditionals with `assume`, loops with `assume` and a back edge.

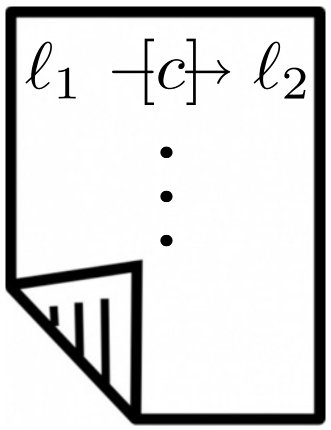
Analysis parameters

$$\{Q'\}\ c\ \{Q\}$$

Backward transfer functions

A program model for a jumping analysis

1 $\ell_1 \mathrel{-\!\![c]\!\!\rightarrow} \ell_2$



Program

Unstructured control-flow: atomic commands connected by pre/post labels

Represent conditionals with `assume`, loops with `assume` and a back edge.

Analysis parameters

$$\{Q'\}\ c\ \{Q\}$$

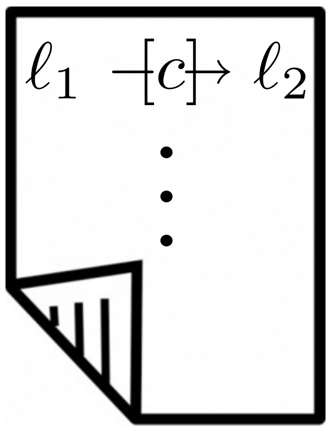
Backward transfer functions

$$\ell_{\mathrm{cur}}$$

Current location

A program model for a jumping analysis

$\ell_1 \;{-}\!\![c]\!\!\rightarrow\; \ell_2$



Program

Unstructured control-flow: atomic commands connected by pre/post labels

Represent conditionals with `assume`, loops with `assume` and a back edge.

Analysis parameters

$$\{Q'\}\; c\; \{Q\}$$

Backward transfer functions

$$\ell_{\mathrm{cur}}$$

Current location

$$Q_{\mathrm{cur}}$$

Current query

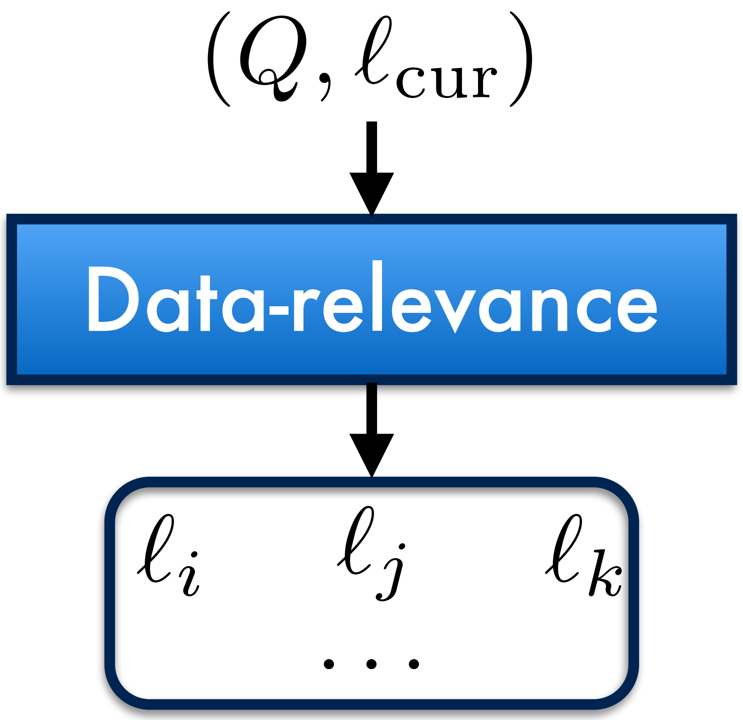
Data-relevance identifies writes

1 $\ell_1 \xrightarrow{[c]} \ell_2$

(Q, ℓ_{cur})

Data-relevance identifies writes

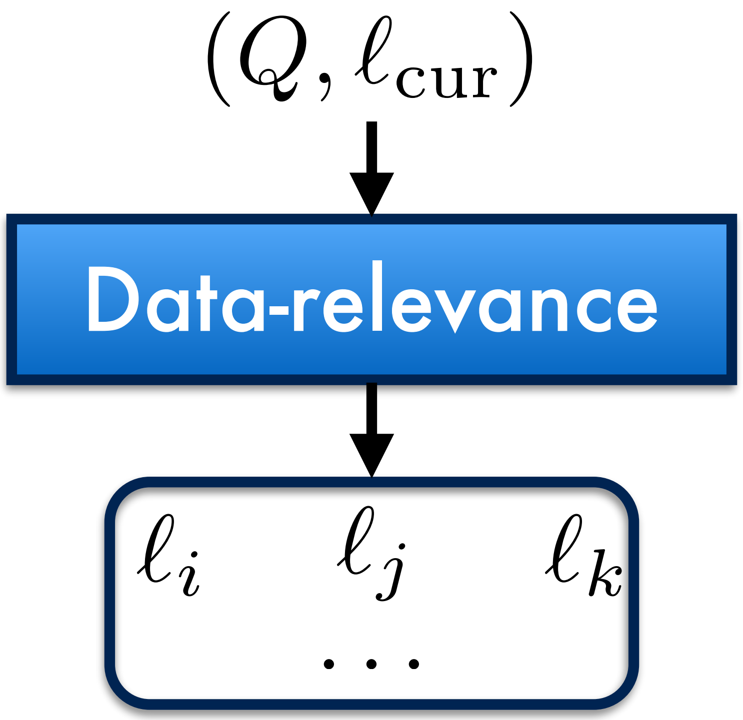
1 $\ell_1 \;-\!\![c]\!\!\rightarrow\; \ell_2$



Identify locations that can write to the query Q.

Data-relevance identifies writes

1 $\ell_1 \; \text{-}[c]\!\rightarrow \; \ell_2$

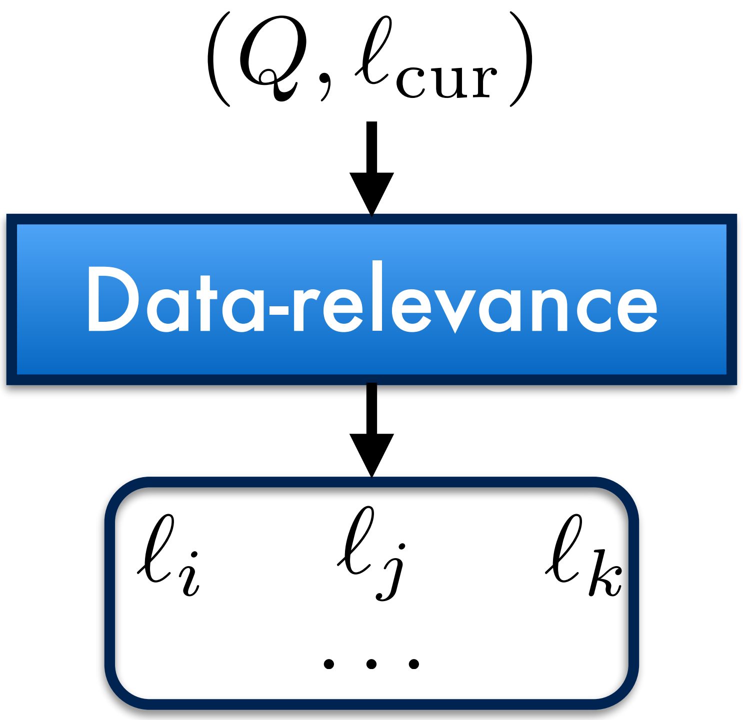


Identify locations that can write to the query Q.

Computed using pre-pass points-to analysis, types, field-based, ...

Data-relevance identifies writes

1 $\ell_1 \xrightarrow{[c]} \ell_2$



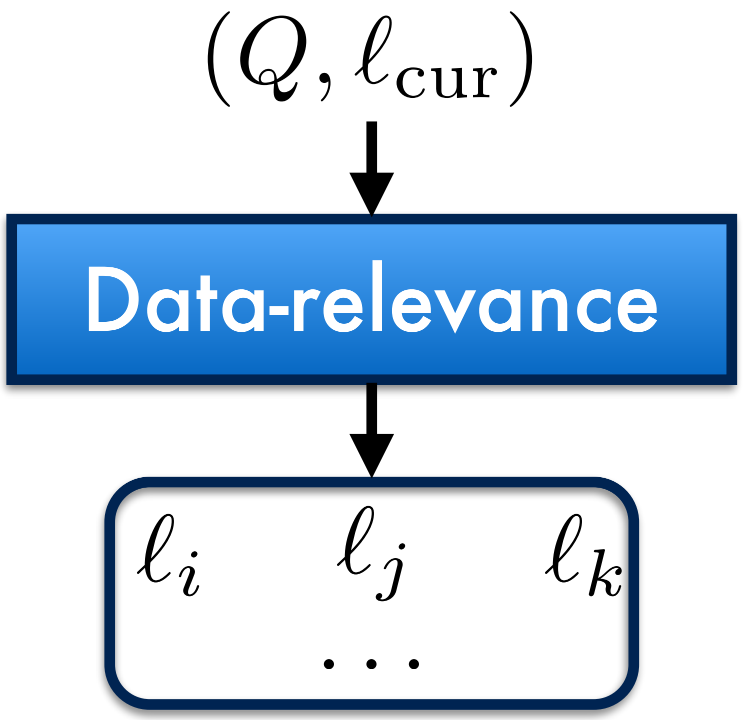
Identify locations that can write to the query Q.

Computed using pre-pass points-to analysis, types, field-based, …

Classic idea: Following data dependencies yields a sparse analysis (but, here, flow-insensitive)

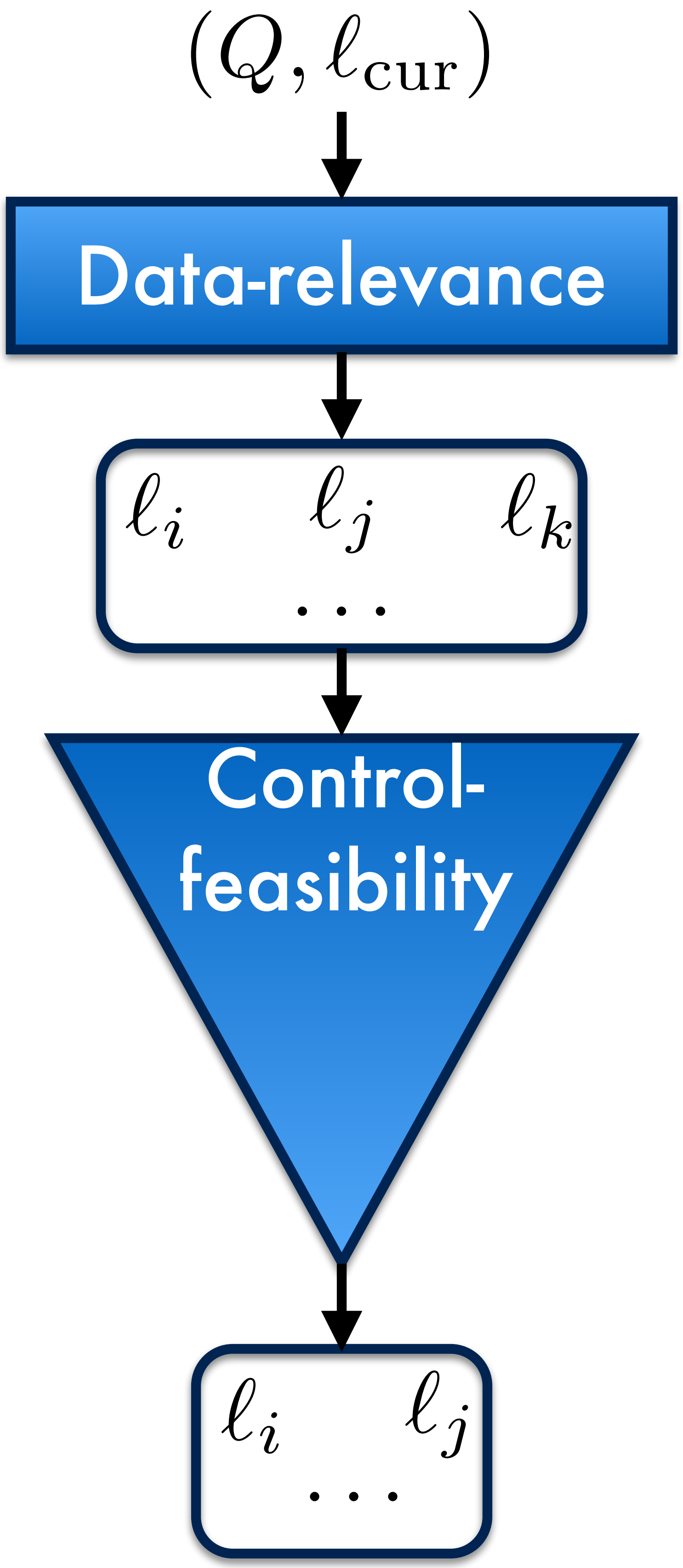
Control-feasibility **selectively** recovers flow-sensitivity

1 $\ell_1 \;\text{—}[c]\!\!\rightarrow\; \ell_2$



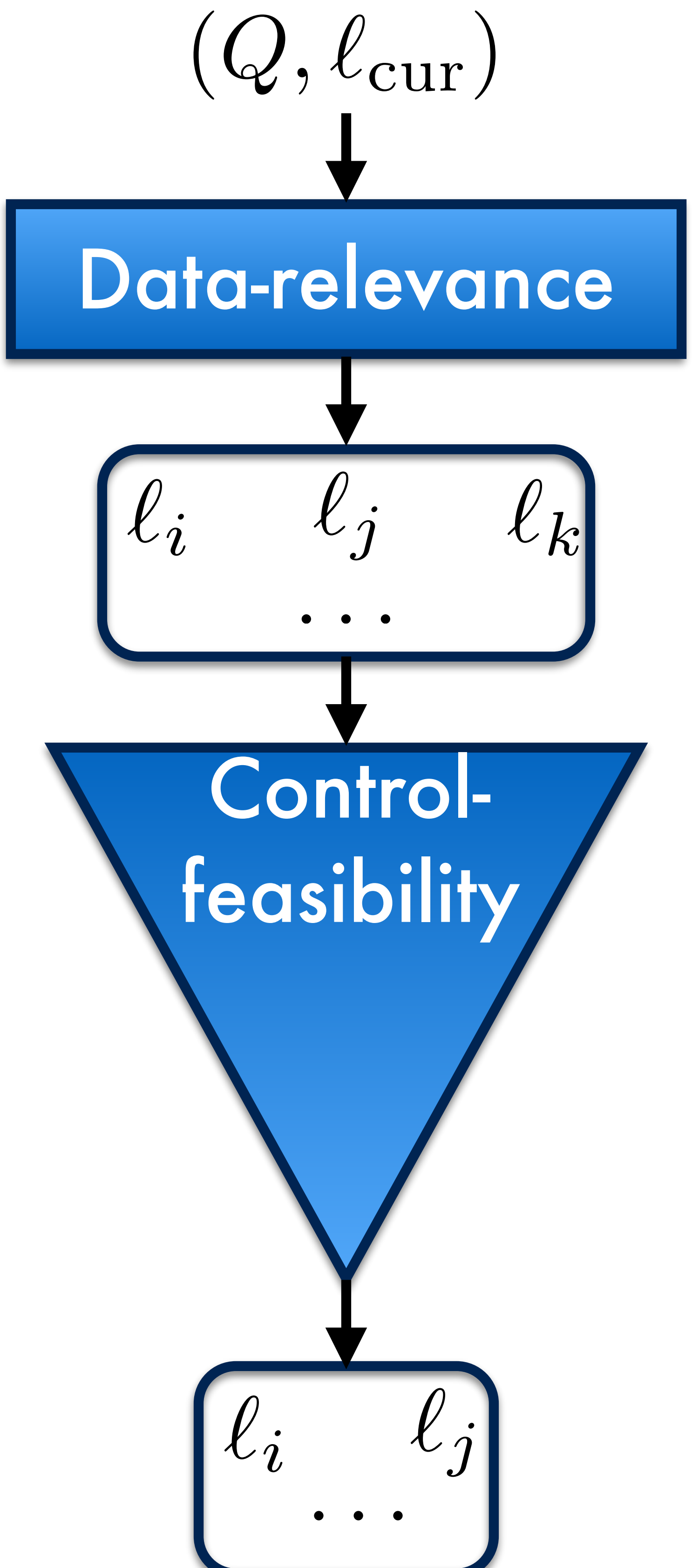
Control-feasibility selectively recovers flow-sensitivity

1 $\ell_1 \dashv\!\![c]\!\!\vdash\!\!\rightarrow \ell_2$



Control-feasibility selectively recovers flow-sensitivity

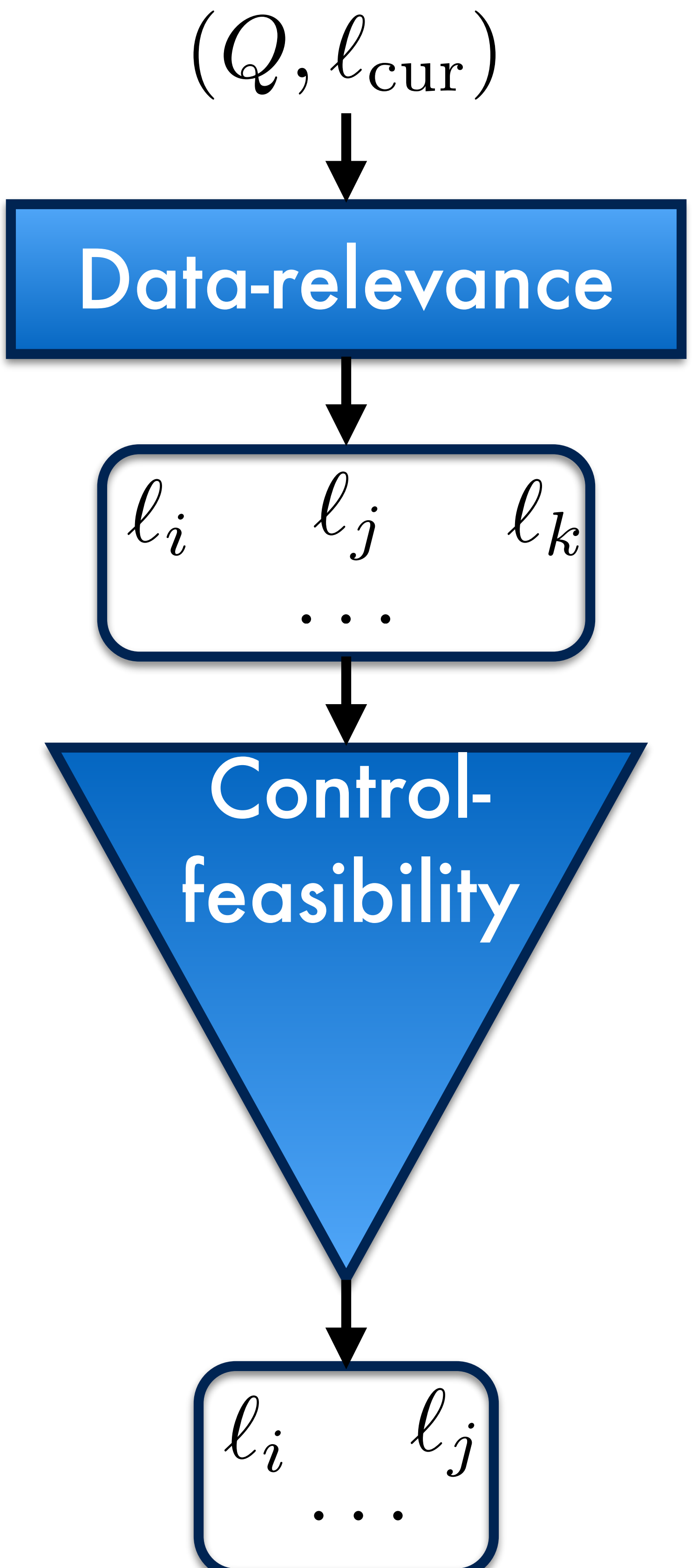
1 $\ell_1 \;\dashv\![c]\!\!\rightarrow\; \ell_2$



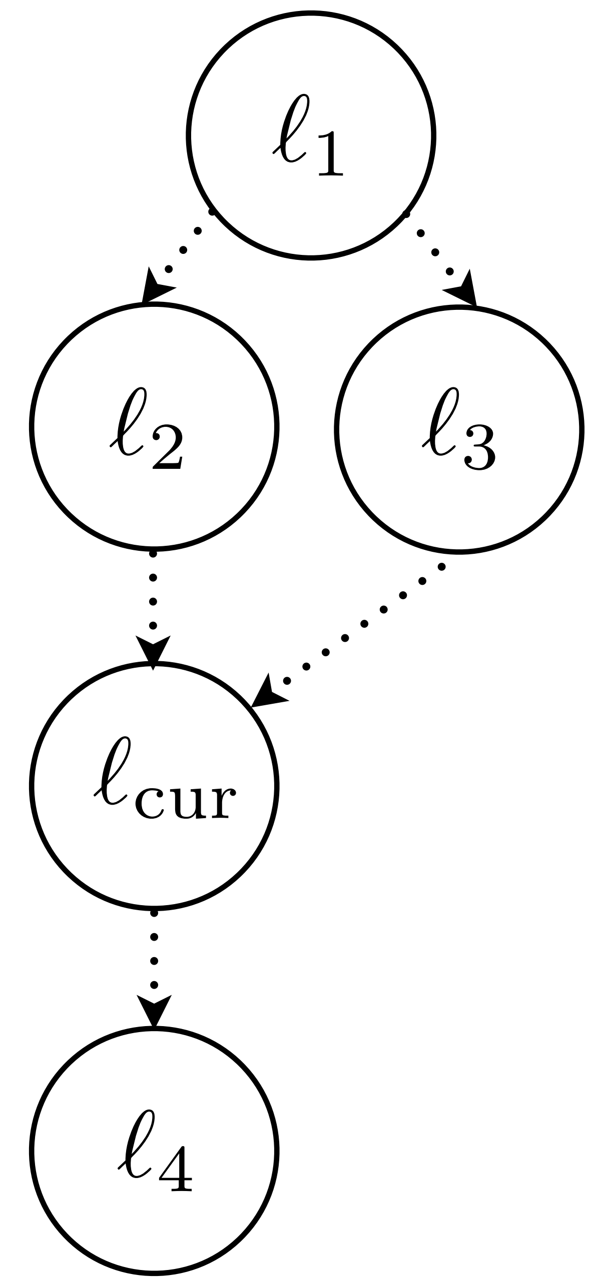
Filter the set of data-relevant locations using control flow and the current program point

Control-feasibility selectively recovers flow-sensitivity

1 $\ell_1 \xrightarrow{[c]} \ell_2$

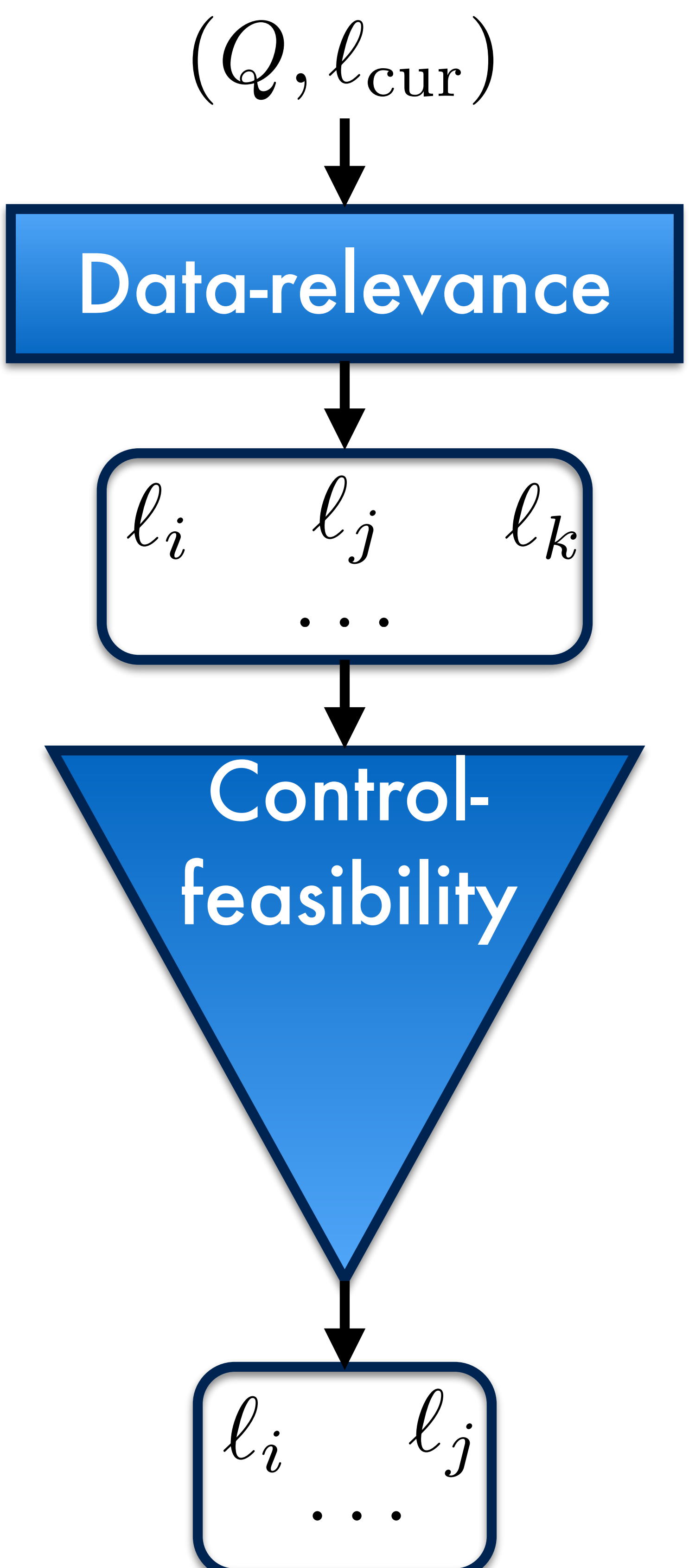


Filter the set of data-relevant locations using control flow and the current program point



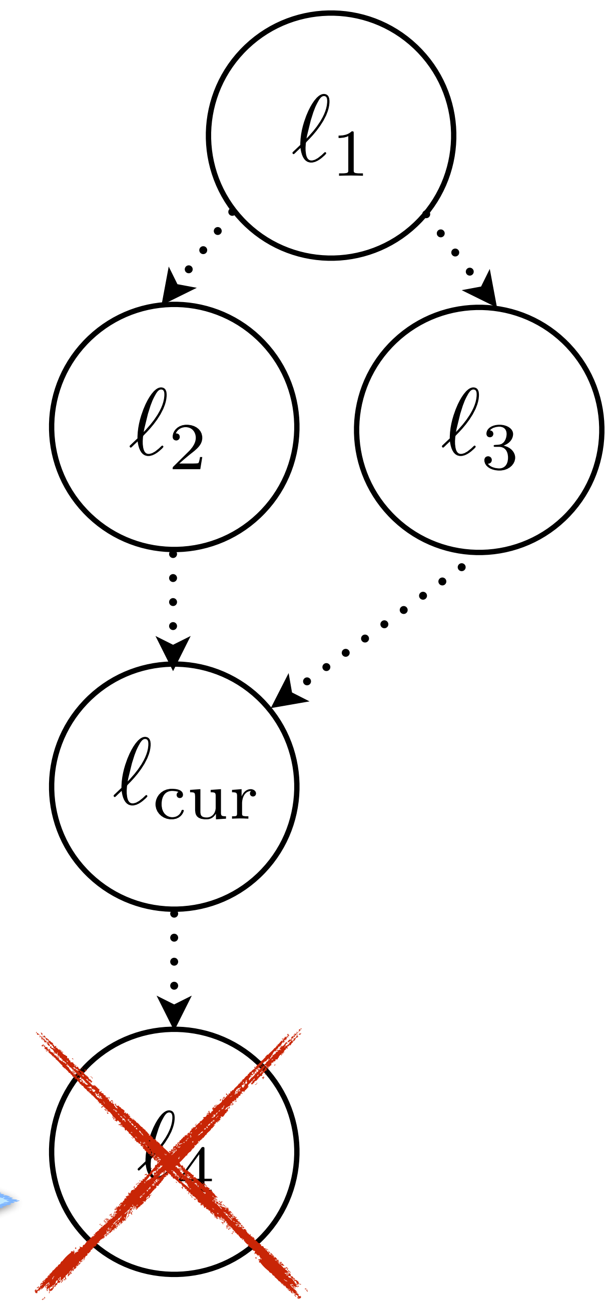
Control-feasibility selectively recovers flow-sensitivity

1 $\ell_1 \xrightarrow{[c]} \ell_2$



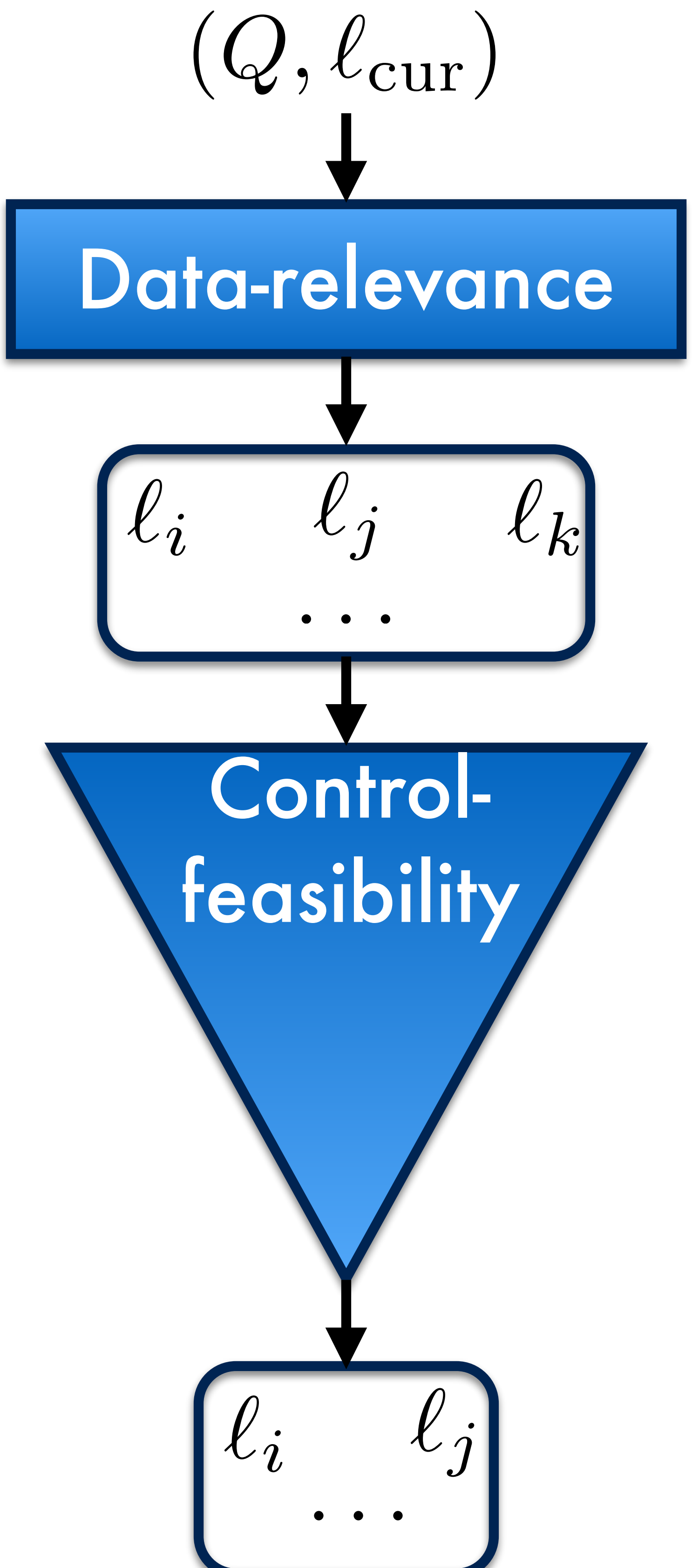
Filter the set of data-relevant locations using control flow and the current program point

Not backward-reachable from current location



Control-feasibility **selectively** recovers flow-sensitivity

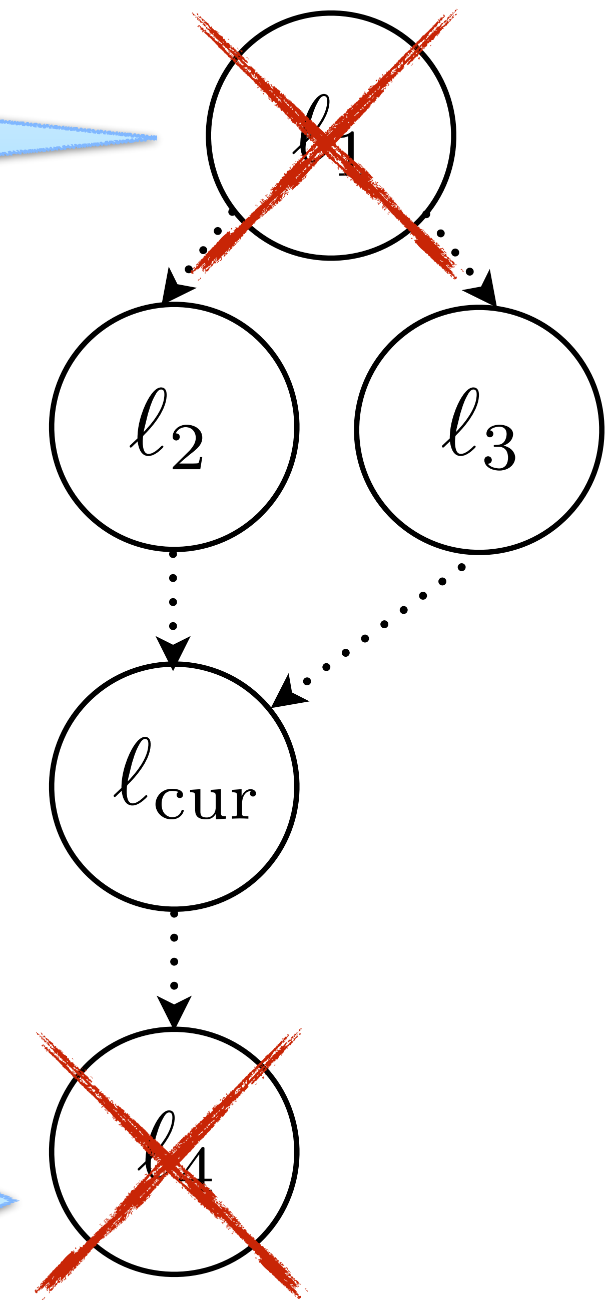
1 $\ell_1 \; \text{-}[c]\text{-}\!\!\rightarrow \; \ell_2$



Must visit another relevant location first.

Filter the set of data-relevant locations using control flow and the current program point

Not backward-reachable from current location



Jumping enables sparse, selective, on-the-fly control-flow abstraction

1

$$\ell_1 \xrightarrow{[c]} \ell_2$$

Jumping enables sparse, selective, on-the-fly control-flow abstraction

1

$$\ell_1 \xrightarrow{[c]} \ell_2$$

ℓ_{cur}

Current location

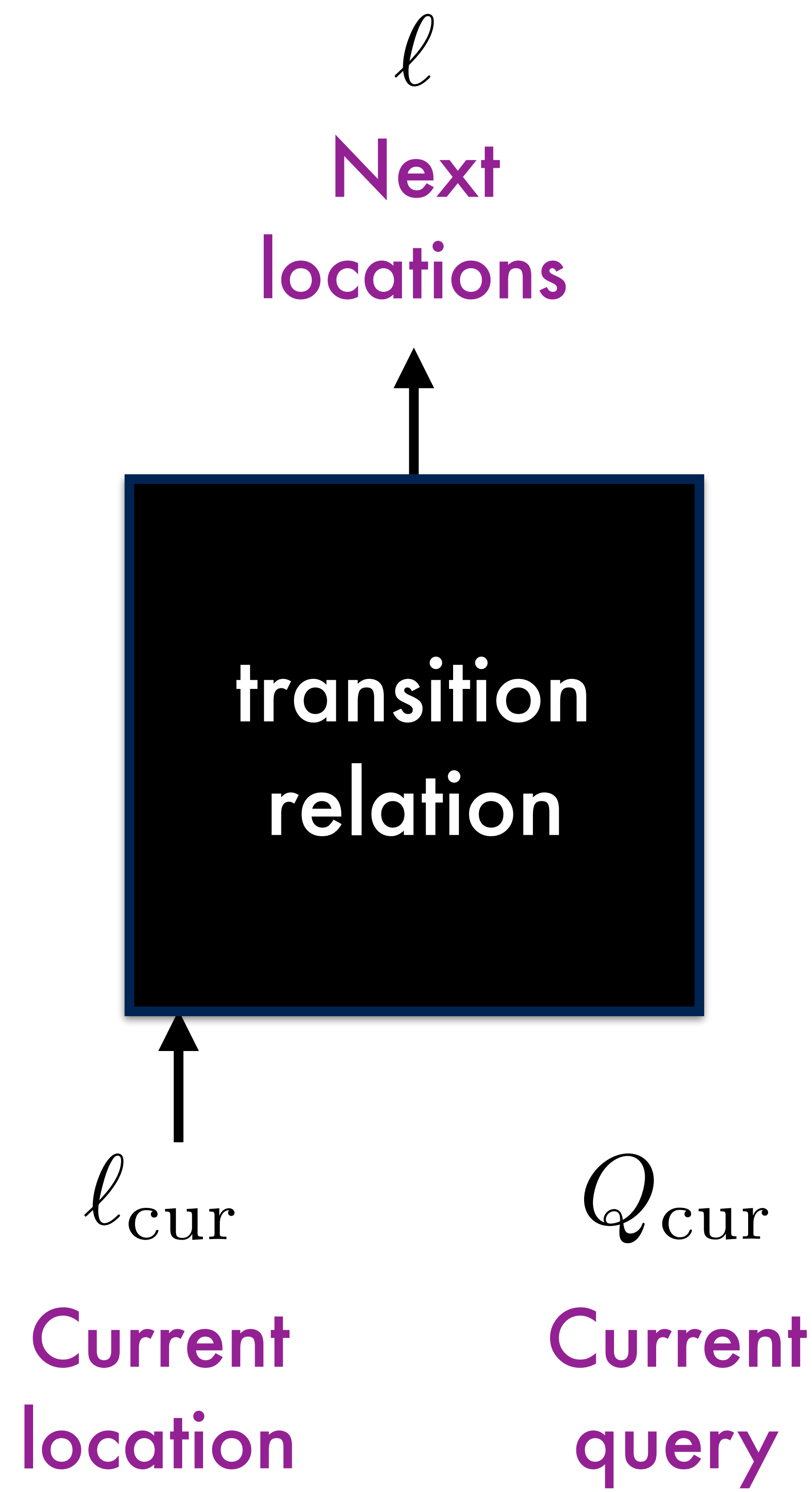
Q_{cur}

Current query

Jumping enables sparse, selective, on-the-fly control-flow abstraction

1

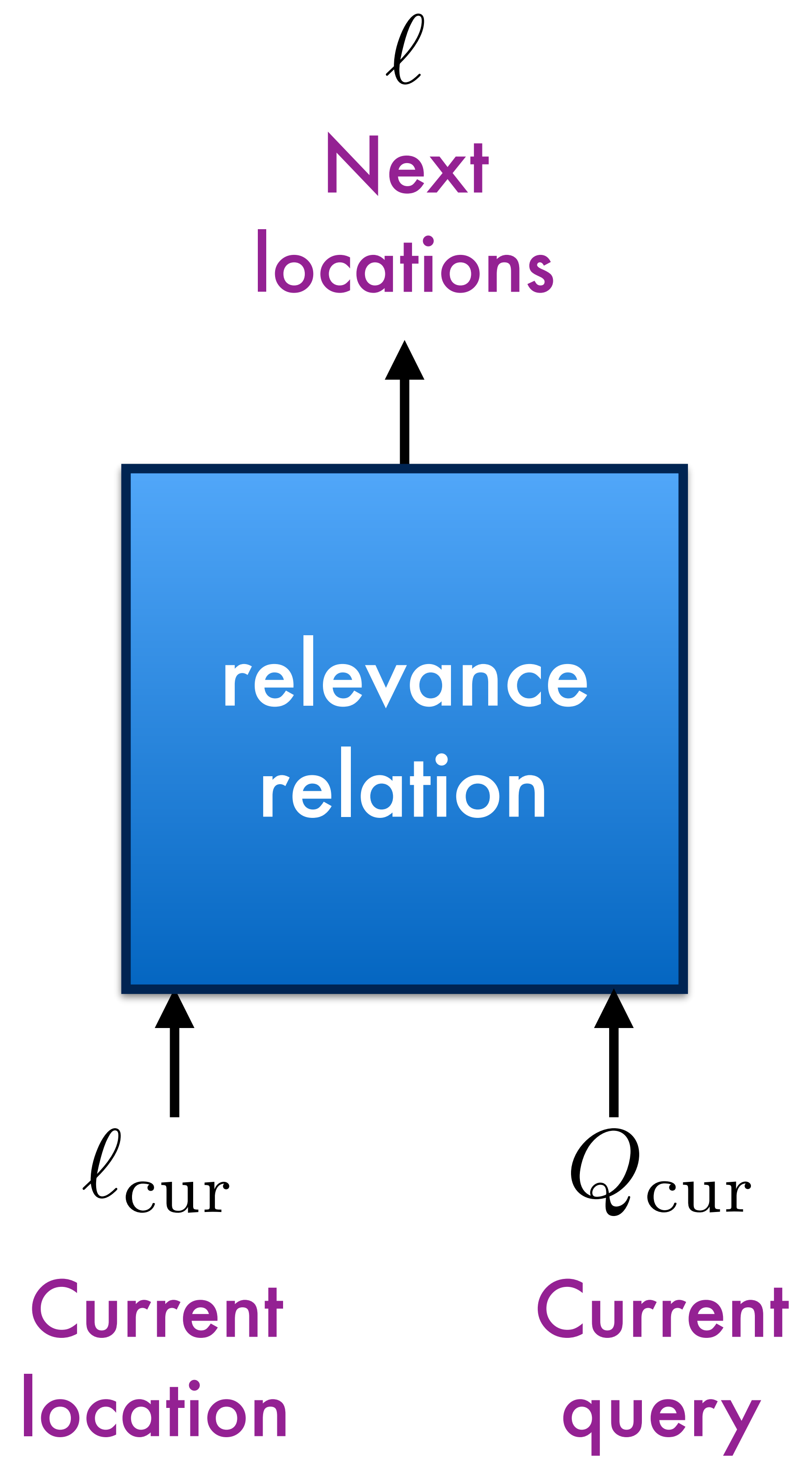
$\ell_1 \xrightarrow{[c]} \ell_2$



Jumping enables sparse, selective, on-the-fly control-flow abstraction

1

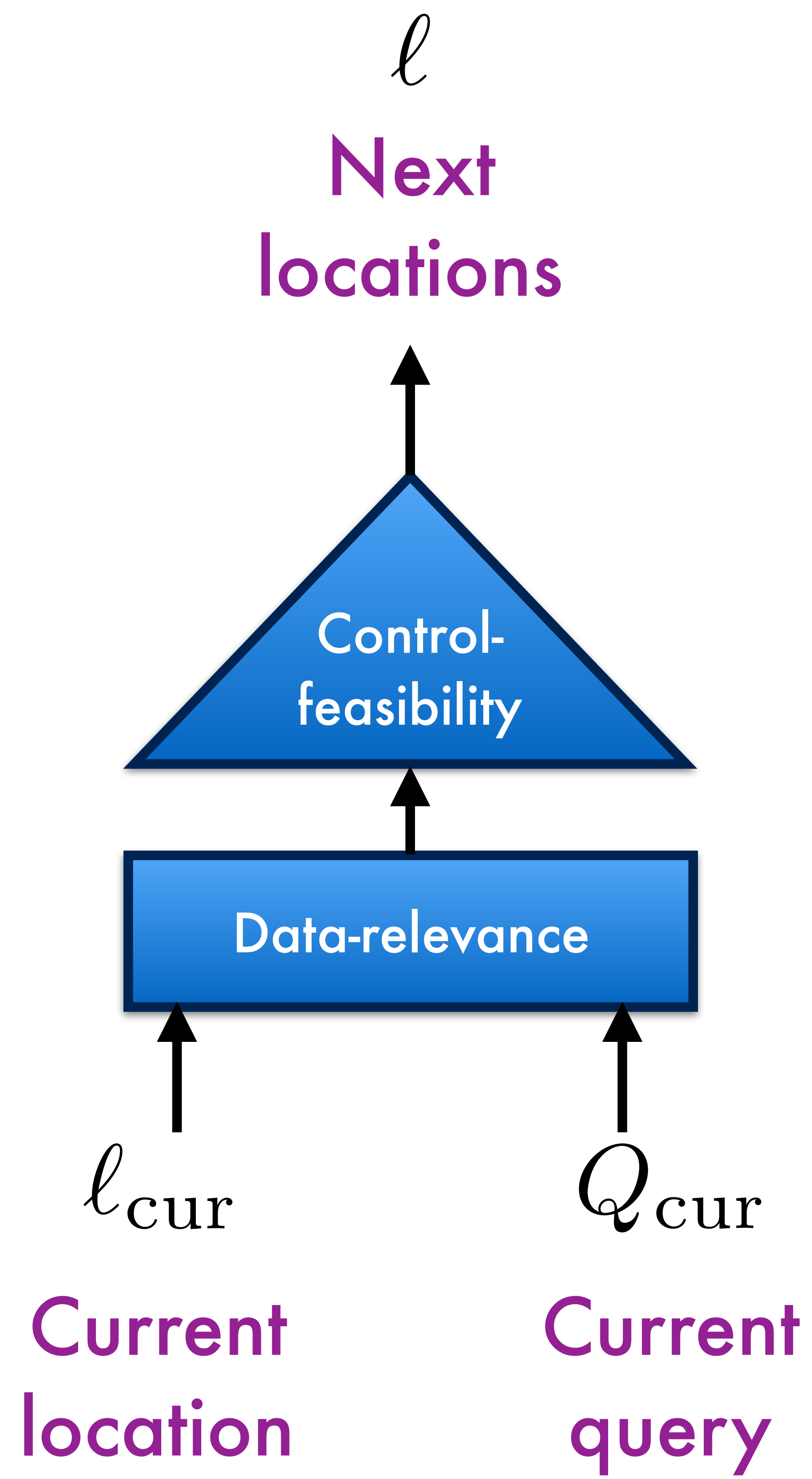
$\ell_1 \;\text{-}[c]\!\!\to \ell_2$



Jumping enables sparse, selective, on-the-fly control-flow abstraction

1

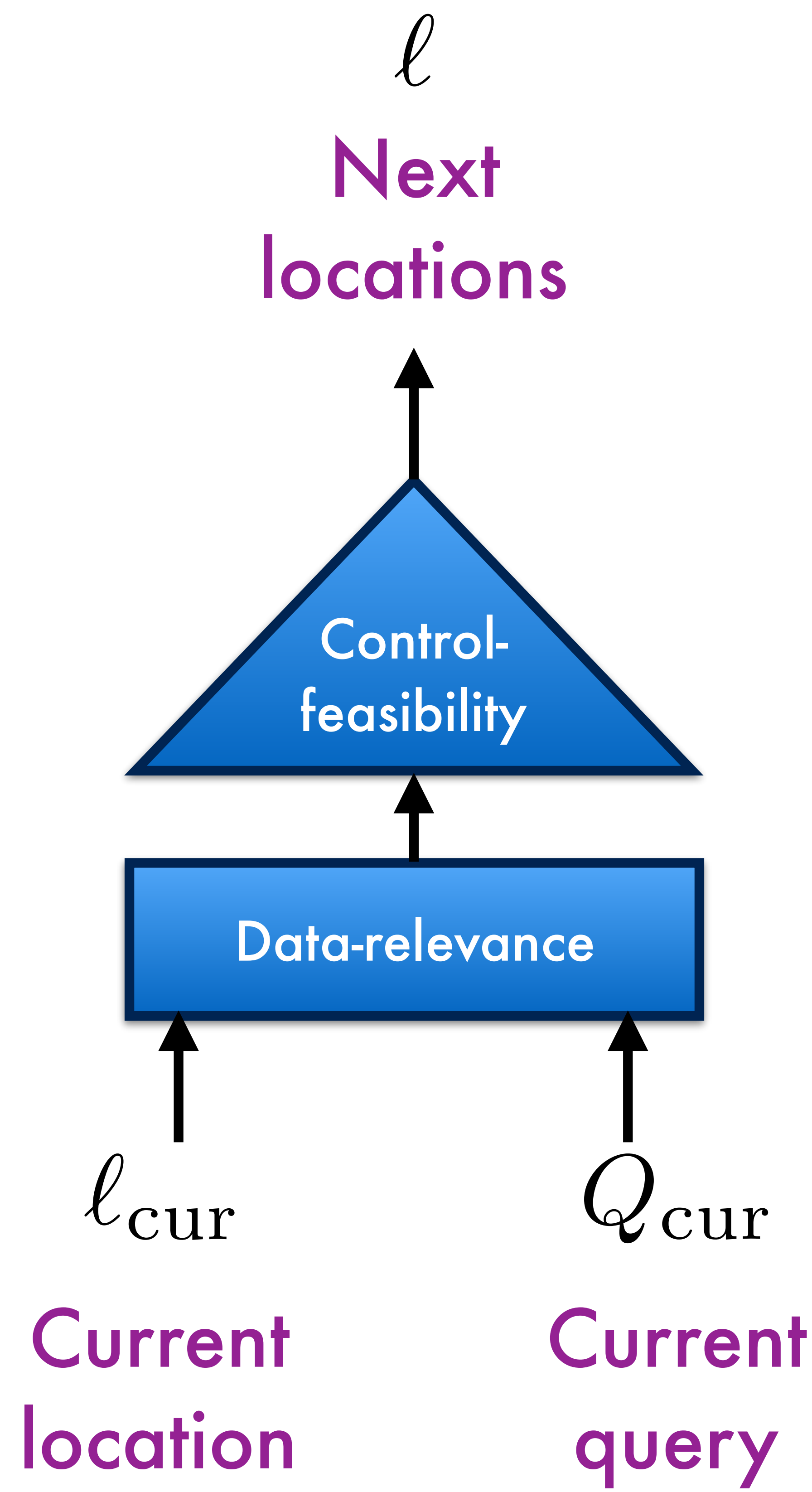
$\ell_1 \xrightarrow{[c]} \ell_2$



Jumping enables sparse, selective, on-the-fly control-flow abstraction

1 $\ell_1 \;\text{-}[c]\!\!\rightarrow\; \ell_2$

Precision	-insensitive	-sensitive
flow		
path		
context		

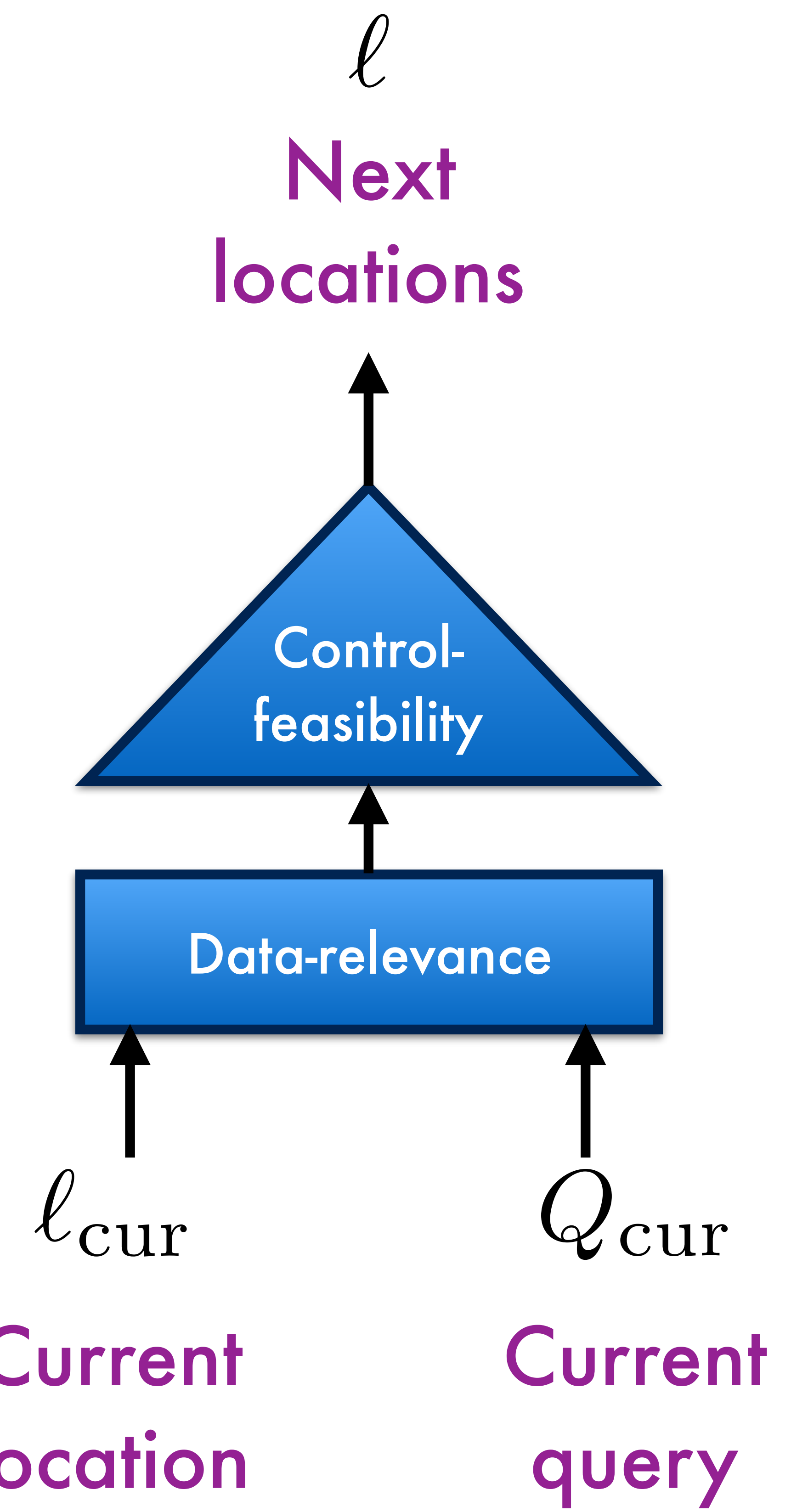


Jumping enables sparse, selective, on-the-fly control-flow abstraction

1 $\ell_1 \;\dashv\![c]\!\vdash\!\rightarrow\; \ell_2$

Precision	-insensitive	-sensitive
flow		
path		
context		

no filter



Jumping enables sparse, selective, on-the-fly control-flow abstraction

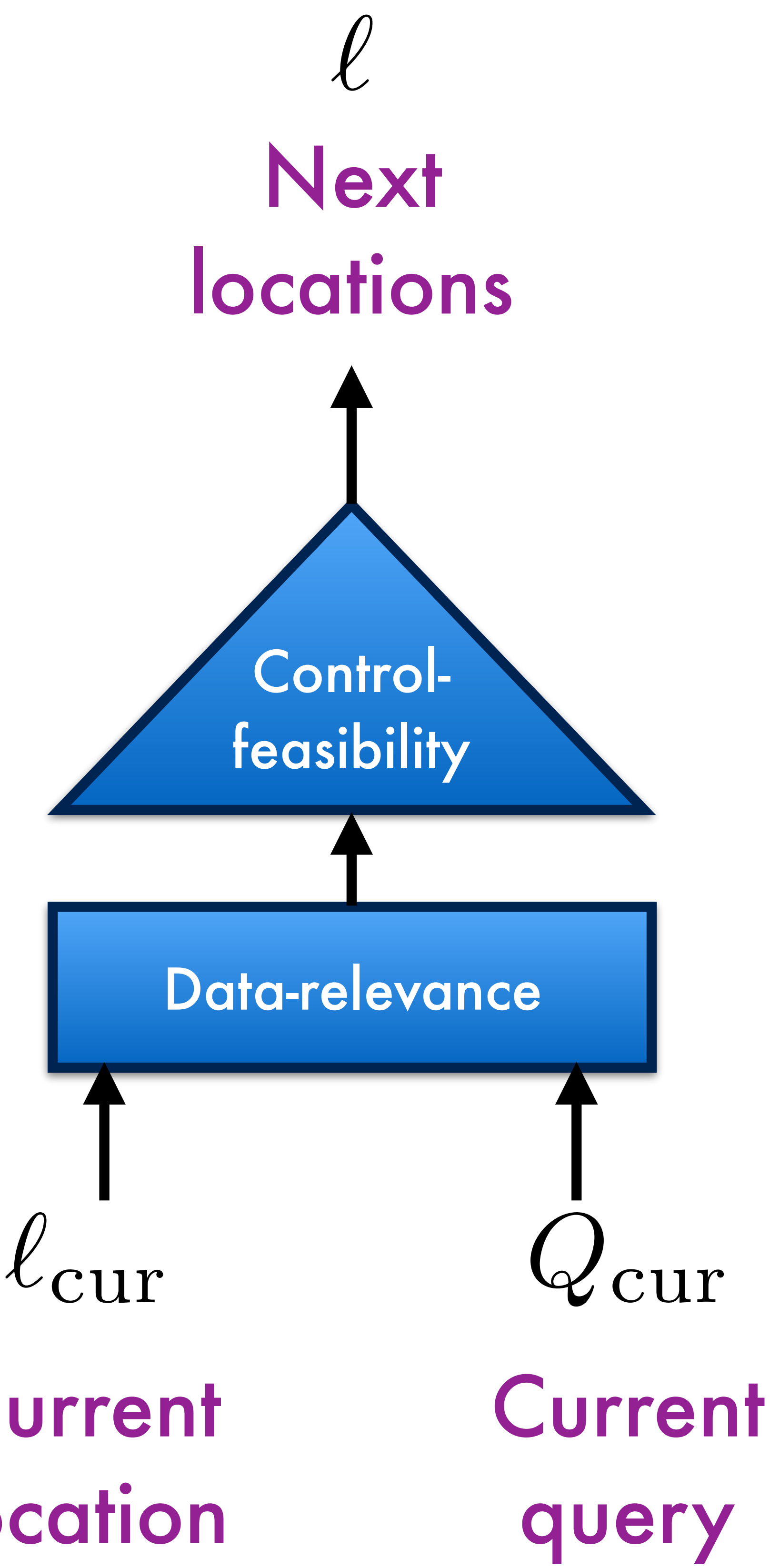
1

$\ell_1 \;\text{-}[c]\!\!\rightarrow \ell_2$

Precision	-insensitive	-sensitive
flow		
path		
context		

no filter

all locations



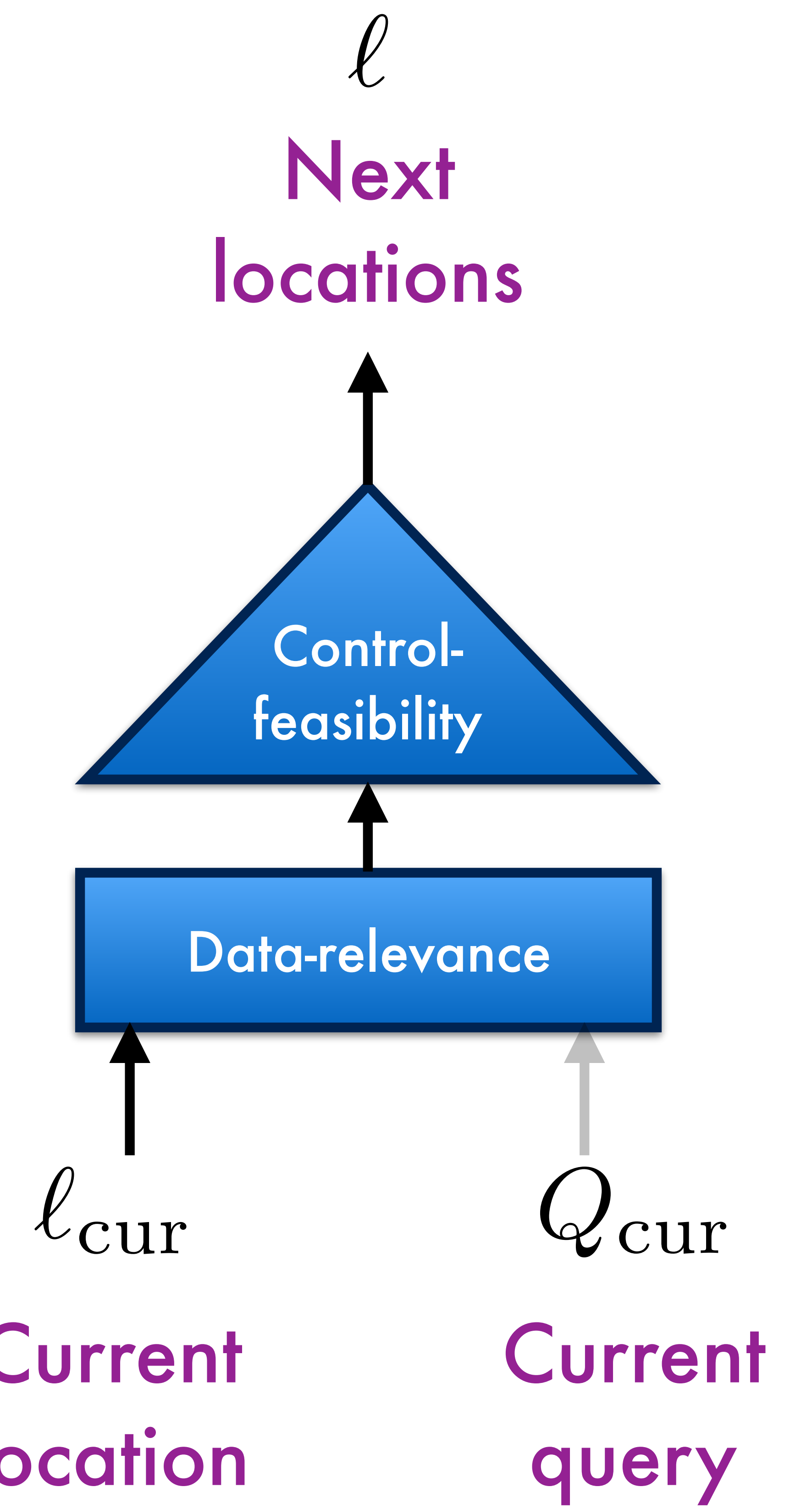
Jumping enables sparse, selective, on-the-fly control-flow abstraction

1 $\ell_1 \dashrightarrow[c] \ell_2$

Precision	-insensitive	-sensitive
flow		
path		
context		

no filter

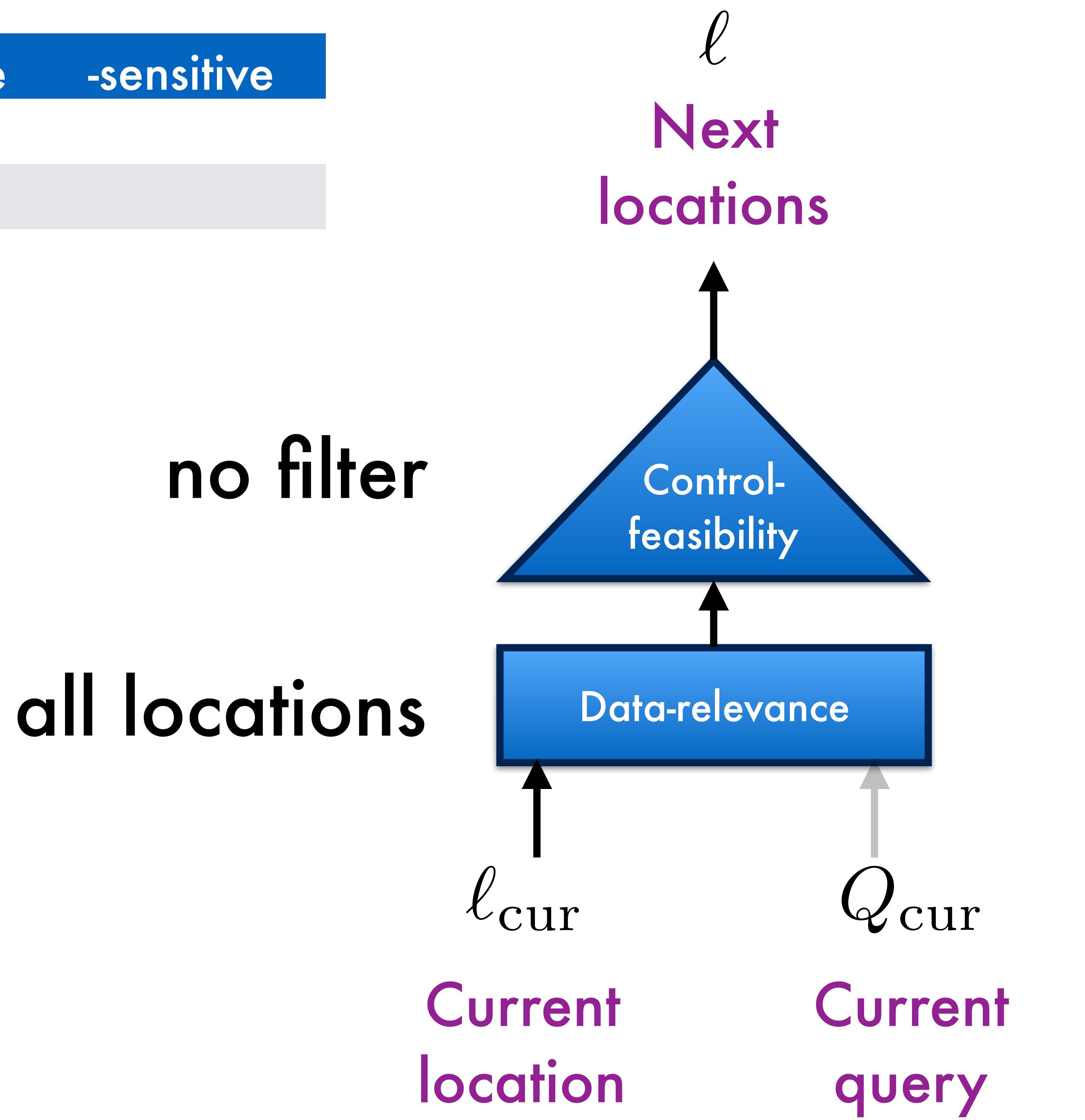
all locations



Jumping enables sparse, selective, on-the-fly control-flow abstraction

1 $\ell_1 \;\text{-}\![c]\!\!\rightarrow \ell_2$

Precision	-insensitive	-sensitive
flow	✔	
path		
context		



Jumping enables sparse, selective, on-the-fly control-flow abstraction

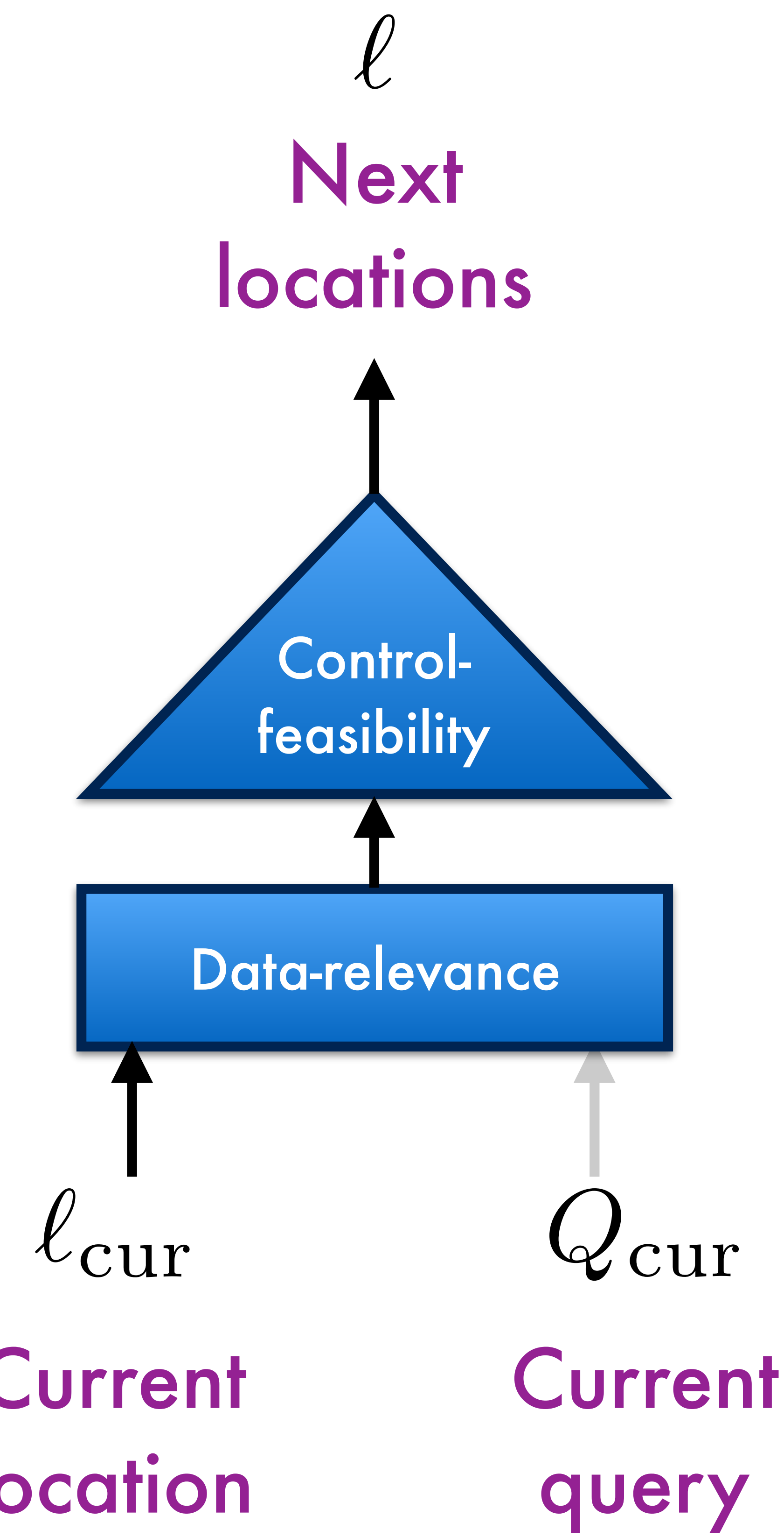
1

$\ell_1 \; \text{-}[c]\!\!\rightarrow \; \ell_2$

Precision	-insensitive	-sensitive
flow	✔	
path		
context		

only preds(ℓ_{cur})

all locations



Jumping enables sparse, selective, on-the-fly control-flow abstraction

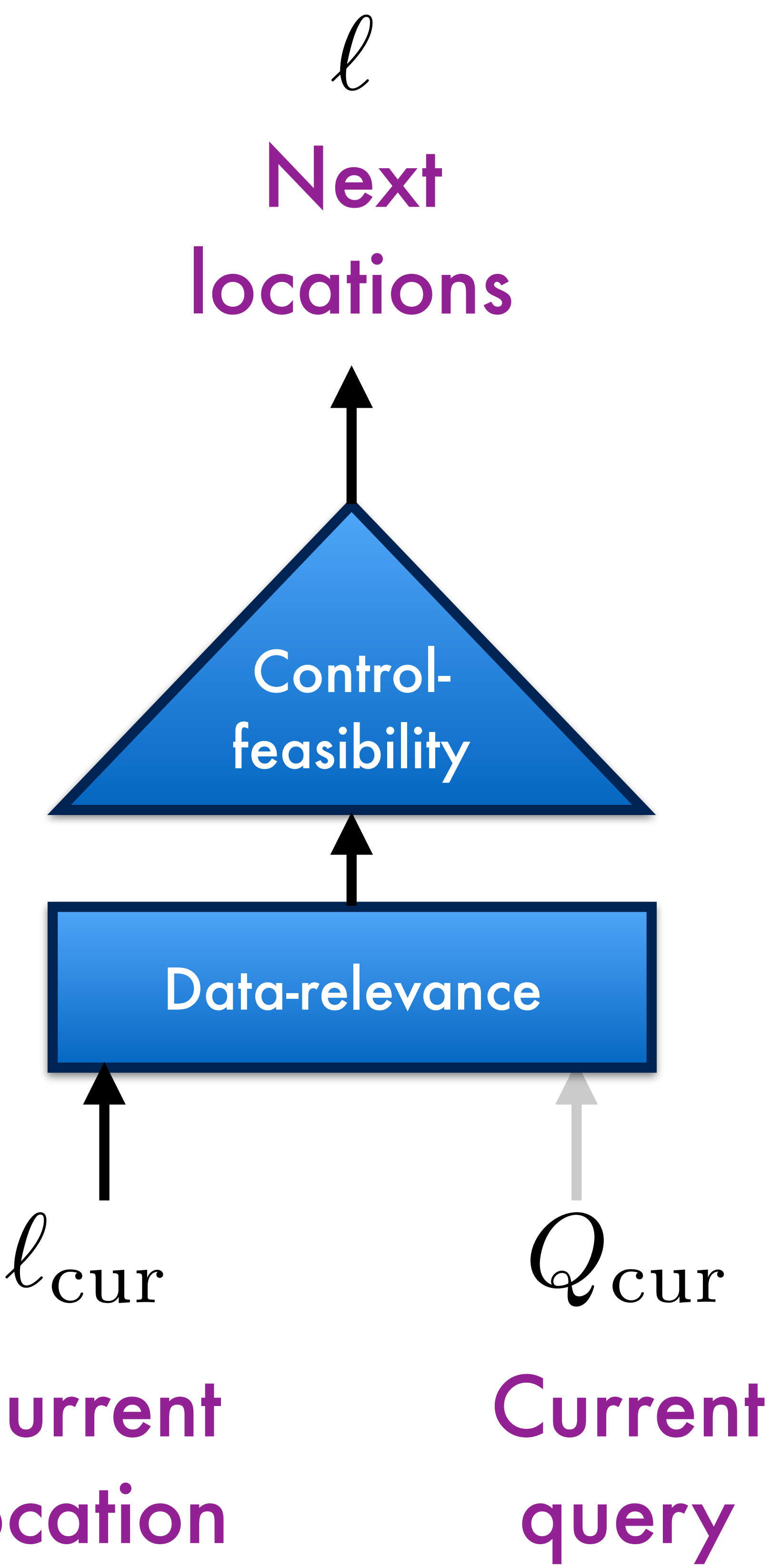
1

$$\ell_1 \xrightarrow{[c]} \ell_2$$

Precision	-insensitive	-sensitive
flow	✔	
path		
context		

only preds(ℓ_{cur})

all locations



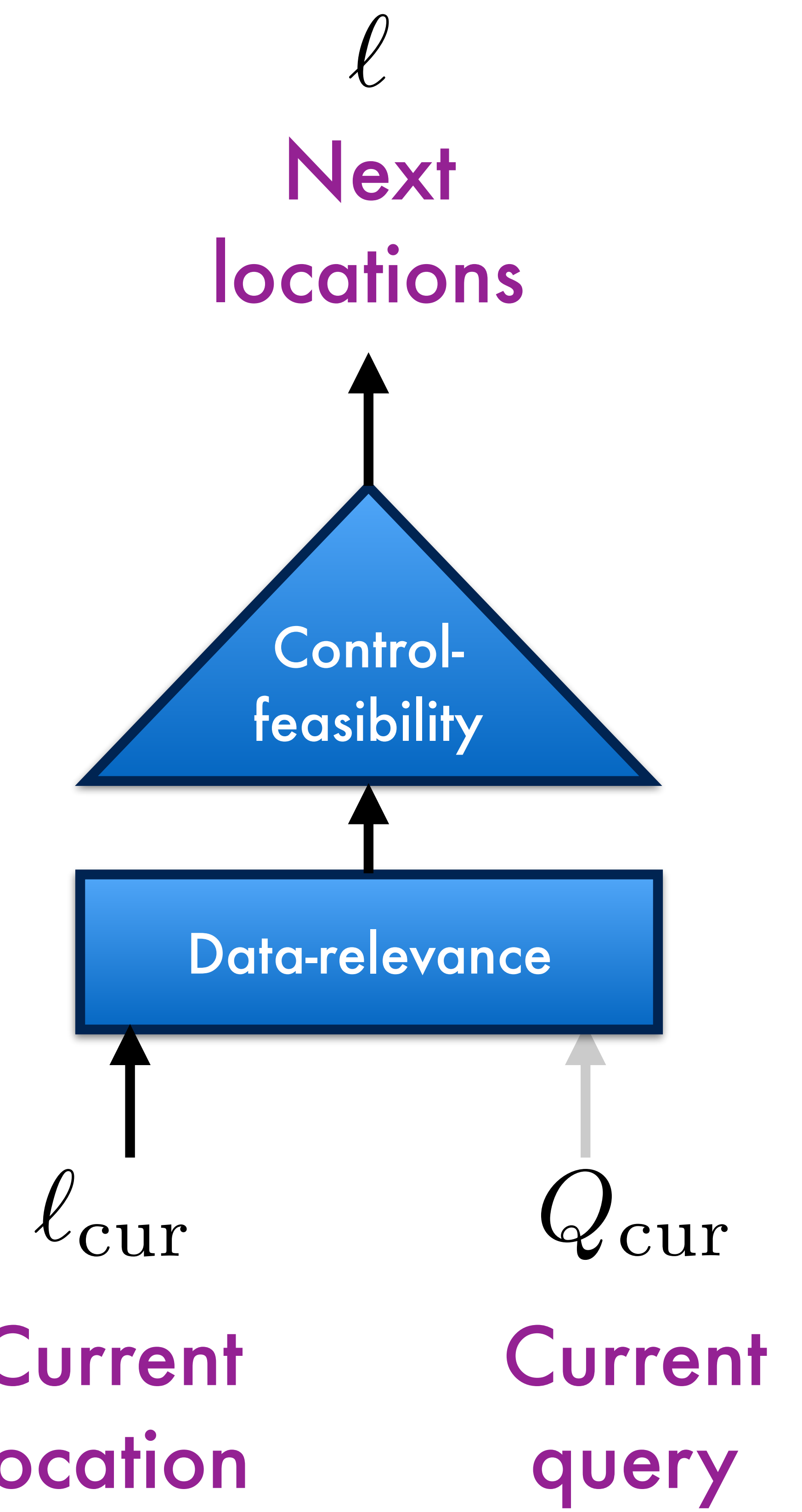
Jumping enables sparse, selective, on-the-fly control-flow abstraction

1 $\ell_1 \overset{[c]}{\longrightarrow} \ell_2$

Precision	-insensitive	-sensitive
flow	✔	✔
path		✔
context	✔	

only preds(ℓ_{cur})

all locations



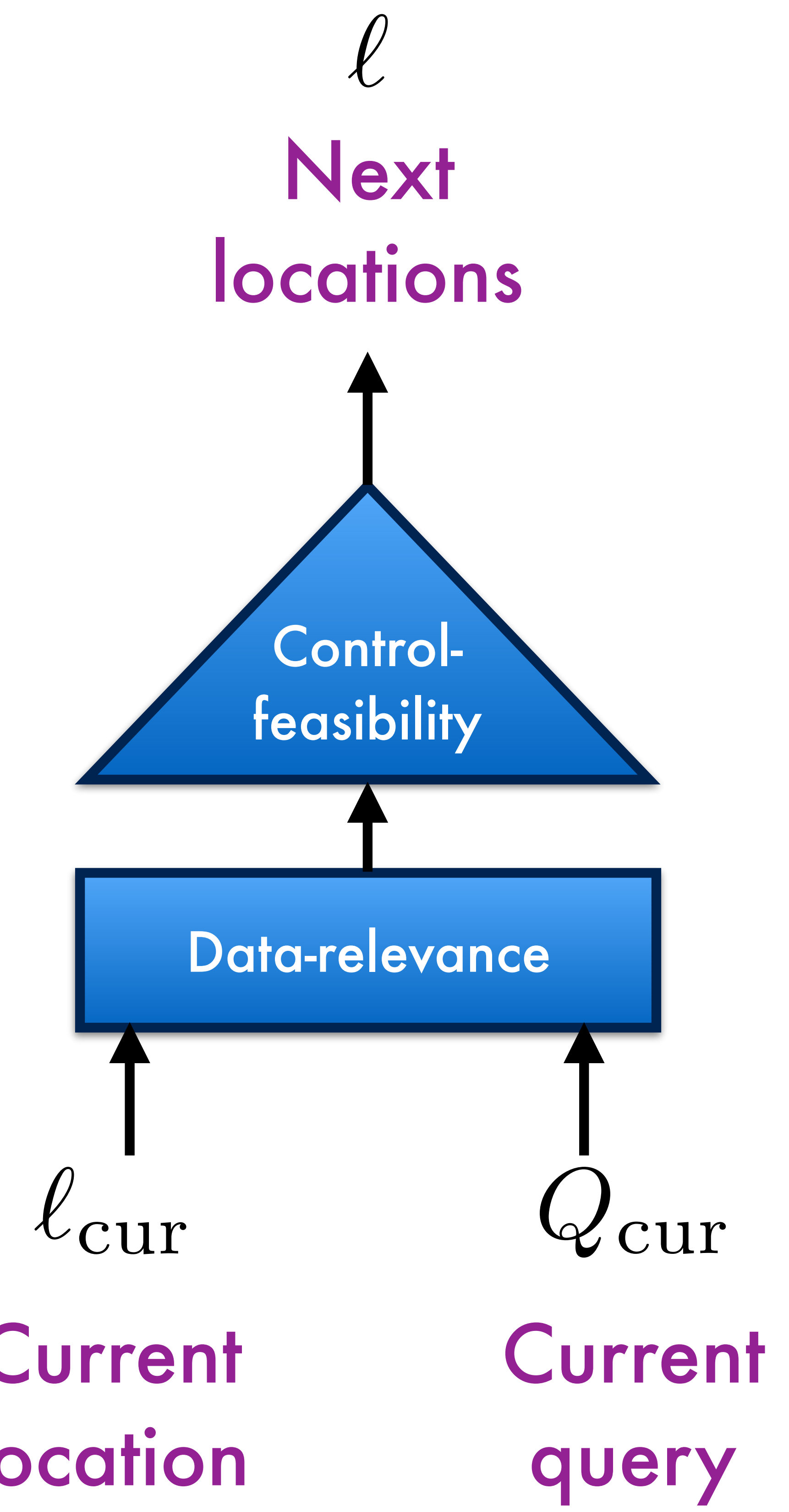
Jumping enables sparse, selective, on-the-fly control-flow abstraction

1 $\ell_1 \overset{[c]}{\longrightarrow} \ell_2$

Precision	-insensitive	-sensitive
flow	✔	✔
path		✔
context	✔	

only feasible-preds$(\ell_{\mathrm{cur}}, Q_{\mathrm{cur}})$

all locations



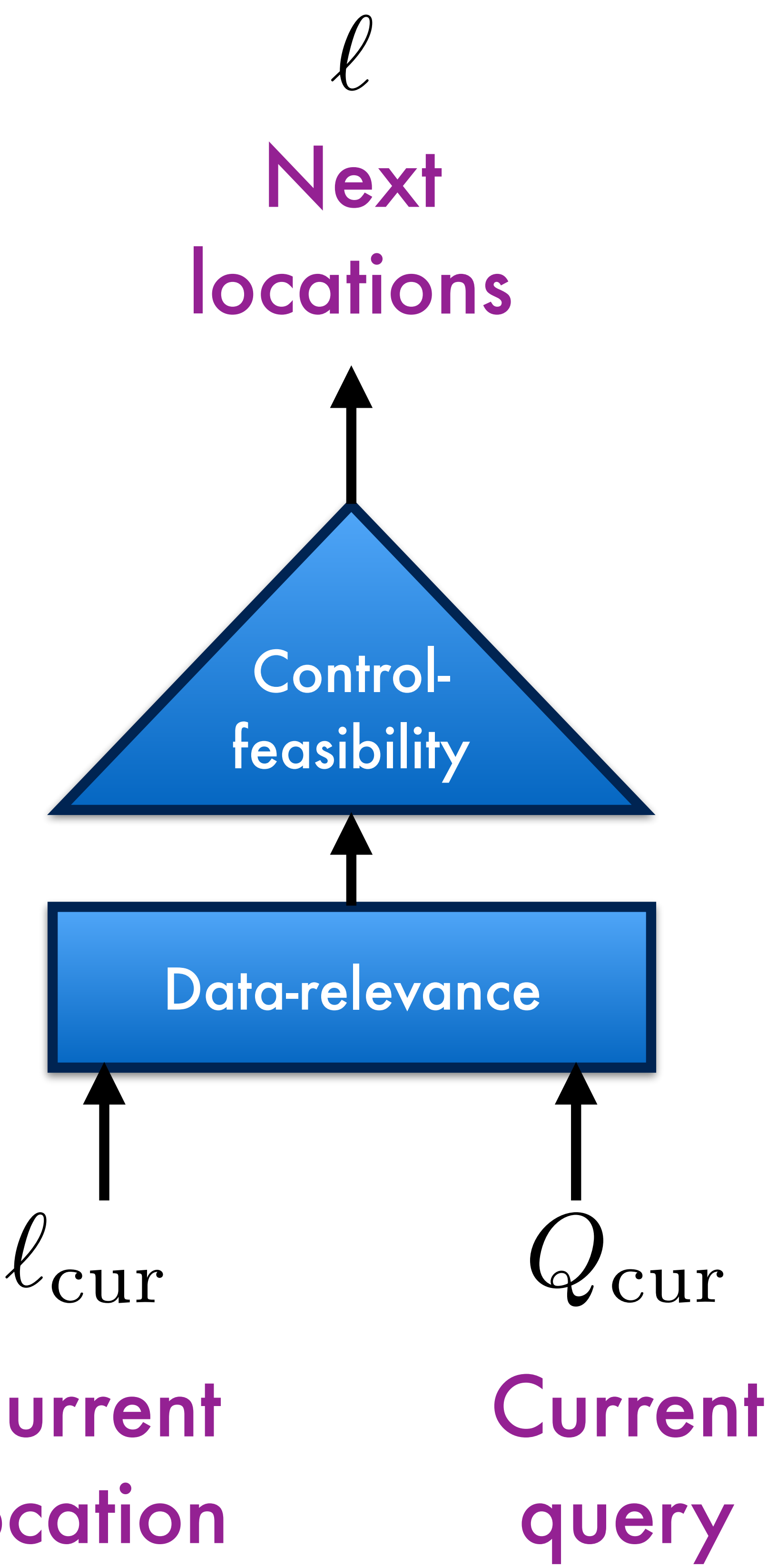
Jumping enables sparse, selective, on-the-fly control-flow abstraction

1 $\ell_1 \;\text{-}[c]\!\!\rightarrow\; \ell_2$

Precision	-insensitive	-sensitive
flow	✔	✔
path		✔
context	✔	

only feasible-preds($\ell_{\mathrm{cur}}, Q_{\mathrm{cur}}$)

all locations



Jumping enables sparse, selective, on-the-fly control-flow abstraction

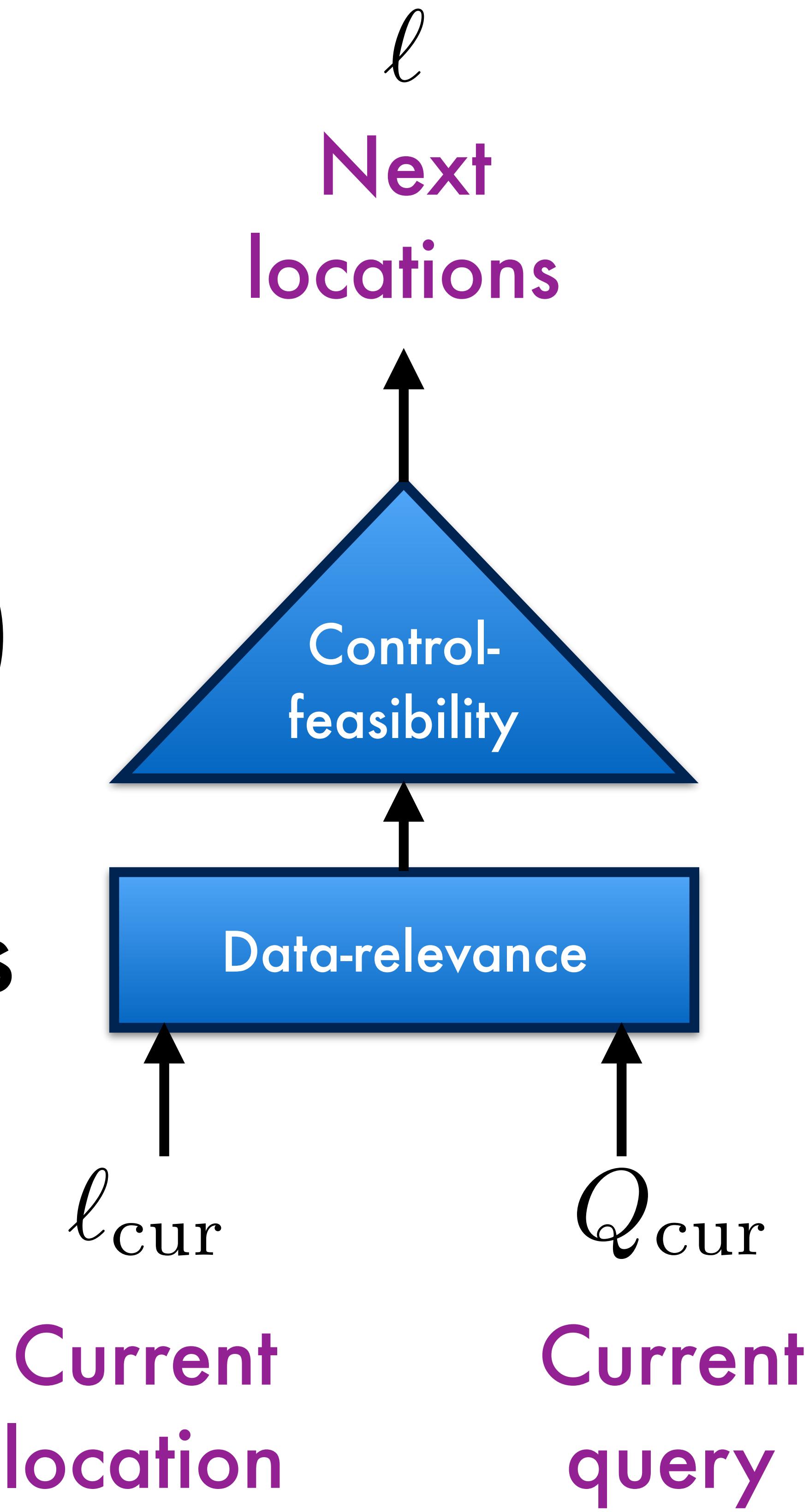
1

$\ell_1 \;\mathbin{-\!\![c]\!\!\rightarrow}\; \ell_2$

Precision	-insensitive	-sensitive
flow	✔	✔
path		✔
context	✔	✔

only feasible-preds($\ell_{\mathrm{cur}}, Q_{\mathrm{cur}}$)

all locations



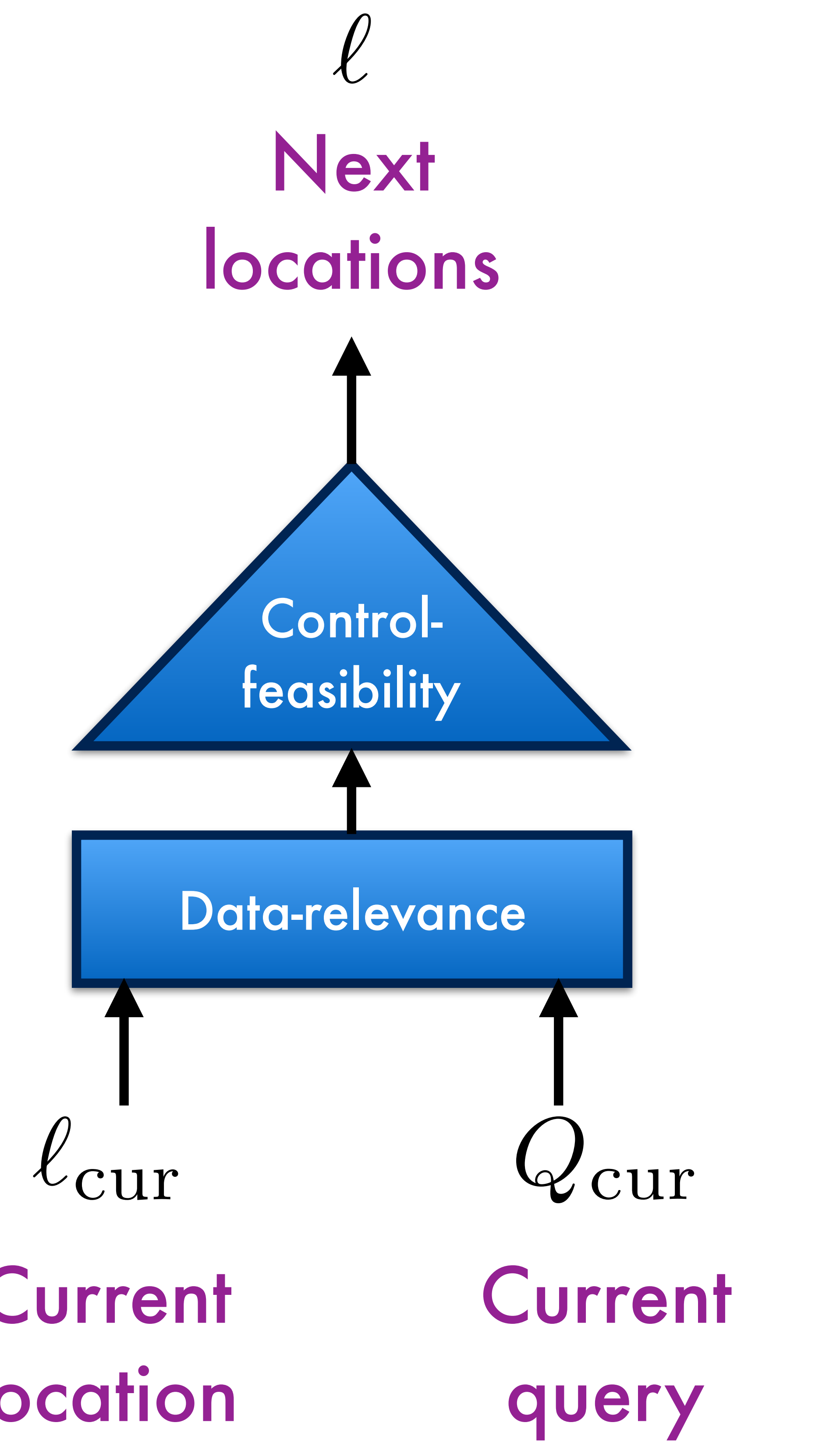
Jumping enables sparse, selective, on-the-fly control-flow abstraction

1 $\ell_1 \; \text{-}[c]\!\!\rightarrow \; \ell_2$

Precision	-insensitive	-sensitive
flow	✔	✔
path		✔
context	✔	✔

only feasible-preds($\ell_{\mathrm{cur}}, Q_{\mathrm{cur}}$)

all locations, except `assumes`



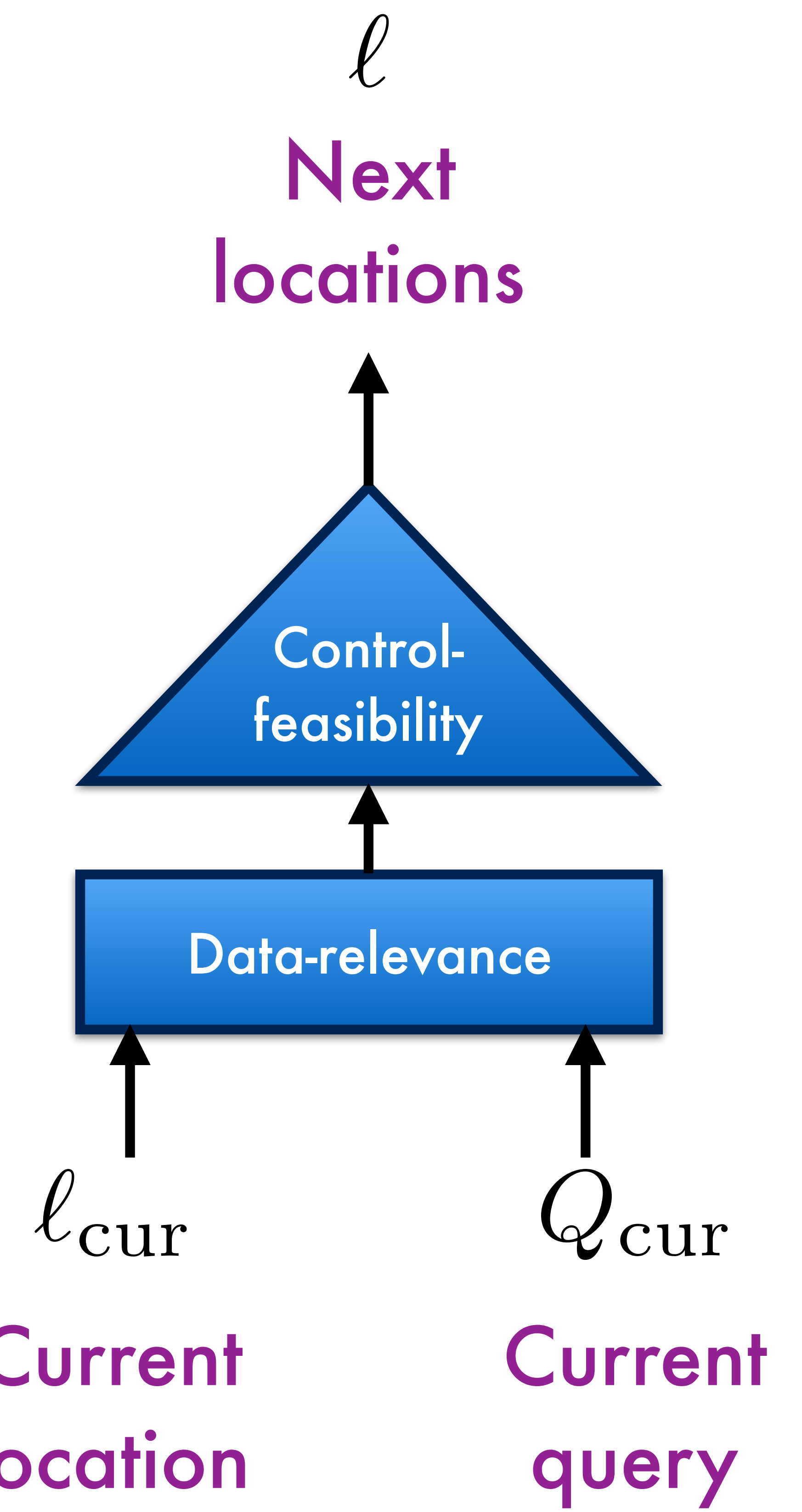
Jumping enables sparse, selective, on-the-fly control-flow abstraction

$\ell_1 \xrightarrow{[c]} \ell_2$

Precision	-insensitive	-sensitive
flow	✔	✔
path		✔
context	✔	✔

only feasible-preds($\ell_{\mathrm{cur}}, Q_{\mathrm{cur}}$)

all locations, except `assumes`



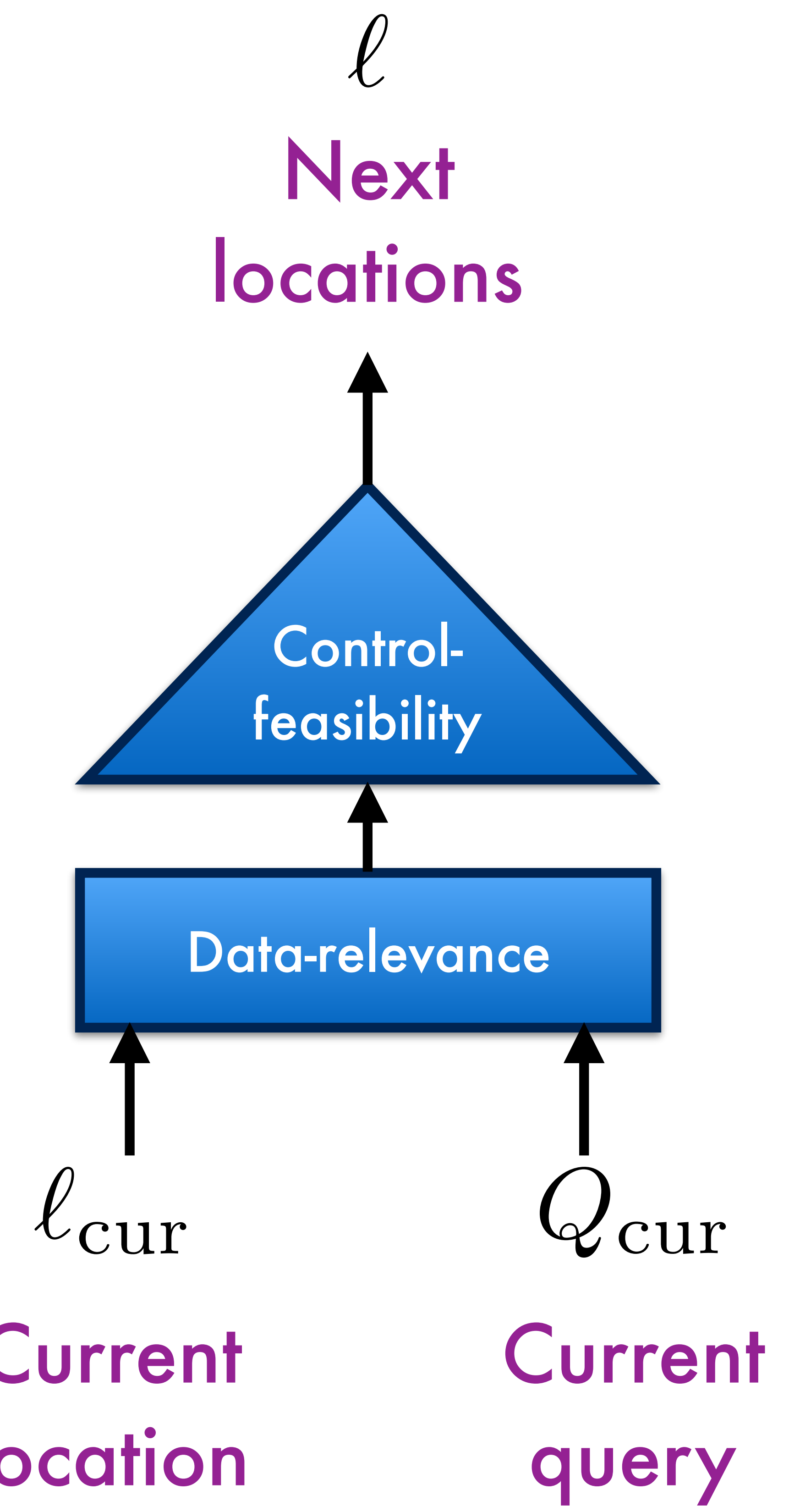
Jumping enables sparse, selective, on-the-fly control-flow abstraction

1 $\ell_1 \xrightarrow{[c]} \ell_2$

Precision	-insensitive	-sensitive
flow	✔	✔
path	✔	✔
context	✔	✔

only feasible-preds($\ell_{\mathrm{cur}}, Q_{\mathrm{cur}}$)

all locations, except `assumes`



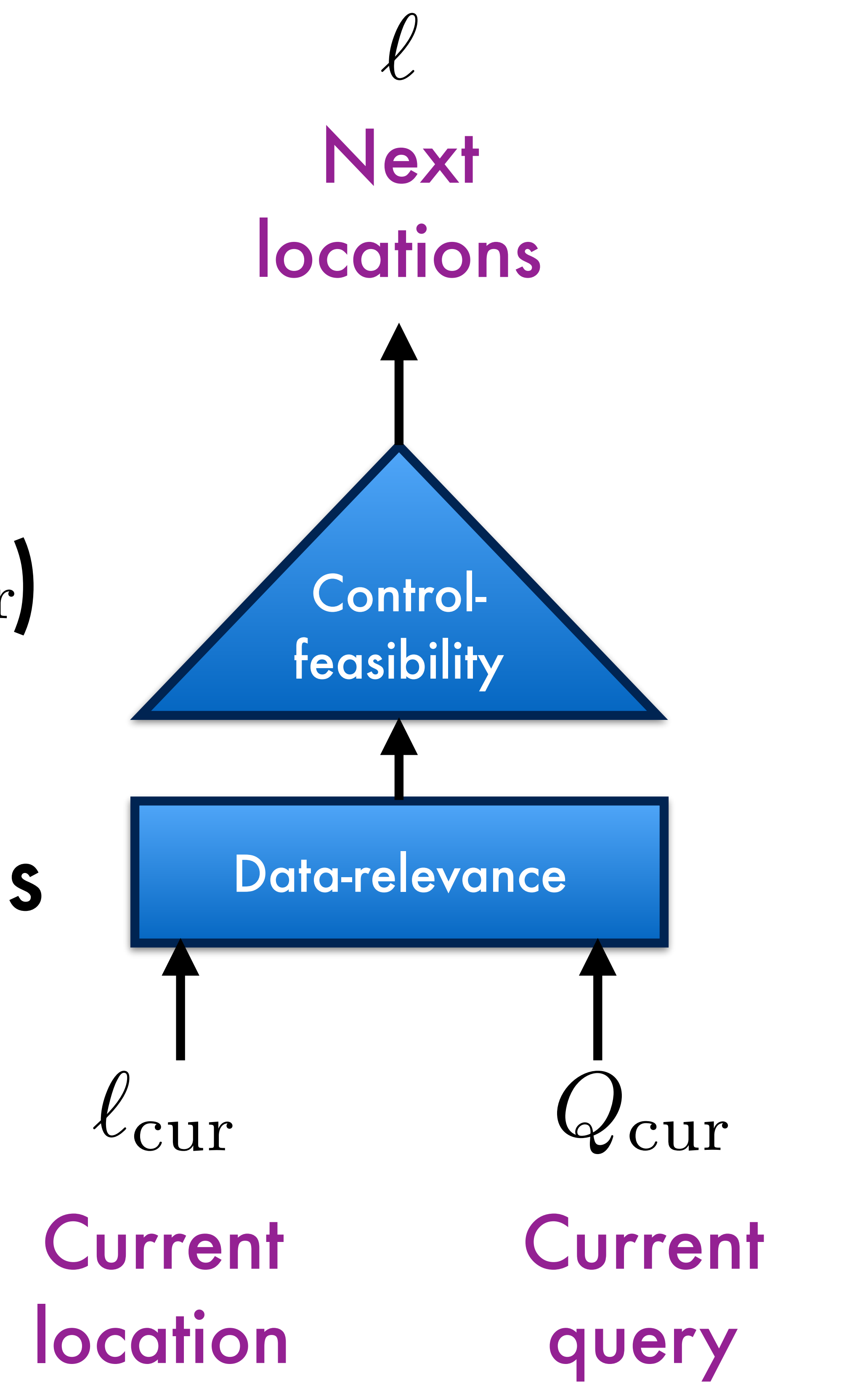
Jumping enables sparse, selective, on-the-fly control-flow abstraction

1

$\ell_1 \dashv\!\![c]\!\!\mapsto \ell_2$

only feasible-preds($\ell_{\mathrm{cur}}, Q_{\mathrm{cur}}$)

all locations



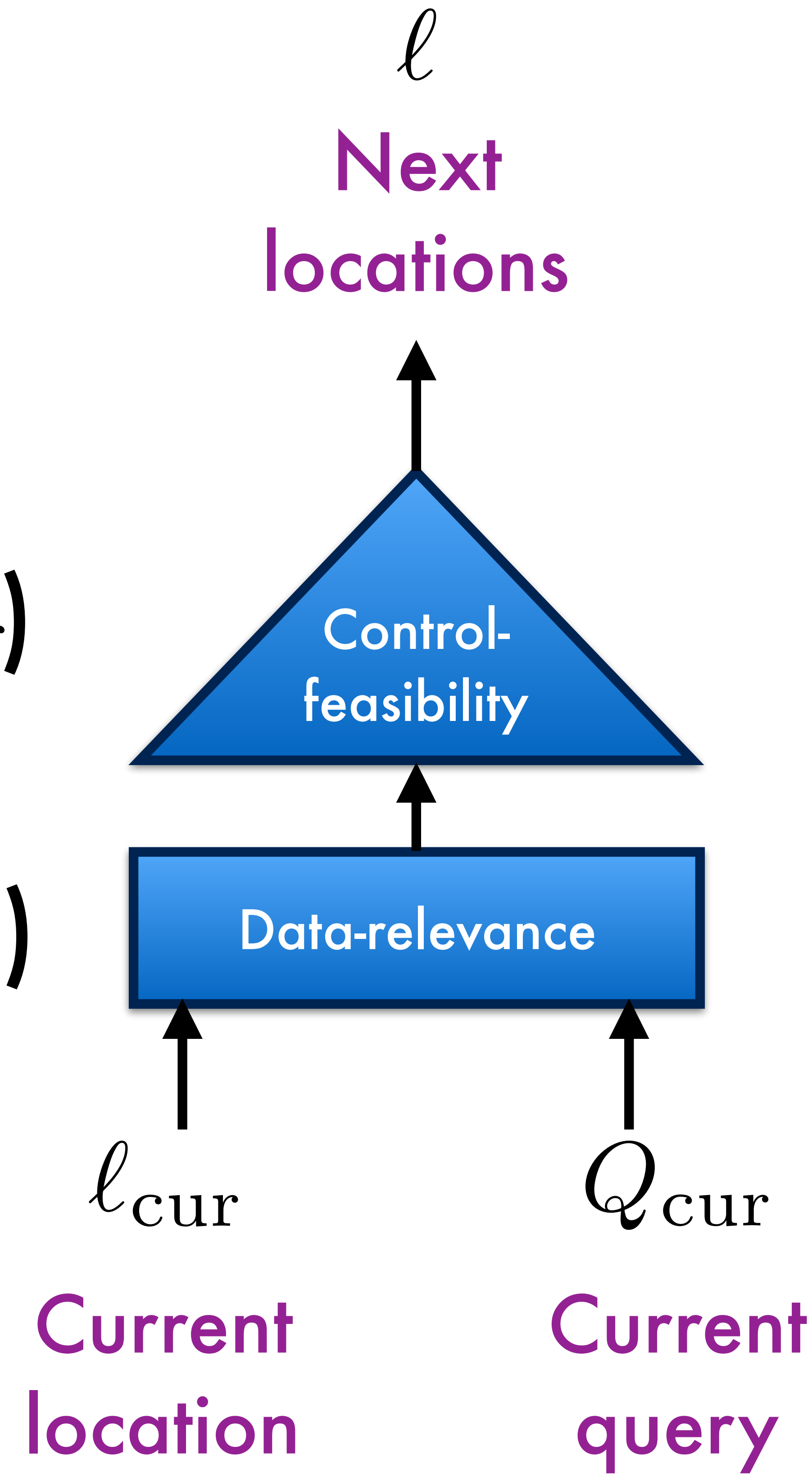
Jumping enables sparse, selective, on-the-fly control-flow abstraction

1

$\ell_1 \xrightarrow{[c]} \ell_2$

only feasible-preds($\ell_{\mathrm{cur}}, Q_{\mathrm{cur}}$)

all mods(Q_{cur})



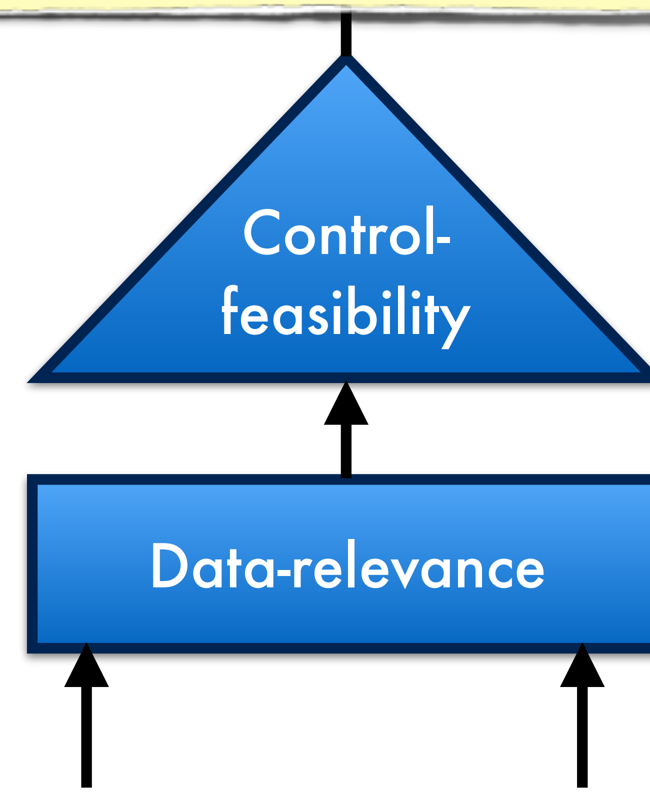
Jumping enables sparse, selective, on-the-fly (1) control-flow abstraction

$\ell_1 \;\dashv[c]\!\!\rightarrow\; \ell_2$

Be sparse by using data relevance with desired control-flow abstraction

only feasible-preds($\ell_{\mathrm{cur}}, Q_{\mathrm{cur}}$)

all mods(Q_{cur})



ℓ_{cur} — Current location

Q_{cur} — Current query

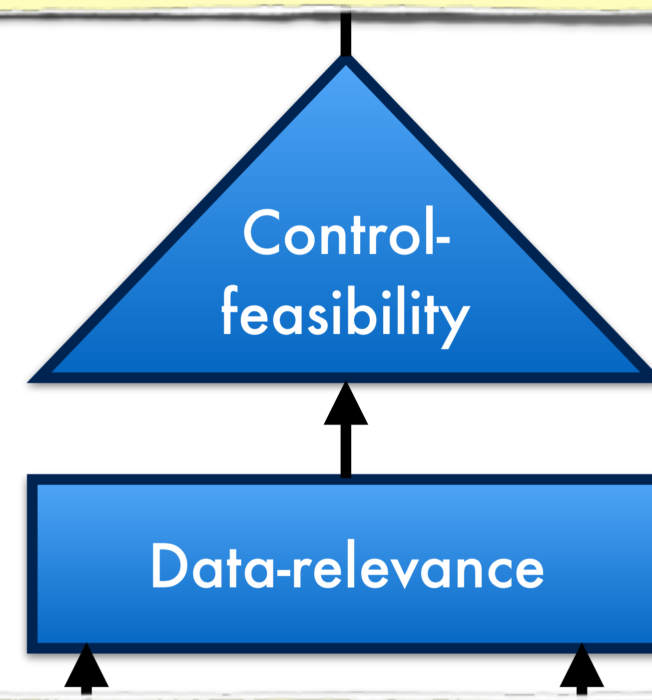
Jumping enables sparse, selective, on-the-fly control-flow abstraction

1 $\ell_1 \;-\!\![c]\!\!\rightarrow\; \ell_2$

Be sparse by using data relevance with desired control-flow abstraction

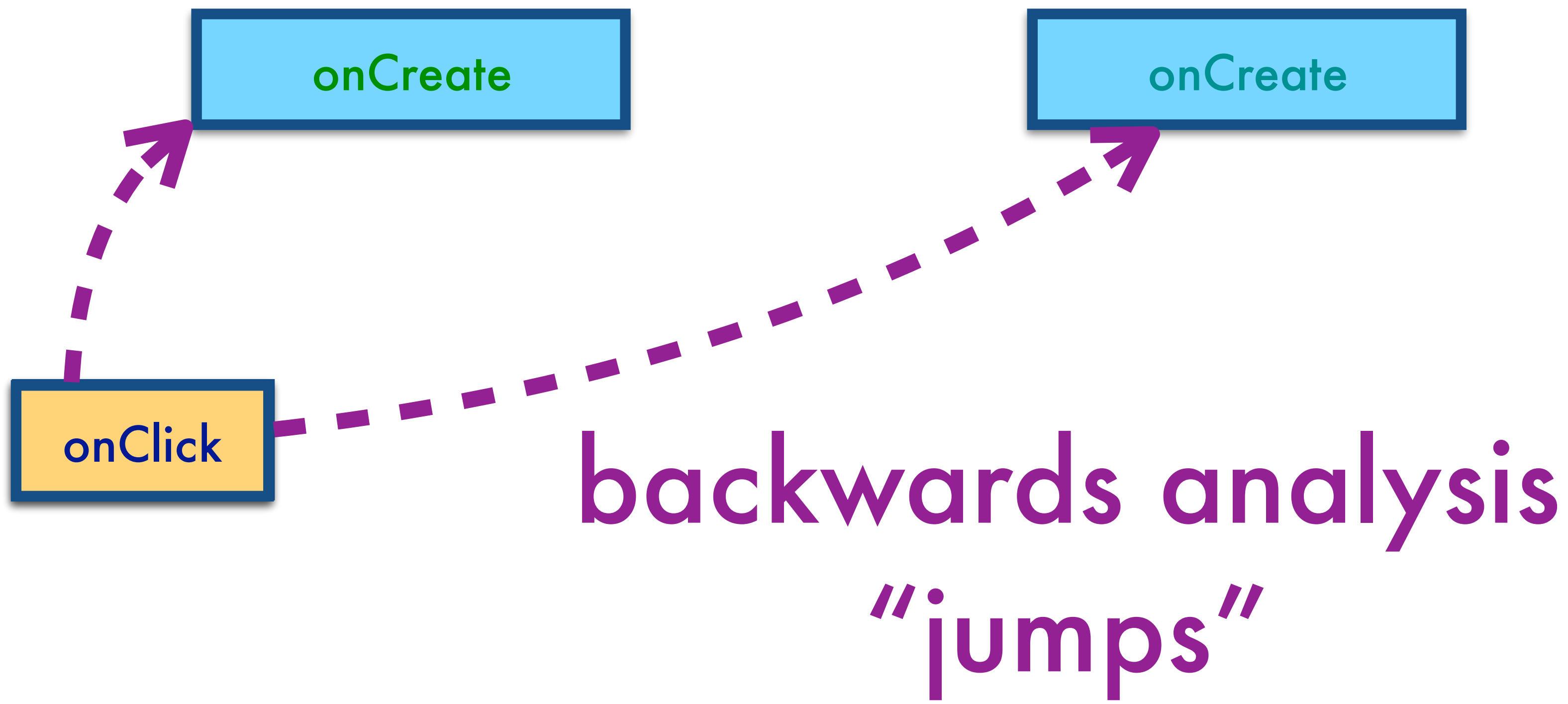
only feasible-preds($\ell_{\mathrm{cur}}, Q_{\mathrm{cur}}$)

all mods(Q_{cur})



Be selective by varying the relevance relation at each analysis step

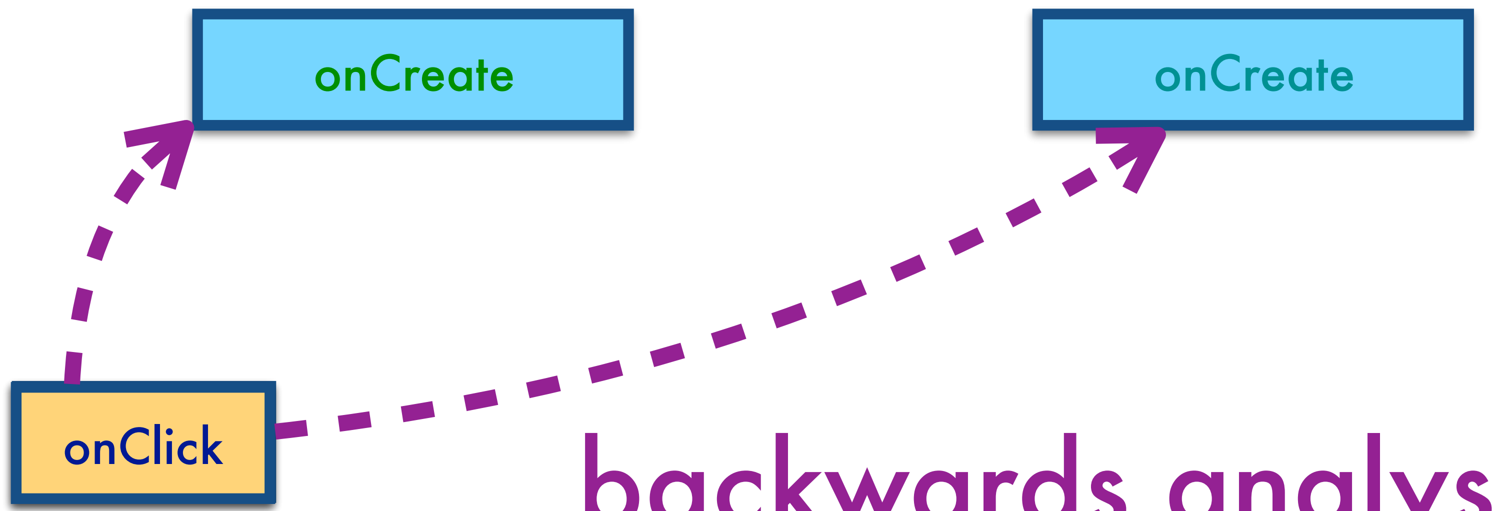
Contributions



1 $\ell_1 \;\dashv\!\![c]\!\!\rightarrow\; \ell_2$ Framework for jumping analyses

Applied to Android events

Contributions

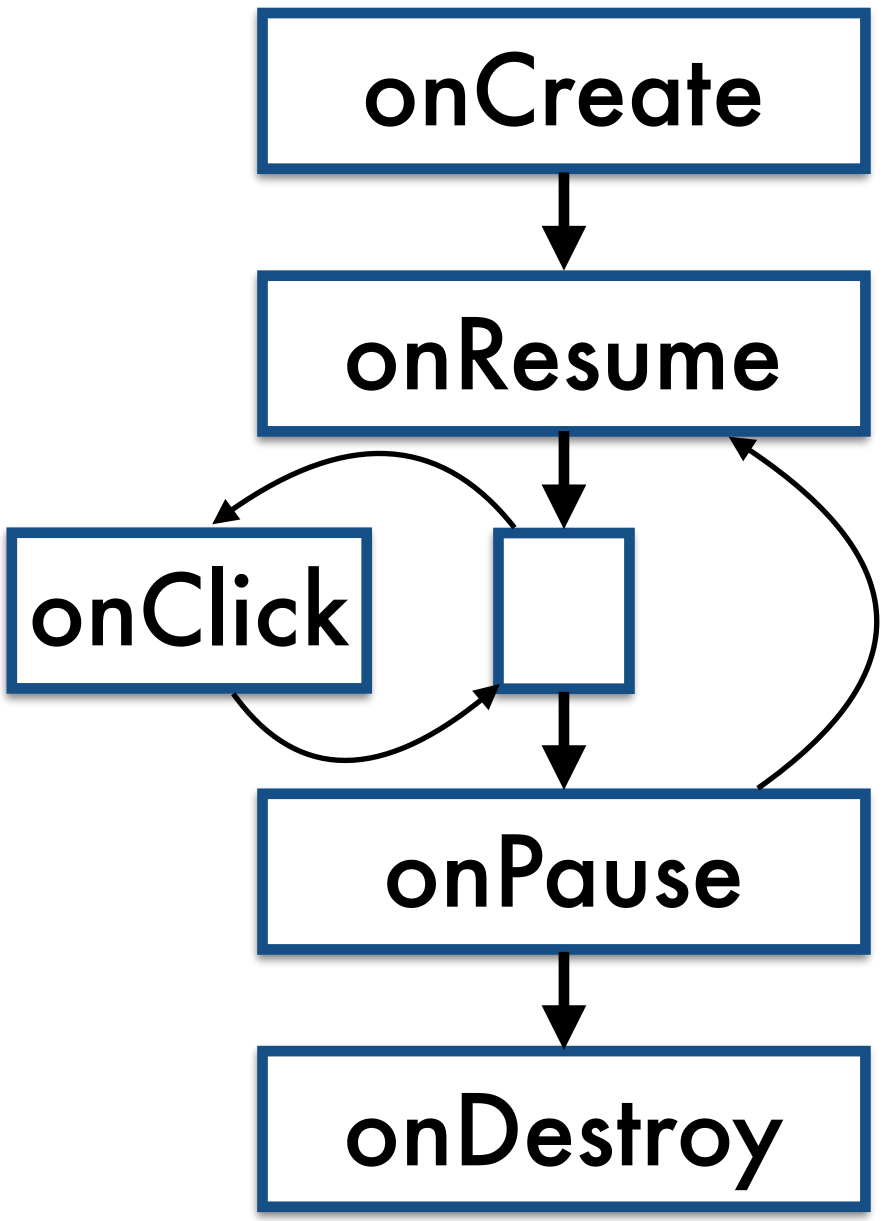


backwards analysis
"jumps"

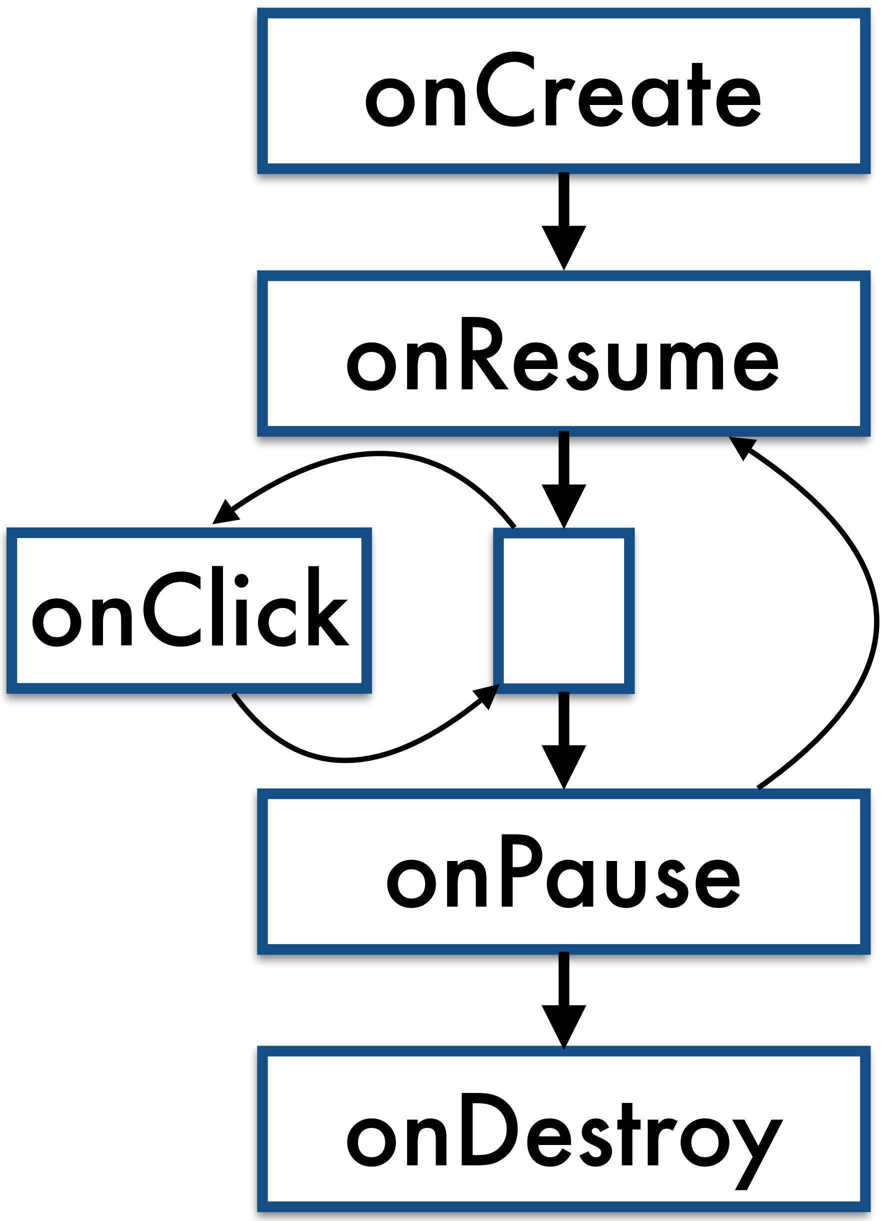
1 $\ell_1 \;{-}[c]{\rightarrow}\; \ell_2$ Framework for jumping analyses

2 Applied to Android events

Formalizing lifecycle graphs

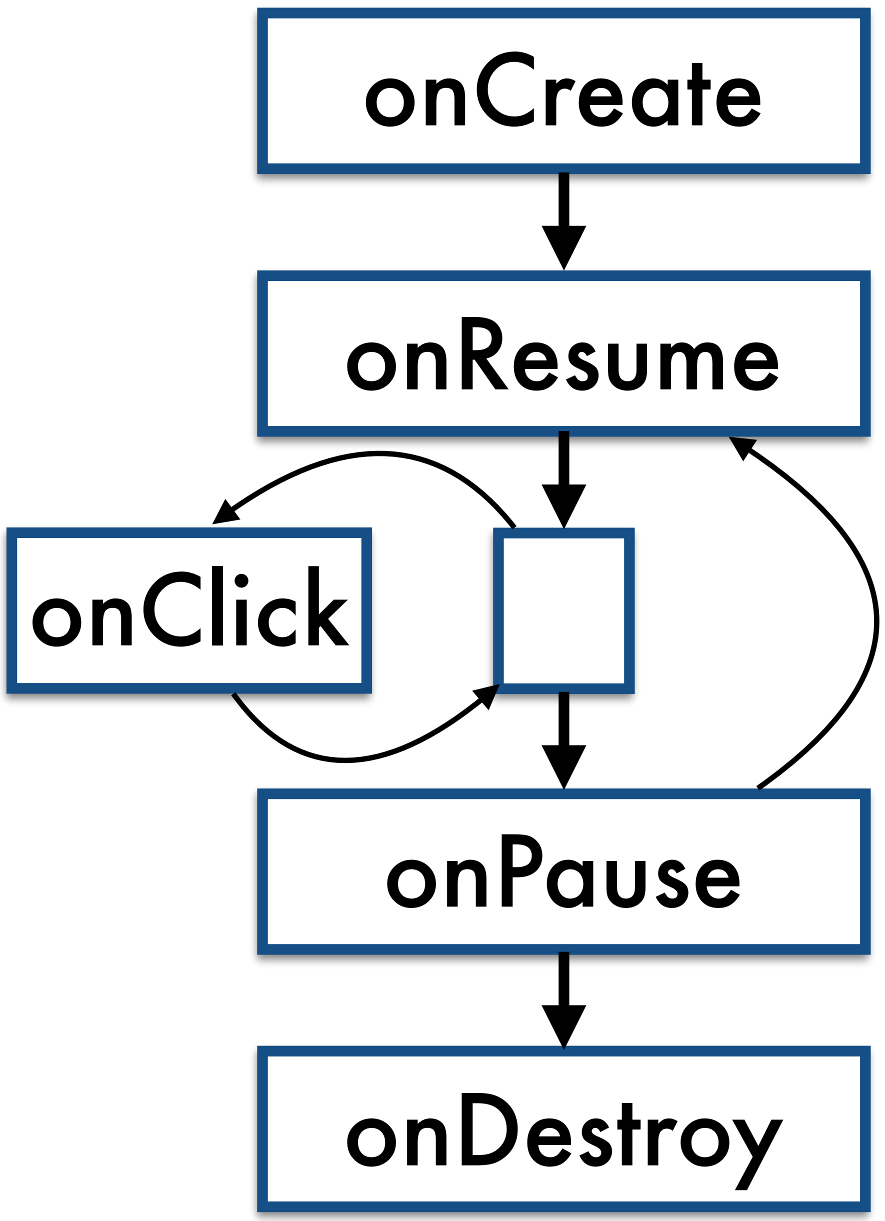


Formalizing lifecycle graphs



"Lifecycle automata" accepts the concrete trace of callbacks projected onto its transitions

Formalizing lifecycle graphs



"Lifecycle automata" accepts the concrete trace of callbacks projected onto its transitions

Further scalability challenge: Lifecycle spec per class but analysis applies per object

A jumping policy for Android analysis

A jumping policy for Android analysis

Within an event-callback (intra-event), follow predecessor transitions

no jumping, path/context-sensitive

A jumping policy for Android analysis

Within an event-callback (intra-event), follow predecessor transitions

no jumping, path/context-sensitive

Between event-callbacks (inter-event), jump using lifecycle graphs for control-feasibility filtering

Hypothesis: Jumping is an effective approach to path-sensitive, inter-event analysis

Setup: Proving dereferences safe

Setup: Proving dereferences safe

10 open source Android apps

- 3,000 to 55,000 lines of code
- 10 to 100 components
- 120 to 1,320 callbacks

Setup: Proving dereferences safe

10 open source Android apps

3,000 to 55,000 lines of code
10 to 100 components
120 to 1,320 callbacks

Event product graph would have 10^{10} to 10^{111} nodes (with one instance per class)

Setup: Proving dereferences safe

10 open source Android apps

3,000 to 55,000 lines of code
10 to 100 components
120 to 1,320 callbacks

Setup: Proving dereferences safe

10 open source Android apps

- 3,000 to 55,000 lines of code
- 10 to 100 components
- 120 to 1,320 callbacks

Compared 3 analyses

- **Nit**: flow-insensitive
- **Thr**esher: no jumping
- **Hop**per: jumping

Is jumping effective?

	KLOC	Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Is jumping effective?

	KLOC	Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
[illegible]	[illegible]	5921	3643	698	330 (53)
[illegible]	[illegible]	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Many derefs

Is jumping effective?

	KLOC	unproven derefs Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Is jumping effective?

Flow-insensitive

No jumping

Jumping

unproven derefs

	KLOC	Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Is jumping effective?

	KLOC	Deref	Nit	Thr	Hop (Impr %)
		unproven derefs			
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Is jumping effective?

		unproven derefs			
	KLOC	Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Thr and Hop: 10 second budget per deref

Is jumping effective?

	KLOC	Deref	Nit	Thr	Hop (Impr %)
		unproven derefs			
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Is jumping effective?

	KLOC	unproven derefs Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Thresher has 74% fewer than Nit

Is jumping effective?

unproven derefs

	KLOC	Deref	Nit	Thr	Hop (Impr %)
drupaleditor	3	928	679	179	72 (60)
npr	5	829	617	181	51 (72)
duckduckgo	11	1969	1341	518	143 (72)
lastfm	13	4840	3528	954	477 (50)
github	19	3603	2520	601	290 (52)
seriesguide	32	8184	5438	986	625 (37)
connectbot	33	2190	1562	316	74 (77)
textsecure	38	5921	3643	698	330 (53)
k-9	55	19032	11968	3104	1988 (36)
wordpress	57	15066	9775	2431	1362 (44)
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**

Hopper has 54% fewer than Thresher

Is jumping effective?

	KLOC	unproven derefs Deref	Nit	Thr	Hop (Impr %)	% Proven
drupaleditor	3	928	679	179	72 (60)	92
npr	5	829	617	181	51 (72)	94
duckduckgo	11	1969	1341	518	143 (72)	93
lastfm	13	4840	3528	954	477 (50)	90
github	19	3603	2520	601	290 (52)	92
seriesguide	32	8184	5438	986	625 (37)	92
connectbot	33	2190	1562	316	74 (77)	97
textsecure	38	5921	3643	698	330 (53)	94
k-9	55	19032	11968	3104	1988 (36)	90
wordpress	57	15066	9775	2431	1362 (44)	91
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**	**92**

Is jumping effective?

		unproven derefs				
	KLOC	**Deref**	**Nit**	**Thr**	**Hop (Impr %)**	**% Proven**
drupaleditor	3	928	679	179	72 (60)	92
npr	5	829	617	181	51 (72)	94
duckduckgo	11	1969	1341	518	143 (72)	93
lastfm	13	4840	3528	954	477 (50)	90
github	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	92
seriesguide	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	92
connectbot	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	97
textsecure	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	94
k-9	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	90
wordpress	57	15066	9775	2431	1362 (44)	91
Total	**266**	**62562**	**41071**	**9968**	**5412 (54)**	$\boxed{\textbf{92}}$

Compare with state-of-the art NPE checking work that reports 84-91% proven on normal Java programs!

Triaging alarms to find bugs

Triaging alarms to find bugs

Triaged 200 alarms (from Hopper), 189 false

Triaging alarms to find bugs

Triaged 200 alarms (from Hopper), 189 false

- Reasons: insufficient Android modeling, imprecise container and string domains

Triaging alarms to find bugs

Triaged 200 alarms (from Hopper), 189 false

- Reasons: insufficient Android modeling, imprecise container and string domains
- Only 17 false alarms due to timeouts

Triaging alarms to find bugs

Triaged 200 alarms (from Hopper), 189 false

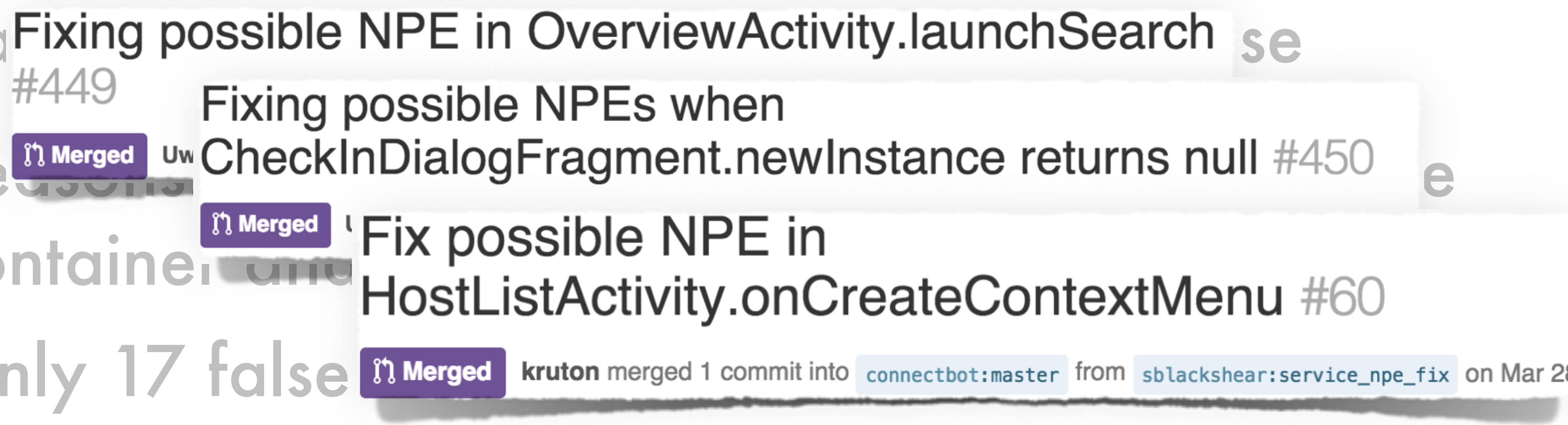
- Reasons: insufficient Android modeling, imprecise container and string domains
- Only 17 false alarms due to timeouts

Found 11 bugs in 4 apps
(lastfm, seriesguide, connectbot, wordpress)

5 bugs due to bad ordering assumptions

10/11 patches accepted

Triaging alarms to find bugs



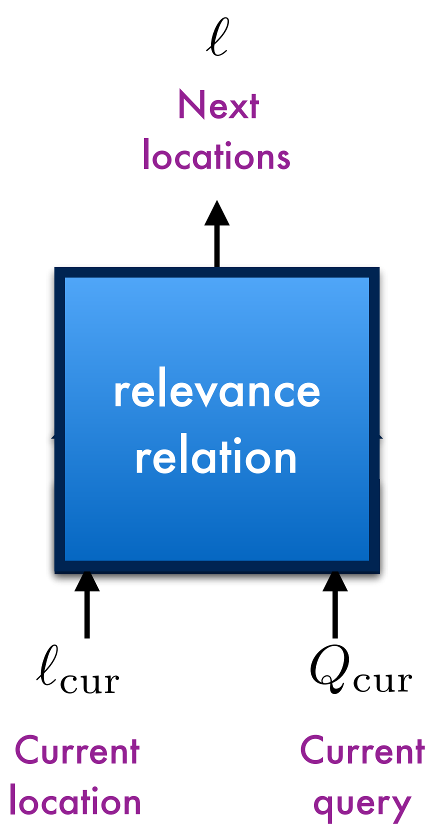
Found 11 bugs in 4 apps
(lastfm, seriesguide, connectbot, wordpress)

5 bugs due to bad ordering assumptions

10/11 patches accepted

Summary

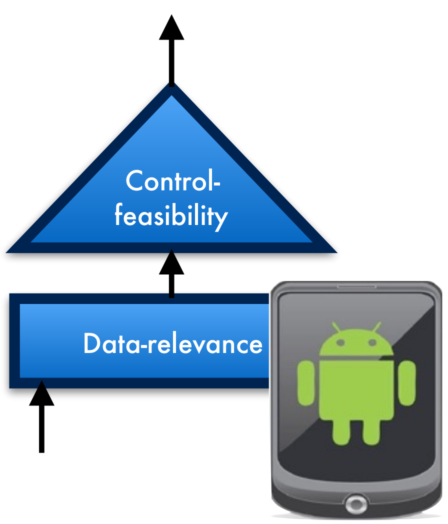
Summary



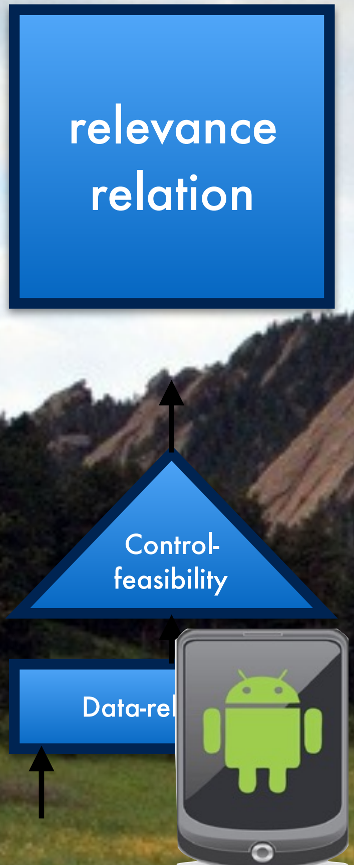
Selective control-flow abstraction via a sound relevance relation

Summary

Selective control-flow abstraction via a sound relevance relation



Effective inter-event ordering-sensitive reasoning via data-relevance and control-feasibility



Selective control-flow abstraction via a sound relevance relation

Effective inter-event ordering-sensitive reasoning via data-relevance and control-feasibility

www.cs.colorado.edu/~bec
pl.cs.colorado.edu

PLV


Cerny


Chang


Hammer


Sankaranarayananan


Somenzi

Goal-directed:
abstraction-refinement versus abstraction-coarsening

staged abstraction refinement
(e.g., CEGAR)

on-the-fly abstraction
coarsening

Goal-directed: abstraction-refinement versus abstraction-coarsening

staged abstraction refinement (e.g., CEGAR)

on-the-fly abstraction coarsening

Staged

Run analysis multiple times, abstraction changes only between runs.

On-the-fly

Run analysis once, abstraction changes during analysis.

Goal-directed: abstraction-refinement versus abstraction-coarsening

staged abstraction refinement
(e.g., CEGAR)

on-the-fly abstraction
coarsening

Staged

Run analysis multiple times,
abstraction changes only
between runs.

On-the-fly

Run analysis once,
abstraction changes
during analysis.

Refinement

Start with imprecise
abstraction,
become more precise.

Coarsening

Start with precise
abstraction,
become less precise.