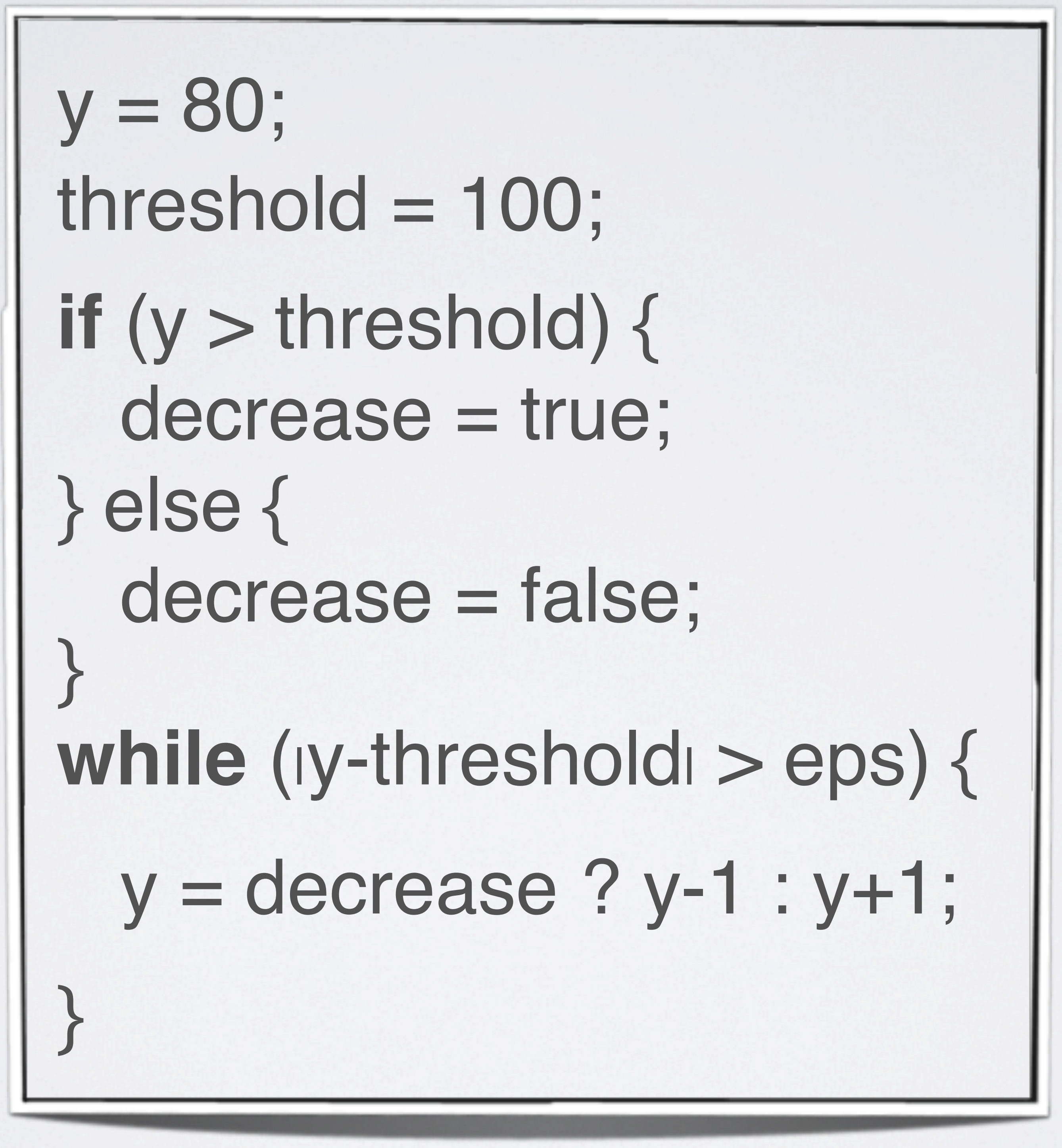
COMBINING PARTIAL EVALUATION AND SYMBOLIC EXECUTION

Reiner Hähnle & Richard Bubel
Chalmers University

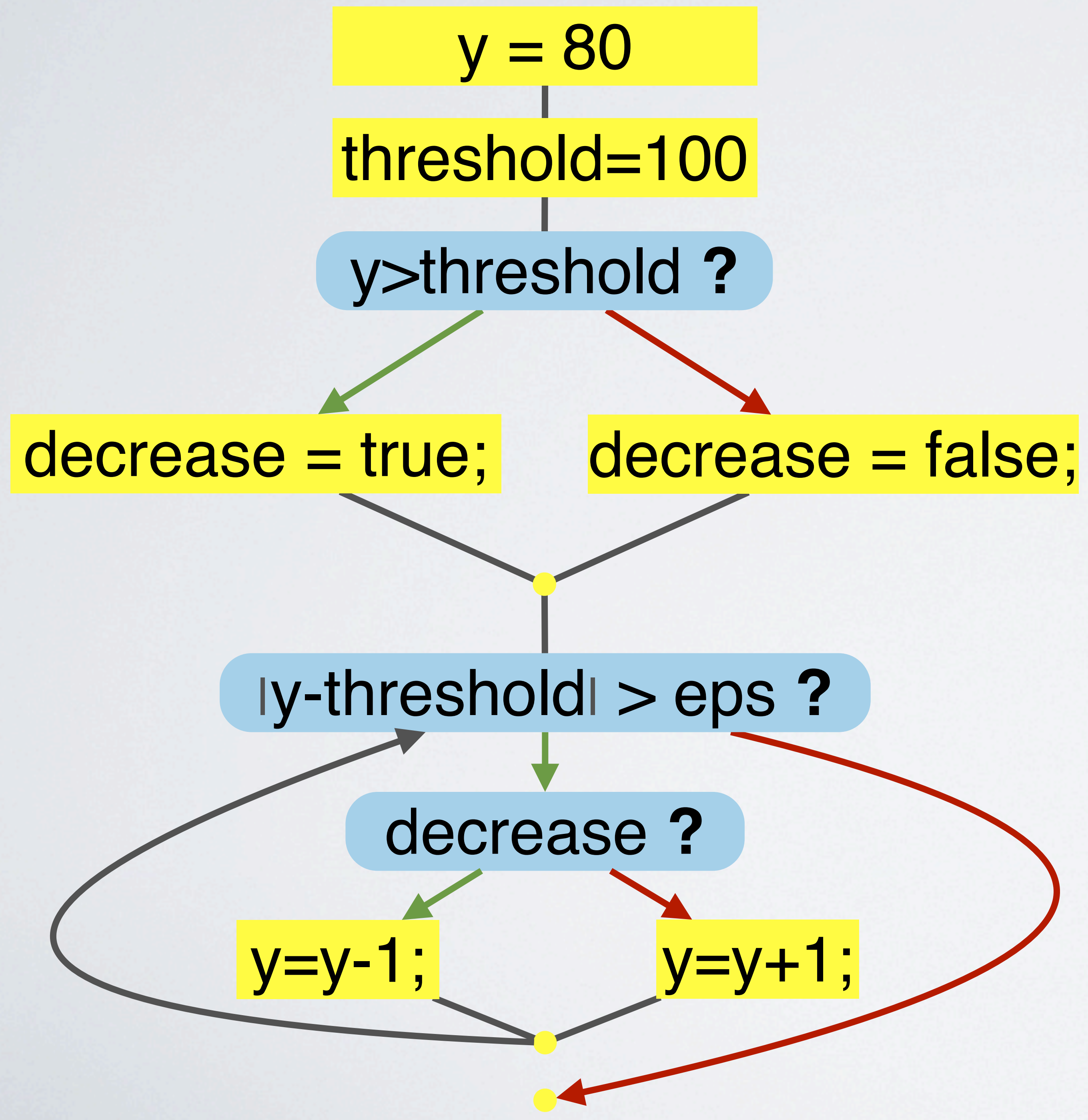
Symposium'09
Speyer

CONTROL CIRCUIT



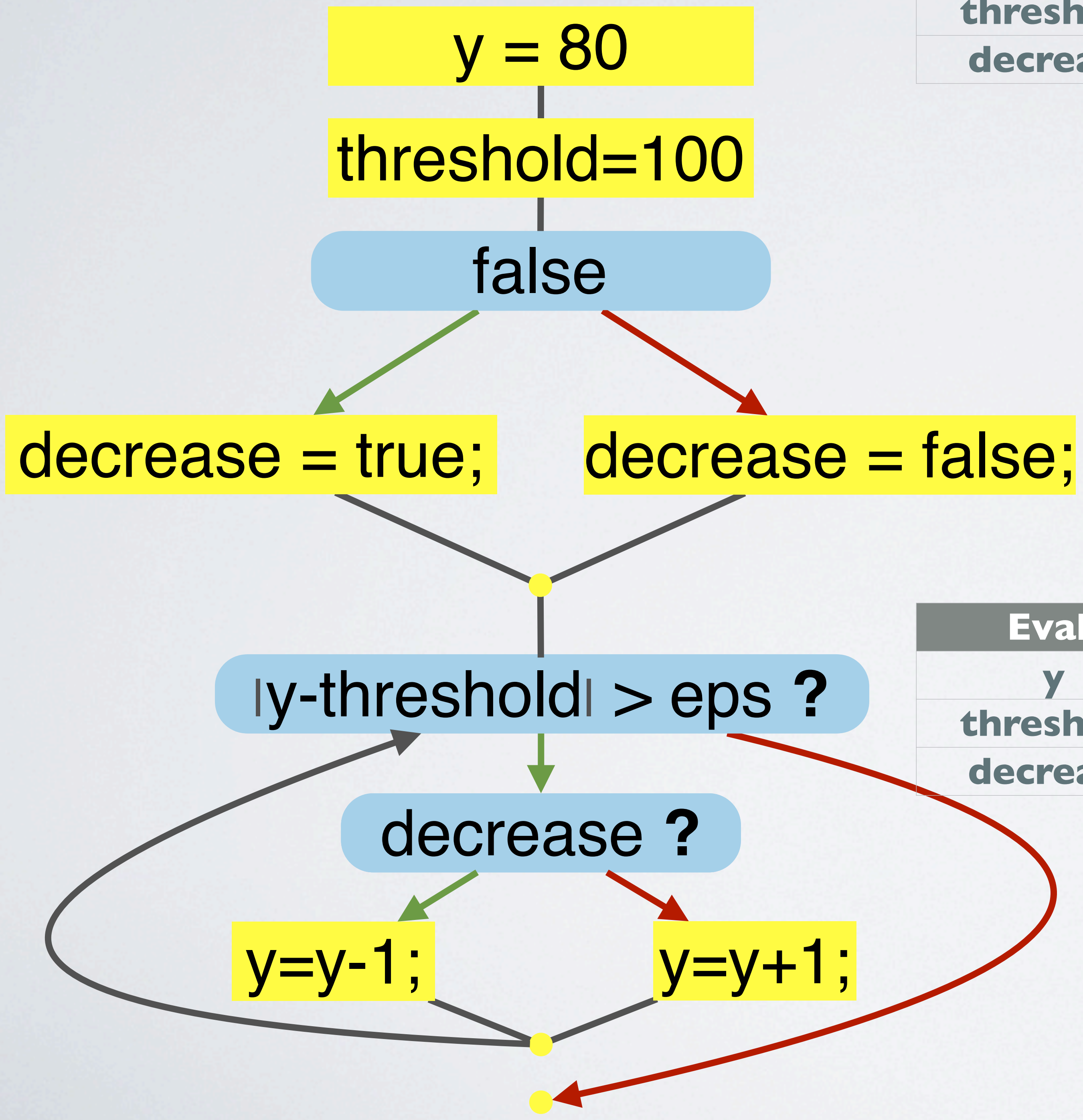
```
y = 80;
threshold = 100;
if (y > threshold) {
   decrease = true;
} else {
   decrease = false;
}
while (|y-threshold| > eps) {

   y = decrease ? y-1 : y+1;

}
```

CONTROL-FLOW GRAPH



```
y = 80;
threshold = 100;
if (y > threshold) {
    decrease = true;
} else {
    decrease = false;
}
while (|y-threshold| > eps) {
    y = decrease ? y-1 : y+1;
}
```

PARTIAL EVALUATION



Evaluator	
y	80
threshold	100
decrease	false

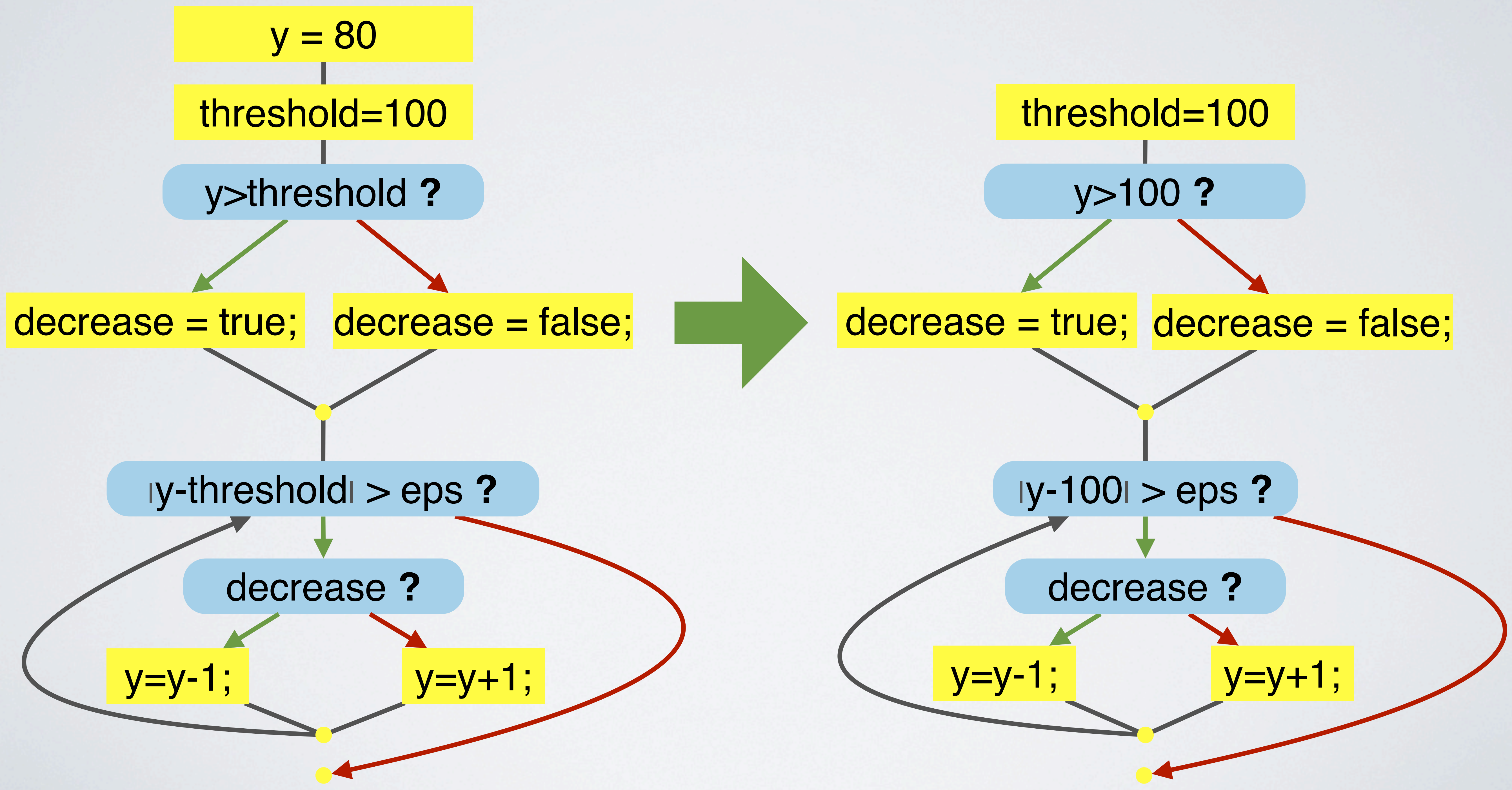
Evaluator	
y	-
threshold	100
decrease	false

<u>Static information propagated along CFG</u>

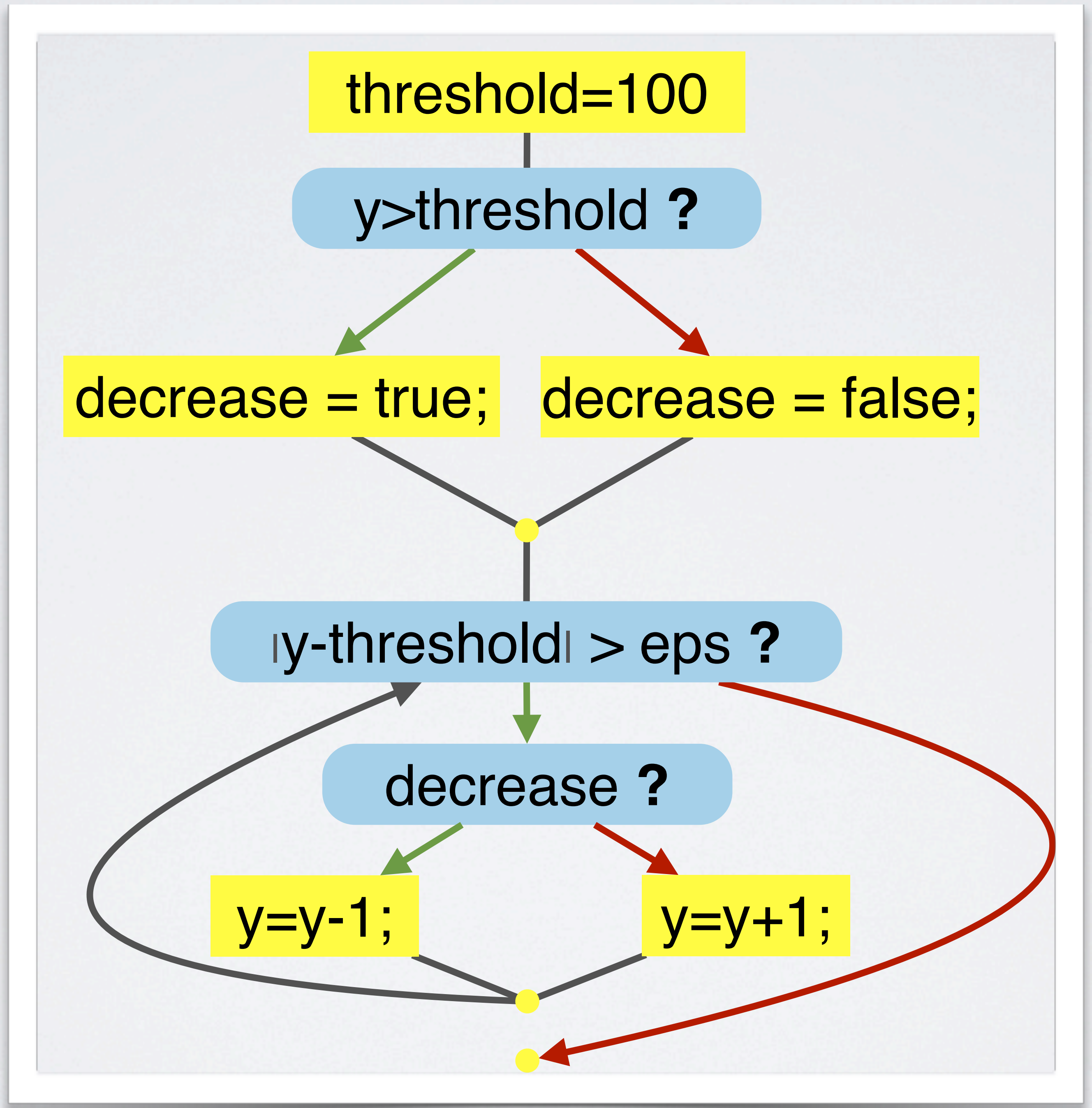
- constant propagation
- constant expression evaluation
- dead-code elimination
- other: type coercion, safe dereferencing etc.

PARTIAL EVALUATION

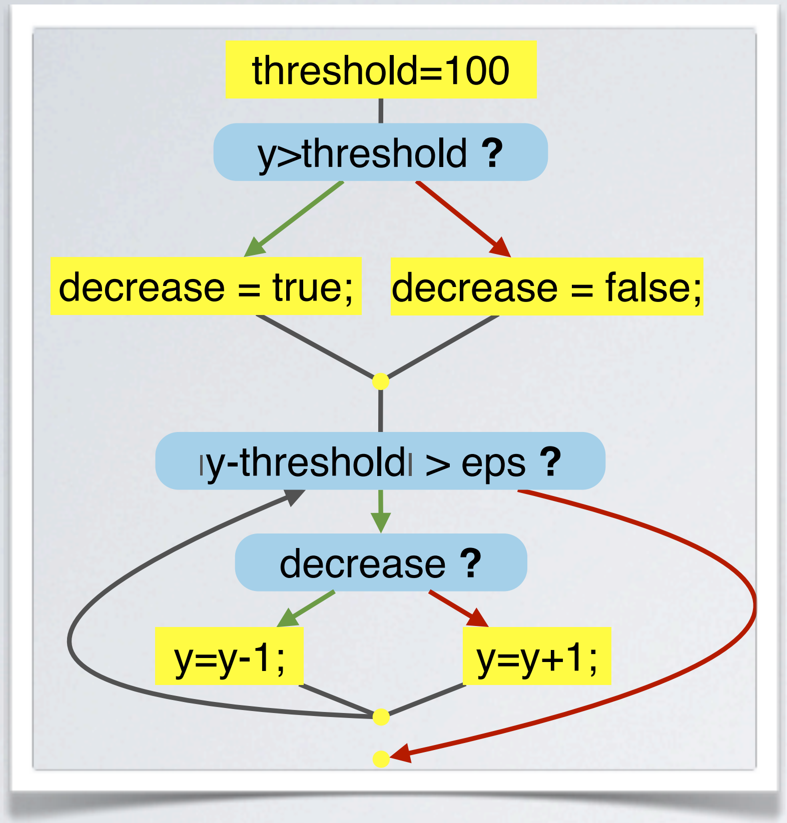
A Bit More Realistic

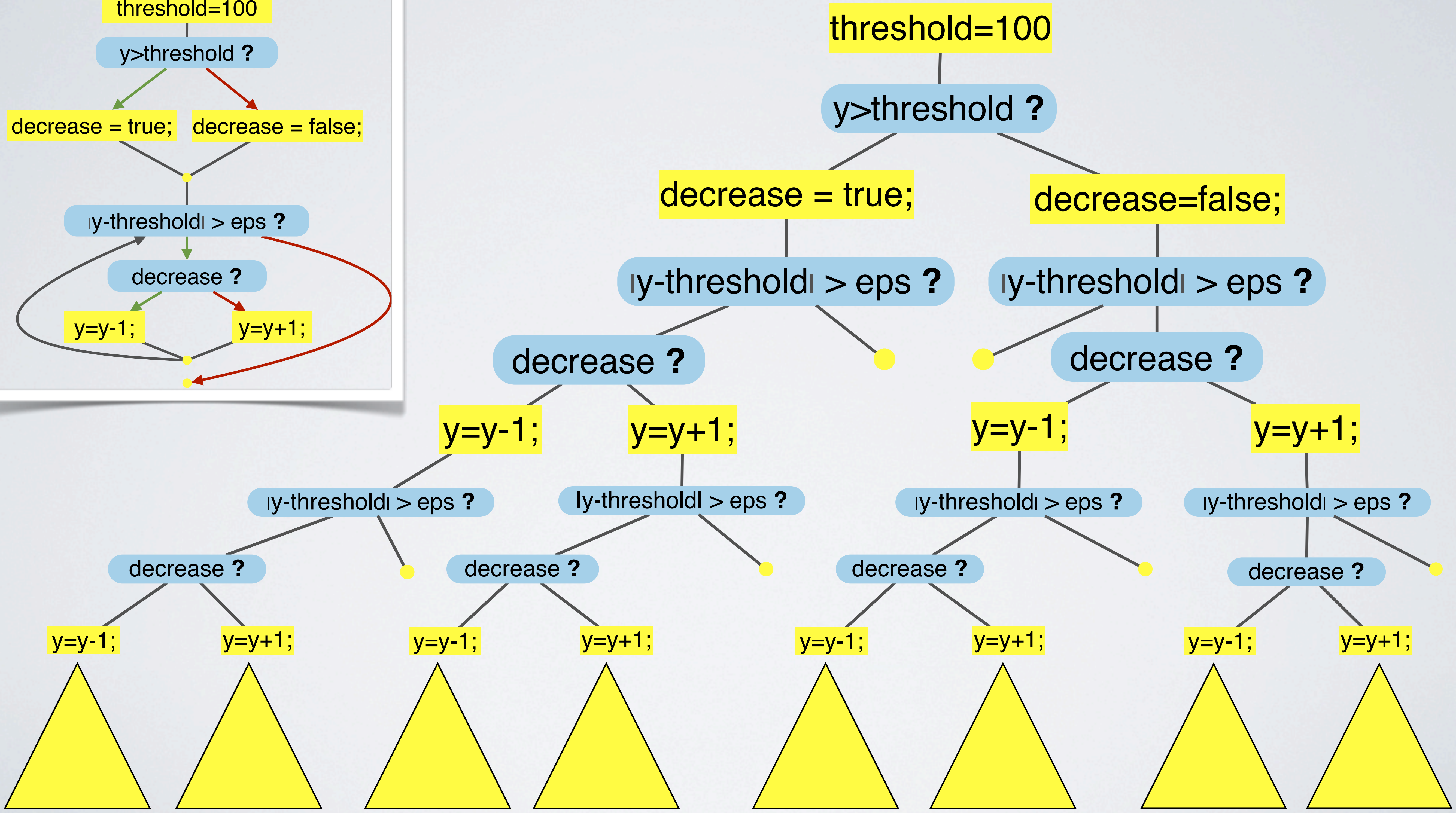


SYMBOLIC EXECUTION



SYMBOLIC EXECUTION





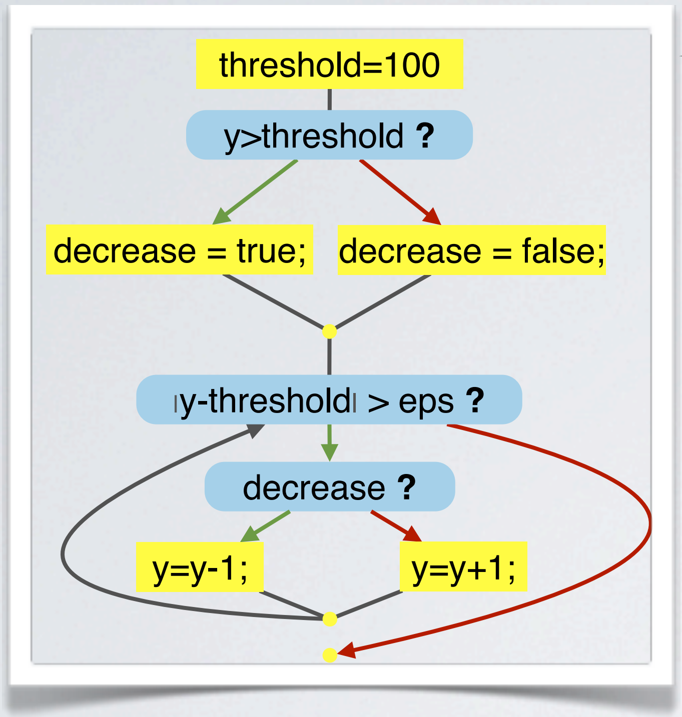
OPTIMIZING SYMBOLIC EXECUTION

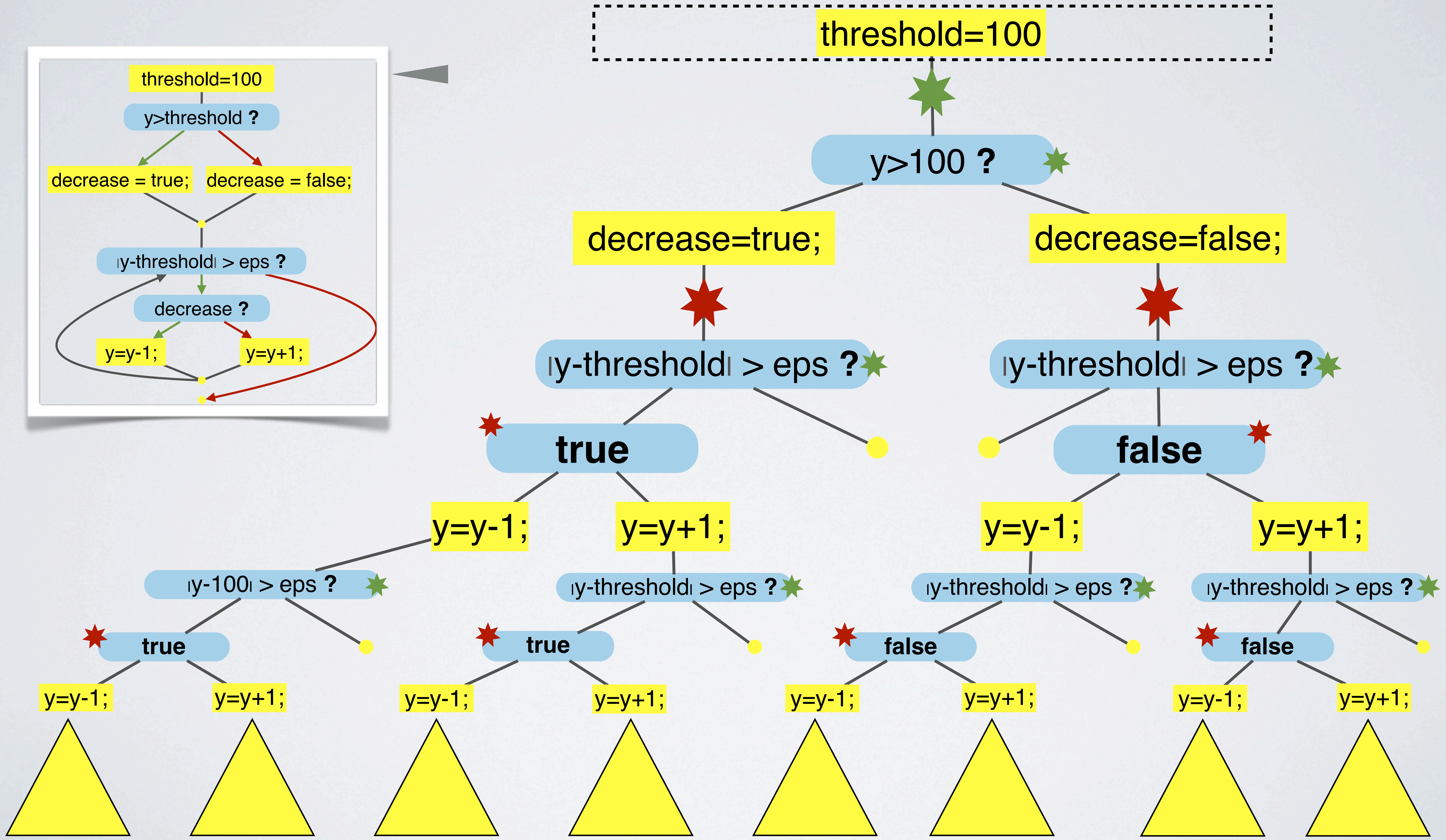
Symbolic Execution

- unfolds control-flow graph into tree
- unfeasible paths must be closed by first-order proof search

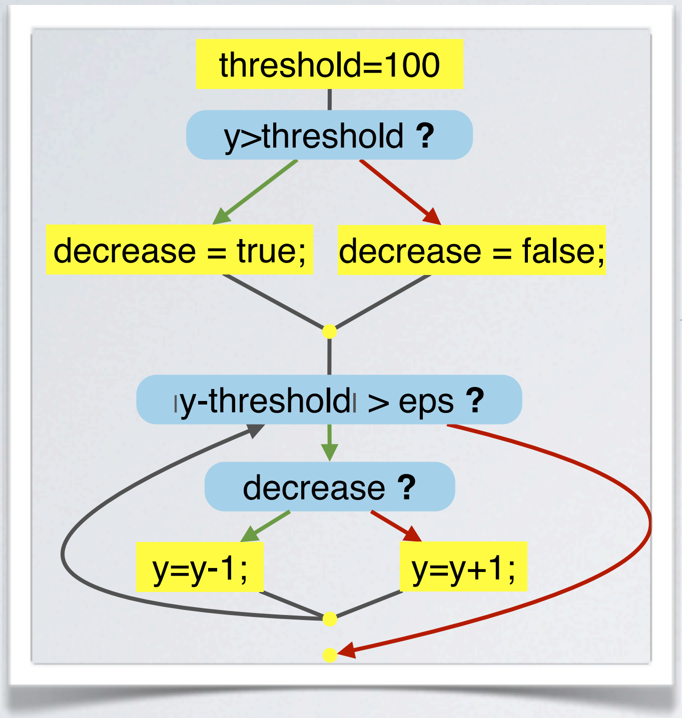
→ interleave partial evaluation and symbolic execution

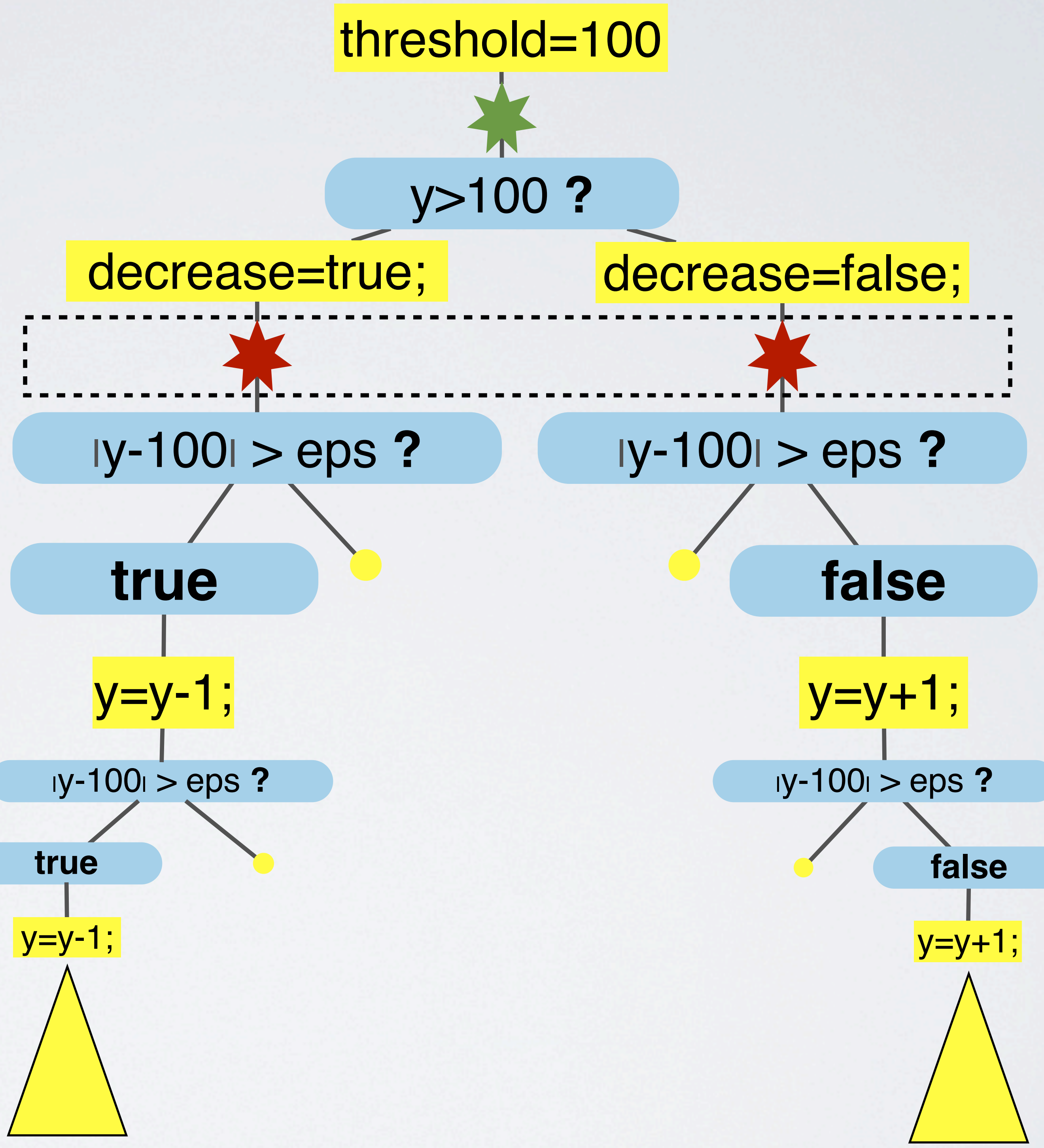
INTERLEAVING





INTERLEAVING





PROGRAM LOGIC

Programming Language

Simple OO-Programming Language

- single inheritance
- dereferencing null, division by zero etc. cause non-termination
- dynamic method binding
- no nested expressions

PROGRAM LOGIC

Syntax

Dynamic Logic with Updates: (as usual)

Specialisation operator

$$\downarrow : PrgEl \times Upd \times For \rightarrow PrgEl$$

where $PrgEl = Statement \,\dot{\cup}\, Expression$

$\mathtt{p} \downarrow (\mathcal{U}, \varphi)$ denotes a program equivalent to $\mathtt{p}$ if $\mathtt{p}$ is executed in a state s satisfying φ and coinciding on $\mathcal{U}$

Examples:

- $\mathtt{x} = (\mathtt{y}) \downarrow (y := 3, true) + \mathtt{3};$
- $(\mathtt{x} = \mathtt{o.a} + \mathtt{3}) \downarrow (o.a := 10, o != \mathtt{null})$

PROGRAM LOGIC

Notions

Signature Σ:

Program variables and attributes are modelled as non-rigid constants and unary function symbols

First-order structure (D, I):

- Domain D: sorted universe (interpretes sorts)
- Interpretation I: interpretes rigid function and predicate symbols

States $s \in S$:
interpretes program variables and attributes

PROGRAM LOGIC

Signature Extension

Partial Evaluation may extend the signature
(temporary variables, anonymous updates)

$$\mathtt{p} \downarrow_{\Sigma' \supseteq \Sigma} (\mathcal{U}, \varphi)$$

where

$$(D, I)_{\Sigma'} \supseteq (D, I)_{\Sigma} \text{ and } s_{\Sigma'} \supseteq s_{\Sigma} \text{ and } \beta_{\Sigma'} \supseteq \beta_{\Sigma}$$

PROGRAM LOGIC

Soundness Condition on the Specialisation Operator

$$\mathbf{p} \downarrow_{\Sigma' \supseteq \Sigma} (\mathcal{U}, \varphi)$$

For all formulas ψ over Σ, for all $(D, I)_{\Sigma'}, s_{\Sigma'}, \beta_{\Sigma'}$:

$$(D, I)_{\Sigma'}, s_{\Sigma'}, \beta_{\Sigma'} \models \langle \mathtt{p} \downarrow (\mathcal{U}, \varphi) \rangle \psi \rightarrow \left(\{\mathcal{U}\}(\varphi \rightarrow \langle \mathtt{p} \rangle \psi) \right)$$

PROGRAM LOGIC

Partial Evaluation Rules

	Rewrite Action	Correctness Requirement
Dead-Code Elimination	$\texttt{if(b)\{p\}else\{q\}}\downarrow(\mathcal{U},\varphi) \rightsquigarrow \texttt{p}\downarrow(\mathcal{U},\varphi)$	$\mathcal{U}(\varphi \longrightarrow b \doteq \texttt{true})$
Safe Field Access	$\texttt{o.a}\downarrow(\mathcal{U},\varphi) \rightsquigarrow @(\texttt{o.a})\downarrow(\mathcal{U},\varphi)$	$\mathcal{U}(\varphi \longrightarrow !(o \doteq \texttt{null}))$
Partial Evaluator Propagation	$(\texttt{p};\texttt{q})\downarrow(\mathcal{U},\varphi) \rightsquigarrow \texttt{p}\downarrow(\mathcal{U},\varphi);\texttt{q}\downarrow(\mathcal{U}',\varphi')$	$\vdash \mathrm{respModStrong}(\texttt{p}, mod)$ $\mathcal{U}' := \mathcal{U}\mathcal{V}_{mod}$ $(D,I) \models \{\mathcal{U}\}\{\mathcal{V}_{mod}\}\varphi' \Rightarrow (D,I) \models \{\mathcal{U}\}\langle\texttt{p}\rangle\varphi$

PROGRAM LOGIC

Partial Evaluator Introduction Rules

$$\frac{\begin{array}{l}\Gamma \vdash \mathcal{U}!(\mathtt{o} \doteq \mathtt{null}), \Delta \\ \Gamma \vdash \mathcal{U}\{\mathtt{o.a} := \mathtt{t}\}\langle \mathtt{q} \downarrow (\mathtt{o.a} := \mathtt{t}, !\mathtt{o} \doteq \mathtt{null})\rangle \phi, \Delta\end{array}}{\Gamma \vdash \mathcal{U}\langle \mathtt{o.a} = \mathtt{t}; \mathtt{q}\rangle \phi, \Delta}$$

and several others

PROGRAM LOGIC

Type Inference Rules

$$\texttt{res} = \texttt{o.m}(\texttt{a}_1, \ldots, \texttt{a}_\texttt{n}) \downarrow (\varphi, \mathcal{U})$$

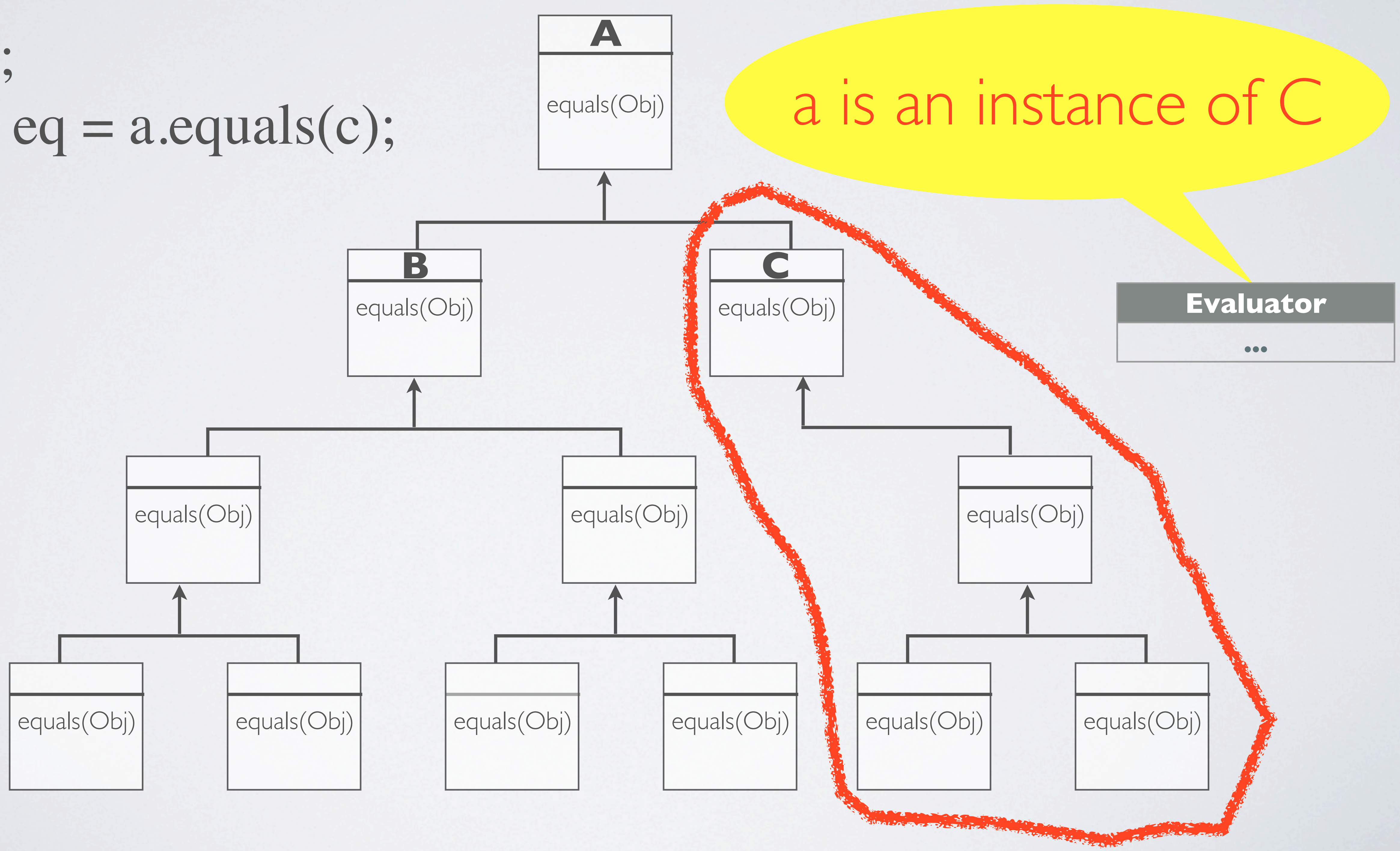
$$\texttt{o}' = \texttt{o} \downarrow (\varphi, \mathcal{U}) \qquad \Big\Downarrow \qquad \vdash \mathcal{U}(\varphi \rightarrow \texttt{o!} \doteq \texttt{null} \ \& \ \texttt{C} :: \texttt{instance}(\texttt{o}))$$

$$\texttt{res} = @((\texttt{C})\texttt{o}').\texttt{m}(\texttt{a}_1 \downarrow (\varphi, \mathcal{U}), \ldots, \texttt{a}_\texttt{n} \downarrow (\varphi, \mathcal{U}))$$

PROGRAM LOGIC

Type Inference Rules

```
A a = ...;
boolean eq = a.equals(c);
```



DEMO

FUTURE WORK

- Simplification of specifications
  - Partial evaluation of contracts and loop invariants
  - Applicable to JavaCardDL / JML / OCL
- Investigate applicability to application engineering

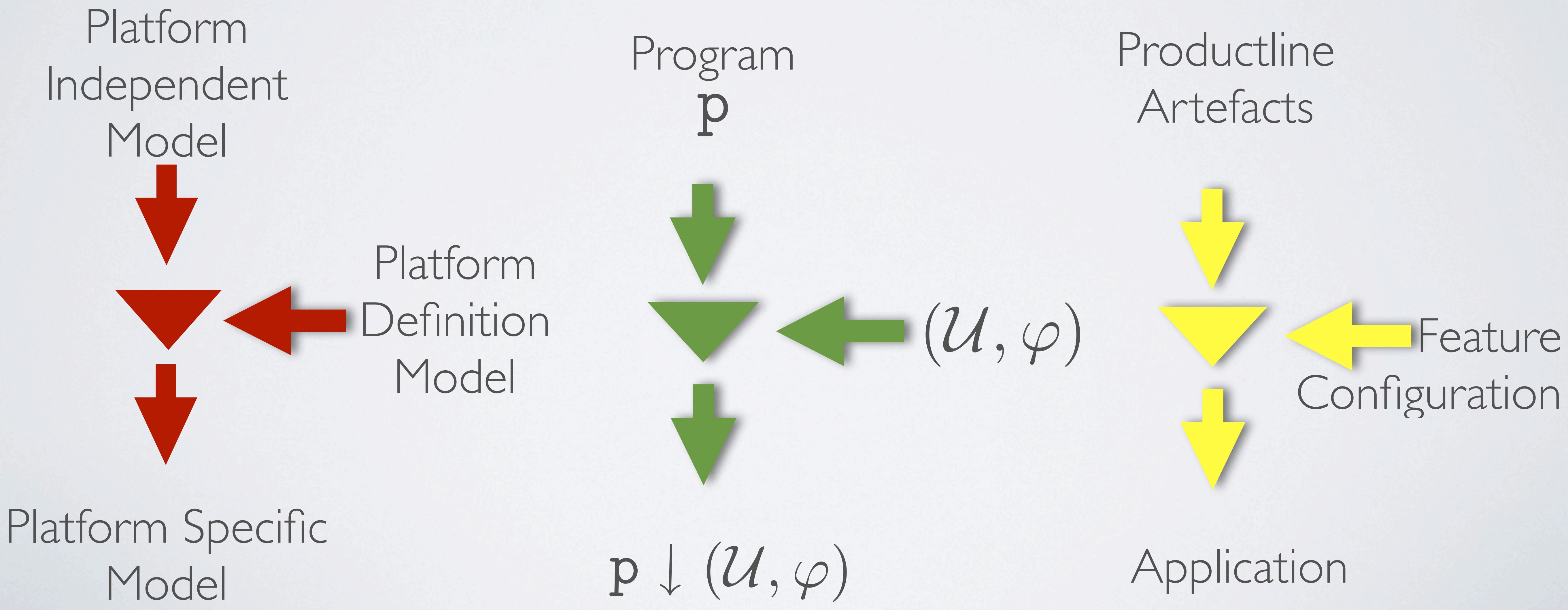
FUTURE WORK

Application Engineering

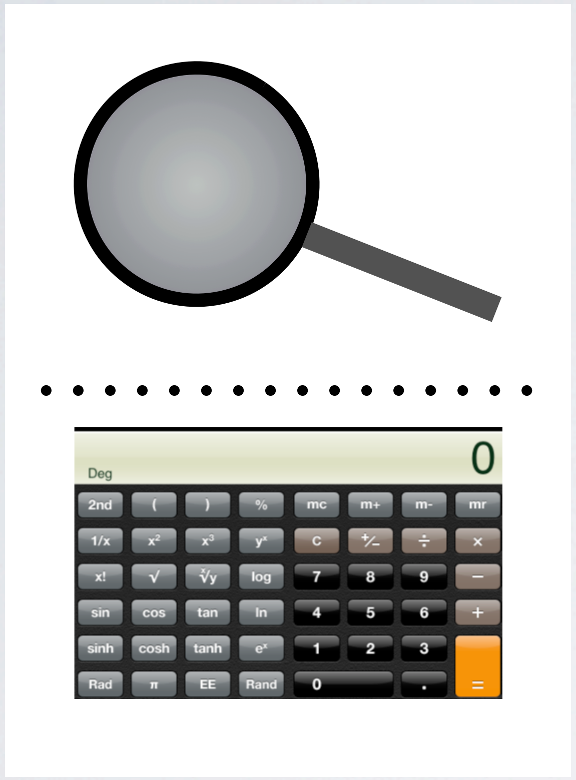
Model Driven Architecture

Partial Evaluation

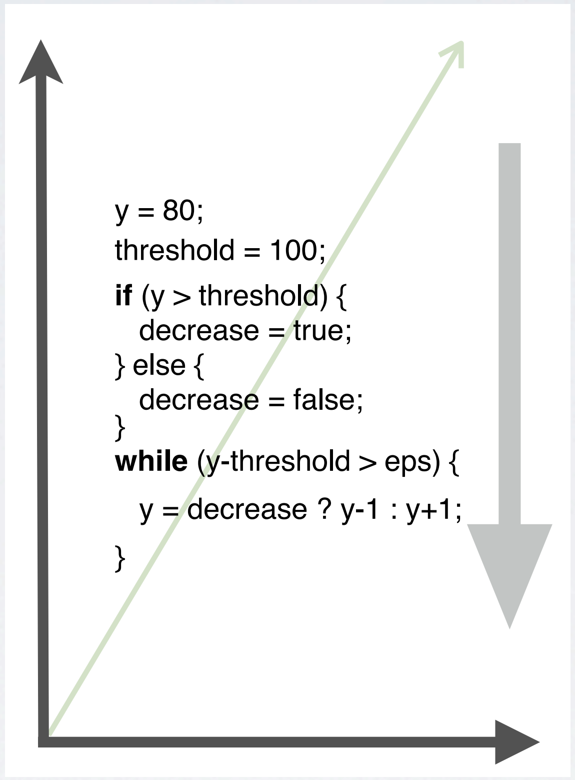
Application Engineering



CONCLUSION
(for the moment)

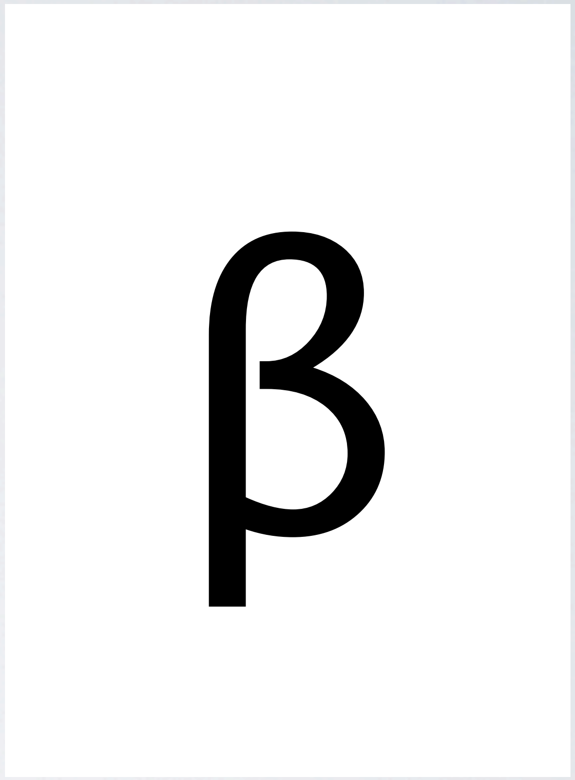


replaced
proof search ⊢
by
computation ↓



```
y = 80;
threshold = 100;
if (y > threshold) {
   decrease = true;
} else {
   decrease = false;
}
while (y-threshold > eps) {
   y = decrease ? y-1 : y+1;
}
```

computation ↓
linear in
number of locs



partial eval. as
generalisation of
β-reduction
in
Hoare/VCG