An executable formal semantics for PHP

Daniele Filaretti & Sergio Maffeis

www.phpsemantics.org

Imperial College London

Dagstuhl 2014

phpMyAdmin

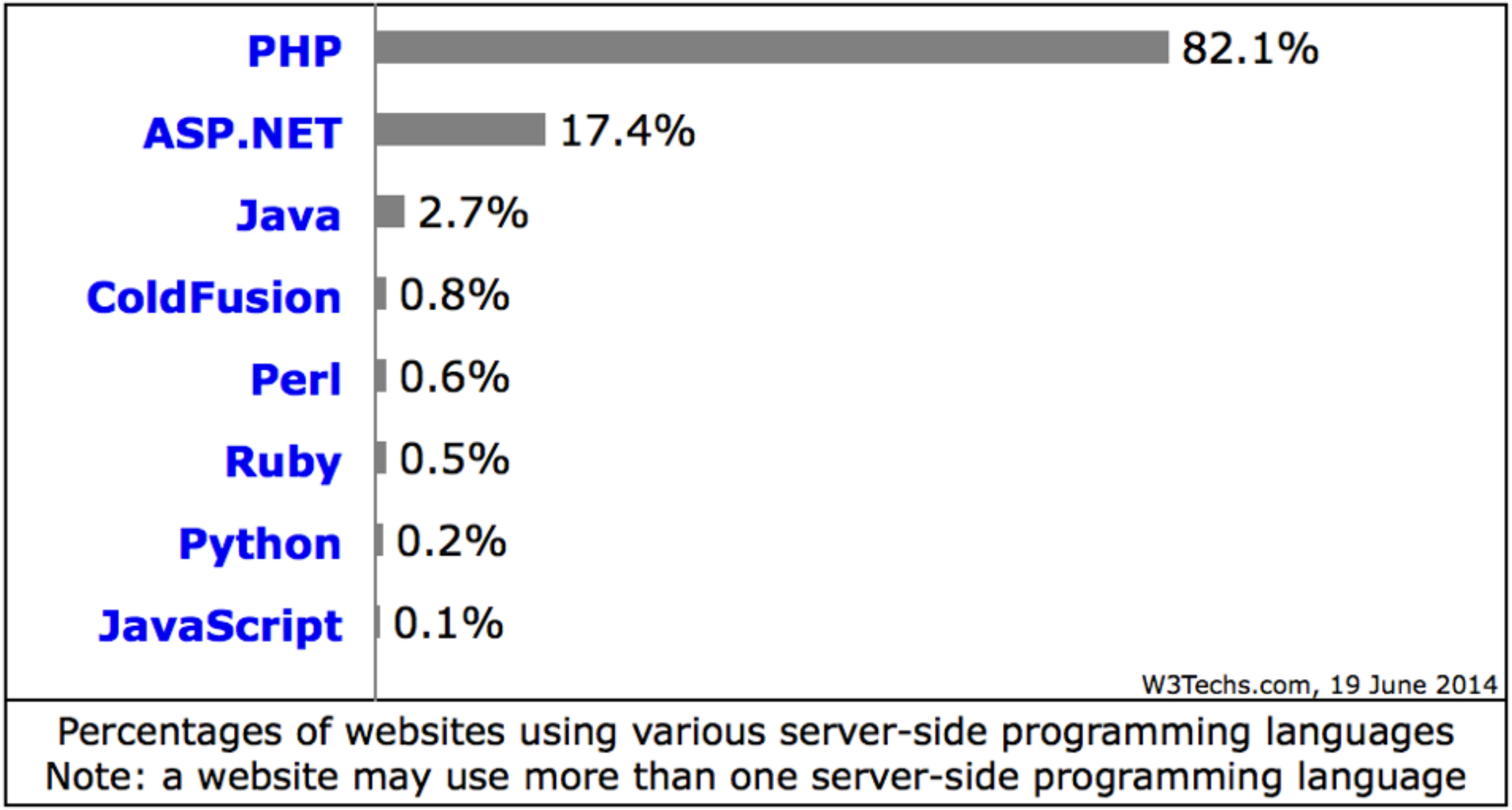
YAHOO!

php

moodle

WordPress

f



Percentages of websites using various server-side programming languages
Note: a website may use more than one server-side programming language

http://w3techs.com/technologies/overview/programming_language/all

```
$a = array("one");
```

```php
$a = array("one");
$c = $a[0] . ($a[0] = "two");
```

```php
$a = array("one");
$c = $a[0] . ($a[0] = "two");
echo $c;
```

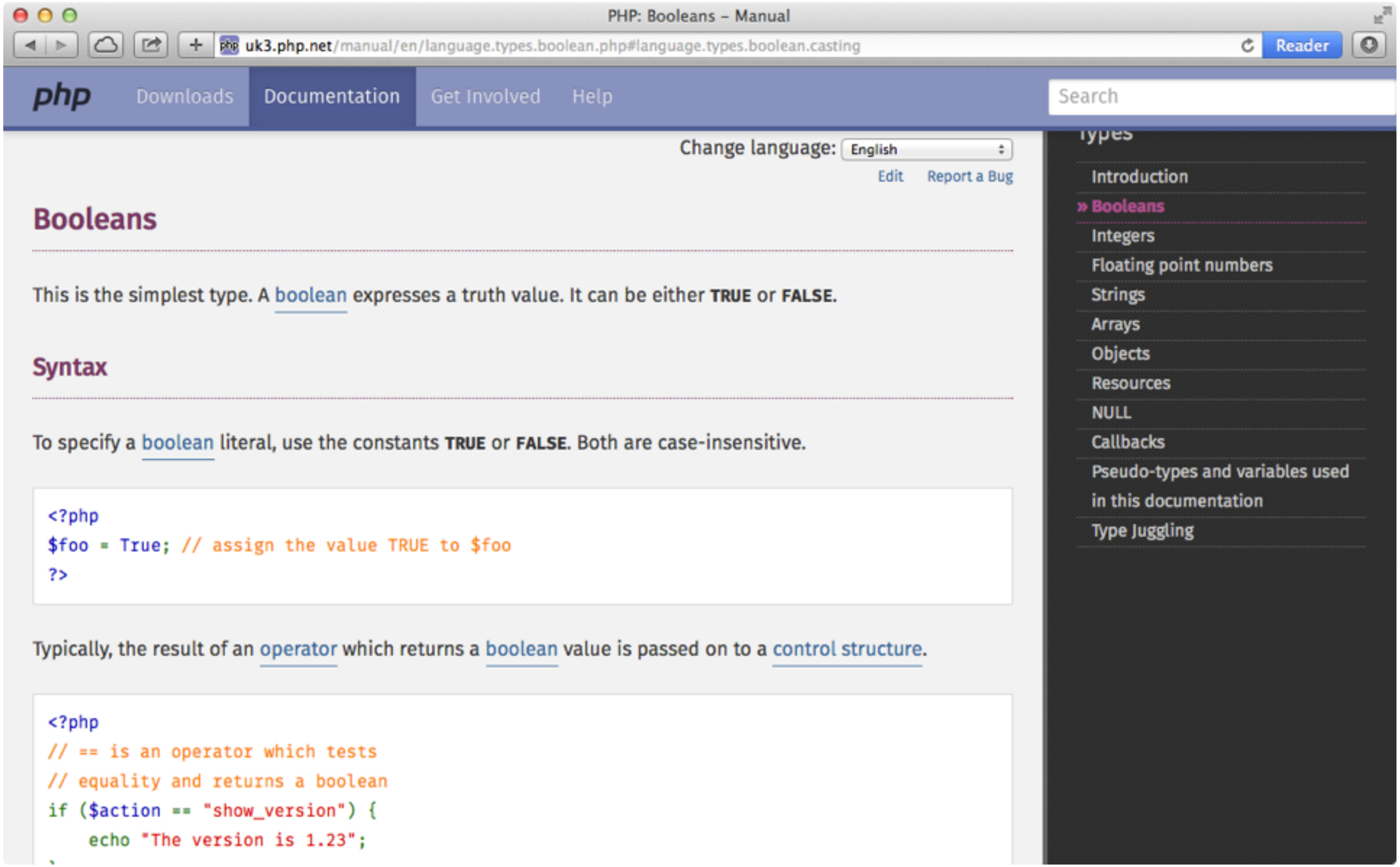
```php
$a = array("one");
$c = $a[0] . ($a[0] = "two");
echo $c; // “onetwo"
```

```
$a = array("one");
$c = $a[0] . ($a[0] = "two");
echo $c; // “onetwo"

$a = "one";
$c = $a . ($a = "two");
echo $c;
```

```php
$a = array("one");
$c = $a[0] . ($a[0] = "two");
echo $c; // “onetwo"

$a = "one";
$c = $a . ($a = "two");
echo $c; // "twotwo"
```

PHP tricky features

- **Aliasing**
- Complex array and object **iteration**
- Automatic **type conversions**
- Complex **array copy**
- Complex instance variable **lookup**



Booleans

This is the simplest type. A boolean expresses a truth value. It can be either **TRUE** or **FALSE**.

Syntax

To specify a boolean literal, use the constants **TRUE** or **FALSE**. Both are case-insensitive.

```
<?php
$foo = True; // assign the value TRUE to $foo
?>
```

Typically, the result of an operator which returns a boolean value is passed on to a control structure.

```
<?php
// == is an operator which tests
// equality and returns a boolean
if ($action == "show_version") {
    echo "The version is 1.23";
```

dynamic language

dynamic language

+

no spec

dynamic language

+

no spec

+

poor documentation

dynamic language

+

no spec

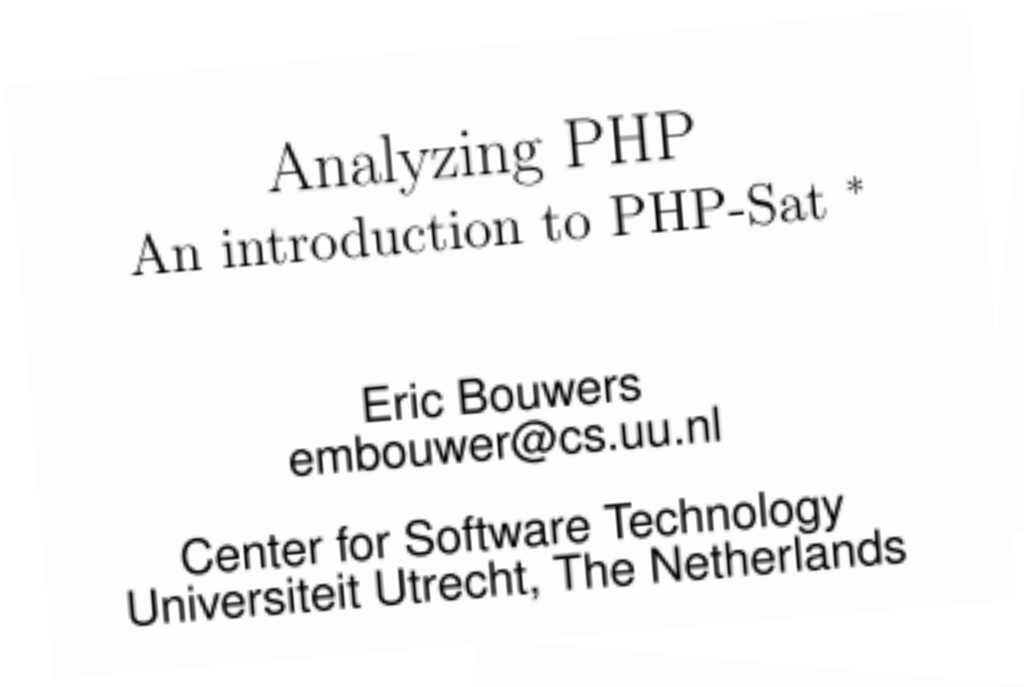
+

poor documentation

=>

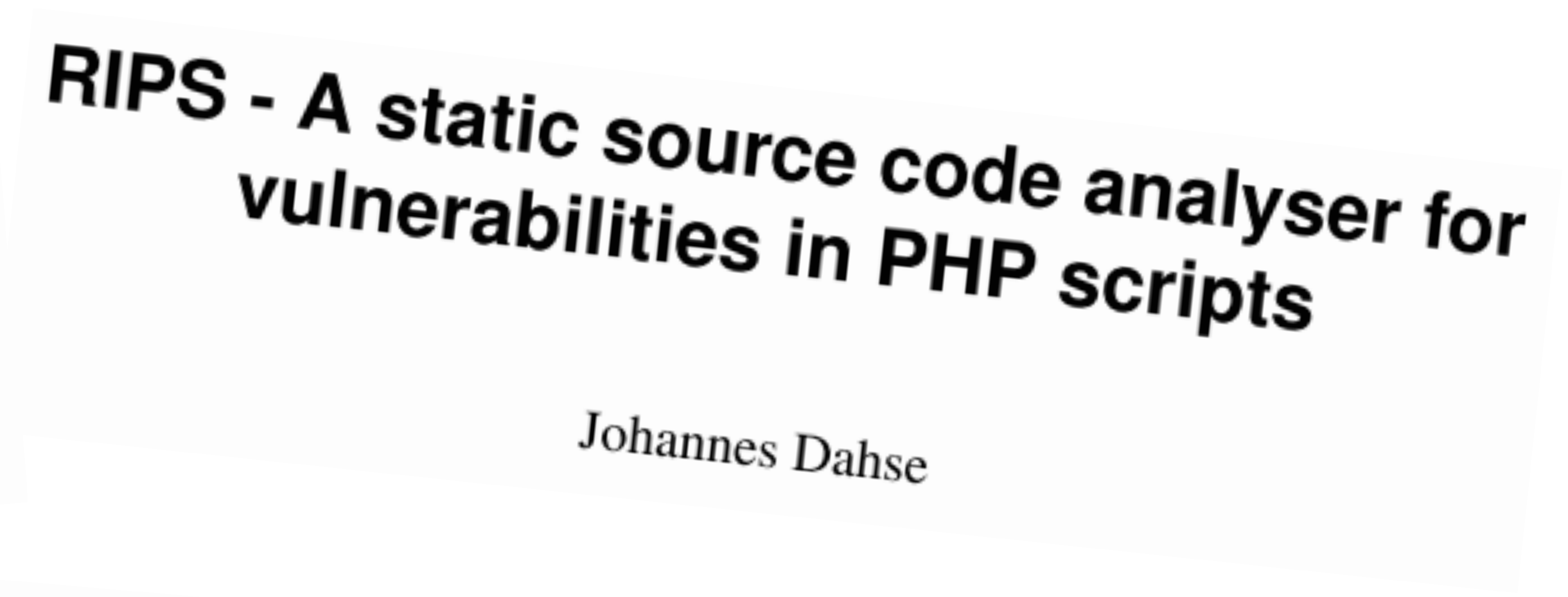
bugs, confusion, etc.

OWASP Top 10 – 2013 (New)
A1 – Injection
A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References
A5 – Security Misconfiguration
A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control
A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards
Merged with 2010-A7 into new 2013-A6

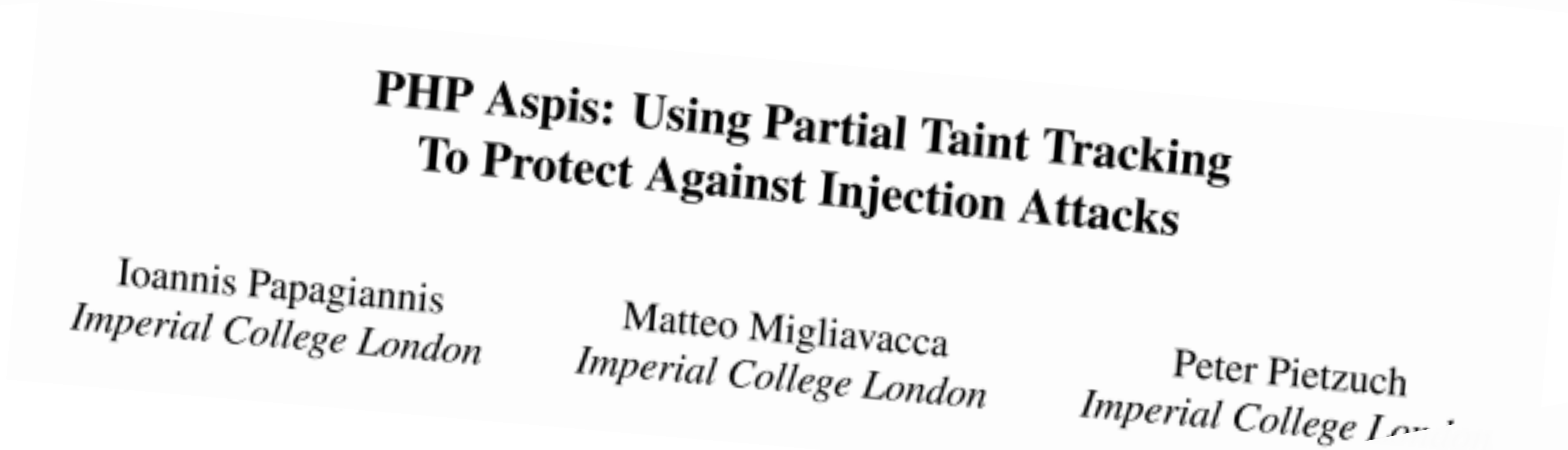

Analyzing PHP
An introduction to PHP-Sat *

Eric Bouwers
embouwer@cs.uu.nl

Center for Software Technology
Universiteit Utrecht, The Netherlands


RIPS - A static source code analyser for vulnerabilities in PHP scripts

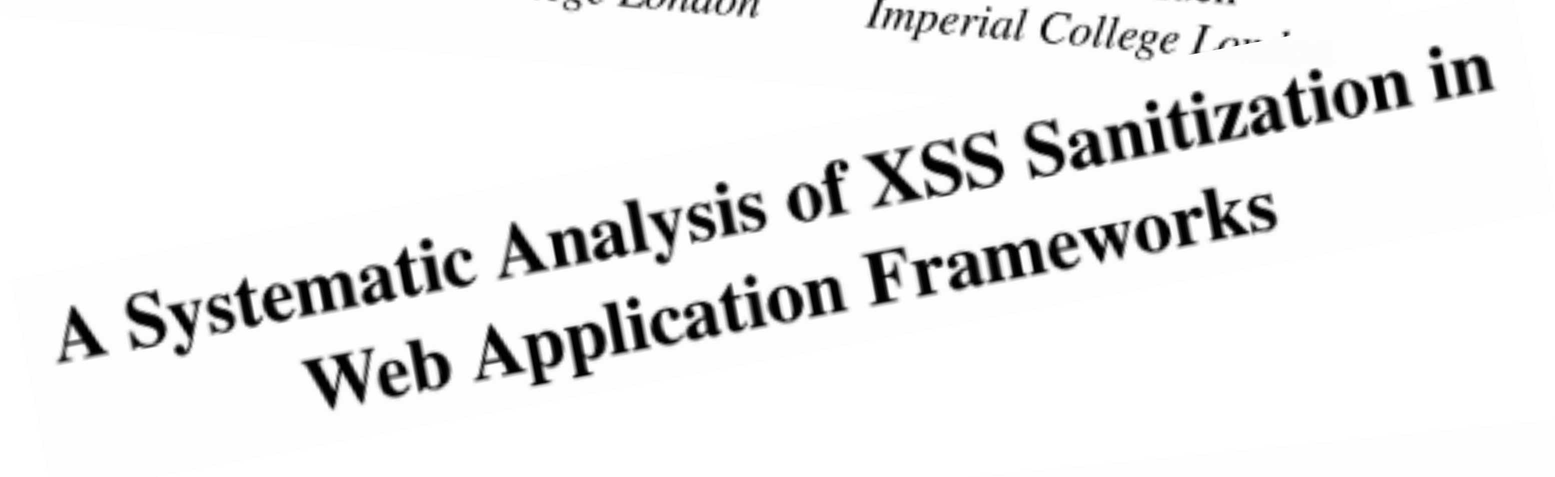
Johannes Dahse

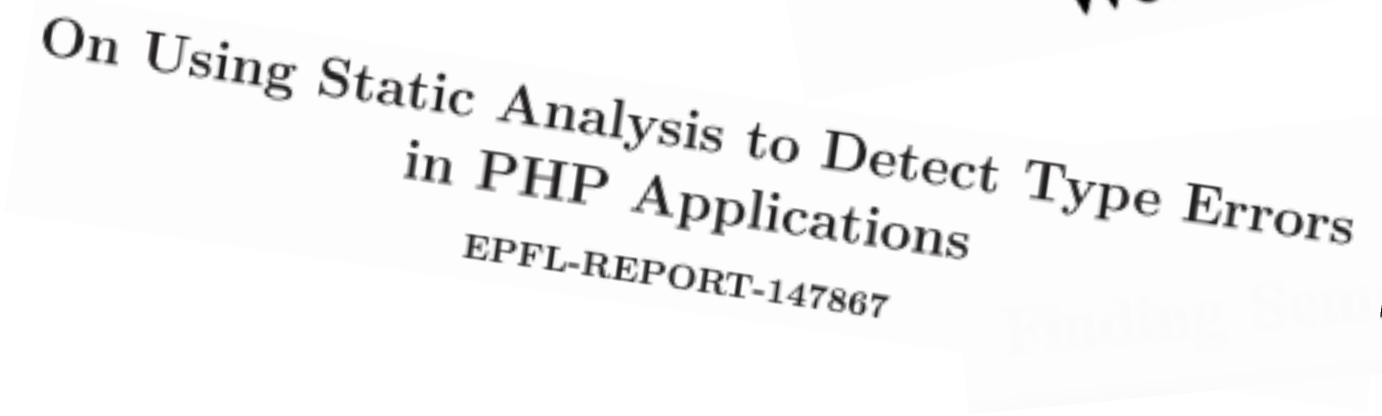

PHP Aspis: Using Partial Taint Tracking To Protect Against Injection Attacks

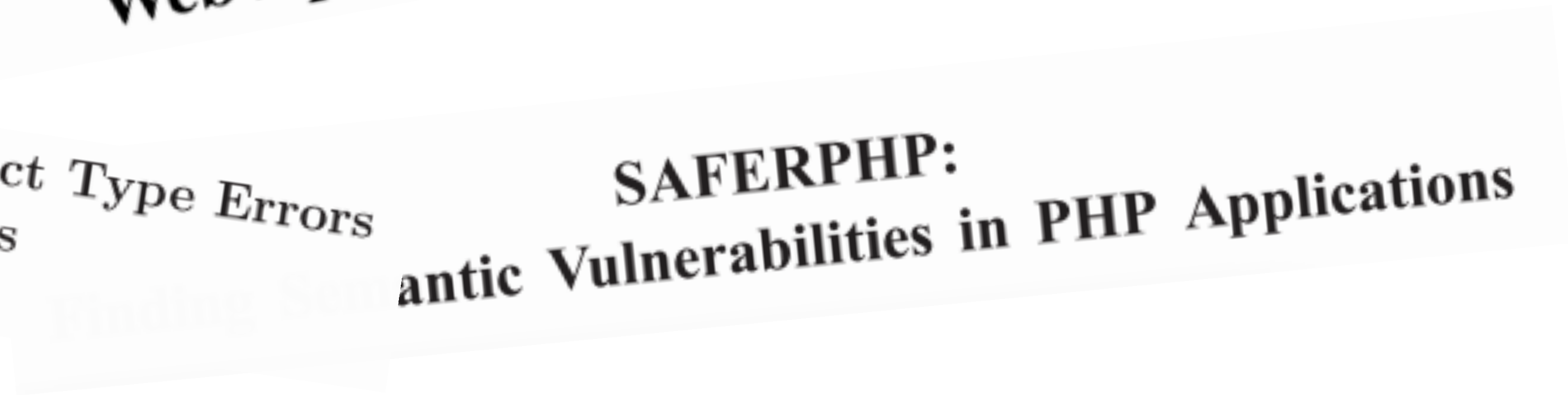
Ioannis Papagiannis
Imperial College London

Matteo Migliavacca
Imperial College London

Peter Pietzuch
Imperial College Lon


A Systematic Analysis of XSS Sanitization in Web Application Frameworks


On Using Static Analysis to Detect Type Errors in PHP Applications
EPFL-REPORT-147867


SAFERPHP:
Finding Semantic Vulnerabilities in PHP Applications

Limitations

- partial coverage of the language - i.e. features ignored because “too hard” for analysis
- sometimes, features modelled incorrectly
- no formal guarantees of soundness

Our goal: A *Trusted* Executable Formal Semantics of PHP

Our goal: A *Trusted* Executable Formal Semantics of PHP

framework for
reliable tools
development

Our goal: A *Trusted* Executable Formal Semantics of PHP

framework for *reliable tools* development

the missing *specification*

Our tools and methodology:

The $\mathbb{K}$ Framework

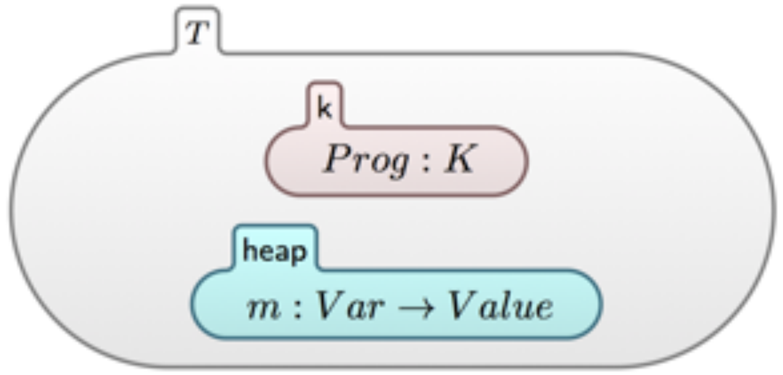
Scales to real languages
(C semantics - POPL’10)

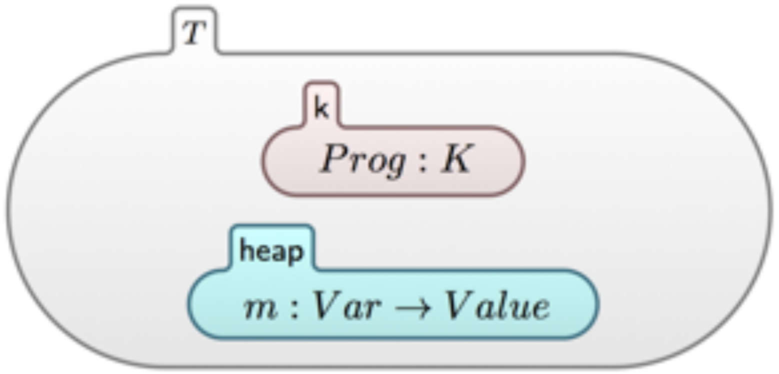
The $\mathbb{K}$ Framework

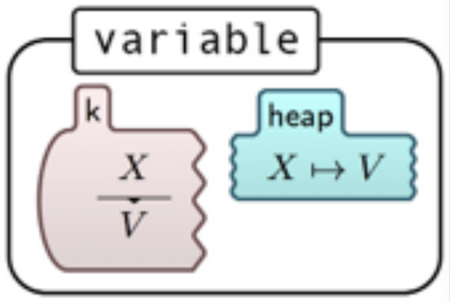
Formal
(rewriting)

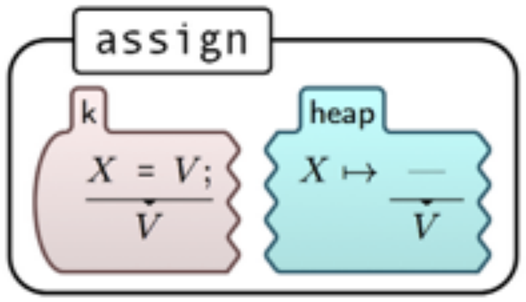
Executable
(Maude/Java)

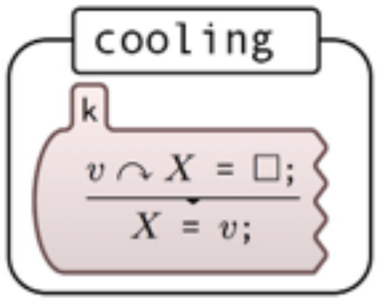
Verification
(deductive, LTL, symbolic exec.)

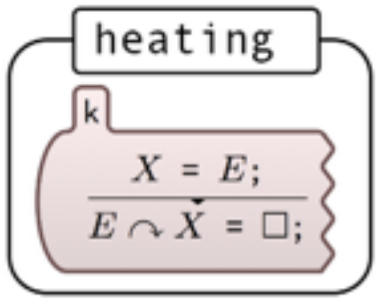

T
k
$Prog : K$
heap
$m : Var \to Value$

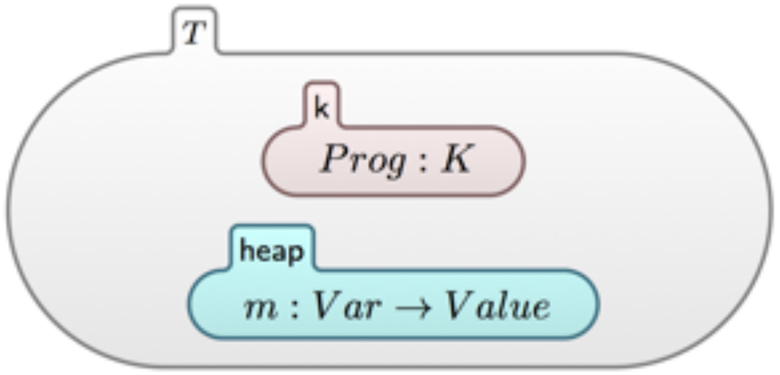

T
k
$Prog : K$
heap
$m : Var \to Value$

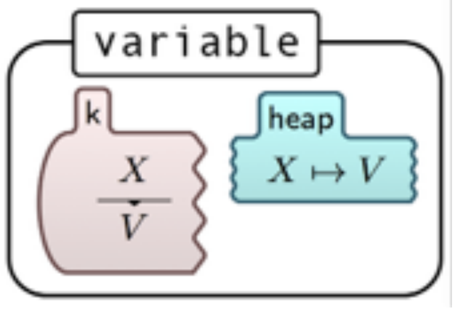

variable
k
X
V
heap
$X \mapsto V$

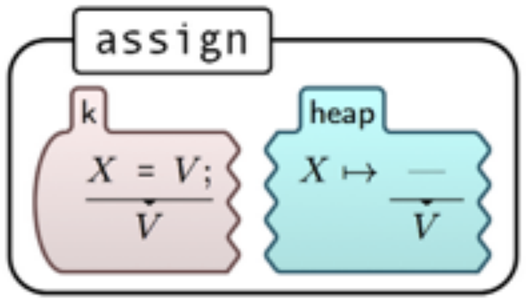

assign
k
$X = V;$
V
heap
$X \mapsto$ —
V

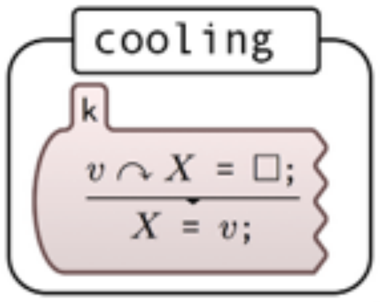

cooling
k
$v \curvearrowright X = \square;$
$X = v;$

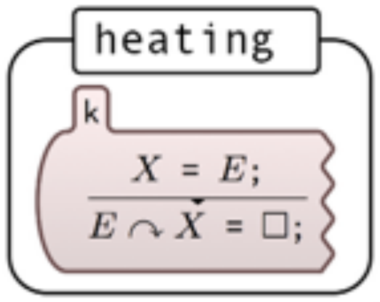

heating
k
$X = E;$
$E \curvearrowright X = \square;$


T
k
Prog : K
heap
m : Var → Value

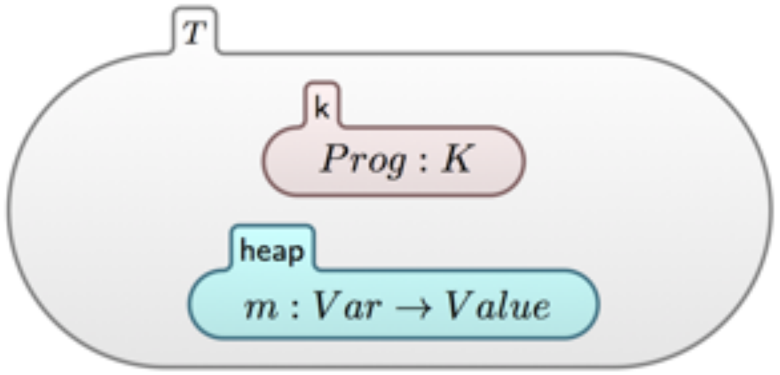

variable
k
X
V
heap
X ↦ V

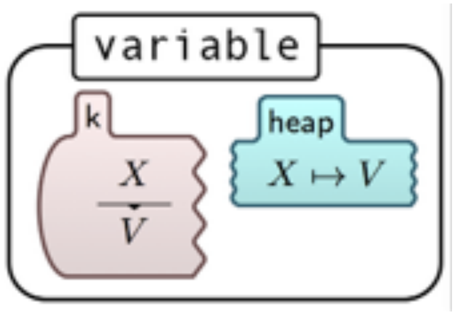

assign
k
X = V;
V
heap
X ↦ —
V

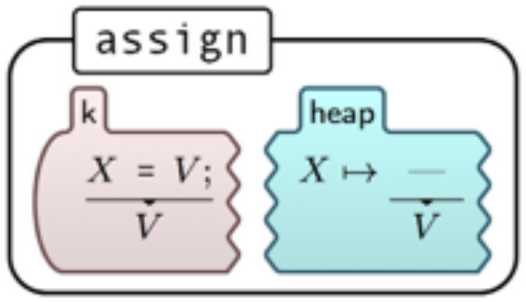

cooling
k
v ↷ X = □;
X = v;

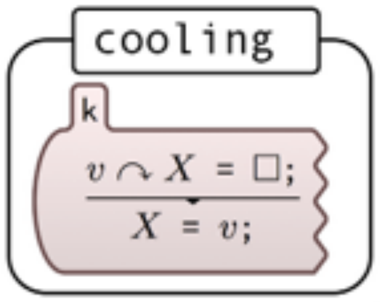

heating
k
X = E;
E ↷ X = □;

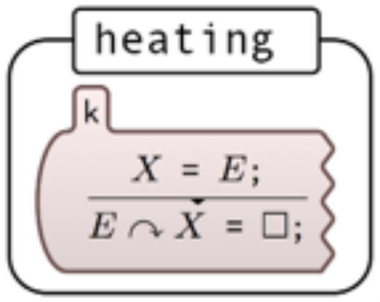
```
$x = $y
```


T
k
Prog : K
heap
m : Var → Value

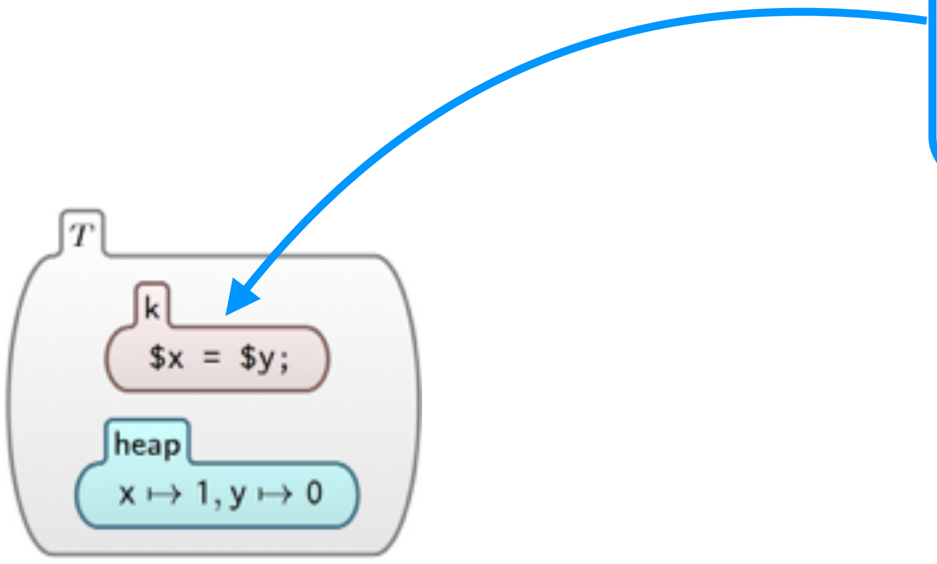

variable
k
X
V
heap
X ↦ V

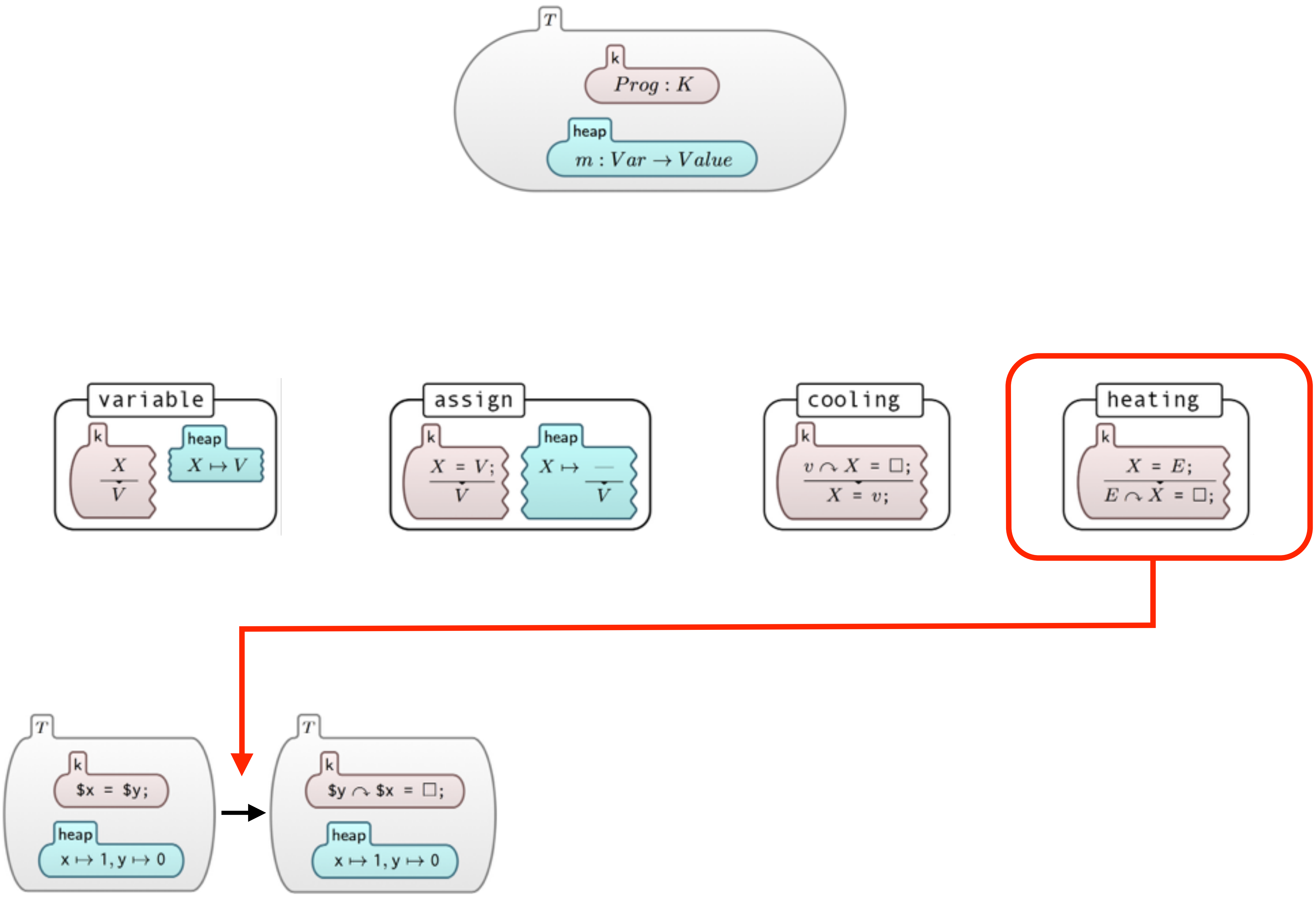

assign
k
X = V;
V
heap
X ↦ —
V

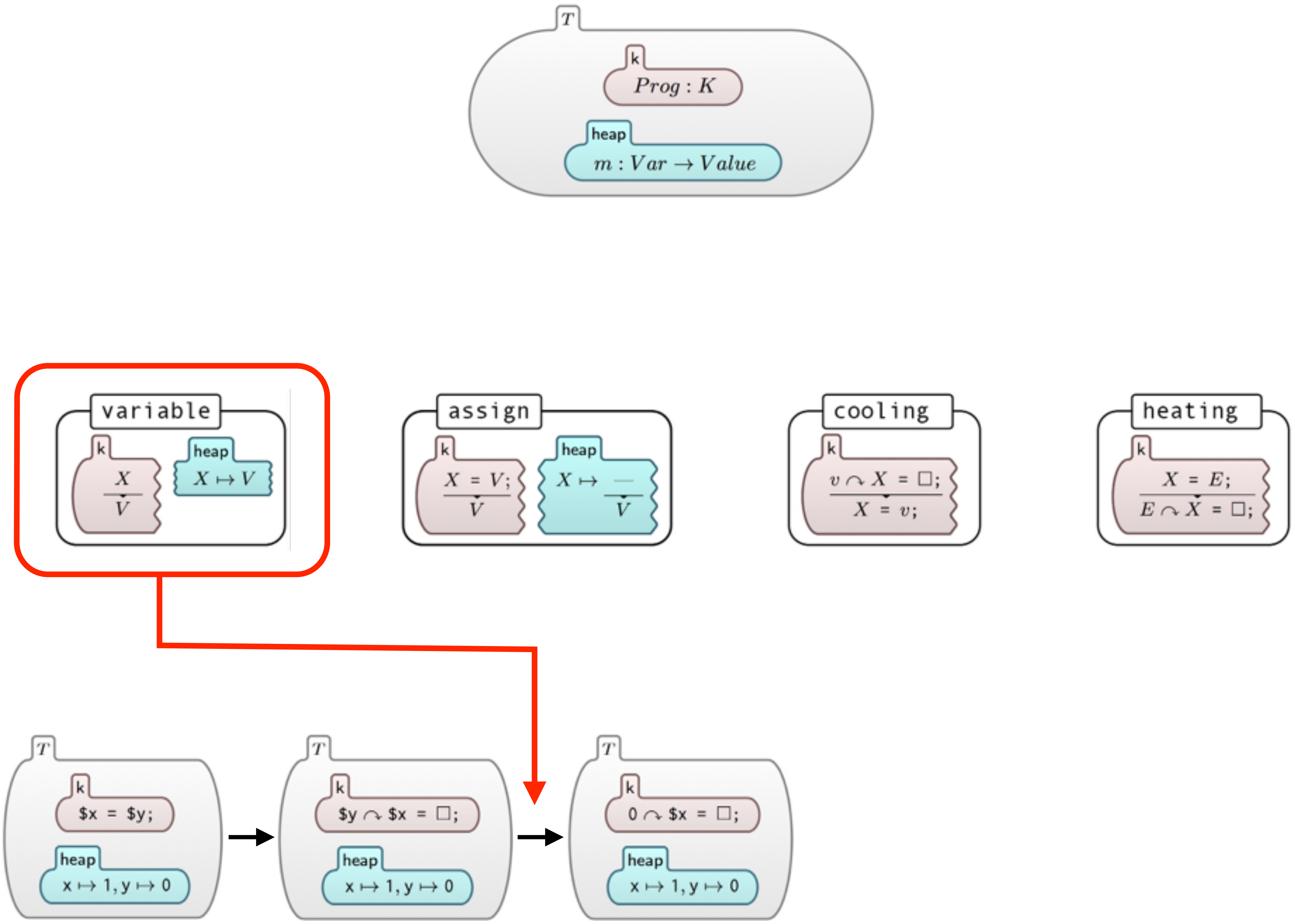

cooling
k
v ↷ X = □;
X = v;

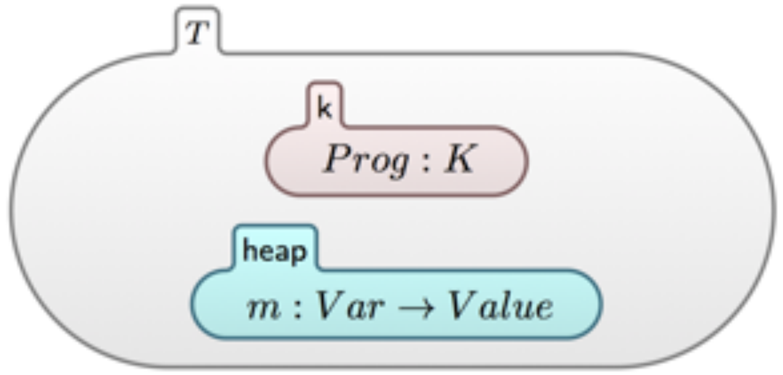

heating
k
X = E;
E ↷ X = □;

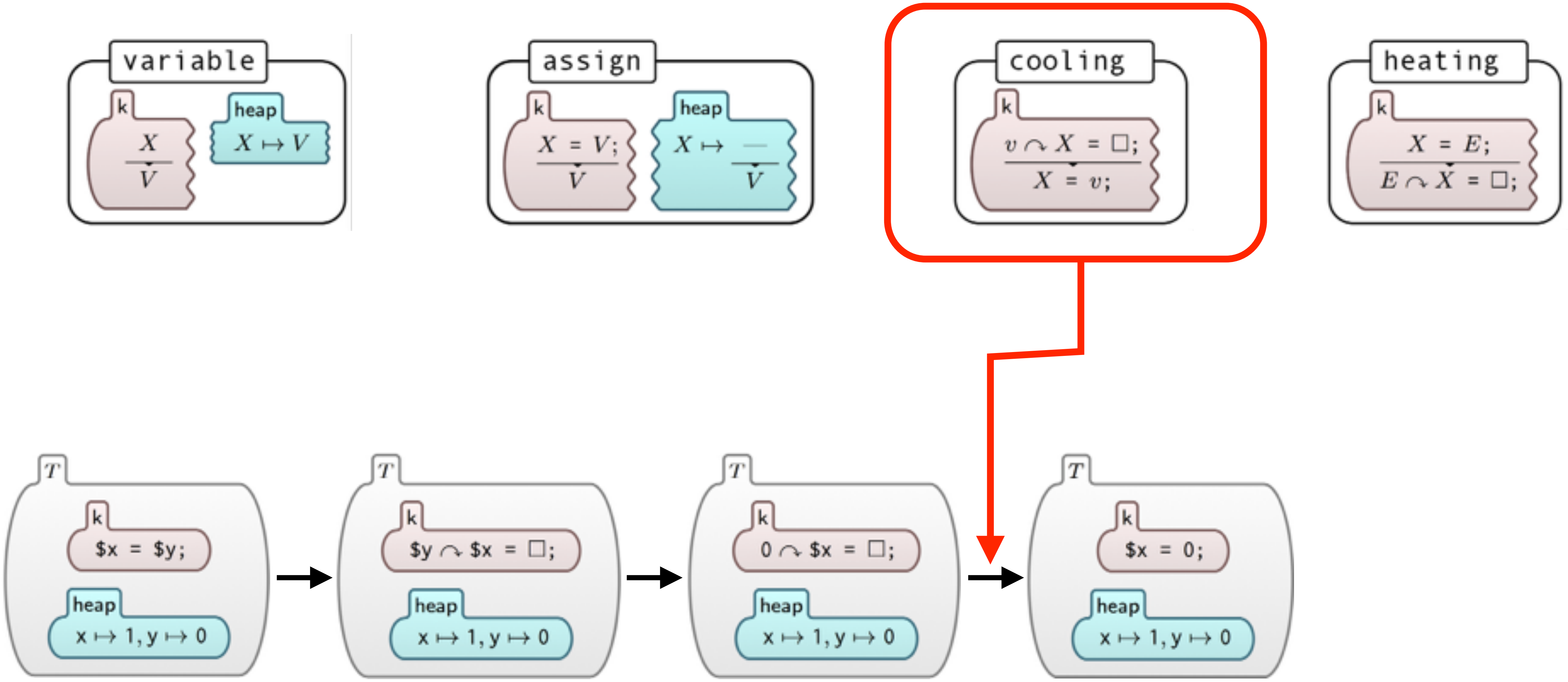
$x = $y

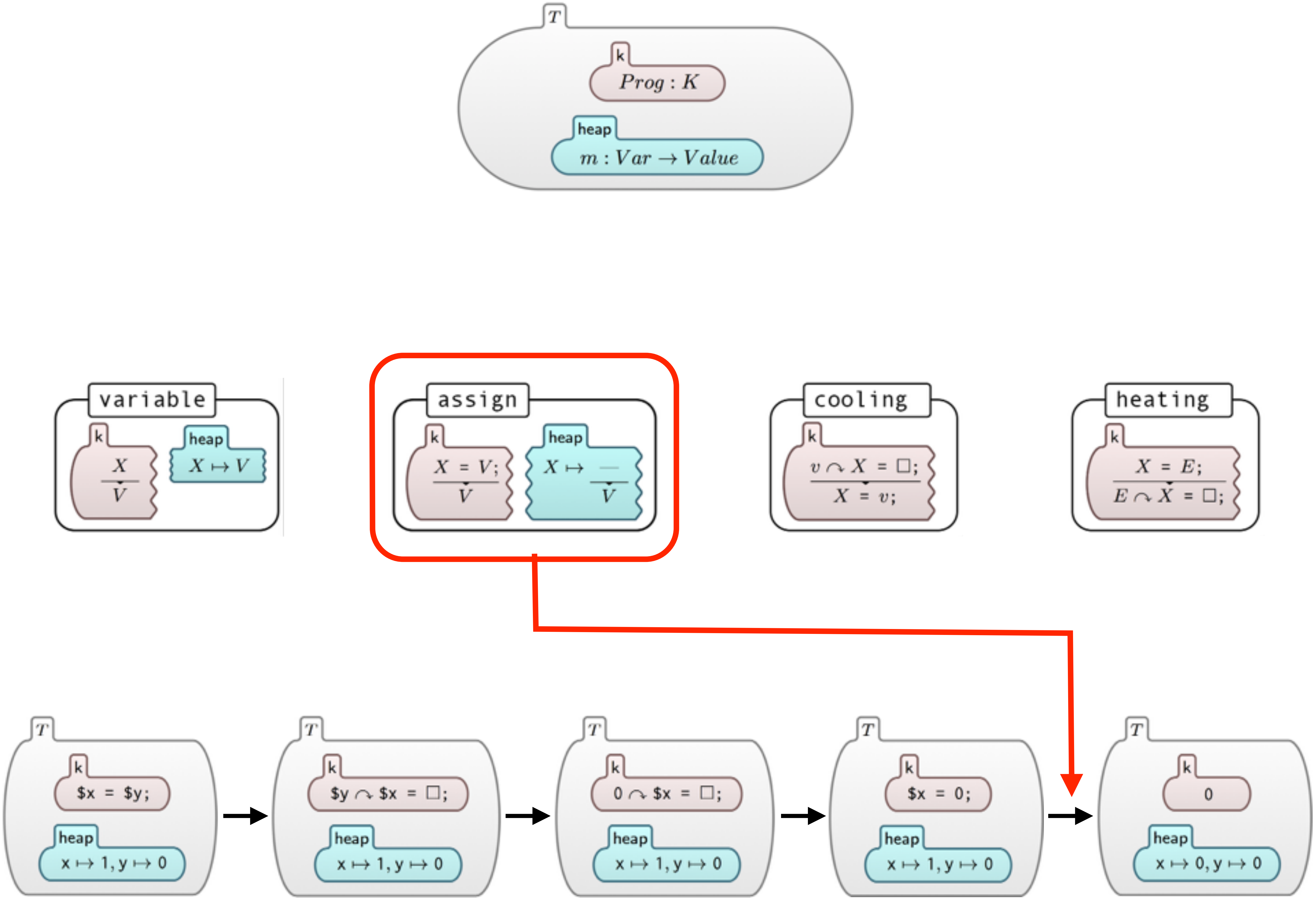

T
k
$x = $y;
heap
x ↦ 1, y ↦ 0


T
k
Prog : K
heap
m : Var → Value
variable
k
X / V
heap
X ↦ V
assign
k
X = V; / V
heap
X ↦ — / V
cooling
k
v ↷ X = □; / X = v;
heating
k
X = E; / E ↷ X = □;
T
k
$x = $y;
heap
x ↦ 1, y ↦ 0
T
k
$y ↷ $x = □;
heap
x ↦ 1, y ↦ 0


T
k
Prog : K
heap
m : Var → Value
variable
k
X / V
heap
X ↦ V
assign
k
X = V; / V
heap
X ↦ — / V
cooling
k
v ↷ X = □; / X = v;
heating
k
X = E; / E ↷ X = □;
T
k
$x = $y;
heap
x ↦ 1, y ↦ 0
T
k
$y ↷ $x = □;
heap
x ↦ 1, y ↦ 0
T
k
0 ↷ $x = □;
heap
x ↦ 1, y ↦ 0


T
k
Prog : K
heap
m : Var → Value

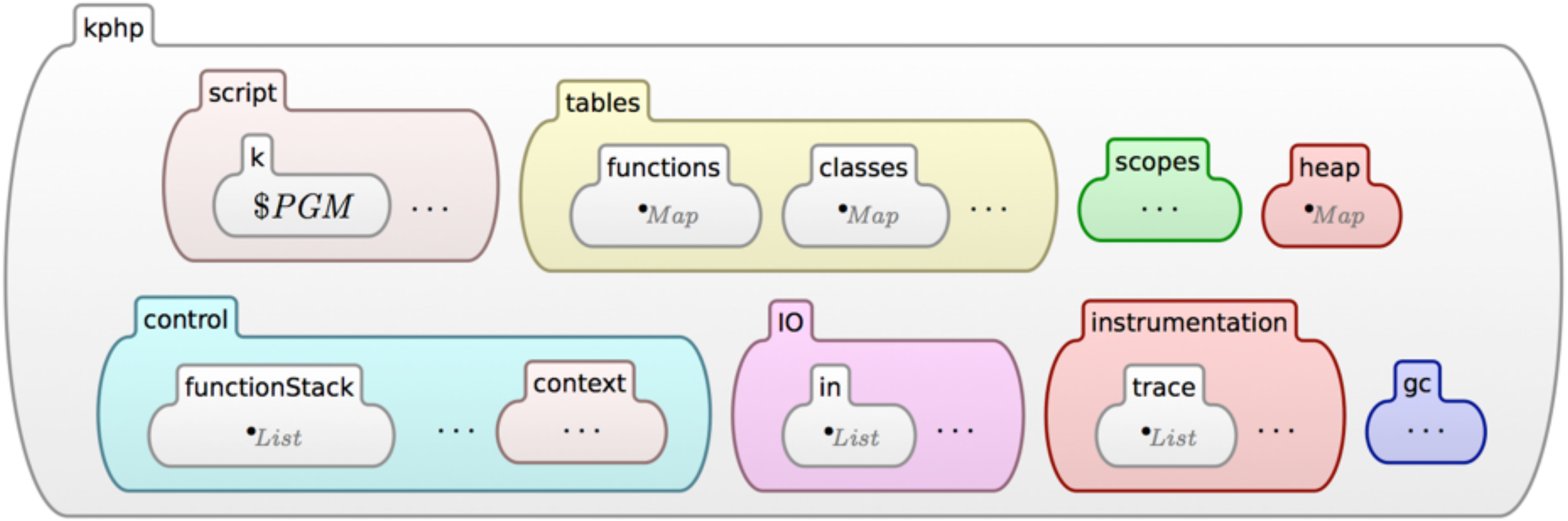

variable
k
X / V
heap
X ↦ V
assign
k
X = V; / V
heap
X ↦ — / V
cooling
k
v ↷ X = □; / X = v;
heating
k
X = E; / E ↷ X = □;
T
k
$x = $y;
heap
x ↦ 1, y ↦ 0
T
k
$y ↷ $x = □;
heap
x ↦ 1, y ↦ 0
T
k
0 ↷ $x = □;
heap
x ↦ 1, y ↦ 0
T
k
$x = 0;
heap
x ↦ 1, y ↦ 0


T
k
Prog : K
heap
m : Var → Value
variable
assign
cooling
heating
$x = $y;
$y ↷ $x = □;
0 ↷ $x = □;
$x = 0;
0
x ↦ 1, y ↦ 0
x ↦ 0, y ↦ 0

KPHP
(Formalising PHP in K)

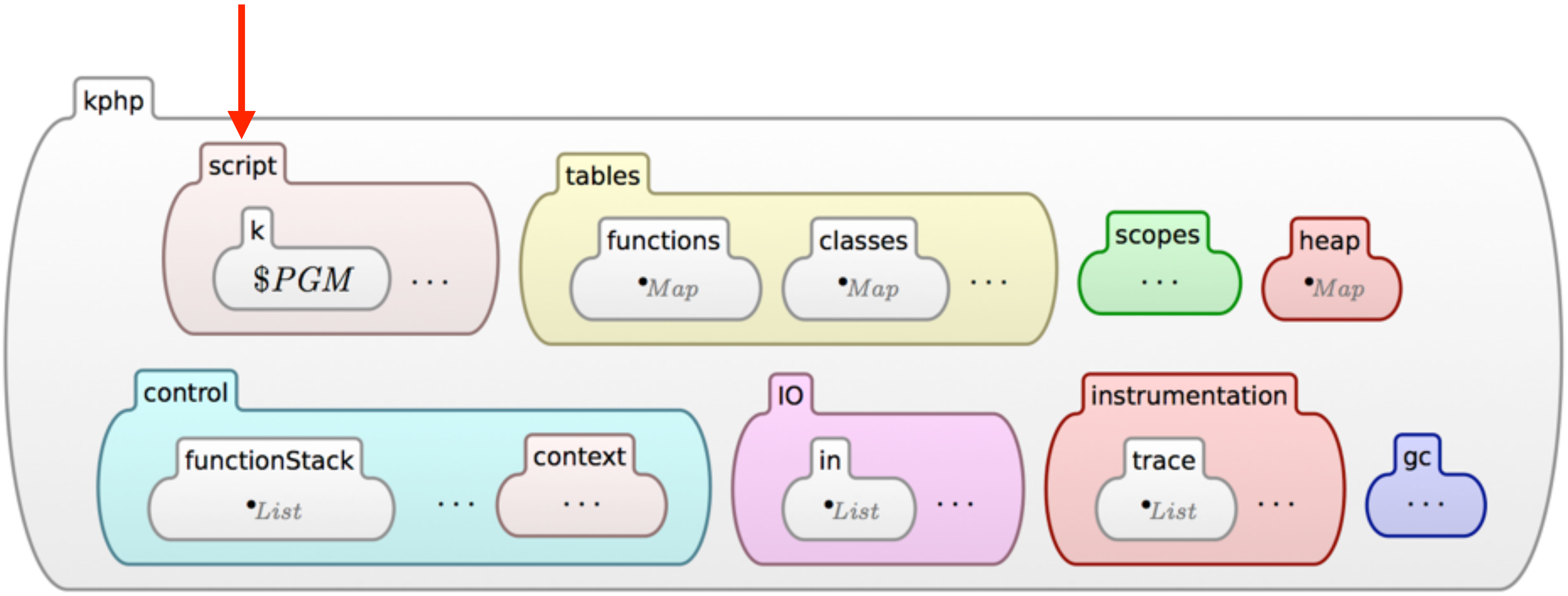
[ECOOP 2014]

Configuration (~30 cells)

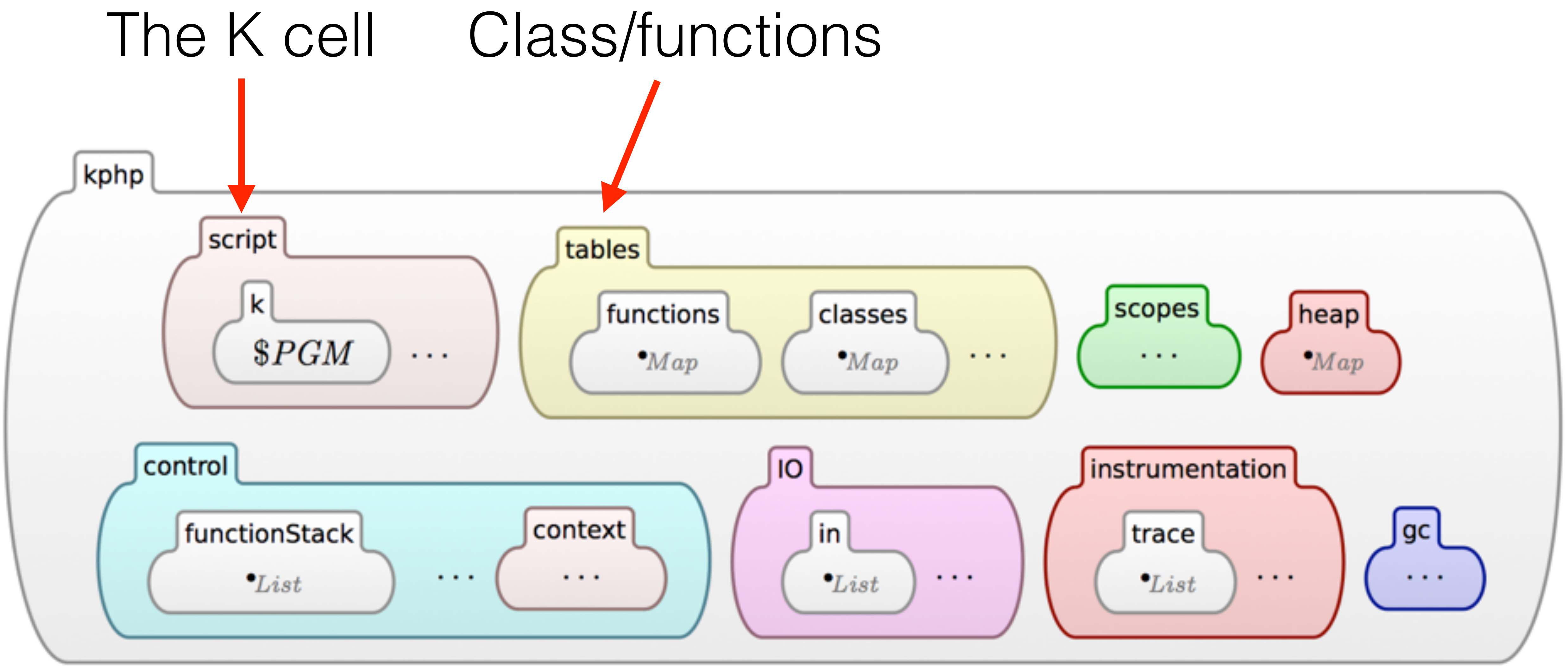


Configuration (~30 cells)

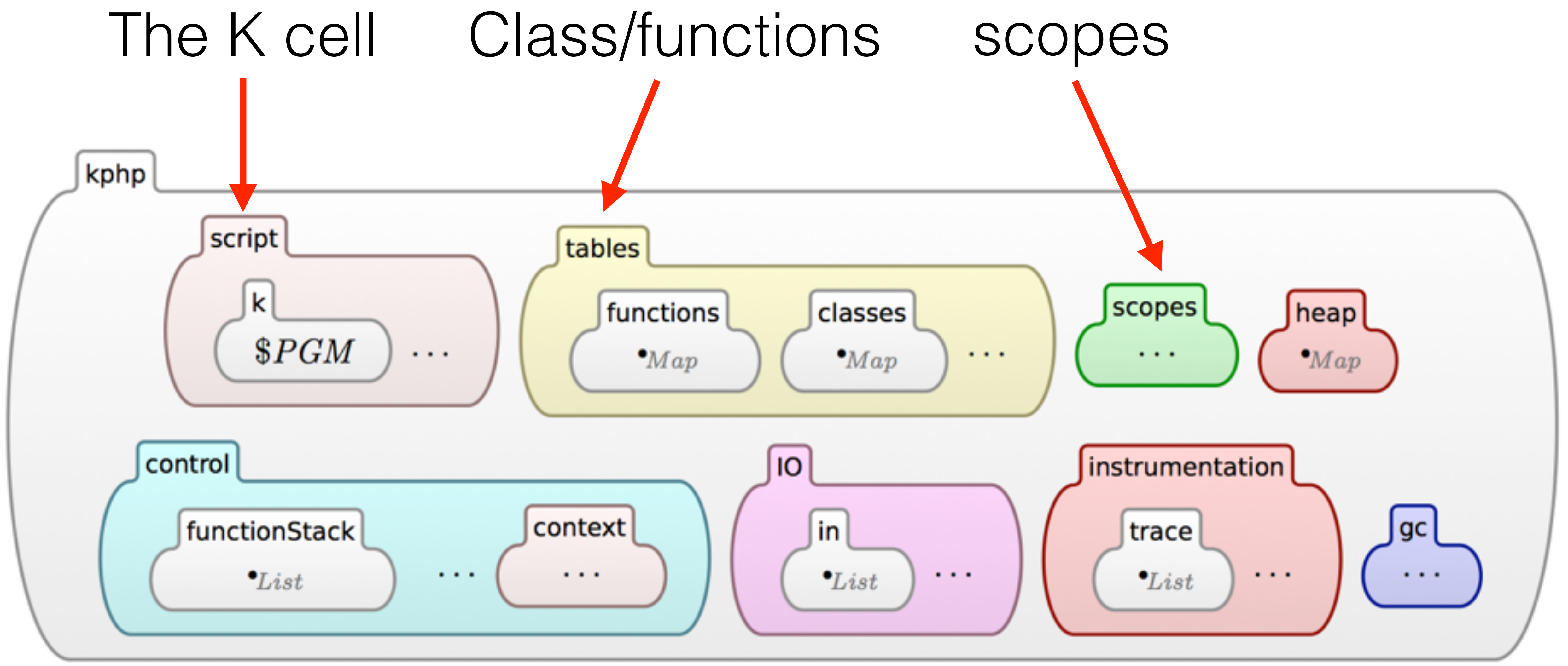
The K cell



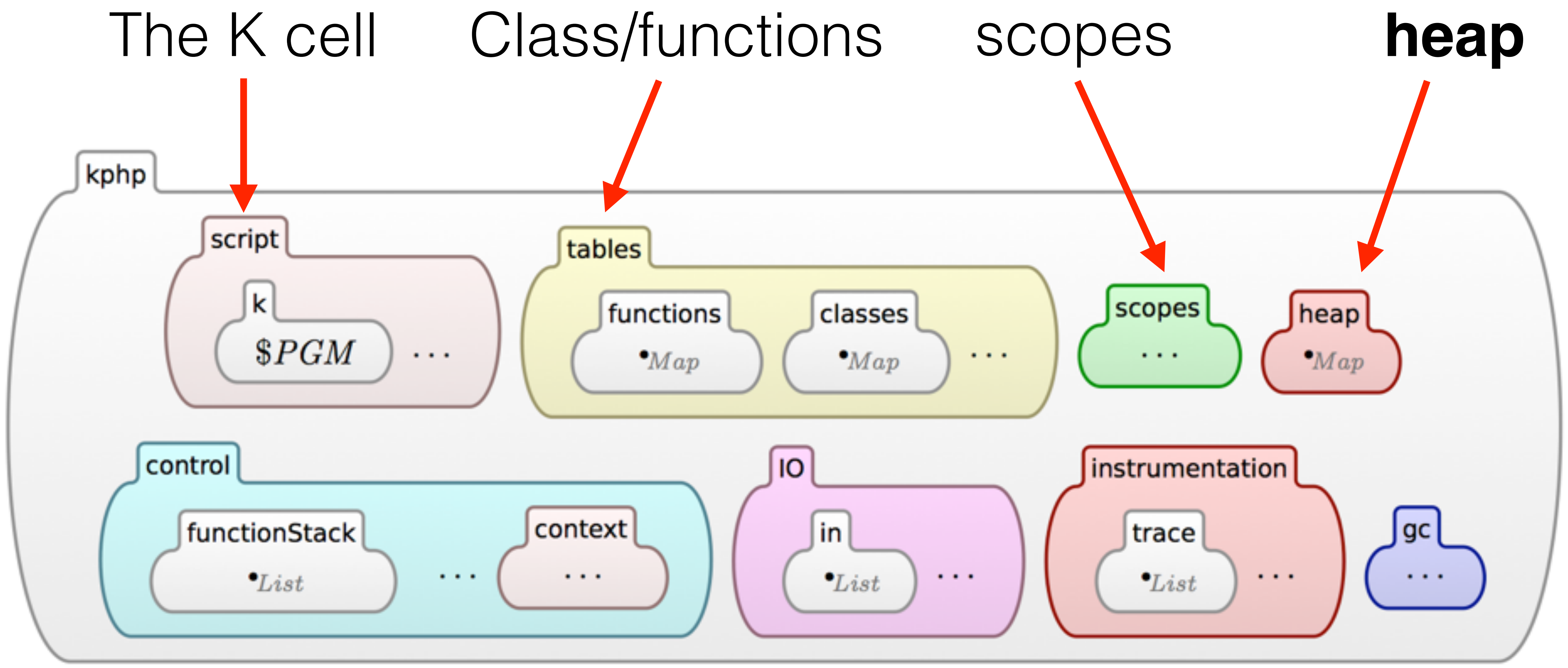
Configuration (~30 cells)



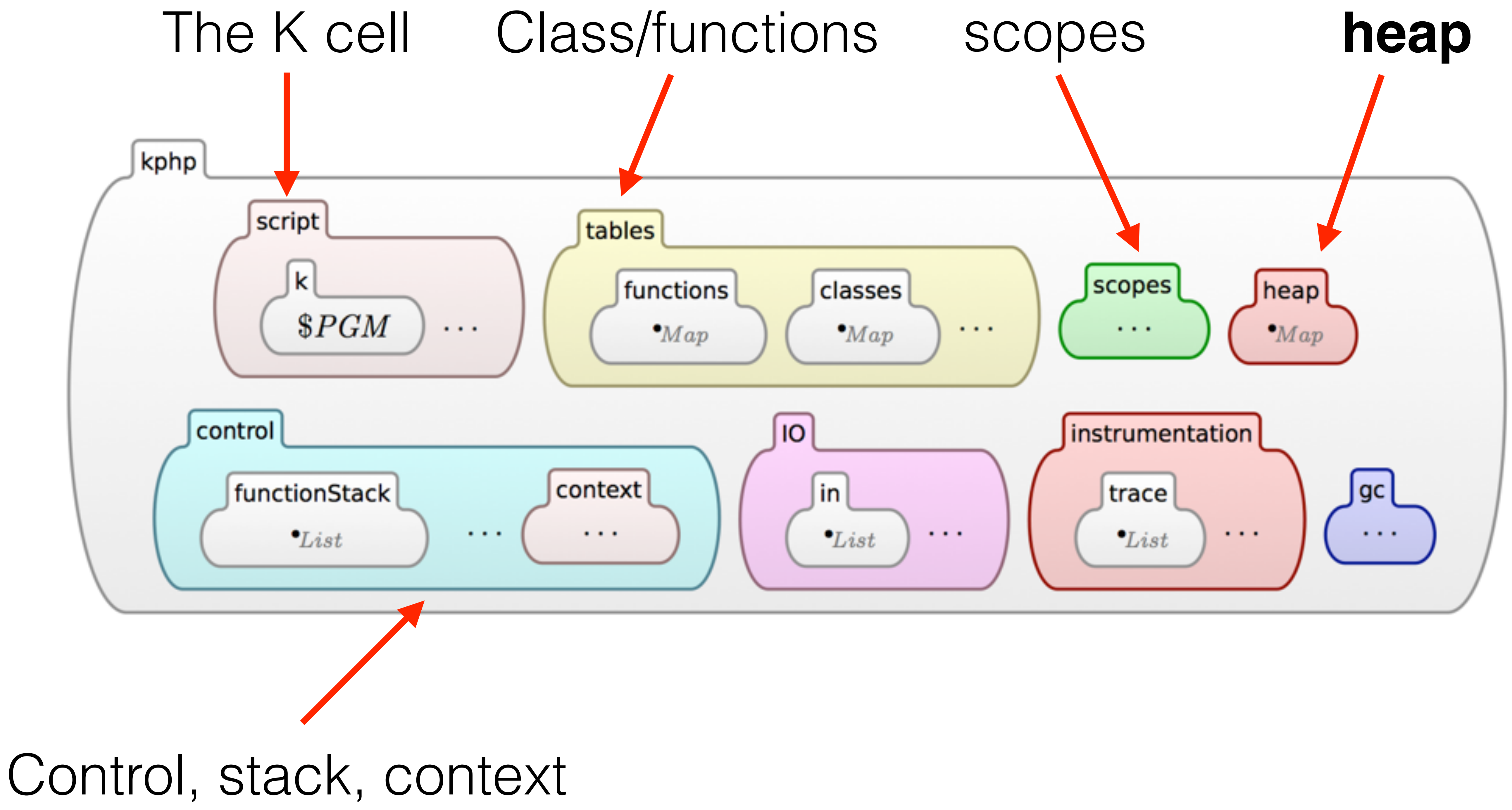
Configuration (~30 cells)



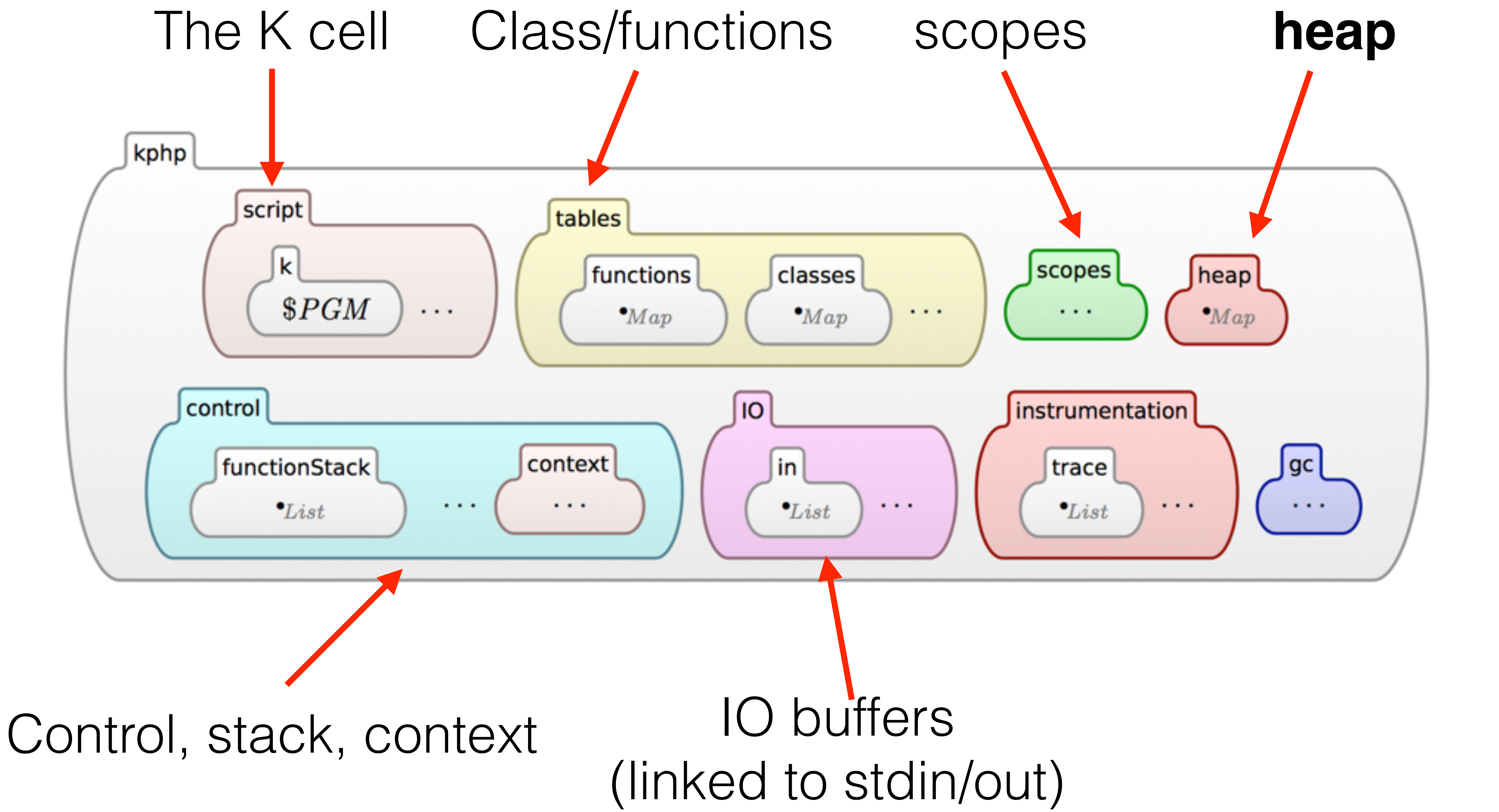
Configuration (~30 cells)



Configuration (~30 cells)

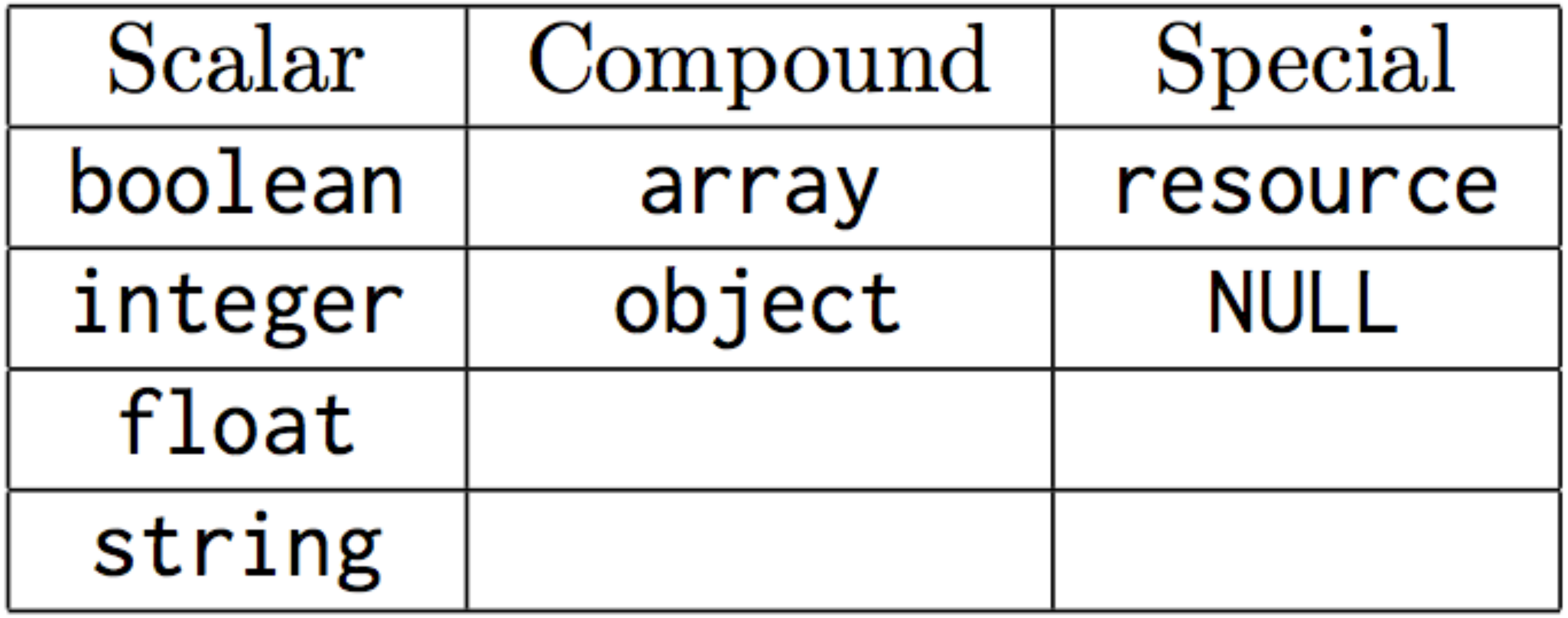


Configuration (~30 cells)



Memory Layout

Language Values



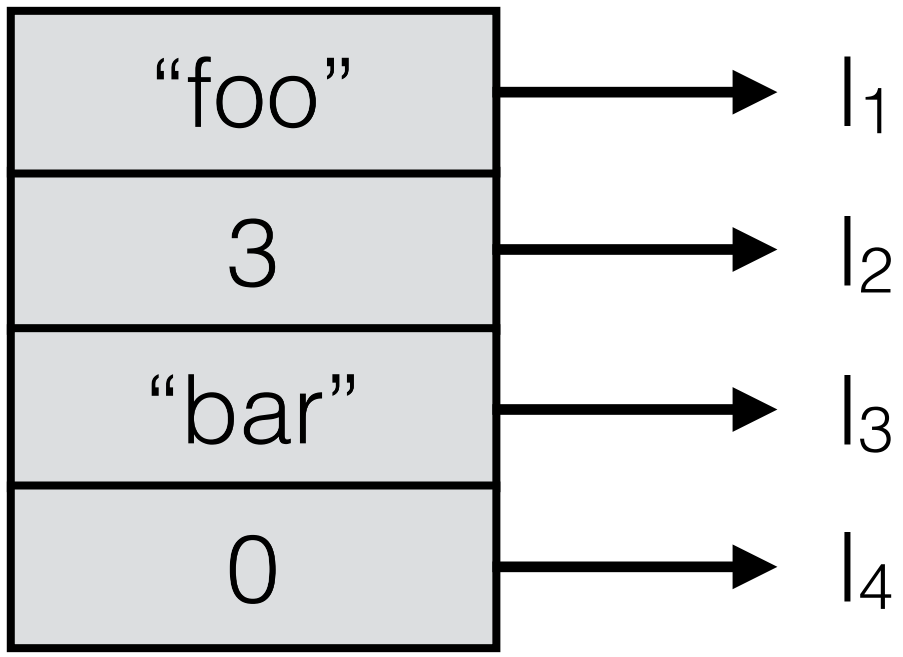
Scalar	Compound	Special
`boolean`	`array`	`resource`
`integer`	`object`	`NULL`
`float`		
`string`		

Language Values

Scalar	Compound	Special
boolean	array	~~resource~~
integer	object	NULL
float		
string		

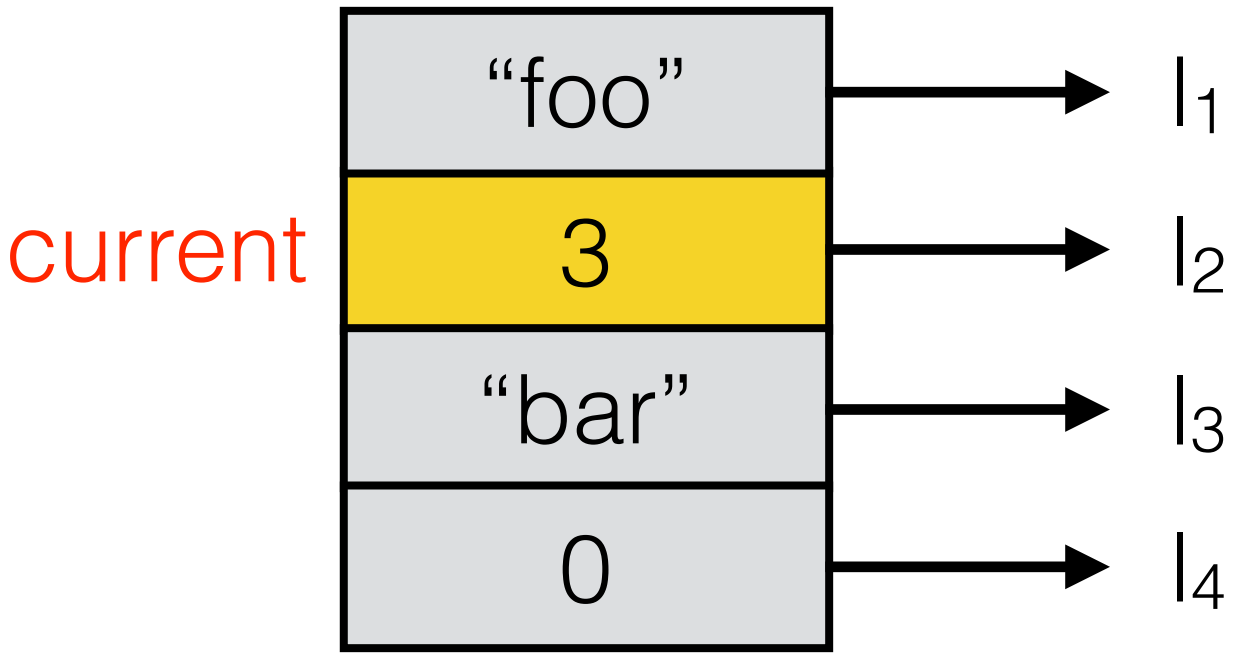
Arrays

Int U String —> Locations

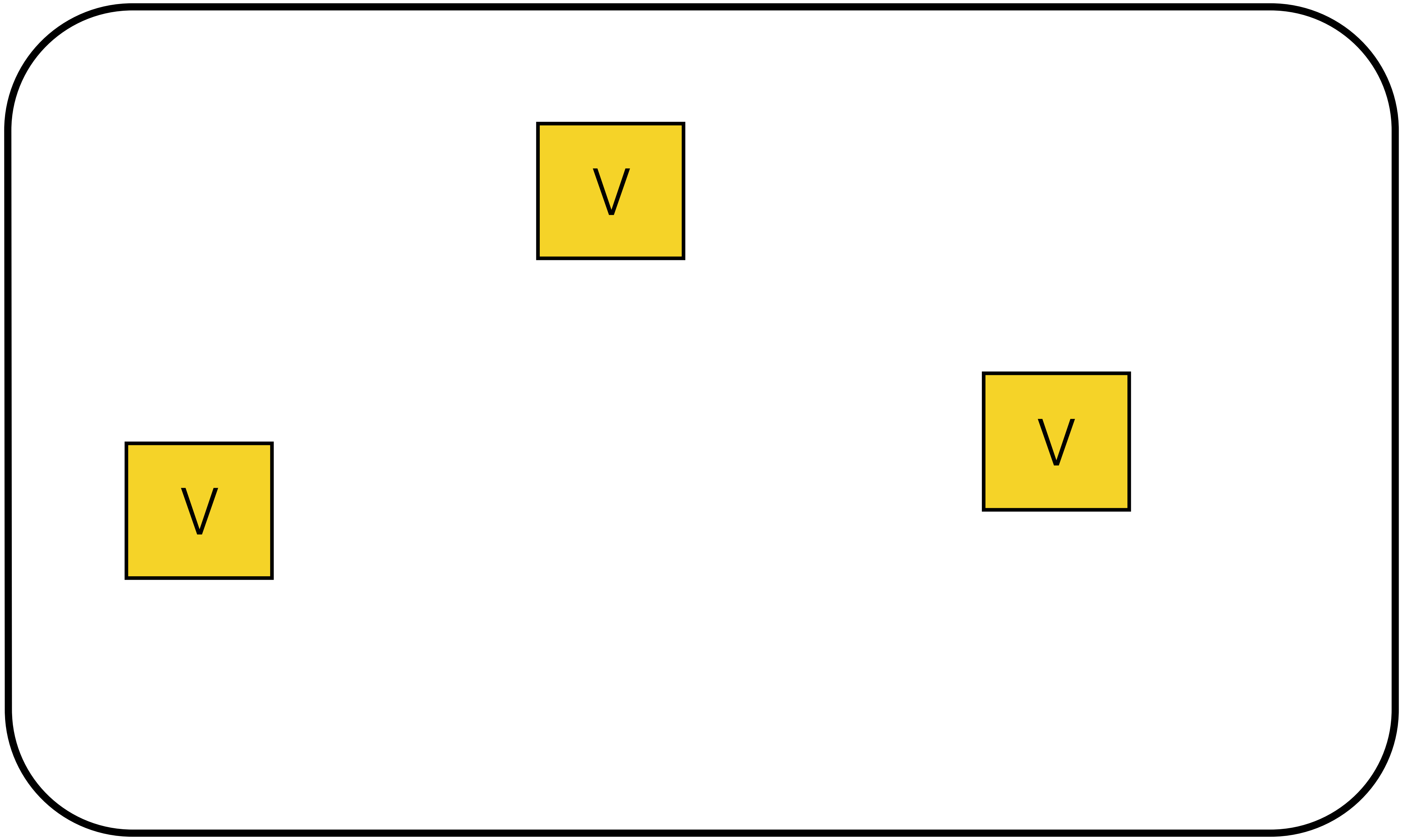


Arrays

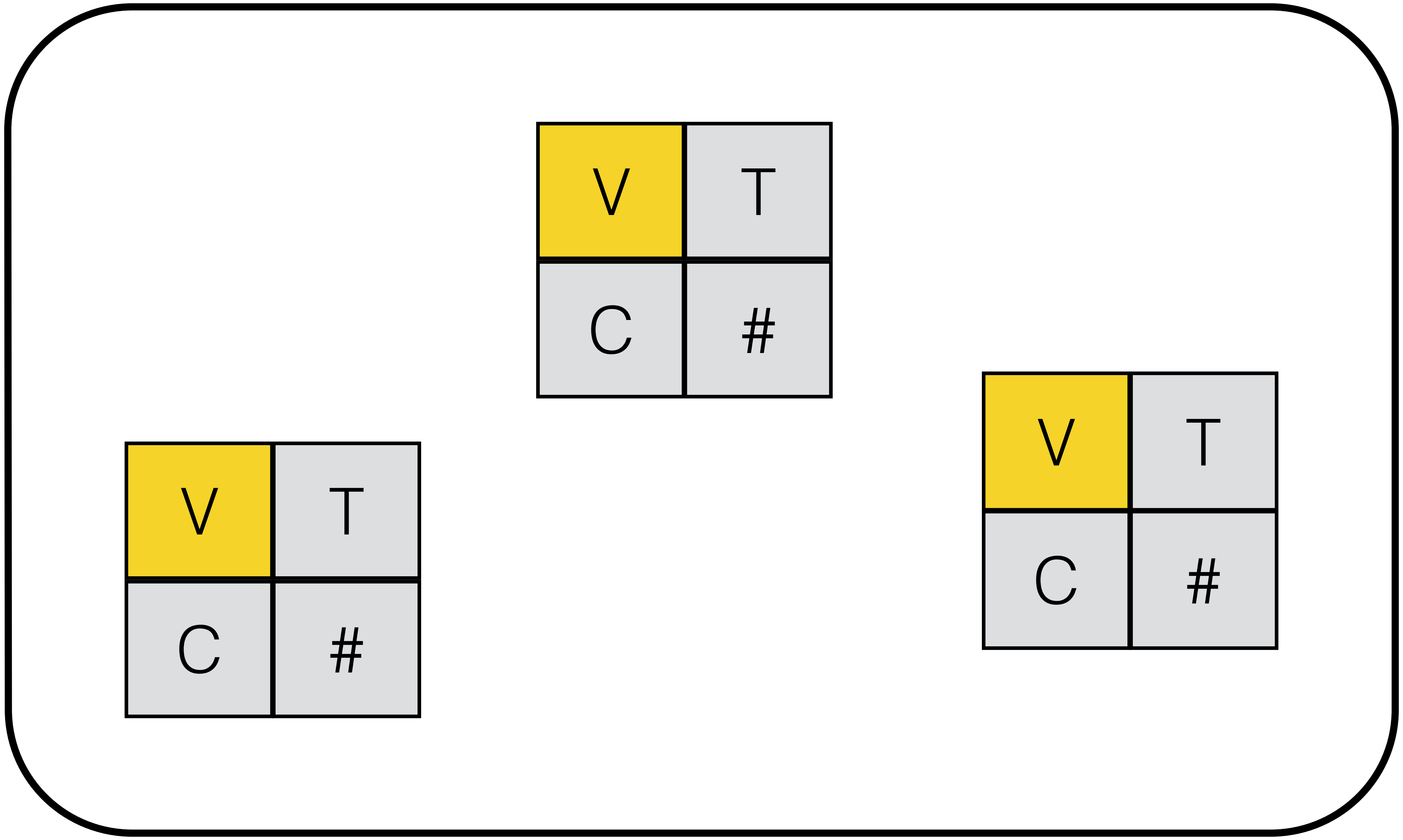
Int U String —> Locations



Values and Z-Values

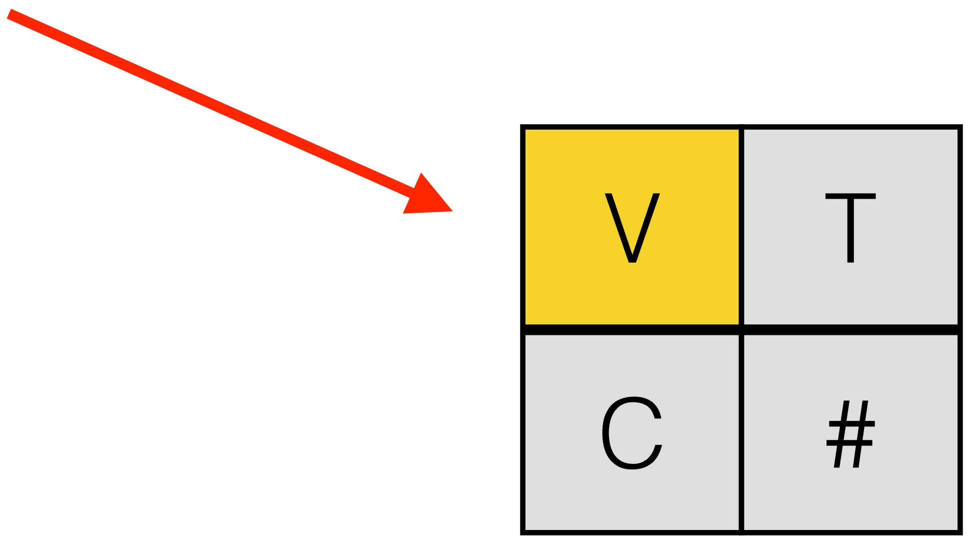


Values and **Z-Values**

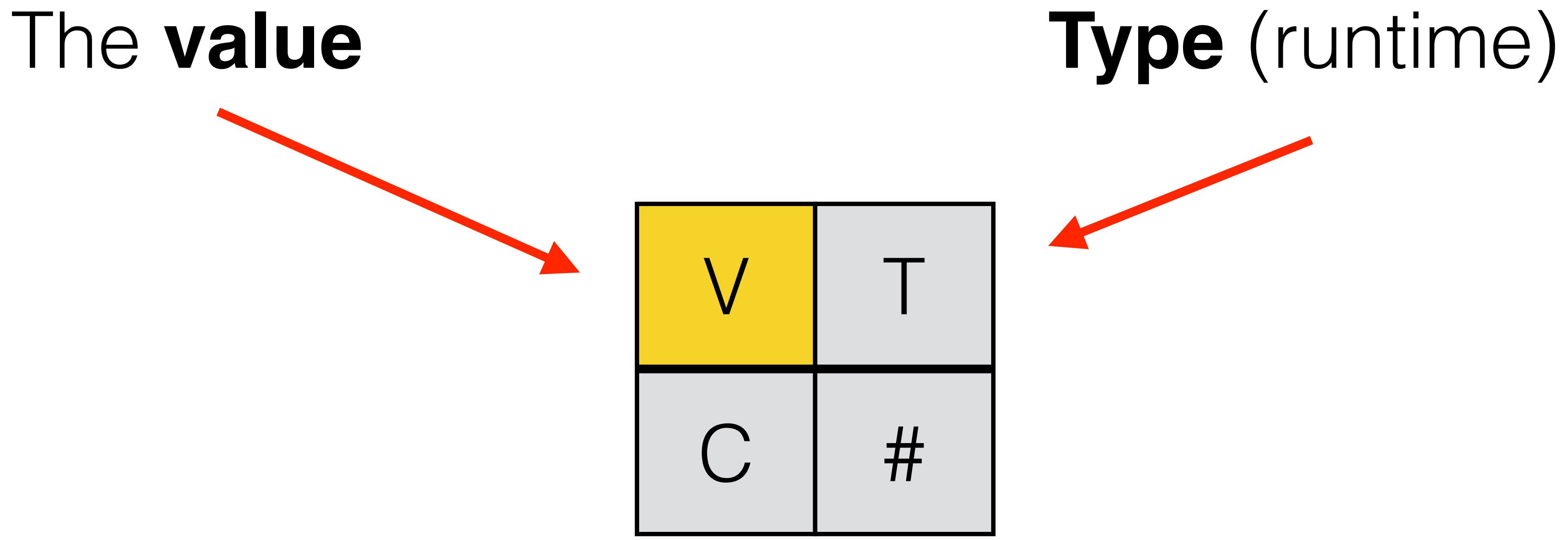


Z-Values

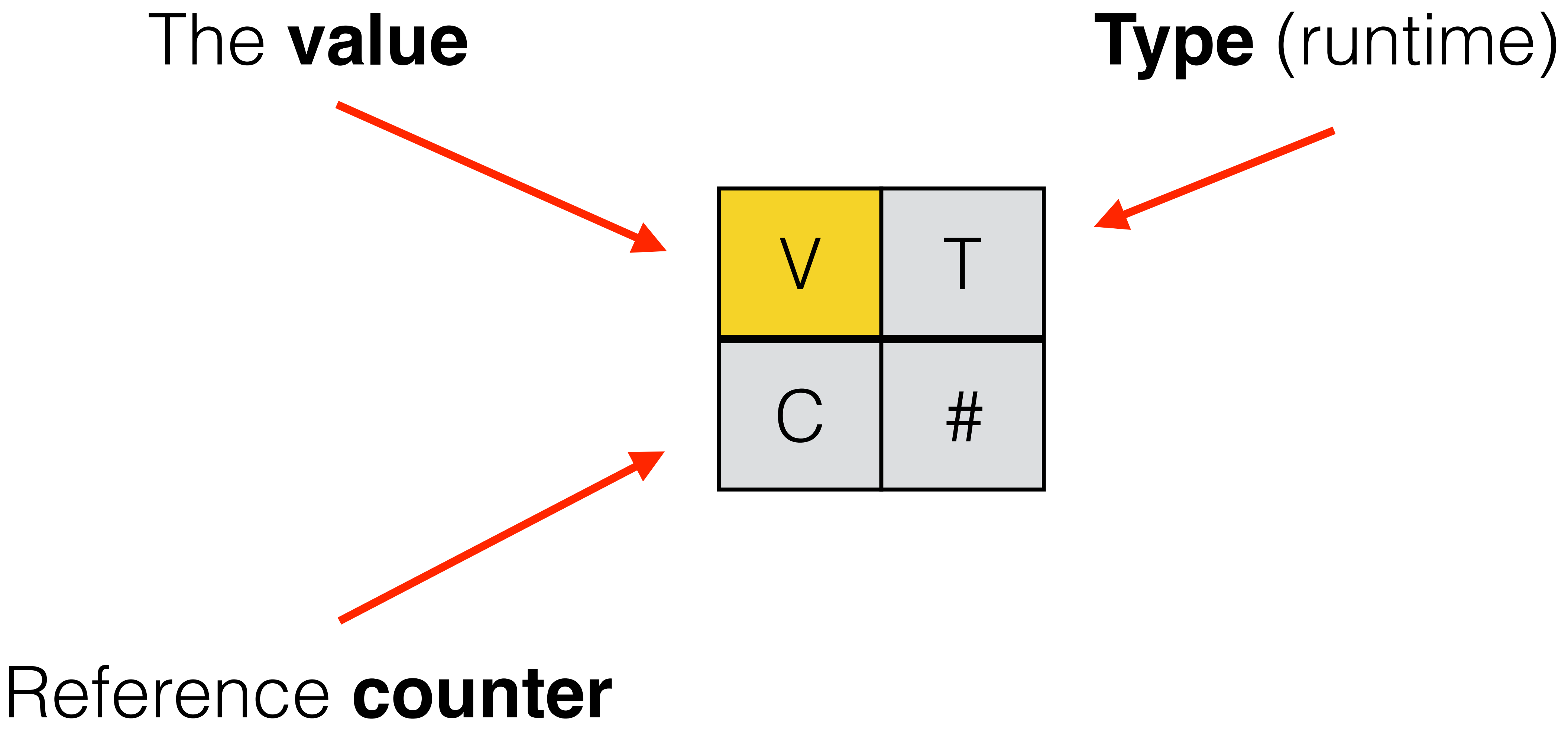
The **value**



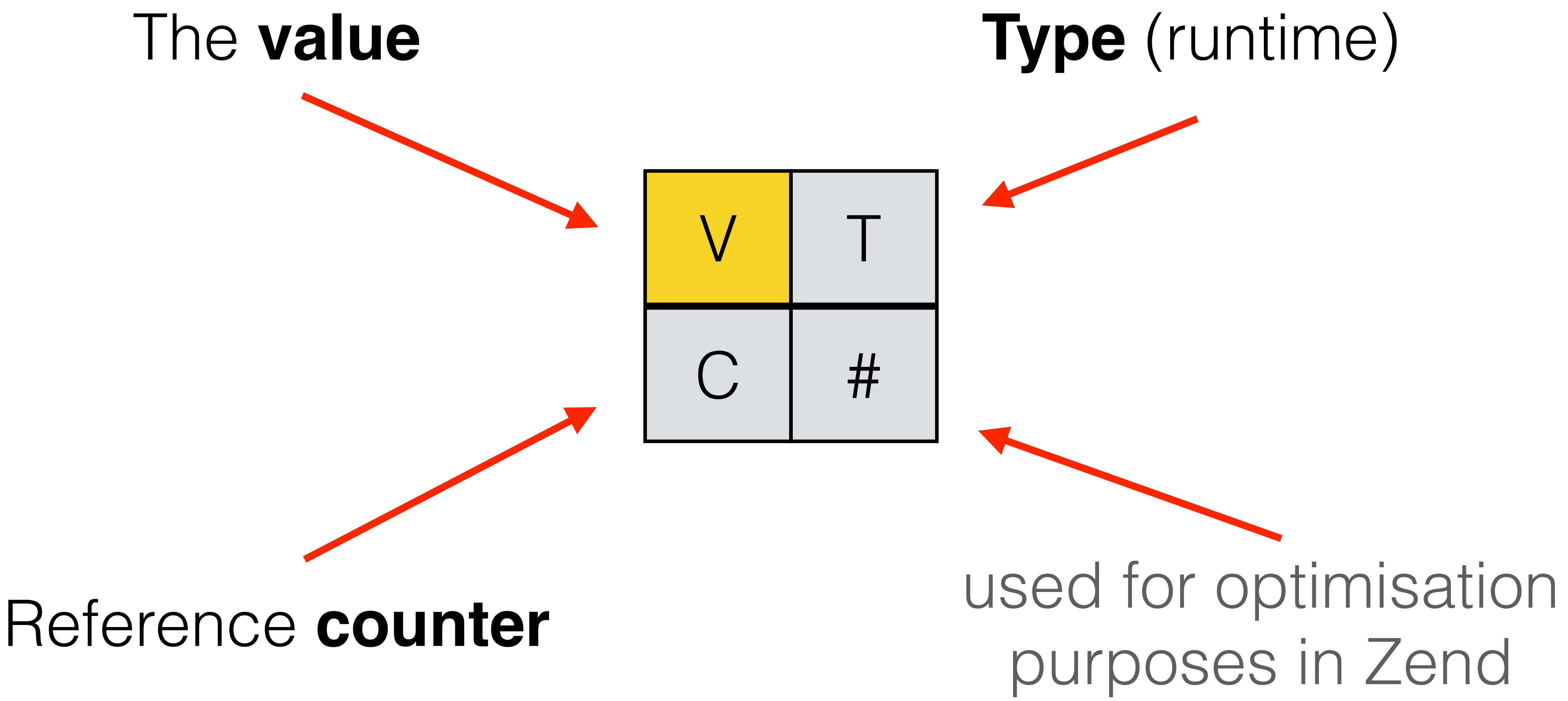
Z-Values



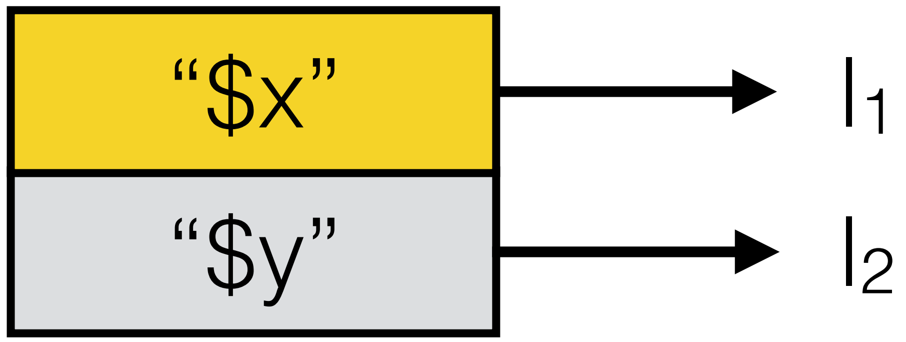
Z-Values



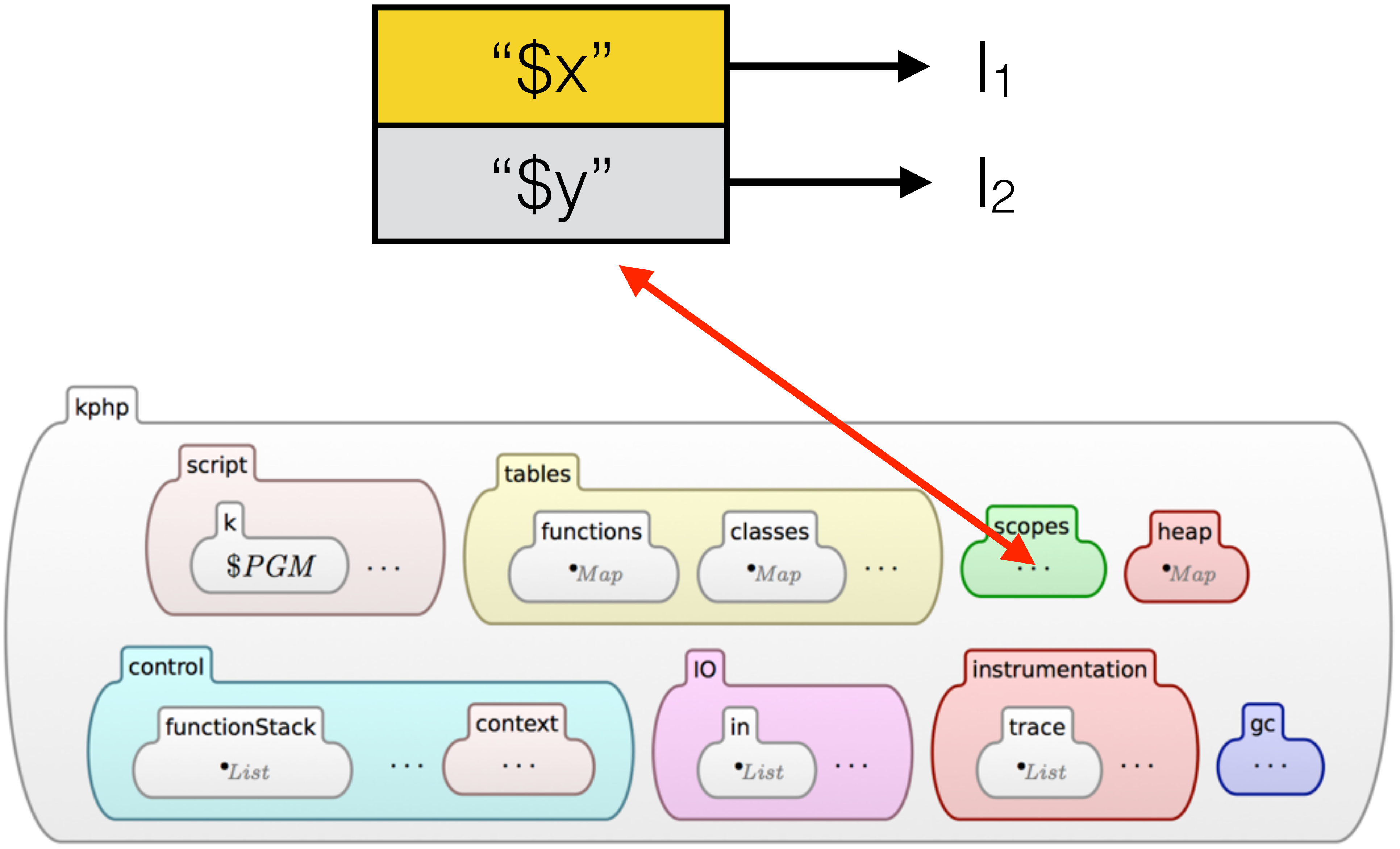
Z-Values


The **value**
Type (runtime)
V
T
C
#
Reference **counter**
used for optimisation purposes in Zend

Environments via arrays

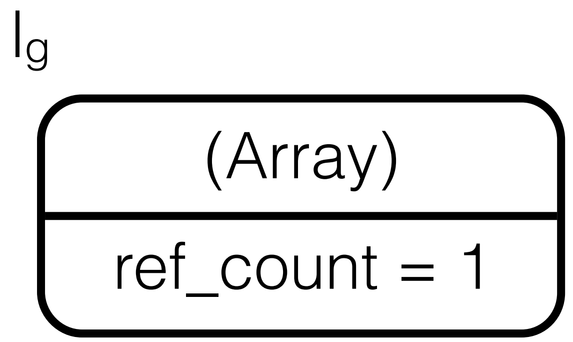


Environments via arrays



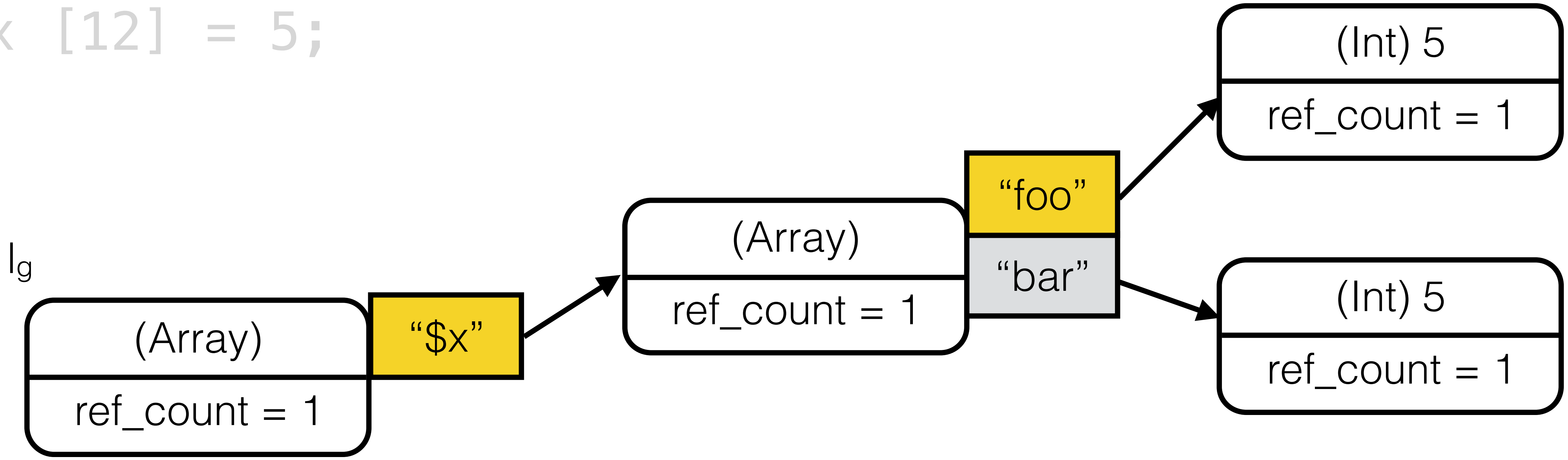
Put it all together

```
$x = array("foo" => 5, "bar" => 5); $y = 5;
next($x);
$x["baz"] = &$x["bar"];
$x [12] = 5;
```



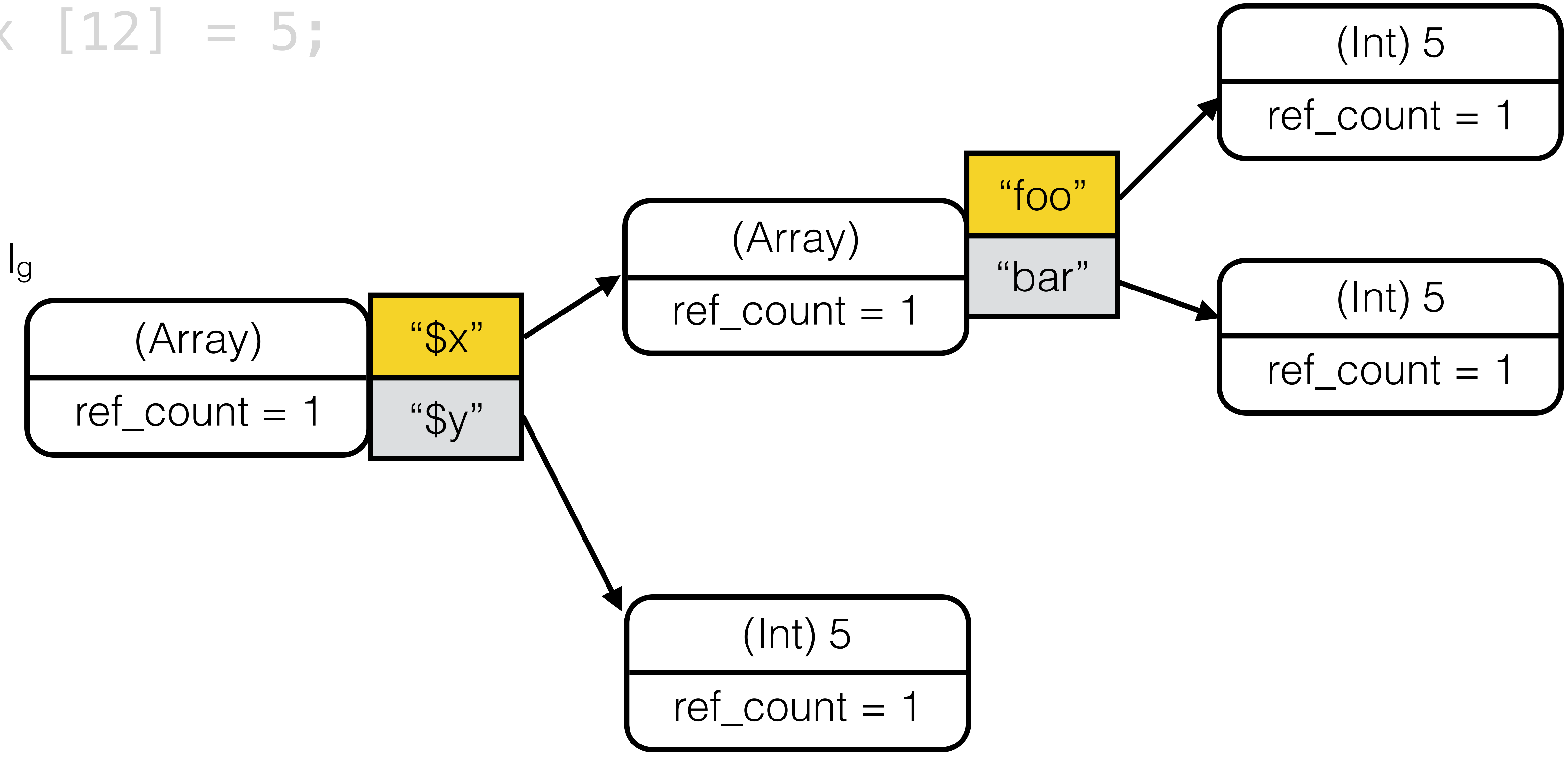
Put it all together

```
$x = array("foo" => 5, "bar" => 5); $y = 5;
next($x);
$x["baz"] = &$x["bar"];
$x [12] = 5;
```



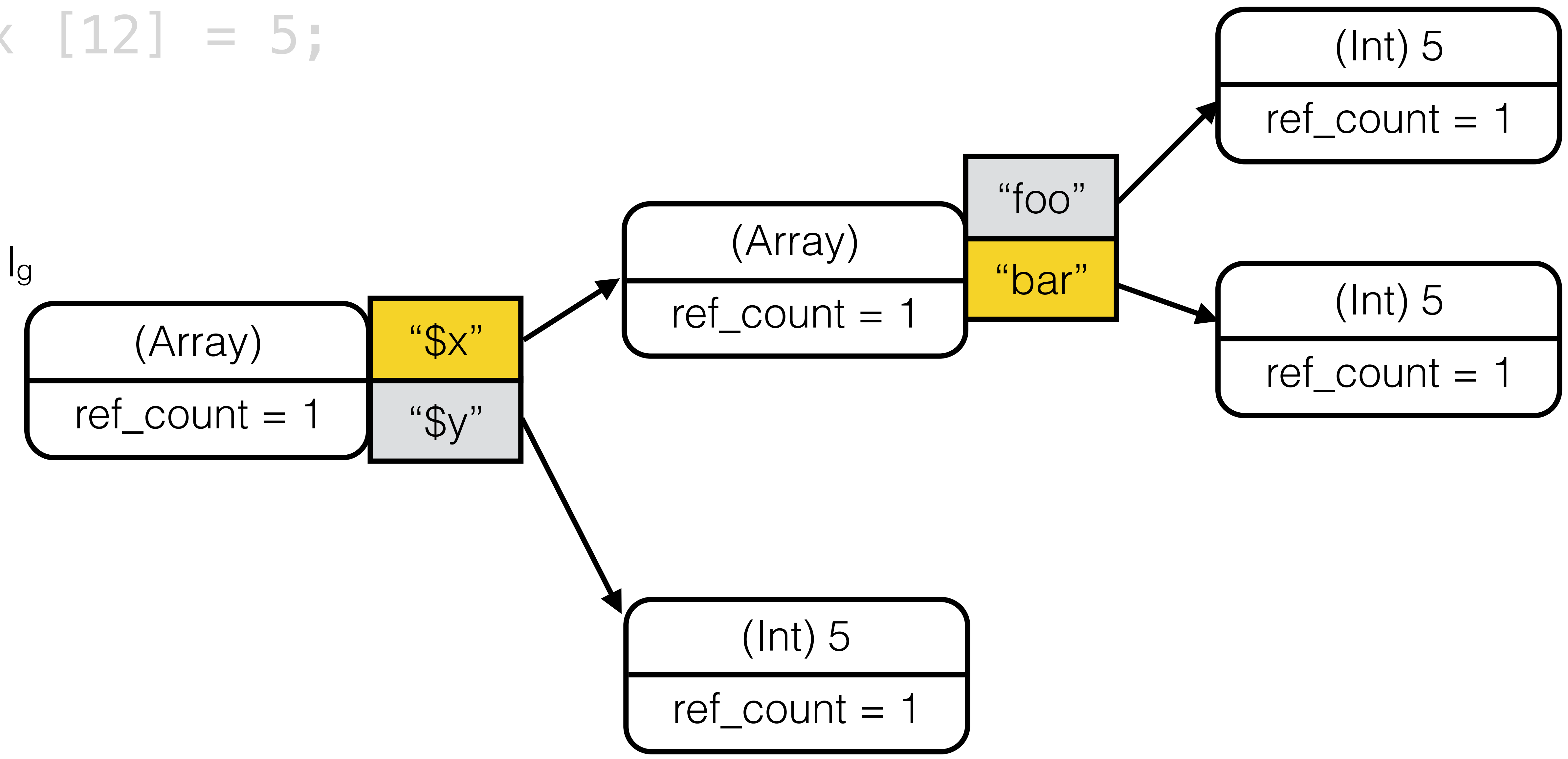
Put it all together

```
$x = array("foo" => 5, "bar" => 5); $y = 5;
next($x);
$x["baz"] = &$x["bar"];
$x [12] = 5;
```



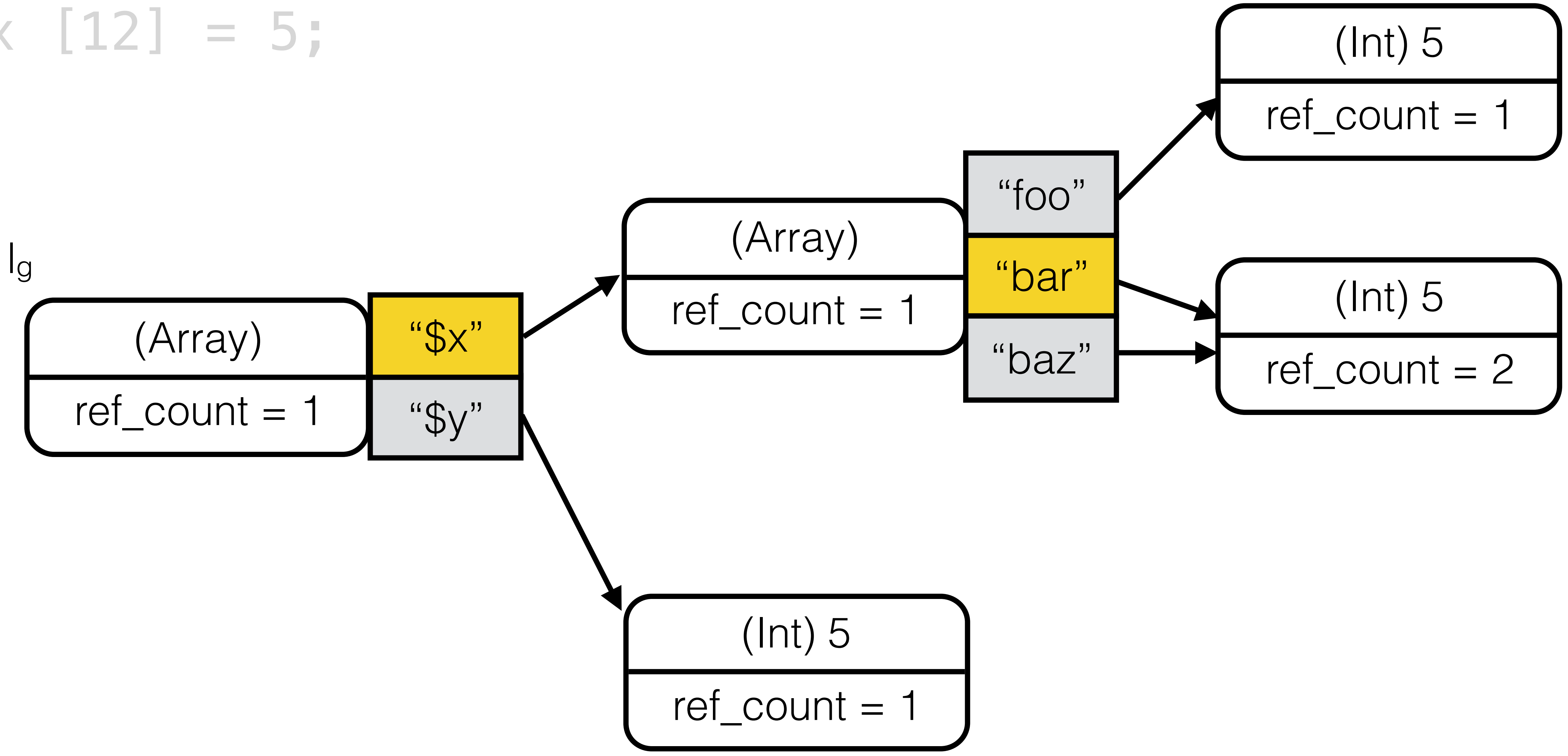
Put it all together

```
$x = array("foo" => 5, "bar" => 5); $y = 5;
next($x);
$x["baz"] = &$x["bar"];
$x [12] = 5;
```



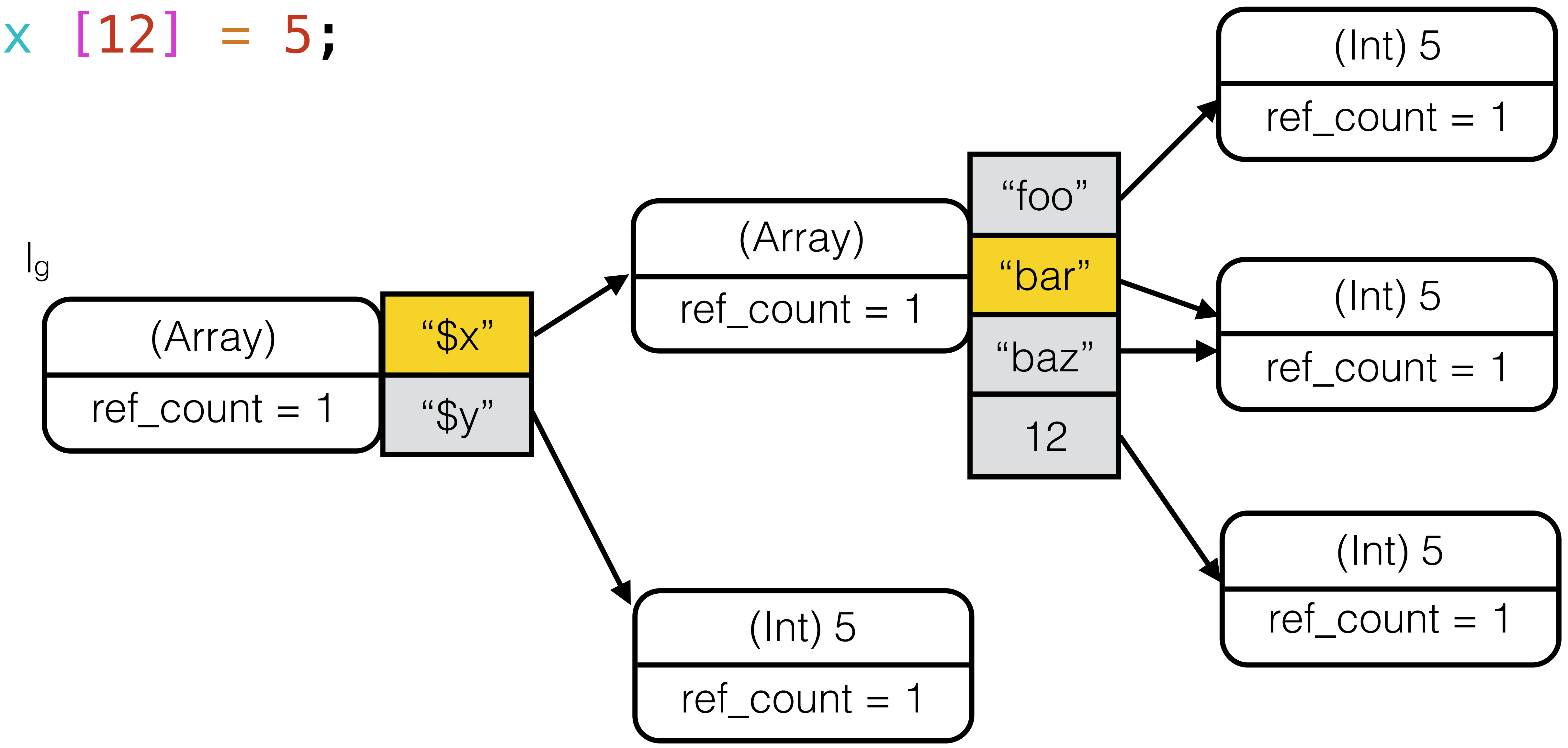
Put it all together

```
$x = array("foo" => 5, "bar" => 5); $y = 5;
next($x);
$x["baz"] = &$x["bar"];
$x [12] = 5;
```



Put it all together

```
$x = array("foo" => 5, "bar" => 5); $y = 5;
next($x);
$x["baz"] = &$x["bar"];
$x [12] = 5;
```



Internal values

- Locations: $\mathbf{l_1, l_2, l_3}$....
- References: **ref(l, “x”)**

Internal values

$$\begin{tikzcd}
\boxed{\$x} \arrow[d, bend right, "\text{evaluation to a reference}"'] \\
\boxed{\mathrm{ref}(l_g, \text{“x”})} \arrow[d, bend left, "\text{convert to location}"] \\
\boxed{l_1} \arrow[d, bend right, "\text{convert to language value}"'] \\
\boxed{5}
\end{tikzcd}$$

Semantic rules: numbers

- ~ 800 rules
- ~ 8000 LOC
- 29 *.k files

Layers

Low-level rules
(copy values, inc. ref. counter, update scope etc.)

Layers

Language **features**
(e.g.: assignment, function call)

Low-level rules
(copy values, inc. ref. counter, update scope etc.)

Layers

Derived Construct
(e.g. x++ —> x = x + 1)

Language **features**
(e.g.: assignment, function call)

Low-level rules
(copy values, inc. ref. counter, update scope etc.)

Example: assignment

(A) CONTEXT `'Assign(□,_)`

(B) CONTEXT `'Assign(_:KResult,□)`

(C) $\texttt{'Assign}\left(\dfrac{\texttt{R:Ref}}{\texttt{convertToLoc(R)}}, \_\right)$ `[intermediate]`

(D) $\dfrac{\texttt{'Assign(L:Loc, V:Value)}}{\texttt{copyValueToLoc(V, L)}\curvearrowright\texttt{V}}$ `[step]`

(E) $\texttt{'Assign}\left(\_ : \texttt{KResult}, \dfrac{\texttt{V:ConvertibleToLoc}}{\texttt{convertToLoc(V,r)}}\right)$

`[intermediate]`

(F) $\dfrac{\texttt{'Assign(L:Loc,L1:Loc)}}{\texttt{reset(L1)} \curvearrowright \texttt{'Assign(L, L1)}}$

`when currentOverflow(L1)` `[intermediate]`

(G) $\dfrac{\texttt{'Assign(L,L1)}}{\texttt{'Assign(L, convertToLanguageValue(L1))}}$

`when` $\neg$ `currentOverflow(L1)` `[intermediate]`

Example: assignment

(A) `CONTEXT  'Assign(□,_)`

(B) `CONTEXT  'Assign(_:KResult,□)`

(C) $\texttt{'Assign}\left(\frac{\texttt{R:Ref}}{\texttt{convertToLoc(R)}}, \_\right)$ `[intermediate]`

(D) $\frac{\texttt{'Assign(L:Loc, V:Value)}}{\texttt{copyValueToLoc(V, L)} \curvearrowright \texttt{V}}$ `[step]`

(E) $\texttt{'Assign}\left(\_ : \texttt{KResult}, \frac{\texttt{V:ConvertibleToLoc}}{\texttt{convertToLoc(V,r)}}\right)$

`[intermediate]`

(F) $\frac{\texttt{'Assign(L:Loc,L1:Loc)}}{\texttt{reset(L1)} \curvearrowright \texttt{'Assign(L, L1)}}$

`when currentOverflow(L1)    [intermediate]`

(G) $\frac{\texttt{'Assign(L,L1)}}{\texttt{'Assign(L, convertToLanguageValue(L1))}}$

`when` $\neg$ `currentOverflow(L1)    [intermediate]`

Example: assignment

(A) CONTEXT $\texttt{'Assign(}\square\texttt{,\_)}$

(B) CONTEXT $\texttt{'Assign(\_:KResult,}\square\texttt{)}$

(C) $$\texttt{'Assign}\left(\frac{\texttt{R:Ref}}{\texttt{convertToLoc(R)}}, \_\right) \quad \texttt{[intermediate]}$$

(D) $$\frac{\texttt{'Assign(L:Loc, V:Value)}}{\texttt{copyValueToLoc(V, L)} \curvearrowright \texttt{V}} \quad \texttt{[step]}$$

(E) $$\texttt{'Assign}\left(\_ : \texttt{KResult}, \frac{\texttt{V:ConvertibleToLoc}}{\texttt{convertToLoc(V,r)}}\right)$$
$\texttt{[intermediate]}$

(F) $$\frac{\texttt{'Assign(L:Loc,L1:Loc)}}{\texttt{reset(L1)} \curvearrowright \texttt{'Assign(L, L1)}}$$
$\texttt{when currentOverflow(L1)} \quad \texttt{[intermediate]}$

(G) $$\frac{\texttt{'Assign(L,L1)}}{\texttt{'Assign(L, convertToLanguageValue(L1))}}$$
$\texttt{when} \neg \texttt{currentOverflow(L1)} \quad \texttt{[intermediate]}$

Example: assignment

(A) `CONTEXT    'Assign(□,_)`

(B) `CONTEXT    'Assign(_:KResult,□)`

(C) $\texttt{'Assign}\left(\dfrac{\texttt{R:Ref}}{\texttt{convertToLoc(R)}}, \_\right)$ `[intermediate]`

(D) $\dfrac{\texttt{'Assign(L:Loc, V:Value)}}{\texttt{copyValueToLoc(V, L)} \curvearrowright \texttt{V}}$ `[step]`

(E) $\texttt{'Assign}\left(\_ : \texttt{KResult}, \dfrac{\texttt{V:ConvertibleToLoc}}{\texttt{convertToLoc(V,r)}}\right)$
`[intermediate]`

(F) $\dfrac{\texttt{'Assign(L:Loc,L1:Loc)}}{\texttt{reset(L1)} \curvearrowright \texttt{'Assign(L, L1)}}$
`when currentOverflow(L1)    [intermediate]`

(G) $\dfrac{\texttt{'Assign(L,L1)}}{\texttt{'Assign(L, convertToLanguageValue(L1))}}$
`when` $\neg$ `currentOverflow(L1)    [intermediate]`

Example: assignment

(A) CONTEXT $\quad$ `'Assign(□,_)`

(B) CONTEXT $\quad$ `'Assign(_:KResult,□)`

(C) $\texttt{'Assign}\left(\dfrac{\texttt{R:Ref}}{\texttt{convertToLoc(R)}},\texttt{\_}\right)$ [intermediate]

(D) $\dfrac{\texttt{'Assign(L:Loc, V:Value)}}{\texttt{copyValueToLoc(V, L)}\curvearrowright\texttt{V}}$ [step]

(E) $\texttt{'Assign}\left(\texttt{\_ : KResult},\dfrac{\texttt{V:ConvertibleToLoc}}{\texttt{convertToLoc(V,r)}}\right)$ [intermediate]

(F) $\dfrac{\texttt{'Assign(L:Loc,L1:Loc)}}{\texttt{reset(L1)}\curvearrowright\texttt{'Assign(L, L1)}}$
when currentOverflow(L1) [intermediate]

(G) $\dfrac{\texttt{'Assign(L,L1)}}{\texttt{'Assign(L, convertToLanguageValue(L1))}}$
when $\neg$ currentOverflow(L1) [intermediate]

Example: assignment

(A) `CONTEXT    'Assign(□,_)`

(B) `CONTEXT    'Assign(_:KResult,□)`

(C) $\texttt{'Assign}\left(\frac{\texttt{R:Ref}}{\texttt{convertToLoc(R)}}, \_\right)$ `[intermediate]`

(D) $\dfrac{\texttt{'Assign(L:Loc, V:Value)}}{\texttt{copyValueToLoc(V, L)} \curvearrowright \texttt{V}}$ `[step]`

(E) $\texttt{'Assign}\left(\_ : \texttt{KResult}, \frac{\texttt{V:ConvertibleToLoc}}{\texttt{convertToLoc(V,r)}}\right)$
`[intermediate]`

(F) $\dfrac{\texttt{'Assign(L:Loc,L1:Loc)}}{\texttt{reset(L1)} \curvearrowright \texttt{'Assign(L, L1)}}$
`when currentOverflow(L1)    [intermediate]`

(G) $\dfrac{\texttt{'Assign(L,L1)}}{\texttt{'Assign(L, convertToLanguageValue(L1))}}$
`when ¬ currentOverflow(L1)    [intermediate]`

Example: assignment

(A) CONTEXT $\quad$ `'Assign(□,_)`

(B) CONTEXT $\quad$ `'Assign(_:KResult,□)`

(C) $\texttt{'Assign}\left(\dfrac{\texttt{R:Ref}}{\texttt{convertToLoc(R)}}, \_\right)$ $\quad$ [intermediate]

(D) $\dfrac{\texttt{'Assign(L:Loc, V:Value)}}{\texttt{copyValueToLoc(V, L)} \curvearrowright \texttt{V}}$ $\quad$ [step]

(E) $\texttt{'Assign}\left(\_ : \texttt{KResult}, \dfrac{\texttt{V:ConvertibleToLoc}}{\texttt{convertToLoc(V,r)}}\right)$

[intermediate]

(F) $\dfrac{\texttt{'Assign(L:Loc,L1:Loc)}}{\texttt{reset(L1)} \curvearrowright \texttt{'Assign(L, L1)}}$

when currentOverflow(L1) $\quad$ [intermediate]

(G) $\dfrac{\texttt{'Assign(L,L1)}}{\texttt{'Assign(L, convertToLanguageValue(L1))}}$

when $\neg$ currentOverflow(L1) $\quad$ [intermediate]

Example: assignment

(A) `CONTEXT 'Assign(□,_)`

(B) `CONTEXT 'Assign(_:KResult,□)`

(C) $$\texttt{'Assign}\left(\frac{\texttt{R:Ref}}{\texttt{convertToLoc(R)}}, \_\right) \quad \texttt{[intermediate]}$$

(D) $$\frac{\texttt{'Assign(L:Loc, V:Value)}}{\texttt{copyValueToLoc(V, L)} \curvearrowright \texttt{V}} \quad \texttt{[step]}$$

(E) $$\texttt{'Assign}\left(\_ : \texttt{KResult}, \frac{\texttt{V:ConvertibleToLoc}}{\texttt{convertToLoc(V,r)}}\right) \quad \texttt{[intermediate]}$$

(F) $$\frac{\texttt{'Assign(L:Loc,L1:Loc)}}{\texttt{reset(L1)} \curvearrowright \texttt{'Assign(L, L1)}}$$
`when currentOverflow(L1)` `[intermediate]`

(G) $$\frac{\texttt{'Assign(L,L1)}}{\texttt{'Assign(L, convertToLanguageValue(L1))}}$$
`when` $\neg$ `currentOverflow(L1)` `[intermediate]`

Example - revisited

```
$a = array("one");
$c = $a[0] . ($a[0] = "two");
echo $c; // “onetwo"
```

```
$a = "one";
$c = $a . ($a = "two");
echo $c; // "twotwo"
```

Evaluation order: LR or RL?

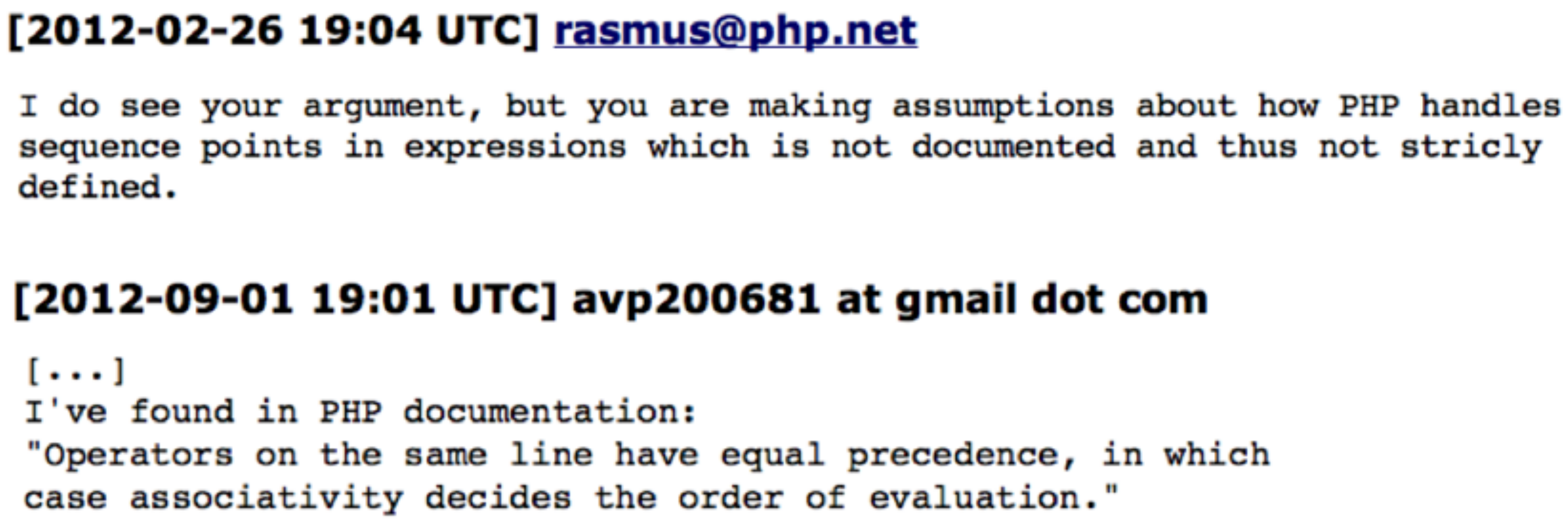
Example - revisited

```php
$a = array("one");
$c = $a[0] . ($a[0] = "two");
echo $c; // “onetwo"
```

```php
$a = "one";
$c = $a . ($a = "two");
echo $c; // "twotwo"
```

PHP bug 61188

Evaluation order: LR or RL?



[2012-02-26 19:04 UTC] rasmus@php.net

```
I do see your argument, but you are making assumptions about how PHP handles
sequence points in expressions which is not documented and thus not stricly
defined.
```

[2012-09-01 19:01 UTC] avp200681 at gmail dot com

```
[...]
I've found in PHP documentation:
"Operators on the same line have equal precedence, in which
case associativity decides the order of evaluation."
```

Example - explained

```
$a = array("one");
$c = $a[0] . ($a[0] = "two");
echo $c; // “onetwo"
```

```
$a = "one";
$c = $a . ($a = "two");
echo $c; // "twotwo"
```

- evaluation order **is left-to-right**
- array access evaluates to values
- variables evaluate to references
- references are resolved lazily

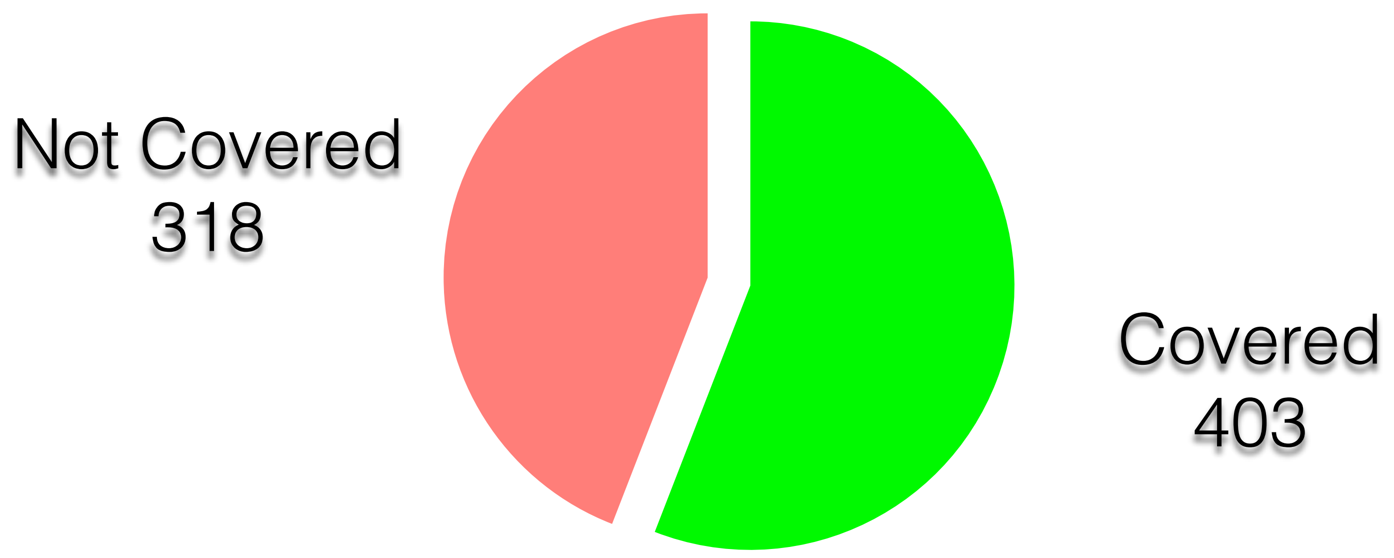
Validation

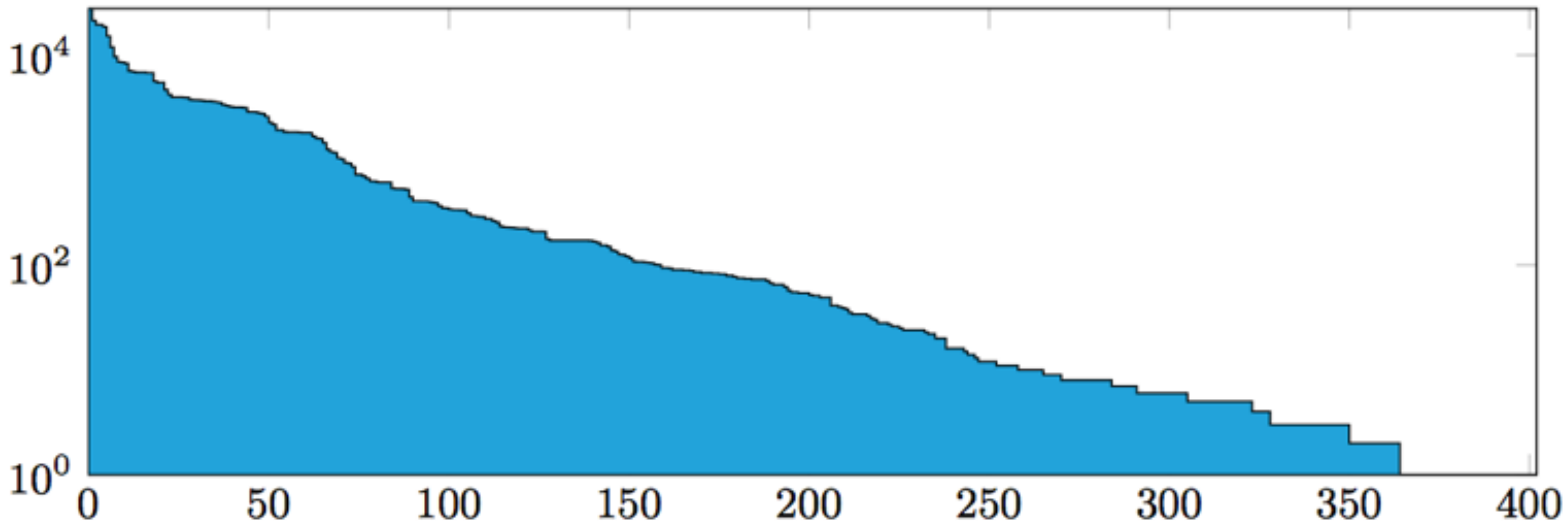
- Testing against the **Zend test suite.**
- Tests **categorised** (HTTP, date/time, crypto…)
- focusing on **core language** section of test suite
- passing all tests supported by our semantics

`zend/lang/002.phpt`

```
--TEST--
Simple While Loop Test
--FILE--
<?php
   $a=1;
   while ($a<10) {
      echo $a;
      $a++;
   }
?>
--EXPECT--
123456789
```

Coverage


Not Covered
318
Covered
403


10^4
10^2
10^0
0
50
100
150
200
250
300
350
400

Coverage

100%

(Zend + **Own test suite**)

Temporal verification of PHP programs

Extension of LTL with
predicates over KPHP configurations

LTL example

```
function foo() {
    global $y;
    $x = &$y;
}
```

LTL example

```
function foo() {
    global $y;
    $x = &$y;
}
$y = #symbolic_input();
```

LTL example

```
function foo() {
    global $y;
    $x = &$y;
}
$y = #symbolic_input();
foo();
```

LTL example

```
function foo() {
    global $y;
    $x = &$y;
}
$y = #symbolic_input();
foo();
```

```
◊alias(fv('foo',var('x')),gv(var('y')))
```

Add language features

Fix bugs

Future Work

Add language features

Fix bugs

Future Work

Deductive verification (**Reachability Logic**)

Add language features

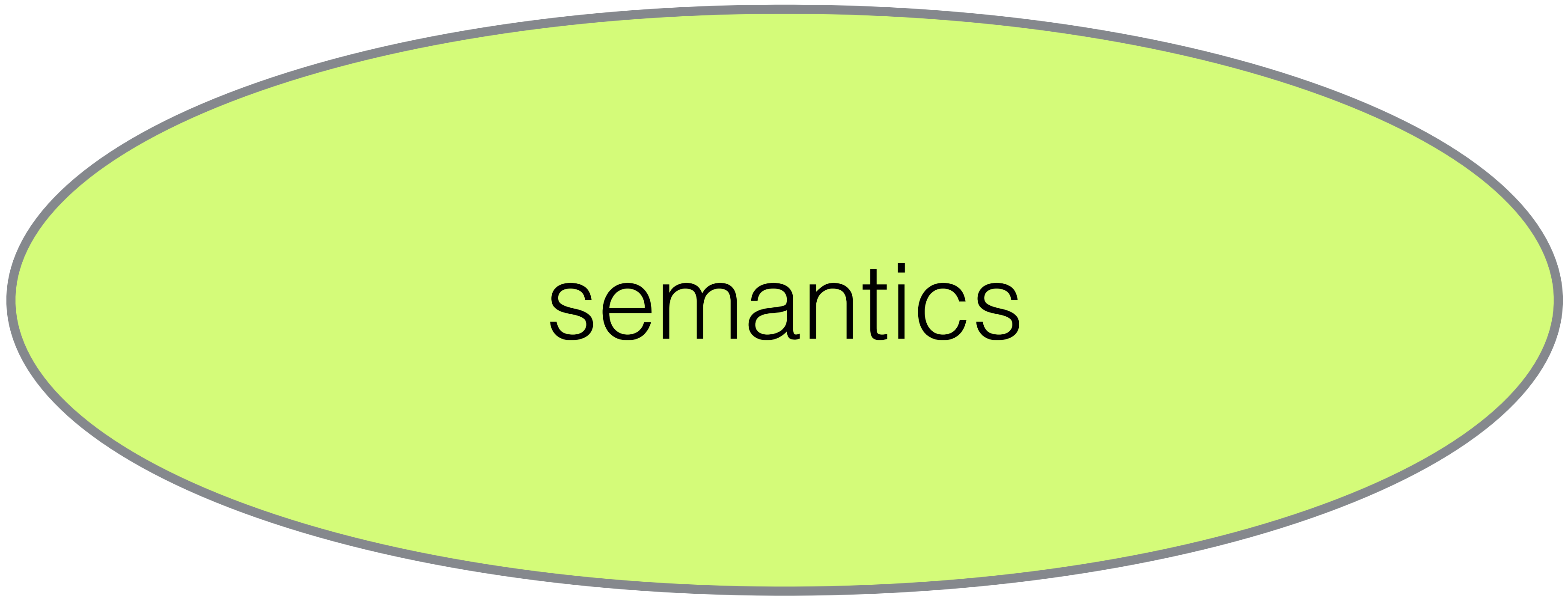
Fix bugs

Future Work

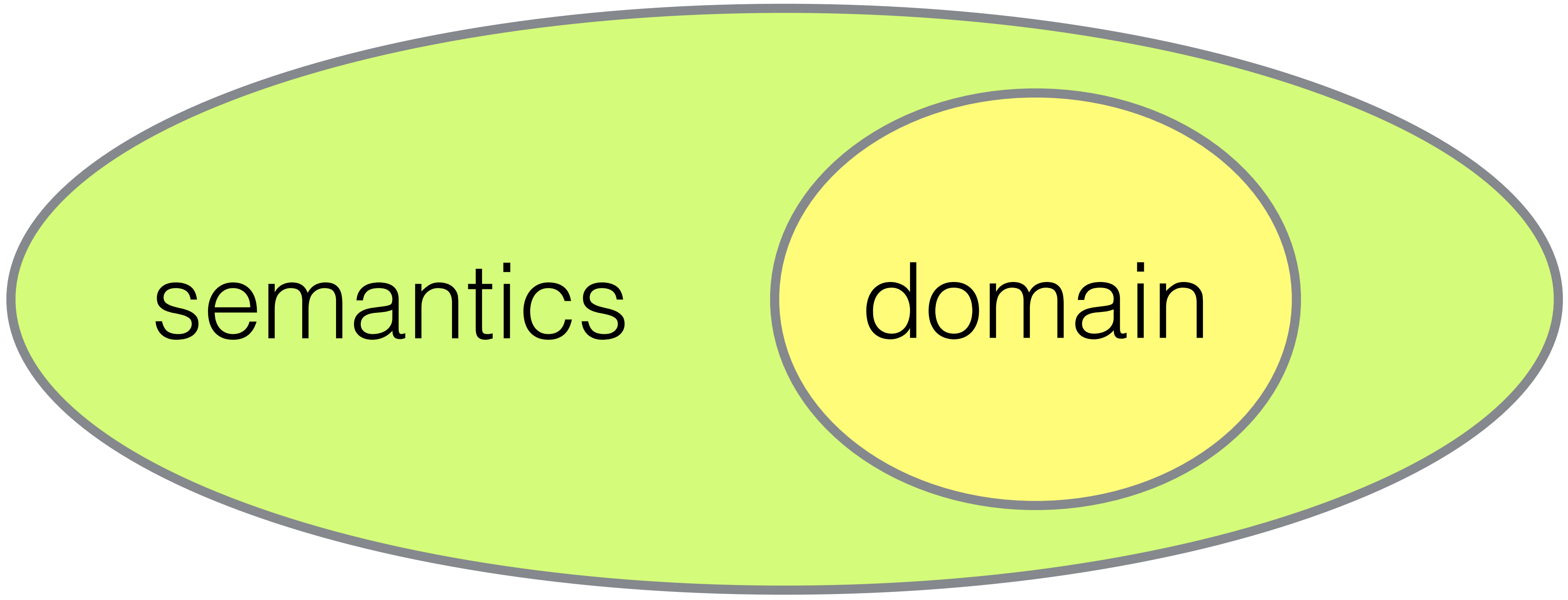
Deductive verification (**Reachability Logic**)

Static Analysis (**Abstract Interpretation**)

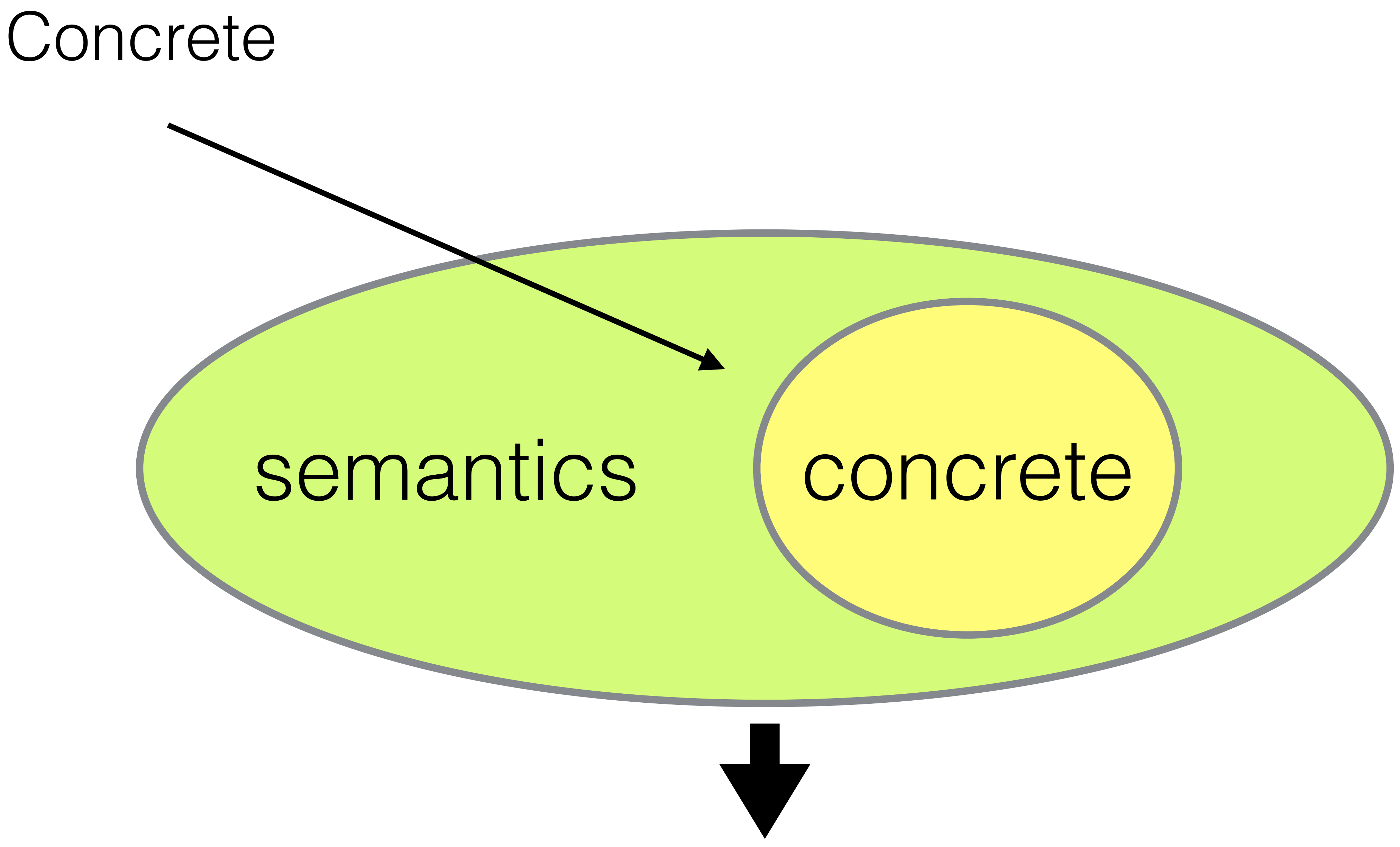
Abstract Interpretation



Abstract Interpretation

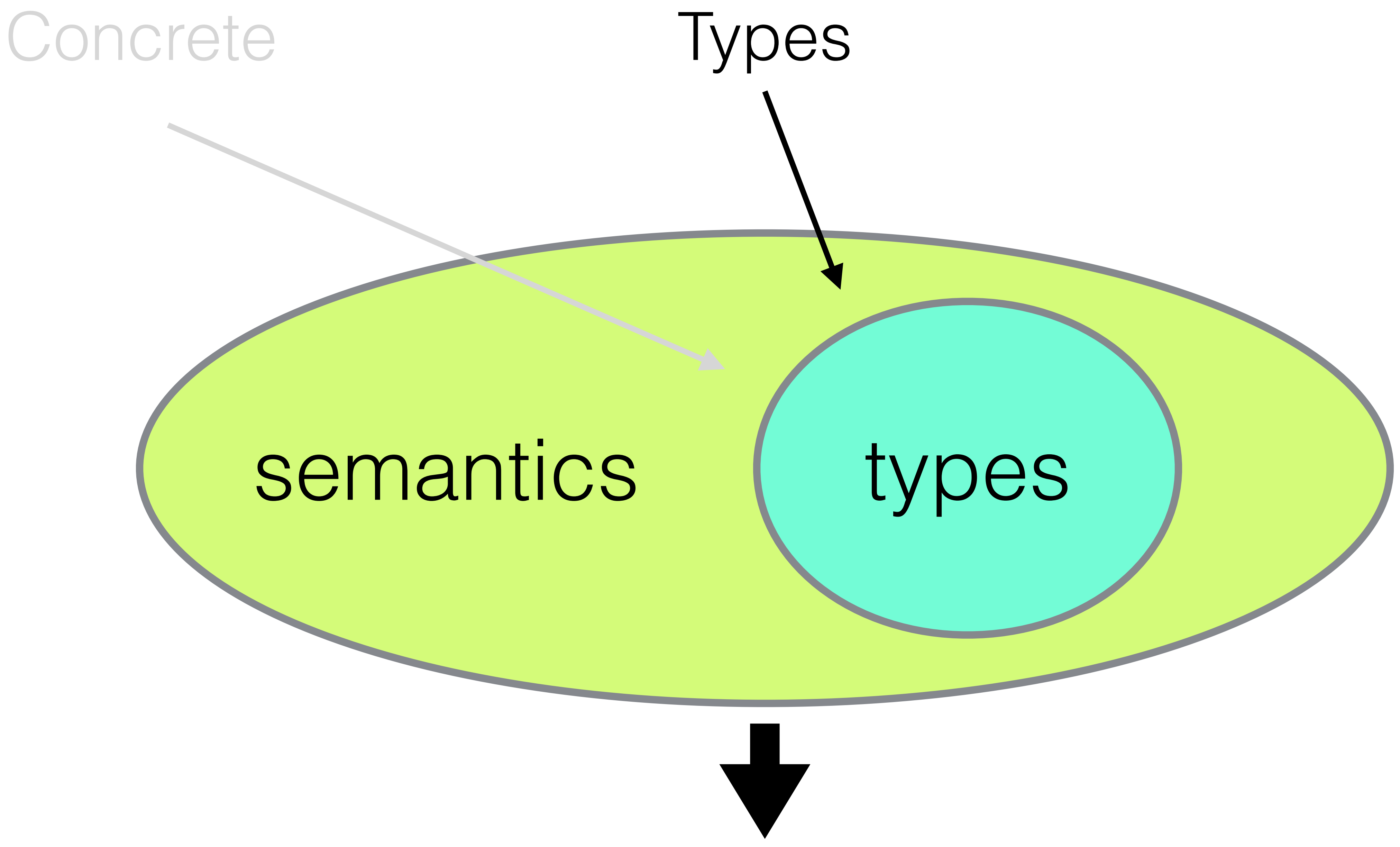


Abstract Interpretation



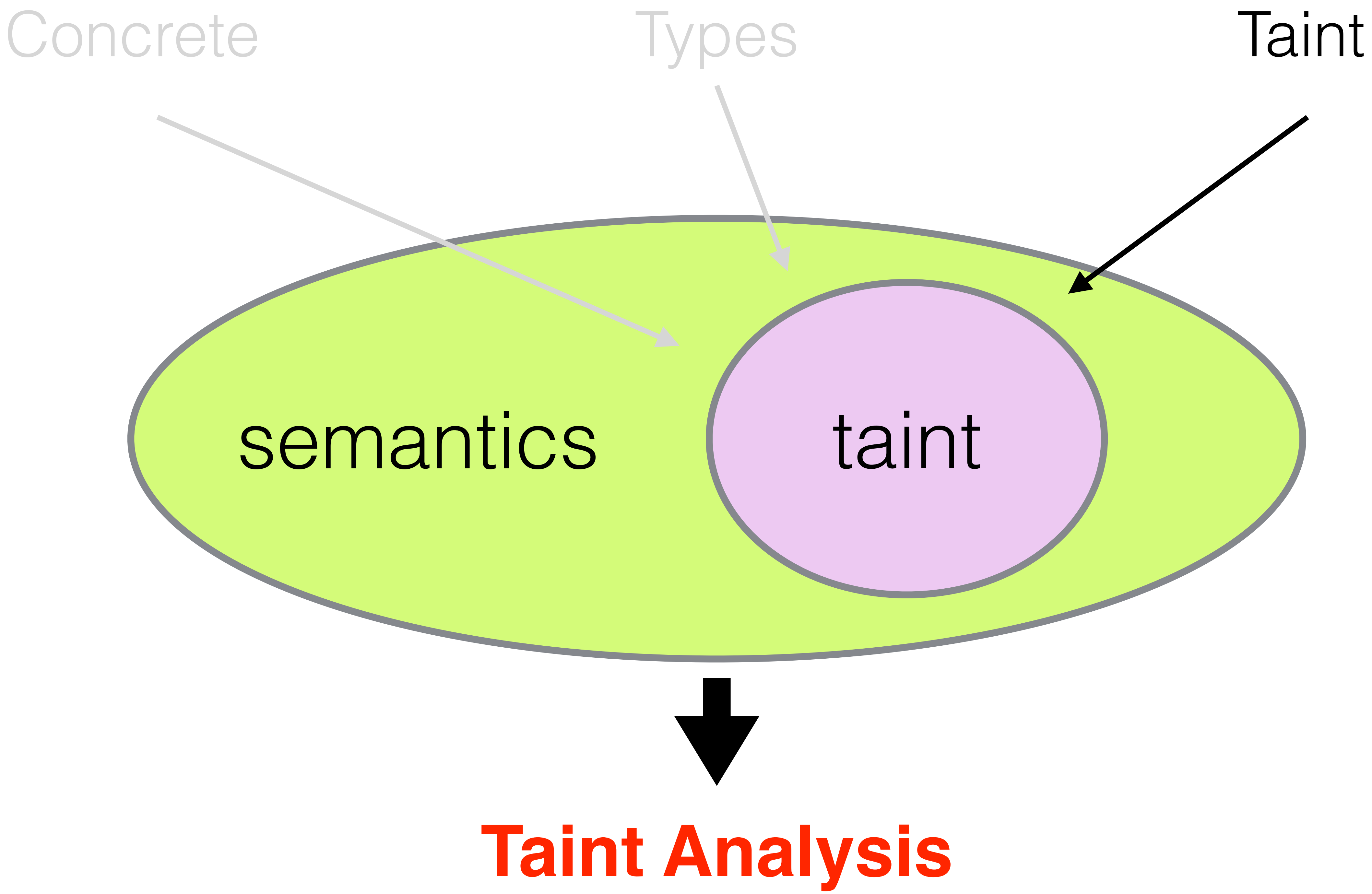
PHP interpreter

Abstract Interpretation

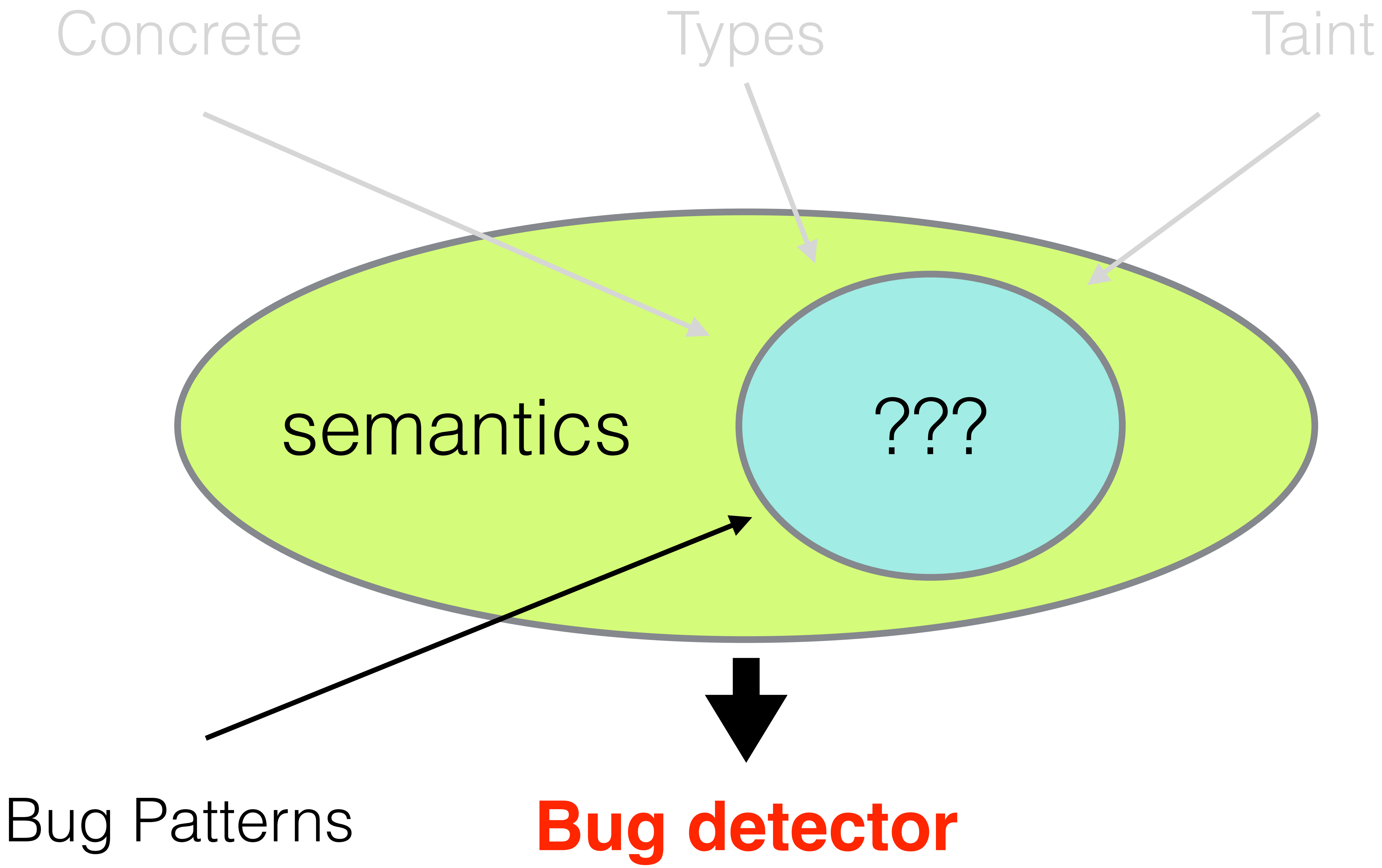


Type Analysis

Abstract Interpretation



Abstract Interpretation

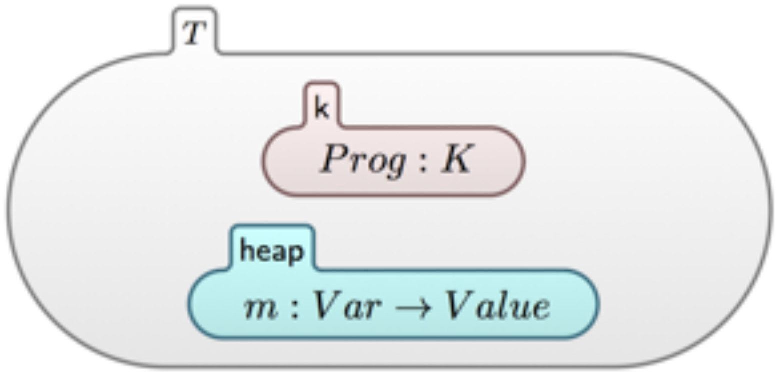


Thank you!

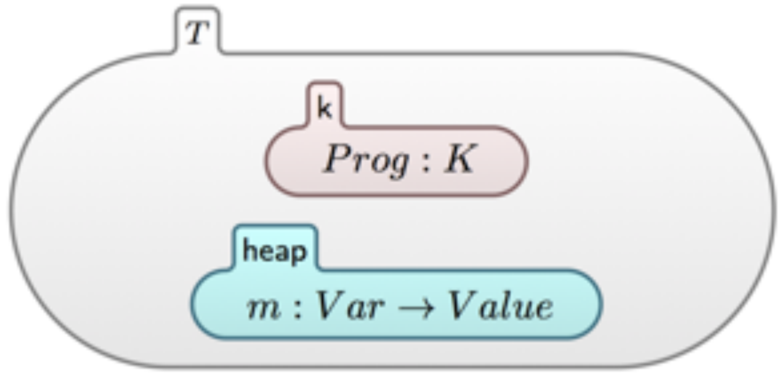
www.phpsemantics.org

paper, sources, web interface

Appendix/misc/old

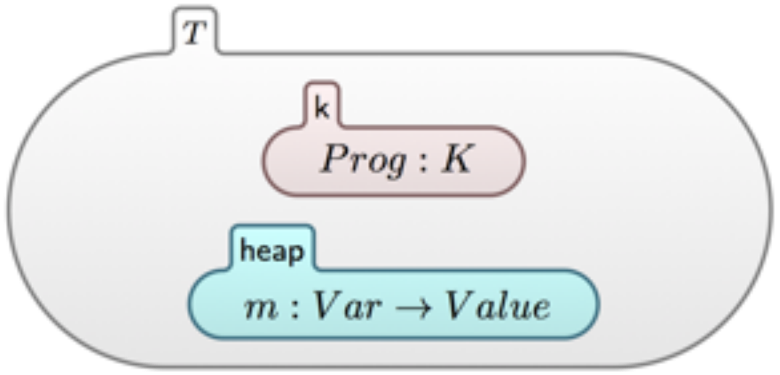


Komputations



Komputations

(i) the K cell holds **a list of computations separated by** $\curvearrowright$



Komputations

(i) the K cell holds **a list of computations separated by** $\curvearrowright$

(ii) the input program goes into the k cell

Example: function call

$$\left\langle \frac{\texttt{runFunction(FN:String, f(FP:K, FB:K, RT:RetType, LS:Loc), Args:K)}\curvearrowright\texttt{K}}{\left(\begin{array}{r}\texttt{processFunArgs(FP, Args)} \curvearrowright \\ \texttt{pushStackFrame(FN, K, L, CurrentClass, CurrentObj, RT, D)} \curvearrowright \\ \texttt{ArrayCreateEmpty(L1)} \curvearrowright \texttt{setCrntScope(L1)} \curvearrowright \texttt{incRefCount(L1)} \curvearrowright \\ \texttt{copyFunArgs} \curvearrowright \texttt{FB} \curvearrowright \texttt{'Return(NULL)}\end{array}\right)} \right\rangle_{\texttt{k}}$$

$$\langle \texttt{L:Loc} \rangle_{\texttt{currentScope}} \quad \langle \texttt{CurrentClass:Id} \rangle_{\texttt{class}} \quad \langle \texttt{CurrentObj:Loc} \rangle_{\texttt{object}}$$

$$\left\langle \frac{\texttt{D:K}}{\cdot} \right\rangle_{\texttt{functionArgumentDeclaration}}$$

`when fresh(L1)     [internal]`

```
$a = array("a", "b", "c");
```

```php
$a = array("a", "b", "c");
foreach($a as &$v) {};
```

```
$a = array("a", "b", "c");
foreach($a as &$v) {};
foreach($a as $v) {};
```

```php
$a = array("a", "b", "c");
foreach($a as &$v) {};
foreach($a as $v) {};
var_dump($a);
```

```
$a = array("a", "b", "c");
foreach($a as &$v) {};
foreach($a as $v) {};
var_dump($a);
```

```
  array(3) {
    [0]=> string(1) "a"
    [1]=> string(1) "b"
    [2]=> &string(1) "b"
  }
```