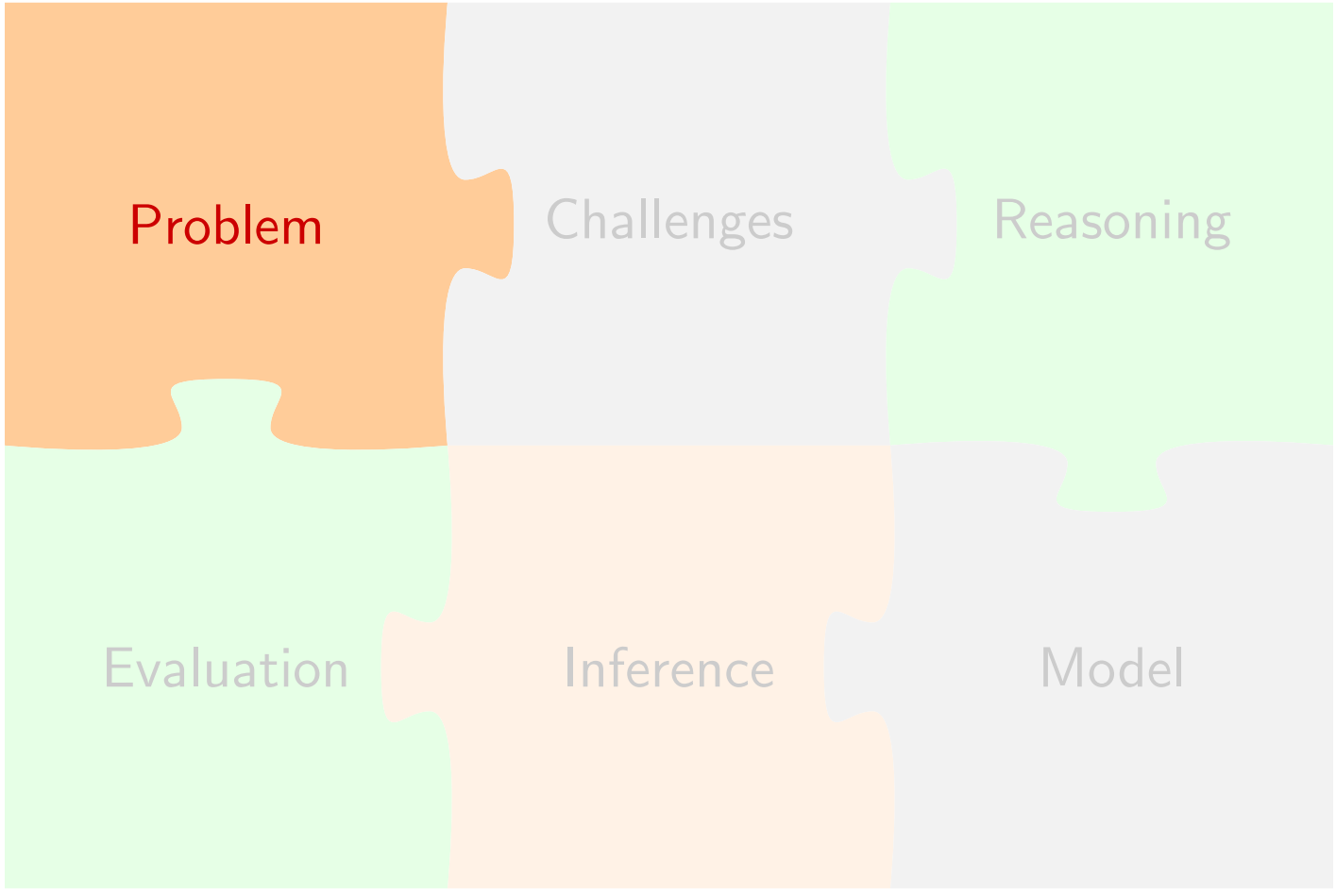
Safe Initialization of Objects

Fengyun Liu

LAMP, EPFL

July 6, 2020


Problem
Challenges
Reasoning
Evaluation
Inference
Model

A 50-year old problem

```
1  class Hello {
2    val message = "Hello, " + name
3    val name    = "world"
4  }
5
6  println(new Hello().message))
```

A 50-year old problem

```
1  class Hello {
2    val message = "Hello, " + name
3    val name    = "world"
4  }
5
6  println(new Hello().message))
```

At runtime: Hello, null

A 50-year old problem

```
1  class Hello {
2    val message = "Hello, " + name
3    val name    = "world"
4  }
5
6  println(new Hello().message))
```

At runtime: Hello, null

Status quo

All mainstream object-oriented languages still suffer from the problem, including

C++, Java, Scala, C#, Kotlin, ...

Why the problem is important?

Not only are partially-constructed objects a source of consternation for everyday programmers, they are also a challenge for language designers wanting to provide guarantees around invariants, immutability and concurrency-safety, and non-nullability.

— Joe Duffy, "On partially-constructed objects"

http://joeduffyblog.com/2010/06/27/on-partiallyconstructed-objects/

Is it only a problem of OOP?

```
1  let rec even n = n = 0 || odd (x - 1)
2      and odd n = n = 1 || even (x - 1)
3      and flag = odd 3
```

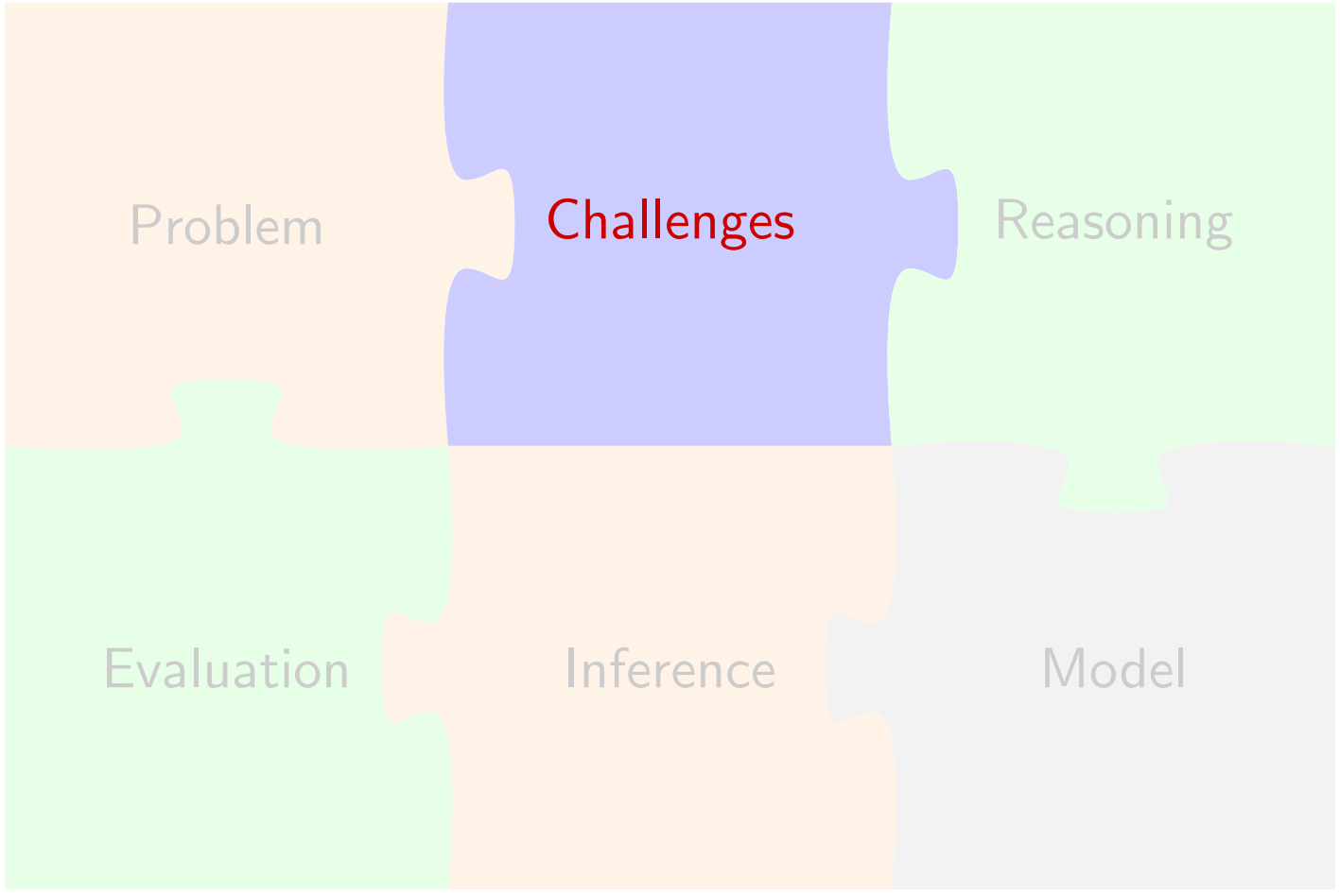
The latest OCaml compiler rejects the code!

Is it only a problem of strict languages?

The following Haskell code loops forever:

```
1  a = if b then 10 else 20
2  b = a >= 10
3
4  main = putStrLn (show a)
```

Initialization errors are disguised as non-termination!


Problem
Challenges
Reasoning
Evaluation
Inference
Model

Can we disallow usage of this before initialization?

Can we disallow usage of this before initialization?

Requirement 1. Call methods and access fields on `this`.

Gil et al. (2009) report that over 8% constructors include method calls on `this`.

Can we disallow usage of this before initialization?

Requirement 1. Call methods and access fields on `this`.

Gil et al. (2009) report that over 8% constructors include method calls on `this`.

Requirement 2. Creation of cyclic data structures.

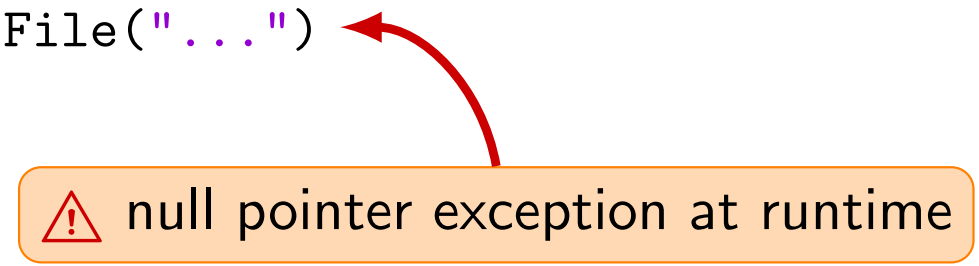
```
1 class Parent { val child: Child = new Child(this) }
2 class Child(parent: Parent)
```

Challenge 1: virtual method calls

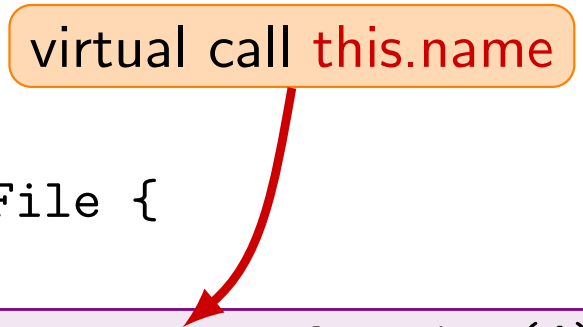
```
1  abstract class AbstractFile {
2    def name: String
3    val extension:  String = name.substring(4)
4  }
5
6  class RemoteFile(url:String) extends AbstractFile {
7    val localFile: String = url.hashCode
8    def name:  String = localFile
9  }
10
11 new RemoteFile("...")
```

Challenge 1: virtual method calls

```
1  abstract class AbstractFile {
2    def name: String
3    val extension:  String = name.substring(4)
4  }
5
6  class RemoteFile(url:String) extends AbstractFile {
7    val localFile: String = url.hashCode
8    def name:  String = localFile
9  }
10
11 new RemoteFile("...")
```



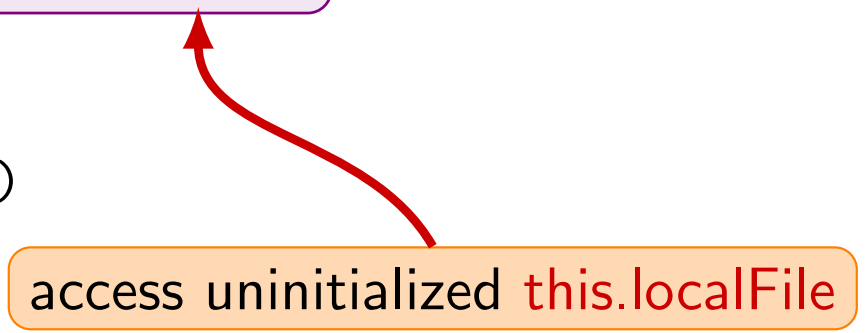
Challenge 1: virtual method calls



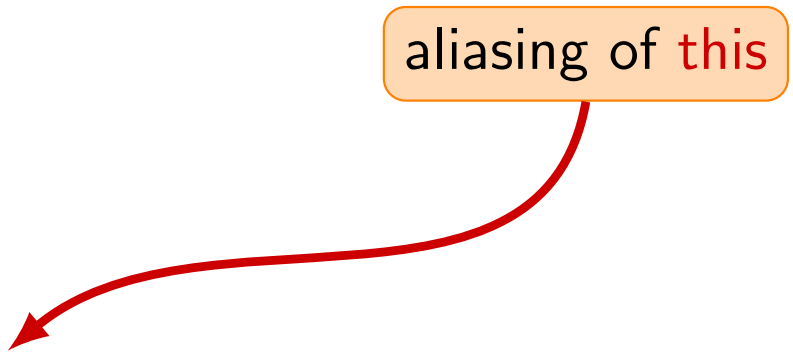

```
1  abstract class AbstractFile {
2    def name: String
3    val extension:  String = name.substring(4)
4  }
5
6  class RemoteFile(url:String) extends AbstractFile {
7    val localFile: String = url.hashCode
8    def name:  String = localFile
9  }
10
11 new RemoteFile("...")
```

Challenge 1: virtual method calls

```
1  abstract class AbstractFile {
2    def name: String
3    val extension:  String = name.substring(4)
4  }
5
6  class RemoteFile(url:String) extends AbstractFile {
7    val localFile: String = url.hashCode
8    def name:  String = localFile
9  }
10
11 new RemoteFile("...")
```

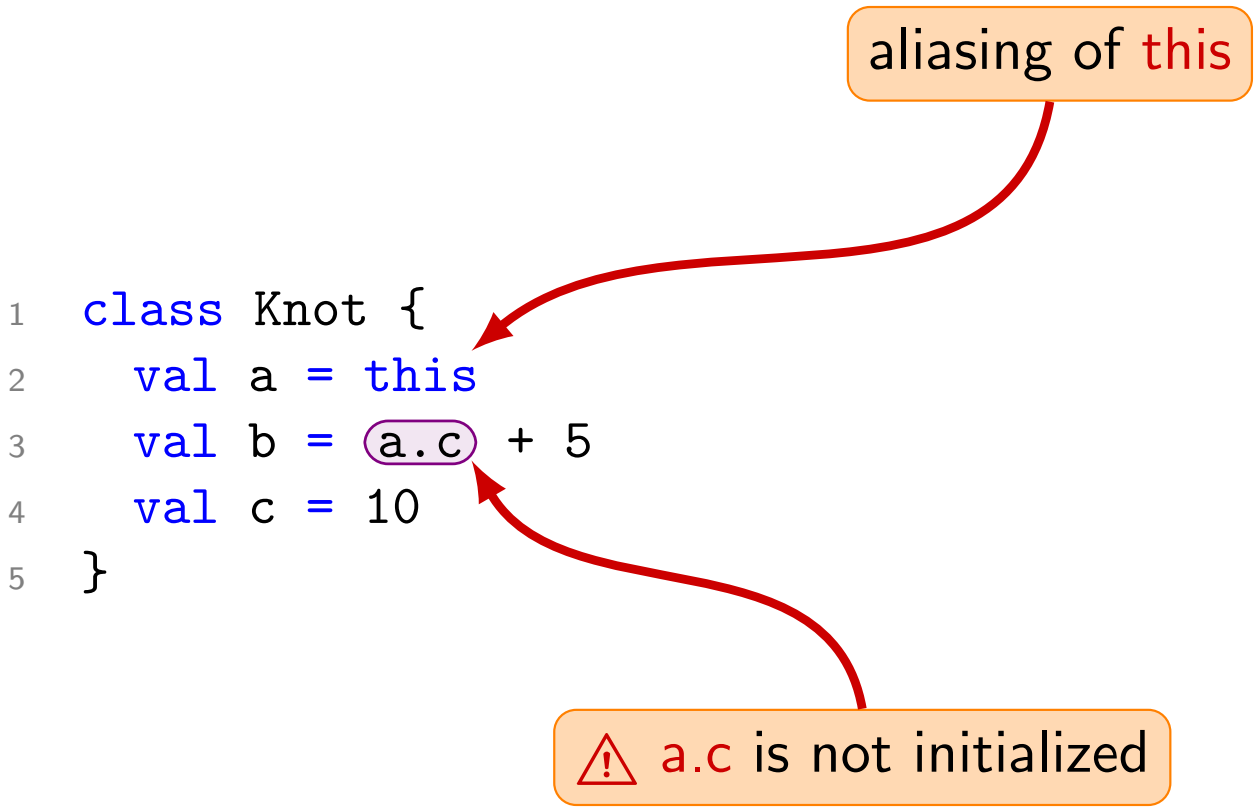


Challenge 2: aliasing



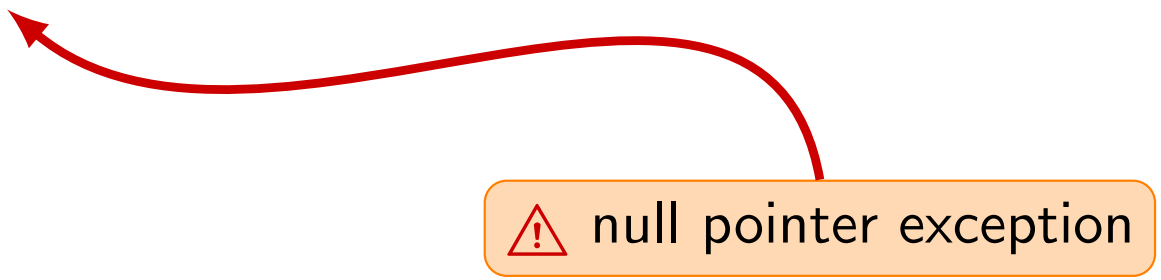

```
1  class Knot {
2    val a = this
3    val b = a.c + 5
4    val c = 10
5  }
```

Challenge 2: aliasing

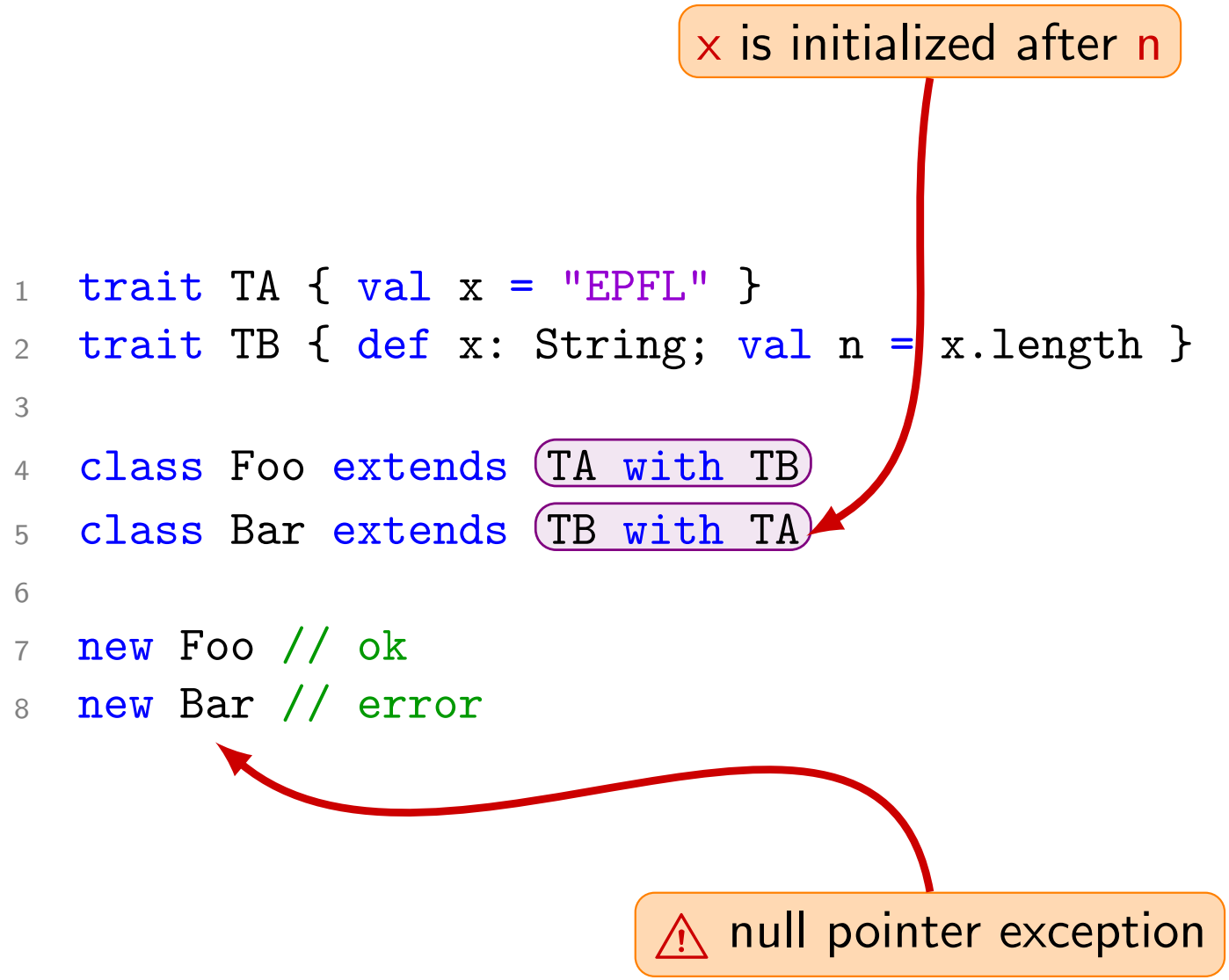


Challenge 3: traits

```
1  trait TA { val x = "EPFL" }
2  trait TB { def x: String; val n = x.length }
3
4  class Foo extends TA with TB
5  class Bar extends TB with TA
6
7  new Foo // ok
8  new Bar // error
```



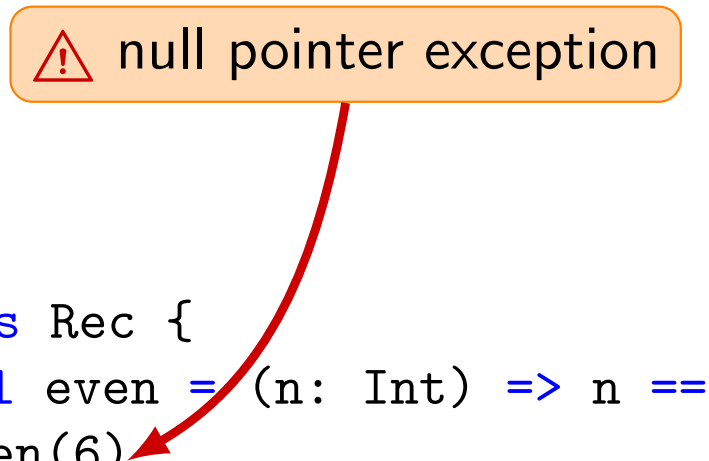
Challenge 3: traits



Challenge 4: first-class functions

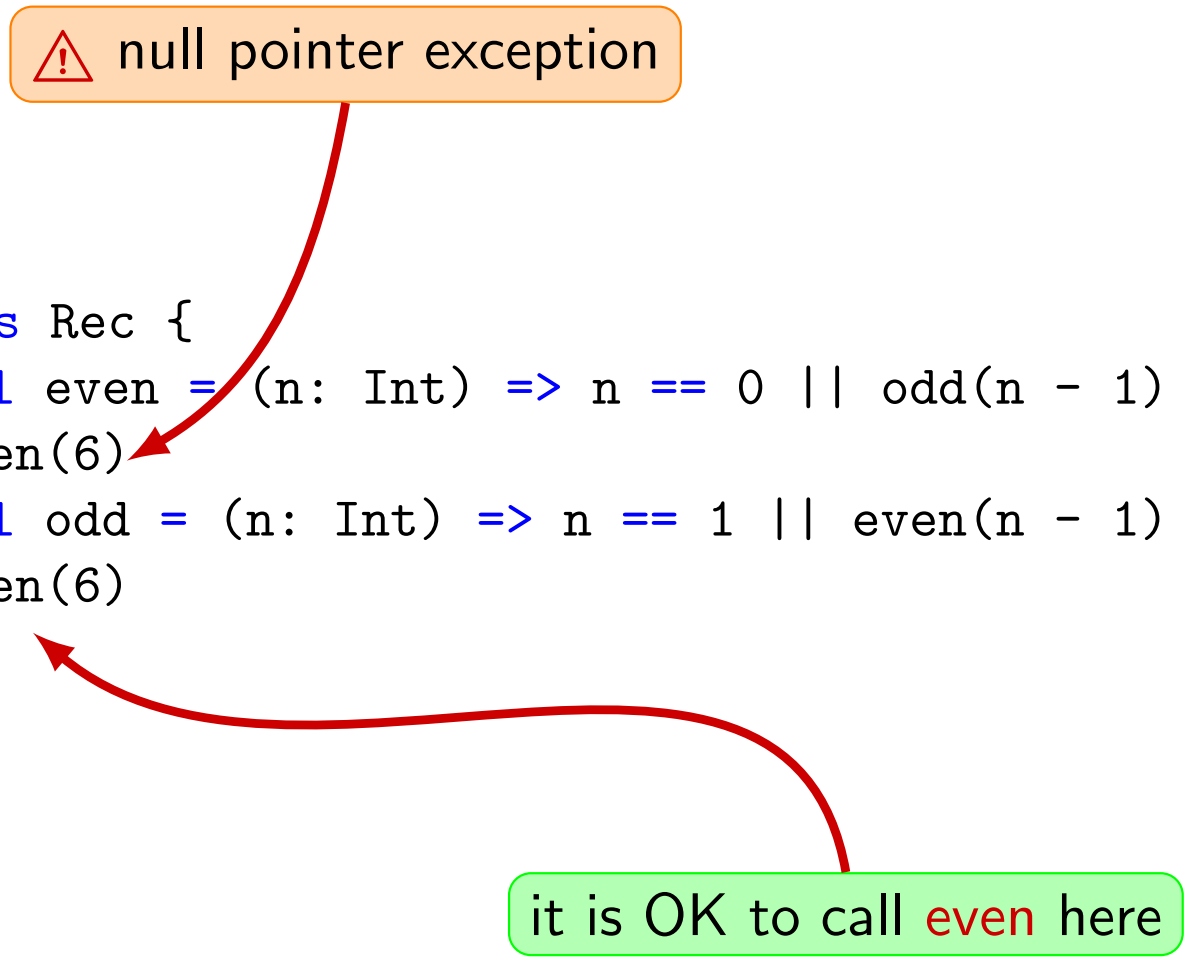
```
1  class Rec {
2    val even = (n: Int) => n == 0 || odd(n - 1)
3    even(6)
4    val odd = (n: Int) => n == 1 || even(n - 1)
5    even(6)
6  }
```

Challenge 4: first-class functions




```
1  class Rec {
2    val even = (n: Int) => n == 0 || odd(n - 1)
3    even(6)
4    val odd = (n: Int) => n == 1 || even(n - 1)
5    even(6)
6  }
```

Challenge 4: first-class functions



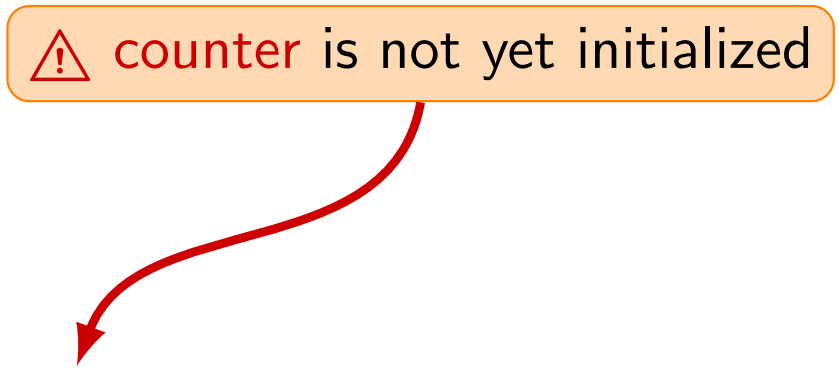
Challenge 5: inner classes

```
1  class Trees {
2    class ValDef { counter += 1 }
3    class EmptyValDef extends ValDef
4
5    val theEmptyValDef = new EmptyValDef
6    private var counter = 0
7  }
```

Challenge 5: inner classes

```
1  class Trees {
2    class ValDef { counter += 1 }
3    class EmptyValDef extends ValDef
4
5    val theEmptyValDef = new EmptyValDef
6    private var counter = 0
7  }
```

Challenge 5: inner classes

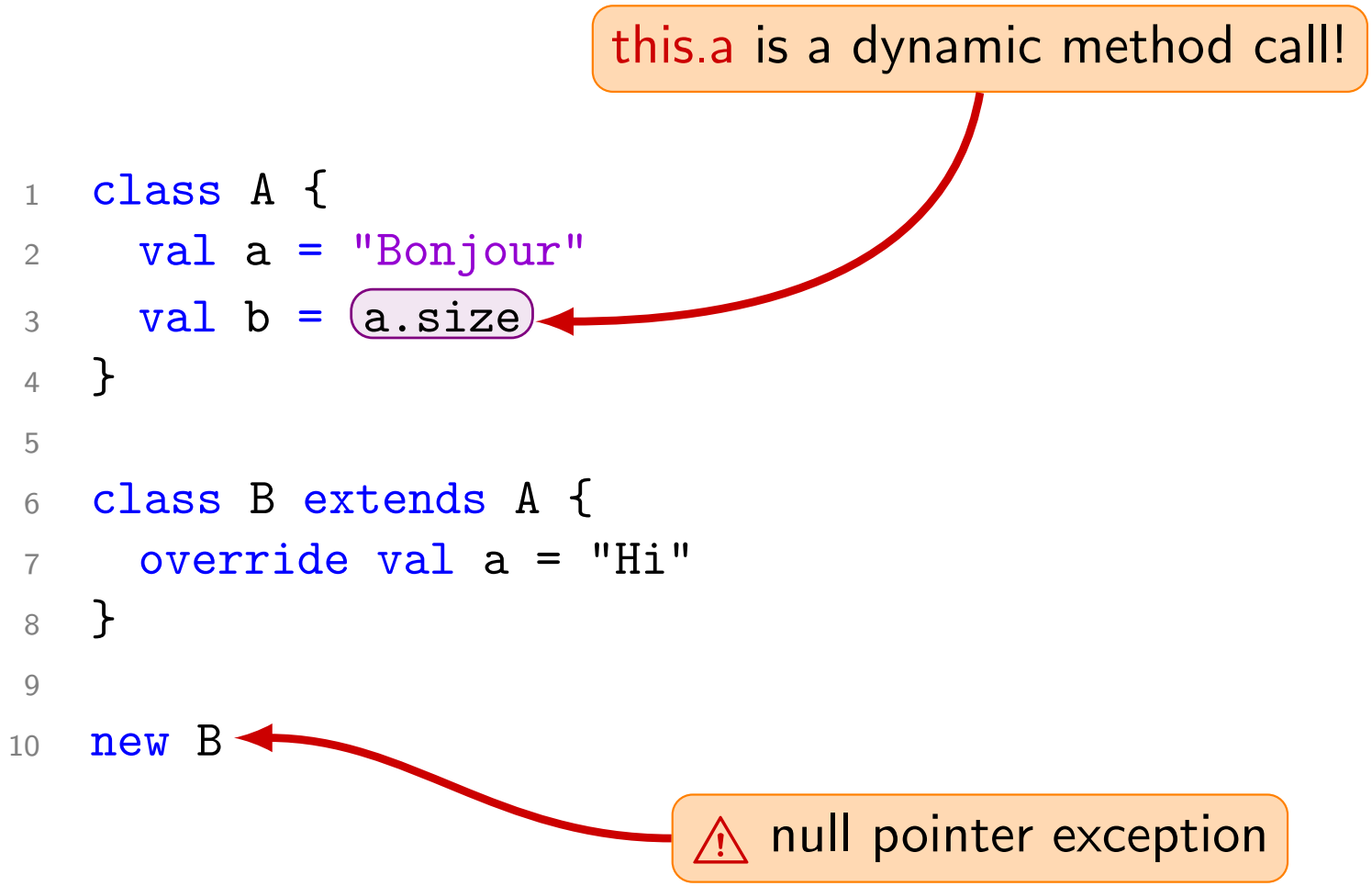



```
1  class Trees {
2    class ValDef { counter += 1 }
3    class EmptyValDef extends ValDef
4
5    val theEmptyValDef = new EmptyValDef
6    private var counter = 0
7  }
```

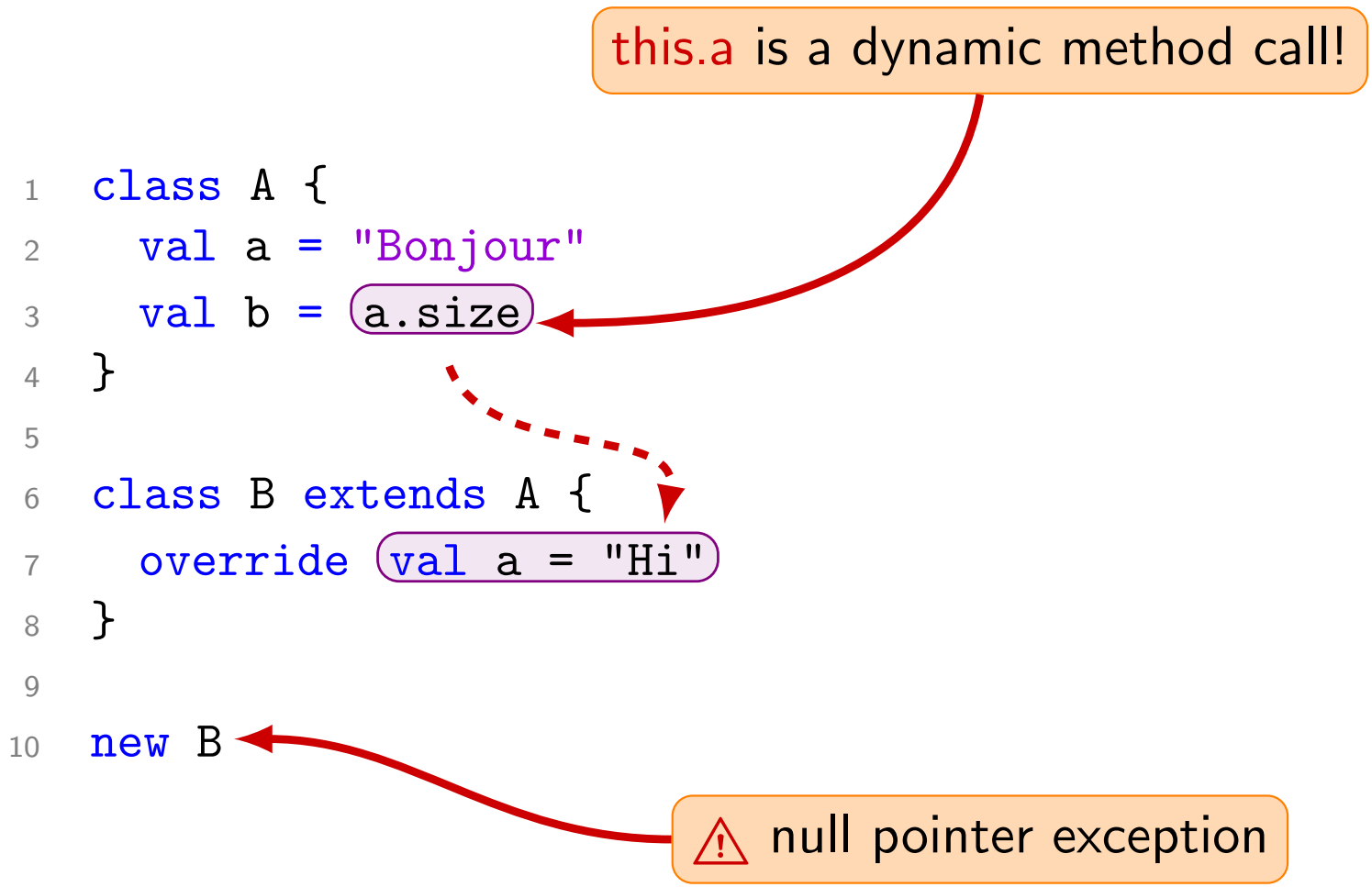
Challenge 6: properties

```
1  class A {
2    val a = "Bonjour"
3    val b = a.size
4  }
5
6  class B extends A {
7    override val a = "Hi"
8  }
9
10 new B
```

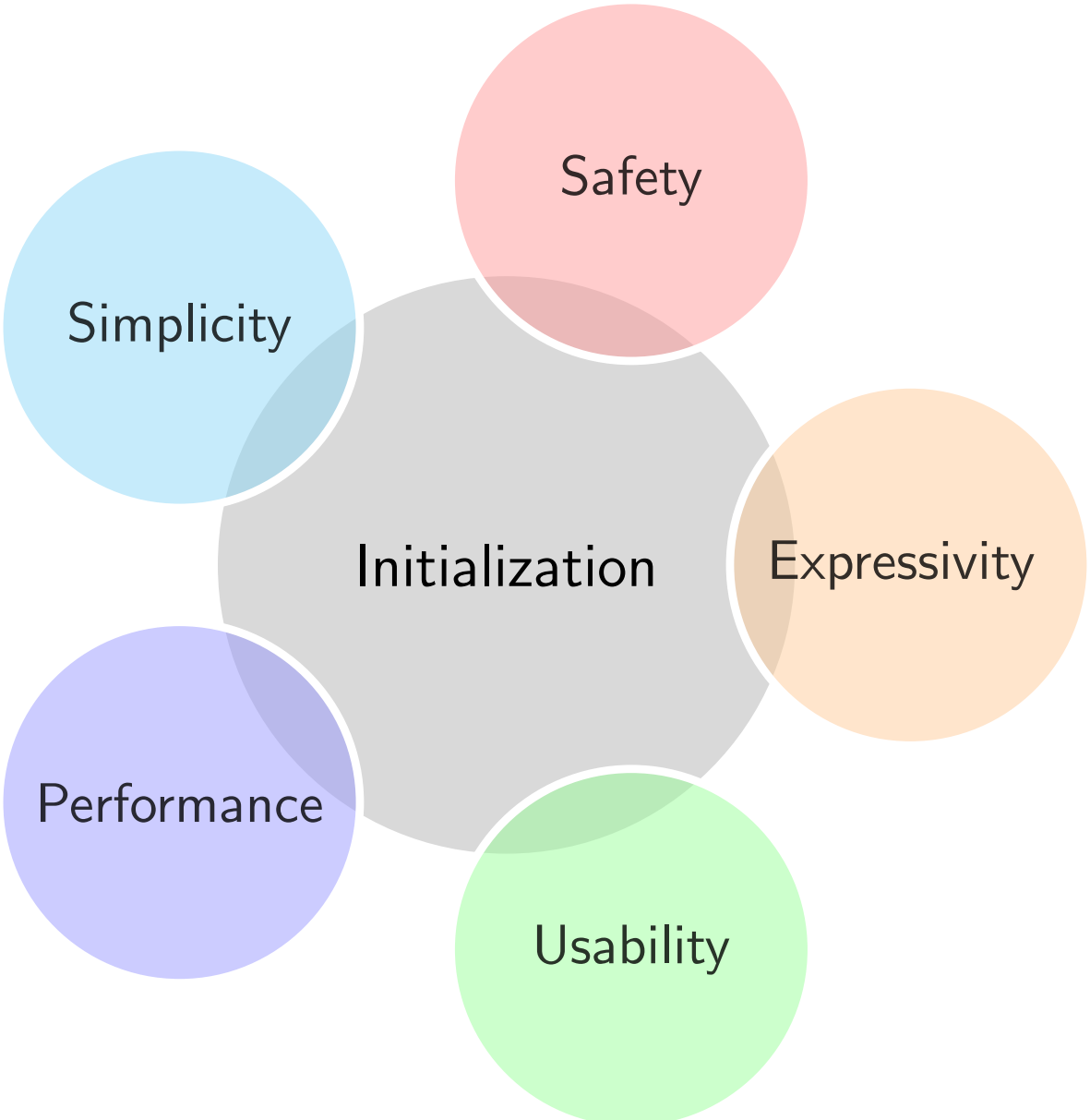
Challenge 6: properties

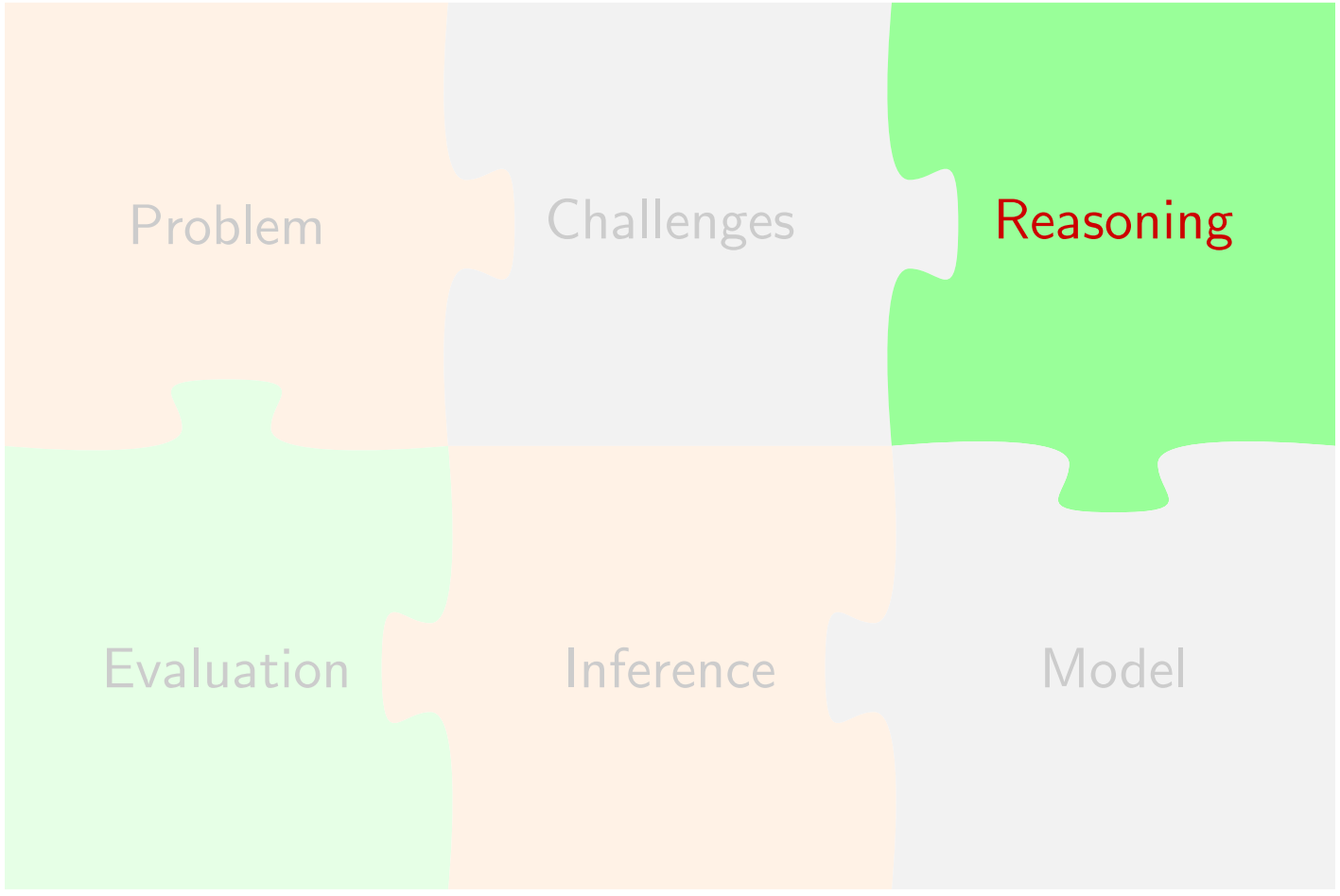


Challenge 6: properties



Engineering Challenges




Problem
Challenges
Reasoning
Evaluation
Inference
Model

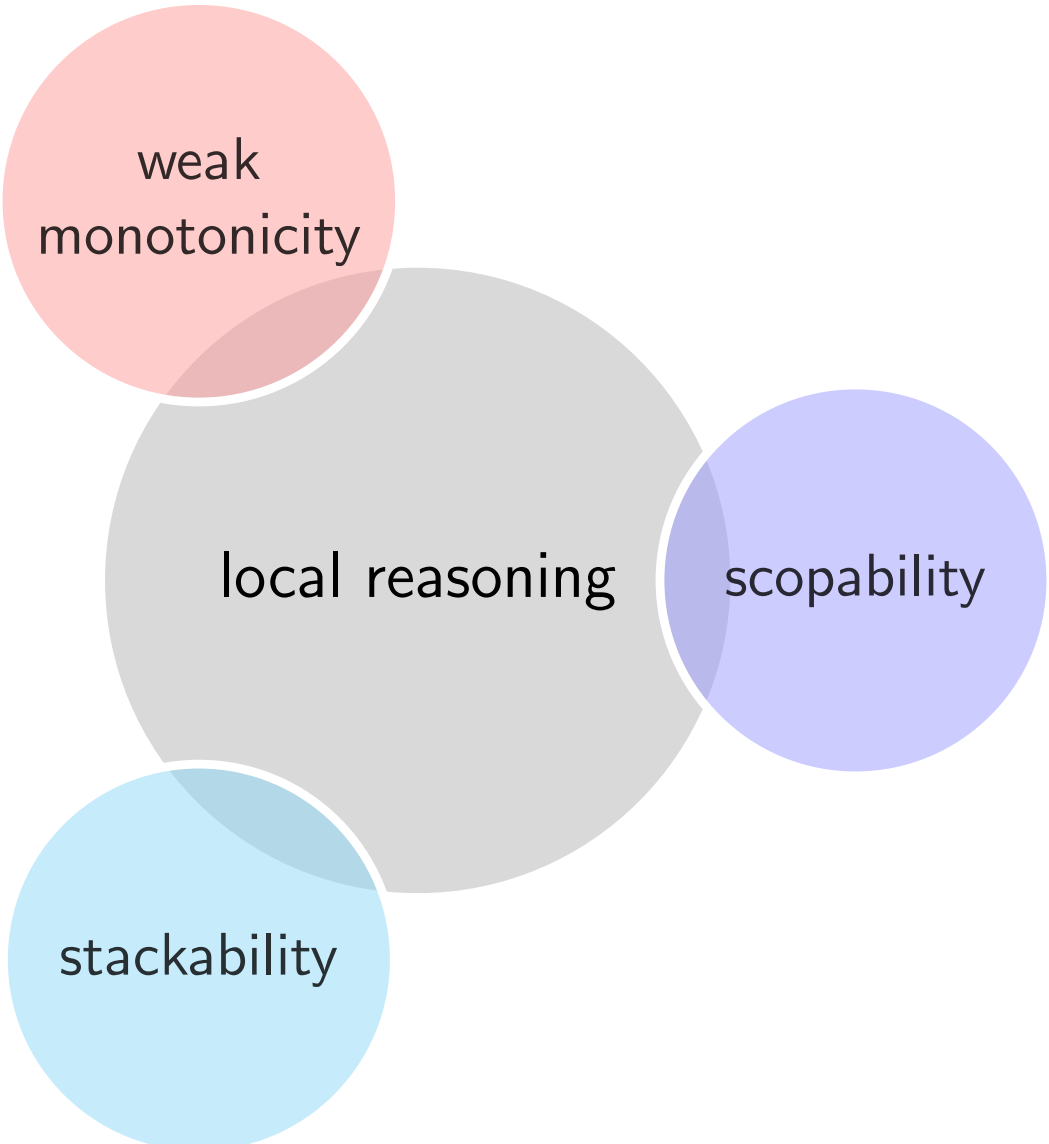
A golden idea: local reasoning

If a constructor is called with only transitively initialized arguments, so is the resulting object.

If the receiver and arguments of a method call are transitively initialized, so is the result.

Summers, Alexander J. and Peter Müller. "Freedom before commitment." OOPSLA '11

Three pillars of local reasoning



A small language

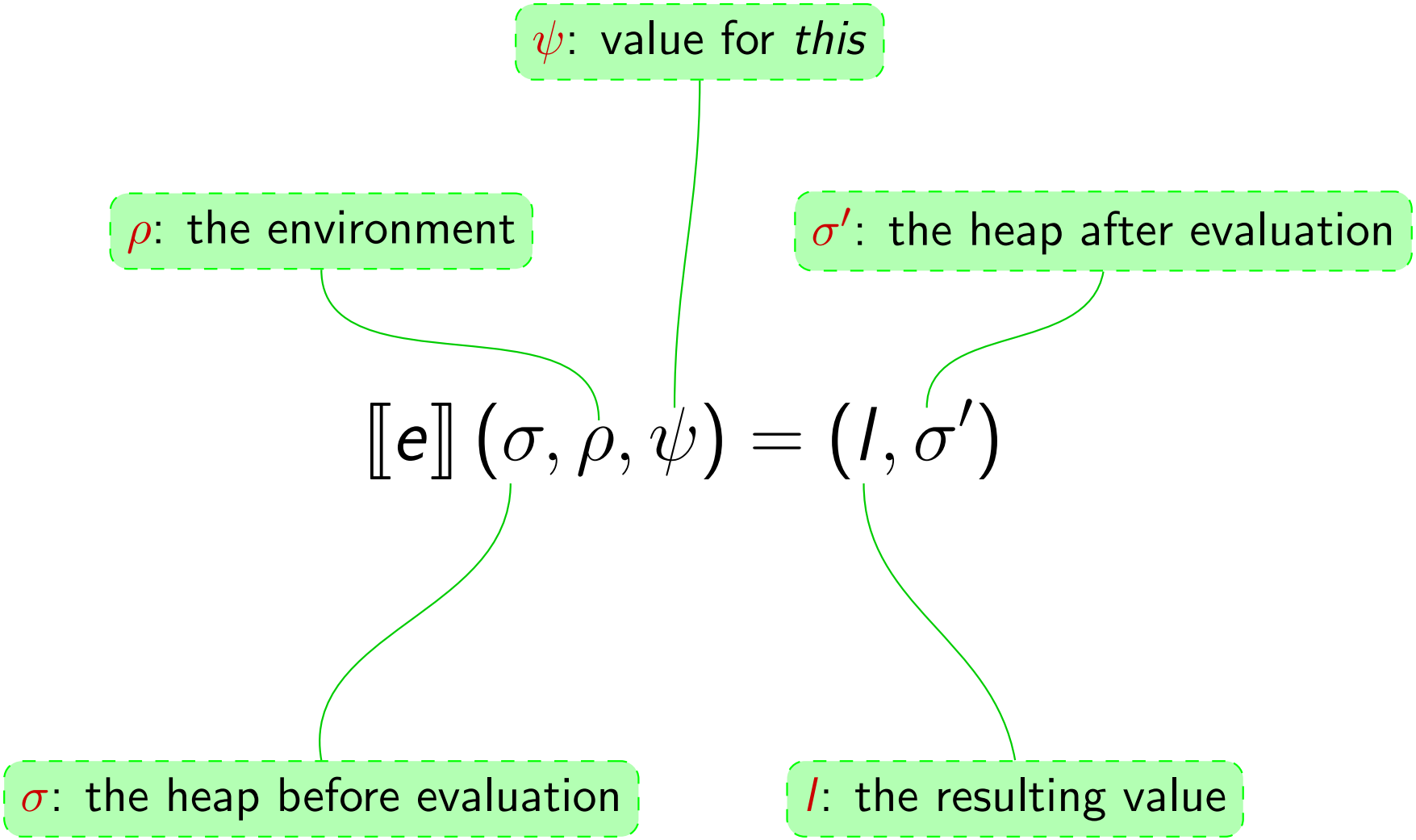
$$
\begin{array}{rcl}
\mathcal{P} \in \mathit{Program} & ::= & (\overline{\mathcal{C}}, D) \\
\mathcal{C} \in \mathit{Class} & ::= & \mathit{class}\ C(\overline{\hat{f}{:}T})\ \{\ \overline{\mathcal{F}}\ \overline{\mathcal{M}}\ \} \\
\mathcal{F} \in \mathit{Field} & ::= & \mathit{var}\ f{:}T = e \\
e \in \mathit{Exp} & ::= & x \mid \mathit{this} \mid e.f \mid e.m(\overline{e}) \mid \\
 & & \mathit{new}\ C(\overline{e}) \mid e.f = e; e \\
\mathcal{M} \in \mathit{Method} & ::= & \mathit{def}\ m(\overline{x{:}T}) : T = e \\
S, T, U \in \mathit{Type} & ::= & C
\end{array}
$$

```
1  class Parent { var child: Child = new Child(this) }
2  class Child(parent: Parent)
```

Semantics

$$[\![e]\!]\,(\sigma, \rho, \psi) = (l, \sigma')$$

Semantics


ψ: value for *this*
ρ: the environment
σ': the heap after evaluation
$[\![e]\!](\sigma, \rho, \psi) = (l, \sigma')$
σ: the heap before evaluation
l: the resulting value

Semantics - Examples

$$
\begin{array}{lcl}
[\![x]\!]\,(\sigma,\rho,\psi) & = & (\rho(x),\sigma) \\
[\![\mathit{this}]\!]\,(\sigma,\rho,\psi) & = & (\psi,\sigma) \\
[\![e.f]\!]\,(\sigma,\rho,\psi) & = & (\omega(f),\sigma_1) \text{ where} \\
 & & (l_0,\sigma_1) = [\![e]\!]\,(\sigma,\rho,\psi) \\
 & & \text{and } (\_,\omega) = \sigma_1(l_0)
\end{array}
$$

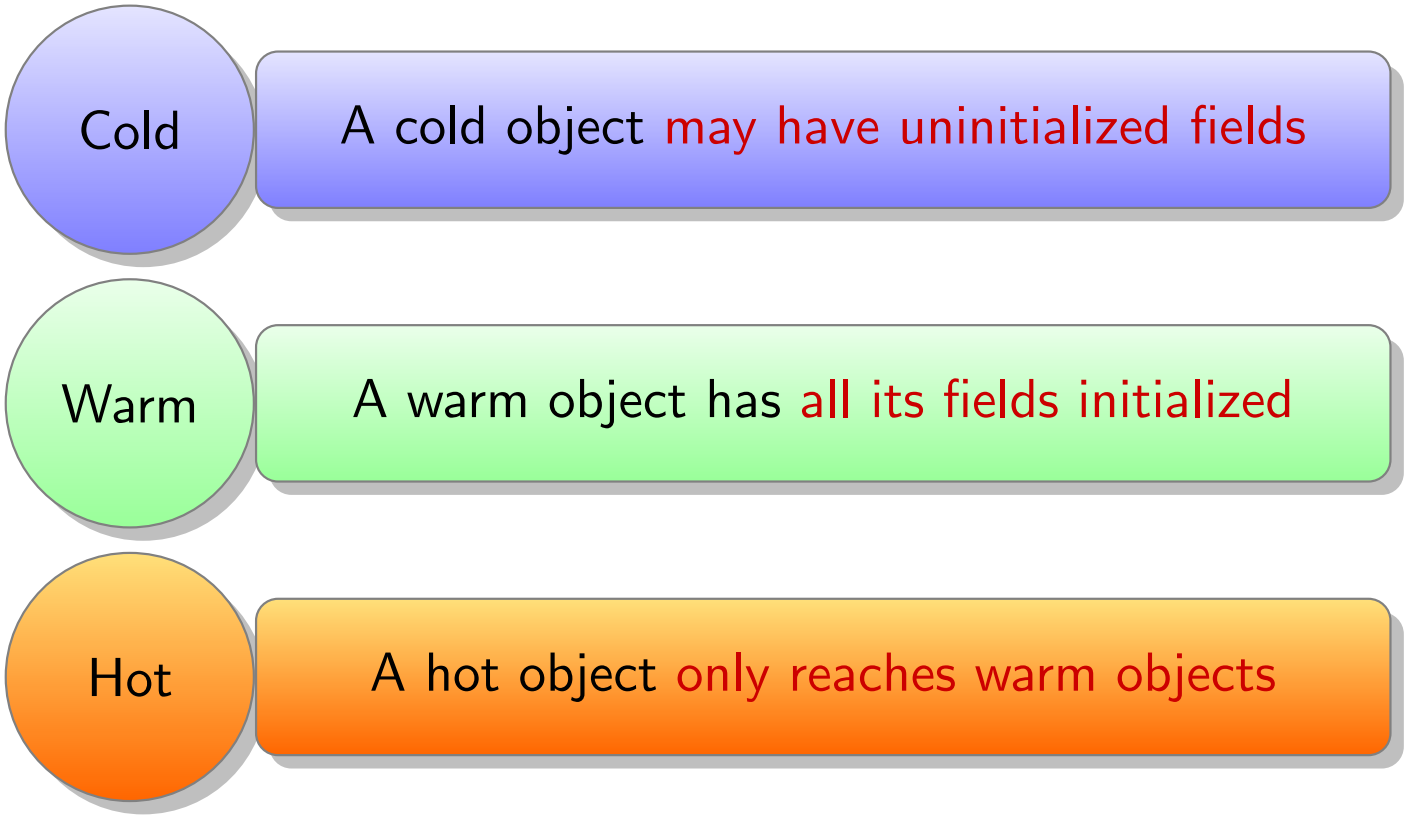
Reachability

An object l' is reachable from l in the heap σ, written $\sigma \vDash l \rightsquigarrow l'$, is defined below:

$$\frac{l \in dom(\sigma)}{\sigma \vDash l \rightsquigarrow l}$$

$$\frac{\sigma \vDash l_0 \rightsquigarrow l_1 \qquad (\_, \omega) = \sigma(l_1) \qquad \exists f.\, \omega(f) = l_2 \qquad l_2 \in dom(\sigma)}{\sigma \vDash l_0 \rightsquigarrow l_2}$$

Abstractions: cold, warm and hot



Formal definition

Definition (Cold)

$$\sigma \vDash l : \mathit{cold} \quad \triangleq \quad l \in \mathit{dom}(\sigma)$$

Definition (Warm)

$$\sigma \vDash l : \mathit{warm} \quad \triangleq \quad \exists (C, \omega) = \sigma(l) \bigwedge \mathit{fields}(C) \subseteq \mathit{dom}(\omega)$$

Definition (Hot)

$$\sigma \vDash l : \mathit{hot} \quad \triangleq \quad l \in \mathit{dom}(\sigma) \bigwedge \forall l'.\sigma \vDash l \rightsquigarrow l' \Longrightarrow \sigma \vDash l' : \mathit{warm}$$

Weak Monotonicity

Definition (Weak Monotonicity)

$$\sigma \preceq \sigma' \quad \triangleq \quad \forall l \in dom(\sigma).\ (C, \omega) = \sigma(l) \implies (C, \omega') = \sigma'(l)$$

Theorem (Weak Monotonicity)

$$[\![e]\!](\sigma, \rho, \psi) = (l, \sigma') \implies \sigma \preceq \sigma'$$

Stackability, formally

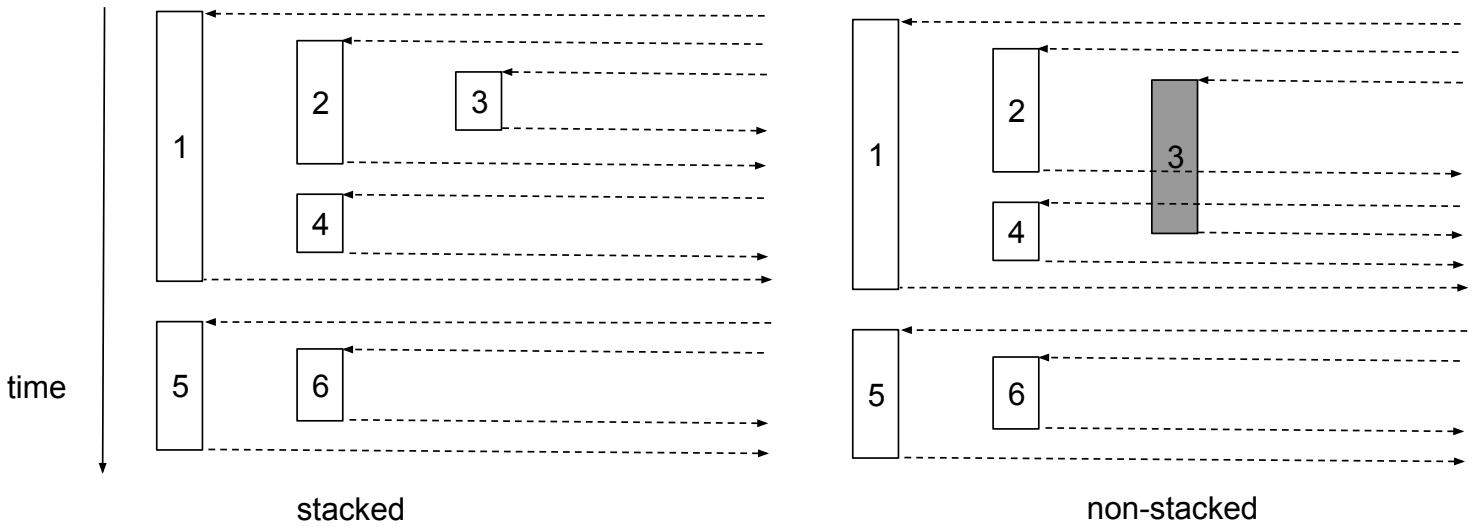
Definition (Stacking)

$$\sigma \ll \sigma' \quad \triangleq \quad \forall l \in \mathit{dom}(\sigma').\ \sigma' \vDash l : \mathit{warm} \bigvee l \in \mathit{dom}(\sigma)$$

Theorem (Stackability)

$$[\![e]\!]\,(\sigma, \rho, \psi) = (l, \sigma') \Longrightarrow \sigma \ll \sigma'$$

Stackability, visually



Scopability, formally

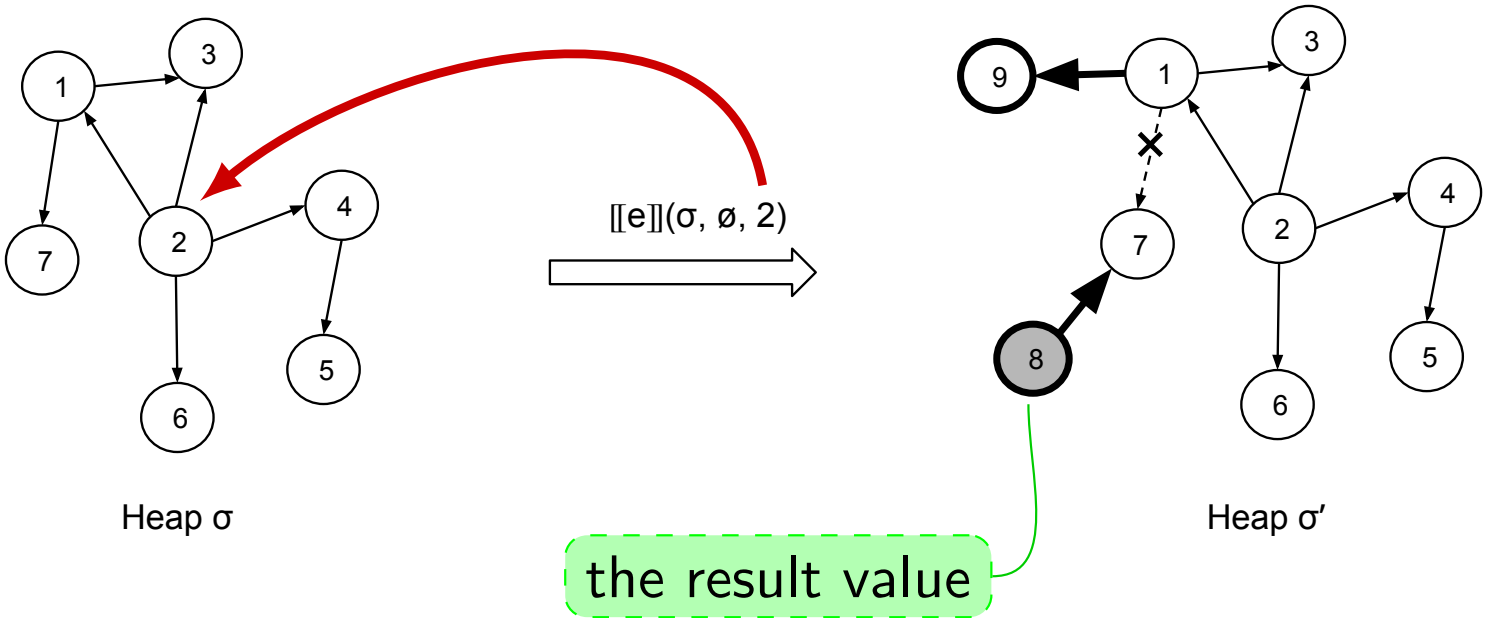
Definition (Scoping)

$$(\sigma, L) \lessdot (\sigma', L') \quad \triangleq \quad \forall l \in \mathit{dom}(\sigma). \quad \sigma' \vDash L' \rightsquigarrow l \Longrightarrow \sigma \vDash L \rightsquigarrow l$$

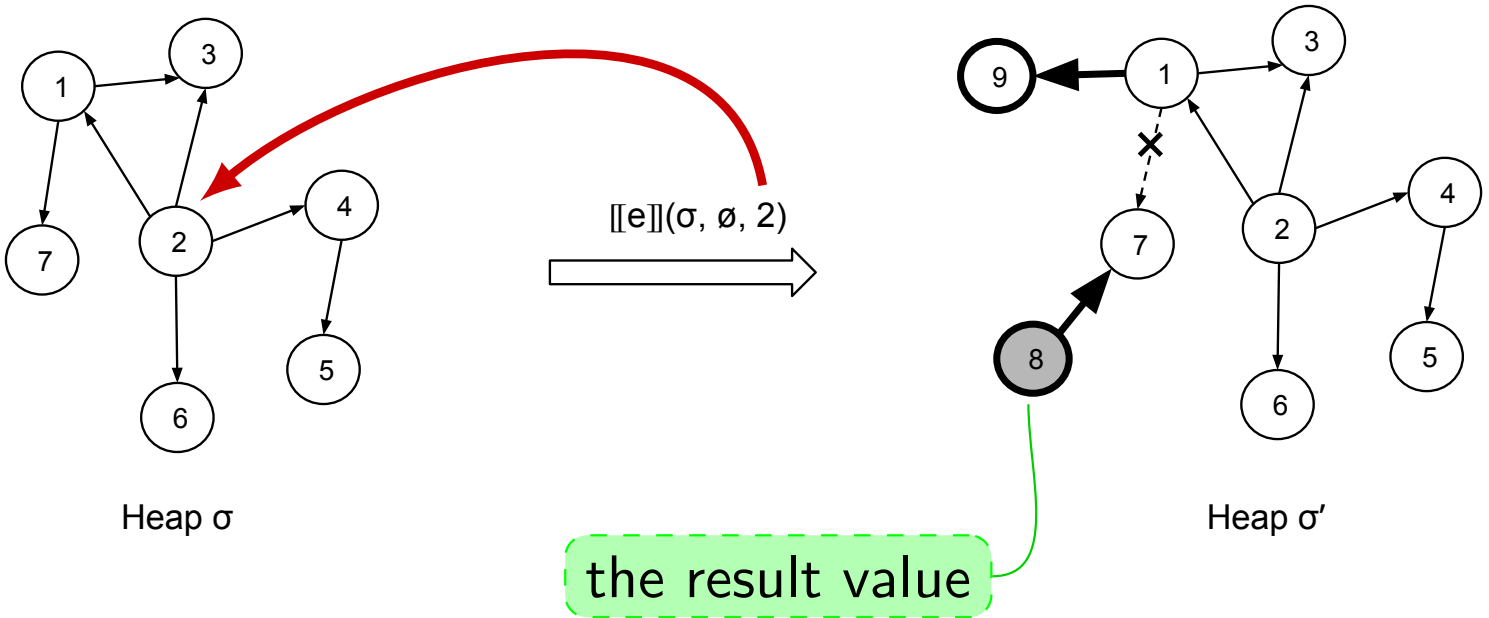
Theorem (Scopability)

$$[\![e]\!]\,(\sigma, \rho, \psi) = (l, \sigma') \Longrightarrow (\sigma, \mathit{codom}(\rho) \cup \{\ \psi\ \}) \lessdot (\sigma', \{\ l\ \})$$

Scopability, visually



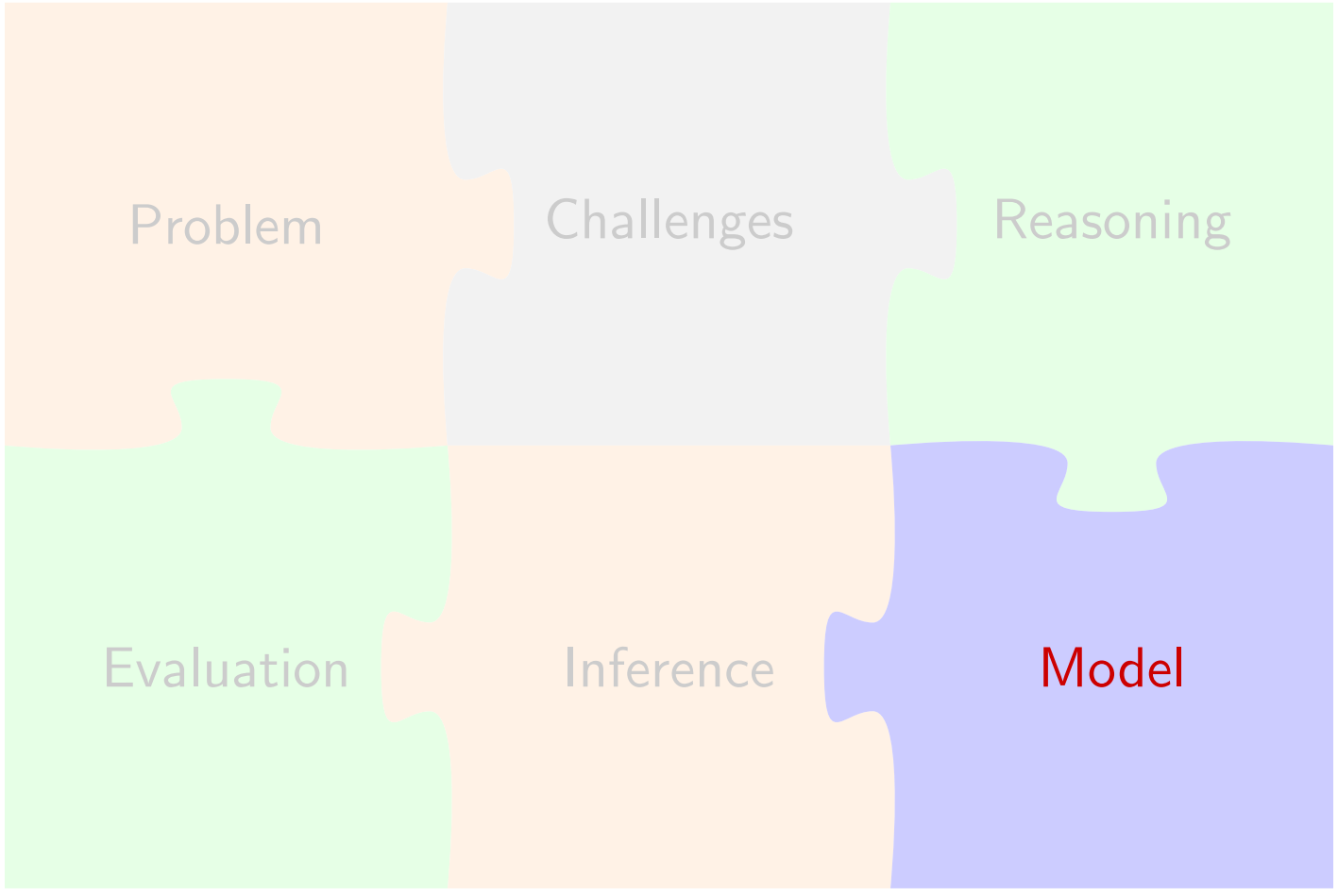
Scopability, visually



$$\frac{\sigma' \vDash 8 \rightsquigarrow 7 \qquad 7 \in \mathit{dom}(\sigma) \qquad \sigma \vDash 2 \rightsquigarrow 7}{(\sigma, 2) \lessdot (\sigma', 8)}$$

Local reasoning theorem

$$\frac{[\![e]\!]\,(\sigma, \rho, \psi) = (l, \sigma') \qquad \sigma \vDash \{\ \psi\ \} \cup \mathit{codom}(\rho) : \mathit{hot}}{\sigma' \vDash l : \mathit{hot}}$$

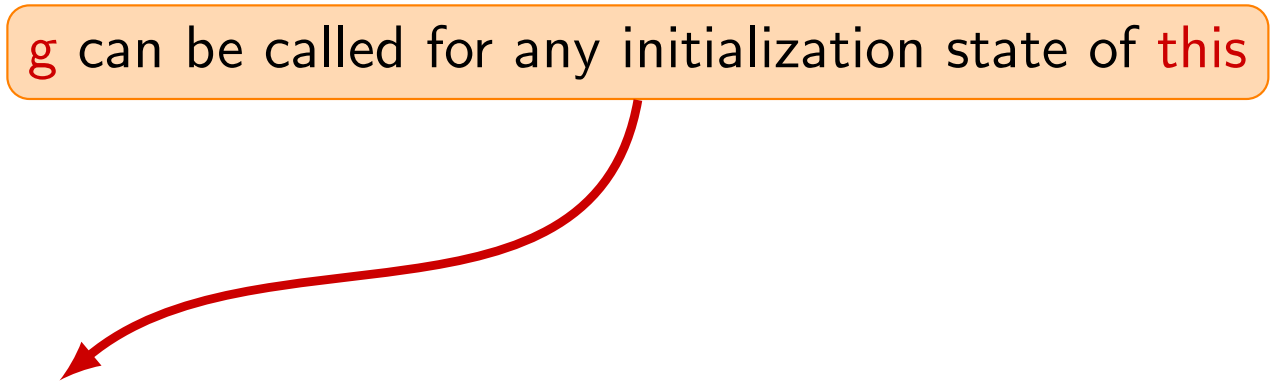

Problem
Challenges
Reasoning
Evaluation
Inference
Model

The basic model

$$\begin{array}{lcl} \mu & ::= & \mathit{cold} \mid \mathit{warm} \mid \mathit{hot} \\ T & ::= & C^{\mu} \end{array}$$

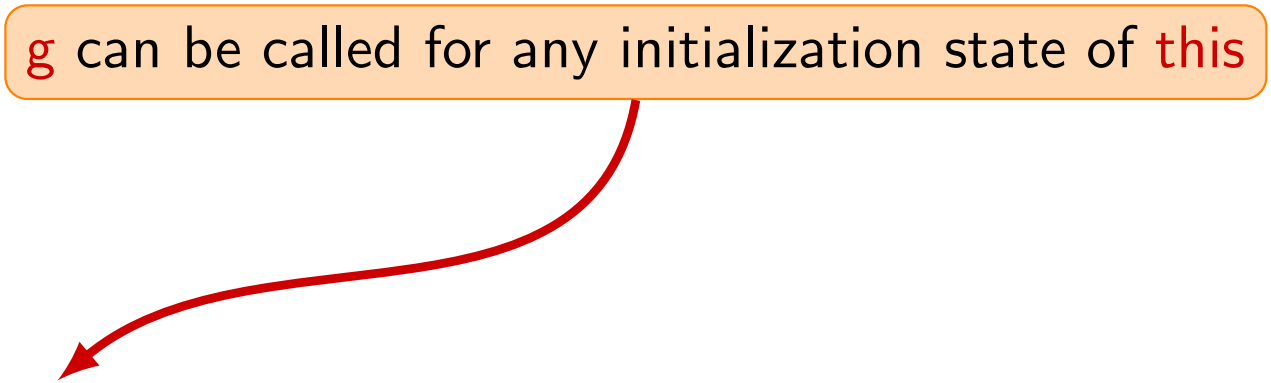
$$\mathit{hot} \sqsubseteq \mathit{warm} \sqsubseteq \mathit{cold}$$

Typestate polymorphism




```
1  class C {
2    // ...
3    def g(): Int = 100
4  }
```

Typestate polymorphism




```
1  class C {
2    // ...
3    def g(): Int = 100
4  }
```

- how to express the polymorphism in the system?
- how to avoid syntactic overhead for polymorphism?

Flow-sensitivity and typestate polymorphism

In a flow-sensitive system, we need to resort to parametric polymorphic because a method has to represent the typestates of *this* both before and after the method call.

```
1  class C {
2    def g(): Int = 100 // g:  ∀M.M ⇝ M
3  }
```

Qi, Xin and Andrew C. Myers. "Masked types for sound object initialization." POPL '09

Flow-insensitivity and typestate polymorphism

In a flow-insensitive system, we may resort to subtyping polymorphism to support *typestate polymorphism*.

```
1  class C {
2    @cold def g(): Int = 100
3  }
```

Summers, Alexander J. and Peter Müller. "Freedom before commitment." OOPSLA '11

Strong monotonicity

Hot/warm objects continue to be hot/warm:

$$\sigma \preceq \sigma' \quad \triangleq \quad \forall l \in dom(\sigma).\ \sigma \vDash l : \mu \Longrightarrow \sigma' \vDash l : \mu$$

Fähndrich, Manuel and K. Rustan M. Leino. "Heap Monotonic Typestate." (2003).

Perfect monotonicity

Hot/warm fields continue to be hot/warm:

$$\sigma \preceq \sigma' \quad \triangleq \quad \forall l \in dom(\sigma).\quad (C,\omega) = \sigma(l) \Longrightarrow$$
$$(C,\omega') = \sigma'(l) \quad \bigwedge$$
$$\forall f \in dom(\omega).\sigma \vDash \omega(f) : \mu \Longrightarrow \sigma' \vDash \omega'(f) : \mu$$

Typing rules - T-New

$$\frac{\overline{T_i} = \mathit{constrType}(C) \qquad \Gamma;\, T \vdash e_i : C_i^{\mu_i} \qquad C_i^{\mu_i} <: T_i \qquad \mu = (\sqcup\, \mu_i) \sqcap \mathit{warm}}{\Gamma;\, T \vdash \mathit{new}\ C(\overline{e}) : C^{\mu}}\ (\text{T-New})$$

local reasoning and stackability

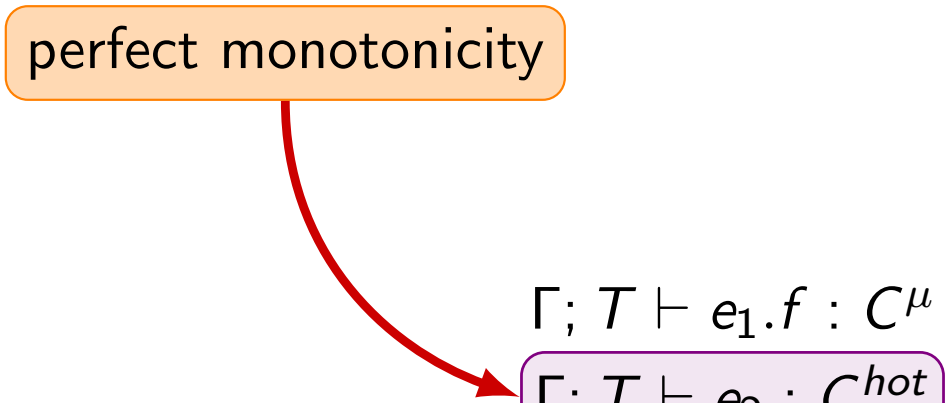
Typing rules - T-Invoke

typestate polymorphism

$$\frac{\Gamma; T \vdash e : C^{\mu_0} \quad (\mu_m, \overline{T_i}, D^{\mu_r}) = methodType(C, m) \quad \mu_0 \sqsubseteq \mu_m \quad \Gamma; T \vdash e_i : D_i^{\mu_i} \quad D_i^{\mu_i} <: T_i \quad \mu = (\sqcup\, \mu_i = hot)?hot : \mu_r}{\Gamma; T \vdash e.m(\overline{e}) : D^{\mu}} \text{(T-Invoke)}$$

local reasoning

Typing rules - T-Block



$$\frac{\Gamma; T \vdash e_1.f : C^{\mu} \qquad \Gamma; T \vdash e_2 : C^{hot} \qquad \Gamma; T \vdash e : T_1}{\Gamma; T \vdash e_1.f = e_2; e : T_1} \quad \text{(T-Block)}$$

Typing rules - selection

$$\frac{\Gamma; T \vdash e : D^{hot} \qquad C^{\mu} = fieldType(D, f)}{\Gamma; T \vdash e.f : C^{hot}} \text{ (T-SelHot)}$$

$$\frac{\Gamma; T \vdash e : D^{warm} \qquad U = fieldType(D, f)}{\Gamma; T \vdash e.f : U} \text{ (T-SelWarm)}$$

Authority, flow-insensitivity and strong updates

The meta-theory depends on authority:

$$\frac{\forall l \in dom(\Sigma).\Sigma(l) = C^{\mu} \Longrightarrow \Sigma'(l) = C^{\mu}}{\Sigma \rhd \Sigma'}$$

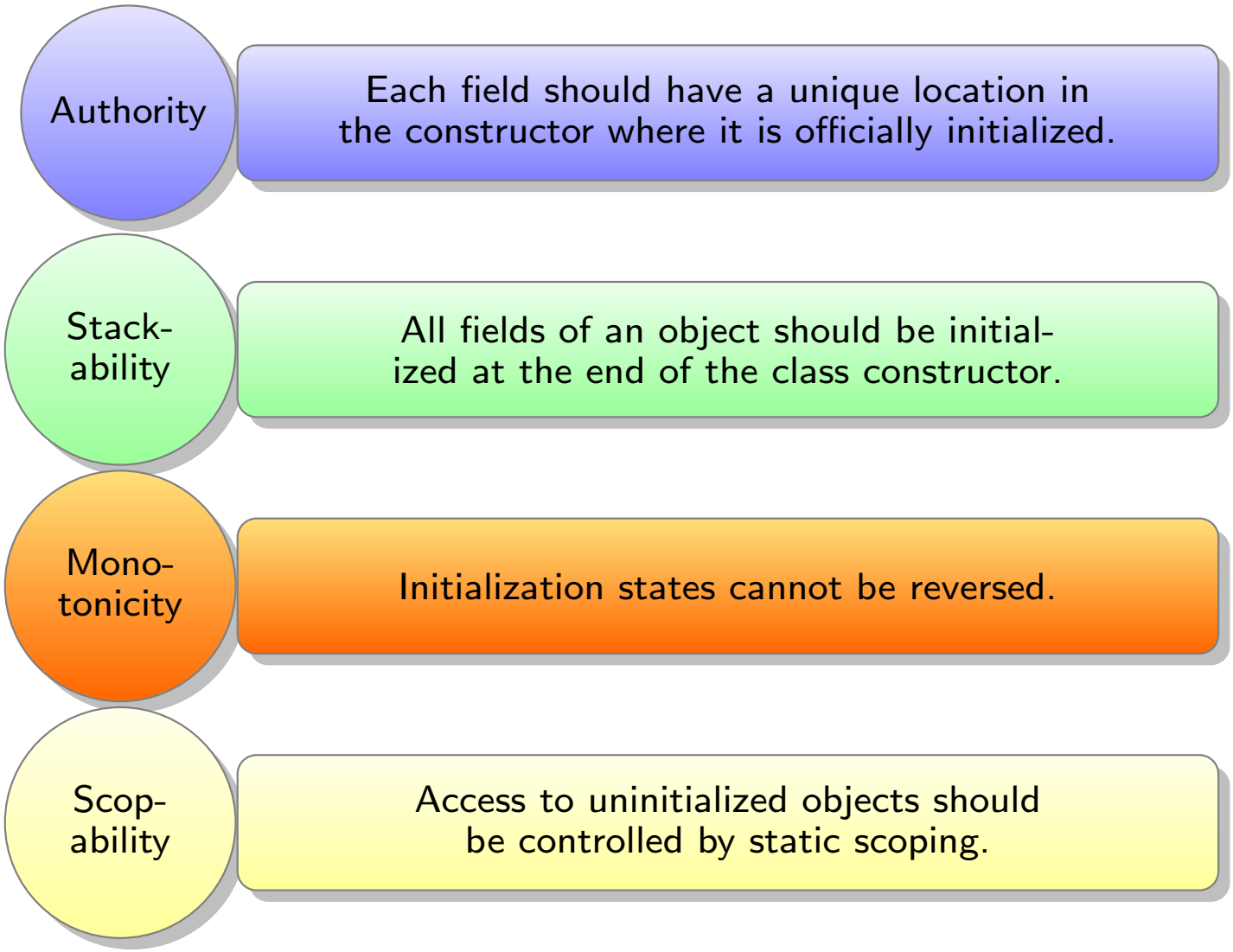
Authority, flow-insensitivity and strong updates

The meta-theory depends on authority:

$$\frac{\forall l \in dom(\Sigma).\Sigma(l) = C^{\mu} \Longrightarrow \Sigma'(l) = C^{\mu}}{\Sigma \rhd \Sigma'}$$

In a flow-insensitive system without aliasing tracking, it is only safe to perform strong updates of initialization states via the outstanding alias `this` in a local flow inside the constructor.

Design principles of initialization



Principled design of constructors

To align with the principles, we advocate

- class parameters
- mandatory field initializers

Principled design of constructors

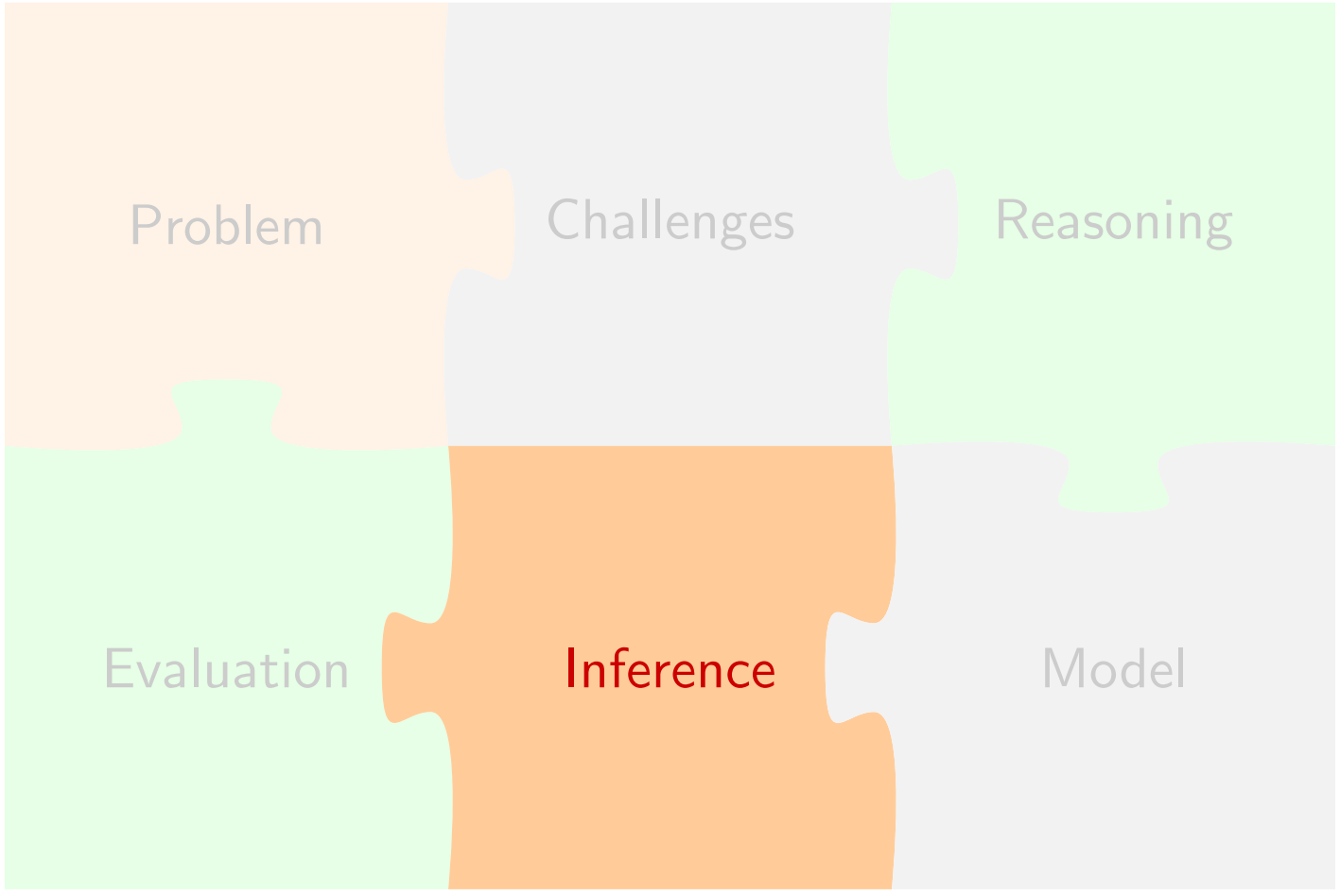
To align with the principles, we advocate

- class parameters
- mandatory field initializers

```
1  class RemoteDoc(url: String) {
2    val localFile: String = url.hashCode
3    def name: String = localFile
4  }
```

Languages in industry

language	`class C(f:Int)`	`var f:Int = e`	**year**
Java	○	○	1995
C#	○	○	2000
D	○	○	2001
Scala	●	◐	2003
Ceylon	●	○	2011
Dart	○	○	2011
Kotlin	●	◐	2011
TypeScript	○	○	2012
Crystal	○	○	2014
Swift	○	○	2014


Problem
Challenges
Reasoning
Evaluation
Inference
Model

What is wrong with type-based approach?

To be honest, the reason this approach has likely not yet seen widespread use is that the cost is not commensurate with the benefit. ... For systems programmers, this makes sense. For many other programmers, it would be useless ceremony with no perceived value.

— Joe Duffy, "On partially-constructed objects"

http://joeduffyblog.com/2010/06/27/on-partiallyconstructed-objects/

Drawbacks of type-based approach

- Annotation overhead
- Inadequate to handle traits, inner classes, properties
- Does not handle inheritance well
- Difficult to integrate in compilers

Type-and-effect systems

$$\frac{\Gamma, x : S \vdash t : T \mid \epsilon}{\Gamma \vdash \lambda x{:}S.t : S \xrightarrow{\epsilon} T \mid \emptyset} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t_1 : S \xrightarrow{\epsilon_3} T \mid \epsilon_1 \qquad \Gamma \vdash t_2 : S \mid \epsilon_2}{\Gamma \vdash t_1\ t_2 : T \mid \epsilon_1 \cup \epsilon_2 \cup \epsilon_3} \quad \text{(T-App)}$$

Lucassen, J.M., Gifford, D.K. (1988). Polymorphic effect systems. POPL '88.

Potentials and effects

Potentials represent aliasing information of objects *possibly* under initialization.

$$\begin{array}{lcl} \beta & ::= & \mathit{this} \mid \mathit{warm}[C] \mid \mathit{cold} \\ \pi & ::= & \beta \mid \pi.f \mid \pi.m \end{array}$$

Potentials and effects

Potentials represent aliasing information of objects *possibly* under initialization.

$$\begin{array}{lll} \beta & ::= & \mathit{this} \mid \mathit{warm}[C] \mid \mathit{cold} \\ \pi & ::= & \beta \mid \pi.f \mid \pi.m \end{array}$$

Effects represent operations on objects *possibly* under initialization.

$$\phi \quad ::= \quad \pi.f! \mid \pi.m\Diamond \mid \pi\uparrow$$

Potentials and effects

Potentials represent aliasing information of objects *possibly* under initialization.

$$\begin{array}{lcl} \beta & ::= & \mathit{this} \mid \mathit{warm}[C] \mid \mathit{cold} \\ \pi & ::= & \beta \mid \pi.f \mid \pi.m \end{array}$$

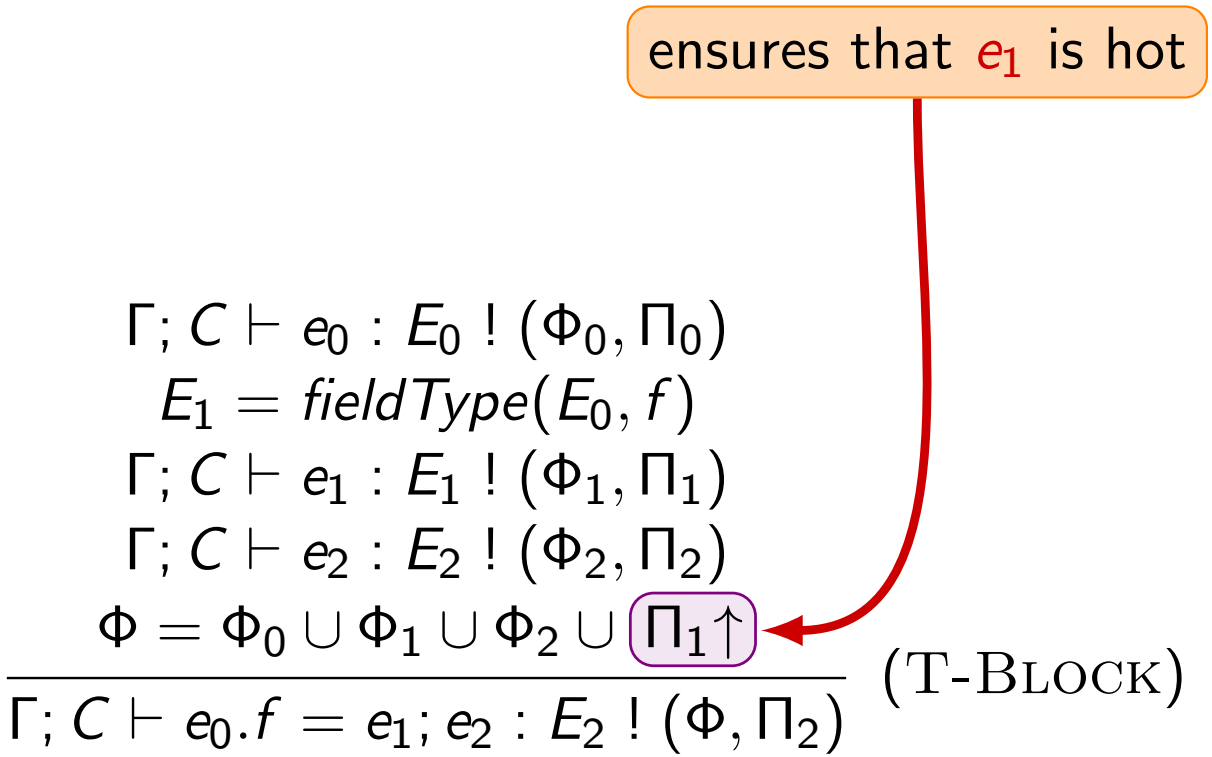
Effects represent operations on objects *possibly* under initialization.

$$\phi \quad ::= \quad \pi.f! \mid \pi.m\Diamond \mid \pi\uparrow$$

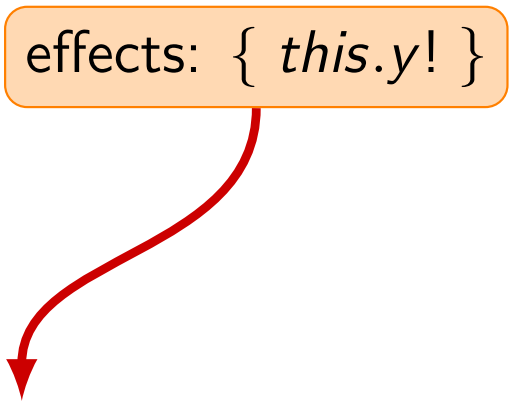
$\Gamma; C \vdash e : D \,!\, (\Phi, \Pi)$

If $\Pi = \emptyset$, the value of e must be hot.

Expression Typing - Block



Example – field access effect

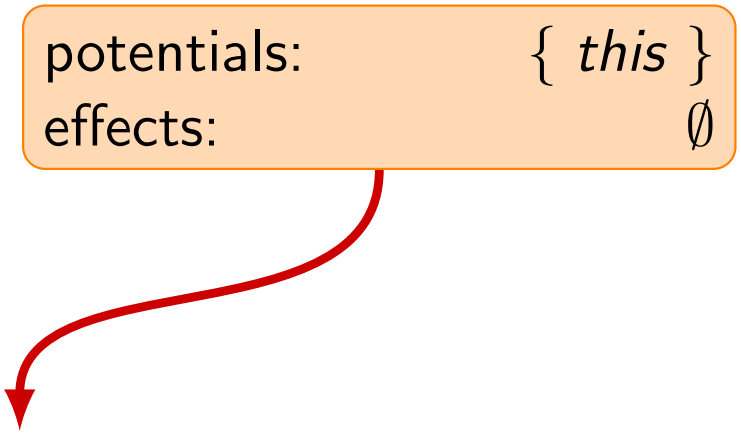



```
1  class C {
2      var x: Int = this.y
3      var y: Int = 10
4  }
```

Example – potentials

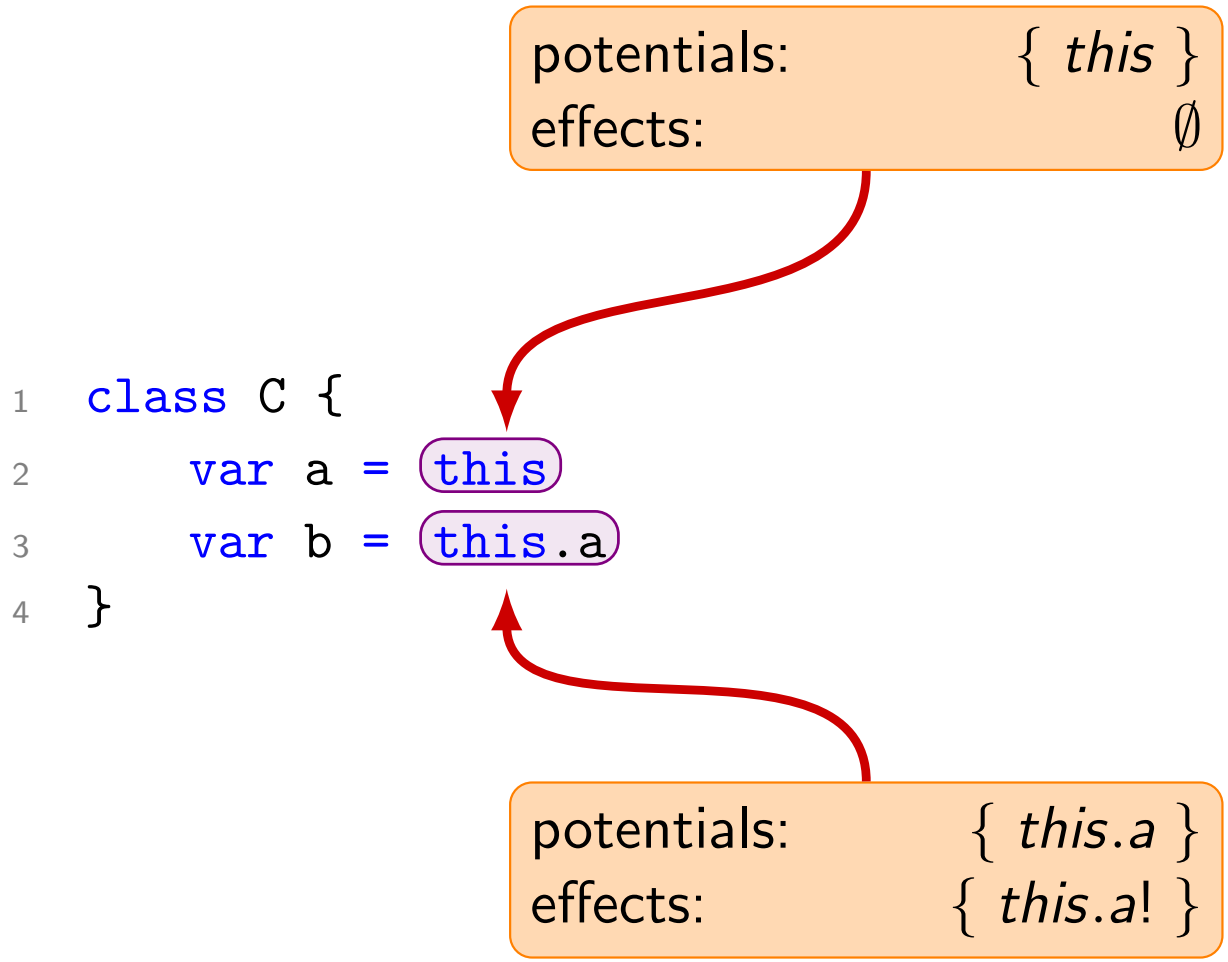
```
1  class C {
2      var a = this
3      var b = this.a
4  }
```

Example – potentials

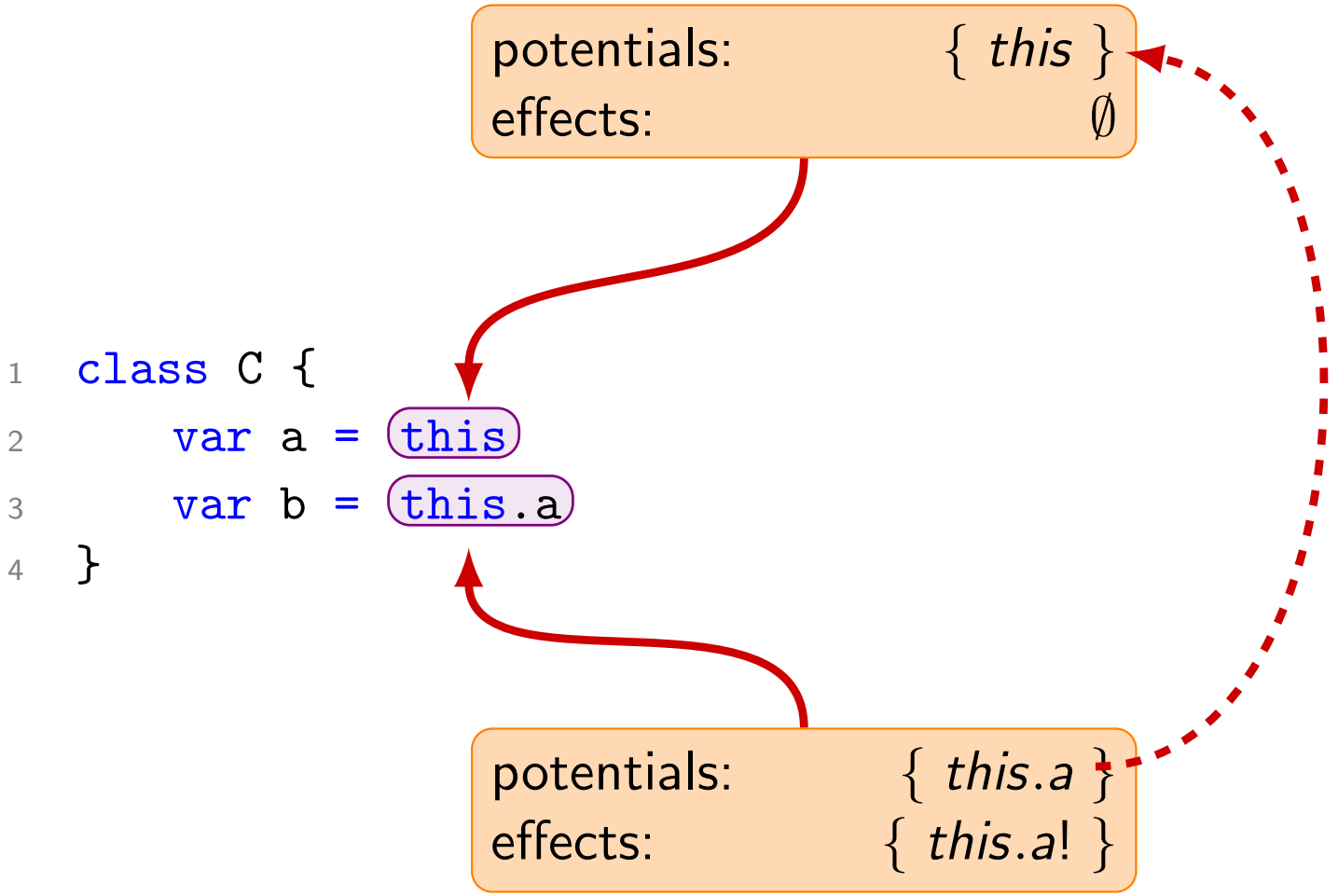



```
1  class C {
2      var a = this
3      var b = this.a
4  }
```

Example – potentials



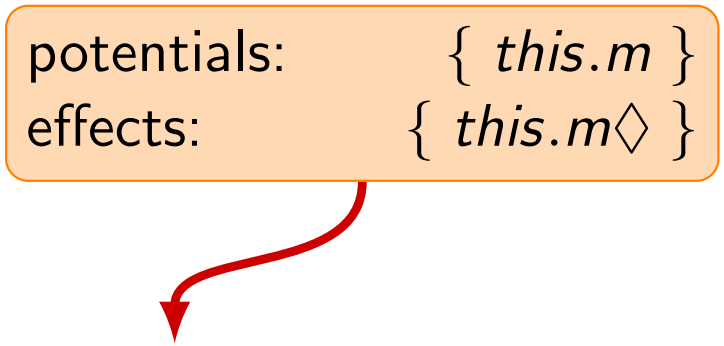
Example – potentials



Example – method call effects

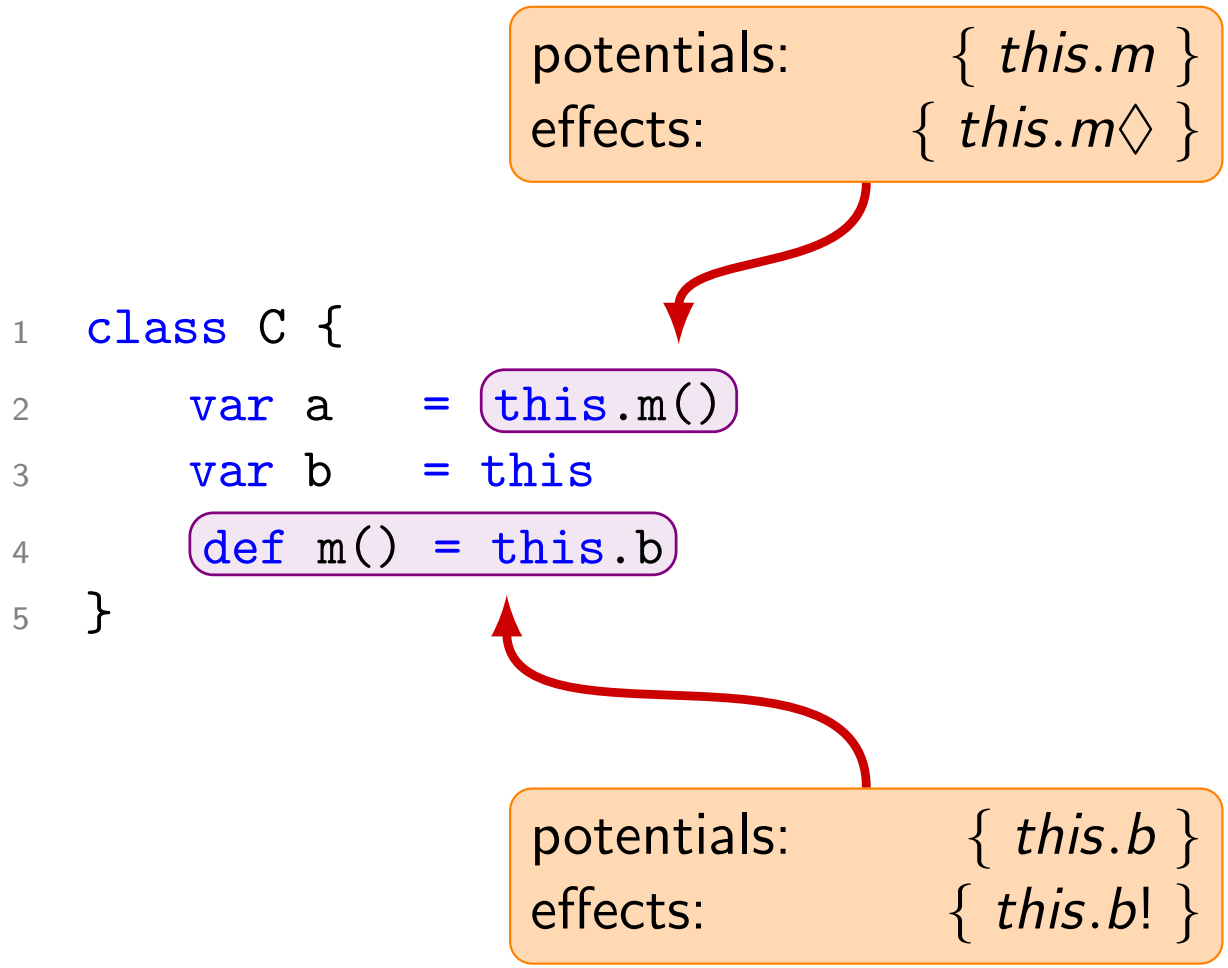
```
1  class C {
2      var a   = this.m()
3      var b   = this
4      def m() = this.b
5  }
```

Example – method call effects

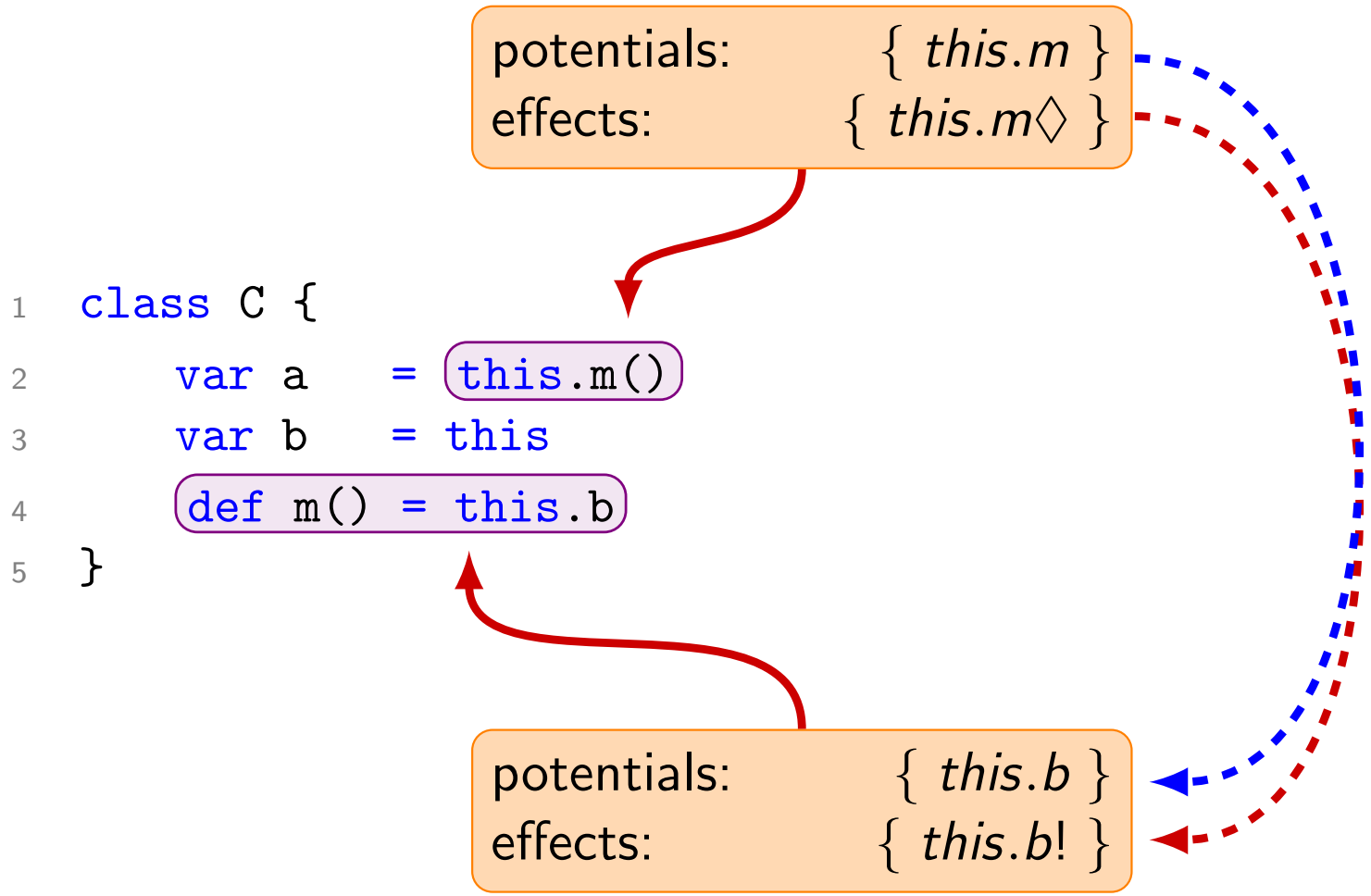



```
1  class C {
2      var a   = this.m()
3      var b   = this
4      def m() = this.b
5  }
```

Example – method call effects



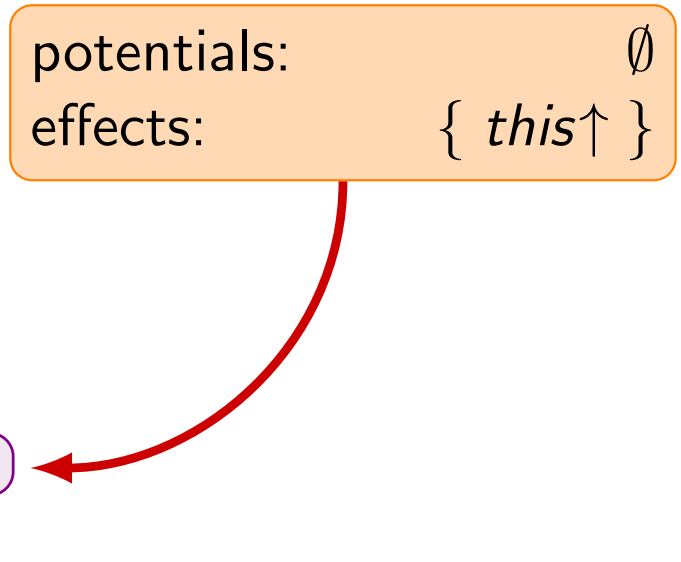
Example – method call effects



Example – promotion effects

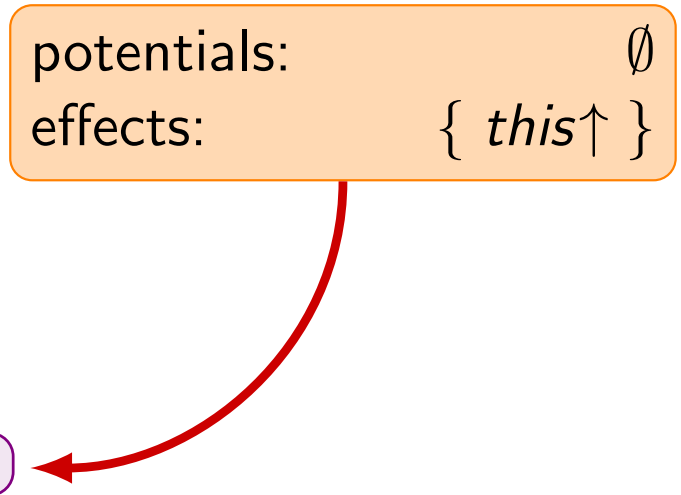
```
1  class C(fun: C => Int) {
2      var x: Int = fun(this)
3  }
```

Example – promotion effects




```
1  class C(fun: C => Int) {
2      var x: Int = fun(this)
3  }
```

Example – promotion effects




```
1  class C(fun: C => Int) {
2      var x: Int = fun(this)
3  }
```

Promotion is the same as requiring a value to be hot.

Two-phase checking

```
1  class C {
2    var a:  Int = h()
3    def h(): Int = g()
4    def g(): Int = h()
5  }
```

Two-phase checking

```
1  class C {
2    var a:  Int = h()
3    def h(): Int = g()
4    def g(): Int = h()
5  }
```

First phase

method	effects	potentials
h	$\{\ this.g\Diamond\ \}$	$\{\ this.g\ \}$
g	$\{\ this.h\Diamond\ \}$	$\{\ this.h\ \}$

Two-phase checking

```
1  class C {
2    var a:  Int = h()
3    def h(): Int = g()
4    def g(): Int = h()
5  }
```

First phase

method	effects	potentials
h	$\{\ this.g\Diamond\ \}$	$\{\ this.g\ \}$
g	$\{\ this.h\Diamond\ \}$	$\{\ this.h\ \}$

Second phase

$fixpoint(\{\ this.h\Diamond\ \}) = \{\ this.g\Diamond, this.h\Diamond\ \}$

Two-phase checking

```
1  class C {
2    var a:  Int = h()
3    def h(): Int = g()
4    def g(): Int = h()
5  }
```

First phase

method	effects	potentials
h	$\{\ this.g\Diamond\ \}$	$\{\ this.g\ \}$
g	$\{\ this.h\Diamond\ \}$	$\{\ this.h\ \}$

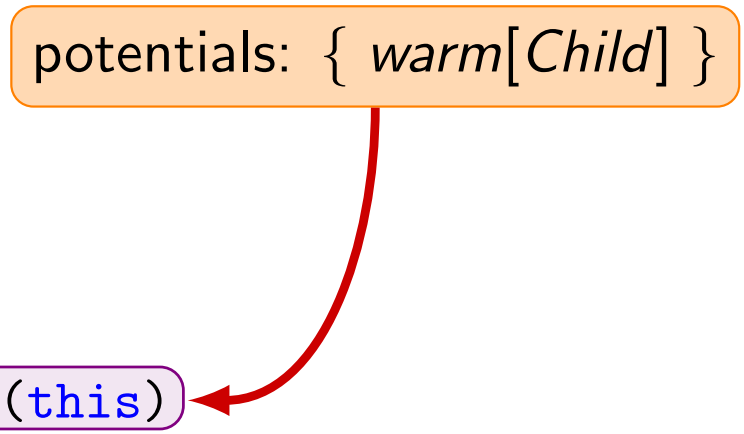
Second phase

$fixpoint(\{\ this.h\Diamond\ \}) = \{\ this.g\Diamond, this.h\Diamond\ \}$ ✔

Cyclic data structures

```
1  class Parent {
2    val child: Child = new Child(this)
3    child.name  // OK
4  }
5
6  class Child(parent:  Parent @cold) {
7    val name = "Jack"
8  }
```

Cyclic data structures



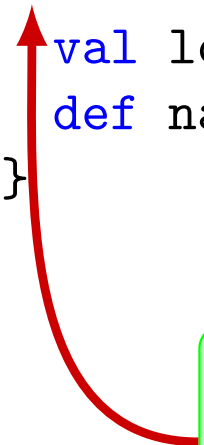

```
1  class Parent {
2    val child: Child = new Child(this)
3    child.name  // OK
4  }
5
6  class Child(parent:  Parent @cold) {
7    val name = "Jack"
8  }
```

Full-construction analysis

```
1  abstract class AbstractFile {
2    def name: String
3    val extension: String = name.substring(4)
4  }
5
6  class RemoteFile(url:String) extends AbstractFile {
7    val localFile: String = url.hashCode
8    def name:  String = localFile
9  }
```

Full-construction analysis

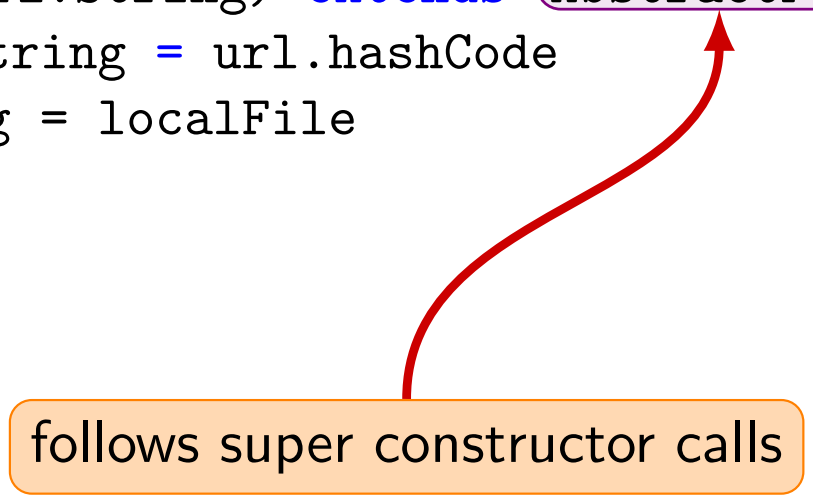
```
1  abstract class AbstractFile {
2    def name: String
3    val extension: String = name.substring(4)
4  }
5
6  class RemoteFile(url:String) extends AbstractFile {
7    val localFile: String = url.hashCode
8    def name:  String = localFile
9  }
```



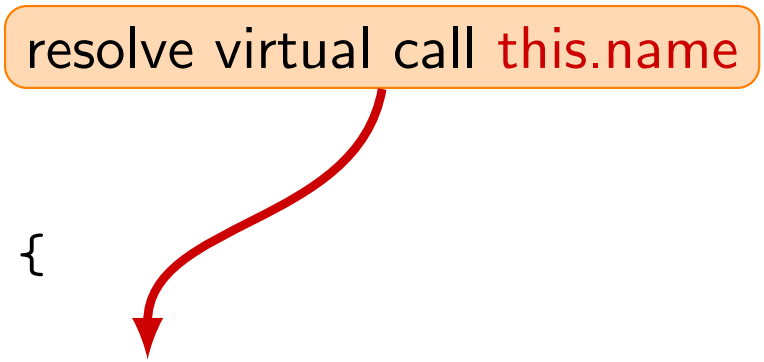
Analysis entry point: primary constructor of concrete classes

Full-construction analysis

```
1  abstract class AbstractFile {
2    def name: String
3    val extension: String = name.substring(4)
4  }
5
6  class RemoteFile(url:String) extends AbstractFile {
7    val localFile: String = url.hashCode
8    def name:  String = localFile
9  }
```



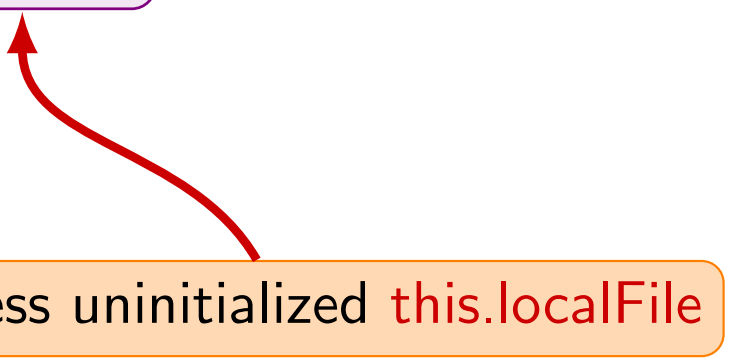
Full-construction analysis




```
1  abstract class AbstractFile {
2    def name: String
3    val extension: String = name.substring(4)
4  }
5
6  class RemoteFile(url:String) extends AbstractFile {
7    val localFile: String = url.hashCode
8    def name:  String = localFile
9  }
```

Full-construction analysis

```
1  abstract class AbstractFile {
2    def name: String
3    val extension: String = name.substring(4)
4  }
5
6  class RemoteFile(url:String) extends AbstractFile {
7    val localFile: String = url.hashCode
8    def name:  String = localFile
9  }
```

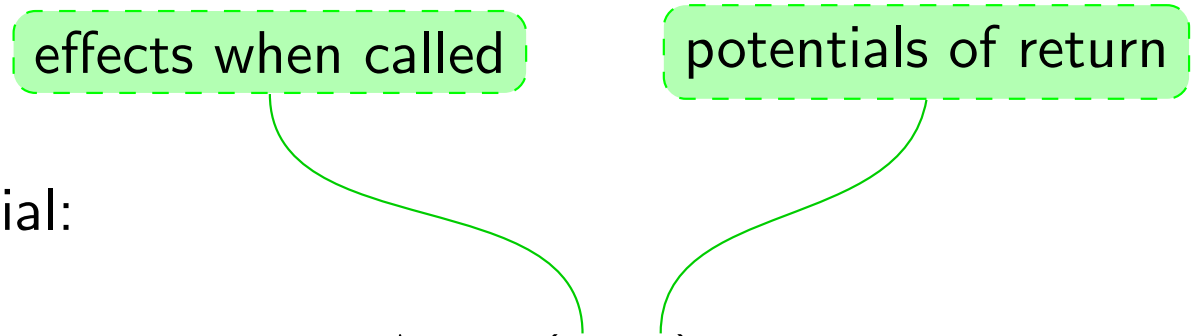


Functions

Function potential:

$$\pi \quad ::= \quad \ldots \mid \mathit{Fun}(\Phi, \Pi)$$

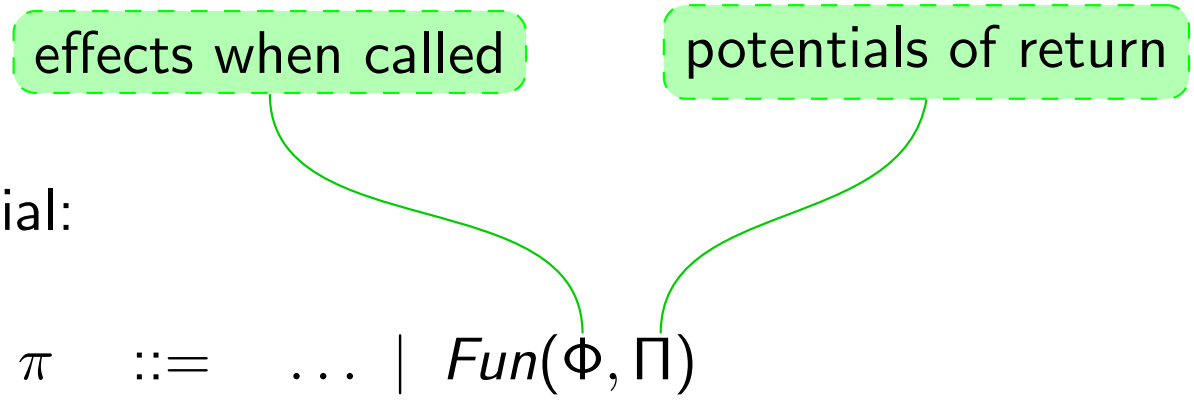
Functions



Function potential:

$$\pi \quad ::= \quad \ldots \mid \mathit{Fun}(\Phi, \Pi)$$

Functions

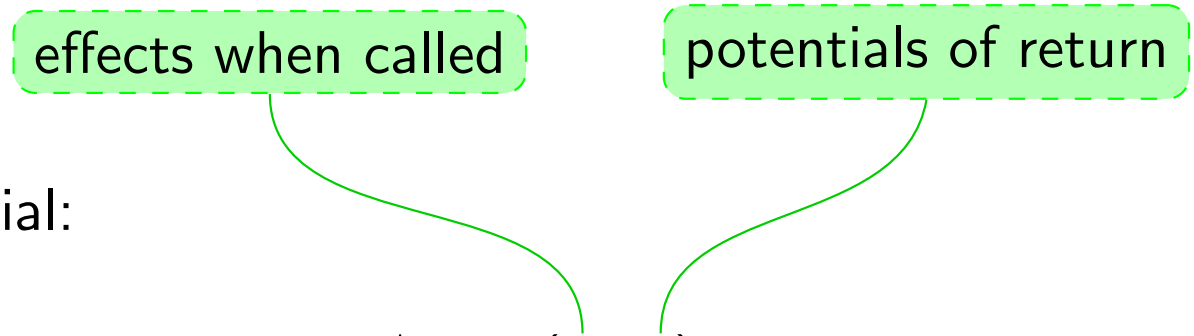


Function potential:

$$\pi \quad ::= \quad \ldots \mid \mathit{Fun}(\Phi, \Pi)$$

```
1  class Rec {
2    val even = (n: Int) => n == 0 || odd(n - 1)
3    val odd = (n: Int) => n == 1 || even(n - 1)
4    val flag: Boolean  = odd(6)
5  }
```

Functions



Function potential:

$$\pi \quad ::= \quad \ldots \mid \mathit{Fun}(\Phi, \Pi)$$

```
1  class Rec {
2    val even = (n: Int) => n == 0 || odd(n - 1)
3    val odd = (n: Int) => n == 1 || even(n - 1)
4    val flag: Boolean  = odd(6)
5  }
```

Inner classes

$$
\begin{array}{lcl}
\beta & ::= & \mathit{this} \mid \mathit{cold} \mid \mathit{warm}(C, \pi) \\
\pi & ::= & \beta \mid \pi.f \mid \pi.m \mid \pi.\mathit{outer}(C) \\
\phi & ::= & \pi\uparrow \mid \pi.f! \mid \pi.m\Diamond \mid \pi.\mathit{init}(C)
\end{array}
$$

Inner classes

$$
\begin{array}{lcl}
\beta & ::= & \mathit{this} \mid \mathit{cold} \mid \mathit{warm}(C, \pi) \\
\pi & ::= & \beta \mid \pi.f \mid \pi.m \mid \pi.\mathit{outer}(C) \\
\phi & ::= & \pi\uparrow \mid \pi.f! \mid \pi.m\Diamond \mid \pi.\mathit{init}(C)
\end{array}
$$

- $\mathit{warm}(C, \pi)$ — outer potential π
- $\pi.\mathit{outer}(C)$ — potential of outer
- $\pi.\mathit{init}(C)$ — constructor effect

Inner classes

$$\beta \quad ::= \quad \mathit{this} \mid \mathit{cold} \mid \mathit{warm}(C, \pi)$$

outer potential π

$$\pi \quad ::= \quad \beta \mid \pi.f \mid \pi.m \mid \pi.\mathit{outer}(C)$$

potential of outer

$$\phi \quad ::= \quad \pi\uparrow \mid \pi.f! \mid \pi.m\Diamond \mid \pi.\mathit{init}(C)$$

constructor effect

```
1  class C {
2    private var counter = 0
3    class D { counter += 1 }
4    val d = new D
5  }
```

Inner classes

$$\beta \quad ::= \quad this \mid cold \mid warm(C, \pi)$$
$$\pi \quad ::= \quad \beta \mid \pi.f \mid \pi.m \mid \pi.outer(C)$$
$$\phi \quad ::= \quad \pi\uparrow \mid \pi.f! \mid \pi.m\Diamond \mid \pi.init(C)$$

outer potential π

potential of outer

constructor effect

```
1  class C {
2    private var counter = 0
3    class D { counter += 1 }
4    val d = new D
5  }
```

$warm(D, this).init(D)$

$warm(D, this).outer(D).counter!$

$this.counter!$

$this.outer(D).counter!$

Termination - limit length

```
1  class C {
2    var a = this.g
3    def g = this.g.g
4  }
```

Termination - limit length

```
1  class C {
2    var a = this.g
3    def g = this.g.g
4  }
```

Infinite Loop

$$
\begin{aligned}
& this.g\Diamond \\
\Longrightarrow\ & this.g.g\Diamond \\
\Longrightarrow\ & this.g.g.g\Diamond \\
\Longrightarrow\ & this.g.g.g.g\Diamond \\
\Longrightarrow\ & \ldots
\end{aligned}
$$

Termination - limit length

```
1  class C {
2    var a = this.g
3    def g = this.g.g
4  }
```

Infinite Loop

$$
\begin{aligned}
& this.g\Diamond \\
\Longrightarrow\ & this.g.g\Diamond \\
\Longrightarrow\ & this.g.g.g\Diamond \\
\Longrightarrow\ & this.g.g.g.g\Diamond \\
\Longrightarrow\ & \ldots
\end{aligned}
$$

Limit length

$$
\begin{aligned}
& this.g\Diamond \\
\Longrightarrow\ & this.g.g\Diamond \\
\Longrightarrow\ & this.g.g\uparrow \\
\Longrightarrow\ & \ldots \\
\Longrightarrow\ & \checkmark
\end{aligned}
$$

Termination - widening (1)

```
1  class B {
2    class C extends B
3    val c: C = new C
4  }
```

Termination - widening (1)

```
1  class B {
2    class C extends B
3    val c: C = new C
4  }
```

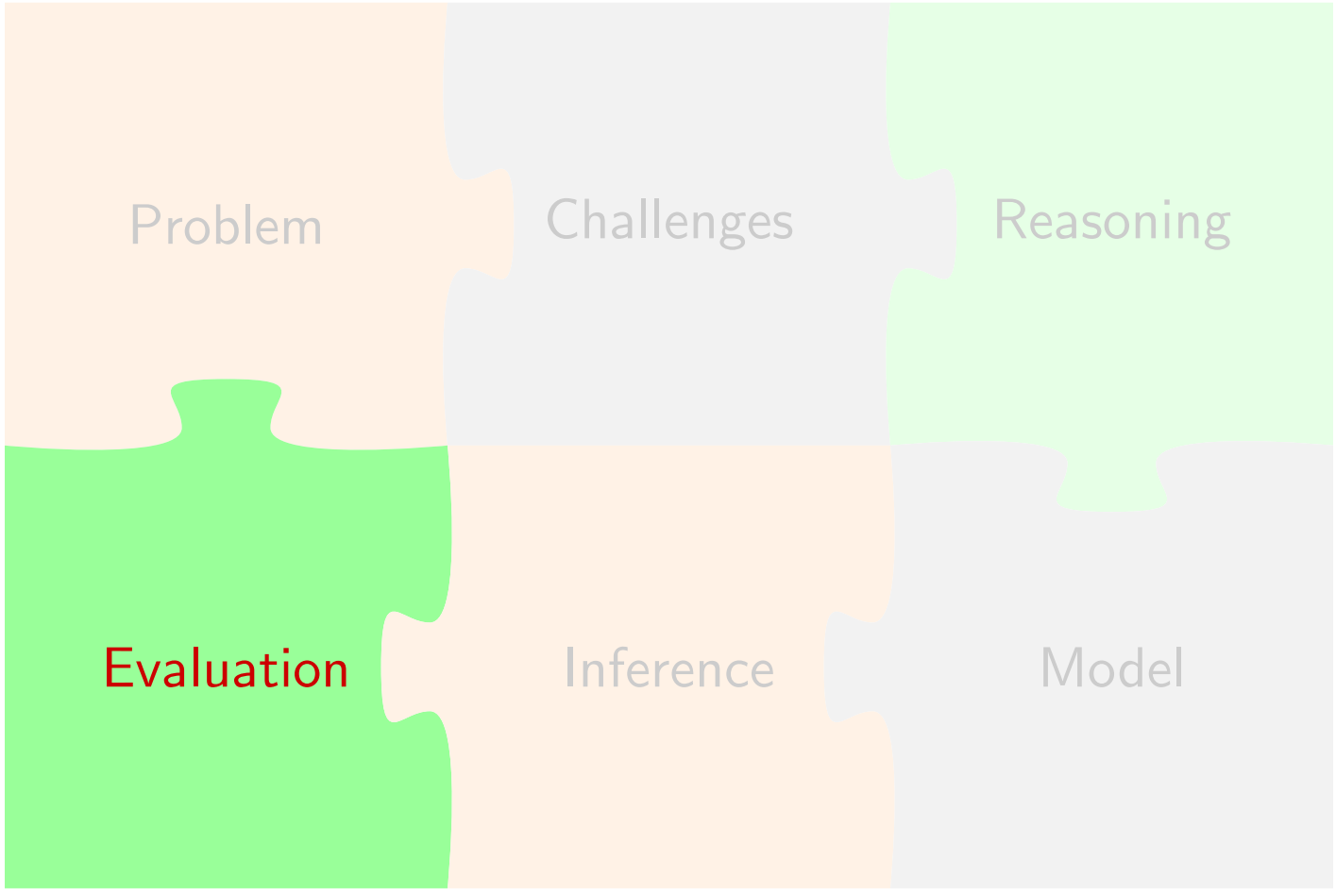
$$
\begin{aligned}
& warm(C, this).init(C) \\
\Longrightarrow\ & warm(C, warm(C, this)).init(C) \\
\Longrightarrow\ & warm(C, warm(C, warm(C, this))).init(C) \\
\Longrightarrow\ & \ldots
\end{aligned}
$$

Termination - widening (2)

```
1  class B {
2    class C extends B
3    val c: C = new C
4  }
```

It is safe to widen $\mathit{warm}(C, \pi)$ to cold:

$$
\begin{aligned}
& \mathit{warm}(C, \mathit{this}).\mathit{init}(C) \\
\Longrightarrow\ & \mathit{warm}(C, \mathit{warm}(C, \mathit{cold})).\mathit{init}(C) \\
\Longrightarrow\ & \mathit{warm}(C, \mathit{warm}(C, \mathit{cold})).\mathit{init}(C) \\
\Longrightarrow\ & \checkmark
\end{aligned}
$$


Problem
Challenges
Reasoning
Evaluation
Inference
Model

Implementation

Implemented in Scala 3 compiler, under the option `-Ycheck-init`.

- Lazy summarization of methods
- Takes advantage of TASTy for separate compilation
- Zero changes to core type system

Debuggability

```
1  class Greeting {
2    val message: String = this.welcome()
3    val name: String  = "Jack"
4    def welcome()     = "Hello, " + name    // error
5  }
```

Debuggability

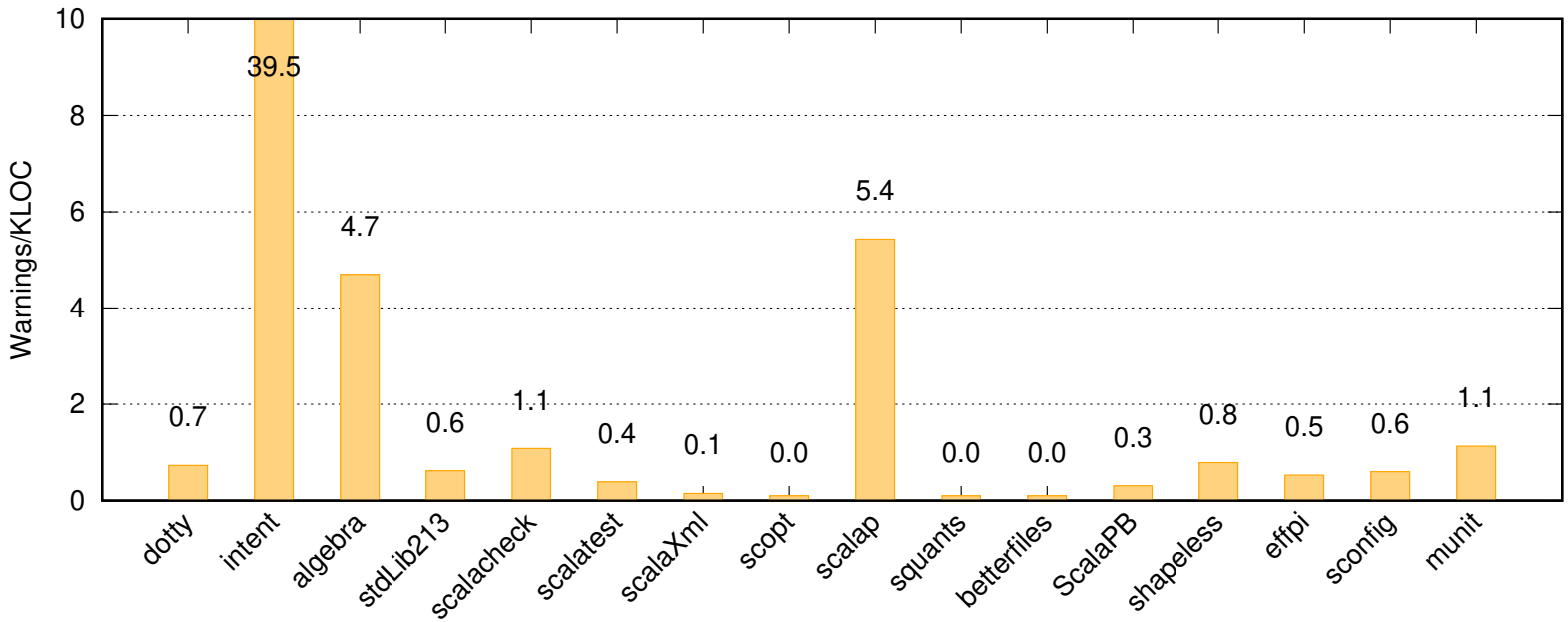
```scala
class Greeting {
  val message: String = this.welcome()
  val name: String  = "Jack"
  def welcome()     = "Hello, " + name    // error
}
```

```
-- Error: code/greeting.scala:3:6 ------------------------------
3 |  val name: String  = "Jack"
  |      ^
  |Access non-initialized field name. Calling trace:
  | -> val message: String = this.welcome()   [ greeting.scala:2 ]
  |  -> def welcome()     = "Hello, " + name  [ greeting.scala:4 ]
```

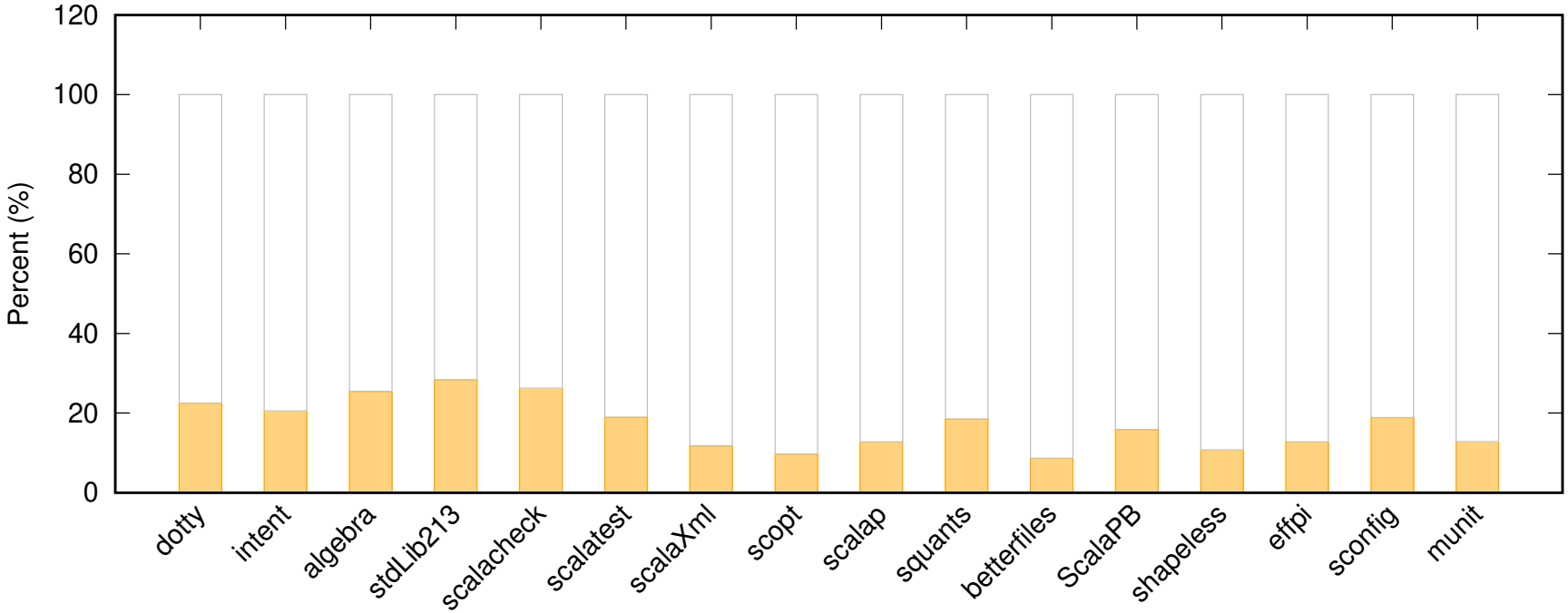
Experiment

We run the checker for 0.6 million real-world Scala code.

- It finds bugs in Dotty, ScalaTest and Scala stdlib
- It reports 0.6 warnings/KLOC



Performance



The percentage of time for initialization check relative to the whole compilation time.

Conclusion

- modular understanding of local reasoning
- monotonicity, stackability, scopability, authority
- abstractions of cold, warm, hot
- type-and-effect inference system with potentials
- implementation in Scala 3 compiler

Problem
Challenges
Reasoning
Evaluation
Inference
Model

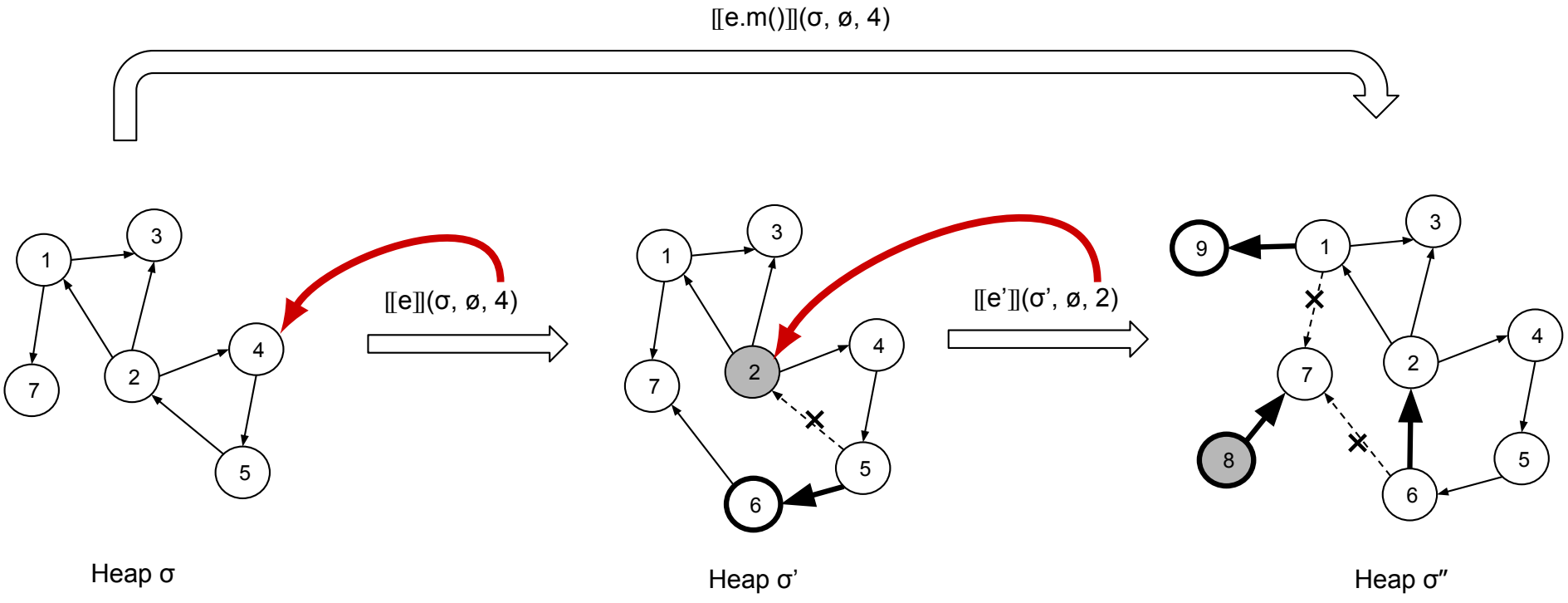
Proof of scopability (1)

Definition (Scoping Preservation)

$$\sigma_1 \rightsquigarrow \sigma_2 \lessdot L \quad \triangleq \quad \forall \sigma_0, L_0, L_1.$$
$$(\sigma_0, L_0) \lessdot (\sigma_1, L) \bigwedge (\sigma_0, L_0) \lessdot (\sigma_1, L_1)$$
$$\Longrightarrow (\sigma_0, L_0) \lessdot (\sigma_2, L_1)$$

Without scoping preservation, in an evaluation $[\![e]\!](\sigma, \rho, \psi) = (l, \sigma')$, we cannot even conclude that $(\sigma, L) \lessdot (\sigma', L)$, where $L = \mathit{codom}(\rho) \cup \{ \psi \}$.

Proof of scopability (2)



- $(\sigma, 1) \lessdot (\sigma', 6)$ is not preserved in $\sigma' \rightsquigarrow \sigma''$
- $(\sigma, 4) \lessdot (\sigma', 6)$ is preserved in $\sigma' \rightsquigarrow \sigma''$

Proof of scopability (3)

Theorem (Scopability)

If $[\![e]\!](\sigma, \rho, \psi) = (l, \sigma')$, *then we have*

- $(\sigma, \mathit{codom}(\rho) \cup \{\ \psi\ \}) \lessdot (\sigma', l)$
- $\sigma \rightsquigarrow \sigma' \lessdot \mathit{codom}(\rho) \cup \{\ \psi\ \}$

Local reasoning lemma

Lemma (Local Reasoning)

$$\frac{(\sigma, L) \lessdot (\sigma', L') \qquad \sigma \ll \sigma' \qquad \sigma \preceq \sigma' \qquad \sigma \vDash L : \mathit{hot}}{\sigma' \vDash L' : \mathit{hot}}$$

Proof.

Let's consider a reachable object l from L', i.e. $\sigma' \vDash L' \rightsquigarrow l$. Depending on whether $l \in \mathit{dom}(\sigma)$, there are two cases.

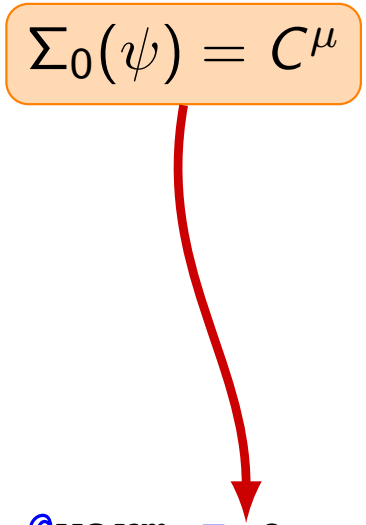
- **Case** $l \notin \mathit{dom}(\sigma)$.
  Use the fact that $\sigma \ll \sigma'$, we know $\sigma' \vDash l : \mathit{warm}$.
- **Case** $l \in \mathit{dom}(\sigma)$.
  Use the fact that $(\sigma, L) \lessdot (\sigma', L')$, we have $\sigma \vDash L \rightsquigarrow l$.
  From the premise $\sigma \vDash L : \mathit{hot}$, we have $\sigma \vDash l : \mathit{warm}$.
  From $\sigma \preceq \sigma'$, we have $\sigma' \vDash l : \mathit{warm}$.

In both cases, we have $\sigma' \vDash l : \mathit{warm}$, by definition we have $\sigma' \vDash L' : \mathit{hot}$. □

Meta-theory

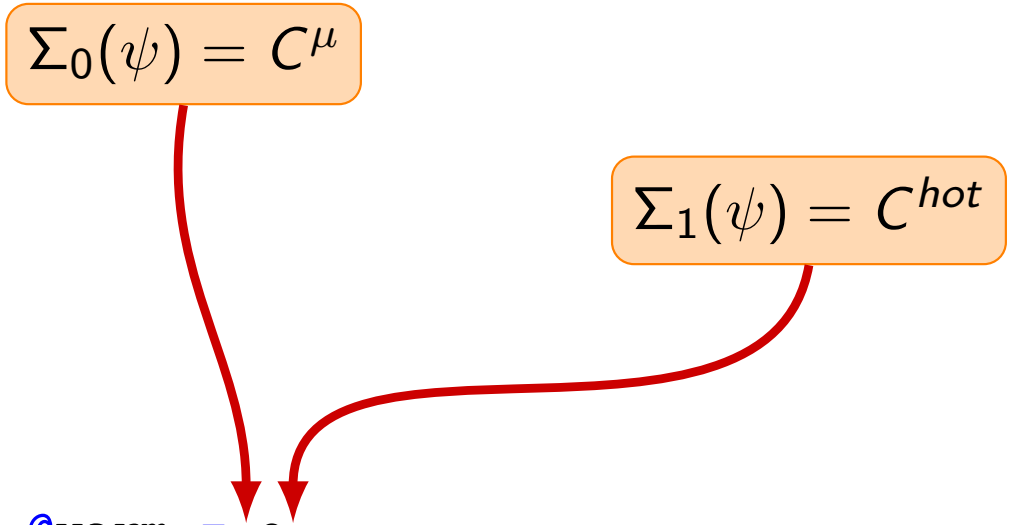
```
1  class C {
2    // ...
3    var x: D @warm = e
4    ------------------
5    var y: Int = 10
6  }
```

Meta-theory



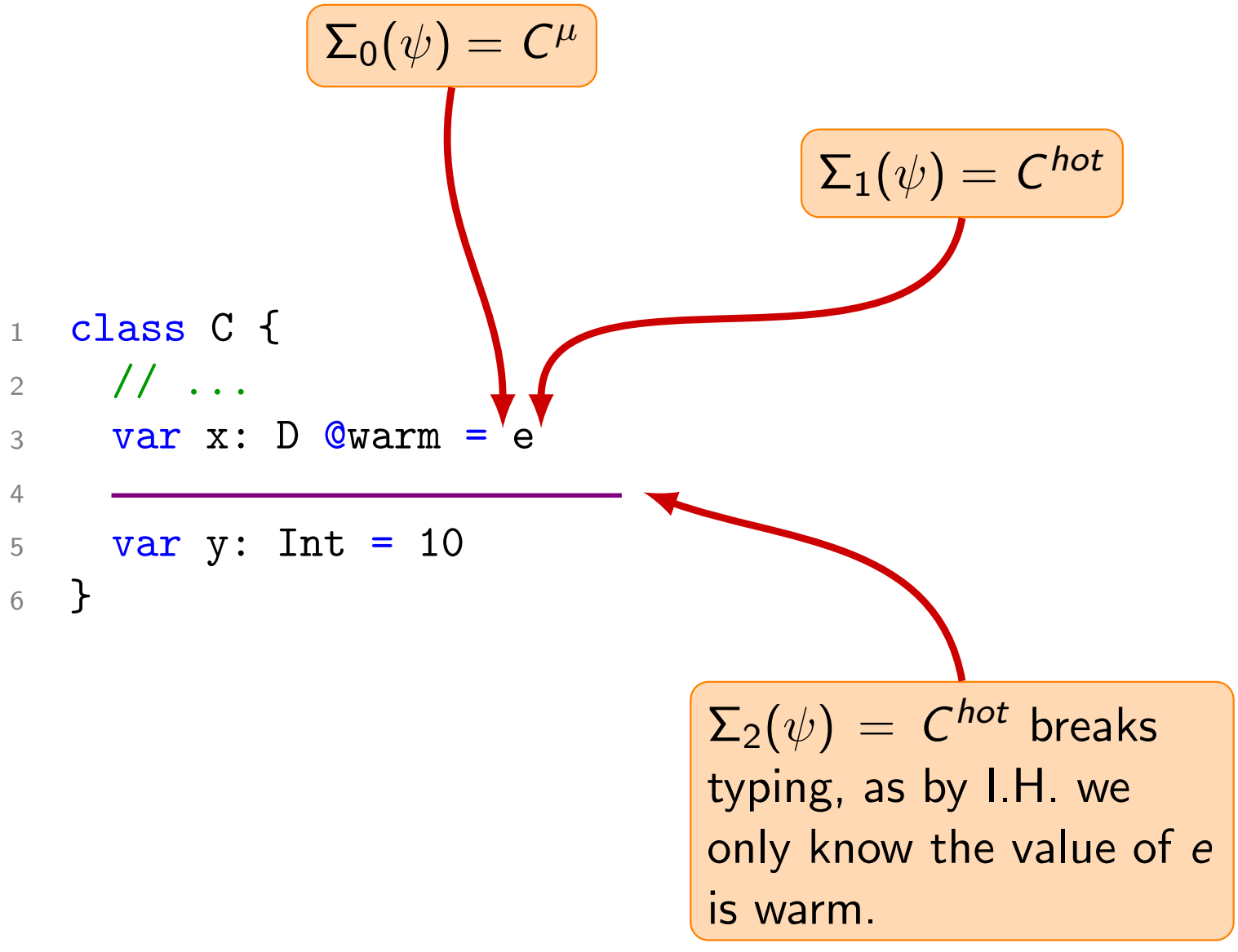

```
1  class C {
2    // ...
3    var x: D @warm = e
4    ------------------
5    var y: Int = 10
6  }
```

Meta-theory




```
1  class C {
2    // ...
3    var x: D @warm = e
4    ------------------
5    var y: Int = 10
6  }
```

Meta-theory


$\Sigma_0(\psi) = C^{\mu}$
$\Sigma_1(\psi) = C^{hot}$
1 class C {
2 // ...
3 var x: D @warm = e
4
5 var y: Int = 10
6 }
$\Sigma_2(\psi) = C^{hot}$ breaks typing, as by I.H. we only know the value of e is warm.

Challenging examples (1)

```
1  class RecType(parentExp: RecType => Type) {
2    val parent = parentExp(this)
3  }
```

Challenging examples (2)

```
1  object Foo {
2    case class Student(name: String, age: Int)
3    call(Student("Jack", 30) // currently a warning
4  }
```

Challenging examples (3)

```
1    def foo(x: => Int) = new A(x)
2    class A(init: => Int)
3    class Foo {
4      val a: A = foo(b) // warning!
5      val b: Int = 100
6    }
```

Fragile base class

```
1  class Base {
2    def g(): String = "hello"
3  }
4
5  class Foo extends Base {
6    val a = this.g()
7  }
8
9  class Bar extends Base {
10   val b: String = "b"
11   override def g(): String = this.b
12 }
```

Strong and weak updates

Suppose that Σ maps addresses to a lattice L:

$$\Sigma : Loc \rightharpoonup L$$

Weak update

$$\Sigma(l) \sqsubseteq \Sigma'(l)$$

Strong update

$$\Sigma(l) \not\sqsubseteq \Sigma'(l)$$

Scala Puzzle

What does the following Scala code prints at runtime?

```
1  class A(n: Int) { println(n) }
2
3  class B(val m: Int) extends A(m) {
4    println(m)
5  }
6
7  new B(10) { override val m = 10 }
```

Scala Puzzle

What does the following Scala code prints at runtime?

```
1  class A(n: Int) { println(n) }
2
3  class B(val m: Int) extends A(m) {
4    println(m)
5  }
6
7  new B(10) { override val m = 10 }
```

Answer: 10, 0

Swift

The Swift compiler rejects the following code:

```
1  class Position {
2      var x, y: Int
3      var f: () -> Int
4      init() {
5          x = 4
6          y = x * x                          // OK
7          f = { () -> Int in self.y } // error
8      }
9  }
```

Overly restrictive and inconsistent!