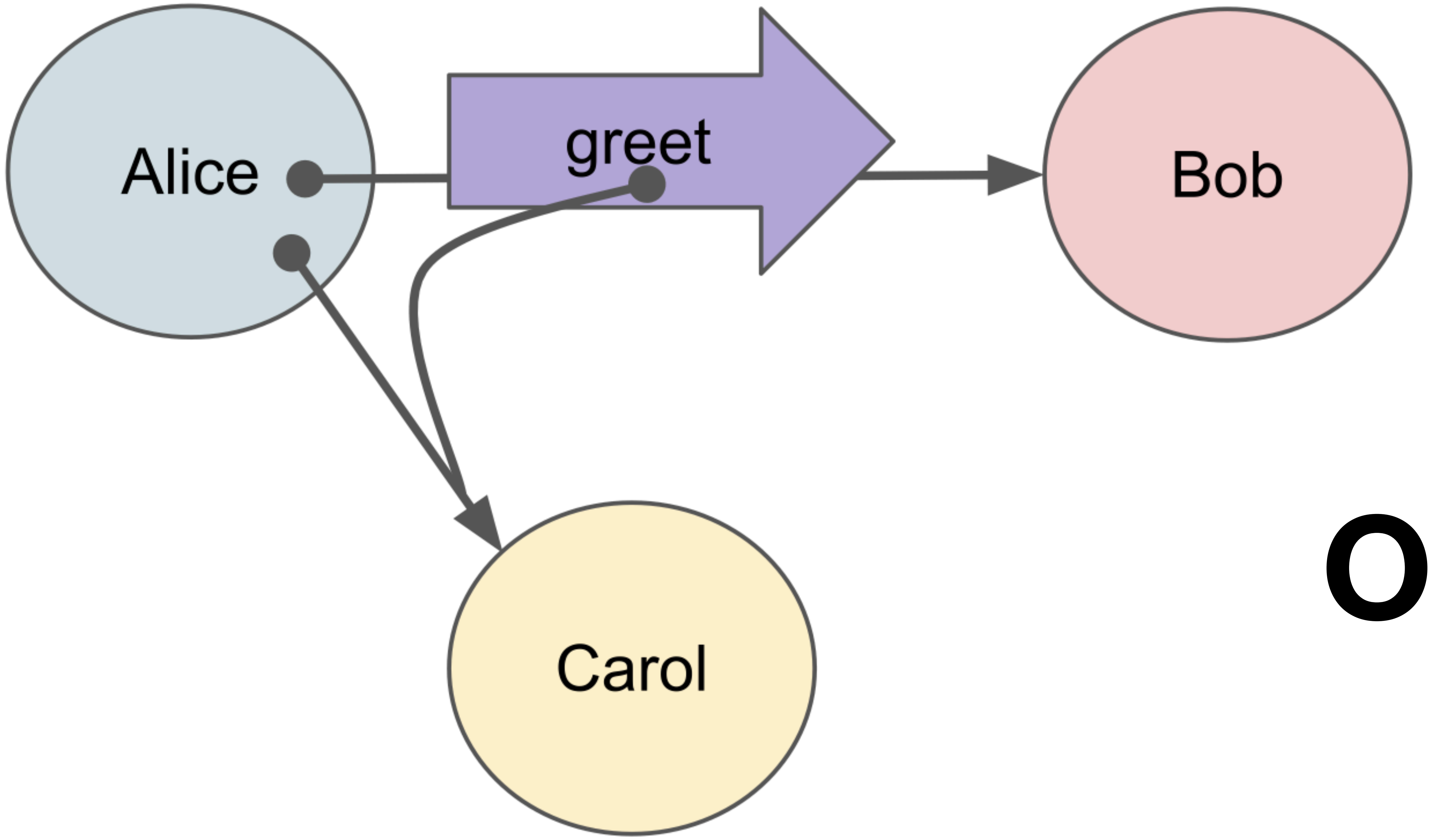


Reasoning with Object Capabilities

POCL 2024

James Noble

Susan Eisenbach

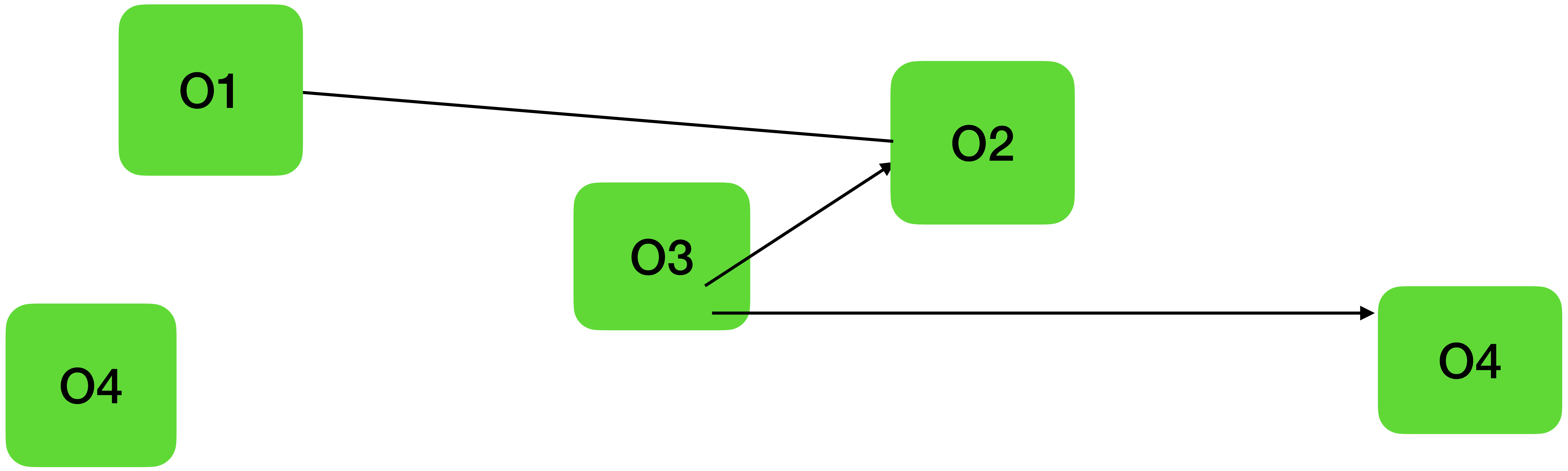
Julian Mackay

and in earlier works

Mark Miller

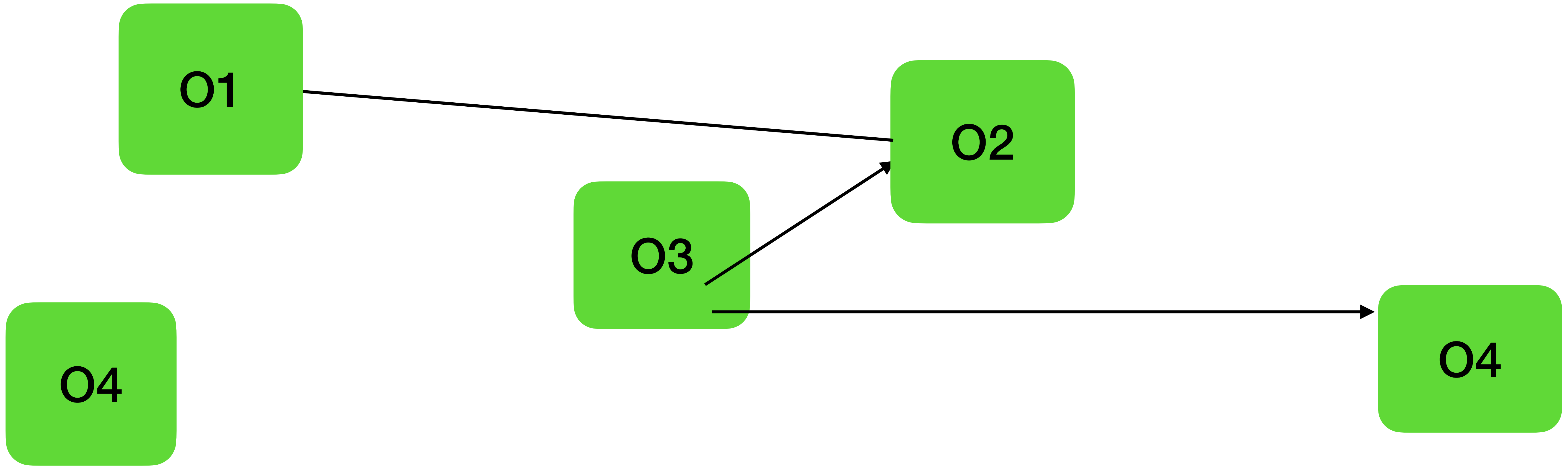
Toby Murray

Our research Question



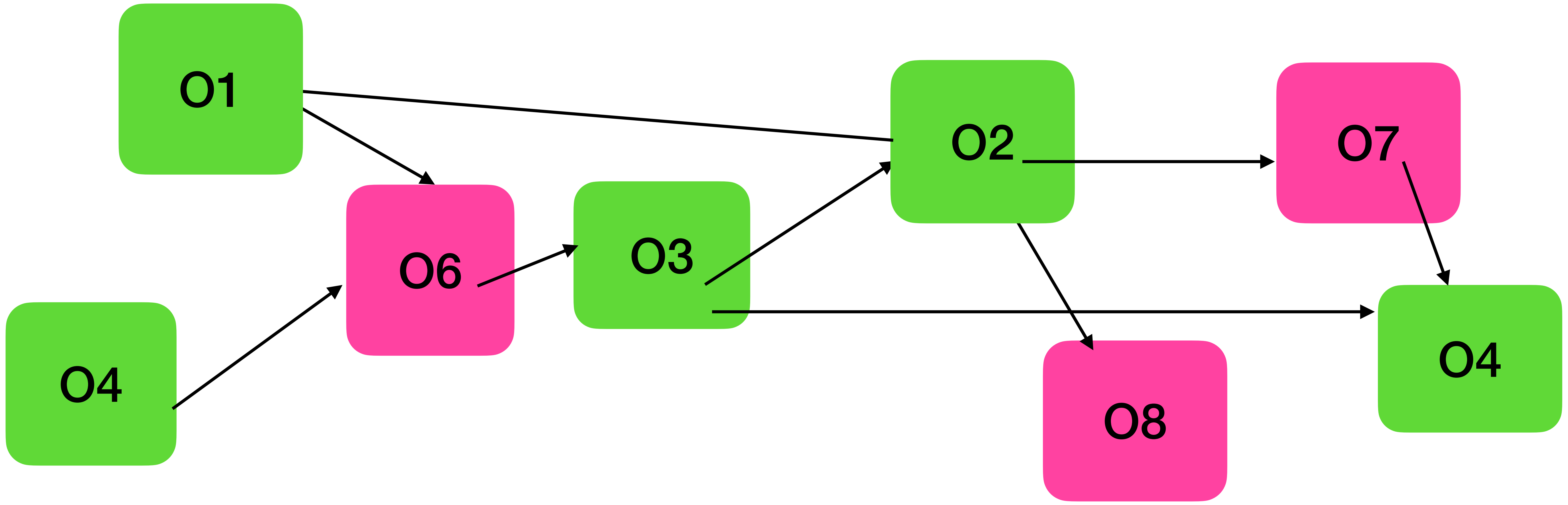
Our research Question

Reason about how our, internal, trusted objects



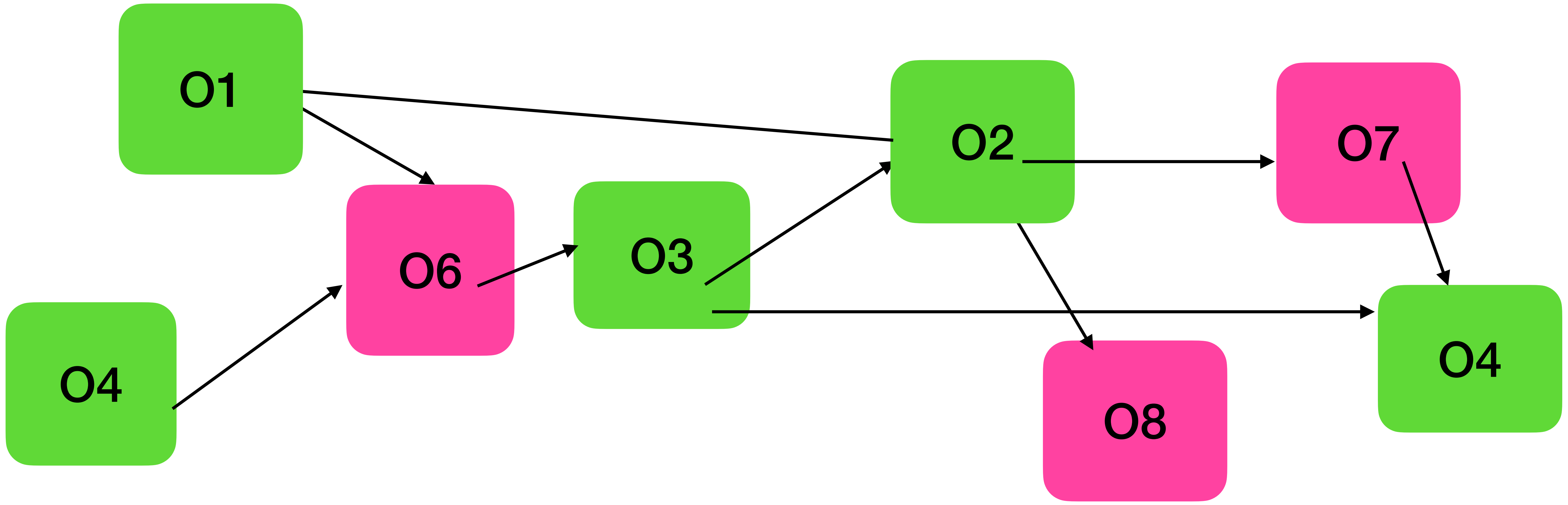
Our research Question

Reason about how our, internal, trusted objects can interact securely with the
external, untrusted, potentially malicious, objects.



Our research Question

Reason about how our, internal, trusted objects can interact securely with the external, untrusted, potentially malicious, objects. using object capabilities.



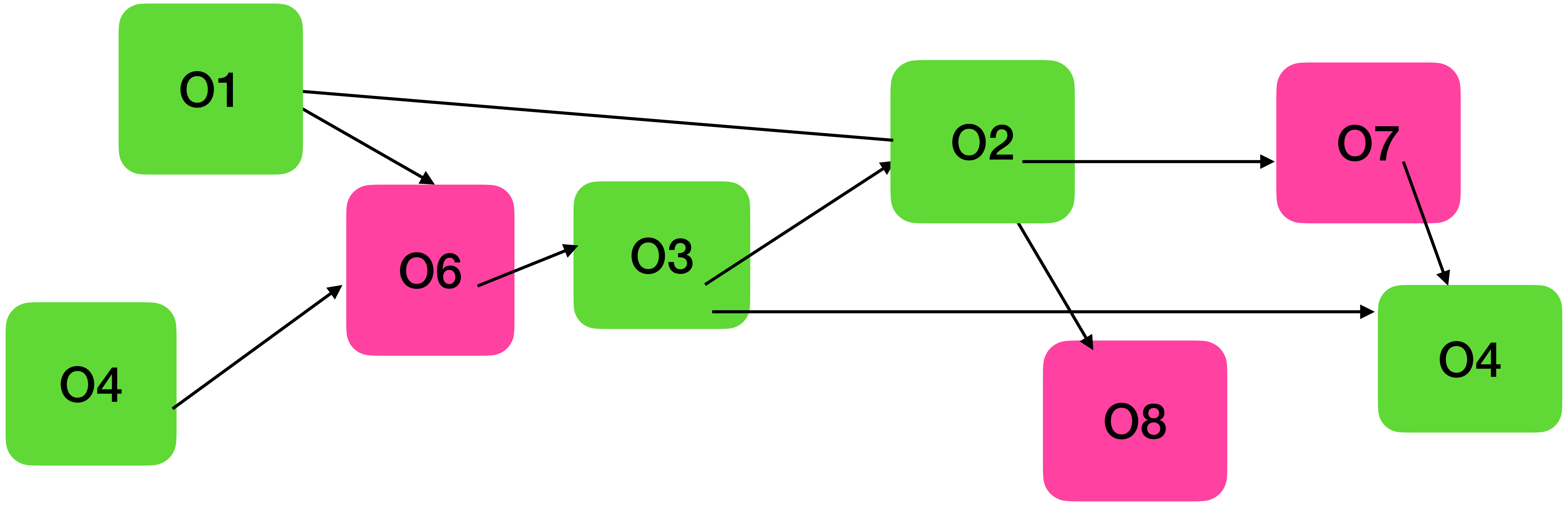
Our research Question

Reason about how our, internal, trusted objects can interact securely with the
external, untrusted, potentially malicious, objects. using object capabilities.

Literature:

*"A **capability** describes a transferable right to perform one (or more) operations."*

*"References cannot be forged.
Capability transferred only through messages or through creation."*



Literature:

“A capability describes a transferable right to perform one (or more) operations.”

“References cannot be forged. Capability transferred only through messages or through creation.”

Literature:

“A capability describes a transferable right to perform one (or more) operations.”

“References cannot be forged. Capability transferred only through messages or through creation.”

Our Insights:

Capabilities are necessary conditions for *effects* (not operations)

Tracking access to capabilities is paramount.

Example
and
Four Challenges

Three Modules

.. assuming all methods are public, and fields are private

```
module Mod$_{good}$
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) -> void
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

Three Modules

.. assuming all methods are public, and fields are private

```
module Mod_good
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) -> void
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

```
module Mod_bad
  class Account
    field balance:int
    field pwd: Password
    method transfer(..) ...
      ... as earlier ...
    method init(...) ...
       ... as earlier ...
    method set(pwd': Password)
      this.pwd=pwd'

  class Password
```

Three Modules

.. assuming all methods are public, and fields are private

```
module Mod$_{good}$
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) -> void
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

```
module Mod$_{bad}$
  class Account
    field balance:int
    field pwd: Password
    method transfer(..) ...
      ... as earlier ...
    method init(...) ...
       ... as earlier ...
    method set(pwd': Password)
      this.pwd=pwd'

  class Password
```

Three Modules

.. assuming all methods are public, and fields are private

```
module Mod$_{good}$
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) -> void
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

```
module Mod$_{bad}$
  class Account
    field balance:int
    field pwd: Password
    method transfer(..) ...
      ... as earlier ...
    method init(...) ...
       ... as earlier ...
    method set(pwd': Password)
      this.pwd=pwd'

  class Password
```

```
module Mod$_{better}$
  class Account
    field balance:int
    field pwd: Password
    method transfer(..)
      ... as earlier ...


    method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
  class Password
```

Three Modules

.. assuming all methods are public, and fields are private

```
module Mod_good
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) -> void
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

```
module Mod_bad
  class Account
    field balance:int
    field pwd: Password
    method transfer(..) ...
      ... as earlier ...
    method init(...) ...
       ... as earlier ...
    method set(pwd': Password)
      this.pwd=pwd'

  class Password
```

```
module Mod_better
  class Account
    field balance:int
    field pwd: Password
    method transfer(..)
      ... as earlier ...


    method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
  class Password
```

```
module Mod_good
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) ->
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

Challenge_1: A module spec S, such that

$$M_{good} \models S$$
$$M_{bad} \not\models S$$
$$M_{better} \models S$$

```
module Mod_bad
  class Account
    field balance:int
    field pwd: Password
    method transfer(..) ...
      ... as earlier ...
    method init(...) ...
       ... as earlier ...
    method set(pwd': Password)
      this.pwd=pwd'

  class Password
```

```
module Mod_better
  class Account
    field balance:int
    field pwd: Password
    method transfer(..)
      ... as earlier ...


    method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
  class Password
```

```
module Modgood
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) ->
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
    method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

Challenge_2: An inference system, such that

$$M_{good} \vdash S$$
$$M_{bad} \nvdash S$$
$$M_{better} \vdash S$$

```
module Modbad
  class Account
    field balance:int
    field pwd: Password
    method transfer(..) ...
      ... as earlier ...
    method init(...) ...
       ... as earlier ...
    method set(pwd': Password)
      this.pwd=pwd'

  class Password
```

```
module Modbetter
  class Account
    field balance:int
    field pwd: Password
    method transfer(..)
      ... as earlier ...

    method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
  class Password
```

```
module Mod$_{good}$
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) ->
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

Challenge_3: Inference system should be algorithmic

```
module Mod$_{bad}$
  class Account
    field balance:int
    field pwd: Password
    method transfer(..) ...
      ... as earlier ...
    method init(...) ...
       ... as earlier ...
    method set(pwd': Password)
      this.pwd=pwd'

  class Password
```

```
module Mod$_{better}$
  class Account
    field balance:int
    field pwd: Password
    method transfer(..)
      ... as earlier ...


    method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
  class Password
class Shop

  fld myAccount : Account
  fld inventory : Inventory

  void buy(buyer: Object, anItem: Item)
    int price = anItem.price
    int oldBalance = this.myAccount.balance
    buyer.payMe(myAccount,price)
    if (this.myAccount.balance == oldBalance+price)
       this.send(buyer,anItem)
    else
       buyer.tell("you have not paid me")
```

External call

```
class Shop

  fld myAccount : Account
  fld inventory : Inventory

  void buy(buyer: Object, anItem: Item)
    int price = anItem.price
    int oldBalance = this.myAccount.balance
    buyer.payMe(myAccount,price)
    if (this.myAccount.balance == oldBalance+price)
       this.send(buyer,anItem)
    else
       buyer.tell("you have not paid me")
```

External call

```
class Shop

  fld myAccount : Account
  fld inventory : Inventory

  void buy(buyer: Object, anItem: Item)
    int price = anItem.price
    int oldBalance = this.myAccount.balance
    buyer.payMe(myAccount,price)
    if (this.myAccount.balance == oldBalance+price)
       this.send(buyer,anItem)
    else
       buyer.tell("you have not paid me")
```

If `Account` comes from a “good” module,
and upon call, `buyer` has no eventual access to `myAccount.password`,

External call

```
class Shop

  fld myAccount : Account
  fld inventory : Inventory

  void buy(buyer: Object, anItem: Item)
    int price = anItem.price
    int oldBalance = this.myAccount.balance
    buyer.payMe(myAccount,price)
    if (this.myAccount.balance == oldBalance+price)
       this.send(buyer,anItem)
    else
       buyer.tell("you have not paid me")
```

If `Account` comes from a “good” module,
and upon call, `buyer` has no eventual access to `myAccount.password`,

then
the balance of `myAccount` does not decrease by the call `payMe`.

External call

```
class Shop

  fld myAccount : Account
  fld inventory : Inventory

  void buy(buyer: Object, anItem: Item)
    int price = anItem.price
    int oldBalance = this.myAccount.balance
    buyer.payMe(myAccount,price)
    if (this.myAccount.balance == oldBalance+price)
       this.send(buyer,anItem)
    else
       buyer.tell("you have not paid me")
```

Challenge_4: An inference system, such that we can prove **external** calls.

If `Account` comes from a “good” module,
and upon call, `buyer` has no eventual access to `myAccount.password`,

then
the balance of `myAccount` does not decrease by the call `payMe`.

Challenge_1: A module spec S, such that

$$M_{good} \models S$$
$$M_{bad} \not\models S$$
$$M_{better} \models S$$

Challenge_1: A module spec S, such that

$$M_{good} \models S$$
$$M_{bad} \not\models S$$
$$M_{better} \models S$$

Remember: A capability represents a transferable right to perform one or more operations on a given object

Challenge_1: A module spec S, such that

$$M_{good} \models S$$
$$M_{bad} \not\models S$$
$$M_{better} \models S$$

Remember: A capability represents a transferable right to perform one or more operations on a given object

So: “The password enables withdrawal from the account”?

Challenge_1: A module spec S, such that

$$M_{good} \models S$$
$$M_{bad} \not\models S$$
$$M_{better} \models S$$

Remember: A capability represents a transferable right
to perform one or more operations on a given object

~~***So***: "The password enables withdrawal from the account"?~~

Or: "Without the password call of withdraw will fail"?

Challenge_1: A module spec S, such that

$$M_{good} \models S$$
$$M_{bad} \not\models S$$
$$M_{better} \models S$$

Remember: A capability represents a transferable right to perform one or more operations on a given object

~~***So***: "The password enables withdrawal from the account"?~~

~~***Or***: "Without the password call of withdraw will fail"?~~

Or: "Without the password no reduction of the balance of the account"?

Challenge_1: A module spec S, such that

$M_{good} \models S$
$M_{bad} \not\models S$
$M_{better} \models S$

Remember: A capability represents a transferable right to perform one or more operations on a given object

~~***So***: "The password enables withdrawal from the account"?~~

~~***Or***: "Without the password call of withdraw will fail"?~~

Or: "Without the password no reduction of the balance of the account"?

So: ∀ s:Statement. {without a.password ∧ a.balance=b} s { a.balance >= b }

Challenge_1: A module spec S, such that

$M_{good} \models S$
$M_{bad} \not\models S$
$M_{better} \models S$

Motto:
Capability is a *necessary* condition for some effect

Remember: A capability represents a transferable right to perform one or more operations on a given object

~~***So***: "The password enables withdrawal from the account"?~~

~~***Or***: "Without the password call of withdraw will fail"?~~

Or: "Without the password no reduction of the balance of the account"?

So: $\forall$ s:Statement. {without a.password $\wedge$ a.balance=b} s { a.balance >= b }

Challenge_1: A module spec S, such that …

So: $\forall$ s:Statement {without a.password $\land$ a.balance=b} s { a.balance >= b }

Challenge_1: A module spec S, such that …

So: ∀ s:Statement {without a.password ∧ a.balance=b} s { a.balance >= b }

So: ∀ a: Account. ∀ n: Num. (| a.password **prt** ∧ a.balance=b |) (| a.balance>=b |)

Challenge_1: A module spec S, such that …

So: ∀ s:Statement {without a.password ∧ a.balance=b} s { a.balance >= b }

So: ∀ a: Account. ∀ n: Num. (| a.password **prt** ∧ a.balance=b |) (| a.balance>=b |)

In general: ∀ x1:C1,x2:C2… (| A |) (| A’ |)

Challenge_1: A module spec S, such that …

So: $\forall$ s:Statement {without a.password $\wedge$ a.balance=b} s { a.balance >= b }

So: $\forall$ a: Account. $\forall$ n: Num. (| a.password **prt** $\wedge$ a.balance=b |) (| a.balance>=b |)

In general: $\forall$ x1:C1,x2:C2… (| A |) (| A' |)

Challenge_1_a : Meaning of x **prt**

Challenge_1: A module spec S, such that …

So: ∀ s:Statement {without a.password ∧ a.balance=b} s { a.balance >= b }

So: ∀ a: Account. ∀ n: Num. (| a.password **prt** ∧ a.balance=b |) (| a.balance>=b |)

In general: ∀ x1:C1,x2:C2… (| A |) (| A’ |)

Challenge_1_a : Meaning of x **prt**

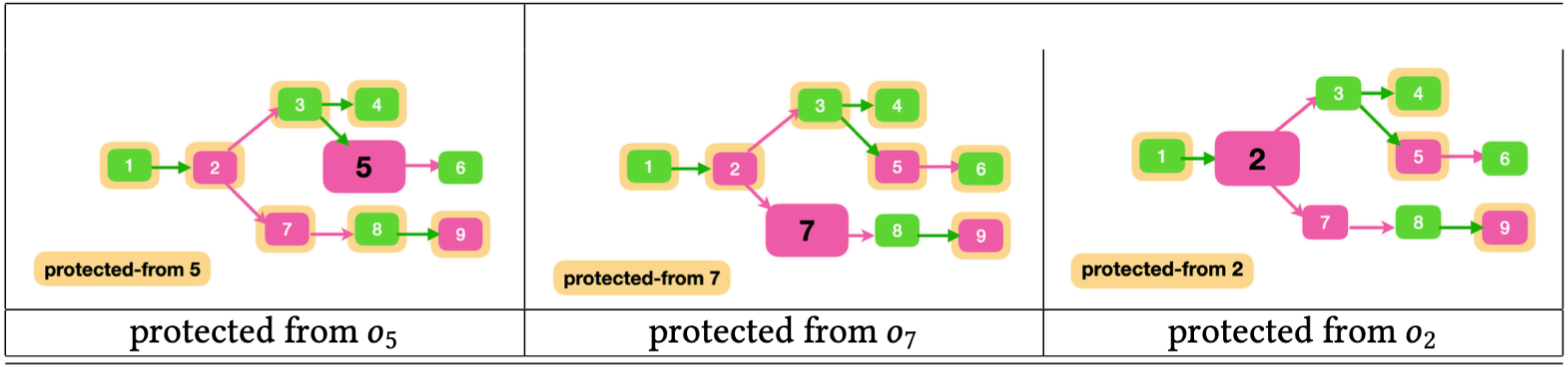
Challenge_1_b : Meaning of ∀ x1:C1,x2:C2… (| A |) (| A’ |)

Challenge_1_a : Meaning of x **prt**

Remember:

…how our, internal, trusted objects will interact securely with the external, untrusted, potentially malicious, objects.

Def: o **prt-frm** o' Iff o=/=o' and the penultimate object on any path from o' to o is internal.



protected from o_5 | protected from o_7 | protected from o_2

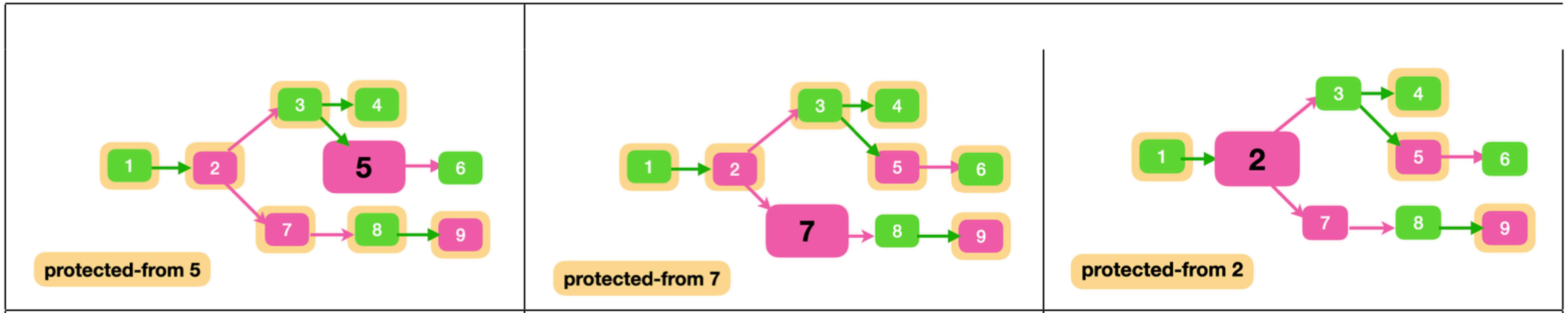
Challenge_1_a : Meaning of x **prt**

Remember:

…how our, internal, trusted objects will interact securely with the external, untrusted, potentially malicious, objects.

Def: **o prt-frm o’** Iff o=/=o’ and the penultimate object on any path from o’ to o is internal.

For example:



protected from o_5 | protected from o_7 | protected from o_2

Challenge_1_a : Meaning of x **prt**

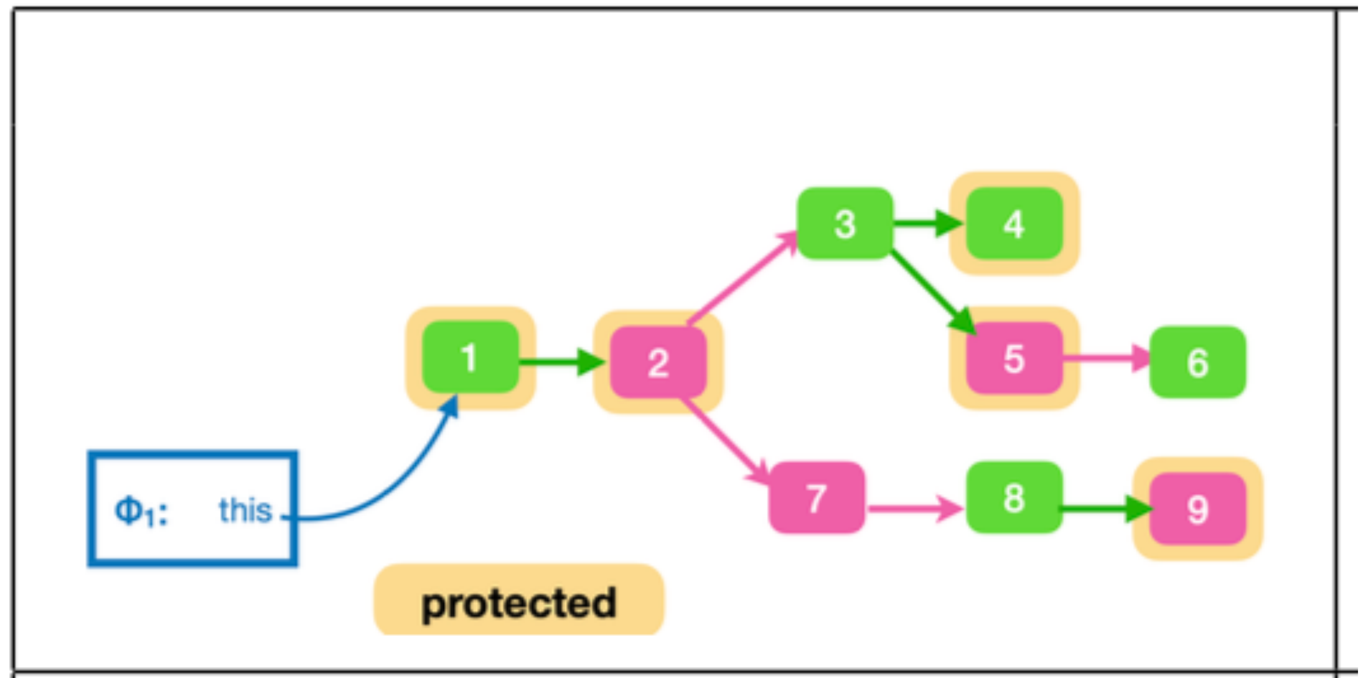
Def: o **prt-frm o’**, if **extr** o’ and any path from o’ to o goes through an internal object.

Def: o **prt** , if o **prt-from** any external object accessible from top frame

Challenge_1_a : Meaning of x **prt**

Def: o **prt-frm o'**, if **extr** o' and any path from o' to o goes through an internal object.

Def: o **prt** , if o **prt-from** any external object accessible from top frame

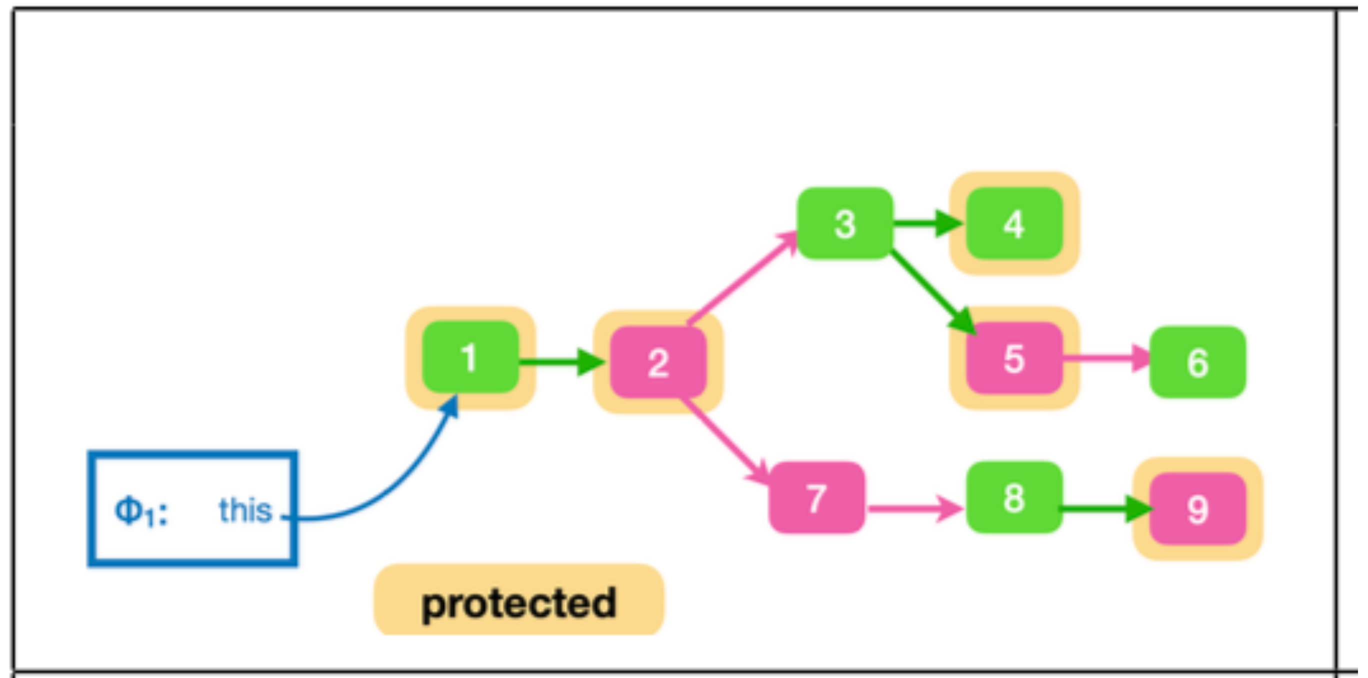


protected with top frame ϕ_1

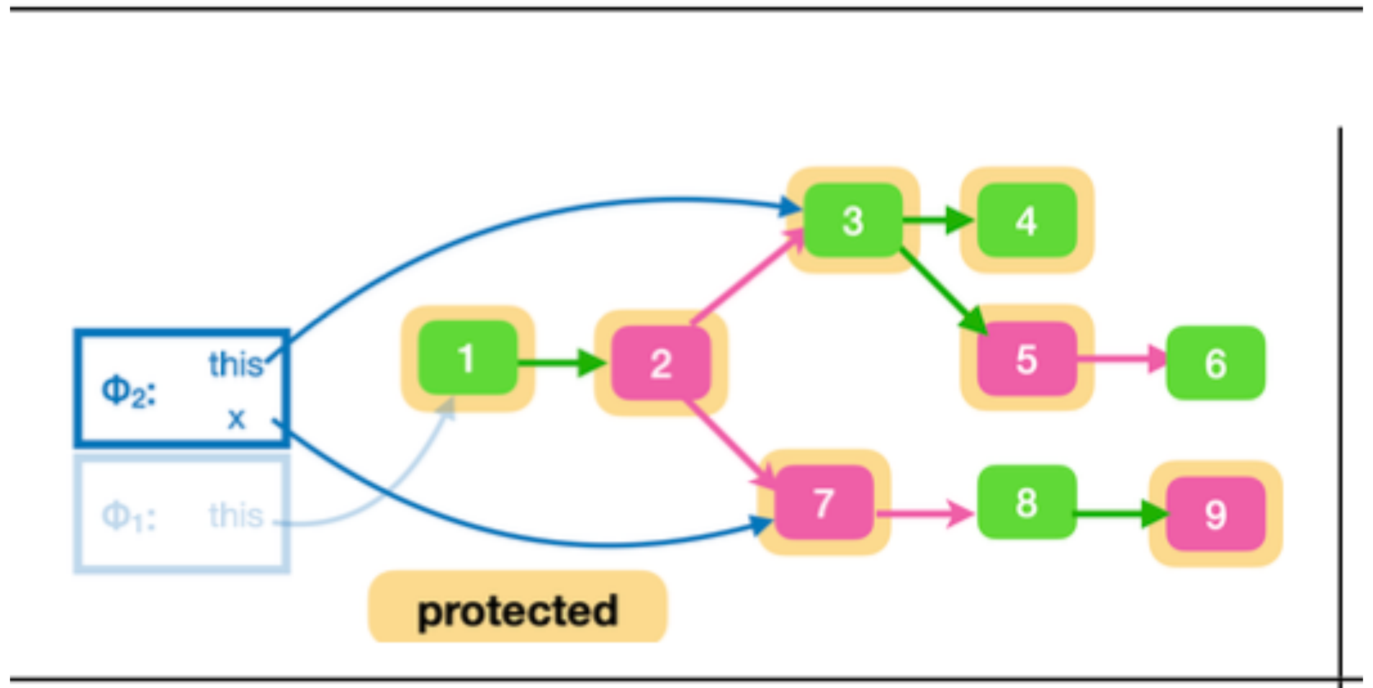
Challenge_1_a : Meaning of x **prt**

Def: o **prt-frm o'**, if **extr** o' and any path from o' to o goes through an internal object.

Def: o **prt** , if o **prt-from** any external object accessible from top frame



protected with top frame ϕ_1

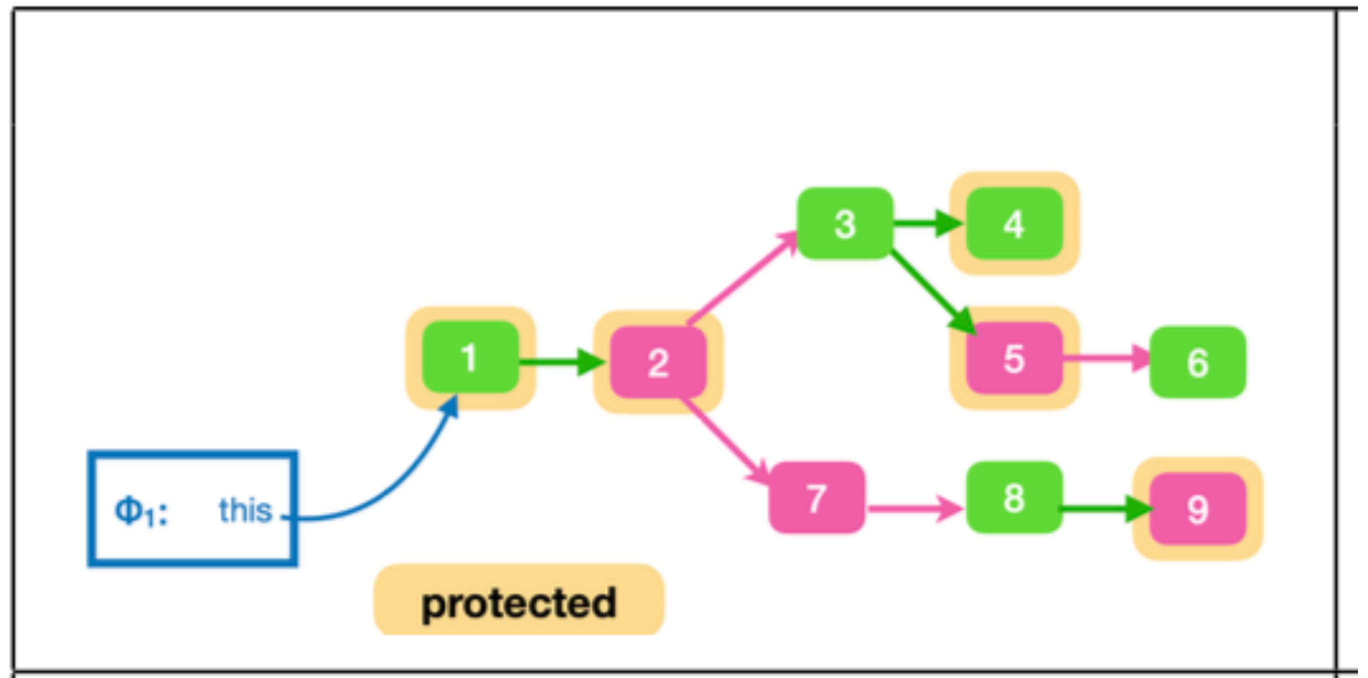


protected with top frame ϕ_2

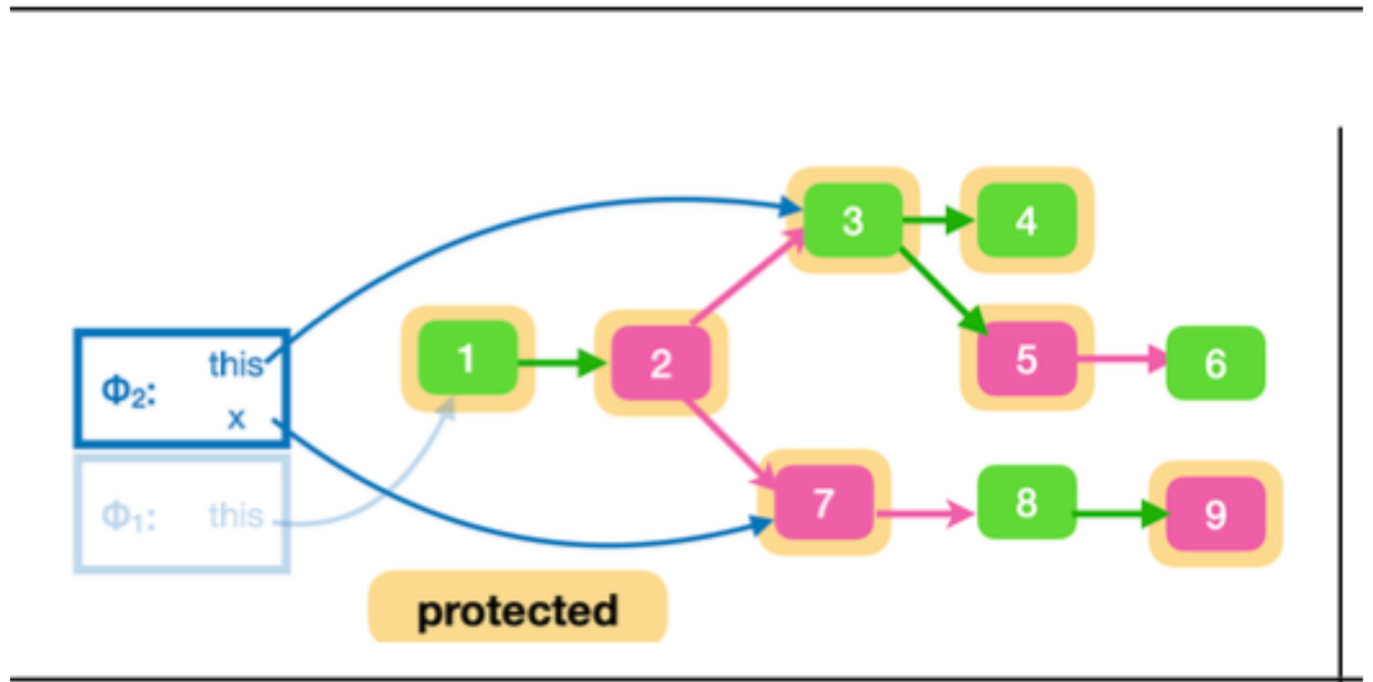
Challenge_1_a : Meaning of x **prt**

Def: o **prt-frm o'**, if **extr** o' and any path from o' to o goes through an internal object.

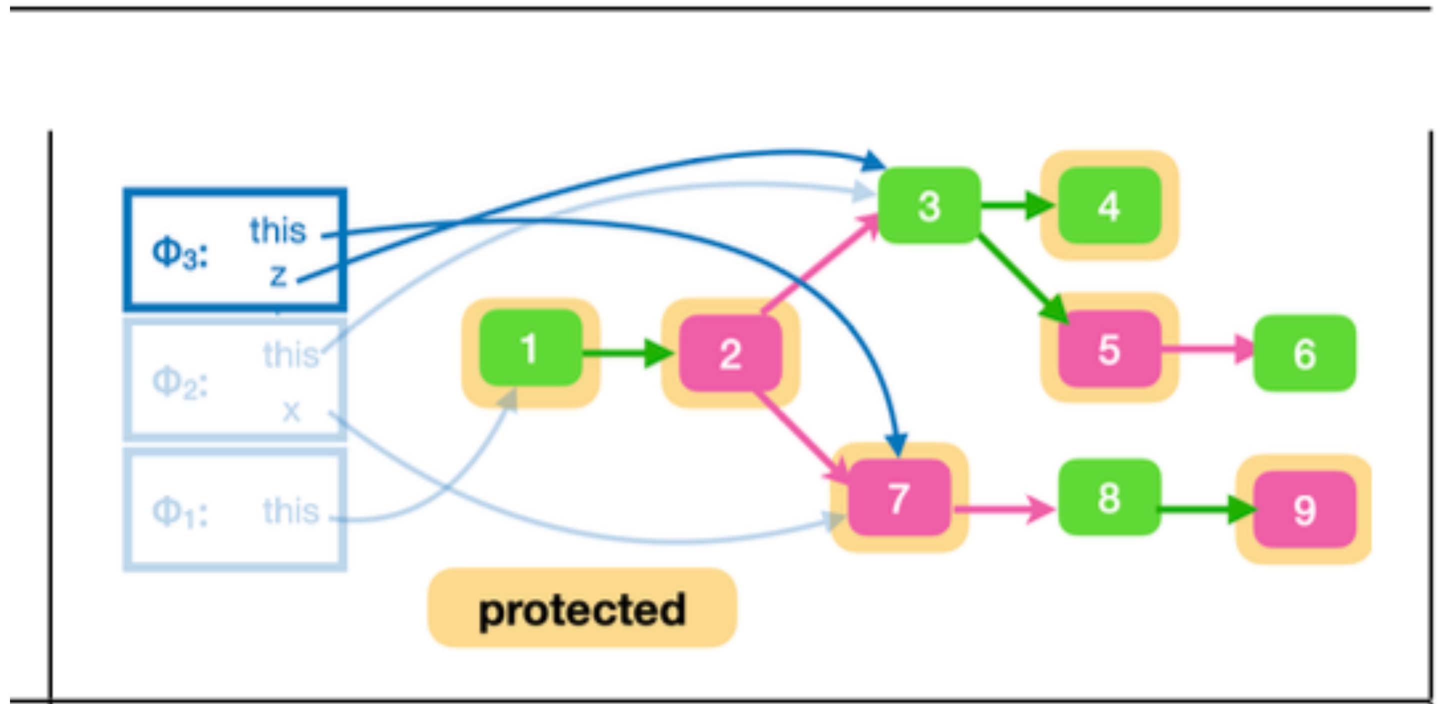
Def: o **prt** , if o **prt-from** any external object accessible from top frame



protected with top frame ϕ_1



protected with top frame ϕ_2



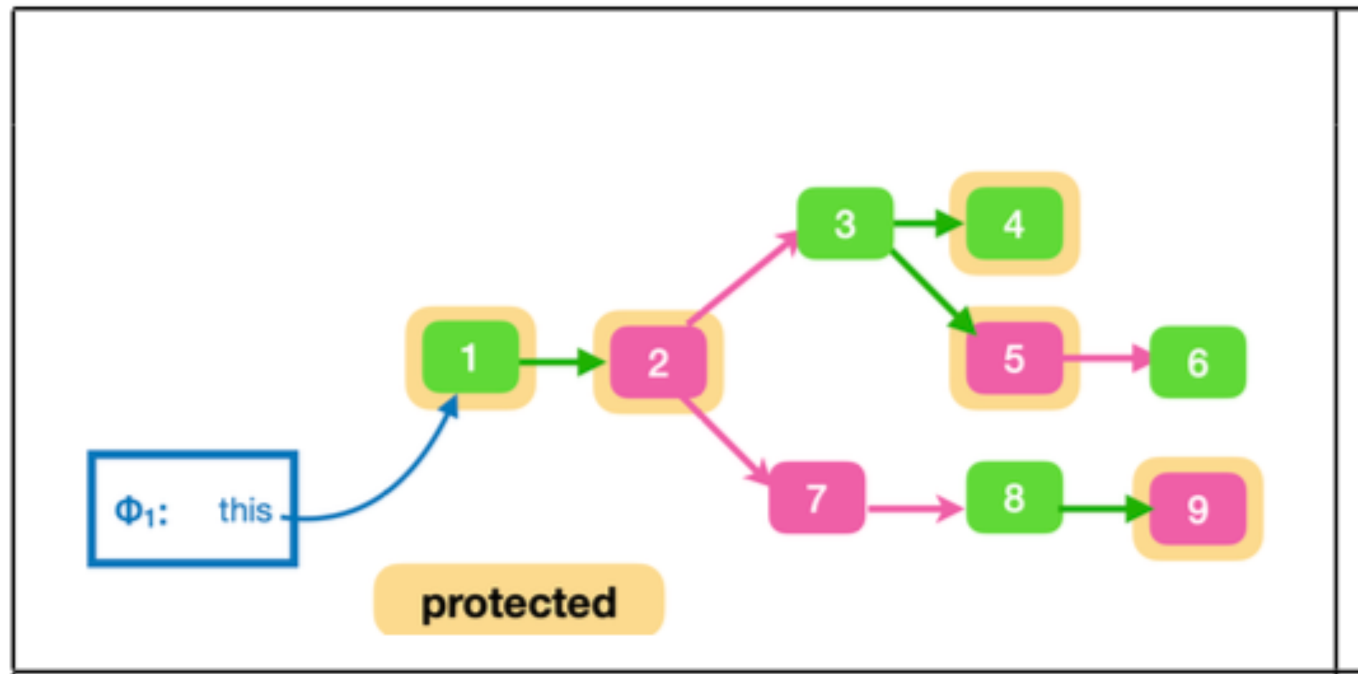
protected with top frame ϕ_3

Challenge_1_a : Meaning of x **prt**

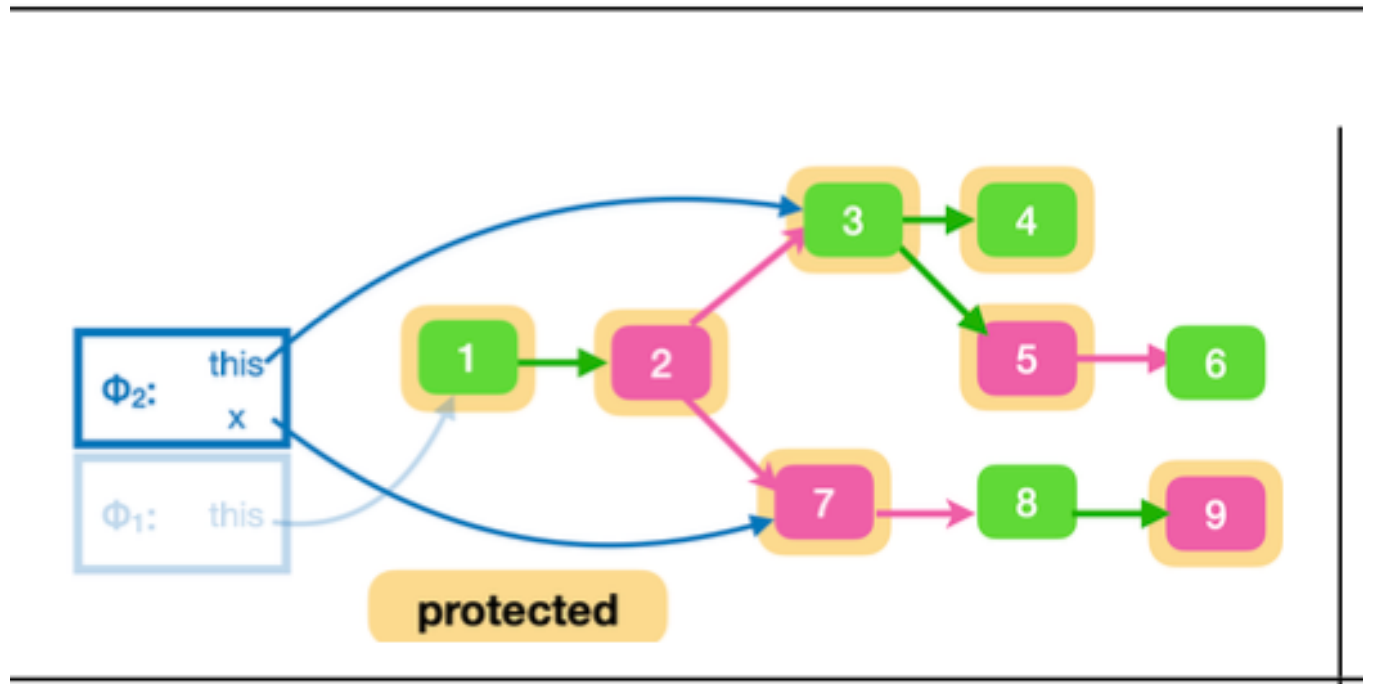
Def: o **prt-frm o'**, if **extr** o' and any path from o' to o goes through an internal object.

Def: o **prt** , if o **prt-from** any external object

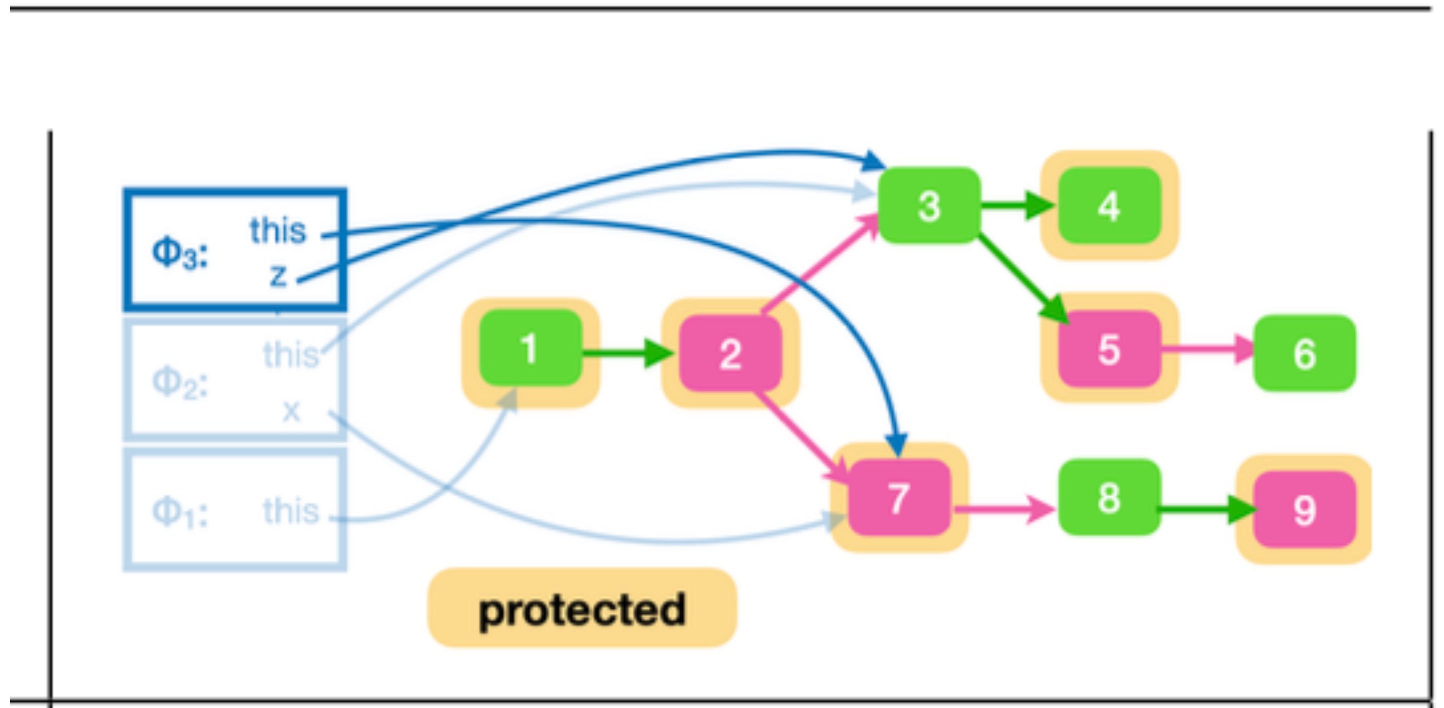
Motto:
Tracking access to Capabilities



protected with top frame ϕ_1



protected with top frame ϕ_2



protected with top frame ϕ_3

Challenge_1_b : Meaning of $\forall x1{:}C1,x2{:}C2\ldots (| A |) (| A' |)$

Definition

$$M \models \forall x1{:}C1,x2{:}C2\ldots (| A |) (| A' |)$$

iff

$\wedge$

Challenge_1_b : Meaning of $\forall$ x1:C1,x2:C2… (| A |) (| A’ |)

Definition

M ⊨ ∀ x1:C1,x2:C2… (| A |) (| A’ |)

iff

For all modules M’,
For all states σ arising from execution of (M, M’) ∧
For all objects o1, .. on globally accessible at σ of class C1, … Cn,
For all states σ’ in the future from σ whithout returning from σ ’s top frame,

Challenge_1_b : Meaning of $\forall$ x1:C1,x2:C2… (| A |) (| A’ |)

Definition

$M \models \forall\ x1{:}C1, x2{:}C2 \ldots\ (\!|\ A\ |\!)\ (\!|\ A'\ |\!)$

iff

For all modules M’,
For all states σ arising from execution of (M, M’) ∧
For all objects o1, .. on globally accessible at σ of class C1, … Cn,
For all states σ’ in the future from σ whithout returning from σ ’s top frame,

If

$M, \sigma \models o1{:}C1 \land \ \ldots\ \land\ on{:}Cn \land A[\ o1/x1, \ldots\ on/xn\ ]$

then

$M, \sigma' \models A'[\ o1/x1, \ldots\ on/xn\ ]$

Putting it together

Putting it together

Therefore,

$\forall$ x: Object (| **prt** x $\wedge$ A(x) |) (| A(x) |)

guarantees that

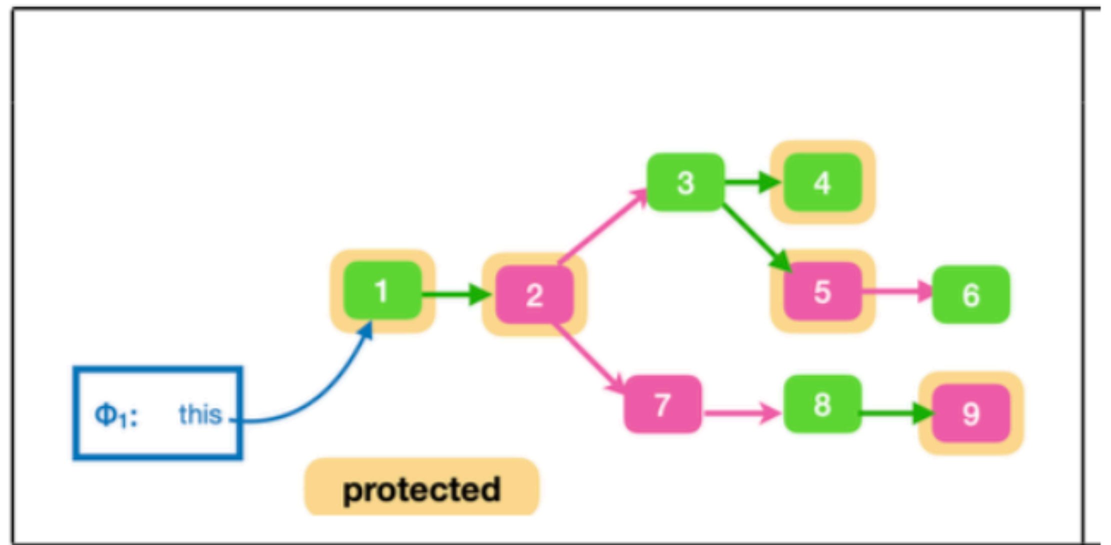
Putting it together

Therefore,

∀ x: Object (| **prt** x ∧ A(x) |) (| A(x) |)

guarantees that

if we start below,



protected with top frame ϕ_1

then

A(o1), A(o2), A(o4),
A(o5), A(o9)

will be preserved

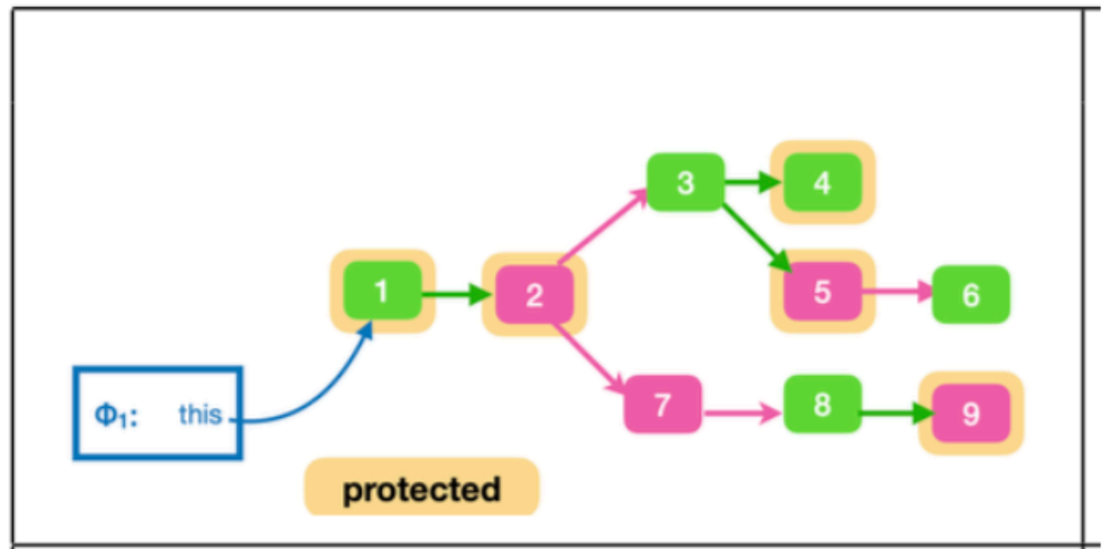
Putting it together

Therefore,

∀ x: Object (| **prt** x ∧ A(x) |) (| A(x) |)

guarantees that

if we start below,



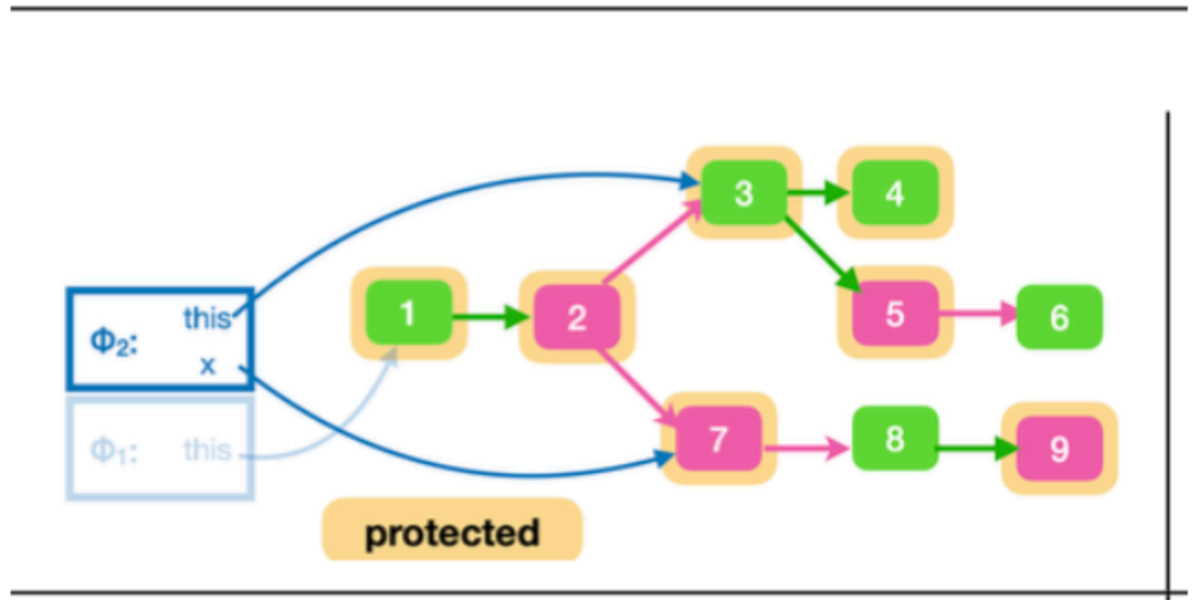
protected with top frame ϕ_1

then

A(o1), A(o2), A(o4),
A(o5), A(o9)

will be preserved

And if we start below,



protected with top frame ϕ_2

then

A(o1), A(o2), A(o3), A(o4),
A(o5), A(o7), A(o9)

will be preserved

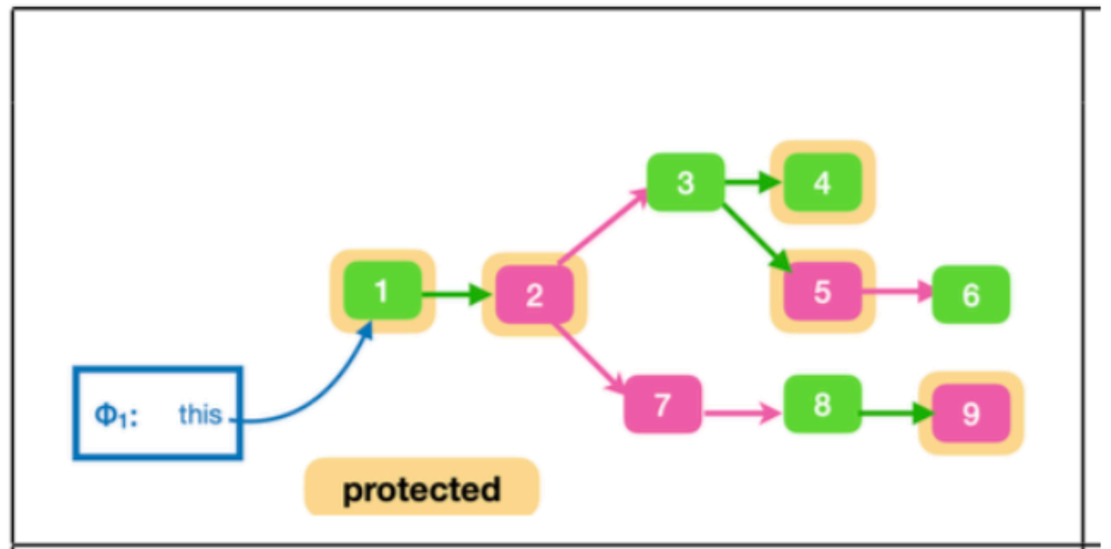
Putting it together

Therefore,

∀ x: Object (| **prt** x ∧ A(x) |) (| A(x) |)

guarantees that

if we start below,



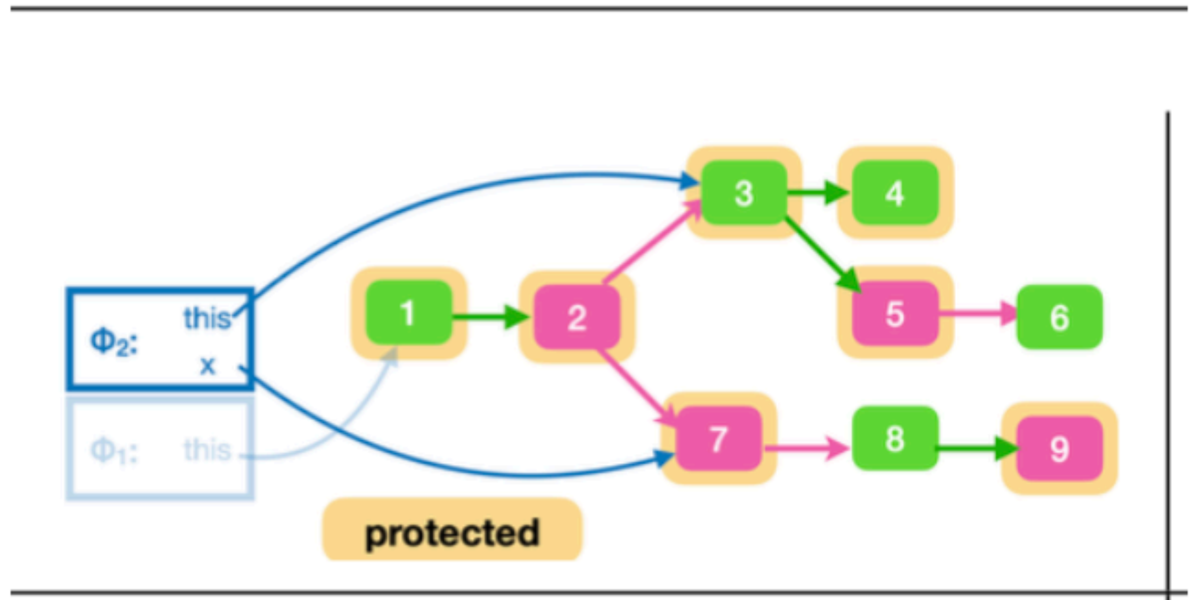
protected with top frame ϕ_1

then

A(o1), A(o2), A(o4),
A(o5), A(o9)

will be preserved

And if we start below,



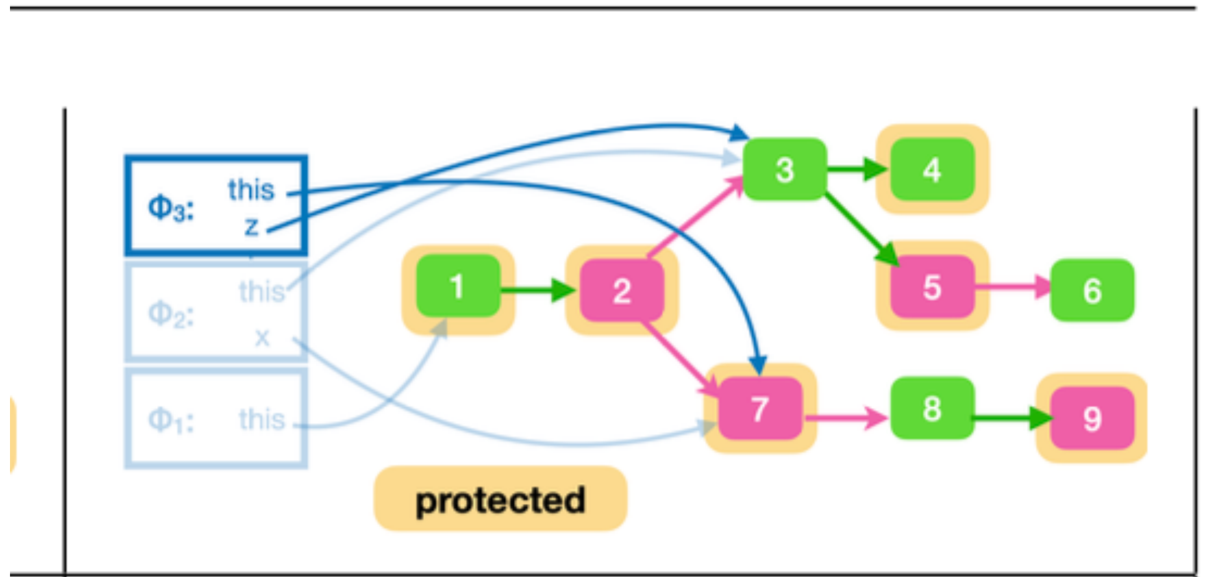
protected with top frame ϕ_2

then

A(o1), A(o2), A(o3), A(o4),
A(o5), A(o7), A(o9)

will be preserved

And if we start below,



protected with top frame ϕ_3

then

A(o1), A(o2), A(o4),
A(o5), A(o7), A(o9)

will be preserved

Challenge_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{better} \models S$

S = ∀ a:Account (| **prt** a |) (| **prt** a |)

∧

∀ a:Account (| **prt** a.password |) (| **prt** a.password |)

∧

∀ a:Account. ∀ b:Num (| **prt** a ∧ a.balance = b |) (| a.balance = b |)

∧

∀ a:Account. ∀ b:Num (| **prt** a.password ∧ a.balance = b |) (| a.balance ≥ b |)

Challenge_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{better} \models S$

API - agnostic:
a.balance, a.password can be ghost

$$
\begin{aligned}
S = \;& \forall\, a{:}Account\ (|\ \mathbf{prt}\ a\ |)\ (|\ \mathbf{prt}\ a\ |) \\
\wedge\;& \\
& \forall\, a{:}Account\ (|\ \mathbf{prt}\ a.password\ |)\ (|\ \mathbf{prt}\ a.password\ |) \\
\wedge\;& \\
& \forall\, a{:}Account.\ \forall\, b{:}Num\ (|\ \mathbf{prt}\ a \wedge a.balance = b\ |)\ (|\ a.balance = b\ |) \\
\wedge\;& \\
& \forall\, a{:}Account.\ \forall\, b{:}Num\ (|\ \mathbf{prt}\ a.password \wedge a.balance = b\ |)\ (|\ a.balance \geq b\ |)
\end{aligned}
$$

Talks about effects

Talks about emergent behaviour

Challenge_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{better} \models S$

$M_{better} \models \forall\, a{:}\mathrm{Account}\ (\!|\ \mathbf{prt}\ a\ |\!)\ (\!|\ \mathbf{prt}\ a\ |\!)$

$M_{better} \models \forall\, a{:}\mathrm{Account}\ (\!|\ \mathbf{prt}\ a.\mathrm{password}\ |\!)\ (\!|\ \mathbf{prt}\ a.\mathrm{password}\ |\!)$

$M_{better} \models \forall\, a{:}\mathrm{Account}.\ \forall\, b{:}\mathrm{Num}\ (\!|\ \mathbf{prt}\ a \wedge a.\mathrm{balance} = b\ |\!)\ (\!|\ a.\mathrm{balance} = b\ |\!)$

$M_{better} \models \forall\, a{:}\mathrm{Account}.\ \forall\, b{:}\mathrm{Num}\ (\!|\ \mathbf{prt}\ a.\mathrm{password} \wedge a.\mathrm{balance} = b\ |\!)\ (\!|\ a.\mathrm{balance} \geq b\ |\!)$

Challenge_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{better} \models S$

$M_{better} \models \forall a{:}Account\ (| \mathbf{prt}\ a |)\ (| \mathbf{prt}\ a |)$

$M_{better} \models \forall a{:}Account\ (| \mathbf{prt}\ a.password |)\ (| \mathbf{prt}\ a.password |)$

$M_{better} \models \forall a{:}Account.\ \forall b{:}Num\ (| \mathbf{prt}\ a \wedge a.balance = b |)\ (| a.balance = b |)$

$M_{better} \models \forall a{:}Account.\ \forall b{:}Num\ (| \mathbf{prt}\ a.password \wedge a.balance = b |)\ (| a.balance \geq b |)$

$M_{bad} \models \forall a{:}Account\ (| \mathbf{prt}\ a |)\ (| \mathbf{prt}\ a |)$

$M_{bad} \not\models \forall a{:}Account\ (| \mathbf{prt}\ a.password |)\ (| \mathbf{prt}\ a.password |)$

$M_{bad} \models \forall a{:}Account.\ \forall b{:}Num\ (| \mathbf{prt}\ a \wedge a.balance = b |)\ (| a.balance = b |)$

$M_{bad} \not\models \forall a{:}Account.\ \forall b{:}Num\ (| \mathbf{prt}\ a.password \wedge a.balance = b |)\ (| a.balance \geq b |)$

Challenge_2: An inference system, such that

$$M_{good} \vdash S$$
$$M_{bad} \nvdash S$$
$$M_{better} \vdash S$$

Challenge_4: An inference system, such that we can prove external calls

In the context of arbitrary, unlimited calls from internal to external,
and arbitrary, unlimited calls from external to internal,

Challenge_2: An inference system, such that

$$M_{good} \vdash S$$
$$M_{bad} \nvdash S$$
$$M_{better} \vdash S$$

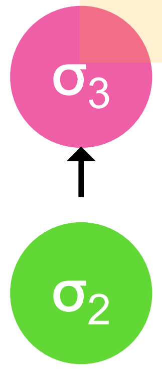
$\longrightarrow$

σ_2

In the context of arbitrary, unlimited calls from internal to external,
and arbitrary, unlimited calls from external to internal,

Challenge_2: An inference system, such that

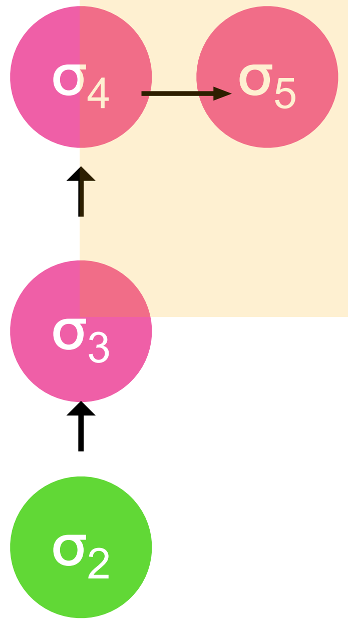
$$M_{good} \vdash S$$
$$M_{bad} \nvdash S$$
$$M_{better} \vdash S$$



In the context of arbitrary, unlimited calls from internal to external,
and arbitrary, unlimited calls from external to internal,

Challenge_2: An inference system, such that

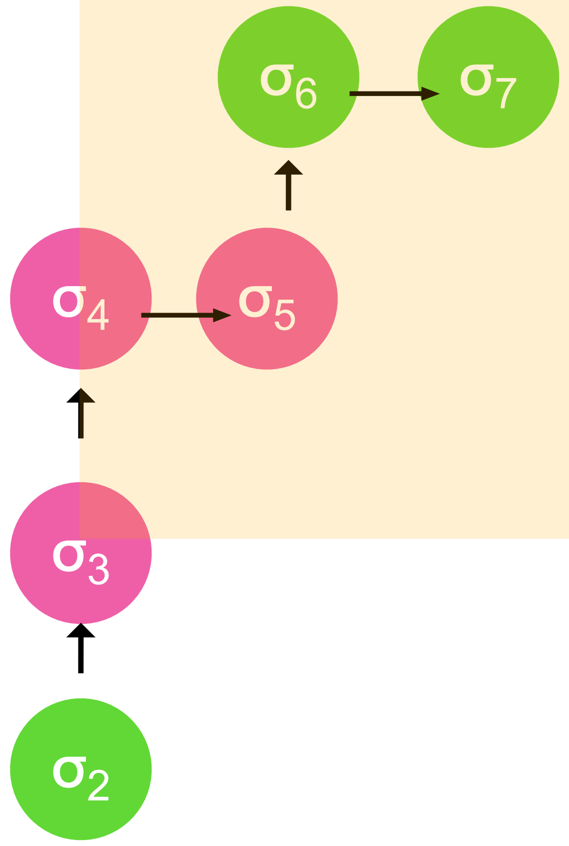
$$M_{good} \vdash S$$
$$M_{bad} \nvdash S$$
$$M_{better} \vdash S$$



In the context of arbitrary, unlimited calls from internal to external,
and arbitrary, unlimited calls from external to internal,

Challenge_2: An inference system, such that

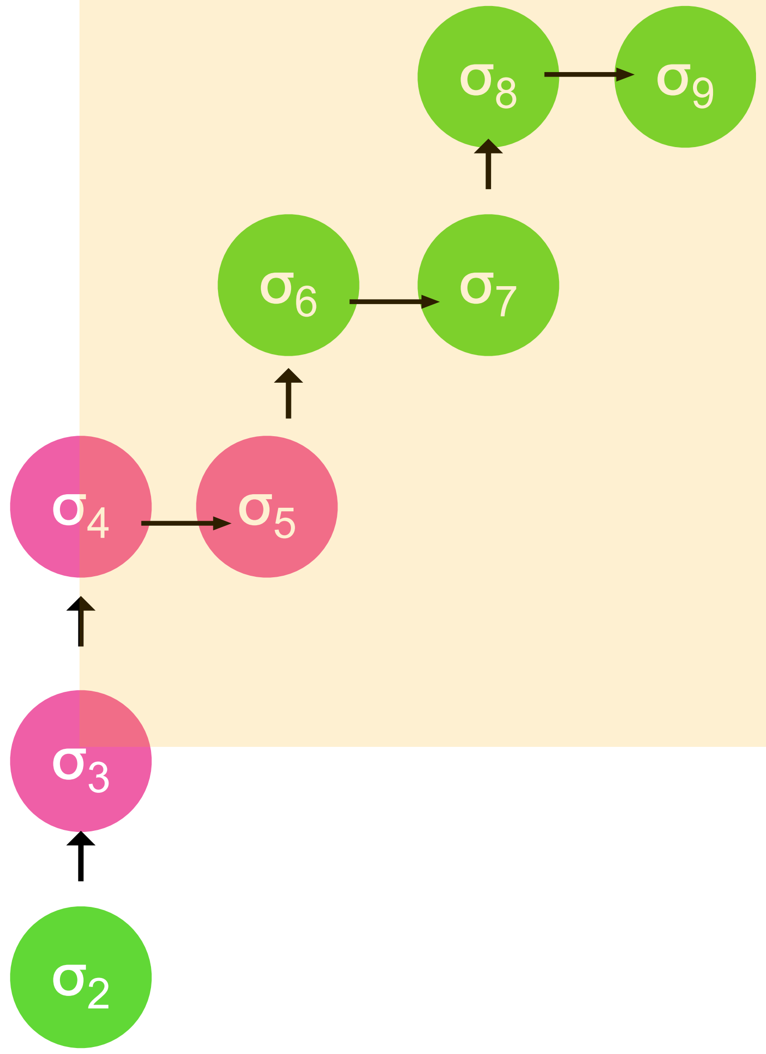
$M_{good} \vdash S$
$M_{bad} \nvdash S$
$M_{better} \vdash S$



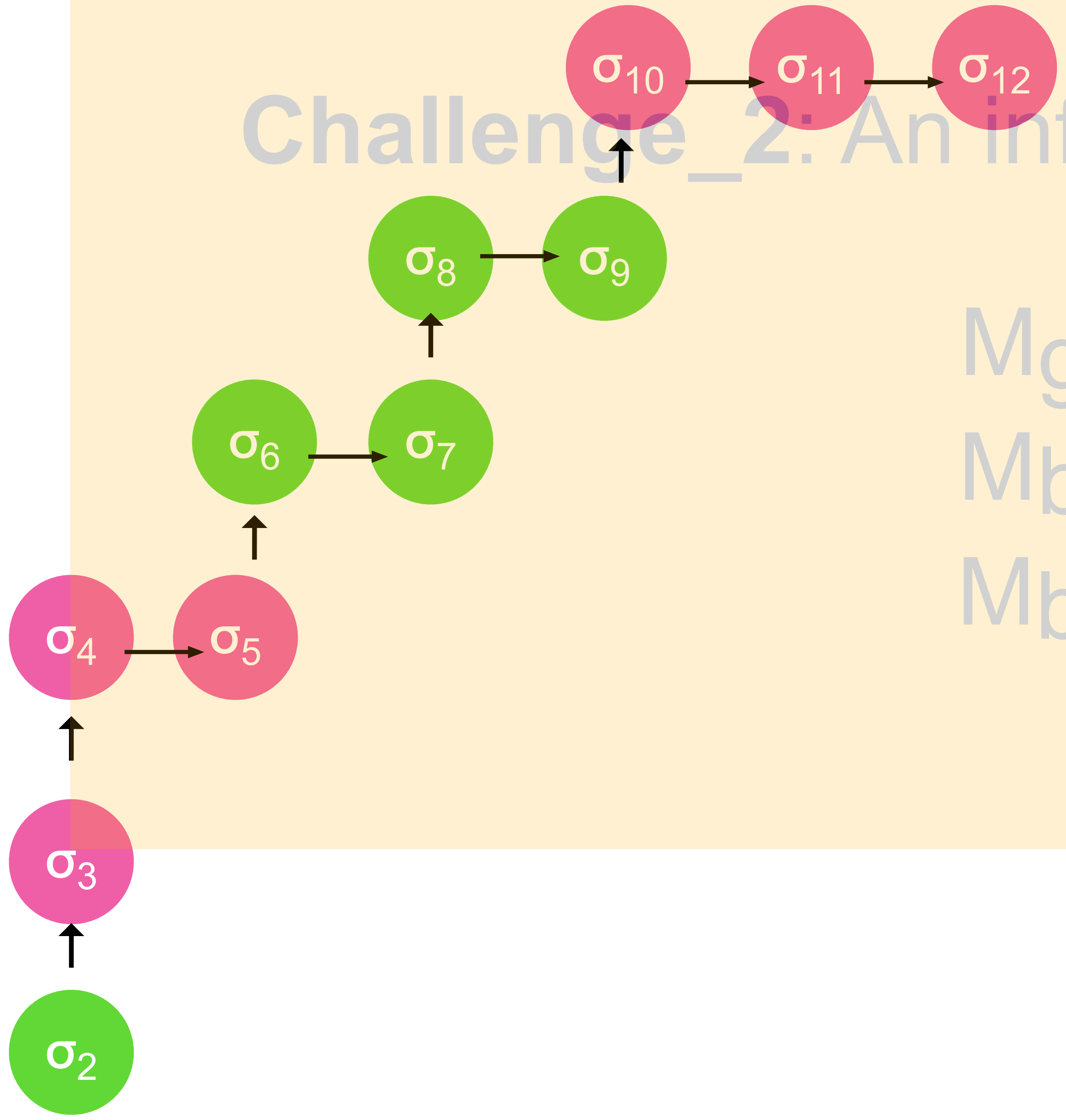
In the context of arbitrary, unlimited calls from internal to external,
and arbitrary, unlimited calls from external to internal,

Challenge_2: An inference system, such that

$M_{good} \vdash S$
$M_{bad} \nvdash S$
$M_{better} \vdash S$



In the context of arbitrary, unlimited calls from internal to external, and arbitrary, unlimited calls from external to internal,



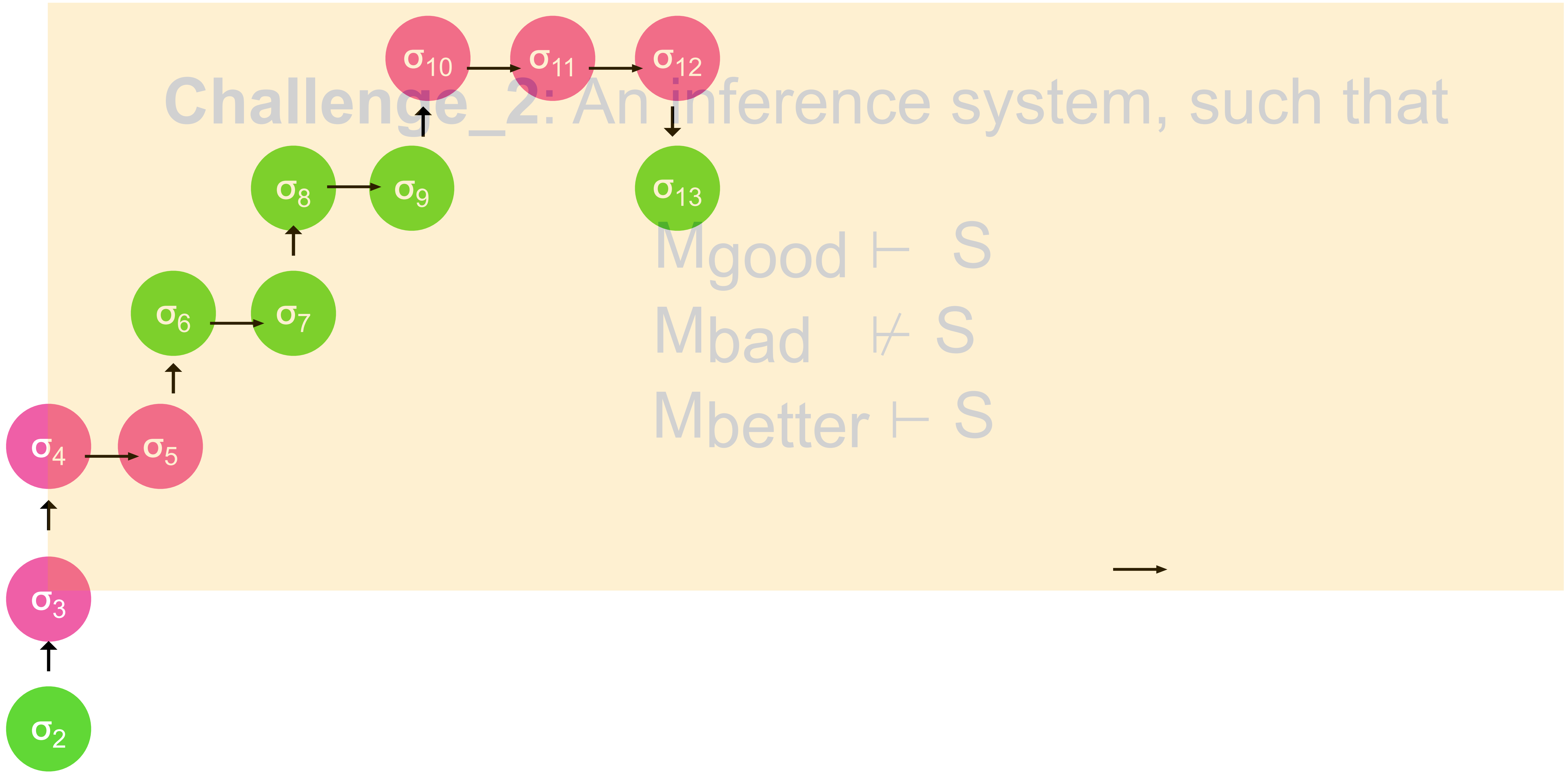
Challenge_2: An inference system, such that

$M_{good} \vdash S$

$M_{bad} \nvdash S$

$M_{better} \vdash S$

In the context of arbitrary, unlimited calls from internal to external,
and arbitrary, unlimited calls from external to internal,



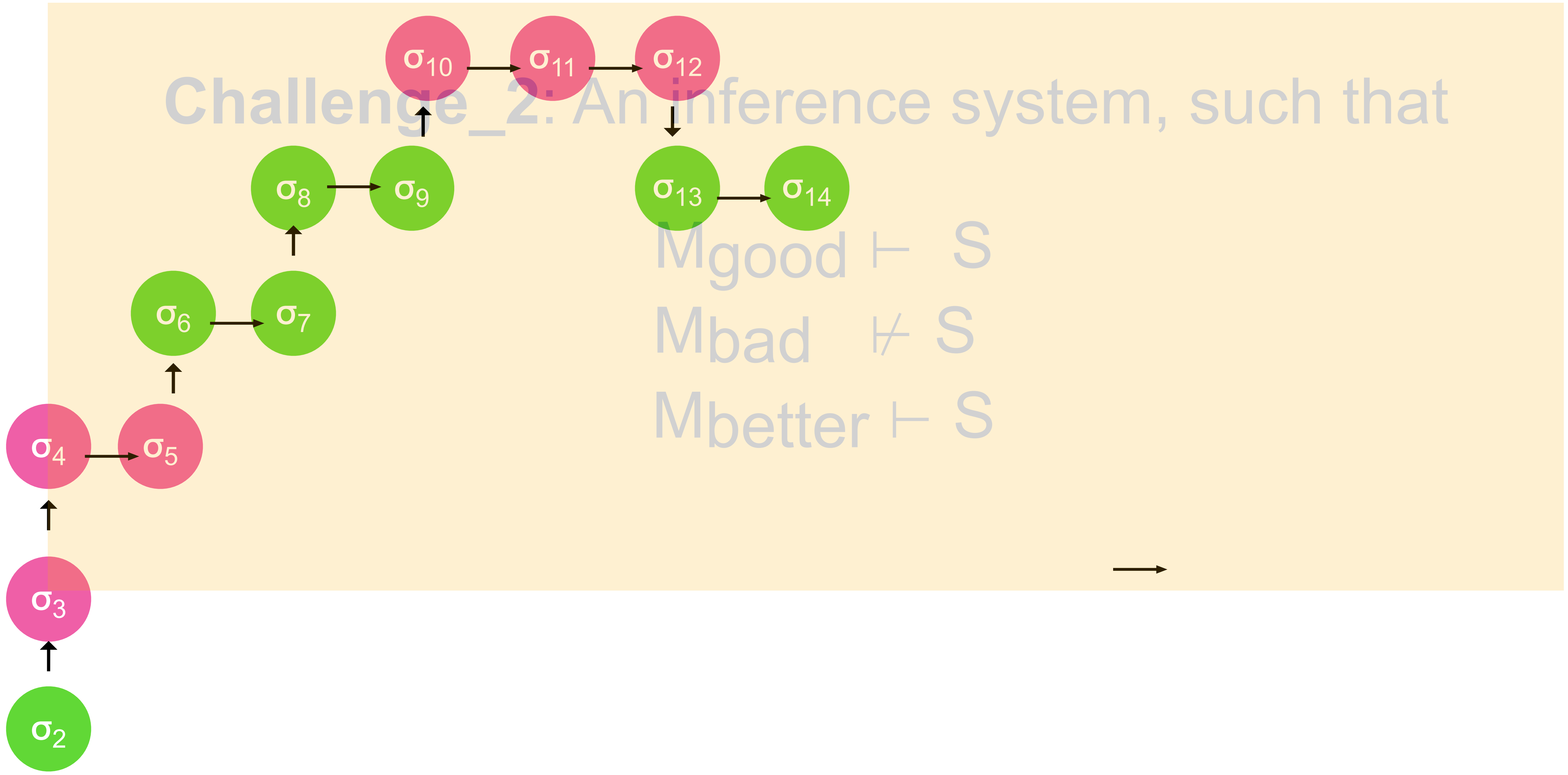
Challenge_2: An inference system, such that

$M_{good} \vdash S$

$M_{bad} \nvdash S$

$M_{better} \vdash S$

In the context of arbitrary, unlimited calls from internal to external,
and arbitrary, unlimited calls from external to internal,



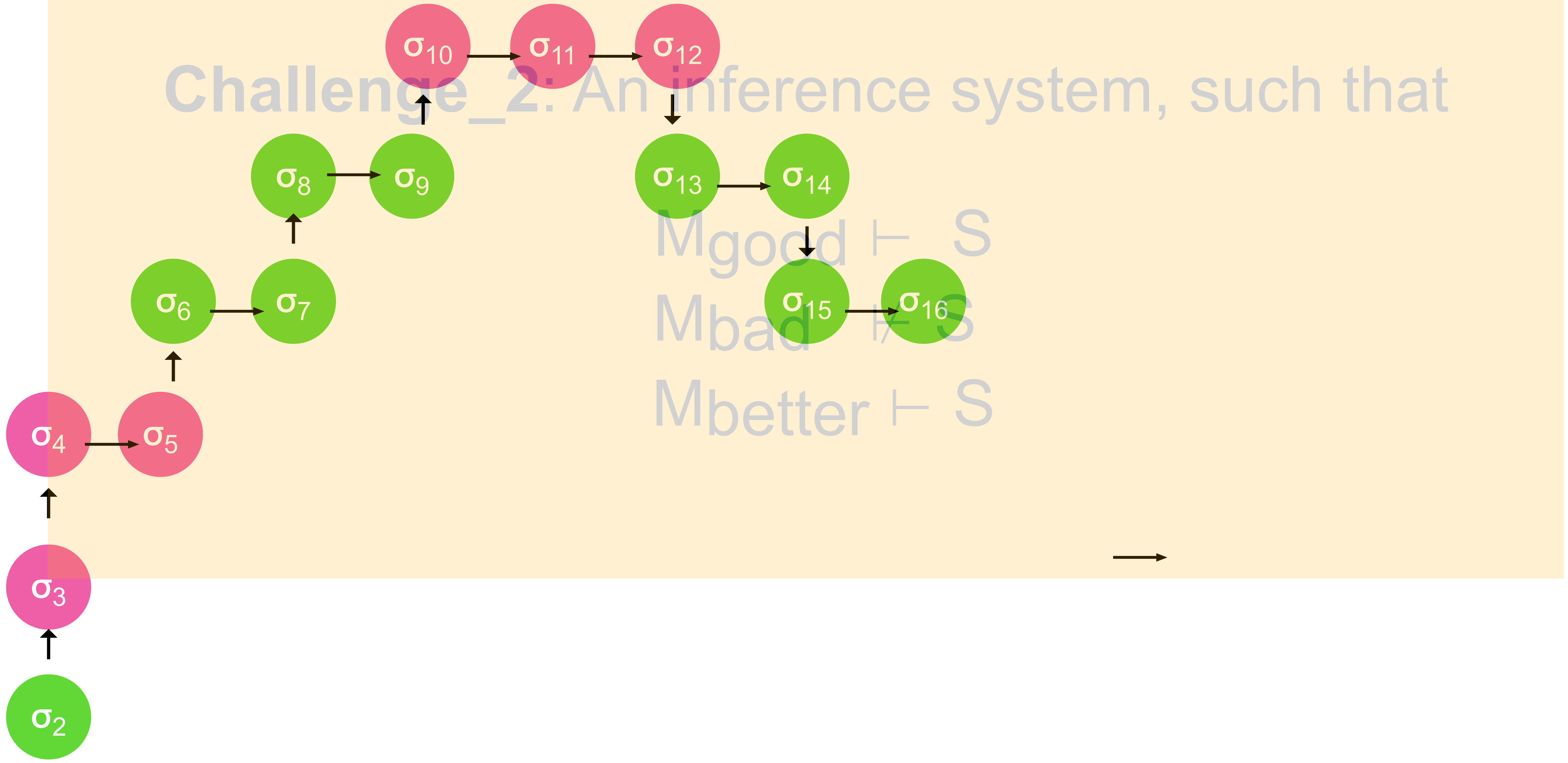
Challenge_2: An inference system, such that

$M_{good} \vdash S$

$M_{bad} \nvdash S$

$M_{better} \vdash S$

In the context of arbitrary, unlimited calls from internal to external,
and arbitrary, unlimited calls from external to internal,



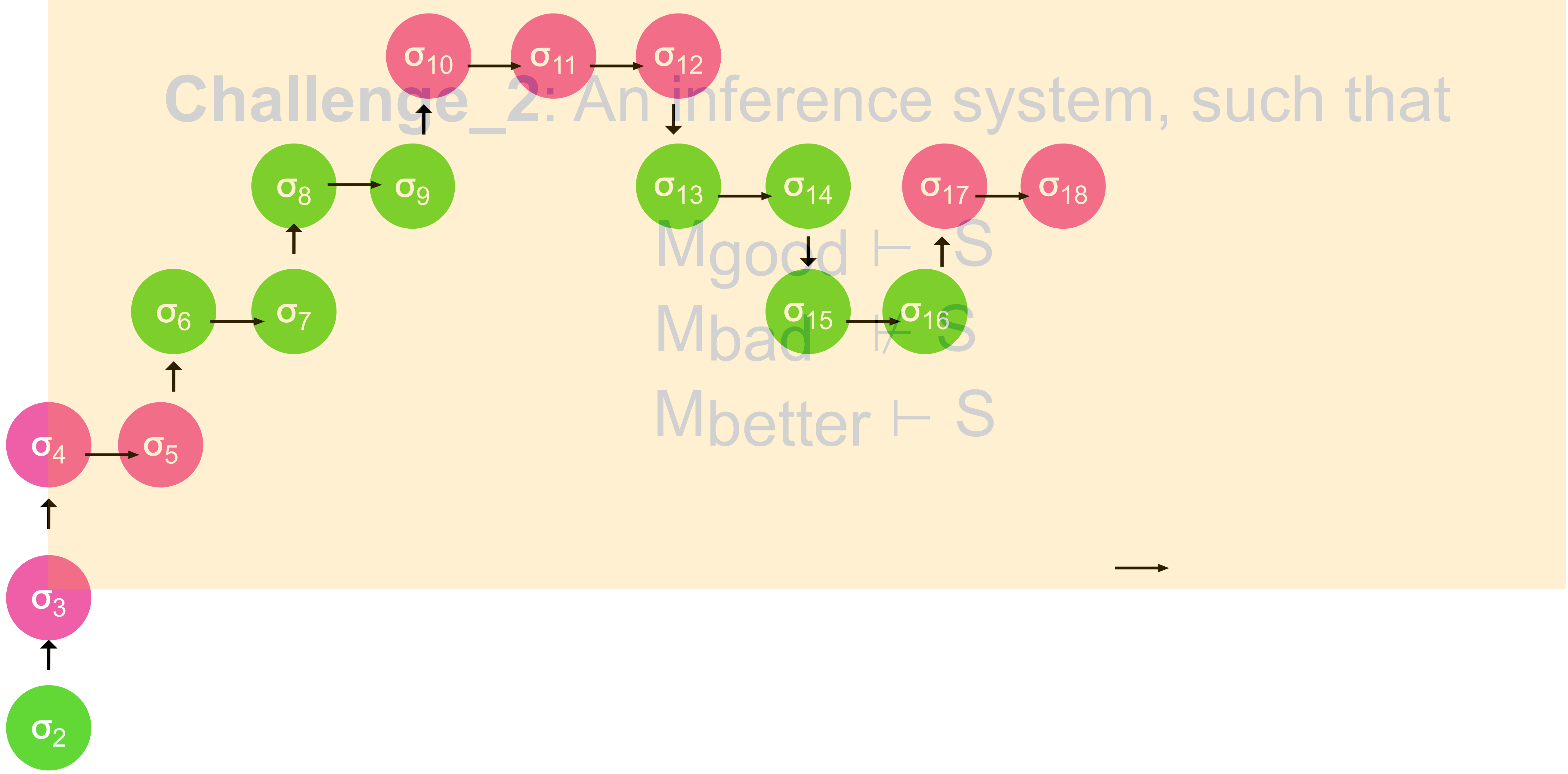
Challenge_2: An inference system, such that

$M_{good} \vdash S$

$M_{bad} \nvdash S$

$M_{better} \vdash S$

In the context of arbitrary, unlimited calls from internal to external,
and arbitrary, unlimited calls from external to internal,



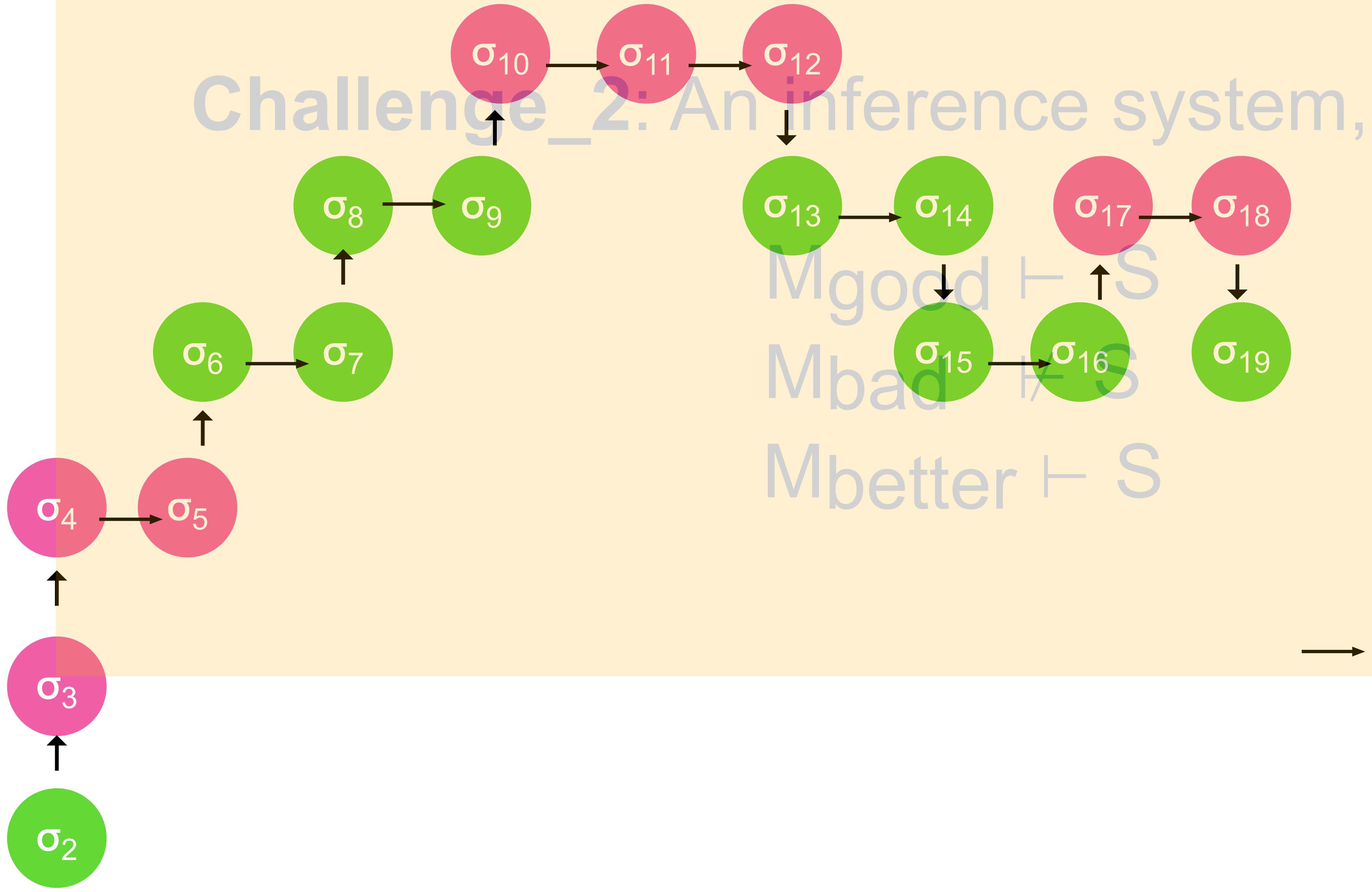
Challenge_2: An inference system, such that

$M_{good} \vdash S$

$M_{bad} \nvdash S$

$M_{better} \vdash S$

In the context of arbitrary, unlimited calls from internal to external,
and arbitrary, unlimited calls from external to internal,



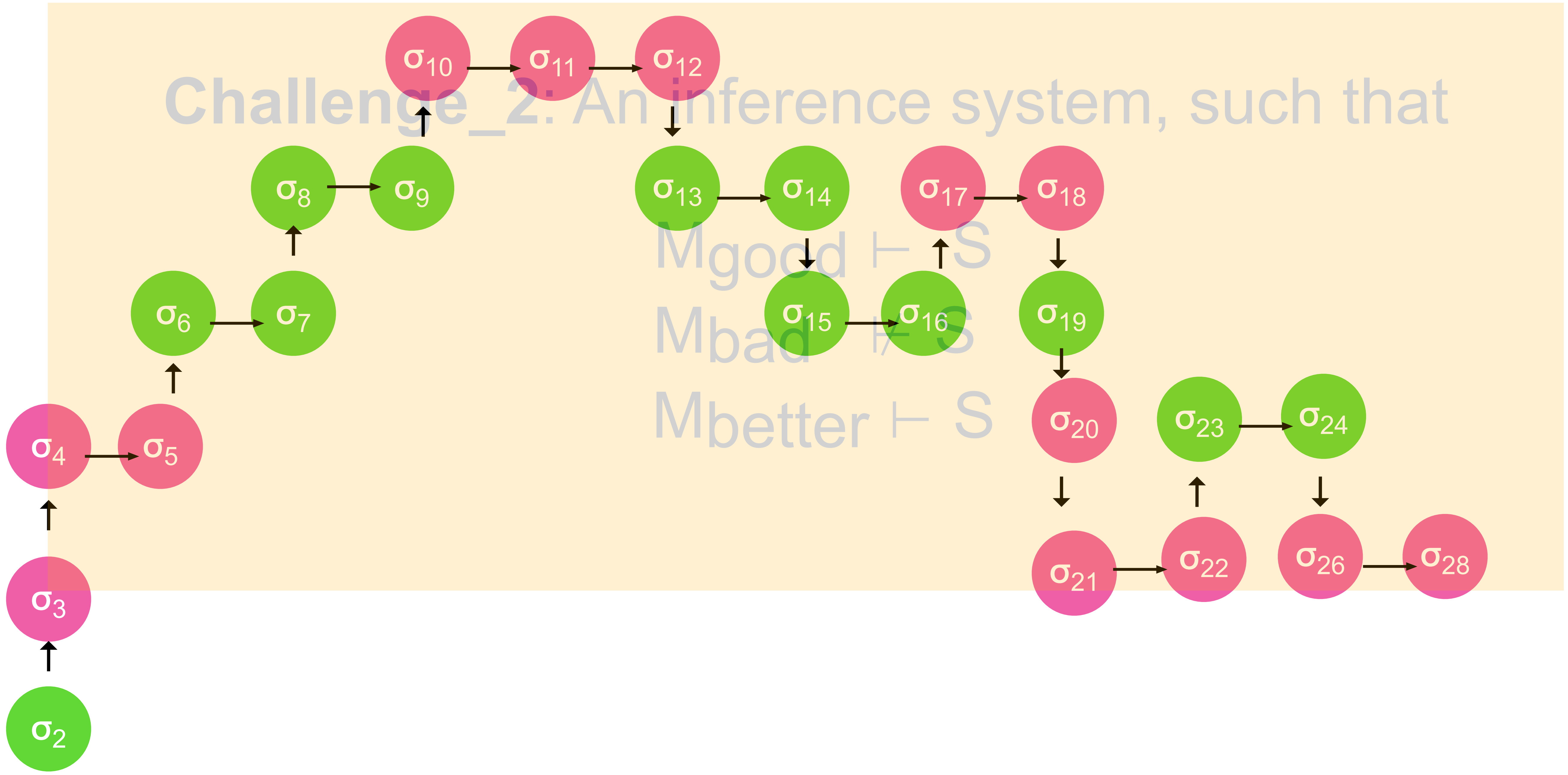
Challenge_2: An inference system, such that

$M_{good} \vdash S$

$M_{bad} \nvdash S$

$M_{better} \vdash S$

In the context of arbitrary, unlimited calls from internal to external,
and arbitrary, unlimited calls from external to internal,



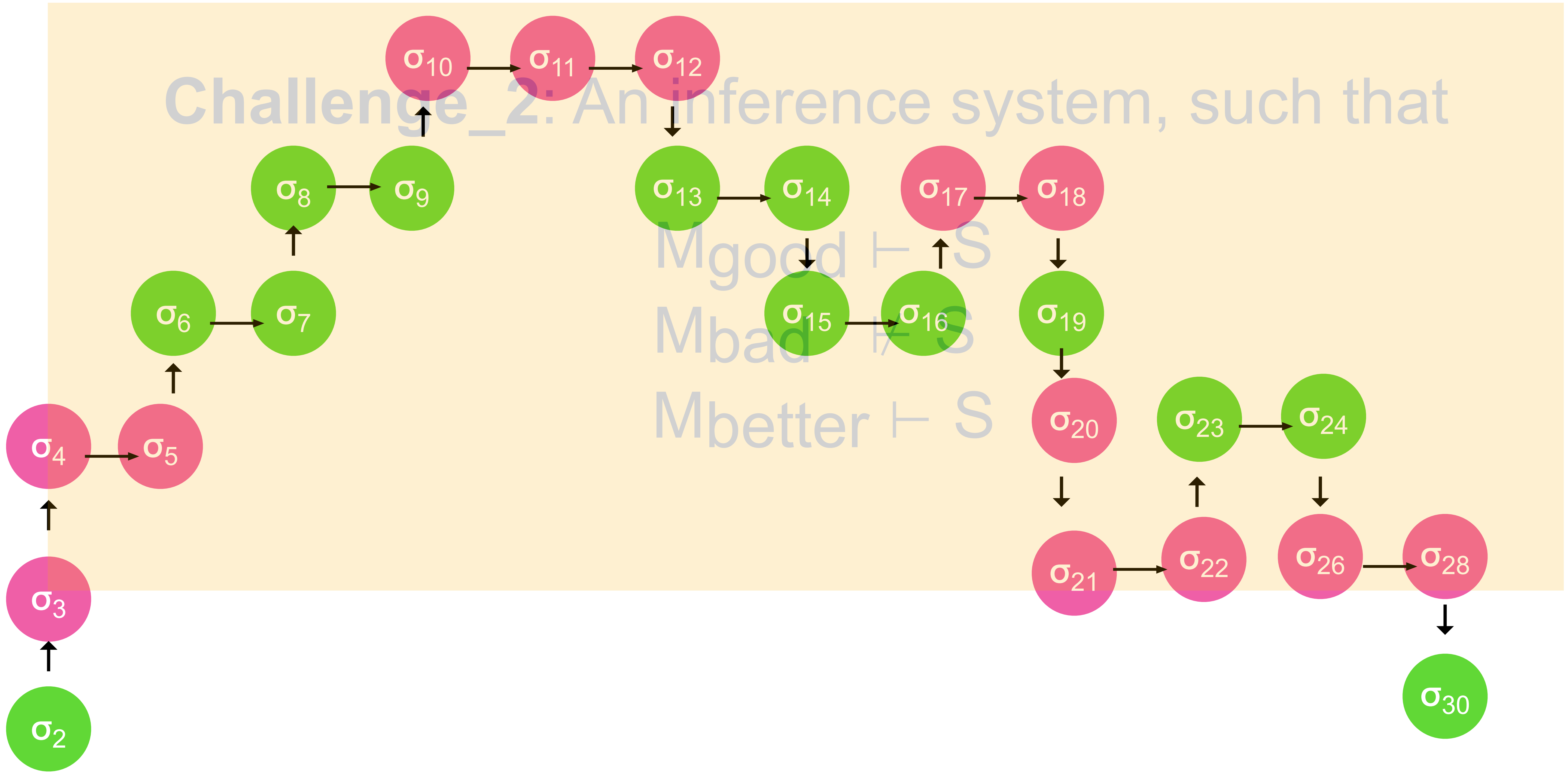
Challenge_2: An inference system, such that

$M_{good} \vdash S$

$M_{bad} \nvdash S$

$M_{better} \vdash S$

In the context of arbitrary, unlimited calls from internal to external,
and arbitrary, unlimited calls from external to internal,



Challenge_2: An inference system, such that

$M_{good} \vdash S$

$M_{bad} \nvdash S$

$M_{better} \vdash S$

Challenge_2/4: An inference system, such that …

An assertion A is **encapsulated** by module M, if
it can only be invalidated through calls to methods from M.

Challenge_2/4: An inference system, such that …

An assertion A is **encapsulated** by module M, if
it can only be invalidated through calls to methods from M.

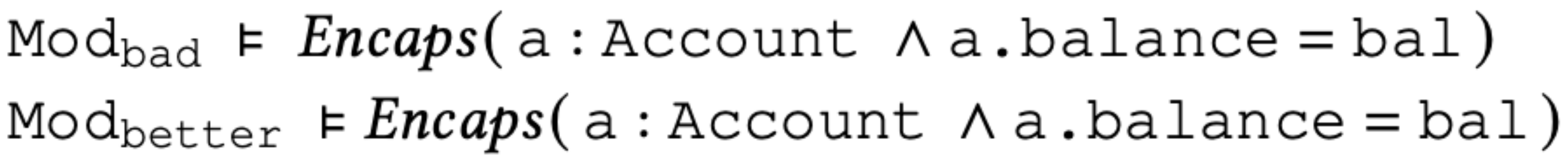
For example:

$$\texttt{Mod}_{\texttt{bad}} \models \mathit{Encaps}(\texttt{a} : \texttt{Account} \ \land \texttt{a.balance} = \texttt{bal})$$
$$\texttt{Mod}_{\texttt{better}} \models \mathit{Encaps}(\texttt{a} : \texttt{Account} \ \land \texttt{a.balance} = \texttt{bal})$$

Challenge_2/4: An inference system, such that …

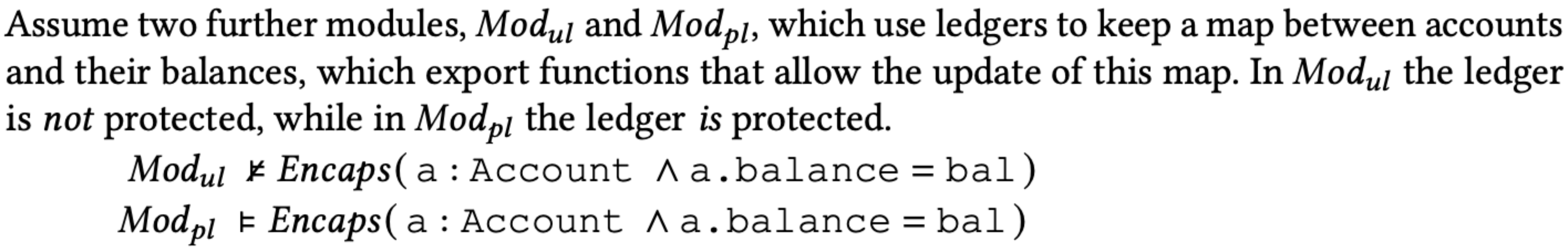
An assertion A is **encapsulated** by module M, if
it can only be invalidated through calls to methods from M.

For example:



$$\texttt{Mod}_{\texttt{bad}} \models \mathit{Encaps}(\texttt{a}:\texttt{Account} \wedge \texttt{a.balance}=\texttt{bal})$$

$$\texttt{Mod}_{\texttt{better}} \models \mathit{Encaps}(\texttt{a}:\texttt{Account} \wedge \texttt{a.balance}=\texttt{bal})$$



Assume two further modules, Mod_{ul} and Mod_{pl}, which use ledgers to keep a map between accounts and their balances, which export functions that allow the update of this map. In Mod_{ul} the ledger is *not* protected, while in Mod_{pl} the ledger *is* protected.

$$Mod_{ul} \not\models \mathit{Encaps}(\texttt{a}:\texttt{Account} \wedge \texttt{a.balance}=\texttt{bal})$$

$$Mod_{pl} \models \mathit{Encaps}(\texttt{a}:\texttt{Account} \wedge \texttt{a.balance}=\texttt{bal})$$

Challenge_2/4: An inference system, such that …

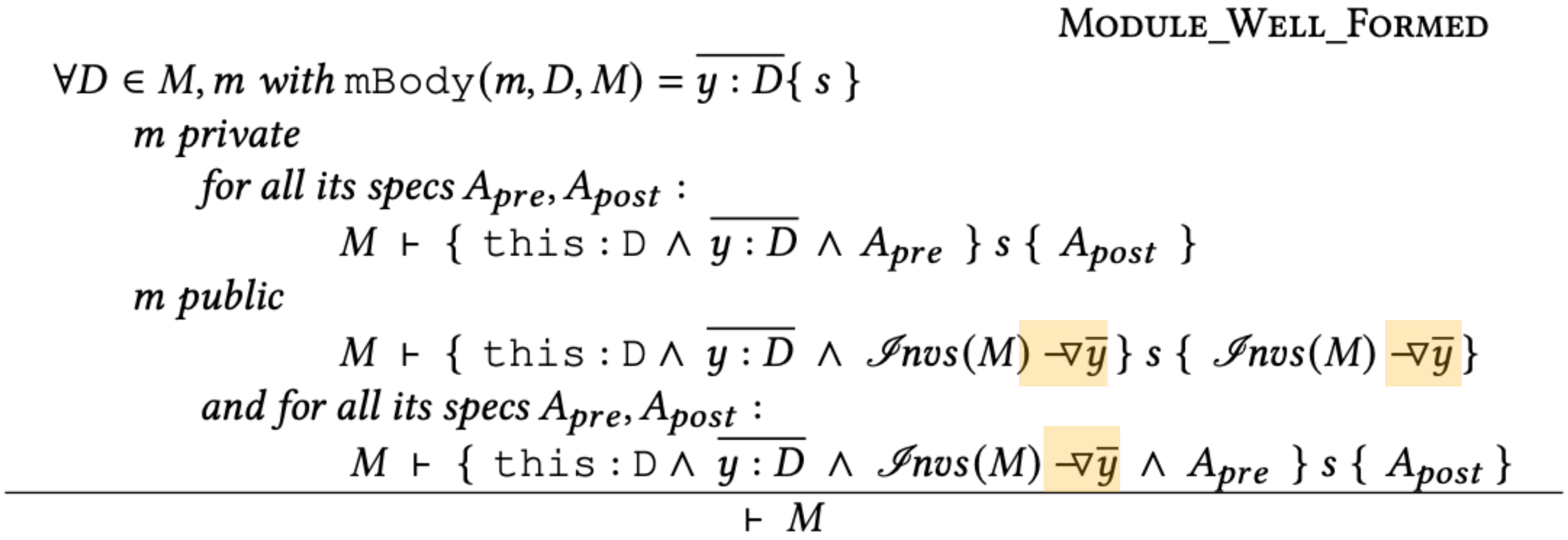
$$\frac{\begin{array}{l} \forall D \in M, m \textit{ with } \texttt{mBody}(m,D,M) = \overline{y:D}\{\ s\ \} \\ \quad m \textit{ private} \\ \quad\quad \textit{for all its specs } A_{pre}, A_{post}\ : \\ \quad\quad\quad M\ \vdash\ \{\ \texttt{this}:\texttt{D}\ \wedge\ \overline{y:D}\ \wedge\ A_{pre}\ \}\ s\ \{\ A_{post}\ \} \\ \quad m \textit{ public} \\ \quad\quad\quad M\ \vdash\ \{\ \texttt{this}:\texttt{D} \wedge\ \overline{y:D}\ \wedge\ \text{???}\ \}\ s\ \{\ \text{???}\ \} \\ \quad\quad \textit{and for all its specs } A_{pre}, A_{post}\ : \\ \quad\quad\quad M\ \vdash\ \{\ \texttt{this}:\texttt{D} \wedge\ \overline{y:D}\ \wedge\ \text{???}\ \wedge\ A_{pre}\ \}\ s\ \{\ A_{post}\ \} \end{array}}{\vdash\ M}\ \textsc{Module\_Well\_Formed}$$

Challenge_2/4: An inference system, such that …

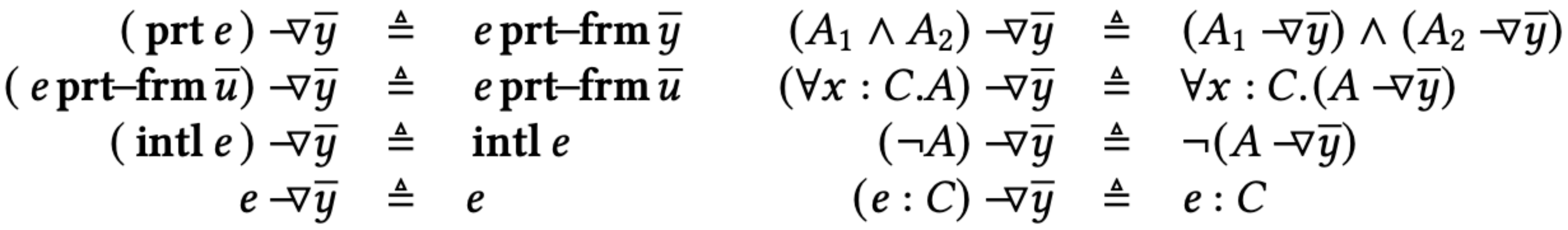
$$
\dfrac{
\begin{array}{l}
\forall D \in M, m \textit{ with } \mathtt{mBody}(m, D, M) = \overline{y : D}\{\ s\ \} \\
\quad m \textit{ private} \\
\qquad \textit{for all its specs } A_{pre}, A_{post} : \\
\qquad\qquad M \vdash \{\ \mathtt{this : D} \wedge \overline{y : D} \wedge A_{pre}\ \}\ s\ \{\ A_{post}\ \} \\
\quad m \textit{ public} \\
\qquad\qquad M \vdash \{\ \mathtt{this : D} \wedge \overline{y : D} \wedge \mathscr{I}\mathit{nvs}(M)\ ??\ \}\ s\ \{\ \mathscr{I}\mathit{nvs}(M)\ ??\ \} \\
\qquad \textit{and for all its specs } A_{pre}, A_{post} : \\
\qquad\qquad M \vdash \{\ \mathtt{this : D} \wedge \overline{y : D} \wedge \mathscr{I}\mathit{nvs}(M)\ ??\ \wedge A_{pre}\ \}\ s\ \{\ A_{post}\ \}
\end{array}
}{\vdash\ M}\ \textsc{Module\_Well\_Formed}
$$

$$
\begin{array}{lcl}
\mathscr{I}\mathit{nvs}(\forall \overline{x : C}.\llparenthesis A \rrparenthesis) & \triangleq & \forall \overline{x}.[\ \overline{x : C} \rightarrow A\ ] \\
\mathscr{I}\mathit{nvs}(\forall \overline{x : C}.\llparenthesis A \rrparenthesis \llparenthesis A \rrparenthesis) & \triangleq & \overline{x : C} \wedge A \\
\mathscr{I}\mathit{nvs}(S_1 \wedge S_2) & \triangleq & \mathscr{I}\mathit{nvs}(S_1)\ \wedge S_2\ \mathscr{I}\mathit{nvs}(S_2) \\
\mathscr{I}\mathit{nvs}(M) & \triangleq & \mathscr{I}\mathit{nvs}(HS(M))
\end{array}
$$

Challenge_2/4: An inference system, such that …

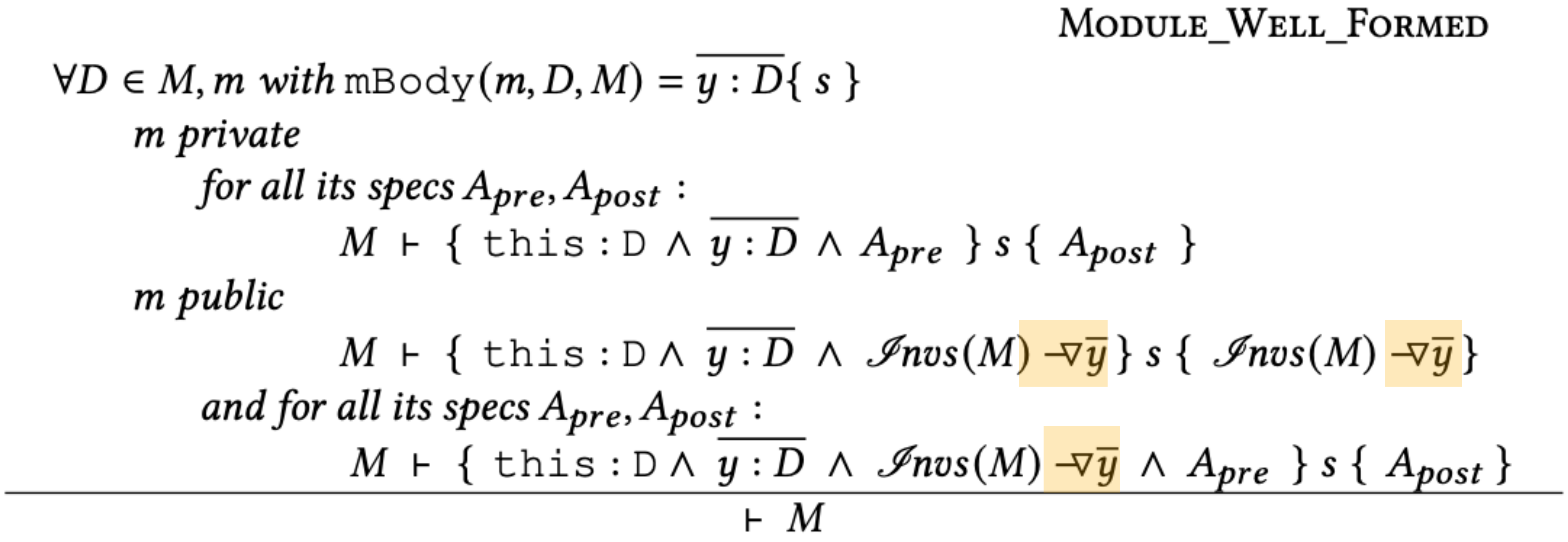


$$
\dfrac{
\begin{array}{l}
\forall D \in M, m \textit{ with } \texttt{mBody}(m, D, M) = \overline{y : D}\{\ s\ \} \\
\quad m \textit{ private} \\
\qquad \textit{for all its specs } A_{pre}, A_{post}\ : \\
\qquad\qquad M\ \vdash\ \{\ \texttt{this}:\texttt{D} \wedge \overline{y : D} \wedge A_{pre}\ \}\ s\ \{\ A_{post}\ \} \\
\quad m \textit{ public} \\
\qquad\qquad M\ \vdash\ \{\ \texttt{this}:\texttt{D} \wedge \overline{y : D} \wedge \mathscr{I}nvs(M) \mathbin{-\!\nabla} \overline{y}\ \}\ s\ \{\ \mathscr{I}nvs(M) \mathbin{-\!\nabla} \overline{y}\ \} \\
\qquad \textit{and for all its specs } A_{pre}, A_{post}\ : \\
\qquad\qquad M\ \vdash\ \{\ \texttt{this}:\texttt{D} \wedge \overline{y : D} \wedge \mathscr{I}nvs(M) \mathbin{-\!\nabla} \overline{y} \wedge A_{pre}\ \}\ s\ \{\ A_{post}\ \}
\end{array}
}{
\vdash\ M
}
\quad \textsc{Module\_Well\_Formed}
$$



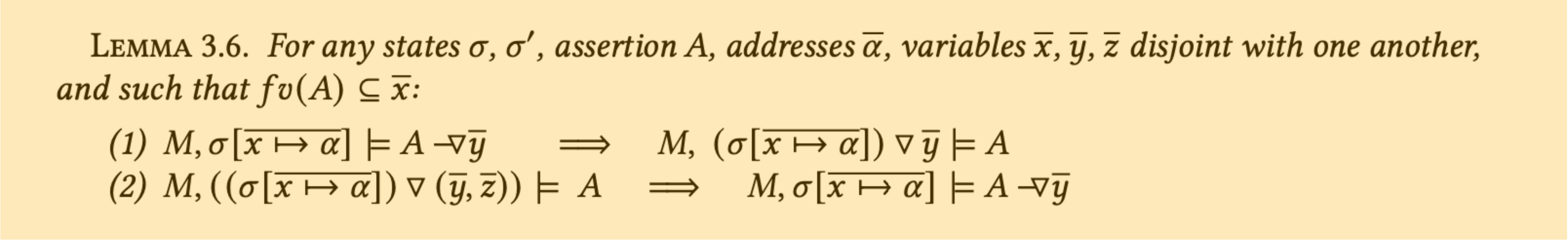
$$
\begin{array}{rcl@{\qquad}rcl}
(\,\mathbf{prt}\ e\,) \mathbin{-\!\nabla} \overline{y} & \triangleq & e\ \mathbf{prt\text{–}frm}\ \overline{y} & (A_1 \wedge A_2) \mathbin{-\!\nabla} \overline{y} & \triangleq & (A_1 \mathbin{-\!\nabla} \overline{y}) \wedge (A_2 \mathbin{-\!\nabla} \overline{y}) \\
(\,e\ \mathbf{prt\text{–}frm}\ \overline{u}) \mathbin{-\!\nabla} \overline{y} & \triangleq & e\ \mathbf{prt\text{–}frm}\ \overline{u} & (\forall x : C.A) \mathbin{-\!\nabla} \overline{y} & \triangleq & \forall x : C.(A \mathbin{-\!\nabla} \overline{y}) \\
(\,\mathbf{intl}\ e\,) \mathbin{-\!\nabla} \overline{y} & \triangleq & \mathbf{intl}\ e & (\neg A) \mathbin{-\!\nabla} \overline{y} & \triangleq & \neg(A \mathbin{-\!\nabla} \overline{y}) \\
e \mathbin{-\!\nabla} \overline{y} & \triangleq & e & (e : C) \mathbin{-\!\nabla} \overline{y} & \triangleq & e : C
\end{array}
$$

Challenge_2/4: An inference system, such that …



MODULE_WELL_FORMED

$$\dfrac{\begin{array}{l} \forall D \in M, m\ \textit{with}\ \texttt{mBody}(m, D, M) = \overline{y:D}\{\ s\ \} \\ \quad m\ \textit{private} \\ \quad\quad \textit{for all its specs}\ A_{pre}, A_{post}: \\ \quad\quad\quad M\ \vdash\ \{\ \texttt{this}:\texttt{D} \wedge \overline{y:D} \wedge A_{pre}\ \}\ s\ \{\ A_{post}\ \} \\ \quad m\ \textit{public} \\ \quad\quad\quad M\ \vdash\ \{\ \texttt{this}:\texttt{D} \wedge \overline{y:D} \wedge \mathscr{I}nvs(M) -\!\!\!\triangledown \overline{y}\ \}\ s\ \{\ \mathscr{I}nvs(M) -\!\!\!\triangledown \overline{y}\ \} \\ \quad\quad \textit{and for all its specs}\ A_{pre}, A_{post}: \\ \quad\quad\quad M\ \vdash\ \{\ \texttt{this}:\texttt{D} \wedge \overline{y:D} \wedge \mathscr{I}nvs(M) -\!\!\!\triangledown \overline{y} \wedge A_{pre}\ \}\ s\ \{\ A_{post}\ \} \end{array}}{\vdash\ M}$$



LEMMA 3.6. *For any states σ, σ', assertion A, addresses $\overline{\alpha}$, variables $\overline{x}$, $\overline{y}$, $\overline{z}$ disjoint with one another, and such that $fv(A) \subseteq \overline{x}$:*

(1) $M, \sigma[\overline{x \mapsto \alpha}] \models A -\!\!\!\triangledown \overline{y} \quad \Longrightarrow \quad M,\ (\sigma[\overline{x \mapsto \alpha}]) \triangledown \overline{y} \models A$

(2) $M, ((\sigma[\overline{x \mapsto \alpha}]) \triangledown (\overline{y}, \overline{z})) \models\ A \quad \Longrightarrow \quad M, \sigma[\overline{x \mapsto \alpha}] \models A -\!\!\!\triangledown \overline{y}$

Challenge_2/4: An inference system, such that …

$$\textsc{Module\_Well\_Formed}$$

$$\frac{\begin{array}{l}
\forall D \in M, m \textit{ with } \texttt{mBody}(m, D, M) = \overline{y : D}\{\ s\ \} \\
\quad m \textit{ private} \\
\quad\quad \textit{for all its specs } A_{pre}, A_{post} : \\
\quad\quad\quad M\ \vdash\ \{\ \texttt{this} : \texttt{D} \land \overline{y : D} \land A_{pre}\ \}\ s\ \{\ A_{post}\ \} \\
\quad m \textit{ public} \\
\quad\quad\quad M\ \vdash\ \{\ \texttt{this} : \texttt{D} \land \overline{y : D} \land \mathscr{I}nvs(M) -\!\nabla \overline{y}\ \}\ s\ \{\ \mathscr{I}nvs(M) -\!\nabla \overline{y}\ \} \\
\quad\quad \textit{and for all its specs } A_{pre}, A_{post} : \\
\quad\quad\quad M\ \vdash\ \{\ \texttt{this} : \texttt{D} \land \overline{y : D} \land \mathscr{I}nvs(M) -\!\nabla \overline{y} \land A_{pre}\ \}\ s\ \{\ A_{post}\ \}
\end{array}}{\vdash\ M}$$

$$\textsc{Weaken} \qquad \frac{M \vdash S \qquad M \vdash S \sqsubseteq S'}{M \vdash S'}$$

$$\textsc{Multi} \qquad \frac{M\ \vdash\ S \qquad M\ \vdash\ S'}{M\ \vdash\ S \land S'}$$

Challenge_2/4: … prove method bodies ….

EXTEND

$$\frac{M \vdash_{ul} \{\, A \,\}\ s\ \{\, A' \,\}}{M\ \vdash\ \{\, A \,\}\ s\ \{\, A' \,\}}$$

COMBINE

$$\frac{M\ \vdash\ \{\, A_1 \,\}\ s\ \{\, A_1' \,\} \quad M\ \vdash\ \{\, A_2 \,\}\ s\ \{\, A_2' \,\}}{M\ \vdash\ \{\, A_1 \wedge A_2 \,\}\ s\ \{\, A_1' \wedge A_2' \,\}}$$

SEQU

$$\frac{M\ \vdash\ \{\, A \,\}\ s_1\ \{\, A'' \,\} \quad M\ \vdash\ \{\, A'' \,\}\ s_2\ \{\, A' \,\}}{M\ \vdash\ \{\, A \,\}\ s_1;\ s_2\ \{\, A' \,\}}$$

CONSEQU

$$\frac{M \vdash A \rightarrow A'' \quad M\ \vdash\ \{\, A'' \,\}\ s\ \{\, A''' \,\} \quad M \vdash A''' \rightarrow A'}{M\ \vdash\ \{\, A \,\}\ s\ \{\, A' \,\}}$$

[IntCall_WithSpec]

$$\frac{M, C, m : (\, A_1,\ A_2 \,) \qquad fv(A_1) = \overline{x}}{M\ \vdash\ \{\ \ \mathbf{intl}\, y_0 \wedge y : C\ \wedge\ A_1[\overline{y}/\mathtt{this}, \overline{x}]\ \ \}\ u := y_0.m(y_1, ..y_n)\ \{\ A_2[u/result, \overline{y}/\mathtt{this}, \overline{x}]\ \ \}}$$

Challenge_2/4: …. prove external calls

[CallAndAlias

$$\frac{x \overset{\text{txt}}{\neq} u \overset{\text{txt}}{\neq} x'}{M \ \vdash\ \{\ x = x' \ \wedge\ (\,\mathbf{extl}\, y_0 \ \longrightarrow\ \mathscr{I}nvs(M) \ -\!\nabla\overline{y})\ \}\ u := y_0.m(y_1, ..y_n)\ \{\ x = x'\ \}}$$

[CallNonAlias]

$$\frac{x \overset{\text{txt}}{\neq} u \overset{\text{txt}}{\neq} x'}{M \ \vdash\ \{\ x \neq x' \ \wedge\ (\,\mathbf{extl}\, y_0 \ \longrightarrow\ \mathscr{I}nvs(M) \ -\!\nabla\overline{y})\ \}\ u := y_0.m(y_1, ..y_n))\ \{\ x \neq x'\ \}}$$

Challenge_2/4: …. prove external calls

$$[\textsc{ExtCal}]$$

$$\frac{}{M \ \vdash \ \{\ \ \mathbf{extl}\, y_0 \ \ \wedge \ \mathscr{I}nvs(M) \mathrel{-\!\!\!\nabla} \overline{y} \ \ \}\ u := y_0.m(y_1, ..y_n)\ \{\ \mathscr{I}nvs(M) \mathrel{-\!\!\!\nabla} \overline{y} \ \ \}}$$

Challenge_2/4: …. prove external calls

$$\frac{\vdash\ M\ :\ \ ???}{M\ \vdash\ \{\ \mathbf{extl}\ y_0\ \wedge\ \overline{x:C}\ \wedge\ ???\ \wedge\ \mathscr{I}nvs(M)\ -\!\!\nabla\overline{y}\ \}\ u := y_0.m(y_1,..y_n)\ \{\ \ ???\ \ \}}\quad[\textsc{ExtCall\_WithSpec\_Weak}]$$

Challenge_2/4: …. prove external calls

[ExtCall_WithSpec_Weak]

$$\frac{\vdash\ M\ :\ \text{???}}{M\ \vdash\ \{\ \mathbf{extl}\, y_0\ \wedge\ \overline{x : C}\ \wedge\ \text{???}\ \wedge\ \mathscr{I}nvs(M) -\!\nabla \overline{y}\ \}\ u := y_0.m(y_1, ..y_n)\ \{\ \text{???}\ \}}$$

Motto:
Capability is a *necessary* condition
for some effect

Challenge_2/4: …. prove external calls

[ExtCall_WithSpec_Weak]

$$\frac{\vdash \ M \ : \ \forall \overline{x : C}.\llparenthesis\, A_1 \,\rrparenthesis\llparenthesis\, A_2 \,\rrparenthesis}{M \ \vdash \ \{ \ \mathbf{extl}\, y_0 \ \wedge \ \overline{x : C} \ \wedge \ ??? \ \wedge \ \mathscr{I}nvs(M) -\!\nabla \overline{y} \ \} \ u := y_0.m(y_1, ..y_n) \ \{ \ ??? \ \}}$$

Motto:
Capability is a *necessary* condition for some effect

Challenge_2/4: …. prove external calls

[ExtCall_WithSpec_Weak]

$$\frac{\vdash \;\; M \;:\; \forall \overline{x:C}.\llparenthesis\, A_1 \,\rrparenthesis \llparenthesis\, A_2 \,\rrparenthesis}{M \;\vdash\; \{\;\; \mathbf{extl}\, y_0 \;\wedge\; \overline{x:C} \;\wedge\; A_1 -\!\!\nabla \overline{y} \;\wedge\; \mathscr{I}nvs(M) -\!\!\nabla \overline{y} \;\}\; u := y_0.m(y_1, ..y_n) \;\{\;\; ??? \;\;\}}$$

Motto:
Capability is a *necessary* condition
for some effect

Challenge_2/4: …. prove external calls

[ExtCall_WithSpec_Weak]

$$\frac{\vdash\ M\ :\ \forall \overline{x:C}.(\!| A_1 |\!)(\!| A_2 |\!)}{M\ \vdash\ \{\ \mathbf{extl}\, y_0\ \wedge\ \overline{x:C}\ \wedge\ A_1 -\!\nabla \overline{y}\ \wedge\ \mathscr{I}nvs(M) -\!\nabla \overline{y}\ \}\ u := y_0.m(y_1, ..y_n)\ \{\ A_2 -\!\nabla \overline{y}\ \}}$$

Motto:
Capability is a *necessary* condition
for some effect

Challenge_2/4: …. prove external calls

[EXTCALL_WITHSPEC_STRONG]

$$\frac{\vdash \;\; M \;:\; \forall \overline{x:C}.(\!|\, A_1 \,|\!)(\!|\, A_2 \,|\!)}{M \;\vdash\; \{\;\; \mathbf{extl}\, y_0 \;\wedge\; \overline{x:C} \;\wedge\; A_1 -\!\!\nabla \overline{y} \;\wedge\; ??? \;\wedge\; \mathscr{I}nvs(M) -\!\!\nabla \overline{y} \;\;\}\; u := y_0.m(y_1, ..y_n) \;\{\;\; A_2 -\!\!\nabla \overline{y} \quad ??? \;\;\}}$$

Challenge_2/4: …. prove external calls

[ExtCall_WithSpec_Strong]

$$\frac{\vdash\ M\ :\ \forall\overline{x:C}.(\!|\ A_1\ |\!)(\!|\ A_2\ |\!)}{M\ \vdash\ \{\ \mathbf{extl}\,y_0\ \wedge\ \overline{x:C}\ \wedge\ A_1 \mathrel{-\!\nabla} \overline{y}\ \wedge\ ???\ \wedge\ \mathscr{I}nvs(M) \mathrel{-\!\nabla} \overline{y}\ \}\ u := y_0.m(y_1,..y_n)\ \{\ A_2 \mathrel{-\!\nabla} \overline{y}\ \ ???\ \}}$$

Motto:
Capability is a *necessary* condition for some effect

Challenge_2/4: …. prove external calls

[ExtCall_WithSpec_Strong]

$$\frac{\vdash \;\; M \;:\; \forall \overline{x:C}.\llparenthesis\, A_1 \,\rrparenthesis \llparenthesis\, A_2 \,\rrparenthesis}{M \;\vdash\; \{\;\; \textbf{extl}\, y_0 \;\;\wedge\; \overline{x:C} \;\wedge\; A_1 \multimap\!\nabla \overline{y} \;\wedge\; A_1 \;\wedge\; \mathscr{I}nvs(M) \multimap\!\nabla \overline{y} \;\;\}\; u := y_0.m(y_1, ..y_n) \;\{\;\; A_2 \multimap\!\nabla \overline{y} \;\wedge\; A_2 \;\;\}}$$

Motto:
Capability is a *necessary* condition for some effect

Challenge_2/4: …. Revisiting the Shop example

```
class Shop

  fld myAccount : Account
  fld inventory : Inventory

  void buy(buyer: Object, anItem: Item)
    int price = anItem.price
    int oldBalance = this.myAccount.balance
    buyer.payMe(myAccount,price)
    if (this.myAccount.balance == oldBalance+price)
       this.send(buyer,anItem)
    else
       buyer.tell("you have not paid me")
```

Challenge_2/4: …. Revisiting the Shop exam

```
class Shop

  fld myAccount : Account
  fld inventory : Inventory

  void buy(buyer: Object, anItem: Item)
    int price = anItem.price
    int oldBalance = this.myAccount.balance
    buyer.payMe(myAccount,price)
    if (this.myAccount.balance == oldBalance+price)
       this.send(buyer,anItem)
    else
       buyer.tell("you have not paid me")
```

External call

Challenge_2/4: …. Revisiting the Shop exam

External call

```
class Shop

  fld myAccount : Account
  fld inventory : Inventory

  void buy(buyer: Object, anItem: Item)
    int price = anItem.price
    int oldBalance = this.myAccount.balance
    buyer.payMe(myAccount,price)
    if (this.myAccount.balance == oldBalance+price)
       this.send(buyer,anItem)
    else
       buyer.tell("you have not paid me")
```

`Shop` part of `Banking` module;
`buy` a public method of `Shop`.

Challenge_2/4: …. Revisiting the Shop exam

External call

```
class Shop

   fld myAccount : Account
   fld inventory : Inventory

   void buy(buyer: Object, anItem: Item)
     int price = anItem.price
     int oldBalance = this.myAccount.balance
     buyer.payMe(myAccount,price)
     if (this.myAccount.balance == oldBalance+price)
        this.send(buyer,anItem)
     else
        buyer.tell("you have not paid me")
```

`Shop` part of `Banking` module;
`buy` a public method of `Shop`.

`buy` satisfies

PRE: `this.myAccount.passdw` **`prt-frm`** `buyer && this.myAccount.balance==b`
POST: `this.myAccount.balance >= b`

Challenge_2/4: …. Revisiting the Shop exam

External call

```
class Shop

  fld myAccount : Account
  fld inventory : Inventory

  void buy(buyer: Object, anItem: Item)
    int price = anItem.price
    int oldBalance = this.myAccount.balance
    buyer.payMe(myAccount,price)
    if (this.myAccount.balance == oldBalance+price)
       this.send(buyer,anItem)
    else
       buyer.tell("you have not paid me")
```

`Shop` part of `Banking` module;
`buy` a public method of `Shop`.

`buy` satisfies **implicitly**

PRE: `this.myAccount.passdw` **`prt-frm`** `buyer && this.myAccount.balance==b`
POST: `this.myAccount.balance >= b`

This spec *derivable* from the `Banking` spec

Challenge_2/4: …. Revisiting the Shop exam

External call

```
class Shop

   fld myAccount : Account
   fld inventory : Inventory

   void buy(buyer: Object, anItem: Item)
     int price = anItem.price
     int oldBalance = this.myAccount.balance
     buyer.payMe(myAccount,price)
     if (this.myAccount.balance == oldBalance+price)
        this.send(buyer,anItem)
     else
        buyer.tell("you have not paid me")
```

`Shop` part of `Banking` module;
`buy` a public method of `Shop`.

`buy` possible spec

PRE: `myAccount.balance=b && this.inventory = list`
POST: `myAccount.balance=b+anItem.price`
`-> this.inventory = list\anItem`

`buy` satisfies **implicitly**

PRE: `this.myAccount.passdw` **`prt-frm`** `buyer && this.myAccount.balance==b`
POST: `this.myAccount.balance >= b`

This spec *derivable* from the `Banking` spec

Challenge_2/4: …. Revisiting the Shop exam

External call

```
class Shop

  fld myAccount : Account
  fld inventory : Inventory

  void buy(buyer: Object, anItem: Item)
    int price = anItem.price
    int oldBalance = this.myAccount.balance
    buyer.payMe(myAccount,price)
    if (this.myAccount.balance == oldBalance+price)
       this.send(buyer,anItem)
    else
       buyer.tell("you have not paid me")
```

🤔🤔 this spec is implied 🤔🤔

Of course, we have to prove that `buy` preserves `Banking’s` spec (as per rule Module_Well_Fomed)

`Shop` part of `Banking` module;
`buy` a public method of `Shop`.

`buy` possible spec

PRE: `myAccount.balance=b && this.inventory = list`
POST: `myAccount.balance=b+anItem.price`
`—> this.inventory = list\anItem`

`buy` satisfies **implicitly**

PRE: `this.myAccount.passdw` **`prt-frm`** `buyer && this.myAccount.balance==b`
POST: `this.myAccount.balance >= b`

This spec *derivable* from the `Banking` spec

Challenge_2/4: …. Revisiting the Shop exam

External call

```
class Shop

  fld myAccount : Account
  fld inventory : Inventory

  void buy(buyer: Object, anItem: Item)
    int price = anItem.price
    int oldBalance = this.myAccount.balance
    buyer.payMe(myAccount,price)
    if (this.myAccount.balance == oldBalance+price)
       this.send(buyer,anItem)
    else
       buyer.tell("you have not paid me")
```

🤔🤔 this spec is implied 🤔🤔

Of course, we have to prove that `buy` preserves `Banking’s` spec (as per rule Module_Well_Fomed)

`Shop` part of `Banking` module;
`buy` a public method of `Shop`.

`buy` possible spec
PRE: `myAccount.balance=b && this.inventory = list`
POST: `myAccount.balance=b+anItem.price`
`–> this.inventory = list\anItem`

`buy` satisfies **implicitly**
PRE: `this.myAccount.passdw` **`prt-frm`** `buyer && this.myAccount.balance==b`
POST: `this.myAccount.balance >= b`

This spec *derivable* from the `Banking` spec

Challenge_3: The inference system should be algorithmic

Happy!

Convinced!

Surprised!

Summary

- Distinction between external/internal objects

- Specifications talk about necessary conditions for effect:

  ∀ x: Object (| **denial of capability** ∧ A |) (| A |)

  means that **capability** is needed in order to invalidate A

- **prt** x: expresses that capability x is protected from external objects

- Design "Fineties"
  - **prt** x only protects from *locally-relevant* objects;
  - ∀ x: ..(|..|)(|…|) tallks about *globally-relevant* objects
  - "future is shallow"

- API-agrnositc spec, "Algorithmic" inference system system, open calls

- Hand-written soundness and adherence proof

Happy!

Convinced:

Object capabilities are about necessary conditions for effects caused by external objects.

Surprised!

Happy!

Convinced!

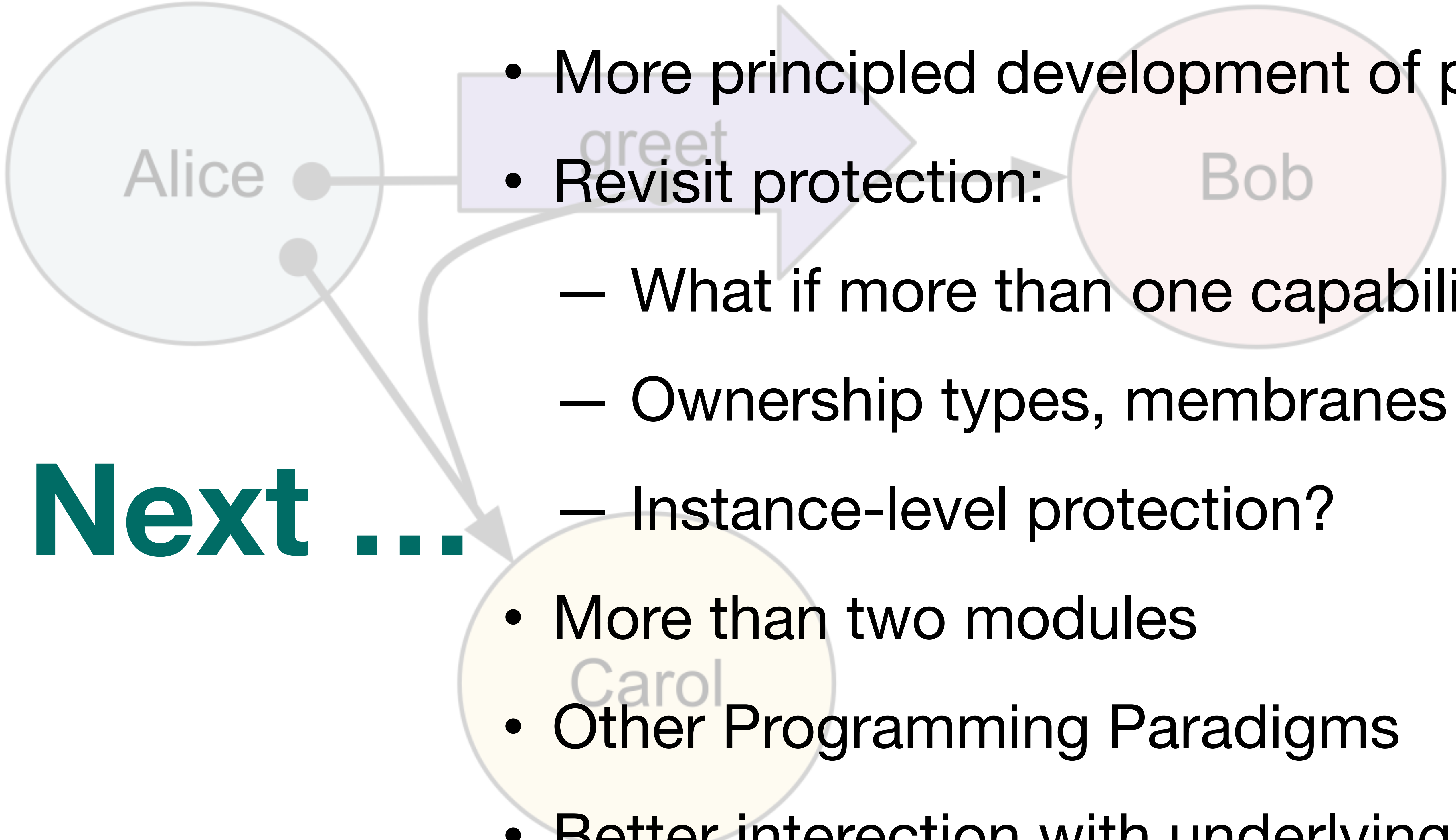
Surprised:

- talk about *necessary* conditions, but reason with *sufficient* conditions.

- No need for temporal logic specifications.

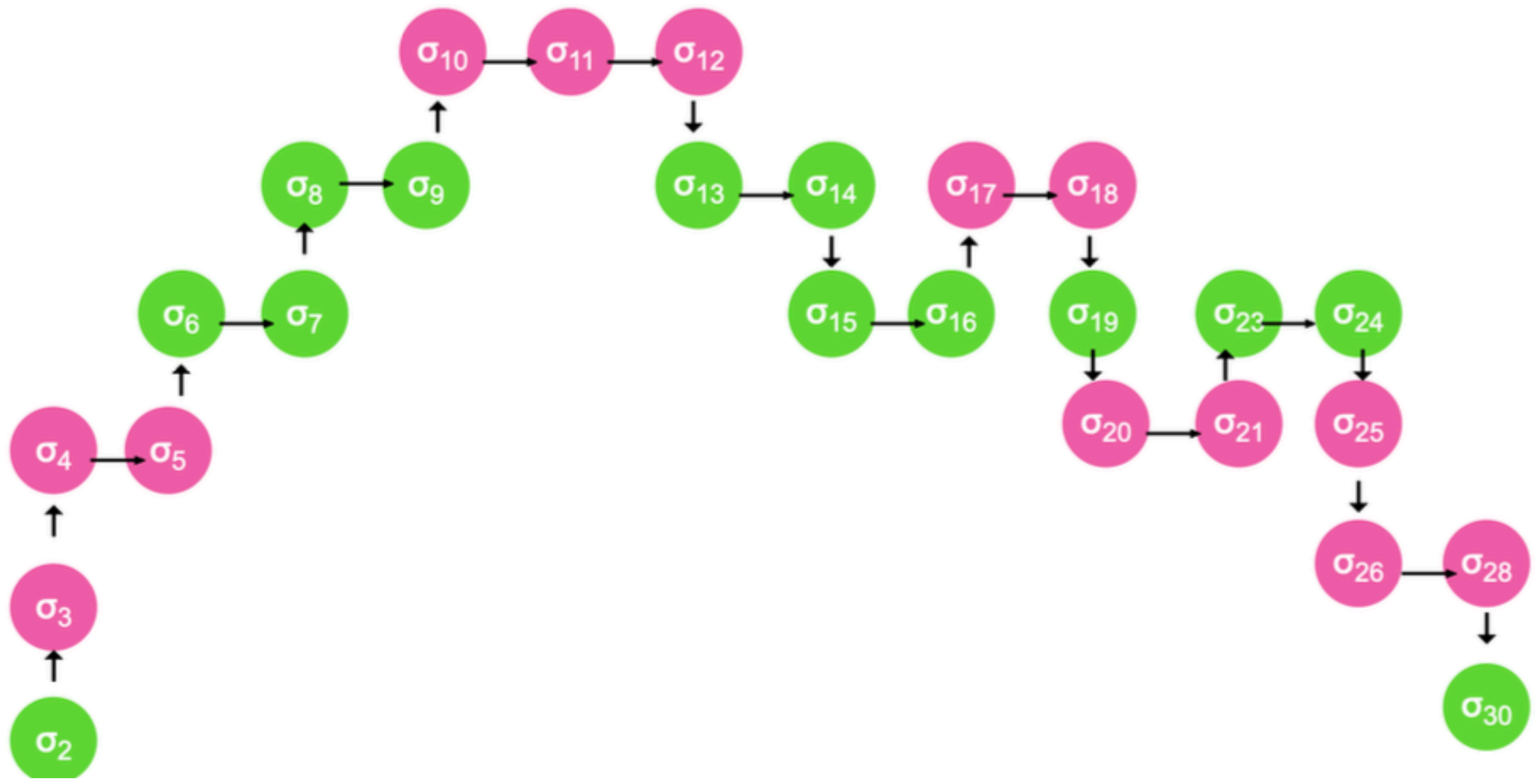
- Hoare-logic extension

Next ...

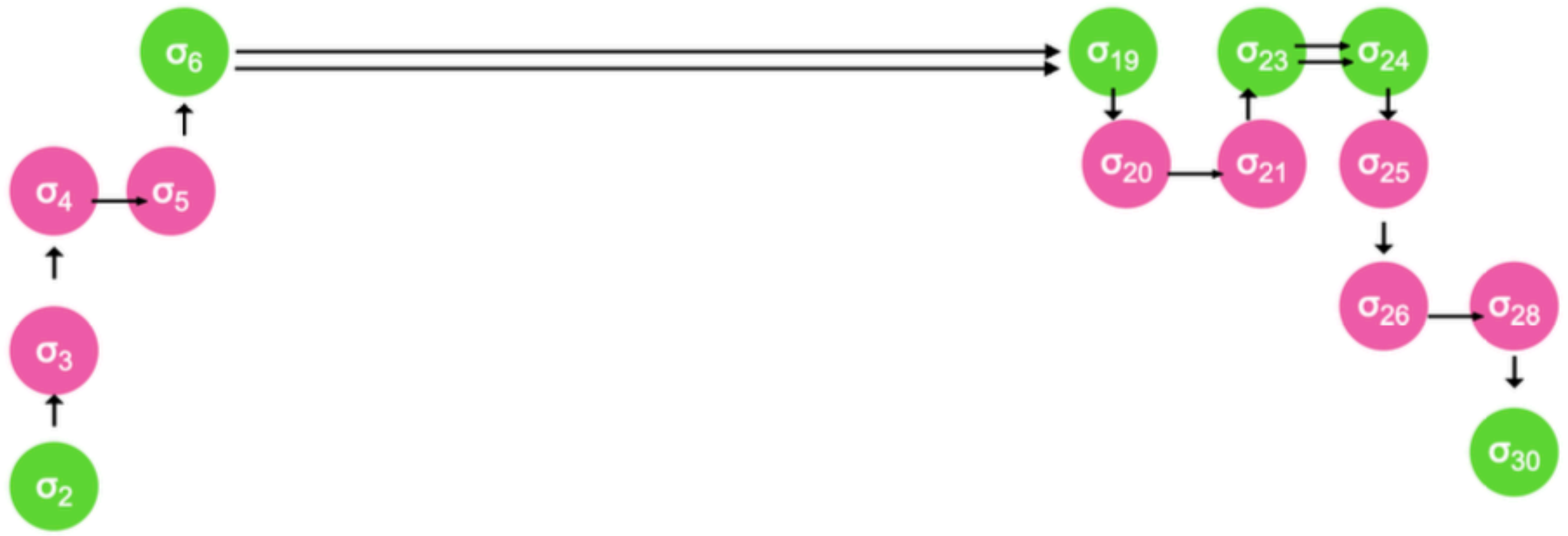


- Smooth the edges
- Mechanize proofs
- Completeness?
- Adversarial logic
- More principled development of ptr-Hoare rules
- Revisit protection:
  - — What if more than one capability for an effect?
  - — Ownership types, membranes etc?
  - — Instance-level protection?
- More than two modules
- Other Programming Paradigms
- Better interection with underlying Hoare logics, esp. “modifies” clauses, *
- Parametric with language and Hoare Logic
- Inference of Classic Specs given Invariants
- Tool

the original execution:



the summarised execution:


σ6
σ19
σ23
σ24
σ4
σ5
σ20
σ21
σ25
σ26
σ28
σ3
σ30
σ2