Stuck macros

deterministically interleaving
macro-expansion and type-checking

presented at C◦mp◦se NYC 2019 by Samuel Gélineau

Stuck macros

deterministically interleaving
macro-expansion and type-checking

presented at C◦mp◦se NYC 2019 by Samuel Gélineau

Type-aware ~~Stuck~~ macros

deterministically interleaving
macro-expansion and type-checking

presented at C◦mp◦se NYC 2019 by Samuel Gélineau

Type-aware ~~Stuck~~ macros

non-deterministically interleaving
macro-expansion and type-checking

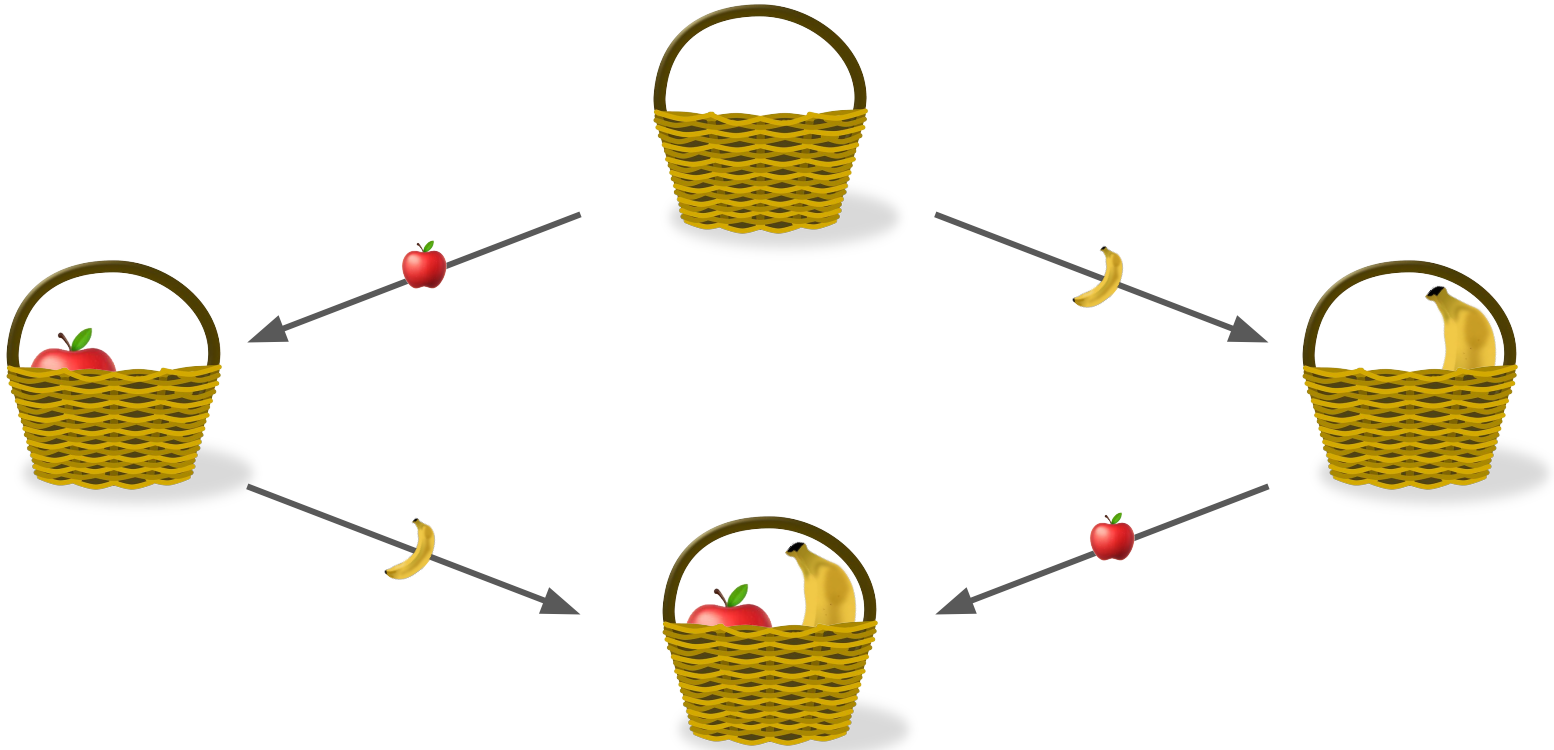
presented at C◦mp◦se NYC 2019 by Samuel Gélineau

Type-aware Stuck macros

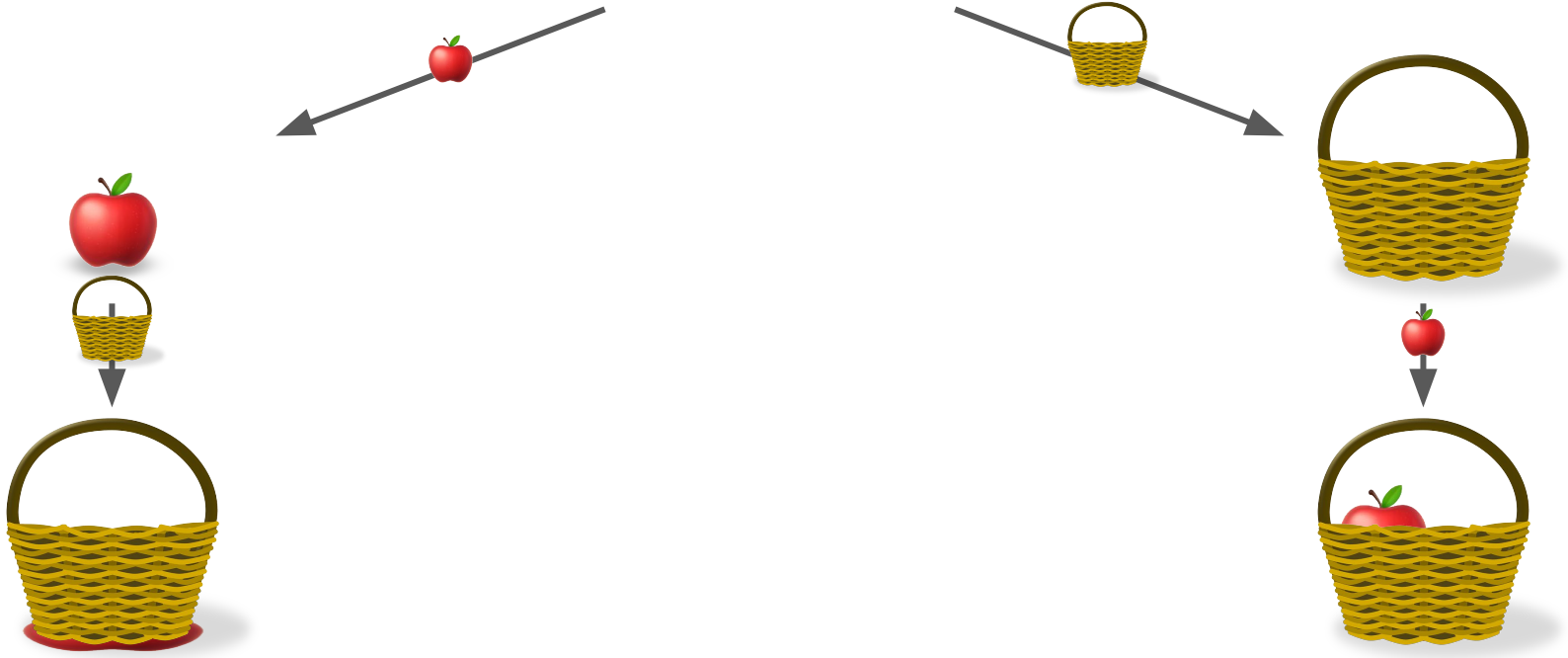
deterministically interleaving macro-expansion and type-checking

presented at C◦mp◦se NYC 2019 by Samuel Gélineau

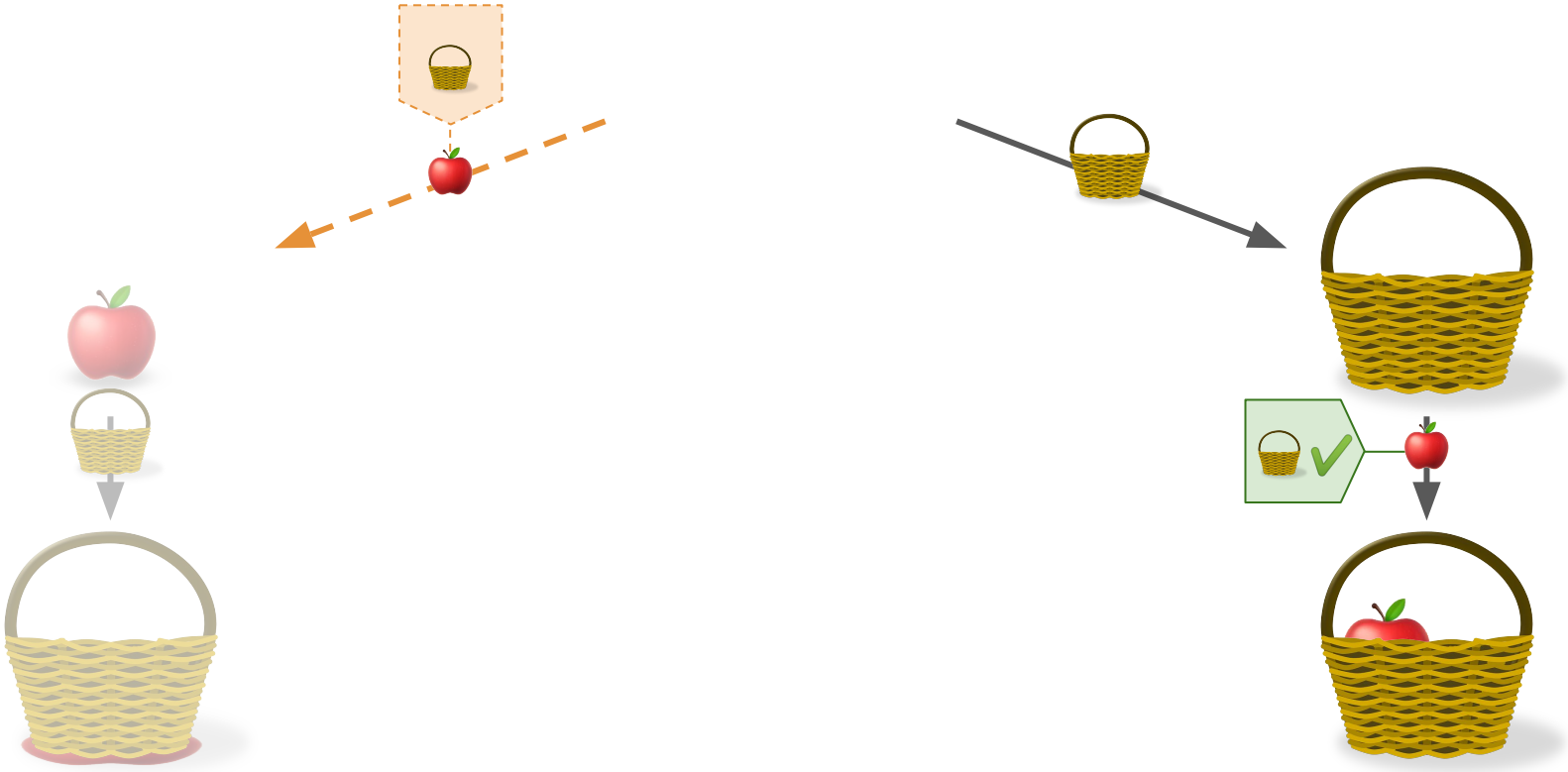
confluent



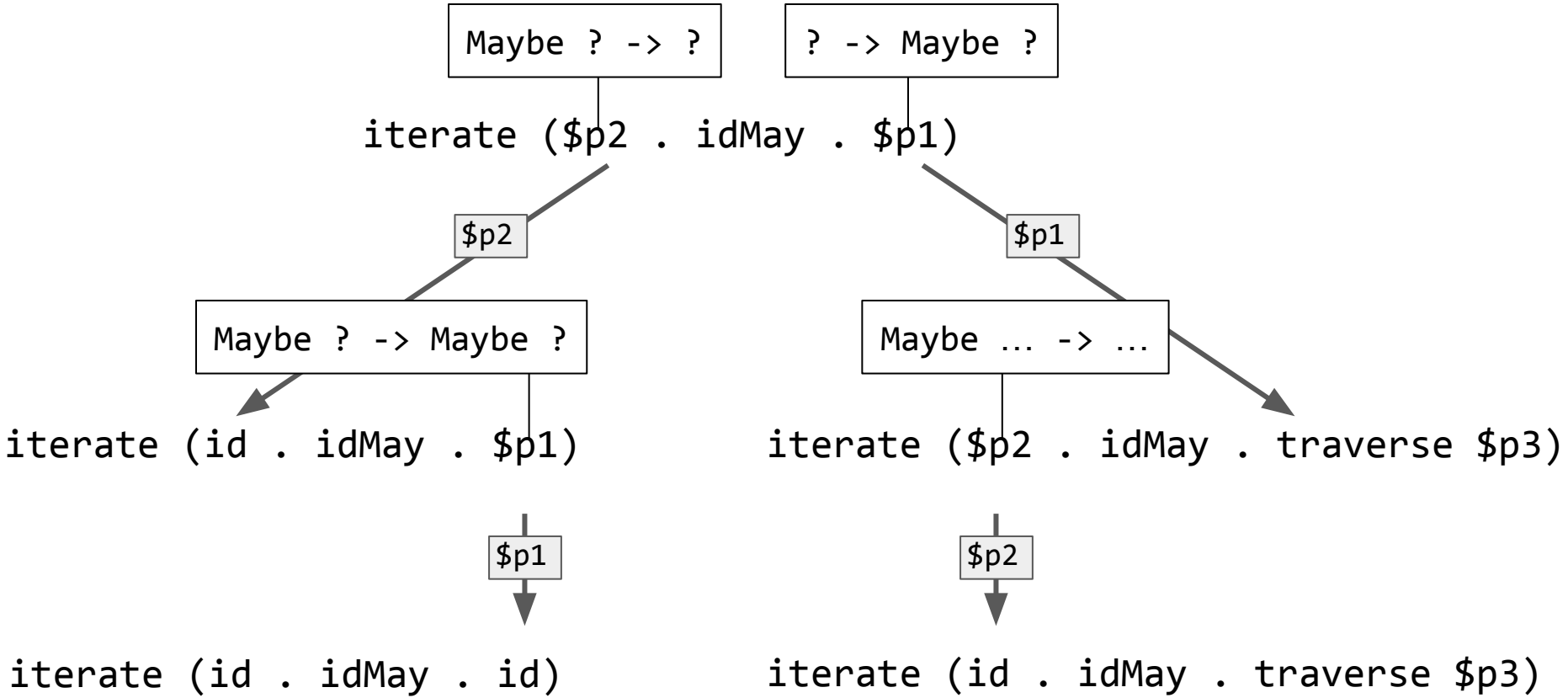
not confluent



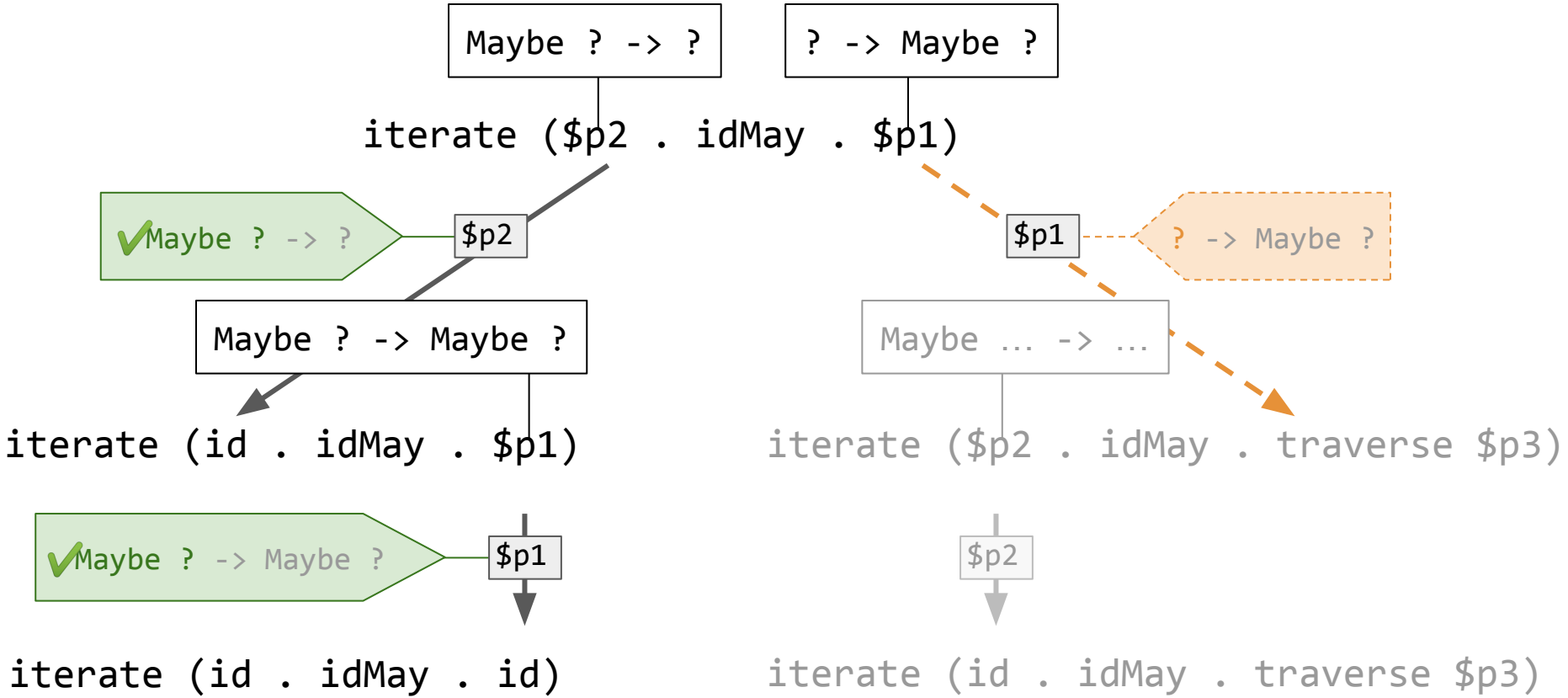
stuck fruits are confluent



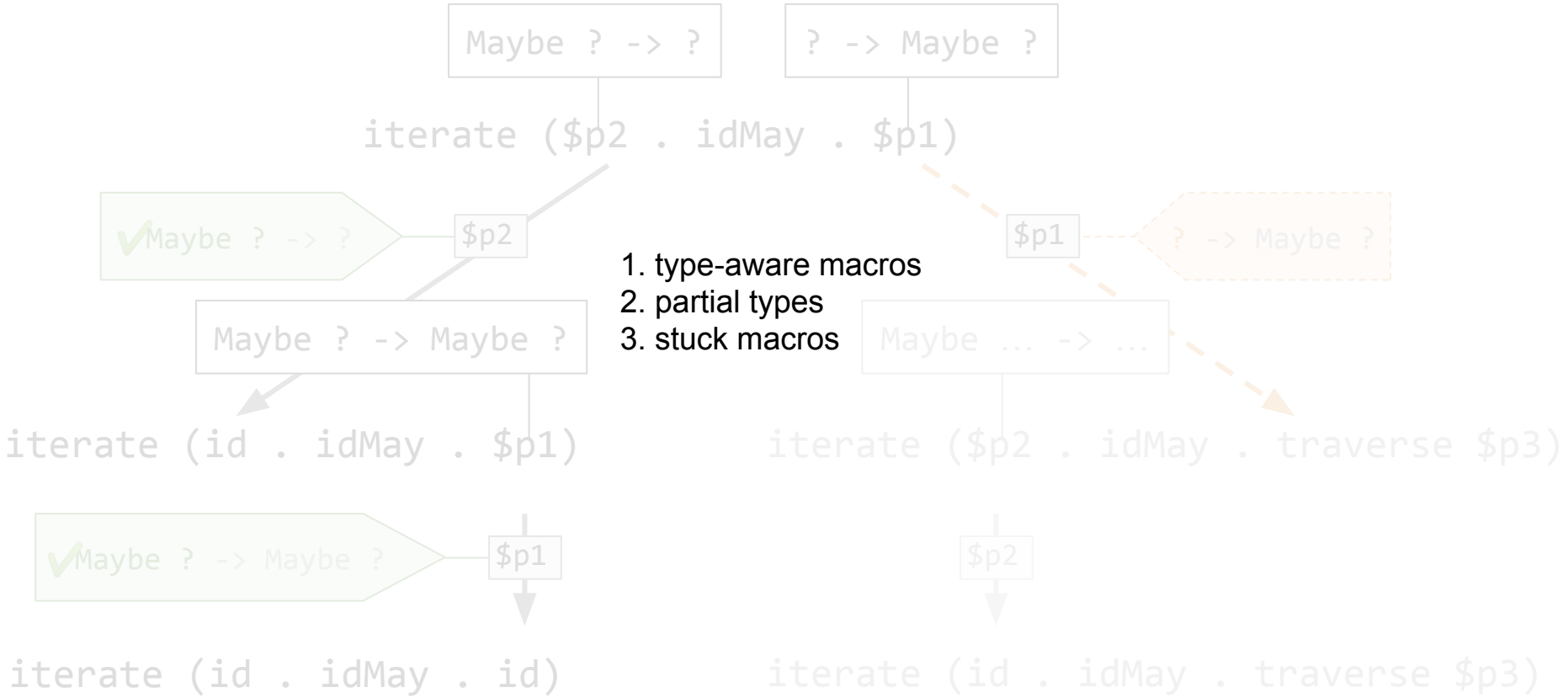
type-aware macros are not confluent



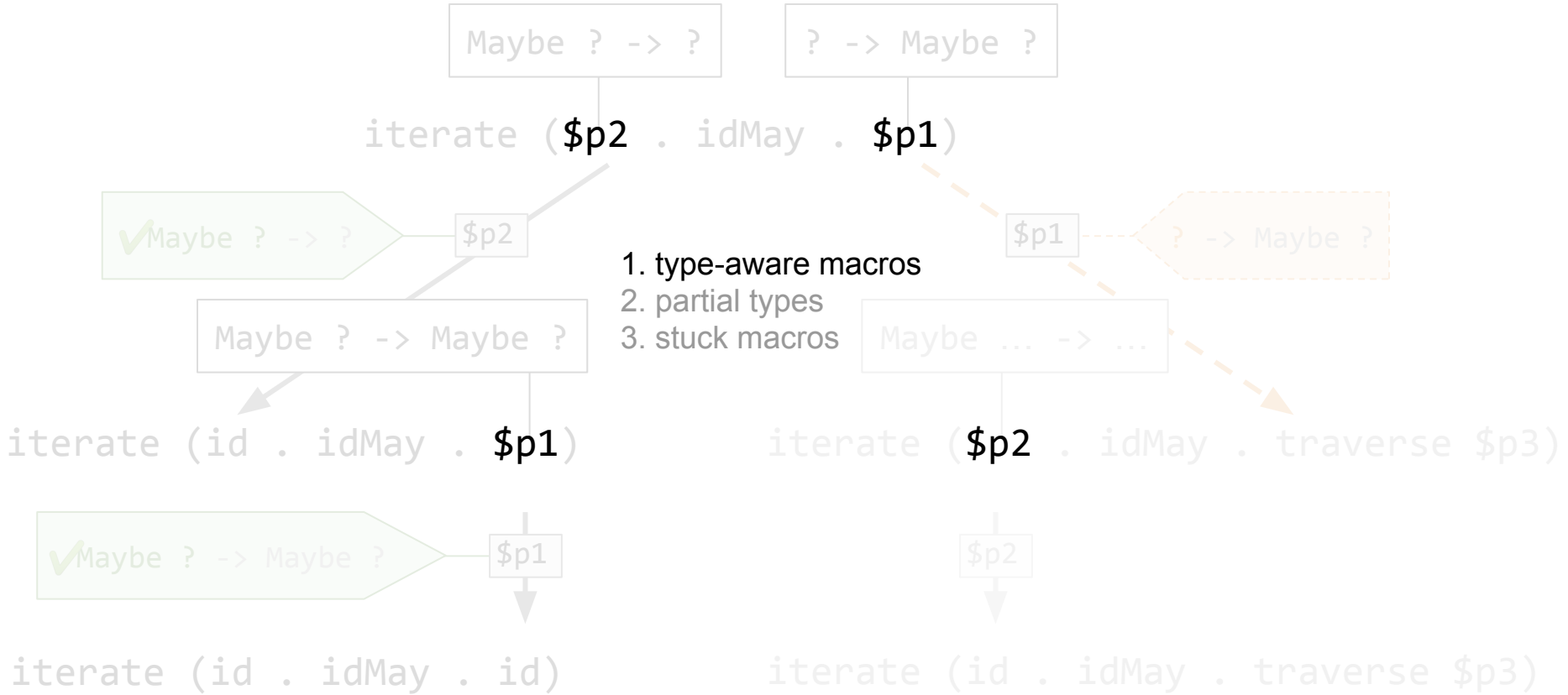
stuck macros are confluent



outline


Maybe ? -> ?
? -> Maybe ?
iterate ($p2 . idMay . $p1)
Maybe ? -> ?
$p2
$p1
? -> Maybe ?
1. type-aware macros
2. partial types
3. stuck macros
Maybe ? -> Maybe ?
Maybe ... -> ...
iterate (id . idMay . $p1)
iterate ($p2 . idMay . traverse $p3)
Maybe ? -> Maybe ?
$p1
$p2
iterate (id . idMay . id)
iterate (id . idMay . traverse $p3)

outline


Maybe ? -> ?
? -> Maybe ?
iterate ($p2 . idMay . $p1)
✔Maybe ? -> ?
$p2
$p1
? -> Maybe ?
1. type-aware macros
2. partial types
3. stuck macros
Maybe ? -> Maybe ?
Maybe ... -> ...
iterate (id . idMay . $p1)
iterate ($p2 . idMay . traverse $p3)
✔Maybe ? -> Maybe ?
$p1
$p2
iterate (id . idMay . id)
iterate (id . idMay . traverse $p3)

type-aware macros

```
pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

type-aware macros

```
id :: a -> a
traverse :: (a -> Maybe b) -> [a] -> Maybe [b]
```

```
pullMaybe0 ::   Maybe a    -> Maybe    a
pullMaybe0 = id

pullMaybe1 ::  [Maybe a]   -> Maybe   [a]
pullMaybe1 = traverse id

pullMaybe2 :: [[Maybe a]]  -> Maybe  [[a]]
pullMaybe2 = traverse (traverse id)

pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

type-aware macros

```
id :: a -> a
traverse :: (a -> Maybe b) -> [a] -> Maybe [b]
```

```
pullMaybe0 ::   Maybe a    -> Maybe    a
pullMaybe0 = id

pullMaybe1 ::  [Maybe a]   -> Maybe   [a]
pullMaybe1 = traverse id

pullMaybe2 :: [[Maybe a]]  -> Maybe  [[a]]
pullMaybe2 = traverse (traverse id)

pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

type-aware macros

```
id :: a -> a
traverse :: (a -> Maybe b) -> [a] -> Maybe [b]
```

```
pullMaybe0 ::   Maybe a    -> Maybe    a
pullMaybe0 = id

pullMaybe1 ::  [Maybe a]   -> Maybe   [a]
pullMaybe1 = traverse id

pullMaybe2 :: [[Maybe a]]  -> Maybe  [[a]]
pullMaybe2 = traverse (traverse id)

pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

type-aware macros

```
id :: a -> a
traverse :: (a -> Maybe b) -> [a] -> Maybe [b]
```

```
pullMaybe0 ::   Maybe a    -> Maybe    a
pullMaybe0 = id

pullMaybe1 ::  [Maybe a]   -> Maybe   [a]
pullMaybe1 = traverse id

pullMaybe2 :: [[Maybe a]]  -> Maybe  [[a]]
pullMaybe2 = traverse (traverse id)

pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

type-aware macros

```
id :: a -> a
traverse :: (a -> Maybe b) -> [a] -> Maybe [b]
```

```
pullMaybe0 ::   Maybe a    -> Maybe    a
pullMaybe0 = id

pullMaybe1 ::  [Maybe a]   -> Maybe   [a]
pullMaybe1 = traverse id

pullMaybe2 :: [[Maybe a]]  -> Maybe  [[a]]
pullMaybe2 = traverse (traverse id)

pullMaybe :: Q Exp
pullMaybe = do t <- getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

if you only remember one thing!

type-aware macros

```
id :: a -> a
traverse :: (a -> Maybe b) -> [a] -> Maybe [b]
```

```
pullMaybe0 ::   Maybe a    -> Maybe    a
pullMaybe0 = id

pullMaybe1 ::  [Maybe a]   -> Maybe   [a]
pullMaybe1 = traverse id

pullMaybe2 :: [[Maybe a]]  -> Maybe  [[a]]
pullMaybe2 = traverse (traverse id)

pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

type-aware macros

```
id :: a -> a
traverse :: (a -> Maybe b) -> [a] -> Maybe [b]
```

```
pullMaybe0 ::   Maybe a    -> Maybe    a
pullMaybe0 = $pullMaybe

pullMaybe1 ::  [Maybe a]   -> Maybe   [a]
pullMaybe1 = $pullMaybe

pullMaybe2 :: [[Maybe a]]  -> Maybe  [[a]]
pullMaybe2 = $pullMaybe

pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

type-aware macros

```
id :: a -> a
traverse :: (a -> Maybe b) -> [a] -> Maybe [b]
```

```
pullMaybe0 ::   Maybe a    -> Maybe    a
pullMaybe0 = $pullMaybe

pullMaybe1 ::  [Maybe a]   -> Maybe   [a]
pullMaybe1 = $pullMaybe

pullMaybe2 :: [[Maybe a]]  -> Maybe  [[a]]
pullMaybe2 = $pullMaybe

pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

type-aware macros

```
id :: a -> a
traverse :: (a -> Maybe b) -> [a] -> Maybe [b]
```

```
pullMaybe0 ::   Maybe a    -> Maybe    a
pullMaybe0 = $pullMaybe

pullMaybe1 ::  [Maybe a]   -> Maybe   [a]
pullMaybe1 = $pullMaybe

pullMaybe2 :: [[Maybe a]]  -> Maybe  [[a]]
pullMaybe2 = $pullMaybe

pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

```
getExpectedType :: Q Type

data Type = Arr Type Type
          | Maybe Type
          | ...
```

type-aware macros

```
id :: a -> a
traverse :: (a -> Maybe b) -> [a] -> Maybe [b]
```

```
pullMaybe0 ::   Maybe a    -> Maybe    a
pullMaybe0 = $pullMaybe

pullMaybe1 ::  [Maybe a]   -> Maybe   [a]
pullMaybe1 = $pullMaybe

pullMaybe2 :: [[Maybe a]]  -> Maybe  [[a]]
pullMaybe2 = $pullMaybe

pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

```
getExpectedType :: Q Type

data Type = Arr Type Type
          | Maybe Type
          | ...
```

type-aware macros

```
id :: a -> a
traverse :: (a -> Maybe b) -> [a] -> Maybe [b]
```

```
pullMaybe0 ::   Maybe a     -> Maybe     a
pullMaybe0 = $pullMaybe

pullMaybe1 ::  [Maybe a]    -> Maybe    [a]
pullMaybe1 = $pullMaybe

pullMaybe2 :: [[Maybe a]]   -> Maybe  [[a]]
pullMaybe2 = $pullMaybe

pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

```
getExpectedType :: Q Type

data Type = Arr Type Type
          | Maybe Type
          | ...
```

type-aware macros

```
id :: a -> a
traverse :: (a -> Maybe b) -> [a] -> Maybe [b]
```

```
pullMaybe0 ::   Maybe a    -> Maybe    a
pullMaybe0 = id

pullMaybe1 ::  [Maybe a]   -> Maybe   [a]
pullMaybe1 = traverse $pullMaybe

pullMaybe2 :: [[Maybe a]]  -> Maybe  [[a]]
pullMaybe2 = traverse $pullMaybe

pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

```
getExpectedType :: Q Type

data Type = Arr Type Type
          | Maybe Type
          | ...
```

type-aware macros

```
id :: a -> a
traverse :: (a -> Maybe b) -> [a] -> Maybe [b]
```

```
pullMaybe0 ::   Maybe a    -> Maybe    a
pullMaybe0 = id

pullMaybe1 ::  [Maybe a]   -> Maybe   [a]
pullMaybe1 = traverse $pullMaybe

pullMaybe2 :: [[Maybe a]]  -> Maybe  [[a]]
pullMaybe2 = traverse $pullMaybe

pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

```
getExpectedType :: Q Type

data Type = Arr Type Type
          | Maybe Type
          | ...
```

type-aware macros

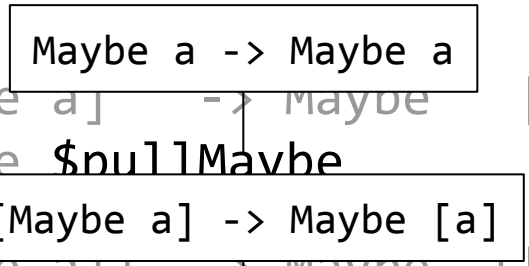
```
id :: a -> a
traverse :: (a -> Maybe b) -> [a] -> Maybe [b]
```

```
pullMaybe0 ::   Maybe a    -> Maybe    a
pullMaybe0 = id

pullMaybe1 :: [Maybe a]    -> Maybe   [a]
pullMaybe1 = traverse $pullMaybe

pullMaybe2 :: [[Maybe a]]  -> Maybe  [[a]]
pullMaybe2 = traverse $pullMaybe

pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```




```
getExpectedType :: Q Type

data Type = Arr Type Type
          | Maybe Type
          | ...
```

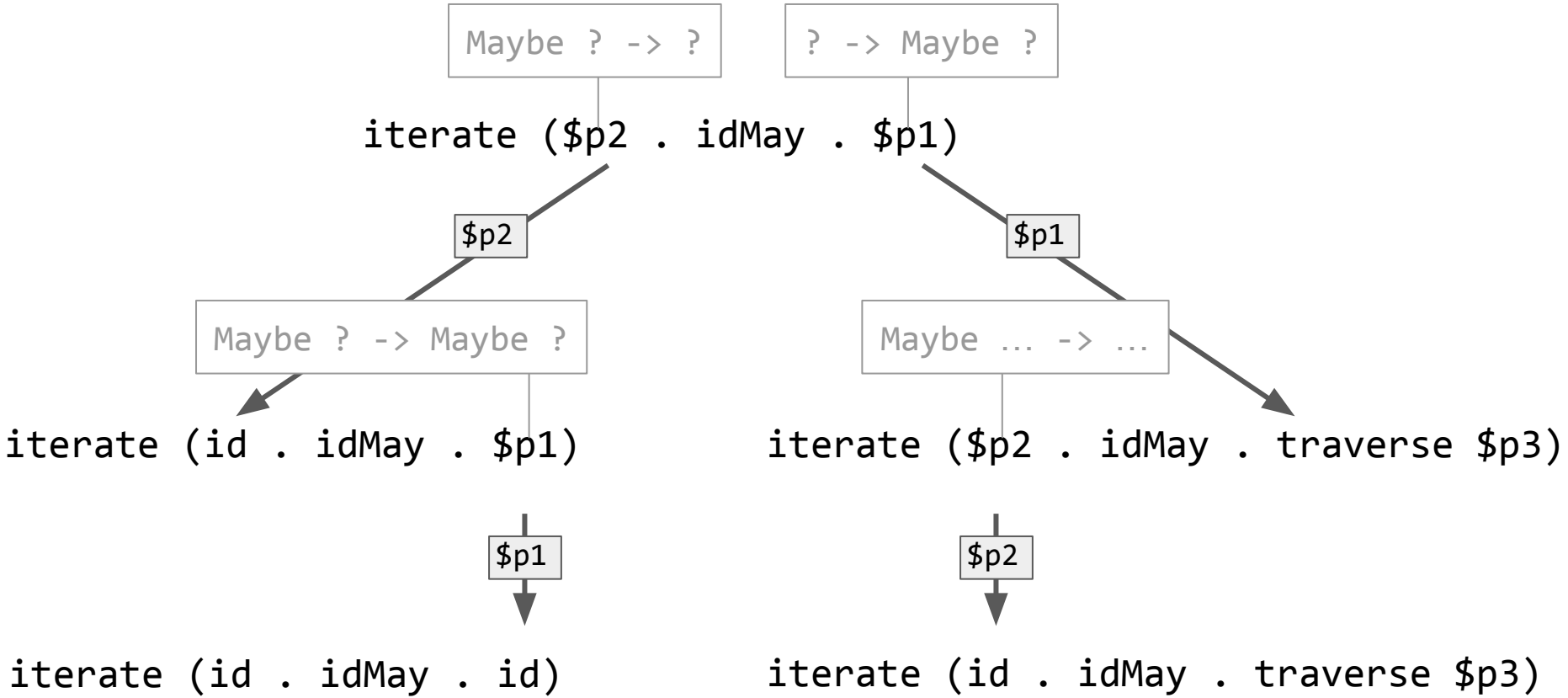
Stuck macros

deterministically interleaving
macro-expansion and type-checking

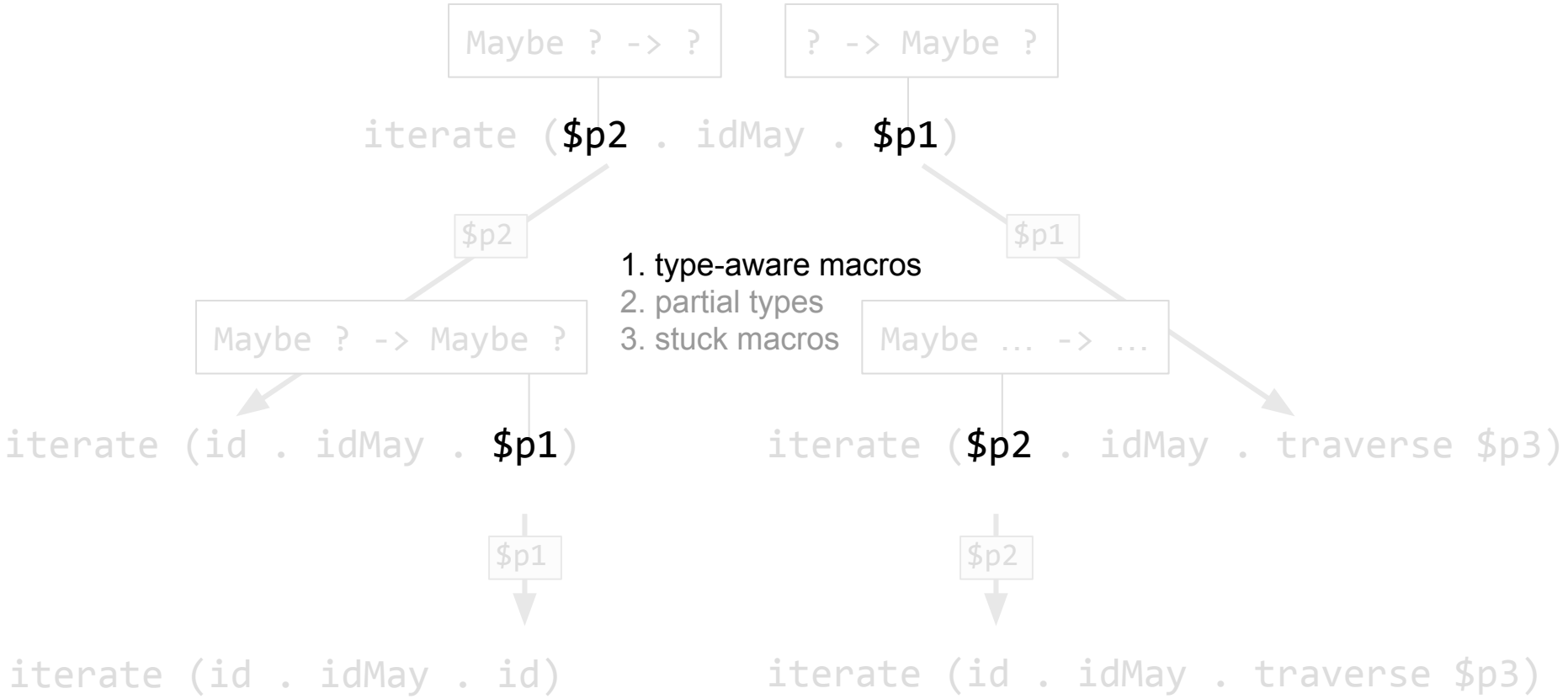
presented at C◦mp◦se NYC 2019 by Samuel Gélineau

type-aware macros are not confluent

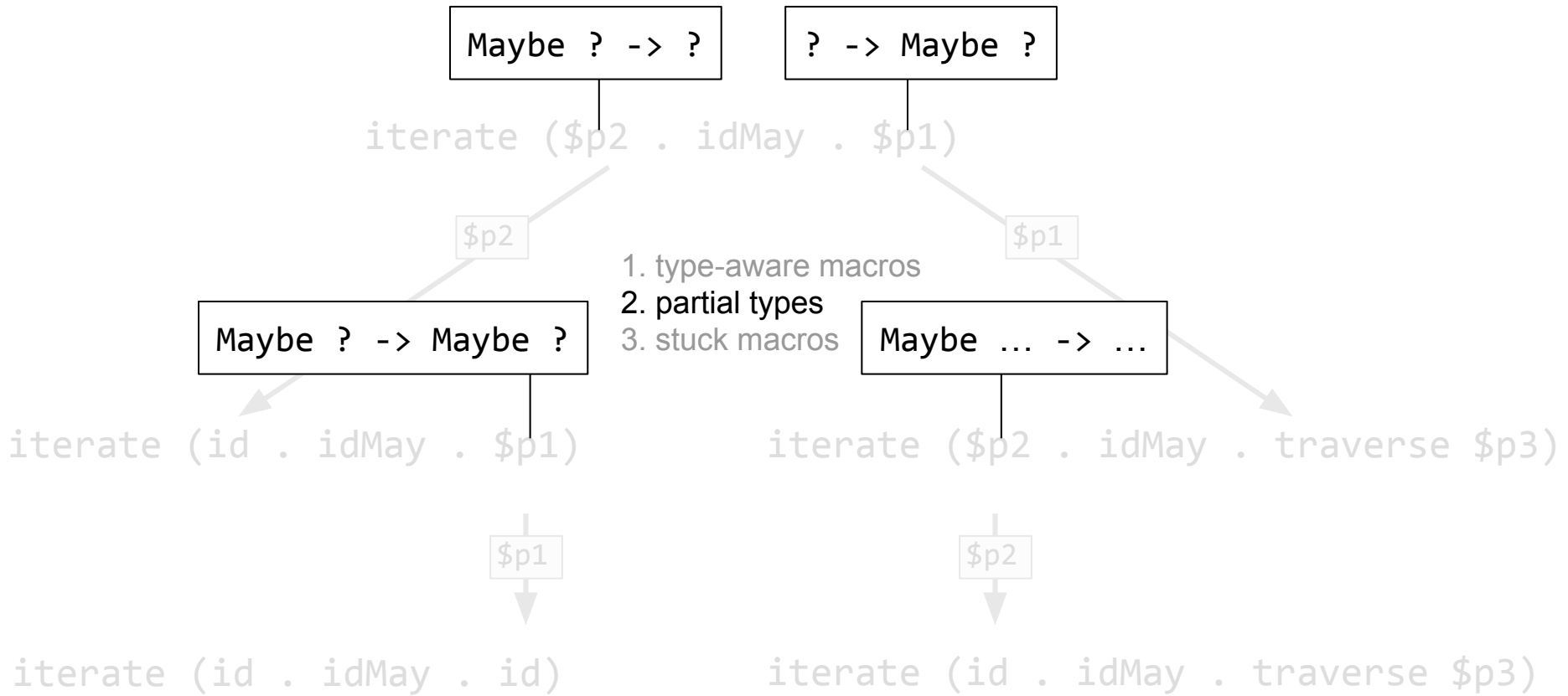
```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



outline


p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
idMay :: Maybe a -> Maybe a
Maybe ? -> ?
? -> Maybe ?
iterate ($p2 . idMay . $p1)
$p2
$p1
1. type-aware macros
2. partial types
3. stuck macros
Maybe ? -> Maybe ?
Maybe … -> …
iterate (id . idMay . $p1)
iterate ($p2 . idMay . traverse $p3)
$p1
$p2
iterate (id . idMay . id)
iterate (id . idMay . traverse $p3)

outline


Maybe ? -> ?
? -> Maybe ?
iterate ($p2 . idMay . $p1)
$p2
$p1
1. type-aware macros
2. partial types
3. stuck macros
Maybe ? -> Maybe ?
Maybe ... -> ...
iterate (id . idMay . $p1)
iterate ($p2 . idMay . traverse $p3)
$p1
$p2
iterate (id . idMay . id)
iterate (id . idMay . traverse $p3)

A message from our non-sponsor:

Hackett

A message from
our non-sponsor:

Hackett

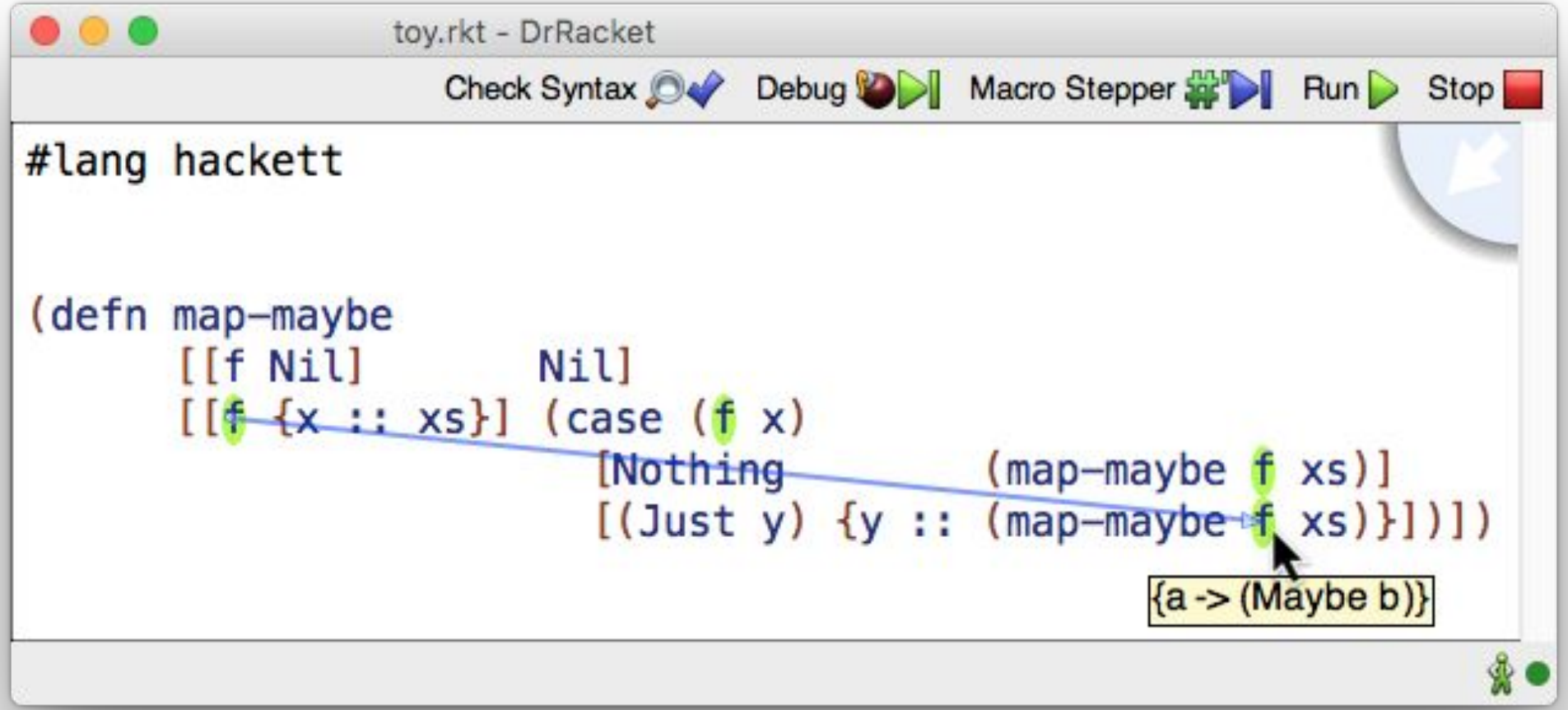
now with type-aware macros!

Hackett, a language by Alexis King (@lexi_lambda)

```
mapMaybe f []     = []
mapMaybe f (x:xs) = case f x of
                      Nothing ->     mapMaybe f xs
                      Just y  -> y : mapMaybe f xs
```

```
(defn map-maybe
  [[f Nil]       Nil]
  [[f {x :: xs}] (case (f x)
                   [Nothing        (map-maybe f xs)]
                   [(Just y) {y :: (map-maybe f xs)}])])
```

Hackett, a language by Alexis King (@lexi_lambda)

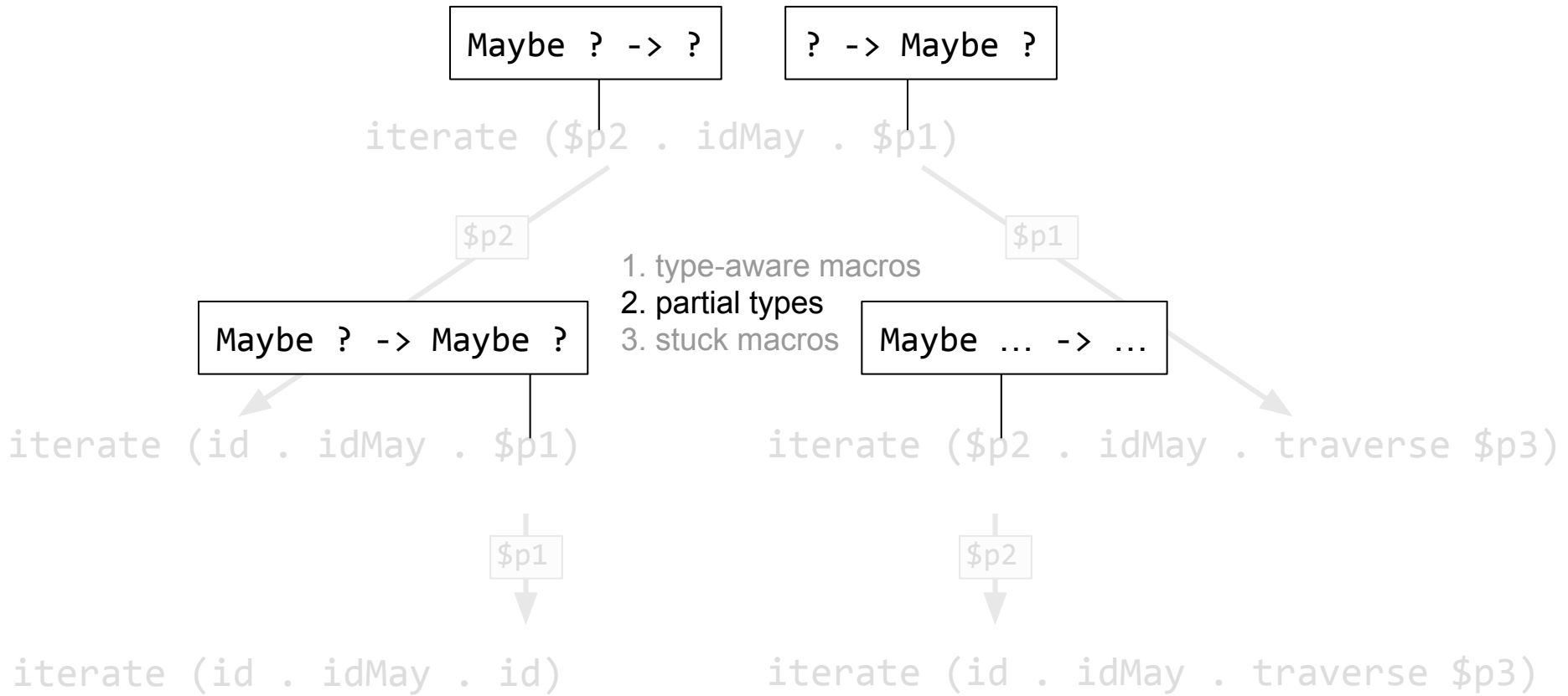


To learn more about

Hackett

visit `github.com/lexi-lambda/hackett` today!

outline


Maybe ? -> ?
? -> Maybe ?
iterate ($p2 . idMay . $p1)
$p2
$p1
1. type-aware macros
2. partial types
3. stuck macros
Maybe ? -> Maybe ?
Maybe ... -> ...
iterate (id . idMay . $p1)
iterate ($p2 . idMay . traverse $p3)
$p1
$p2
iterate (id . idMay . id)
iterate (id . idMay . traverse $p3)

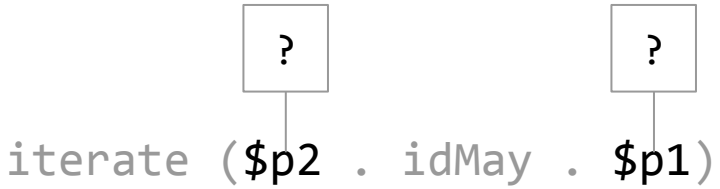
partial types

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```

```
iterate ($p2 . idMay . $p1)
```

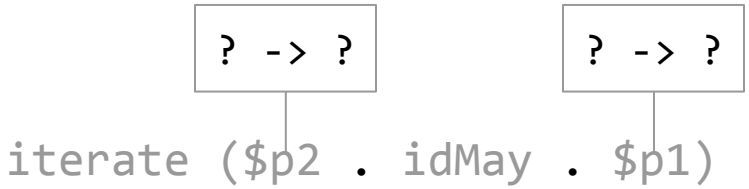
partial types

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



partial types

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



partial types

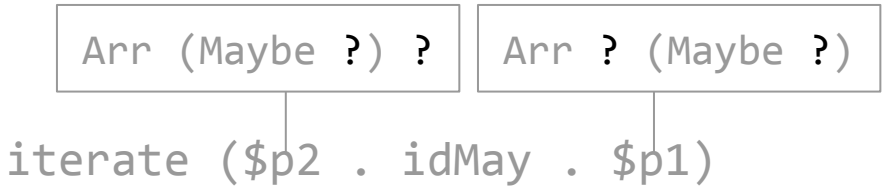
```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```

partial types

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```

partial types

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```

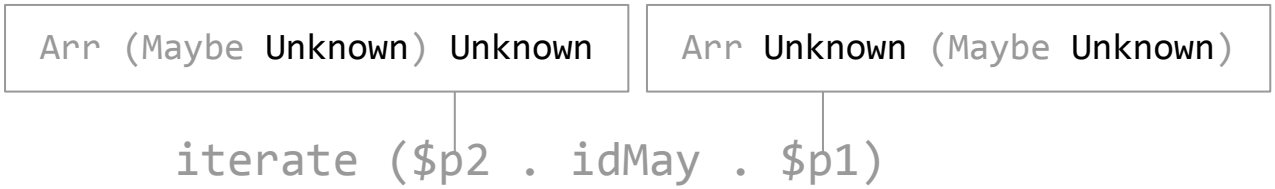



```
getExpectedType :: Q Type

data Type = Arr Type Type
          | Maybe Type
          | ...
```

partial types

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



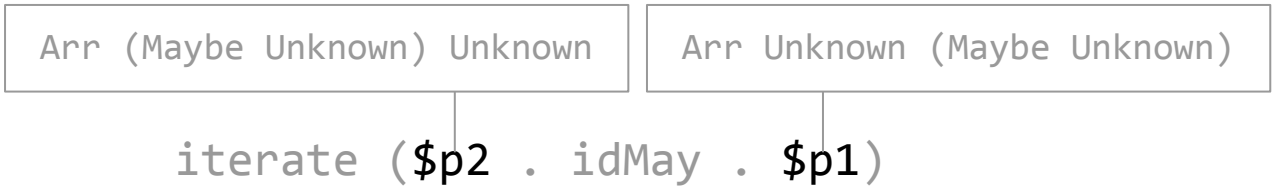

```
data PartialType = Arr PartialType PartialType
                 | Maybe PartialType
                 | ...
                 | Unknown
```

```
getExpectedType :: Q Type

data Type = Arr Type Type
          | Maybe Type
          | ...
```

partial types

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```




```
getExpectedType :: Q Type
```

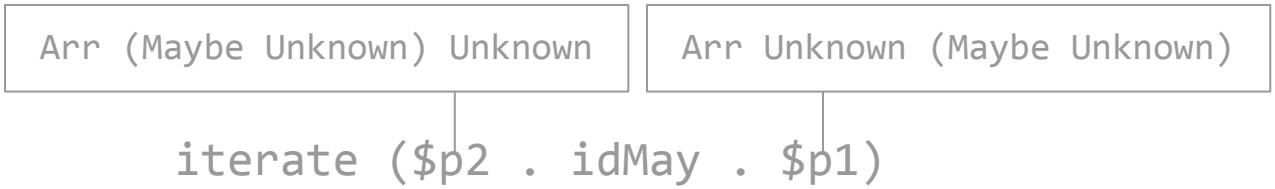
```
data PartialType = Arr PartialType PartialType
                 | Maybe PartialType
                 | ...
                 | Unknown
```

```
data Type = Arr Type Type
          | Maybe Type
          | ...
```

```
pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

partial types

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```




```
getExpectedType :: Q PartialType

data PartialType = Arr PartialType PartialType
                 | Maybe PartialType
                 | ...
                 | Unknown
```

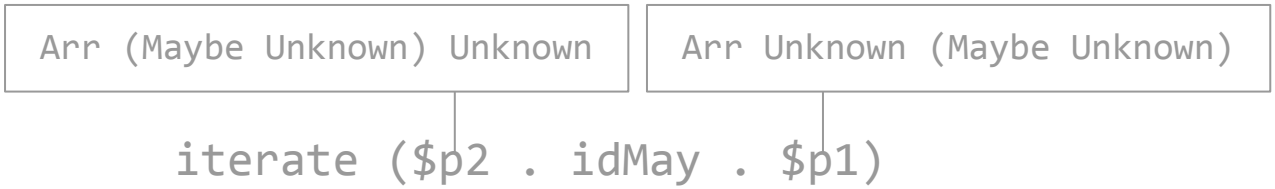
~~getExpectedType :: Q Type~~

```
data Type = Arr Type Type
          | Maybe Type
          | ...
```

```
pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

partial types

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```




```
getExpectedType :: Q PartialType

data PartialType = Arr PartialType PartialType
                 | Maybe PartialType
                 | ...
                 | Unknown
```

~~getExpectedType :: Q Type~~

```
data Type = Arr Type Type
          | Maybe Type
          | ...
```

```
pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  Arr Unknown   _ -> error "please add a type annotation"
  _               -> [| traverse $pullMaybe |]
```

partial types

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```


```
getExpectedType :: Q PartialType

data PartialType = Arr PartialType PartialType
                 | Maybe PartialType
                 | ...
                 | Unknown
```

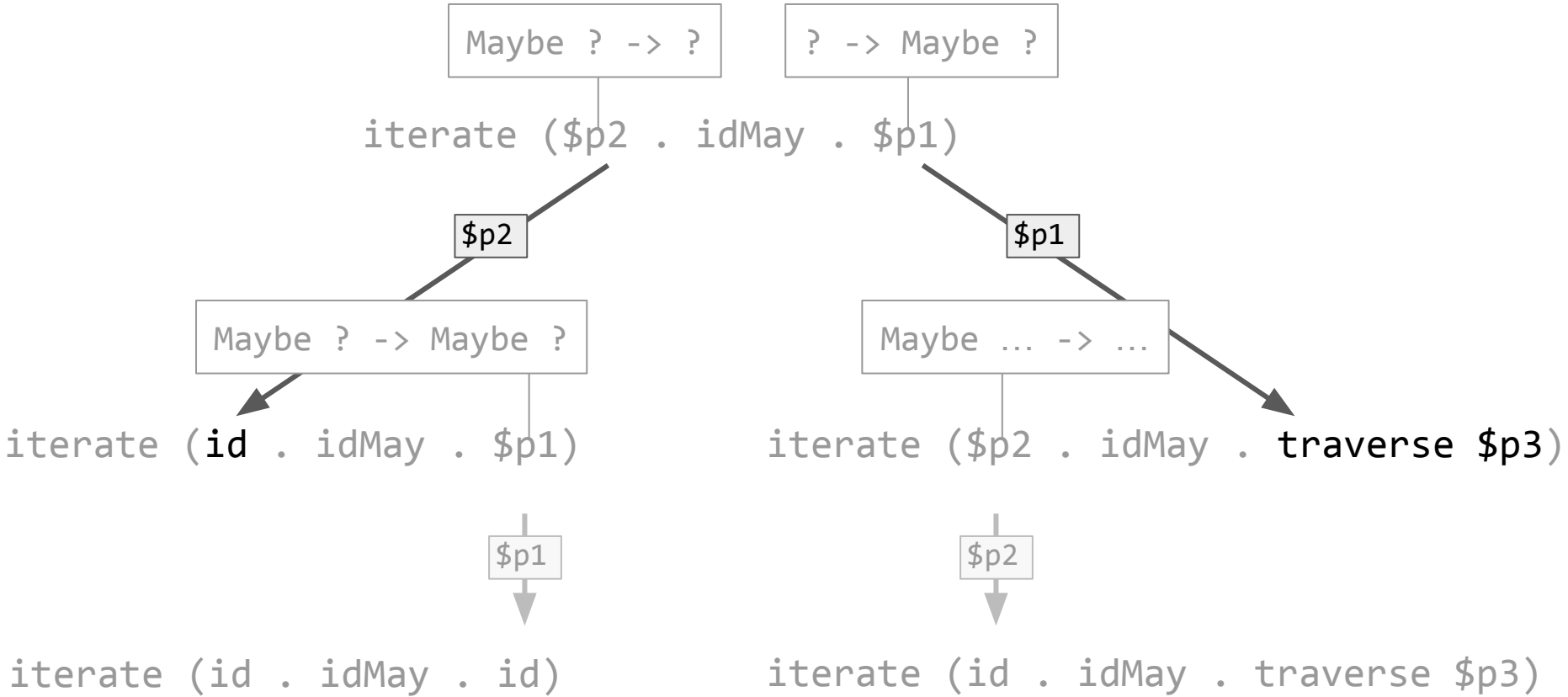
~~getExpectedType :: Q Type~~

```
data Type = Arr Type Type
          | Maybe Type
          | ...
```

```
pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |] $p2
  Arr Unknown   _ -> error "please add a type annotation"
  _               -> [| traverse $pullMaybe |] $p1
```

type-aware macros are not confluent

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



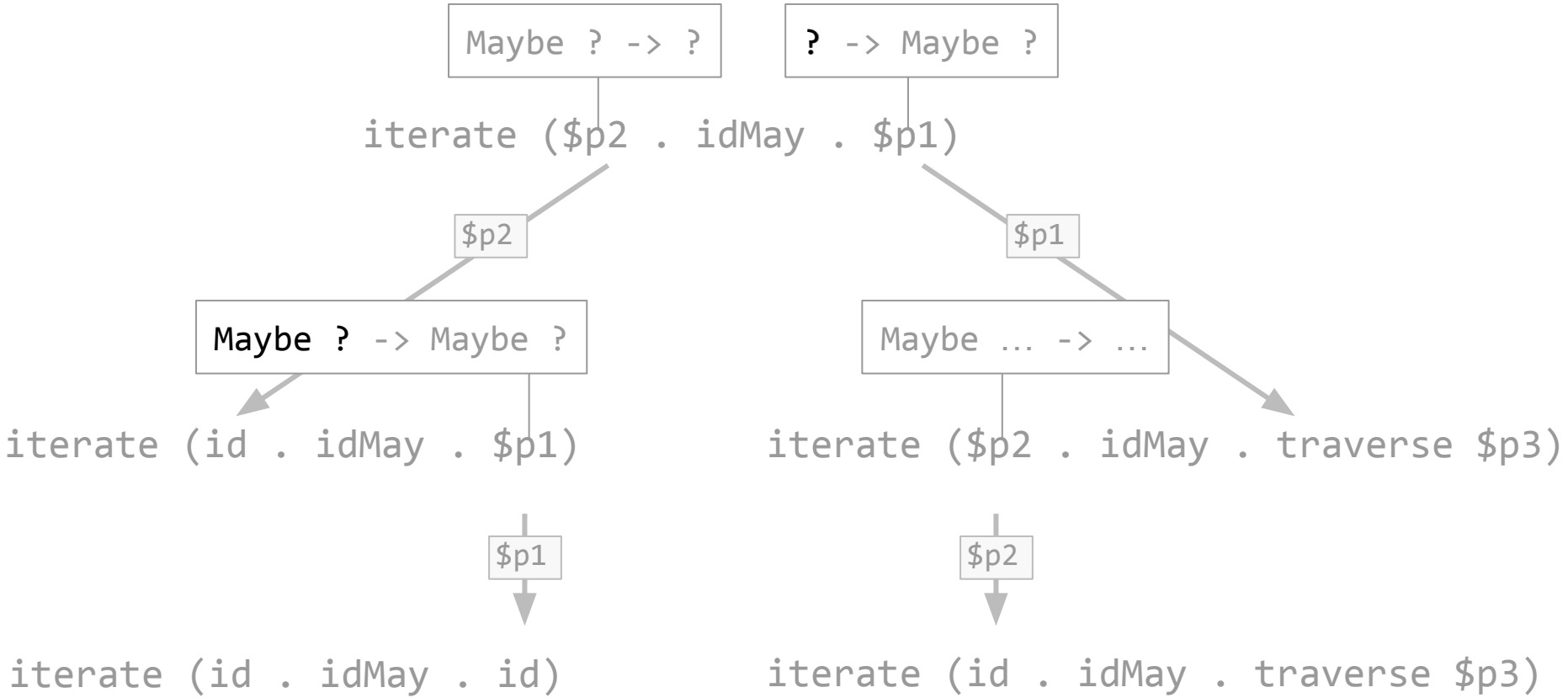
Stuck macros

deterministically interleaving
macro-expansion and type-checking

presented at C◦mp◦se NYC 2019 by Samuel Gélineau

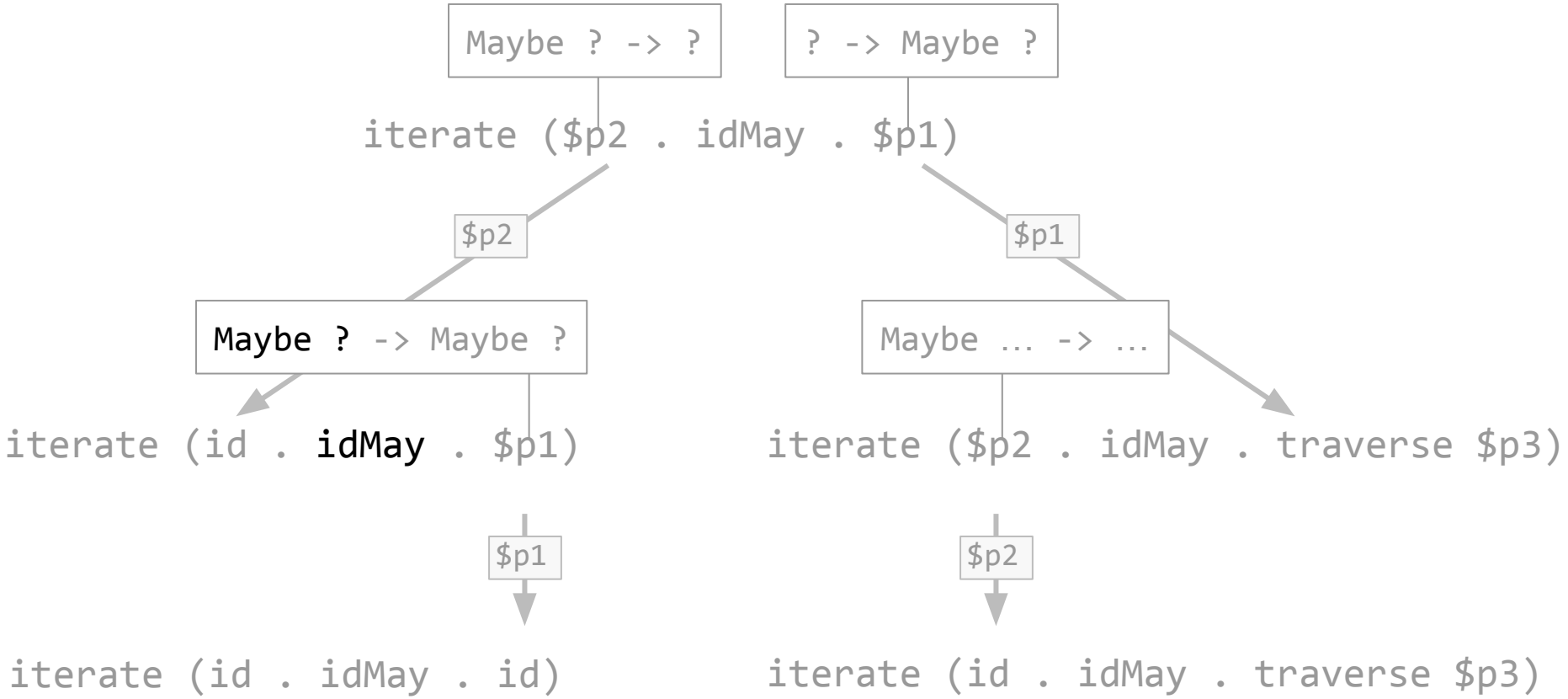
type-aware macros are not confluent

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



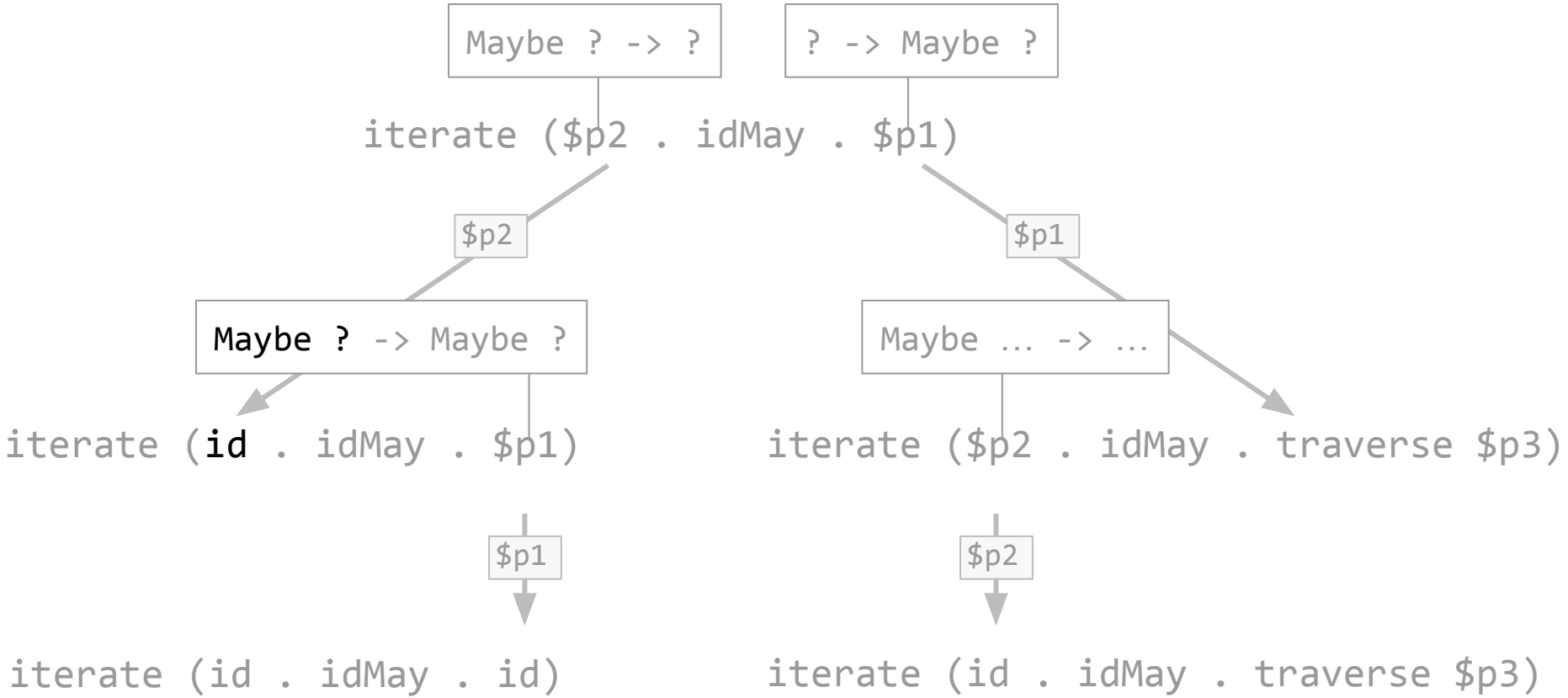
type-aware macros are not confluent

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



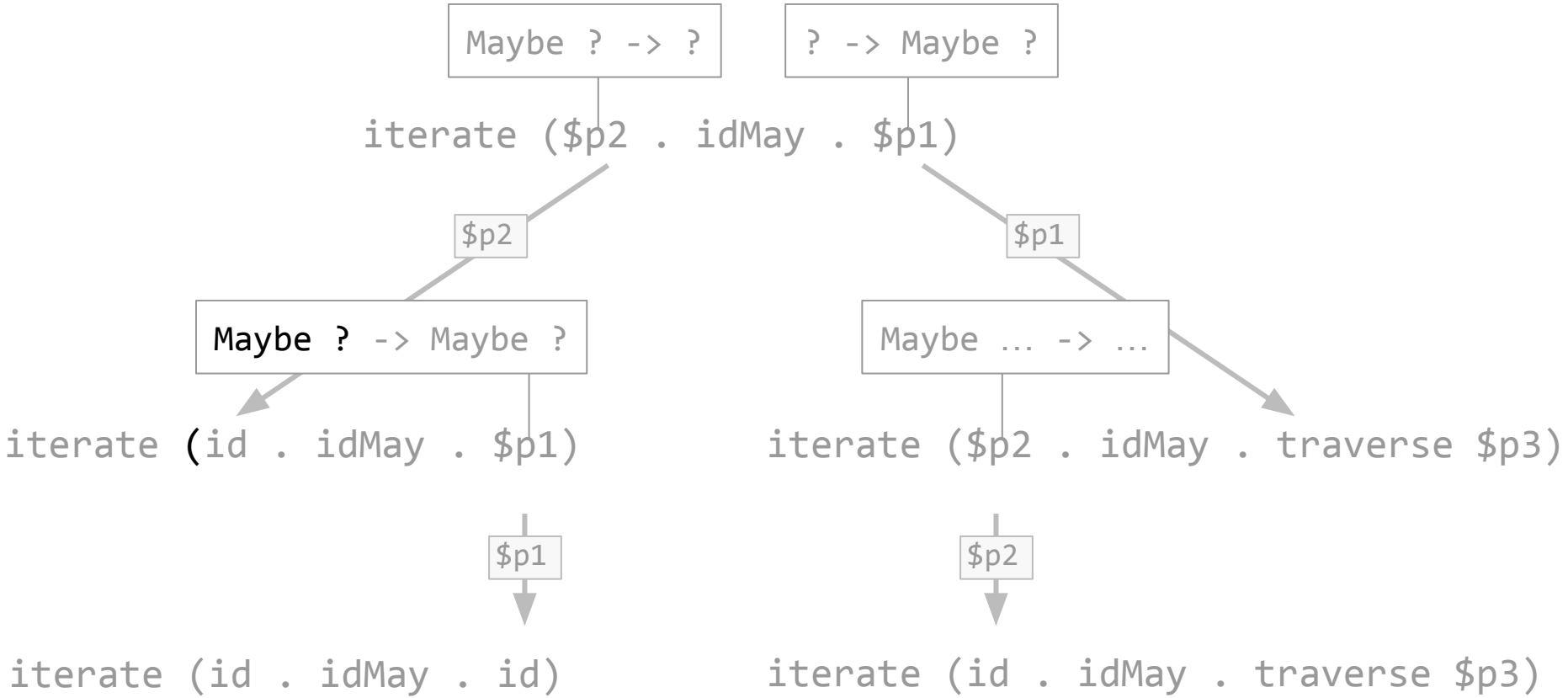
type-aware macros are not confluent

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



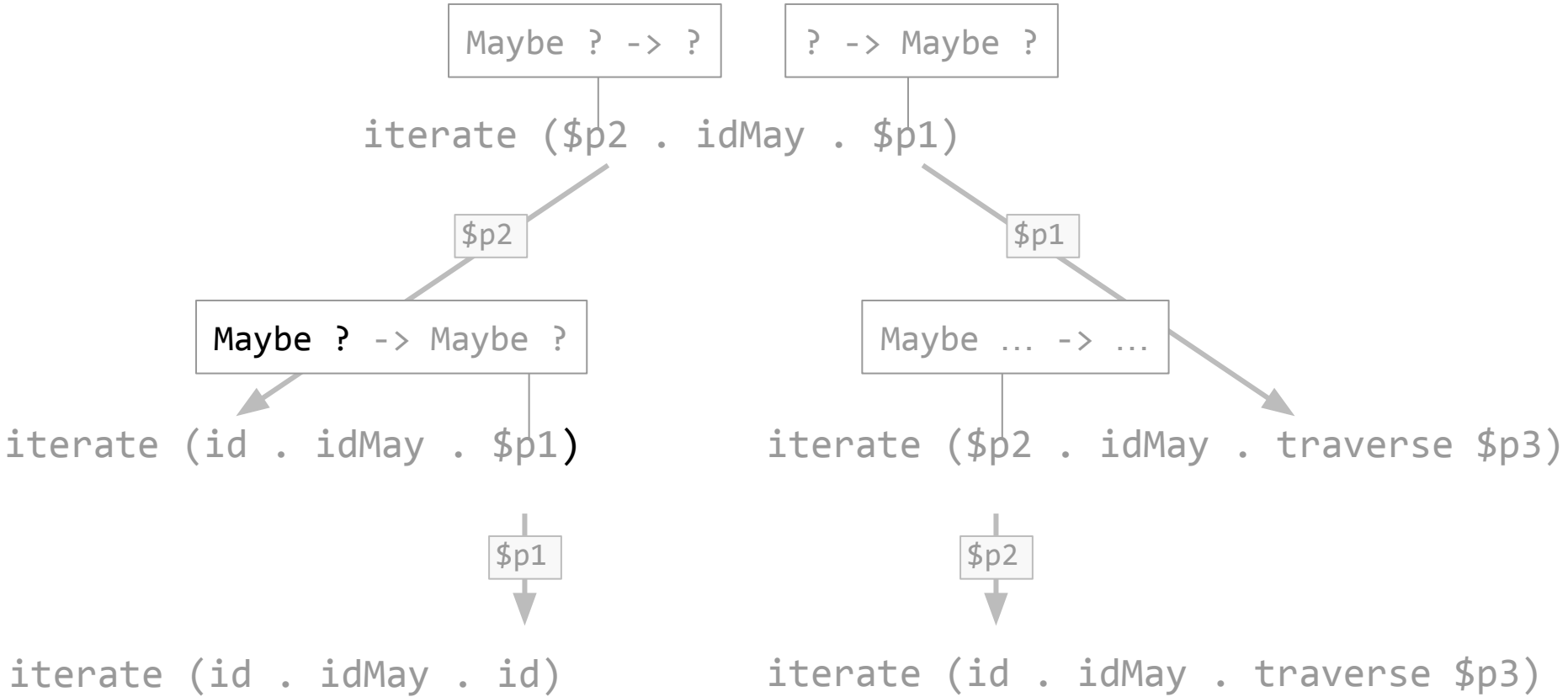
type-aware macros are not confluent

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



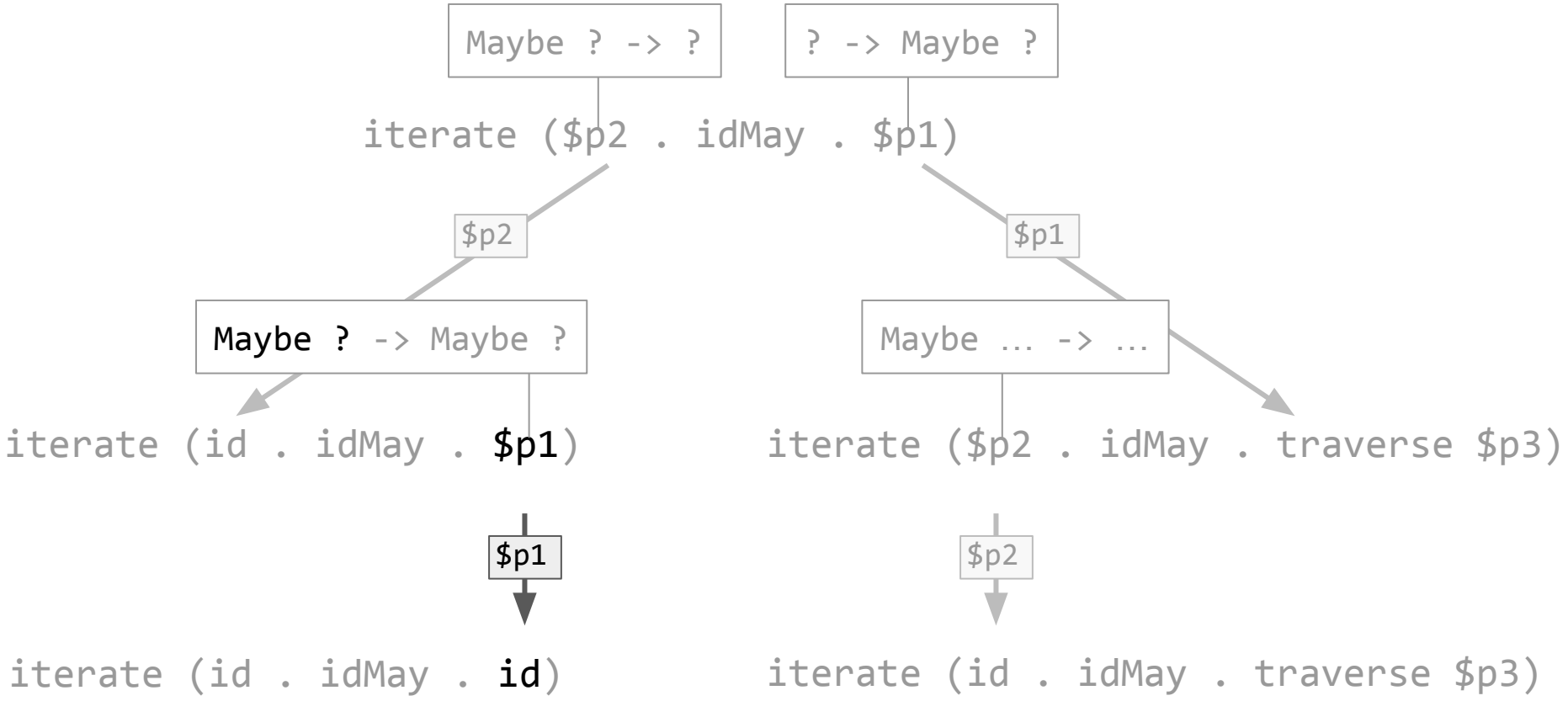
type-aware macros are not confluent

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



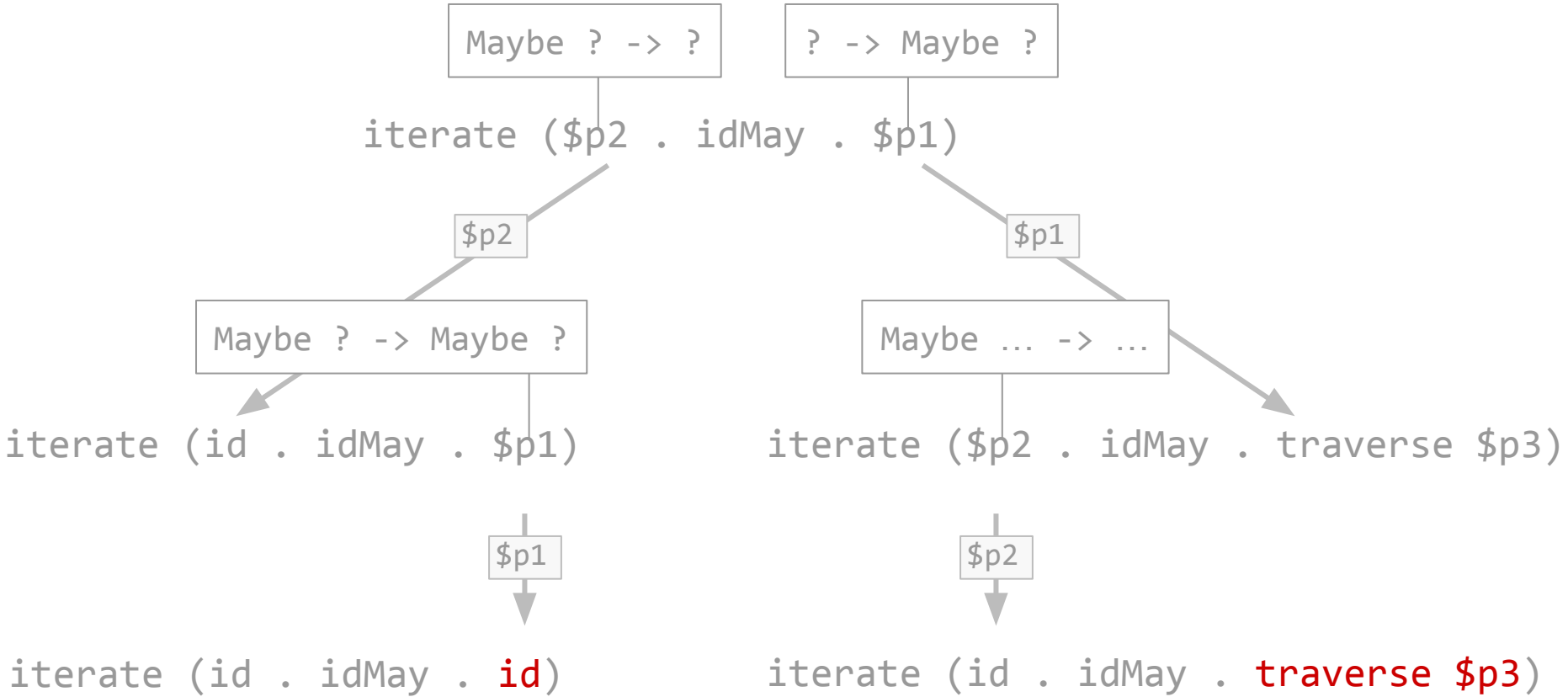
type-aware macros are not confluent

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```

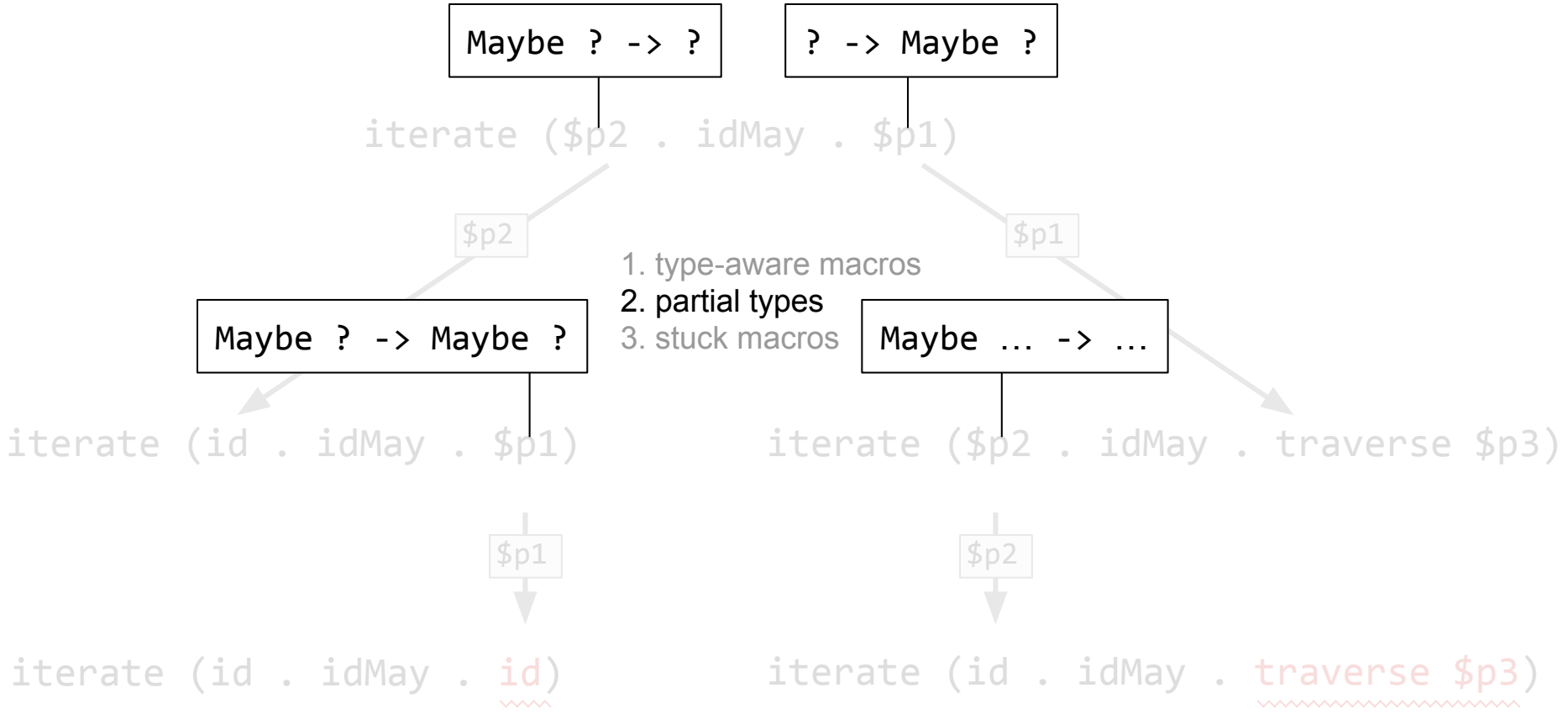


type-aware macros are not confluent

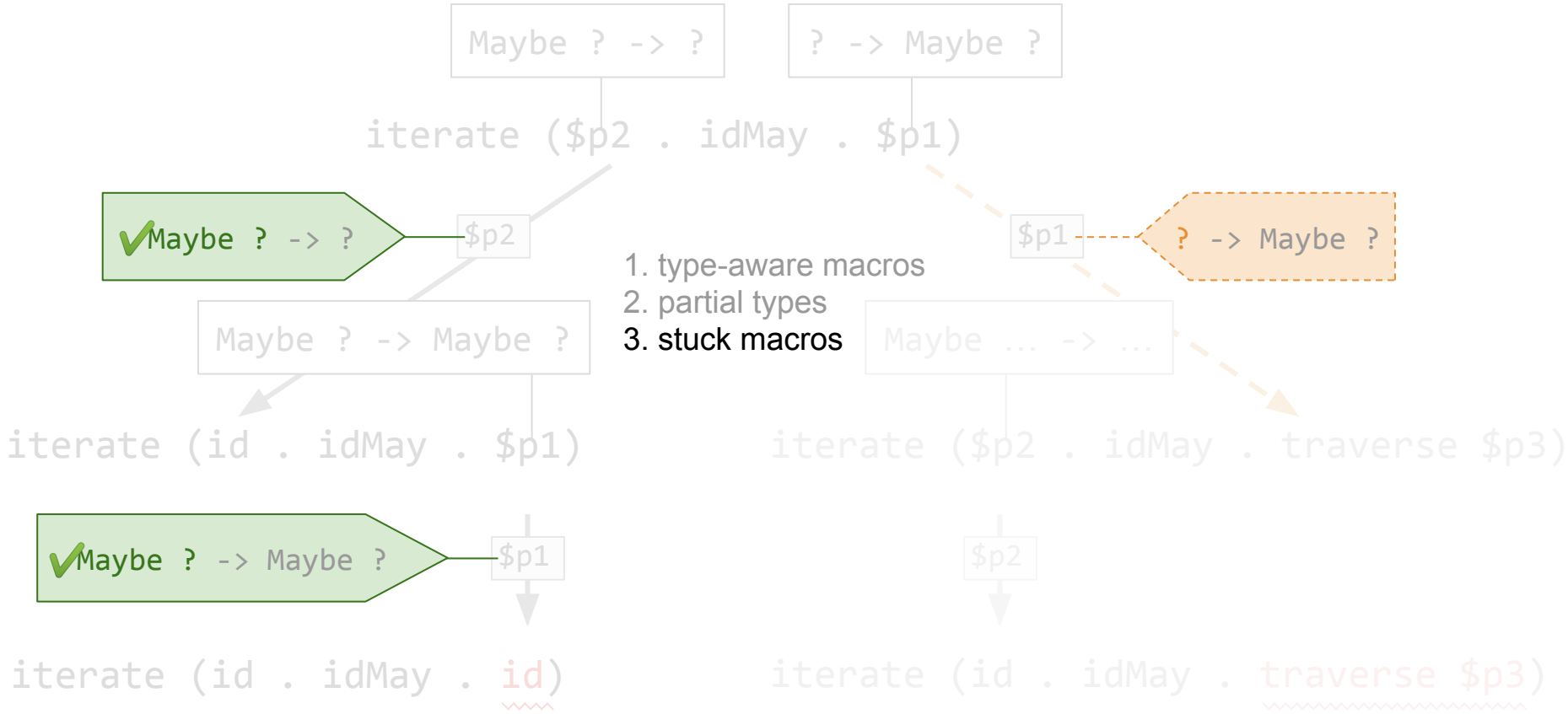
```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



outline


Maybe ? -> ?
? -> Maybe ?
iterate ($p2 . idMay . $p1)
$p2
$p1
1. type-aware macros
2. partial types
3. stuck macros
Maybe ? -> Maybe ?
Maybe ... -> ...
iterate (id . idMay . $p1)
iterate ($p2 . idMay . traverse $p3)
$p1
$p2
iterate (id . idMay . id)
iterate (id . idMay . traverse $p3)

outline


Maybe ? -> ?
? -> Maybe ?
iterate ($p2 . idMay . $p1)
✔Maybe ? -> ?
$p2
$p1
? -> Maybe ?
1. type-aware macros
2. partial types
3. stuck macros
Maybe ? -> Maybe ?
Maybe ... -> ...
iterate (id . idMay . $p1)
iterate ($p2 . idMay . traverse $p3)
✔Maybe ? -> Maybe ?
$p1
$p2
iterate (id . idMay . id)
iterate (id . idMay . traverse $p3)

A message from our non-sponsor:

Typer

A message from our non-sponsor:

Typer

coming soon: stuck macros!

Typer, a language by Stefan Monnier (Université de Montréal)

```
if 2 + 2 == 4
  then "sane"
  else "crazy"
```

```
(if_then_else_ (_==_ (_+_ 2 2) 4)
  "sane"
  "crazy")
```

```
define-macro (infix-replicate n op arg) = ...
```

```
triple x = infix-replicate 3 _*_ x
```

```
triple x = x * x * x
```

```
macro : (List Sexp -> Sexp) -> Macro;
```

```
macro : (List Sexp -> Sexp) -> Macro;
infix-replicate : Int -> Macro;

    triple x = infix-replicate 3 _*_ x
```

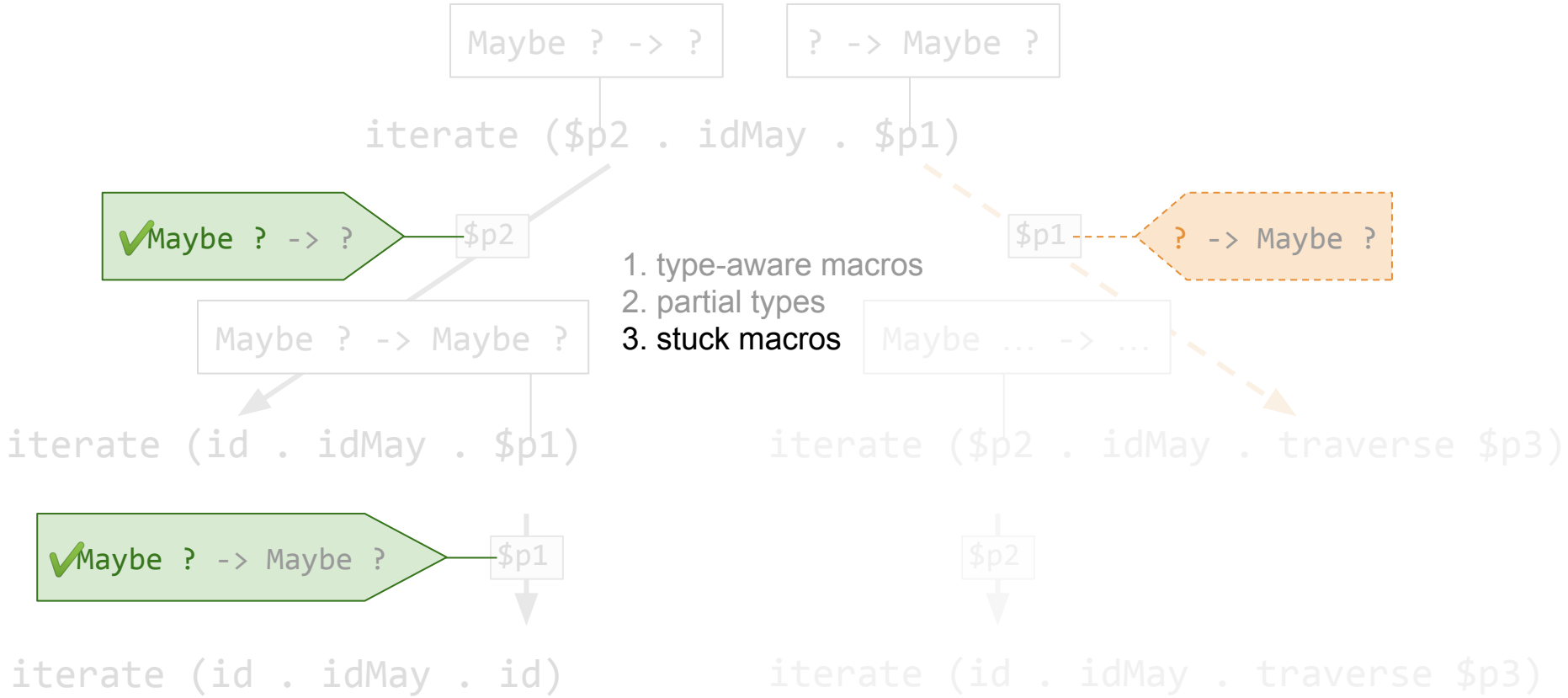
```
    triple x = x * x * x
```

To learn more about

Typer

visit gitlab.com/monnier/typer today!

outline


Maybe ? -> ?
? -> Maybe ?
iterate ($p2 . idMay . $p1)
✔Maybe ? -> ?
$p2
$p1
? -> Maybe ?
1. type-aware macros
2. partial types
3. stuck macros
Maybe ? -> Maybe ?
Maybe … -> …
iterate (id . idMay . $p1)
iterate ($p2 . idMay . traverse $p3)
✔Maybe ? -> Maybe ?
$p1
$p2
iterate (id . idMay . id)
iterate (id . idMay . traverse $p3)

stuck macros

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```

`Arr (Maybe Unknown) Unknown`	`Arr Unknown (Maybe Unknown)`

```
iterate ($p2 . idMay . $p1)
```

```
getExpectedType :: Q PartialType

data PartialType = Arr PartialType PartialType
                 | Maybe PartialType
                 | ...
                 | Unknown
```

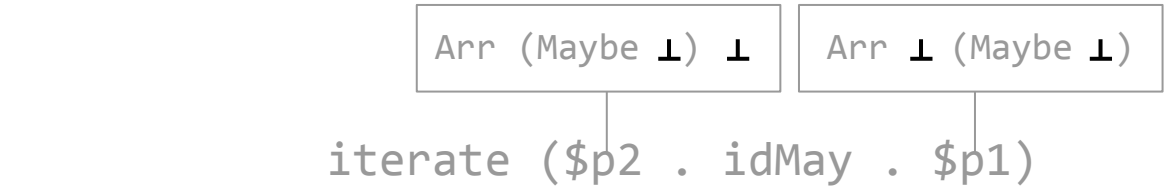
```
data Type = Arr Type Type
          | Maybe Type
          | ...
```

```
pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

`$p1`

stuck macros

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



~~getExpectedType :: Q PartialType~~

```
data PartialType = Arr PartialType PartialType
                 | Maybe PartialType
                 | ...
                 | Unknown
```

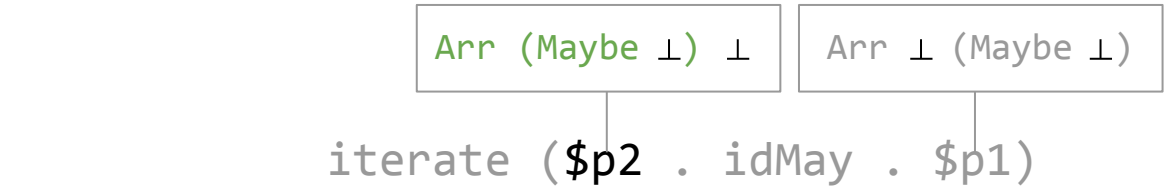
```
getExpectedType :: Q Type

data Type = Arr Type Type
          | Maybe Type
          | ...
```

```
pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |]
  _               -> [| traverse $pullMaybe |]
```

stuck macros

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



~~getExpectedType :: Q PartialType~~

```
data PartialType = Arr PartialType PartialType
                 | Maybe PartialType
                 | ...
                 | Unknown
```

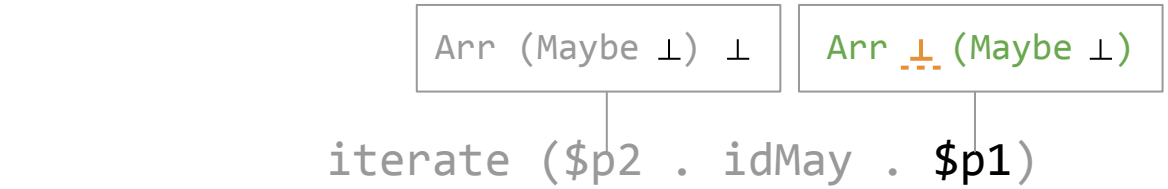
```
getExpectedType :: Q Type

data Type = Arr Type Type
          | Maybe Type
          | ...
```

```
pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |] $p2
  _               -> [| traverse $pullMaybe |]
```

stuck macros

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



~~getExpectedType :: Q PartialType~~

```
data PartialType = Arr PartialType PartialType
                 | Maybe PartialType
                 | ...
                 | Unknown
```

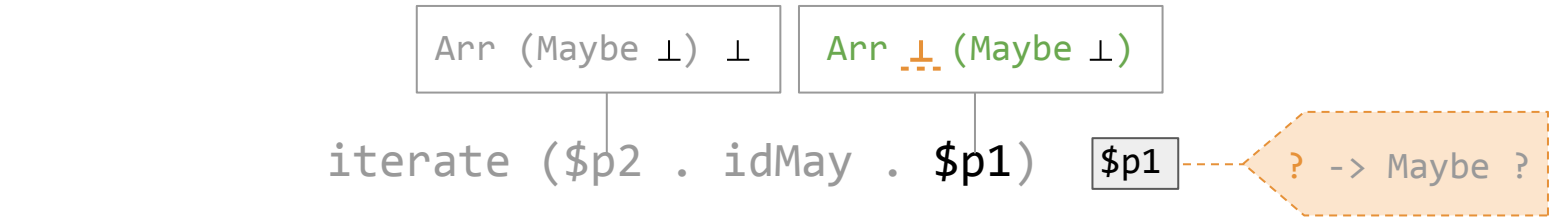
```
getExpectedType :: Q Type

data Type = Arr Type Type
          | Maybe Type
          | ...
```

```
pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |] ~~$p1~~
  _               -> [| traverse $pullMaybe |] ~~$p1~~
```

stuck macros

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



~~getExpectedType :: Q PartialType~~

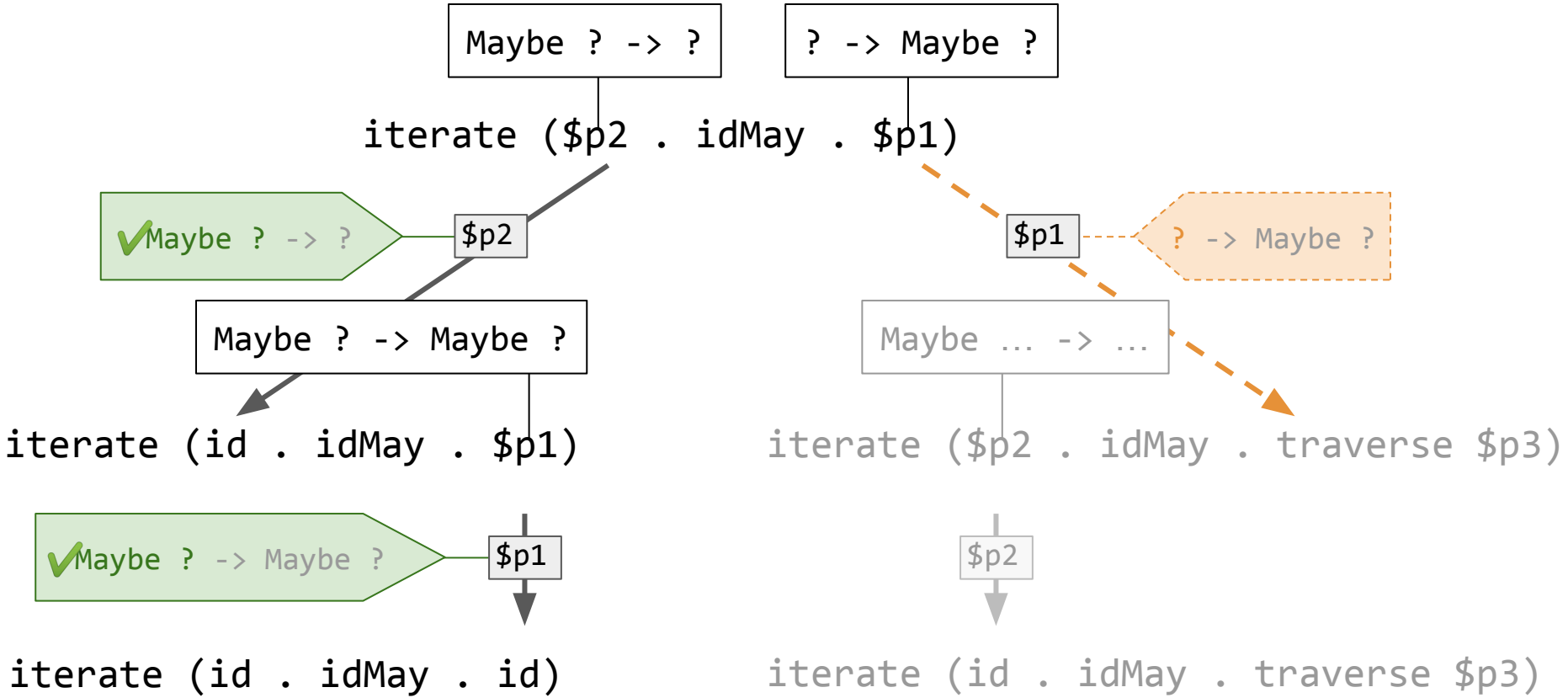
```
data PartialType = Arr PartialType PartialType
                 | Maybe PartialType
                 | ...
                 | Unknown
```

```
getExpectedType :: Q Type

data Type = Arr Type Type
          | Maybe Type
          | ...
```

```
pullMaybe :: Q Exp
pullMaybe = getExpectedType >>= \case
  Arr (Maybe _) _ -> [| id |] $p1
  _               -> [| traverse $pullMaybe |] $p1
```

stuck macros are confluent



these slides: gelisam.com/files/stuck-macros.pdf

Haskell jobs:  remote! (work from anywhere in 🇺🇸 or 🇨🇦

Questions?

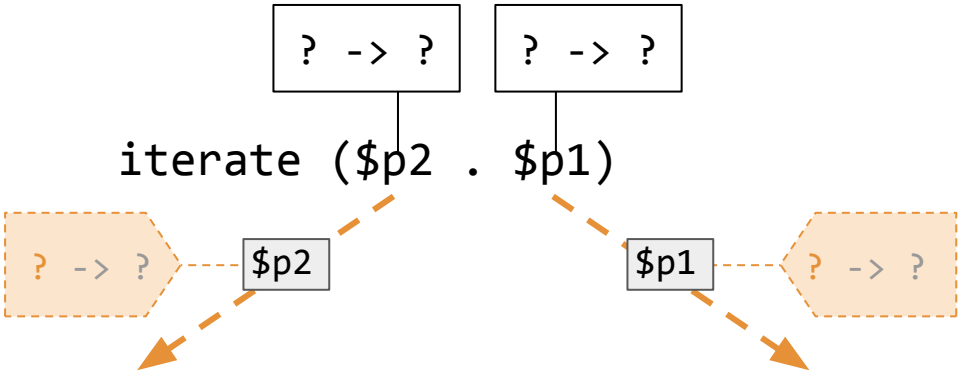
recommended questions (I have bonus slides) :

- can two macros get stuck on each other?
- why is this confluent in general?

presented at C◦mp◦se NYC 2019 by Samuel Gélineau

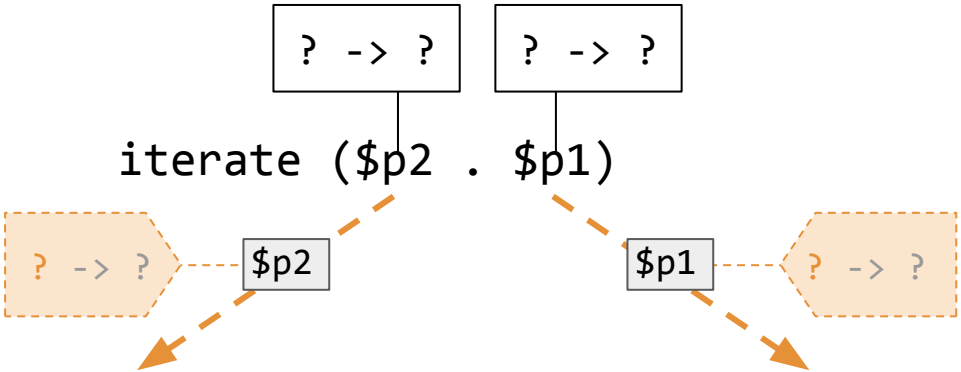
selfish macros

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```



selfish macros

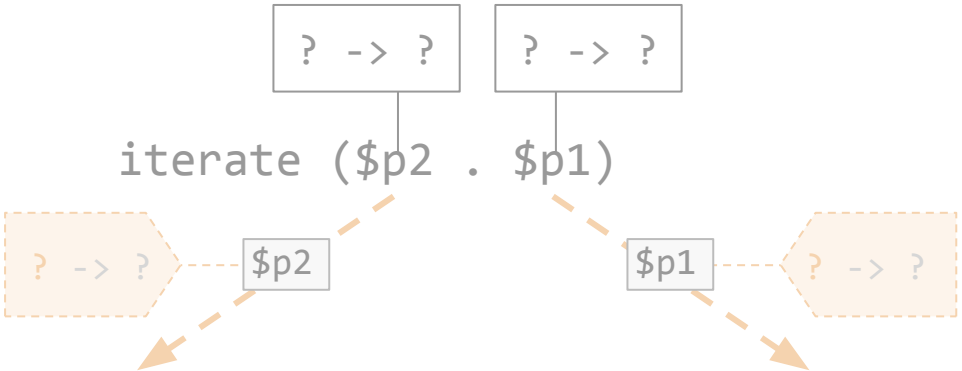
```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```




```
type ambiguity error:
please add a type annotation
```

selfish macros

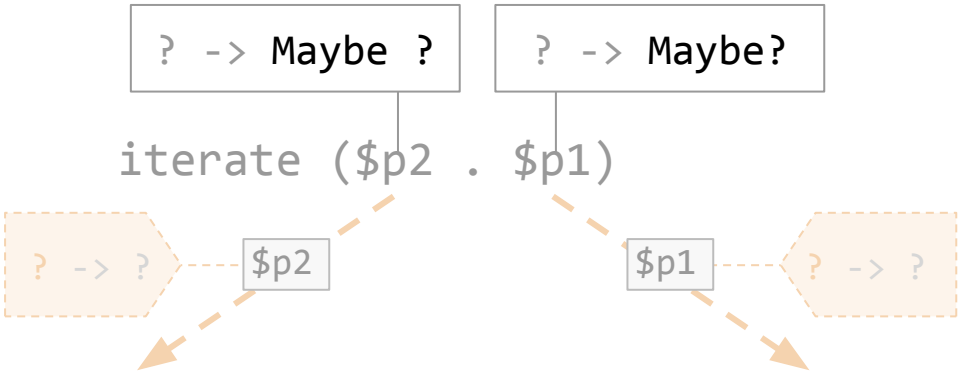
```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```




```
$pullMaybe ::   Maybe a    -> Maybe    a
$pullMaybe ::  [Maybe a]   -> Maybe   [a]
$pullMaybe :: [[Maybe a]]  -> Maybe  [[a]]
```

selfish macros

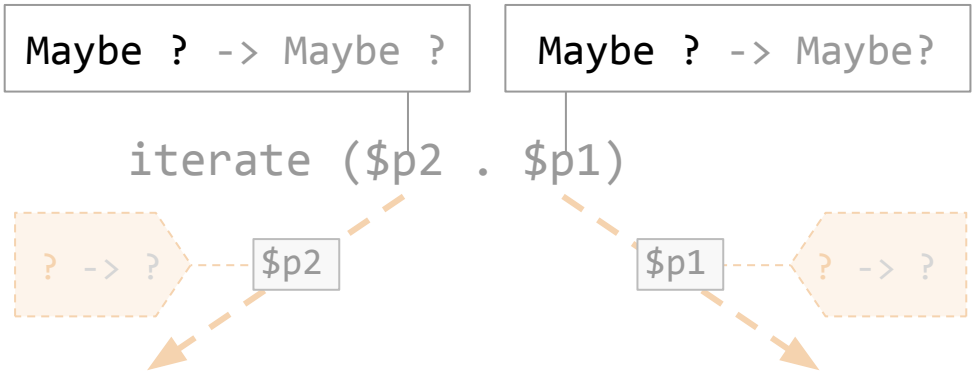
```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```




```
$pullMaybe ::   Maybe a    -> Maybe    a
$pullMaybe ::  [Maybe a]   -> Maybe   [a]
$pullMaybe :: [[Maybe a]]  -> Maybe  [[a]]
```

selfish macros

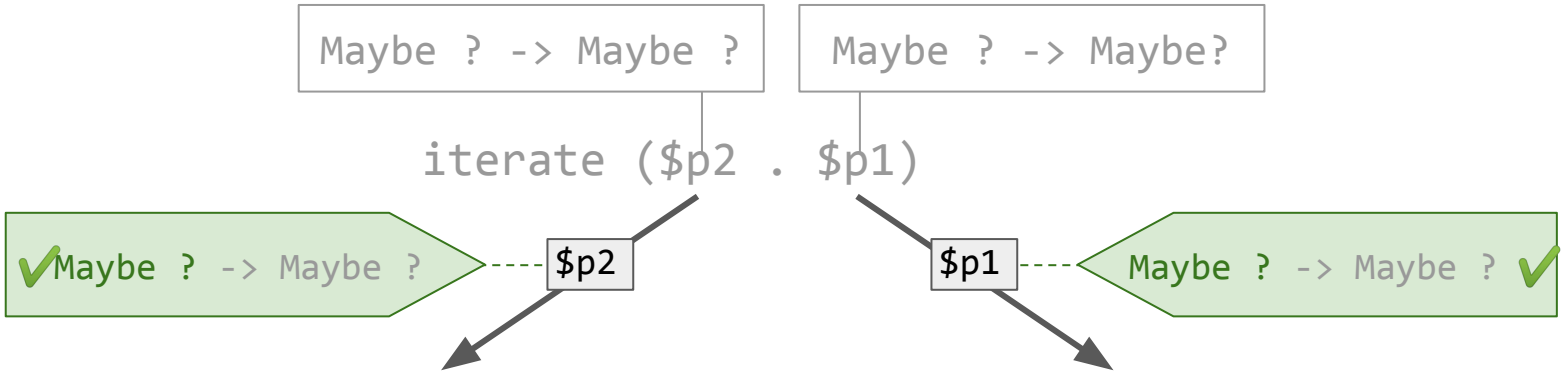
```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```




```
$pullMaybe ::   Maybe a    -> Maybe    a
$pullMaybe ::  [Maybe a]   -> Maybe   [a]
$pullMaybe :: [[Maybe a]]  -> Maybe  [[a]]
```

selfish macros

```
p1 = p2 = p3 = pullMaybe
iterate :: (a -> a) -> (a -> a)
  idMay :: Maybe a -> Maybe a
```




```
$pullMaybe ::   Maybe a    -> Maybe    a
$pullMaybe ::  [Maybe a]   -> Maybe   [a]
$pullMaybe :: [[Maybe a]]  -> Maybe  [[a]]
```

selfish macros

```
pullMaybe :: Q Exp
pullMaybe = do
  getExpectedType >>= \case
    Arr (Maybe _) _ -> [| id |]
    _               -> [| traverse $pullMaybe |]
```

preliminary output

```
pullMaybe :: Q Exp
pullMaybe = do
  setPartialType [| Maybe _ -> Maybe _ |]
  getExpectedType >>= \case
    Arr (Maybe _) _ -> [| id |]
    _               -> [| traverse $pullMaybe |]
```

preliminary output

```
pullMaybe :: Q Exp
pullMaybe = do
  -- setPartialType [| Maybe _ -> Maybe _ |]
  getExpectedType >>= \case
    Arr (Maybe _) _ -> [| id |]
    _               -> [| traverse $pullMaybe |]
```

preliminary output

```
pullMaybe :: Q Exp
pullMaybe = do
  -- setPartialType [| Maybe _ -> Maybe _ |]
  getExpectedType >>= \case
    Arr (Maybe _) _ -> [| id |]
    _               -> [| traverse $pullMaybe |]
```

preliminary output

```
pullMaybe :: Q Exp
pullMaybe = do
  [setPartialType| Maybe s -> Maybe t |]
  case s of
    Arr (Maybe _) _ -> [| id |]
    _               -> [| traverse $pullMaybe |]
```

these slides: gelisam.com/files/stuck-macros.pdf

Haskell jobs:  remote! (work from anywhere in 🇺🇸 or 🇨🇦

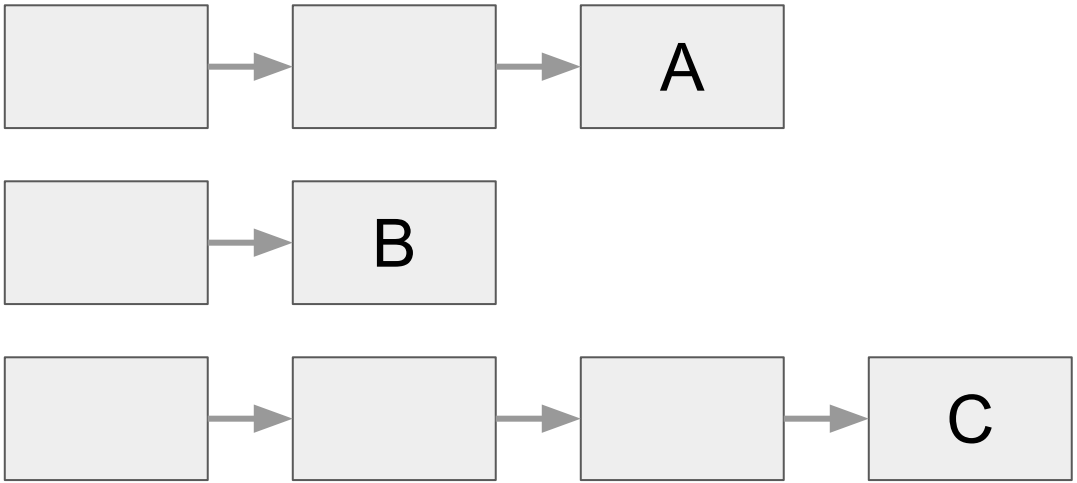
Questions?

recommended questions (I have bonus slides) :

- can two macros get stuck on each other?
- why is this confluent in general?

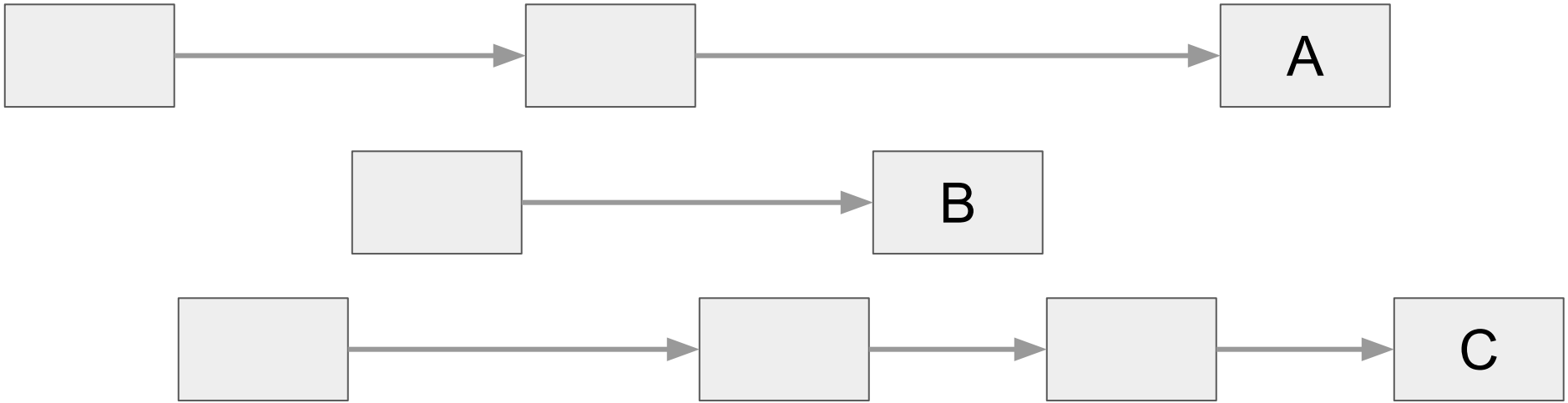
presented at C◦mp◦se NYC 2019 by Samuel Gélineau

`N independent processes`



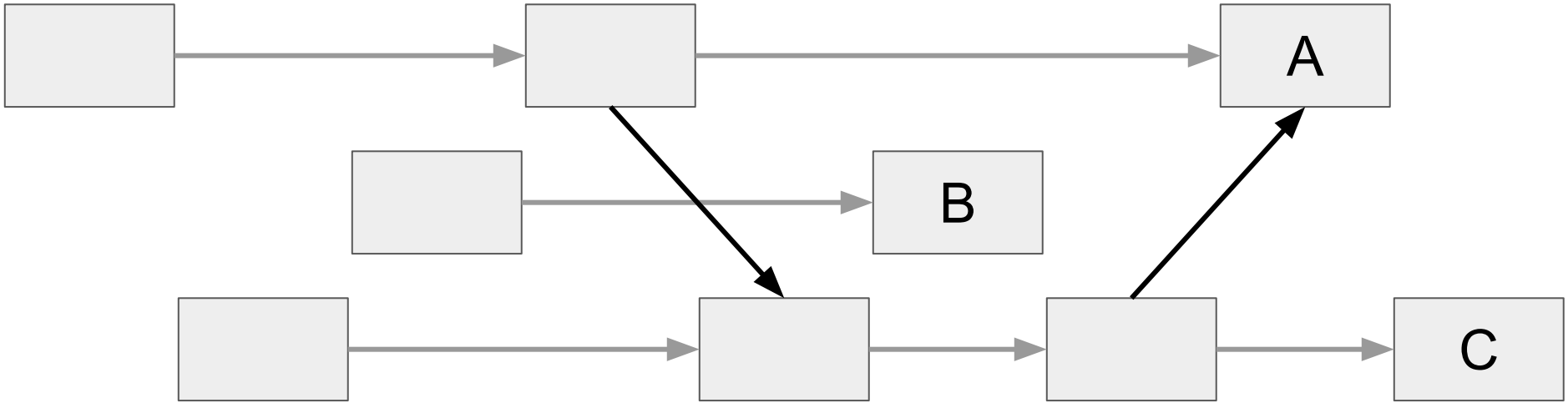
proof sketch

N independent processes:
same results regardless of interleaving



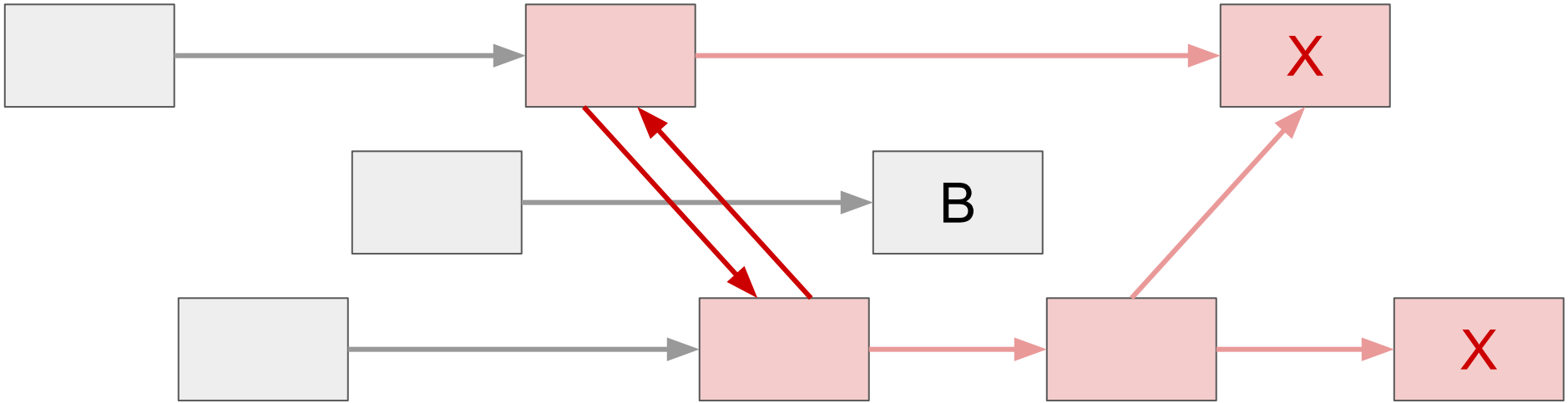
proof sketch

N independent processes with ordering constraints:
same results regardless of interleaving



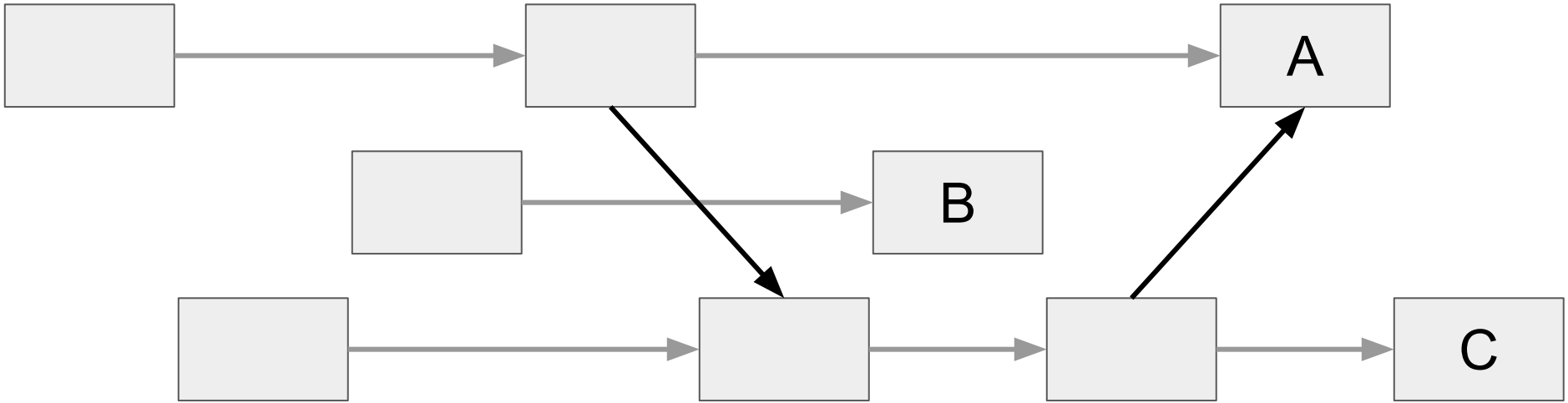
proof sketch

N independent processes with ordering constraints:
same results regardless of interleaving



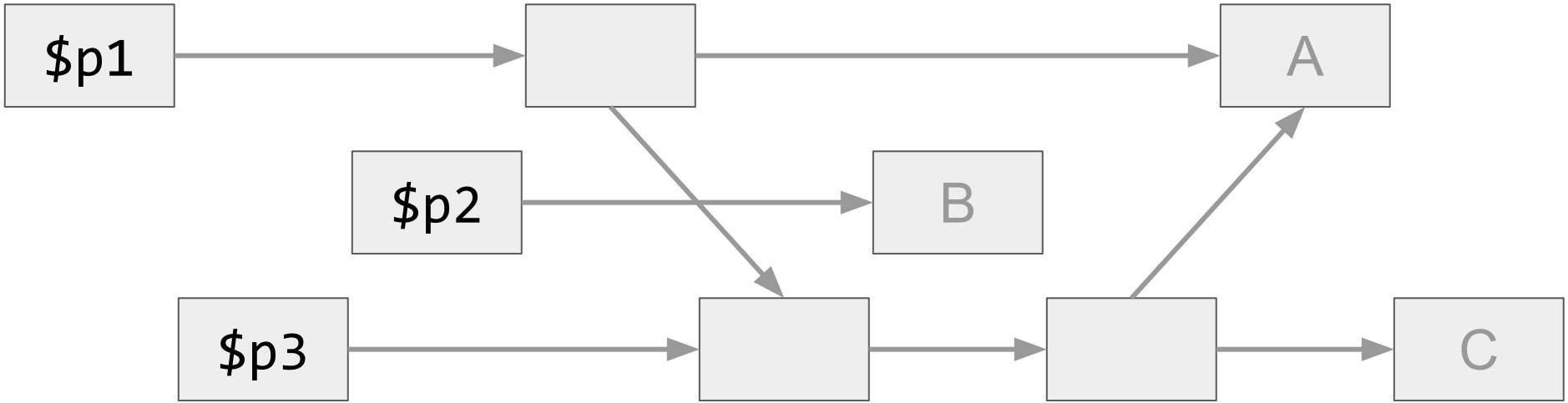
proof sketch

N independent processes with ordering constraints:
same results regardless of interleaving



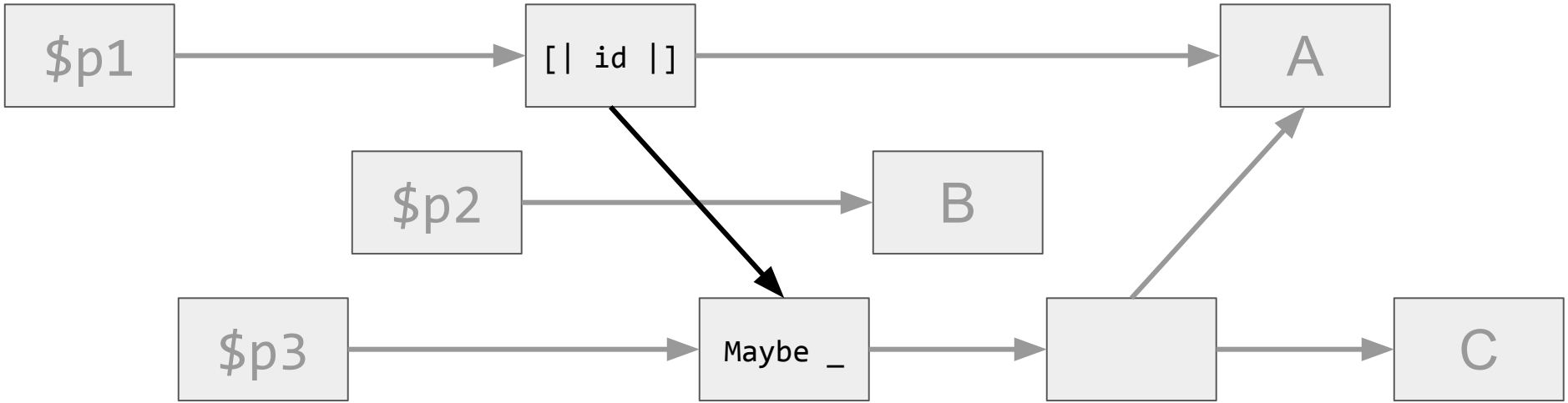
proof sketch

N independent macro expansions with ordering constraints:
same results regardless of interleaving



proof sketch

N independent macro expansions which sometimes get stuck:
same results regardless of interleaving



these slides: gelisam.com/files/stuck-macros.pdf

Haskell jobs:  remote! (work from anywhere in 🇺🇸 or 🇨🇦

Questions?

recommended questions (I have bonus slides) :

- can two macros get stuck on each other?
- why is this confluent in general?

presented at C◦mp◦se NYC 2019 by Samuel Gélineau