Relax!

The Semilenient Core of Choreographic Programming

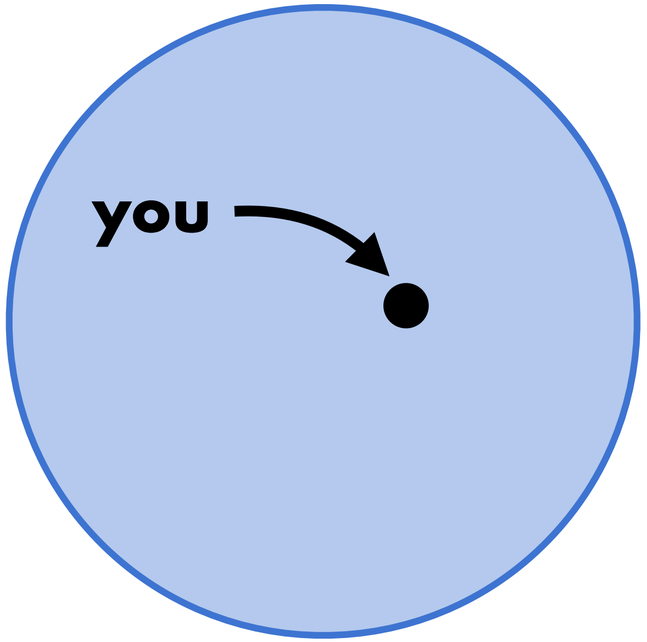
Dan Plyukhin

Xueying Qin

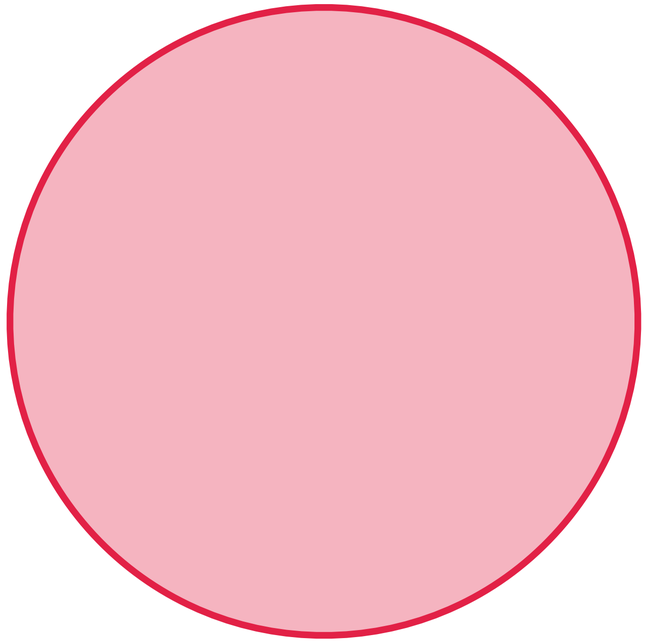
Fabrizio Montesi

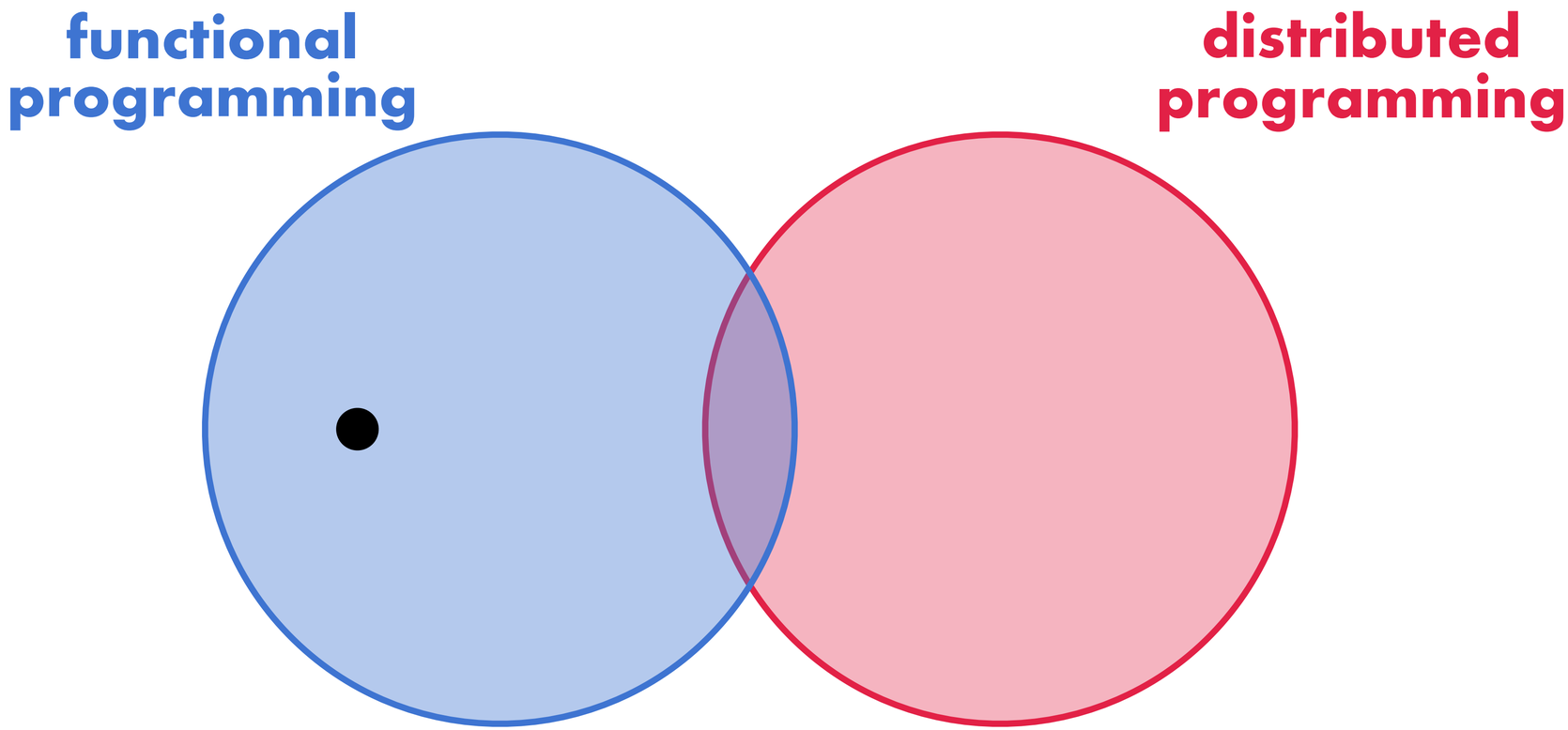
SDU

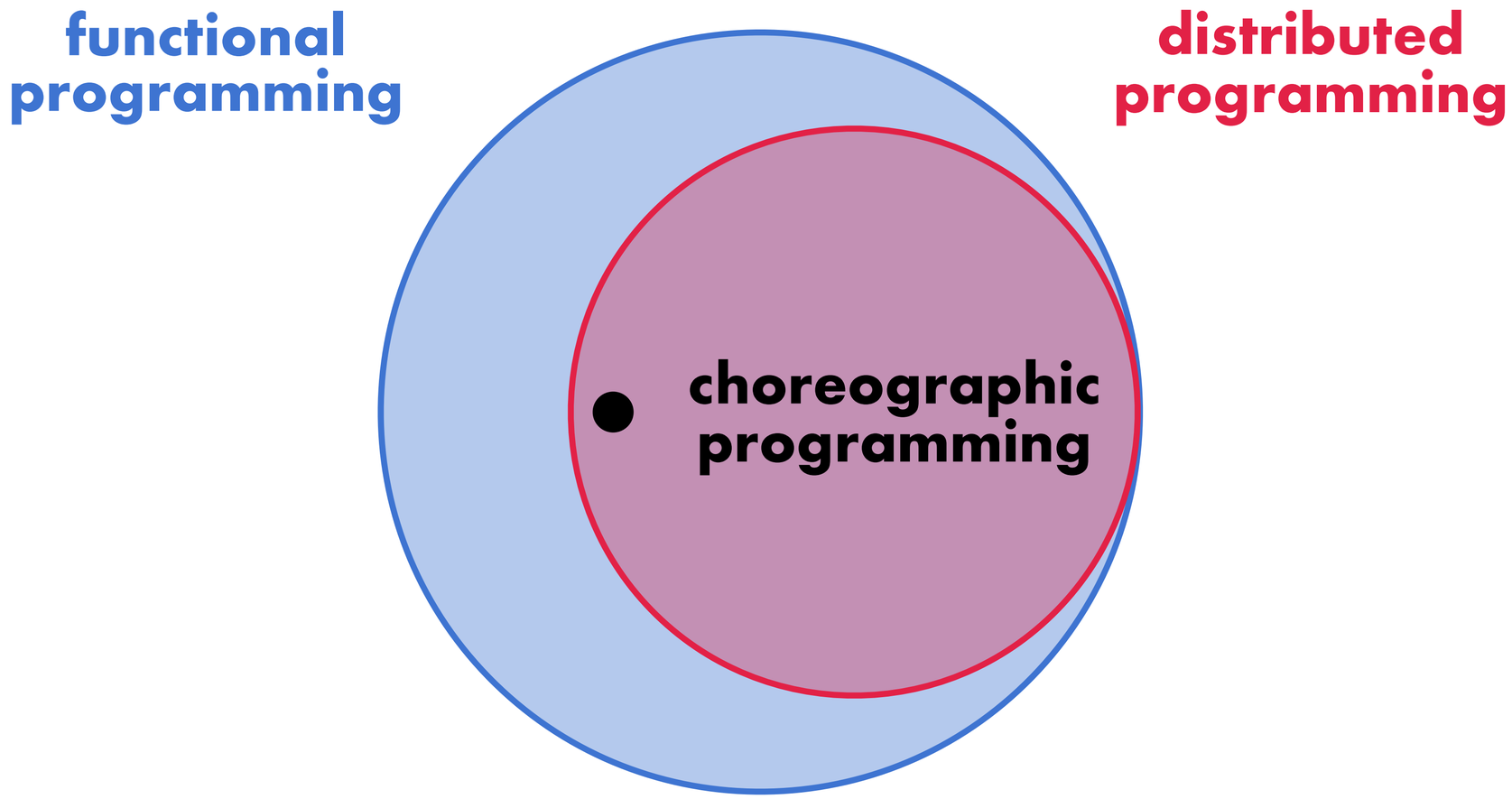
functional programmers



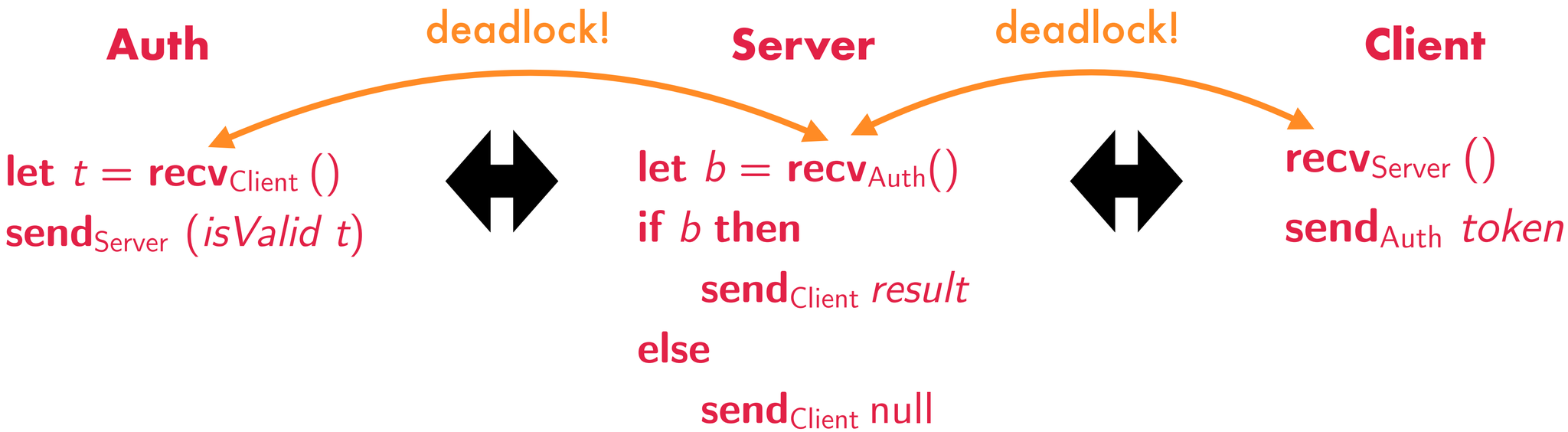
distributed programmers




functional
programming
distributed
programming


functional
programming
distributed
programming
choreographic
programming

the problem with
distributed programming



the problem with
distributed programming

Auth

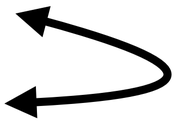
$\textbf{let } t = \textbf{recv}_{\text{Client}}\ ()$
$\textbf{send}_{\text{Server}}\ (\mathit{isValid}\ t)$

Server

$\textbf{let } b = \textbf{recv}_{\text{Auth}}()$
$\textbf{if } b \textbf{ then}$
$\quad \textbf{send}_{\text{Client}}\ \mathit{result}$
$\textbf{else}$
$\quad \textbf{send}_{\text{Client}}\ \text{null}$

Client

$\textbf{send}_{\text{Auth}}\ \mathit{token}$
$\textbf{recv}_{\text{Server}}\ ()$



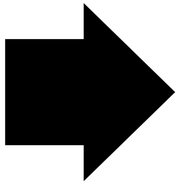
the problem with
distributed programming

that sounds like...
...multitier?
...program partitioning?
...*macroprogramming*?

"choreography"

??????
??????
??????
??????

"projection"



Auth

let $t =$ **recv**$_{\text{Client}}$ ()
send$_{\text{Server}}$ (*isValid t*)

Server

let $b =$ **recv**$_{\text{Auth}}$()
if b **then**
    send$_{\text{Client}}$ *result*
else
    send$_{\text{Client}}$ null

Client

send$_{\text{Auth}}$ *token*
recv$_{\text{Server}}$ ()

baby's first
choreographic program

1. start local

```
let t = getToken()
let b = isValid t
let r =
    if b then
        newResult()
    else
        null
printLine r
```

baby's first
choreographic program

1. start local
2. add locations

```
def myChoreo(Client, Auth, Server) =
    let t = getToken(Client)
    let b = isValid@Auth t
    let r =
        if b then
            newResult(Server)
        else
            null@Server
    printLine@Client r
```

baby's first
choreographic program

1. start local
2. add locations

```
def myChoreo(Client, Auth, Server) =
  let t = getToken(Client)
  let b = isValid@Auth t
  let r =
    if b then
      newResult(Server)
    else
      null@Server
  printLine@Client r
```

Token@**Client**

Bool@**Auth**

⚠ can't apply isValid at **Auth** to token at **Client**

baby's first
choreographic program

```
def myChoreo(Client, Auth, Server) =
  let t = com_{Client,Auth} getToken(Client)
  let b = com_{Auth,Server} (isValid@Auth t)
  let r =
    if b then
      newResult(Server)
    else
      null@Server
  printLine@Client (com_{Server,Client} r)
```

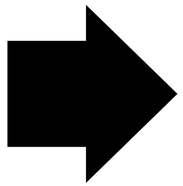
Token@**Auth**

Token@**Client**

1. start local
2. add locations
3. add communications

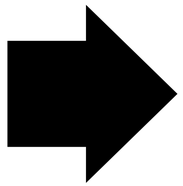
grow your own
choreographic programming language

```
let t = com_{Client,Auth} getToken(Client)
let b = com_{Auth,Server} (isValid@Auth t)
let r =
    if b then
        newResult(Server)
    else
        null@Server
printLine@Client (com_{Server,Client} r)
```



grow your own
choreographic programming language

```
let t = com_{Client,Auth} getToken(Client)
let b = com_{Auth,Server} (isValid@Auth t)
let r =
    if b then
        newResult(Server)
    else
        null@Server
printLine@Client (com_{Server,Client} r)
```

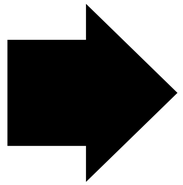


Client

```
send_{Auth} getToken()
printLine (recv_{Server}())
```

grow your own
choreographic programming language

```
let t = com_{Client,Auth} getToken(Client)
let b = com_{Auth,Server} (isValid@Auth t)
let r =
    if b then
        newResult(Server)
    else
        null@Server
printLine@Client (com_{Server,Client} r)
```



Client

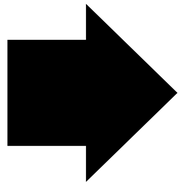
```
send_{Auth} getToken()
printLine (recv_{Server}())
```

Auth

```
let t = recv_{Client}()
send_{Server} (isValid t)
```

grow your own
choreographic programming language

```
let t = com_{Client,Auth} getToken(Client)
let b = com_{Auth,Server} (isValid@Auth t)
let r =
    if b then
        newResult(Server)
    else
        null@Server
printLine@Client (com_{Server,Client} r)
```



Client

```
send_{Auth} getToken()
printLine (recv_{Server}())
```

Auth

```
let t = recv_{Client}()
send_{Server} (isValid t)
```

Server

```
let b = recv_{Auth}()
let r =
    if b then newResult()
    else null
send_{Client} r
```

✅ deadlock-free
✅ type-safe

we don’t agree on the semantics!

(in higher-order models)

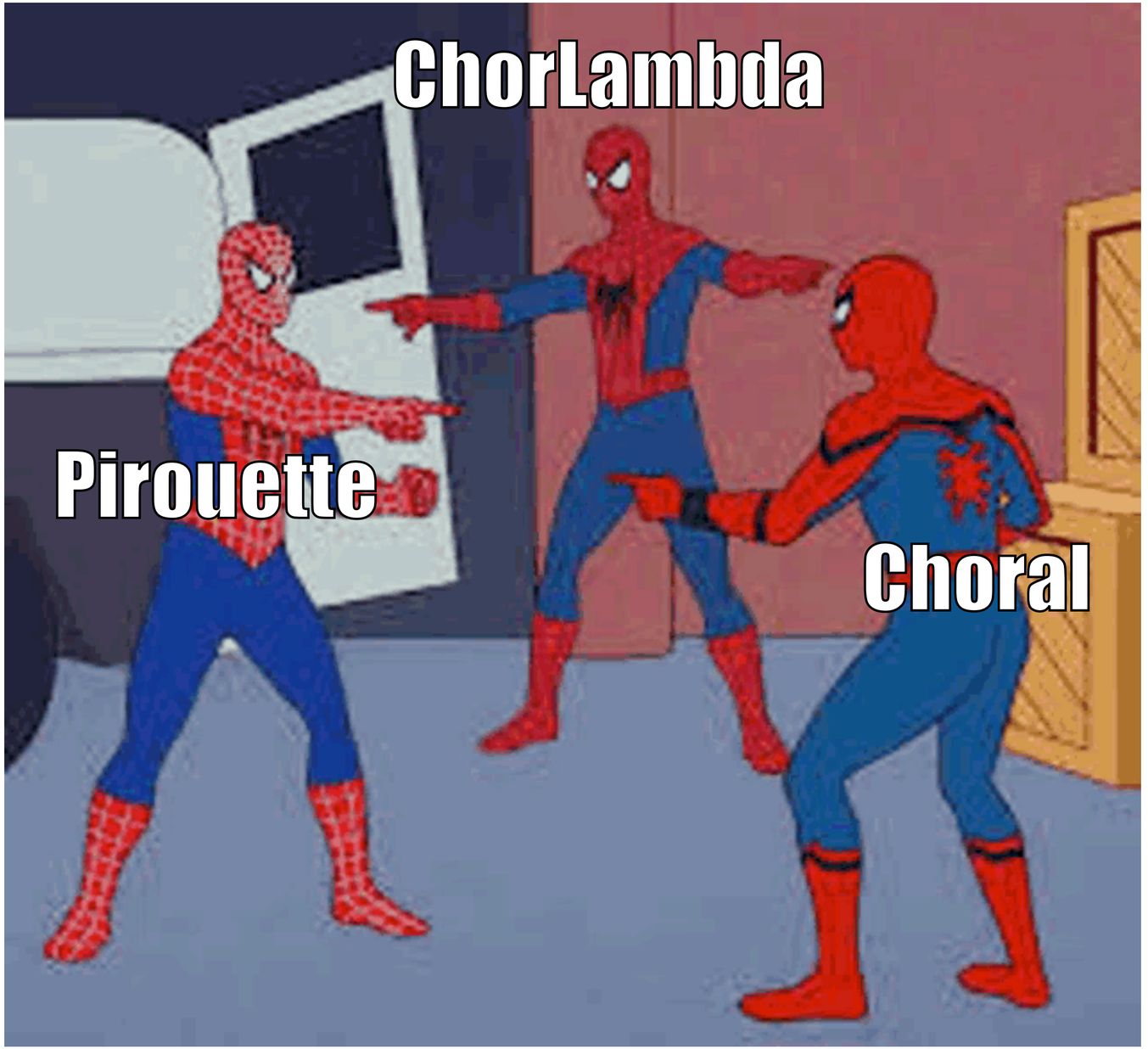
```
void main() {
  println@A(helper());
}

String@A helper() {
  loop(0@C);
  return "hello"@A;
}
```

```
void main() {
  if (helper()) {
    println@A("hello"@A);
  }
}

bool@A helper() {
  loop(0@C);
  return true@A;
}
```

Choral (Java impl.):	✅ print “hello”	✅ print “hello”
Chorλ (ICTAC 2022):	✅ print “hello”	⛔ no effects
Pirouette (POPL 2022):	⛔ no effects	⛔ no effects



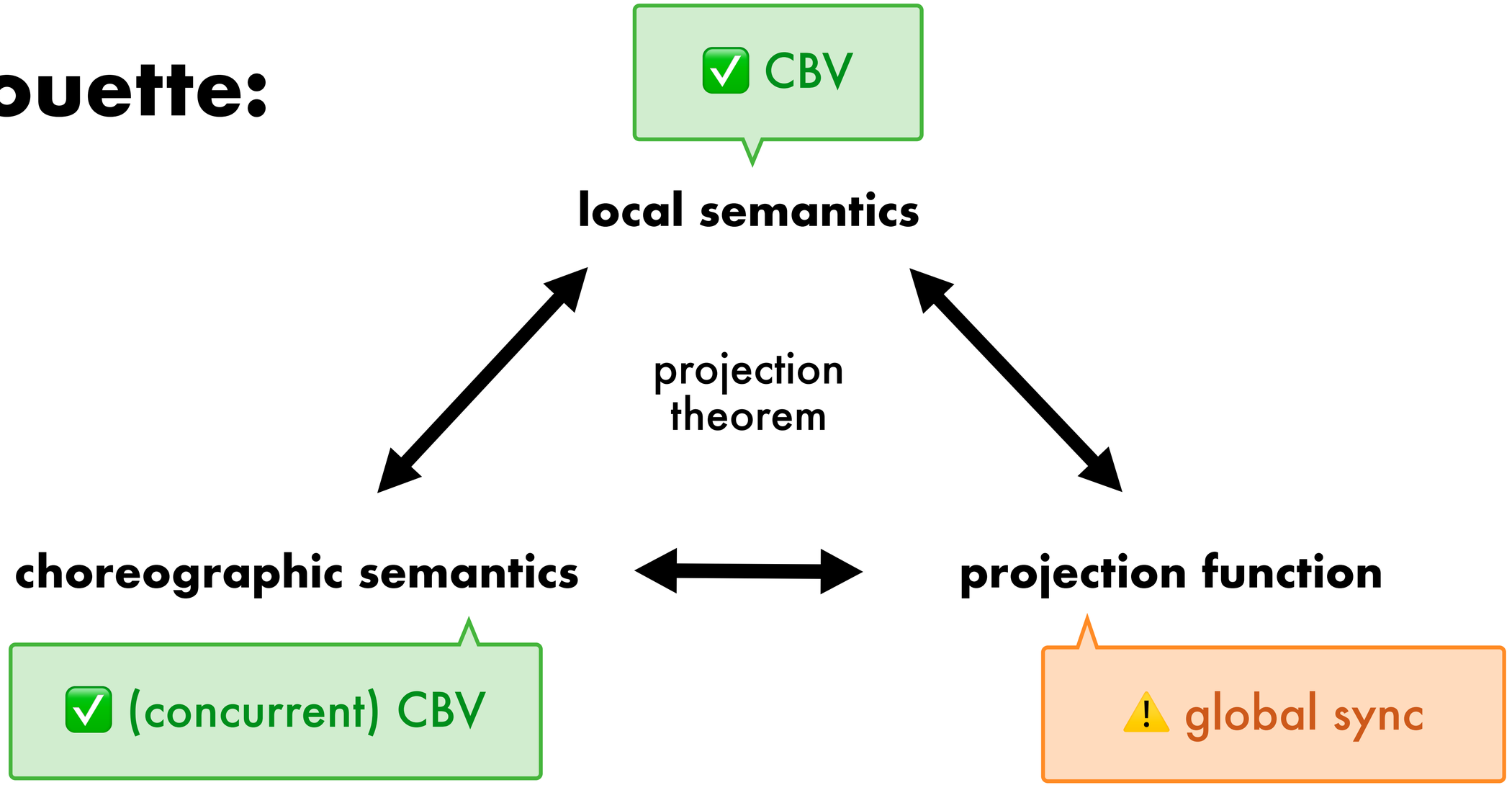
who cares?

🦺 type safety

🚧 refactoring

🏎️ optimization

Pirouette:


✅ CBV
local semantics
projection
theorem
choreographic semantics
projection function
✅ (concurrent) CBV
⚠ global sync

Chorλ:

✅ CBV

local semantics

projection theorem

choreographic semantics ⟷ **projection function**

⚠️ **looks weird:**
evaluation under lambda
ad-hoc commuting conversions

$$\frac{M \xrightarrow{\ell,\mathsf{P}}_D M'}{\lambda x : T.M \xrightarrow{\lambda,\mathsf{P}}_D \lambda x : T.M'}\ [\textsc{InAbs}]$$

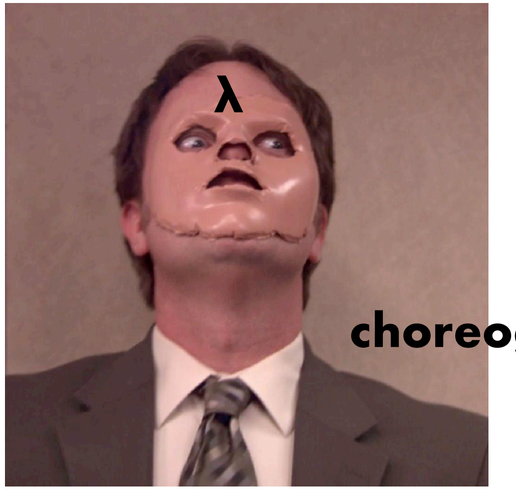
✅ no global sync

$$\frac{x \notin \mathrm{fv}(M') \quad \mathrm{spn}(M') \cap \mathrm{pn}(N) = \varnothing}{M'\ ((\lambda x : T.M)\ N) \rightsquigarrow (\lambda x : T.(M'\ M))\ N)}\ [\text{R-}\textsc{AbsL}]$$

choreographies
(first-order)

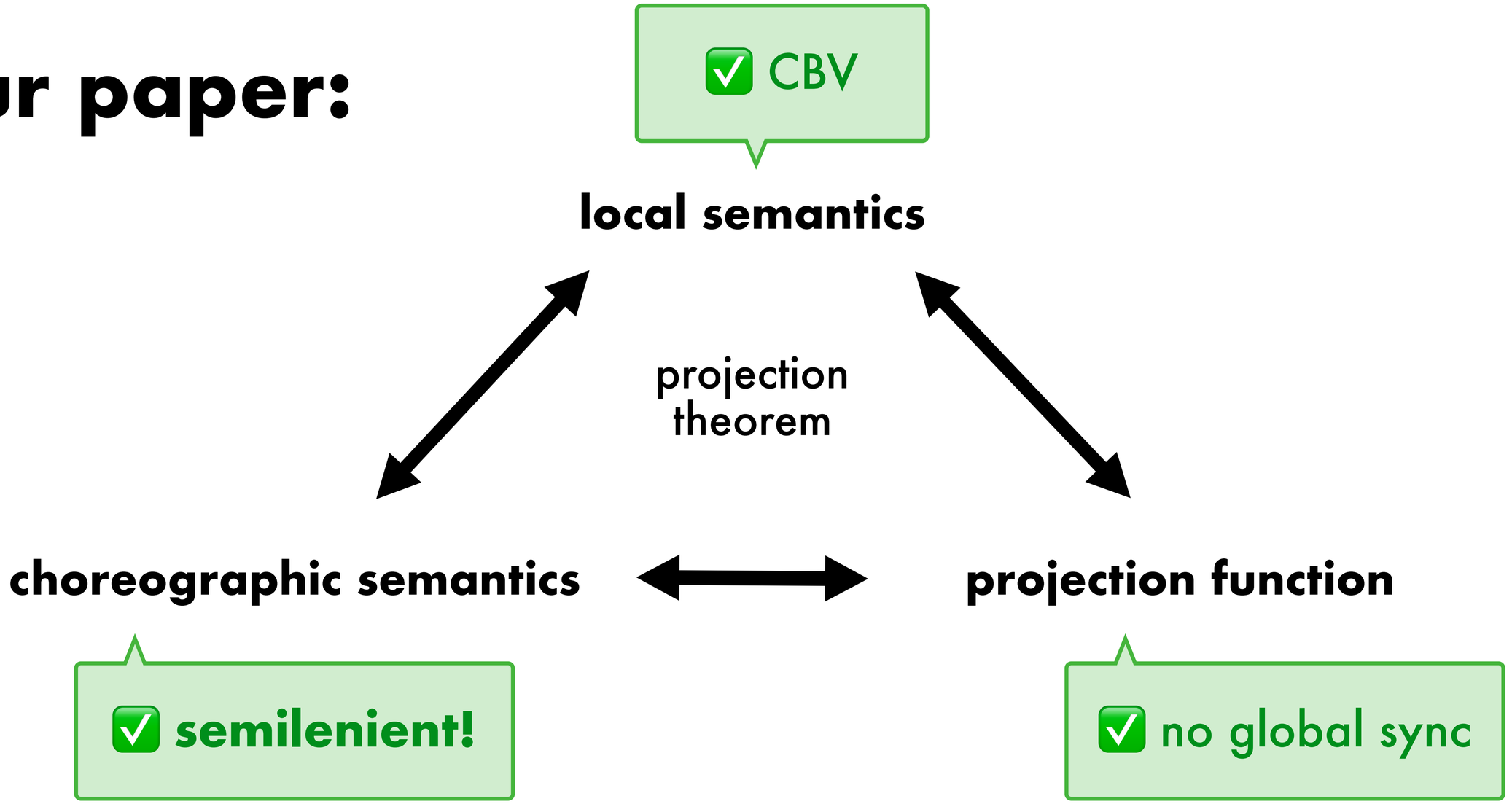
λ-calculus
(higher-order)

how it looks to **CP fans**



how it looks to **FP fans**

our paper:


✅ CBV
local semantics
projection
theorem
choreographic semantics
projection function
✅ semilenient!
✅ no global sync

operational semantics

notion of reduction: redexes

evaluation strategy: evaluation contexts

notion of reduction: redexes

evaluation strategy: evaluation contexts

$$\textbf{if true@A then } M_1 \textbf{ else } M_2 \longrightarrow M_1$$

$$\textbf{if false@A then } M_1 \textbf{ else } M_2 \longrightarrow M_2$$

$$\textbf{com}_{A,\,B}\ \mathit{const}@A \longrightarrow \mathit{const}@B$$

$$\textbf{let } x = V \textbf{ in } M \longrightarrow M[x := V]$$

$$(\lambda\, x.\ M)\ N \longrightarrow \textbf{let } x = N \textbf{ in } M$$

Call-by-Value:

notion of reduction: redexes

evaluation strategy: evaluation contexts

$$E ::= \bullet \mid E\ M \mid V\ E \mid \textbf{if } E \textbf{ then } M \textbf{ else } M \mid \textbf{let } x = E \textbf{ in } M$$

let x = *print*@A “hello”@A
let y = *print*@B “foo”@B
print@A “world”@A

$E =$ **let** x = •
let y = *print*@B “foo”@B
print@A “world”@A

Δ = *print*@A “hello”@A

Call-by-Value:

notion of reduction: redexes

evaluation strategy: evaluation contexts

$$E \quad ::= \quad \bullet \;\mid\; E\ M \;\mid\; V\ E \;\mid\; \textbf{if } E \textbf{ then } M \textbf{ else } M \;\mid\; \textbf{let } x = E \textbf{ in } M$$

let x = *print*@A “hello”@A
let y = *print*@B “foo”@B
print@A “world”@A

determinism:
at most one decomposition $M = E[\Delta]$

prints “hello” “foo” “world”

Call-by-Value:

notion of reduction: redexes

evaluation strategy: evaluation contexts

$$E \quad ::= \quad \bullet \;\mid\; E\ M \;\mid\; V\ E \;\mid\; \textbf{if } E \textbf{ then } M \textbf{ else } M \;\mid\; \textbf{let } x = E \textbf{ in } M$$

let x = *print*@A “hello”@A
let y = *print*@B “foo”@B
print@A “world”@A

what’s the projection?

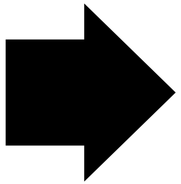
Call-by-Value:

notion of reduction: redexes

evaluation strategy: evaluation contexts

$$E \quad ::= \quad \bullet \;\mid\; E\ M \;\mid\; V\ E \;\mid\; \textbf{if } E \textbf{ then } M \textbf{ else } M \;\mid\; \textbf{let } x = E \textbf{ in } M$$

let x = *print*@A “hello”@A
let y = *print*@B “foo”@B
print@A “world”@A



process A

let x = *print* “hello”
print “world”

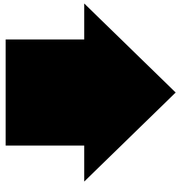
Call-by-Value:

notion of reduction: redexes

evaluation strategy: evaluation contexts

$$E \quad ::= \quad \bullet \;\mid\; E\ M \;\mid\; V\ E \;\mid\; \textbf{if } E \textbf{ then } M \textbf{ else } M \;\mid\; \textbf{let } x = E \textbf{ in } M$$

let x = *print*@A “hello”@A
let y = *print*@B “foo”@B
print@A “world”@A



process A

let x = *print* “hello”
print “world”

process B

print “foo”

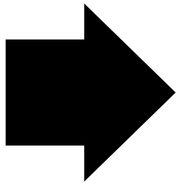
Call-by-Value:

notion of reduction: redexes

evaluation strategy: evaluation contexts

$$E \quad ::= \quad \bullet \;\mid\; E\ M \;\mid\; V\ E \;\mid\; \textbf{if } E \textbf{ then } M \textbf{ else } M \;\mid\; \textbf{let } x = E \textbf{ in } M$$

let x = *print*@A “hello”@A
let y = *print*@B “foo”@B
print@A “world”@A



process A

let x = *print* “hello”
print “world”

process B

print “foo”

1. rewrites are **nondeterministic**
2. rewrites can **evaluate under “let”**

1. “hello” “foo” “world”
2. “hello” “world” “foo”
3. “foo” “hello” “world”

Lenient:

1. rewrites are **nondeterministic**
2. rewrites can **evaluate under "let"**

E ::= $\bullet$ | $E\ M$ | $V\ E$ | **if** E **then** M **else** M | **let** x = E **in** M | **let** x = M **in** E

let x = *print*@A "hello"@A
let y = *print*@B "foo"@B
print@A "world"@A

E = **let** x = $\bullet$
let y = *print*@B "foo"@B
print@A "world"@A

Δ = *print*@A "hello"@A

Lenient:

1. rewrites are **nondeterministic**
2. rewrites can **evaluate under "let"**

E ::= • | E M | V E | **if** E **then** M **else** M | **let** x = E **in** M | **let** x = M **in** E

let x = *print*@A "hello"@A
let y = *print*@B "foo"@B
print@A "world"@A

let x = *print*@A "hello"@A
E = **let** y = •
print@A "world"@A

Δ = *print*@B "foo"@B

Lenient:

1. rewrites are **nondeterministic**
2. rewrites can **evaluate under "let"**

$E ::= \bullet \mid E\ M \mid V\ E \mid \textbf{if } E \textbf{ then } M \textbf{ else } M \mid \textbf{let } x = E \textbf{ in } M \mid \textbf{let } x = M \textbf{ in } E$

let x = *print*@A "hello"@A
let y = *print*@B "foo"@B
print@A "world"@A

$E =$ **let** x = *print*@A "hello"@A
let y = *print*@B "foo"@B
$\bullet$

Δ = *print*@A "world"@A

Lenient:

1. rewrites are **nondeterministic**
2. rewrites can **evaluate under "let"**

$E ::= \bullet \mid E\ M \mid V\ E \mid \textbf{if } E \textbf{ then } M \textbf{ else } M \mid \textbf{let } x = E \textbf{ in } M \mid \textbf{let } x = M \textbf{ in } E$

let x = *print*@A "hello"@A
let y = *print*@B "foo"@B
print@A "world"@A

1. "hello" "foo" "world"
2. "hello" "world" "foo"
3. "foo" "hello" "world"
4. ⛔"world" "hello" "foo"

3. rewrites are **deterministic per process**

SemiLenient:

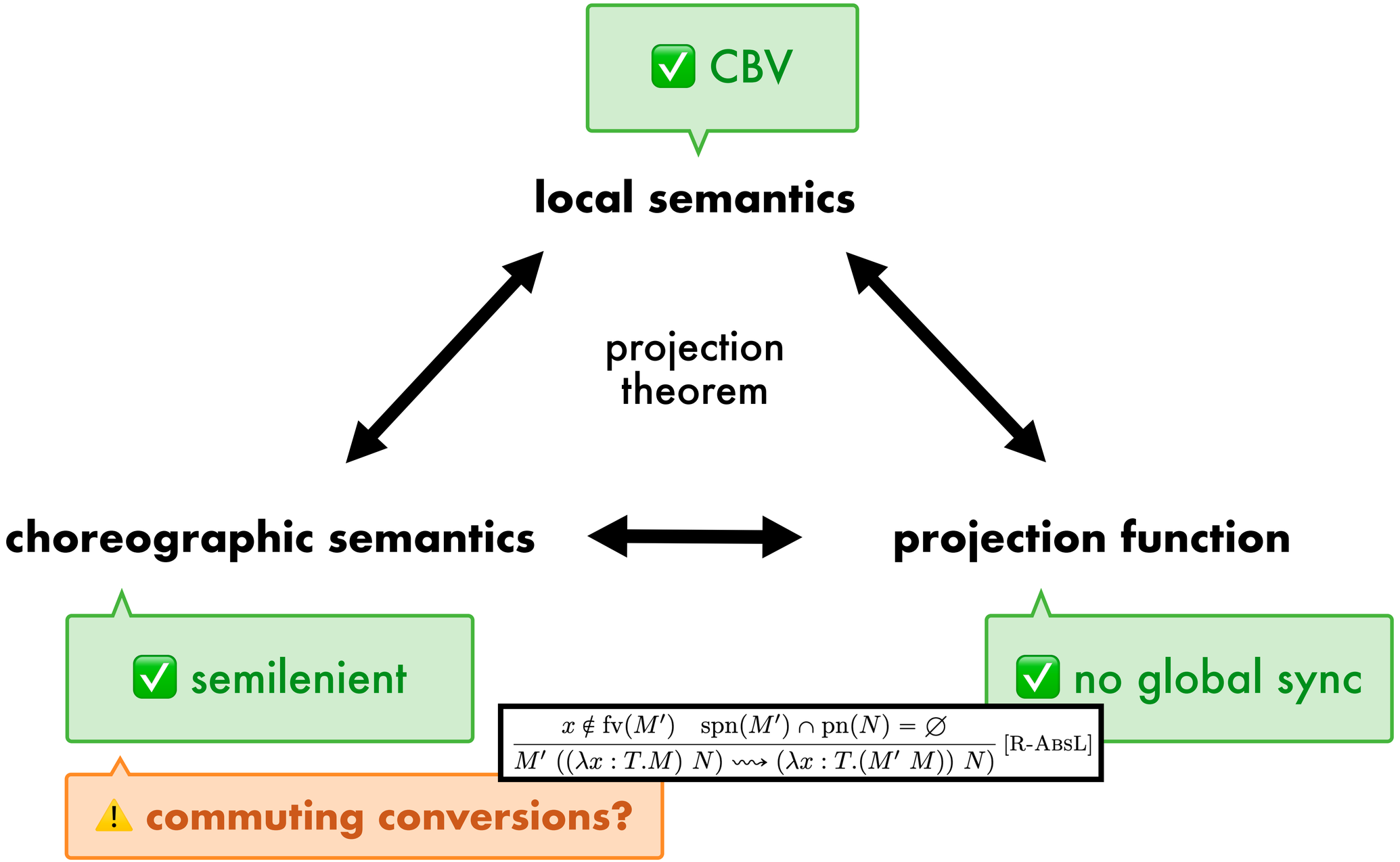
1. rewrites are **nondeterministic**
2. rewrites can **evaluate under "let"**
3. rewrites are **deterministic per process**

$E_P ::= \bullet \mid E_P\, M \mid V\, E_P \mid \textbf{if } E_P \textbf{ then } M \textbf{ else } M \mid \textbf{let } x = E_P \textbf{ in } M \mid \textbf{let } x = M \textbf{ in } E_P \textit{ if } p \notin M$

explains evaluation under lambda!

$$\frac{M \xrightarrow{\ell,\mathsf{P}}_D M'}{\lambda x : T.M \xrightarrow{\lambda,\mathsf{P}}_D \lambda x : T.M'}\ [\textsc{InAbs}]$$

what does that buy us?

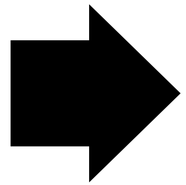

✅ CBV
local semantics
projection
theorem
choreographic semantics
projection function
✅ semilenient
✅ no global sync
⚠ commuting conversions?
$\frac{x \notin \mathrm{fv}(M') \quad \mathrm{spn}(M') \cap \mathrm{pn}(N) = \varnothing}{M'\ ((\lambda x : T.M)\ N) \rightsquigarrow (\lambda x : T.(M'\ M))\ N)}$ [R-AbsL]

motivation:
commuting conversions

(**let** *x* = *loop*(A)
print@B) “hello!”@B

motivation:
commuting conversions

(**let** *x* = *loop*(A)
print@B) “hello!”@B

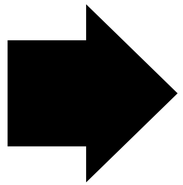


process A

loop()

motivation:
commuting conversions

(**let** *x* = *loop*(A)
print@B) “hello!”@B



process A

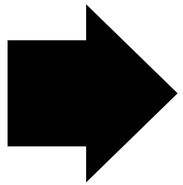
loop()

process B

print “hello”

motivation:
commuting conversions

(**let** *x* = *loop*(A)
print@B) “hello!”@B



process A

loop()

process B

print “hello”

⚠️ choreography never prints

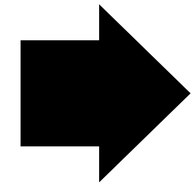
✅ local code prints “hello”

ok... add a rule:

(let x = *M* **in** *N*) *M′* ⟶ **let** x = *M* **in** *N M′*

motivation:
commuting conversions

print@B (**let** *x* = *loop*(A)
"hello!"@*B*)

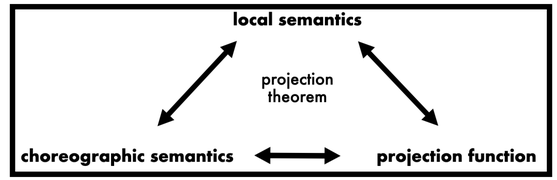


process A

loop()

process B

print "hello"



ok... add a rule:

V (**let** x = *M* **in** *N*) ⟶ **let** x = *M* **in** *V M′*

A Call-By-Need Lambda Calculus

Zena M. Ariola
Computer & Information Science Department
University of Oregon
Eugene, Oregon

John Maraist and *Martin Odersky*
Institut für Programmstrukturen
Universität Karlsruhe
Karlsruhe, Germany

Matthias F
Department of Co
Rice Univ
Houston,

Philip Wadler
Department of Computi
University of Glas
Glasgow, Scotla

I-Structures: Data Structures for Parallel

Compiling without Continuations

Luke Maurer Paul Downen
Zena M. Ariola
University of Oregon, USA
{maurerl,pdownen,ariola}@cs.uoregon.edu

Simon Peyton Jones
Microsoft Research, UK
simonpj@microsoft.com

A Semantical and Operational Account
ll-by-Value Solvability

Carraro[1,2] and Giulio Guerrieri[2]
IS, Università Ca' Foscari Venezia
alberto.carraro@unive.it
niv Paris Diderot, Sorbonne Paris Cité
errieri@pps.univ-paris-diderot.fr

SemiLenient:

Compiling without Continuations

Luke Maurer Paul Downen
Zena M. Ariola
University of Oregon, USA
{maurerl,pdownen,ariola}@cs.uoregon.edu

Simon Peyton Jones
Microsoft Research, UK
simonpj@microsoft.com

$$E_P ::= \bullet \mid E_P\ M \mid V\ E_P \mid \textbf{if } E_P \textbf{ then } M \textbf{ else } M \mid \textbf{let } x = E_P \textbf{ in } M \mid \textbf{let } x = M \textbf{ in } E_P \ \textit{if } p \notin M$$

SemiLenient:

Compiling without Continuations

Luke Maurer Paul Downen
Zena M. Ariola
University of Oregon, USA
{maurerl,pdownen,ariola}@cs.uoregon.edu

Simon Peyton Jones
Microsoft Research, UK
simonpj@microsoft.com

$$F ::= E_P\ M \mid V\ E_P \mid \textbf{if } E_P \textbf{ then } M \textbf{ else } M \mid \textbf{let } x = E_P \textbf{ in } M$$

$$A ::= \textbf{let } x = M \textbf{ in } E_P \quad \textit{if } p \notin M$$

$$E_P ::= \bullet \mid F[E_P] \mid A[E_P]$$

SemiLenient:

Compiling without Continuations

Luke Maurer Paul Downen
Zena M. Ariola
University of Oregon, USA
{maurerl,pdownen,ariola}@cs.uoregon.edu

Simon Peyton Jones
Microsoft Research, UK
simonpj@microsoft.com

$F ::= \bullet\ M \mid V\ \bullet \mid \textbf{if}\ \bullet\ \textbf{then}\ M\ \textbf{else}\ M \mid \textbf{let}\ x = \bullet\ \textbf{in}\ M$

frames consume values

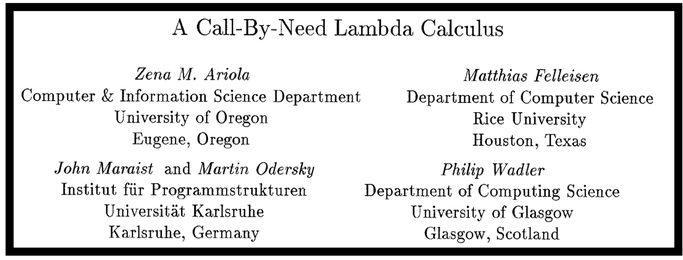
$A ::= \textbf{let}\ x = M\ \textbf{in}\ \bullet \quad \textit{if}\ p \notin M$

answering contexts produce values

$E_P ::= \bullet \mid F[E_P] \mid A[E_P]$

$$F[\,A[\,M\,]\,] \longrightarrow A[\,F[\,M\,]\,]$$

$$\begin{array}{rcll}
\text{if } (\text{let } x = M_1 \text{ in } M_2) \text{ then } M_3 \text{ else } M_4 & \mapsto & \text{let } x = M_1 \text{ in } (\text{if } M_2 \text{ then } M_3 \text{ else } M_4) & \textit{(if-let)} \\
\text{let } y = (\text{let } x = M_1 \text{ in } M_2) \text{ in } M_3 & \mapsto & \text{let } x = M_1 \text{ in } (\text{let } y = M_2 \text{ in } M_3) & \textit{(let-let)} \\
V\ (\text{let } x = M_1 \text{ in } M_2) & \mapsto & \text{let } x = M_1 \text{ in } V\ M_2 & \textit{(app-let)} \\
(\text{let } x = M_1 \text{ in } M_2)\ M_3 & \mapsto & \text{let } x = M_1 \text{ in } M_2\ M_3 & \textit{(let-app)} \\
V\ (\text{select}_{p,q}\ l\ M) & \mapsto & \text{select}_{p,q}\ l\ (V\ M) & \textit{(app-sel)} \\
(\text{select}_{p,q}\ l\ M_1)\ M_2 & \mapsto & \text{select}_{p,q}\ l\ (M_1\ M_2) & \textit{(sel-app)} \\
\text{if } (\text{select}_{p,q}\ l\ M_1) \text{ then } M_2 \text{ else } M_3 & \mapsto & \text{select}_{p,q}\ l\ (\text{if } M_1 \text{ then } M_2 \text{ else } M_3) & \textit{(if-sel)} \\
\text{let } x = \text{select}_{p,q}\ l\ M_1 \text{ in } M_2 & \mapsto & \text{select}_{p,q}\ l\ (\text{let } x = M_1 \text{ in } M_2) & \textit{(let-sel)}
\end{array}$$


A Call-By-Need Lambda Calculus

Zena M. Ariola
Computer & Information Science Department
University of Oregon
Eugene, Oregon

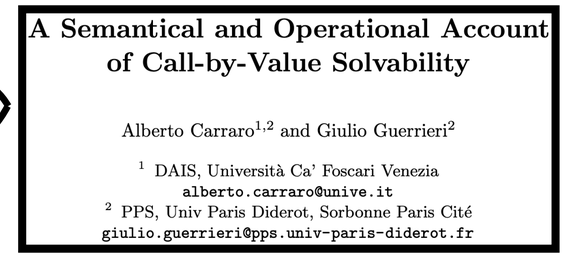
Matthias Felleisen
Department of Computer Science
Rice University
Houston, Texas

John Maraist and *Martin Odersky*
Institut für Programmstrukturen
Universität Karlsruhe
Karlsruhe, Germany

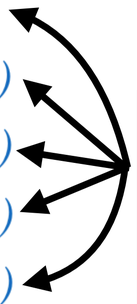
Philip Wadler
Department of Computing Science
University of Glasgow
Glasgow, Scotland

$$
\begin{array}{rcll}
\text{if } (\text{let } x = M_1 \text{ in } M_2) \text{ then } M_3 \text{ else } M_4 & \mapsto & \text{let } x = M_1 \text{ in } (\text{if } M_2 \text{ then } M_3 \text{ else } M_4) & \textit{(if-let)} \\
\text{let } y = (\text{let } x = M_1 \text{ in } M_2) \text{ in } M_3 & \mapsto & \text{let } x = M_1 \text{ in } (\text{let } y = M_2 \text{ in } M_3) & \textit{(let-let)} \\
V\ (\text{let } x = M_1 \text{ in } M_2) & \mapsto & \text{let } x = M_1 \text{ in } V\ M_2 & \textit{(app-let)} \\
(\text{let } x = M_1 \text{ in } M_2)\ M_3 & \mapsto & \text{let } x = M_1 \text{ in } M_2\ M_3 & \textit{(let-app)} \\
V\ (\text{select}_{p,q}\ l\ M) & \mapsto & \text{select}_{p,q}\ l\ (V\ M) & \textit{(app-sel)} \\
(\text{select}_{p,q}\ l\ M_1)\ M_2 & \mapsto & \text{select}_{p,q}\ l\ (M_1\ M_2) & \textit{(sel-app)} \\
\text{if } (\text{select}_{p,q}\ l\ M_1) \text{ then } M_2 \text{ else } M_3 & \mapsto & \text{select}_{p,q}\ l\ (\text{if } M_1 \text{ then } M_2 \text{ else } M_3) & \textit{(if-sel)} \\
\text{let } x = \text{select}_{p,q}\ l\ M_1 \text{ in } M_2 & \mapsto & \text{select}_{p,q}\ l\ (\text{let } x = M_1 \text{ in } M_2) & \textit{(let-sel)}
\end{array}
$$

$$\begin{array}{rcll}
\text{if } (\text{let } x = M_1 \text{ in } M_2) \text{ then } M_3 \text{ else } M_4 & \mapsto & \text{let } x = M_1 \text{ in } (\text{if } M_2 \text{ then } M_3 \text{ else } M_4) & \textit{(if-let)} \\
\text{let } y = (\text{let } x = M_1 \text{ in } M_2) \text{ in } M_3 & \mapsto & \text{let } x = M_1 \text{ in } (\text{let } y = M_2 \text{ in } M_3) & \textit{(let-let)} \\
V\ (\text{let } x = M_1 \text{ in } M_2) & \mapsto & \text{let } x = M_1 \text{ in } V\ M_2 & \textit{(app-let)} \\
(\text{let } x = M_1 \text{ in } M_2)\ M_3 & \mapsto & \text{let } x = M_1 \text{ in } M_2\ M_3 & \textit{(let-app)} \\
V\ (\text{select}_{p,q}\ l\ M) & \mapsto & \text{select}_{p,q}\ l\ (V\ M) & \textit{(app-sel)} \\
(\text{select}_{p,q}\ l\ M_1)\ M_2 & \mapsto & \text{select}_{p,q}\ l\ (M_1\ M_2) & \textit{(sel-app)} \\
\text{if } (\text{select}_{p,q}\ l\ M_1) \text{ then } M_2 \text{ else } M_3 & \mapsto & \text{select}_{p,q}\ l\ (\text{if } M_1 \text{ then } M_2 \text{ else } M_3) & \textit{(if-sel)} \\
\text{let } x = \text{select}_{p,q}\ l\ M_1 \text{ in } M_2 & \mapsto & \text{select}_{p,q}\ l\ (\text{let } x = M_1 \text{ in } M_2) & \textit{(let-sel)}
\end{array}$$



$$
\begin{aligned}
\text{if } (\text{let } x = M_1 \text{ in } M_2) \text{ then } M_3 \text{ else } M_4 &\mapsto \text{let } x = M_1 \text{ in } (\text{if } M_2 \text{ then } M_3 \text{ else } M_4) && \textit{(if-let)}\\
\text{let } y = (\text{let } x = M_1 \text{ in } M_2) \text{ in } M_3 &\mapsto \text{let } x = M_1 \text{ in } (\text{let } y = M_2 \text{ in } M_3) && \textit{(let-let)}\\
V\ (\text{let } x = M_1 \text{ in } M_2) &\mapsto \text{let } x = M_1 \text{ in } V\ M_2 && \textit{(app-let)}\\
(\text{let } x = M_1 \text{ in } M_2)\ M_3 &\mapsto \text{let } x = M_1 \text{ in } M_2\ M_3 && \textit{(let-app)}\\
V\ (\text{select}_{\text{p,q}}\ l\ M) &\mapsto \text{select}_{\text{p,q}}\ l\ (V\ M) && \textit{(app-sel)}\\
(\text{select}_{\text{p,q}}\ l\ M_1)\ M_2 &\mapsto \text{select}_{\text{p,q}}\ l\ (M_1\ M_2) && \textit{(sel-app)}\\
\text{if } (\text{select}_{\text{p,q}}\ l\ M_1) \text{ then } M_2 \text{ else } M_3 &\mapsto \text{select}_{\text{p,q}}\ l\ (\text{if } M_1 \text{ then } M_2 \text{ else } M_3) && \textit{(if-sel)}\\
\text{let } x = \text{select}_{\text{p,q}}\ l\ M_1 \text{ in } M_2 &\mapsto \text{select}_{\text{p,q}}\ l\ (\text{let } x = M_1 \text{ in } M_2) && \textit{(let-sel)}
\end{aligned}
$$



Modular Compilation for Higher-Order Functional Choreographies

Luís Cruz-Filipe
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

Eva Graversen
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

Lovro Lugović
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

Fabrizio Montesi
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

Marco Peressotti
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

✅ simplified rules!

$$\text{if } (\text{let } x = M_1 \text{ in } M_2) \text{ then } M_3 \text{ else } M_4 \mapsto \text{let } x = M_1 \text{ in } (\text{if } M_2 \text{ then } M_3 \text{ else } M_4) \quad \textit{(if-let)}$$

$$\text{let } y = (\text{let } x = M_1 \text{ in } M_2) \text{ in } M_3 \mapsto \text{let } x = M_1 \text{ in } (\text{let } y = M_2 \text{ in } M_3) \quad \textit{(let-let)}$$

$$V\ (\text{let } x = M_1 \text{ in } M_2) \mapsto \text{let } x = M_1 \text{ in } V\ M_2$$

$$\frac{x \notin \mathrm{fv}(M') \quad \mathrm{spn}(M') \cap \mathrm{pn}(N) = \varnothing}{M'\ ((\lambda x : T.M)\ N) \rightsquigarrow (\lambda x : T.(M'\ M))\ N} \text{ [R-AbsL]}$$

$$(\text{let } x = M_1 \text{ in } M_2)\ M_3 \mapsto \text{let } x = M_1 \text{ in } M_2\ M_3 \quad \textit{(let-app)}$$

$$V\ (\text{select}_{p,q}\ l\ M) \mapsto \text{select}_{p,q}\ l\ (V\ M) \quad \textit{(app-sel)}$$

$$(\text{select}_{p,q}\ l\ M_1)\ M_2 \mapsto \text{select}_{p,q}\ l\ (M_1\ M_2) \quad \textit{(sel-app)}$$

$$\text{if } (\text{select}_{p,q}\ l\ M_1) \text{ then } M_2 \text{ else } M_3 \mapsto \text{select}_{p,q}\ l\ (\text{if } M_1 \text{ then } M_2 \text{ else } M_3) \quad \textit{(if-sel)}$$

$$\text{let } x = \text{select}_{p,q}\ l\ M_1 \text{ in } M_2 \mapsto \text{select}_{p,q}\ l\ (\text{let } x = M_1 \text{ in } M_2) \quad \textit{(let-sel)}$$

✅ **simplified rules!**

✅ **found bugs in the old model!**

$$\text{if } (\text{let } x = M_1 \text{ in } M_2) \text{ then } M_3 \text{ else } M_4 \mapsto \text{let } x = M_1 \text{ in } (\text{if } M_2 \text{ then } M_3 \text{ else } M_4) \quad \textit{(if-let)}$$

$$\text{let } y = (\text{let } x = M_1 \text{ in } M_2) \text{ in } M_3 \mapsto \text{let } x = M_1 \text{ in } (\text{let } y = M_2 \text{ in } M_3) \quad \textit{(let-let)}$$

$$V\ (\text{let } x = M_1 \text{ in } M_2) \mapsto \text{let } x = M_1 \text{ in } V\ M_2 \quad \textit{(app-let)}$$

$$(\text{let } x = M_1 \text{ in } M_2)\ M_3 \mapsto \text{let } x = M_1 \text{ in } M_2\ M_3 \quad \textit{(let-app)}$$

$$V\ (\text{select}_{p,q}\ l\ M) \mapsto \text{select}_{p,q}\ l\ (V\ M) \quad \textit{(app-sel)}$$

$$(\text{select}_{p,q}\ l\ M_1)\ M_2 \mapsto \text{select}_{p,q}\ l\ (M_1\ M_2) \quad \textit{(sel-app)}$$

$$\text{if } (\text{select}_{p,q}\ l\ M_1) \text{ then } M_2 \text{ else } M_3 \mapsto \text{select}_{p,q}\ l\ (\text{if } M_1 \text{ then } M_2 \text{ else } M_3) \quad \textit{(if-sel)}$$

$$\text{let } x = \text{select}_{p,q}\ l\ M_1 \text{ in } M_2 \mapsto \text{select}_{p,q}\ l\ (\text{let } x = M_1 \text{ in } M_2) \quad \textit{(let-sel)}$$

wrapping up

the semantics isn't arbitrary—it's **semilenient!**

there's a **recipe** and **laws** for making choreographic languages

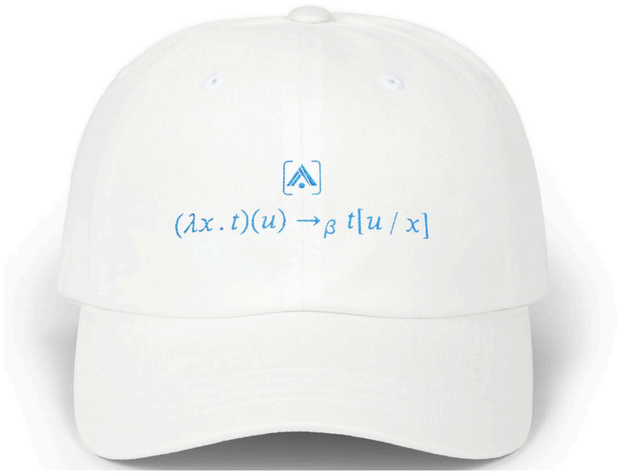
can we extract a projection function?

does it generalize?

can we automate it?

learn more:

dplyukhin.github.io



support our podcast:

store.typetheoryforall.com

*on the job market