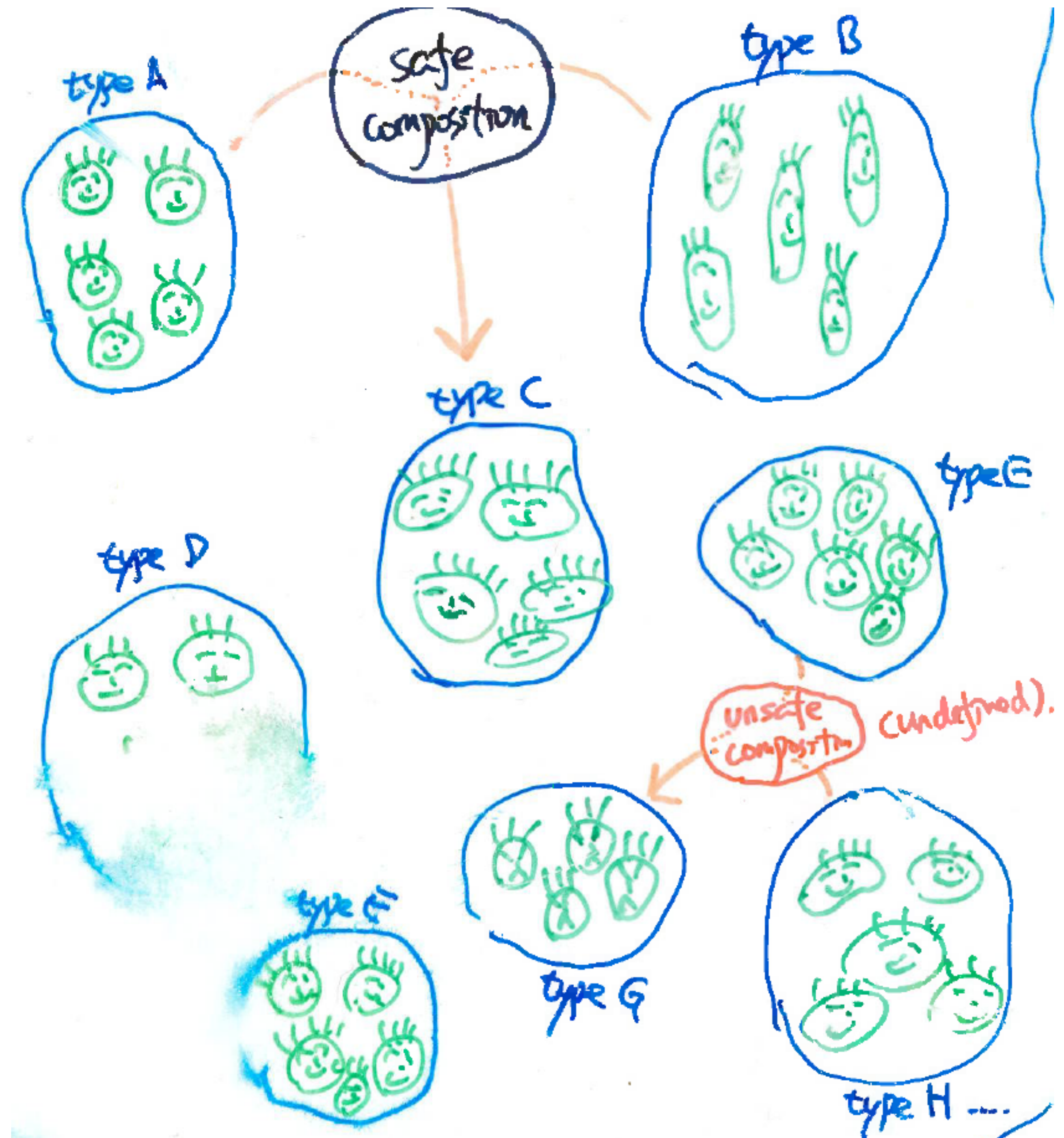


# Idioms for Interaction

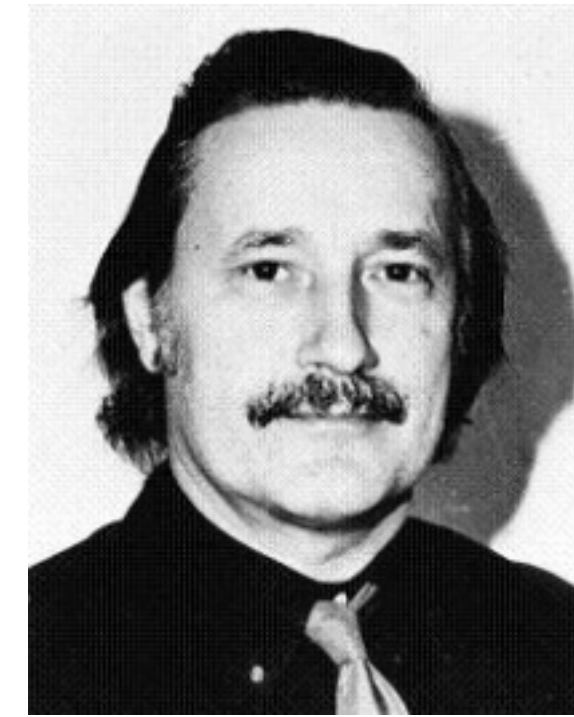
Concurrency Workshop, INI, 11th August 2022  
Nobuko Yoshida

Imperial College  
London



# Idioms for Interaction

## from Conventional to Concurrent/Distributed Programming



1916-1975

- **Christopher Strachey** (sequential computation)
  - ▶ **Types** = abstract computation (data types, polymorphism)
  - ▶ **Structured programming = High-level programming**
- **Session types** (concurrency & communication)
  - ▶ Structured programming = **protocols**



# ETAPS 2019 TEST-OF-TIME AWARD

---

[Language primitives and type discipline for structured communication-based programming](#) by [Kohei Honda](#),  
Vasco T. Vasconcelos and Makoto Kubo

# POPL 2008 MOST INFLUENTIAL PAPER AWARD



POPL 2008 Most Influential Paper Award

Kohei Honda, Nobuko Yoshida and Marco Carbone

Multiparty asynchronous session types





# LICS 2018 TEST OF TIME AWARD

LICS'98

## A fully abstract game semantics for general references

Samson Abramsky      Kohei Honda

LFCS, University of Edinburgh

{samson, kohei}@dcs.ed.ac.uk

Guy McCusker

St John's College, Oxford

mccusker@comlab.ox.ac.uk

### Abstract

*A games model of a programming language with higher-order store in the style of ML-references is introduced. The category used for the model is obtained by relaxing certain behavioural conditions on a category of games previously used to provide fully abstract models of pure functional languages. The model is shown to be fully abstract by means of factorization arguments which reduce the question of definability for the language with higher-order store to that for its purely functional fragment.*

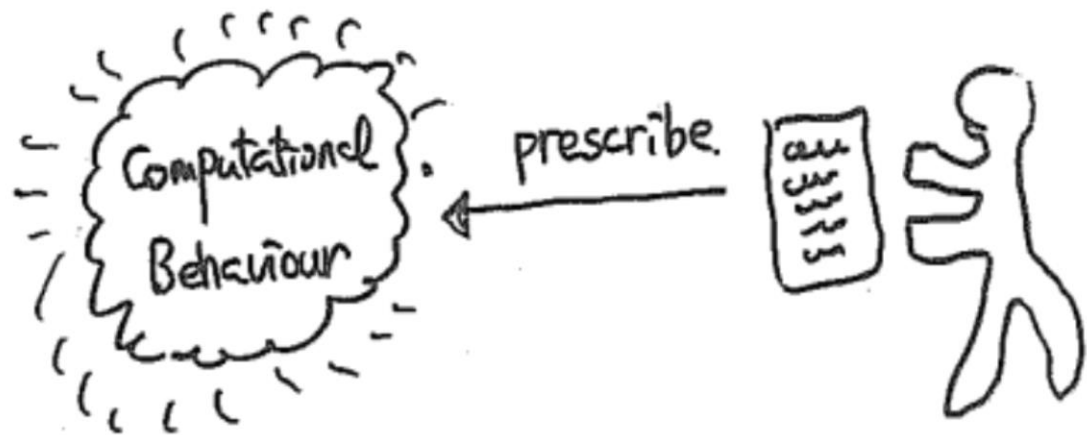
# Idioms for Interaction

— Invitation to Hacking in  $\pi$ -calculus —

Programming languages are tools which offer frameworks of abstraction for such activities – promoting or limiting them

- Imperative
- Functional
- Logical

- Programs : prescription of computational behaviours based on a certain abstraction.



# On Programs and Programming

- The most fundamental element of a PL in this context is a set of operations it is based on:

Imperative: assignment, jump.

Functional:  $\beta$ -reduction.

Logical: unification.

- Another element is how we can combine, on structure, these operations:

Imperative: sequential composition, if-then-else, while, procedures, module, ....

Functional: application, product, union, recursion, modules, .....

# UNSTRUCTURED:

```

data _stkl, 48 } stacks.
data _stkr, 48 }
data _i, 4 } index
data _j, 4 }
data _l, 4 } left/right limit
data _r, 4 }
data _x, 4 } pivot
data _w, 4 } temporary value.
data _s, 4 } stack pointer
data _a, 48 } table to be sorted.

mov $0, _s
mov $0, _stkl } 2nd instruction
mov $11, _stkr

L1: mov _s, rax } Top loop.
    mov _stkl(, rax, 4), rcx } l := stkl(s)
    mov rcx, _l
    mov _s, rax
    mov _stkr(, rax, 4), rcx } r := stkr(s)
    mov rcx, _r
    dec _s

L2: mov _l, rcx } j := l.
    mov rcx, _i
    mov _r, rcx } j := r.
    mov rcx, _j
    mov _l, rcx } rdx := l+r
    add _r, rcx
    mov rcx, rax } rdx := (l+r)/2
    mov rax, rdx
    shr $31, rdx
    add rdx, rax
    mov rax, rdx
    sar $1, rdx
    mov _a(, rdx, 4), rcx } x := a(rcx)
    mov rcx, _x

L3: mov _i, rax } third loop
    mov _a(, rax, 4), rdx } rdx := a(i)
    cmp rdx, _x
    jle L4
    inc _i
    jmp L3 } if rdx >= x goto (4)
    } else i := i+1
    } goto (3) (loop)

L4: mov _j, rax
    mov _a(, rax, 4), rdx } rdx := a(j)

```

# STRUCTURED:

```

Var a: array[MAX] of int;

Procedure sort(l, r: int);
  Var i, j, x: int;
  i := l; j := r;
  x := [(l+r) div 2];
  • Choose a pivot.
  repeat
    while a[i] < x do i := i+1 end
    while a[j] > x do j := j-1 end
    if i < j then swap(i, j); i := i+1; j := j-1; end
  until i > j;
  if l < j then sort(l, j);
  if l < i then sort(i, r);
  end
  } Partition into two parts.
  } Recursively sort two parts.

Procedure swap(i, j: int)
  Var w: int;
  w := a[i]; a[i] := a[j]; a[j] := w;
end

```

# Quicksort in pure lambda:

$((\lambda xy. y(xy))(\lambda xy. y(xy)))\lambda q. \lambda l.$   
 $((\lambda x. x(\lambda xy. x))l)(\lambda x. x)$  } if l is not then nil.  
 $((\lambda xy. y(xy))(\lambda xy. y(xy)))(\lambda c. \lambda xy. x((\lambda x. x(\lambda py. x))x)y$  } concat.  
 $((\lambda xy. \lambda z. z(\lambda xy. y)xy)((\lambda x. x(\lambda xyz. y))x)(c((\lambda x. x(\lambda xyz. z))x)y$   
 $(q(\lambda xy. y(xy))(\lambda xy. y(xy)))(\lambda f. \lambda px. ((\lambda x. x(\lambda py. x))x)(\lambda x. x))$  } sort and  
 $(p((\lambda x. x(\lambda xyz. y))x))((\lambda xy. \lambda z. z(\lambda xy. y)xy)x$  Filter  
 $(f((\lambda x. x(\lambda xyz. z))x)))(f((\lambda x. x(\lambda xyz. z))x)))$   
 $(\lambda y. (((\lambda xy. y(xy))(\lambda xy. y(xy)))\lambda f'. \lambda xy. ((\lambda x. x(\lambda py. x))x)y$  }  $(\lambda y. LT y. Car$   
 $(\lambda xy. y)((\lambda x. x(\lambda py. x))x)(\lambda xy. y)(f'((\lambda x. x(\lambda py. y))x)((\lambda x. x(\lambda py. y)$   
 $y((\lambda x. x(\lambda xyz. y))l))((\lambda x. x(\lambda xyz. z))l))$  } cdr l  
 $((\lambda xy. \lambda z. z(\lambda xy. y)xy)((\lambda x. x(\lambda xyz. y))l)$  } cons (car l)  
 $(q((\lambda xy. y(xy))(\lambda xy. y(xy)))(\lambda f. \lambda px. ((\lambda x. x(\lambda py. x))x)$  } sort and  
 $(\lambda x. x)(p((\lambda x. x(\lambda xyz. y))x))((\lambda xy. \lambda z. z(\lambda xy. y)xy)x$  Filter  
 $(f((\lambda x. x(\lambda xyz. z))x)))(f((\lambda x. x(\lambda xyz. z))x)))$   
 $(\lambda y. (((\lambda xy. y(xy))(\lambda xy. y(xy)))\lambda f''. \lambda xy. ((\lambda x. x(\lambda py. x))x)$  }  $\lambda y. ME y$   
 $((\lambda x. x(\lambda py. x))y)(\lambda xy. x)(\lambda xy. y)$  Car l  
 $((\lambda x. x(\lambda py. x))y)(\lambda xy. x)(f''((\lambda x. x(\lambda py. y))x)$   
 $((\lambda x. x(\lambda py. y))y))y((\lambda x. x(\lambda xyz. y))l)((\lambda x. x(\lambda xyz. z))l))$  } cdr l

# Quicksort with combinators:

$Y(\lambda f. \lambda l.$   
 $(Isnil l)$   
 $(Concat (f Filter (\lambda y. LT y (Car l))$   
 $(Cdr l))$   
 $(Cons (Car l)$   
 $(f Filter (\lambda y. ME y$   
 $(Car l)))))$

$I = \lambda x. x$     $T = \lambda xy. x$     $F = \lambda xy. y$     $Y = (\lambda xy. y(xy))(\lambda xy. y(xy))$   
 $Cons = \lambda xy. \lambda z. z F xy$     $Isnil = \lambda x. x T$   
 $Car = \lambda x. x (\lambda yz. y)$     $Cdr = \lambda x. x (\lambda yz. z)$   
 $Concat = Y (\lambda c. \lambda xy. x (Isnil x) y (Cons (Car x) (c (Cdr x) y))$   
 $Filter = Y (\lambda f. \lambda px. (Isnil x) I (p (Car x)) (Cons x (Filter (Cdr x) )))$   
 $Iszero = \lambda x. x T$     $Pred = \lambda x. x F$     $(Filter (Cdr x))$   
 $LT = Y (\lambda f. \lambda xy. (Iszero y) F ((Iszero x) F (f (Pred x) (Pred y)$   
 $ME = Y (\lambda f. \lambda xy. (Iszero x) ((Iszero y) I F) ((Iszero y) (T) y))$   
 $(f (Pred x) (Pred y))$

## Quicksort in ML:

```
fun qs nil: int list = nil
```

```
| qs (x::r) = let val small =  
                filter (fn y => y < x) r  
                and large =  
                filter (fn y => y >= x) r
```

```
    in qs small @ [x] @ qs large
```

```
    end
```

```
fun filter p nil = nil
```

```
| filter p (x::r) =
```

```
    if p x then x := filter p r
```

```
    else filter p r
```

# The $\pi$ -calculus as a Descriptive Tool

$$\lambda \quad M ::= x \mid \lambda x.M \mid MN.$$

$$\pi \quad P ::= \sum \pi_i.P_i \mid P|Q \mid \nu x.P \mid !P \mid \emptyset.$$

$$\text{with } \pi ::= x(y) \mid \bar{x}(y).$$

$\lambda$  in  $\pi$

$$[x]_u \stackrel{\text{def}}{=} \bar{x}(u).$$

$$[\lambda x.M]_u \stackrel{\text{def}}{=} u(x). [M]_u.$$

$$[MN]_u \stackrel{\text{def}}{=} (\nu f_x) ([M]_f \mid \bar{f}(x)) \mid [x=N]_u$$

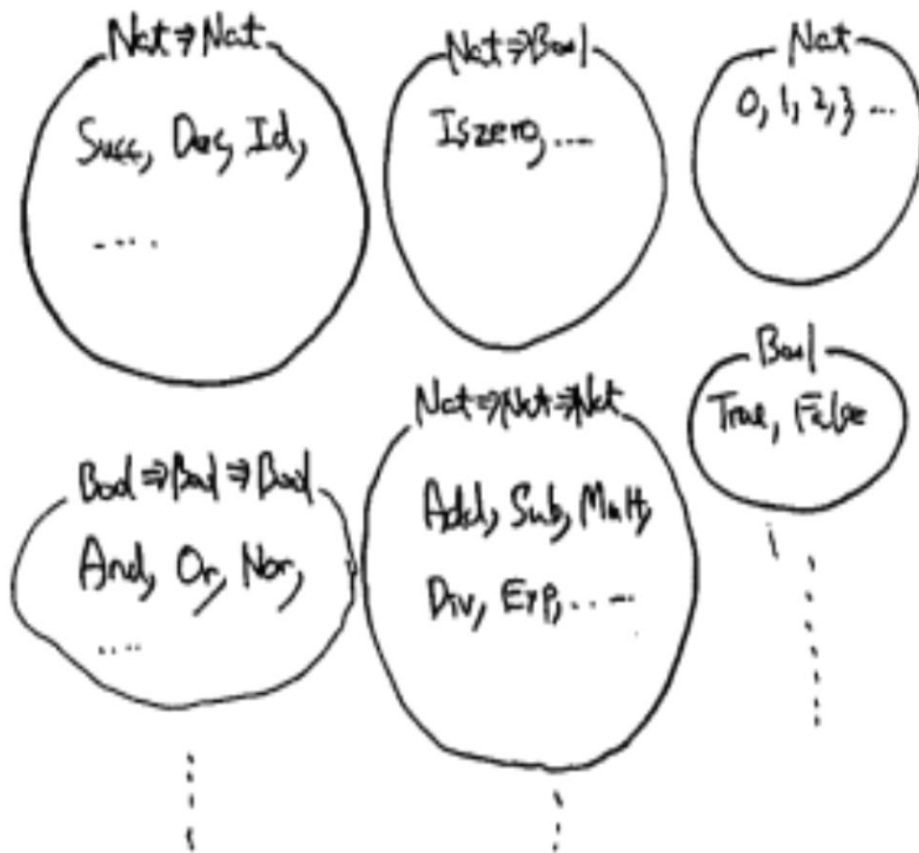
$$\text{with } [x=N]_u \stackrel{\text{def}}{=} !x(u). [N]_u.$$

# The Role of Types in $\pi$ -Calculus.

- (classification) How can we classify name-passing interactive behaviours, i.e. behaviours representable in  $\pi$ -calculus? What classes ("types") of behaviours can we find in the calculus?

- (safety) Is this program/system in the safe (or correct, relevant,...) classes of behaviours? Can the safety be preserved compositionally?

# Functional Types



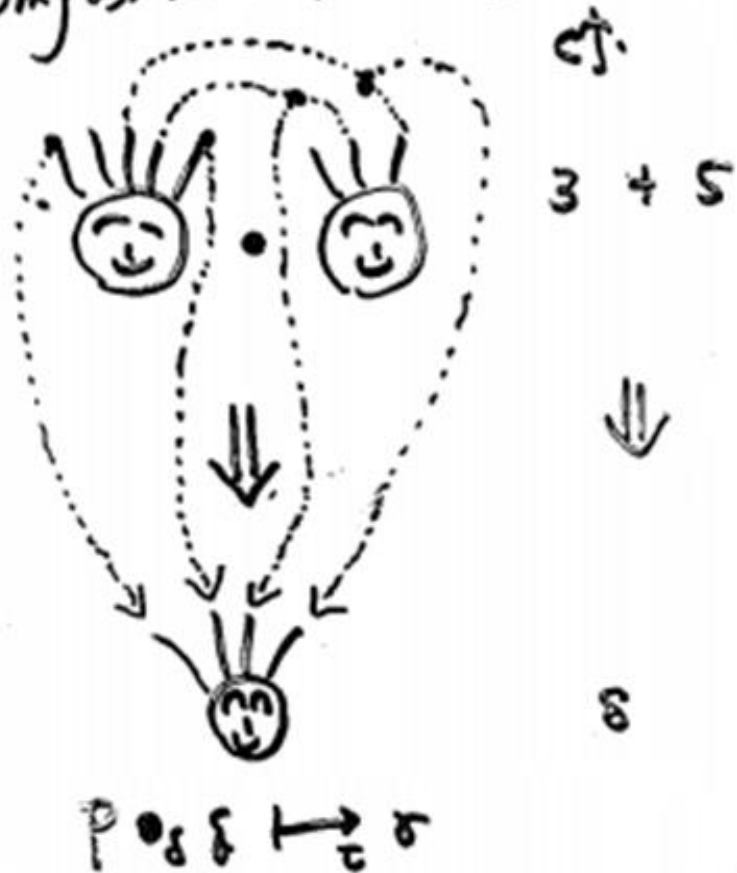
with operation!

$$\left\{ \begin{array}{l} f : \alpha \Rightarrow \beta \bullet e : \alpha = f \bullet e : \beta. \\ \text{else undefined.} \end{array} \right.$$

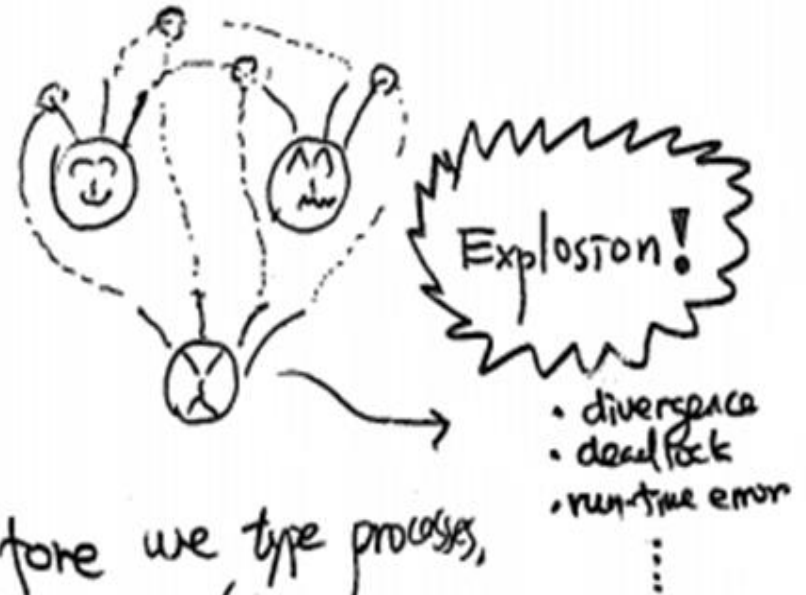
function application.

# Process Types

- When it comes to processes, composition becomes:



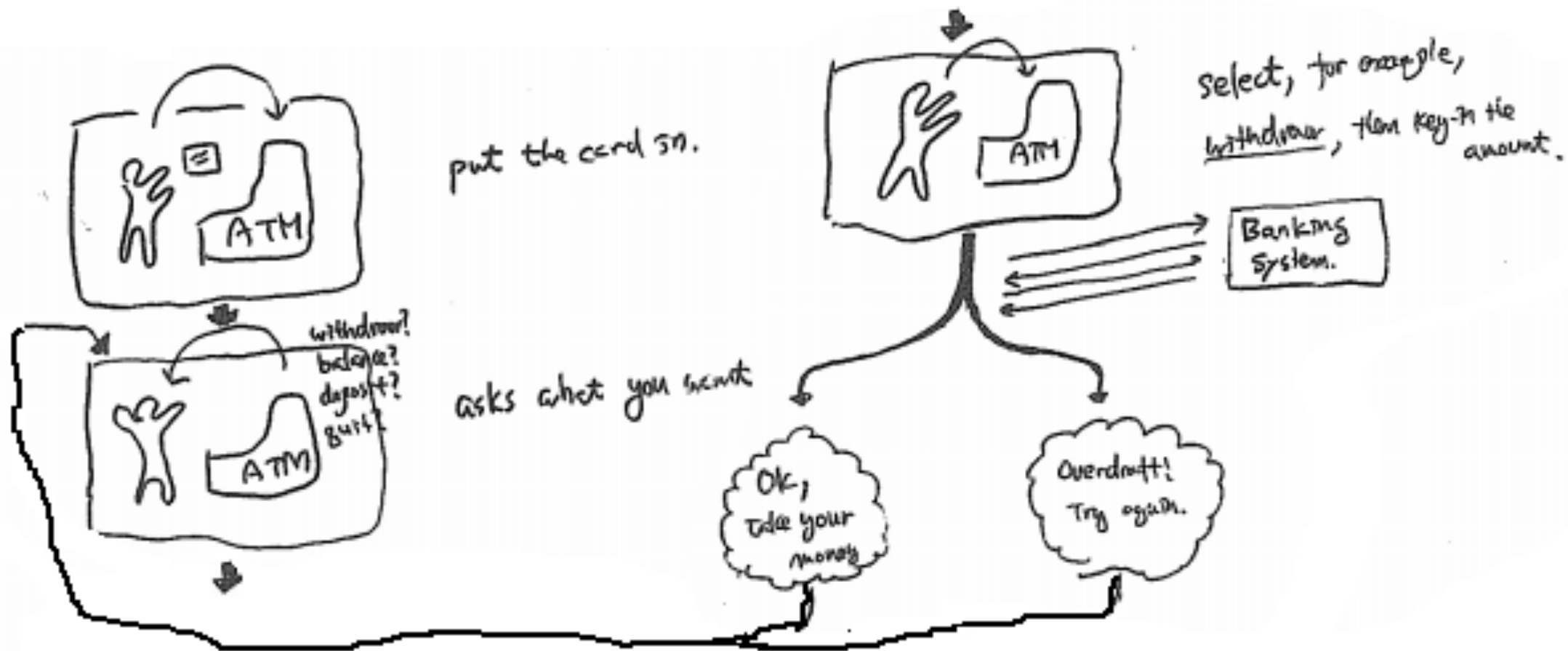
- But some composition is dangerous!



- Therefore we type processes,



# Implementing ATM



# Implementing ATM

$ATM^{cb} =$

$z \langle z \rangle :: z ? ID ; [ ? \underline{wd} ; ? X ;$	$\# \leftrightarrow \text{user.}$
$\bar{b} \langle w \rangle :: w ! \underline{wt} ; ! ID ; ! X )$	$* \leftrightarrow \text{bank.}$
$[ ? \underline{ok} ;$	$\# :$
$z ! \underline{ok} ; ! X ; \theta ATM^{cb} ,$	$\# \leftrightarrow \text{user}$
$? \underline{overdraft} :$	$\# \leftrightarrow \text{bank}$
$z ! \underline{over} ; ATM^{cb} ] ,$	$\# \leftrightarrow \text{user.}$
$? \underline{bal} ;$	$\# \leftrightarrow \text{user}$
$\bar{b} \langle w \rangle :: w ! \underline{bal} ; ? X ;$	$* \leftrightarrow \text{bank}$
$z ! X ; \theta ATM^{cb} ]$	

## ENCODING

$z \langle z \rangle :: z ? ID ; [ ? \underline{wd} ; ? X ;$

$\bar{b} \langle w \rangle :: w ! \underline{wt} ; ! ID ; ! X )$

$[ ? \underline{ok} ;$

$z ! \underline{ok} ; ! X ; \theta ATM^{cb} ,$

$? \underline{overdraft} :$

$z ! \underline{over} ; ATM^{cb} ] ,$

$? \underline{bal} ;$

$\bar{b} \langle w \rangle :: w ! \underline{bal} ; ? X ;$

$z ! X ; \theta ATM^{cb} ]$

$\bar{b} \langle w \rangle :: w ! \underline{wt} ; ! ID ; ! X )$   
 $[ ? \underline{ok} ;$   
 $z ! \underline{ok} ; ! X ; \theta ATM^{cb} ,$   
 $? \underline{overdraft} :$   
 $z ! \underline{over} ; ATM^{cb} ] ,$   
 $? \underline{bal} ;$   
 $\bar{b} \langle w \rangle :: w ! \underline{bal} ; ? X ;$   
 $z ! X ; \theta ATM^{cb} ]$

# Session Idioms: Sequencing, Branching and Selection

Intermediate Summary: Two Idioms.

(1) Sequential Non-Passing.

$$a: (x_1 \dots x_n). P \mid \bar{a}: [v_1 \dots v_n]. Q$$

$$\rightarrow \approx P[\sigma/x] \mid Q$$

(2) Branching/Selection.

$$a: [(\omega.P_1) \& (\omega.P_2)] \mid \bar{a}: \text{sel}(\omega). Q$$

$$\rightarrow \approx P_1[\sigma/x] \mid Q.$$

$$\text{cf. } (\lambda(x_1 \dots x_n). M) \langle N_1, \dots, N_n \rangle \rightarrow M[\bar{N}/\bar{x}]$$

$$\left( \begin{array}{l} \text{case } x \text{ of} \\ \text{inl}(y) \Rightarrow M_1 \\ \text{inr}(z) \Rightarrow M_2 \end{array} \right) \text{inl}(N) \rightarrow M_1[\bar{N}/\bar{x}]$$

- dyadicity.

- sharing/interference.

on Polymorphic  
and Sessions  
Functions

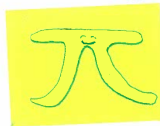
a Tale of Two  
Encodings

Bernardo Toninho @Nova

Nobuko Yoshida @Imperial

# on Polymorphic Sessions

and Functions



a Tale of Two

Fully Abstract

Encodings

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

# on Polymorphic Sessions

and Functions



a Tale of Two Fully Abstract Encodings

Bernardo Toninho @ Nova

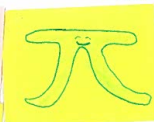
Nobuko Yoshida @ Imperial

on Polymorphic

Sessions

and

Function



a Tale

of Two

Encodings

Bernardo Toninho @ Nova

Nobuko Yoshida @ Imperial

# The $\pi$ -calculus as a Descriptive Tool

by Kohei  
Honda  
1995

$$\lambda \quad M ::= x \mid \lambda x.M \mid MN.$$

$$\pi \quad P ::= \sum \pi_i.P_i \mid P|Q \mid \langle \nu a \rangle P \mid !P \mid \emptyset.$$

$$\text{with } \pi ::= x \langle \bar{a} \rangle \mid \bar{x} \langle a \rangle.$$

Milner's  
Encoding  
1991

$\lambda$  in  $\pi$

$$[x]_u \triangleq \bar{x}(u).$$

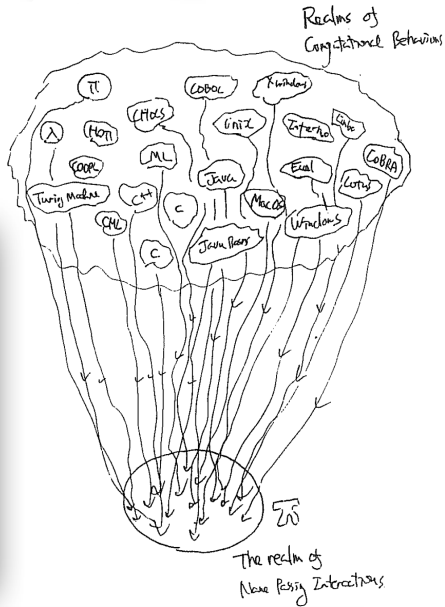
$$[\lambda x.M]_u \triangleq u(x).[M]_u.$$

$$[MN]_u \triangleq (vz) ([M]_y \mid \bar{z}(u) \mid [z=N]_y)$$

$$\text{with } [z=N]_y \triangleq !z(y).[N]_y.$$

# \* Examples of Representable Computation.

- $\lambda$ -calculus [MPW83, Milner 90, Milner 92, ...]
  - Concurrent Object [Walker 81]
  - $\omega$ -order term passing [Sangiorgi 92]
  - Various data structures [Milner 92, ...]
  - Proof Nets [Bellare and Scott 93]
  - Abstract 'constant' interaction [HY94]
  - Strategies on Games [HO95]
- ⋮



# \* Examples of Representable Computation.

Operationally

Sound

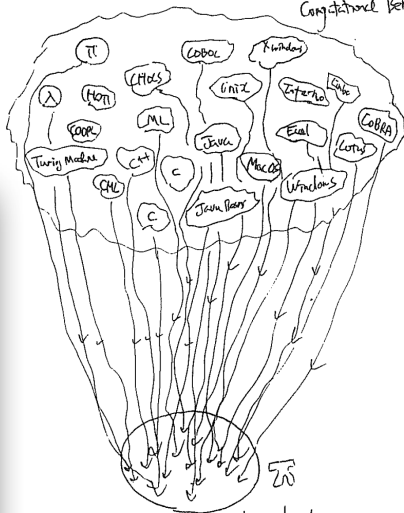
but NOT

Fully Abstract

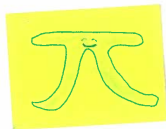
- $\lambda$ -calculus [MPW89, Milner 90, Milner 92, ...]
- Concurrent Object [Walker 81]
- $\omega$ -order term passing [Sangiorgi 92]
- Various data structures [Milner 92, ...]
- Proof Nets [Bellare and Scott 93]
- Abstract 'constant' interaction [HY94]
- Strategies on Games [HO95]

⋮

Realms of  
Computational Behaviors



The realm of  
Non Passing Interactions



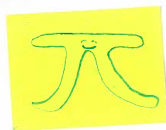
$M \approx N$



$\llbracket M \rrbracket \approx \llbracket N \rrbracket$

Contextual  
Congruence

Contextual  
Congruence



$M \approx N$



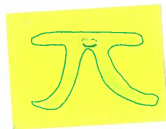
$\llbracket M \rrbracket \approx \llbracket N \rrbracket$

Contextual  
Congruence

Contextual  
Congruence

$C[P] \Downarrow_a \text{ iff } C[Q] \Downarrow_a$

$C[\llbracket M \rrbracket] \Downarrow_a \text{ iff } C[\llbracket N \rrbracket] \Downarrow_a$



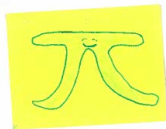
$M \approx N$



$\llbracket M \rrbracket \approx \llbracket N \rrbracket$

$C[P] \Downarrow_a \text{ iff } C[Q] \Downarrow_a$

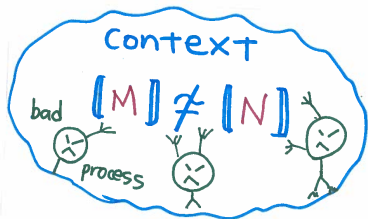
$C[\llbracket M \rrbracket] \Downarrow_a \text{ iff } C[\llbracket N \rrbracket] \Downarrow_a$



$M \approx N$

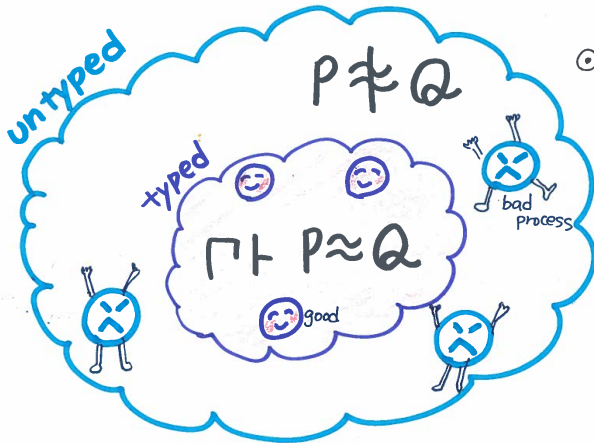


$[M] \approx [N]$



# Typed Semantics in $\pi$ 1991 $\rightarrow$

IO-subtyping, Linear types, Secure Information Flow, ...



- ⊙ Correctness of Encoding
- ⊙ Limit environment  $\Pi$   
 $\Rightarrow$  Equate more processes
- ⊙ Compositional

# \* Examples of Representable Computation.

- $\lambda$ -calculus [MPW89, Milner 90, Milner 92, ...]
- Concurrent Object [Walker 81]
- low-order term passing [Sangiorgi 92]
- Various data structures [Milner 92, ...]
- Proof Nets [Bellare and Scott 93]
- Abstract 'constant' interaction [HY94]
- strategies on Games [HO95]
- ...

Game Semantics



Fully Abstract

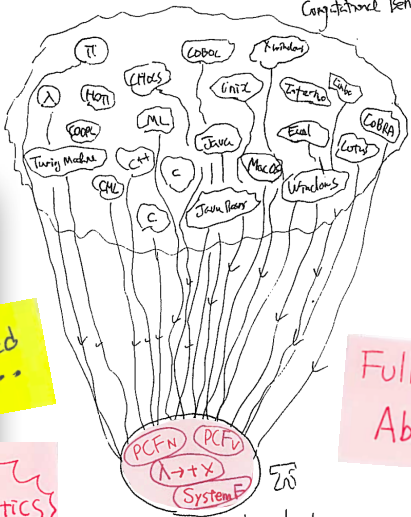
# \* Examples of Representable Computation.

- $\lambda$ -calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker81]
- $\omega$ -order term passing [Sangiorgi 92]
- Various data structures [Milner90]
- Proof Nets [Bellare and Scott 93]
- Abstract 'constant' interaction [Milner90]
- Strategies on Games [HO95]

Complicated  
...

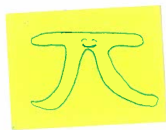
Game Semantics

Realms of Computational Behaviors



Fully Abstract

The realm of Name Passing Interactions



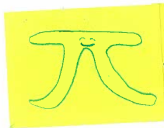
$M \approx N$



$[M] \approx [N]$



System



Session



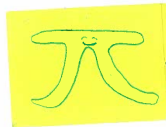
$M \approx N$



$\llbracket M \rrbracket \approx \llbracket N \rrbracket$



System



Session



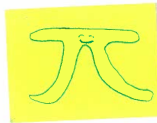
$M \approx N$



$\llbracket M \rrbracket \approx \llbracket N \rrbracket$



System



Session



$$M \approx N$$

$$[[M]] \approx [[N]]$$

$$[[P]] \approx [[Q]]$$

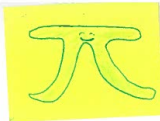


$$P \approx Q$$

Reverse



System  
F



Session



$$M \approx N$$

$$\llbracket M \rrbracket \approx \llbracket N \rrbracket$$

$$\llbracket P \rrbracket \approx \llbracket Q \rrbracket$$



$$P \approx Q$$

Reverse  
New

# Session $\pi$ Caires, Pérez, Pfenning, Toninho 13

## Types

$$A ::= \underline{A \otimes B} \mid \underline{A \multimap B} \mid \underline{1} \mid \underline{!A}$$
$$\mid \underline{\forall x. A} \mid \underline{\exists x. A} \mid X$$

## Processes

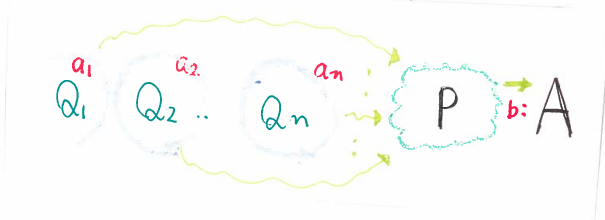
$$P ::= \underline{x \langle y \rangle. P} \mid \underline{x(y). P} \mid \underline{0} \mid \underline{!x(y). P}$$
$$\mid \underline{x(Y). P} \mid \underline{x \langle B \rangle. P} \mid [x \leftrightarrow y]$$
$$\mid (\nu x) P \mid (P \mid Q)$$

# Judgement

$$X_1, \dots, X_k ; a_1 : A_1, \dots, a_n : A_n \vdash P :: b : A$$

Annotations:  
-  $X_1, \dots, X_k$ : Poly Vars  
-  $a_1, \dots, a_n$ : name  
-  $A_1, \dots, A_n$ : Type  
-  $P$ : Process  
-  $b$ : name  
-  $A$ : type

process  $P$  provides  $A$  along  $b$  if composed with sessions  $\vec{a} : \vec{A}$



# Judgement

$$\underbrace{X_1, \dots, X_k}_{\text{Poly Vars}} ; \underbrace{a_1: A_1, \dots, a_n: A_n}_{\substack{\uparrow \\ \text{name} \quad \text{Type}}} \vdash \underbrace{P}_{\text{Process}} :: \underbrace{b: A}_{\substack{\uparrow \\ \text{name} \quad \text{type}}}$$

process  $P$  provides  $A$  along  $b$  if composed with sessions  $\vec{a}: \vec{A}$



# Judgement

$$X_1, \dots, X_k ; a_1 : A_1, \dots, a_m : A_m \vdash P :: b : A$$

Diagram annotations:  
-  $X_1, \dots, X_k$  is underlined and labeled "Poly Vars".  
-  $a_1 : A_1, \dots, a_m : A_m$  is labeled "name" (pointing to  $a_i$ ) and "Type" (pointing to  $A_i$ ).  
-  $P$  is labeled "Process".  
-  $b : A$  is labeled "name" (pointing to  $b$ ) and "type" (pointing to  $A$ ).

## Cut Elimination

$$\frac{\Delta_1 \vdash P_1 :: a : A \quad \Delta_2, a : A \vdash P_2 :: b : B}{\Delta_1, \Delta_2 \vdash (va) (P_1 | P_2) :: b : B}$$

## Identity

$$a : A \vdash [a \leftrightarrow b] :: b : A$$

# Polymorphic Session

$$\forall R \quad \frac{x; \Delta \vdash P :: a:A}{\Delta \vdash a(X).P :: a:\forall X.A} \quad \text{Input}$$

$$\exists R \quad \frac{\Delta \vdash P :: a:A\{B/X\}}{\Delta \vdash a\langle B \rangle.P :: a:\exists X.A} \quad \text{output}$$

Left rules define how to **use** a session of a given type

# Polymorphic Session

Deadlock  
Free  
Live

Strong  
Normalising

$$\forall R \quad \frac{x; \Delta \vdash P :: a:A}{\Delta \vdash a(X).P :: a:\forall X.A}$$

Input

$$\Delta \vdash P :: a:A\{B/X\}$$

$$\Delta \vdash a\langle B \rangle.P :: a:\exists X.A$$

output

$\approx \pi$

Barbed  
Congruence

Left rules define how to **use** a session of a given type

# Linear F

zhao, Zhang, Zdancewicz 2010



Types

$$A ::= A \otimes B \mid A \multimap B \mid !A \mid 1 \mid 2 \\ \mid \forall x. A \mid \exists x. A \mid X$$

Terms

$$M, N ::= \lambda x. M \mid MN \mid \langle M \otimes N \rangle \mid \text{let } x \otimes y = M \text{ in } N \\ \mid !M \mid \text{let } !u = M \text{ in } N \\ \mid \Lambda X. M \mid M[A] \mid \text{pack } A \text{ with } M \mid \text{let } (X, y) = M \\ \text{in } N \\ \mid \text{let } \mathbf{1} = M \text{ in } N \mid \langle \rangle \mid \mathbf{T} \mid \mathbf{F}$$

Encoding: from  to  Milner 90 + CPPT'12

From Natural Deduction to Sequent Calculus

Intro  $\Rightarrow$  Right / Elim  $\Rightarrow$  Left + Cut + Identity

Encoding is FUN 

Encoding: from  to  Milner 90 + CPPT'12

From Natural Deduction to Sequent Calculus

Intro  $\Rightarrow$  Right / Elim  $\Rightarrow$  Left + Cut + Identity



$$[x]_a = [x \leftrightarrow a]$$

$$[\langle \rangle]_a = 0$$

$$[\lambda x. M]_a = a(x). [M]_a$$

$$[MN]_a = [M]_x \mid \bar{x}(y). [N]_y \mid [x \leftrightarrow a]$$

# The $\pi$ -calculus as a Descriptive Tool

$\lambda$  in  $\pi$

$$[[x]]_u \stackrel{\text{def}}{=} \bar{x}(u).$$

$$[[\lambda x.M]]_u \stackrel{\text{def}}{=} u(x.u). [[M]]_u.$$

$$[[MN]]_u \stackrel{\text{def}}{=} (\nu f_x) ( [[M]]_f \mid F(x,u) \mid [[x=N]]_u )$$

with  $[[x=N]]_u \stackrel{\text{def}}{=} !x(u). [[N]]_u.$

Milner's  
Encoding  
1991

Encoding: from  $\lambda$  to  $\pi$  Milner 90 + CPPT'12

From Natural Deduction to Sequent Calculus

Intro  $\Rightarrow$  Right / Elim  $\Rightarrow$  Left + Cut + Identity

$\llbracket M \rrbracket_a$   
 $\uparrow$   
 name

$$\llbracket x \rrbracket_a = \llbracket x \leftrightarrow a \rrbracket$$

$$\llbracket \langle \rangle \rrbracket_a = 0$$

$$\llbracket \lambda x. M \rrbracket_a = a(x). \llbracket M \rrbracket_a$$

$$\llbracket MN \rrbracket_a = \llbracket M \rrbracket_x \mid \bar{x}(y). \llbracket N \rrbracket_y \mid \llbracket x \leftrightarrow a \rrbracket$$

$\lambda$  in  $\pi$

$$\llbracket x \rrbracket_u \stackrel{\text{def}}{=} \bar{x}(u).$$

$$\llbracket \lambda x. M \rrbracket_u \stackrel{\text{def}}{=} u(x.u). \llbracket M \rrbracket_u.$$

$$\llbracket MN \rrbracket_u \stackrel{\text{def}}{=} (\nu \bar{z}) (\llbracket M \rrbracket_z \mid \bar{z}(u) \mid \llbracket x \leftrightarrow N \rrbracket)$$

with  $\llbracket x \leftrightarrow N \rrbracket \stackrel{\text{def}}{=} !x(u). \llbracket N \rrbracket_u.$

From  to 

From Sequent Calculus to Natural Deduction

$\llbracket P \rrbracket \Delta \vdash a:A$  with  $\Delta \vdash P :: \underline{a:A}$

$\llbracket 0 \rrbracket = \langle \rangle$

$\llbracket [x \leftrightarrow a] \rrbracket = x$

$\llbracket a(x). P \rrbracket = \lambda x. \llbracket P \rrbracket$

$\llbracket a(x). P \rrbracket = \Lambda X. \llbracket P \rrbracket$

Parallel

$$\left[ \frac{\Delta_1 \vdash P :: a:A \quad \Delta_2, a:A \vdash Q :: b:C}{\Delta_1, \Delta_2 \vdash (\nu a) (P | Q) :: b:C} \right]$$

Substitute P into a in Q

$$= \frac{\Delta_2, a:A \vdash [Q]_b :: C \quad \Delta_1 \vdash [P]_a :: A}{\Delta_1, \Delta_2 \vdash [Q] \{ [P] / a \} :: C}$$

Poly

$$\left[ x \langle B \rangle. P \right] = \left[ P \right] \{ x[B] / x \}$$

Type

Application of B to x

replace x by x[B]

$$\text{cf. } \left[ x \langle b \rangle. (P \mid Q) \right] = \left[ Q \right] \{ (x \langle P \rangle) / x \}$$

Application of P to x

# Theorems

Operational Correspondence / Type Preserving

Inverse

$$(\llbracket M \rrbracket_z) \cong M$$

$$\llbracket (PD) \rrbracket_z \cong P$$

Full Abstraction

$$\llbracket M \rrbracket_z \cong \llbracket N \rrbracket_z \text{ iff } M \cong N$$

$$(PD) \cong (QD) \text{ iff } P \cong Q$$

# Theorems

Operational Correspondence / Type Preserving

Inverse

$$(\llbracket M \rrbracket_z) \cong M$$

$$\llbracket (P) \rrbracket_z \cong P$$

Definability

$$\forall P \exists M \llbracket M \rrbracket_z \cong P$$

$$\forall M \exists P (P) \cong M$$

$$\llbracket M \rrbracket_z \cong P$$

$$(P) \cong M$$

Full Abstraction

$$\llbracket M \rrbracket_z \cong \llbracket N \rrbracket_z \text{ iff } M \cong N$$

$$(P) \cong (Q) \text{ iff } P \cong Q$$

# Theorems

## Operational Correspondence / Type Preserving

### Inverse

$$(\llbracket M \rrbracket_z) \cong M$$



$$\llbracket (P) \rrbracket_z \cong P$$



### Full Abstraction

$$\llbracket M \rrbracket_z \cong \llbracket N \rrbracket_z \text{ iff } M \cong N$$



$$(P) \cong (Q) \text{ iff } P \cong Q$$

Derived

# Theorems

Operational Correspondence / Type Preserving

Inverse

$$(\llbracket M \rrbracket_z) \cong M$$



$$\llbracket (PD) \rrbracket_z \cong P$$



Full Abstraction

$$\llbracket M \rrbracket_z \cong \llbracket N \rrbracket_z$$



$$M \cong N$$

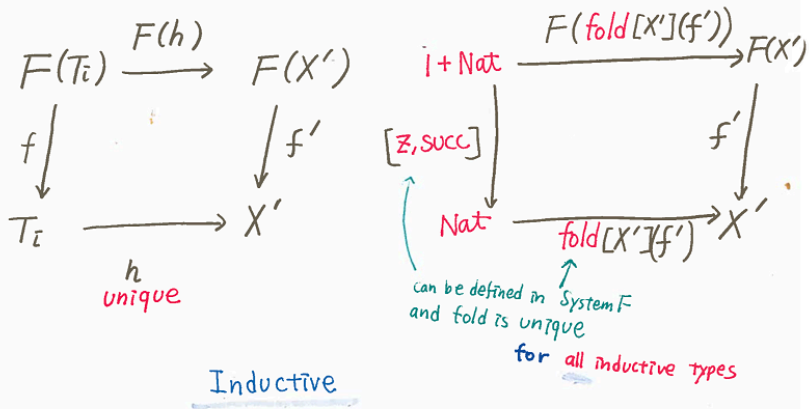
$$(PD) \cong (QD)$$



$$P \cong Q$$

# Application 1 Inductive and Coinductive Types

Parametric Poly is Expressive Enough to Encode Inductive/Co-Inductive Types as Initial / Final (Co)Algebra



## Coinductive Types

$$\begin{array}{ccc} X' & \xrightarrow{h} & T_f \\ f' \downarrow & & \downarrow f \\ F(X') & \xrightarrow{F(h)} & F(T_f) \end{array} \qquad \begin{array}{ccc} X' & \xrightarrow{\text{unfold } [X](f')} & \text{NatStream} \\ f' \downarrow & & \downarrow [\text{hd}, \text{tl}] \\ F(X') & \xrightarrow{F(\text{unfold } [X](f'))} & \text{Nat} \times \text{NatStream} \end{array}$$

Question Sess Poly  $\pi$  is Expressive Enough?

## Theorems

$$\forall Q \text{ s.t. } u: F(X) \rightarrow X, y_1: T_z \vdash Q :: y_2: X. \text{Fold}(X) \cong Q$$

$$\forall_Q \text{ s.t. } u: A \rightarrow F(A), y_1: A \vdash Q :: y_2: T, Q \cong \text{Unfold}(A)$$

## Application (2) $HO\pi + PROcess\ Monad$

- Full Abstraction / Inverse  $x \langle M \rangle, P$
- Strong Normalisation via Strong Normalisation

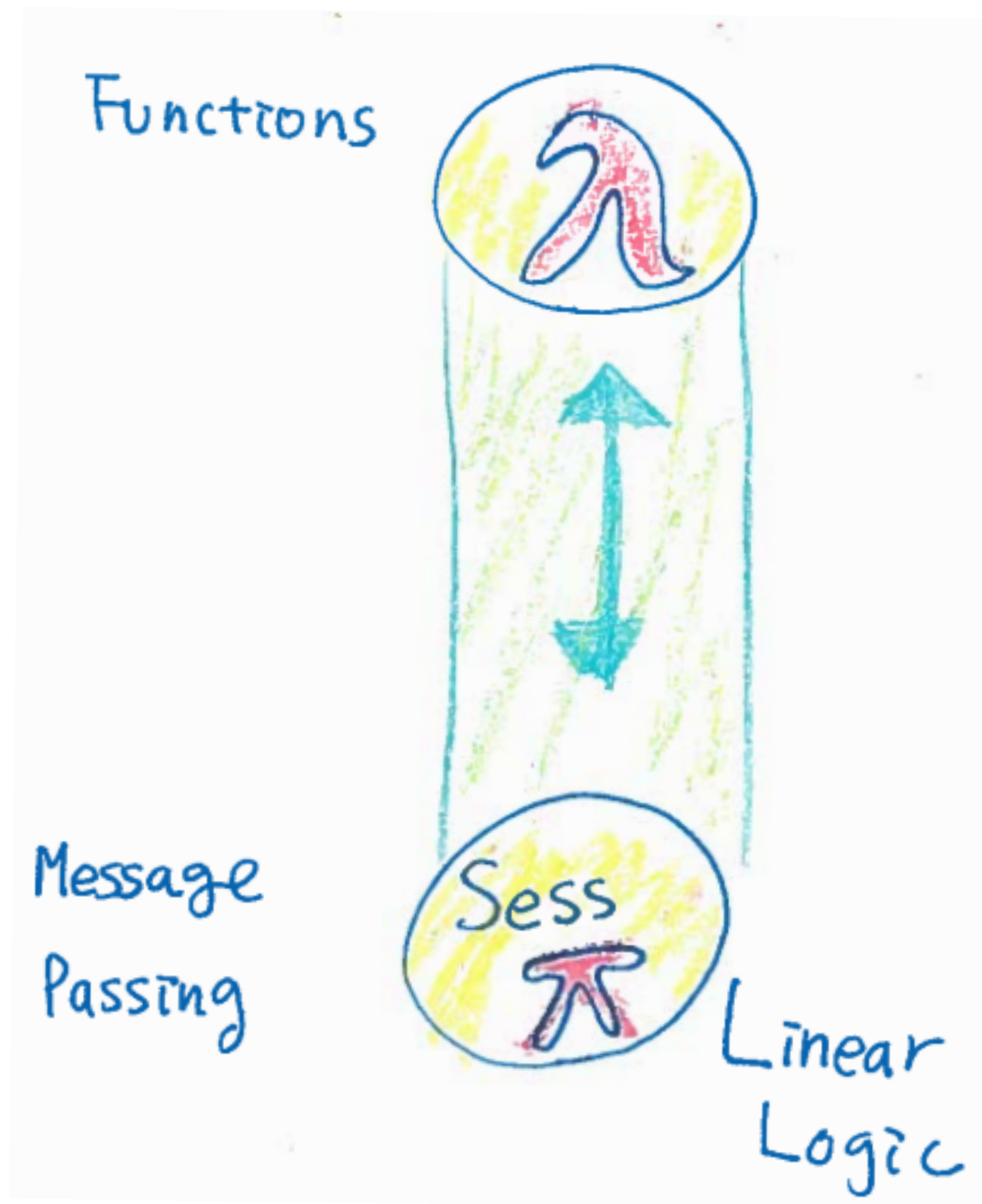
$HO\pi$

$\lambda$

$$P \rightarrow Q \quad \Rightarrow \quad (P) \dashv\vdash (Q)$$

cf. [CPPT'13] logical relation

# Summary



# Summary

SAD?



Functions



Message  
Passing



Linear  
Logic

# Summary

Functions



Linear  
Logic

Message  
Passing

SAD?



NO!



# Summary

SAD?



Functions



Linear  
Logic

- Use Sess  $\pi$  to articulate LL-based boarder computations

NO!



Message  
Passing

# Summary

Functions



Message  
Passing



Linear  
Logic

SAD?



- Use <sup>LL-based</sup> Sess  $\pi$  to articulate boarder computations

NO!



## LL-Based Session Types

- Balzer and Pfenning, Manifest sharing with session types, ICFP'17
- Rocha and Caires, Propositions-as-types and shared state, ICFP'21
- Zien et al, Client-server sessions in linear logic, ICFP'21
- Frumin et al, A propositions-as-sessions interpretation of bunched implications in channel-based concurrency, OOPSLA'22

# Summary

Functions



Message  
Passing



Linear  
Logic

SAD?



- Use <sup>LL-based</sup> Sess  $\pi$  to articulate boarder computations

NO!



## LL-Based Session Types

- Balzer and Pfenning, Manifest sharing with session types, ICFP'17
- Rocha and Caires, Propositions-as-types and shared state, ICFP'21
- Zien et al, Client-server sessions in linear logic, ICFP'21
- Frumin et al, A propositions-as-sessions interpretation of bunched implications in channel-based concurrency, OOPSLA'22

## Game Semantics

- Rideau and Winskel, Concurrent strategies, LICS'11
- Ghica and Tzevelekos, System level game semantics, ETCS'12
- Vale et al, Layered and object-based game semantics, POPL'22