Balzer 1.1

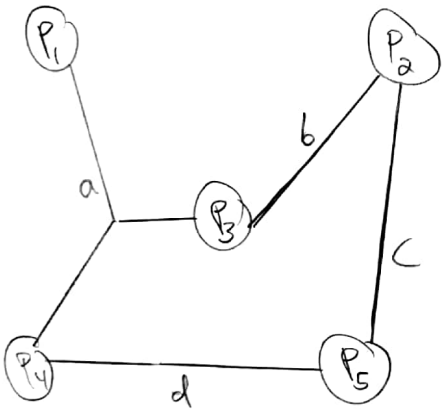
Session typed Concurrent Programming (STCP)

RoadMap:
1. message-passing concurrent programming
2. session types
3. Linear-logic-Session types
4. Manifest sharing

learning objectives:
1. how can we program using message-passing concurre
2. what session types are about
3. Benefits of Linear logic to programming
4. How to accomodate sharing in a logically motivated way

message-passing concurrency

processes that compute by exchanging messages along channels.



a network ↦ processes P_i connected by channels $\{a,b,c,d\}$.

channels are n-ary.

$ar(a) = 3$

$ar(b) = ar(c) = ar(d) = 2$

i.e., P_1 can send a message to either P_3 or P_4 on channel a.

thus, non-determinism.

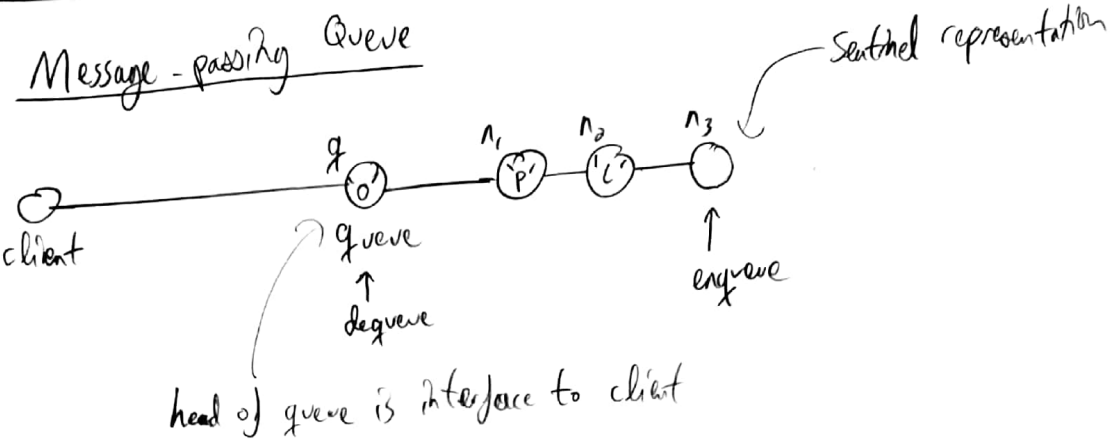
π-calculus (Milner, 1992)

↳ "universal"

Functional Programming is to λ-calculi as STCP are to process-calculi.

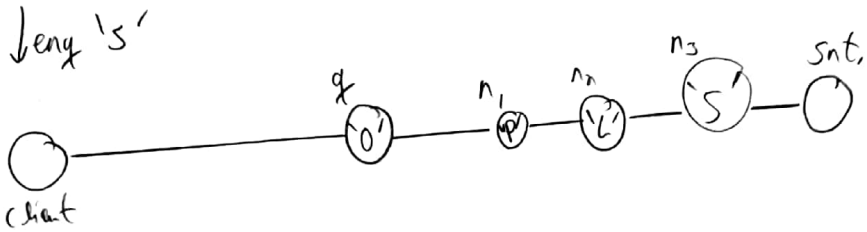
Balzer 1.2

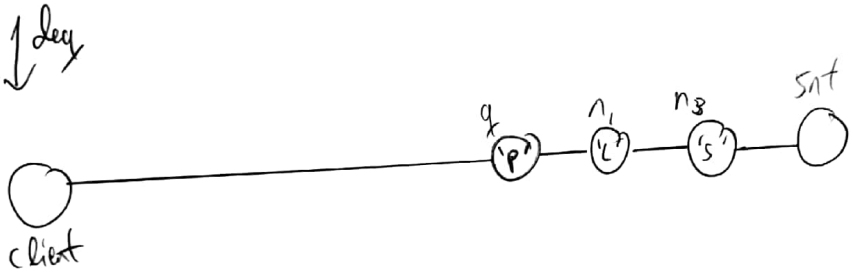
Message-passing Queue

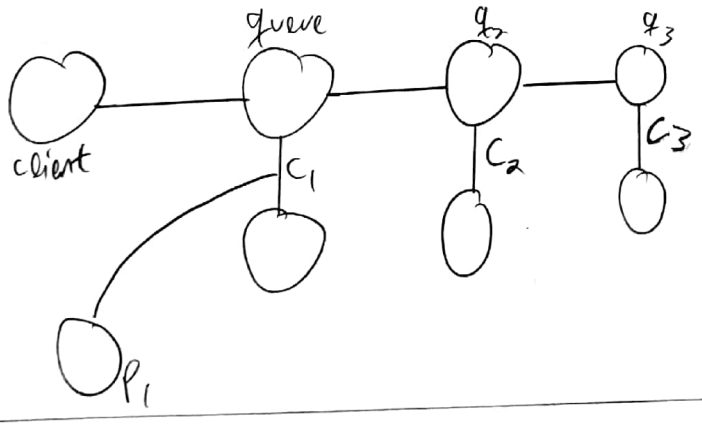


this is the process.

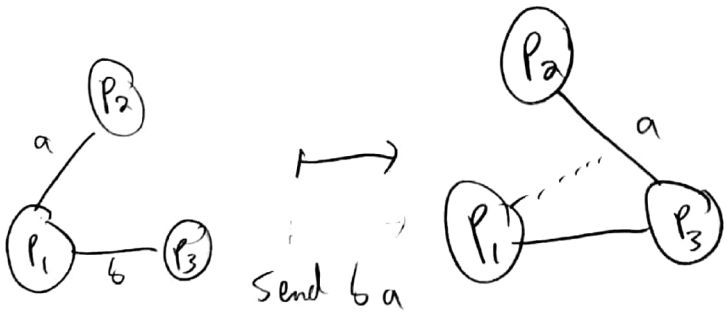
head of queue is interface to client







Channels passing other channels



"Send along b a"

"mobility"

"higher order channels"

Balzer 1,3

How do you type message-passing protocols

- types for protocols of message-exchange

session types (Kohei Honda 1993)

$$A ::= ?[T].A' \mid ![T].A' \mid \&\{l_1:A_1, \dots, l_n:A_n\} \mid \oplus\{l_1:A_1, \dots, l_n:A_n\} \mid \text{end} \mid X \mid \mu X.A'$$

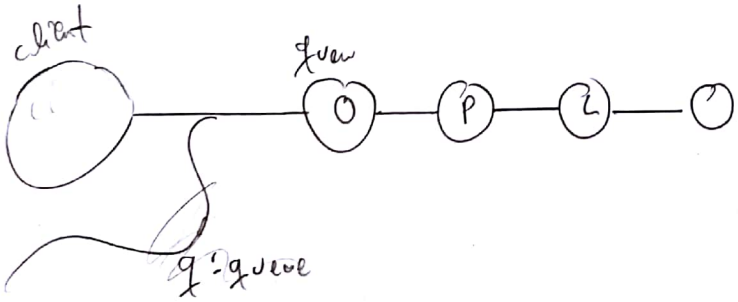
("with" over &; "plus" over ⊕; "type var" over X)

- $?[T].A'$: "I am willing to receive an input of type T, and after I will continue to be type A'"
- $![T].A'$: "I am willing to send ..."
- $\&\{l_1:A_1, \dots, l_n:A_n\}$: "choose between different types by label" / give choice to client / external choice
- $\oplus\{l_1:A_1, \dots, l_n:A_n\}$: give choice to process / internal choice.

$$T ::= A \mid \text{int} \mid \text{char} \mid \dots$$

a queue is an equi-recursive type

$$\text{queue} = \&\{\text{enq}: ?[\text{char}].\text{queue},\ \text{deq}: \oplus\{\text{none}: \text{end},\ \text{some}: ![\text{char}].\text{queue}\}\}$$



q : &{enq: ..., deq: ...} → "send 'enq' along q"

q : ?[char].queue → "send 's' along q"

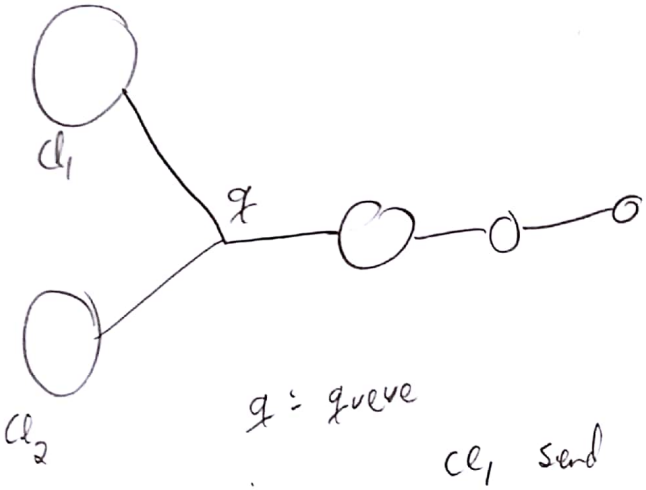
q : queue

IC stateful, types change over the

type of channel/process changes w message exchange.

Balzer 1.4

Consider two clients



q : queue

cl_1 send eng along q

q : ?[char], queue

cl_2 send deg along q

preservation in session types, called session fidelity guarantees that expectation of client matches with the one of provider if they match initially

R channels are resources (from linear logic)

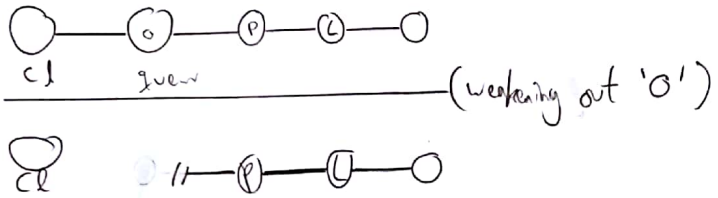
linear logic rejects weakening and contraction

Balzer 2.1

linear logic is substructural, omits weakening and contraction

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}\ (\text{weakening}) \qquad \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}\ (\text{contraction})$$

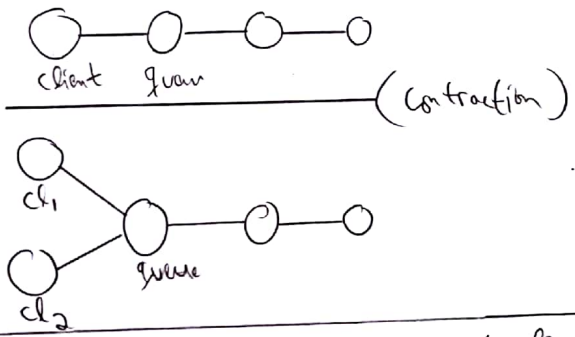
"dropping a resource"



if weakening is allowed, we could have the queue .PC.. cut off from its head, and by extension its client

this is a problem

R ≡ no dining philosophers in linear logic system



parent: client
child: provider (queue)

~~propositions as types~~ correspondence between intuitionistic LL and session typed π-calculus

---

there is curry-howard correspondence between intuitionistic linear logic and session typed π-calculus

logic		programming
linear Props		session types
Proofs	⟨processes⟩	programs
cut reduction		communication

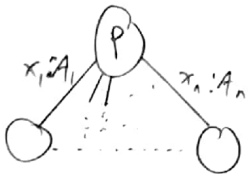
$A, B, C ::= A \multimap B$ (multiplicative implication)
(no "⅋ par", because intuitionistic)
$\mid A \otimes B$ (multiplicative conjunction)
$\mid A \,\&\, B$ (additive conjunction)
$\mid A \oplus B$ (additive disjunction)
$\mid\ !A$ (of course, persistent truth)

require: $\&, \oplus$ have to have at least one label, no unit or neutral val

Balzer 2.2

$\overbrace{x_1:A_1, \dots, x_n:A_n}^{\Delta} \vdash P :: (x:A)$ is read

"process P offers a session of type A along channel x, using sessions of types $A_1, \dots, A_n$ offered along channels $x_0, \dots, x_n$"



R typing judgment is always expressed by the provider

N Δ is linear context

External Choice

$$\frac{\Delta \vdash P_1 :: (x:A) \qquad \Delta \vdash P_2 :: (x:B)}{\Delta \vdash \text{case } x \text{ of } (P_1, P_2) :: (x : A \& B)} (\&R)$$

$$\frac{\Delta, x:A \vdash Q :: z:C}{\Delta, x:A \& B \vdash x.\text{inl}; Q} (\&L_1) \qquad \frac{\Delta x:B \vdash Q :: z:C}{\Delta, x:A \& B \vdash x.\text{inr}; Q :: z:C} (\&L_2)$$

Internal Choice

$$\frac{\Delta \vdash Q :: (x:A)}{\Delta \vdash x.\text{inl}; Q :: (x:A \oplus B)} (\oplus R_1) \qquad \frac{\Delta \vdash \quad :: (x:A)}{\Delta \vdash x.\text{inr}; Q :: (x:A \oplus B)} (\oplus R_2)$$

$$\frac{\Delta, x:A \vdash P_1 :: z:() \qquad \Delta, x:B \vdash P_2 :: z:()}{\Delta, x:A \oplus B \vdash \text{case } x \text{ of } (P_1, P_2) :: z:()} (\oplus L)$$

$$\frac{\Delta, y:A \vdash P :: x:B}{\Delta \vdash y \leftarrow \text{recv} x; P :: A \multimap B} (\multimap R) \text{ (channel input)}$$

$$\frac{\Delta \vdash Q :: y:A \qquad \Delta' \vdash Q' :: (x:B)}{\Delta, \Delta' \vdash \text{send} x (y \leftarrow Q); Q' :: (x : A \otimes B)} (\otimes R)$$

$$\frac{\Delta, x:B, y:A \vdash P :: z:()}{\Delta, x:A \otimes B \vdash y \leftarrow \text{recv} x; P :: z:C} (\otimes L)$$

Channel output

$$\frac{\Delta \vdash Q :: (y:A) \qquad \Delta, x:B \vdash \quad :: (:C)}{\Delta, \Delta', x:A \multimap B \vdash \text{send} x (y \leftarrow Q); Q' :: z:C} (\multimap L)$$

$$\frac{}{\cdot \vdash \text{close} x :: (x:1)} (1R) \text{ ("with no resources, close the channel")}$$

$$\frac{\Delta \vdash P :: z:C}{\Delta, x:1 \vdash \text{wait} x; P :: z:C}$$

Termination

Balzer 3.1

curry-howard table		polarity
linear props	session types	
$A \& B$	external choice	−
$A \oplus B$	internal choice	+
$A \multimap B$	channel input	−
$A \otimes B$	channel output	+
1	termination	+
cut	parallel composition spawning a process	
!	output/send	
?	input/receive	

cut is the main facilitator of computation

linear props C_0

$C : \hat{\&}_{k:\emptyset}\{l_k : A_k\}$ $\quad \$c : ?\text{choice}\{\langle A \rangle l\}$

$c : A \otimes B$ $\quad \$c : \langle !A ; B \rangle$

$c : 1$ $\quad \$c : \langle \rangle$

$$\frac{\Delta \vdash P :: x:A \qquad \Delta', A \vdash Q :: z:C}{\Delta, \Delta' \vdash x \leftarrow P, Q :: z:C}\ \text{cut}$$

$$\frac{}{y:A \vdash \text{fwd}\ x\ y :: x:A}\ \text{Identity}$$

multiset rewriting rules

$S \longrightarrow S'$

State := $\text{proc}(c_1, P_1), \ldots, \text{proc}(c_n, P_n)$

(proc: process; channel currently executing process)

① $\text{proc}(c, (\text{wait } a; Q)), \text{proc}(a, \text{close } a) \mapsto \text{proc}(c, Q)$

② $\text{proc}(c, a.l_k; Q), \text{proc}(a, \text{case } a \text{ of } \overrightarrow{l \Rightarrow P}) \mapsto \text{proc}(c, Q), \text{proc}(a, P_k)$

where l, P are seqs of labels and k is an index

(cut) $\text{proc}(a, x \leftarrow P, Q) \mapsto \text{proc}(a, [b/x]Q), \text{proc}(b, P)$

where b is fresh

(fwd) $\text{proc}(a, \text{fwd } a\ b) \mapsto a = b$

Balzer 3.2

actual concurrent C0 code

```
#use <conio>
typedef <?choice queue> queue;
typedef <!choice queue_elem> queue_elem;
choice queue {
    <?int; queue> enq;
    <queue_elem> deq;
}
choice queue_elem {
    <> none
    <!int; queue> some;
}
```

Session types

$ to denote channel

proces →

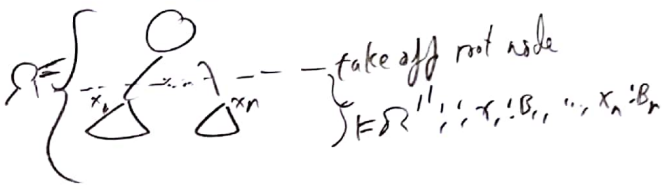
```
queue $q empty () {
    switch ($q) {
        case enq:
            int y = recv($q);
            queue $e = empty();
            $q = elem(y, $e);
        case deq:
            $q.none;
            close($q);
    }
}
```

sends label none along offerb channel.

$\Omega ::= \cdot \mid proc(a_i, P_i), \Omega'$

Preservation: the judgment "Ω is well-typed" — if $\vDash \Omega :: \Delta$ and $\Omega \longrightarrow \Omega'$ then $\vDash \Omega' :: \Delta$

(Ω is a tree) $\vDash \Omega :: 1$



Progress

Balzer 4.1

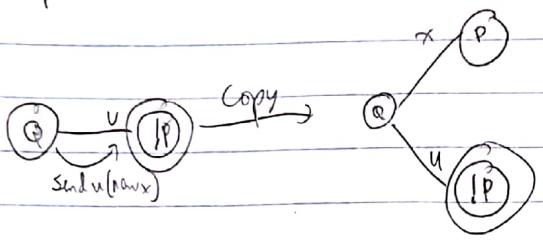
$A,B,C ::= A \otimes B \mid A \multimap B \mid \&\{l_k : A_k\}_{k \in I} \mid \oplus\{l_k : A_k\}_{k \in I} \mid !A$

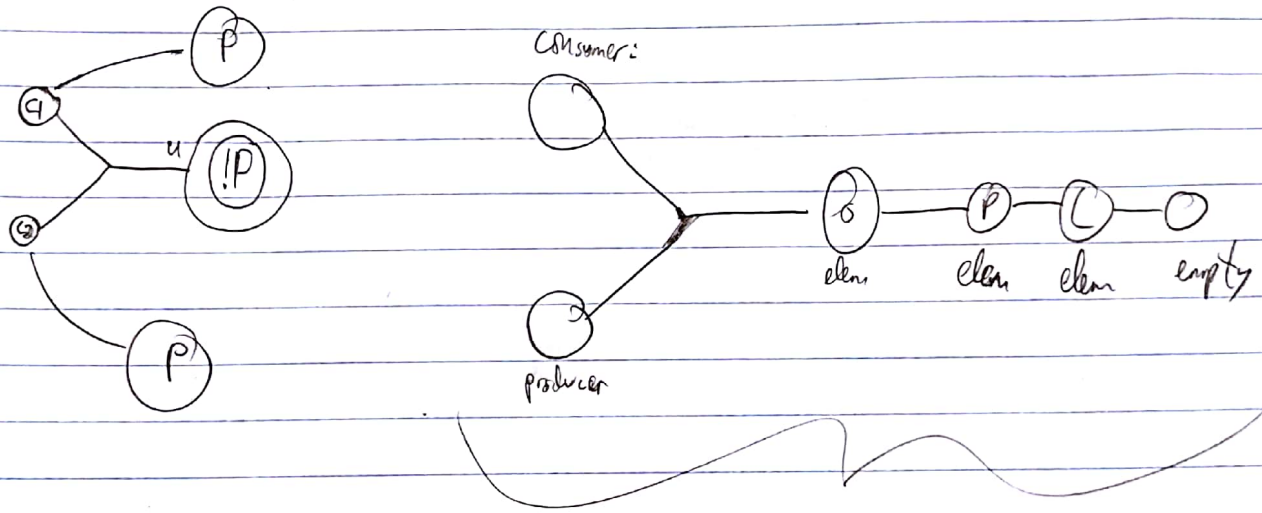
$\Psi; \Delta \vdash P :: x : A$

Ψ: structural, Δ: linear

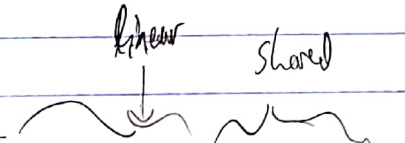
~~cut~~ rule; cut! copy !R !L

process replication: $!P \triangleq P \mid !P$





this picture isn't good for type preservation



queue $A = \&\{$ enq : $A \multimap$ queue A,
deq : $\oplus\{$ none : queue A,
some : $A \otimes$ queue $A\}\}$

shared: $A ::= \uparrow^S_L A_L \mid$

linear: $A_L, B_L ::= \oplus\{l_k : A_k\}_L \mid A_L \otimes B_L \mid 1 \mid \&\{l : A\}_L \mid A_L \multimap B_L \mid \downarrow^S_L A_S \mid \exists x : A_S . B_L \mid \forall x : A_S$

shared layer is "higher" than linear layer.

Balzer 4.2

$$\text{queue}\ A_s = \uparrow_L^S \& \{\text{enq} : \Pi x{:}A_s.\ \downarrow_L \text{queue}\ A_s,\ \text{deq} : \oplus\{\text{none} : \downarrow_L^S \text{queue}\ A_s,\ \text{some} : \exists x{:}A_s.\ \downarrow_L^S \text{queue}\ A_s\}\}$$

linear proc $\Gamma ; \Delta \vdash P :: (x_L : A_L)$

shared proc $\Gamma \vdash P :: (x_S : A_S)$

Δ is a linear context, Γ is a structural context (w weakening, c contraction)

$$\frac{\Gamma, x_S : \uparrow_L^S A_L ; \Delta, x_L : A_L \vdash Q_{x_L} :: (z_L : C_L)}{\Gamma, x : \uparrow_L^S A_L ; \Delta \vdash x_L \leftarrow \text{acquire}\ x_P ; Q_{x_L} :: (z_L : C_L)}\ (\uparrow_L^S \cdot L)$$

$$\frac{\Gamma ; \cdot \vdash P_{x_L} :: (x_L : A_L)}{\Gamma \vdash x_L \leftarrow \text{accept}\ x_S ; P_x :: (x_S : \uparrow_L^S A_L)}\ (\uparrow_L^S \cdot R)$$

$$\frac{\Gamma, x_S : A_S ; \Delta \vdash Q_{x_S} :: (z_L : C_L)}{\Gamma ; \Delta, x_L : \downarrow_L^S A_S \vdash x_S \leftarrow \text{release}\ x_L ; Q_{x_S} :: (z_L : C_L)}\ (\downarrow_L^S \cdot L)$$

$$\frac{\Gamma \vdash P_{x_S} :: (x_S : A_S)}{\Gamma ; \cdot \vdash x_S \leftarrow \text{detach}\ x_L ; P_{x_S} :: (x_L : \downarrow_L^S A_S)}\ (\downarrow_L^S \cdot R)$$

D well-formed session types : equi synchronizing. (Preservation)

(exactly an asynchronous untyped π-calculus for shared session type process calculi.

curry-howard

linear props

proofs

session types

processes

interleaving
- proof construction
- proof reduction
- proof deconstruction

acquire
communication
release

deadlock : proof construction