

# Session TYPES

AS A

Descriptive

$\pi$   $\pi$

Nobuko  
Yoshida  
&  
Raymond Hu

Tool

25th  
Oct  
2017

for Distributed PROTOCOL

# The Kohei Honda Prize for Distributed Systems Queen Mary, University of London

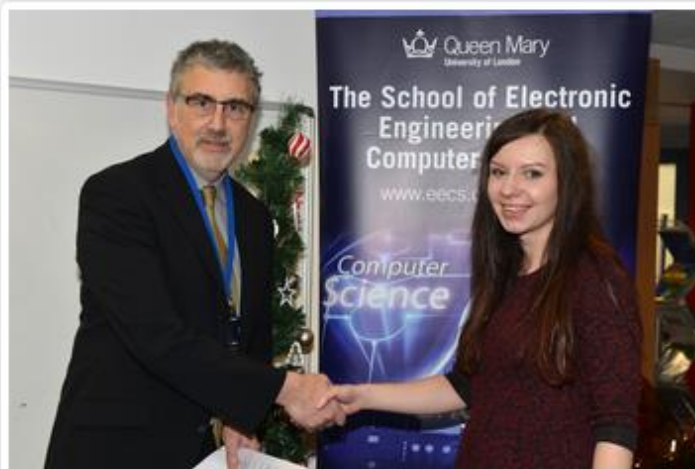
Posted with permission from QMUL on 17<sup>th</sup> Dec 2013. [Original article](#) written by Edmund Robinson.

This prize was instituted in 2013 and is awarded annually to one undergraduate student and one postgraduate student in recognition of their achievement in applying the highest quality scientific and engineering principles in the broad area of Distributed Systems. This is the area in which Dr Honda concentrated most of his teaching, and it is also the area in which he conducted his research. Its primary funding comes from a donation from his family, who wished to commemorate Dr Honda in this way. Additional funding has come from Dr Honda's own ETAPS Award. This prize is sponsored by Springer Verlag, and awarded annually by the ETAPS committee in recognition of an individual's research contribution. Dr Honda received the first such award posthumously, and the awarding panel expressed a wish that the funding be used to supplement this prize fund. The laudation for this award, written by Dr Honda's colleague, Prof Vladimiro Sassone is included later.

## About Dr Honda

Kohei Honda was born and lived the first part of his life in Japan. Like many scientists he was fascinated by the idea of finding basic explanatory theories, like the physicists looking for grand unified theories of the universe. Kohei, though, was passionately interested in finding the right basic explanatory theory for the process of computation. Most academics agree that the basic theory

## Winners 2013



**Ms Anna Pawlicka**

2013 winner (Undergraduate) source: QMUL



**Mr. Valdmir Negacevshi**

2013 winner (Postgraduate) source: QMUL



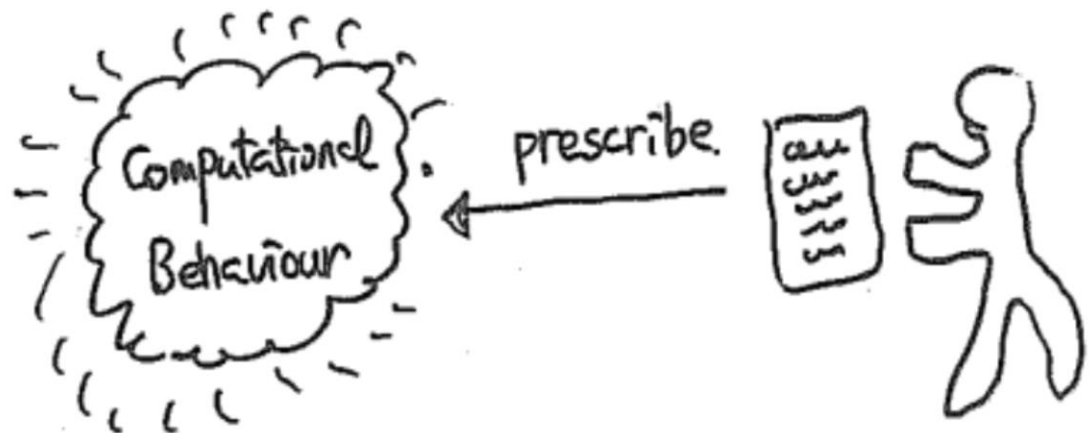
# Idioms for Interaction

— 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 _stk1, 48 } stacks.
data _stkr, 48 }
data _i, 4 } index
data _j, 4 } left/right link
data _r, 4 } pivot
data _x, 4 } temporary value.
data _s, 4 } stack pointer
data _a, 48 } table to be sorted.

mov $0, _s
mov $0, _stk1 } initialisation
mov $11, _stkr

L1: mov _s, rax
    mov _stk1(, 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 } i := l;
    mov rcx, _i
    mov _r, rcx } j := r;
    mov rcx, _j
    mov _l, rcx
    add _r, rcx } rdx := l+r
    mov rdx, rax
    mov rax, rdx
    shr $31, rdx
    add rdx, rax
    mov rax, rdx
    sar $1, rdx
    mov _a(, rdx, 4), rcx } x := a(rdx);
    mov rcx, _x

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

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 nil then nil.

$((\lambda xy. y(xy))(\lambda xy. y(xy)))(\lambda c. \lambda xy. x((\lambda x. x(\lambda xy. x))x)y$   
 $((\lambda xy. \lambda z. z(\lambda xy. y)xy)((\lambda x. x(\lambda xyz. y))x)(c((\lambda x. x(\lambda xyz. z))x)y)$  } concat.

$(q(\lambda xy. y(xy))(\lambda xy. y(xy)))(\lambda f. \lambda px. ((\lambda x. x(\lambda xy. x))x)(\lambda x. x))$   
 $(p((\lambda x. x(\lambda xyz. y))x)((\lambda xy. \lambda z. z(\lambda xy. y)xy)x$  } sort and filter

$(f((\lambda x. x(\lambda xyz. z))x)))(f((\lambda x. x(\lambda xyz. z))x))$

$(\lambda y. (((\lambda xy. y(xy))(\lambda xy. y(xy))) \lambda f'. \lambda xy. ((\lambda x. x \lambda xy. x)y$  }  $\lambda y. LT_y. Car$

$(\lambda xy. y)((\lambda x. x \lambda xy. x)x)(\lambda xy. y)(f'((\lambda x. x \lambda xy. y)x)((\lambda x. x \lambda xy. y$

$y((\lambda x. x(\lambda xyz. y))l)((\lambda x. x(\lambda xyz. z))l))$  }  $Cdr\ l$

$((\lambda xy. \lambda z. z(\lambda xy. y)xy)((\lambda x. x(\lambda xyz. y))l))$  }  $Cons\ (Car\ l)$

$(q((\lambda xy. y(xy))(\lambda xy. y(xy)))(\lambda f. \lambda px. ((\lambda x. x(\lambda xy. x))x)$  } sort and filter

$(\lambda x. x)(p((\lambda x. x(\lambda xyz. y))x)((\lambda xy. \lambda z. z(\lambda xy. y)xy)x$

$(f((\lambda x. x(\lambda xyz. z))x)))(f((\lambda x. x(\lambda xyz. z))x))$

$(\lambda y. ((\lambda xy. y(xy))(\lambda xy. y(xy))) \lambda f''. \lambda xy. ((\lambda x. x \lambda xy. x)x$  }  $\lambda y. ME_y$

$((\lambda x. x \lambda xy. x)y)(\lambda xy. x)(\lambda xy. y)$

$((\lambda x. x \lambda xy. x)y)(\lambda xy. x)(f''((\lambda x. x \lambda xy. y)x$

$((\lambda x. x \lambda xy. y)y))y((\lambda x. x(\lambda xyz. y))l)((\lambda x. x(\lambda xyz. z))l))$  }  $Cdr\ l$

## Quicksort with combinators:

$Y(\lambda f. \lambda l.$

$(Isnil\ l)$

$(Concat\ (f\ Filter\ (\lambda y. LT_y\ (Car\ l))$   
 $(Cdr\ l)))$

$(Cons\ (Car\ l)$

$(f\ Filter\ (\lambda y. ME_y$   
 $(Car\ l)$   
 $(Cdr\ l))))$

$I = \lambda x. x$     $T = \lambda xy. x$     $F = \lambda xy. y$     $Y = (\lambda xy. y(xy))(\lambda xy. y(xy))$

$Cons = \lambda xy. \lambda z. z F xy$     $Isnil = \lambda x. x T$

$Car = \lambda x. x (\lambda xyz. y)$     $Cdr = \lambda x. x (\lambda xyz. z)$

$Concat = Y(\lambda c. \lambda xy. x (Isnil\ x) y (Cons\ (Car\ x) (c\ (Cdr\ x) y)$

$Filter = Y(\lambda f. \lambda px. (Isnil\ x) I (p\ (Car\ x)) (Cons\ x (Filter\ (Cdr\ x) y))$

$Iszero = \lambda x. x T$     $Pred = \lambda x. x F$     $(Filter\ (Cdr\ x))$

$LT = Y(\lambda f. \lambda xy. (Iszero\ y) F ((Iszero\ x) F (f\ (Pred\ x) (Pred\ y)$

$ME = Y(\lambda f. \lambda xy. (Iszero\ x) ((Iszero\ y) I F) ((Iszero\ y) (T) y))$   
 $(f\ (Pred\ x\ y\ (Pred\ x\ y)))$

## 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  $\pi$

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

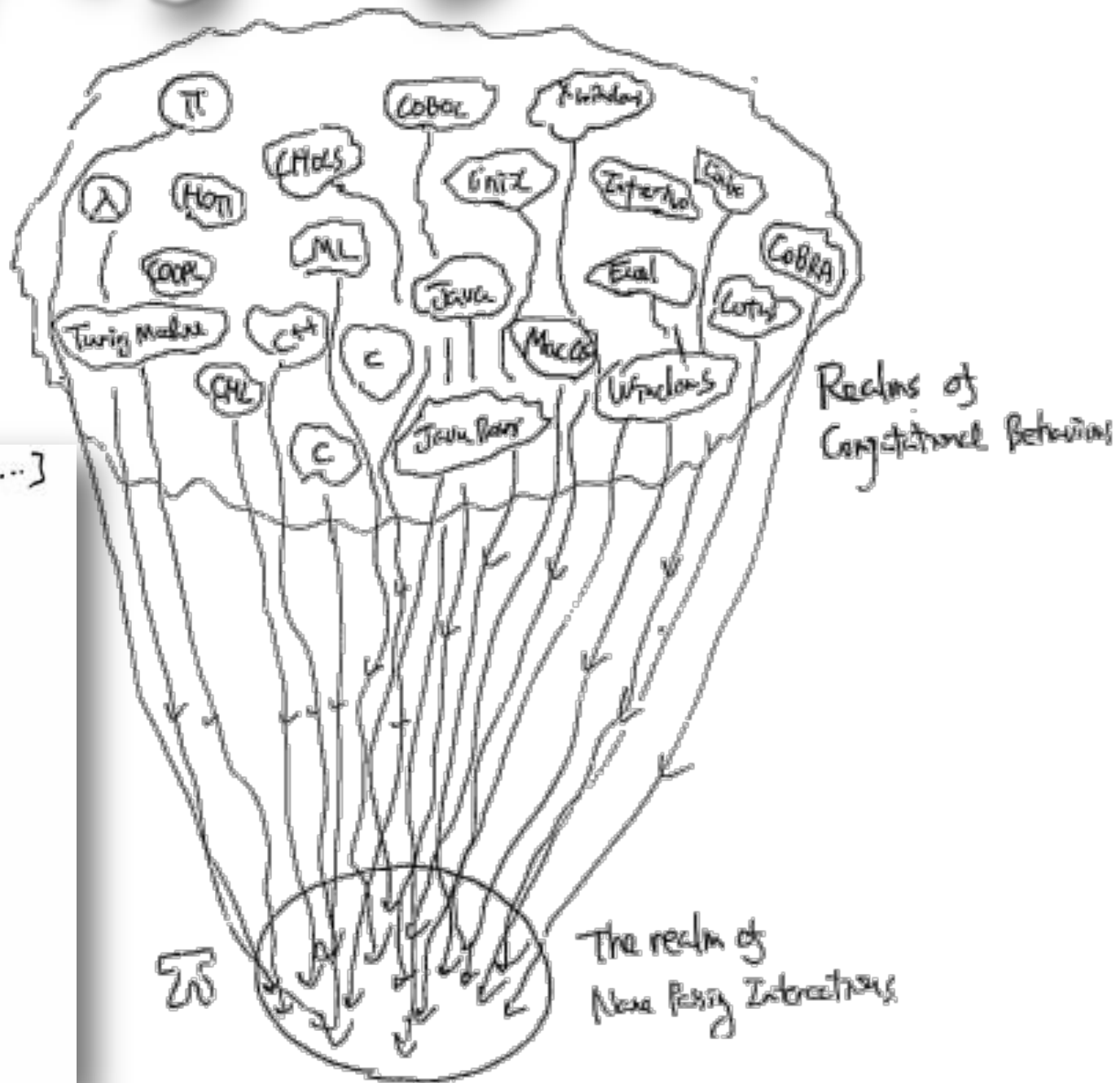
$$[\lambda x.M]_u \stackrel{\text{def}}{=} u(x).[M]_u.$$

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

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



## \* Examples of Representable Computation.



- $\lambda$ -calculus [MPW89, Milner 90, Milner 92, ...]
- Concurrent Object [Walker 91]
- $\omega$ -order term passing [Sangiorgi 92]
- Various data structures [Milner 92, ...]
- Proof Nets [Bellare and Scott 93]
- Abstract 'constant' interaction [HRS4]
- Strategies on Games [HORS5]

# 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



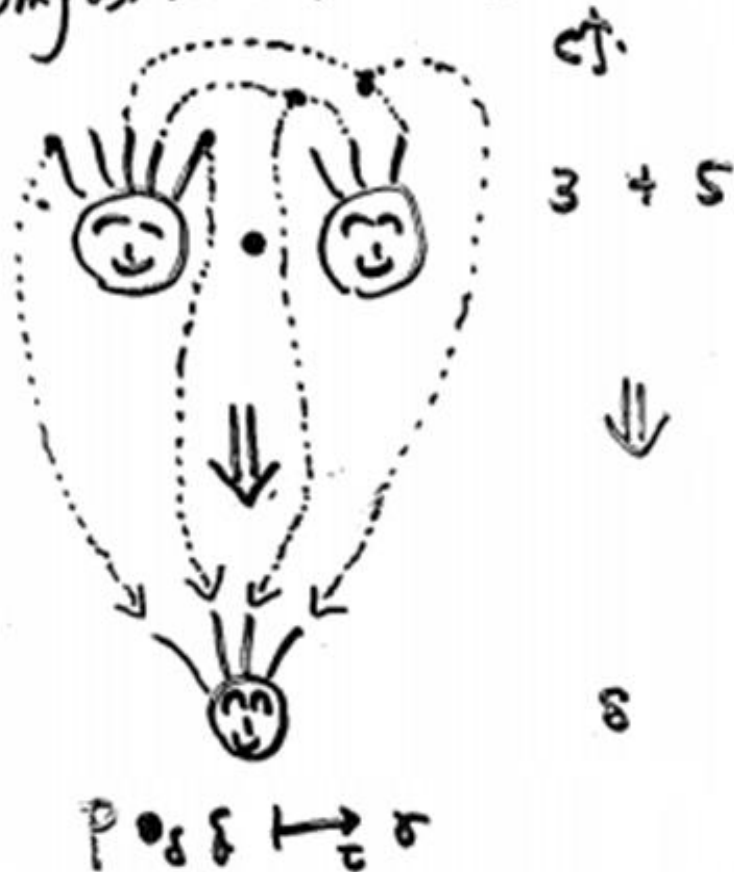
with operation!

$$\begin{cases} f : \alpha \Rightarrow \beta \bullet e : \alpha = f \bullet e : \beta. \\ \text{else undefined.} \end{cases}$$

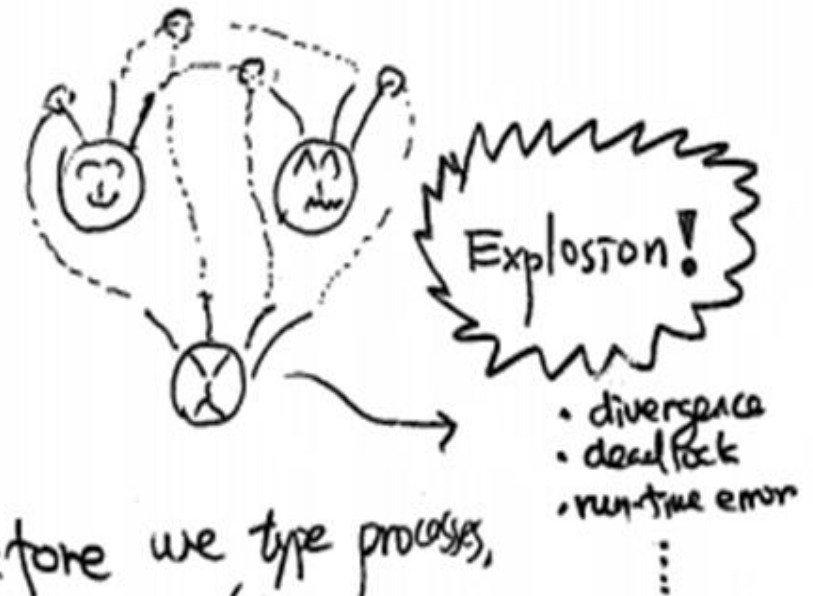
function application.

# Process Types

- When it comes to processes, composition becomes:



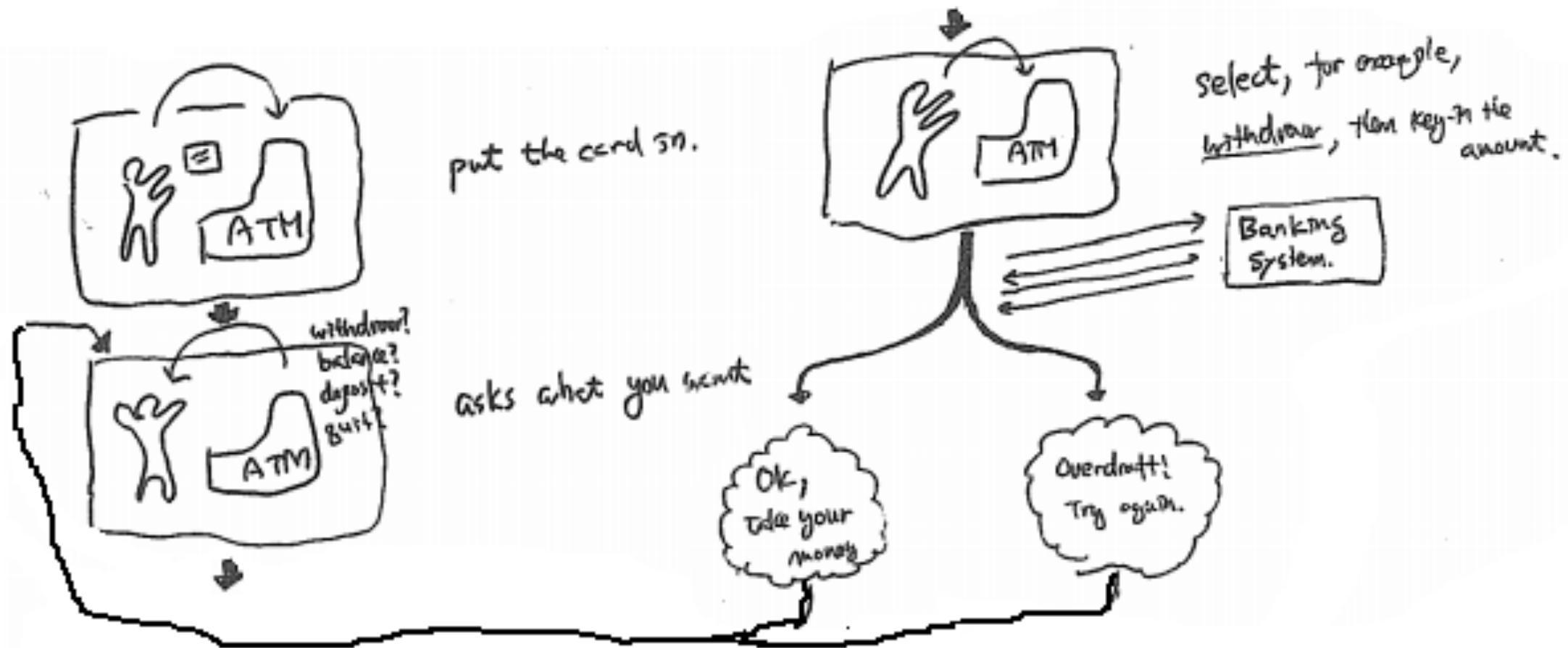
- But some composition is dangerous!



- Therefore we type processes,



# Implementing ATM



## Implementing ATM

$$ATM_{cb}^{\bar{c}} =$$
$$\mathcal{L}(Z) := \mathbb{R}^D; [\underline{w}_d; \underline{x}_i]$$
$$\mathcal{D}(\omega) ::= \omega! \underline{\omega!}; \mathcal{D}; !X)$$

[?ok:

$$\mathbb{Z}[\frac{1}{2}]; \mathbb{Z}; \Theta^M(\omega),$$

? overdraft:

$$2! \text{ over } ATM(cb)]$$

? but;

$$\bar{b}\langle w \rangle ::= w \uparrow \underline{bcl}; ?X;$$

2. X; ATM (20)

 $\frac{p}{k} \leftrightarrow \text{user.}$ 

\*  $\Leftrightarrow$  bank.

22

$$\frac{1}{p} \Leftrightarrow \text{use } r$$

$y \in \text{bank}$

if  $\in \rightarrow$  user.

$\neq \Rightarrow$  user

$$y \in b \cap k$$
$$L_1 L_2 \rightarrow (! f a' e' f', f a. (\bar{e} f. f' b.$$
$$a_2. c_0(z_1, c_2. c_1(z_2, (z_1', c_1. e_1(\bar{a}_1, e_1. (f_1, e_1. f_2, e_1. (f_1, e_1. f_2))))))$$
$$!f''_{2g} \cdot (f''_f, f''_a \cdot (f''_f, f''_b \cdot (f''_f, f''_c \cdot (f''_f, f''_d.$$
$$G_2 G_1 (\tilde{G}_1, C_1 - (\tilde{G}_1, C_2 \cdot \tilde{G}_2, C_3 \cdot \tilde{G}_3, C_4$$

५. (५.३५.००) (४०, ०२.

$$E_{W^p}(\mathbb{T}e_1, e_1 y, (\mathbb{T}w, e. wy), e_2 y, e_2 y_1, \dots$$
$$e_3 + (5e_1, e_3 y_2 - 4e_1 + 5e_4, e_4 - 5e_3) (5e_1,$$

$e_0 \cdot (\bar{v}_1, e_0 \cdot (\bar{v}_2, e_0 \cdot (\bar{w}_2, e_0 \cdot$

(bc) e.g. (bc), e.g. ef, (ef), ef'.

$\varphi_1(\varphi_2, \dots, \varphi_n) \in \mathcal{F}_1(\mathcal{F}_2, \mathcal{F}_3, \dots, \mathcal{F}_n)$

$$\gamma_0, e_0, e_1 \vdash (\exists e_n, e_{n+1}). (\frac{1}{2}x, e_n, e_{n+1}) \vdash (\bar{q}e'_1, e'_2)$$

$2a_1 + 5a_2 = 0$

Суд. деп. (Б. 4, 13. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53. 54. 55. 56. 57. 58. 59. 60. 61. 62. 63. 64. 65. 66. 67. 68. 69. 70. 71. 72. 73. 74. 75. 76. 77. 78. 79. 80. 81. 82. 83. 84. 85. 86. 87. 88. 89. 90. 91. 92. 93. 94. 95. 96. 97. 98. 99. 100. 101. 102. 103. 104. 105. 106. 107. 108. 109. 110. 111. 112. 113. 114. 115. 116. 117. 118. 119. 120. 121. 122. 123. 124. 125. 126. 127. 128. 129. 130. 131. 132. 133. 134. 135. 136. 137. 138. 139. 140. 141. 142. 143. 144. 145. 146. 147. 148. 149. 150. 151. 152. 153. 154. 155. 156. 157. 158. 159. 160. 161. 162. 163. 164. 165. 166. 167. 168. 169. 170. 171. 172. 173. 174. 175. 176. 177. 178. 179. 180. 181. 182. 183. 184. 185. 186. 187. 188. 189. 190. 191. 192. 193. 194. 195. 196. 197. 198. 199. 200. 201. 202. 203. 204. 205. 206. 207. 208. 209. 210. 211. 212. 213. 214. 215. 216. 217. 218. 219. 220. 221. 222. 223. 224. 225. 226. 227. 228. 229. 230. 231. 232. 233. 234. 235. 236. 237. 238. 239. 240. 241. 242. 243. 244. 245. 246. 247. 248. 249. 250. 251. 252. 253. 254. 255. 256. 257. 258. 259. 260. 261. 262. 263. 264. 265. 266. 267. 268. 269. 270. 271. 272. 273. 274. 275. 276. 277. 278. 279. 280. 281. 282. 283. 284. 285. 286. 287. 288. 289. 290. 291. 292. 293. 294. 295. 296. 297. 298. 299. 300. 301. 302. 303. 304. 305. 306. 307. 308. 309. 310. 311. 312. 313. 314. 315. 316. 317. 318. 319. 320. 321. 322. 323. 324. 325. 326. 327. 328. 329. 330. 331. 332. 333. 334. 335. 336. 337. 338. 339. 340. 341. 342. 343. 344. 345. 346. 347. 348. 349. 350. 351. 352. 353. 354. 355. 356. 357. 358. 359. 360. 361. 362. 363. 364. 365. 366. 367. 368. 369. 370. 371. 372. 373. 374. 375. 376. 377. 378. 379. 380. 381. 382. 383. 384. 385. 386. 387. 388. 389. 390. 391. 392. 393. 394. 395. 396. 397. 398. 399. 400. 401. 402. 403. 404. 405. 406. 407. 408. 409. 410. 411. 412. 413. 414. 415. 416. 417. 418. 419. 420. 421. 422. 423. 424. 425. 426. 427. 428. 429. 430. 431. 432. 433. 434. 435. 436. 437. 438. 439. 440. 441. 442. 443. 444. 445. 446. 447. 448. 449. 450. 451. 452. 453. 454. 455. 456. 457. 458. 459. 460. 461. 462. 463. 464. 465. 466. 467. 468. 469. 470. 471. 472. 473. 474. 475. 476. 477. 478. 479. 480. 481. 482. 483. 484. 485. 486. 487. 488. 489. 490. 491. 492. 493. 494. 495. 496. 497. 498. 499. 500. 501. 502. 503. 504. 505. 506. 507. 508. 509. 510. 511. 512. 513. 514. 515. 516. 517. 518. 519. 520. 521. 522. 523. 524. 525. 526. 527. 528. 529. 530. 531. 532. 533. 534. 535. 536. 537. 538. 539. 540. 541. 542. 543. 544. 545. 546. 547. 548. 549. 550. 551. 552. 553. 554. 555. 556. 557. 558. 559. 560. 561. 562. 563. 564. 565. 566. 567. 568. 569. 570. 571. 572. 573. 574. 575. 576. 577. 578. 579. 580. 581. 582. 583. 584. 585. 586. 587. 588. 589. 590. 591. 592. 593. 594. 595. 596. 597. 598. 599. 600. 601. 602. 603. 604. 605. 606. 607. 608. 609. 610. 611. 612. 613. 614. 615. 616. 617. 618. 619. 620. 621. 622. 623. 624. 625. 626. 627. 628. 629. 630. 631. 632. 633. 634. 635. 636. 637. 638. 639. 640. 641. 642. 643. 644. 645. 646. 647. 648. 649. 650. 651. 652. 653. 654. 655. 656. 657. 658. 659. 660. 661. 662. 663. 664. 665. 666. 667. 668. 669. 670. 671. 672. 673. 674. 675. 676. 677. 678. 679. 680. 681. 682. 683. 684. 685. 686. 687. 688. 689. 690. 691. 692. 693. 694. 695. 696. 697. 698. 699. 700. 701. 702. 703. 704. 705. 706. 707. 708. 709. 710. 711. 712. 713. 714. 715. 716. 717. 718. 719. 720. 721. 722. 723. 724. 725. 726. 727. 728. 729. 730. 731. 732. 733. 734. 735. 736. 737. 738. 739. 740. 741. 742. 743. 744. 745. 746. 747. 748. 749. 750. 751. 752. 753. 754. 755. 756. 757. 758. 759. 760. 761. 762. 763. 764. 765. 766. 767. 768. 769. 770. 771. 772. 773. 774. 775. 776. 777. 778. 779. 780. 781. 782. 783. 784. 785. 786. 787. 788. 789. 790. 791. 792. 793. 794. 795. 796. 797. 798. 799. 800. 801. 802. 803. 804. 805. 806. 807. 808. 809. 810. 811. 812. 813. 814. 815. 816. 817. 818. 819. 820. 821. 822. 823. 824. 825. 826. 827. 828. 829. 830. 831. 832. 833. 834. 835. 836. 837. 838. 839. 840. 841. 842. 843. 844. 845. 846. 847. 848. 849. 850. 851. 852. 853. 854. 855. 856. 857. 858. 859. 860. 861. 862. 863. 864. 865. 866. 867. 868. 869.

1)  $\frac{1}{2} \frac{d}{dt} (a_1^2 + a_2^2 + a_3^2) = 0$

# ENCODING



# Dialogue between Industry and Academia

Binary Session Types [\[PARL'94, ESOP'98\]](#)



Milner, Honda and Yoshida joined W3C WS-CDL (2002)



Formalisation of W3C WS-CDL [\[ESOP'07\]](#)



Scribble at  $\pi^4$  Technology

# CDL Equivalent

- Basic example:

```
package HelloWorld {  
    roleType YouRole, WorldRole;  
    participantType You{YouRole}, World{WorldRole};  
    relationshipType YouWorldRel between YouRole and WorldRole;  
    channelType WorldChannelType with roleType WorldRole;  
  
    choreography Main {  
        WorldChannelType worldChannel;  
  
        interaction operation=hello from=YouRole to=WorldRole  
            relationship=YouWorldRel channel=worldChannel {  
                request messageType=Hello;  
            }  
        }  
    }  
}
```

# Scribble Protocol

- *"Scribbling is necessary for architects, either physical or computing, since all great ideas of architectural construction come from that unconscious moment, when you do not realise what it is, when there is no concrete shape, only a whisper which is not a whisper, an image which is not an image, somehow it starts to urge you in your mind, in so small a voice but how persistent it is, at that point you start scribbling" - Kohei Honda 2007*
- **Basic example:**

```
protocol HelloWorld {  
  role You, World;  
  Hello from You to World;  
}
```

# Dialogue between Industry and Academia

Binary Session Types [\[PARL'94, ESOP'98\]](#)



Milner, Honda and Yoshida joined W3C WS-CDL (2002)



Formalisation of W3C WS-CDL [\[ESOP'07\]](#)



Scribble at  $\pi^4$  Technology



Multiparty Session Types [\[POPL'08\]](#)



# Dialogue between Industry and Academia

Binary Session Types [PARL'94, ESOP'98]



Milner, Honda and Yoshida joined W3C WS-CDL (2002)



Formalisation of W3C WS-CDL [ESOP'07]



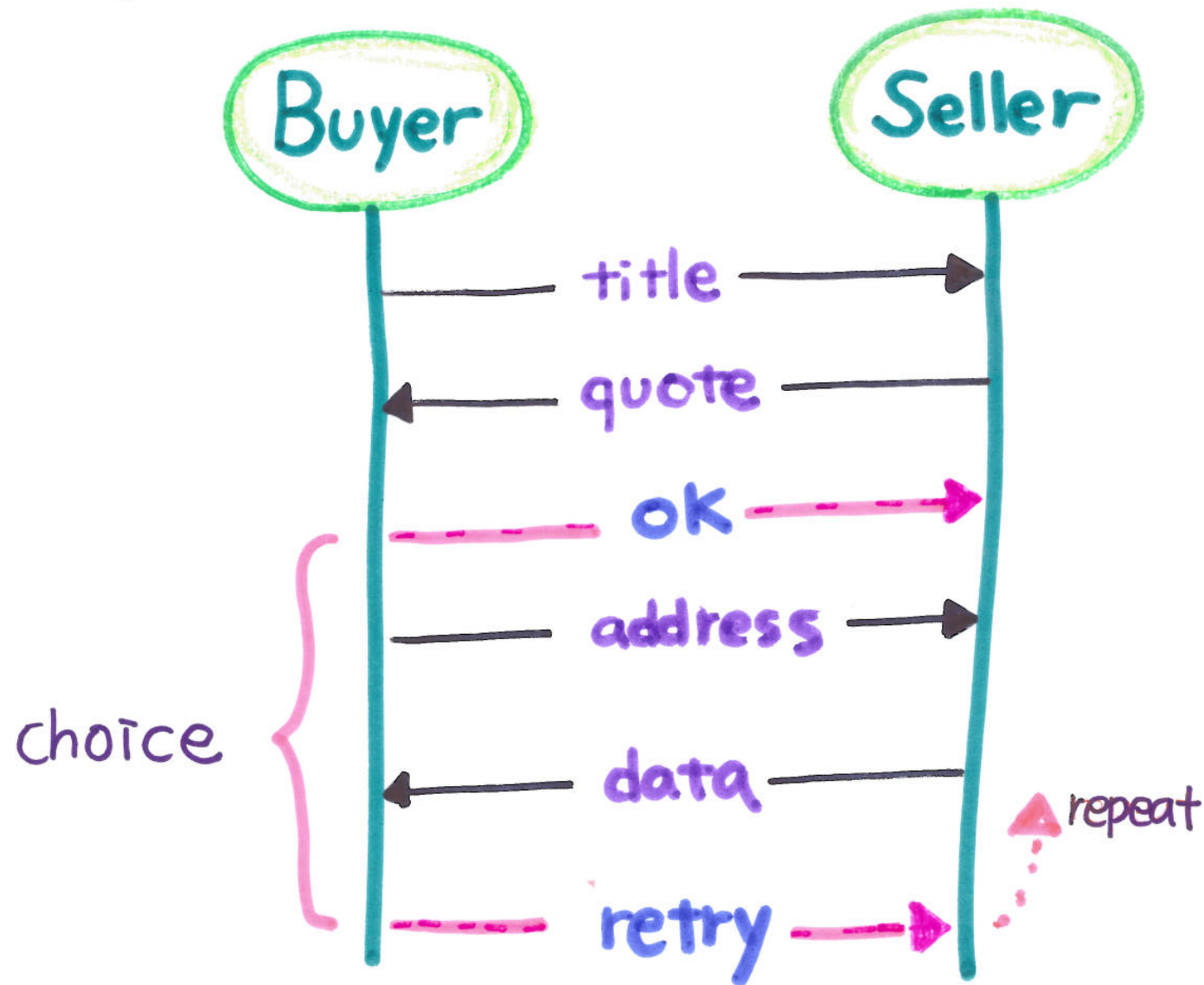
Scribble at  $\pi^4$  Technology



Multiparty Session Types [POPL'08]

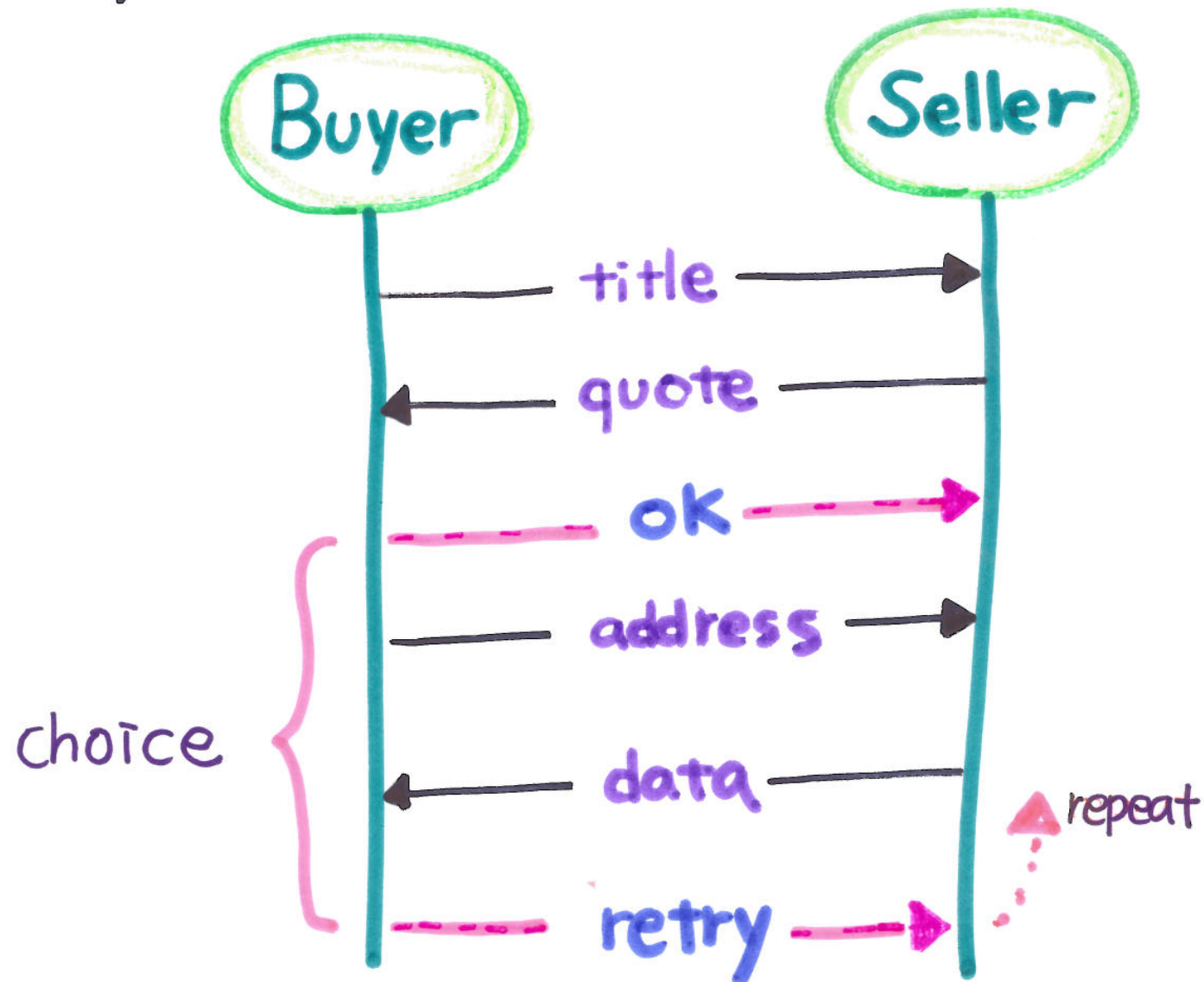


# Binary Session Types: Buyer - Seller Protocol

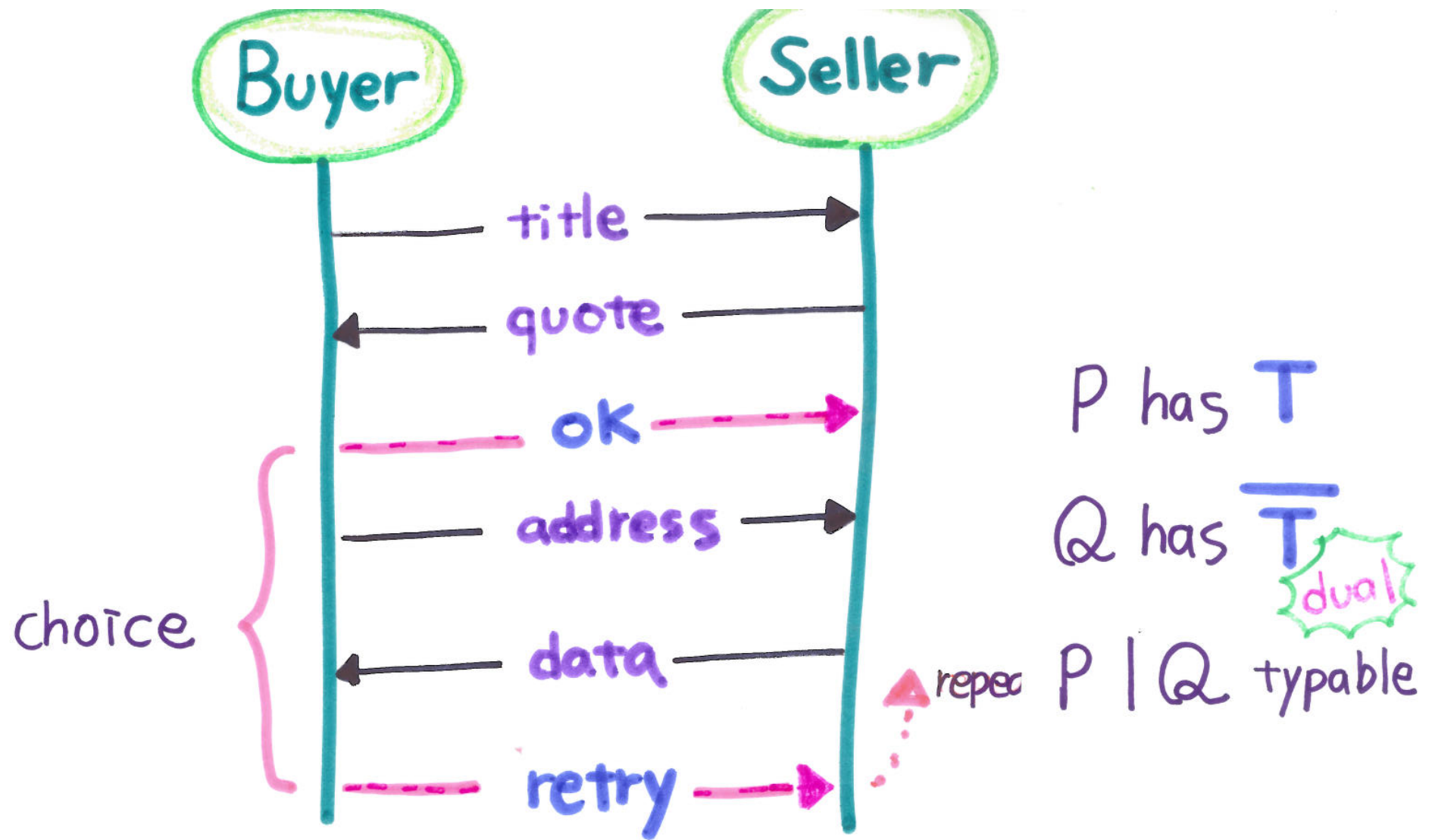




# Binary Session Types: Buyer - Seller Protocol



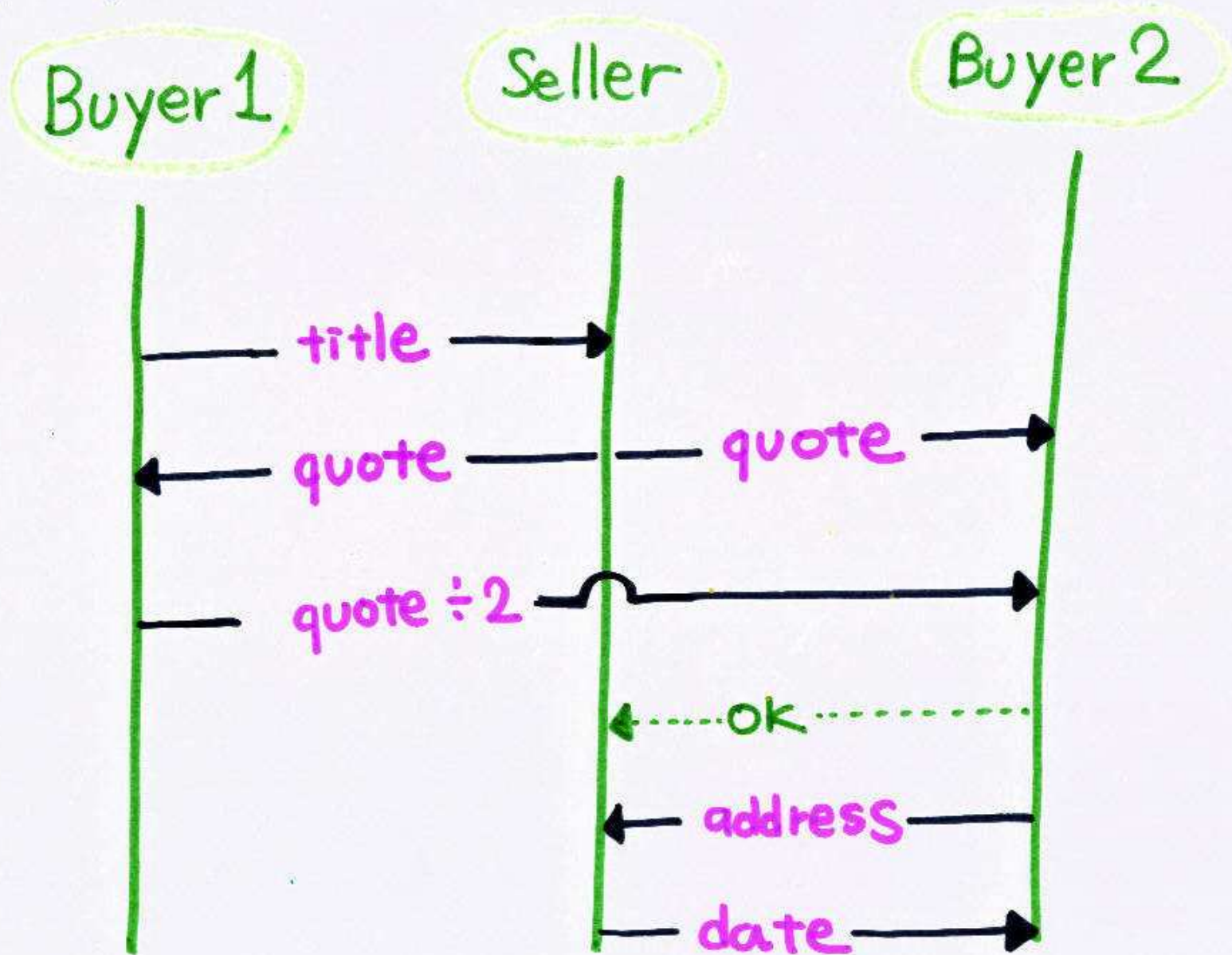
nt! Title ; ? Quote ; ! { **ok**: ! Add ; ? Date , **retry** : t }

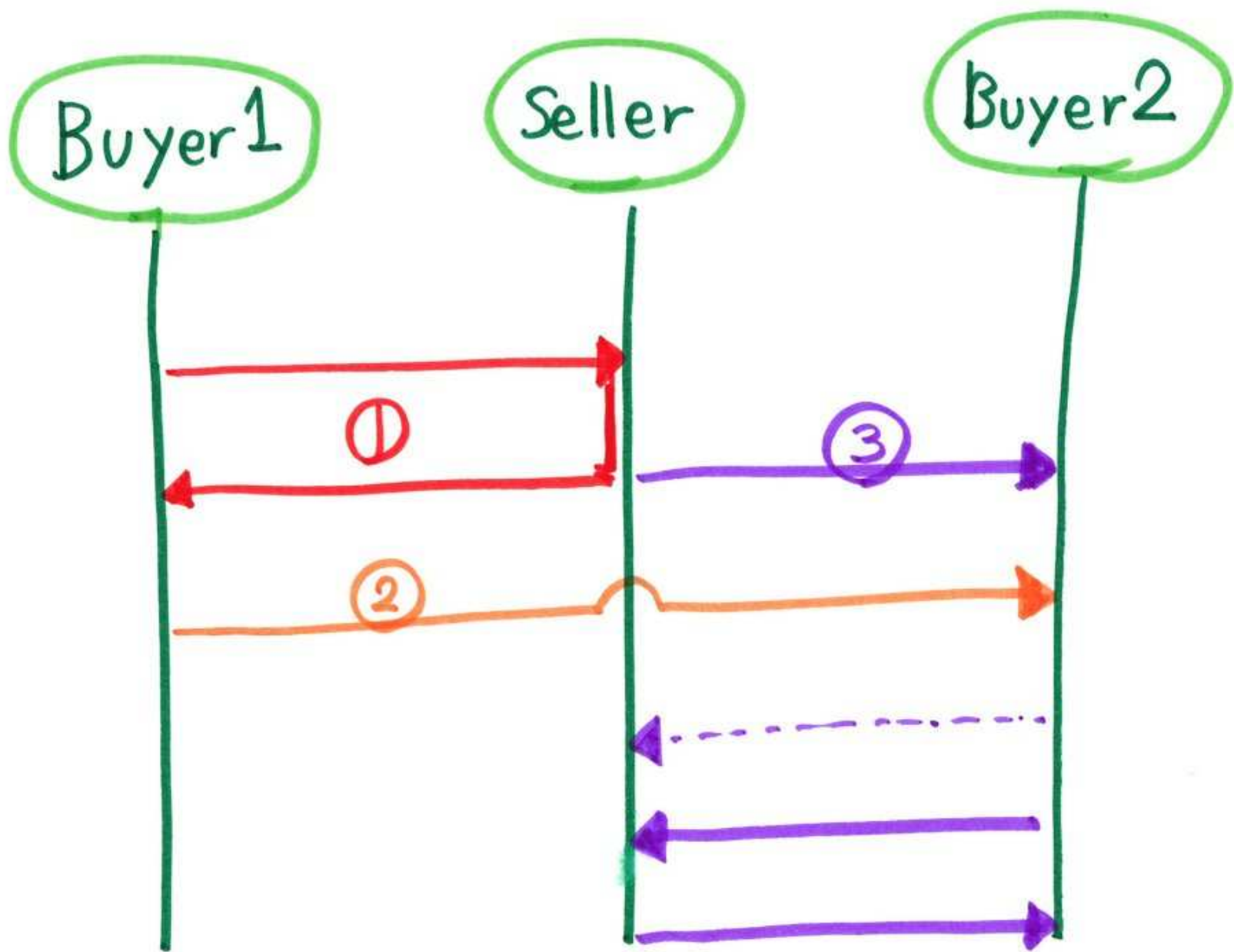


nt! Title ; ? Quote ; ! { ok: ! Add ; ? Date, retry: t }

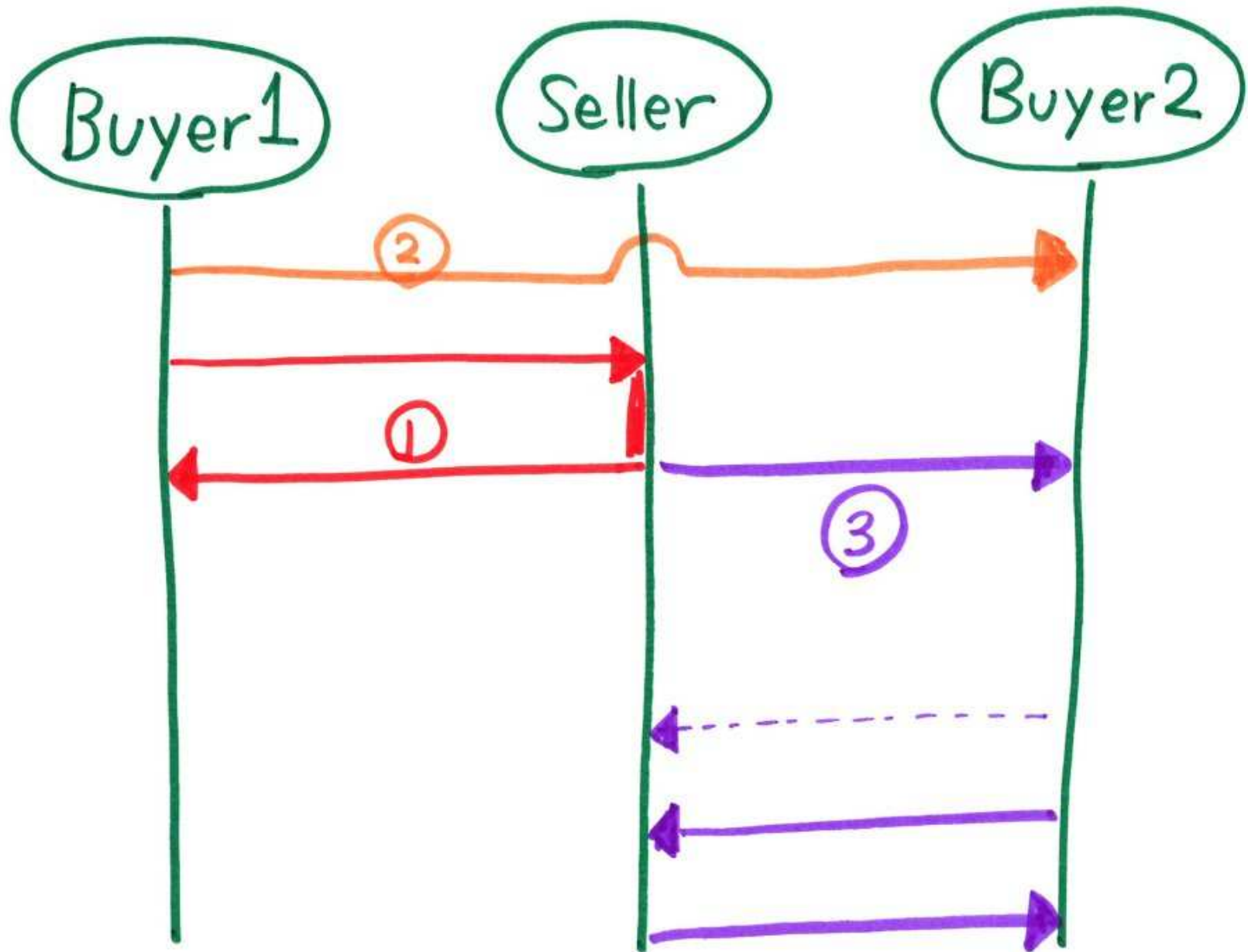
nt? Title ; ! Quote; ? { ok: ? Add; ! Date, retry: t }

# Multiparty Session Types

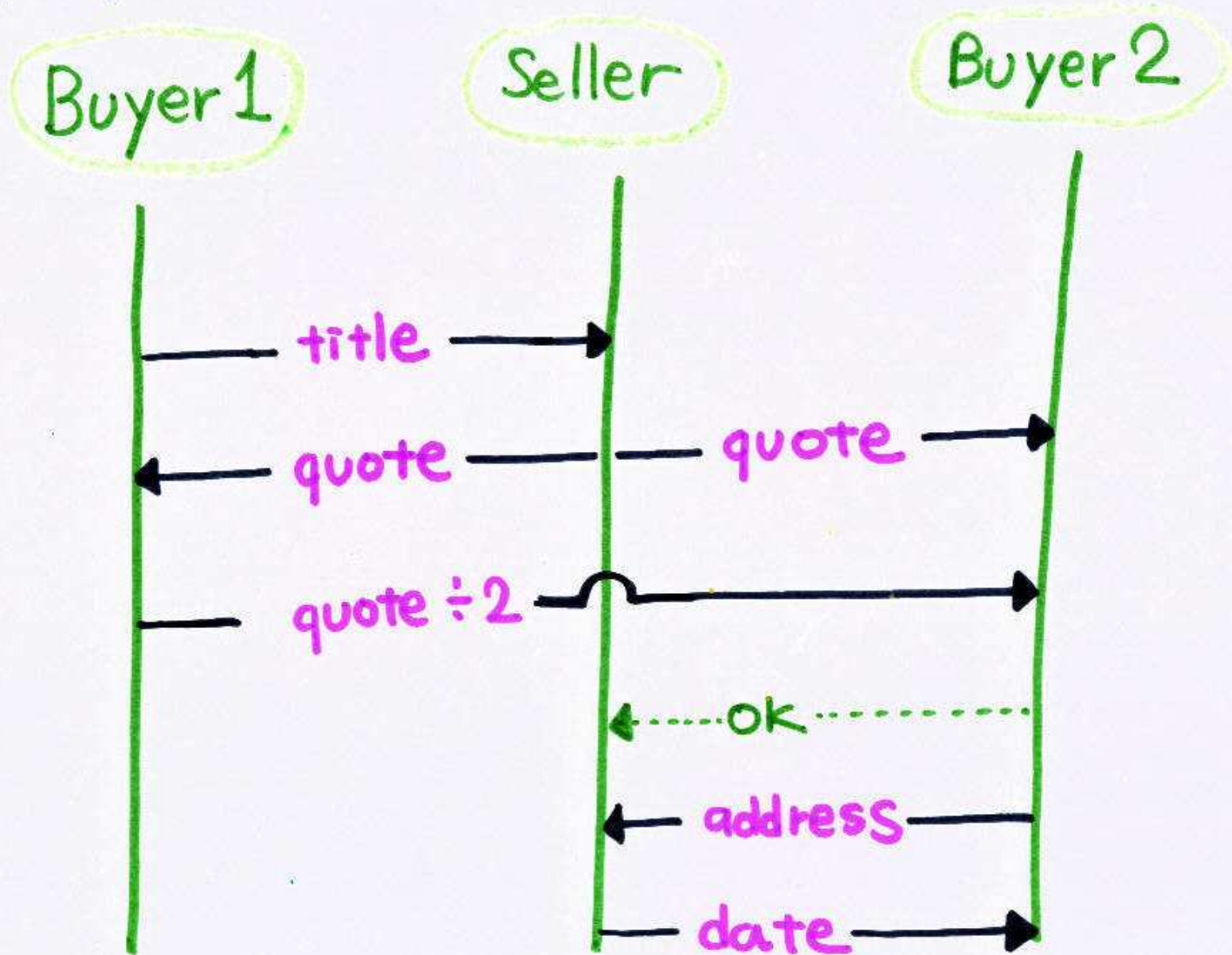








# Multiparty Session Types



Alice

Bob

Carol

CA?c ; AB!a

AB?a ; BC!b

BC?b ; CA?c

dual

dual

dual

3 dual  
pairs

If you use

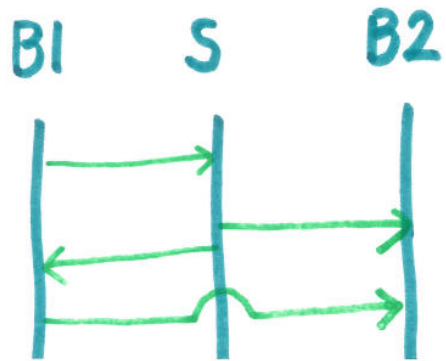
binary Session  
Types ....

Deadlock!



# Multiparty Session Types

[Honda, Yoshida, Carbone 2008]



(G)

$B1 \rightarrow S$  Int.

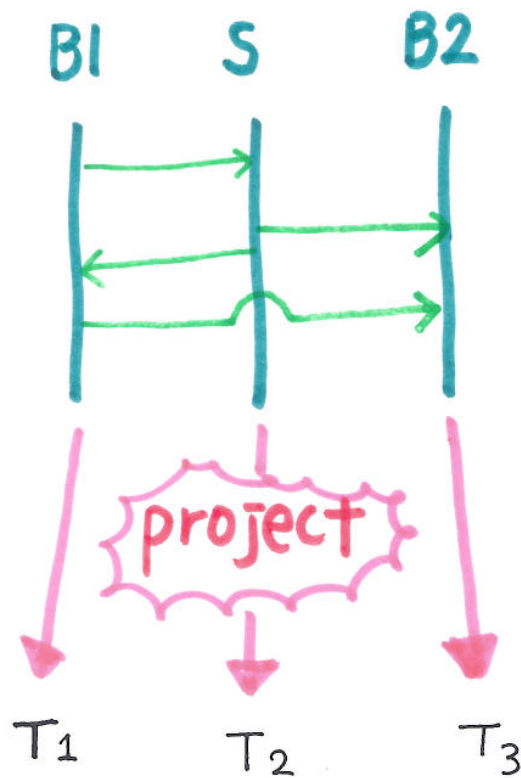
$S \rightarrow B2$  Char

STEP 1

Write Global Type

# Multiparty Session Types

[Honda, Yoshida, Carbone 2008]



(G)

$B_1 \rightarrow S$  Int.

$S \rightarrow B_2$  Char

(T)

$B_1 ? \text{Int}. B_2 ! \text{Char}$

STEP 1

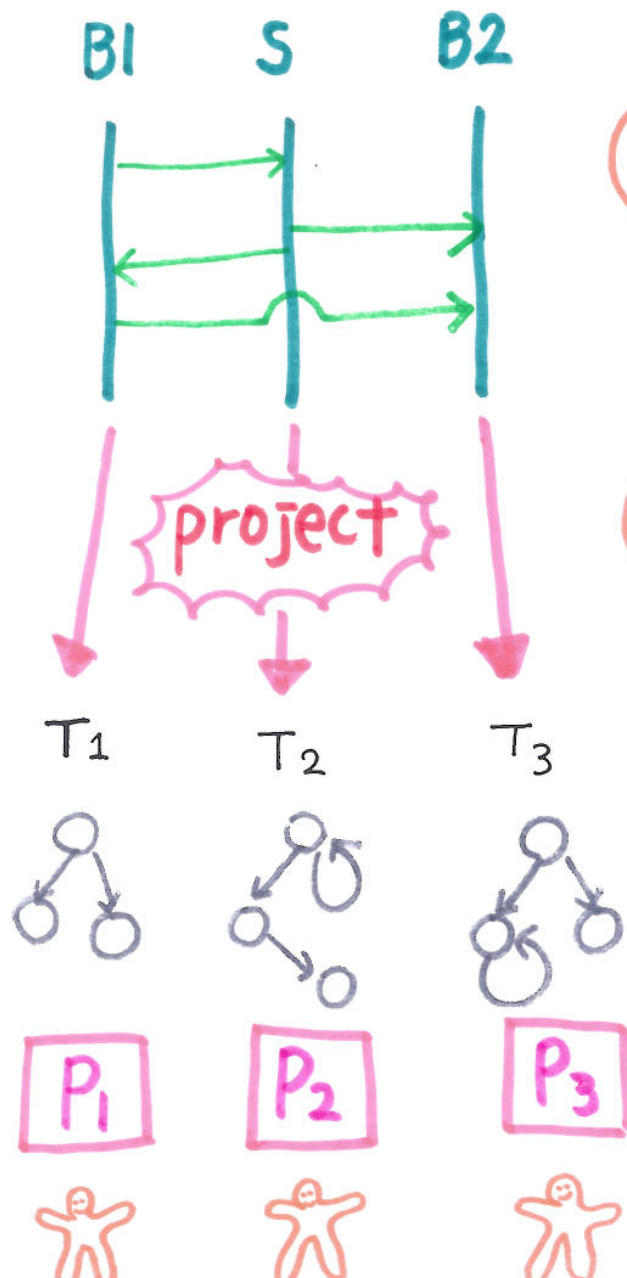
Write Global Type

STEP 2

Project to Local Types

# Multiparty Session Types

[Honda, Yoshida, Carbone 2008]



(G)

$B1 \rightarrow S \text{ Int.}$

$S \rightarrow B2 \text{ Char}$

(T)

$B1? \text{Int. } B2! \text{Char}$

(P)

$B1?(x). B2! \langle \text{"apple"} \rangle$

STEP 1

Write Global Type

STEP 2

Project to Local Type

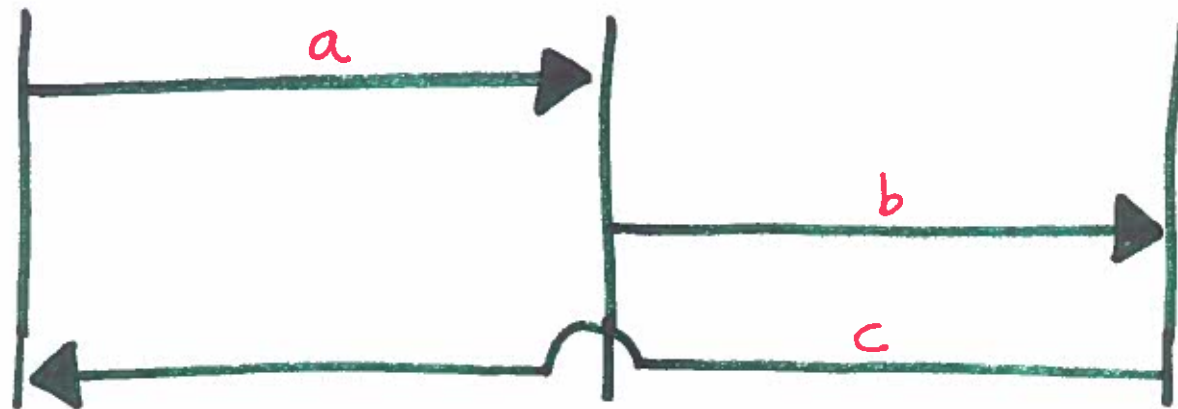
STEP 3

- Static Check
- Generate Code
- Run-time check

Alice

Bob

Carol



Global  
Type

Projection



Alice  $AB!a; CA?c$

Bob  $AB?a; BC!b$

Carol  $BC?b; CA!c;$

LOCAL  
TYPES

NO  
Deadlock

# Us ∈ **M**obility **R**esearch **G**roup



## MobilityReadingGroup

$\pi$ -calculus, Session Types research at Imperial College

[Home](#) [People](#) [Publications](#) [Grants](#) [Talks](#) [Tools](#) [Awards](#) [Kohel Honda](#)

## NEWS

Our recent work [Fencing off Go: Liveness and Safety for Channel-based Programming](#) was summarised on [The Morning Paper](#) blog.

2 Feb 2017

Weizhen passed her viva today, congratulations Dr. Yang!

24 Jan 2017

Mariangiola Dezani-Ciancaglini, a long-term collaborator with our group working on Session Types turns 70 today, more details [here](#).

23 Dec 2016

Rumyana passed her viva today,

## SELECTED PUBLICATIONS

2017

Raymond Hu , Nobuko Yoshida : [Explicit Connection Actions in Multiparty Session Types](#). *To appear in FASE 2017* .

Julien Lange , Nicholas Ng , Bernardo Toninho , Nobuko Yoshida : [Fencing off Go: Liveness and Safety for Channel-based Programming](#). POPL 2017 .

Rumyana Neykova , Nobuko Yoshida : [Let It Recover: Multiparty Protocol-Induced Recovery](#). CC 2017 .

Julien Lange , Nobuko Yoshida : [On the Undecidability of Asynchronous Session Subtyping](#). *To appear in FoSSaCS 2017* .

<http://mrg.doc.ic.ac.uk/>

Academic Staff

Nobuko Yoshida

Research Associate

Raymond Hu

Julien Lange

Nicholas Ng

Xinyu Niu

Alceste Scalas

Bernardo Toninho

PhD Student

Assel Altayeva

Juliana Franco

Rumyana Neykova

Weizhen Yang

# Selected Publications 2016/2017



- **[ECOOP'17]** Alceste Scala, Raymond Hu, Ornela Darda, NY: A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming..
- **[COORDINATION'17]** Keigo Imai, NY and Shoji Yuen: Session-ocaml: a session-based library with polarities and lenses.
- **[FoSSaCS'17]** Julien Lange , NY: On the Undecidability of Asynchronous Session Subtyping.
- **[FASE'17]** Raymond Hu , NY: Explicit Connection Actions in Multiparty Session Types.
- **[CC'17]** Rumyana Neykova , NY: Let It Recover: Multiparty Protocol-Induced Recovery.
- **[POPL'17]** Julien Lange , Nicholas Ng , Bernardo Toninho , NY: Fencing off Go: Liveness and Safety for Channel-based Programming.
- **[FPL'16]** Xinyu Niu , Nicholas Ng , Tomofumi Yuki , Shaojun Wang , NY, Wayne Luk : EURECA Compilation: Automatic Optimisation of Cycle-Reconfigurable Circuits.
- **[ECOOP'16]** Alceste Scala, NY: Lightweight Session Programming in Scala
- **[CC'16]** Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
- **[FASE'16]** Raymond Hu, NY: Hybrid Session Verification through Endpoint API Generation.
- **[TACAS'16]** Julien Lange, NY: Characteristic Formulae for Session Types.
- **[ESOP'16]** Dimitrios Kouzapas, Jorge A. Pérez, NY: On the Relative Expressiveness of Higher-Order Session Processes.
- **[POPL'16]** Dominic Orchard, NY: Effects as sessions, sessions as effects .





# Selected Publications 2016/2017

- [ECOOP'17] Alceste Scala, Raymond Hu, Ornela Darda, NY :A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming.
- [COORDINATION'17] Keigo Imai, NY and Shoji Yuen: Session-ocaml: a session-based library with polarities and lenses.
- [FoSSaCS'17] Julien Lange , NY : On the Undecidability of Asynchronous Session Subtyping.
- [FASE'17] Raymond Hu , NY : Explicit Connection Actions in Multiparty Session Types.
- [CC'17] Rumyana Neykova , NY: Let It Recover: Multiparty Protocol-Induced Recovery.
- [POPL'17] Julien Lange , Nicholas Ng , Bernardo Toninho , NY: Fencing off Go: Liveness and Safety for Channel-based Programming.
- [FPL'16] Xinyu Niu , Nicholas Ng , Tomofumi Yuki , Shaojun Wang , NY, Wayne Luk: EURECA Compilation: Automatic Optimisation of Cycle-Reconfigurable Circuits.
- [ECOOP'16] Alceste Scala, NY: Lightweight Session Programming in Scala
- [CC'16] Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
- [FASE'16] Raymond Hu, NY: Hybrid Session Verification through Endpoint API Generation.
- [TACAS'16] Julien Lange, NY: Characteristic Formulae for Session Types.
- [ESOP'16] Dimitrios Kouzapas, Jorge A. Pérez, NY: On the Relative Expressiveness of Higher-Order Session Processes.
- [POPL'16] Dominic Orchard, NY: Effects as Sessions, Sessions as Effects.

# Relationship with Communicating AUTOMATA

ESOP 12

Characterisation

ICALP 13

Synthesis

CONCUR 15

Timed

Automata

POPL 15

Graphical

Synthesis

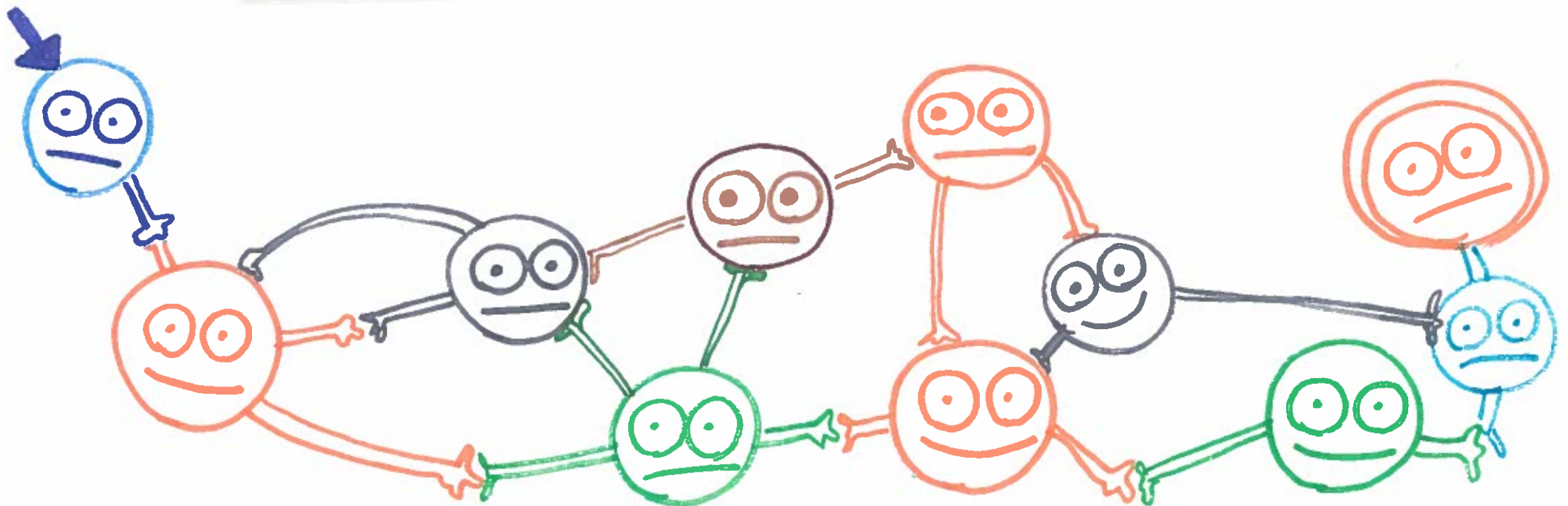
TACAS 16

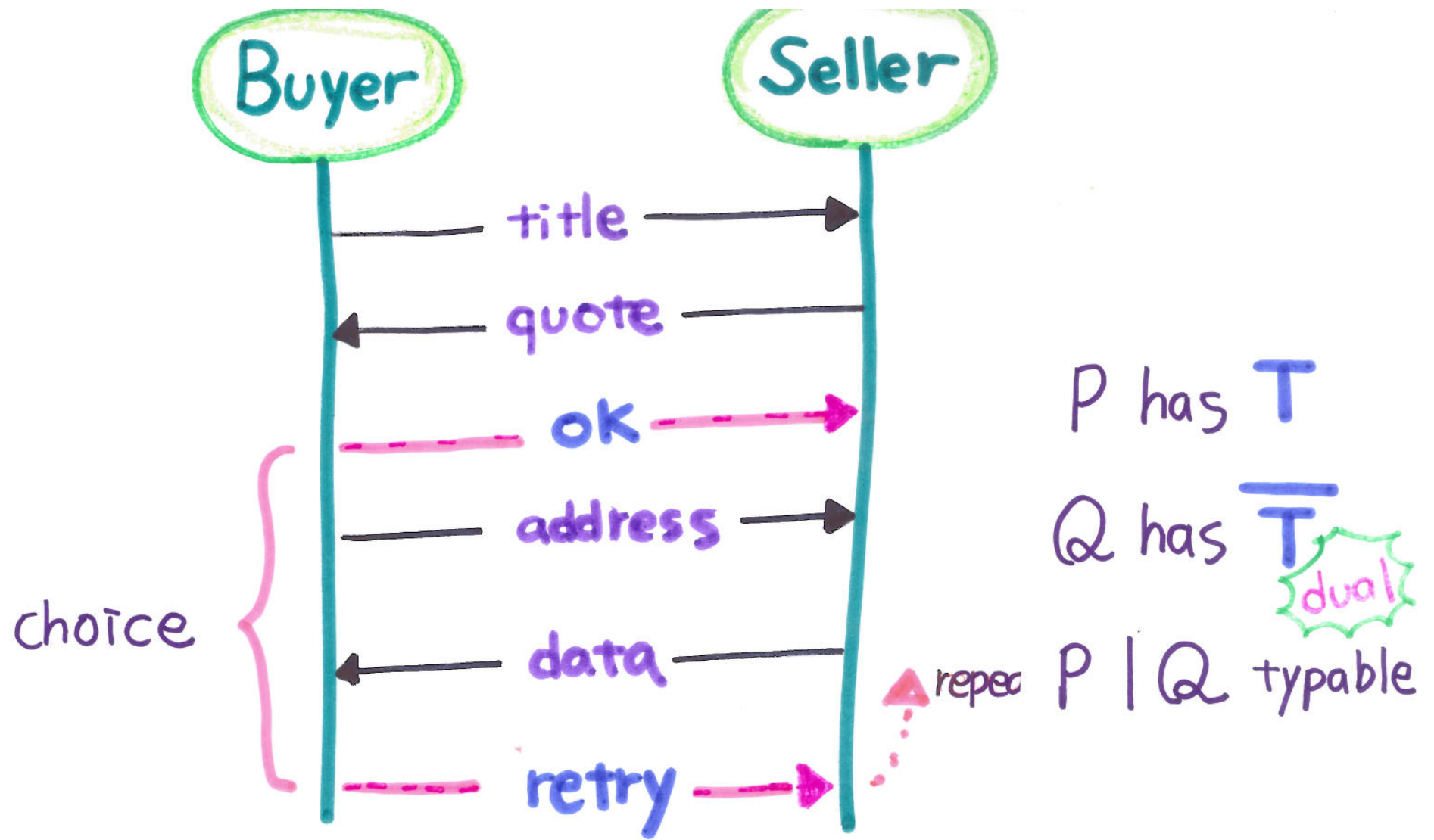
Subtyping

+ Model-checking

FOSACS 17

Undecidability

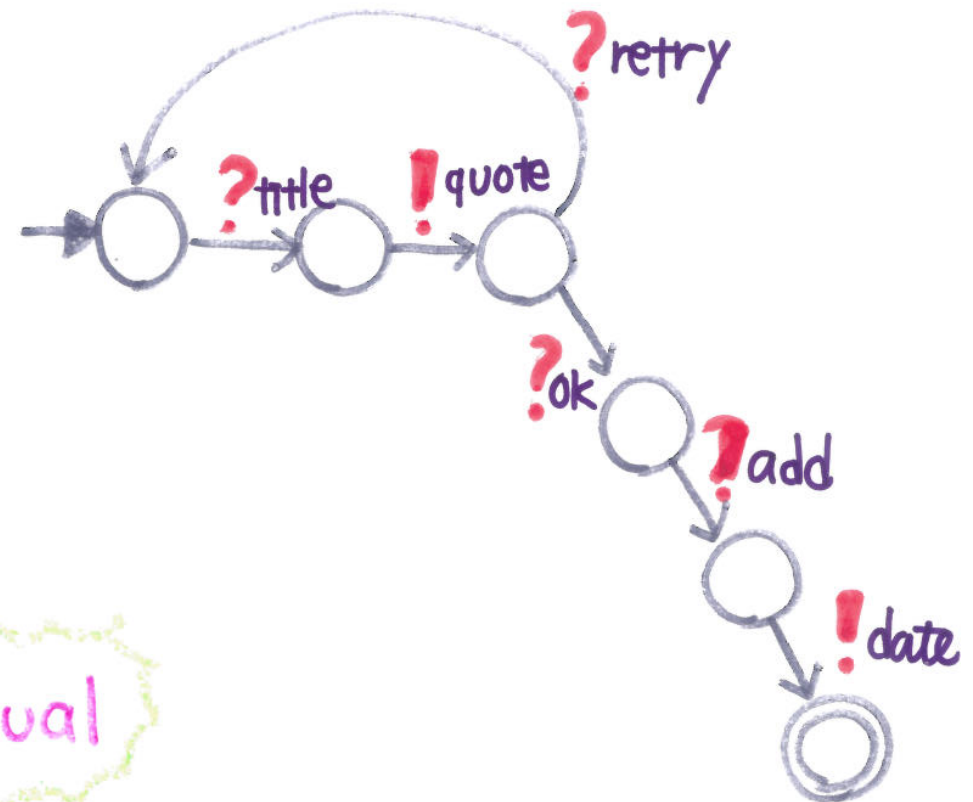
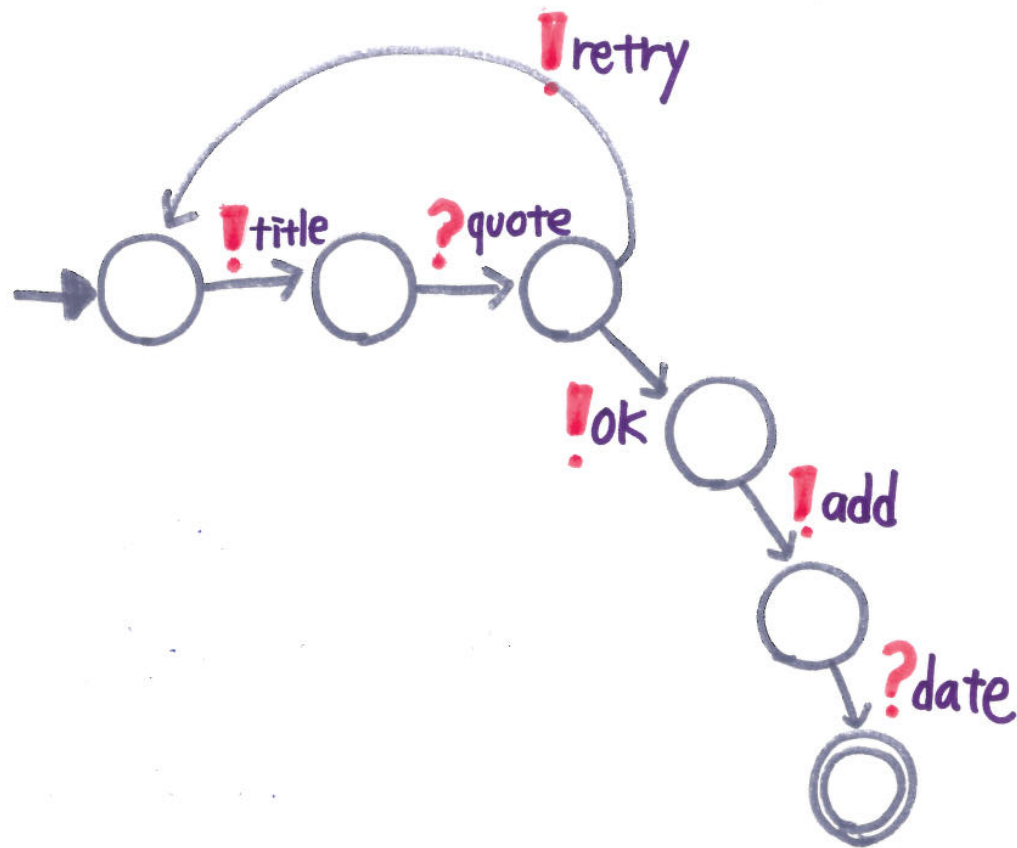




nt! Title ; ? Quote ; ! { ok: ! Add ; ? Date, retry: t }

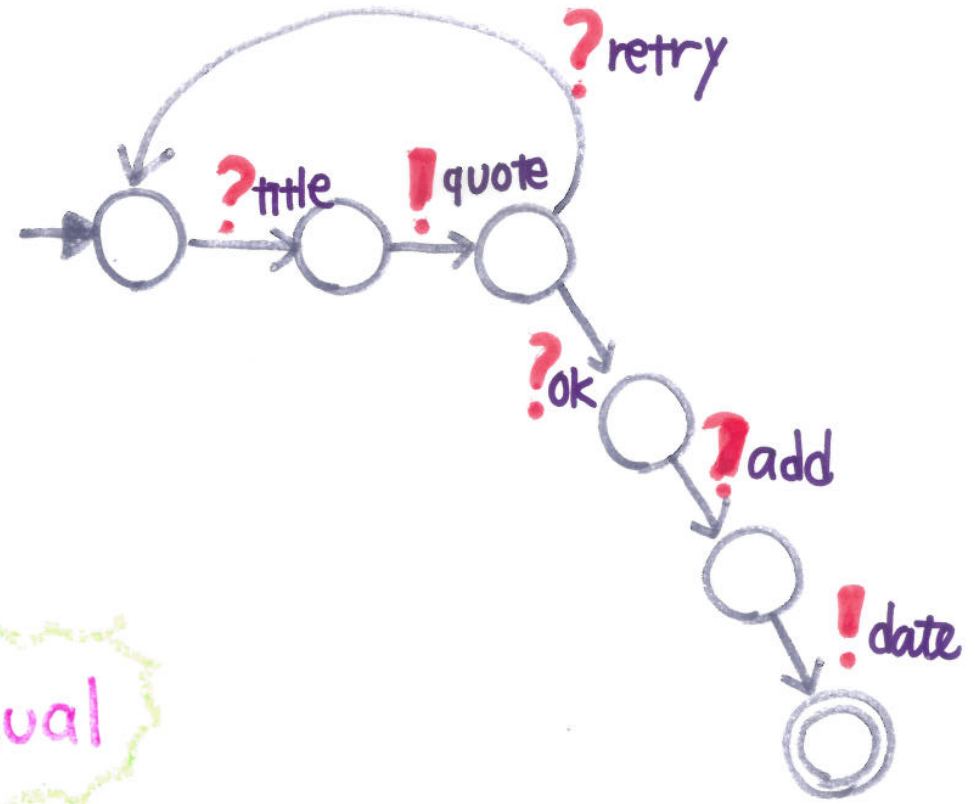
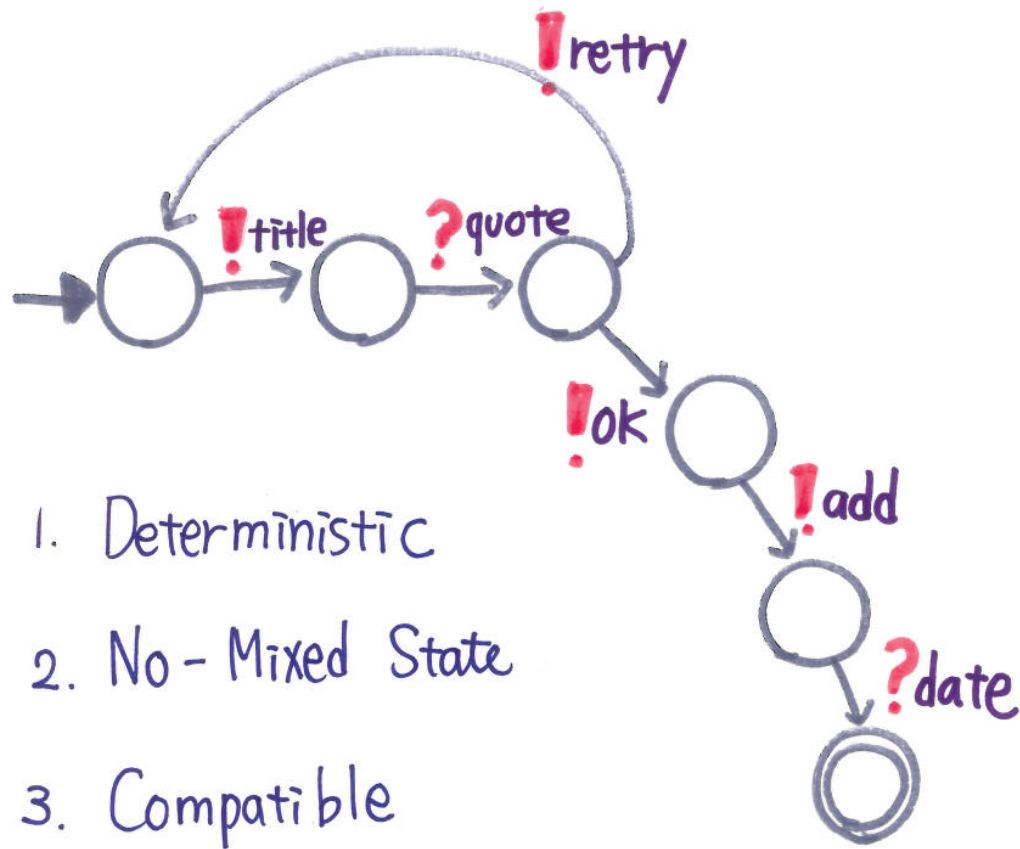
nt? Title ; ! Quote; ? { ok: ? Add; ! Date, retry: t }

# Communicating Automata [1980s]



dual



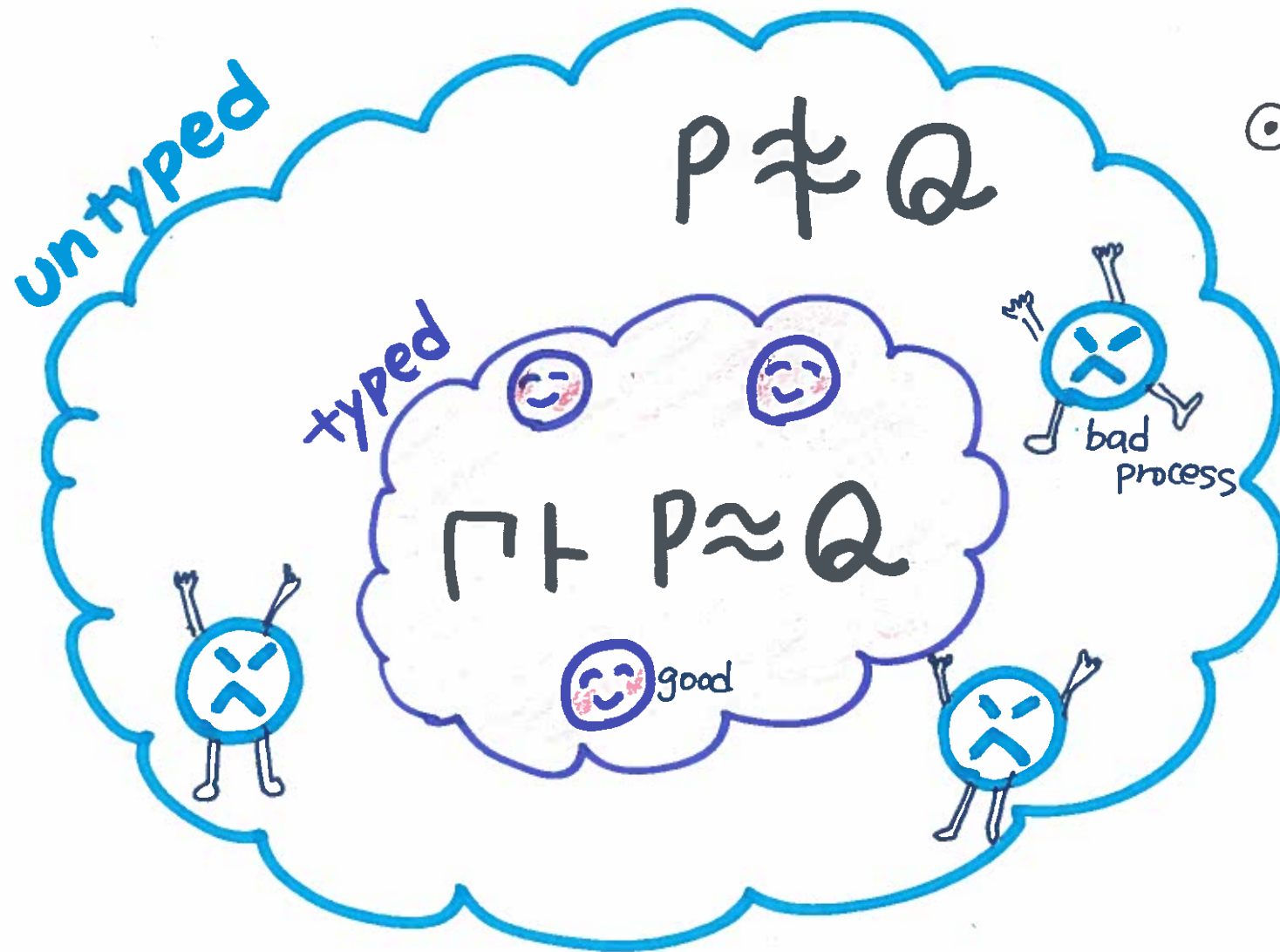


dual

**[Gouda et al 1986]** Two compatible machines without mixed states which are deterministic satisfy deadlock-freedom.

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

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



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



# GLOBALLY GOVERNED SESSION SEMANTICS

CONCUR 15  
LMCS



Dimitrios  
Kouzapas  
Glasgow



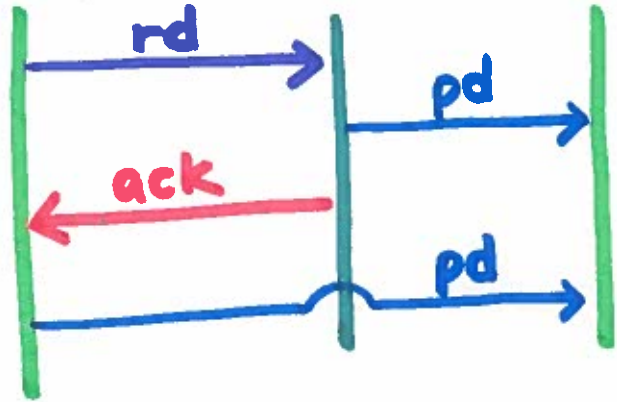
Nobuko  
Yoshida

Imperial  
College London



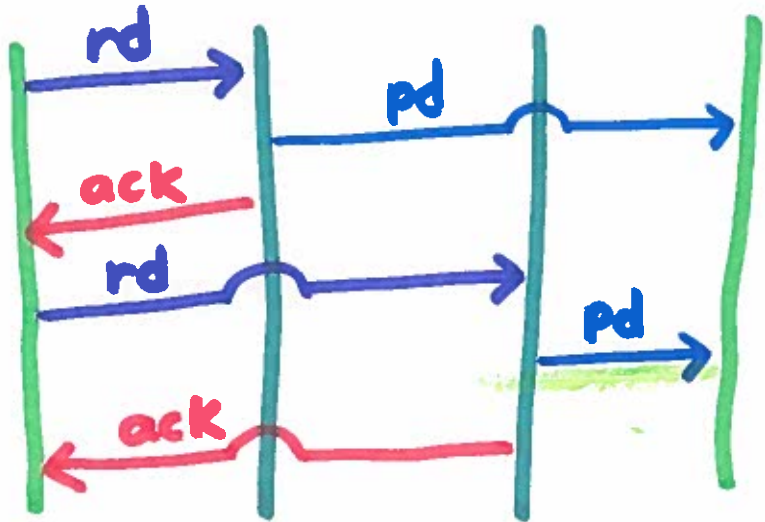
# Usecase : UC.R2.13 "Acquire Data from Instrument" from Ocean Observatories Initiative (OOI)

Instrument Agent User

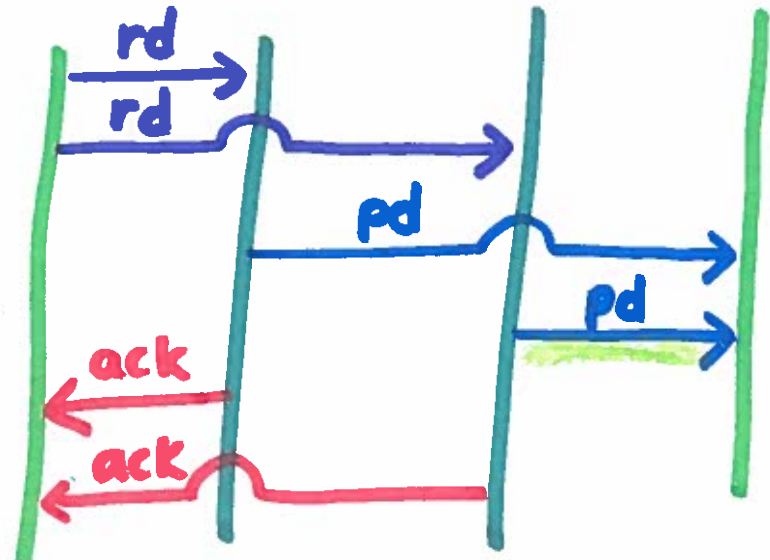


$A1 \rightarrow U: \langle pd \rangle.$   
 $A2 \rightarrow U: \langle pd \rangle.$   
end

I A1 A2 U

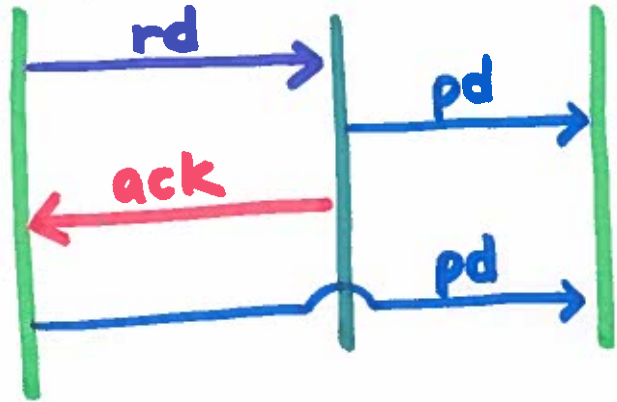


I A1 A2 U



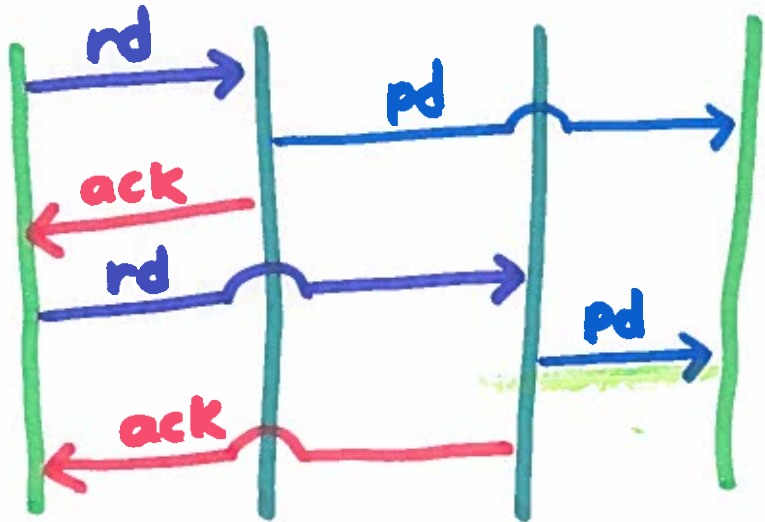
# Usecase : UC.R2.13 "Acquire Data from Instrument" from Ocean Observatories Initiative (OOI)

Instrument Agent User



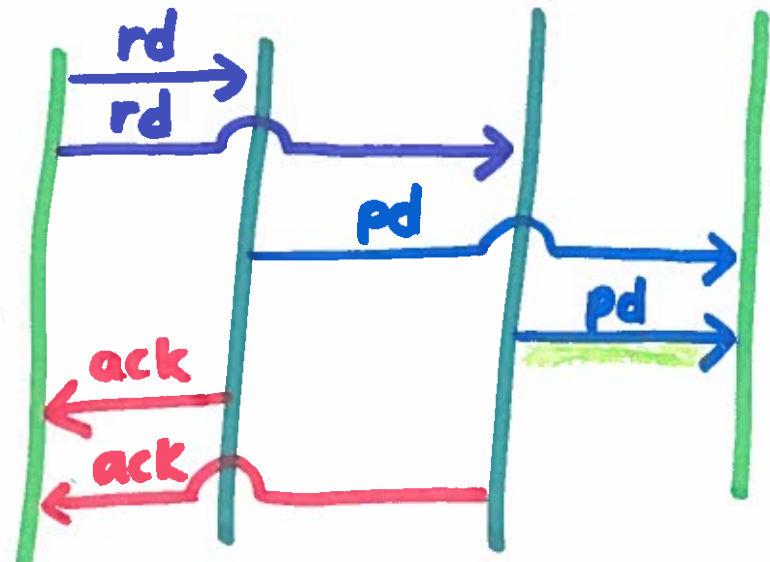
$A1 \rightarrow U: \langle pd \rangle.$   
 $A2 \rightarrow U: \langle pd \rangle.$   
end

I A1 A2 U



$\approx g$

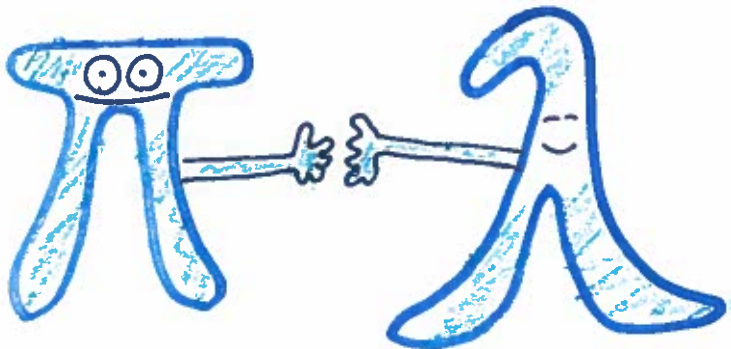
I A1 A2 U





# HOT $\pi$ and

# TYPES

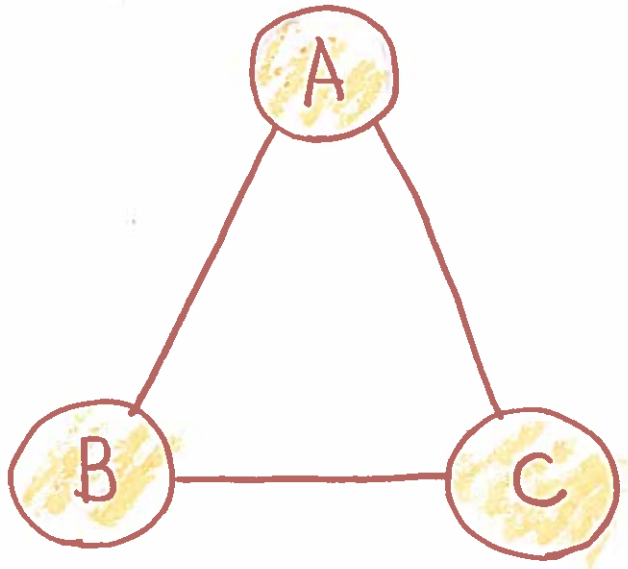


CONCUR 16 Characteristic  
Bisimulations

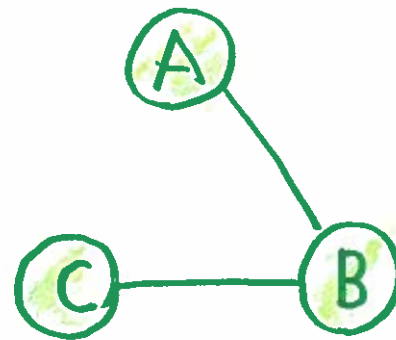
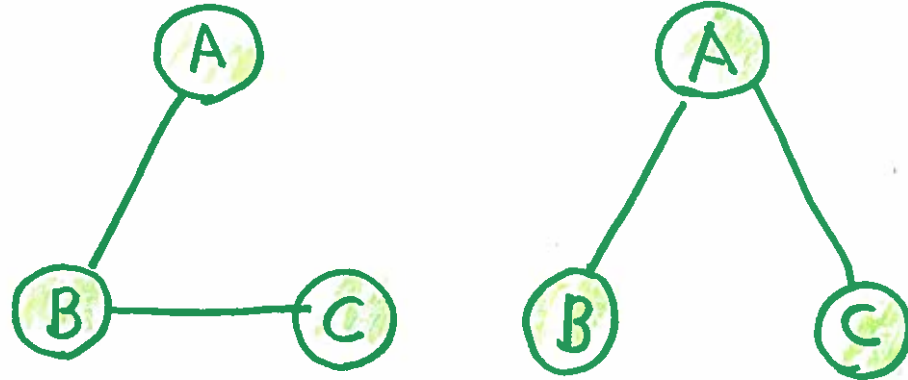
ESOP 16 Expressiveness

ACTA INFORMATICA

# Interconnectability of Session Logical Processes



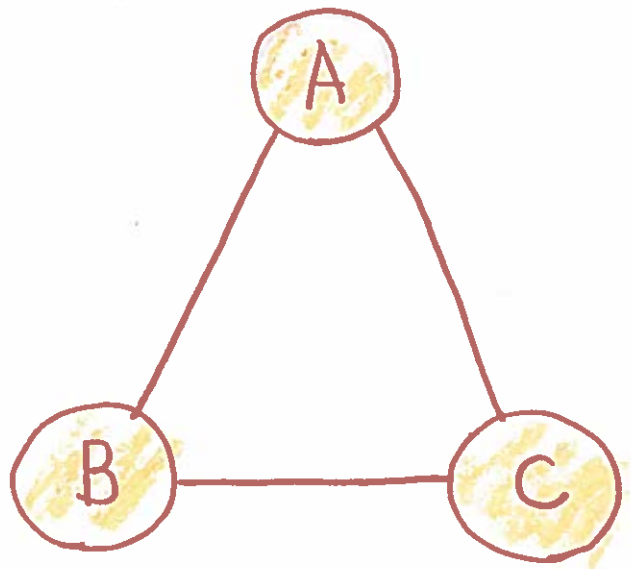
Multi Party



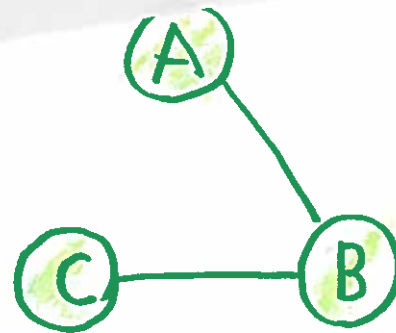
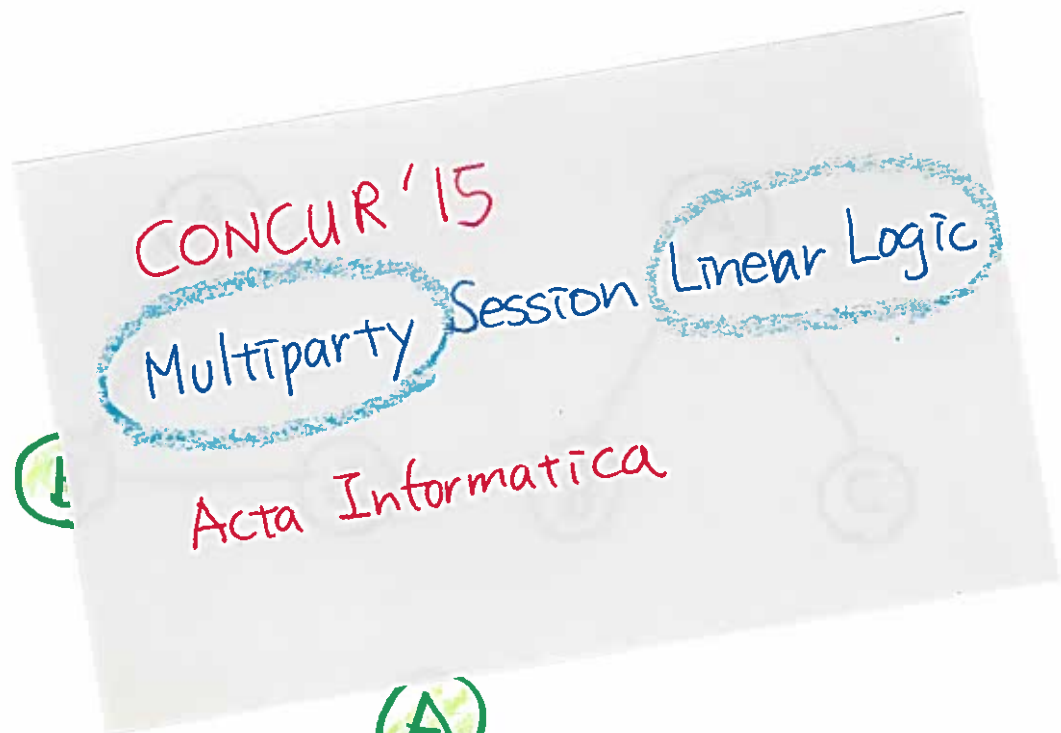
Linear Logical Session

cf. [1995] Specification Structures and Proposition  
ac TVBDC

# Interconnectability of Session Logical Processes



Multi Party



Linear Logical Session

cf. [1995] Specification Structures and Proposition  
as TVBAC





## Scribble: Describing Multi Party Protocols

Scribble is a language to describe application-level protocols among communicating systems. A protocol represents an agreement on how participating systems interact with each other. Without a protocol, it is hard to do meaningful interaction: participants simply cannot communicate effectively, since they do not know when to expect the other parties to send data, or whether the other party is ready to receive data. However, having a description of a protocol has further benefits. It enables verification to ensure that the protocol can be implemented without resulting in unintended consequences, such as deadlocks.

### Describe

Scribble is a language for describing multiparty protocols from a global, or endpoint neutral, perspective.

### Verify

Scribble has a theoretical foundation, based on the Pi Calculus and Session Types, to ensure that protocols described using the language are sound, and do not suffer from deadlocks or livelocks.

### Project

Endpoint projection is the term used for identifying the responsibility of a particular role (or endpoint) within a protocol.

### Implement

Various options exist, including (a) using the endpoint projection for a role to generate a skeleton code, (b) using session type APIs to clearly describe the behaviour, and (c) statically verify the code against the projection.

### Monitor

Use the endpoint projection for roles defined within a Scribble protocol, to monitor the activity of a particular endpoint, to ensure it correctly implements the expected behaviour.

# Online tool : <http://scribble.doc.ic.ac.uk/>

```
1 module examples;
2
3 ▾ global protocol HelloWorld(role Me, role World) {
4   hello() from Me to World;
5   choice at World {
6     goodMorning1() from World to Me;
7   } or {
8     goodMorning1() from World to Me;
9   }
10 }
11
```

Load a sample ▾

Check

Protocol:

Role:

Project

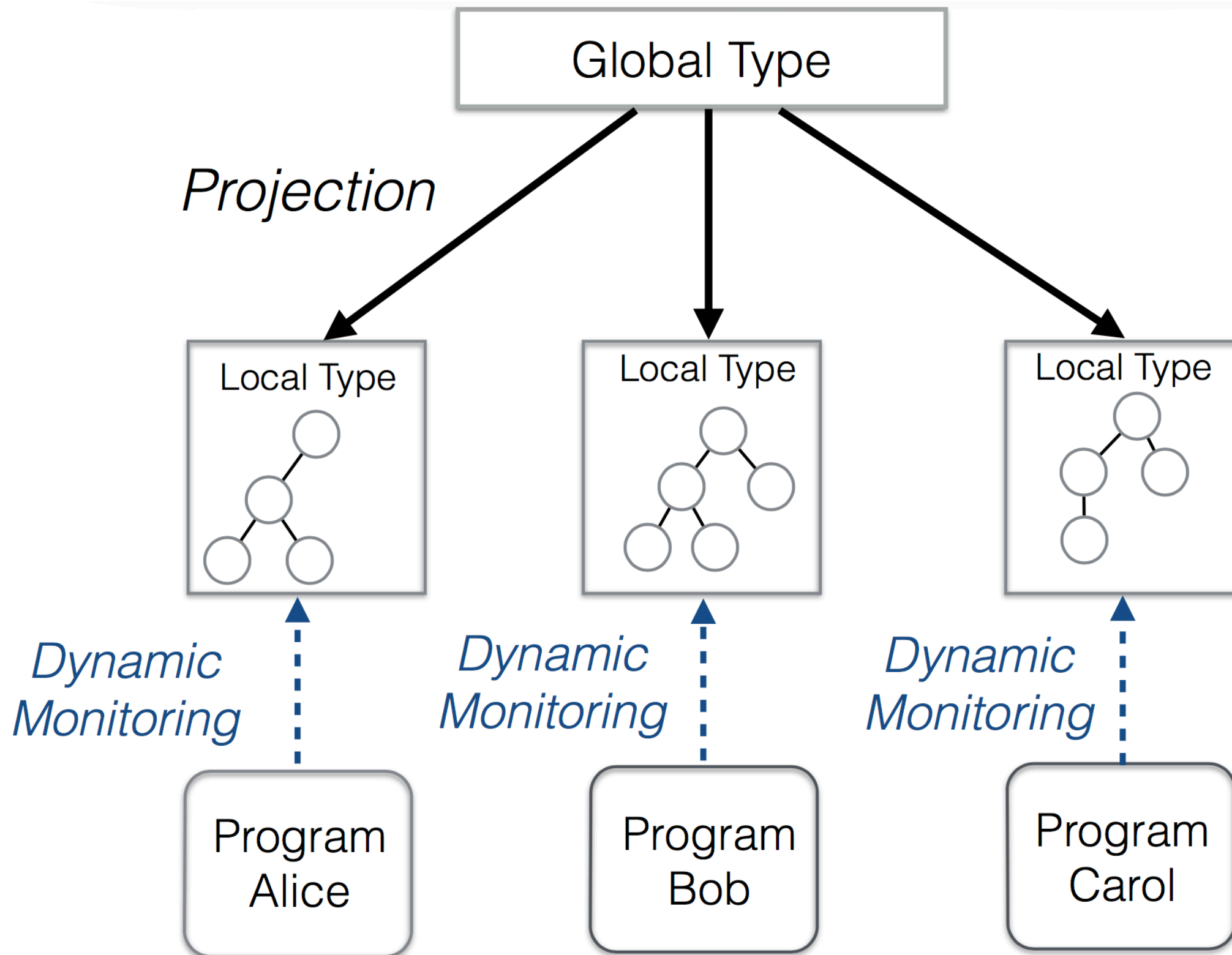
Generate Graph

[illegible]

- **TCS'16:** Monitoring Networks through Multiparty Session Types. Laura Bocchi , Tzu-Chun Chen , Romain Demangeon , Kohei Honda , Nobuko Yoshida
- **LMCS'16:** Multiparty Session Actors. Romyana Neykova, Nobuko Yoshida
- **FMSD'15:** Practical interruptible conversations: Distributed dynamic verification with multiparty session types and Python. Romain Demangeon , Kohei Honda , Raymond Hu , Romyana Neykova , Nobuko Yoshida
- **TGC'13:** The Scribble Protocol Language. Nobuko Yoshida , Raymond Hu , Romyana Neykova , Nicholas Ng

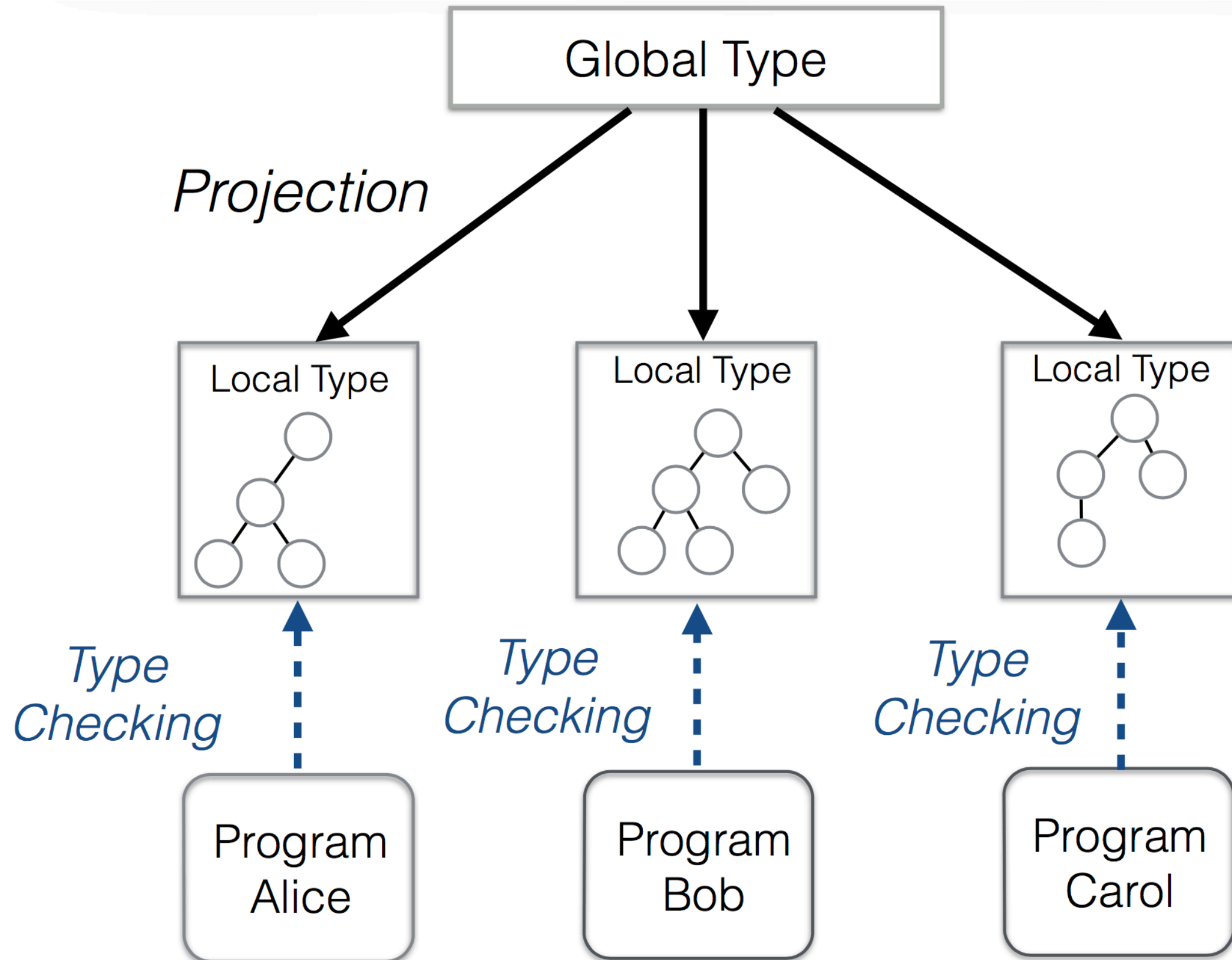
# Dynamic Monitoring

[RV'13, COORDINATION'14, FMDS'15, LMCS'17, CC'17]



# Type Checking

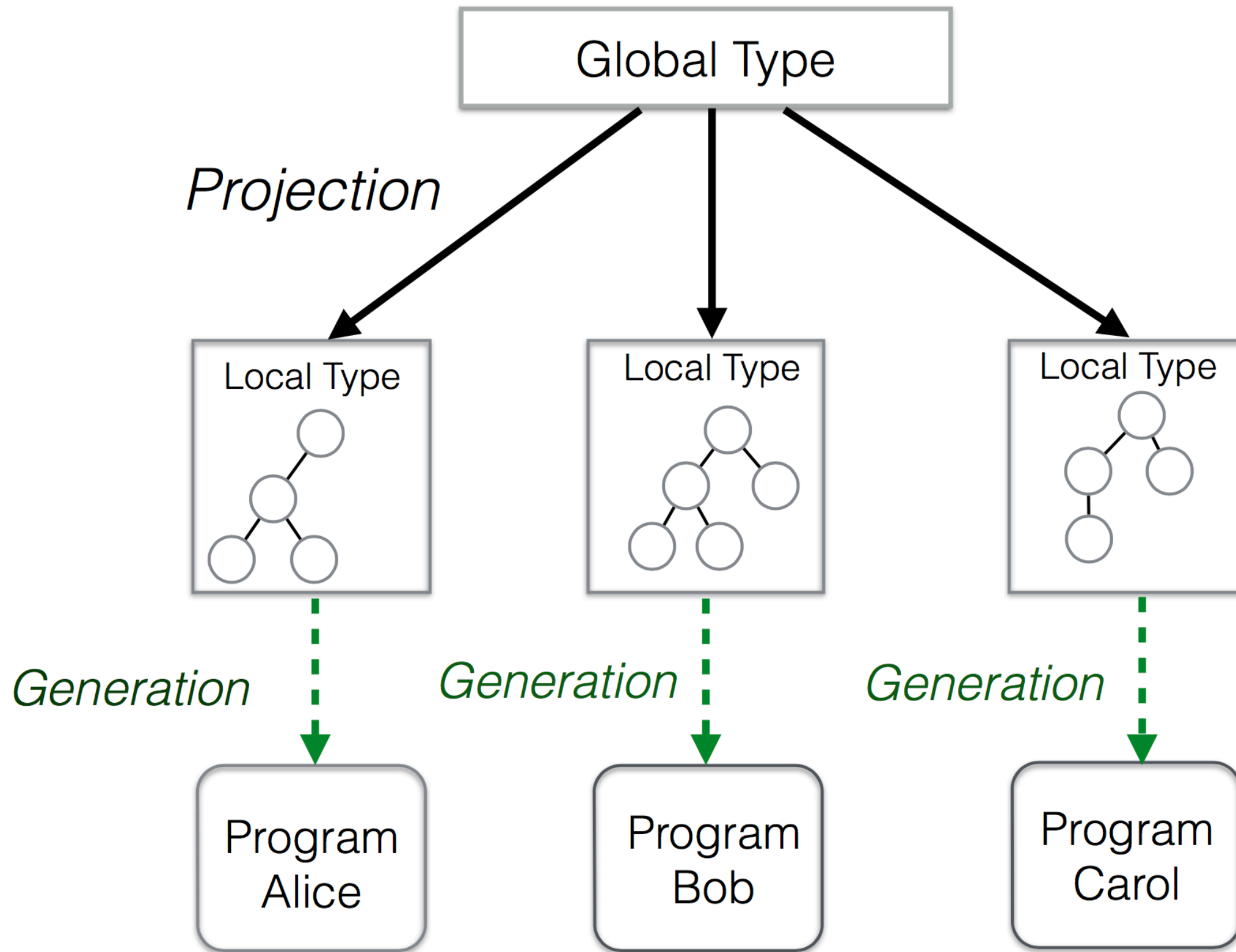
[OOPSLA'15, ECOOP'16, ECOOP'17, COORDINATION'17]





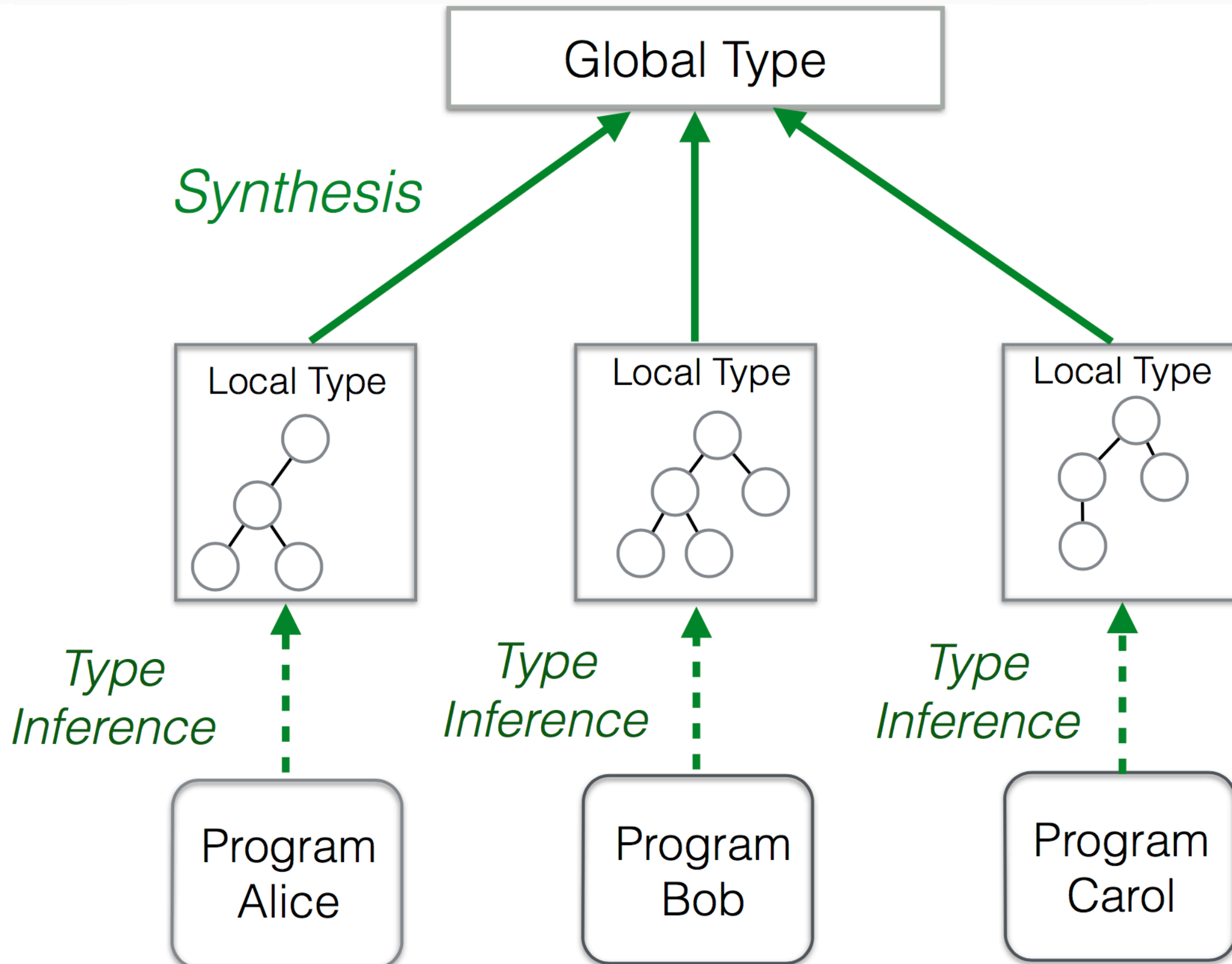
# Code Generation

[CC'15, FASE'16, FASE'17]



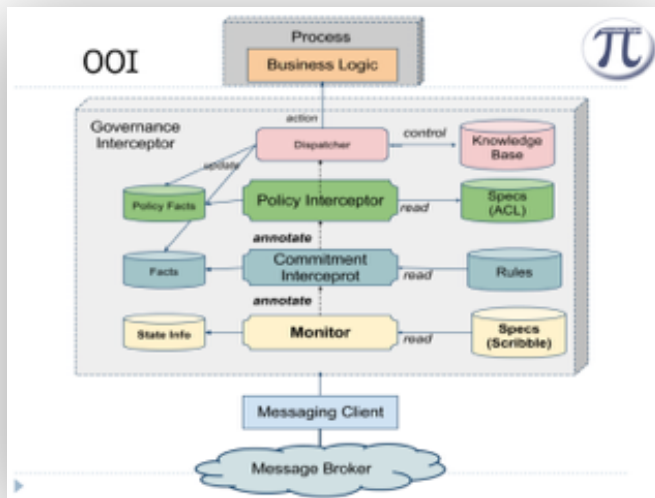
# Synthesis

[ICALP'13, POPL'15, CONCUR'15, TACAS'16, CC'16]

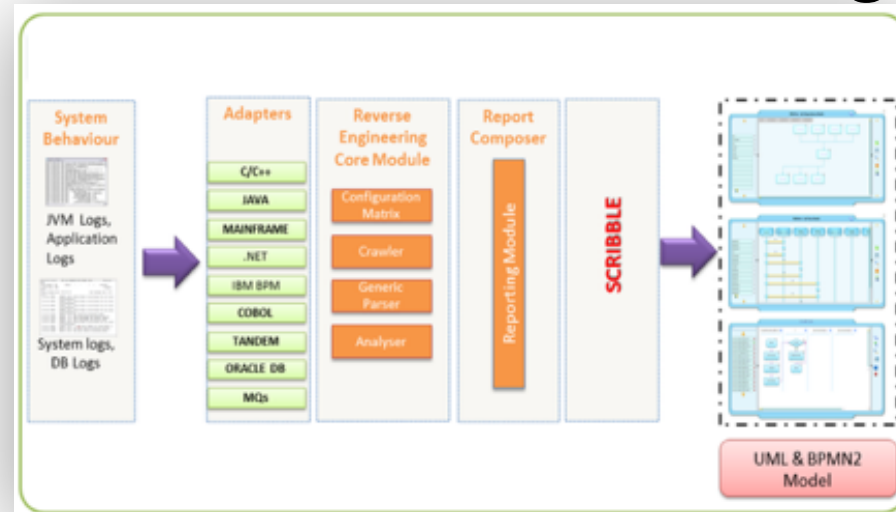


# Session Type based Tools

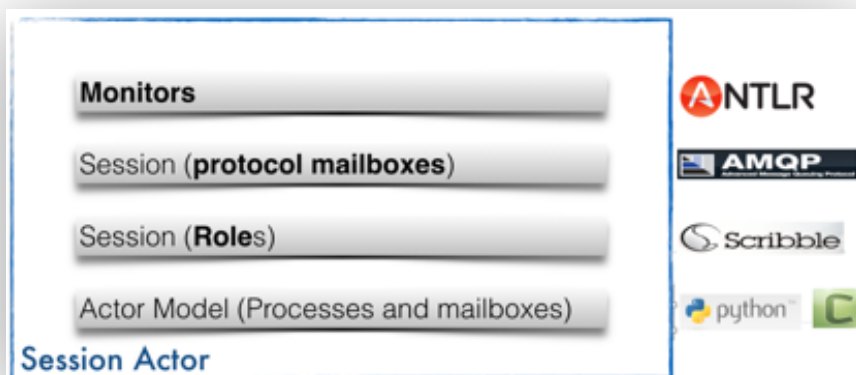
## OOI Governance



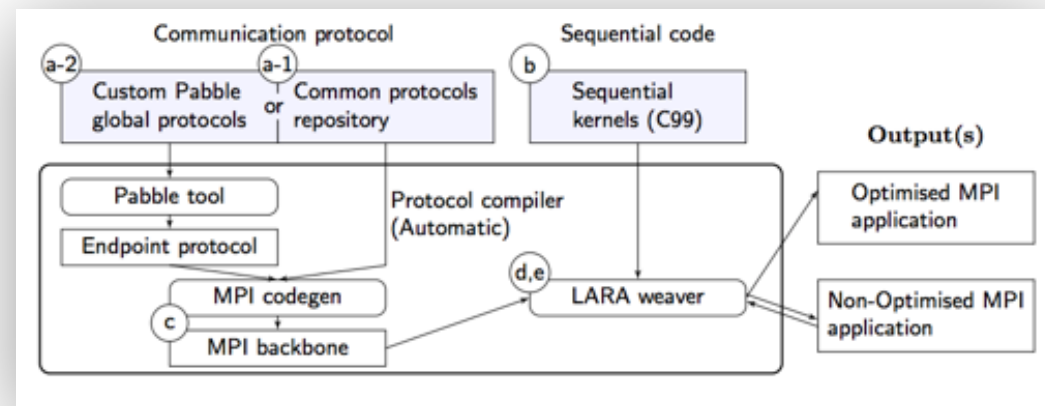
## ZDLC: Process Modeling



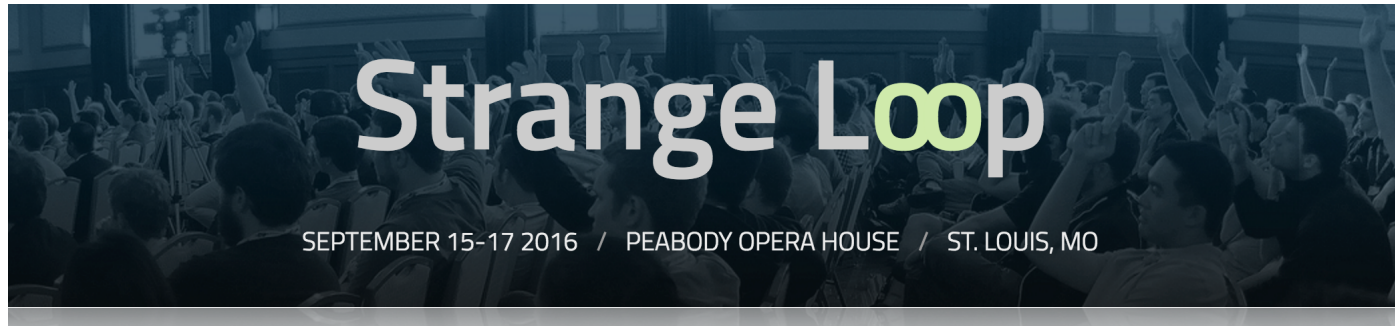
## Actor Verification



## MPI code generations



# Interactions with Industries



**Adam Bowen** @adamnbowen · Sep 15

I didn't even know that session types existed an hour ago, but thanks to Nobuko Yoshida's great talk at [#pwlconf](#), I want to learn more.



**Nobuko Yoshida**  
[Imperial College, London](#)

## DoC researcher to speak at Golang UK conference

by [Vicky Kapogianni](#)  
20 July 2016



**DoC researcher to speak at industry-focused Golang UK conference on results of concurrency research**

[Click here to add content](#)



[.@nicholascwng](#) rocking on [@GolangUKconf](#) about static deadlock detection in [#golang](#) [#gouk16](#)



# Interactions with Industries

## F#unctional Londoners Meetup Group

6 days ago · 6:30 PM

### Session Types with Fahd Abdeljallal



43 Members

Synopsis: Session types are a formalism to codify the structure of a communication, using types to specify the communication protocol used. This formalism provides the... [LEARN MORE](#)

## Distributed Systems vs. Compositionality

Dr. Roland Kuhn  
@rolandkuhn — CTO of Actyx

actyx

### Current State

- behaviors can be composed both sequentially and concurrently
- effects are not yet tracked
- Scribble generator for Scala not yet there
- theoretical work at Imperial College, London (Prof. Nobuko Yoshida & Alceste Scalas)



# Java API Generation [FASE'16]



RFC 821

August 1982  
Simple Mail Transfer Protocol

## TABLE OF CONTENTS

|               |                                           |           |
|---------------|-------------------------------------------|-----------|
| <u>1.</u>     | <u>INTRODUCTION</u>                       | <u>1</u>  |
| <u>2.</u>     | <u>THE SMTP MODEL</u>                     | <u>2</u>  |
| <u>3.</u>     | <u>THE SMTP PROCEDURE</u>                 | <u>14</u> |
| <u>3.1.</u>   | <u>Mail</u>                               | <u>14</u> |
| <u>3.2.</u>   | <u>Forwarding</u>                         | <u>14</u> |
| <u>3.3.</u>   | <u>Verifying and Expanding</u>            | <u>14</u> |
| <u>3.4.</u>   | <u>Sending and Mailing</u>                | <u>14</u> |
| <u>3.5.</u>   | <u>Opening and Closing</u>                | <u>14</u> |
| <u>3.6.</u>   | <u>Relaying</u>                           | <u>14</u> |
| <u>3.7.</u>   | <u>Domains</u>                            | <u>17</u> |
| <u>3.8.</u>   | <u>Changing Roles</u>                     | <u>18</u> |
| <u>4.</u>     | <u>THE SMTP SPECIFICATIONS</u>            | <u>19</u> |
| <u>4.1.</u>   | <u>SMTP Commands</u>                      | <u>19</u> |
| <u>4.1.1.</u> | <u>Command Semantics</u>                  | <u>19</u> |
| <u>4.1.2.</u> | <u>Command Syntax</u>                     | <u>27</u> |
| <u>4.2.</u>   | <u>SMTP Replies</u>                       | <u>24</u> |
| <u>4.2.1.</u> | <u>Reply Codes by Function Group</u>      | <u>25</u> |
| <u>4.2.2.</u> | <u>Reply Codes in Numeric Order</u>       | <u>26</u> |
| <u>4.3.</u>   | <u>Sequencing of Commands and Replies</u> | <u>37</u> |
| <u>4.4.</u>   | <u>State Diagrams</u>                     | <u>39</u> |
| <u>4.5.</u>   | <u>Details</u>                            | <u>41</u> |
| <u>4.5.1.</u> | <u>Minimum Implementation</u>             | <u>41</u> |
| <u>4.5.2.</u> | <u>Transparency</u>                       | <u>41</u> |
| <u>4.5.3.</u> | <u>Sizes</u>                              | <u>42</u> |

□

▼ channels

▼ C

▶ ioifaces

- EndSocket.java
- Smtp\_C\_1\_Future.java
- Smtp\_C\_1.java
- Smtp\_C\_10.java
- Smtp\_C\_11\_Cases.java
- Smtp\_C\_11\_Handler.java
- Smtp\_C\_11.java
- Smtp\_C\_12.java

```
.send(Smtp.S, new DataLine("Session  
.send(Smtp.S, new EndOfData())  
.receive(Smtp.S, Smtp._250, new Buf  
.S
```

- send(S role, Mail m) : Smtp\_C\_11 - Smtp\_C\_10
- send(S role, Quit m) : EndSocket - Smtp\_C\_10



# Scribble – Proving a distributed design

