Towards a Realistic Formal Model for Distributed Systems

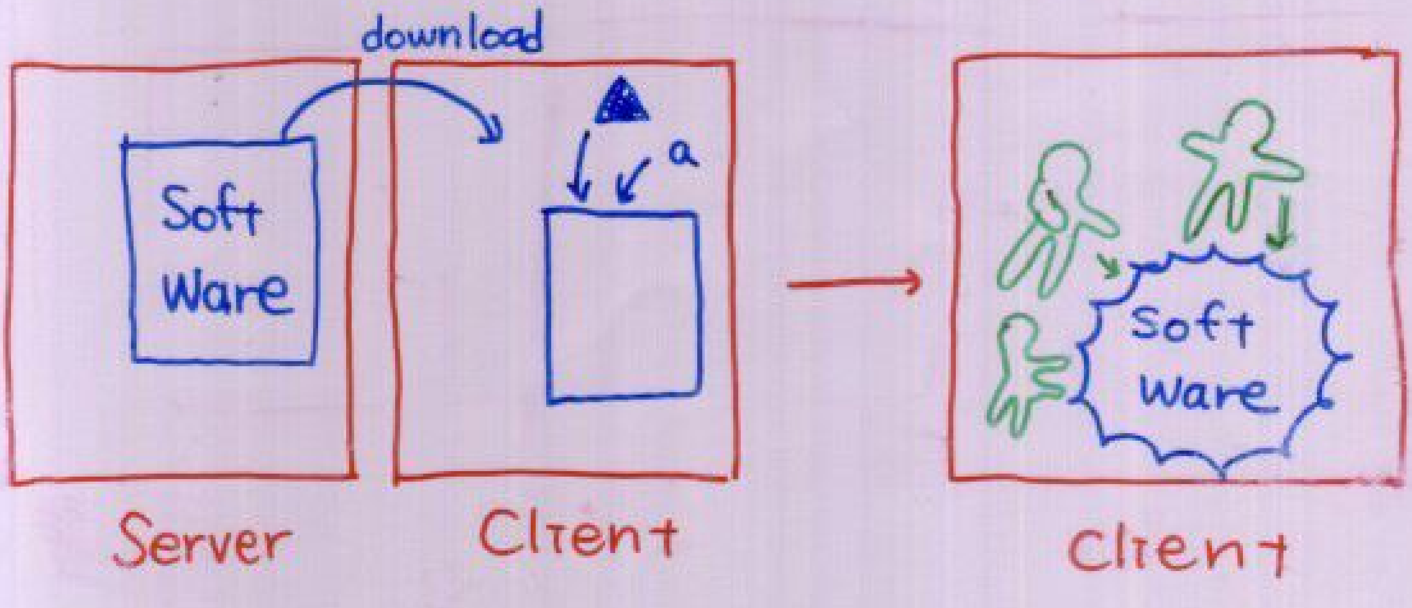
- π calculus [Milner, Parrow, Walker 89]

  Channels/names

  $$a(x).P \mid \bar{a}\langle \underline{b} \rangle \longrightarrow P[\underline{b}/x]$$

  Input Output

- Real World (cf. Applet Passing in Java)



Higher-Order π-Calculus

[Sangiorgi 93]

CML, Facile, LLinda, ...

λ_v $\quad (\lambda x.Q)V \to Q[V/x]$

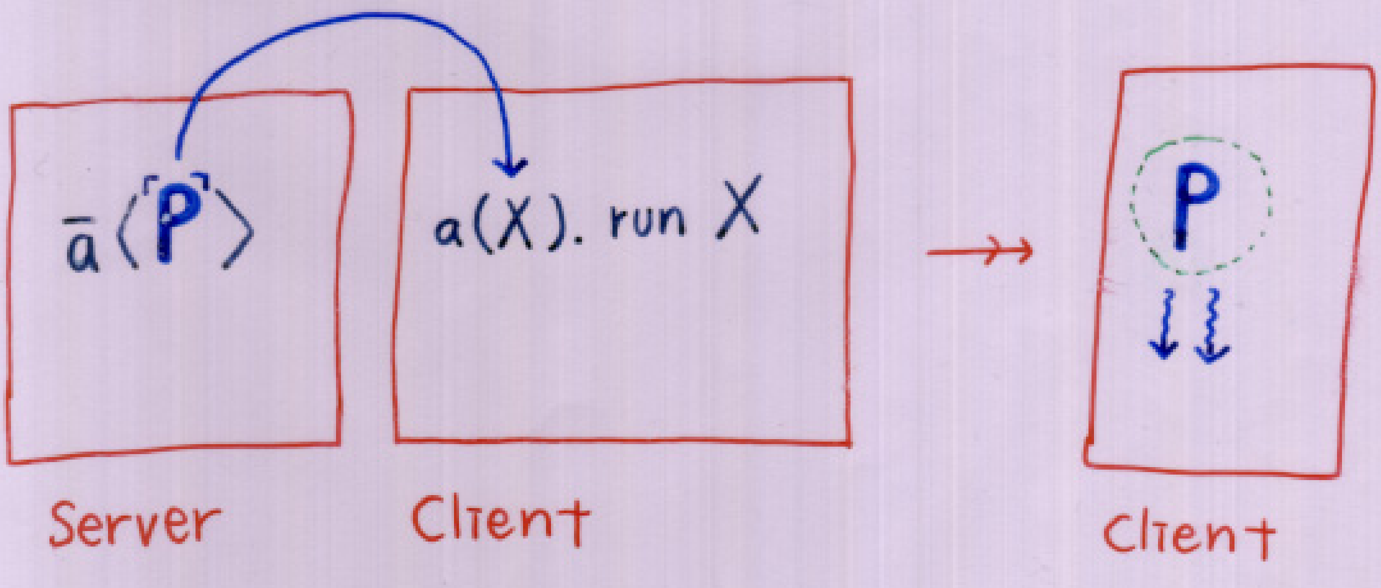
$\lambda\pi_v$ $\quad \bar{a}\langle \underbrace{\ulcorner P \urcorner}_{\text{code}} \rangle \mid a(X).\,\mathrm{run}\,X$

$\to \quad \mathrm{run}\,\ulcorner P \urcorner \to P$

where

$\underbrace{\ulcorner P \urcorner}_{\text{thunk}} = \lambda().\,P$

$\mathrm{run} = \lambda X.\,X()$



Aims of Types / Typechecking

- Using Types to control the effects of Mobile Code / Processes

- Host refuses to execute incoming code unless it conforms to predetermined access policy

$$a(x:\mathbf{T}).P$$

Type

Existing λ/π Typing System

[1993 – 2000]

Before [YH00]

$\lambda_{\to} + \pi_{IO}$

$\tau ::= \mathrm{unit} \mid \mathrm{nat} \mid \tau \to \rho \mid \sigma \mid \mathrm{proc}$

Value Term

constant process type

$\sigma ::= (\tilde{\tau})^{I} \mid (\tilde{\tau})^{O} \mid (\tilde{\tau})^{IO}$

channel — Input — Output — Input-Output

Typing Processes ···· Too Simple

$$\boxed{\Gamma \vdash P : \mathrm{proc}}$$

e.g. $$\frac{\Gamma \vdash P : \mathrm{proc} \qquad \Gamma \vdash Q : \mathrm{proc}}{\Gamma \vdash P \mid Q : \mathrm{proc}}$$

cf. $\Gamma \vdash M : \tau \to \rho$

Problem

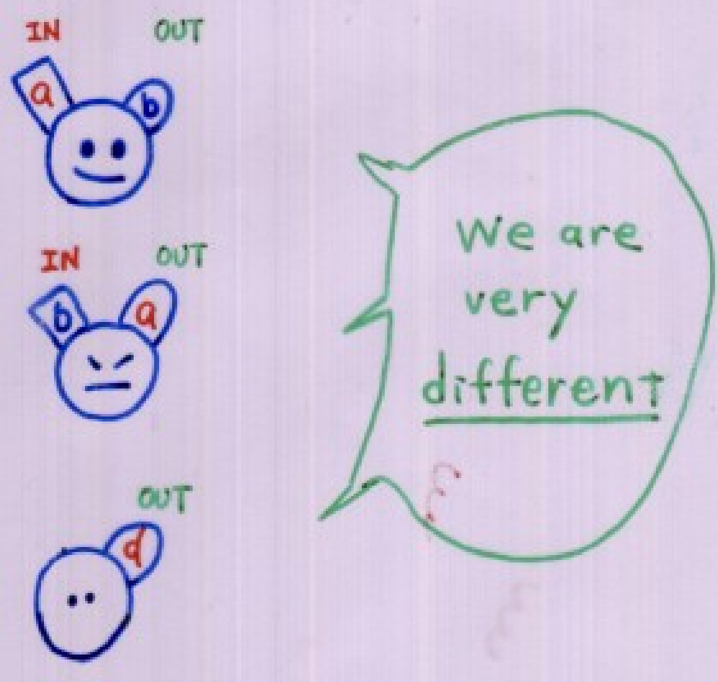
$$c(X : \ulcorner proc \urcorner).\ \mathrm{run}\ X$$

where $\ulcorner proc \urcorner = unit \rightarrow proc$

P $\underline{a}(X).\ \underline{\bar{b}}\langle X \rangle$

Q $\underline{b}(X).\ \underline{\bar{a}}\langle X \rangle$

R $\underline{d}\langle string \rangle$



I will give some technicality

Assigning Types to Processes

[Yoshida and Hennessy 2000]

$$\Gamma \vdash P : [\Delta]$$

channel environment
=
INTERFACE

Example

$P \quad \Gamma \vdash \underline{a}(x).\underline{\overline{b}}\langle x\rangle : [a:(\tau)^{I}, b:(\tau)^{O}]$

$Q \quad \Gamma \vdash \underline{b}(x).\underline{\overline{a}}\langle x\rangle : [a:(\tau)^{O}, b:(\tau)^{I}]$

$R \quad \Gamma \vdash d\langle string\rangle : [d:(string)^{O}]$

$$c(X : \ulcorner a:(\tau)^{I}, b:(\tau)^{O} \urcorner).\ \mathrm{run}\ X$$

P ○

Q ×

R ×

0 ○

History of Types in HOπ/π

HOπ

π

89 π-Calculus

92 Polyadic π

IO-Subtyping

93 HOπ [Sangiorgi PhD]

93 Linear Typings

Polymorphism

Causality-based Typings

98 Fine-Grained Types [Yoshida and Hennessy]

00 Other Mobility Types
- Seal Calculus
- M- Calculus
- Safe-Boxed Ambient

01 Fully Abstraction
- PCF [TCAL 01]
- $\lambda_{\to} + \times$ [LICS 01]
- System F [FoSSacs 01]
- Control [CW04]

02 Secure Info Flow [ESOP 01, FoSSaCs 02 POPL 02]

03 Channel Dependency Existential Types

⇒ Integration with Linearity ⟹ Application: Secure Info, Role-Based Access Control the Idea is simple but ...

quite a bit of work ....

[1998 - 2000 - 2003]

Channels appear both in Types and Processes

$$F = \lambda x.\, \lambda(X : \ulcorner x:(\tau)^{o}, b:(\tau)^{o} \urcorner).\,(\mathrm{run}\, X \mid \overline{x}\langle 1\rangle \mid \overline{b}\langle 2\rangle)$$

$$F a \rightarrow \lambda(X : \ulcorner \underline{a}:(\tau)^{o}, b:(\tau)^{o} \urcorner).\,(\mathrm{run}\, X \mid \overline{\underline{a}}\langle 1\rangle \mid \overline{b}\langle 2\rangle)$$

$$F b \rightarrow \lambda(X : \ulcorner \underline{b}:(\tau)^{o} \urcorner).\,(\mathrm{run}\, X \mid \overline{\underline{b}}\langle 1\rangle \mid \overline{b}\langle 2\rangle)$$

$\Rightarrow$ Kinding / Dependency Types

[Yoshida · Hennessy 2000] • Functional Channel Dependency

POPL 04 New • Channel Dependency (Non-determinism)

POPL 04 New • Existential Types (Scope-Opening)

Higher Order π-calculus

$\lambda\pi_v$

Syntax

$$
\begin{aligned}
P, Q ::=\ & V && \text{value} \\
\mid\ & 0 && \text{nil} \\
\mid\ & P|Q && \text{parallel} \\
\mid\ & \bar{u}\langle V_1, \ldots, V_n \rangle && \text{OUTPUT} \\
\mid\ & u(X_1{:}\tau_1, \ldots, X{:}\tau_n)P && \text{INPUT} \\
\mid\ & !P && \text{Replication} \\
\mid\ & (\nu a{:}\sigma)P && \text{Restriction} \\
\mid\ & PQ && \text{Application} \\
V, W ::=\ & \lambda(x{:}\tau)P && \lambda\text{-abst} \\
\mid\ & 1, 2, \ldots, (), \ldots && \text{constant} \\
\mid\ & x, y, z, \ldots && \text{variables} \\
\mid\ & a, b, c, \ldots && \text{channels/names}
\end{aligned}
$$

Types

Term $\tau ::= \mathrm{unit}\ ,\ \mathrm{nat}$

$\quad\mid\ \tau \to \tau'$

$\quad\mid\ \Pi(x:\sigma)\tau$ functional dependency

$\quad\mid\ [\Delta]$ process types

$\quad\mid\ \sigma$

Channel $\sigma ::= (\Pi[\tilde{x}:\tilde{\sigma}]\tilde{\tau})^{p}$ dependency

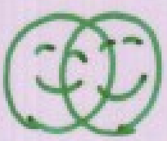
$\quad\mid\ (\exists[\tilde{x}:\tilde{\sigma}]\tilde{\tau})^{p}$ existential

$\quad\mid\ \langle \sigma_I, \sigma_O \rangle$

Typing System $\Gamma \vdash P \triangleright I$

(Zero)
$$\frac{\Gamma \vdash Env}{\Gamma \vdash 0 \triangleright [\,]}$$

(Par)
$$\frac{\Gamma \vdash P \triangleright [\Delta] \quad \Gamma \vdash Q \triangleright [\Delta']}{\Gamma \vdash P \mid Q \triangleright [\Delta \cup \Delta']}$$



(Res)
$$\frac{\Gamma, a:\sigma \vdash P \triangleright [\Delta, a:\sigma]}{\Gamma \vdash (\nu a:\sigma)P \triangleright [\Delta]}$$

(Rep)
$$\frac{\Gamma \vdash P \triangleright [\Delta]}{\Gamma \vdash !P \triangleright [\Delta]}$$

Existential Types for Scope Opening

Client (A) wishes to execute code P and to get ack $\bar{c}\langle V \rangle$ at the time P is executed at the remote location **(B)**

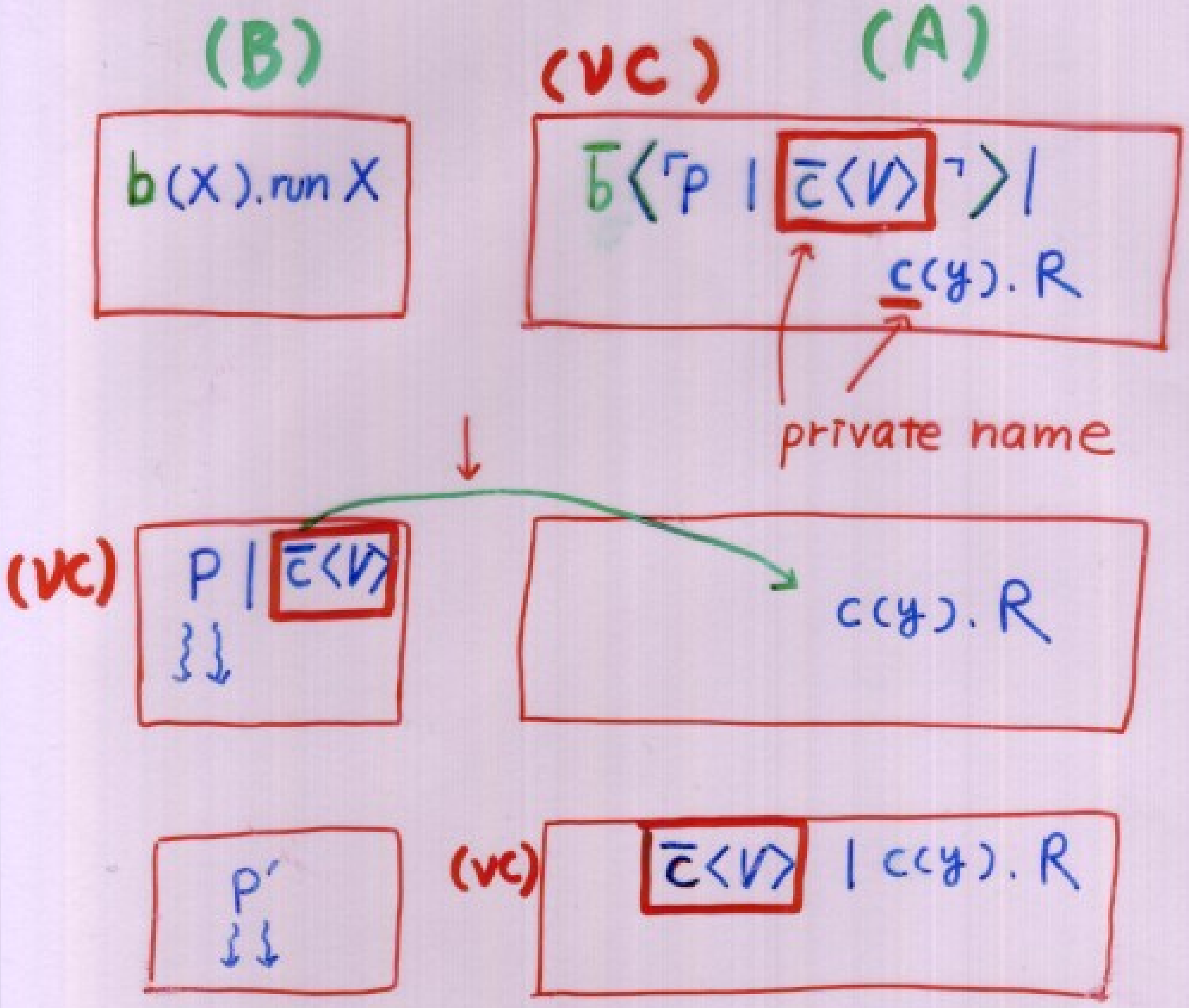
Location (A)

$$(\nu c)\quad \bar{b}\langle \ulcorner \underbrace{P}_{\text{code}} \mid \underbrace{\bar{c}\langle V\rangle}_{\text{ack}} \urcorner \rangle \mid \underbrace{c(x).R}_{\text{wait at } c}$$

(1) c is private

(2) V must not be touched (i.e. compromised)

Existential Types for Scope Opening



previous system

$$(\ulcorner \mathrm{proc} \urcorner)^{I}$$

[POPL 04]
channel existential types

$$(\exists[x:\sigma]\ulcorner \Delta, x:\sigma \urcorner)^{I}$$

anonymous channel of type σ

Typing System for $\exists$

$$(\mathrm{In}^{\exists})\quad \frac{\Gamma \vdash \underline{a} : (\exists[x:\sigma]\tau)^{I} \qquad \Gamma, \overbrace{\{x:\sigma, X:\tau\}}^{\text{pack}} \vdash P \triangleright [\Delta, x:\sigma]}{\Gamma \vdash a(X : \exists[x:\sigma]\tau).\, P \triangleright [\Delta, \underline{a} : (\exists[x:\sigma]\tau)^{I}]}$$

$$(\mathrm{Out}^{\exists})\quad \frac{\Gamma \vdash a : (\exists[x:\bar{\sigma}]\tau)^{O} \qquad \Gamma \vdash \overbrace{\{c, V\}}^{\text{pack}} : \exists[x:\sigma]\tau}{\Gamma \vdash \bar{a}\langle V\rangle \triangleright [a : (\exists[x:\bar{\sigma}]\tau)^{O}, \underline{c:\sigma}]}$$

record a name
to be restricted

Proposition (Minimality)

$$\Gamma \vdash P \triangleright [\Delta] \Rightarrow \exists!\, \Delta' \subseteq \Delta$$
$$\text{s.t. } \Gamma \vdash P \triangleright [\Delta']$$

Main Theorems

Subject Reduction

$$\Gamma \vdash P : \tau, \quad P \twoheadrightarrow P' \Rightarrow \Gamma \vdash P' : \tau$$

Type Safety

$$\Gamma \vdash P : [\Delta] \Rightarrow \quad P \not\xrightarrow{\Gamma, [\Delta]}_{err}$$

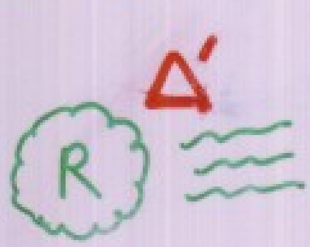
where $P \not\xrightarrow{\Gamma, [\Delta]}_{err}$ means

P can use **at most** resources in Δ

Consequence:

$$a(X : \ulcorner \Delta \urcorner).P \mid \overline{a}\langle \ulcorner R \urcorner \rangle \xrightarrow{\Gamma, \pi}_{err}$$

if $\Gamma \nvdash R : [\Delta]$



Encapsulation of Higher-Order Code by Hidden Name

Theorem

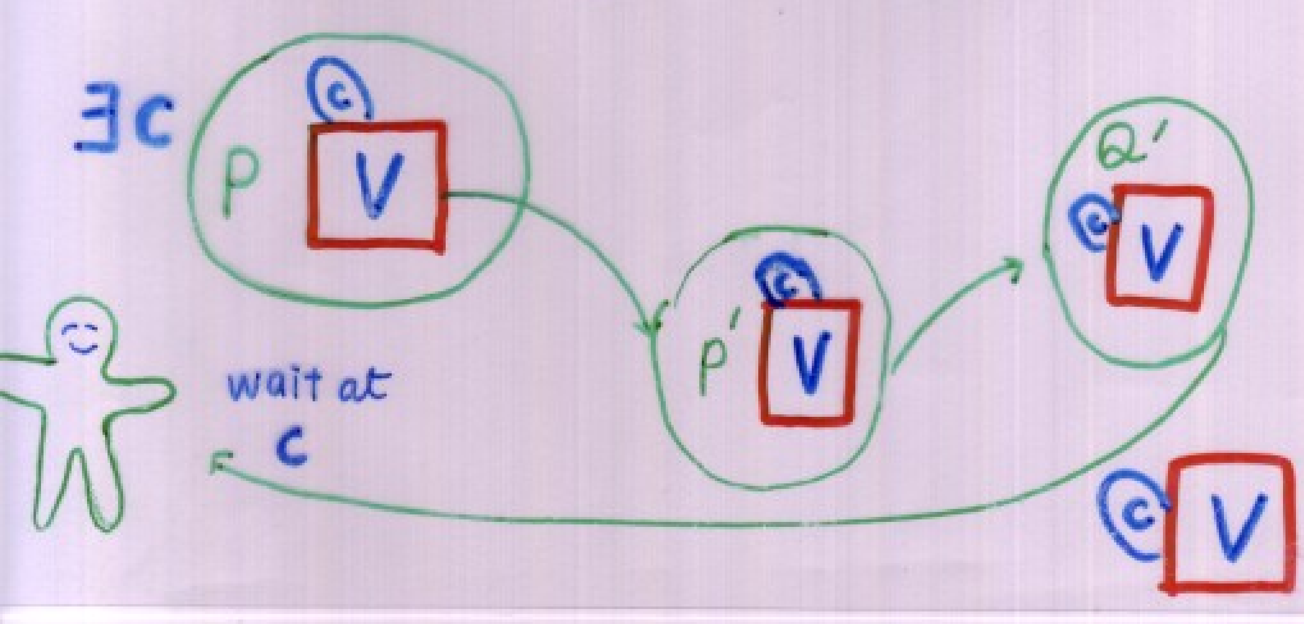
$\Gamma \vdash P : [\Delta]$ and $fv(P) = \emptyset$

and $\Gamma \vdash a : (\exists(x{:}\sigma^{\uparrow_L}) \ulcorner x{:}\sigma^{\uparrow} \urcorner)^{!}$

linear name

Then $P \xrightarrow{a\langle \ulcorner \bar{c}\langle V \rangle \urcorner \rangle} P' \Rightarrow P' \twoheadrightarrow \xrightarrow{\bar{c}\langle V \rangle}$

Mobile Code bound by $\exists$-name is eventually returned to the sender without being touched by the receiver



Secure Information Flow in HOπ

why Fine Grained Process Types?

$a^{L}.\overline{b}^{H}$ ○

$b^{H}.\overline{a}^{L}$ ×

$\overline{b}^{H}\langle \ulcorner \overline{a}^{L} \urcorner \rangle$??

a thunk of a L level program is transfered via H level channel

$\overline{b}^{H}$ L

$b^{H}(x).\,\mathrm{run}\ x \mid a^{L}.\overline{c}^{L}$ ×

$[\, a : ()^{0}_{L} \,]$

$b^{H}(x).(\lambda y.0)\,x \mid a^{L}.\overline{c}^{L}$ ○

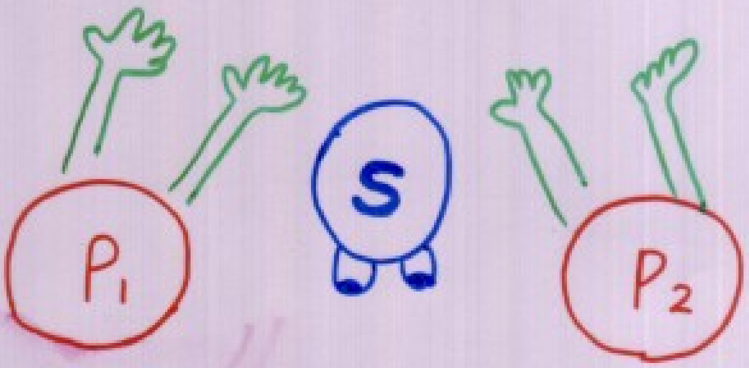
$[\ ]$

Theorem Non-Interference

$\Gamma \vdash P_{1,2} \triangleright [\Delta]$ s.t. $\mathrm{tamp}(\Delta) \not\sqsubseteq s$

Then $P_1 \approx_s P_2$

Two Processes with a secrecy level incompatible with <u>s</u> can be equated by $\approx_s$



Proof : By Type Preserving Translations into πsec [Honda and Yoshida 2002]

Role-Based Access Control

Why Fine-Grained Process Types?

Extensible Architecture SPIN / JVM / CLR

[1] Safe Access to Resources by Name-Space Control ⇒ Process Types

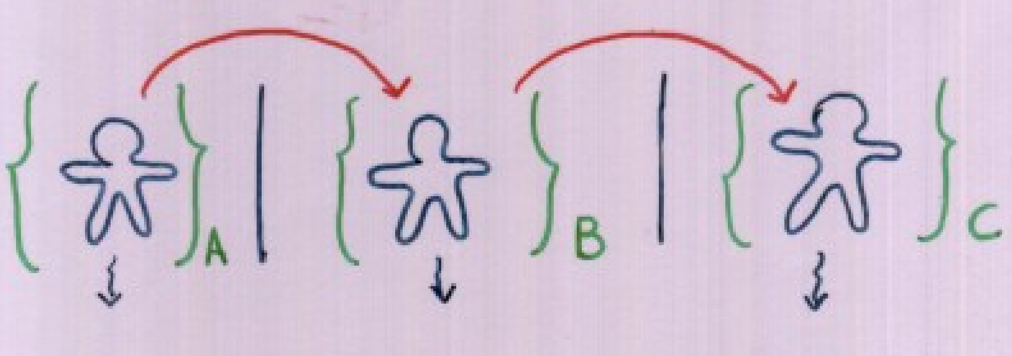
[2] Dynamic / Automatic Update of Access Rights when Crossing Protection Domain

⇒ Linear Typing

[3] Primitives to Alter one's own Access Right

$$
\begin{aligned}
P ::= \;\cdots\; & \mid \; \mathrm{set}(\mathrm{principal})\{P\} \\
& \mid \; !\,a^{B}(\tilde{x} : \tilde{\tau}).\,P \\
& \mid \; \{P\}_{\mathrm{principal}}
\end{aligned}
$$

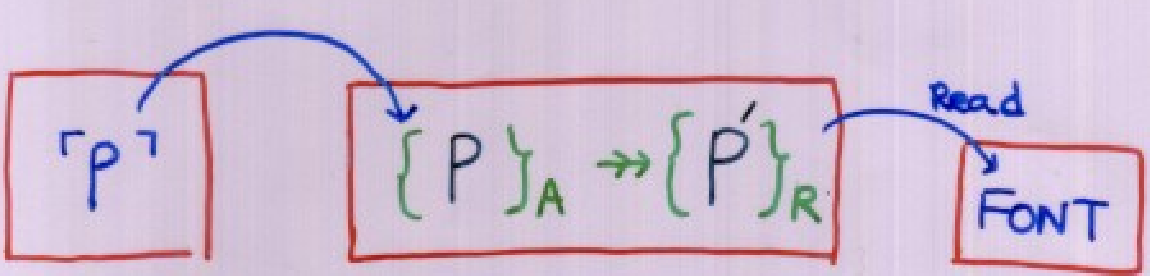
← thread running as principal



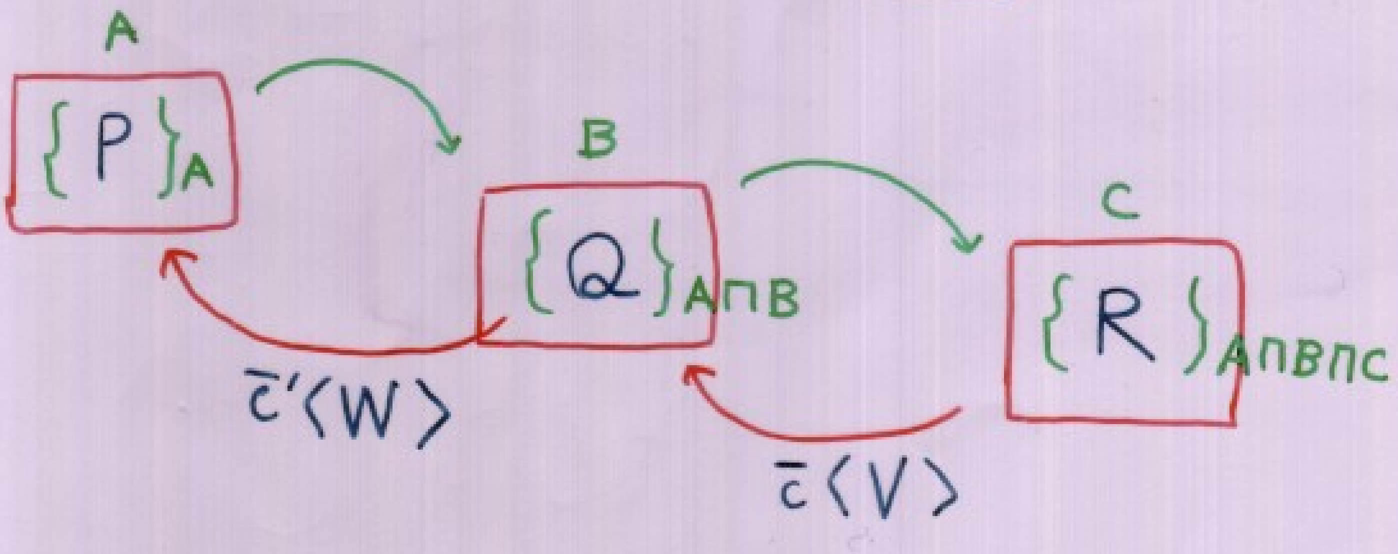
Role-Based Access Control

Reduction

$$\{ \mathrm{set}(A)\{P\} \}_B \longrightarrow \{P\}_A$$



$$\{\bar{a}\langle V \rangle\}_A \mid \{!a(x).R\}_B$$
$$\longrightarrow \{R[V/x]\}_{\underline{A \sqcap B}} \mid \{!a(x).R\}_B$$



cf. Blacketing Condition via Linear / Affine

Theorem Role-Based Resource Error Free

$$\Gamma \vdash P \triangleright [\Delta] \text{ then } P \not\xrightarrow{Aerr}$$

where

$$\left\{ !a^{C}(X).\,R \right\}_A \,\Big|\, \left\{ \bar{a}\langle V\rangle \right\}_B \xrightarrow{Aerr}$$

if $C \not\leq B$

Typable Processes do not violate privilege

- Stack Inspection Mechanism (JMV, SPIN, CLR)
- History-Based Mechanism [AF02]

Conclusion

- A New Expressive Theory of Types for the Higher-Order Code Mobility
- Applications to Secure Information Flow and Role-Based Access Control

Future Work

- Developing Secure Typing Systems for Programming Languages

  JFlow Multi-threaded Dynamic Class / Code Loading (Serialization)

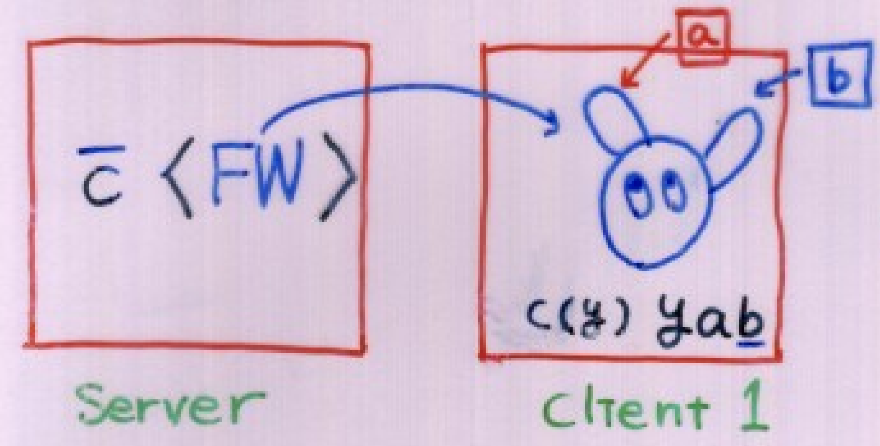
Reference

- Full Version www.doc.ic.ac.uk/~yoshida
- Safe Dpi with Hennessy / Rathke [FoSSaCS04]

Script Server

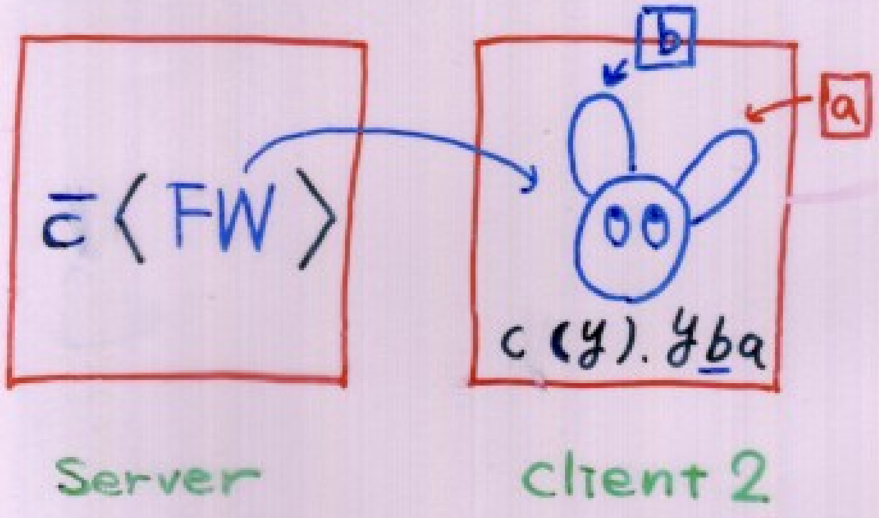
$$FW = \lambda(x:\tau).\lambda(y:\tau).\ x(z).\overline{y}\langle z\rangle$$

forwarder



C1 evolves into

$$FW(ab) : [a:(\tau)^I,\ b:(\tau)^O]$$



C2 evolves into

$$FW(ba) : [a:(\tau)^O,\ b:(\tau)^I]$$

$\boxed{\text{Previous}}$ $(\tau)^I \to (\tau)^O \to \text{proc}$

Func Dep [YH00] $\Pi(\underline{x}:(\tau)^I)\,\Pi(\underline{y}:(\tau)^O)\,[x:(\tau)^I,\ y:(\tau)^O]$