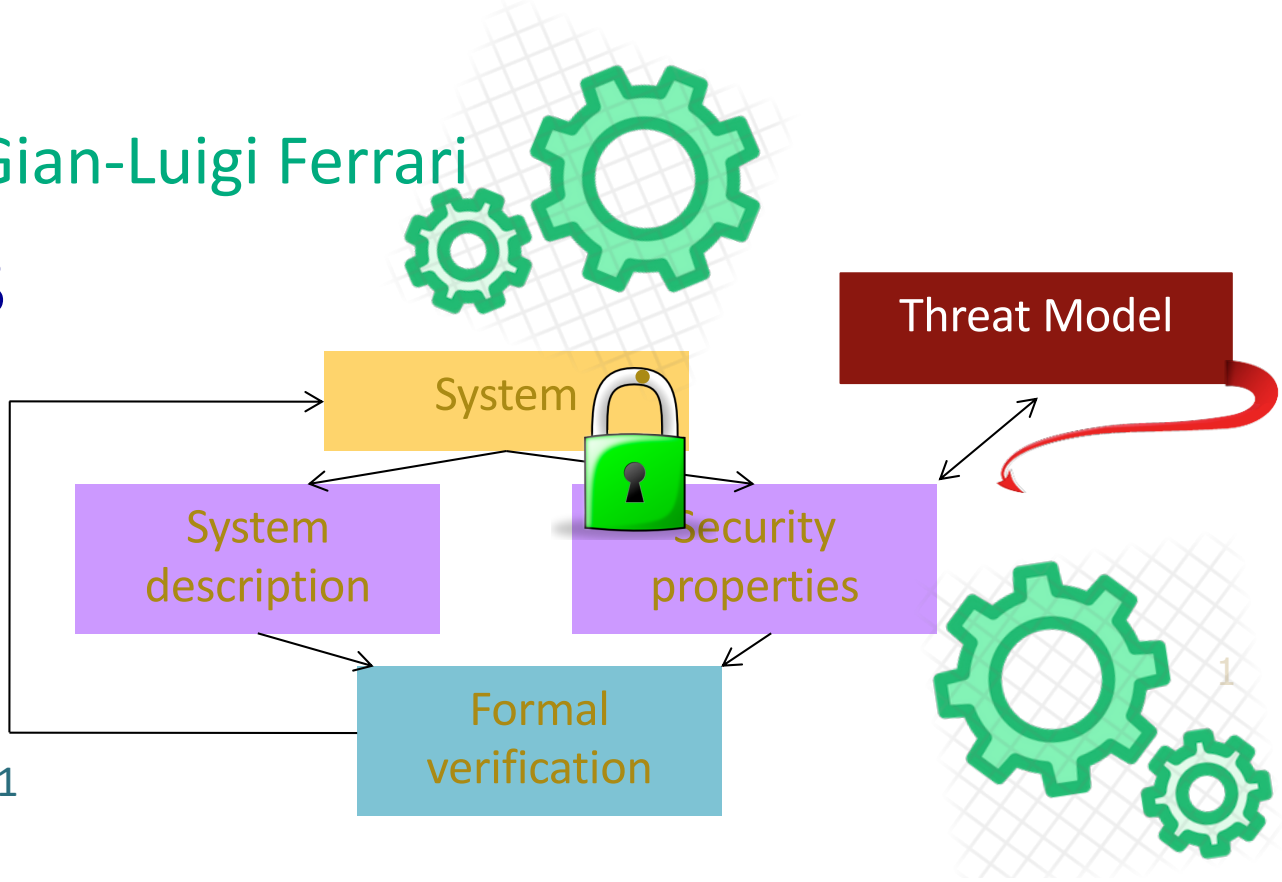
# LANGUAGE-BASED TECHNOLOGY FOR SECURITY

Chiara Bodei, Gian-Luigi Ferrari

## PI CALCULUS




Maggio 2021

# Outline

- **Motivation**
- Key notions
- CCS
- From CCS to Pi calculus
- Pi calculus: syntax and semantics
- The role of restriction for security
- Modeling protocols in Pi calculus
- Spi Calculus: a hint
- Conclusions and what next

# Motivation

π-calculus : concurrent languages = lambda calculus : sequential languages
Process calculi treat processes much like λ-calculus treats computable functions

- **λ-calculus*** is the core language of functional computation, in which "everything is a function" and all computation proceeds by function application
- **π-calculus** (1989) is the core formal calculus of message-based concurrency, in which "everything is a process" and all computation proceeds by communication on channel

interaction

*λ-calculus can be encoded in π-calculus

# Motivation

System behaviour tends to be composed of several processes that are executed concurrently, where these processes exchange data in order to influence each other's behaviour.

- Non determinism
- Parallelism
- Interaction
- Infinite runs

Milner's insight was that concurrent processes have an algebraic structure

# Outline

- Motivation
- **Key notions**
- CCS
- From CCS to Pi calculus
- Pi calculus: syntax and semantics
- The role of restriction for security
- Modeling protocols in Pi calculus
- Spi Calculus: a hint
- Conclusions and what next

# Key notion

**Mobility** is the key notion of Pi calculus. It can refer to:

1. **processes move** in the physical space of computing sites;
2. **processes move** in the virtual space of linked processes;
3. **links move** in the virtual space of linked processes.

- The Pi calculus models the changing connectivity of interactive systems (point 3) [link passing mobility]
- The Pi calculus evolved from CCS

Concurrent processes have an algebraic structure

# Key notion

**Mobility** is the key notion of Pi calculus. It can refer to:

1. **processes move** in the physical space of computing sites;
2. **processes move** in the virtual space of linked processes;
3. **links move** in the virtual space of linked processes.

- The Pi calculus models the changing connectivity of interactive systems (point 3) [link passing mobility]
- The Pi calculus evolved from CCS

Concurrent processes have an algebraic structure

# Process calculi

Milner's general model:

- A concurrent system is a collection of processes
- A process is an independent agent that may perform internal activities in isolation or may interact with the environment to perform shared activities

The insight was that concurrent processes have an algebraic structure

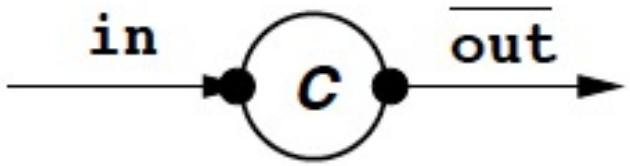
As a consequence, process calculi are also called process algebras

# Outline

- Motivation
- Key notion
- **CCS**
- From CCS to Pi calculus
- Pi calculus: syntax and semantics
- The role of restriction for security
- Modeling protocols in Pi calculus
- Spi Calculus: a hint
- Conclusions and what next

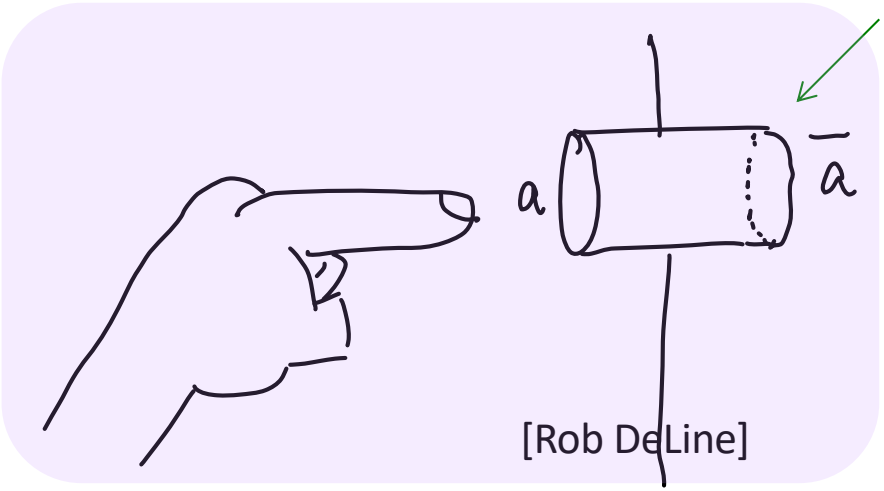
# Calculus of Communicating Systems (CCS)

- A CCS process is a computing agent that may communicate with its environment via its interface [it is an **abstraction** of an independent **thread** of control]
- Interface = Collection of communication ports/channels, together with an indication of whether they are used for input or output.



# CCS: actions and co-actions

- Actions a and $\bar{a}$ are viewed as being **complementary**

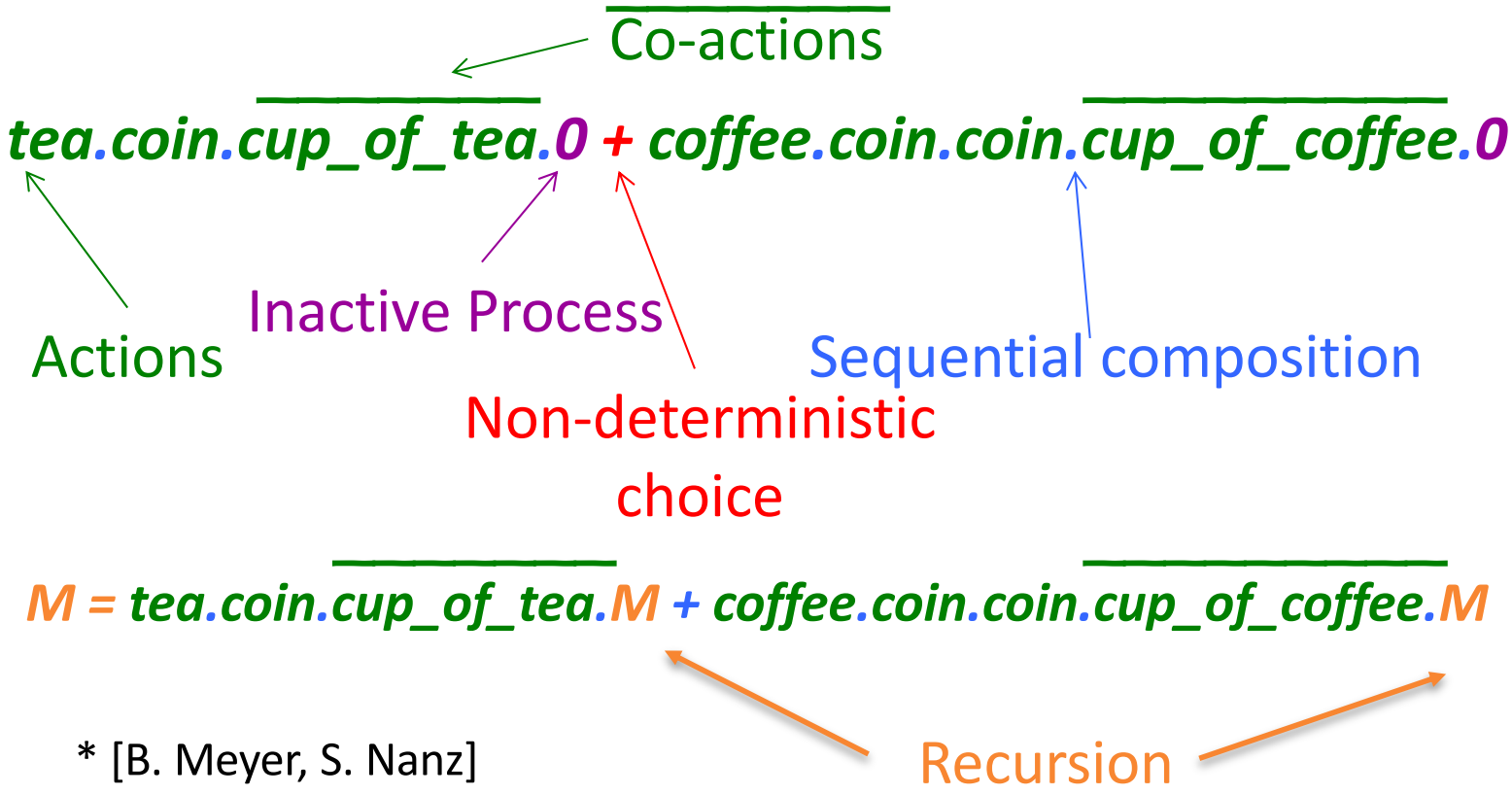



Pushing the button is one action seen from two perspectives

A channel is an abstraction of the communication link between two processes

# CCS: example*

- Coffee Machine: takes an order for tea or coffee, accepts the appropriate payment, pours the ordered drink, and terminates




* [B. Meyer, S. Nanz]

# Calculus of Communicating Systems (CCS)

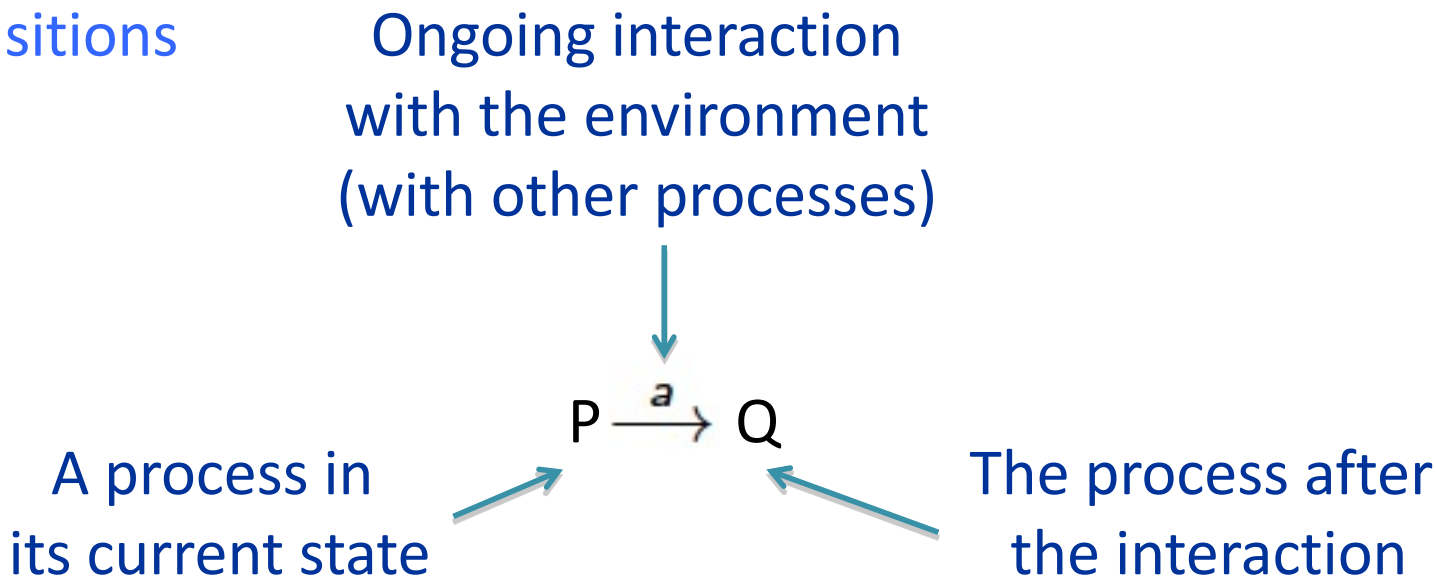
## Basic Entities

- A set **N** = a, b, . . . of names and $\bar{\mathbf{N}}$ = {$\bar{a}$ | a in N} of co-names
- A set **L** = N U $\bar{N}$ of labels (ranged over by l , l', . . .)
- A set **Act** = L U {τ} of actions (ranged over by a, b, . . .)
- Action **τ** is called the silent or unobservable action

# Calculus of Communicating Systems (CCS)

To formally describe the behaviour of CCS processes, we use **Labelled Transition Systems** (LTS) and resort to **Structural Operational Semantics** (SOS)

Transitions




- The transition describes one possible next step of the execution of P

# CCS: syntax

$$
\begin{array}{rl}
P := & K \\
 & \alpha.P \\
 & \sum_{i \in I} P_i \\
 & P_1 | P_2 \\
 & P \setminus L \\
 & P[f]
\end{array}
$$

## Sequential fragment

- Nil (or **0**) process (the atomic process)
- action prexing: **a.P**
- names and recursive definitions:
- non deterministic choice: **+**

## Parallelism and renaming

- parallel composition: **|** (synchronous communication *btw* two components = handshake synchronization)
- Restriction: **P \ L**
- Relabelling: **P[f]**

$$P_1 + P_2 = \sum_{i \in \{1,2\}} P_i$$

$$Nil = 0 = \sum_{i \in \emptyset} P_i$$

# CCS: example (cont.)

- When a process executes it performs some action and becomes a new process. The execution of an action a is represented by a **transition** $\xrightarrow{a}$ with label a

$$\overline{tea}.coin.\overline{cup\_of\_tea}.0 + coffee.coin.coin.\overline{cup\_of\_coffee}.0$$

$$\xrightarrow{tea}$$


$$\overline{tea}.coin.\overline{cup\_of\_tea}.0 + coffee.coin.coin.\overline{cup\_of\_coffee}.0$$

# CCS: example (cont.)

- When a process executes it performs some action and becomes a new process. The execution of an action a is represented by a **transition** $\xrightarrow{a}$ with label a

$\overline{tea}.coin.\overline{cup\_of\_tea}.0 + coffee.coin.coin.\overline{cup\_of\_coffee}.0$

$\xrightarrow{tea}$ Tea

$tea.coin.\overline{cup\_of\_tea}.0 + coffee.coin.coin.\overline{cup\_of\_coffee}.0$

$\xrightarrow{coin}$ 

$tea.coin.\overline{cup\_of\_tea}.0 + coffee.coin.coin.\overline{cup\_of\_coffee}.0$

# CCS: example (cont.)

- When a process executes it performs some action and becomes a new process. The execution of an action a is represented by a **transition** $\xrightarrow{a}$ with label a

$\underline{\quad\quad\quad\quad}$ $\underline{\quad\quad\quad\quad\quad\quad}$

***tea*.coin.cup_of_tea.0 + coffee.coin.coin.cup_of_coffee.0**

tea


$\underline{\quad\quad\quad\quad}$ $\underline{\quad\quad\quad\quad\quad\quad}$

***tea*.coin.cup_of_tea.0** + coffee.coin.coin.cup_of_coffee.0

coin

$\underline{\quad\quad\quad\quad}$ $\underline{\quad\quad\quad\quad\quad\quad}$

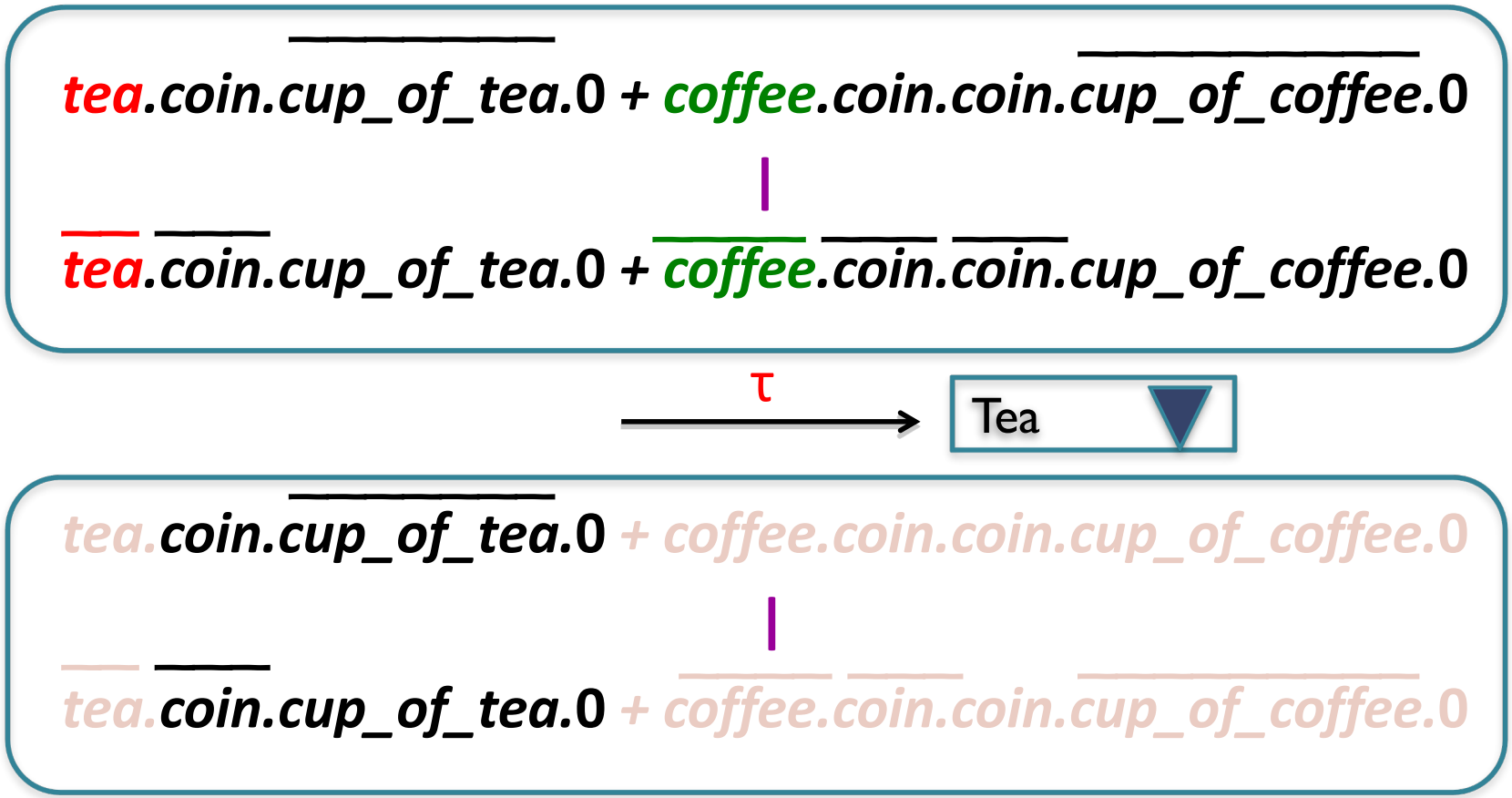
tea.coin.***cup_of_tea*.0** + coffee.coin.coin.cup_of_coffee.0

cup_of_tea

$\underline{\quad\quad\quad\quad}$ $\underline{\quad\quad\quad\quad\quad\quad}$

tea.coin.cup_of_tea.**0** + coffee.coin.coin.cup_of_coffee.0

# CCS: example in parallel composition



# CCS: example (cont.)

$$\overline{tea}.coin.cup\_of\_tea.0 + coffee.coin.coin.\overline{cup\_of\_coffee}.0 \mid \overline{tea}.\overline{coin}.cup\_of\_tea.0 + \overline{coffee}.coin.coin.\overline{cup\_of\_coffee}.0$$

$\tau$

$$tea.coin.\overline{cup\_of\_tea}.0 + coffee.coin.coin.\overline{cup\_of\_coffee}.0 \mid \overline{tea}.\overline{coin}.cup\_of\_tea.0 + \overline{coffee}.coin.coin.\overline{cup\_of\_coffee}.0$$

# CCS: example (cont.)

$$\overline{tea}.coin.\mathbf{cup\_of\_tea}.0 + coffee.coin.coin.\overline{cup\_of\_coffee}.0$$

$$\mid$$

$$\overline{tea}.coin.\mathbf{cup\_of\_tea}.0 + \overline{coffee}.coin.coin.\overline{cup\_of\_coffee}.0$$

$\xrightarrow{\tau}$ 

$$\overline{tea}.coin.\overline{cup\_of\_tea}.\mathbf{0} + coffee.coin.coin.\overline{cup\_of\_coffee}.0$$

$$\mid$$

$$\overline{tea}.coin.\overline{cup\_of\_tea}.\mathbf{0} + \overline{coffee}.coin.coin.\overline{cup\_of\_coffee}.0$$

# CCS: example (cont.)

$$\overline{tea}.coin.\overline{cup\_of\_tea}.0 + coffee.coin.coin.\overline{cup\_of\_coffee}.0$$

$$\mid$$

$$\overline{tea}.\overline{coin}.\overline{cup\_of\_tea}.0 + \overline{coffee}.\overline{coin}.\overline{coin}.\overline{cup\_of\_coffee}.0$$

$$\xrightarrow{\tau}{}^{*}$$

$$0 \mid 0$$

* is the **Kleene star**: means with **0 or many** transitions

# Labelled Transition Systems (LTS)

**Definition**

A labelled transition system (LTS) is a triple $(Proc, Act, \{\xrightarrow{\alpha} \mid \alpha \in Act\})$ where

- $Proc$ is a set of processes (the states),
- $Act$ is a set of actions (the labels), and
- for every $\alpha \in Act$, $\xrightarrow{\alpha} \subseteq Proc \times Proc$ is a binary relation on processes called the transition relation

We can imagine a **graph**, where:

- **Nodes** are **processes**
- **Edges** are **transitions**
- The **start state** is the **initial system**

# Labelled Transition Systems (LTS)

- Terminal process: $0$

  behavior: $0 \not\longrightarrow$

- Action prefixing: $\alpha.P$

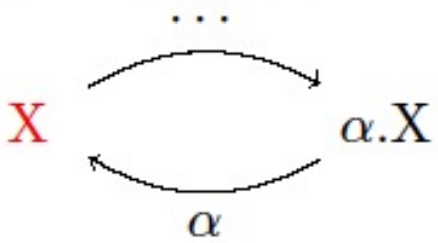
  behavior: $\alpha.P \xrightarrow{\alpha} P$

- Non-deterministic choice: $\alpha.P + \beta.Q$

  behavior: $P \xleftarrow{\alpha} \alpha.P + \beta.Q \xrightarrow{\beta} Q$

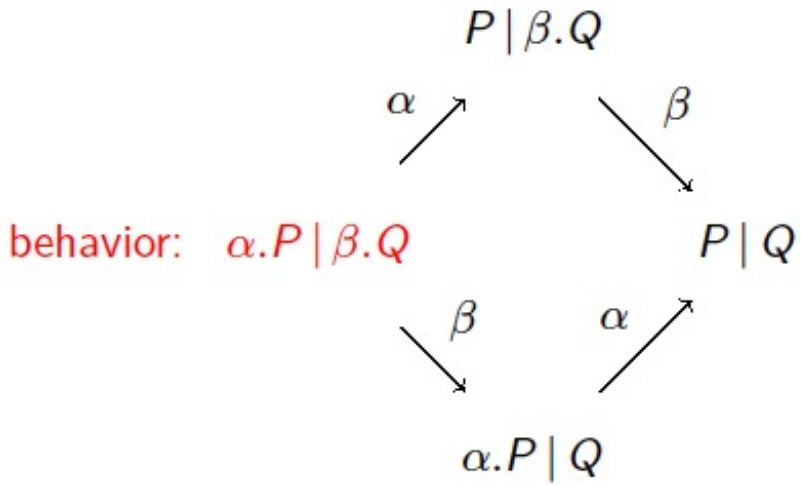
- Recursion: $X \stackrel{\text{def}}{=} \cdots .\alpha.X$

  behavior:



# Labelled Transition Systems (LTS)

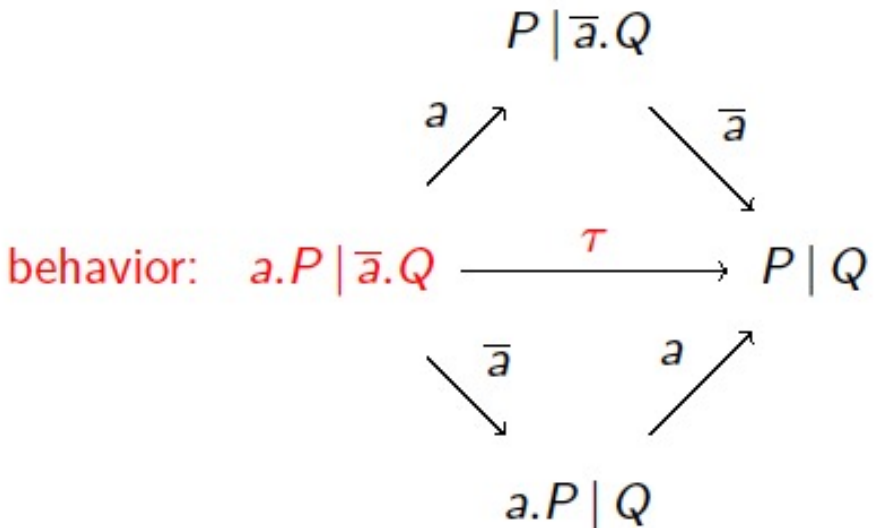
- Parallel composition: $\alpha.P \mid \beta.Q$
  Combines sequential composition and choice to obtain interleaving




- What about interaction?

# Labelled Transition Systems (LTS)

- Concurrent processes, i.e. $P$ and $Q$ in $P \mid Q$, may interact where their interfaces are compatible
- A synchronizing interaction between two processes (sub-systems), $P$ and $Q$, is an activity that is internal to $P \mid Q$
- Parallel composition: $\alpha.P \mid \beta.Q$
  Allows interaction if $\beta = \overline{\alpha}$

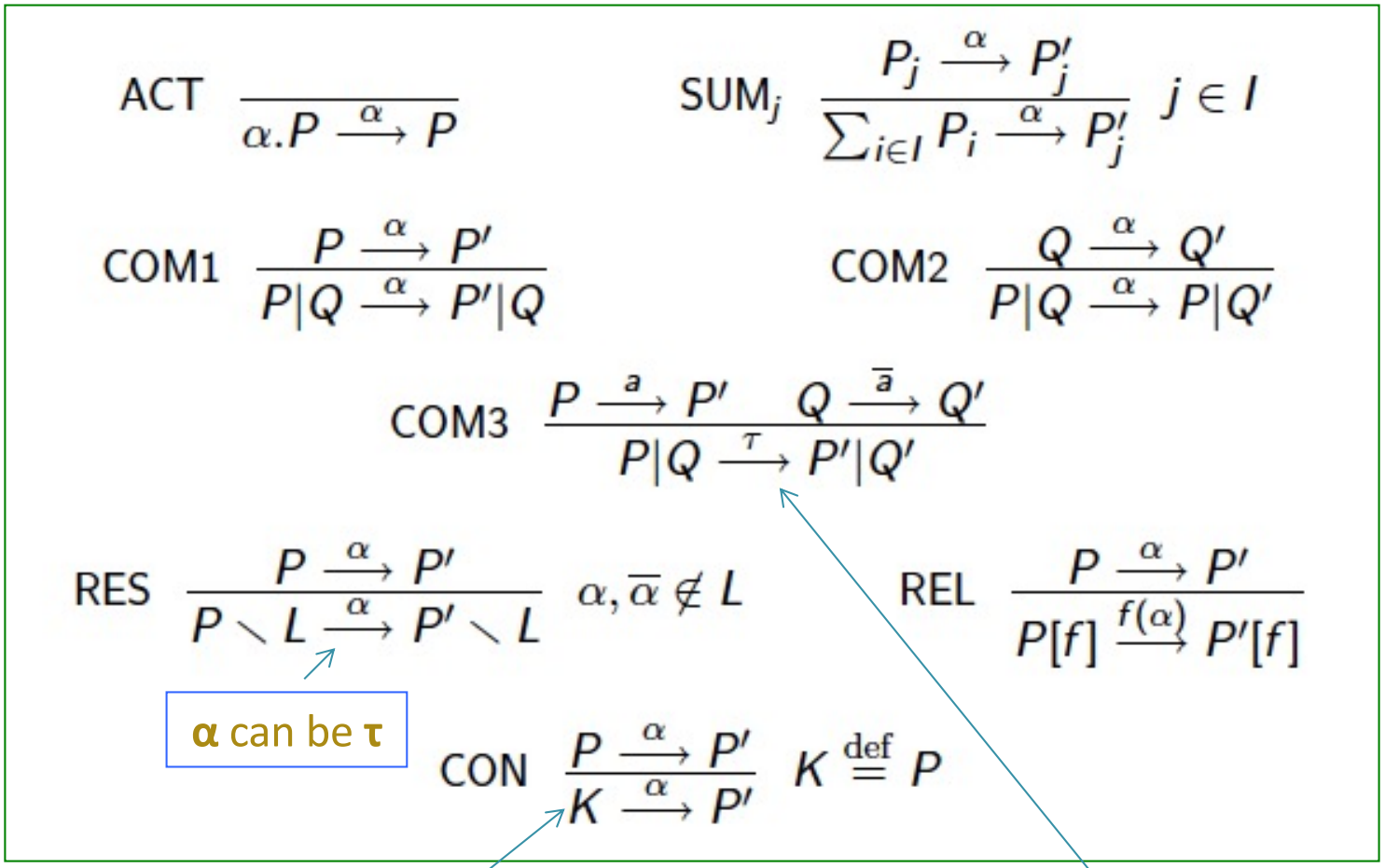


# Structural Operational Semantics (SOS)

The behaviour of a system is inferred using **syntax-driven rules** in the form

$$\frac{premises}{conclusion} \quad conditions$$

# SOS rules for CCS




α can be τ

The constant behaves as its defining expression

τ represents a completed and perfect action, internal to the composite agent P|Q

# Derivations

- Derive

$$((a.0 + b.0) \mid (\bar{a}.0 + c.0)) \mid Q \xrightarrow{\tau} 0 \mid 0 \mid Q$$

# Derivations

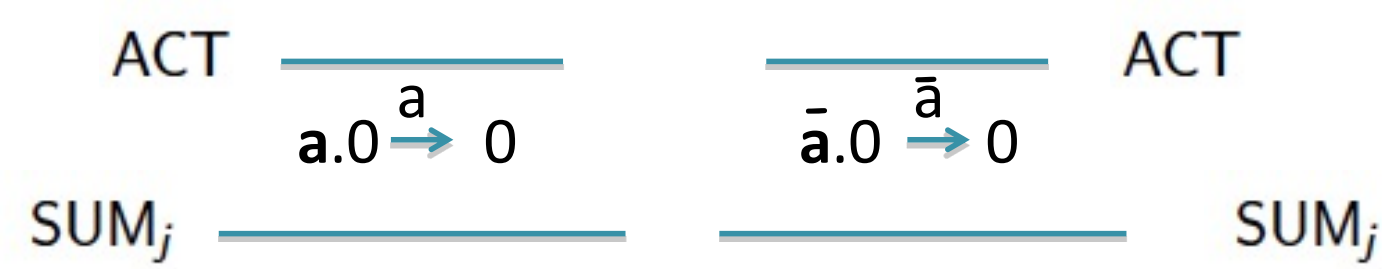
- Derive

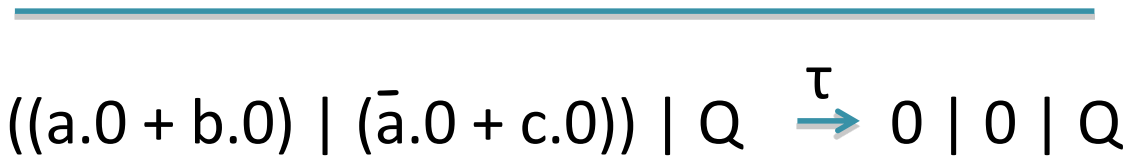
$$\frac{}{\mathbf{a}.0 \xrightarrow{a} 0}\ \text{ACT} \qquad \frac{}{\bar{\mathbf{a}}.0 \xrightarrow{\bar{a}} 0}\ \text{ACT}$$

$$\frac{}{((a.0 + b.0) \mid (\bar{a}.0 + c.0)) \mid Q \xrightarrow{\tau} 0 \mid 0 \mid Q}$$

# Derivations

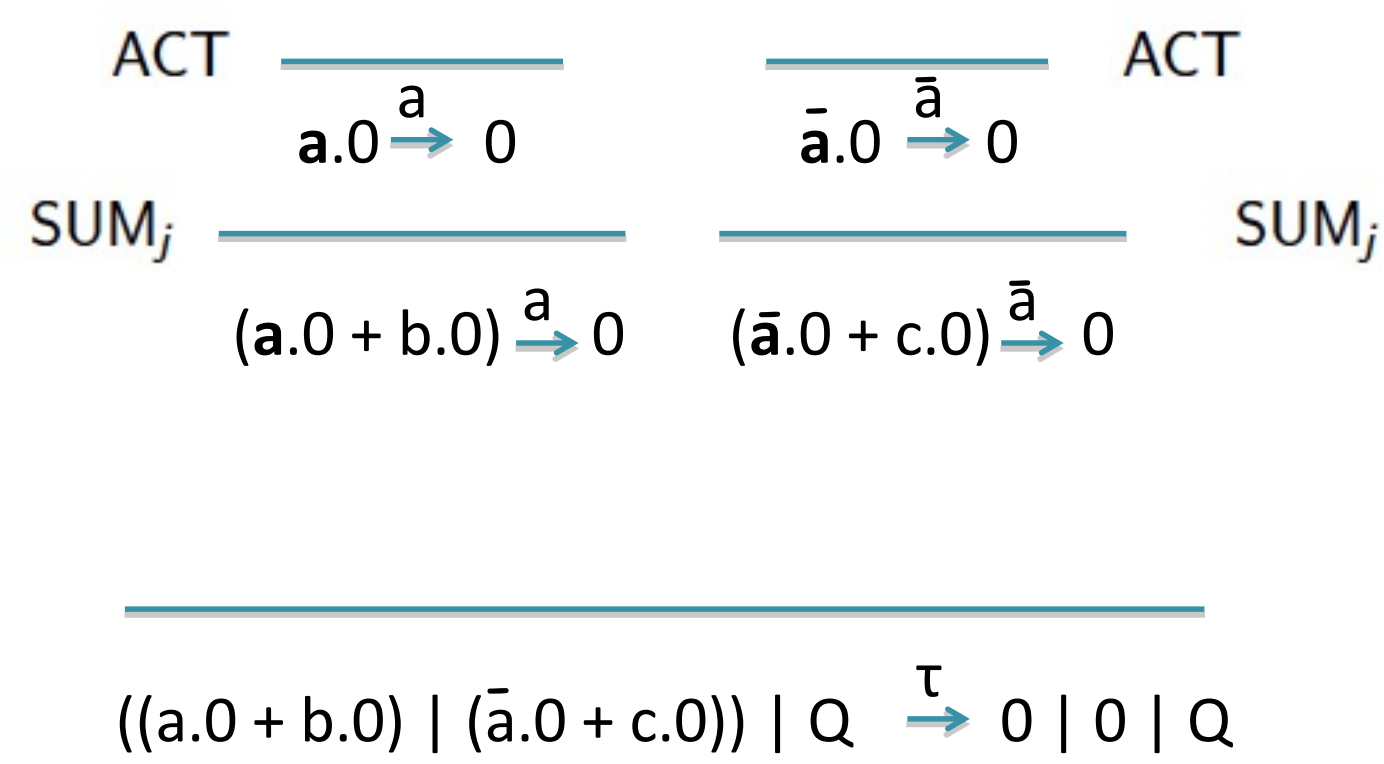
- Derive



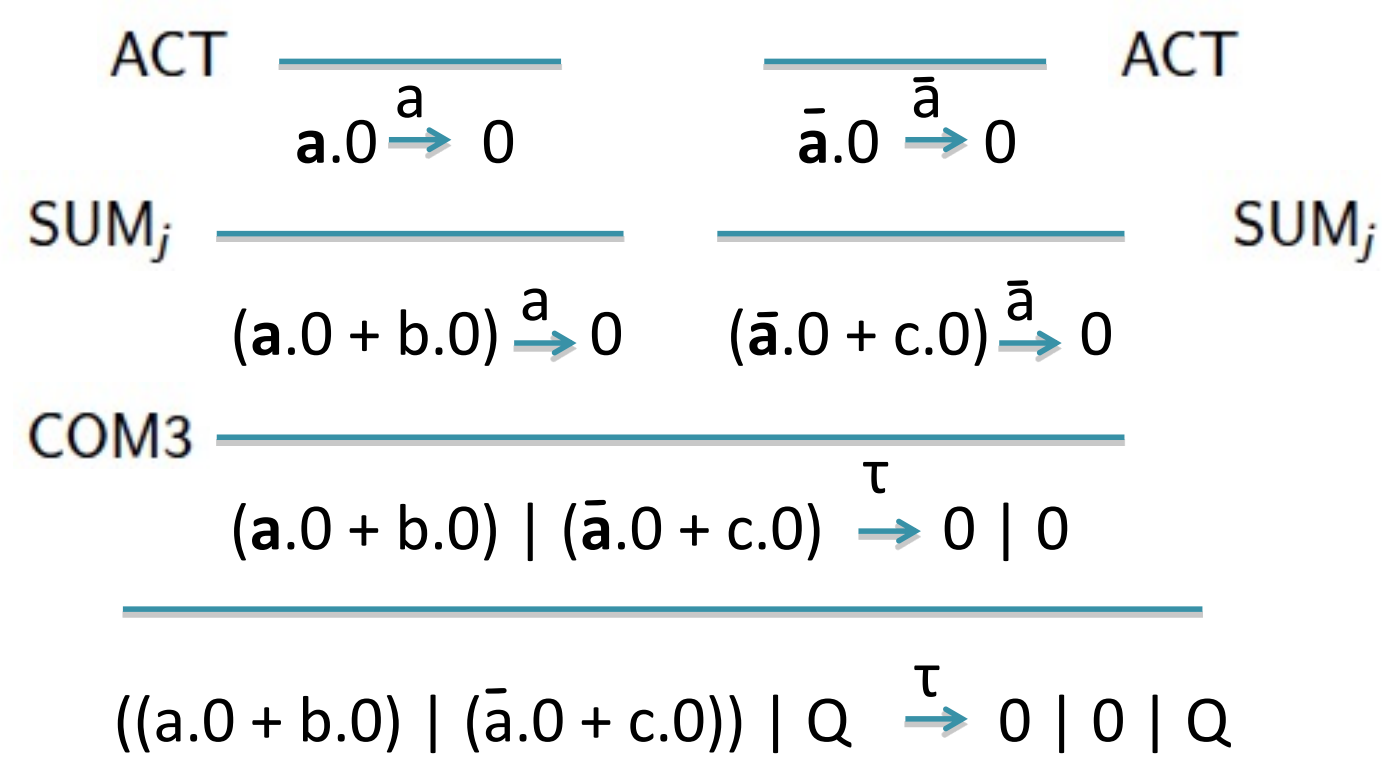
# Derivations

- Derive



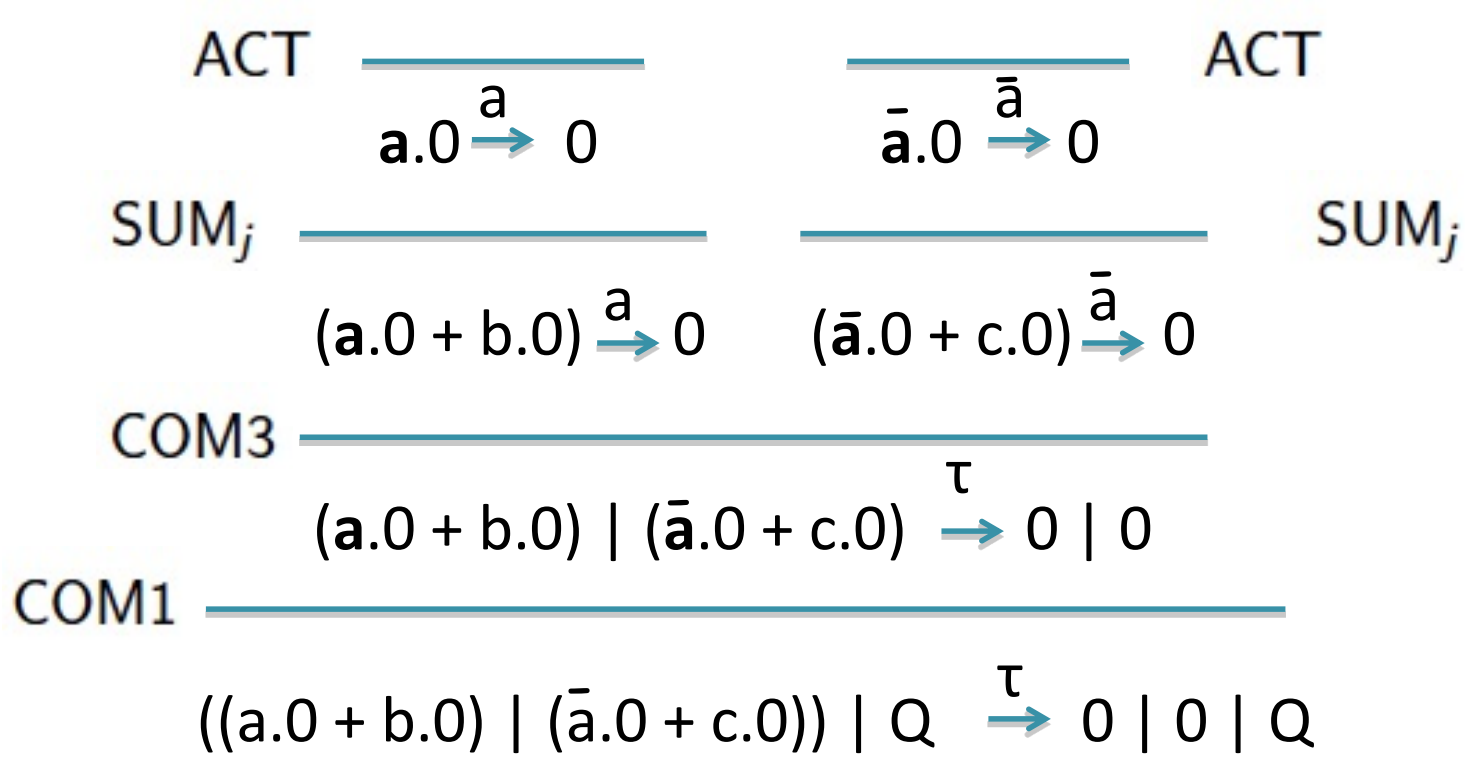
# Derivations

- Derive



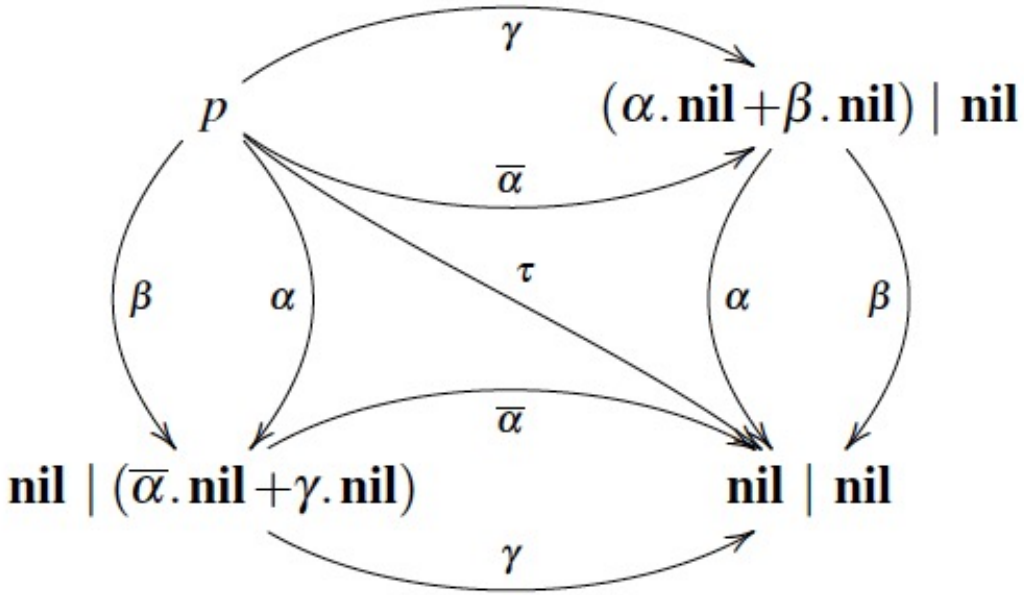
# Derivations

- Derive



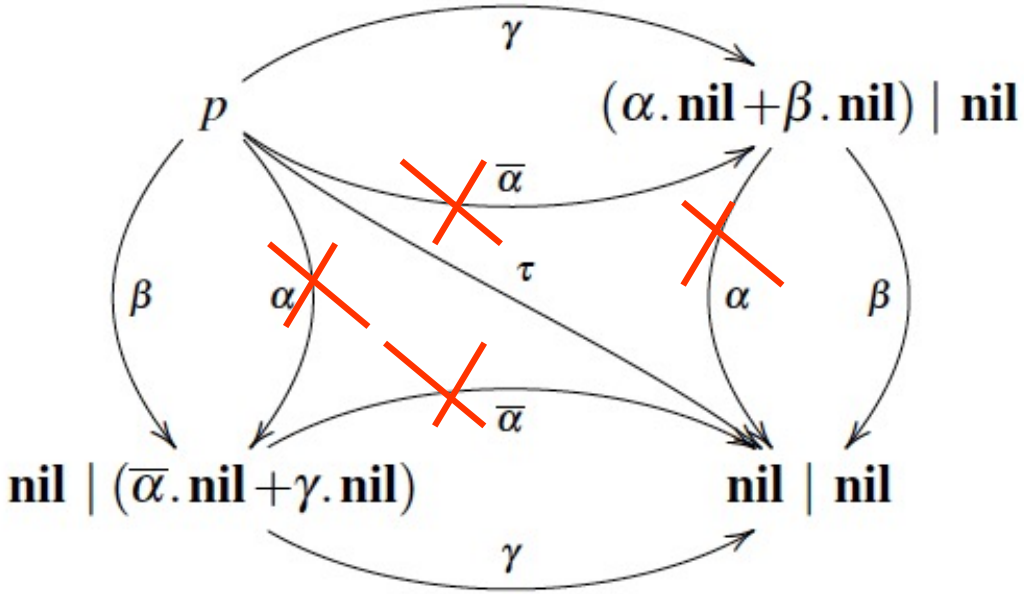
# LTS: an example

$$p \stackrel{\text{def}}{=} (\alpha.\mathbf{nil} + \beta.\mathbf{nil}) \mid (\overline{\alpha}.\mathbf{nil} + \gamma.\mathbf{nil})$$



# LTS: an example (cont.)

$$p \stackrel{\text{def}}{=} (\alpha.\mathbf{nil} + \beta.\mathbf{nil}) \mid (\overline{\alpha}.\mathbf{nil} + \gamma.\mathbf{nil})$$

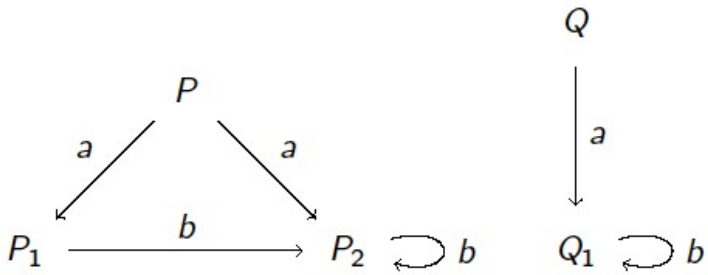



$$q \stackrel{\text{def}}{=} p \backslash \alpha$$

With **restrictions** we can produce **closed** systems: restricted names are unaccessible to the environment

# Behavioural equivalence

- Express the notions that two processes behave in the same way…
- … accordingly to an external observer



# Behavioural equivalence

## Strong Bisimilarity

Let $(Proc, Act, \{ \xrightarrow{\alpha} \mid \alpha \in Act\})$ be an LTS

### Strong Bisimulation

A binary relation $R \subseteq Proc \times Proc$ is a strong bisimulation iff whenever $(P, Q) \in R$ then for each $\alpha \in Act$:

- if $P \xrightarrow{\alpha} P'$ then $Q \xrightarrow{\alpha} Q'$ for some $Q'$ such that $(P', Q') \in R$
- if $Q \xrightarrow{\alpha} Q'$ then $P \xrightarrow{\alpha} P'$ for some $P'$ such that $(P', Q') \in R$

## Strong Bisimilarity

Two processes $P_1, P_2 \in Proc$ are strongly bisimilar ($P_1 \sim P_2$) if and only if there exists a strong bisimulation $R$ such that $(P_1, P_2) \in R$

$$\sim = \cup\{R \mid R \text{ is a strong bisimulation}\}$$

# Behavioural equivalence

## Strong Bisimilarity

Let $(Proc, Act, \{ \xrightarrow{\alpha} \mid \alpha \in Act\})$ be an LTS

### Strong Bisimulation

A binary relation $R \subseteq Proc \times Proc$ is a strong bisimulation iff whenever $(P, Q) \in R$ then for each $\alpha \in Act$:

- if $P \xrightarrow{\alpha} P'$ then $Q \xrightarrow{\alpha} Q'$ for some $Q'$ such that $(P', Q') \in R$
- if $Q \xrightarrow{\alpha} Q'$ then $P \xrightarrow{\alpha} P'$ for some $P'$ such that $(P', Q') \in R$

## Strong Bisimilarity

Two processes $P_1, P_2 \in Proc$ are strongly bisimilar ($P_1 \sim P_2$) if and only if there exists a strong bisimulation $R$ such that $(P_1, P_2) \in R$

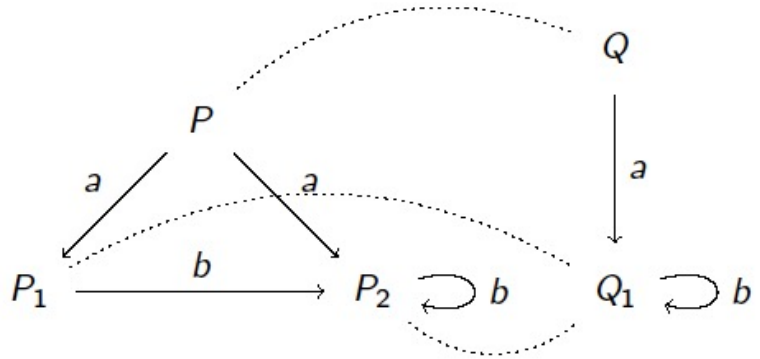
$$\sim = \cup\{R \mid R \text{ is a strong bisimulation}\}$$

- Two processes are bisimilar if there is a concrete strong bisimulation relation that relates them
- To show that two processes are bisimilar it suffices to exhibit such a concrete relation

# Behavioural equivalence

$$\mathcal{R} = \{(P, Q), (P_1, Q_1), (P_2, Q_1)\}$$

- $(P, Q)$ is in $\mathcal{R}$
- $\mathcal{R}$ is a bisimulation:
  - For each pair of states in $\mathcal{R}$, all possible transitions from the first can be matched by corresponding transitions from the second
  - For each pair of states in $\mathcal{R}$, all possible transitions from the second can be matched by corresponding transitions from the first



# Behavioural equivalence

$$\mathrm{P} \stackrel{\text{def}}{=} a.(b.0 + c.0)$$
$$\mathrm{Q} \stackrel{\text{def}}{=} a.b.0 + a.c.0$$

$$P \not\sim Q$$

After the action *a*, in P there is still the choice between the two branches,
While in Q, there is not (the choice has been made)

# Outline

- Motivation
- Key notion
- **CCS**
- **From CCS to Pi calculus**
- Pi calculus: syntax and semantics
- The role of restriction for security
- Modeling protocols in Pi calculus
- Spi Calculus: a hint
- Conclusions and what next

# CCS vs Pi Calculus

**CCS**

$$a.P \mid \overline{a}.Q \xrightarrow{\tau} P \mid Q$$

**Value-passing CCS**

can be encoded in pure CCS

$$a(x).P \mid \overline{a}\langle v\rangle.Q \xrightarrow{\tau} P\{v/x\} \mid Q$$

x is bound in P, v is a value

**Pi calculus**

y is bound in P, z can be a channel name

$$x(y).P \mid \overline{x}\langle z\rangle.Q \xrightarrow{\tau} P\{z/y\} \mid Q$$

# Towards Value-passing CCS

Suppose C wants to buy a pizza from P*

$C \triangleq \overline{askPizza}.\overline{pay}.pizza \qquad P \triangleq askPizza.pay.\overline{pizza}.P$

**CCS**

$$C|P \xrightarrow{\tau} \overline{pay}.pizza|pay.\overline{pizza}.P \xrightarrow{\tau} pizza|\overline{pizza}.P \xrightarrow{\tau} \mathbf{0}|P \equiv P$$

Clear in a while

Handshake synchronization

* [R. De Nicola, R. Bruni]

# Towards Value-passing CCS

Suppose C wants to buy a pizza from P*

$$C \triangleq \overline{askPizza}.\overline{pay}.pizza \qquad P \triangleq askPizza.pay.\overline{pizza}.P$$

**CCS**

$$C|P \xrightarrow{\tau} \overline{pay}.pizza|pay.\overline{pizza}.P \xrightarrow{\tau} pizza|\overline{pizza}.P \xrightarrow{\tau} \mathbf{0}|P \equiv P$$

Clear in a while

$$C \triangleq \overline{askPizza}\langle margherita\rangle.\overline{pay}\langle 5\ Euro\rangle.pizza$$

$$P \triangleq askPizza(x).pay(y).\mathbf{if}\ y = price(x)\ \mathbf{then}\ \overline{pizza}.P\ \mathbf{else\ if}\ y > price(x)\ \mathbf{then}\ \overline{pizza.output}\langle y - price(x)\rangle.P\ \mathbf{else}\ \overline{askMoney}...$$

**Value passing CCS**

Handshake synchronization
+
the possibility to exchange data

* [R. De Nicola, R. Bruni]

# Towards Pi Calculus

Home delivery!

$$C \triangleq \overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle$$
$$P \triangleq askPizza(y).pay.(\nu\, pizza)\overline{y}\langle pizza\rangle.P$$

**Pi Calculus**

$$\overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid askPizza(y).pay.(\nu\, pizza)\overline{y}\langle pizza\rangle.P$$

... + the possibility to create and send channel names

# Towards Pi Calculus

Home delivery

$$C \triangleq \overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle$$
$$P \triangleq askPizza(y).pay.(\nu\, pizza)\overline{y}\langle pizza\rangle.P$$

**Pi Calculus**

$$\overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid askPizza(y).pay.(\nu\, pizza)\overline{y}\langle pizza\rangle.P$$
$$\xrightarrow{\tau} \overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid pay.(\nu\, pizza)\overline{myHome}\langle pizza\rangle.P$$

... + the possibility to create and send channel names

# Towards Pi Calculus

Home delivery

$$C \triangleq \overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle$$
$$P \triangleq askPizza(y).pay.(\nu\, pizza)\overline{y}\langle pizza\rangle.P$$

**Pi Calculus**

$$\overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid askPizza(y).pay.(\nu\, pizza)\overline{y}\langle pizza\rangle.P$$
$$\xrightarrow{\tau} \overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid pay.(\nu\, pizza)\overline{myHome}\langle pizza\rangle.P$$

... + the possibility to exchange channel names

# Towards Pi Calculus

Home delivery

$$C \triangleq \overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle$$
$$P \triangleq askPizza(y).pay.(\nu\, pizza)\overline{y}\langle pizza\rangle.P$$

**Pi Calculus**

$$\overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid askPizza(y).pay.(\nu\, pizza)\overline{y}\langle pizza\rangle.P$$
$$\xrightarrow{\tau} \overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid pay.(\nu pizza)\overline{myHome}\langle pizza\rangle.P$$
$$\xrightarrow{\tau} myHome(x).\overline{eat}\langle x\rangle \mid (\nu pizza)\overline{myHome}\langle pizza\rangle.P$$

... + the possibility to exchange channel names

# Towards Pi Calculus

Home delivery

$$C \triangleq \overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle$$
$$P \triangleq askPizza(y).pay.(\nu\, pizza)\overline{y}\langle pizza\rangle.P$$

**Pi Calculus**

$$\begin{aligned}
&\overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid \\
&\qquad\qquad askPizza(y).pay.(\nu\, pizza)\overline{y}\langle pizza\rangle.P \\
&\xrightarrow{\tau} \overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid pay.(\nu pizza)\overline{myHome}\langle pizza\rangle.P \\
&\xrightarrow{\tau} myHome(x).\overline{eat}\langle x\rangle \mid (\nu pizza)\overline{myHome}\langle pizza\rangle.P \\
&\xrightarrow{\tau} (\nu pizza)(\overline{eat}\langle pizza\rangle \mid P)
\end{aligned}$$

... + the possibility to exchange channel names

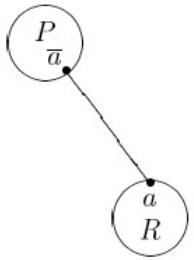
# Towards the Pi Calculus

It is not difficult to extend the calculus (CCS) to allow for the exchange of values such as integers over the channel during a synchronization. Doing so does not fundamentally change the character of the calculus. A variation on this, however, does change the calculus in a highly nontrivial way, and that is to allow for the communication of channel names during synchronization. This yields the pi-calculus. To see why such an extension might be useful, consider the following scenario. It shows that passing channel names around can be used to model process mobility. Intuitively, a process is characterized by the channels it exposes to the world, that can be used to communicate with it. These channels act as an interface to the process. […] A process that sends *x* to another process in some sense sends the capability to access *x* to that process. This captures the mobility of process P, although more accurately it captures the mobility of the capability to access P.

Review of **The pi-calculus: A Theory of Mobile Processes**
**by D. Sangiorgi and D. Walker Riccardo Pucella**

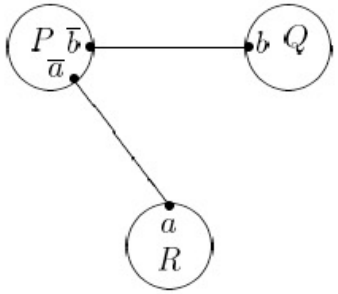
# Mobility

- Suppose P wishes to send 5 to R, along the private channel a




P|R $(\overline{a}5.P' \mid a(x).R')\backslash a$

- Now, suppose that P wish to delegate to Q the transmission of 5 (P is connected with Q via b)



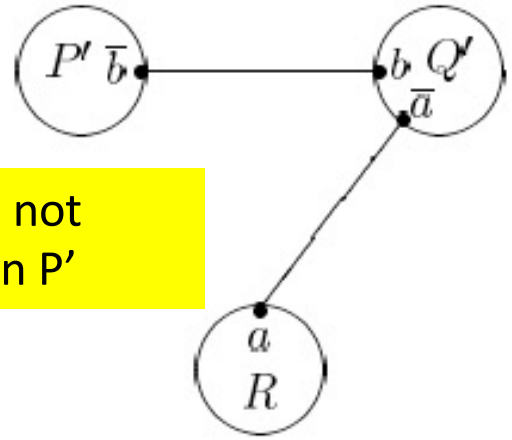

P|Q|R

$(\overline{b}a.\overline{b}5.P' \mid b(y).b(z).\overline{y}z.\mathbf{0} \mid a(x).R')\backslash a\backslash b$

# Mobility: name extrusion

- After two communications, both along b:

$$(P' \mid \overline{a}5.\mathbf{0} \mid a(x).R')\backslash a\backslash b$$




P|R

Restriction is not a static operator

If a does not appear in P'

- Note that a, b, x, y, 5 are all just names
- The other class of entities are agents

- P's link a has moved to Q

as a reference

# Outline

- Motivation
- Key notion
- **CCS**
- From CCS to Pi calculus
- **Pi calculus: syntax and semantics**
- The role of restriction for security
- Modeling protocols in Pi calculus
- Spi Calculus: a hint
- Conclusions and what next

# Pi Calculus

Milner:

*It will appear as though we reduce all concurrent computation to something like a **cocktail party**, in which the only purpose of communication is to transmit (or to receive) a name which will admit further communications. Surprisingly, this meagre basis is enough to encode computation over an arbitrary data types... We tentatively call our new calculus the **pi calculus**, since it aims for universality (at an elementary level) **for concurrent computation**, just as the **lambda calculus** is universal for **functional computation**.*

# Pi Calculus or π calculus

- Introduced by Milner, Parrow, Walker in 1989. It extends CCS [1980]: channel names can be passed between processes

Small but expressive programming language

- Core language for concurrent programming:
  - **Send** and **receive** primitives (point-to-point communication between concurrent processes)
  - **Message passing**
  - **Dynamic creation of channels**: a channel is a transferable capability of communication
  - Creation of **new** tags (e.g., keys)
  - **Mobility** and network reconfiguration: interconnections change with interaction

# Pi Calculus

Warning: many, many variants! E.g.

- what is a message?
  - an atomic name
  - . . .
  - a generic term (e.g., f(g(x), y))
- many ways to define the semantics
- We focus on the main concepts rather than on the details of the particular variant we present here.

# Names and processes

- An infinite set of names (channels, links, ports), with no structure: a, b, . . . , p, q, r, . . . , x, y, . . .
- A set of entities, called processes: A,B, . . . , P, Q, . . .
- Semantics through transitions
- $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow \cdots$
- Sometimes, we shall annotate the transitions with some label, to make some event observable

# Syntax

| | |
|---|---|
| P ::= **0** | inactive process |
| \| τ.P | silent action |
| \| $\bar{x}$y.P | output |
| \| x(y).P | input |
| \| [x = y] P | match |
| \| (x) P [or (υ x) or P\x] | channel creation or restriction |
| \| P \| P | parallel composition |
| \| P + P | non deterministic choice |
| \| ! P | replication |

We take processes up to **alpha-conversion**: a bound names can be renamed with a fresh name

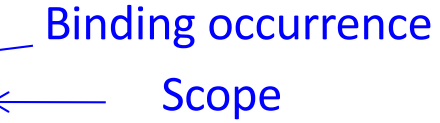
- Prefix "." imposes an order upon actions

# Remarks

- Occurrences of the inactive process **0** will sometimes be omitted, thus, e.g., $\bar{x}\langle y\rangle.\mathbf{0}$ will be written $\bar{x}\langle y\rangle$
- $\tau$ indicates an internal action
- $x(y)$ indicates input, while $\bar{x}y$ indicates output
- $[x = y]P$ is known as name matching: it is equivalent to **if x = y then P**, otherwise it is stuck
- $(x)P$ is alike CCS restriction $P\backslash x$
- $!P$ models replication (and hence recursion) and denotes the parallel composition of an arbitrary number of copies of P, i.e., it behaves like $P|P|P|\ldots$

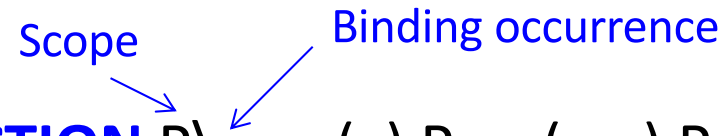
# Two forms of binding

- **INPUT** x(y).P




Like the binding of z in λz.M in λ-calculus

  - y is a *placeholder* for any name which may be received on the channel x. It is a bound name




- **RESTRICTION** P\y or (y) P or (υ y) P
  - y creates a **new name** (**private** to P) and binds it to y in P: y cannot be confused with any other name in P
  - Unlike for CCS, the scope y can be dynamically extended, e.g., the process (y)($\bar{x}$y)|x(z).Q can extrude the **scope** of the restricted name y and then wait to receive some data on it
  - These names are **bound**, the others are **free**

# Remarks (cont.)

- In x(y).P e (y)P, the name y is bound in P (i.e., P is the **scope** of such name). A name that is not bound is called free
- fn(P) e bn(P) are the sets of all free, resp. bound, names of P

$$\mathrm{bn}(x(y).p) \stackrel{\text{def}}{=} \{y\} \cup \mathrm{bn}(p) \qquad \mathrm{bn}((y).p) \stackrel{\text{def}}{=} \{y\} \cup \mathrm{bn}(p)$$

- n(P) = fn(P) U bn(P)
- We take processes up to alpha-conversion, denoted by $=_\alpha$, which permits renaming of a bound name with a fresh name that is not already used

# Substitution

- A substitution $\sigma : N \rightarrow N$ is a function on names that is the identity except on a finite set of names
- We write $\{y_1,\ldots,y_n/x_1,\ldots,x_n\}^*$ for the substitution $\sigma$ such that $x_i = y_i$ for $i = 1,\ldots, n$ and $x = x$ otherwise
- When applying $\sigma$ to a process P we want to rename only the free occurrences of names x in P, **not** the bound ones
- unintended **capture of names** x by binders of P must be avoided
- How to define $P\{z/x\}$ if $P = \mathbf{x}(y).\bar{w}\mathbf{x}$?
- $P\{z/x\}$ =

*also $\{y_1|\text{-->}x_1,\ldots, y_n|\text{-->}x_n\}$

# Substitution

- A substitution $\sigma : N \rightarrow N$ is a function on names that is the identity except on a finite set of names
- We write $\{y_1,\ldots,y_n/x_1,\ldots,x_n\}^*$ for the substitution $\sigma$ such that $x_i = y_i$ for $i = 1,\ldots, n$ and $x = x$ otherwise
- When applying $\sigma$ to a process P we want to rename only the free occurrences of names x in P, **not** the bound ones
- unintended **capture of names** x by binders of P must be avoided
- How to define $P\{z/x\}$ if $P = \mathbf{x}(y).\bar{w}\mathbf{x}$?
- $P\{z/x\} = z(y).\bar{w}z$

*also $\{y_1|\text{-->}x_1,\ldots, y_n|\text{-->}x_n\}$

# Substitution (cont.)

- How to define $P\{x/z\}$ if $P = y(x).\bar{x}z$?

# Substitution (cont.)

- How to define $P\{x/z\}$ if $P = y(x).\bar{x}z$?
- $y(x).\bar{x}x$ NO: name confusion here

# Substitution (cont.)

A bound object is a reference

- How to define P{x/z} if P = y(x).x̄z?
- y(x).x̄x NO: name confusion must be avoided
- Bound names cannot be substituted… but can be alpha-converted first

# Substitution (cont.)

A bound object is a reference

- How to define $P\{x/z\}$ if $P = y(x).\bar{x}z$?
- $y(x).\bar{x}x$ NO: name confusion must be avoided
- Bound names cannot be substituted… but can be alpha-converted first

  $y(\mathbf{x}).\bar{\mathbf{x}}z =_\alpha y(\mathbf{w}).\bar{\mathbf{w}}z$ where **w** is a **fresh** name
- Bound names are a sort of placeholders for actual values

  $y(x).\bar{x}z$: whatever (x or w, who cares!) is received on y is the channel used to send z
- Hence, we first alpha-convert and then substitute
- $y(x).\bar{x}z\{x/z\} =_\alpha y(w).\bar{w}\mathbf{z}\{x/z\} = y(w).\bar{w}\mathbf{x}$ OK

# Semantics: actions

- A transition in the pi calculus is of the form $P \xrightarrow{\alpha} Q$
- Intuitively, it means that P can evolve into Q, and in doing so perform the action α

The possible forms are:

- $P \xrightarrow{\tau} Q$ P can evolve into Q, with no interaction with the environment [silent action]
- $P \xrightarrow{\overline{x}y} Q$ P emits the free name y on the channel x [free output]
- $P \xrightarrow{x(y)} Q$ P can receive any name w on x, becoming Q{w/y} [free in]
- $P \xrightarrow{\overline{x}(y)} Q$ P emits a private name y on x [bound output] Bound outputs arise from free outputs which carry names out of their scope

# Semantics (cont.)

Tau $\tau.P \xrightarrow{\tau} P$

Inp $x(y).P \xrightarrow{x(w)} P\{w/y\} \quad w \notin \mathrm{fn}(\nu y\, P)$ Out $\overline{x}y.P \xrightarrow{\overline{x}y} P$

- y is a placeholder and
- w is fresh, i.e., it does not occur in P: w=z or w not in fn(P)

# Semantics (cont.)

Tau $\tau.P \xrightarrow{\tau} P$

Inp $x(y).P \xrightarrow{x(w)} P\{w/y\} \quad w \notin \mathrm{fn}(\nu y\, P)$ Out $\overline{x}y.P \xrightarrow{\overline{x}y} P$

- y is a placeholder and
- w is fresh, i.e., it does not occur in P: w=z or w not in fn(P)

Match $\dfrac{P \xrightarrow{\alpha} P'}{[x=x]P \xrightarrow{\alpha} P'}$

Bang $\dfrac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$

Sum $\dfrac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$

To avoid name captures

Par $\dfrac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$

# Semantics (cont.)

$$\text{Com}\quad \frac{P \xrightarrow{\overline{x}y} P' \quad Q \xrightarrow{x(y)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\text{Close}\quad \frac{P \xrightarrow{\overline{x}(w)} P' \quad Q \xrightarrow{x(w)} Q'}{P \mid Q \xrightarrow{\tau} \nu w\,(P' \mid Q')}$$

It **closes** the scope of restriction enclosing Q': w becomes private btw P' and Q'

$$\text{Open}\quad \frac{P \xrightarrow{\overline{x}y} P'}{\nu y\, P \xrightarrow{\overline{x}(w)} P'\{w/y\}}\quad y \neq x,\ w \notin \text{fn}(\nu y\, P')$$

It **opens** the scope of restriction

Bound output label

Extruded names must be fresh

$$\text{Res}\quad \frac{P \xrightarrow{\alpha} P'}{\nu y\, P \xrightarrow{\alpha} \nu y\, P'}\quad y \notin \text{n}(\alpha)$$

Free and bound names

# Structural Congruence

The syntax of processes is to some extent too concrete:

- The order processes are composed in parallel should not matter
- The order processes "summed" should not matter
- The order names are restricted should not matter

Processes differing only for the above behave more or less the same

$$(ALPHA)\quad \frac{P =_\alpha Q \quad Q \xrightarrow{\alpha} Q'}{P \xrightarrow{\alpha} Q'}$$

# Structural Congruence (cont.)

$(P_{\cong}, +, 0)$ and $(P_{\cong}, |, 0)$ are commutative monoids

$$p + \mathbf{nil} \equiv p \qquad p + q \equiv q + p \qquad (p+q)+r \equiv p+(q+r)$$
$$p \mid \mathbf{nil} \equiv p \qquad p \mid q \equiv q \mid p \qquad (p \mid q) \mid r \equiv p \mid (q \mid r)$$
$$(x)\mathbf{nil} \equiv \mathbf{nil} \qquad (y)(x)p \equiv (x)(y)p \qquad (x)(p \mid q) \equiv p \mid (x)q \text{ if } x \notin \mathrm{fn}(p)$$
$$[x=y]\mathbf{nil} \equiv \mathbf{nil} \qquad [x=x]p \equiv p \qquad p \mid !p \equiv !p$$

We need fewer semantic rules

# Derivations

Derive

$$(((y)\bar{x}y.P) \mid Q \mid x(z).R) \xrightarrow{\tau} (v)((P\{v/y\} \mid Q) \mid R\{v/z\})$$

# Derivations

Derive

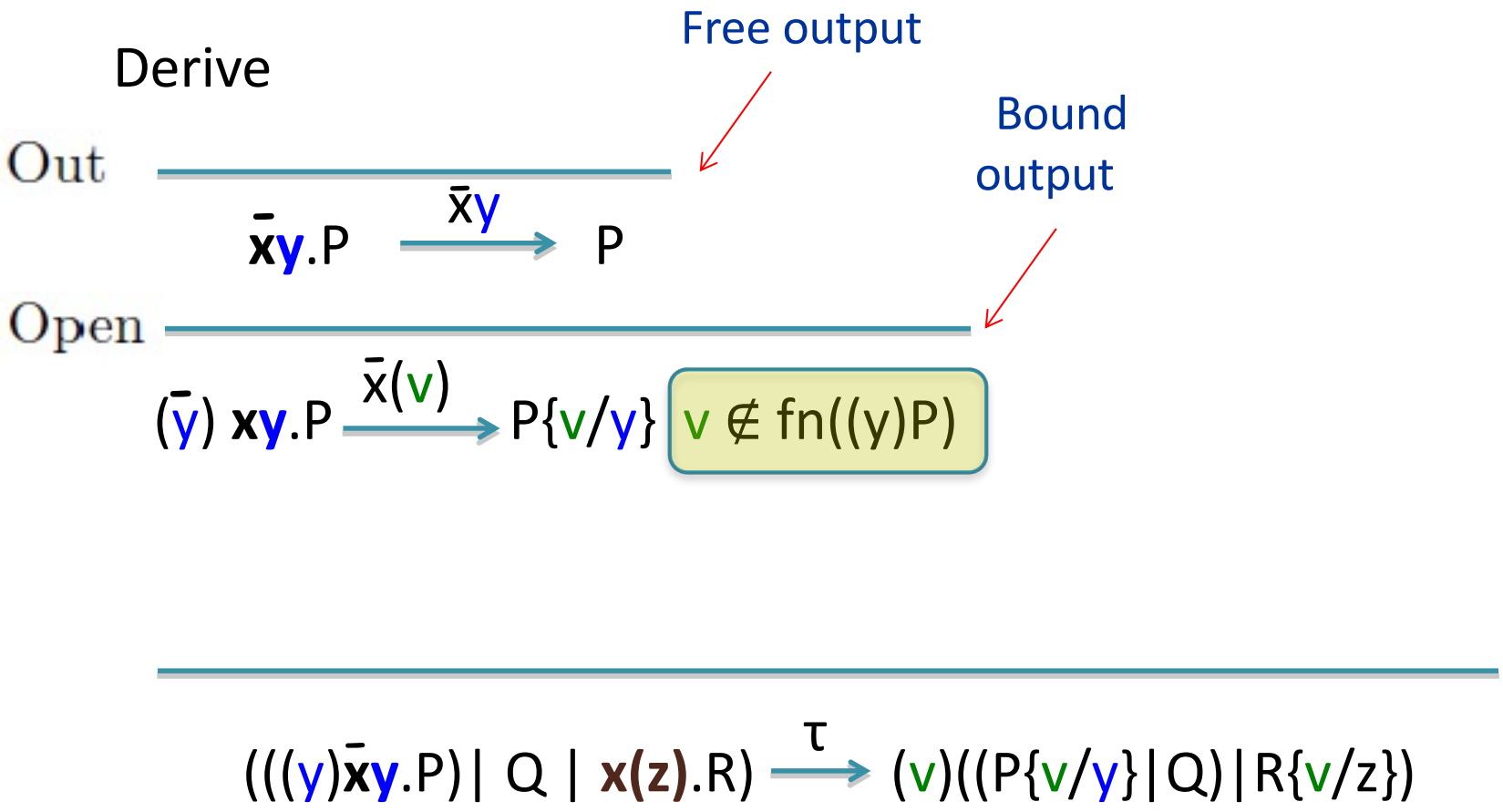
Free output

Out

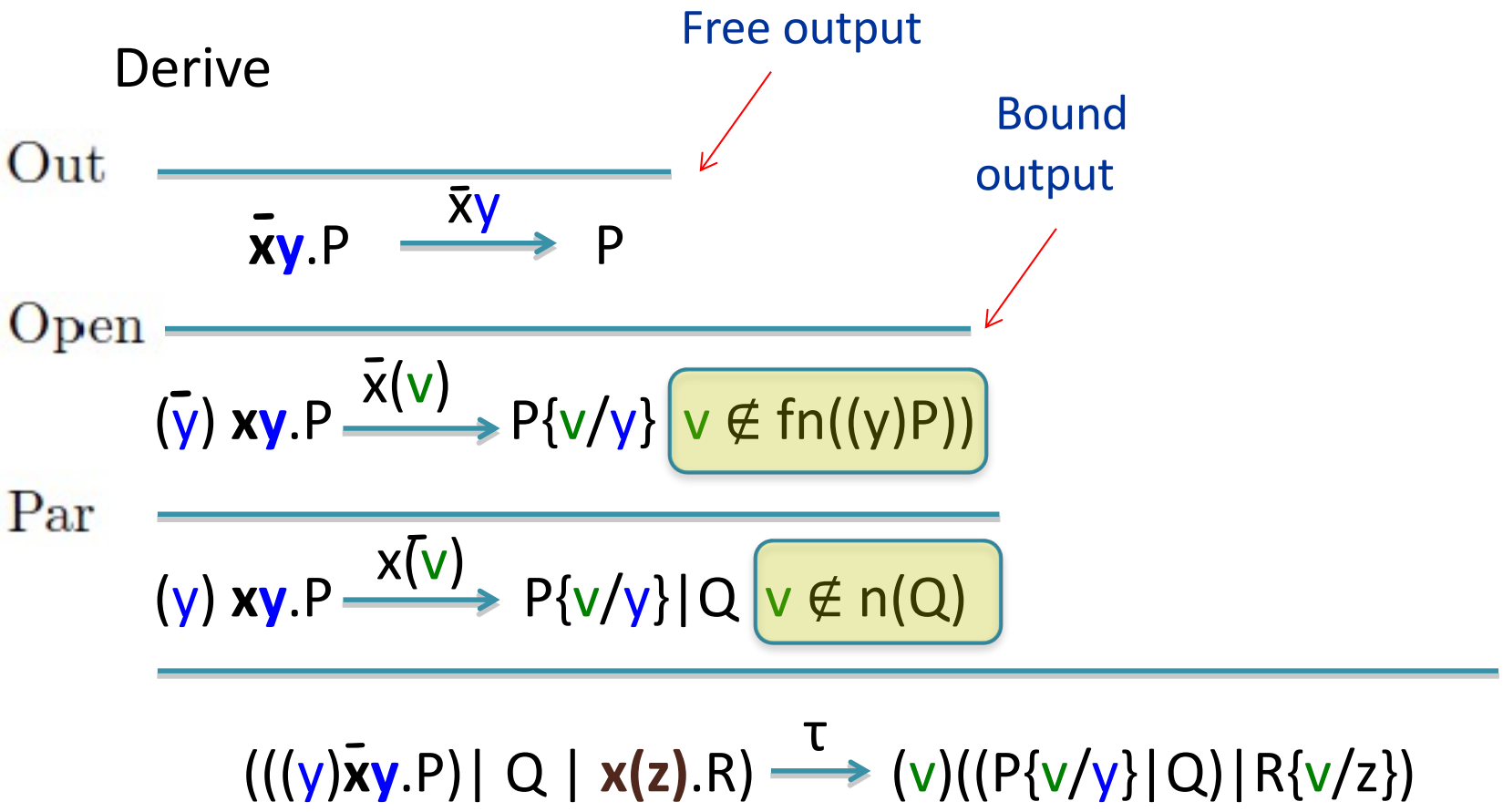
$\bar{x}y.P \xrightarrow{\bar{x}y} P$

$(((y)\bar{x}y.P) \mid Q \mid x(z).R) \xrightarrow{\tau} (v)((P\{v/y\} \mid Q) \mid R\{v/z\})$

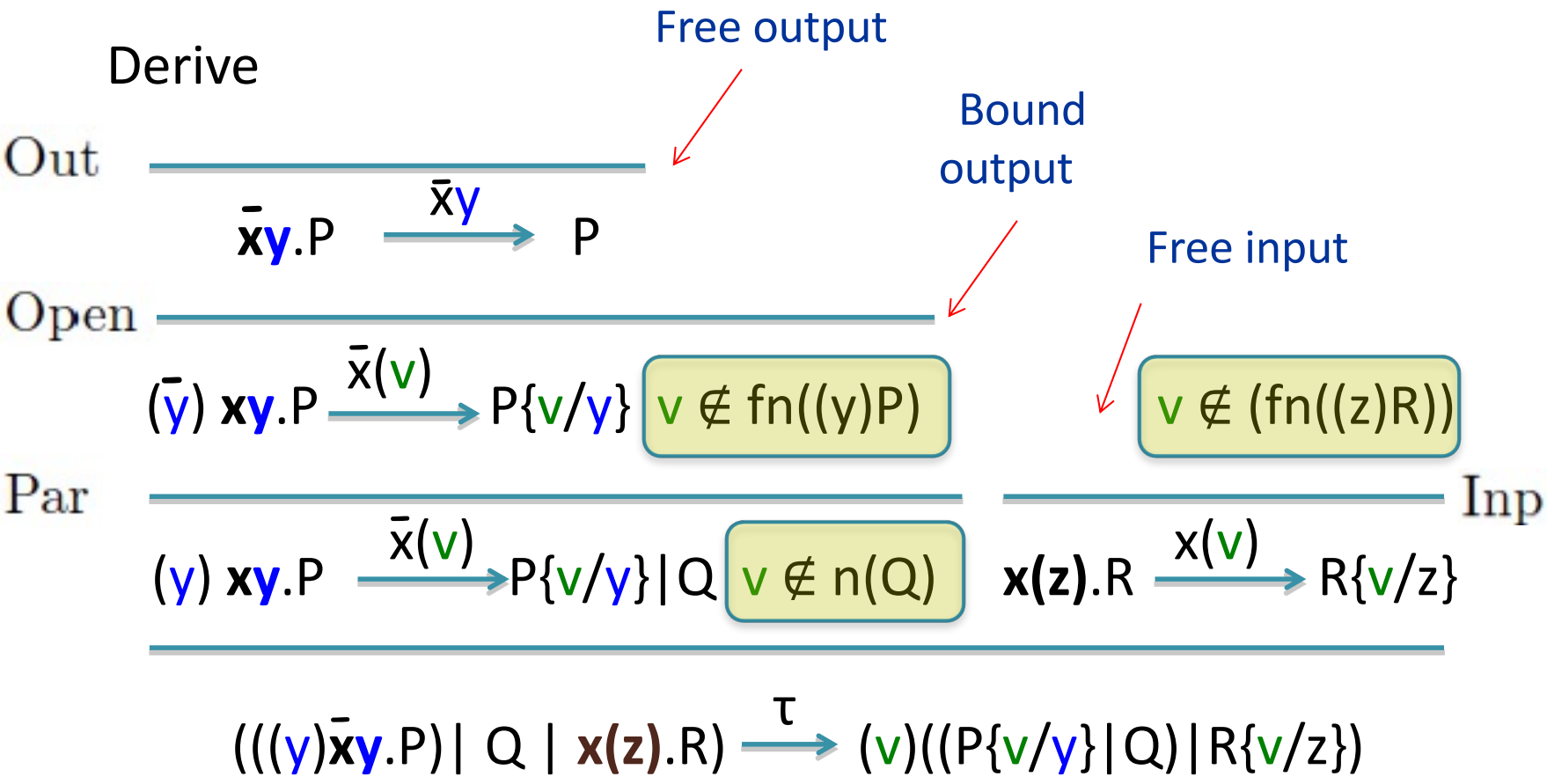
# Derivations



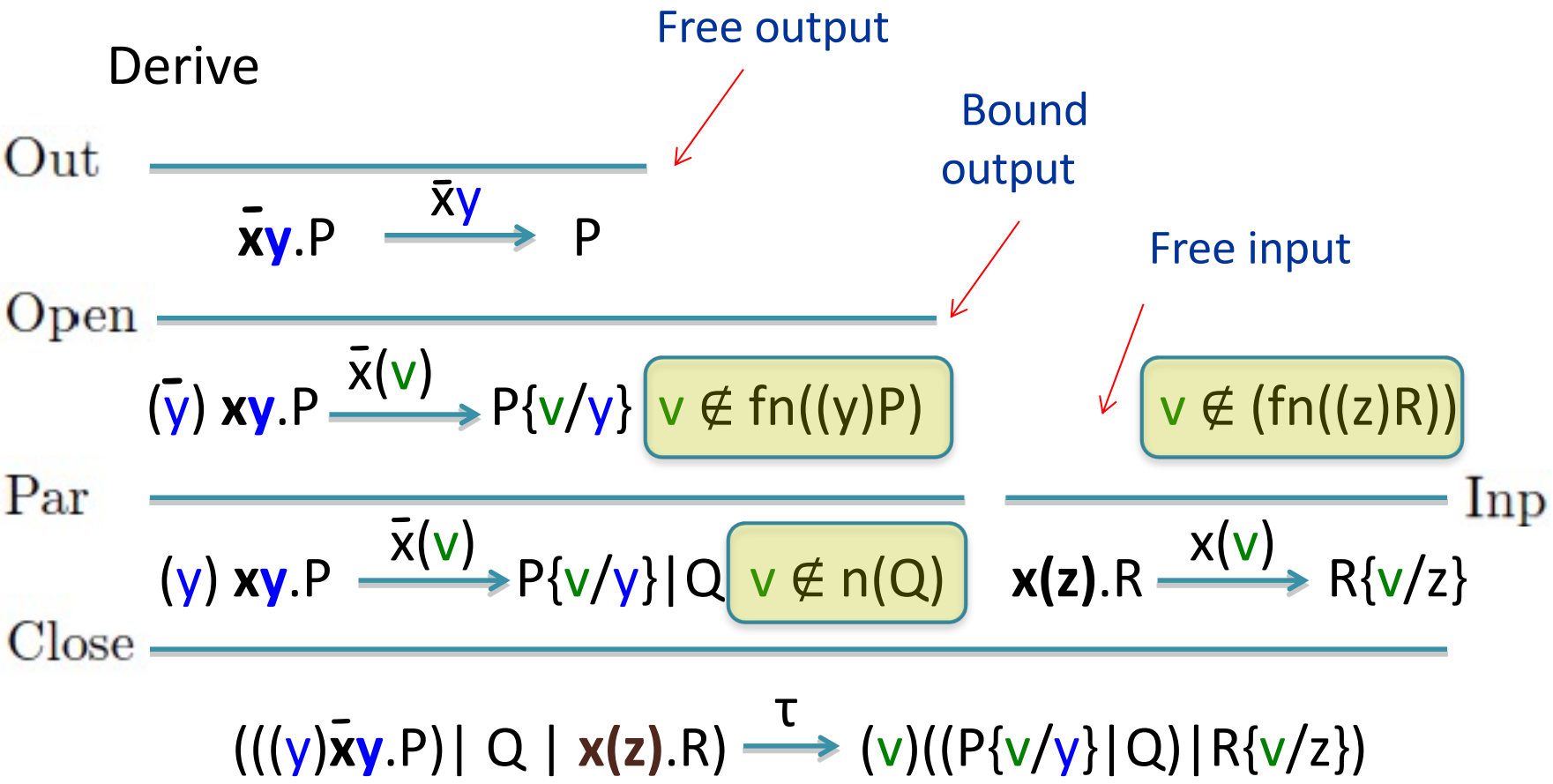
# Derivations



# Derivations

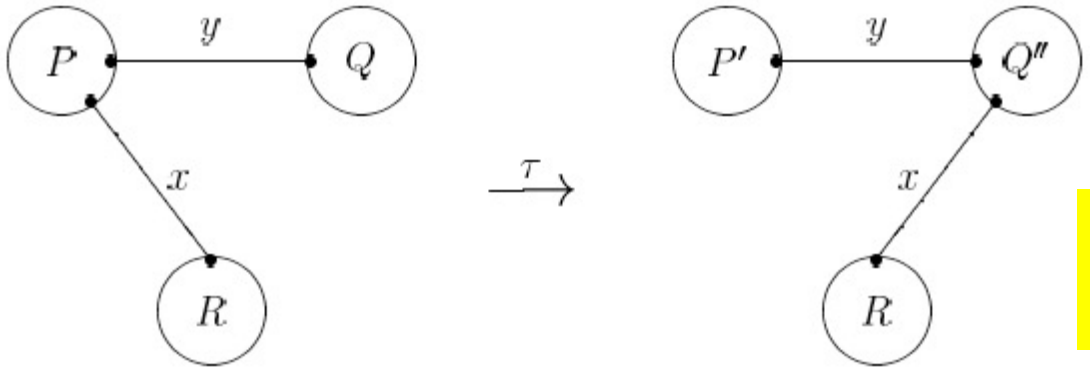


# Derivations



# Restriction

The agent P has a link x to R and wishes to pass x along its link y to Q. Q is willing to receive it

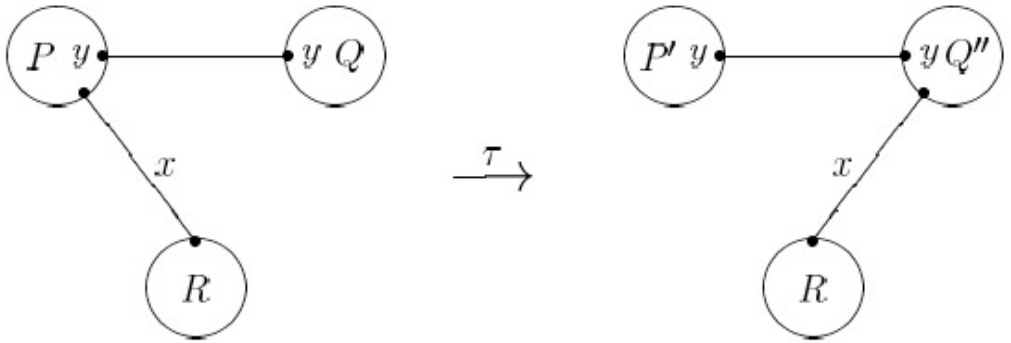



x does not belong to fn(P')

$$\overline{y}x.P' \mid y(z).Q' \mid R \xrightarrow{\tau} P' \mid Q'\{x/z\} \mid R$$

# Example: private link passing

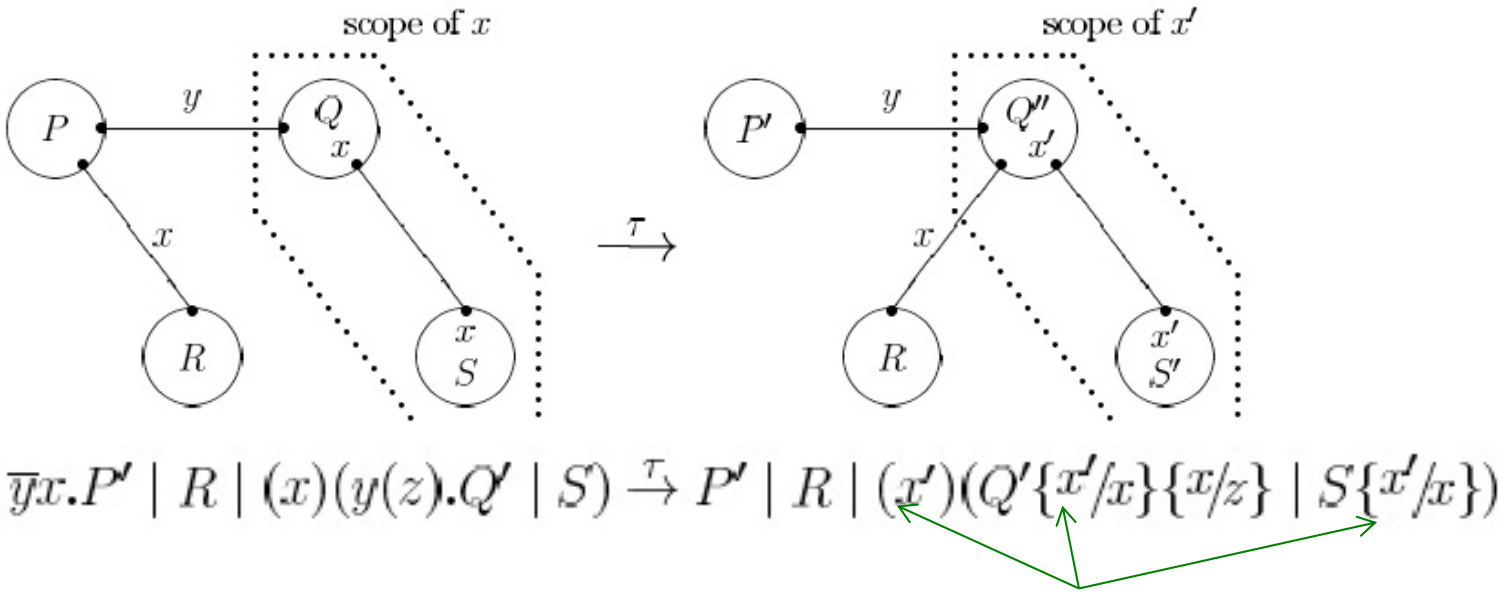
The link is private




$$(y)(\overline{y}x.P' \mid y(z).Q') \mid R \xrightarrow{\tau} (y)(P' \mid Q'\{x/z\}) \mid R$$

# Example: scope intrusion

P has a link x to R and wishes to pass x along its link y to Q. Q is willing to receive it, but already possesses a private link x to S; the latter must be renamed to avoid confusion. P intrudes the scope of the private link x between Q and S.

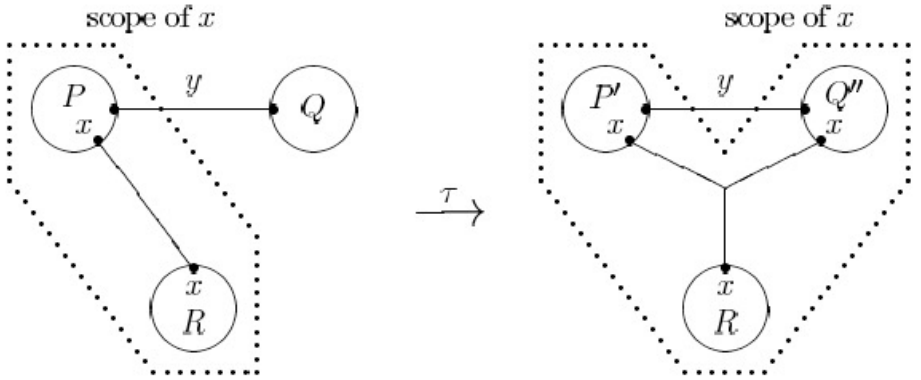



Alpha conversion, i.e., change of bound names is needed

# Example: scope extrusion

P has a link x to R, but we now suppose that this link is private. However, P wishes to pass x along its link y to Q. Q is willing to receive it and possesses no x-link.

Similar to call **by reference** and not by value

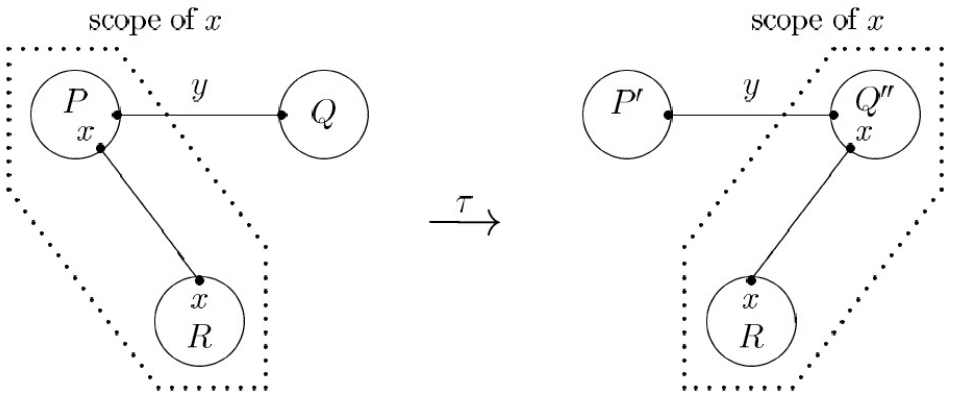



$$(x)(\overline{y}x.P' \mid R) \mid y(z).Q' \xrightarrow{\tau} (x)(P' \mid R \mid Q'\{x/z\})$$

When this link is exported to Q the scope of the restriction is extended, we say that P extrudes the scope of the private x-link. If Q possesses a public x-link than the extruded name must be renamed

# Example: scope extrusion

If P' does not possesses the private x-link after the transition, we have a migration of scope




$$(x)(\overline{y}x.P' \mid R) \mid y(z).Q' \xrightarrow{\tau} (x)(P' \mid R \mid Q'\{x/z\})$$

since $(x)(P_1 \mid P_2) = P_1 \mid (x)P_2$ if $x \notin \mathrm{fn}(P_1)$

we have $(x)(P' \mid R \mid Q'\{x/z\}) = P' \mid (x)(R \mid Q'\{x/z\})$

# Outline

- Motivation
- Key notions
- CCS
- From CCS to Pi calculus
- Pi calculus: syntax and semantics
- **The role of restriction for security**
- Modeling protocols in Pi calculus
- Spi Calculus: a hint
- Conclusions and what next

# Role of restriction

- The restriction operator (ν x)P makes a fresh channel x for use within process P. Hence, restriction governs the visibility of names
- Nobody can monitor a restricted channel, outside the scope
- Scoping is the basis of security

# Restriction against intrusion

- The use of restricted channels can prevent intrusions*

**Naive Campaigning** Vota Antonio

$$
\begin{aligned}
\mathit{Speaker} &\triangleq \overline{air}\langle \mathit{vota\ antonio} \rangle \\
\mathit{Microphone} &\triangleq air(x).\overline{wire}\langle x \rangle \\
\mathit{Loudspeaker} &\triangleq wire(y).\overline{highvolume}\langle y \rangle \\
\mathit{Ad} &\triangleq \mathit{Speaker} \mid \mathit{Microphone} \mid \mathit{Loudspeaker}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Ad} &\longmapsto \overline{wire}\langle \mathit{vota\ antonio} \rangle \mid \mathit{Loudspeaker} \\
&\longmapsto \overline{highvolume}\langle \mathit{vota\ antonio} \rangle
\end{aligned}
$$

* [R. De Nicola, R.Bruni]

# Restriction against intrusion (cont.)

- The use of restricted channels can prevent intrusions*

**Naive Campaigning** Vota Antonio

$$
\begin{aligned}
Speaker &\triangleq \overline{air}\langle vota\ antonio\rangle \\
Microphone &\triangleq air(x).\overline{wire}\langle x\rangle \\
Loudspeaker &\triangleq wire(y).\overline{highvolume}\langle y\rangle \\
Ad &\triangleq Speaker \mid Microphone \mid Loudspeaker
\end{aligned}
$$

$$Rival \triangleq wire(z).\overline{wire}\langle vota\ ciccio\rangle$$

$$
\begin{aligned}
Ad \mid Rival &\longmapsto \overline{wire}\langle vota\ antonio\rangle \mid Loudspeaker \mid Rival \\
&\longmapsto Loudspeaker \mid \overline{wire}\langle vota\ ciccio\rangle \\
&\longmapsto \overline{highvolume}\langle vota\ ciccio\rangle
\end{aligned}
$$

# Restriction against intrusion (cont.)

- The use of restricted channels can prevent intrusions*

**Naive Campaigning** — Vota Antonio

$$\begin{aligned} Speaker &\triangleq \overline{air}\langle vota\ antonio\rangle \\ Microphone &\triangleq air(x).\overline{wire}\langle x\rangle \\ Loudspeaker &\triangleq wire(y).\overline{highvolume}\langle y\rangle \\ Ad &\triangleq Speaker \mid Microphone \mid Loudspeaker \end{aligned}$$

$$Rival \triangleq wire(z).\overline{wire}\langle vota\ ciccio\rangle$$

$$\begin{aligned} Ad \mid Rival &\longmapsto \overline{wire}\langle vota\ antonio\rangle \mid Loudspeaker \mid Rival \\ &\longmapsto Loudspeaker \mid \overline{wire}\langle vota\ ciccio\rangle \\ &\longmapsto \overline{highvolume}\langle vota\ ciccio\rangle \end{aligned}$$

Vota Antonio: secure campaigning

$$SecureAd \triangleq (\nu\ air, wire)(Speaker \mid Microphone \mid Loudspeaker)$$

# RIEPILOGO

dunque allora la volta scorsa abbiamo visto un pochino più da vicino il PI calculus che abbiamo capito che è un'evoluzione del precedente ccs sempre di miller e abbiamo capito che il PI calculus ha come forma diciamo caratterizzante, la dimensione della mobilità, cioè la possibilità di cambiare l'interfaccia di comunicazione dei processi che nel tempo possono acquisire nuove capacità di comunicazione grazie alla ricezione dei nomi di nuovi canali sui quali comunicare quindi vista proprio graficamente un po come hai visto volta scorsa, abbiamo la possibilità di vedere un canale che prima era condiviso, per esempio tra pr arrivare nella nella disponibilità di Q attraverso proprio uno scambio e una sincronizzazione in cui vengono scambiati valori, ma a differenza del ccs, come sottolineavamo la volta scorsa  questi sono i valori che non rappresentano dati, ma anche canali quindi detto in termini formali adesso che abbiamo visto anche la sintassi, abbiamo parlato di soggetti oggetto delle azioni di comunicazione, il tipo di oggetto scambiato con questa forma di mobilità può diventare nella continuazione soggetto oppure oggetto di successive comunicazioni, quindi in particolare rispetto al ccs, può diventare soggetto, quindi si possono scambiare i canali e questo è proprio la visualizzazione del fatto che la rete di canali cambia quindi la la rete di capability di un processo cambia con il tempo perché c'è uno spostamento e la possibilità di passare link da un processo all'altro, proprio come se fosse per riferimento. Abbiamo visto quindi che il PI calculus ci aiuta a modellare i sistemi concorrenti perché ci fornisce tutte le primitive che più o meno servono per rappresentare in maniera astratta il tipo di interazioni ci sono primitive di comunicazione, c'è il passaggio di messaggi, c'è anche quest'altra caratteristica la creazione dinamica di nomi e canali che poi vedremo sarà utile per modellare protocolli di sicurezza e quindi la mobilità di cui abbiamo parlato. Abbiamo parlato di alpha conversione nei termini della possibilità di ridenominare i nomi in maniera diciamo coerente, per evitare possibili confusioni tra oggetti che hanno lo stesso nome, pur rappresentando entità diverse. Abbiamo visto che in particolare ci sono due forme di binding che offre il calculus, ovvero di legame e quindi ha a che fare con lo scope delle variabili, uno di questi è l'input e naturalmente una volta che su x si riceve un certo valore y cioè il valore attuale per y dentro P, ogni volta che ho un occorrenza di y dovrei andare a sostituire con quello che ho ricevuto e quindi diciamo che lo scope di y in questo caso è la continuazione P, mentre nel caso della restrizione di nuovo ho un nome y che lega le occorrenze di y nel processo P in cui è ristretto ma y in questo caso non è qualcosa che ricevo dall'esterno ma qualcosa che creo attraverso la restrizione naturalmente ciò che non è bound e free, cioè ciò che non è legato è libero e poi vedremo che appunto useremo processi up to alpha conversione nel senso che ci sentiremo liberi di ridenominare i nomi bound con nomi freschi che si intende essere nomi non usati per lo meno fino a quel momento. Abbiamo visto qual è il problema della confusione dei nomi e quindi abbiamo visto la semantica con le varie azioni qui l'unica cosa che uno non si aspetta in prima battuta è avere una azione che non corrisponda ai prefissi che abbiamo visto, per cui i prefissi saranno l'azione interna o silenziosa, l' output e l' input che qui assumono anche questo aggettivo di libera in contrapposizione alla quarta azione etichetta delle possibili transazioni dei processi che è quella del bound output che è un modo per ricordare che quello che stiamo immettendo sul canale x non è un nome libero, ma è un nome privato y quindi un nome che appare con il secondo tipo di binding, cioè nella restrizione e quindi questa è la ulteriore forma di output che poi ci consentirà di fare quello che viene chiamata estrusione, cioè il passaggio di un nome da un scope privato di P all' inclusione in questo scope anche del processo che riceverà il nome y. La semantica quindi viene data sempre con regole tipo sos, guidate dalla sintassi e vedete subito che già nell input mi dà la possibilità di rispondere anche a voce alla domanda che mi è arrivata per mail da giuseppe e come vedete l'input in qualche maniera indovina il nome letto w e in particolare usa al posto di y, una variabile w che non appartiene ai nomi liberi e quindi è fresca. Abbiamo visto che di nuovo le side condition regolano sostanzialmente il traffico, cioè cercano di evitare la collisione dei nomi e quindi uno si deve assicurare che ciò che viene spedito sostanzialmente nella comunicazione dentro P non vada a collidere con ciò che è conosciuto perché è libero all'interno di Q. La parte più delicata della semantica, quella un po più nuove, quella, insomma con un più più di technicalities, come dicono gli anglosassoni,è quella della comunicazi

# Modelling protocols

*Security protocols are 3-line programs that people still manage to get wrong.*
**Roger M. Needham**

**System Model**

- The network is modeled as a collection of shared states between the principals involved in the protocol
- Each specific network message details a particular network state setting

**The threat model: Dolev-Yao**

- eavesdropping on any network message
- removing messages from the network
- sending network messages containing new or eavesdropped content to any legitimate party
- attackers can't compromise cryptographic protections

# Modelling protocols in pi calculus

Quindi modellare protocolli in PI Calcolo è un primo passo per vedere bene questo tipo di situazioni ho la possibilità di modellare comunicazione attraverso le primitive ho delle regole di scoping attraverso la restrizione che mi consentono di regolare l'accesso ai canali e ai dati, ho la scope extrusion che mi modella proprio lo scambio di risorse private, quindi, in particolare se ci pensate io creo un nuovo canale privato e lo passo a chi di dovere, quindi posso gestire anche le triangolazioni delle terze entità e naturalmente l'idea è che uno usi in combinazione restrizione e scope extrusion e posso modellare attraverso la replicazione le sessioni che non sono limitate, quindi il numero di sessioni che voglio e l'unico difetto è che il PI calcolo come avete visto non ha primitive per la crittografia, quindi quello che vedremo è un primo modo di modellare protocolli in cui trasformiamo il problema delle primitive crittografiche nel problema di gestire canali privati, quindi è un modo di traslare invece delle chiavi private avrò dei canali privati.

- Pi calculus is convenient for modelling security protocols at an **abstract level**. Protocols are processes
- Communication primitives: simple and powerful.
- Scoping Rules: explicitly control the access to channels and to data.
- Scope extrusions: models exchange of private resources

  To know the name of a channel amounts to having the capability to communicate on it.
- **Idea**: using restriction + scope extrusion for modelling possession and communication of secrets (such as keys)
- Replication is used to model unbounded sessions
- Note that the Pi calculus does **not** include cryptographic primitives

# Modelling protocols in pi calculus (cont.)

**Steps**:

- define the term algebra often, the free algebra
- define the protocol participants
- allow for an unbound number of parallel sessions (use replication)
- define the adversary
- put everything in parallel

Quindi quello che succede è che modellerò un protocollo di sicurezza immaginandolo come un sistema di processi in parallelo che magari possono ripetere il loro comportamento definirò l'avversario e metterò appunto in parallelo i vari elementi

Per semplificare un po questo concetto parto da un famoso esempio di protocollo giocattolo che si chiama rana dalla bocca larga, in cui visto in maniera crittografica, abbiamo tre processi A S e B che vogliono comunicare tra di loro. A per comunicare con B si serve della intermediazione di un server di autenticazione  S considerato trusted. Nell'ambito dei protocolli di sicurezza ci sono più gradi, cioè un gerarchia di chiavi, in particolare, esistono le chiavi di lungo termine, long term e short term, e l'idea è che le chiavi long term siano più sicure quindi anche più dispendiose dal punto di vista sia del tempo che delle energie e vengono mantenuti a lungo, quindi devono essere meno attaccabili e queste sovrintendono lo scambio di quelle che si chiamano chiave di sessione che invece hanno durata breve e vengono rinnovate spesso che possono essere anche meno sicure di quelle long term perché vengono usate in una finestra temporale più piccola. Quindi quello che succede in questo particolare protocollo è che A ed S condividono una chiave long term C_AS stessa cosa tra S e B, quindi per stabilire una chiave di sessione tra A e B, A non può che passare da S quindi l'idea è che crei una chiave di sessione A e la passi al server dicendogli: voglio condividerla con B. Lo schema nella notazione informale che viene usata si chiama notazione Alice-Bob, S fa da tramite e fa arrivare la chiave di sessione pensata da A a B dicendogli: guarda è quella che ti ha mandato A, questo nel terzo passo ad A di spedire un messaggio M protetto della chiave di sessione K_AB. Il PI calcolo questa possibilità non ce la dà e quindi vedremo tutto rimodellato usando canali privati al posto di chiavi, quindi quello che verrà scambiato con S è il nome del canale privato, chiave di sessione C_AB che verrà rispedito ad S in modo che su questo canale che alla fine B conoscerà, A potrà spedirgli il suo messaggio segreto M.

# hannel Establishment: Wide Mouthed Frog*

stablishment/use of a secure channel:

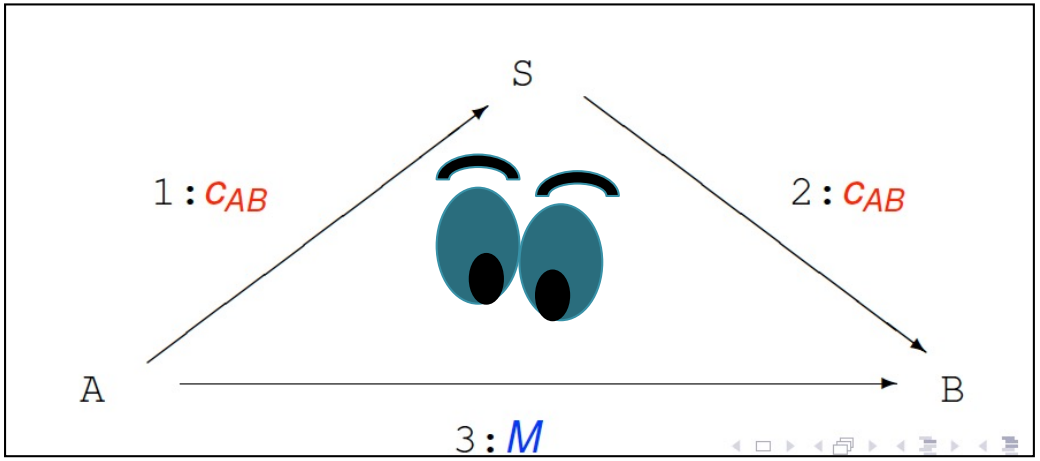
A and B: two clients

S: an authentication server

$c_{AS}$ and $c_{SB}$: channels with the server

$c_{AB}$: a new channel for the clients

1. $A \rightarrow S: \{K_{AB}, B\}_{KAS}$,
2. $S \rightarrow B: \{K_{AB}, A\}_{KBS}$
3. $A \rightarrow B: \{M\}_{KAB}$

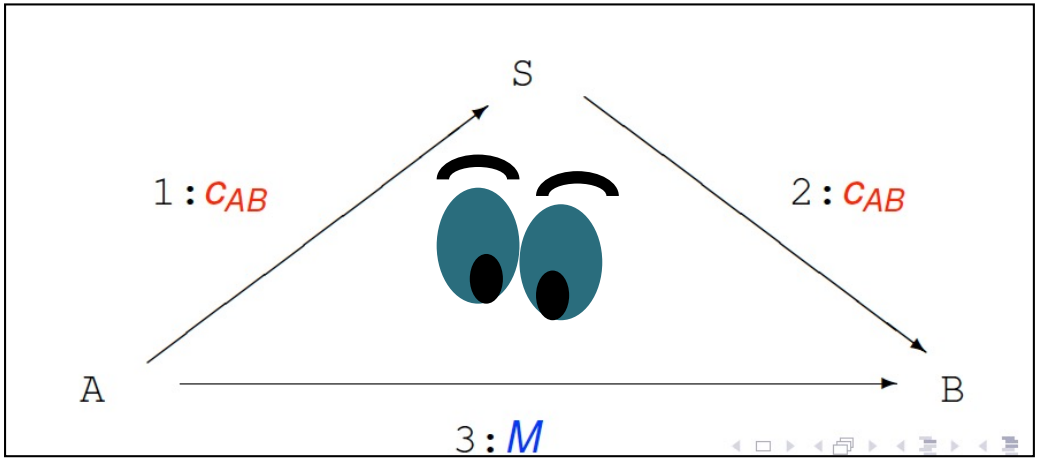



* See BAN paper

# Channel Establishment: Wide Mouthed Frog in pi calculus

Note the use of scope extrusion

$$
\begin{aligned}
P &= (\nu c_{AS})(\nu c_{BS})(\,(A|B)\,|S) \\
A &= (\nu c_{AB})(\overline{c_{AS}}\langle c_{AB}\rangle.\overline{c_{AB}}\langle M\rangle) \\
S &= c_{AS}(x).\overline{c_{BS}}\langle x\rangle \\
B &= c_{BS}(w).w(y)
\end{aligned}
$$



# Outline

- Motivation
- Key notions
- CCS
- From CCS to Pi calculus
- Pi calculus: syntax and semantics
- The role of restriction for security
- Modeling protocols in Pi calculus
- **Spi Calculus: a hint**
- Conclusions and what next

# Modelling protocols in Spi calculus

**Spi Calculus** [Abadi, Gordon] extends the Pi calculus with

- terms (pairs, encryption)
- **cryptographic primitives**

Spi calculus

- is directly executable
- has formal semantics

In Spi calculus, keys too can be dynamically created and communicated

Il limite del PI calcolo è sostanzialmente che non ha primitive di crittografia per ovviare a questo è stato introdotto un certo numero di calcoli che invece utilizzavano questo tipo di modellazione, in particolare lo SPI calculus che estendeva il PI calcolo con primitive di crittografia, in particolare encryption e decryption, e anche dando maggiore struttura ai termini quindi, supponendo che fossero tuple, coppie o comunque inserendo l'encryption. Aveva anche il vantaggio di avere una semantica formale, ma essere anche eseguibile in qualche modo e riprendeva in tutto e per tutto il PI calcolo perché le chiavi potevano essere create dinamicamente e poi utilizzate nel contesto delle encryption e delle decryption. Il PI calculus, così come l'abbiamo introdotto come vedete, è in forma cosiddetta monadica, mi sto limitando negli input e negli output a scambiare un solo singolo messaggio, vi potete immaginare che non debba essere così complicato a pensare a una versione poliadica dove invece di scambiarsi un messaggio si possono scambiare tuple di messaggi.

# Modelling protocols in Spi calculus

M,N ::= *terms*

n | x | 0 | succ(M)

| (M,N) pair

| $\{M\}_N$ [symmetric encryption]

P,Q ::= *processes*

$M\langle N_1,\ldots,N_k\rangle.P \mid M(x_1,\ldots,x_k).P \mid \ldots \mid$

| decrypt L as $\{x_1,\ldots,x_k\}_N$ in P [symmetric decryption]

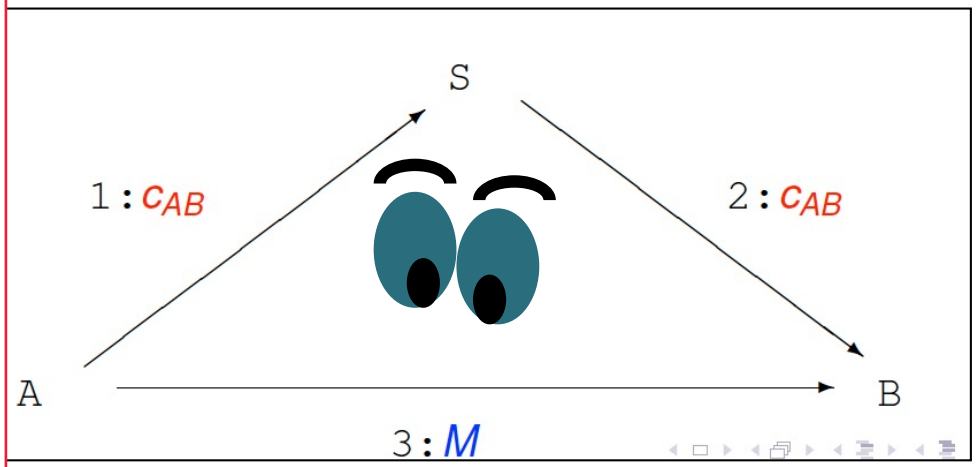
[or case L of $\{x_1,\ldots,x_k\}_N$ in P]

if L is $\{x_1,\ldots,x_k\}_N$ for some x then run P

- decrypt $\{M\}_N$ as $\{x\}_N$ in P becomes P[M/x] if the decryption succeeds
- Keys can be dynamically created and communicated

questa quindi è la versione crittografica della rana dalla bocca larga c'è un'altro modo per scrivere decryption che è "case", quindi quello che succede è che il processo A come prima crea un nuovo nome, in questo caso non è un canale, ma è una chiave che poi posso usare successivamente e lo manda all'interno al server, lo manda in un encryption che confeziona utilizzando proprio la chiave long term che condivide con il server. Dopo che tutto è andato a buon fine, sarà in grado di comunicare proprio sul canale che invece condivide con B il messaggio che vuole spedire a B protetto con la chiave di sessione che ha appena ideato. Dal canto suo, il server riceve l'encryption, quindi su x verrà sostituita un termine, come dicevamo prima, un termine L che sostanzialmente è il messaggio cifrato e proverà a vedere se questa x corrisponde a un encryption fatta con quella chiave che lui si aspetta, che è quella che condivide con il server e si aspetta che dentro trovi un nome di chiave e in questo caso se tutto va a buon fine, sarà il caso che S spedisca quello che ha ricevuto, ricodificandolo questa volta con la chiave che condivide con B e quindi di rimbalzo B riceverà questa chiave codificata, la dovrà decodificare e qui si aspetta di doverlo fare con la chiave che condivide con S per poi utilizzarla nel proseguio della comunicazione per decifrare ciò che gli arriva da A. Ciò che gli arriva da A sara M codificato con K_AB quindi se tutto va per il verso giusto, lui proverà a decodificarlo, vedete qui y a tempo di esecuzione, quando si arriva a fare questo case (in rosso) sarà istanziato esattamente con la chiave di sessione K_AB che B riceve dal server e quindi potrà proseguire come vuole, possedendo il messaggio che voleva ottenere.

# in Spi calculus




## Cryptographic implementation of the Pi calculus version

$$
\begin{aligned}
A(M) &\triangleq (\nu K_{AB})(\overline{c_{AS}}\langle\{K_{AB}\}_{K_{AS}}\rangle.\overline{c_{AB}}\langle\{M\}_{K_{AB}}\rangle) \\
S &\triangleq c_{AS}(x).case\ x\ of\ \{y\}_{K_{AS}}\ in\ \overline{c_{SB}}\langle\{y\}_{K_{SB}}\rangle \\
B &\triangleq c_{SB}(x).case\ x\ of\ \{y\}_{K_{SB}}\ in \\
&\quad\ c_{AB}(z).case\ z\ of\ \{w\}_y\ in\ F(w)
\end{aligned}
$$

# Outline

- Motivation
- Key notions
- CCS
- From CCS to Pi calculus
- Pi calculus: syntax and semantics
- The role of restriction for security
- Modeling protocols in Pi calculus
- Spi Calculus: a hint
- **Conclusions and what next**

# Conclusions and what next

- Process algebras: Pi calculus is an evolution of CCS
- Pi calculus and its derived languages (like Spi calculus) can be used to abstractly model security protocols

## Next

- We will see a static analysis technique applied to crypto-protocols modelled with process algebras to reason about security properties.

# Bibliography


- R. Milner. *Communication and Concurrency*, Prentice Hall
- R. Milner. *Communicating and Mobile Systems: the pi-calculus*, CUP
- Joachim Parrow. *An Introduction to the Pi-Calculus*. In Handbook of Process Algebra, ed. Bergstra, Ponse, Smolka, pages 479-543, Elsevier (2001)
- D. Sangiorgi, D. Walker. *The pi-calculus, a Theory of Mobile Computation*, CUP.
- M. Abadi, A. D. Gordon. *A Calculus for Cryptographic Protocols: The spi Calculus*. Inf. Comput. 148(1): 1-70 (1999)
- M. Abadi. *Secrecy by Typing in Security Protocols*. TACS 1997: 611-638.
- R. Bruni, U. Montanari. *Models of Computation*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016