

# Multi-Peer GAXML systems

Albert and Loic, discussed with Serge



# Motivations

---

## Distributed multi-peer GAXML systems

- Reuse existing models of GAXML [Serge-Luc-Victor, PODS08]
- Reuse existing concurrent models of systems equipped with a nice notion of parallel composition (assoc and commut)
- Specialize the models in the context of GAXML
- Use this for various purposes: *interfaces*



# Motivations (cont'd)

## Distributed multi-peer GAXML systems

- Concurrent Transition Systems; transition relations relating pre- and post-variables
- Central tool: relative patterns, which play the role of (local) variables of the CTS
- Queries play the role of Transition Relations



# Contents

---

- 1 Concurrent Transition Systems
- 2 From CTS to Multipeer GAXML and Execution Schemes
- 3 Interfaces
- 4 Single Peer model for Mailorder
- 5 MultiPeer model for Mailorder
- 6 Interfaces for Mailorder

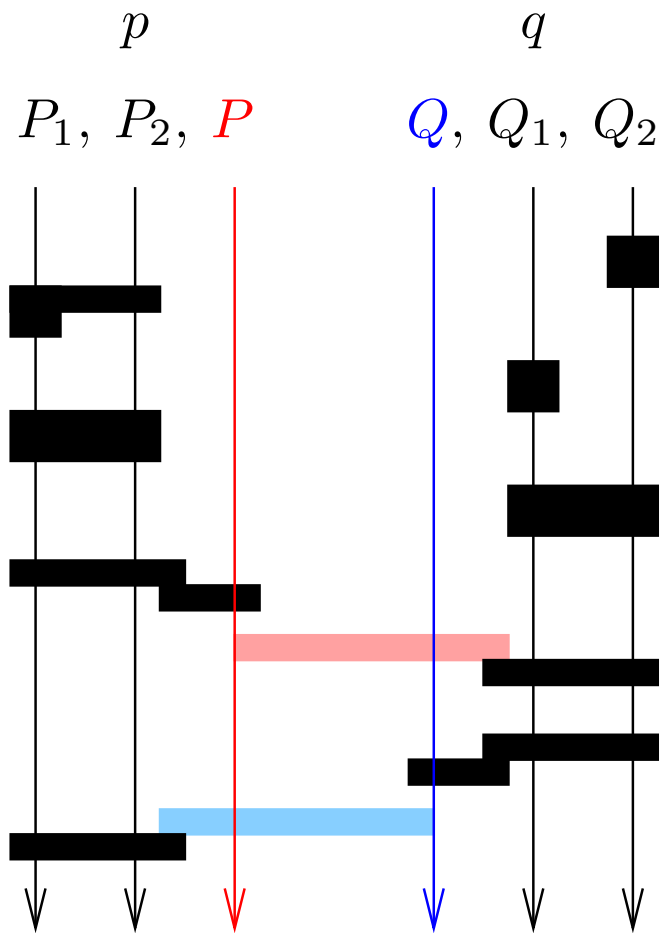
# The Model: CTS

Concurrent Transition System (CTS):

$S = (\mathcal{P}, \mathbf{T}, F_0)$ , where

- $\mathcal{P} \subseteq \mathbb{P}$ : variables; *state*:  $F \in D(\mathcal{P})$
- $\mathbf{T}$ : set of *transitions*  
 $\mathbf{t} = \langle F_-, F_+ \rangle \in D(Q_-) \times D(Q_+)$  where  
 $Q_-, Q_+ \subseteq \mathcal{P}$ : variables *read* and *written* by  $\mathbf{t}$
- $F_0 \in D(\mathcal{P})$ : initial state
- *Firing Sequence*:  $\mathbf{t}(1), \mathbf{t}(2), \dots,$   
 $\mathbf{t}(k) = \langle F_-(k), F_+(k) \rangle$ , such that,  $\forall k \geq 1$ :  
 $F_-(k) = F_{k-1}|_{Q_-(k)}, F_k = F_+(k) \uplus F_{k-1}|_{\mathcal{P} \setminus Q_+(k)}$

# CTS: executions



A concurrent form for the executions, where transitions are only partially ordered by causality.

# CTS: Parallel Composition

CTS compose as follows:

$$S_1 \parallel S_2 = (\mathcal{P}, \mathbf{T}, (F_{1,0}, F_{2,0}))$$

where

$$\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$$

$$\mathbf{T} = \mathbf{T}_1 \cup \mathbf{T}_2$$

$$F_{i,0} = \mathbf{proj}_{\mathcal{P}_i}(F_0)$$

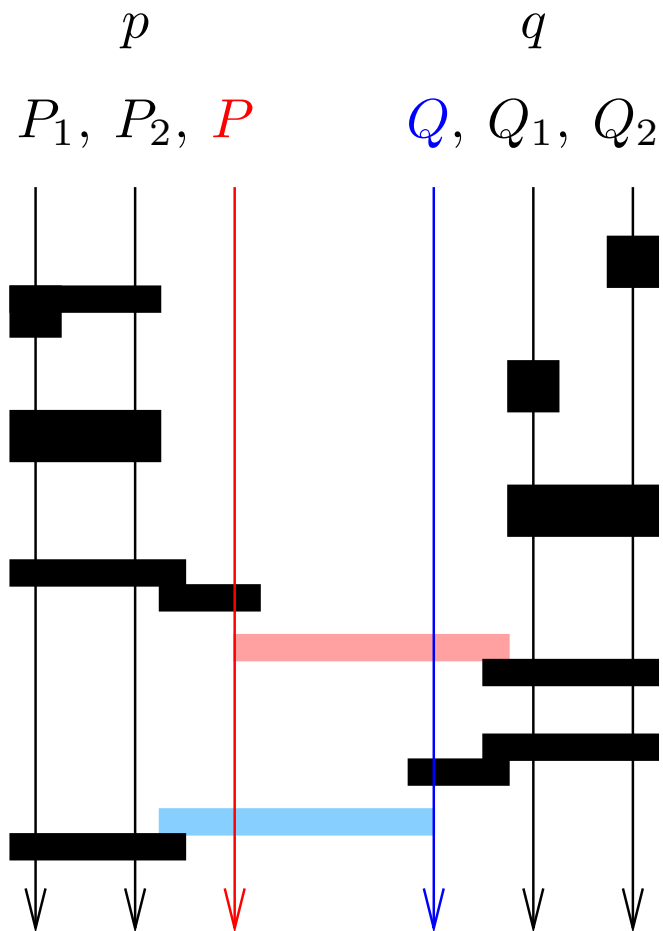
Parallel composition  $\parallel$  is commutative and associative.

# CTS - Specialization

- $\mathbf{T}_1 \cap \mathbf{T}_2 = \emptyset$  (not mandatory but convenient)
- A system has variables it can write (no one else can write them) and variables it can read that are written by other systems
- Transitions can be defined **intensionally** using *transition relations* (constraints between variables and “next” variables)



# CTS - Specialization



Composition of two CTS,  
one per each peer.

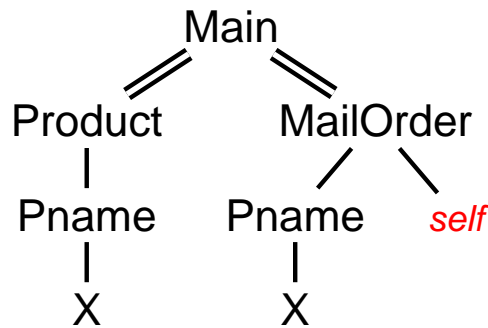
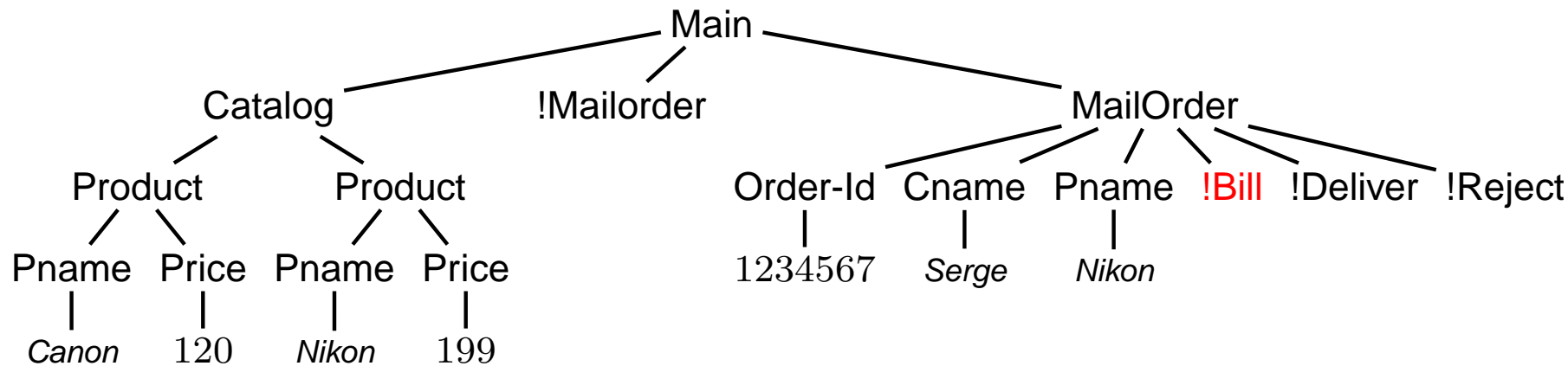


# From CTS to GAXML

Specialize variables, states, and transitions

AXML concept	CTS concept
$\geq 1$ peer	1 CTS
global document	global state
relative pattern	variable
relative query	transition relation

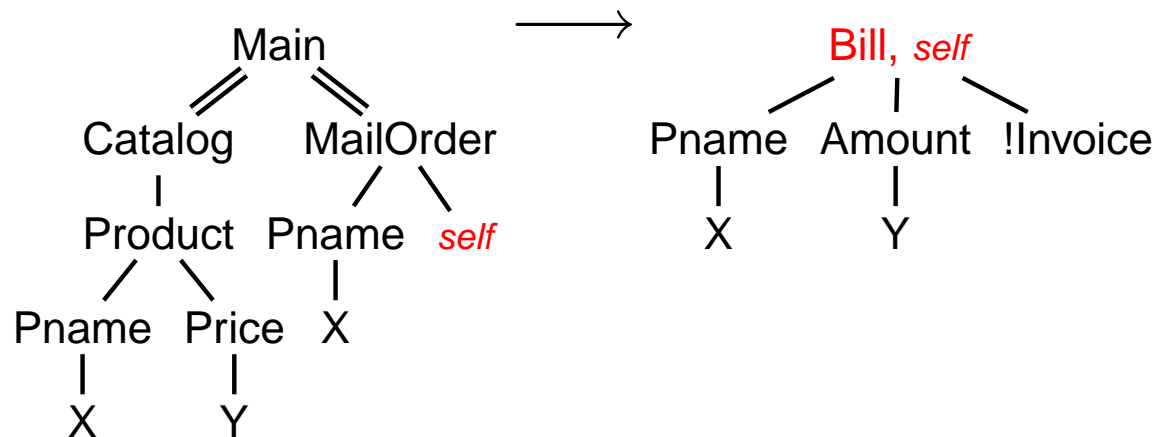
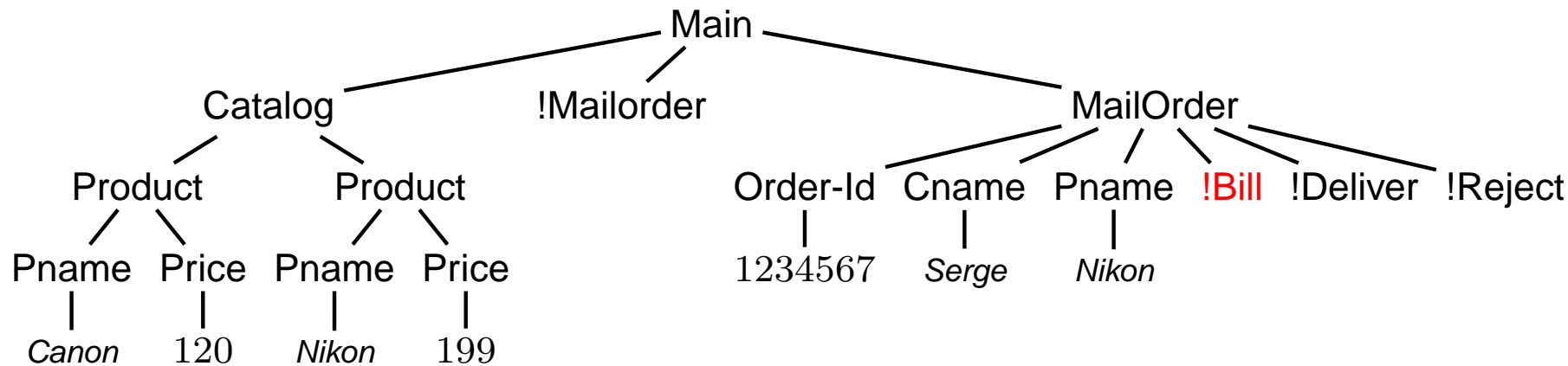
# Relative Patterns/Queries



relative patterns operate on documents  
 by returning the set of all possible matchings  
 ( $\Leftrightarrow$  CTS-variables evaluate on states)

Relative Pattern

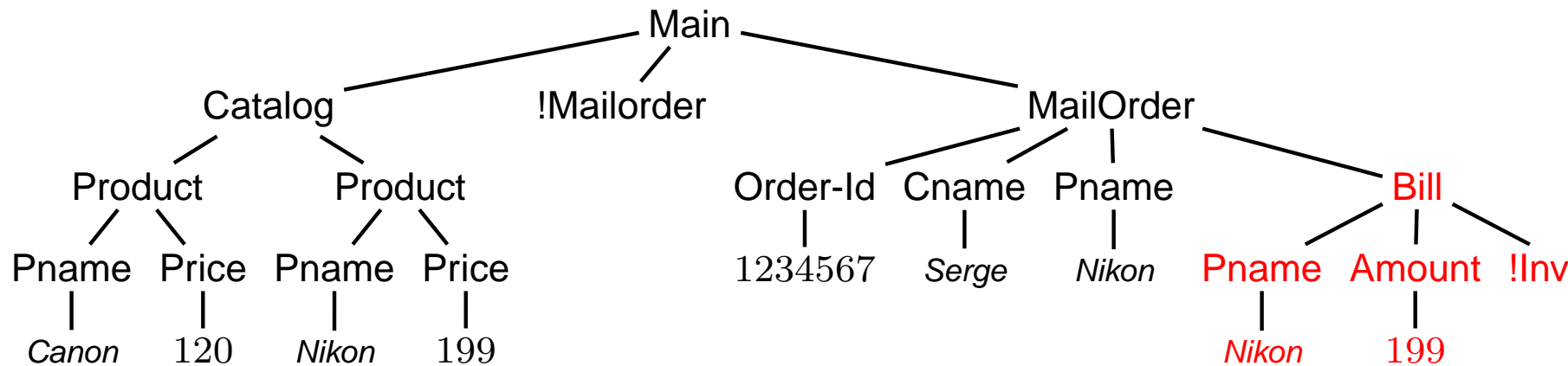
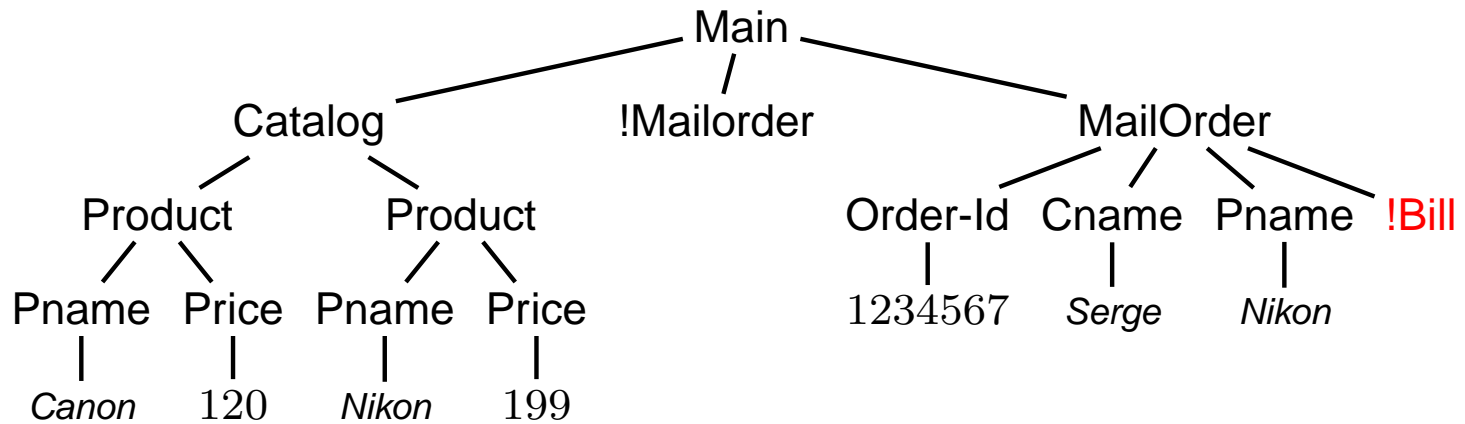
# Relative Patterns/Queries



relative queries  
define  
transition relations

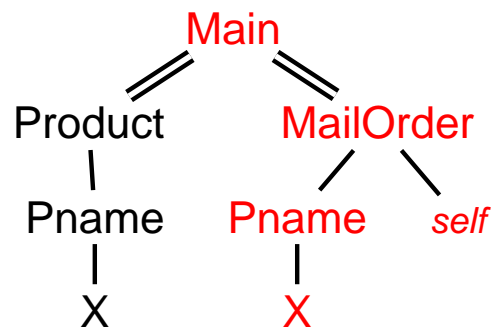
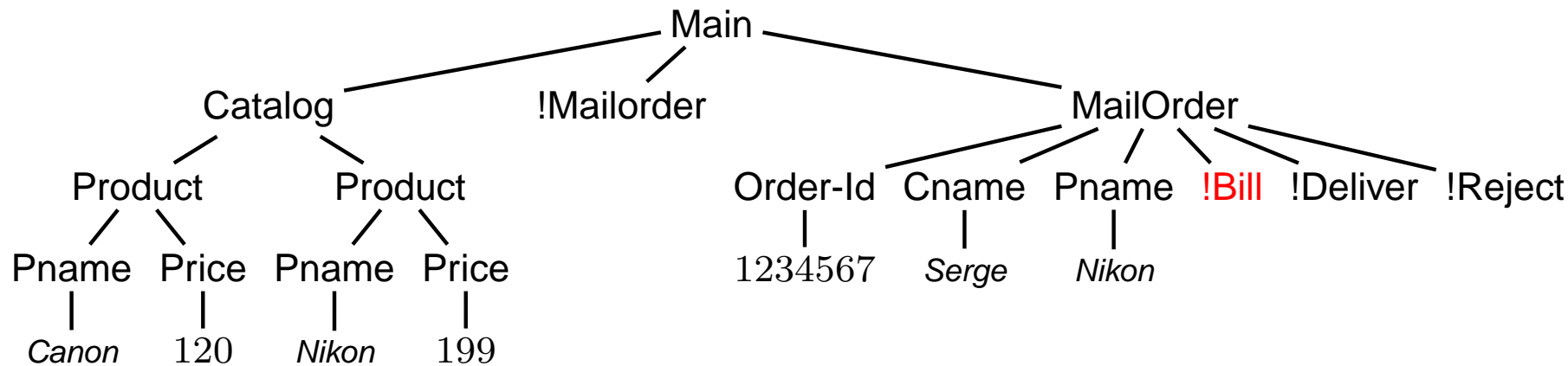
Relative Query

# Relative Patterns/Queries



Relative Queries define Transition Relations

# Relative Patterns/Queries



relative patterns operate on documents  
 by returning the set of all possible matchings  
 ( $\Leftrightarrow$  CTS-variables evaluate on states)  
 relative patterns are not independent

Relative Pattern

# An Order on Relative Patterns

$P' \leq P$  if there exists a total mapping  $\nu$ , from the nodes of  $P'$  to the nodes of  $P$ , such that

- (i) *self* maps to *self*;
- (ii) */* maps to */* and *//* maps to descendants linked by a path with arbitrary branch labels; and
- (iii) the labels of the nodes are preserved

**Lemma** *If  $P' \leq P$  holds, then, for any document  $(F, n)$ , any matching  $\mu'$  of  $P'$  into  $(F, n)$  gives rise to a matching  $\mu = \mu' \circ \nu$  of  $P$  into  $(F, n)$ .*

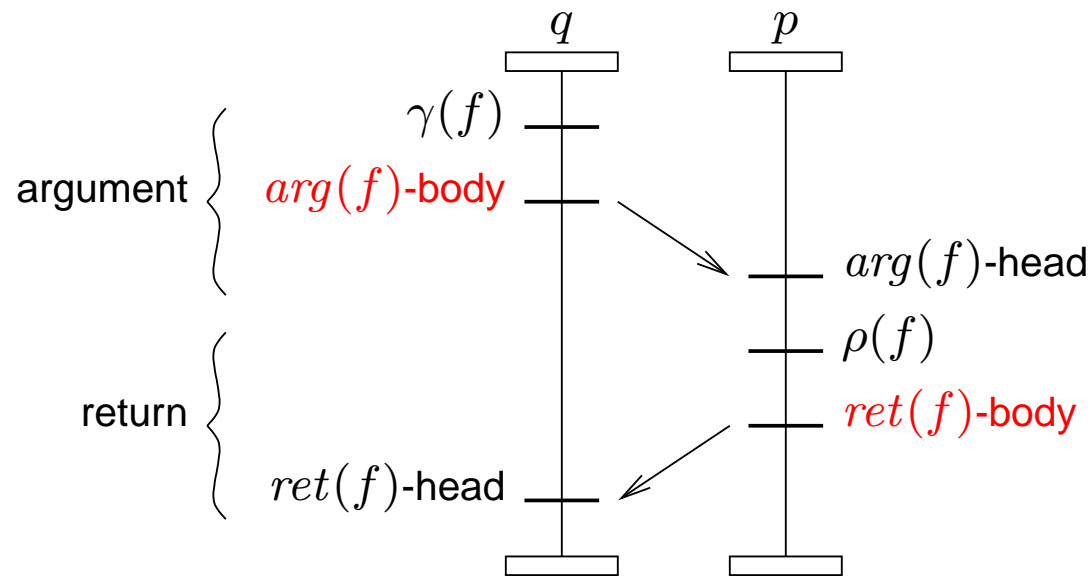


# GAXML seen as CTS

AXML concept	CTS concept
$\geq 1$ peer	1 CTS
global document	global state
relative pattern	variable
relative query	transition relation
function	argument query return query



# Functions



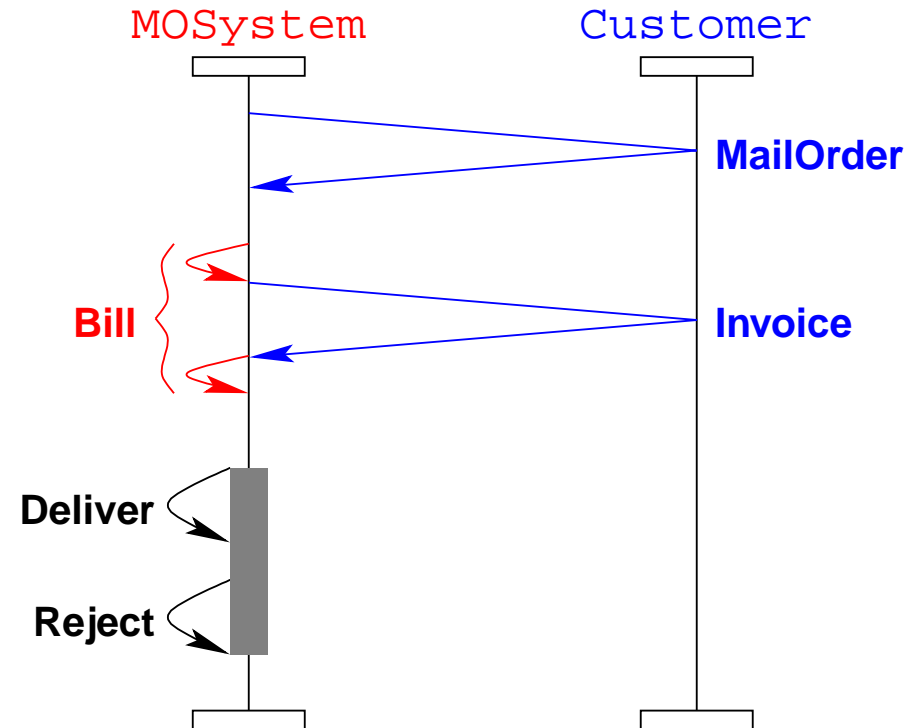
argument: owned by  $p$

return: owned by  $q$

only internal functions are considered; external functions will be replaced by interface information

# Mail Order example

function	owner	caller
<b>MailOrder</b>	Customer	MOSystem
<b>Bill</b>	MOSystem	MOSystem
<b>Invoice</b>	Customer	MOSystem
<b>Deliver</b>	external	MOSystem
<b>Reject</b>	external	MOSystem



See paper for details

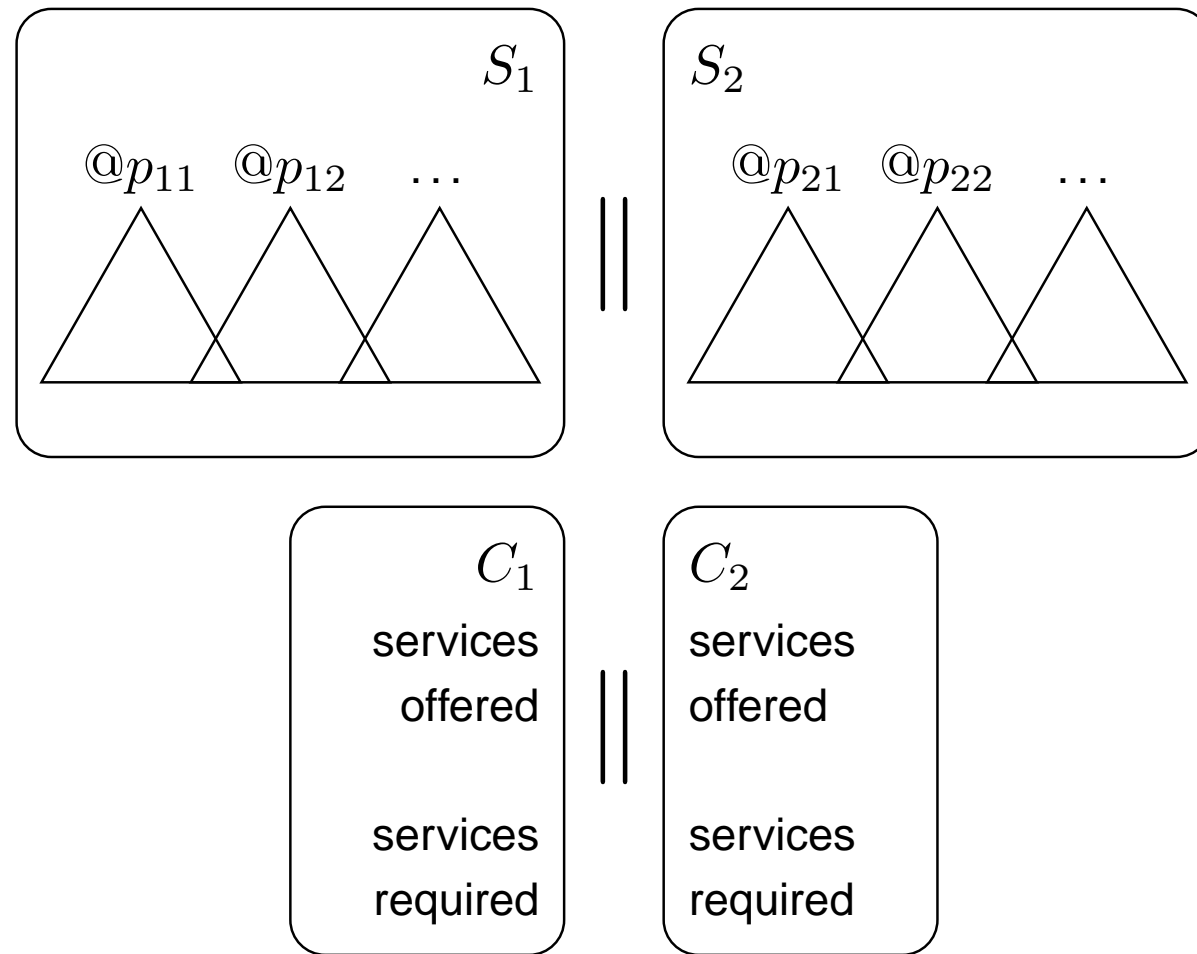


# Interfaces

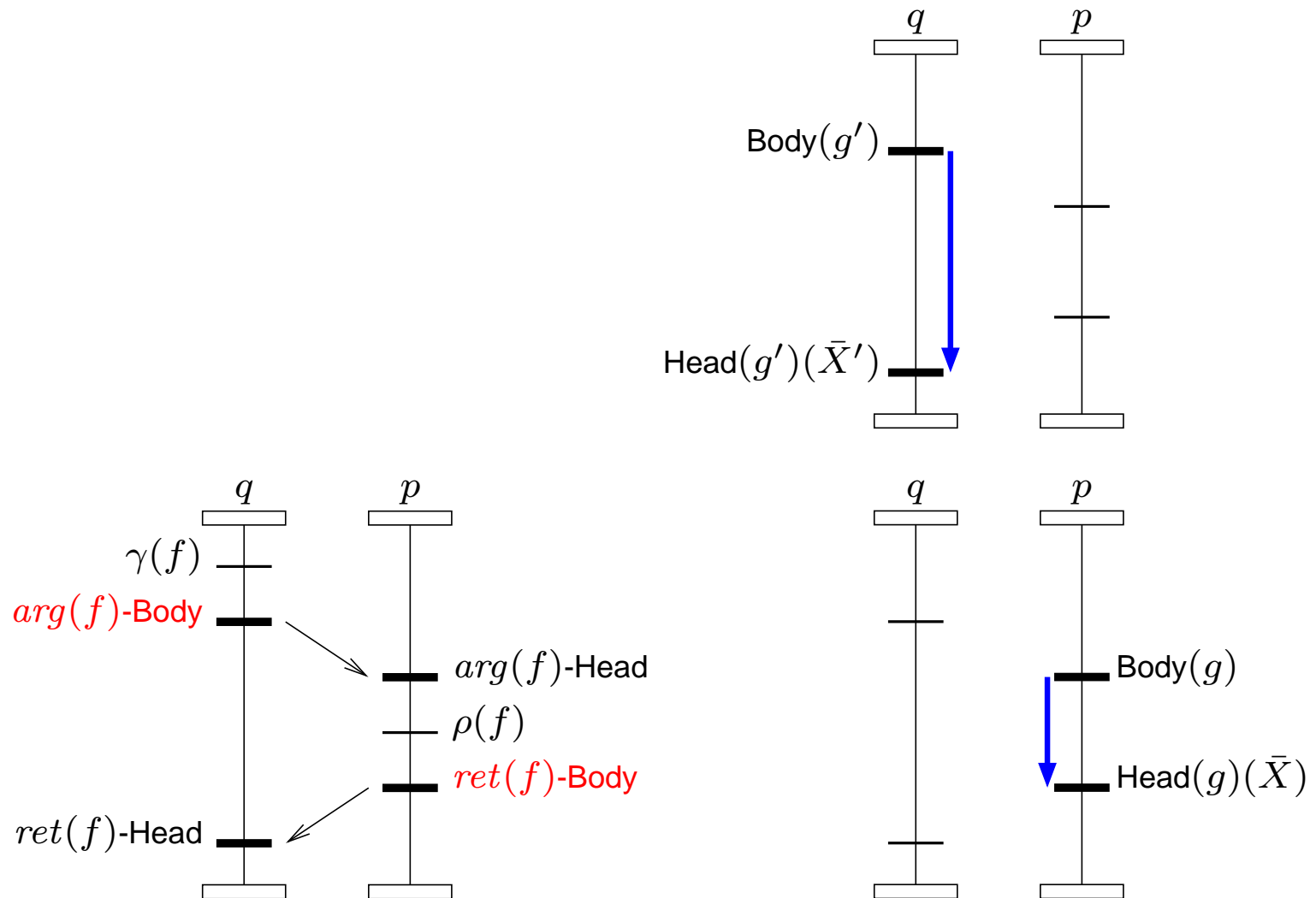
Specifying how a GAXML system can interact with the external world:

- Which properties are expected from the external functions;
- Which properties the considered system offers when called from the external world.

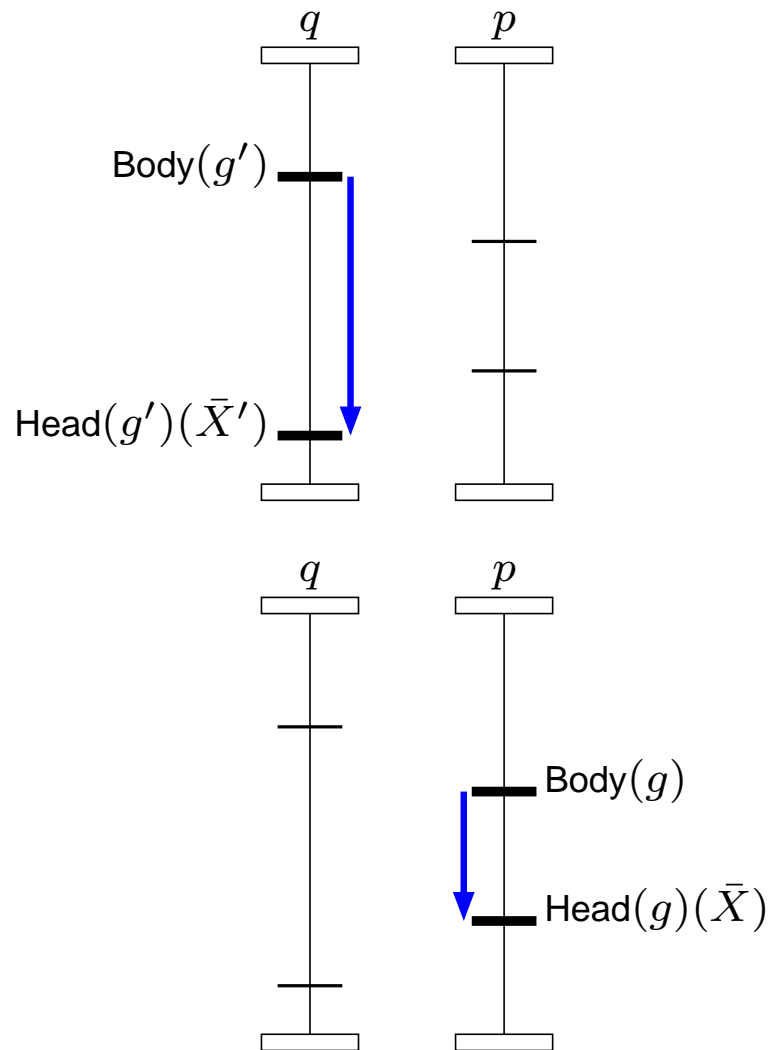
# Interfaces



# Interfaces



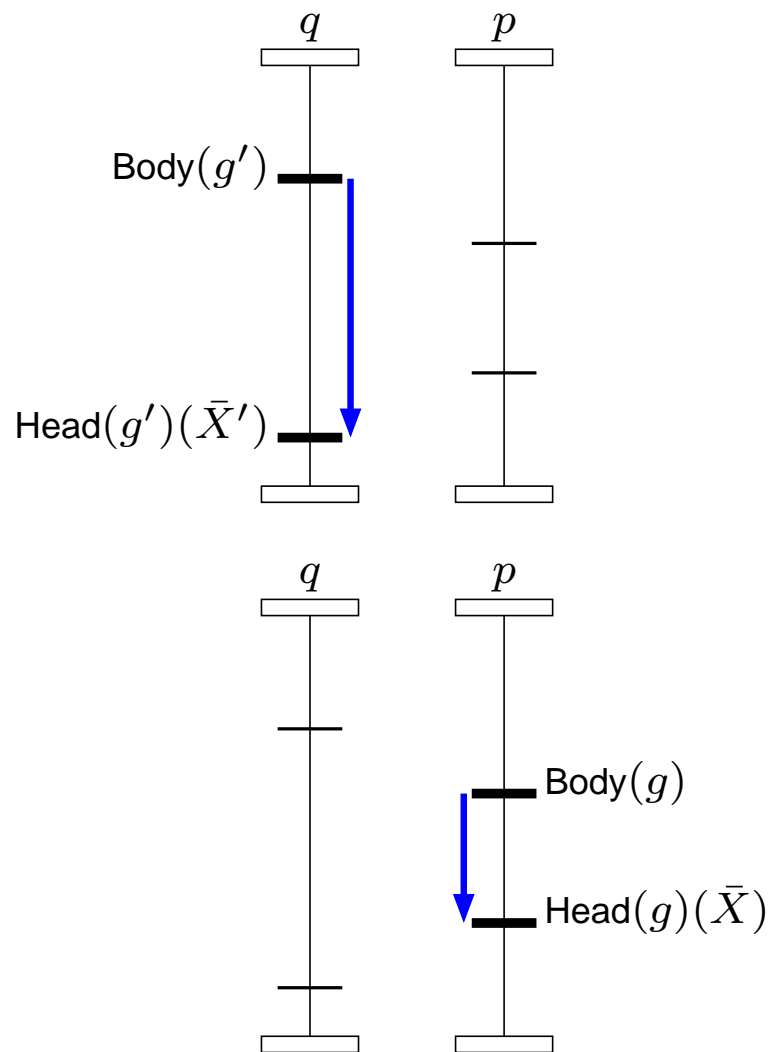
# Interfaces



what the caller  
expects  
from the service

what the callee  
offers

# Interfaces: Qpatterns/Qqueries



interfaces refer to  
variables known  
to the document  
as well as  
variables unknown  
from the document

**Qpatterns & Qqueries**

# Interfaces: Qpatterns/Qqueries

source: Tree-LTL [Serge, Luc, Victor, PODS2008]

**Qpattern:**  $P(\bar{X})$ , where  $P$  is a relative pattern and  $\bar{X}$  is a subset of its variables, designated as *free*. Other variables of  $P$  are seen as quantified existentially, locally to  $P$ . So, formally,  $P(\bar{X})$  means  $\exists \bar{Y}.P$ , where  $\bar{Y}$  is the set of variables of  $P$  not belonging to  $\bar{X}$ . Define

$$P'(\bar{X}') \leq P(\bar{X})$$

by simply ignoring the mention of  $\bar{X}$ .



# Interfaces: Qpatterns/Qqueries

**Qquery:**  $P(\bar{X}) = Body \rightarrow Head(\bar{X})$ , where  $Body$  is a pattern and  $Head(\bar{X})$  is a Qpattern such that,  $\forall H(\bar{X}) \in Head(\bar{X})$ :

- its internal nodes have labels in  $\Sigma$  and its leaves have labels in  $\Sigma \cup \mathcal{F}! \cup \mathcal{V}$ ;
- there is no repeated variable in  $H(\bar{X})$  and each free variable occurring in it also occurs in  $Body$ ; and
- there is one designated node  $c$  in  $H$  called the *constructor* node, such that the subtree rooted at  $c$  contains all variables in  $H$ .

# Implementation relation

$$f \models_{\text{int}} g \text{ iff } \begin{cases} \text{arg-Head}(f) \leq \text{Body}(g) \\ \text{ret-Body}(f) \leq \text{Head}(g)(\bar{X}) \end{cases}$$

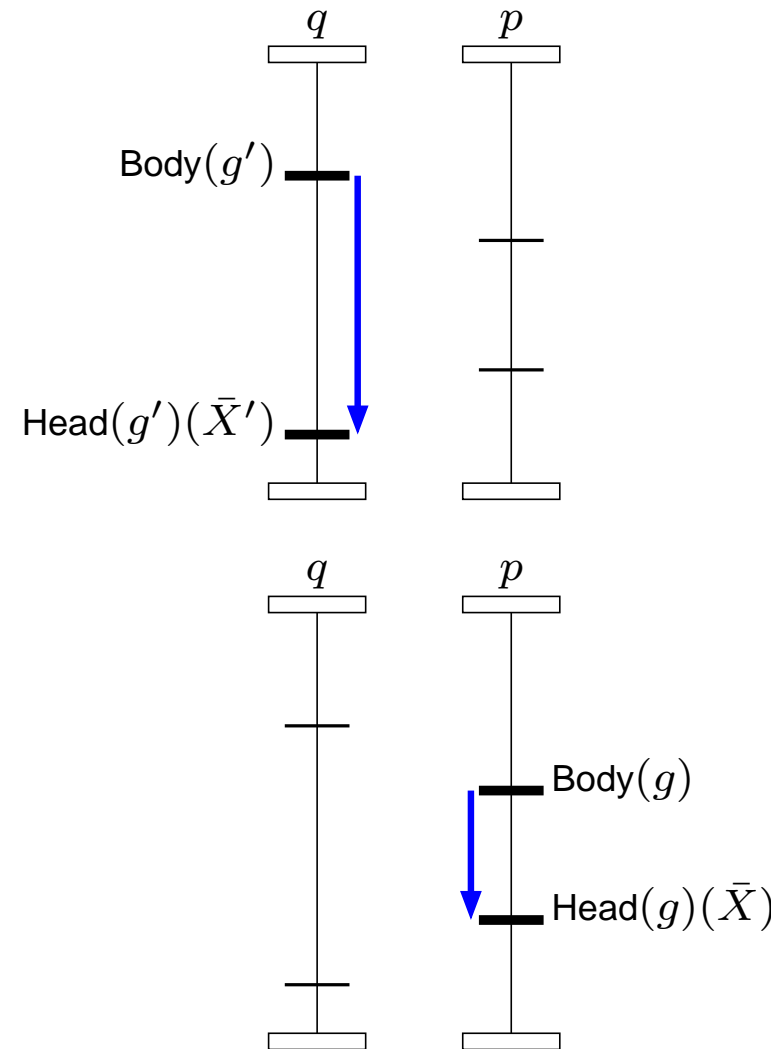
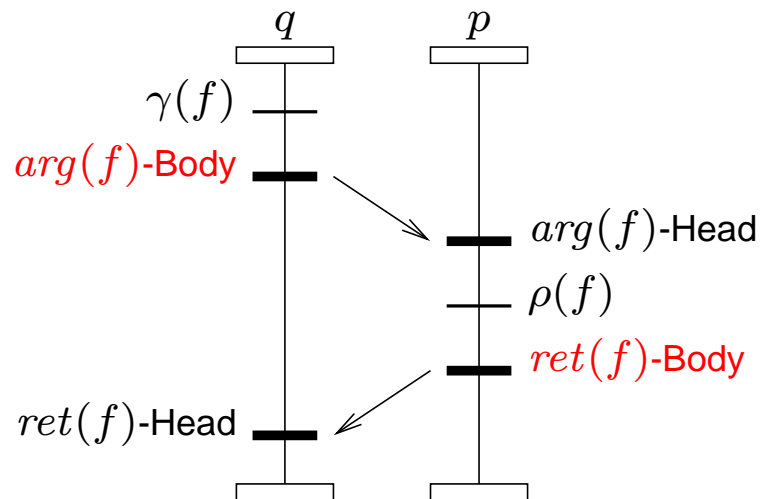
$$f \models_{\text{ext}} g' \text{ iff } \begin{cases} \text{arg-Body}(f) \leq \text{Body}(g') \\ \text{ret-Head}(f) \leq \text{Head}(g')(\bar{X}) \end{cases}$$

$$f \models (g, g') \text{ iff } f \models_{\text{int}} g \text{ and } f \models_{\text{ext}} g'$$

# Implementation relation

$$f \models_{\text{int}} g \text{ iff } \begin{cases} \text{arg-Head}(f) \leq \text{Body}(g) \\ \text{ret-Body}(f) \leq \text{Head}(g)(\bar{X}) \end{cases}$$

$$f \models_{\text{ext}} g' \text{ iff } \begin{cases} \text{arg-Body}(f) \leq \text{Body}(g') \\ \text{ret-Head}(f) \leq \text{Head}(g')(\bar{X}') \end{cases}$$



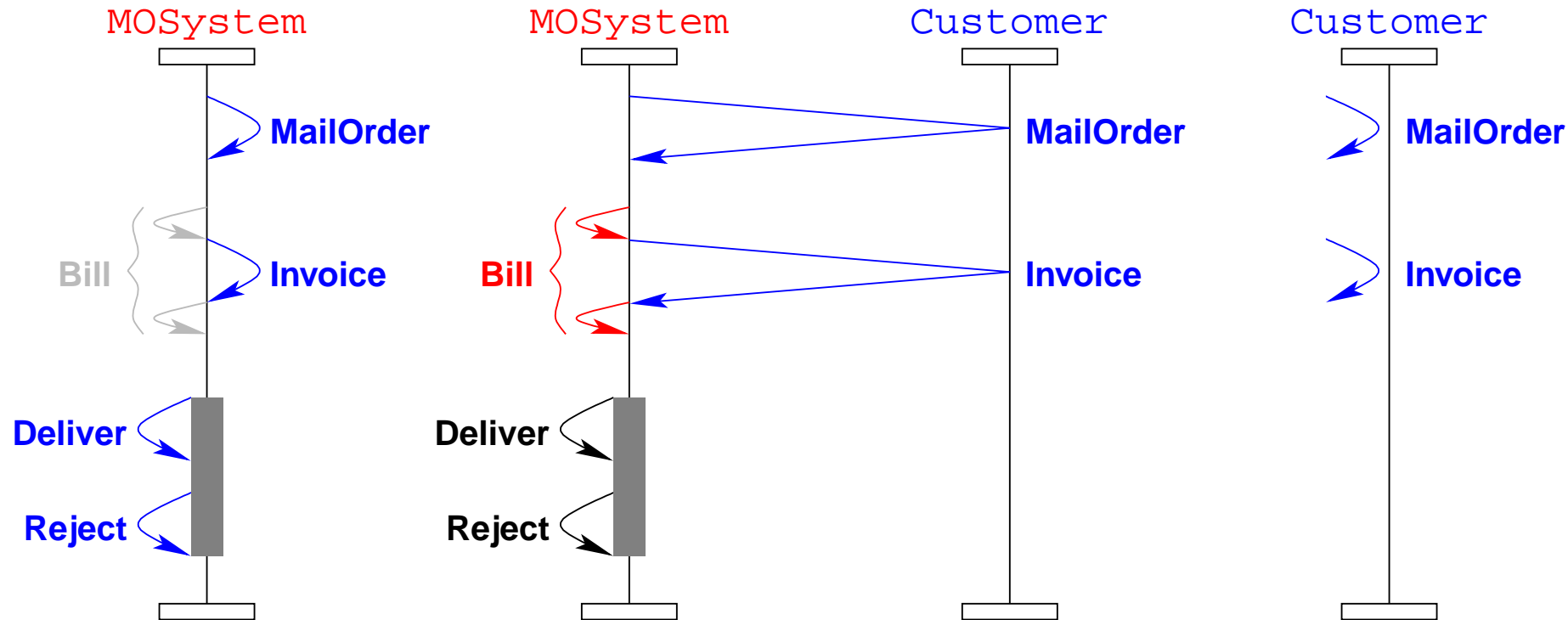


# Implementation relation

For  $C$  an interface and  $S$  a schema, say that  $S$  *implements*  $C$ , written  $S \models C$ , if

1. every external Qfunction of  $C$  either occurs on  $S$  or is implemented by some function that is called but not owned by  $S$ ; and
2. every internal Qfunction of  $C$  that can be called from outside the peer is implemented by some internal function of  $S$ .

# Implementation relation



# Interface compatibility

For  $g, g'$  internal and an external Qfunctions:

$g$  and  $g'$  *compatible* iff  $\exists f : f \models (g, g')$

Compatibility means absence of contradiction:  $g \approx g'$  does not imply that  $g$  and  $g'$  are the two “faces” of a same service. They may indeed correspond to different services, but these two services should be compatible, i.e., should not contradict themselves.

# Interface compatibility

For  $g, g'$  internal and an external Qfunctions:

$g$  and  $g'$  *compatible* iff  $\exists f : f \models (g, g')$

Two interfaces  $C$  and  $C'$  are called *compatible* if, for each pair  $(g, g')$  such that  $g$  is an external Qfunction of  $C$  and  $g'$  is an internal Qfunction of  $C'$ , or vice-versa, then  $g \approx g'$  holds.

# Revisiting composition

Schemas were not given a precise semantics for their external functions. We propose to replace external functions by external Qfunctions:

$$S = (\Phi_{\text{int}}, \Phi_{\text{ext}})$$

$$S_1 \parallel_s S_2 = (\Phi_{\text{int}}, \Phi_{\text{ext}}), \text{ where}$$

$$\Phi_{\text{int}} = \Phi_{\text{int}}^1 \parallel \Phi_{\text{int}}^2, \text{ and}$$

$$\Phi_{\text{ext}} = (\Phi_{\text{ext}}^1 \setminus \Phi_{\text{int}}^2) \cup (\Phi_{\text{ext}}^2 \setminus \Phi_{\text{int}}^1)$$



# Substituability

**Theorem 0.1** *Assume that two interfaces  $C'$  and  $C''$  are compatible, and set  $C = C' \parallel C''$ . Then,*

$$\left. \begin{array}{l} S' \models C' \\ S'' \models C'' \end{array} \right\} \implies S' \parallel_s S'' \models C$$