SUMMERSOC

Hersonissos, June 29, 2016. 9.30 – 10.30

Tutorial
Foundations of SOC

Fundamentals of the SOC Paradigm

Wolfgang Reisig

Humboldt-Universität zu Berlin

Humboldt Universität Informatik

Theory of Programming

Prof. Dr. W. Reisig

Fundamentals of the SOC Paradigm

1. Aspects that exceed classical Theoretical Informatics
2. Towards a Theory of Services
3. Composing *many* services

Fundamentals of the SOC Paradigm

1. Aspects that exceed classical Theoretical Informatics
2. Towards a Theory of Services
3. Composing *many* services

Classical Theoretical Informatics

2012

celebrated as the greatest computer scientist of the 20th century.

Basics of theoretical informatics:

Turing Machines (1936)

Theoretical Informatics in a nutshell

alphabet	Σ;	finitely many symbols	a, b, c, … ,z
words	Σ^*;	countably many	ab, ca, aca, …
functions	f: $\Sigma^* \dashrightarrow \Sigma^*$;	uncountably many	

Some of those functions are "*computable*" (countably many).

Each computable function can effectively be computed

- by a computer (with unbounded store)
- by an amazingly simple kind of machine, a *Turing machine.*

Yet, no computer can compute more functions.

… lots of concepts

useful, undisputed:

- equivalence,
- composition
- complexity
- logical characterizations

- deep theoretical results
- famous open problems.

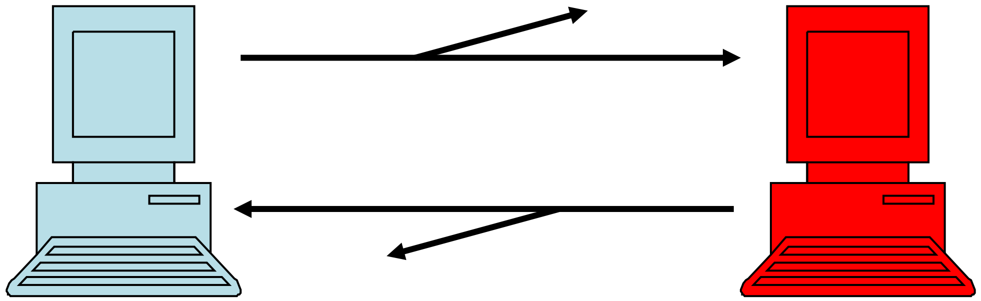
So, the theory of computable functions
is frequently considered ***THE*** theory of informatics.

This talk: *
Informatics comprises formal aspects
that can't be explained as
functions f: $\Sigma^* \dashrightarrow \Sigma^*$

In particular,
service oriented software architectures

here:
three main arguments

1. Informatics comprises *communication*



How establish reliable communication?
By sending acknowledgements, copies, etc. ,
i.e. by means of *distributed algorithms (“protocols”).*

Complexity is not in computation but in communication.

2. Informatics comprises *non-ending behavior*

SOC “always on”

cloud

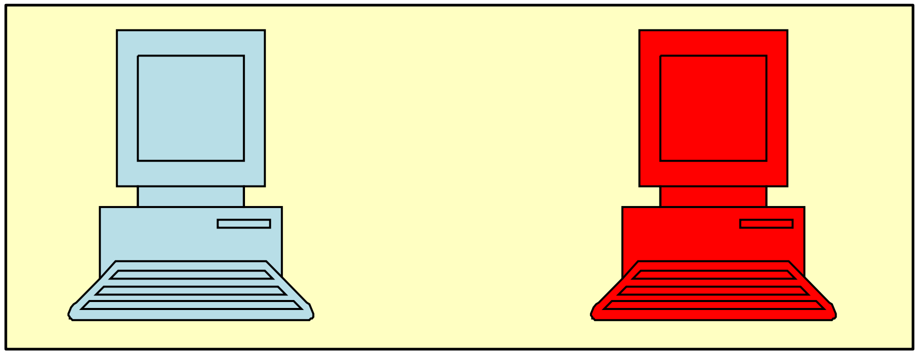
elevator control

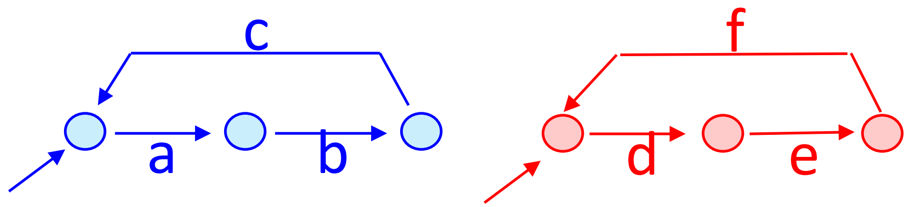
business informatics “24/7”

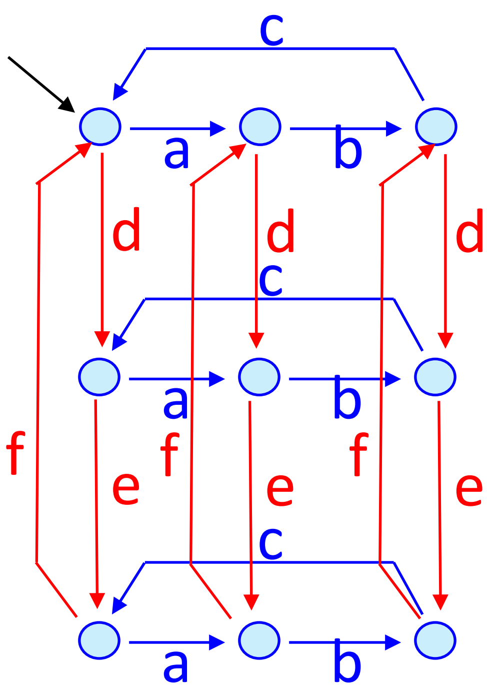
classical view:
terminating behavior is intended,
infinite behavior is mistaken.

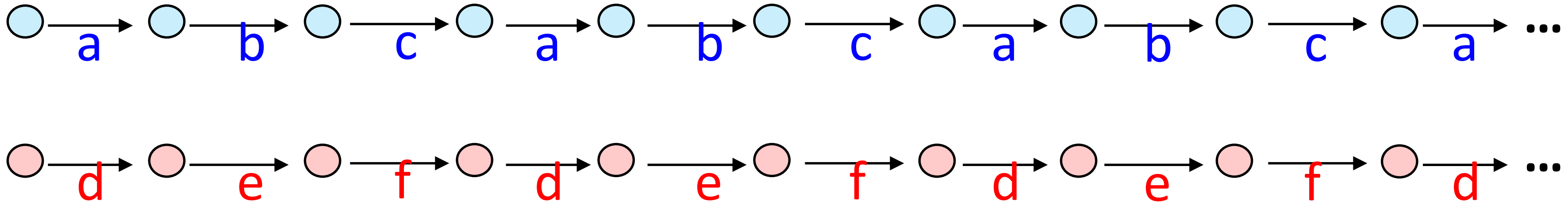
new view:
infinite behavior is intended.
terminating behavior is mistaken.

3. Informatics comprises *causal independence*

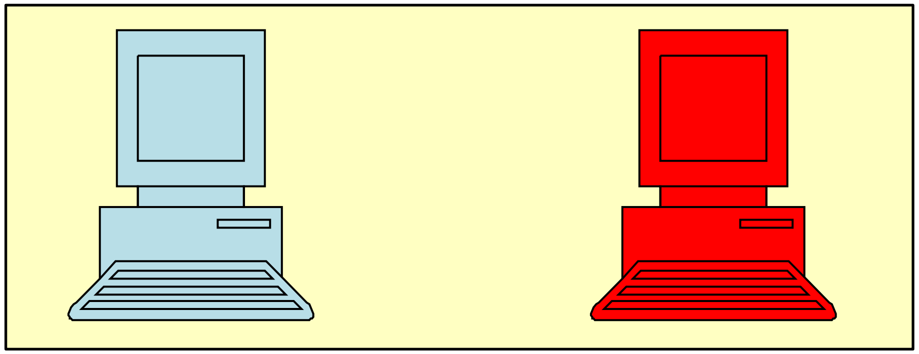


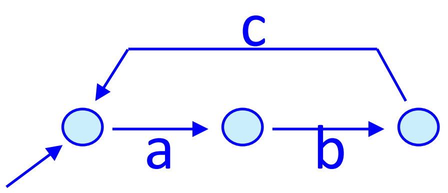


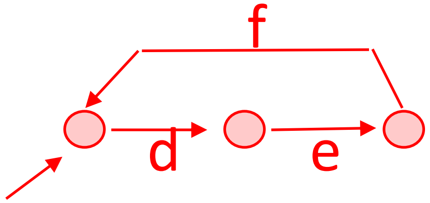


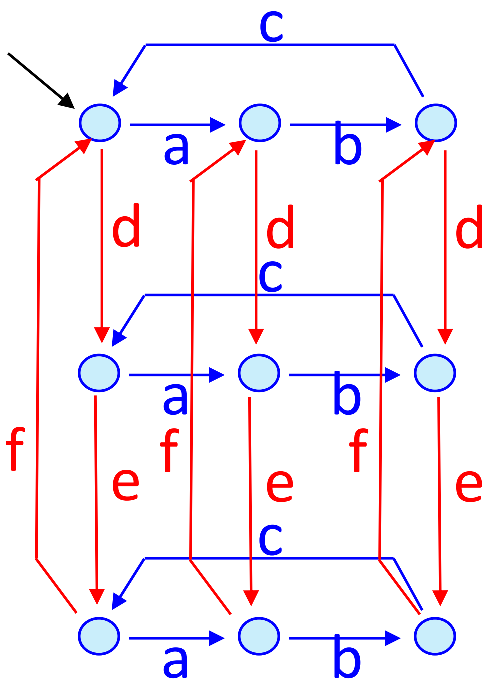


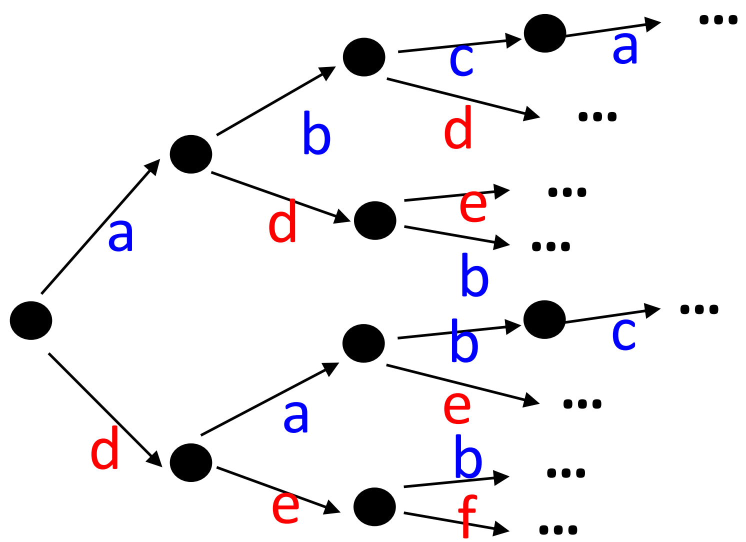
3. Informatics comprises *causal independence*







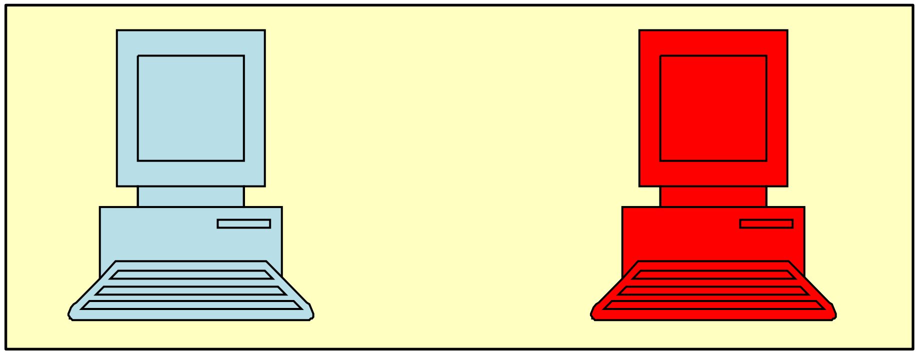


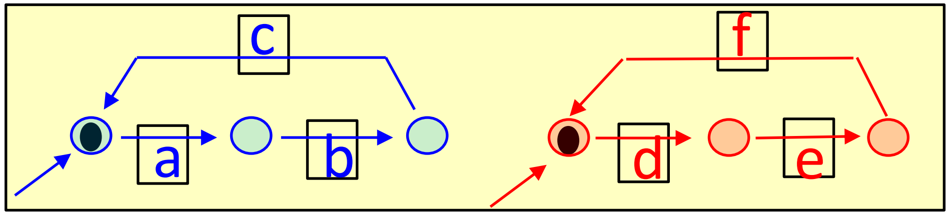


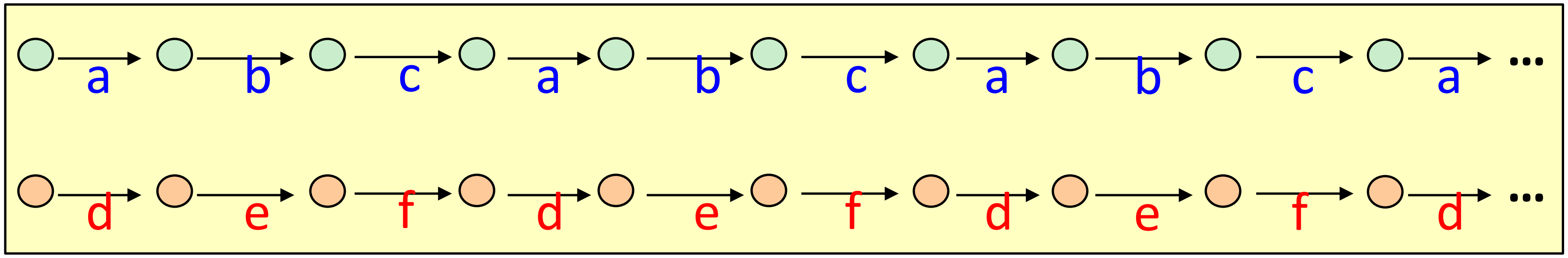
+ fairness assumption

motivated by "observation"

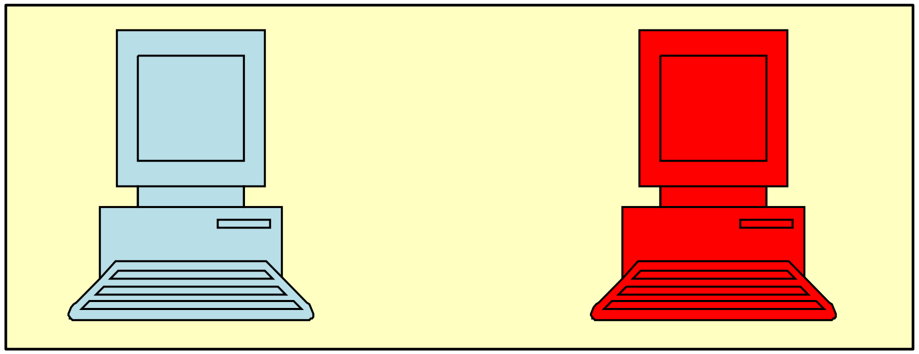
Distributed Systems and Distributed Runs

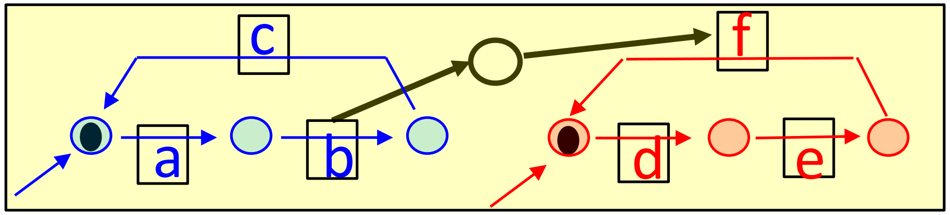




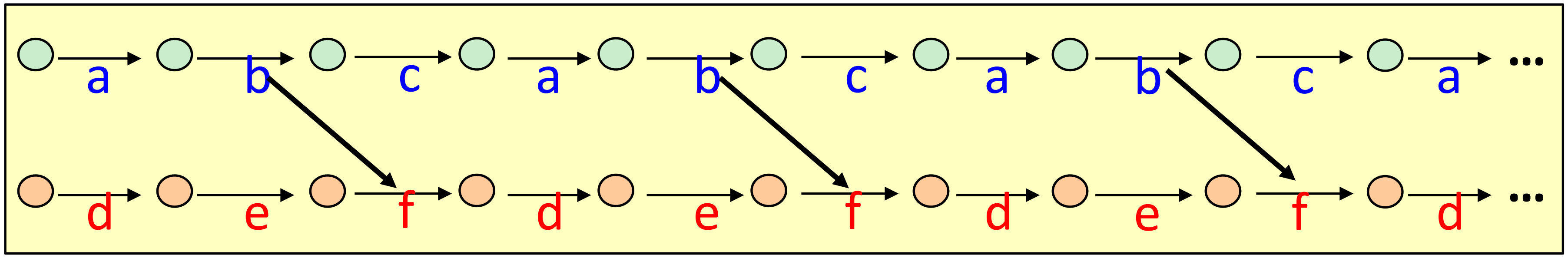


a variant: i-th b before i-th f





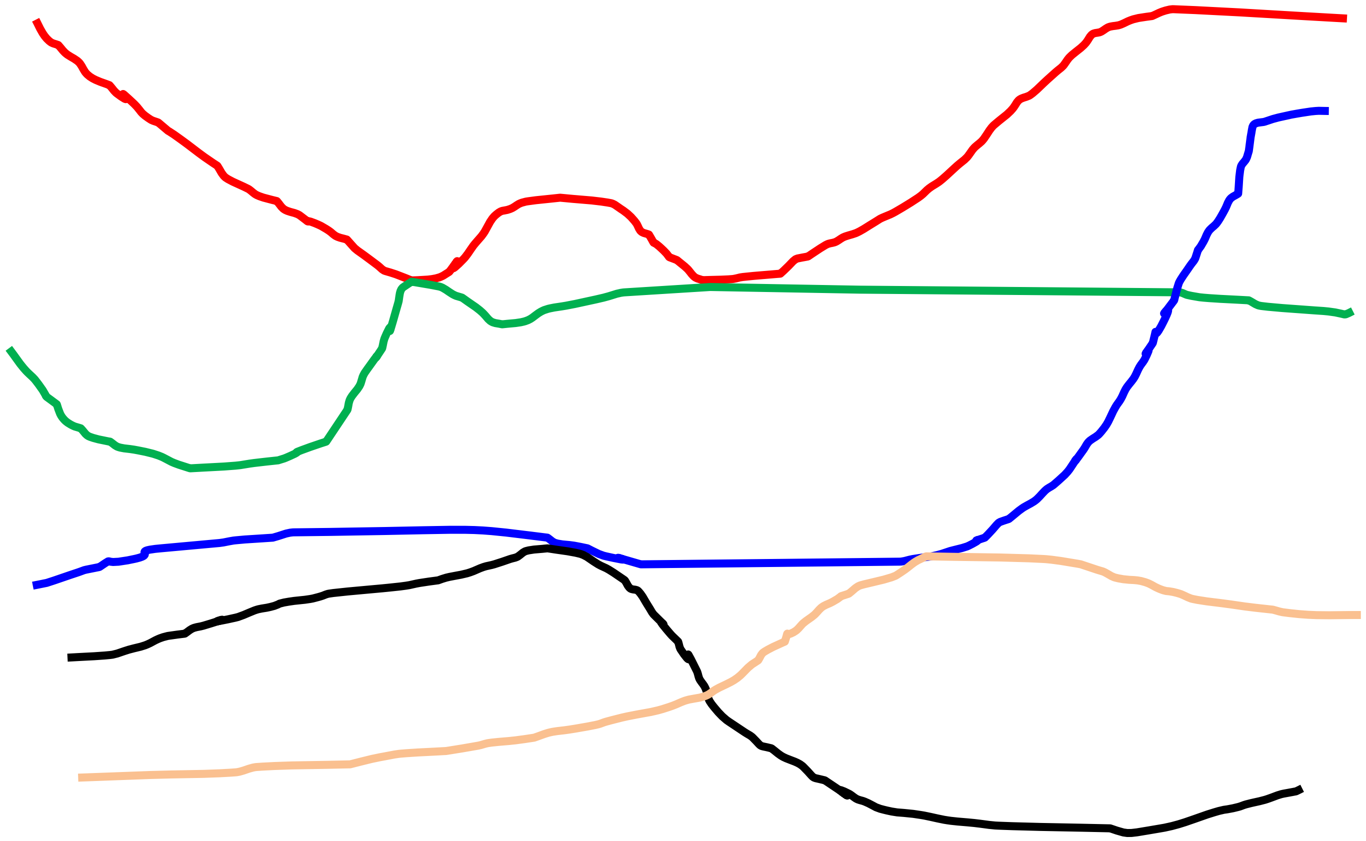
a *deterministic* system
no alternatives
one behavior (run, execution)



more general …

the beer hall pattern:

“ … so that people are continuously criss-crossing from one to another.” … to click their glasses

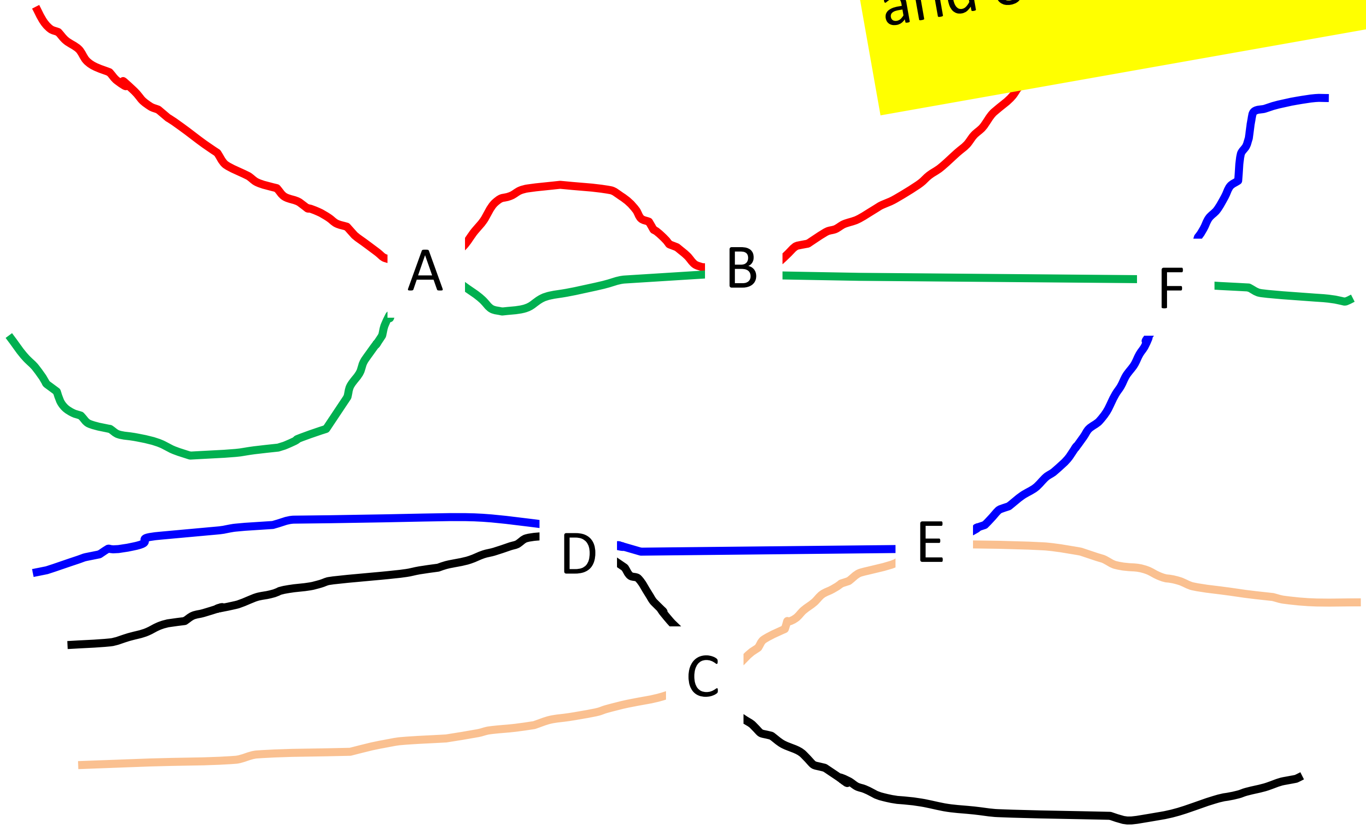


what is this formally?

… a partially ordered set of events

Causality structures the world

Avoid a naïve notion of "time" and of "observation".



This talk:

1. Aspects that exceed classical Theoretical Informatics
2. Towards a Theory of Services
3. Composing *many* services

The World of Software

Classical Programming:

Concepts	Languages	Implementations
f: $\Sigma^* \dashrightarrow \Sigma^*$		
termination is undecidable	Java	very many
one while-loop suffices	C**	
Algorithms		
Semantics of Progr. Lang.		
Verification		

---

the world of SOC:

Concepts	Languages	Implementations
deadlock,		
lifelock,	BPEL	
simulation,	BPNM	*bpel-g*
abstraction,	*standards,*	*open ESP …*
refinement,	*“technical*	*as outlined by*
equivalence,	*neutrality”*	*Jörg Lenhard*
instantiation		
correctness		

The World of Software

Classical Programming:

Concepts	Languages	Implementations
f: $\Sigma^* \dashrightarrow \Sigma^*$		
termination is undecidable	Java	very many
one while-loop suffices	C**	
Algorithms		
Semantics of Progr. Lang.		
Verification		

---

the world of SOC:

Concepts	Languages	Implementations
deadlock,	BPEL	*bpel-g*
lifelock,	BPNM	*open E*
simulation,	*sta ds,*	,
abstraction,	*al*	
refinement	*ality”*	
equiv		
instan		
correct ess		

... just informal, plain English

... comparable to classical programming in the late 1950ies!

Semantics should be mathematics!

Requirement:

In analogy to programming languages:

The semantics of a service is a mathematical object!

True, this is presently not the case.

BUT WE SHOULD spend effort into this!

Interaction is represented as *composition*

Requirements:

The – elementary – notion of composition of services
is a (simple!) mathematical (or logical!) operation.

For services S and T,
the composition $S \oplus T$
is a service again.

(Frequently, $S \oplus T$ does not interact any more.)

ticketing $=_{\text{def}}$
sell_ticket $\oplus$ buy_ticket

The algebraic structure of services

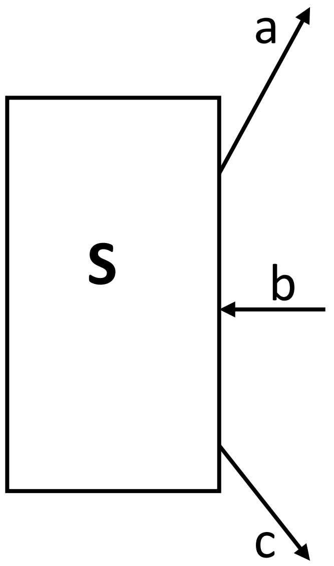
Given:

- a set $\mathbb{S}$ of *services,*
- a *composition* operator $\oplus : \mathbb{S} \times \mathbb{S} \to \mathbb{S}$,

This yields the algebraic structure

$$(\mathbb{S};\ \oplus).$$

Models of services



... a transition system

with *channels*

for *asynchronous* communication

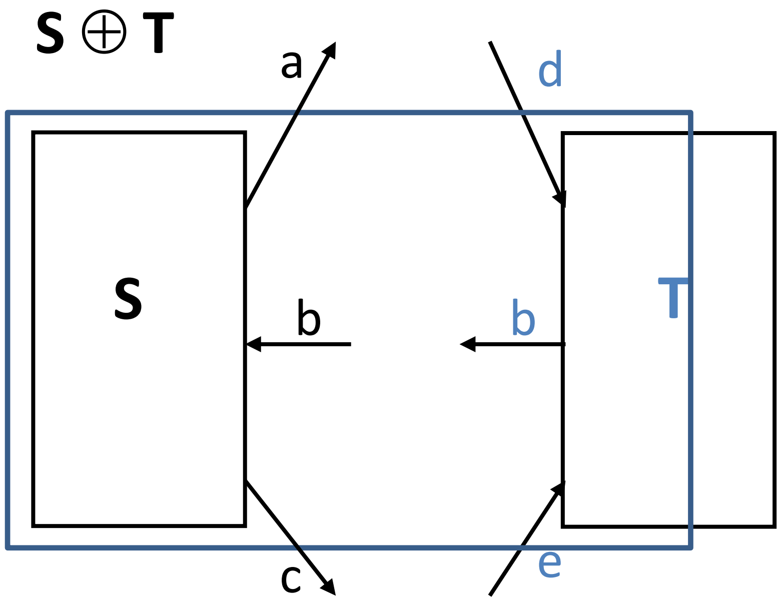
with *its environment*.

Semantics of **S:**
During a computation, each
channel funnels a stream of data.

technically:
a relation on – infinite – streams

not too convincing

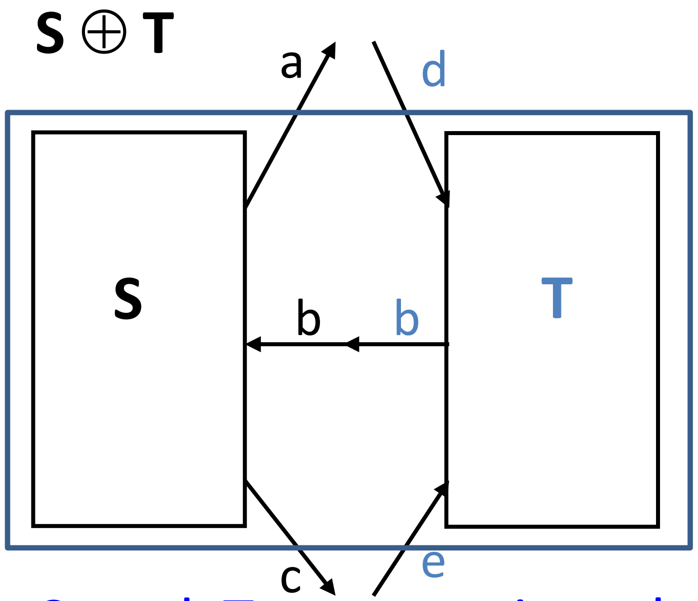
How to compose services?



Composition **S ⊕ T**
has pending channels.
… is a service again.

The world consists of composed services

Requirements at composed services



Together, services may accomplish a *requirement, ρ.*

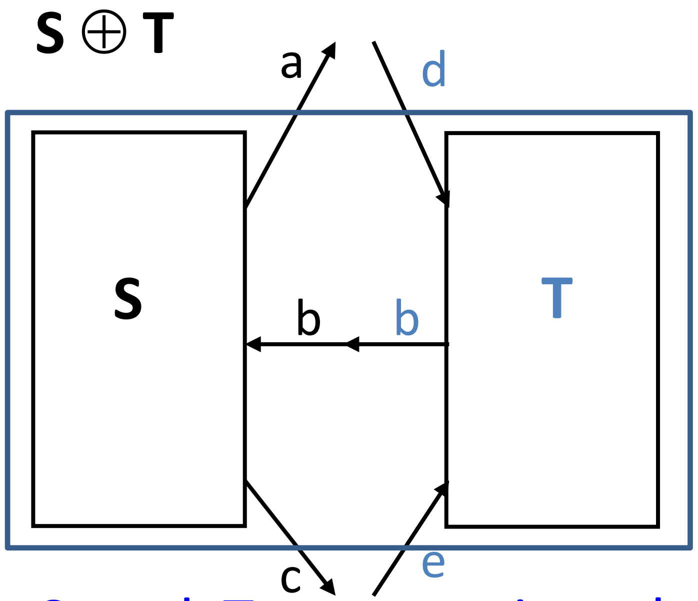
… as CTL* formulas:

S and **T** communicate boundedly	**S** ⊕ **T** ” *AG n-bounded*
S and **T** communicate responsively	**S** ⊕ **T** ” *AGEF responsive*

With *target* states:

S ⊕ **T** weakly terminates	**S** ⊕ **T** ” *AGEF terminal*
S ⊕ **T** is deadlock free	**S** ⊕ **T** ” *AG (terminal ♎ target)*
S ⊕ **T** is lifelock free	**S** ⊕ **T** ” *AGEF target*

For a requirement ρ …



S and **T** communicate boundedly

S and **T** communicate responsively

With *target* states:

S ⊕ **T** weakly terminates

S ⊕ **T** is deadlock free

S ⊕ **T** is lifelock free

Def.: Let ρ be a requirement on services.

(i) **S** and **T** are ρ *-partners* iff **S** ⊕ **T** ” ρ

(ii) **S** is substitutable by **T** iff for all **U**, **S** ⊕ **U** ” ρ implies **T** ⊕ **U** ” ρ

! on-the-fly-substitution

(iii) **U** is a ρ *-adapter* for **S** and **T** iff **S** ⊕ **U** ⊕ **T** ” ρ

properties of services

Quests at the partners of a service, S,

w.r.t a requirement ρ *:*

Does S have ρ-partners at all ? — Controllability

Is T a ρ-partner of S ? — Composability

How construct a *canonical* ρ-partner of S ? — "most liberal"

How characterize *all* ρ-partners of S ? — Operating Guideline

a general goal

Description of
semantics and (in particular) composition of services:

- on a high level of business logic.
- not on a low level of implementation details.

Describe system *properties* !

The algebraic structure of services

Given:

- a set $\mathbb{S}$ of *services,*
- a *composition* operator $\oplus : \mathbb{S} \times \mathbb{S}$ ♎ $\mathbb{S}$,
- a set Q of *requirements* $\rho_1, \ldots, \rho_n \subseteq \mathbb{S}$.

This yields the algebraic structure

$(\mathbb{S}, \oplus, Q)$.

For $S, T \in \mathbb{S}$, $\rho \in Q$,

T is a ρ - *partner* of S,

iff $S \oplus T$ ” ρ.

Let $\text{sem}_\rho(S) =_{\text{def}}$ the set of all ρ - partners of S.

the "classical" requirement

ρ : weak termination

derived notions

(w.r.t some ρ):

S may be *substituted by S' :*

$\text{sem}_\rho(S) \subseteq \text{sem}_\rho(S')$

S and T *are equivalent:*

$\text{sem}_\rho(S) = \text{sem}_\rho(T)$

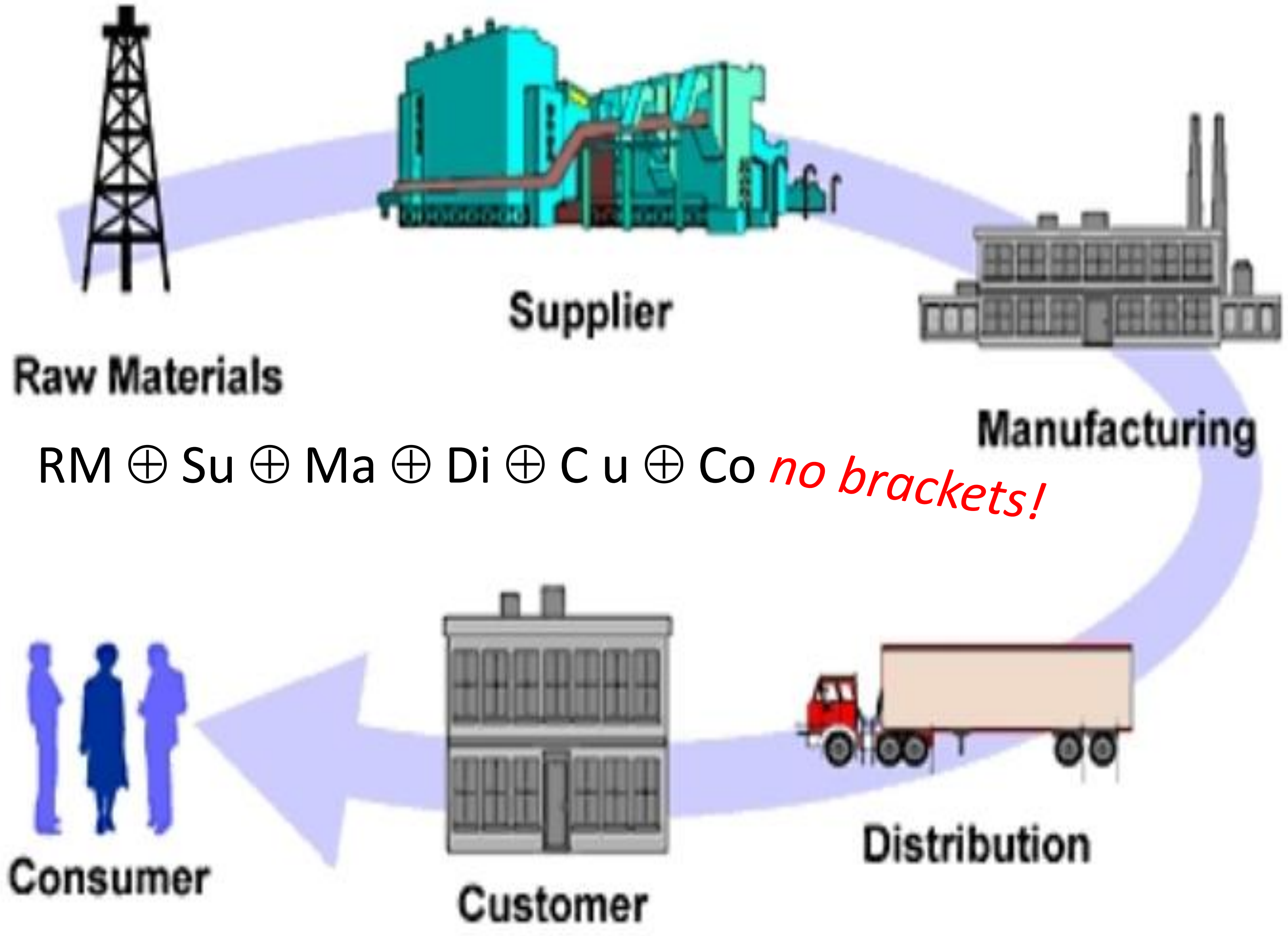
U adapts S and T:

$S \oplus U \oplus T$ ” ρ

This talk:

1. Aspects that exceed classical Theoretical Informatics
2. Towards a Theory of Services
3. Composing *many* services

Example: a supply chain



$\mathrm{RM} \oplus \mathrm{Su} \oplus \mathrm{Ma} \oplus \mathrm{Di} \oplus \mathrm{C\ u} \oplus \mathrm{Co}$ *no brackets!*

Example: an adapter

(

⊕

) ⊕

socket ⊕ adapter ⊕ plug *no brackets!*

The algebraic structure of services

Given:

- a set $\mathbb{S}$ of *services,*
- an ***associative*** *composition* operator $\oplus : \mathbb{S} \times \mathbb{S} \to \mathbb{S}$,
- a set Q of *requirements* $\rho_1, \ldots , \rho_n \subseteq \mathbb{S}$.

This yields the algebraic structure

$$(\mathbb{S}, \oplus, Q).$$

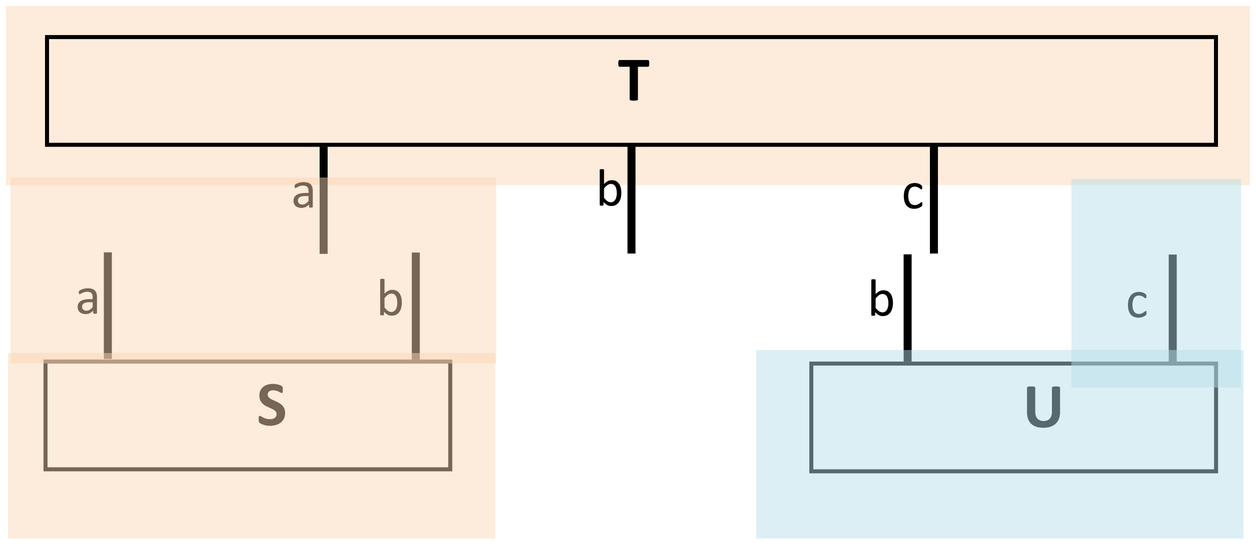
Wanted

A generic notion of "Service" (component) such that:

- A service S has an *interface* and an *inner part.*
- Two services *S* and *T* may be composed along their interfaces, yielding a service *S*$\oplus$*T.*
- The interfaces of *S* and *T* have fitting elements.
- Fitting elements of the interfaces of *S* and *T* turn into inner elements of *S*$\oplus$*T.*

Problem: a minimal set of requirements at such services
and their composition $\oplus$
such that $\oplus$ is total and associative.

a naïve composition

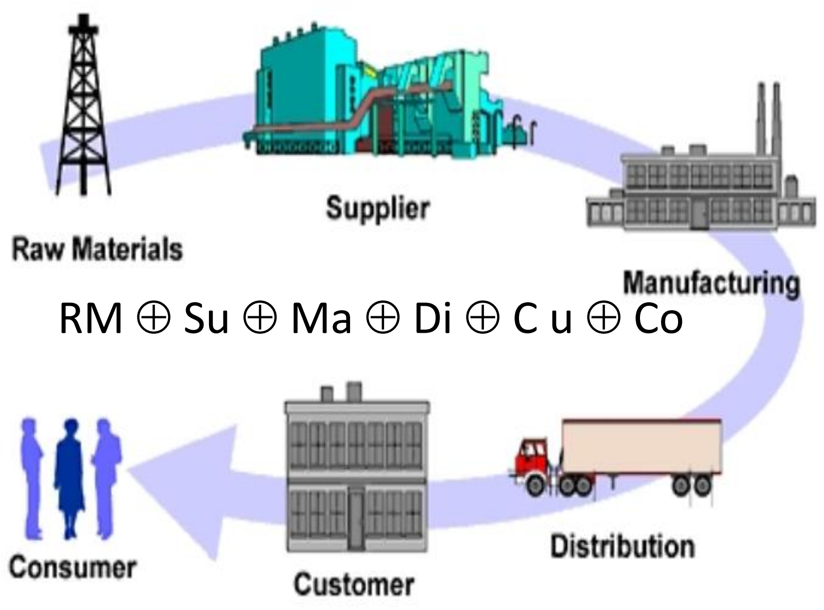


$(S \oplus T) \oplus U$

$\neq$

A fundamental idea:

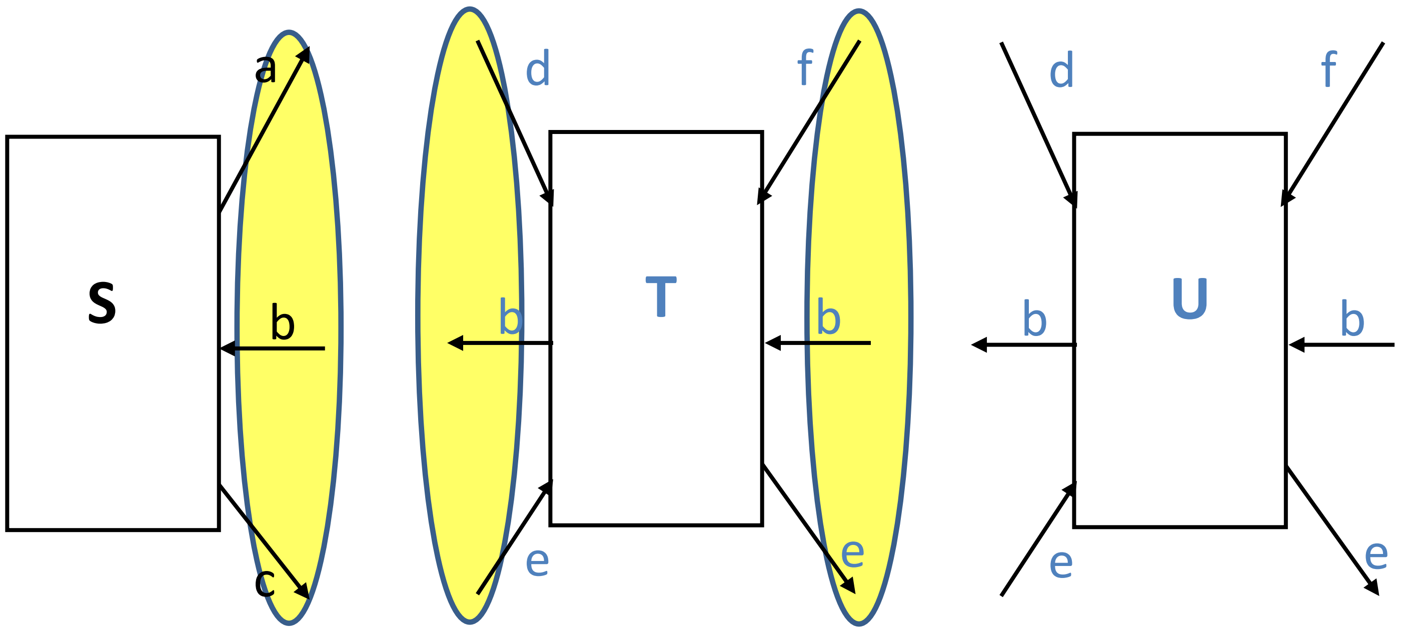
A services' *S* interface is partitioned into a *left* and a *right* port S_l and S_r !



RM ⊕ Su ⊕ Ma ⊕ Di ⊕ C u ⊕ Co

input	and	*output*
customer	and	*supplier*
provider	and	*requester*
producer	and	*consumer*
buy side	and	*sell side*

two Ports

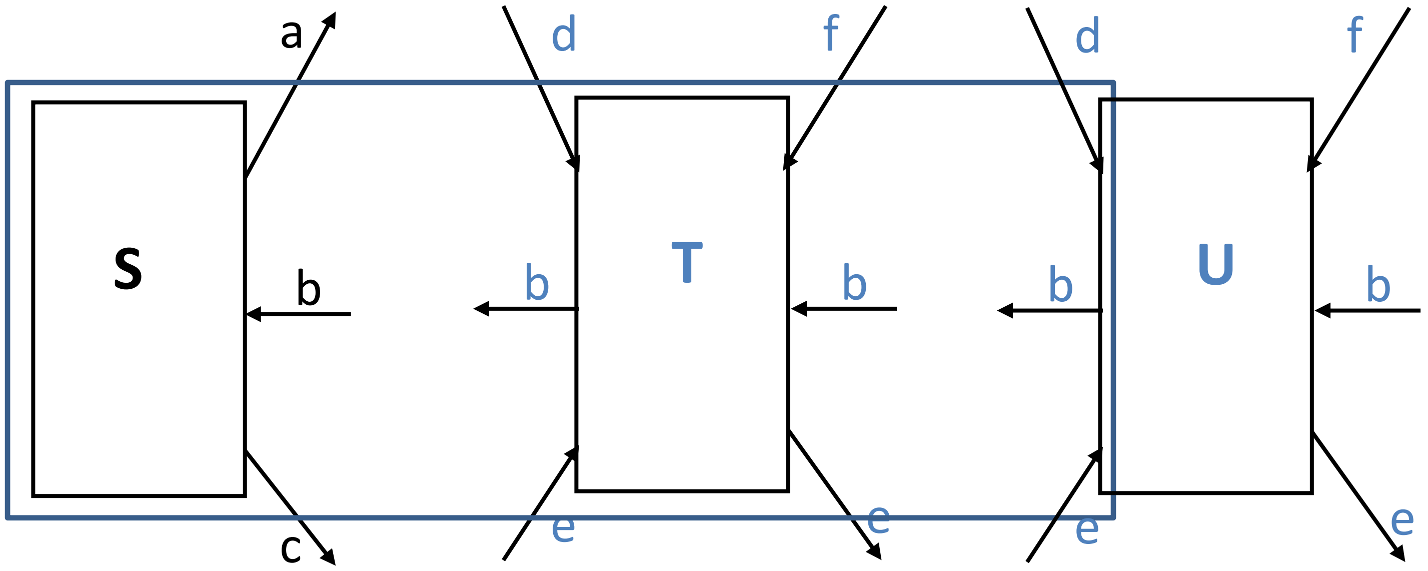


right port S_r left port T_l right port T_r

Idea:
A services' *S* interface is partitioned into a *left* and a *right* port S_l and S_r !

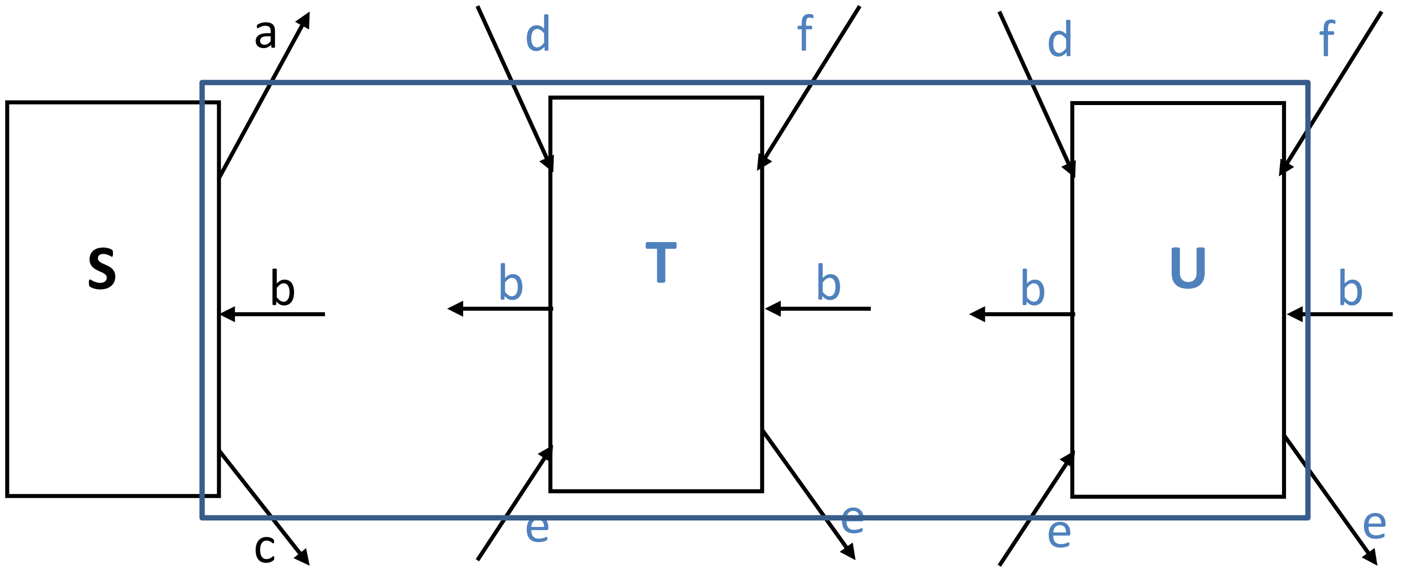
For $S \oplus T$, compose S_r with T_l .

composition along ports



(S ⊕ T) ⊕ U

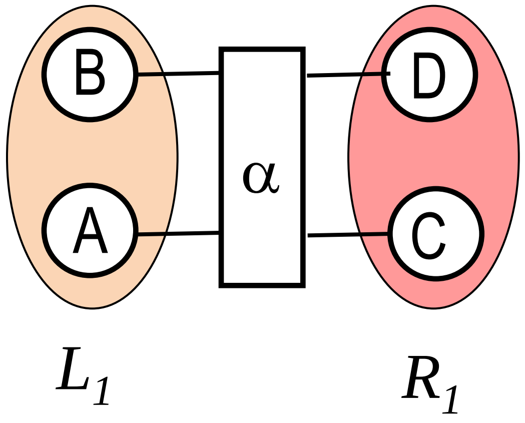
… is associative!



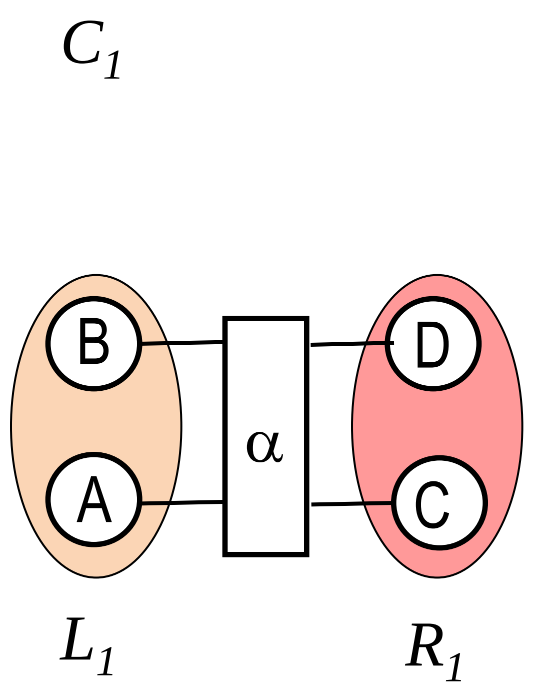
$(S \oplus T) \oplus U \quad = \quad S \oplus (T \oplus U)$

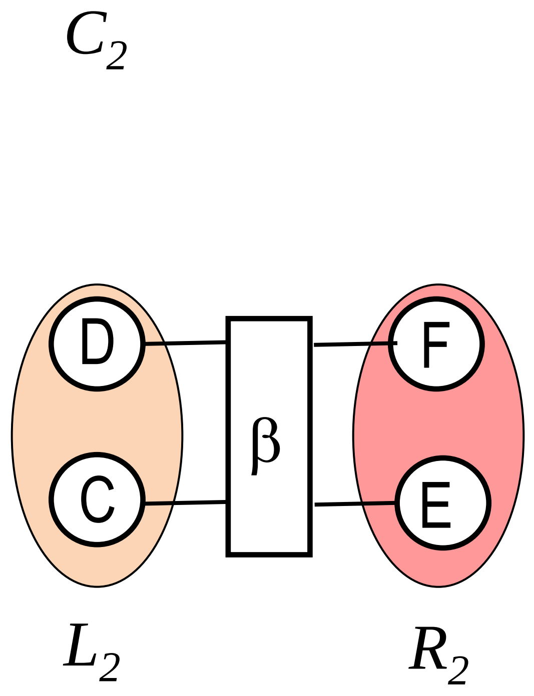
… more detailed

C_1

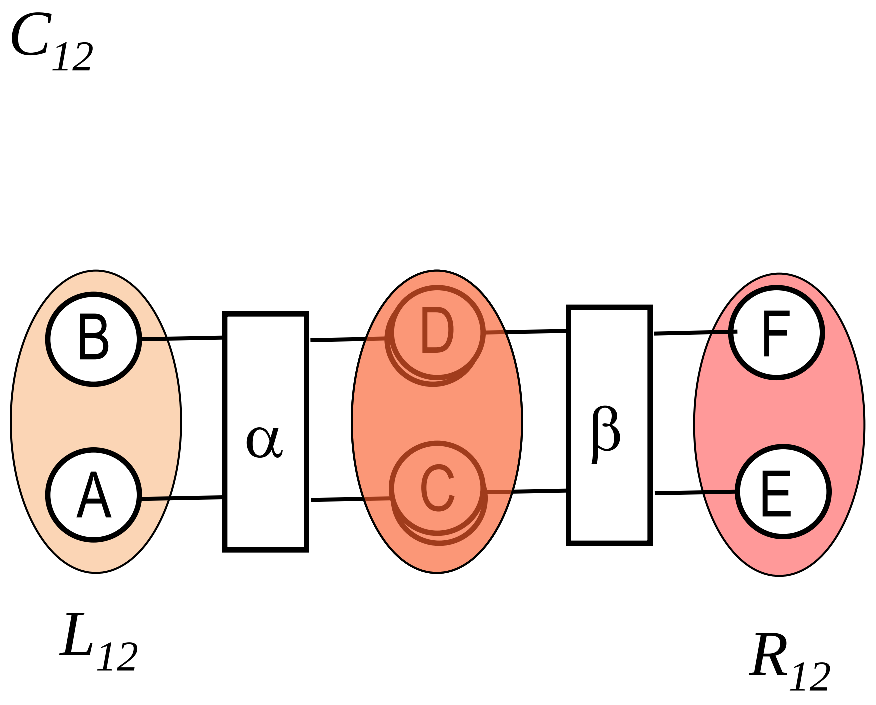


R_1 and L_2 fit perfectly

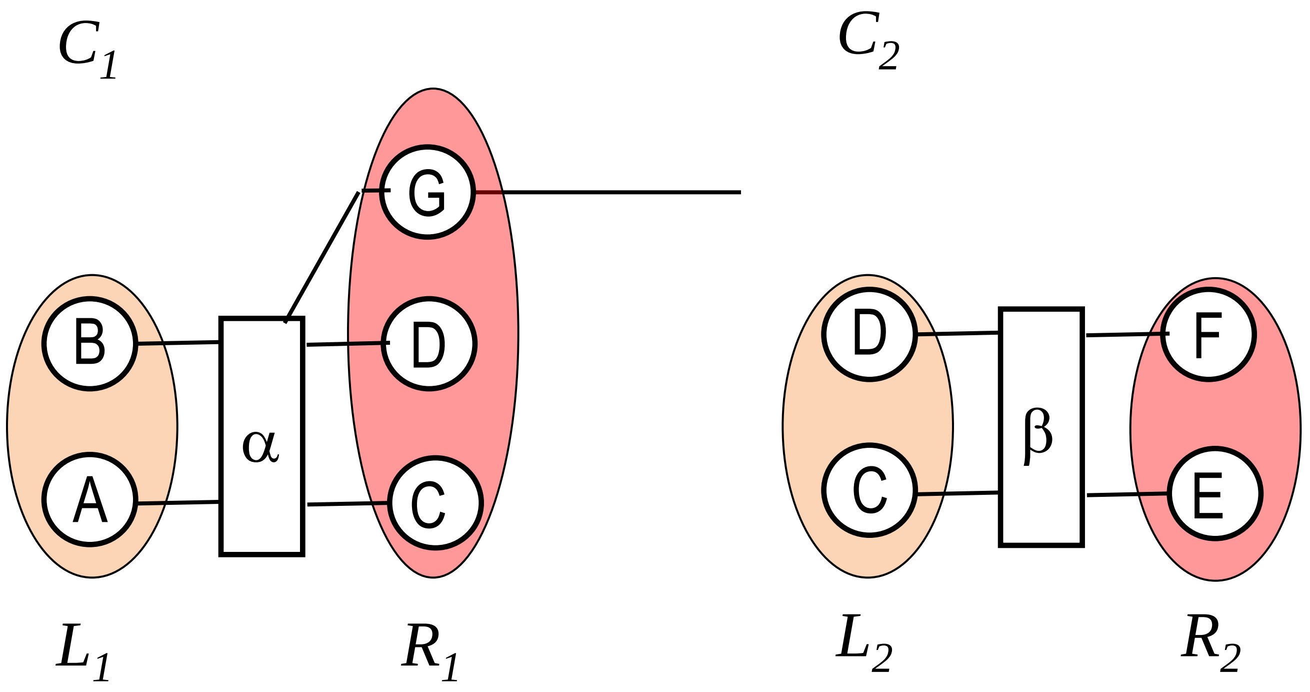




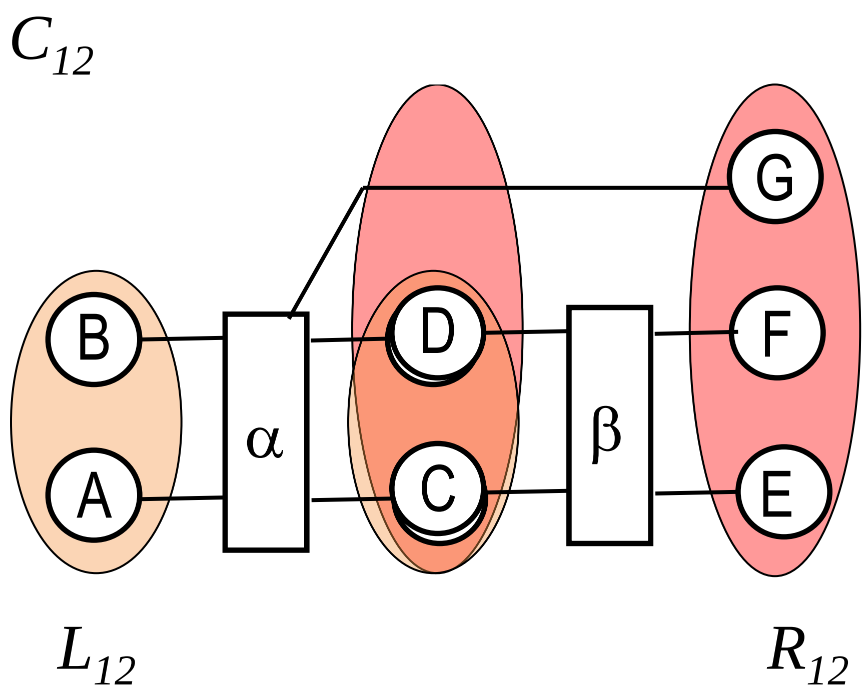
Composition C_1 👓 C_2



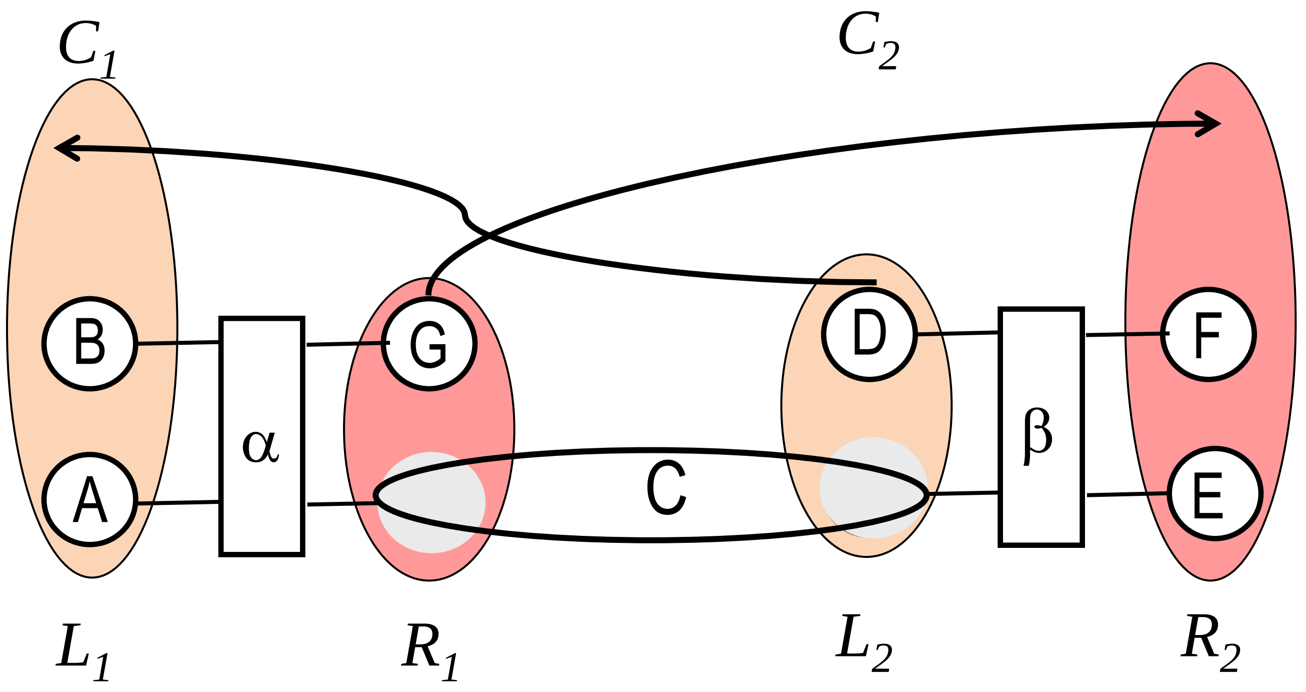
… it is not always that simple



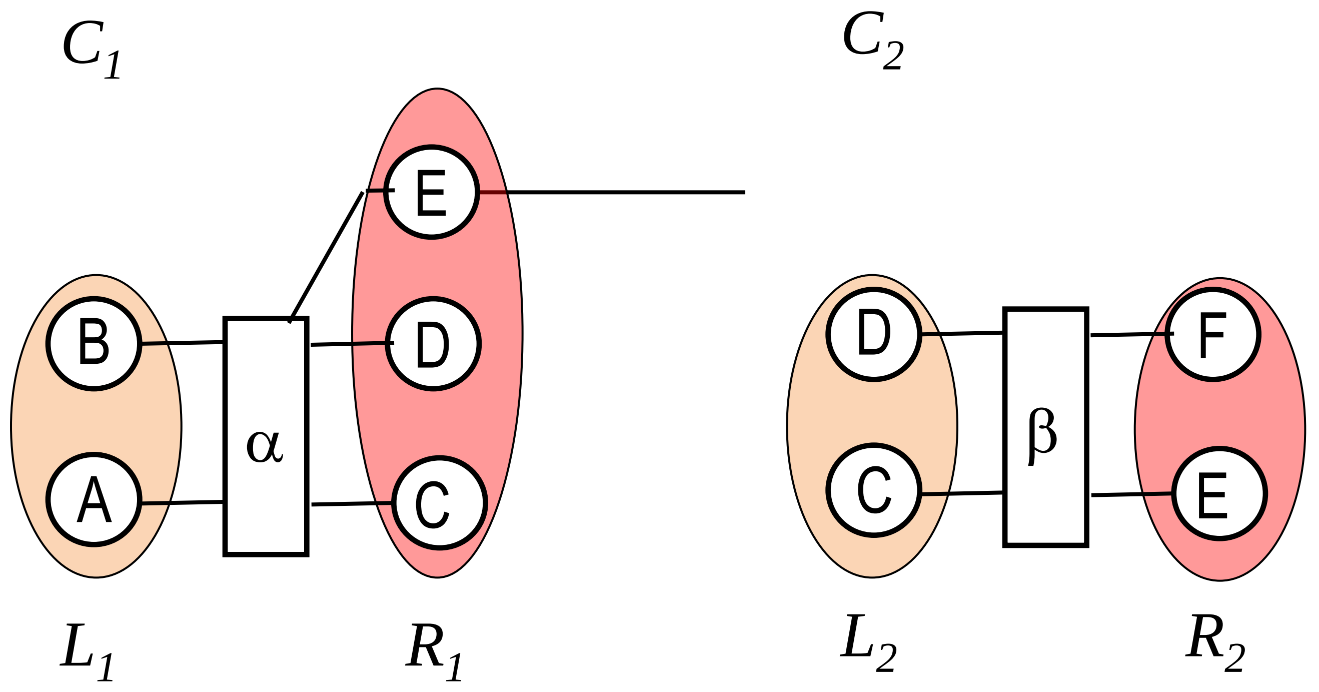
Composition C_1 👓 C_2



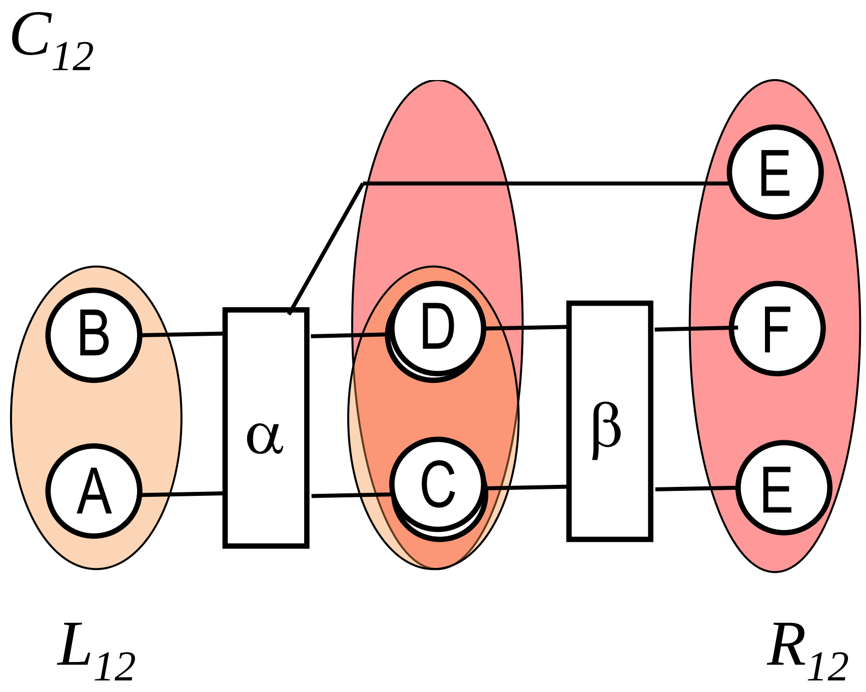
This works nicely:



… unfortunately



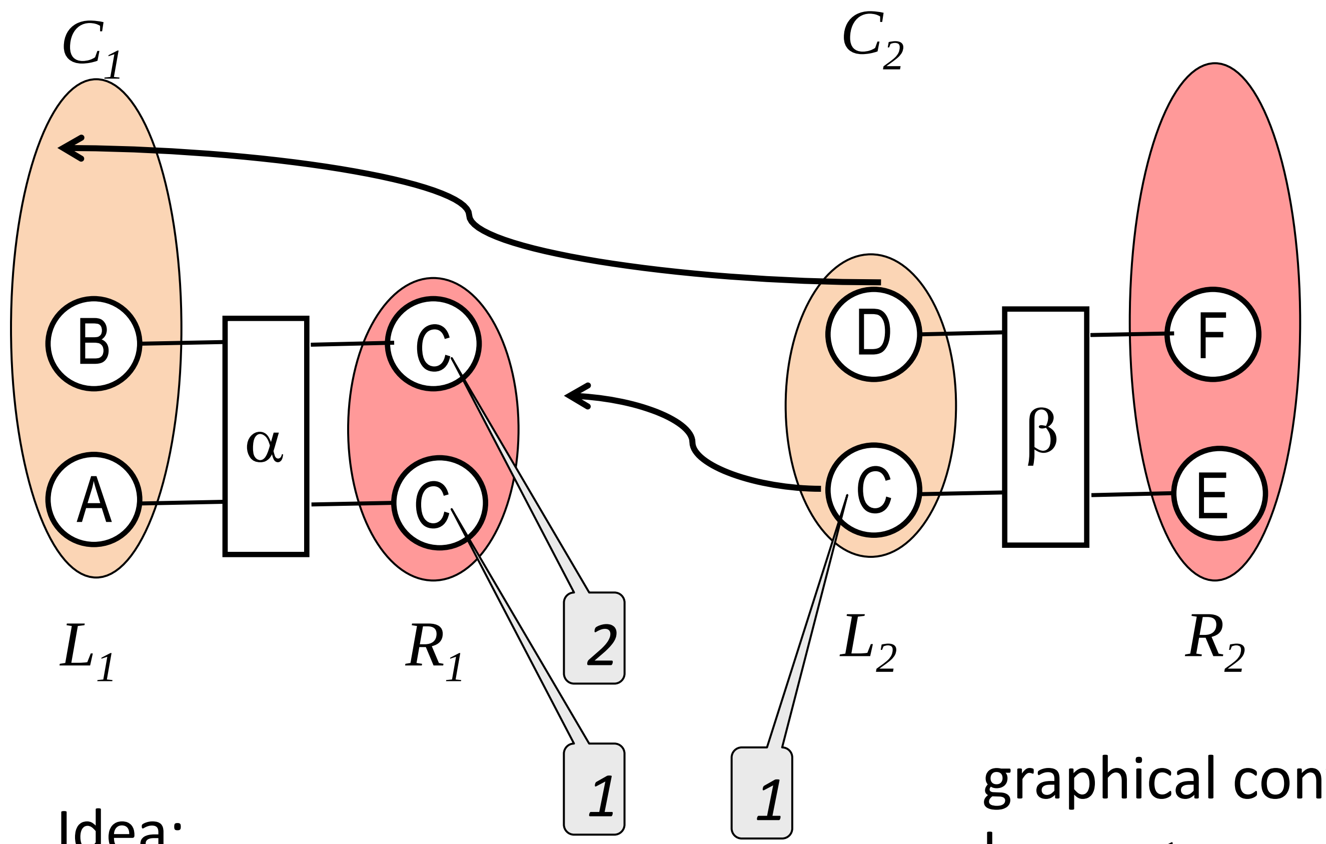
Port with multiple label



Two nodes of R_{12} are labelled alike!

You can not avoid this!

… what to do *here* ???

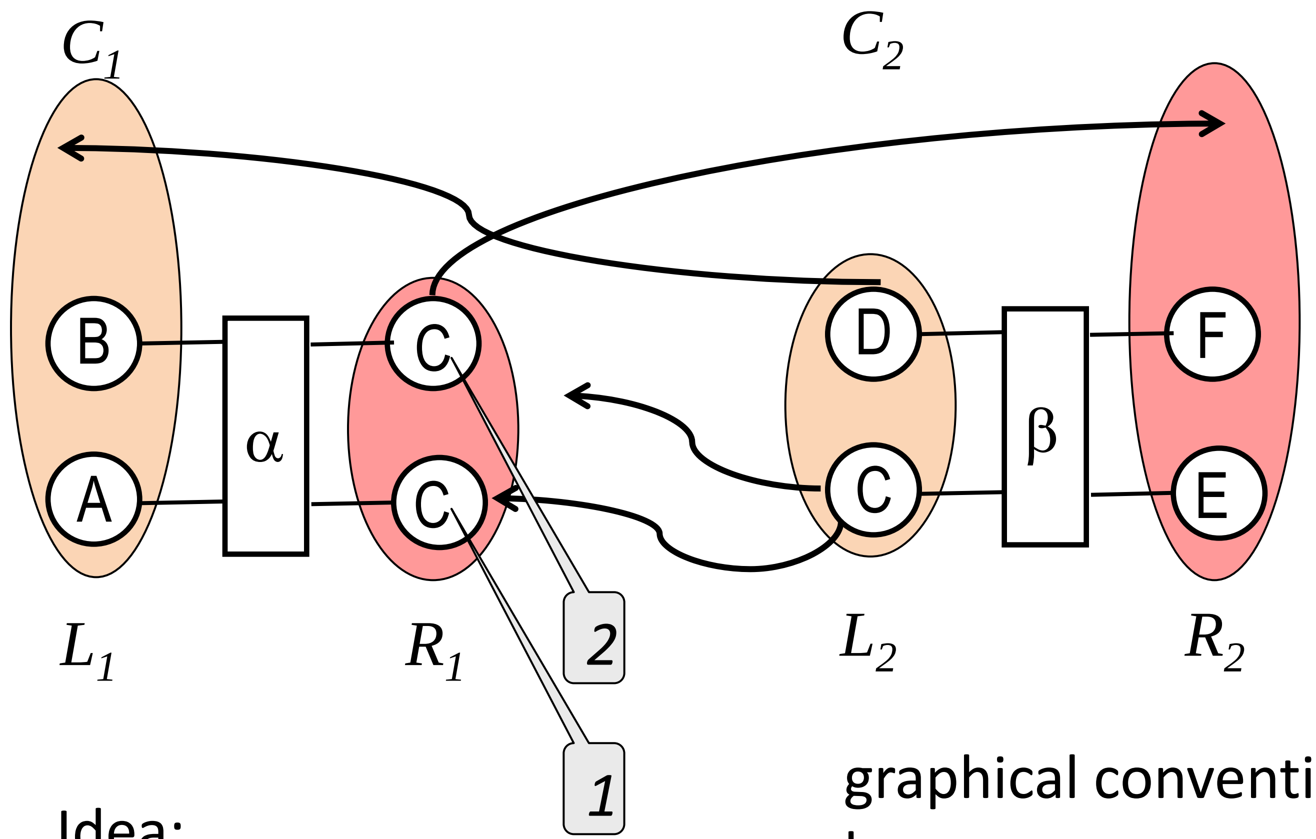


Idea:
n equally labelled nodes in one port are *indexed 1, … n* .

graphical convention:
lower < upper.

Glue
equally labelled and equally indexed nodes.

… what to do *here* ???

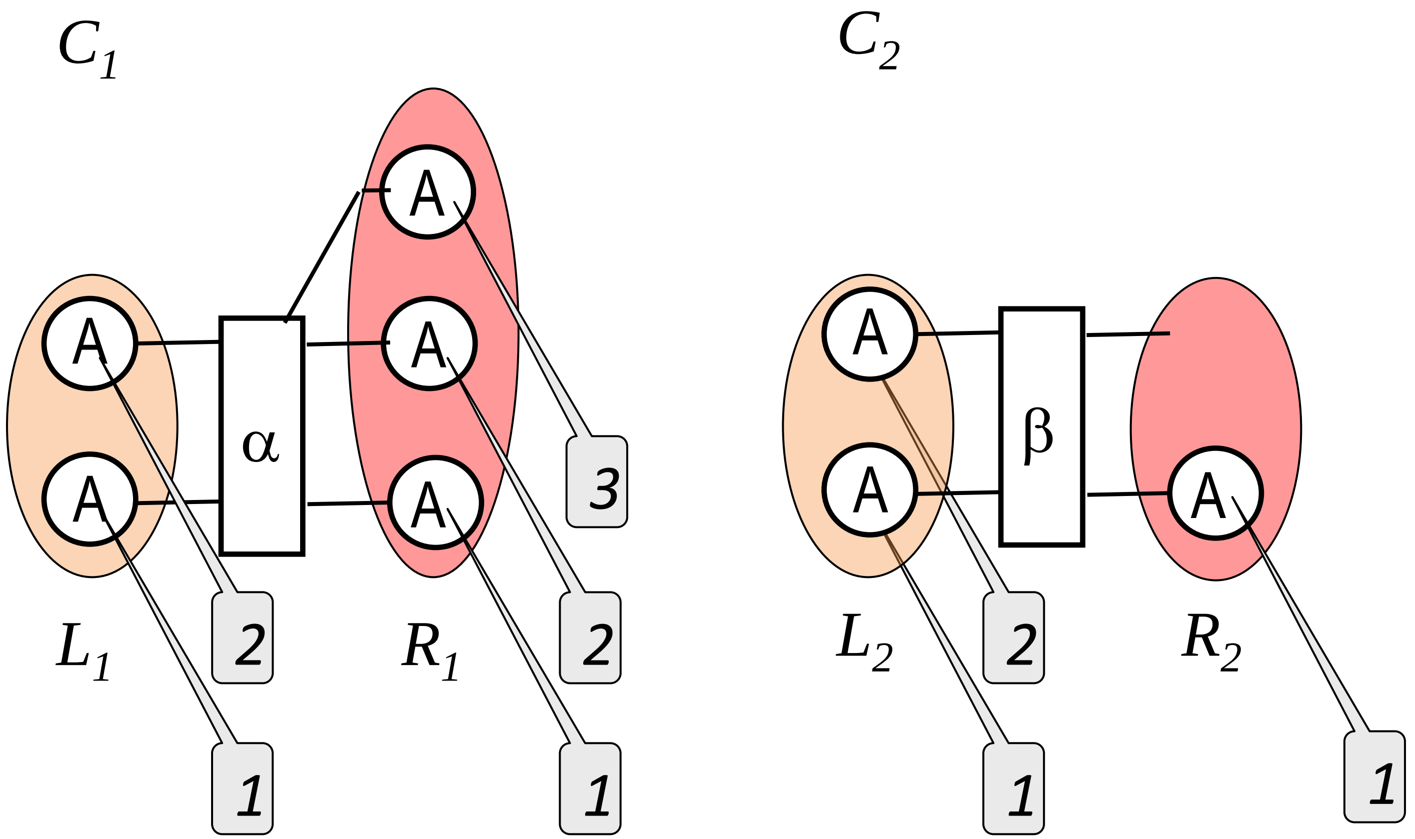


Idea:
Equally labelled nodes
in one port
are *ordered*.

graphical convention:
lower < upper.

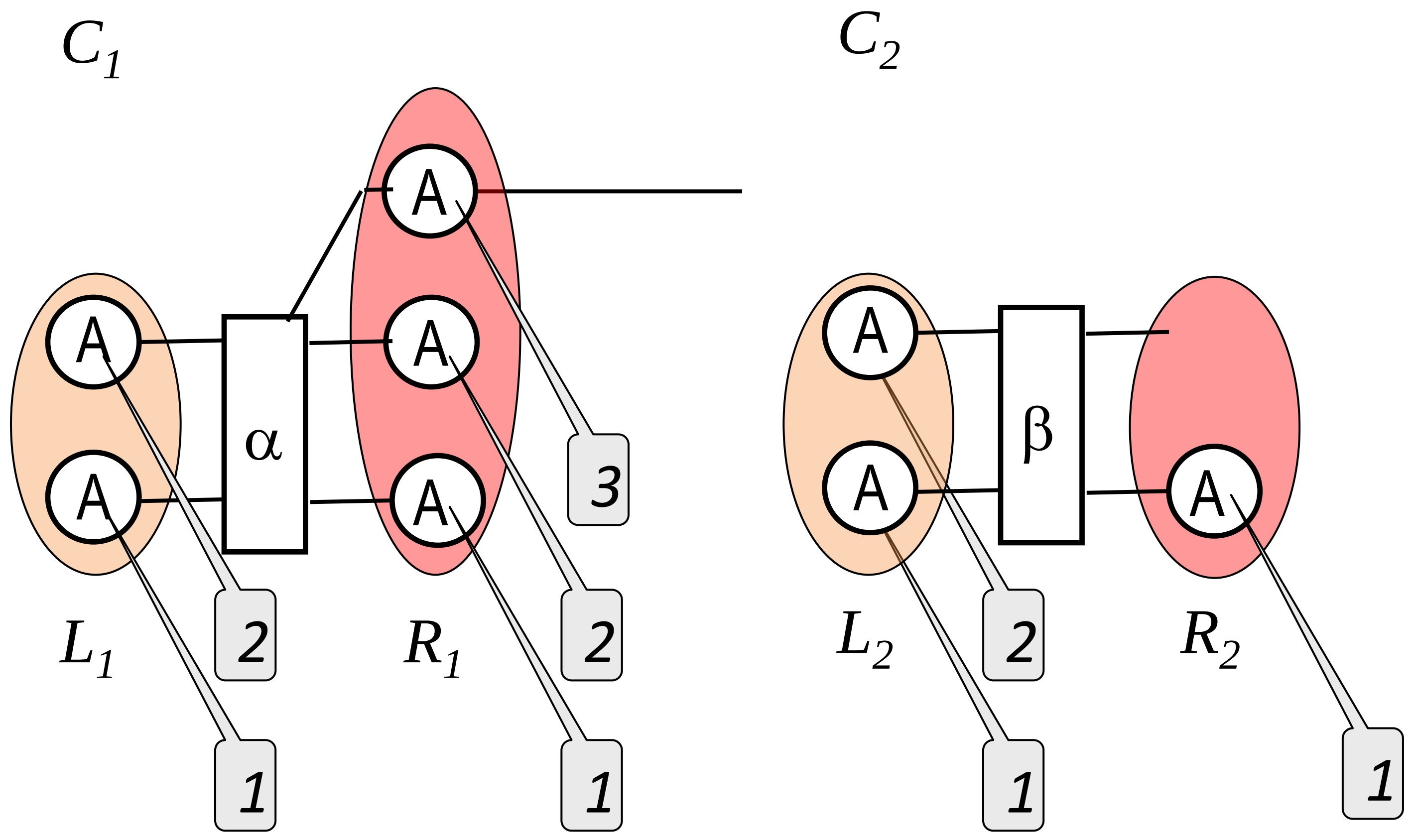
Glue
equally labelled nodes
both n-th in their order.

An extreme case



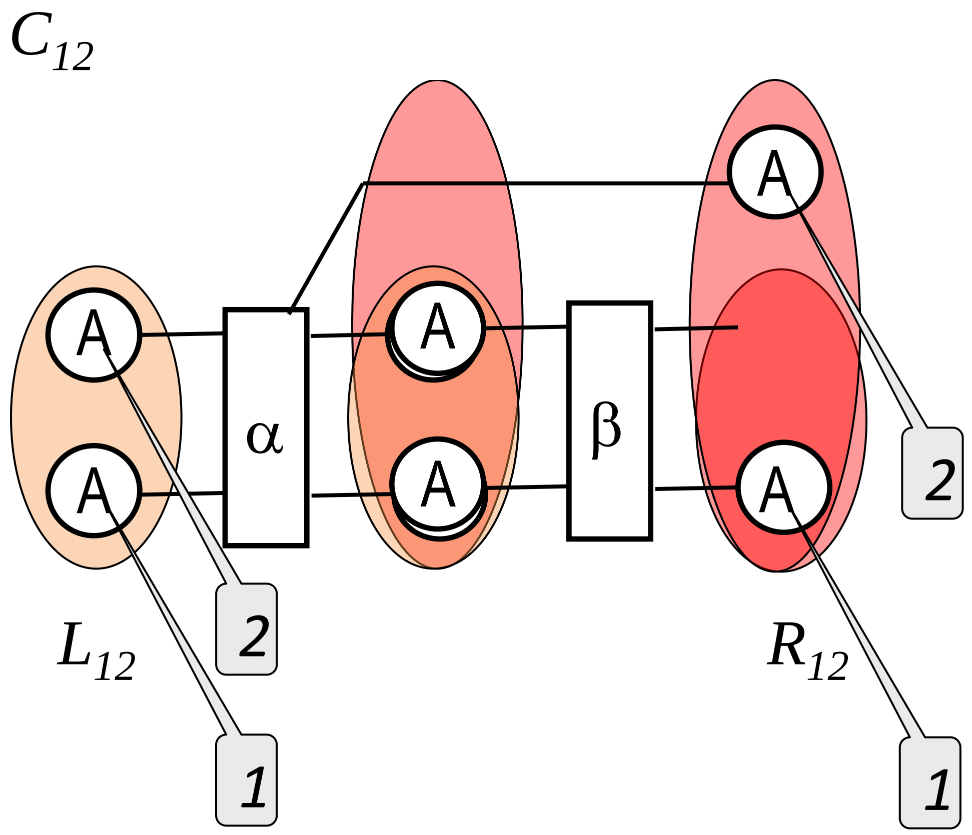
all labels alike.

An extreme case



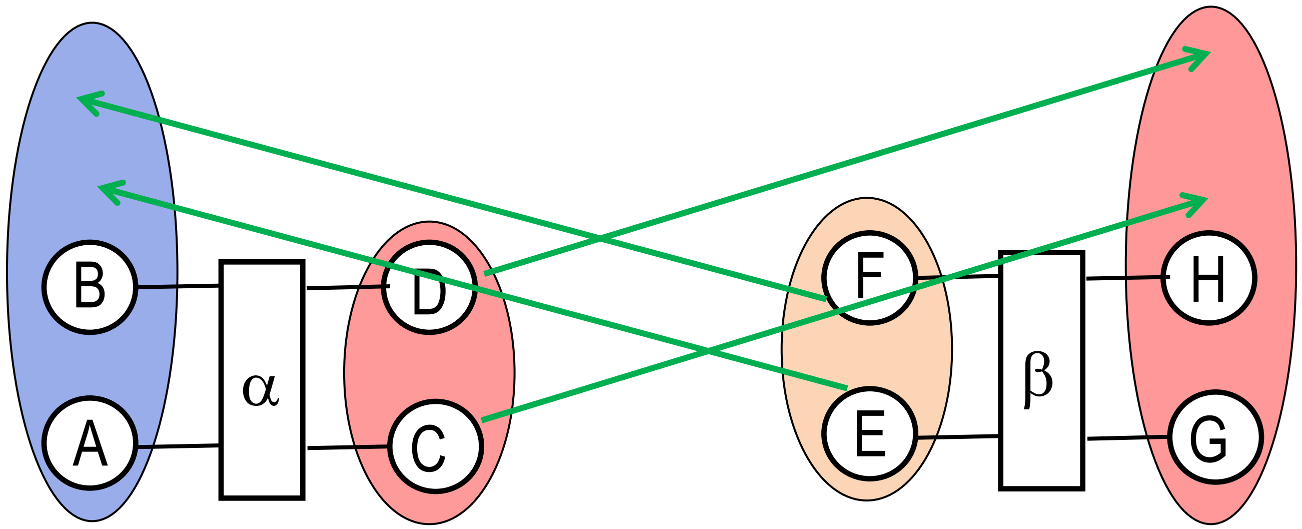
all labels alike.

An extreme case



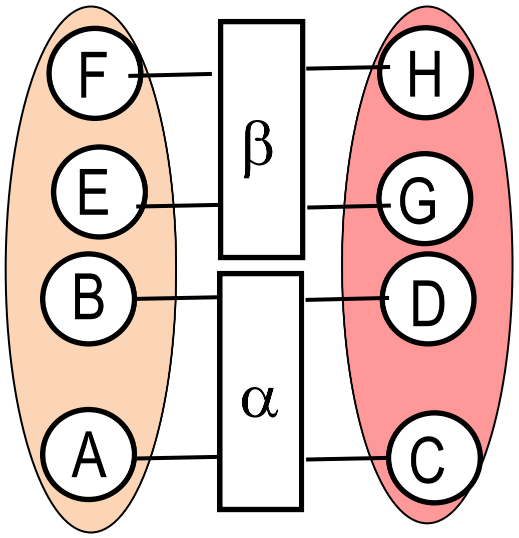
all labels alike.

… another extreme case



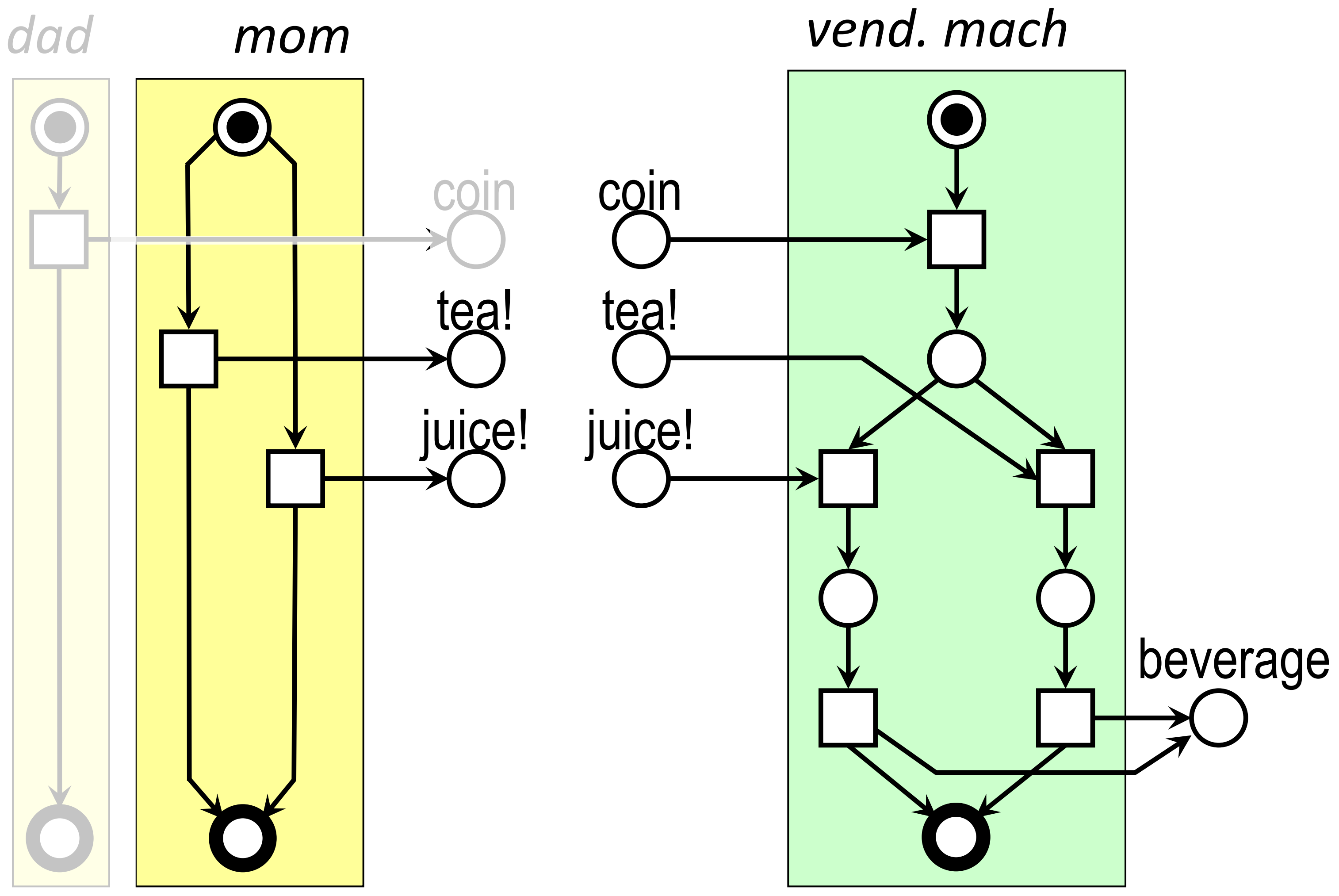
all labels different.

results in



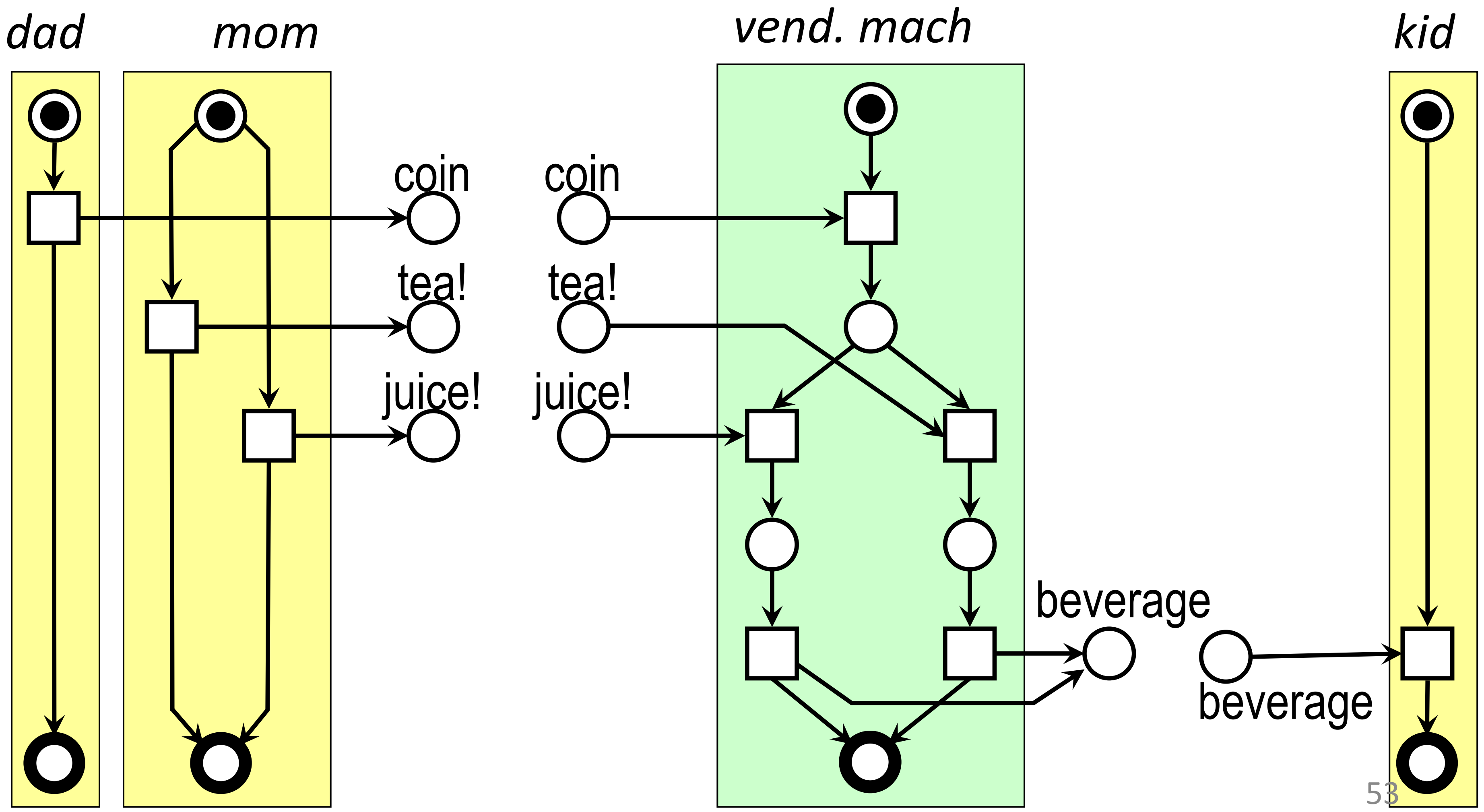
… a tricky property

dad pays, *mom* selects,



… a tricky property

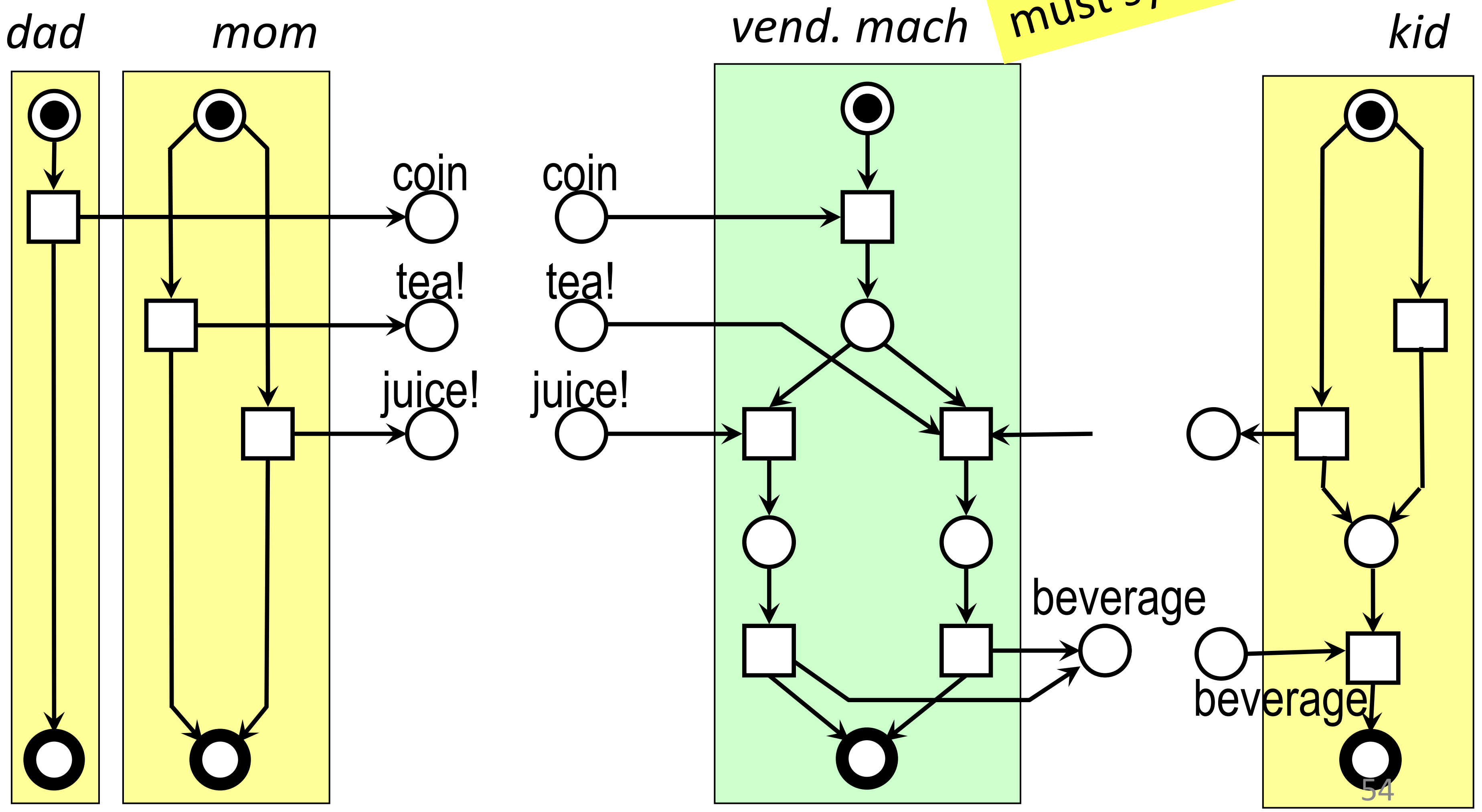
dad pays, *mom* selects, *kid* drinks.



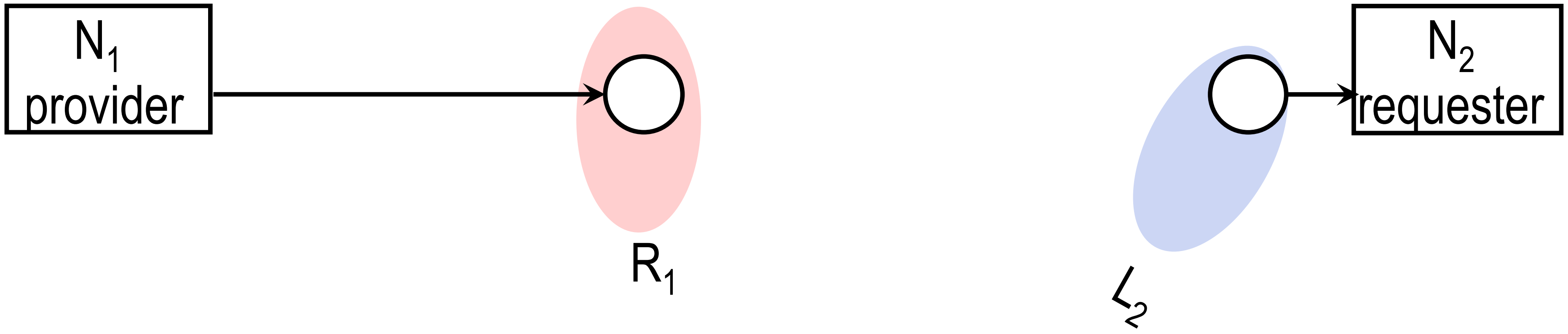
A variant of the vending machine

dad pays, *mom* selects, *kid* drinks.

mom and kid must synchronize!

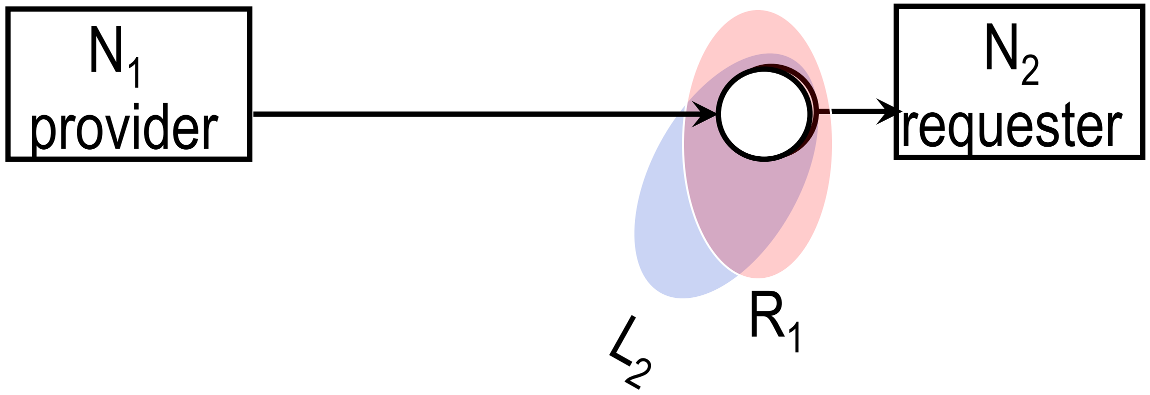


Ports may overlap!

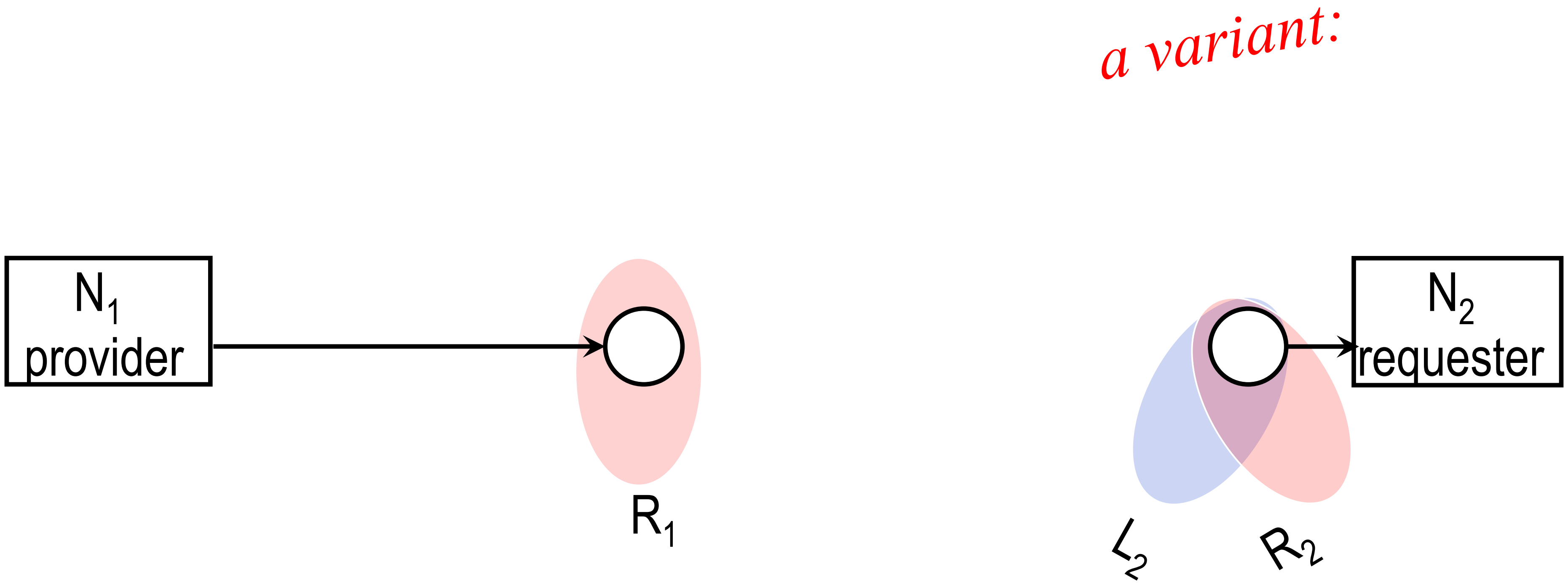


exclusive requester

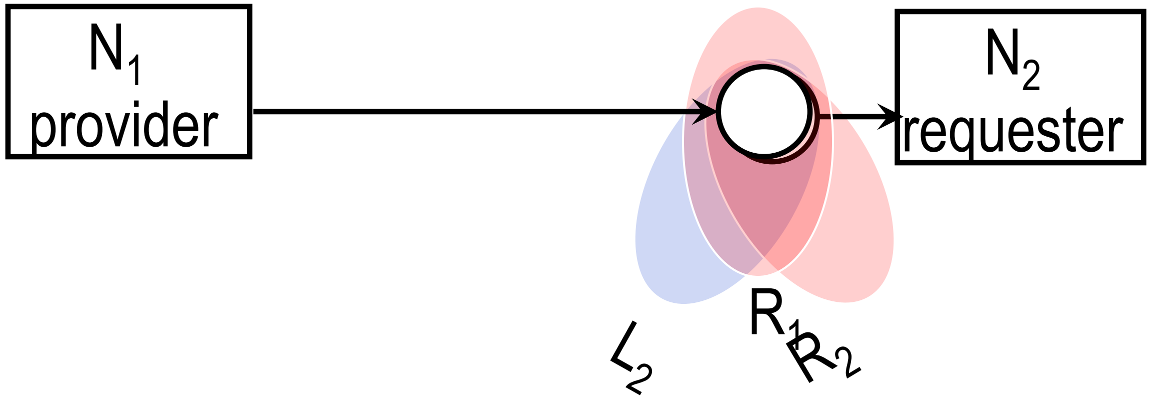
a variant:



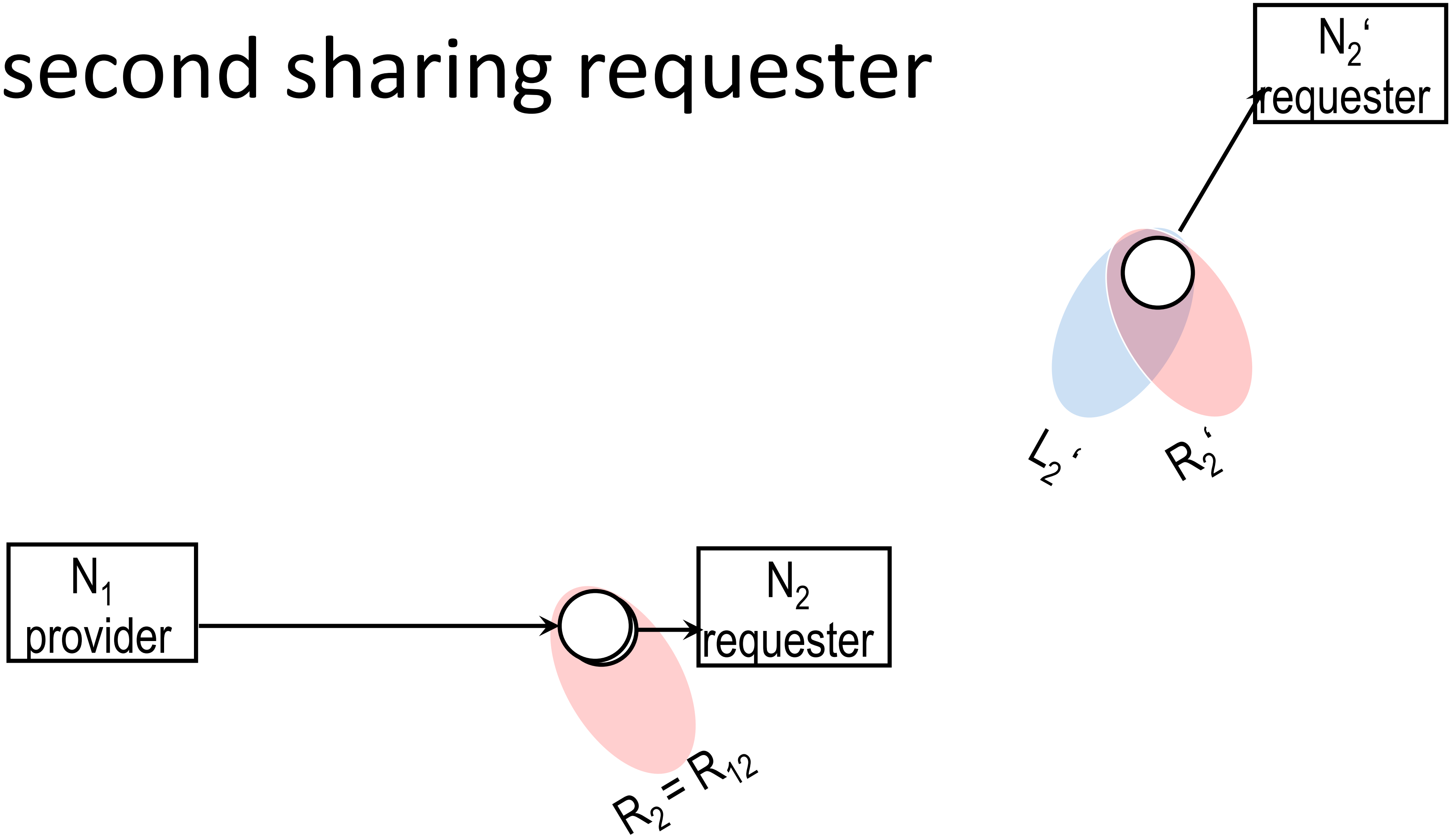
sharing requester



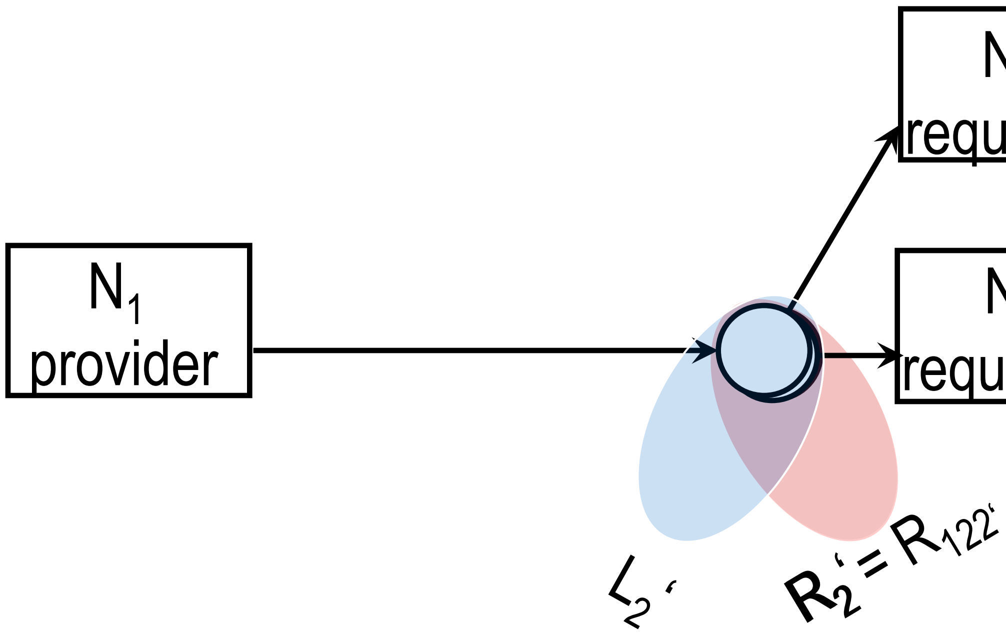
sharing requester



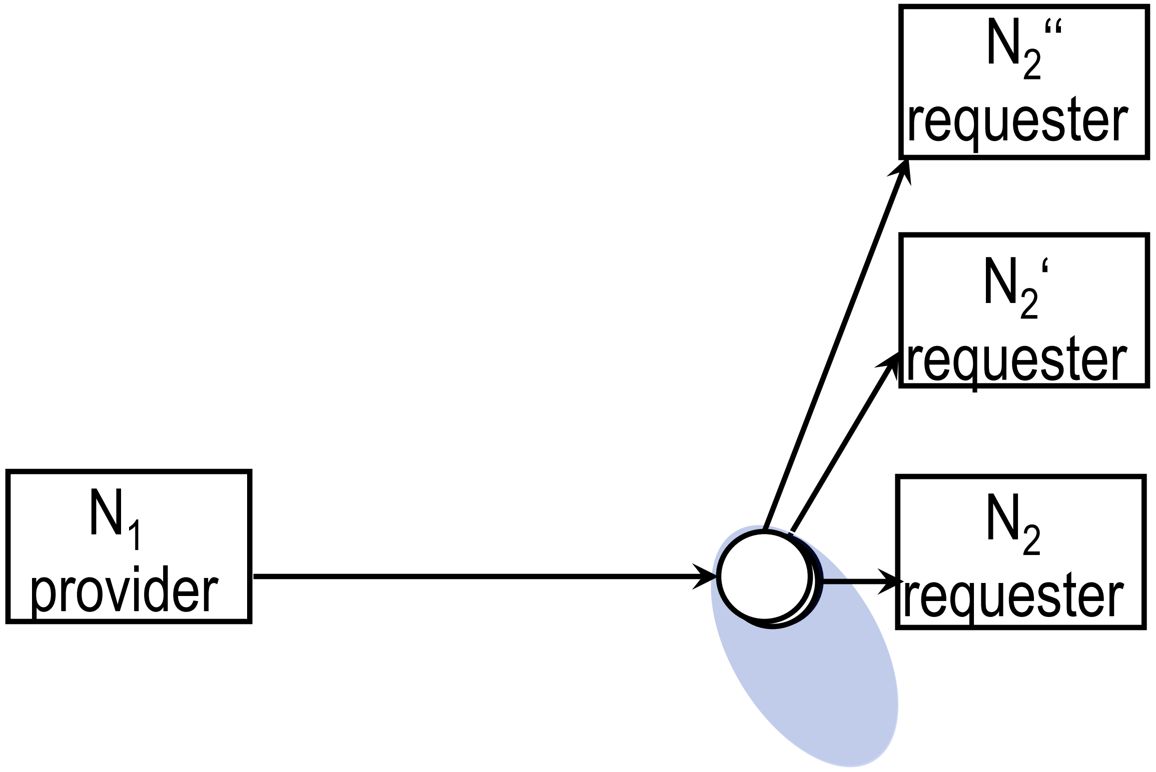
second sharing requester



second sharing requester



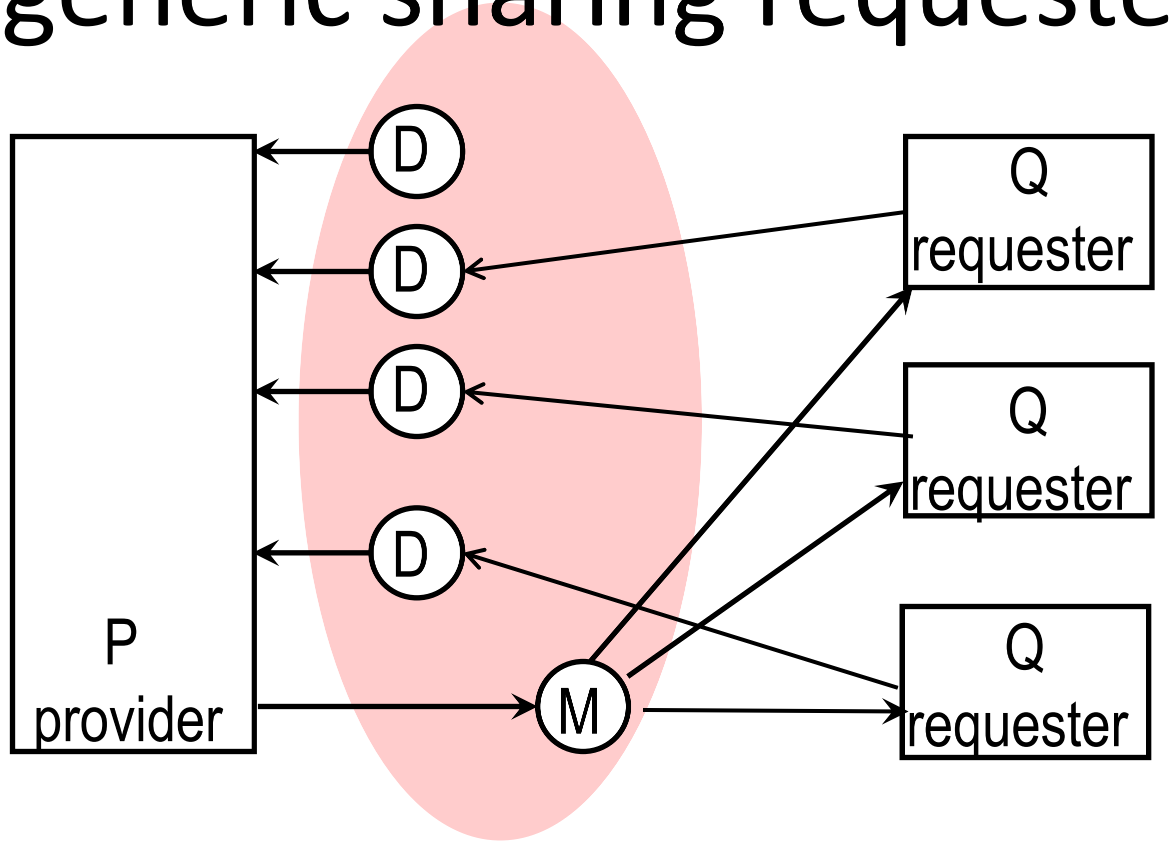
third sharing requester



skip the primes:

N_1 👓 N_2 👓 N_2 👓
N_2

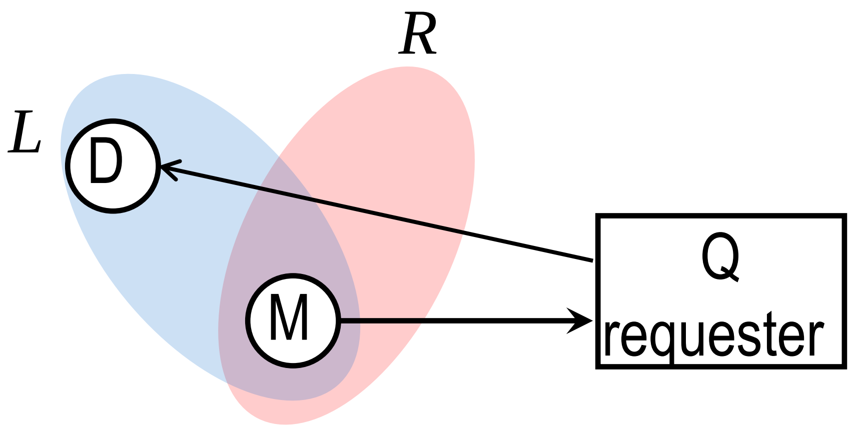
generic sharing requesters



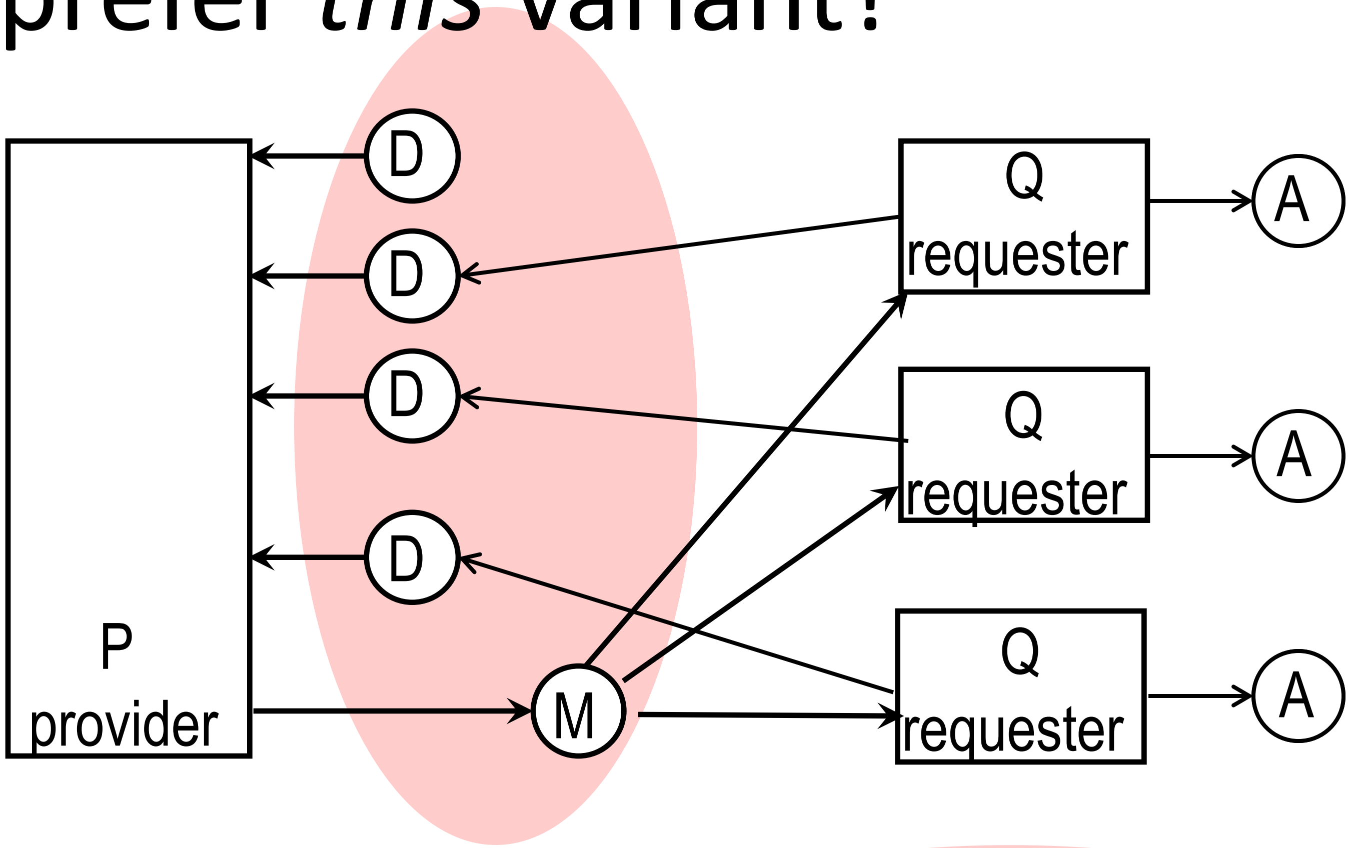
P 👓 Q
👓 Q👓
Q
P 👓 Q
👓 Q

P 👓 Q

generic reques-ter Q :



prefer *this* variant?



P 👓 Q

👓 Q👓

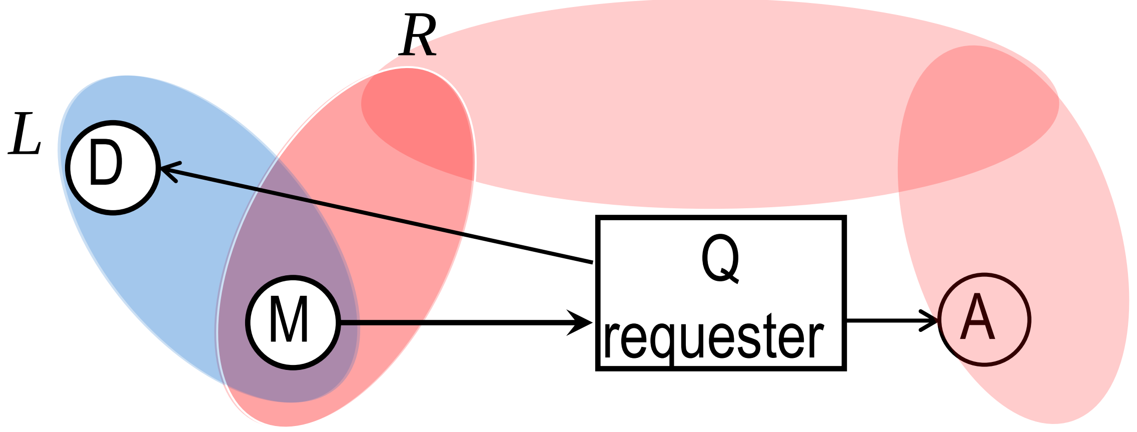
Q

P 👓 Q

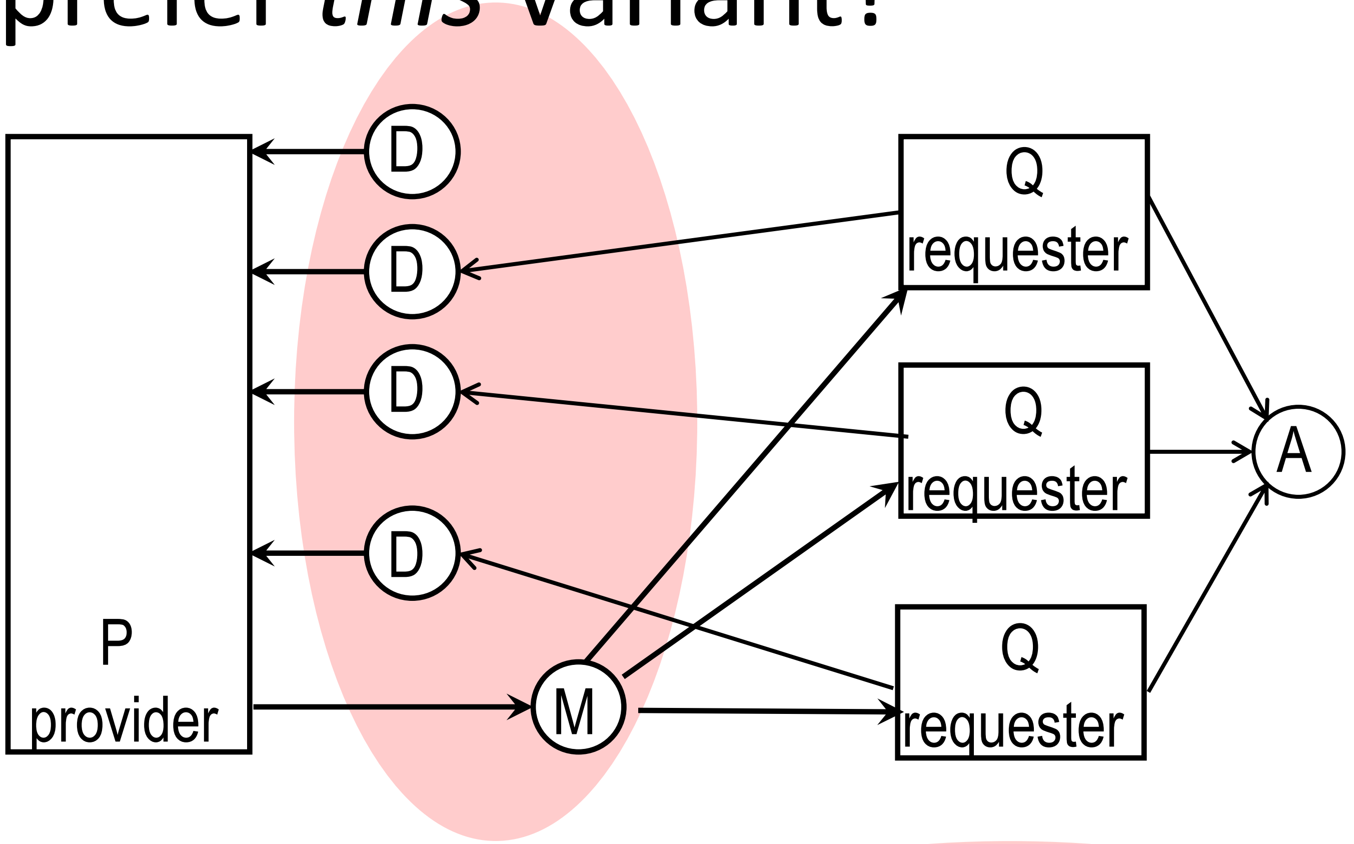
👓 Q

P 👓 Q

generic reques ter Q :



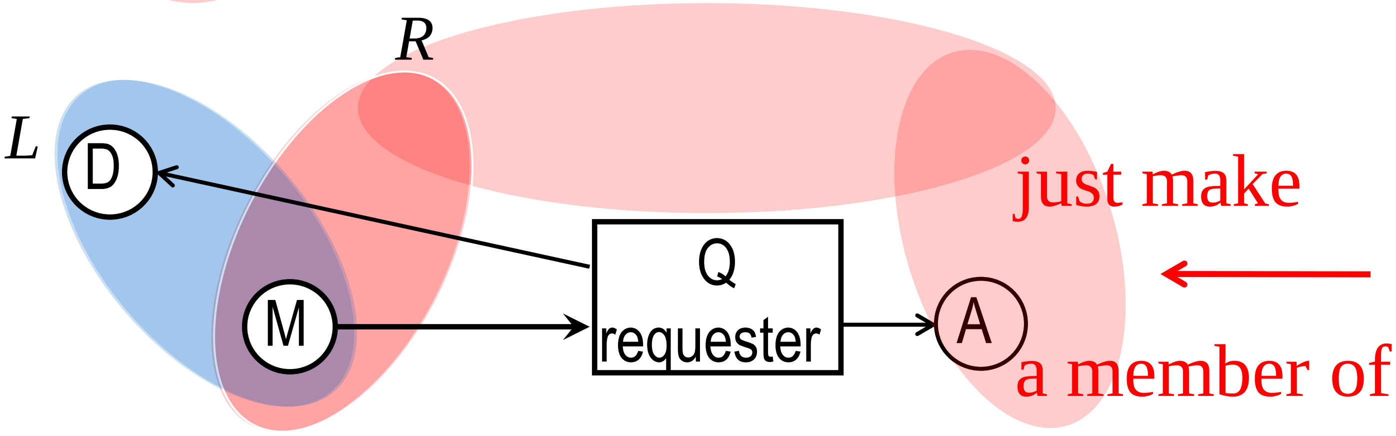
prefer *this* variant?



P 👓 Q
👓 Q👓
Q
P 👓 Q
👓 Q

P 👓 Q

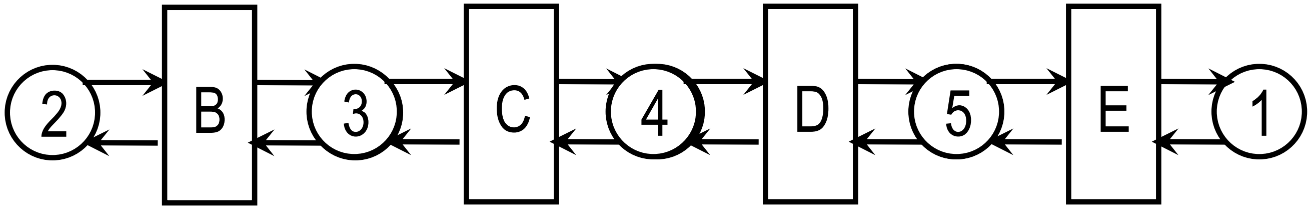
generic reques ter Q :

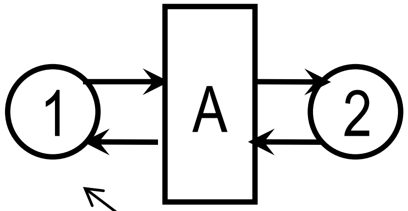


just make

a member of *L*

Cyclic composition: The philosophers



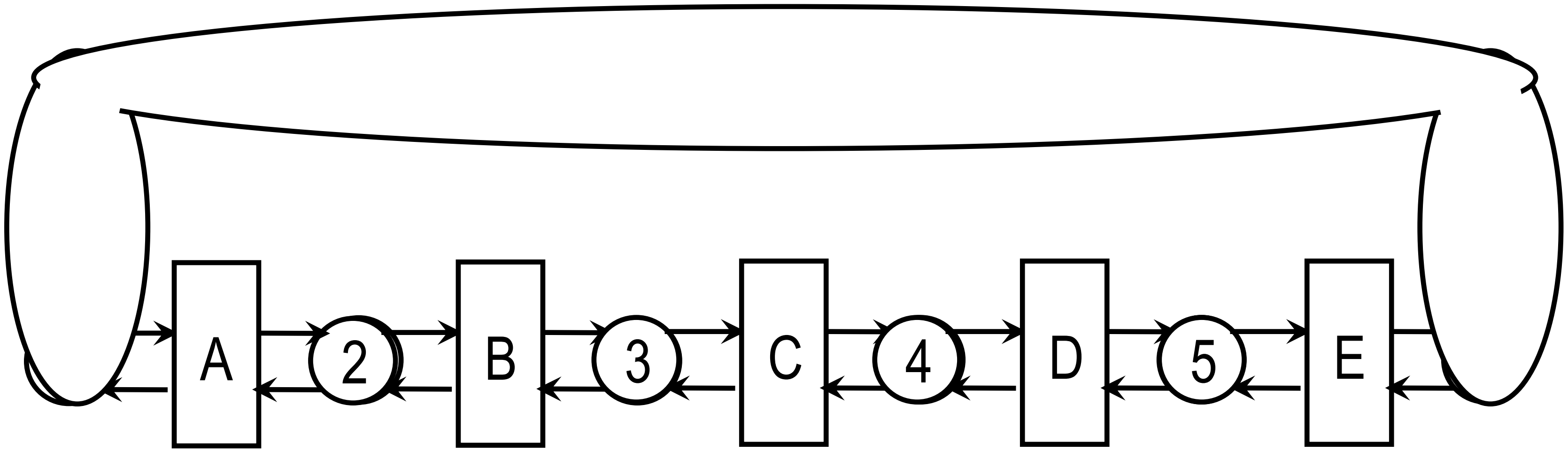


This is A👓 B 👓 C👓 D👓 E

The problem: How glue ?

Construct the *closure* (A👓 B 👓

Cyclic composition: The philosophers



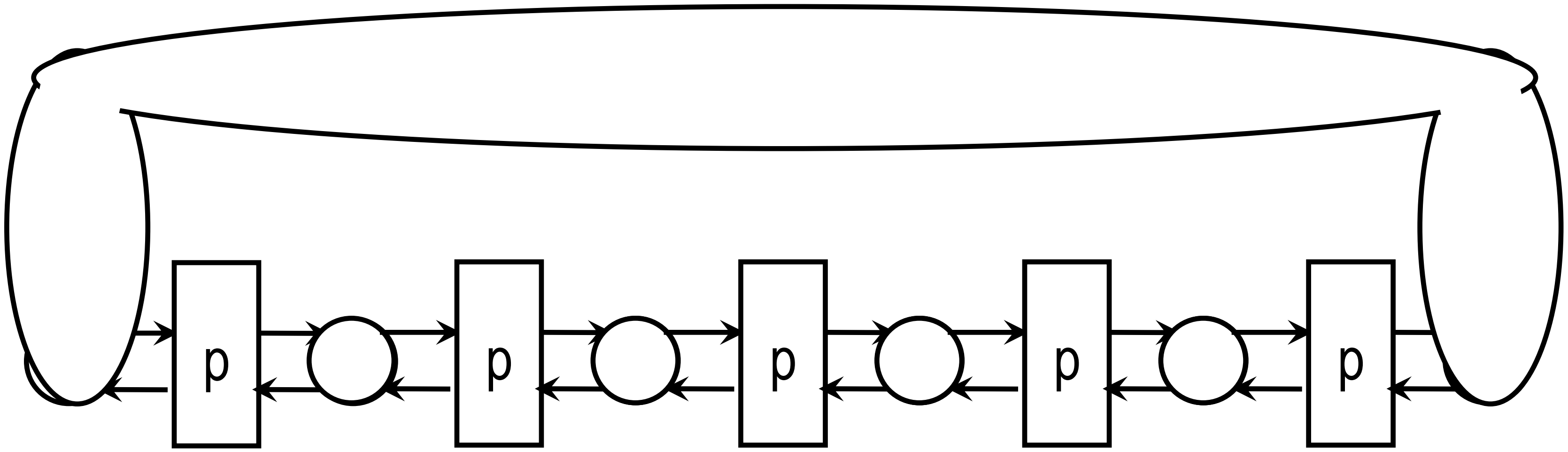
This is A👓 B 👓 C👓 D👓 E

The problem: How glue ?

Construct the *closure* (A👓 B 👓

C D E)^c

... with a generic philosopher



algebraic form: (p👓 p 👓 p👓
p👓 p)^c

The algebraic structure of services

Given:

- a set $\mathbb{S}$ of *services,*
- an associative *composition* operator $\oplus : \mathbb{S} \times \mathbb{S} \bumpeq \mathbb{S}$,
- a unary closure operator, $(\ )^c$
- a set Q of *requirements* $\rho_1, \ldots, \rho_n \subseteq \mathbb{S}$.

This yields the algebraic structure

$$(\mathbb{S}; \oplus, (\ )^c, Q).$$

- *neutral element(s)*
- $((\ )^c)^c = (\ )^c$

Study its algebraic laws!

Extend/refine the structure conservatively!

Build your systems accordingly!

Squeeze it all into tools!

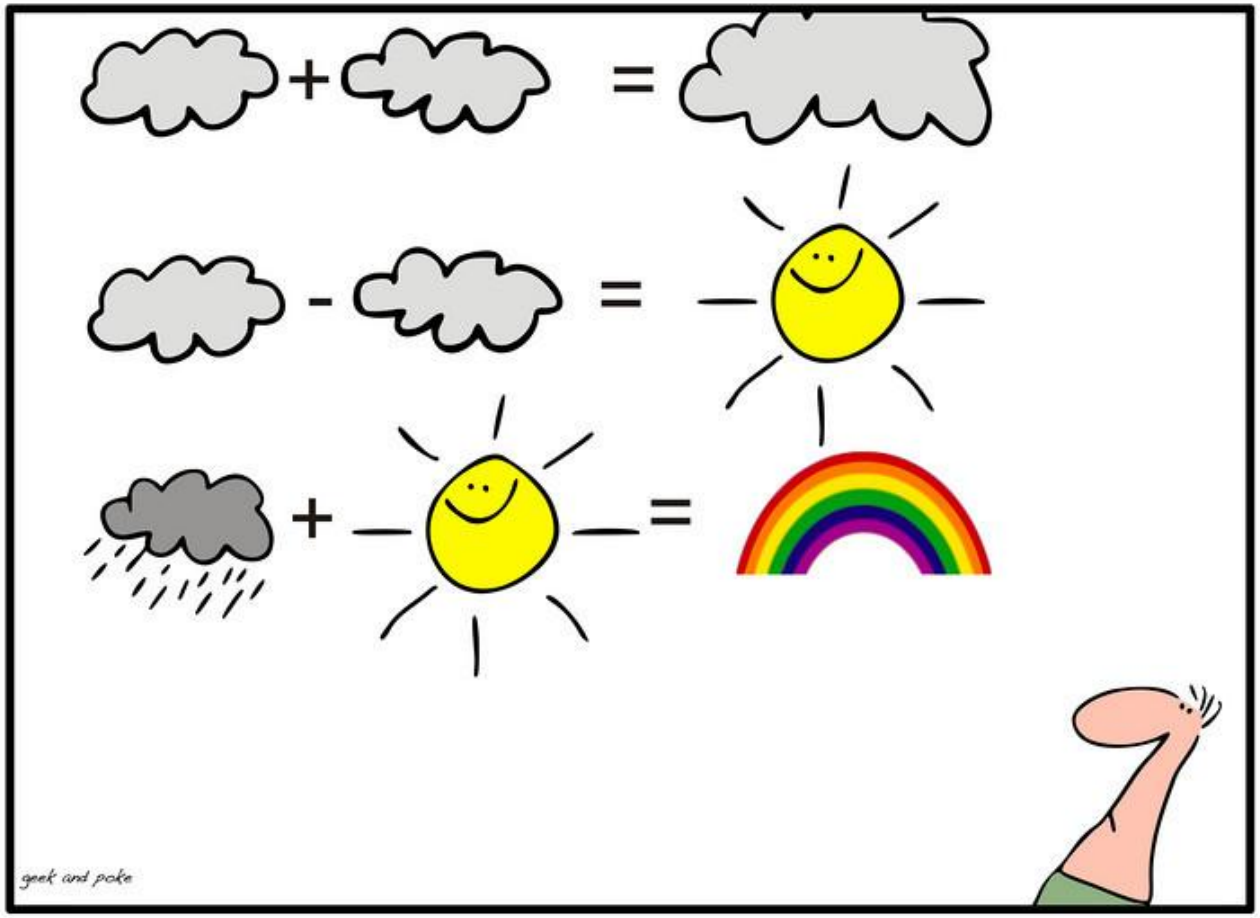
… on your request

Don’t like labels at all? Do with ordered ports.

Prefer *one interface* instead of *two ports*? Take L = R.

However:
Order without labeling,
interface without two ports:
both not too expressive!

The algebraic structure of clouds



CLOUD COMPUTING

ICTERI

Kiev, June 24, 2016

the end

Service Orientation as a paradigm of computing

Wolfgang Reisig

Humboldt-Universität zu Berlin

1. Aspects that exceed classical Theoretical Informatics
2. Towards a Theory of Services
3. Composing *many* services

Theory of Programming

Prof. Dr. W. Reisig