Moscow HSE Nov 11, 2013

Service Oriented computing: Challenges and ideas to meet them

Wolfgang Reisig

Humboldt-Universität zu Berlin

Institut für Informatik
Humboldt-Universität zu Berlin

Theory of Programming

My background

currently:

A PhD school on service-oriented Architectures for the Integration of Software-based Processes, exemplified by Health Care Systems and Medical Technology

SOAMED

DFG Deutsche Forschungsgemeinschaft

***B**erlin - **E**indhove* [illegible] **Technology Program**

... to offer SOC a tool supported foundation

B.E.S.T.

Distributed systems

Reactive ~

Interactive

Open ~

How construct ?

Formal methods

Verification

Tools

Embedded systems

Technical environment

Continuous time

Message loss

Battery power

Milliseconds

SOC

Business environment

Long running processes

Ownership

Privacy

What I intend to speak about

1. Views on SOC
2. The SOA Triangle
3. Actual Challenges

4. A systematic approach to SOC
5. A subtle observation
6. An aspect of composition

1. Views on SOC

A business view on SOC

A technical view on SOC

A conceptual view on SOC ⟵ *my main topic*

A business view on SOC

“*THE* most relevant emerging paradigm”

“A substantial change of view
as it happens at most once each decade”

“The next fundamental software revolution after OO”

“Much more than just an other type of software!”

“The foundational layer for
tomorrow's information systems”

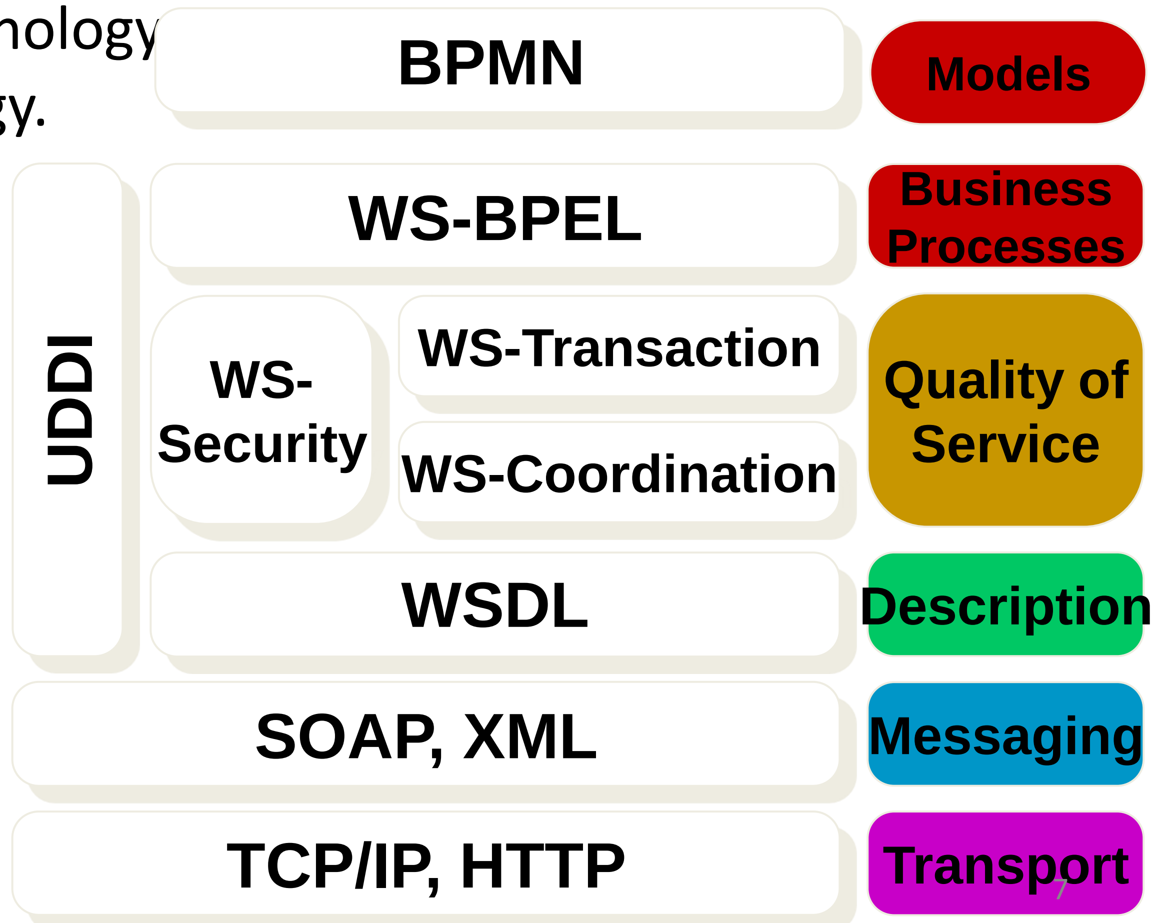
A technical view on SOC

during recent 10 years,
driven by software industry,
based on

- business process technology
- web service technology.

Composed fom
known technologies
in a technology stack.

a typical SOC stack:



A conceptual view on SOC

imperative programing

Object Orientation

A conceptual view on SOC

Service Oriented Computing

A conceptual view on SOC

The Cloud

… not only software

a service may be offered by:

- a software component *books a seat*
- a technical system *provides cash*
- an organization *delivers a pizza*
- a person *informs at the help desk*

Paradigms of Computing	Conceptual Foundations
1960ies: conventional programming	computable functions
1980ies: OO	Model Theory, Algebraic Specifications
2000ies: SOC	nothing! made by industry!
Advantage: quickly and widely accepted.	*Disadvantage:* no unique terminology, no formal analysis, no specific verification, ...

2. The SOA Triangle

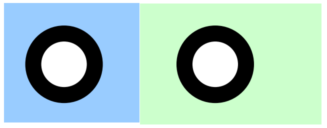
Service
a component with an *interface*

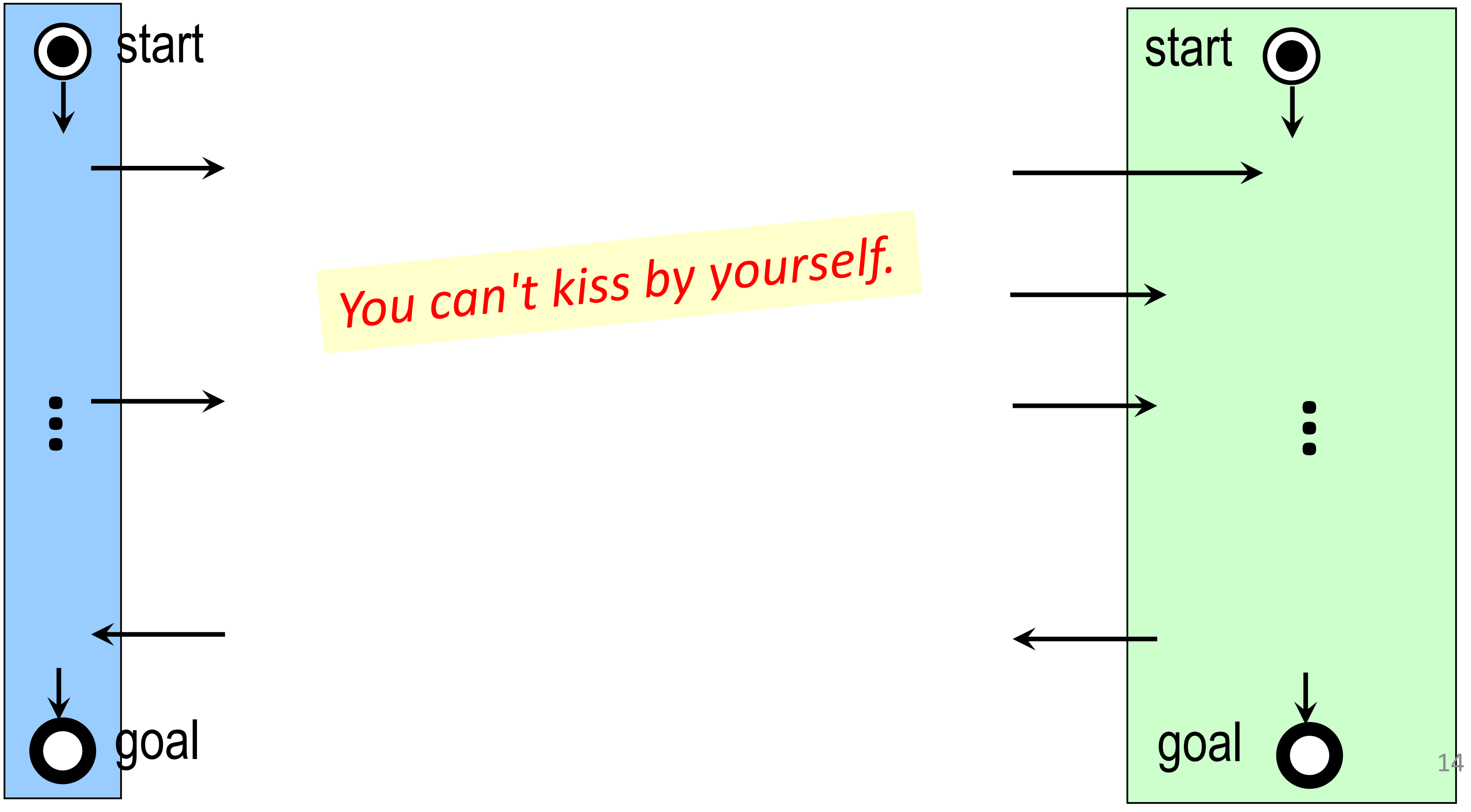
I sell chairs.
I talk to my clients along my interface.

I want to buy a chair.
I talk to sellers along my interface.

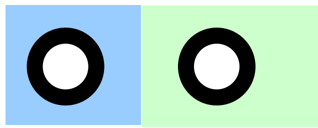
Composition
Two services communicate along their interfaces.

partners may reach a ***goal***

to reach 

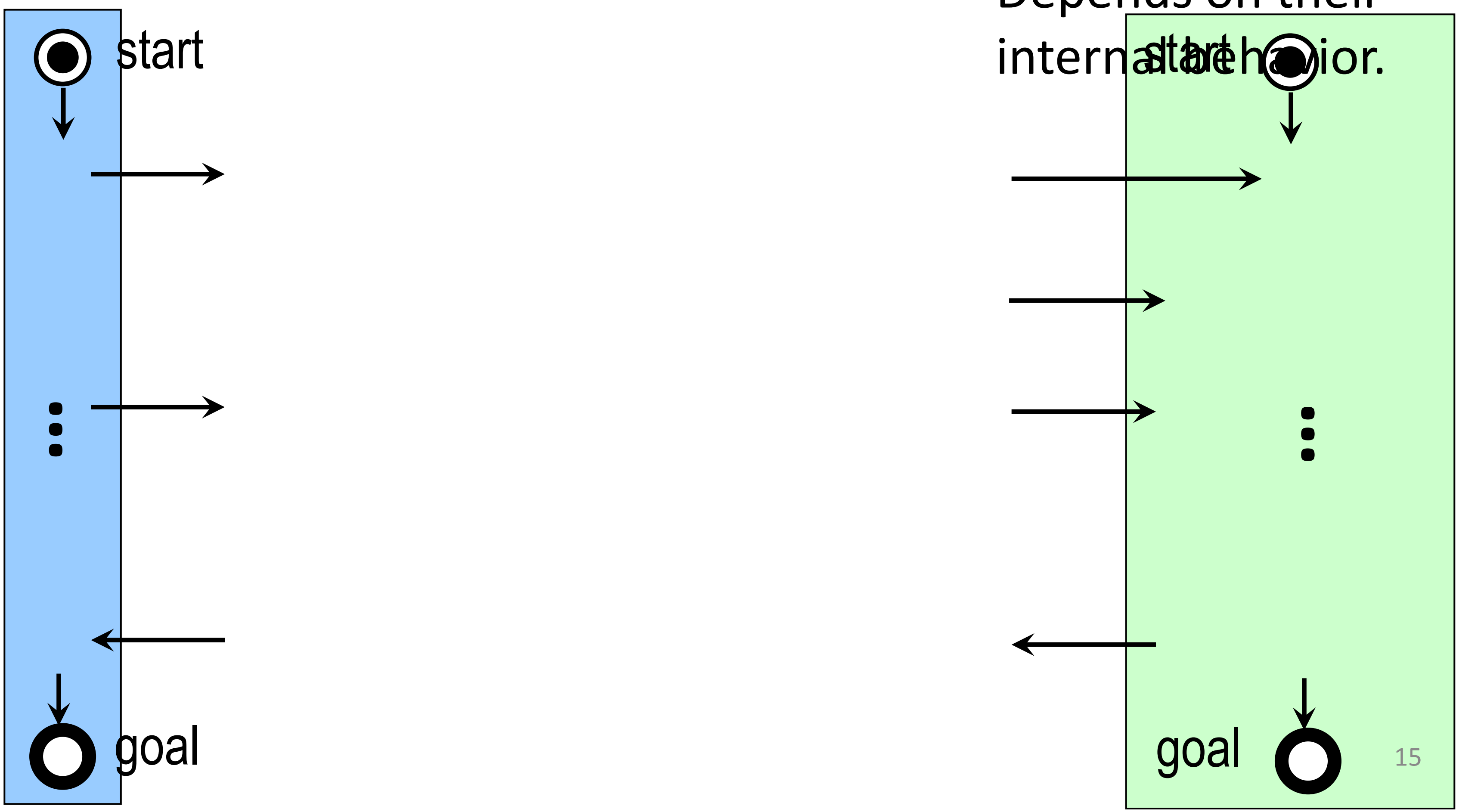


partners may reach a ***goal***

to reach 

Jointly they may reach their goal.

Depends on their internal behavior.



Special case: SOA

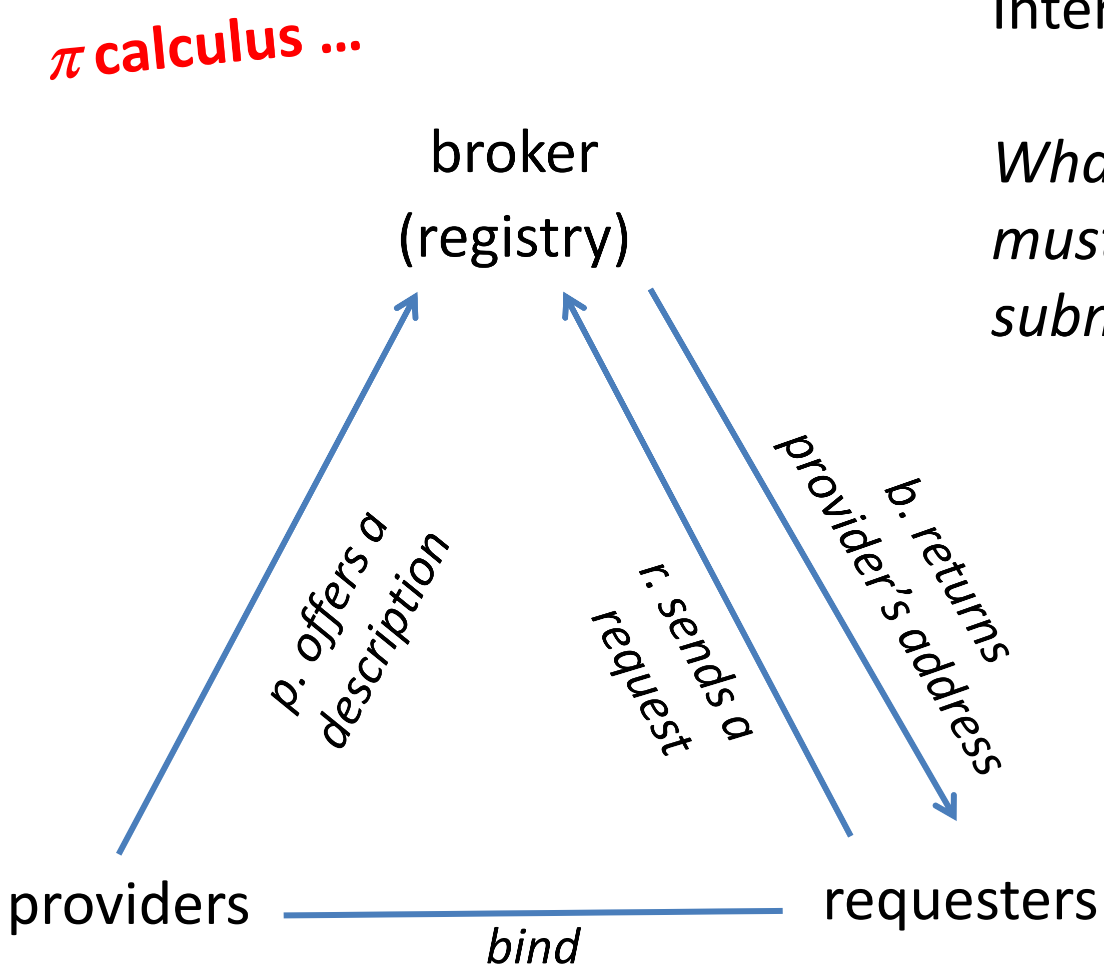
Problem: How can I find a partner?

Provider: *I sell chairs. How find a buyer?*

Requester: *I want to buy a chair. How find a seller?*

Broker: *Requester asks broker for a provider.*
Broker offers him a provider's address

The SOA triangle



Interesting problem:

What information
must requesters and providers
submit to the broker?

3. Actual Challenges

How cope with

- instantiation
- refinement (horizontally, hierarchically)
- correctness
- substitution
- equivalence
- orchestration
- choreography
- brokering
- fault handling
- compensation handling
- design methodology
- compositionality

... questions on
fundamentals of software engineering

SOC *in the cloud*

- Who is responsible for a provided service?
  Legal department? Technical proxy?
- Reliability of a service also depends
  on the reliability of the cloud provider
- Resilience guaranteed by
  the service provider or the cloud provider?
- How transparent is the cloud location to the requesters?
- Open for everyone?
- Elasticity
- Latency for users

Requesting services *from a cloud*

- How can a requestor be sure
  the provided service
  meets his quality standards?
- Who is responsible for privacy protection?
  provider, broker, requester?
- How can the broker ensure
  a predictable uptime of a service?
- Who is allowed to act as a provider?
- What happens if a service
  is retired or changed?
  Will potential requestors even know?
  Regulated by contract?

Requesting services from a *public cloud*

- State of the art: manual selection
- Contract if the service is business critical
- Consuming a cloud service takes considerable ramp-up time
- Who owns the service?
- Cost of service and other metadata known to broker ?
- New compliance challenges (data location etc.)

  might require new rules for consumption

  (forbid e.g. for personal data)

Brokering services *in a cloud*

The broker:

- Which services do I know about?
- How are they related?
- How do I find services from given requester requirements?
- May I offer a composed service, extended by an adapter?

- Which details about the services description, semantics, constraints, capabilities must I store ?
- How do I cope with non-functional properties such as SLA/QoS ?
- How do I cope with security information ?
- How can I guarantee availability ?

4. A systematic approach to SOC

Typical text books on SOC
explain concepts and notions colloquially.

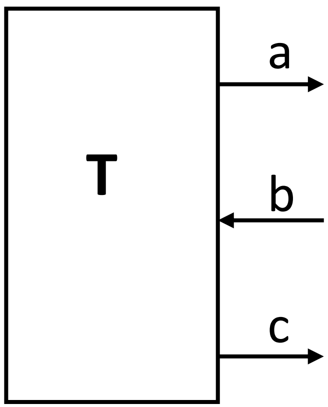
- *"… SOA is an implementation independent concept, …"*
  many notions, poorly related

- show implementations that mix
  substantial and accidental aspects

How improve this?

Use abstractions, ***models.***

Open System



... a transition system
with *channels*
for *asynchronous* communication
with *its environment*.

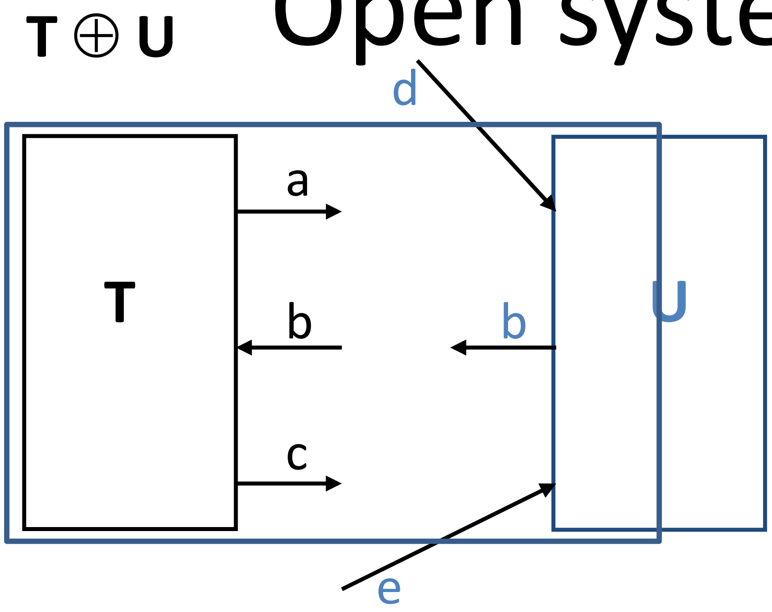
Semantics of **T:**
During a computation,
each channel carries
a stream of data.

Semantics:
a relation on streams

Fundamentally new aspects:

- Infinite runs are sensible.

Open systems are *composed*



... a transition system

with *channels*

for *asynchronous* communication

with *its environment*.

Composition **T ⊕ U**
has pending channels.
Is an open system again.

Fundamentally new aspects:

- Infinite runs are sensible.
- Environment is not trivial,
  deserves its own attention.

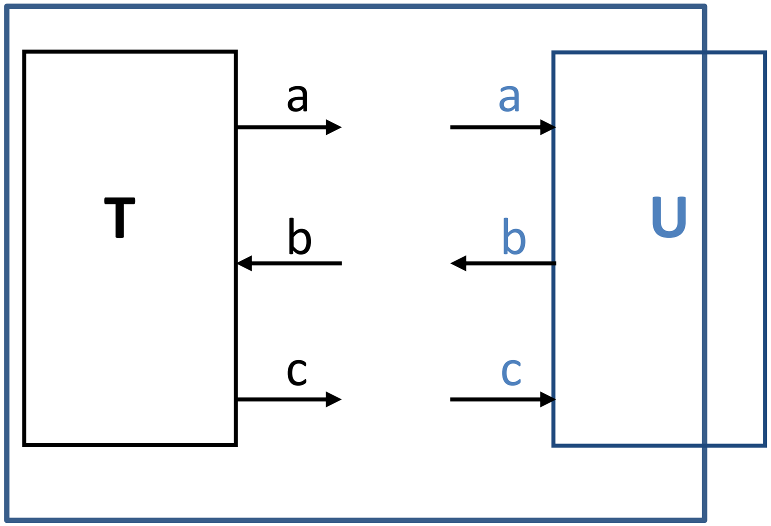
Idea:

! The environment is
an open system, too!

Compose system with environment!

Couples

T ⊕ U



... a transition system

with *channels*

for *asynchronous* communication

with *its environment*.

T and **U** form *a couple*:
channels fit perfectly.

T ⊕ U is a
classical transition system
(with internal channels)

Fundamentally new aspects:

- Infinite runs are sensible.
- Environment is not trivial,
  deserves its own attention.

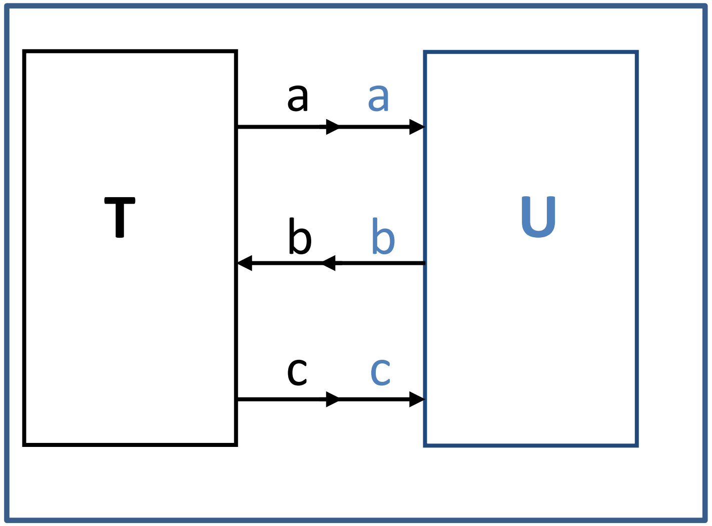
Idea:

! The environment is
an open system, too!

Compose system with environment!

T ⊕ U

Requirements at a couple



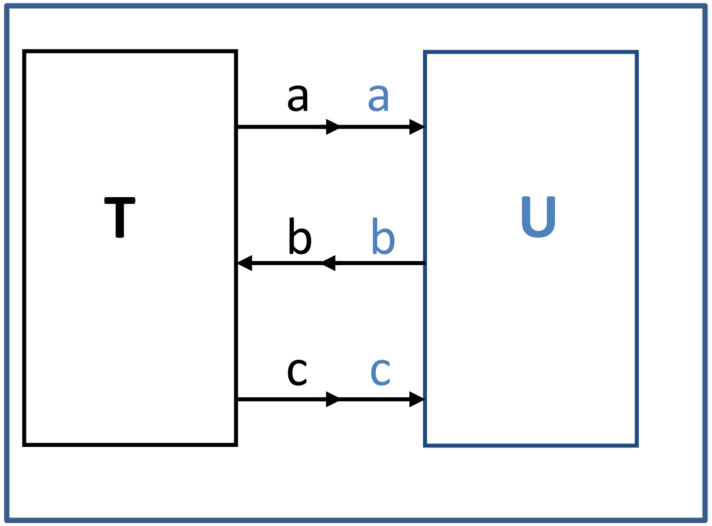
	... as CTL* formulas:
T and **U** communicate boundedly	*AG n-bounded*
T and **U** communicate responsively	*AGEF responsive*

With *target* states:

T ⊕ **U** weakly terminates	*AGEF terminal*
T ⊕ **U** is deadlock free	*AG (terminal ♎ target)*
T ⊕ **U** is livelock free	*AGEF target*

Requirements at a couple

T ⊕ **U**



Def.: A *requirement* **R** is a set of couples … up to bisimulation.

Def.: **U** is an ***R**-partner of **T*** iff **T** ⊕ **U** $\in$ **R** .

T and **U** communicate boundedly

T and **U** communicate responsively

With *target* states:

T ⊕ **U** weakly terminates

T ⊕ **U** is deadlock free

T ⊕ **U** is livelock free

Interesting Problems:

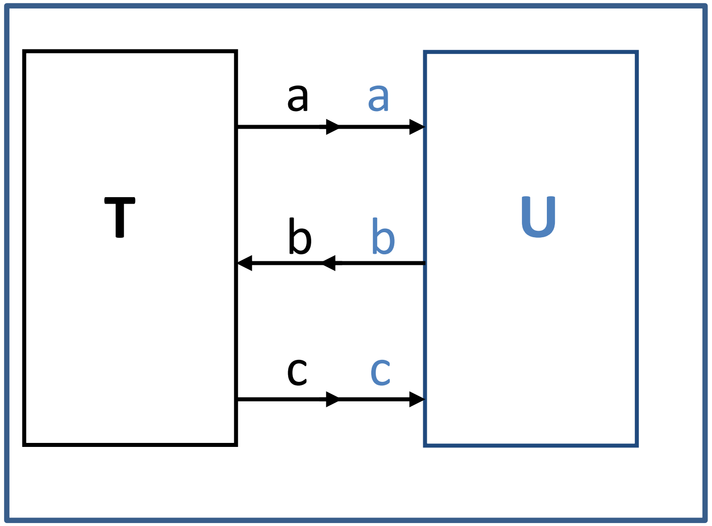
Discovery

Adapter generation

Substitution

Coping with *ALL* **R**-partners

T ⊕ **U**



Observation:

There exists *a most comprehensive* **R**-partner of **T** , *mcp*(**T,R**):

For each **R**-partner **U** of **T** holds:

tree(**U**) is a subtree of *tree*(*mcp*(**T,R**)).

... for all „interresting" R

Idea:

Construct *mcp*(**T,R**).

Discovery: *mcp*(**T,R**).

Adapter generation for T and U:

Discovery for **T** ⊕ **U**

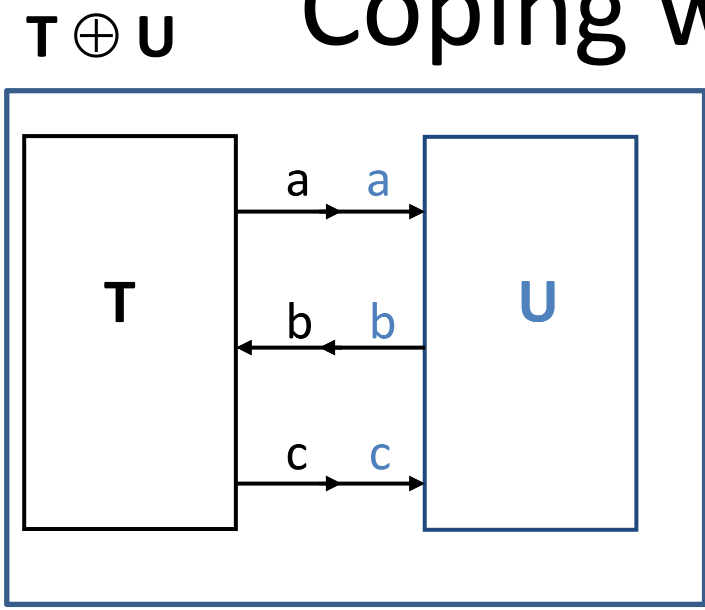
Interesting Problems:

Discovery

Adapter generation

Substitution

Coping with *ALL* **R**-partners



Observation:

There exists *a most comprehensive* **R**-partner of **T** , *mcp*(**T,R**):

For each **R**-partner **U** of **T** holds:

tree(**U**) is a subtree of *tree*(*mcp*(**T,R**)).

Substitution:

Inscribe *conditions* at *mcp*(**T,R**)
that characterize all the
trees of the **R**-partners, *mcpc*(**T,R**) .

Then compare *mcpc*(**T,R**) and *mcpc*(**T',R**).

Interesting Problems:

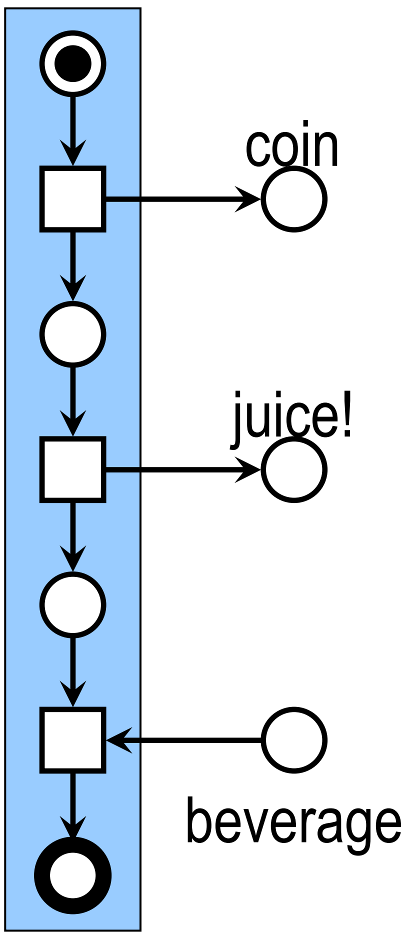
Discovery

Adapter generation

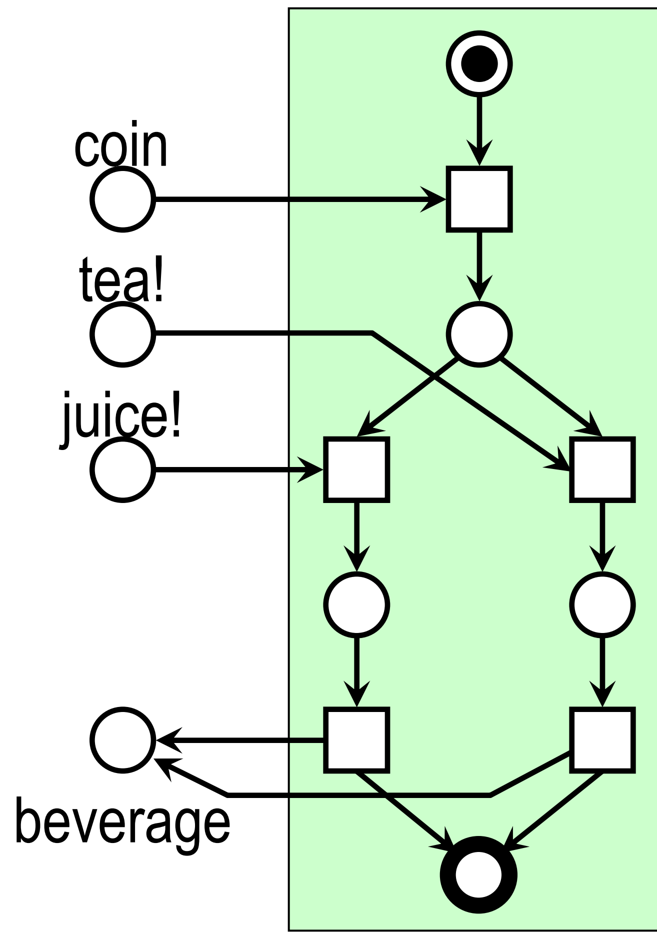
Substitution

5. A subtle observation

B: the juice buyer:



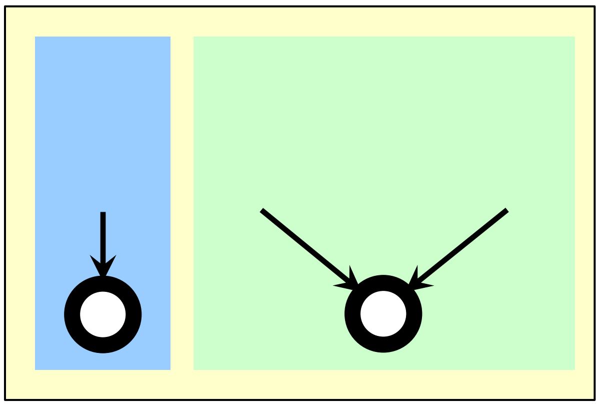
V: the vending machine:



The composed system

Requirement **R**:

B ⊕ V reaches

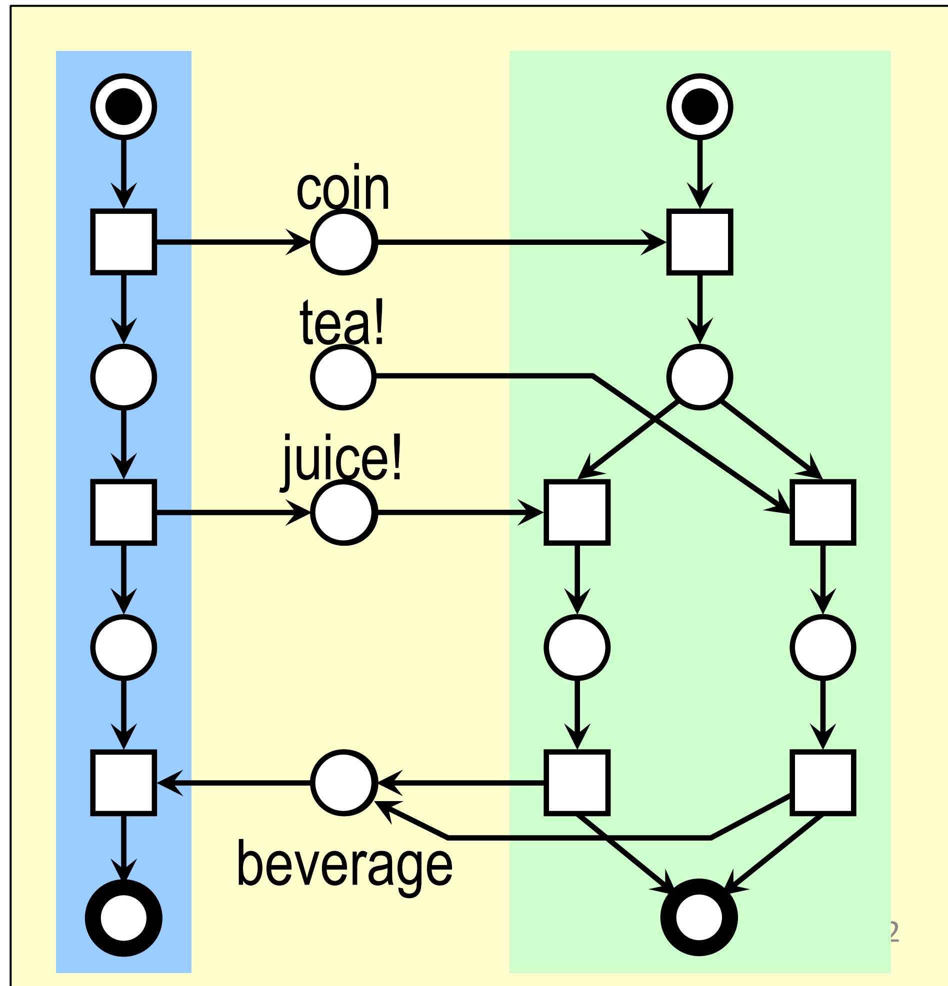


with empty interface.

Observation:

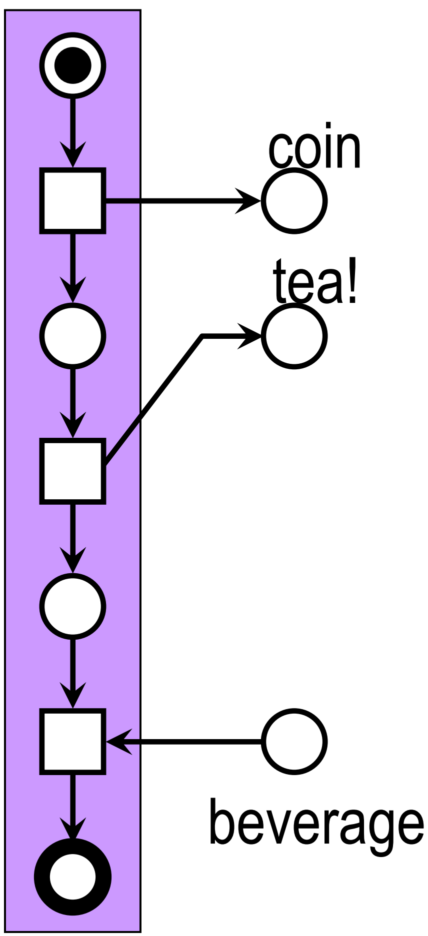
B ⊕ V is an **R**-couple

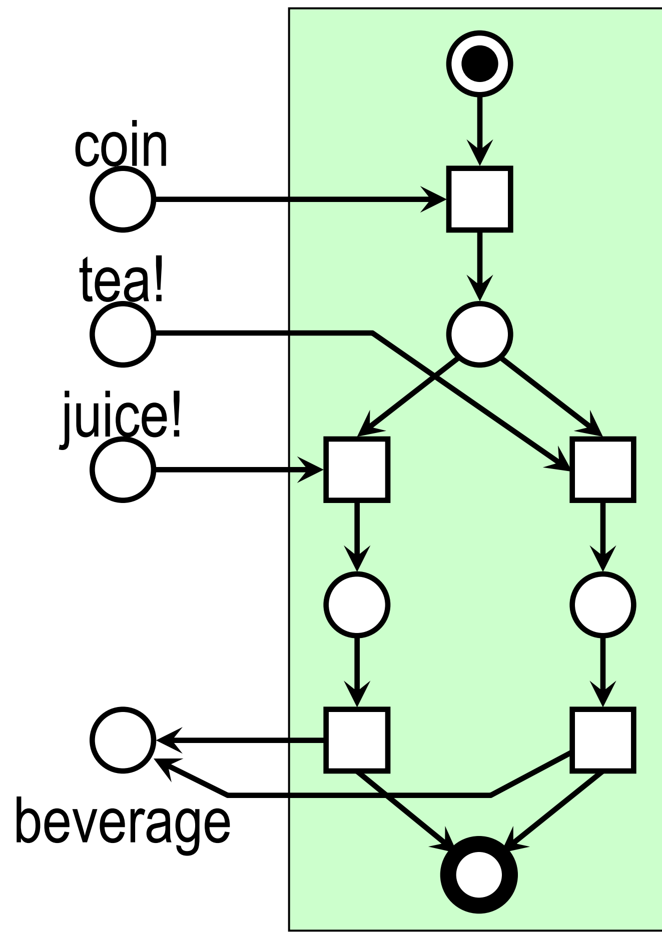
B ⊕ V



Another buyer

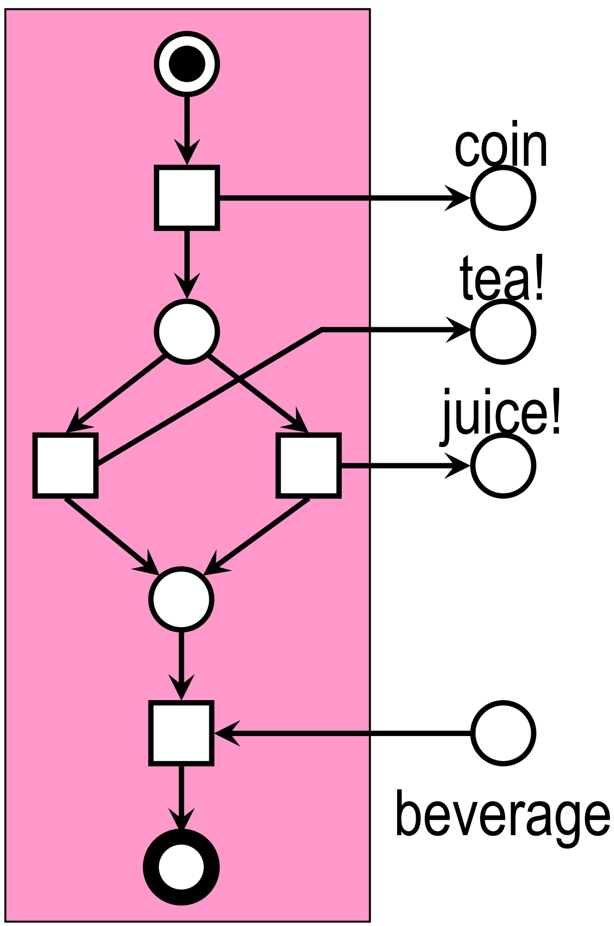
the tea buyer:

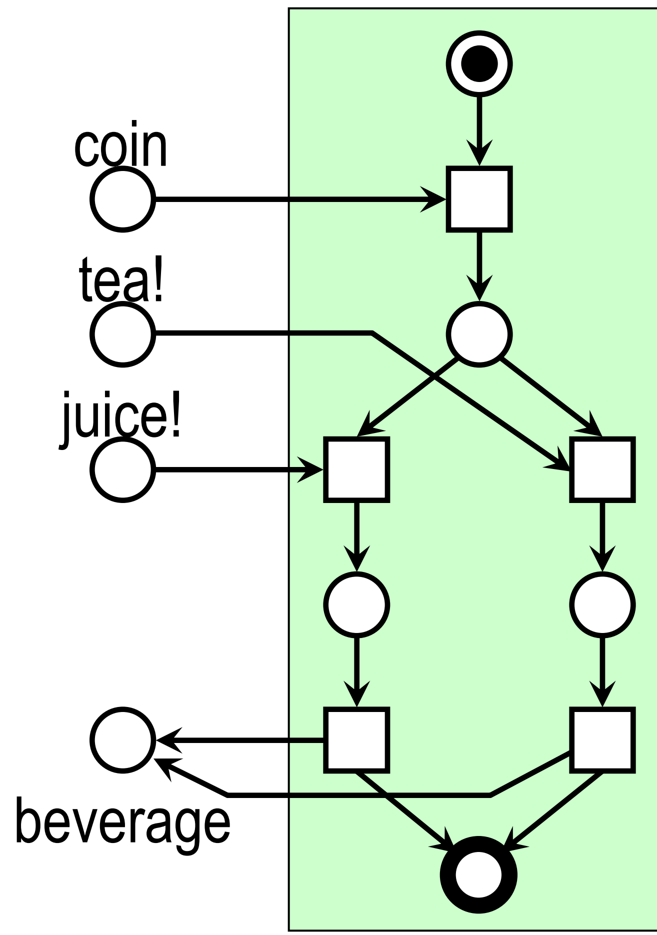




Are there more buyers?

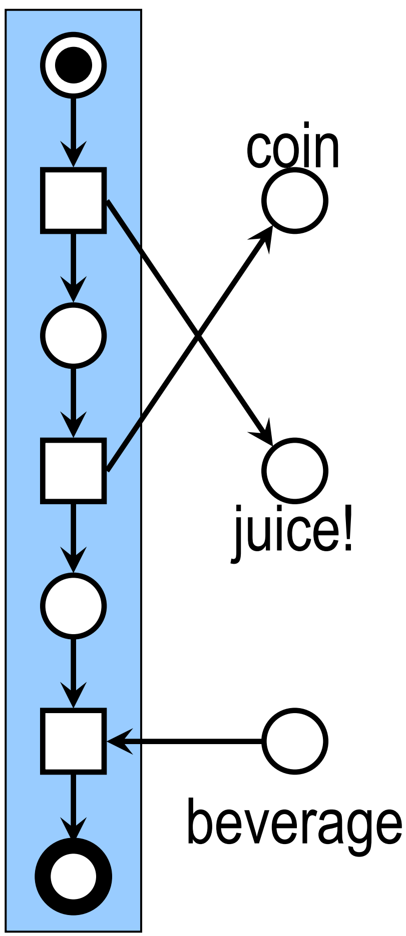
the juice -or-tea buyer:

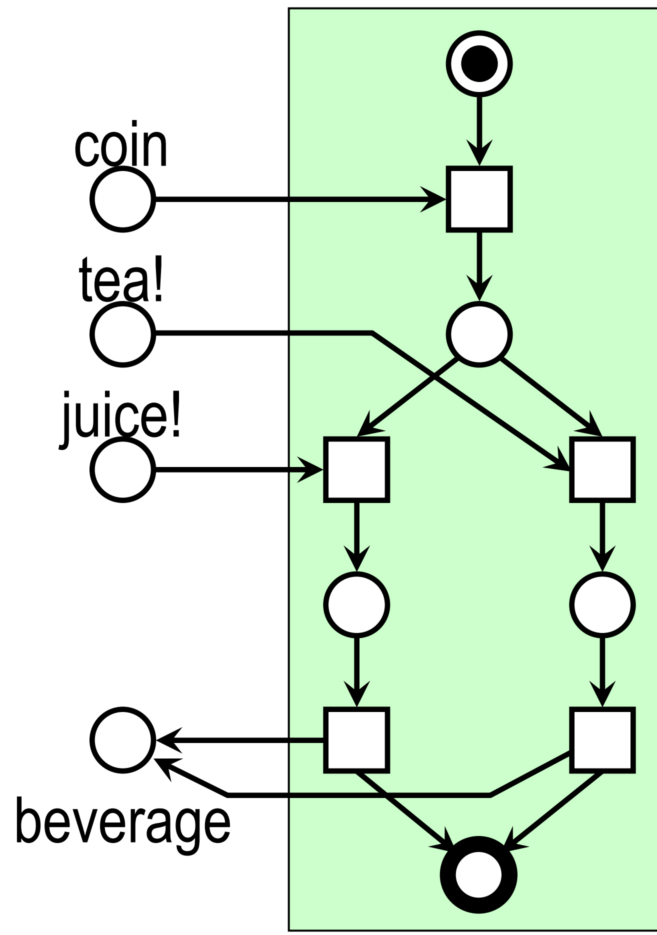




Swap the order

First juice! then coin



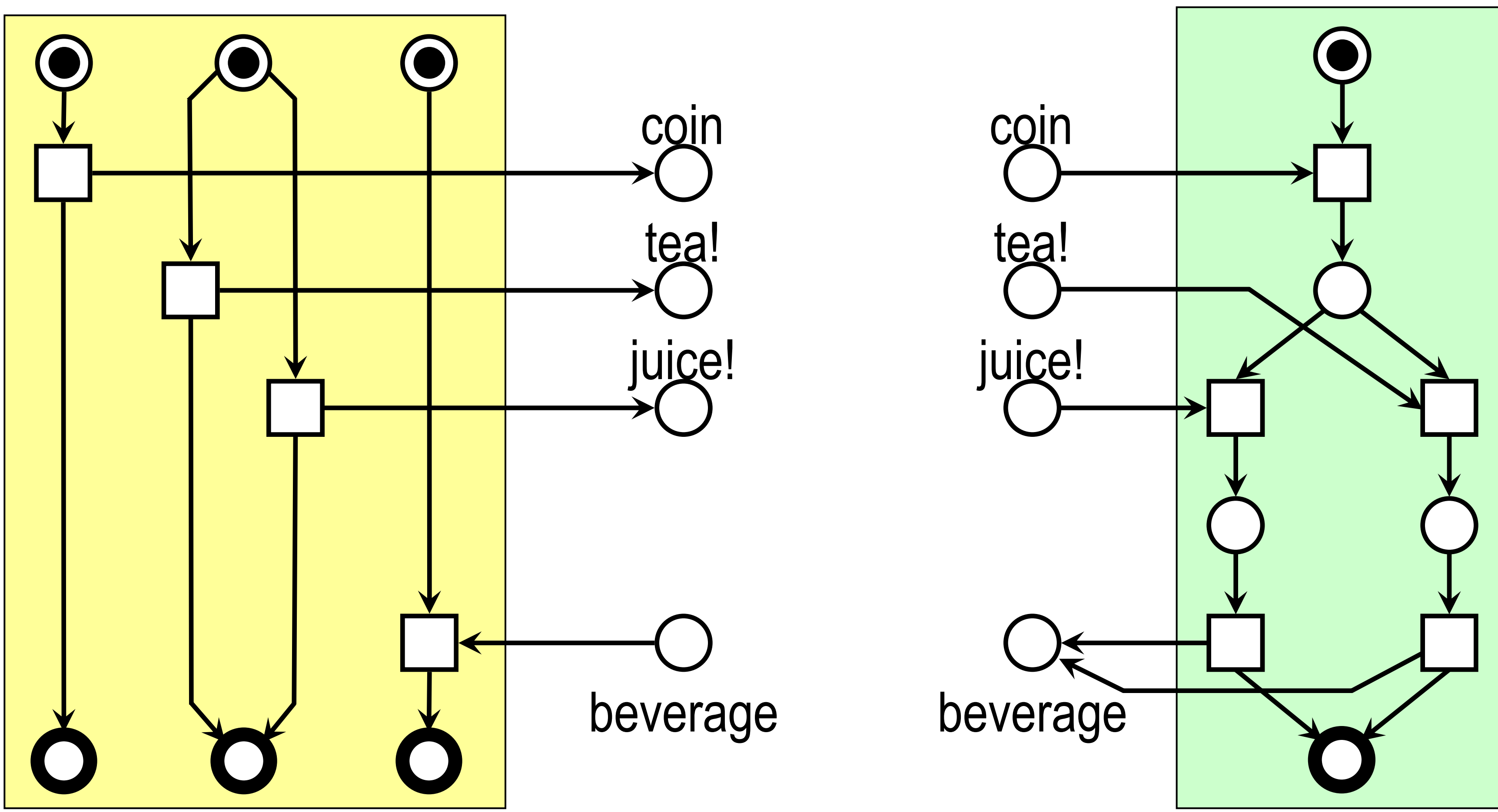


No sequential control

Three independent threads of control

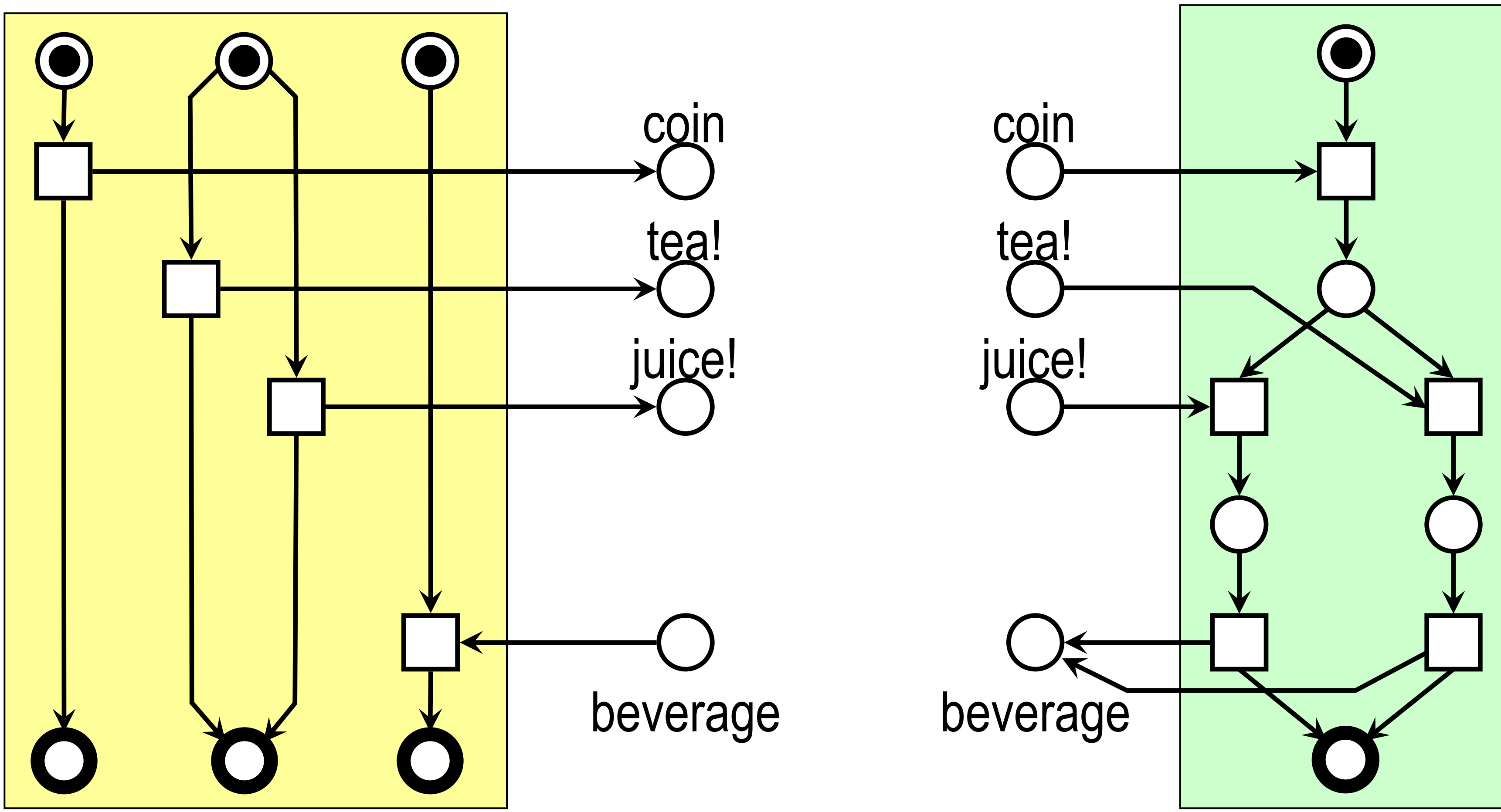
This is the most comprehensive buyer:

Each other buyer can be derived from this one.



New idea: *distributed* buyer

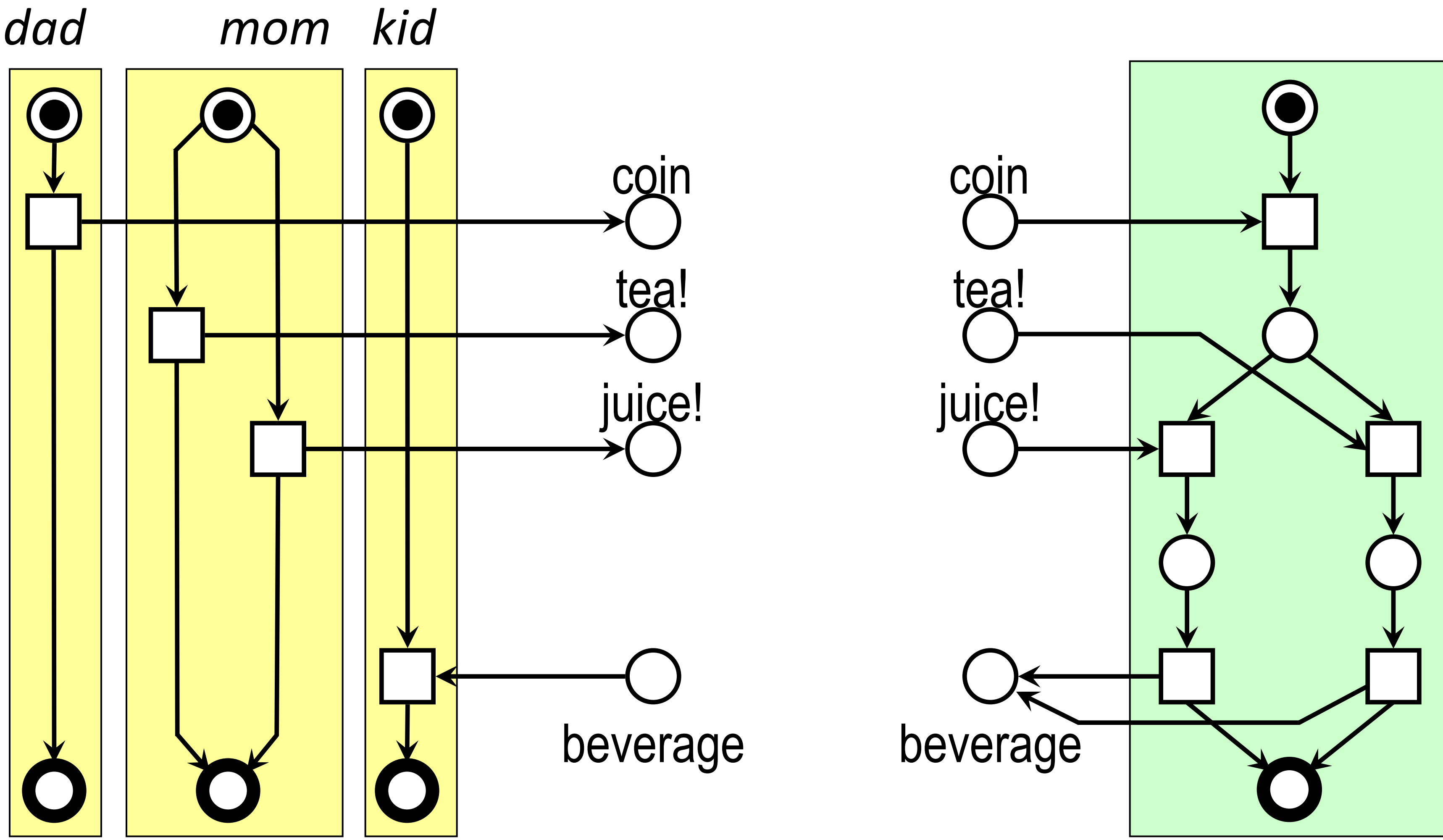
firstly, the partner disintegrates



Construct 3 services: *dad, mom, kid*

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

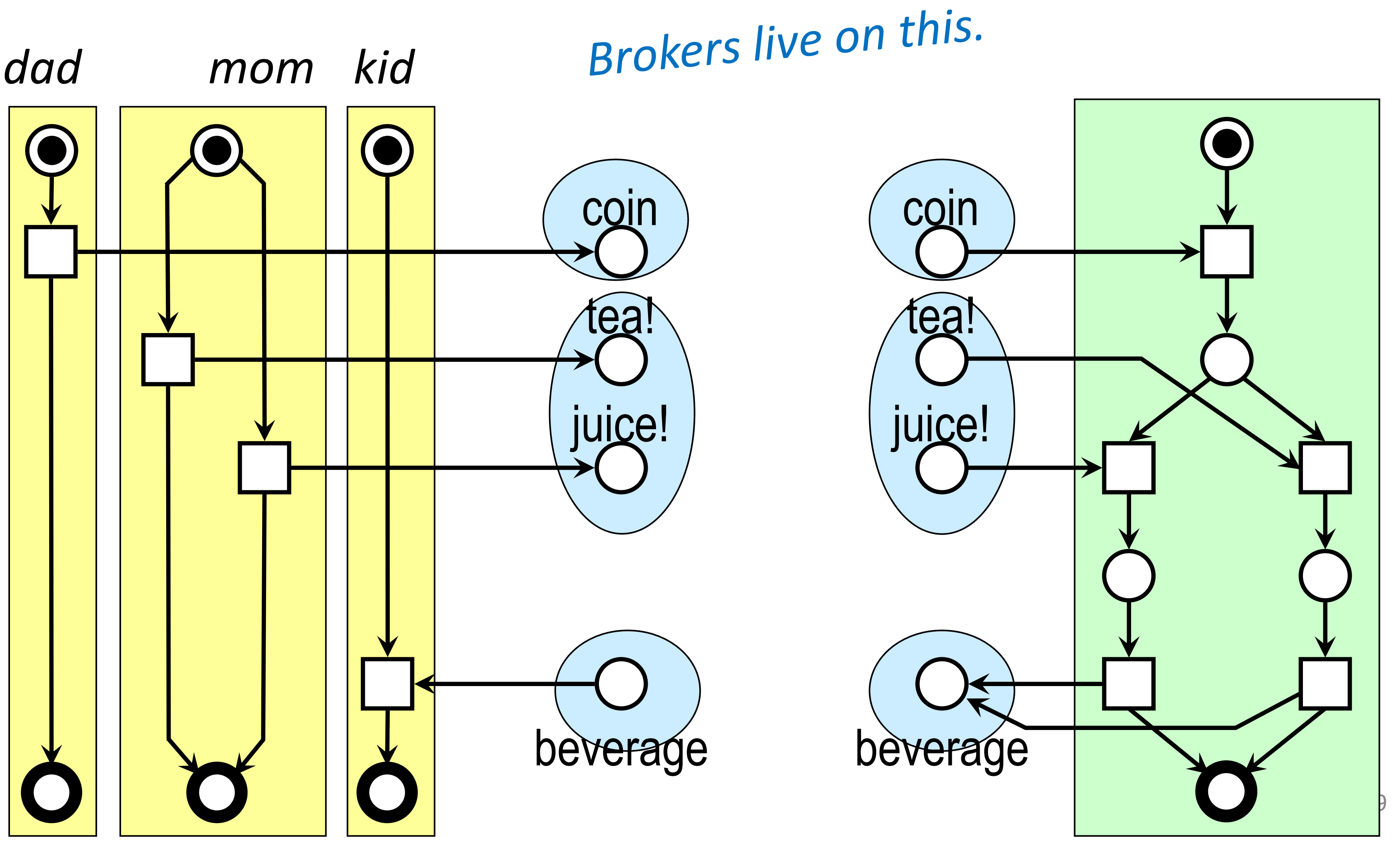
environment of the machine: $dad \oplus mom \oplus kid$



A service may connect *many* others

one at each *port*.

Observation: *dad* and *mom* need not to communicate.

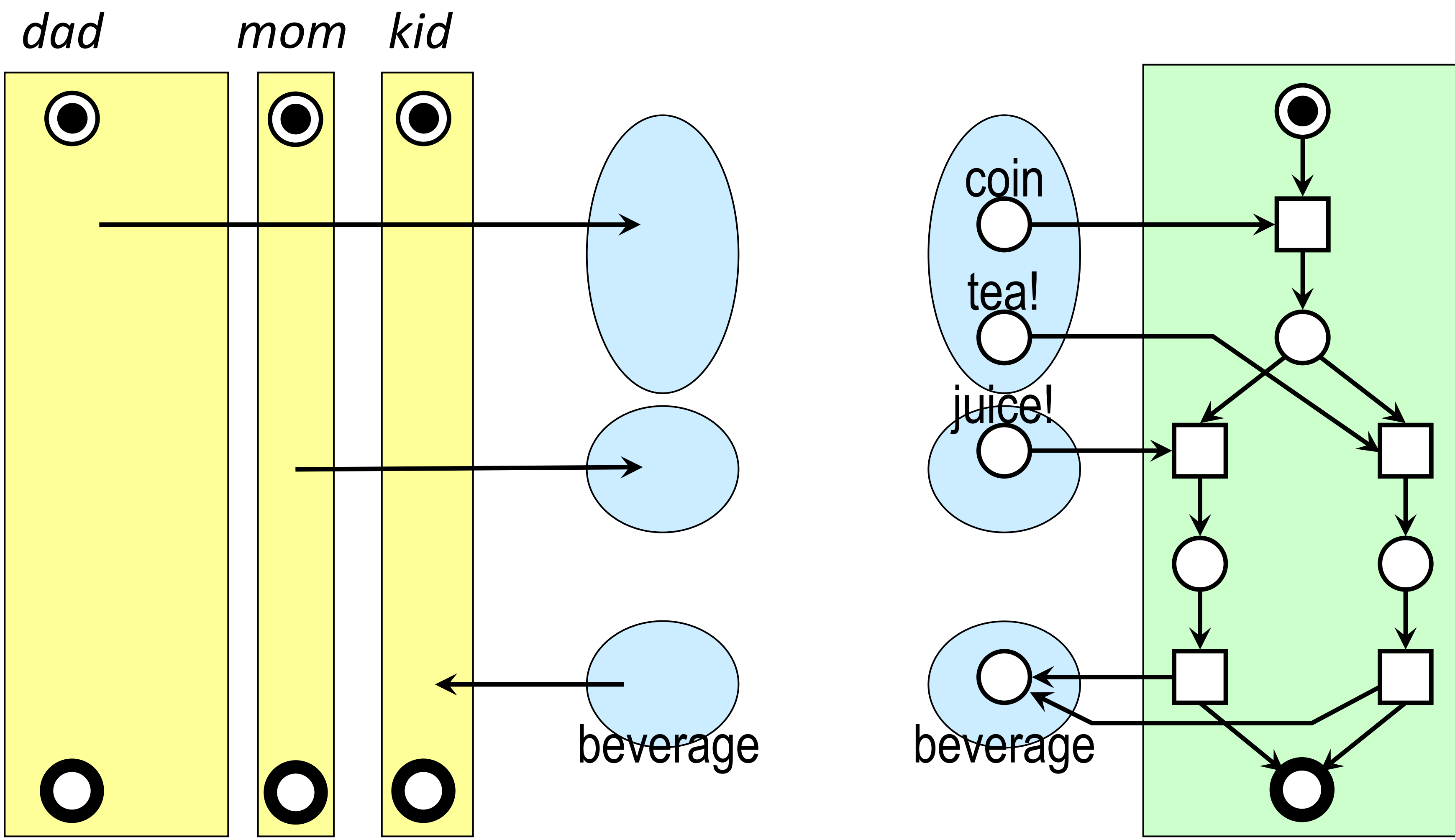


A strange choice of ports

... who orders a beverage ?

One way:

Observation: *dad* and *mom* must communicate.

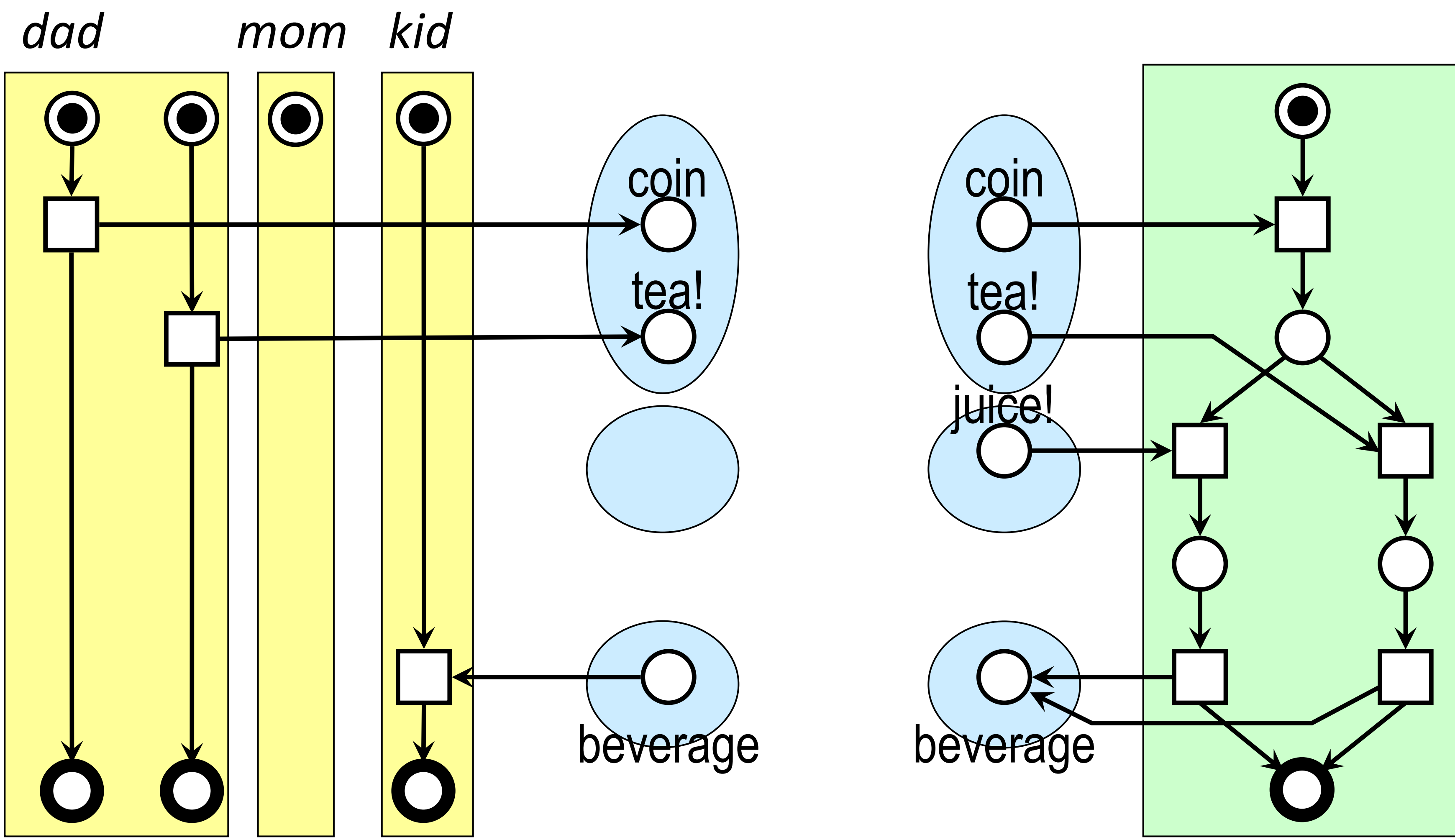


A strange choice of ports

... who orders a beverage ?

One way:

Observation: *dad* and *mom* must communicate.



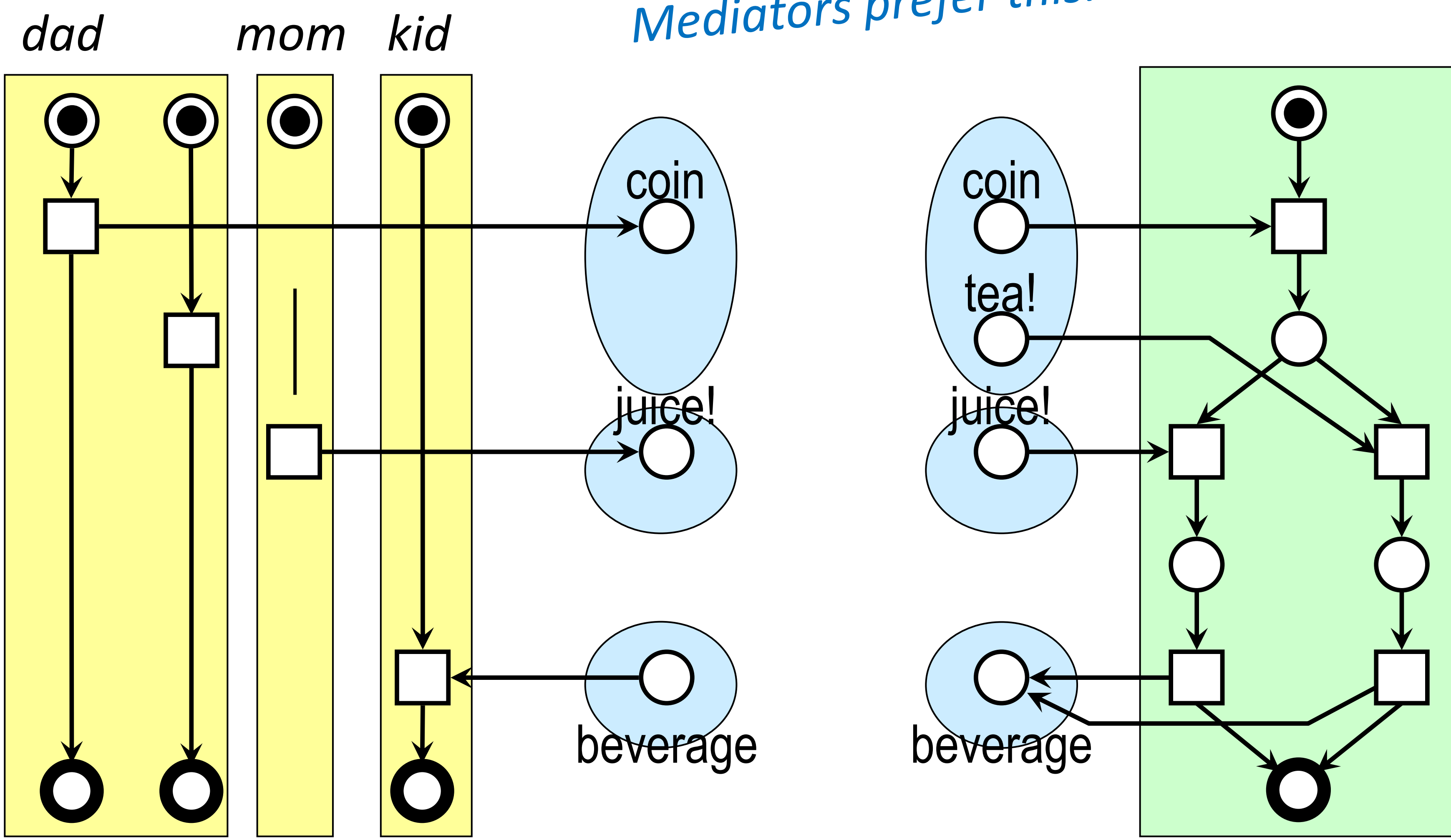
A strange choice of ports

... who orders a beverage ?

Alternative:

Observation: *dad* and *mom* must communicate.

Mediators prefer this.



Observation

Communicating partners
of an open system
can "achieve more"
than detached partners.

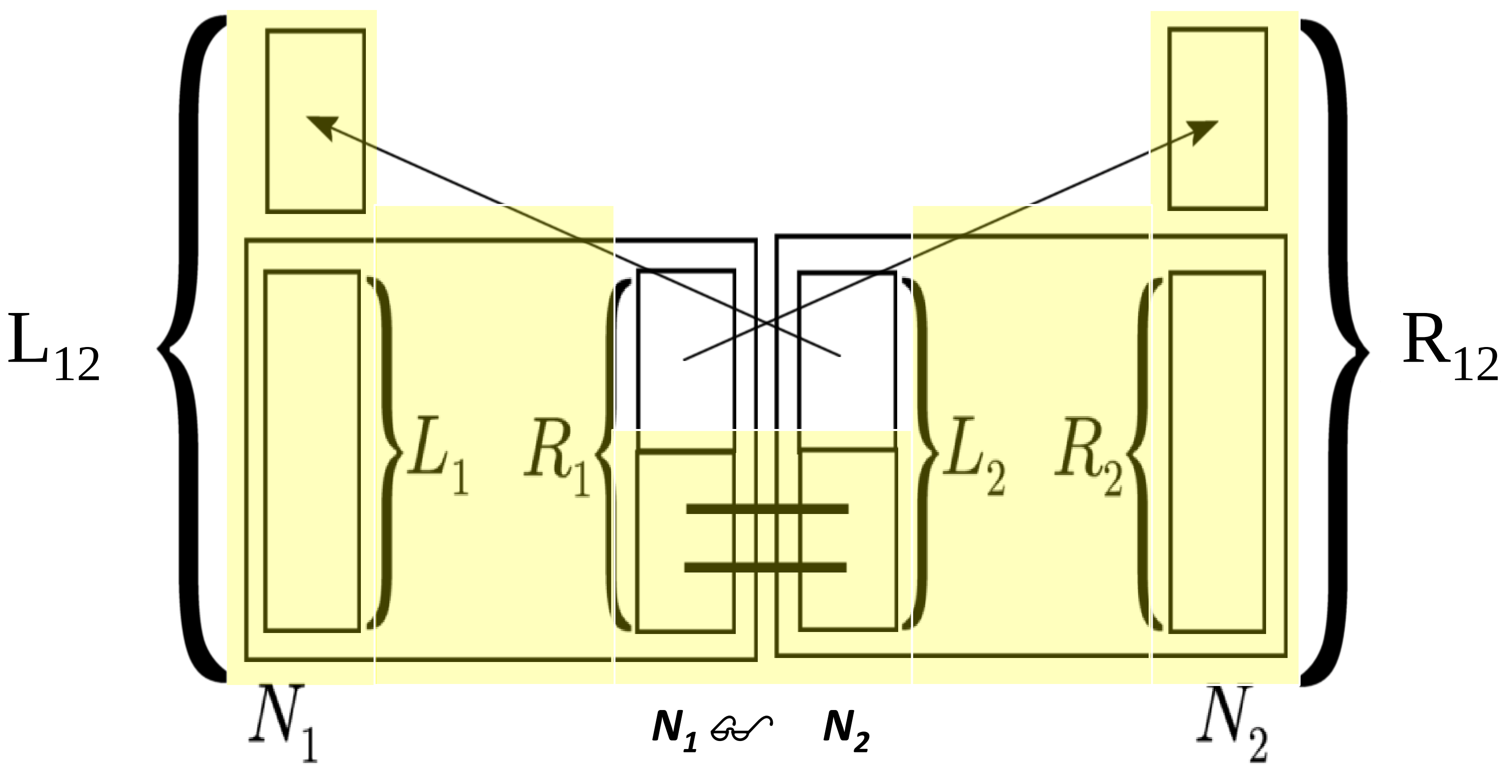
An aspect of composition

Composition is commutative, but not associative.

Sometimes you wish an associative composition.

buyer ⊕ *shop* ⊕ *producer*.

Feasible with a *left* port and a *right* port



Finish with commercials

More on tools for SOC?

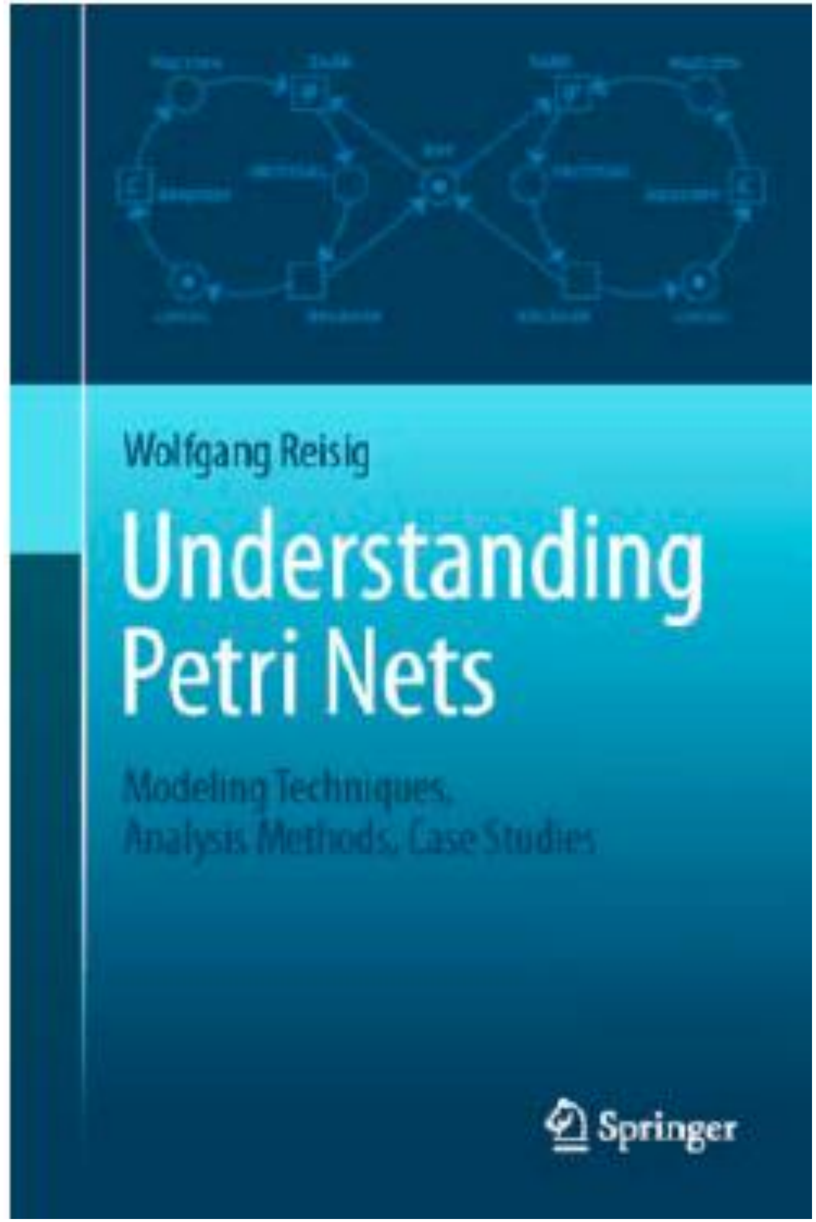
service-technology.org

solutions that make services behave well

Finish with a commercial

More on Petri Nets?

Read *this:*



Moscow HSE Nov 11, 2013

Service Oriented computing: Challenges and ideas to meet them

Wolfgang Reisig

Humboldt Universität Informatik

Institut für Informatik
Humboldt-Universität zu Berlin

Theory of Programming

- *Challenges are many and are not trivial*
- *worth to be attacked*
- *need research into fundamentals of Software Engineering*
- *requires tools*