Next generation modeling in the era of cyber-physical systems

SummerSoc Friday, June 30, 2023

Peter Fettke

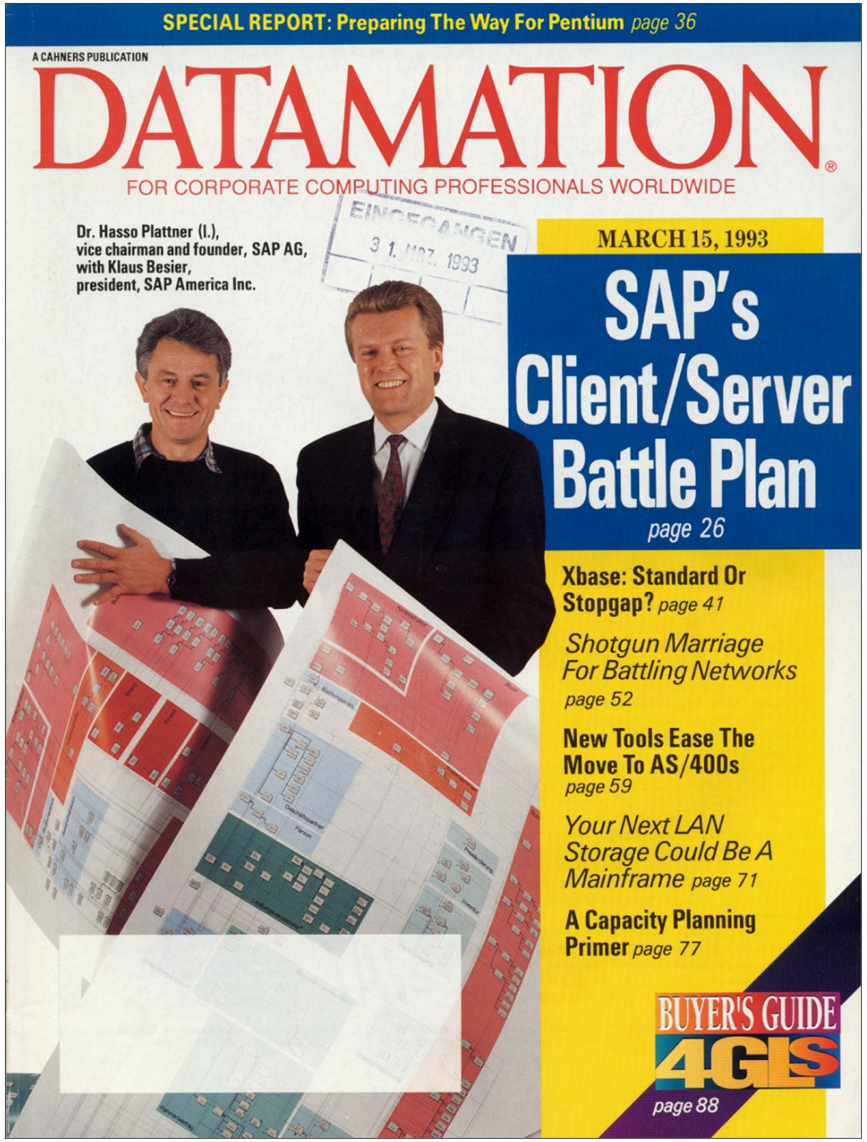
German Research Center for Artificial Intelligence (DFKI) and Saarland University, Saarbrücken, Germany

Wolfgang Reisig

Humboldt-Universität zu Berlin Germany

First generation modeling anno 1993: Hasso Plattner and Klaus Besier pose with the SAP ERP Reference Model

Peter



SPECIAL REPORT: Preparing The Way For Pentium *page 36*

A CAHNERS PUBLICATION

DATAMATION®

FOR CORPORATE COMPUTING PROFESSIONALS WORLDWIDE

EINGEGANGEN 3 1. MRZ. 1993

Dr. Hasso Plattner (l.), vice chairman and founder, SAP AG, with Klaus Besier, president, SAP America Inc.

MARCH 15, 1993

SAP's Client/Server Battle Plan *page 26*

Xbase: Standard Or Stopgap? *page 41*

Shotgun Marriage For Battling Networks *page 52*

New Tools Ease The Move To AS/400s *page 59*

Your Next LAN Storage Could Be A Mainframe *page 71*

A Capacity Planning Primer *page 77*

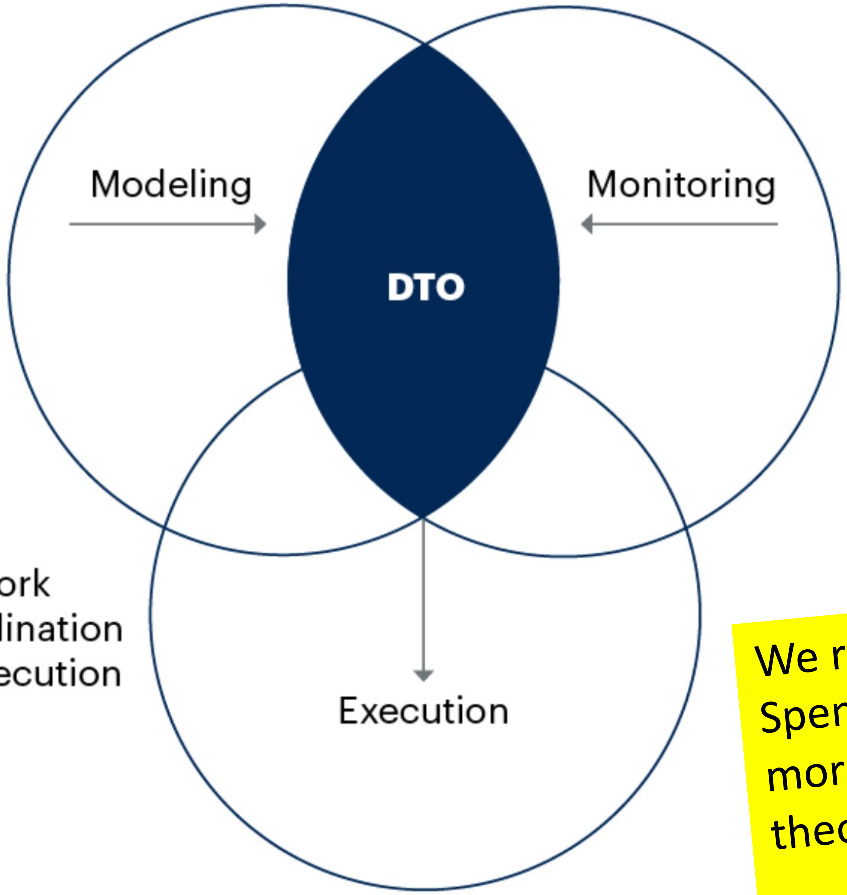
BUYER'S GUIDE 4GLS *page 88*

What is needed today? According to Gartner operations in the digital world (“DigitalOps”) combine three domains

DigitalOps Combines Three Domains

Process Modeling
Planning and Understanding Work
- Process and Decision
- Data and Events
- Goals and Motivations
- Machine Learning

Process Monitoring
Knowing What Happened
- Business Operations Monitoring
- Business Performance Dashboards
- Analytics, BAM and BI
- KPIs and KRIs



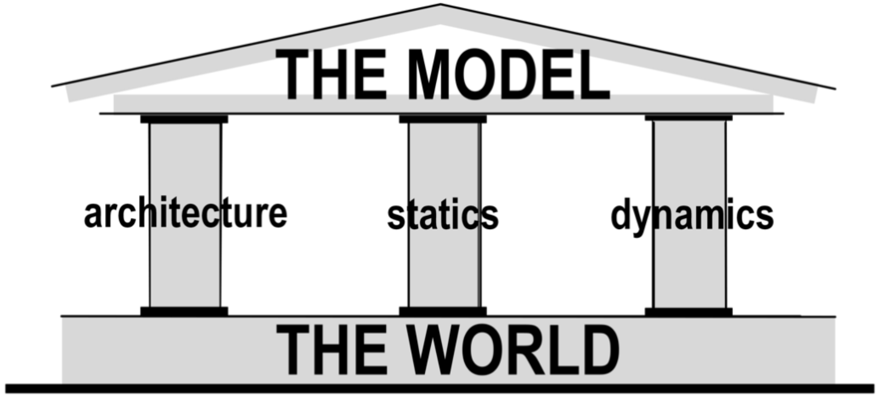
Process/Task Execution
Supporting and Driving Work
- Orchestration and Coordination
- Process and Decision Execution
- Event Management
- Algorithms and Bots
- Integration Interfaces

Source: Gartner (November 2021)

DTO = digital twin of an organization

Note: Organizations are moving to combine these domains, rather than dealing with them separately.

We recommend:
Spend effort in understanding *models* of the digital world: more expressive, more comprehensive, and better theoretically founded!



Part I examples

- 1. architecture
- 2. single behavior
- 3. elementary systems
- 4. items and data

Part II A glimpse at concepts:

The three HERAKLIT pillars

5. architecture: Two-faced modules

6. dynamics: steps: from requirements to models

7. statics: Breathing live into logic: structures and signatures

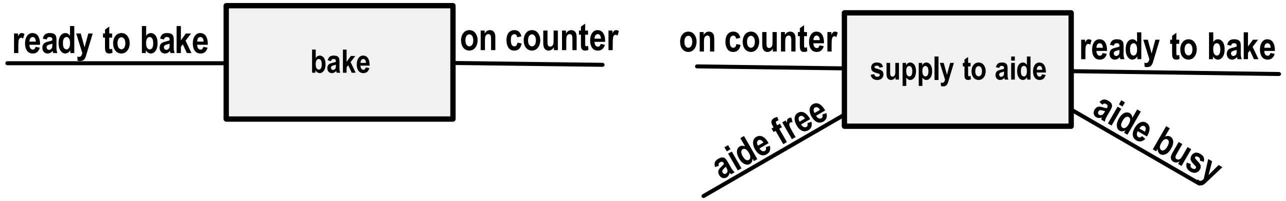
Part III A big case study: an apetizer

Part I Examples
1. Architecture

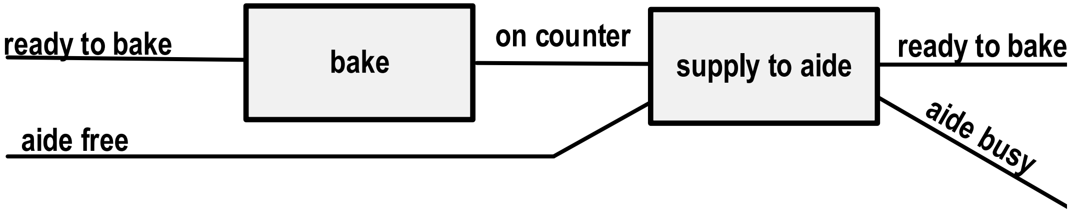
The four activities of a bakery

Wolfgang

- A big system consists of *modules*.
- A module has an *interface*.
- An interface consists of *gates*.
- Each gate is *labeled*.



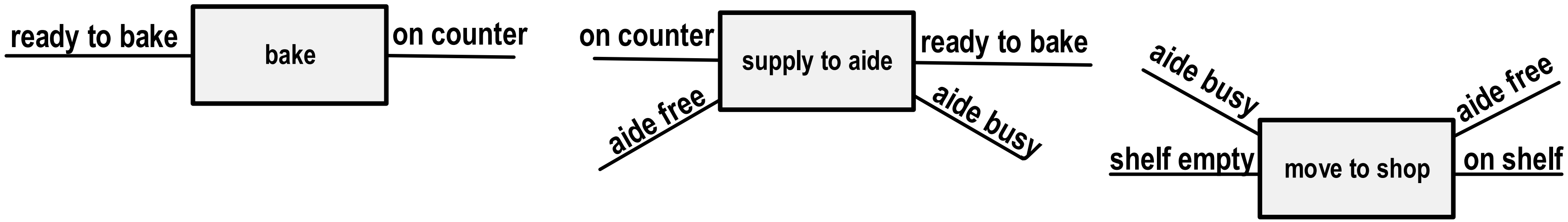
To *compose* two modules, merge equally labeled gates:



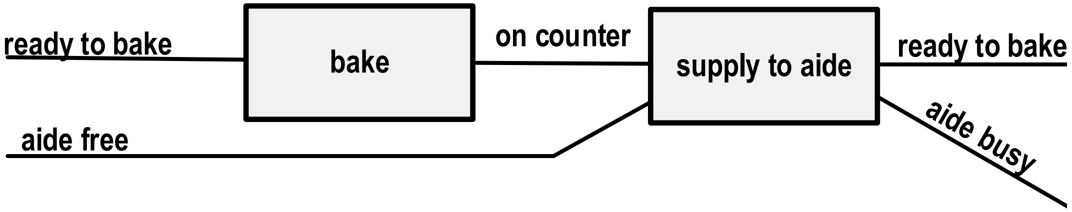
Part I Examples
1. Architecture

The four activities
of a bakery

bake • supply to aide



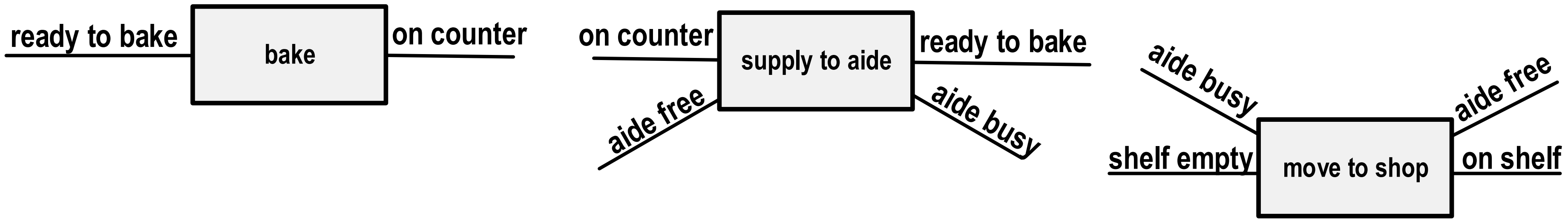
To *compose* two modules, merge equally labeled gates:



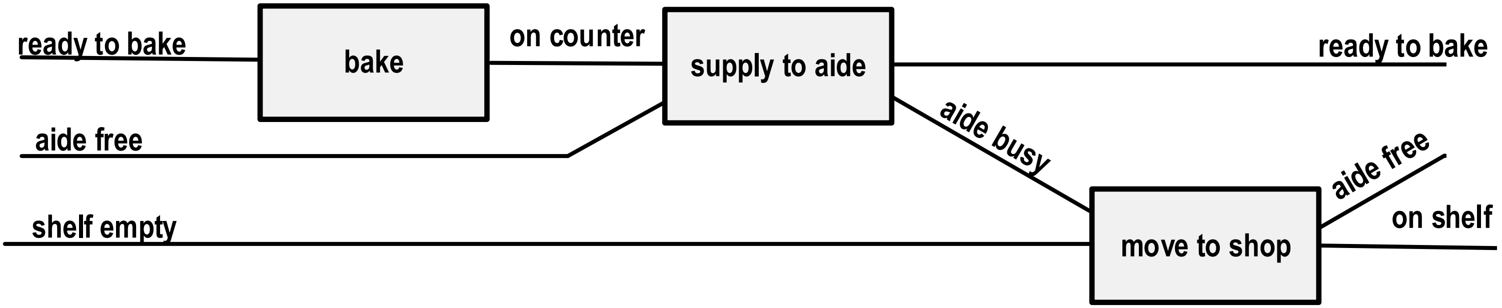
Part I Examples
1. Architecture

The four activities of a bakery

bake • supply to aide • move to shop



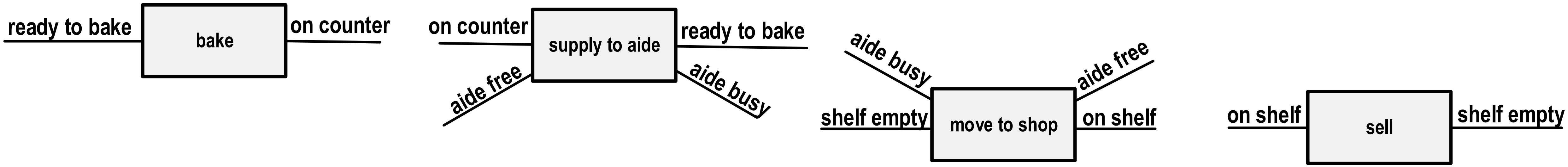
To *compose* two modules, merge equally labeled gates:



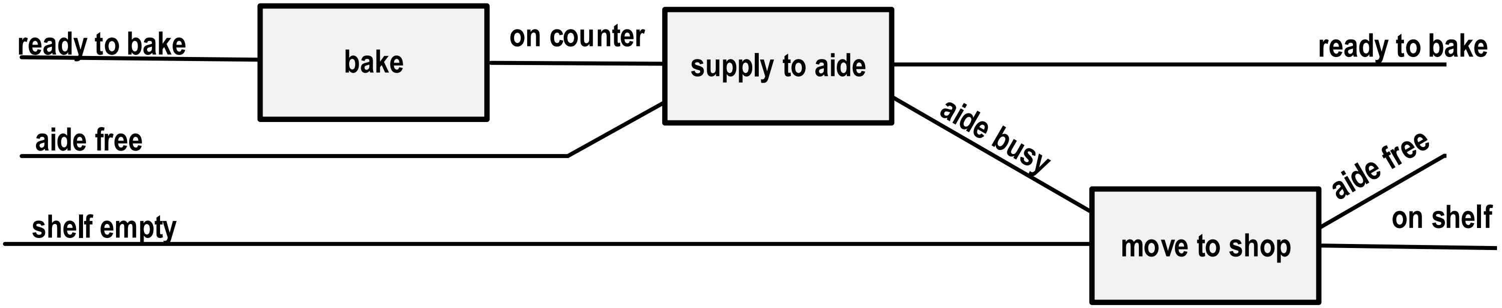
Part I Examples
1. Architecture

The four activities
of a bakery

bake • supply to aide • move to shop



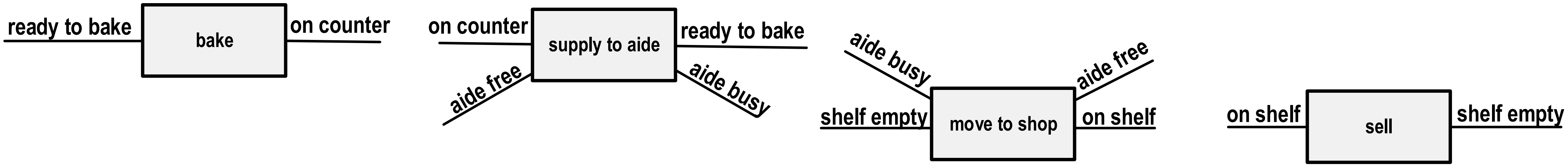
To *compose* two modules, merge equally labeled gates:



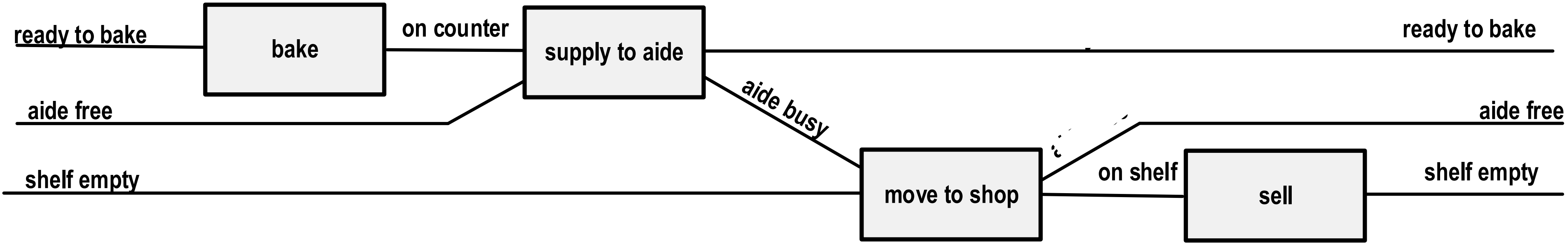
Part I Examples
1. Architecture

The four activities
of a bakery

bake • supply to aide • move to shop • sell



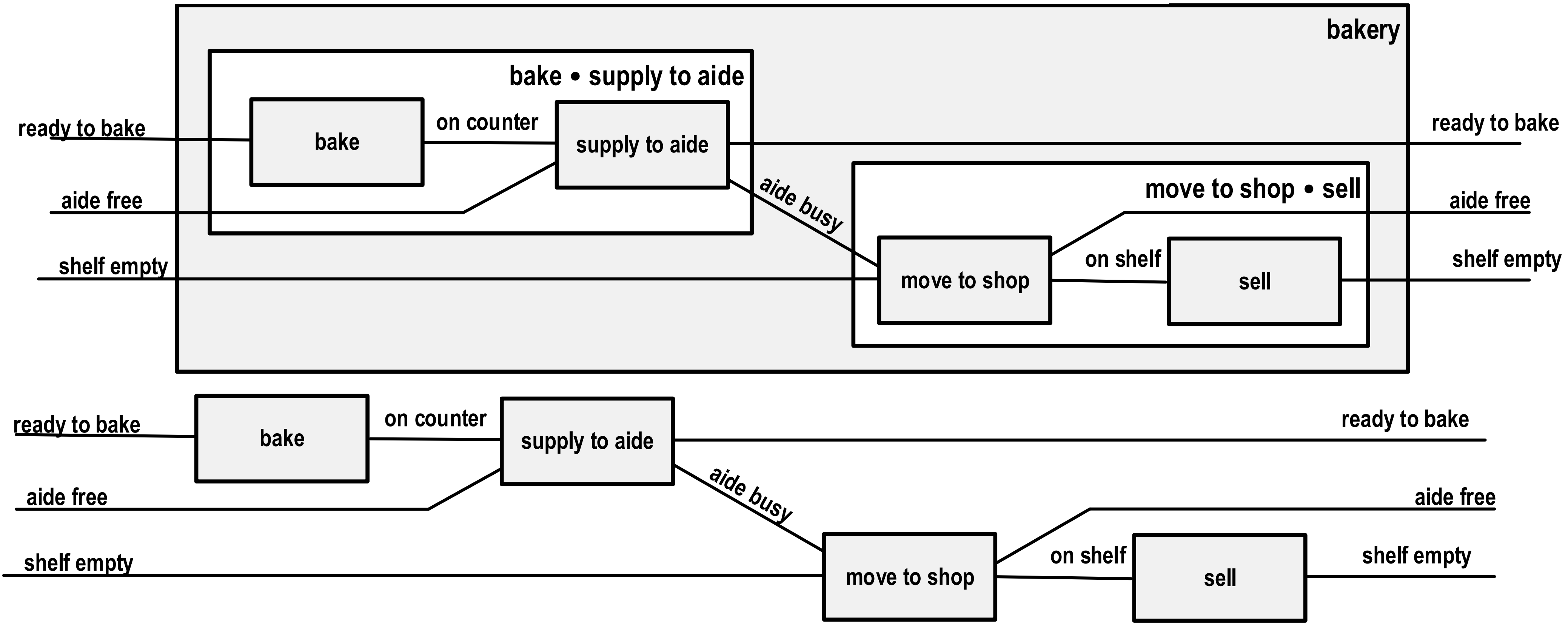
To *compose* two modules, merge equally labeled gates:



Part I Examples
1. Architecture

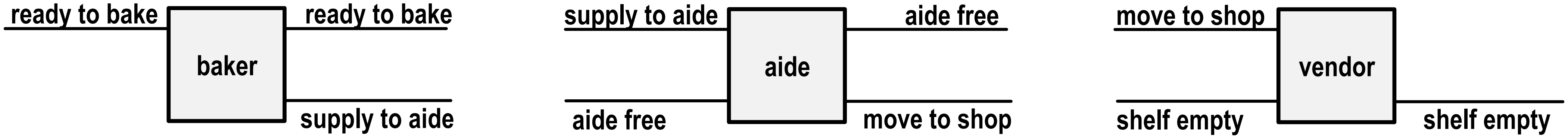
The four activities of a bakery

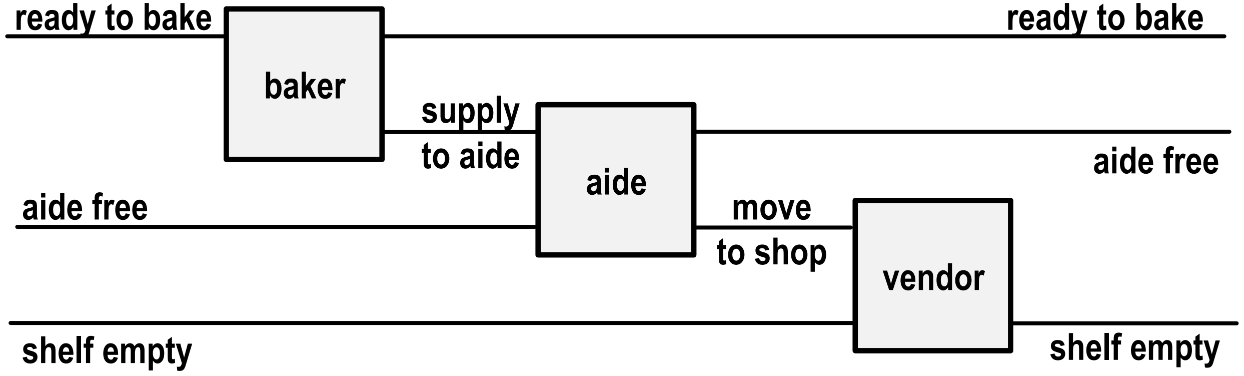
bake • supply to aide • move to shop • sell



Part I Examples
1. Architecture

The three staff
of a bakery

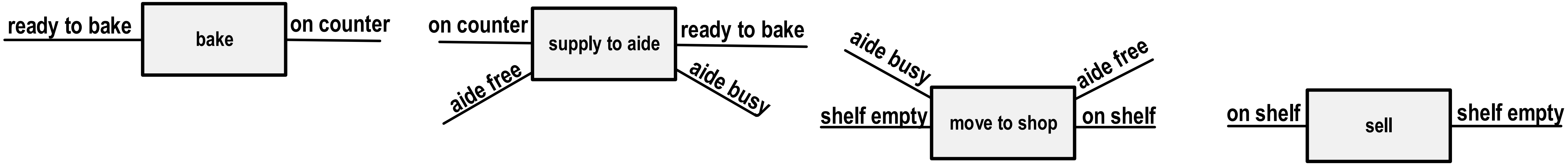




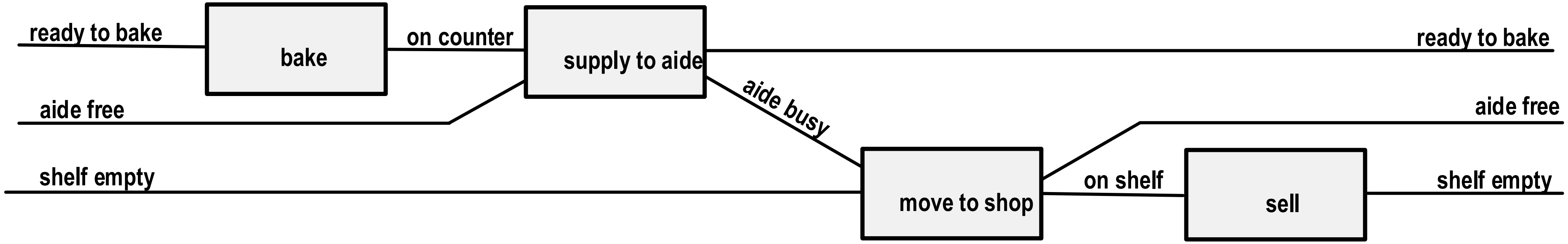
Later:
the four activities
and the three staff
abstract the same behavior

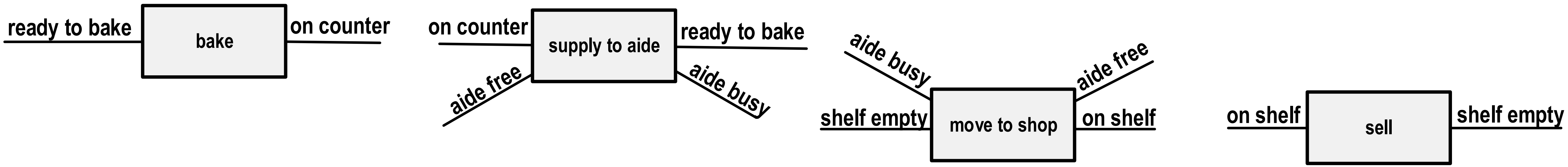
Part I Examples
2 behavior

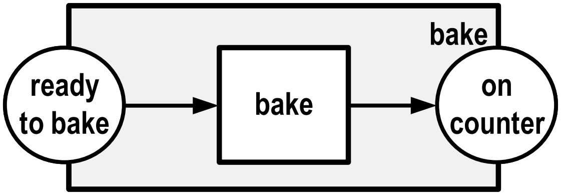
Peter



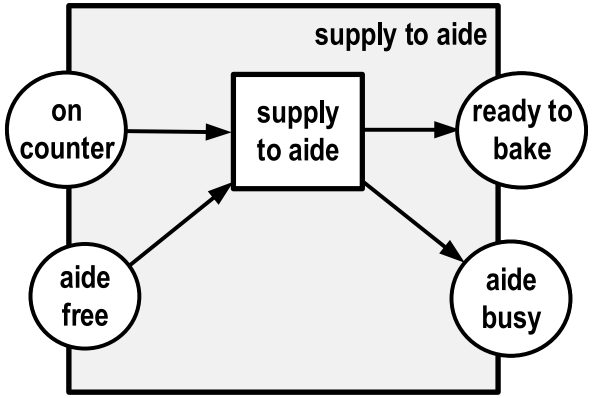
Remember the modules

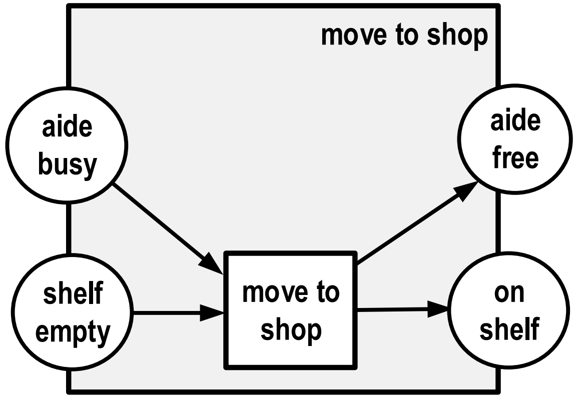


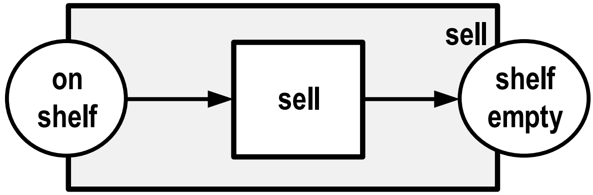




Four steps describe behavior



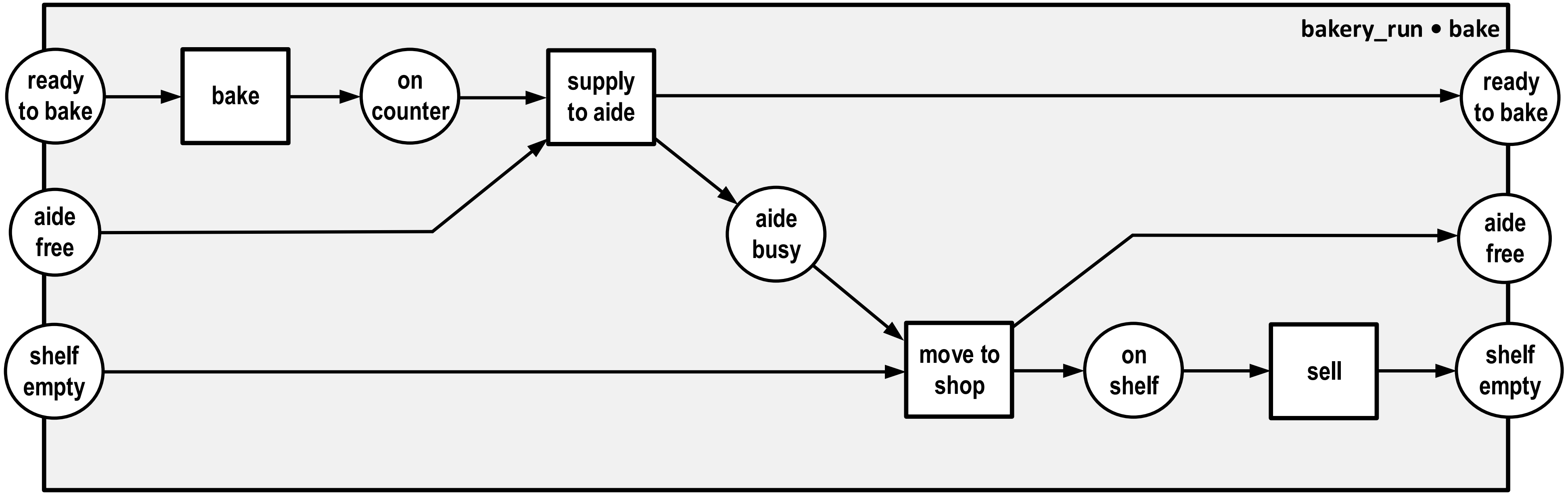


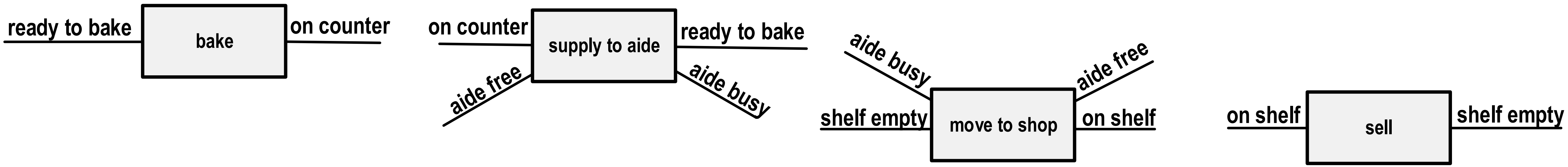


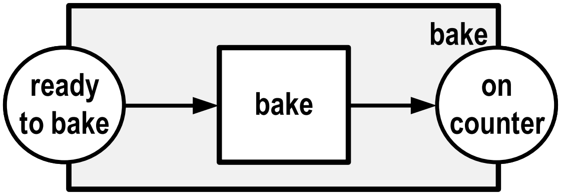
Composed steps

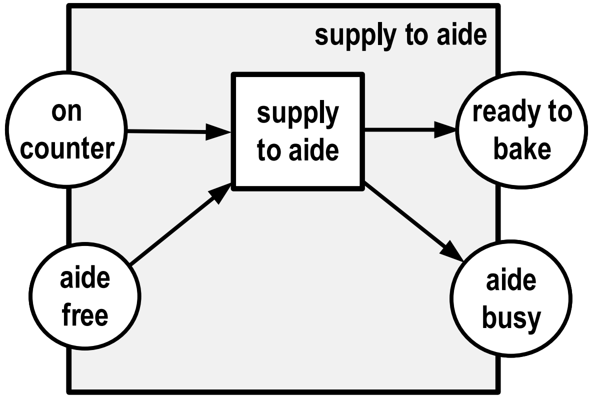
bakery-run:

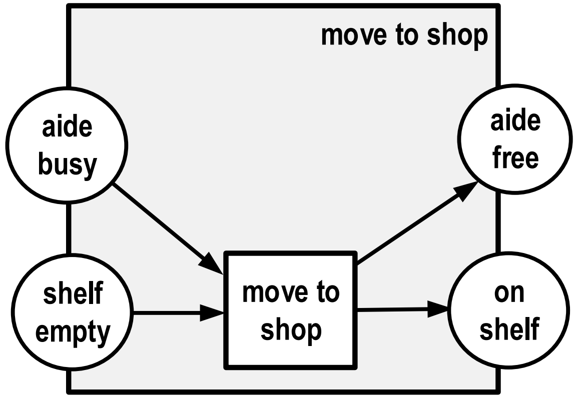
bake • supply to aide • move to shop • sell

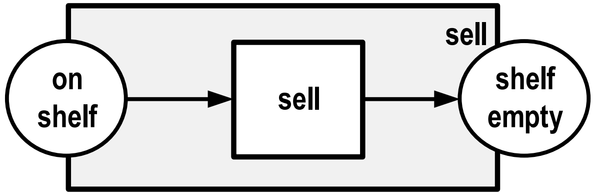










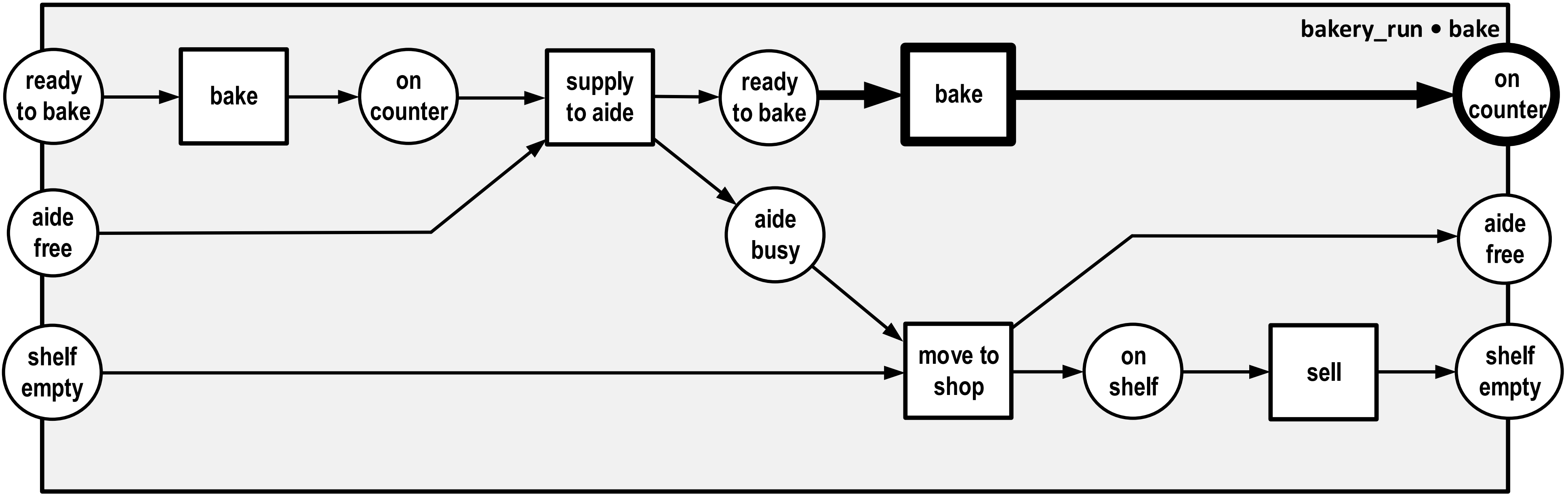


Four steps describe behavior

Composed steps

bakery-run:

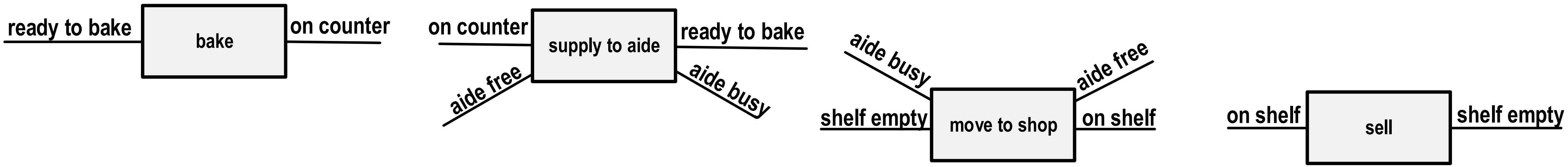
bake • supply to aide • move to shop • sell

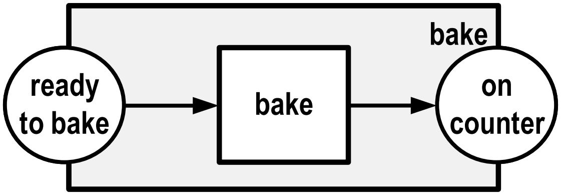


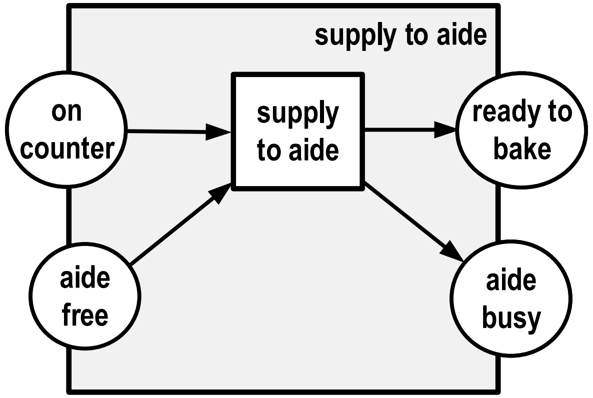
extension

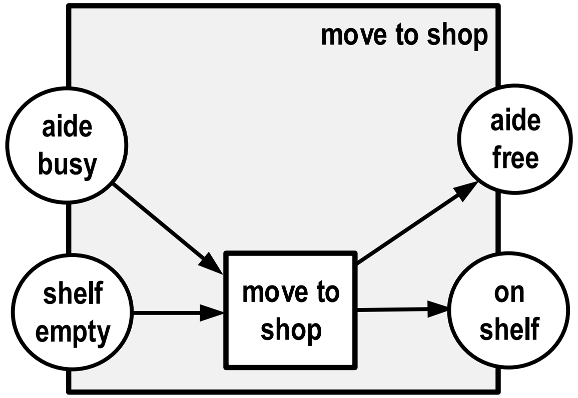
bakery-run • bake

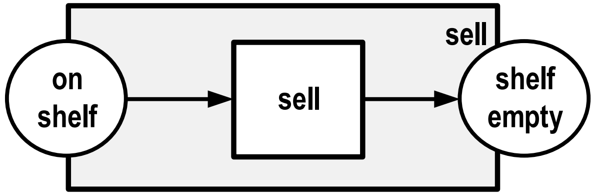
No more totally ordered!







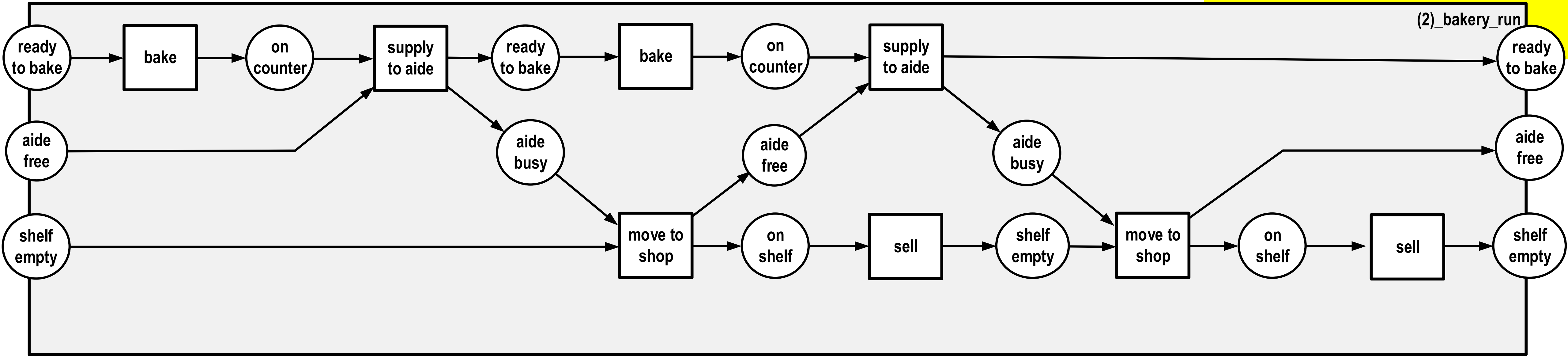




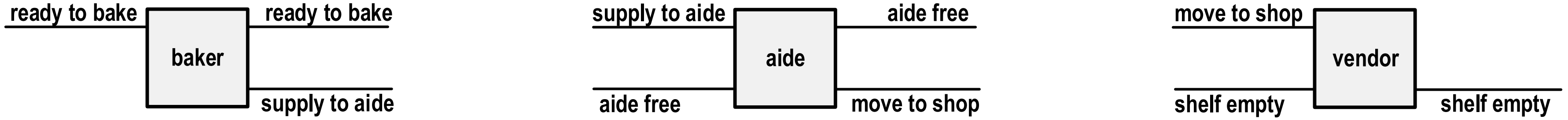
Four steps
describe behavior

Composed steps

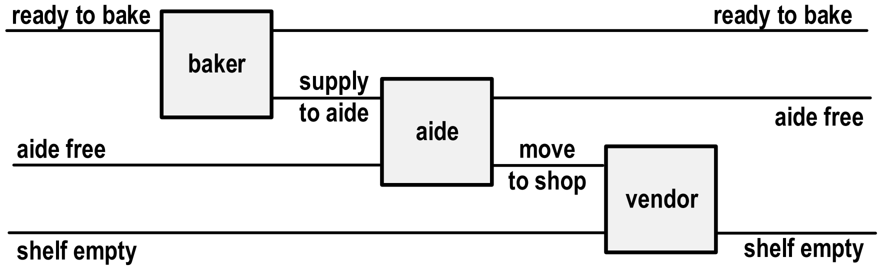
bakery-run • bakery-run



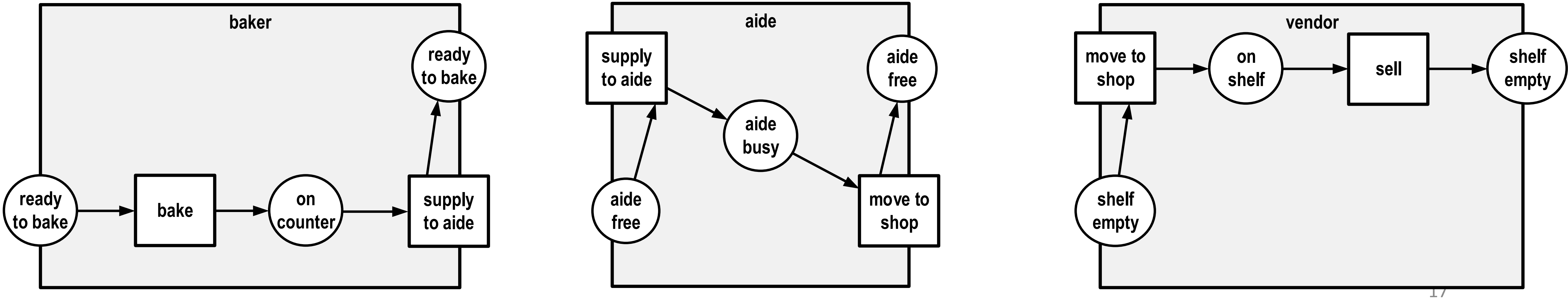
Example: modules of staff



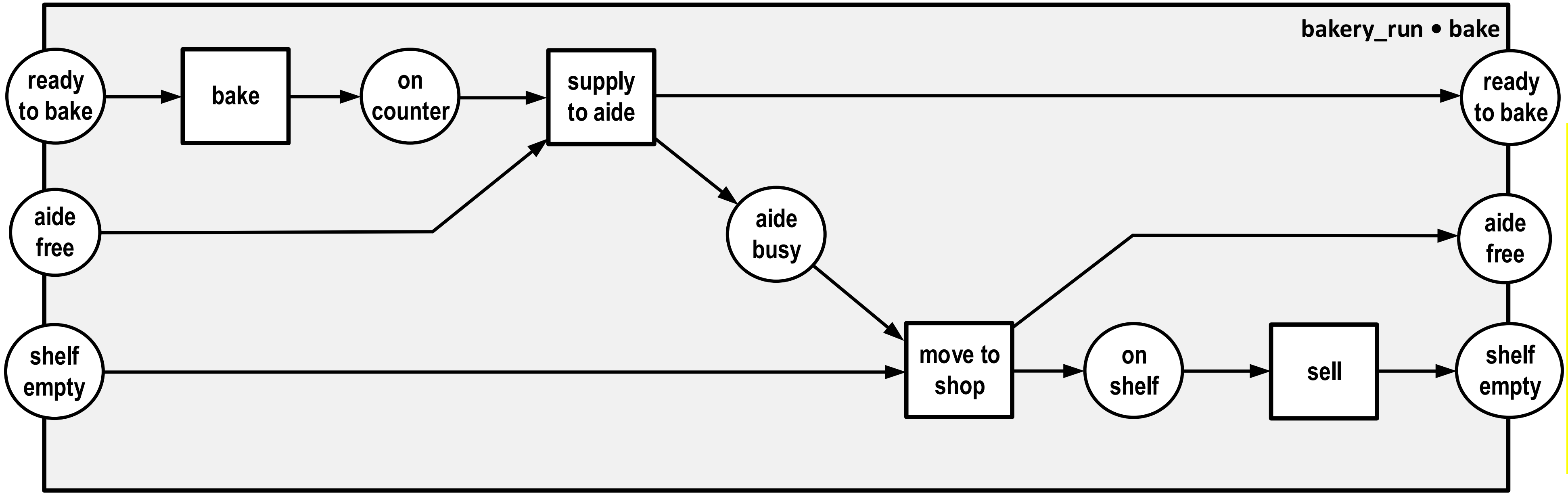
Remember the modules



Operational behavior:



Example: modules of staff

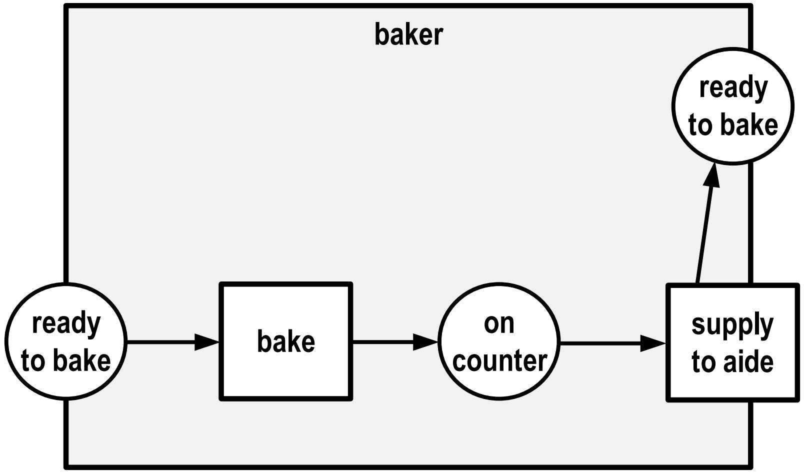


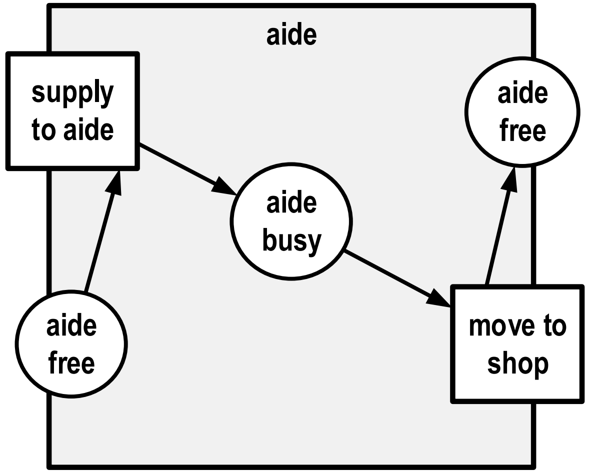
Remember

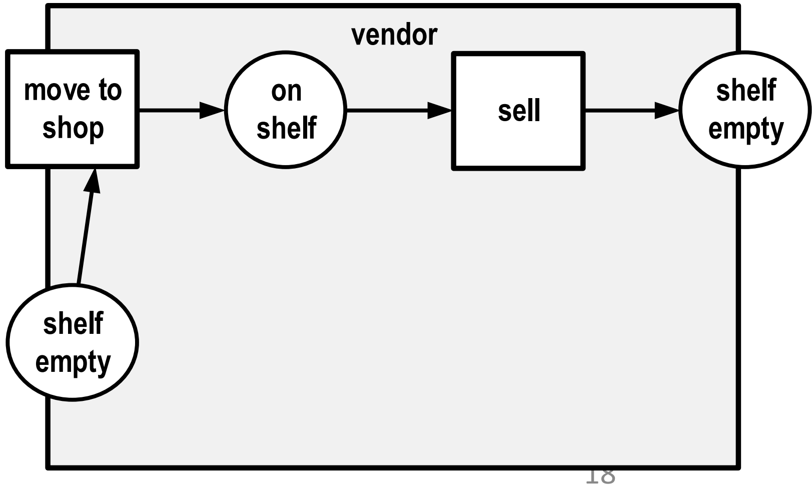
bake • supply to aide • move to shop • sell

=

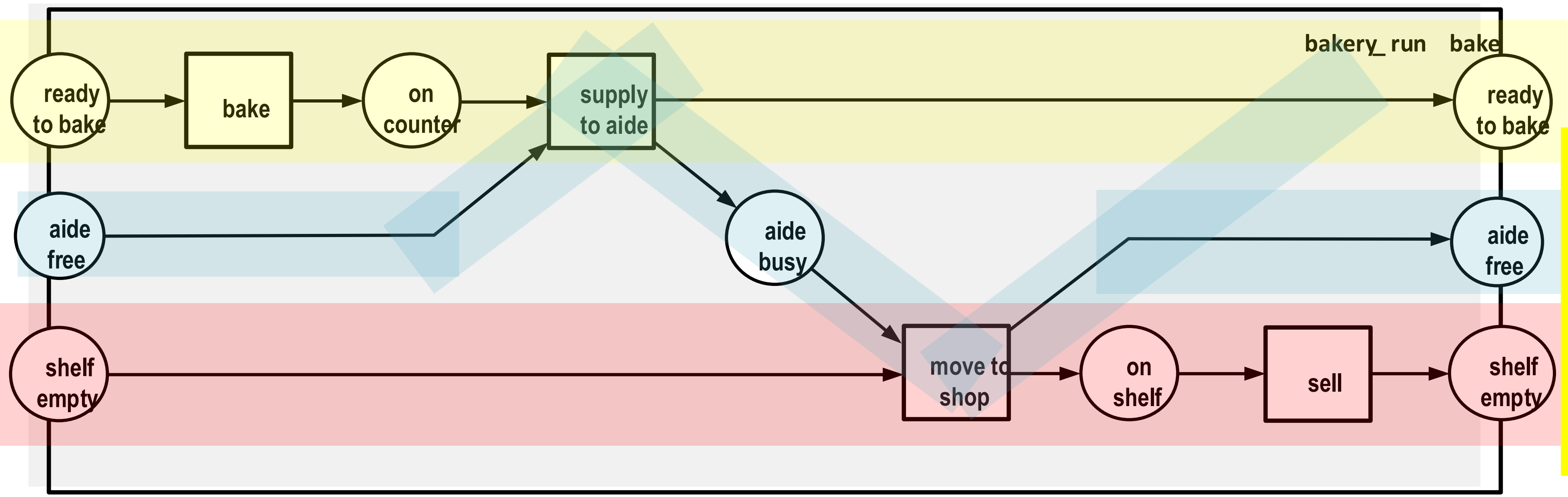
baker • aide • vendor







Example: modules of staff

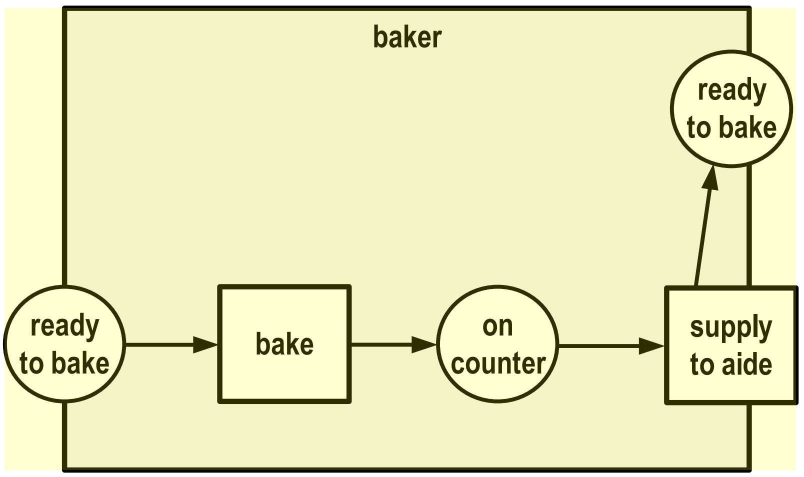


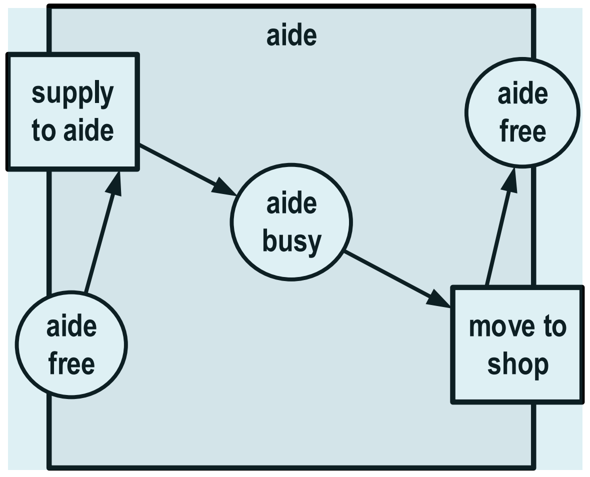
Remember

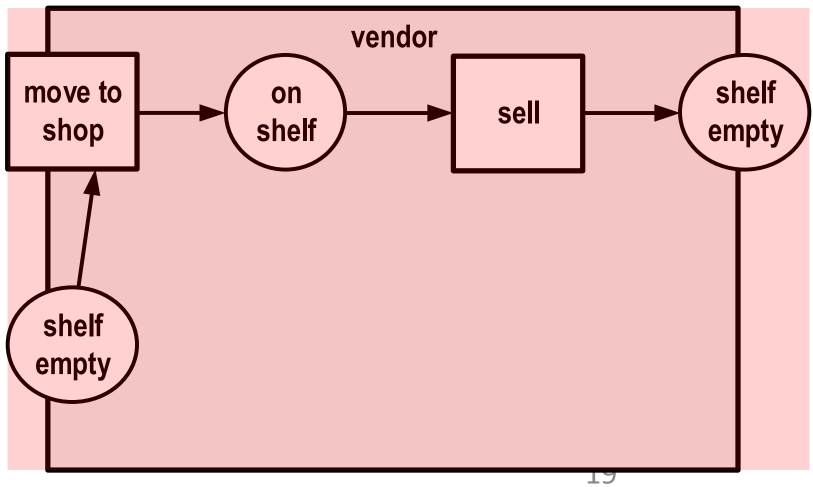
bake • supply to aide • move to shop • sell

=

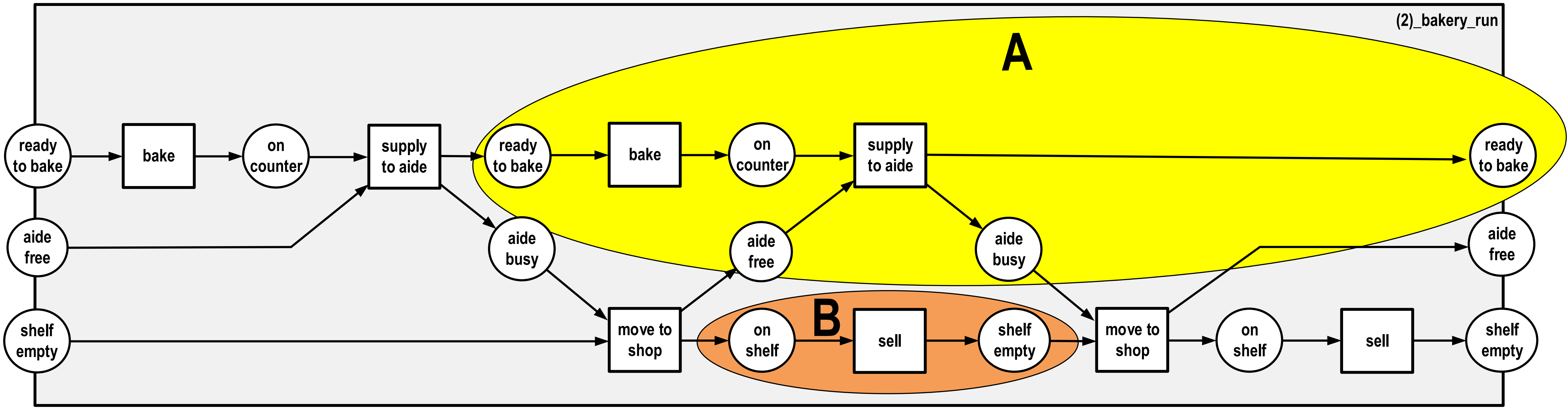
baker • aide • vendor







Concurrency is not transitive!

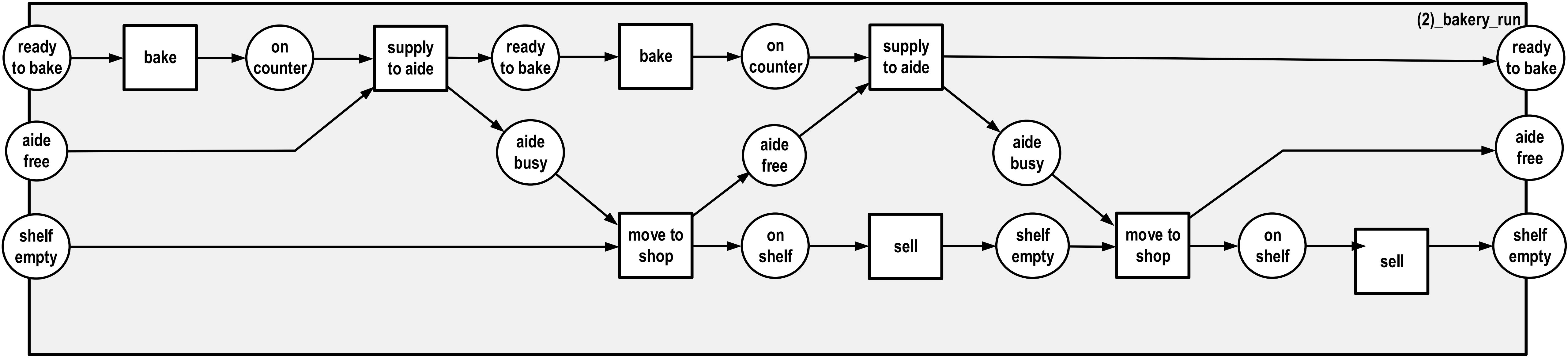


each node in A is concurrent to each node in B

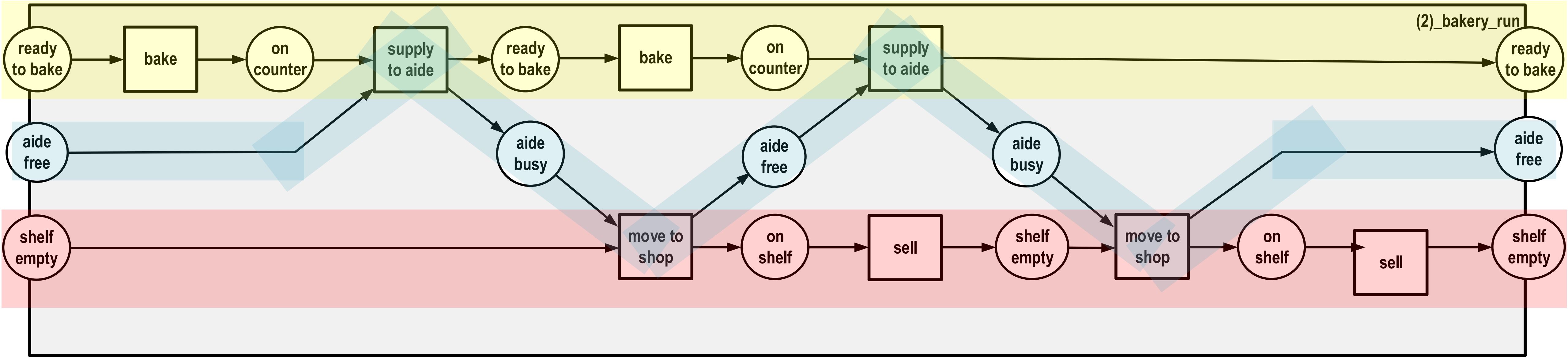
Part I Examples
3. Elementary systems

Wolfgang

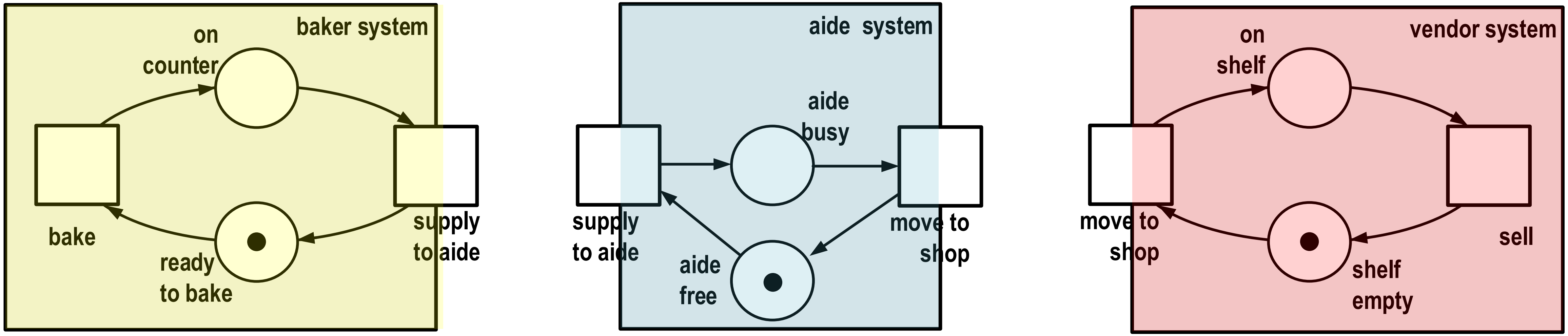
Part I Examples
3. Elementary systems

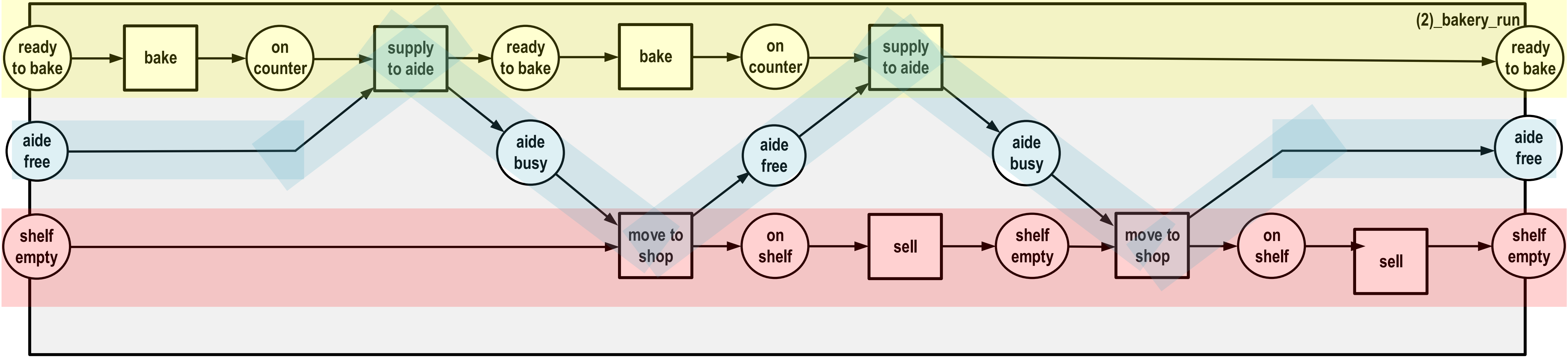


Part I Examples
3. Elementary systems

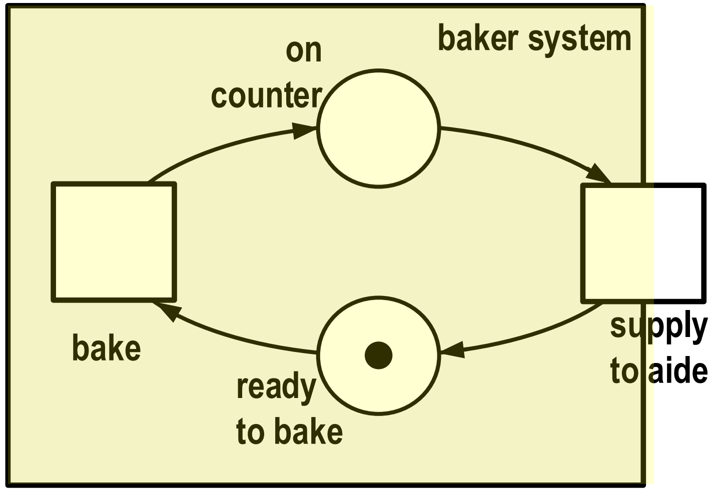


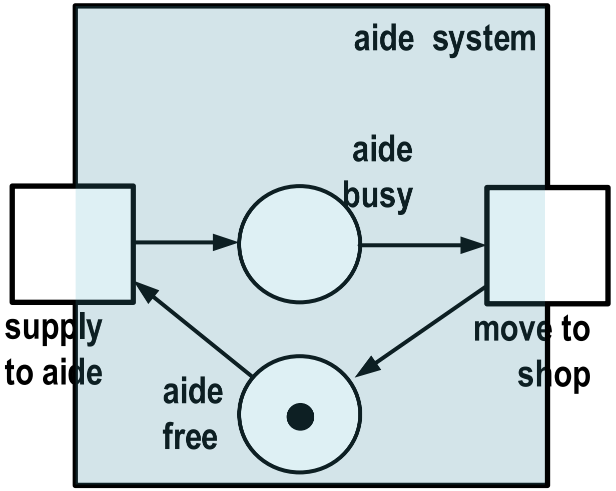
Part I Examples
3. Elementary systems

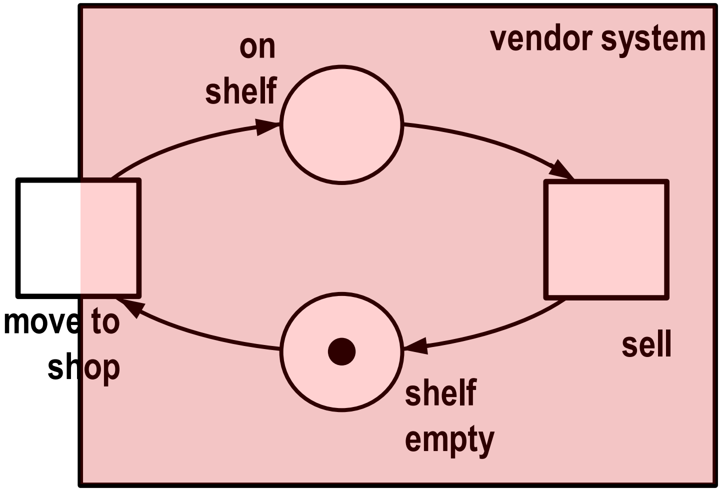


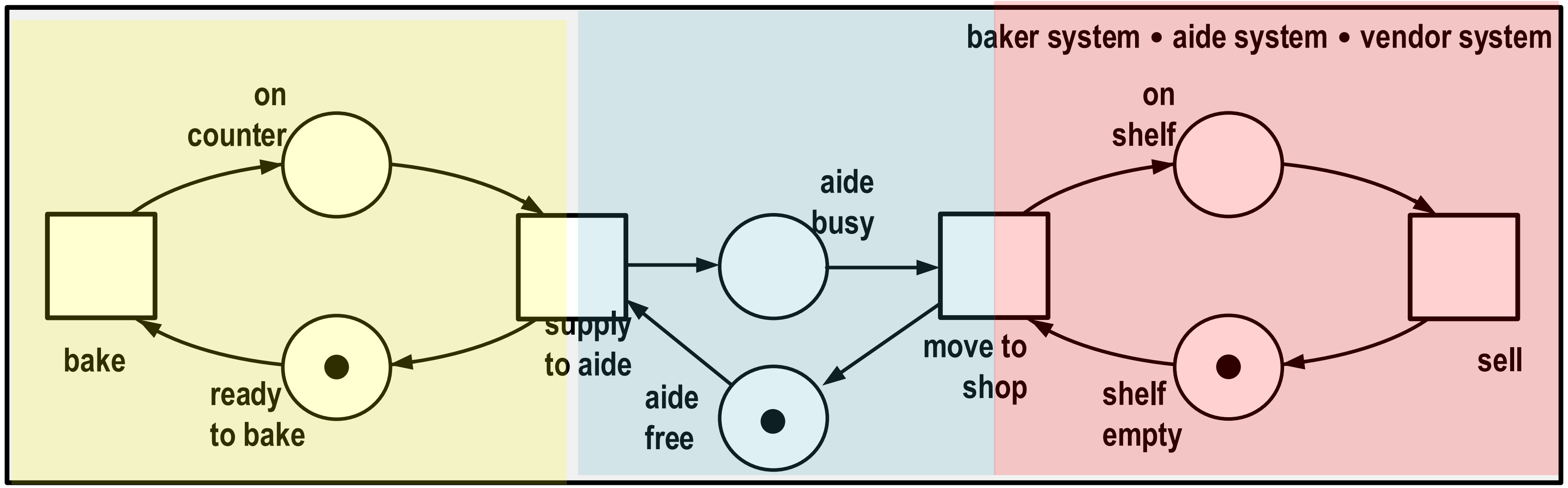


Part I Examples
3. Elementary systems

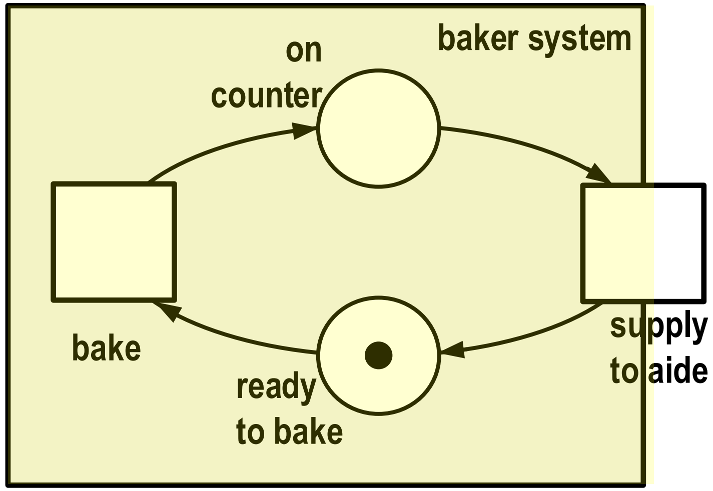


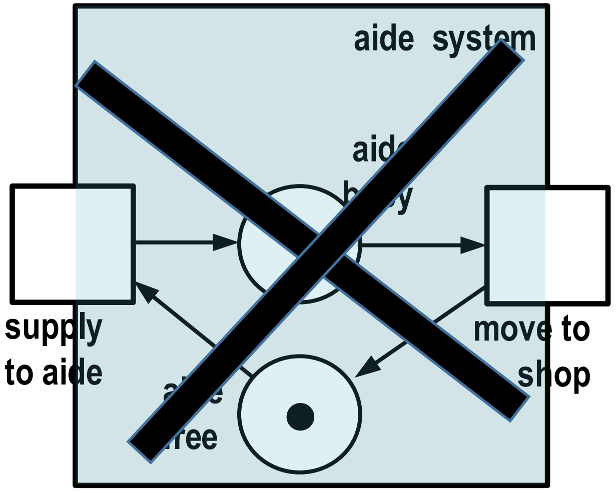


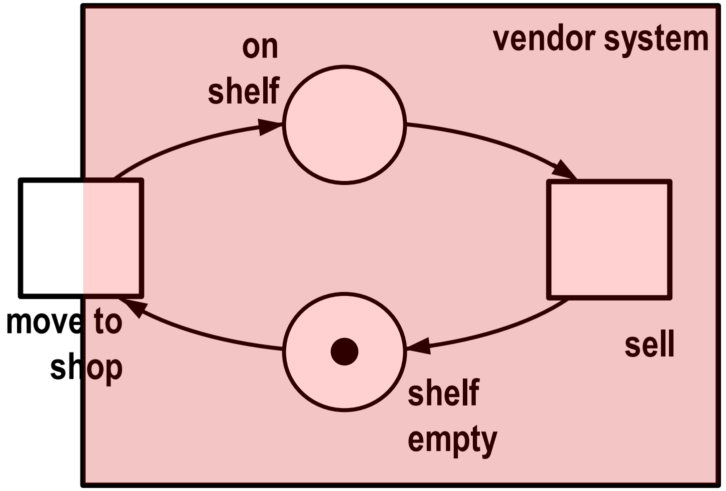




Part I Examples
3. Elementary systems



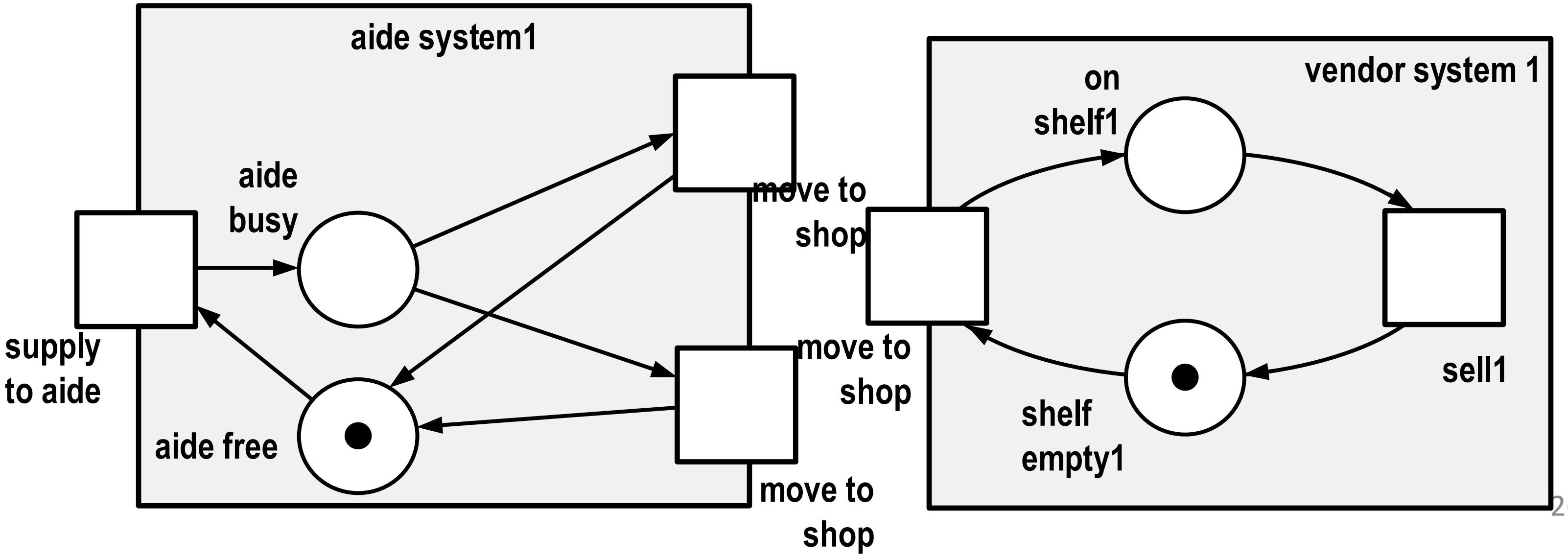




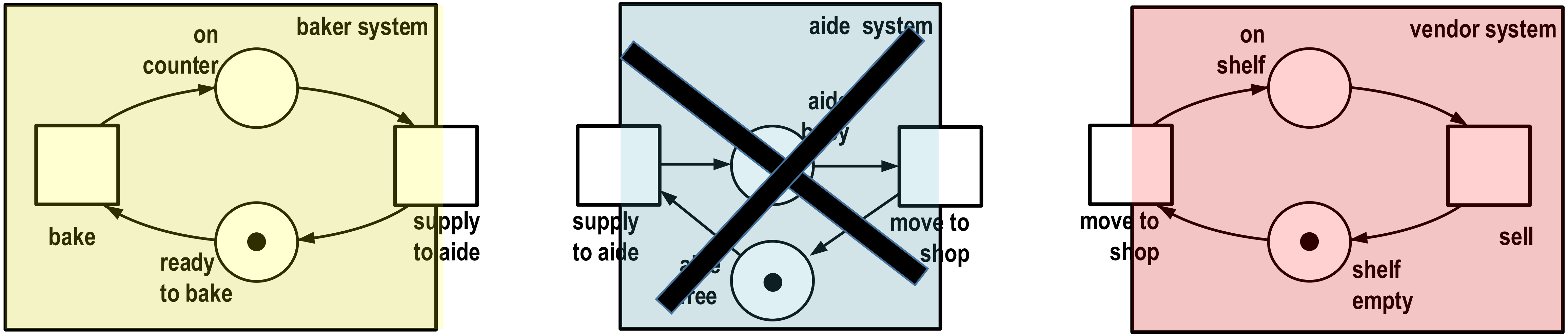
baker system • aide system 1 •
vendor system • vendor system 1

versus

baker system • aide system 1 •
vendor system 1 • vendor system



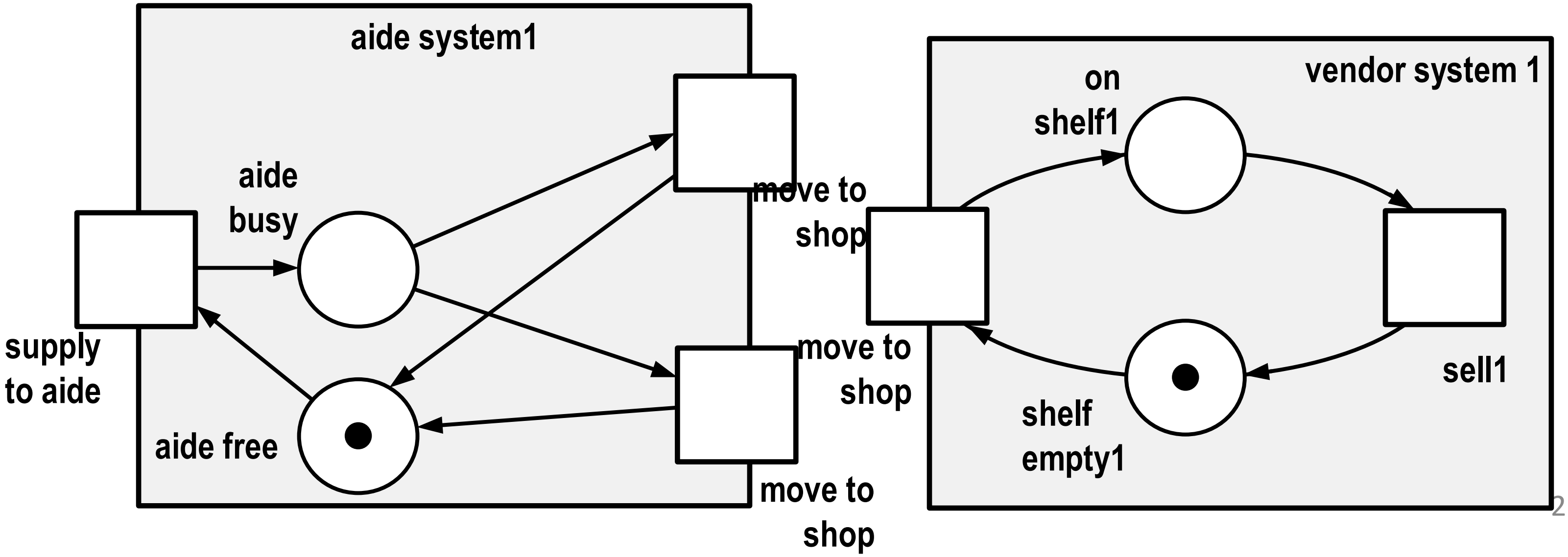
Part I Examples
3. Elementary systems



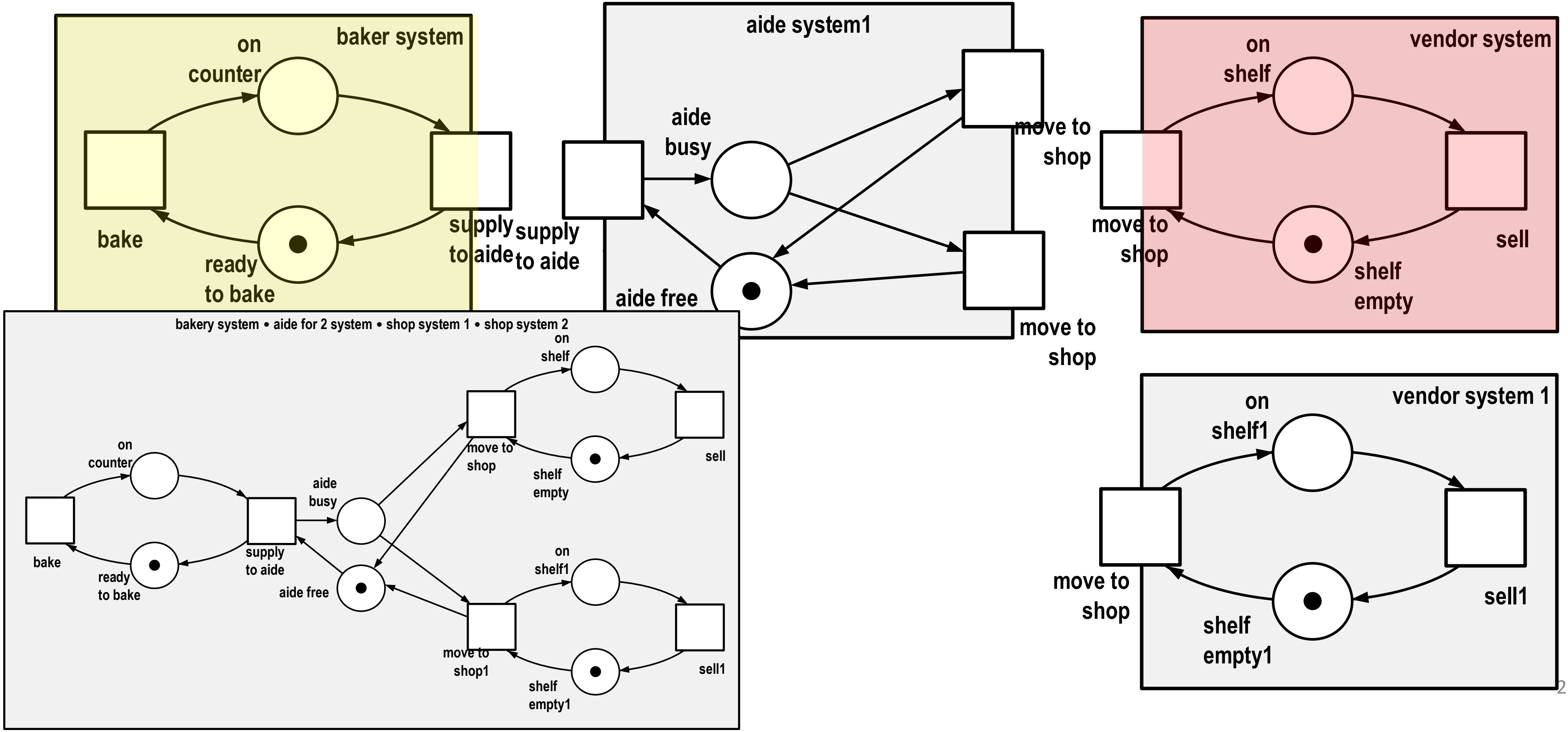
baker system • aide system 1 •
vendor system • vendor system 1

versus

baker system • aide system 1 •
vendor system 1 • vendor system

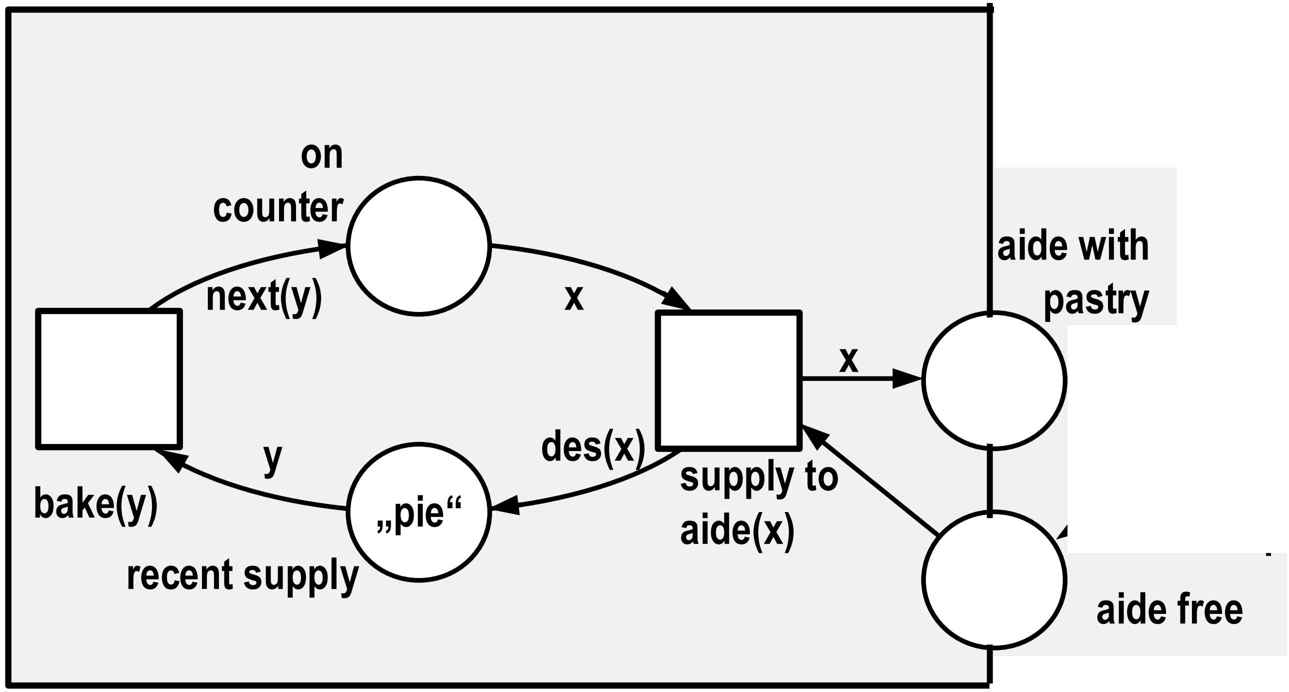


Part I Examples
3. Elementary systems



Peter

4. items and data



bakery with three pastries

domains
pastries = {bread, cake, pie}
descriptions = {„bread“, „cake“, „pie“}

function
next: descriptions ⟶ pastries
next("bread") = cake
next("cake") = pie
next("pie") = bread

function
des: pastries ⟶ descriptions
des(bread) = "bread"
des(cake) = "cake"
des(pie) = "pie"

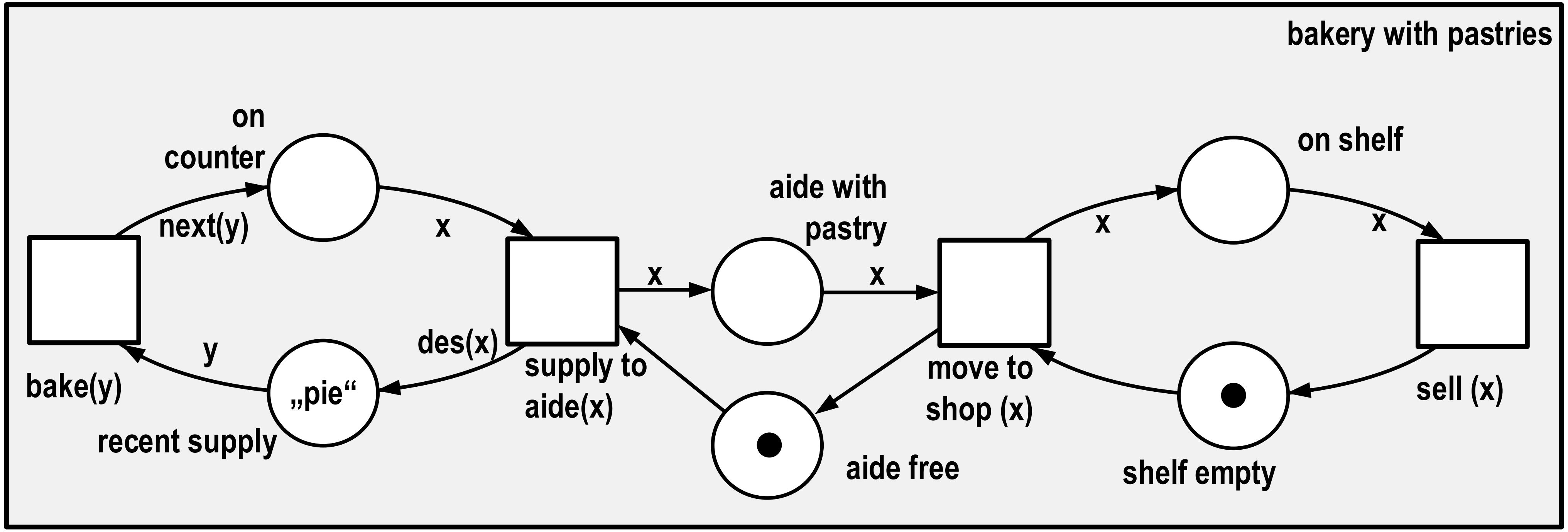
constants
„pie“: descriptions

predicates
on counter, aide with pastry, on shelf: pastries
recent supply: descriptions,

propositions
aide free, shelf empty

variables
x: pastries
y: descriptions

4. items and data



bakery with three pastries

domains
pastries = {bread, cake, pie}
descriptions = {„bread“, „cake“, „pie“}

function
next: descriptions ⟶ pastries
next("bread") = cake
next("cake") = pie
next("pie") = bread

function
des: pastries ⟶ descriptions
des(bread) = "bread"
des(cake) = "cake"
des(pie) = "pie"

constants
„pie“: descriptions

predicates
on counter, aide with pastry, on shelf: pastries
recent supply: descriptions,

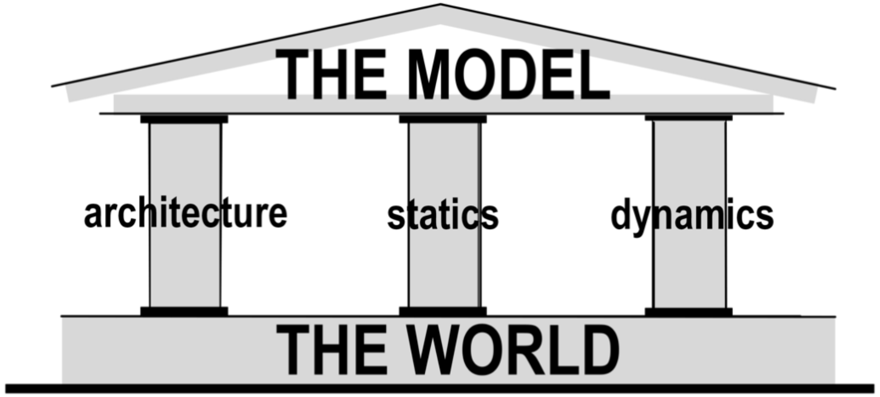
propositions
aide free, shelf empty

variables
x: pastries
y: descriptions

Pause

Part II a glimpse at concepts

Wolfgang



Part II A glimpse at concepts:

The three HERAKLIT pillars

5. architecture: Two-faced modules

6. dynamics: steps: from requirements to models

7. statics: Breathing live into logic: structures, signatures and schamata

Part III A big case study: an apetizer

5. Architecture: two faced modules

Theoretical informatics

Given an alphabet Λ {α, β, γ}.

Canonical constructs:

- word over Λ βγ βαα
- Set of all words over Λ, written Λ*
- Composition of words: αββ • βαγ • βγγ
  = αβββαγβγγ

Monoid (Λ*, •, ε):

THE formal fundament of computing.

HERAKLIT

Given an alphabet Λ {α, β, γ}.

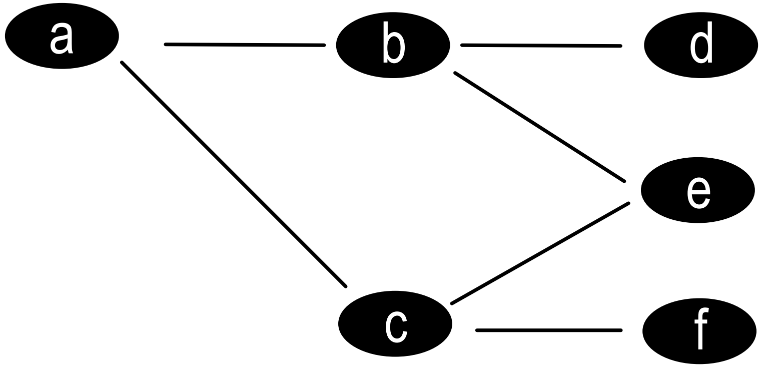
Canonical constructs:

- Modules with gate labels in Λ
- Set of all modules over Λ, written ΛM
- Composition of modules M • N

Monoid (ΛM, •, ε):

THE formal fundament of modeling.

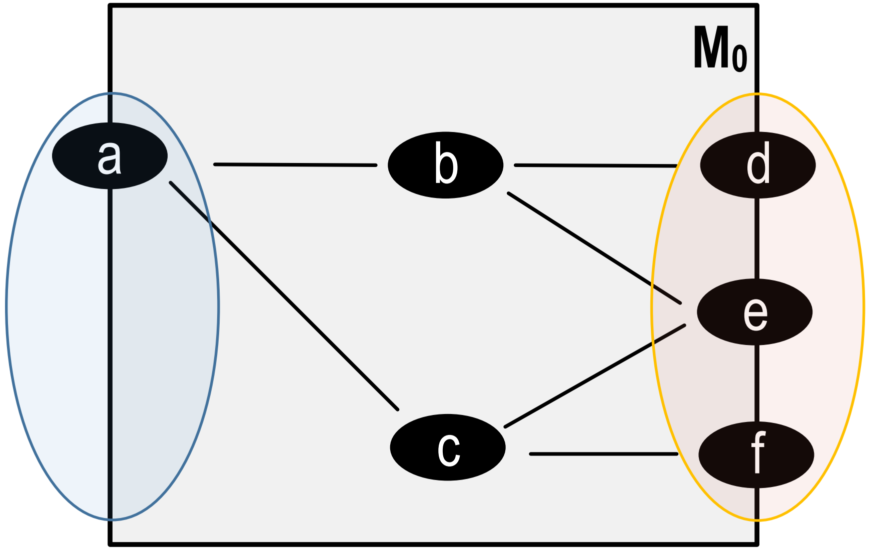
5. Architecture: two faced modules



A module is

- a *graph*

5. Architecture: two faced modules

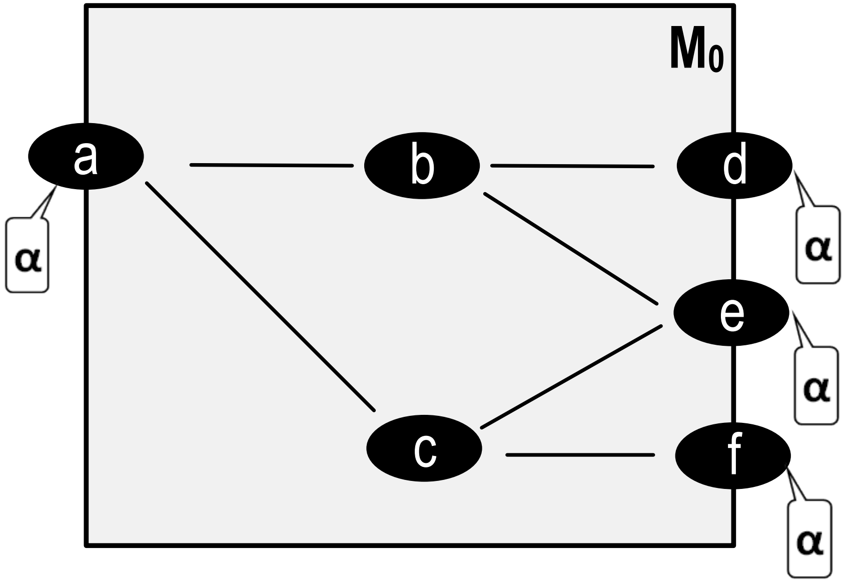


A module is

- a *graph*
- With two distinguished sets of nodes

(left and right *interface*) *„gates“*

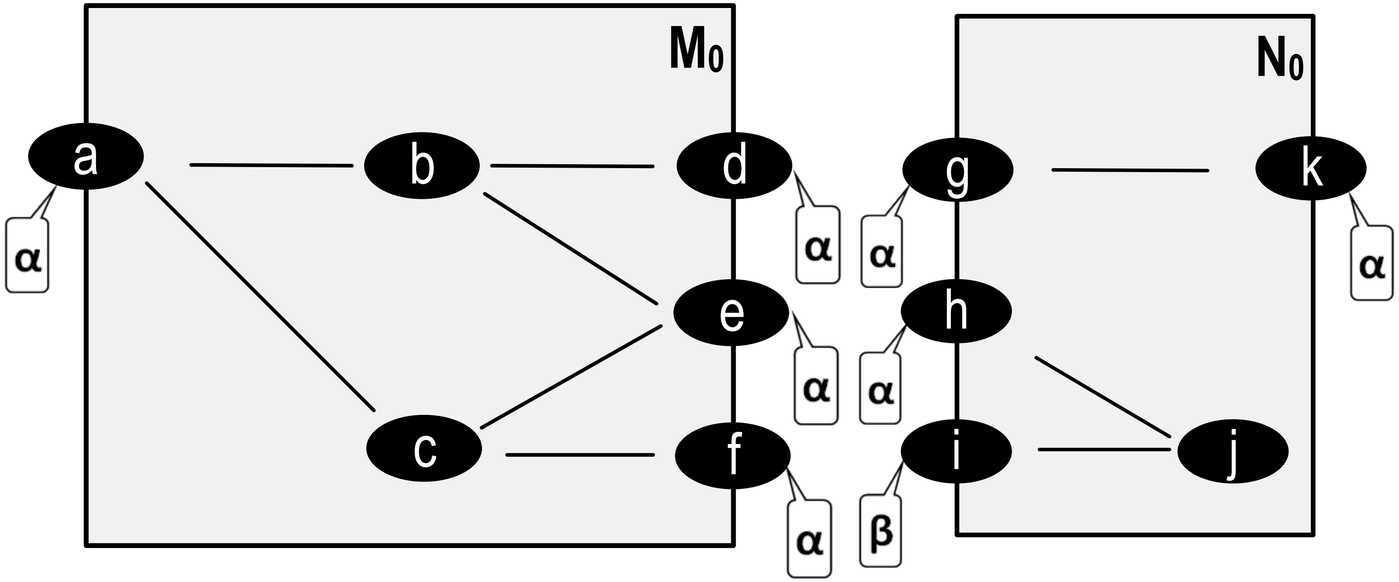
5. Architecture: two faced modules



A module is

- a *graph,*
- With two distinguished sets of nodes (left and right *interface*) ”*gates”*.
- Each gate is labeled.

5. Architecture: two faced modules



A module is

- a *graph,*
- With two distinguished sets of nodes (left and right *interface*) "*gates*".
- Each gate is labeled.

Here a second module, N_0.

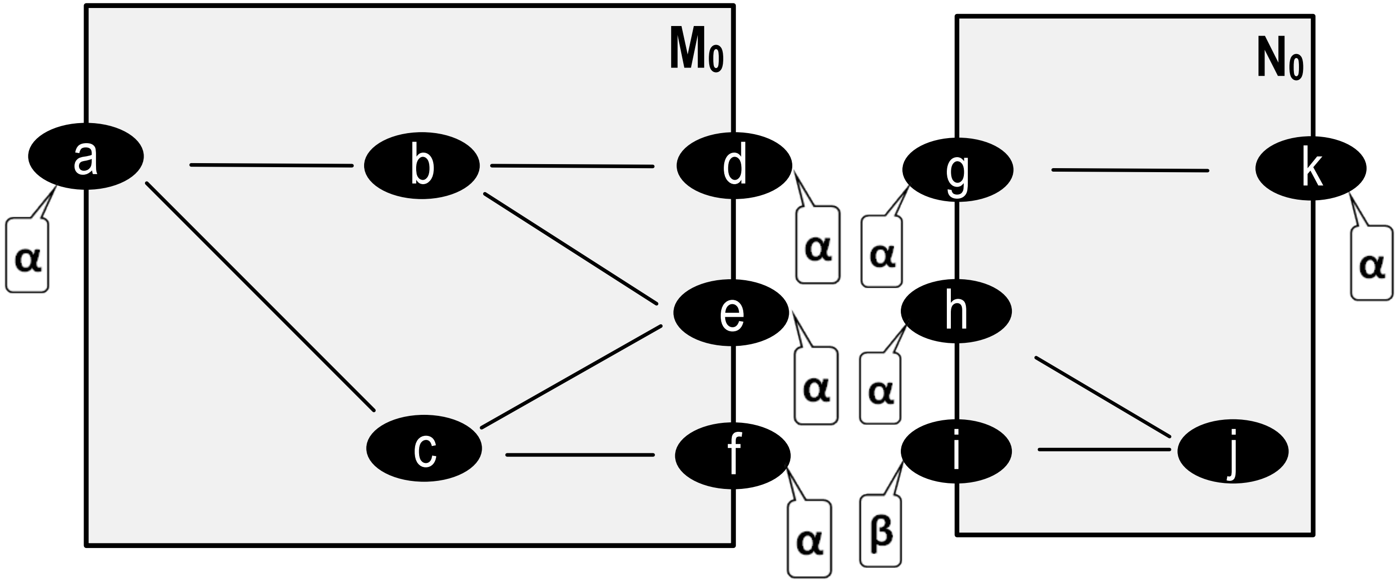
5. Architecture: two faced modules

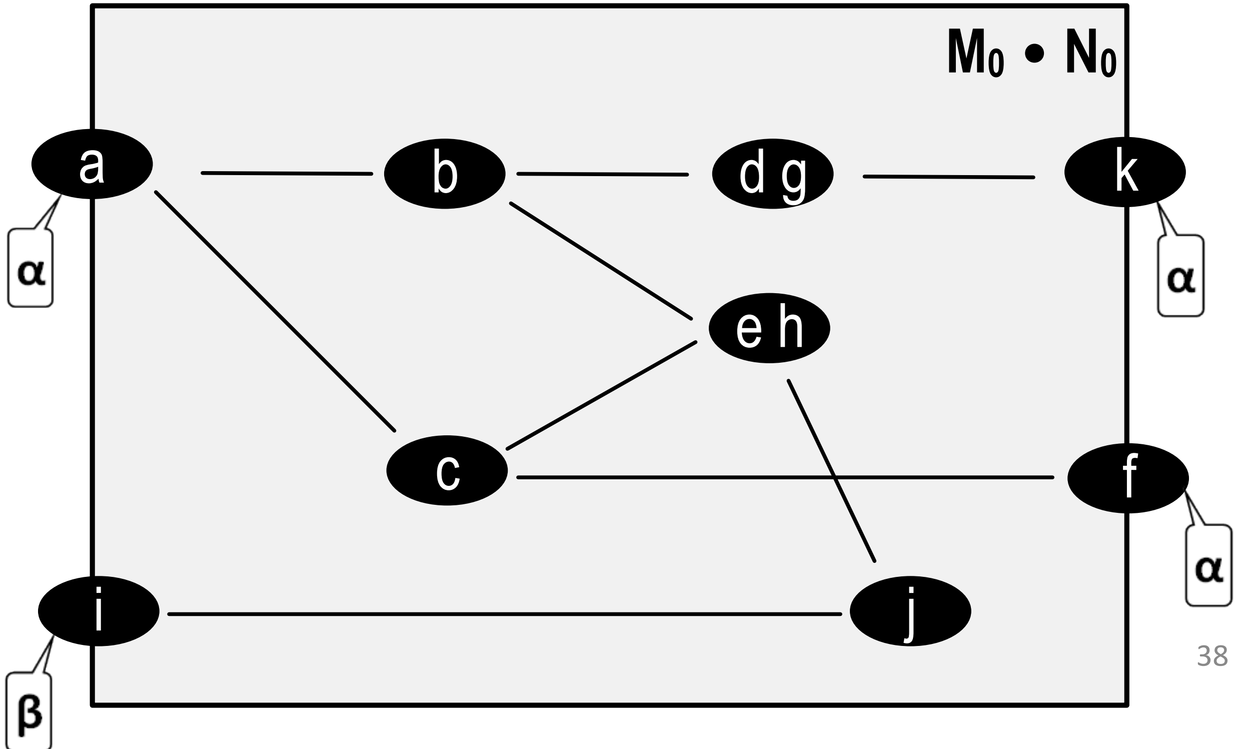
A module is

- a *graph,*
- With two distinguished sets of nodes (left and right *interface*) “*gates*”.
- Each gate is labeled.

Here a second module, N_0.

Composition of M_0 and N_0





5. Architecture: two faced modules

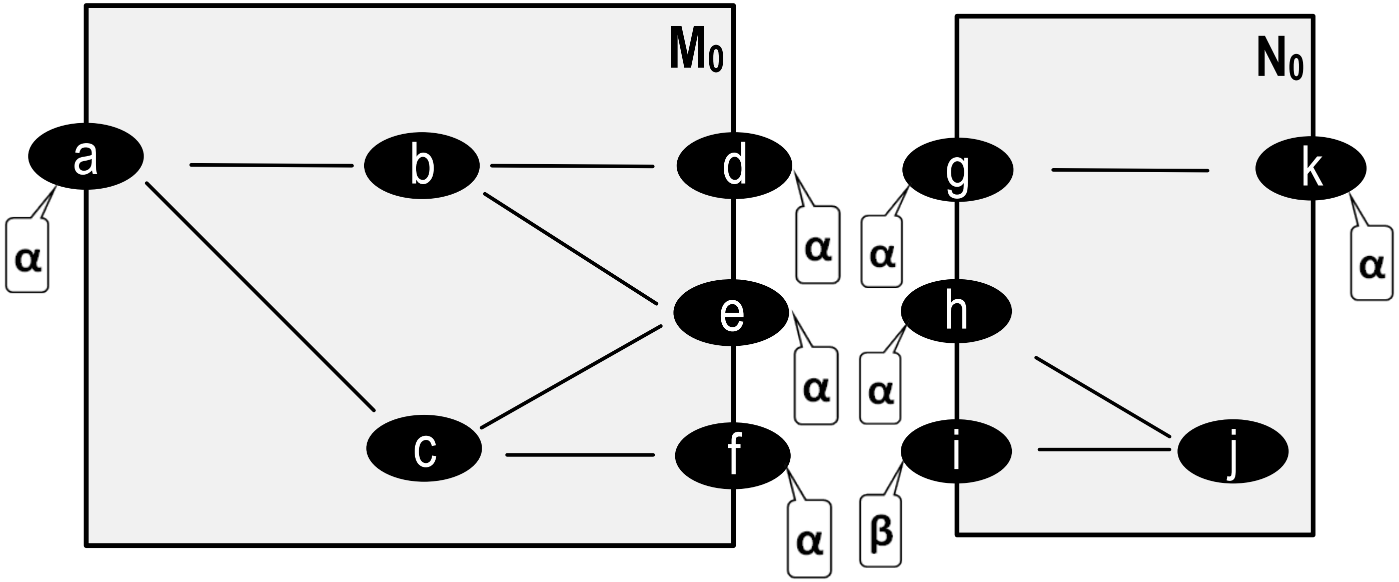
- Any two modules can be composed, resulting in a module.
- Compsosition is associative:

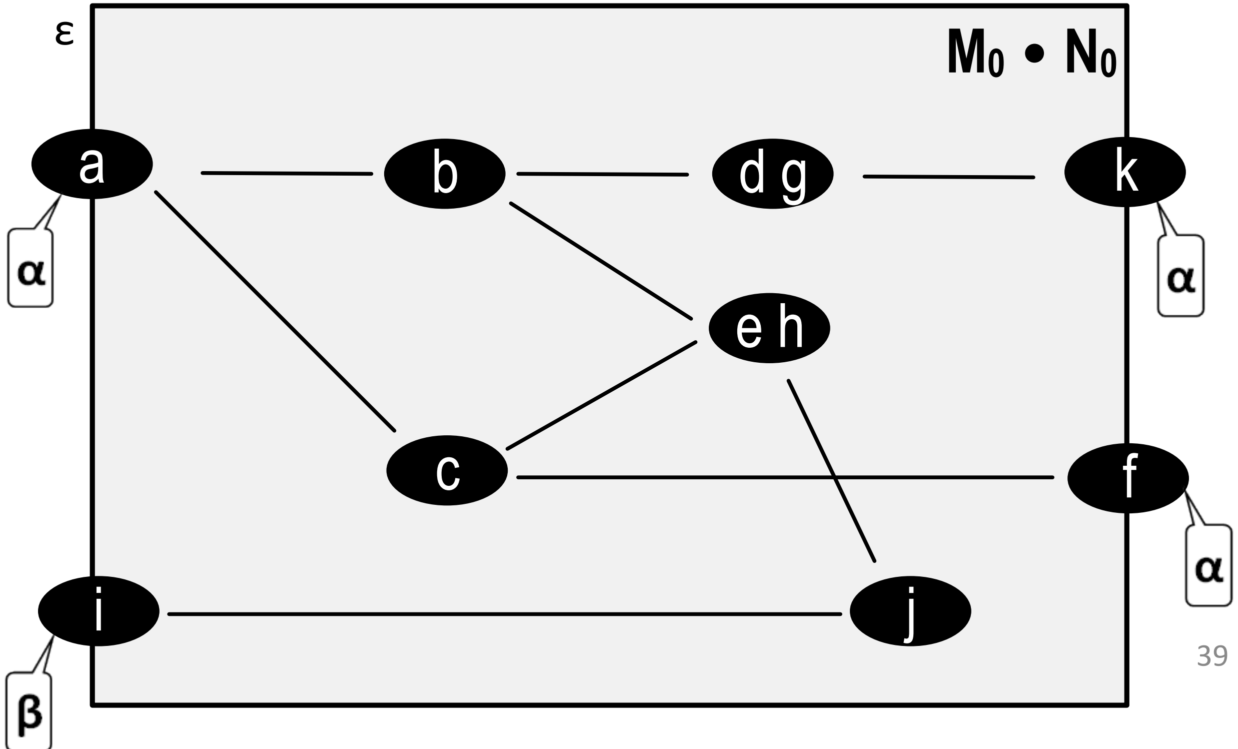
L•(M•N) = (L•M)•N

- For the empty module ε holds:

M• ε = ε • M = M

- A gate may lie in the left as well as in the right interface





6. dynamics: steps: from requirements to models

Peter

6. dynamics: steps: from requirements to models

Peter

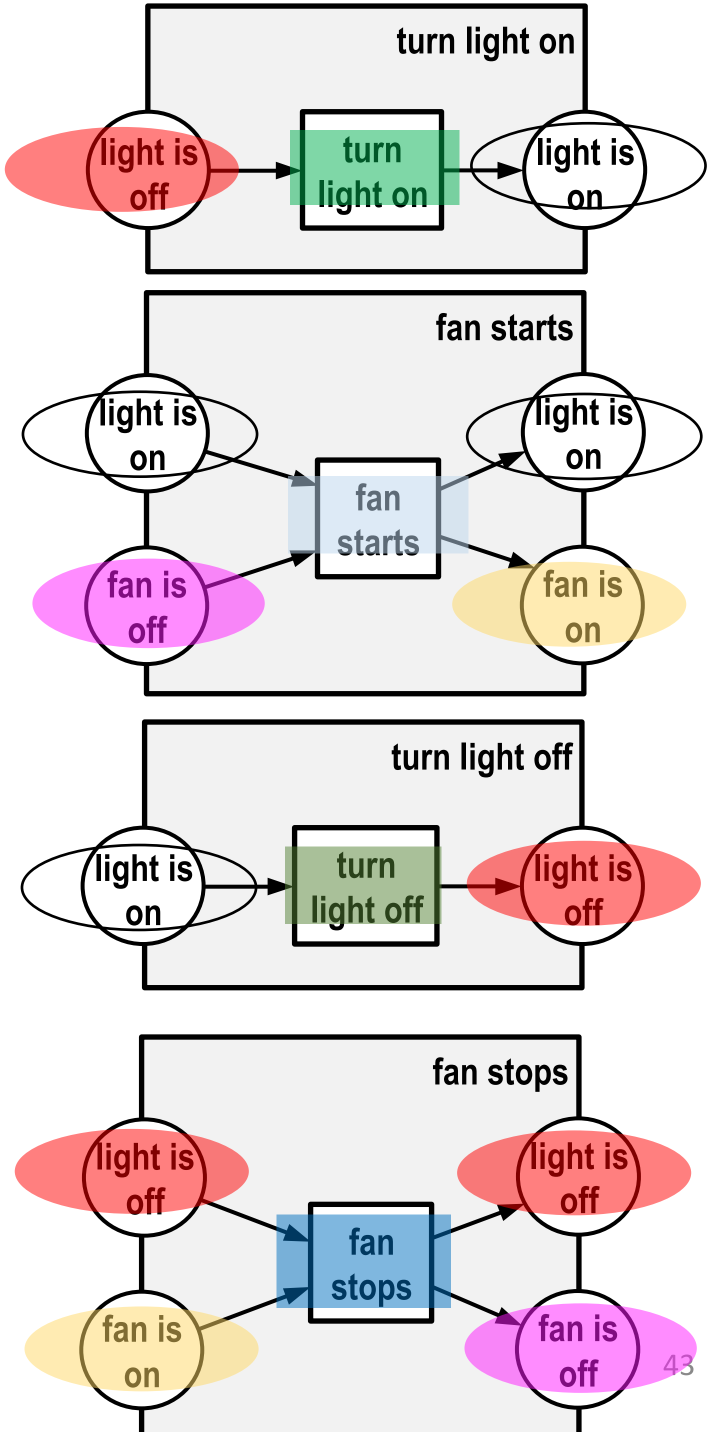
In the case of the fan off, when you turn on the light, after some time, the fan will start running. In this situation, if you turn off the light, the fan continues running for some time. Hence, in the case of the fan off, when you turn on and off the light quickly, the fan will not start running at all. And in the case of the fan on, when you turn off and on the light quickly, the fan will continuously run.

6. dynamics: steps: from requirements to models

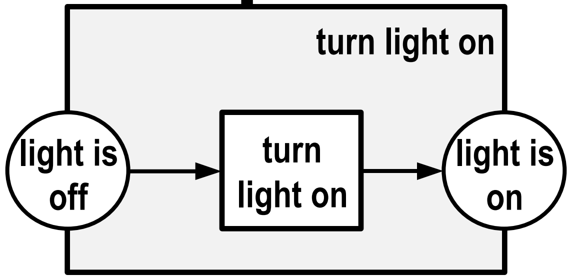
In the case of the fan off, when you turn on the light, after some time, the fan will start running. In this situation, if you turn off the light, the fan continues running for some time. Hence, in the case of the fan off, when you turn on and off the light quickly, the fan will not start running at all. And in the case of the fan on, when you turn off and on the light quickly, the fan will continuously run.

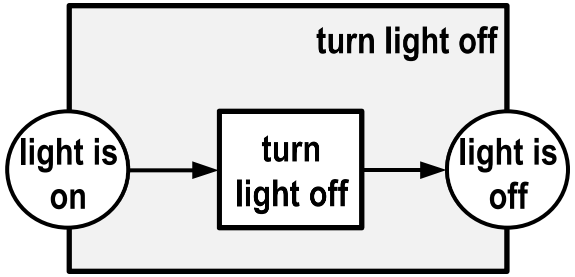
6. dynamics: steps: from requirements to models

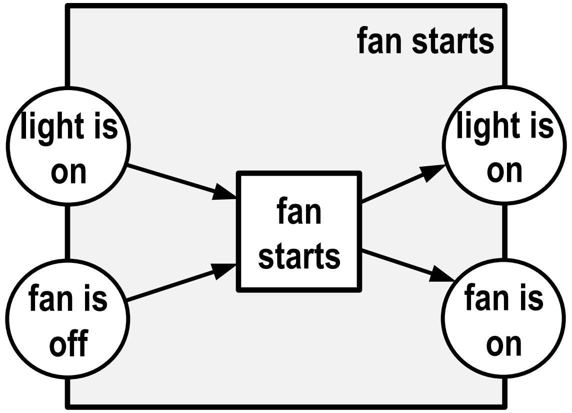
In the case of the fan off, when you turn on the light, after some time, the fan will start running. In this situation, if you turn off the light, the fan continues running for some time. Hence, in the case of the fan off, when you turn on and off the light quickly, the fan will not start running at all. And in the case of the fan on, when you turn off and on the light quickly, the fan will continuously run.

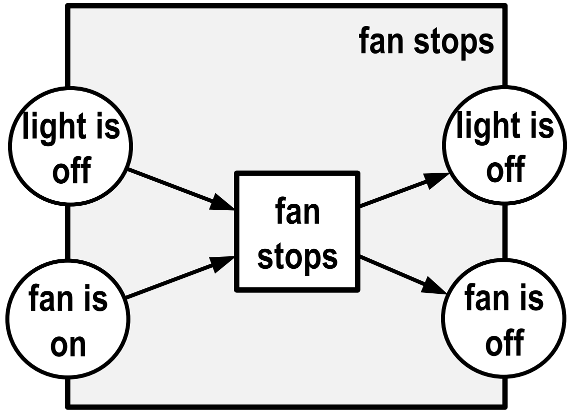


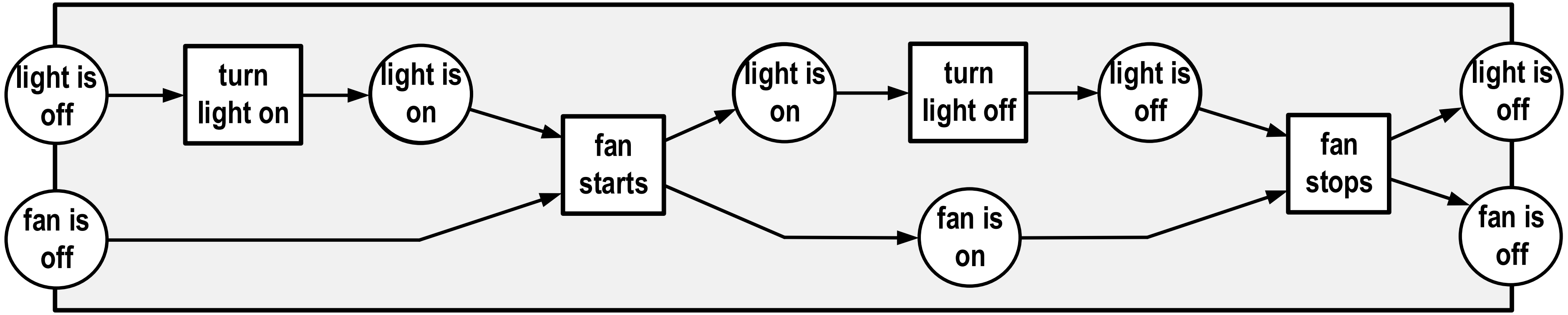
6. dynamics: steps: from requirements to models



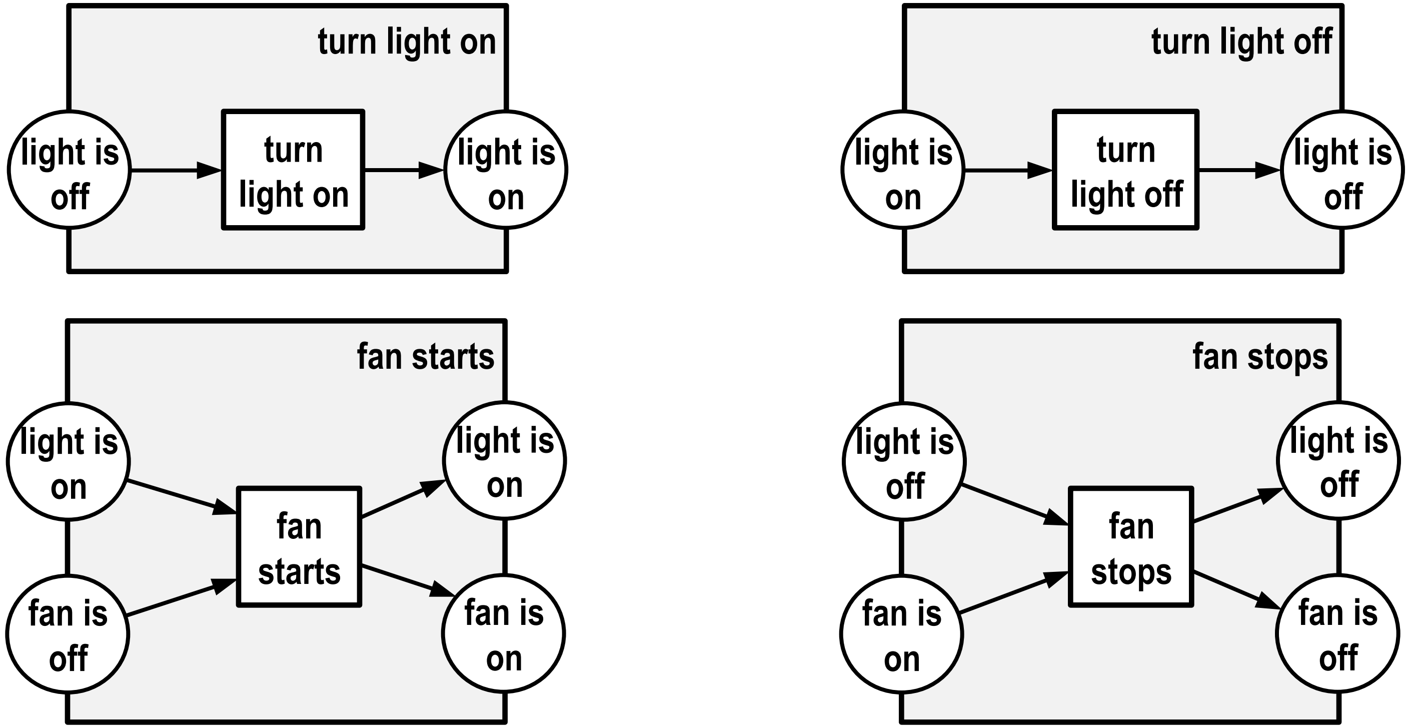


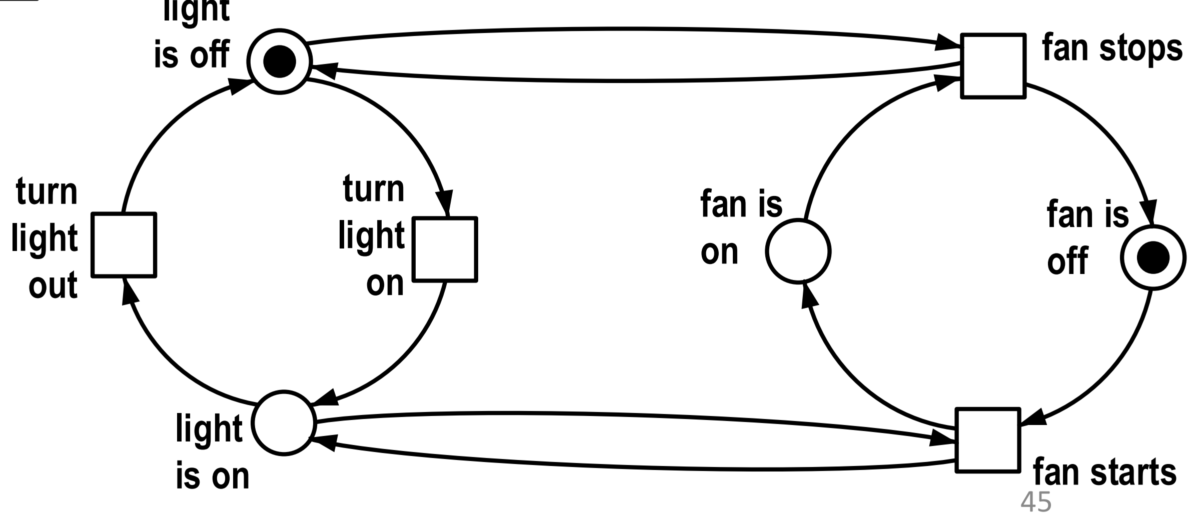






6. dynamics: Example: bathroom fan





7. Breathing live into logic
Varying propositions

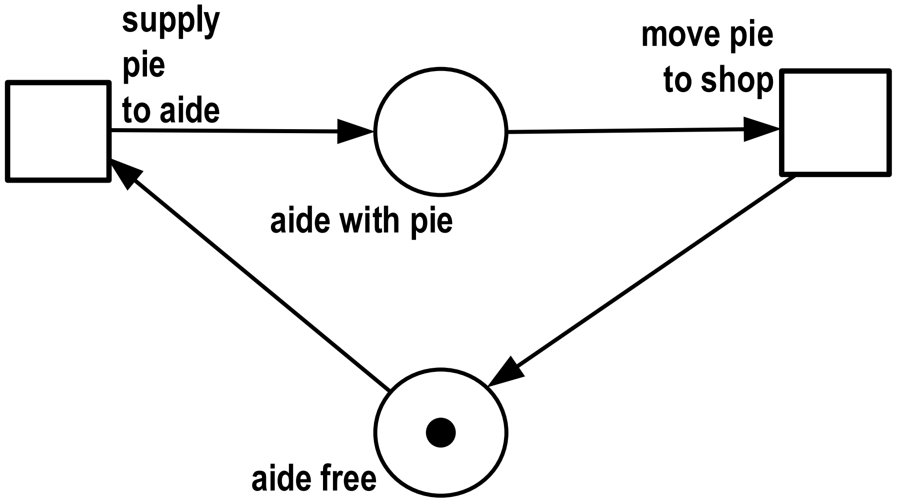
Wolfgang

The notion of *proposition*

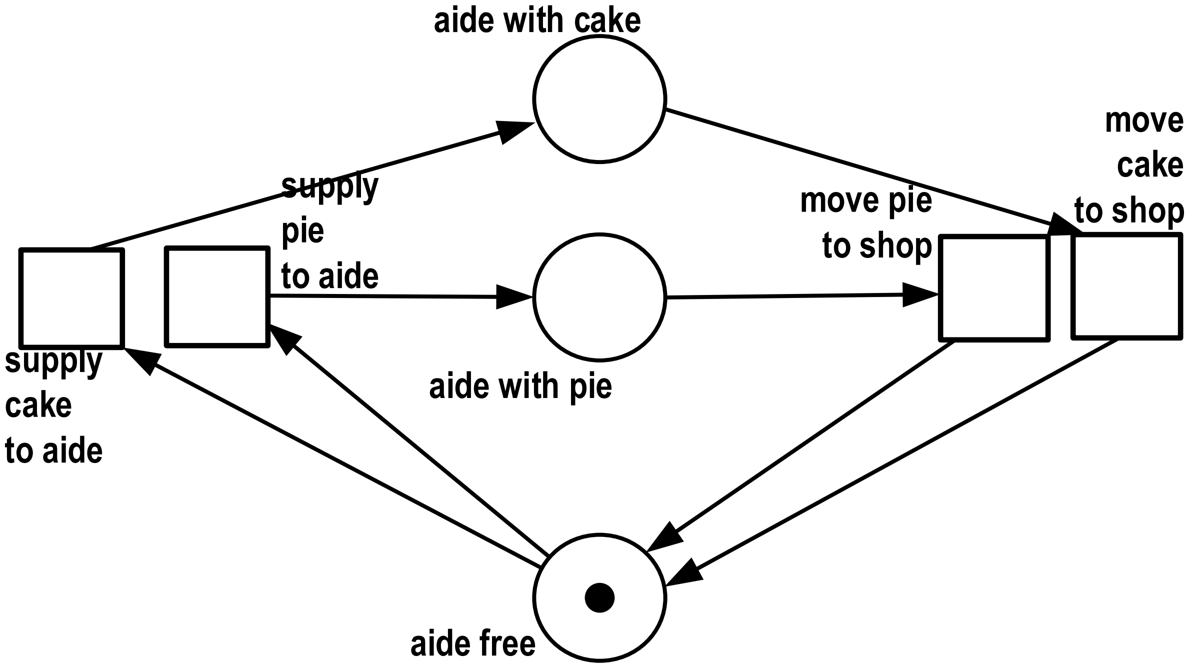
Aristotle:	Petri:
Always true	sometimes true
$e = mc^2$	*A bread lies on the shelve*

This is ***not*** temporal logic!
TL: a proposition abstracts a global state
Petri: a proposition *IS* a (locally confined) state

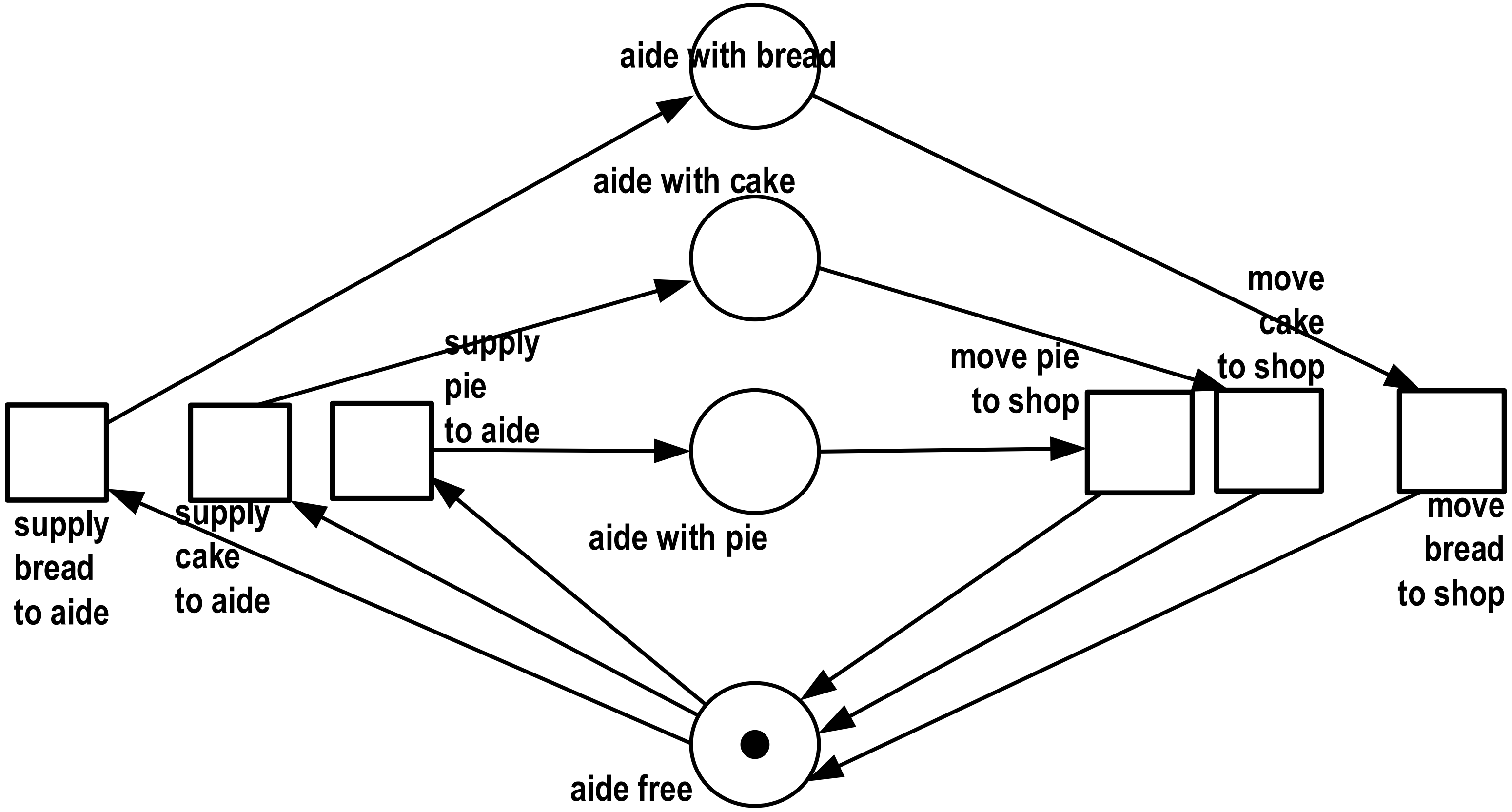
7. Breathing live into logic example



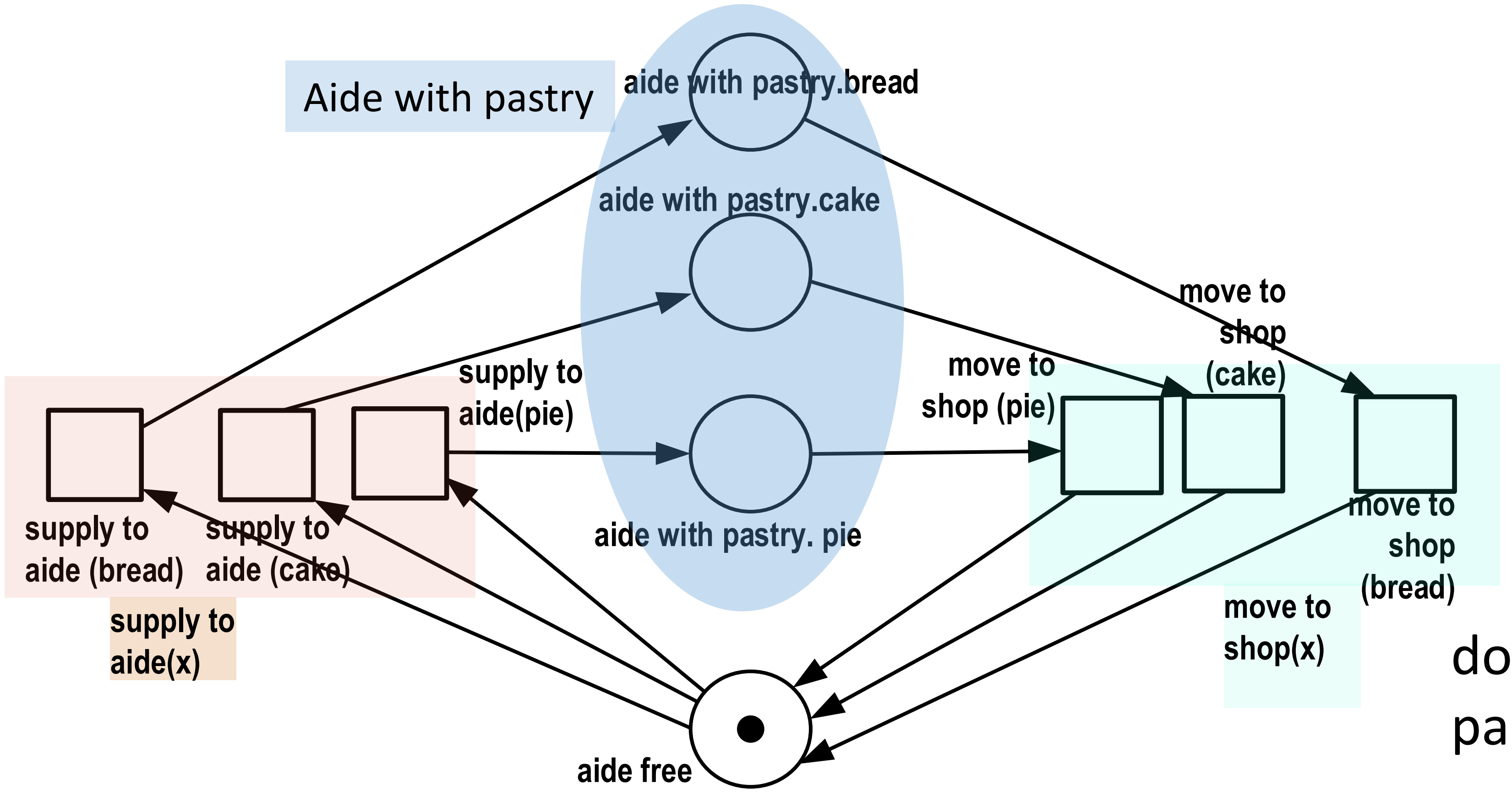
7. Breathing live into logic
Two propositions

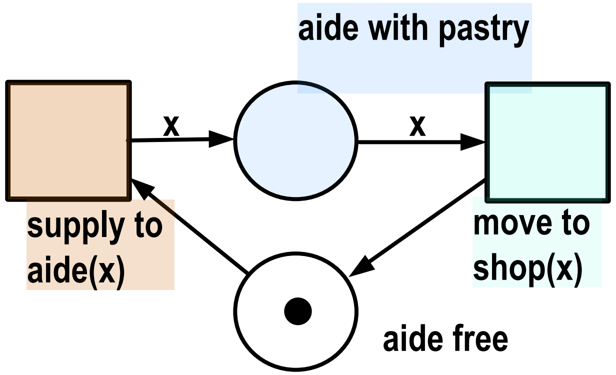


7. Breathing live into logic three propositions



7. Breathing live into logic
Predicates and parameterized events

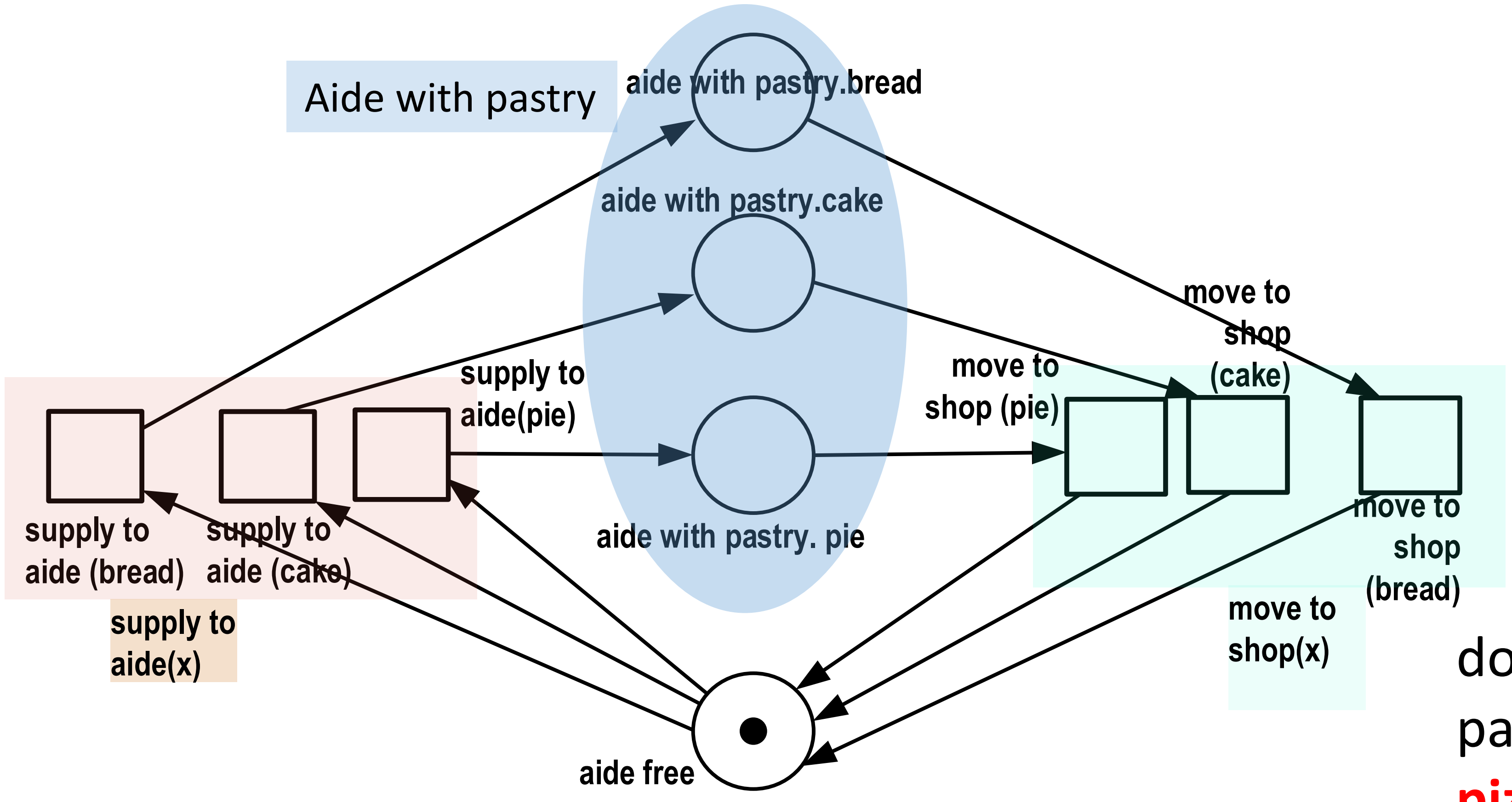


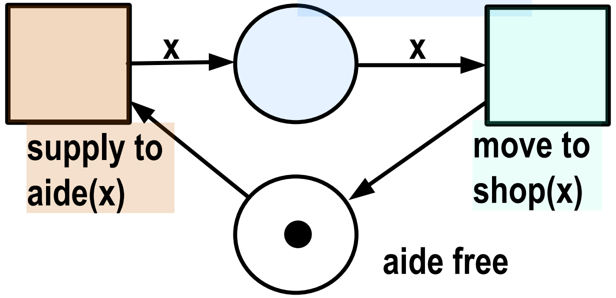


domain
pastry = {pie, cake, bread}

Variable x: pastry

7. Breathing live into logic
Predicates and parameterized e



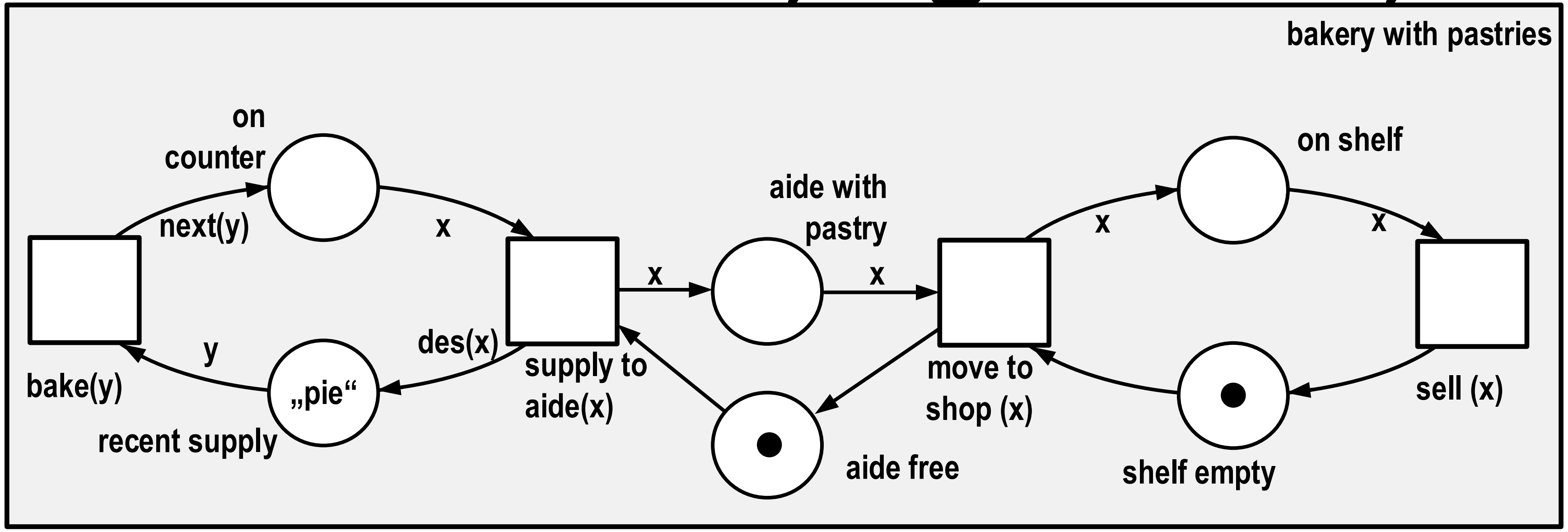


domain

pastry = {pie, cake, bread, **pizza, nudles, fish head**}

Variable x: pastry

7. Structures, signatures, and schemata



Remember …
Three pastries

bakery with three pastries

domains
pastries = {bread, cake, pie}
descriptions = {„bread“, „cake“, „pie“}

function
next: descriptions ⟶ pastries
next("bread") = cake
next("cake") = pie
next("pie") = bread

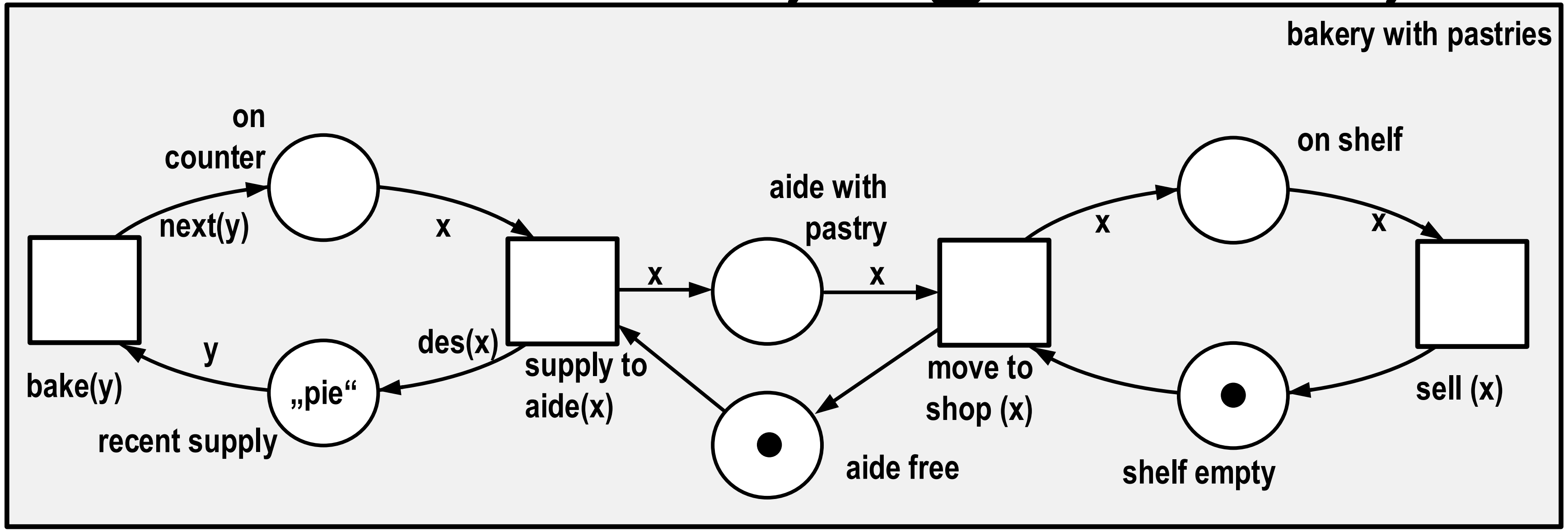
function
des: pastries ⟶ descriptions
des(bread) = "bread"
des(cake) = "cake"
des(pie) = "pie"

constants
„pie“: descriptions

predicates
on counter, aide with pastry, on shelf: pastries
recent supply: descriptions,

propositions
aide free, shelf empty

7. Structures, signatures, and schemata



Now:
Five pastries

five pastries

domains
pastries = {bread, cake, pie, rol, biscuit}
descriptions = {„bread“, „cake“, „pie“, „rol“, „biscuit“}

function
des: pastries ⟶ descriptions
des(bread) = "bread"
des(cake) = "cake"
des(pie = "pie"
des(rol) = "rol"
des(biscuit) = "biscuit"

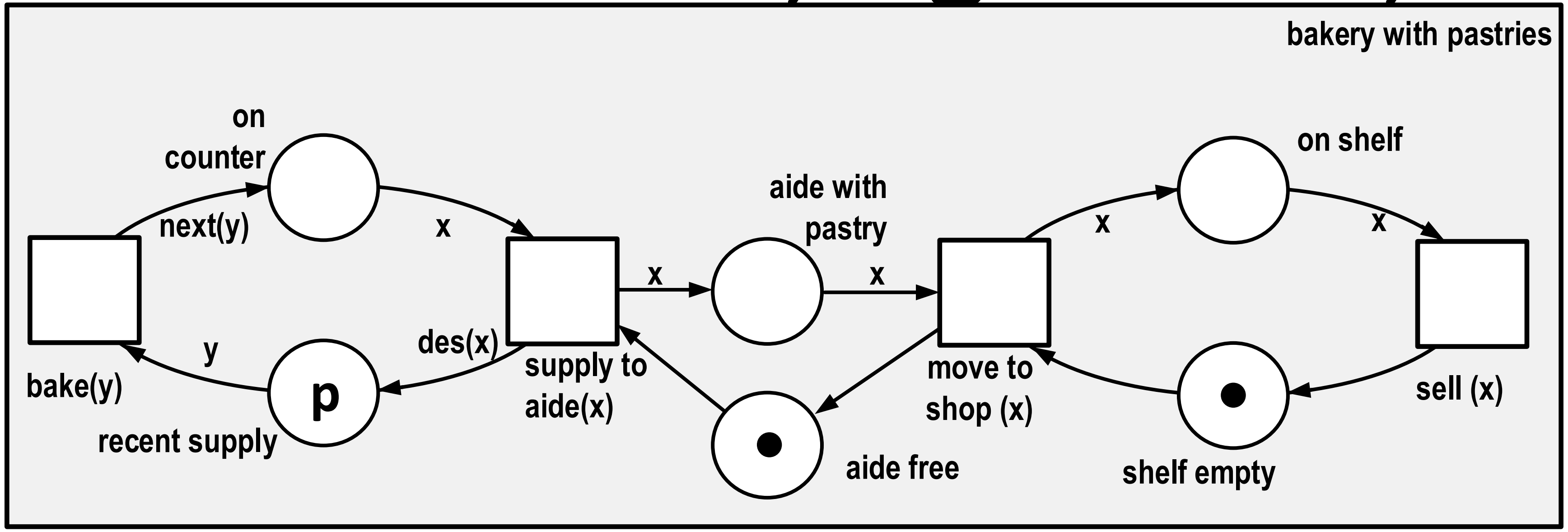
function
next: descriptions ⟶ pastries
next("bread") = cake
next("cake") = pie
next("pie") = rol
next("rol") = biscuit
next("biscuit") = bread

constants
„bread“: descriptions

predicates
on counter, aide with pastry, on shelf: pastries
recent supply: descriptions

propositions
aide free, shelf empty

7. Structures, signatures, and schemata



Schema:
Any set of pastries

signature bakery

domains
pastries
descriptions

function
next: descriptions ⟶ pastries

function
des: pastries ⟶ descriptions

constants
p: descriptions

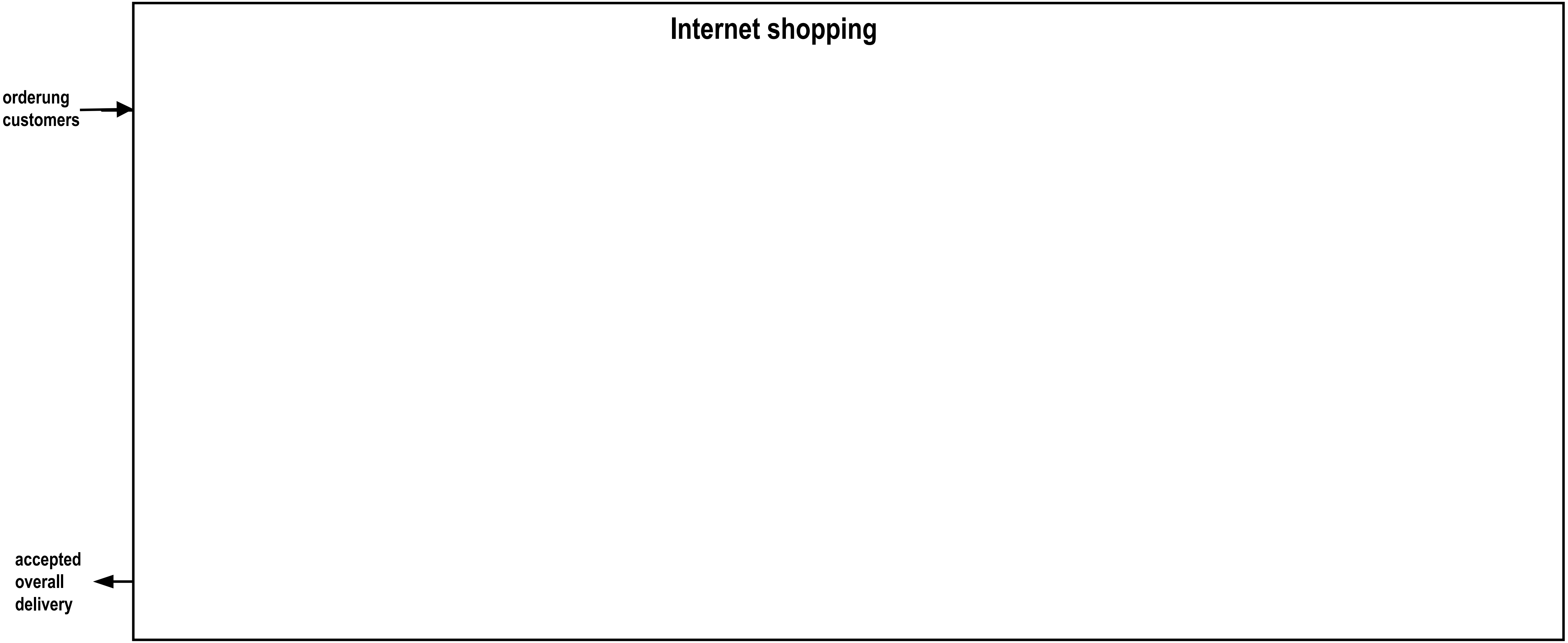
predicates
on counter, aide with pastry, on shelf: pastries
recent supply: descriptions

propositions
aide free, shelf empty

Part III A big case study: an apetizer

abstract view

the busines = customers • [retailer] • [supplier] • [freight forwarder]

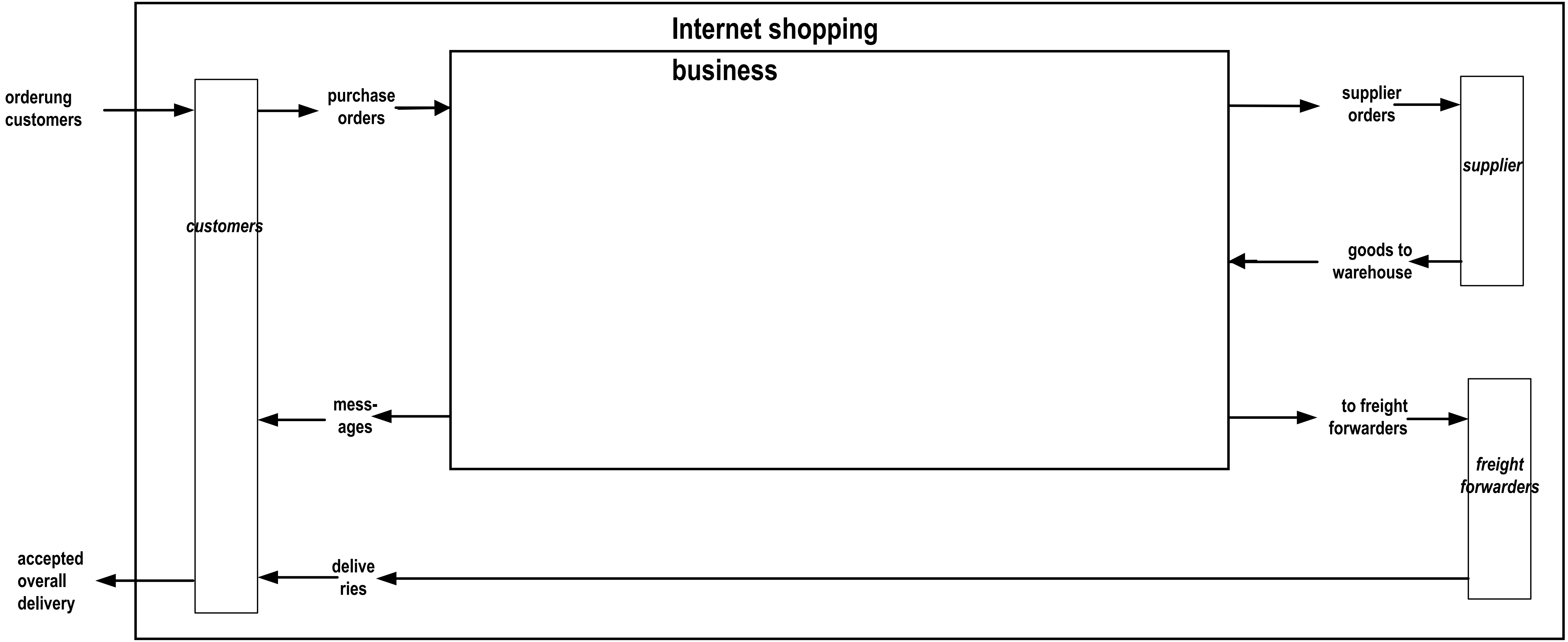


Internet shopping

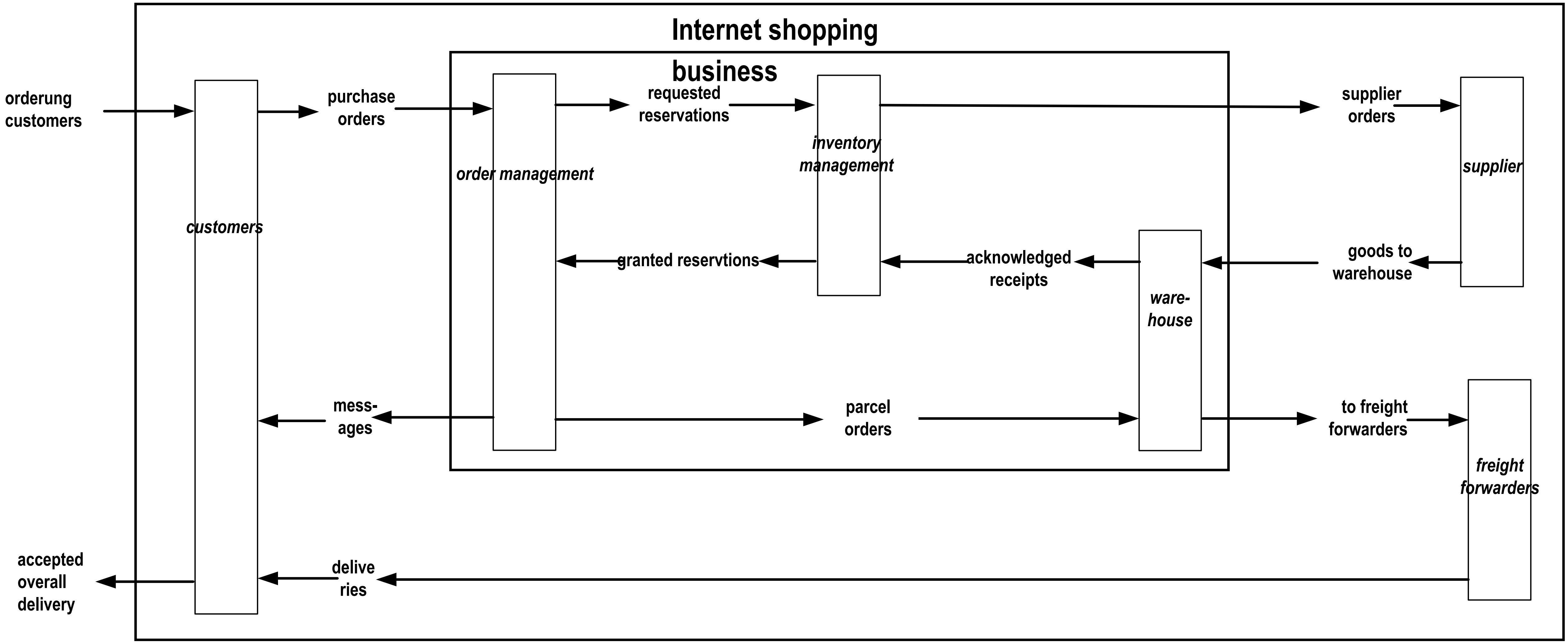
orderung customers

accepted overall delivery

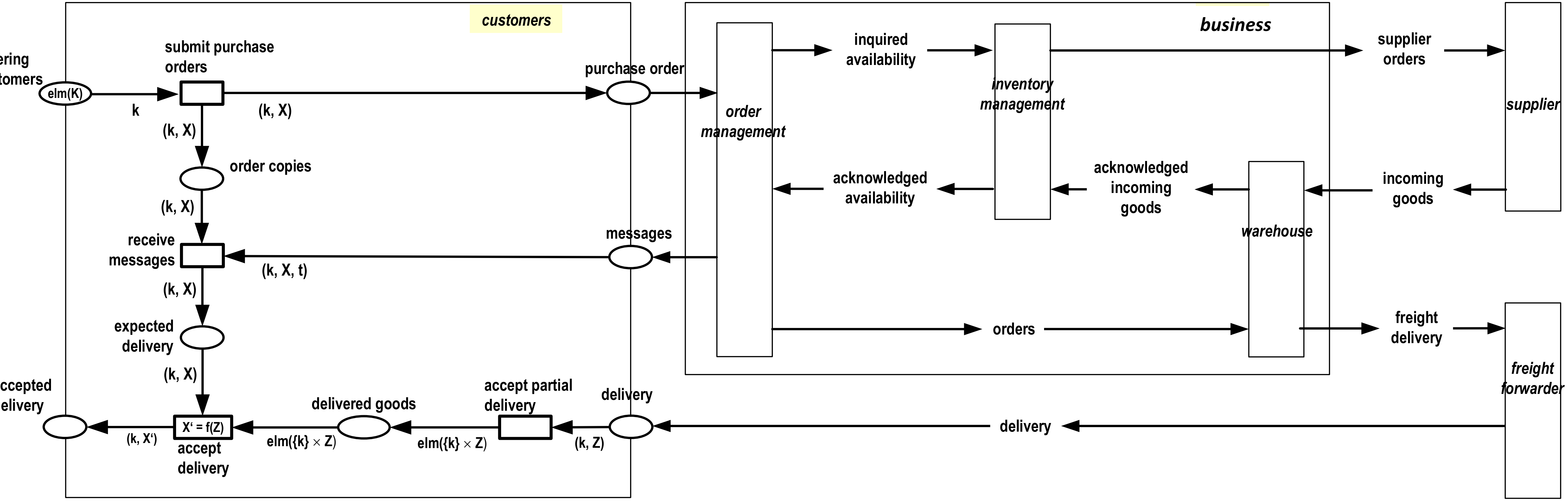
Part III A big case study: an apetizer abstract view



customers • business • freight forwarders • suppliers

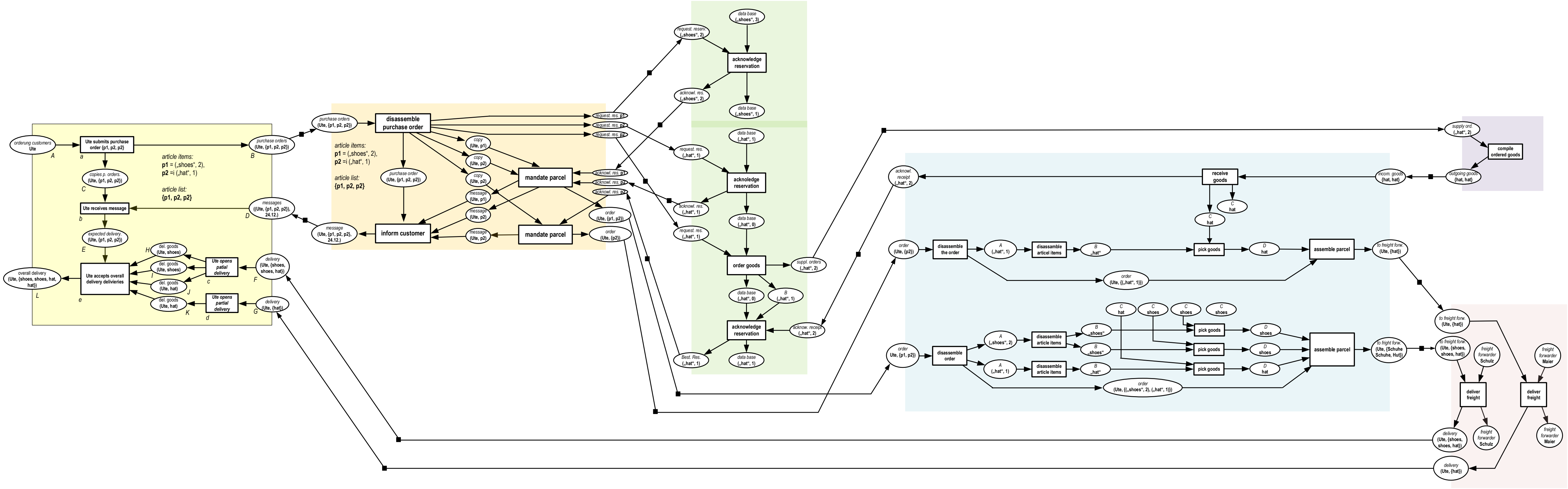


customers • business • freight forwarders • suppliers
where business = order management • inventory management • warehouse



the business = customers • [retailer] • [supplier] • [freight forwarder]

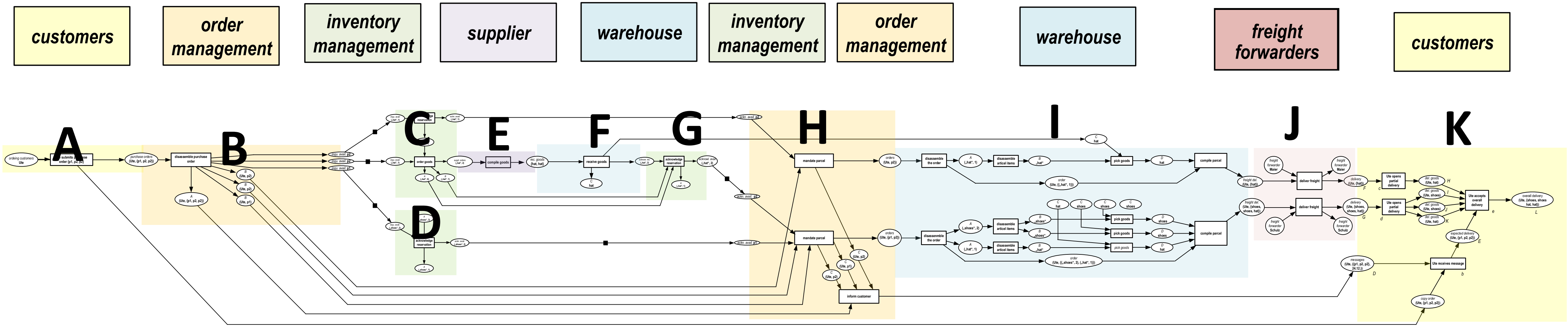
overall run of Alice‘s order



just write

customers • **order management** • **inventory management** • **warehouse** • **supplier** • **freight forwarders**

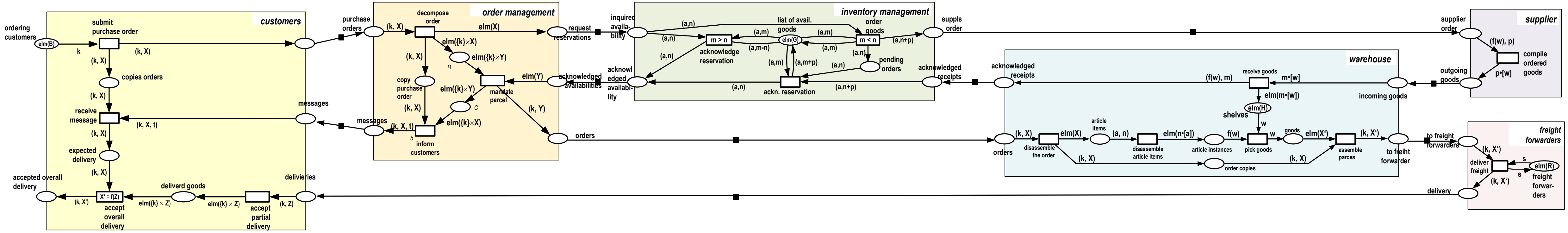
How to represent composition of run snippets?



just write

A • B • C • D • E • F • G • H • I • J • K

schema for interent shopping

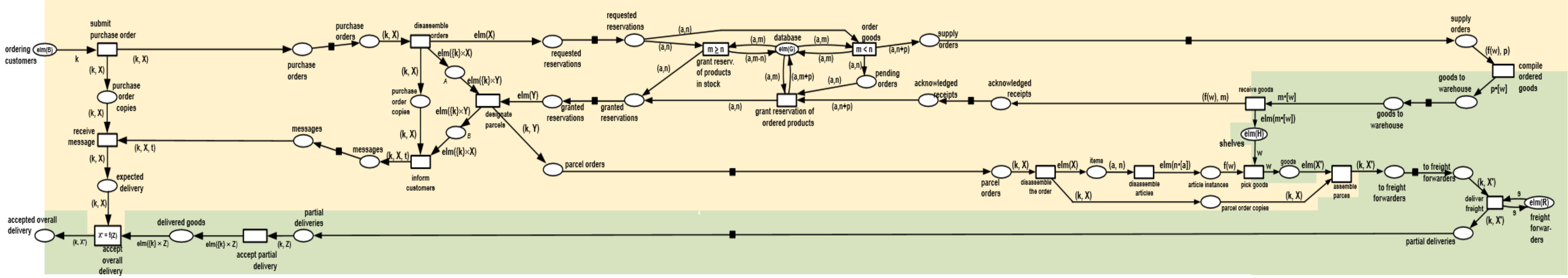


just write

customers • order management • inventory management • warehouse • supplier • freight forwarders

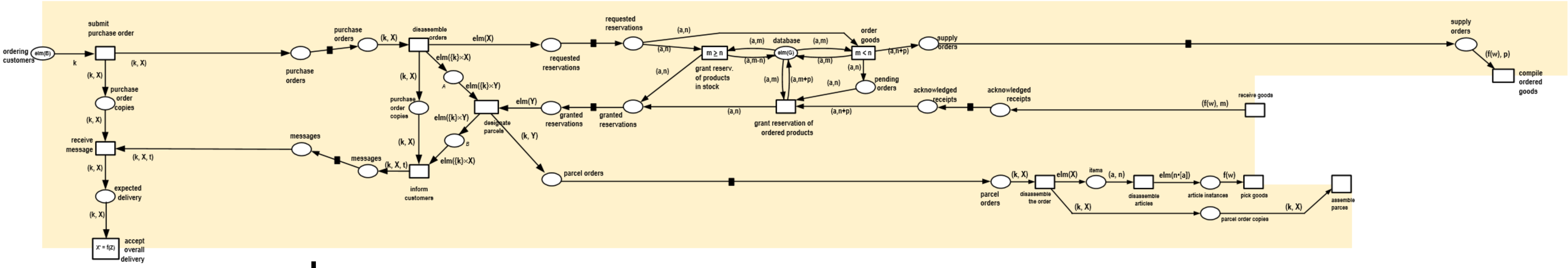
alternative refinement

business = paperwork • items

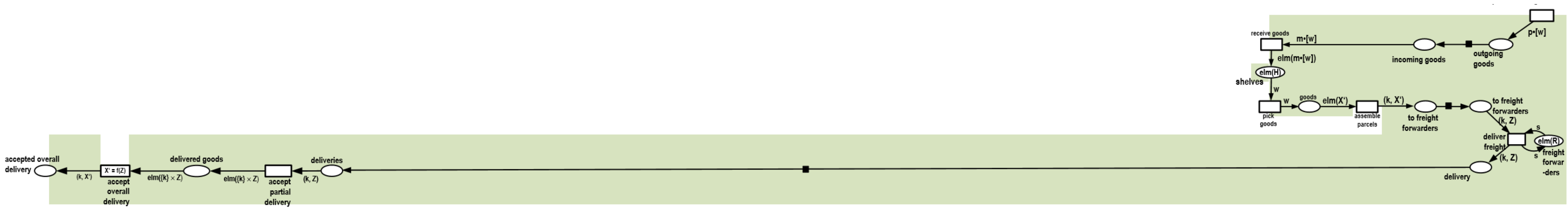


alternative refinement

business = paperwork • items

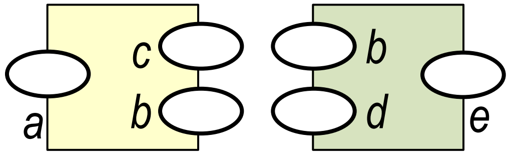
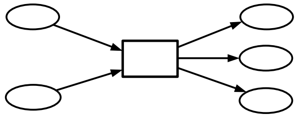


paperwork



items

Summing up: central ideas of HERAKLIT

classical computer science	***… yes, but …***	***… adjusted***	***… such as***	***… technically***
modules and composition: merge “equal” interface elements	yes, however not *one* interface	but two!		composition calculus
statics (data, items): symbolic representation	yes, however not with symbol *chains* (“strings”)	but with terms over a signature!	**f(x, g(a,y))**	predicate logic, algebraic specification
dynamics: steps	yes, however, not *global* states and steps	but local ones!		Petri nets

classical computer science

- jumps in the right direction
- but falls short

HERAKLIT adjusts this!

Programming is about symbol crunching.
Modeling is about the digital world!