**Intern. Symposium on Quality Engineering for Embedded Systems**
**June 13, 2008**
**Berlin, Germany**

# Improving Quality Factors in Model-Based Embedded Software

Luca Pazzi, Marco Pradelli
University of Modena and Reggio Emilia
Department of Information Engineering DII-UNIMORE
Via Vignolese 905, I-41125 Modena, Italy
{luca.pazzi,marco.pradelli}@unimore.it

# A direct road from software quality to state semantics computability

- In this talk we show how Part –Whole Statecharts, originally created in order to improve software quality, showed themselves to have a semantics which is directly computable;
- We will try to suggest in the talk that a direct relationship can be established amongst software quality factors, true compositionality and semantics computability;

# Software Quality Factors in Behavioral Abstractions

- Meyer software quality factors:
  - Reusability;
  - Understandability;
  - Manutenability;
- Depend critically on *well known* prerequisites
  - self-containment;
  - loose coupling;
  - information hiding.
- We will show that no one of the prerequisites are met by current state-of-the-art modelling tools.

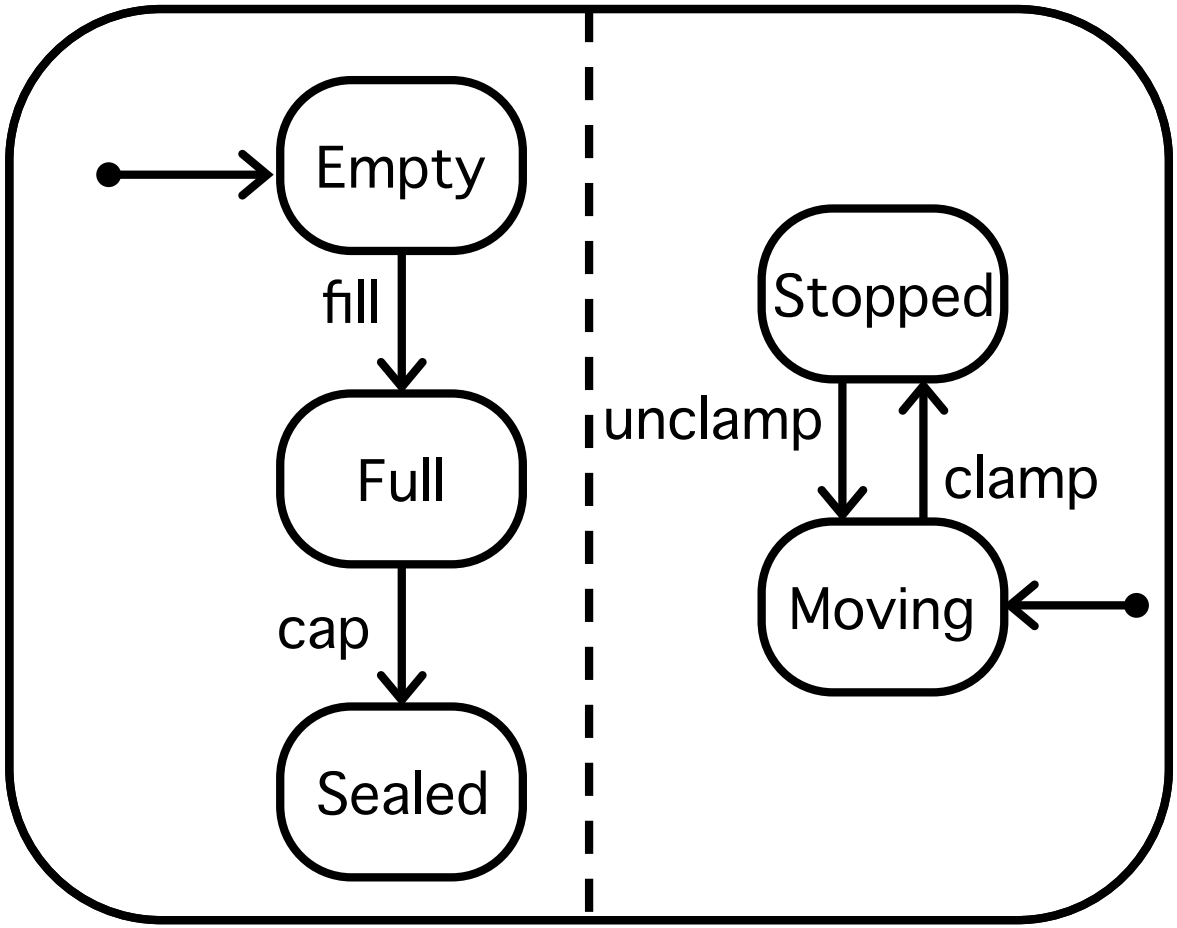
# Modelling behavioral abstractions by state machines

- State-based formalism are
  - Very expressive;
  - Easy to understand and to exchange;
  - Easily formalisable;
    - A state can be seen as a snapshot of the world without further explanations:
      - For example: Open, Closed, Middle, Filling, and so on.
    - A state can be seen at the same time as a bunch of properties being modelled:
      - Open == (ValveDoorAngle = 90  && MagnetVoltage = 0.0).
  - Directly executable!

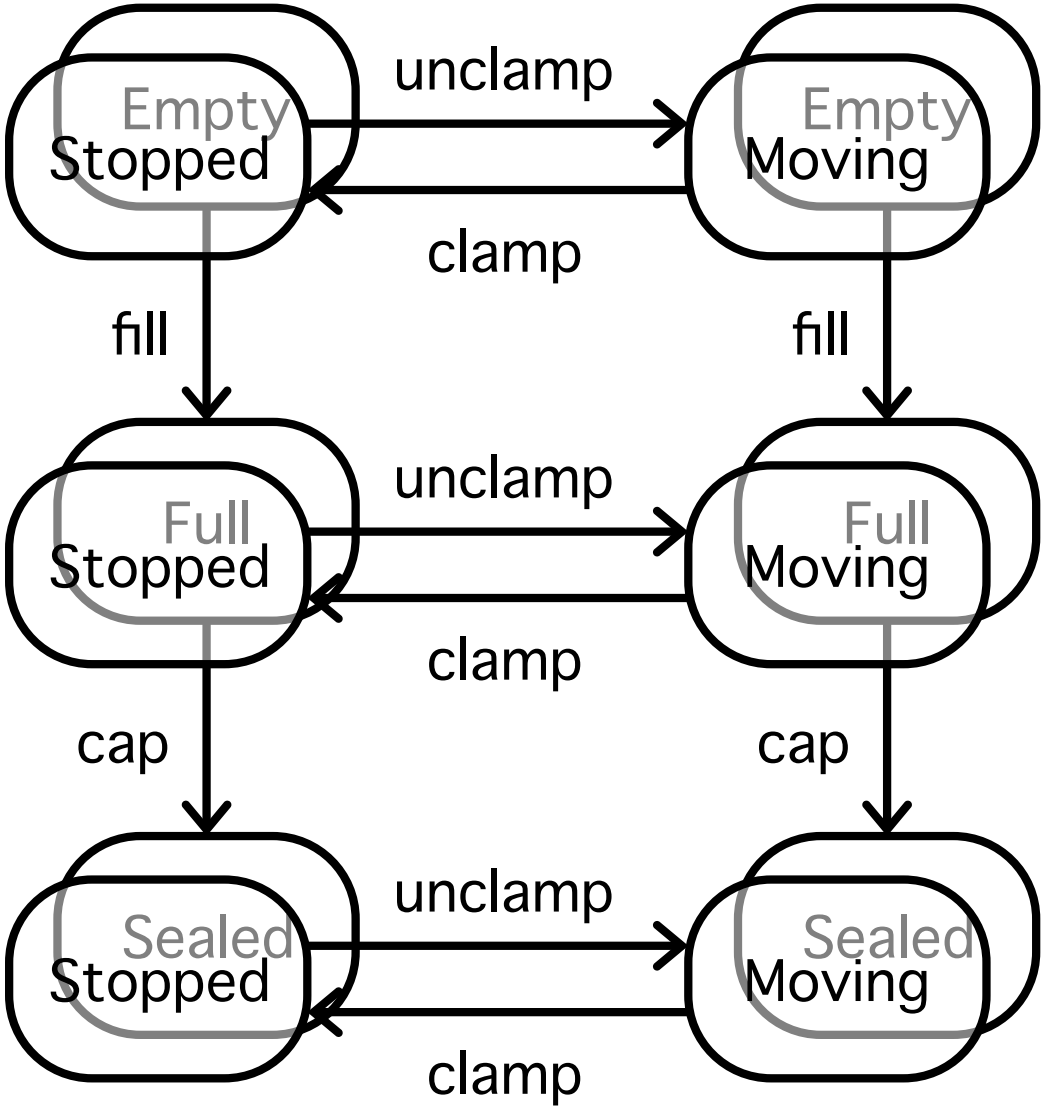
# Composing behavior

- In order to be effective, state-based abstractions need to be specifiable, modifiable as well as testable and certifiable *incrementally*:
  - Off-the-shelf abstractions… was the dream achieved after 40 years?
  - Harel state modelling paradigm provides modelling primitives that can be used in order to compose separate behaviors…

# Composing different aspects of a bottle behavior



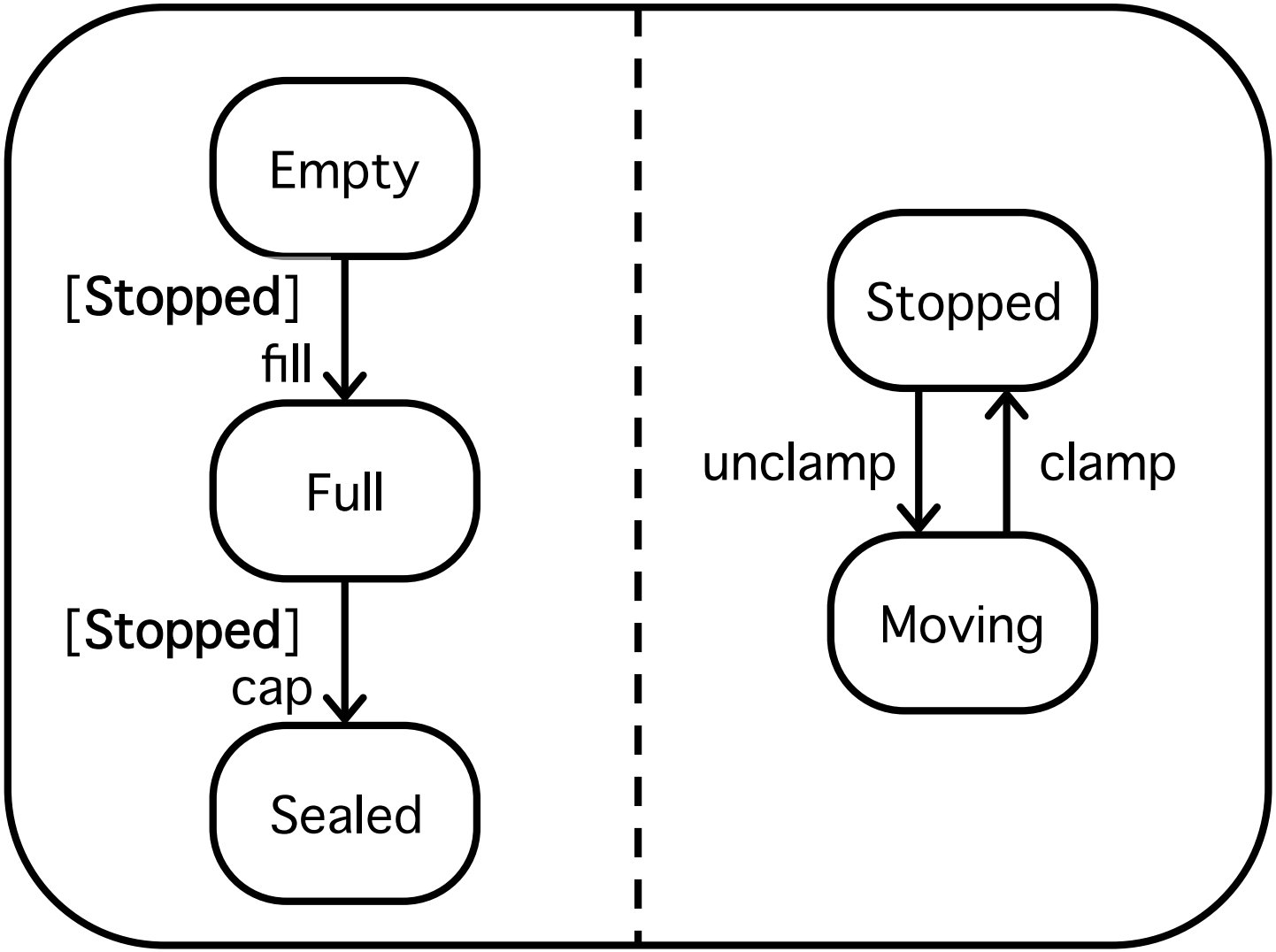
# Unrestricted Cartesian product automaton

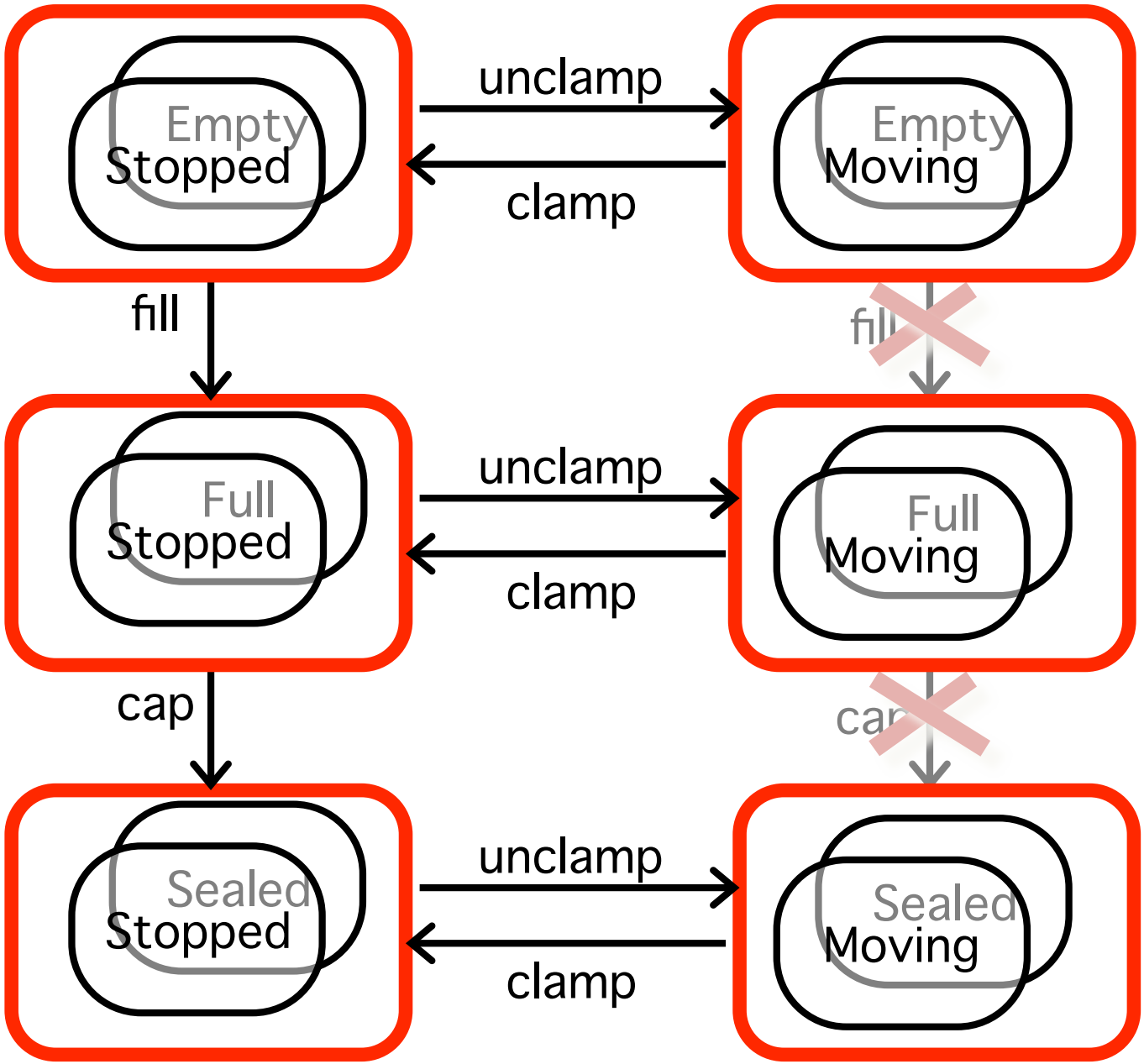


# Composing behavior

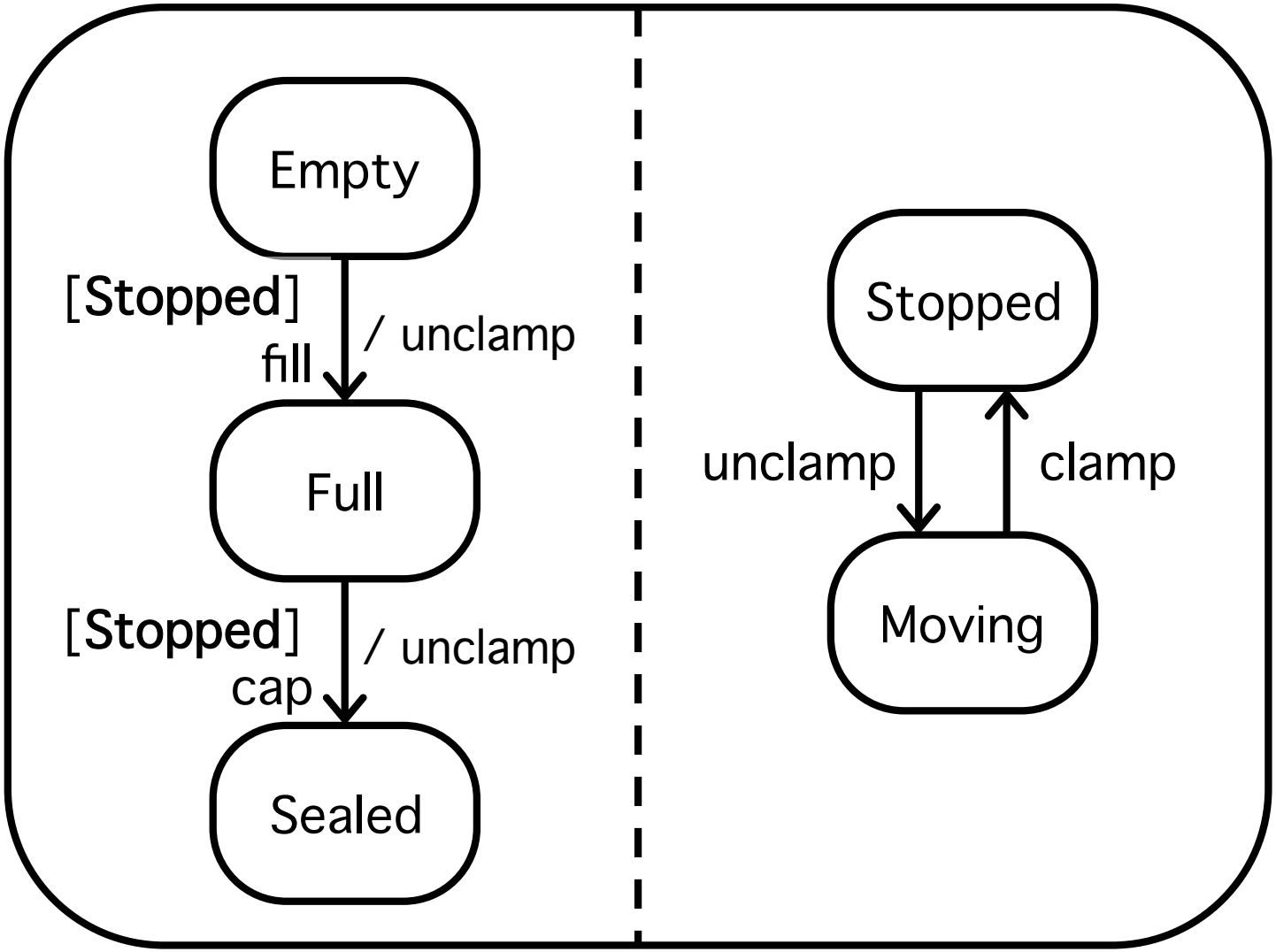
- In order to obtain something useful, we have to constrain the unrestricted global Cartesian automaton:
  - We have to **stop** the bottle in order to fill it!
  - We have to **stop** the bottle in order to seal it!

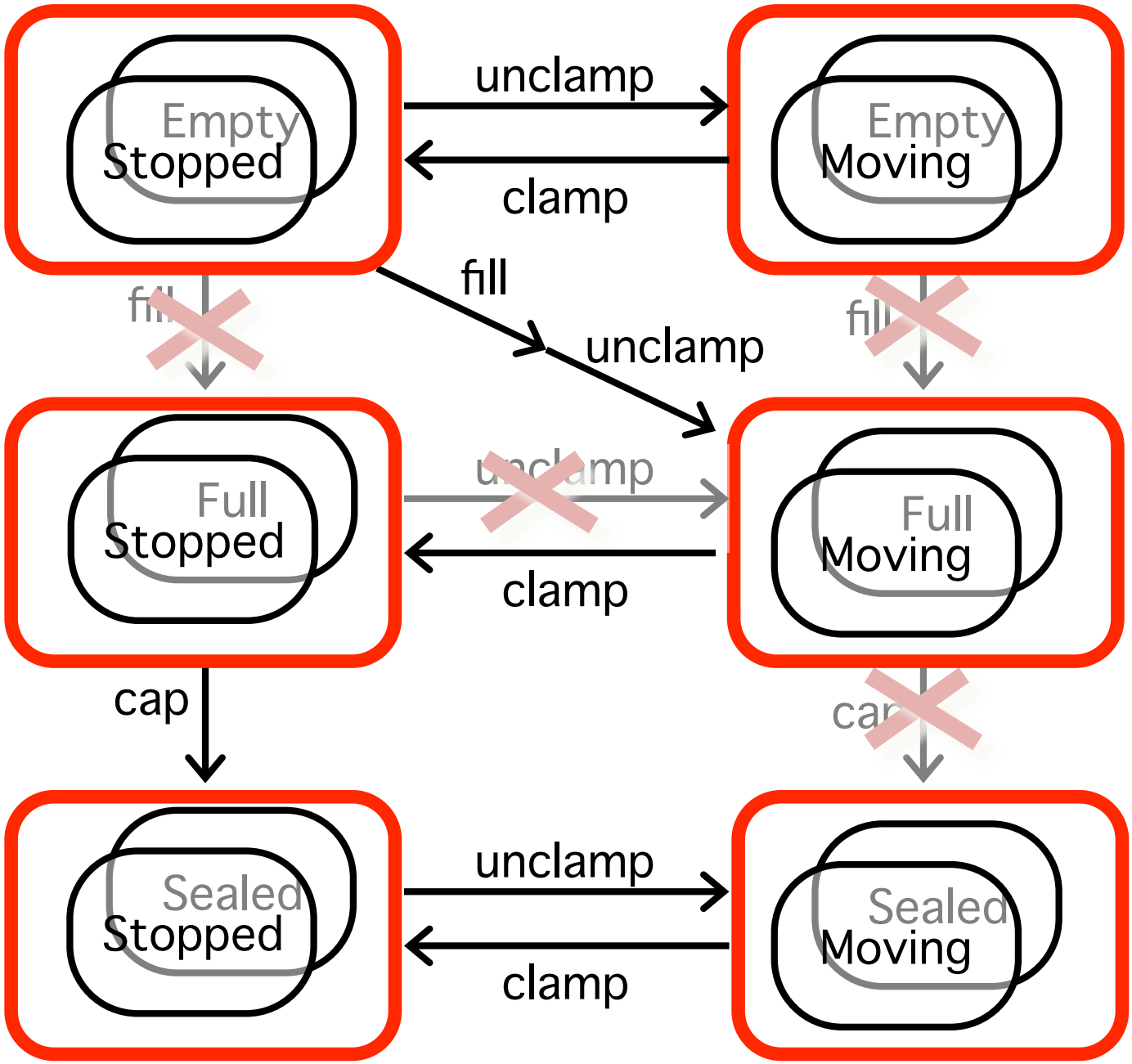
# Constraining behavior



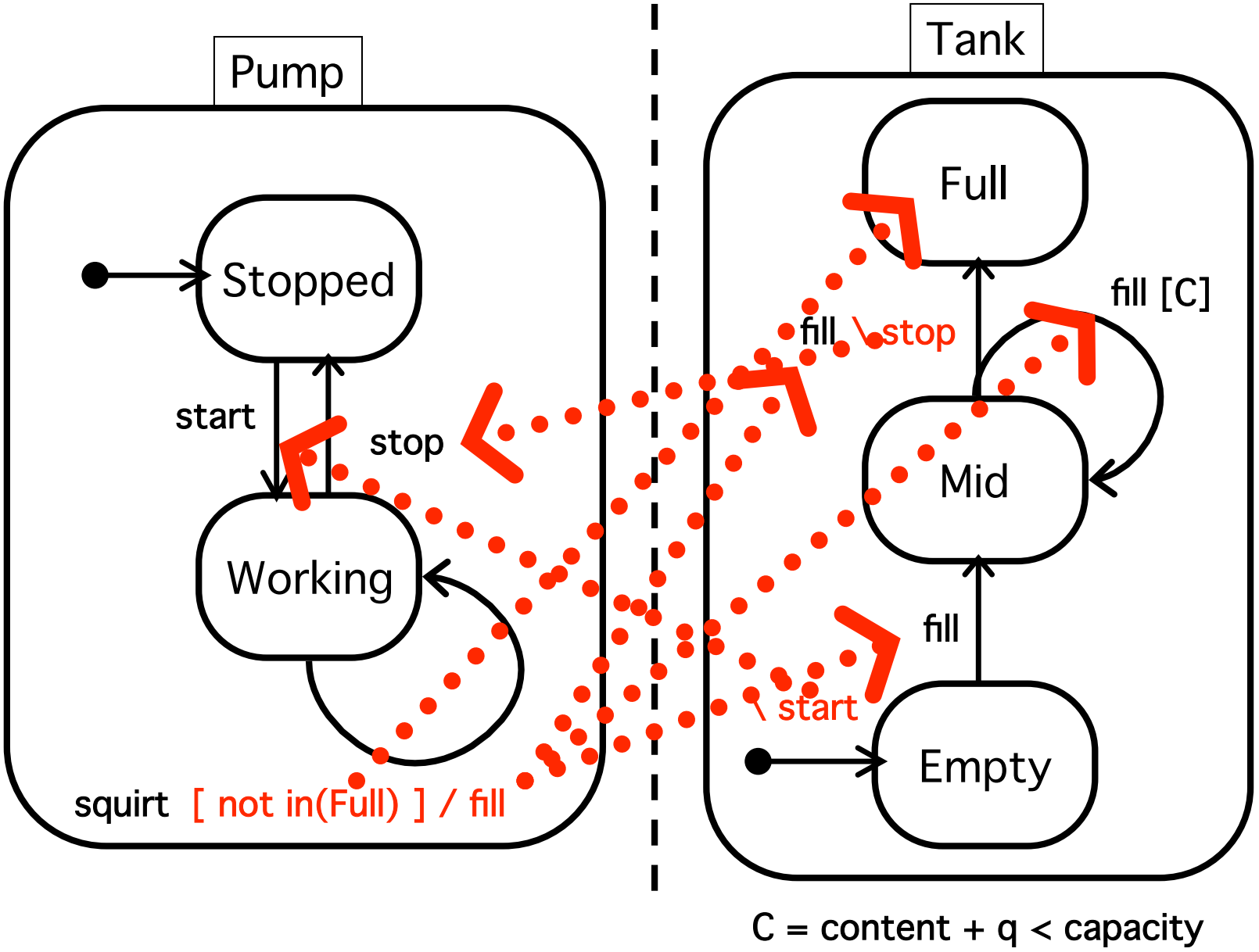

Empty
Stopped
unclamp
clamp
Empty
Moving
fill
fill
Full
Stopped
unclamp
clamp
Full
Moving
cap
cap
Sealed
Stopped
unclamp
clamp
Sealed
Moving

# Further constraining behavior




Empty
Stopped
unclamp
clamp
Empty
Moving
fill
unclamp
Full
Stopped
clamp
Full
Moving
cap
Sealed
Stopped
unclamp
clamp
Sealed
Moving

# Filling a tank through Statecharts!

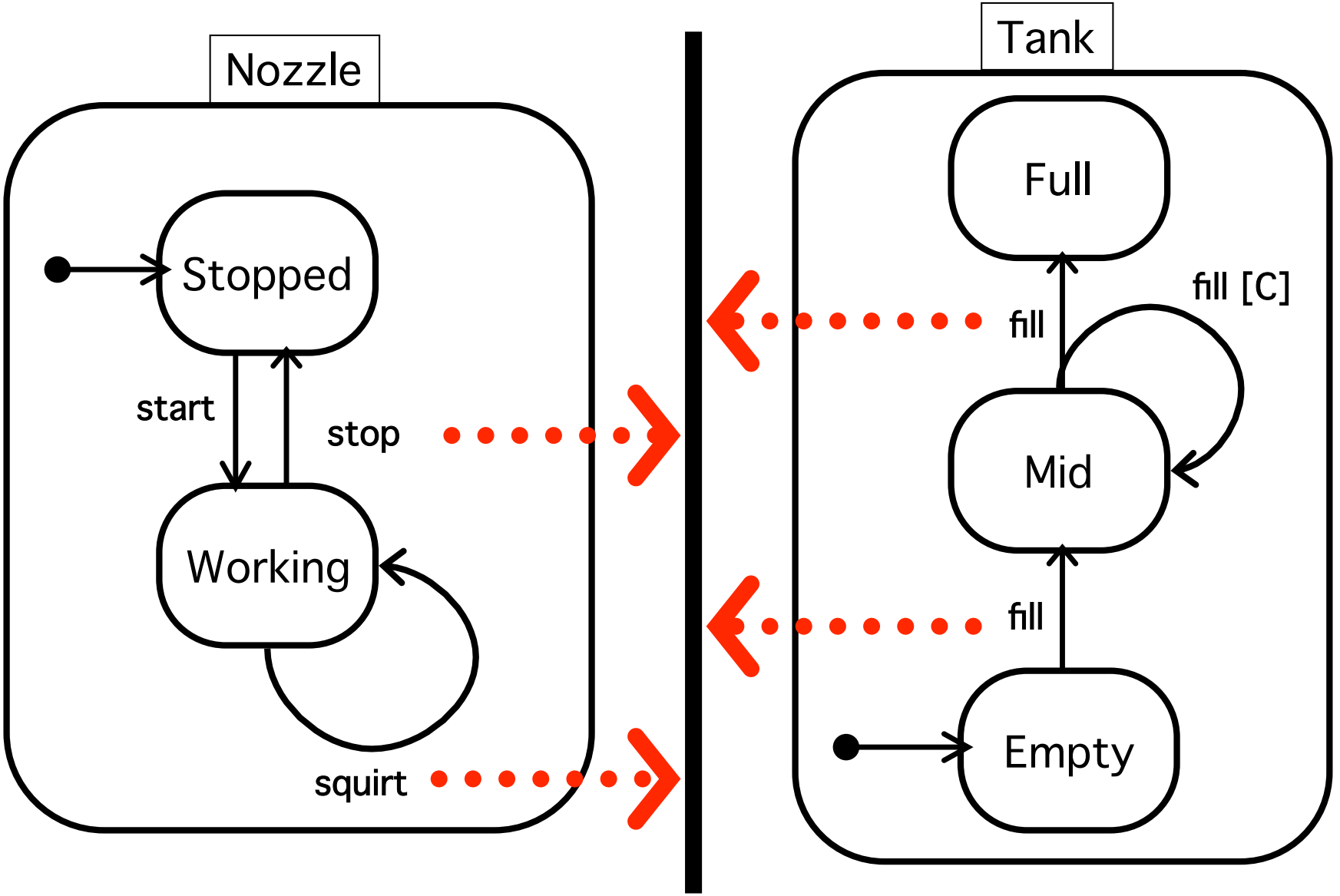


# Composing behaviors by ordinary Statecharts is not effective

- Tightly coupled abstractions jeopardize
  - **Reusability**: behaviors mutually depend on the other behaviors;
  - **Understandability**: we have to look around in order to grasp the exact meaning;
  - **Modifiability**: any new functionality added to one of the component behaviors requires potentially to the other;

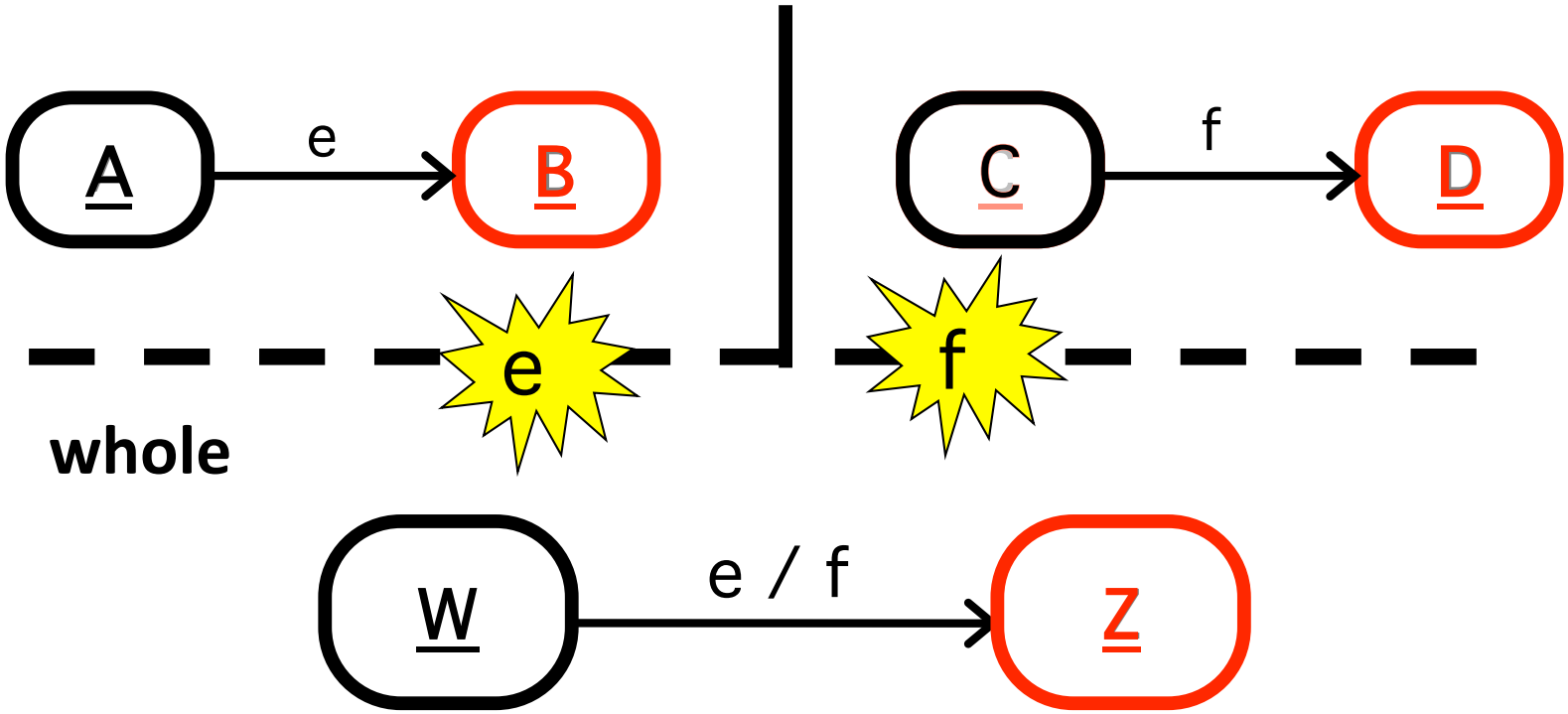
  - **Semantics** not computable… **pinball effect**! Event bounce from a behavioral abstraction to the another: will they stop? Mutual dependence among abstractions… **deadlock**!

# Avoid mutual knowledge…

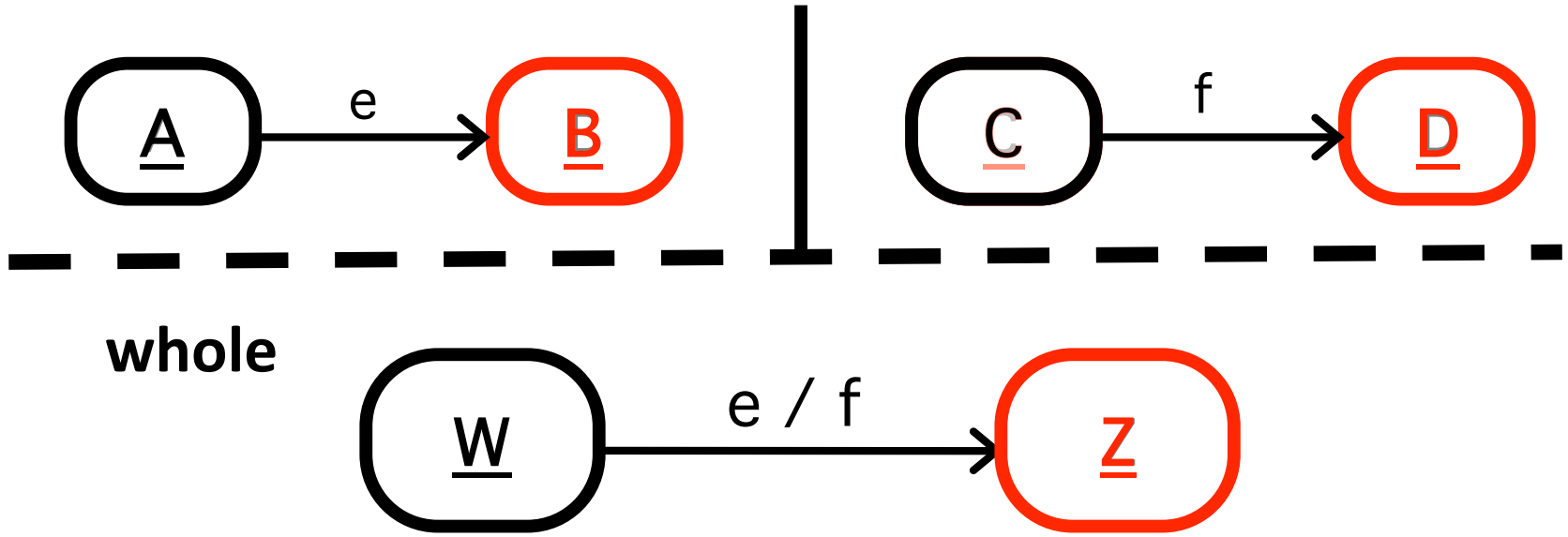


# How can we communicate… if we can not?

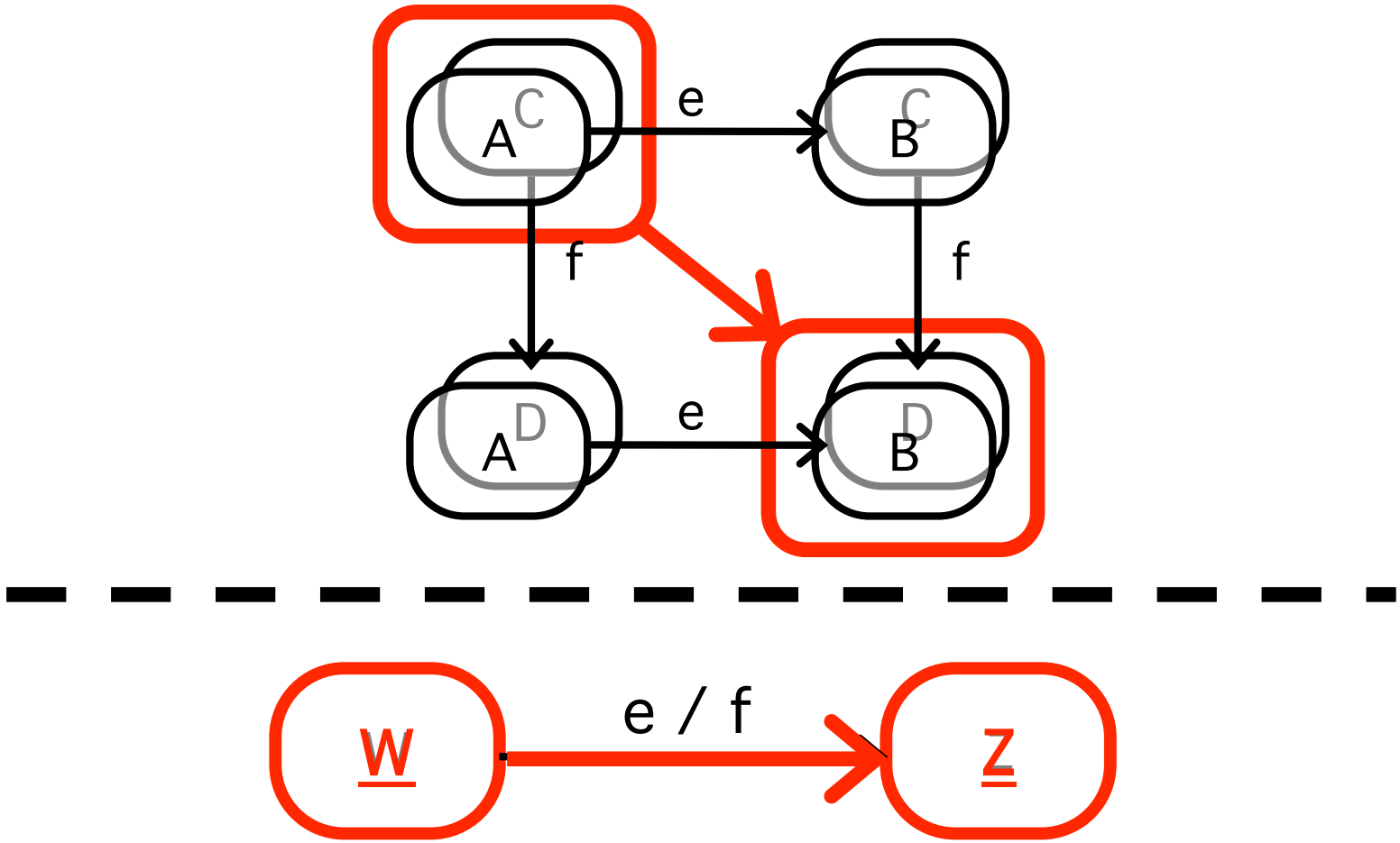
We have necessarily to add a third state machine acting as a bridge: we call it the “whole”…




Which is the meaning of states W and Z?



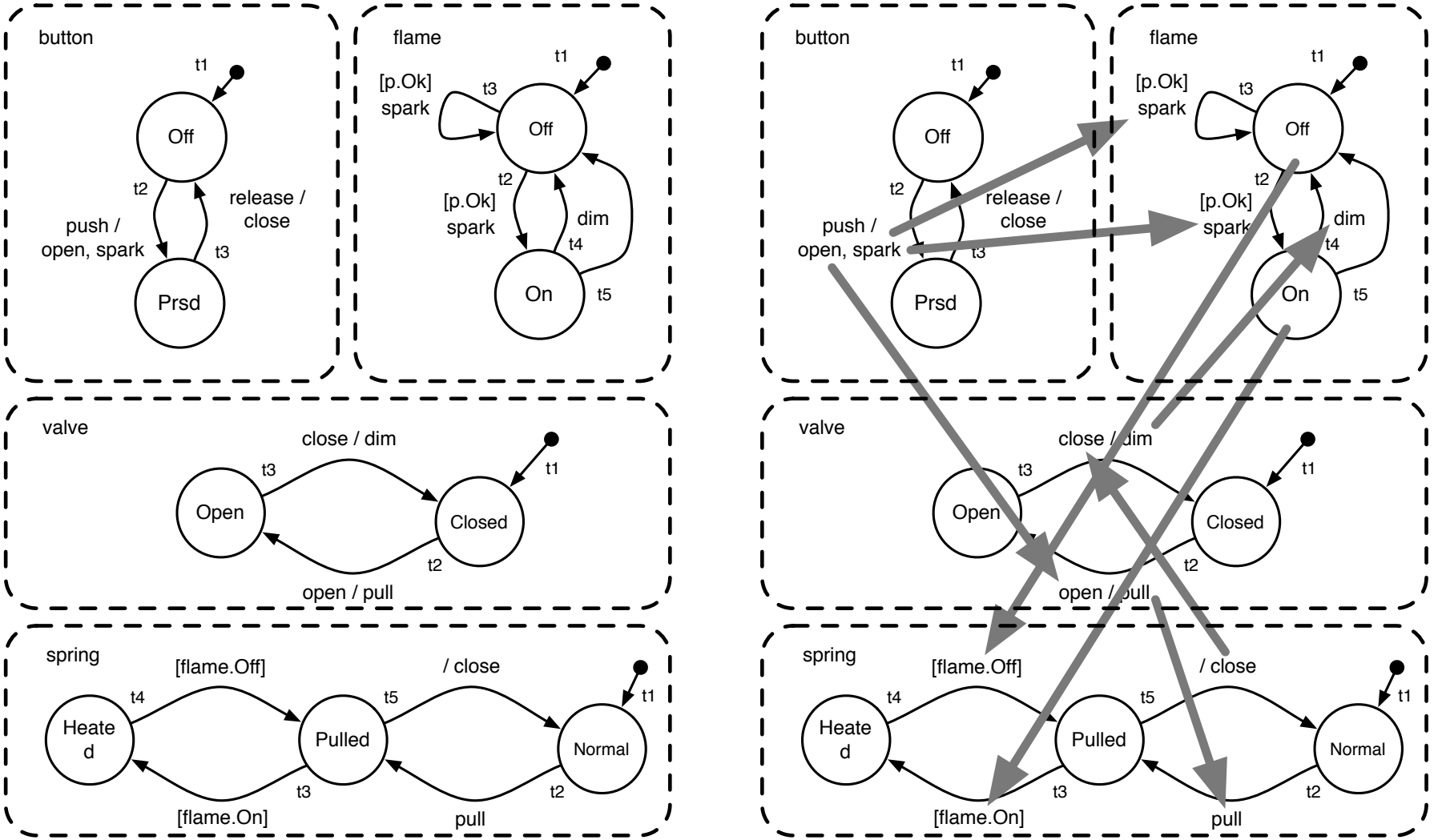

- "Whole" state machine introduced in order to reduce coupling among behavioral abstractions;
- It has however some interesting interpretations:
  - It embeds the semantics of composition which has been removed from the component behaviors, which are now self-contained;
  - Such a semantics is computable!
    - state W denotes the global state (A,C);
    - state Z denotes the the global state (B,D).

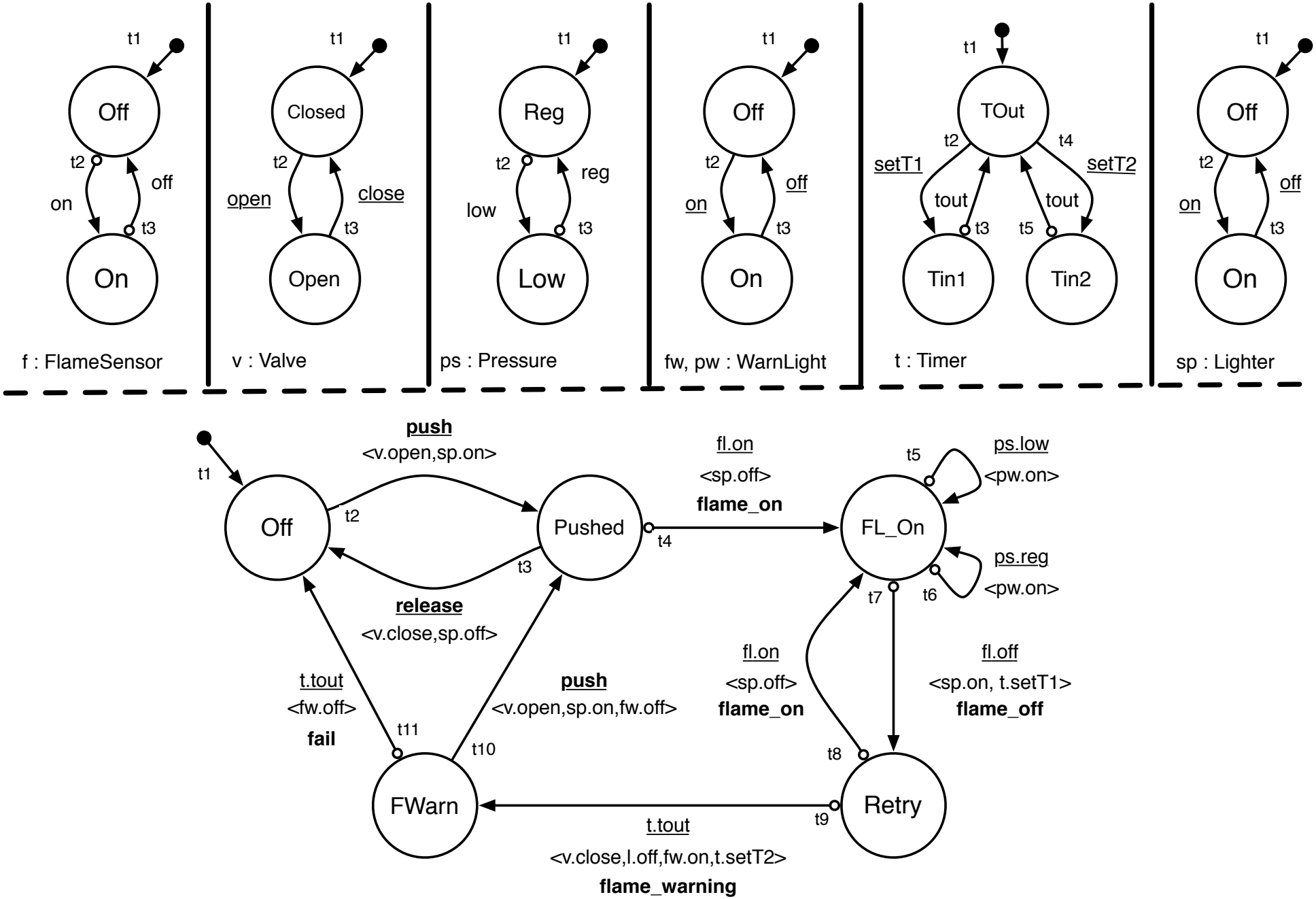

A
C
e
B
C
f
f
A
D
e
B
D
W
e / f
Z

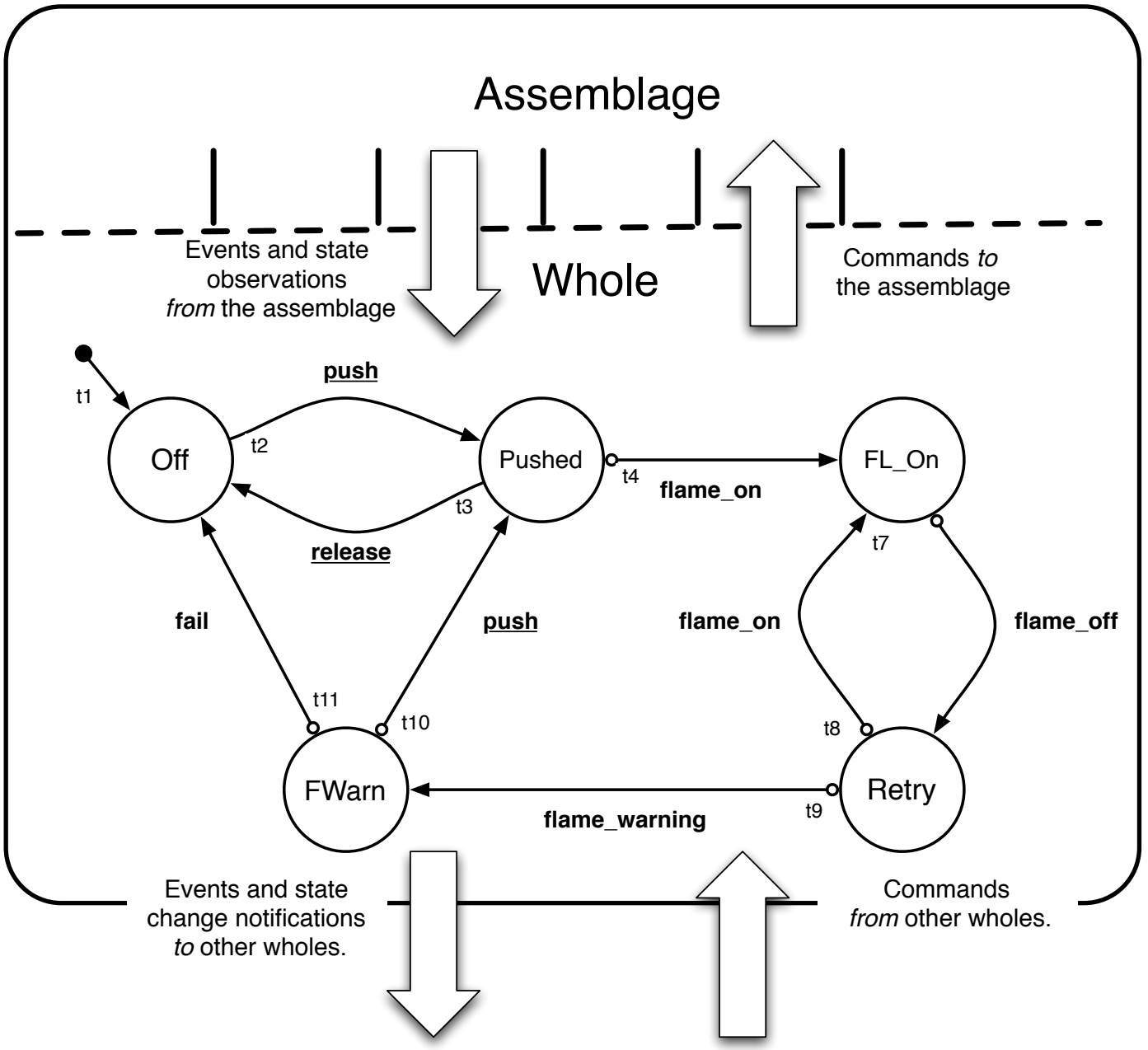
# Part-Whole Statecharts

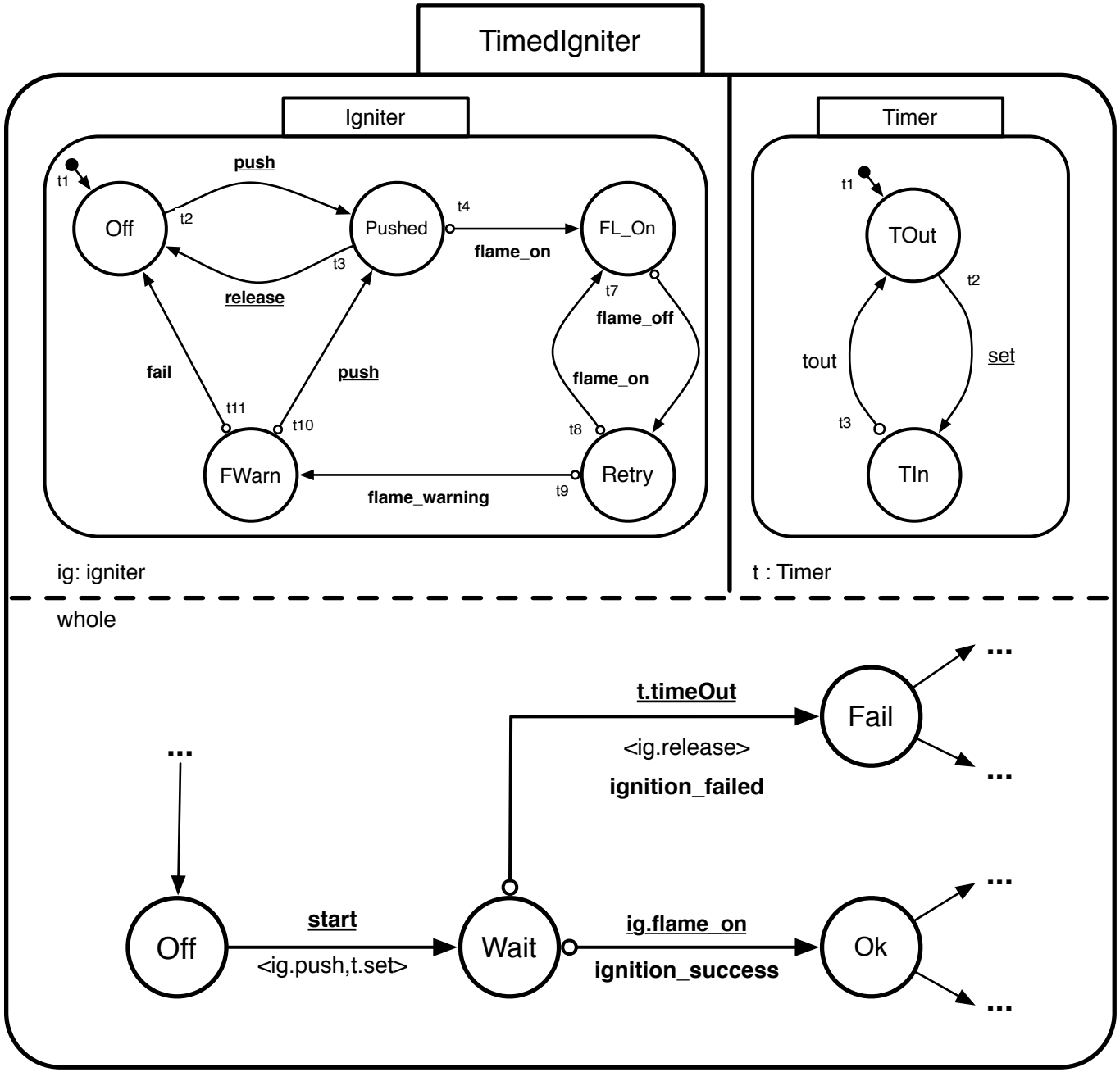
- Introduced with the aim of improving software quality of abstractions;
- Emergent behavior denoted explicitly by the "whole" state machine, which reduces coupling among behavioral abstractions; moreover
  - It works as an interface for the system of interacting entities;
  - It embeds the semantics of composition which has been removed from the component behaviors, which are now self-contained;
  - It has a semantics which is computable!

# Case study: multiple dynamical relationships within an ignition device




t1
Off
t2
on
off
t3
On
f : FlameSensor
Closed
open
close
Open
v : Valve
Reg
low
reg
Low
ps : Pressure
fw, pw : WarnLight
TOut
setT1
setT2
tout
t4
t5
Tin1
Tin2
t : Timer
sp : Lighter
push
<v.open,sp.on>
release
<v.close,sp.off>
Pushed
fl.on
<sp.off>
flame_on
FL_On
ps.low
<pw.on>
ps.reg
<pw.on>
t6
t7
fl.off
<sp.on, t.setT1>
flame_off
t.tout
<fw.off>
fail
t11
t10
push
<v.open,sp.on,fw.off>
t8
FWarn
Retry
t9
t.tout
<v.close,l.off,fw.on,t.setT2>
flame_warning


Assemblage
Events and state observations *from* the assemblage
Whole
Commands *to* the assemblage
t1
push
Off
t2
Pushed
t4
flame_on
FL_On
t3
release
t7
fail
push
flame_on
flame_off
t11
t10
t8
FWarn
Retry
t9
flame_warning
Events and state change notifications *to* other wholes.
Commands *from* other wholes.


TimedIgniter
Igniter
Timer
t1
Off
t2
push
Pushed
t4
FL_On
flame_on
t3
release
t7
flame_off
fail
push
flame_on
t11
t10
t8
FWarn
Retry
flame_warning
t9
t1
TOut
t2
tout
set
t3
TIn
ig: igniter
t : Timer
whole
...
t.timeOut
Fail
<ig.release>
ignition_failed
...
...
Off
start
<ig.push,t.set>
Wait
ig.flame_on
ignition_success
Ok
...
...

# Conclusions

- We surveyed Part-Whole Statecharts, an evolution of David Harel formalism created having in mind basic software quality principles;
- By an explicit composition state machine, called the “whole”, it allows arbitrary composition of behavioral abstractions;
- Such a formalism exhibits a computable semantics, which allows to check for safety and liveness without resorting to model checking techniques.

# Further Development & Research

- A method for checking safety & liveness properties at design time in a state-based design is patent pending under PCT/EP2008/051300;
- Incremental safety certification for dependable system should be feasible with the above methodology:
  - Composing certified components should lead directly to a new certified system;
  - Business model based on safety certification?

Thank you!