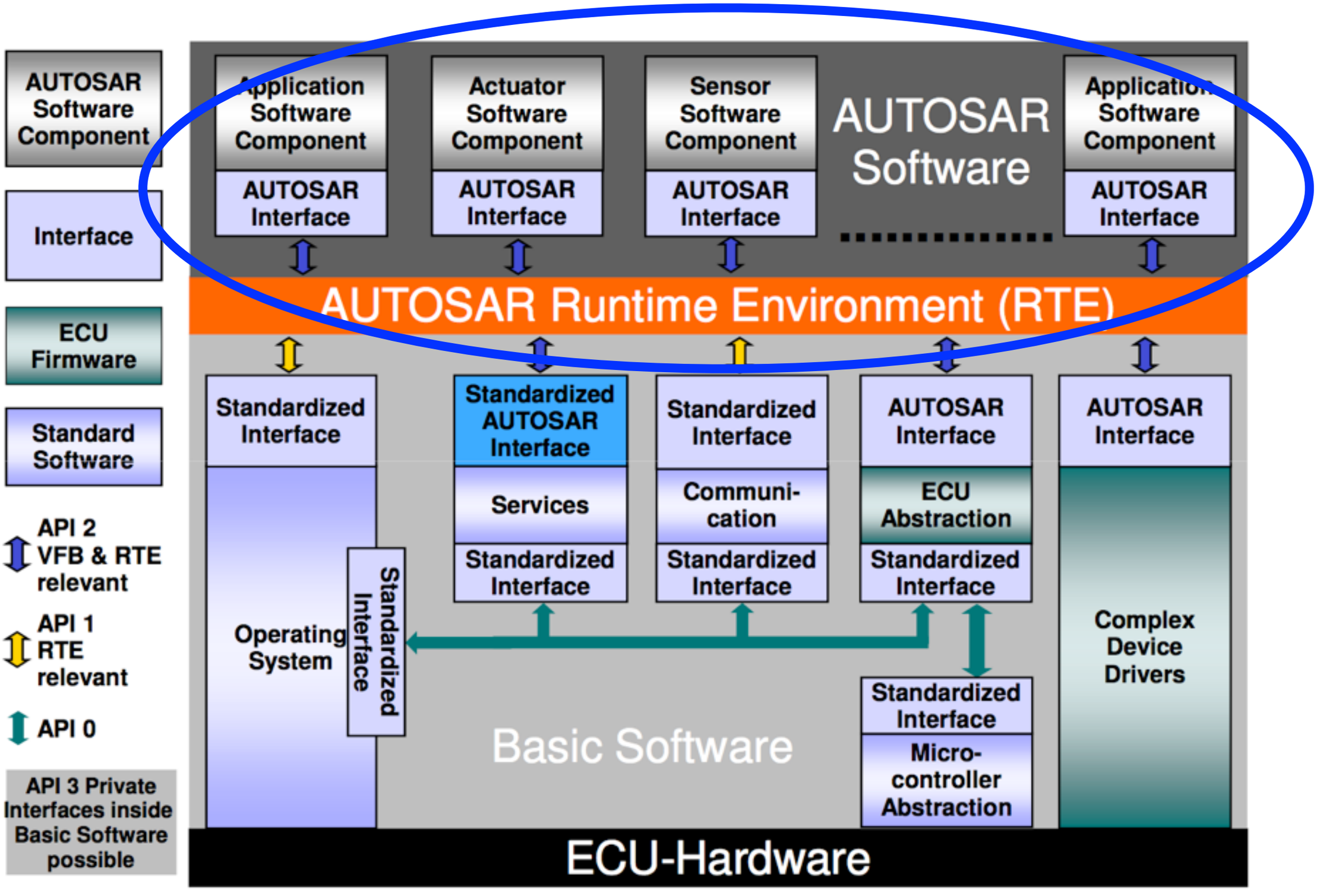
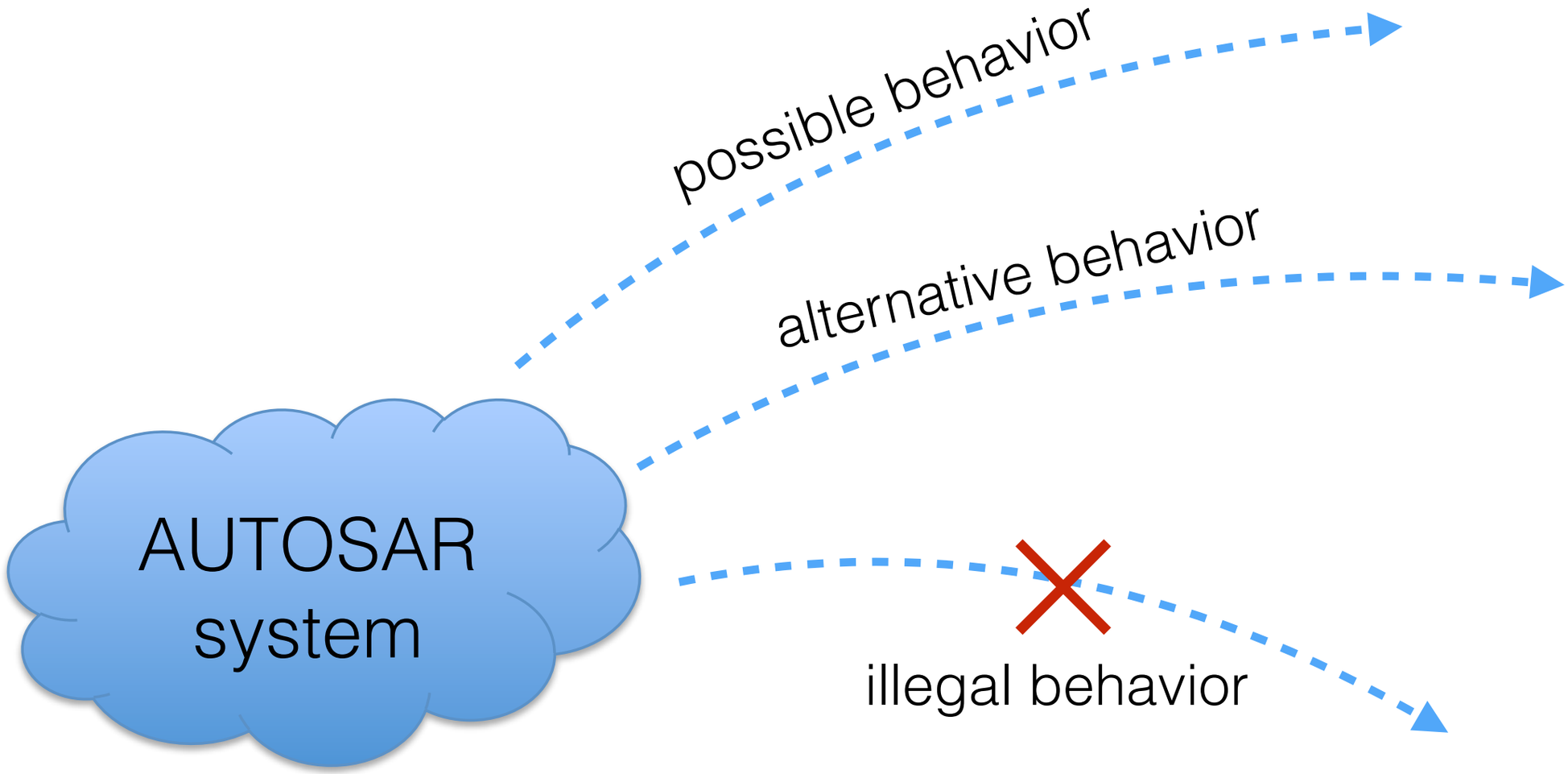
A formalization of AUTOSAR Software Components

Johan Nordlander
jointly with Patrik Jansson
Dec 4, 2014

AUTOSAR


AUTOSAR Software Component
Interface
ECU Firmware
Standard Software
API 2 VFB & RTE relevant
API 1 RTE relevant
API 0
API 3 Private Interfaces inside Basic Software possible
Application Software Component
Actuator Software Component
Sensor Software Component
AUTOSAR Software
Application Software Component
AUTOSAR Interface
AUTOSAR Runtime Environment (RTE)
Standardized Interface
Standardized AUTOSAR Interface
Services
Communi-cation
ECU Abstraction
Micro-controller Abstraction
Operating System
Complex Device Drivers
Basic Software
ECU-Hardware

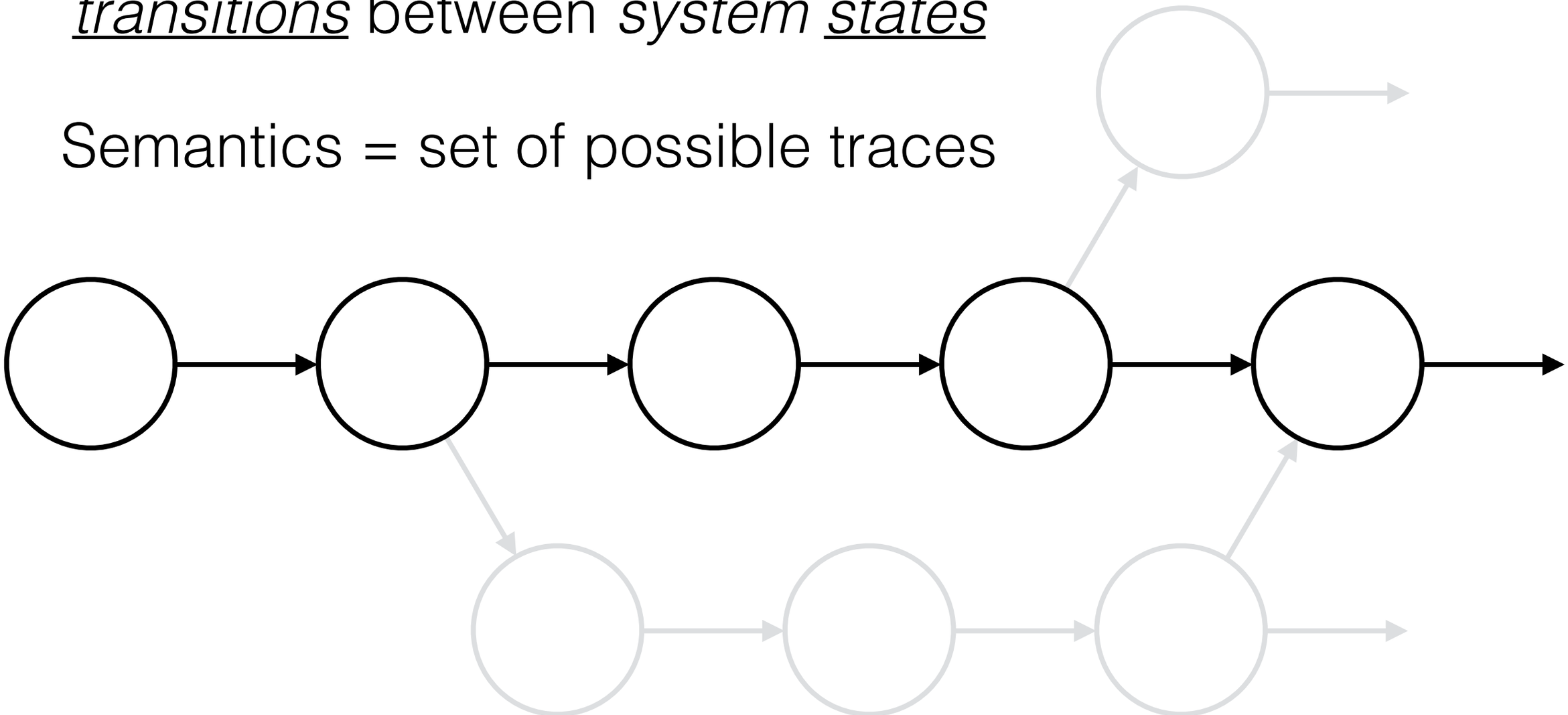
AUTOSAR at run-time



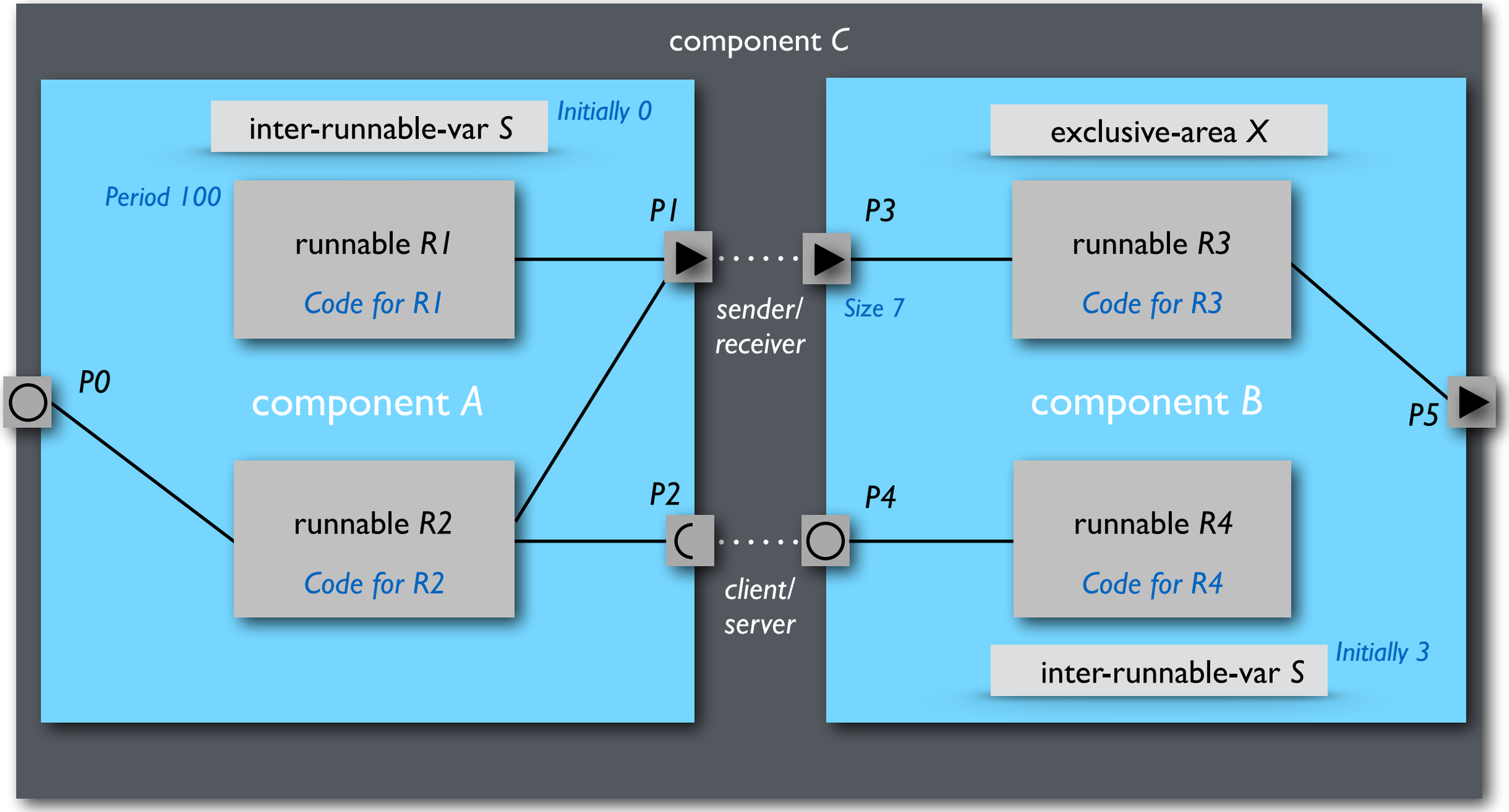
Behaviors

Behavior = trace = sequence of
transitions between *system states*

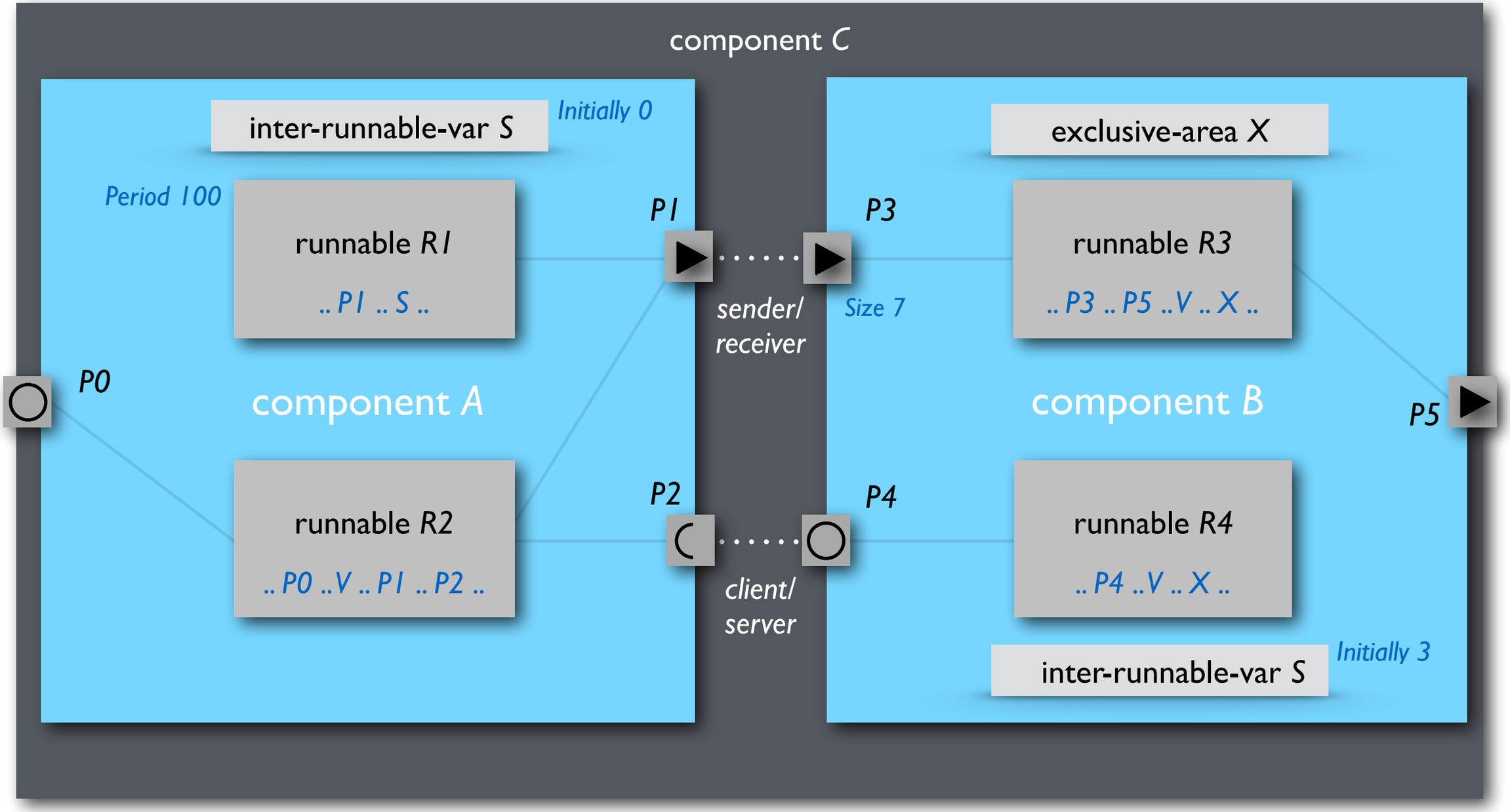
Semantics = set of possible traces



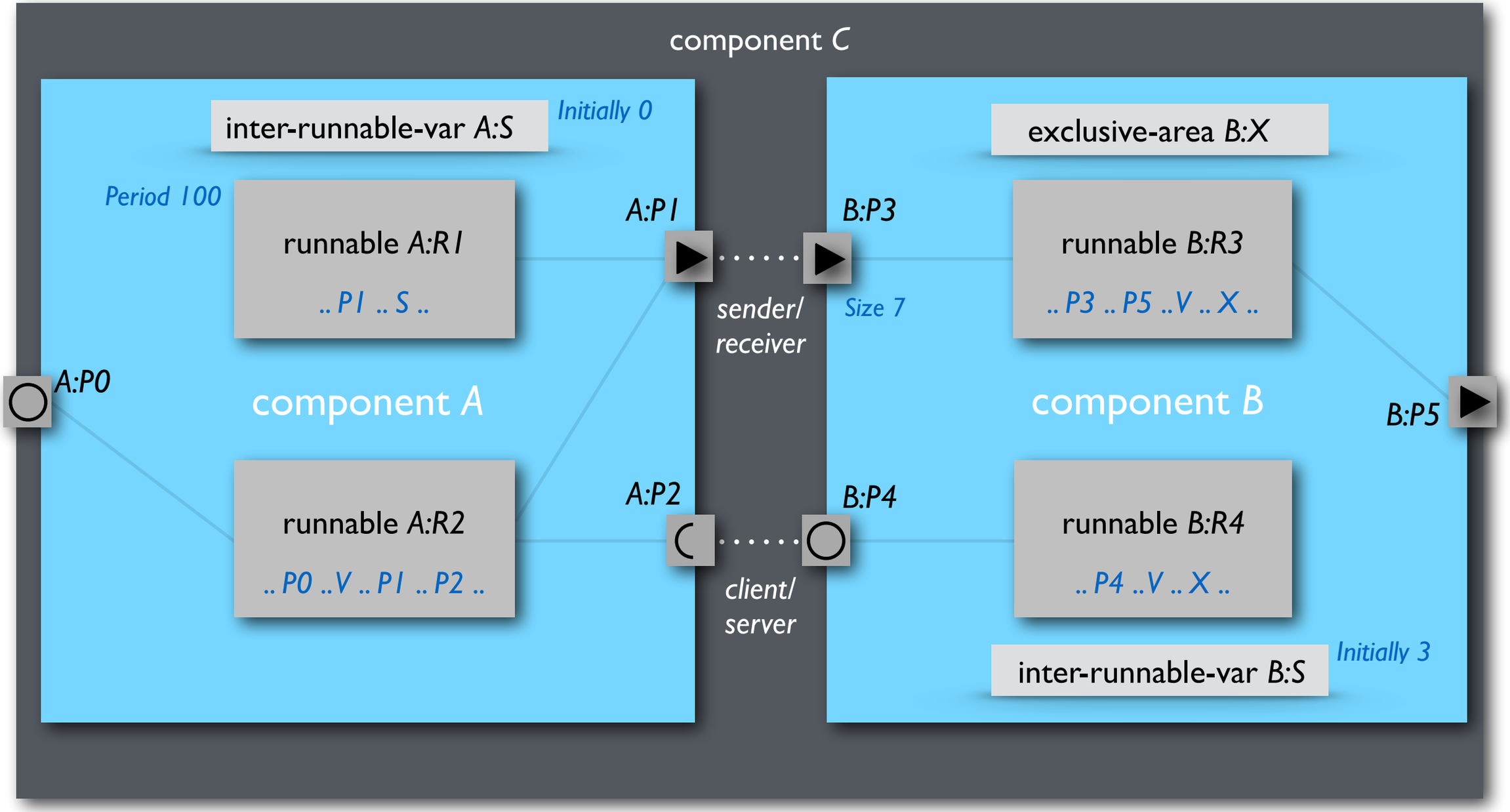
An AUTOSAR system



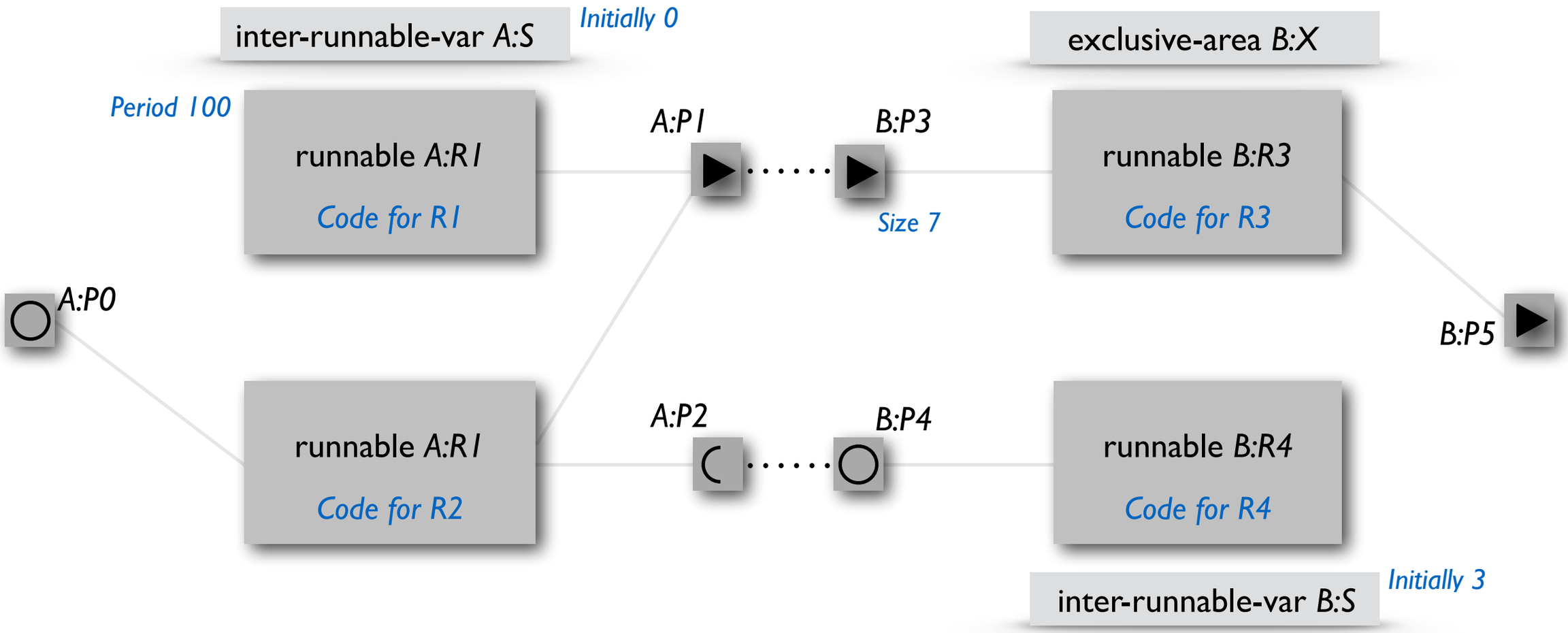
An AUTOSAR system



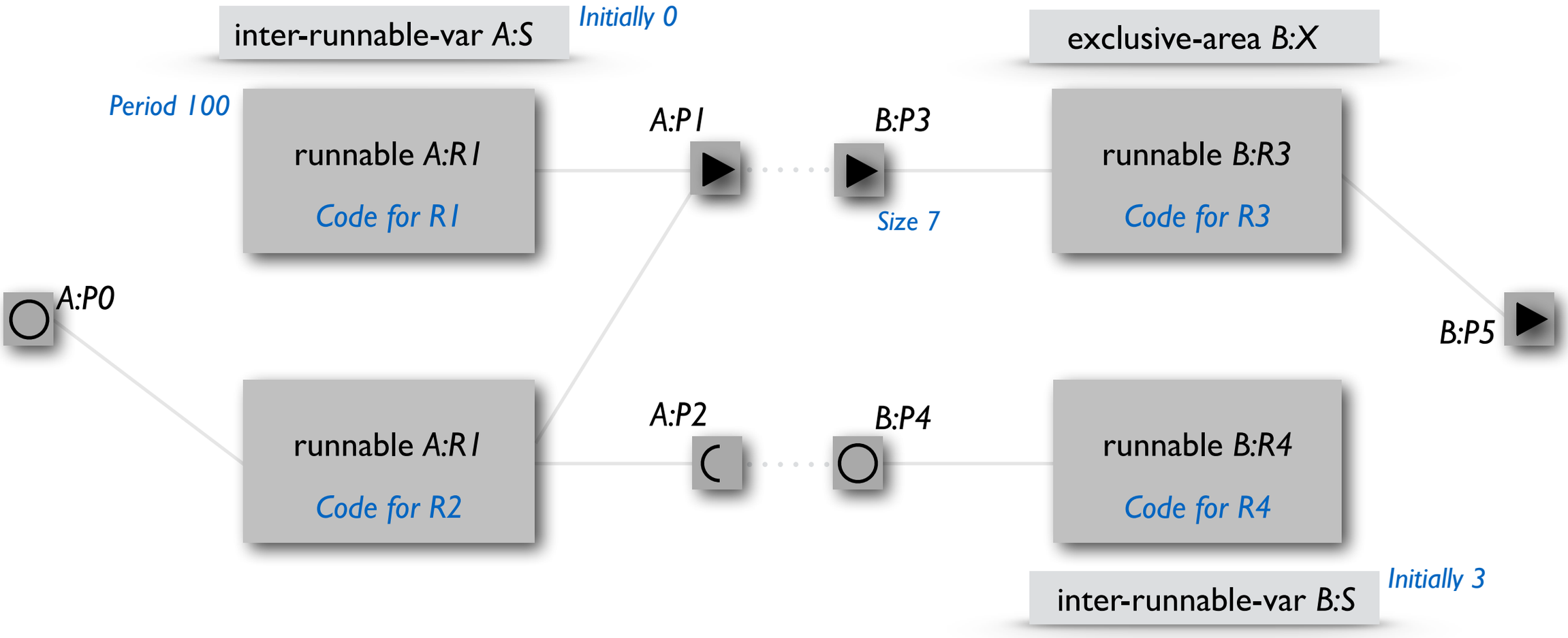
An AUTOSAR system



An AUTOSAR system



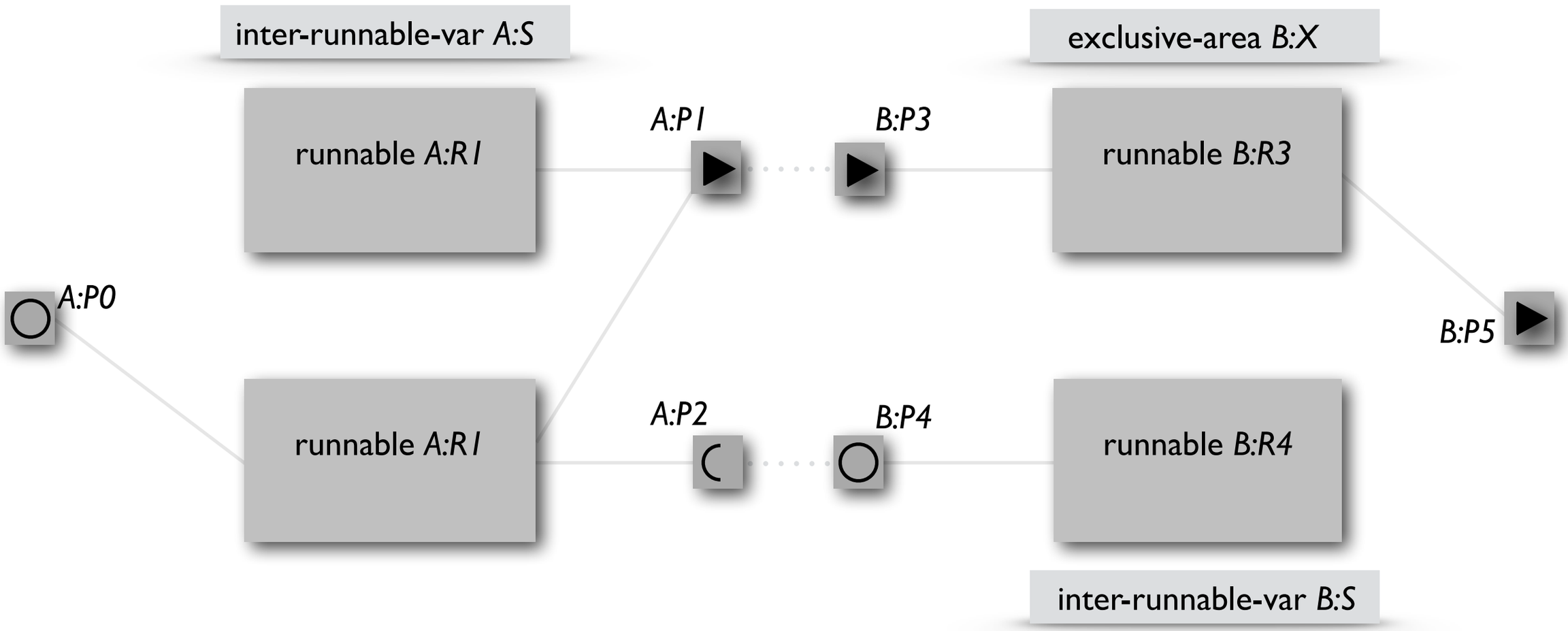
An AUTOSAR system



A:P1 ⇒ B:P3

A:P2 ⇒ B:P4

An AUTOSAR system



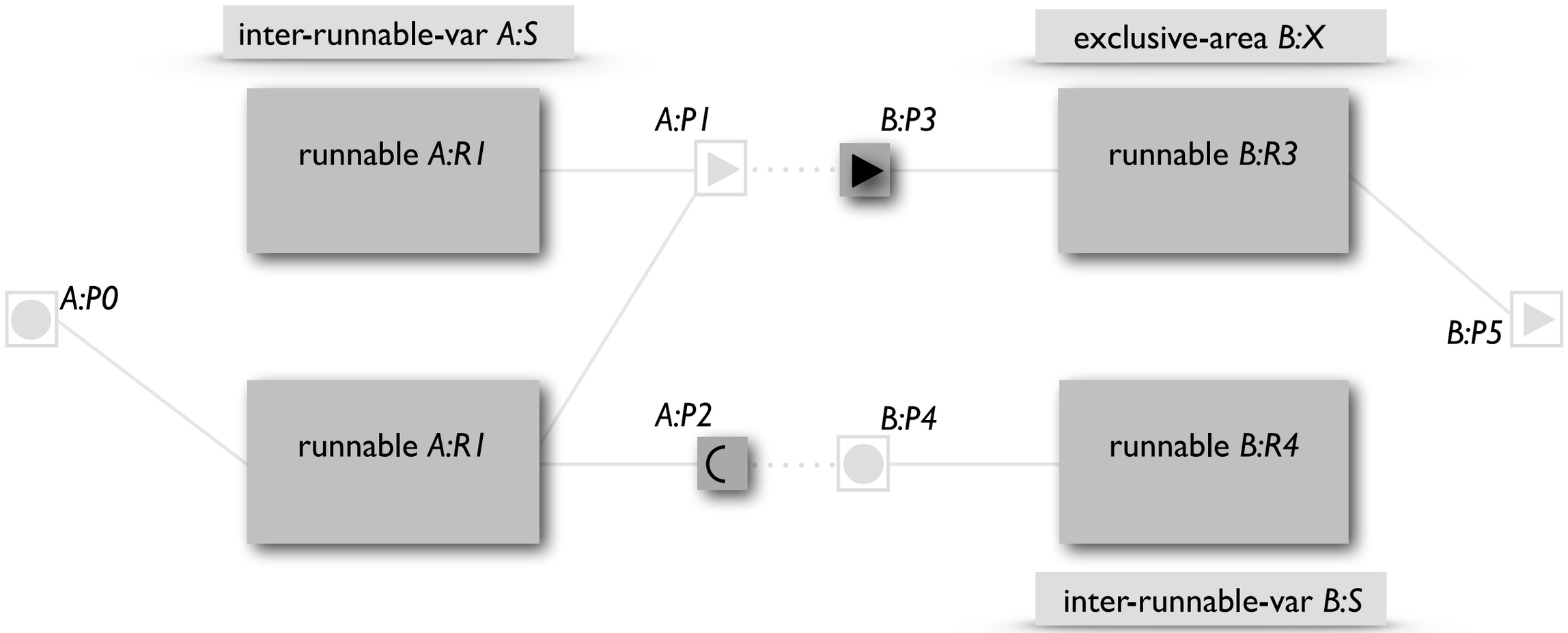
implementation(*A:R1*, *Code for R1*)
implementation(*A:R1*, *Code for R2*)
implementation(*A:R1*, *Code for R3*)
implementation(*A:R1*, *Code for R4*)

A:P1 ⇒ *B:P3*

A:P2 ⇒ *B:P4*

initial(*A:S*, 0)
period(*A:R1*, 100)
initial(*B:S*, 3)
size(*B:P3*, 7)

An AUTOSAR system



implementation(*A:R1*, *Code for R1*)
implementation(*A:R1*, *Code for R2*)
implementation(*A:R1*, *Code for R3*)
implementation(*A:R1*, *Code for R4*)

A:P1 ⇒ *B:P3*

A:P2 ⇒ *B:P4*

initial(*A:S*, 0)
period(*A:R1*, 100)
initial(*B:S*, 3)
size(*B:P3*, 7)

An AUTOSAR system

inter-runnable-var *A:S*

exclusive-area *B:X*

runnable *A:R1*

qelem *B:P3*

runnable *B:R3*

runnable *A:R1*

opres *A:P2*

runnable *B:R4*

inter-runnable-var *B:S*

implementation(*A:R1*, *Code for R1*)
implementation(*A:R1*, *Code for R2*)
implementation(*A:R1*, *Code for R3*)
implementation(*A:R1*, *Code for R4*)

A:P1 ⇒ *B:P3*

A:P2 ⇒ *B:P4*

initial(*A:S*, 0)
period(*A:R1*, 100)
initial(*B:S*, 3)
size(*B:P3*, 7)

System state

inter-runnable-var(*A:S*, *DYN*)

exclusive-area(*B:X*, *DYN*)

runnable(*A:R1*, *DYN*)

qelem(*B:P3*, *DYN*)

runnable(*B:R3*, *DYN*)

runnable(*A:R2*, *DYN*)

opres(*A:P2*, *DYN*)

runnable(*B:R4*, *DYN*)

inter-runnable-var(*B:S*, *DYN*)

---

implementation(*A:R1*, *Code for R1*)
implementation(*A:R1*, *Code for R2*)
implementation(*A:R1*, *Code for R3*)
implementation(*A:R1*, *Code for R4*)

A:P1 ⇒ *B:P3*

A:P2 ⇒ *B:P4*

initial(*A:S*, 0)
period(*A:R1*, 100)
initial(*B:S*, 3)
size(*B:P3*, 7)

System state

inter-runnable-var(*A:S*, *DYN*)

runnable(*A:R1*, *DYN*)

rinst(*i:a*, *DYN*)

rinst(*i:a*, *DYN*)

…

runnable(*A:R2*, *DYN*)

opres(*A:P2*, *DYN*)

qelem(*B:P3*, *DYN*)

exclusive-area(*B:X*, *DYN*)

runnable(*B:R3*, *DYN*)

rinst(*j:c*, *DYN*)

…

runnable(*B:R4*, *DYN*)

inter-runnable-var(*B:S*, *DYN*)

---

implementation(*A:R1*, *Code for R1*)
implementation(*A:R1*, *Code for R2*)
implementation(*A:R1*, *Code for R3*)
implementation(*A:R1*, *Code for R4*)

A:P1 ⇒ *B:P3*

A:P2 ⇒ *B:P4*

initial(*A:S*, 0)
period(*A:R1*, 100)
initial(*B:S*, 3)
size(*B:P3*, 7)

System state

atomic processes

parallel composition

inter-runnable-var(*A:S*, *DYN*)

exclusive-area(*B:X*, *DYN*)

runnable(*A:R1*, *DYN*)

rinst(*i:a*, *DYN*)

rinst(*i:a*, *DYN*)

…

qelem(*B:P3*, *DYN*)

runnable(*B:R3*, *DYN*)

rinst(*j:c*, *DYN*)

…

runnable(*A:R2*, *DYN*)

opres(*A:P2*, *DYN*)

runnable(*B:R4*, *DYN*)

inter-runnable-var(*B:S*, *DYN*)

---

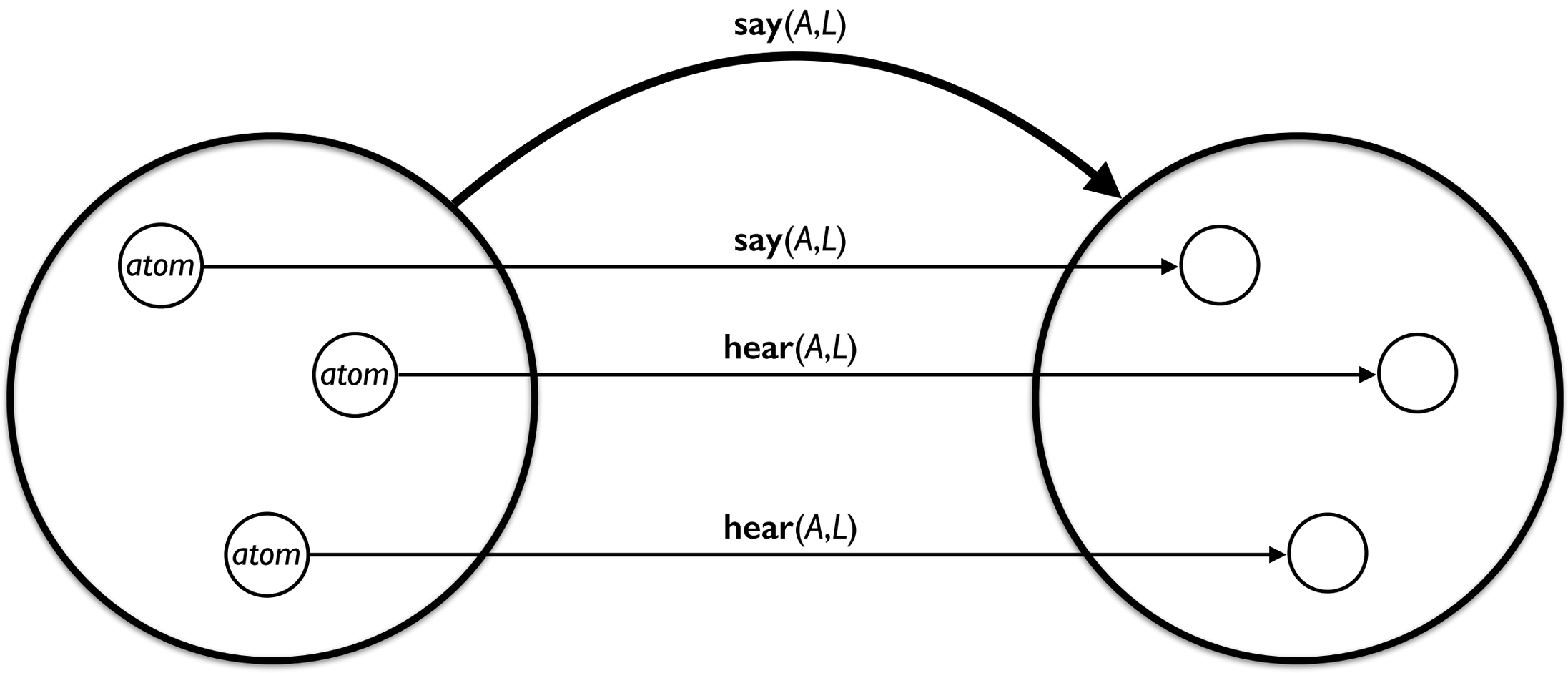
implementation(*A:R1*, *Code for R1*)
implementation(*A:R1*, *Code for R2*)
implementation(*A:R1*, *Code for R3*)
implementation(*A:R1*, *Code for R4*)

A:P1 ⇒ *B:P3*

A:P2 ⇒ *B:P4*

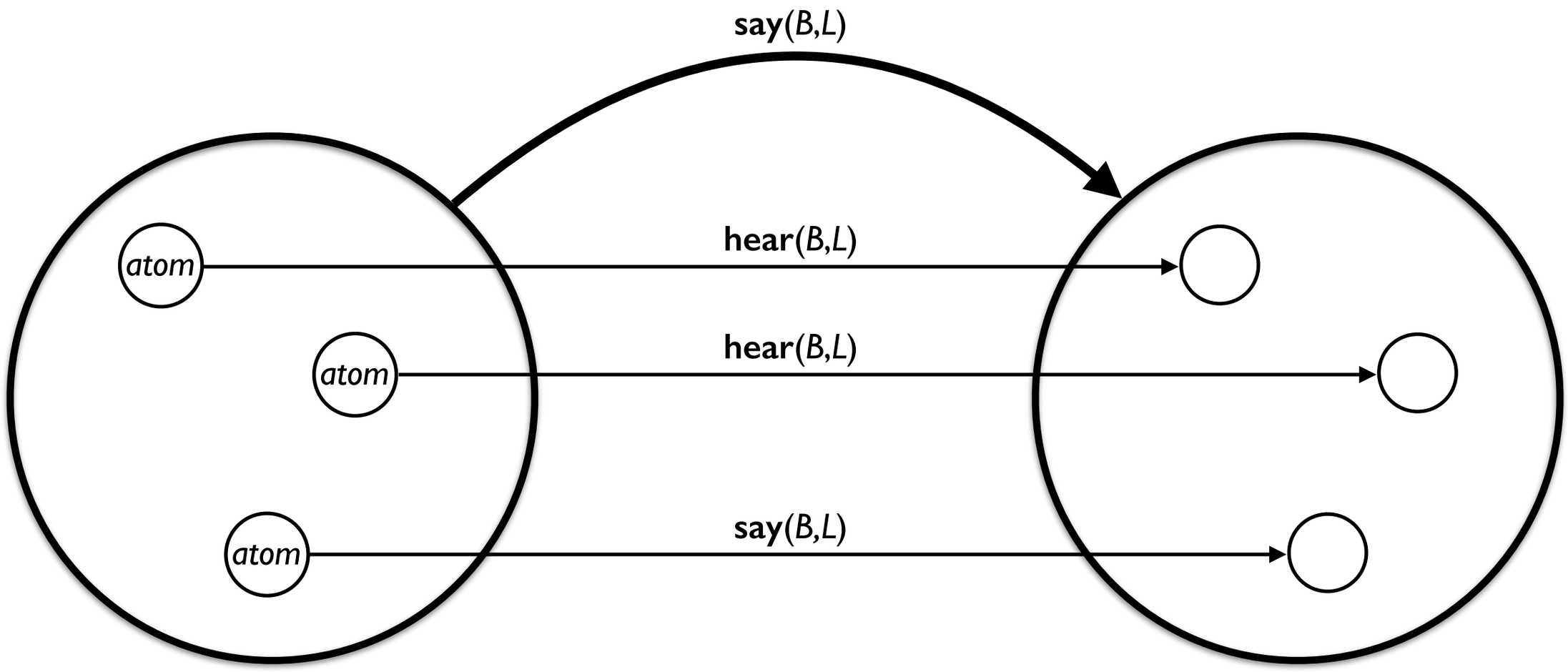
initial(*A:S*, 0)
period(*A:R1*, 100)
initial(*B:S*, 3)
size(*B:P3*, 7)

Labelled transitions



broadcast

Labelled transitions



non-determinism

Some simple transitions

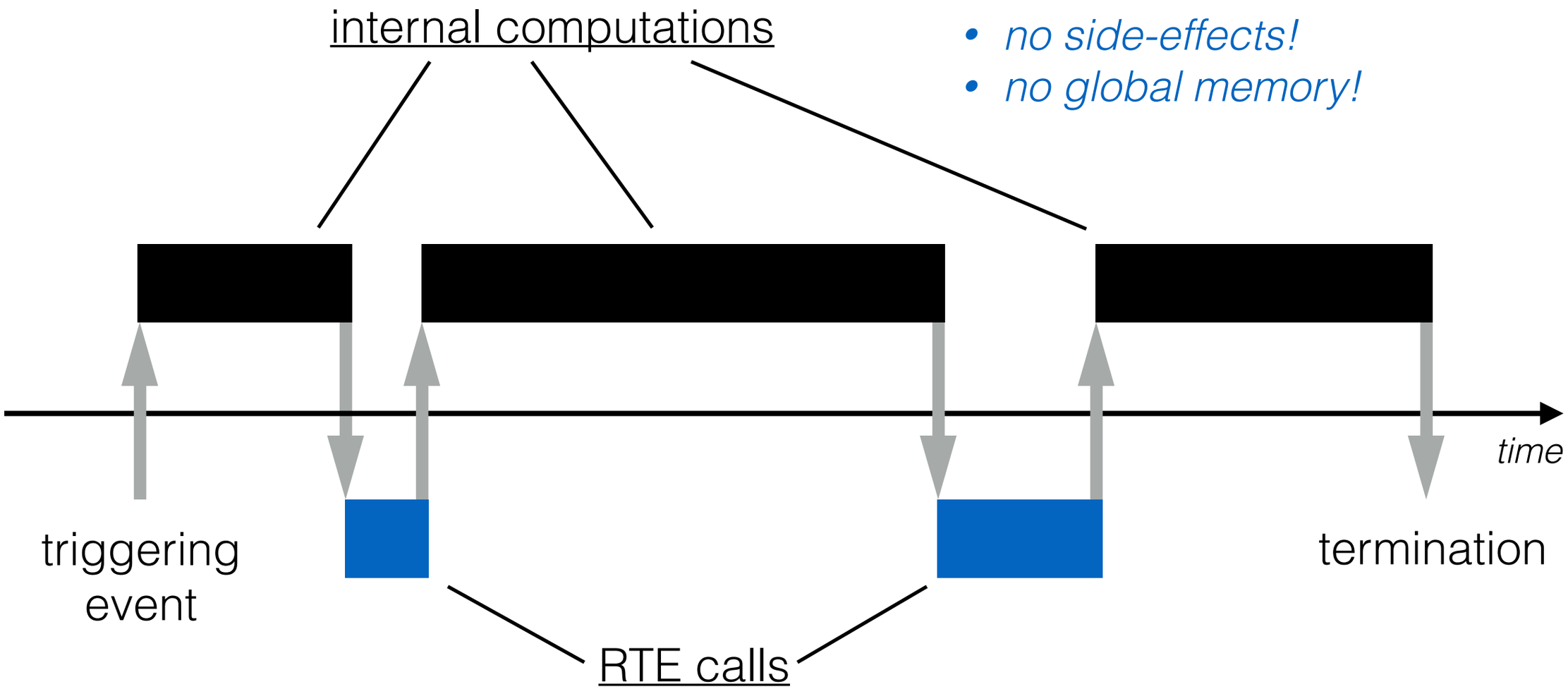
exclusive-area(*l:X*, free) —**hear**(*l:X*, enter)→ exclusive-area(*l:X*, taken)

exclusive-area(*l:X*, taken) —**hear**(*l:X*, exit)→ exclusive-area(*l:X*, free)

inter-runnable-var(*l:S*, *V*) —**hear**(*l:S*, irv-read(*V*))→ inter-runnable-var(*l:S*, *V*)

inter-runnable-var(*l:S*, _) —**hear**(*l:S*, irv-write(V))→ inter-runnable-var(*l:S*, *V*)

The timeline of a runnable instance



- *no side-effects!*
- *no global memory!*

- *sequential*
- *observable*

The Run-Time Environment

rte_send(*P*, *V*)	*asynchronous send*
rte_receive(*P*)	*poll receiver port*
rte_call(*P*, *V*)	*synchronous call*
rte_irv_write(*S*, *V*)	*write shared state*
rte_irv_read(*S*)	*read shared state*
rte_enter(*X*)	*acquire a lock*
rte_exit(*X*)	*release a lock*

+ a few more

The Run-Time Environment

rte_send(*P*, *V*, *Cont*)	*asynchronous send*
rte_receive(*P*, *Cont*)	*poll receiver port*
rte_call(*P*, *V*, *Cont*)	*synchronous call*
rte_irv_write(*S*, *V*, *Cont*)	*write shared state*
rte_irv_read(*S*, *Cont*)	*read shared state*
rte_enter(*X*, *Cont*)	*acquire a lock*
rte_exit(*X*, *Cont*)	*release a lock*
…	
return(*V*)	*terminate*

Compute next RTE call: *Cont*(*V*)

More simple transitions

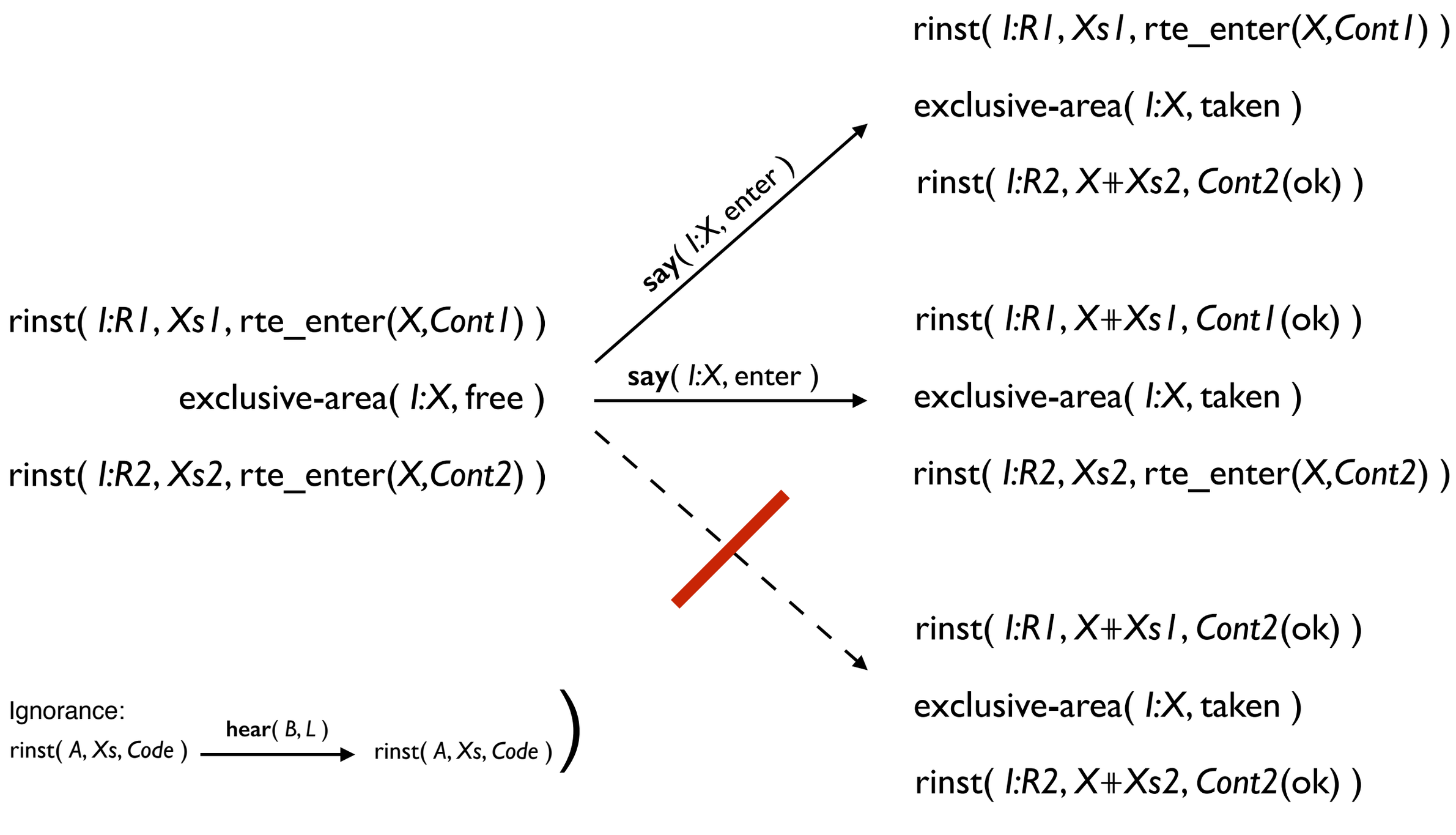
rinst(*I:R*, *Xs*, rte_enter(*X*,*Cont*)) —**say**(*I:X*, enter)→ rinst(*I:R*, *X*+*Xs*, *Cont*(ok))

rinst(*I:R*, *X*+*Xs*, rte_exit(*X*,*Cont*)) —**say**(*I:X*, exit)→ rinst(*I:R*, *Xs*, *Cont*(ok))

(exclusive-area(*I:X*, free) —**hear**(*I:X*, enter)→ exclusive-area(*I:X*, taken))

rinst(*I:R*, *Xs*, rte_enter(*X*,*Cont*))
exclusive-area(*I:X*, free)
—**say**(*I:X*, enter)→
rinst(*I:R*, *X*+*Xs*, *Cont*(ok))
exclusive-area(*I:X*, taken)

More simple transitions

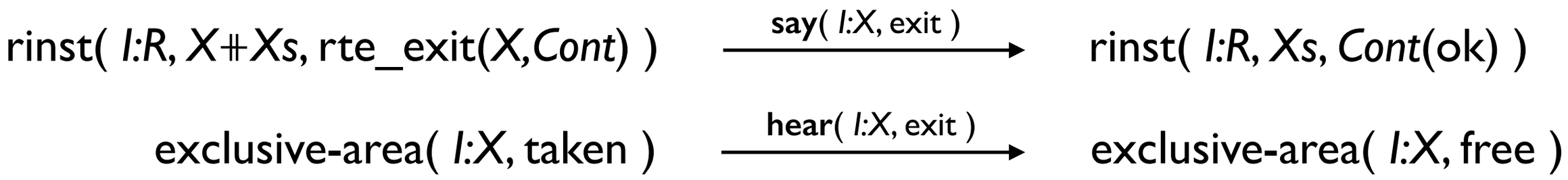


(Ignorance: rinst(A, Xs, Code) —hear(B, L)→ rinst(A, Xs, Code))

Ambiguities

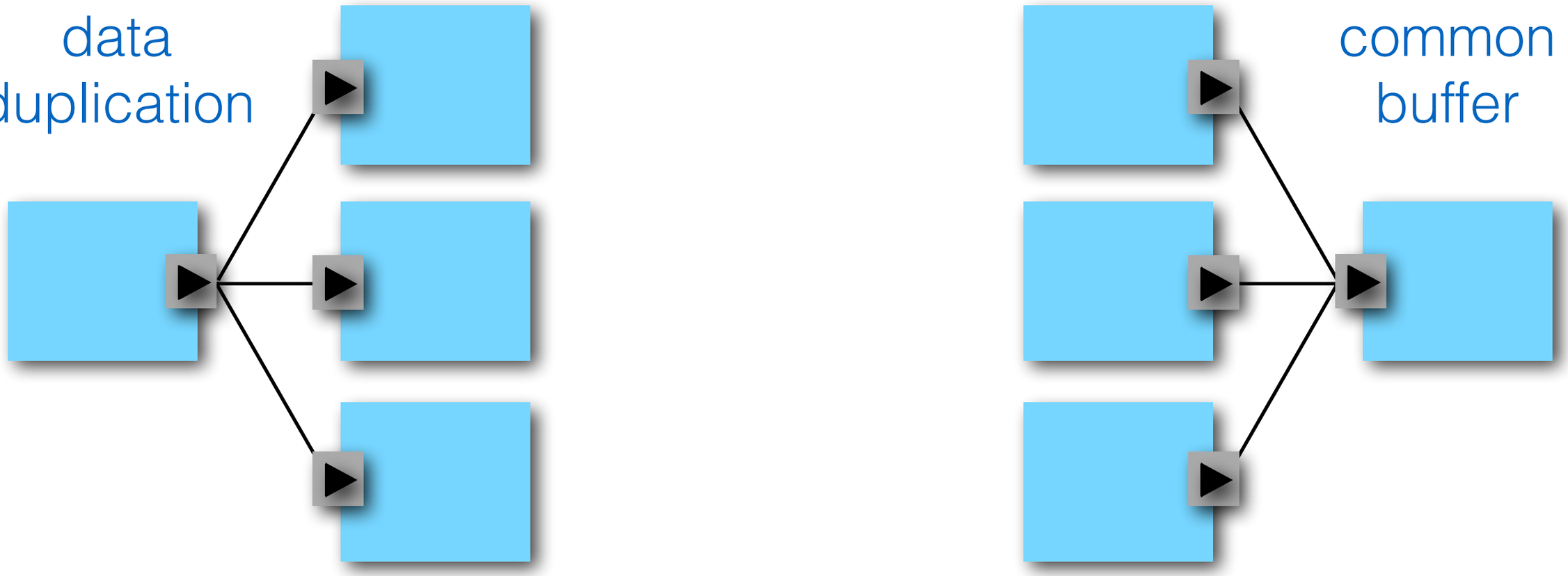
”The RTE is <u>not required</u> to support nested invocations of Rte_Exit for the same exclusive area.” [Is it allowed?]

”Requirement [SWS_Rte_01122] <u>permits</u> calls to Rte_Enter and Rte_Exit to be nested as long as different exclusive areas are exited in the reverse order they were entered.” [What if they don’t?]



[Interestingly, deadlock isn’t mentioned in the spec.]

1:N or N:1



[Interestingly, N:M explicitly not allowed by spec.]

Sending & receiving

rinst(*I:R*, *Xs*, rte_receive(*P*,*Cont*)) —**say**(*I:P*, rcv(*V*))→ rinst(*I:R*, *Xs*, *Cont*(*V*))

qelem(*I:P*, *N*, *V*+*Vs*) —**hear**(*I:P*, rcv(*V*))→ qelem(*I:P*, *N*, *Vs*)

qelem(*I:P*, *N*, []) —**hear**(*I:P*, rcv(no_data))→ qelem(*I:P*, *N*, [])

rinst(*I:R*, *Xs*, rte_send(*P*,*V*,*Cont*)) —**say**(*I:P*, snd(*V*,Res))→ rinst(*I:R*, *Xs*, *Cont*(Res))

if *I:P*⇒*A*, |*Vs*|<*N* : qelem(*A*, *N*, *Vs*) —**hear**(*I:P*, snd(*V*,ok))→ qelem(*A*, *N*, *Vs*+*V*)

if *I:P*⇒*A*, |*Vs*|=*N* : qelem(*A*, *N*, *Vs*) —**hear**(*I:P*, snd(*V*,limit))→ qelem(*A*, *N*, *Vs*)

if *I:P*⇒*A*, |*Vs*|<*N* : qelem(*A*, *N*, *Vs*) —**hear**(*I:P*, snd(*V*,limit))→ qelem(*A*, *N*, *Vs*+*V*)

Spawning instances

if *A*⇒*I:P*, events(*I:R*, dataReceived(*P*)) :

runnable(*I:R*, *T*, _, *N*) —**hear**(*A*, snd(_,_))→ runnable(*I:R*, *T*, pending, *N*)

one bit of info

if *N*=0 | canBeInvokedConcurrently(*I:R*) :

runnable(*I:R*, 0, pending, *N*) —**say**(*I:R*, new)→ runnable(*I:R*, *T*, idle, *N*+1)

rinst(*I:R*, [], *Code*)

if minimumStartInterval(*I:R*,*T*),
implementation(*I:R*, *Code*)

A semantic pitfall

runnable(*I:R*, 0, idle, 0)
qelem(*I:P*, *N*, [])

hear(*A*, snd(1,ok))

runnable(*I:R*, 0, pending, 0)
qelem(*I:P*, *N*, [1])

say(*I:R*, new)

runnable(*I:R*, 0, idle, 1)
rinst(*I:R*, [], *Code*)
qelem(*I:P*, *N*, [1])

hear(*A*, snd(2,ok))

runnable(*I:R*, 0, pending, 1)
rinst(*I:R*, [], *Code*)
qelem(*I:P*, *N*, [1,2])

say(*I:R*, new)

runnable(*I:R*, 0, idle, 2)
rinst(*I:R*, [], *Code*)
rinst(*I:R*, [], *Code*)
qelem(*I:P*, *N*, [1,2])

2 elements,
2 instances

A semantic pitfall

runnable(*I:R*, 0, idle, 0)
qelem(*I:P*, *N*, [])

hear(*A*, snd(1,ok))

runnable(*I:R*, 0, pending, 0)
qelem(*I:P*, *N*, [1])

hear(*A*, snd(2,ok))

runnable(*I:R*, 0, pending, 0)
rinst(*I:R*, [], *Code*)
qelem(*I:P*, *N*, [1,2])

say(*I:R*, new)

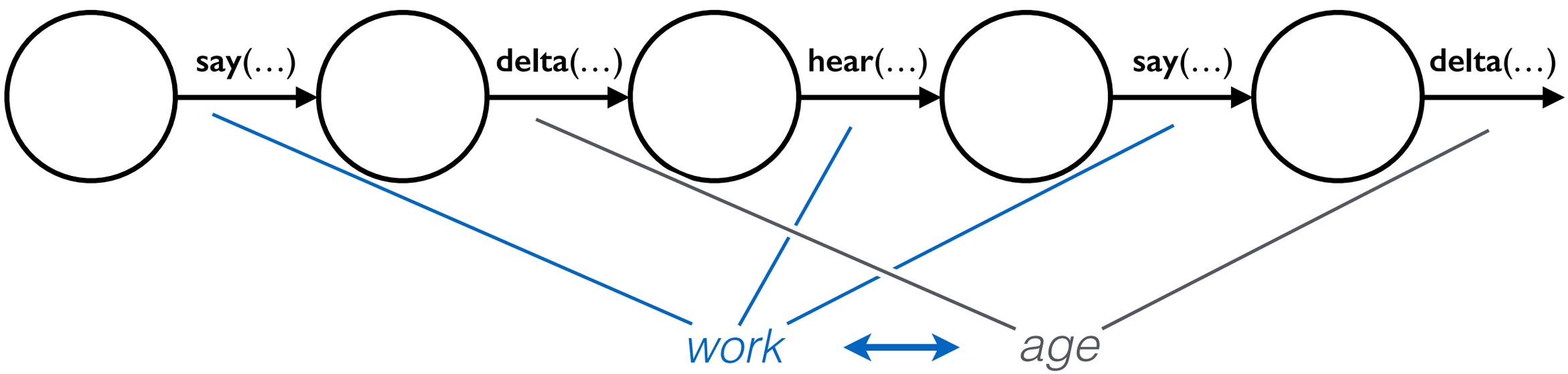
runnable(*I:R*, 0, idle, 1)
rinst(*I:R*, [], *Code*)
qelem(*I:P*, *N*, [1,2])

2 elements,
only 1 instance!

Passing time

if $V \leq T$:

runnable(*I:R*, *T*, *Act*, *N*) —**delta**(*V*)→ runnable(*I:R*, *T-V*, *Act*, *N*)



relationship not restricted (arbitrarily fast platform)

Prolog formulation

Code

```
rinst(I:R, Xs, rte_receive(P,Cont)) ---say(I:P,rcv(V))---> rinst(I:R, Xs, Cont(V))
                                                          :- eval(ap(Cont,V),Code).
```

Negation and arithmetics… careful ordering of predicates!

Good for exhaustive searches of single (few) transitions

A good format for communicating semantic detail?

Wrap up

- Formalized AUTOSAR Software Components & RTE
- Parallel atomic processes, broadcast, work/time steps
- Abstract code, work = RTE operations
- Right framework for capturing/discussing tricky details
- Not yet: modes, category>1 runnables, COM service, …
- Prolog def for reference, *simulator in Haskell for efficiency*