Clutching a Grip on AUTOSAR using Haskell

Johan Nordlander
Chalmers University of Technology

BOB 2015

Tool-neutral

Platform-neutral

Vendor-neutral

Component architecture

Automotive domain

Development methodology

Industry standard

AUTOSAR

Distribution

Real-time

OS kernel

I/O abstraction

Concurrency

Standard library

Communication

Black box interoperability

Standardized interfaces

The AUTOSAR spec.

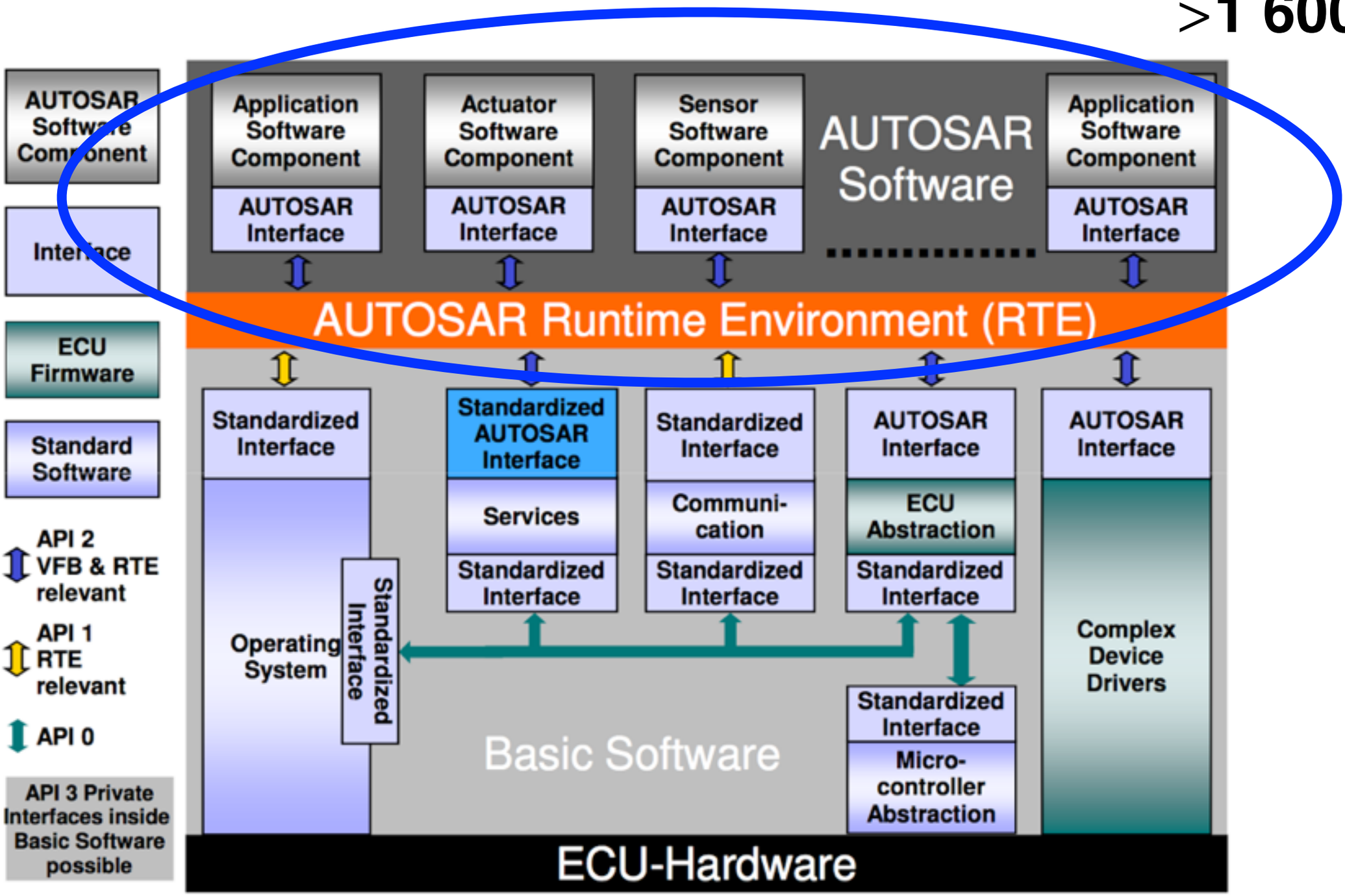
Informal text / UML diagrams / C headers

Mixed with (assumed) implementation details

>100 documents! **>12 500 pages!**

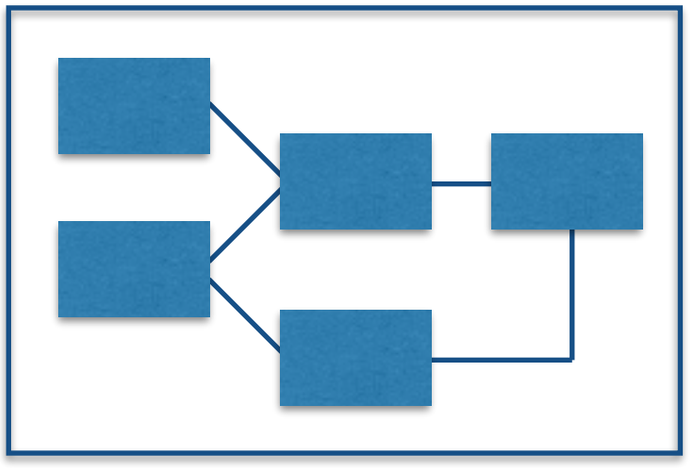
Software Components

>**1 600** pages



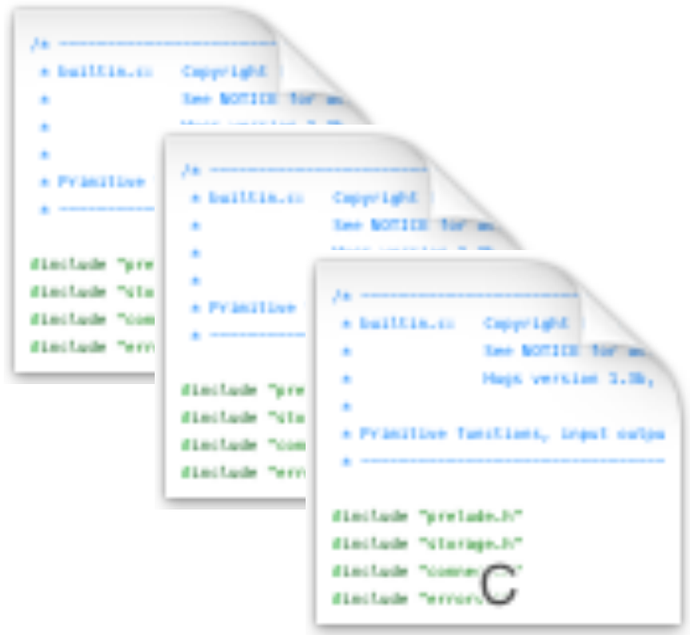
AUTOSAR development

AUTOSAR Model



Manual steps

Implementation



- Structure & constraints
- Platform independent
- Lacks code
- Not executable

?

- C files & config tables
- Platform dependent
- Only code
- Executable

Consequences

Can't test an AUTOSAR model

- until after all implementation steps
    - unless all subsystems are present
        - without committing to a particular tool/platform

Can't simulate a model "in the abstract"

Can't really talk about **black box** AUTOSAR behaviour

RAWFP @ Chalmers

Resource-aware functional programming

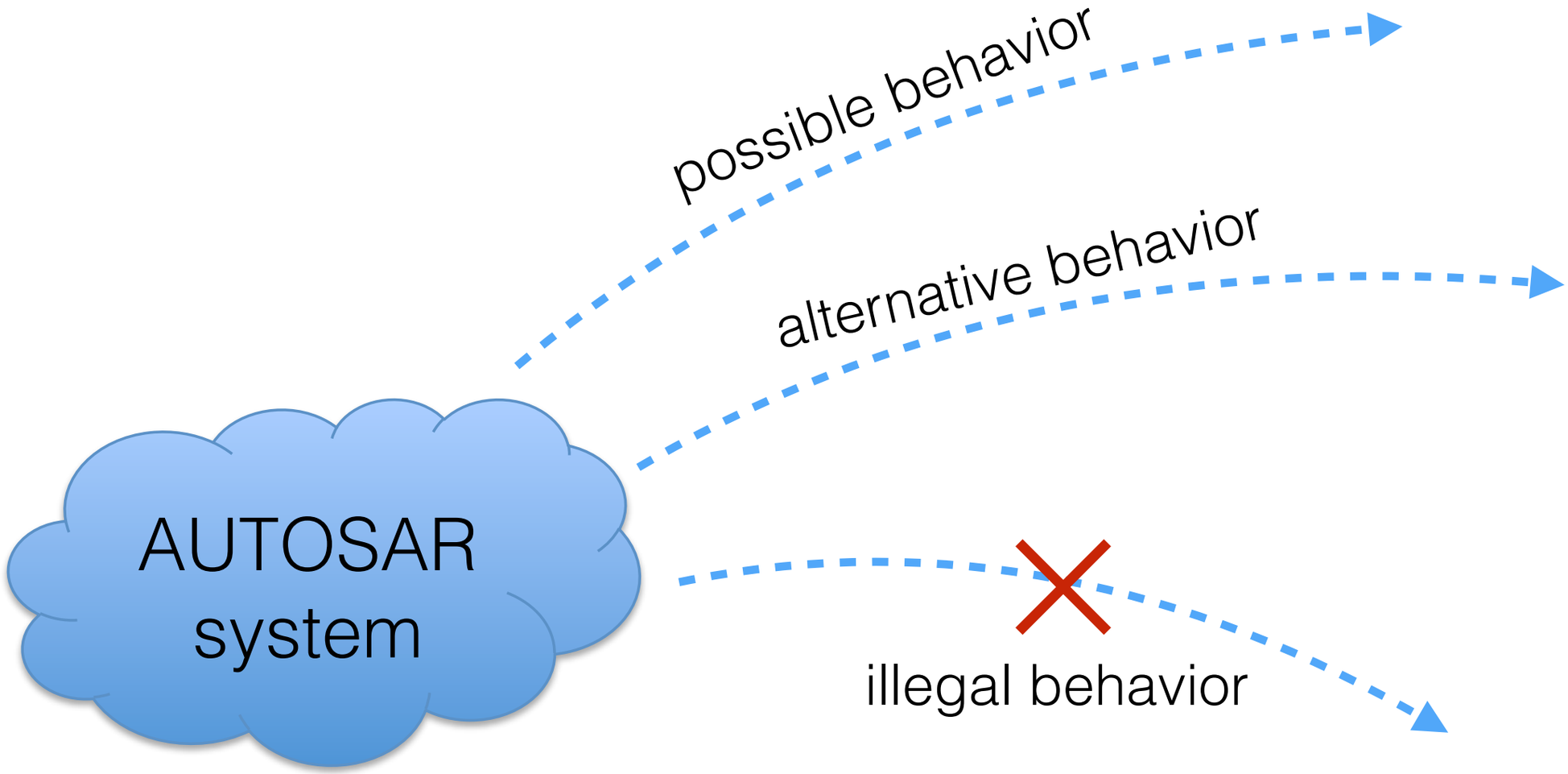
(Exploring Domain-Specific Languages in Haskell)

Theme:
semantics-based analysis, testing & verification in Haskell;
efficient execution after compilation to preferred target code

Validator track 1:
AUTOSAR Software Components as a Haskell DSL

(structure + constraints + <u>code</u>)

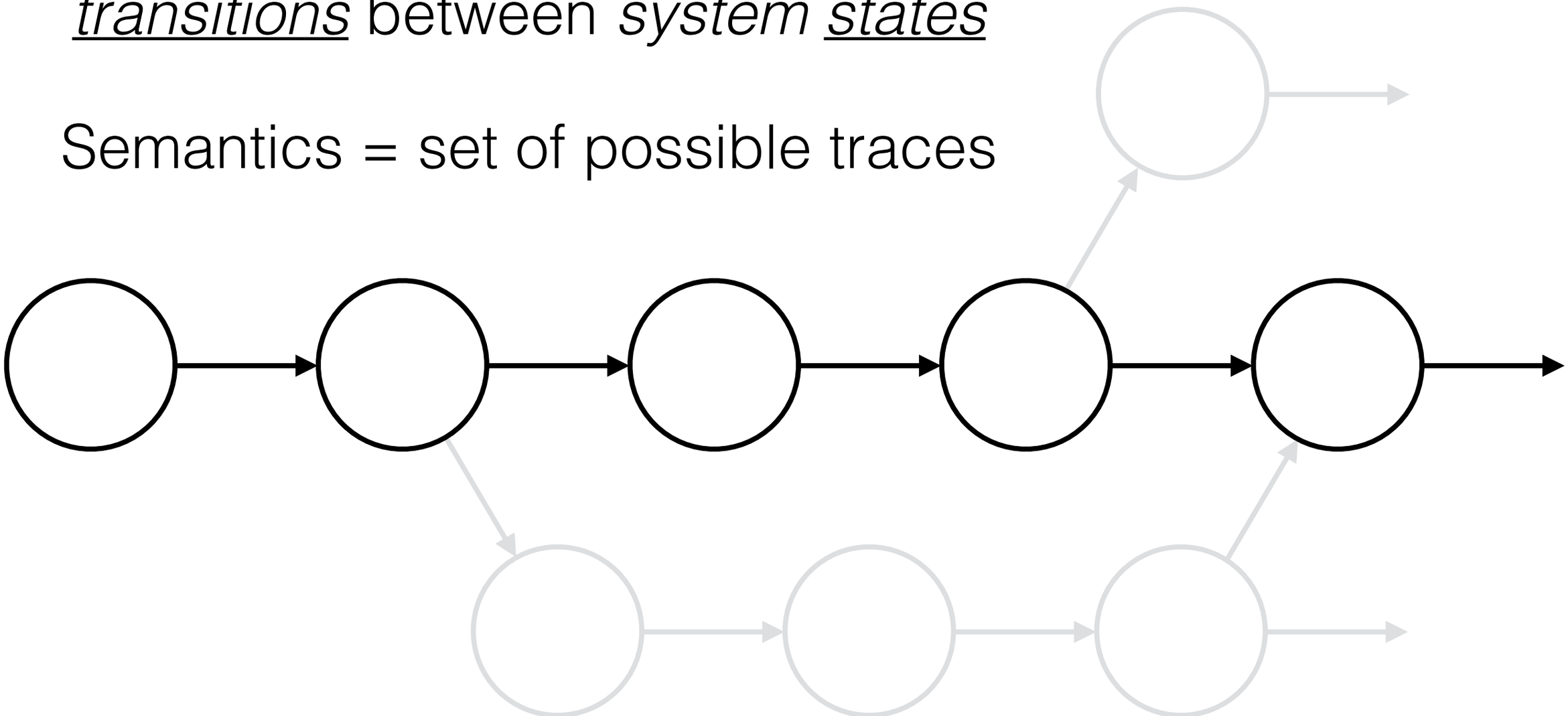
AUTOSAR semantics



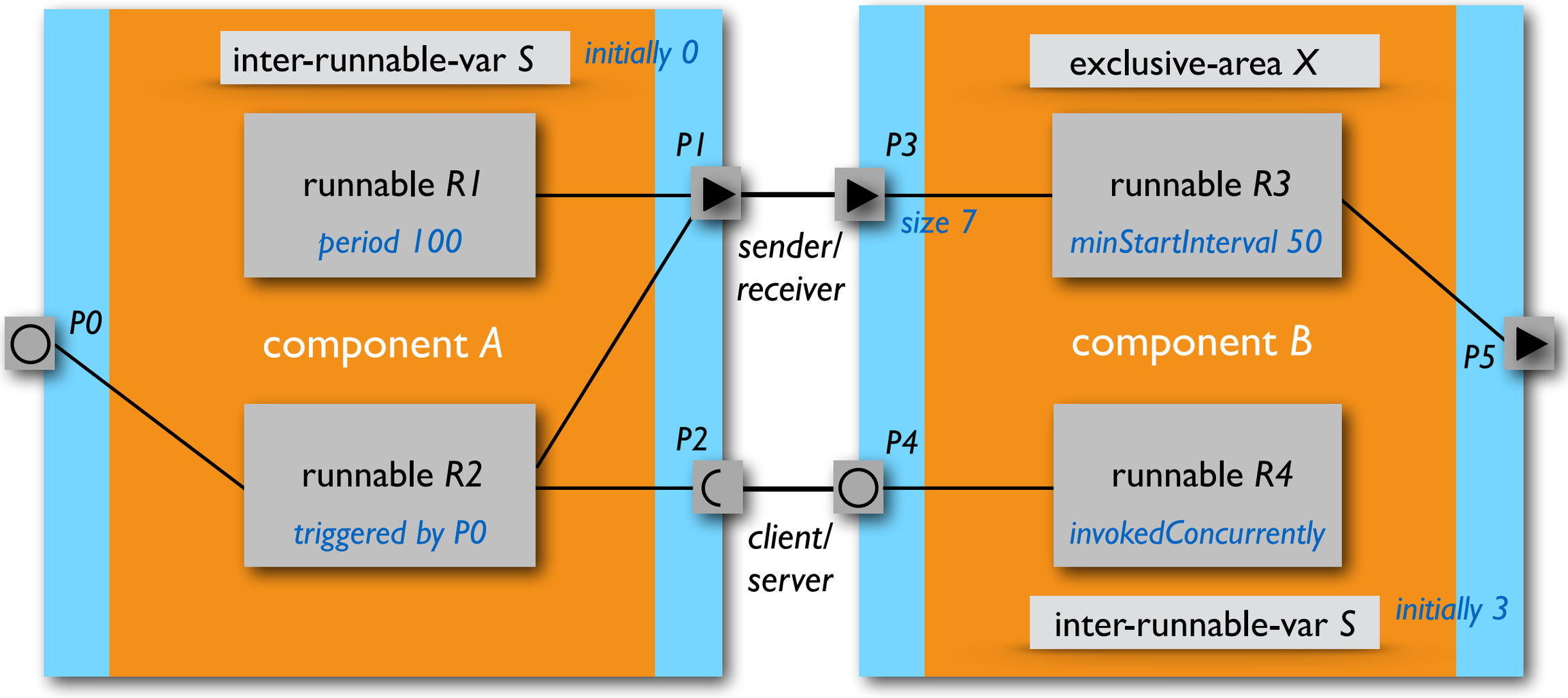
Behaviors

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

Semantics = set of possible traces



An AUTOSAR system



+ constraints and annotations

An AUTOSAR system

parallel composition

atomic processes

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

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

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

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

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

rinst(R1:A, ...)

rinst(R1:A, ...)

rinst(R3:B, ...)

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

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

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

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

---

facts

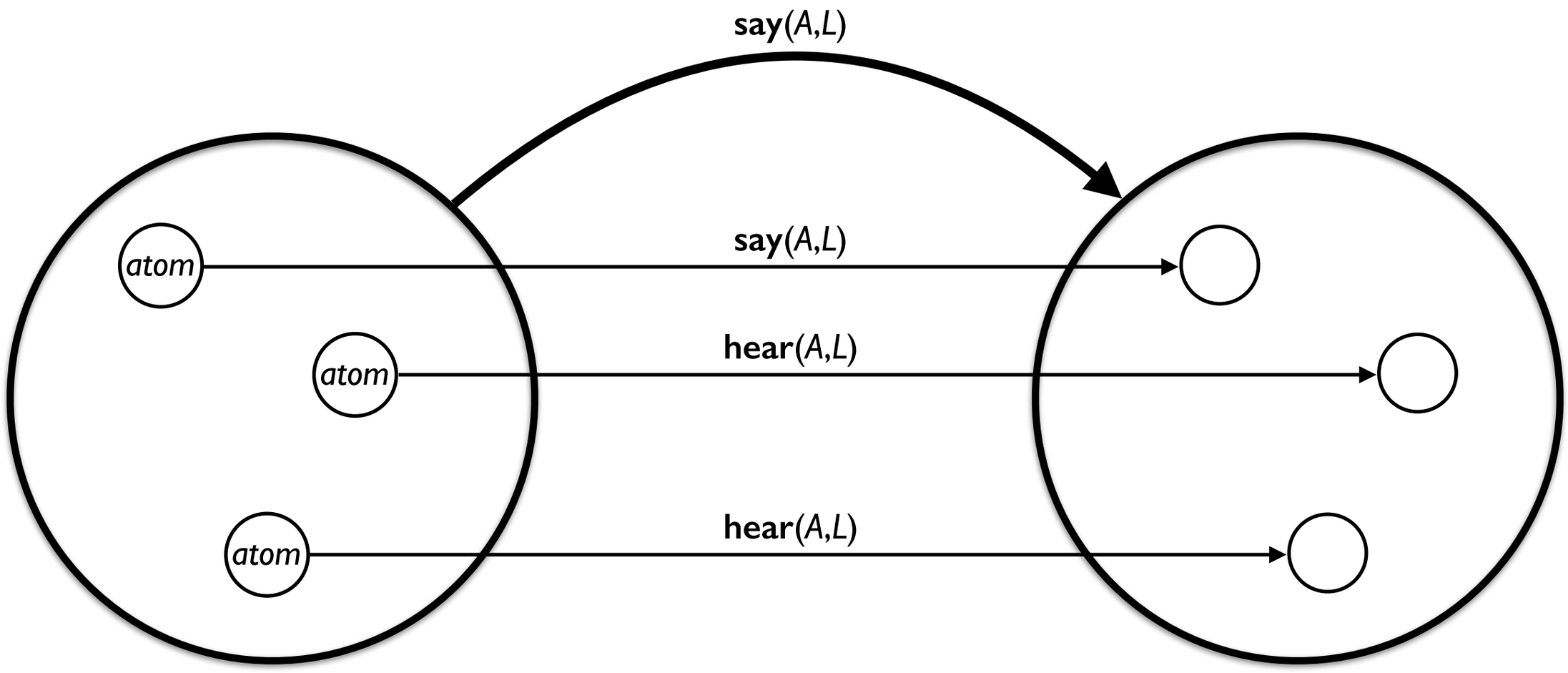
implementation(*R1:A*, *Code for R1*)
implementation(*R2:A*, *Code for R2*)
implementation(*R3:A*, *Code for R3*)
implementation(*R4:A*, *Code for R4*)

P1:A ⇒ *P3:B*

P2:A ⇒ *P4:B*

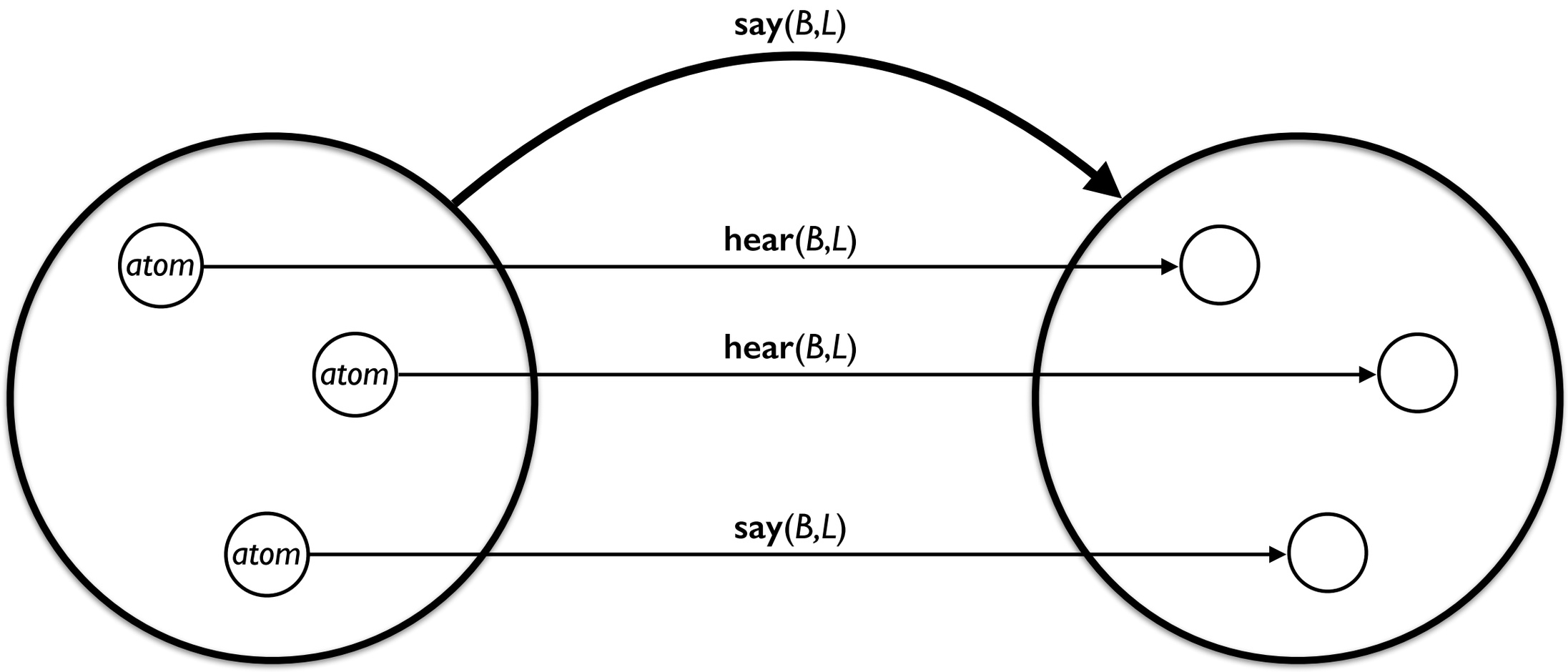
initial(*S:A*, 0)
period(*R1:A*, 100)
initial(*S:B*, 3)
size(*P3:B*, 7)

Labelled transitions



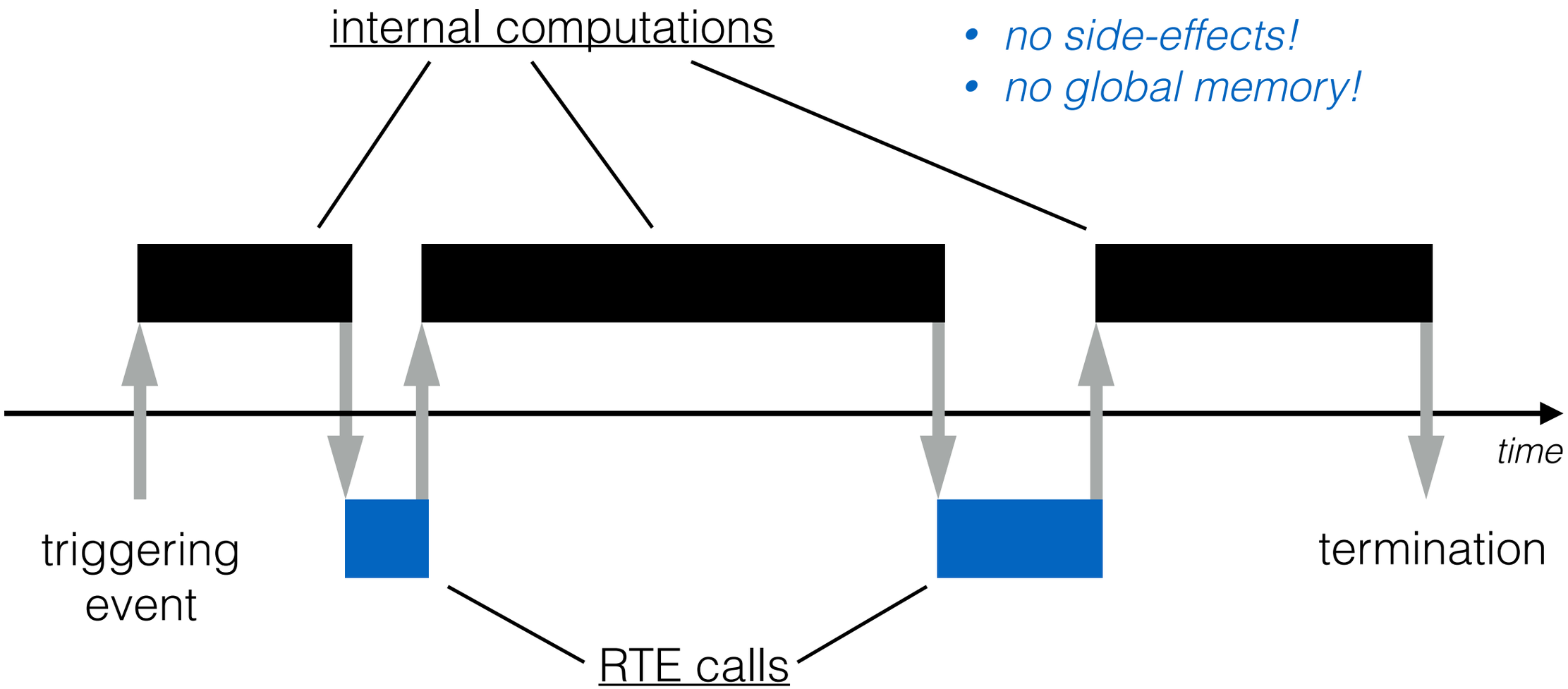
broadcast

Labelled transitions



non-determinism

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*)

Some simple transitions

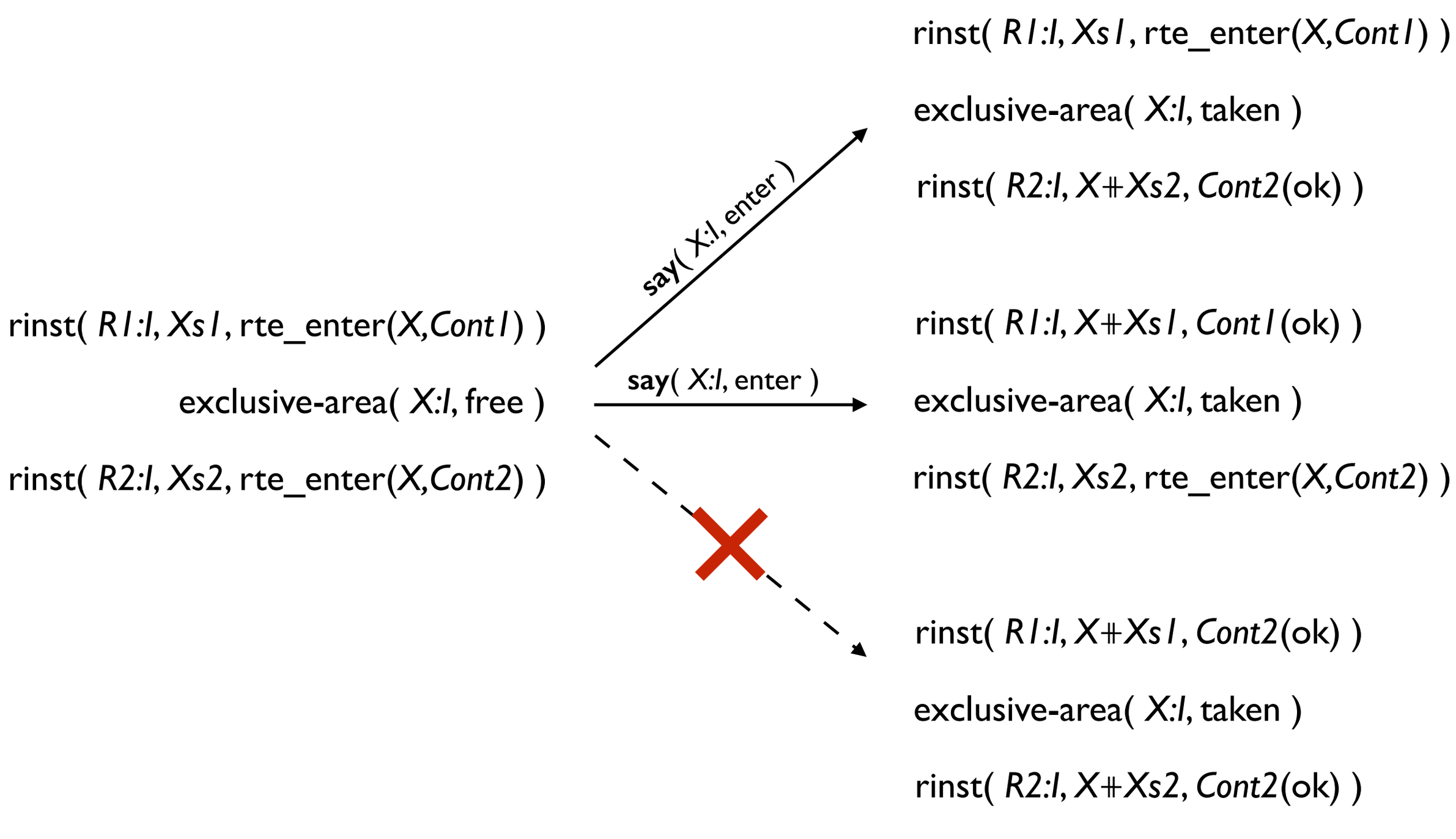
rinst(*R:I*, *X*, rte_enter(*X*,*Cont*)) —**say**(*X:I*, enter)→ rinst(*R:I*, *X*#*Xs*, *Cont*(ok))

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

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

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

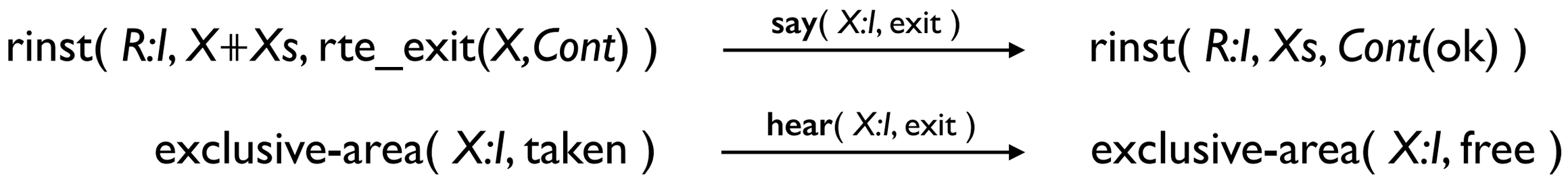
Resulting behaviors



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 <u>are exited</u> in the reverse order they were entered.” [What if they aren’t?]



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

Spawning instances

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

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

one bit of info

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

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

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

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

A semantic pitfall

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

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

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

say(*I:R*, new)

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

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

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

say(*I:R*, new)

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

2 elements,
2 instances

A semantic pitfall

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

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

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

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

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

say(*R:I*, new)

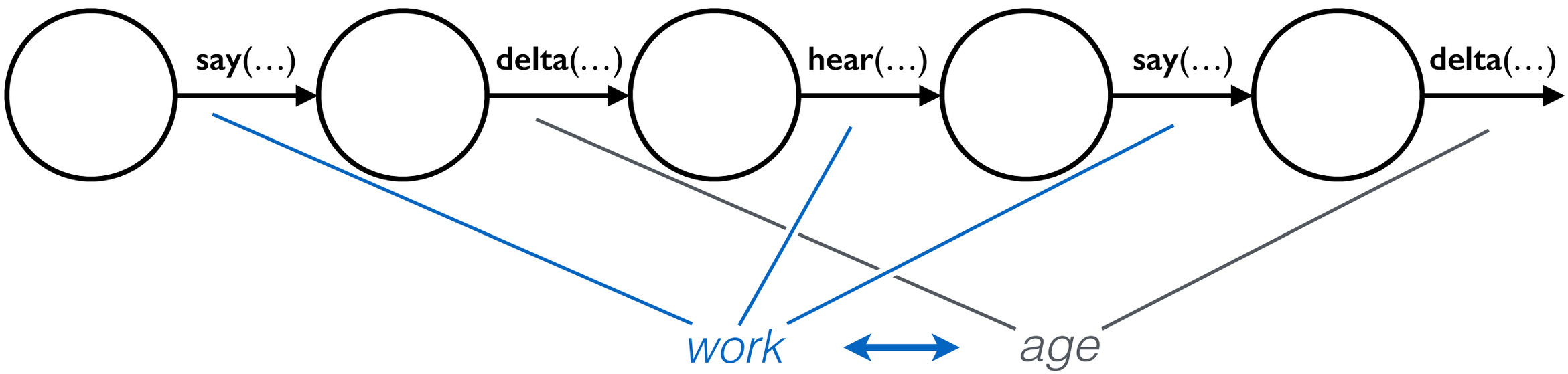
runnable(*R:I*, 0, idle, 1)
rinst(*R:I*, [], *Code*)
qelem(*P:I*, *N*, [1,2])

2 elements,
only 1 instance!

Passing time

if $V \leq T$:

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



relationship not restricted (arbitrarily fast platform)

Prolog formulation

Code

```
rinst(R:I, Xs, rte_receive(P,Cont)) ---say(P:I,rcv(V))---> rinst(R:I, 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?

Not for simulating systems — for this we turn to...

AUTOSAR DSL in Haskell

Embedding Haskell computations inside AUTOSAR

Embedding AUTOSAR simulations inside Haskell

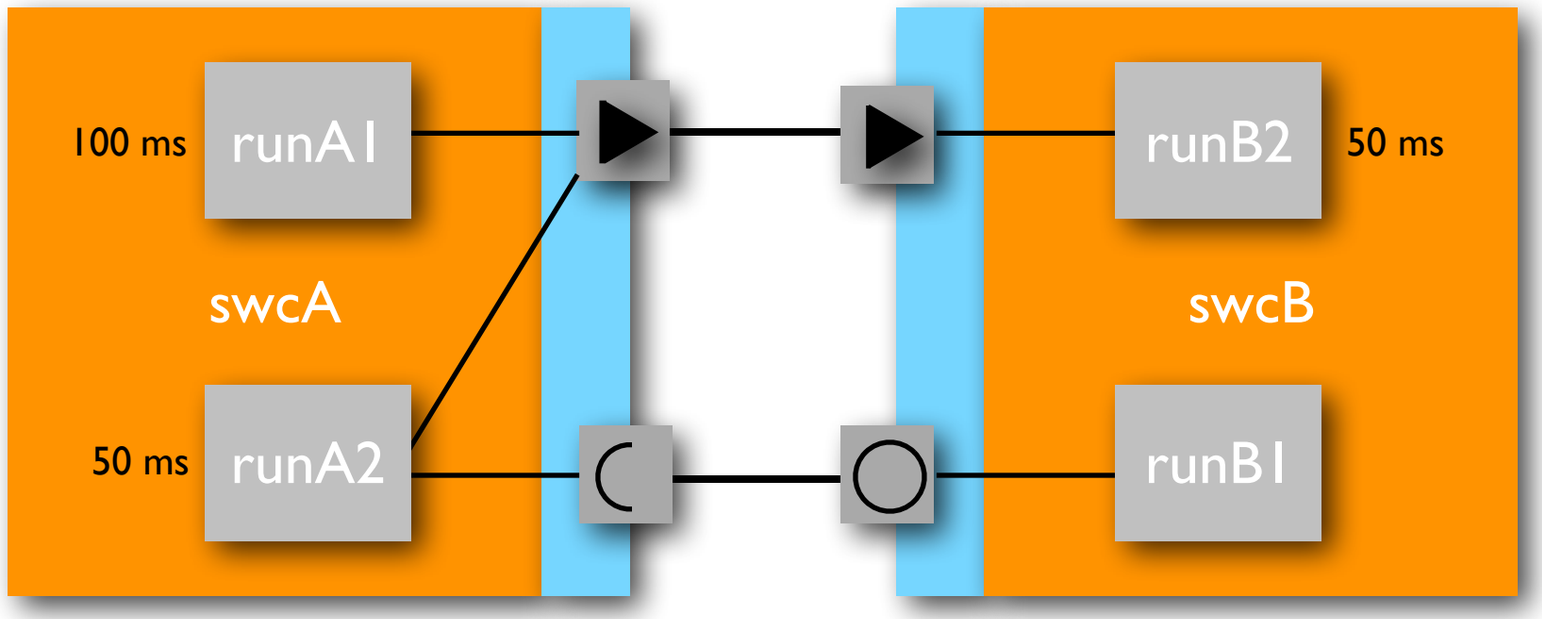
```
instance Monad (RTE c)          -- a monad of RTE operations

enter       ::                  ExclusiveArea c -> RTE c (StdRet ())
exit        ::                  ExclusiveArea c -> RTE c (StdRet ())

irvWrite    :: Data a =>        InterRunnableVariable a c -> a -> RTE c (StdRet ())
irvRead     :: Data a =>        InterRunnableVariable a c -> RTE c (StdRet a)

send        :: Data a =>        ProvidedQueueElement a c -> a -> RTE c (StdRet ())
receive     :: Data a =>        RequiredDataElement a c -> RTE c (StdRet a)

write       :: Data a =>        ProvidedDataElement a c -> a -> RTE c (StdRet ())
read        :: Data a =>        RequiredDataElement a c -> RTE c (StdRet a)
isUpdated   ::                  RequiredDataElement a c -> RTE c (StdRet Bool)
invalidate  ::                  ProvidedDataElement a c -> RTE c (StdRet ())

call        :: (Data a, Data b) =>
                                RequiredOperation a b c -> a -> RTE c (StdRet b)
```

AUTOSAR DSL in Haskell

instance Monad (AR c) -- *a monad of* ***structural building blocks***

```
requiredDataElement     :: AR c (RequiredDataElement a c)
providedDataElement     :: AR c (ProvidedDataElement a c)

requiredQueueElement    :: Int -> AR c (RequiredQueueElement a c)
providedQueueElement :: AR c (ProvidedQueueElement a c)

requiredOperation       :: AR c (RequiredOperation a b c)
providedOperation       :: AR c (ProvidedOperation a b c)

interRunnableVariable   :: Data a => a -> AR c (InterRunnableVariable a c)
exclusiveArea           :: AR c (ExclusiveArea c)

runnable                :: Invocation -> [Trigger c] -> RTE c a -> AR c ()
serverRunnable          :: (Data a, Data b) =>
                           Invocation -> [ProvidedOperation a b c] -> (a -> RTE c b) -> AR c ()

component               :: (forall c . AR c a) -> AR c' a
connect                 :: Connectable a b => a -> b -> AR c ()
```

Simple example



Simple example

```
swcA =  component $ do
    pport1 <- providedDataElement
    rport1 <- requiredOperation
    runnable (MinInterval 0) [Timed 0.1] (runA1 pport1)
    runnable (MinInterval 0) [Timed 0.05] (runA2 pport1 rport1)
    return (seal pport1, seal rport1)

swcB = component $ do
    rport2 <- requiredDataElement
    pport2 <- providedOperation
    serverRunnable Concurrent [pport2] runB1
    runnable (MinInterval 0) [Timed 0.05] (runB2 rport2)
    return (seal pport2, seal rport2)

root = do
    (pdata,rop) <- swcA
    (pop,rdata) <- swcB
    connect pdata rdata
    connect rop pop
```

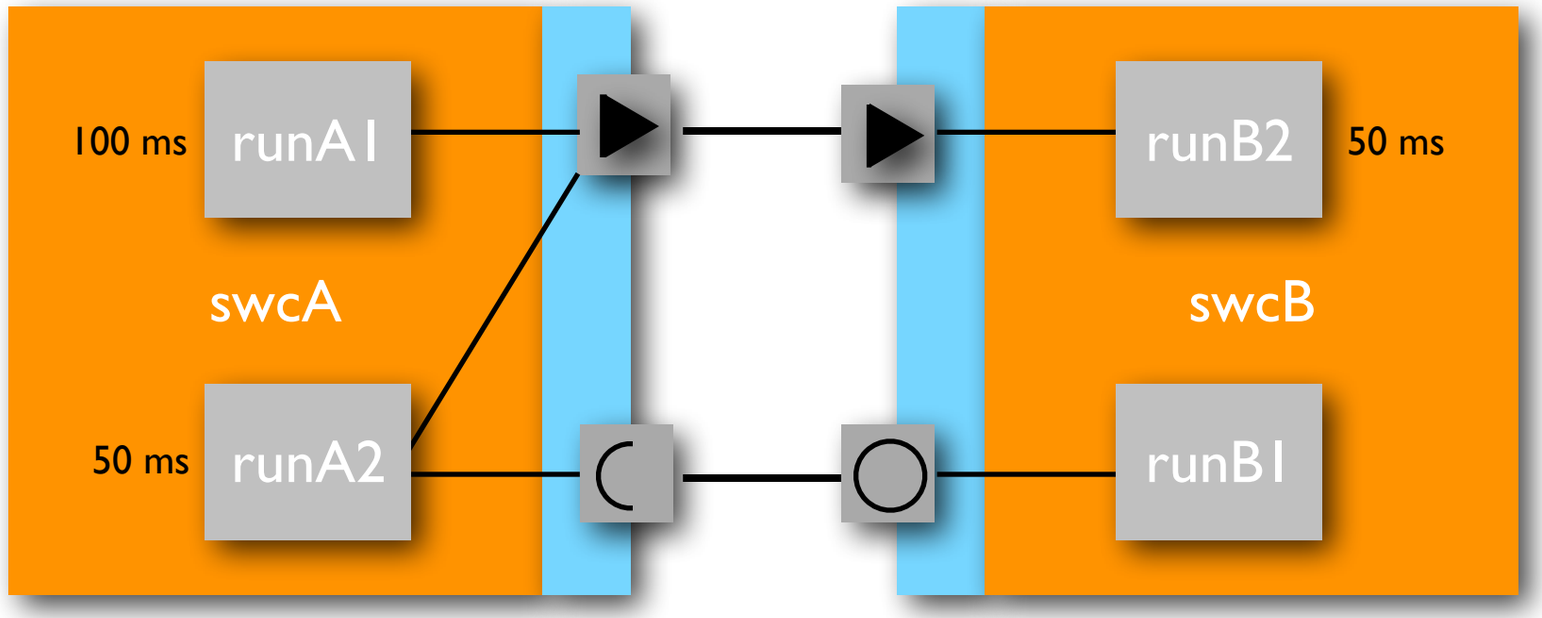
```
runA1 pport1 = do
    rte_write pport1 val
    ...

runA2 pport1 rport1 = do
    val2 <- rte_call rport1 val1
    ...
    rte_write pport1 val2

runB1 arg = do
    ... arg ...
    return res

runB2 rport2 = do
    val <- rte_read rport2
    ...
```

Simple example (trad)

```
FUNC(void, RTE APPL CODE) runA1(void) {
    Int16 val;
    ...
    Rte_Write_pport1_intValue1(val);
    ...
}
```

```
FUNC(void, RTE APPL CODE) runB2(void) {
    Int16 val;
    ...
    Rte_Read_rport2_intValue(&val);
    ...
}
```

```
TASK(Task1) {
   Rte_RECount_Task1_divby2_0--;
   if ( Rte_RECount_Task1_divby2_0 == 0 ) {
      runA1();
   }
   runA2();
   runB2();
   if ( Rte_RECount_Task1_divby2_0 == 0 )
      Rte_RECount_Task1_divby2_0 = 2;
   TerminateTask();
}
```




```
FUNC(void, RTE APPL CODE) runA2(void) {
    String8 val1;
    Int16 val2;
    ...
    Rte_Call_rport1_parse(val1, &val2);
    ...
    Rte_Write_pport1_intValue1(val2);
    ...
}
```

```
FUNC(void, RTE APPL CODE) runB1(String8 arg, Int16 *res ) {
    ... arg ...
    ...
    *res = ...
}
```

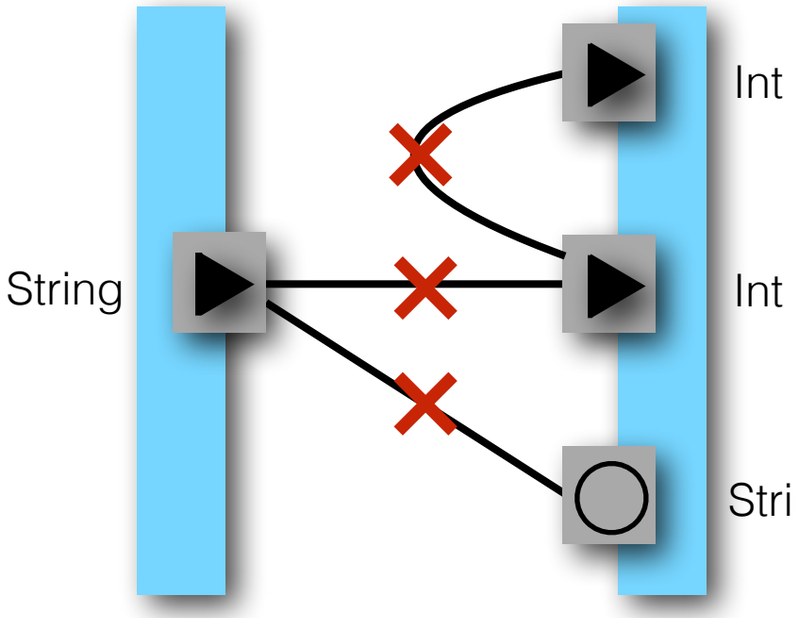
Simple example (trad)

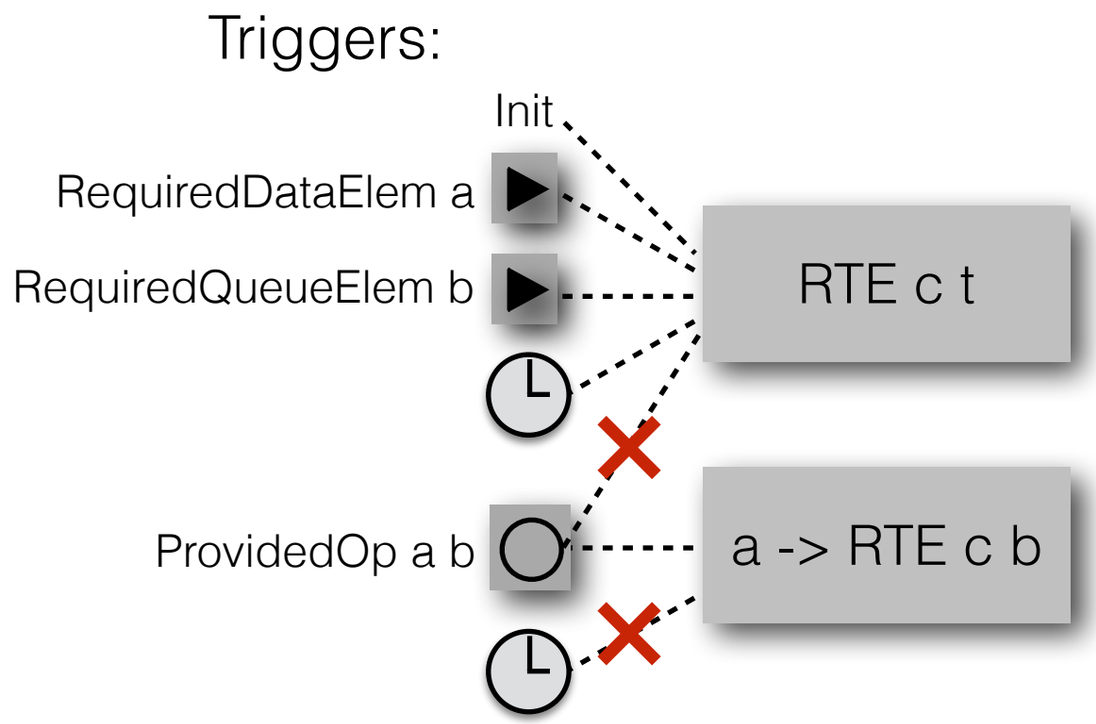
```
<AR-PACKAGE>
  <SHORT-NAME>root</SHORT-NAME>
  <ELEMENTS>
    <ATOMIC-SOFTWARE-COMPONENT-TYPE>
      <SHORT-NAME>swcA</SHORT-NAME>
      <PORTS>
        <P-PORT-PROTOTYPE>
          <SHORT-NAME>pportA1</SHORT-NAME>
          <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
            /interfaces/SR Int16
          </PROVIDED-INTERFACE-TREF>
        </P-PORT-PROTOTYPE>
        <R-PORT-PROTOTYPE>
          <SHORT-NAME>rportA1</SHORT-NAME>
          <REQUIRED-INTERFACE-TREF DEST="CLIENT-SERVER-INTERFACE">
            /interfaces/CS string to int
          </REQUIRED-INTERFACE-TREF>
        </R-PORT-PROTOTYPE>
      </PORTS>
    </ATOMIC-SOFTWARE-COMPONENT-TYPE>
    <ATOMIC-SOFTWARE-COMPONENT-TYPE>
      <SHORT-NAME>swcB</SHORT-NAME>
      <PORTS>
        <P-PORT-PROTOTYPE>
          <SHORT-NAME>pportB1</SHORT-NAME>
          <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-INTERFACE">
            /interfaces/CS string to int
          </PROVIDED-INTERFACE-TREF>
        </P-PORT-PROTOTYPE>
        <R-PORT-PROTOTYPE>
          <SHORT-NAME>rportB1</SHORT-NAME>
          <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
            /interfaces/SR Int16
          </REQUIRED-INTERFACE-TREF>
        </R-PORT-PROTOTYPE>
      </PORTS>
    </ATOMIC-SOFTWARE-COMPONENT-TYPE>
```

```
<INTERNAL-BEHAVIOR>
  <SHORT-NAME>intBehSwc1</SHORT-NAME>
  <COMPONENT-REF DEST="ATOMIC-SOFTWARE-COMPONENT-TYPE">/swc root/swc1</COMPONENT-REF>
  <EVENTS>
    <TIMING-EVENT>
      <SHORT-NAME>Time100ms</SHORT-NAME>
      <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
        /swc root/intBehSwc1/run11
      </START-ON-EVENT-REF>
      <PERIOD>0.1</PERIOD>
    </TIMING-EVENT>
    <TIMING-EVENT>
      <SHORT-NAME>Time50ms</SHORT-NAME>
      <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
        /swc root/intBehSwc1/run12
      </START-ON-EVENT-REF>
      <PERIOD>0.05</PERIOD>
    </TIMING-EVENT>
  </EVENTS>
  <RUNNABLES>
    <RUNNABLE-ENTITY>
      <SHORT-NAME>run11</SHORT-NAME>
      <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
      <DATA-SEND-POINTS>
        <DATA-SEND-POINT>
          <SHORT-NAME>dwa1</SHORT-NAME>
          <DATA-ELEMENT-IREF>
            <P-PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">
              /swc root/swc1/pport1
            </P-PORT-PROTOTYPE-REF>
            <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-PROTOTYPE">
              /interfaces/SR Int16/intValue1
            </DATA-ELEMENT-PROTOTYPE-REF>
          </DATA-ELEMENT-IREF>
        </DATA-SEND-POINT>
        <DATA-SEND-POINT>
          <SHORT-NAME>dwa2</SHORT-NAME>
          <DATA-ELEMENT-IREF>
            <P-PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">
              /swc root/swc1/pport1
            </P-PORT-PROTOTYPE-REF>
            <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-PROTOTYPE">
              /interfaces/SR Int16/intValue2
            </DATA-ELEMENT-PROTOTYPE-REF>
          </DATA-ELEMENT-IREF>
        </DATA-SEND-POINT>
      </DATA-SEND-POINTS>
      <SYMBOL>run11</SYMBOL>
    </RUNNABLE-ENTITY>
    <RUNNABLE-ENTITY>
      <SHORT-NAME>run12</SHORT-NAME>
      <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
      <DATA-SEND-POINTS>
        <DATA-SEND-POINT>
          <SHORT-NAME>dwa2</SHORT-NAME>
          <DATA-ELEMENT-IREF>
            <P-PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">
              /swc root/swc1/pport1
            </P-PORT-PROTOTYPE-REF>
            <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-PROTOTYPE">
              /interfaces/SR Int16/intValue1
            </DATA-ELEMENT-PROTOTYPE-REF>
          </DATA-ELEMENT-IREF>
        </DATA-SEND-POINT>
      </DATA-SEND-POINTS>
      <SERVER-CALL-POINTS>
        <SYNCHRONOUS-SERVER-CALL-POINT>
          <SHORT-NAME>sscp</SHORT-NAME>
          <OPERATION-IREFS>
            <OPERATION-IREF>
              <R-PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">
                /swc root/swc1/rport1
              </R-PORT-PROTOTYPE-REF>
              <OPERATION-PROTOTYPE-REF DEST="OPERATION-PROTOTYPE">
                /interfaces/CS string to int/parse
              </OPERATION-PROTOTYPE-REF>
            </OPERATION-IREF>
          </OPERATION-IREFS>
        </SYNCHRONOUS-SERVER-CALL-POINT>
      </SERVER-CALL-POINTS>
      <SYMBOL>run12</SYMBOL>
    </RUNNABLE-ENTITY>
  </RUNNABLES>
  <SUPPORTS-MULTIPLE-INSTANTIATION>false</SUPPORTS-MULTIPLE-INSTANTIATION>
</INTERNAL-BEHAVIOR>
```

```
<INTERNAL-BEHAVIOR>
  <SHORT-NAME>intBehSwc2</SHORT-NAME>
  <COMPONENT-REF DEST="ATOMIC-SOFTWARE-COMPONENT-TYPE">/swc root/swc2</COMPONENT-REF>
  <EVENTS>
    <TIMING-EVENT>
      <SHORT-NAME>Time50ms</SHORT-NAME>
      <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
        /swc root/intBehSwc2/run22
      </START-ON-EVENT-REF>
      <PERIOD>0.05</PERIOD>
    </TIMING-EVENT>
    <OPERATION-INVOKED-EVENT>
      <SHORT-NAME>operationInvoke</SHORT-NAME>
      <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
        /swc root/intBehSwc2/run21
      </START-ON-EVENT-REF>
      <OPERATION-IREF>
        <P-PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">
          /swc root/swc2/pport1
        </P-PORT-PROTOTYPE-REF>
        <OPERATION-PROTOTYPE-REF DEST="OPERATION-PROTOTYPE">
          /interfaces/CS string to int/parse
        </OPERATION-PROTOTYPE-REF>
      </OPERATION-IREF>
    </OPERATION-INVOKED-EVENT>
  </EVENTS>
  <RUNNABLES>
    <RUNNABLE-ENTITY>
      <SHORT-NAME>run21</SHORT-NAME>
      <CAN-BE-INVOKED-CONCURRENTLY>true</CAN-BE-INVOKED-CONCURRENTLY>
      <SYMBOL>run21</SYMBOL>
    </RUNNABLE-ENTITY>
    <RUNNABLE-ENTITY>
      <SHORT-NAME>run22</SHORT-NAME>
      <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
      <DATA-RECEIVE-POINTS>
        <DATA-RECEIVE-POINT>
          <SHORT-NAME>dra1</SHORT-NAME>
          <DATA-ELEMENT-IREF>
            <R-PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">
              /swc root/swc2/rport1
            </R-PORT-PROTOTYPE-REF>
            <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-PROTOTYPE">
              /interfaces/SR Int16/intValue1
            </DATA-ELEMENT-PROTOTYPE-REF>
          </DATA-ELEMENT-IREF>
        </DATA-RECEIVE-POINT>
        <DATA-RECEIVE-POINT>
          <SHORT-NAME>dra2</SHORT-NAME>
          <DATA-ELEMENT-IREF>
            <R-PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">
              /swc root/swc2/rport1
            </R-PORT-PROTOTYPE-REF>
            <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-PROTOTYPE">
              /interfaces/SR Int16/intValue2
            </DATA-ELEMENT-PROTOTYPE-REF>
          </DATA-ELEMENT-IREF>
        </DATA-RECEIVE-POINT>
      </DATA-RECEIVE-POINTS>
      <SYMBOL>run22</SYMBOL>
    </RUNNABLE-ENTITY>
  </RUNNABLES>
  <SUPPORTS-MULTIPLE-INSTANTIATION>false</SUPPORTS-MULTIPLE-INSTANTIATION>
</INTERNAL-BEHAVIOR>
<IMPLEMENTATION>
  <SHORT-NAME>implSwc1</SHORT-NAME>
  <BEHAVIOR-REF DEST="INTERNAL-BEHAVIOR">/swc root/intBehSwc1</BEHAVIOR-REF>
  <CODE-DESCRIPTOR>
    <SHORT-NAME>src</SHORT-NAME>
    <TYPE>SRC</TYPE>
  </CODE-DESCRIPTOR>
  <PROGRAMMING-LANGUAGE>C</PROGRAMMING-LANGUAGE>
</IMPLEMENTATION>
<IMPLEMENTATION>
  <SHORT-NAME>implSwc2</SHORT-NAME>
  <BEHAVIOR-REF DEST="INTERNAL-BEHAVIOR">/swc root/intBehSwc2</BEHAVIOR-REF>
  <CODE-DESCRIPTOR>
    <SHORT-NAME>src</SHORT-NAME>
    <TYPE>SRC</TYPE>
  </CODE-DESCRIPTOR>
  <PROGRAMMING-LANGUAGE>C</PROGRAMMING-LANGUAGE>
</IMPLEMENTATION>
</ELEMENTS>
</AR-PACKAGE>
```

Captured by types



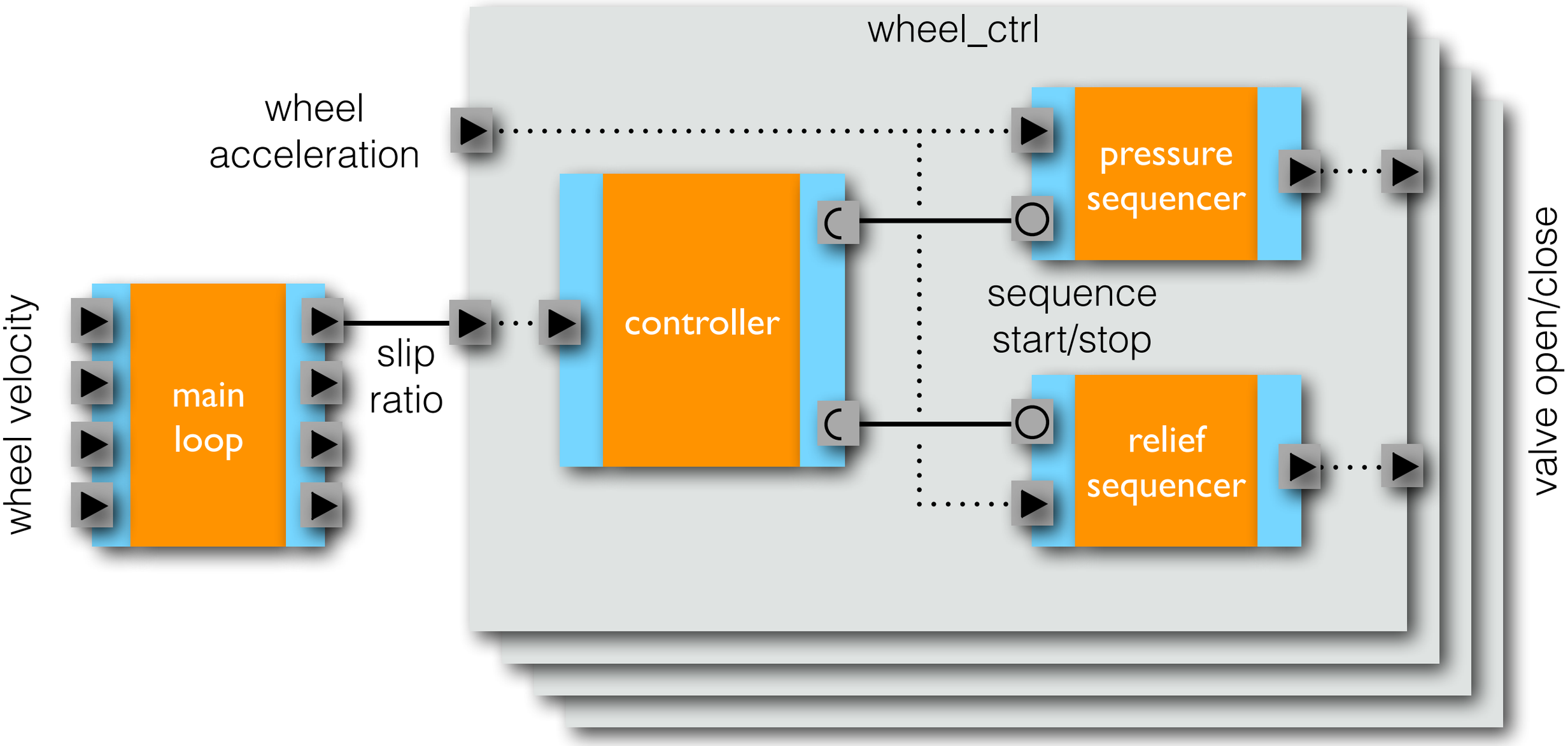



```
escaped <- component $ do
      v <- interRunnableVariable
      ...
      return v
```

(the "runST" trick)

```
method =  do
      x <- irvRead escaped
      ...
```

Demo: An ABS system



ABS top level

```
loop velostreams slipstreams = do
    velos <- mapM (\re -> do Ok v <- rteRead re; return v) velostreams
    let v0 = maximum velos
    mapM (\(v,pe) -> rteSend pe (slip v0 v)) (velos `zip` slipstreams)

main_loop = component $ do
    velostreams <- mapM (const requiredDataElement)   [1..4]
    slipstreams <- mapM (const providedQueueElement)  [1..4]
    runnable (MinInterval 0) [Timed 0.01] (loop velostreams slipstreams)
    return (map seal velostreams, map seal slipstreams)
                        :: [RequiredDataElement Double c]
                :: [RequiredDataElem Double]

abs_system = component $ do
    (velos_in, slips_out) <- main_loop
    wheelctrls <- mapM wheel_ctrl ([1..4] `zip` slips_out)
    return (velos_in, wheelctrls)
```

Code

Structure

ABS wheel controller

```
wheel_ctrl (i,slipstream) = component $ do
        (slip, onoff_pressure, onoff_relief) <- controller
        (accel_p, ctrl_p, valve_p) <- pressure_seq
        (accel_r, ctrl_r, valve_r) <- relief_seq
```

Create sub-components

```
connect slipstream slip
connect onoff_pressure ctrl_p
connect onoff_relief ctrl_r
```

Setup the internal wiring

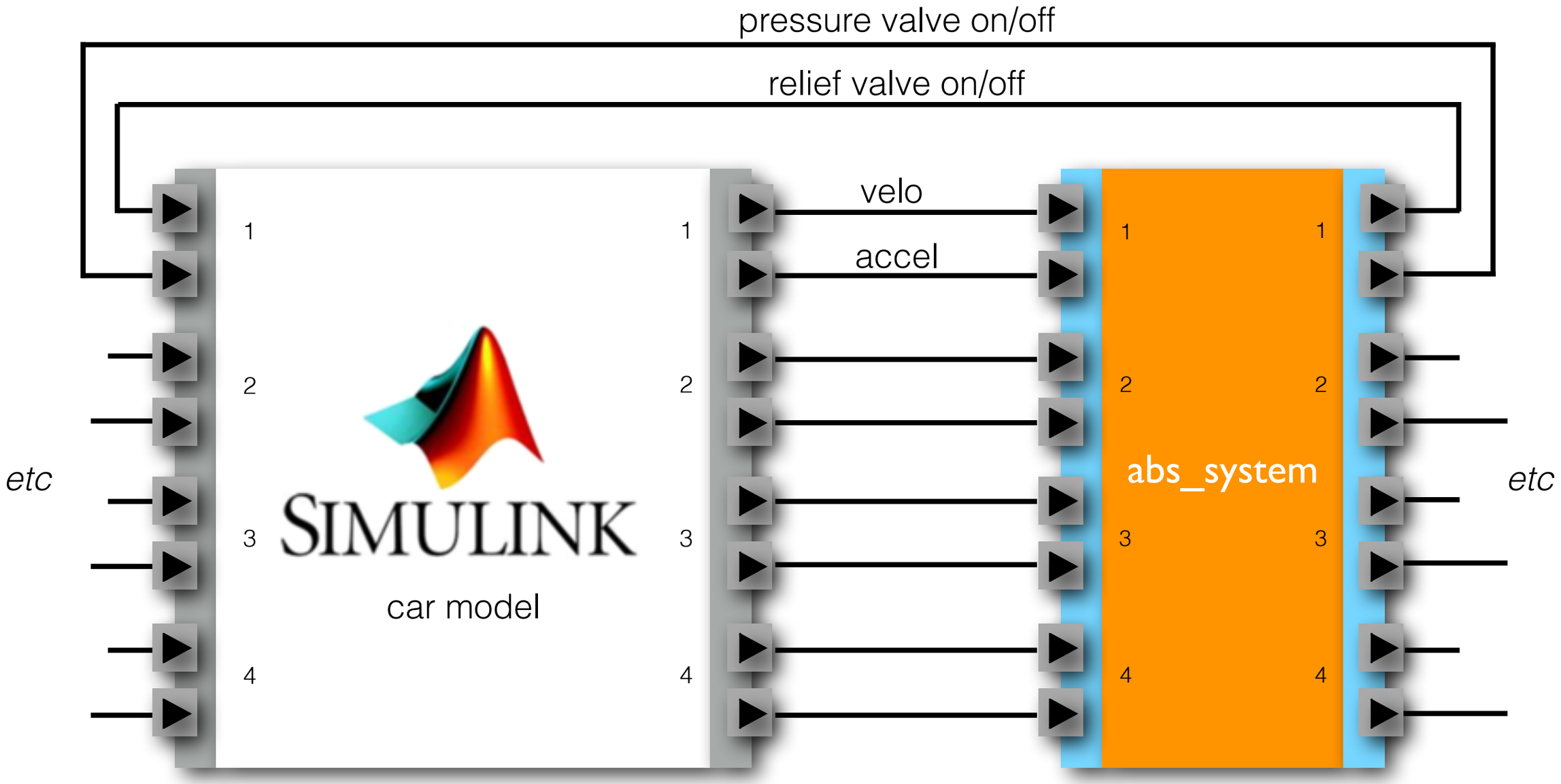
```
when (i==1) $ do
    probeWrite "relief" valve_r ((+2.0) . boolToDouble)
    probeWrite "pressure" valve_p scaleValve_p ((+5.0) . boolToDouble)
```

Attach test probes

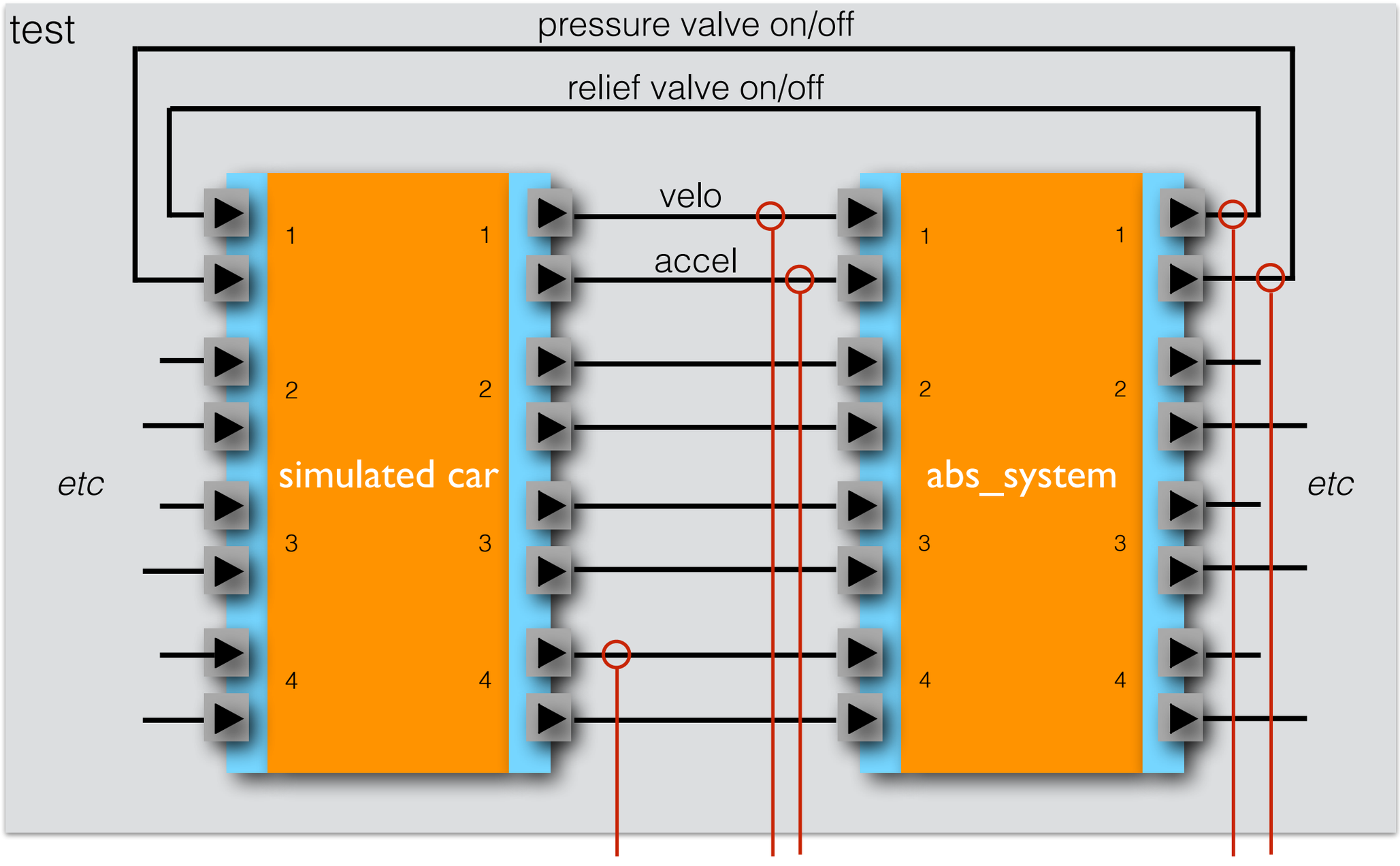
```
return (accel_r, accel_p, valve_r, valve_p)
```

Trivial delegation connectors

Simulation setup

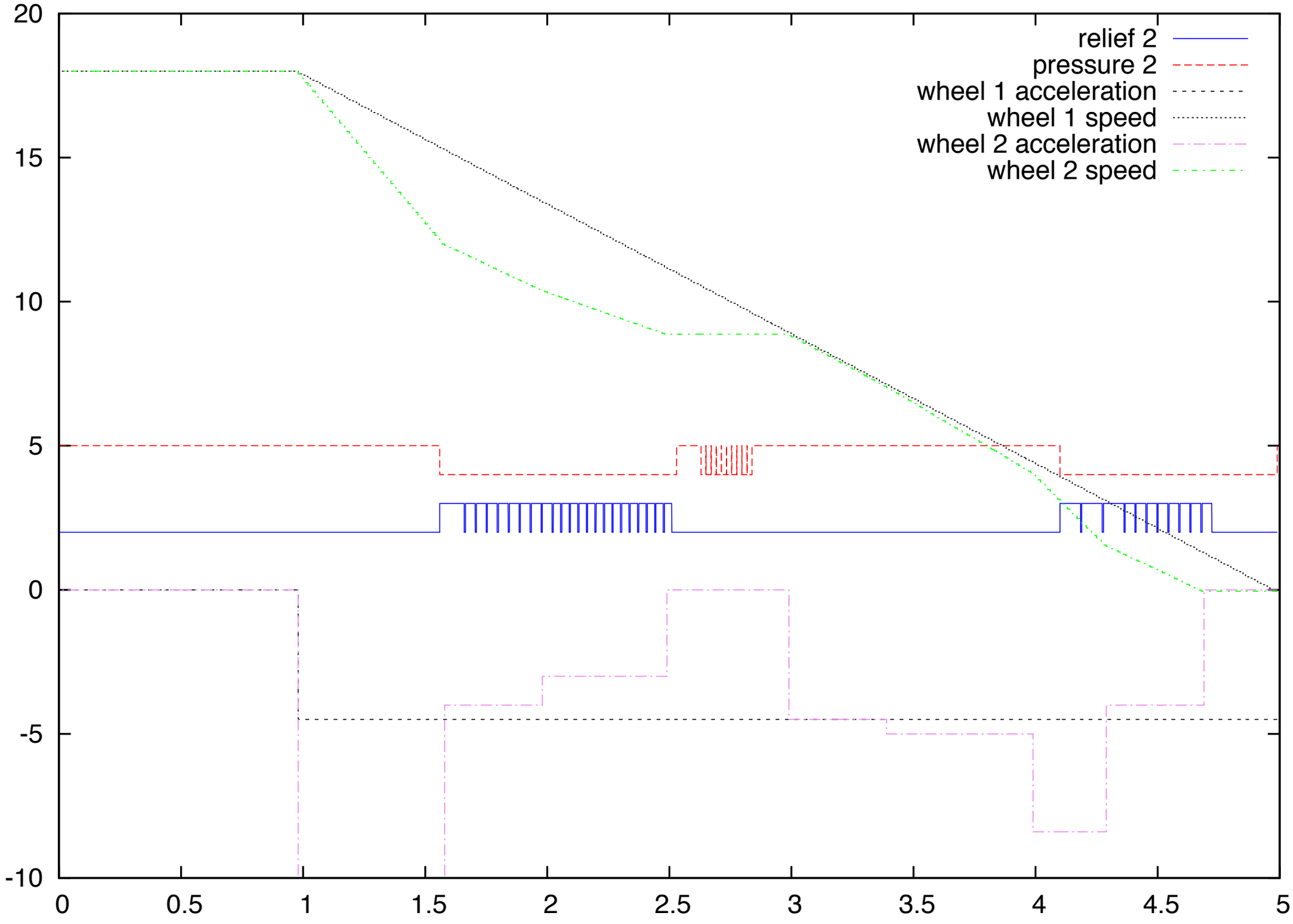


Simulation setup

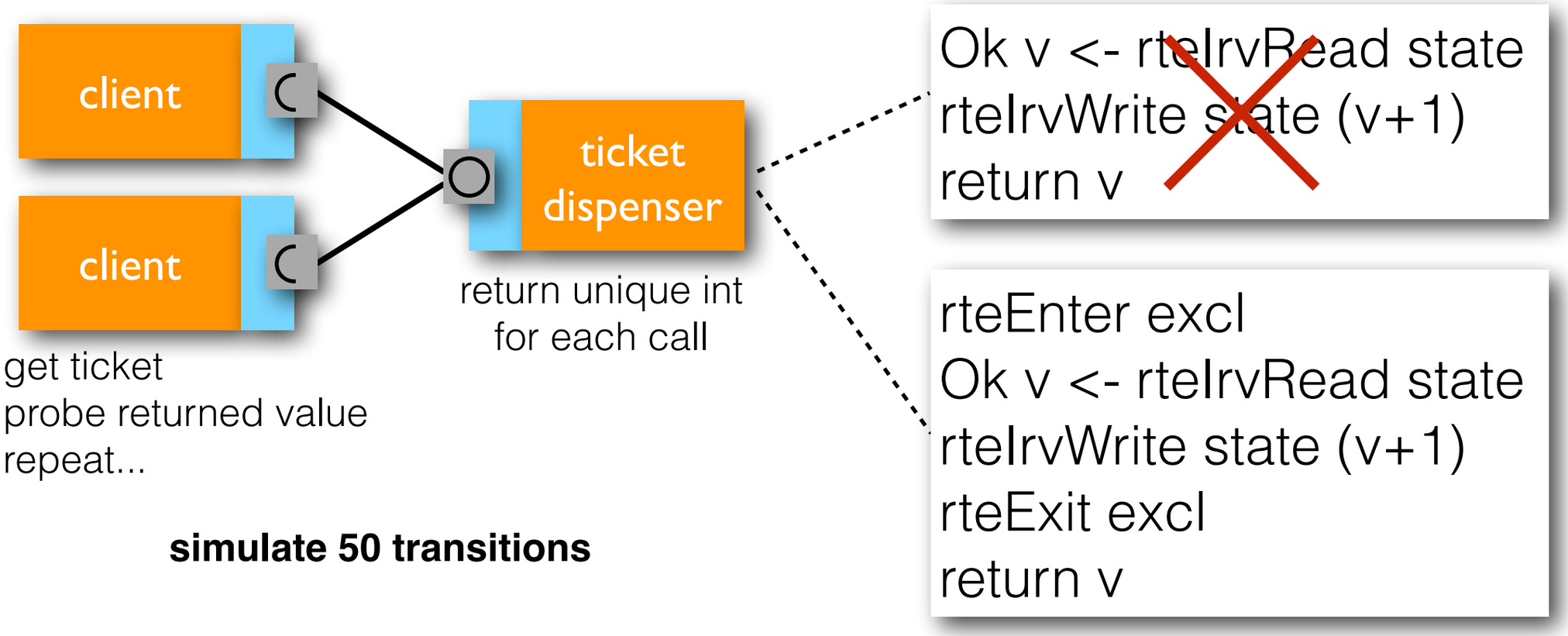



```
main = printLogs trace >> makePlot trace
    where trace = limitTime 5.0 $ execSim (RandomSched (mkStdGen 111)) test
```

Simulation output



Detecting a race condition



simulate 50 transitions

With round-robin scheduling:

```
> ./TicketDispenser
[0,1,2,3,4,5]
> ./TicketDispenser
[0,1,2,3,4,5]
> ./TicketDispenser
[0,1,2,3,4,5]
>
```

With random scheduling:

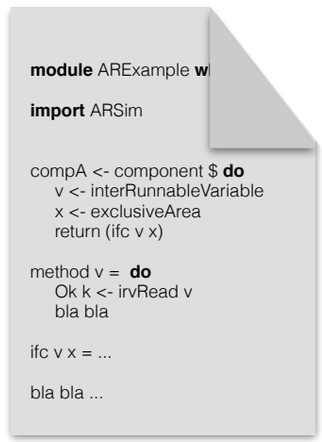
```
> ./TicketDispenser
[1,0,2,3,4]
> ./TicketDispenser
[0,1,2,3,4,3]
> ./TicketDispenser
[0,1,2,3,4,5]
>
```

Corrected:

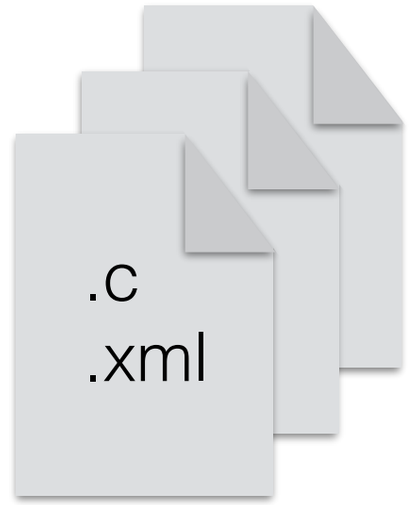
```
> ./TicketDispenser
[0,1,2,3,4]
> ./TicketDispenser
[0,1,2,3]
> ./TicketDispenser
[1,0,2,3]
```

DSL use cases

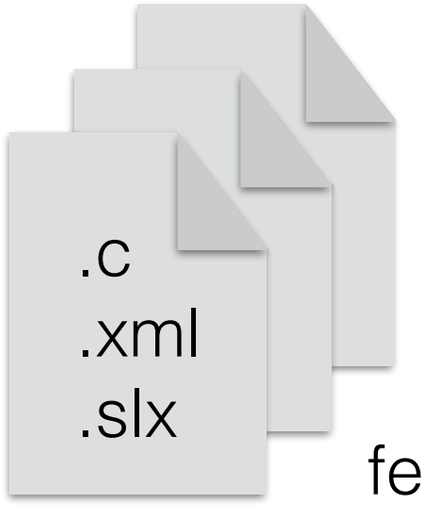
Manual construction



compilation



"decompilation"

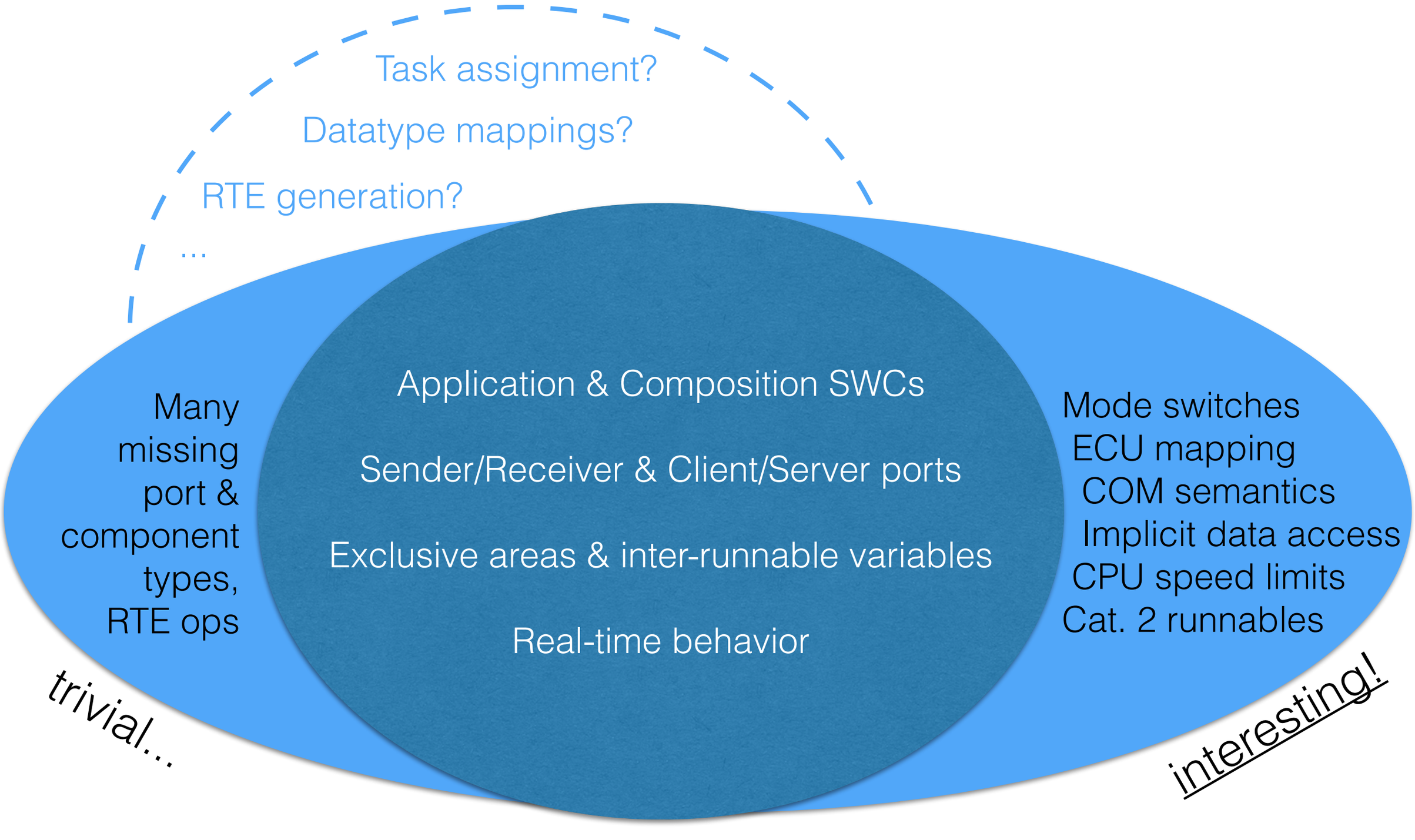


feasible subset?

optimization
refactoring

Simulator
output

Outlook


Task assignment?
Datatype mappings?
RTE generation?
...
Application & Composition SWCs
Sender/Receiver & Client/Server ports
Exclusive areas & inter-runnable variables
Real-time behavior
Many missing port & component types, RTE ops
trivial...
Mode switches
ECU mapping
COM semantics
Implicit data access
CPU speed limits
Cat. 2 runnables
interesting!

Take away bullets

- Platform-independent standard ➜ **platform-independent testing & simulation!**

- Concurrent semantics ➜ **random scheduling!**

- Haskell ⟷ embedded automotive programming! (via a DSL and simulation)

- AUTOSAR runnables ➜ strictly controlled side-effects ➜ **a Haskell monad!**