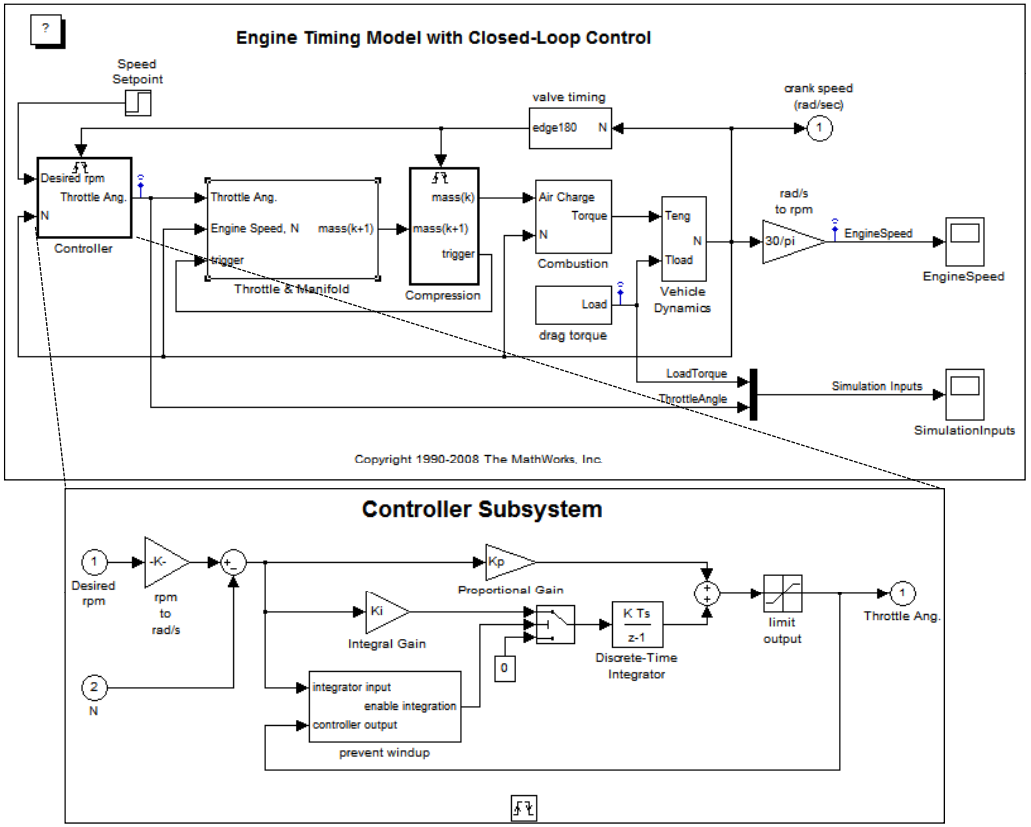
EE 144/244: Fundamental Algorithms for System Modeling, Analysis, and Optimization
Fall 2014

Compositionality

Stavros Tripakis
University of California, Berkeley

Block Diagrams

We should be able to express any such diagram using formal composition operators.

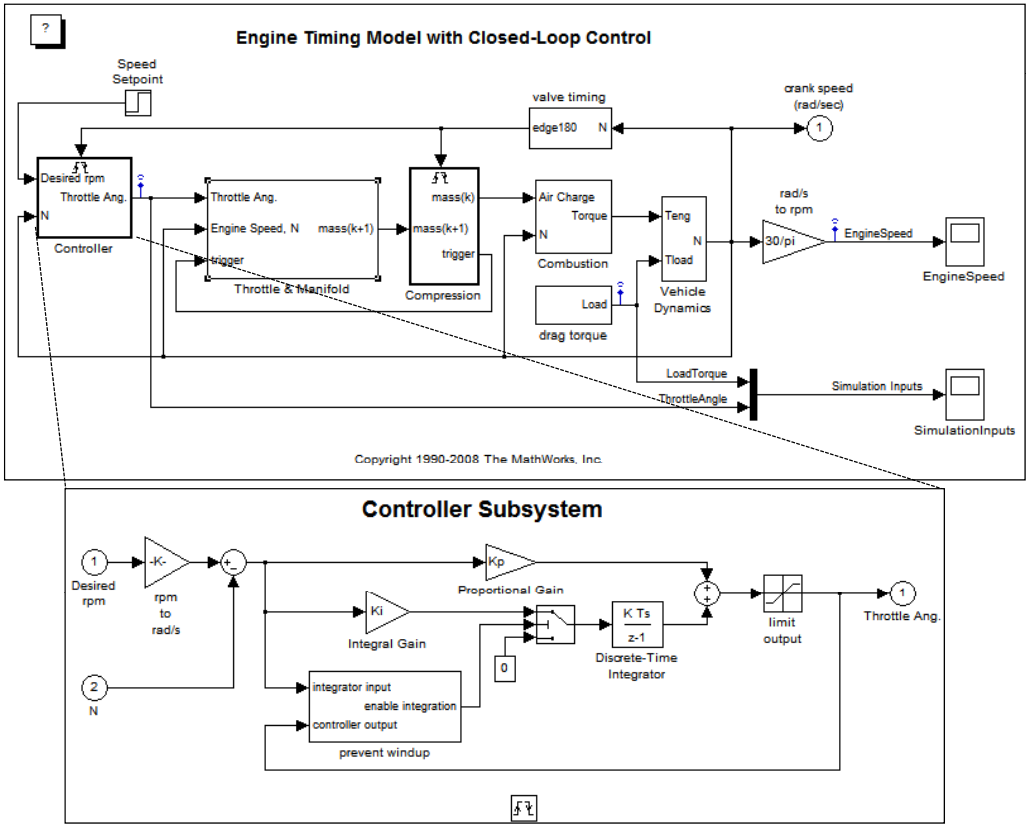


Block Diagrams

We should be able to express any such diagram using formal composition operators.

Basic primitives:

- Serial composition.
- Parallel composition.
- Feedback composition.



Compositionality

The term **compositionality** has many meanings.

One useful meaning:

A language is compositional if

1. *It has a notion of component.*
2. *It has mechanisms to compose components.*
3. *The composition of two or more components is a component.*

Compositionality

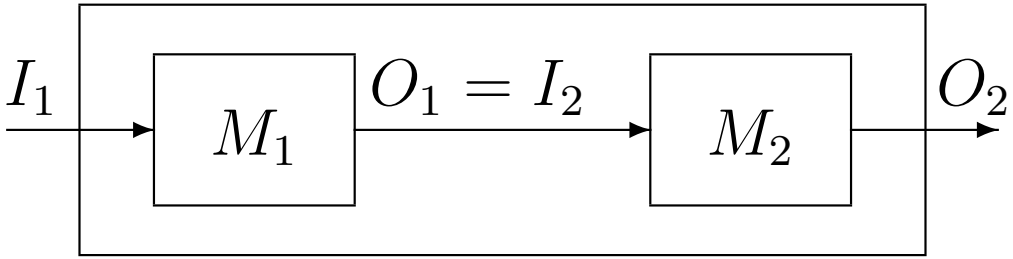
The term **compositionality** has many meanings.

One useful meaning:

> *A language is compositional if*
>
> 1. *It has a notion of component.*
> 2. *It has mechanisms to compose components.*
> 3. *The composition of two or more components is a component.*

Is the serial composition of FSMs an FSM? What about parallel composition? What about feedback?

Synchronous Serial Composition of FSMs



Given two Mealy machines M_1 and M_2 with

$$M_i = (I_i, O_i, S_i, s_0^i, \delta_i, \lambda_i)$$

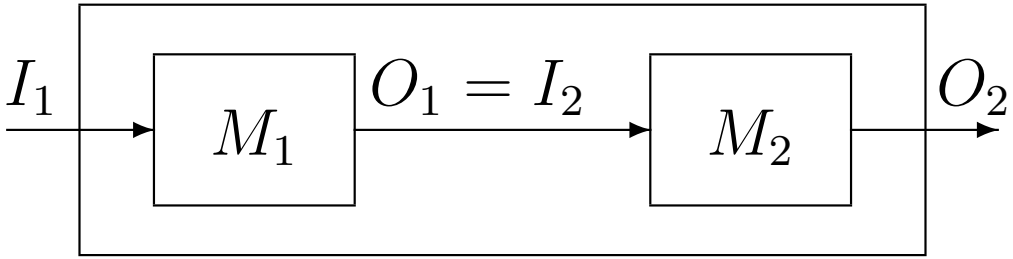
the serial synchronous composition of M_1 and M_2 is a new Mealy machine

$$M = (I_1, O_2, S_1 \times S_2, (s_0^1, s_0^2), \delta, \lambda)$$

where

- $\delta\big((s_1, s_2), a\big) =$

Synchronous Serial Composition of FSMs



Given two Mealy machines M_1 and M_2 with

$$M_i = (I_i, O_i, S_i, s_0^i, \delta_i, \lambda_i)$$

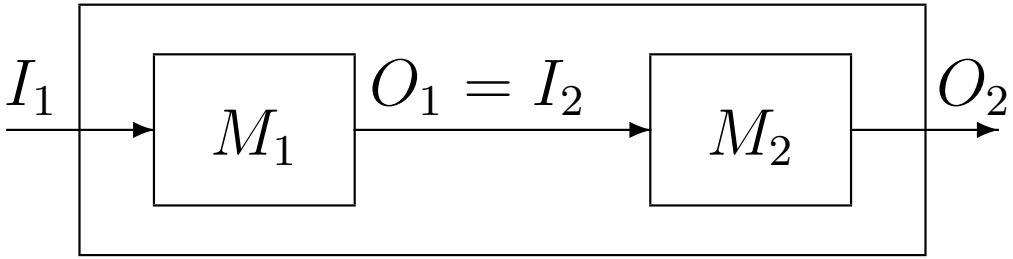
the serial synchronous composition of M_1 and M_2 is a new Mealy machine

$$M = (I_1, O_2, S_1 \times S_2, (s_0^1, s_0^2), \delta, \lambda)$$

where

- $\delta\big((s_1, s_2), a\big) = \big(\delta_1(s_1, a), \delta_2(s_2, \lambda_1(s_1, a))\big)$
- $\lambda\big((s_1, s_2), a\big) =$

Synchronous Serial Composition of FSMs



Given two Mealy machines M_1 and M_2 with

$$M_i = (I_i, O_i, S_i, s_0^i, \delta_i, \lambda_i)$$

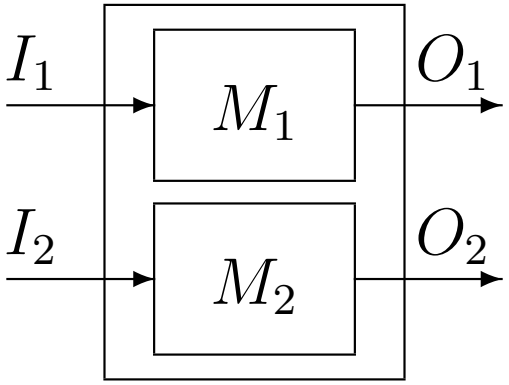
the serial synchronous composition of M_1 and M_2 is a new Mealy machine

$$M = (I_1, O_2, S_1 \times S_2, (s_0^1, s_0^2), \delta, \lambda)$$

where

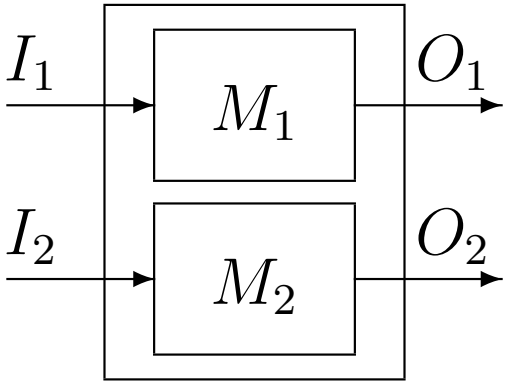
- $\delta\big((s_1, s_2), a\big) = \big(\delta_1(s_1, a), \delta_2(s_2, \lambda_1(s_1, a))\big)$
- $\lambda\big((s_1, s_2), a\big) = \lambda_2(s_2, \lambda_1(s_1, a))$

Synchronous Parallel Composition of FSMs



Quiz: Is the synchronous parallel composition of two Moore machines a Moore machine?

Synchronous Parallel Composition of FSMs



Quiz: Is the synchronous parallel composition of two Moore machines a Moore machine?

Is the synchronous parallel composition of two Mealy machines a Mealy machine?

Parallel Composition of Mealy Machines – the Monolithic Approach

Given two Mealy machines M_1 and M_2 with

$$M_i = (I_i, O_i, S_i, s_0^i, \delta_i, \lambda_i)$$

the parallel composition of M_1 and M_2 is a new Mealy machine

$$M = (I_1 \times I_2, O_1 \times O_2, S_1 \times S_2, (s_0^1, s_0^2), \delta, \lambda)$$

where

- $\delta\big((s_1, s_2), a\big) =$

Parallel Composition of Mealy Machines – the Monolithic Approach

Given two Mealy machines M_1 and M_2 with

$$M_i = (I_i, O_i, S_i, s_0^i, \delta_i, \lambda_i)$$

the parallel composition of M_1 and M_2 is a new Mealy machine

$$M = (I_1 \times I_2, O_1 \times O_2, S_1 \times S_2, (s_0^1, s_0^2), \delta, \lambda)$$

where

- $\delta\big((s_1, s_2), a\big) = \big(\delta_1(s_1, a), \delta_2(s_2, a)\big)$
- $\lambda\big((s_1, s_2), a\big) =$

Parallel Composition of Mealy Machines – the Monolithic Approach

Given two Mealy machines M_1 and M_2 with

$$M_i = (I_i, O_i, S_i, s_0^i, \delta_i, \lambda_i)$$

the parallel composition of M_1 and M_2 is a new Mealy machine

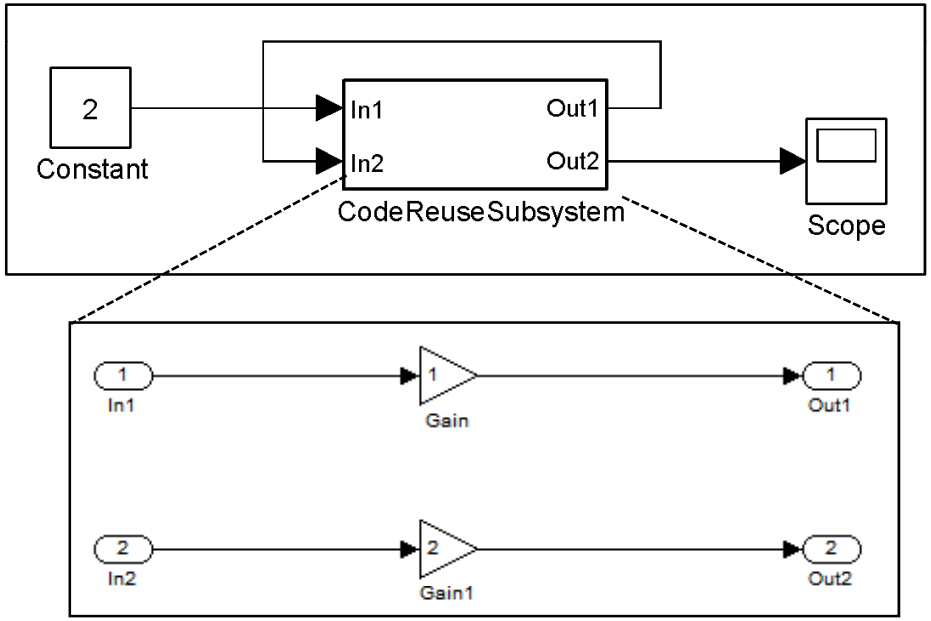
$$M = (I_1 \times I_2, O_1 \times O_2, S_1 \times S_2, (s_0^1, s_0^2), \delta, \lambda)$$

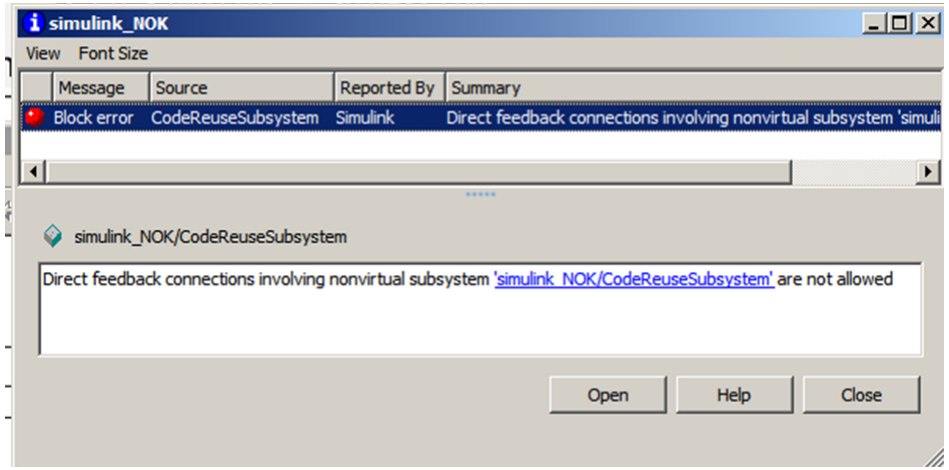
where

- $\delta\big((s_1, s_2), a\big) = \big(\delta_1(s_1, a), \delta_2(s_2, a)\big)$
- $\lambda\big((s_1, s_2), a\big) = \big(\lambda_1(s_1, a), \lambda_2(s_2, a)\big)$

Problem with the Monolithic Approach

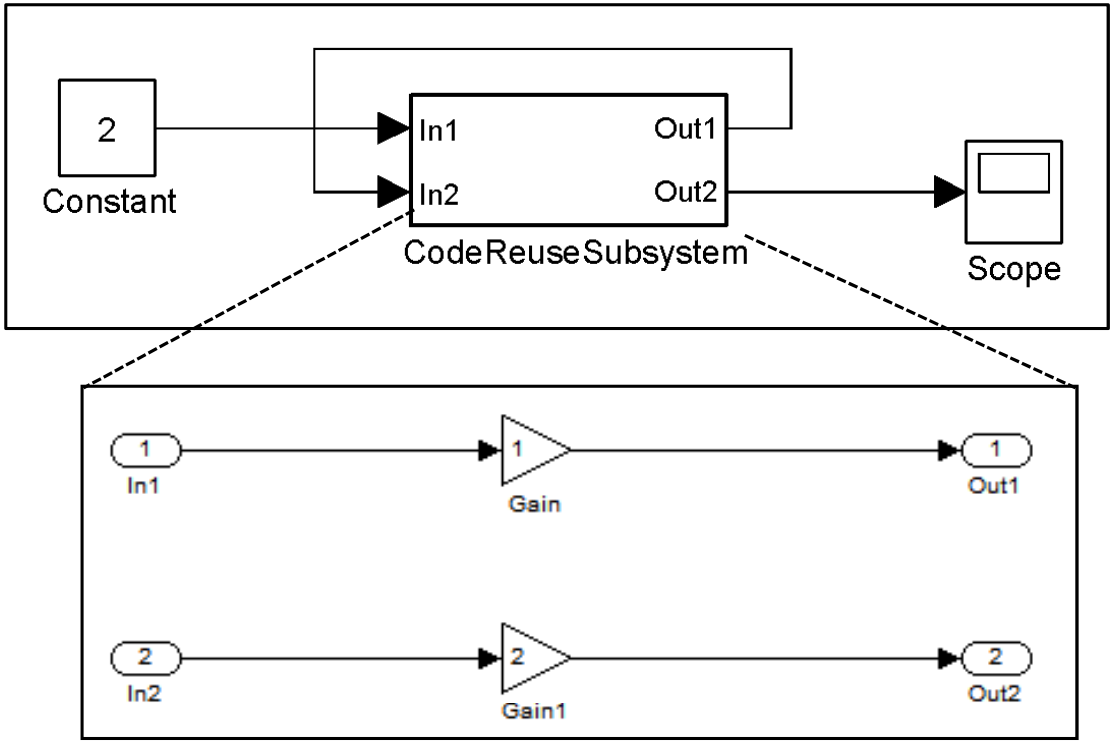
The monolithic approach is **not** compositional.





Problem with the Monolithic Approach

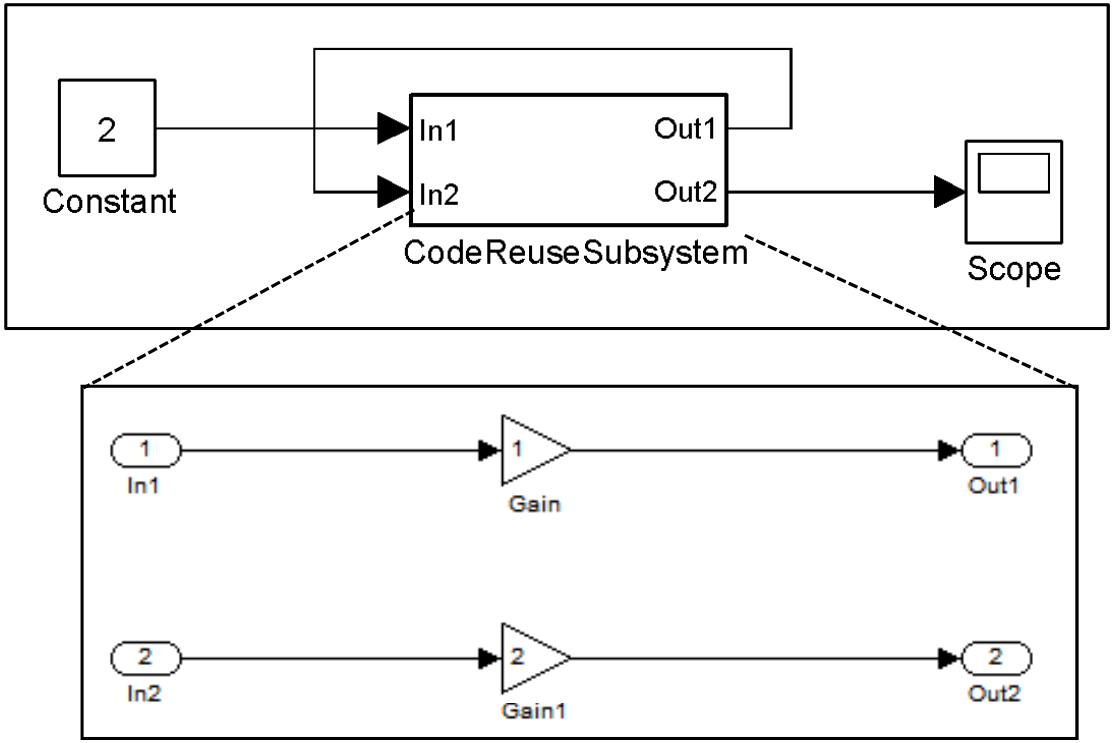
The monolithic approach is **not** compositional.



- Note: replacing the *CodeReuseSubsystem* with a normal *Subsystem* allows Simulink to run the model without problems. Why?

Problem with the Monolithic Approach

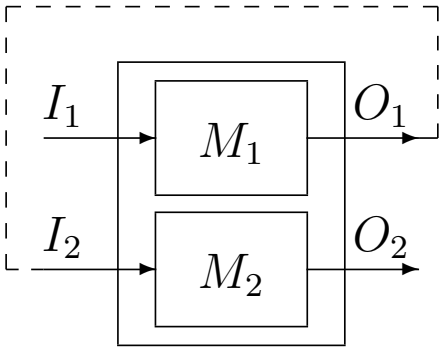
The monolithic approach is **not** compositional.



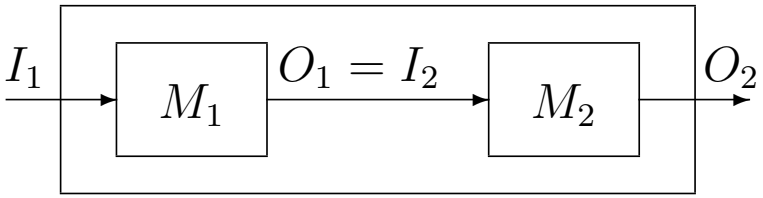
- Note: replacing the *CodeReuseSubsystem* with a normal *Subsystem* allows Simulink to run the model without problems. Why?
  - Simulink *flattens Subsystems* but not *CodeReuseSubsystems*.

Problem with the Monolithic Approach

The monolithic approach is **not** compositional.



$\neq$



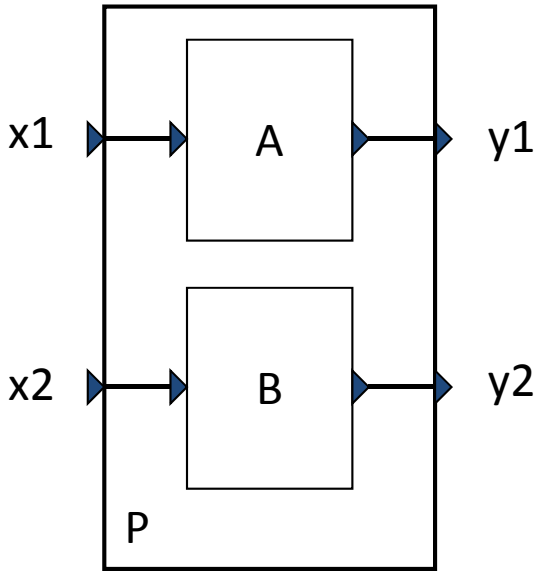
The two compositions are not equivalent!

The Non-Monolithic Approach

[Lublinerman and Tripakis, 2008, Lublinerman et al., 2009]

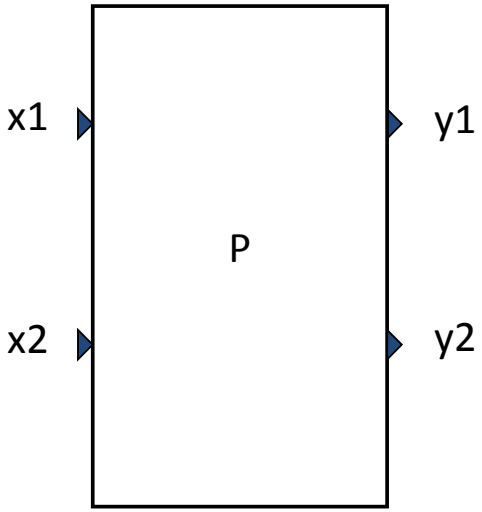
Solution: non-monolithic code

A Mealy machine with multiple output functions




```
P.fire1( x1 ) returns y1 {
   return A.fire( x1 );
}

P.fire2( x2 ) returns y2 {
   return B.fire( x2 );
}
```

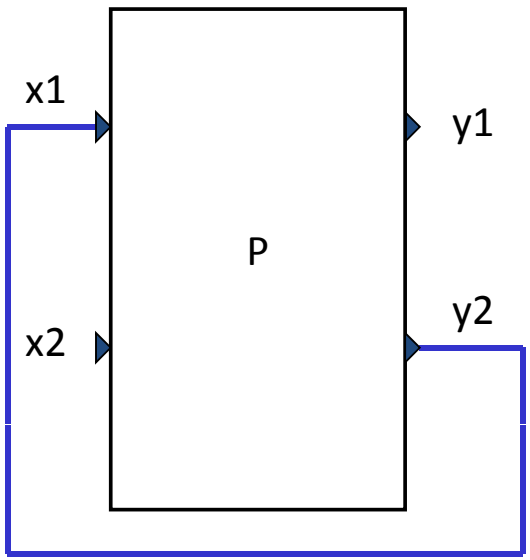
Non-monolithic interface




```
P.fire1( x1 ) returns y1 ;

P.fire2( x2 ) returns y2 ;
```

18

Non-monolithic interface

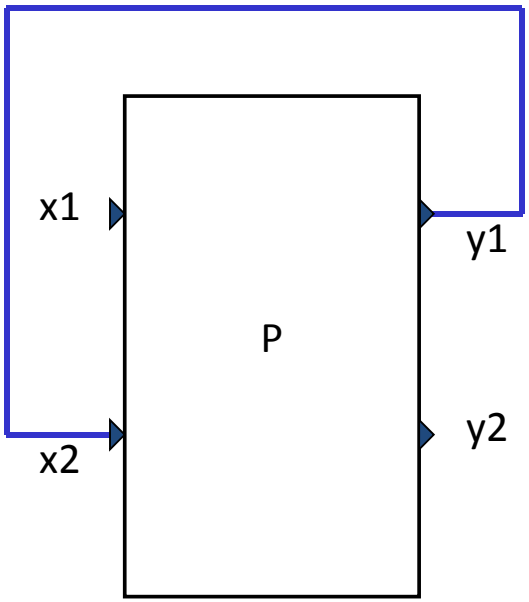



```
P.fire1( x1 ) returns y1 ;

P.fire2( x2 ) returns y2 ;
```

19

Non-monolithic interface

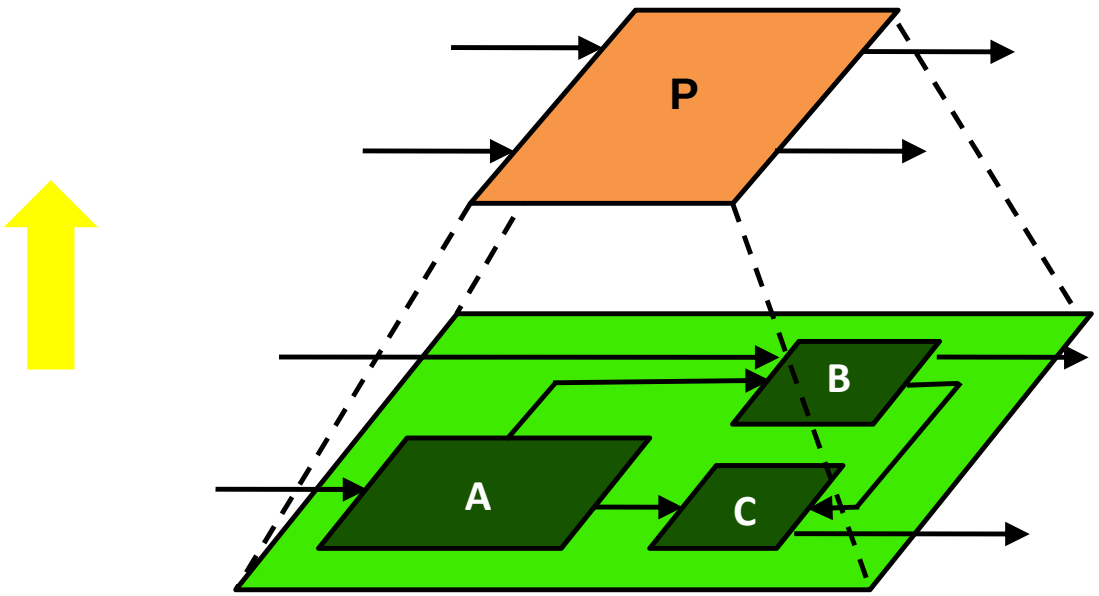
interface does not restrict usage




```
P.fire1( x1 ) returns y1 ;

↓

P.fire2( x2 ) returns y2 ;
```

20

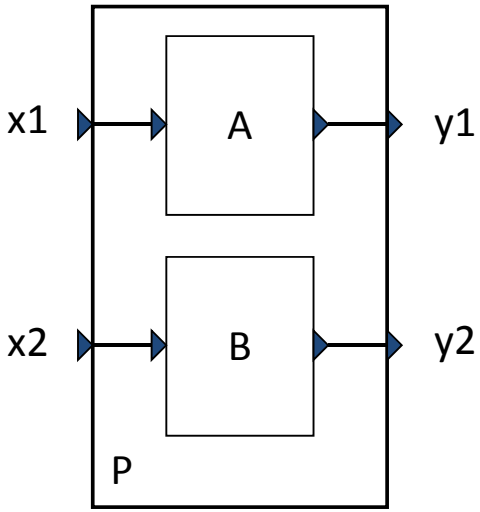
Bottom-up interface synthesis



Given interfaces for sub-blocks A, B, C, compute interface for composite block P.

21

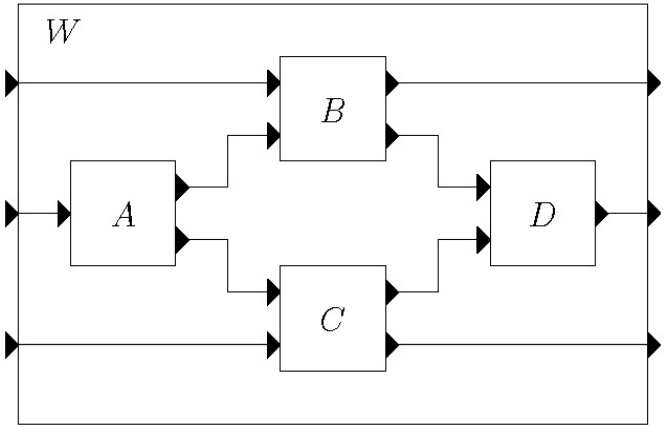
Simple non-monolithic interface



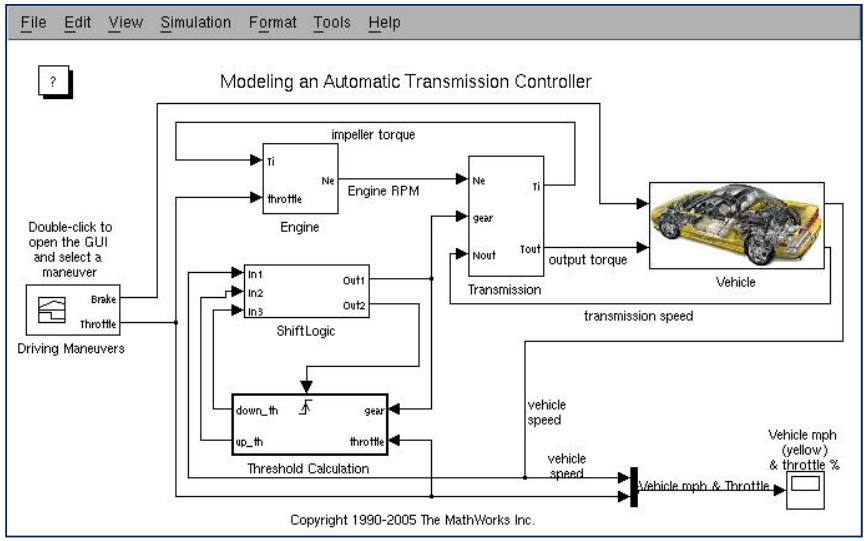

```
P.fire1( x1 ) returns y1 ;

P.fire2( x2 ) returns y2 ;
```

22

What about more complex diagrams?

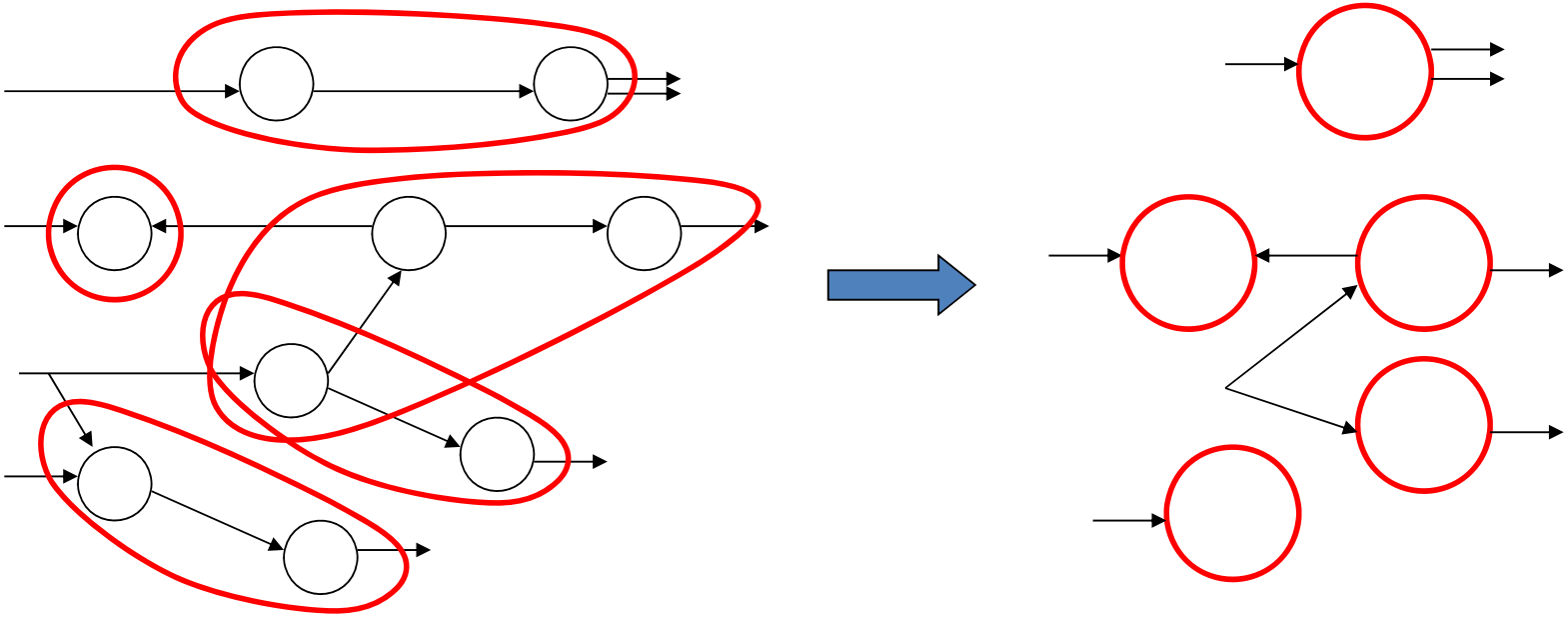


what about this?



or this?

Interface synthesis for block diagrams = graph clustering

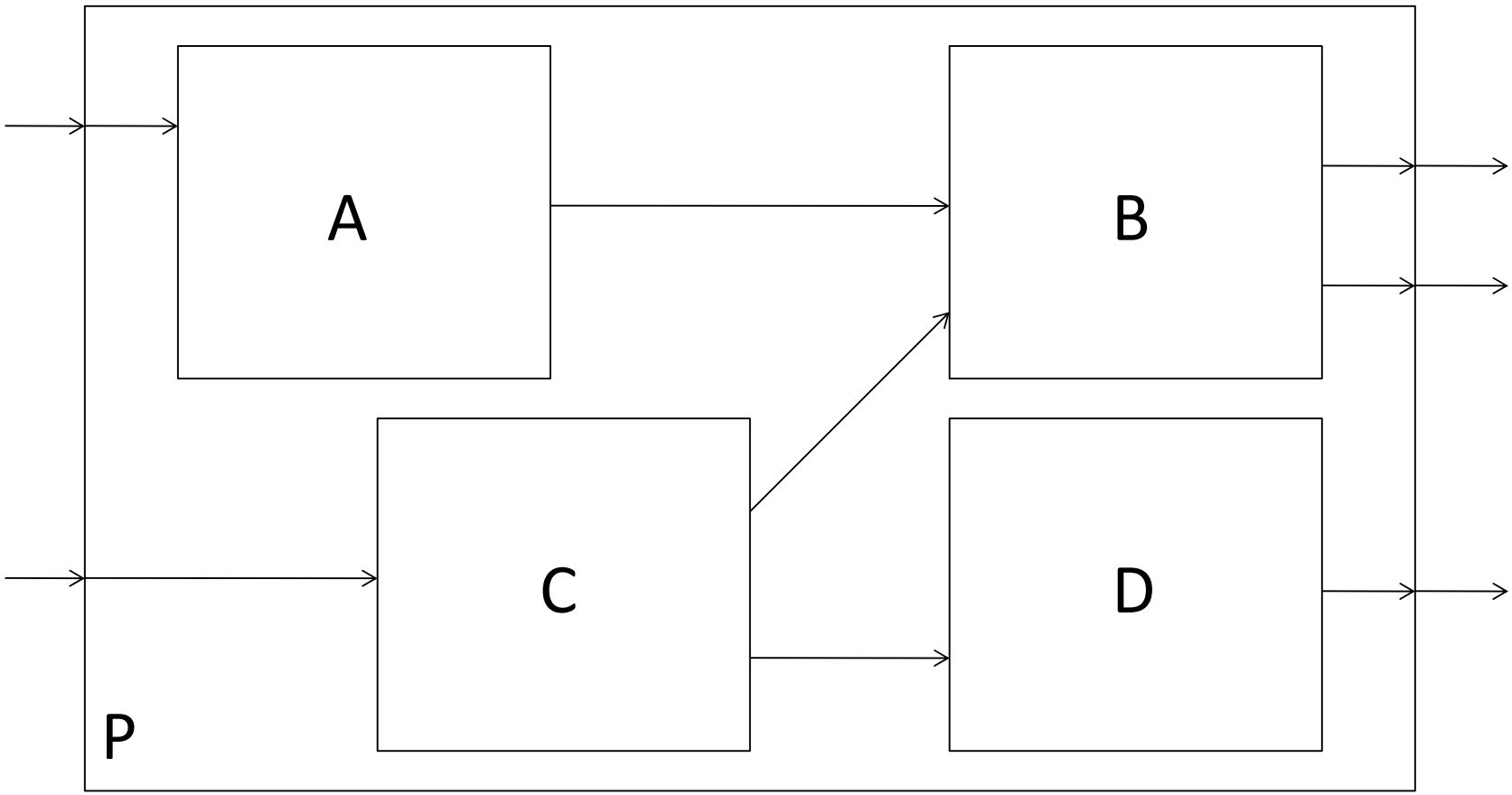


block diagram

interface

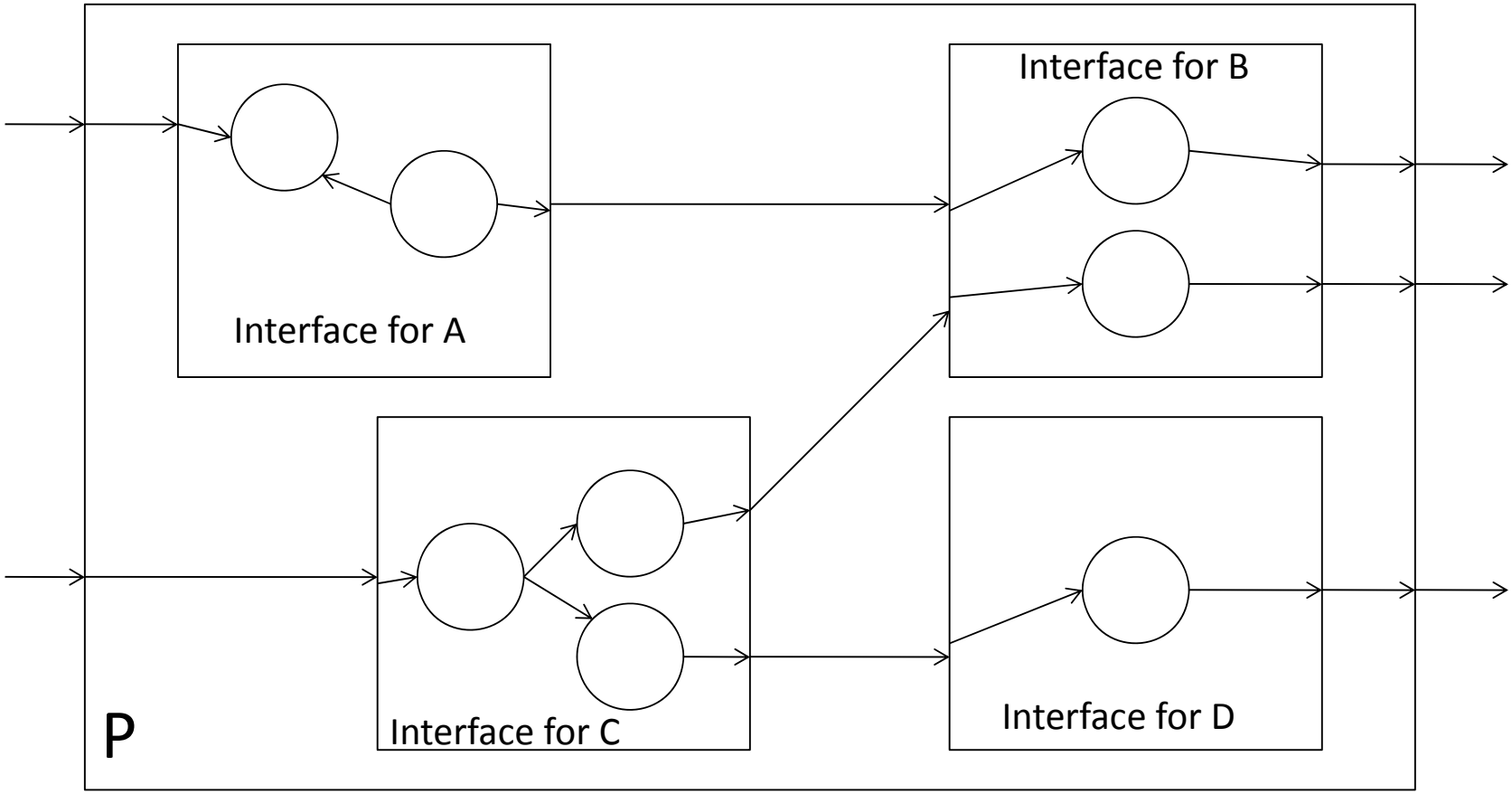
24

How it's done



26

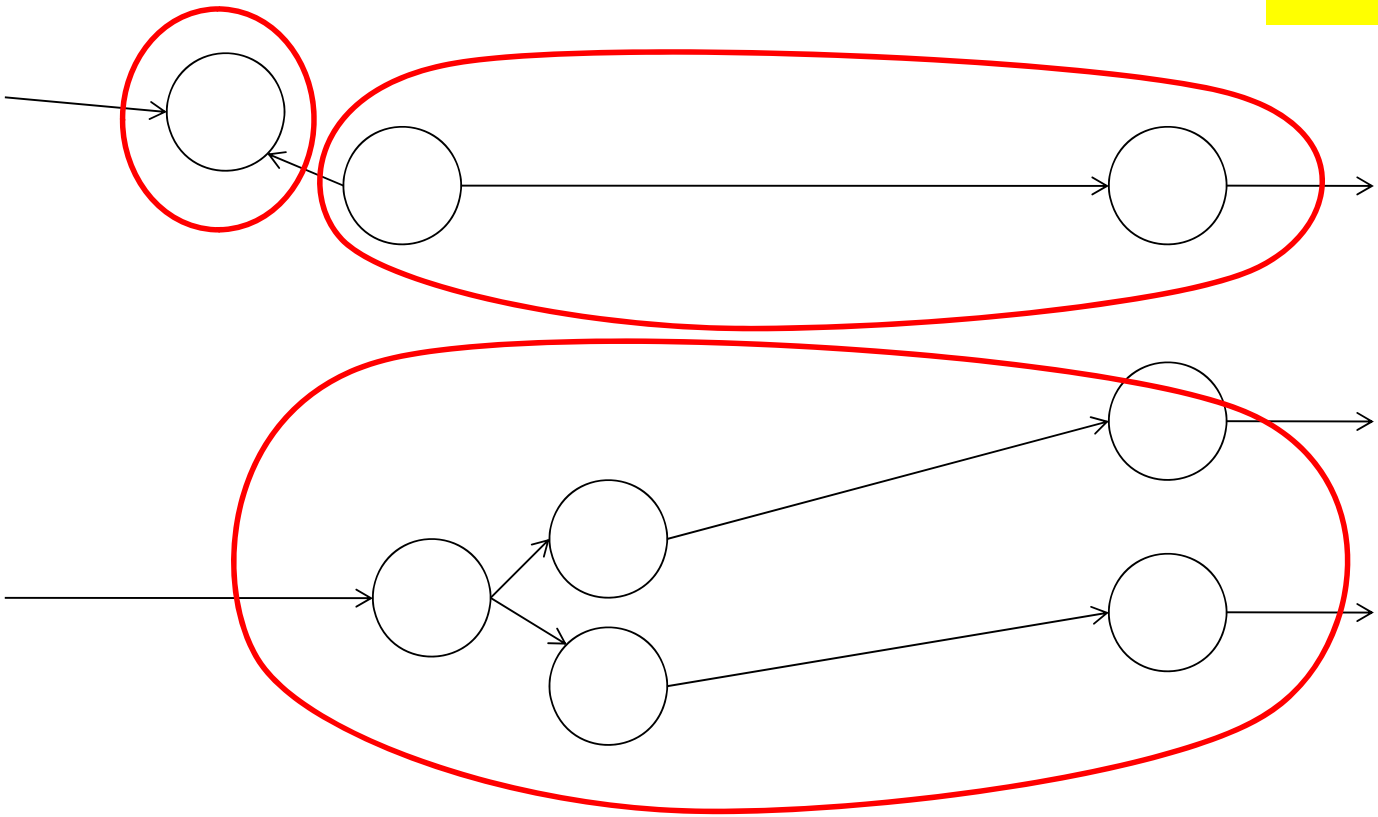
How it's done



27

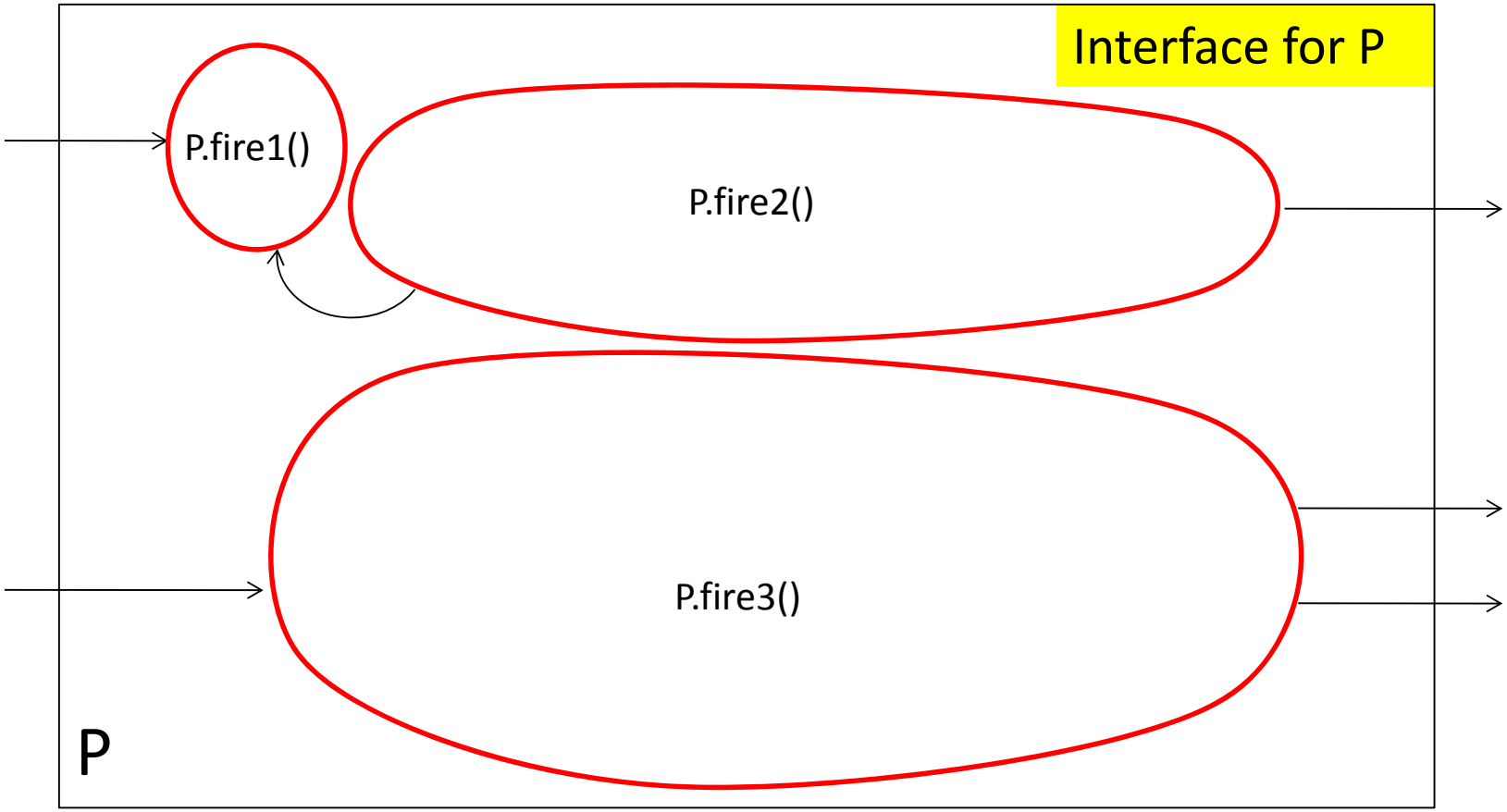
How it's done

clustering



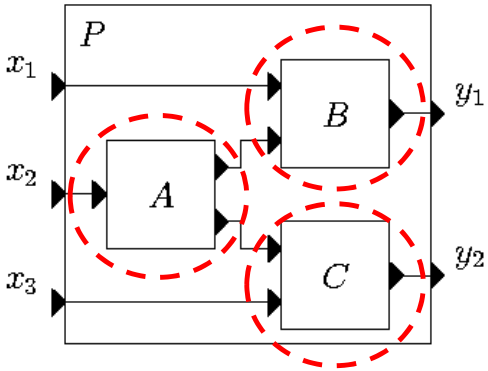
28

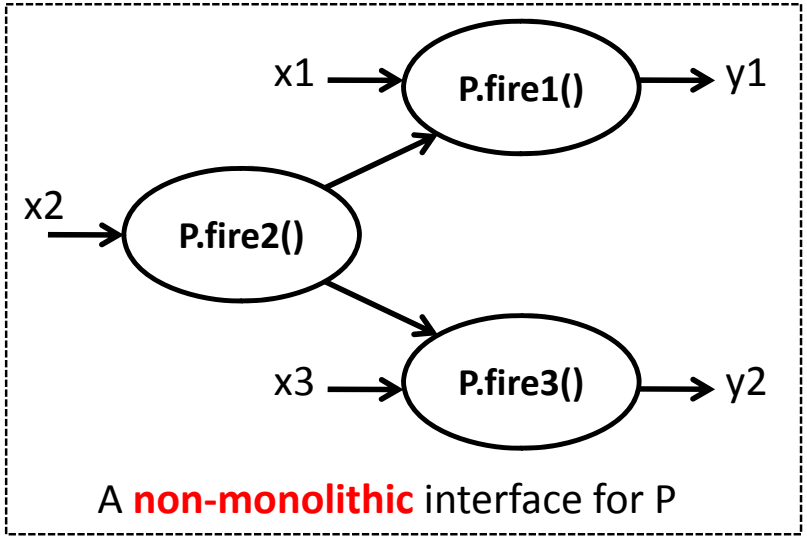
How it's done

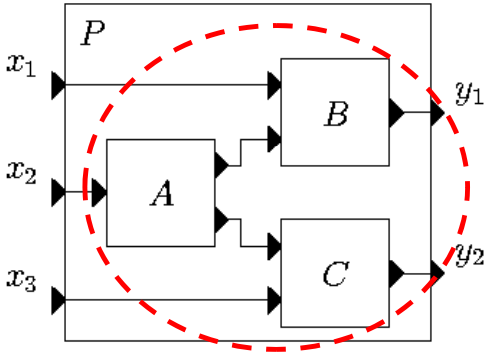


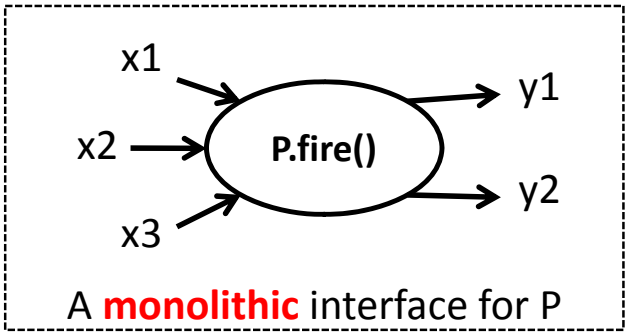
29

Different clusterings => different interfaces









trade-off: interface size vs. reusability

30

Different clustering algorithms = different tradeoffs

Clustering method	Complexity	Achieves maximal reusability?	Achieves minimal interf. size?	Modularity bound?	Achieves minimal code size?
“step-get”	Polynomial	No	Almost	<=2 functions	Yes
“dynamic”	Polynomial	Yes	Yes	<=N+1 functions*	No
“disjoint”	NP-complete	Yes	Yes	?	Yes
“greedy”	Polynomial	Yes	No	?	Yes

* N = number of block outputs

31

Bibliography

Lublinerman, R., Szegedy, C., and Tripakis, S. (2009).
Modular Code Generation from Synchronous Block Diagrams – Modularity vs. Code Size.
In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'09)*, pages 78–89. ACM.

Lublinerman, R. and Tripakis, S. (2008).
Modularity vs. Reusability: Code Generation from Synchronous Block Diagrams.
In *Design, Automation, and Test in Europe (DATE'08)*, pages 1504–1509. ACM.