

Causal Block Diagrams (CBDs) a family of formalisms

Hans Vangheluwe

Physical Systems Modelling

- Problem-Specific (technological)
- Domain-Specific (e.g., translational mechanical)
- (general) Laws of Physics
- Power Flow/Bond Graphs (physical: energy/power)
- Computationally a-causal
(Mathematical and Object-Oriented) ← **Modelica**
- Causal Block Diagrams (data flow)
- Numerical (Discrete) Approximations
- Computer Algorithmic + Numerical
(Floating Point vs. Fixed Point)
- As-Fast-As-Possible vs. Real-time (XiL)
- Hybrid (discrete-continuous) modelling/simulation
- Hiding IP: Composition of Functional Mockup Units (FMI)
- Dynamic Structure

Paulo Carreira · Vasco Amaral · Hans Vangheluwe
Editors

Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems



 **cost**
EUROPEAN COOPERATION
IN SCIENCE & TECHNOLOGY

 Springer Open

Gomes C., Denil J., Vangheluwe H. (2020) Causal-Block Diagrams: A Family of Languages for Causal Modelling of Cyber-Physical Systems.
In: Carreira P., Amaral V., Vangheluwe H. (eds) Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems. Springer, Cham.
https://doi.org/10.1007/978-3-030-43946-0_4

CBD CAUSAL BLOCK DIAGRAM

"COMPUTATIONAL CAUSALITY"

ABD A-CAUSAL BLOCK DIAGRAM

CAUSE $\longrightarrow$ CONSEQUENCE

$+: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$

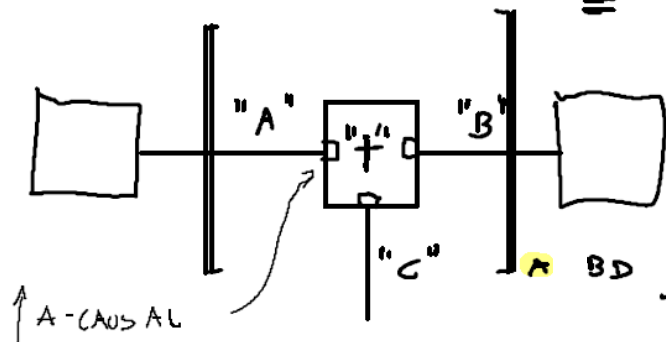
MATH

"WHAT"

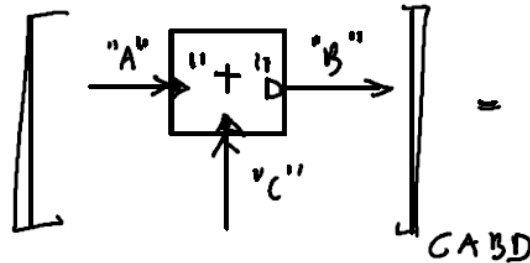
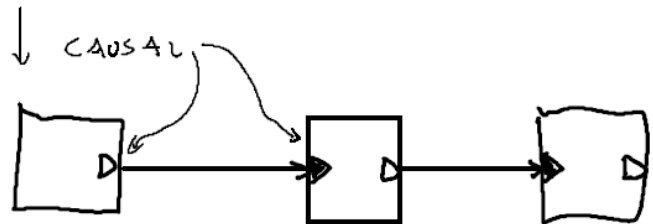
$$mVA + mVB + mVC = \phi$$

$$(mVA, mVB, mVC) \in \mathbb{R}^3$$

A-CAUSAL



CAUSALITY ASSIGNMENT

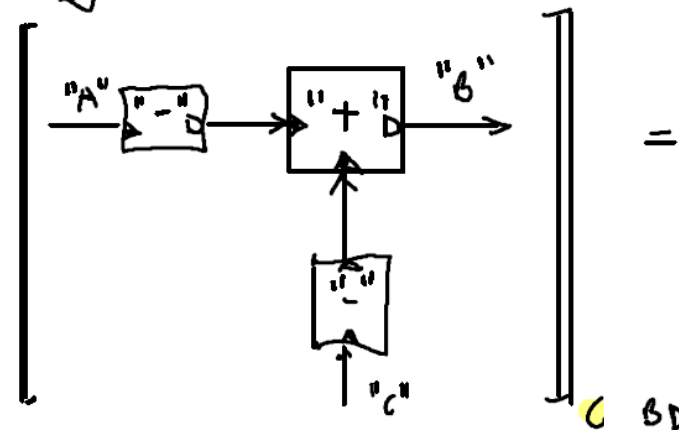
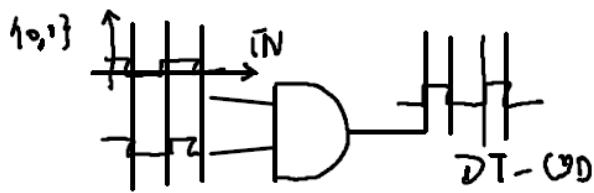


$vA, vB, vC : \text{float}$

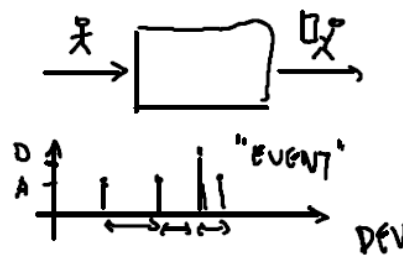
$vB := vA + vC$

CODE

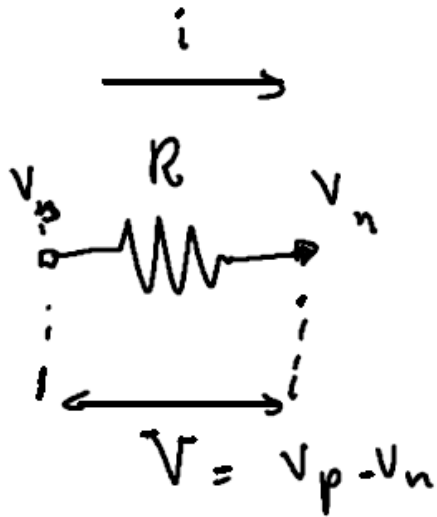
"How"



$vB := -vA - vC$



(computationally) a-causal vs. causal



$$V := R \times i$$

$$R := V / i$$

$$i := V / R$$

$$V - Ri = \phi$$

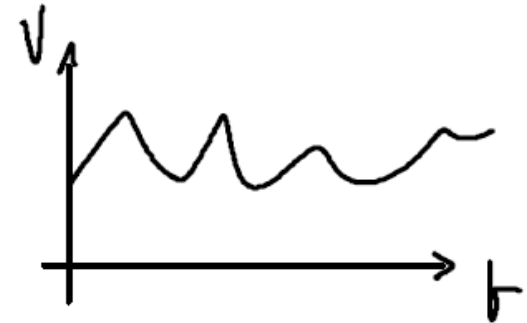


MODÉLICA

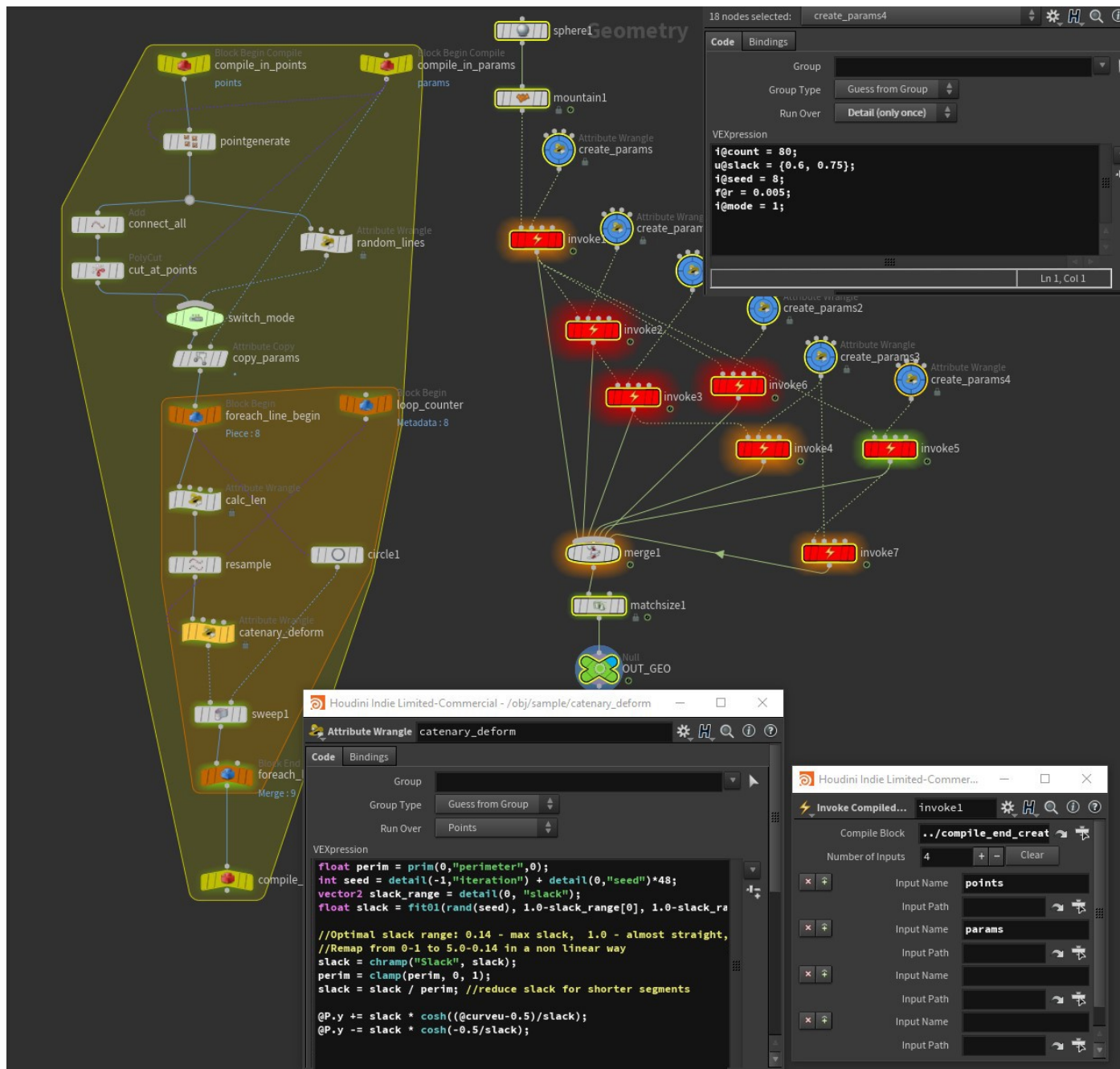


SIMULINK

CAUSALITY
ASGN.

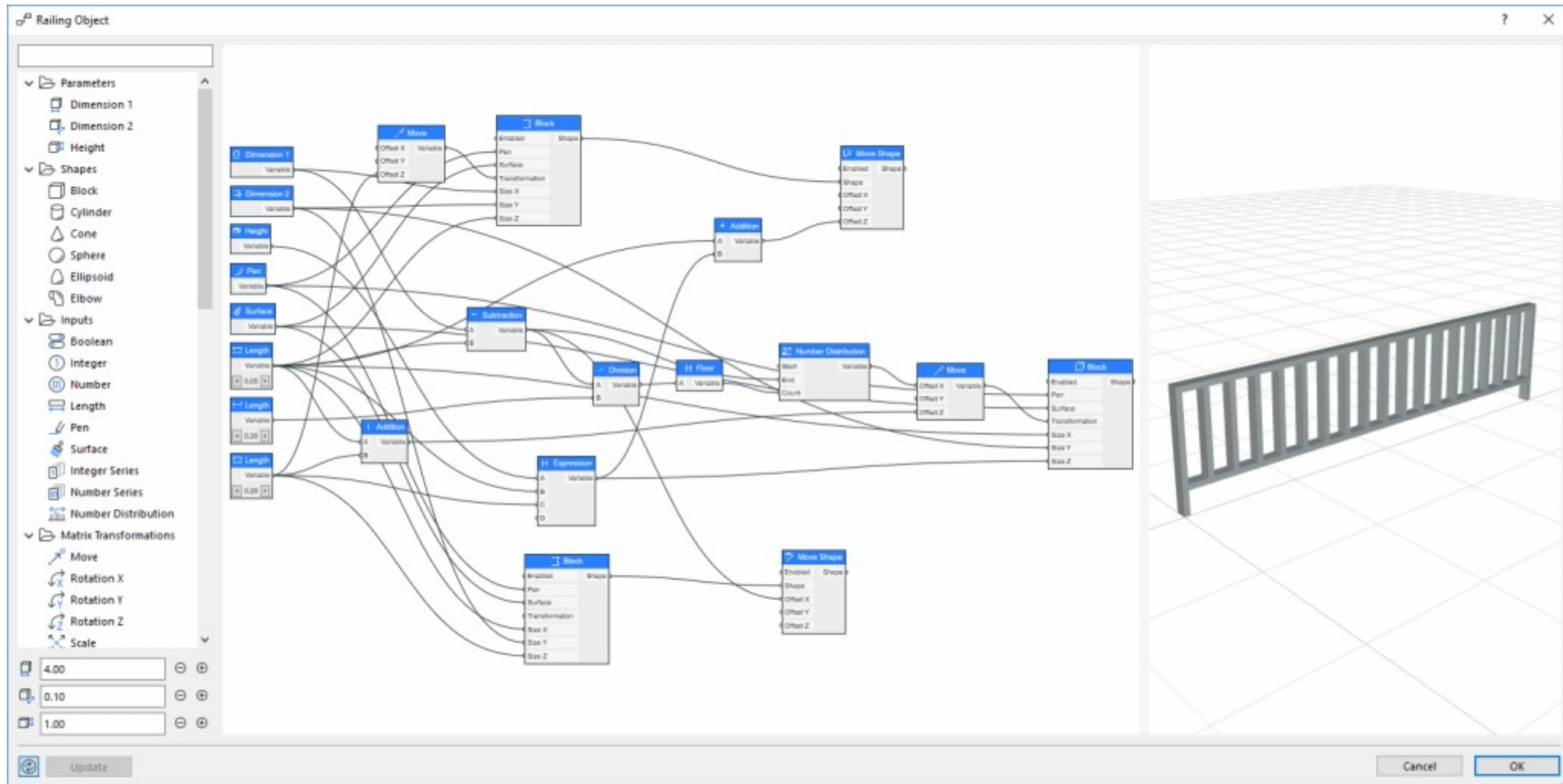


ALG-CBD



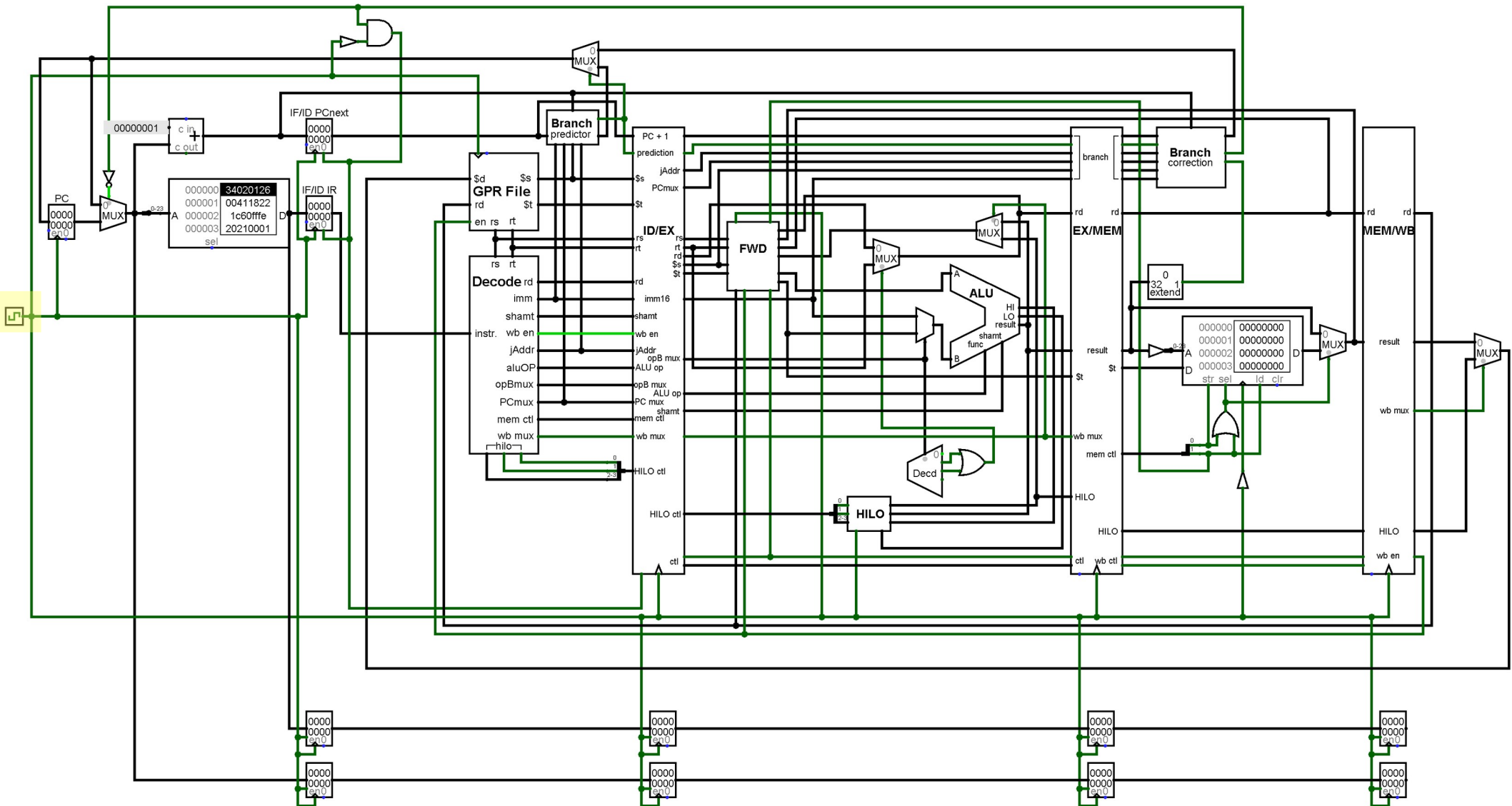
<https://www.rendereverything.com/houdini-compiled-blocks/>

ALG-CBD

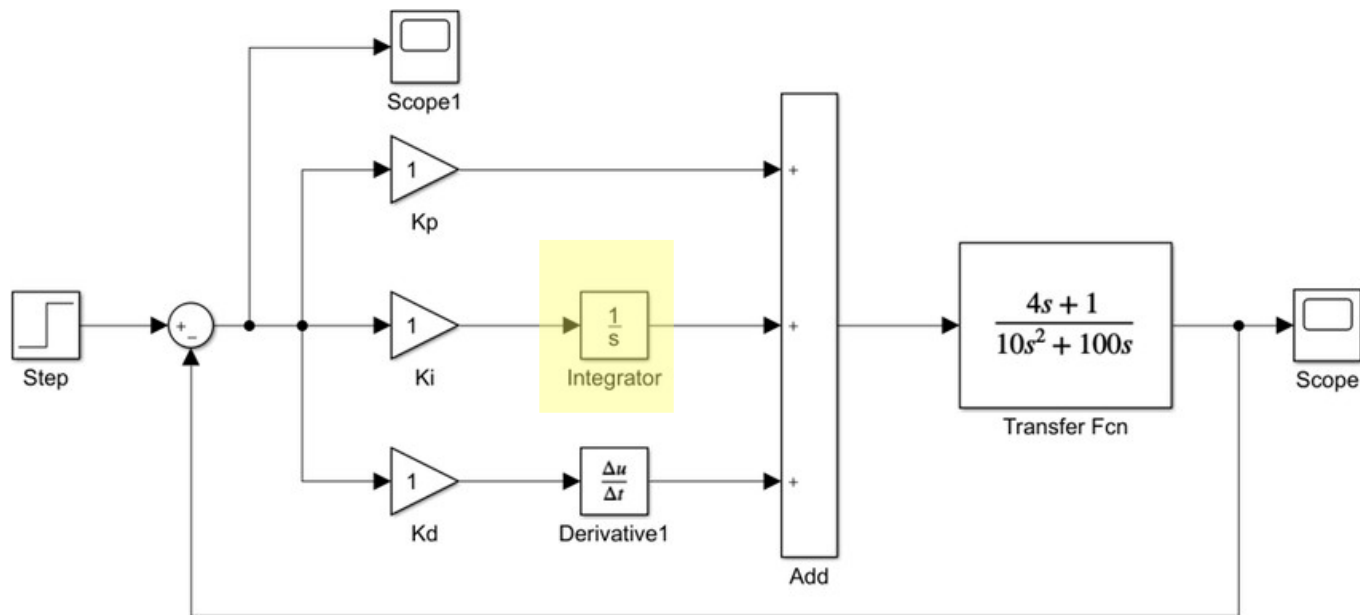


<https://graphisoft.vn/en/news/what-is-param-o/>

DT-CBD



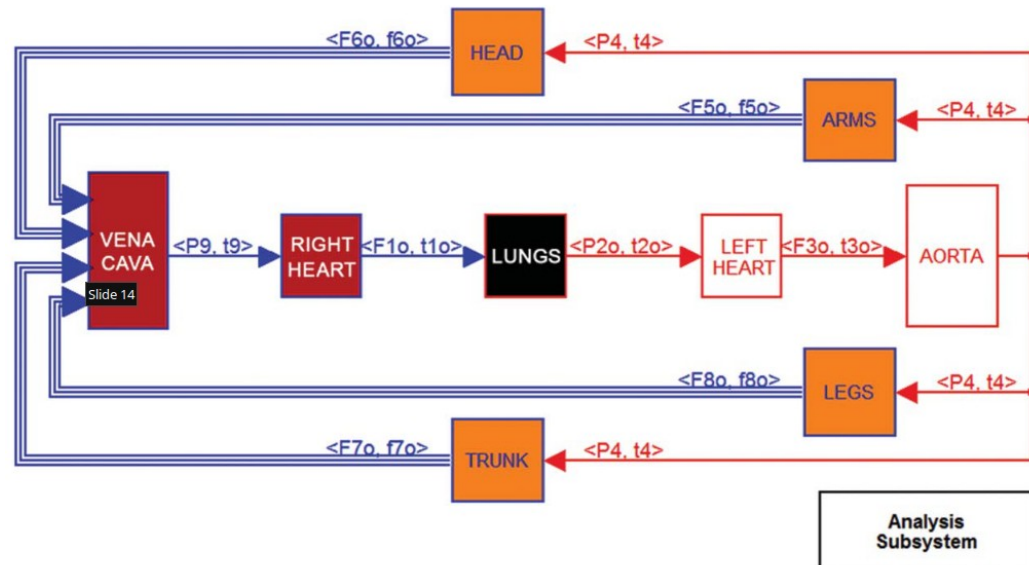
CT-CBD



PHYSBE The Circulatory System

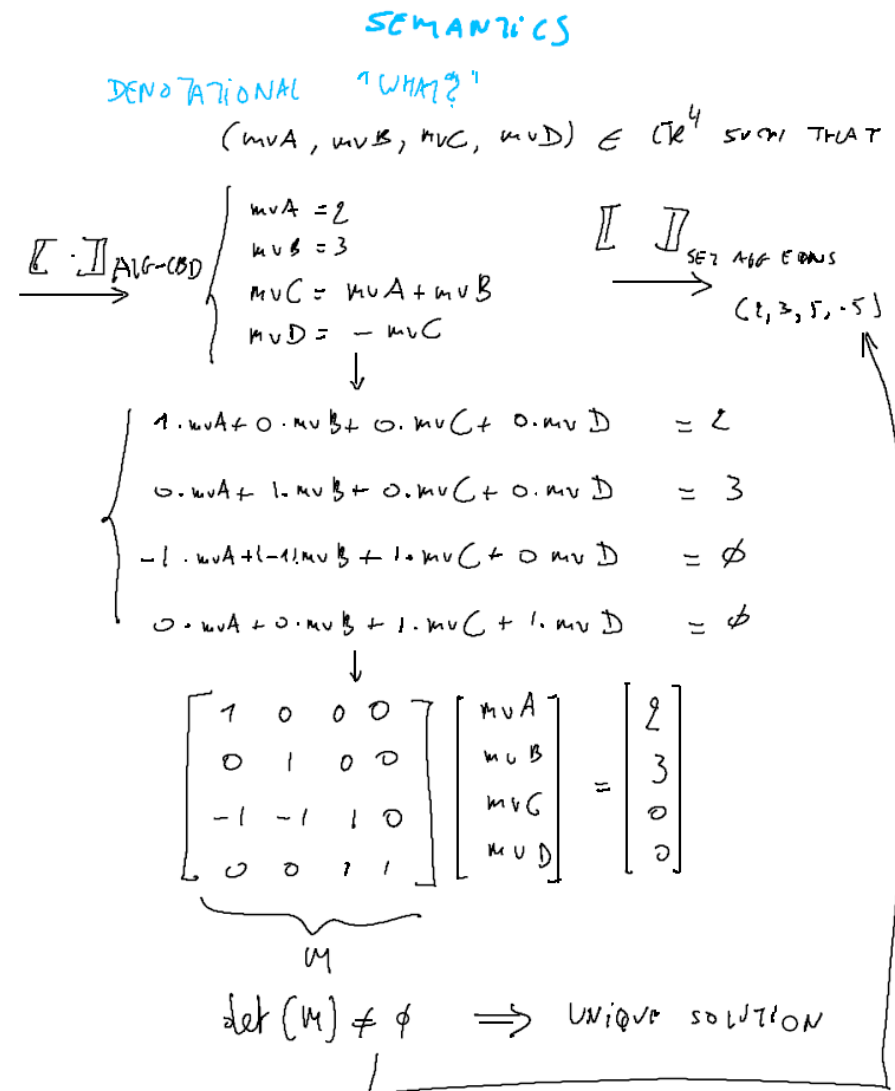
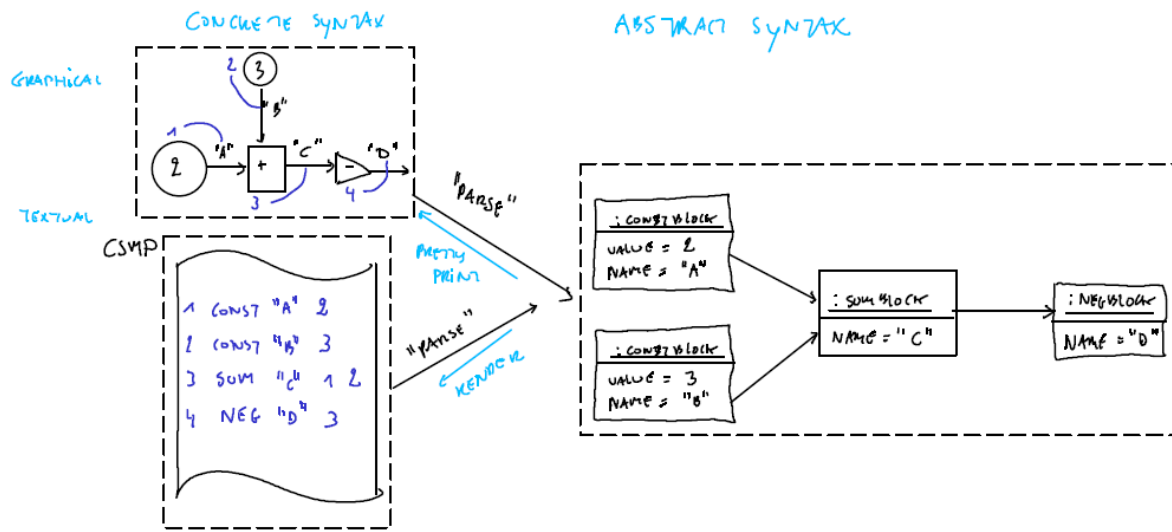
Scoped Variables

{F1i}	{P4}	{F6i}
{P2i}	{P9}	{F7i}
{F3i}	{F5i}	{F8i}



concrete/abstract syntax, semantic mapping/domain

-- denotational semantics



concrete/abstract syntax, semantic mapping/domain

-- operational semantics



OPERATIONAL SEMANTICS "How"

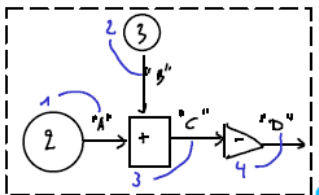
```

float vA := 0;
float vB := 0;
float vC := 0;
float vD := 0;
vA := 2;
vB := 3;
vC := vA + vB;
vD := -vC;
    
```

vD := -vC;
vA := 2;
vB := 3;
vC := vA + vB;



CONCRETE SYNTAX



ABSTRACT SYNTAX

SEMANTIC DOMAIN

SMP

```

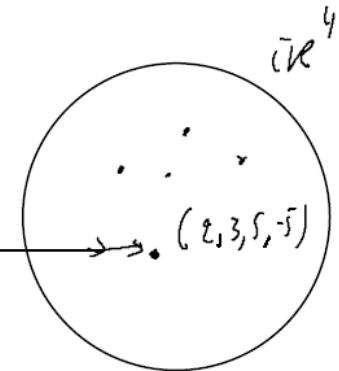
1 CONST "A" 2
2 CONST "B" 3
3 SUM "C" 1 2
4 NEG "D" 3
    
```

Alg-CBD

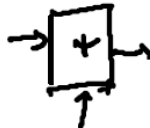
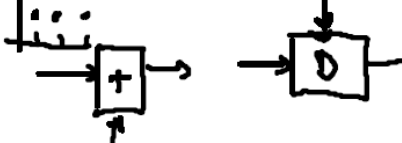
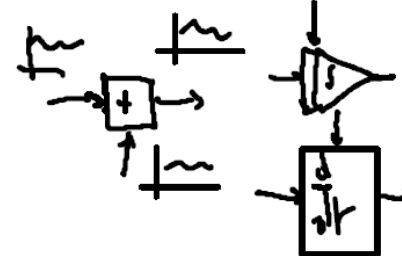
SEMANTIC MAPPING

$\llbracket \cdot \rrbracket$

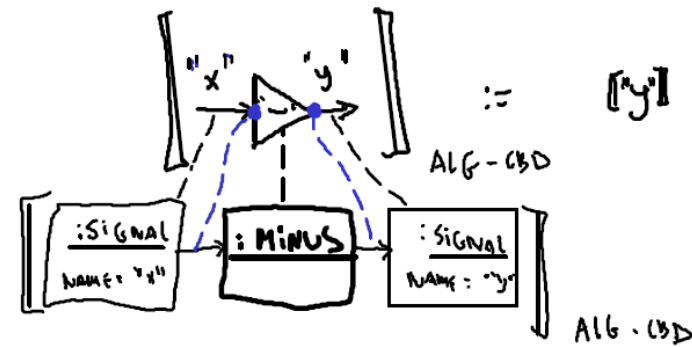
~ DENOTATIONAL
~ OPERATIONAL



a family of Causal Block Diagram (CBD) formalisms

TIME ↓	HIERARCHY ↗ FLAT CBD		SYNTAX ↗ ~~FLATTEN~~		SEMANTICS ↗ DENOTATIONAL "WHAT" OPERATIONAL "HOW"	
{NOW}	ALGEBRAIC (ALG-CBD)		NO LOOPS WITH LOOPS			
IN	DISCRETE-TIME (DT-CBD)					
IR	CONTINUOUS-TIME (CT-CBD)					

denotational semantics of individual blocks

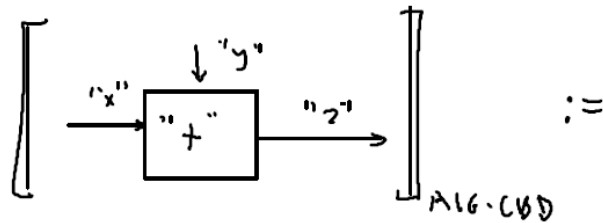


$$:= \llbracket 'y' \rrbracket = - \llbracket 'x' \rrbracket$$

$$mv_y = - mv_x$$

$$mv_x, mv_y \in \mathbb{R}$$

$$- : \mathbb{R} \rightarrow \mathbb{R}$$



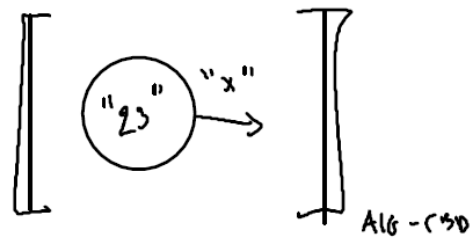
$$:=$$

$$\llbracket 'z' \rrbracket = \llbracket 'x' \rrbracket + \llbracket 'y' \rrbracket$$

$$mv_z = mv_x + mv_y$$

$$mv_x, mv_y, mv_z \in \mathbb{R}$$

$$+ : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

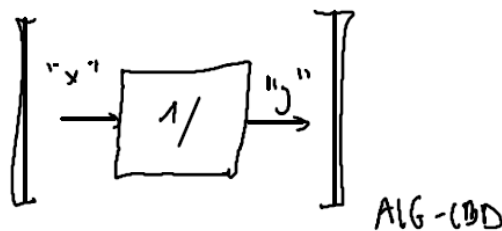


$$:=$$

$$\llbracket 'x' \rrbracket = \llbracket '23' \rrbracket$$

$$mv_x = 23$$

$$mv_x \in \mathbb{R}$$



$$:=$$

$$\llbracket 'y' \rrbracket = 1 / \llbracket 'x' \rrbracket$$

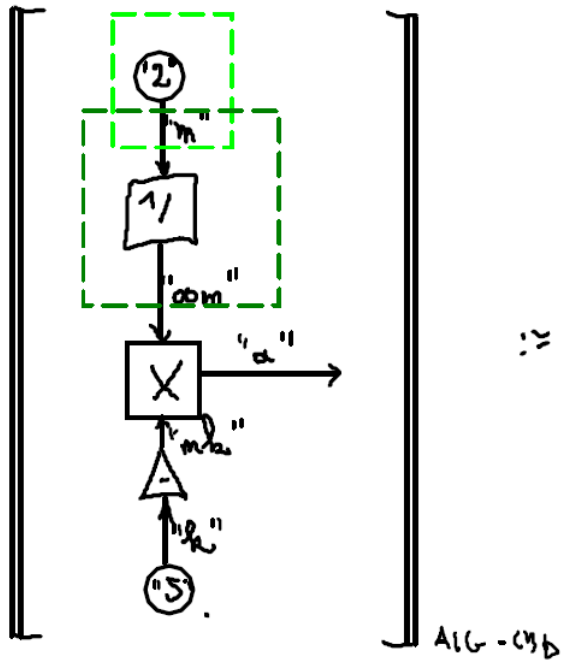
$$mv_y = 1 / mv_x$$

$$mv_x \in \mathbb{R} \setminus \{0\}, mv_y \in \mathbb{R}$$

$$mv_x, mv_y \in$$

$$\mathbb{R} \cup \{\text{exception}\}$$

denotational semantics of "composition"



$\approx$

$mvm, mvoom, mvu, mva, mvuh, mvle \in \mathbb{R}$

such that

$mvm \neq 0$

$$mvm = 2$$

$$mvoom = 1/mvm$$

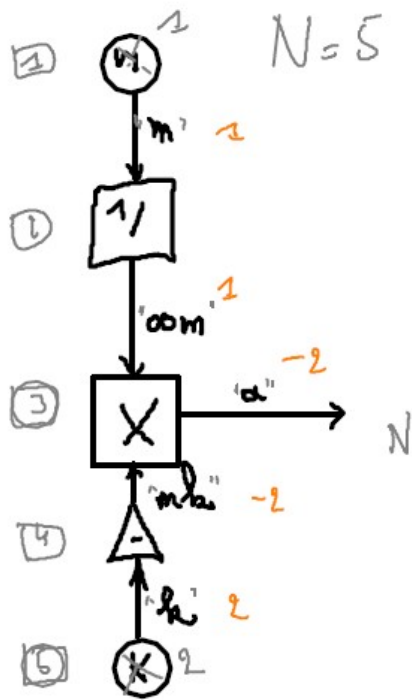
$$mva = mvoom \times mvuh$$

$$mvuh = -mvle$$

$$mvle = 5$$

$$= (2, 0.5, 2.5, -5, 5) \in \mathbb{R}^5$$

operational semantics naive/optimal approach



	v_m	v_{om}	v_a	$v_{m'h}$	v_h
1	UK	UK	UK	UK	UK
2	1	UK	UK	UK	UK
3	1	1	UK	UK	UK
4	1	1	UK	UK	UK
5	1	1	UK	UK	2
1	1	1	UK	UK	2
2	1	1	UK	UK	2
3	1	1	UK	UK	2
4	1	1	UK	-2	2
5	1	1	UK	-2	2
1	1	1	UK	-2	2
2	1	1	UK	-2	2
3	1	1	-2	-2	2
4					
5					

2 2 ... 5

"SCHEDULE"



Worst-case

iter

$\mathcal{O}(N) \mathcal{O}(1)$

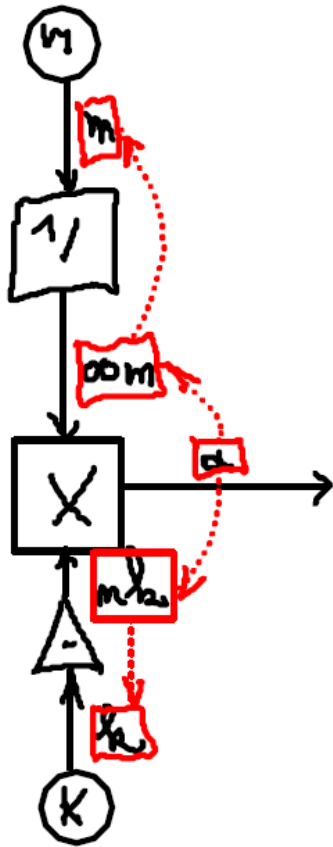
comp

$\mathcal{O}(N^2) \mathcal{O}(N)$

Disrupted



operational semantics scheduling



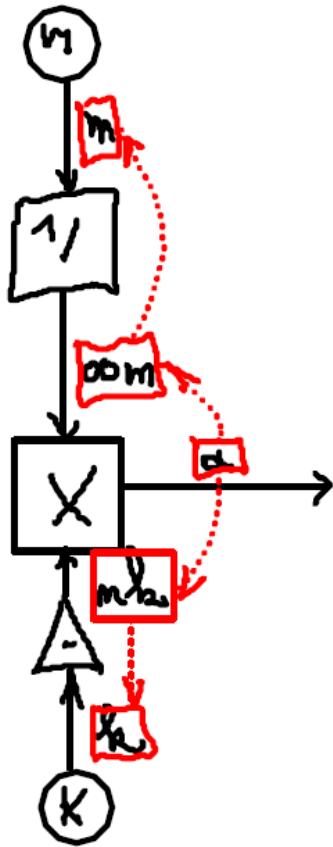
DEPENDENCY GRAPH



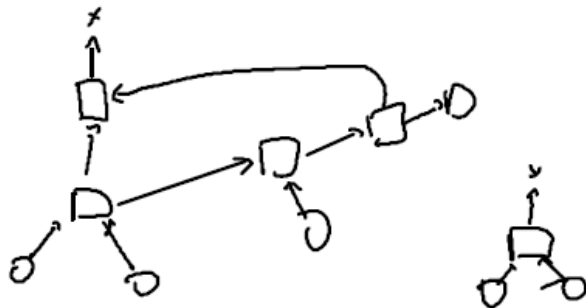
schedule = [m, oom, k, mk, a]

← 1 2 3 4 5

operational semantics scheduling


$$\text{schedule} = [\text{m}, \text{oom}, \text{k}, \text{mk}, \text{a}]$$

← 1 2 3 4 5



operational semantics

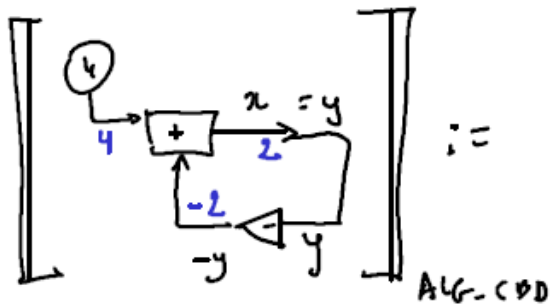
$$\mathcal{O}(N)$$

```
depGraph = buildDepGraph(CBD)
schedule = topologicalSort(depGraph)
```

$$O(N)$$

```
for block in schedule:
    block.compute()
```

$$\mathcal{O}(N)$$



$$\begin{cases} x + y = 4 \\ x - y = 0 \end{cases} \quad \begin{aligned} x &= 4 - y \\ y &= x \end{aligned}$$

SET LINEAR EQNS.

2 UNKNOWN

2 EQNS

$$\begin{cases} 1x + 1y = 4 \\ 1x - 1y = 0 \end{cases}$$

operational semantics
cyclic dependency
aka algebraic loop

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$$

$$\det \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = -2 \neq 0$$

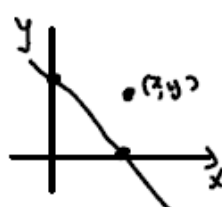
$$x = \frac{\begin{vmatrix} 4 & 1 \\ 0 & -1 \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix}} = \frac{-4}{-2} = 2$$

$$y = \frac{\begin{vmatrix} 1 & 4 \\ 1 & 0 \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix}} = \frac{-4}{-2} = 2$$

**under-determined
set of equations**

$$\begin{cases} 2x + 2y = 4 \\ x + y = 2 \end{cases}$$

$y = 2 - x$



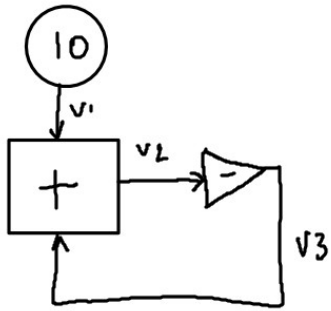
→ **solution is
sub-space**

$$\begin{bmatrix} 2 & 2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

$$\begin{vmatrix} 2 & 2 \\ 1 & 1 \end{vmatrix} = 0$$

**over-determined
set of equations**

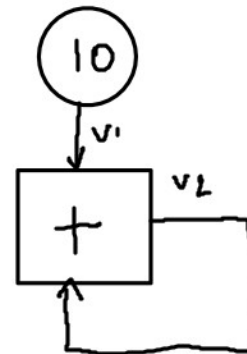
→ **no solution**



$$\begin{cases} v_1 = 10 \\ v_2 = v_1 + v_3 \\ v_3 = -v_2 \end{cases}$$

$$\begin{cases} v_1 = 10 \\ v_2 = v_1 - v_2 \end{cases}$$

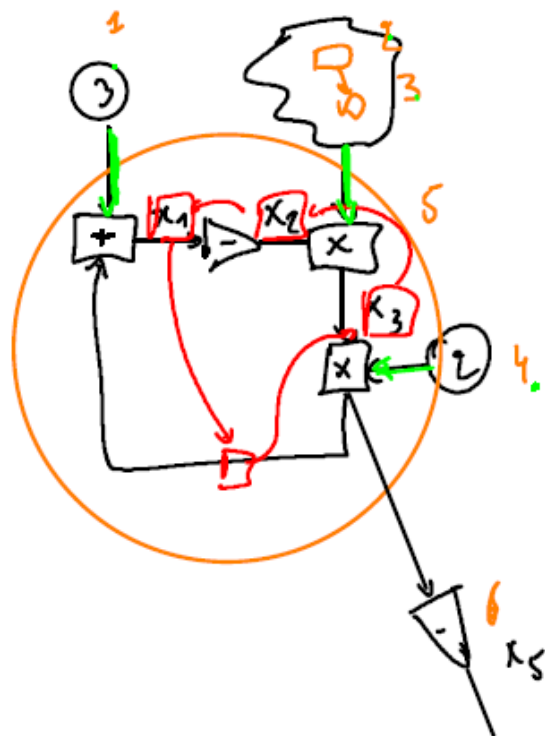
$$\begin{aligned} v_2 &= 10 - v_2 \\ v_2 &= 5 \end{aligned}$$



$$\begin{cases} v_1 = 10 \\ v_2 = v_1 + v_2 \end{cases}$$

$$\begin{aligned} v_2 &= 10 + v_2 \\ \rightarrow 10 &= \emptyset? \end{aligned}$$

“solving”
a linear loop



$$\begin{aligned} h_{11}x_1 + h_{12}x_2 + \dots + h_{1n}x_n &= k_1 \\ h_{21}x_1 + h_{22}x_2 + \dots + h_{2n}x_n &= k_2 \\ &\vdots \\ h_{n1}x_1 + h_{n2}x_2 + \dots + h_{nn}x_n &= k_n \end{aligned}$$

$$\begin{bmatrix} h_{11} & \dots & h_{1n} \\ \vdots & & \vdots \\ h_{n1} & \dots & h_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} k_1 \\ \vdots \\ k_n \end{bmatrix}$$

invertible? $\det(K) \neq 0$

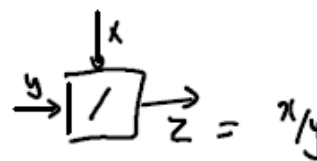
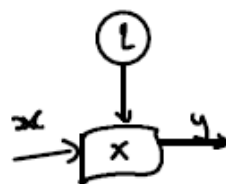
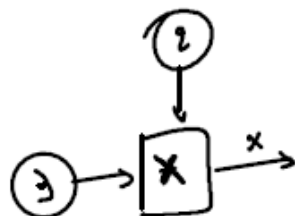
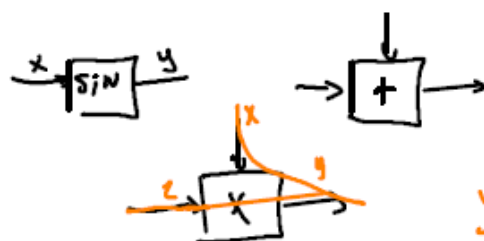
$(x_1, \dots, x_n)$

GAUSSIAN elim $\mathcal{O}(n^3)$

$$y = x \times z$$

$$xy - x \times z = 0$$

BLOCKS IN “LINEAR LOOP”



“solving”
a linear loop

OMEdit - Simulation Setup - harmonicEquation

Simulation Setup - harmonicEquation

General Interactive Simulation Translation Flags **Simulation Flags** Output

Model Setup File (Optional): Browse...

Initialization Method (Optional):

Equation System Initialization File (Optional): Browse...

Equation System Initialization Time (Optional):

Clock (Optional):

Linear Solver (Optional):

Non Linear Solver (Optional):

Linearization Time (Optional):

Output Variables (Optional):

☐ Save experiment annotation inside model i.e., __OpenModel

☐ Save translation flags inside model i.e., __OpenModel

☐ Save simulation flags inside model i.e., __OpenModel

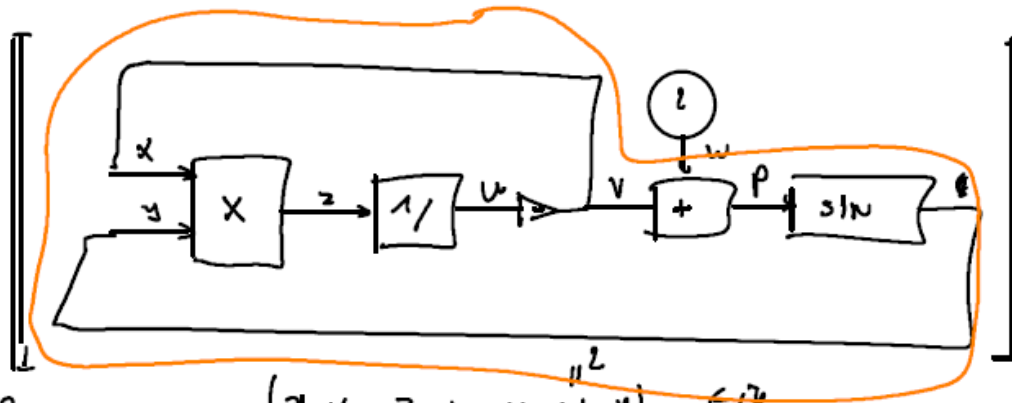
☒ Simulate

lapack
lis
klu
umfpack
totalpivot
default

method using LAPACK LU factorization

OK Cancel

non-linear loop



UNIQUE?

COMPLEXITY? $\mathcal{O}(\dots)$

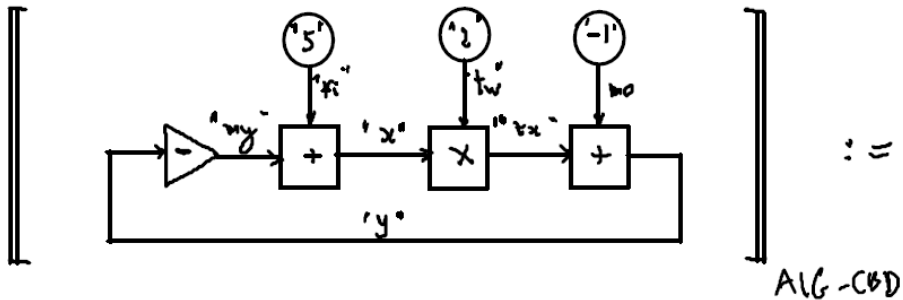
$$(x, y, z, u, v, w, p) \in \mathbb{R}^7$$

$$\begin{aligned} z \times y - z &= 0 \\ -1/w &= 0 \end{aligned} \rightarrow \begin{pmatrix} \epsilon_1^2 \\ \epsilon_1^1 \\ \epsilon_2^1 \\ \vdots \end{pmatrix} \rightarrow \phi$$

$$\begin{aligned} (x_0, y_0, z_0, u_0, v_0, w_0, p_0) \\ \downarrow \\ (x_1, y_1, z_1, u_1, v_1, w_1, p_1) \end{aligned}$$

how to find an algebraic loop?

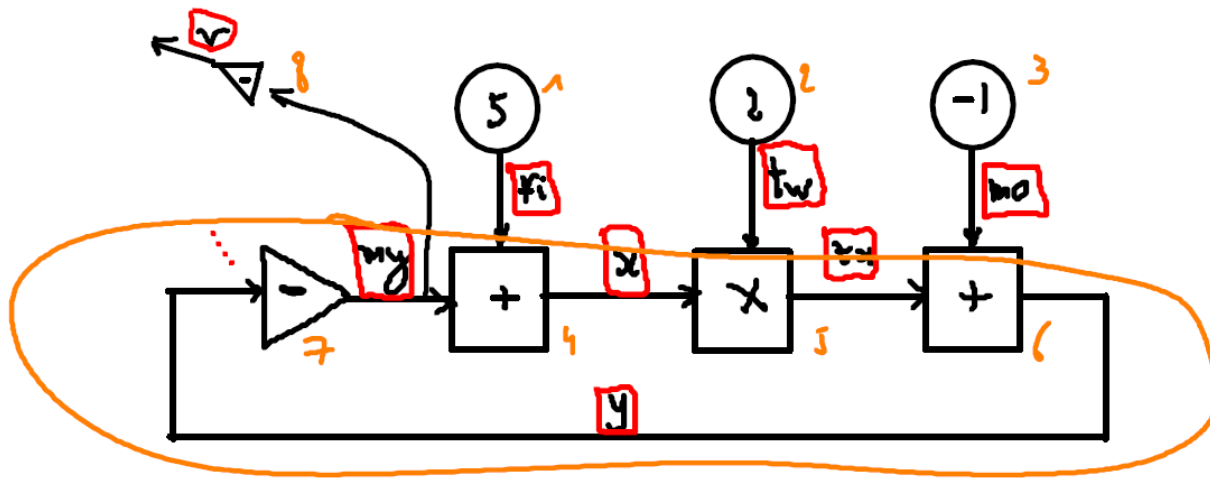
"algebraic loop"



$mvfi, mvx, mvw, mvtx, mvmo, mvy, mvny \in \mathbb{R}$
such that

$$\begin{cases} mvfi = 5 \\ mvw = mvfi + mvny \\ mvw = 2 \\ mvtx = mvw \times mvx \\ mvmo = -1 \\ mvy = mvmo + mvx \\ mvny = -mvy \end{cases} \quad \approx \quad (5, 2, 2, 4, -1, 3, -3) \in \mathbb{R}^7$$

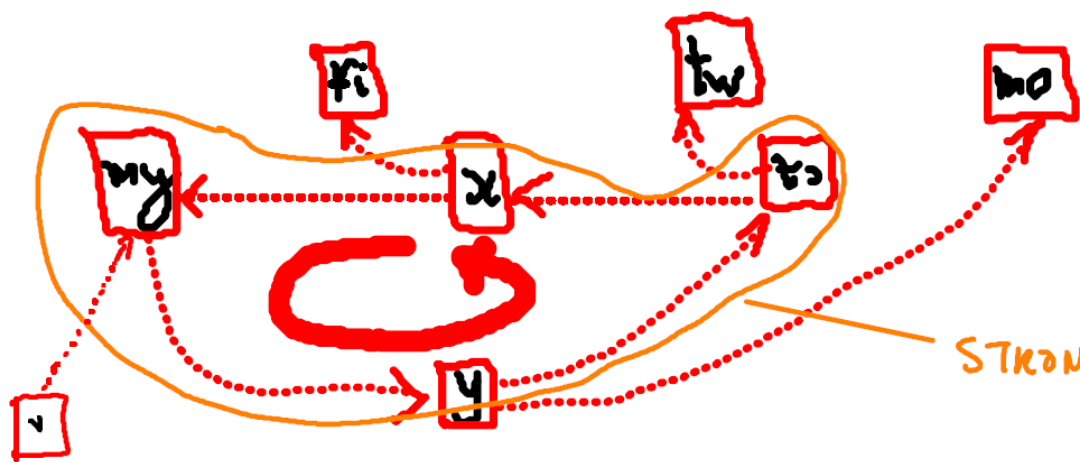
how to find an algebraic loop?



$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{matrix} 1 & 2 & 3 \\ \leftarrow & & \end{matrix} \quad \{4, 5, 6, 7\} \quad 8$$

$$[\{1\}, \{2\}, \{3\}, \{4, 5, 6, 7\}, \{8\}]$$



STRONG COMPONENT

TARJAN 1974

$$O(N+E)$$

operational semantics

```
schedule = topologicalSortAndLoopDetect(depGraph(CBD))
```

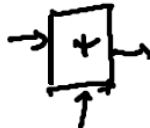
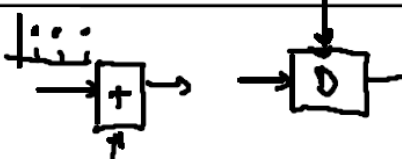
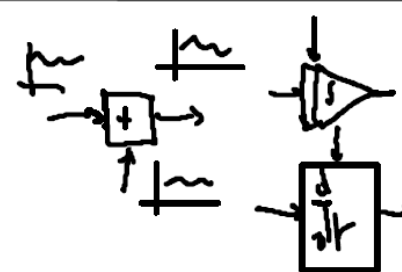
$\Theta(N+E)$

```
for genBlock in schedule:  
    genBlock.compute()
```

$\Theta(N)$?

SOME SETS OF COUPLED EQNS (SIZE m)
SOLVER COMPLEXITY $\leq \Theta(m^3)$
USUALLY $m \ll N$

a family of Causal Block Diagram (CBD) formalisms

TIME ↓	FLAT	HIERARCHY ↗	SYNTAX ↗	SEMANTICS ↗	
	CBD	✓ FLATTEN		DENOTATIONAL "WHAT"	OPERATIONAL "HOW"
{NOW}	ALGEBRAIC (ALG-CBD)		NO LOOPS WITH LOOPS	✓ ✓	✓ ✓
IN	DISCRETE-TIME (DT-CBD)				
IR	CONTINUOUS-TIME (CT-CBD)				

DISCRETE-TIME CBD (DT-CBD)

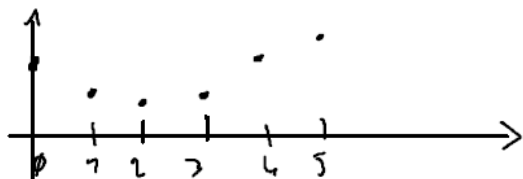
$$\left[\begin{array}{c} \text{"x"} \\ \rightarrow \text{ } \triangle \text{ } \rightarrow \text{"y"} \\ \downarrow \text{ } \uparrow \\ \text{ } \end{array} \right] := \llbracket y \rrbracket = - \llbracket x \rrbracket \quad \text{my}(i) = -\text{mx}(i), \forall i \in \mathbb{N}_\phi$$

$$\text{mx}, \text{my} \in \mathbb{N}_\phi \rightarrow \mathbb{R}$$

$$- : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

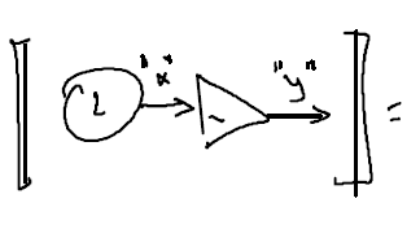
$$\begin{array}{l} x_i \\ x(i) \end{array} \quad i \in \mathbb{N}_\phi$$

mx



$$\text{mx} = \{ \phi \rightarrow 1.5, 1 \rightarrow 0.5, 2 \rightarrow 0.5, \dots \}$$

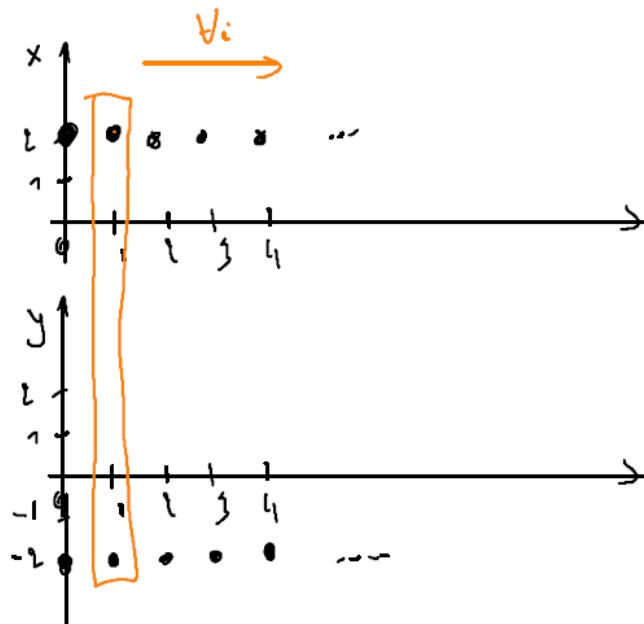
$$\left[(\phi, 1.5), (1, 0.5), (2, 0.5), \dots \right]$$

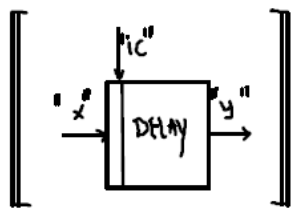


$$x(i) = 2$$

$$y(i) = -x(i)$$

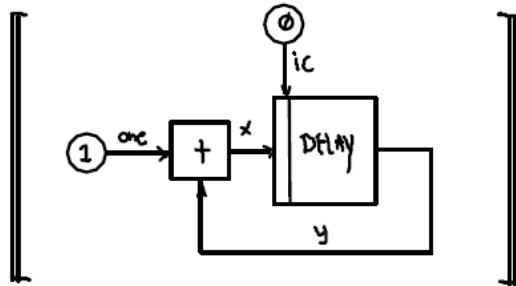
$$\forall i \in \mathbb{N}_\phi$$





$$:= \begin{cases} r_y(i) = r_z(i-1), & \forall i \in \mathbb{N} \\ r_y(\phi) = r_z(\phi) \end{cases}$$

$$v_L, v_y, v_x, v_{one} \in \mathbb{N}_\phi \rightarrow \mathbb{R}$$



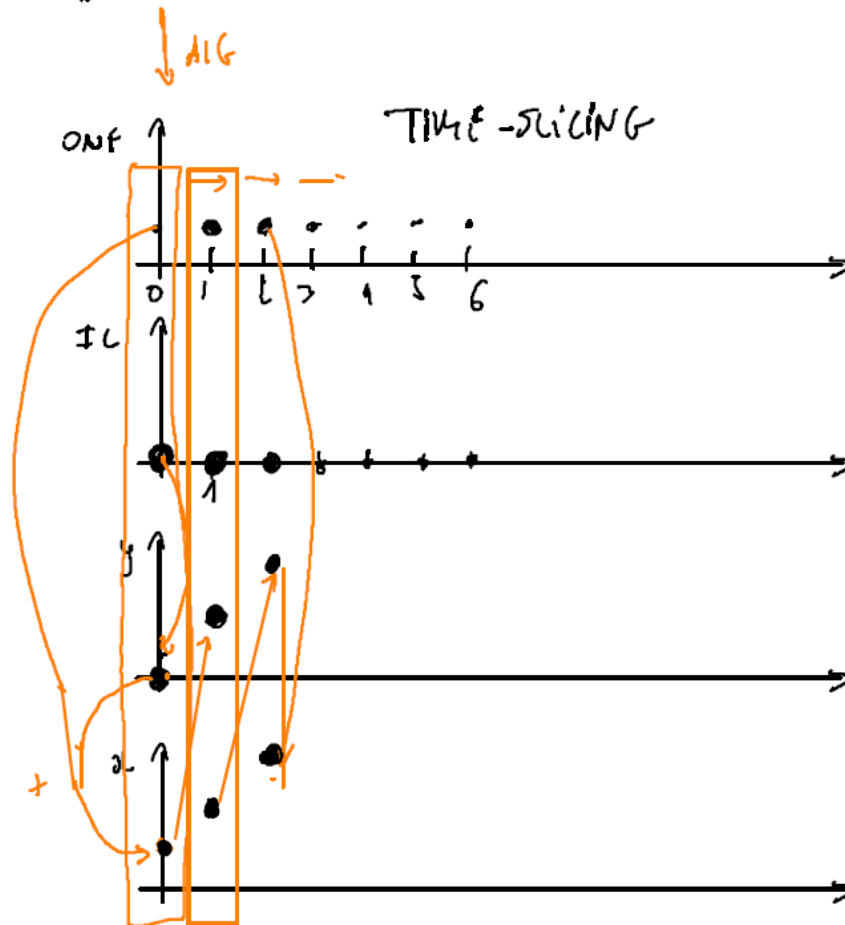
:=

$$\begin{cases} v_L(i) = \phi, & \forall i \in \mathbb{N}_\phi \\ r_y(i) = r_z(i-1), & \forall i \in \mathbb{N} \\ r_y(\phi) = ic(\phi) \\ v_x(i) = v_{one}(i) + r_y(i), & \forall i \in \mathbb{N}_\phi \\ v_{one}(i) = 1, & \forall i \in \mathbb{N}_\phi \end{cases}$$

$$= ([0, 0, 0, \dots], [0, 1, 1, \dots], [1, 2, 3, \dots], [1, 1, 1, \dots]) \in \mathbb{R}$$

"behaviour trace"
"state snapshot trace"

COUNTER



Operational Semantics of DT-CBD

```
i = 0
```

```
while (not end_condition(i, ...)):
```

```
    depGraph = buildDepGraph(CBD)
```

```
    schedule = loopDetectAndTopSort(depGraph)
```

```
    for gblock in schedule:
```

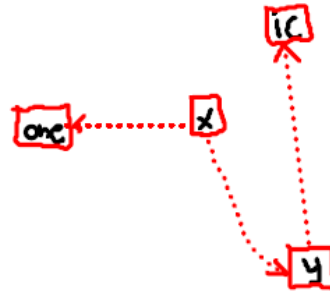
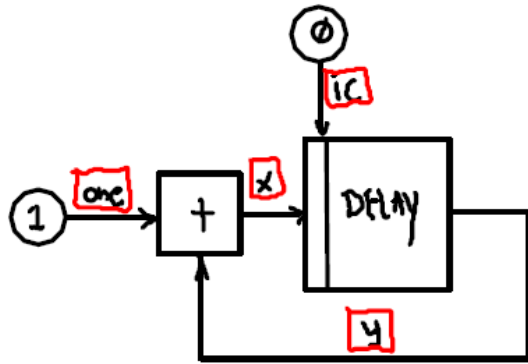
```
        gblock.compute()
```

```
    i++
```

```
else:
```

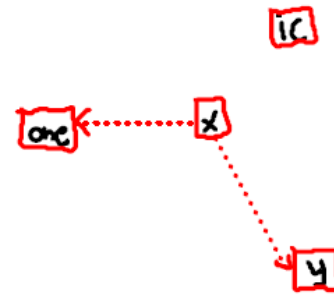
```
    exit()
```

Dependency graph (and hence schedule) depends on time i



$i = \phi$

schedule = [one, IC, y, x]



$\forall i \in \mathbb{N}$

schedule = [y, one, x, IC]

Operational Semantics of DT-CBD (different dependency graphs)

i = 0

```
if (not end_condition(i, ...)) :  
    depGraph = buildDepGraph(CBD)  
    schedule = loopDetectAndTopSort(depGraph)  
    for gblock in schedule:  
        gblock.compute()  
else:  
    exit()
```

i = 1

```
while (not end_condition(i, ...)) :  
    depGraph = buildDepGraph(CBD)  
    schedule = loopDetectAndTopSort(depGraph)  
    for gblock in schedule:  
        gblock.compute()  
    i++  
else:  
    exit()
```

Operational Semantics of DT-CBD

(loop invariant)

```
i = 0
if (not end_condition(i, ...)):
    depGraph = buildDepGraph(CBD)
    schedule = loopDetectAndTopSort(depGraph)
    for gblock in schedule:
        gblock.compute()
else:
    exit()
```

```
i = 1
```

```
depGraph = buildDepGraph(CBD)
schedule = loopDetectAndTopSort(depGraph)
```

```
while (not end_condition(i, ...)):
    for gblock in schedule:
        gblock.compute()
    i++
else:
    exit()
```

Operational Semantics of DT-CBD

Optimizations:

1. Constant folding
2. Introduce "scope" blocks → remove all parts of the CBD not needed for the "scope" (see 3.)
3. Identify "must compute part of CBD"
= any part feeding into a memory block
(delay, integrator, differentiator).
Remove all parts of the CBD that are not connected
(via dependency) to the "must compute part"
or to a scope block from the schedule.
4. Integration interval (II) vs.
Communication Interval (CI). $CI = K * II$

A schedule containing only "must compute part of CBD"
is evaluated every II.

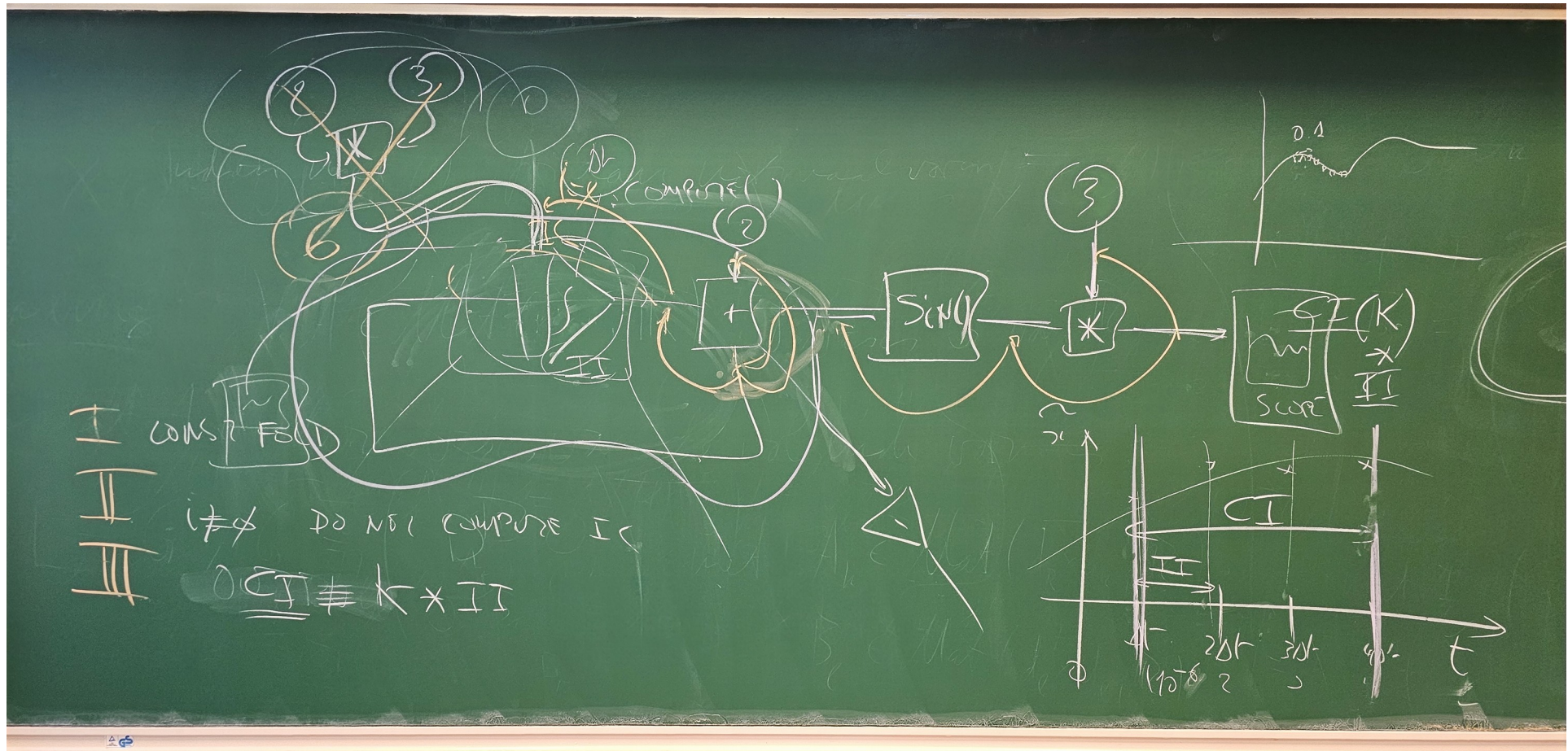
A schedule containing all blocks
(after removal of unused blocks) is evaluated every CI.

Operational Semantics of DT-CBD

Optimizations:

1. Constant folding
2. Introduce "scope" blocks → remove all parts of the CBD not needed for the "scope" (see 3.)
3. Identify "must compute part of CBD"
= any part feeding into a memory block (delay, integrator, differentiator).
Remove all parts of the CBD that are not connected (via dependency) to the "must compute part" or to a scope block from the schedule.
4. Integration interval (II) vs. Communication Interval (CI). $CI = K \times II$

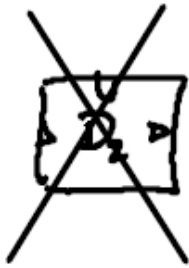
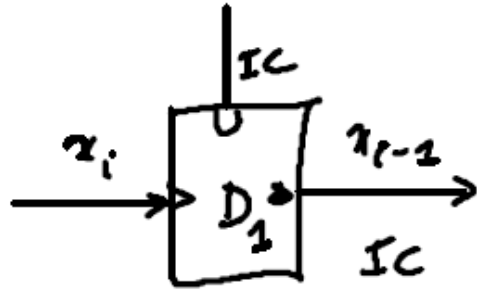
A schedule containing only "must compute part of CBD" is evaluated every II.
A schedule containing all blocks (after removal of unused blocks) is evaluated every CI.



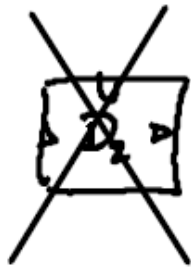
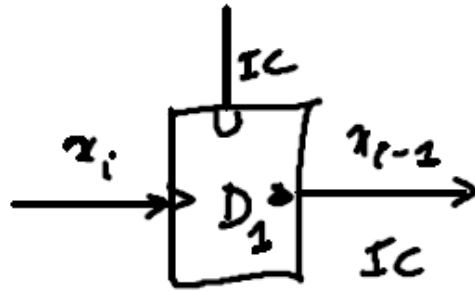
Compilation (rather than interpretation)

- **unroll** the "for gblock in schedule:" loop
- **inline** each `gblock.compute()`

memory over more THAN 1 Time-slice

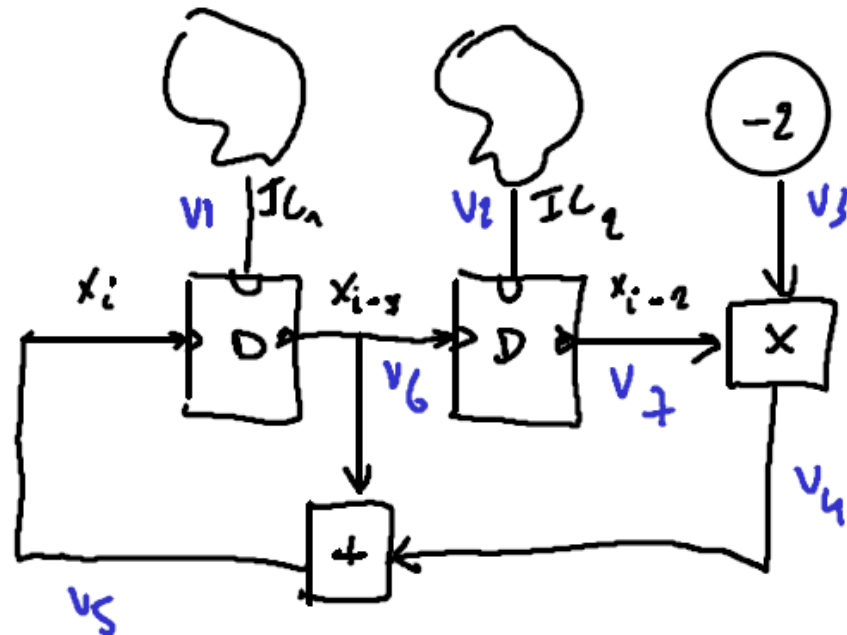


memory over more THAN 1 Time-slice

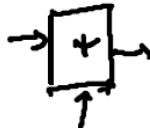
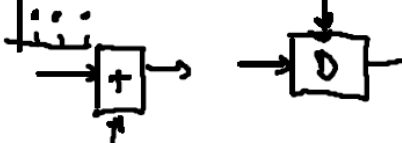
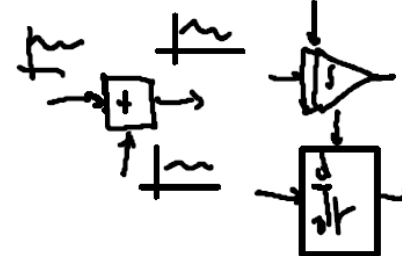


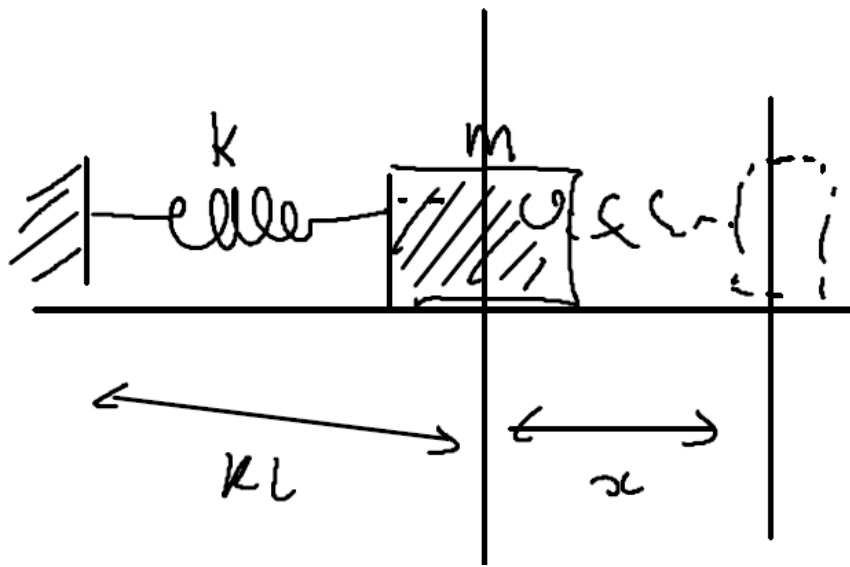
$$x_{i+1} = x_i - 2 \times x_{i-1}$$

$$x_i = x_{i-1} - 2 \times x_{i-2}$$



a family of Causal Block Diagram (CBD) formalisms

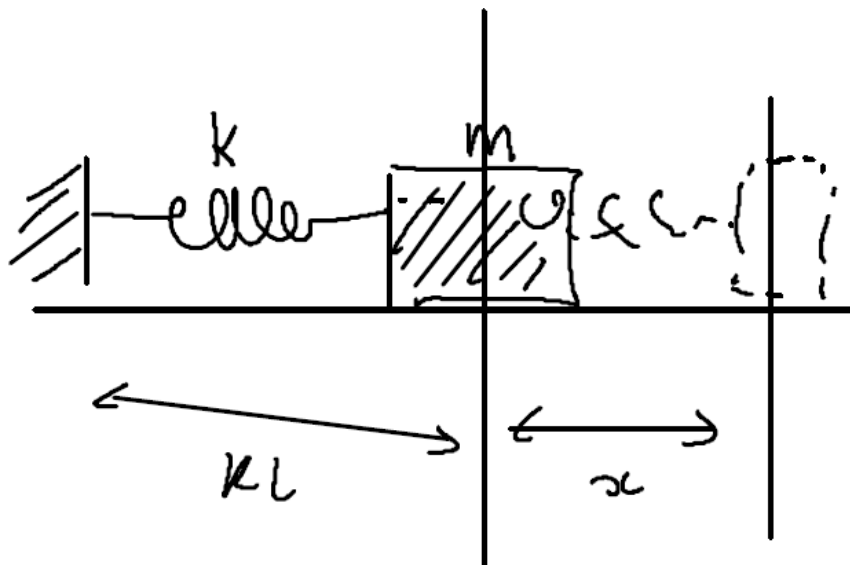
TIME ↓	HIERARCHY ↗		SEMANTICS ↗	
	FLAT CBD	SYNTAX ✓ FLATTEN	DENOTATIONAL "WHAT"	OPERATIONAL "HOW"
{NOW}	ALGEBRAIC (ALG-CBD)	 NO LOOPS WITH LOOPS	✓ ✓	✓ ✓
IN	DISCRETE-TIME (DT-CBD)		✓	✓
IR	CONTINUOUS-TIME (CT-CBD)			



HOOKE'S LAW

$$F = -kx$$

$$x \propto$$



HOORKE'S LAW

$$F = -kx$$

$x \ll$

$$F = ma$$

$$\left\{ \begin{aligned} F &= m \frac{dv}{dt} = m \frac{d^2x}{dt^2} \\ v &= \frac{dx}{dt} \end{aligned} \right.$$

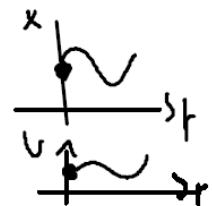
$$\frac{dv}{dt} = -\frac{k}{m} x, v(0)$$

$$\frac{dx}{dt} = v, x(0)$$

$$x(t), v(t)$$

ODE
ORDINARY
DIFFERENTIAL
EQUATIONS

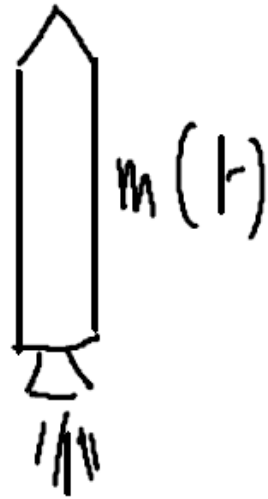
PDE



A black short-sleeved t-shirt is laid flat on a light-colored, wrinkled fabric surface. The t-shirt has a white tag at the collar. In the center of the chest, there is a white printed design consisting of a physics equation and a sentence below it.
$$\vec{F} = m \cdot \vec{a}$$

It's **not** rocket science!

F = m a is not “rocket science”

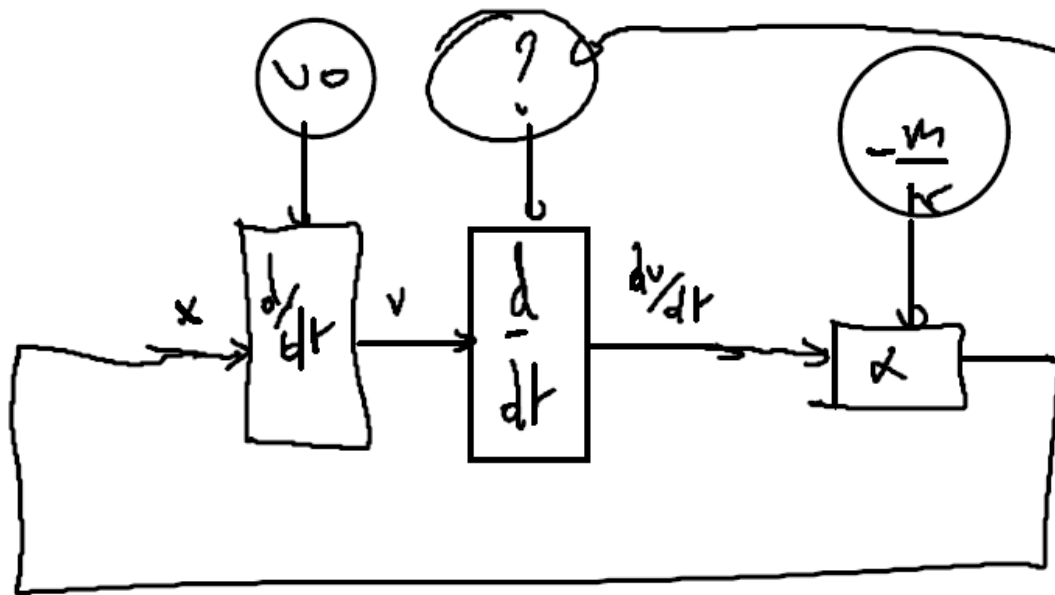


$$v = \frac{d}{dt} x$$

$$a = \frac{d}{dt} v = \frac{d^2}{dt^2} x$$

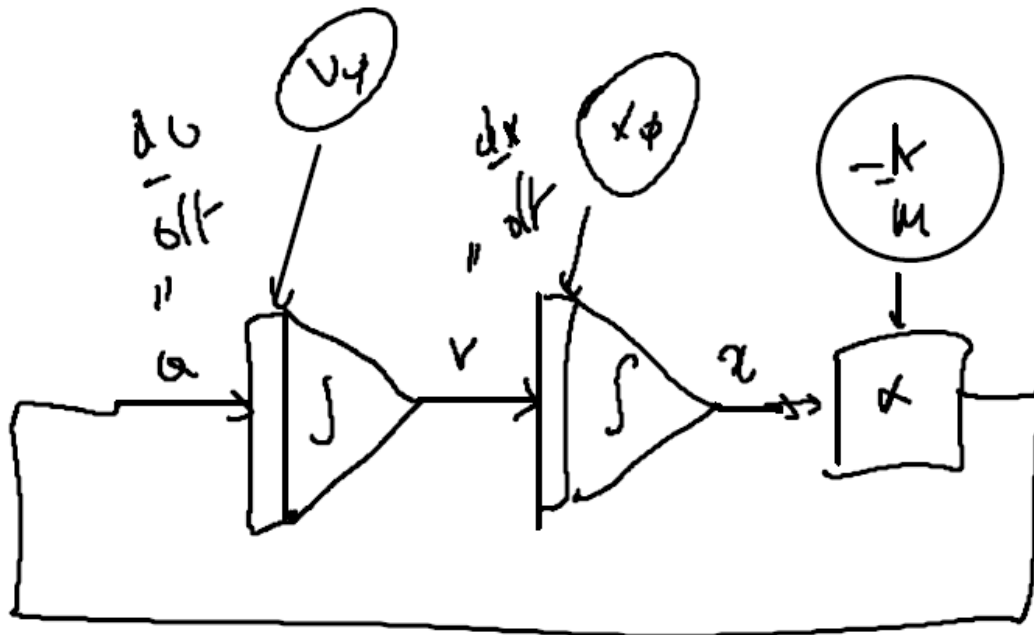
$$F = \frac{dP}{dt} = \frac{d}{dt} (mv)$$

$$= \underbrace{\frac{d}{dt} m}_{\text{}} \cdot v + m \left(\frac{dv}{dt} \right)^a$$



$$\alpha = -\frac{m}{k} \frac{dv}{dt}$$

$$\frac{dv}{dt}(0) = -\frac{k}{m} x(0)$$

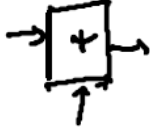
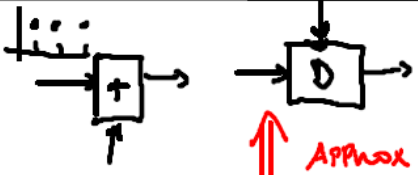
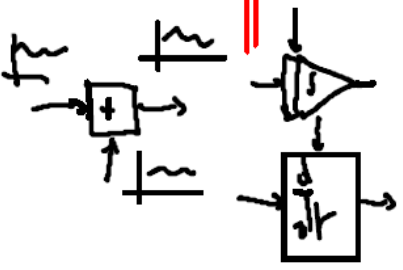


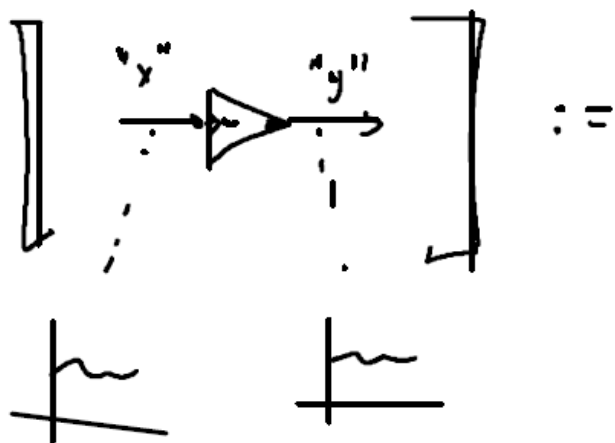
$$\lim_{\Delta t \rightarrow 0} \phi$$

Physical Systems Modelling

- Problem-Specific (technological)
- Domain-Specific (e.g., translational mechanical)
- (general) Laws of Physics
- Power Flow/Bond Graphs (physical: energy/power)
- Computationally a-causal
(Mathematical and Object-Oriented) ← **Modelica**
- Causal Block Diagrams (data flow)
- Numerical (Discrete) Approximations
- Computer Algorithmic + Numerical
(Floating Point vs. Fixed Point)
- As-Fast-As-Possible vs. Real-time (XiL)
- Hybrid (discrete-continuous) modelling/simulation
- Hiding IP: Composition of Functional Mockup Units (FMI)
- Dynamic Structure

a family of Causal Block Diagram (CBD) formalisms

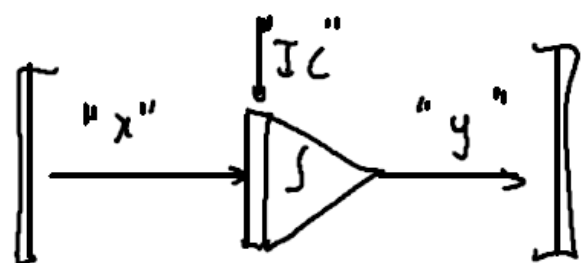
Time ↓	Hierarchy ↗		Semantics ↗	
	FLAT	SYNTAX	DENOTATIONAL "WHAT"	OPERATIONAL "HOW"
	CBD			
{NOW}	ALGEBRAIC (ALG-CBD)	 NO LOOPS WITH LOOPS	✓ ✓	✓
DN	DISCRETE-TIME (DT-CBD)		✓	✓
CR	CONTINUOUS-TIME (CT-CBD)		✓	✗



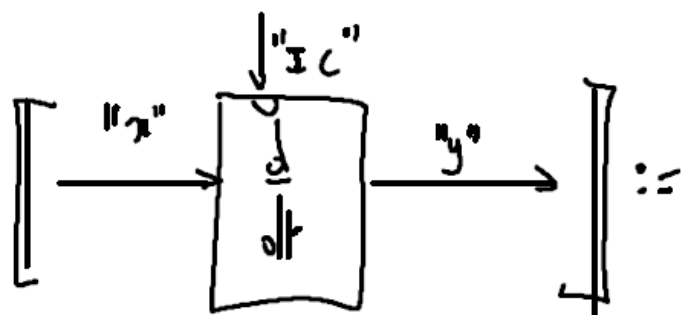
$$u y(t) = -u x(t),$$

$$\forall t \in \mathbb{R}$$

$$-: \mathcal{X} \times \mathcal{U} \rightarrow \mathcal{Y}$$



$$u y(t) = \underbrace{u y(\phi)}_{I_C(\phi)} + \int_{\phi}^t x(\tau) d\tau, \quad \forall t \in \mathbb{R}$$



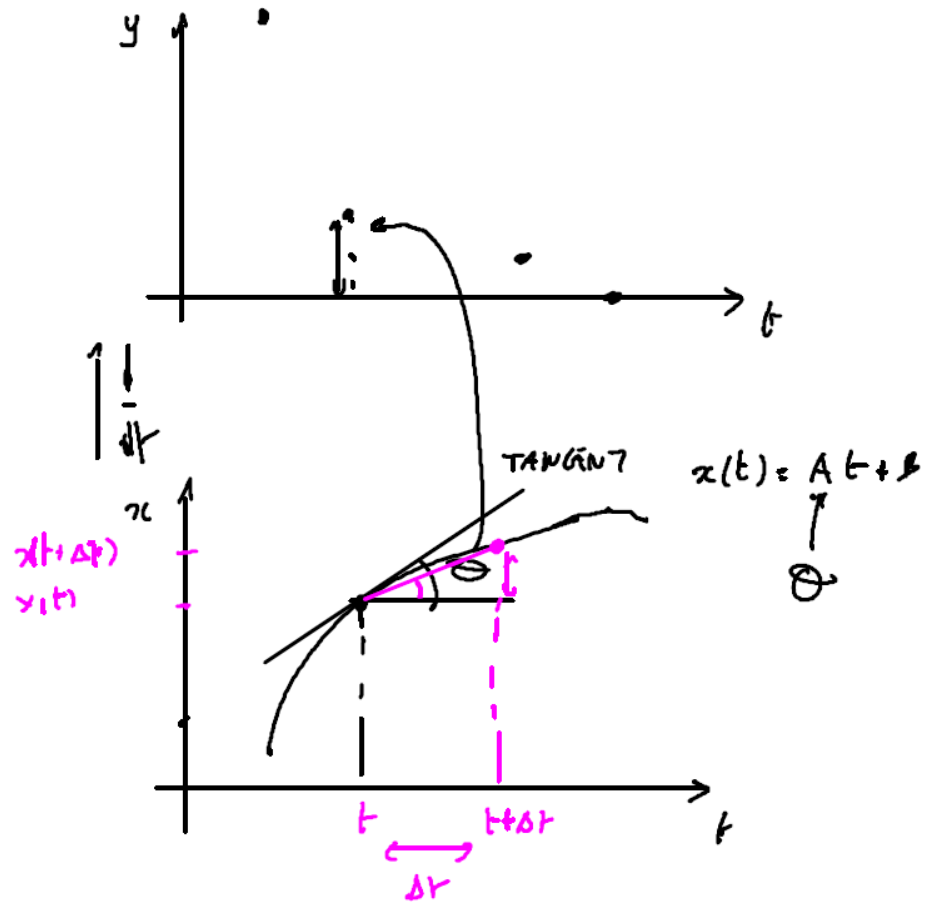
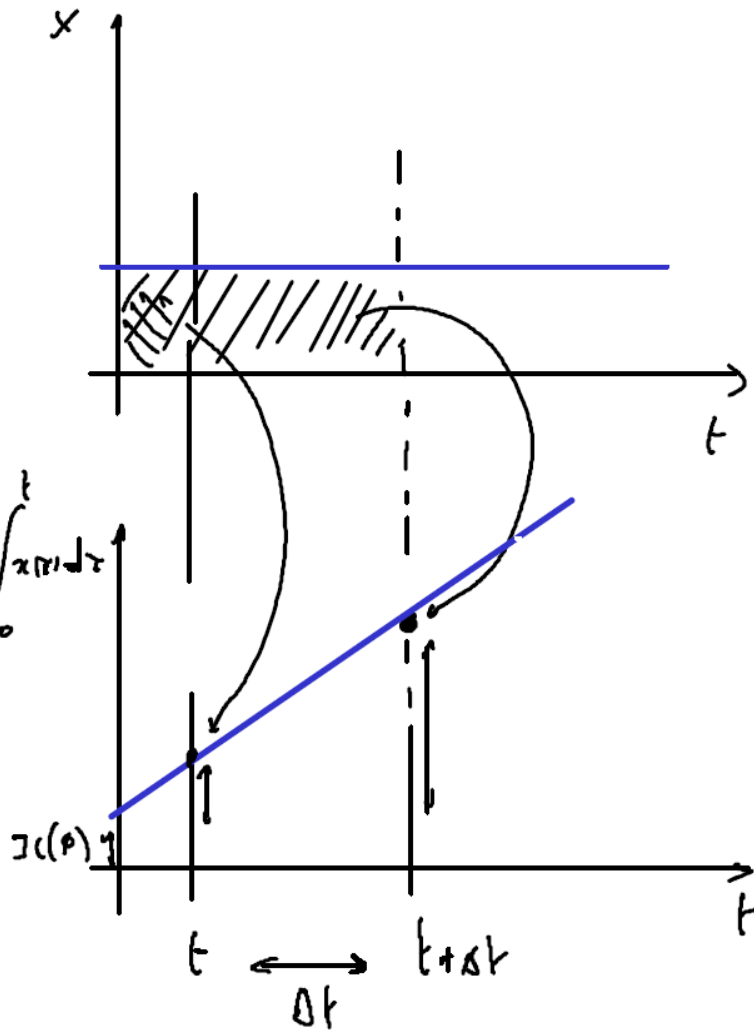
$$u y(t) = \frac{d}{dt} u x(t), \quad \forall t \in \mathbb{R} \setminus \{p\}$$

$$u y(\phi) = I_C(\phi)$$

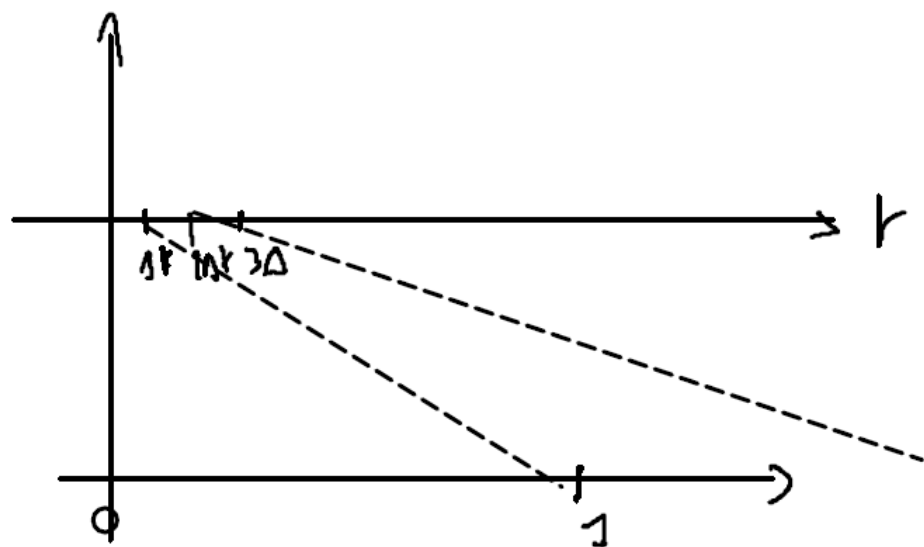
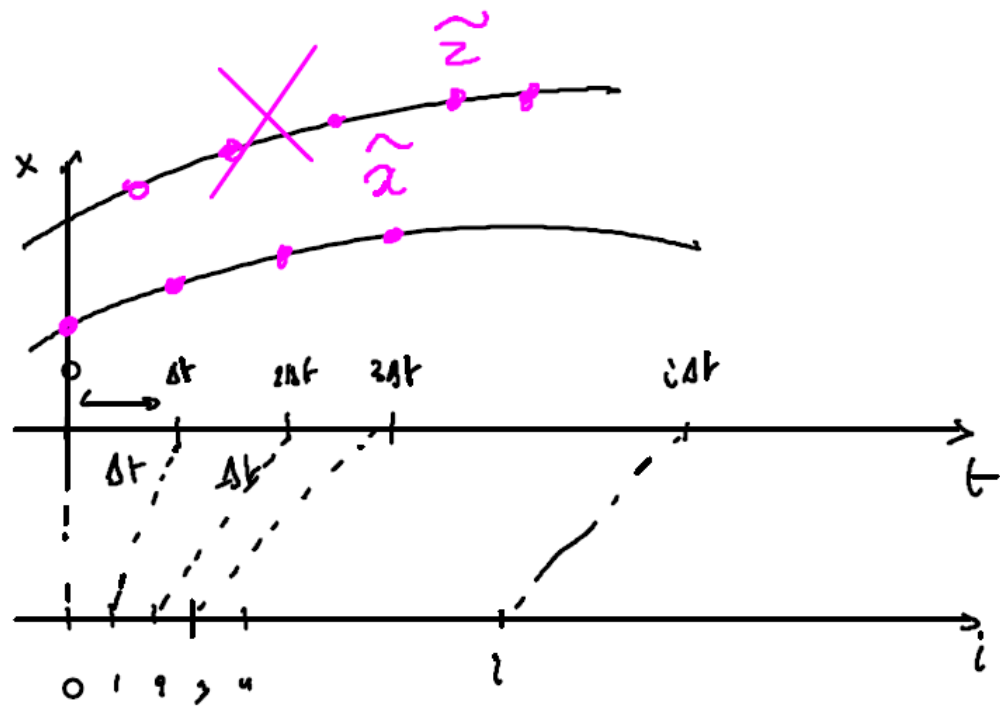
$\frac{d}{dt}$

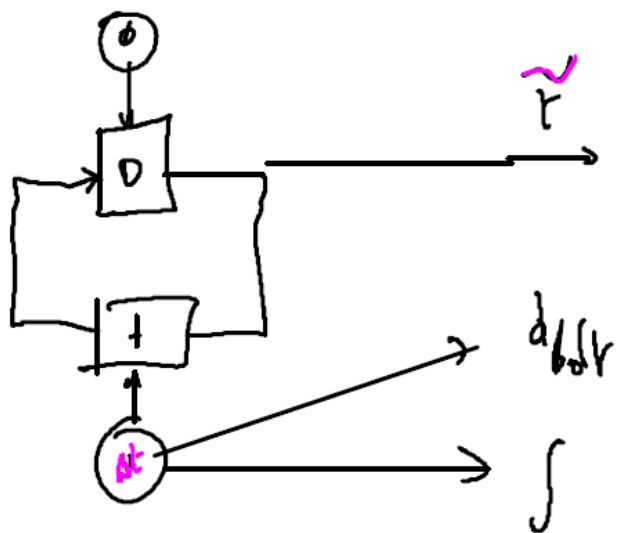
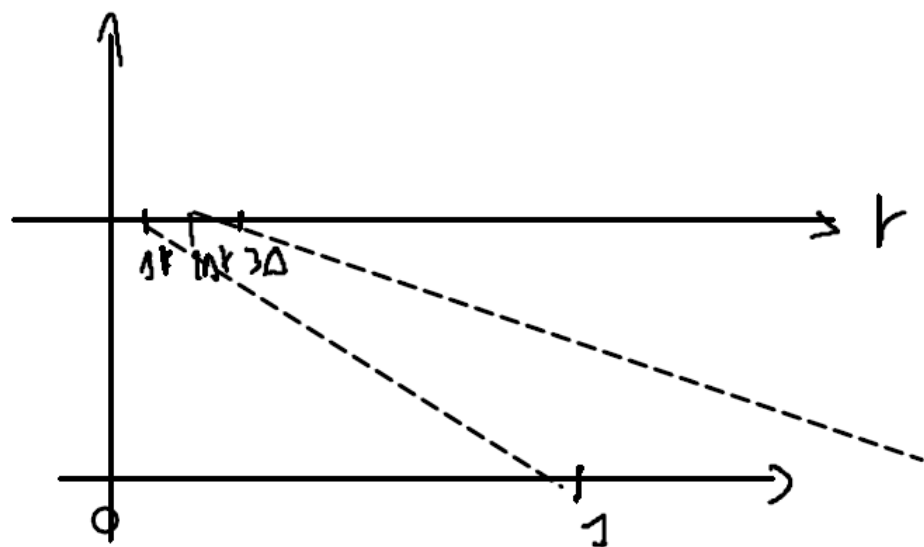
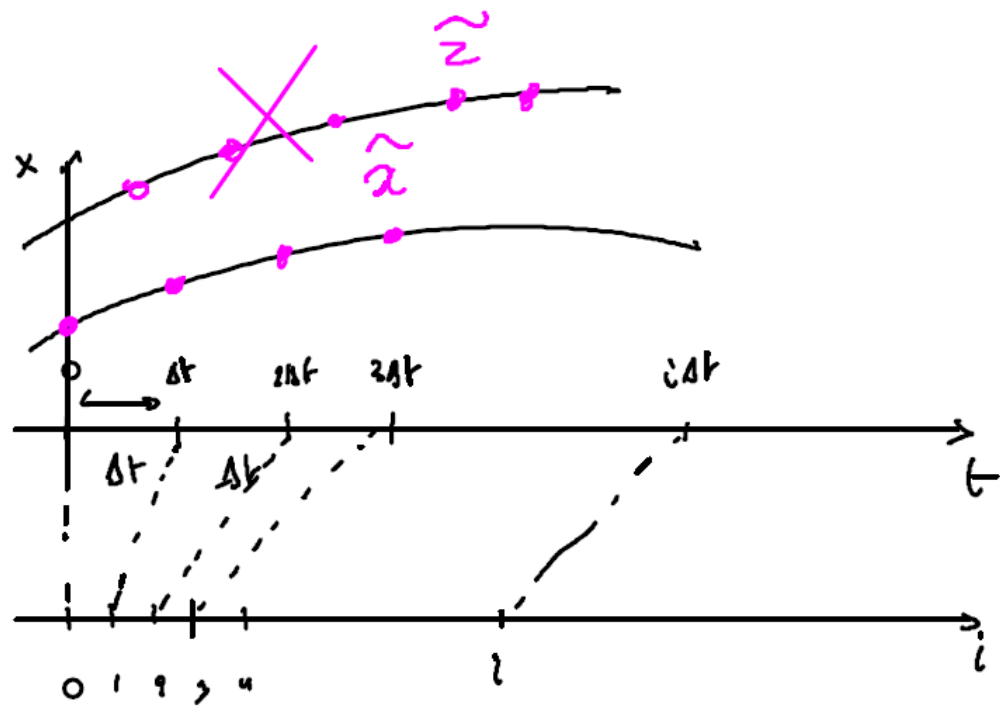


$$IC(\phi) = \int_0^t x(\tau) d\tau$$



$$\lim_{\Delta t \rightarrow 0} \frac{x(t + \Delta t) - x(t)}{\Delta t}$$

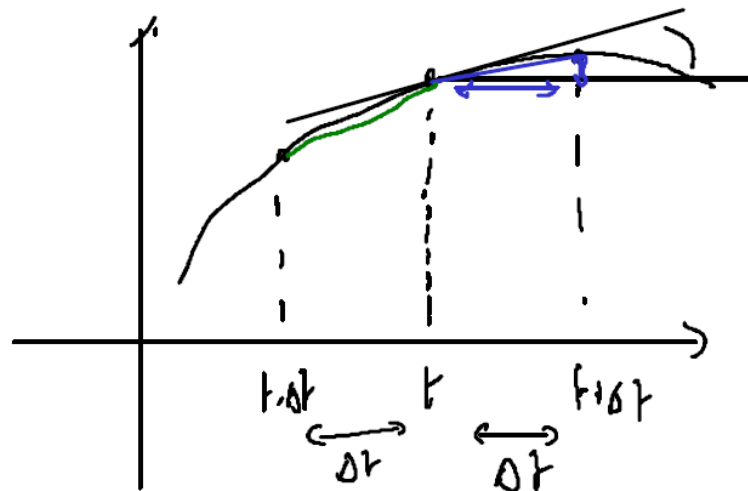




$$\tilde{y}(t) = \frac{\tilde{x}(t) - \tilde{x}(t-\Delta t)}{\Delta t}$$

$$\tilde{\tilde{y}}(t) = \frac{\tilde{\tilde{x}}(t+\Delta t) - \tilde{\tilde{x}}(t)}{\Delta t}$$

$$\Delta t \ll$$



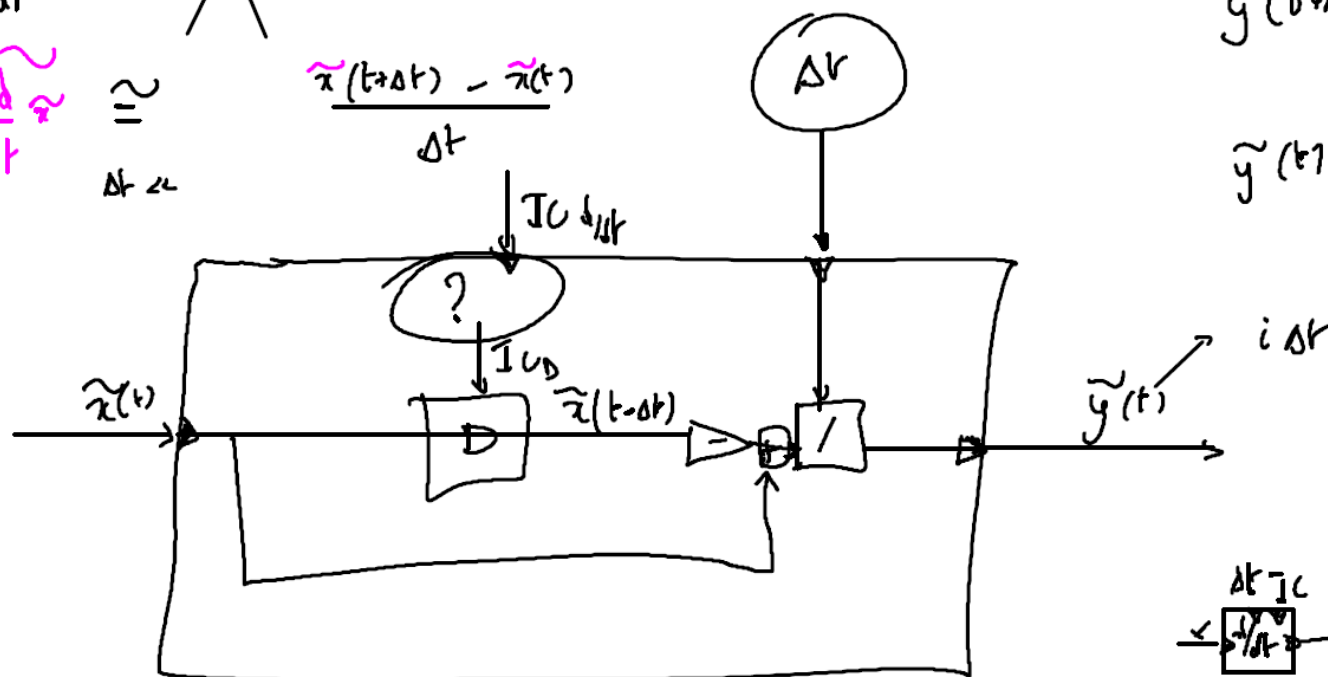
$$\lim_{\Delta t \rightarrow 0}$$

$$\tilde{y}(t) = \frac{d\tilde{x}}{dt}(t) = \frac{\tilde{x}(t+\Delta t) - \tilde{x}(t)}{\Delta t}$$

$$\tilde{\tilde{y}}(t) = \frac{d\tilde{\tilde{x}}}{dt}(t) = \frac{\tilde{\tilde{x}}(t) - \tilde{\tilde{x}}(t-\Delta t)}{\Delta t}$$

$$\frac{d}{dt} x(t) = \lim_{\Delta t \rightarrow 0} \frac{x(t+\Delta t) - x(t)}{\Delta t}$$

$$\tilde{y}(t) = \frac{d}{dt} \tilde{x} \approx \frac{\tilde{x}(t+\Delta t) - \tilde{x}(t)}{\Delta t}$$



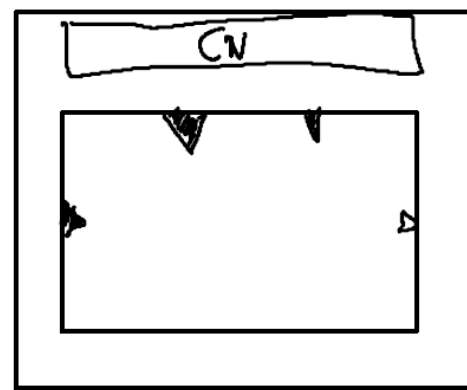
$$\tilde{y}(t+\Delta t) = \frac{x(t+\Delta t) - x(t)}{\Delta t}$$

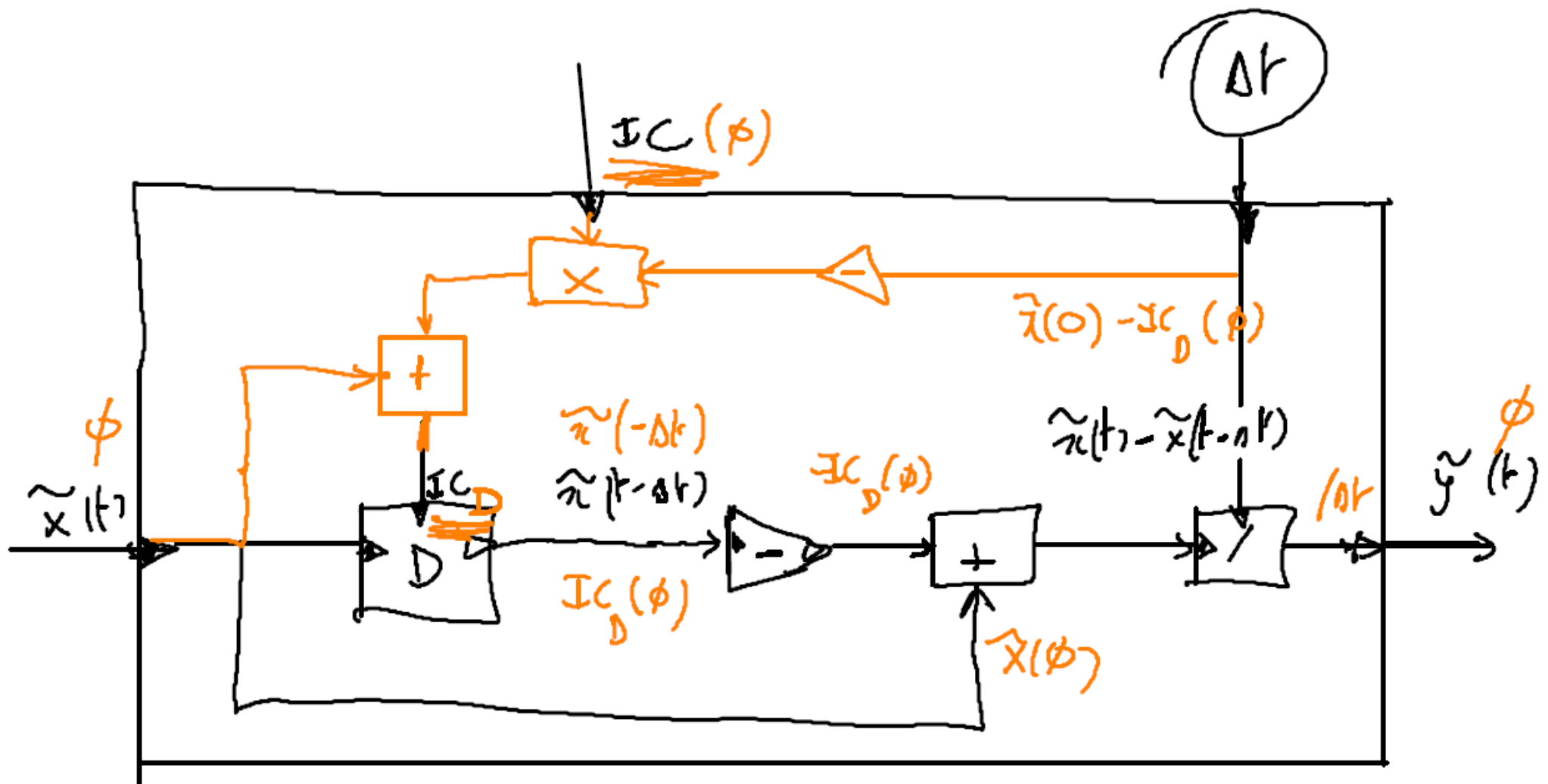
$$\tilde{y}(t) = \frac{\tilde{x}(t) - \tilde{x}(t-\Delta t)}{\Delta t}$$

$$\frac{\Delta t I_C}{\Delta t} y = \frac{d}{dt} x \quad y(0) = I_C(0)$$

All blocks : $= \Delta t$

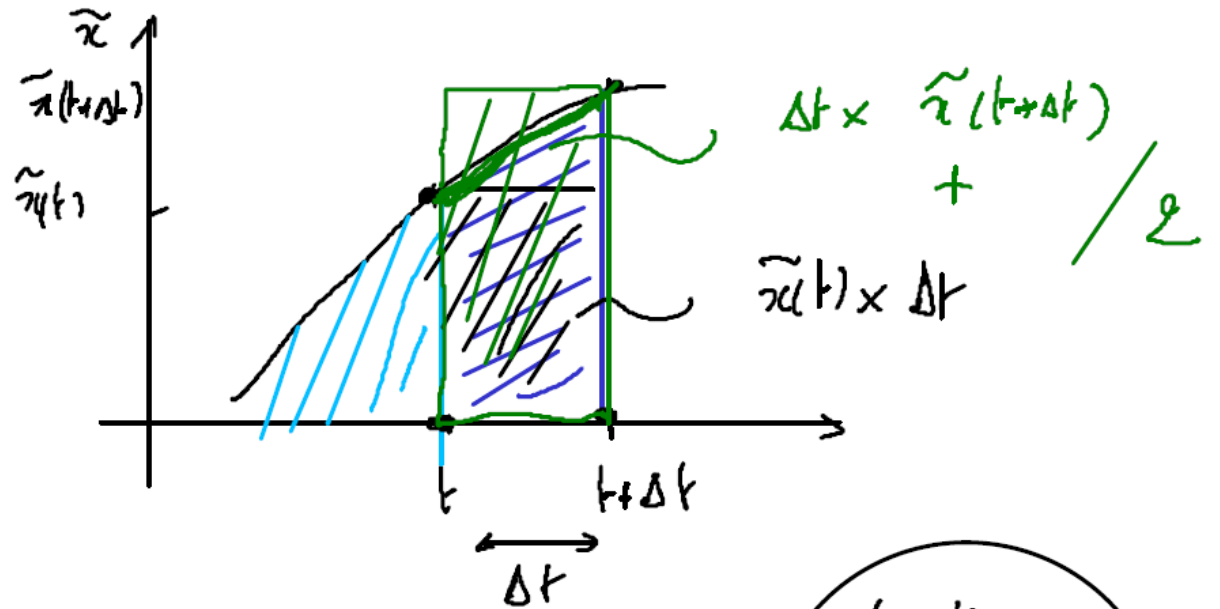
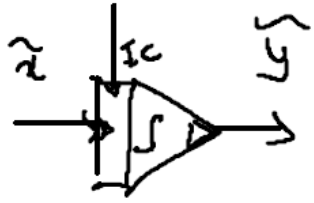
SYNCHRONOUS BLOCK DIAGRAMS





$$\frac{\tilde{x}(0) - I_{C_D}(0)}{\Delta t} = I_{C_D}(0) \quad \tilde{y}(t) = I_{C_D}(0)$$

$$\tilde{x}(0) - I_{C_D}(0) \cdot \Delta t = I_{C_D}(0)$$

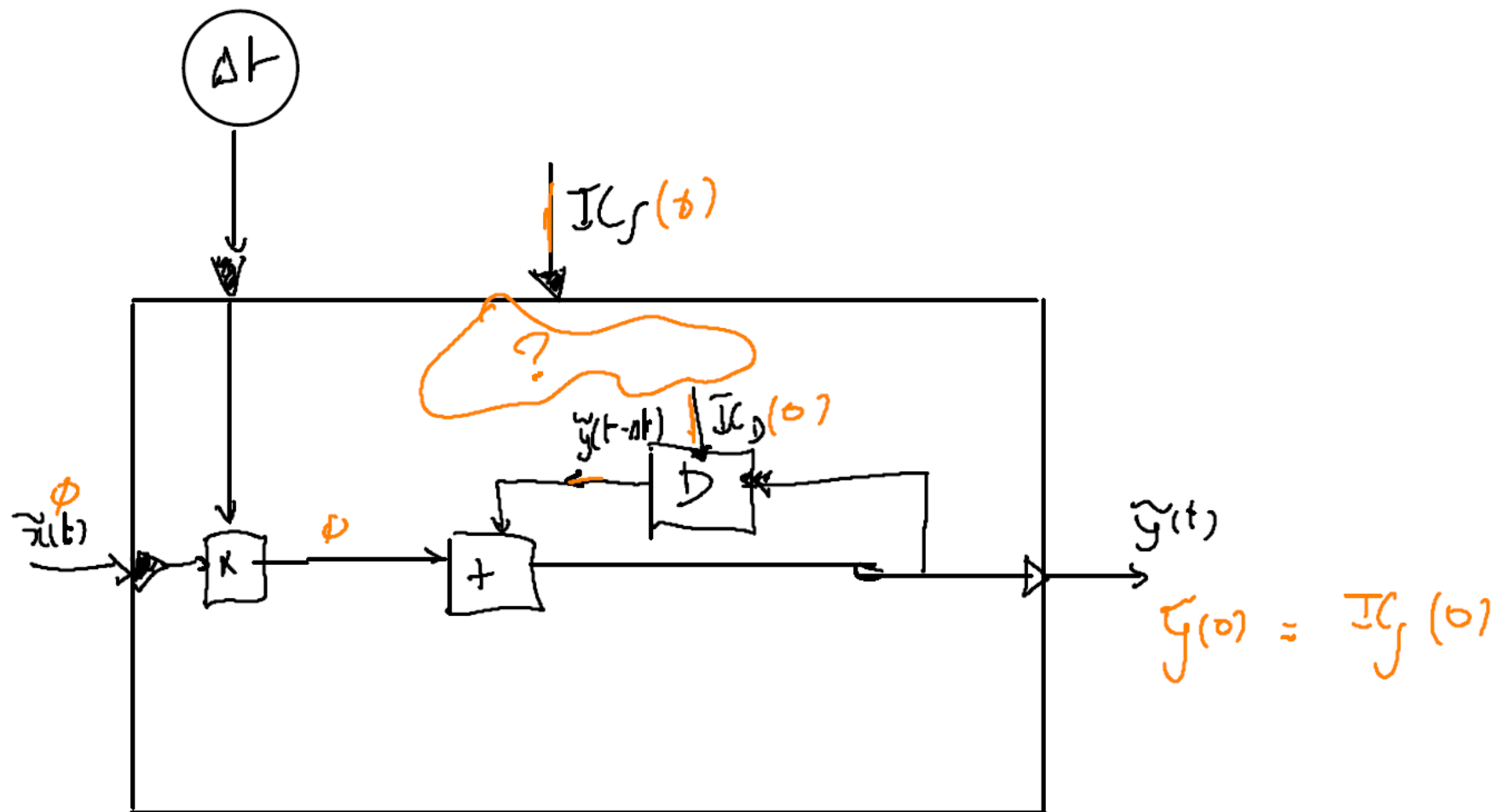


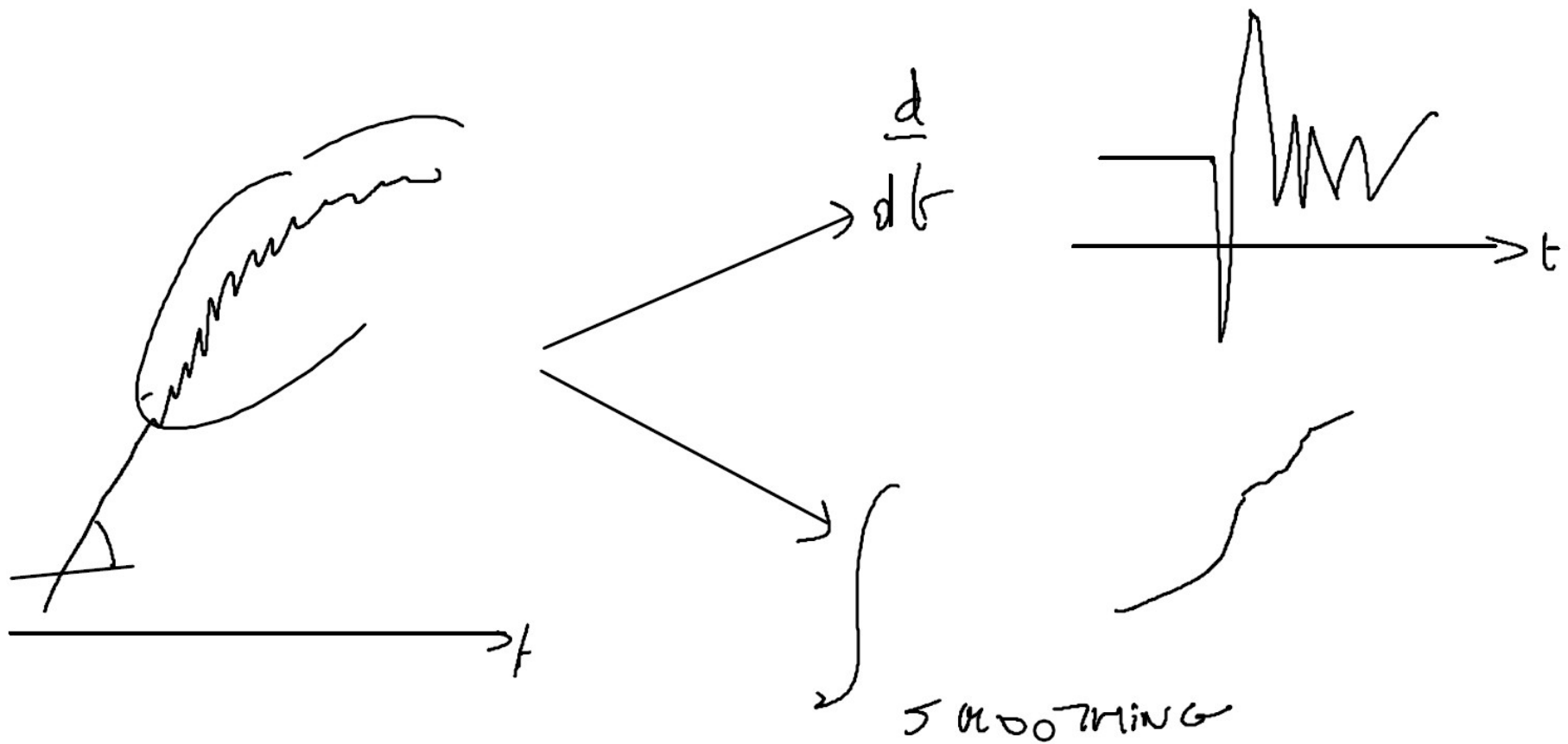
$$\tilde{y}(t+\Delta t) = \tilde{y}(t) + \int_t^{t+\Delta t} \tilde{x}(\tau) d\tau$$

$$\lim_{\Delta t \rightarrow 0} \frac{\Delta t \tilde{x}(t)}{\Delta t \tilde{x}(t+\Delta t)}$$

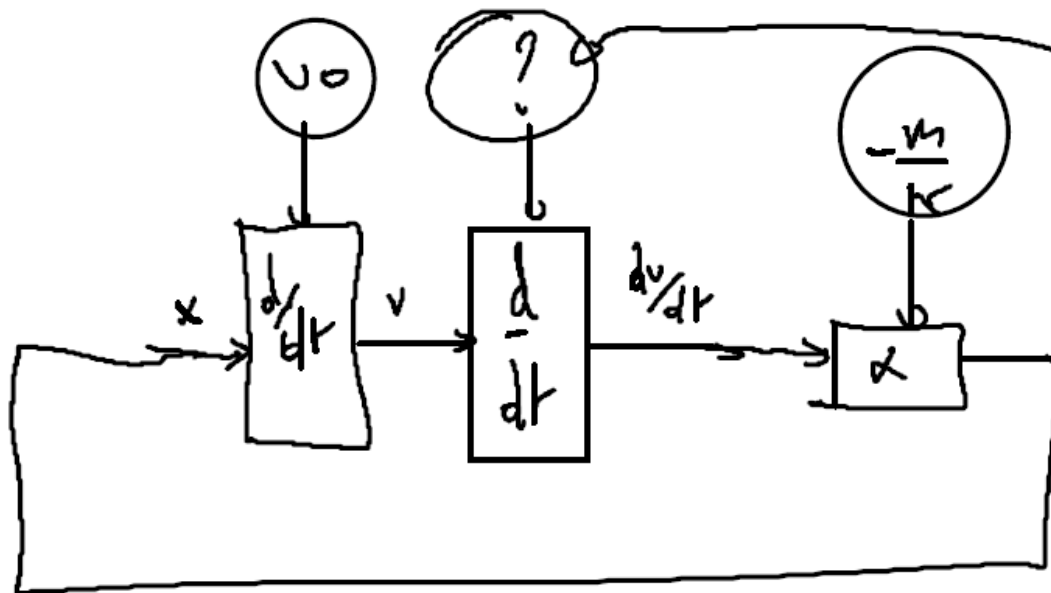
$$\tilde{y}(t) = y(t-\Delta t) + \Delta t \tilde{x}(t)$$

Approx





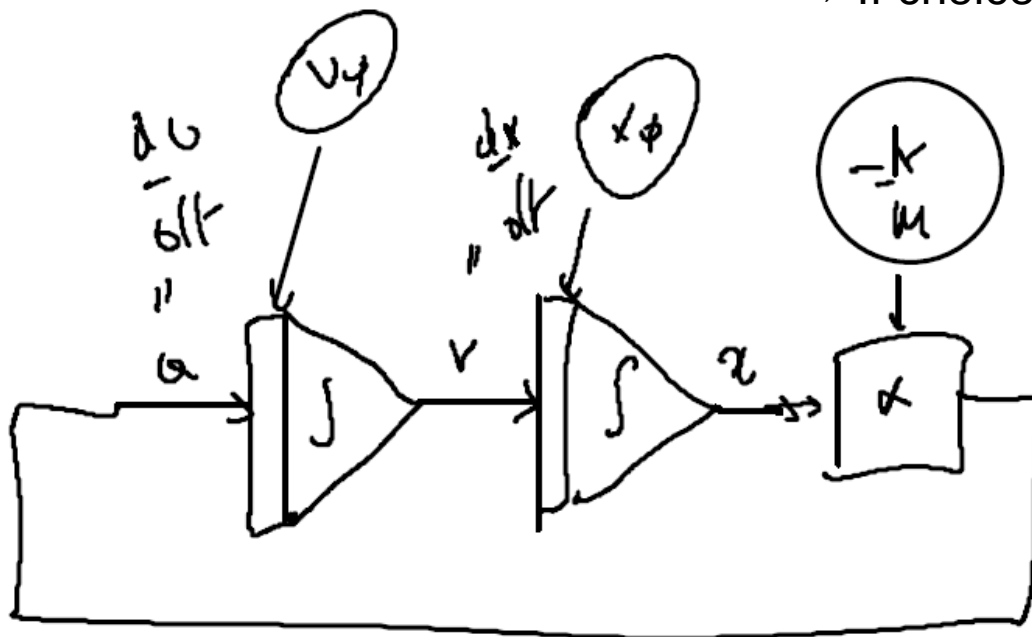
→ If choice: prefer intergration over differentiations



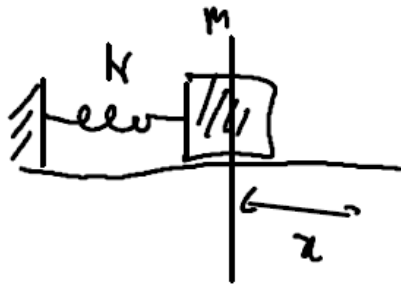
$$x = -\frac{m}{k} \frac{dv}{dt}$$

$$\frac{dv}{dt}(0) = -\frac{k}{m} x(0)$$

→ If choice: prefer integration over differentiation

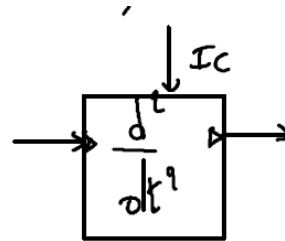


$$\lim_{\Delta t \rightarrow 0} \phi$$

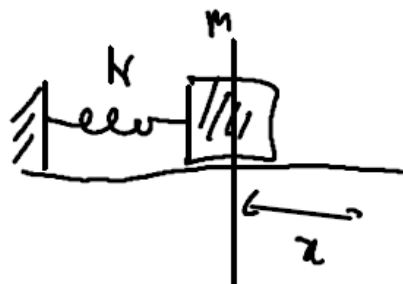


HARMONIC EPN.

$$\frac{d^2 x}{dt^2} = -x, \quad x(0) = x_p, \quad \frac{dx}{dt}(0) = v(0) = v_p \quad \text{ODE}$$

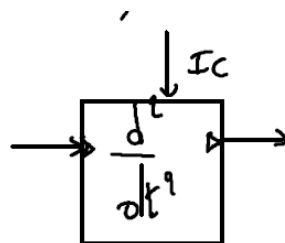


$$\begin{cases} \frac{dx}{dt} = v, & x(p) = x_p \\ \frac{dv}{dt} = -x, & v(p) = v_p \end{cases}$$



HARMONIC EQN.

$$\frac{d^2 x}{dt^2} = -x, \quad x(0) = x_0, \quad \frac{dx}{dt}(0) = v(0) = v_0 \quad \text{ODE}$$



$$\begin{cases} \frac{dx}{dt} = v, & x(0) = x_0 \\ \frac{dv}{dt} = -x, & v(0) = v_0 \end{cases}$$

HIGHER-ORDER ODES

$$\frac{d^h x}{dt^h} = \frac{d}{dt} \left(\frac{d}{dt} \left(\dots \frac{d}{dt} x \right) \right)$$

$$\begin{cases} \frac{dx}{dt} = x_{den-1} \\ \frac{dx_{den-1}}{dt} = x_{den-2} \\ \vdots \\ \frac{dx_{den-hen-1}}{dt} = x_{den-h} \end{cases}$$

$$\frac{dx}{dt} = -x, \quad x(0) = \phi, \quad \frac{dx}{dt}(0) = 1$$

$$\frac{dx}{dt} = x$$

$$\begin{cases} x(t) = A \sin(t) + B \cos(t) + C \\ x(0) = \phi, \quad \frac{dx}{dt}(0) = 1 \end{cases}$$

$$x(t) = e^x + C$$

$$\frac{dx}{dt} = A \cos(t) - B \sin(t)$$

$$\frac{dx}{dt} = -A \sin(t) - B \cos(t) = -(A \sin(t) + B \cos(t)) = -x(t) \quad \text{q.e.d.}$$

$$x(0) = B = \phi$$

$$\frac{dx}{dt}(0) = A = 1$$

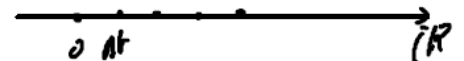
$$\begin{cases} x(t) = \sin(t) \\ \frac{dx}{dt} = \cos(t) \end{cases}$$

$$\tilde{x}(\tilde{t})$$

$$\frac{dx}{dt} = \tilde{v}(\tilde{t})$$

$$\tilde{t} = i \times \Delta t$$

$$\wedge$$

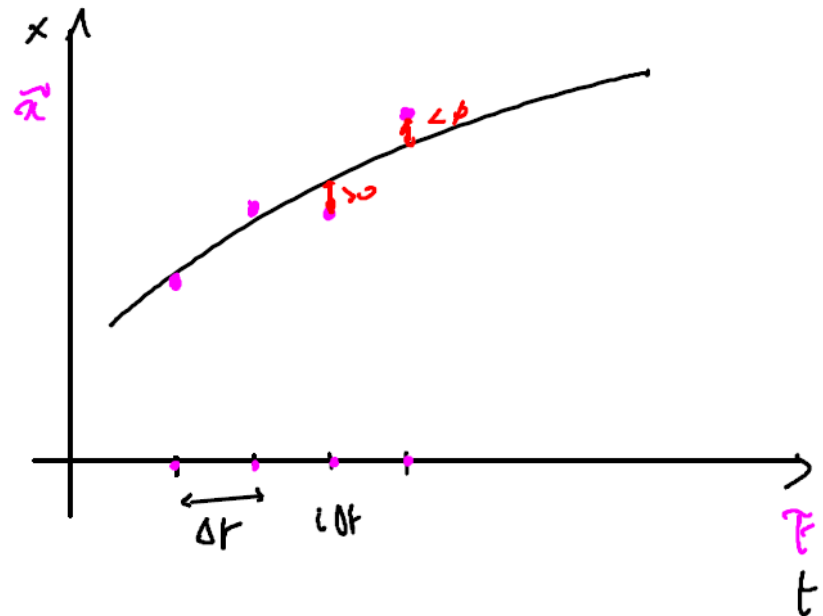
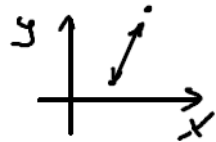


$$\text{Error} \sim e(i \Delta t) = x(i \Delta t) - \tilde{x}(i \Delta t)$$

$$\text{Total Error} \sim \sum_{i=0}^{i_{\text{end}}} e(i \Delta t)$$

$$e(i \Delta t) = |x(i \Delta t) - \tilde{x}(i \Delta t)|$$

$$e(i \Delta t) = (x(i \Delta t) - \tilde{x}(i \Delta t))^2$$

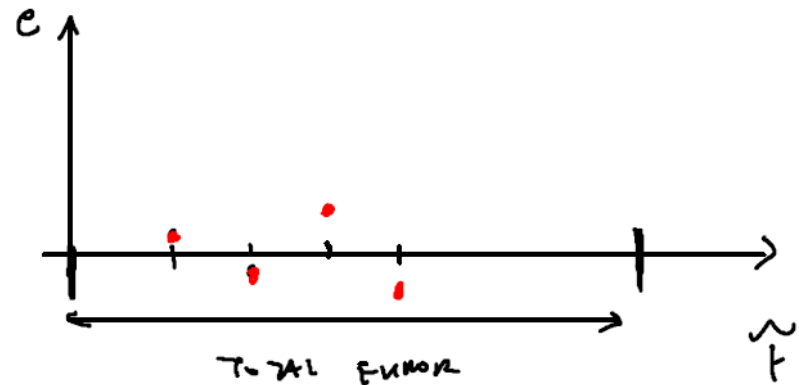


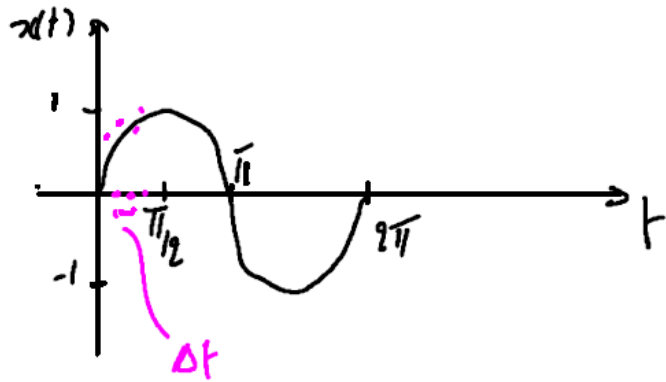
"SUM OF SQUARES"

$$\text{Total Error} = \sum_{i=0}^{i_{\text{end}}} (x(i \Delta t) - \tilde{x}(i \Delta t))^2$$

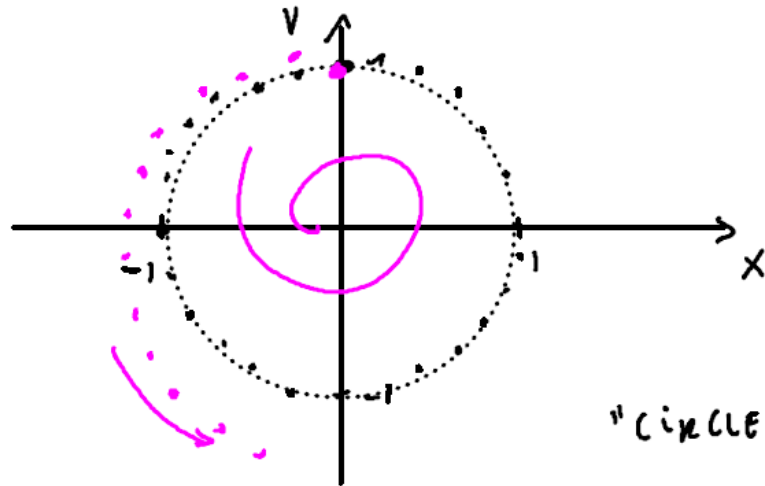
$$= \int \underbrace{(x(\tau))}_{\substack{\uparrow \\ x_{\text{in}}(\tau) \\ (x_0)}} - \underbrace{\tilde{x}(\tau)}_{(x_0)}^2 d\tau$$

(b) $\int$

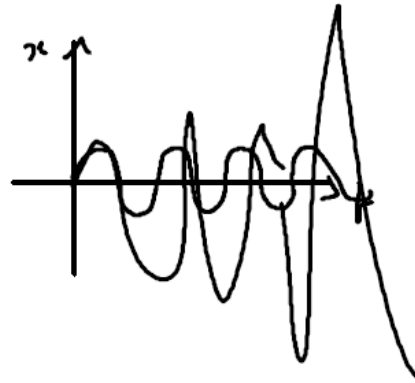




PHASE PLOT



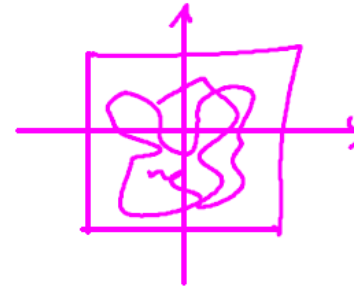
"CIRCLE TEST"



UNSTABLE

SYSTEM ?
NUMERICAL ?

BOUNDED REGION



$\sim \Delta t$, SCHEME



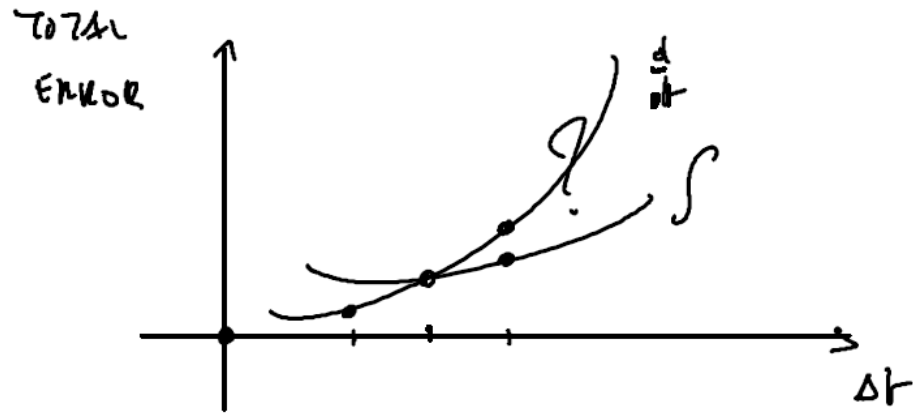
$\sim$ STABILITY

- SYSTEM (SUS)
- NUMERICAL

$(x(t), v(t))$

$$x(0) = \psi, \quad v(0) = 1$$

$$x\left(\frac{\pi}{2}\right) = \quad, \quad v\left(\frac{\pi}{2}\right) =$$



DISCRETIZATION
SCHEME
FIXED

$$\text{TOTAL ERROR } (\Delta t) = \mathcal{O}(\Delta t)$$

$$x(t + \Delta t) = x(t) + \frac{x'(t) \Delta t}{1!} + \frac{x''(t) \Delta t^2}{2!} + \frac{x^{(3)}(t) \Delta t^3}{3!} + \dots$$

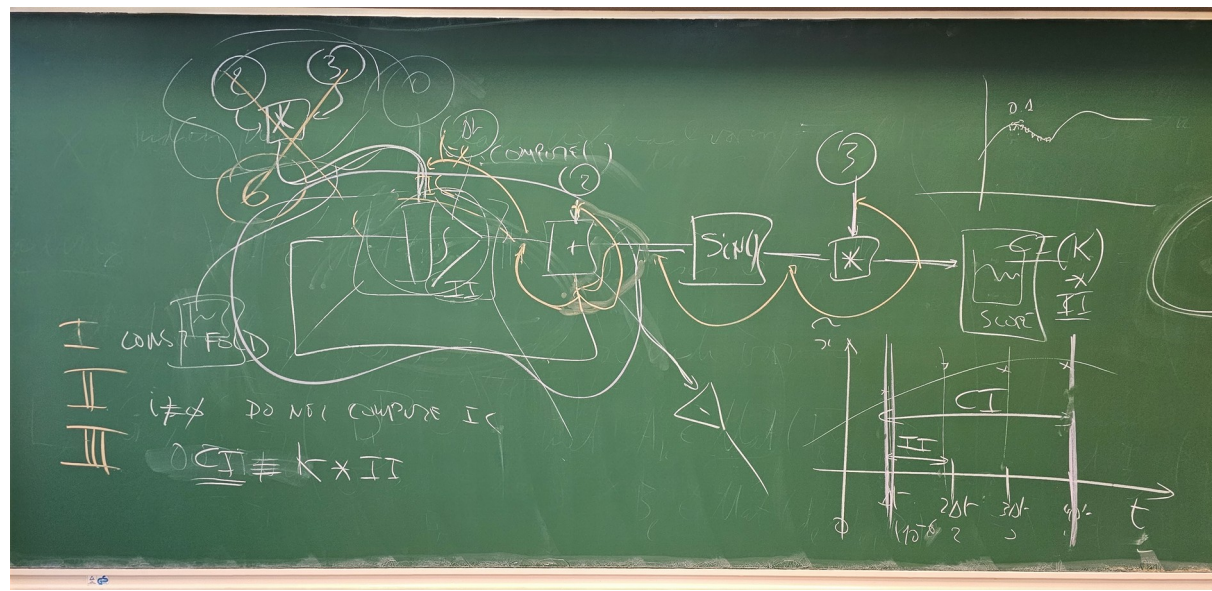
$\xrightarrow{\hspace{2cm}}$
 $\mathcal{O}(\Delta t^2)$

Taylor Expansion

Optimization: Adaptive Step-Size

- Naive: keep halving/doubling step-size
- Efficient (e.g., RK45):
Use difference with higher-order approximation
to estimate step-wise error.
Adapt step-size proportionally to error.

Optimization: CI vs. II



OMEdit - Simulation Setup - harmonicEquation

Simulation Setup - harmonicEquation

General Interactive Simulation Translation Flags Simulation Flags Output

Simulation Interval

Start Time: 0 secs

Stop Time: 100 secs

☐ Number of Intervals: 500

☒ Interval: 0.2 Communication Interval (CI) secs

Integration

Method: euler

Tolerance: 1e-06 ~ Integration Interval (II)

Jacobian:

☐ Save experiment annotation inside model i.e., experiment annotation

☐ Save translation flags inside model i.e., __OpenModelica_commandLineOptions annotation

☐ Save simulation flags inside model i.e., __OpenModelica_simulationFlags annotation

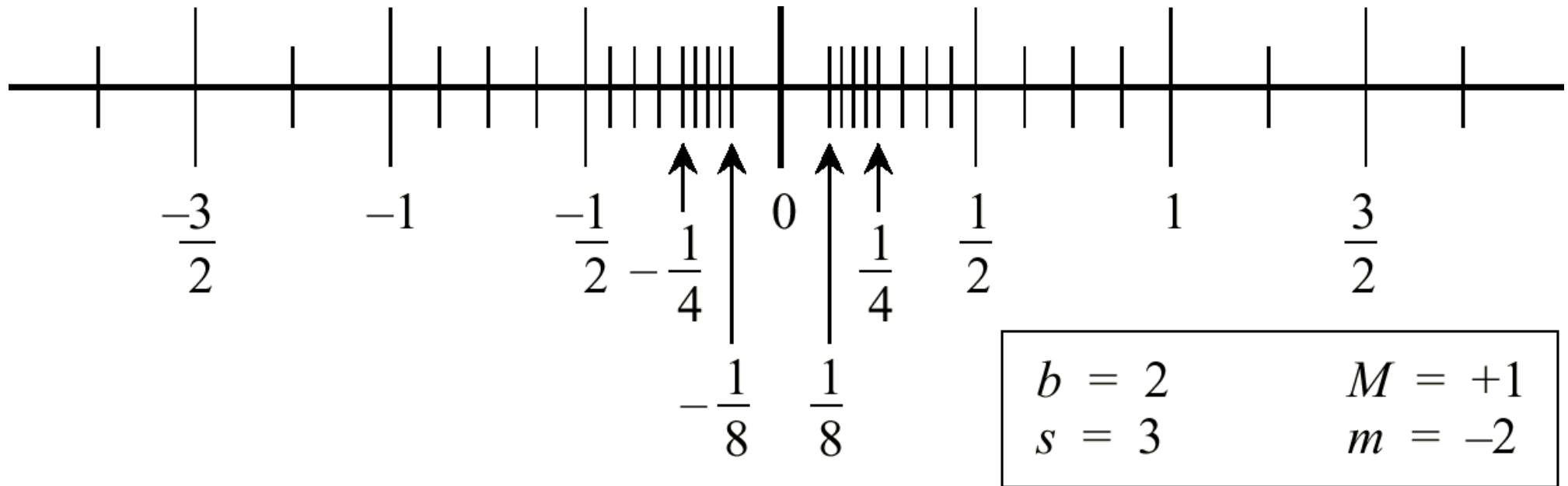
☒ Simulate

OK Cancel

Physical Systems Modelling

- Problem-Specific (technological)
- Domain-Specific (e.g., translational mechanical)
- (general) Laws of Physics
- Power Flow/Bond Graphs (physical: energy/power)
- Computationally a-causal
(Mathematical and Object-Oriented) ← **Modelica**
- Causal Block Diagrams (data flow)
- Numerical (Discrete) Approximations
- Computer Algorithmic + Numerical
(Floating Point vs. Fixed Point)
- As-Fast-As-Possible vs. Real-time (XiL)
- Hybrid (discrete-continuous) modelling/simulation
- Hiding IP: Composition of Functional Mockup Units (FMI)
- Dynamic Structure

Example Floating Point Format

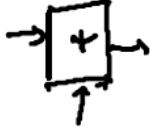
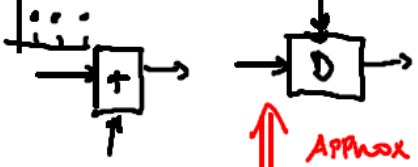
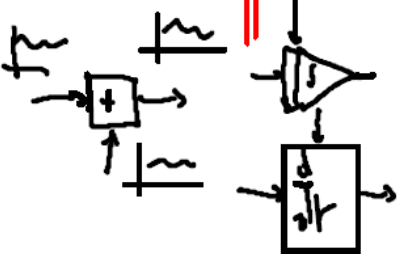


- Smallest non-zero positive number = $b^m \times b^{-1} = 1/8$
- Largest non-zero positive number = $b^M \times (1 - b^{-s}) = 7/4$
- Smallest gap = $b^m \times b^{-s} = 1/32$
- Largest gap = $b^M \times b^{-s} = 1/4$
- Number of representable numbers = $2 \times ((M - m) + 1) \times (b - 1) \times b^{s-1} + 1 = 33$
 ... fits into available bits? Optimal number of bits?
- Note: fill the gap around 0: **de-normalized**

Physical Systems Modelling

- Problem-Specific (technological)
 - Domain-Specific (e.g., translational mechanical)
 - (general) Laws of Physics
 - Power Flow/Bond Graphs (physical: energy/power)
 - Computationally a-causal
(Mathematical and Object-Oriented) ← **Modelica**
 - Causal Block Diagrams (data flow)
 - Numerical (Discrete) Approximations
 - Computer Algorithmic + Numerical
(Floating Point vs. Fixed Point)
 - As-Fast-As-Possible vs. Real-time (XiL)
 - Hybrid (discrete-continuous) modelling/simulation
 - Hiding IP: Composition of Functional Mockup Units (FMI)
 - Dynamic Structure
- 
- ++ hierarchy

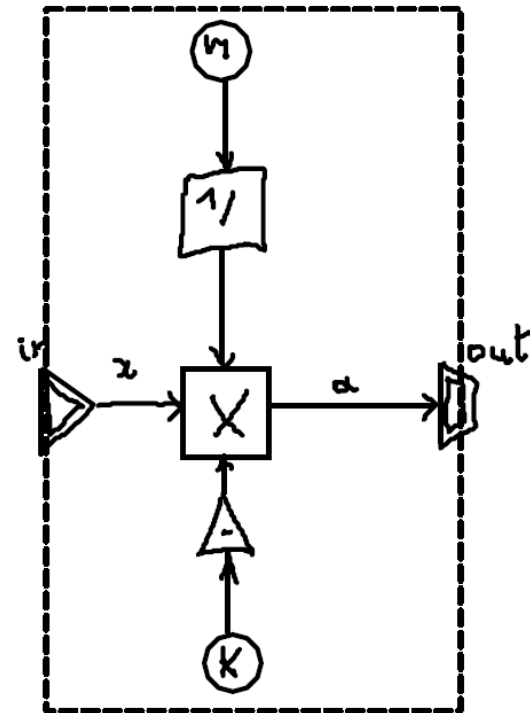
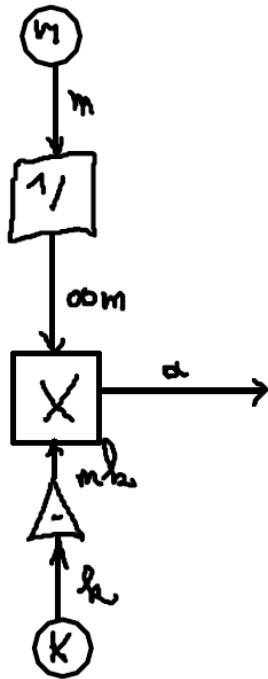
adding hierarchy to CBD formalisms

Time ↓	Hierarchy ↗		Semantics ↗	
	FLAT	SYNTAX	DENOTATIONAL "WHAT"	OPERATIONAL "HOW"
	CBD			
{NOW}	ALGEBRAIC (ALG-CBD)	 NO LOOPS WITH LOOPS	✓ ✓	✓
DN	DISCRETE-TIME (DT-CBD)	 APPROX	✓	✓
CR	CONTINUOUS-TIME (CT-CBD)		✓	✗

"flatten"

adding hierarchy to CBD formalisms

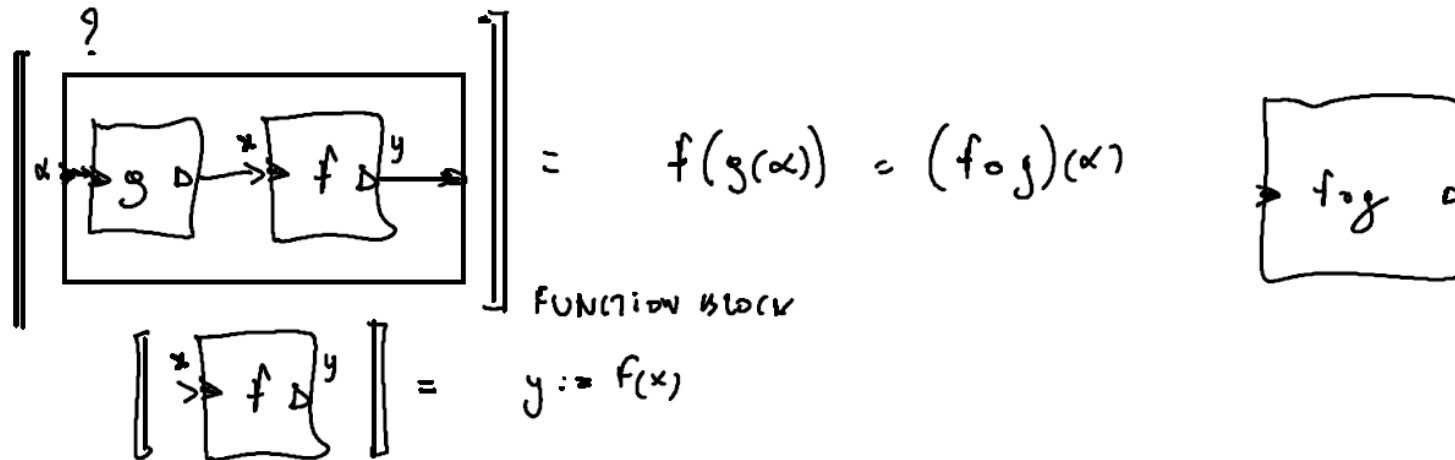
modularizing: introducing ports/interfaces



adding hierarchy to CBD formalisms

semantics?

HIERARCHY



“flattening”

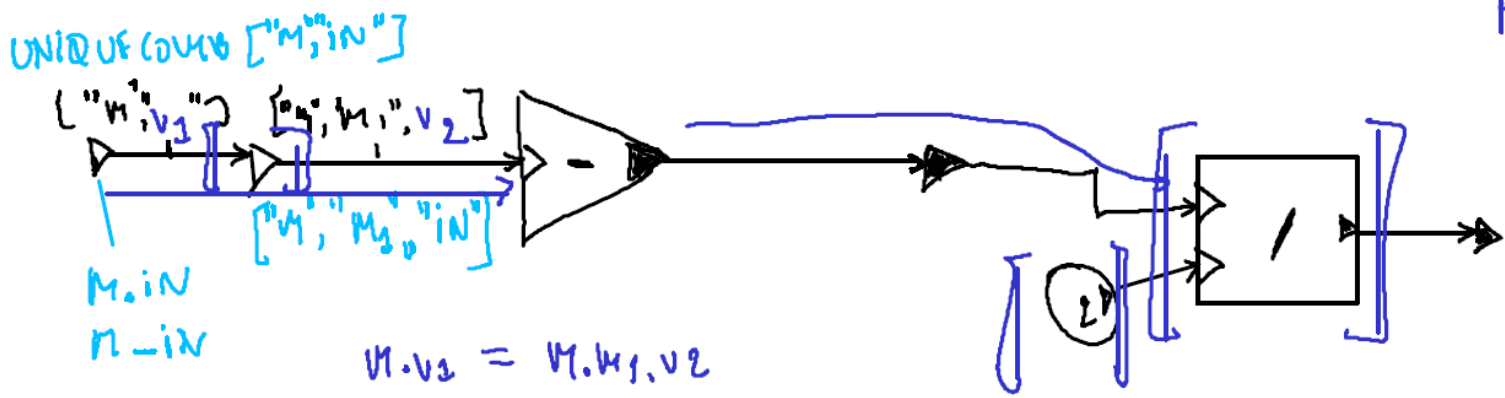


2 EVERY LEVEL :
DISTINCT NAMES



⁷¹FLATZEN¹⁷

I



Physical Systems Modelling

- Problem-Specific (technological)
- Domain-Specific (e.g., translational mechanical)
- (general) Laws of Physics
- Power Flow/Bond Graphs (physical: energy/power)
- Computationally a-causal
(Mathematical and Object-Oriented) ← **Modelica**
- Causal Block Diagrams (data flow)
- Numerical (Discrete) Approximations
- Computer Algorithmic + Numerical
(Floating Point vs. Fixed Point)
- As-Fast-As-Possible vs. Real-time (XiL)
- Hybrid (discrete-continuous) modelling/simulation
- Hiding IP: Composition of Functional Mockup Units (FMI)
- Dynamic Structure

Hybrid (discrete-continuous) modelling/simulation



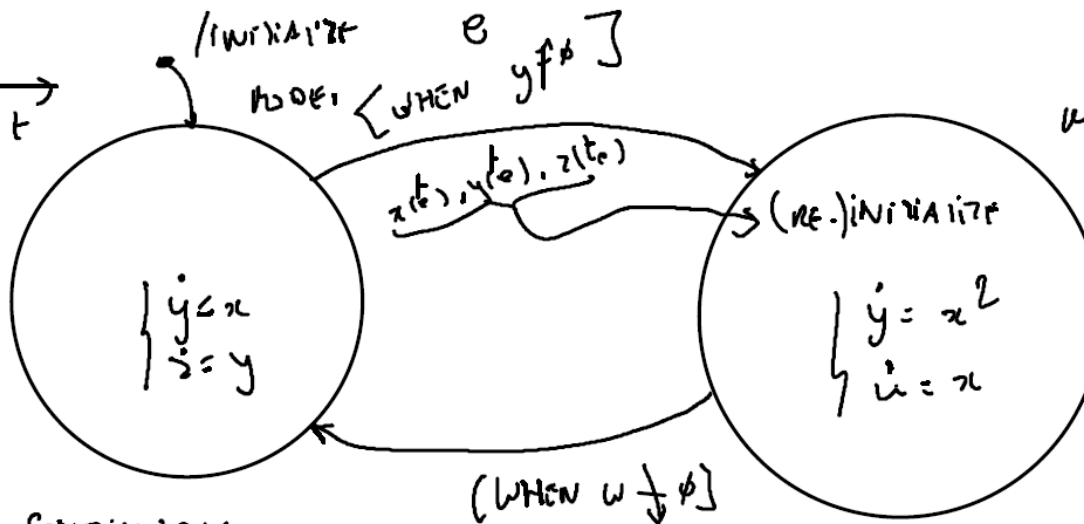
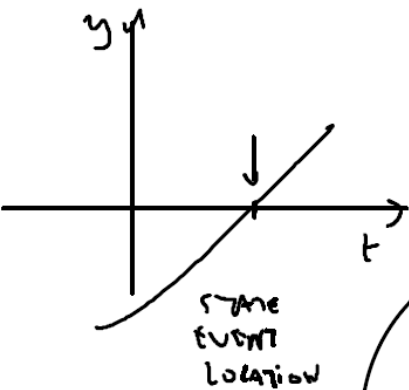
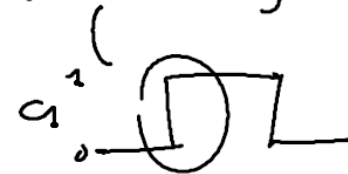
$$y = c_1 x + c_2 x^2$$

	c_1	c_2
MODE1	1	0
MODE2	0	1
	$c_1 + c_2 = 1$	$c_1, c_2 \in \{0, 1\}$

- MASS OPERATIONS (e.g. LFN, ALC-loop)
 - NUMERICAL PROBLEMS (CONTINUITY)

$$y = x$$

$$y = x^2$$

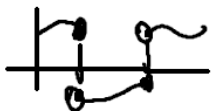


MODE2

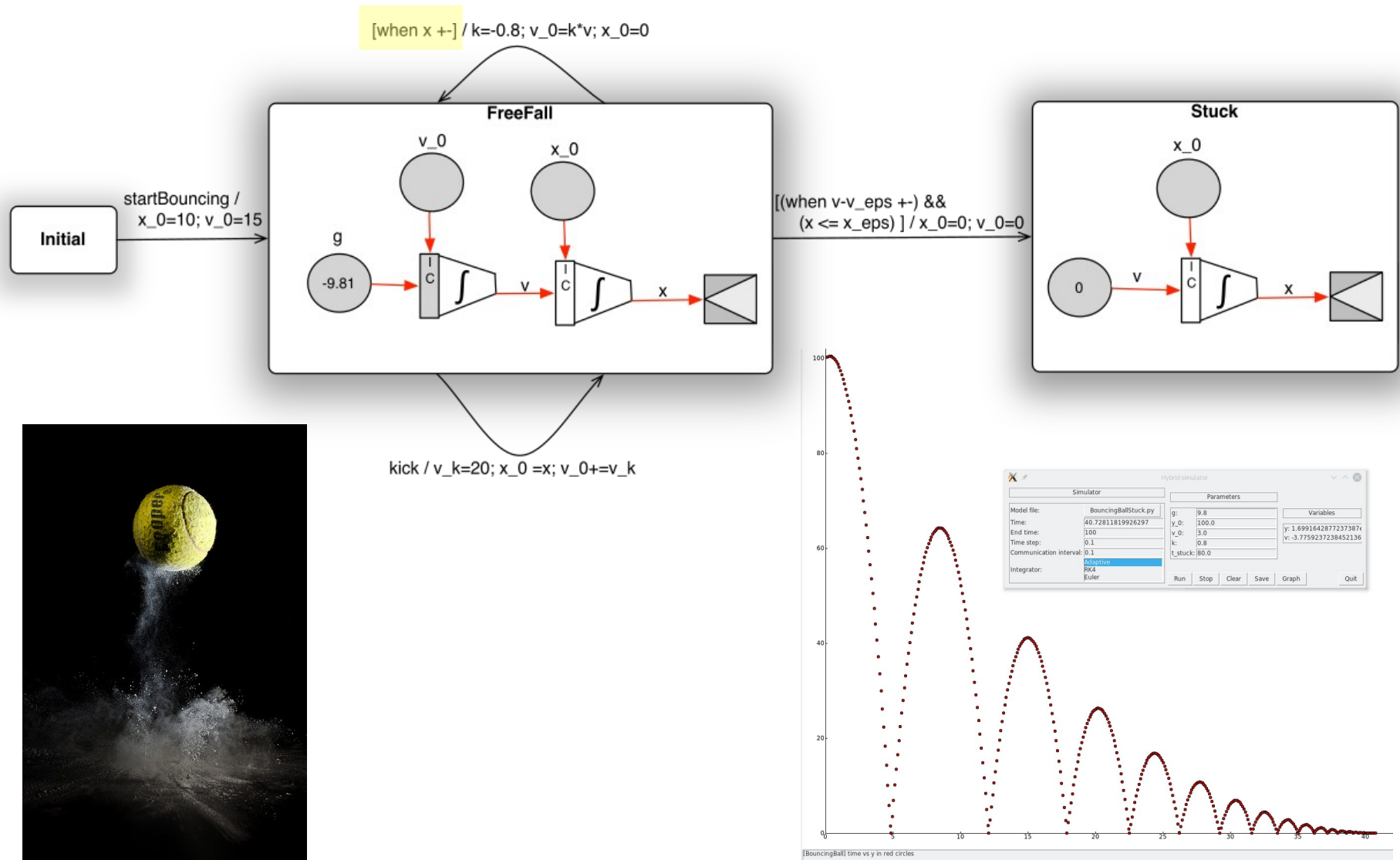
MODEL14

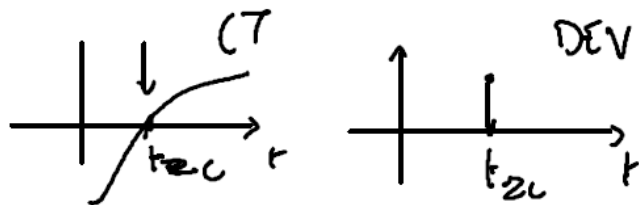
$$\begin{cases}
 \dot{y} = c_1 x + c_2 x^2 \\
 \dot{z} = c_1 y & (\dot{z} = 0 \text{ WHEN } w \text{ USED}) \\
 \dot{w} = c_2 x
 \end{cases}$$

SD PIECEWISE CONTINUOUS



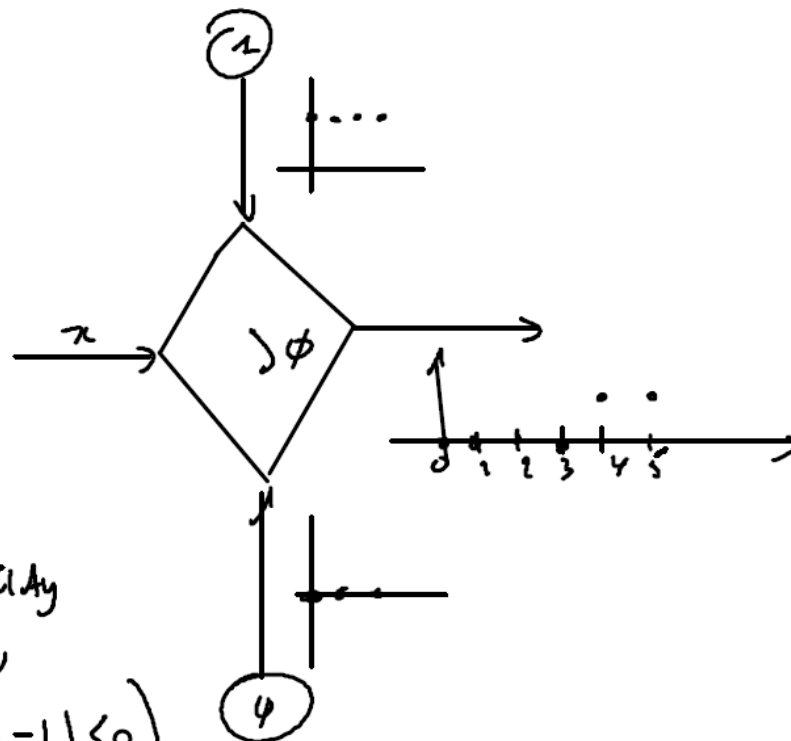
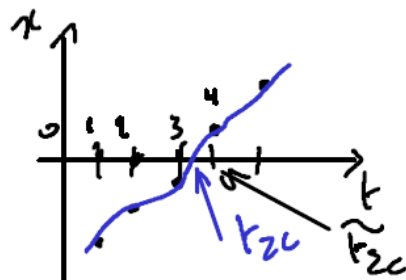
“bouncing ball” hybrid abstraction



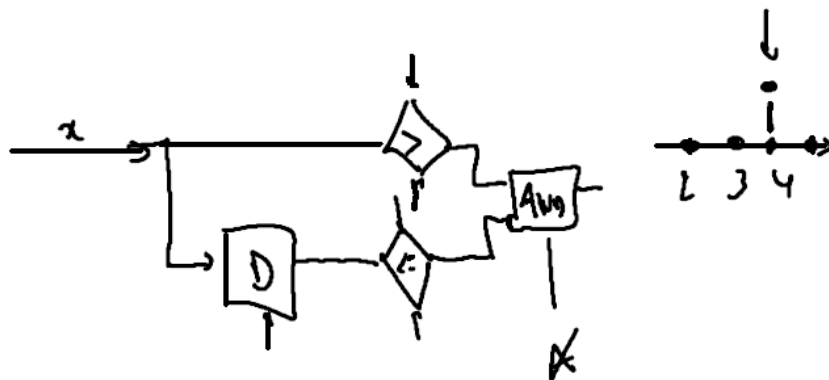


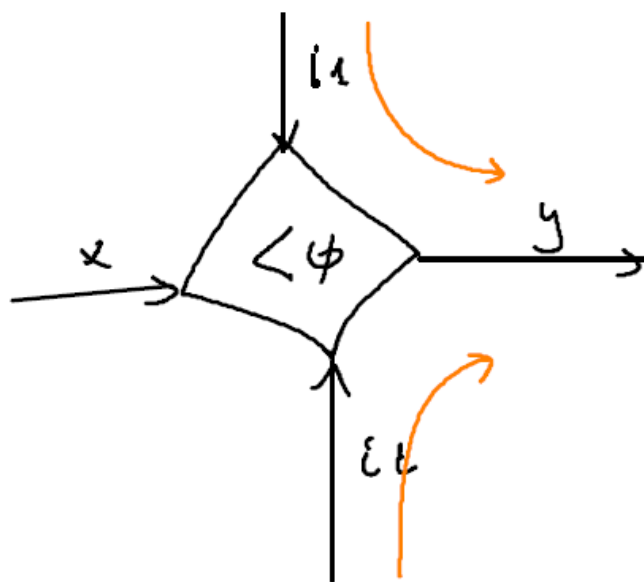
IF

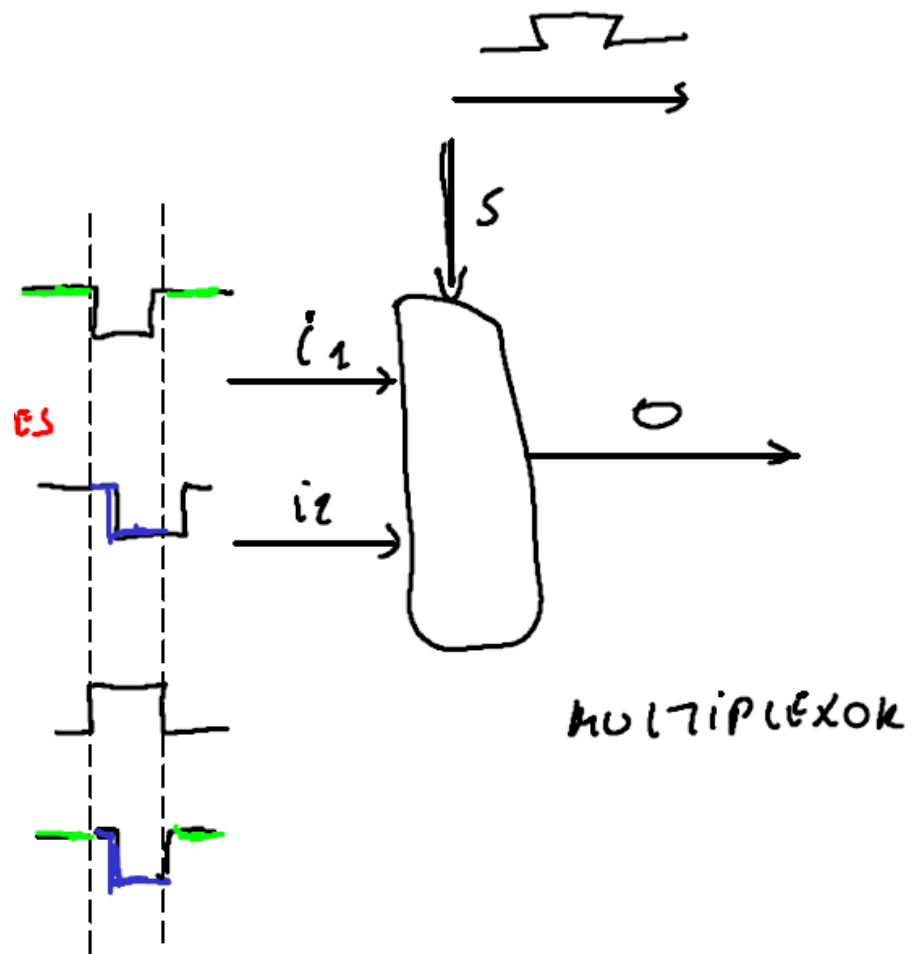
WHEN: ($1 \rightarrow 0$) WHEN ($x(t) \neq \phi$)
 $\rightarrow$ DEV signal.

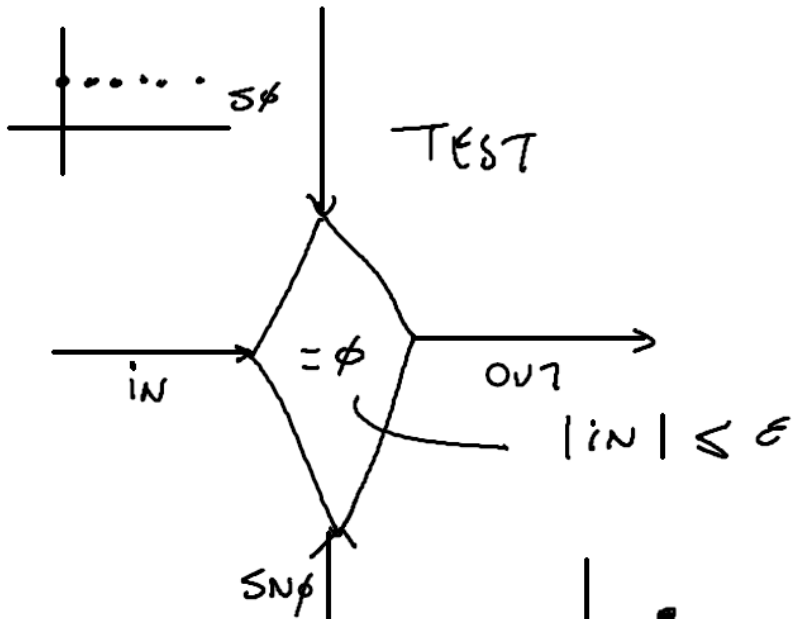


$t_{zc}?$: ($\tilde{x}(\tilde{t}_{zc}) > 0$) & ($\tilde{x}(\tilde{t}_{zc}-1) \leq 0$)









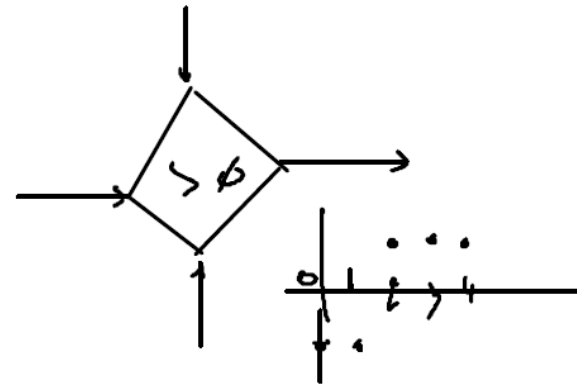
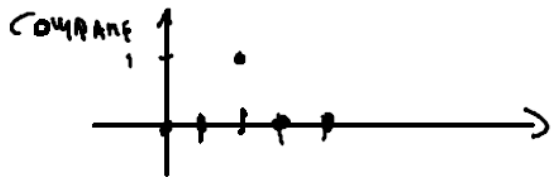
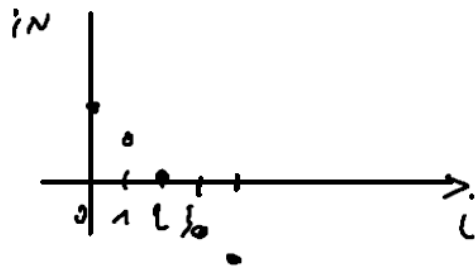
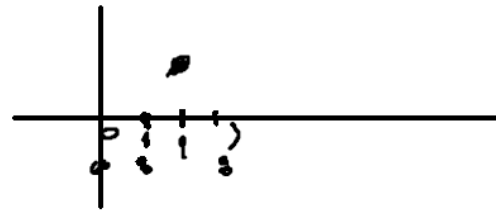
$\mathbb{N}, \mathbb{Q}$
int

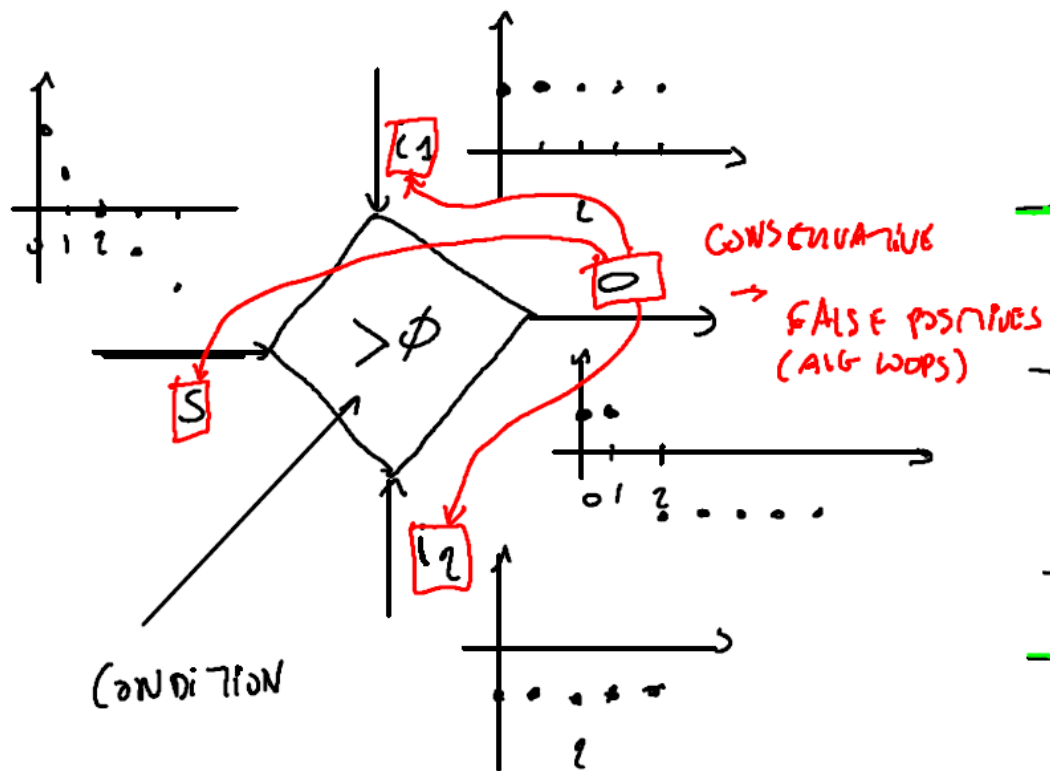
$$i_1 == i_2$$

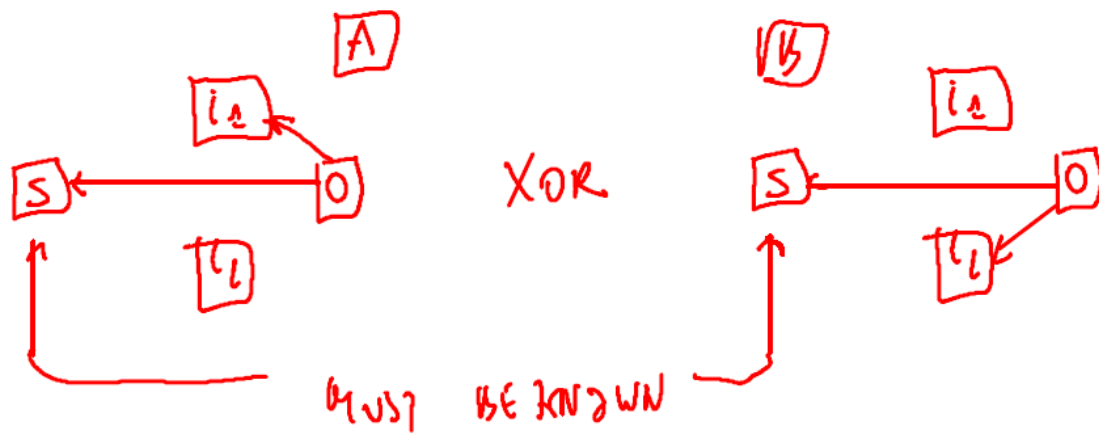
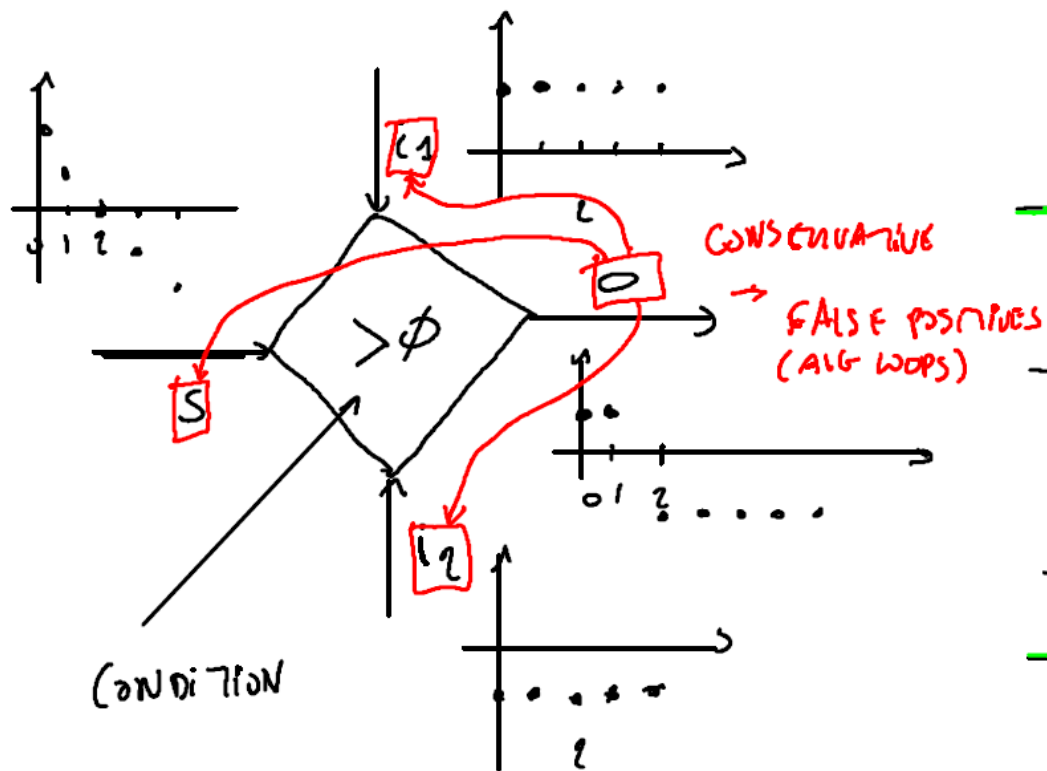
$\mathbb{R}$
float, double

$$r_1 == r_2$$

$$|r_1 - r_2| \leq \epsilon$$







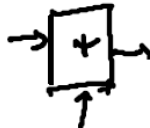
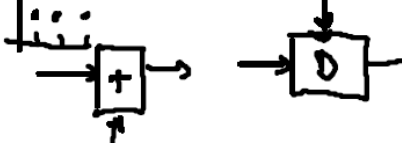
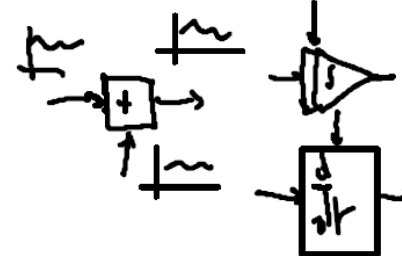
1. BUILD DEP. GRAPH UP TO S
2. PARTIAL SORT/LOOP DEP UP TO S
3. COMPLETE DEP GRAPH WITH A OR B
4. FULL SORT/LOOP DEPEND

MULTIPLE TEST BLOCK
FIXED POINT

Physical Systems Modelling

- Problem-Specific (technological)
- Domain-Specific (e.g., translational mechanical)
- (general) Laws of Physics
- Power Flow/Bond Graphs (physical: energy/power)
- Computationally a-causal
(Mathematical and Object-Oriented) ← **Modelica**
- Causal Block Diagrams (data flow)
- Numerical (Discrete) Approximations
- Computer Algorithmic + Numerical
(Floating Point vs. Fixed Point)
- As-Fast-As-Possible vs. Real-time (XiL)
- Hybrid (discrete-continuous) modelling/simulation
- Hiding IP: Composition of Functional Mockup Units (FMI)
- Dynamic Structure

++ Dynamic Structure

TIME ↓	HIERARCHY ↗		SEMANTICS ↗	
	FLAT CBD	SYNTAX ✓ ~~FLATTEN~~	DENOTATIONAL "WHAT"	OPERATIONAL "HOW"
{NOW}	ALGEBRAIC (ALG-CBD)	 NO LOOPS WITH LOOPS	✓	✓
			✓	✓
IN	DISCRETE-TIME (DT-CBD)		✓	✓
IR	CONTINUOUS-TIME (CT-CBD)		✓	✓

Dynamic Structure: CBD model blocks/signals can be added/deleted during simulation

aka Dynamics Structure DS-(ALG?|DT|CT)CBD

operational semantics

```
i = 0
```

```
while (not end_condition(i, ...)):
```

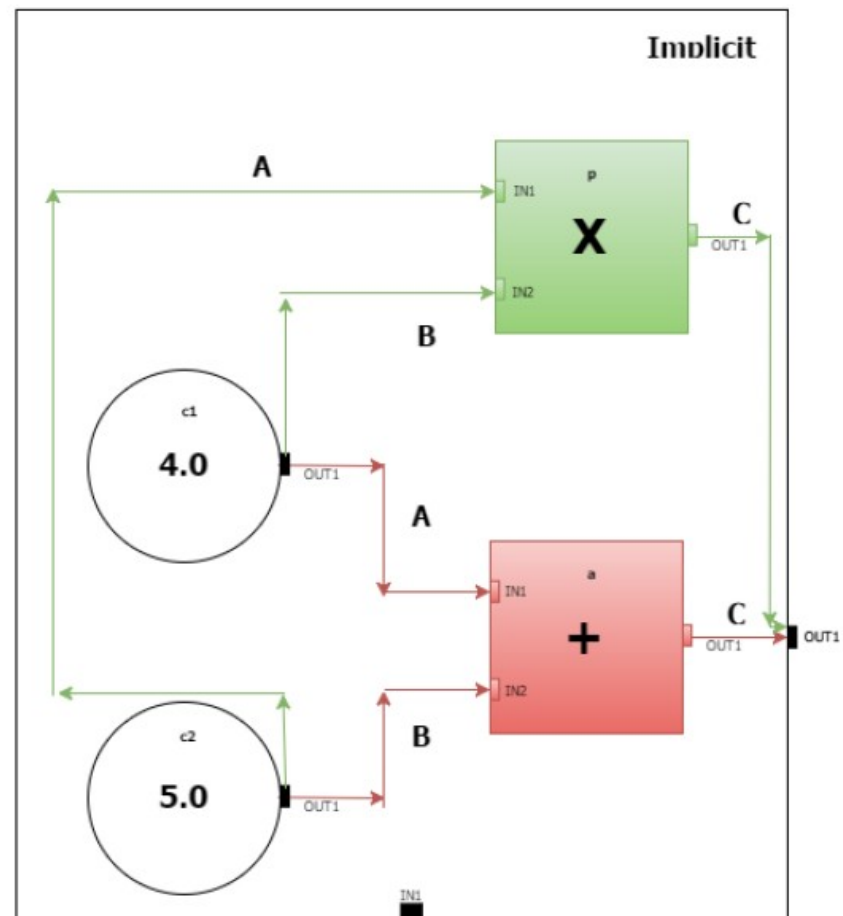
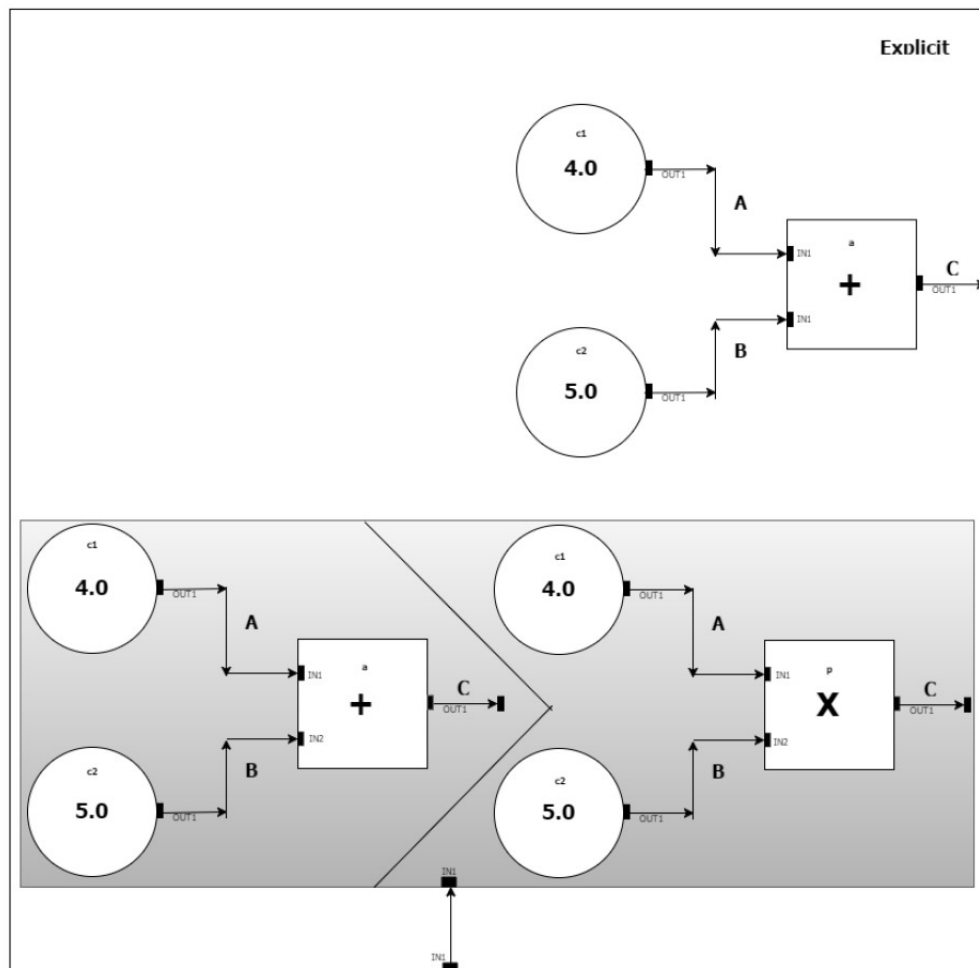
```
    depGraph = buildDepGraph(CBD)
```

```
    schedule = loopDetectAndTopSort(depGraph)
```

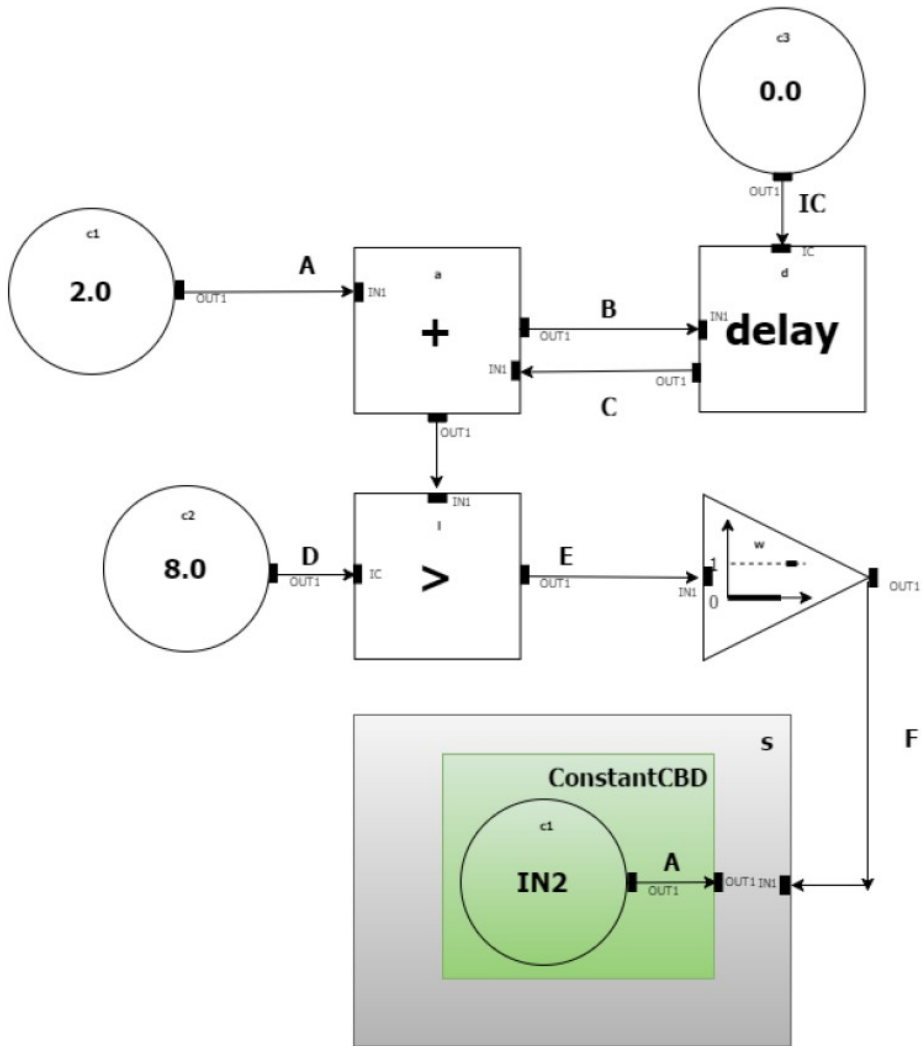
```
    for gblock in schedule:
```

```
        gblock.compute()
```

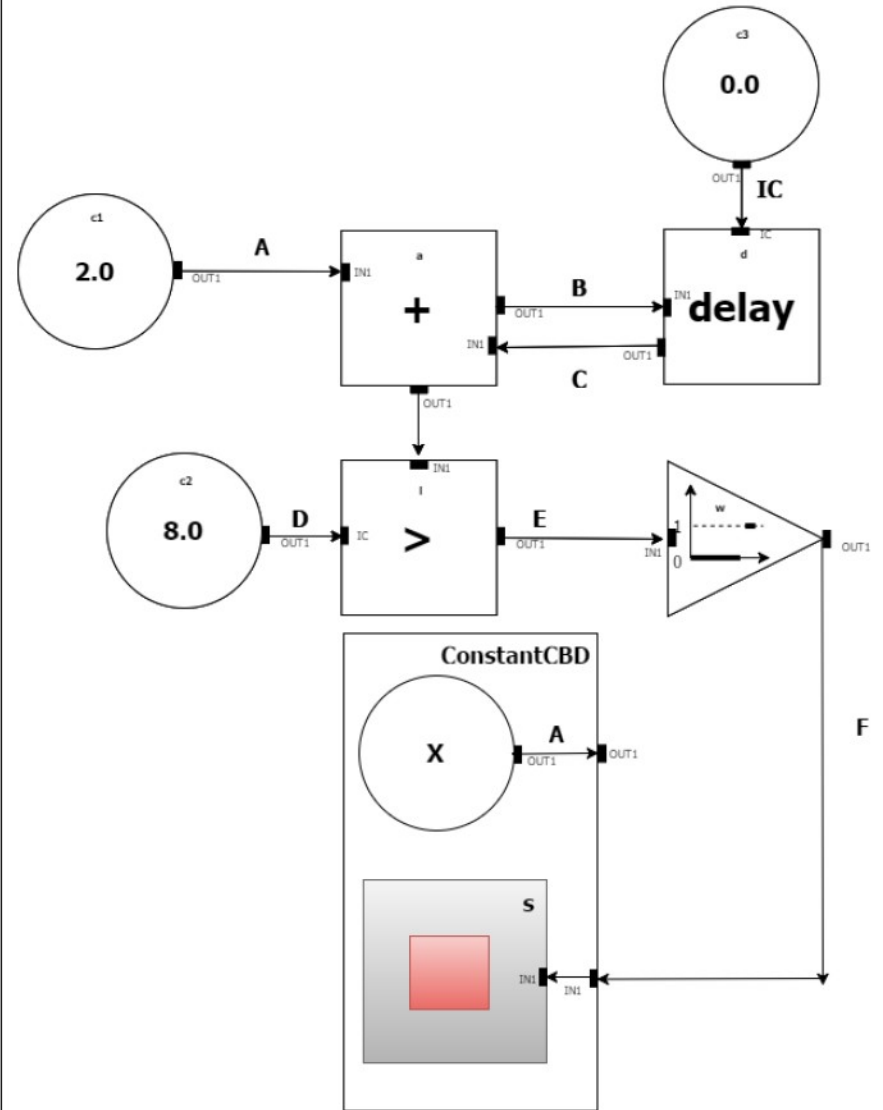
```
    i++
```



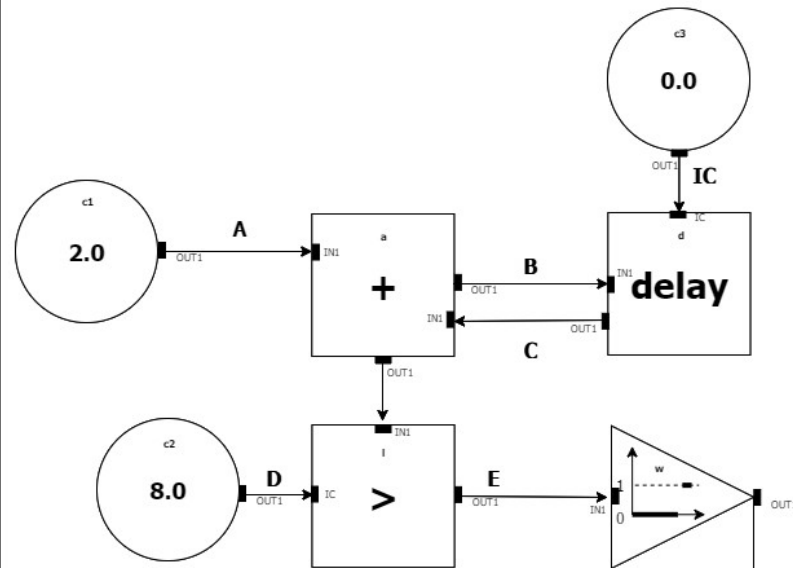
Addition



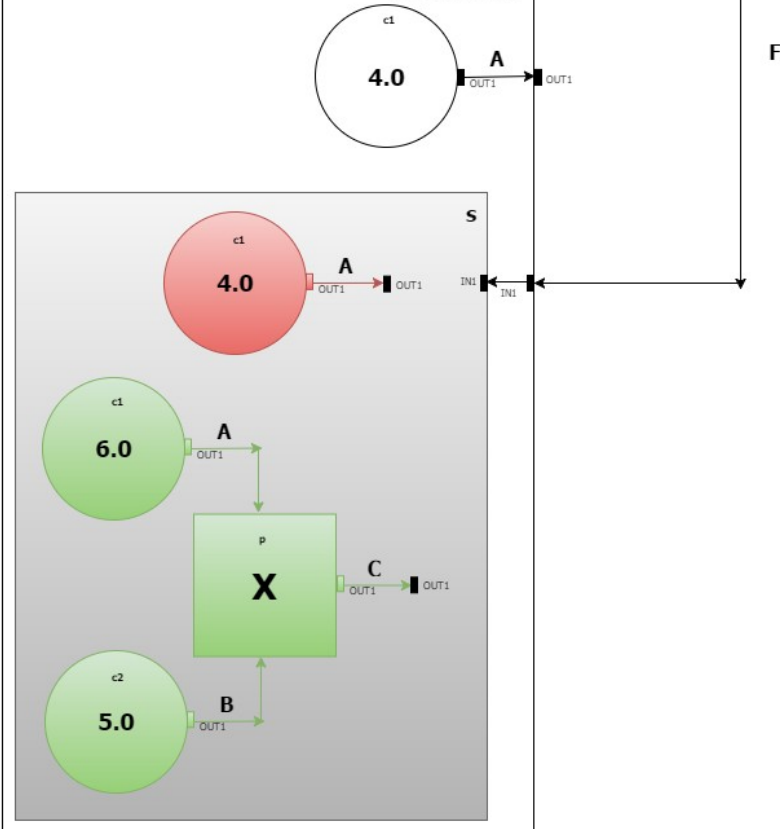
Removal

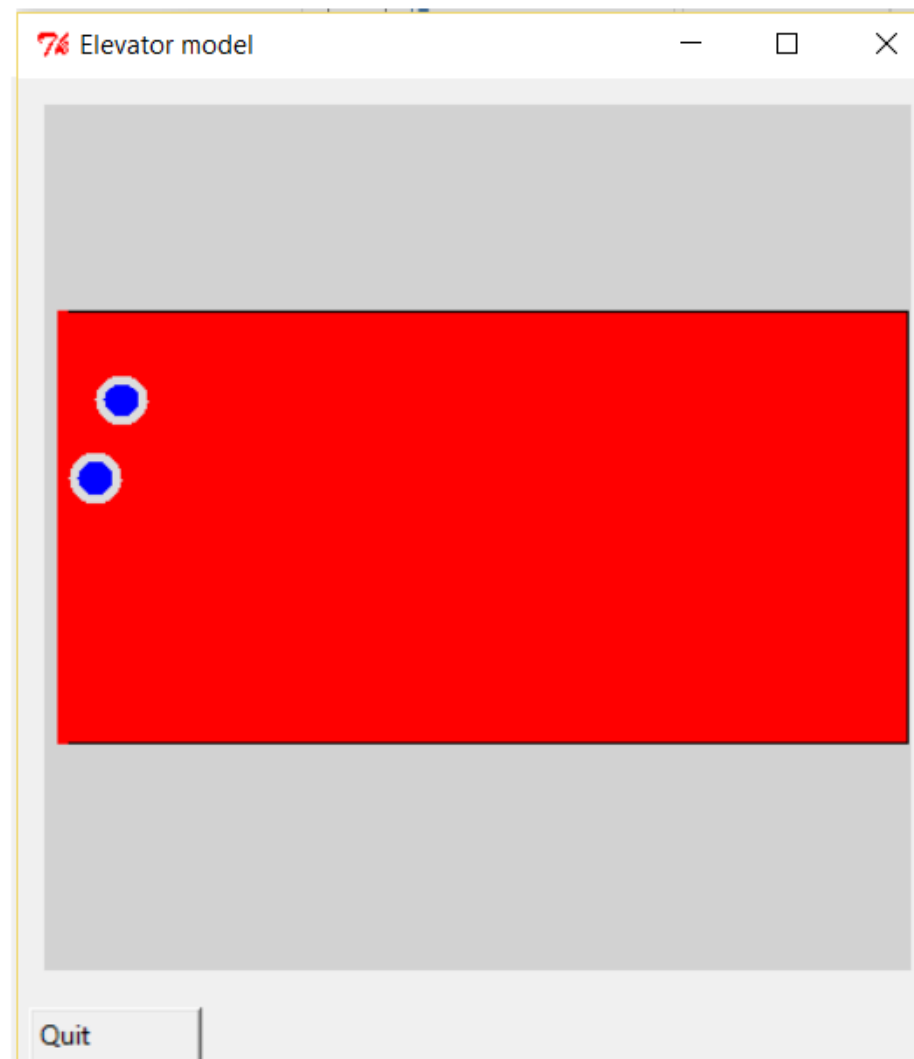
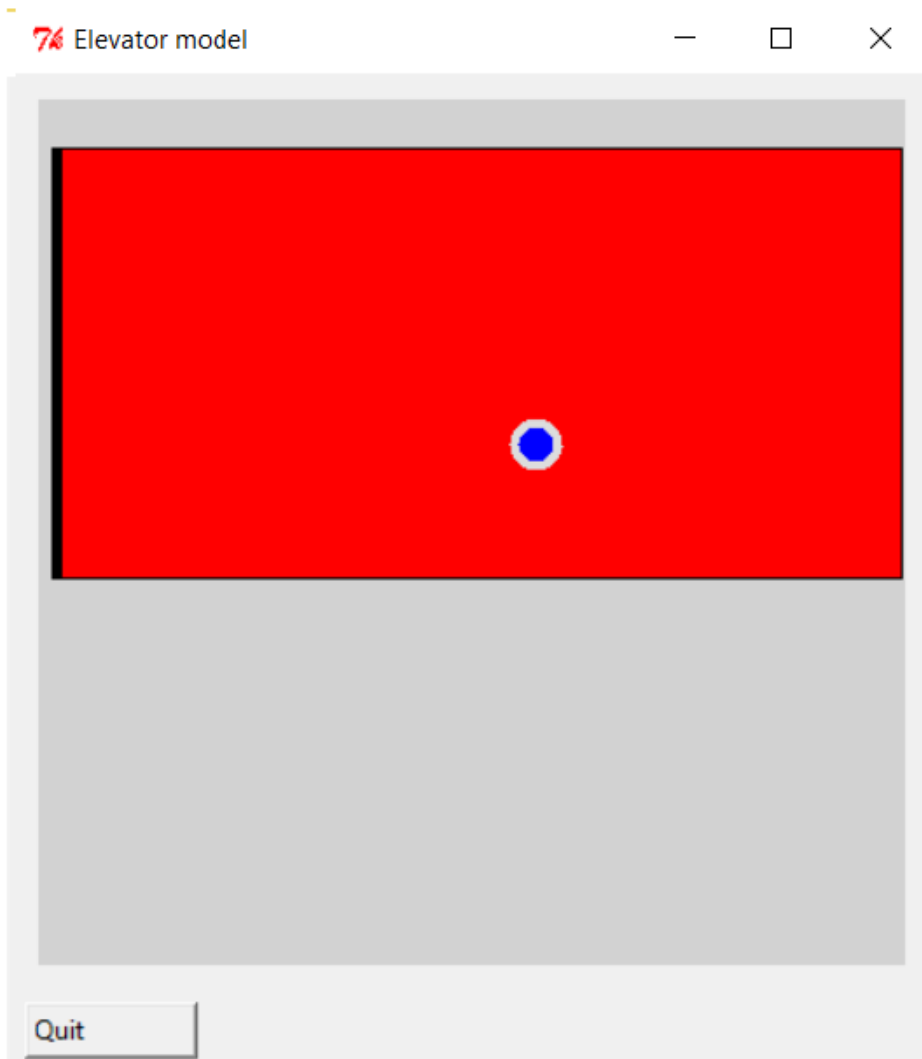


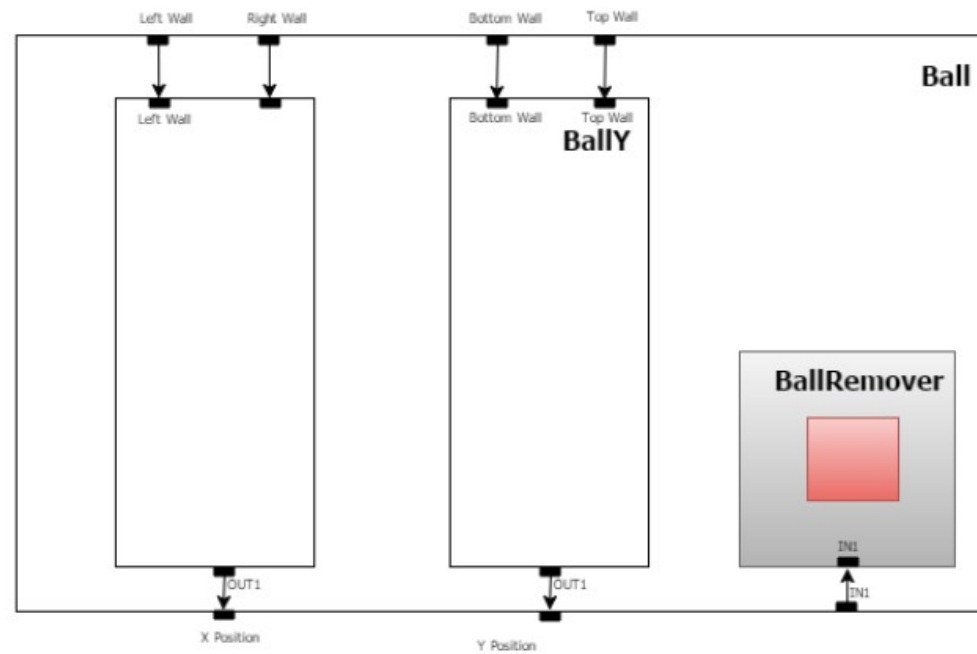
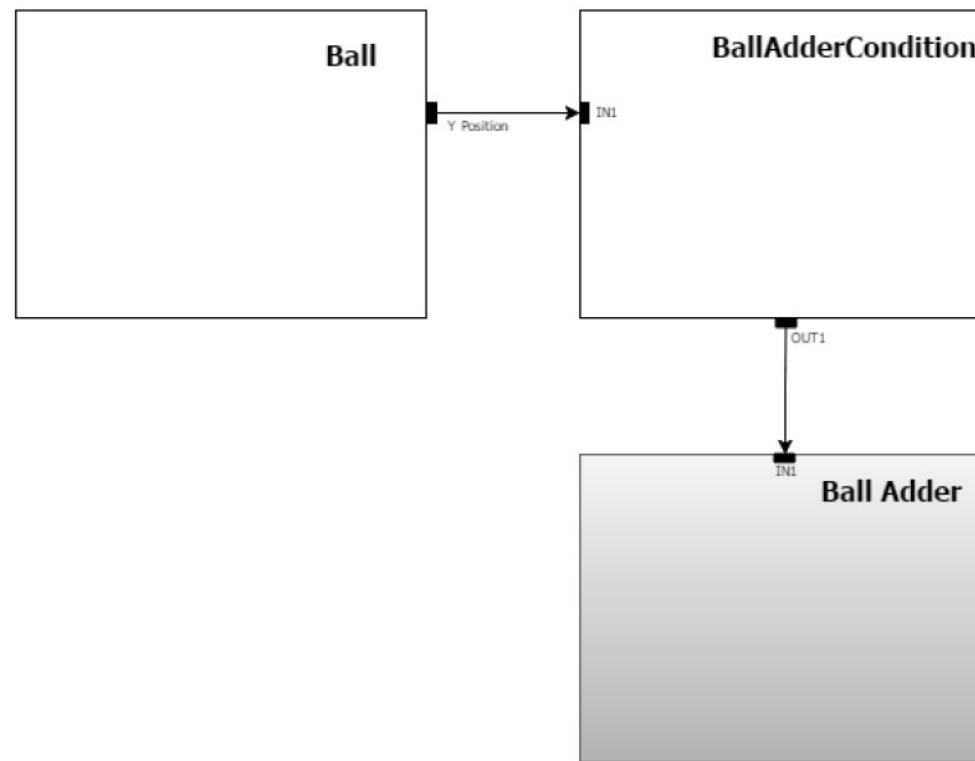
Behaviour



ConstantCBD







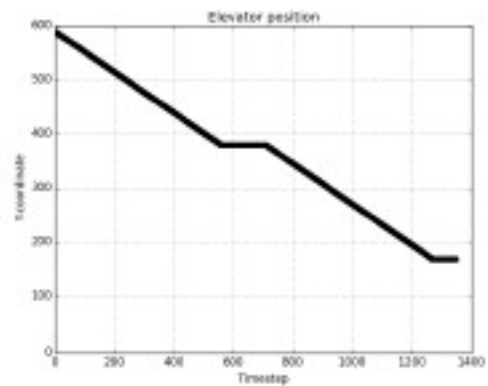


Figure 4.10: Y-position of the elevator

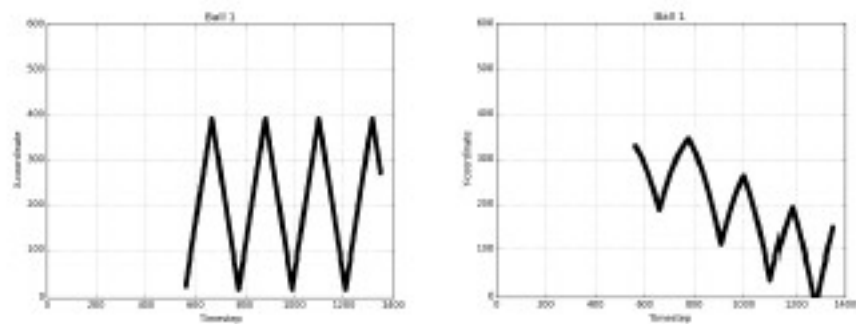


Figure 4.11: Position of ball 1

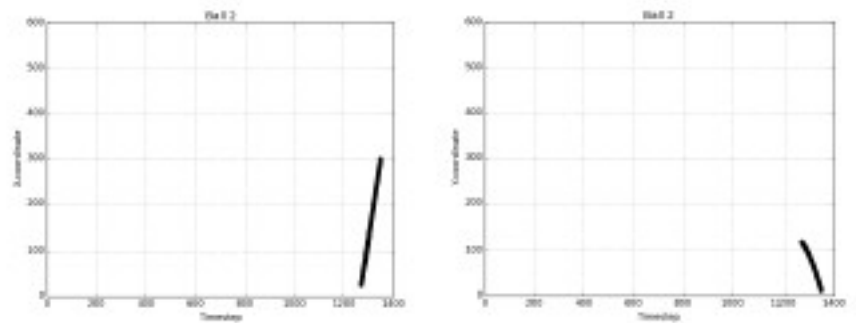
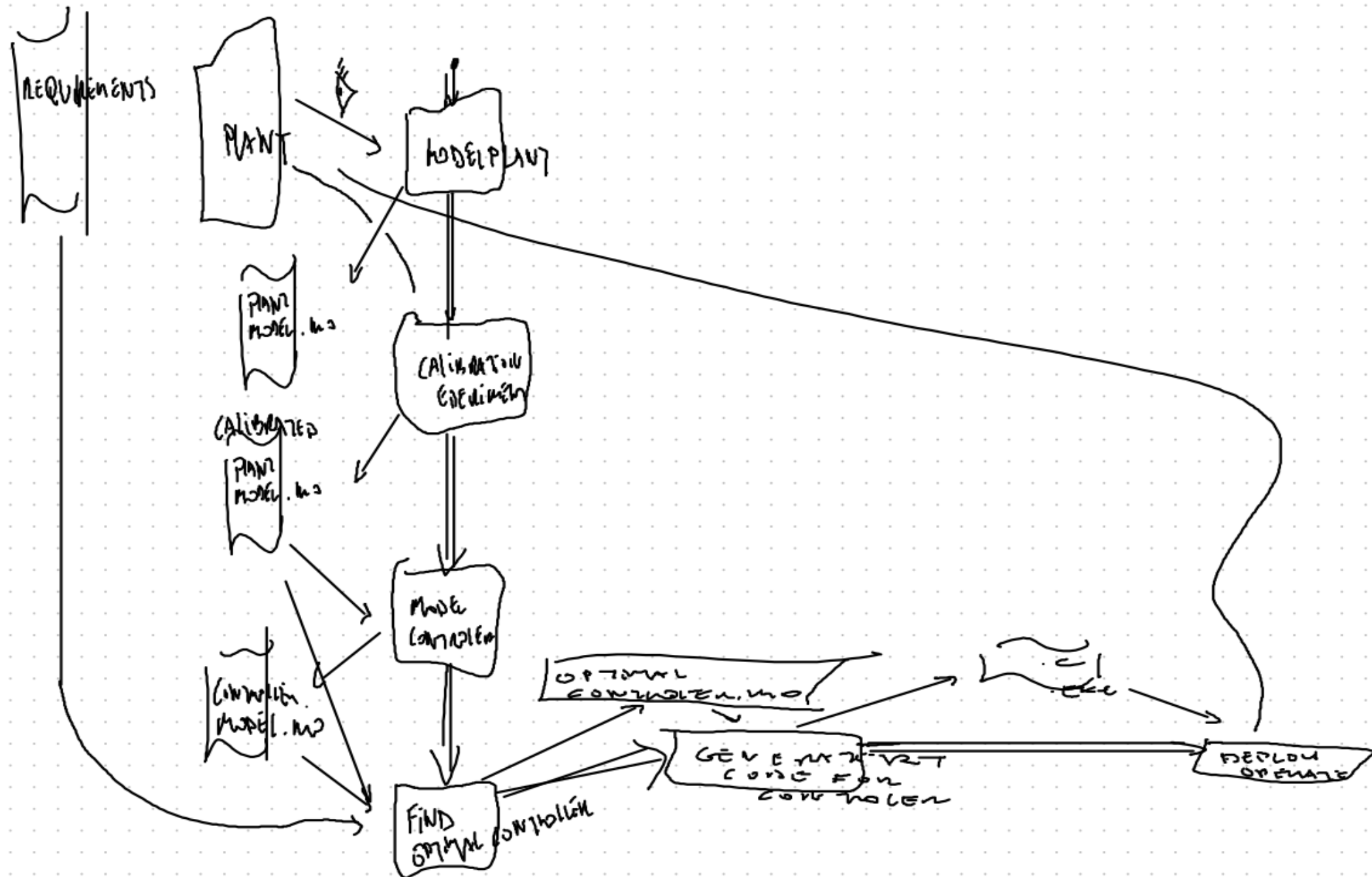


Figure 4.12: Position of ball 2

Physical Systems Modelling

- Problem-Specific (technological)
- Domain-Specific (e.g., translational mechanical)
- (general) Laws of Physics
- Power Flow/Bond Graphs (physical: energy/power)
- Computationally a-causal
(Mathematical and Object-Oriented) ← **Modelica**
- Causal Block Diagrams (data flow)
- Numerical (Discrete) Approximations
- Computer Algorithmic + Numerical
(Floating Point vs. Fixed Point)
- As-Fast-As-Possible vs. Real-time (XiL)
- Hybrid (discrete-continuous) modelling/simulation
- Hiding IP: Composition of Functional Mockup Units (FMI)
- Dynamic Structure

Workflow (Process Model)

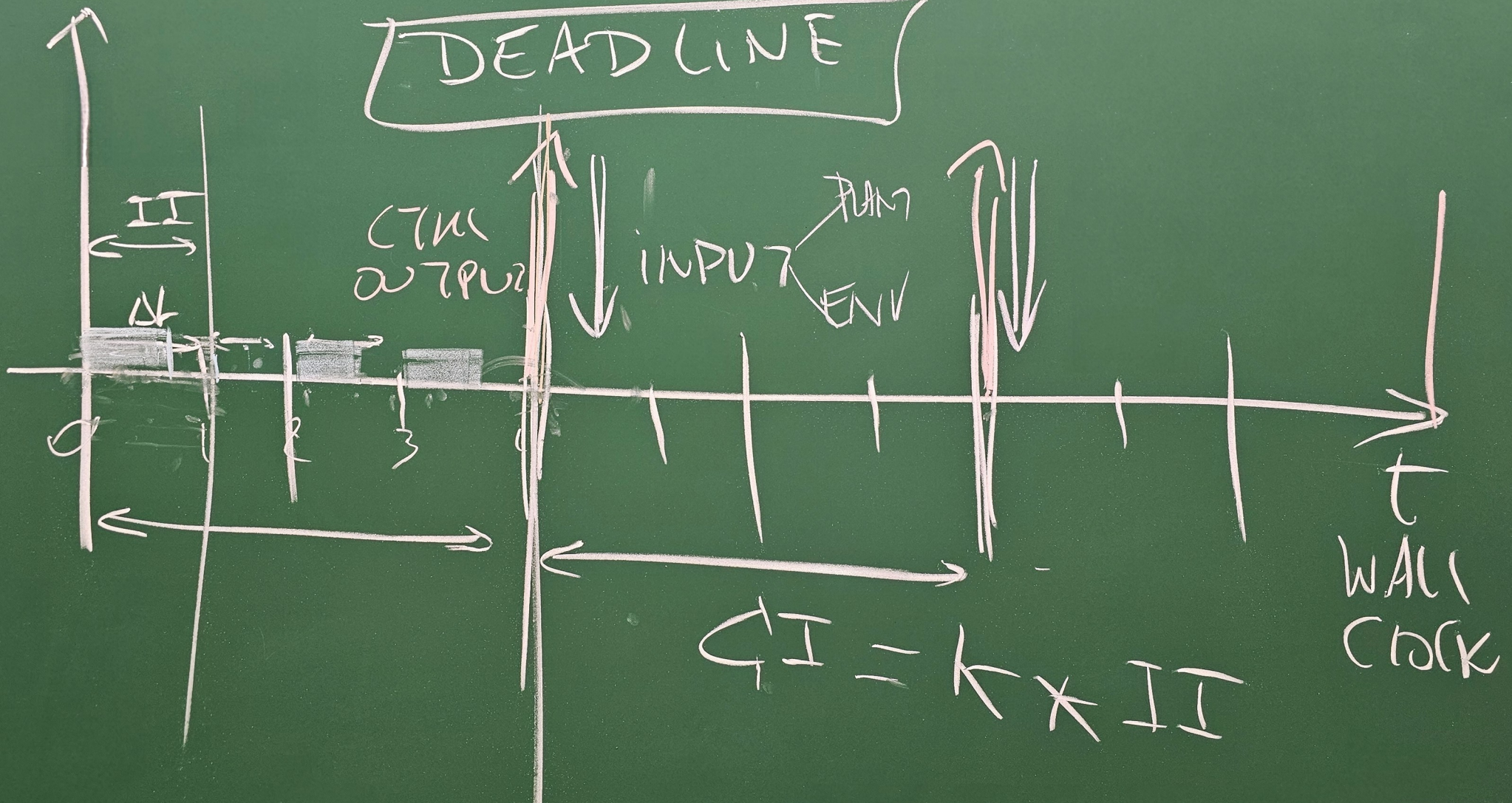


As-Fast-As-Possible vs. Real-time

MEE7

REAL-TIME

DEADLINE



As-Fast-As-Possible vs. Real-time

$i = 0$

DT-CBD

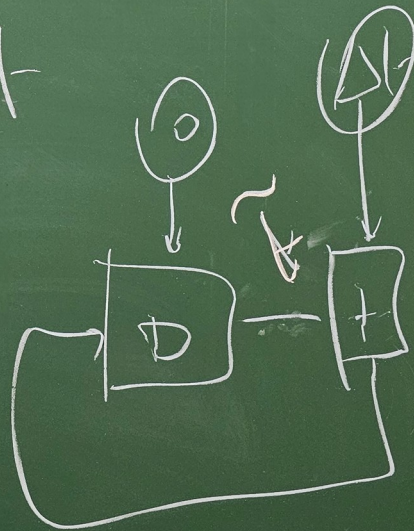
SCHEDULE = SORTNODES(DEP(CBD))

WHILE (NOT END-COND)

[FOR Gblock in SCHEDULE:
 Gblock.compute()

 i++

$$\tilde{t} = i \times \Delta t$$



Operational Semantics of DT-CBD

(“soft” real-time, very naive)

```
i = 0
```

```
while (not end_condition(i, ...)):
```

```
    depGraph = buildDepGraph(CBD)
```

```
    schedule = loopDetectAndTopSort(depGraph)
```

```
    for gblock in schedule:
```

```
        gblock.compute()
```

```
    i++
```

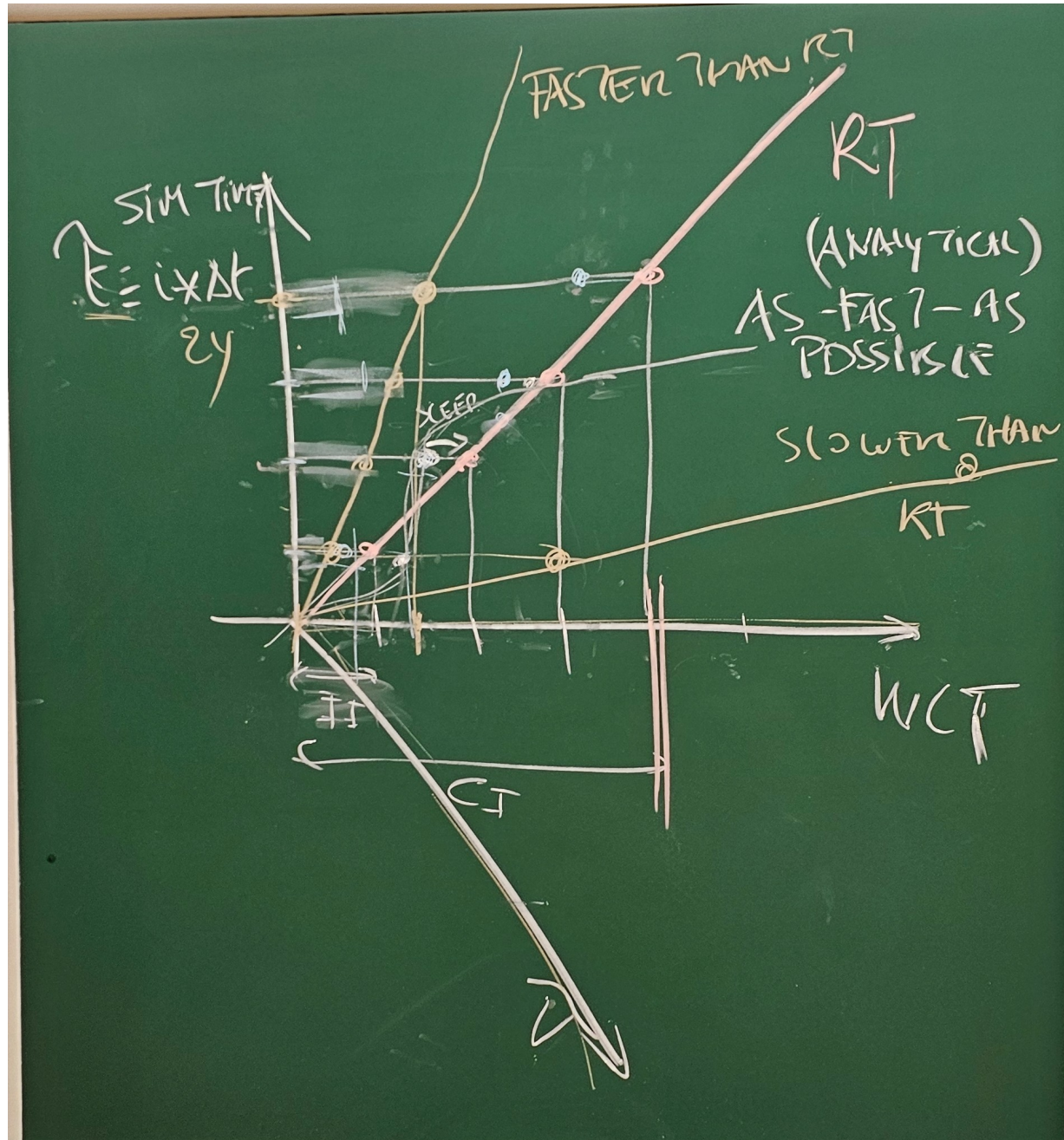
```
    sleep_until_next_CI()
```

```
    produce (control) output
```

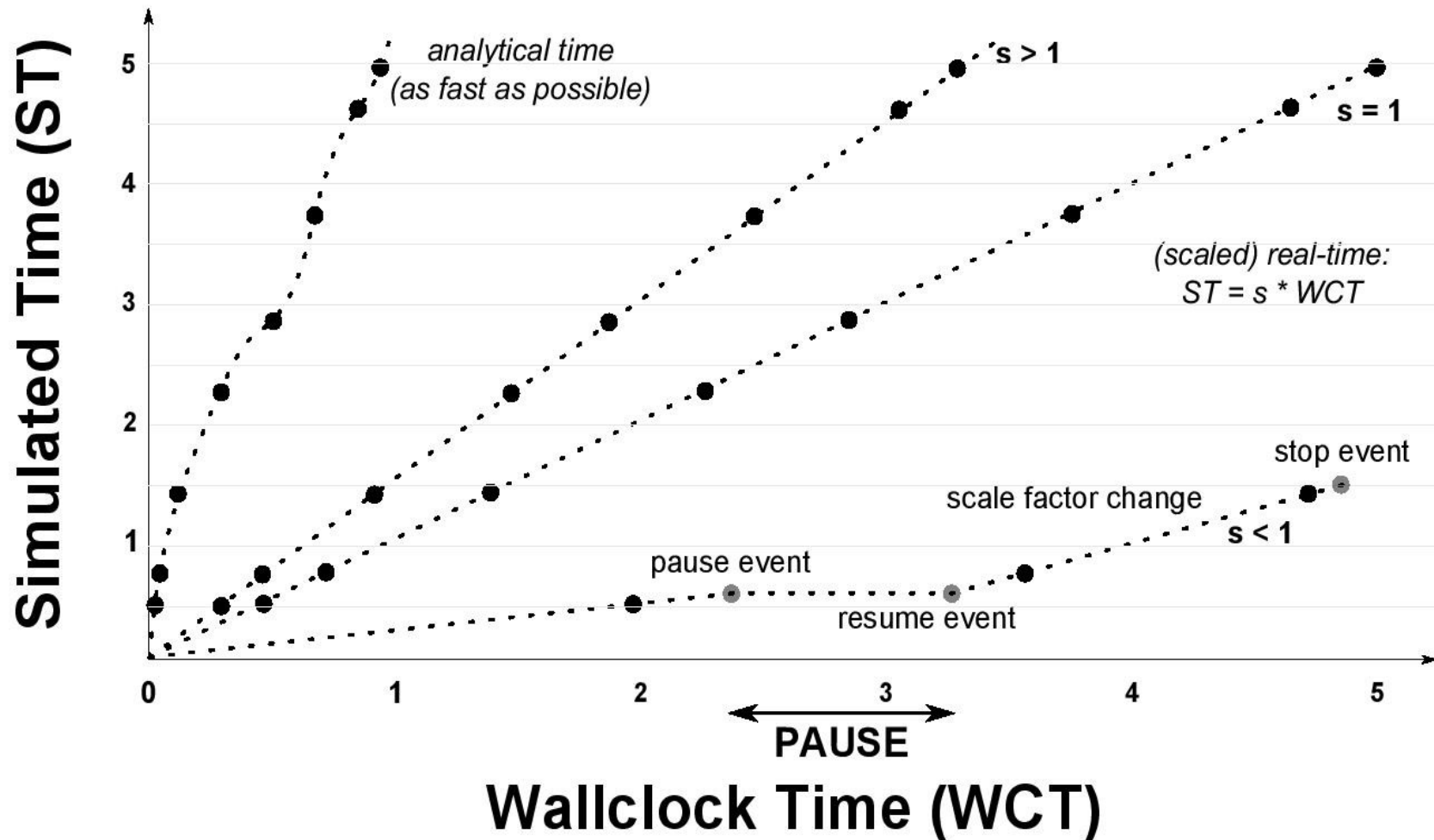
```
else:
```

```
    exit()
```

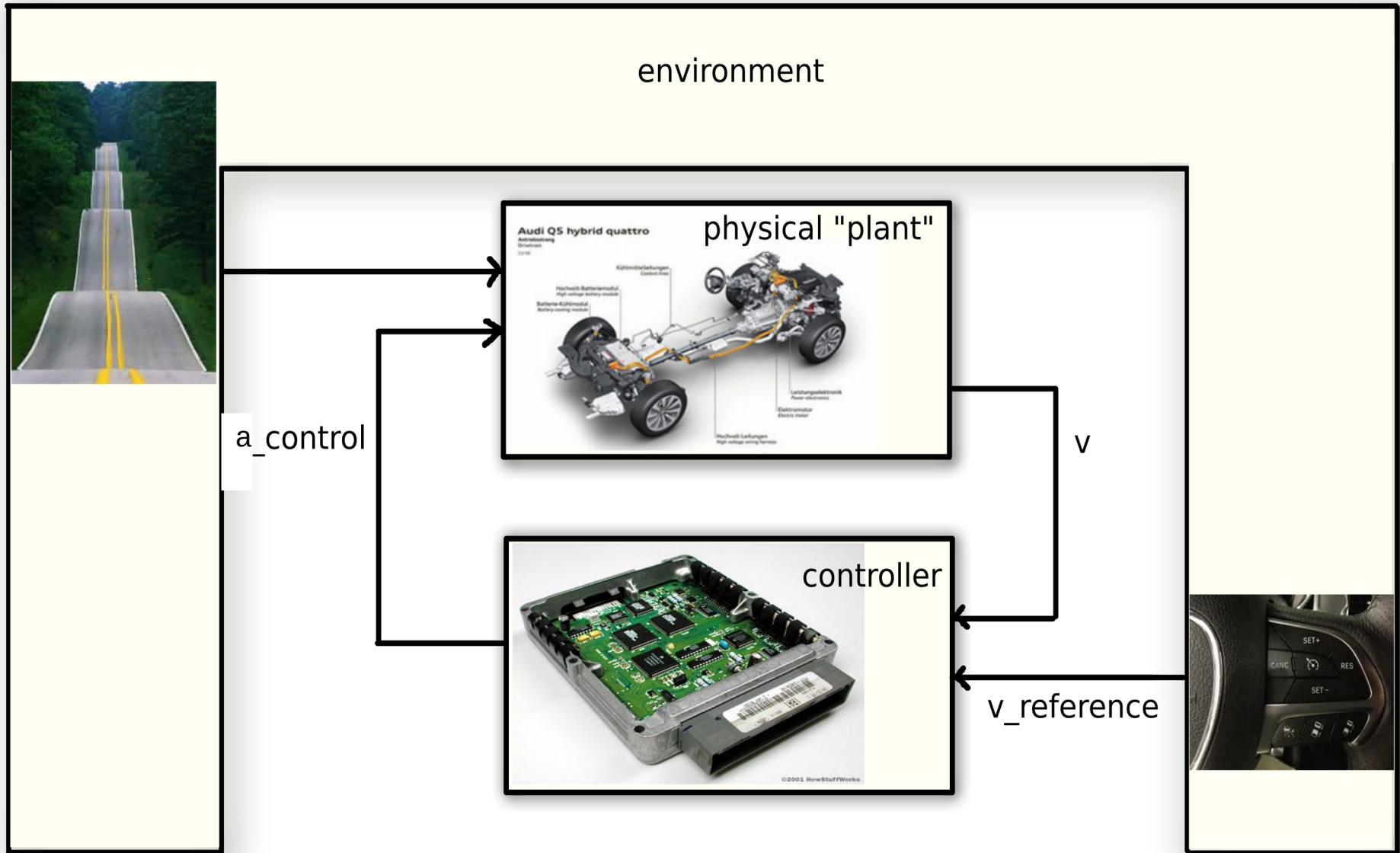
As-Fast-As-Possible vs. Real-time



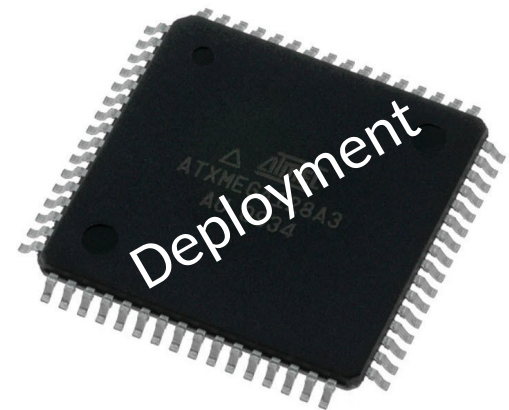
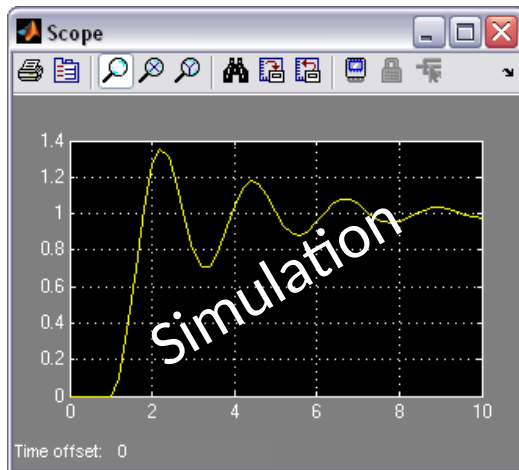
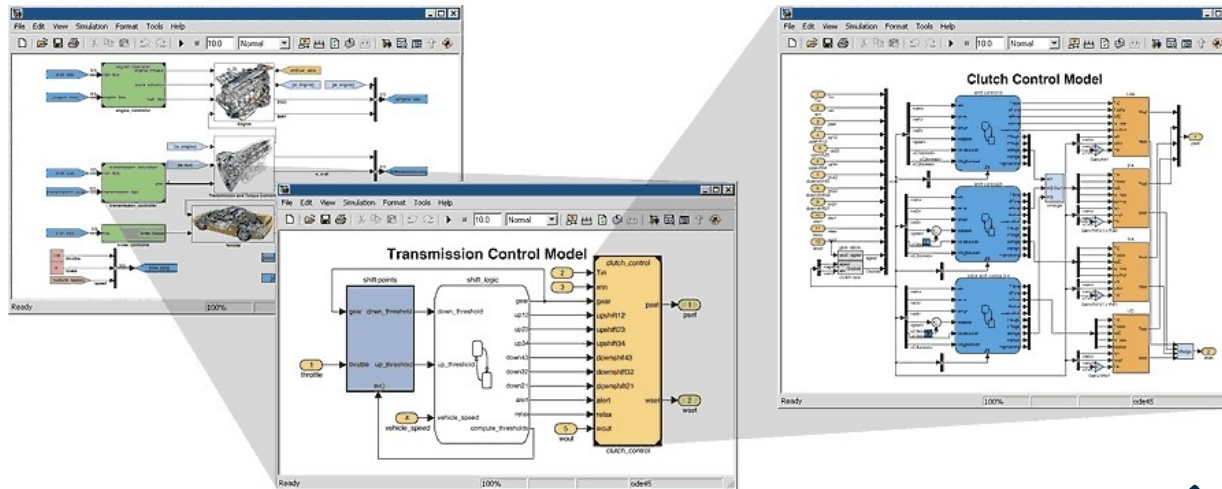
As-Fast-As-Possible vs. Real-time



Model-Based Systems Engineering (MBSE)

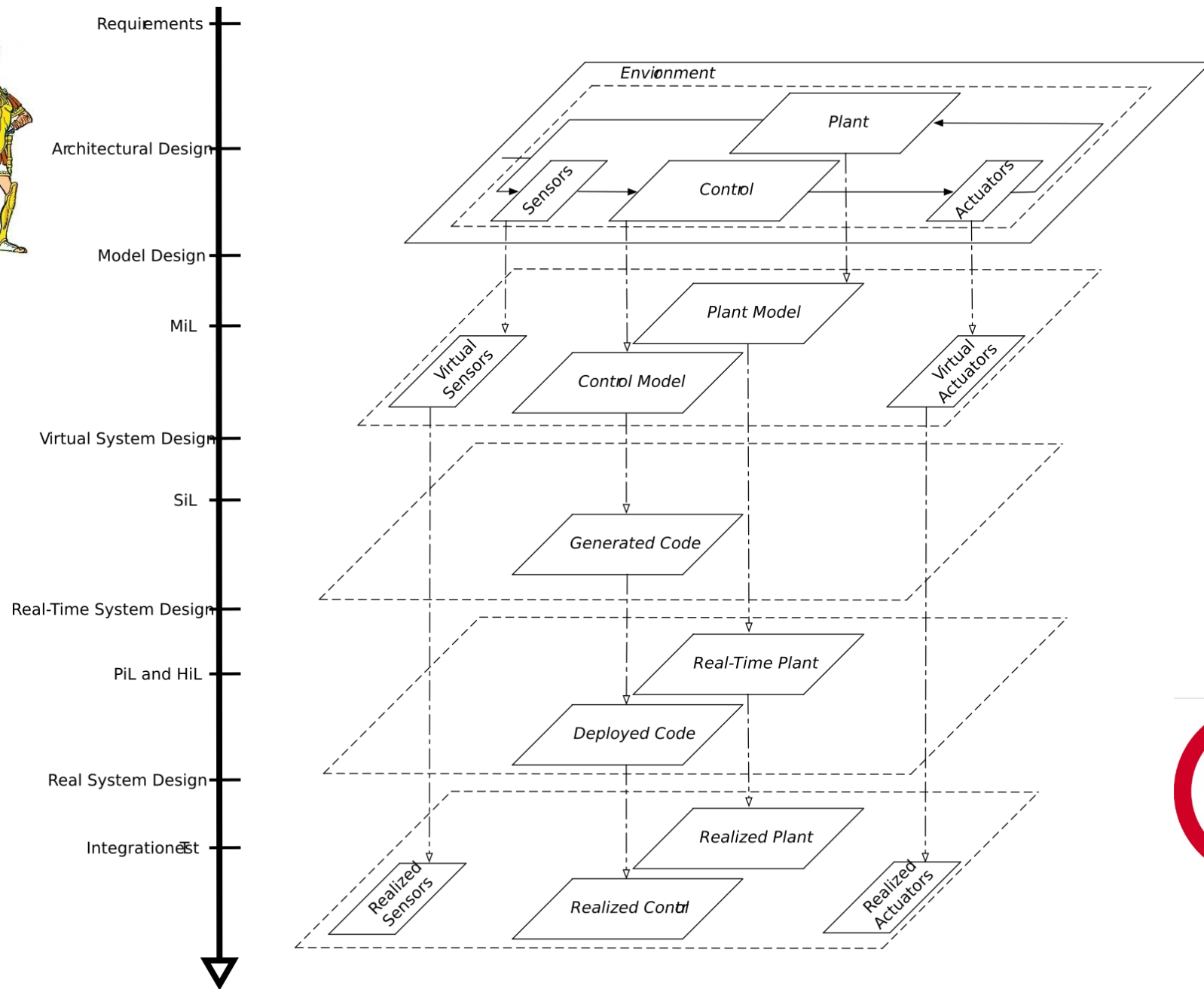
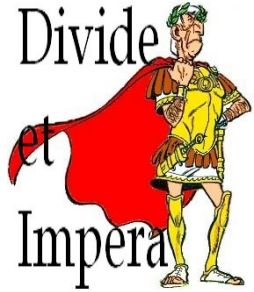


Model-Based Systems Engineering (MBSE)



MiL, HiL, SiL, ...

XiL: X = Model, Software, Processor, Hardware



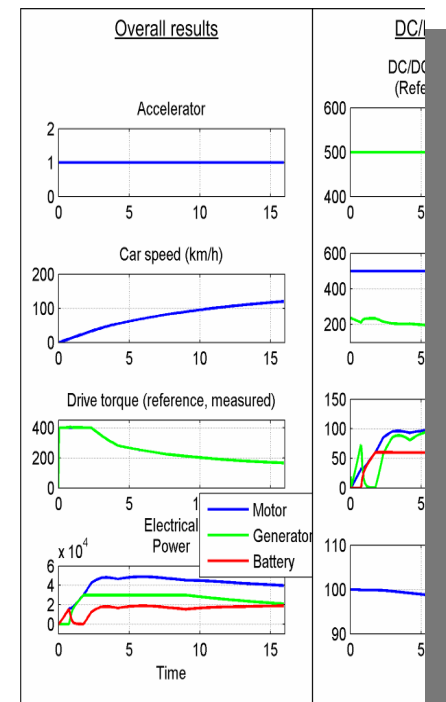
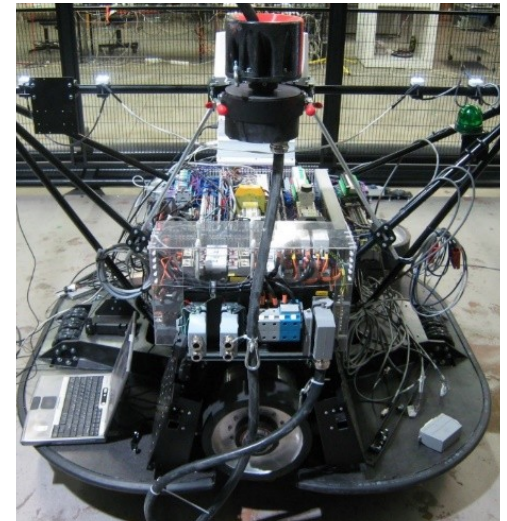
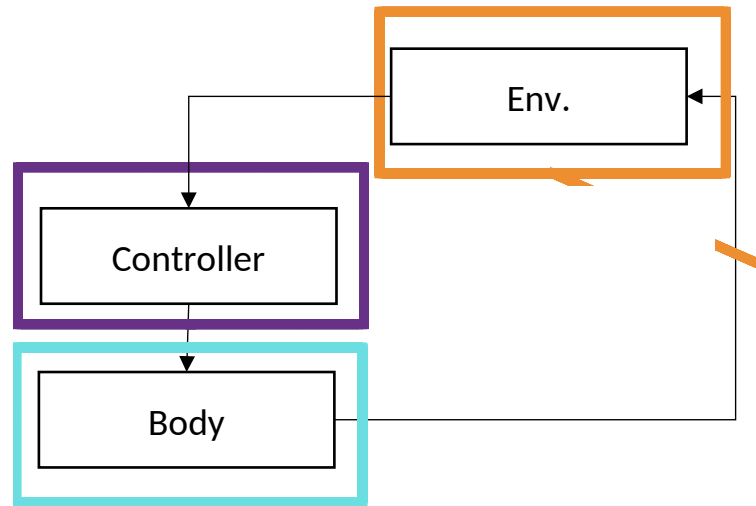
vertical consistency!



Physical Systems Modelling

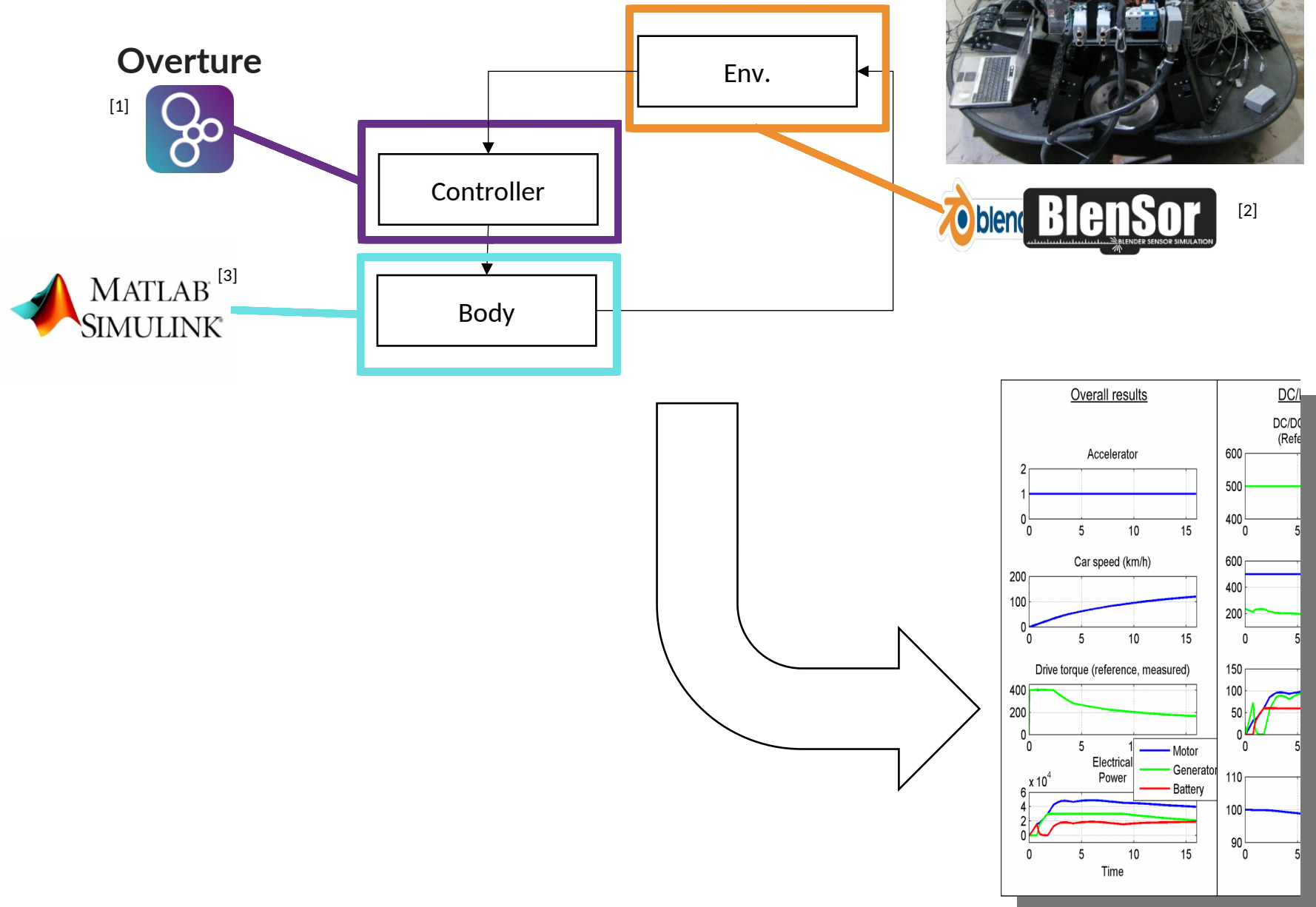
- Problem-Specific (technological)
- Domain-Specific (e.g., translational mechanical)
- (general) Laws of Physics
- Power Flow/Bond Graphs (physical: energy/power)
- Computationally a-causal
(Mathematical and Object-Oriented) ← **Modelica**
- Causal Block Diagrams (data flow)
- Numerical (Discrete) Approximations
- Computer Algorithmic + Numerical
(Floating Point vs. Fixed Point)
- As-Fast-As-Possible vs. Real-time (XiL)
- Hybrid (discrete-continuous) modelling/simulation
- Hiding IP: Composition of Functional Mockup Units (FMI)
- Dynamic Structure

problem: **full-system analysis**



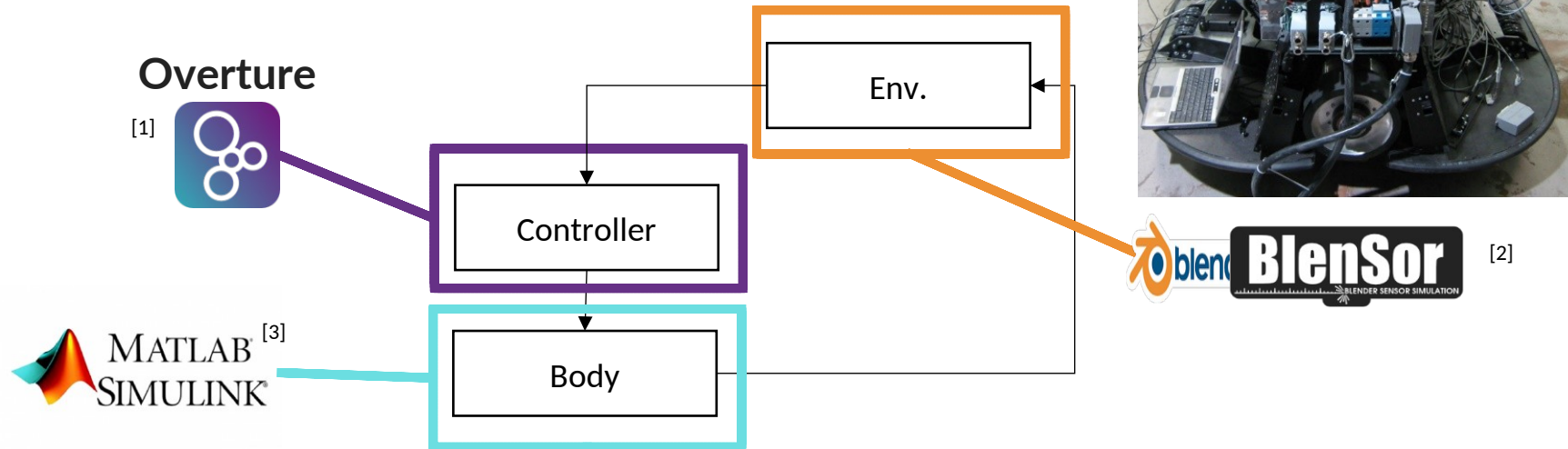
problem: **full-system analysis**

(also when IP protected)

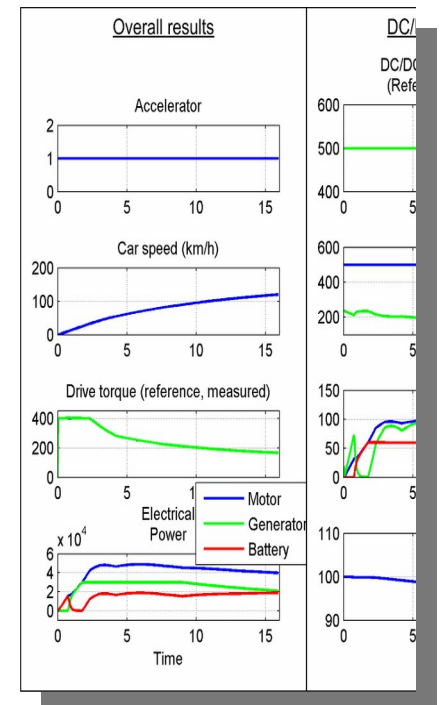


problem: **full-system analysis**

(also when IP protected)

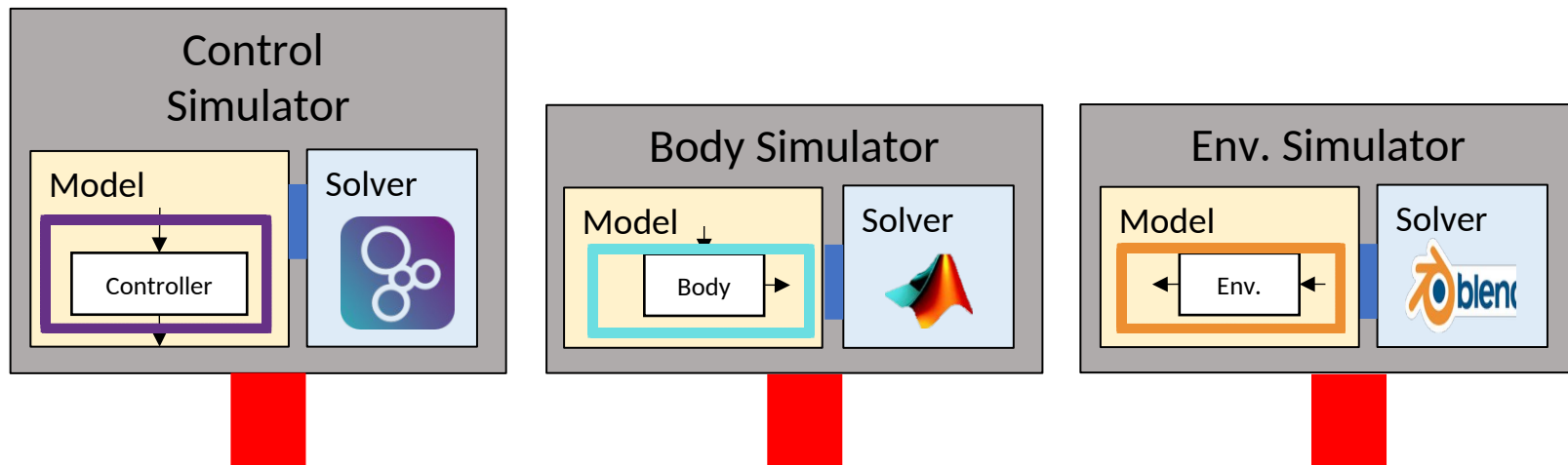
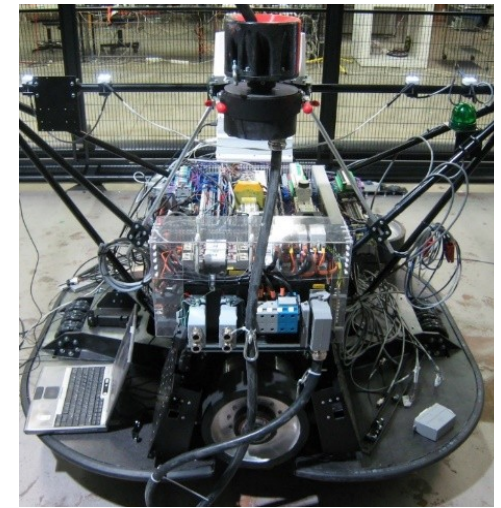
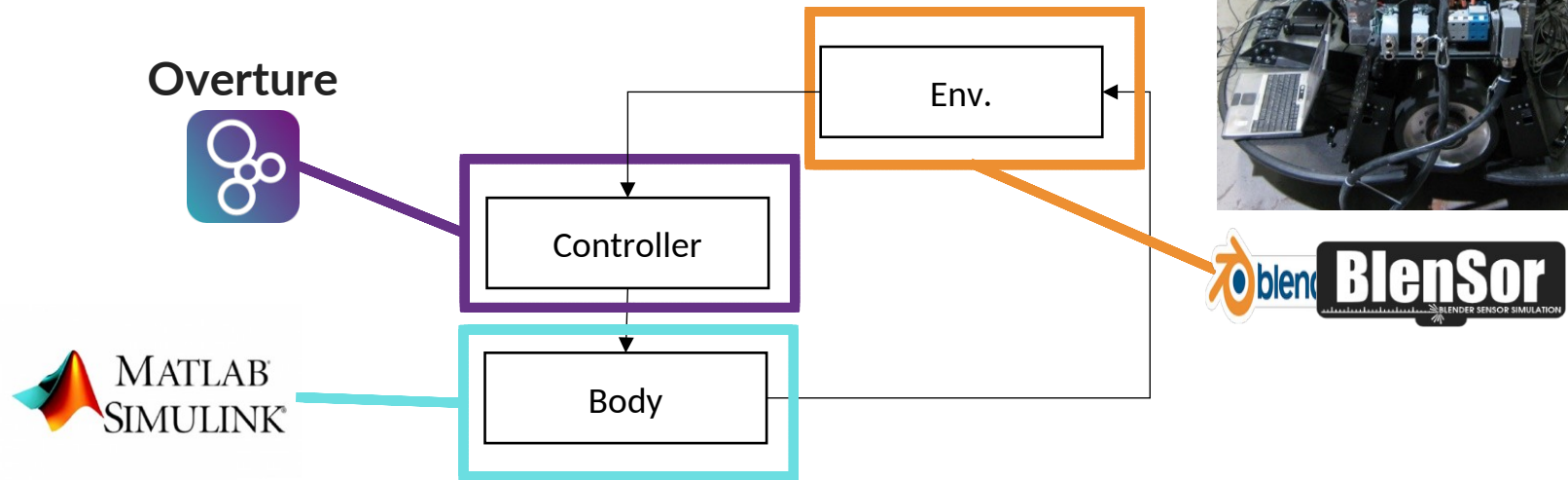


solution: combine
sub-system **simulators**
aka **co-simulation**



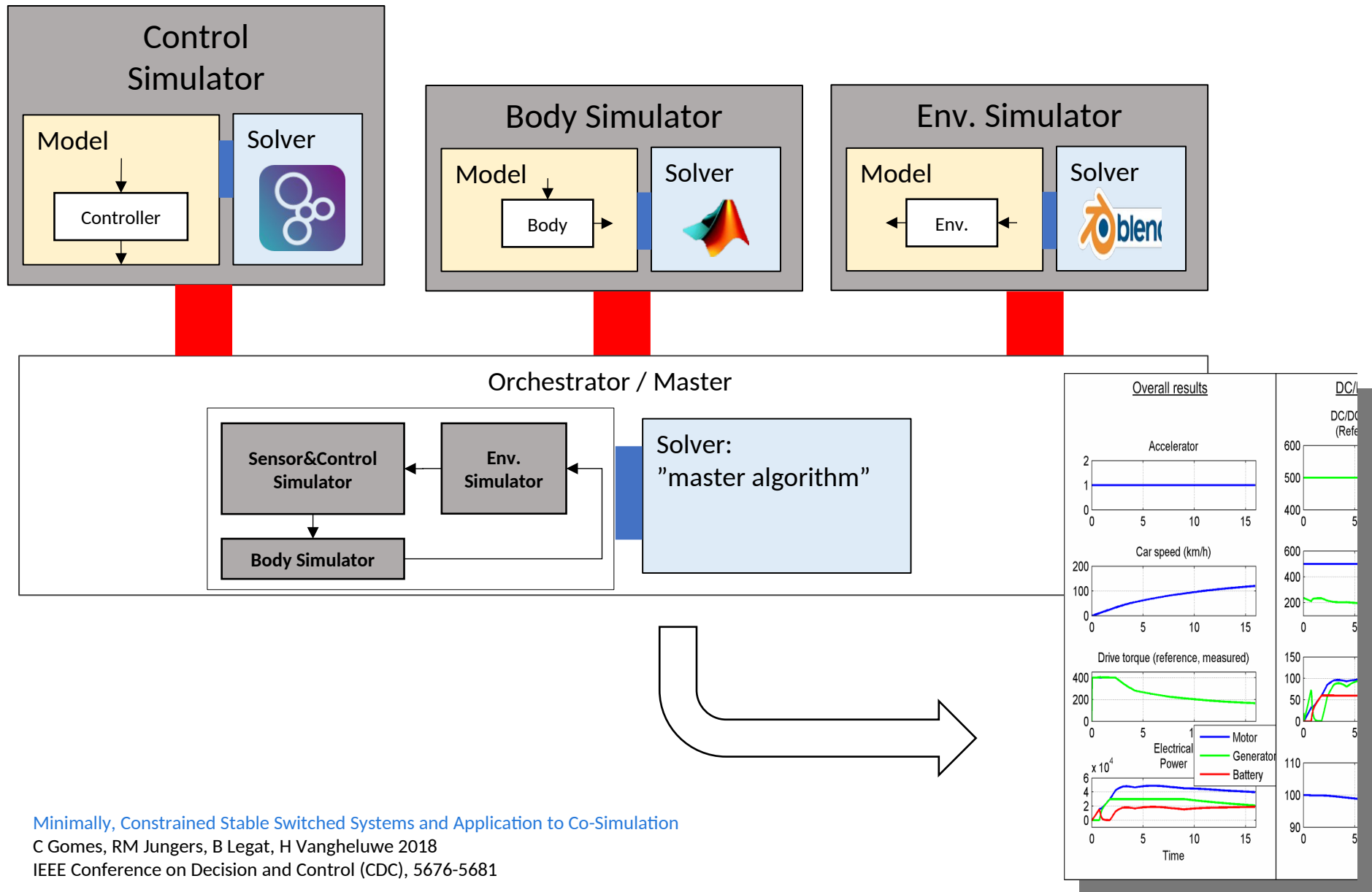
co-simulation: how?

(when IP protected)



co-simulation: how?

(when IP protected)

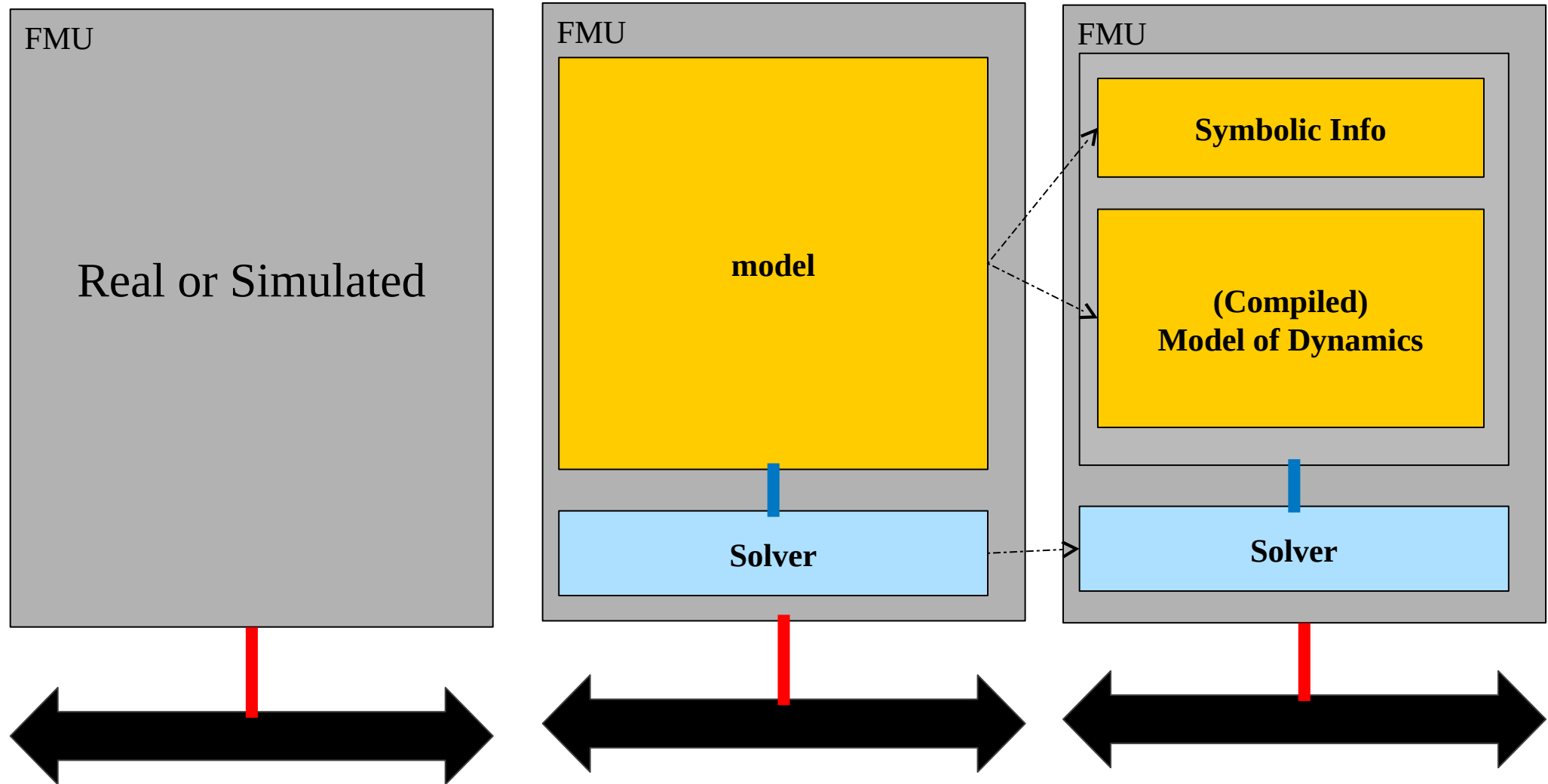


Minimally, Constrained Stable Switched Systems and Application to Co-Simulation

C Gomes, RM Jungers, B Legat, H Vangheluwe 2018

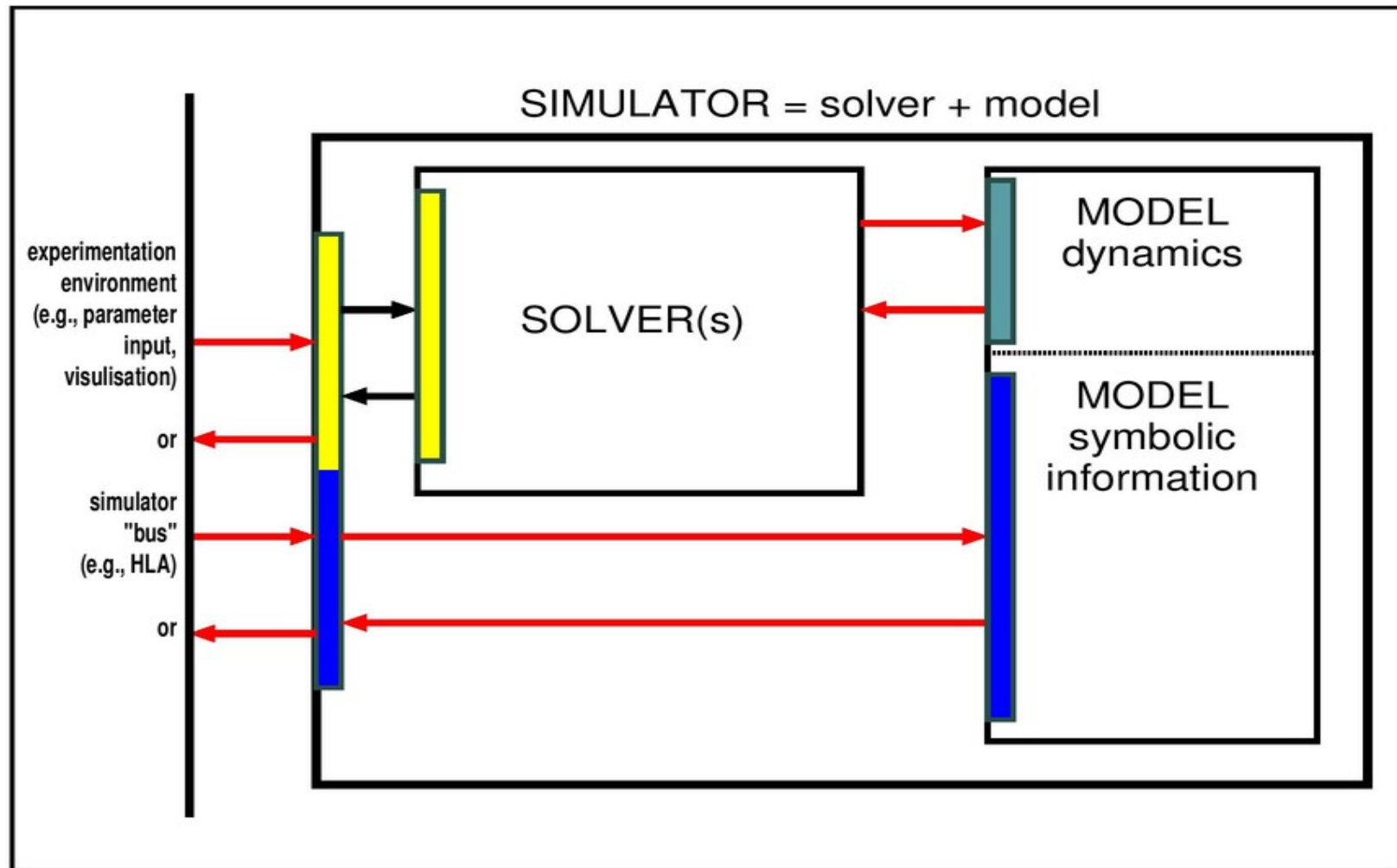
IEEE Conference on Decision and Control (CDC), 5676-5681

Deconstructing a Simulator: Functional Mockup Unit (FMU)



Model-Solver Interface

Simulator-Environment Interface



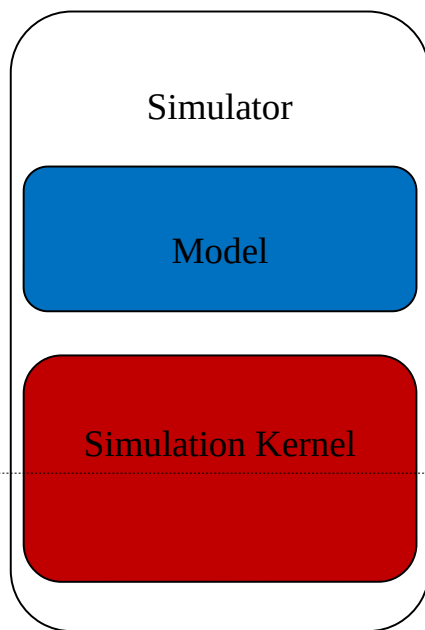
DSblock

Martin Otter and Hilding Elmquist.
The DSblock interface for exchanging model components. Eurosime '95 Simulation Congress. pp. 505- 510. 1995.

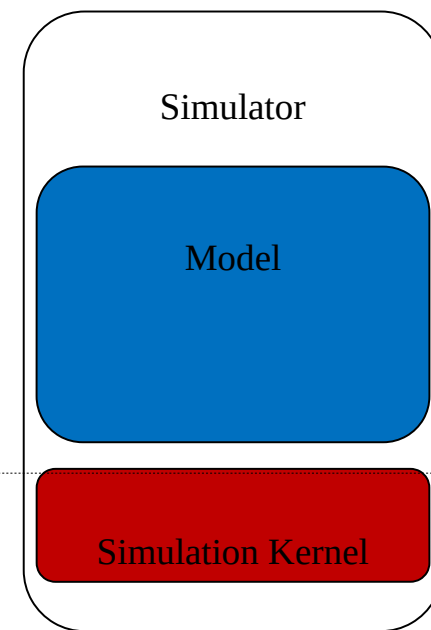
MSL-EXEC

Henk Vanhooren, Jurgen Meirlaen, Yuri Amerlinck, Filip Claeys, Hans Vangheluwe, and Peter A. Vanrolleghem.
WEST: Modelling biological wastewater treatment. Journal of Hydroinformatics , 5(1):27--50, 2003.

meaningful operational semantics (Models of Computation)



Traditional
Simulator



Explicit
Computational Semantics
“inline integration” (Cellier)



The leading standard to exchange dynamic simulation models

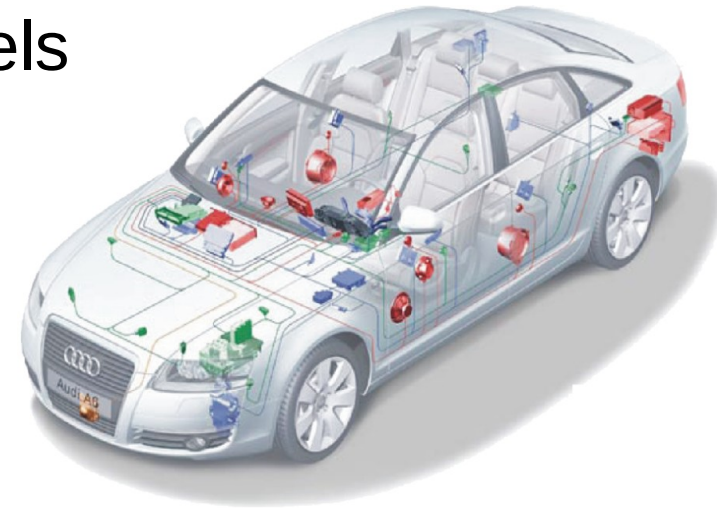
The Functional Mock-up Interface is a free standard that defines a container and an interface to exchange dynamic simulation models using a combination of XML files, binaries and C code, distributed as a ZIP file. It is supported by **180+ tools** and maintained as a Modelica Association Project.

[Why FMI](#)[Complete Package 3.0.1](#)[Specification 3.0.1](#)[Implementers' Guide](#)

<https://fmi-standard.org/>

Functional Mock-up Interface (FMI)

- **XML** + Binary Representation for Models
 - Standard
 - Modelling Tool Independent
 - +/- Black box ...



Symbolic information: FMU - XML

```
<?xml version="1.0" encoding="iso-8859-1"?>
<fmiModelDescription fmiVersion="1.0"
  modelName="BackEulerExmpl"
  modelIdentifier="BackEulerExmpl"
  guid="{7c4e810f-3da3-4a00-8276-176fa3c9f000}"
  numberOfContinuousStates="1"
  numberOfEventIndicators="0">
  <ModelVariables>
    <ScalarVariable
      name="Delay" valueReference="0">
      <Real start="0.0" fixed="true" />
    </ScalarVariable>
    <ScalarVariable
      name="stepSize" valueReference="1">
      <Real start="0.1" fixed="true" />
    </ScalarVariable>
    <ScalarVariable
      name="Product" valueReference="2">
      <Real start="0.0" fixed="true" />
    </ScalarVariable>
    <ScalarVariable
      name="Negator" valueReference="3">
      <Real start="0.0" fixed="true" />
    </ScalarVariable>
    <ScalarVariable
      name="Adder" valueReference="4">
      <Real start="0.0" fixed="true" />
    </ScalarVariable>
  </ModelVariables>
</fmiModelDescription>
```

Behaviour information: FMU - C

```
fmi2Status fmi2DoStep(fmi2Component fc , fmi2Real currentCommPoint, fmi2Real commStepSize, fmi2Boolean
noPrevFMUState)
{
    FMUInstance* fi = (FMUInstance *)fc;
    fmi2Status simStatus = fmi2OK;
    printf("%s in fmi2DoStep()\n", fi->instanceName);
    fi->currentTime = currentCommPoint + commStepSize;
    printf("Motor_in: %f\n", fi->r[_motor_in]);
    printf("slave CBD_PART2 now at time: %f\n", fi->currentTime);

    fi->r[_position] = fi->r[_position] + fi->r[_velocity] * commStepSize;
    fi->r[_velocity] = fi->r[_velocity] + fi->r[_acceleration_after_friction] * commStepSize;
    fi->r[_friction] = fi->r[_velocity] * 5.81;
    fi->r[_motor_acceleration] = fi->r[_motor_in] * 40;
    fi->r[_acceleration_after_friction] = fi->r[_motor_acceleration] - fi->r[_friction];

    return simStatus;
}

fmi2Status fmi2GetReal(fmi2Component fc, const fmi2ValueReference vr[], size_t nvr, fmi2Real value[])
{
    FMUInstance* comp = (FMUInstance *)fc;
    int i;
    for (i = 0; i < nvr; i++)
    {
        value[i] = comp->r[(vr[i])];
    }
    return fmi2OK;
}
```

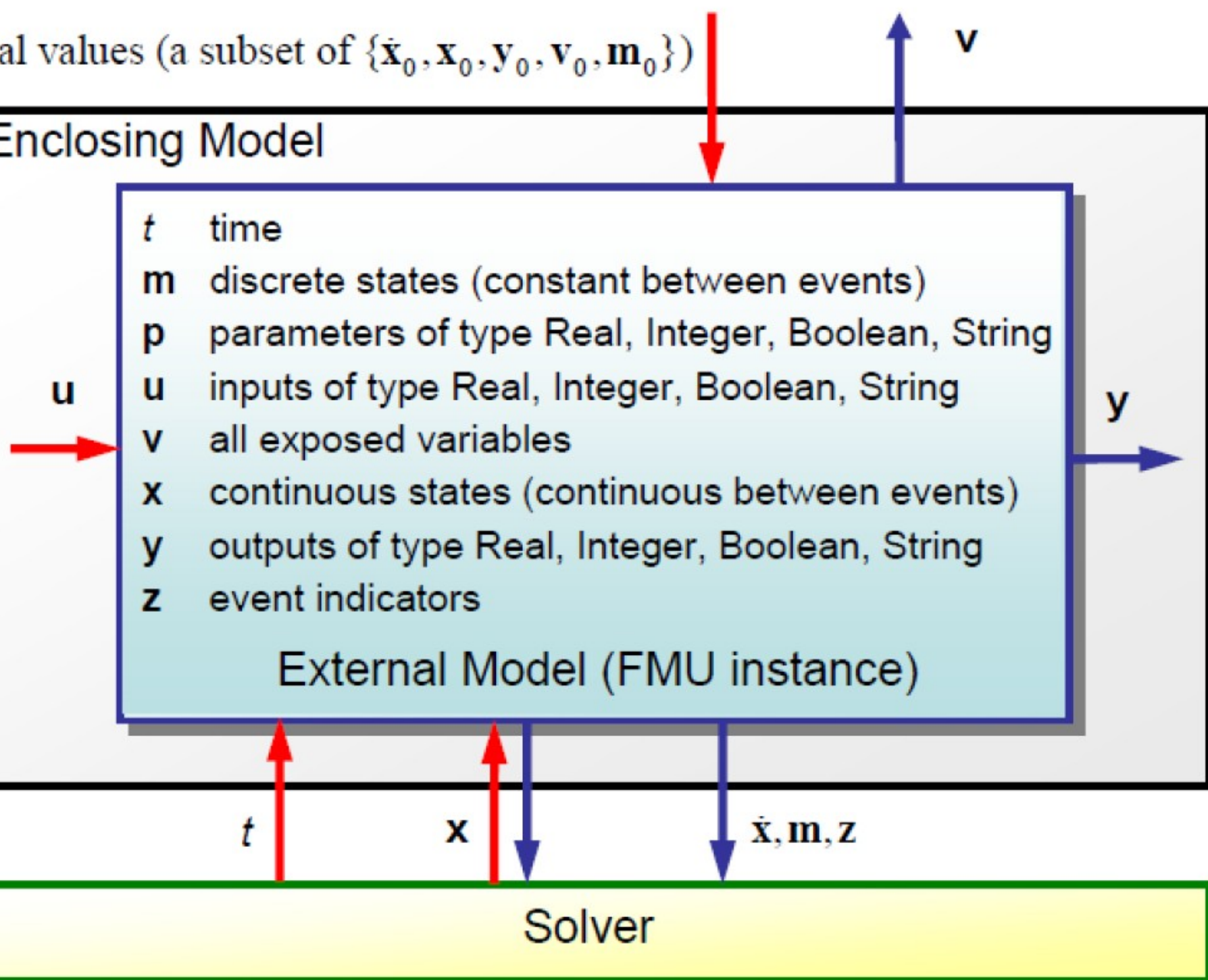
$t_0, \mathbf{p}$, initial values (a subset of $\{\dot{\mathbf{x}}_0, \mathbf{x}_0, \mathbf{y}_0, \mathbf{v}_0, \mathbf{m}_0\}$)

Enclosing Model

t time
 $\mathbf{m}$ discrete states (constant between events)
 $\mathbf{p}$ parameters of type Real, Integer, Boolean, String
 $\mathbf{u}$ inputs of type Real, Integer, Boolean, String
 $\mathbf{v}$ all exposed variables
 $\mathbf{x}$ continuous states (continuous between events)
 $\mathbf{y}$ outputs of type Real, Integer, Boolean, String
 $\mathbf{z}$ event indicators

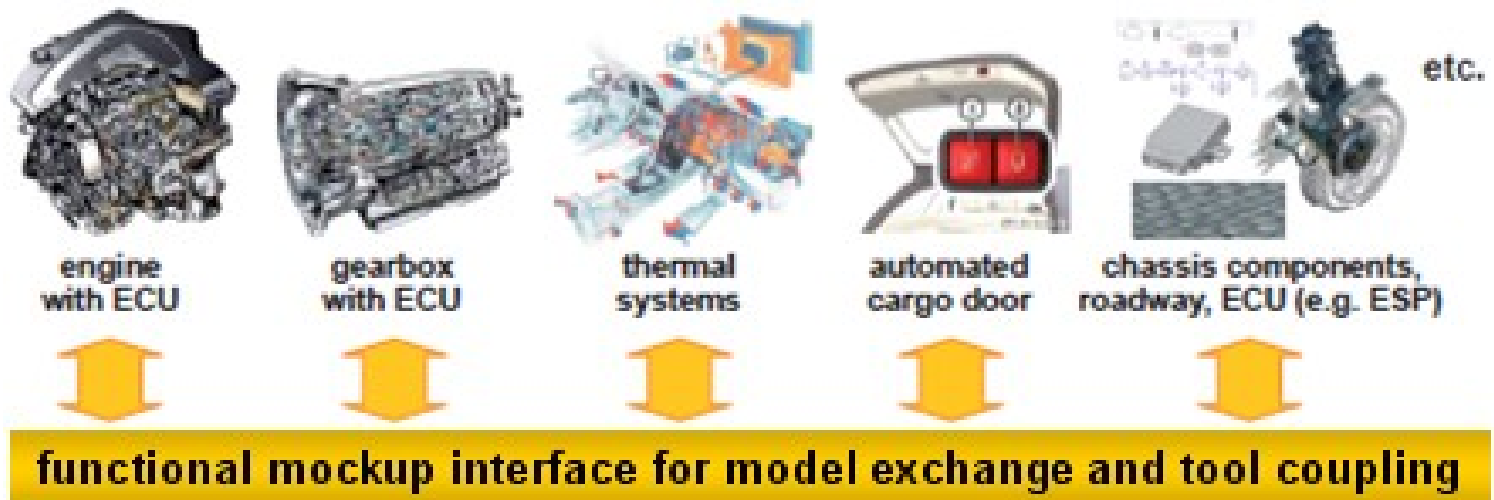
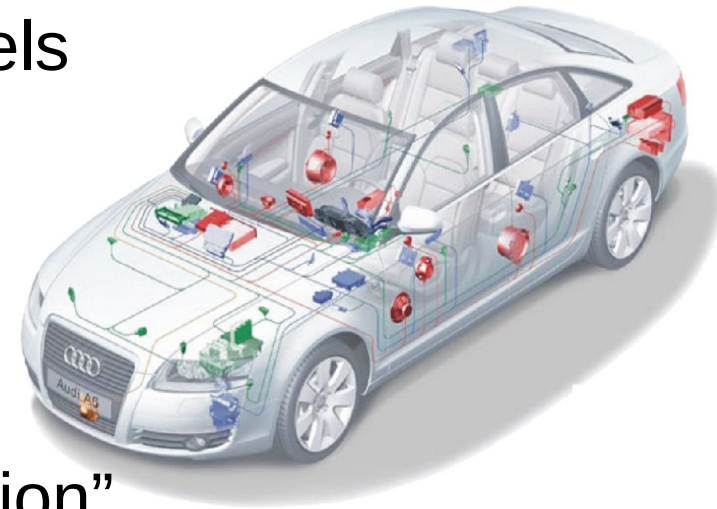
External Model (FMU instance)

Solver



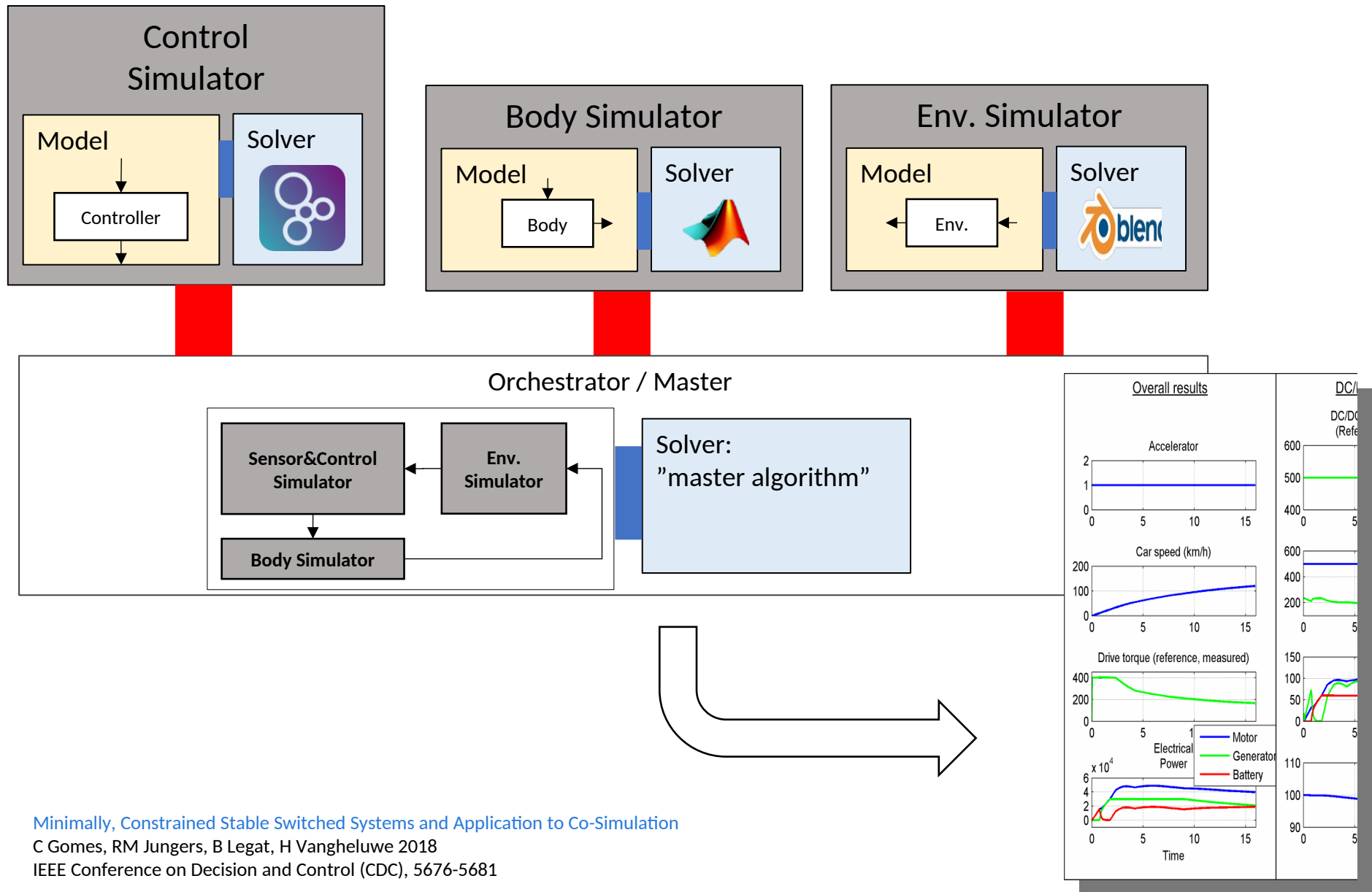
Functional Mock-up Interface (FMI)

- **XML** + Binary Representation for Models
 - Standard
 - Modelling Tool Independent
 - +/- Black box ...
- Composed FMUs still need “orchestration”



co-simulation: how?

(when IP protected)

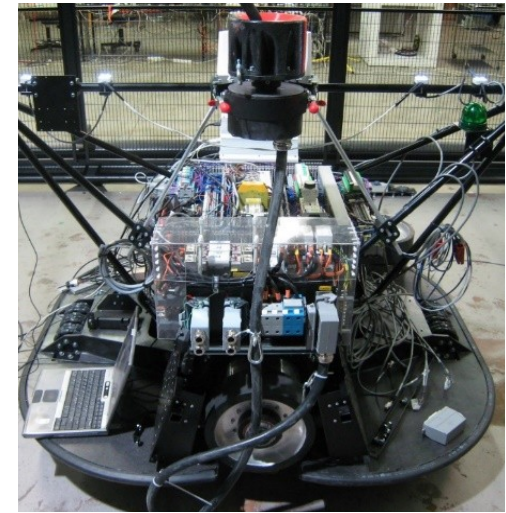
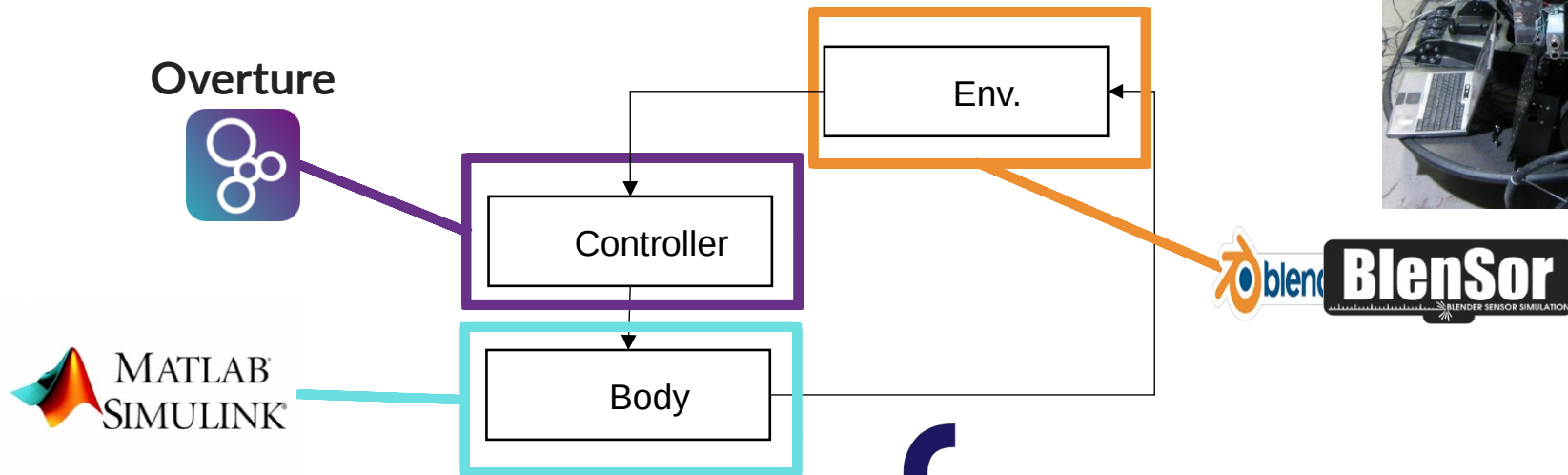


Minimally, Constrained Stable Switched Systems and Application to Co-Simulation

C Gomes, RM Jungers, B Legat, H Vangheluwe 2018

IEEE Conference on Decision and Control (CDC), 5676-5681

Co-simulation: how?



fmi FUNCTIONAL
MOCK-UP
INTERFACE

[1]

Control Simulator
FMU.zip
|-- .dll or .so
|-- .xml

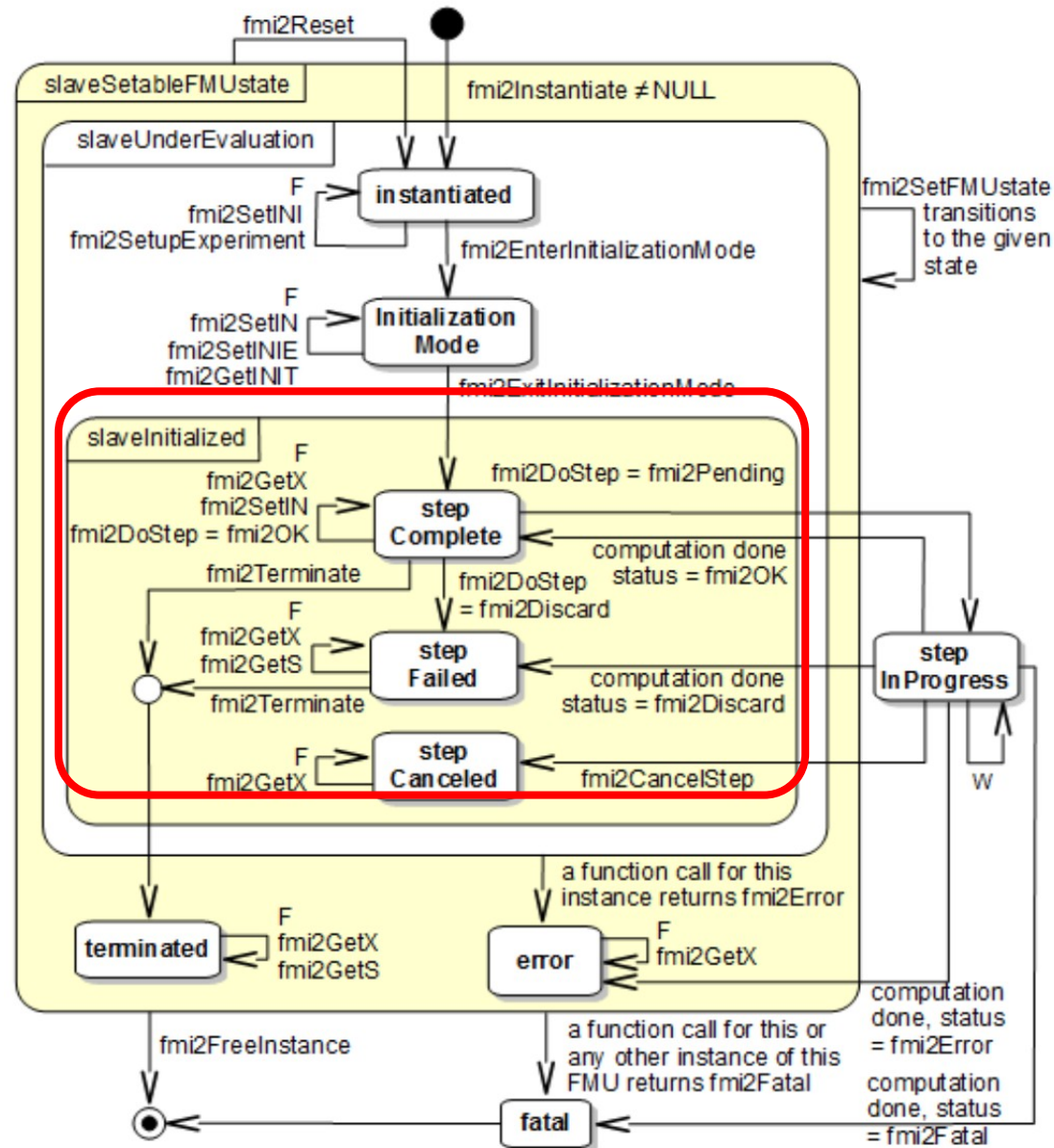
Body Simulator

Env. Simulator

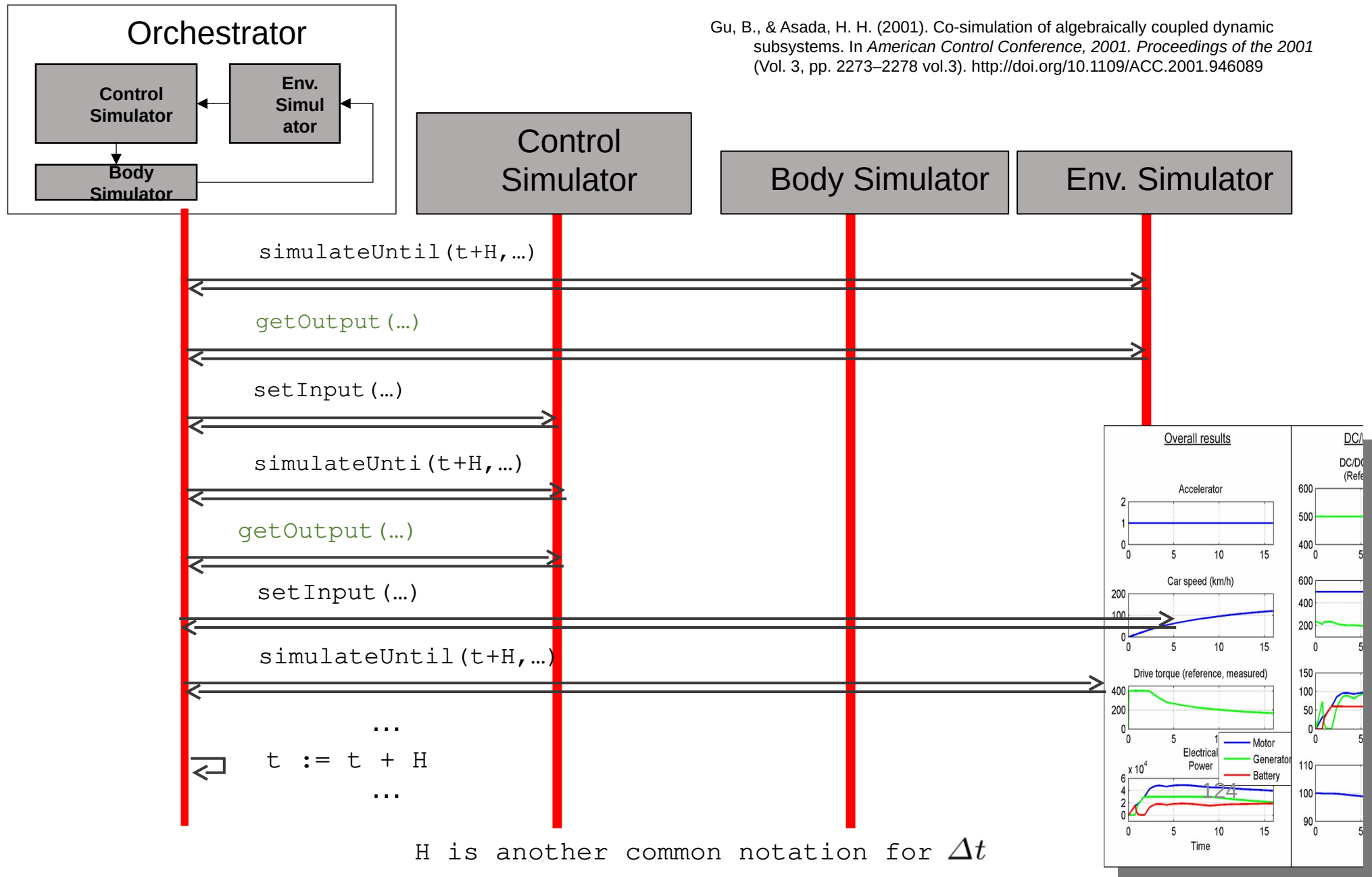
fmi2DoStep(...),
fmi2SetReal(...), ...

Co-simulation master

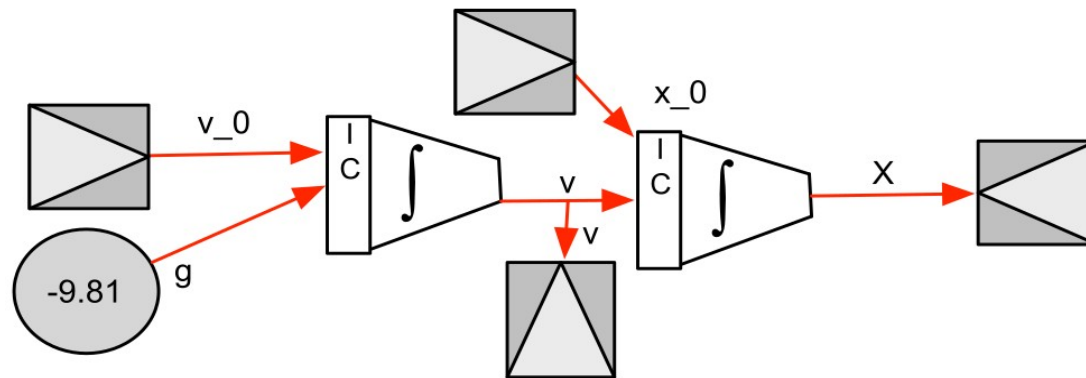
State machine calling sequence Master-Slave



Co-simulation: how?



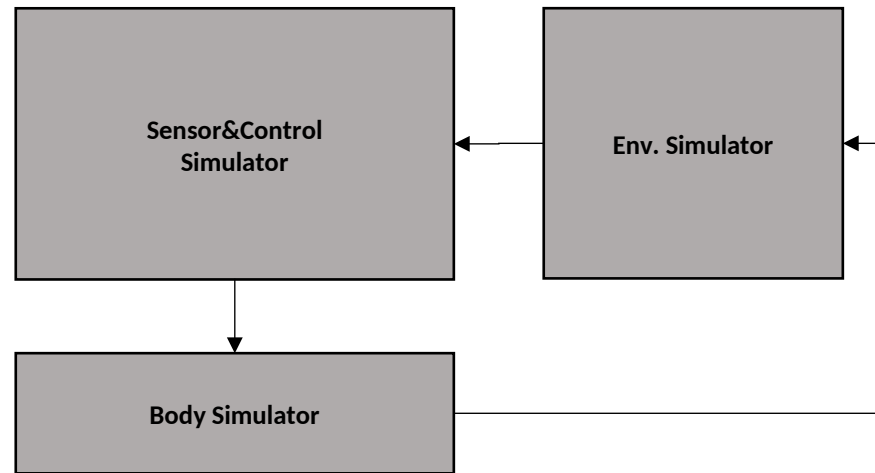
Looking inside a simulator/solver for Causal Block Diagrams (CBDs)



```
logicalTime  $\leftarrow$  0
while not end_condition do
    schedule  $\leftarrow$  LOOPDETECT(DEPGRAPH(cbd))
    for gblock in schedule do
        COMPUTE(gblock)
    end for
    logicalTime  $\leftarrow$  logicalTime +  $\Delta t$ 
end while
```

CBD simulation algorithm can also be used for scheduling at level of Master
need “zero-delay feedthrough” information (algebraic loops)

Caveat: naive



```
logicalTime  $\leftarrow$  0  
while not end_condition do  
    schedule  $\leftarrow$  LOOPDETECT(DEPGRAPH(cbd))  
    for gblock in schedule do  
        COMPUTE(gblock)  
    end for  
    logicalTime  $\leftarrow$  logicalTime +  $\Delta t$   
end while
```

