Modeling complex engineered systems in industry using MATLAB and Simulink

Pieter J. Mosterman

Chief Research Scientist, Director
Advanced Research & Technology Office
MathWorks

Adjunct Professor
School of Computer Science
McGill

Machines are connecting and collaborating

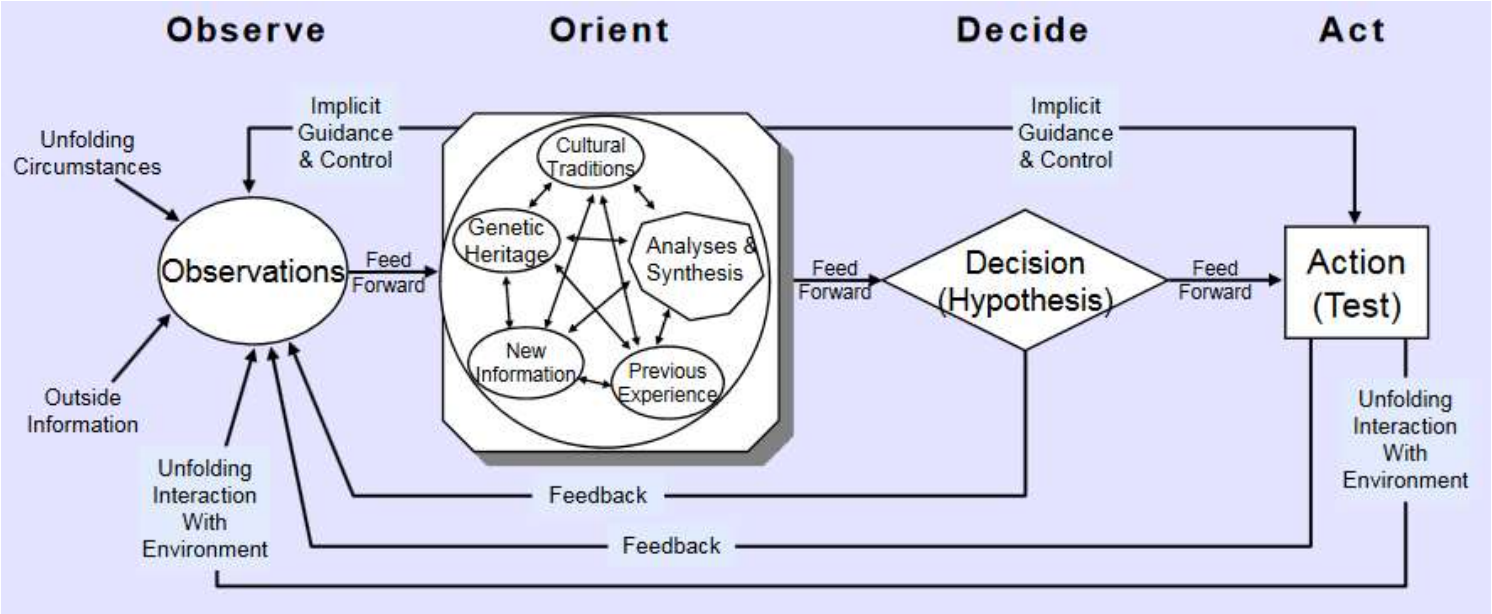
Where can we have impact, which solutions are needed, what challenges these solutions, and how can we overcome the challenges?

A smart emergency response system

- System characterization
- A requirements perspective
- The Towers of Hanoi revisited
- Multiformalism

The Observe-Orient-Decide-Act (OODA) loop

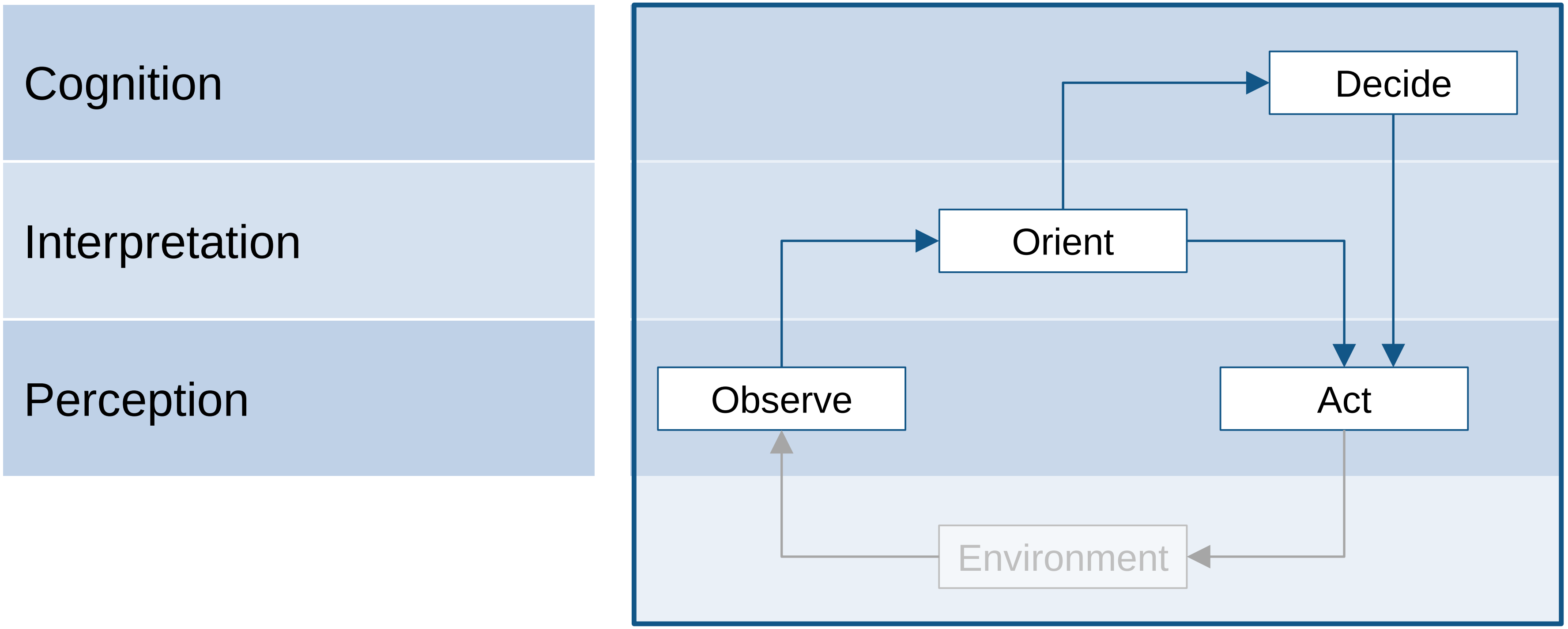
Colonel John Richard Boyd



MESSAGE

Orientation, <u>seen as a result</u>, represents images, views, or impressions of the world shaped by <u>genetic heritage</u>, <u>cultural tradition</u>, <u>previous experiences</u>, and <u>unfolding circumstances</u>.

OODA and the stages of cognition



A feature classification

	Perceive	Interpret	Reason
Ensemble	Distributed	Connected	Collaborative
Individual	Automatic	Adaptive	Autonomous

System characterization

A requirements perspective

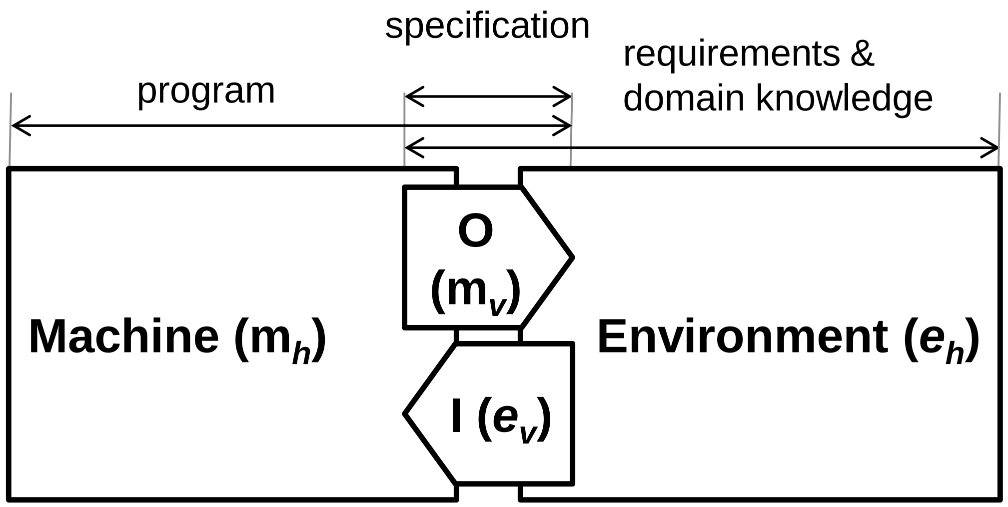
The Towers of Hanoi revisited

Multiformalism

Requirements engineering

Michael Anthony Jackson

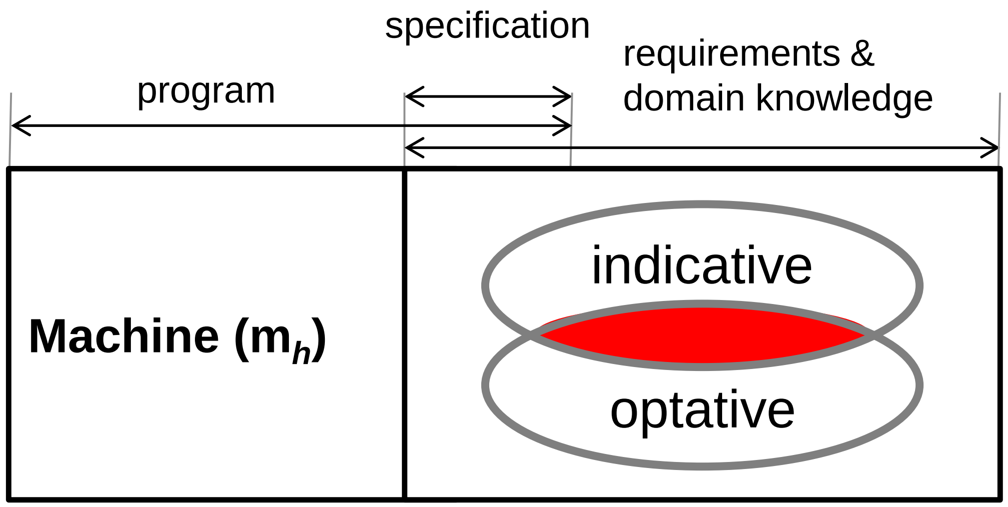
- A **requirement** is a desired relationship among **phenomena** (e.g., actions/events, states) of the environment
- Phenomena are categorized as
  - ***e_h***: controlled (or initiated) by the **e**nvironment and **h**idden from (i.e., invisible to, not shared with) the machine
  - ***e_v***: controlled by the **e**nvironment but **v**isible to (i.e., shared with) the machine
  - ***m_v***: controlled by the **m**achine but **v**isible to (shared with) the environment
  - ***m_h***: controlled by the **m**achine and **h**idden from (i.e., not shared with) the environment



Requirements engineering

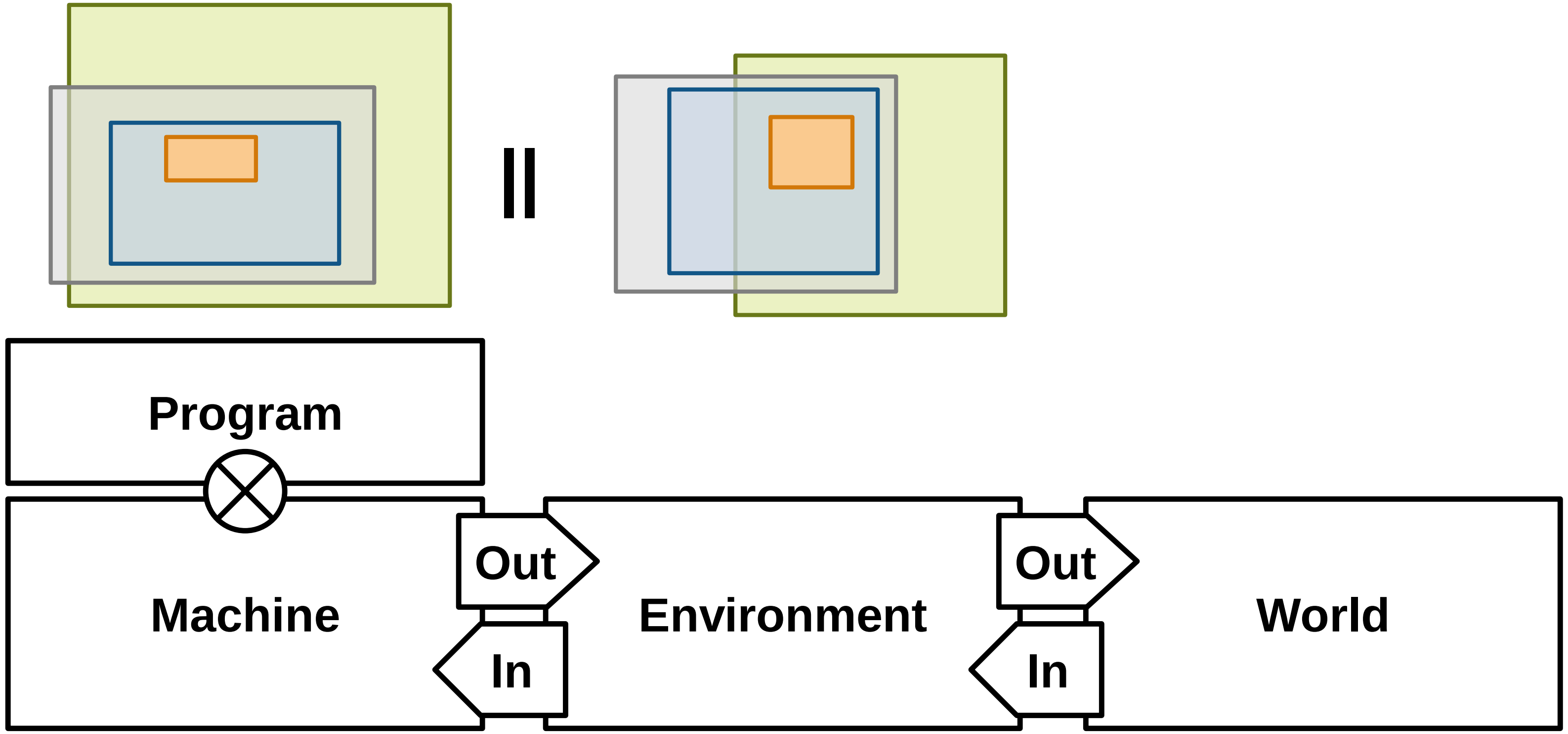
Michael Anthony Jackson

- A **requirement** is a desired relationship among **phenomena** (e.g., actions/events, states) of the environment
- Phenomena are categorized as
  - $\boldsymbol{e_h}$: controlled (or initiated) by the **e**nvironment and **h**idden from (i.e., invisible to, not shared with) the machine
  - $\boldsymbol{e_v}$: controlled by the **e**nvironment but **v**isible to (i.e., shared with) the machine
  - $\boldsymbol{m_v}$: controlled by the **m**achine but **v**isible to (shared with) the environment
  - $\boldsymbol{m_h}$: controlled by the **m**achine and **h**idden from (i.e., not shared with) the environment



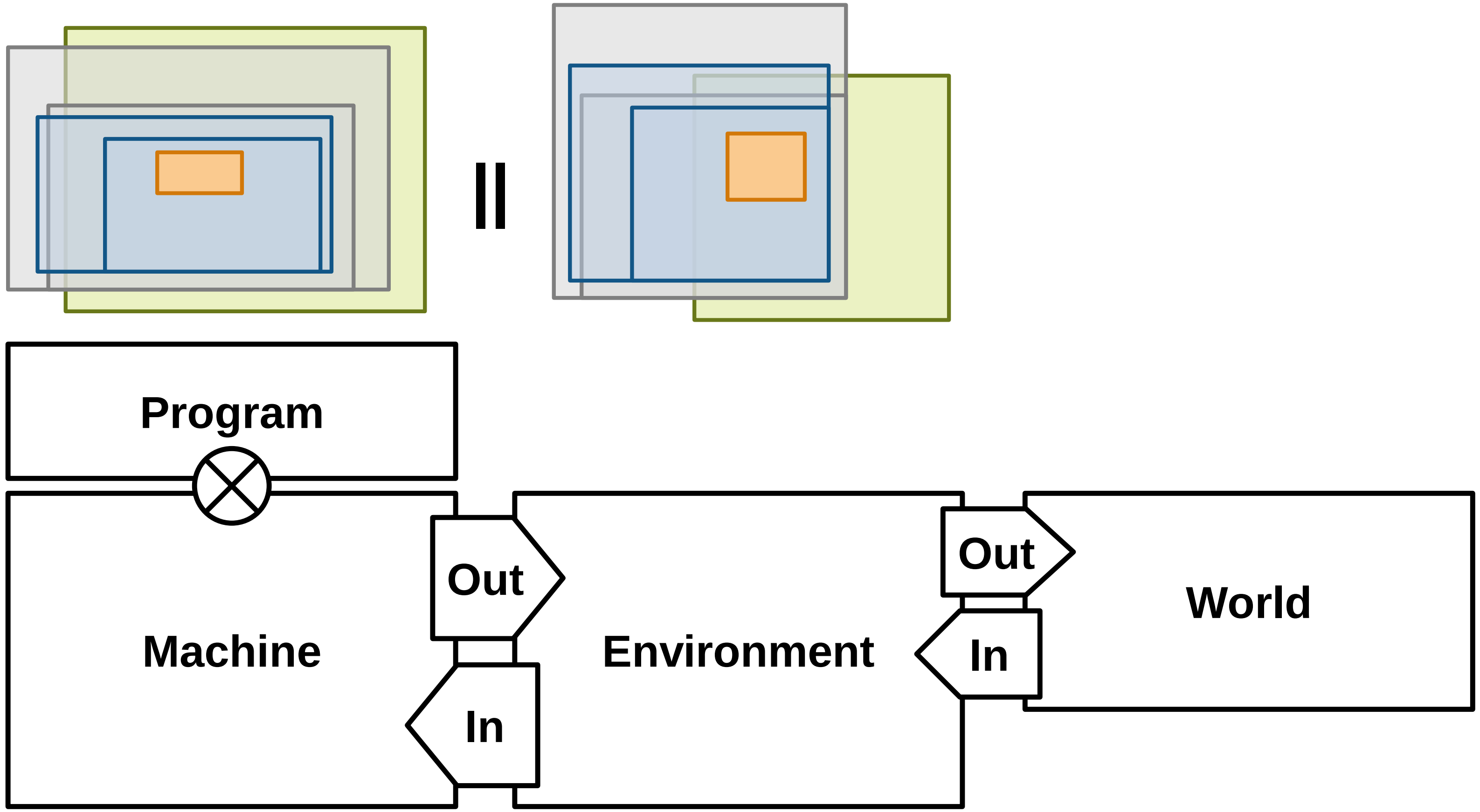
A behavioral view

Closed loop designed behavior
Property satisfying behavior
Closed loop possible behavior
Open loop possible behavior



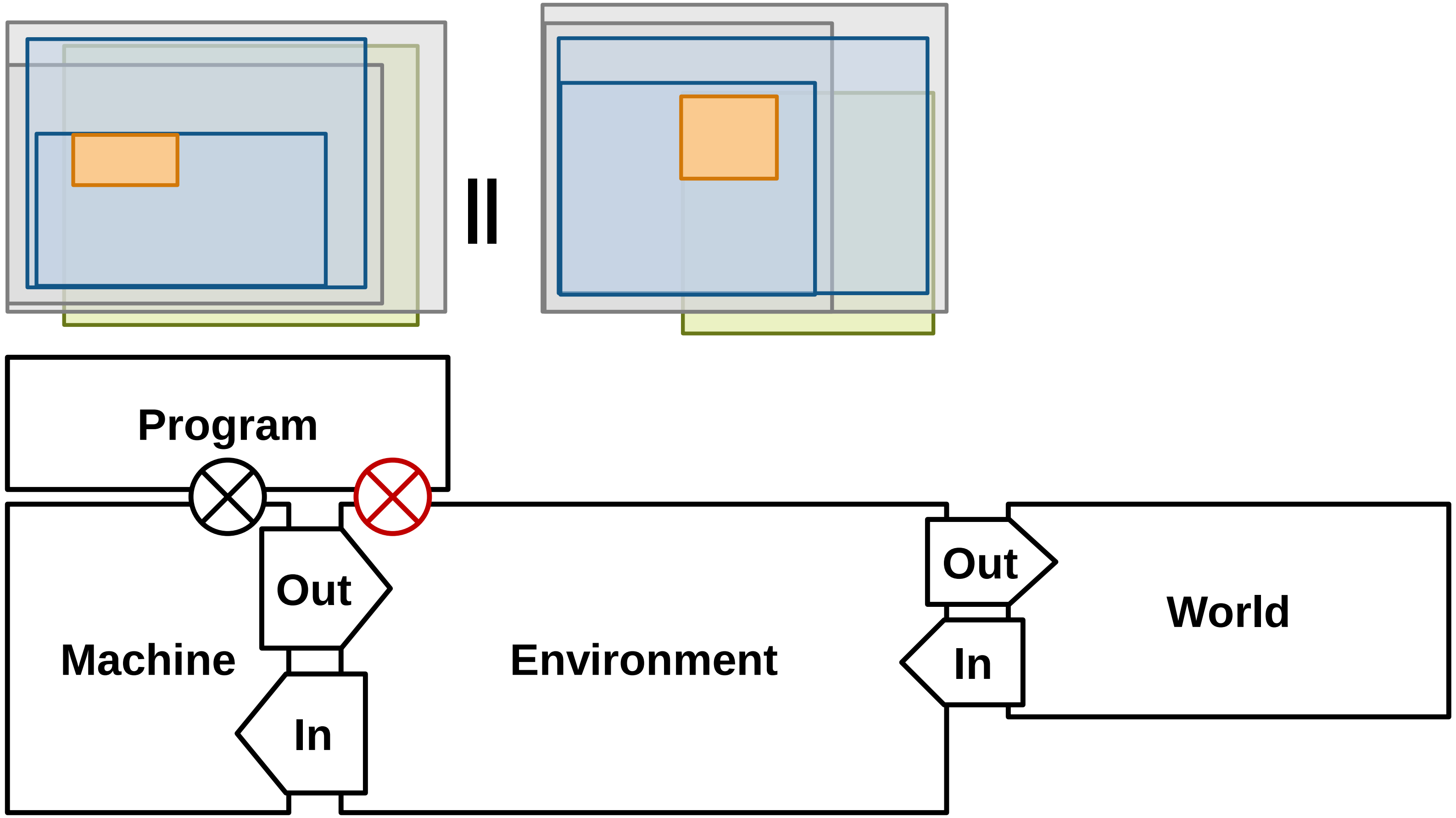
A behavioral view

Closed loop designed behavior
Property satisfying behavior
Closed loop possible behavior
Open loop possible behavior



A behavioral view

Closed loop designed behavior
Property satisfying behavior
Closed loop possible behavior
Open loop possible behavior



Implementing a specification

Functional

- Interface phenomena e_v and m_v
  - May not uniquely determine I/O mappings
  - May construct hidden state from environment model environment designations (e.g., observer, filter)
  - May require processing with state (e.g., signal to symbol)

Behavioral

- Configure a machine
  - Internal state of a machine

State

Environment

- In our mind
- FEM up to 400k degrees of (Real) freedom

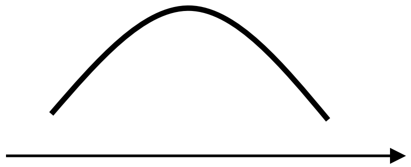
Machine (System)

- In our realization
- Up to ~4G*8 degrees of (binary) freedom

Characterization

Physics

- Dynamic models
- Real values
- Continuous (ODE, DAE)

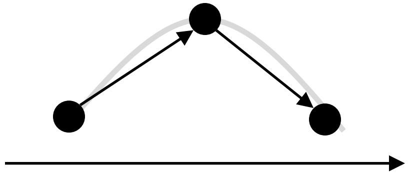


$f(dx/dt, x, u, t) = 0$

- State coupling!
  – For control design, 400k states reduce to ~10 states

Computation

- Steady state models (clocked)
- Binary values
- Discrete (LTS, FSM)



$\langle \Theta, \Lambda, T \rangle$

- Entirely independent!
  – Engineered as such, ~$32 \cdot 10^9$ states

State is complex!

Tackle large state spaces by analyzing sets of states

- System variation on individual traces
- Different conditions (e.g., failure modes)

- Physically—sensitivities

("within engineering tolerance")

$$s_p^x = \left.\frac{\partial \xi}{\partial p}\right|_{p,x_0} (t)$$

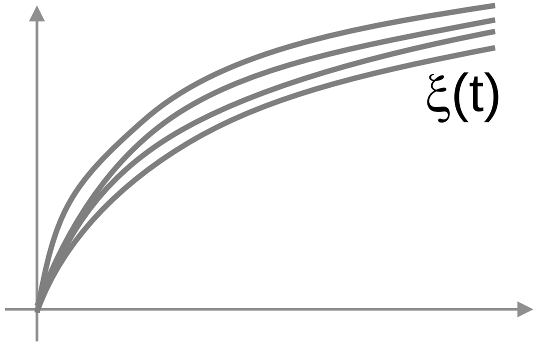
$$\dot{s}_p^x = \frac{\partial f}{\partial x} s_p^x + \frac{\partial f}{\partial p}, s_p^x(0) = \frac{\partial x_0}{\partial p}$$

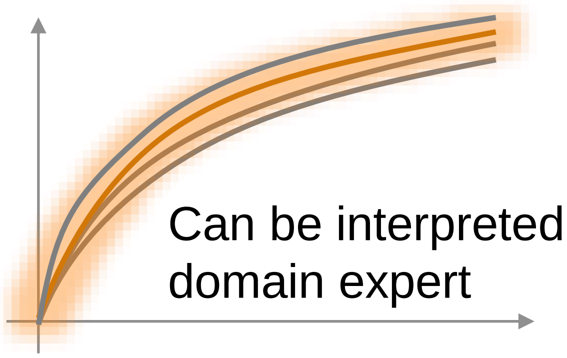
- Computationally—abstraction

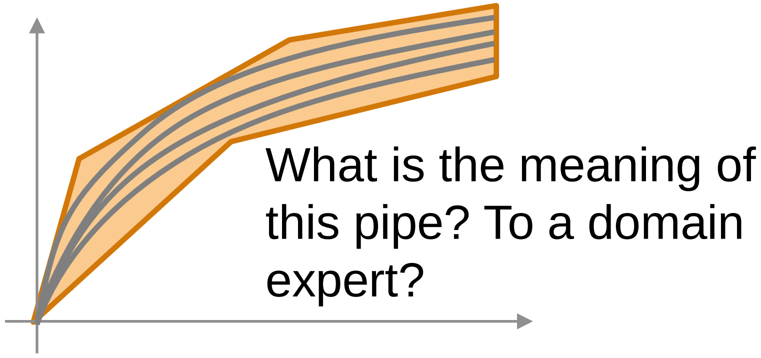
$$\underline{y} = \bigcup_{t\in[0,T]} \min\big(y(t)\big) = \bigcup_{t\in[0,T]} \min\big(g(x(t),u\big)$$

$$\overline{y} = \bigcup_{t\in[0,T]} max\big(y(t)\big) = \bigcup_{t\in[0,T]} max\big(g(x(t),u\big)$$

$$s.t. \begin{cases} x(t) = x_0 + \int_0^t f(x,u,\lambda,\tau)d\tau \\ \lambda \in [\underline{\lambda},\overline{\lambda}], x_0 \in [\underline{x_0},\overline{x_0}] \end{cases}$$







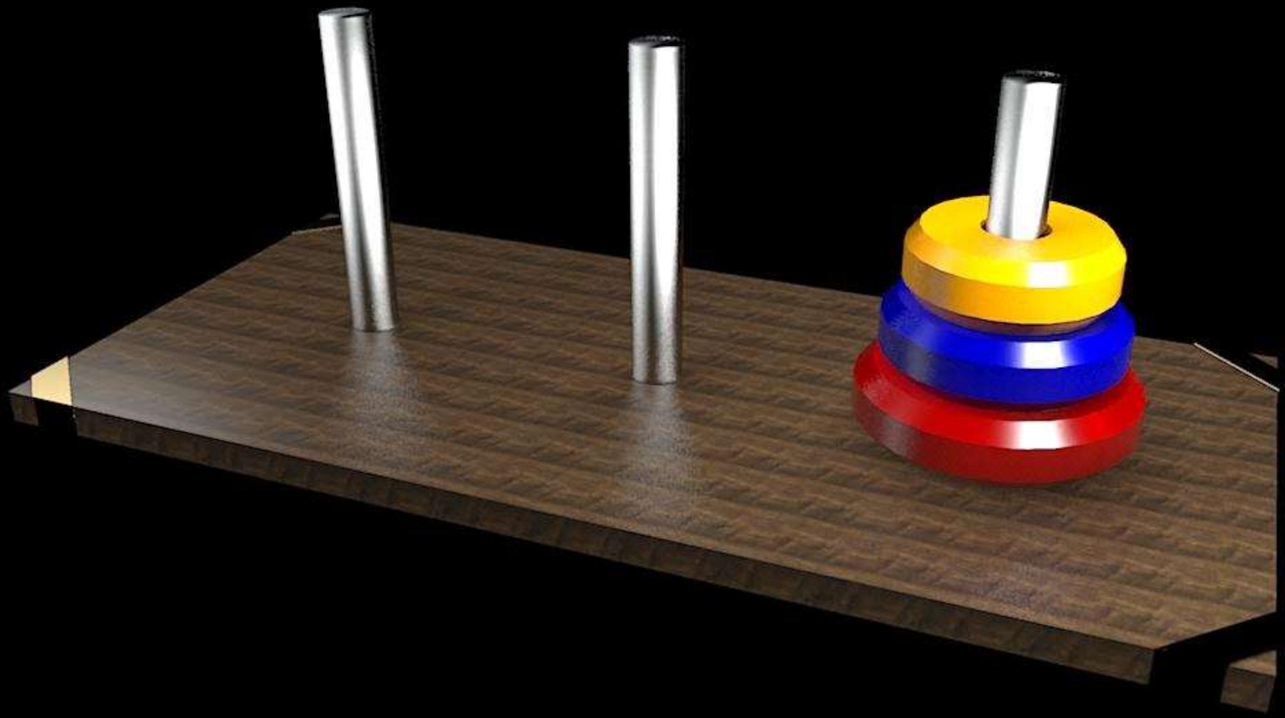
Selectively analyze the state space

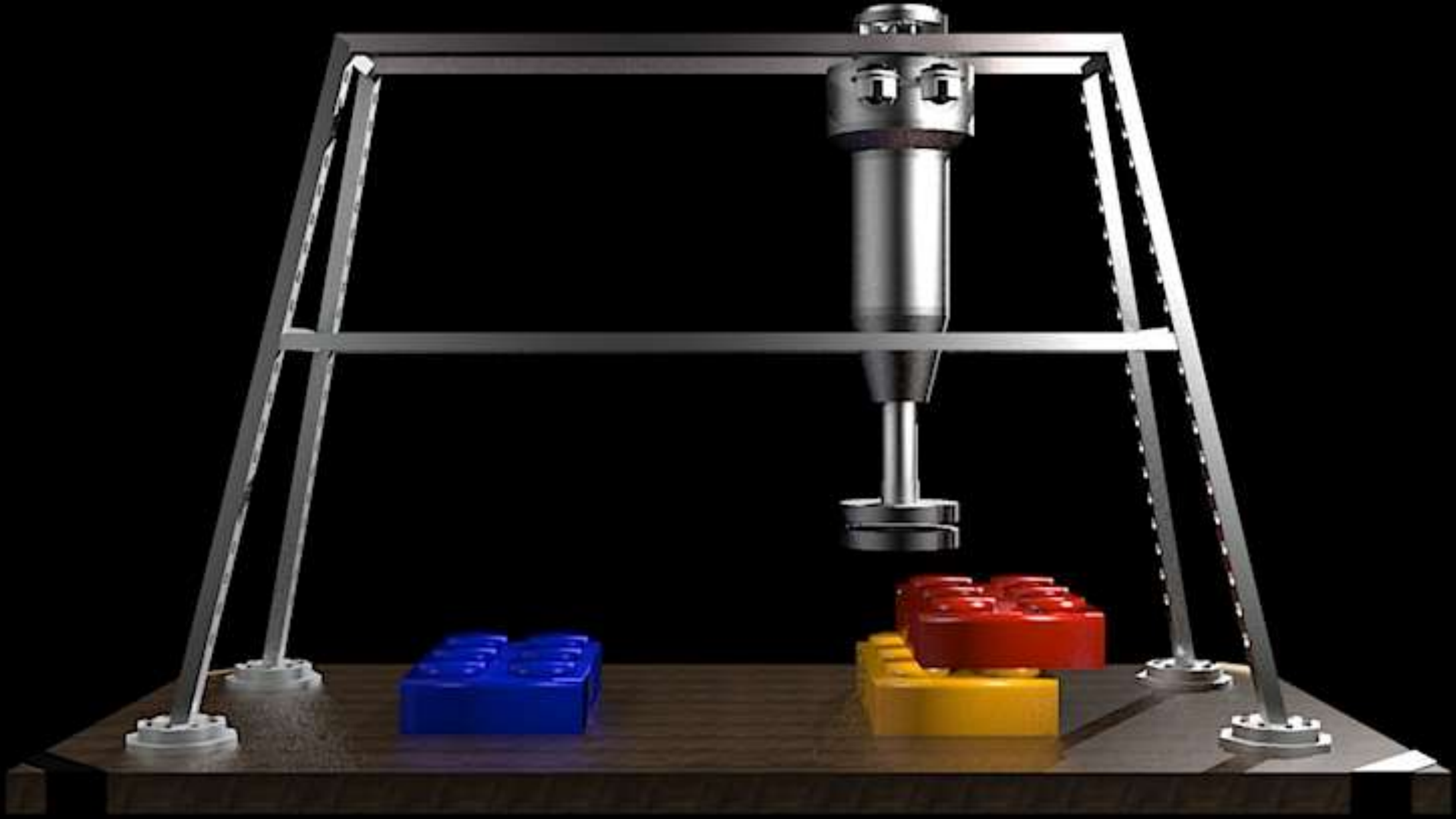
- Restrictions over state space exist
  - Analyze sets of states
  - Exclude the set of infeasible states
- Open loop analysis is problematic
  - Deep (prohibitively) input traces build up an offending state
  - Incomprehensible input sequences
- Knowing feasible states is key
  - Restrict input to achievable traces
  - Include feasible environment reaction
- Close the loop
  - Analyze combined system and environment (set amenable) models

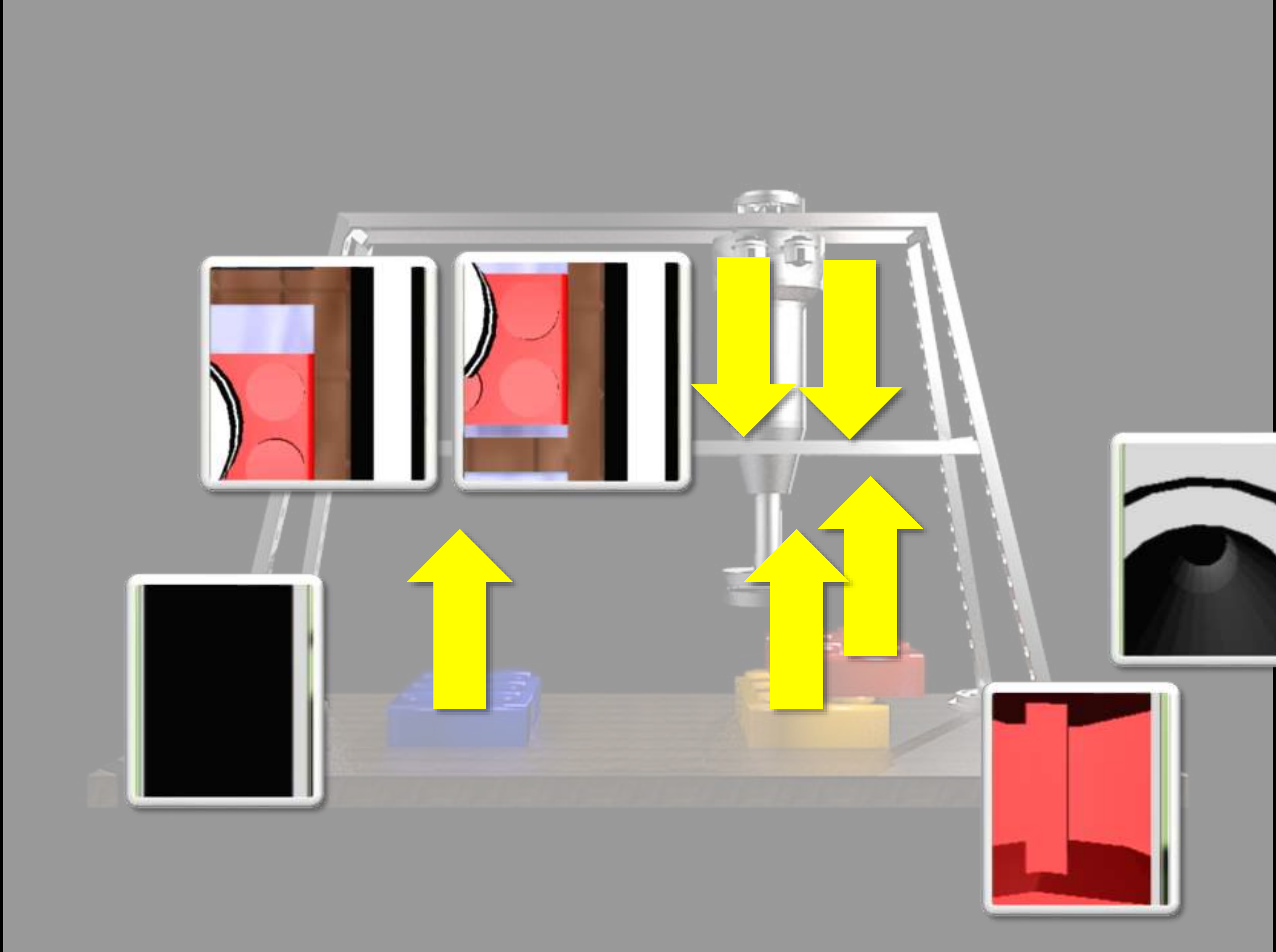
- Want to use minimal models
  - Model checking is computationally expensive
  - The temporal dimension exacerbates (1 minute trace @ 100ms sample time @ 10^9 states @ 2^8 values)
- Based on what you want to achieve
- Model at the appropriate level of abstraction!

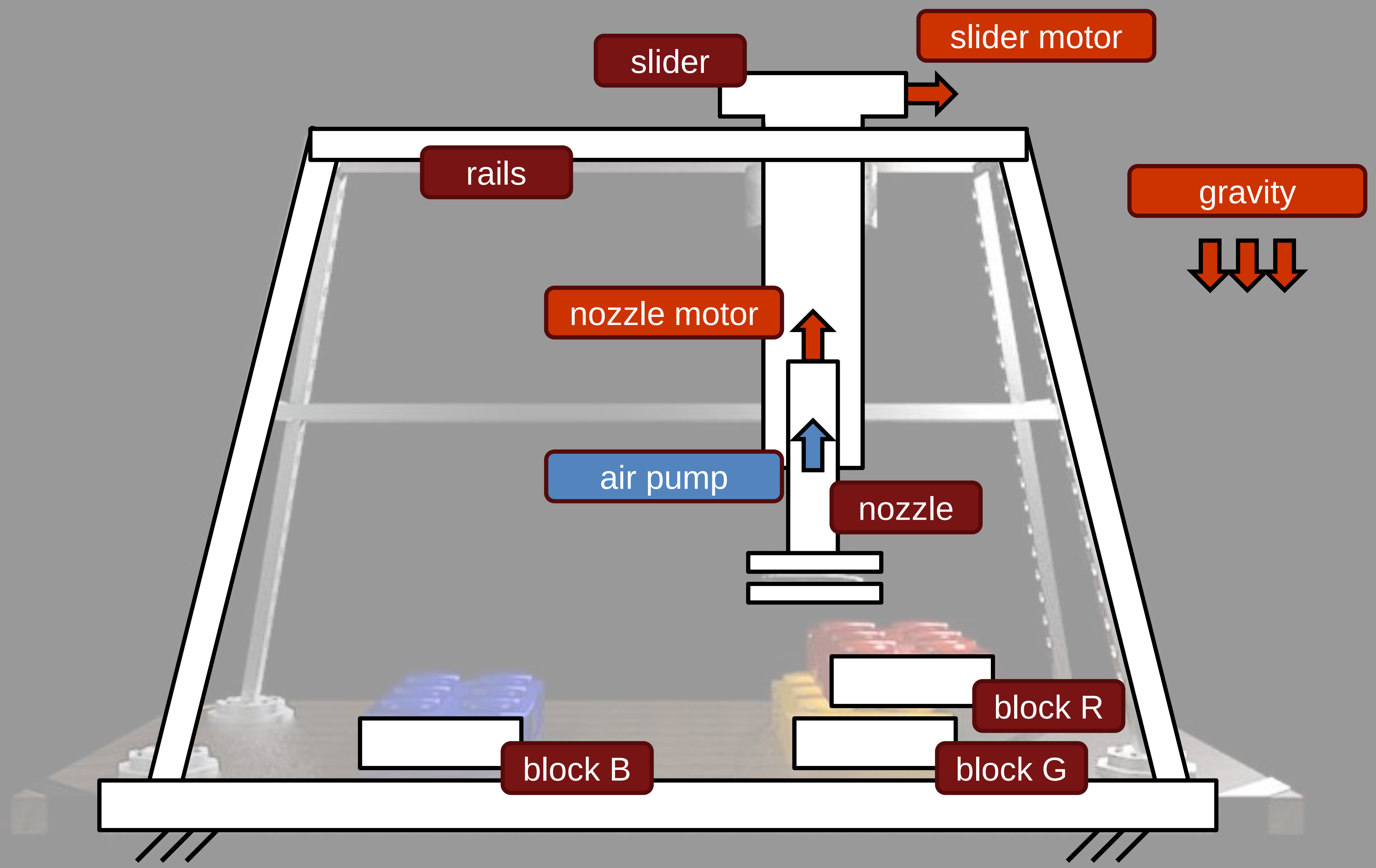

System characterization
A requirements perspective
The Towers of Hanoi revisited
Multiformalism

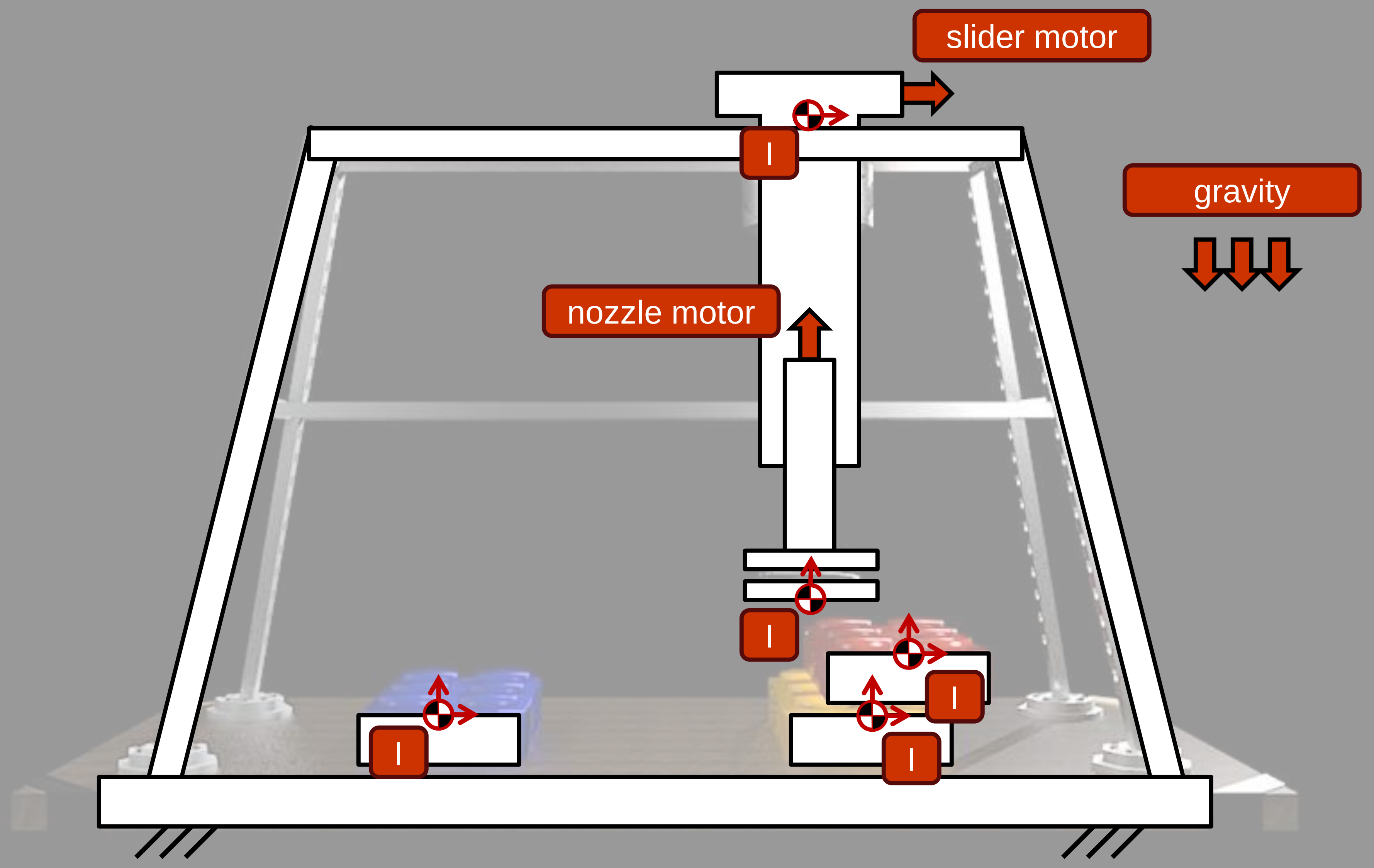
Towers of Hanoi

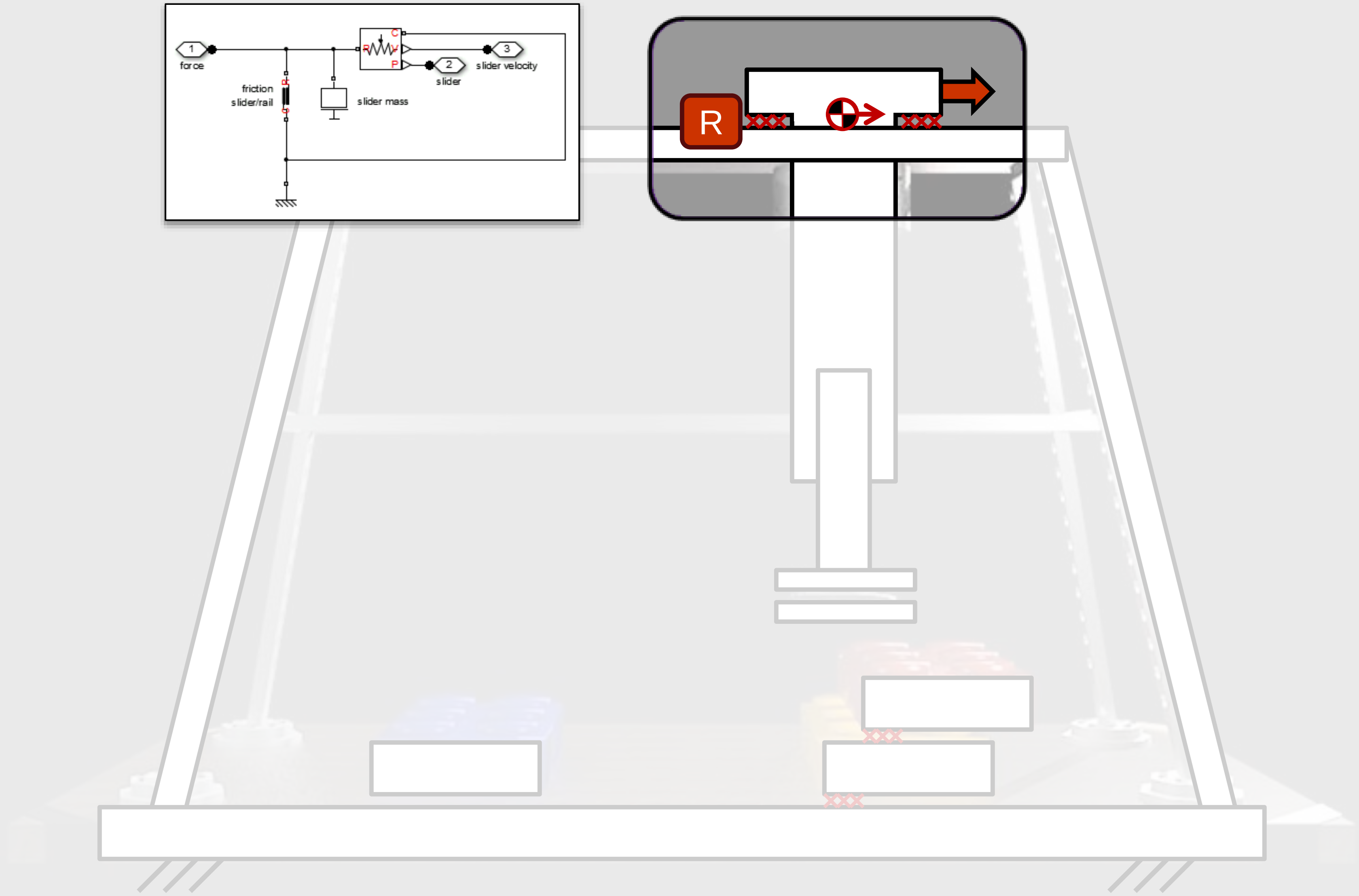


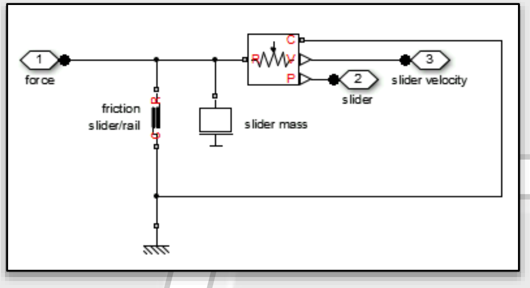


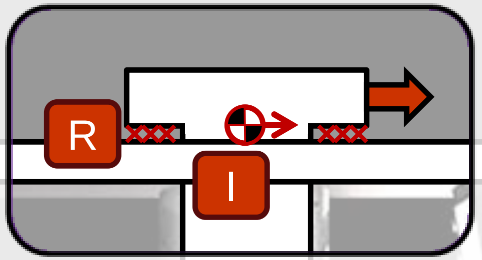



slider
slider motor
rails
gravity
nozzle motor
air pump
nozzle
block R
block B
block G

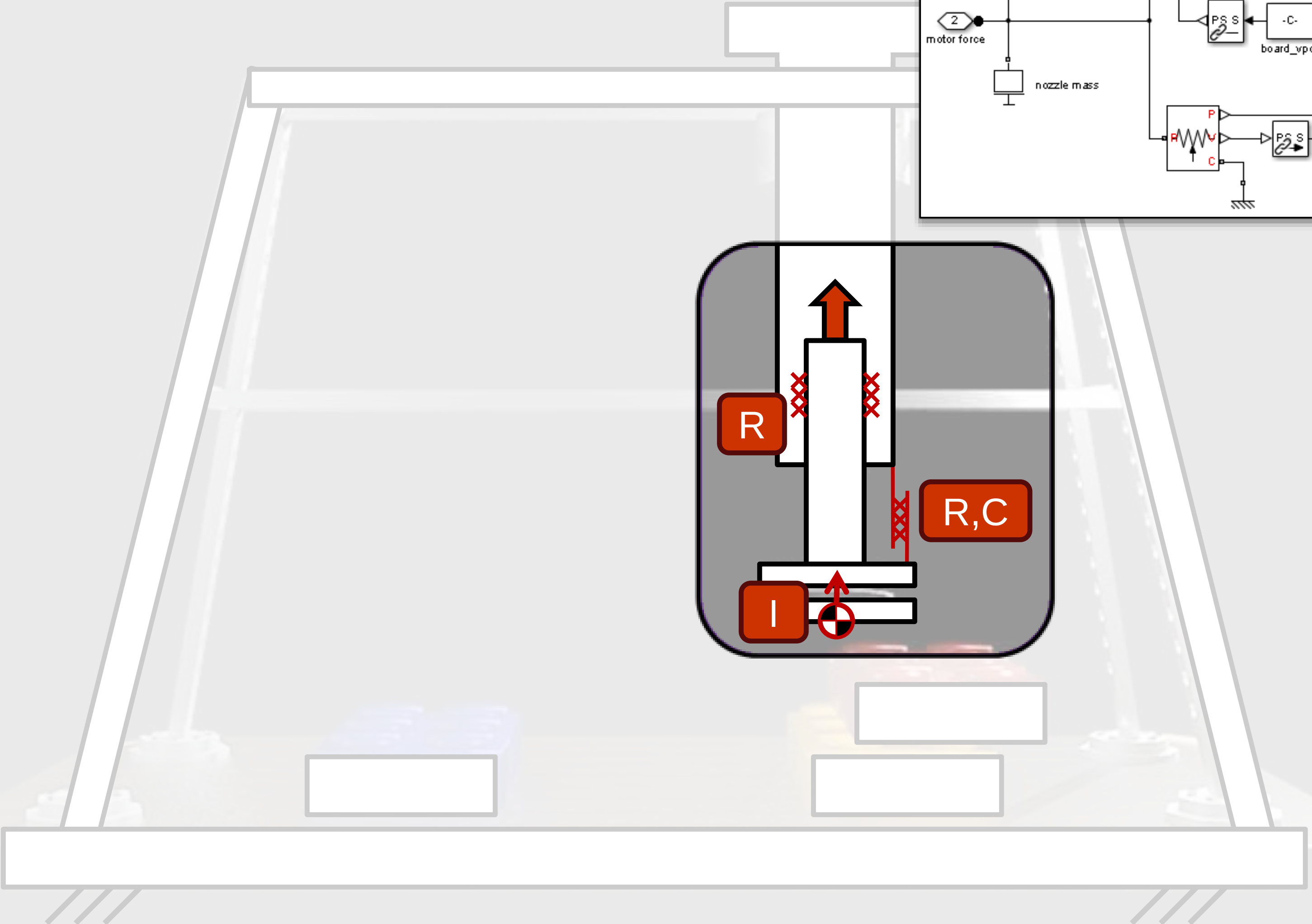

slider motor
gravity
nozzle motor

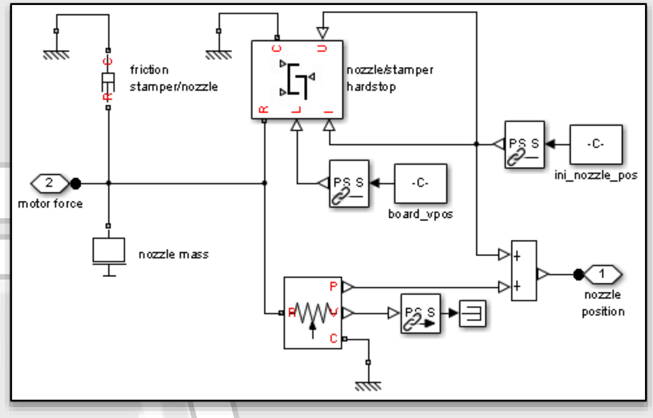

1
force
friction
slider/rail
slider mass
2
slider
3
slider velocity
R



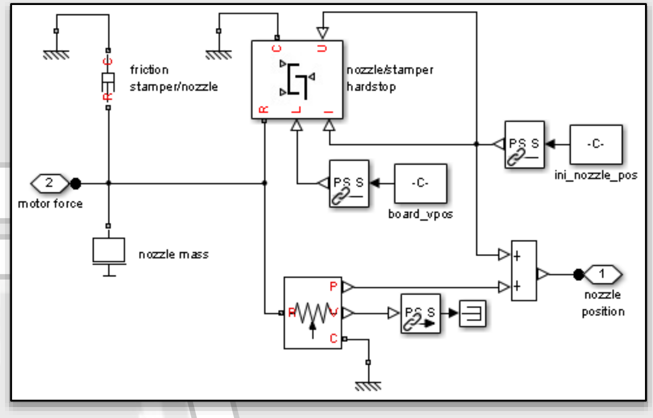


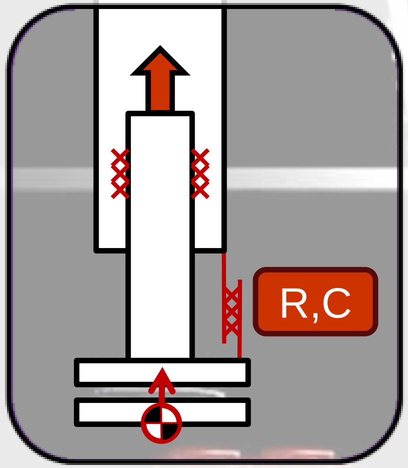

```
parameters
   brkwy_frc = { 25, 'N' };           % Breakaway friction force
   Col_frc = { 20, 'N' };             % Coulomb friction force
   visc_coef = { 100, 'N*s/m' };      % Viscous friction coefficient
   trans_coef = { 10, 's/m' };        % Transition approximation coefficient
   vel_thr = { 1e-4, 'm/s' };         % Linear region velocity threshold
 end

 parameters (Access=private)
   brkwy_frc_th = { 24.995, 'N' };    % Breakaway force at threshold velocity
 end

 function setup
   % Computing breakaway friction force at threshold velocity
   brkwy_frc_th = visc_coef * vel_thr + Col_frc + (brkwy_frc - Col_frc) * ...
       exp(-trans_coef * vel_thr);
 end

 equations
   if (abs(v) <= vel_thr)
       % Linear region
       f == brkwy_frc_th * v / vel_thr;
   elseif v > 0
       f == visc_coef * v + Col_frc + ...
           (brkwy_frc - Col_frc) * exp(-trans_coef * v);
   else
       f == visc_coef * v - Col_frc - ...
           (brkwy_frc - Col_frc) * exp(-trans_coef * abs(v));
   end
 end
```


R
R,C
I

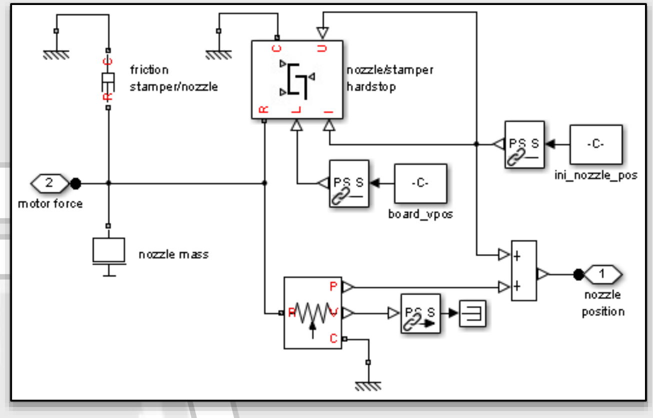

friction
stamper/nozzle
nozzle/stamper
hardstop
2
motor force
nozzle mass
-C-
board_vpos
-C-
ini_nozzle_pos
1
nozzle
position

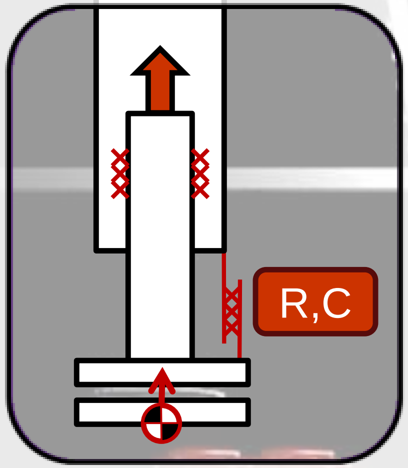
```
component hardstop_external_ini < foundation.mechanical.translational.branch
% Translational Hard Stop With External Initial Position

  parameters
    stiff_up = { 1e6, 'N/m' };        % Contact stiffness at upper bound
    stiff_low = { 1e6, 'N/m' };       % Contact stiffness at lower bound
    D_up = { 150, 'N*s/m'};           % Contact damping at upper bound
    D_low = { 150, 'N*s/m'};          % Contact damping at lower bound
  end

  inputs
    upper_bnd = { 0.1, 'm' };         % U:right
    lower_bnd = { -0.1, 'm' };        % L:left
    x_initial = { 0.0, 'm' };         % I:left
  end

  variables
    x = { 0, 'm'};
  end

  function setup
    if stiff_up <= 0
      pm_error('simscape:GreaterThanZero','Stiffness')
    end
    x = 0.0;
  end

  equations
    if ((x + x_initial) > upper_bnd)
        % Slider hits upper bound
        f == stiff_up * ((x + x_initial) - upper_bnd) + D_up * v;
    elseif ((x + x_initial) < lower_bnd)
        % Slider hits lower bound
        f == stiff_low * ((x + x_initial) - lower_bnd) + D_low * v;
    else
        % Slider is between hardstops
        f == {0 'N'};
    end
    x.der == v;
  end
end
```

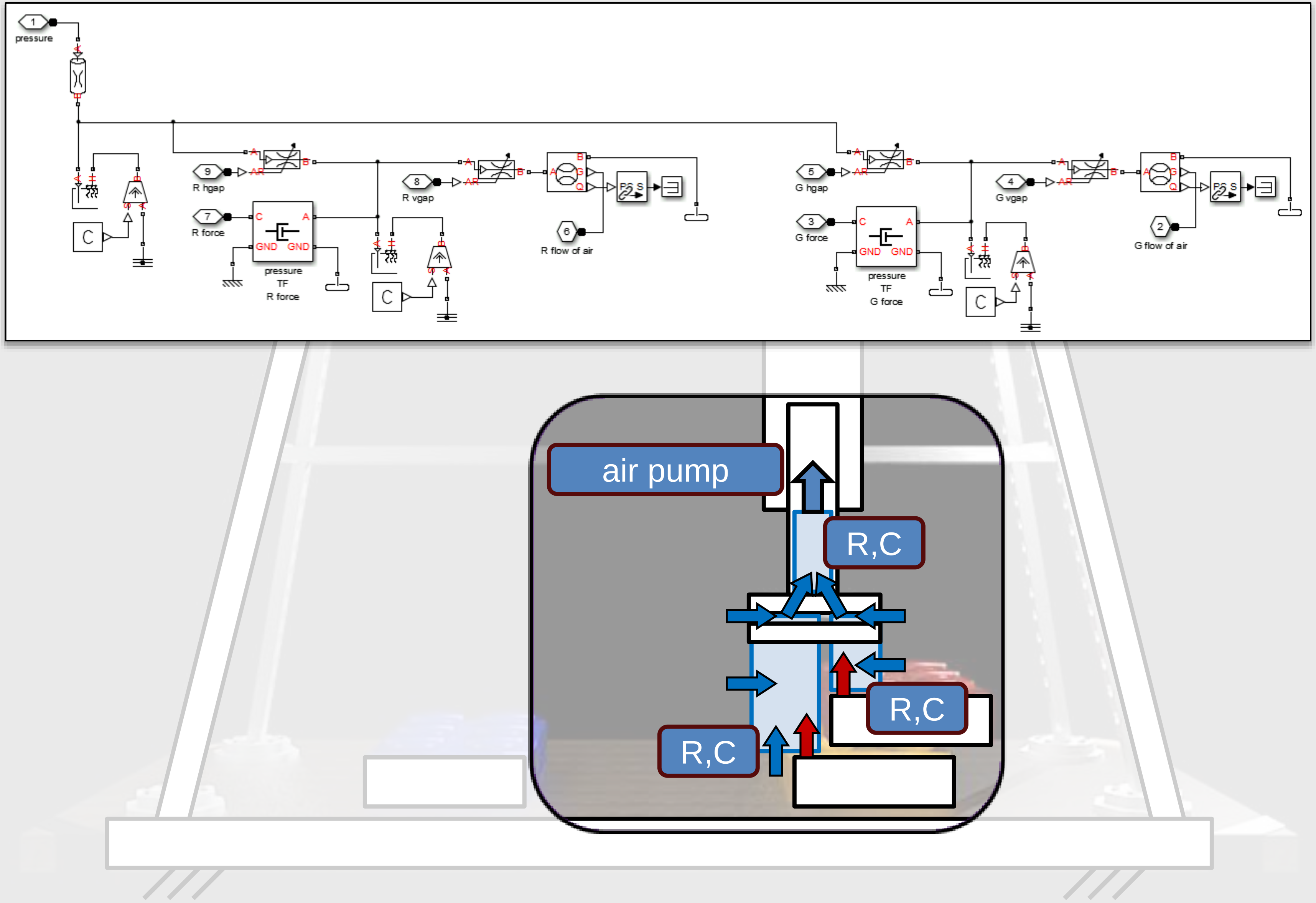


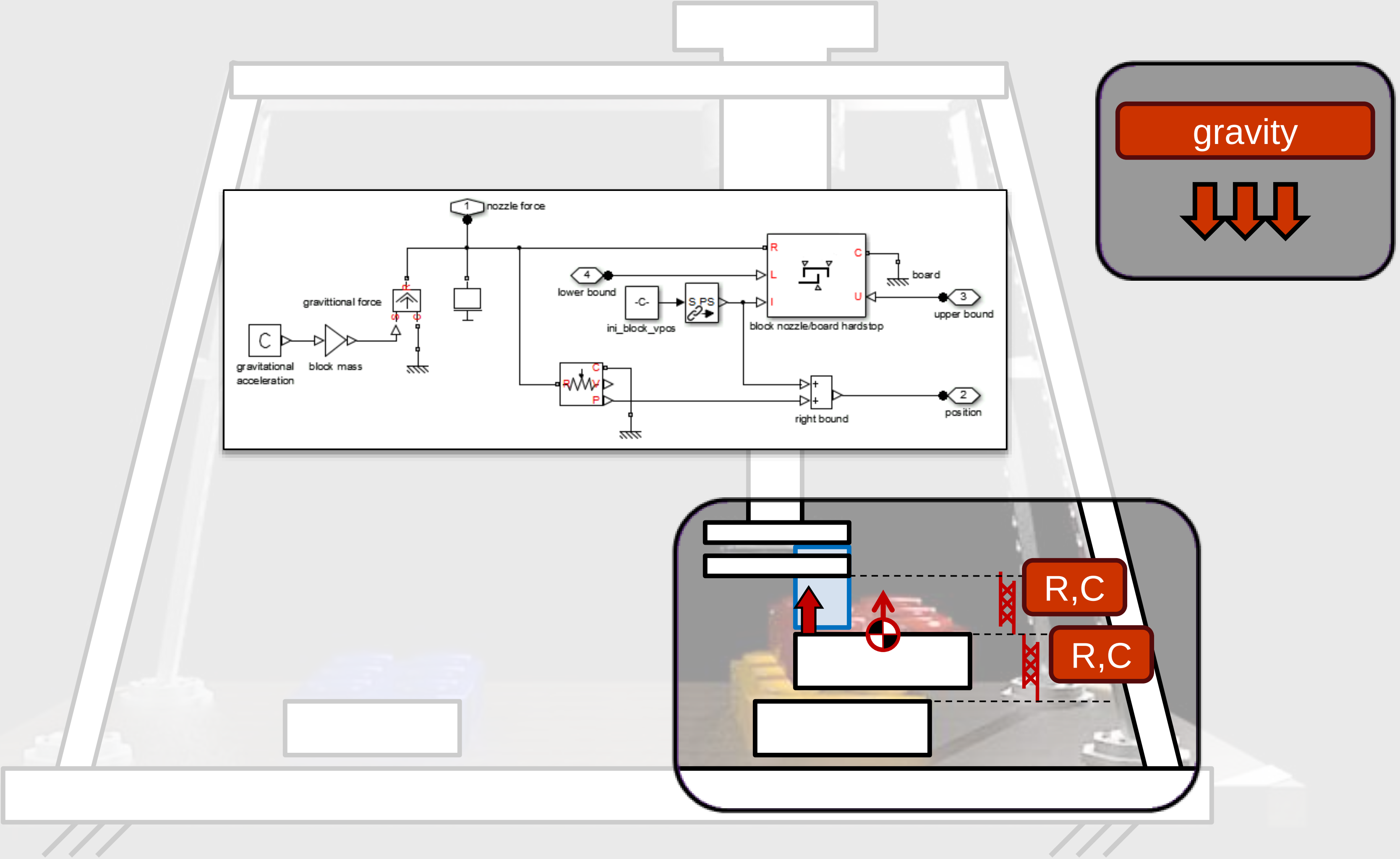


```
component(Hidden=true) branch
% Translational Branch
% Defines a translational branch with R and C external nodes.
% Also defines associated through and across variables.

% Copyright 2005-2008 The MathWorks, Inc.

  nodes
    R = foundation.mechanical.translational.translational; % R:left
    C = foundation.mechanical.translational.translational; % C:right
  end

  variables
    f = { 0, 'N' };
    v = { 0, 'm/s' };
  end

  function setup
    through( f, R.f, C.f );
    across( v, R.v, C.v );
  end

end
```

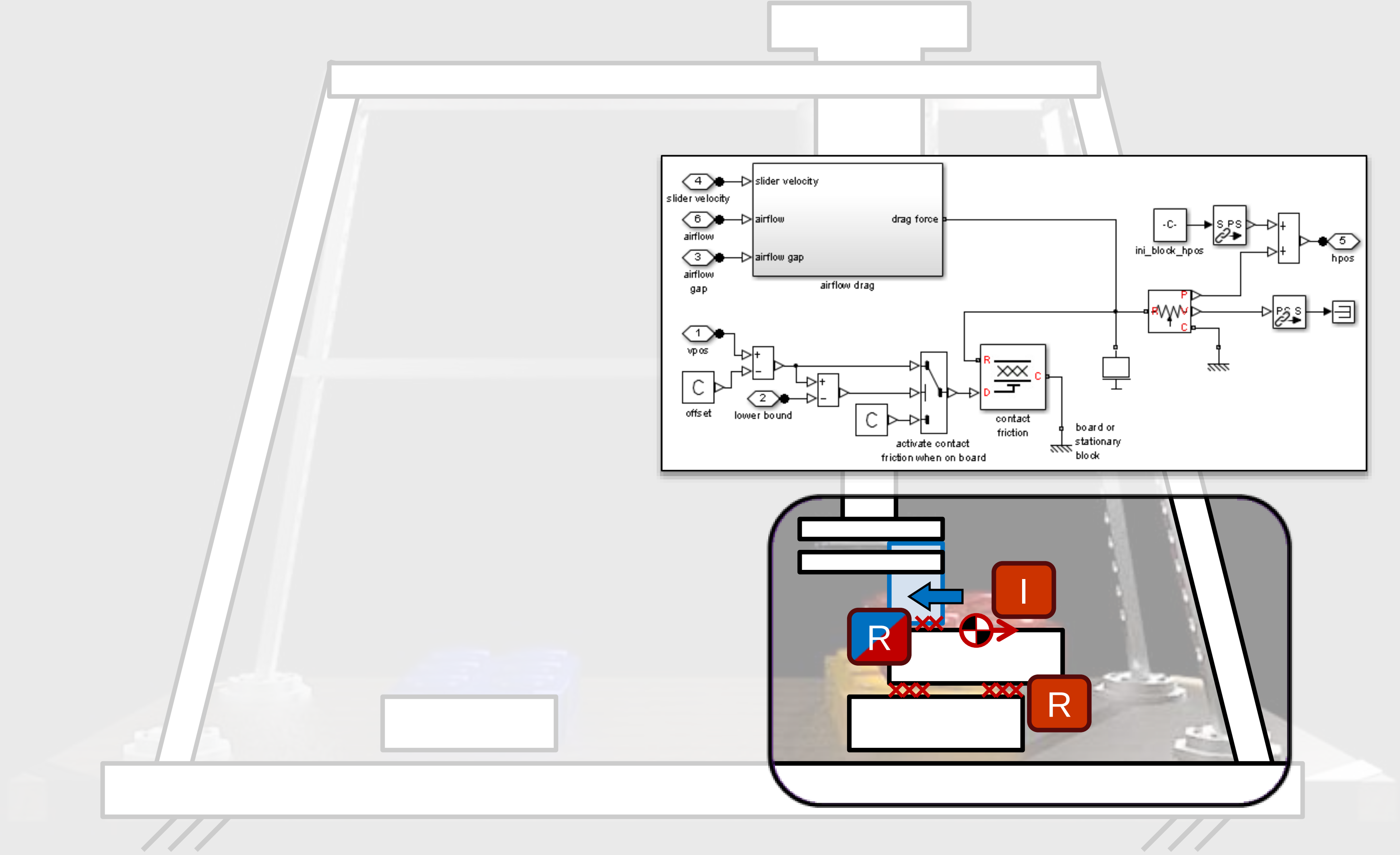




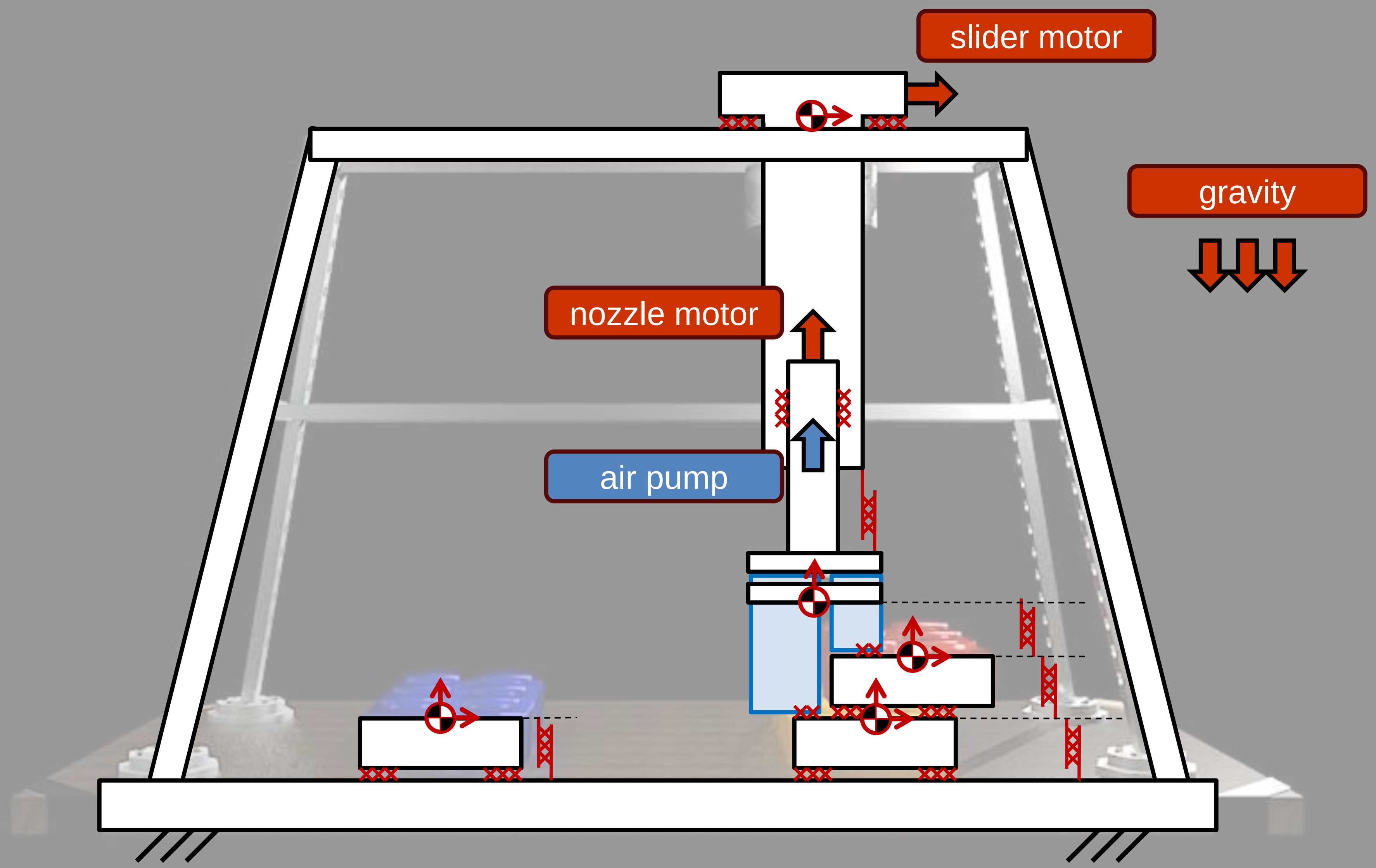

pressure
R hgap
R force
pressure
TF
R force
R vgap
R flow of air
G hgap
G force
pressure
TF
G force
G vgap
G flow of air
air pump
R,C
R,C
R,C



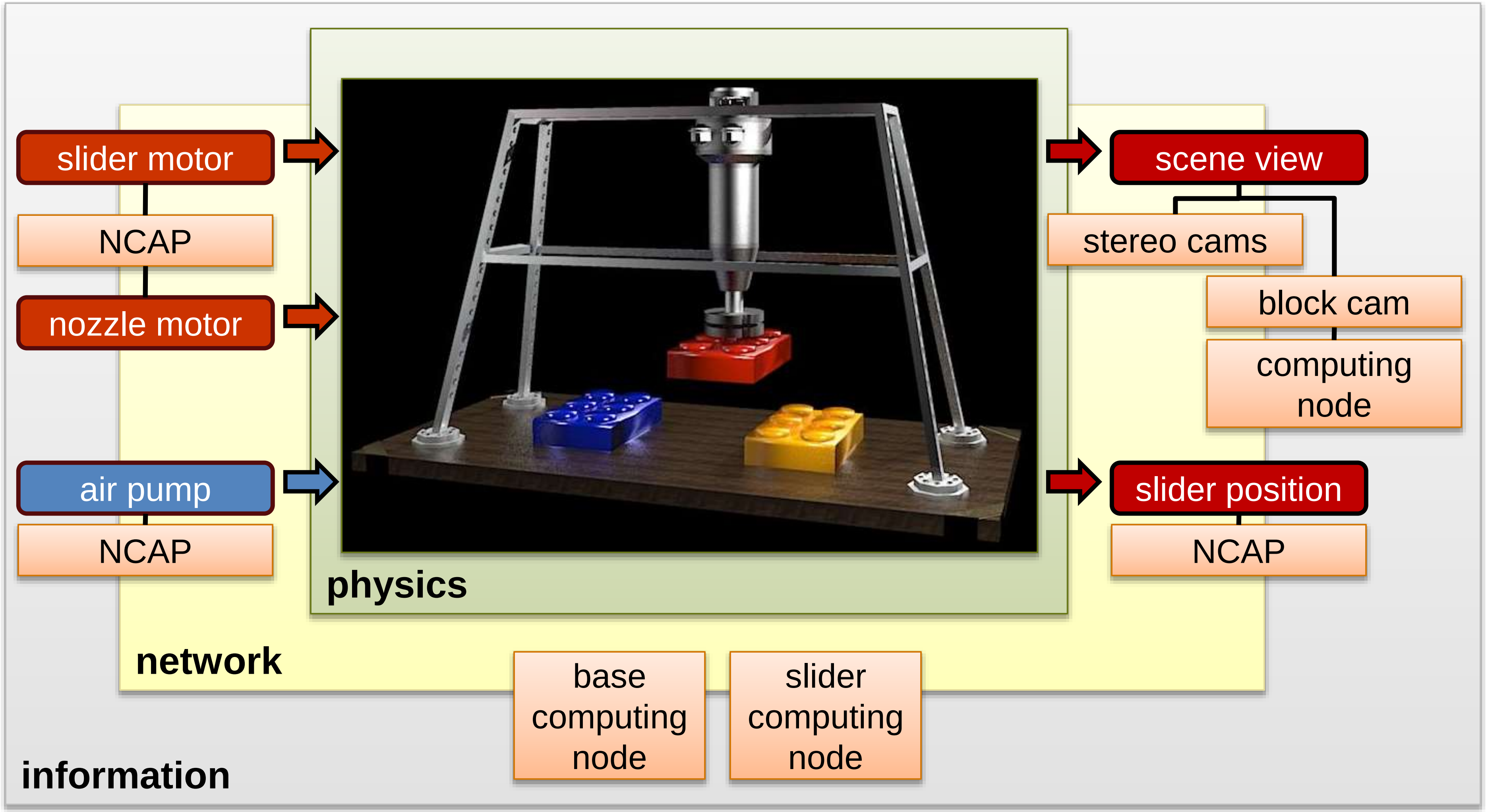
nozzle force
gravittional force
gravitational acceleration
block mass
lower bound
ini_block_vpos
block nozzle/board hardstop
board
upper bound
right bound
position
R,C
R,C
gravity



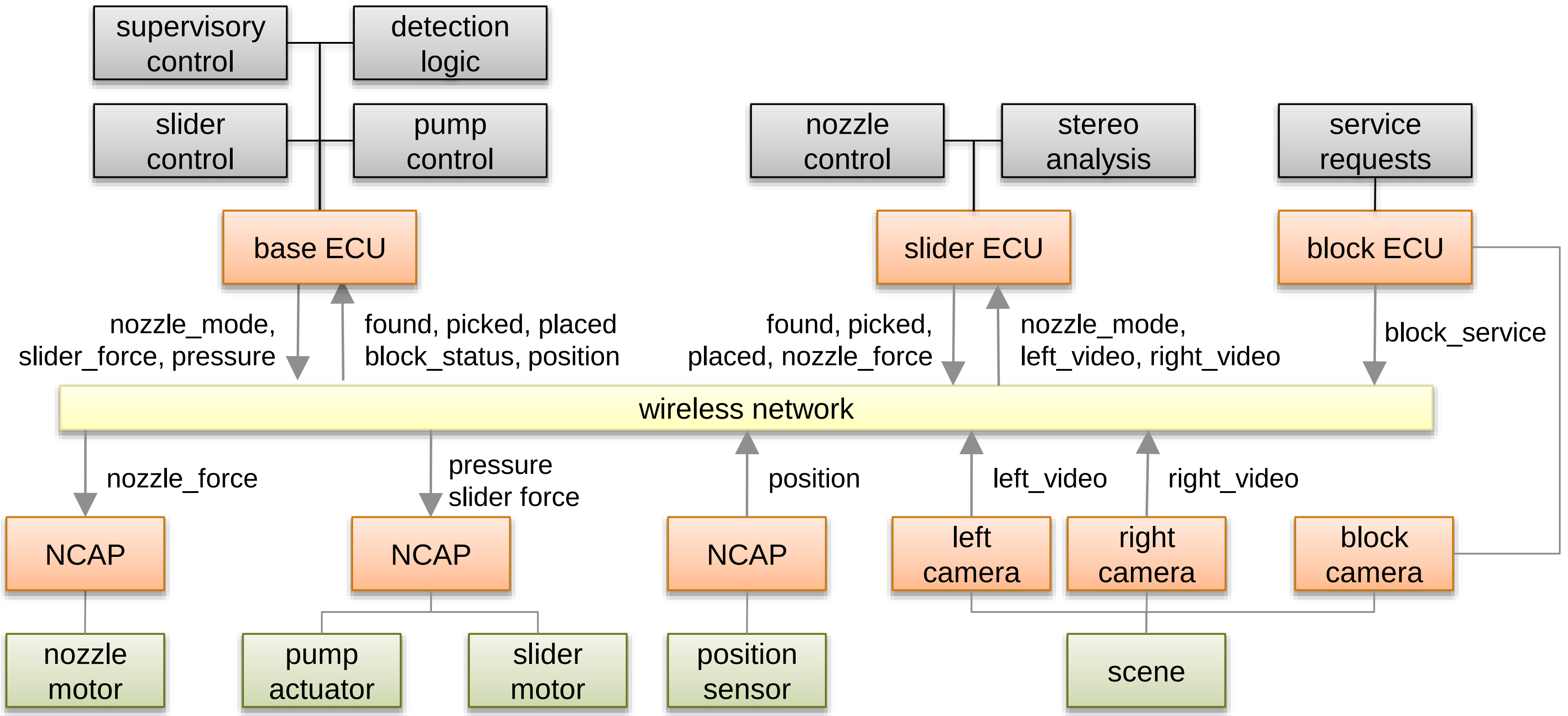
slider velocity
airflow
airflow gap
drag force
airflow drag
ini_block_hpos
hpos
vpos
offset
lower bound
activate contact friction when on board
contact friction
board or stationary block
I
R
R


slider motor
gravity
nozzle motor
air pump


slider motor
nozzle motor
air pump
scene view
slider position
physics

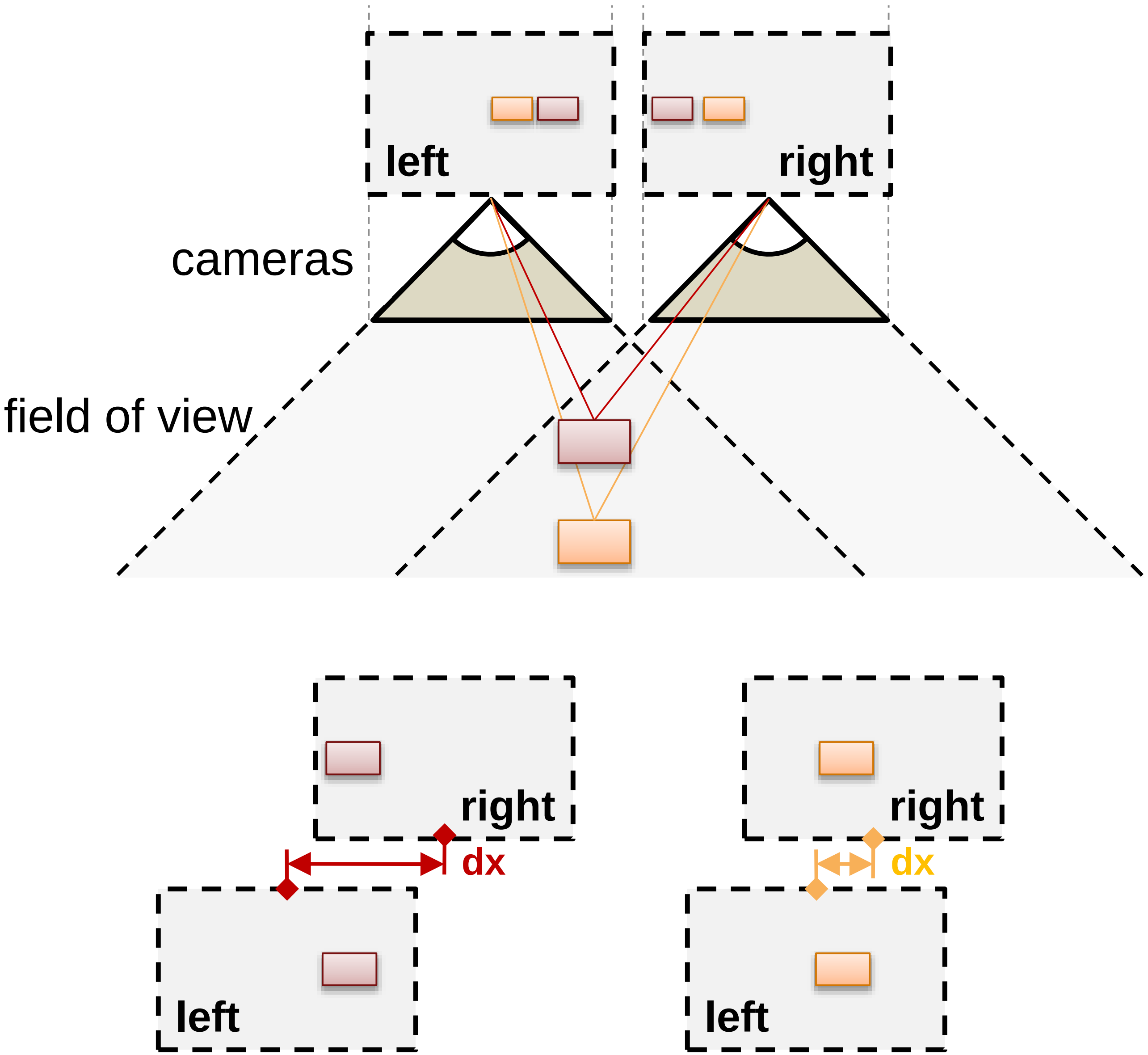

slider motor
NCAP
nozzle motor
air pump
NCAP
scene view
stereo cams
block cam
computing node
slider position
NCAP
physics
network
base computing node
slider computing node
information

An architecture

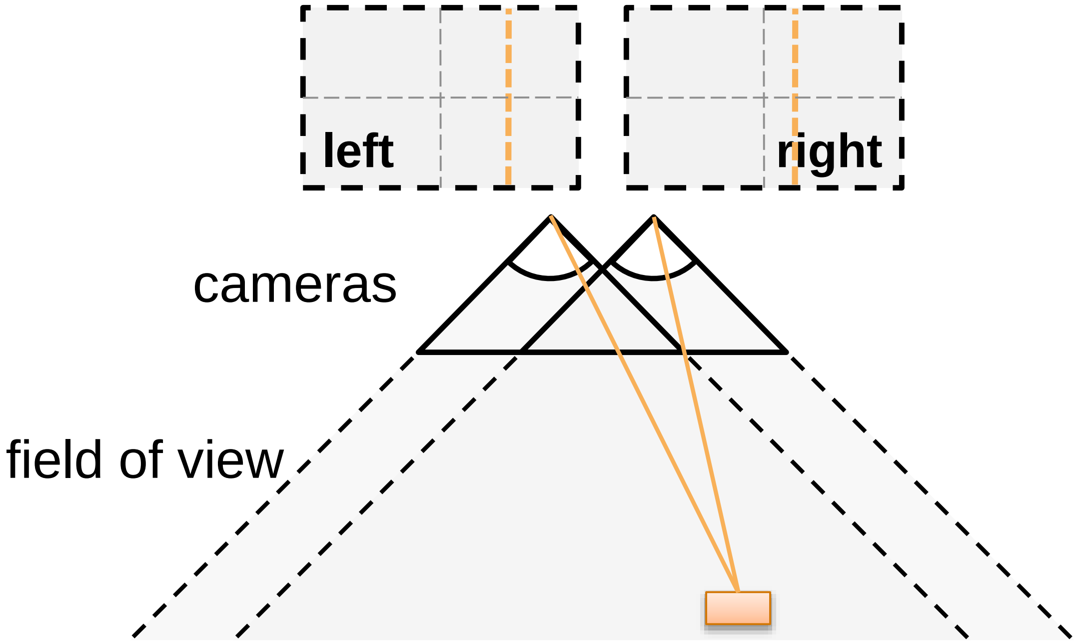


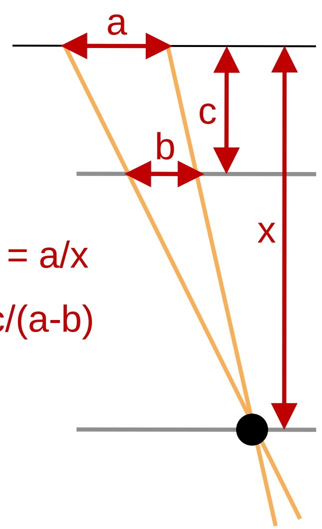
Stereoscopic vision on a synthesized video stream

Stereopsis



Stereopsis

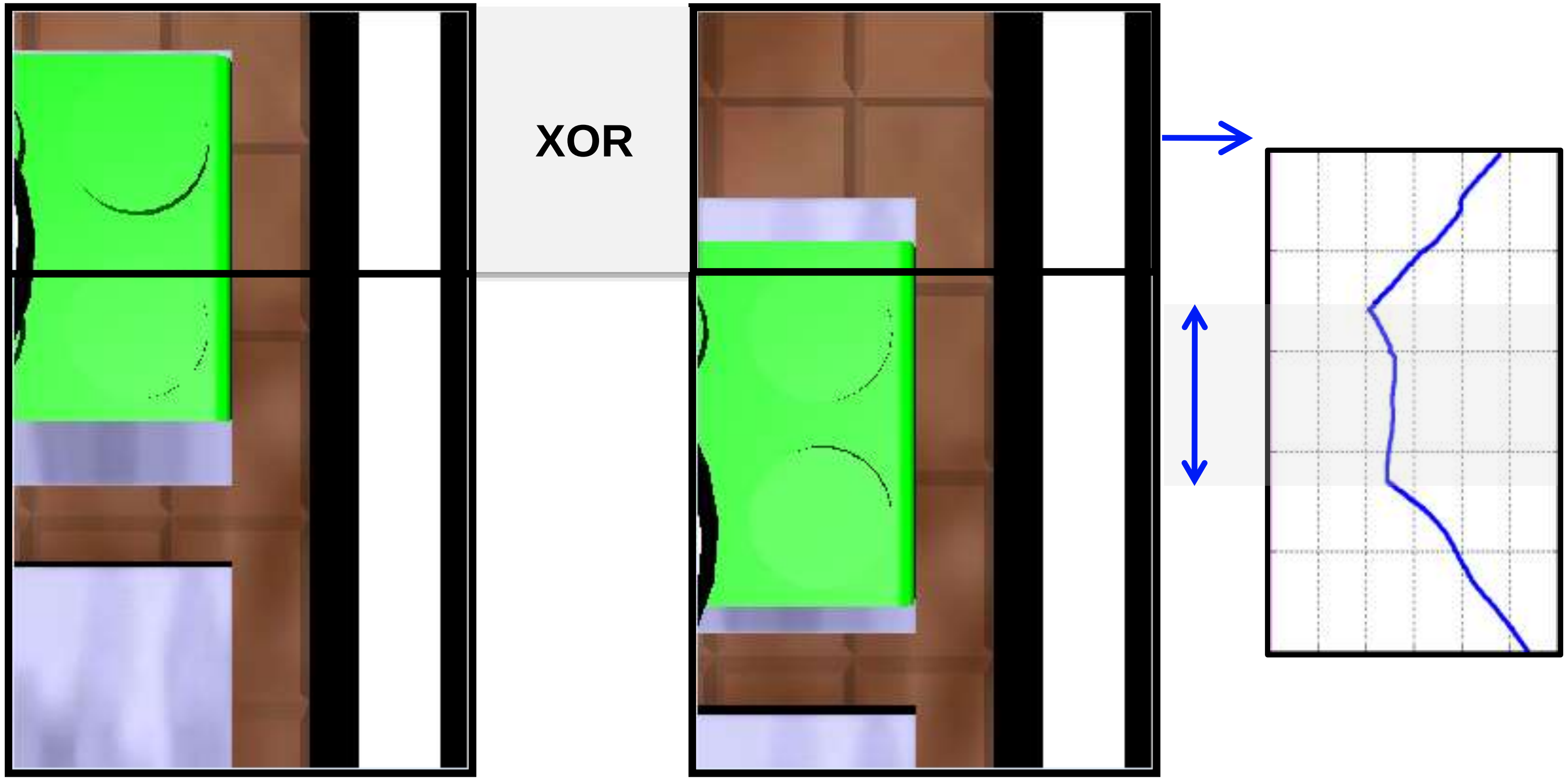




http://www.alexandria.nu/ai/blog/entry.asp?E=32

Stereoscopic analysis on a synthesized video stream

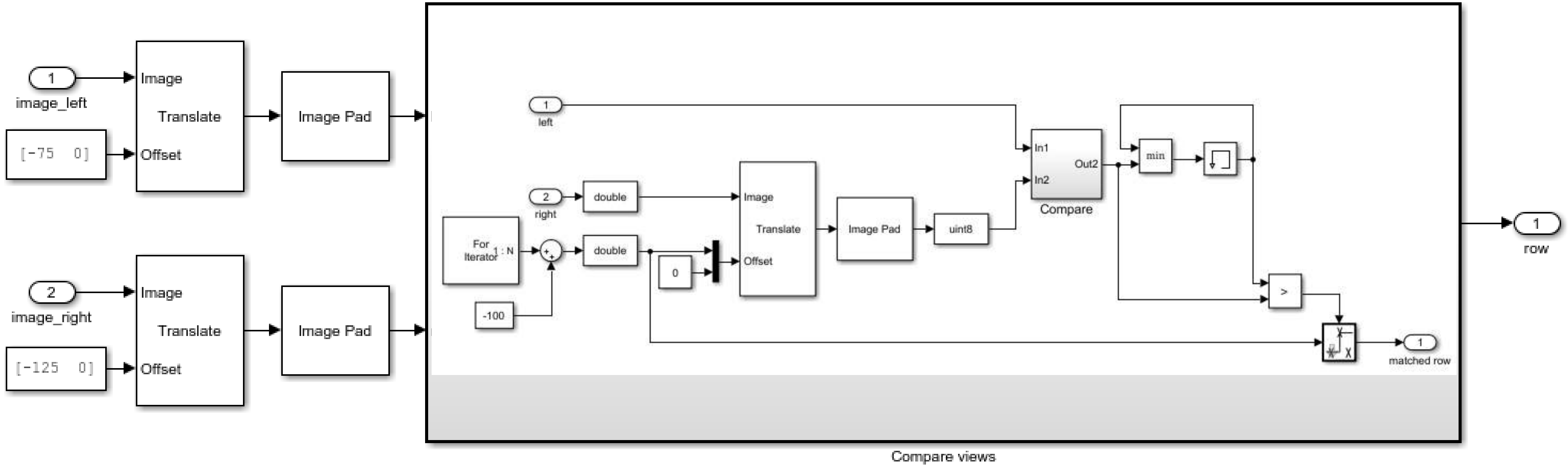
- Embarrassingly parallel



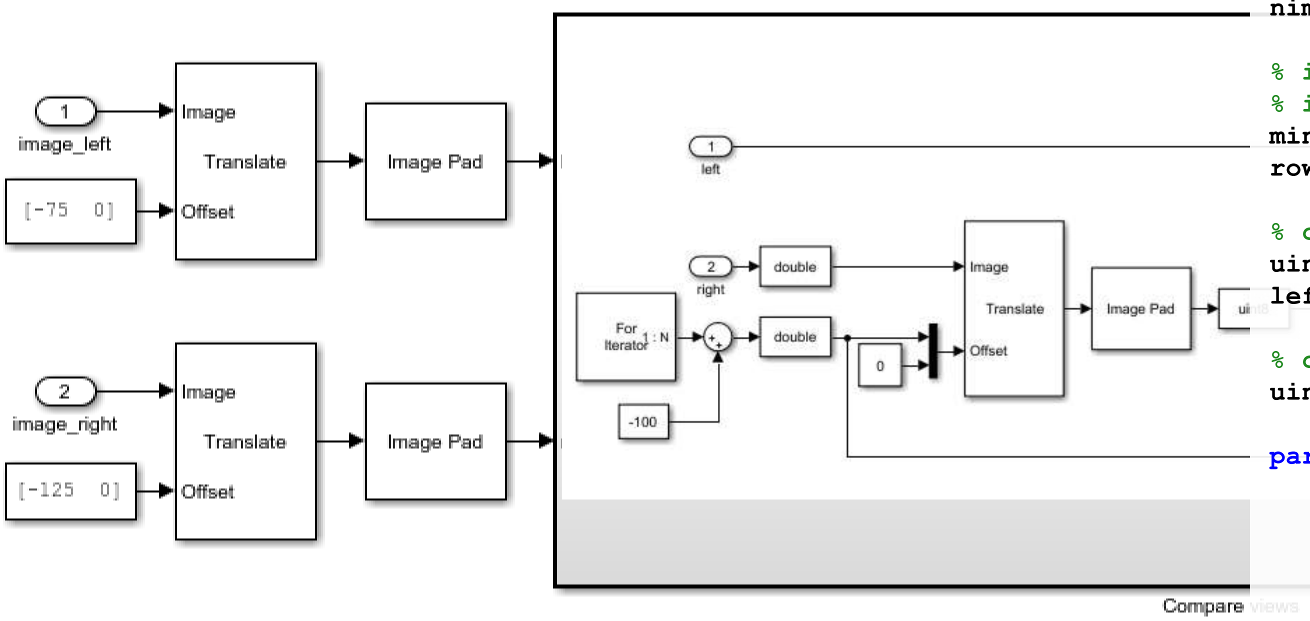
left video frame

right video frame

Stereoscopy implementations



Stereoscopy implementations

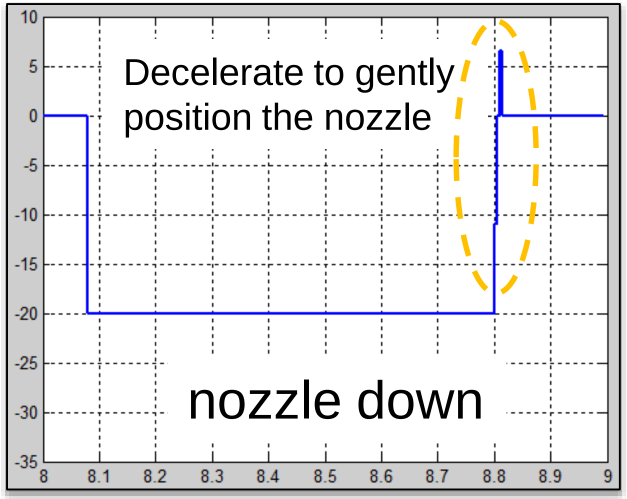


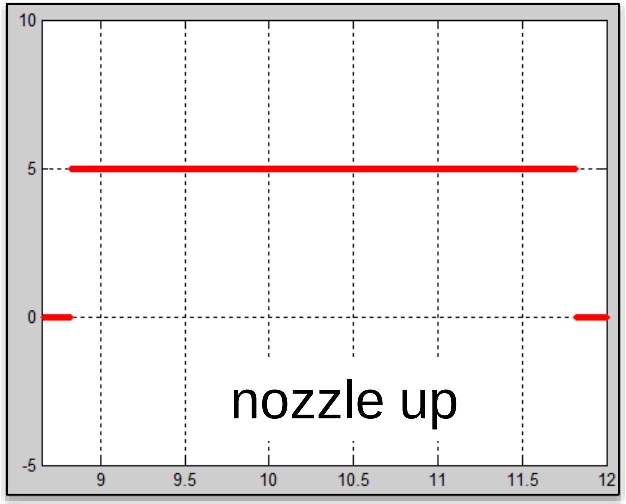

```
function row = fcn(image_left, image_right)
%#codegen

persistent hmean;
if isempty(hmean)
    hmean = vision.Mean();
end

% number of successive image comparisons
nimages = 100;

% initialize the minimum mean and the corresponding row at which this mean
% is found for the number of successive image comparisons
min = 1e5;
row = 0;

% compute left image submatrix to successively compare
uint8_video_left = uint8(image_left);
left = uint8_video_left(75:224, 1:120, :);

% compute uint8 version of right image
uint8_video_right = uint8(image_right);

parfor k=1:nimages
    % compute successive right image submatrices for comparison
    right = uint8_video_right(125+k:125+149+k, 1:120, :);

    % compare left and right image submatrices
    cmp = bitxor(left,right);

    % compute the mean over all pixels of the comparison results
    pixel_mean = step(hmean,double(cmp));

    % in case of the final row only accept a substantially less
    % (at least 2) value of the mean
    if (pixel_mean < min && k < 100) || (pixel_mean < min - 2)
        min = pixel_mean;
        row = k;
    end
end
```

Feedforward (fast) control

Nozzle control profile

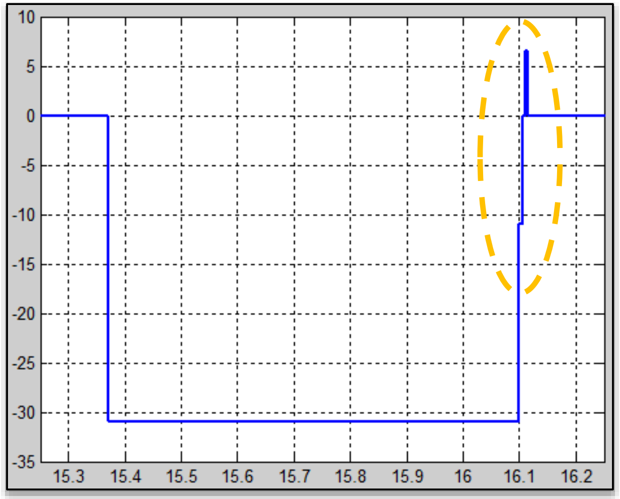
Pick/place for a stack of two blocks

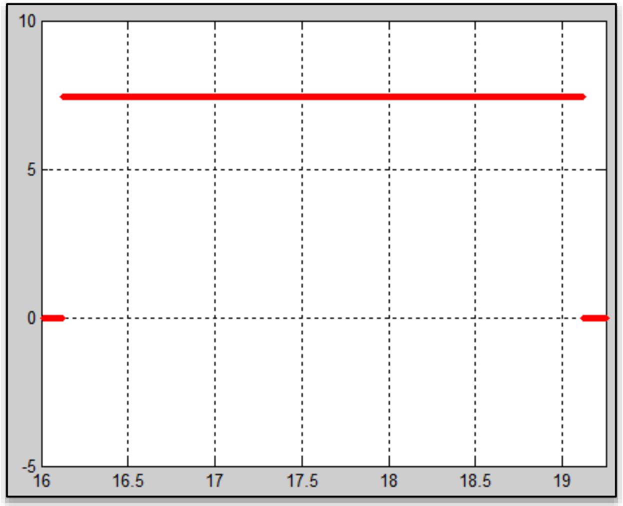




A physical nozzle/slider hardstop limits the up motion and provides a guaranteed reference starting point for the next operation, thus enabling feedforward control

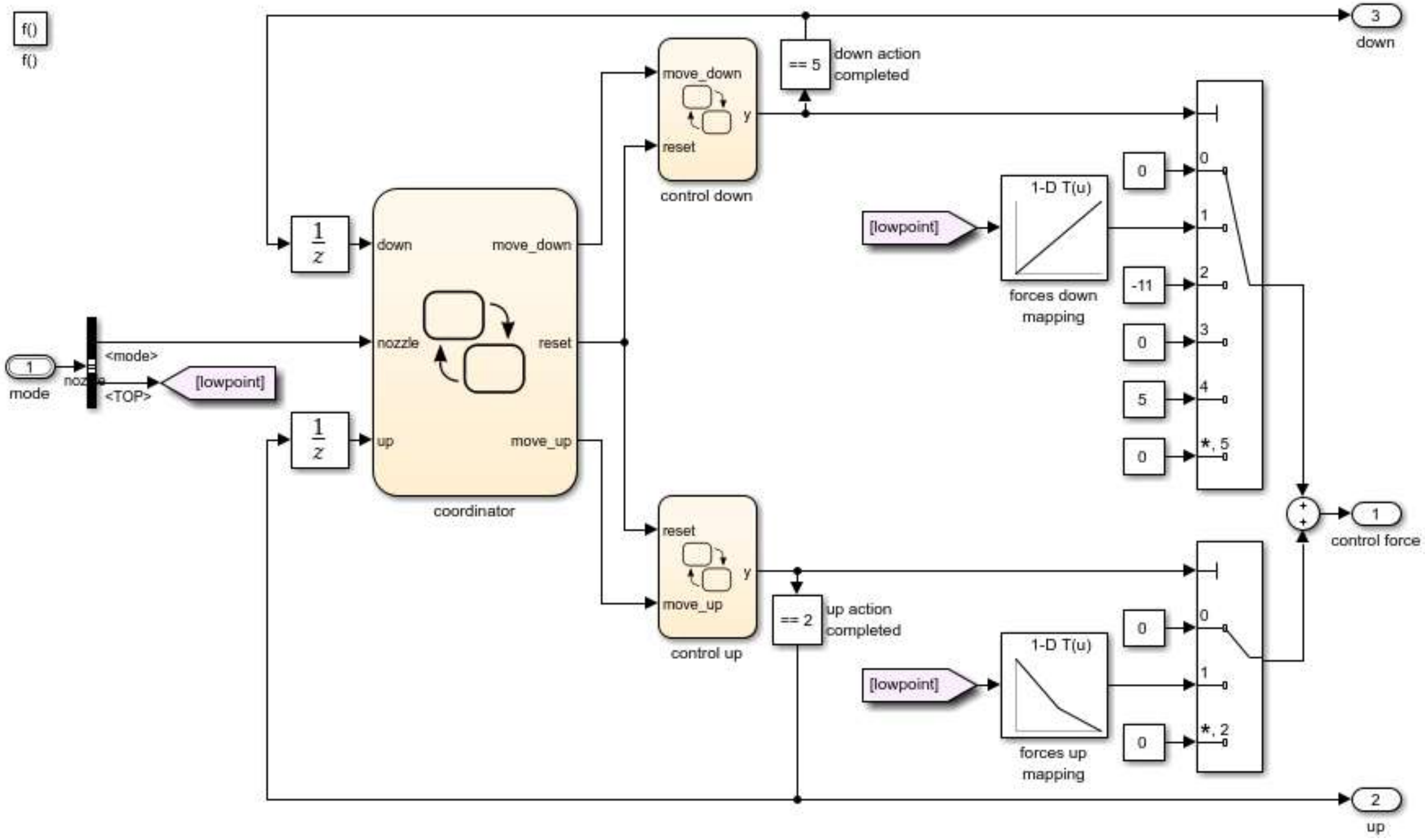
Pick/place for a stack of one block



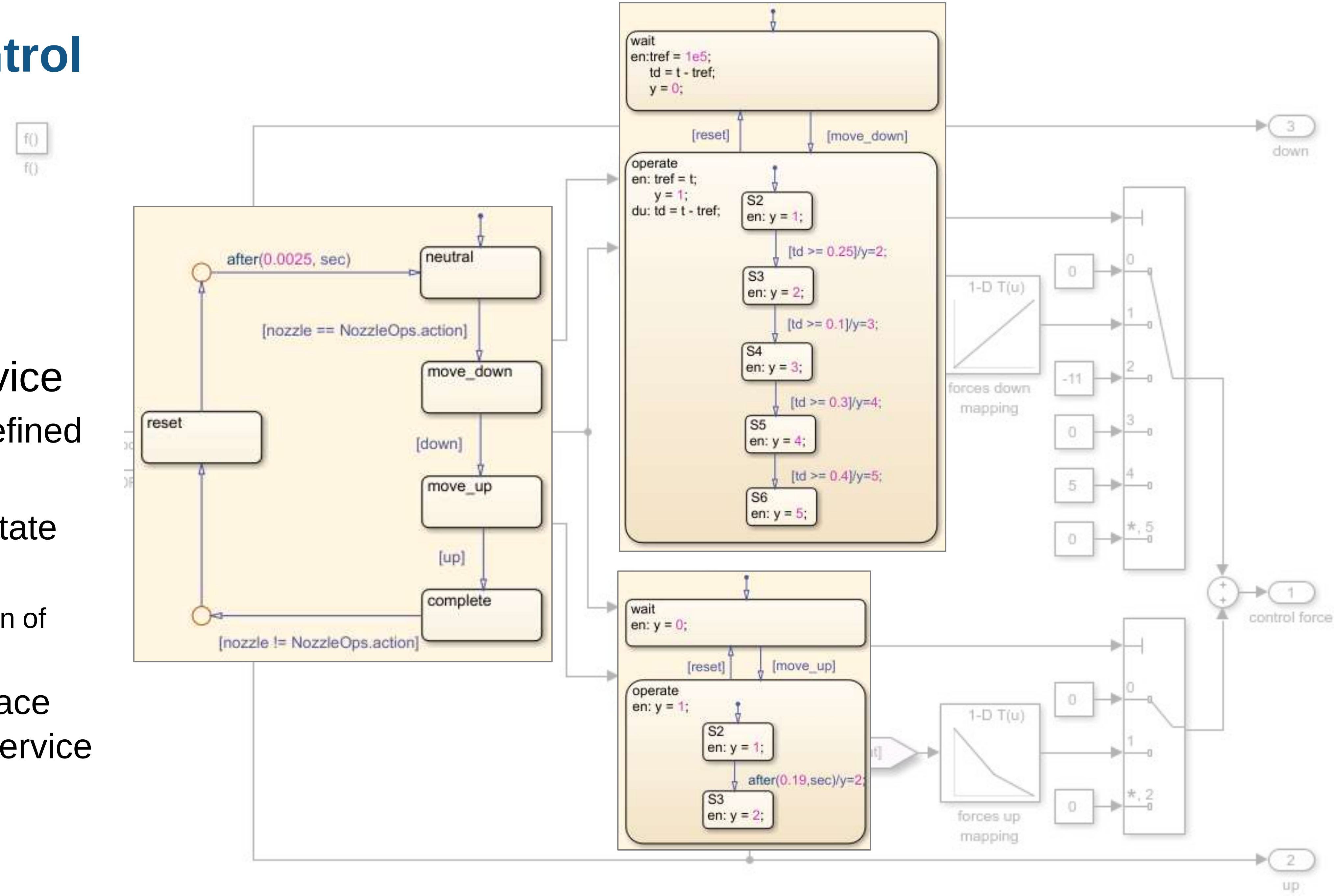


Nozzle control

- Feedforward (very fast) control
- Two phases (down/up)
  - Staged force profiles
  - Predetermined profiles for set of possible lowpoints
    - Top of a stack of two blocks
    - Top of a stack of one block
  - Lookup table for each lowpoint



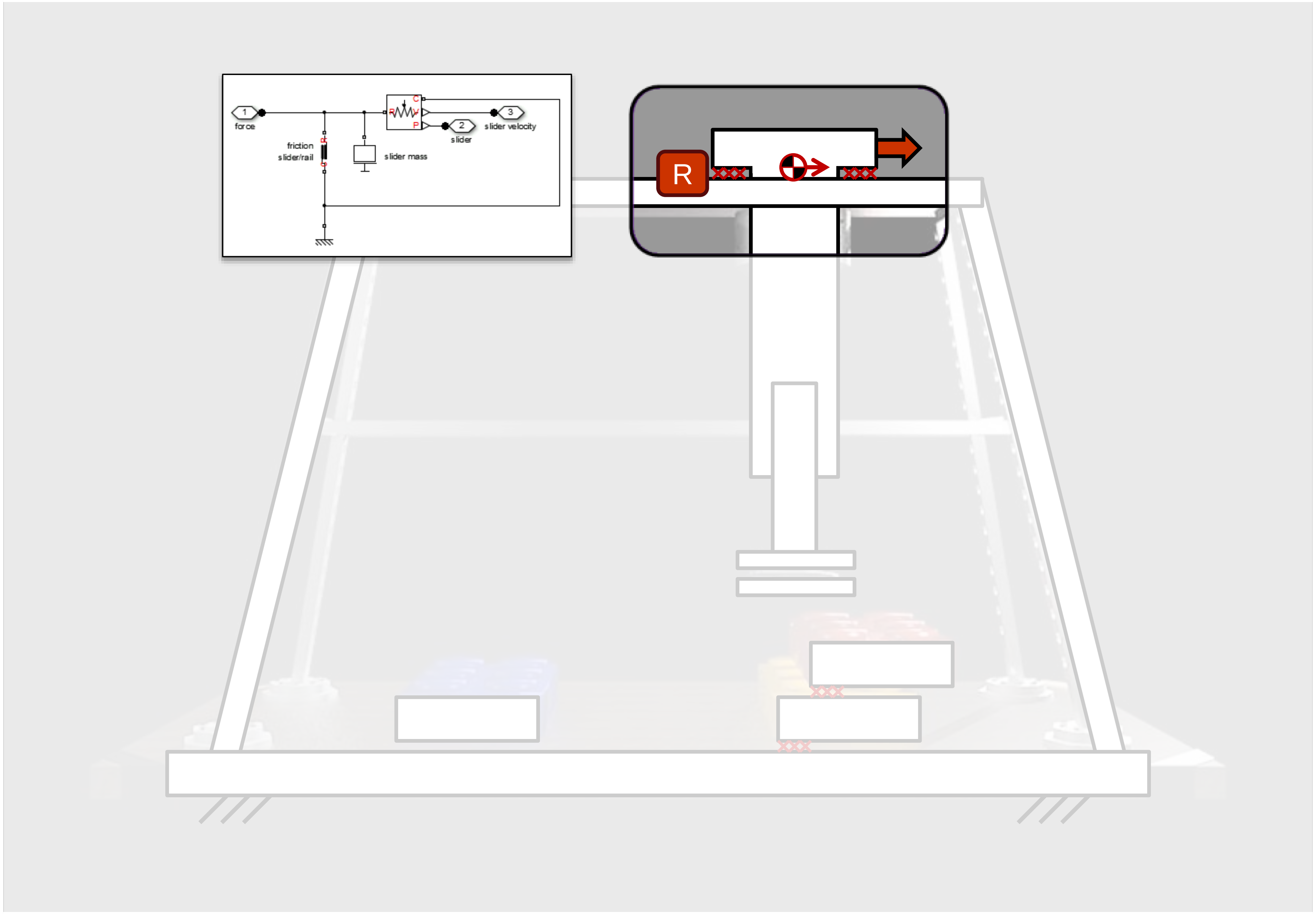
Nozzle control



Provide as a service

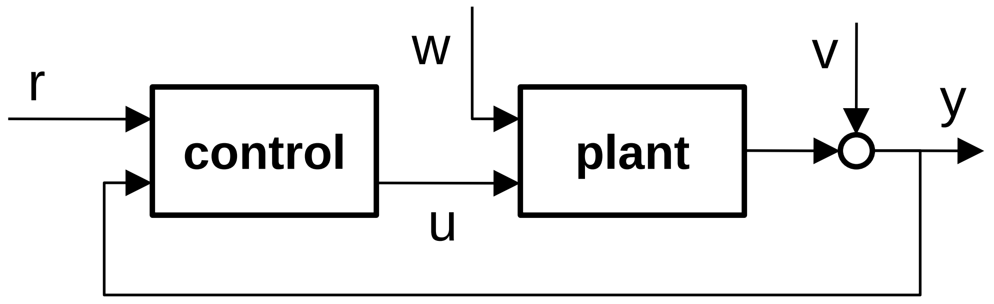
– Profile must be defined in relative time
– Reset operation state after completion
    ▪ Allows initialization of relative variables
– Hold off pick or place operation till the service is available

Feedback mode-switched control of electric drive


1
force
friction
slider/rail
slider mass
2
slider
3
slider velocity
R

Slider control

- Exert a motor force to move the slider to a give position
- Compute a Gaussian (lqg) regulator (output feedback)
  - r = desired slider position
  - u = motor force

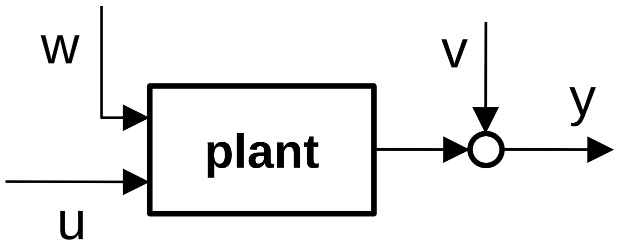


$$\min_u J(u) = E\left\{ \lim_{T\to\infty} \frac{1}{T} \int_0^T \left([x', u'] QXU \begin{bmatrix} x \\ u \end{bmatrix} + x_i' Q_i x_i \right) dt \right\}$$

$$x_i = \int_0^T (r - y) dt$$

Slider control

- Plant model



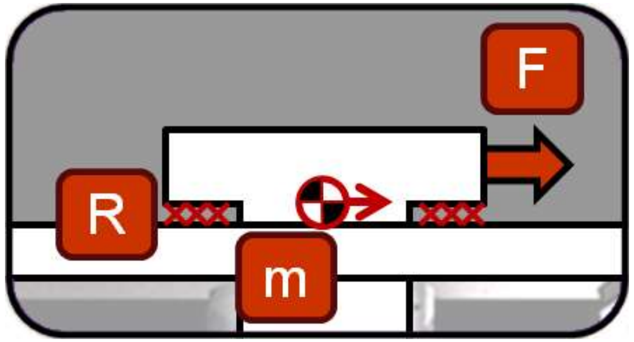
$$\frac{dx}{dt} = Ax + Bu + w$$
$$y = Cx + Du + v$$

- Requires a linear plant model
  - The slider/rail friction ruins it …

Plant model

- Gaussian regulator

```
parameters
    brkwy_frc = { 25, 'N' };          % Breakaway friction force
    Col_frc = { 20, 'N' };            % Coulomb friction force
    visc_coef = { 100, 'N*s/m' };     % Viscous friction coefficient
    trans_coef = { 10, 's/m' };       % Transition approximation coefficient
    vel_thr = { 1e-4, 'm/s' };        % Linear region velocity threshold
 end

 parameters (Access=private)
    brkwy_frc_th = { 24.995, 'N' };   % Breakaway force at threshold velocity
 end

 function setup
    % Computing breakaway friction force at threshold velocity
    brkwy_frc_th = visc_coef * vel_thr + Col_frc + (brkwy_frc - Col_frc) * ...
        exp(-trans_coef * vel_thr);
 end

 equations
    if (abs(v) <= vel_thr)
        % Linear region
        f == brkwy_frc_th * v / vel_thr;
    elseif v > 0
        f == visc_coef * v + Col_frc + ...
            (brkwy_frc - Col_frc) * exp(-trans_coef * v);
    else
        f == visc_coef * v - Col_frc - ...
            (brkwy_frc - Col_frc) * exp(-trans_coef * abs(v));
    end
 end
```



Linearization

- Linearization harness with I/O
- Breakaway at about 3 seconds

t = 0 seconds

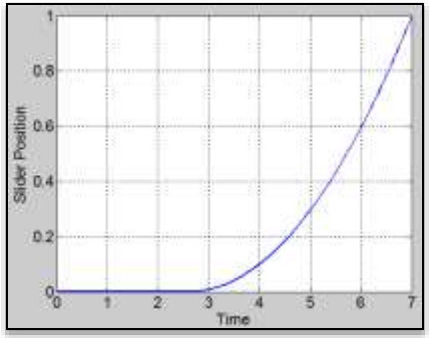
```
a =
              x1          x2
   x1          0           1
   x2          0   -2.5e+05

b =
       u1
   x1   0
   x2   1

c =
       x1   x2
   y1   1    0

d =
       u1
   y1   0
```

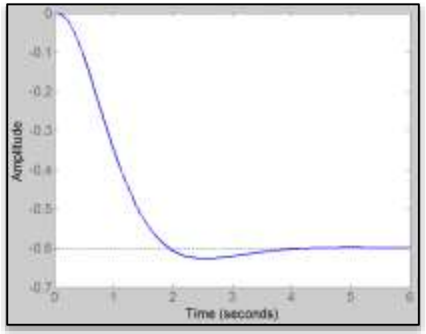
t = 5 seconds

```
a =
            x1        x2
   x1        0         1
   x2        0    -9.609

b =
       u1
   x1   0
   x2   1

c =
       x1   x2
   y1   1    0

d =
       u1
   y1   0
```

- Gaussian regulator for t = 5 seconds

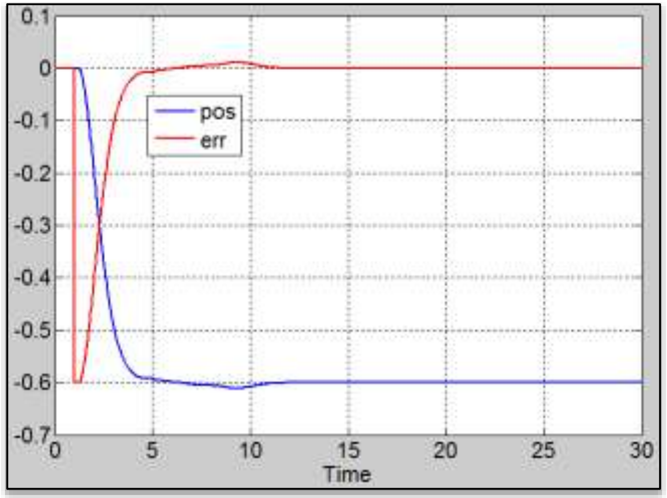
```
Pss = ss(a, b, c, d);
set(Pss,'inputn', {'force'});
set(Pss,'staten', {'pos', 'vel'});
set(Pss, 'outputn', {'ypos'});

G = lqg(Pss,eye(3),1*eye(3),1e3*eye(1));

Gd = c2d(G,0.005,'tustin');
```



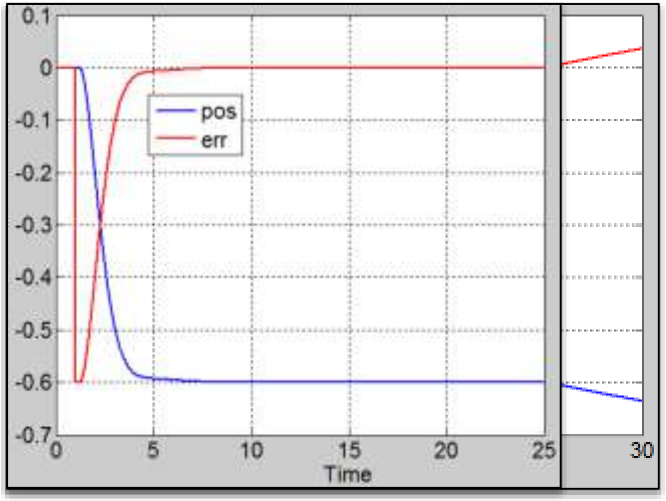


Control performance

- Works well when the slider is in motion
- At low velocity the linearized model is off
  - Continuous time control works well
  - Discrete time (sample time = 0.005) … not so much



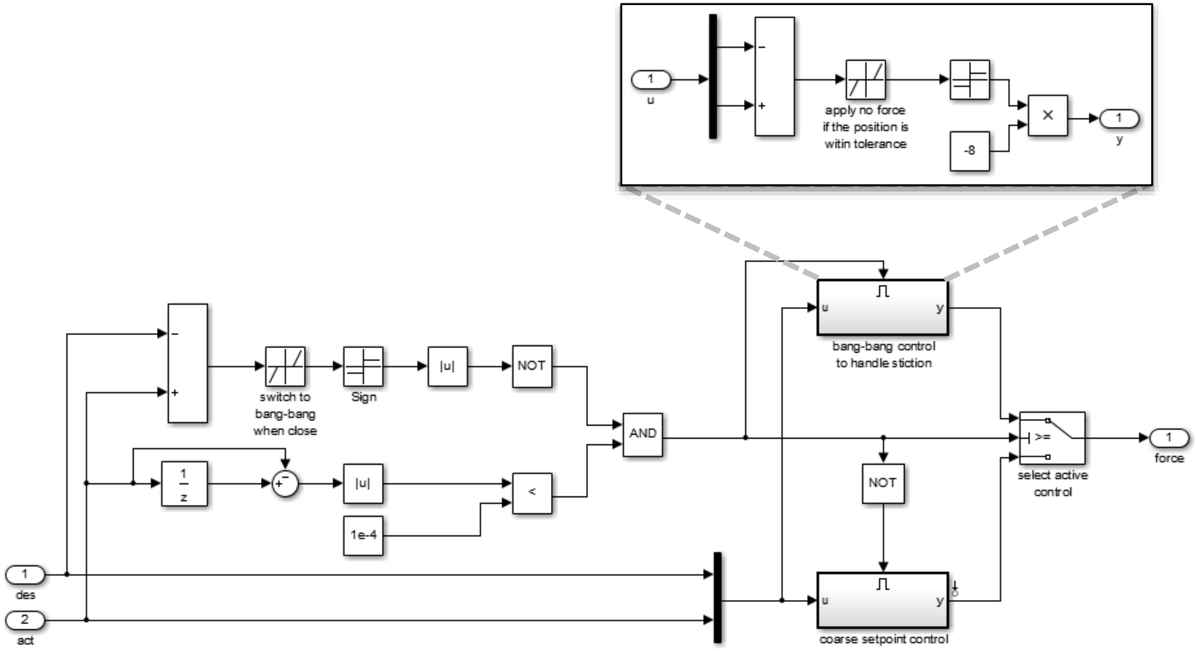
Continuous-time control

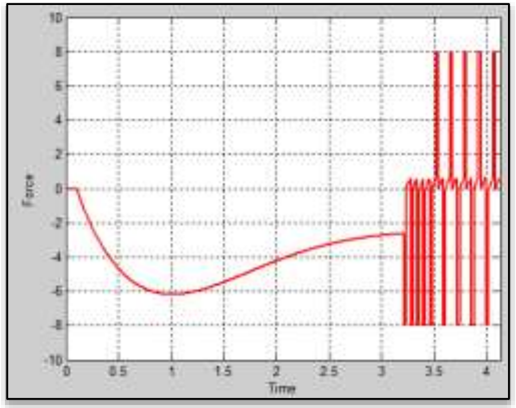


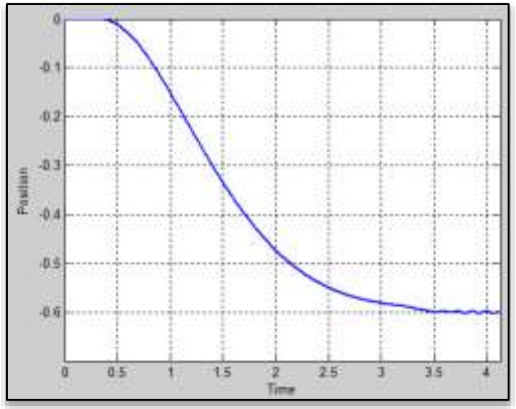
Discrete-time control

Mode-switching control

- Coarse setpoint control using a Gaussian controller
- Fine tune control using ‘bang/bang control’

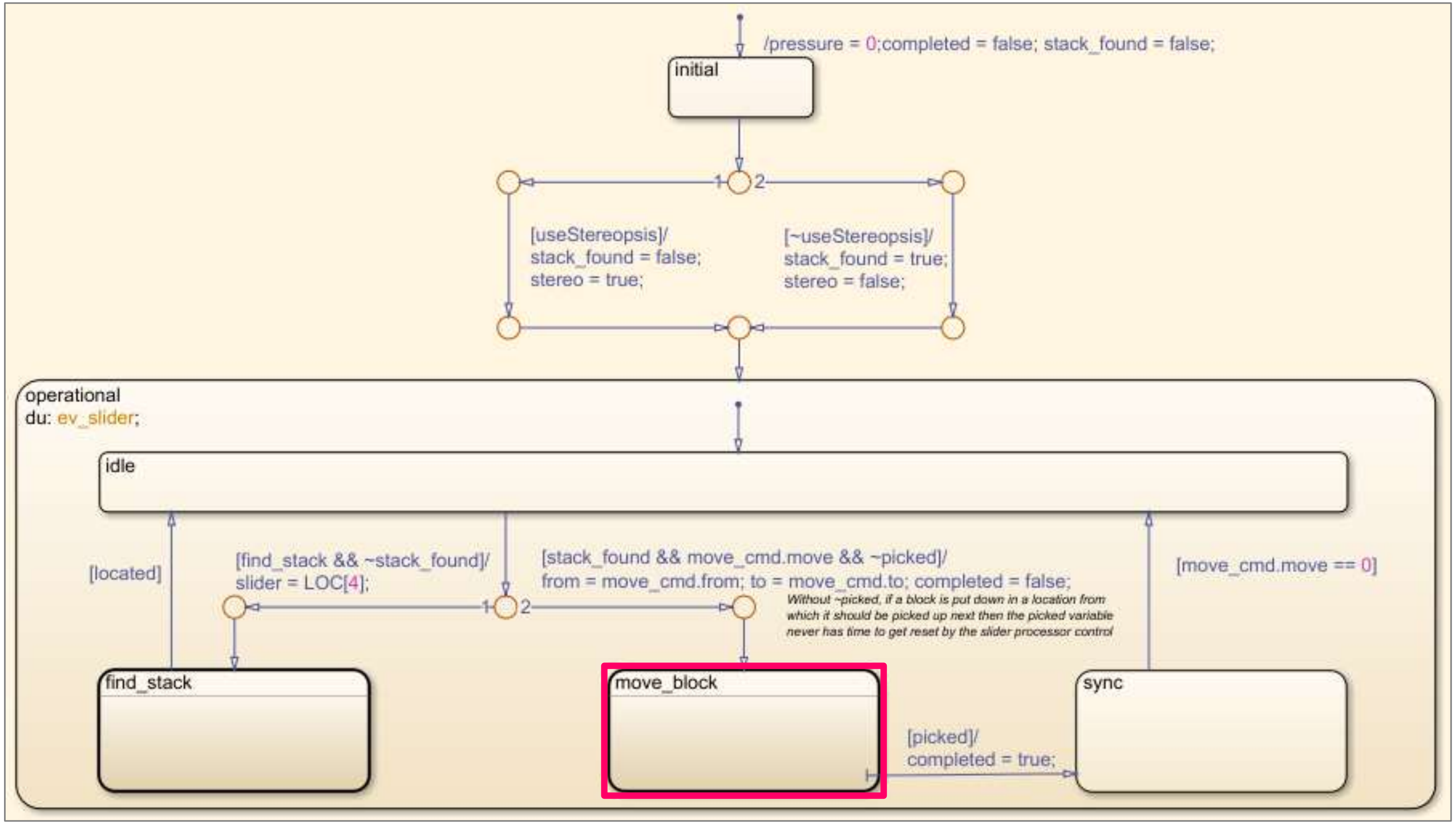




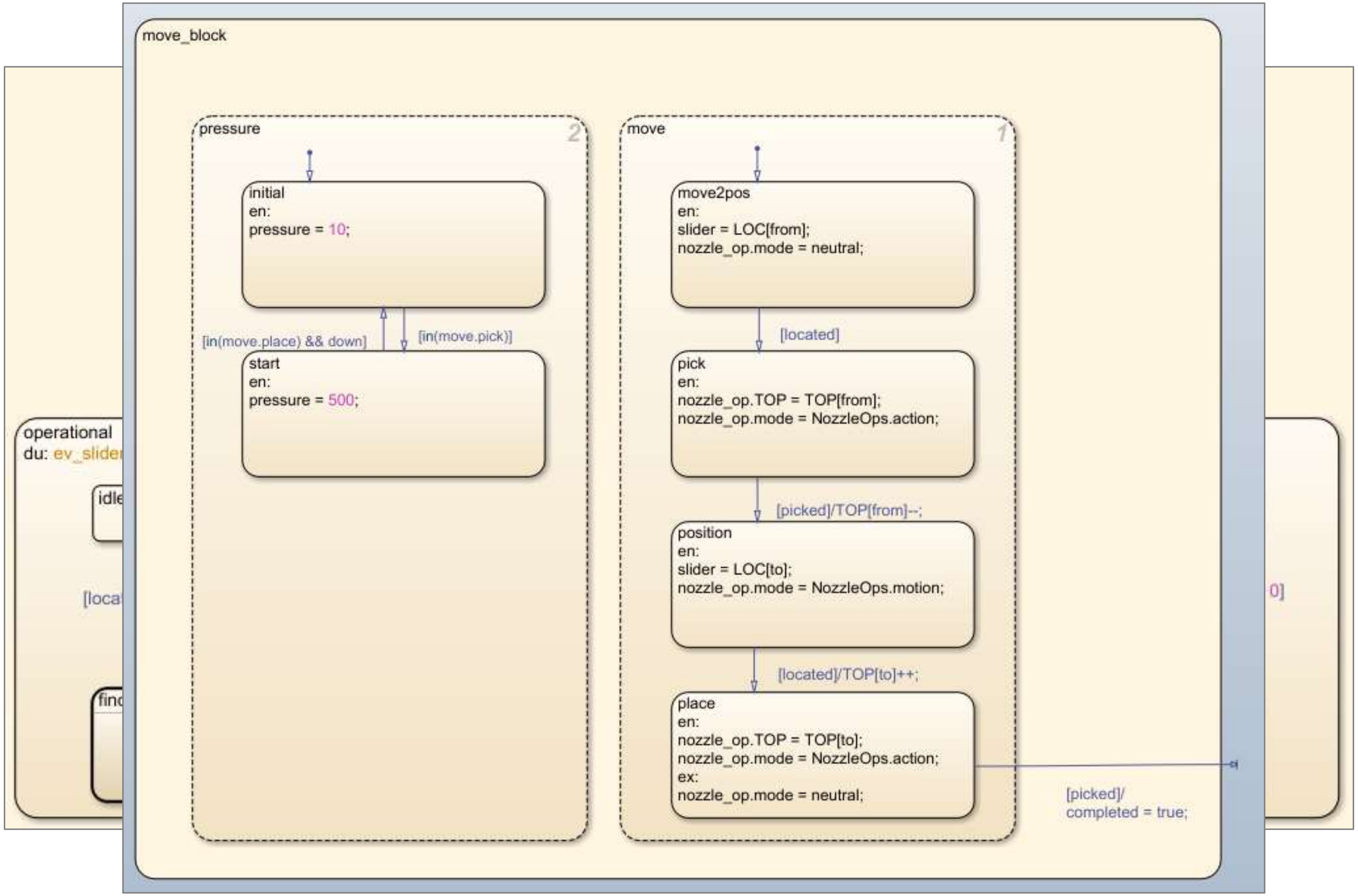


Supervisory and sequence control of operation

Hierarchical state machine with concurrency

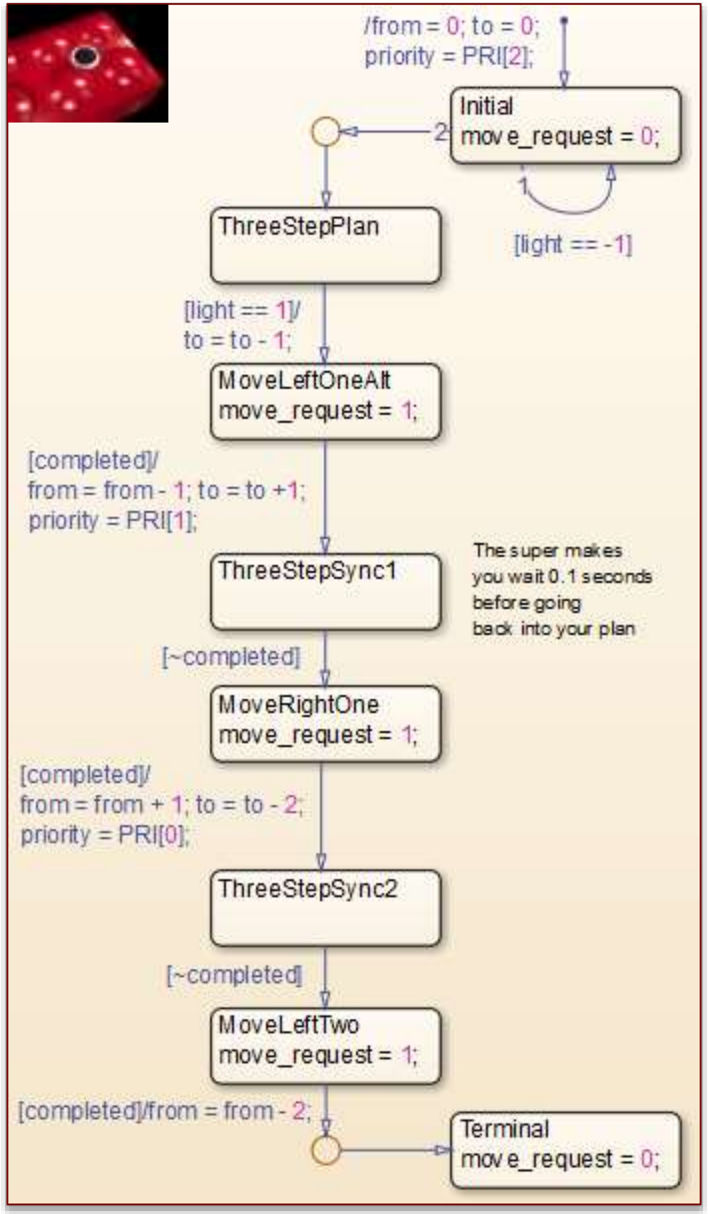


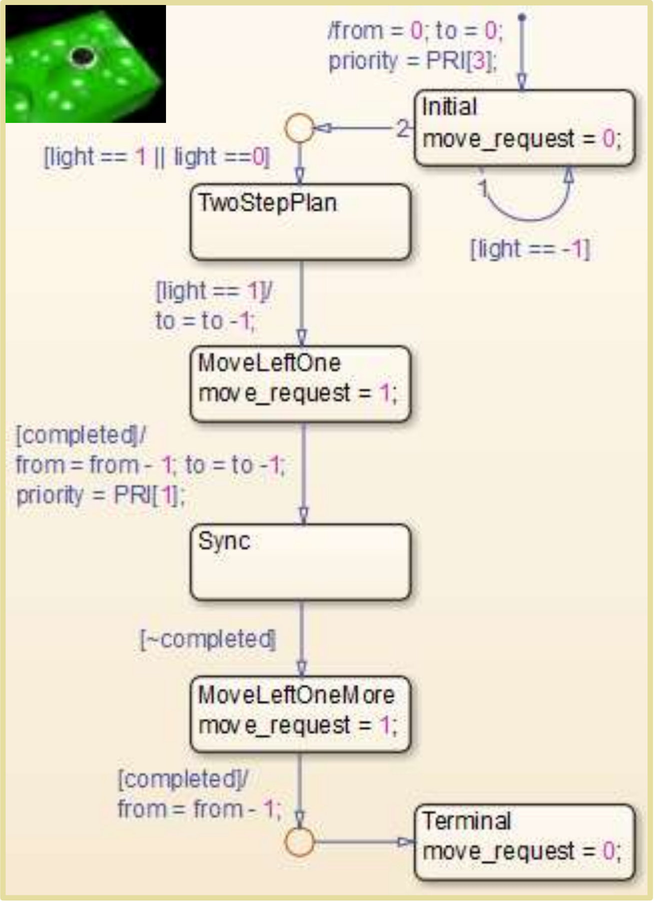
Hierarchical state machine with concurrency

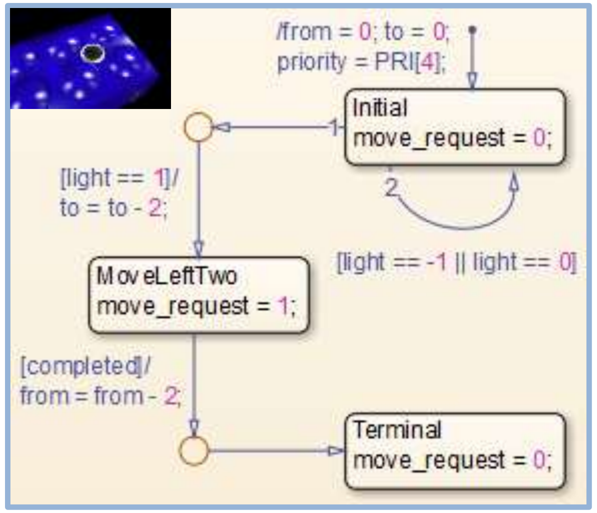


Distributed control

Block plans as state transition diagrams (nonoptimized)

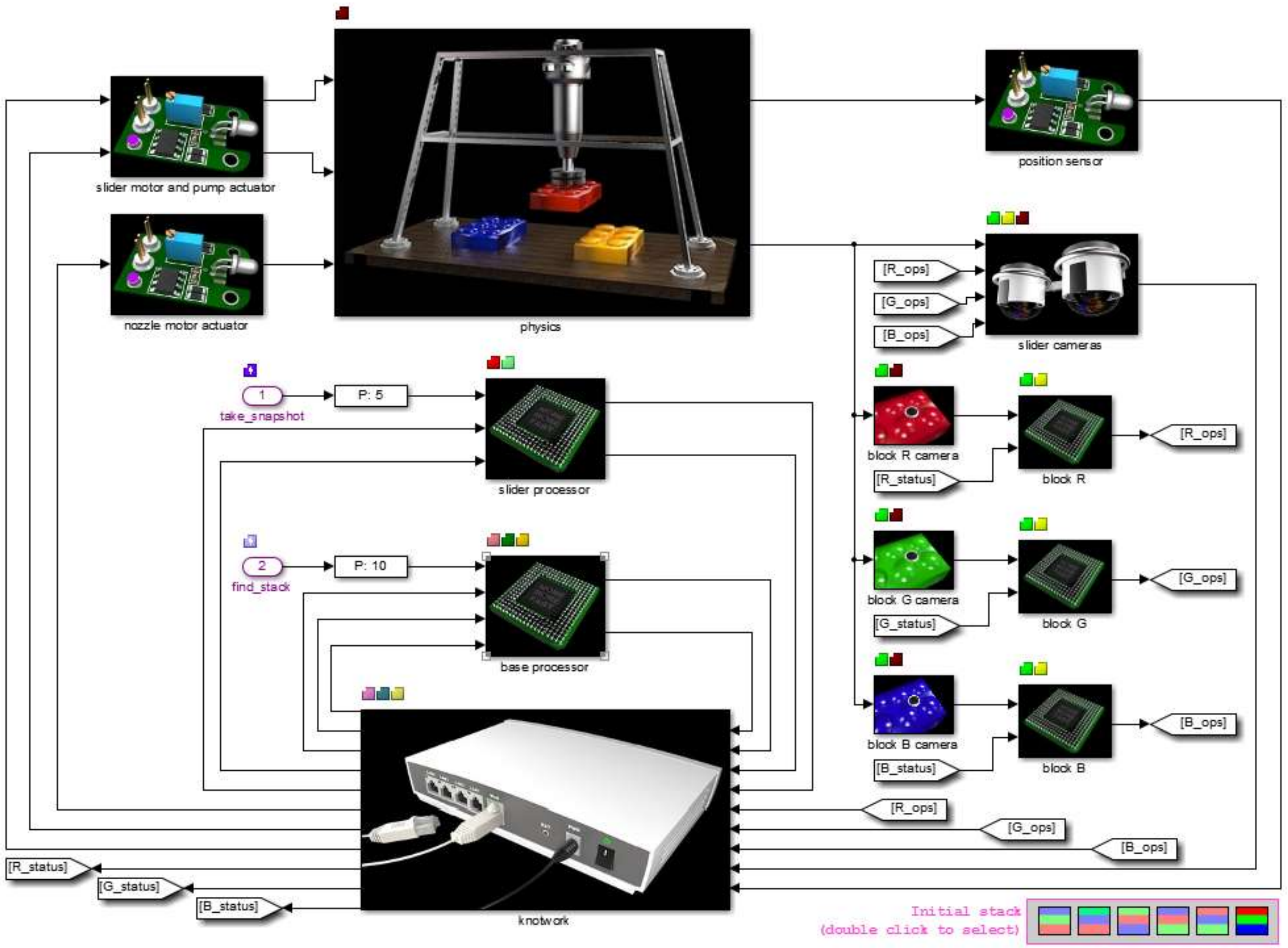




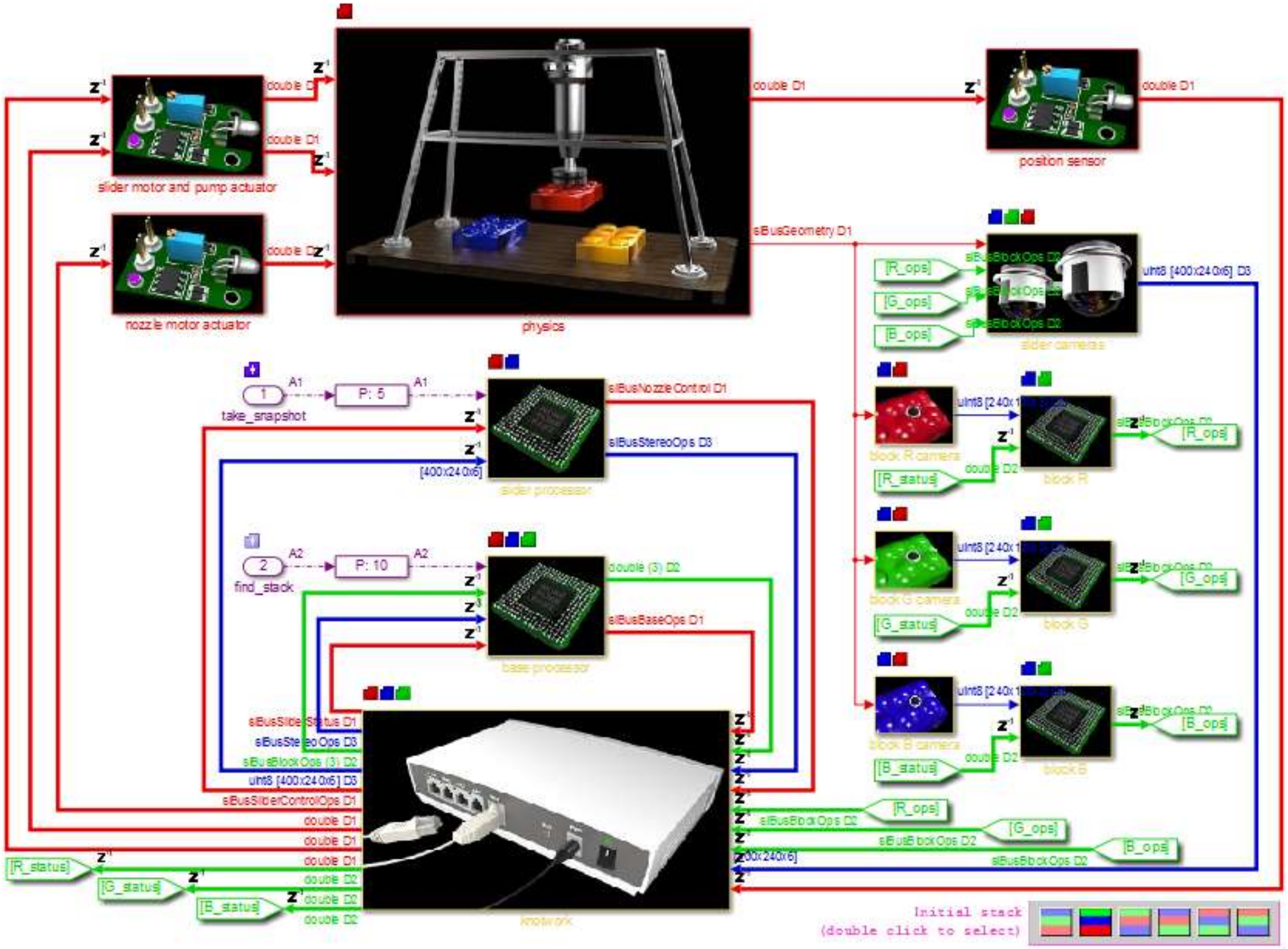


Putting it together

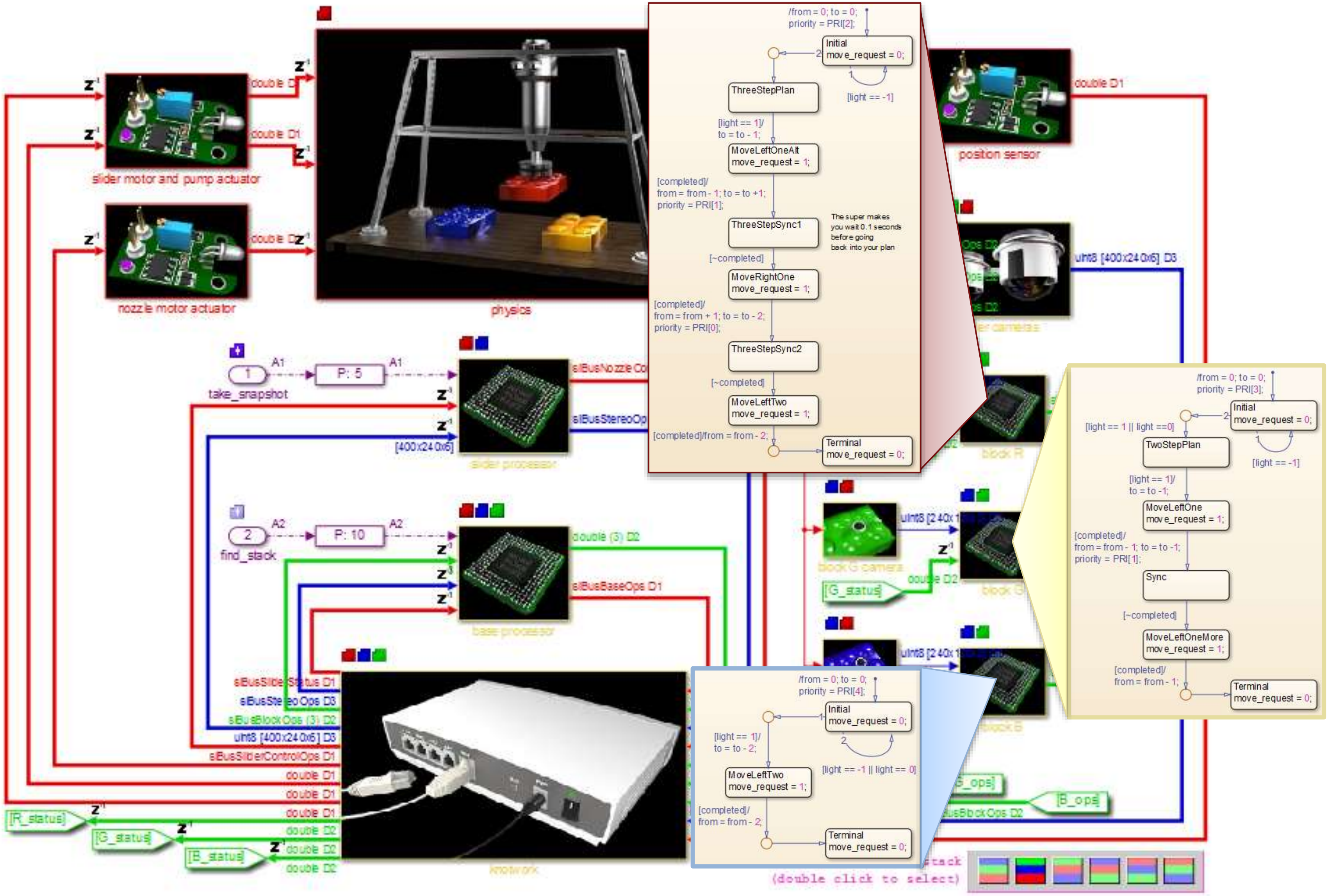
The distributed Towers of Hanoi



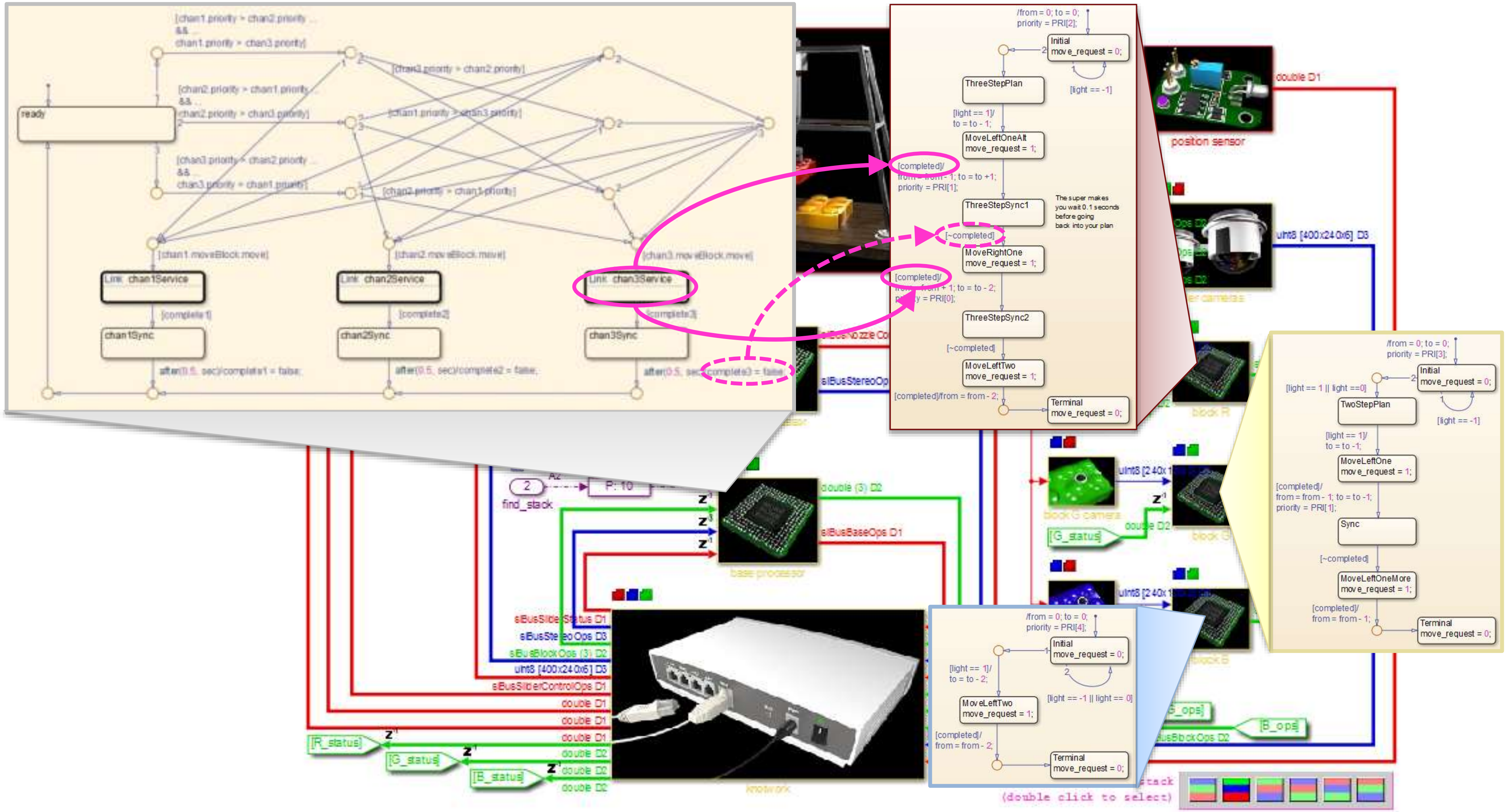
The distributed Towers of Hanoi



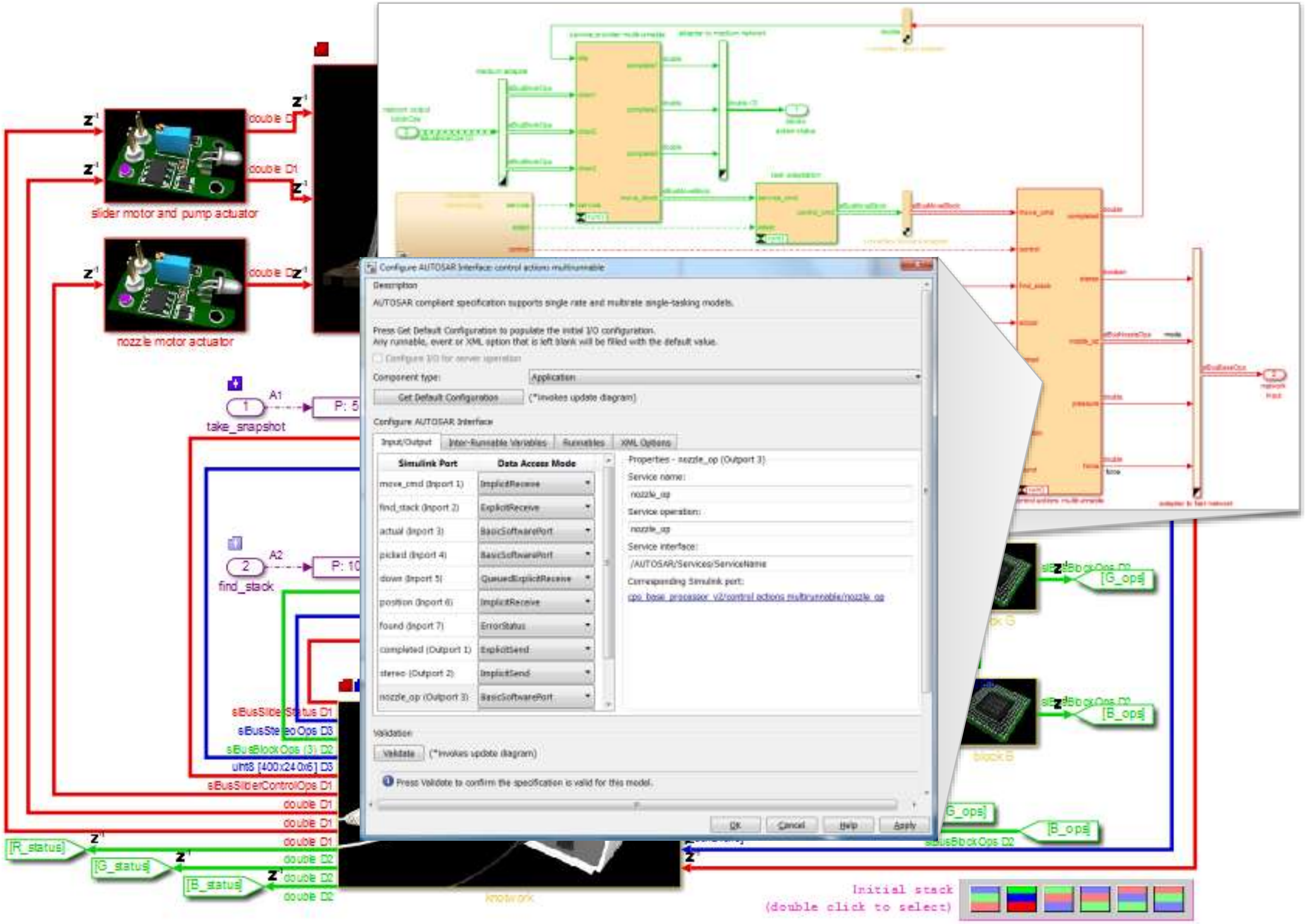
Open in a horizontal sense



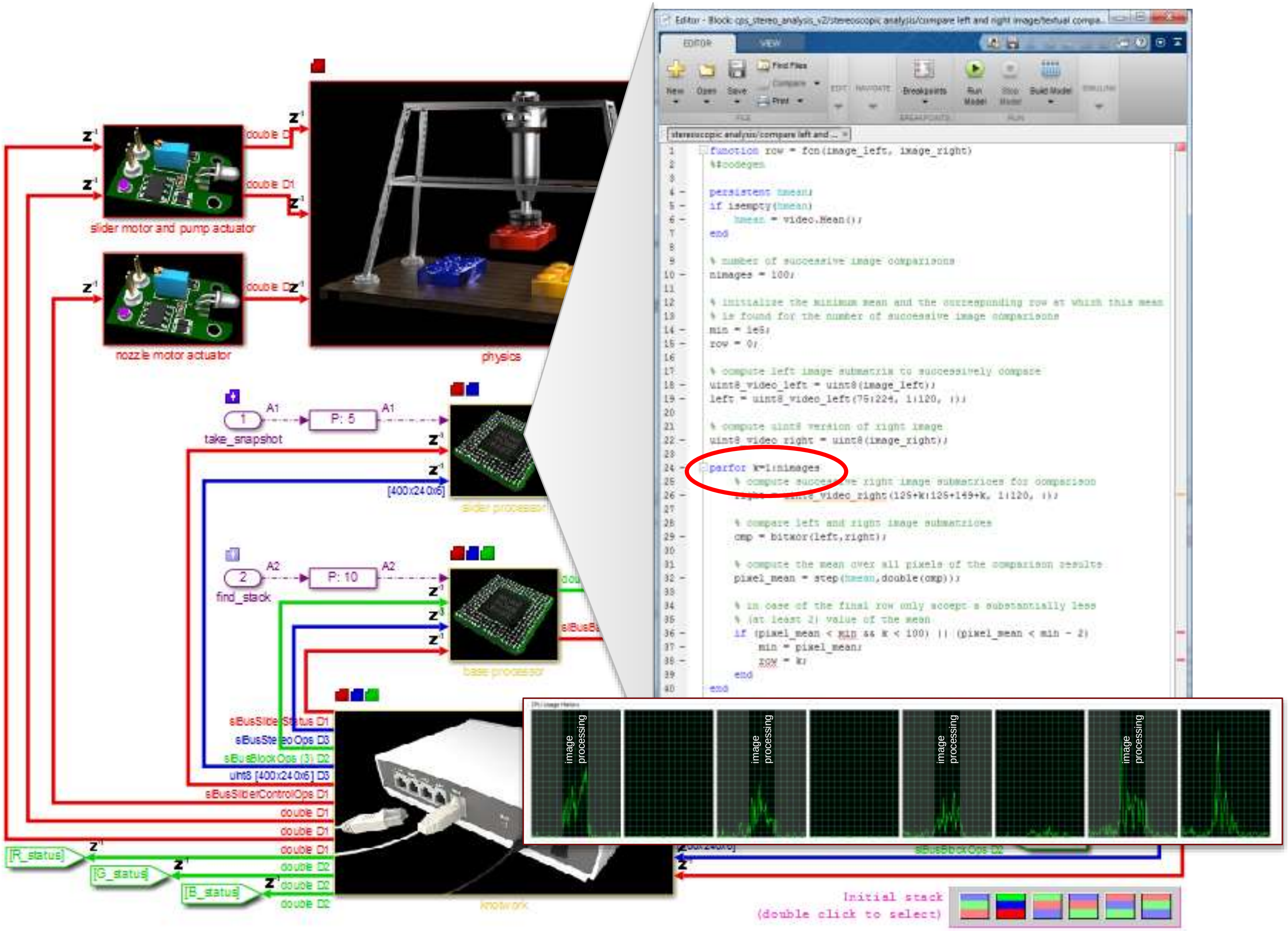
Open in a horizontal sense



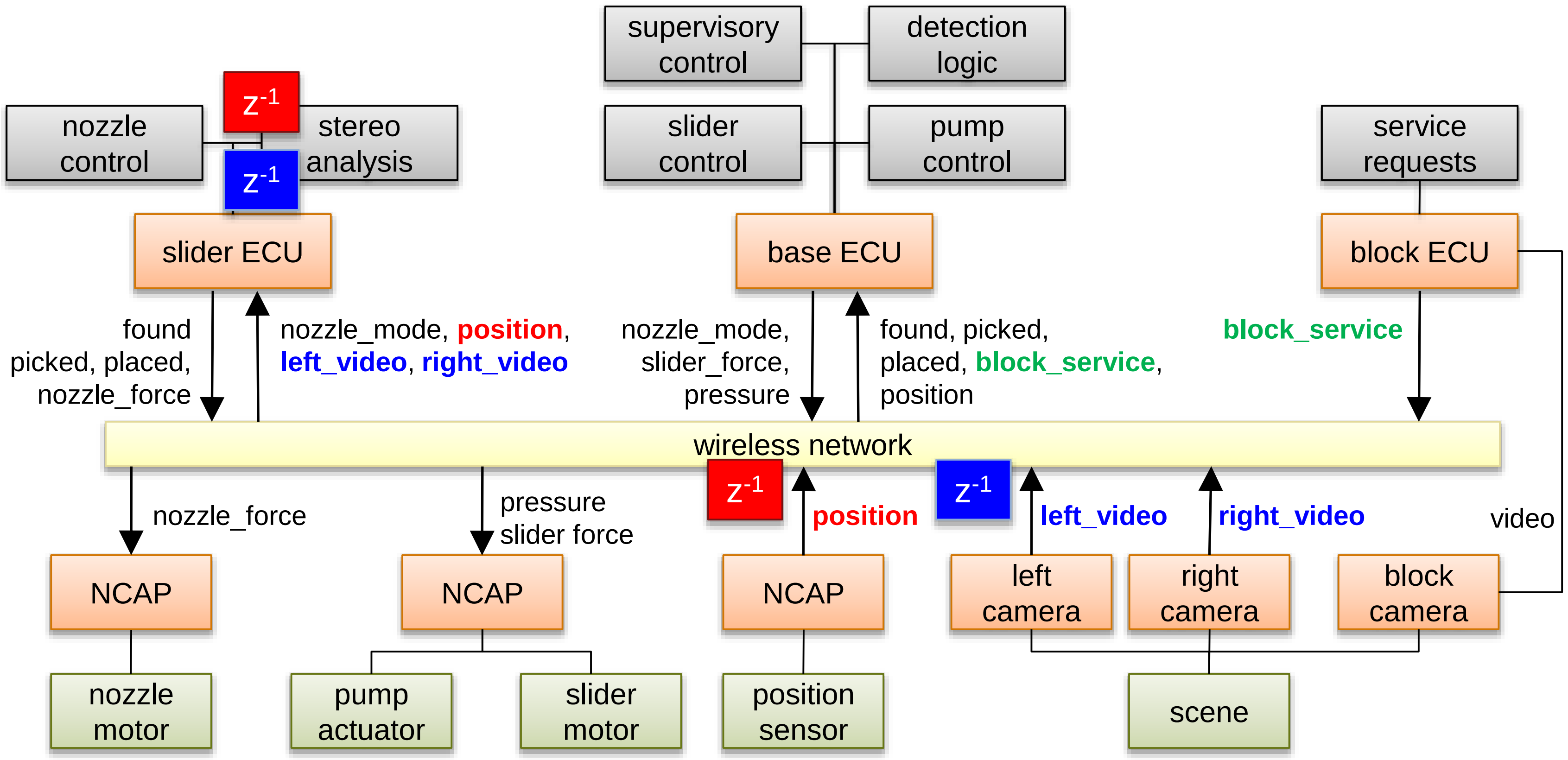
Open in vertical sense



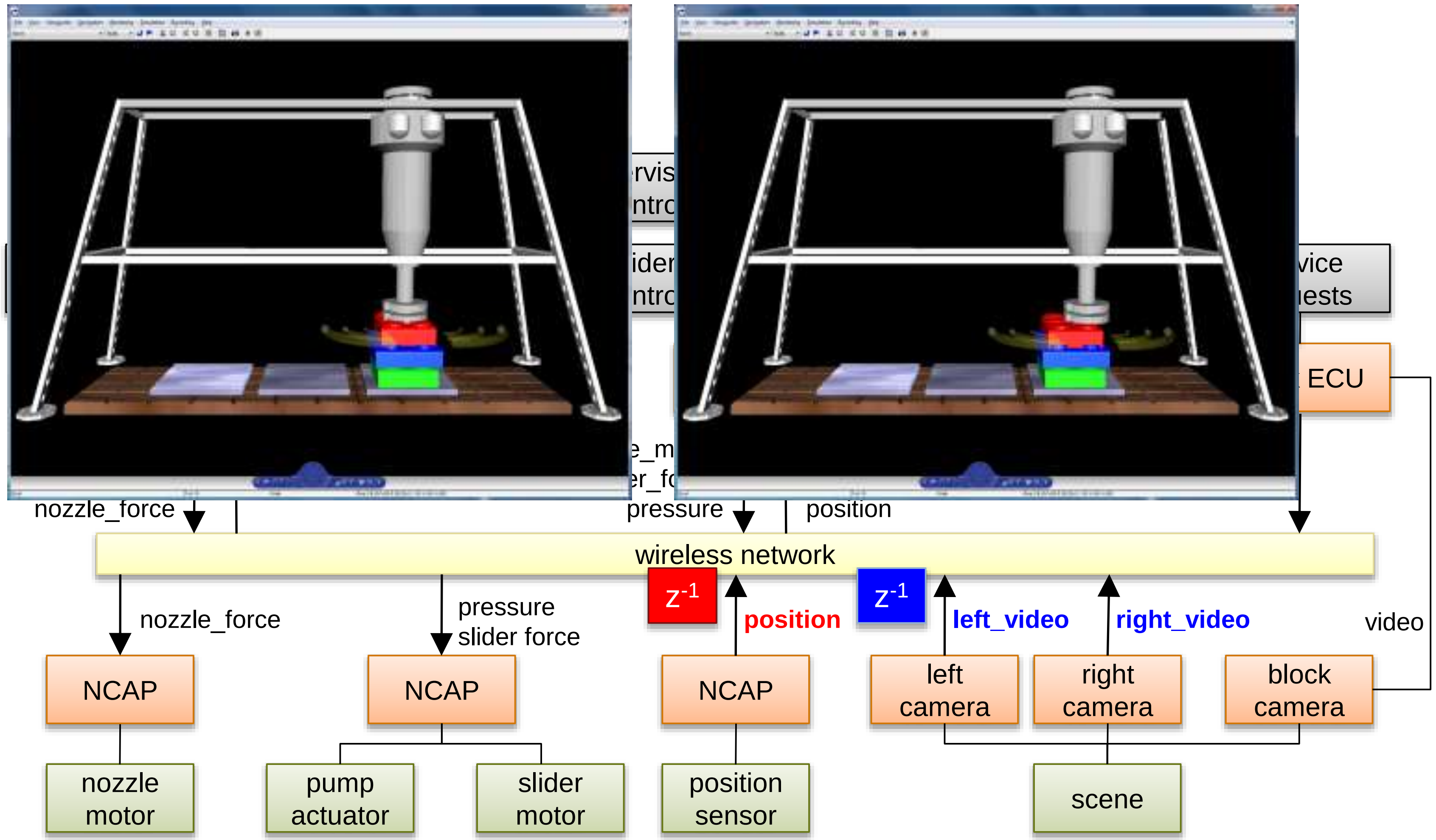
Open in vertical sense



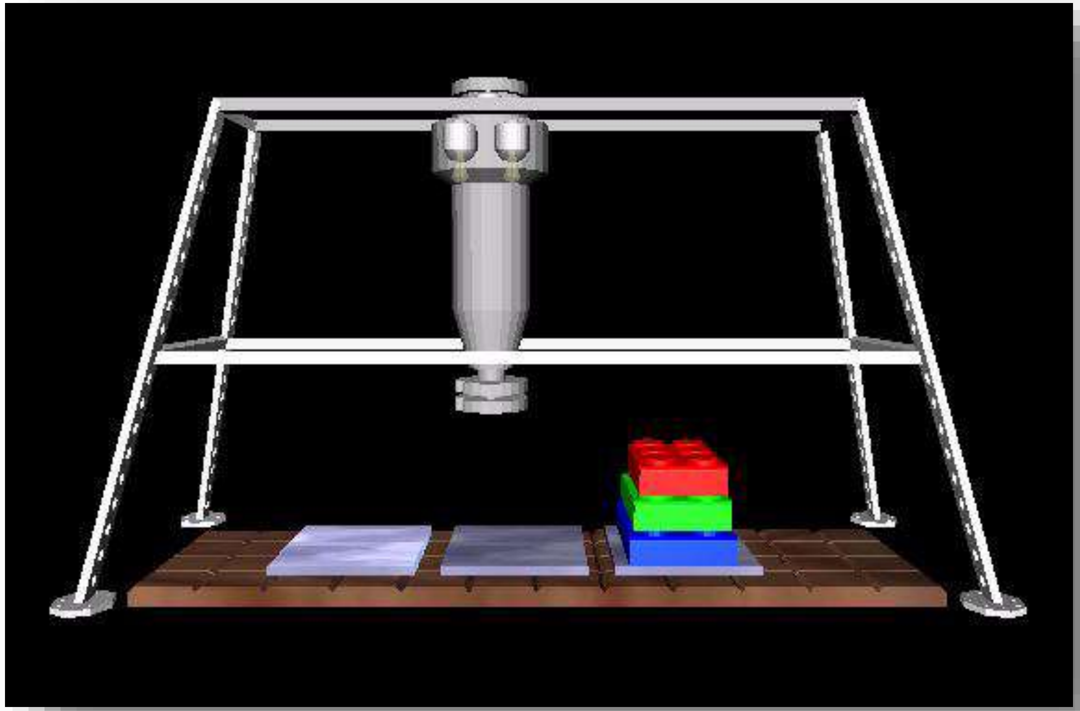
A multirate distributed architecture

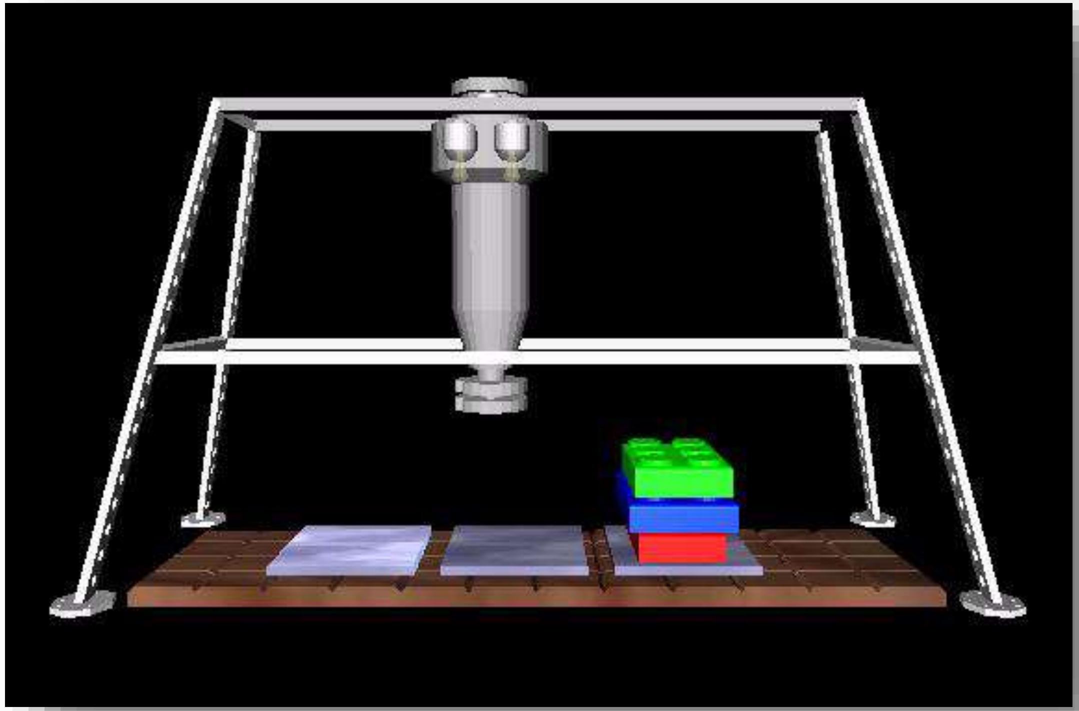


A multirate distributed architecture



Emerging behavior





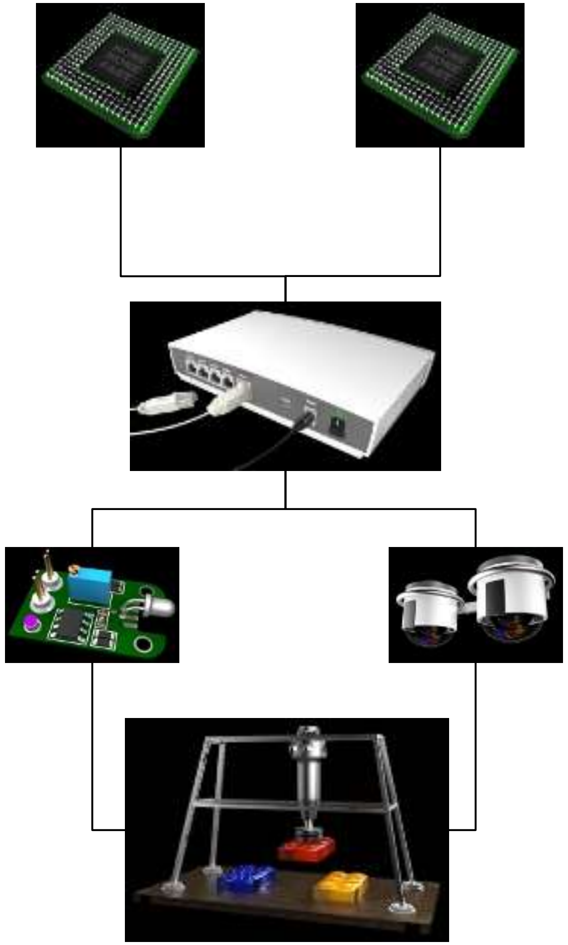
System characterization

A requirements perspective

The Towers of Hanoi revisited

Multiformalism

A broad range of modeling paradigms



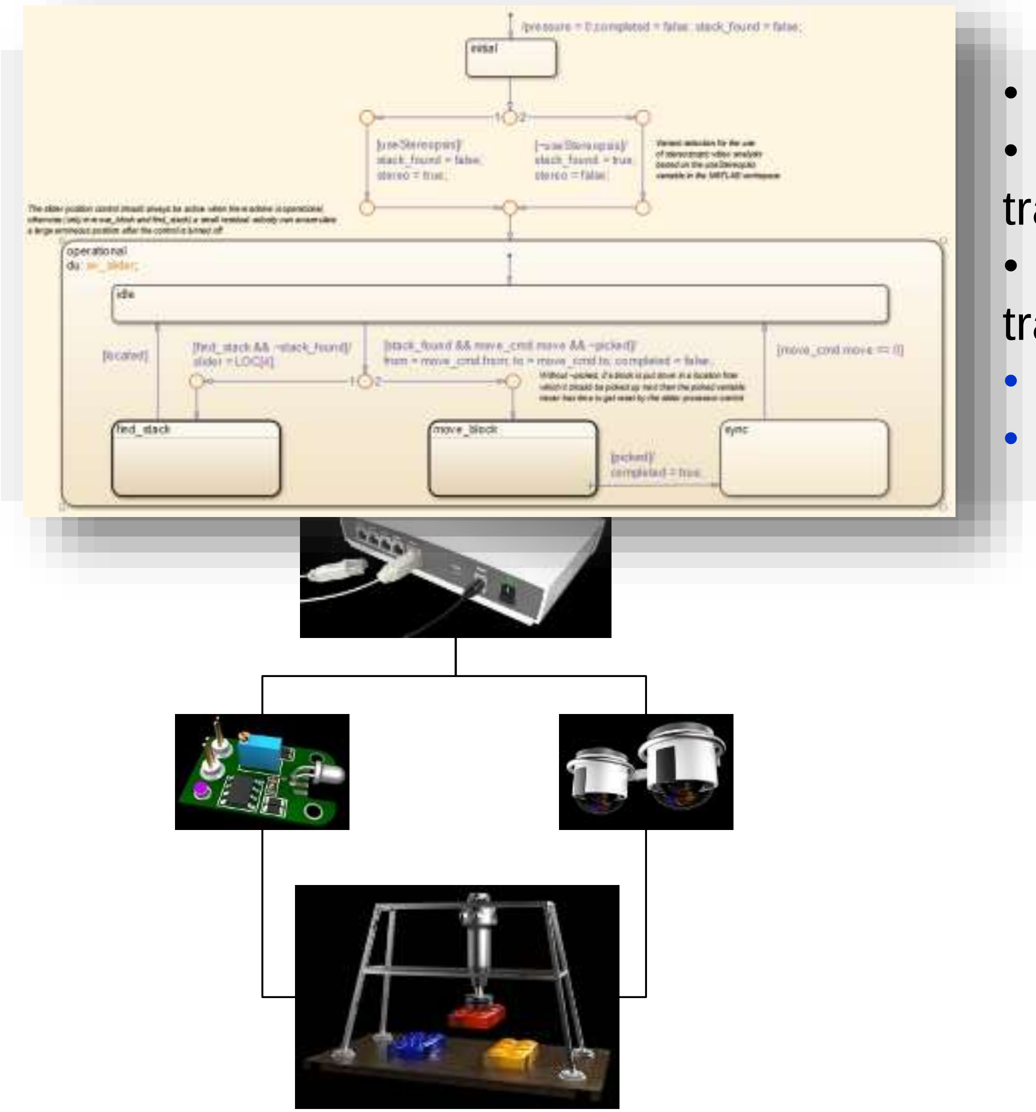
- Signal processing
- Control
  - Supervisory and sequence control
  - Feedforward and feedback control
  - Switched control
- Network, communication
- Physics, plant

Modeling

```
function row = fcn(image_left, image_right)
%#codegen

persistent hmean;
if isempty(hmean)
    hmean = video.Mean();
end

% number of successive image comparisons
nimages = 100;

% initialize the minimum mean and the corresponding row at which this mean
% is found for the number of successive image comparisons
min = 1e5;
row = 0;

% compute left image submatrix to successively compare
uint8_video_left = uint8(image_left);
left = uint8_video_left(75:224, 1:120, :);

% compute uint8 version of right image
uint8_video_right = uint8(image_right);

parfor k=1:nimages
    % compute successive right image submatrices for comparison
    right = uint8_video_right(125+k:125+149+k, 1:120, :);

    % compare left and right image submatrices
    cmp = bitxor(left,right);

    % compute the mean over all pixels of the comparison results
    pixel_mean = step(hmean,double(cmp));

    % in case of the final row only accept a substantially less
    % (at least 2) value of the mean
    if (pixel_mean < min && k < 100) || (pixel_mean < min - 2)
        min = pixel_mean;
        row = k;
    end
end

% to be consistent with negative offset in graphical version
row = - row;
```

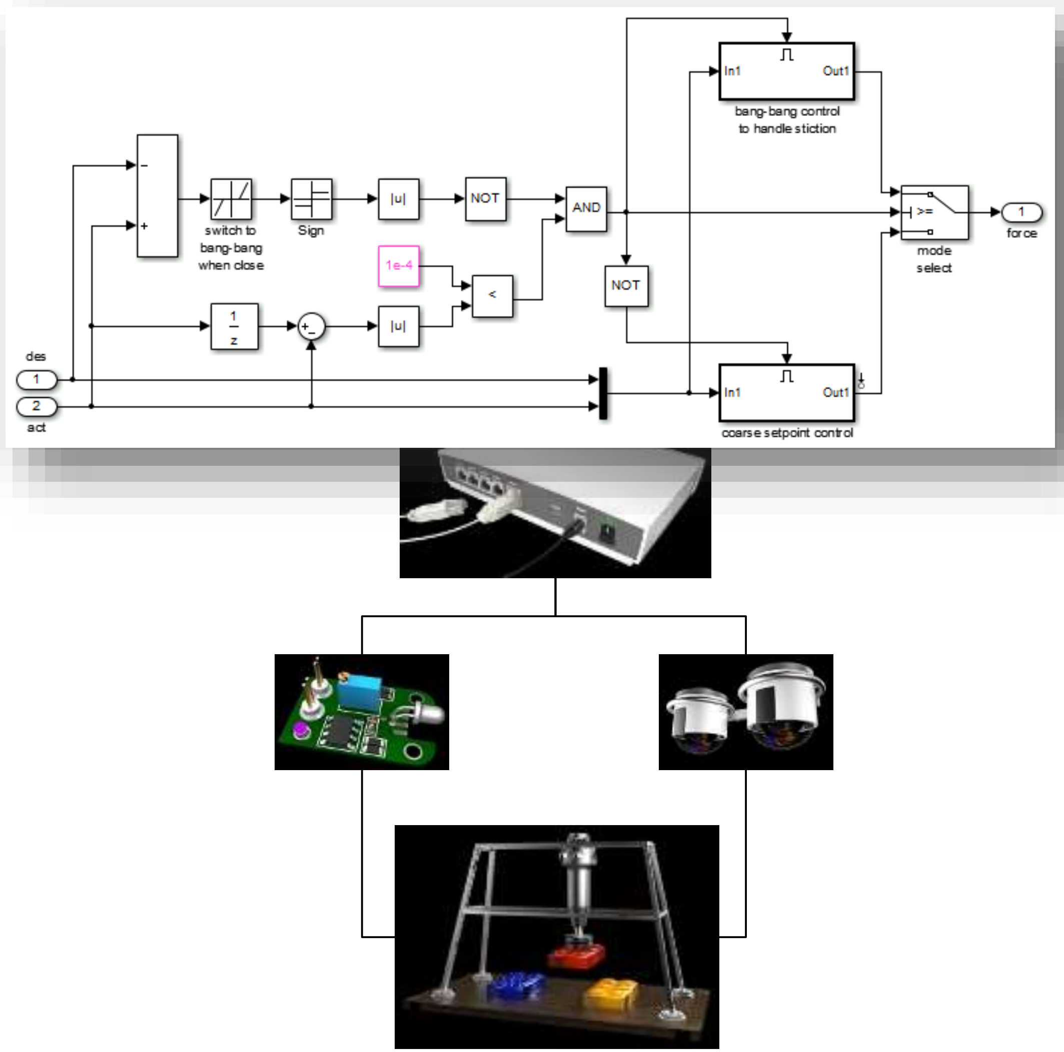
- Algorithmic
- Assignments
  - Destructive state access
- Untimed
- Data centric

Modeling the supervisory and sequence control



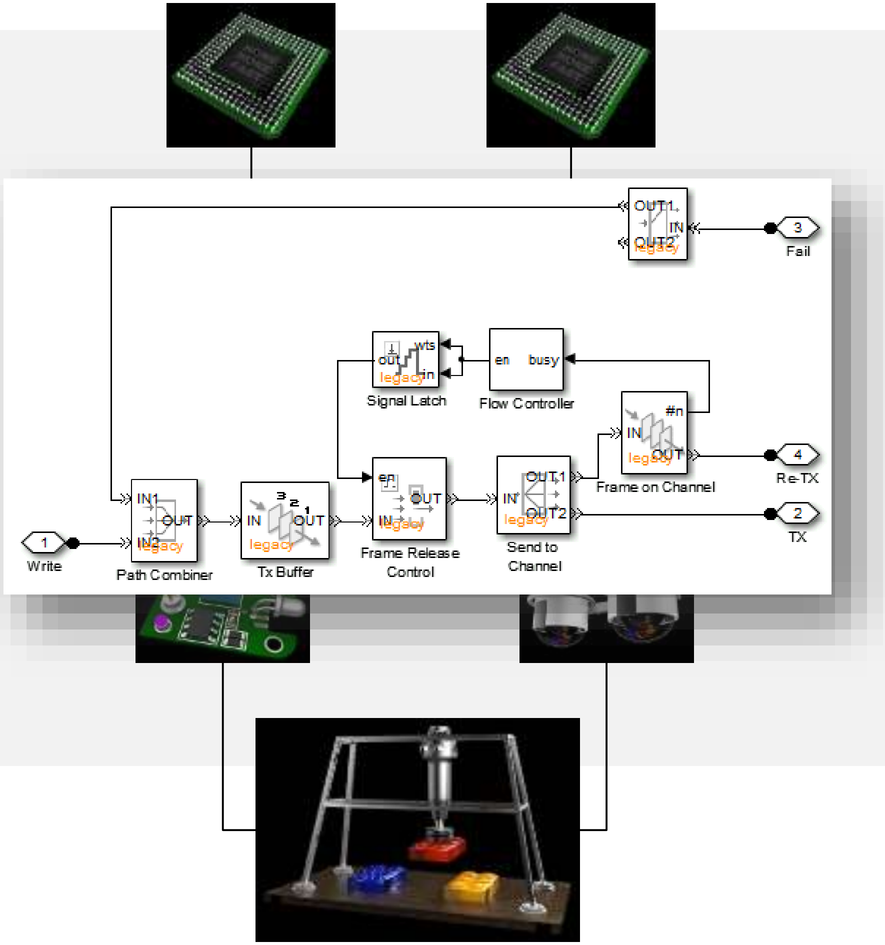
- Discrete state based
- Discrete events cause transitions between states
- Conditions to guard the transition
- Untimed
- Control centric

Modeling the feedback control



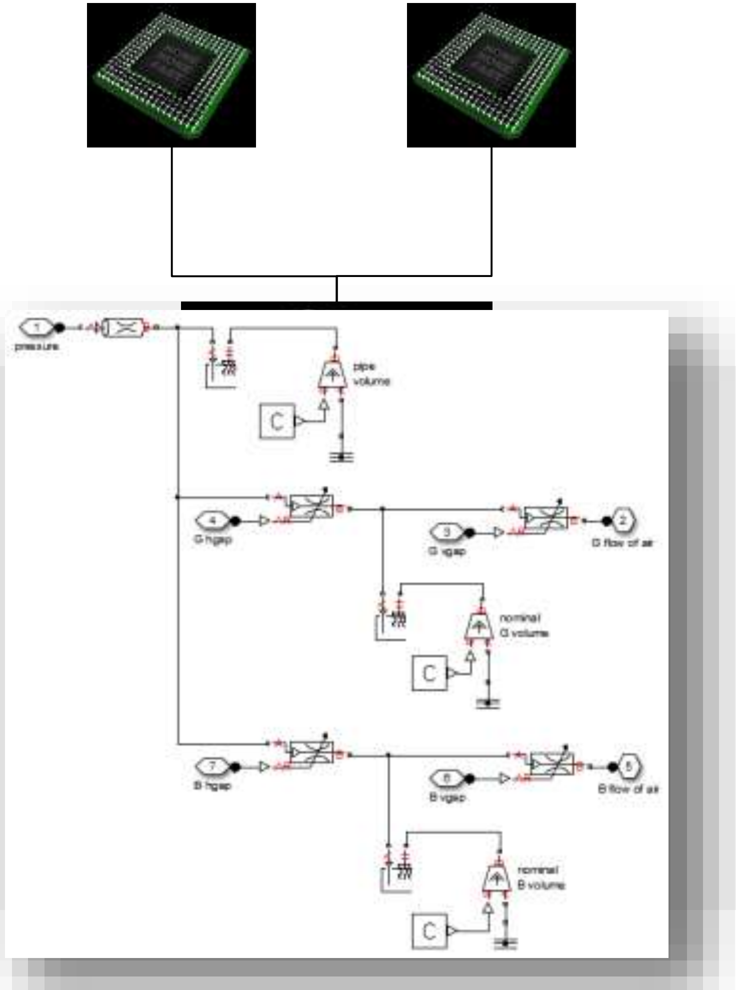
- Sampled discrete time
- Fixed sample time
- Periodic
- Data centric

Modeling network traffic



- Entity flow through a graph
- Attributes
  - Source
  - Destination
  - service time
  - Priority
  - …
- Discrete events
- Preemption
- Data centric
- Aperiodic
- Often stochastic

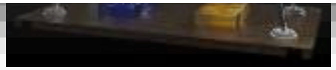
Modeling the plant physics

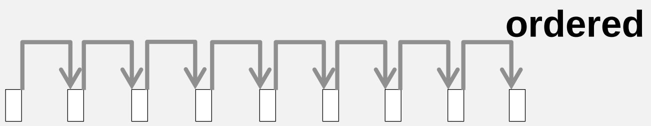


- Domain-specific modeling—Simscape
  - Electrical
  - Pneumatic
  - Thermal
  - …
- Differential equation based
- Noncausal, energy-based, modeling

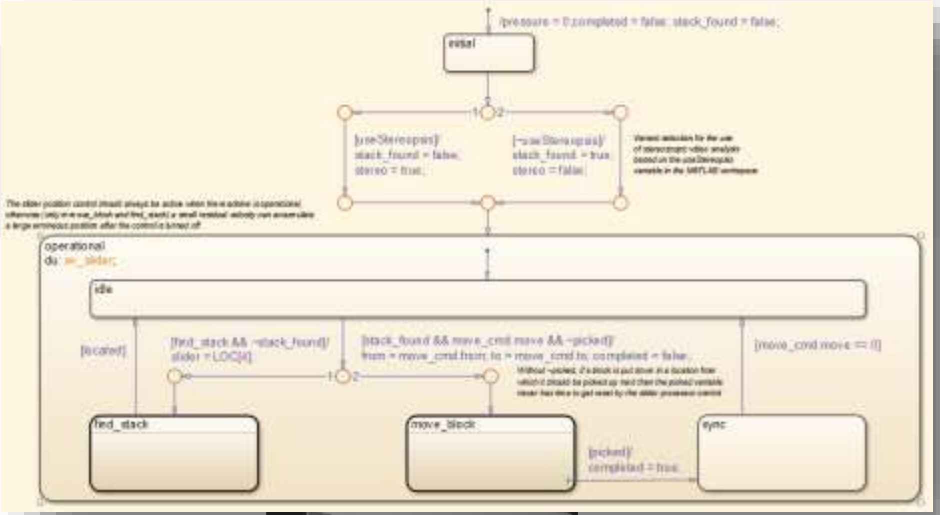
With a broad range of semantic domains

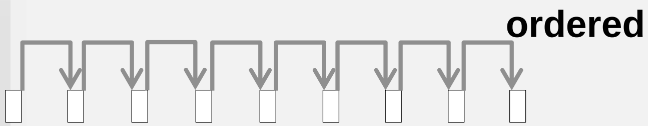
```
function row = fcn(image_left, image_right)
%#codegen

persistent hmean;
if isempty(hmean)
    hmean = video.Mean();
end

% number of successive image comparisons
nimages = 100;

% initialize the minimum mean and the corresponding row at which this mean
% is found for the number of successive image comparisons
min = 1e5;
row = 0;

% compute left image submatrix to successively compare
uint8_video_left = uint8(image_left);
left = uint8_video_left(75:224, 1:120, :);

% compute uint8 version of right image
uint8_video_right = uint8(image_right);

parfor k=1:nimages
    % compute successive right image submatrices for comparison
    right = uint8_video_right(125+k:125+149+k, 1:120, :);

    % compare left and right image submatrices
    cmp = bitxor(left,right);

    % compute the mean over all pixels of the comparison results
    pixel_mean = step(hmean,double(cmp));

    % in case of the final row only accept a substantially less
    % (at least 2) value of the mean
    if (pixel_mean < min && k < 100) || (pixel_mean < min - 2)
        min = pixel_mean;
        row = k;
    end
end

% to be consistent with negative offset in graphical version
row = - row;
```

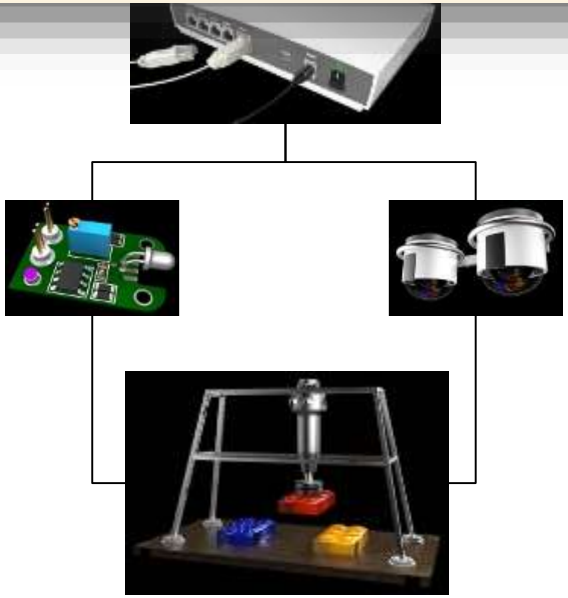




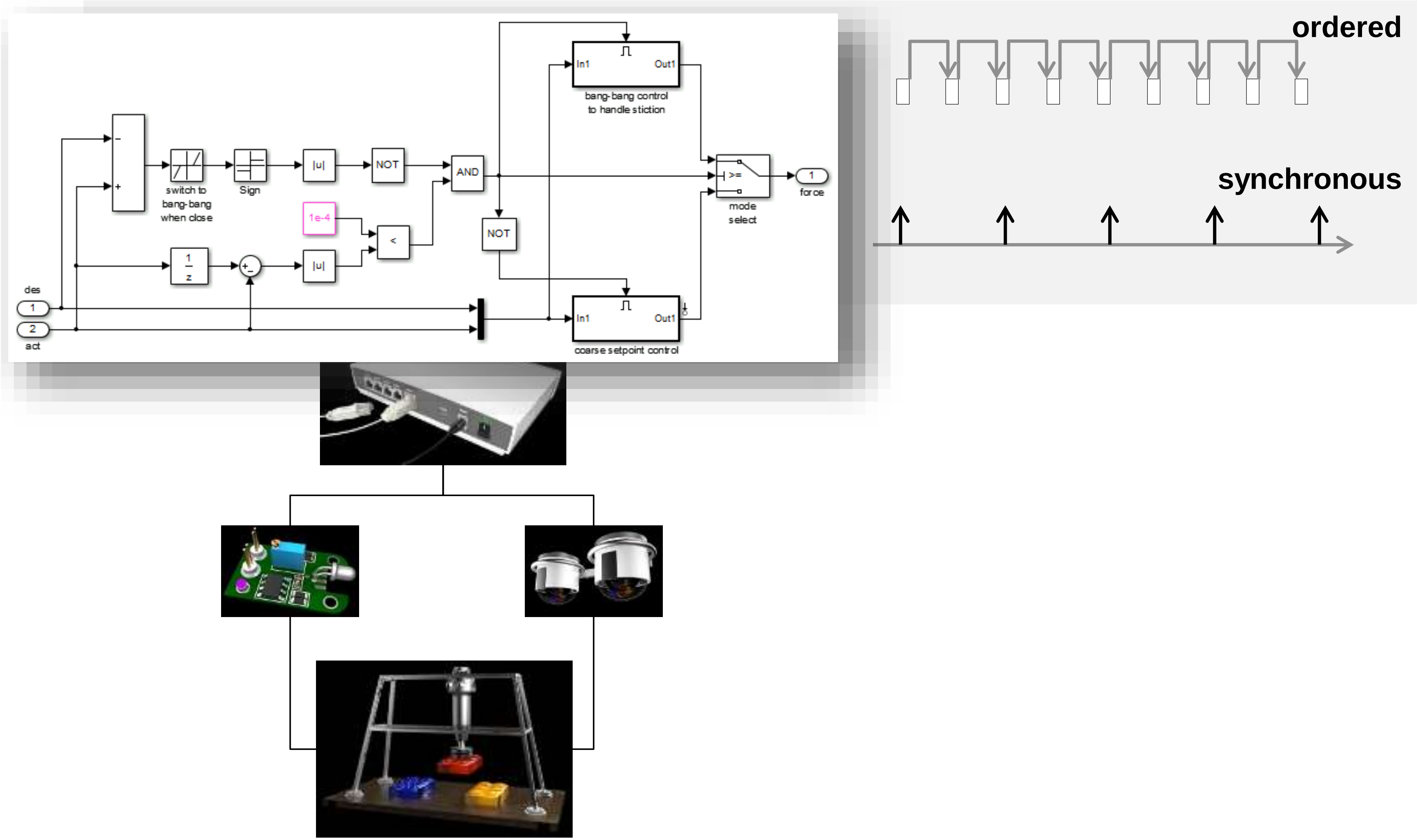
With a broad range of semantic domains



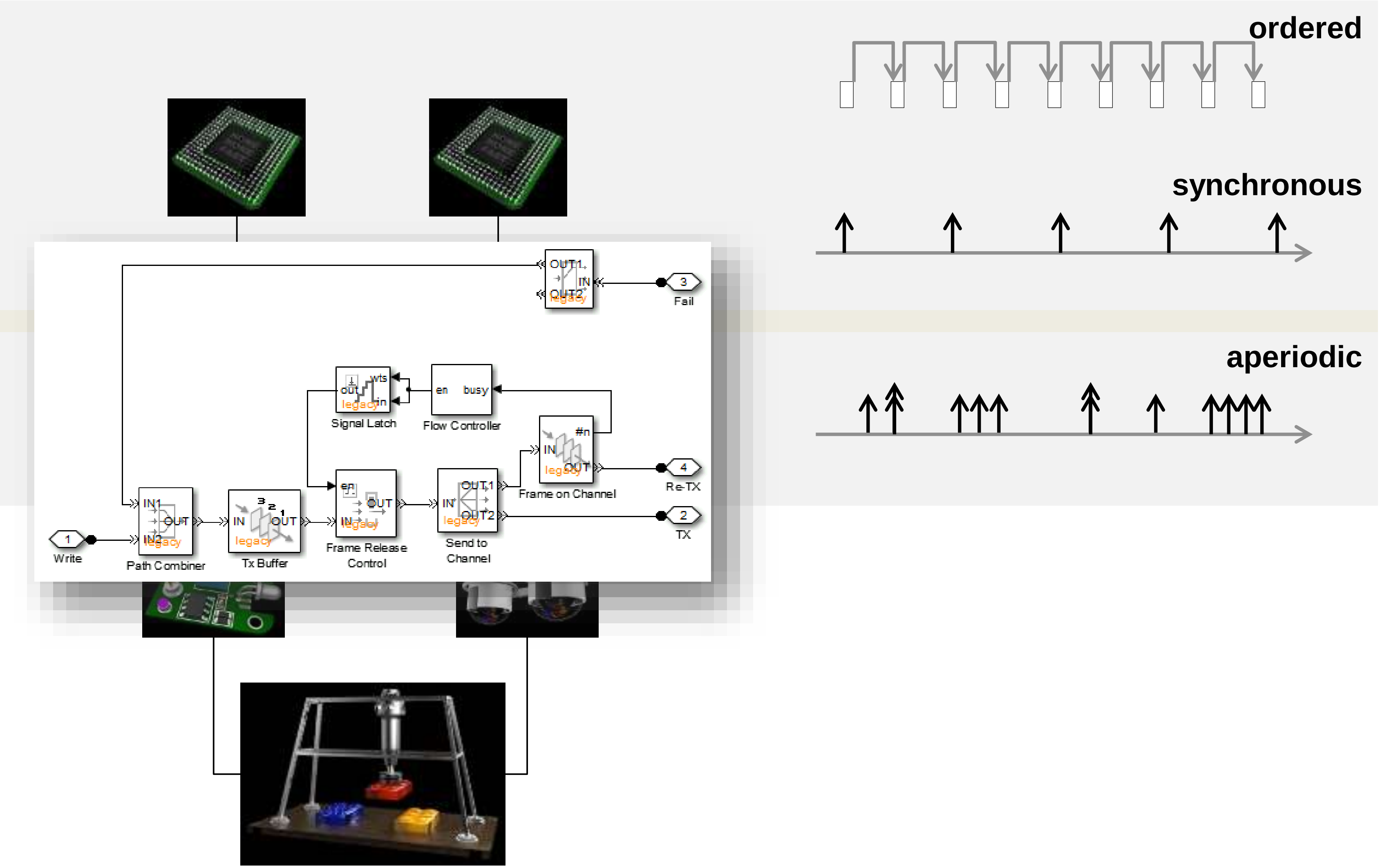




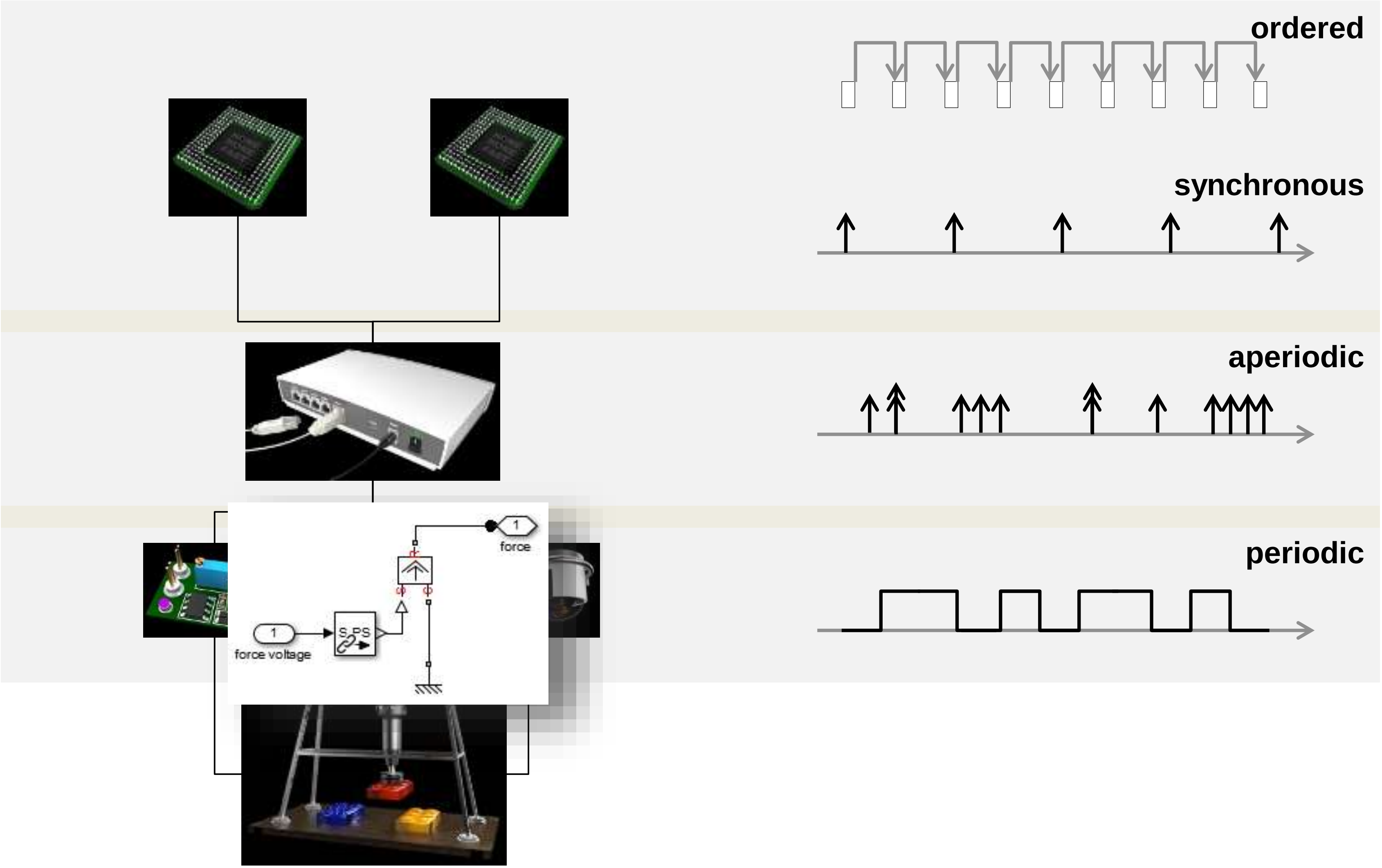
With a broad range of semantic domains



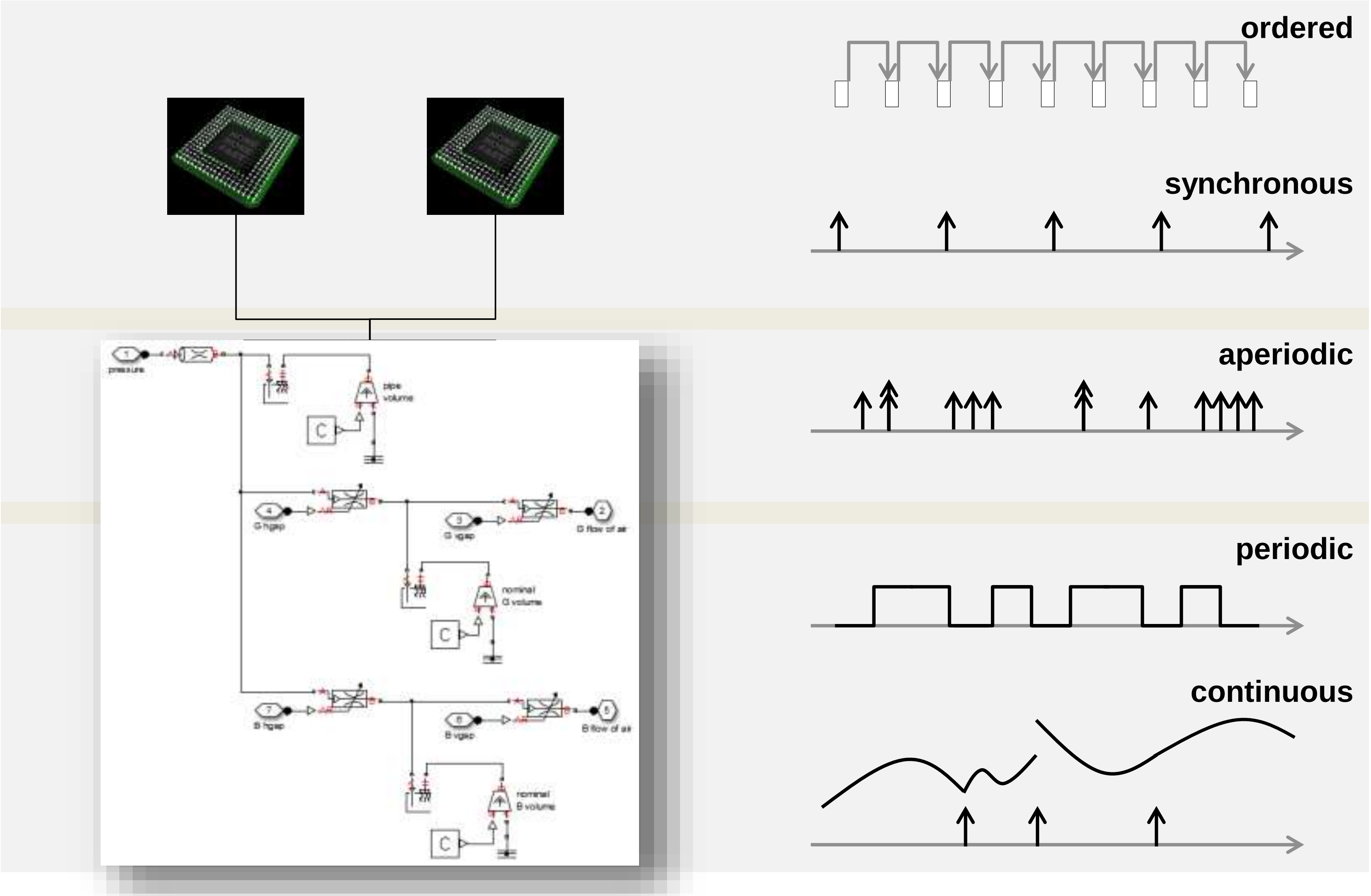
With a broad range of semantic domains



With a broad range of semantic domains

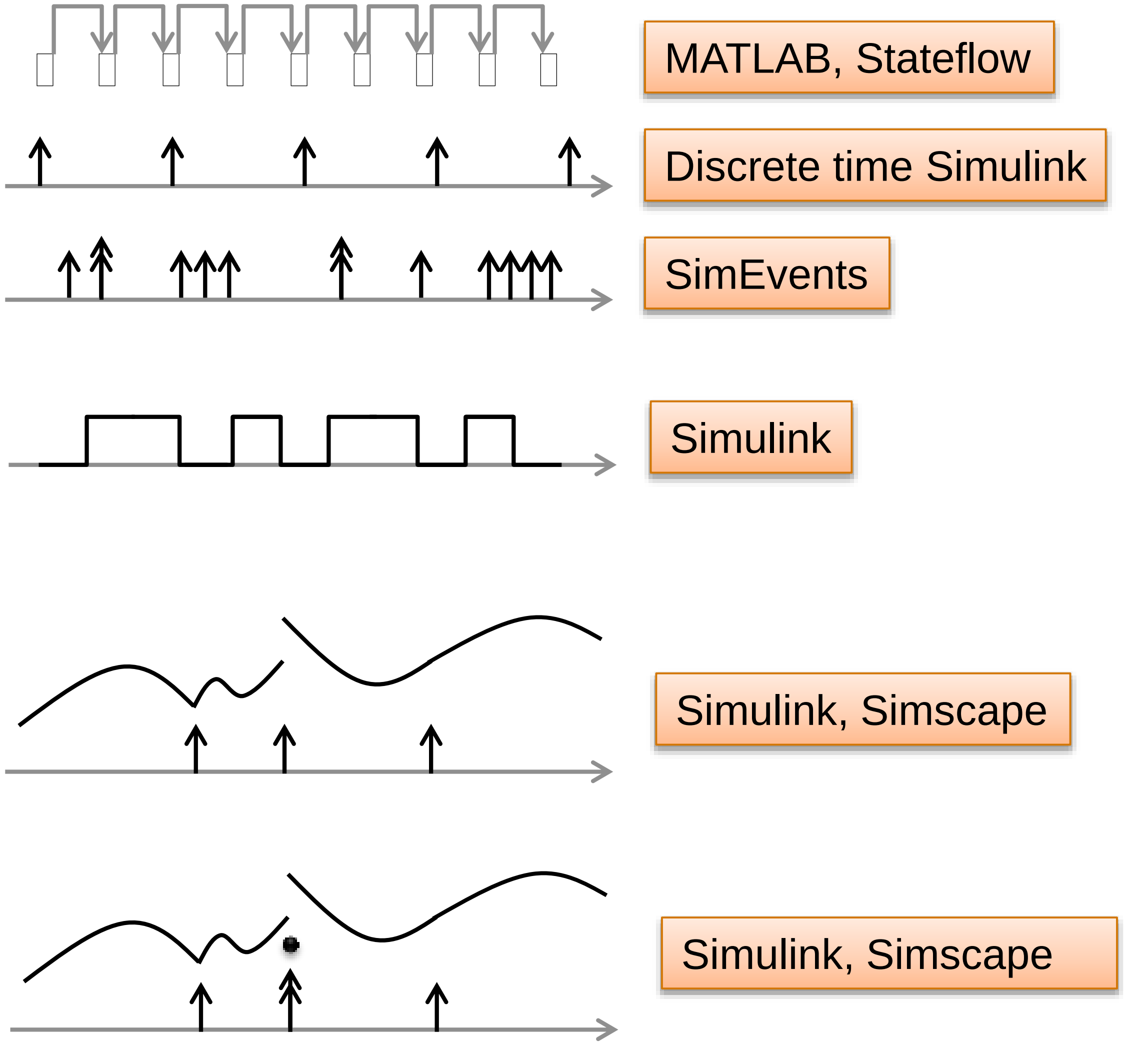


With a broad range of semantic domains



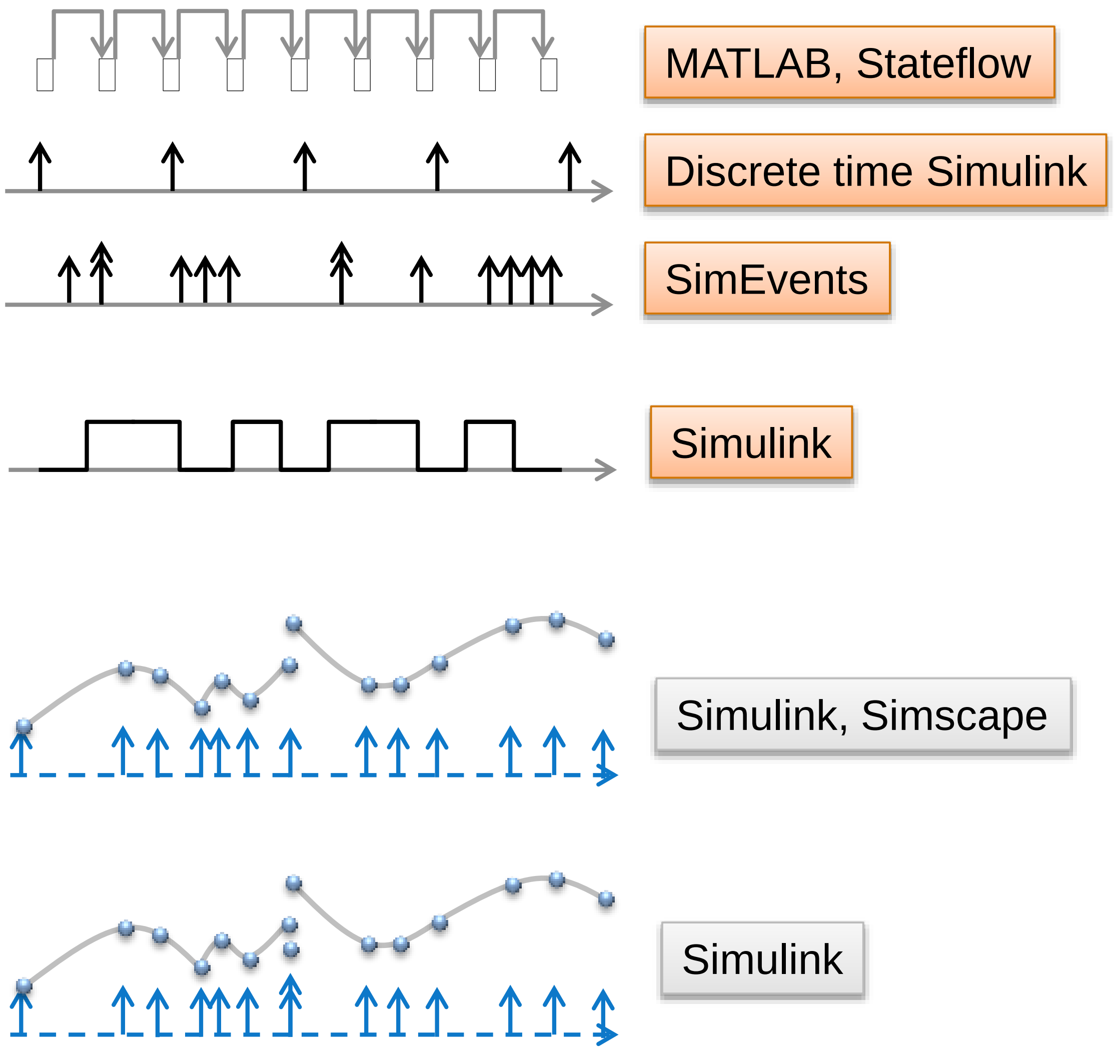
The semantic domain of a dynamic system

- Points, [ ]
  - On **N**
  - On **R** x **N**
- Intervals, [⟩ (⟨ ⟩, ⟨])
  - On **R**
- Hybrid point/interval
  - On **R**
  - On **R** x **N**



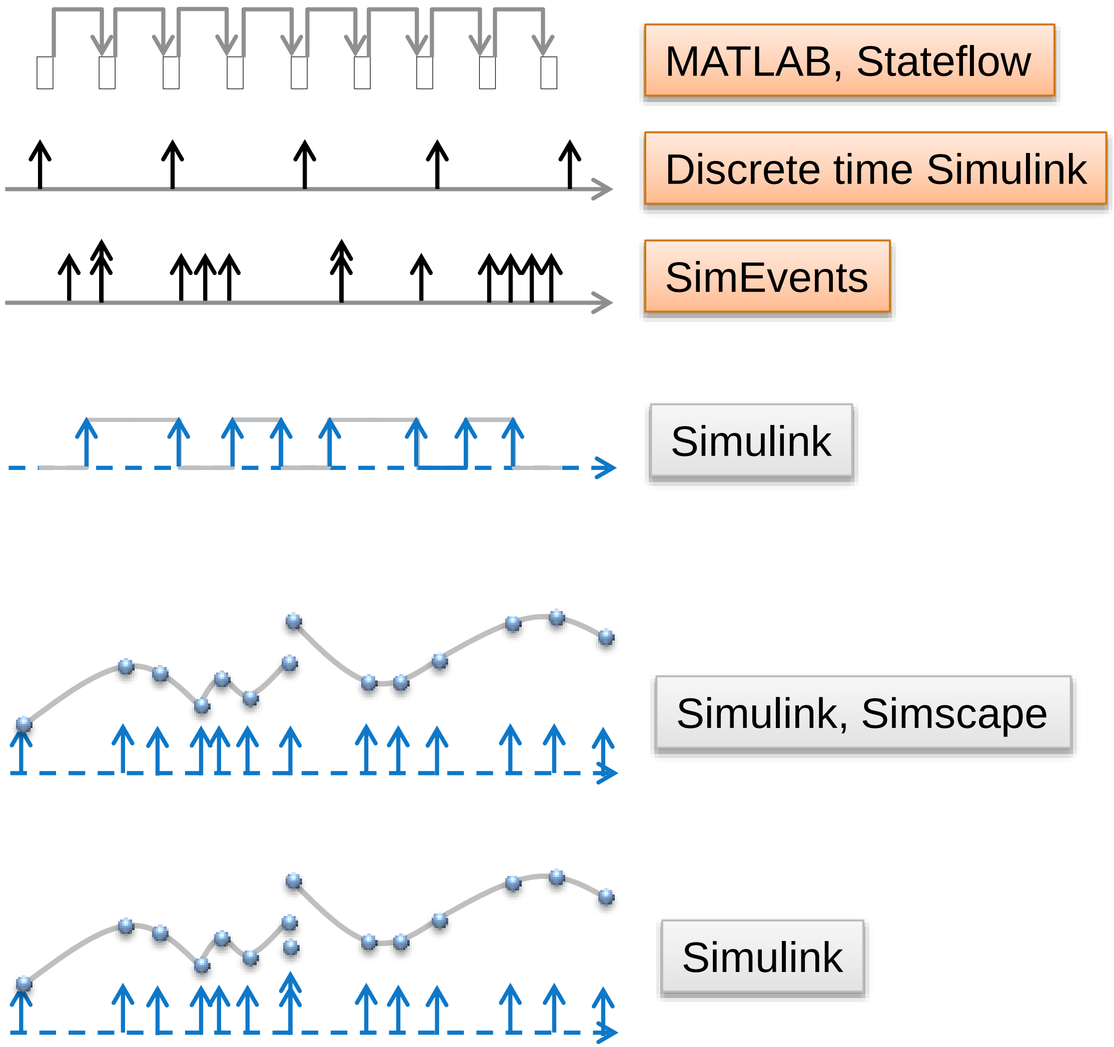
The semantic domain of a dynamic system

- Points, [ ]
  – On **N**
  – On **R** x **N**
- Intervals, [〉 (〈 〉, 〈])
  – On **R**
- Hybrid point/interval
  – On **R**
  – On **R** x **N**



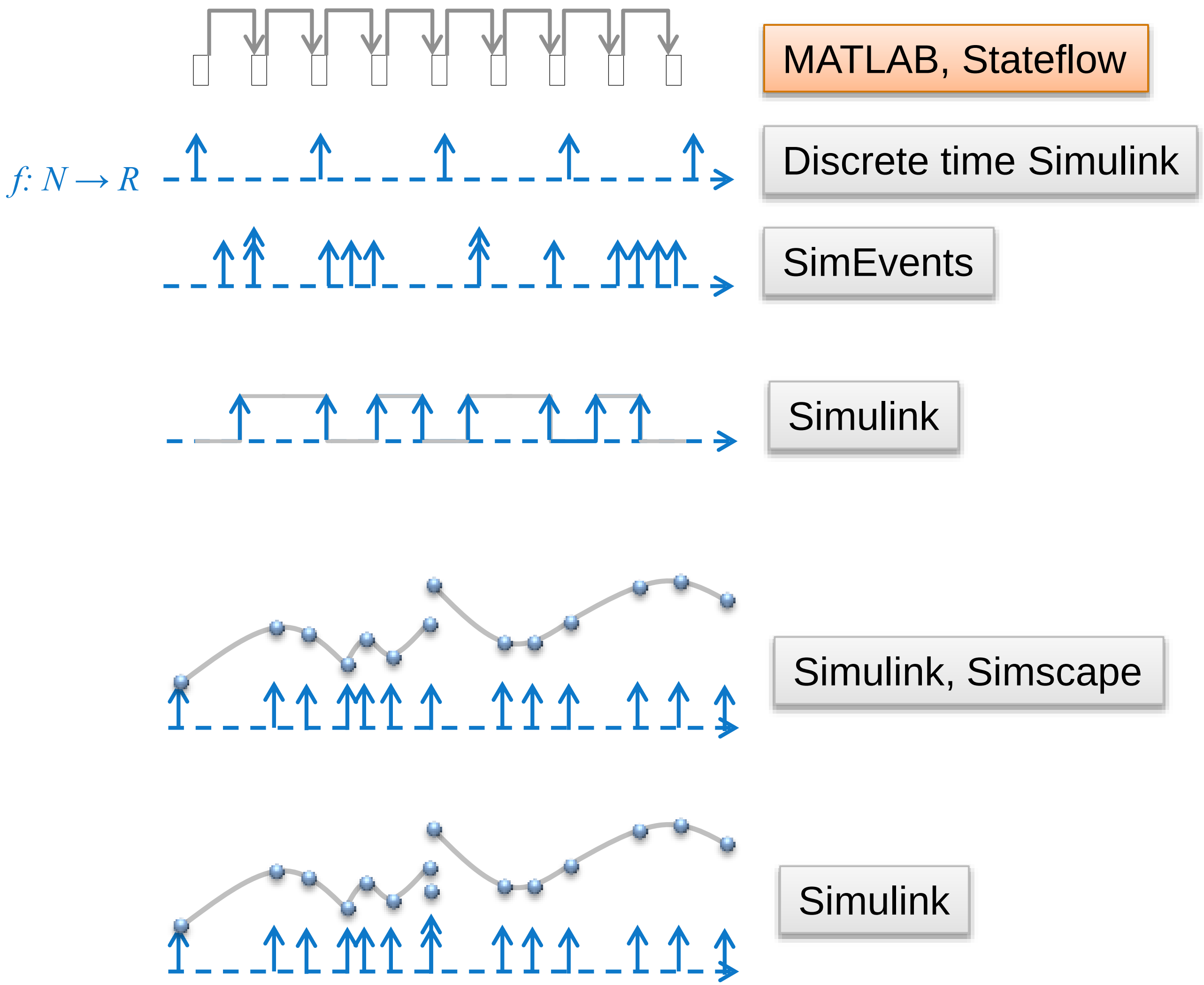
The semantic domain of a dynamic system

- Points, [ ]
  - On **N**
  - On **R** x **N**
- Intervals, [〉 (〈 〉, 〈])
  - On **R**
- Hybrid point/interval
  - On **R**
  - On **R** x **N**



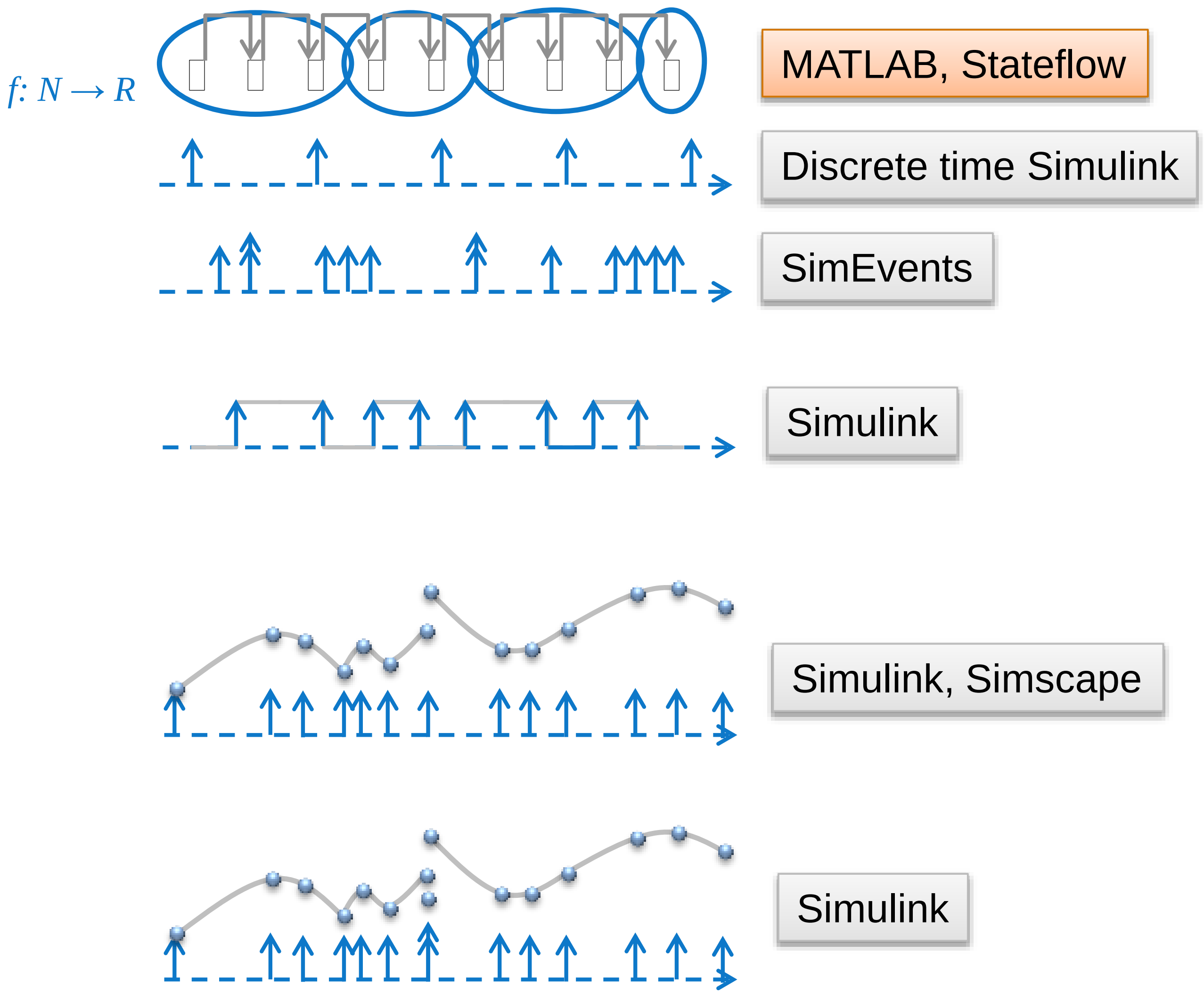
The semantic domain of a dynamic system

- Points, [ ]
  - On **N**
  - On **R** x **N**
- Intervals, [〉 (〈 〉, 〈])
  - On **R**
- Hybrid point/interval
  - On **R**
  - On **R** x **N**



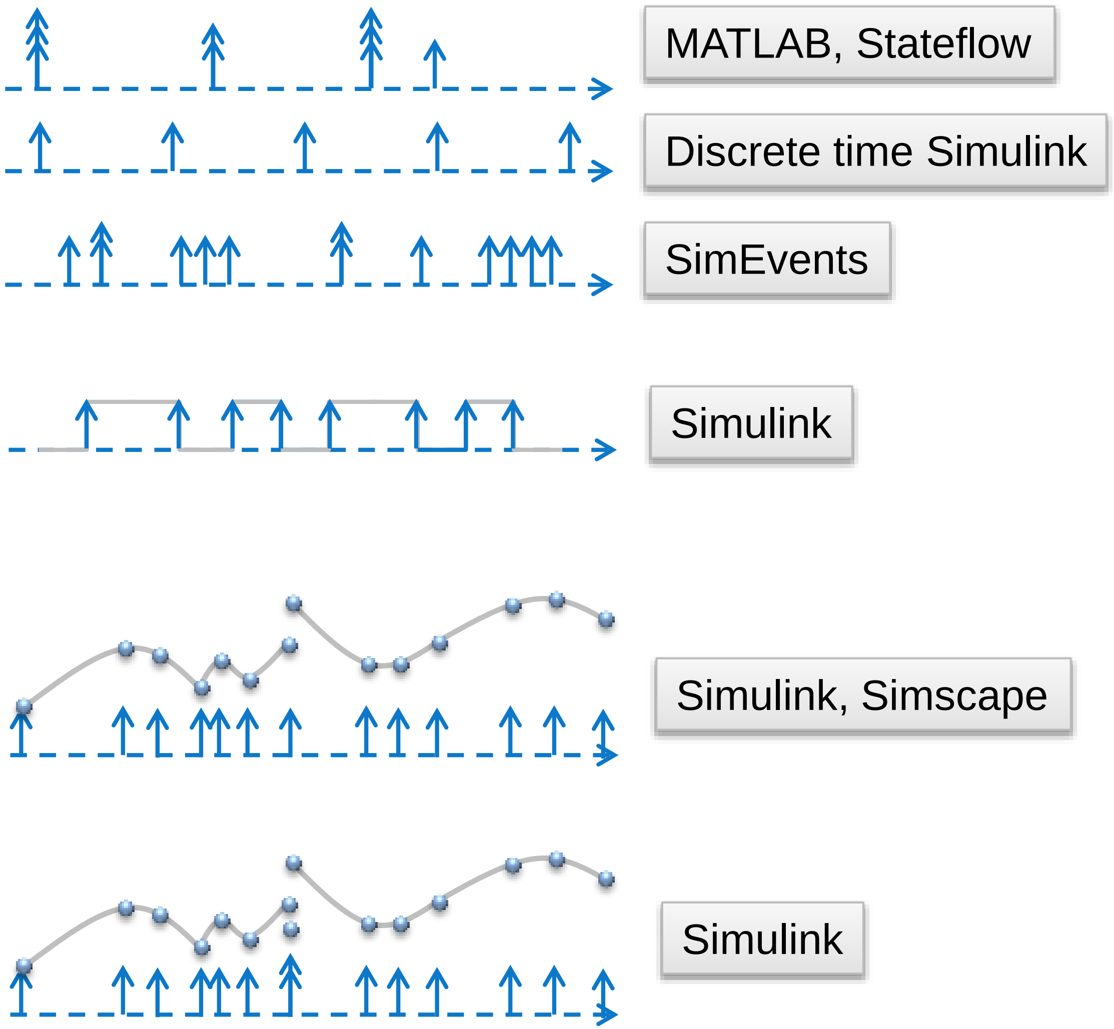
The semantic domain of a dynamic system

- Points, [ ]
  - On **N**
  - On **R** x **N**
- Intervals, [〉 (〈 〉, 〈])
  - On **R**
- Hybrid point/interval
  - On **R**
  - On **R** x **N**



The semantic domain of a dynamic system

- Points, [ ]
  - On **N**
  - On **R** x **N**
- Intervals, [〉 (〈 〉, 〈])
  - On **R**
- Hybrid point/interval
  - On **R**
  - On **R** x **N**



A formalism classification

	Expression systems	Finite state machines	Discrete time systems	Discrete event systems	Explicit differential equation systems	Implicit differential equation system
Paradigm	Imperative Declarative	Imperative	Declarative (typically)	Imperative	Declarative Causal	Declarative Noncausal
Domain	Point $\in$ **N** (untimed)	Point $\in$ **N** (untimed)	Point $\in$ **N** (timed)	Point $\in$ **R** (timed)	Interval $\in$ **R** (timed)	Interval $\in$ **R** (timed)
Codomain	Point $\in$ **N**, **R**	Point $\in$ **N**, **R**	Point $\in$ **R**	Point $\in$ **R**	Interval $\in$ **R**	Interval $\in$ **R**

MATLAB

- Imperative
- Assignment (destructive)
- Single control path
- Points on **N**

A formalism classification

	Expression systems	Finite state machines	Discrete time systems	Discrete event systems	Explicit differential equation systems	Implicit differential equation system
Paradigm	Imperative Declarative	Imperative	Declarative (typically)	Imperative	Declarative Causal	Declarative Noncausal
Domain	Point $\in$ **N** (untimed)	Point $\in$ **N** (untimed)	Point $\in$ **N** (timed)	Point $\in$ **R** (timed)	Interval $\in$ **R** (timed)	Interval $\in$ **R** (timed)
Codomain	Point $\in$ **N**, **R**	Point $\in$ **N**, **R**	Point $\in$ **R**	Point $\in$ **R**	Interval $\in$ **R**	Interval $\in$ **R**

SimEvents

- Imperative
- Assignment
- Multiple control path
- Points on **R** x **N**

A formalism classification

	Expression systems	Finite state machines	Discrete time systems	Discrete event systems	Explicit differential equation systems	Implicit differential equation system
Paradigm	Imperative Declarative	Imperative	Declarative (typically)	Imperative	Declarative Causal	Declarative Noncausal
Domain	Point $\in$ **N** (untimed)	Point $\in$ **N** (untimed)	Point $\in$ **N** (timed)	Point $\in$ **R** (timed)	Interval $\in$ **R** (timed)	Interval $\in$ **R** (timed)
Codomain	Point $\in$ **N**, **R**	Point $\in$ **N**, **R**	Point $\in$ **R**	Point $\in$ **R**	Interval $\in$ **R**	Interval $\in$ **R**

Stateflow

- Imperative
- Assignment
- Single control path
- Points on **R** x **N**

A formalism classification

	Expression systems	Finite state machines	Discrete time systems	Discrete event systems	Explicit differential equation systems	Implicit differential equation system
Paradigm	Imperative Declarative	Imperative	Declarative (typically)	Imperative	Declarative Causal	Declarative Noncausal
Domain	Point ∊ **N** (untimed)	Point ∊ **N** (untimed)	Point ∊ **N** (timed)	Point ∊ **R** (timed)	Interval ∊ **R** (timed)	Interval ∊ **R** (timed)
Codomain	Point ∊ **N**, **R**	Point ∊ **N**, **R**	Point ∊ **R**	Point ∊ **R**	Interval ∊ **R**	Interval ∊ **R**

Simulink

- Declarative
- Equations (causal)
- No control path
- Points, intervals on **R** x **N**

A formalism classification

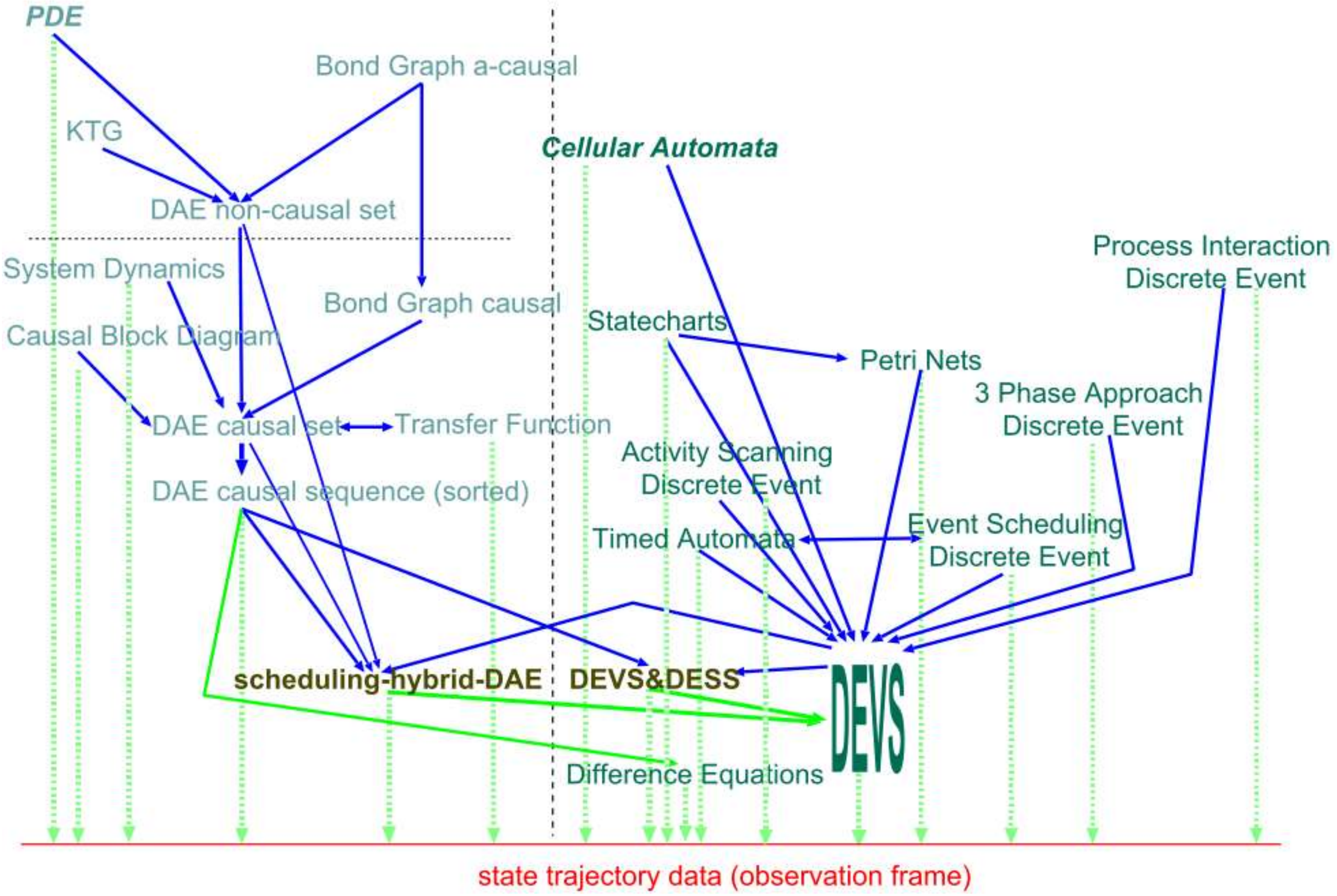
	Expression systems	Finite state machines	Discrete time systems	Discrete event systems	Explicit differential equation systems	Implicit differential equation system
Paradigm	Imperative Declarative	Imperative	Declarative (typically)	Imperative	Declarative Causal	Declarative Noncausal
Domain	Point ∊ **N** (untimed)	Point ∊ **N** (untimed)	Point ∊ **N** (timed)	Point ∊ **R** (timed)	Interval ∊ **R** (timed)	Interval ∊ **R** (timed)
Codomain	Point ∊ **N**, **R**	Point ∊ **N**, **R**	Point ∊ **R**	Point ∊ **R**	Interval ∊ **R**	Interval ∊ **R**

Simscape

- Declarative
- Noncausal equations
- No control path
- Intervals on **R**

A general operational (computational) semantic domain

- Points, [ ]
  - On **R** x **N**



H. Vangheluwe and G. C. Vansteenkiste.
" A Multi-Paradigm Modeling and Simulation Methodology: Formalisms and languages,".
European Simulation Symposium (ESS), Genoa, Italy. pp. 168--172. SCS, October 1996.

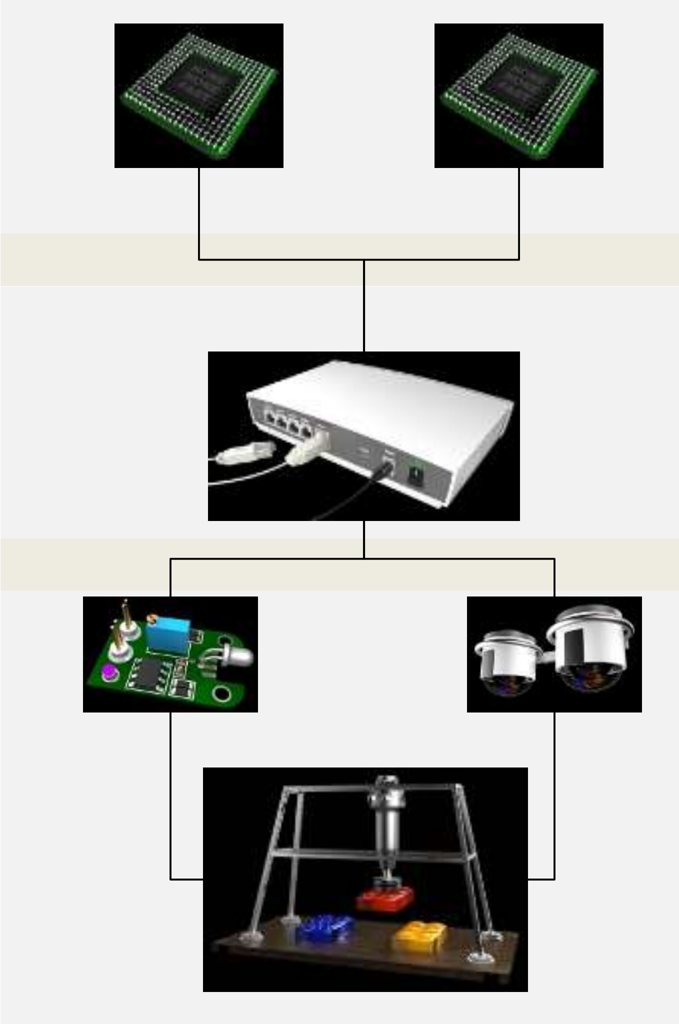
A general operational (computational) semantic domain

- Points, [ ]
  - On **R** x **N**

- Without losing the analysis ability and efficiency
  - Integer precision
  - Clock calculus
  - Scheduling
    - Static when possible
    - Dynamic when necessary

Classes of execution behavior

- Controller, sampled time (ST)
  - Discrete time
  - Frequent periodic time-events
  - Events are known before execution
- Network/OS, event driven (ED)
  - Discrete time
  - Frequent aperiodic time-events
  - Events occur during execution
- Plant, time integrated (TI)
  - Continuous time
  - Sporadic aperiodic state-events (zero crossings)
  - Events occur during execution

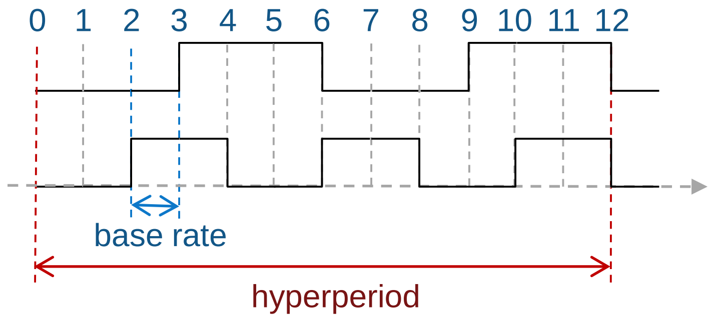


Event classification

	predict	unknown
periodic	**ST**	-
aperiodic	**ED**	**TI**

Controller—dedicated time-driven solver

- Discrete time
  - Greatest common divisor

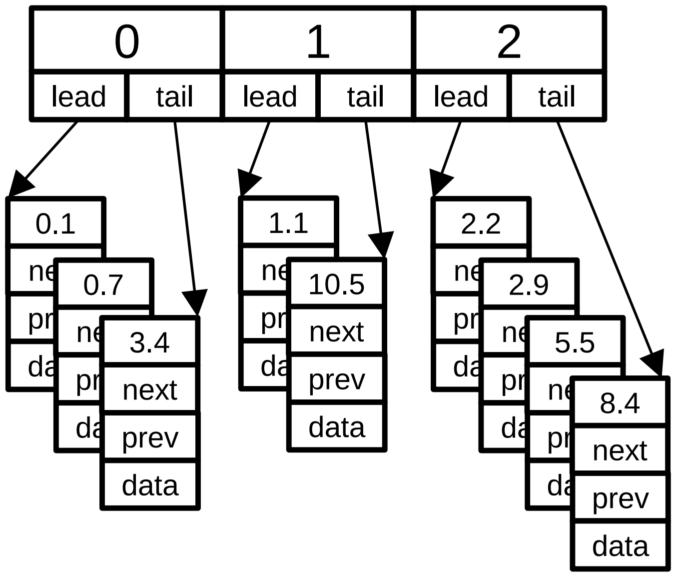


  - Hyperperiod (more restrictive ‘harmonic’ for multitasking)
  - Create a static (integer) schedule
- Map integer schedule onto logical time
  - Base rate has a logical time duration
- Earliest future event time
  - Time up to which to integrate

Network—dedicated event-driven solver

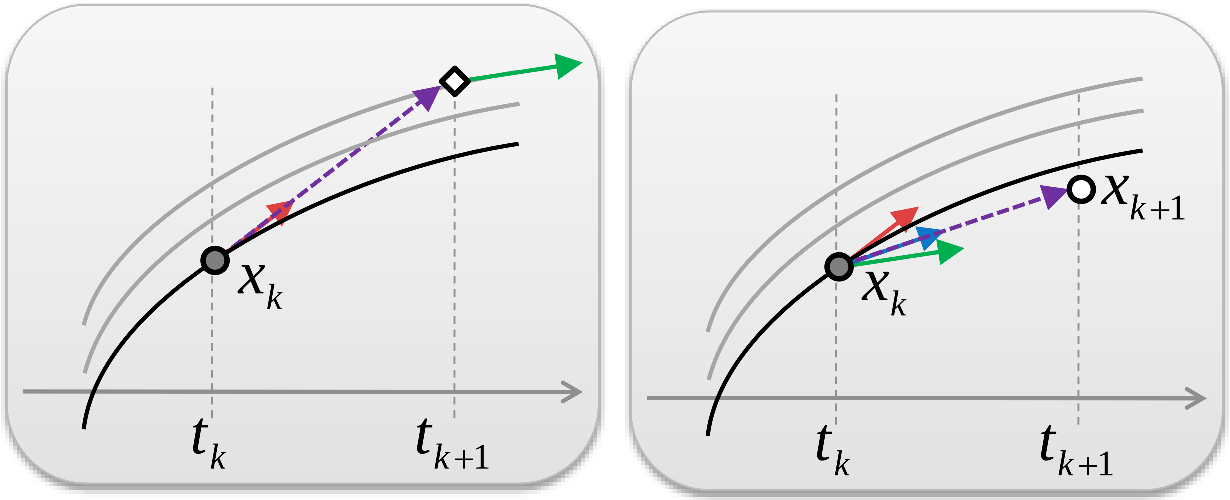
- Dynamically generate events
- Keep list of events
  - Ordered based on time of occurrence
  - A (max, plus) algebra
- An event calendar
  - Data structure for efficient
    - Event insertion
    - Event deletion
  - Often times a hybrid
    - For example, an array + doubly linked list
- Set time to first event time
  - Process this event

20 [ms]	send_service_req
220 [ms]	resend_service_req
370 [ms]	receive_trans_ack
650 [ms]	receive_arrived_ack



Physics—dedicated time-driven solver

- Numerical solver integrates time (step *h*)



$$\hat{x}_t(t_{k+1}) = x(t_k) + \frac{\dot{x}(t_{k+1}) + \dot{x}(t_k)}{2} h_k$$

$$\dot{x}(t_k) = f\left(x(t_k), t_k\right)$$

$$\dot{x}(t_{k+1}) = f\left(x(t_k) + h\dot{x}(t_k), t_{k+1}\right)$$


System characterization
A requirements perspective
The Towers of Hanoi revisited
Multiformalism

Do not fall in love with your model

-- Jacques LeFèvre

MathWorks®

Accelerating the pace of engineering and science