Overview

1. Ordinary Differential Equations, Algebraic Equations
2. Block Diagrams
3. Causal Block Diagrams
   - denotational semantics (modelling)
   - operational semantics (simulation)
4. Virtual Experimentation
5. Assignment
6. The Modelling and Simulation Process

The Mass-Spring problem (recap)

- physical system: mass - spring
- a priori knowledge: Newton's Law, ideal spring law
- measurement data: position in function of time, with noise
- goals:
  - determine model structure
  - determine spring constant (strength) parameter value
  - perform what-if experiments
- model structure: use ideal spring constant amplitude **feature**
- model: `x'' = -K/M*x` with Initial Conditions
- estimate parameter (**calibrate** model) through

  - accurate simulation of model
  - optimization of a “fit” criterion
- once the spring constant is known: what-if experiments

Algebraic Equations, Ordinary Differential Equations

- declarative relationship (constraints) between variables

$$a + b = c + 2$$

- declarative relationship (constraints) between functions (of time)

$$\frac{dx(t)}{dt} = f(x(t), t, P)$$

- Derivative with respect to time is **rate of change**
- Graphically: derivative is **slope of tangent**
- Need initial conditions for unique solution
- Higher order differential equations

Modelling Formalisms: Block Diagrams

- Visual representation of structure and behaviour of systems
- Can be complete and rigourous **if** formally specified
  - denotational semantics: declarative ($\equiv$ modelling operations)
  - operational semantics: how to solve ($\rightarrow$ simulator)
- Examples: causal block diagrams, flow charts, state charts, ...
- Can be used to analyze, design, specify, implement systems.
- Important issues:
  - concurrency (implementation may be sequential)
  - hierarchy (closure under composition ?)

Causal Block Diagrams

- Denotational Semantics: **sets** of algebraic equations and ODEs
- Concurrency is inherent (no order is specified), but . . .
- Order matters when simulating !!!!!!!!
- Analog Computers: solution of DB are signals . . .
- Continuous System Modelling Program (CSMP): emulate analog computers
- Operational Semantics: data flow ?

Representing ODEs in BD form

$$\frac{dx(t)}{dt} = f(x)$$

- Non-causal Diagram
- Causal Block Diagram

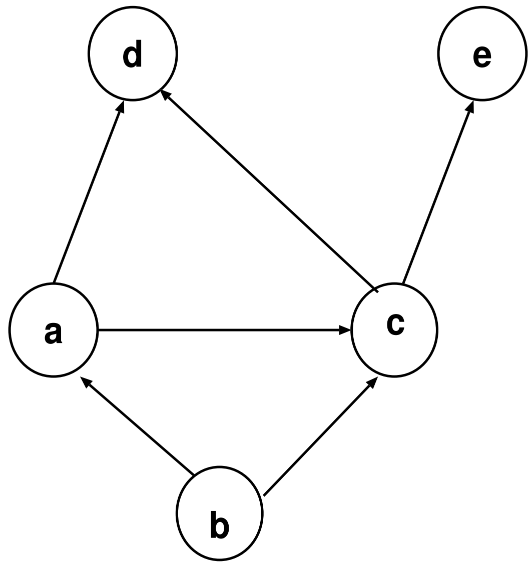
Representing algebraic equations in BD form

Can a data-flow appoach be used to build a simulator ?

- No loop
- Loop

Determining Order: Dependency Graph

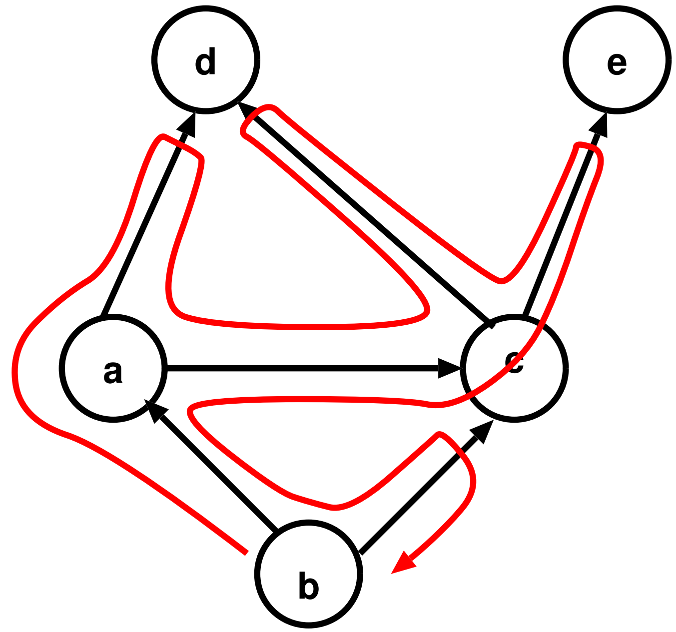
$$\begin{cases} a = d - c \\ b = c + a - 3 \\ c = 6 + d * e \\ d = 2 \\ e = 3 \end{cases}$$

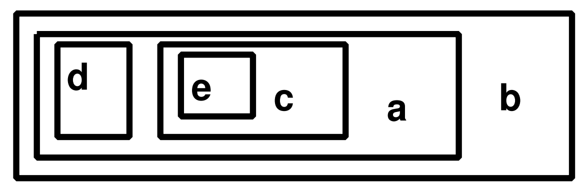


Problems with model re-use: sorting post-order traversal depth-first search

1. find_root(s)
2. DFS(root)

```
DFS(node)
{
    if (not_visited(node))
    {
        mark_visited(node)
        foreach child_node of node
        {
            DFS(child_node)
        }
        print(node)
    }
}
```





Dependency Cycle (aka Algebraic Loop)

$$\begin{cases} x &= y+16 \\ y &= -x-z \\ z &= 5 \end{cases}$$

can *never* be sorted due to a dependency *cycle*

aka *strong component* (every vertex in the component is reachable from every other)

$$x \to y \to x$$

May be solved implicitly

$$
\left[
\begin{array}{c}
z = 5 \\
\left\{
\begin{array}{rcl}
x - y & = & -6 \\
x + y & = & -z
\end{array}
\right.
\end{array}
\right.
$$

Implicit set of n equations in n unknowns.

- non-linear $\rightarrow$ non-linear residual solver.
- linear $\rightarrow$ numeric or symbolic solution.

May be solved symbolically (if linear and not too large)

$$x = \frac{\begin{vmatrix} -6 & -1 \\ -z & 1 \end{vmatrix}}{\begin{vmatrix} 1 & -1 \\ 1 & 1 \end{vmatrix}} = \frac{-6-z}{2} \; ; \; y = \frac{\begin{vmatrix} 1 & -6 \\ 1 & -z \end{vmatrix}}{\begin{vmatrix} 1 & -1 \\ 1 & 1 \end{vmatrix}} = \frac{6-z}{2}$$

$$\left[\begin{array}{lcl} z & = & 5 \\ x & = & \frac{-6-z}{2} \\ y & = & \frac{6-z}{2} \end{array} \right.$$

Simple Loop Detection

1. Build dependency matrix D
2. Calculate transitive closure D^*
3. If $True$ on diagonal of D^*, a loop exists

Even with Warshall's algorithm, still $O(n^3)$ and don't know immediately which nodes involved in the loop(s).

Tarjan's $O(n+m)$ Loop Detection (1972)

1. Complete Depth First Search (DFS) on G
   (possibly multiple DFS trees), postorder numbering

```
FOREACH v IN V
 dfsNr[v] <- 0
FOREACH v IN V
 IF dfsNr[v] == 0
  DFS(v)
```

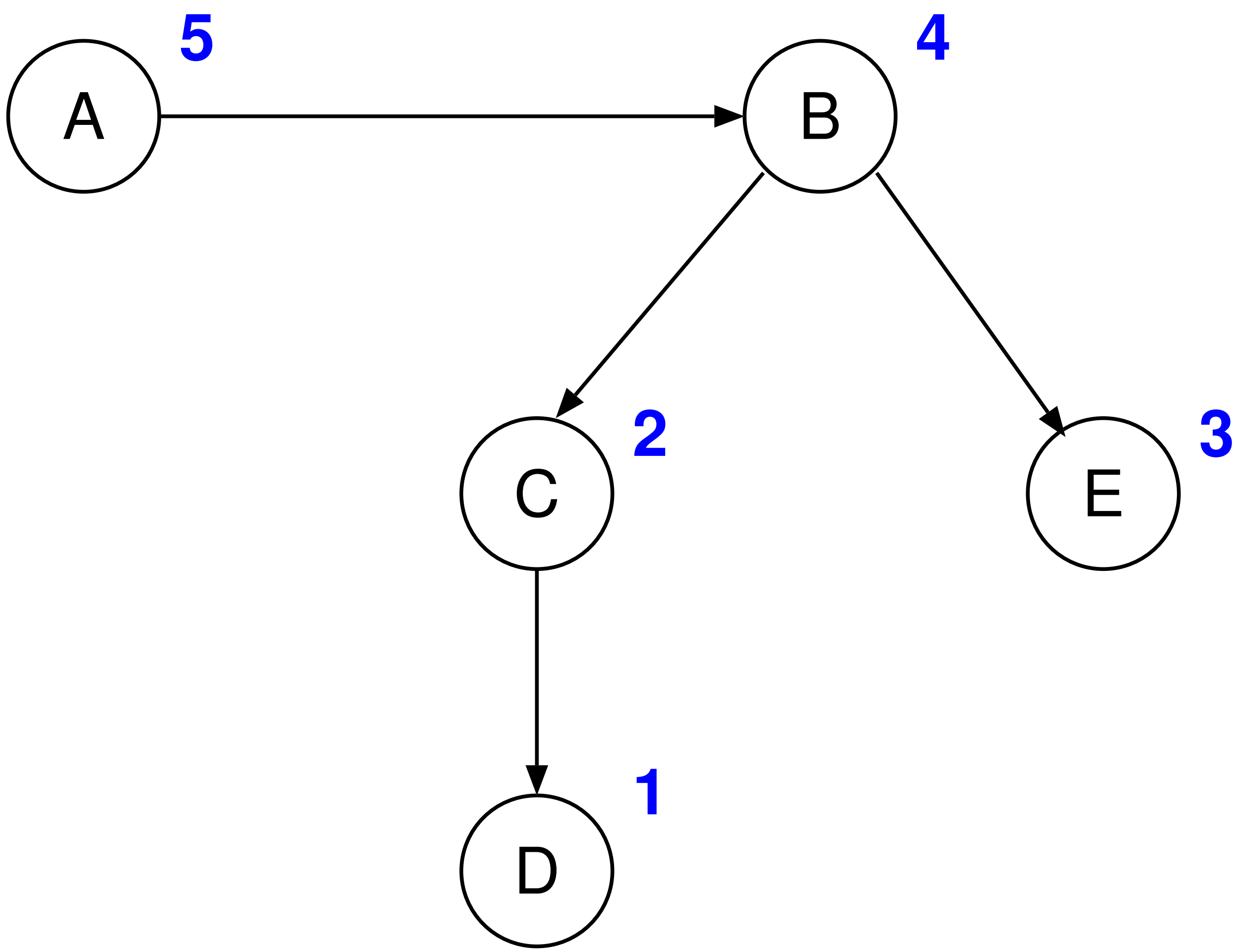
2. Reverse edges in the annotated $G \to G_R$

3. DFS on G_R starting with highest numbered v
   set of vertices in each DFS tree $=$ strong component.
   Remove strong component and repeat.

Set of Algebraic Eqns, no Loops

$$
\begin{cases}
a & = & b^2 + 3 \\
b & = & sin(c \times e) \\
c & = & \sqrt{d - 4.5} \\
d & = & \pi/2 \\
e & = & u()
\end{cases}
$$

Sorting, no Loops



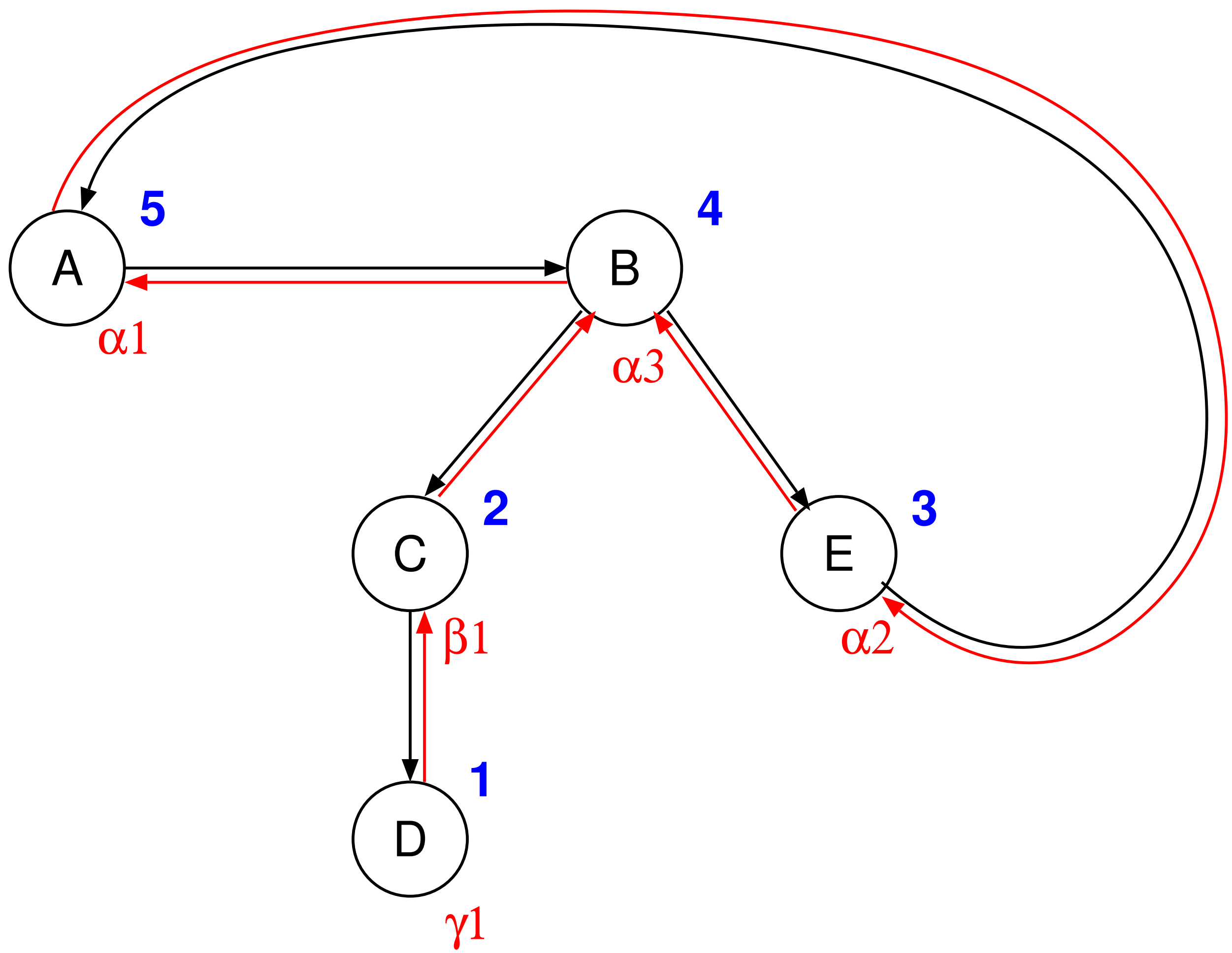
Sorting Result

$$\left[\begin{array}{lcl} d & = & \pi/2 \\ e & = & u() \\ c & = & \sqrt{d-4.5} \\ b & = & sin(c \times e) \\ a & = & b^2+3 \end{array}\right.$$

Algebraic Loop (Cycle) Detection

$$
\begin{cases}
a &= b^2 + 3 \\
b &= sin(c \times e) \\
c &= \sqrt{d - 4.5} \\
d &= \pi/2 \\
e &= a^2 + u()
\end{cases}
$$

Algebraic Loop (Cycle) Detection

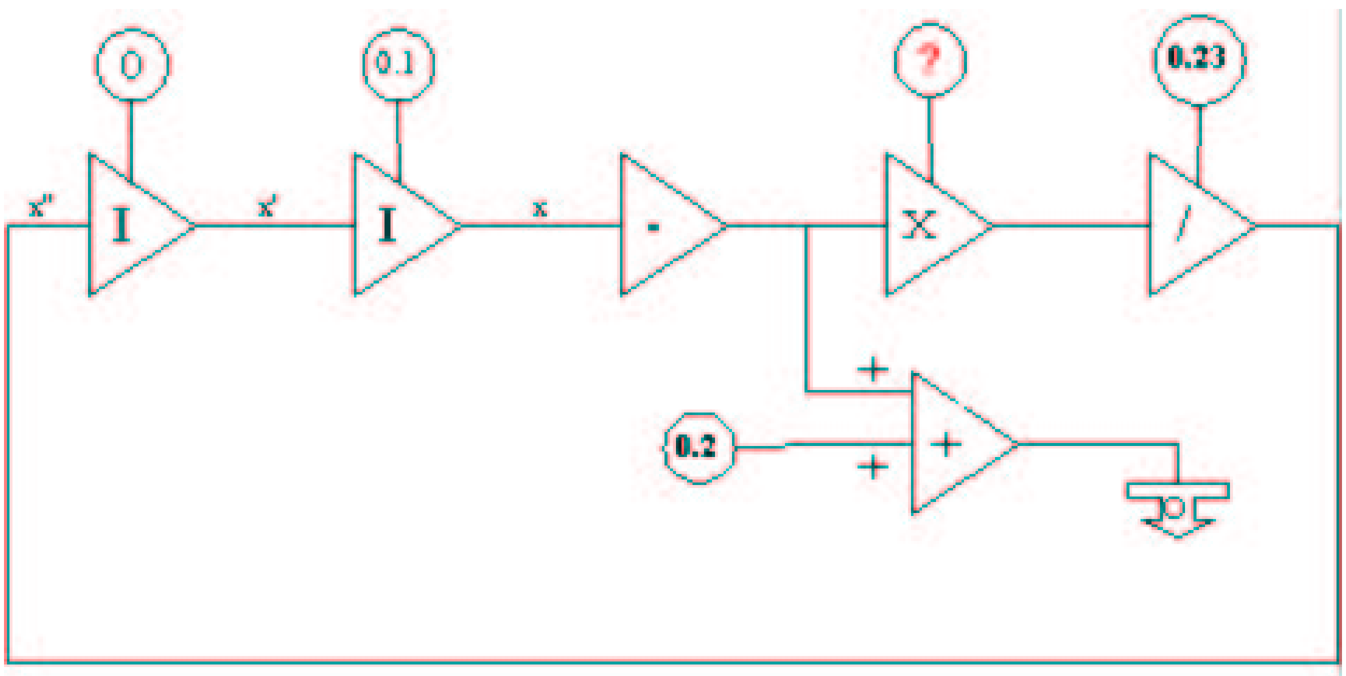


Algebraic Loop (Cycle) Detection Result

$$\left[\begin{array}{l} d = \pi/2 \\ c = \sqrt{d-4.5} \\ \left\{\begin{array}{lcl} b & = & sin(c \times e) \\ a & = & b^2 + 3 \\ e & = & a^2 + u() \end{array}\right. \end{array}\right. \; ; \; \left[\begin{array}{l} d = \pi/2 \\ c = \sqrt{d-4.5} \\ \left\{\begin{array}{cccccc} & b & -sin(c \times e) & & = & 0 \\ a & -b^2 & & -3 & = & 0 \\ a^2 & & -e & +u() & = & 0 \end{array}\right. \end{array}\right.$$

Causal Block Diagram for the Mass-Spring Model

$$\frac{d^2x}{dt^2} = -\frac{K}{M}x$$



Time-slicing simulator pseudo-code

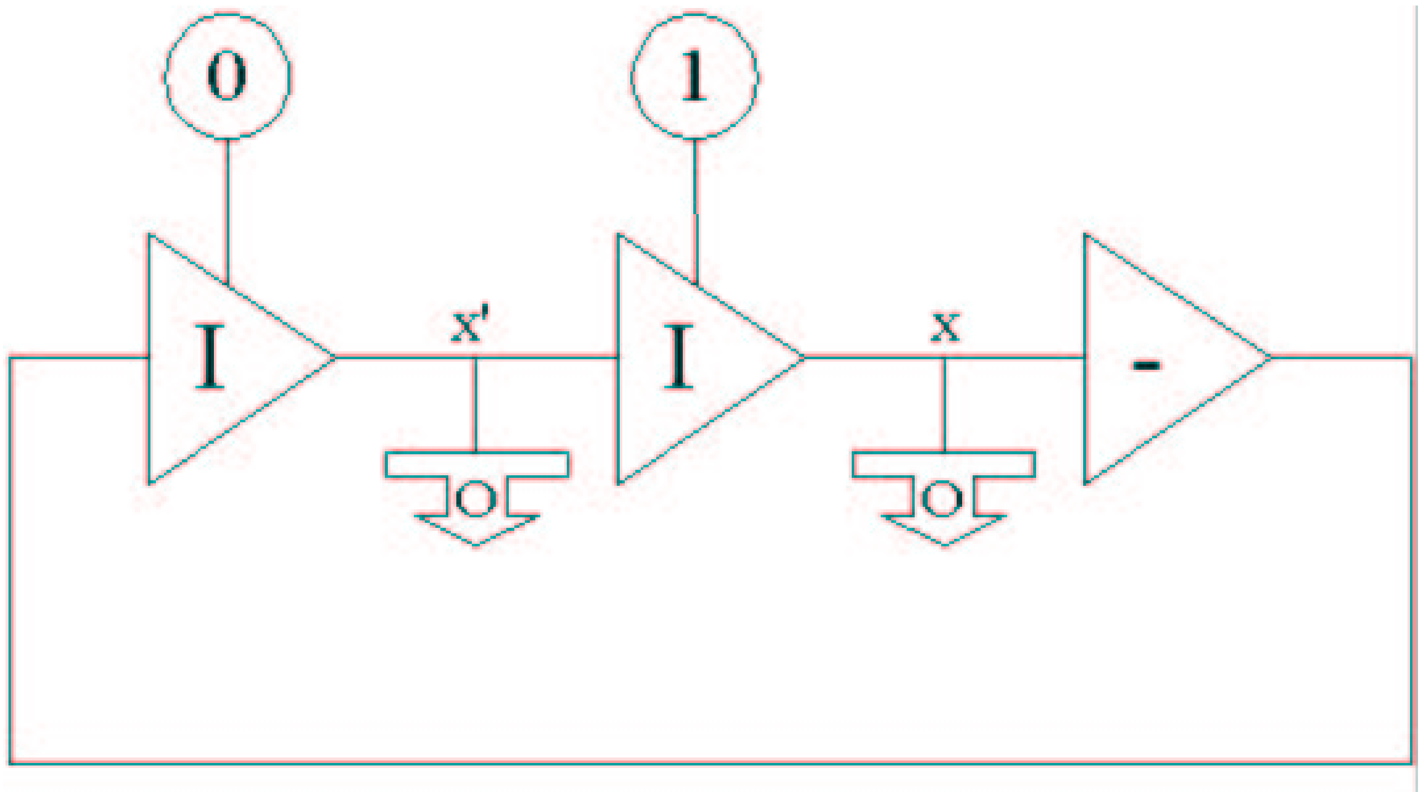
```
Main program:

 FOREACH block IN system
  IF block is integrator
   initialise block's output with initial condition (IC)
  ELSE
   initialise output with 0

 READ system network (graph) structure
 DETECT algebraic loops (replace by single block with solution or halt)
 SORT the non-integrator blocks

 READ integrator configuration info
  step_size
  communication_interval

 READ experiment info
  start_time
  end_time
  parameters
  initial conditions
```

Time-slicing simulator pseudo-code (ctd.)

```
Simulation Kernel Loop:
 WHILE (NOT End_of_simulation) DO
  Update_blocks
  Output time and state variables

Update_blocks:
 FOREACH{in proper order} block IN system
  given current block inputs
  (get input from output of influencer)
  Calculate_block_output for block
 increment current_time with stepsize

End_of_simulation:
 termination condition such as
  current_time >= end_time
  condition(state_values) == TRUE
```

```
Calculate_block_output: ([...] means optional)

  WeightedSum
    W, block_number, P1, e1[, P2, e2[, P3, e3]] ; --->
       output= SUM_i(Pi*ei)
  Summer
    +, block_number, P1, e1[, P2, e2[, P3, e3]] ; --->
       output = SUM_i(sign(Pi)*ei)
       (only the sign of Pi is used)
  Integrator
    I, block_number, IC, e1 ; --->
       output= previous_output +  step_size*e1
       (simple fixed-step Euler integration)
  Minus (Sign Inverter)
    -, block_number, e1 ; --->
       output= -e1.
  Multiplier
    X, block_number, e1, e2 ; --->
       output= e1*e2.
  Divider
    /, block_number, e1, e2 ; --->
       output= e1/e2.
  Constant
    K, block_number, P1 ; --->
       output= P1.
  Output
    O, block_number, e1 ; --->
       output= e1.
       (As a side-effect, the (time, e1) tuple is put
        on the output stream at every communication point).
```

Time Slicing: Circle Test



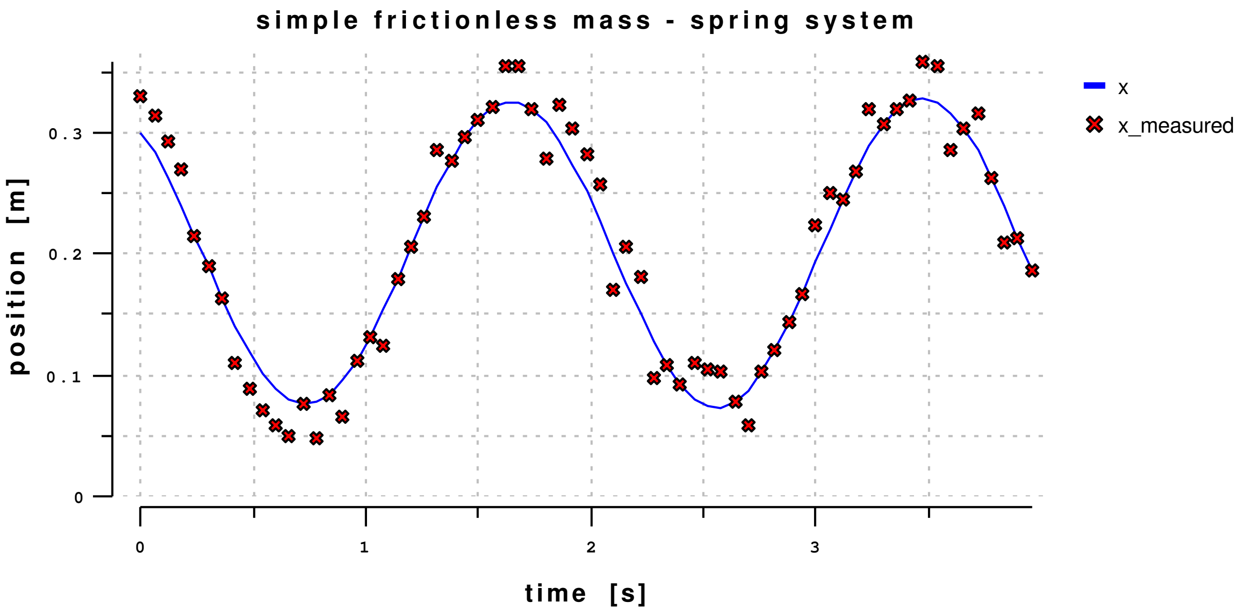
```
; Circle Test for
; CSMP-style Time Slicing Simulator

$endtime = 100;
$timestep = ?;
$comminterval = 1.5;

I, 1, 0, 3        ; x' is integral of x'', IC=0
I, 2, 1, 1        ; x  is integral of x',  IC=1
-, 3, 2           ; -x
O, 4, 1           ; output x'
O, 5, 2           ; output x
```

Experimentation

Needed (to be specified by user in a tool):

1. Model
2. Parameters (constant for each simulation run)
3. Initial Conditions
4. Input (file, interactive, real system)
5. Output (file, plot, real system)
6. Solver Configuration
7. Experiment type (simulation, optimization, parameter estimation $=$ model calibration)

Model Calibration: Parameter Fit



From Here On . . .

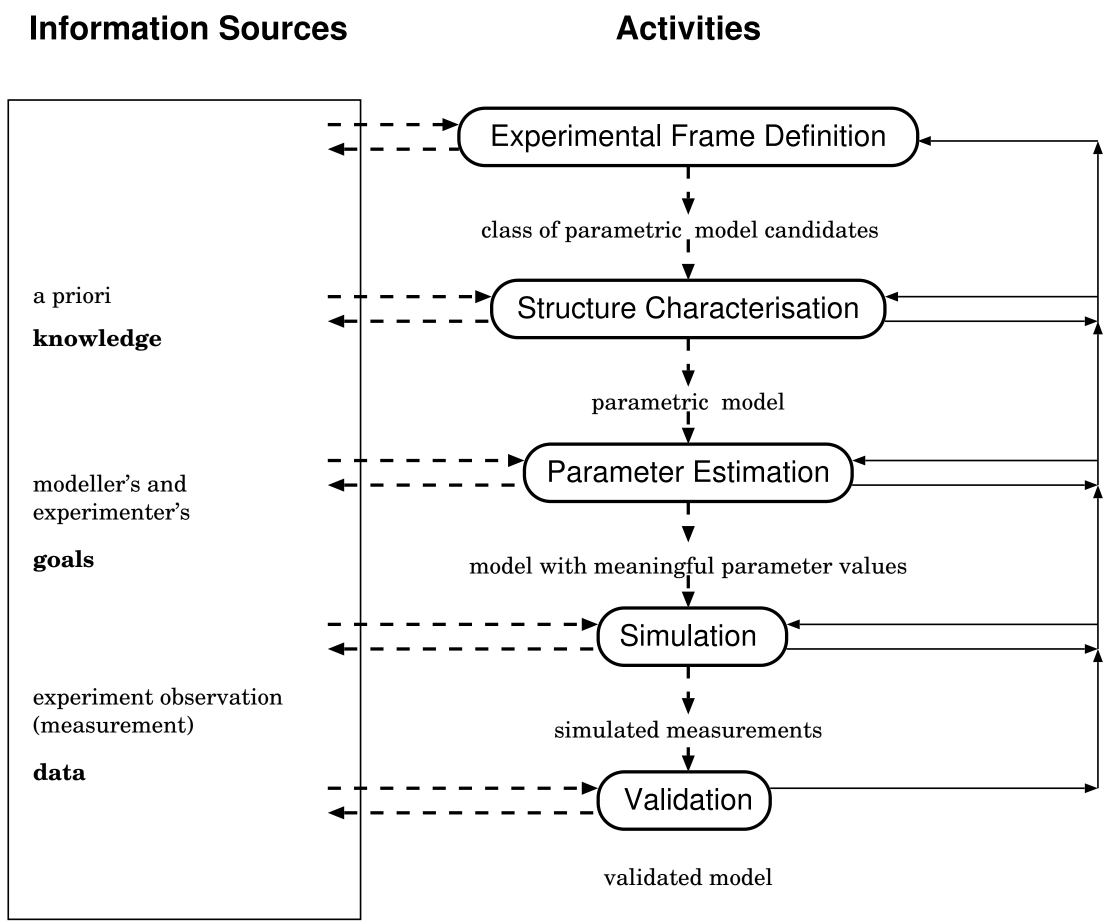
- Virtual Experiments: simulation, optimisation, what-if, . . .
- Validation/Falsification

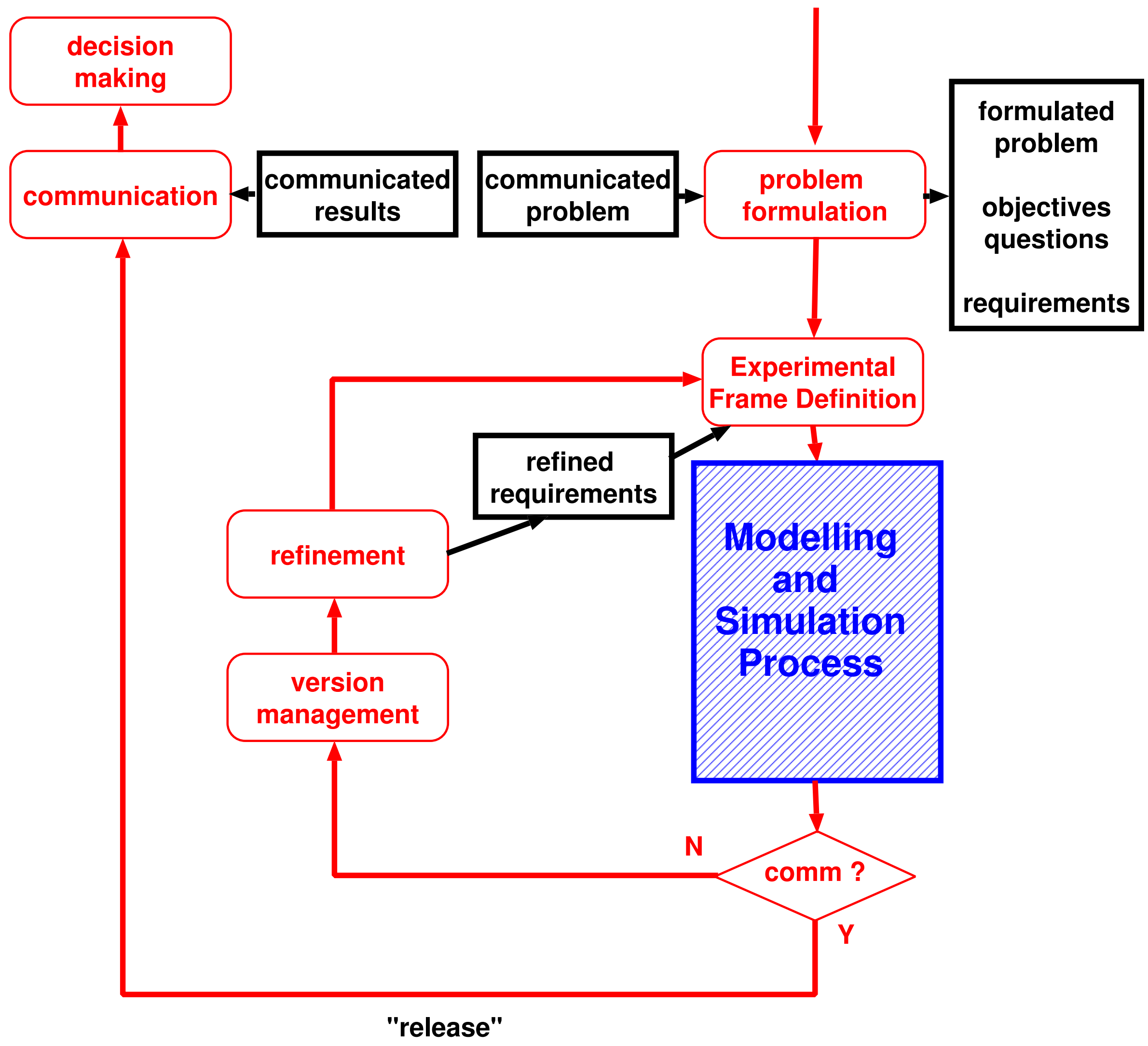
Assignment

Experimentation Environment:

- Cases: circle test (phase plots), mass-spring
- Starting points: modelling environment, saving, plotter
- See above (include experiment saving !).
- Report on WWW.
- Document !
- 2 weeks: 26 September.

Modelling and Simulation *Process*




decision
making
communication
communicated
results
communicated
problem
problem
formulation
formulated
problem
objectives
questions
requirements
Experimental
Frame Definition
refined
requirements
refinement
Modelling
and
Simulation
Process
version
management
N
comm ?
Y
"release"