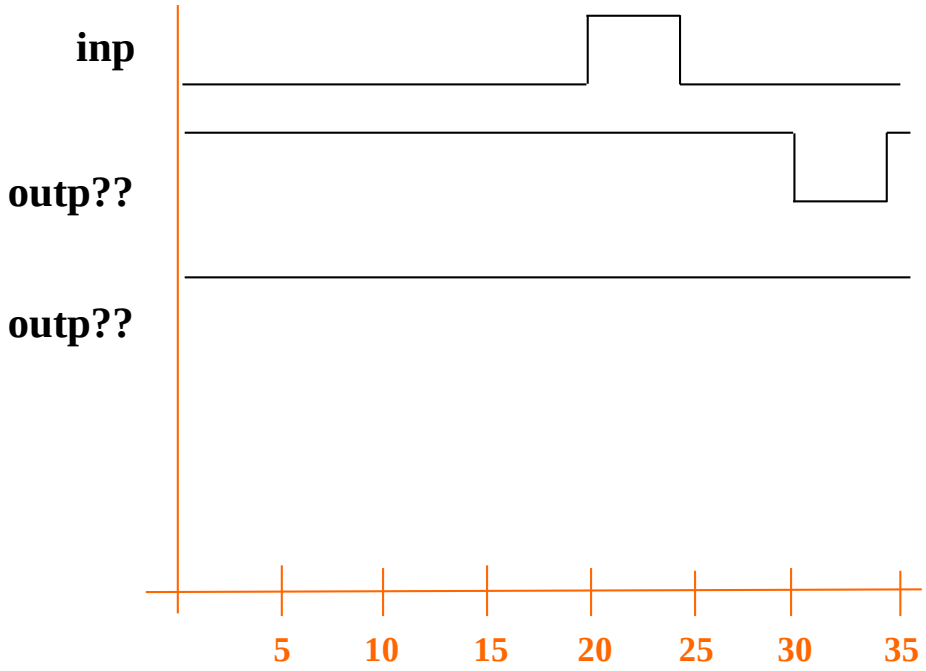
Embedded Systems 11

REVIEW: Inertial and transport delay model

- Example for signal assignment:

  outp <= **not** inp **after** 10 ns;



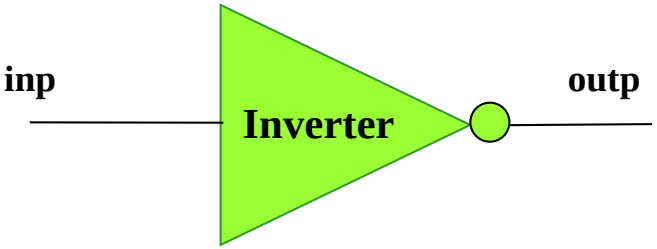
REVIEW: Semantics of transport delay model

Signal assignments change transaction list.

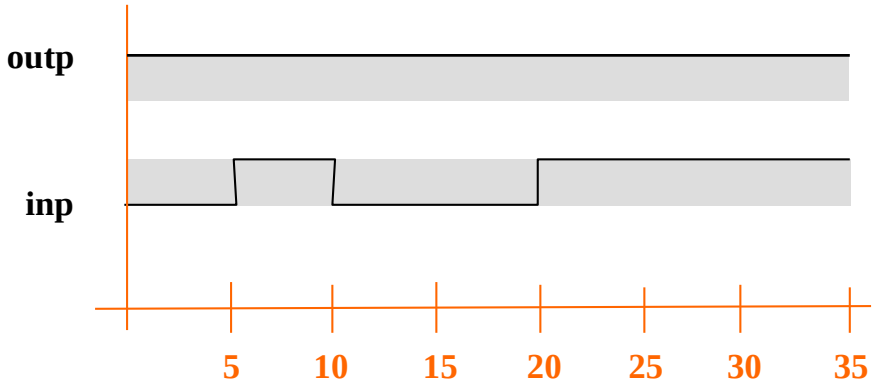
- Before transaction (s, t_1, v_1) is inserted into transaction list, all transactions in the transaction list (s, t_2, v_2) with $t_2 \geq t_1$ are removed from transaction list.

Example for transport delay model

```
inv : process(inp)
begin
    if inp=`1` then
        outp <= transport `0` after 20 ns;
    elsif inp=`0` then
        outp <= transport `1` after 12.5 ns
    end if;
end process inv;
```



- Transaction list:
  - At 5ns:
    (outp, 25ns, `0`)
  - At 10 ns:
    (outp, 22.5ns, `1`), (outp, 25ns, `0`)
    Remove (outp, 25ns, `0`)!
    → (outp, 22.5ns, `1`)



Semantics of inertial delay model

- Semantics for more general version of inertial delay statement:
  - Inertial delay absorbs pulses at the inputs which are shorter than the delay specified for the gate / operation.
  - Key word **reject** permits absorbing only pulses which are shorter than specified delay:
    - Example:
      - outp <= **reject** 3 ns **inertial not** inp **after** 10 ns;
      - Only pulses smaller than 3 ns are absorbed.
      - outp <= **reject** 10 ns **inertial not** inp **after** 10 ns;
        and
        outp <= **not** inp **after** 10 ns;
        are equivalent.

Semantics of inertial delay model

- Rule 1 as for transport delay model:
  Before transaction (s, t_1, v_1) is inserted into transaction list, all transactions in the transaction list (s, t_2, v_2) with $t_2 \geq t_1$ are removed from transaction list.
- Rule 2 removes also some transactions with times < t_1:
  - Suppose the time limit for reject is rt.
  - Transactions for signal s with time stamp in the intervall (t_1 – rt, t_1) are removed.
  - Exception:
    If there is in (t_1 – rt, t_1) a subsequence of transactions for s immediately before (s, t_1, v_1) which also assign value v_1 to s, then these transactions are preserved.

Example

```
process
begin
    o1 <= transport `0`, `0` after 5ns, `1` after 15 ns, `0` after 20ns,
    `1` after 25 ns, `1` after 30ns, `1` after 45 ns,
                                `0` after 50 ns;
    -- same signal assignment for o2
    o2 <= transport `0`, `0` after 5ns, `1` after 15 ns, `0` after 20ns,
    `1` after 25 ns, `1` after 30ns, `1` after 45 ns,
                                `0` after 50 ns;
    wait for 15 ns;
     o2 <= reject 22 ns inertial `1` after 25 ns;
    wait;
end process;
```

- Transaction list until „**wait for** 15 ns“:
  (o1, 0ns, `0`), (o1, 5ns, `0`), (o1, 15ns, `1`), (o1, 20ns, `0`), (o1, 25ns, `1`), (o1, 30ns, `1`), (o1, 45ns, `1`), (o1, 50ns, `0`),
  (o2, 0ns, `0`), (o2, 5ns, `0`), (o2, 15ns, `1`), (o2, 20ns, `0`), (o2, 25ns, `1`), (o2, 30ns, `1`), (o2, 45ns, `1`), (o2, 50ns, `0`)
- Transaction list when process is reactivated at time 15ns:
  (o1, 20ns, `0`), (o1, 25ns, `1`), (o1, 30ns, `1`), (o1, 45ns, `1`), (o1, 50ns, `0`),
  (o2, 20ns, `0`), (o2, 25ns, `1`), (o2, 30ns, `1`), (o2, 45ns, `1`), (o2, 50ns, `0`)
- …

Example

```
process
begin
    o1 <= transport `0`, `0` after 5ns, `1` after 15 ns, `0` after 20ns,
    `1` after 25 ns, `1` after 30ns, `1` after 45 ns,
                                `0` after 50 ns;
    -- same signal assignment for o2
    o2 <= transport `0`, `0` after 5ns, `1` after 15 ns, `0` after 20ns,
    `1` after 25 ns, `1` after 30ns, `1` after 45 ns,
                                `0` after 50 ns;
    wait for 15 ns;
     o2 <= reject 22 ns inertial `1` after 25 ns;
    wait;
end process;
```

- At time 15ns:
  - insert transaction (o2, 40ns, `1`).
  - Remove transactions with time stamp ≥ 40ns.
- Results in preliminary transaction list:
  (o1, 20ns, `0`), (o1, 25ns, `1`), (o1, 30ns, `1`), (o1, 45ns, `1`), (o1, 50ns, `0`),
  (o2, 20ns, `0`), (o2, 25ns, `1`), (o2, 30ns, `1`), (o2, 40ns, `1`)
- …

Example

```
process
begin
    o1 <= transport `0`, `0` after 5ns, `1` after 15 ns, `0` after 20ns,
    `1` after 25 ns, `1` after 30ns, `1` after 45 ns,
                                `0` after 50 ns;
    -- same signal assignment for o2
    o2 <= transport `0`, `0` after 5ns, `1` after 15 ns, `0` after 20ns,
    `1` after 25 ns, `1` after 30ns, `1` after 45 ns,
                                `0` after 50 ns;
    wait for 15 ns;
     o2 <= reject 22 ns inertial `1` after 25 ns;
    wait;
end process;
```

- Results in preliminary transaction list:
  (o1, 20ns, `0`), (o1, 25ns, `1`), (o1, 30ns, `1`), (o1, 45ns, `1`), (o1, 50ns, `0`), (o2, 20ns, `0`), (o2, 25ns, `1`), (o2, 30ns, `1`), (o2, 40ns, `1`)
- Rule 2:
  - (o2, 25ns, `1`), (o2, 30ns, `1`) are preserved,
  - (o2, 20ns, `0`) is removed.
- Resulting transaction list:
  (o1, 20ns, `0`), (o1, 25ns, `1`), (o1, 30ns, `1`), (o1, 45ns, `1`), (o1, 50ns, `0`), (o2, 25ns, `1`), (o2, 30ns, `1`), (o2, 40ns, `1`)

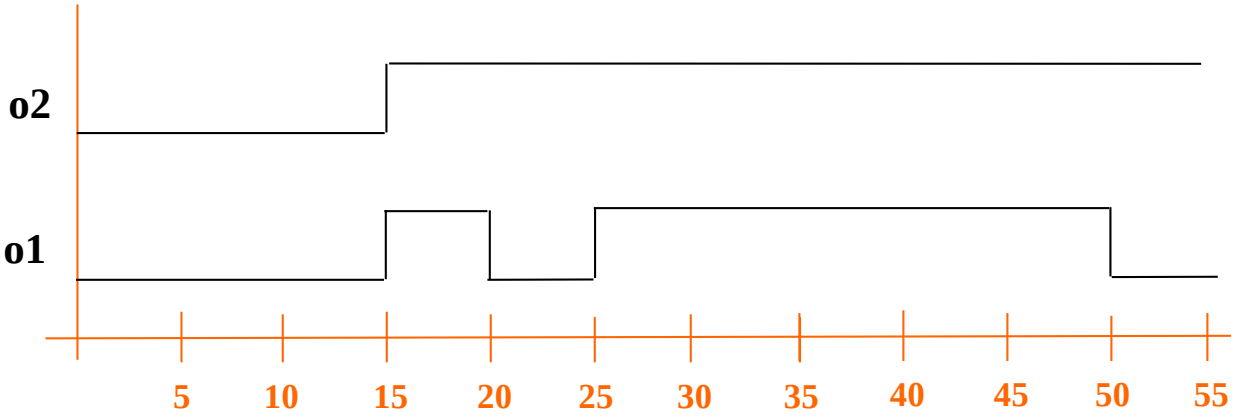
Rule 2:

- Transactions for signal o2 with time stamp in the intervall (40ns – 22ns, 40ns) = (18ns, 40ns) are removed.
- Exception:
  If there is in (18ns, 40ns) a subsequence of transactions for o2 immediately before (o2, 40ns, `1`) which also assign value `1` to o2, then these transactions are preserved.

Example

```
process
begin
    o1 <= transport `0`, `0` after 5ns, `1` after 15 ns, `0` after 20ns,
    `1` after 25 ns, `1` after 30ns, `1` after 45 ns,
                                `0` after 50 ns;
    -- same signal assignment for o2
    o2 <= transport `0`, `0` after 5ns, `1` after 15 ns, `0` after 20ns,
    `1` after 25 ns, `1` after 30ns, `1` after 45 ns,
                                `0` after 50 ns;
    wait for 15 ns;
     o2 <= reject 22 ns inertial `1` after 25 ns;
    wait;
end process;
```

- Resulting wave form:



Functions and procedures

- Apart from entities / architectures there are also functions and procedures in the usual (software) sense.
- Functions are typically used for providing conversion between data types or for defining operators on user-defined data types.
- Procedures may have parameters of directions **in**, **out** and **inout**.
  - **in** comparable to *call by value*,
  - **out** for providing results,
  - **inout** comparable to *call by reference*.

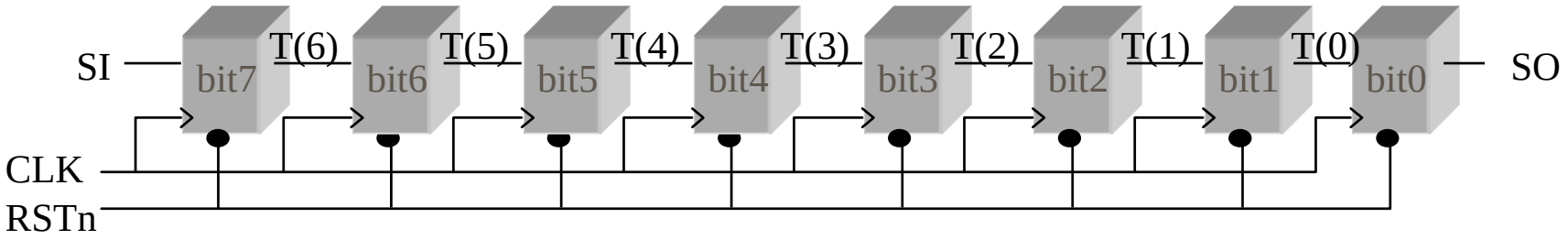
Example

```
architecture RTL of TEST is
  function BOOL2BIT (BOOL: boolean) return bit is
  begin
    if BOOL then return '1'; else return '0'; end if;
  end BOOL2BIT;

  procedure EVEN_PARITY (
       signal D: in bit_vector(7 downto 0);
       signal PARITY : out bit ) is
    variable temp : bit;
  begin
    ....
  end;

  signal DIN : bit_vector(7 downto 0);
  signal BOOL1 : boolean;
  signal BIT1, PARITY : bit;
begin
  do_it: process (BOOL1, DIN)
  begin
    BIT1 <= BOOL2BIT(BOOL1);
    EVEN_PARITY(DIN, PARITY);
  end process;
  ....
end;
```

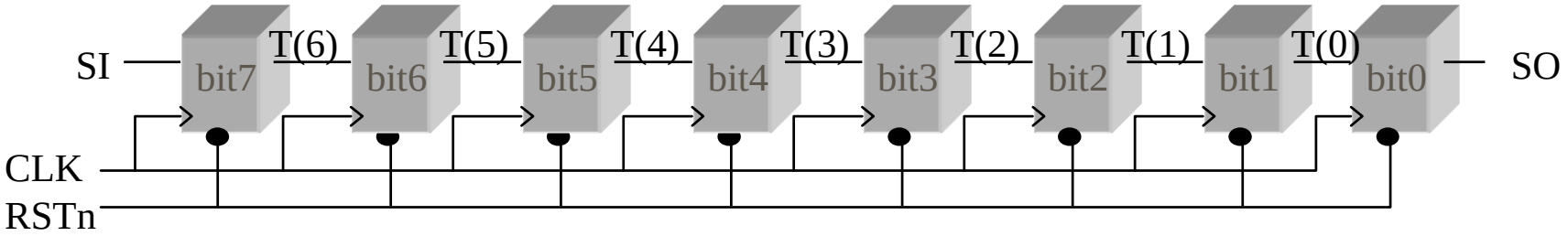
Parameterized hardware

- Conditional component instantiation with **if … generate** construct.
- Iterative component instantiation with **for … generate** construct.
- Parameterized design with **generic** parameters.

Example: 8-bit shift register




```
entity SHIFT8 is
port ( RSTn, CLK, SI : in std_logic;
       SO : out std_logic );
end SHIFT8;
```




```
architecture RTL1 of SHIFT8 is

  component DFF
  port ( RSTn, CLK, D: in std_logic;
         Q           : out std_logic );
  end component;
  signal T: std_logic_vector(6 downto 0);

begin

  bit7 : DFF
       port map (RSTn => RSTn, CLK => CLK,
                 D => SI, Q => T(6) );
  bit6 : DFF
       port map (RSTn => RSTn, CLK => CLK,
                 D => T(6), Q => T(5) );
  bit5 : DFF
       port map (RSTn, CLK, T(5), T(4) );
  ...
  bit1 : DFF
       port map (RSTn, CLK, T(1), T(0) );
  bit0 : DFF
       port map (RSTn, CLK, T(0), S0 );

end RTL1;
```

Example: 1024-bit shift register

```
architecture RTL2 of SHIFT1024 is
    component DFF
    port ( RSTn, CLK, D: in std_logic;
           Q           : out std_logic );
    end component;
    signal T: std_logic_vector(1022 downto 0);
begin
    g0: for i in 1023 downto 0 generate
        g1: if (i = 1023) generate
              bit1023 : DFF port map (RSTn,CLK,SI,T(1022));
            end generate;
        g2: if (i>0) and (i<1023) generate
              bitm : DFF port map (RSTn,CLK,T(i),T(i-1));
            end generate;
        g3: if (i=0) generate
              bit0 : DFF port map (RSTn,CLK,T(0),S0);
            end generate;
      end generate;
end RTL2;
```

Example: n-bit shift register

```
entity SHIFTn is
generic ( n : positive);
port ( RSTn, CLK, SI : in std_logic;
       SO : out std_logic );
end SHIFTn;
```

```
architecture RTL3 of SHIFTn is
  component DFF
  port ( RSTn, CLK, D: in std_logic;
         Q           : out std_logic );
  end component;
  signal T: std_logic_vector(n-2 downto 0);
begin
  g0: for i in n-1 downto 0 generate
      g1: if (i = n-1) generate
            bit_high : DFF port map (RSTn,CLK,SI,T(n-2));
          end generate;
      g2: if (i>0) and (i<n-1) generate
            bitm : DFF port map (RSTn,CLK,T(i),T(i-1));
          end generate;
      g3: if (i=0) generate
            bit0 : DFF port map (RSTn,CLK,T(0),S0);
          end generate;
    end generate;
end RTL3;
```

Example: n-bit shift register

- Component instantiation

```
…
  component SHIFTn is
  generic ( n : positive);
  port ( RSTn, CLK, SI : in std_logic;
           SO : out std_logic );
  end component;
```

```
…
begin
  …
  Shift32comp : SHIFTn
    generic map (n => 32)
    port map(RSTn => …,
                  CLK => …,
                  SI => …,
                  SO => …);
  …
end;
```

VHDL: Evaluation

- Hierarchical specification by entities / architectures / components, (procedures and functions)
- no nested processes
- Static number of processes
- Complicated simulation semantics
- May be too low level for initial, abstract specification of very large systems
- Mainly used for hardware simulation+synthesis

REVIEW: computational models

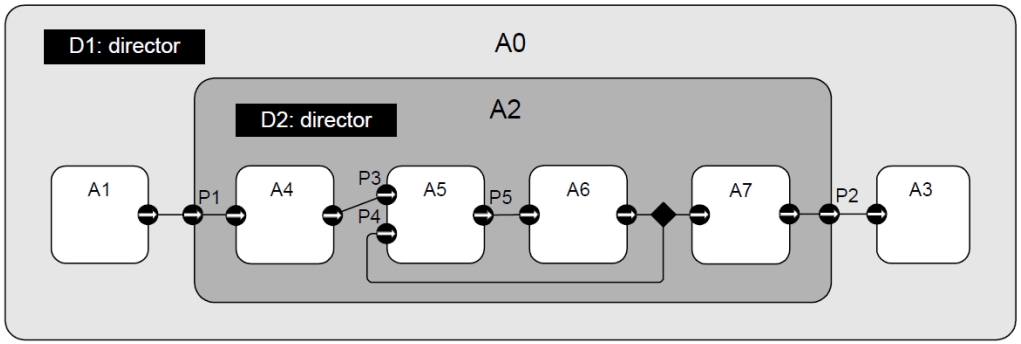
Communication/ local computations	Shared memory	Asynchronous message passing
Communicating finite state machines	Statecharts, hybrid automata, synchronous composition	
Data flow		Petri nets, Kahn process networks, SDF
Discrete event (DE) model	Simulink, VHDL	Distributed DE

Ptolemy

- Ptolemy (UC Berkeley) is an environment for simulating multiple models of computation.

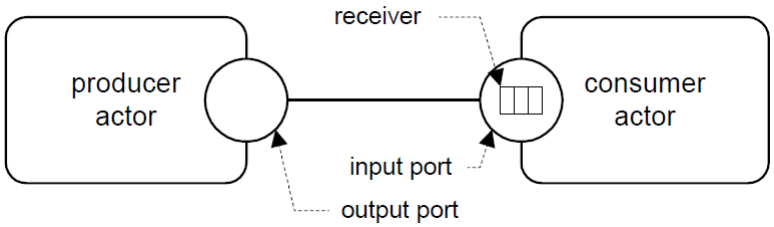
  http://ptolemy.berkeley.edu/

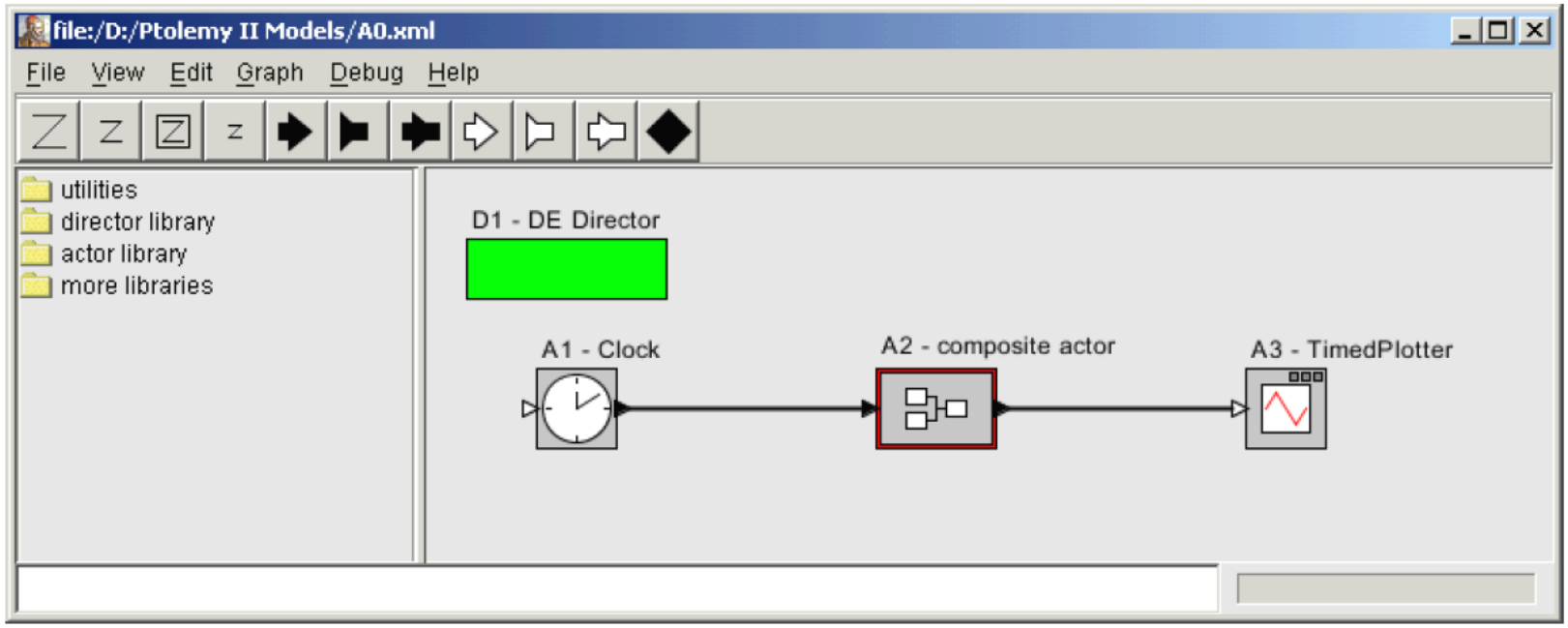
  - discrete-event systems
  - SDF
  - process networks
  - Petri nets
  - priority-based schedules
  - synchronous/reactive
  - Finite-state machines
  - continuous-time
  - modal systems
  - Graphics, 3D animations

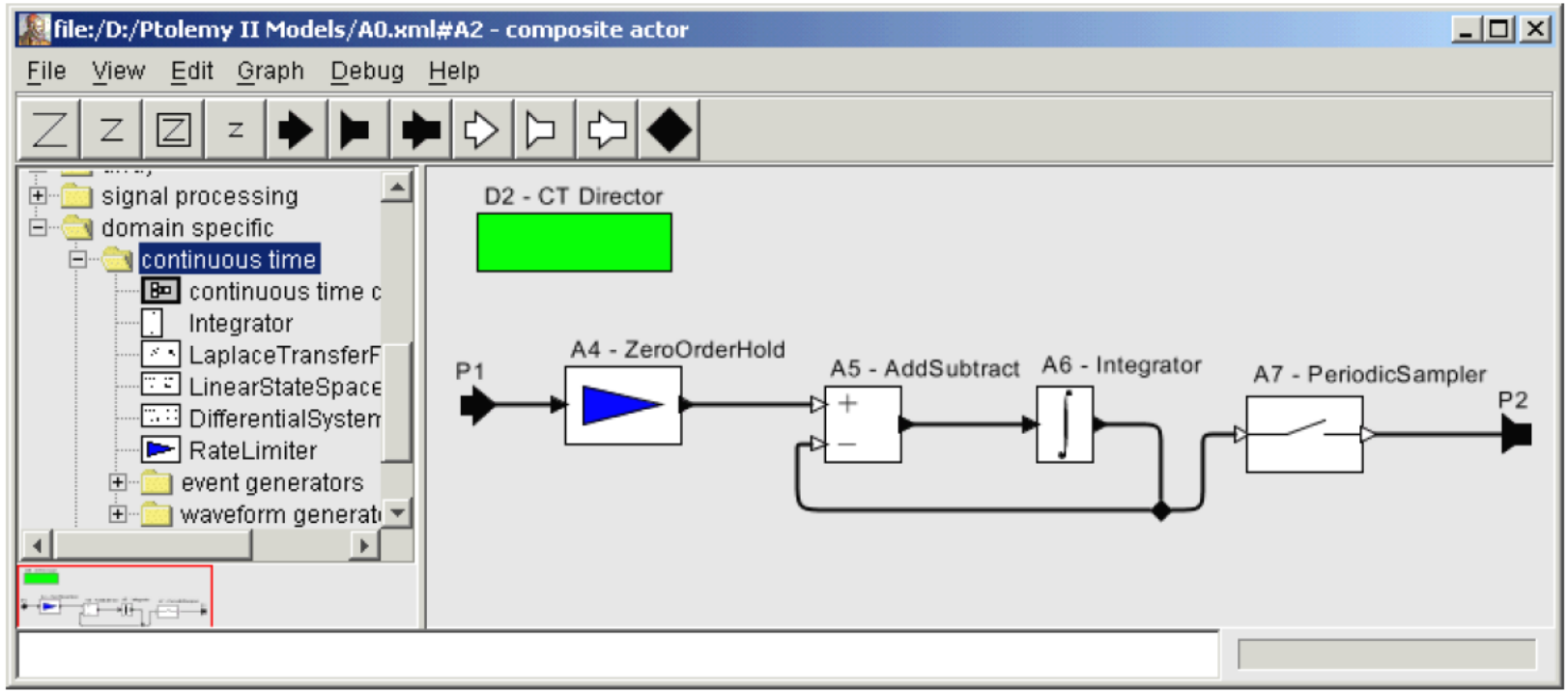


Ptolemy

- A model is a set of interconnected *actors* and one *director*
- Actor
  - Input & output *ports*, states, & parameters
- Models of computation
  - Define the interaction semantics
  - Implemented in Ptolemy II by a *domain*
    - Director + Receiver
- Director
  - Manages the data flow and the scheduling of the actors
  - The director fires the actors
- Receiver
  - Defines the semantics of the port buffers




file:/D:/Ptolemy II Models/A0.xml
File View Edit Graph Debug Help
utilities
director library
actor library
more libraries
D1 - DE Director
A1 - Clock
A2 - composite actor
A3 - TimedPlotter


file:/D:/Ptolemy II Models/A0.xml#A2 - composite actor
File View Edit Graph Debug Help
signal processing
domain specific
continuous time
Integrator
LaplaceTransferF
LinearStateSpace
DifferentialSystem
RateLimiter
event generators
D2 - CT Director
P1
A4 - ZeroOrderHold
A5 - AddSubtract
A6 - Integrator
A7 - PeriodicSampler
P2

Example: Inverted Pendulum

- Classic control problem
- Swing up the pendulum and then keep it in the upright position

Heterogeneous Modeling and Design of Control Systems, Liu/Liu/Eker/Lee, 2003

The Ptolemy II Model

director

CT

FurutaPendulur

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

y=g(x, u, t)

continuous process

atomic actor

composite actor

Sampler

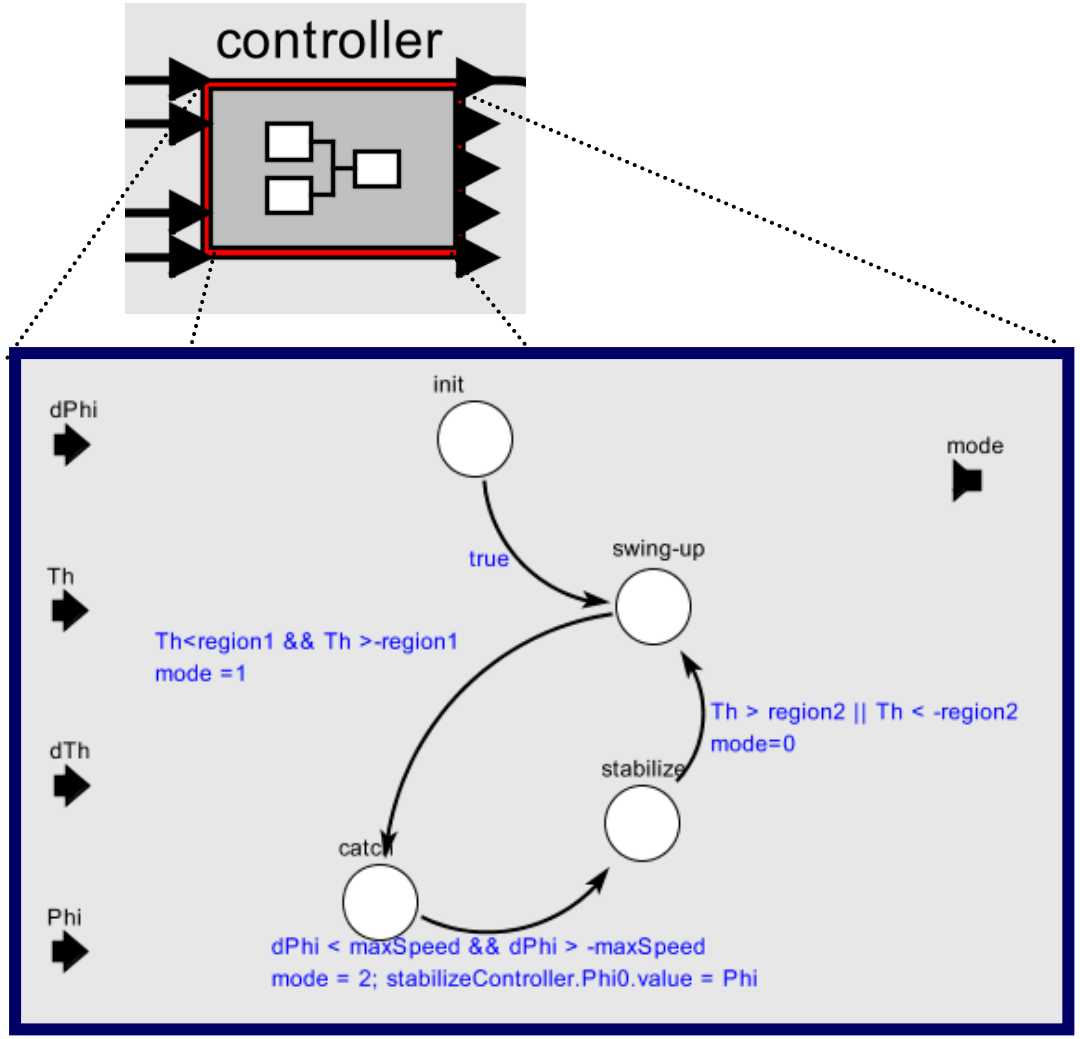
controller

ZOH

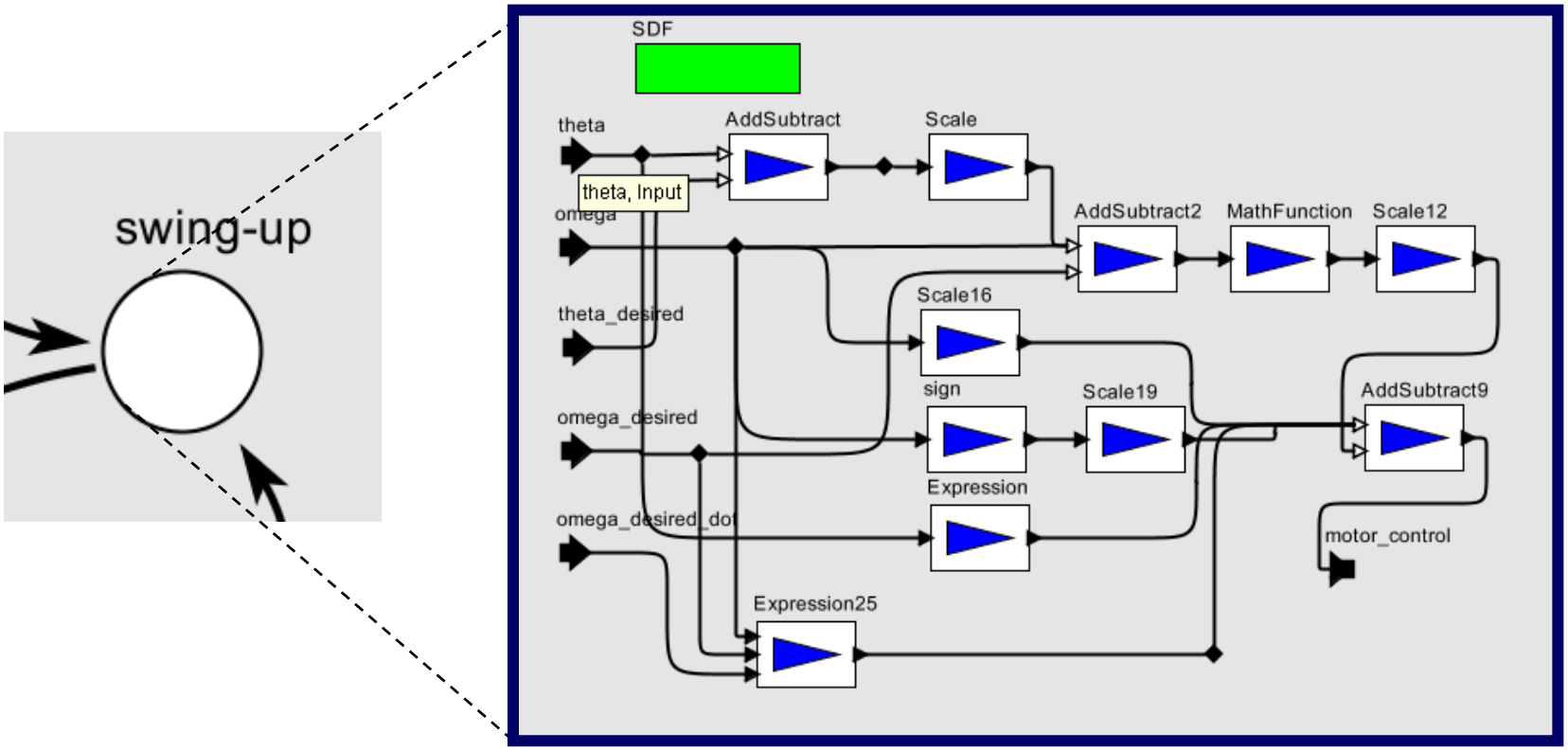
model

discrete controller

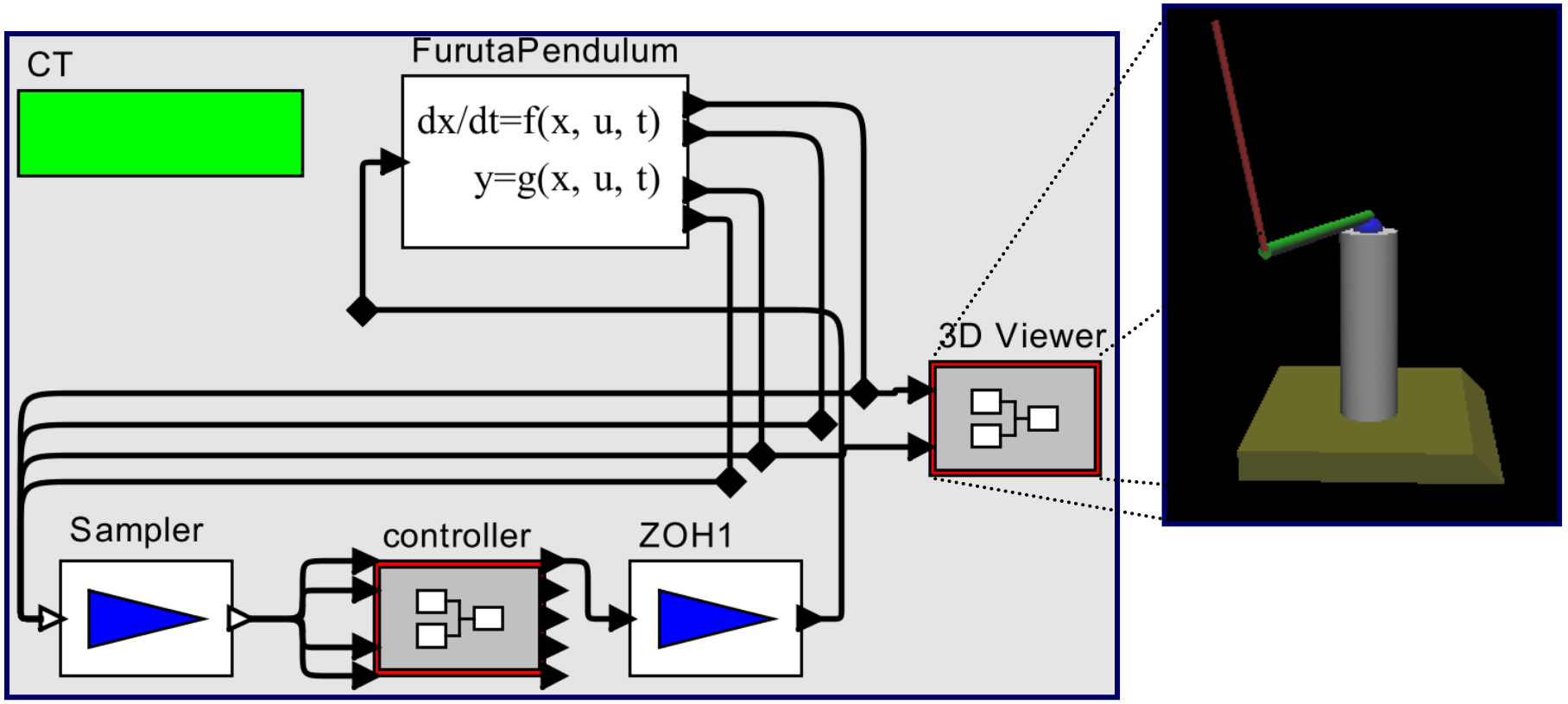
Controller Logic in FSM - Finite State Machine



Subcontrollers in SDF - Synchronous Data flow



Visualization in GR - Graphics Domain

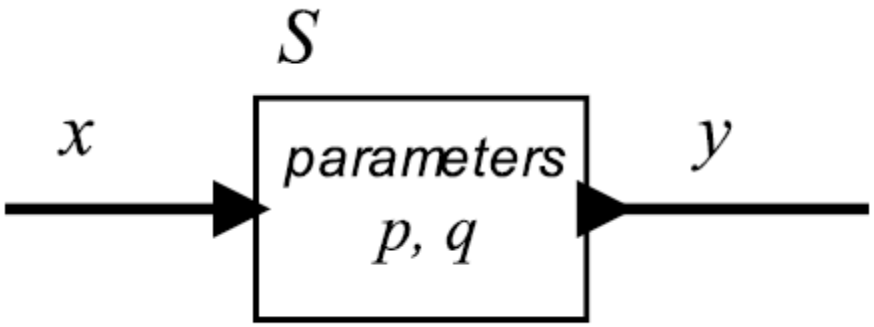


Actor model

- Motivation
  - Combining components into larger systems
  - Combining different computational models
- Major points
  - Continuous actors, discrete actors
  - Ptolemy
  - Virtual Prototyping ($\rightarrow$ Matlab/Simulink/Stateflow)

Actor Model of Continuous-Time Systems

A *system* is a function that accepts an input *signal* and yields an output signal.



The domain and range of the system function are sets of signals, which themselves are functions.

$$x: \mathbb{R} \to \mathbb{R}, \quad y: \mathbb{R} \to \mathbb{R}$$

$$S: X \to Y$$

$$X = Y = (\mathbb{R} \to \mathbb{R})$$

Parameters may affect the definition of the function *S*.

Discrete Signals

Let *e* be a signal $\mathbb{R} \rightarrow \{absent\} \cup X$
where *X* is any set of values.

Let $T = \{t \in \mathbb{R} : e(t) \neq absent\}$

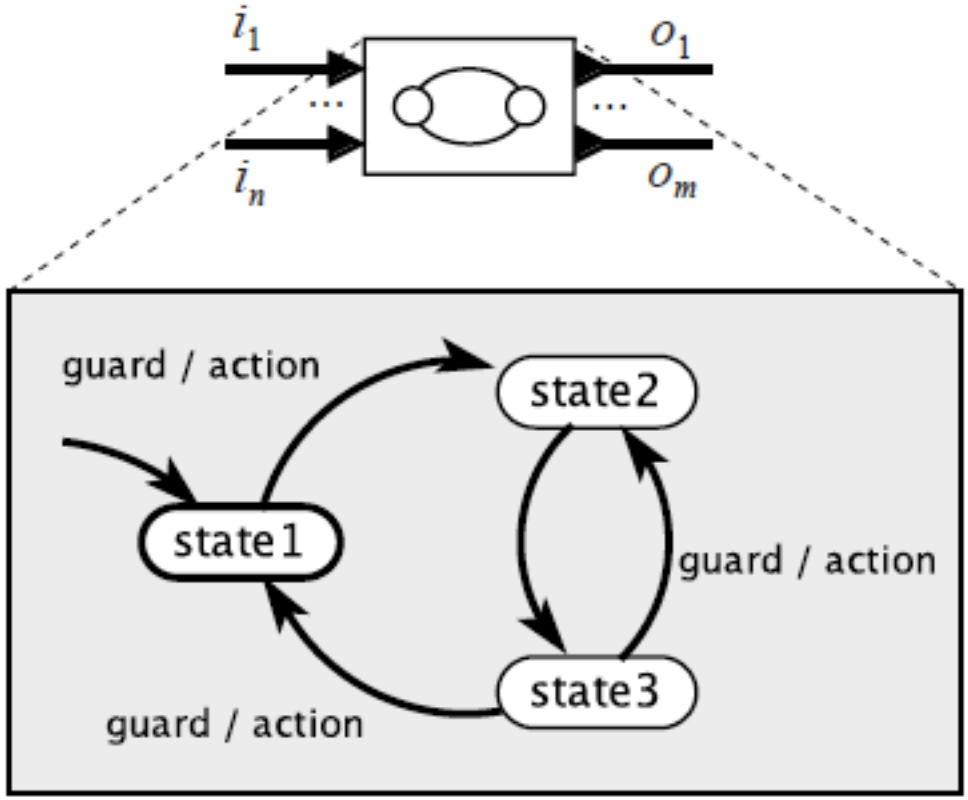
Then *e* is **discrete** iff there exists a one-to-one function

$$f : T \rightarrow \mathbb{N}$$

that is order-preserving, i.e., for all $t_1 \leq t_2$, $f(t_1) \leq f(t_2)$.

Actor Model for State Machines

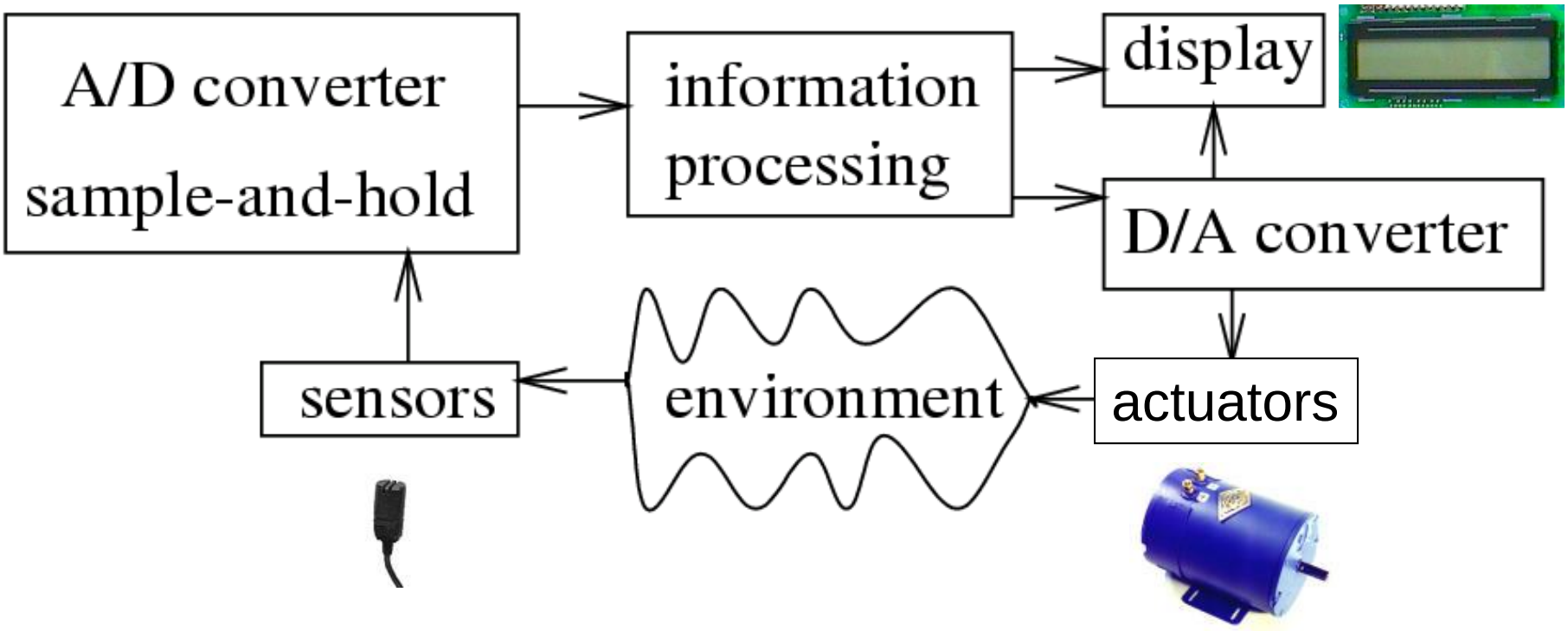
Expose inputs and outputs, enabling composition:



Sensors & Actuators

Embedded System Hardware

- Embedded system hardware is frequently used in a loop (***„hardware in a loop“***):



Sensors and Actuators

Sensors:

- Cameras
- Accelerometers
- Rate gyros
- Strain gauges
- Microphones
- Magnetometers
- Radar/Lidar
- Chemical sensors
- Pressure sensors

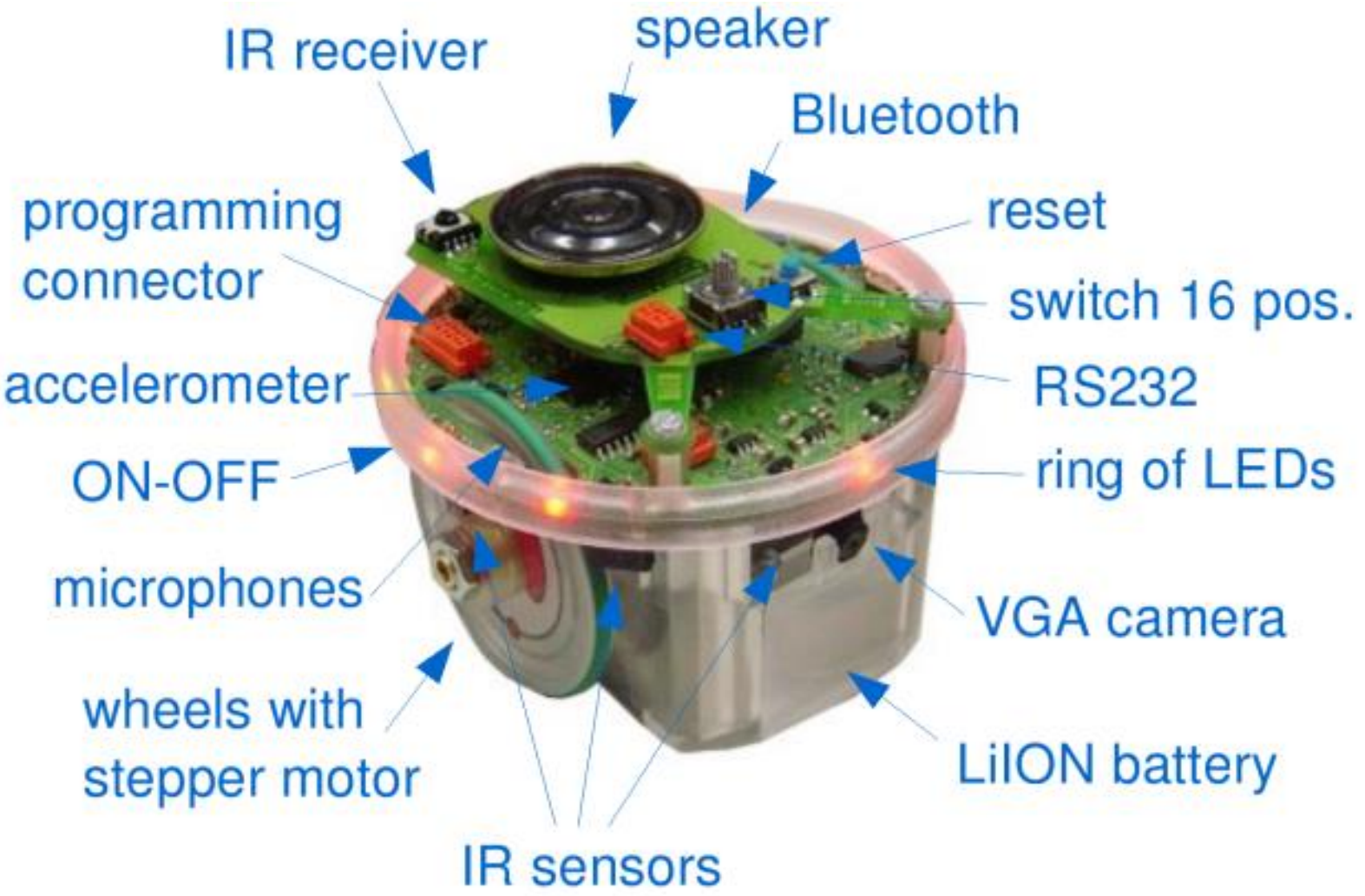
Actuators:

- Motor controllers
- Solenoids
- LEDs, lasers
- LCD and plasma displays
- Loudspeakers

Modeling Issues:

- Physical dynamics
- Noise
- Bias
- Sampling
- Interactions

E-puck

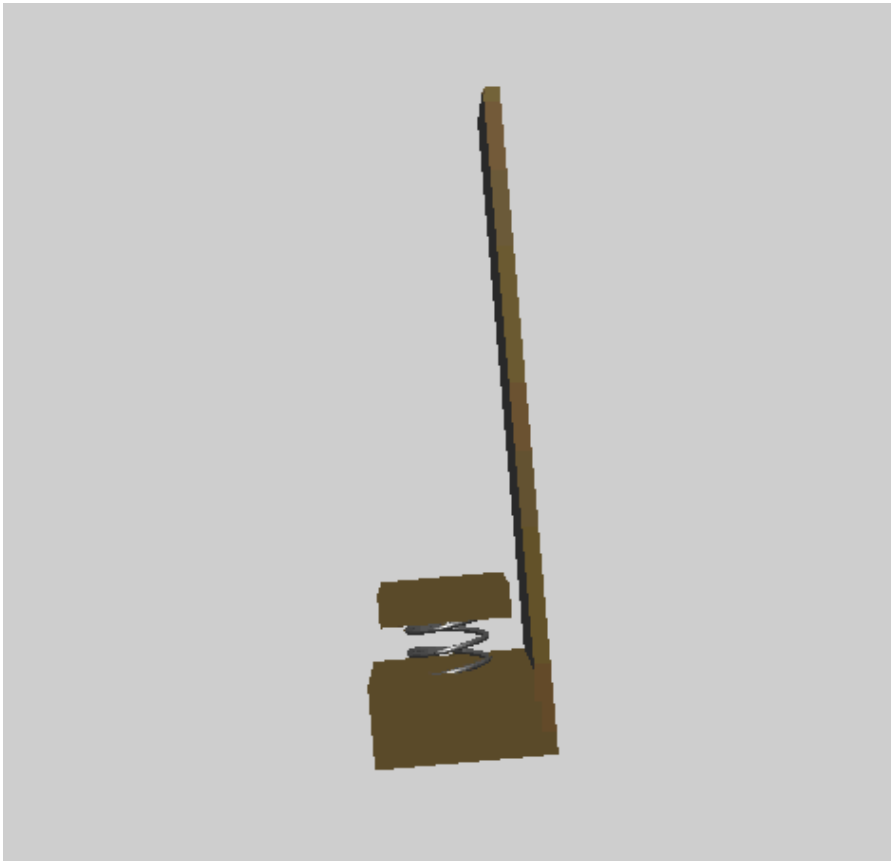

IR receiver
speaker
Bluetooth
programming connector
reset
switch 16 pos.
accelerometer
RS232
ON-OFF
ring of LEDs
microphones
VGA camera
wheels with stepper motor
LiION battery
IR sensors

Spring-Mass-Damper Accelerometer

By Newton's second law, F=ma.

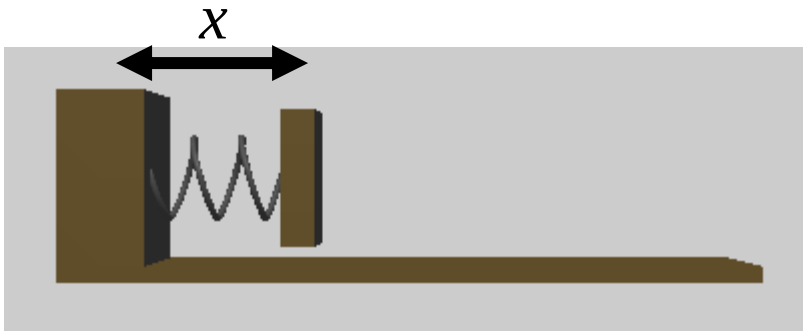
For example, F could be the earth's gravitational force.

The force is balanced by the restoring force of the spring.



Spring-Mass-Damper System

- mass: M
- spring constant: k
- spring rest position: p
- position of mass: x
- viscous damping constant: c



Force due to spring extension:

$$F_1(t) = k(p - x(t))$$

Force due to viscous damping:

$$F_2(t) = -c\dot{x}(t)$$

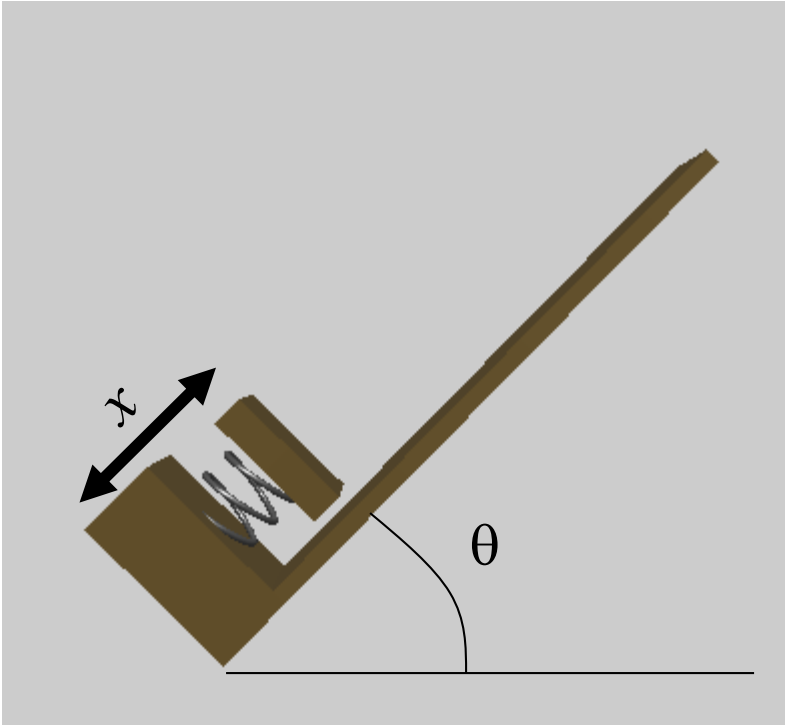
Newton's second law:

$$F_1(t) + F_2(t) = M\ddot{x}(t)$$

or

$$M\ddot{x}(t) + c\dot{x}(t) + kx(t) = kp.$$

Measuring tilt



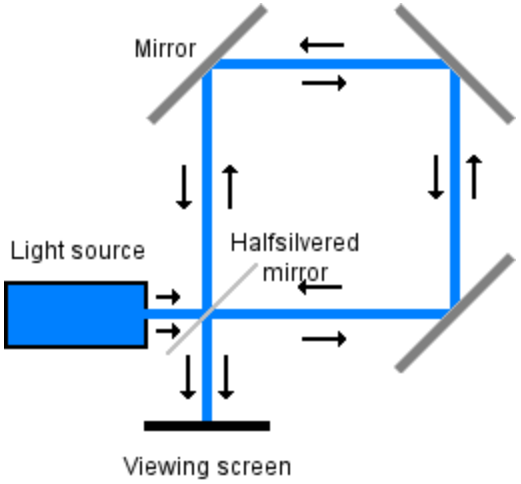
Component of gravitational force in the direction of the accelerometer axis must equal the spring force:

$$Mg\sin(\theta) = k(p - x(t))$$

Difficulties Using Accelerometers

- Separating tilt from acceleration
- Integrating twice to get position: Drift
- Vibration
- Nonlinearities in the spring or damper

Measuring Changes in Orientation: Gyroscopes



Optical gyros: Leverage the Sagnac effect, where a laser light is sent around a loop in opposite directions and the interference is measured. When the loop is rotating, the distance the light travels in one direction is smaller than the distance in the other. This shows up as a change in the interference.

Images from the Wikipedia Commons

Inertial Navigation Systems

Combinations of:

- GPS (for initialization and periodic correction)
- Three axis gyroscope measures orientation
- Three axis accelerometer, double integrated for position after correction for orientation

Magnetometers

- A very common type is the Hall Effect magnetometer.
- Charge particles (electrons) flow through a conductor (2) serving as a Hall sensor. Magnets (3) induce a magnetic field (4) that causes the charged particles to accumulate on one side of the Hall sensor, inducing a measurable voltage difference from top to bottom.
- The four drawings at the right illustrate electron paths under different current and magnetic field polarities.

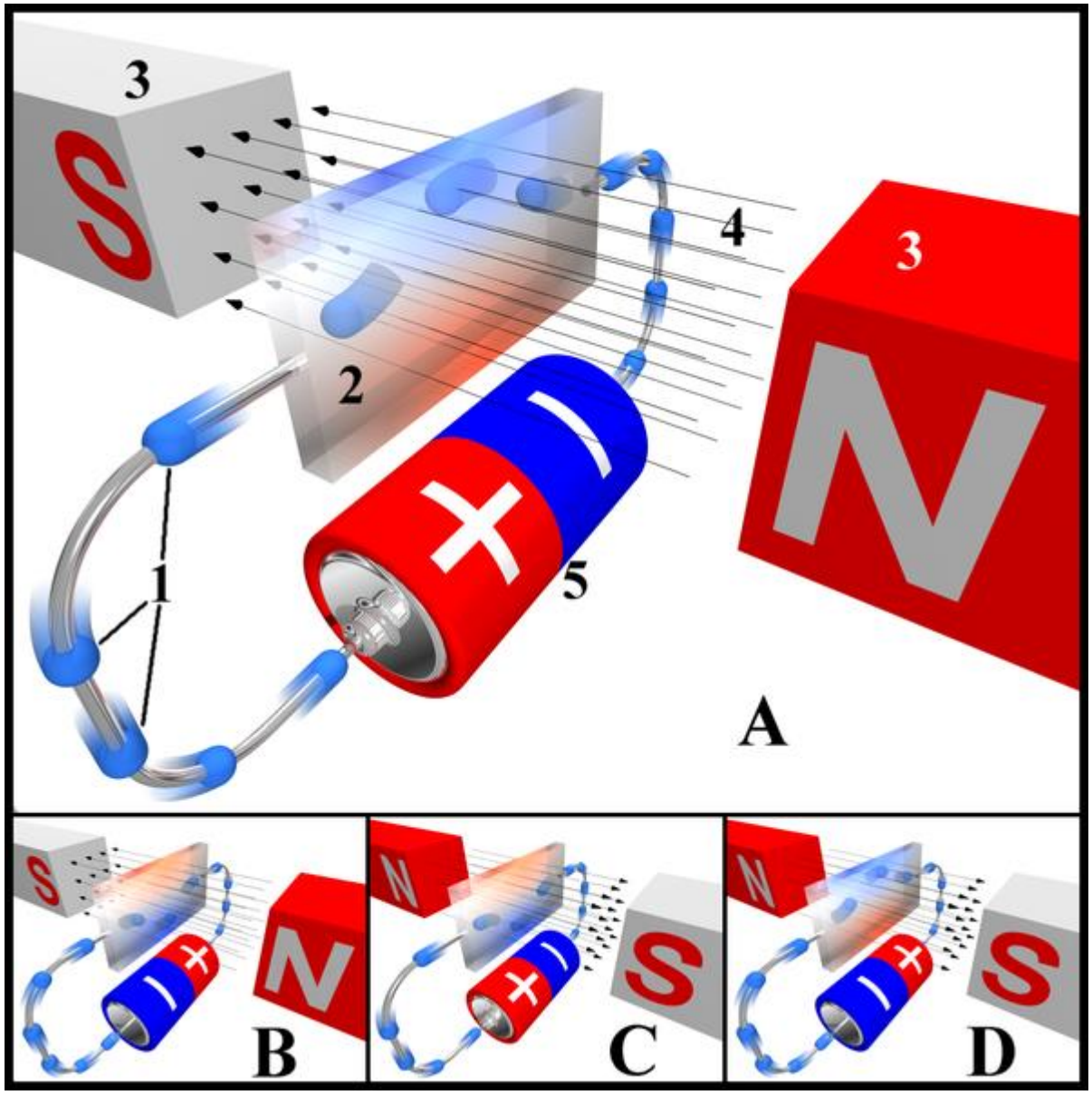
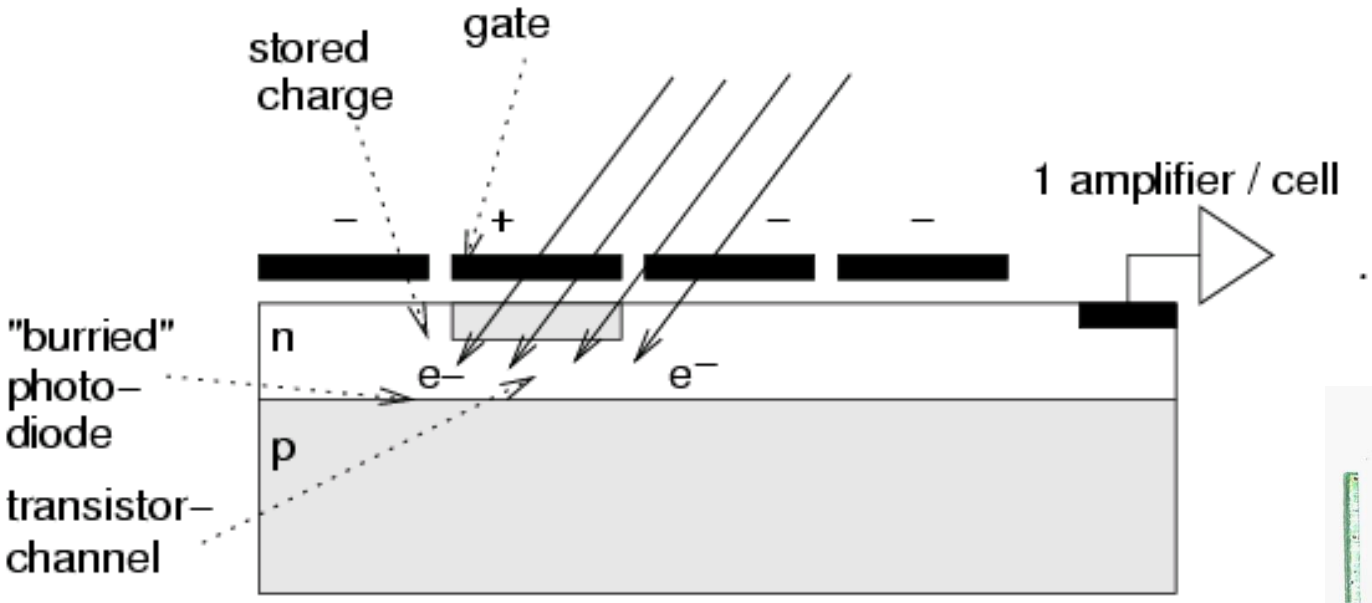


Image source: Wikipedia Commons

Edwin Hall discovered this effect in 1879.

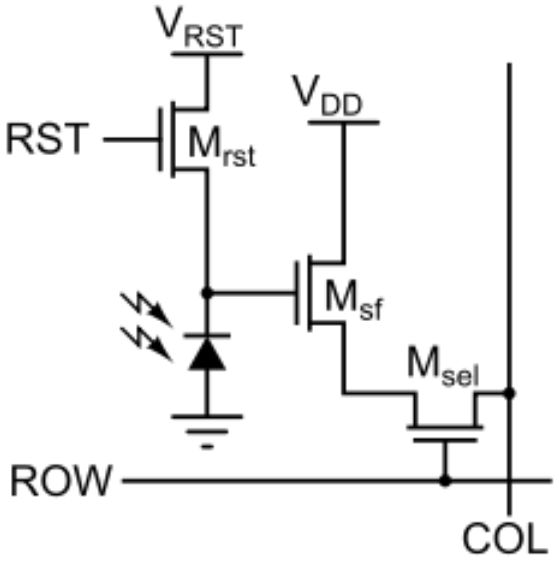
Charge-coupled devices (CCD) image sensors

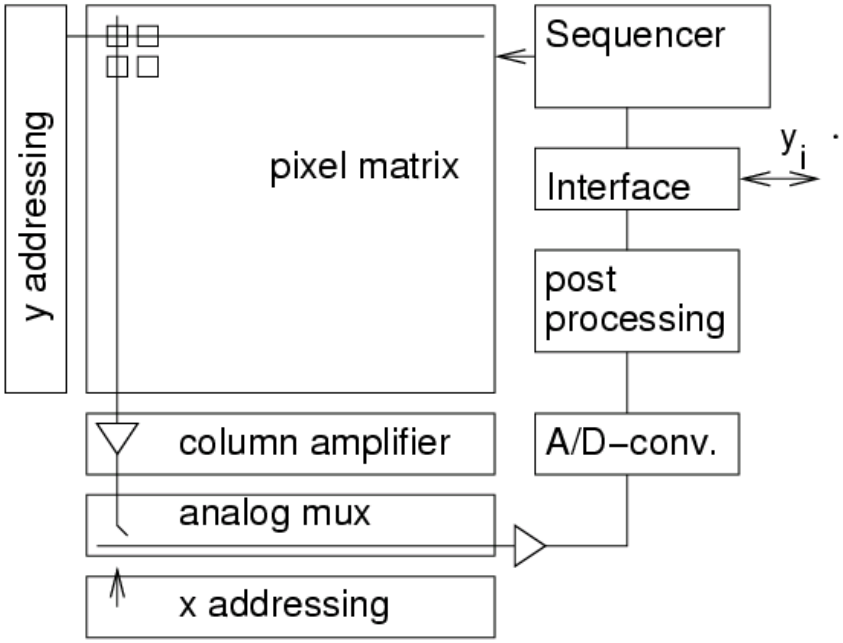
Based on charge transfer to next pixel cell



- Mature technology
- Medium to high-end compact digital cameras

CMOS image sensors





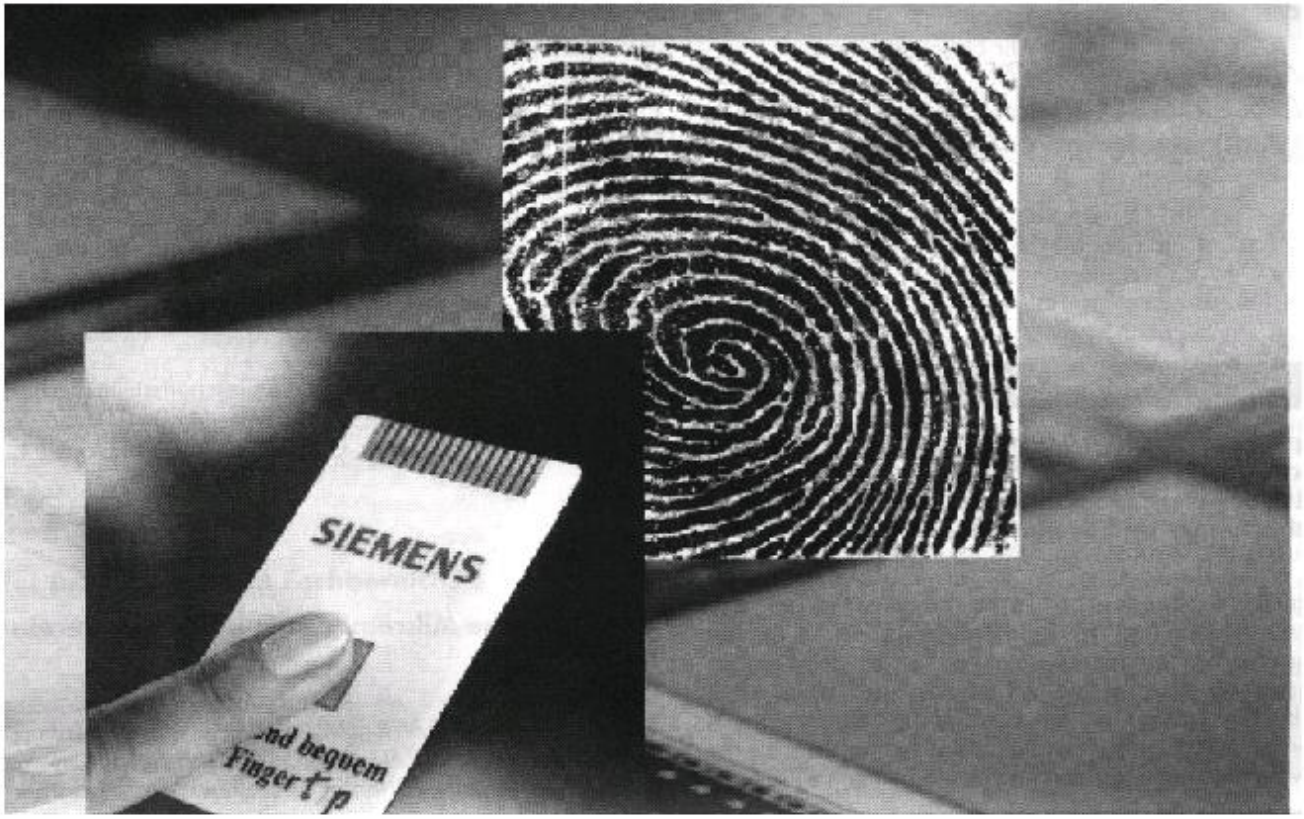
- Lower power consumption
- Lower cost

- Based on standard production process for CMOS chips, allows integration with other components

- low cost devices
- Automotive
- medical

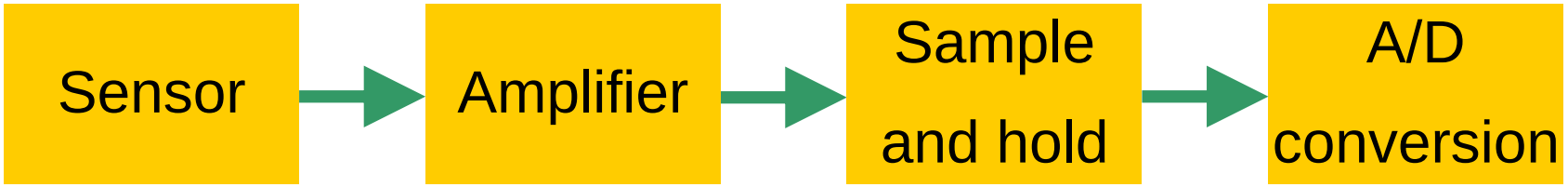
Example: Biometrical Sensors

Example: Fingerprint sensor (© Siemens, VDE):



Matrix of 256 x 256 elem. Voltage ~ distance. Resistance also computed

Standard layout of sensor systems



- Sensor: detects/measures entity and converts it to electrical domain
  - May entail ES-controllable actuation: e.g. charge transfer in CCD
- Amplifier: adjusts signal to the dynamic range of the A/D conversion
  - Often dynamically adjustable gain: e.g. ISO settings at digital cameras, input gain for microphones (sound or ultrasound), extremely wide dynamic ranges in seismic data logging
- Sample + hold: samples signal at discrete time instants
- A/D conversion: converts samples to digital domain