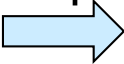


# Embedded Systems

9



# Overview: computational models

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

# REVIEW: SDF Scheduling Algorithm

## Lee/Messerschmitt 1987

### 1. Establish **relative execution rates**

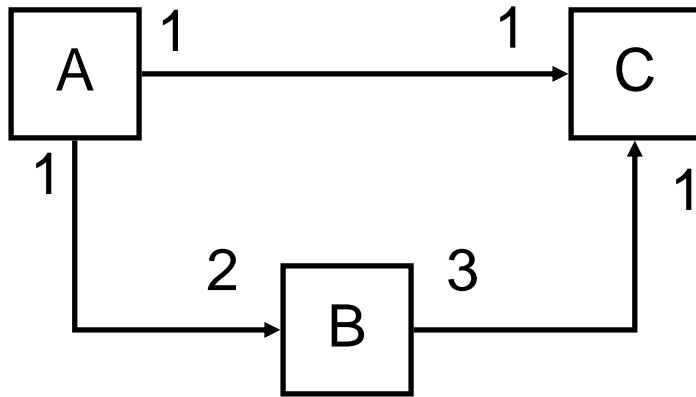
- Generate balance equations
- Solve for smallest positive integer vector  $\mathbf{c}$

### 2. Determine **periodic schedule**

- Form an arbitrarily ordered list of all nodes in the system
- Repeat:
  - For each node in the list, schedule it if it is runnable, trying each node once
  - If each node has been scheduled  $\mathbf{c}_n$  times, stop.
  - If no node can be scheduled, indicate deadlock.

Source: Lee/Messerschmitt, Synchronous Data Flow (1987)

# REVIEW: An inconsistent system



$$a - c = 0$$

$$a - 2b = 0$$

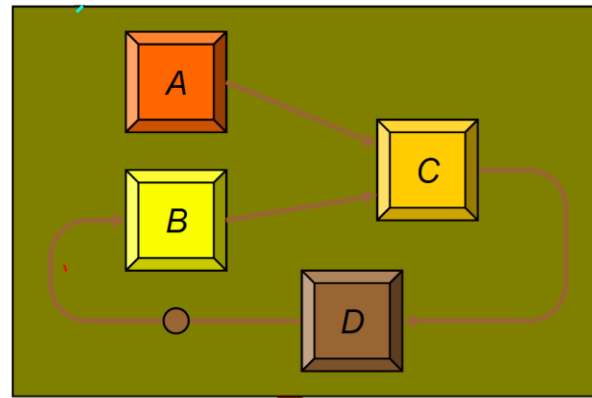
$$3b - c = 0$$

$$3a - 2c = 0$$

- No way to execute without an unbounded accumulation of tokens
- Only consistent solution is „do nothing“

# REVIEW: Scheduling SDF models

SDF is suitable for automated mapping onto parallel processors and synthesis of parallel circuits.



**Sequential**

periodic admissible sequential schedule (PASS)

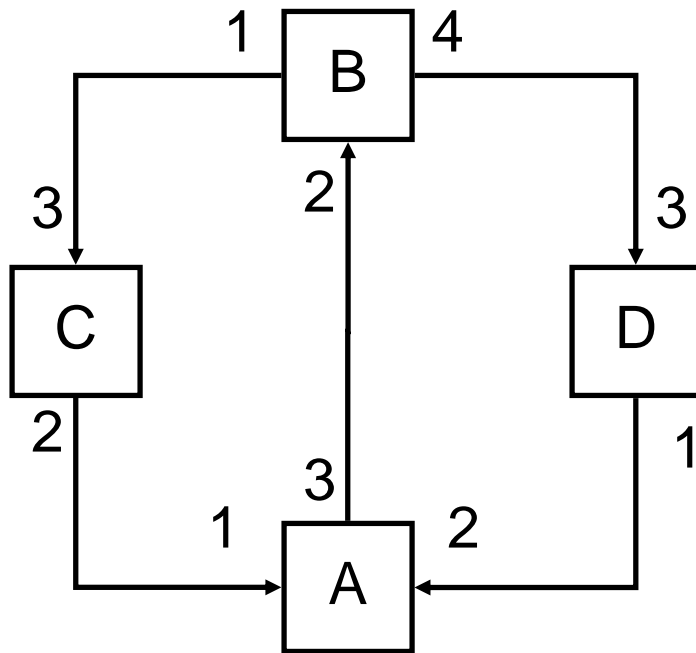


**Parallel**

periodic admissible parallel schedule (PAPS)

**(admissible = correct schedule, finite amount of memory required)**

# PASS example: 1) firing rates



$$d(AB)=6$$

$$3a - 2b = 0$$

$$4b - 3d = 0$$

$$b - 3c = 0$$

$$2c - a = 0$$

$$d - 2a = 0$$

Solution:

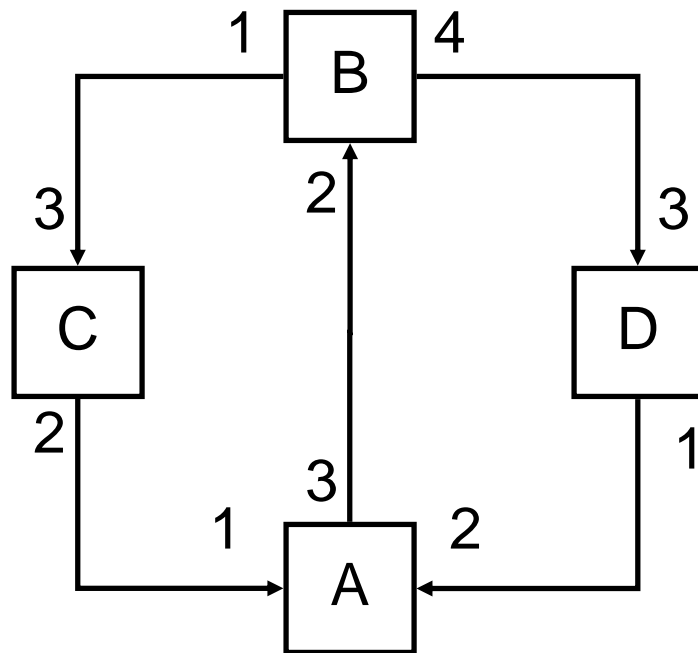
$$a = 2c$$

$$b = 3c$$

$$d = 4c$$

Smallest solution:  $a=2$ ;  $b=3$ ;  $d=4$ ;  $c=1$

## REVIEW: example: 2) Simulation



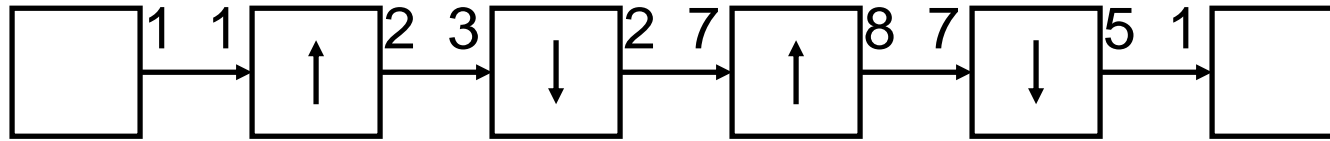
$$d(AB)=6$$

Smallest solution:  
 $a=2; b=3; d=4; c=1$

Possible schedules:  
BBBCDDDDAA  
BDBDBCADDA  
BBDDDBDDCAA  
(and many more)

BC... not valid

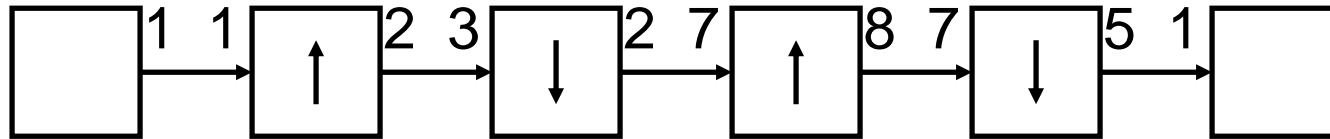
# CD-to-DAT rate converter



ABABCABCABABCABCDEAFFFFFBABABCABCABABCDE  
 AFFFFFBCABABCABCABABCDEAFFFFFBCABABCABC  
 DEAFFFFFBCABABCABCABABCDEAFFFFFBCABABC  
 BABABCDEAFFFFFBCABABCABCABABCDEAFFFFFBCA  
 FFFFBCABABCABCDEAFFFFFBCABABCABCABCDEAF  
 FFFFBCABABCABCABABCDEAFFFFFBCABABCABCABC  
 DEAFFFFFBCABABCABCDEAFFFFFBCABABCABCABC  
 BCDEAFFFFFBCABABCABCABABCDEAFFFFFBCAFFFFF  
 ABCABCABCABABCDEAFFFFFBCABABCABCDEAFFFFF  
 BCABCABCABCABCDEAFFFFFBCABABCABCABCDEAFF  
 FFBCABCABCABCABCDEAFFFFFBCABCABCABCDEAF  
 FFFFBCABCABCABCABCDEAFFFFFBCAFFFFFBCABC  
 ABABCDEAFFFFFBCABCABCABCABCDEAFFFFFBCA  
 BABCABCDEAFFFFFBCABCABCABCABCDEAFFFFFBC  
 ABCABCABCABCDEAFFFFFBCABCABCABCABCDEAF  
 FFFFBCABCABCABCDEFFFFFEFFFFF

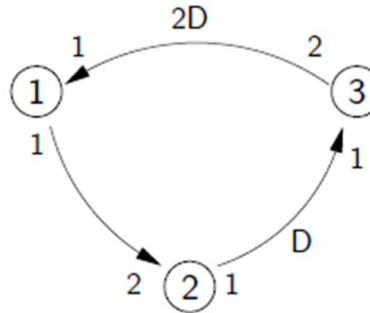


# CD-to-DAT rate converter

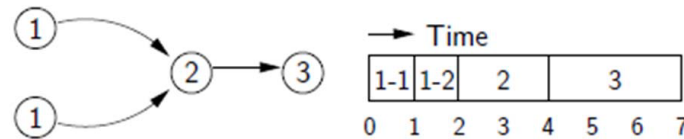


Scheduling strategy	Code	Data
Minimum buffer schedule, no looping	13735	32
Minimum buffer schedule, with looping	9400	32
Worst minimum code size schedule	170	1021
Best minimum code size schedule	170	264

# Periodic admissible parallel schedules (PAPS)

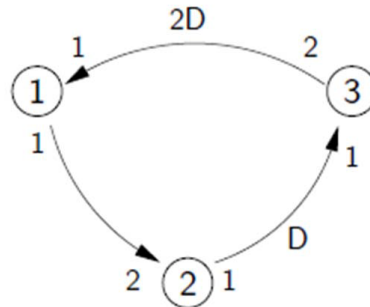


Assumption: Block 1 : 1 time unit  
 Block 2 : 2 time units  
 Block 3 : 3 time units

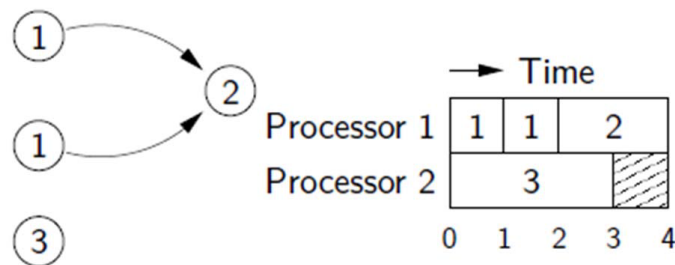


Trivial Case - All computations are scheduled on same processor

# Periodic admissible parallel schedules (PAPS)

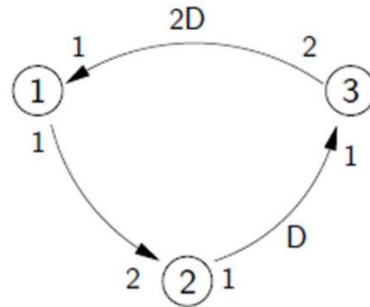


The performance can be improved, if a schedule is constructed that exploits the potential parallelism in the SDF-graph. Here the schedule covers one single period.

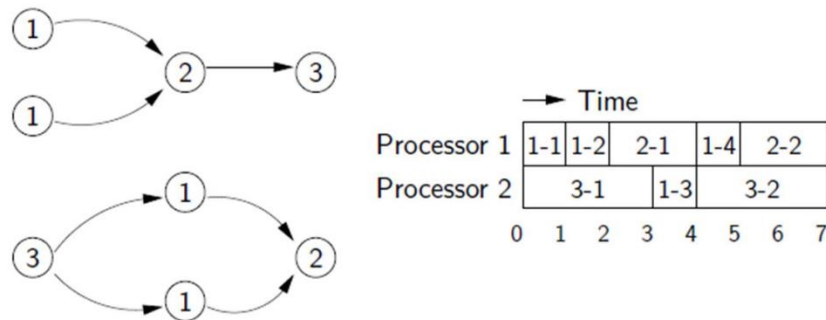


Single Period Schedule

# Periodic admissible parallel schedules (PAPS)

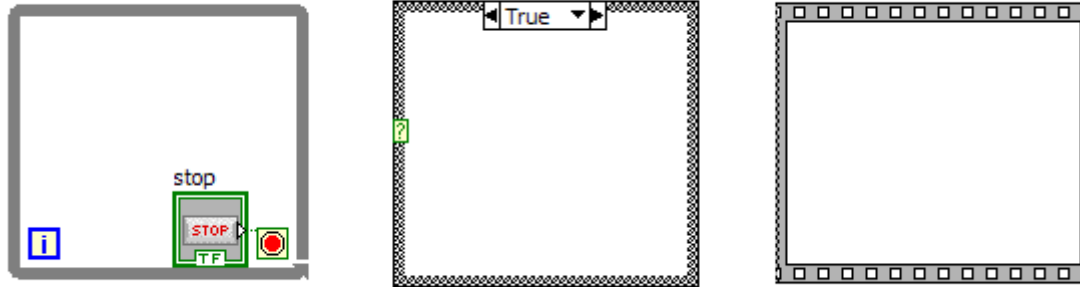


The performance can be further improved, if the schedule is constructed over two periods.



Double Period Schedule

# Variations of SDF: Structured Dataflow

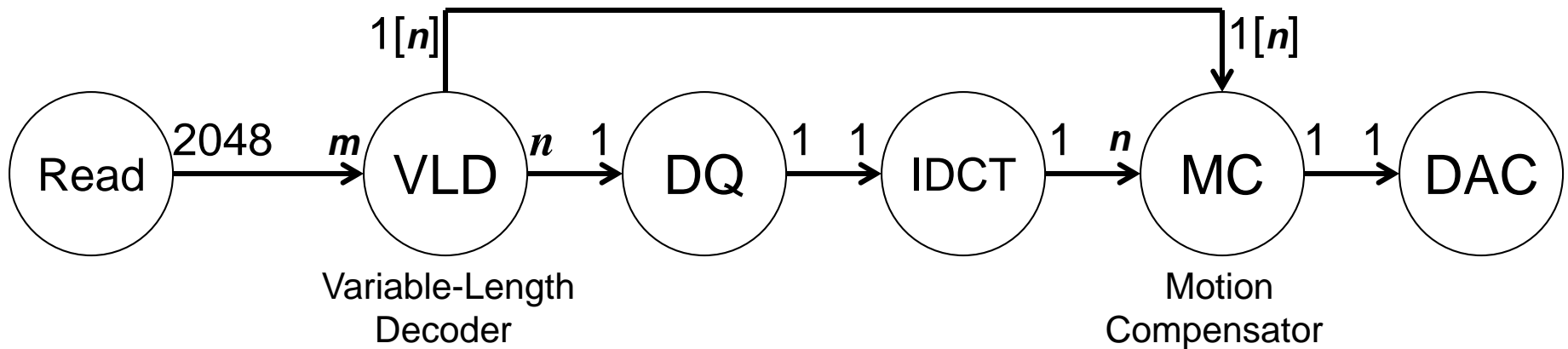


LabVIEW (National Instruments) uses homogeneous SDF augmented with syntactically constrained forms of feedback and rate changes: while loops, conditionals,...

Such structured dataflow models are decidable.

[Kodosky 86]

# Variations of SDF: Data-dependent communication H.263 video codec



Wiggers/Bekooj/Smit: Buffer Capacity Computation for Throughput Constrained Streaming Applications with Data-Dependent Inter-Task Communication, 2008

# Summary dataflow

- Communication exclusively through FIFOs
- Kahn process networks
  - Blocking read, nonblocking write
  - Deterministic
  - Schedulability undecidable
- SDF
  - Useful for DSP
  - Fixed token consumption/production
  - Compile-time scheduling: balance equations
- Decidable extensions of SDF
  - Structured Dataflow

# Discrete-event systems

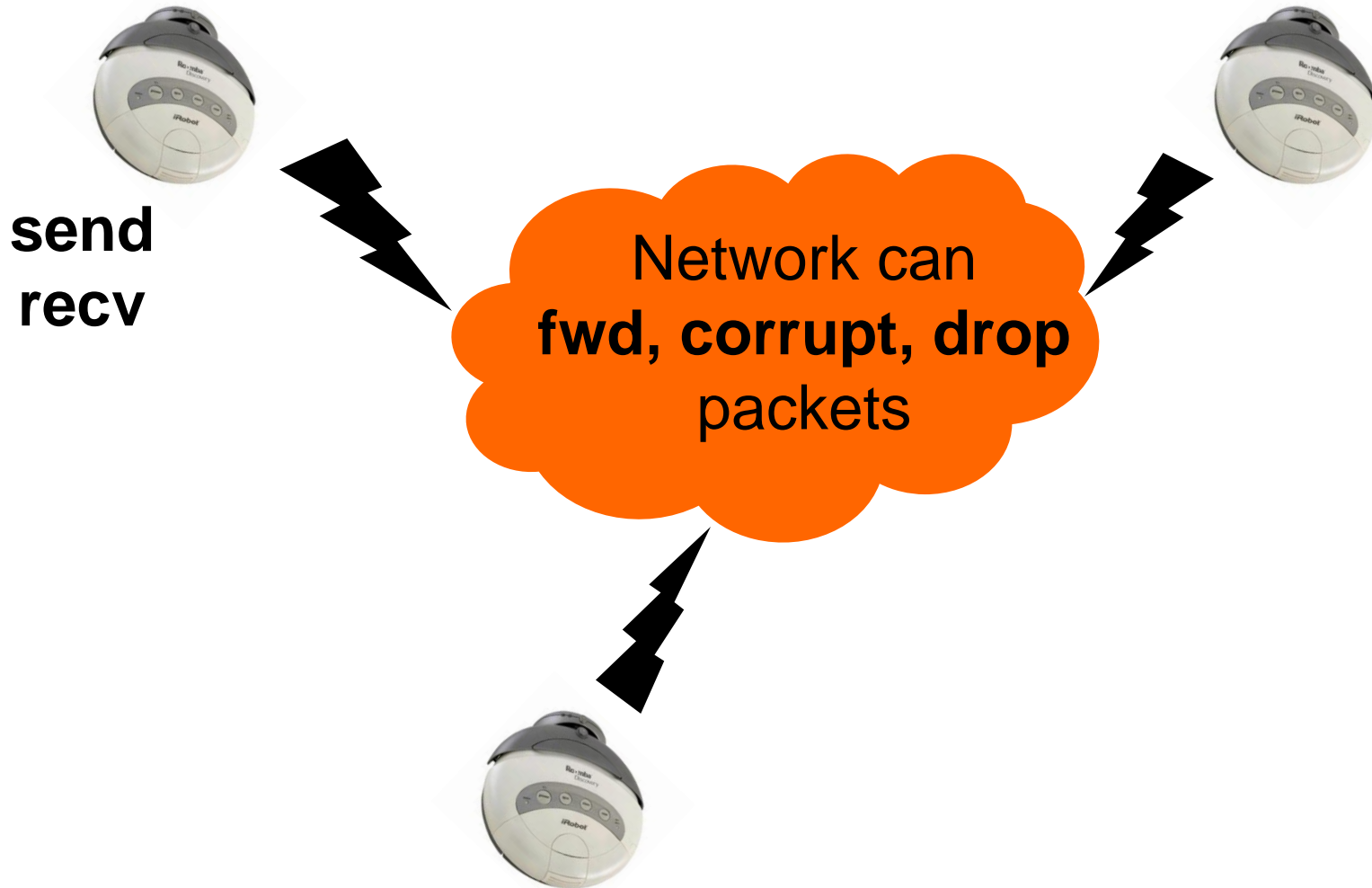
Dynamical systems whose evolution is governed by the *occurrence of events at discrete time points*, at possibly irregularly-spaced intervals

Many cyber-physical systems are modeled as discrete-event systems:

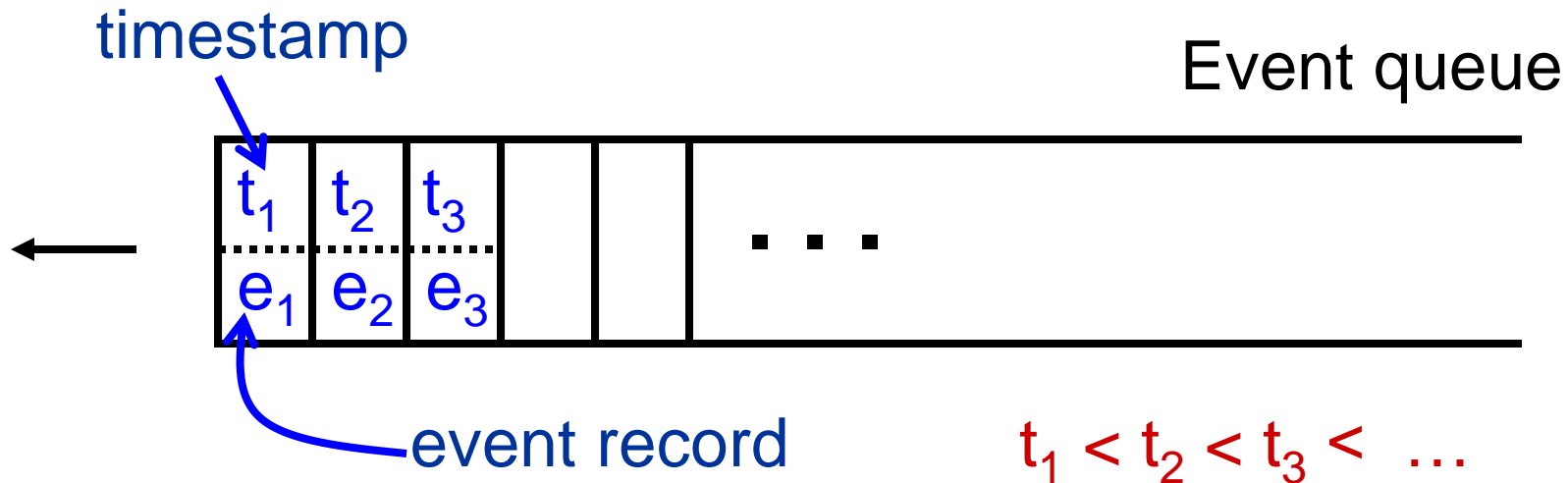
- Communication networks
- Microprocessors
- Manufacturing facilities
- Communicating robots



# Example: Communicating Robots/Sensor Nodes



# Simulating the System with an Event Queue



- Simulation Timer,  $T = 0$
- Repeat while there are events in the event queue:
  1. Dequeue event at head of queue (“imminent event”)
  2. Advance simulation timer to time of imminent event
  3. Execute imminent event: update system state
  4. Generate future events and enqueue them

# VHDL

- HDL = hardware description language
- VHDL = VHSIC hardware description language
- VHSIC = very high speed integrated circuit
  
- Initiated by US Department of Defense
- 1987 IEEE Standard 1076
- Reviews of standard: 1993, 2000, 2002, 2008
  
- Standard in (European) industry
  
- Extension: VHDL-AMS, includes analog modeling

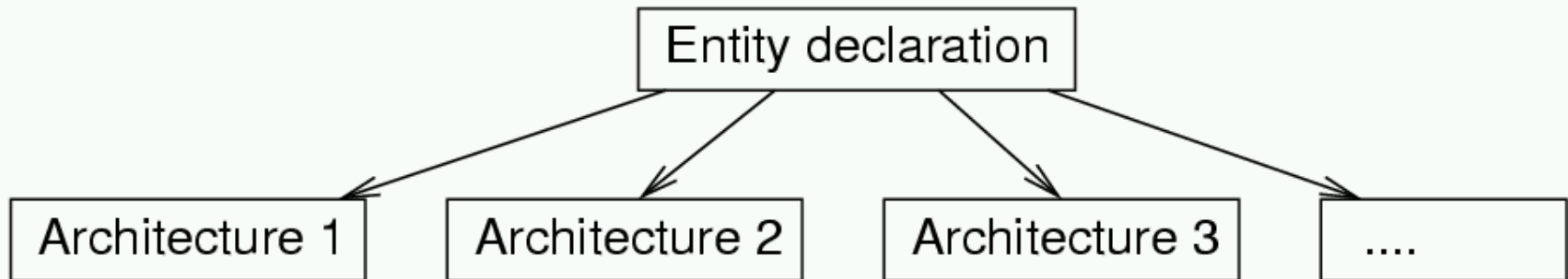
# Goals

- Two goals: simulation and synthesis
- Synthesis: compilation into an implementation technology such as ASIC or FPGA
- Not all constructs in VHDL are suitable to synthesis
- Modelling at various levels of abstraction
- Technology-independent
  - Re-Usability of specifications
- Standard
  - Portability (different synthesis and analysis tools possible)
- Validation of designs based on the same description language for different levels of abstraction

**Here:** Only some aspects of VHDL, not complete language.

# Entities and architectures

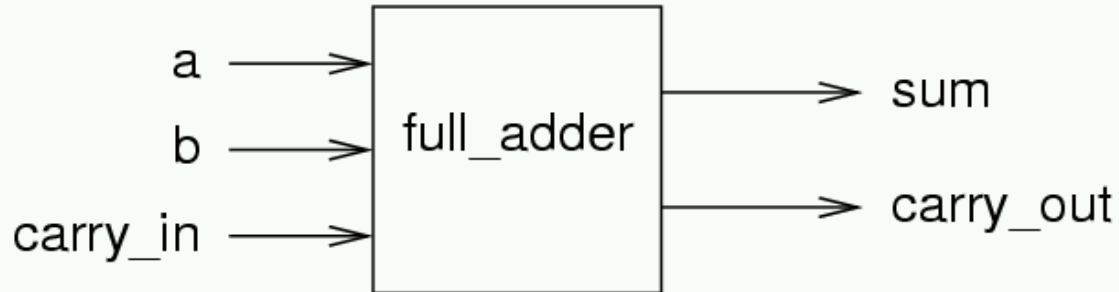
- Each design unit is called an **entity**.
- Entities are comprised of **entity declarations** and one or several **architectures**.



Each architecture includes a model of the entity. By default, the most recently analyzed architecture is used. The use of another architecture can be requested in a **configuration**.

# Example: full adder

## - Entity declaration -



- **Entity declaration:**

**entity** full\_adder **is**

**port**(a, b, carry\_in: **in** Bit; -- input ports

sum, carry\_out: **out** Bit); -- output ports

**end** full\_adder;

## Example: full adder

### - Architecture with behavioural body

**architecture** behavior **of** full\_adder **is**

**begin**

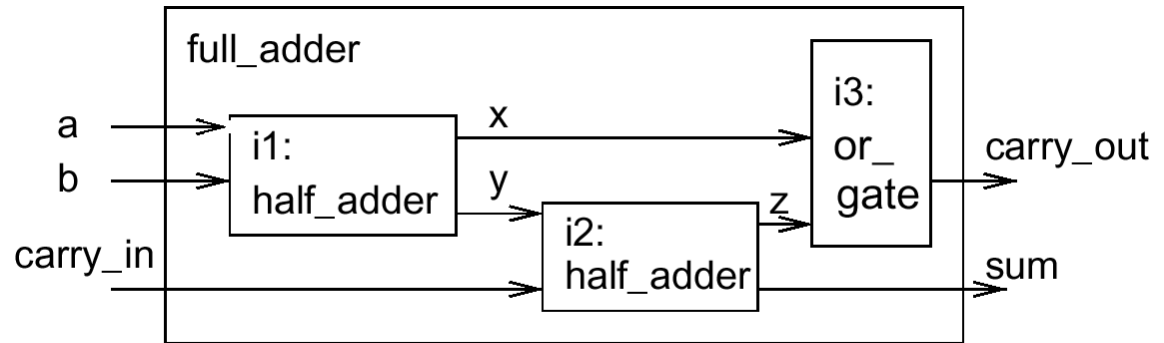
sum           <= (a xor b) xor carry\_in **after** 10 Ns;

carry\_out <= (a and b) or (a and carry\_in) or  
                 (b and carry\_in)           **after** 10 Ns;

**end** behavior;

# Example: full adder

## - structural body



**architecture structure of full\_adder is**

**component half\_adder**

**port (in1,in2:in Bit; carry:out Bit; sum:out Bit);**

**end component;**

**component or\_gate**

**port (in1, in2:in Bit; o:out Bit);**

**end component;**

**signal x, y, z: Bit; -- local signals**

**begin -- port map section**

**i1: half\_adder port map (a, b, x, y);**

**i2: half\_adder port map (y, carry\_in, z, sum);**

**i3: or\_gate port map (x, z, carry\_out);**

**end structure;**



## Example: full adder

### - Architectures

- Architectures describe implementations of entities.
- For component half\_adder we need
  - An entity, e.g.

```
entity half_adder
  port (in1,in2:in Bit; carry:out Bit; sum:out Bit);
end half_adder;
```
  - (At least) one architecture
    - This architecture may contain components, too.
- Architectures and their components can define a hierarchy of arbitrary depth.

# Structural and behavioural descriptions

- Structural descriptions use component instances.
- Behavioural descriptions describe behaviour without defining the structure of the system.
  
- Mixtures are **possible**.
- Mixtures are **needed**, at least for the leaves in structural hierarchy.
  
- Structural hierarchy is essential for a compact and clear modelling of large (hardware) systems.
- To define semantics of VHDL, we can assume that the structural hierarchy is „flattened“, i.e., we can assume w.l.o.g. that we have just an behavioural description.

# Processes

- Behavioural descriptions consist of a set of concurrently executed processes.

- Syntax:

```
[label:]  
process[(sensitivity list)]  
    declarations  
begin  
    statements  
end process [label]
```

# Processes – Examples (1)

```
architecture RTL of NANDXOR is
begin
  process
  begin
    if (C='0') then
      D <= A nand B after 5 ns;
    else
      D <= A and B after 10 ns;
    end if;
    wait on A, B, C;
  end process;
end RTL;
```

## Processes – Examples (2)

```
signal clk : std_logic;  
  
...  
  
clk_gen : process  
begin  
    clk <= 0;  
    wait for 5 ns;  
    clk <= 1;  
    wait for 5 ns;  
end process clk_gen;
```

## Processes – Examples (3)

```
architecture RTL of DFF is
begin
  p : process
  begin
    if (clk'event) and (clk='1`) then
      Q <= D;
    end if;
    wait on clk;
  end process p;
end RTL;
```

# Processes - Execution

- Processes are not allowed to have subprocesses (no hierarchy of processes).
- Processes are executed sequentially until a wait statement is encountered.
- Processes are reactivated according to conditions of wait-statements.
- Different types of wait-statements

# Wait-statements

Four possible types of **wait**-statements:

- **wait on *signal list***;
  - wait until at least one of the signals in signal list changes;
  - Example: **wait on** a;
- **wait until *condition***;
  - wait until condition is met;
  - Example: **wait until** c='1';
- **wait for *duration***;
  - wait for specified amount of time;
  - Example: **wait for** 10 ns;
- **wait**;
  - suspend indefinitely



# Processes - Sensitivity lists

- Sensitivity lists are a shorthand for a single **wait on**-statement at the end of the process body:
- **process** (x, y)  
  **begin**  
    prod <= x and y ;  
  **end process**;

is equivalent to

- **process**  
  **begin**  
    prod <= x and y ;  
    **wait on** x,y;  
  **end process**;

# Signal assignments

- Signal assignments outside processes can be viewed as implicit processes:

**a <= b and c after 10 ns**

is equivalent to

```
process(b, c)  
begin  
  a <= b and c after 10 ns  
end
```

# Constants, signals and variables

- **Constants**

- the value of a constant cannot be changed.

- **Examples:**

- `constant PI : real := 3.1415;`

- `constant DEFAULT : bit_vector(0 to 3) := „1001“;`

- `constant PERIOD : time := 100 ns;`

# Constants, signals, and variables

- Variables
  - Variables are declared locally in processes (and procedures / functions) and are only visible in this scope.
- Signals
  - Can be viewed as a wire
  - Signals cannot be declared in processes (procedures / functions), but in architectures (outside processes).
- Syntax:
  - **variable\_assignment** ::=  
target := expression
    - Example:  
Sum := 0
  - **signal\_assignment** ::=  
target <= [ **delay\_mechanism** ] waveform\_element  
{ , waveform\_element }
  - **waveform\_element** ::=  
value\_expression [ **after time\_expression** ]
    - Example:  
Inpsig <= '0', '1' after 5 ns, '0' after 10 ns, '1' after 20 ns;

# Variable versus signal assignment

- Variable assignments are performed **sequentially** and directly after their occurrence,
- Signal assignments are performed **concurrently**, i.e. they are (sequentially) collected until the process is stopped and are performed in parallel after all processes are stopped.

```
signal a : std_logic := `0`;
signal b : std_logic := `1`;
...
swap : process
variable c : std_logic := `1`;
variable d : std_logic := `0`;
begin
    a <= b; b <= a;
    c := d; d := c;
    wait on a, b;
end process swap;
```

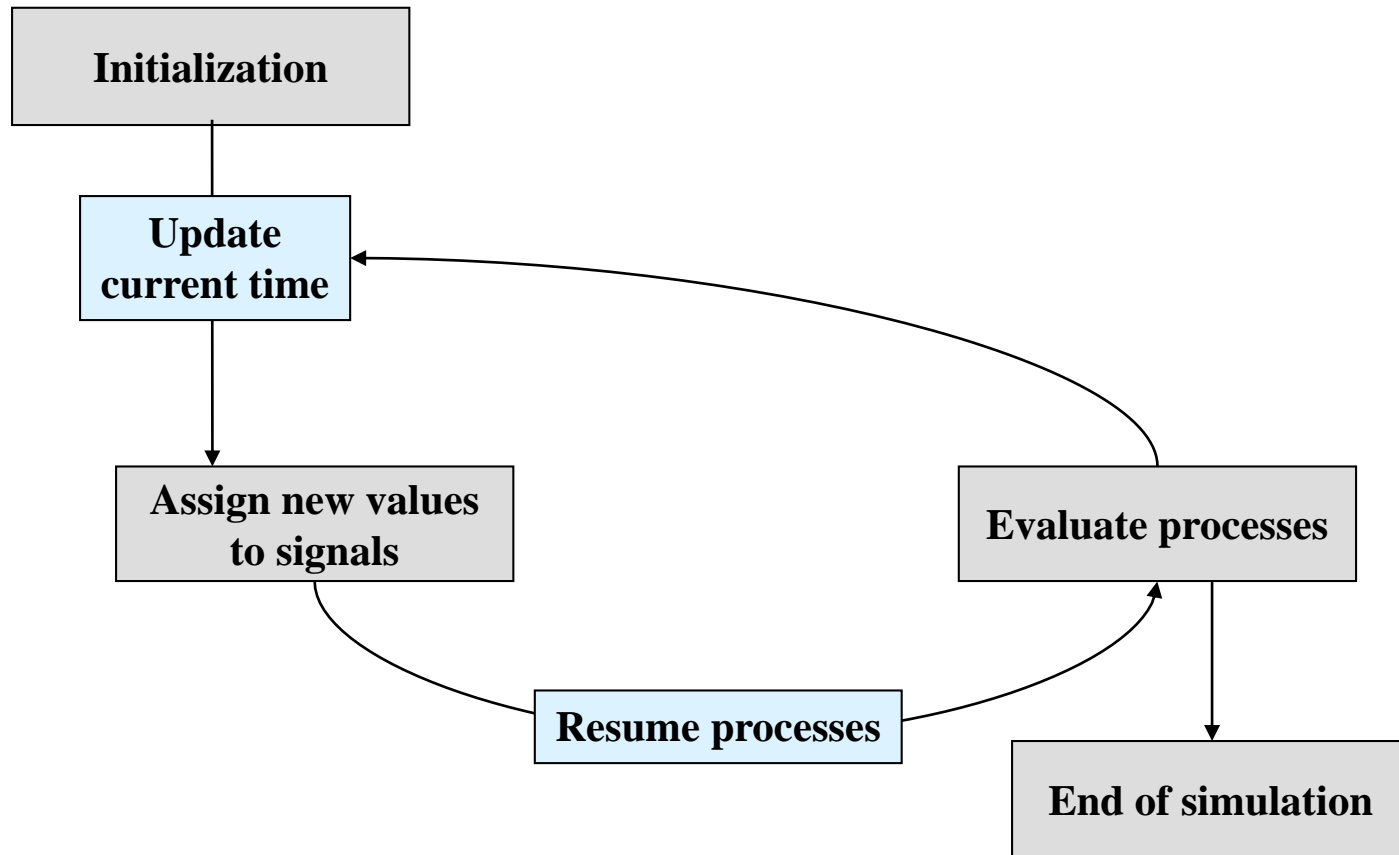
# Semantics of VHDL:

## Basic concepts

- „Discrete event driven simulation“
- Step-based semantics as in StateCharts:
  - Computation as a series of basic steps
  - Time does not necessarily proceed between two steps
  - Like superstep semantics of StateCharts
- Concurrent assignments (of signals) like concurrent assignments in StateCharts.

Steps consist of two stages.

# Overview of simulation



# Transaction list and process activation list

- Transaction list
  - For signal assignments
  - Entries of form  $(s, v, t)$  meaning „signal  $s$  is set to value  $v$  at time  $t$ “
  - Example: (clock, '1', 10 ns)
- Process activation list
  - For reactivating processes
  - Entries of form  $(p_i, t)$  meaning „process  $p_i$  resumes at time  $t$ “.