System Modelling and SDL-Matlab Cosimulation

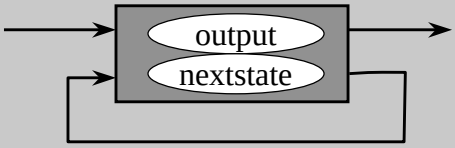
Axel Jantsch, Royal Institute of Technology
Stockholm, Sweden

Overview

- Models of Concurrency
  - Data flow
  - Perfect synchrony
  - Discrete event
- SDL/Matlab Cosimulation
  - Synchronisation and Communication
  - Design Flow

Computational Paradigms

- **State machine**
  - ⇨ **Hierarchy**
  - ⇨ **Data path**



- **Algorithm**
  - ⇨ **Procedural hierarchy**
  - ⇨ **Imperative**

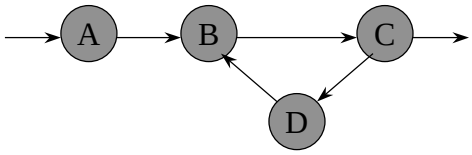
```
for i = 1 to 10 do
    if i == 5
    then …
    else …
    endif
enddo
```

- **Declarative**
  - ⇨ **Functions**
  - ⇨ **Relations**

$f(x) = 3x + 9$
$g(x,y) = \{filter(x,p1,p2), mode(x,y,p3,p4)\}$

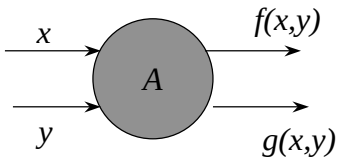
$r(x) > 3x + 9$
$s(A,I) : \forall x_i, x_j \in A, i,j \in I : i < j \Rightarrow x_i \leq x_j$

Dataflow Process Networks



- Networks of actors connected with streams
- Hierarchy of networks
- Communication is buffered with unbounded FIFOs

Functional Actors



- No side effects
- For the same input values produce the same output values
  - ⇨ Functional for each firing cycle
  - ⇨ Functional over the entire streams

Firing Rules

- **Sequential with blocking read**
- **An actor with $p \geq 1$ input streams can have N firing rules:**

$$\Re = \{\mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_N\}$$

$$\mathbf{R}_i = \{R_{i,1}, R_{i,2}, ..., R_{i,p}\}$$

adder: $\mathbf{R}_1 = \{[*],[*]\}$

selector: $\mathbf{R}_1 = \{[*], \perp, [T]\}$
$\mathbf{R}_2 = \{\perp, [*], [F]\}$

Nondeterminate merge: $\mathbf{R}_1 = \{[*], \perp\}$
$\mathbf{R}_2 = \{\perp, [*]\}$

Sequential Firing Rules

Firing rules are sequential if the following procedure succeeds:

1) Find an input ***j*** such, that all firing rules require at least one token from that input; If no such input exists, fail;

2) For the input ***j*** divide the firing rules into subsets, one for each specific token value mentioned in the first position of the firing rules;

3) Remove the first element for the input ***j*** of all firing rules;

4) If all subsets have empty firing rules, succeed; Otherwise repeat these steps for any subset with nonempty firing rules;

Nondeterminate merge: $\mathbf{R}_1 = \{[*], \perp\}$
$\mathbf{R}_2 = \{\perp, [*]\}$

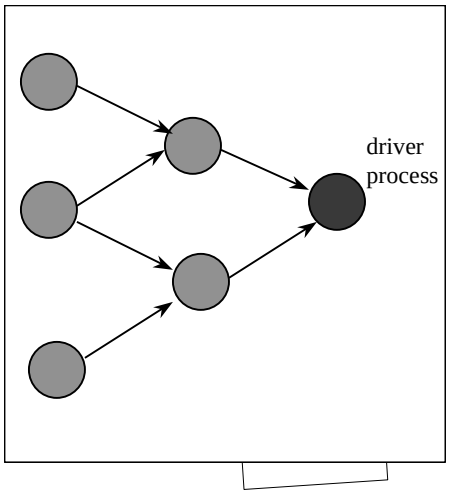
1) fails.

Execution Models

- Concurrent processes
- Dynamic scheduling
- Static scheduling
- Execution on parallel architectures

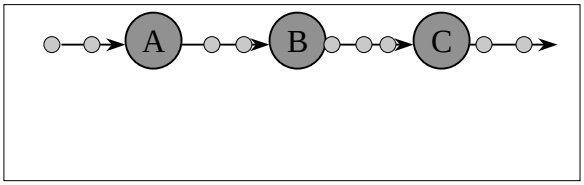
Execution Models

- Concurrent processes



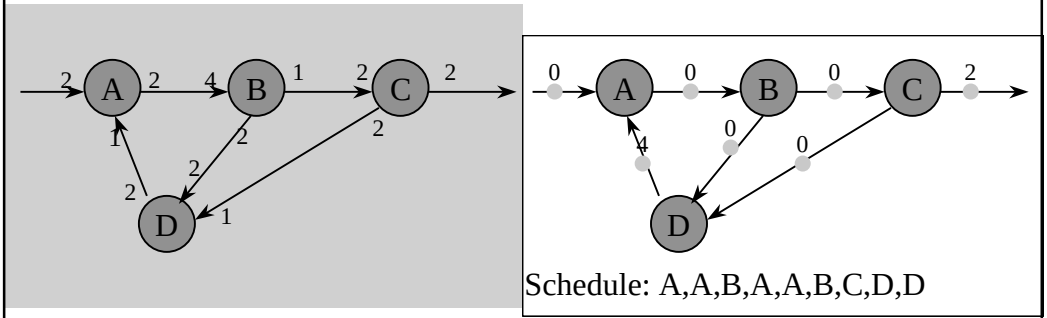
Execution Models - cont'd

- **Dynamic scheduling - data flow networks can in general not be statically scheduled**



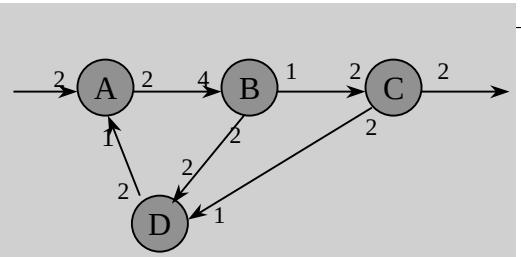
Execution Models - cont'd

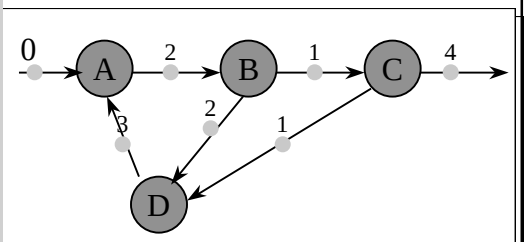
- **Static scheduling: For Synchronous Data Flow a static schedule can always be found and a complete cycle can be executed with bounded memory.**



Execution Models - cont'd

- **Schedule on a parallel architecture**



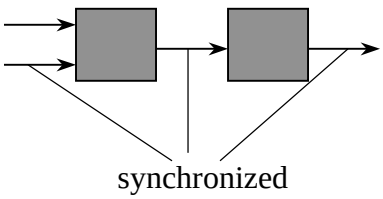


Schedule: A,A,A,A,A,A,A,A,A,A,A
B B,C,B,D,B,C,B
D D

- Good heuristics exist for a maximum throughput schedule on a parallel architecture.

Perfect Synchrony

- **Perfect synchrony assumption:**
  - ⇨ **Computation takes no time**
  - ⇨ **Communication takes no time (synchronous broadcast)**



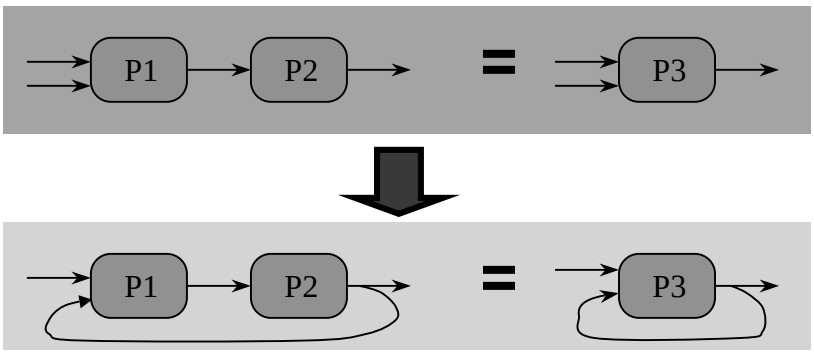

```
<Initialize memory>
foreach period do
        <Read inputs>
        <Compute outputs>
        <Update memory>
end
```

- **Assumption: The system reacts rapidly enough to perceive all external events in suitable order.**

Features of Synchronous Languages

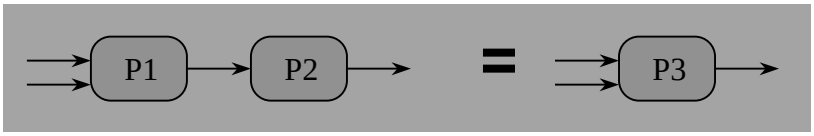
- **Deterministic**
- **Amenable to formal analysis**
- **Efficient synthesis**
- **Substitution of equivalent blocks preserves behaviour**

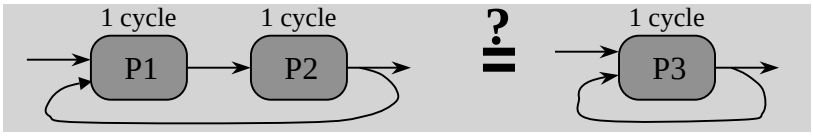
Substitution of Equivalent Blocks



Clocked Synchronous Models

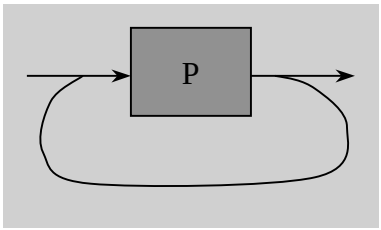
- **Computation takes 1 clock cycle**
- **Communication takes no time**
- **Substitution of blocks must consider timing behavour**





Causality in Synchronous Languages

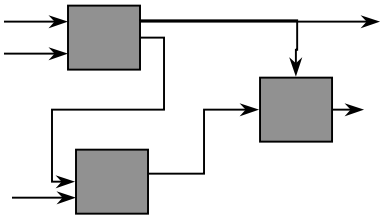
- **Programs in a synchronous language represent equations.**
- **Recursive equations may have 0, 1 or more solutions.**



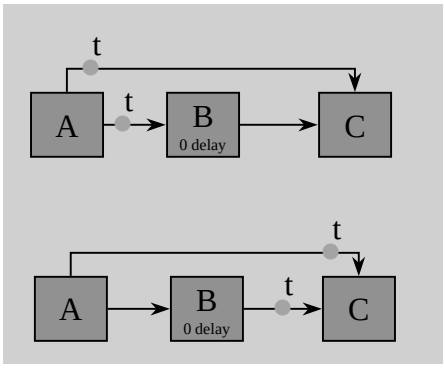
- x = not x
- $y = y$
- $z = (z*z + 1.0)/2.0$
- u = if c then v else w;
  v = if c then w else u;

Discrete Event Models

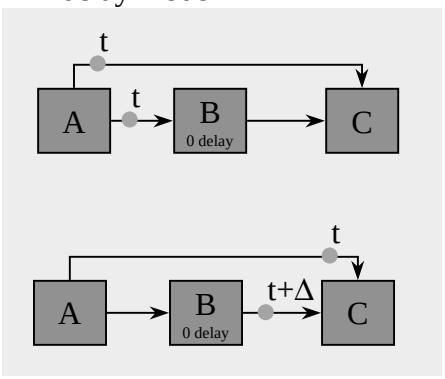
- Event driven dynamics
- Events:
  - ⇨ Primary input stimuli
  - ⇨ Internally generated events
- Events have totally ordered time stamps
- Components have arbitrary delays
- Discrete or continuous time
- Most general timing model
- Primarily targeted to simulation



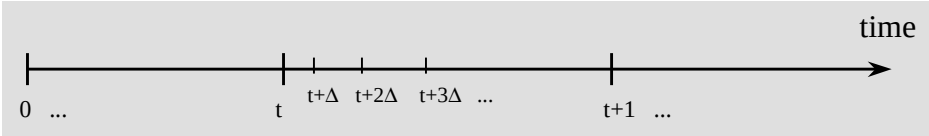
Simultaneous Events

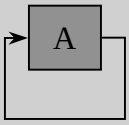


Δ delay model



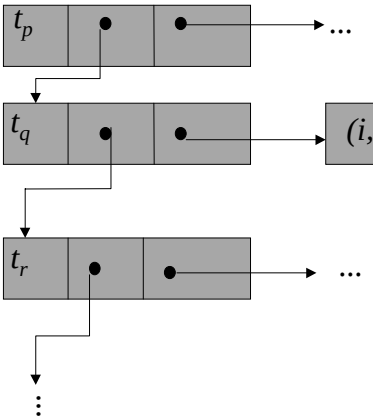
Delta Time





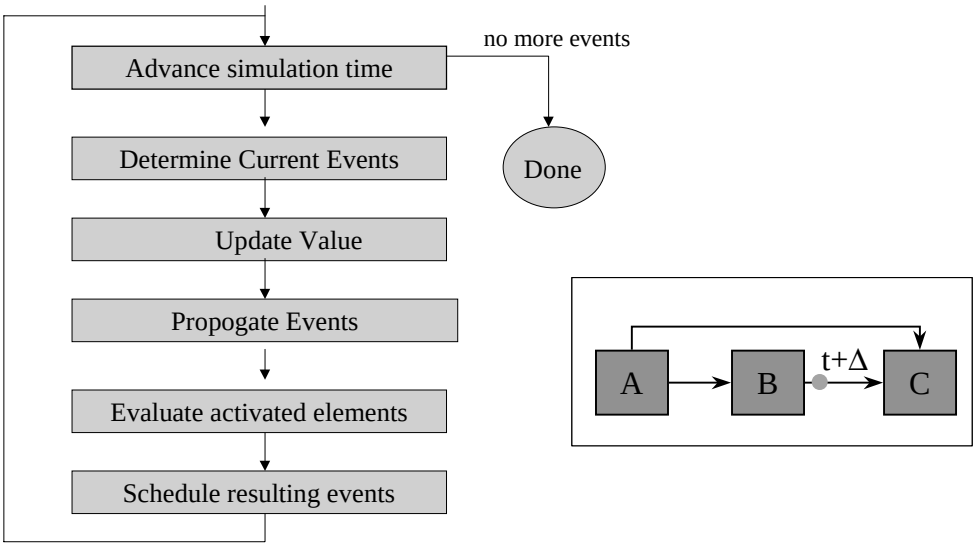
The model allows infinite feed back loops between t and t+1

Event List



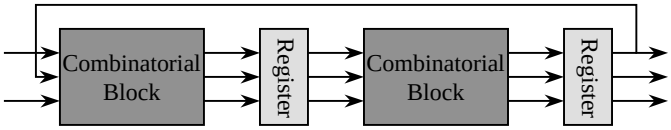

```
While ( event list not empty)
  begin
    t = next time in list
    process entries for time t
  end
```

Event Driven Simulation



Discrete Event Models

- Global event queue is a bottleneck
- Timing model is close to physical time
  - ⇨ Good to simulate timing behaviour of existing components;
  - ⇨ Difficult to synthesize
  - ⇨ Difficult to formally verify
- DE Models are interpreted according to a different timing model: Clocked synchronous model



Heterogeneous System Modelling

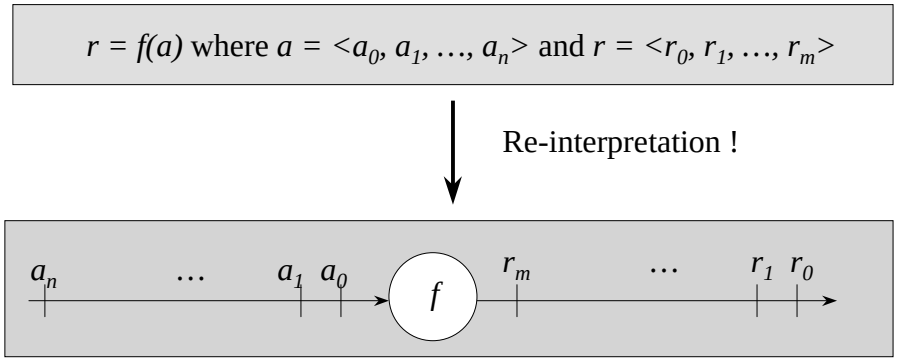
- Heterogeneous Systems
- Different Communities of Engineers
- Established Languages with different profiles
- Established design flows

SDL and Matlab

- SDL
  - ⇨ Communicating State Machines
  - ⇨ Communication is buffered with infinite FIFOs
  - ⇨ Non-deterministic elements
  - ⇨ Partially or totally ordered global time
  - ⇨ Discrete events govern the execution
- Matlab
  - ⇨ Data flow model
  - ⇨ Demand driven execution
  - ⇨ Deterministic
  - ⇨ Partially ordered events; no global time
  - ⇨ Vector oriented computation

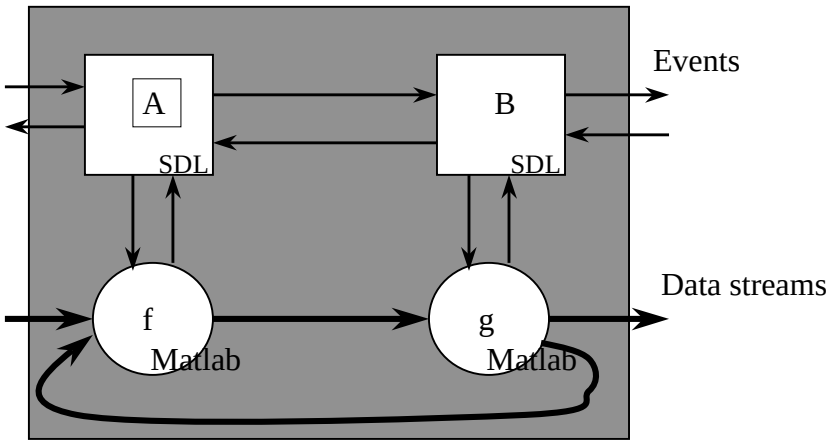
Matlab - SDL Integration: Timing

- **Equip Matlab with a timing model with totally ordered events**

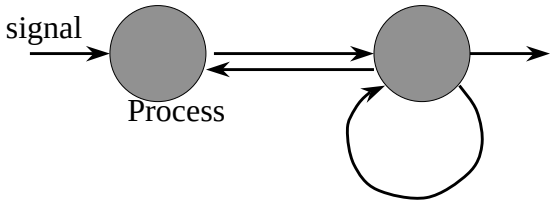


Matlab - SDL: Synchronisation

- **Provide a synchronization mechanism which preserves Matlab's vector oriented computation**



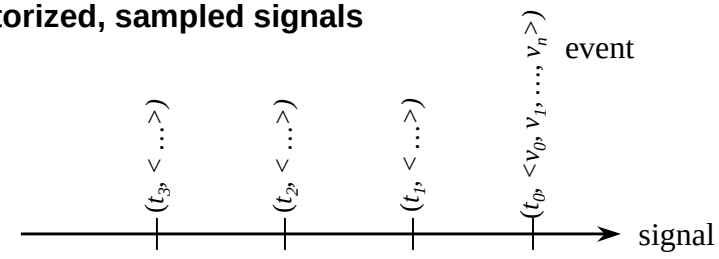
Composite Signal Flow



- **Execution Model**
  - ⇨ **Data flow process**
  - ⇨ **Processes may have state**
- **Signals**
  - ⇨ **Signals are sets of events**
  - ⇨ **An event is a (value, tag) pair**

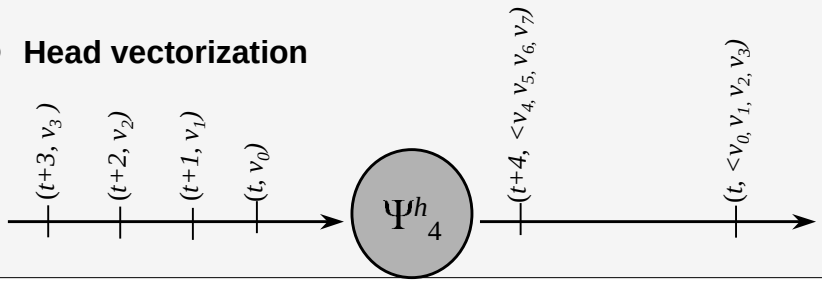
Signals

- **Signals**
  - ⇨ **Signals are sets of events**
  - ⇨ **An event is a (value, tag) pair**
- **Sampled Signals**
  - ⇨ **Values are only defined for tags $t = t_0 + n\,\lambda$**
- **Vectorized Signals**
  - ⇨ **Event values are vectors of constant length**
- **Vectorized, sampled signals**

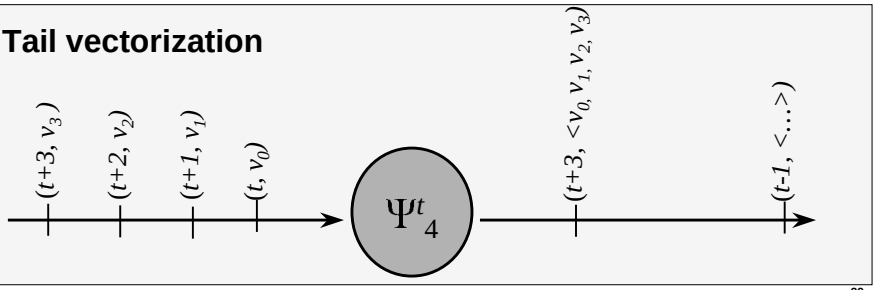


Vectorization

- Head vectorization

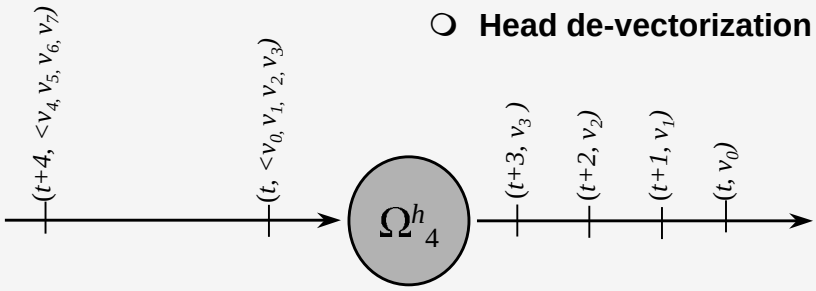


- Tail vectorization

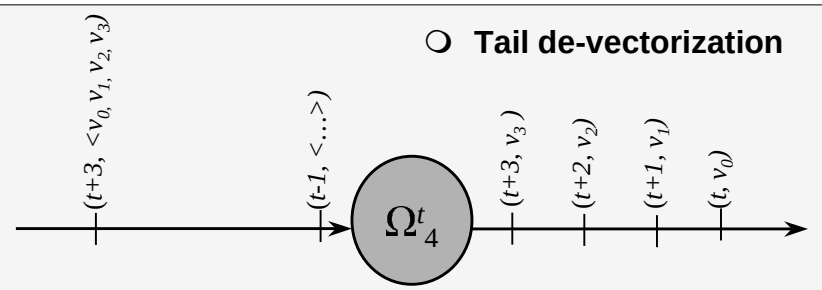


De-Vectorization

- Head de-vectorization

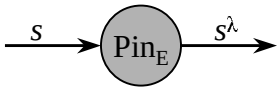


- Tail de-vectorization

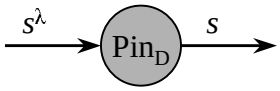


Conversion and Synchronization

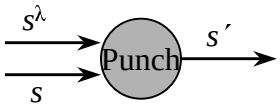
- **Pin Encoding**



- **Pin Decoding**

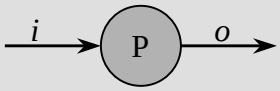


- **Punch**



Causality

- **A process is causal if for all possible input and output streams two output streams never differ earlier than the corresponding two input streams.**



$$i_1 = p_1\,\alpha\,p_2 \qquad o_1 = q_1\,\beta\,q_2$$

$$i_2 = p_1\,\gamma\,p_3 \qquad o_2 = q_1\,\delta\,q_3$$

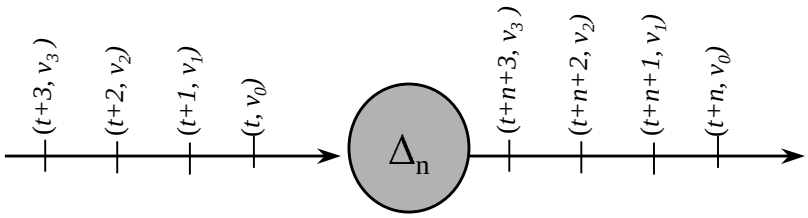
$$\alpha \neq \gamma \qquad \beta \neq \delta$$

P is causal if and only if $\text{tag}(\alpha) \leq \text{tag}(\beta)$

- **Tail vectorization is causal**
- **Tail de-vectorization is <u>not</u> causal**
- **Head vectorization is <u>not</u> causal**
- **Head de-vectorization is causal**

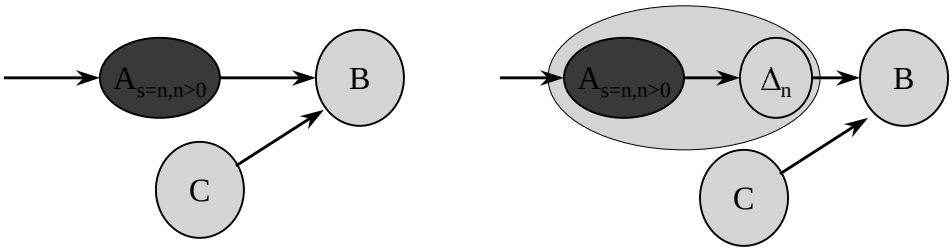
Causality and Delay Processes

- By combining a non-causal process with a delay process, the resulting compound process can be causal
- A delay process outputs every input event delayed by a specific time.



Constraints on Modelling

- Modelling constraints must ensure that processes have data available when they need it.



Unsafe situation

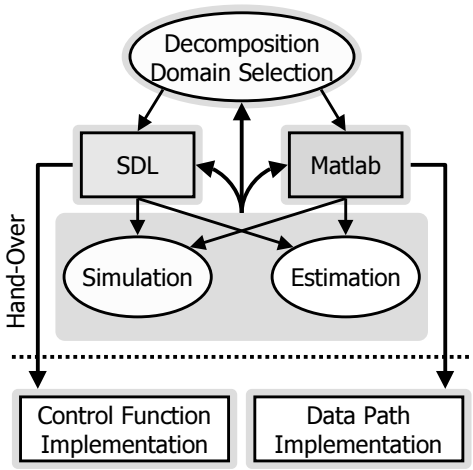
Safe situation

Applications

- Co-Modelling of Matlab and SDL
  - ⇨ Causality constraints imply modelling constraints to safely mix Matlab and SDL processes
- Timing analysis
  - ⇨ Causality constraints can be interpreted as timing constraints derived from the timing of streams
- Parallel Simulation
  - ⇨ A partition must be a causal process
  - ⇨ Only periodic signals may cross partition boundaries

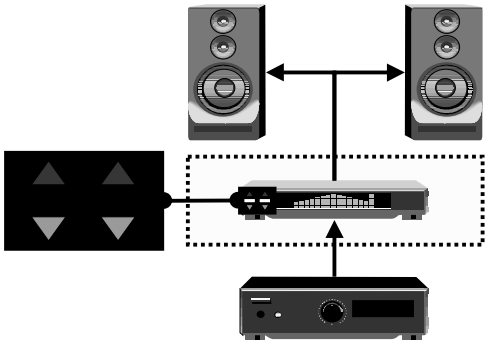
SDL - Matlab Design Flow

- Decomposition
- Domain Selection
- Simulation
- Estimation
- Iteration
- Hand-Over
- Implementation



Equalizer Example

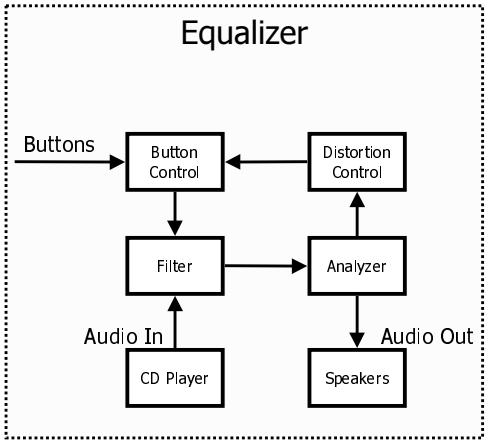
- Demonstration of Concept
- Input
  - ⇨ Audio In
  - ⇨ Bass Up/Down
  - ⇨ Treble Up/Down
- Output
  - ⇨ Audio Out
- Distortion Guard



Decomposition

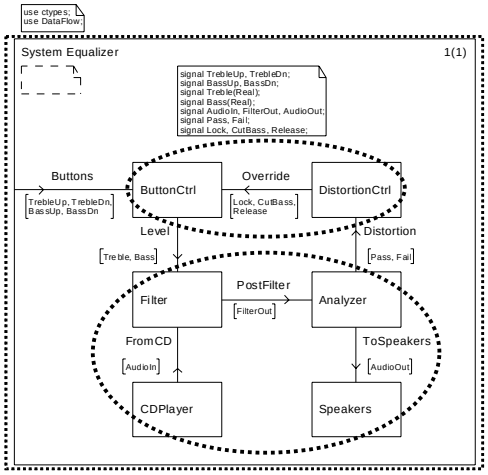
+ Audio Processing
+ Button Handling
+ Distortion Guard
+ Source & Sink

= System Model



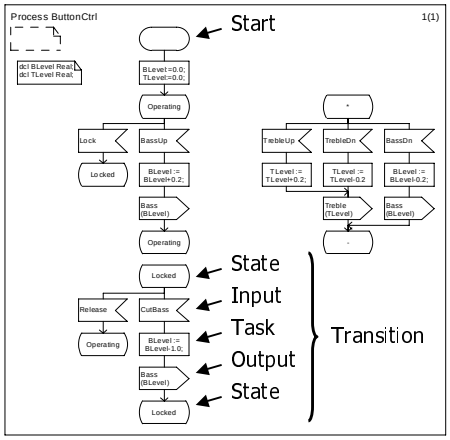
Domain Selection

- Control
  - ButtonCtrl
  - DistortionCtrl
- Data Flow
  - Filter
  - Analyzer
  - CDPlayer
  - Speakers



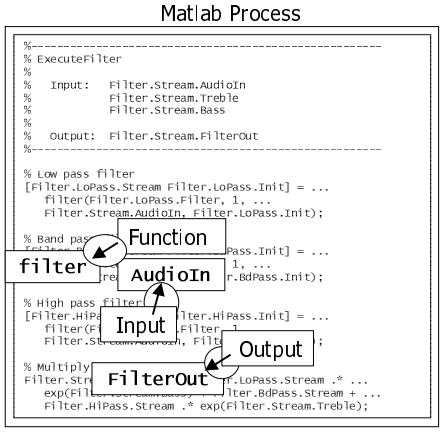
Control Flow Modeling

- **Reactive**
- **SDL Processes**
  - ⇨ **Extended Finite State Machines**
  - ⇨ **Asynchronous Communication**
  - ⇨ **Synchronous Transitions**



Data Flow Modeling

- Transformation
- Matlab Processes
  - ⇨ Streams & Vectors
  - ⇨ Functions
  - ⇨ State

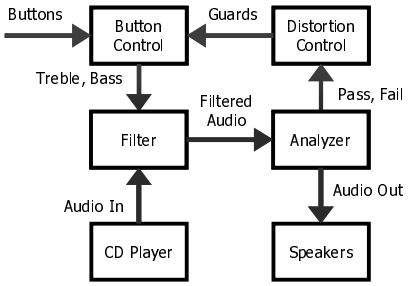


Signal Types

- Data Flow Signals
  - ⇨ Continuously Sampled Data
  - ⇨ Only Data Flow (Matlab) Processes
- Control Signals
  - ⇨ Only Control (SDL) Processes
- Notification Signals
- Message Signals
  - ⇨ Containing Data
- Punch Signals

Choosing Signal Type

- Control Signals
  - Buttons and Guards
- Data Flow Signals
  - Audio
- Message Signals
  - Treble and Bass
- Notification Signals
  - Pass and Fail

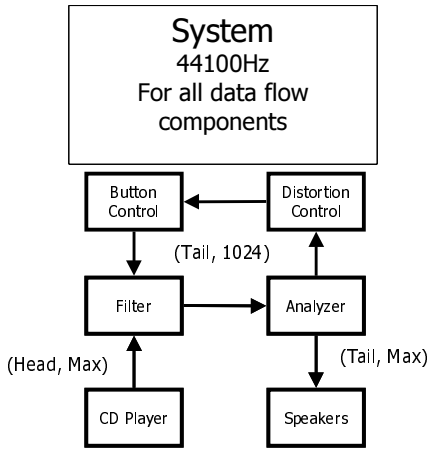


Synchronization Guidelines

- **Head Synchronization**
  - ⇨ **Use for sources**
  - ⇨ **Start with large frames - split**
  - ⇨ **Keep as long as possible**
- **Tail Synchronization**
  - ⇨ **Begin as late as possible**
  - ⇨ **Start with small frames - concatenate**
  - ⇨ **Use for sinks**
- **Avoid Switching between Head and Tail**

Choosing Synchronization

- CD Player (Source)
  - Head Synchronization
  - Max Frame Size
- Filter Process
  - Tail Synchronization
  - Frame Size = 1024
- Analyzer Process
  - Tail Synchronization
  - Frame Size = 1024
- Speakers (Sink)
  - Tail Synchronization
  - Max Frame Size



Summary

- **Heterogeneous system modeling is a necessity;**
- **Multiple Models of Computation and Concurrency must be integrated;**
- **Conceptual integration is the basis for the integration of simulation, synthesis, and formal analysis**

References

- **Data flow process networks:**

E. A. Lee and T. M. Parks, "Dataflow Process Networks", *Proceedings of the IEEE*, May 1995.

Edward Ashford Lee and David G. Messerschmitt, "Synchronous Data Flow", *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235-1245, September 1987.

- **Perfectly synchronous models:**

Gerard Berry, "The Foundations of Esterel", *Proof, Language and Interaction: Essays in Honour of Robin Milner*, MIT Press, edited by G. Plotkin and C. Stirling and M. Tofte, 1998.

Nicolas Halbwachs, "Synchronous Programming of Reactive Systems", *Proceedings of Computer Aided Verification (CAV),* 2000.

Nicolas Halbwachs, *Synchronous Programming of Reactive Systems*, Kluwer Academic Publishers, 1993.

G. Berry, P. Chouronné, and and G. Gonthier, "The Synchronous Approach to Reactive and Real-Time Systems", *IEEE Proceedings*, September" 1991.

- **Discrete Event Models:**

Christos G. Cassandras, *Discrete Event Systems*, Aksen Associates, 1993.

Most VHDL books have a chapter on the simulation cycle, the event queue and the concept of delta delays.

- **On the integration of SDL and Matlab:**

Axel Jantsch and Per Bjureus, "Composite Signal Flow: A Computational Model Compining Events, Sampled Streams, and Vectors", *Proceedings of the Design and Test Europe Conference (DATE),* 2000.

Per Bjureus and Axel Jantsch, "MASCOT: A Specification and Cosimulation Method Integrating Data and Control Flow", *Proceedings of the Design and Test Europe Conference (DATE),* 2000.