ECE382N.23: Embedded System Design and Modeling

Lecture 6 – System Simulation

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

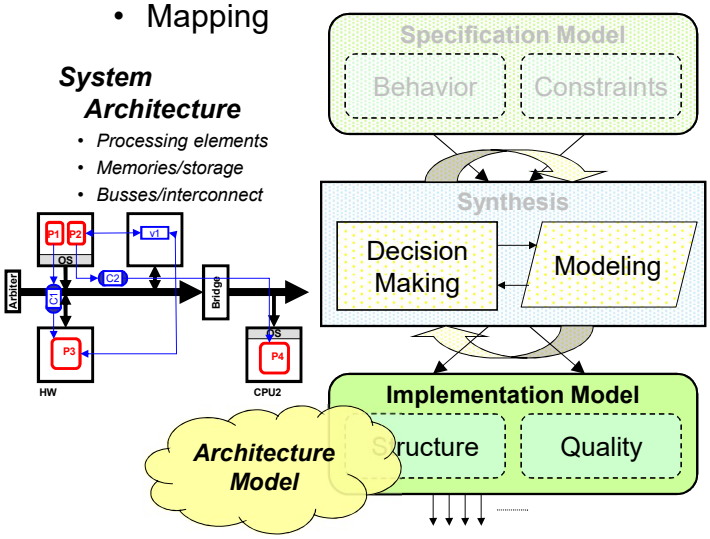
The University of Texas at Austin
Chandra Department of Electrical and Computer Engineering
Cockrell School of Engineering

Lecture 6: Outline

- **System-level architecture models**
  - Computation & communication
  - Modeling & abstraction levels
  - Virtual protoyping & virtual platform models
- **System simulation**
  - System-level design languages (SLDLs)
  - The SystemC language
  - Discrete-event model of computation

System Modeling

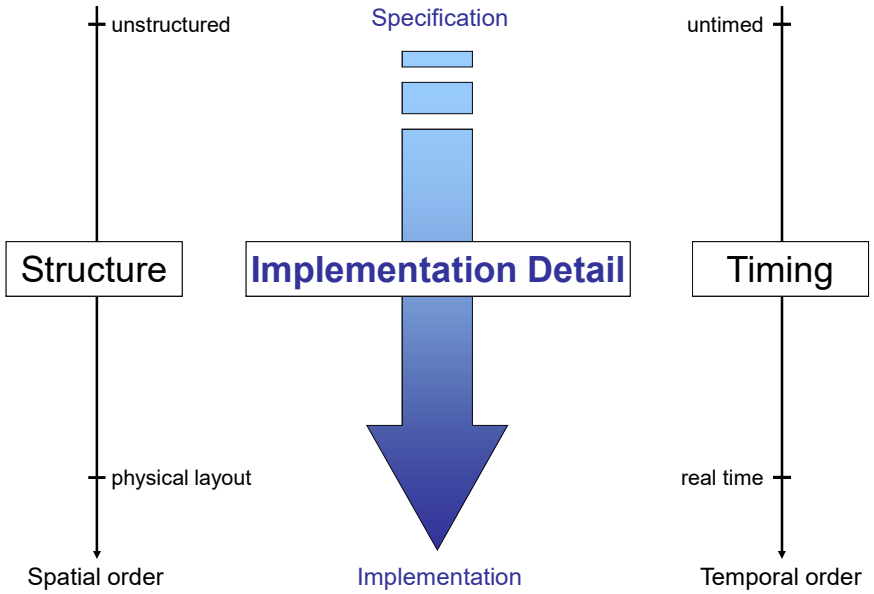
- **System definition**
  - Platform netlist
  - Mapping
- **System quality**
  - Performance, power, …



SoC Components

- **Computation**
  - CPUs
    - Operating systems
  - GPUs
  - Accelerators
  - FPGAs/CGRAs
  - Custom hardware
  - …
- **Storage**
  - Caches
  - Scratchpads
  - …
- **Communication**
  - Busses & bridges
  - Crossbars
  - Network-on-Chip (NoC)
    - Topology
    - Routing
  - …
- **I/O**
  - Bus & memory interfaces
  - Peripherals
    - Audio & video
    - …
  - …

Abstraction Levels



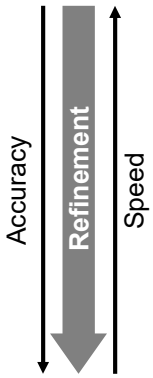
Models of Time (Order)

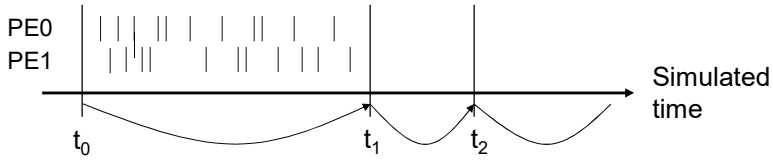
- **Untimed**
  - Partial order based on causality only
    - No ordering in time, explicit dependencies only
    - ➢ Free of implementation (purely behavioral)
    - ➢ Specification & programming, Models of computation (Lecture 2)
- **Logical**
  - Discrete time, partial order
    - Discrete instants of time (time tags $t_0 < t_1 < \cdots t_k < \cdots$), nothing in between
    - Unspecified interleaving of events with same time tag
    - ➢ Freedom of implementation
    - ➢ Simulation & execution, design languages
- **Physical**
  - Continuous time, total order
    - Physical components naturally interleaved in (very fine) time
    - ➢ Differential equations, Hybrid models

Model Speed vs. Accuracy

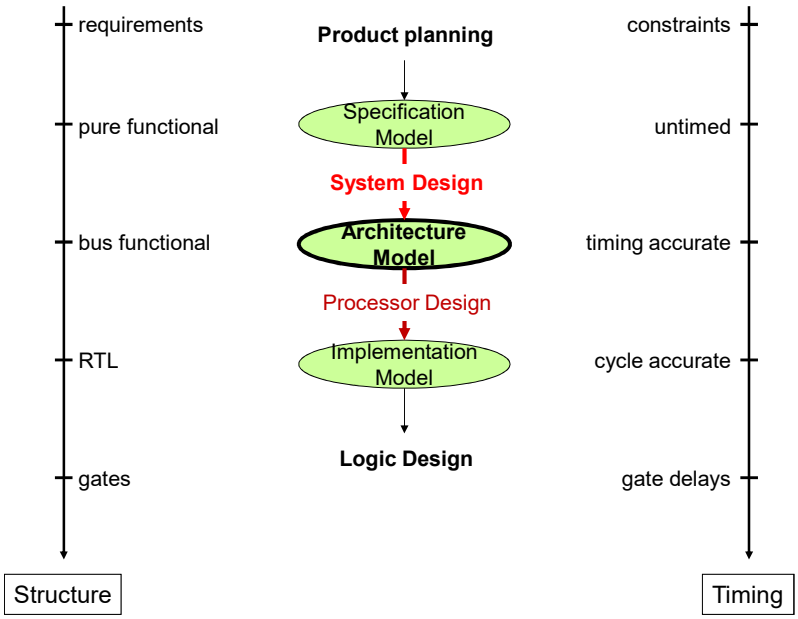
- **Simulation speed**
  - Proportional to number of simulated events
  - Proportional to granularity of simulated time/detail
    - "Real-time": simulated vs. simulation time > 1
- **Simulation accuracy**
  - Proportional to simulated implementation order
  - Inversely proportional to simulated granularity
    - Where order matters (structural concurrency)

➢ **Fundamental modeling tradeoff**



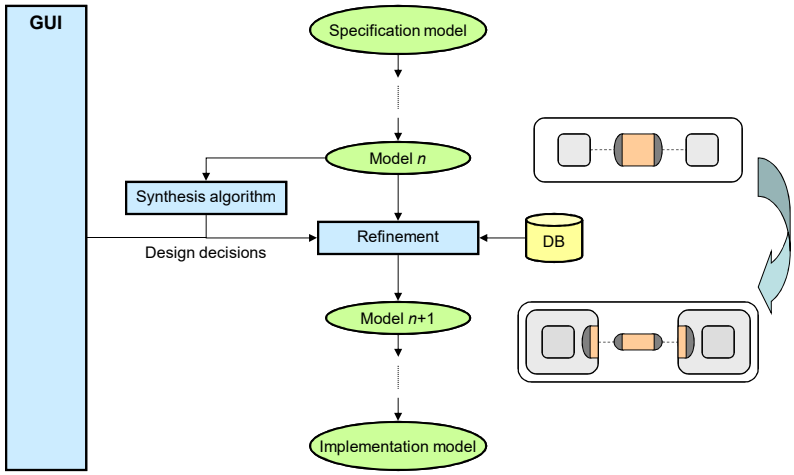


Abstraction Levels



Refinement Flow

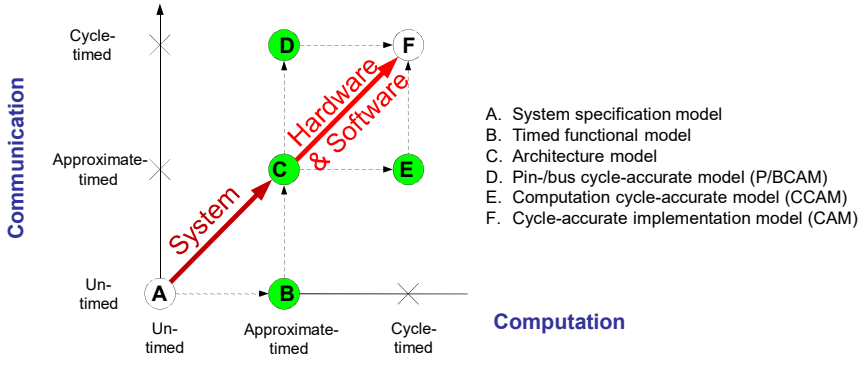
- **Synthesis = Decision making + model refinement**



- **Successive, stepwise model refinement**
- **Layers of implementation detail**

Computation vs. Communication

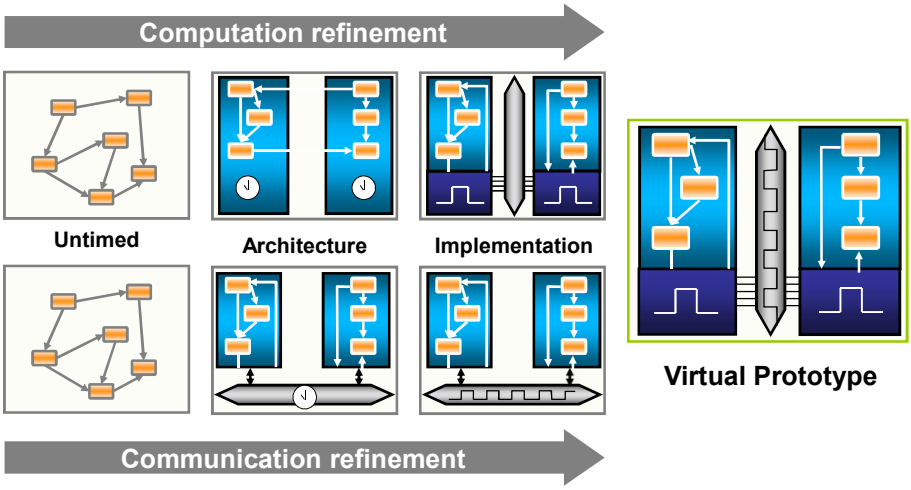
- **System design flow**
  - Path from model A to model F



Source: L. Cai, D. Gajski. "Transaction level modeling: An overview", ISSS 2003

- **Design methodology and modeling flow**
  - Set of models and transformations between models

System-Level Modeling & Refinement



Source: C. Haubelt, Univ. of Rostock

Languages

- **Capture models in machine-readable form**
  - At different levels of abstraction
  - Simulate & execute
  - Automatic refinement, analysis, synthesis
- **Syntax defines grammar**
  - Possible strings over an alphabet
  - Textual or graphical
- **Semantics defines meaning**
  - Execution/simulation model
  - Operational or denotational

Software Programming Languages

- **Imperative programming models**
  - Statements that manipulate program state, control flow
  - ➢ Sequential programming languages [C, C++, …]
    - ➢ Van Neuman semantics
- **Declarative programming models**
  - Rules for data manipulation, data flow
  - ➢ Functional programming [Haskell, Lisp, Excel]
  - ➢ Logic programming [Prolog]

➢ **No concurrency, structure or time**
  - ➢ Sequential behavior at task/block level
  - ➢ Implicit or explicit operation-level parallelism

Hardware Design Languages

- **Netlists**
  - Structure only: components and connectivity
  - ➢ Gate-level [EDIF], system-level [SPIRIT/XML]
- **Hardware description languages (HDLs)**
  - Event-driven behavior: signals/wires, clocks
  - Register-transfer level (RTL): boolean logic
  - ➢ Discrete event [VHDL, Verilog]
- **System-level design languages (SLDLs)**
  - Software behavior: sequential functionality/programs
  - ➢ C-based, event-driven [SpecC, SystemC, SystemVerilog]

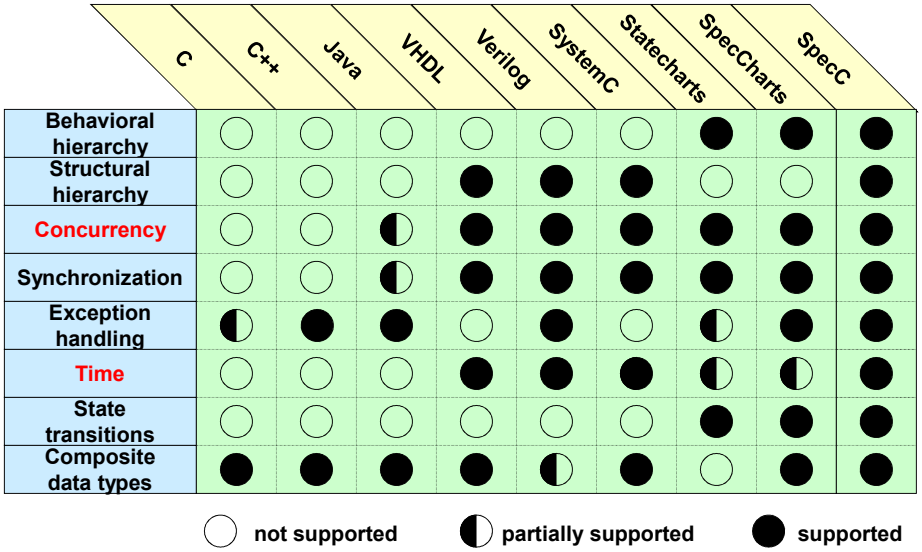
➢ **Structural (block-level) concurrency and time**
  - ➢ Purely behavioral (task-level) concurrency?

System-Level Design Languages (SLDLs)

- **C/C++**
  - ANSI standard programming languages, software design
  - Traditionally used for system design because of practicality, availability
- **SpecC**
  - C extension
  - Developed at UC Irvine, standard by SpecC Technology Open Consortium (STOC)
- **SystemC**
  - C++ API and class library
  - Initially developed at UC Irvine, standard by Open SystemC Initiative (OSCI)
- **SystemVerilog**
  - Verilog with C extensions for testbench development
- **Matlab/Simulink**
  - Specification and simulation in engineering, algorithm design
- **Unified Modeling Language (UML)**
  - Requirements specification, no execution semantics, graphical
  - Extensible (meta-modeling), MARTE & SysML profiles for real-time/embedded/arch
- **IP-XACT**
  - XML schema for IP component documentation, standard by SPIRIT consortium
- **Rosetta (formerly SLDL)**
  - Formal specification of constraints, requirements
- **SDL**
  - Telecommunication area, standard by ITU
- **…**

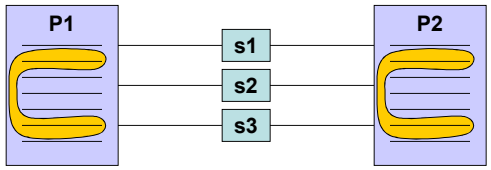
System-Level Design Languages (SLDLs)

- **Requirements**



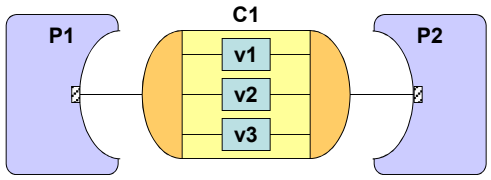
Computation vs. Communication

- **Traditional model**



  - Processes and signals
  - Mixture of computation and communication
  - Automatic replacement impossible

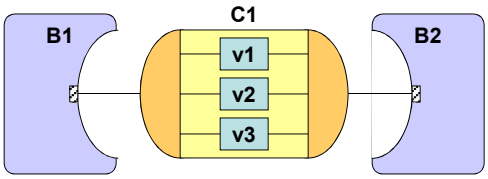
- **SLDL model**



  - Processes and channels
  - Separation of computation and communication
  - Plug-and-play

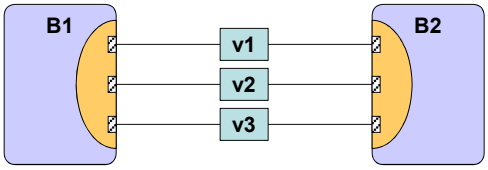
Computation vs. Communication

- **Protocol Inlining**
  - Specification model
  - Exploration model



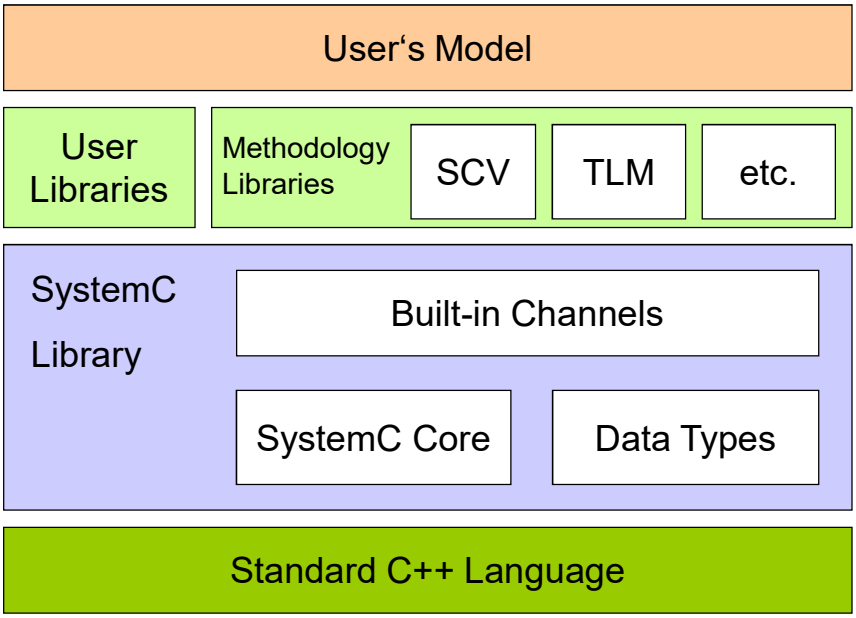
    - Computation in behaviors
    - Communication in channels

  - Implementation model



    - Channel disappears
    - Communication inlined into behaviors
    - Wires exposed

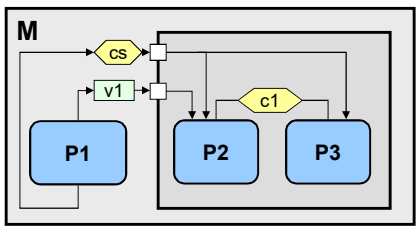
SystemC Language



Source: M. Radetzki, Univ. of Stuttgart

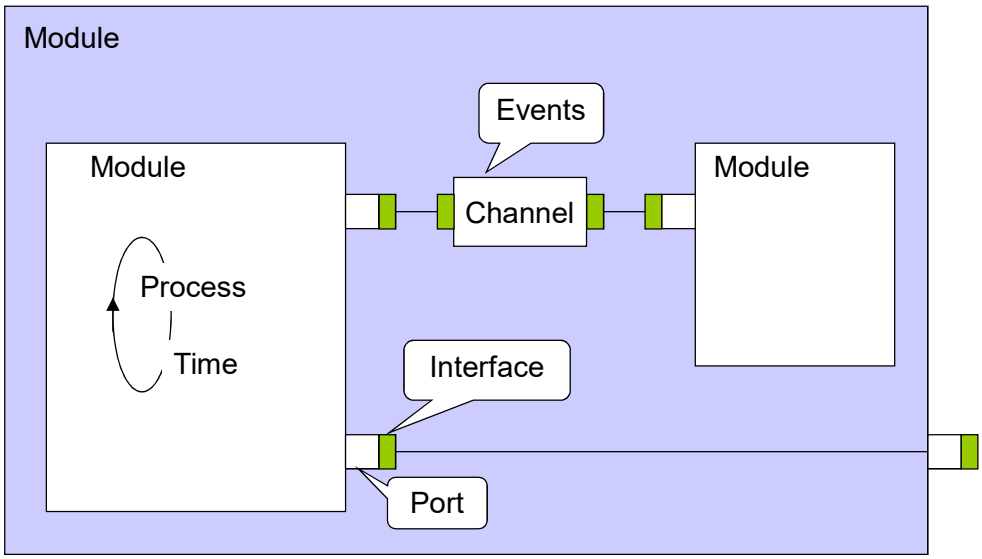
SystemC Fundamentals

- **SystemC structural hierarchy**
  - Modules
  - Ports and variables
  - Channels* and interfaces*
- **SystemC behavioral hierarchy**
  - Parallel leaf processes
    - SC_METHOD (combinatorial)
    - SC_THREAD (behavior)



** SystemC 2.0*

SystemC Structure



+ Bit-true data types

Source: M. Radetzki, Univ. of Stuttgart

SystemC Events

- **Declaration:** `sc_event <name>;`
- **Immediate triggering:** `<name>.notify();`
- **Waiting for occurrence:** `wait( <name> );`

```
int x;
int y;
sc_event new_stimulus;

void Testbench::stim()
{
  x = 3; y = 4;
  new_stimulus.notify();
  x = 7; y = 0;
  new_stimulus.notify();
  // stimulus 7, 0 again
  new_stimulus.notify();
  ...
}
```

```
void Testbench::check()
{
  for(;;)
  {
    wait(new_stimulus);
    if( s == x+y )
      cout << "OK" << ...;
    else
      cout << "ERROR" << ...;
  }
}
```

Source: M. Radetzki

SystemC Built-In Channels

Channel	Matching Ports (Shortcuts)	Events
sc_signal<T>	sc_in<T> sc_out<T> sc_inout<T>	value changed
sc_buffer<T>	sc_in<T> sc_out<T> sc_inout<T>	value written (also if same as previous value)
sc_fifo<T>	sc_fifo_in<T> sc_fifo_out<T>	fifo contents changed
sc_semaphore	--	--
sc_mutex	--	--
sc_clock	sc_in<bool>	value changed

SystemC Time

- `sc_time` **data type**
- **Time units:**
  - SC_FS femtosecond 10^{-15}s
  - SC_PS picosecond 10^{-12}s
  - SC_NS nanosecond 10^{-9}s
  - SC_US microsecond 10^{-6}s
  - SC_MS millisecond 10^{-3}s
  - SC_SEC second 10^{0}s
- **Time object:** `sc_time <name>(<magnitude>, <unit>);`
- **e.g.:** `sc_time delay(10, SC_NS);`
- **usage, e.g.:** `wait(delay);`
- **alternative:** `wait(10, SC_NS);`

Source: M. Radetzki

System-Level Language Semantics

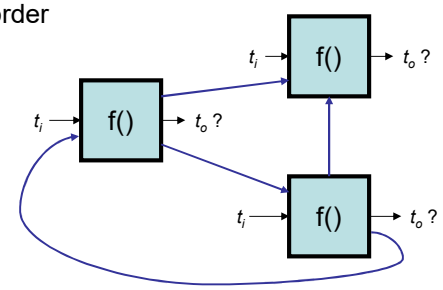
- **Language concepts (syntax)**
  - Behavioral and structural hierarchy
  - Concurrency and time
  - Synchronization and communication
  - Exception handling
  - State transitions
- **Language semantics needed to define the *meaning***
  - Semantics of execution for modeling, simulation, synthesis
  - Model of concurrency, time, synchronization, communication
    - Determinism vs. non-determinism
    - Atomicity
    - …

➢ **Discrete-event model for simulation**

Source: R. Doemer, UC Irvine

Recall: (Logical) Concurrency

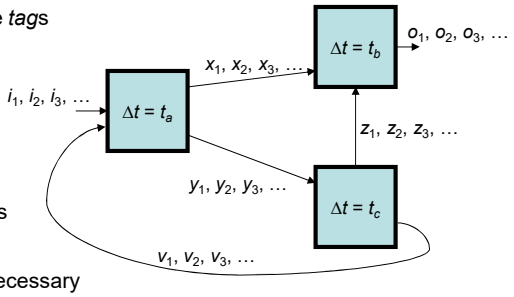
- **Events/actions happening "at the same time"**
  - Undefined, unspecified or unknown order
    - Implementation/simulation determines actual interleaving
  - Communication/synchronization establishes causal order
  - Logical time establishes additional order
    - Partial order



➢ **Fundamental issues**
  - ➢ Non-determinism
  - ➢ Causality loops

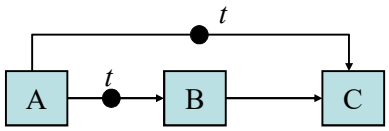
Discrete Event (DE) Model

- **General, universal model for system simulation**
  - Hardware [VHDL, Verilog], system [SpecC, SystemC], network [OMNet]
- **Formal, generic discrete time model**
  - Signals = globally ordered streams of events
    - Event e_i = (*value*, *tag*), discrete time *tags*
  - Asynchronous event processing
    - Execute block on input event
    - Process input and generate output events with *tag*+Δt
  - Flexible
    - Multi-scale, arbitrary dynamic delays
  - Efficient
    - Event-driven, only evaluate when necessary



- **Synthesis challenges**
  - Semantics: simultaneous events (non-determinism), zero-delay cycles
  - Implementation: global order (maintain global notion of time) [PTIDES]

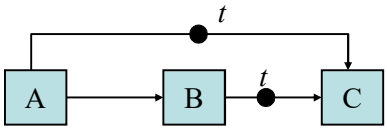
Simultaneous Events




```
void Top::B(void)
{
  void main(void){
    while (true) {
      wait(a);
      …
      b.notify();
    }
  }
};
```

```
void Top::C(void)
{
  void main(void) {
    while(true) {
      // a or b
      wait(a | b);
      …
    }
  }
};
```

Suppose B is invoked first

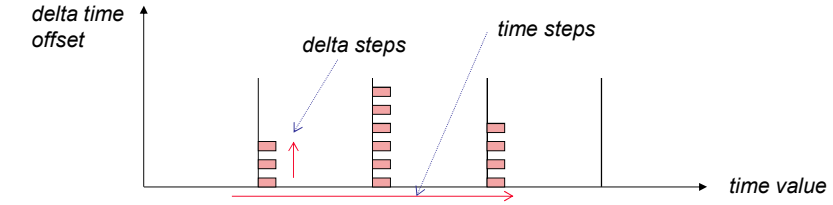


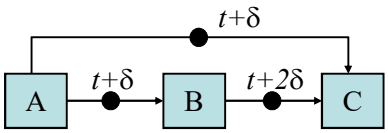
- **Depending on simulator**
  - Process C might be invoked once, observing both inputs in one invocation
  - Process C might be invoked twice, processing events one at a time
  - Non-deterministic order of event processing

Source: M. Jacome, UT Austin.

Delta (Superdense) Time

- **Two-level model of time**
  - Break each time instant into multiple delta steps
  - Each "zero" delay event results in a delta step
  - Delta time has zero delay but imposes semantic order



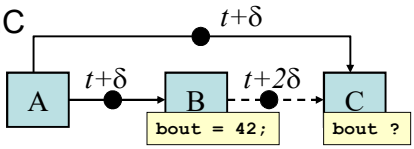


Answer: C is invoked twice

Source: M. Jacome, UT Austin.

Delta Semantics

- **Resolve some non-determinism**
  - As long as each output has "unique" delay
  - Often not the case, e.g. in SystemC

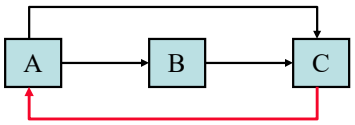


- **Ambiguity still exists**
  - Shared resource/variable accesses in same delta cycle
    - With B->C sharing: B or C first? Undefined order, non-deterministic
- **Alternative semantics based on precedence analysis**
  - Graph is executed in topologically sorted order [Ptolemy]
    - C only invoked once, with B->C dependency: B invoked first
  - Potentially still ambiguous
    - If there is no B->C dependency: B or C first?
    - Only matters if there are other side effects… (`printf()`)

Source: M. Jacome, UT Austin.

Zero-Delay Feedback Loops

- **Causality loop**
  - Where to start & stop?
  - Progress?



- **Reject zero-delay cycles**
  - Forbid all zero delay (every Δt must be strictly > 0) [DEVS]
  - Detect (compile error on zero cycle)
- **Delta cycle semantics**
  - Oscillate with no time progress [Zeno machine/model]
- **Approaches based on topological sorting**
  - Annotate feedback arcs to "break" for ordering purposes
    - Insert explicit δ delay block [Ptolemy]
    - Potentially same oscillation without progress

Source: M. Jacome, UT Austin.

Lecture 6: Summary

- **System architecture modeling**
  - At varying levels of abstraction
  - Computation & communication
- **System-level design languages (SLDLs)**
  - Capture system models in executable form
  - Structural concurrency, synchronization & time
  - Discrete-event model of computation