Concurrent Models of Computation

Edward A. Lee
Robert S. Pepper Distinguished Professor, UC Berkeley
EECS 219D
Concurrent Models of Computation
Fall 2011

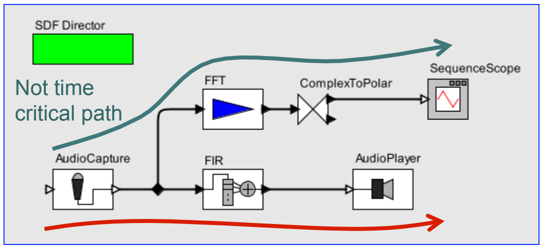
Week 11: Time-Triggered Models

The Synchronous Abstraction Has a Drawback

- "Model time" is discrete: Countable ticks of a clock.
- WRT model time, computation does not take time.
- All actors execute "simultaneously" and "instantaneously" (WRT to model time).

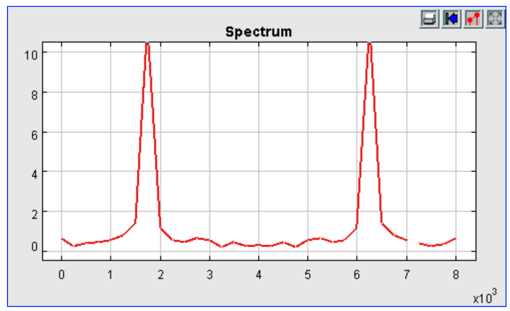
As a consequence, long-running tasks determine the maximum clock rate of the *fastest* clock, irrespective of how frequently those tasks must run.

Simple Example: Spectrum Analysis

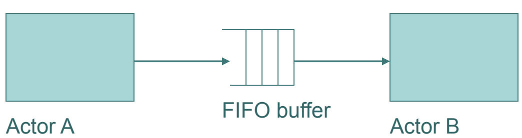


Time critical path

How do we keep the non-time critical path from interfering with the time-critical path?



Dataflow Models



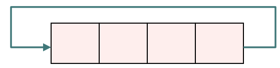
Buffered communication between concurrent components (*actors*).

Static scheduling: Assign to each thread a sequence of actor invocations (*firings*) and repeat forever.

Dynamic scheduling: Each time dispatch() is called, determine which actor can fire (or is firing) and choose one.

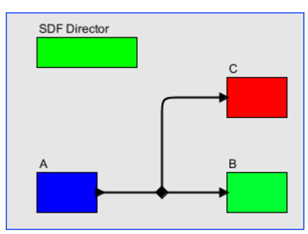
May need to implement interlocks in the buffers.

Buffers for Dataflow



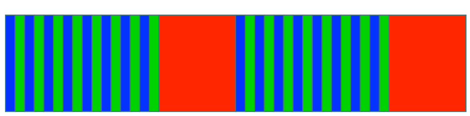
- Unbounded buffers require memory allocation and deallocation schemes.
- Bounded size buffers can be realized as *circular buffers* or *ring buffers,* in a statically allocated array.
  - A *read pointer* r is an index into the array referring to the first empty location. Increment this after each read.
  - A *fill count* n is unsigned number telling us how many data items are in the buffer.
  - The next location to write to is $(r + n)$ modulo buffer length.
  - The buffer is empty if $n == 0$
  - The buffer is full if $n ==$ buffer length
  - Can implement n as a semaphore, providing mutual exclusion for code that changes n or r.

Abstracted Version of the Spectrum Example: Non-preemptive scheduling



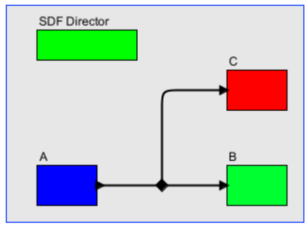
Assume infinitely repeated invocations, triggered by availability of data at A.

Suppose that C requires 8 data values from A to execute. Suppose further that C takes much longer to execute than A or B. Then a schedule might look like this:

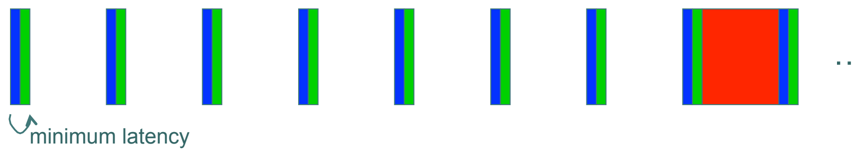


…

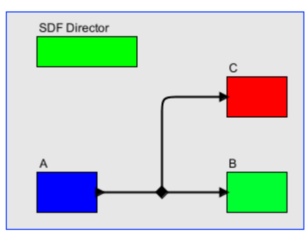
Uniformly Timed Schedule



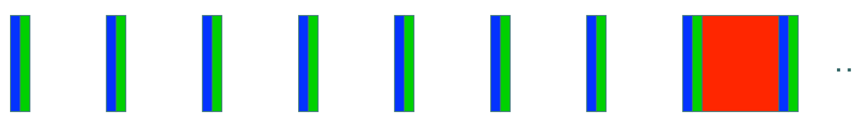
A preferable schedule would space invocations of A and B uniformly in time, as in:



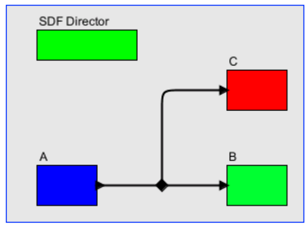
Non-Concurrent Uniformly Timed Schedule



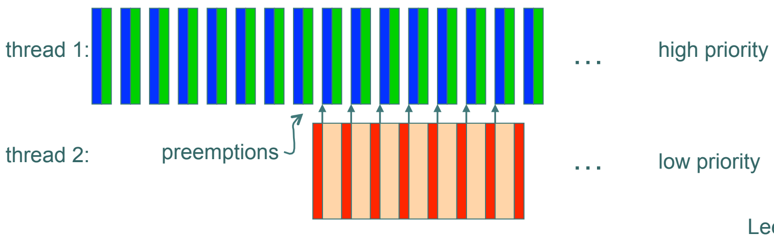
Notice that in this schedule, the rate at which A and B can be invoked is limited by the execution time of C.



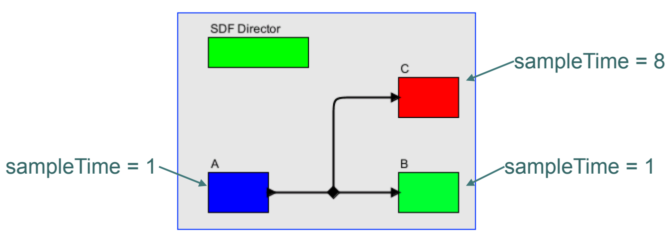
Concurrent Uniformly Timed Schedule: Preemptive schedule



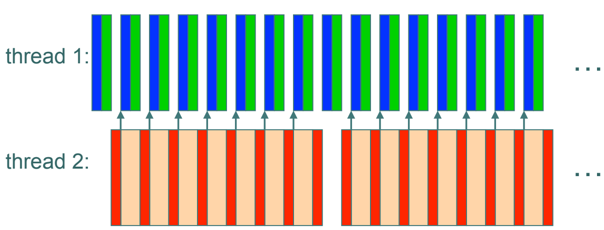
With preemption, the rate at which A and B can be invoked is limited only by total computation:



Ignoring Initial Transients, Abstract to Periodic Tasks

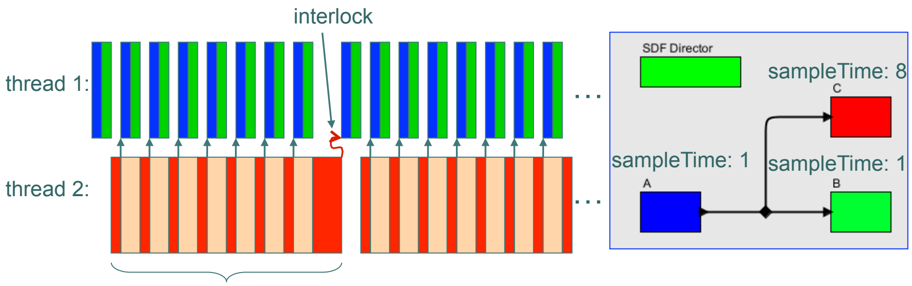


In steady-state, the execution follows a simple periodic pattern:



This follows the principles of rate-monotonic scheduling (RMS).

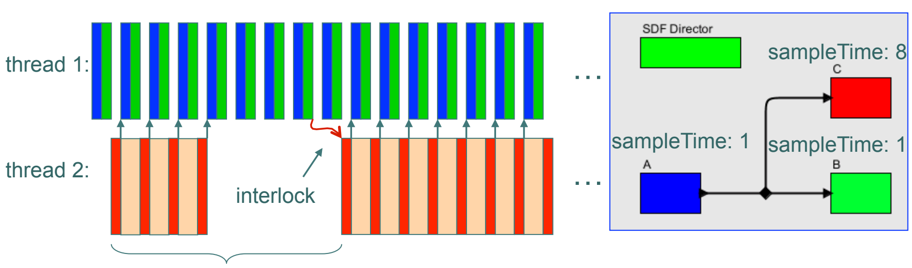
Requirement 1 for Determinacy: Periodicity



With a fixed-length circular buffer, If the execution of C runs longer than expected, data determinacy requires that thread 1 be delayed accordingly. This can be accomplished with semaphore synchronization. But there are alternatives:

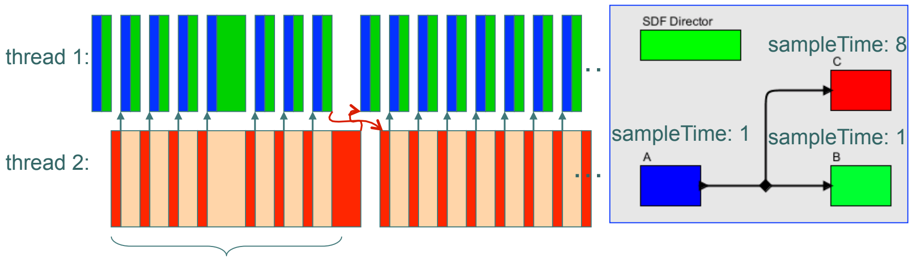
- Throw an exception to indicate timing failure.
- "Anytime" computation: use incomplete results of C

Requirement 1 for Determinacy: Periodicity



If the execution of C runs shorter than expected, data determinacy requires that thread 2 be delayed accordingly. That is, it must not start the next execution of C before the data is available.

Semaphore Synchronization Required Exactly Twice Per Major Period

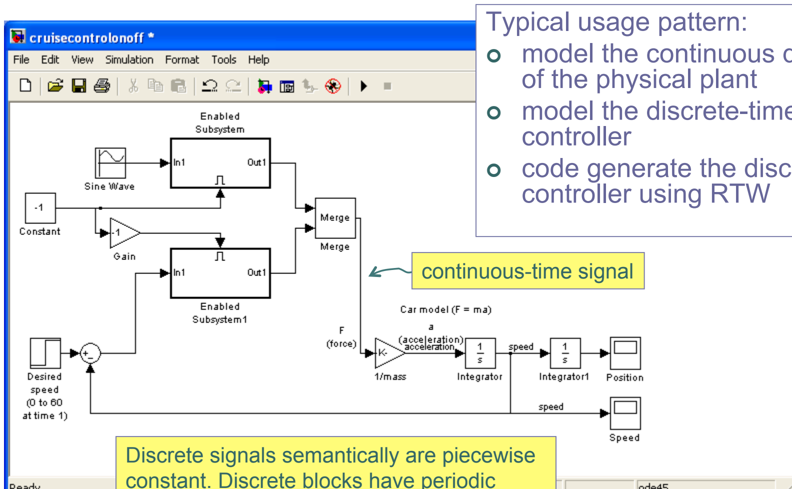


Note that semaphore synchronization is *not* required if actor B runs long because its thread has higher priority. Everything else is automatically delayed.

Simulink and Real-Time Workshop (The MathWorks)

Typical usage pattern:

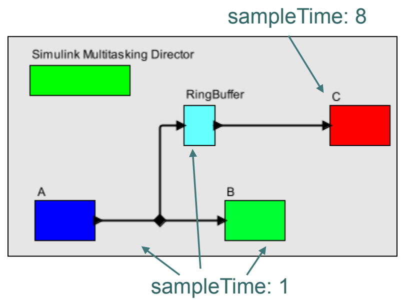
- model the continuous dynamics of the physical plant
- model the discrete-time controller
- code generate the discrete-time controller using RTW



continuous-time signal

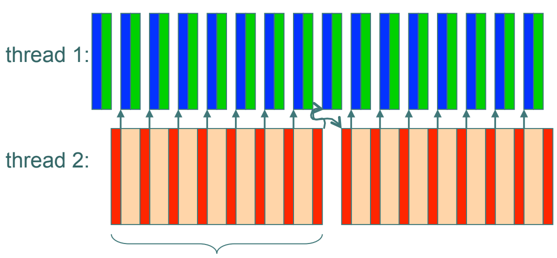
Discrete signals semantically are piecewise constant. Discrete blocks have periodic execution with a specified “sample time.”

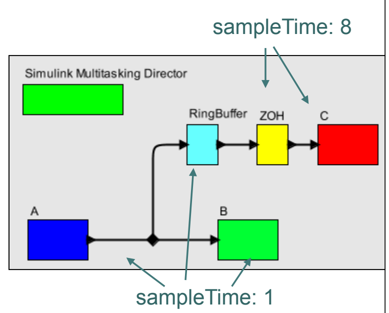
Explicit Buffering is required in Simulink



In Simulink, unlike dataflow, there is no buffering of data. To get the effect of presenting to C 8 successive samples at once, we have to explicitly include a buffering actor that outputs an array.

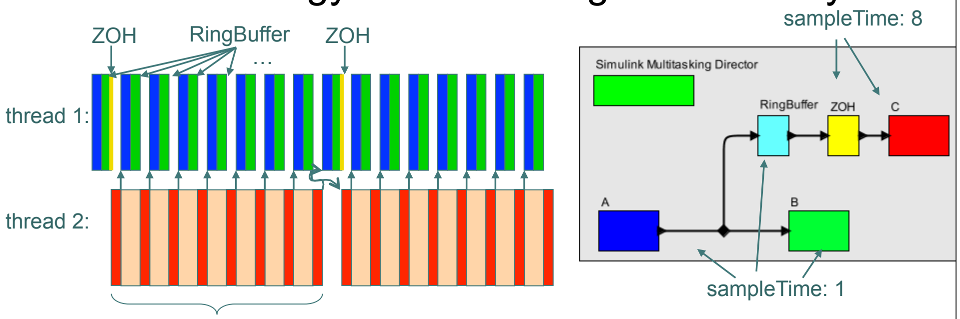
Requirement 2 for Determinacy: Data Integrity During Execution





It is essential that input data remains stable during one complete execution of C, something achieved in Simulink with a zero-order hold (ZOH) block.

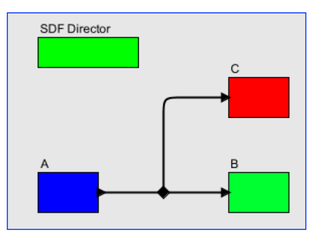
Simulink Strategy for Preserving Determinacy



In "Multitasking Mode," Simulink requires a Zero-Order Hold (ZOH) block at any downsampling point. The ZOH runs at the slow rate, but at the priority of the fast rate. The ZOH holds the input to C constant for an entire execution.

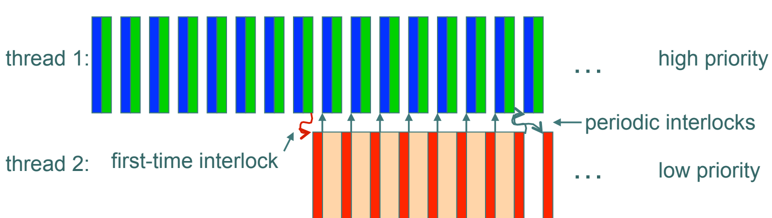
Lee 11: 17

In Dataflow, Interlocks and Built-in Buffering take care of these dependencies



No ZOH block is required!

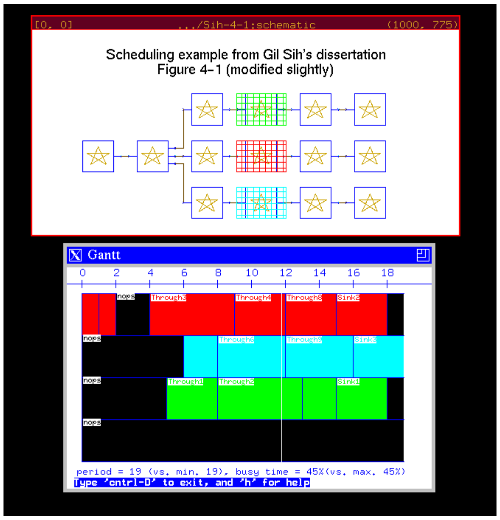
For dataflow, a one-time interlock ensures sufficient data at the input of C:

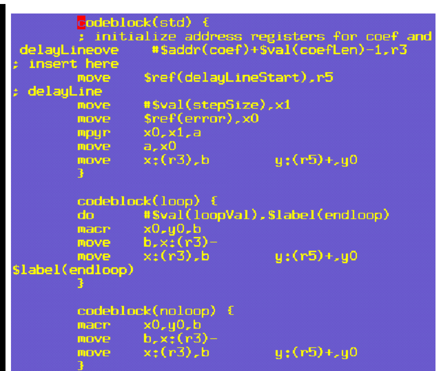


Lee 11: 18

Aside: Ptolemy Classic Code Generator Used Such Interlocks (since about 1990)

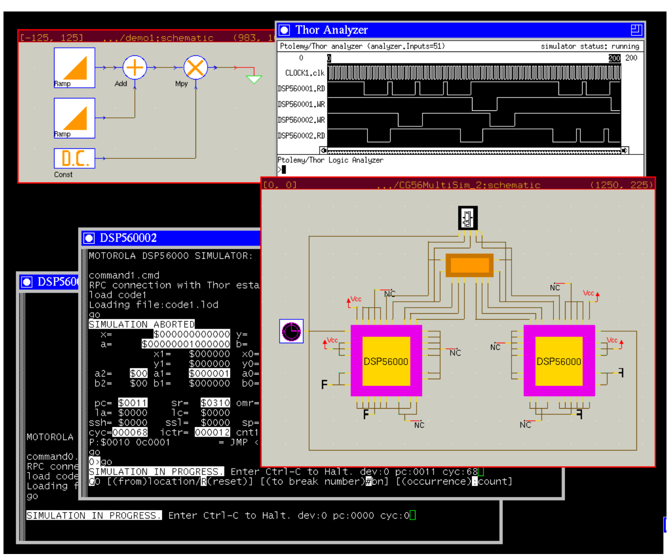
SDF model, parallel schedule, and synthesized DSP assembly code





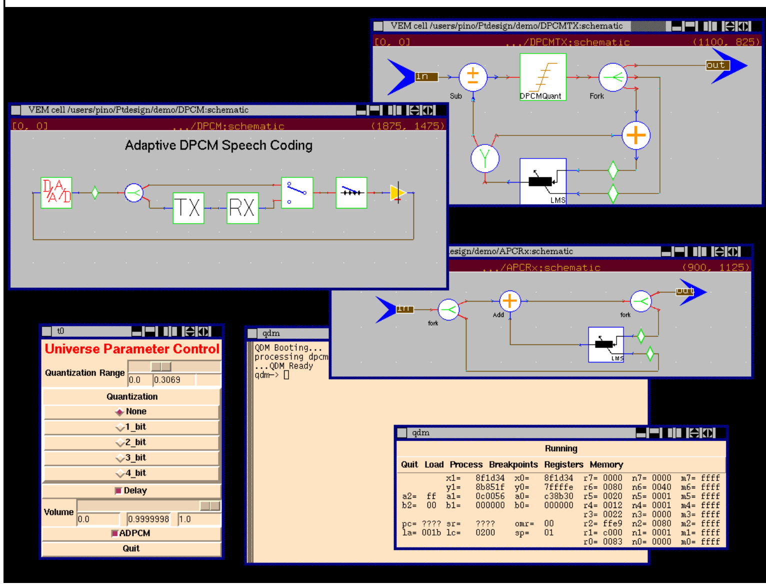
It is an interesting (and rich) research problem to minimize interlocks in complex multirate applications.

Aside: Ptolemy Classic Development Platform (1990)



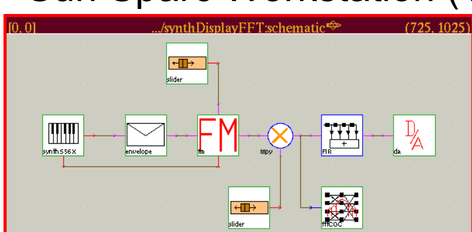
An SDF model, a “Thor” model of a 2-DSP architecture, a “logic analyzer” trace of the execution of the architecture, and two DSP code debugger windows, one for each processor.

Aside: Application to ADPCM Speech Coding (1993)

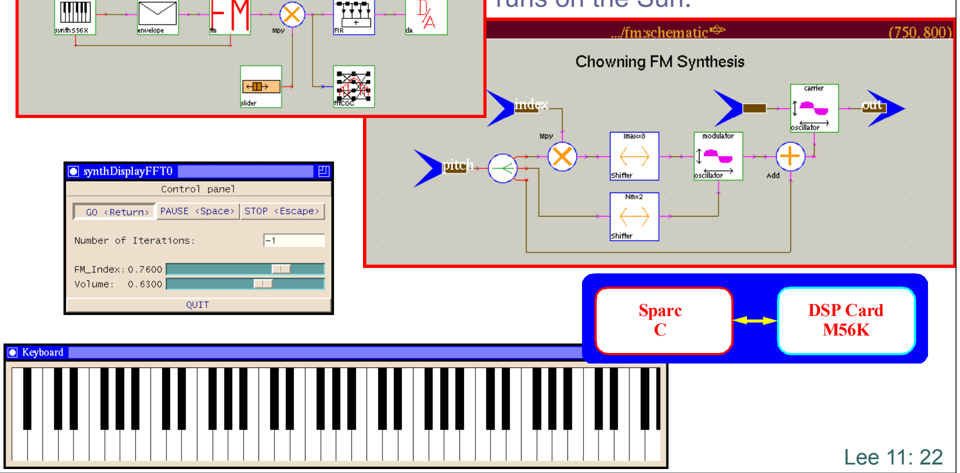


Note updated DSP debugger interface with host/ DSP interaction.

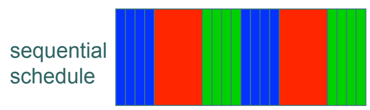
Aside: Heterogeneous Architecture with DSP and Sun Sparc Workstation (1995)

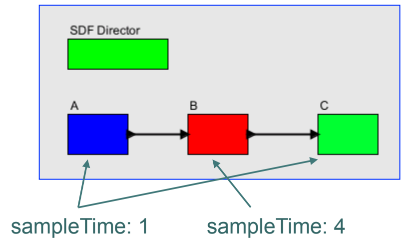


DSP card in a Sun Sparc Workstation runs a portion of a Ptolemy model; the other portion runs on the Sun.



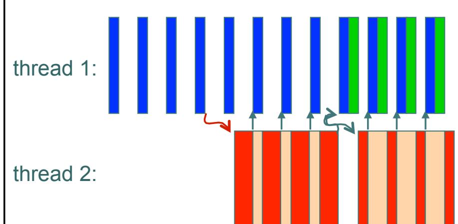
Consider a Low-Rate Actor Sending Data to a High-Rate Actor

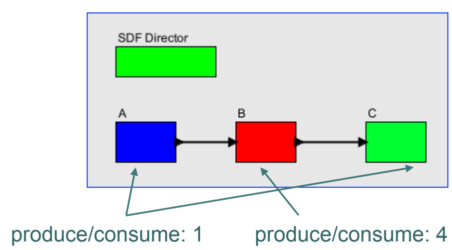




Note that data precedences make it impossible to achieve uniform timing for A and C with the periodic non-concurrent schedule indicated above.

Overlapped Iterations Can Solve This Problem

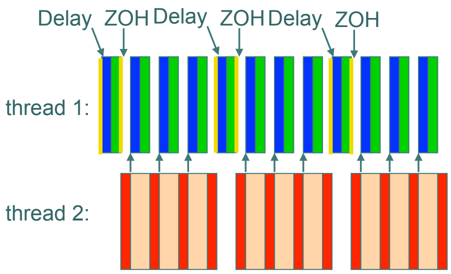


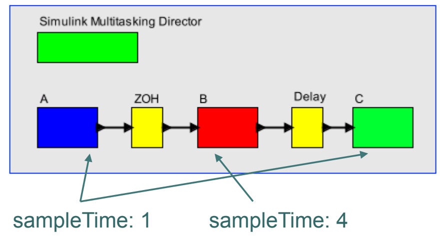


This solution takes advantage of the intrinsic buffering provided by dataflow models.

For dataflow, this requires the initial interlock as before, and the same periodic interlocks.

Simulink Strategy



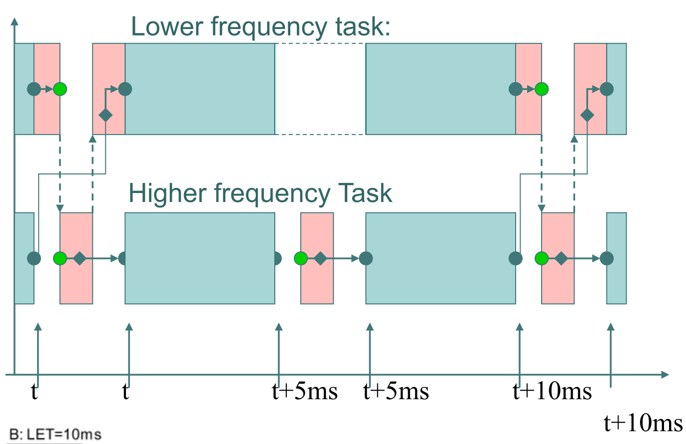


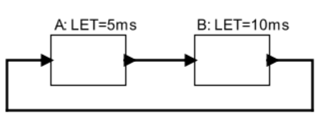
Without buffering, the Delay provides just one initial sample to C (there is no buffering in Simulink). The Delay and ZOH run at the rates of the slow actor, but at the priority of the fast ones.

Part of the objective seems to be to have no initial transient. Why?

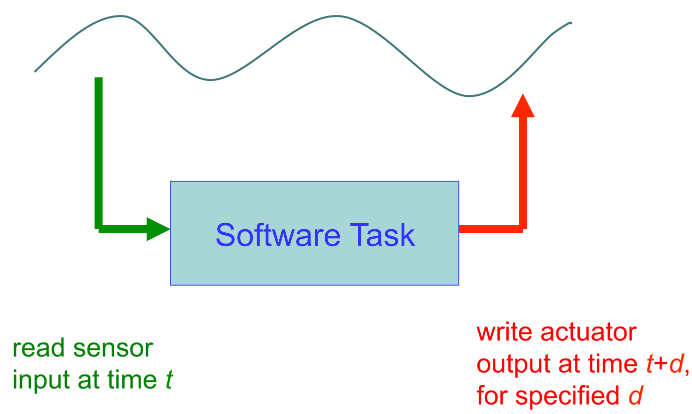
Time-Triggered Models and Logical Execution Time (LET)

In time-triggered models (e.g. Giotto, TDL, Simulink/RTW), each actor has a logical execution time (LET). Its actual execution time always appears to have taken the time of the LET.





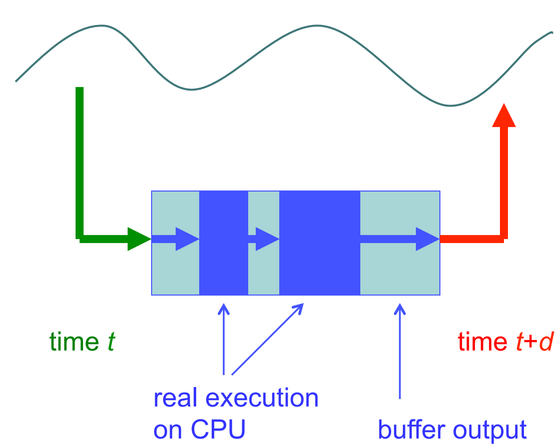
The LET (Logical Execution Time) Programming Model



Examples: Giotto, TDL,

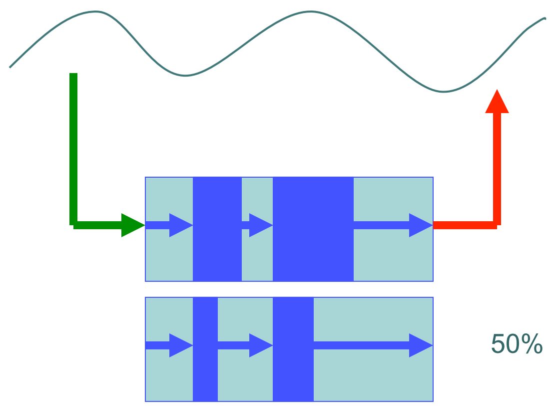
Slide from Tom Henzinger

The LET (Logical Execution Time) Programming Model



Slide from Tom Henzinger

Portability

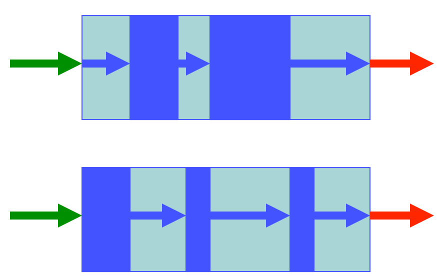


50% CPU speedup

Slide from Tom Henzinger

Lee 11: 29

Composability



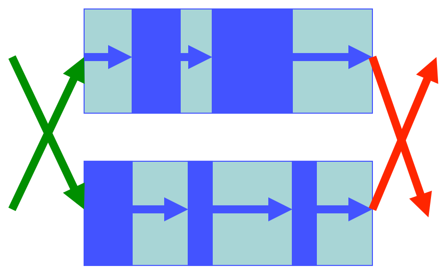
Task 1

Task 2

Slide from Tom Henzinger

Lee 11: 30

Determinism

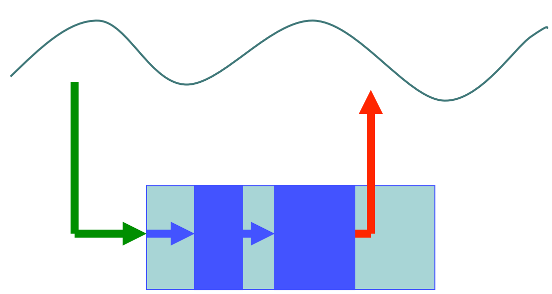


Timing predictability: minimal jitter
Function predictability: no race conditions

Slide from Tom Henzinger

Lee 11: 31

Contrast LET with Standard Practice

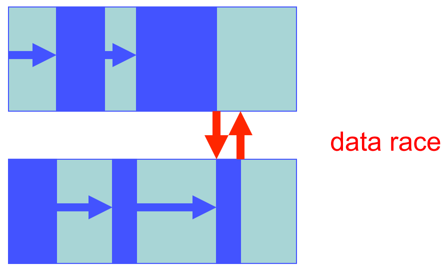


make output available
as soon as ready

Slide from Tom Henzinger

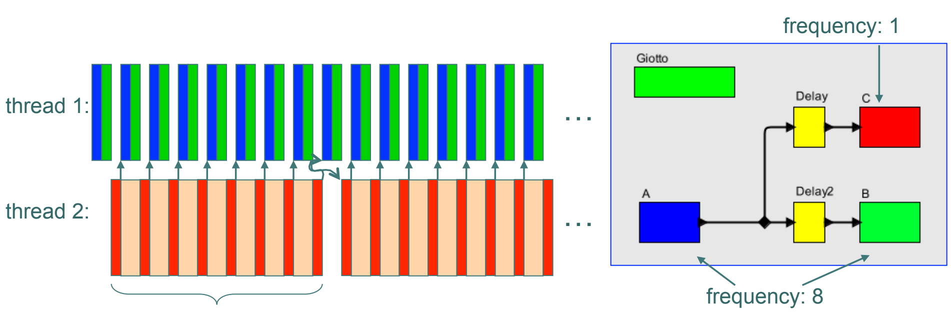
Lee 11: 32

Contrast LET with Standard Practice



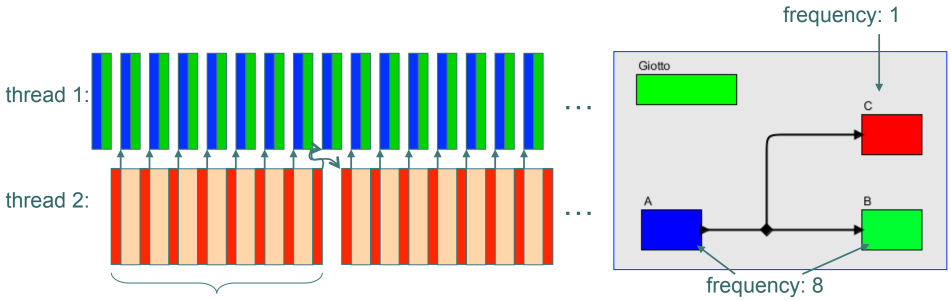
Slide from Tom Henzinger

Giotto Strategy for Preserving Determinacy



First execution of C operates on initial data in the delay. Second execution operates on the result of the 8-th execution of A.

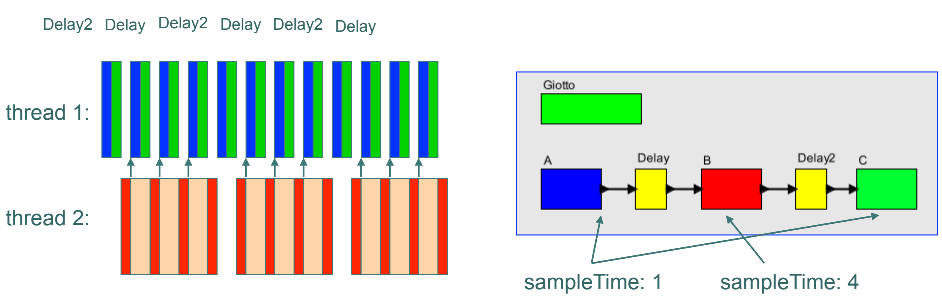
Giotto: A Delay on Every Arc



Since Giotto has a delay on every connection, there is no need to show it. It is implicit.

Is a delay on every arc a good idea?

Giotto Strategy for the Pipeline Example



Giotto uses delays on all connections. The effect is the same, except that there is one additional sample delay from input to output.

Discussion Questions

- What about more complicated rate conversions (e.g. a task with sampleTime 2 feeding one with sampleTime 3)?
- What are the advantages and disadvantages of the Giotto delays?
- Could concurrent execution be similarly achieved with synchronous languages?
- How does concurrent execution of dataflow compare to Giotto and Simulink?
- Which of these approaches is more attractive from the application designer's perspective?
- How can these ideas be extended to non-periodic execution? (modal models, Timed Multitasking, xGiotto, Ptides)