Department of Computer Science
National Tsing Hua University

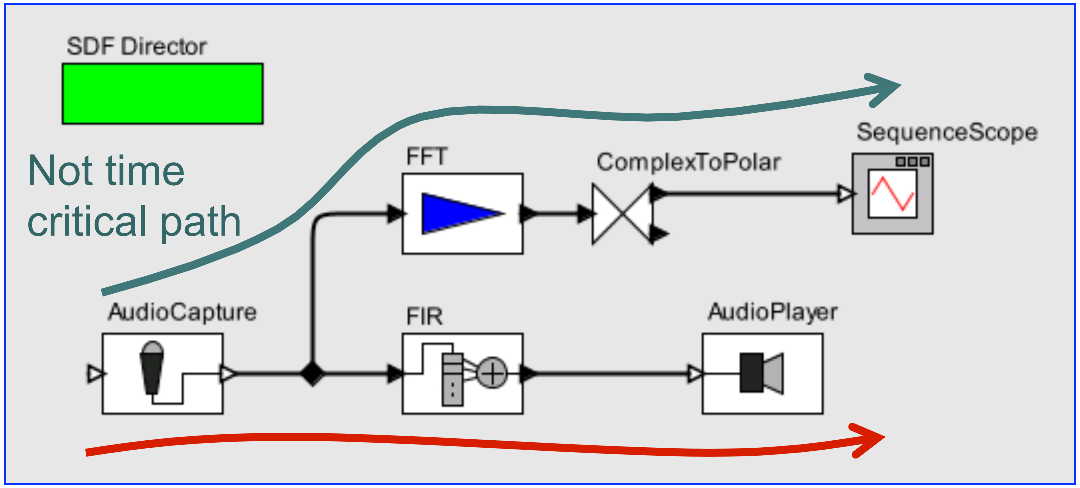
CS 5244: Introduction to Cyber Physical Systems

Unit 18: Dataflow Models 1 (Ch. 6)

Instructor: Cheng-Hsin Hsu

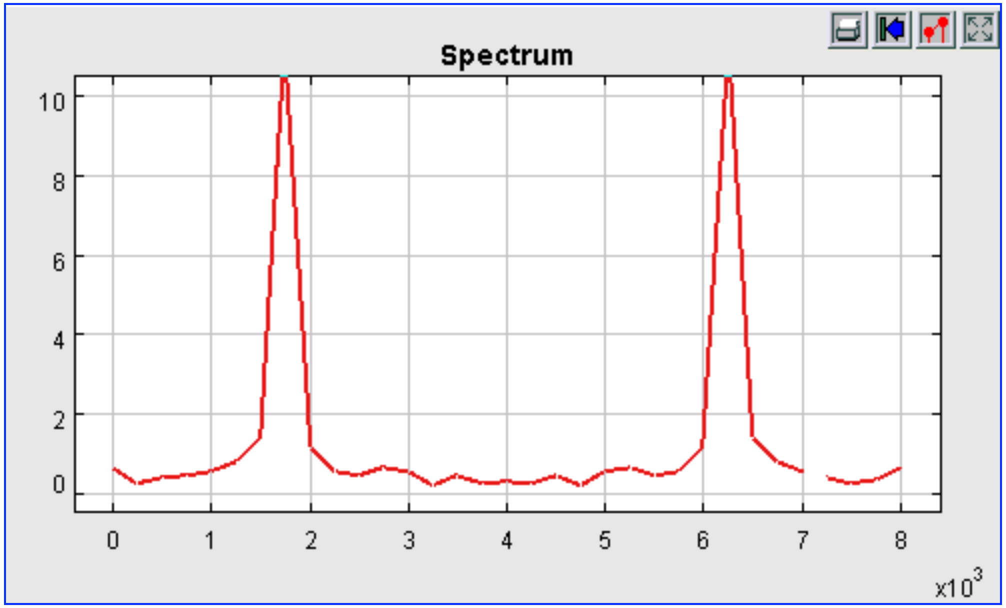
Acknowledgement: The instructor thanks Profs. Edward A. Lee & Sanjit A. Seshia at UC Berkeley for sharing their course materials

Simple Example: Spectrum Analysis

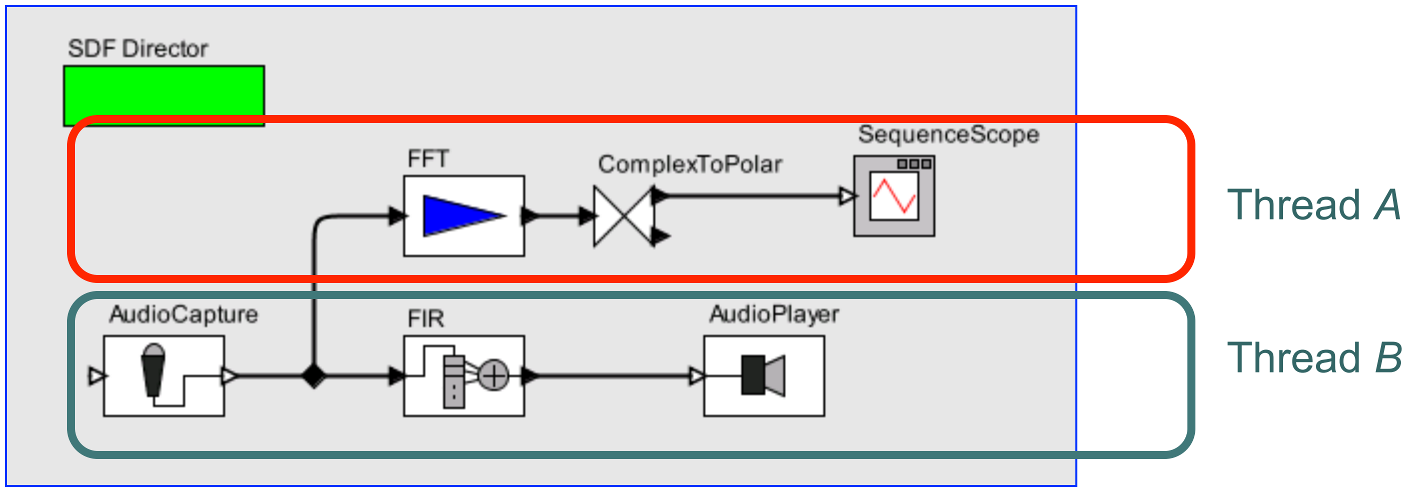


Time critical path

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



A Solution with Threads



Time critical path

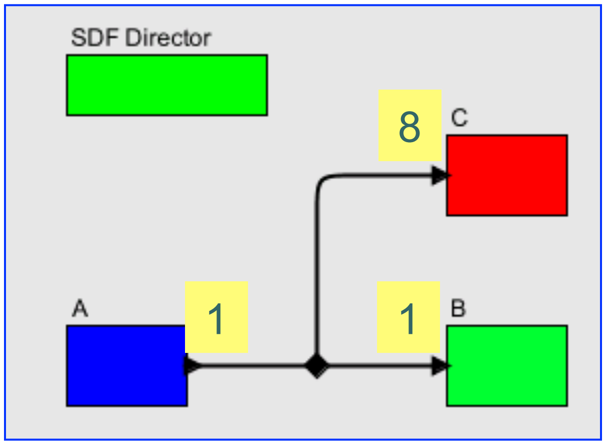
Create two threads:

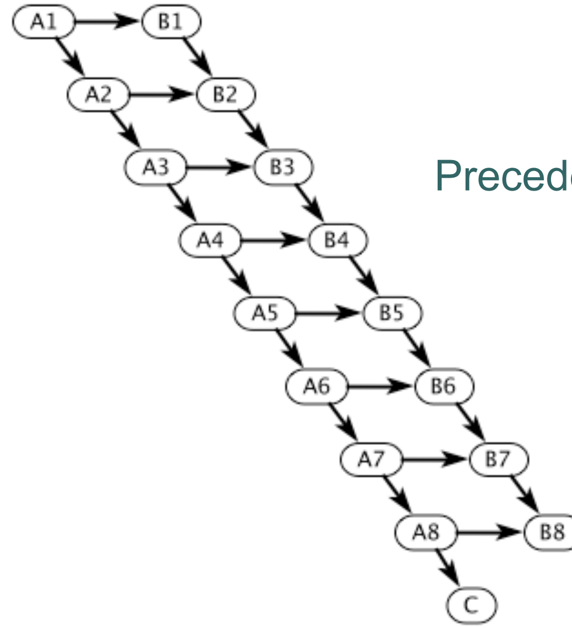
- A has low priority
- B has high priority

Why?

- RMS does not apply because there are dependencies.
- EDF with precedences applies and is optimal w.r.t. feasibility, except for how to assign deadlines.
- How to implement the communication between threads?

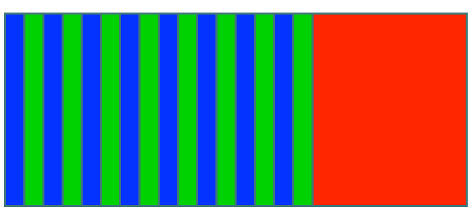
Abstracted Version of the Spectrum Example: EDF scheduling





Precedence graph

Suppose that C requires 8 data values from A to execute. Suppose further that C takes much longer to execute than A or B. EDF schedule:



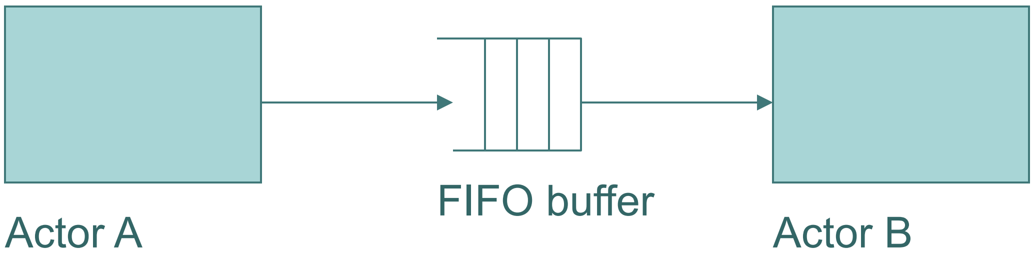
schedule

FIXME: Pthreads Buffer Implementation

Version 1: Without arrivals

Version 2: With arrivals

Dataflow Models



Buffered communication between concurrent components (*actors*).

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

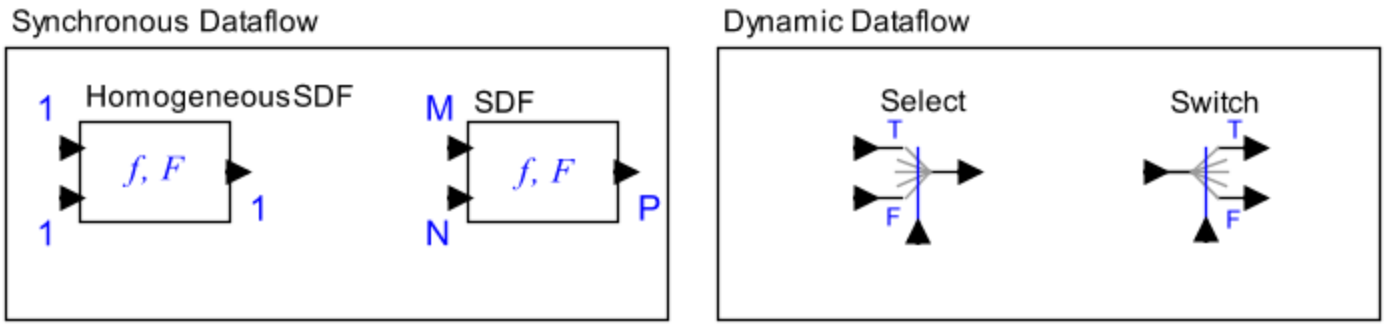
Dynamic scheduling: Each time dispatch() is called, determine which actor can fire (or is firing) and choose one.

May need to implement interlocks in the buffers.

Streams: The basis for Dataflow models

A stream is a signal $x\colon \mathbb{N} \to R$, for some set R. There is not necessarily any relationship between $x(n)$, an element in a stream, and $y(n)$, an element in another stream. Unlike discrete-time models or SR models, they are not "simultaneous."

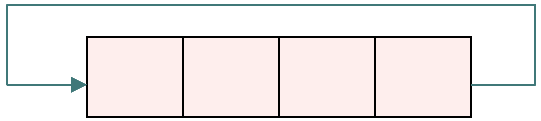
Dataflow



Each signal has form $x : \mathbb{N} \to R$. The function F maps such signals into such signals. The function f (the "firing function") maps prefixes of these signals into prefixes of the output. Operationally, the actor *consumes* some number of tokens and *produces* some number of tokens to construct the output signal(s) from the input signal(s). If the number of tokens consumed and produced is a constant over all firings, then the actor is called a *synchronous dataflow* (SDF) actor.

Firing rules: the number of tokens required to fire an actor.

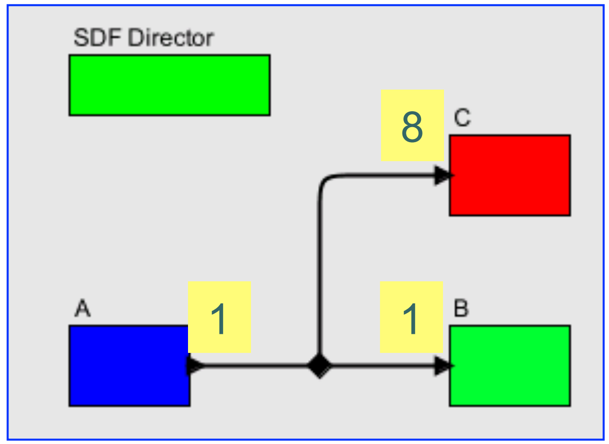
Buffers for Dataflow



- 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.

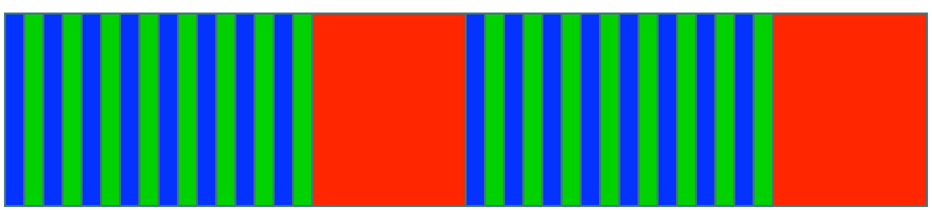
Abstracted Version of the Spectrum Example: Non-preemptive scheduling

Is this dataflow model dynamic?
Is it homogeneous?



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:



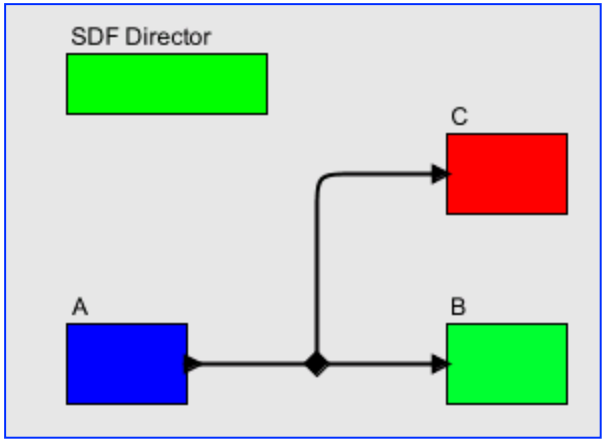
…

This suffers from jitter.

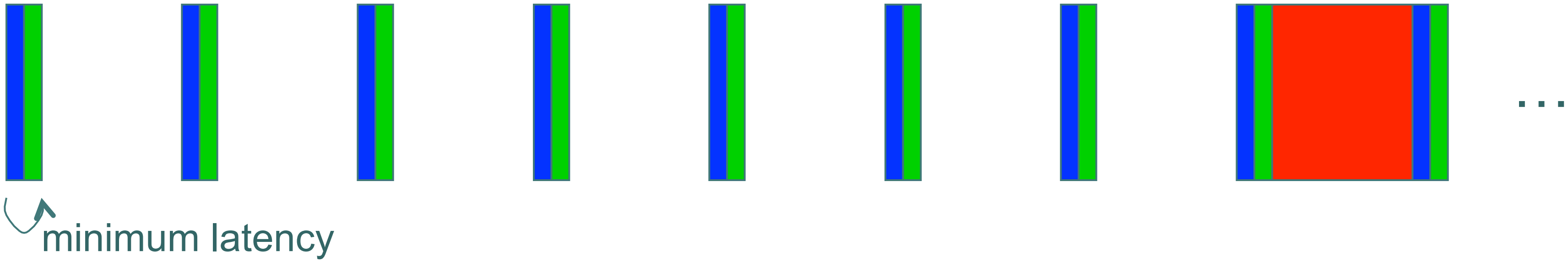
schedule

Note that each period is EDF, but it fails to account for arrivals of tasks.

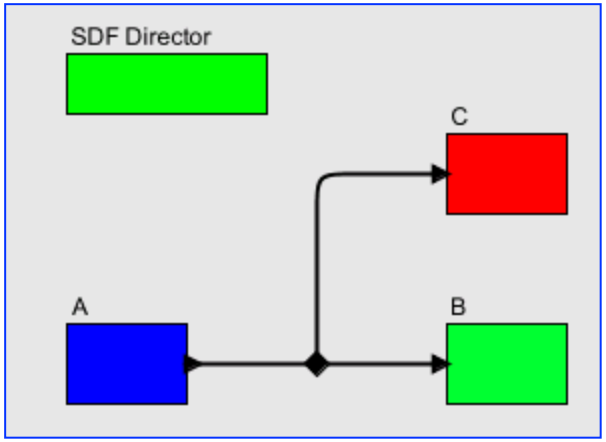
Uniformly Timed Schedule



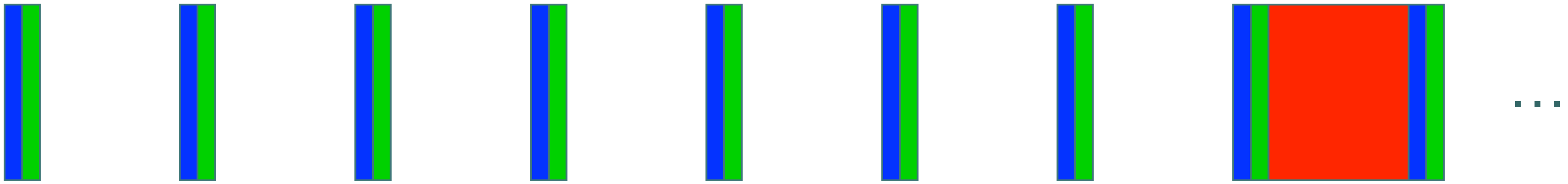
A preferable schedule would space invocations of A and B uniformly in time, as in:



Non-Concurrent Uniformly Timed Schedule

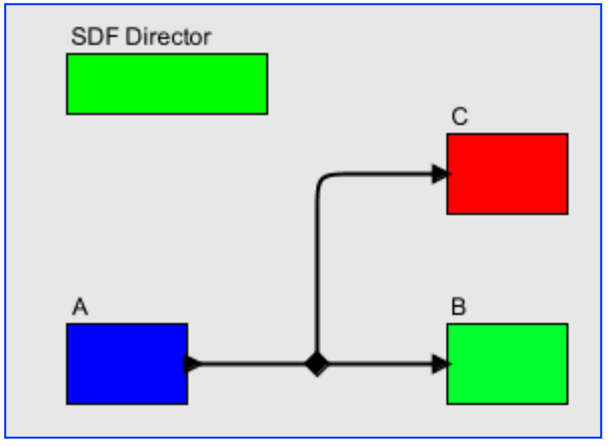


Notice that in this schedule, the rate at which A and B can be invoked is limited by the execution time of C.

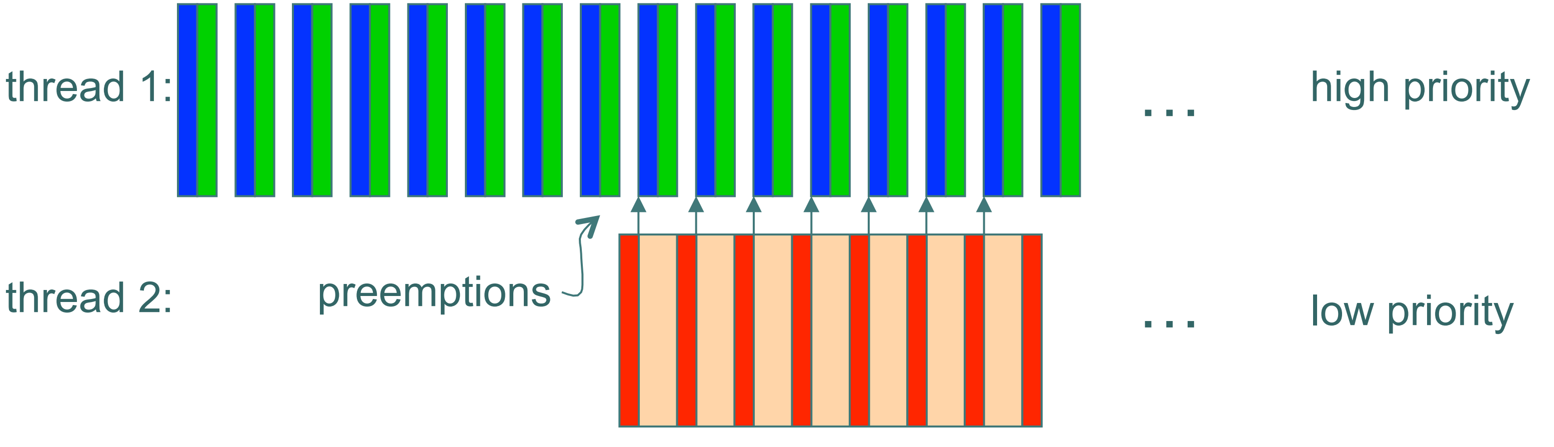


No jitter, but utilization is poor.

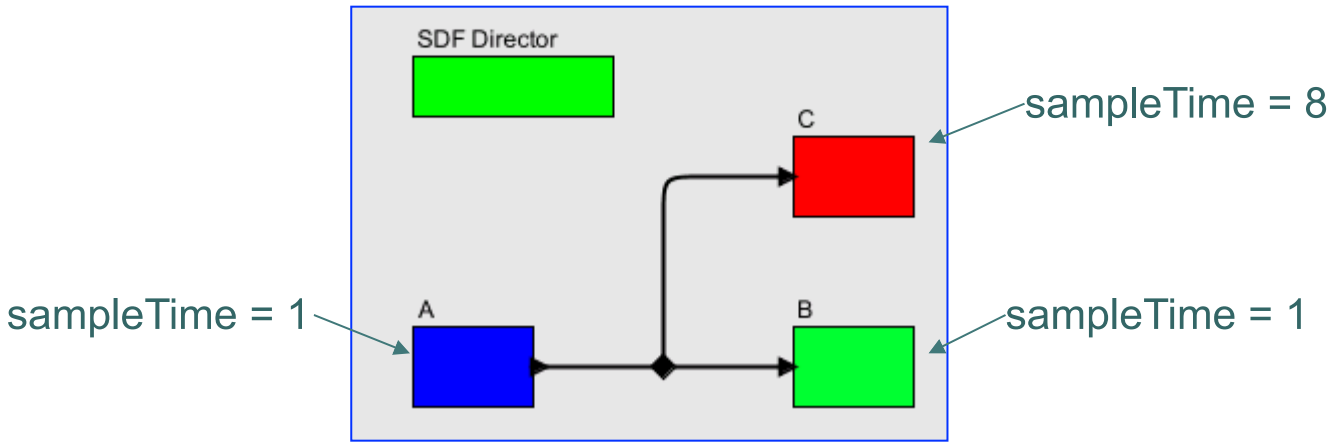
Concurrent Uniformly Timed Schedule: Preemptive schedule



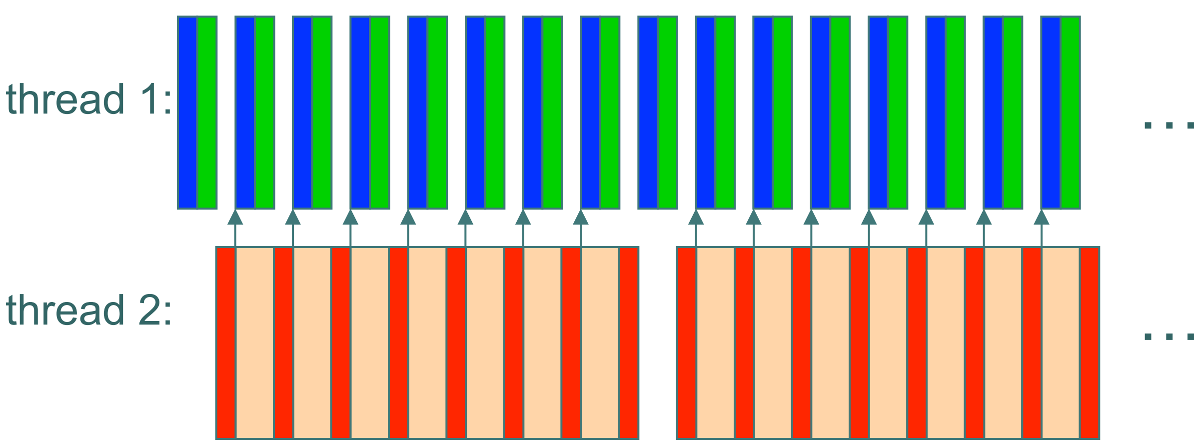
With preemption, the rate at which A and B can be invoked is limited only by total computation:



Ignoring Initial Transients, Abstract to Periodic Tasks

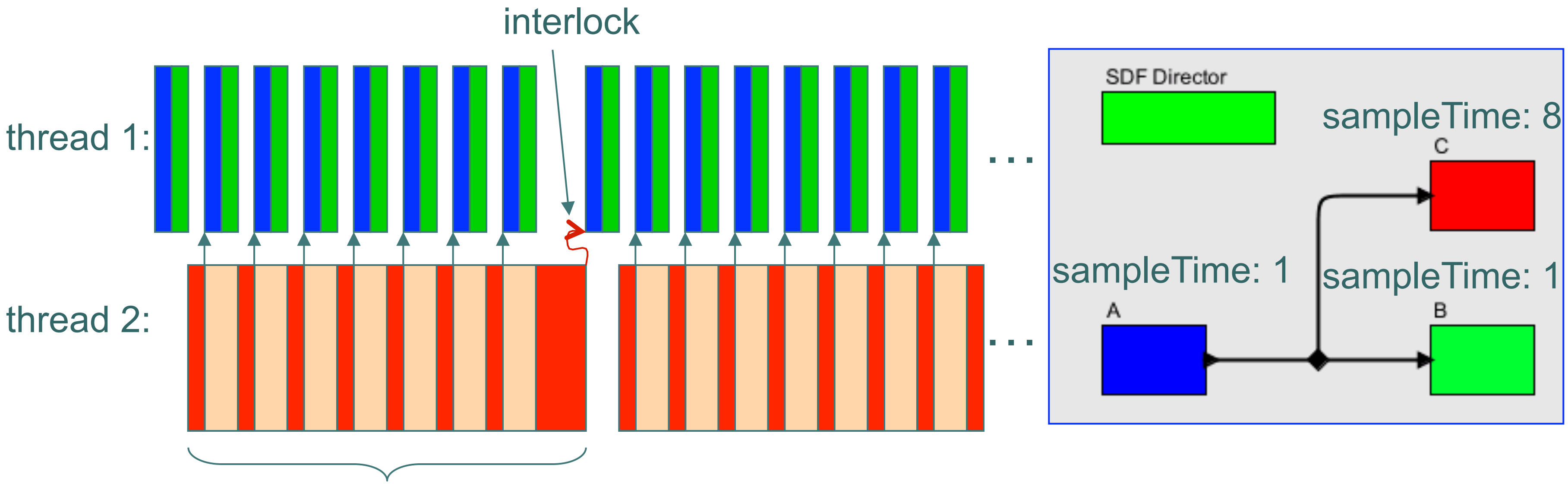


In steady-state, the execution follows a simple periodic pattern:



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

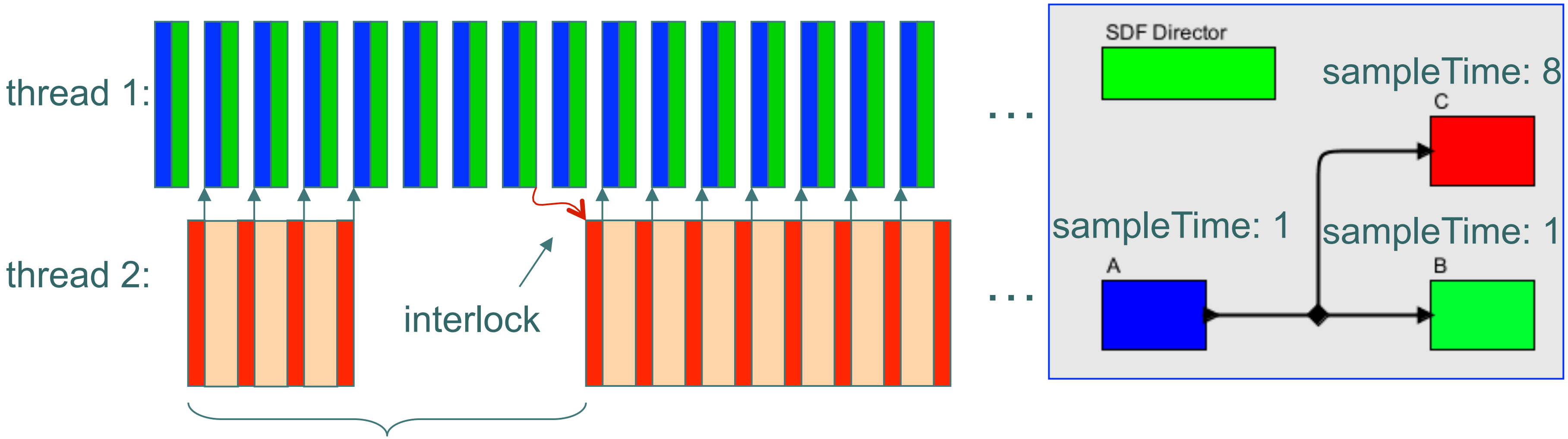
Requirement 1 for Determinacy: Periodicity



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:

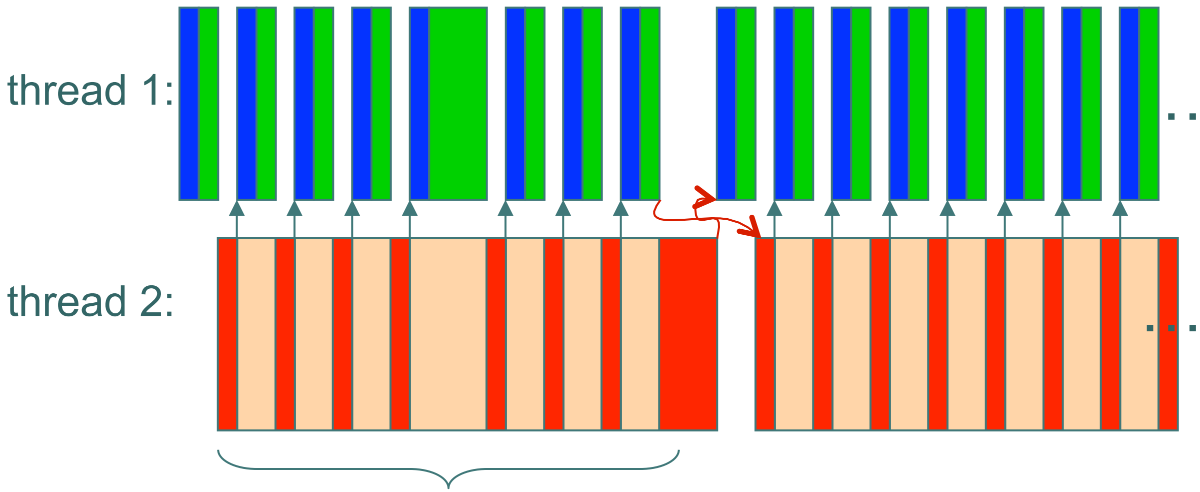
- Throw an exception to indicate timing failure.
- “Anytime” computation: use incomplete results of C

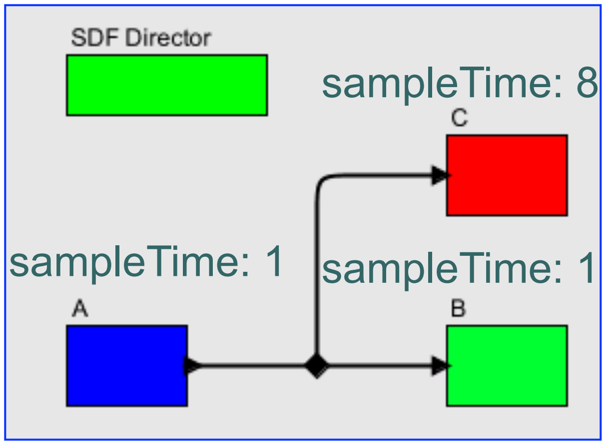
Requirement 1 for Determinacy: Periodicity



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.

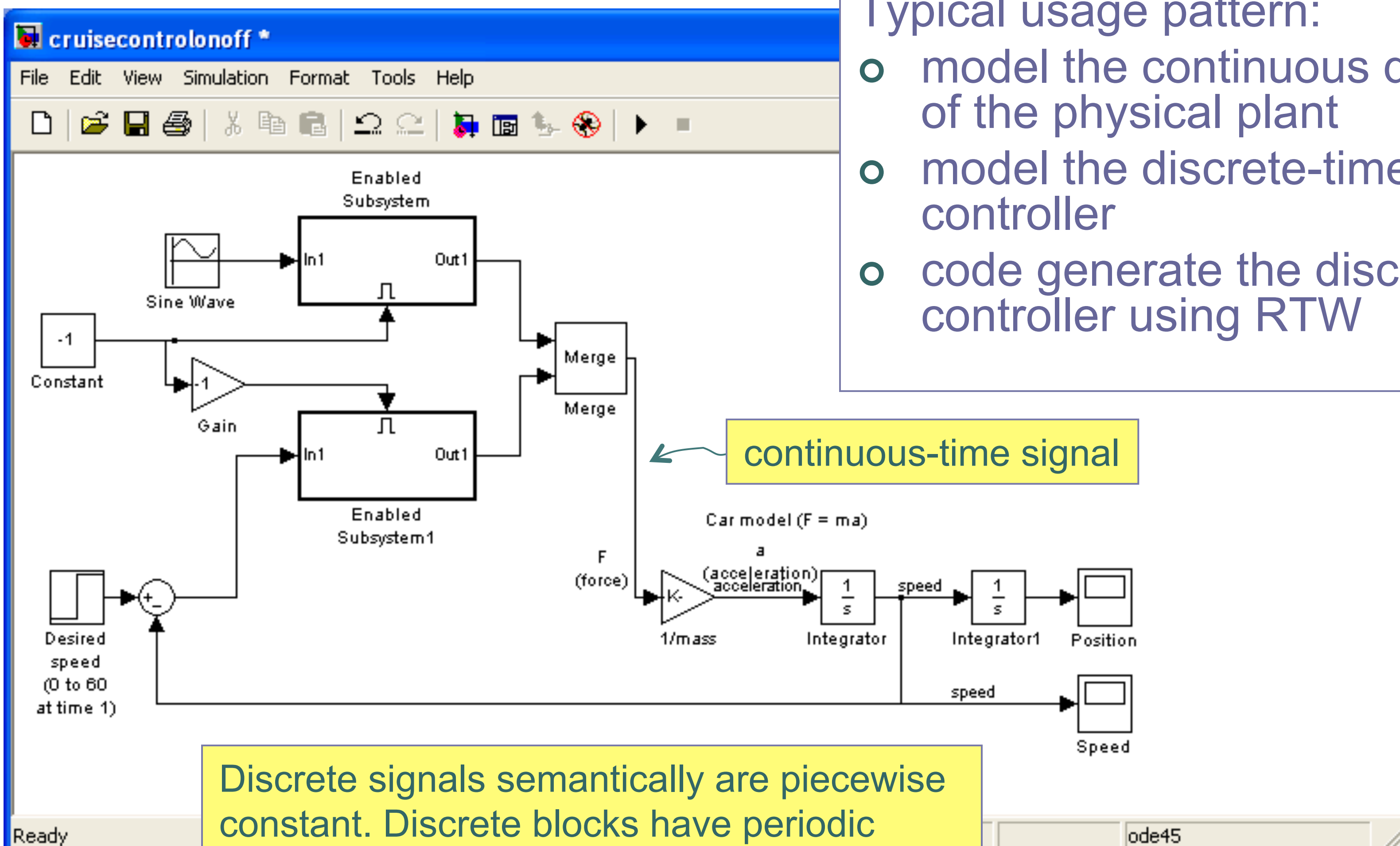
Semaphore Synchronization Required Exactly Twice Per Major Period





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)



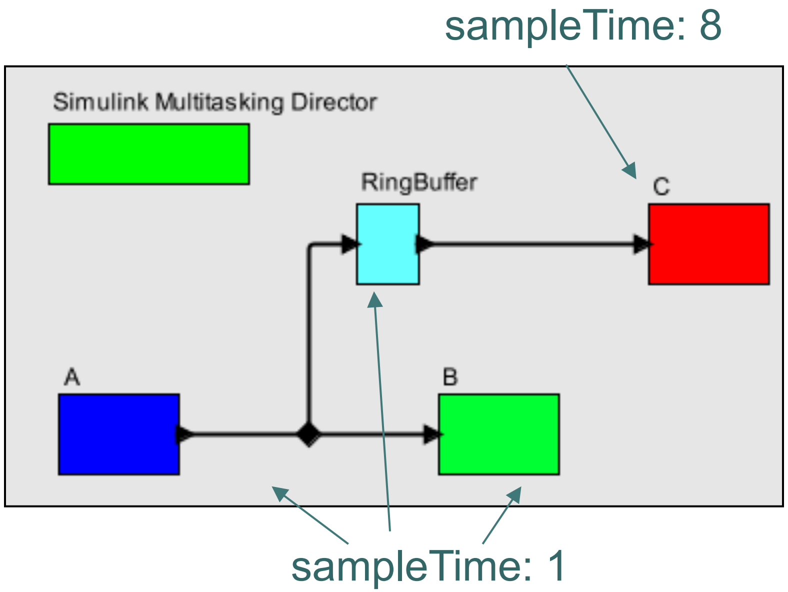
Typical usage pattern:

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

continuous-time signal

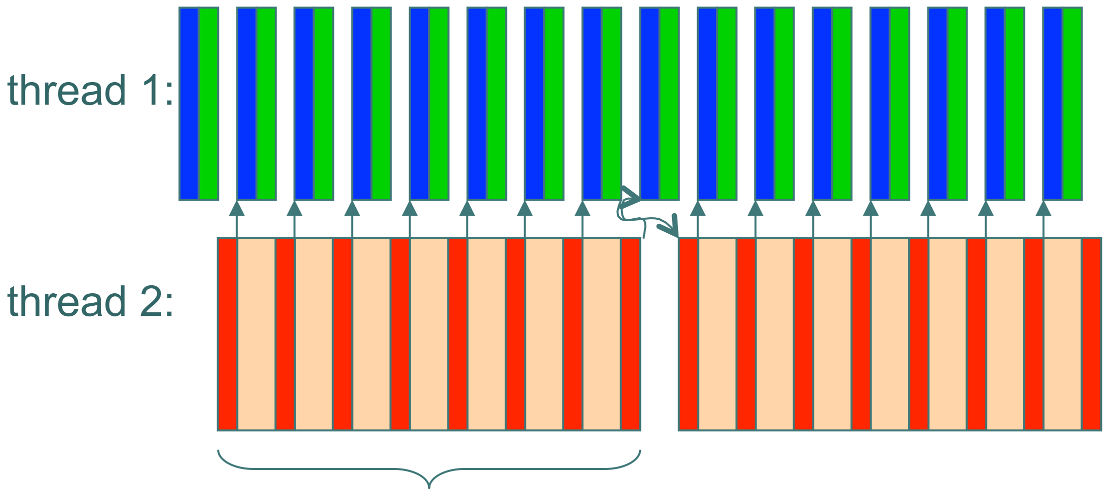
Discrete signals semantically are piecewise constant. Discrete blocks have periodic execution with a specified "sample time."

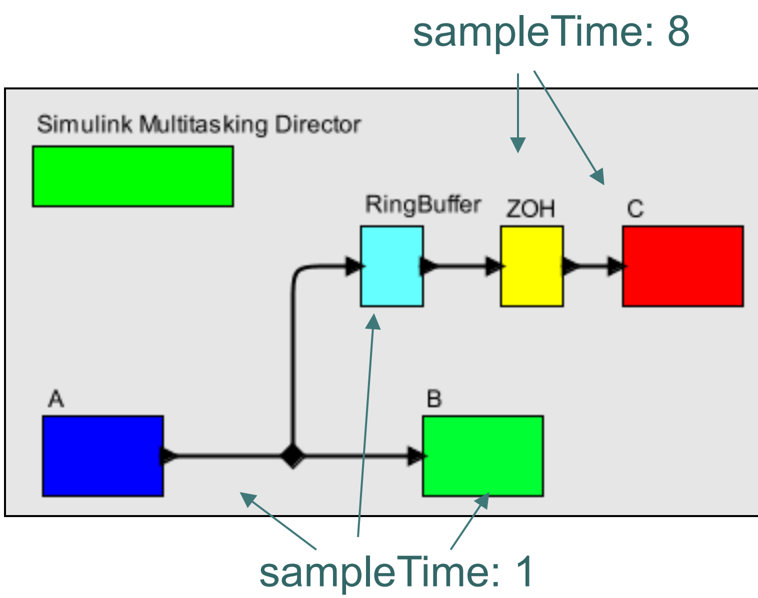
Explicit Buffering is required in Simulink



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.

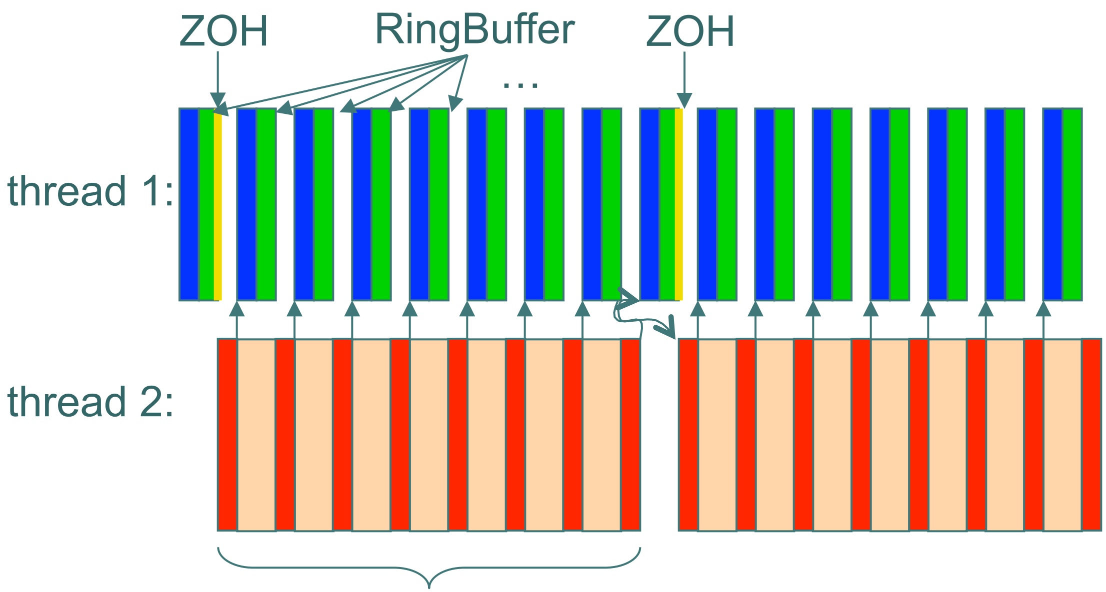
Requirement 2 for Determinacy: Data Integrity During Execution

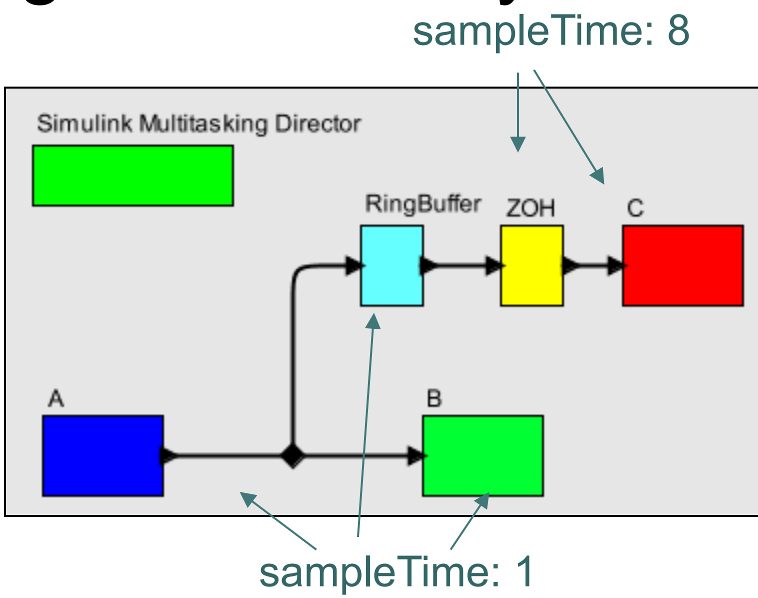




It is essential that input data remains stable during one complete execution of C, something achieved in Simulink with a zero-order hold (ZOH) block.

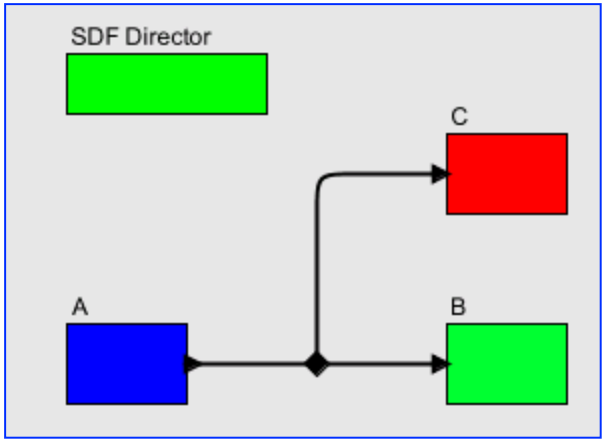
Simulink Strategy for Preserving Determinacy





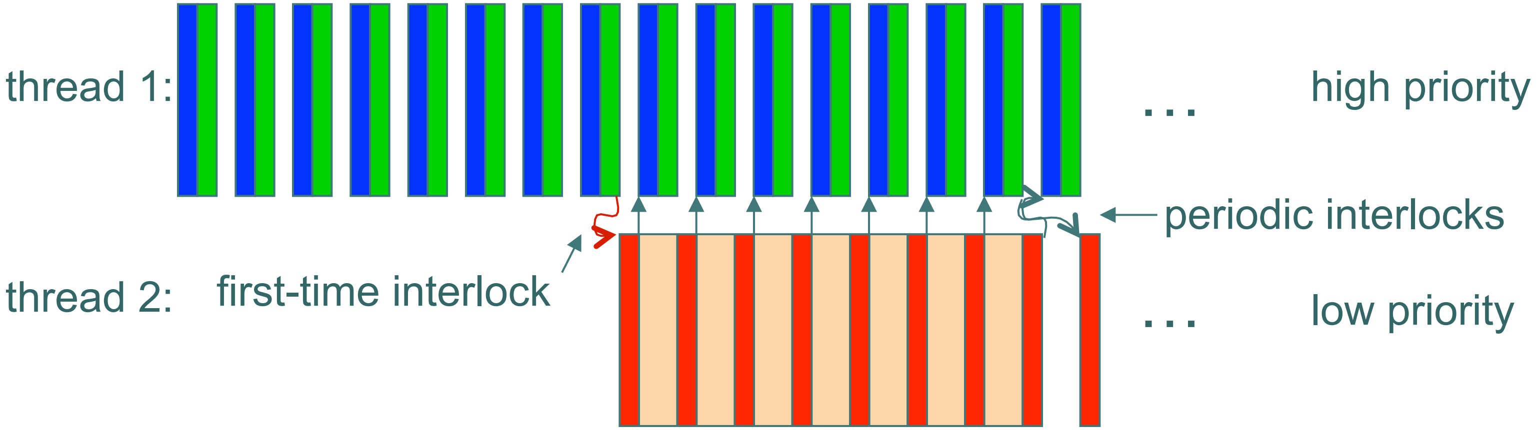
In "Multitasking Mode," Simulink requires a Zero-Order Hold (ZOH) block at any downsampling point. The ZOH runs at the slow rate, but at the priority of the fast rate. The ZOH holds the input to C constant for an entire execution.

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



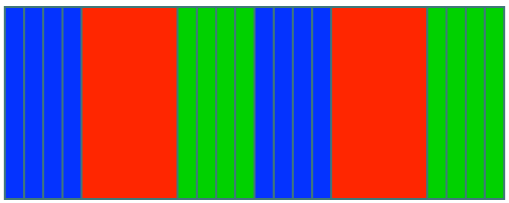
No ZOH block is required!

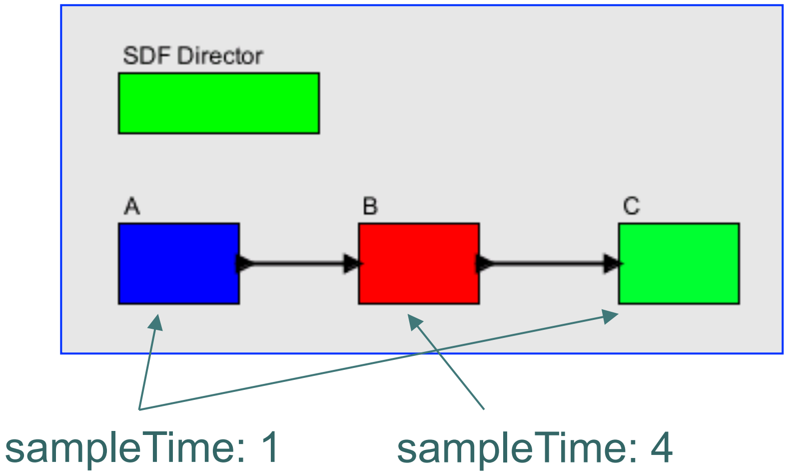
For dataflow, a one-time interlock ensures sufficient data at the input of C:



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

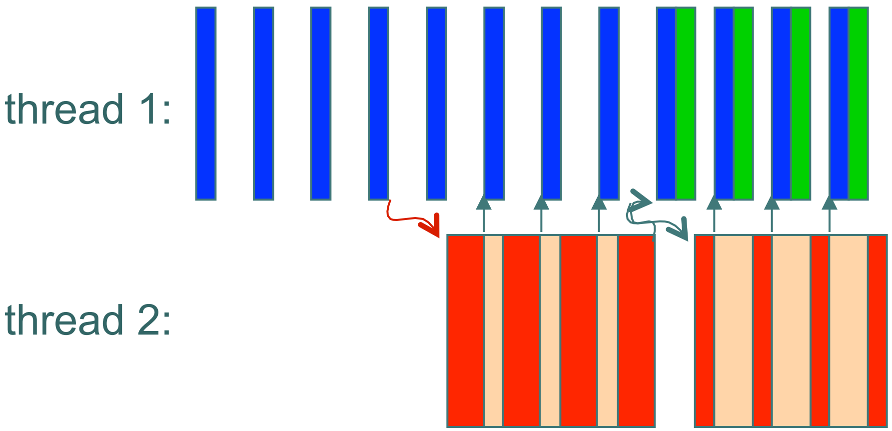
sequential schedule

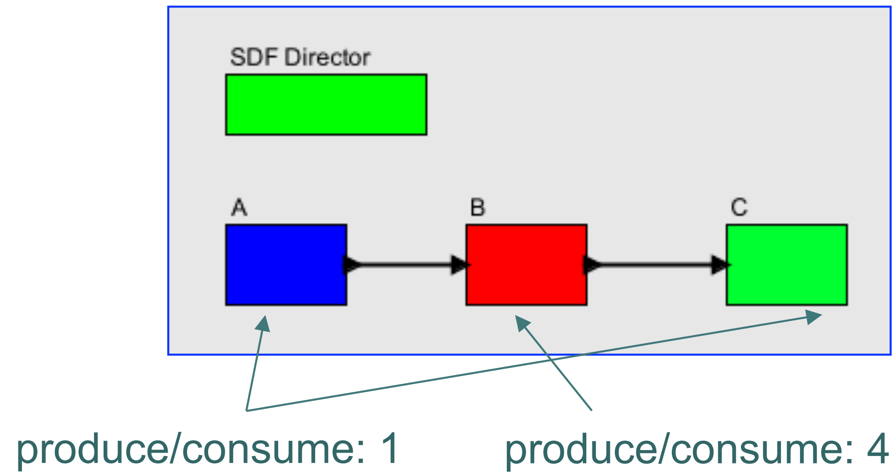




Note that data precedences make it impossible to achieve uniform timing for A and C with the periodic non-concurrent schedule indicated above.

Overlapped Iterations Can Solve This Problem

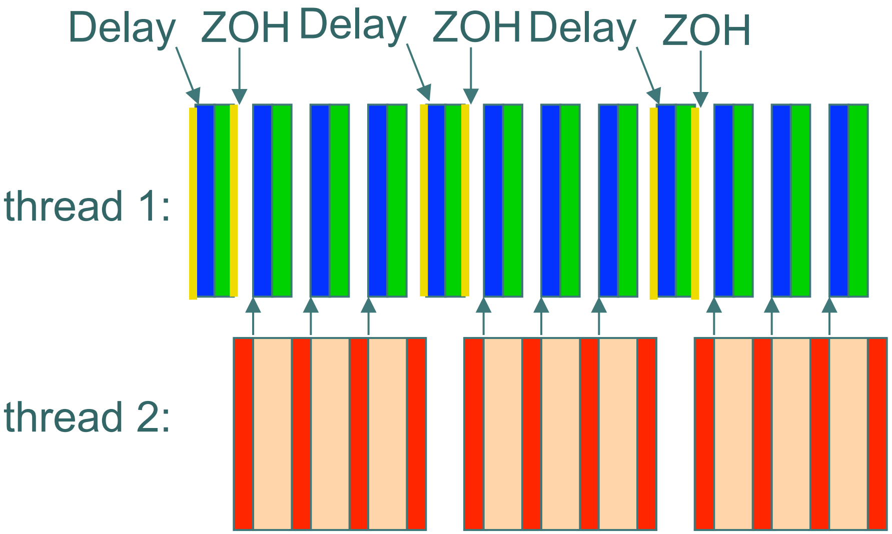


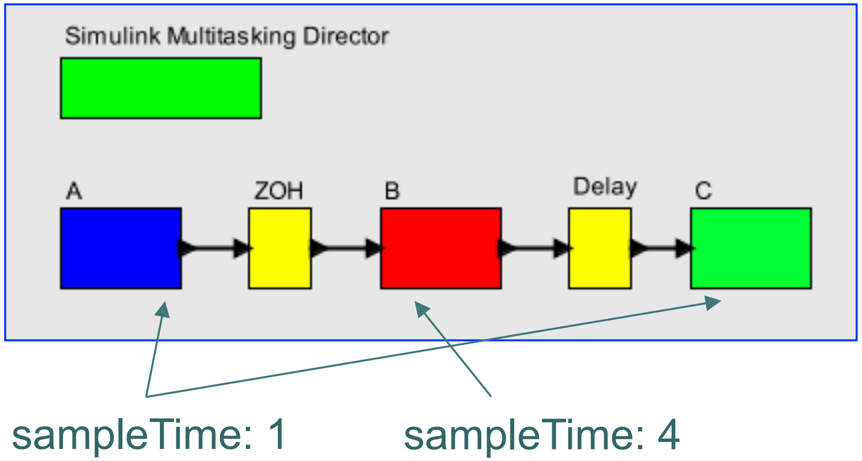


This solution takes advantage of the intrinsic buffering provided by dataflow models.

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?

Discussion Questions

- What about more complicated rate conversions (e.g. a task with sampleTime 2 feeding one with sampleTime 3)?
- How can these ideas be extended to non-periodic execution?