[dcc]

CONCURRENT MODELS OF COMPUTATION

Pedro Brandão, Sérgio Crisóstomo
Sistemas Embutidos 2020/21

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

1

[dcc]

References

- Slides are from Edward A. Lee & Sanjit Seshia, UC Berkeley, EECS 149 Fall 2013
  - *Copyright © 2008-date, Edward A. Lee & Sanjit A. Seshia, All rights reserved*
- Petri networks' slides are from Reinhard von Hanxleden, Christian-Albrechts-Universität zu Kiel
- Time triggered slides are from Thomas A. Henzinger, IST Austria

U.PORTO

2

[dcc]

DATAFLOW MODELS OF COMPUTATION

U.PORTO

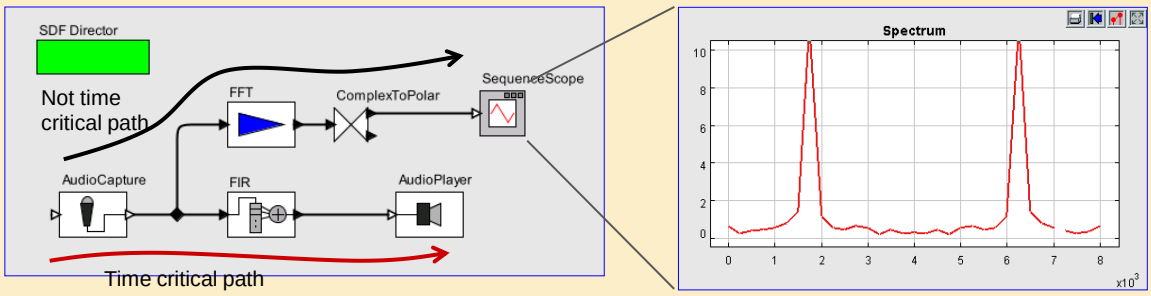
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

3

[dcc]

Simple Example: Spectrum Analysis

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



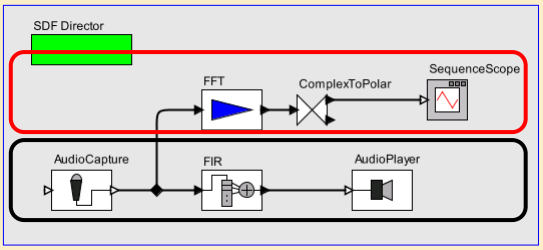
U.PORTO

4

4

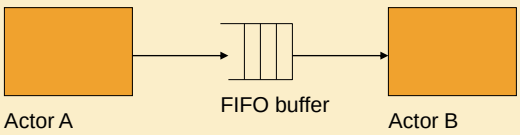
A Solution with Threads

- Create two threads:
  - *A has low priority*
  - *B has high priority*
- Why?
- How to implement the communication between threads?



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.

[dcc]

Streams: The basis for Dataflow models

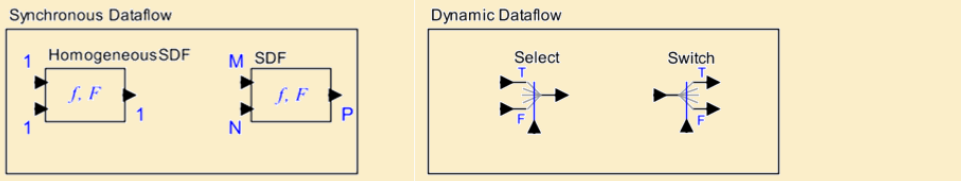
- A stream is a signal $x: \mathbb{N} \rightarrow R$ some set R.
- There is not necessarily any relationship between $x(n)$, an element in a stream, and $y(n)$, an element in another stream.
- Unlike discrete-time models or SR models
  - *Streams are not simultaneous*

U.PORTO

[dcc]

Dataflow

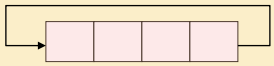
Firing rules: the number of tokens required to fire an actor.



- Each signal has the form $x: \mathbb{N} \rightarrow R$
- 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 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.

U.PORTO

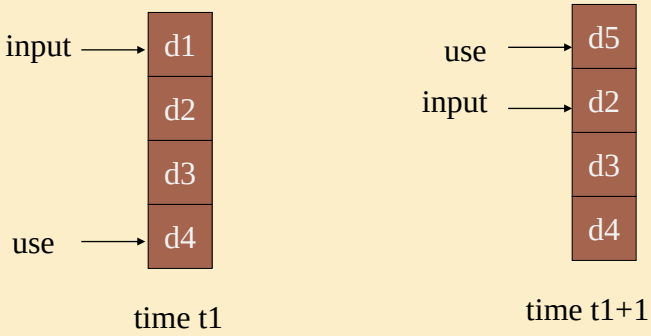
Buffers for Dataflow



- Unbounded buffers require memory allocation and de-allocation 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 before each read.*
  - *A fill count `n` (unsigned number) indicates nr of data items in 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`.*

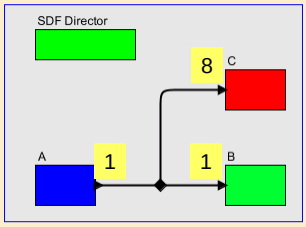
Circular buffers

- Indexes locate currently used data, current input data:



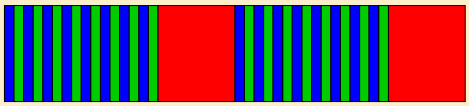
[dcc]

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:

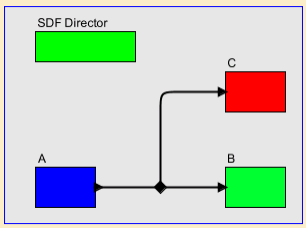
schedule  …

This suffers from jitter.

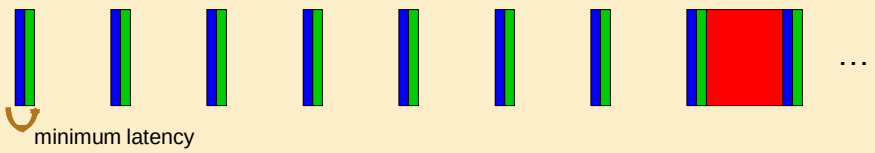
U.PORTO

[dcc]

Uniformly Timed Schedule



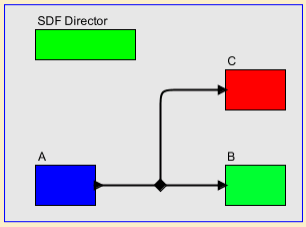
- A preferable schedule would space invocations of A and B uniformly in time, as in:



U.PORTO

[dcc]

Non-Concurrent Uniformly Timed Schedule

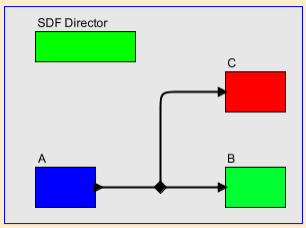


- Notice that in this schedule, the rate at which A and B can be invoked is limited by the execution time of C.

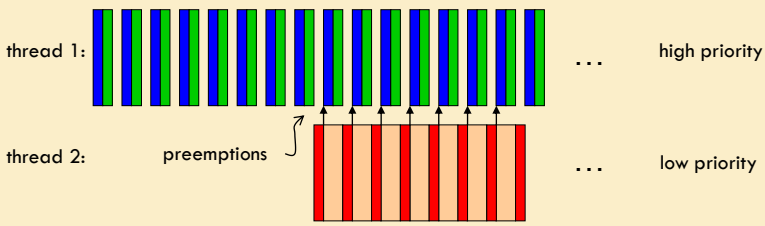
No jitter, but utilization is poor.

[dcc]

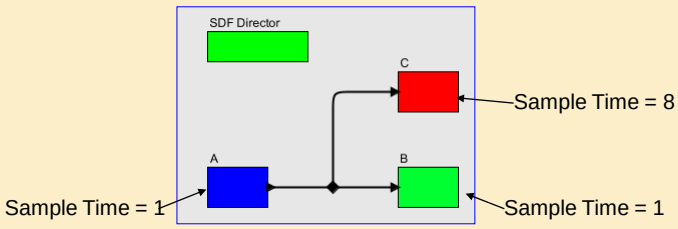
Concurrent Uniformly Timed Schedule: Pre-emptive schedule



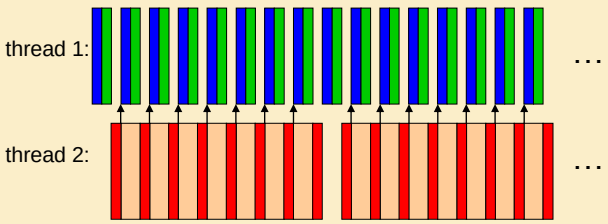
- With pre-emption, the rate at which A and B can be invoked is limited only by total computation:



Ignoring Initial Transients, Abstract to Periodic Tasks

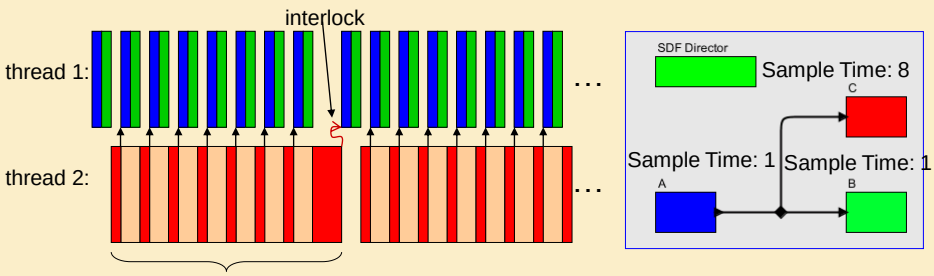


- In steady-state, the execution follows a simple periodic pattern:



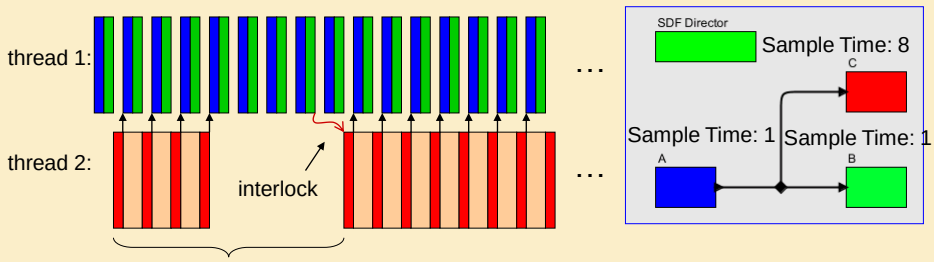
This follows the principles of rate-monotonic scheduling (RMS).

Requirement 1 for Determinacy: Periodicity



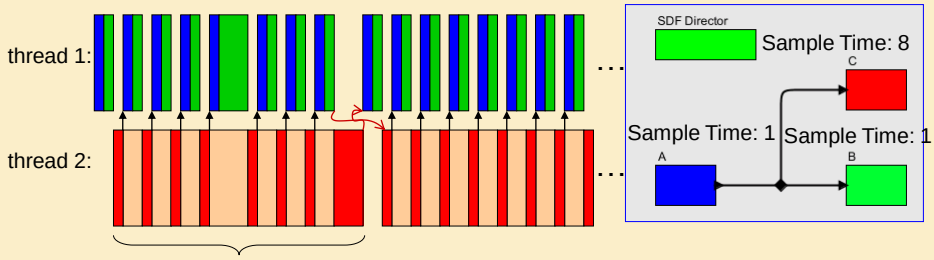
- If the execution of C runs longer than expected, data determinacy requires that thread 1 be delayed accordingly.
- This can be accomplished with semaphore synchronization. But there are alternatives:
  - *Throw an exception to indicate timing failure.*
  - *"Anytime" computation: use incomplete results 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

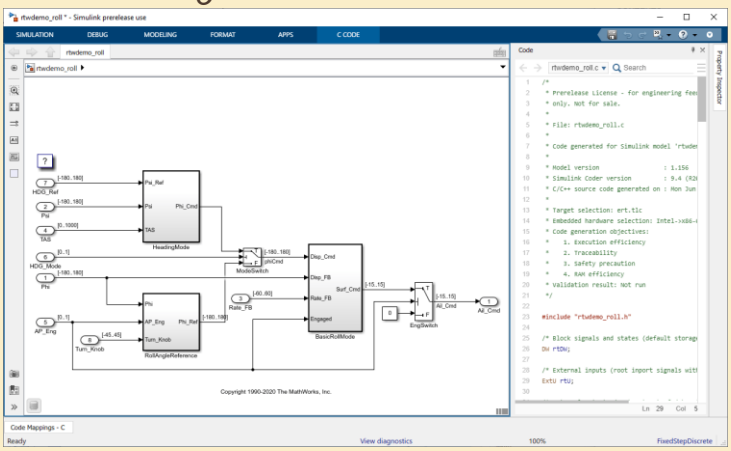


- Note that semaphore synchronization is not required if actor 1 runs long because its thread has higher priority.
  - *Everything else is automatically delayed.*

[dcc]

Simulink and Coder (The MathWorks)

- Typical usage pattern:
  - *model the continuous dynamics of the physical plant*
  - *model the discrete-time controller*
  - *code generated for the discrete-time controller*



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

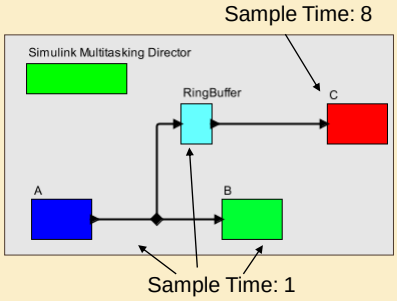
U.PORTO

19

19

[dcc]

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 must explicitly include a buffering actor that outputs an array.

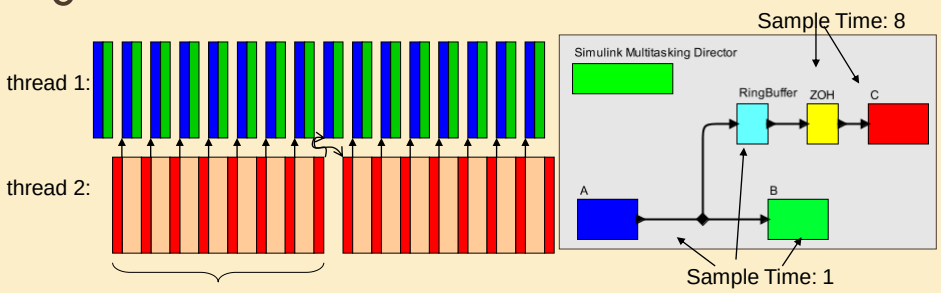
U.PORTO

20

20

[dcc]

Requirement 2 for Determinacy: Data Integrity During Execution

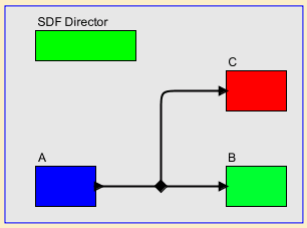


- It is essential that input data remains stable during one complete execution of C, something achieved in Simulink with a zero-order hold (ZOH) block.

U.PORTO

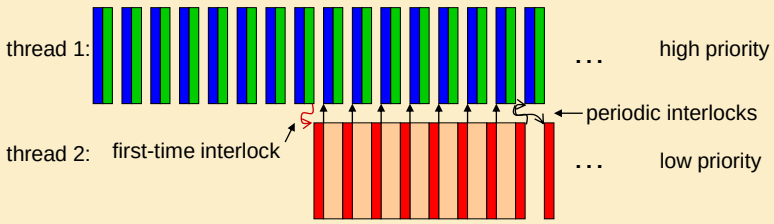
[dcc]

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



No ZOH block is required!

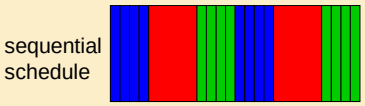
- For dataflow, a one-time interlock ensures sufficient data at the input of C:

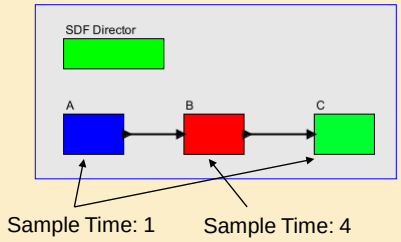


U.PORTO

[dcc]

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



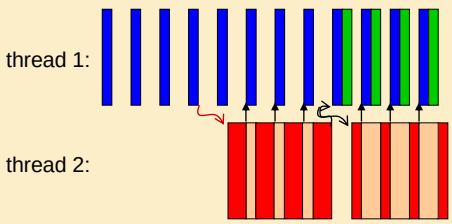


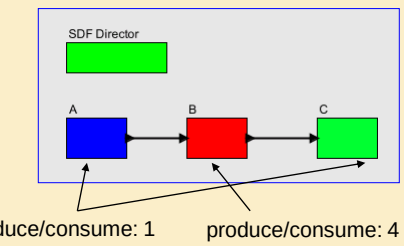
- Note that data precedencies make it impossible to achieve uniform timing for A and C with the periodic non-concurrent schedule indicated above.

U.PORTO

[dcc]

Overlapped Iterations Can Solve This Problem



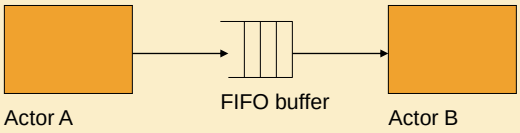


- This solution takes advantage of the intrinsic buffering provided by dataflow models.
- For dataflow, this requires the initial interlock as before, and the same periodic interlocks.

U.PORTO

[dcc]

Dataflow Models



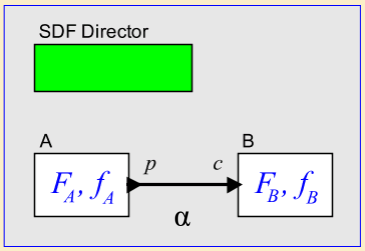
- Buffered communication between concurrent components (actors).
- An actor can fire whenever it has enough data (tokens) in its input buffers. It then produces some data on its output buffers.
- In principle, buffers are unbounded. But for implementation on a computer, we want them bounded (and as small as possible).

U.PORTO

[dcc]

Synchronous Dataflow (SDF)

- If the number of tokens consumed and produced by the firing of an actor is constant, then static analysis can tell us whether we can schedule the firings to get a useful execution, and if so, then a finite representation of a schedule for such an execution can be created.



U.PORTO

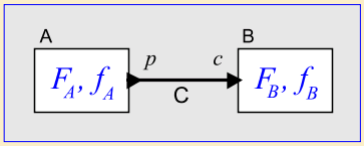
[dcc]

Balance Equations

- Let q_A, q_B be the number of firings of actors A and B.
- Let p_α, c_α be the number of tokens produced and consumed on a connection α.
- Then the system is in balance if for all connections C

$$q_A \cdot p_\alpha = q_B \cdot c_\alpha$$

- where A produces tokens on α and B consumes them.

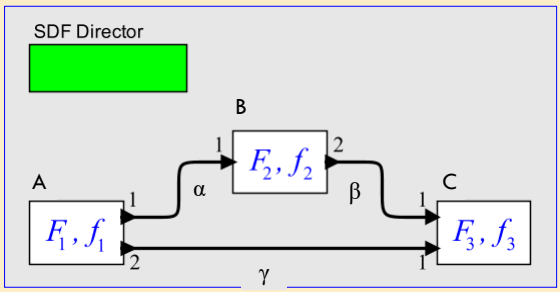


U.PORTO

[dcc]

Example

- Consider this example, where actors and arcs are numbered:

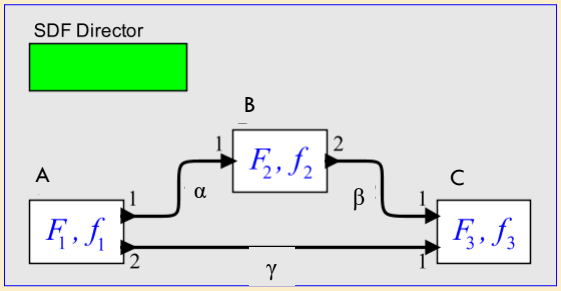


- The balance equations imply that actor C must fire twice as often as the other two actors.

U.PORTO

[dcc]

Compactly Representing the Balance Equations



production/consumption matrix

$$\Gamma = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & -1 \\ 2 & 0 & -1 \end{bmatrix}$$

Connector α

Actor A

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

firing vector

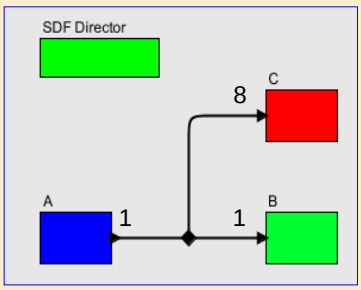
balance equations

$$\Gamma q = \vec{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

[dcc]

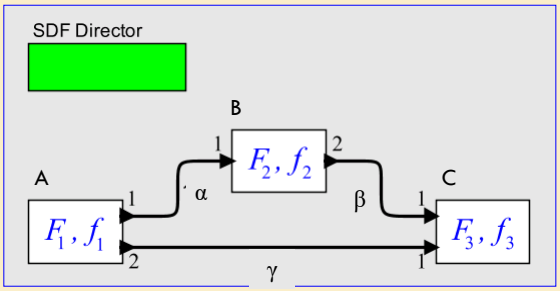
Recalling…

- What is the production/consumption matrix in this case?



$$\Gamma = \begin{bmatrix} 1 & 0 & -1 \\ 1 & -8 & 0 \end{bmatrix}$$

Example

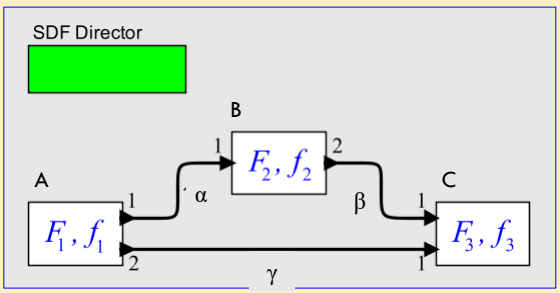


- A solution to the balance equations:

$$\Gamma = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & -1 \\ 2 & 0 & -1 \end{bmatrix} \qquad \Gamma q = \vec{0} \qquad q = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

This tells us that actor C must fire twice as often as actors 1 and 2.

Example



- But there are many solutions to the balance equations:

$$q = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \quad q = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad q = \begin{bmatrix} 2 \\ 2 \\ 4 \end{bmatrix} \quad q = \begin{bmatrix} -1 \\ -1 \\ -2 \end{bmatrix} \quad q = \begin{bmatrix} \pi \\ \pi \\ 2\pi \end{bmatrix}$$

- For "well-behaved" models, there is a unique least positive integer solution.

[dcc]

Least Positive Integer Solution to the Balance Equations

- If p_α, c_α , nr of tokens produced and consumed on connection α, are non-negative integers, then the balance equation,

$$q_A \cdot p_\alpha = q_B \cdot c_\alpha$$

- implies:
  - *q_A is rational if and only if q_B is rational.*
  - *q_A is positive if and only if q_B is positive.*

- Consequence: Within any connected component, if there is any non-zero solution to the balance equations, then there is a unique least positive integer solution.

U.PORTO

[dcc]

Rank of a Matrix

- The rank of a matrix Γ is the number of linearly independent rows or columns. The equation

$$\Gamma q = \vec{0}$$

- is forming a linear combination of the columns of Γ.
  - *Such a linear combination for $q \neq \vec{0}$ can only yield the zero vector if the columns are linearly dependent.*
    - If all columns/rows are independent only $q = \vec{0}$ is a solution

- If Γ has a columns and b rows, the rank cannot exceed $\min(a, b)$.
- If the columns or rows of Γ are re-ordered, the resulting matrix has the same rank as Γ.

U.PORTO

[dcc]

Rank of the Production/Consumption Matrix

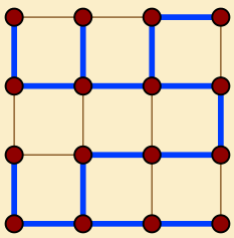
1. Let a be the number of actors in a connected graph. Then the rank of the production/consumption matrix $\Gamma \leq a$.
   - *Why?*
2. If the model is a spanning tree (meaning that there are barely enough connections to make it connected) then Γ has a columns (actors) and $a - 1$ rows (connections).
3. Theorem [Lee-Messerschmitt'87]: Its rank is $a - 1$.
4. Corollary: the rank of any production/consumption matrix of a connected graph is either a or $a - 1$.
   - *Why?*

U.PORTO

[dcc]

Max rank is a or $a - 1$

- Grid graph (prod/consumption graph)
- In blue: sub graph as a spanning tree



- Rank of spanning tree sub graph is rank $a - 1$
- Rank of full graph is at most a from 1st bullet

Image: "4x4 grid spanning tree" by David Eppstein - Own work.
Licensed under Public domain via Wikimedia

U.PORTO

[dcc] U.PORTO

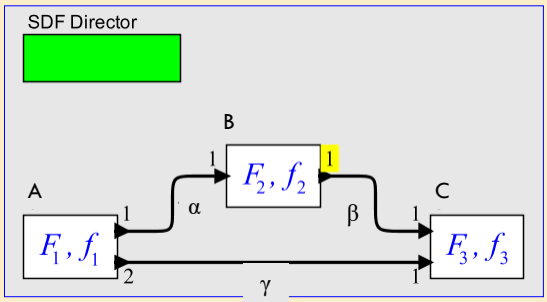
Consistent Models

- Let a be the number of actors in a connected model. The model is consistent if Γ has rank $a - 1$.
- If the rank is a, then the balance equations have only a trivial solution (zero firings).
- When Γ has rank $a - 1$, then the balance equations always have a non-trivial solution.

[dcc] U.PORTO

Example of an Inconsistent Model: No Non-Trivial Solution to the Balance Equations

$$\Gamma = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 2 & 0 & -1 \end{bmatrix}$$



- This production/consumption matrix has rank 3, so there are no nontrivial solutions to the balance equations.
- Note that this model can execute forever, but it requires unbounded memory.

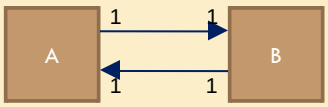
Necessary and sufficient conditions

- Consistency is a necessary condition to have a (bounded-memory) infinite execution.

- Is it sufficient?
  - *No. Could get deadlock*

Deadlock 1

- Is this diagram consistent?

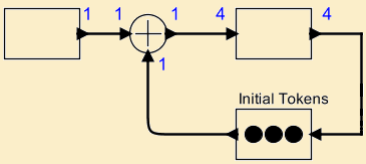


$$\Gamma = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

[dcc]

Deadlock 2

- Some dataflow models cannot execute forever. In the model below, the feedback loop injects initial tokens, but not enough for the model to execute.



U.PORTO

[dcc]

SDF: from static analysis to scheduling

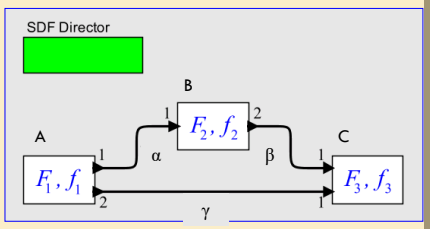
- Given: SDF diagram
- Find: a bounded-buffer schedule, if it exists
- Step 0: check whether diagram is consistent. If not, then no bounded-buffer schedule exists.
- Step 1: find an integer solution to $\Gamma q = \vec{0}$.
- Step 2: "decompose" the solution q into a schedule, making sure buffers never become negative.

U.PORTO

Step 2: "decomposing" the firing vector

- Example 1:

Schedule = (A;B;C;C)



$$b = \begin{bmatrix}0\\0\\0\end{bmatrix} \quad q = \begin{bmatrix}1\\1\\2\end{bmatrix}$$

Fire A

$$b = \begin{bmatrix}1\\0\\2\end{bmatrix} \quad q = \begin{bmatrix}0\\1\\2\end{bmatrix}$$

Fire B

$$b = \begin{bmatrix}0\\2\\2\end{bmatrix} \quad q = \begin{bmatrix}0\\0\\2\end{bmatrix}$$

Fire C

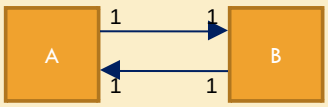
$$b = \begin{bmatrix}0\\1\\1\end{bmatrix} \quad q = \begin{bmatrix}0\\0\\1\end{bmatrix}$$

Fire C

$$b = \begin{bmatrix}0\\0\\0\end{bmatrix} \quad q = \begin{bmatrix}0\\0\\0\end{bmatrix}$$

Step 2: "decomposing" the firing vector

- Example 2:



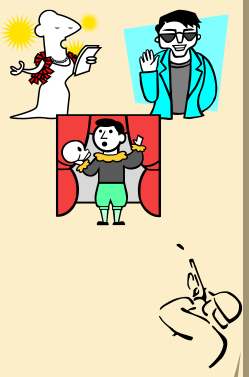
$$\Gamma = \begin{bmatrix}1 & -1\\-1 & 1\end{bmatrix} \qquad b = \begin{bmatrix}1\\1\end{bmatrix}$$

What happens if we try to run the previous procedure on this example?

So, we have both necessary and sufficient conditions for scheduling SDF graphs.

If More Than One Actor is Fireable

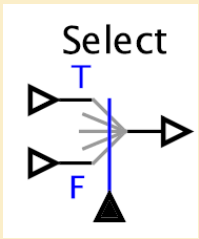
- A Key Question: If More Than One Actor is Fireable in Step 2, How do I Select One?



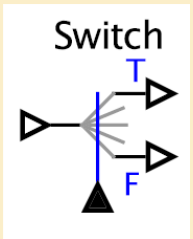
- Optimization criteria that might be applied:
  - *Minimize buffer sizes.*
  - *Minimize the number of actor activations.*
  - *Minimize the size of the representation of the schedule (code size).*

Dynamic Dataflow components

- **Select:** Based on the input on ▲ (True or False) the input from △ (T or F respectively) will be selected to the output △



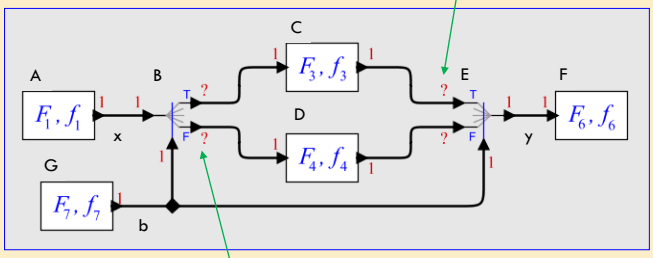
- **Switch:** Based on the input on ▲ (True or False) the input from △ will be switched to the output △ (T or F respectively).



Dynamic Dataflow

- Imperative equivalent:

```
while (true) {
    x = f1();
    b = f7();
    if (b) {
      y = f3(x);
    } else {
      y = f4(x);
    }
    f6(y);
}
```



The if-then-else model is not SDF. But we can clearly give a bounded *quasi-static* schedule for it:

(A, G, B, b?C, !b?D, E, F)

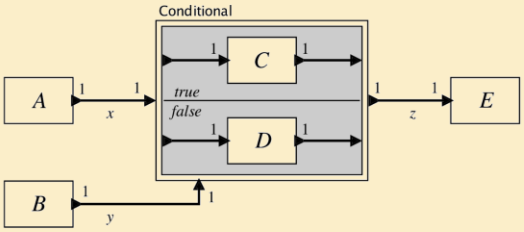
guard

Facts about (general) dynamic dataflow

- Whether there exists a schedule that does not deadlock is undecidable.
- Whether there exists a schedule that executes forever with bounded memory is undecidable.
- Undecidable means that there is no algorithm that can answer the question in finite time for all finite models.
- Note that for Synchronous Data Flow deadlock and bounded mem are decidable

Structured Data Flows

- Higher-order actor called Conditional.
  - *A higher-order actor is an actor that has one or more models as parameters.*
- When Conditional fires, it consumes one token from each input port and produces one token on its output port ➔ SDF actor
- Action performed is dependent on token at the lower input port.
  - *True* ➔ *actor C fires*
  - *False* ➔ *actor D fires.*



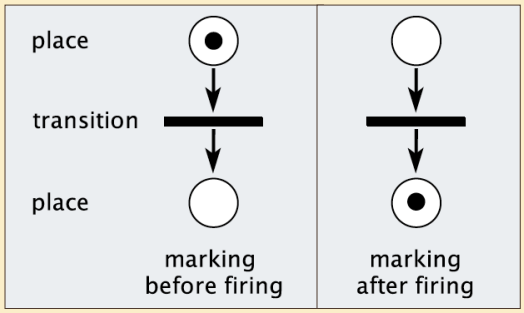
Structured Data Flows

- Control constructs are nested hierarchically
- Overall model is still an SDF model.
- Analysable for deadlock and bounded buffers.

[dcc]

Petri Nets

- A variant of dataflow
- Places can contain arbitrary number of tokens
- Transitions are enabled if all places connected to it contain at least one token
- Enabled transitions can fire, consuming one token from each input and putting one token on each output
- State of the network, the marking, is the number of tokens on each place



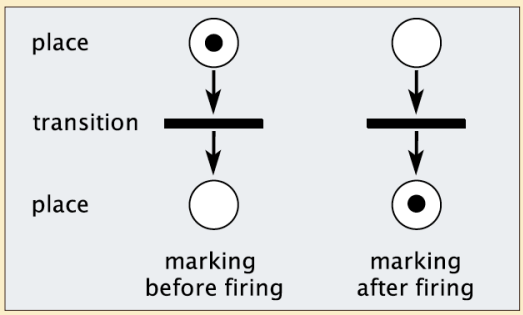
U.PORTO

51

[dcc]

Petri Nets

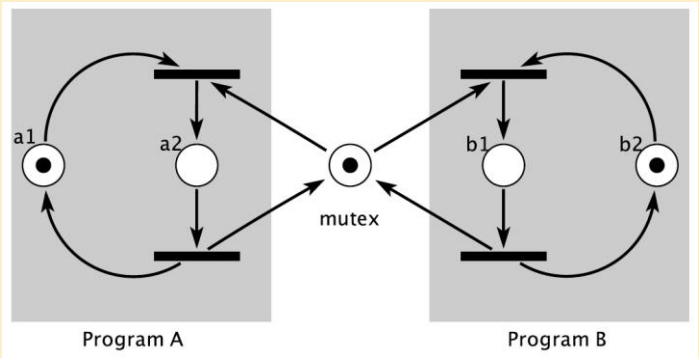
- If no place provides input to more than one transition, then the network is deterministic
- Unlike dataflow buffers, places do not preserve an ordering of tokens
- Petri nets with a finite number of markings are equivalent to FSMs



U.PORTO

52

Petri Net Example – Mutual Exclusion



- Programs A and B both have a critical section (a2 and b1)
- Only one of the programs may be in its critical section at any time

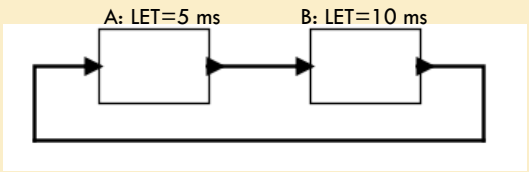
TIMED MODELS OF COMPUTATION

Time-Triggered Models of Computation

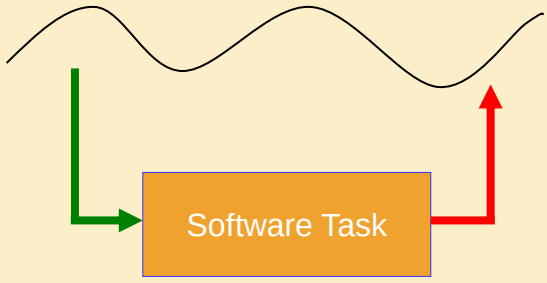
- Periodically triggering distributed computations according to a distributed clock
- Time-triggered MoC is similar to Sync-Reactive ➔ there is a global clock that coordinates the computation
- But computations take time instead of being simultaneous and instantaneous.
  - *A logical execution time (LET)*
  - *Outputs are not visible to other computations until the next tick of the global clock*

Time-Triggered Models

- Between ticks, there is no interaction between the computations ➔concurrency difficulties such as race conditions do not exist
- Computations are not (logically) instantaneous ➔ there are no difficulties with feedback, and all models are constructive



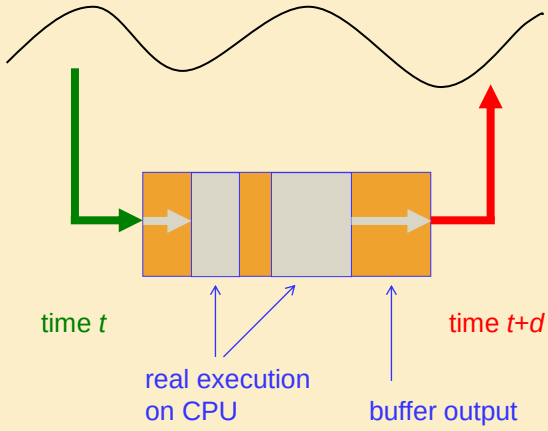
The LET (Logical Execution Time) Programming Model



read sensor input at time *t*

write actuator output at time *t+d*, for specified *d*

The LET (Logical Execution Time) Programming Model



[dcc]

Discrete Event Systems

- Been used for decades to build simulations (e.g.: ns3)
- Events are endowed with a time stamp
  - *a value in some model of time*
- 2 distinct time stamps must be comparable
  - *Either equal or one is earlier than the other*
- DE model: a network of actors where each actor:
  - *reacts to input events in time stamp order*
  - *produces output events in time stamp order.*

U.PORTO

[dcc]

Discrete Event Systems

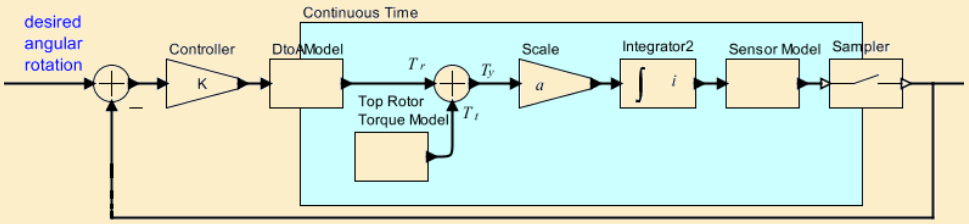
- Have an event queue
  - *Events sorted by time stamp*
- Each actor in the network is interrogated for any initial events it wishes to place on the queue
- Events have destinations and (of course) a time stamp
- Execution continues selecting the earliest event in queue and determining which actor should receive it
- Actor fires/reacts
- Reaction can:
  - *produce output events*
  - *events that simply request a later firing of the same actor*

U.PORTO

[dcc]

Continuous Time

- Recall: helicopter feedback control system



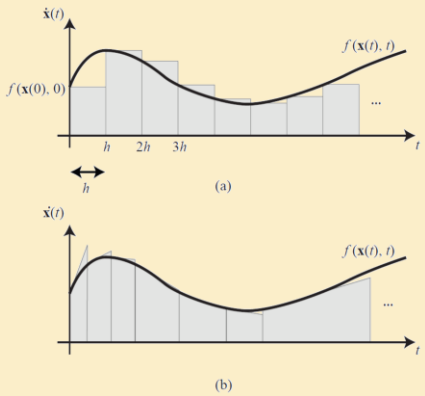
- Actor compositions under a continuous-time model of computation.
- However: cannot strictly be executed on a digital computer.
  - *A digital computer cannot directly deal with the time continuum*

U.PORTO

[dcc]

Continuous time - Approximation

- Approximate execution of a continuous-time model is accomplished by a solver
  - *constructs a numerical approximation to the solution of an Ordinary Differential Equation*



U.PORTO

[dcc] U.PORTO

Continuous time model

- Network of actors, each of which is a cascade composition of a simple memory less computation actor and a state machine
- Actor reactions are simultaneous and instantaneous.
- The times of the reactions are determined by a solver.

- Mechanisms required to achieve a continuous-time model of computation are not much different from SR and DE.

[dcc] U.PORTO

Summary

- Data flow models of computation
  - *Synchronous dataflow*
  - *Dynamic dataflow*
  - *Structured Dataflow*
  - *Petri Nets*
- Timed Models of Computation
  - *Time Triggered*
  - *Discrete Event*
  - *Continuous Time*