2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION TAIWAN

Model-Based Design
The Top-Level System Design Method

Alan P. Su, Ph.D.

EE, NCKU and eNeural Technologies, Inc.

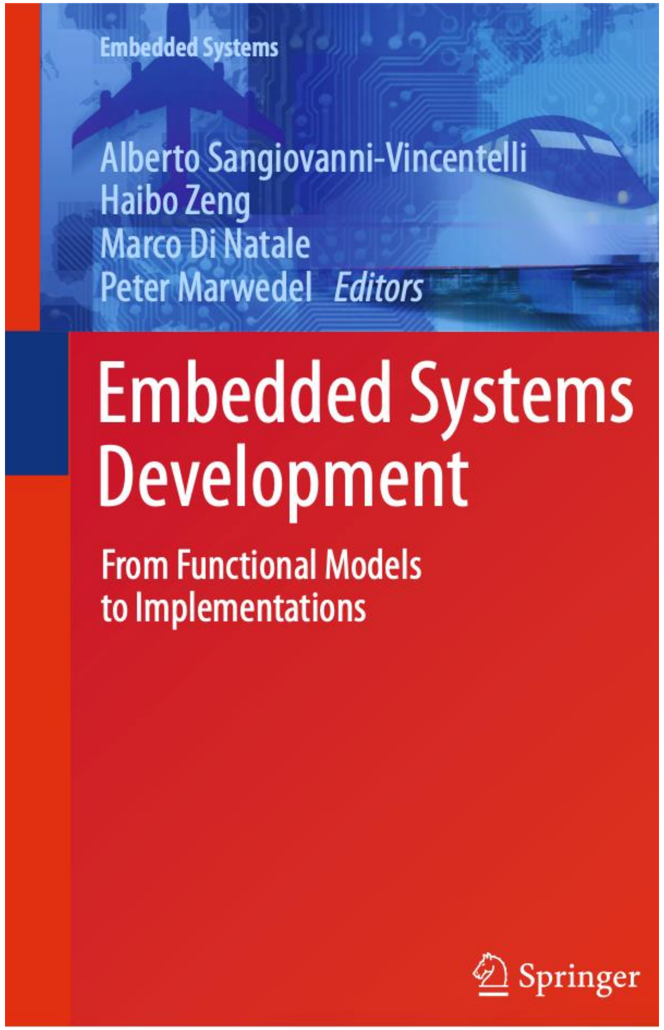
1931

eNeural Technologies, Inc.

accellera SYSTEMS INITIATIVE

Embedded Systems Development

From functional Models to Implementations



Editors

Alberto Sangiovanni-Vincentelli
Department of Electrical Engineering and Computer Science
University of California
Berkeley, CA
USA

Haibo Zeng
Department of ECE
McGill University
Montreal
Canada

Marco Di Natale
TeCIP Institute
Scuola Superiore Sant'Anna
Pisa
Italy

Peter Marwedel
Embedded Systems Group
TU Dortmund University
Dortmund
Germany

ISSN 2193-0155 ISSN 2193-0163 (electronic)
ISBN 978-1-4614-3878-6 ISBN 978-1-4614-3879-3 (eBook)
DOI 10.1007/978-1-4614-3879-3
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2013939569

This lecture is inspired and derived from this book

What is a Model?

- How to translate ‘model’ into Chinese?
  - 模型 – very good! And, what does a model do?
  - Modeling – great! Right again!
    - Then, how to describe the behavior of ‘modeling’ in a simple sentence?
- 模特兒(model) 走秀(modeling)
  - Simulate how you look in these dresses

We use models to
simulate
target objects

Mathematical modeling, RTL modeling/simulation, physical simulation, p-spice, TCAD, behavior simulation, etc. etc. etc.

Why Model Based Design?

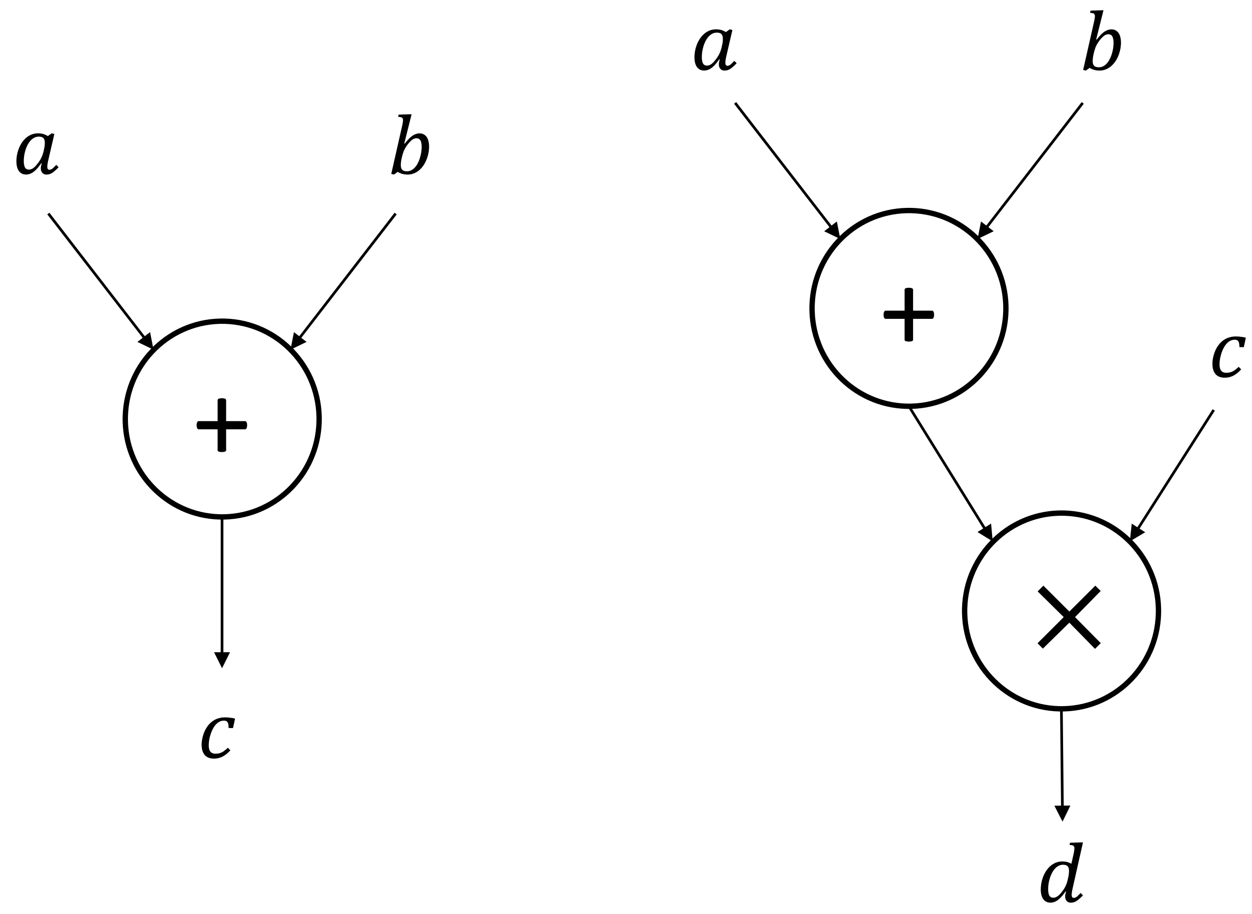
- Every engineering system starts from math.
- We use math to model a system, and the process is mathmatical modeling, a.k.a. designing. Then math models thus designed are used to simulate the system for verification.
- The design process is a cycle of mathmatical modeling, simulation, verification, and modification till the system is fully verified to meet the target functionality.
- And this design process, is the so-called Model Based Design. And engineers like us do MBD everyday.

Model-Based Design (MBD)

- The Model-Based Design approach (MBD) prescribes the use of models based on a mathematical formalism and executable semantics to represent the controller system (to be realized in SW and/or HW) and the controlled device or environment (often referred to as *Plant*).
- Examples of available commercial tools for model-based development are Simulink, SCADE, NI LabVIEW[1], and Modelica. Academic projects that fit this definition are Ptolemy [2] and Metro II [3]. These tools are feature-rich and allow the modeling of continuous or discrete time, or hybrid systems in which functionality is typically represented using a dataflow or an extended finite-state machine formalism (or a combination of them).

1. Andrade, H. A., Kovner, S.: Software synthesis from data flow models for G and LabVIEW. In: Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers, 1705– 1709 (1998)
2. Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity-the Ptolemy approach. Proc. IEEE **91**(1), 127–144 (2003)
3. Davare, A., Densmore, D., Meyerowitz, T., Pinto, A., Sangiovanni-Vincentelli, A., Yang, G., Zeng, H., Zhu, Q.: A next-generation design framework for platform-based design. DVCon, In (2007)

Math Is A Model-Based Language



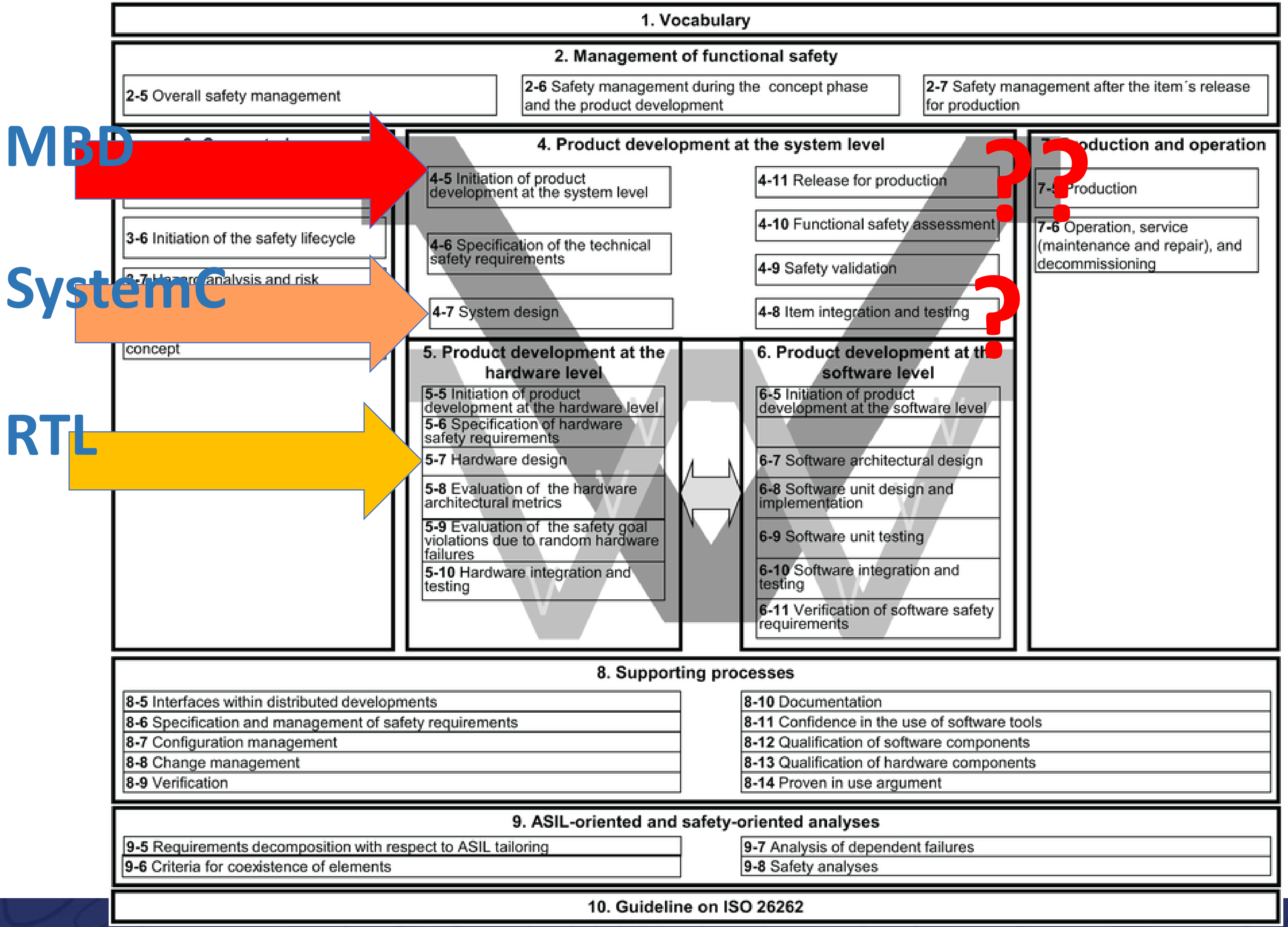
Why MBD from the book

- Embedded systems are increasingly complex, function-rich and required to perform tasks that are mission- or safety-critical.
- The use of models to specify the functional contents of the system and its execution platform is today the most promising solution to reduce the productivity gap and improve the quality, correctness and modularity of software subsystems and systems.
- Models allow to advance the analysis, validation, and verification of properties in the design flow, and enable the exploration and synthesis of cost-effective and provably correct solutions.

Using MBD

- Traditional programming techniques, including object-oriented languages, are not able to reduce the productivity gap, and embedded system development processes demand new methods and techniques that can improve the quality, correctness, and modularity of systems and subsystems by advancing the analysis and verification of properties as early as possible in the design flow.
- The use of models can help the analysis of the system properties and verification by simulation, the documentation of the design decisions, and possibly the automatic generation of the software implementation. Each of the previous topics is the subject of a number of relevant research domains, but all of them are also part of the industrial practice, at least to some degree, backed by several commercial products and standards.

ISO 26262 – The V Process

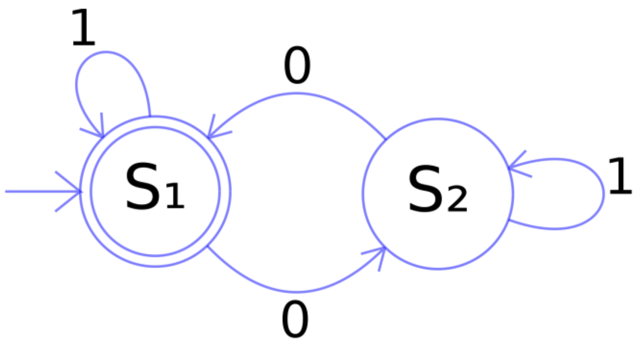


Dataflow Models of Computation

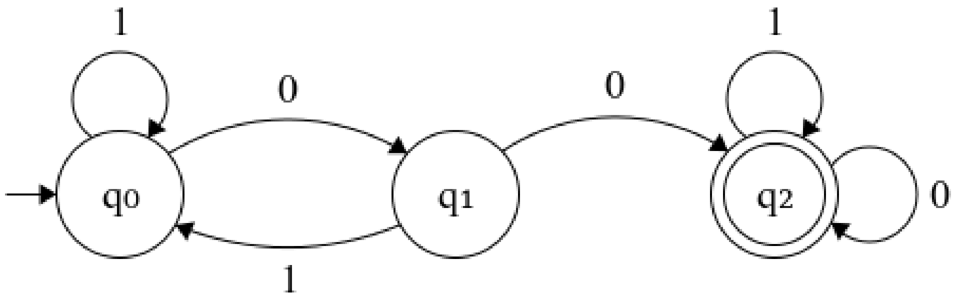
- Dataflow models are characterized by a data-driven style of control; data are processed while flowing through a network of computation nodes. There are three major variants of dataflow models in the literature, namely, dataflow process networks, Kahn Process Networks, and dataflow synchronous languages. Examples:
  - Finite State Machine
  - Petri net (PN) – by Carl Adam Petri in 1962
  - Kahn Process Networks (KPN) – by Gilles Kahn in 1974
  - Communicating Sequential Process (CSP) – by C. A. R Hoare in 1978
  - Synchronous Data Flow (SDF) – by Edward A. Lee in 1987

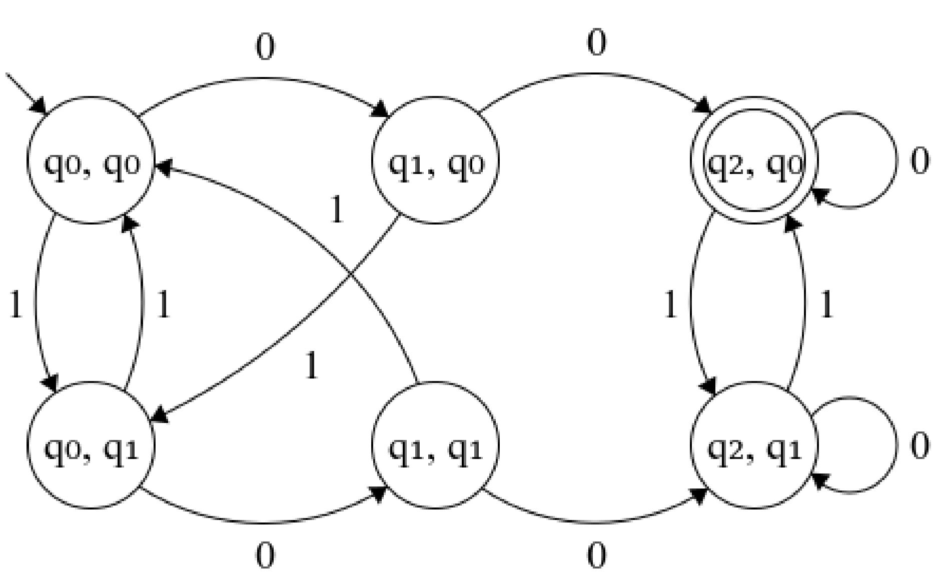
Deterministic Finite Automata

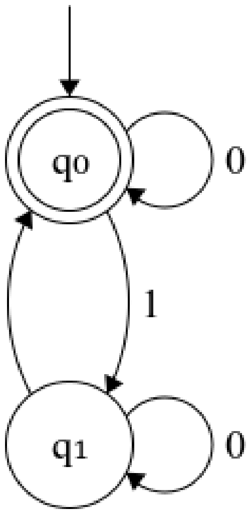
- A Deterministic Finite Automata (DFA) is described by a five-element tuple: $(Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of states
  - Σ is a finite, nonempty input alphabet
  - $\delta: Q \times \Sigma \rightarrow Q$ is a series of transition functions
  - $q_0 \in Q$ is the initial state
  - $F \subseteq Q$ is the set of accepting states



DFA Examples





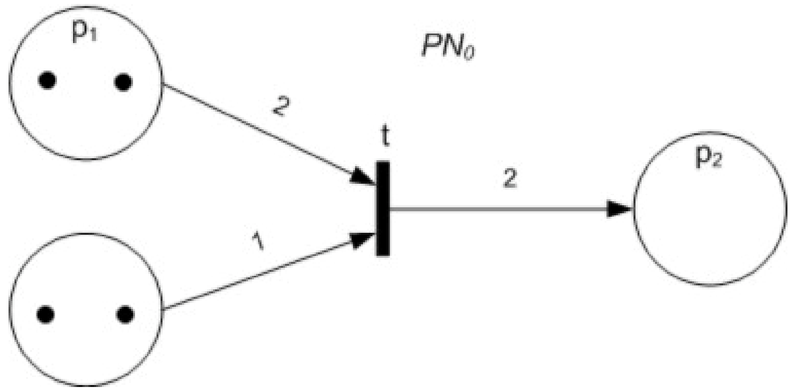


Petri net

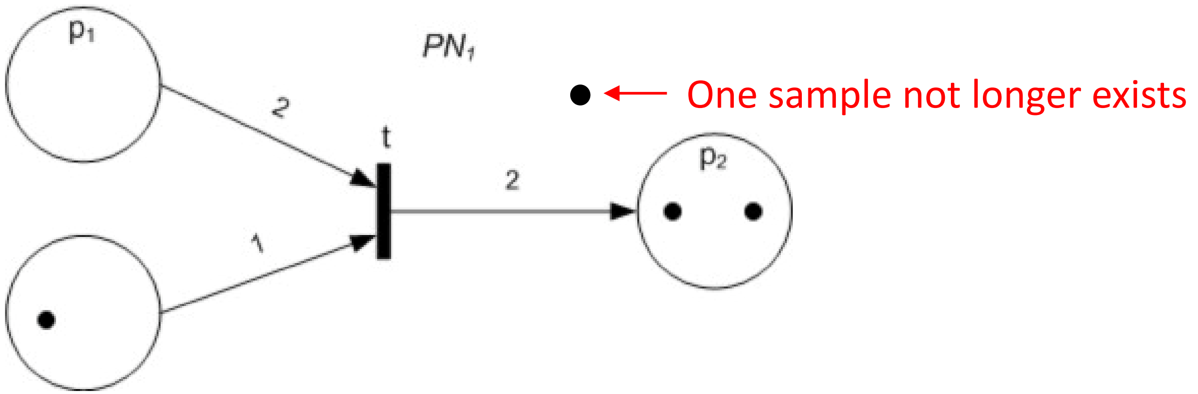
A.K.A. Place/Transition net, is a mathematical modeling language of distributed systems

- Definition 1. A net is a 3-tuple $N = (P, T, F)$ where
  - P and T are disjoint finite sets of places and transitions, respectively
  - $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (or flow relations)
- Definition 2. Given a net $N = (P, T, F)$, a configuration is a set C so that $C \subseteq P$
- Definition 3. An elementary net is a net of the form $EN = (N, C)$ where
  - $N = (P, T, F)$ is a net
  - C is such that $C \subseteq P$ is a configuration
- Definition 4. A Petri Net is a net of the form $PN = (N, M, W)$, which extends the elementary net so that
  - $N = (P, T, F)$ is a net
  - $M: P \rightarrow Z$ is a place multiset, where Z is a countable set. M extends the concept of configuration and is commonly described with reference to Petri Net diagrams as marking
  - $W: F \rightarrow Z$ is an arc multiset, so that the count (or weight) for each arc is a measure of the arc multiplicity

A Petri net Example



A Petri net with an enabled transition



The Petri net after the transition

Kahn Process Network

A common model for describing signal processing systems where infinite streams of data are incrementally transformed by processes executing in sequence or parallel

Execution model

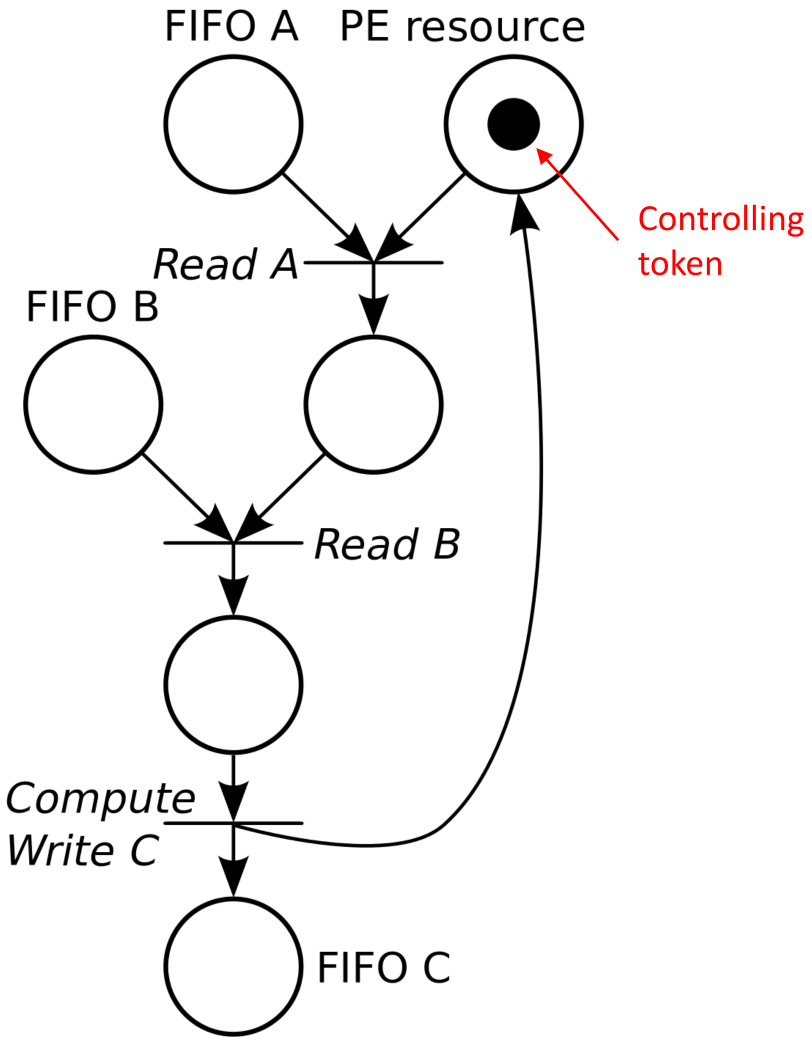
- In a KPN, processes communicate via unbounded FIFO channels. Processes read and write atomic data elements, or alternatively called tokens, from and to channels.
- Writing to a channel is non-blocking, i.e. it always succeeds and does not stall the process,
- while reading from a channel is *blocking*, i.e. a process that reads from an empty channel will stall and can only continue when the channel contains sufficient data items (*tokens*).
- Processes are not allowed to test an input channel for existence of tokens without consuming them.
- A FIFO cannot be consumed by multiple processes, nor can multiple processes produce to a single FIFO.
- Given a specific input (token) history for a process, the process must be deterministic so that it always produces the same outputs (tokens).
- Timing or execution order of processes must not affect the result and therefore testing input channels for tokens is forbidden.

KPN Processes

- A process need not read any input or have any input channels as it may act as a pure data source
- A process need not write any output or have any output channels
- Testing input channels for emptiness (or *non-blocking reads*) could be allowed for optimization purposes, but it should not affect outputs. It can be beneficial and/or possible to do something in advance rather than wait for a channel. For example, assume there were two reads from different channels. If the first read would stall (wait for a token) but the second read could be read a token directly, it could be beneficial to read the second one first to save time, because the reading itself often consumes some time (e.g. time for memory allocation or copying).

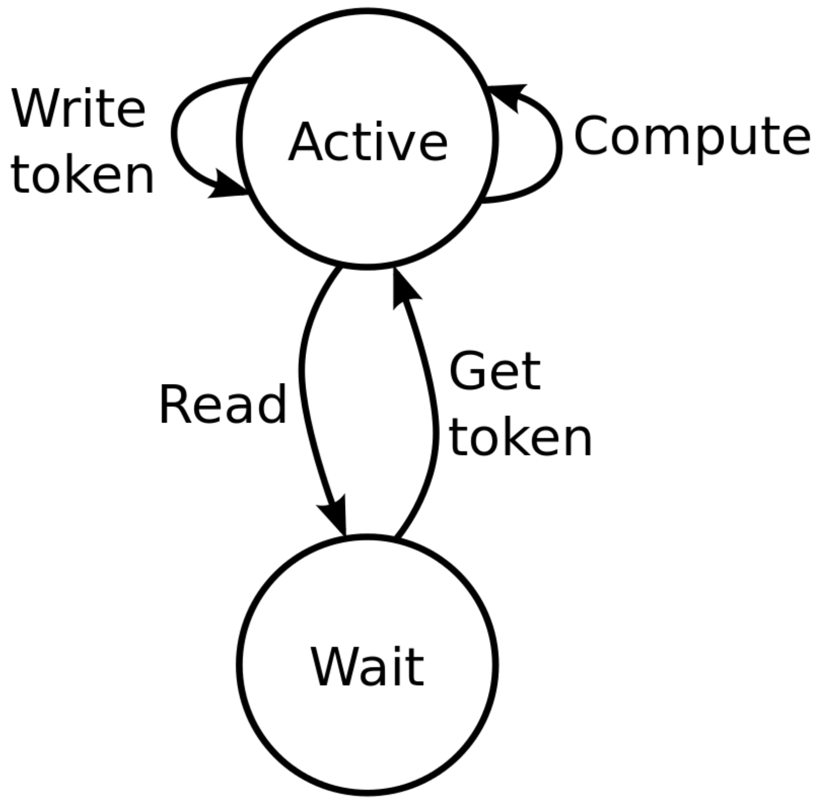
Process Firing Semantics

- Assuming process *P* in the KPN above is constructed so that it first reads data from channel *A*, then channel *B*, computes something and then writes data to channel *C*, the execution model of the process can be modeled with the Petri net shown on the right. The single token in the *PE resource* place forbids that the process is executed simultaneously for different input data. When data arrives at channel *A* or *B*, tokens are placed into places *FIFO A* and *FIFO B* respectively. The transitions of the Petri net are associated with the respective I/O operations and computation. When the data has been written to channel *C*, *PE resource* is filled with its initial marking again allowing new data to be read.



The FSM of KPN

- A process can be modeled as a finite state machine that is in one of two states:
  - Active; the process computes or writes data
  - Wait; the process is blocked (waiting) for data
- Assuming the finite state machine reads program elements associated with the process, it may read three kinds of tokens, which are "Compute", "Read" and "Write token". Additionally, in the *Wait* state it can only come back to *Active* state by reading a special "Get token" which means the communication channel associated with the wait contains readable data.



Boundedness of Channels

- A channel is strictly bounded by b if it has at most b unconsumed tokens for any possible execution. A KPN is strictly bounded by b if all channels are strictly bounded by b.
- The number of unconsumed tokens depends on the execution order (scheduling) of processes. A spontaneous data source could produce arbitrarily many tokens into a channel if the scheduler would not execute processes consuming those tokens.
- A real application can not have unbounded FIFOs and therefore scheduling and maximum capacity of FIFOs must be designed into a practical implementation. The maximum capacity of FIFOs can be handled in several ways:
  - FIFO bounds can be mathematically derived in design to avoid FIFO overflows. This is however not possible for all KPNs. It is an undecidable problem to test whether a KPN is strictly bounded by b. Moreover, in practical situations, the bound may be data dependent.
  - FIFO bounds can be grown on demand.
  - Blocking writes can be used so that a process blocks if a FIFO is full. This approach may unfortunately lead to an artificial deadlock unless the designer properly derives safe bounds for FOFIs. Local artificial detection at run-time may be necessary to guarantee the production of the correct output.

Communicating Sequential Process

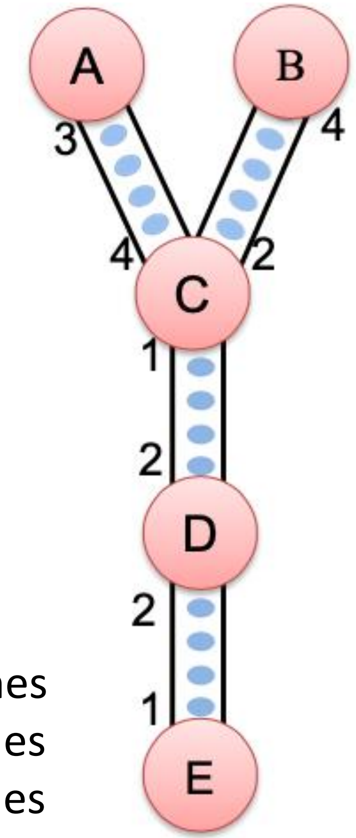
Initially a concurrent programming language and later developed into a process algebra. Its industrial use to system design is in safety-critical systems.

- CSP allows the description of systems in terms of component processes that operate independently and interact with each other solely through message-passing communication. However, the *"Sequential"* part of the CSP name needs to be carefully considered, since modern CSP allows component processes to be defined both as sequential processes, and as the parallel composition of more primitive processes. The relationships between different processes, and the way each process communicates with its environment, are described using various process algebraic operators. Using this algebraic approach, quite complex process descriptions can be easily constructed from a few primitive elements.
- Informal descriptions to CSP please refer to https://en.wikipedia.org/wiki/Communicating_sequential_processes

Synchronous Data Flow

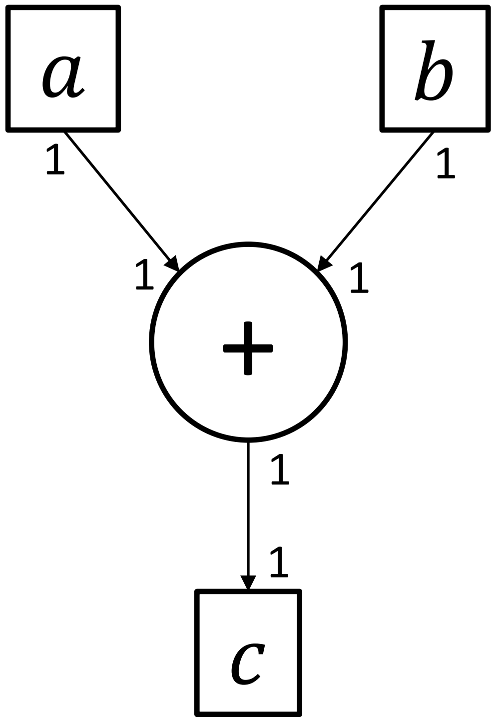
is a restriction of KPN where nodes produce and consume a fixed number of data items per firing. This allows static scheduling

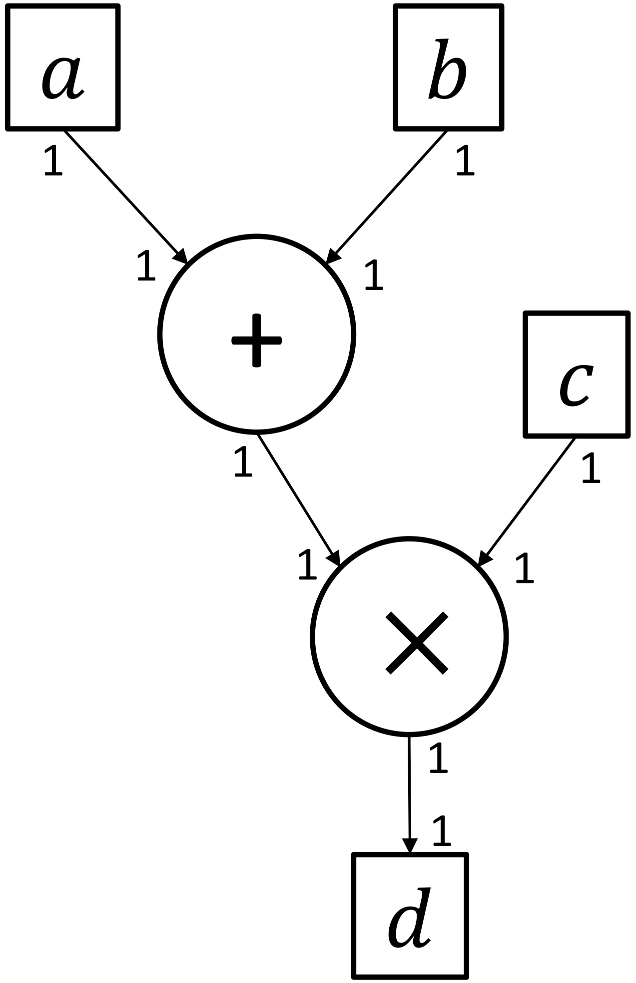
- Synchronous Data Flow (SDF) is represented as a graph
  - – Node (actor): Computation
  - – Edge: First In First Out (FIFO) Queue
- Each edge has two weights: produce rate and consume rate
- Each edge can also have initial data
- Formally, SDF is a 3-tuple $(N, E, E_{p,c,i})$
  - N is a set of nodes
  - E is a set of edges
  - $E_{p,c,i}$ where
    - p is the produce rate
    - c is the consume rate
    - i is the initial data



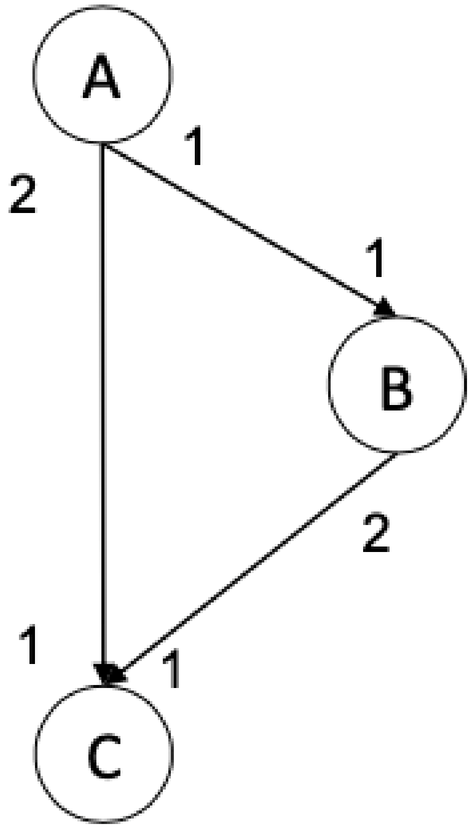
A: fires 8 times
B: fires 3 times
C: fires 6 times
D: fires 3 times
E: fires 6 times

Example: Adder, Adder-Multiplier

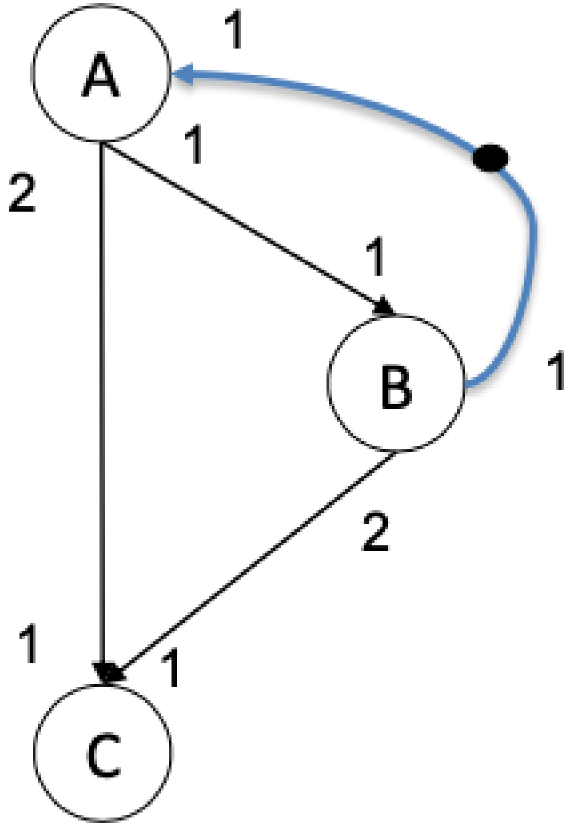




SDF Examples

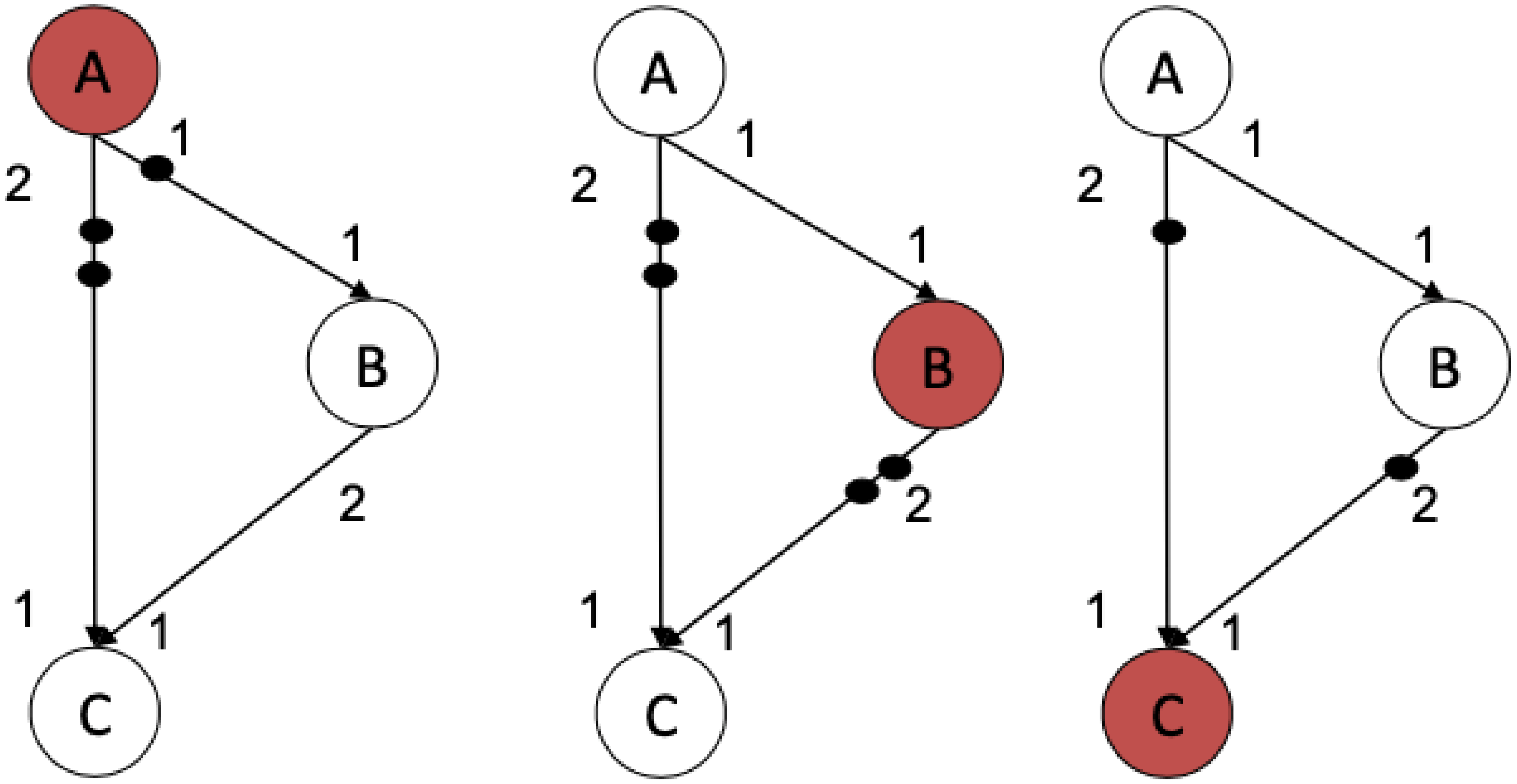


SDF without initial tokens



SDF with initial tokens and loop

Consistent SDF Simulation I



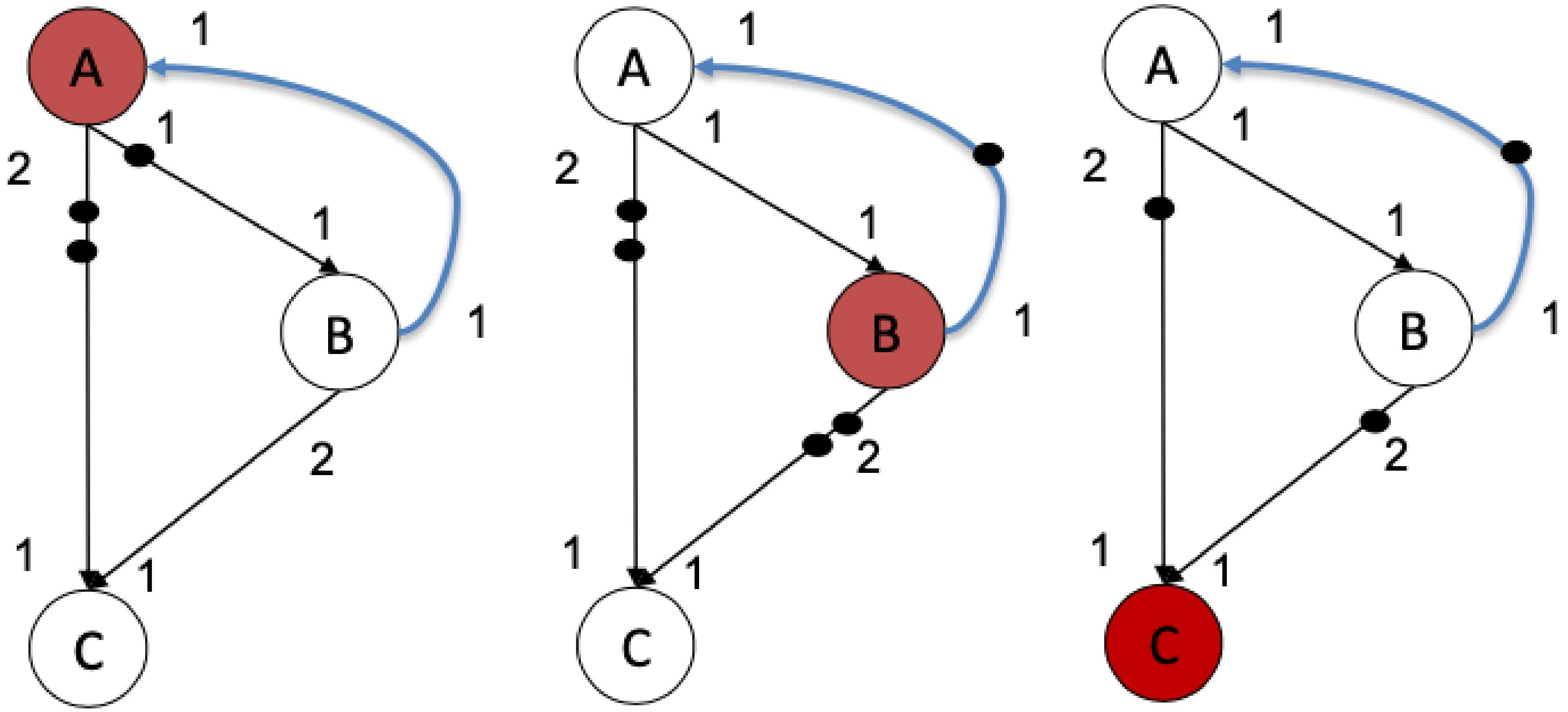
A fires once

B fires ones

C fires twice

Periodic schedule: ABCC

Consistent SDF Simulation II



A fires once

B fires ones

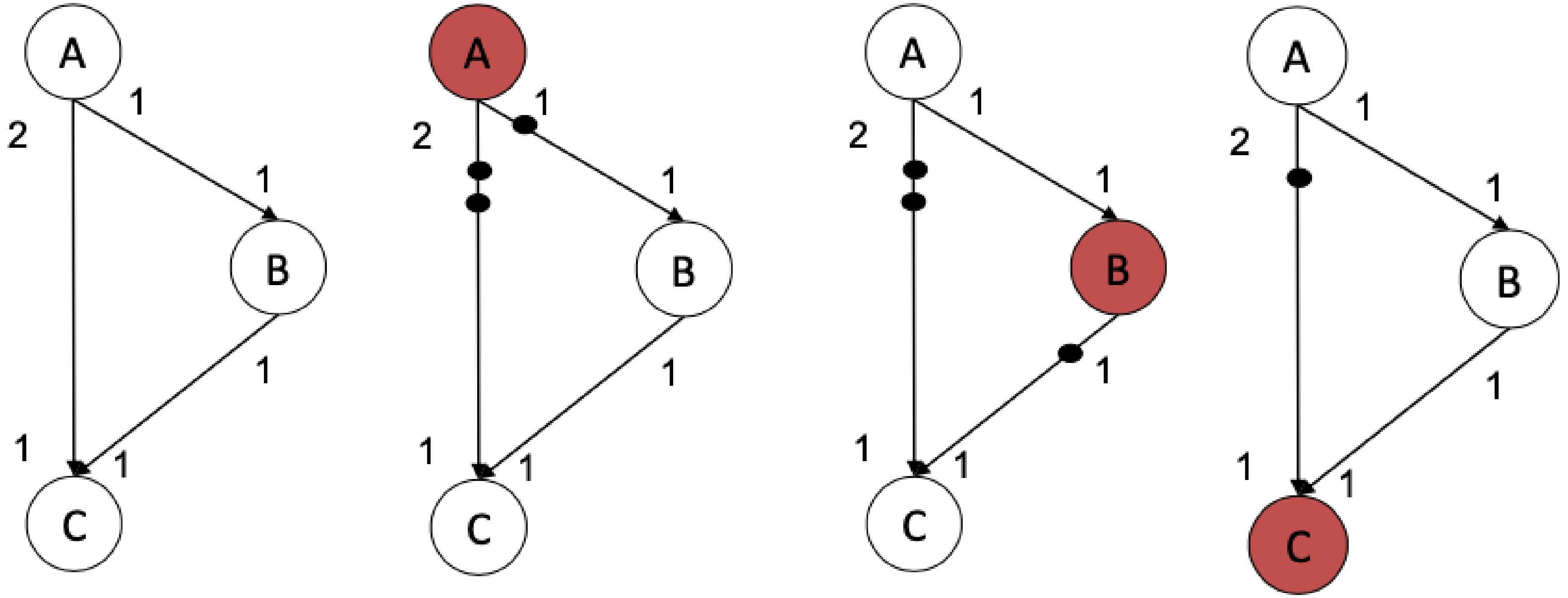
C fires twice

Periodic schedule: ABCC

Periodic Schedule and Consistency

- Firing sequence of a SDF is called a schedule
- A periodic schedule of an SDF clears all channels and returns to its initial status after each node repeats execution a specific finite number of times
- Periodic schedule permits SDF can process unbounded data with bounded memory
- A SDF is consistent iff (if-and-only-if) a periodic schedule exists

Inconsistent SDF



SDF

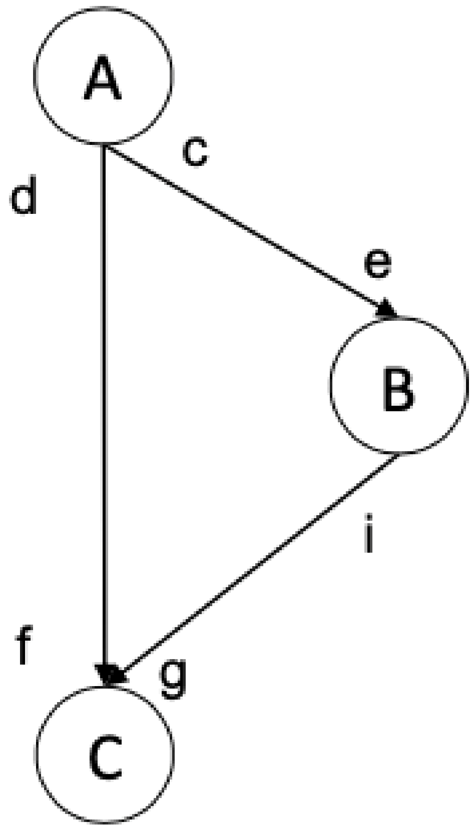
A fires once

B fires ones

C fires twice
and 1 token cannot
be consumed

No periodic schedule

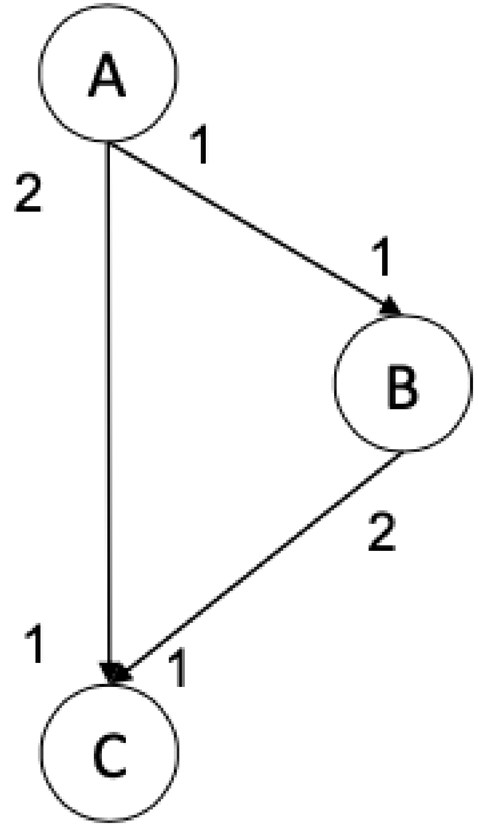
Periodic Schedule and Consistency



- Topology Matrix
  - Each row presents the edge
  - Each column presents a node
  - (i, j): the number of data items placed on i after each invocation of j
  - If i is an input channel for j, element (i, j) is negative

$$\begin{array}{c} \begin{matrix} A & B & C \end{matrix} \\ \begin{pmatrix} c & -e & 0 \\ d & 0 & -f \\ 0 & i & -g \end{pmatrix} \begin{matrix} A \to B \\ A \to C \\ B \to C \end{matrix} \end{array}$$

Periodic Schedule and Consistency



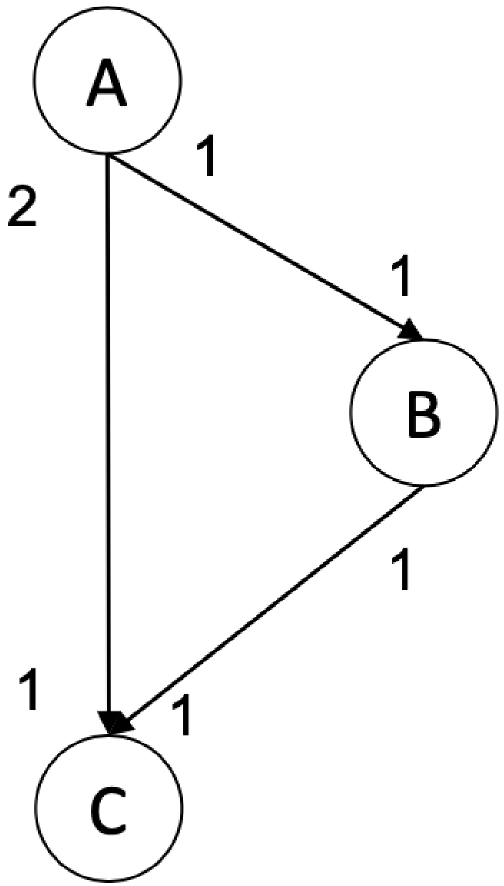
- A necessary condition of a periodic schedule
  - The rank of the topology matrix is $s - 1$, where s is the number of nodes
  - Please refer to Lee's 87 paper for the proof

$$\begin{array}{ccc} A & B & C \end{array}$$

$$\begin{pmatrix} 1 & -1 & 0 \\ 2 & 0 & -1 \\ 0 & 2 & -1 \end{pmatrix} \begin{array}{l} A \rightarrow B \\ A \rightarrow C \\ B \rightarrow C \end{array}$$

Rank = 2

Periodic Schedule and Consistency



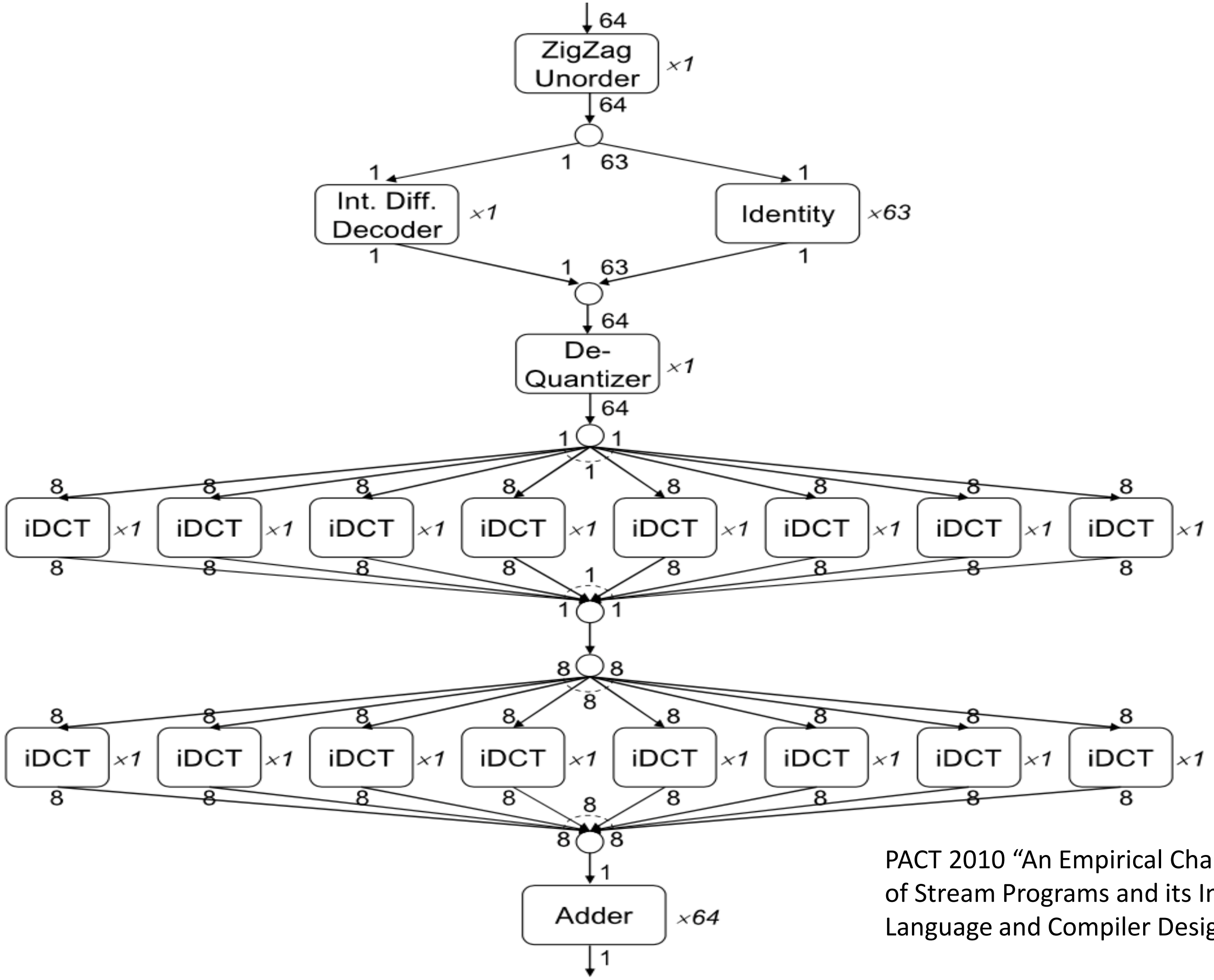
- A necessary condition of a periodic schedule
  - The rank of the topology matrix is $s-1$, where s is the number of nodes
  - Please refer to Lee's 87 paper for the proof

$$\begin{matrix} A & B & C \end{matrix}$$

$$\begin{pmatrix} 1 & -1 & 0 \\ 2 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix} \begin{matrix} A \to B \\ A \to C \\ B \to C \end{matrix}$$

Rank = 3 > 2

Example: Part of JPEG Transcoder



PACT 2010 “An Empirical Characterization of Stream Programs and its Implications for Language and Compiler Design ”

Example Systems Use MBD

- Communications – routers, switches, modem
- Image and/or acoustic processing – CODEC, video broadcasting
- Luggage conveyor systems
- Manufacturing – assembly line
- Vehicles - engine, fuel injection, powertrain

The End

Thanks to you all!