THE UNIVERSITY of York

Design and Validation of Application-specific Multiprocessor Platforms based on Networks-on-Chip

Leandro Soares Indrusiak

lsi@cs.york.ac.uk
http://www-users.cs.york.ac.uk/lsi

CREDES Workshop – Tallinn - September 2010

RTS York

About the speaker

- Leandro Soares Indrusiak
  - born in Santa Maria, RS, Brazil in 1974
  - Electrical Engineer, UFSM, Santa Maria, 1995
  - MSc Computer Science, UFRGS, Porto Alegre, 1998
  - Dr.-Ing Computer Science, jointly issued by UFRGS and TU Darmstadt (Germany), 2003

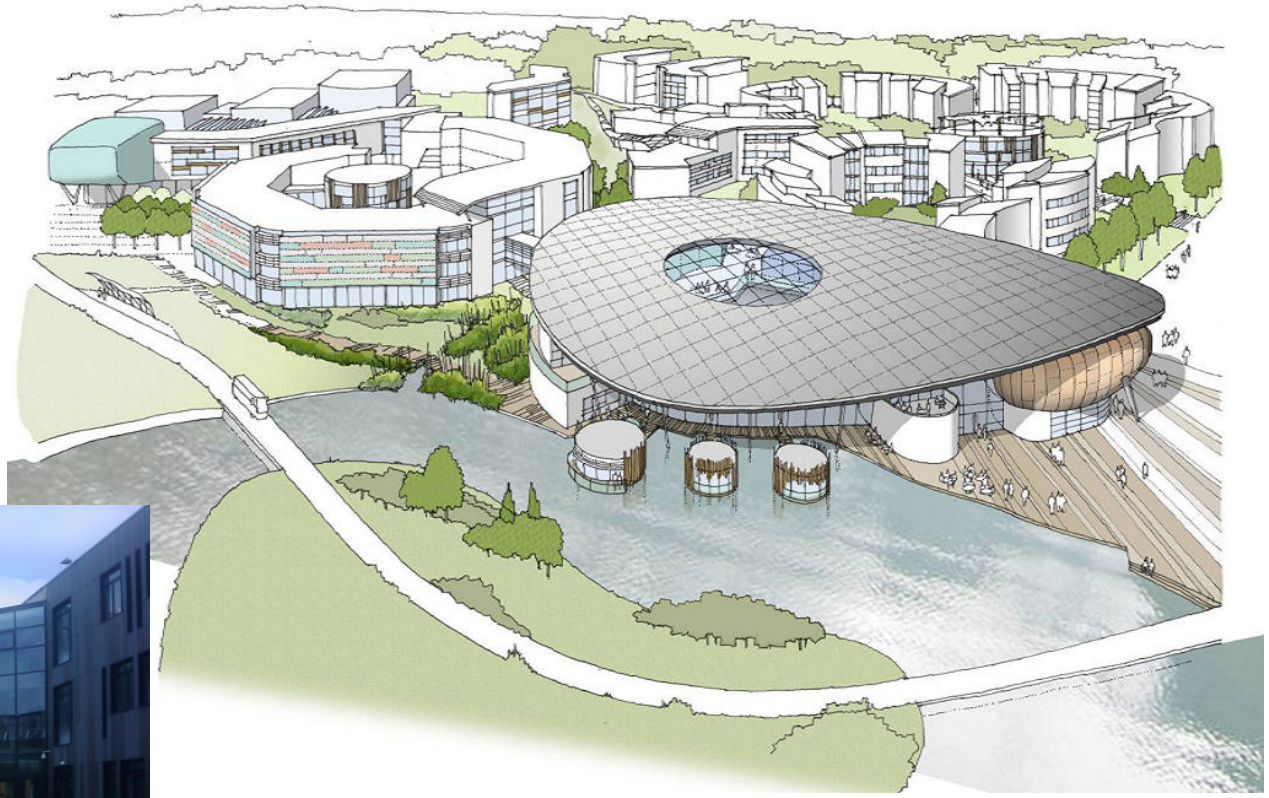
  - 1998 - 2000 Lecturer in Computer Science
    PUCRS, Uruguaiana, Brazil
  - 2003 - 2008 Research Fellow in Microelectronics Design
    TU Darmstadt, Germany
  - 2008 - Lecturer in Real Time Embedded Systems
    University of York, UK

York - Location and history

- York is 200 miles north of London
  - London: 2 hours
  - Manchester airport: 1.5 hours
- Ancient city – historic capital of north – with over 2,000 years of history and heritage, and outstanding quality of life

Heslington West

Heslington East – Phase 1



York's position

UK top 10
World top 100
UK University of the Year 2003
Queen's Anniversary Prize 1996, 2006 & 2008, 2010
Computer Science, Plant Biology, Health Economics, and Social Policy

"One of Britain's academic success stories, forging a reputation to rival Oxford and Cambridge in the space of 40 years" ***Sunday Times***

"The Cambridge of the North"

Financial Times

Computer Science at York

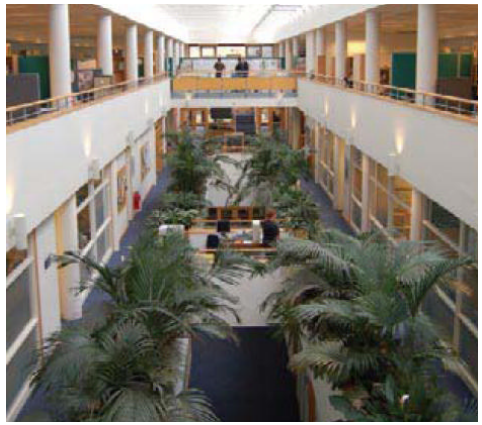
- In numbers
  - 38 full-time academics (lecturers, senior lecturers, readers, professors)
  - 400 undergraduates (BEng, MEng)
  - 130 taught postgraduates (MSc)
  - 100 research students (PhD)

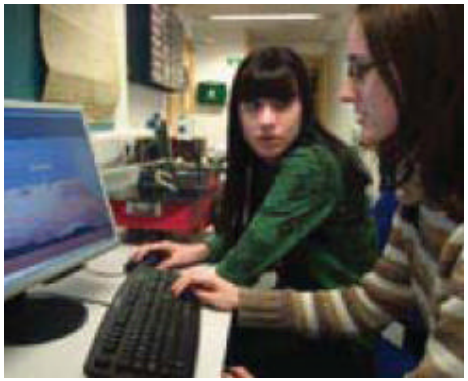
- Consistently ranked among the top CS departments in the UK
- Teaching judged “Excellent” by HEFCE
- Majority of the department’s research judged “world leading” or “internationally excellent”

Computer Science at York - Teaching

- Undergraduates

Programmes
BSc/BEng degree programmes (3 year)
Computer Science
Computer Science with Embedded Systems
Computer Science/Mathematics (Equal)
BSc/BEng degree programmes (4 year sandwich)
Computer Science
Computer Science with Embedded Systems
Computer Science/Mathematics (Equal)
MEng/MMath degree programmes (4 year)
Computer Science with Embedded Systems
Computer Science and Software Engineering
Mathematics / Computer Science (Equal)
MEng/MMath degree programmes (5 year sandwich)
Computer Science with Embedded Systems
Computer Science and Software Engineering
Mathematics / Computer Science (Equal)





Computer Science at York - Teaching

- Graduates

MRes in Computational Biology
MSc/Diploma/Certificate in Gas Turbine Control
MSc in Information Technology
MSc in Natural Computation
MSc/Diploma/Certificate in Safety Critical Systems Engineering
MSc in Software Engineering
Certificate in System Safety Engineering
MSc in Computer Science (by research)
MPhil and PhD

Computer Science at York – Research

Advanced Computer Architectures

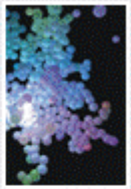
- Parallel Architectures
- Neural Networks
- Dedicated Hardware

Enterprise Systems

- Enterprise Systems
- Computer-Based Learning
- Virtual Working

Non-Standard Computation

- Quantum Computing
- Bio-inspired Computing
- Self-organising Systems



Artificial Intelligence

- Constraint Programming
- Computational Linguistics
- Adaptive & Learning Agents
- Games & Interactive Drama

High Integrity Systems Engineering

- Safety Analysis
- Requirements Assessment & Certification
- Formal Method Support

Computer Vision

- Object Recognition
- Shape Analysis
- Visual Learning
- Probabilistic Networks

Real-Time Systems

- Complex Embedded Systems
- Timing Analysis
- Programming Languages
- Operating System Kernels

Programming Languages & Systems

- Functional Programming
- Concurrency Models
- Reactive Systems Design
- Graph Transformation

Human-Computer Interaction

- Engagement & User Experience
- Multimodal Interaction
- Interactive Design
- Interfaces for Disabled Users

THE UNIVERSITY of York

Design and Validation of Application-specific Multiprocessor Platforms based on Networks-on-Chip

Leandro Soares Indrusiak

lsi@cs.york.ac.uk
http://www-users.cs.york.ac.uk/lsi

CREDES Workshop – Tallinn - September 2010

RTS York

Outline

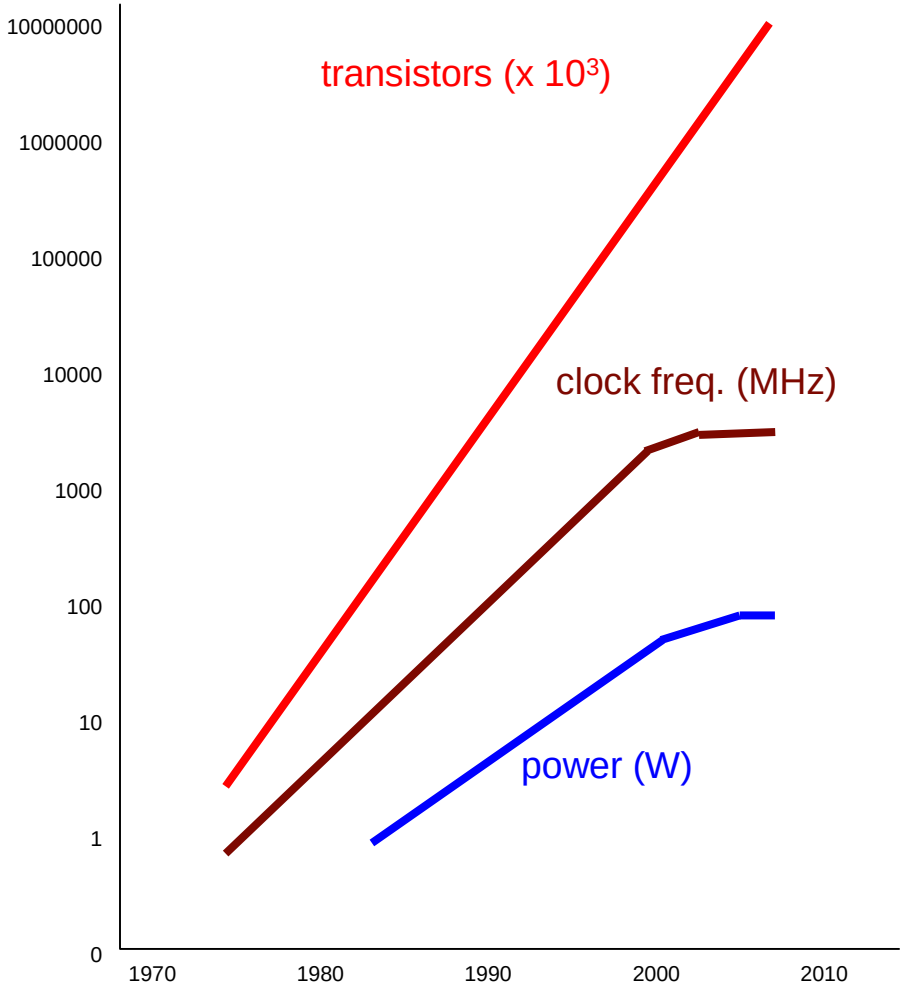
- On-Chip Multiprocessing
  - motivation, review
- Application-specific Multiprocessor Platforms
  - validation, concurrency issues, application-platform mapping
- Platform models
  - evaluation, accuracy
- Application models
  - concurrency
- Joint execution of application and platform models
  - case study
- Background theory and tool support
  - actor orientation, Ptolemy II, custom made extensions
  - demo

Outline

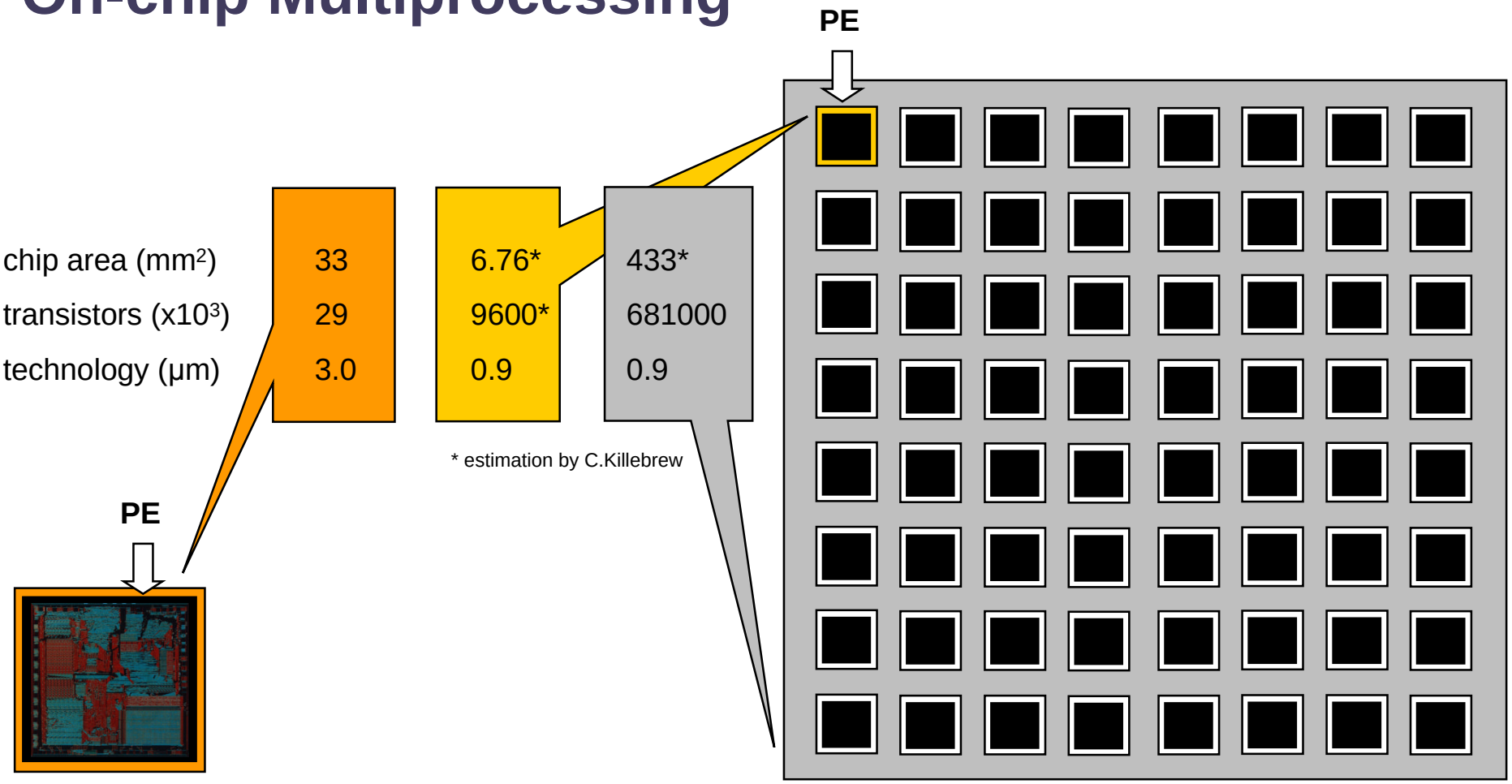
- On-Chip Multiprocessing
  - motivation, review
- Application-specific Multiprocessor Platforms
  - validation, concurrency issues, application-platform mapping
- Platform models
  - evaluation, accuracy
- Application models
  - concurrency
- Joint execution of application and platform models
  - case study
- Background theory and tool support
  - actor orientation, Ptolemy II, custom made extensions
  - demo

The problem with CMOS

- CMOS limits can be felt in many ways
  - number of transistors
  - clock frequency
  - power dissipation
- Expectations of performance increase must be met somehow
  - on-chip multiprocessing seems to be the best shot

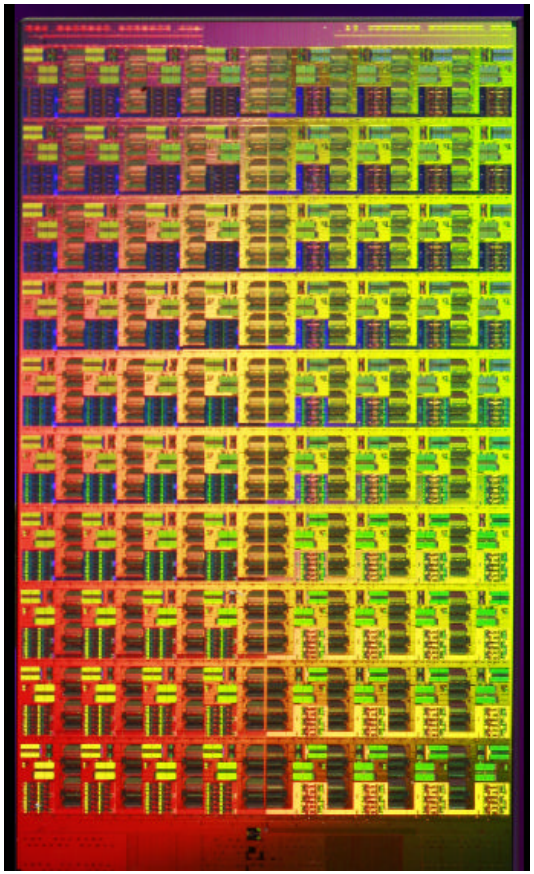


On-chip Multiprocessing



On-chip Multiprocessing

- Many architectural alternatives
  - homogeneous vs. heterogeneous processing
  - shared memory vs. distributed memory
  - on-chip interconnect
    - point-to-point, crossbar
    - on-chip bus
    - network-on-chip

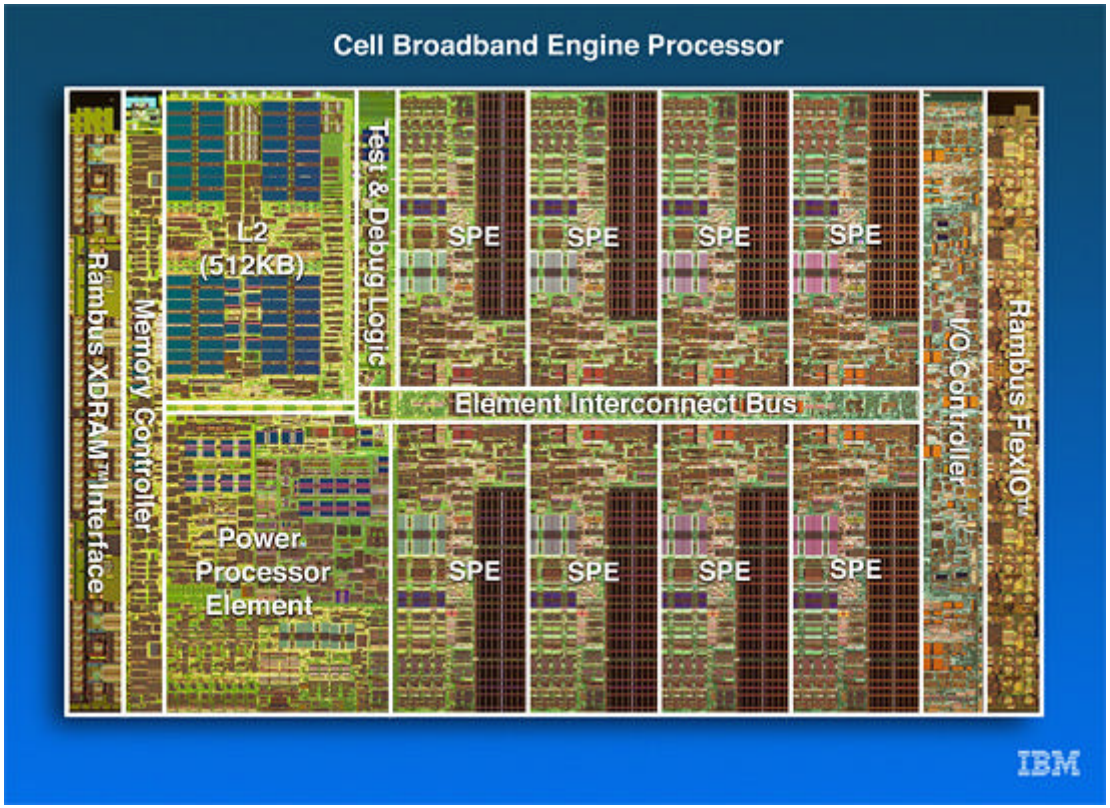


source: Intel

Multiprocessor Platforms for Embedded Systems

Cell Processor

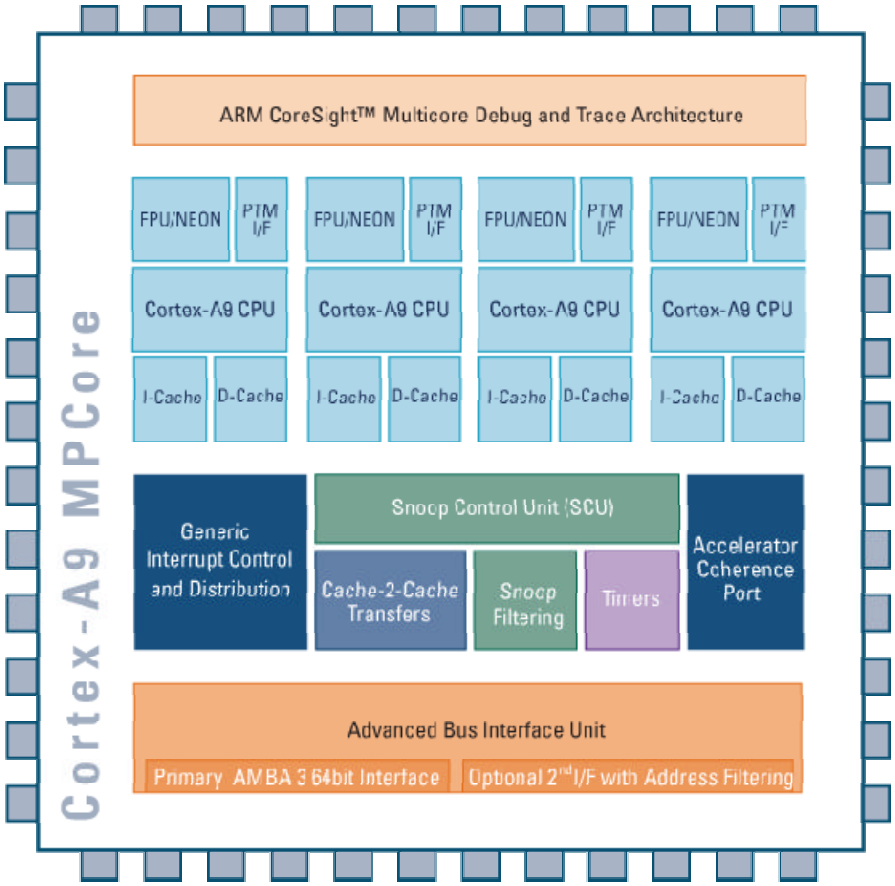
• nine processing elements: one Power Processor Element (PPE) and eight Synergistic Processing Elements (SPE)
• PPE has separated I & D L1 cache (32KB each)
• Each SPE can only access its 256KB of local storage (LS) and uses its memory flow controller (MFE) to perform DMA operations to/from LS (non-blocking)
• Bandwidth (3.2 GHz)
    • SPE <-> LS = 2x 25.6 GB/s
    • MFC <-> EIB = 2x 25.6 GB/s
    • MIC <-> EIB = 2x 25.6 GB/s
    • L1 <-> L2 = 2x 51.2 GB/s



Multiprocessor Platforms for Embedded Systems

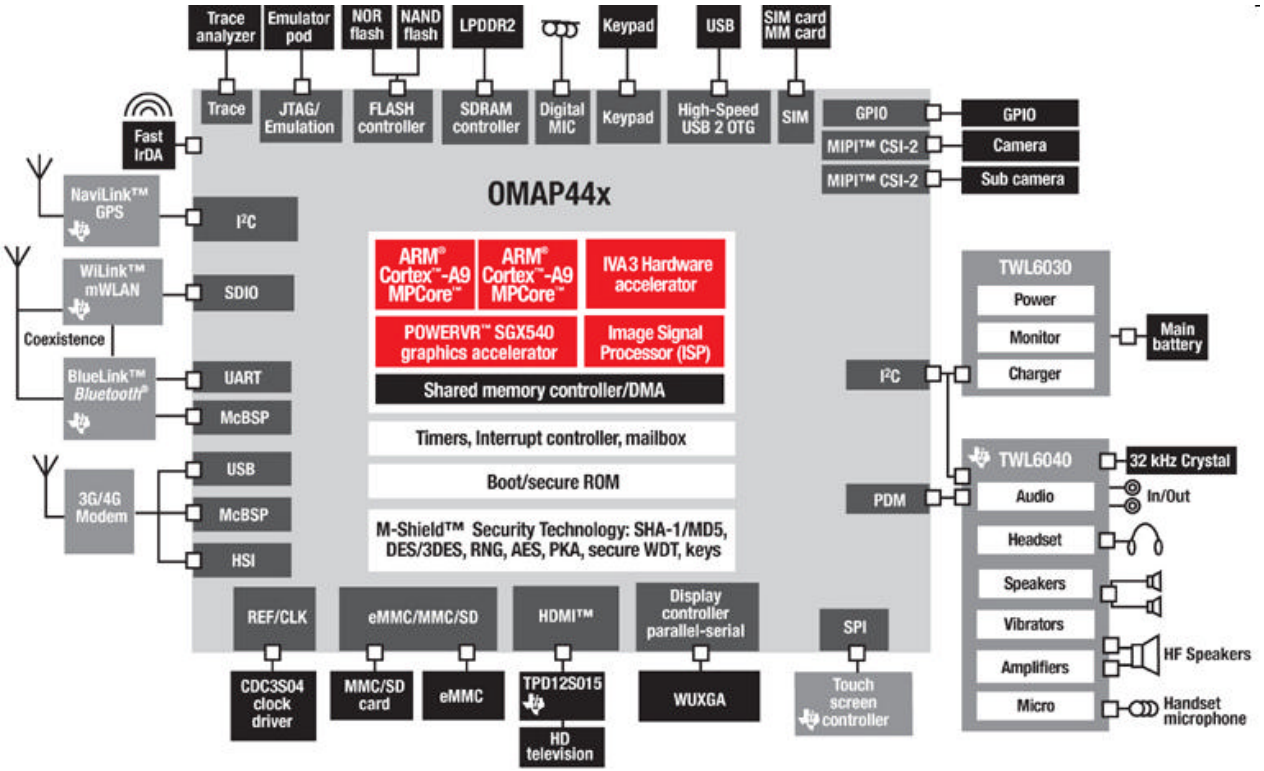
- ARM Cortex-9 MPCore

• can have 1-4 Cortex-A9 cores with separated I & D L1 cache (32KB each)
• data caches of all cores are fully coherent
• interface to external components be made cache-coherent
• on-chip interconnect based on AMBA standard



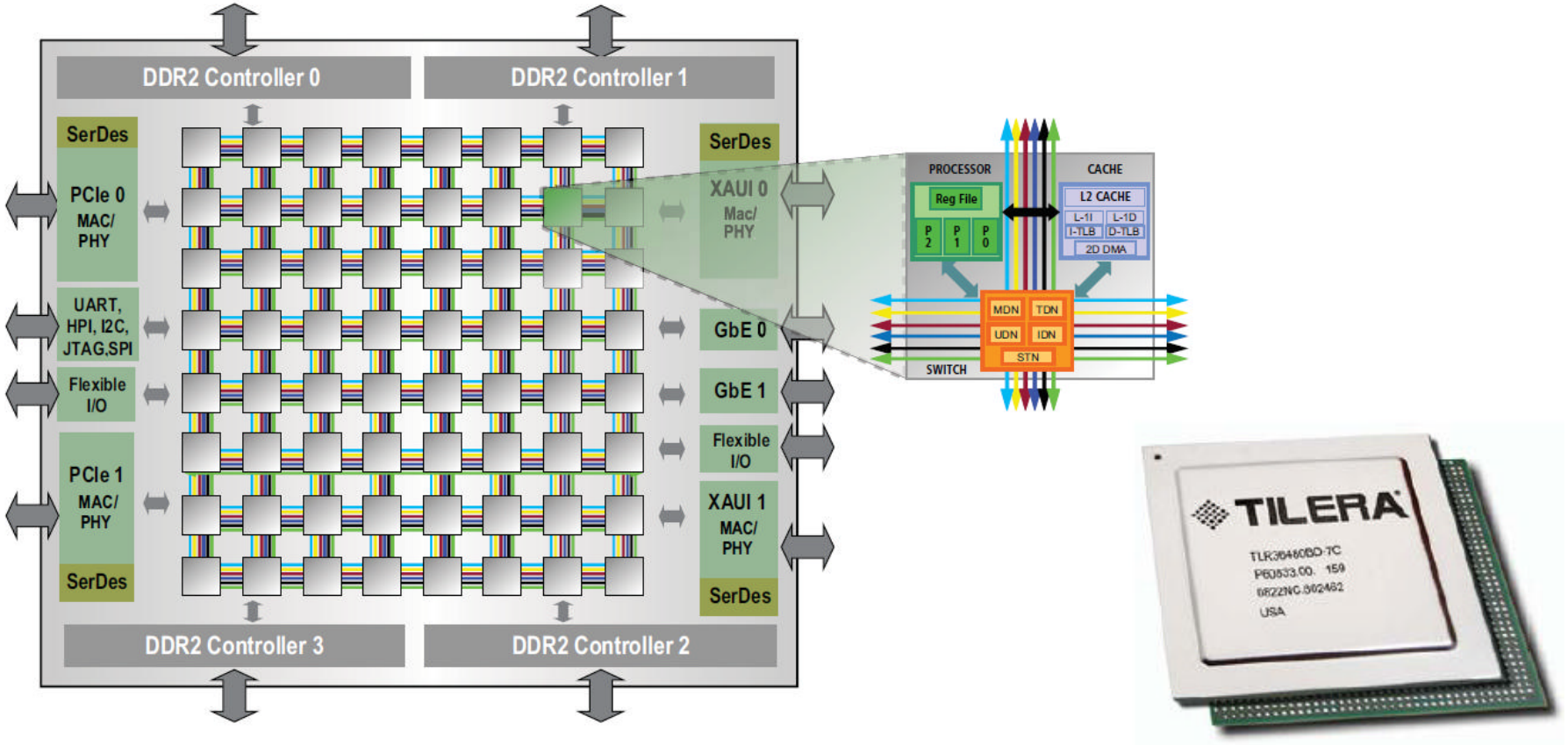
Multiprocessor Platforms for Embedded Systems

- ARM Cortex-9 MPCore – TI OMAP 4 Platform



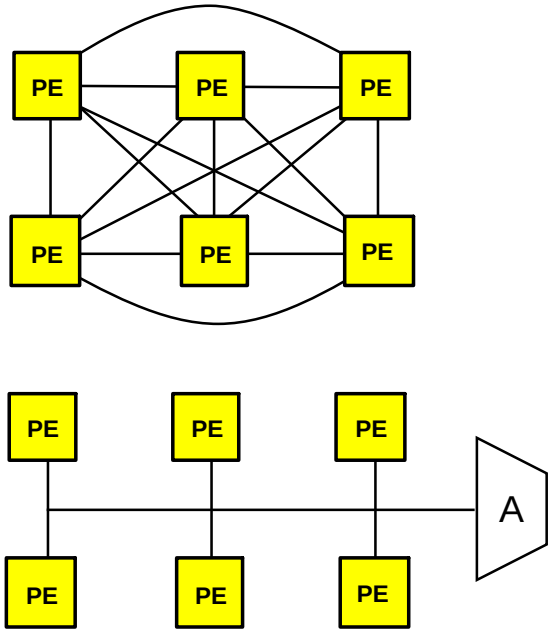
Multiprocessor Platforms for Embedded Systems

- Tilera TILEPro64



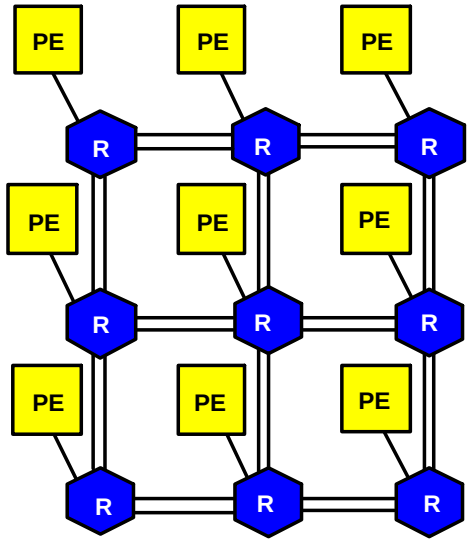
Multiprocessor Platforms for Embedded Systems

- As the number of processing elements increase, on-chip communication becomes a major issue
  - point-to-point: huge area overhead
  - on-chip bus: doesn't scale (restricted parallelism)
  - in both cases: long wires cause timing issues and excessive power consumption



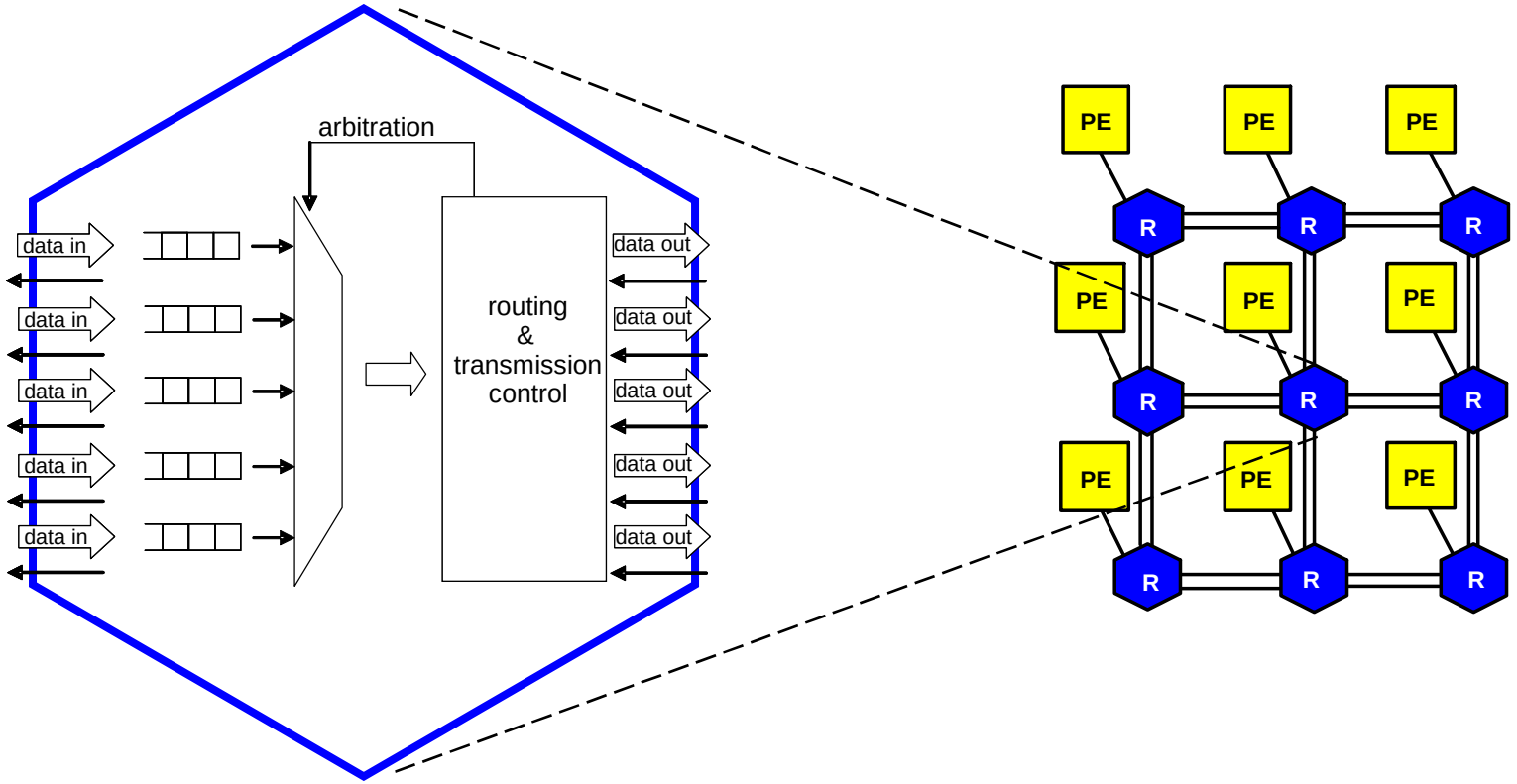
Multiprocessor Platforms for Embedded Systems

- Networks-on-Chip (NoC)
  - multi-hop, packet-based communication
  - scalable, high bandwidth
  - low power (shorter wires)
  - distributed arbitration
  - regular, reusable
  - supports GALS paradigm
  - communication latency
  - area
  - complex to design



Multiprocessor Platforms for Embedded Systems

- Networks-on-Chip (NoC)



Outline

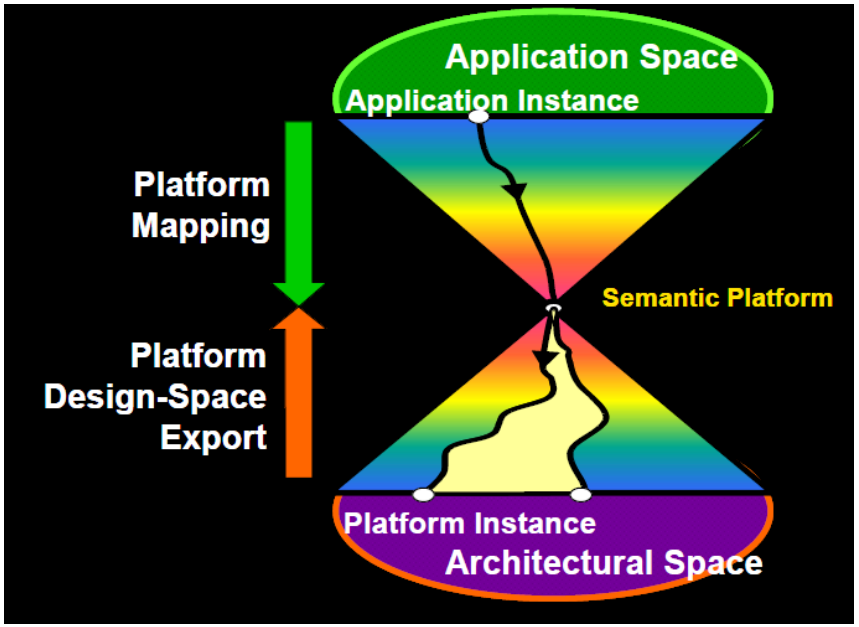
- On-Chip Multiprocessing
  - motivation, review
- Application-specific Multiprocessor Platforms
  - validation, concurrency issues, application-platform mapping
- Platform models
  - evaluation, accuracy
- Application models
  - concurrency
- Joint execution of application and platform models
  - case study
- Background theory and tool support
  - actor orientation, Ptolemy II, custom made extensions
  - demo

Application-specific Multiprocessor Platforms

- Fundamental questions
  - can I meet the constraints of a given application A if I run it over platform P?
    - performance and timing
    - power
  - no? isn't there a way to optimise it?
    - mapping and scheduling
    - voltage and frequency scaling
  - yes? can I guarantee it?
    - hard/soft real-time

Application-specific Multiprocessor Platforms

- Many architectural alternatives
  - large design space
  - hard to evaluate which alternative platform is the optimal match for the requirements imposed by a given application

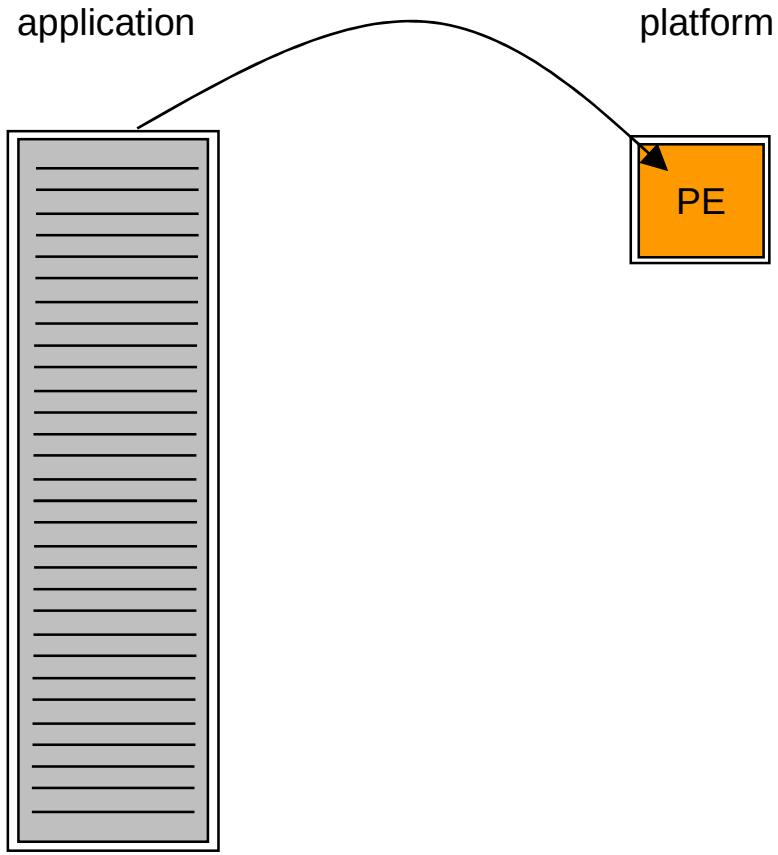


Alberto Sangiovanni-Vincentelli

- Application-platform mapping plays a critical role

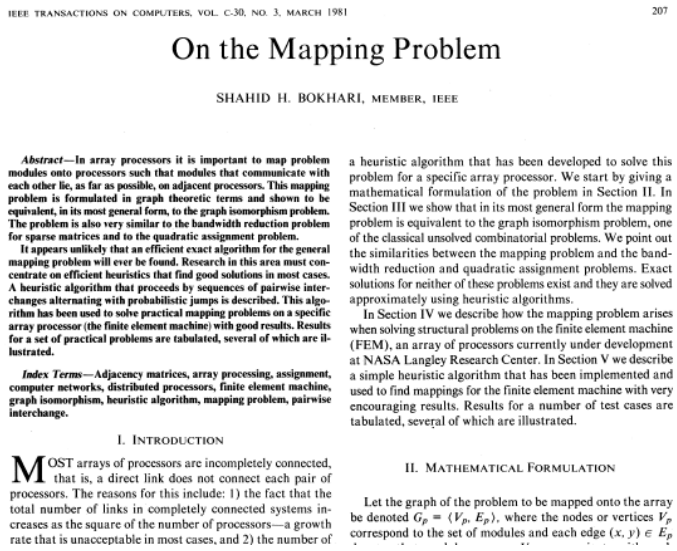
Application-specific Multiprocessor Platforms

- Application-Platform mapping isn't really a problem when dealing with sequential software and uniprocessor hardware



Application-specific Multiprocessor Platforms

- It has been studied for decades by researchers in parallel and distributed computing



IEEE TRANSACTIONS ON COMPUTERS, VOL. C-30, NO. 3, MARCH 1981 207

On the Mapping Problem

SHAHID H. BOKHARI, MEMBER, IEEE

Abstract—In array processors it is important to map problem modules onto processors such that modules that communicate with each other lie, as far as possible, on adjacent processors. This mapping problem is formulated in graph theoretic terms and shown to be equivalent, in its most general form, to the graph isomorphism problem. The problem is also very similar to the bandwidth reduction problem for sparse matrices and to the quadratic assignment problem.

It appears unlikely that an efficient exact algorithm for the general mapping problem will ever be found. Research in this area must concentrate on efficient heuristics that find good solutions in most cases. A heuristic algorithm that proceeds by sequences of pairwise interchanges alternating with probabilistic jumps is described. This algorithm has been used to solve practical mapping problems on a specific array processor (the finite element machine) with good results. Results for a set of practical problems are tabulated, several of which are illustrated.

Index Terms—Adjacency matrices, array processing, assignment, computer networks, distributed processors, finite element machine, graph isomorphism, heuristic algorithm, mapping problem, pairwise interchange.

I. Introduction

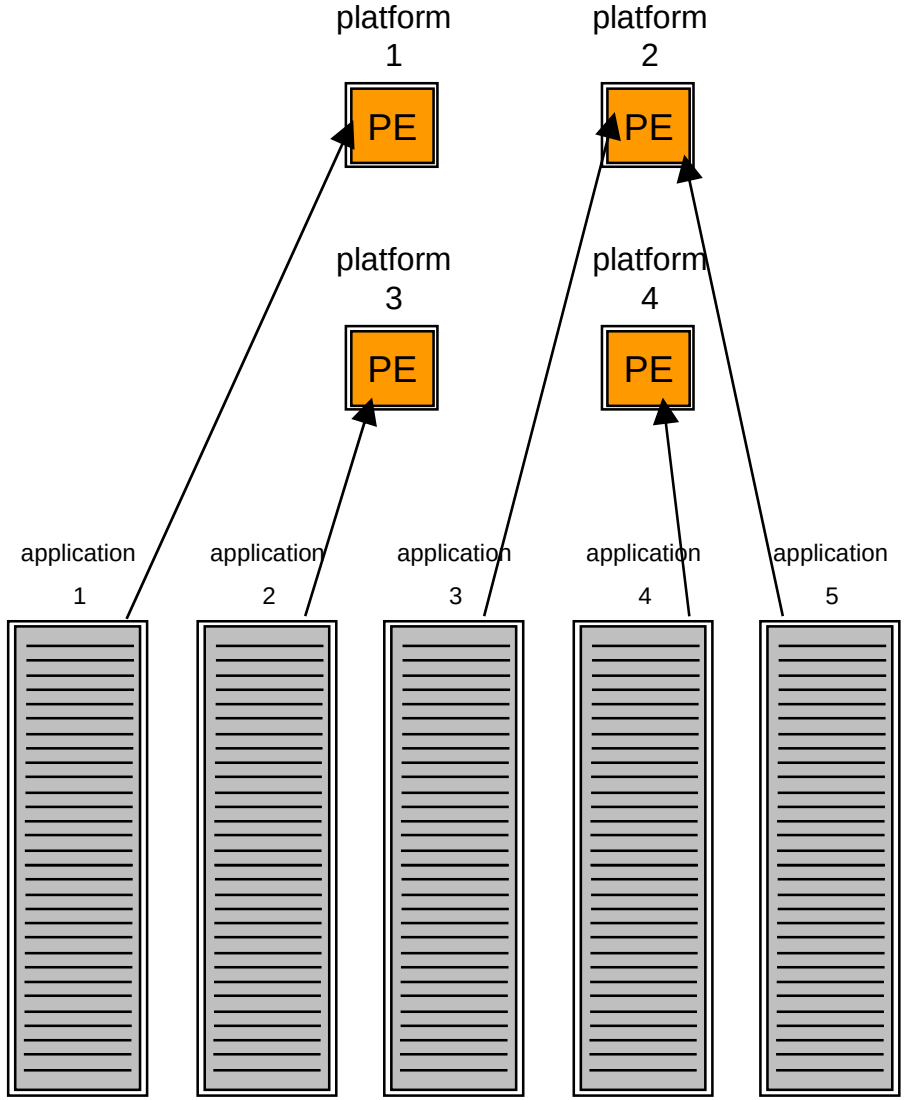
MOST arrays of processors are incompletely connected, that is, a direct link does not connect each pair of processors. The reasons for this include: 1) the fact that the total number of links in completely connected systems increases as the square of the number of processors—a growth rate that is unacceptable in most cases, and 2) the number of

a heuristic algorithm that has been developed to solve this problem for a specific array processor. We start by giving a mathematical formulation of the problem in Section II. In Section III we show that in its most general form the mapping problem is equivalent to the graph isomorphism problem, one of the classical unsolved combinatorial problems. We point out the similarities between the mapping problem and the bandwidth reduction and quadratic assignment problems. Exact solutions for neither of these problems exist and they are solved approximately using heuristic algorithms.

In Section IV we describe how the mapping problem arises when solving structural problems on the finite element machine (FEM), an array of processors currently under development at NASA Langley Research Center. In Section V we describe a simple heuristic algorithm that has been implemented and used to find mappings for the finite element machine with very encouraging results. Results for a number of test cases are tabulated, several of which are illustrated.

II. Mathematical Formulation

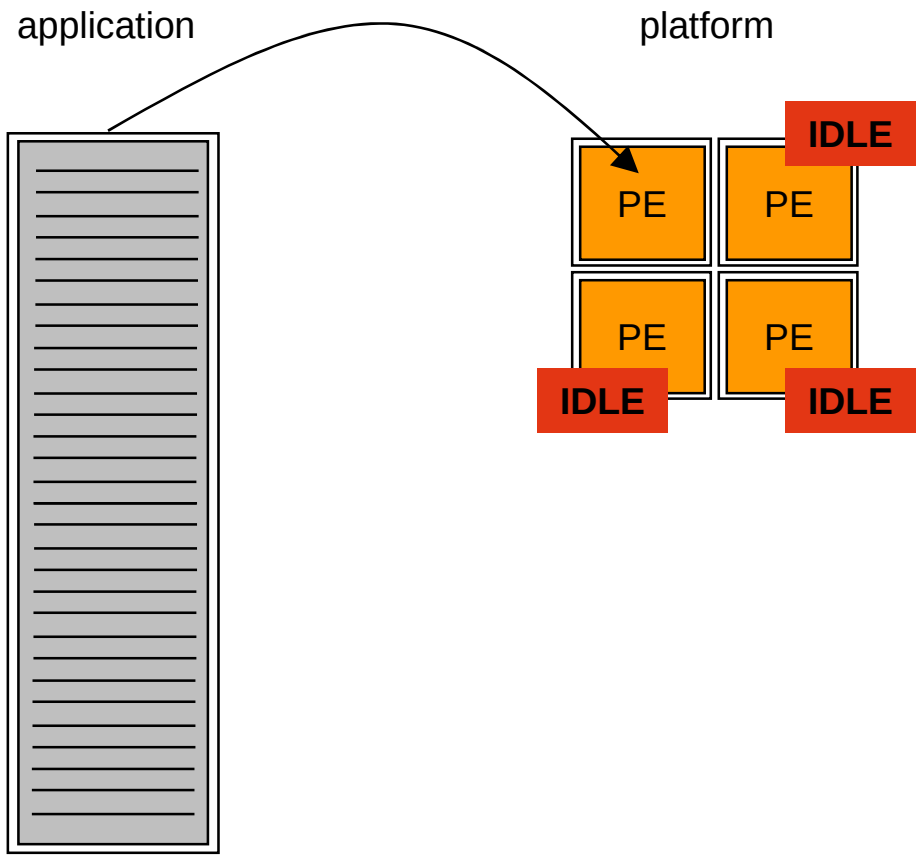
Let the graph of the problem to be mapped onto the array be denoted $G_p = (V_p, E_p)$, where the nodes or vertices V_p correspond to the set of modules and each edge $(x, y) \in E_p$



Application-specific Multiprocessor Platforms

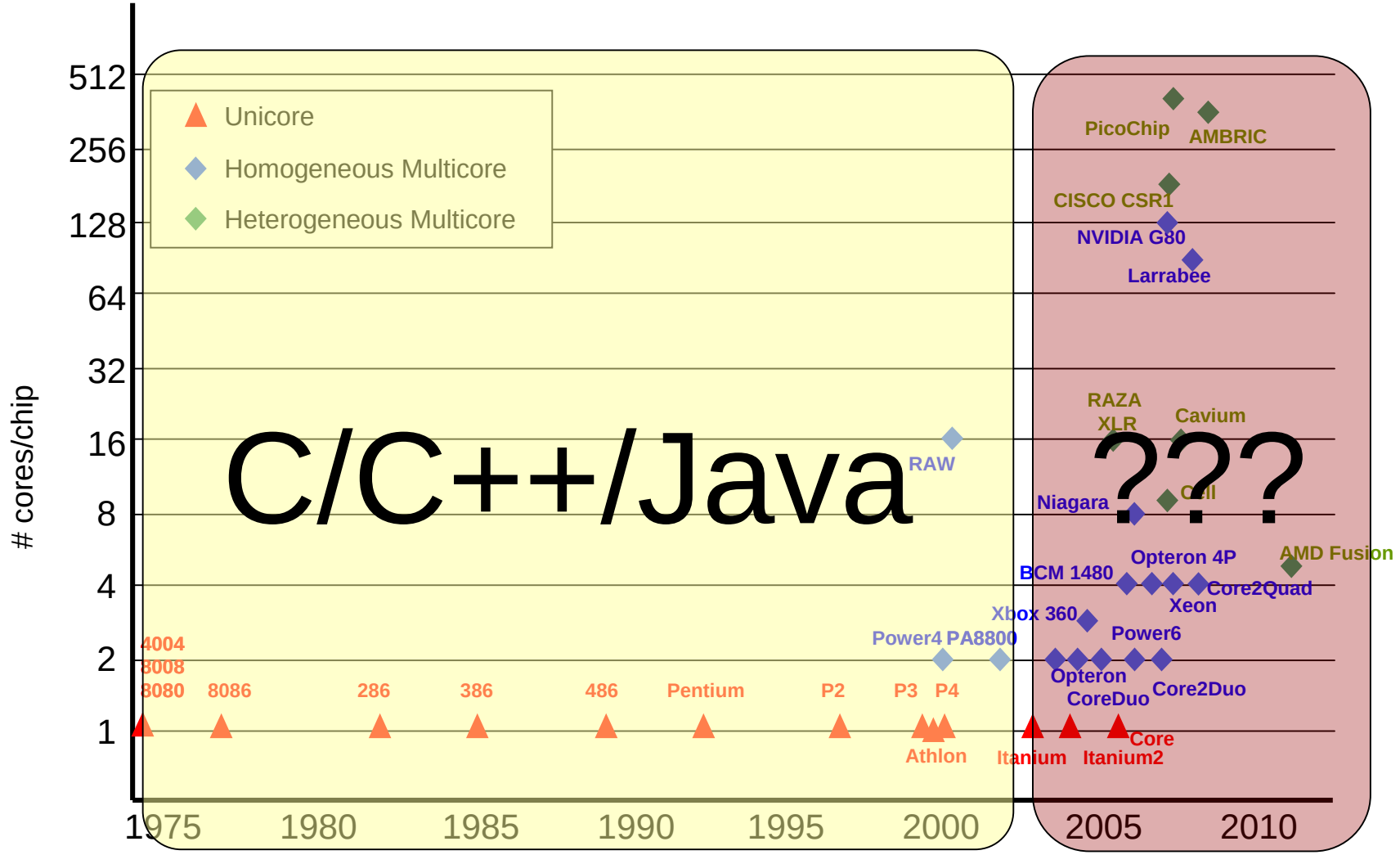
- Application-Platform mapping is a critical issue when it comes to multiprocessor platforms

REALLY?



Application-specific Multiprocessor Platforms

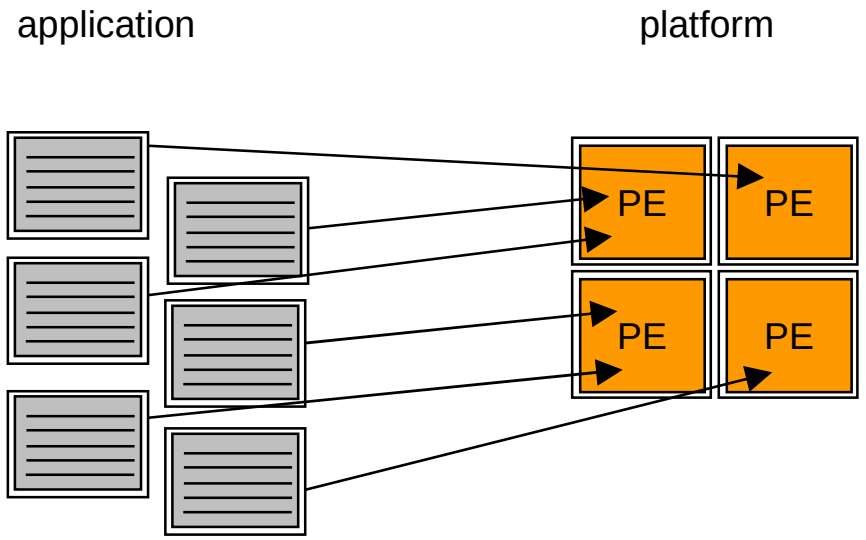
Sources: Gordon ASPLOS'06, Kudlur PLDI'08



Application-specific Multiprocessor Platforms

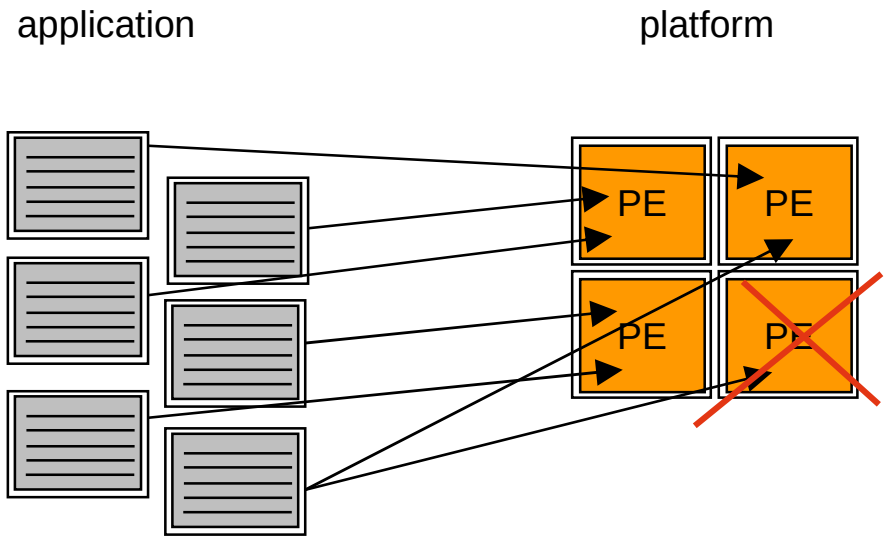
- Application-Platform mapping is a critical issue when it comes to multiprocessor platforms

must be able to explore concurrency at the application level



Application-specific Multiprocessor Platforms

- Application-Platform mapping is a critical issue when it comes to multiprocessor platforms



can be done dynamically to improve performance or to increase dependability

Application-Platform Mapping

- The simplest formulation of the mapping problem resembles a graph isomorphism problem
  - application is a graph $\mathcal{G}$ = G(A, C), where $a_i \in$ A is an application task and $c_{i,j} \in$ C represents the communication from a_i to a_j
  - platform is a graph $\mathcal{G}'$ = G(B, D), where $b_i \in$ B is a processor and $d_{i,j} \in$ D represents the channel from b_i to b_j
  - objective is to map tasks onto processors such that task that communicate with each other lie on adjacent processors
- There is no known polynomial time solution for the graph isomorphism problem – NP-hard

Application-Platform Mapping

- More sophisticated problem formulations may include additional information to the application and platform graphs, so that different objectives can be met

  platform:
  processing power of b_i
  power consumption and latency of $d_{i,j}$

  application:
  computational cost and deadline of a_i
  volume, max latency and required bandwidth of $c_{i,j}$

  - objectives: reduce bandwidth requirements on the platform, reduce power consumption, balance thermal dissipation, etc.

Application-Platform Mapping

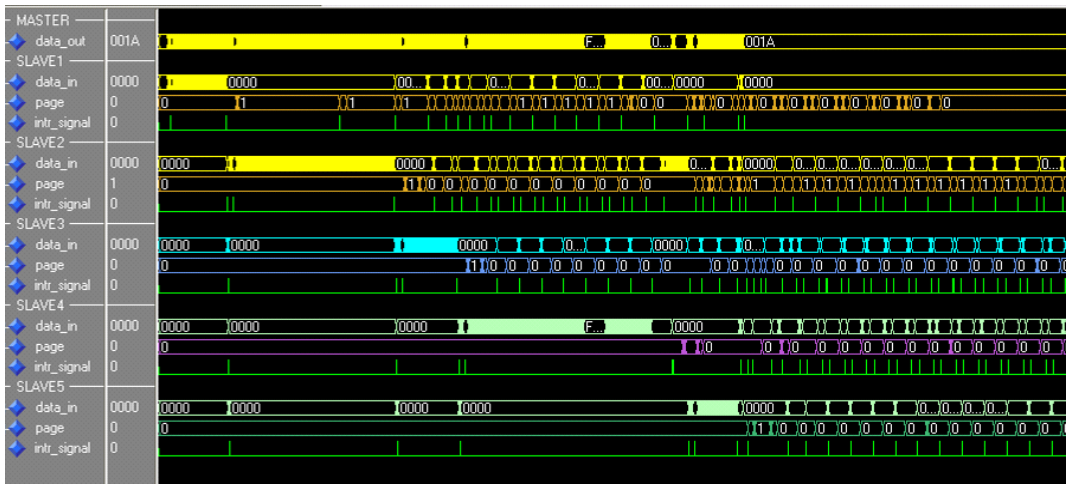
- Platform models over-simplify the complex design space of on-chip multiprocessor platforms
- Application and Platform models disregard the temporal dimension
  - must include further details on time and concurrency

Outline

- On-Chip Multiprocessing
  - motivation, review
- Application-specific Multiprocessor Platforms
  - validation, concurrency issues, application-platform mapping
- Platform models
  - evaluation, accuracy
- Application models
  - concurrency
- Joint execution of application and platform models
  - case study
- Background theory and tool support
  - actor orientation, Ptolemy II, custom made extensions
  - demo

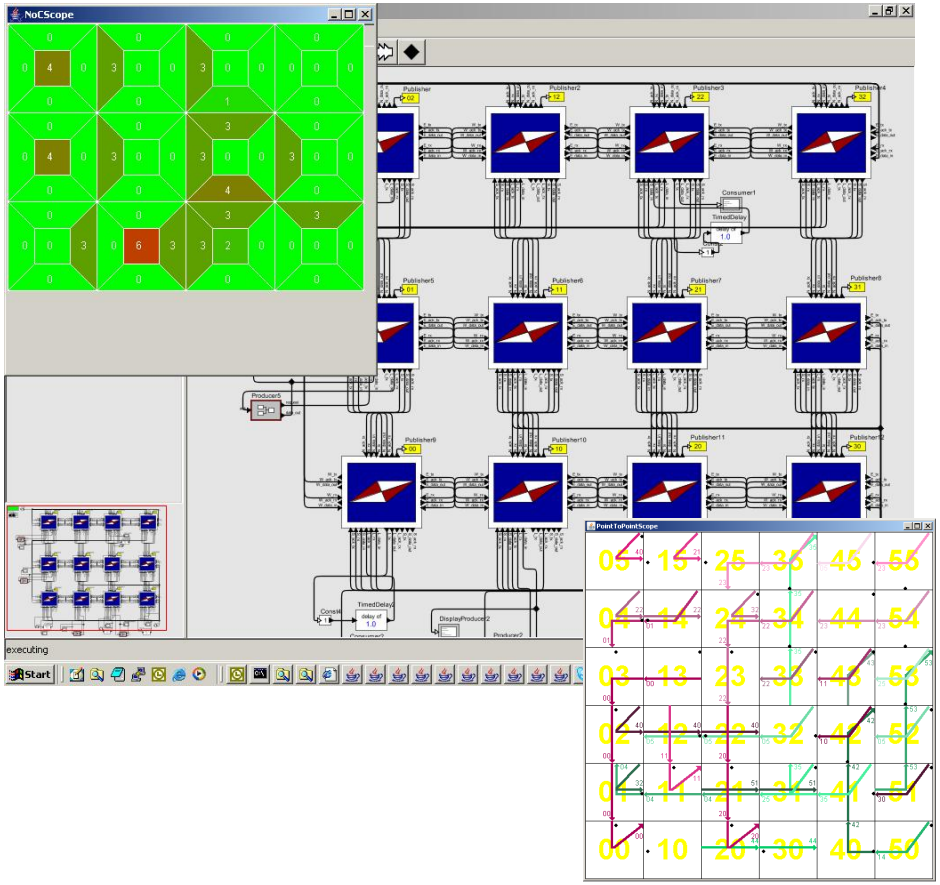
Platform models

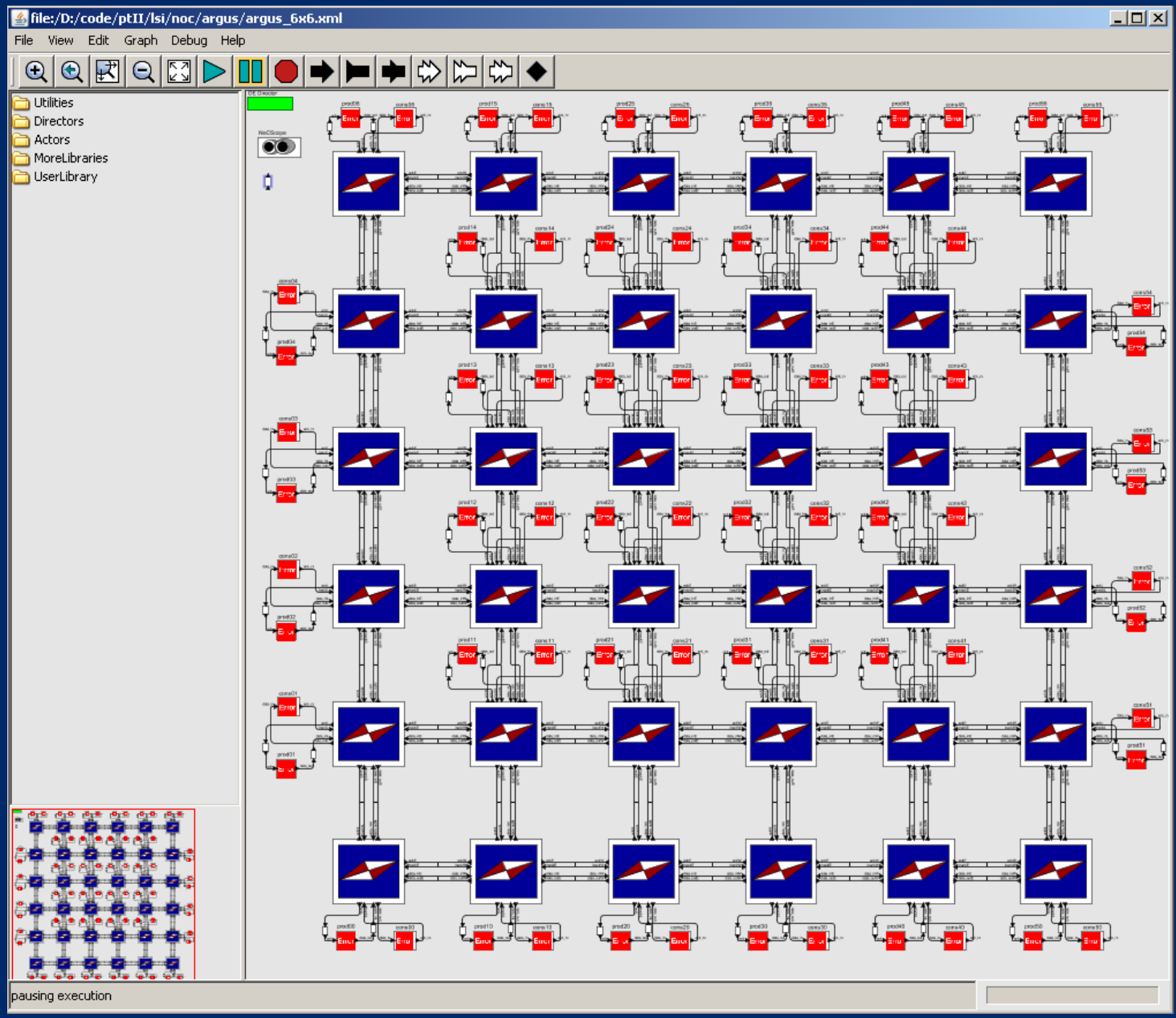
- How to evaluate the quality of platform alternatives?
- HDL-based approach
  - cycle-accurate model of the platform
    - hard to create, debug and modify
    - synthesisable
  - traffic generators
  - DE simulator

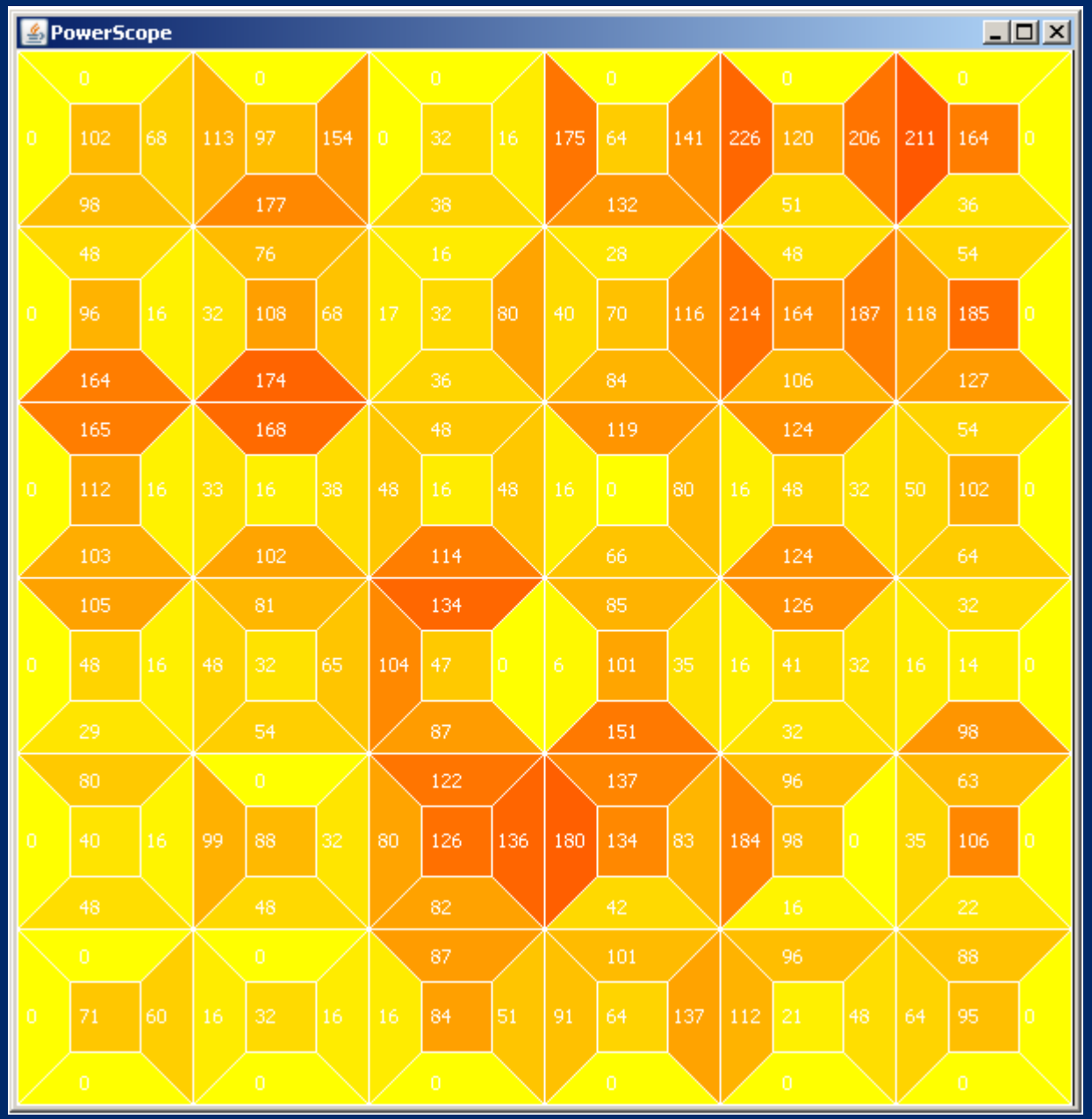


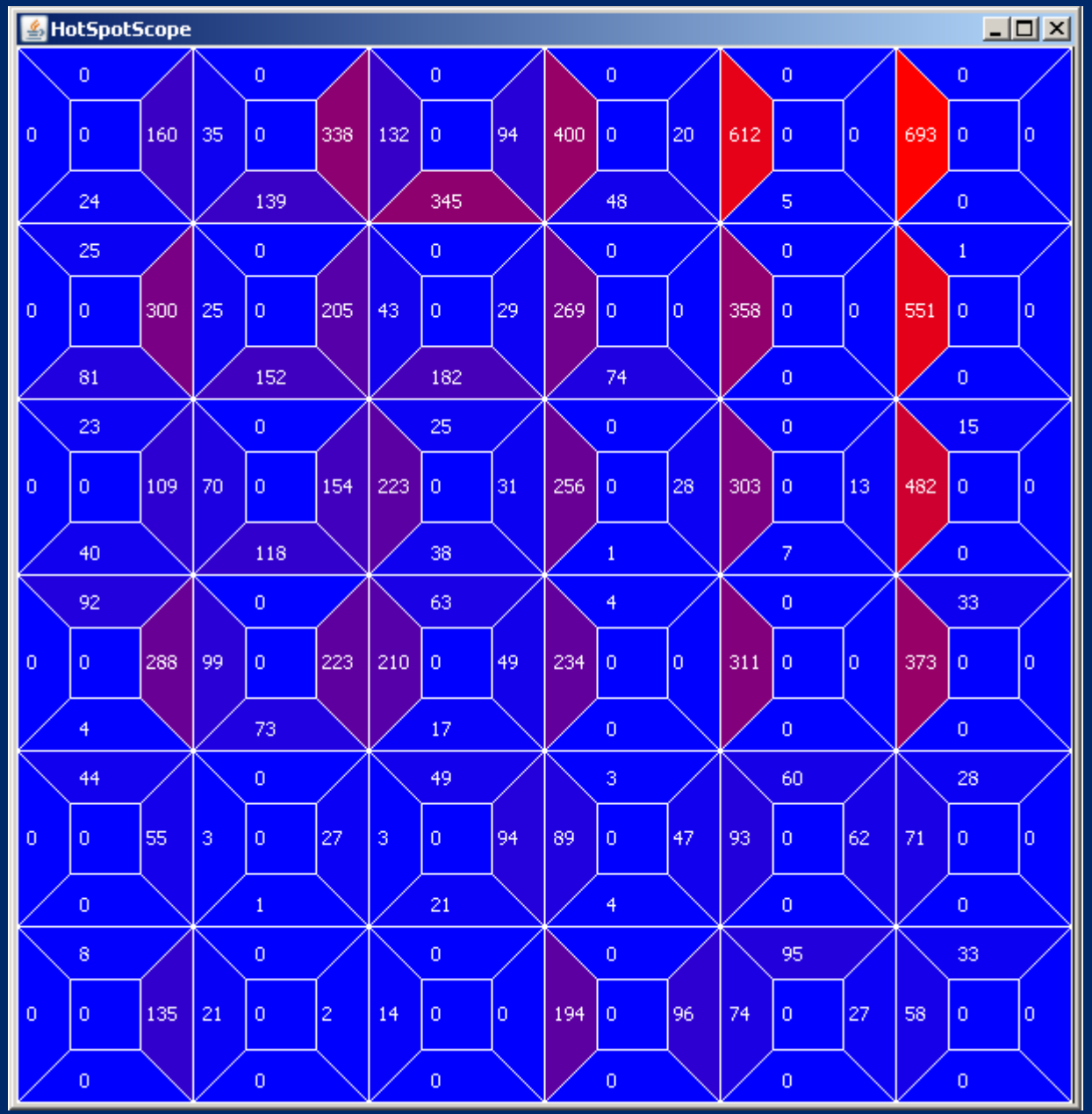
Platform models

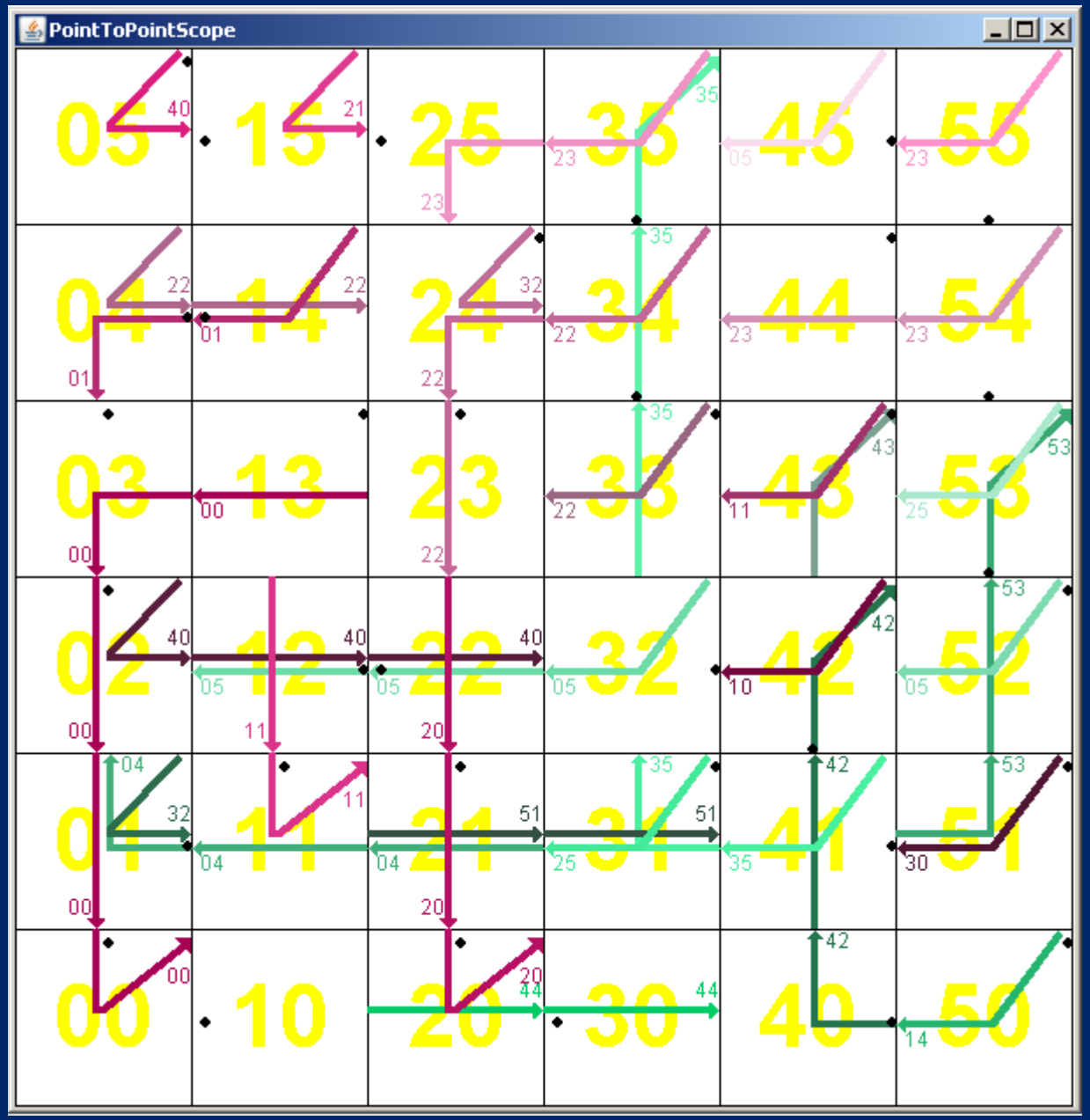
- How to evaluate the quality of platform alternatives?
- Abstract NoC models
  - cycle/flit-level accuracy
  - PAT (payload abstraction)
  - static analysis
  - hybrid
- Modelling trade-offs
  - accuracy
  - observability
  - simulation speed




file:/D:/code/ptII/lsi/noc/argus/argus_6x6.xml
File View Edit Graph Debug Help
Utilities
Directors
Actors
MoreLibraries
UserLibrary
pausing execution


PowerScope
0 0 0 0 0 0
0 102 68 113 97 154 0 32 16 175 64 141 226 120 206 211 164 0
98 177 38 132 51 36
48 76 16 28 48 54
0 96 16 32 108 68 17 32 80 40 70 116 214 164 187 118 185 0
164 174 36 84 106 127
165 168 48 119 124 54
0 112 16 33 16 38 48 16 48 16 0 80 16 48 32 50 102 0
103 102 114 66 124 64
105 81 134 85 126 32
0 48 16 48 32 65 104 47 0 6 101 35 16 41 32 16 14 0
29 54 87 151 32 98
80 0 122 137 96 63
0 40 16 99 88 32 80 126 136 180 134 83 184 98 0 35 106 0
48 48 82 42 16 22
0 0 87 101 96 88
0 71 60 16 32 16 16 84 51 91 64 137 112 21 48 64 95 0
0 0 0 0 0 0


HotSpotScope
0 0 0 0 0 0
0 0 160 35 0 338 132 0 94 400 0 20 612 0 0 693 0 0
24 139 345 48 5 0
25 0 0 0 0 1
0 0 300 25 0 205 43 0 29 269 0 0 358 0 0 551 0 0
81 152 182 74 0 0
23 0 25 0 0 0 15
0 0 109 70 0 154 223 0 31 256 0 28 303 0 13 482 0 0
40 118 38 1 7 0
92 0 63 4 0 33
0 0 288 99 0 223 210 0 49 234 0 0 311 0 0 373 0 0
4 73 17 0 0 0
44 0 49 3 60 28
0 0 55 3 0 27 3 0 94 89 0 47 93 0 62 71 0 0
0 1 21 4 0 0
8 0 0 0 95 33
0 0 135 21 0 2 14 0 0 194 0 96 74 0 27 58 0 0
0 0 0 0 0 0

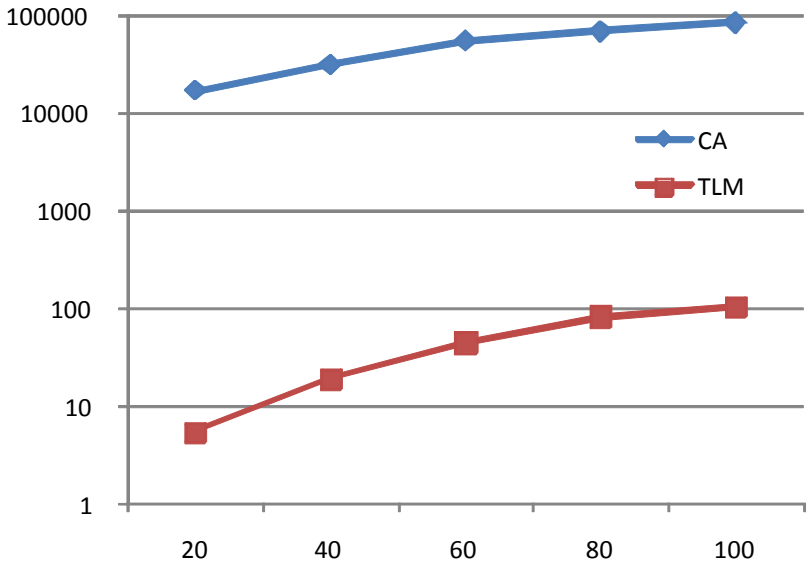

PointToPointScope

Platform models

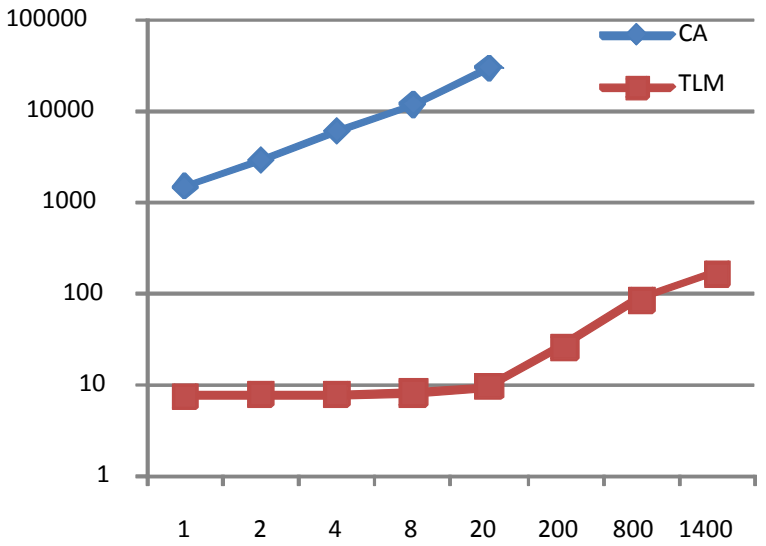
- Simplified models
  - PAT
    - packet-level accuracy, 1-position buffers
    - complete packet is abstracted by header and trailler
    - header‘s way through the network is fully simulated, trailler latencies are calculated
    - less than 5% error for average latency (compared with cycle-accurate HDL)
  - real-time analysis
    - priority-based virtual channels
    - static analysis of worst-case interference between network flows
  - hybrid model - transaction-level (TLM)
    - priority-based virtual channels
    - analysis of interference between network flows
    - less than 10% error for average and worst case latency (compared with cycle-accurate model)

Platform models

- Simplified models

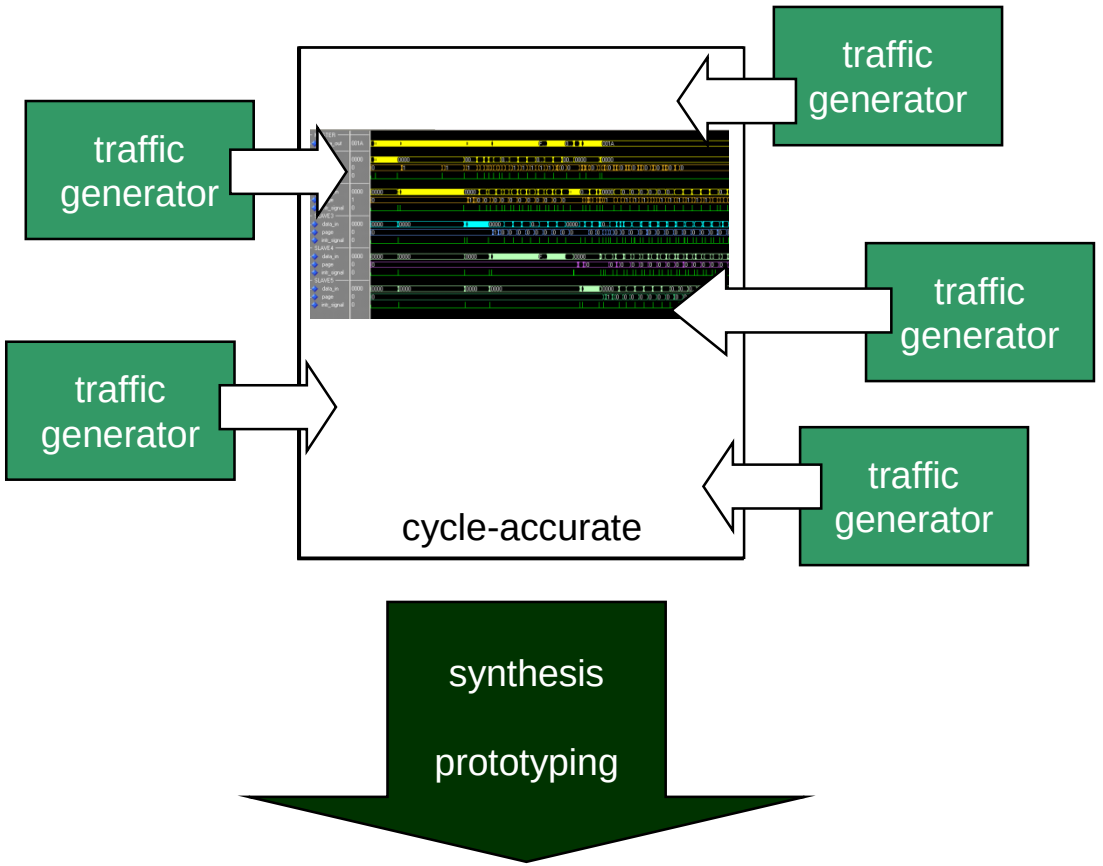


application flows	simulation time (s) cycle-accurate	TLM
20	17105.42	5.47
40	31490.13	19.17
60	55561.87	45.78
80	70231.76	83.06
100	86626.07	105.28



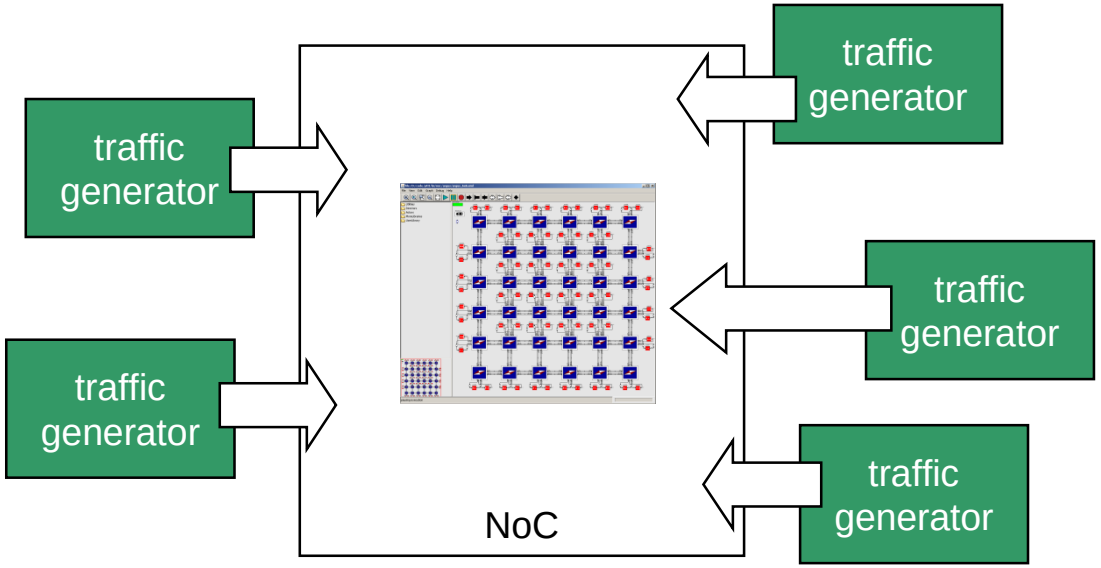
application time (s)	simulation time (s) cycle-accurate	TLM
1	1465.18	7.67
2	2895.69	7.89
4	5978.32	7.98
8	11806.91	8.28
20	29927.24	9.77
200		26.55
800		86.29
1400		169.06

Platform models



Platform models

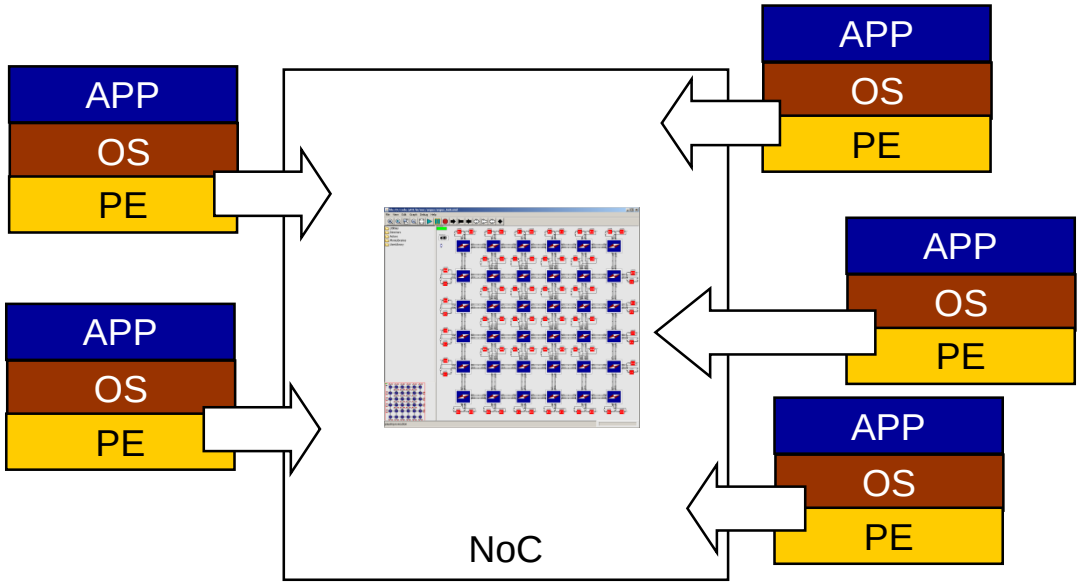
- But how useful are those traffic generators?



- critical point if we are targeting application-specific platforms

Platform models

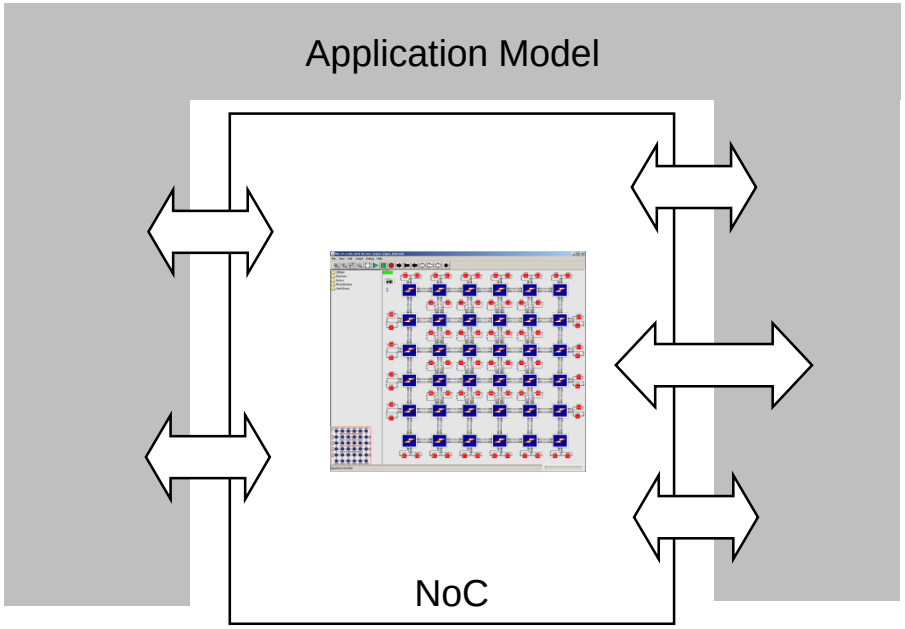
- Can we have models of actual processors, OS and application software?



- yes, but it will be slow to simulate and hard to debug/modify

Application models

- We use an application model instead

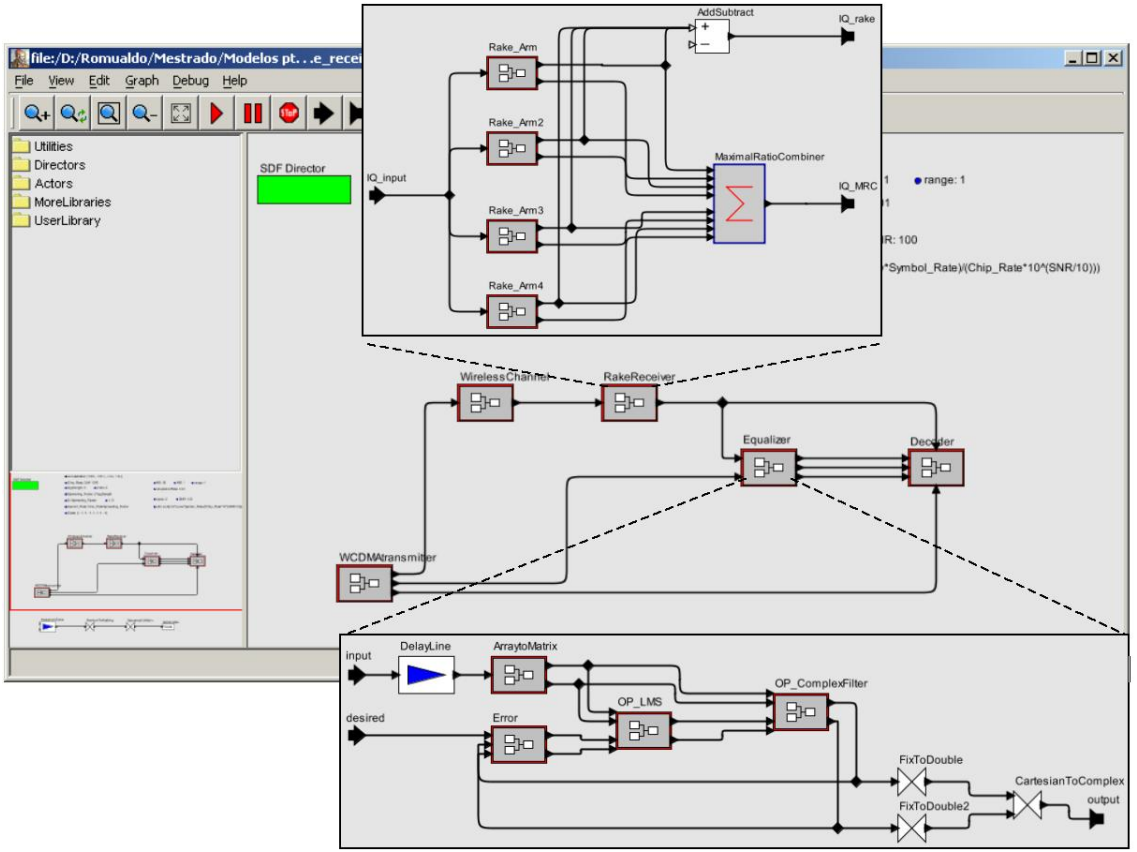


Course Outline

- On-Chip Multiprocessing
  - motivation, review
- Application-specific Multiprocessor Platforms
  - validation, concurrency issues, application-platform mapping
- Platform models
  - evaluation, accuracy
- Application models
  - concurrency
- Joint execution of application and platform models
  - case study
- Background theory and tool support
  - actor orientation, Ptolemy II, custom made extensions
  - demo

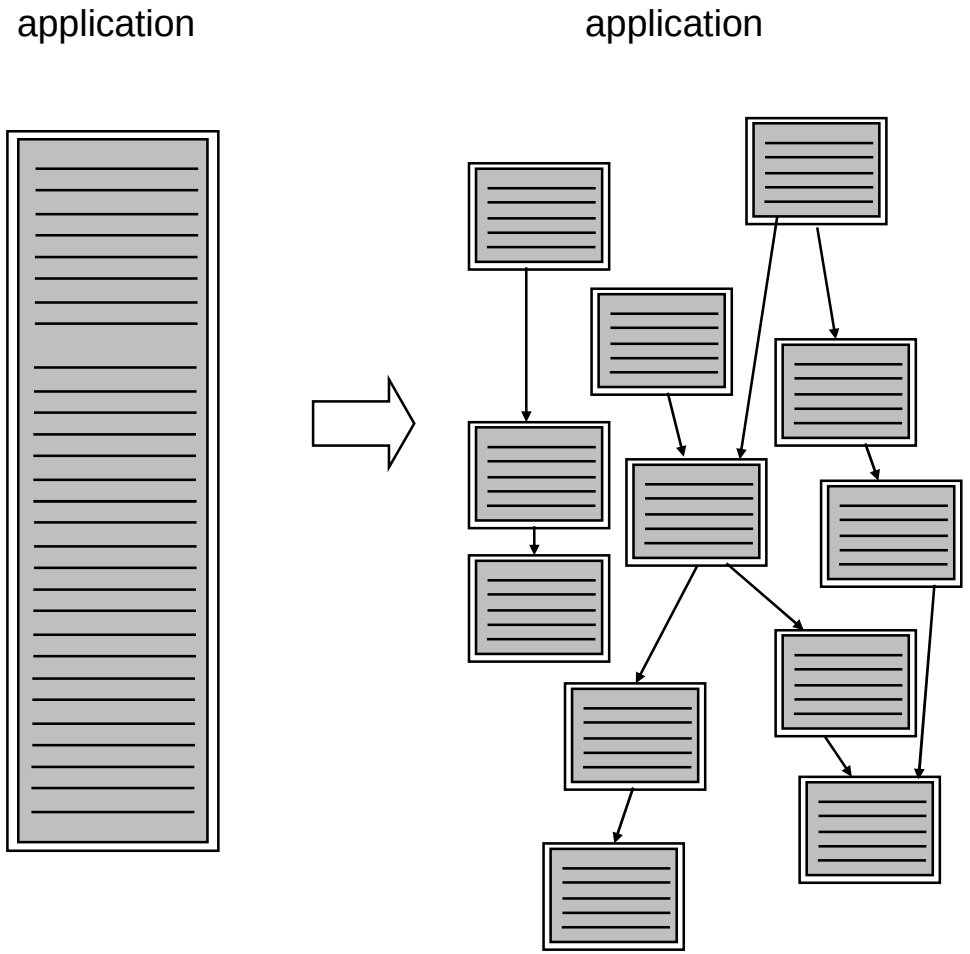
Application models

- Main focus on
  - concurrency model
  - inter-process communication behaviour
- Executable



Application models

- Many concurrent programming models available
  - threads
  - concurrent processes with message-passing
  - actors
  - streams/dataflow
  - CSP
  - timed automata
  - <add your favorite here>

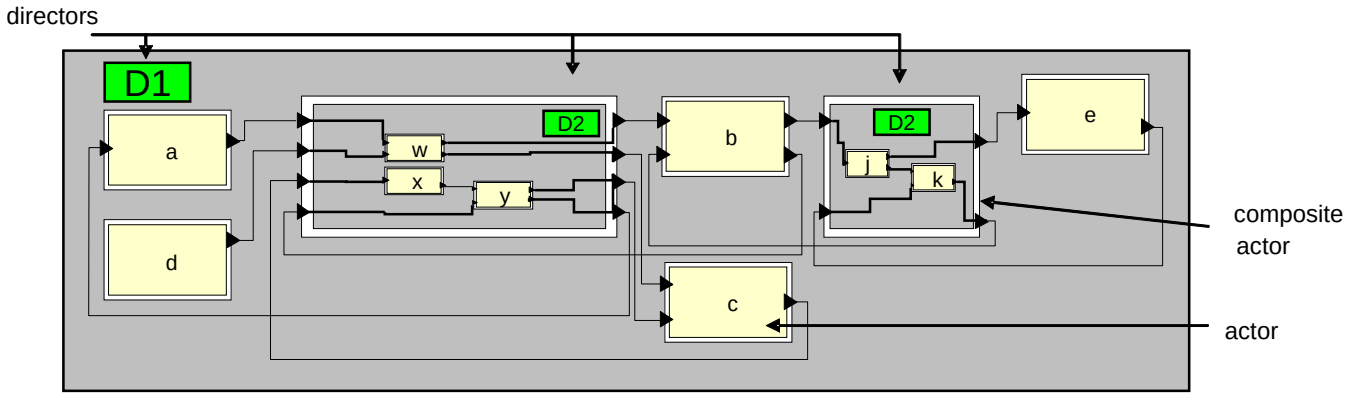


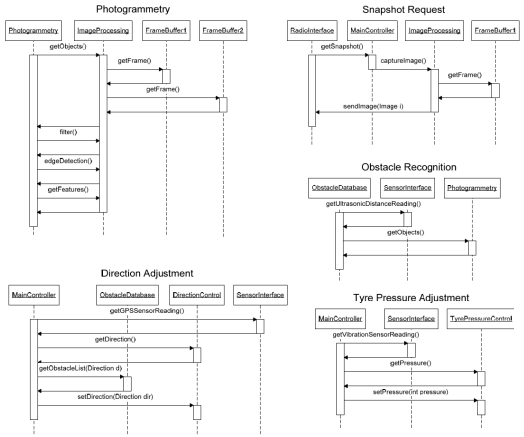
Application models

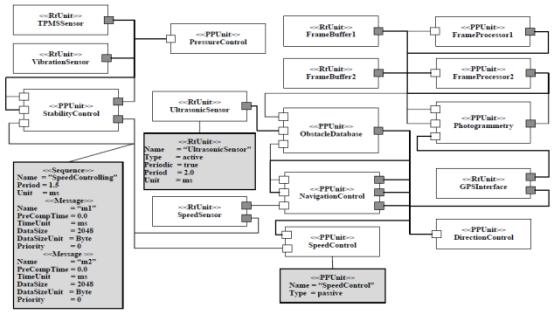
- The choice of which concurrent programming model should be adopted depends on
  - application domain
  - familiarity by the designer/developer
  - availability of stable flows and tools
  - predictability, and ability to match underlying HW/SW

model	tools
threads	**libraries in Java and C++**
message passing	**OpenMP**
actors	**Simulink, Ptolemy**
streams / dataflow	**StreamIt, CUDA**
timed automata	**UPAAL**

Application models

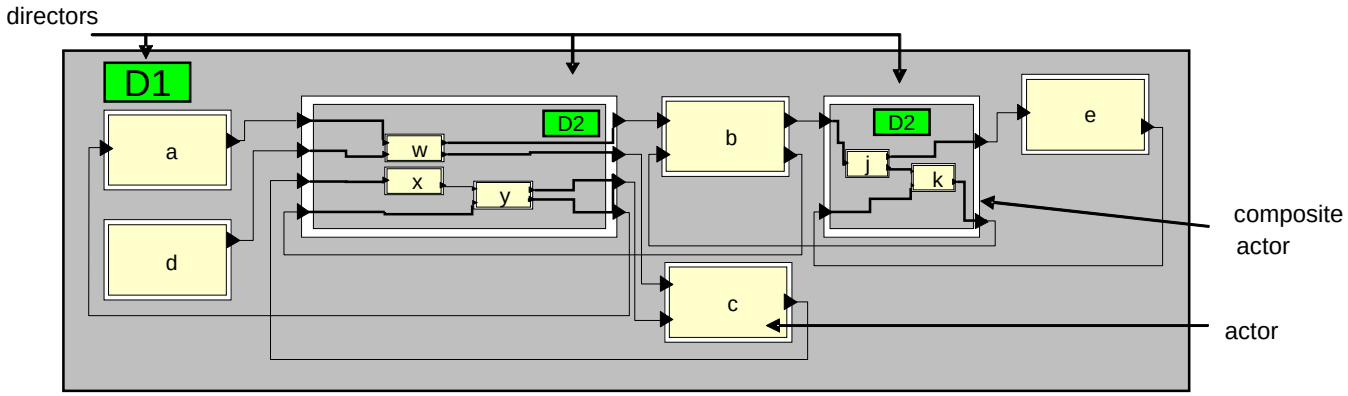







```
public class QAMMod extends TypedAtomicActor {

    /** Creates a new instance of QAM */
    public QAMMod(CompositeEntity container, String name)
            throws NameDuplicationException, IllegalActionException {

        super(container, name);
        // input Port
        input = new SDFIOPort(this, "input", true, false);
        //output Ports
        reqam = new TypedIOPort(this, "reqam", false, true);
        imqam = new TypedIOPort(this, "imqam", false, true);
        // Parameter
        order = new Parameter(this, "order", new IntToken(4));
        order.setTypeEquals(BaseType.INT);
        //debug
//      ai = new TypedIOPort(this, "ai", false, true);
//      aq = new TypedIOPort(this, "aq", false, true);

        _attachText("_iconDescription", "<svg>\n"
                + "<rect x=\"-30\" y=\"-20\" "
                + "width=\"60\" height=\"40\" "
                + "style=\"fill:white\"/>\n"
                + "<polygon points=\"-20,10 20,-10 20,10 -20,-10\" "
                + "style=\"fill:blue\"/>\n"
                + "</svg>\n");
    }
```

Application models



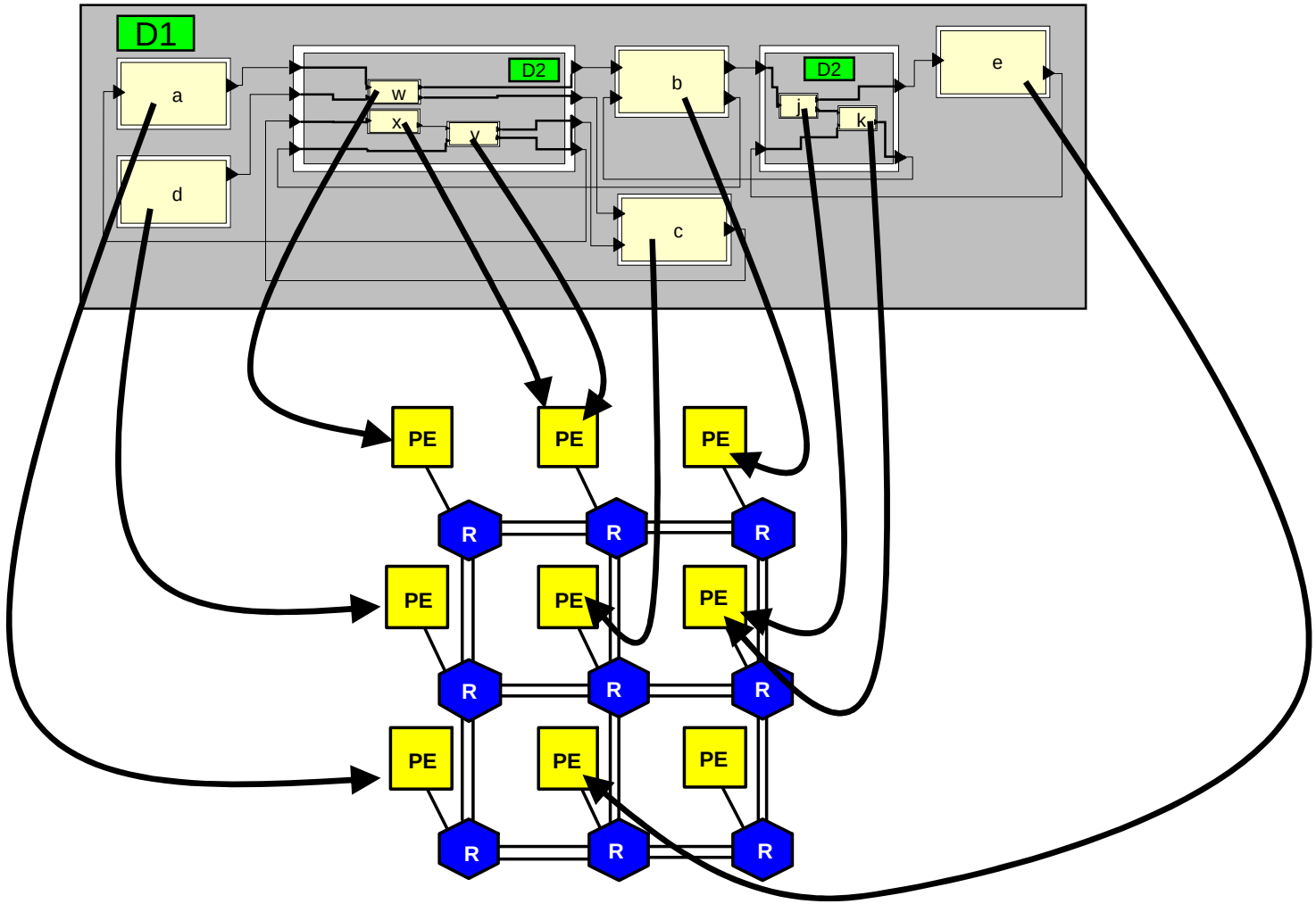
- captures the concurrent behaviour of the application
- completely independent from the platform model
- for a given mapping, it can be jointly executed with the platform model
  - joint execution allows back-annotation of performance and power consumption figures

Course Outline

- On-Chip Multiprocessing
  - motivation, review
- Application-specific Multiprocessor Platforms
  - validation, concurrency issues, application-platform mapping
- Platform models
  - evaluation, accuracy
- Application models
  - concurrency
- Joint execution of application and platform models
  - case study
- Background theory and tool support
  - actor orientation, Ptolemy II, custom made extensions
  - demo

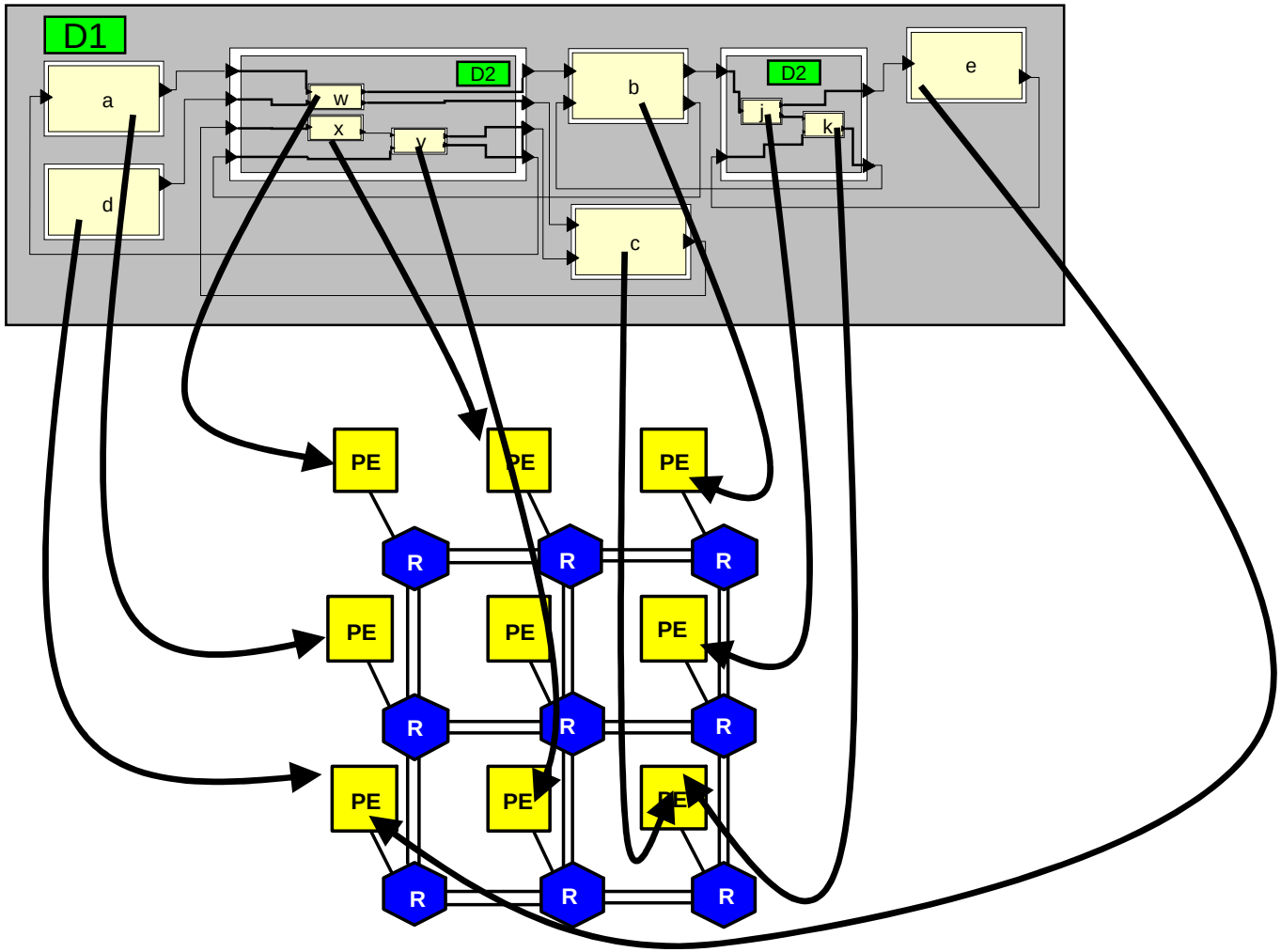
Joint execution of application and platform models

Mapping 1



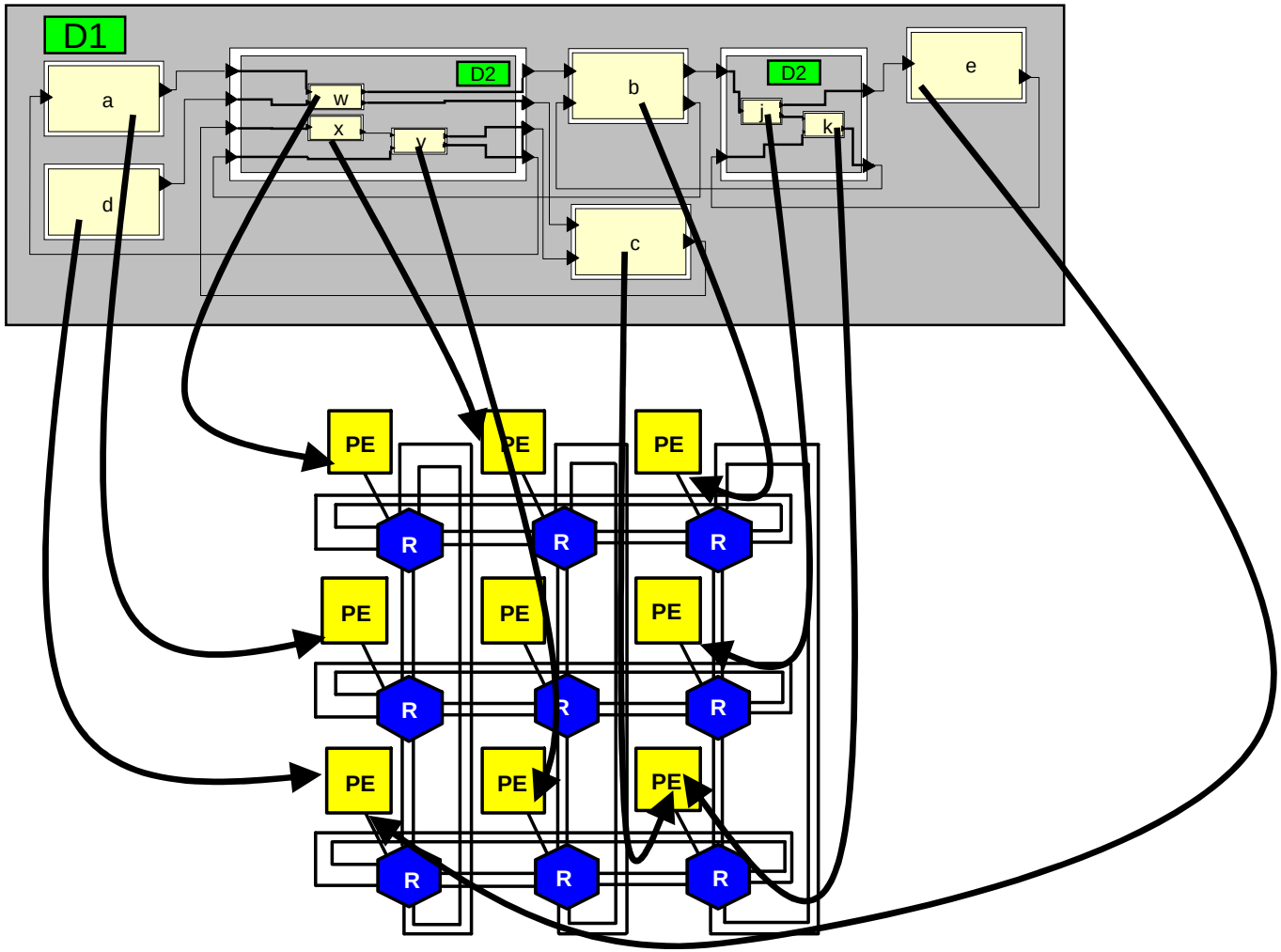
Joint execution of application and platform models

Mapping 2

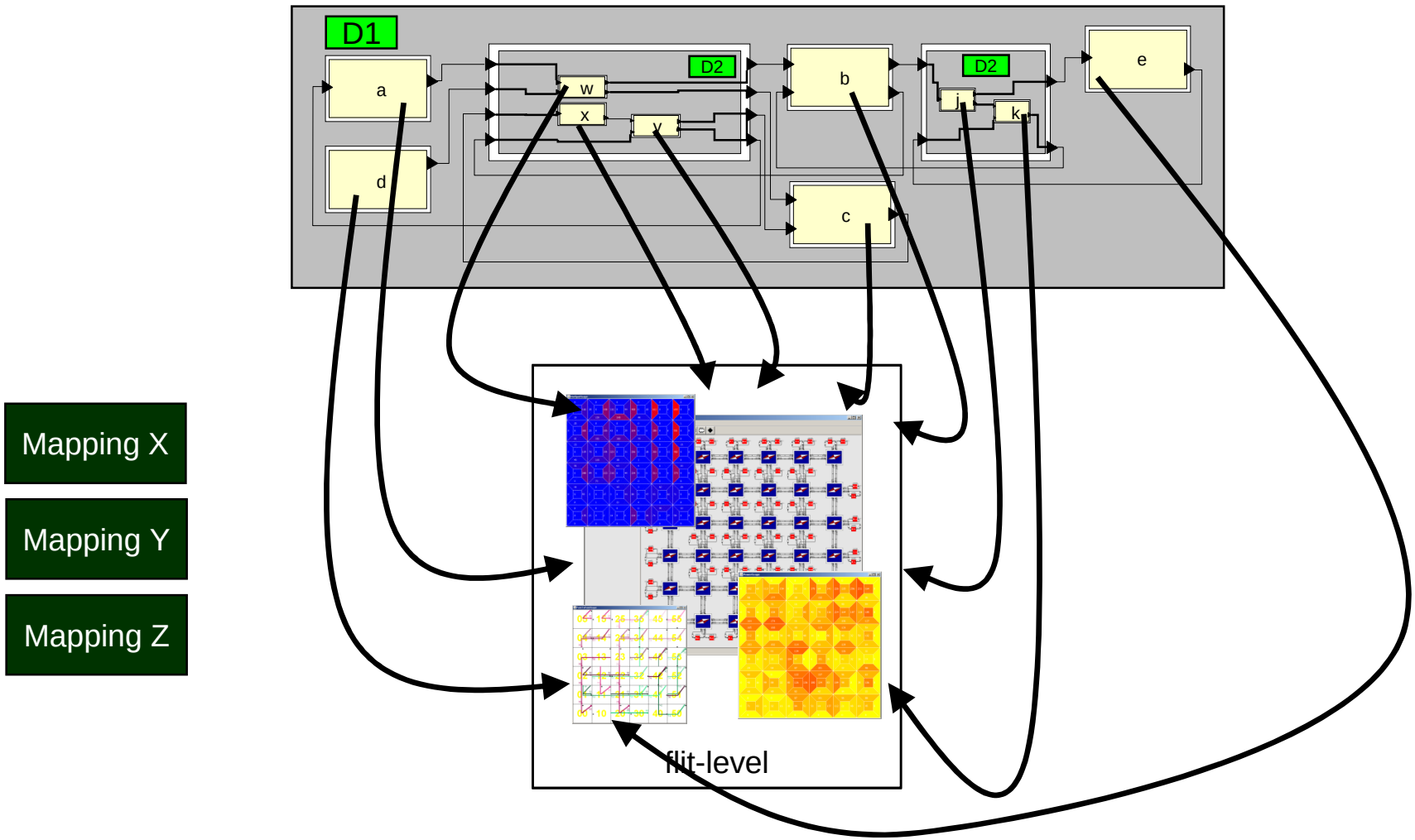


Joint execution of application and platform models

Mapping 2

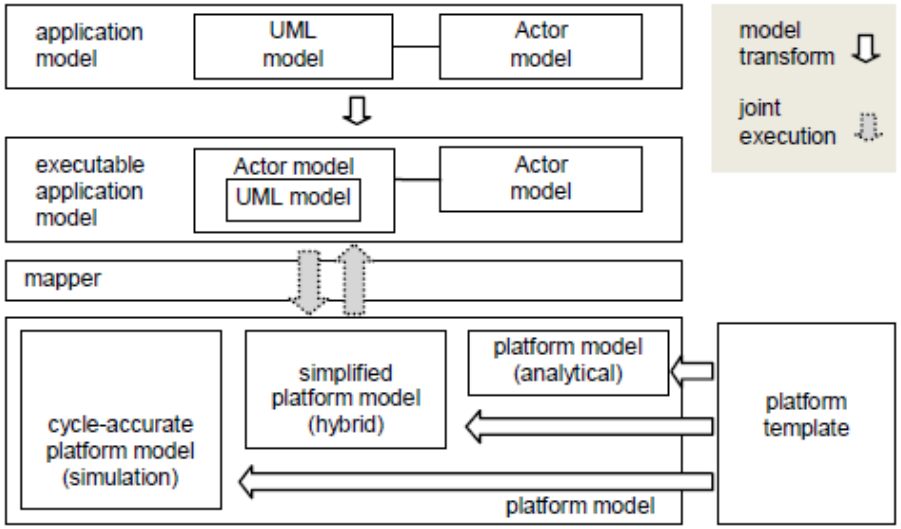


Joint execution of application and platform models



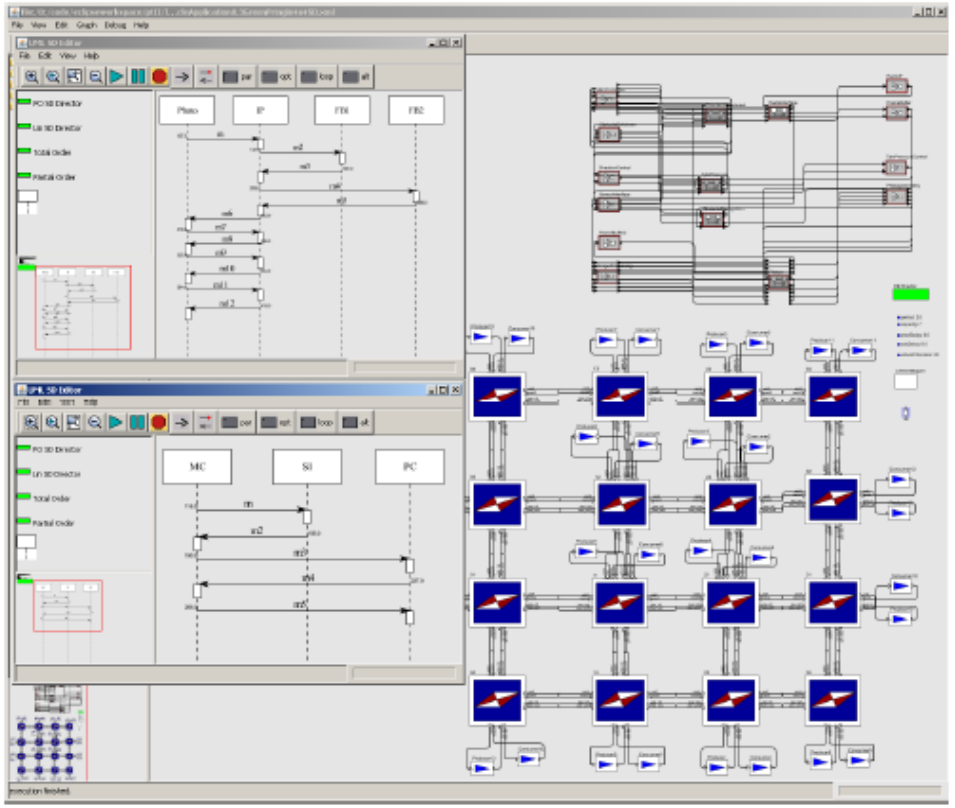
Case study

- Application model
  - autonomous vehicle
  - model based on actors and message sequence charts
- Platform models
  - analytical
  - hybrid
  - cycle-accurate

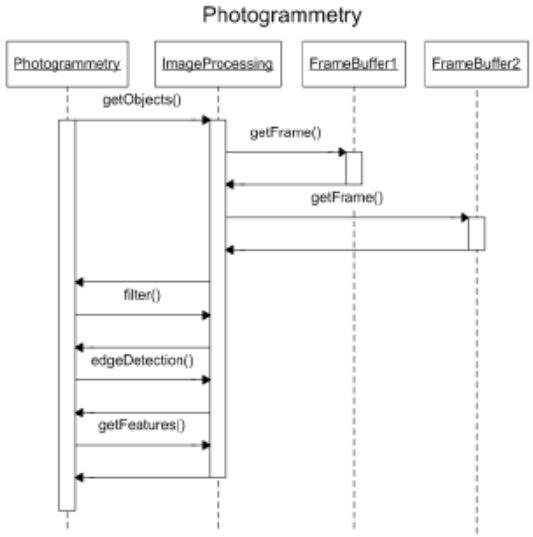


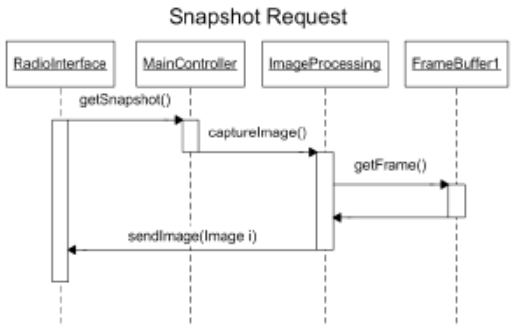
Case study

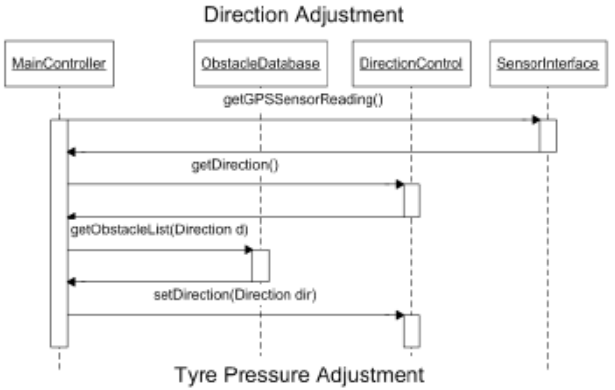
- Application model
  - autonomous vehicle
  - model based on actors and message sequence charts
- Platform models
  - analytical
  - hybrid
  - cycle-accurate

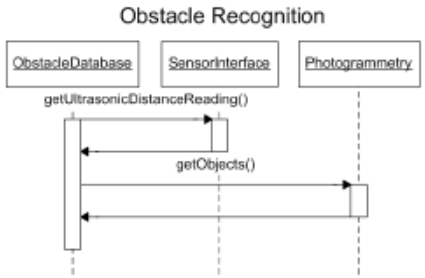


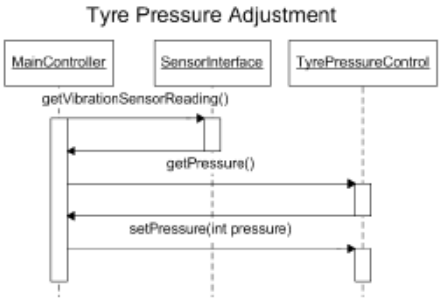
Case study


Photogrammetry
Photogrammetry
ImageProcessing
FrameBuffer1
FrameBuffer2
getObjects()
getFrame()
getFrame()
filter()
edgeDetection()
getFeatures()

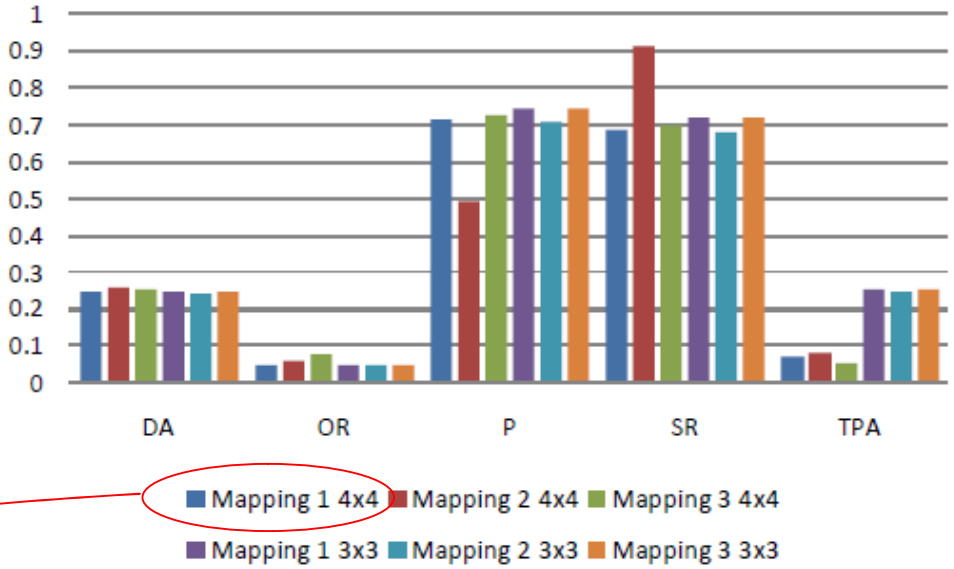

Snapshot Request
RadioInterface
MainController
ImageProcessing
FrameBuffer1
getSnapshot()
captureImage()
getFrame()
sendImage(Image i)


Direction Adjustment
MainController
ObstacleDatabase
DirectionControl
SensorInterface
getGPSSensorReading()
getDirection()
getObstacleList(Direction d)
setDirection(Direction dir)


Obstacle Recognition
ObstacleDatabase
SensorInterface
Photogrammetry
getUltrasonicDistanceReading()
getObjects()


Tyre Pressure Adjustment
MainController
SensorInterface
TyrePressureControl
getVibrationSensorReading()
getPressure()
setPressure(int pressure)

Case study



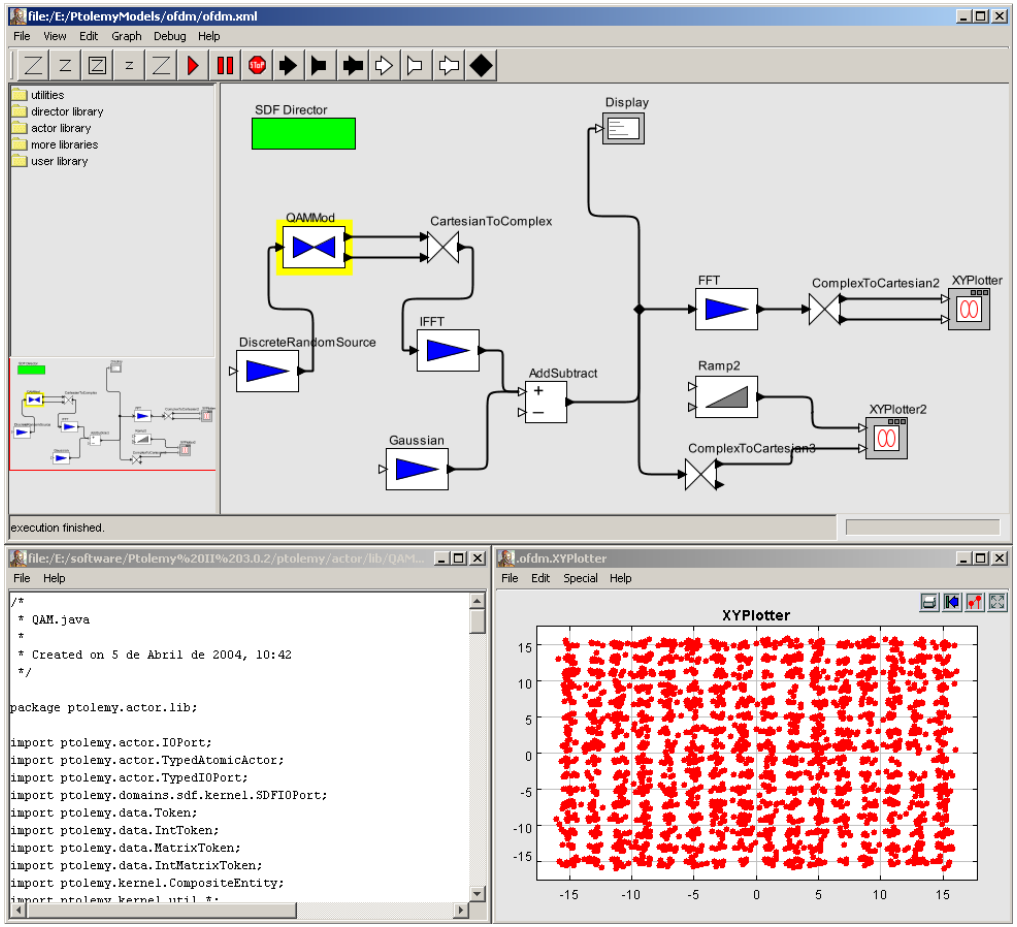
Interaction	Communication Latency (ms)		
	Abstract	*Hybrid*	*Cycle-accurate*
OR	0.04	0.09	0.12
TPA	0.07	0.09	0.14
DA	0.24	0.27	0.39
SR	0.68	0.94	1.35
P	0.71	0.99	1.41

Course Outline

- On-Chip Multiprocessing
  - motivation, review
- Application-specific Multiprocessor Platforms
  - validation, concurrency issues, application-platform mapping
- Platform models
  - evaluation, accuracy
- Application models
  - concurrency
- Joint execution of application and platform models
  - case study
- Background theory and tool support
  - actor orientation, Ptolemy II, custom made extensions
  - demo

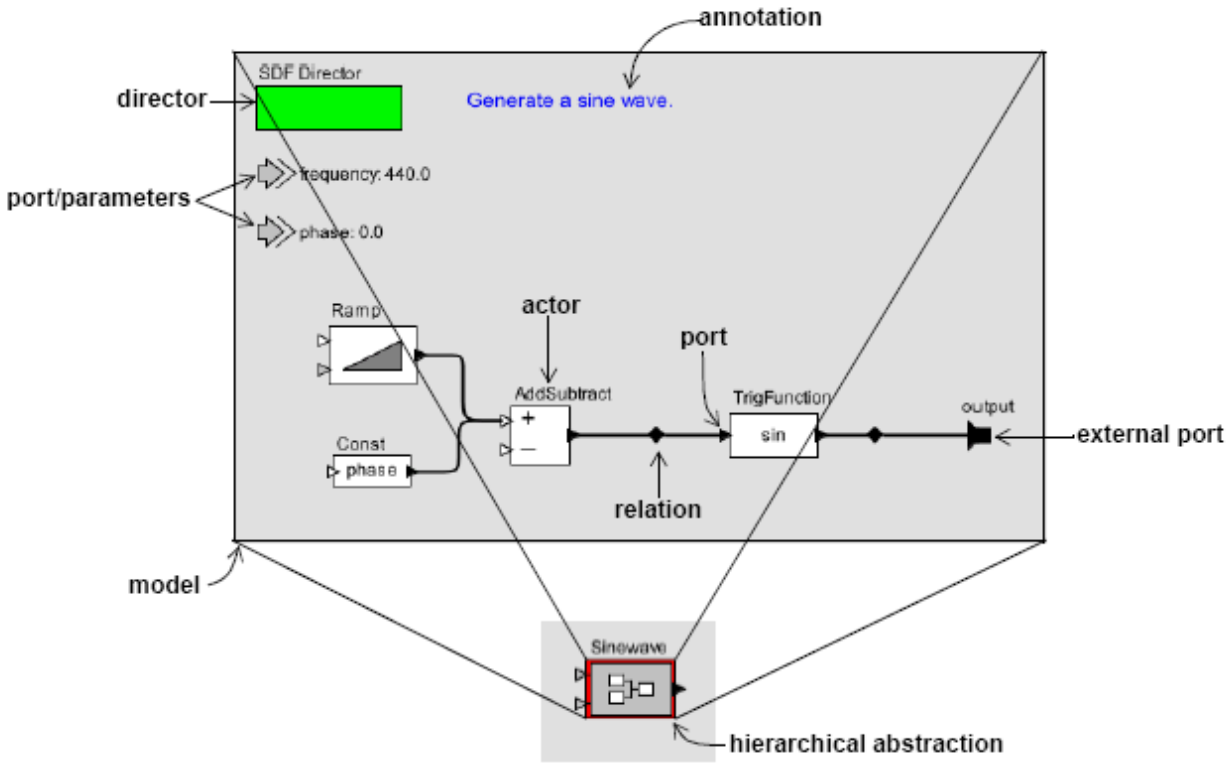
Ptolemy II

- Modeling and simulation framework developed at UC Berkeley
- Focus on modeling concurrent components



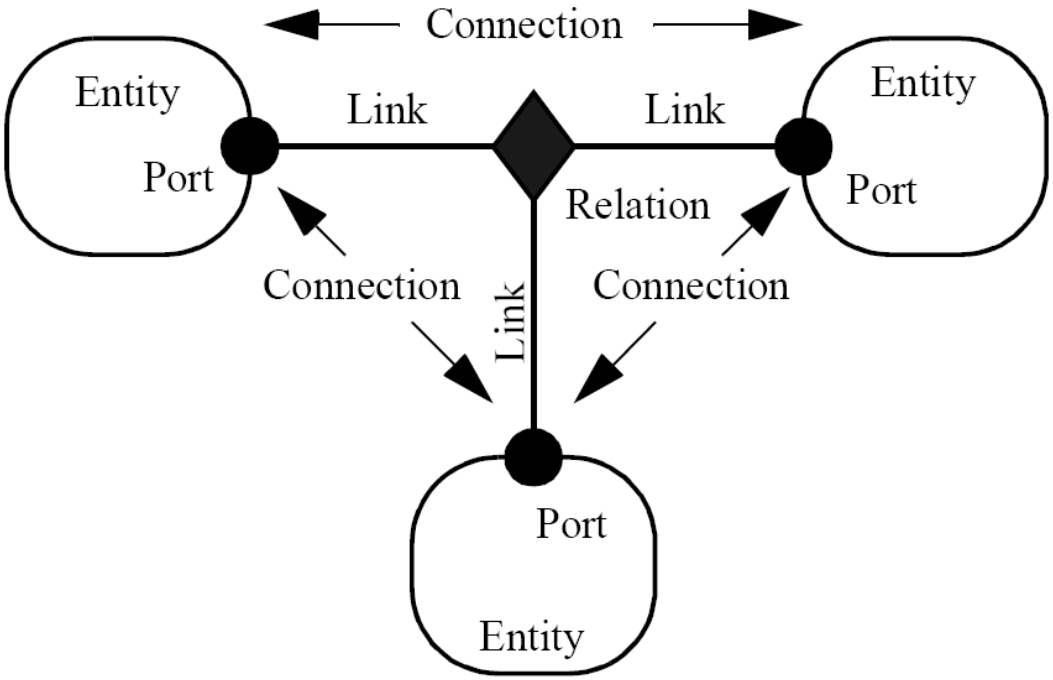
- http://ptolemy.eecs.berkeley.edu

Ptolemy II



source: UC Berkeley

Ptolemy II



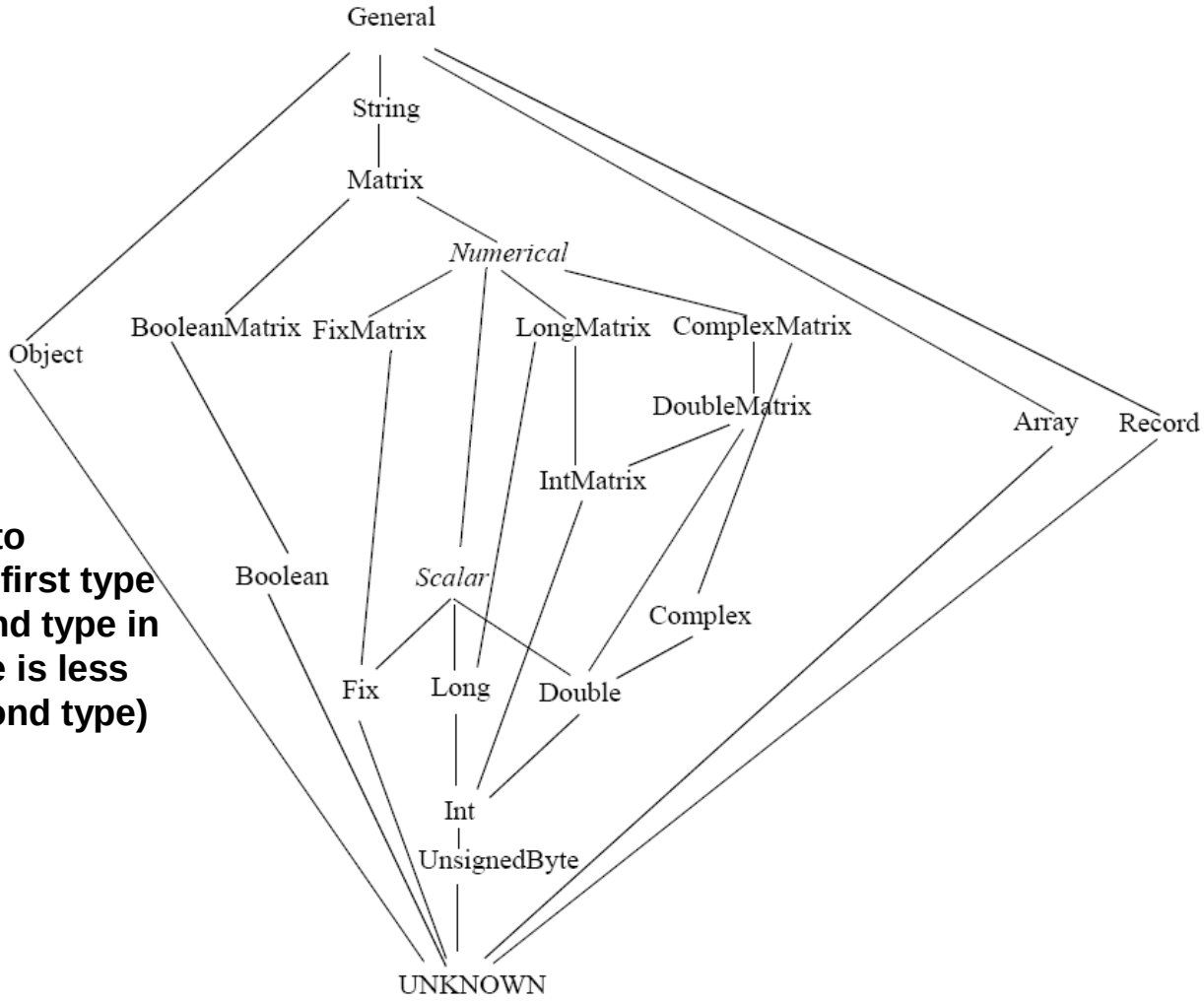
source: UC Berkeley

Ptolemy II

- Data Types

partial ordering of types

a conversion from one type to another is allowed if the first type appears below the second type in the type lattice (first type is less than or equal to the second type)

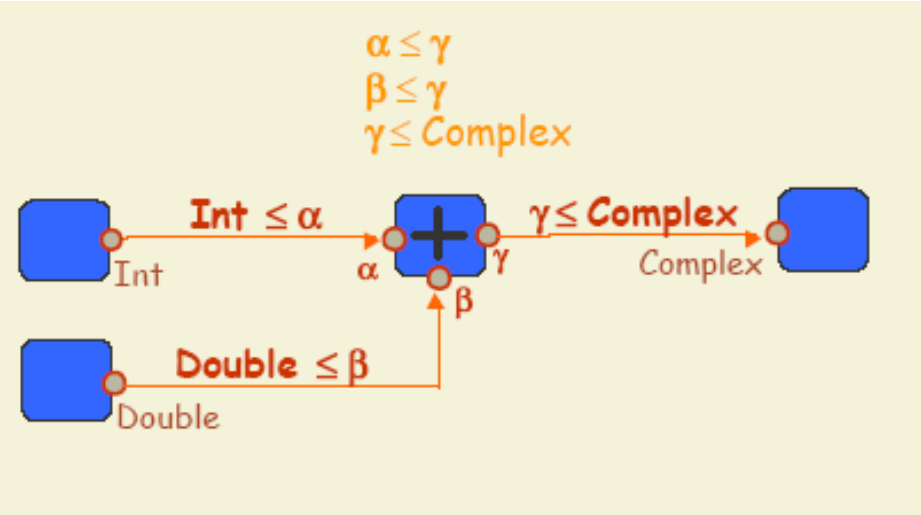


source: UC Berkeley

Ptolemy II

- Data Types

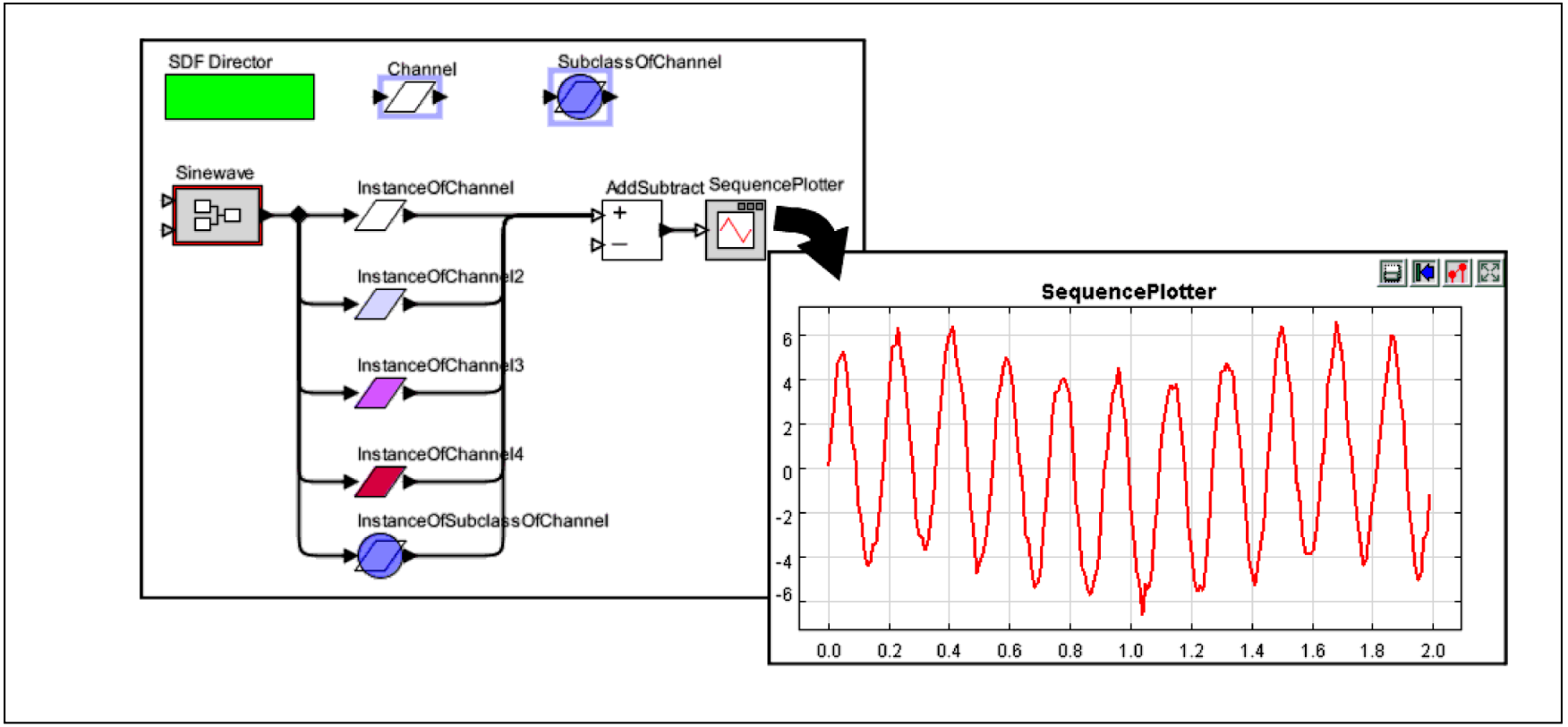
$$outType \leq inType$$



source: UC Berkeley

Ptolemy II

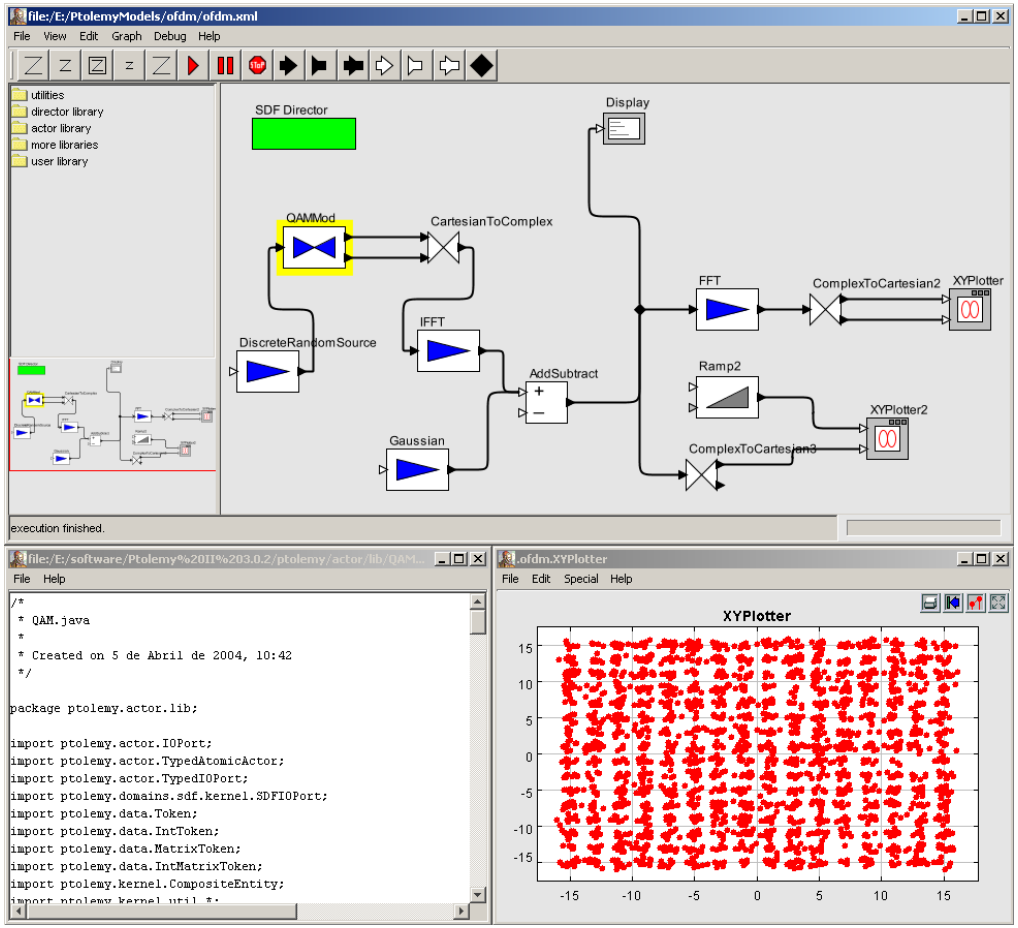
- Functional composition



source: UC Berkeley

Concurrency in Ptolemy II

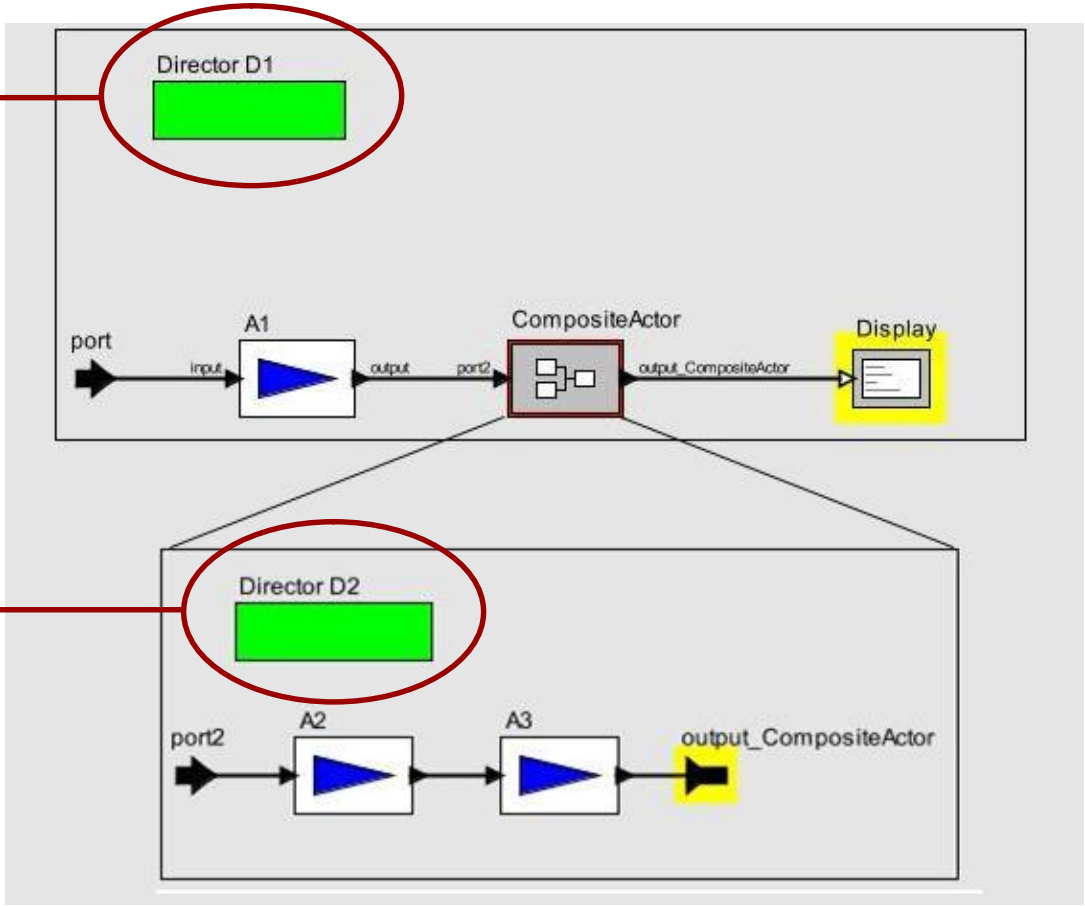
- Concurrency modeling is a primary goal of Ptolemy II
- Based on the concept of actor-orientation
  - explicit definition of the model of time and concurrency
  - support heterogeneous models
  - support domain polymorphism



Concurrency in Ptolemy II

execution semantic of the top level model

execution semantic of the composite model



Actors

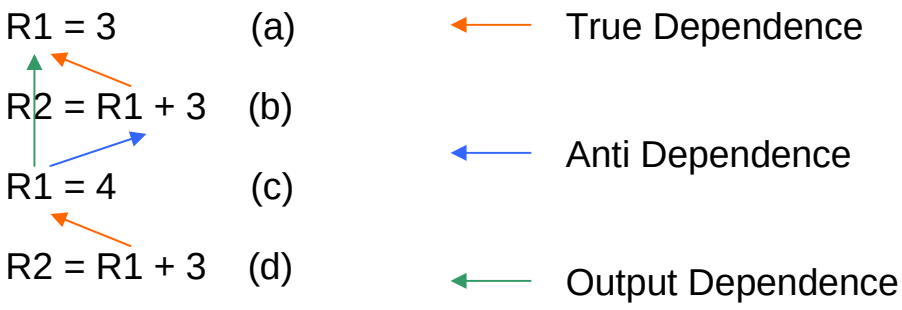
- According to Atkinson, Hewitt (MIT, 1977)
  - Conceptually an actor is an object which has both procedural and data aspects
  - A process is a totally ordered sequence of computational events where each event consists of sending a message to an actor
  - The actor model of computation has encouraged the development of a paradigm based on a society of experts communicating by passing messages

Actors

- According to Gul Agha (MIT, 1986)
  - Actors are computational agents which may function in parallel (have their own thread of control) and communicate through asynchronous message passing
  - The idea behind actor languages is to provide the syntactic constructs that can free the programmer from having to worry about the details of a concurrent execution

Actors

- How to free the programmer from having to worry about the details of a concurrent execution?
- One way is to remove unnecessary data dependencies created by the assignment operation (von Neumann bottleneck)



Actors

- Functional programming allows for the possibility of concurrent evaluation of expressions in a program

- (sum (4 5))
- 9

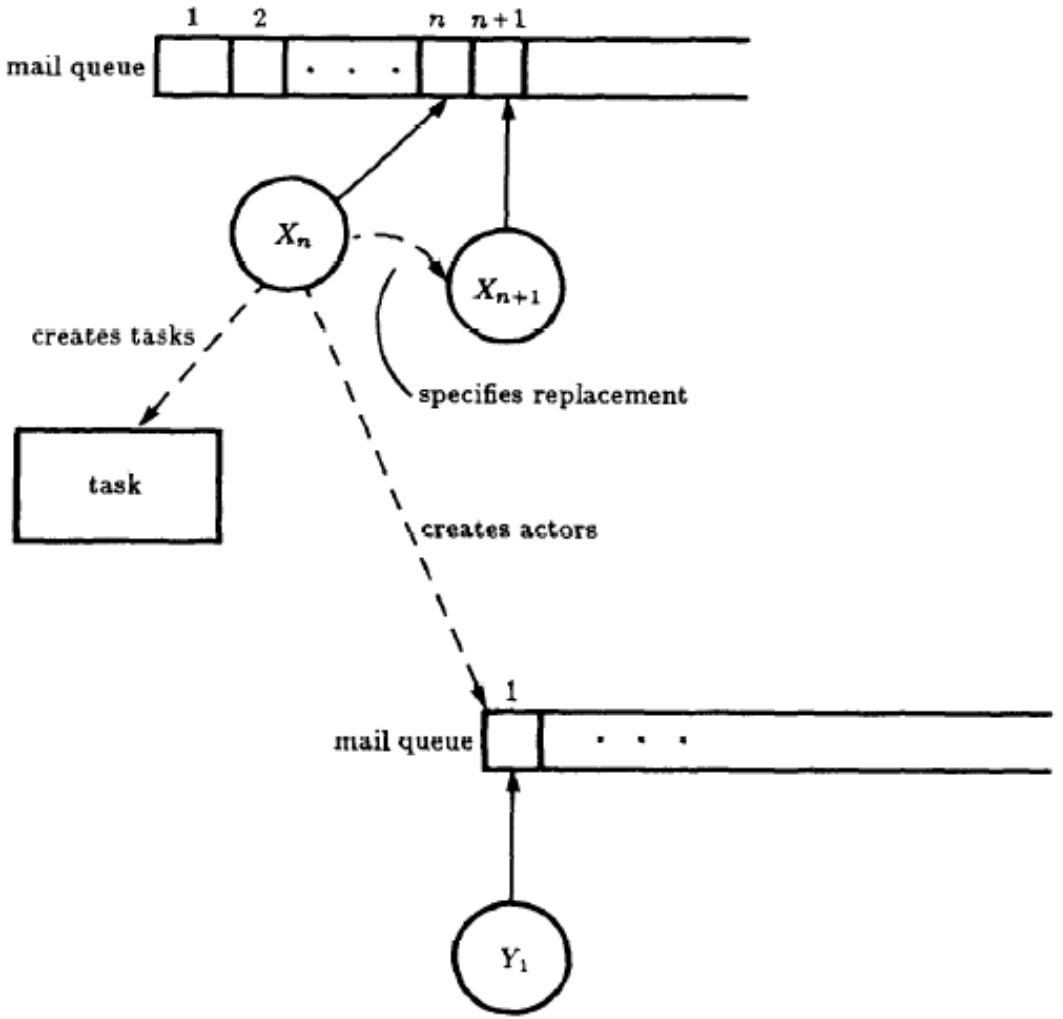
Actors

- Functional programming allows for the possibility of concurrent evaluation of expressions in a program

```
(sum (      (mult (
                (sum (4 5)) (sum (4 5))
            )
            (sum (4 5))
        )
)
```

Actors

- Functional programming is unable to address the problem of history-sensitive behaviors (functions don‘t keep state)
- Actors avoid assignment operations but allow the modeling of history-sensitive behaviors
  - Gul Agha proposed the concept of replacement, which states that an actor must specify/create the actor which will handle the next message/communication



source: Gul Agha 1986

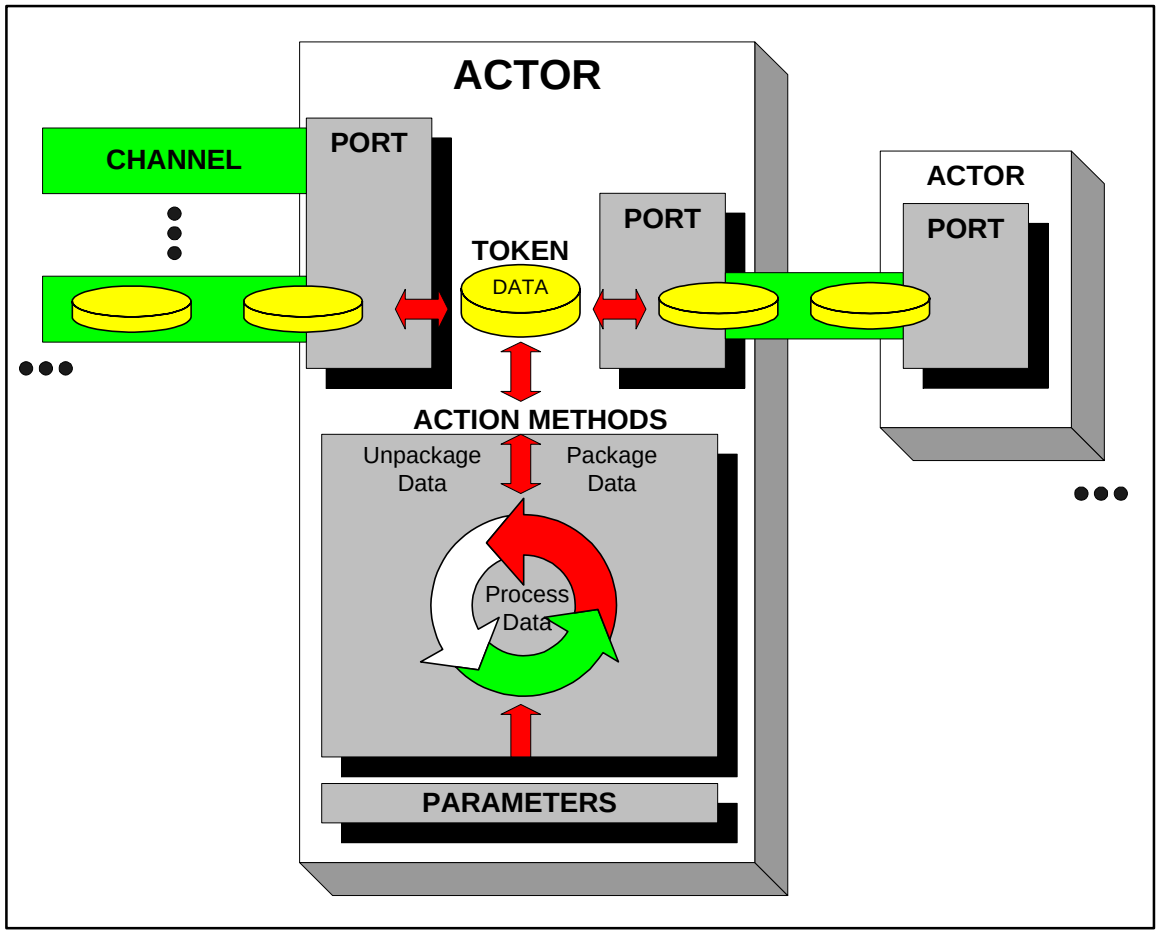
Actors

- According to Lee (Berkeley, 2004)
  - In actor-oriented design, components are concurrent objects that communicate via messaging, as opposed to abstract data structures that interact via procedure calls
  - Actors are conceptually concurrent, but unlike Agha's actors, they need not have their own thread of control
  - Although communication is still through some form of message passing, it need not be strictly asynchronous

Actor-orientation

- Actors execute and communicate with other actors in a model
- A have a well defined component interface, which abstracts the internal state and behavior of an actor, and restricts how an actor interacts with its environment
- The interface includes ports that represent points of communication for an actor, and parameters that are used to configure the operation of an actor

Actor-orientation

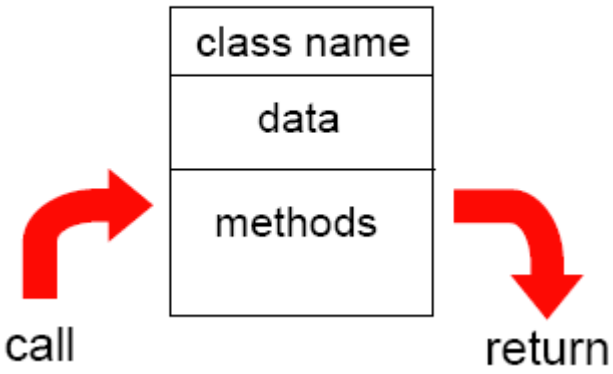


source: Lee 2004

Actor-orientation

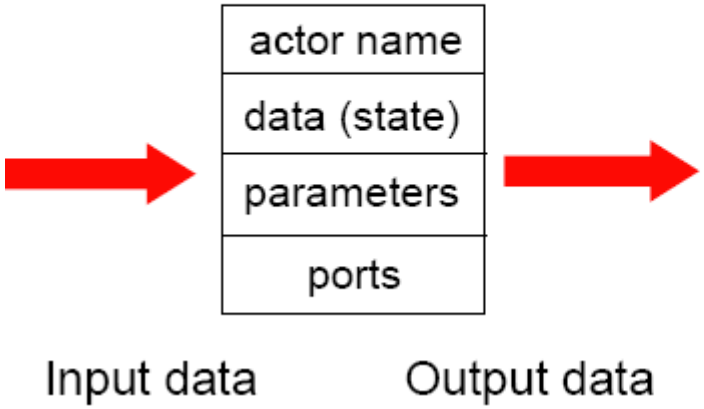
- Actors communicate through channels that pass data from one port to another according to some messaging scheme
- In object-oriented design, components interact primarily by transferring control through method calls, while in actor-oriented design they interact by sending messages through channels
- Actors interact only with the channels that they are connected to and not directly with other actors

Method-call based:



What flows through an object is sequential control

Actor oriented:



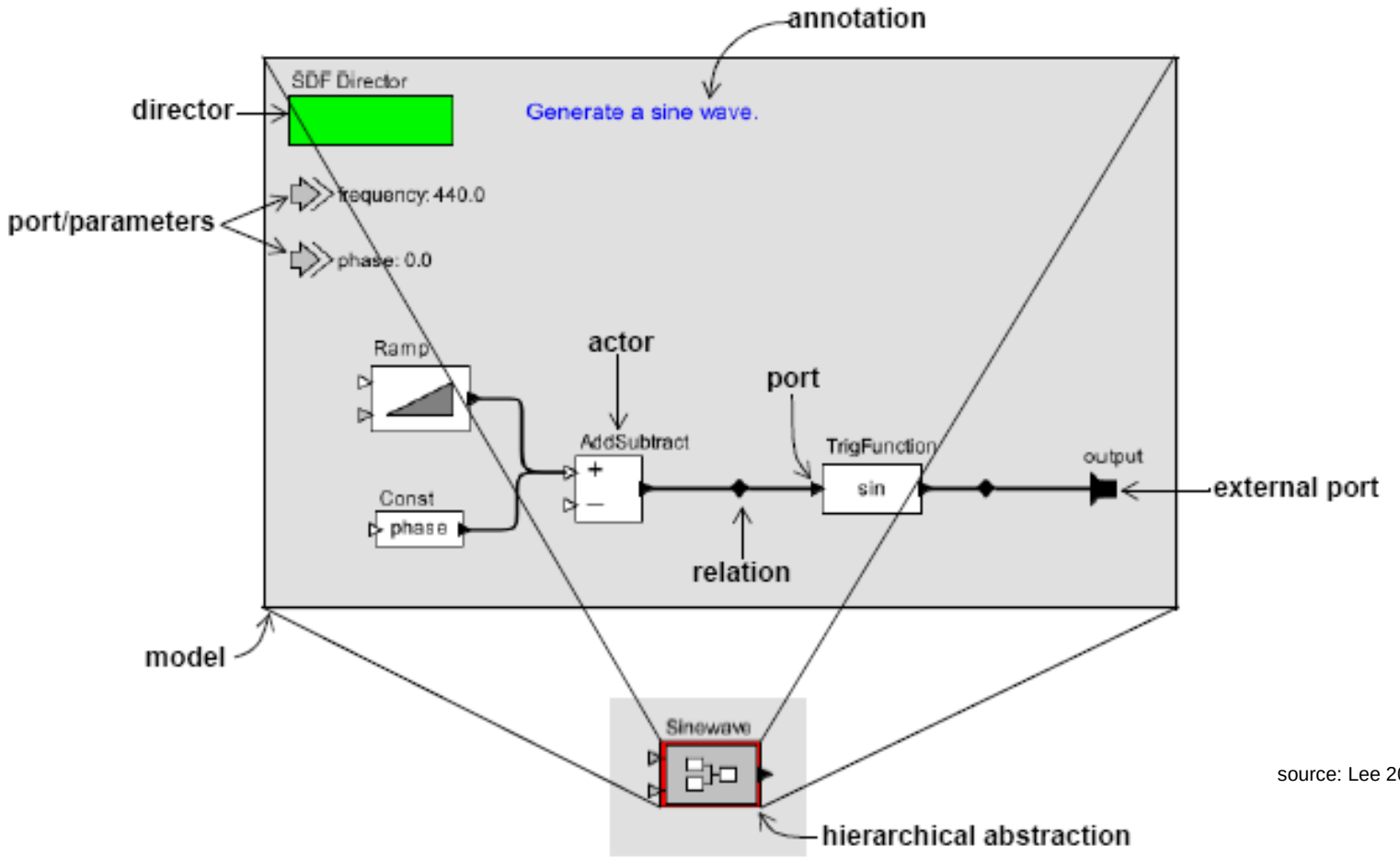
What flows through an object is streams of data

source: Lee 2004

Actor-orientation

- The syntactic structure of an actor-oriented design model says little about the semantics
- The semantics is determined by a model of computation (MoC), which define the operational rules for executing a model
- Such rules determine when actors perform internal computation, update their internal state, and perform external communication
- The model of computation also defines the nature of communication between components
- According to Lee, the MoC is expressed as a Director in an actor-oriented model

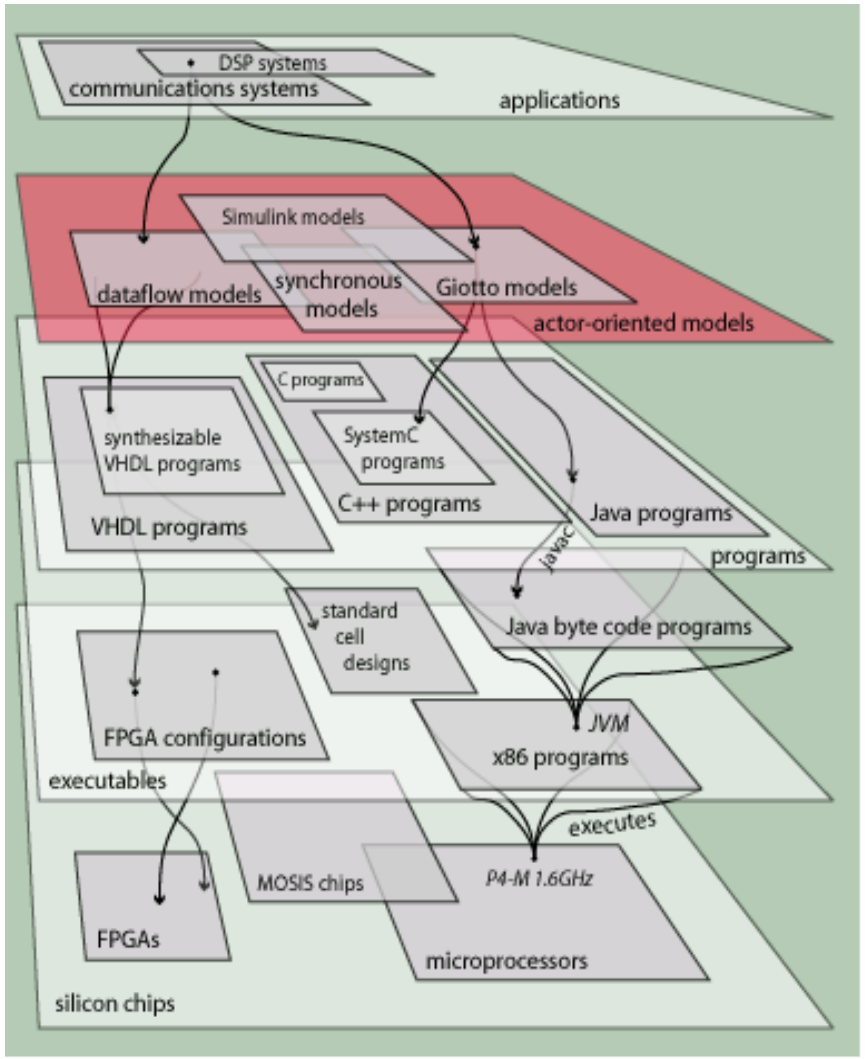
Actor-orientation



source: Lee 2005

Actor-orientation

- MoCs formalize a platform of well defined execution semantics that can help mapping applications onto implementation-related platforms
- Heterogeneous models can be created through the hierarchical composition of different MoCs



Source: E. A. Lee

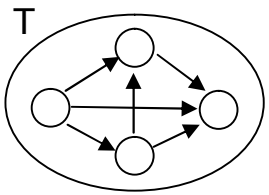
Concurrent Models of Computation

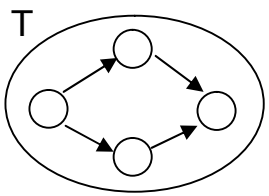
- Rules that define how actors behave and communicate concurrently in a given model
- According to the rule definitions, models can have specific properties, such as being statically schedulable or time safe
- Examples: Discrete Events, Communicating Sequential Processes, Continuous Time, Process Networks, Synchronous Dataflow, Giotto, etc.

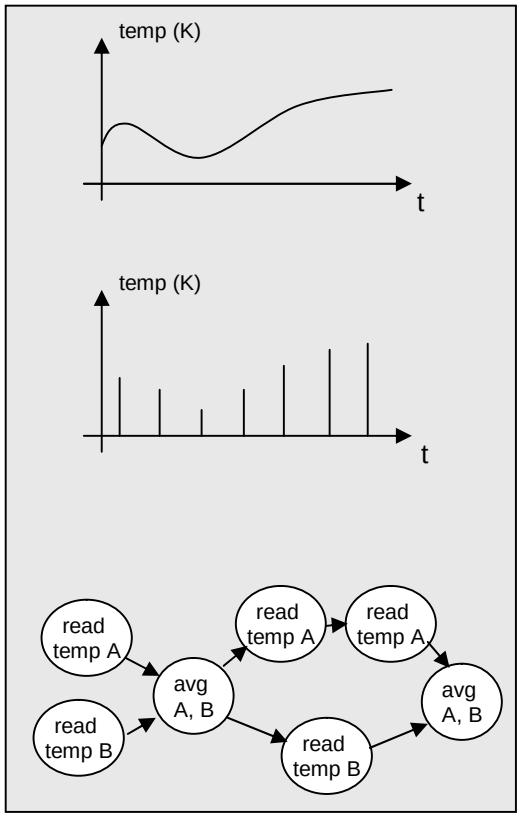
Concurrent Models of Computation

- Models can have different representations of time

  - absolute time
    - T is a totally ordered closed connected set
    - T = $\Re$
  - discrete time
    - T is a totally ordered discrete set
  - untimed
    - precedences
    - T is a partially ordered discrete set







Concurrent Models of Computation

- Models can have different representations of time
  - absolute time
    - continuous time (CT)
  - discrete time
    - discrete events (DE)
  - untimed
    - dataflow models
      - Kahn process networks (KPN)
      - synchronous dataflow (SDF)

Discrete Event (DE)

- In the DE MoC, actors communicate by sending events, which comprehends a token and a tag
  - token is a data value
  - tag includes a timestamp and a microstep (used to order events with the same timestamp)
- Events are processed chronologically according to their timestamps
- Actors whose available input events are the oldest (having the earliest time stamp of all pending events) are activated

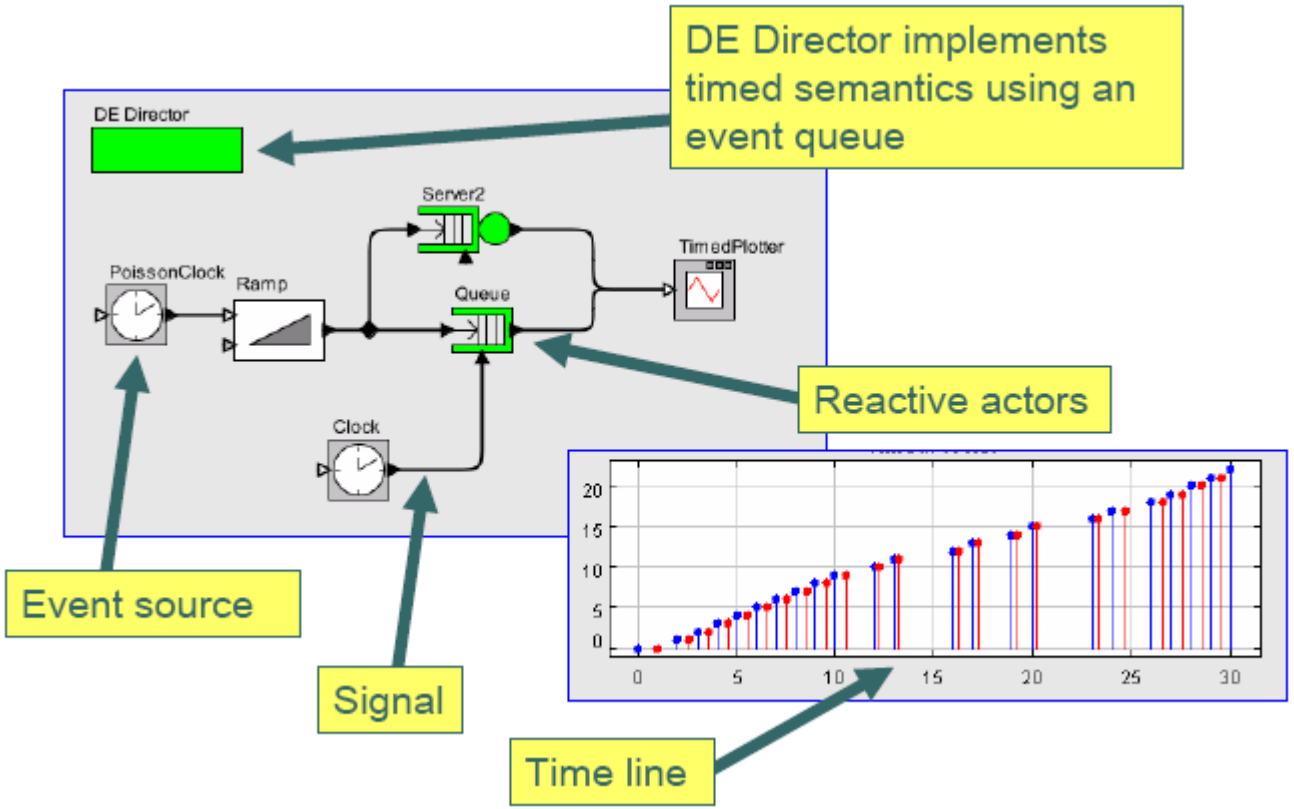
Discrete Event (DE)

- Model time is global: all actors share the same timestamp and microstep
- When actors send a token through an output port, this token is encapsulated as an event and stored in a global event queue
- Unlike specified otherwise by the actor, the event is tagged with the current model time and microstep (actor processes tokens without delay)

Discrete Event (DE)

- Actors can be activated at arbitrary times by sending *pure events* to themselves with a future timestamp (no need for ports)
- In the global event queue, events are sorted based on their tags, including time stamps and microsteps
- An event is removed from the global event queue when the model time reaches its time stamp, and if it has a data token, then that token is put into the destination input port

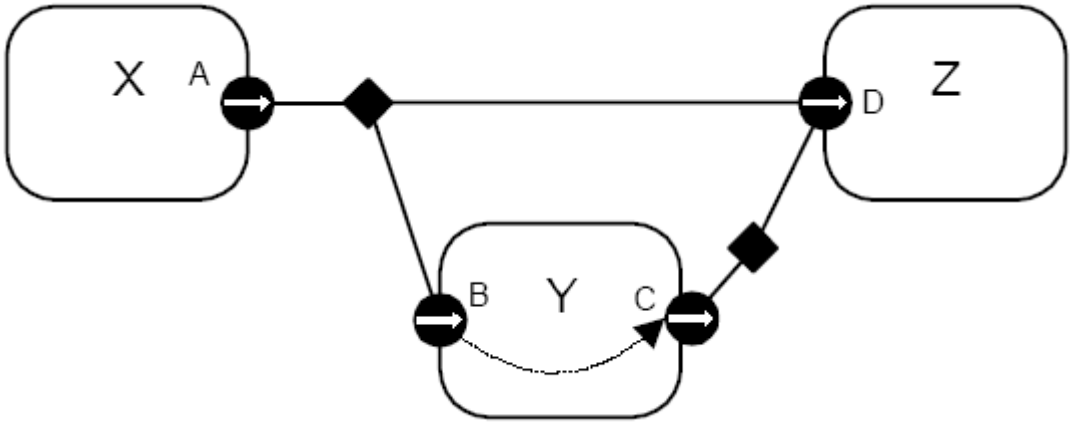
Discrete Event (DE)



source: Lee 2004

Discrete Event (DE)

- Simultaneous events
  - the execution of events with the same timestamp must be ordered (in a deterministic way)
  - achieve a dataflow behavior for events with the same timestamp



source: Lee 2005

Discrete Event (DE)

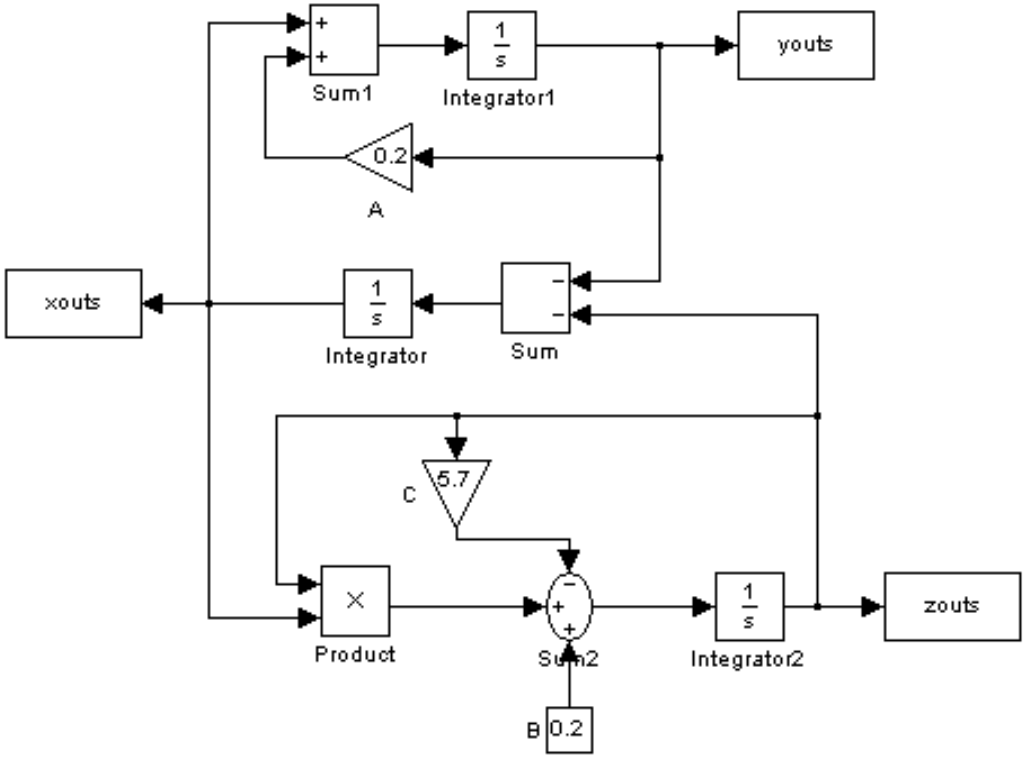
- Typical uses:
  - digital systems
  - queue systems
  - networks
  - transportation systems
  - commerce
- Similar to:
  - VHDL, Verilog, SES Workbench

Continuous Time (CT)

- Actors represent components that interact via continuous time signals
- They typically specify algebraic or differential relations between inputs and outputs, and can be modeled by ordinary differential equations (ODEs)
- Such models are „solved“ rather than simulated

Continuous Time (CT)

$$\frac{dx}{dt} = -(y+z)$$
$$\frac{dy}{dt} = x + Ay$$
$$\frac{dz}{dt} = B + z(x-C)$$

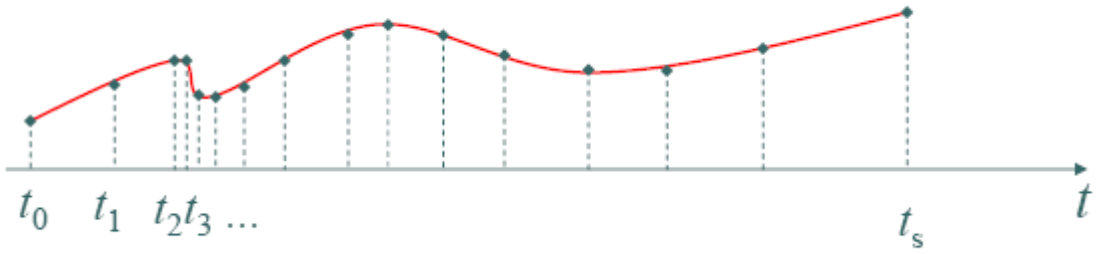


Continuous Time (CT)

- Solution is a continuous function of time (waveform)
- The precise solution of a set of ODEs is usually impossible to be found using digital computers
- Numerical solutions are approximations of the precise solution

Continuous Time (CT)

- Numerical solution approximate the state trajectory of a differential equation by estimating its value at discrete points
- The choice of the points depends on the function
- Using numerical solvers, the solution is a discrete-event signal



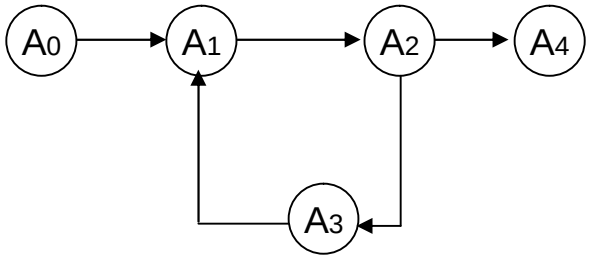
source: Lee 2004

Continuous Time (CT)

- Typical uses:
  - analog circuits
  - plant dynamics in control systems
  - mechanical systems
  - heat flows
  - Communication channels
- Similar to:
  - Spice, Agilent ADS, Simulink

Dataflow Model of Computation

- Dataflow models represent a given system
  - as a directed graph
  - where each node performs computation (actors)
  - and edges represent channels through which nodes can exchange data (tokens)



In Von Neumann imperative style, program counter rules the execution

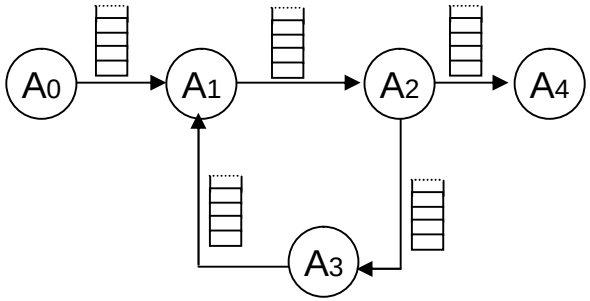
Dataflow models don‘t define the sequence on which actors are executed (fired)

Actors can fire when inputs are available (actors without inputs can fire anytime)

Exposes potential concurrency: data movement rules the execution

Dataflow Model of Computation

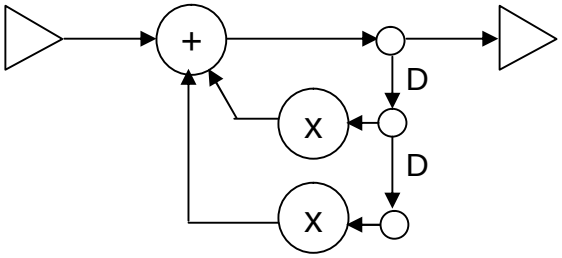
- Actors execute concurrently and communicate by exchanging tokens through FIFO-buffered channels
  - computation performed by a given actor can be described by imperative language

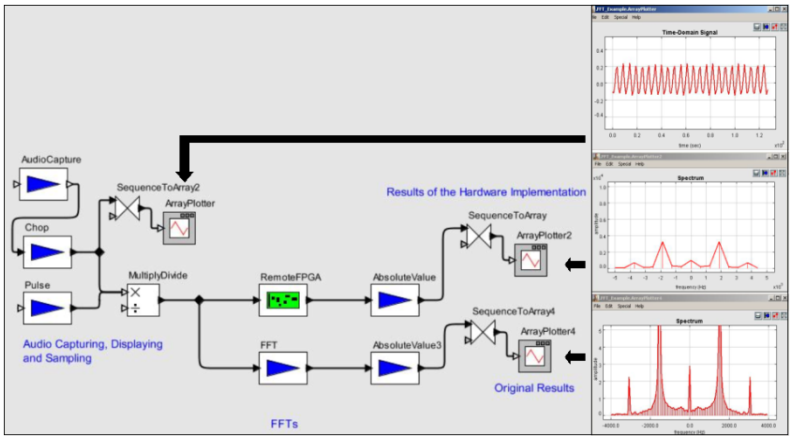


- Buffers usually treated as unbounded for flexibility
  - buffer sizes on final implementation are a design decision
- Read operations are destructive: once read, the token is removed from the buffer

Dataflow Model of Computation

- Applications of dataflow models
  - signal processing systems
  - systems dealing with continuous streams of data





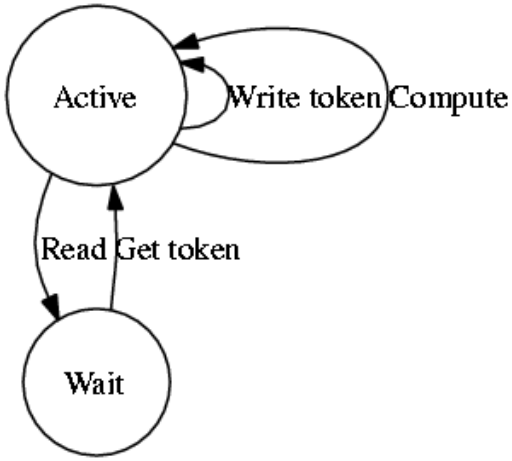
- Easier to extract parallelism
- Buffered channels are used in such systems anyway

Kahn Process Networks (KPN)

- Proposed by Gilles Kahn in 1974
- Processes (actors) communicate asynchronously through unbounded buffers
  - non-blocking write
  - blocking read
- A process that reads from an empty channel will stall and can only continue when the channel contains sufficient tokens
- Processes are not allowed to test an input channel for existence of tokens without consuming them

Kahn Process Networks (KPN)

- At any given point, a process is either doing some computation (active) or it is blocked waiting for data (read blocked) on exactly one of its input channels
  - it cannot wait for data from more than one channel simultaneously

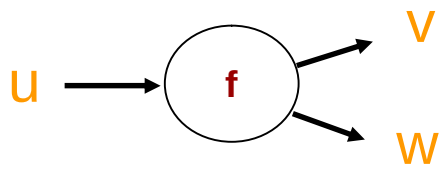


Kahn Process Networks (KPN)

- Important property: DETERMINISM
  - the sequence of values communicated through the channels is completely determined by the model
  - a KPN can be executed using a completely parallel schedule, a completely sequential schedule, or any schedule in between, always yielding the same output results for a given input sequence
  - which process to activate at a certain moment has to be decided during execution time, based on the current situation

Kahn Process Networks (KPN)

- Example



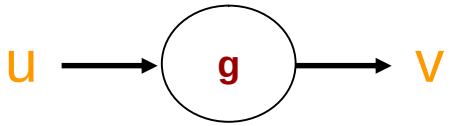

```
process f (InFIFO<int> u, OutFIFO<int> v, OutFIFO<int> w)
{
  int k;
  while (true){
      k = u.wait();
      if (k % 2 = 0) v.send (k);
      else w.send(k);

}
```

wait() returns the next token in an input FIFO, blocking if it's empty

send() writes a data value on an output FIFO

Kahn Process Networks (KPN)

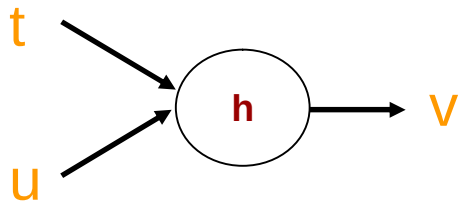
- Example




```
process g (InFIFO<int> u, OutFIFO<int> v)
{
   int k;
   while (true){
         k = u.wait();
         v.send (k);

}
```

Kahn Process Networks (KPN)

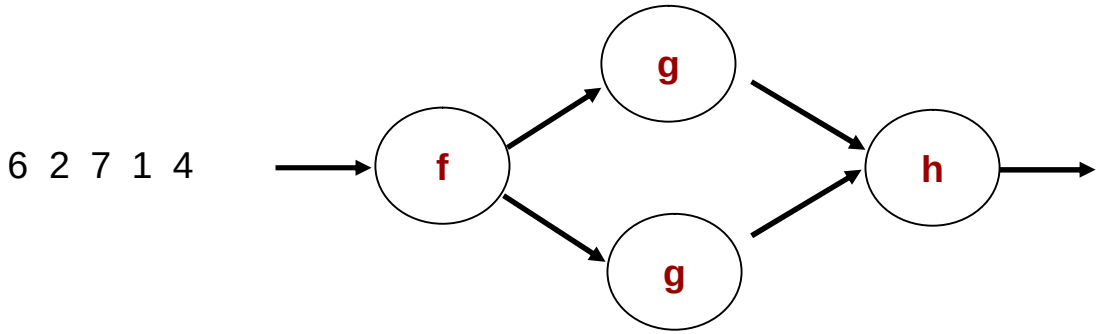
- Example




```
process h (InFIFO<int> t, InFIFO<int> u, OutFIFO<int> v)
{
   int k; boolean s=true;
   while (true){
         if (s) then k = t.wait();
         else k = u.wait();
         v.send (k);
         s=!s;}
}
```

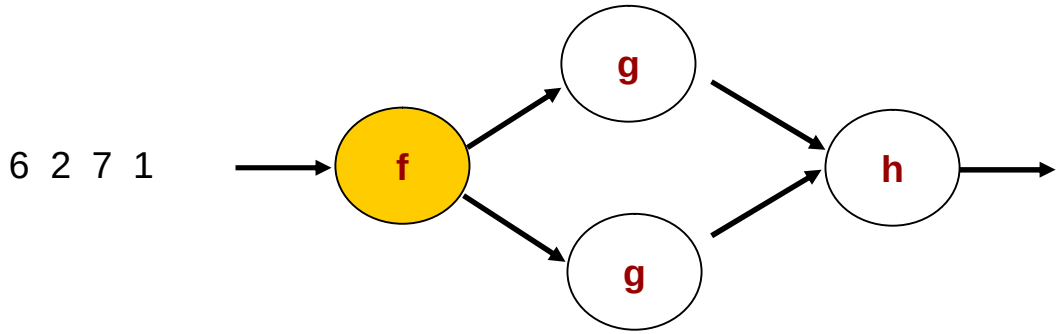
Kahn Process Networks (KPN)

- Example



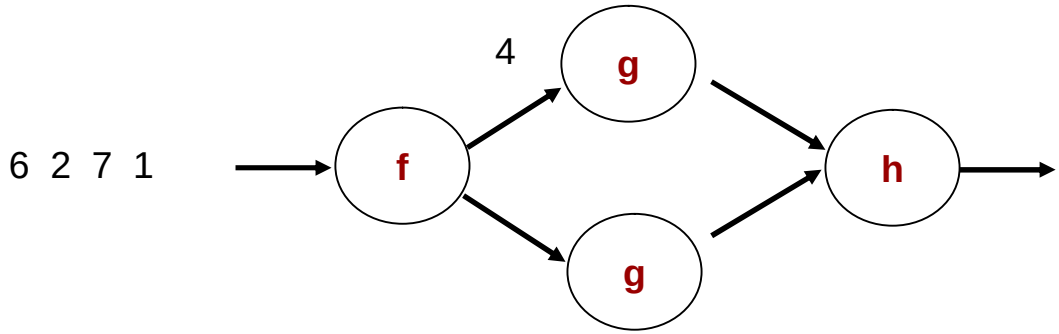
Kahn Process Networks (KPN)

- Example



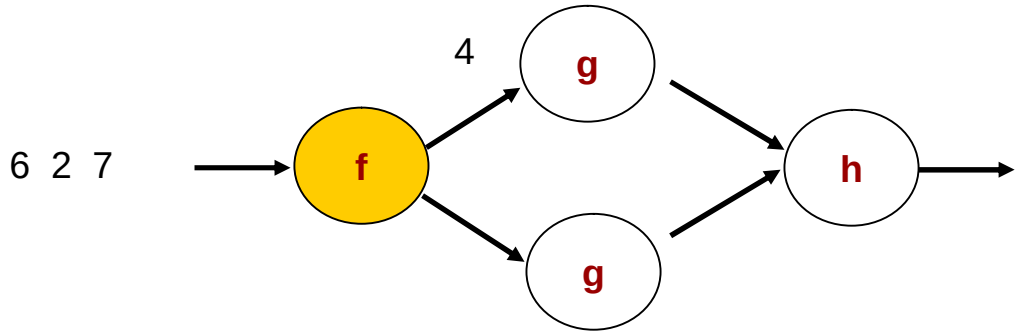
Kahn Process Networks (KPN)

- Example



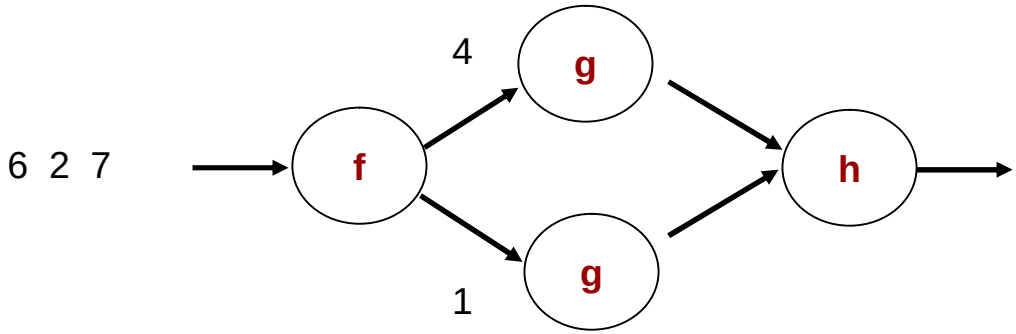
Kahn Process Networks (KPN)

- Example



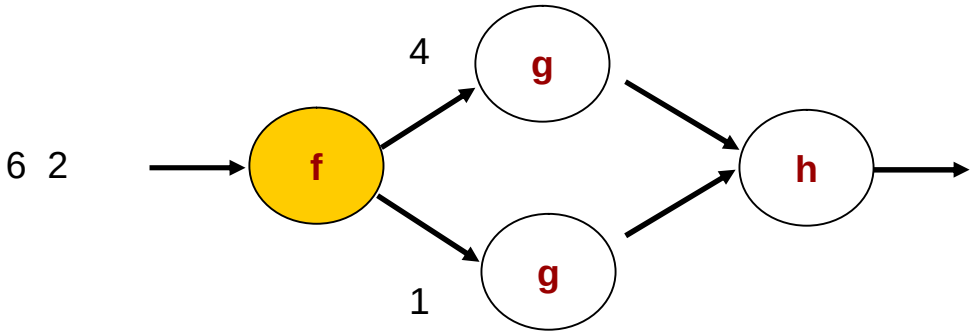
Kahn Process Networks (KPN)

- Example



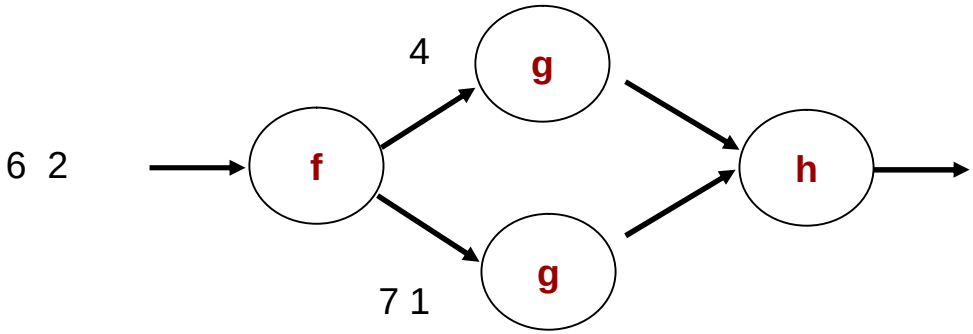
Kahn Process Networks (KPN)

- Example



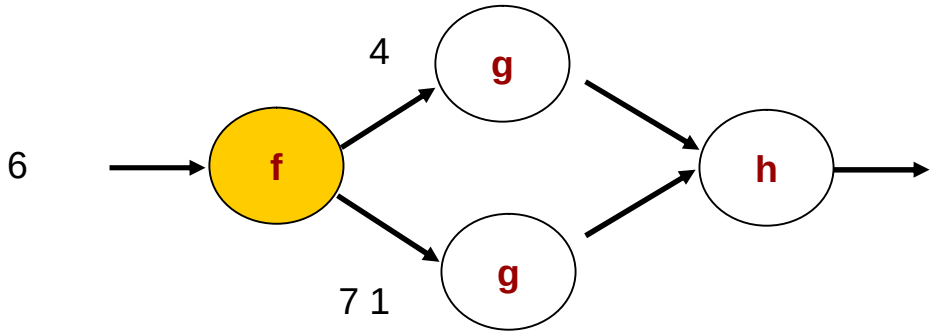
Kahn Process Networks (KPN)

- Example



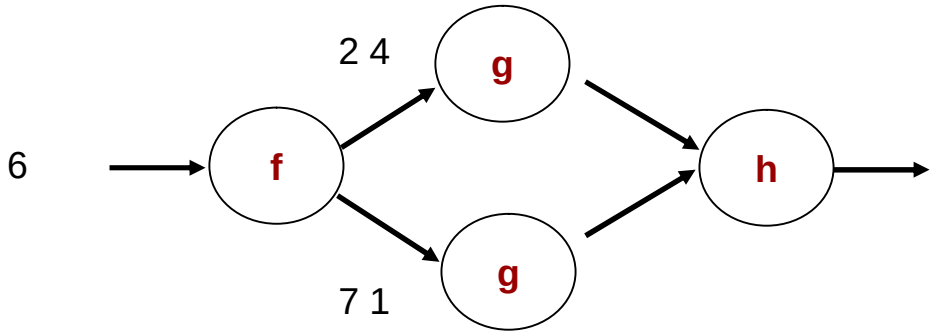
Kahn Process Networks (KPN)

- Example



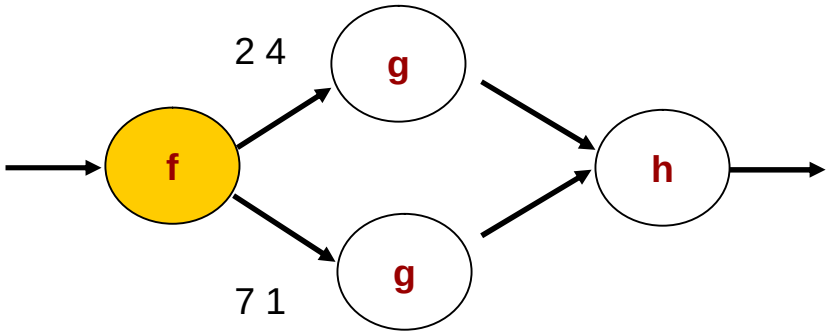
Kahn Process Networks (KPN)

- Example



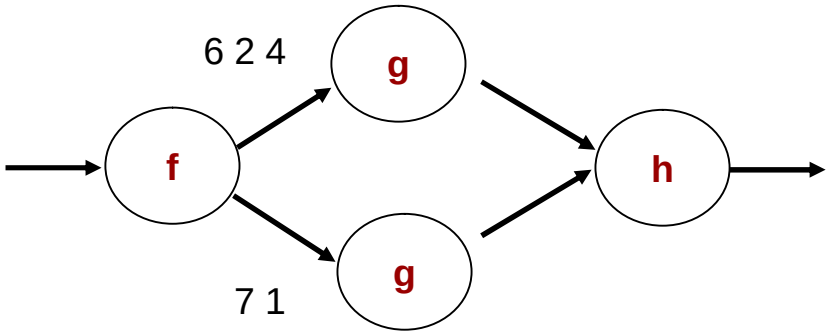
Kahn Process Networks (KPN)

- Example



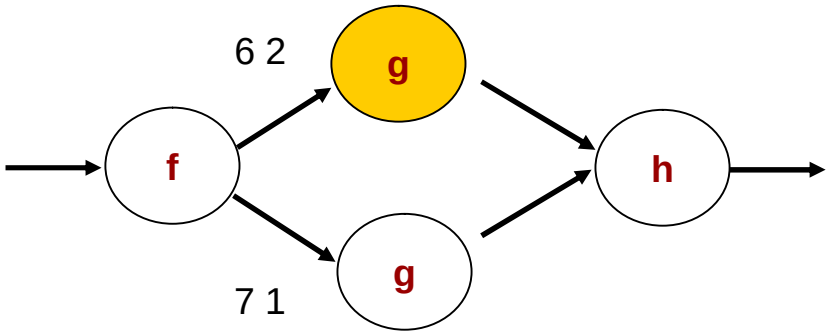
Kahn Process Networks (KPN)

- Example



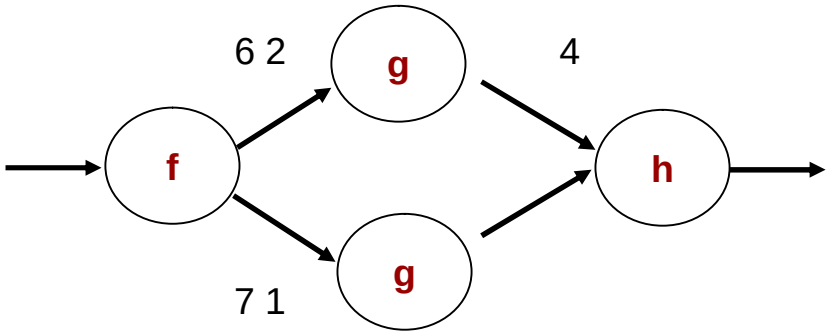
Kahn Process Networks (KPN)

- Example



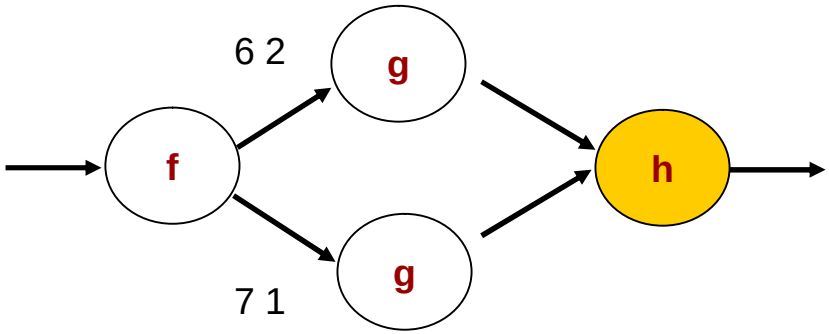
Kahn Process Networks (KPN)

- Example



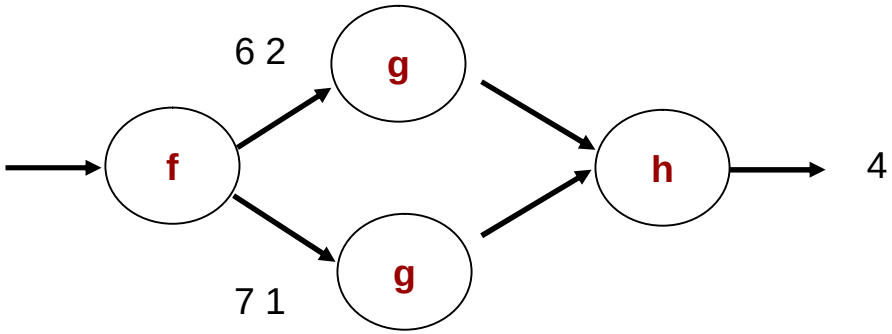
Kahn Process Networks (KPN)

- Example



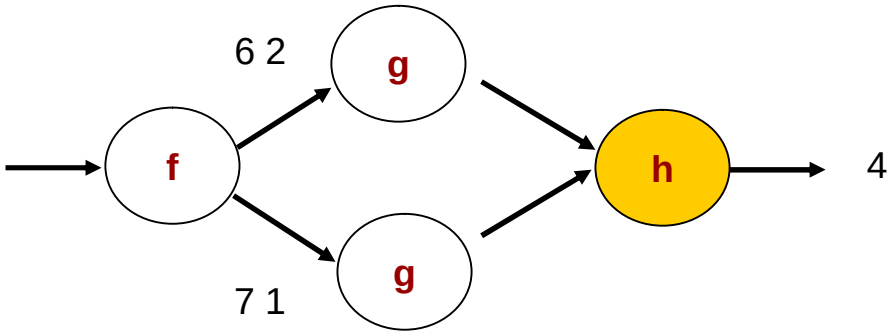
Kahn Process Networks (KPN)

- Example



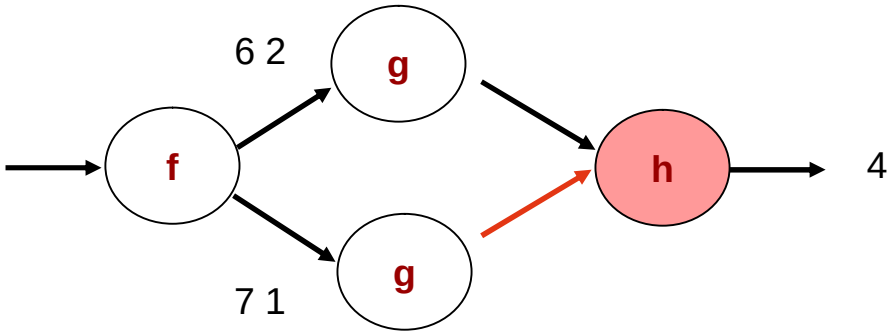
Kahn Process Networks (KPN)

- Example



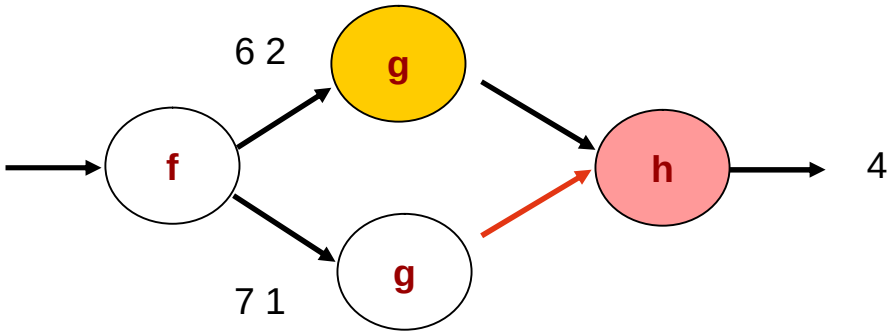
Kahn Process Networks (KPN)

- Example



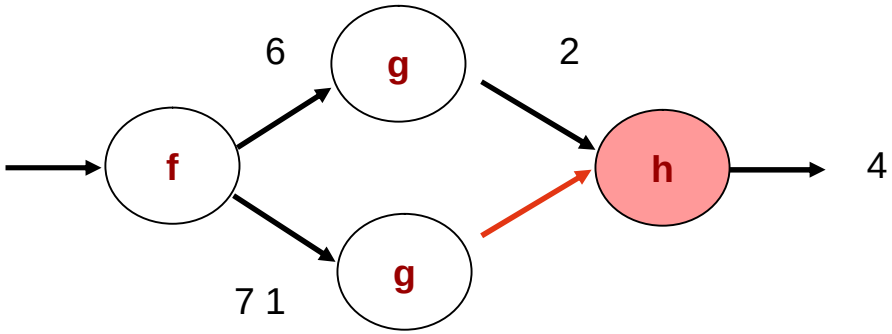
Kahn Process Networks (KPN)

- Example



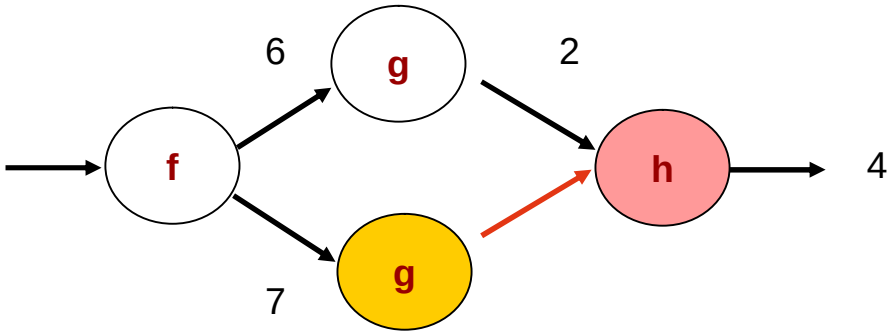
Kahn Process Networks (KPN)

- Example



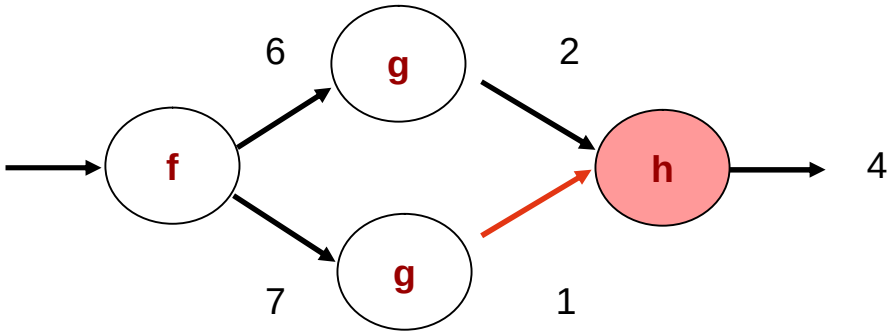
Kahn Process Networks (KPN)

- Example



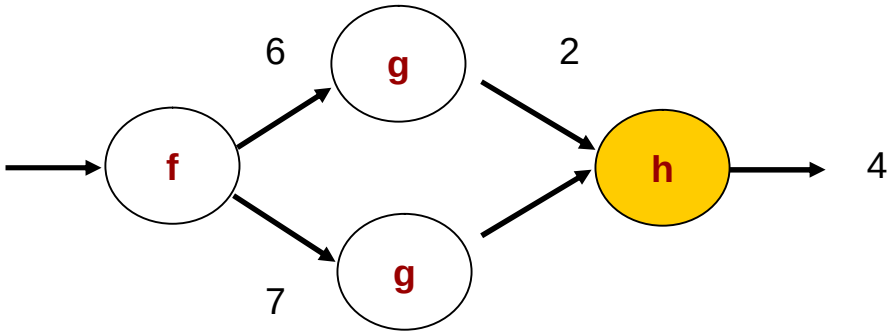
Kahn Process Networks (KPN)

- Example



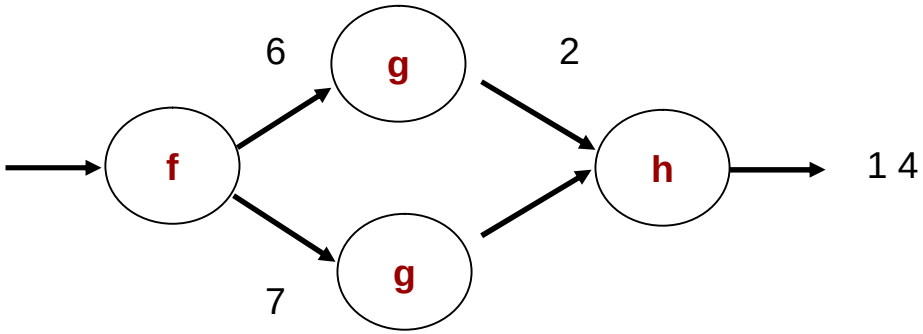
Kahn Process Networks (KPN)

- Example



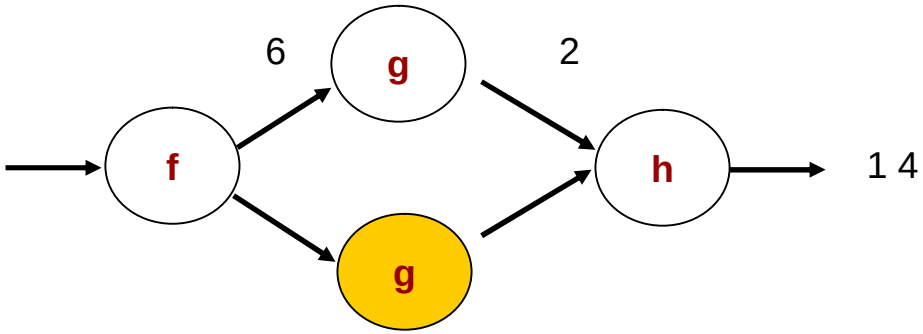
Kahn Process Networks (KPN)

- Example



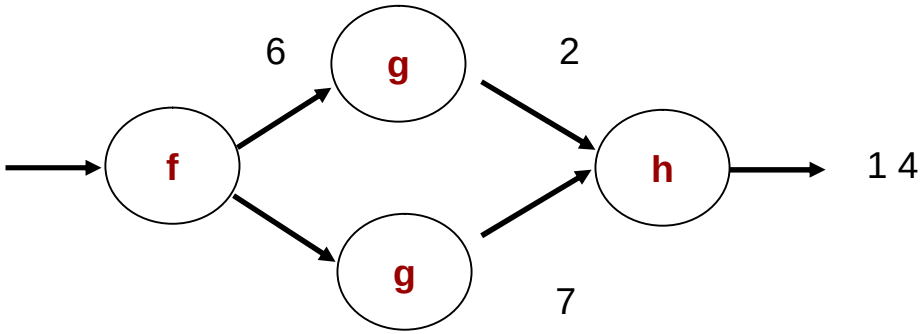
Kahn Process Networks (KPN)

- Example



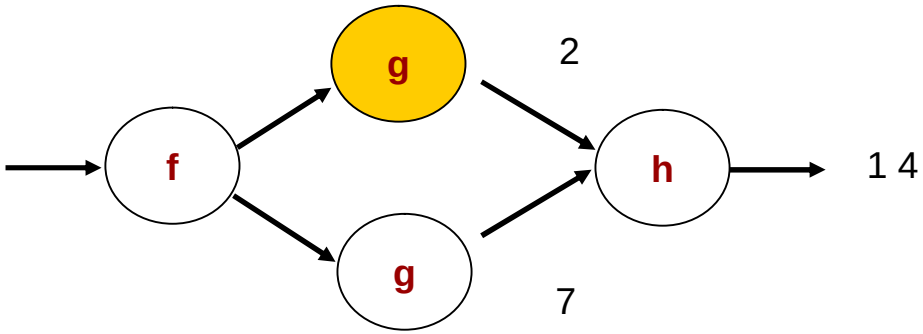
Kahn Process Networks (KPN)

- Example



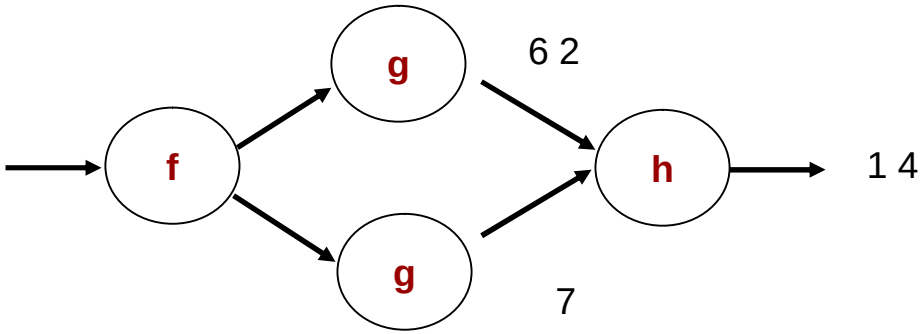
Kahn Process Networks (KPN)

- Example



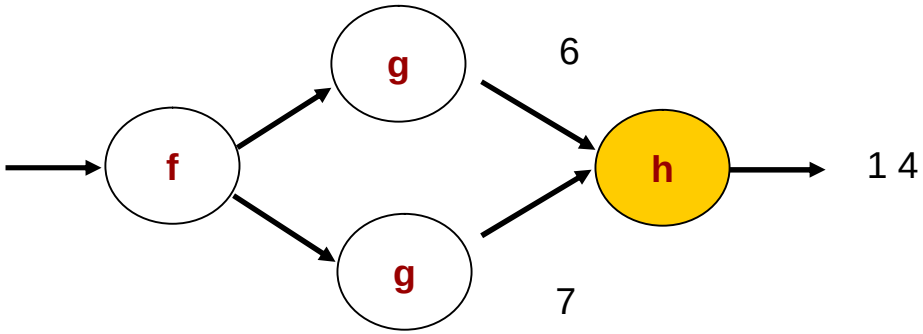
Kahn Process Networks (KPN)

- Example



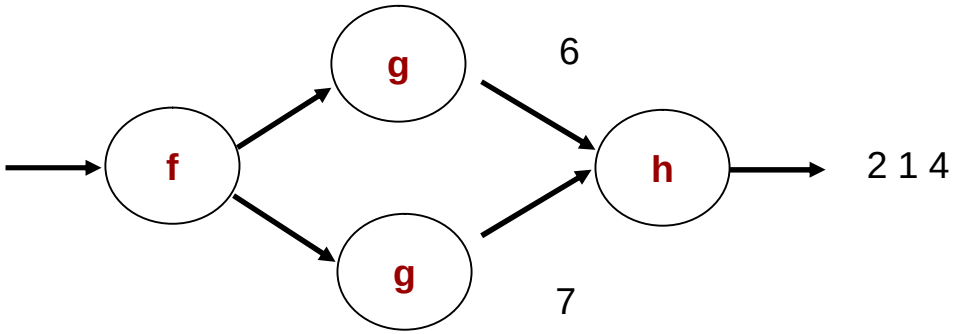
Kahn Process Networks (KPN)

- Example



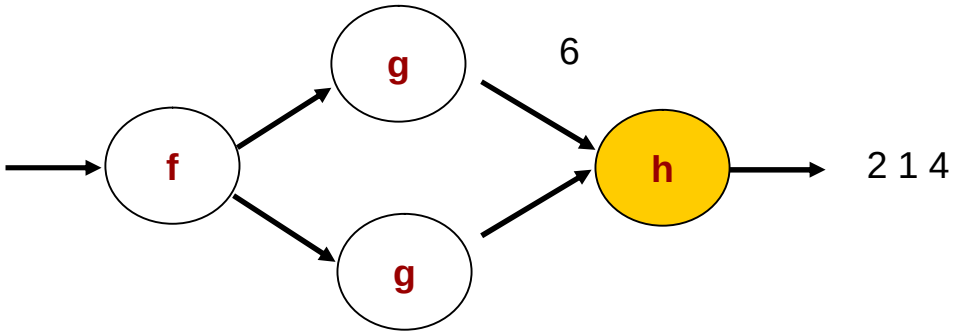
Kahn Process Networks (KPN)

- Example



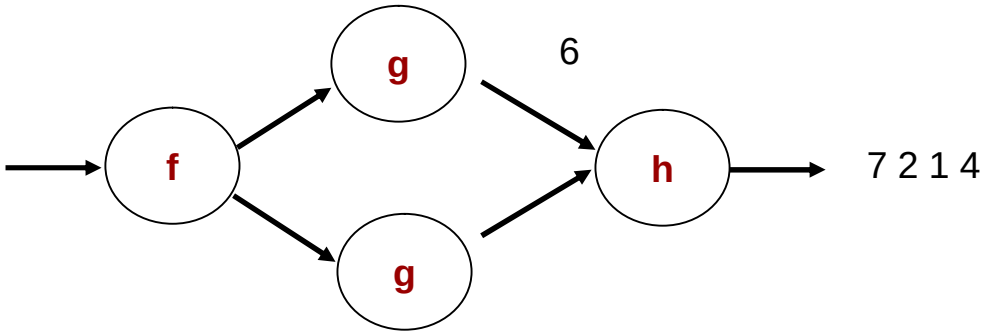
Kahn Process Networks (KPN)

- Example



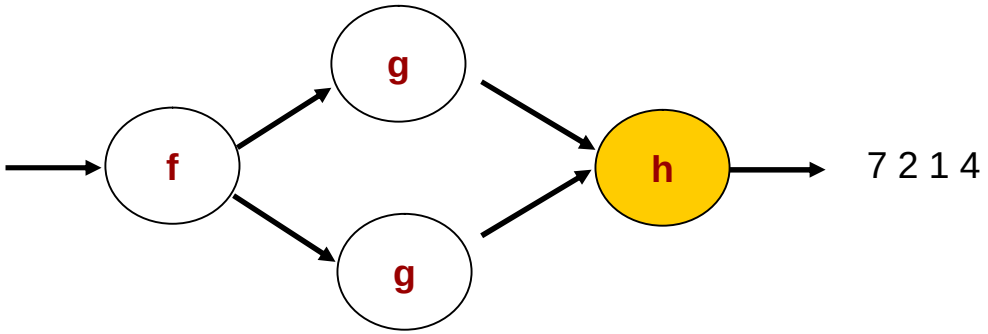
Kahn Process Networks (KPN)

- Example



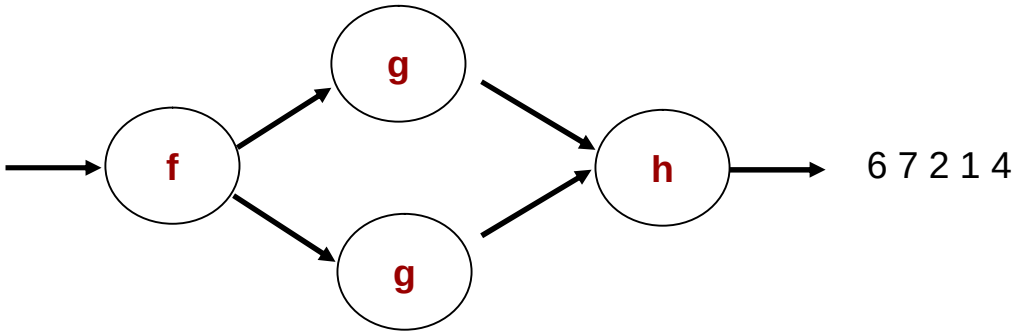
Kahn Process Networks (KPN)

- Example



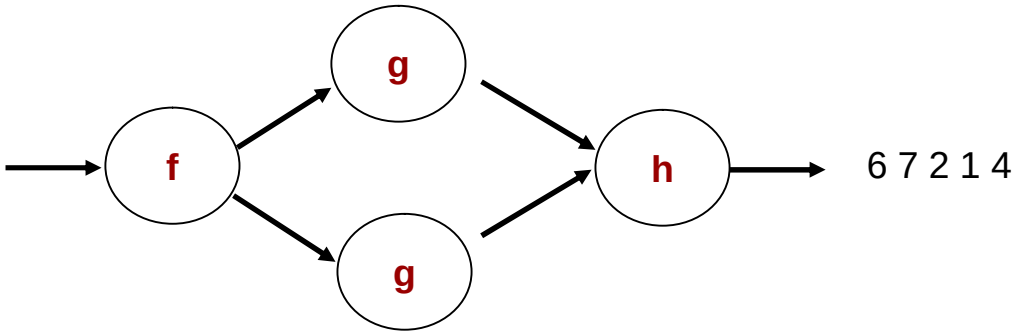
Kahn Process Networks (KPN)

- Example



Kahn Process Networks (KPN)

- Example



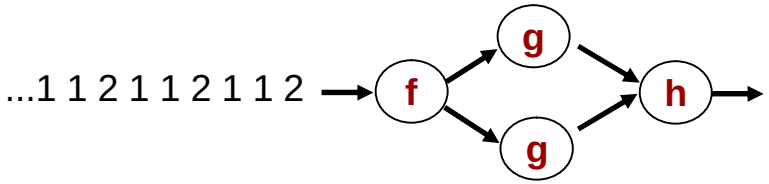
- Schedule
  - f f f f f g_u h h g_u g_d g_d g_u h h h

Kahn Process Networks (KPN)

- A given process is only affected by the sequence of tokens on its inputs
  - it can't tell whether they arrive early, late, or in what order
  - it will behave the same in any case
  - the sequence of tokens it puts on its outputs is the same regardless of the timing of the tokens on the inputs

Kahn Process Networks (KPN)

- Schedule must be determined at runtime
  - it doesn‘t affect functional behavior
  - challenge is to avoid accumulation of tokens
  - not all systems can be scheduled without token accumulation



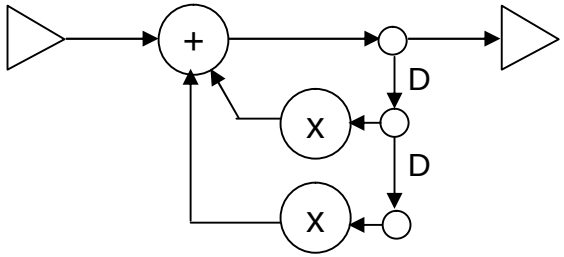
  - whether a KPN can execute in bounded memory is undecidable
  - a number of algorithms can find a schedule to execute a KPN in bounded memory (if such schedule exists)
    - usually require run-time deadlock detection

Synchronous Dataflow (SDF)

- Proposed by Lee and Messerschmitt in 1987
- Restricted type of KPN useful for modeling simple dataflow systems
- Constraint: each actor in a model reads and writes a fixed number of tokens every time it is fired
- Schedule can be statically determined
  - simpler and faster implementation of the model execution engine, bounded memory usage
  - deadlocks are avoided

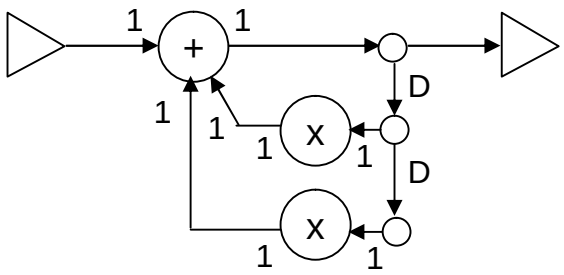
Synchronous Dataflow (SDF)

- Adequate for signal processing systems
  - number of consumed and produced tokens per firing is independent of the data and known beforehand



Synchronous Dataflow (SDF)

- Adequate for signal processing systems
  - number of consumed and produced tokens per firing is independent of the data and known beforehand



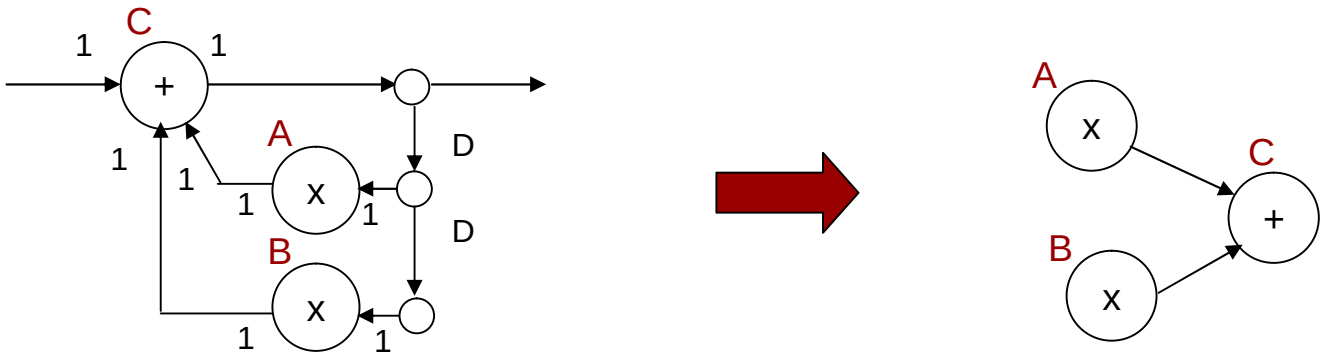
homogeneous SDF graph

Delay D – n^{th} token consumed by its successor is the $n-1^{th}$ token produced by its predecessor

- **initialized with d „zero“ tokens**

Synchronous Dataflow (SDF)

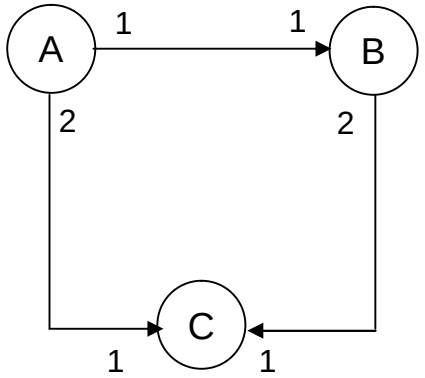
- Scheduling SDF
  - must respect precedence graph
  - no process fired unless all tokens it consumes are available



  - valid schedules: A B C , B A C
  - invalid schedule: C A B

Synchronous Dataflow (SDF)

- Periodic Admissable Sequential Schedule (PASS)
  - assumes infinite stream of data
  - periodic schedule can be applied repetitively on input stream without accumulating tokens in the buffers

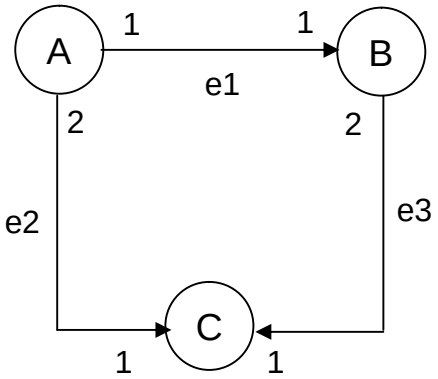


PASS:

A B C C

Synchronous Dataflow (SDF)

- Periodic Admissable Sequential Schedule (PASS)
  - formalism to determine PASS
  - 1. build the topology matrix Γ



$$\Gamma = \begin{array}{c} \\ e1 \\ e2 \\ e3 \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 1 & -1 & 0 \\ 2 & 0 & -1 \\ 0 & 2 & -1 \end{bmatrix} \end{array}$$

token consumption is negative

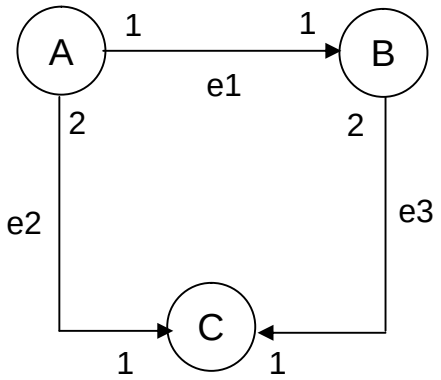
Synchronous Dataflow (SDF)

- Periodic Admissable Sequential Schedule (PASS)
  - formalism to determine PASS
  - 2. determine the relative firing frequency of each node by finding the smallest positive integer vector q such as Γ q = 0

$$\begin{bmatrix} 1 & -1 & 0 \\ 2 & 0 & -1 \\ 0 & 2 & -1 \end{bmatrix} \begin{bmatrix} q_A \\ q_B \\ q_C \end{bmatrix} = 0 \qquad \mathbf{q} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

Synchronous Dataflow (SDF)

- Periodic Admissable Sequential Schedule (PASS)
  - formalism to determine PASS
  - 3. if rates can be established, any scheduling algorithm that avoids buffer underflow will produce a correct schedule if it exists



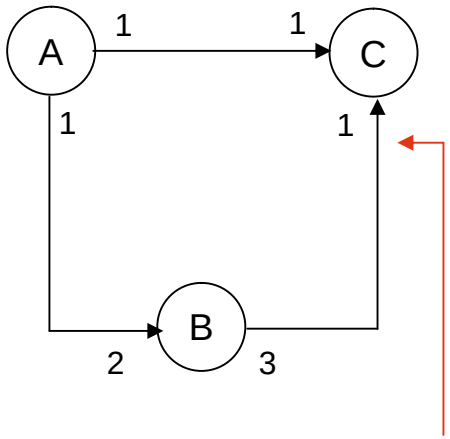
- for each node, schedule if runnable, trying each node once
- if each node has been scheduled $\mathbf{q}_i$ times, stop

$$\mathbf{q} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

PASS: A B C C

Synchronous Dataflow (SDF)

- Periodic Admissable Sequential Schedule (PASS)
  - not always possible to find a PASS



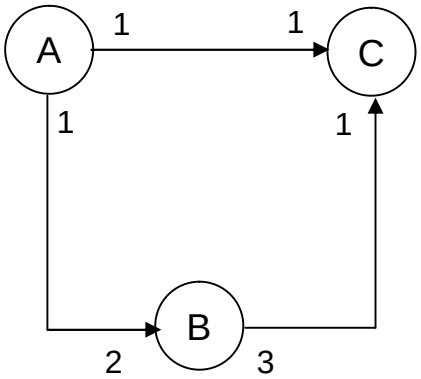
$$\Gamma = \begin{bmatrix} 1 & 0 & -1 \\ 1 & -2 & 0 \\ 0 & 3 & -1 \end{bmatrix} \qquad \mathbf{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

tokens will accumulate here

rate inconsistency

Synchronous Dataflow (SDF)

- Periodic Admissable Sequential Schedule (PASS)
  - how to test?
  - rank(Γ) = s – 1 (s stands for the number of actors on the graph)
    - necessary condition for the existence of a PASS



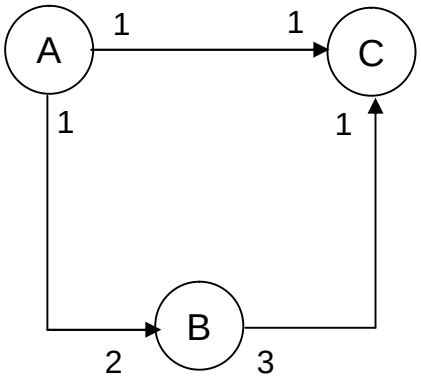
$$\Gamma = \begin{bmatrix} 1 & 0 & -1 \\ 1 & -2 & 0 \\ 0 & 3 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & -2 & 1 \\ 0 & 0 & 1/2 \end{bmatrix}$$

using Gaussian elimination

row echelon form

Synchronous Dataflow (SDF)

- Periodic Admissable Sequential Schedule (PASS)
  - how to test?
  - rank(Γ) = s – 1 (s stands for the number of actors on the graph)
    - necessary condition for the existence of a PASS



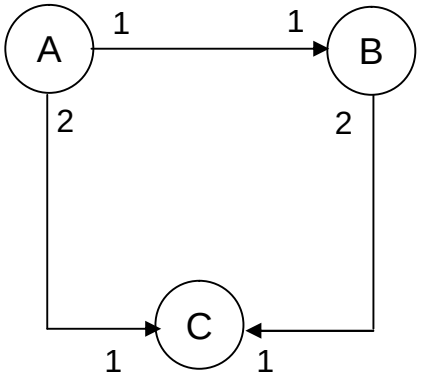
$$\Gamma = \begin{bmatrix} 1 & 0 & -1 \\ 1 & -2 & 0 \\ 0 & 3 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & -2 & 1 \\ 0 & 0 & 1/2 \end{bmatrix}$$

s = 3

rank(Γ) = 3

Synchronous Dataflow (SDF)

- Periodic Admissable Sequential Schedule (PASS)
  - how to test?
  - rank(Γ) = s – 1 (s stands for the number of actors on the graph)
    - necessary condition for the existence of a PASS



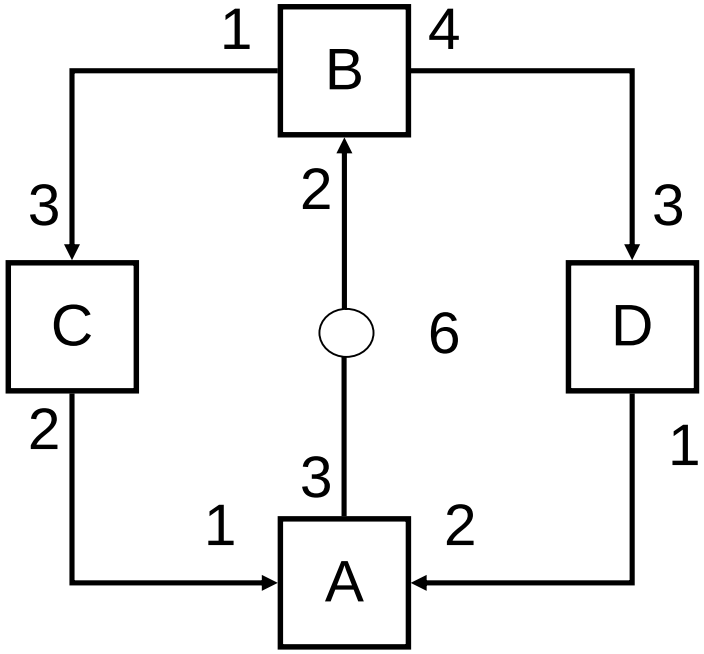
$$\Gamma = \begin{bmatrix} 1 & -1 & 0 \\ 2 & 0 & -1 \\ 0 & 2 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix}$$

s = 3

rank(Γ) = 2

Synchronous Dataflow (SDF)

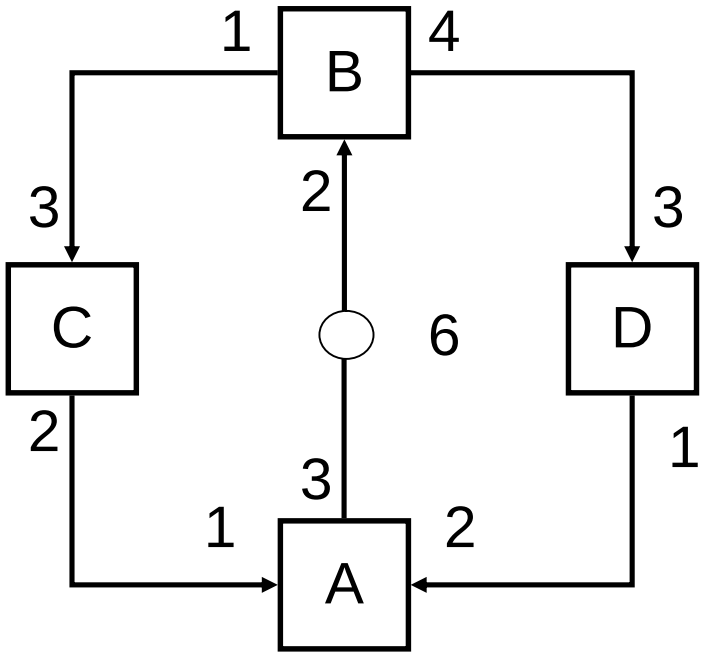
- Periodic Admissable Sequential Schedule (PASS)
  - slightly more complex example



$$\Gamma = \begin{bmatrix} -2 & 0 & 0 & 1 \\ -1 & 0 & 2 & 0 \\ 0 & 1 & -3 & 0 \\ 0 & 4 & 0 & -3 \\ 3 & -2 & 0 & 0 \end{bmatrix}$$

Synchronous Dataflow (SDF)

- Periodic Admissable Sequential Schedule (PASS)
  - slightly more complex example



$$q = \begin{bmatrix} 2 \\ 3 \\ 1 \\ 4 \end{bmatrix}$$

Possible schedules:

BBBCDDDDAA

BDBDBCADDA

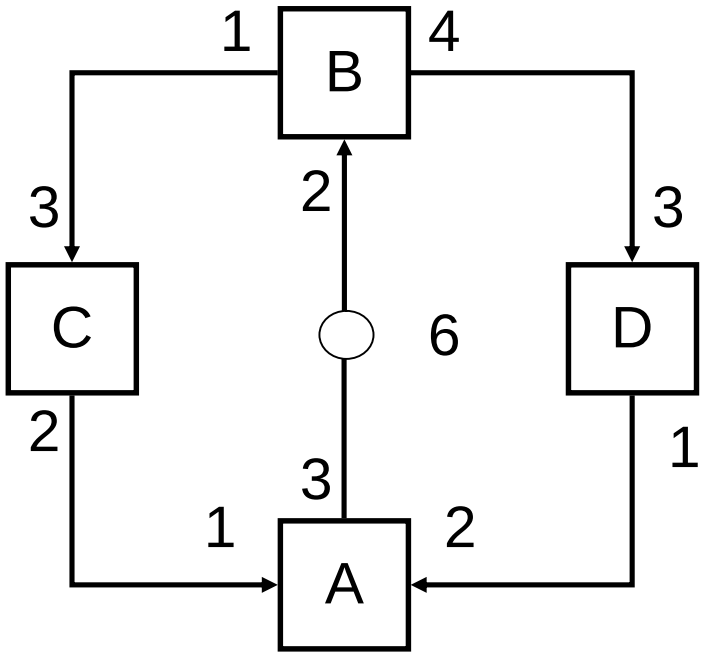
BBDDBDDCAA

… many more

BC … is not valid

Synchronous Dataflow (SDF)

- Periodic Admissable Sequential Schedule (PASS)
  - slightly more complex example



Possible schedules:

BBBCDDDDAA

BDBDBCADDA

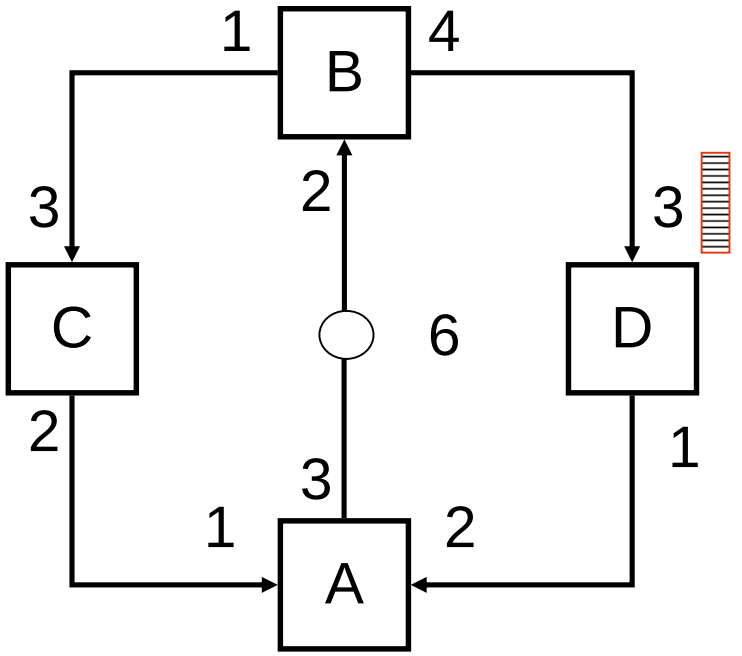
BBDDBDDCAA

… many more

Which one should I choose?

Synchronous Dataflow (SDF)

- Periodic Admissable Sequential Schedule (PASS)
  - how to choose a schedule?



may require larger buffers

Possible schedules:

BBBCDDDDAA

BDBDBCADDA

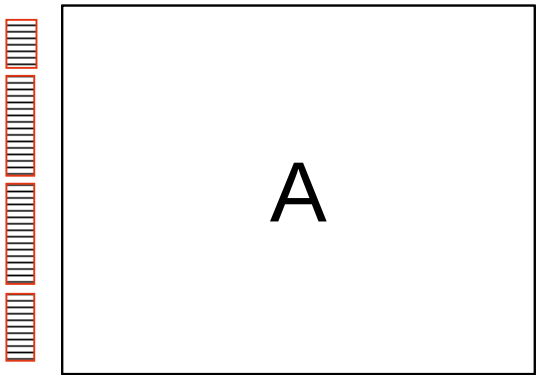
BBDDBDDCAA

… many more

single appearance schedule

Synchronous Dataflow (SDF)

- Periodic Admissable Sequential Schedule (PASS)
  - how to choose a schedule?



useful for a reconfigurable hardware implementation

Possible schedules:

BBBCDDDDAA

BDBDBCADDA

BBDDBDDCAA

… many more

single appearance schedule

Synchronous Dataflow

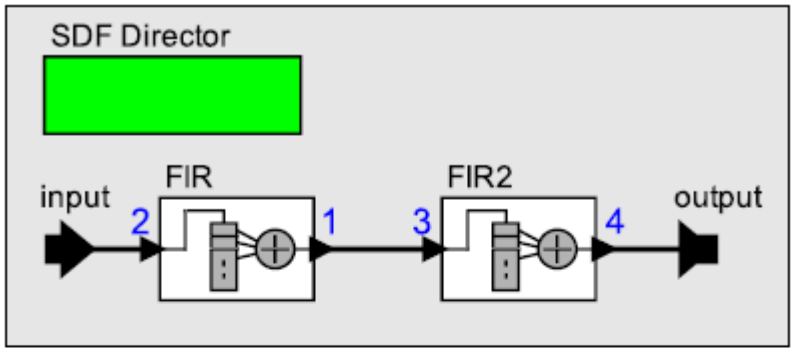
- Typical uses:
  - signal processing

Synchronous Dataflow

Another example:

SDF scheduling in two steps

- **Establish relative execution rates by solving a system of linear equations**
- **Determine periodic schedule by simulating system for a single round**



(a) Example Model

$$\begin{aligned} \text{input.tokenConsumptionRate} &= 2 * \text{FIR.firingCount} \\ 1 * \text{FIR.firingCount} &= 3 * \text{FIR2.firingCount} \\ 4 * \text{FIR2.firingCount} &= \text{output.tokenProductionRate} \end{aligned}$$

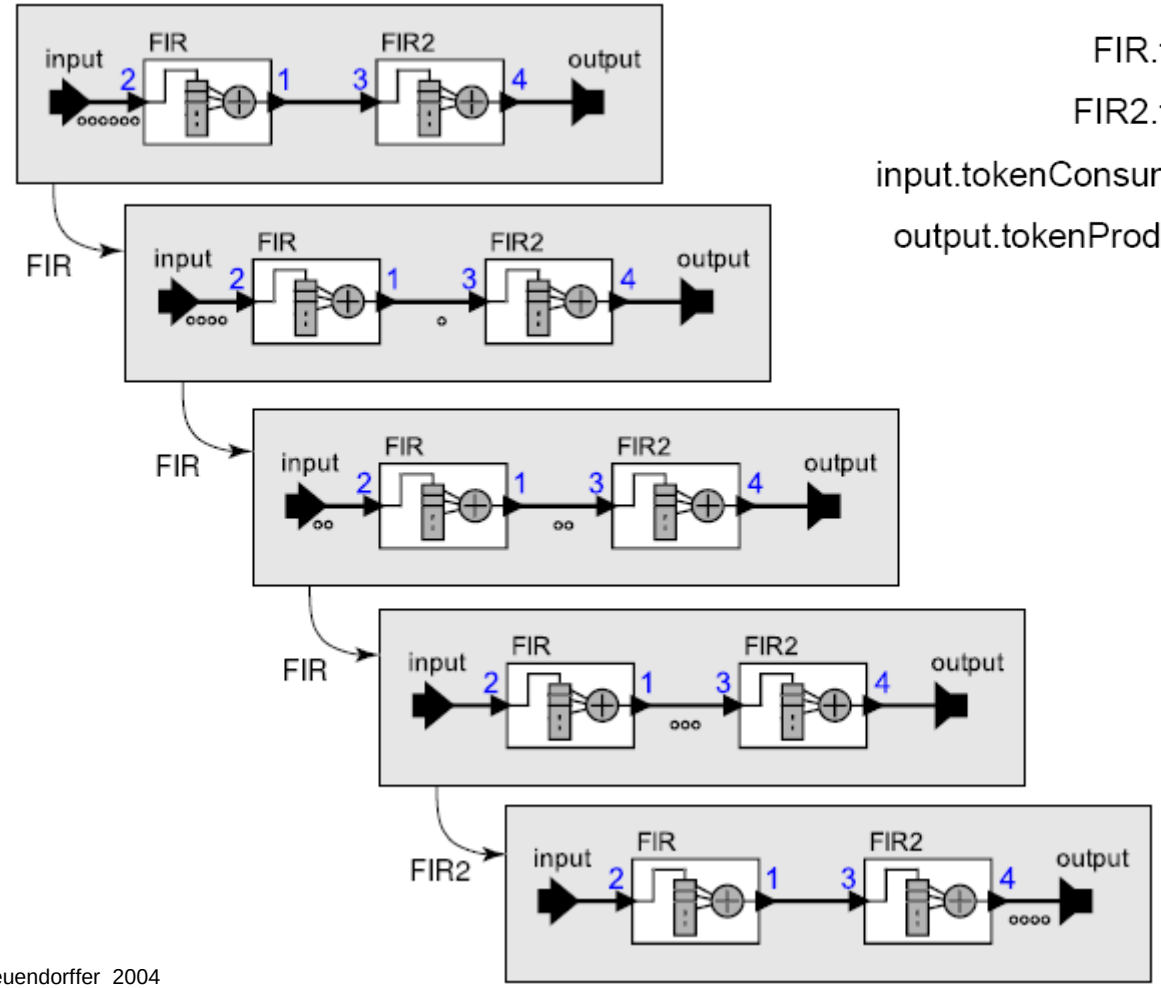
(b) Balance Equations

$$\begin{aligned} \text{FIR.firingCount} &= 3 \\ \text{FIR2.firingCount} &= 1 \\ \text{input.tokenConsumptionRate} &= 6 \\ \text{output.tokenProductionRate} &= 4 \end{aligned}$$

(c) Balance Equation Solution

source: Neuendorffer 2004

Synchronous Dataflow



FIR.firingCount = 3

FIR2.firingCount = 1

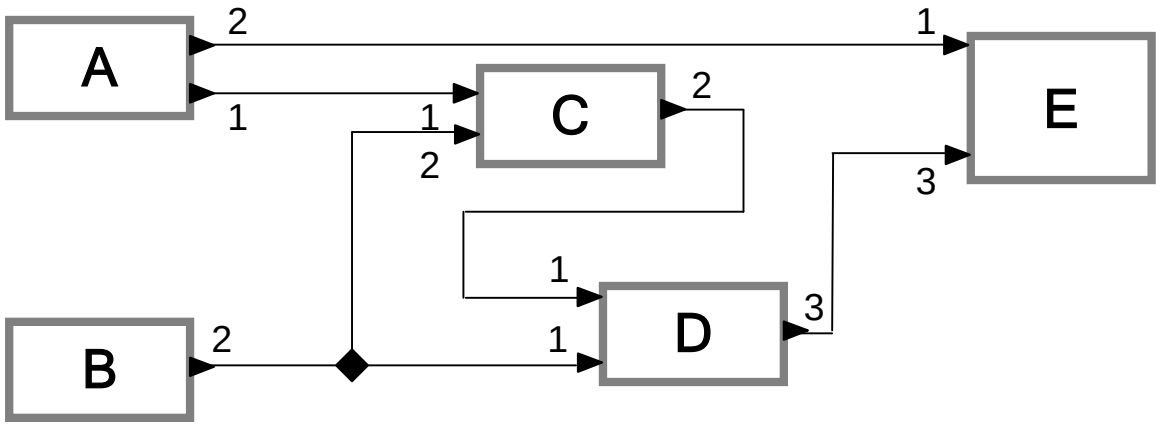
input.tokenConsumptionRate = 6

output.tokenProductionRate = 4

source: Neuendorffer 2004

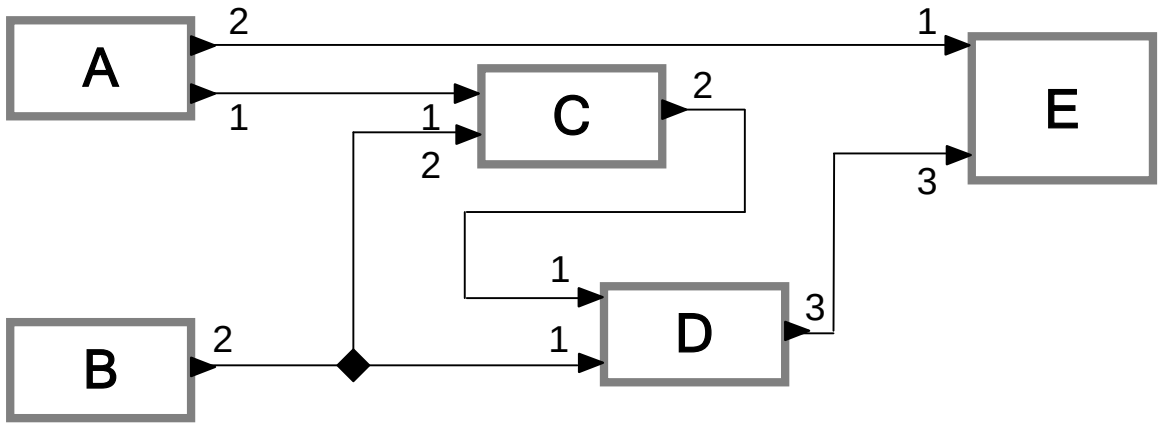
Synchronous Dataflow

- Exercise:
  - Consider the actor oriented model below following a synchronous dataflow model of computation. Calculate the balance equations and find a valid schedule for it in such a way that no actor requires buffers larger than 3 on its input ports.



Synchronous Dataflow

- Exercise:
  - 2A – E = 0
  - A – C = 0
  - 2B – 2C = 0
  - 2B – D = 0
  - 2C – D = 0
  - 3D – 3 E = 0



  - A = B = C = 1
  - D = E = 2

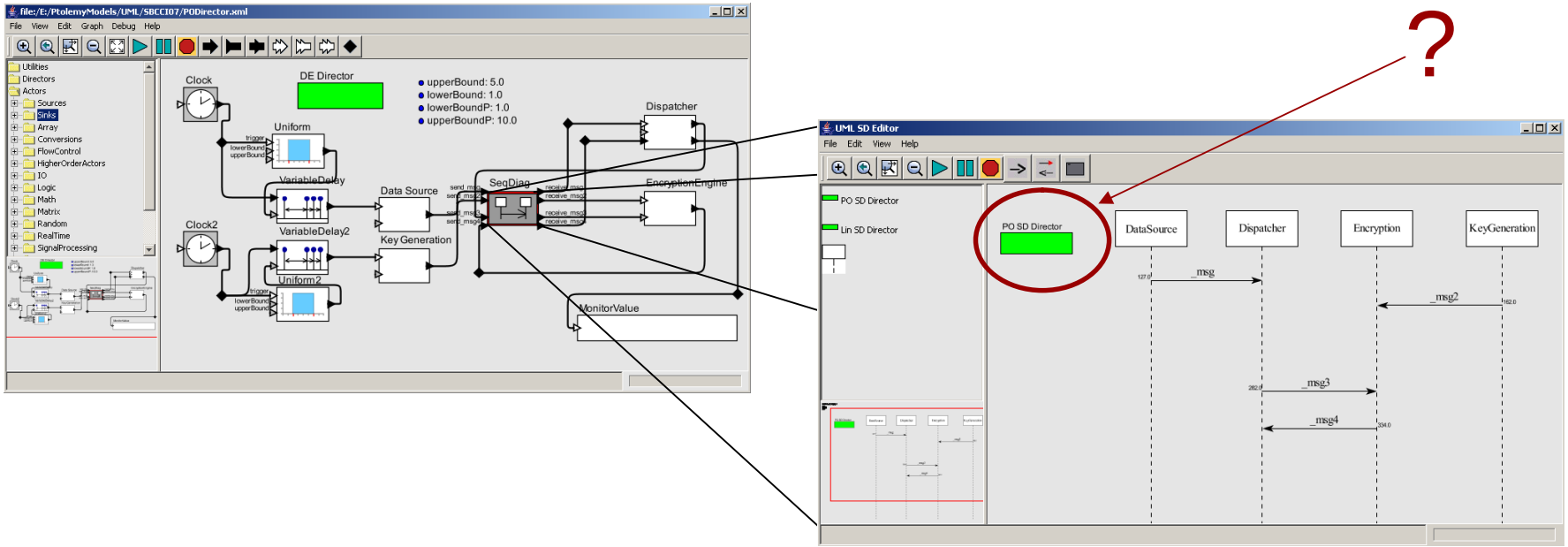
- Valid schedules:
  - A B C D D E E (requires 6 position buffer at D-E channel)
  - A B C D E D E (all buffers have 3 positions or less)

Course Outline

- On-Chip Multiprocessing
  - motivation, review
- Application-specific Multiprocessor Platforms
  - validation, concurrency issues, application-platform mapping
- Platform models
  - evaluation, accuracy
- Application models
  - concurrency
- Joint execution of application and platform models
  - case study
- Background theory and tool support
  - actor orientation, Ptolemy II, custom made extensions
  - demo

Application Modelling

- Extending actor-orientation with types and explicit ordering
  - UML suitable visual representation for the definition of polymorphic type systems (class diagrams) and ordering relations (sequence diagram)
  - but UML is not an executable specification language

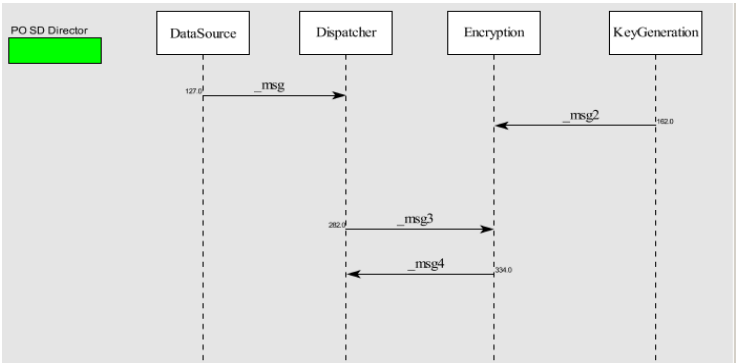


UML sequence diagrams within actors

- Recall the formal definition of MSC
-
- tuple $\langle P, E, C, l, m, < \rangle$ where
  - P is a finite set of processes
  - E is a finite set of events
  - C is a finite set of names for messages
  - $l: E \rightarrow T = \{ p!q(a), p?q(a), p(a) \mid p \neq q \in P, a \in C \}$
  - $m: S \rightarrow R$
  - $< \subseteq E \times E$ *is a acyclic relation between events consisting of:*
    - *a total order on* E_P *for every* $p \in P$*, and*
    - $s < r$*, whenever* $m(s)=r$

UML sequence diagrams within actors

- Recall the formal definition of MSC
  - order of events (message occurrence) within a process (lifeline) is a total order
  - the reflexive-transitive closure of < (denoted as <*) is a partial order on the complete set E
  - enough for the definition of an untimed model of computation



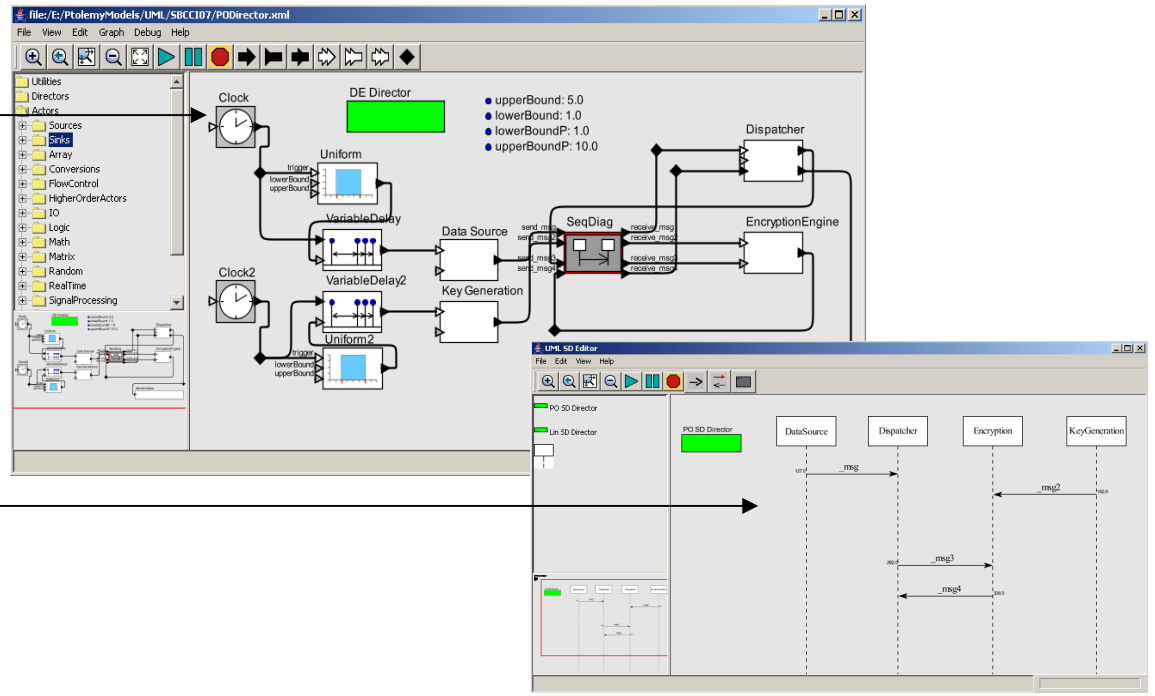
  - different possibilities were explored and integrated as a library of directors on an extended version of Ptolemy II

Application modelling

- Application modeling based on behavioral patterns and polymorphic type systems

application functionality described using actors

constraints to concurrent execution described using UML



Platform modelling