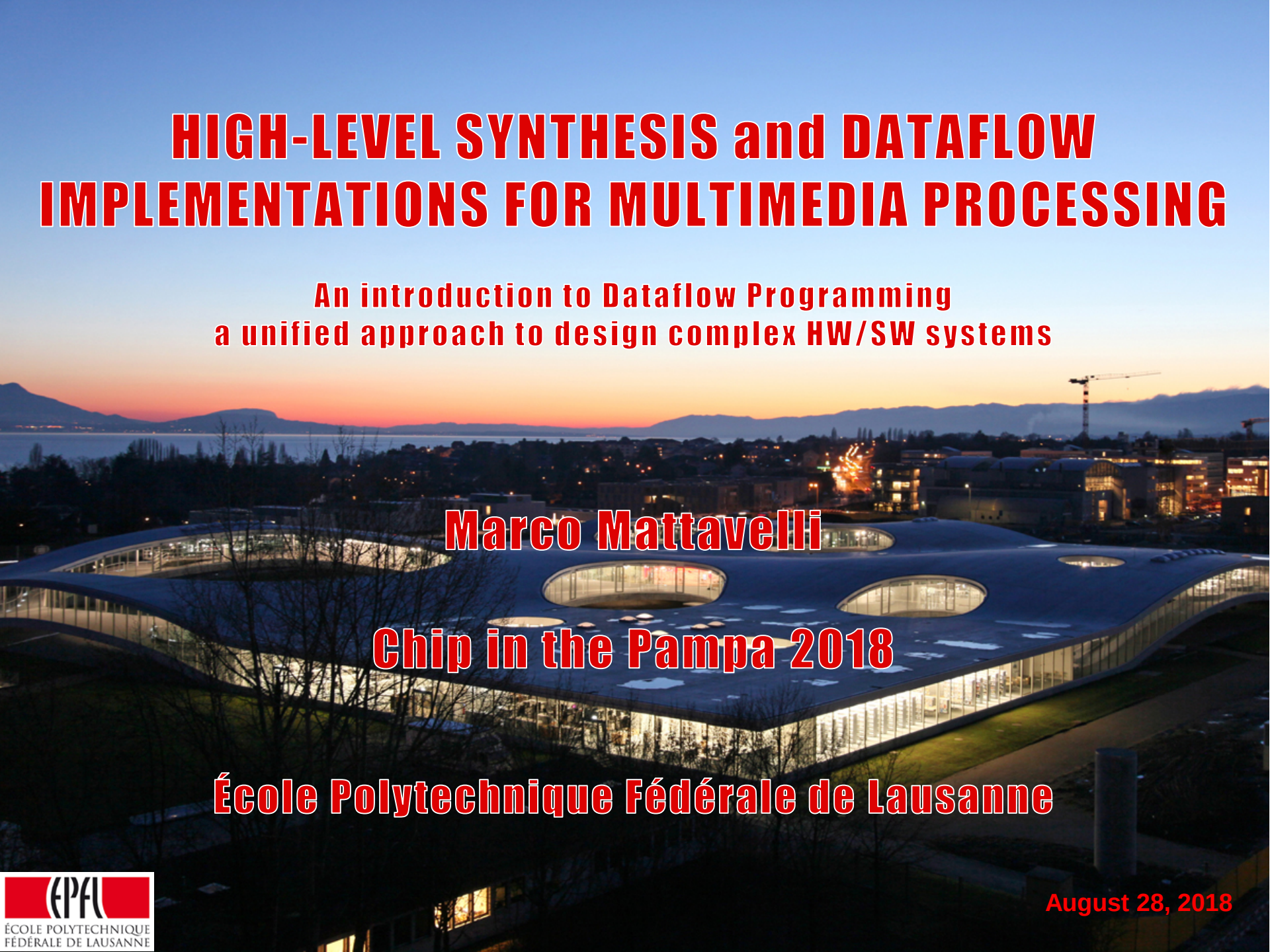
HIGH-LEVEL SYNTHESIS and DATAFLOW IMPLEMENTATIONS FOR MULTIMEDIA PROCESSING

An introduction to Dataflow Programming
a unified approach to design complex HW/SW systems



Marco Mattavelli

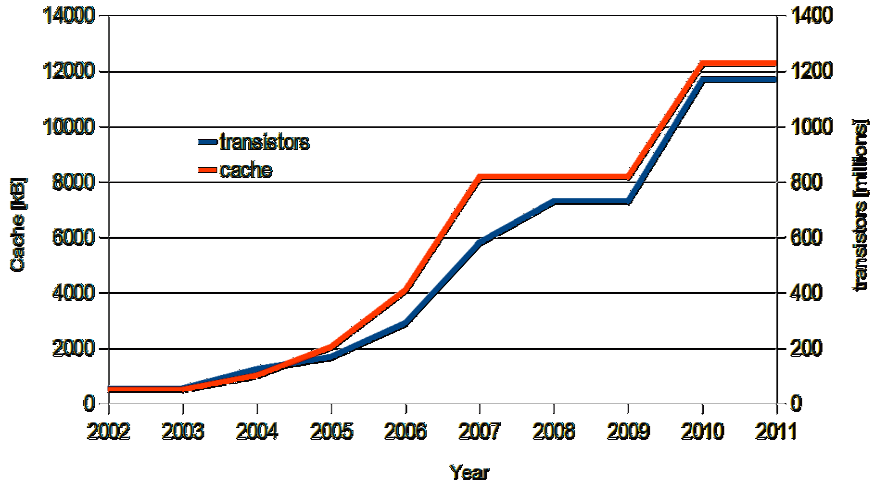
Chip in the Pampa 2018

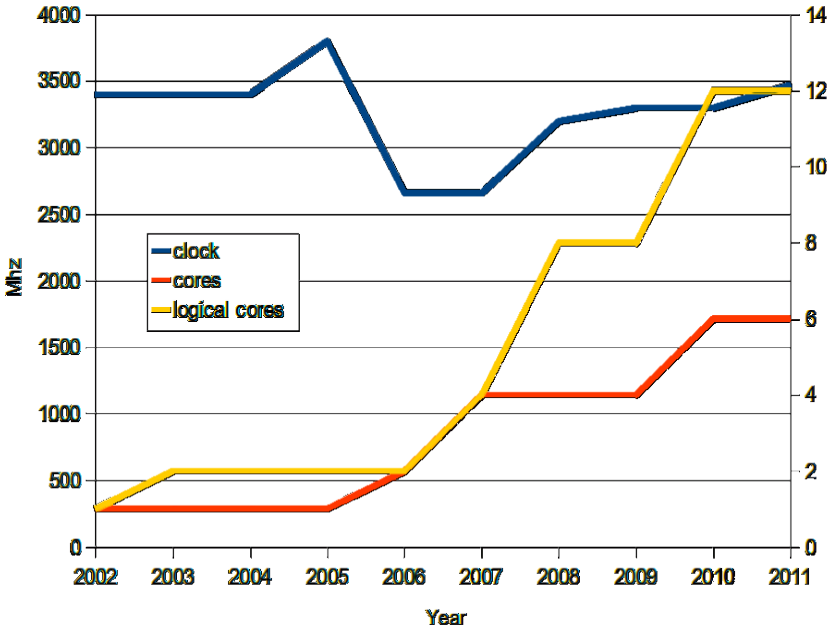
École Polytechnique Fédérale de Lausanne

August 28, 2018

EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Clock and cores



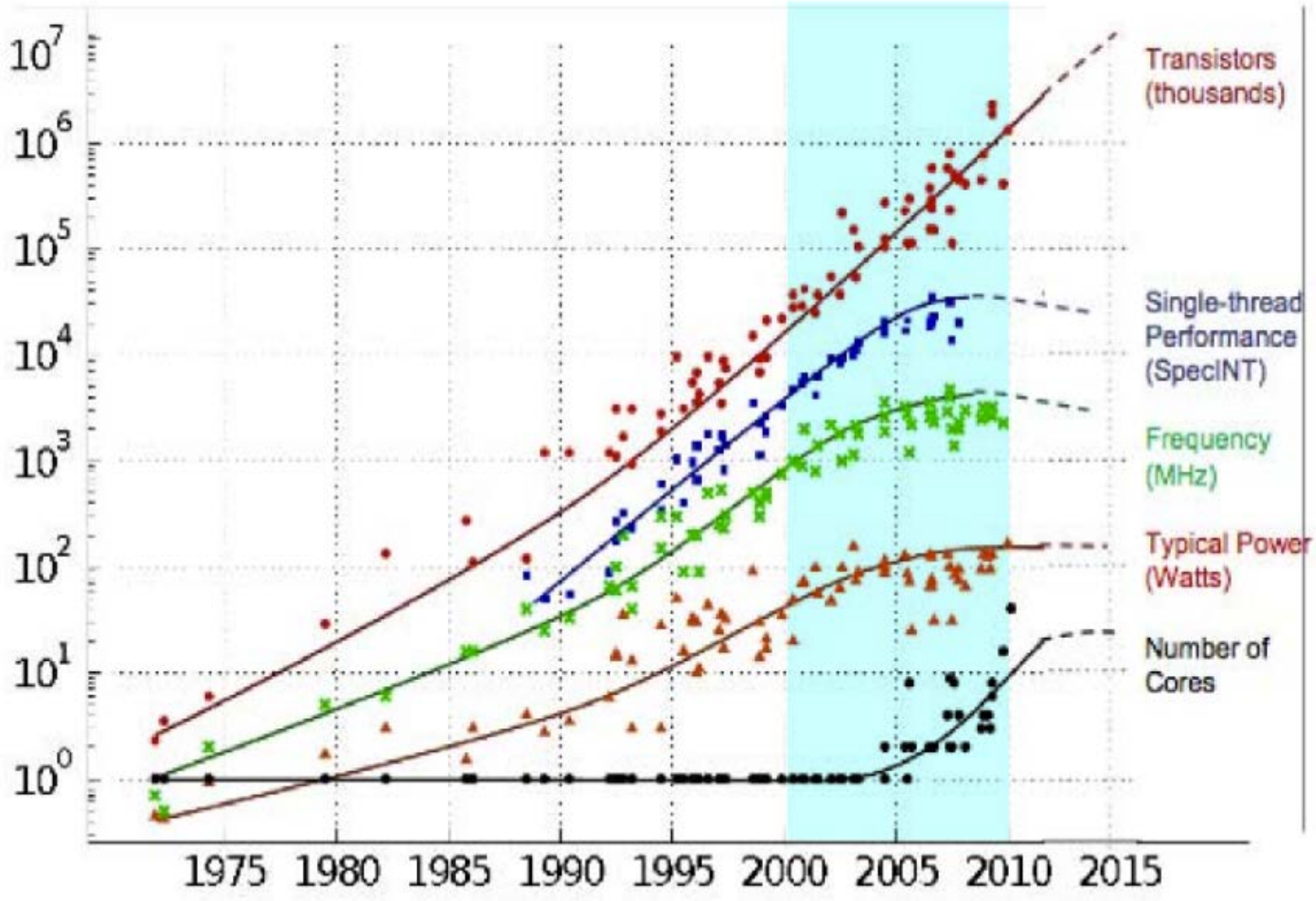


Intel desktop processor top model

Circuits are still getting larger at a steady pace...

… but speed-up is no longer the result of a faster clock. Instead, we get more parallel devices

The end of scaling

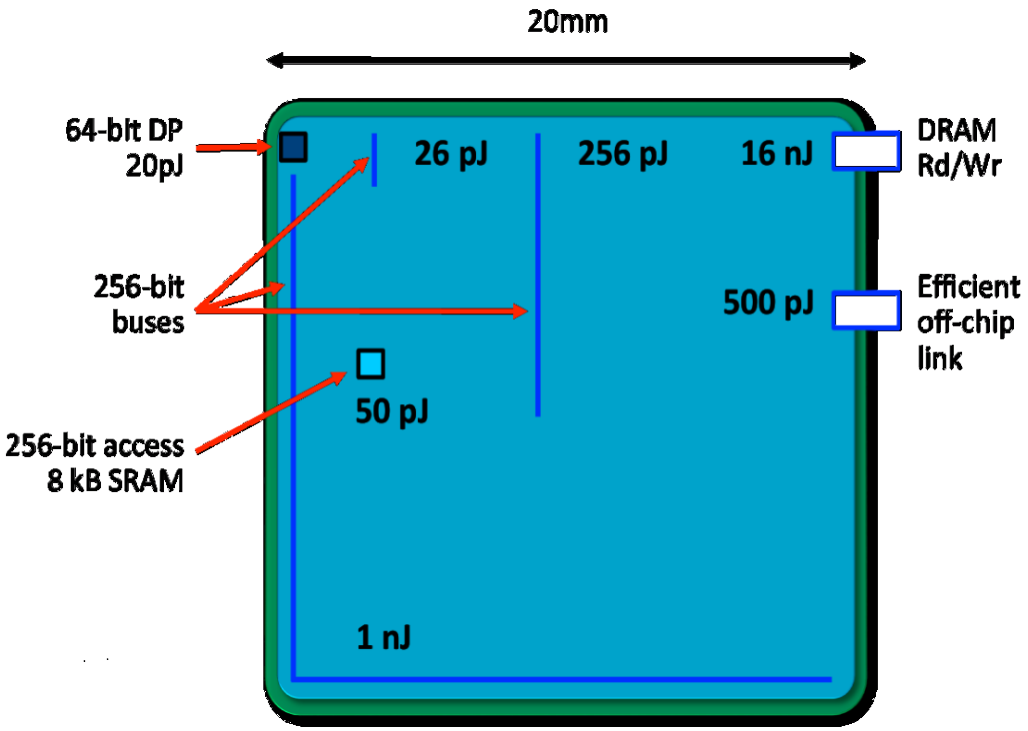


Original data collected and plotted by M. Horowitz, F. Labonte, O.Shacham, K. Olukotun, L. Hammond, and C. Batten. Dotted line extrapolations by C. Moore.
C. Moore, Data Processing in ExaScale-ClassComputer Systems, April 2011

Power efficiency in CMOS technology

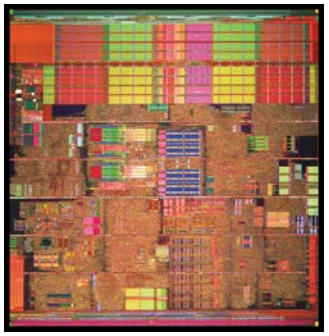
Performance = parallelism. Efficiency = locality.

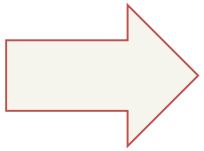
It really is that simple. (Bill Dally)

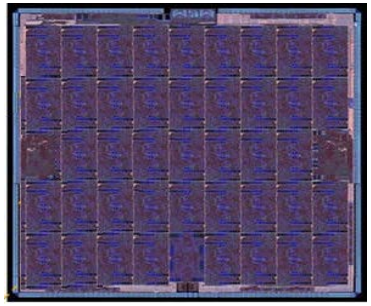


approximate power in
28nm processor
Bill Dally

Parallelism and locality

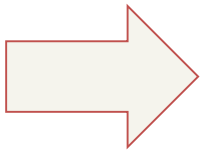




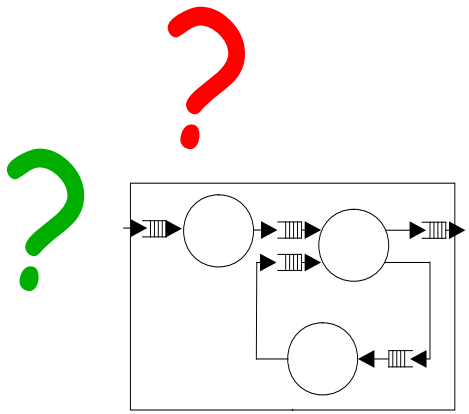


Computer architecture

```
for(int i=1;i<=N_a;i++){
  for(int j=1;j<=N_b;j++){
    temp[0] = H[i-1][j-1]+
            sim_score(seq_a[i-1],seq_b[j-1]);
    temp[1] = H[i-1][j]-delta;
    temp[2] = H[i][j-1]-delta;
    temp[3] = 0.;
    H[i][j] = find_array_max(temp,4);
    switch(ind){
    case 0:
      I_i[i][j] = i-1;
      I_j[i][j] = j-1;
      break;
    case 1:
      I_i[i][j] = i-1;
      I_j[i][j] = j;
      break;
    case 2:
      I_i[i][j] = i;
      I_j[i][j] = j-1;
      break;
    case 3:
      I_i[i][j] = i;
      I_j[i][j] = j;
      break;
    }
  }
}
```



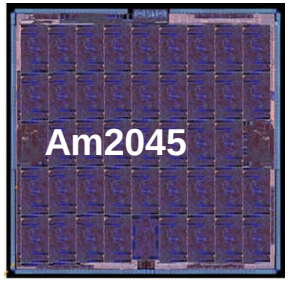
Programming

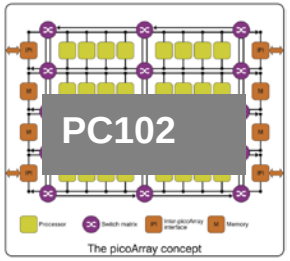


Programming Applications on Parallel Machines

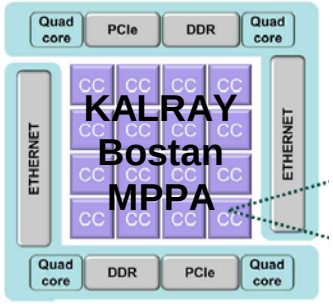
SOPM (“ajava”)

C + VHDL



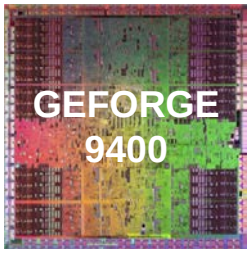


C w/ “iLib”



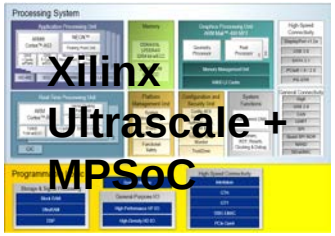
VentureForth

CUDA



C / C++ + Open CL

HDL



C + VHDL + Open CL

Programming Applications on Sequential Machines

C,
Pascal,
Forth,
Java,
Lisp,
Fortran,
Haskell,
...

“Von Neumann” machine

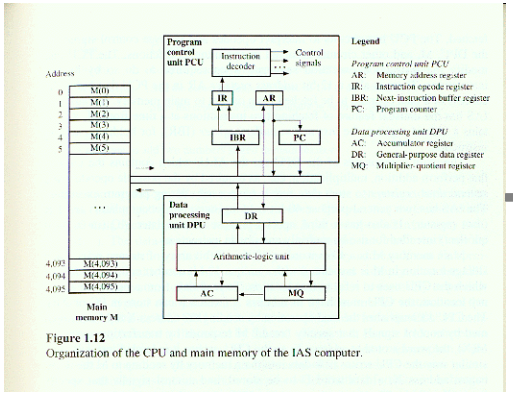
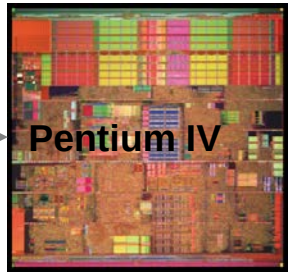
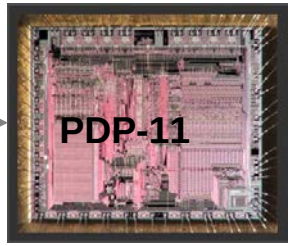
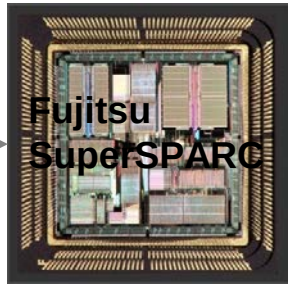
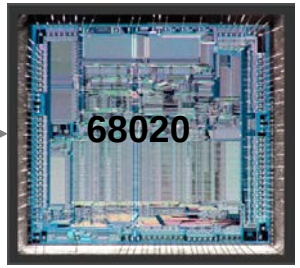


Figure 1.12
Organization of the CPU and main memory of the IAS computer.

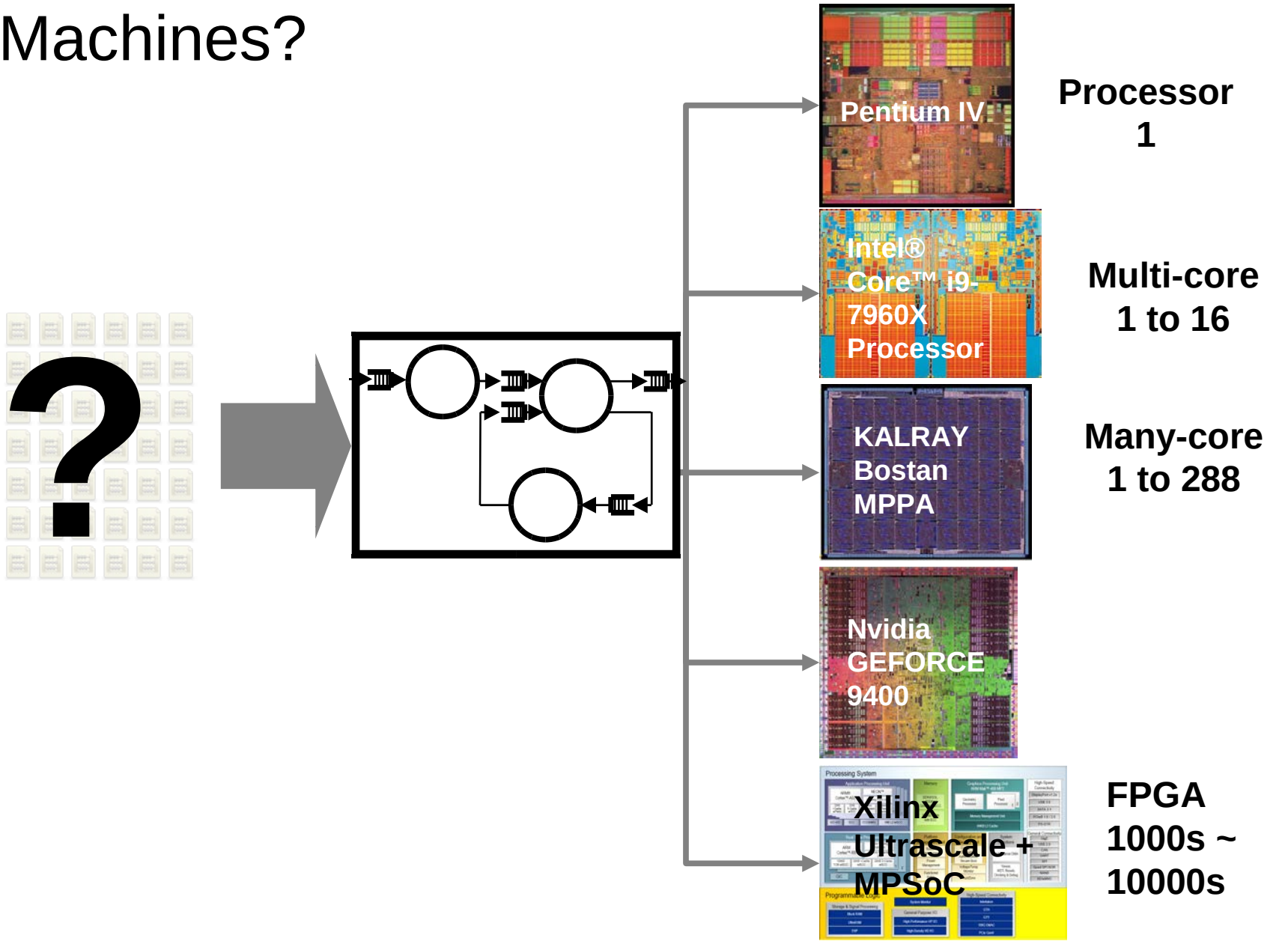




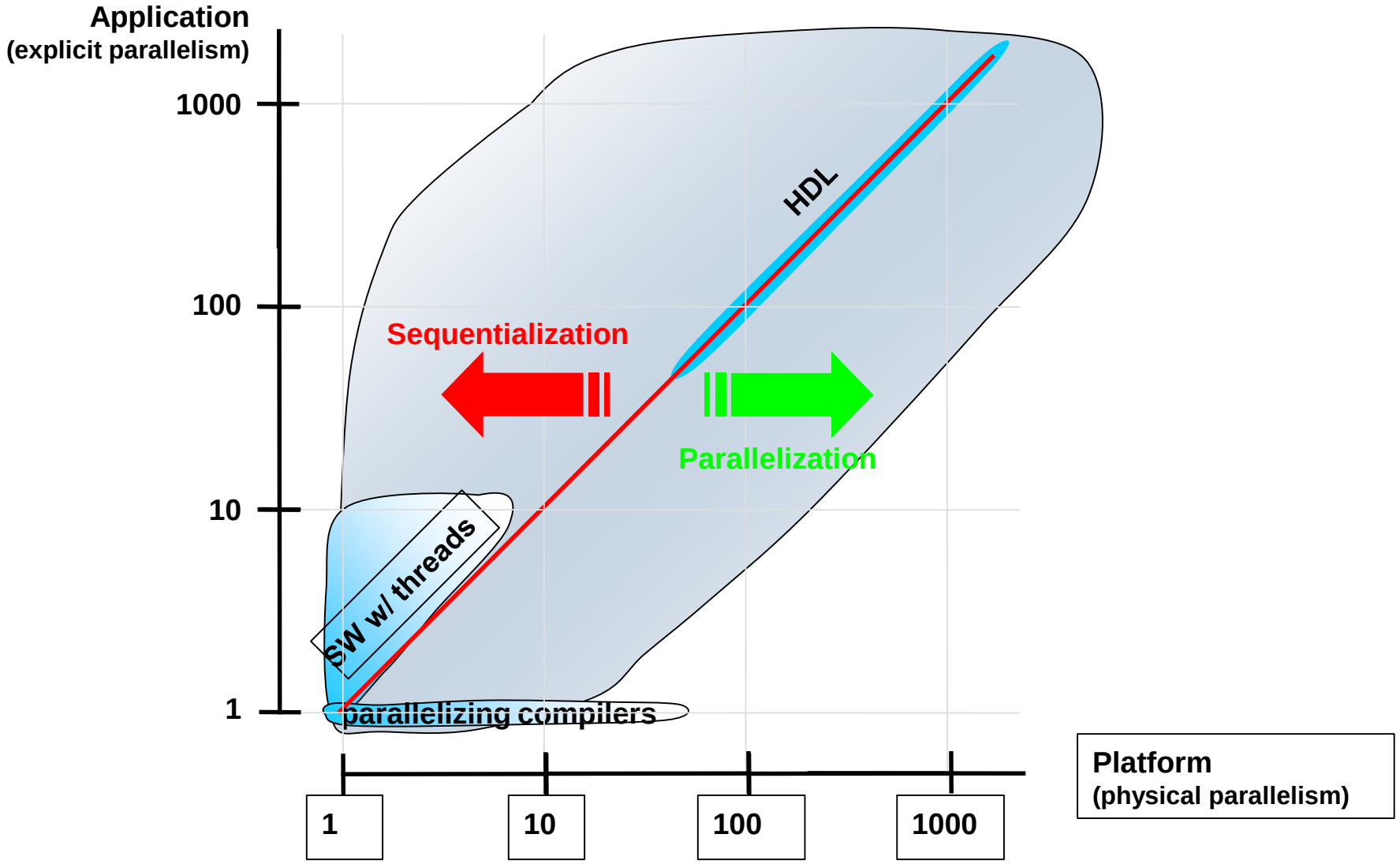




Programming Concurrent Heterogeneous Machines?



Portable Concurrency: the Need of a Paradigm Shift



Which Features do we need?

For software to scale to future parallel computing platforms as seamlessly as possible, it will be necessary to describe algorithms in a way that:
1) exposes as much parallelism of the application as practical,
2) provides simple and natural abstractions that help manage the high degree of parallelism and permits principled composition of and interaction between modules,
3) makes minimal assumptions about the physical architecture of the computing machine it is implemented on,
4) is efficiently implementable on a wide range of computing substrates, including traditional sequential processors, shared-memory multicores, manycore processor arrays, and programmable logic devices, as well as their combinations.

Looking at the above criteria:
shared-memory threads fulfill the first requirement, but essentially **fail** on the other three.
hardware description languages such as VHDL and Verilog fulfill the first two criteria, but as they fail on the third point by assuming a particular model of (clocked) time, their implementability is essentially limited to hardware and hardware-like programmable logic (FPGAs).

Problems with sequential computation models

No explicit parallelism is available: **lack of expressivity**

A deterministic sequence of low level operation is fully specified: no distinction between what is necessary for a correct semantics and what is not (memory bandwidth optimizations are more difficult)

Memory bandwidth does not scale with the number of cores/processors

Why threading does not solve the problem?

Threads are sequential processes that share memory.

Pros:
Languages require little or no syntactic changes to support threads, and operating systems and architectures have evolved to efficiently support them.

Cons:
They discard the most essential and appealing properties of sequential computation: understandability, predictability, and determinism.

Threads do not compose

n = 0;

t1 = n;
n = t1 + 1;

Thread 1:

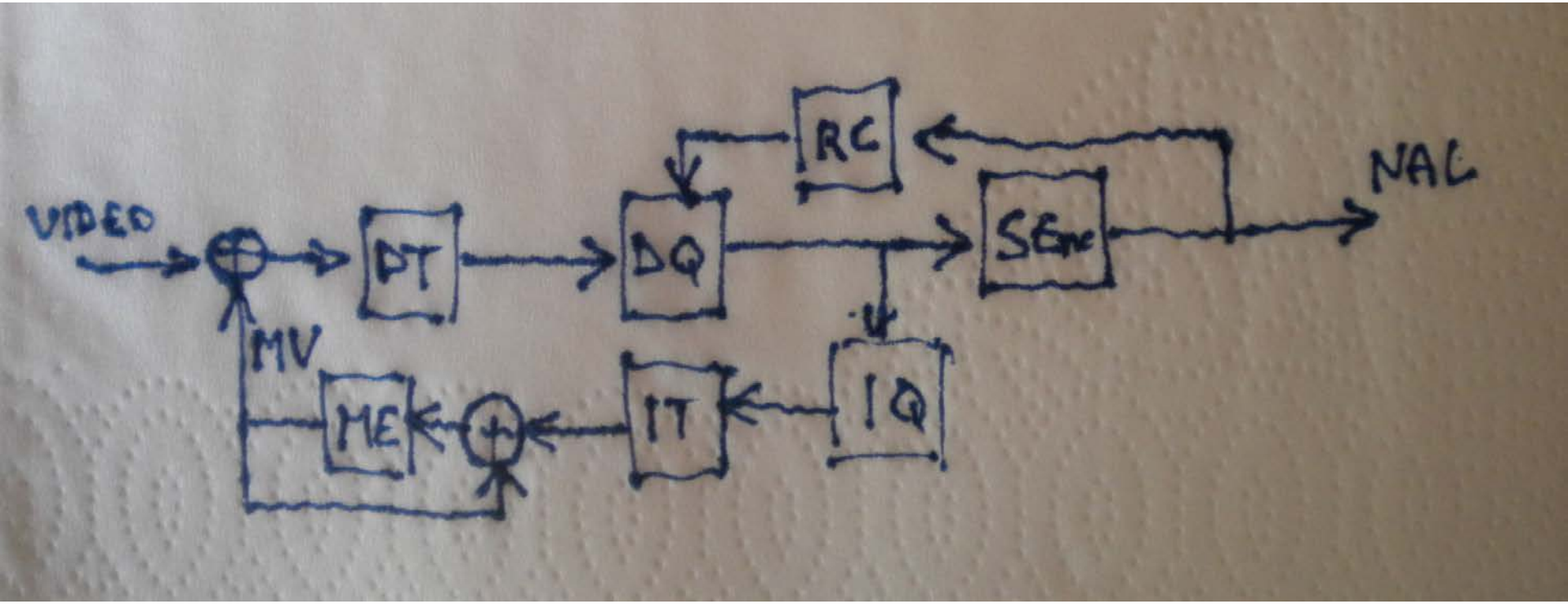
t2 = n;
n = t2 + 1;

Thread 2:

Thread race on shared state

Each thread is determinate.
Their composition is not.
In the domain of threads, determinacy is *not compositional.*

Dataflow programs: not exactly a new model …….



digital signal processing
media processing
video coding
image processing / analytics
audio
networking / packet processing
...

Data flow schemas (Dennis et al.)

DATA FLOW SCHEMAS

J.B. Dennis, J.B. Fosseen and J.P. Linderman

Massachusetts Institute of Technology
Cambridge, Mass., July 27,1972

INTRODUCTION

A data flow schema is a representation of the logical scheme of a program in a form in which the sequencing of function and predicate applications and the flow of values between applications are specified together. In a data flow schema, application of a function or predicate is free to proceed as soon as the values required for its application are available and the need for application of the function or predicate is determined. Since the availability of one computed value may simultaneously enable the application of several functions or predicates, concurrency of action is an inherent aspect of a data flow schema.

We present here some basic properties of a class of data flow schemas which

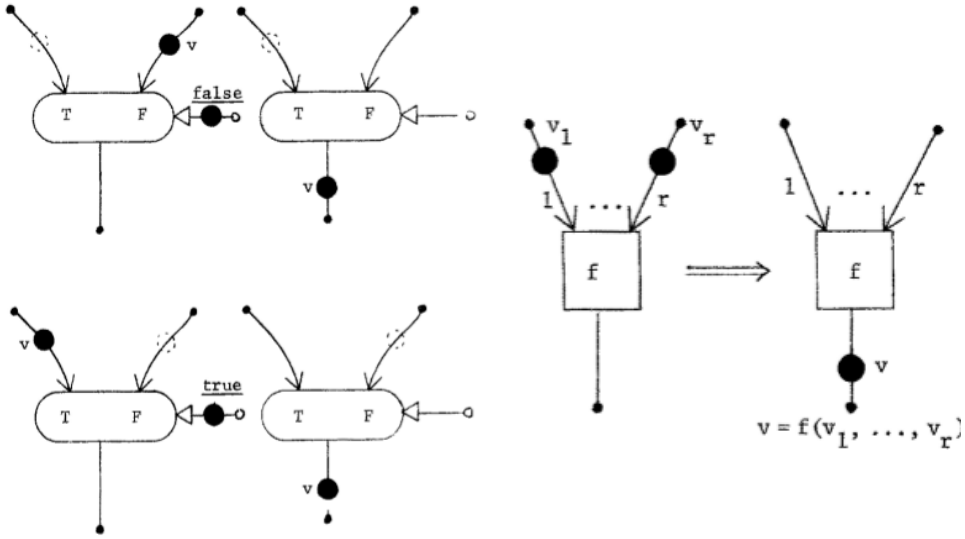


Figure 4. Firing rules for data flow schemas.

networks of (potentially concurrent) functions
triggered by the availability of input data
executing in a sequence of atomic transitions
described by firing rules

Kahn process networks (Kahn)

INFORMATION PROCESSING 74 – NORTH-HOLLAND PUBLISHING COMPANY (1974)

THE SEMANTICS OF A SIMPLE LANGUAGE FOR PARALLEL PROGRAMMING

Gilles KAHN

IRIA-Laboria, Domaine de Voluceau, 78150 Rocquencourt, France

and

Commissariat à l'Energie Atomique, France

In this paper, we describe a simple language for parallel programming. Its semantics is studied thoroughly. The desirable properties of this language and its deficiencies are exhibited by this theoretical study. Basic results on parallel program schemata are given. We hope in this way to make a case for a more formal (i.e. mathematical) approach to the design of languages for systems programming and the design of operating systems.

i) To every line e, of type D_e, associate a variable X_e ranging over D_e.

ii) If $X_1, X_2, \ldots, X_n$ are the variables associated to the input lines and $i_1, \ldots i_k$ are the sequences fed as inputs on the lines include the equations :

$$\begin{cases} X_1 = i_1 \\ \vdots \\ X_k = i_k \end{cases}$$

iii) For each node f, interpreted with the functions $f_1, \ldots f_p$, with input variables $X_1, \ldots, X_n$ output variables $X'_1, \ldots X'_p$ include p equations in $\sum_p$:

$$\begin{cases} X'_1 = f_1(X_1 \ldots X_n) \\ \vdots \\ X'_p = f_p(X_1 \ldots X_n) \end{cases}$$

```
Begin
(1) Integer channel X, Y, Z, T1, T2 ;
(2) Process f(integer  in U,V; integer out
    Begin integer I ; logical B ;
          B := true ;
          Repeat Begin
(4)         I := if B then wait(U) else w
(7)         print (I) ;
(5)         send I on W ;
            B := ¬B ;
            end ;
    End ;
  Process g(integer in U ; integer out V,
    Begin integer I ; logical B ;
      B := true ;
      Repeat Begin
        I := wait (U) ;
        if B then send I on V else send I
        B := ¬B ;
        End ;
    End ;
(3) Process h(integer in U;integer out V; i
    Begin integer I ;
      send INIT on V ;
      Repeat Begin
        I := wait(U) ;
        send I on V ;
        End ;
    End ;
  Comment : body of mainprogram ;
(6) f(Y,Z,X) par g(X,T1,T2) par h(T1,Y,0) par h(T2,Z,1
  End ;
```

Fig.1. Sample parallel program S.

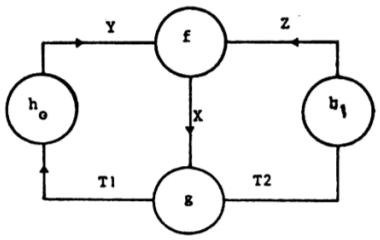


Fig.2. The schema P for the program S.

networks of stream functions
… described as communicating tasks with blocking reads
precise criterion for determinacy (prefix-monotonicity)
formal fixpoint semantics

Actors (Hewitt et al.)

MIT/LCS/TR -194

Actors and Continuous Functionals

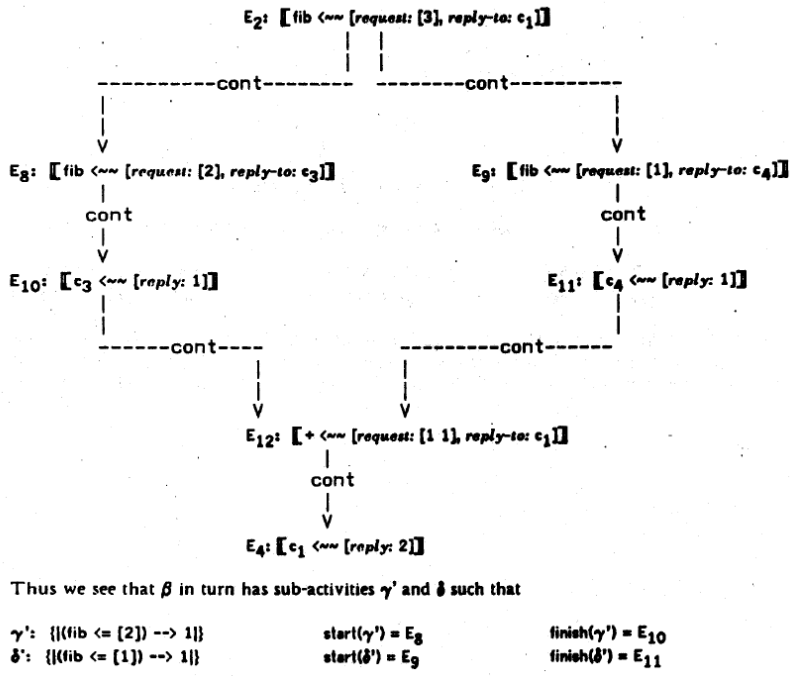
by

Carl Hewitt

and

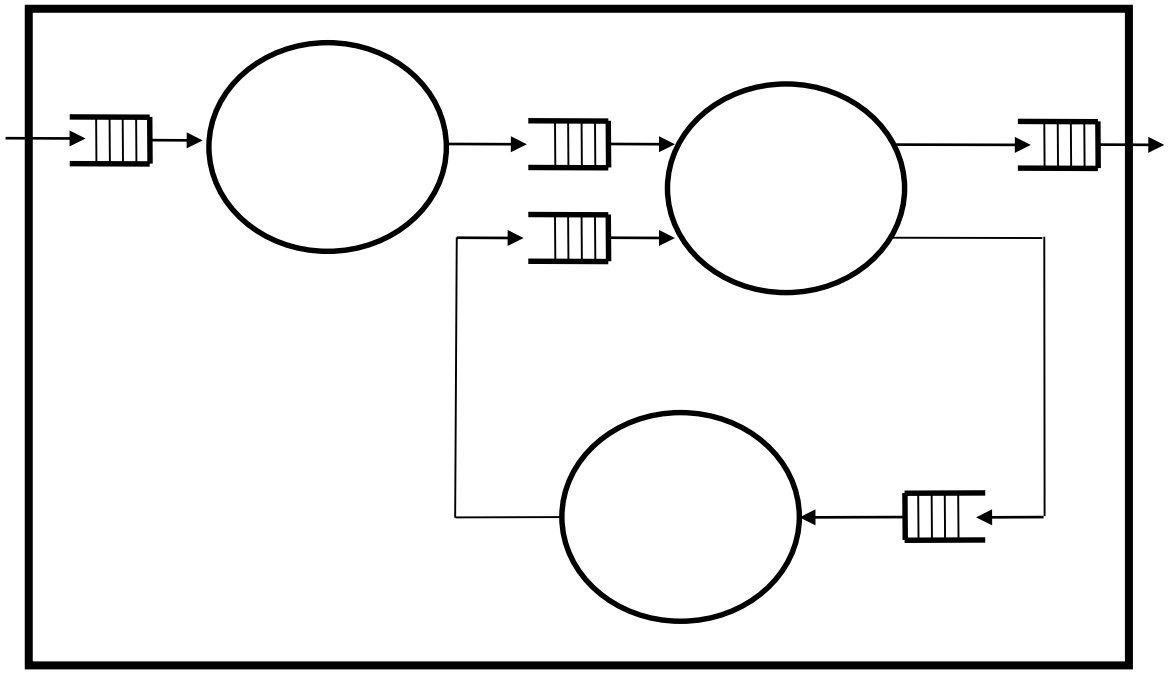
Henry Baker Jr.

December 1977



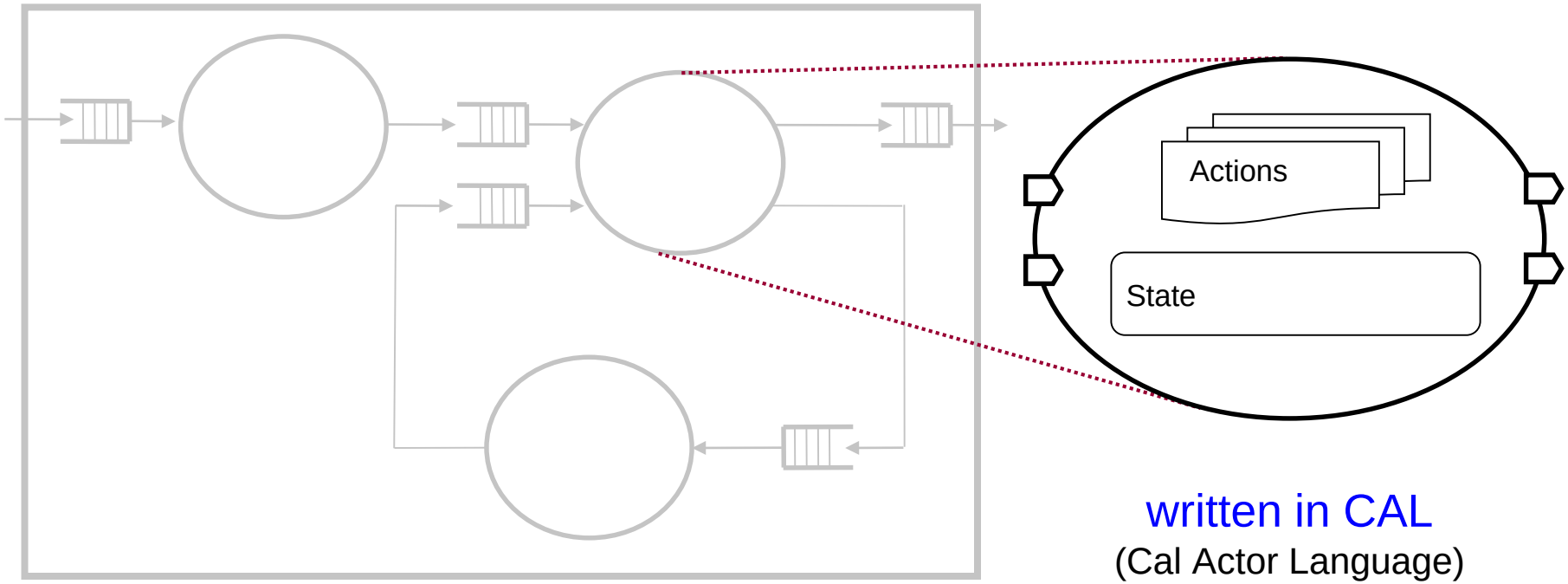
Stateful, active objects (actors)
Communicating via message passing
Non-determinism and partial orders
Functional actors with fixpoint semantics

More Recent Dataflow Concepts



- computational kernels (aka “actors”)
- directed point-to-point connections
  - lossless, order-preserving
  - asynchronous
  - conceptually unbounded (but...)
- communicating discrete data packets (“tokens”)

Actors (& Actions)



written in CAL
(Cal Actor Language)

- actors encapsulate state
- only interact via ports
- execution is a sequence of steps
- each step executes an “action”
- an action can: 1) read tokens 2) write tokens 3) modify actor state

CAL a formal language for dataflow: origin and features

The CAL language is a **data flow oriented** language that has been specified and developed as subproject of the Ptolemy project at the University of California at Berkeley.

The initial **CAL** language specification has been released in December 2003.
A ISO standard version of CAL has been released in Oct 2009.

CAL is a textual language that is used to define the functionality of dataflow components called “actors”.

CAL is implemented in at least three publicly available modeling and simulation environments, **Ptolemy II**, **OpenDF and Orcc**.

An **actor** is a modular component that **encapsulates its own state**, no other actor has access to it, and **nothing other actors can do can modify the state of an actor**.

The only interaction between actors is through **FIFO channels** connecting **“output ports” to “input ports,”** which they use to send and receive **“tokens”.**

This **strong encapsulation** leads to loosely coupled systems with very manageable and controllable actor interfaces.

The **modularity** of an actor assembly fosters **concurrent development**, it facilitates **maintainability and understandability** and makes systems constructed in this way more robust and **easier to modify**.

CAL - A crash course

1. consumption & production of tokens

```
actor Add () A, B ==> X:

    action A: [a], B: [b] ==> X: [a + b] end
end
```

2. state

```
actor Sum () A ==> X:

    s := 0;

    action A: [a] ==> X: [s]
    do
        s := s + a;
    end
end
```

3. multiple actions

```
actor Merge () A, B ==> X:

    action A: [v] ==> X: [v] end
    action B: [v] ==> X: [v] end
end
```

4. guards

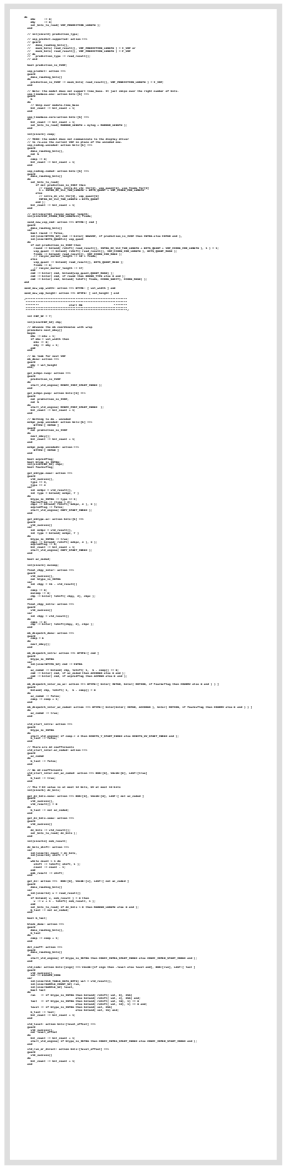
```
actor Split () A ==> P, N:

    action A: [v] ==> P: [v]
    guard v >= 0 end

    action A: [v] ==> N: [-v]
    guard v < 0 end
end
```

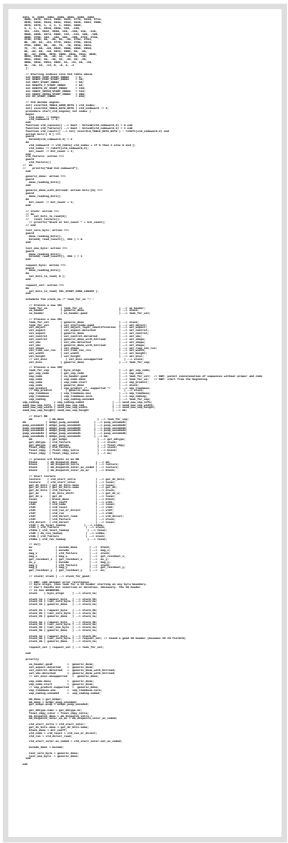
5. priorities

```
actor BiasedMerge () A, B ==> X:

    CopyA: action A: [v] ==> X: [v] end
    CopyB: action B: [v] ==> X: [v] end

    priority CopyA > CopyB; end
end
```

Actors: big and small

Compare
23 lines
(without header comments)

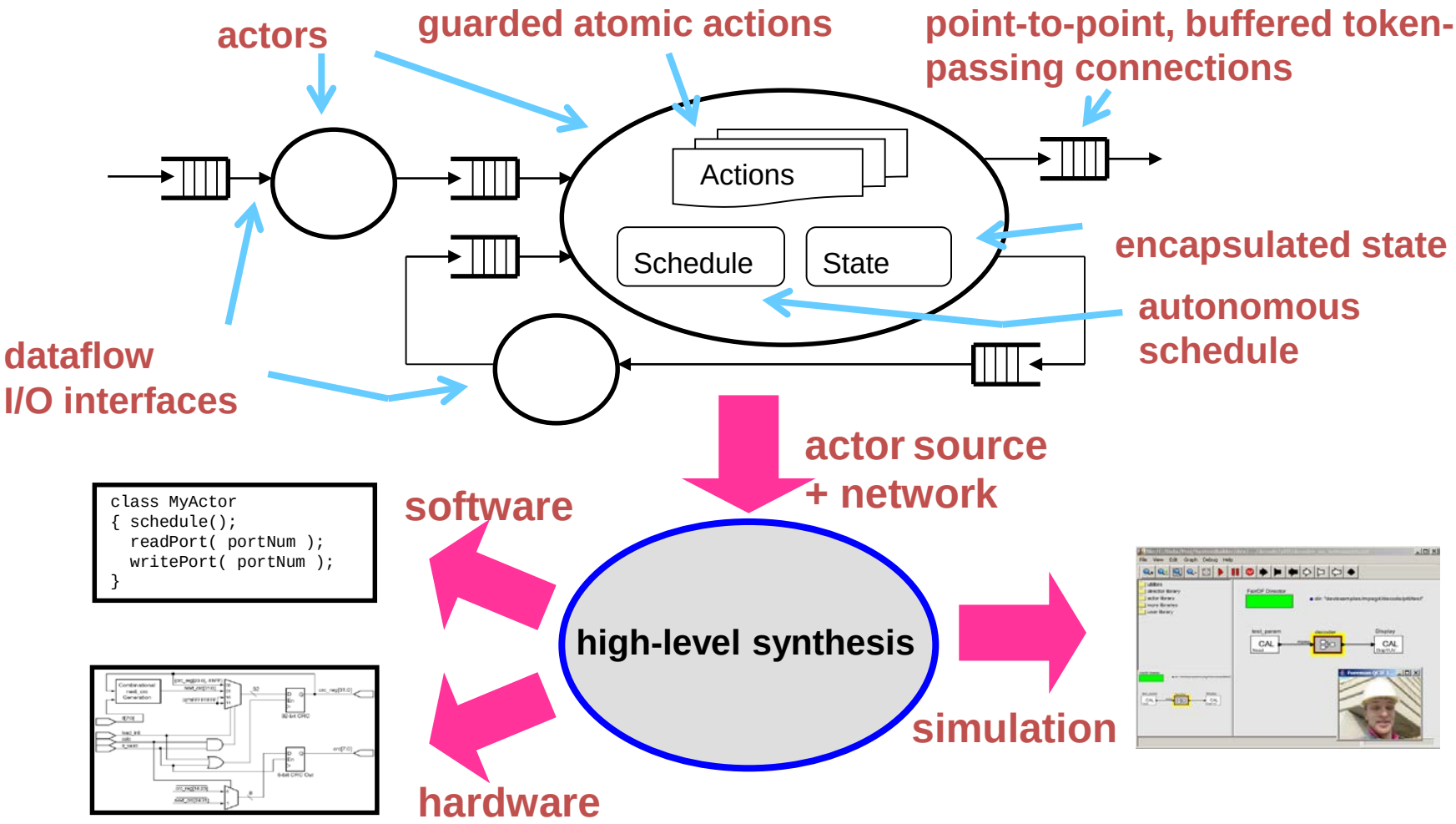




ParseHeaders
1320 lines
(without header comments)

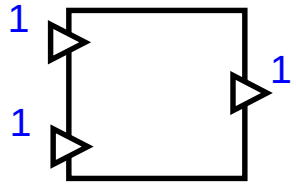
http://opendf.svn.sourceforge.net/viewvc/opendf/models/MPEG4_SP_Decoder/

Actor/Dataflow Programming Model

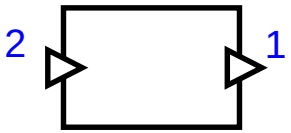


Static scheduling: token rates

```
actor Add () Input1, Input2 ==> Output:

    action [a], [b] ==> [a + b] end
end
```

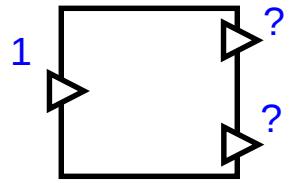



```
actor AddSeq () Input ==> Output:

    action [a, b] ==> [a + b] end
end
```

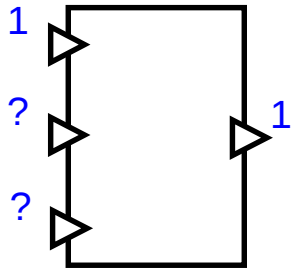


Static scheduling: token rates

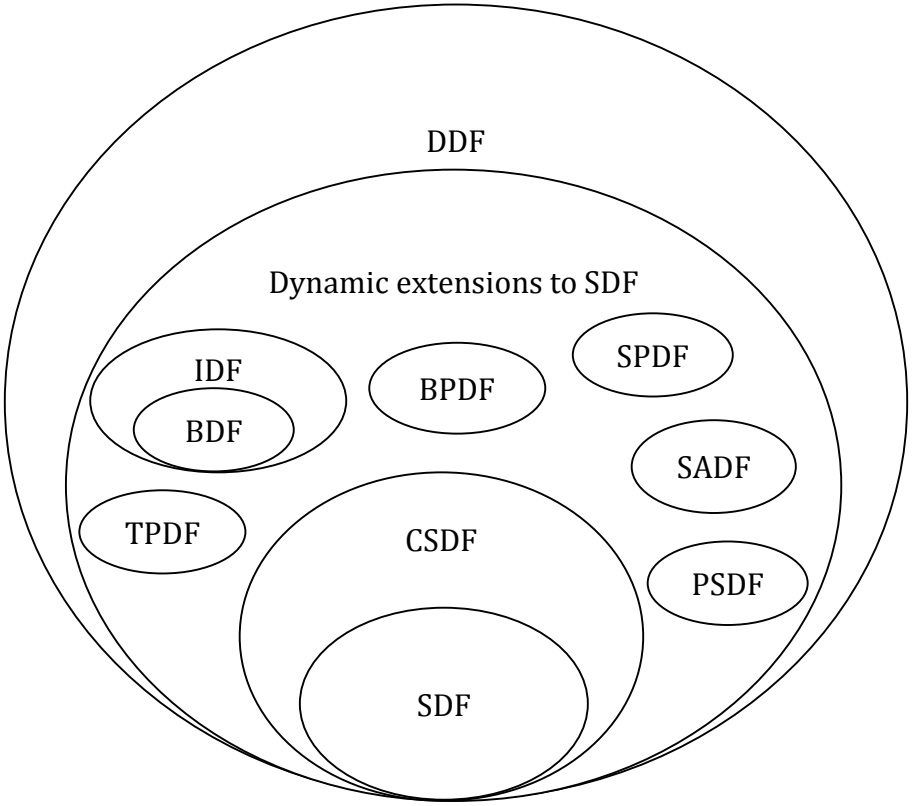
```
actor Split () Input ==> Pos, Neg:

    action [a] ==> Pos: [a]
    guard a >= 0 end

    action [a] ==> Neg: [a]
    guard a < 0 end
end
```




```
actor Multiplex () S, A, B ==> Output:

    action S: [sel], A: [v] ==> [v]
    guard sel end

    action S: [sel], B: [v] ==> [v]
    guard not sel end
end
```

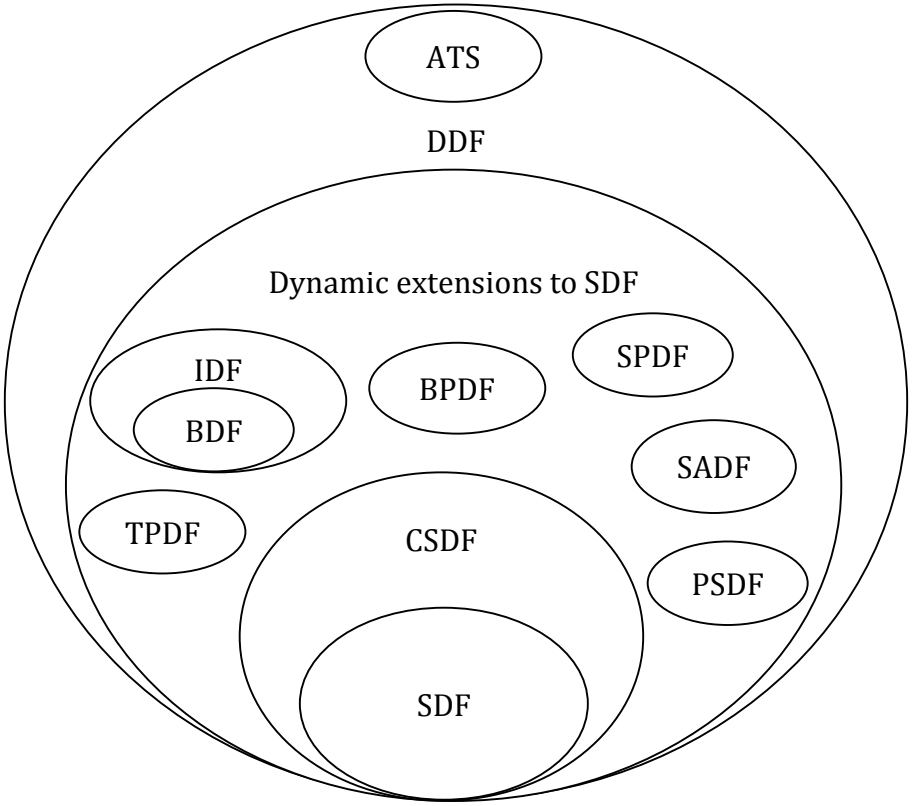


Dataflow programs: MoCs



- SDF:
  static, fixed token exchange rates, compile-time analysis
- CSDF:
  extension of SDF with the notion of state, the same compile-time properties
- Dynamic extensions:
  different aspects of dynamic behavior, run-time changes of the topology or token exchange rates
- DDF:
  firing rules and firing functions can be data-dependent, run-time analysis only

Dataflow programs: MoCs



- SDF:
  static, fixed token exchange rates, compile-time analysis
- CSDF:
  extension of SDF with the notion of state, the same compile-time properties
- Dynamic extensions:
  different aspects of dynamic behavior, run-time changes of the topology or token exchange rates
- DDF:
  firing rules and firing functions can be data-dependent, run-time analysis only

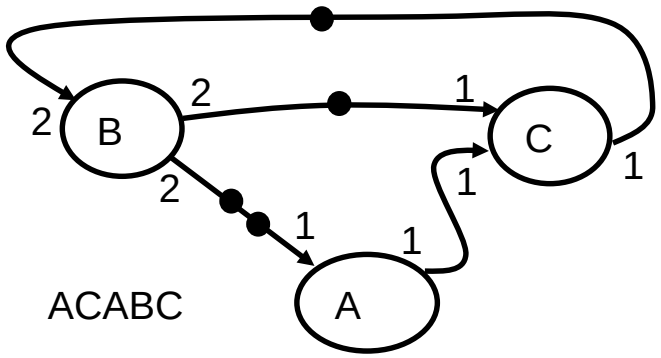
Static scheduling conundrum

Synchronous Dataflow (SDF)	Dataflow
static token rates	arbitrary token rates, may change over time, be data dependent
static schedule, static buffer analysis, guaranteed deadlock-free	no static schedule, no hard buffer-bounds at compile time, deadlock undecidable
efficient implementation due to compile-time analysis, low or zero runtime overhead	runtime scheduling, requires runtime decision making

Static scheduling

Synchronous Dataflow (SDF)	Dataflow w/ CAL	Dataflow
static token rates	arbitrary token rates, may change over time, be data dependent	arbitrary token rates, may change over time, be data dependent
static schedule, static buffer analysis, guaranteed deadlock-free	**partial static scheduling**: statically schedulable when possible	no static schedule, no hard buffer-bounds at compile time, deadlock undecidable
efficient implementation due to compile-time analysis, low or zero runtime overhead	adaptive implementation: use compile-time information, make dynamic choices at runtime	runtime scheduling, requires runtime decision making

Static dataflow scheduling

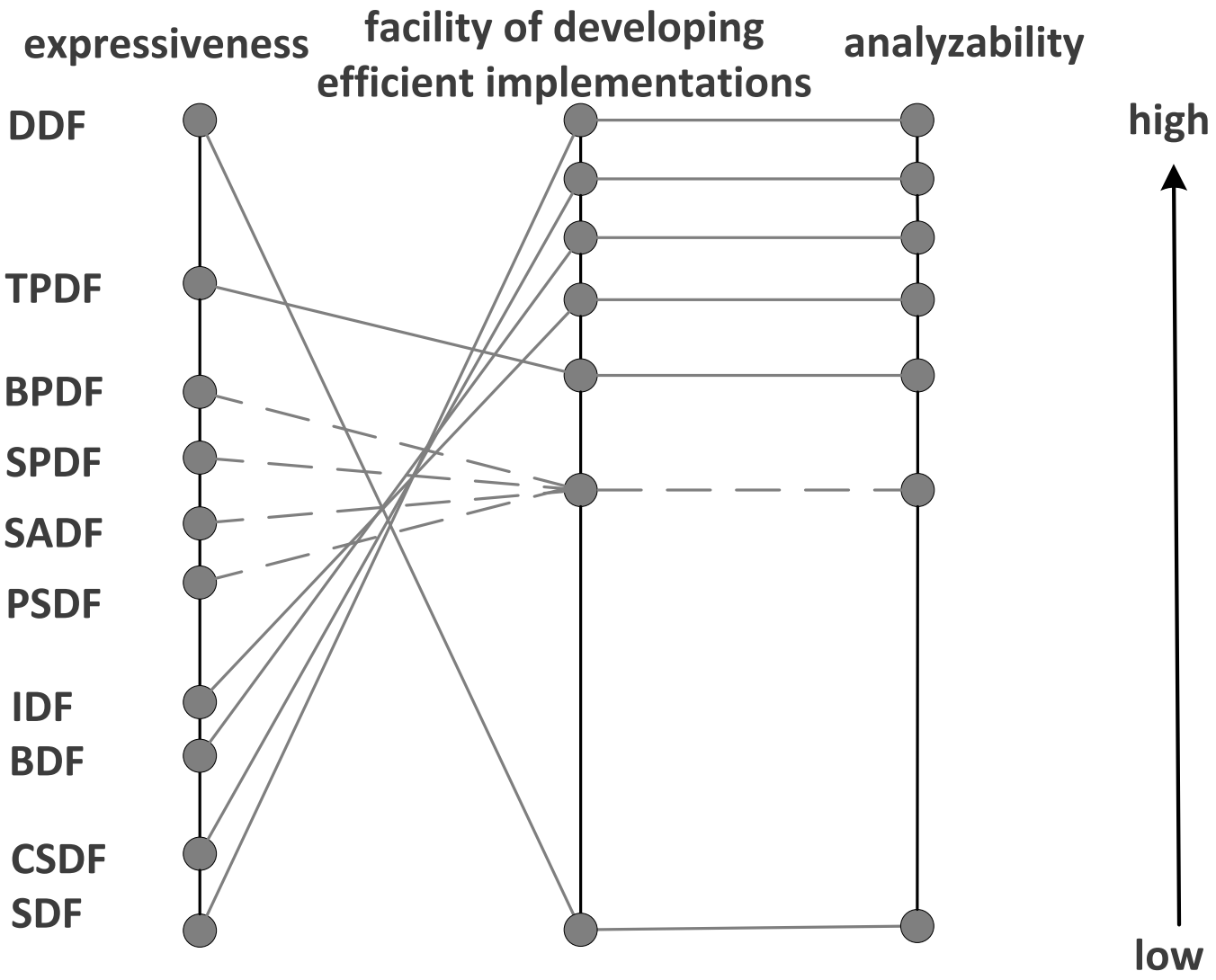


statically schedulable networks desirable
efficient software implementations
no deadlocks, yet small buffers
reduced synchronization overhead (in hardware)
.

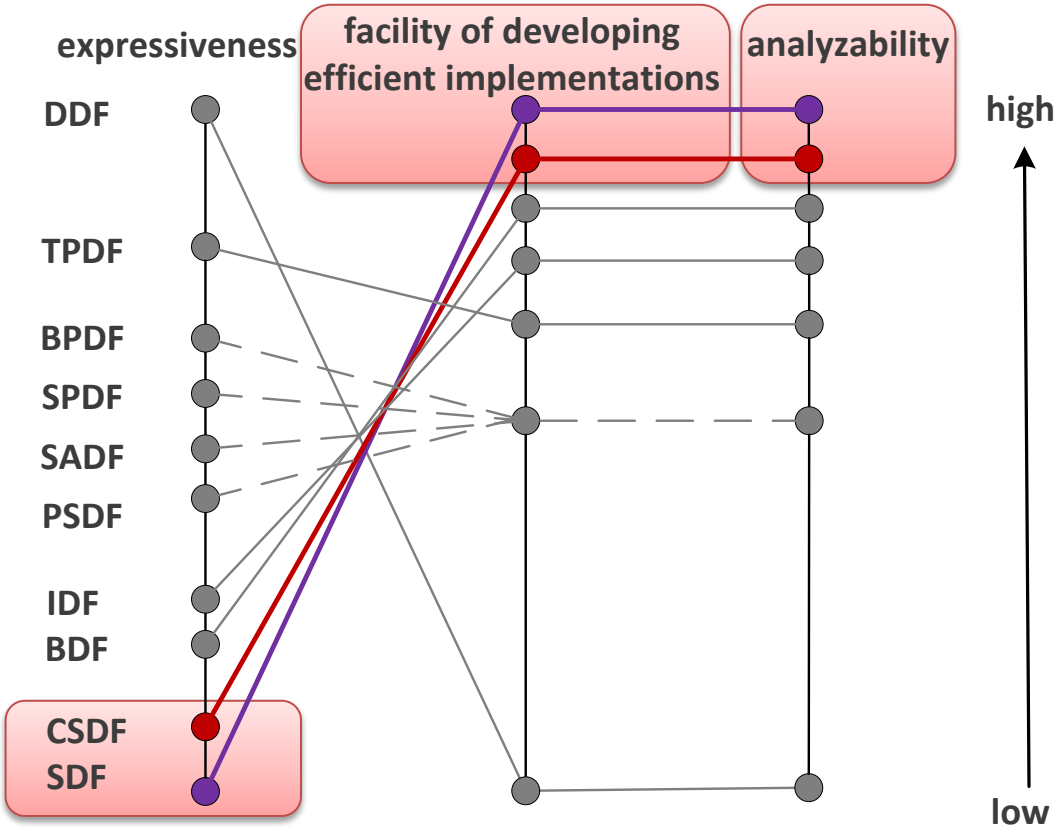
Real-world dataflow networks are seldom completely statically schedulable.

Use internal actor structure to identify analyzable ***regions.***

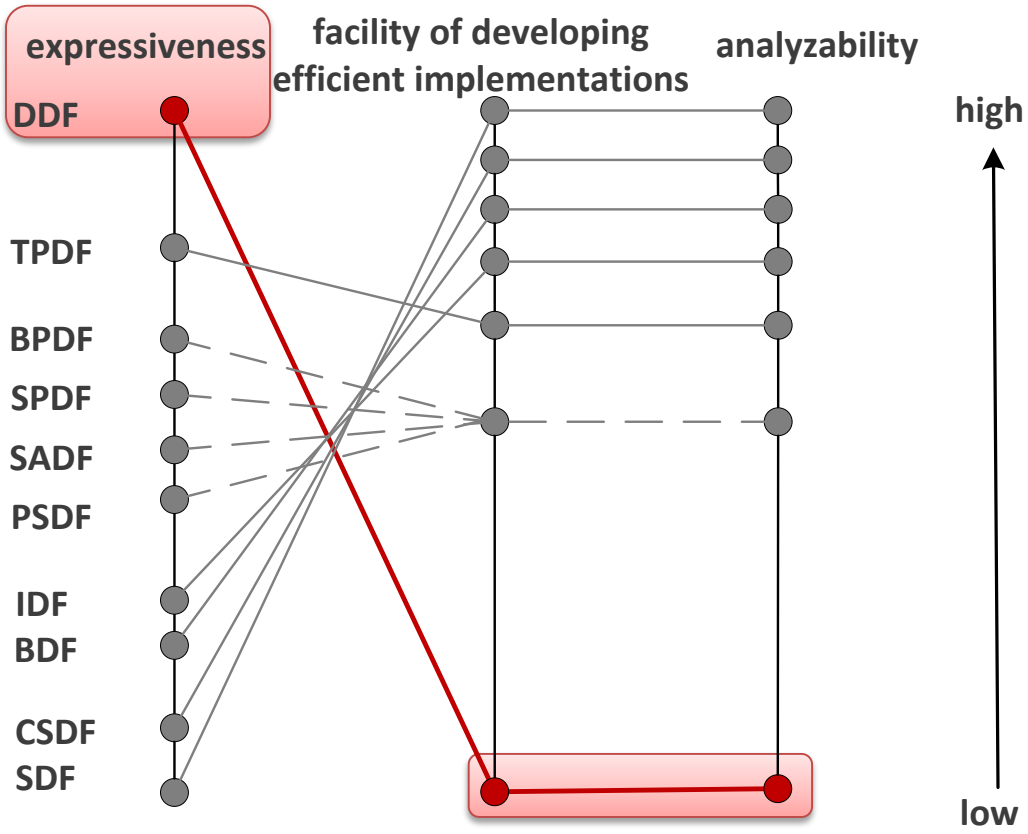
State-of-the-art: (dataflow) parallel implementations



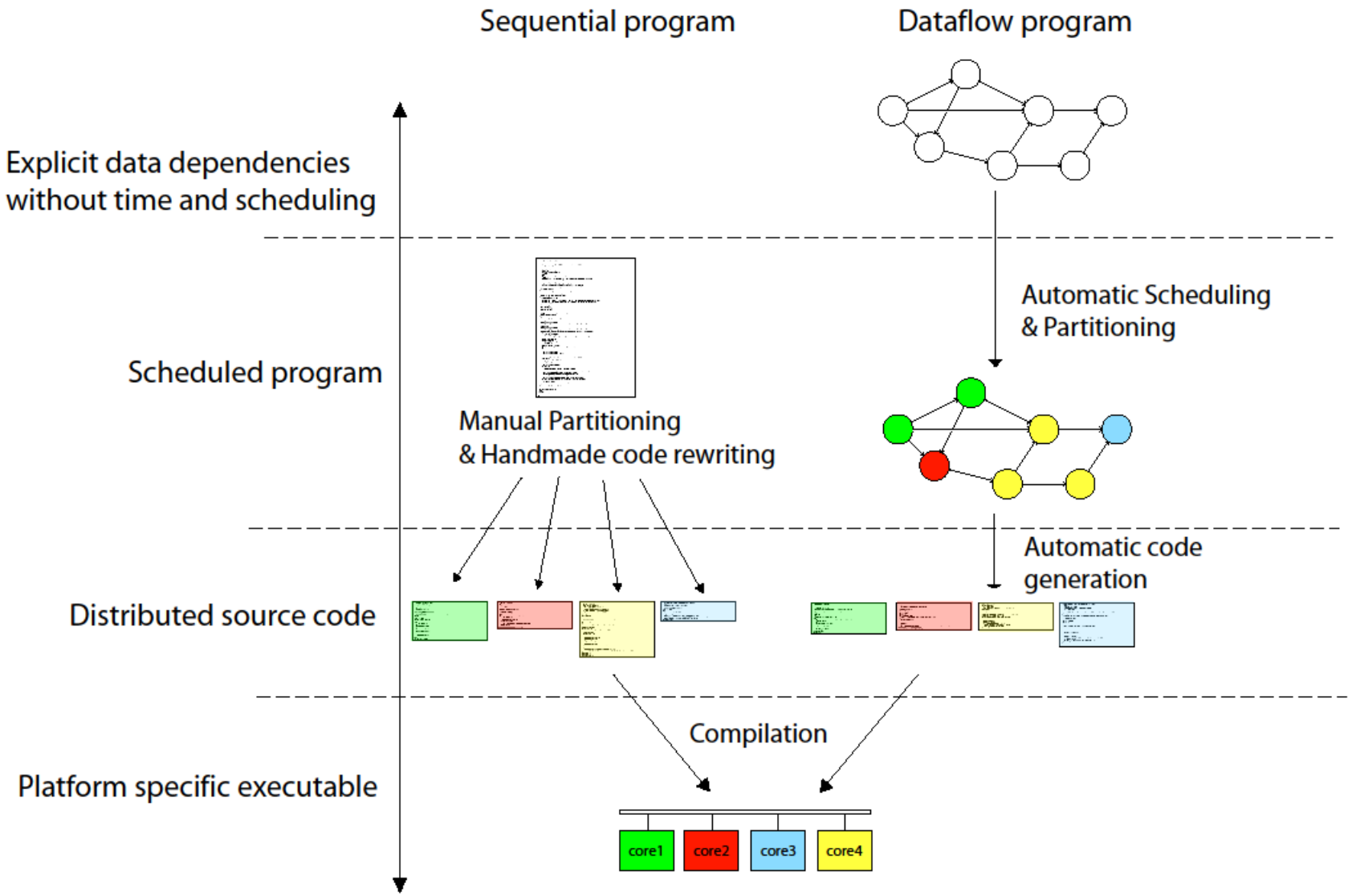
State-of-the-art: (dataflow) parallel implementations



State-of-the-art: (dataflow) parallel implementations



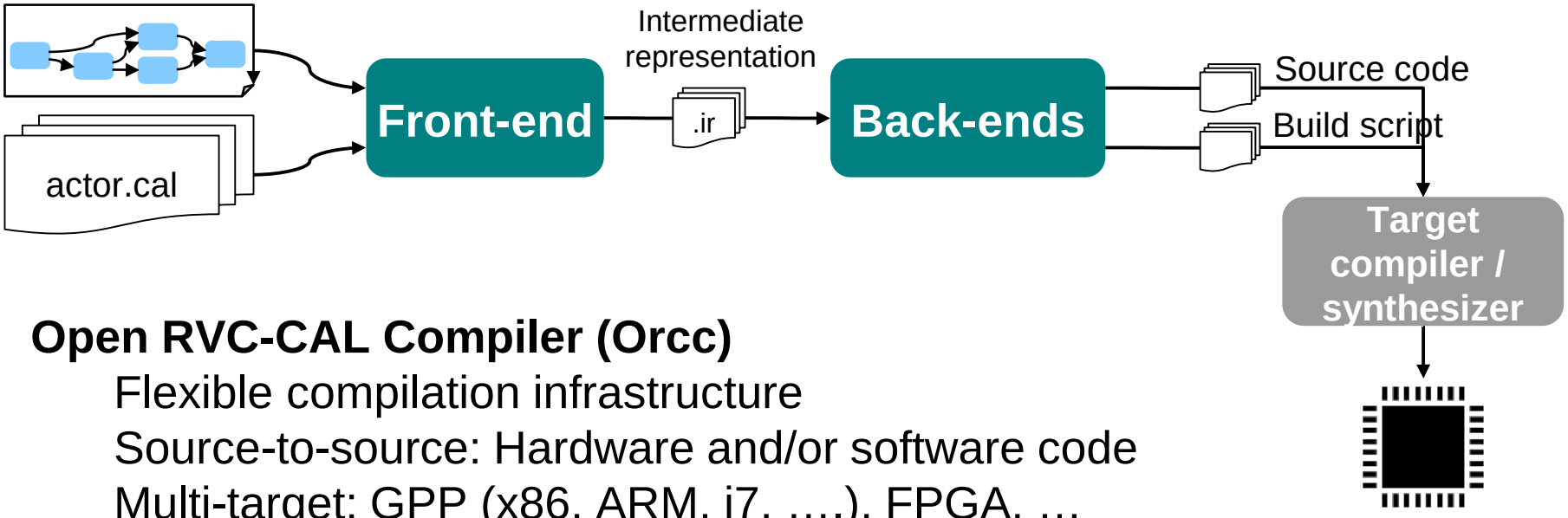
Sequential versus Dataflow



CAL MPEG-4 SP DECODER

Design Unit	MPEG-4 SP CAL lines of code	MPEG-4 Part 7 (Optimized reference SW) lines of C code	MPEG-4 SP (Optimized Proprietary SW) lines of C code
Parser (2 actors)	1585		
AC/DC Reconstruction (3 actors)	468		
2D-IDCT (2 actors)	52		
Motion Compensation (4 actors)	631		
TOTALS	2736	31452 (.src + .h files)	6115 with display control and other tools (.src + .h files)

Dataflow programing framework



Open RVC-CAL Compiler (Orcc)

- Flexible compilation infrastructure
- Source-to-source: Hardware and/or software code
- Multi-target: GPP (x86, ARM, i7, ….), FPGA, …

Cal Actor Language: Tool infrastructure

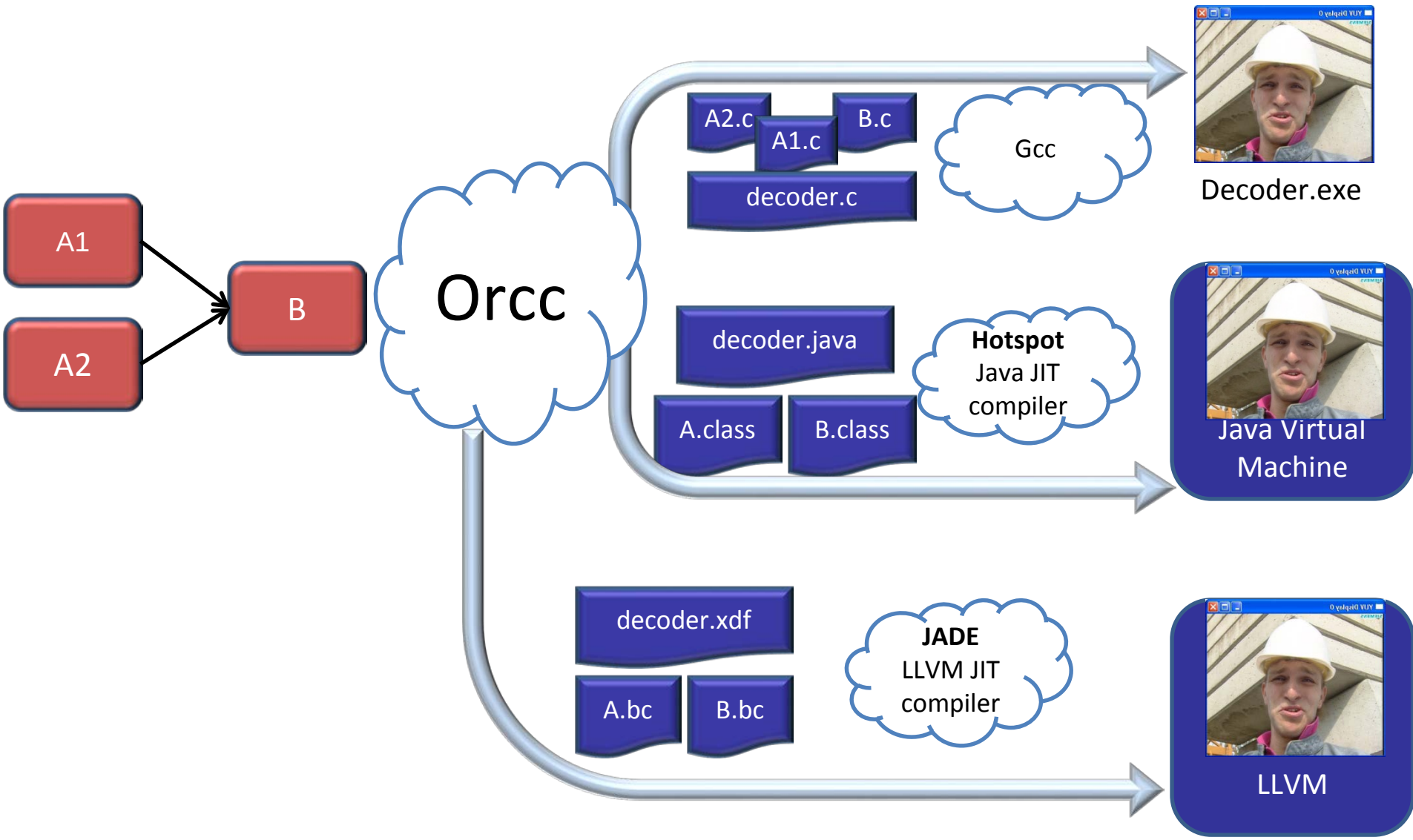
Frontend

Intermediate representation simpler than the direct AST

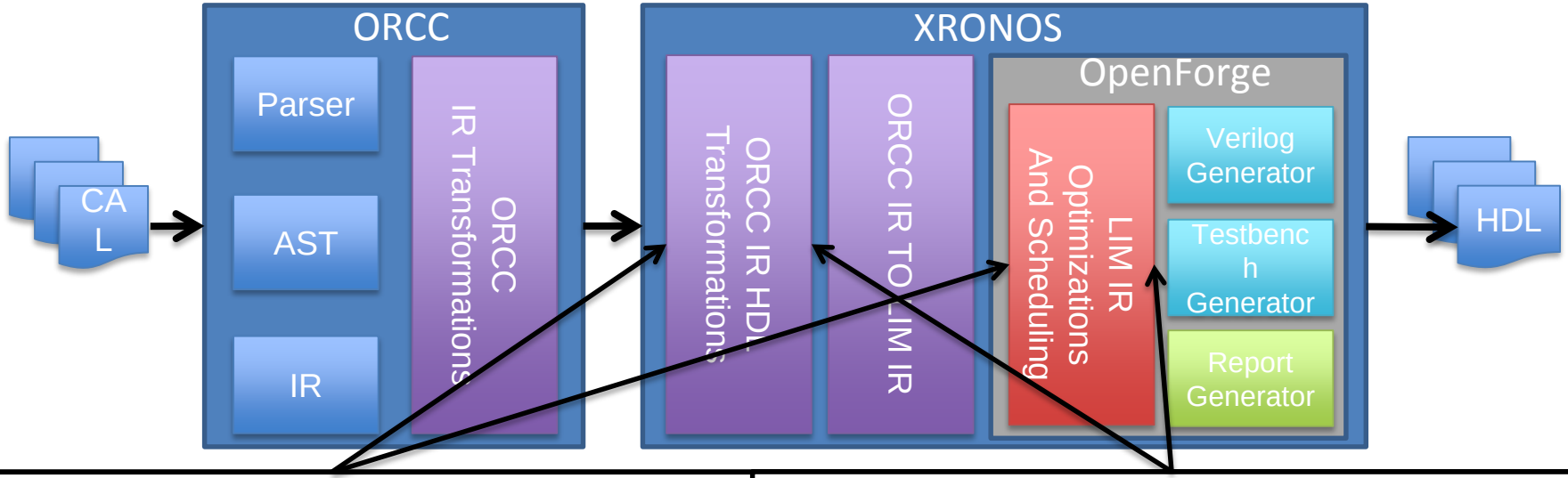
Backends Transformations

Classification, SSA transformation, 3AC, repeat transformation…
Interpreter, Profiling analysis (EPFL)
Quasi static analysis (Oulu)
Code generation: actor instantiation
C / C++
Verilog (Xilinx)
C – Vivado HLS
TTA (network + actor in pure LLVM code generation) (IRISA)
Promela backend (Abo)
Code generation: actor as class
LLVM (LLVM + metadata)

Code generation with Orcc



CAL HLS Tool Flow based on Xronos



- The graph shows data dependencies:
  - Operations that must complete before others can begin
  - e.g (a*b) must be complete before (a*b)+c

- The graph shows control dependencies:
  - If/else constructs will turn into mux operators
  - Loops constructs to a set of @always

Xronos offers a full support of the RVC-CAL programming language

Xronos HLS – OpenForge Optimizations

Memory Optimizations:

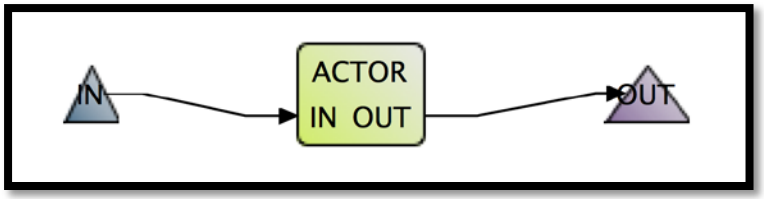
- **Dual port memory conversion**, one clock for read or write
- **Memory ROM reduction**, reduces the allocated size of each element
- **Memory merging**, reduces size by grouping memories
- **Memory resizing,** resizes the length of a memory to a power of 2
- **Read only field reducing**, ROM to constant
- **ROM replicating**, reduces the clock access by replicating the ROM

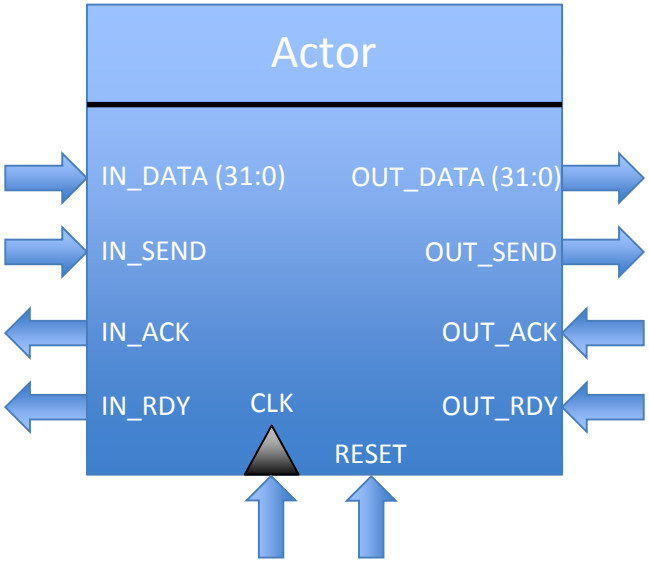
Control Flow Optimization:

- **Loop unrolling**, unrolls loops
- **Pipelining**, inserting registers to break the combinatorial path into multiple stages

Xronos HLS – Actors I/Os

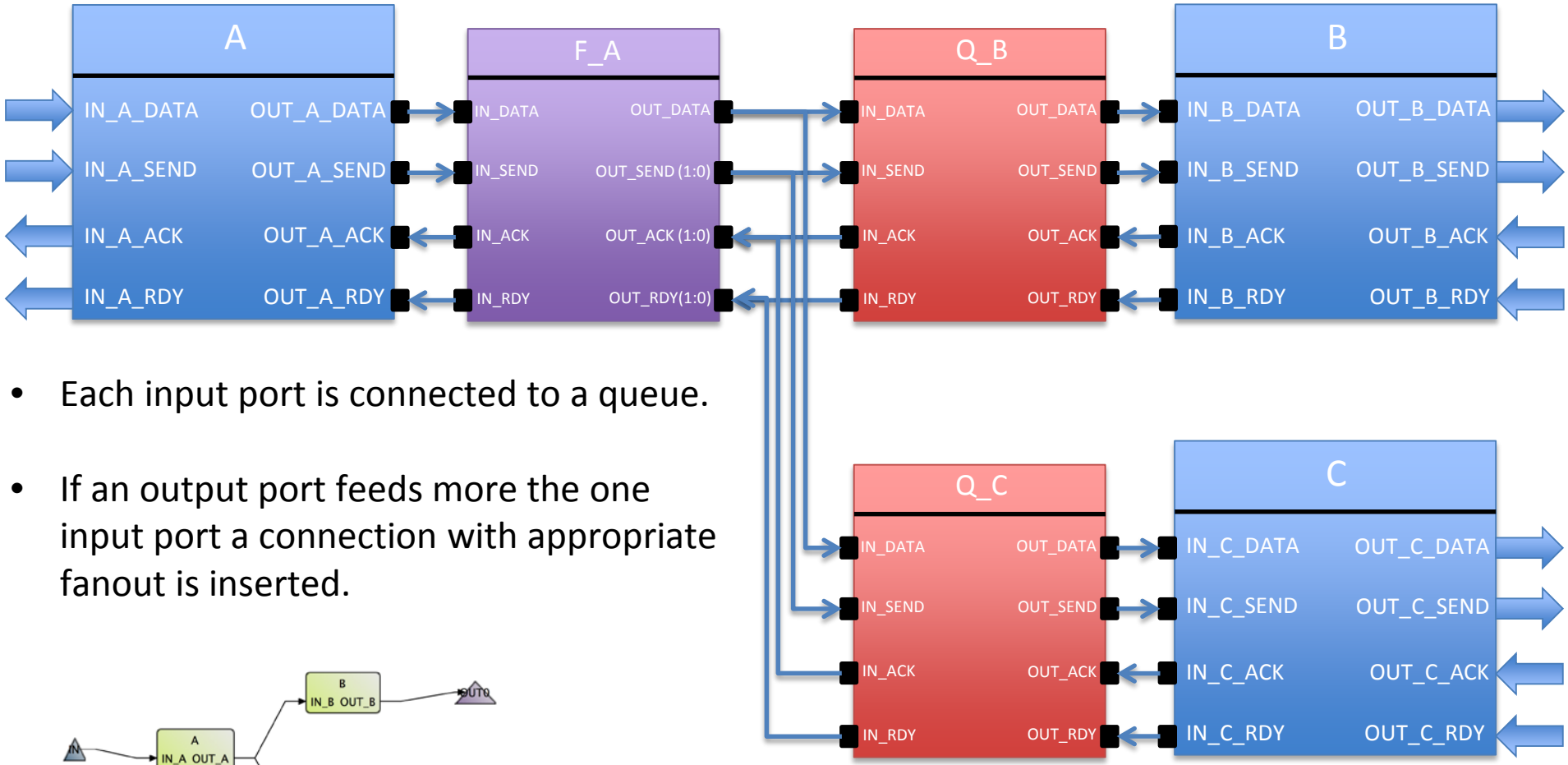
```
actor ACTOR() int IN ==> int OUT:
    ...
end
```





Each input and output of an actor is transformed into a double handshake interface

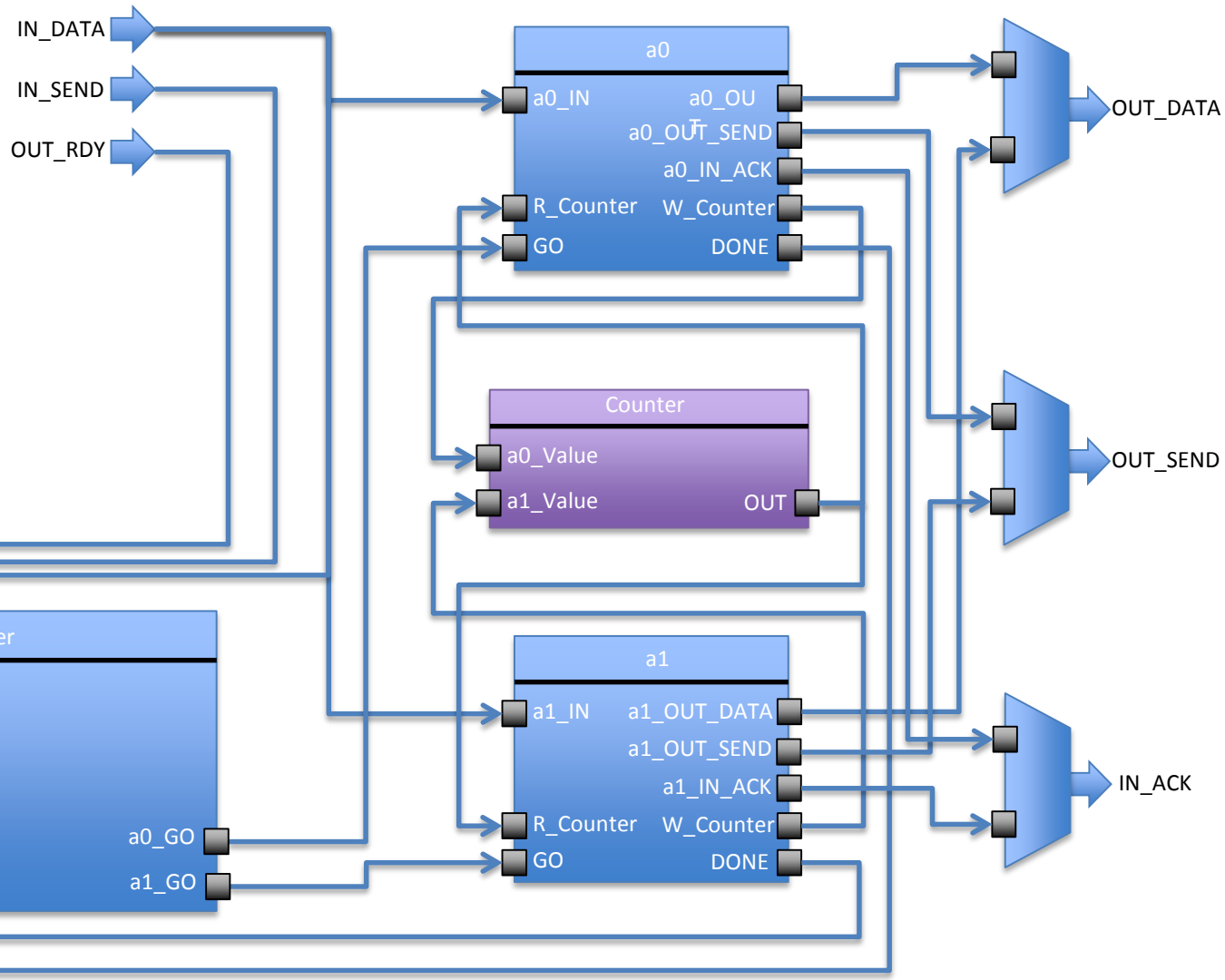
Xronos HLS – Dataflow Application



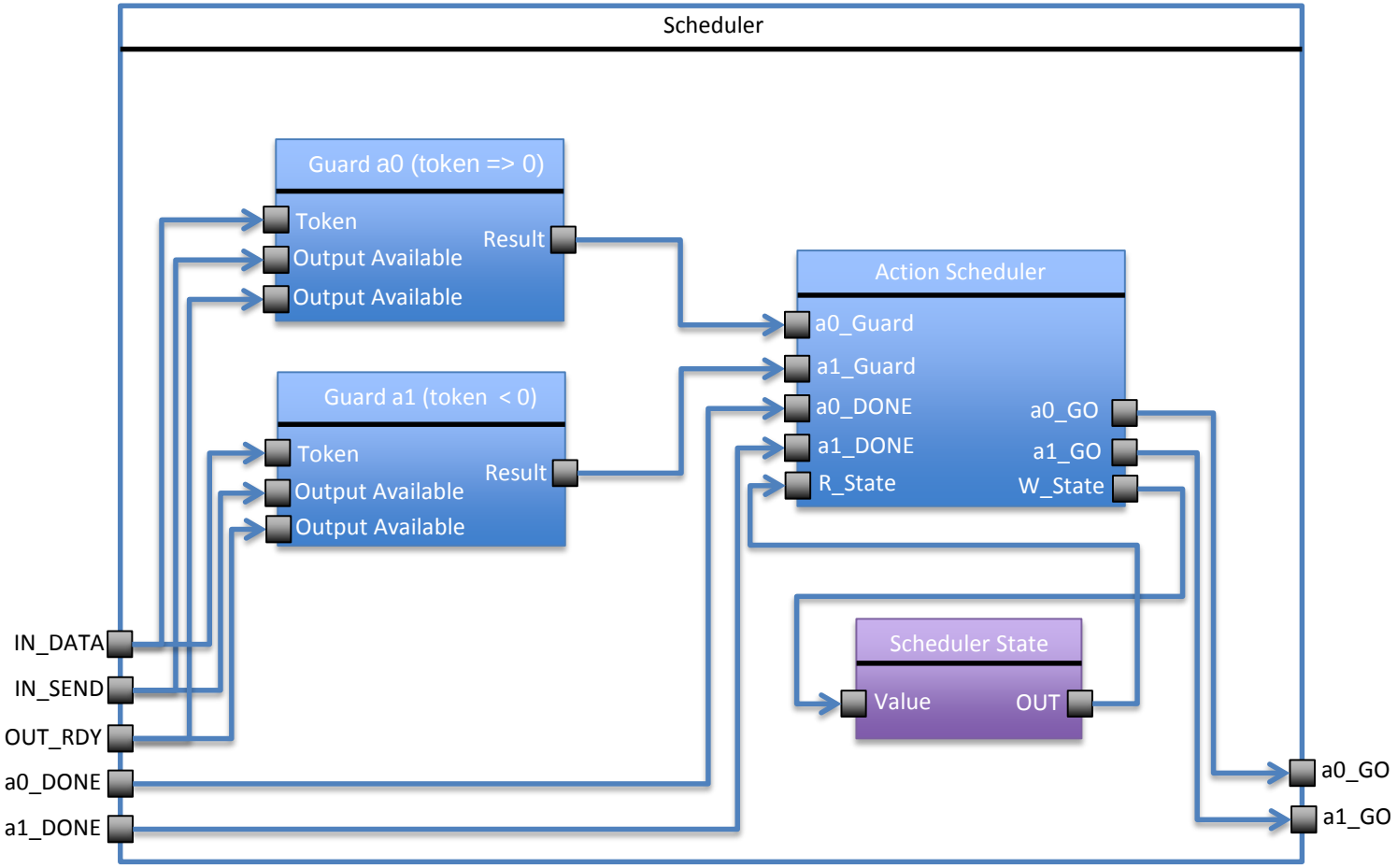
- Each input port is connected to a queue.
- If an output port feeds more the one input port a connection with appropriate fanout is inserted.

Xronos HLS – Inside an Actor

```
actor CondCounter() int IN ==> int OUT:
        int Counter := 0;
        a0: action In:[token] ==> OUT:[Counter]
        guard token >= 0
        do
                Counter := Counter + 1;
        end

        a1: action In:[token] ==> OUT:[Counter]
        guard token < 0
        do
                Counter := Counter + 2;
        end
        ...
        schedule fsm s0:
                s0 (a0) → s1;
                s1 (a1) → s2;
                ...
        end
end
```



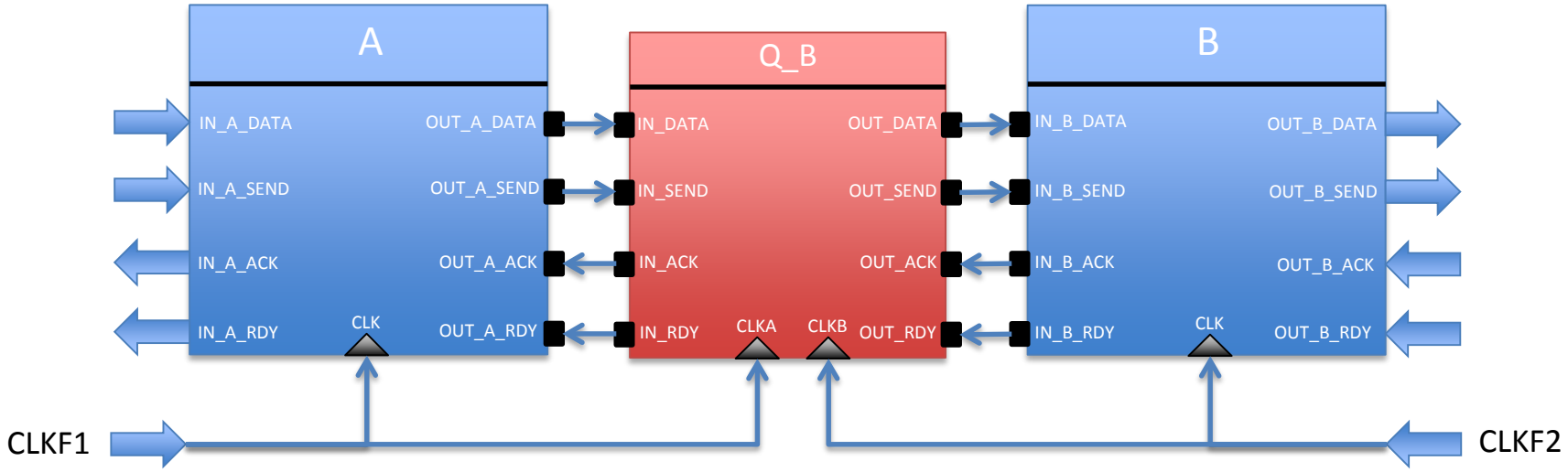
Xronos HLS – Inside the Actor scheduler



Xronos HLS – Clock Domains

Multi-Clock Domain Support

- The user may attribute (graphically or by a file) a different clock domain for each actor
- A Multi-Clock queue is inserted
- **Power reduction with multiple clock domains and clock gating techniques**



Xronos HLS – Testbench Generation

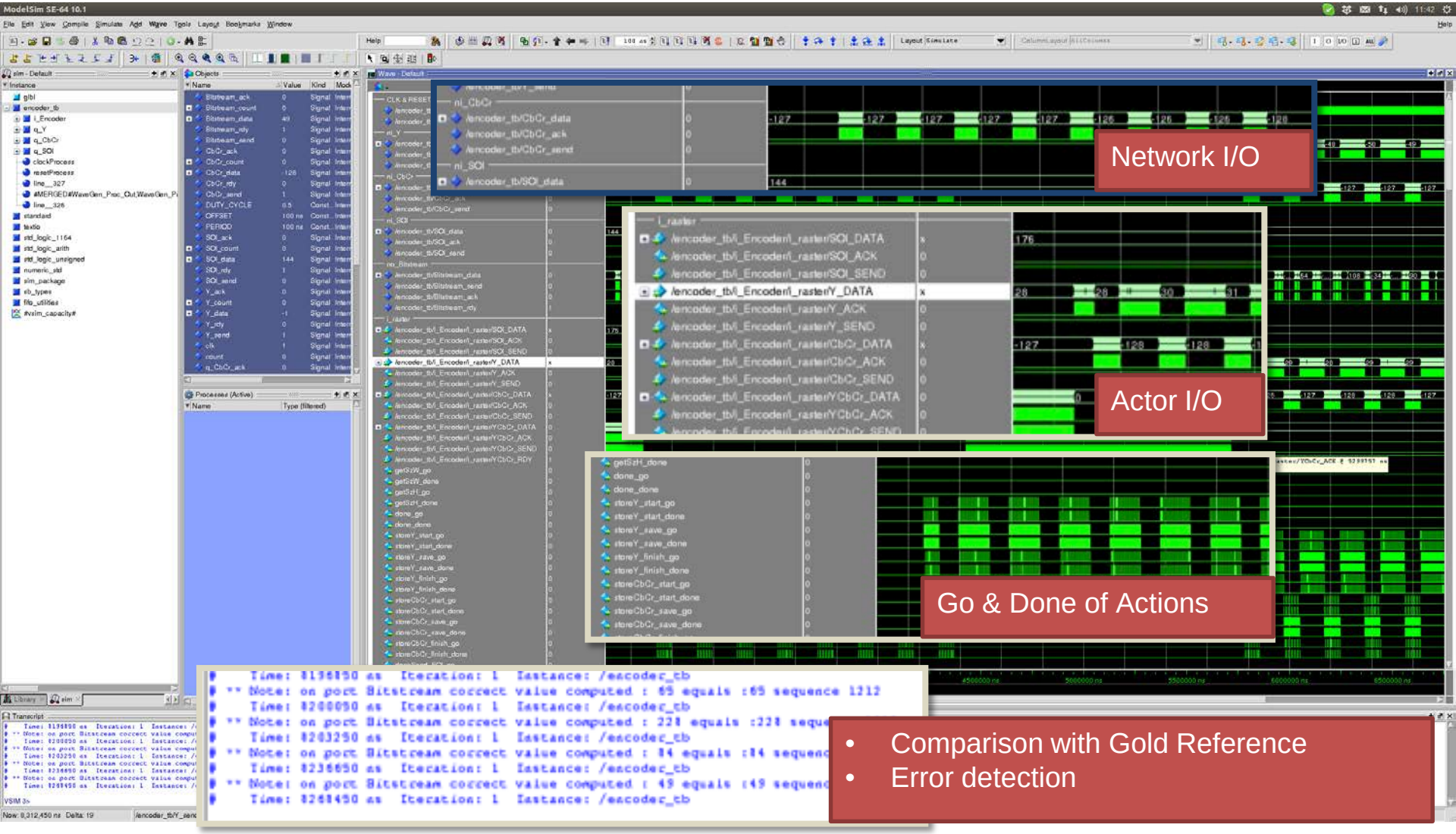
Automatic testbench generation:

- Only the input and output tokens, generated by the CAL simulator or a synthesized C code is needed
- Error detection, simulation stops and shows in which token the error has occurred
- VHDL testbench for each actor and network

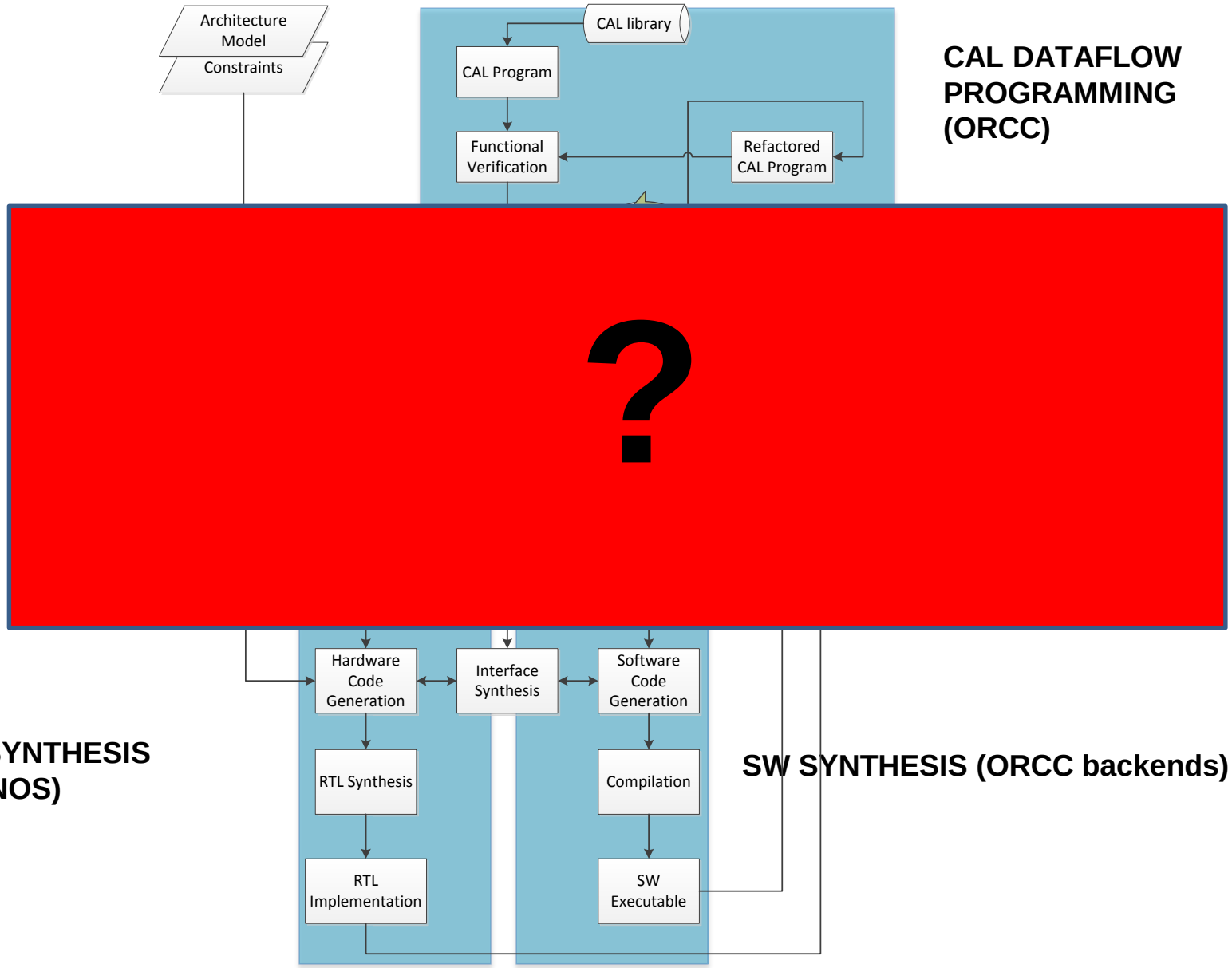
Input files generated for “Modelsim”:

- TCL file for each actor and network that:
  - Adds all inputs and outputs of the actor or network (in case of actor simulation or network simulation)
  - Adds the “Go” and “Done” signals for each action

Xronos HLS – Testbench Generation



CAL DATAFLOW THE DESIGN FLOW

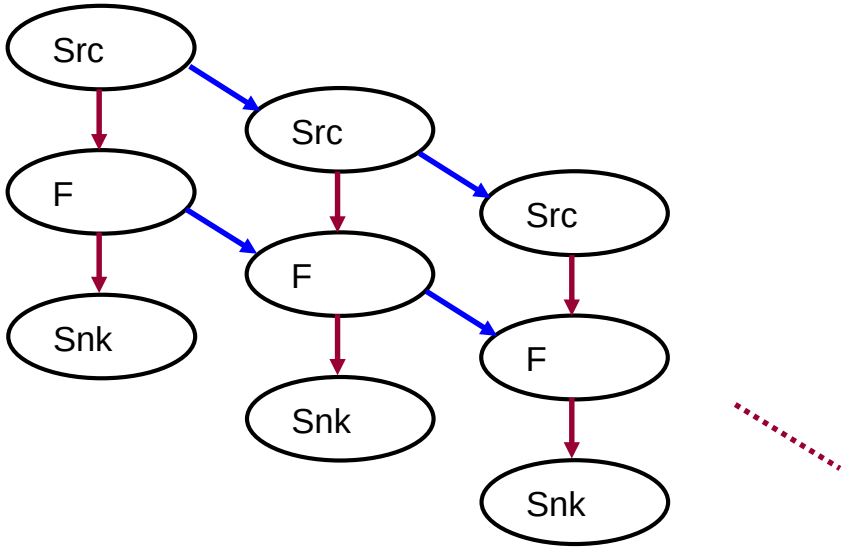


The structure of a computation

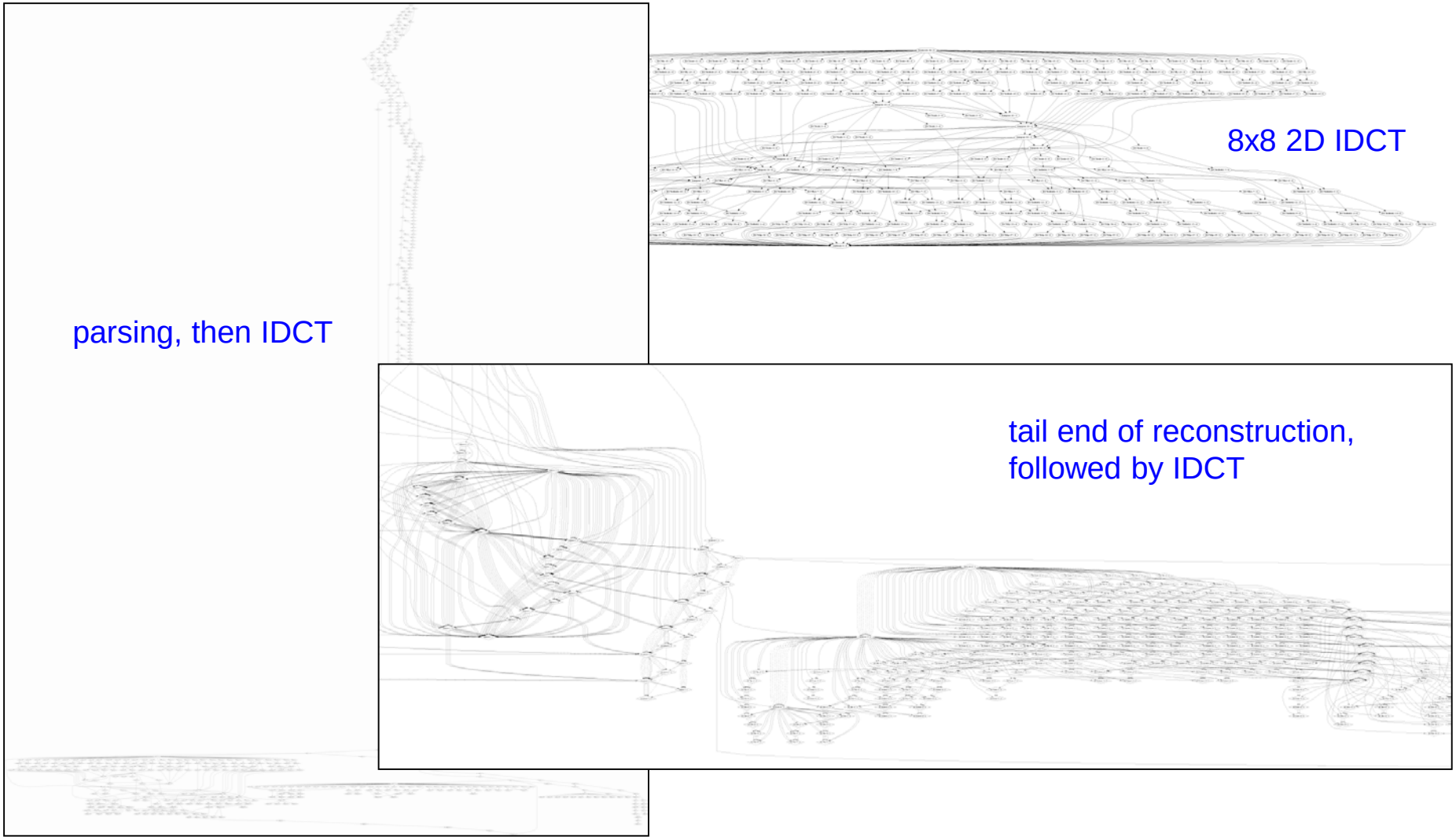
```
actor Src ()  ==> X:

    s := 0;

    action ==> X: [s]
    do
        s := h(s);
    end
end
```

```
actor F () A ==> X:

    x := 0;

    action A: [a] ==> X: [f(x)]
    do
        x := g(a, x);
    end
end
```

```
actor Snk () A ==> X:

    action A: [a] ==> end
end
```



Profiling: traces of a decoder

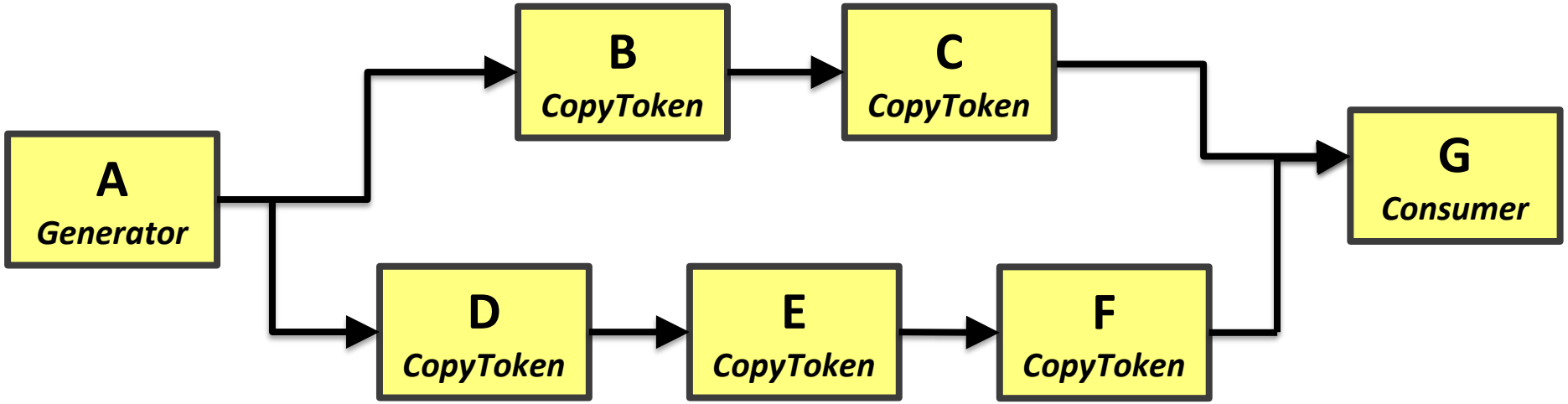


Execution Trace Graph generation

```
Actor Generator ==> O
counter = 0;
gen: action ==> O[counter]
   guard counter < 3
   do
      counter = counter + 1;
   end
end
```

```
Actor CopyToken X ==> Z
counter = 0;
copy: action X[x] ==> Z[z]
   do
      counter = counter + 1;
      z = x;
   end
end
```

```
Actor Consumer X, Y ==>
counter = 0;
cons: action X[x], Y[y] ==>
   do
      counter = counter + 1;
   end
end
```

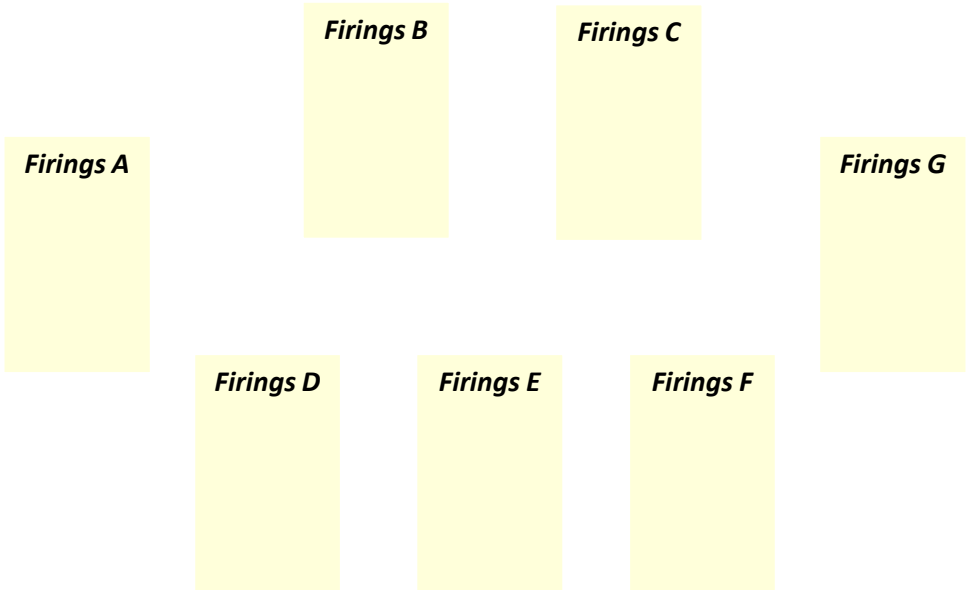


Example 1)

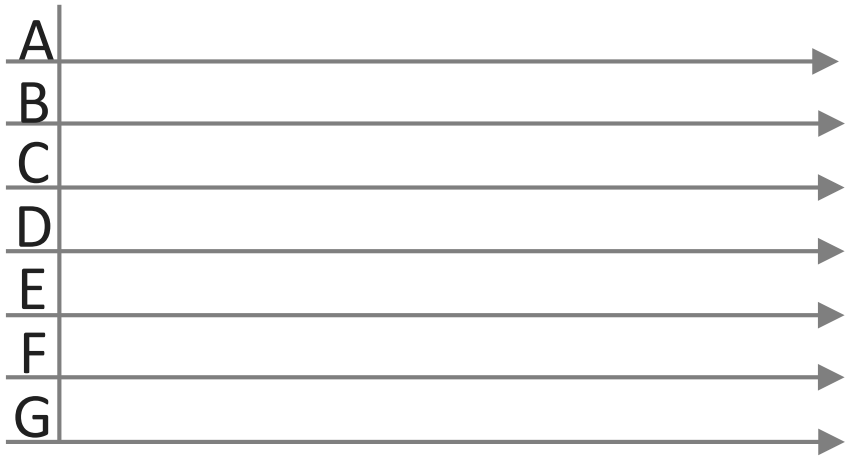
- **Partitioning:** all actors in one processing component
- **Scheduling:** pre-defined static scheduling
- **Buffer dimensioning:** infinite buffer sizes

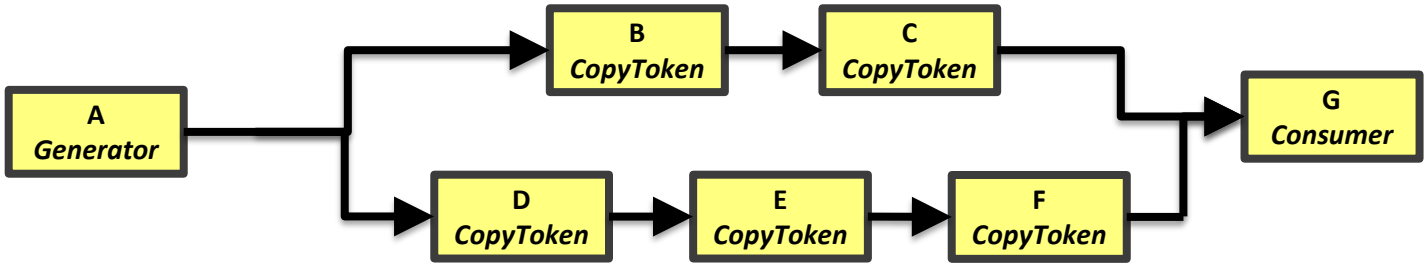
Example 1)

Execution Trace Graph



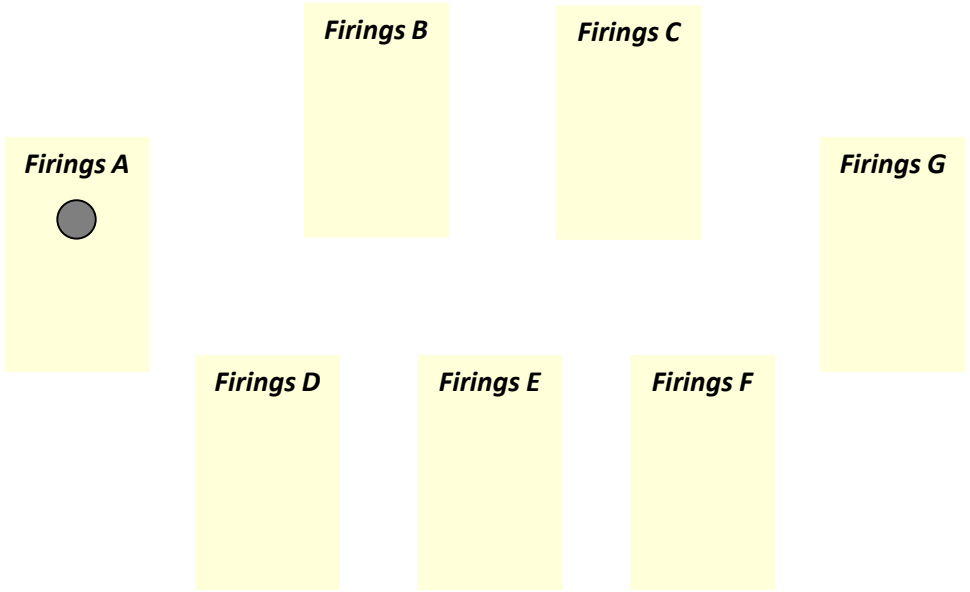
Execution Gantt Chart



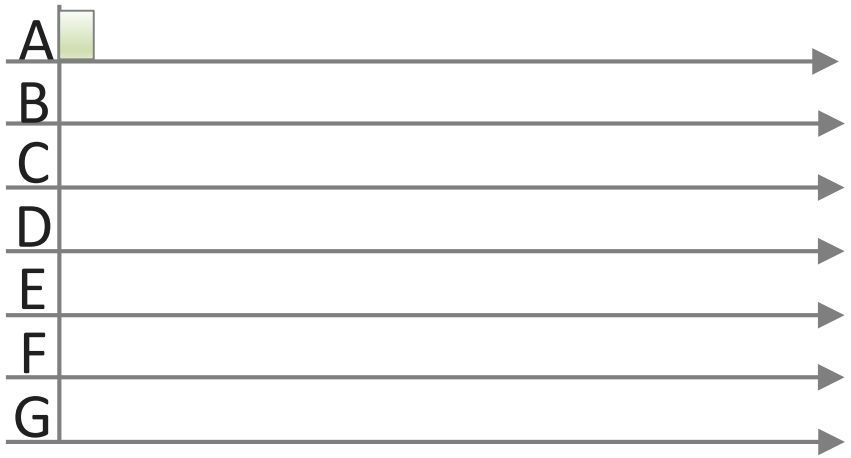


Example 1)

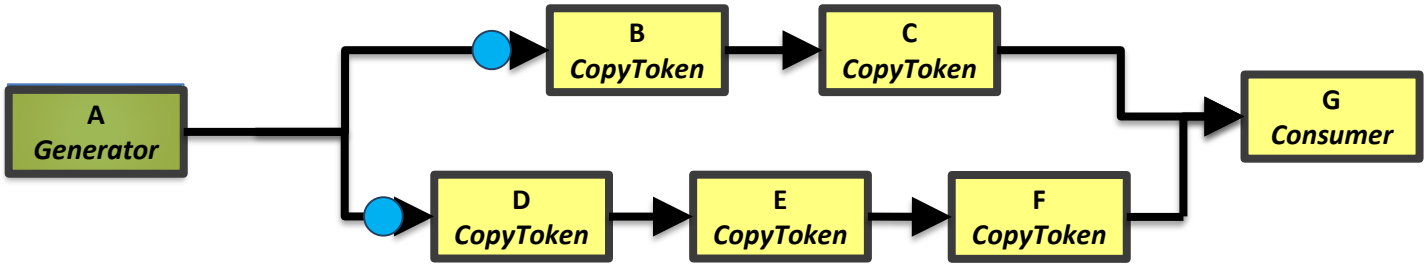
Execution Trace Graph



Execution Gantt Chart

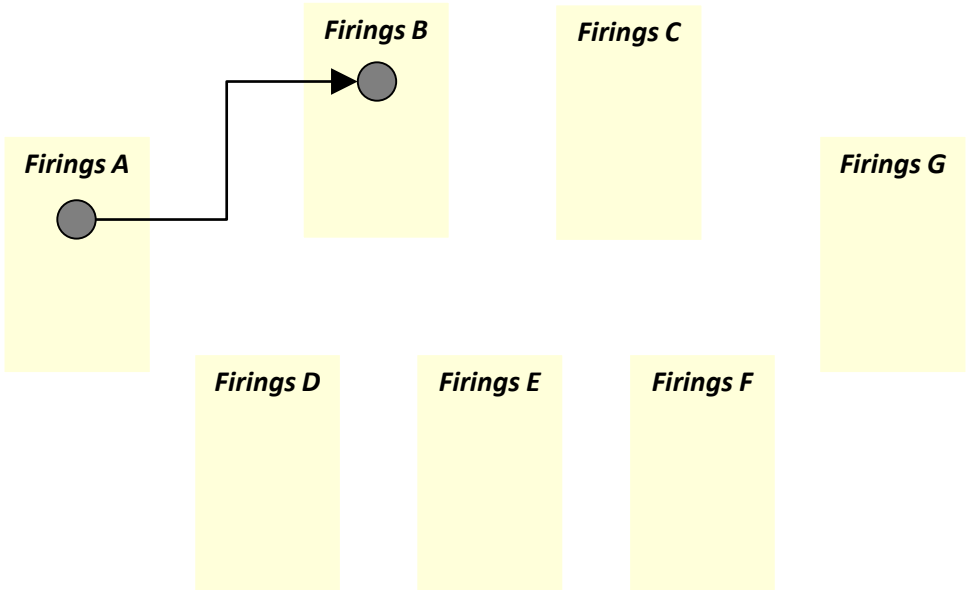


Firing Step: A:gen:1

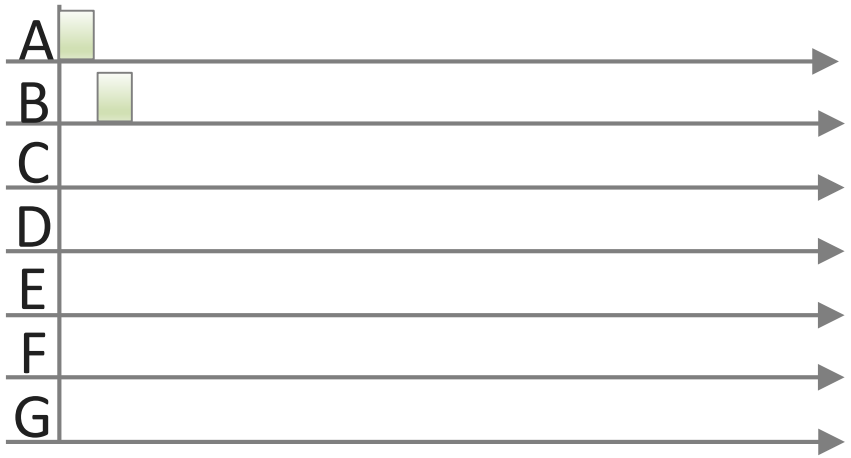


Example 1)

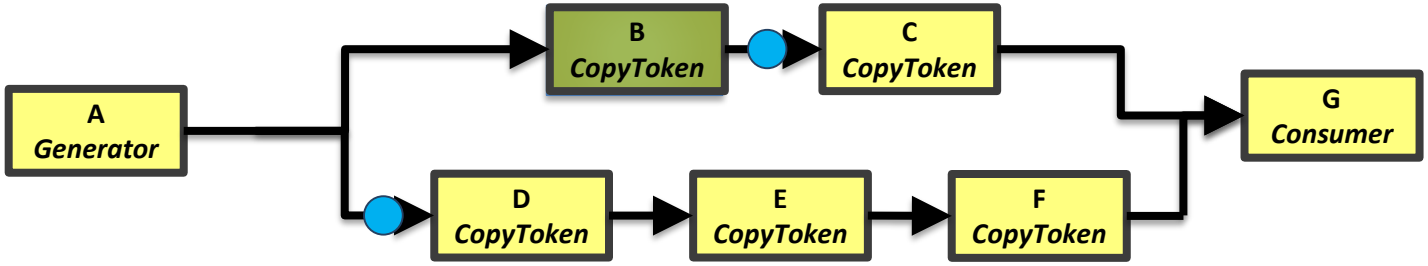
Execution Trace Graph



Execution Gantt Chart

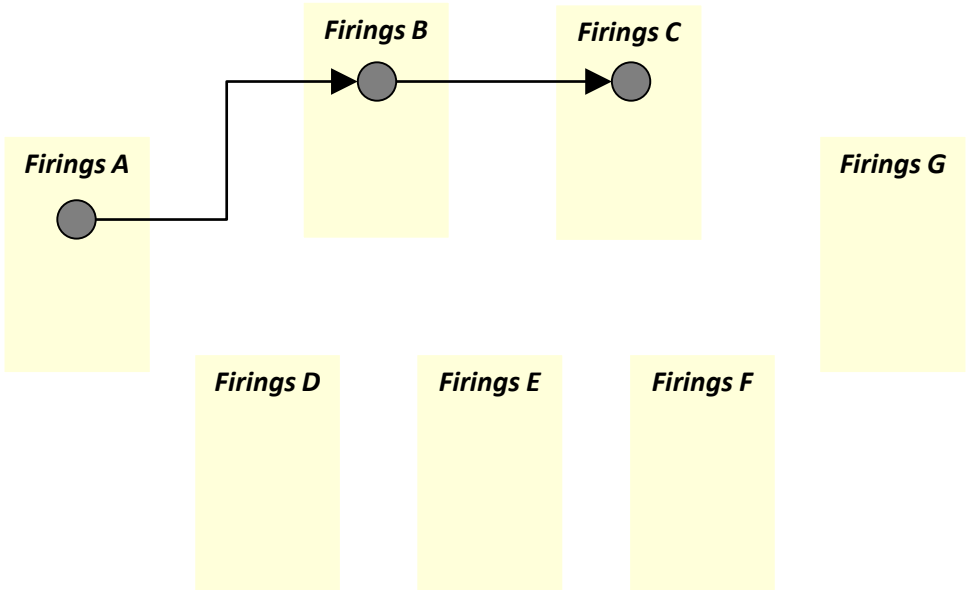


Firing Step: B:copy:1

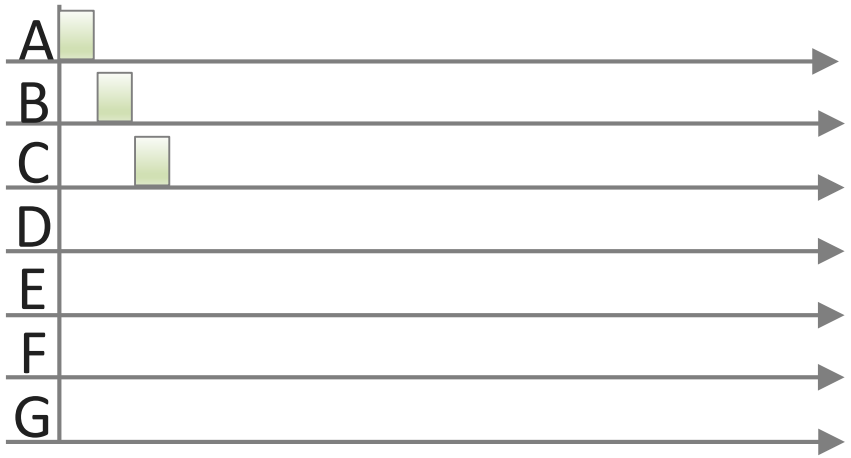


Example 1)

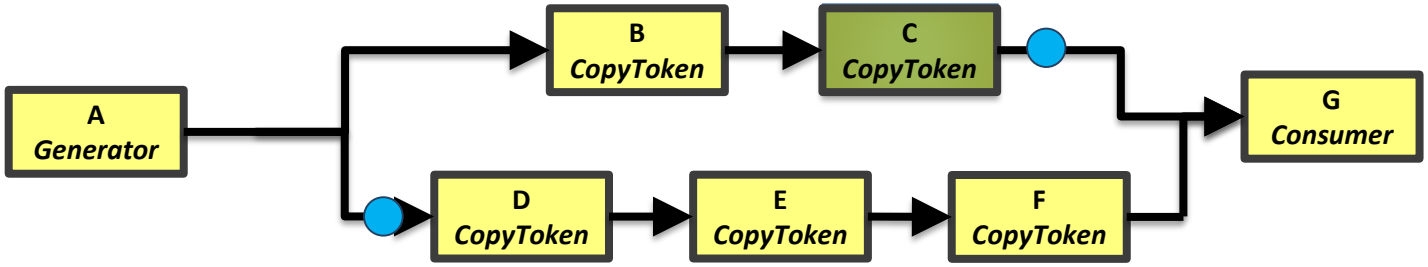
Execution Trace Graph



Execution Gantt Chart

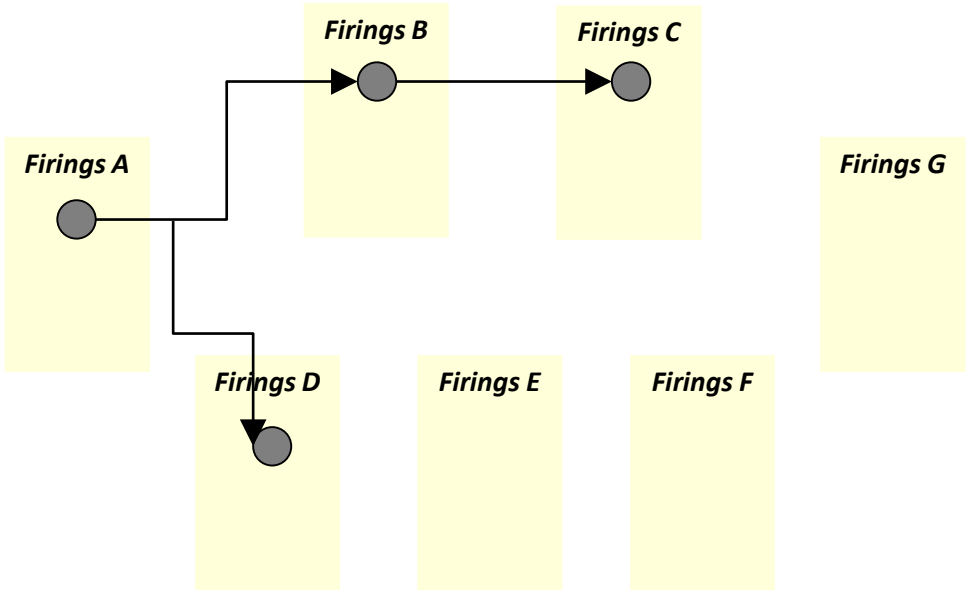


Firing Step: C:copy:1

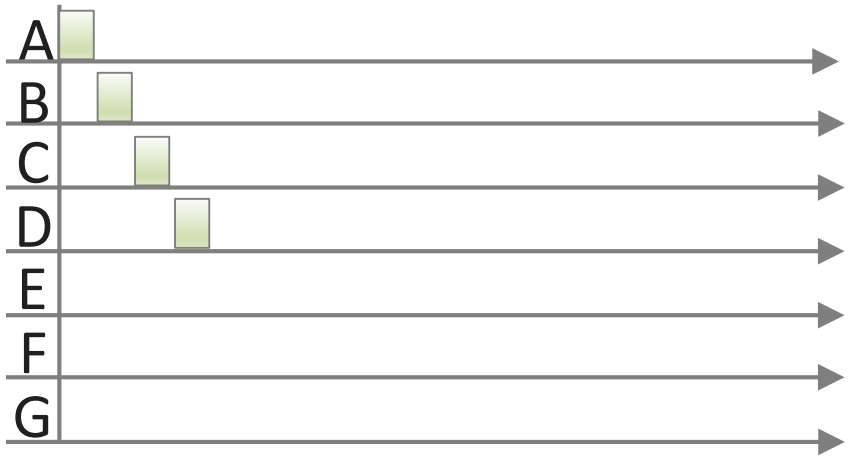


Example 1)

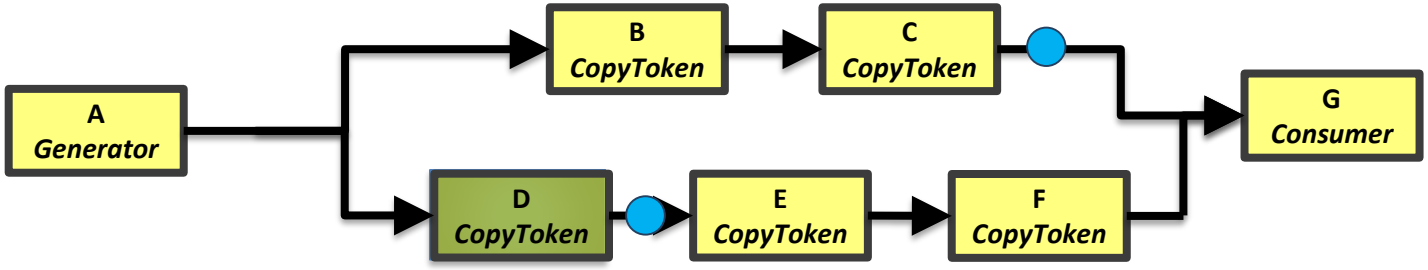
Execution Trace Graph



Execution Gantt Chart

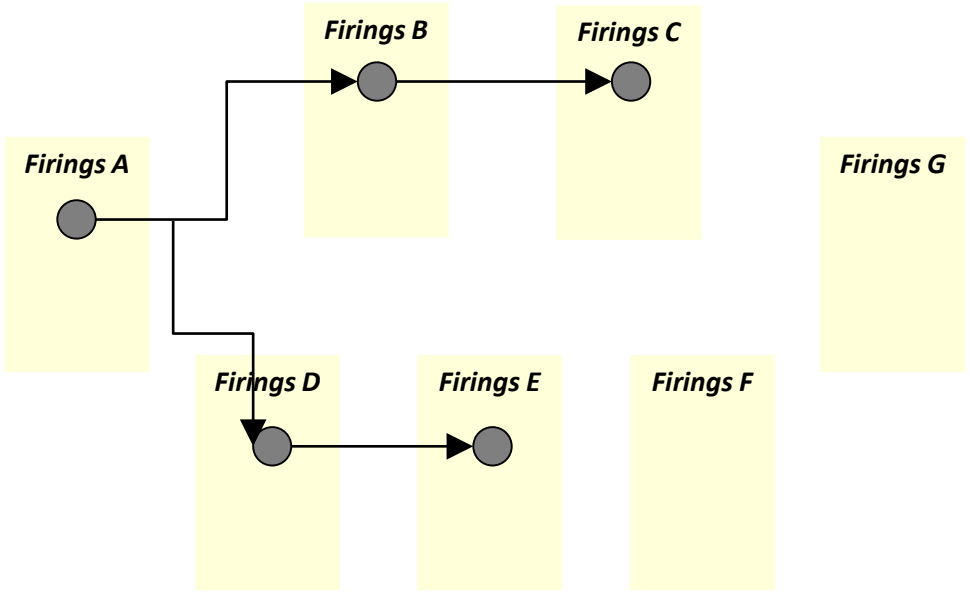


Firing Step: D:copy:1

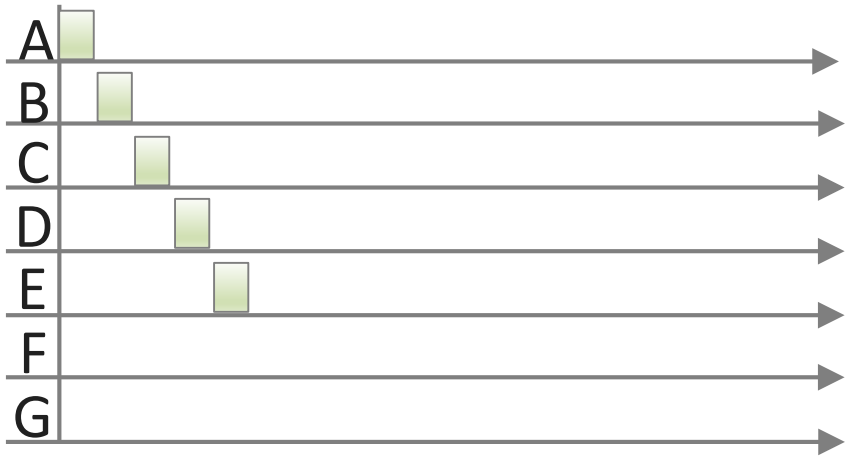


Example 1)

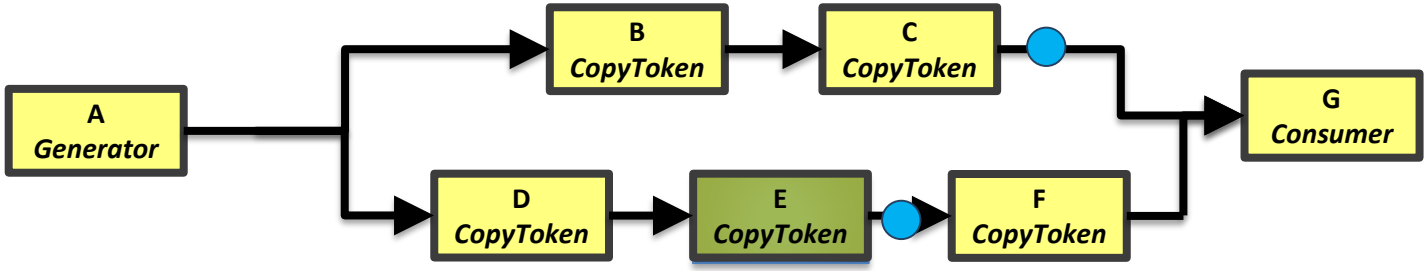
Execution Trace Graph



Execution Gantt Chart

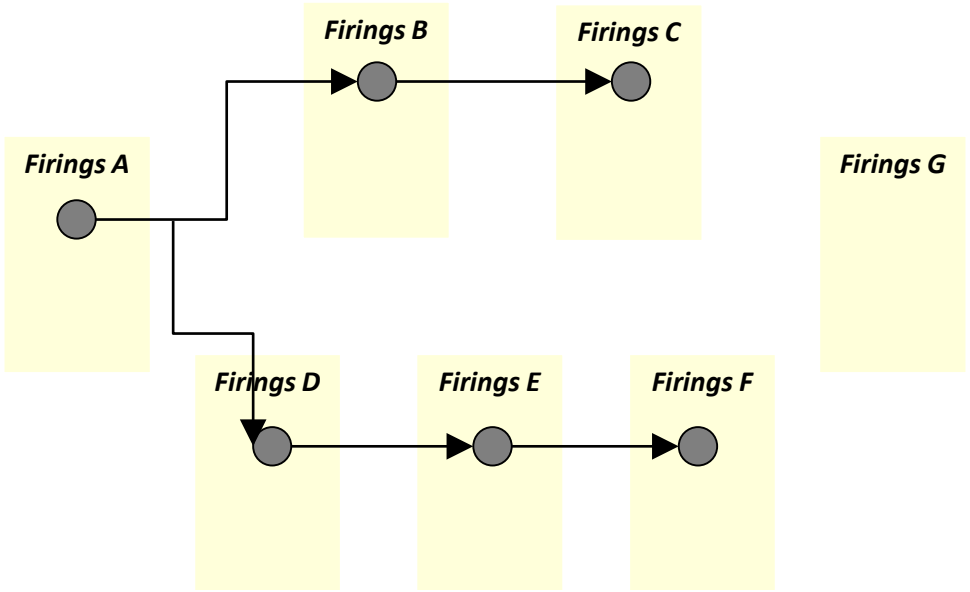


Firing Step: E:copy:1

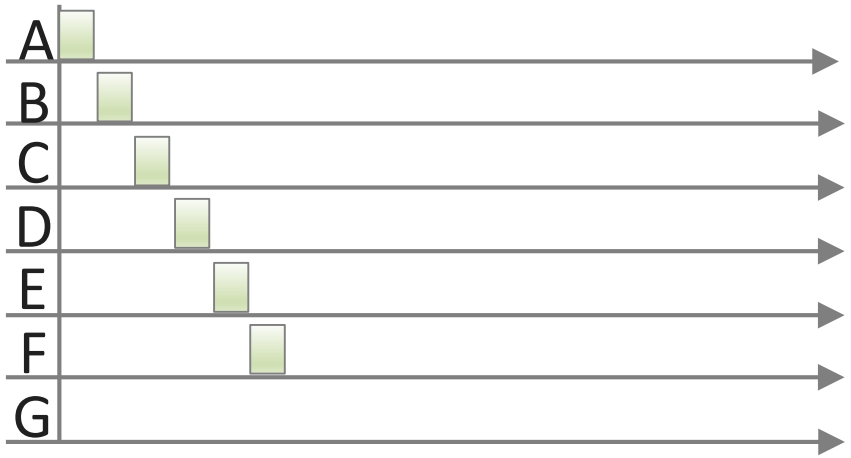


Example 1)

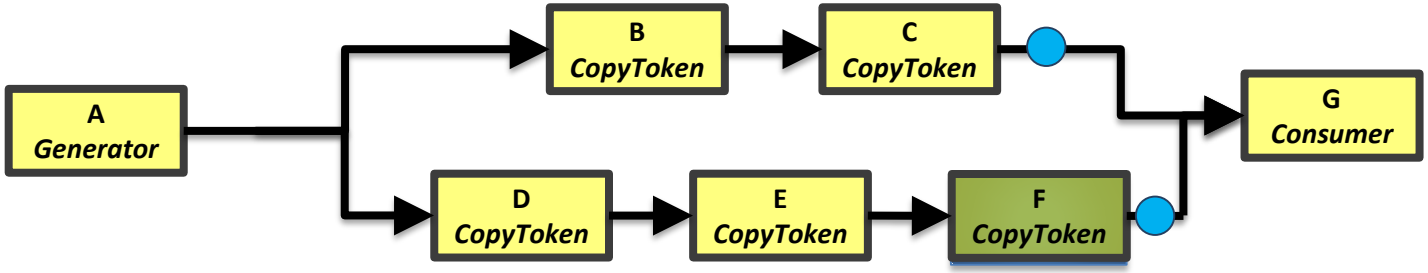
Execution Trace Graph



Execution Gantt Chart

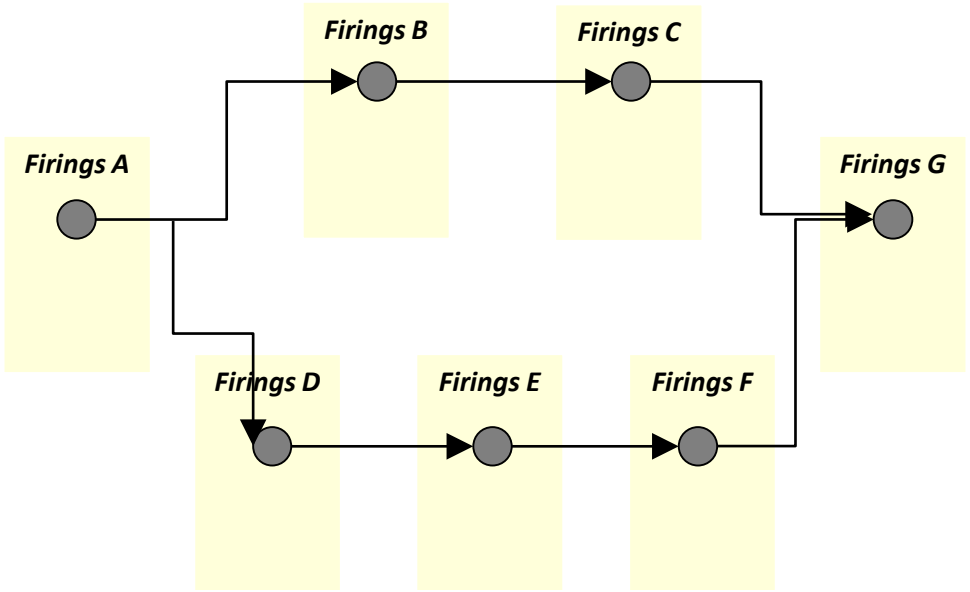


Firing Step: F:copy:1

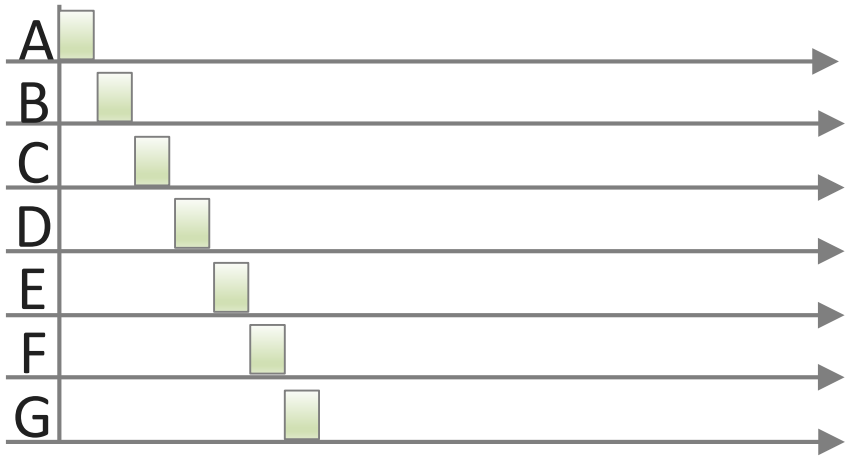


Example 1)

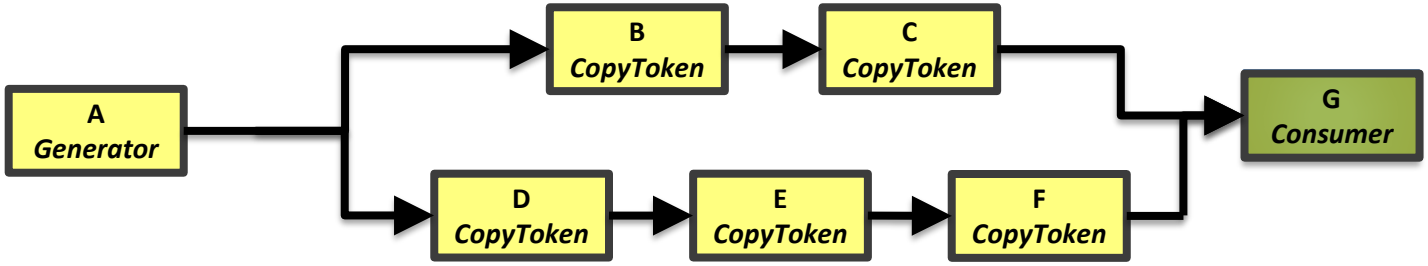
Execution Trace Graph



Execution Gantt Chart

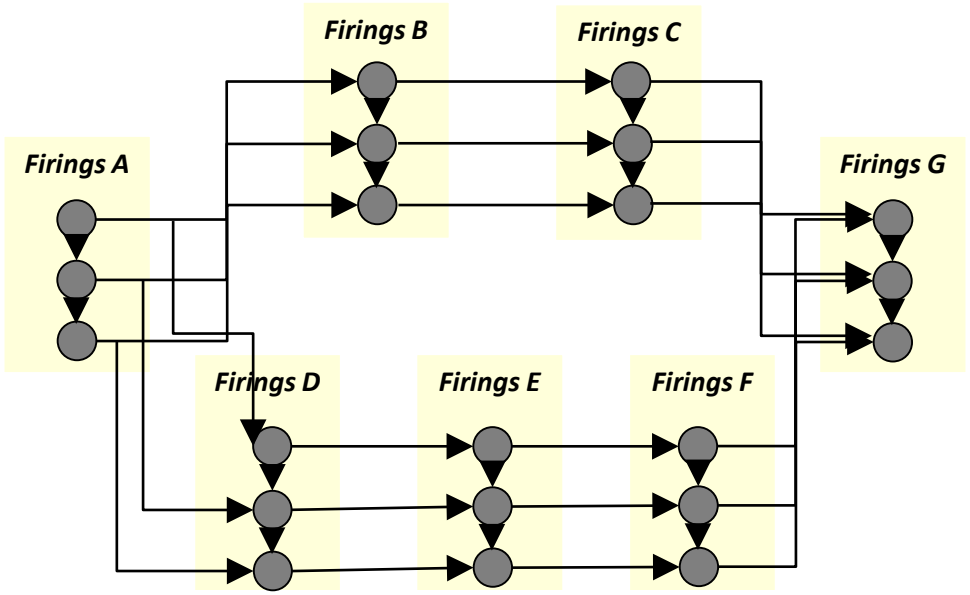


Firing Step: G:cons:1

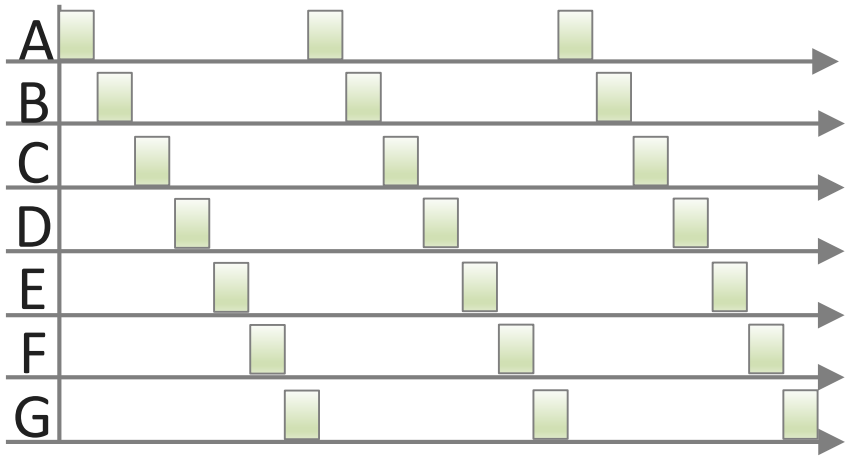


Example 1)

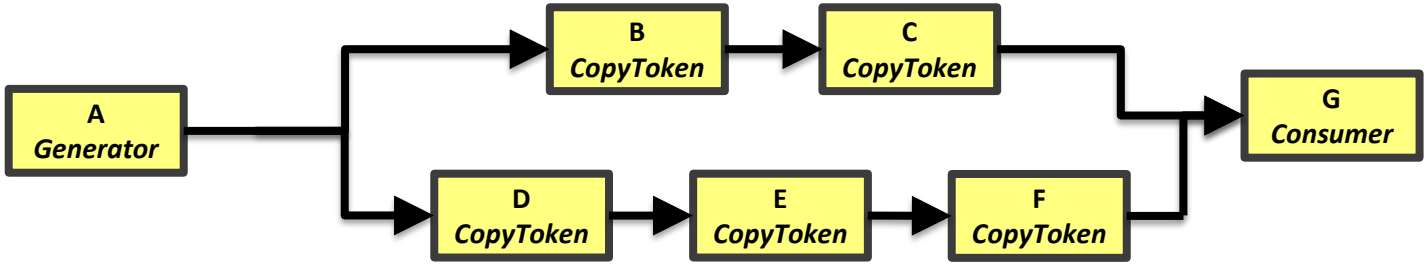
Execution Trace Graph



Execution Gantt Chart



At the end of the execution

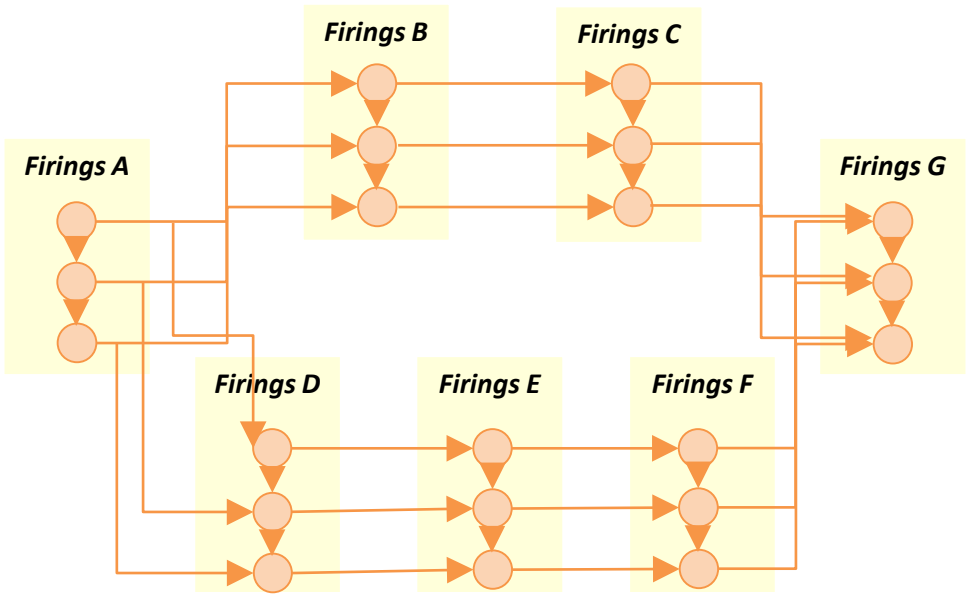


Example 2)

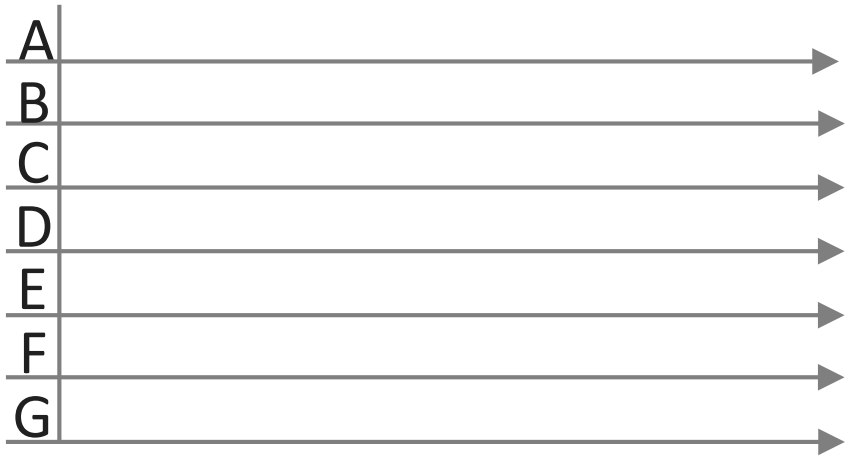
- **Partitioning:** each actor in a separate processing component
- **Scheduling:** pre-defined static scheduling
- **Buffer dimensioning:** infinite buffer sizes

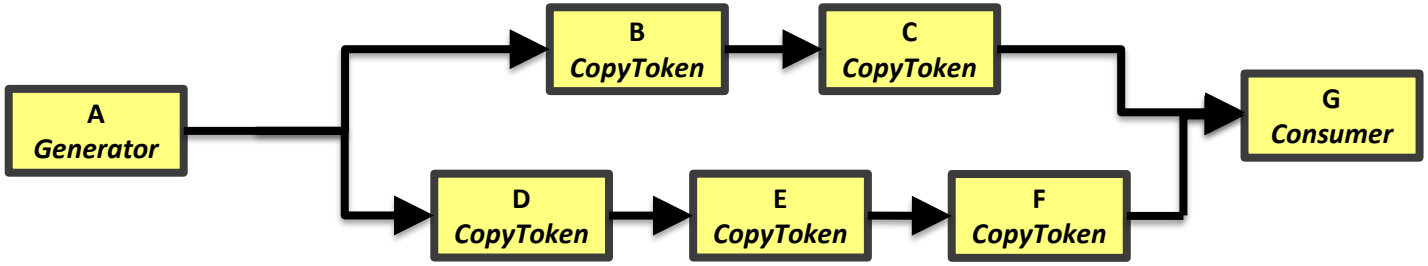
Example 2)

Execution Trace Graph



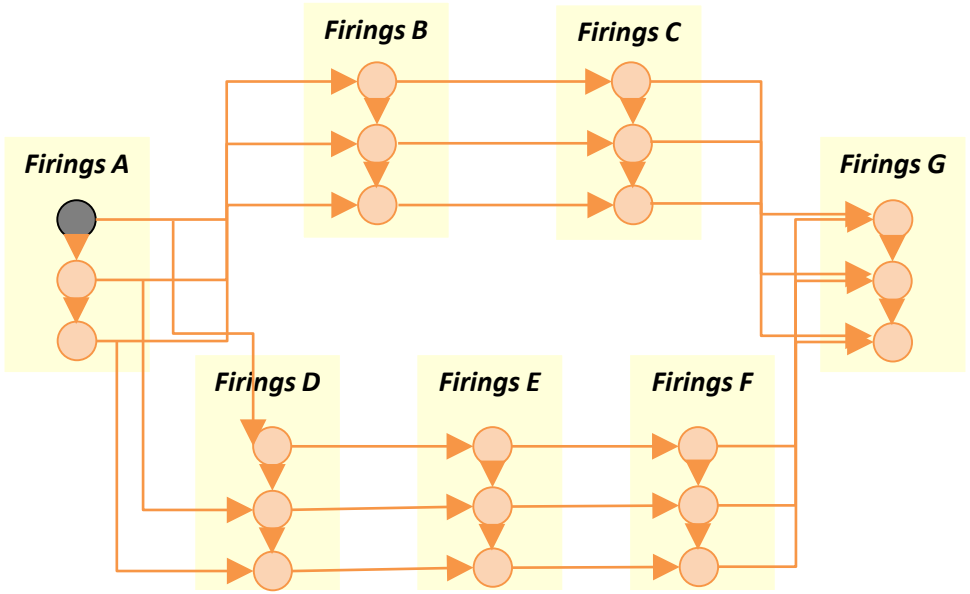
Execution Gantt Chart



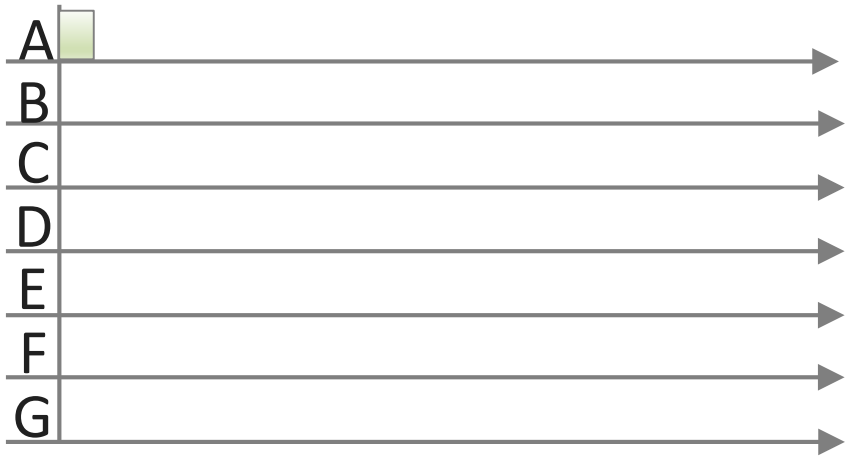


Example 2)

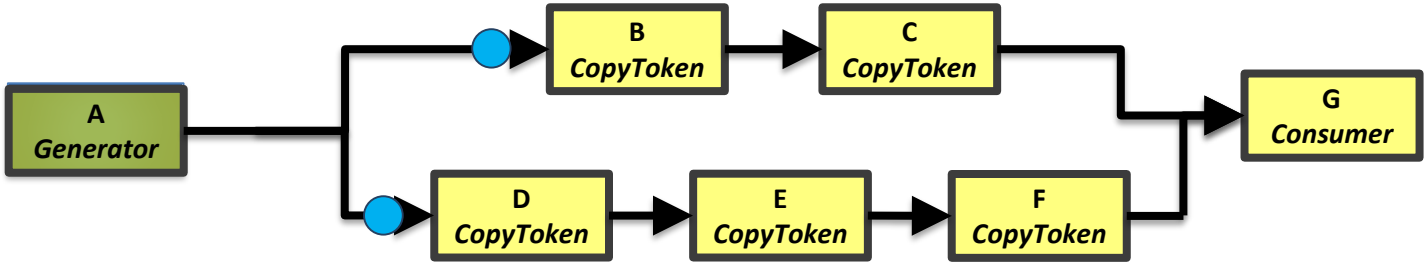
Execution Trace Graph



Execution Gantt Chart

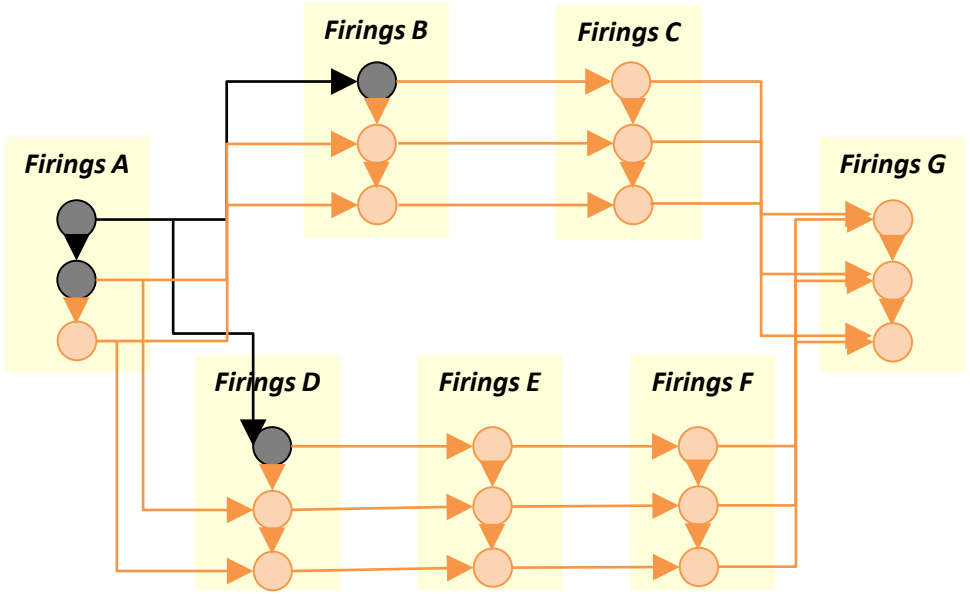


Firing Step: A:gen:1

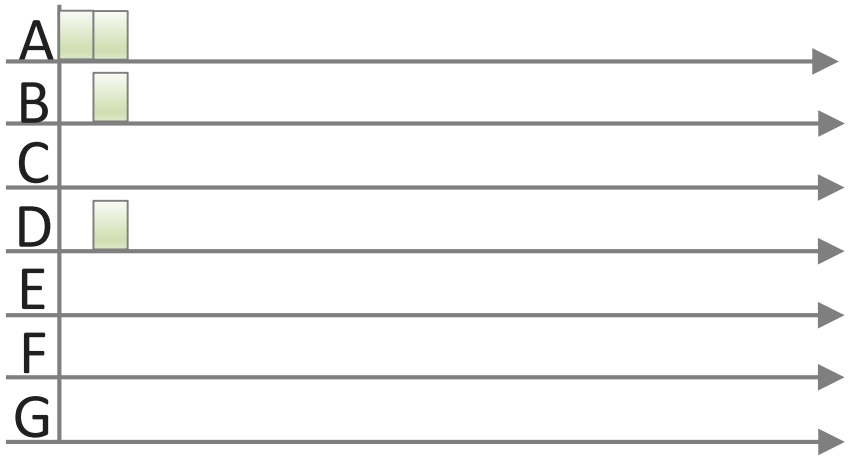


Example 2)

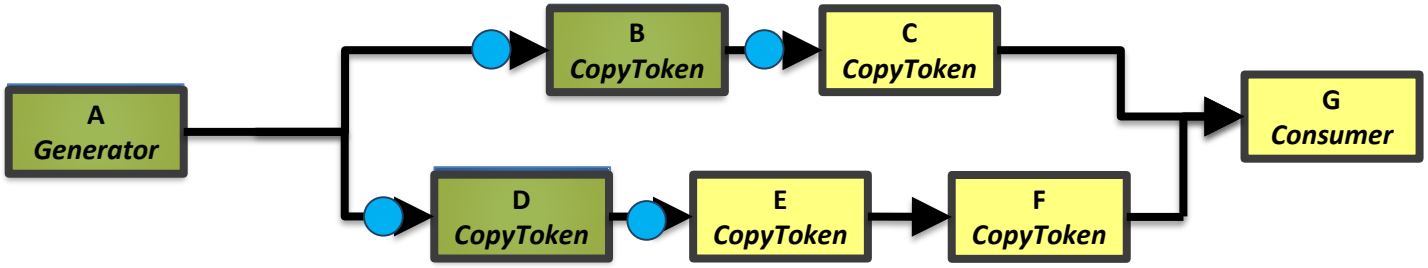
Execution Trace Graph



Execution Gantt Chart

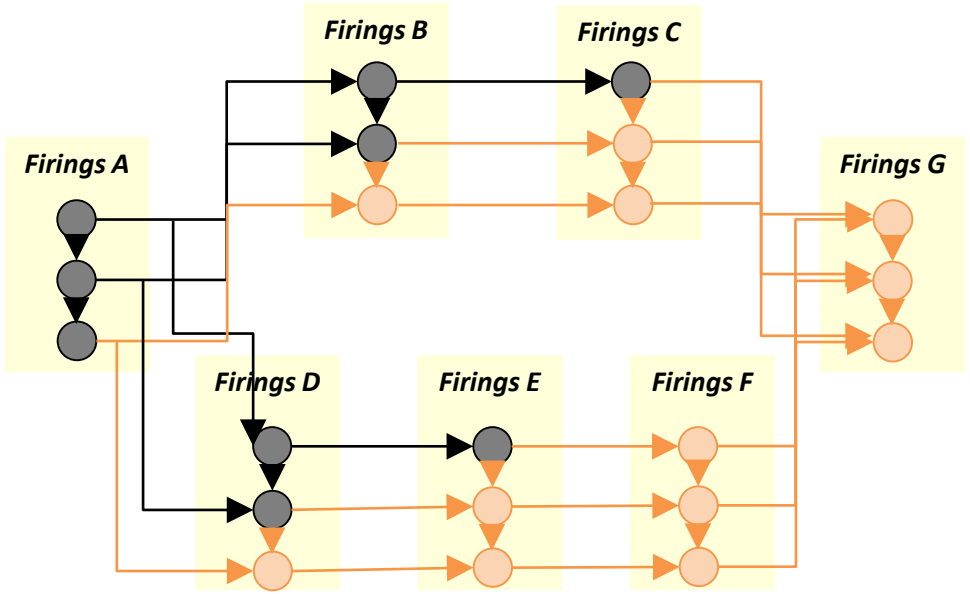


Firing Step: A:gen:2, B:copy:1, D:copy:1

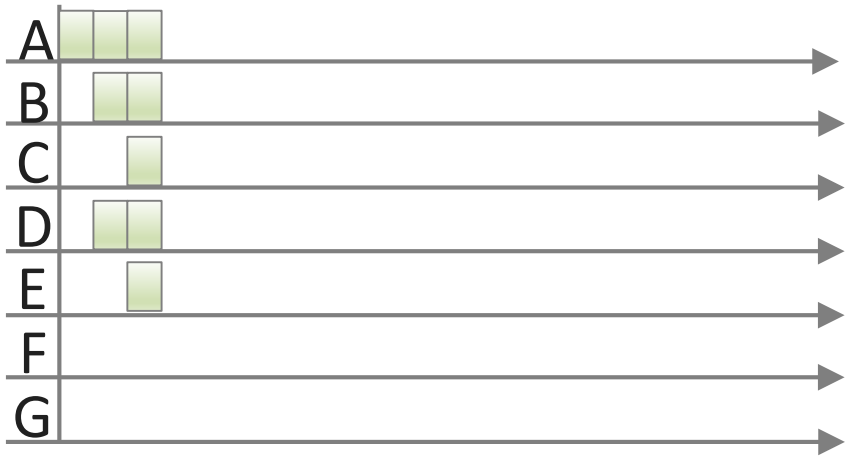


Example 2)

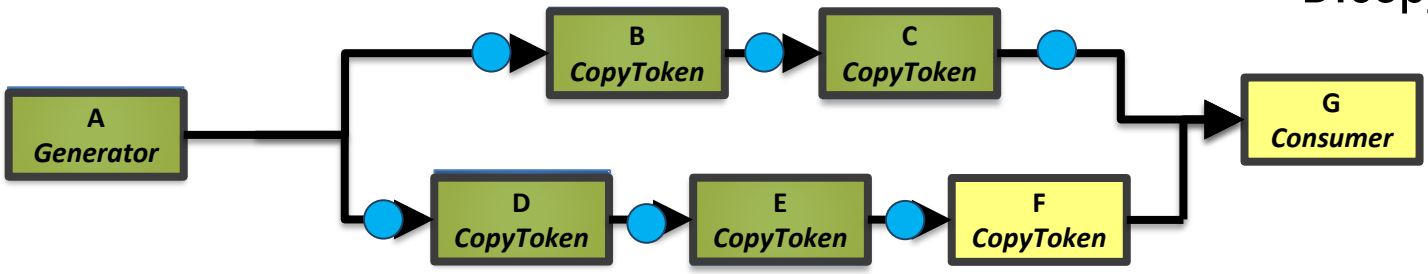
Execution Trace Graph



Execution Gantt Chart

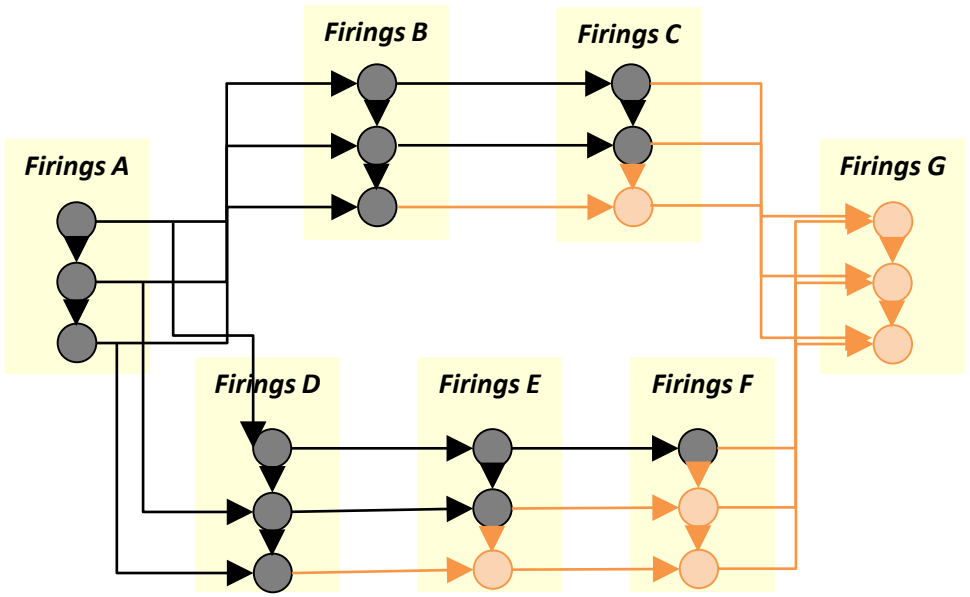


Firing Step: A:gen:3, B:copy:2, C:copy:1
D:copy:2, E:copy:1



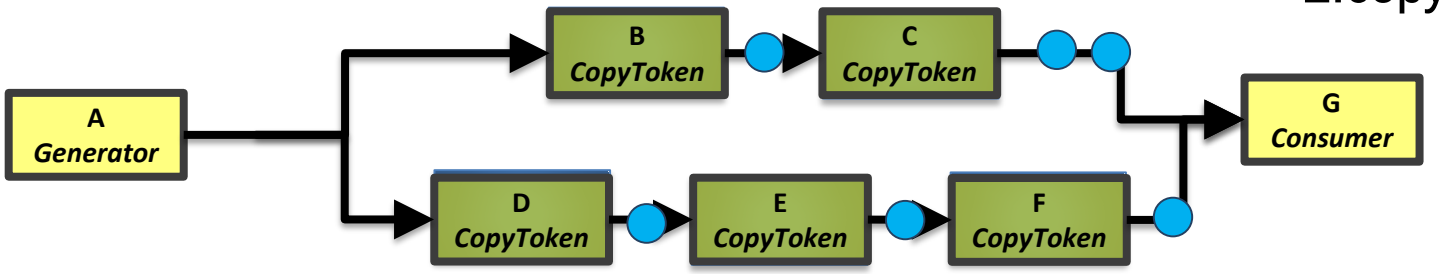
Example 2)

Execution Trace Graph



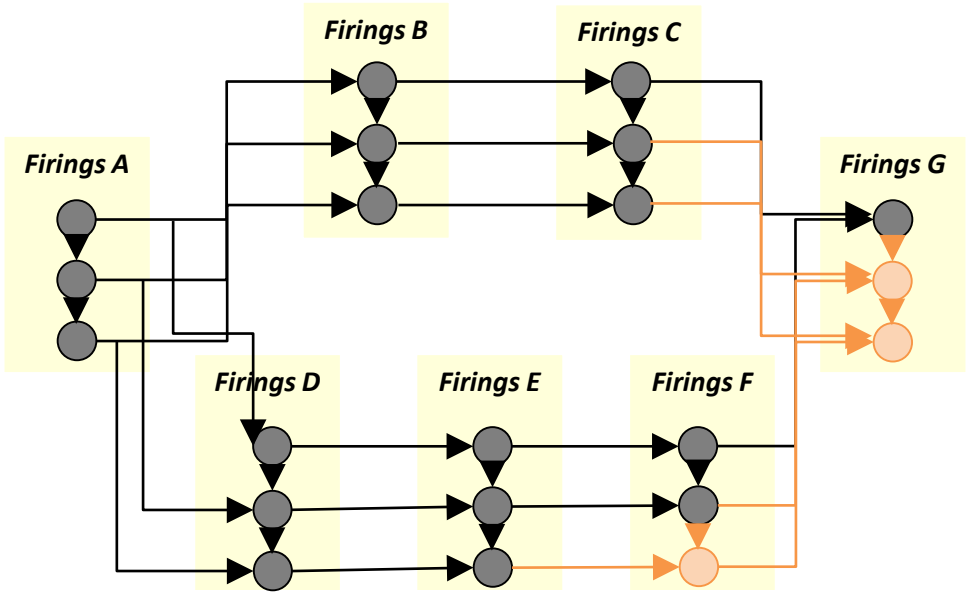
Execution Gantt Chart

Firing Step: B:copy:3, C:copy:2, D:copy:3
E:copy:2, F:copy:1

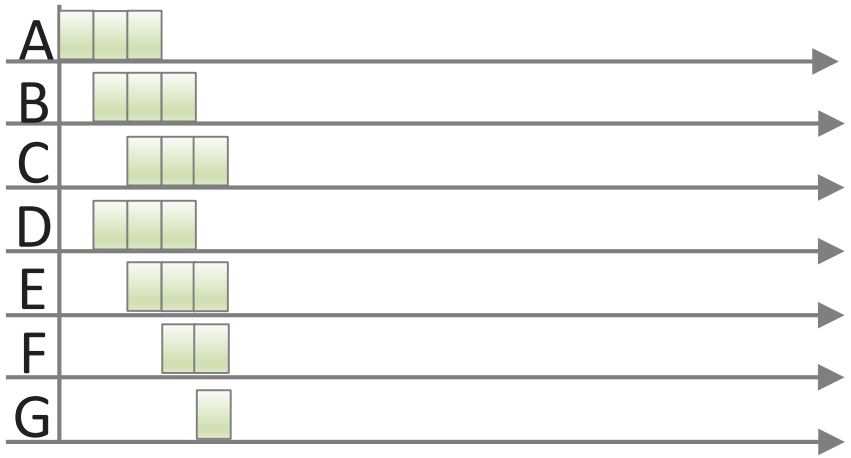


Example 2)

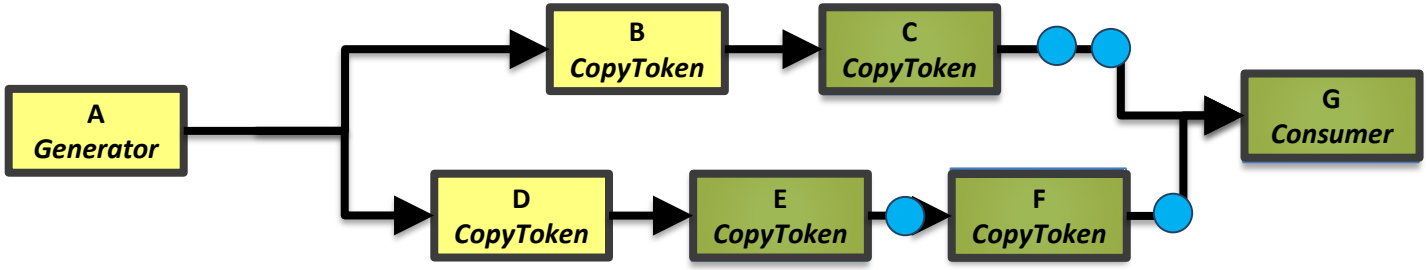
Execution Trace Graph



Execution Gantt Chart

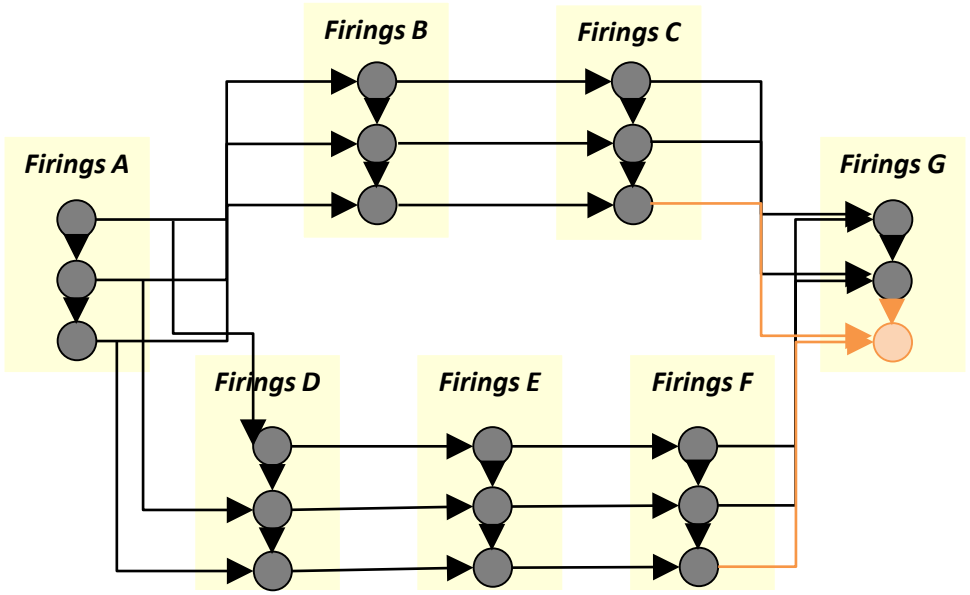


Firing Step: C:copy:3, E:copy:3, F:copy:3
G:cons:1

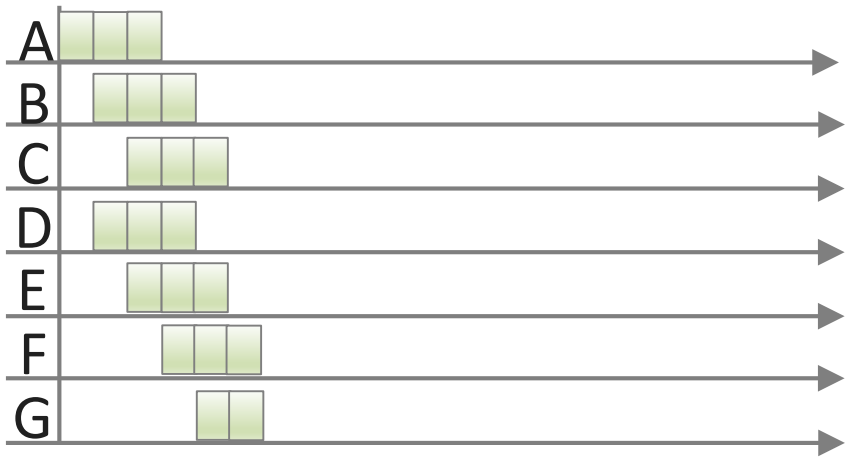


Example 2)

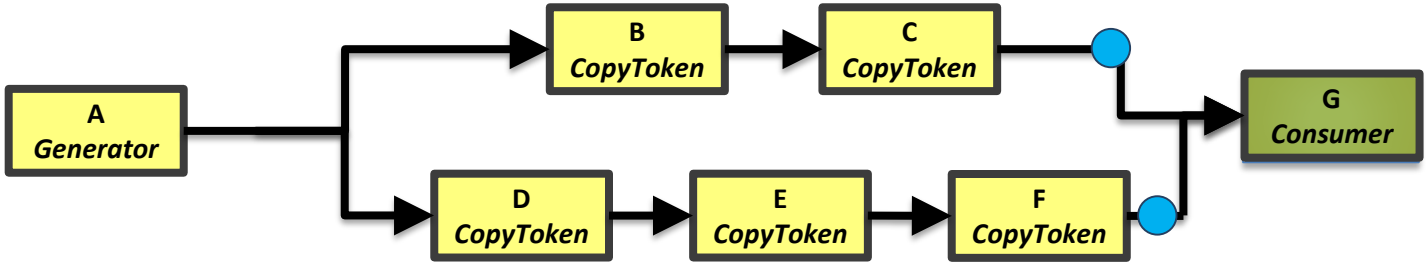
Execution Trace Graph



Execution Gantt Chart

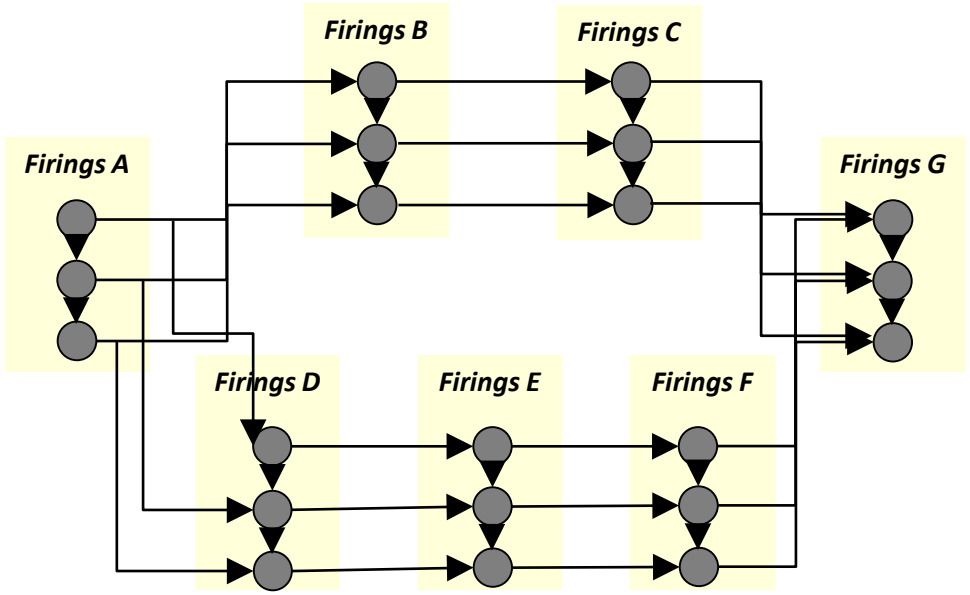


Firing Step: F:copy:3, G:cons:2

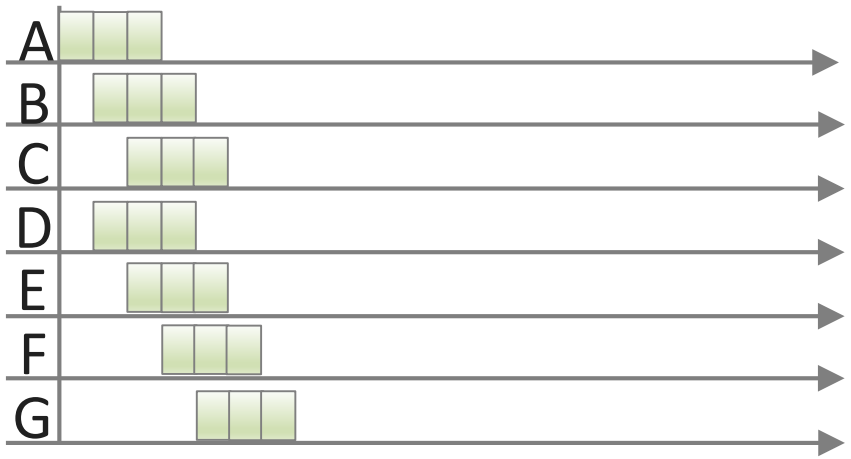


Example 2)

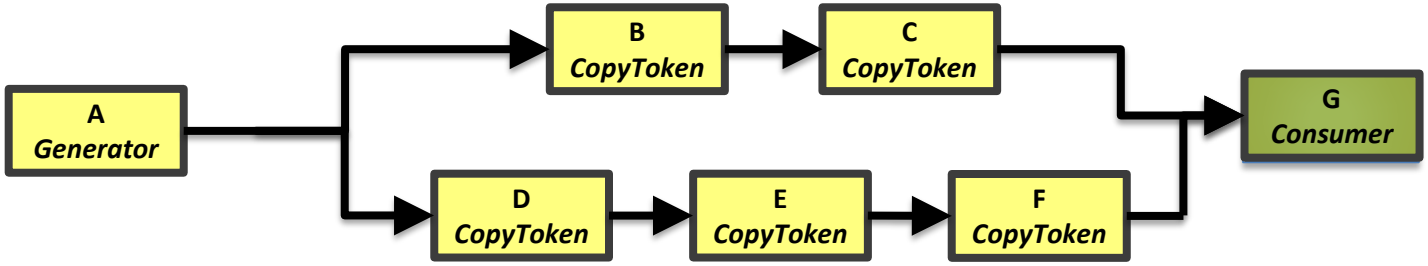
Execution Trace Graph



Execution Gantt Chart

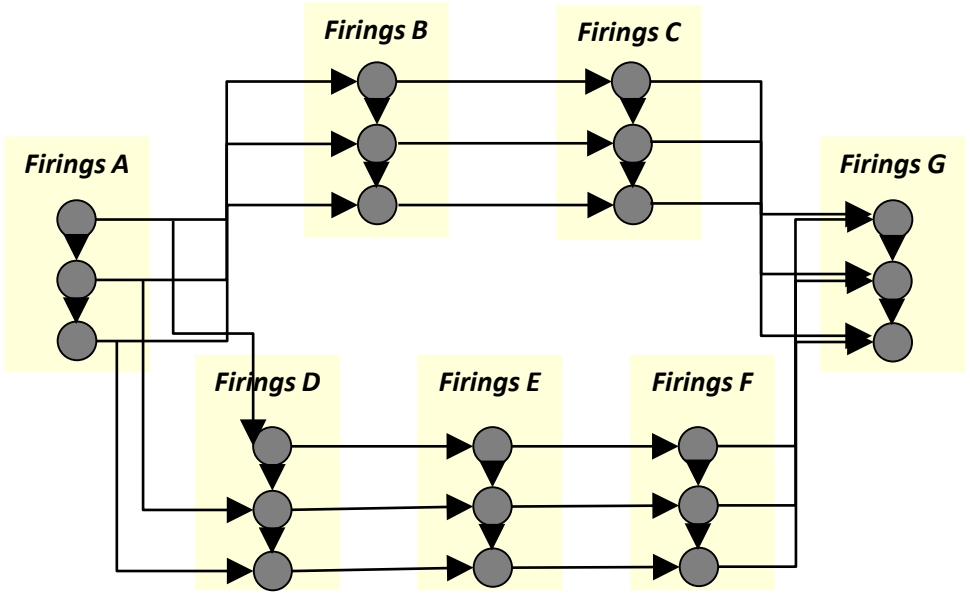


Firing Step: G:cons:3

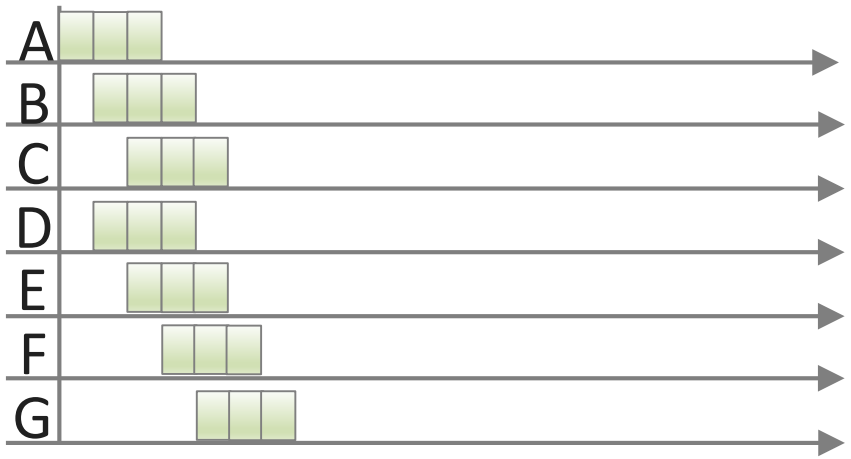


Example 2)

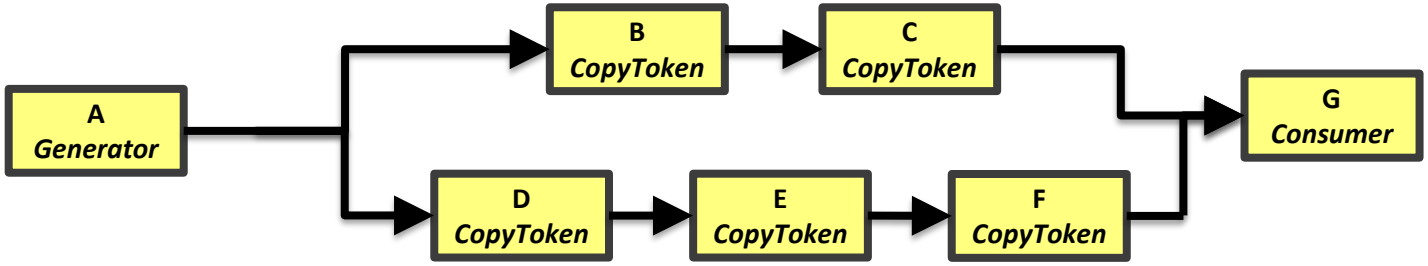
Execution Trace Graph



Execution Gantt Chart



At the end of the execution

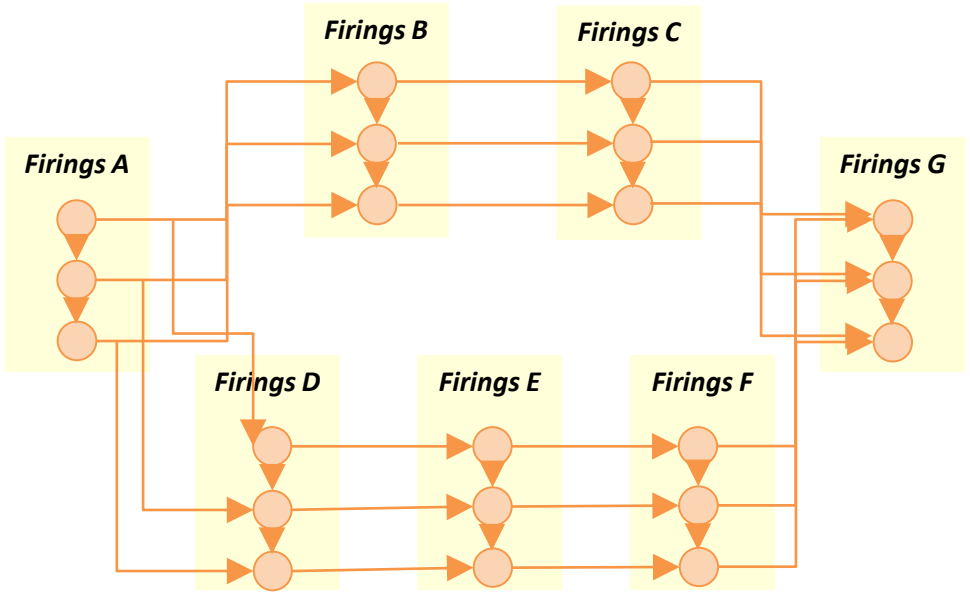


Example 3)

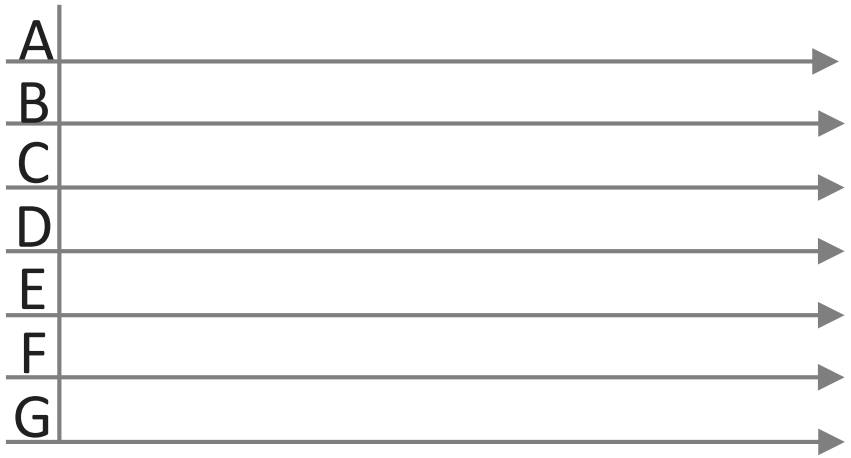
- **Partitioning:** each actor in a separate processing component
- **Scheduling:** pre-defined static scheduling
- **Buffer dimensioning:** minimal buffer sizes (1)

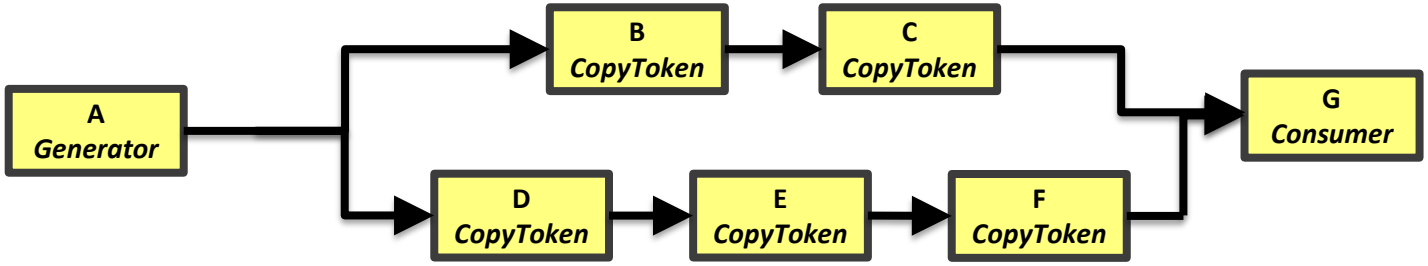
Example 3)

Execution Trace Graph



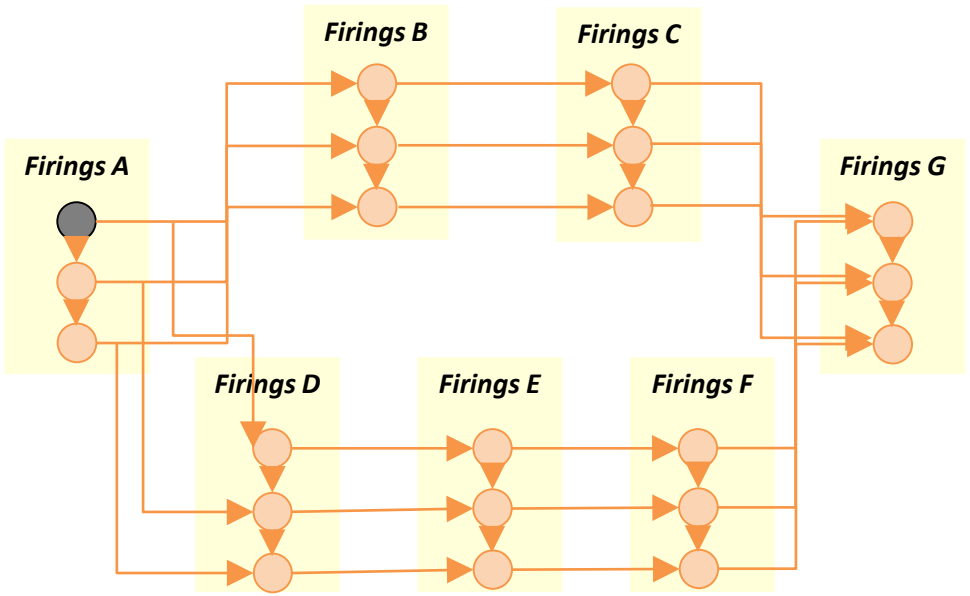
Execution Gantt Chart



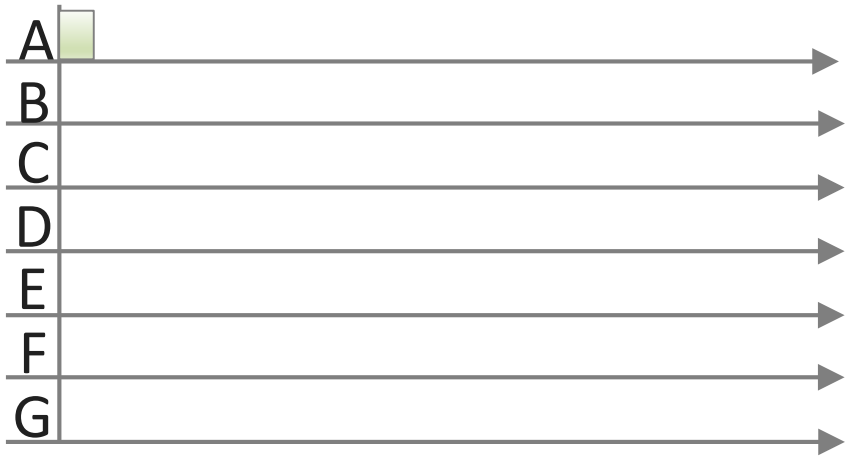


Example 3)

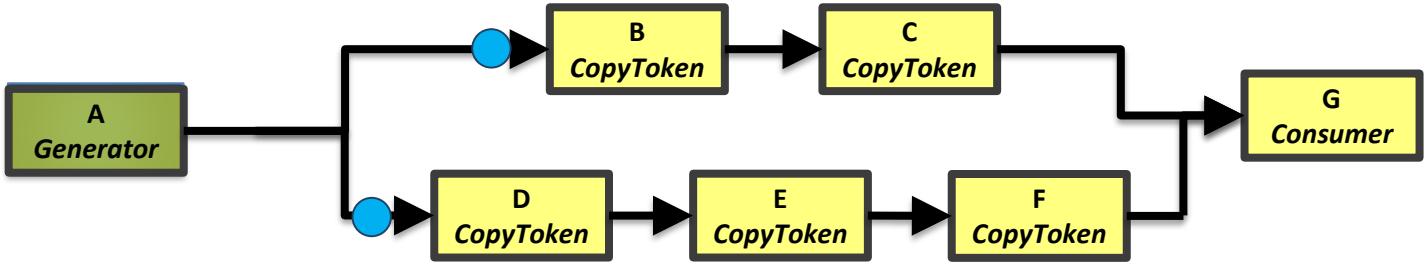
Execution Trace Graph



Execution Gantt Chart

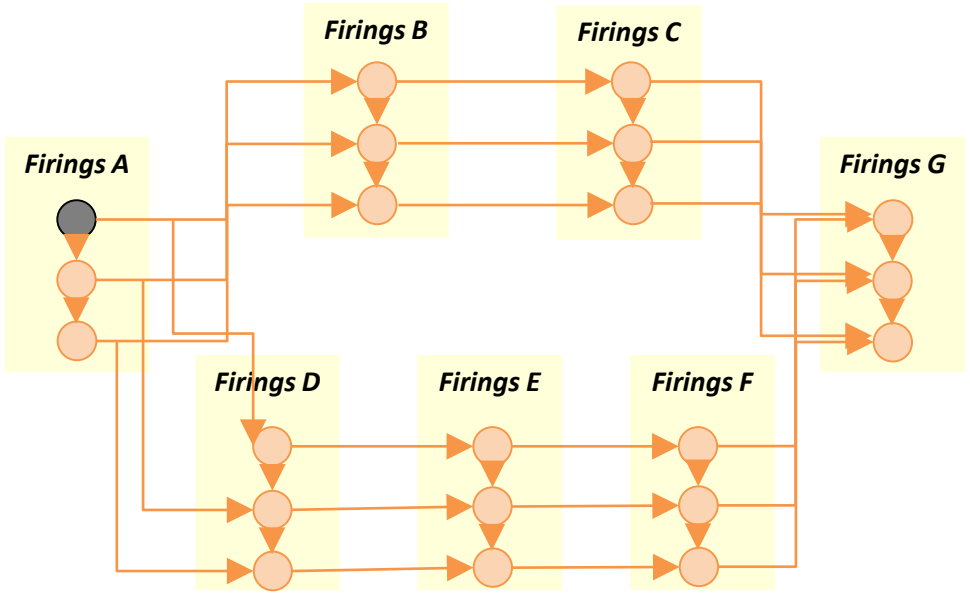


Firing Step: A:gen:1

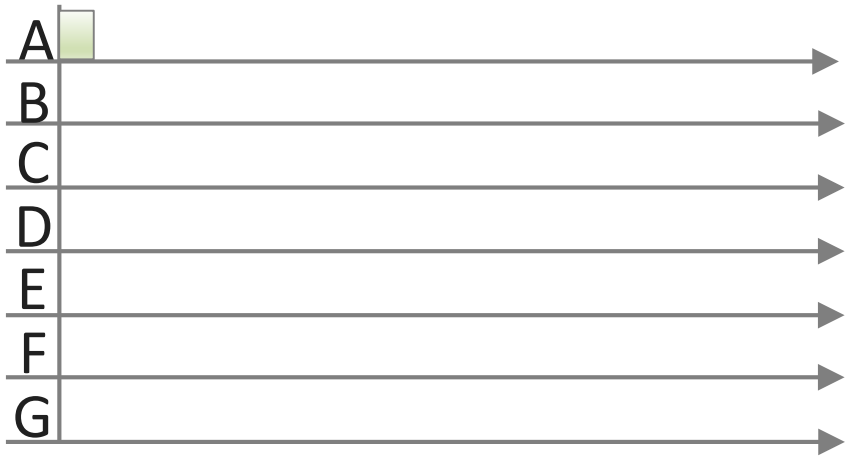


Example 3)

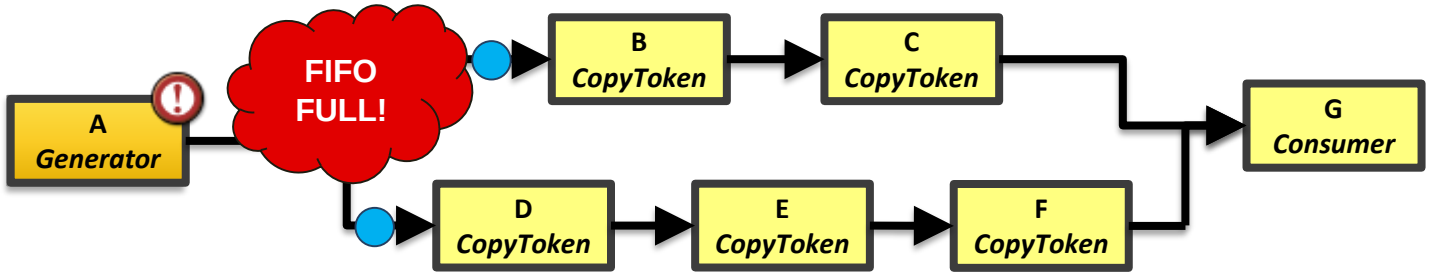
Execution Trace Graph



Execution Gantt Chart

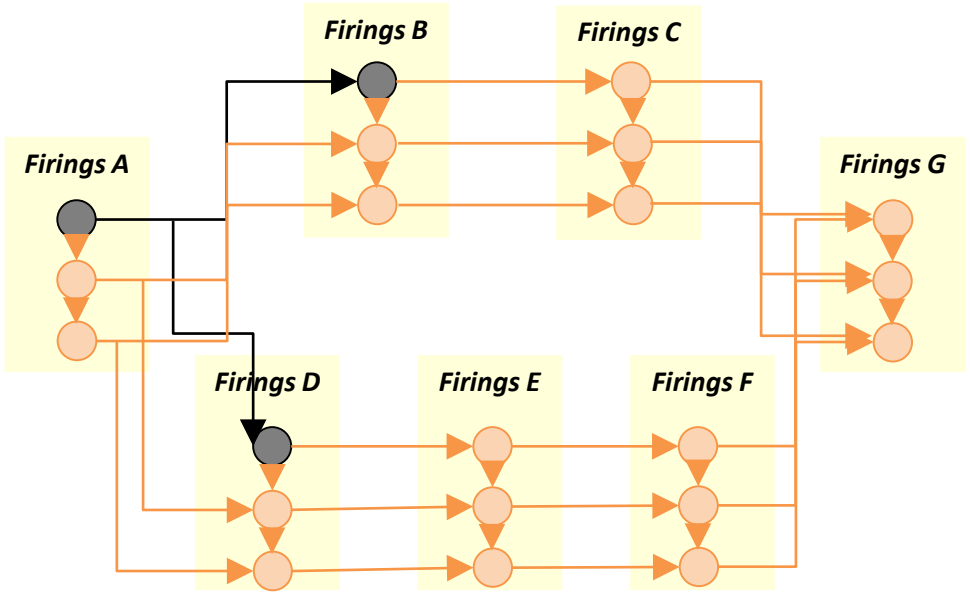


Firing Step: B:copy:1, D:copy:1

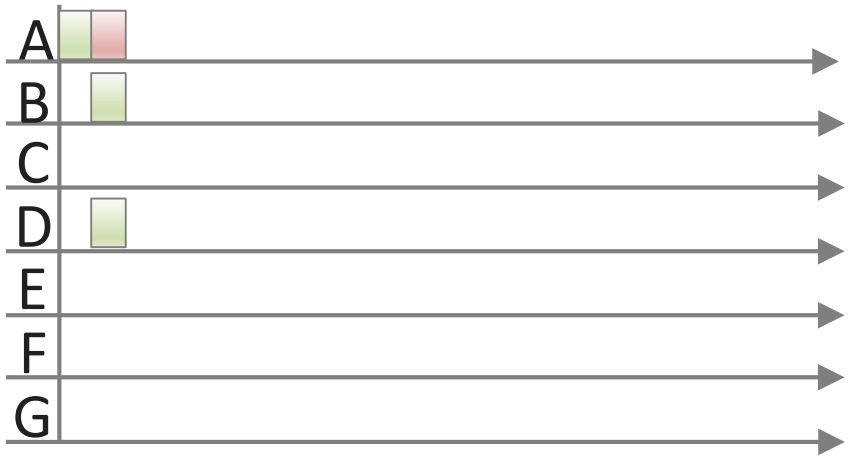


Example 3)

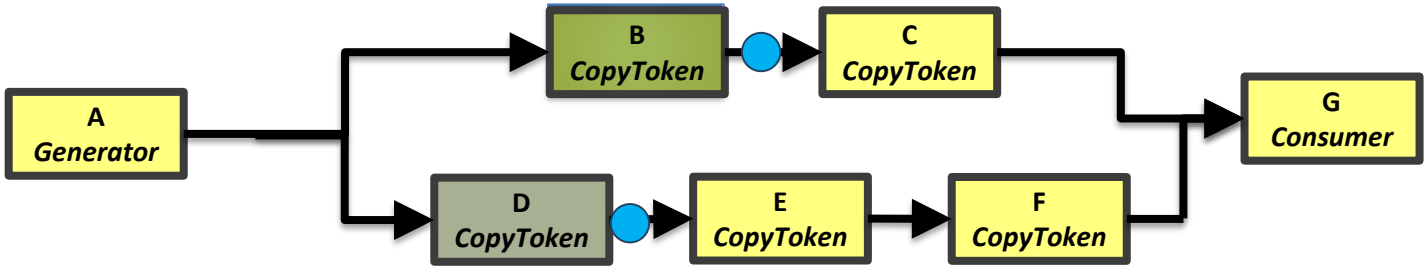
Execution Trace Graph



Execution Gantt Chart

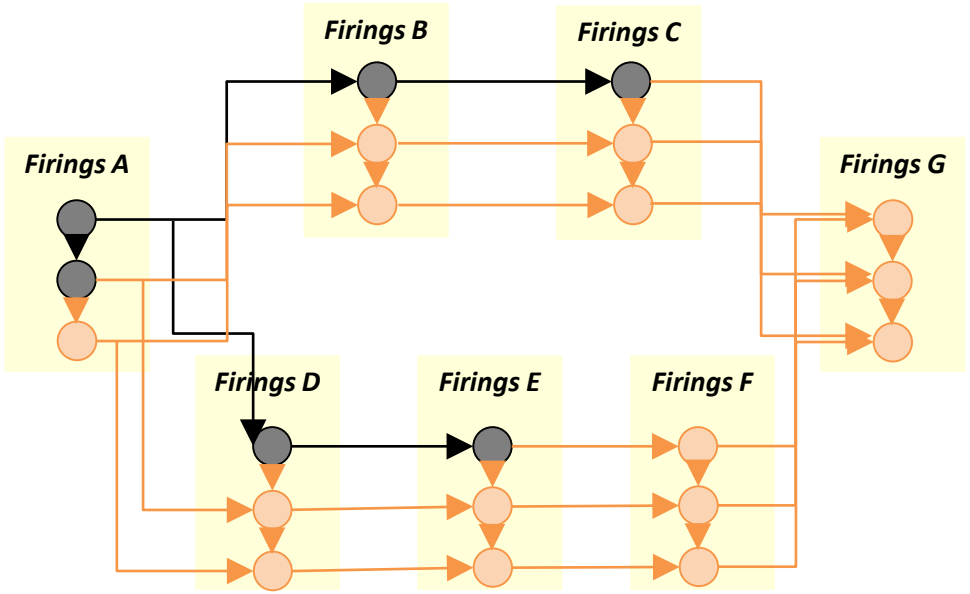


Firing Step: B:copy:1, D:copy:1

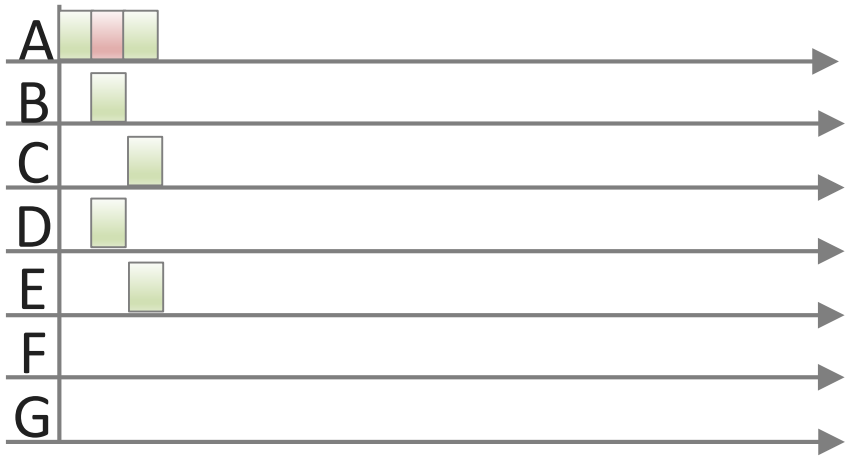


Example 3)

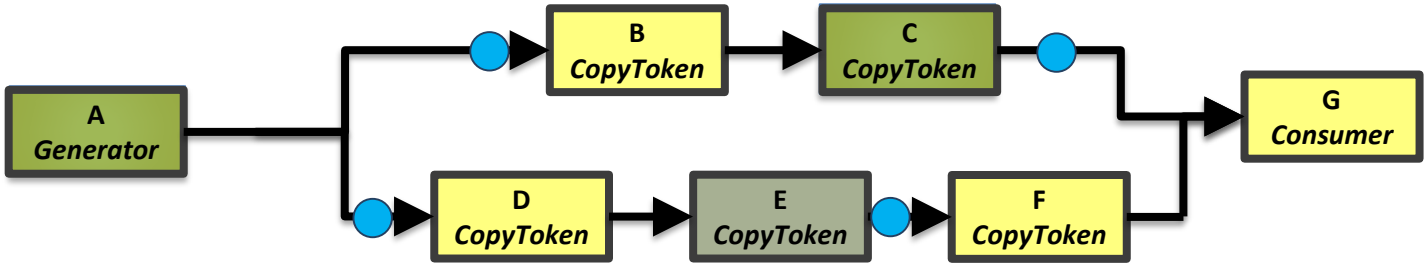
Execution Trace Graph



Execution Gantt Chart

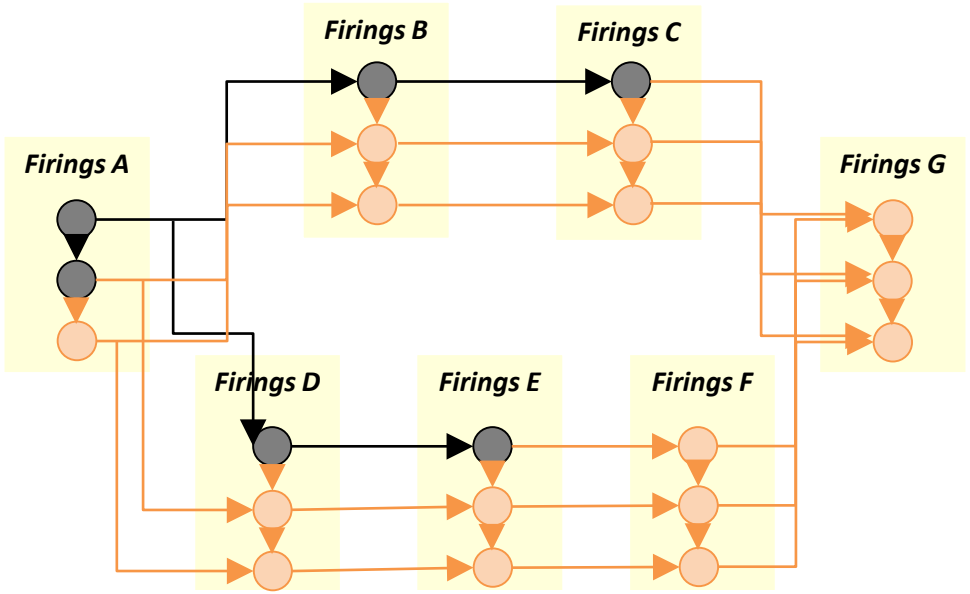


Firing Step: A:gen:2, C:copy:1, E:copy:1

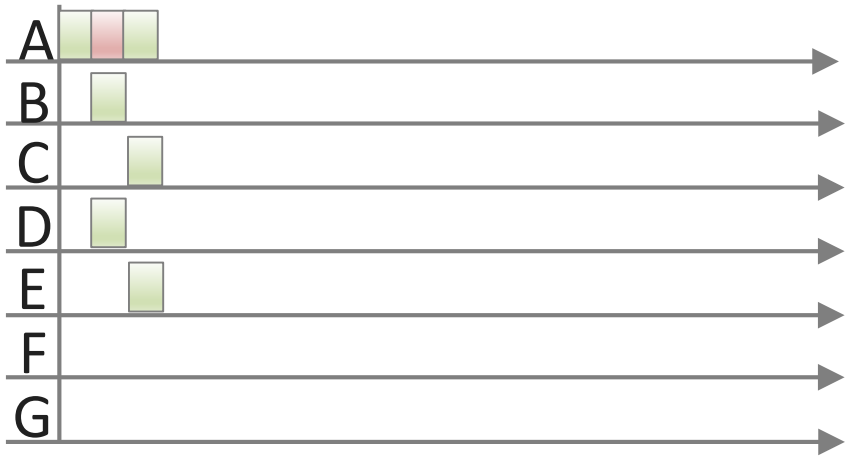


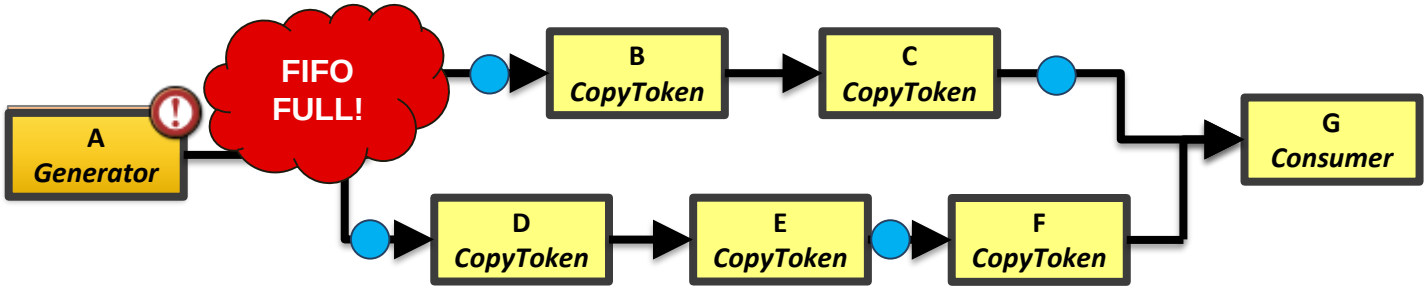
Example 3)

Execution Trace Graph



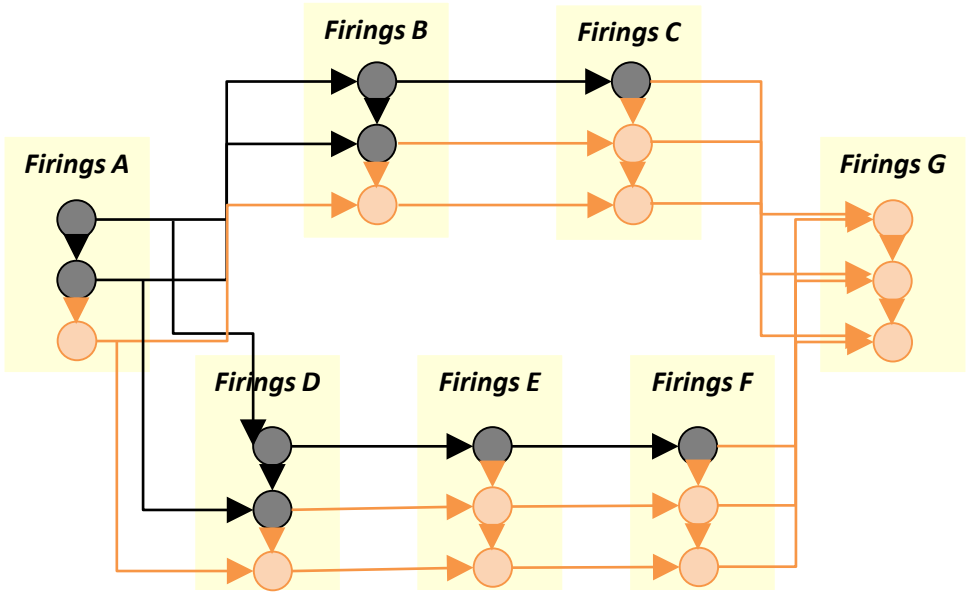
Execution Gantt Chart



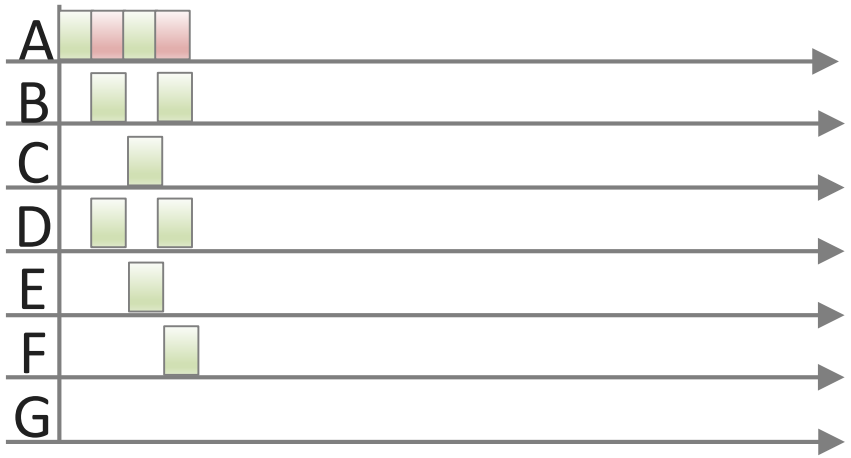


Example 3)

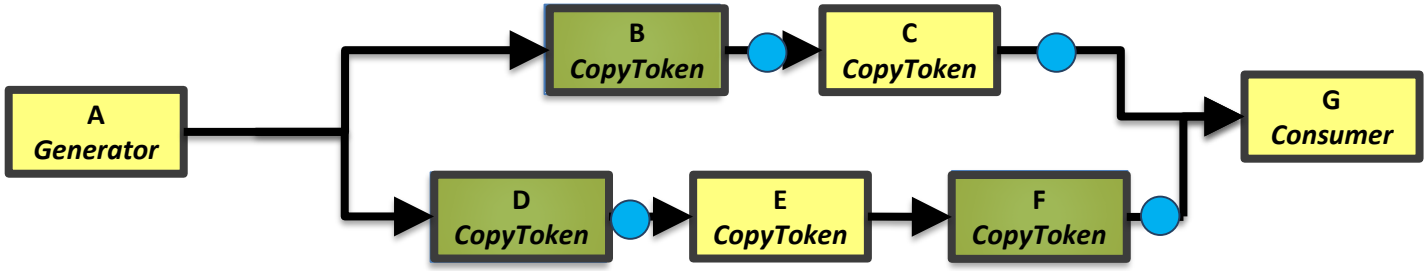
Execution Trace Graph



Execution Gantt Chart

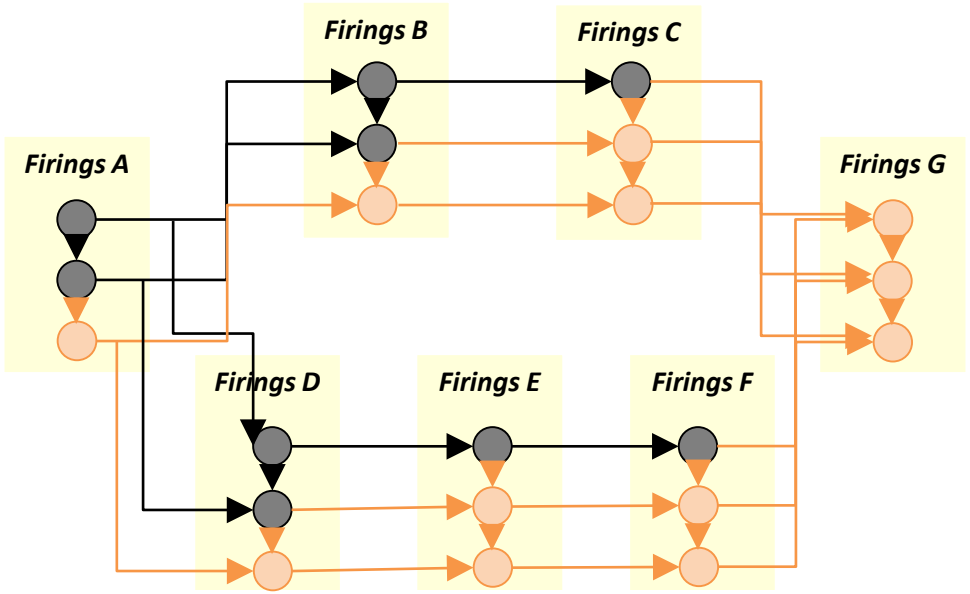


Firing Step: B:copy:2, D:copy:2, F:copy:1

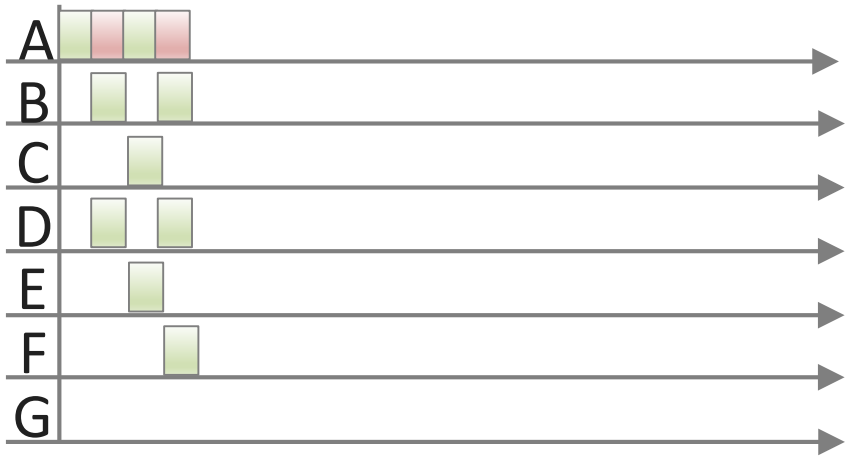


Example 3)

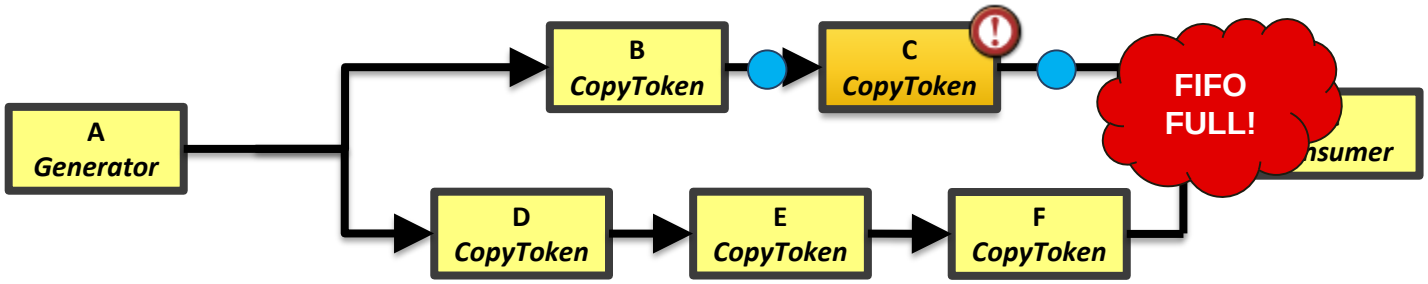
Execution Trace Graph



Execution Gantt Chart

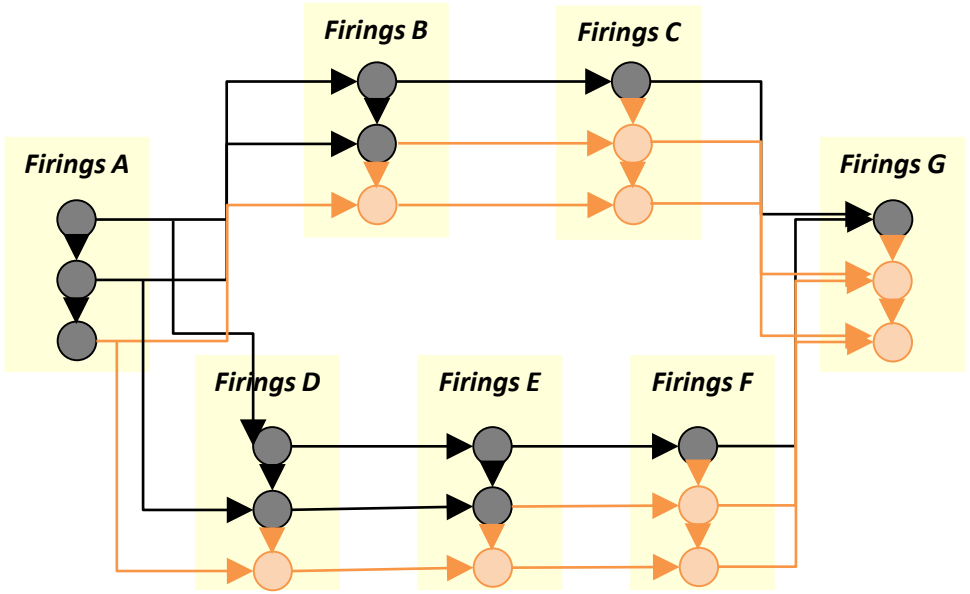


Firing Step: B:copy:2, D:copy:2, F:copy:1

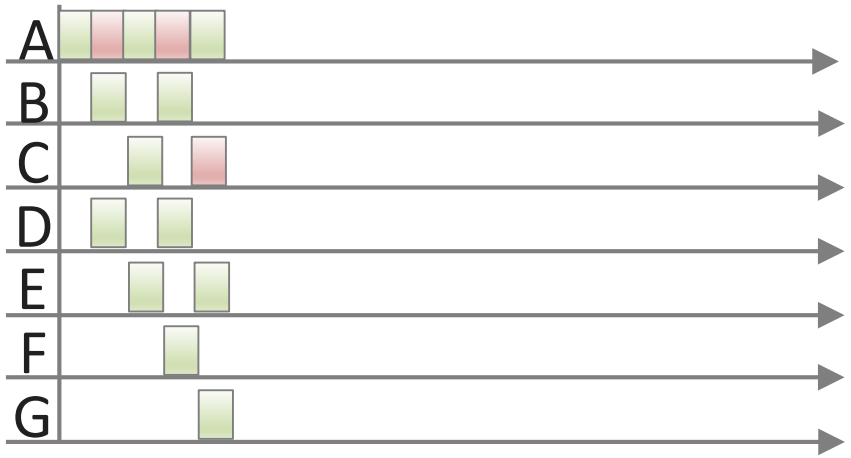


Example 3)

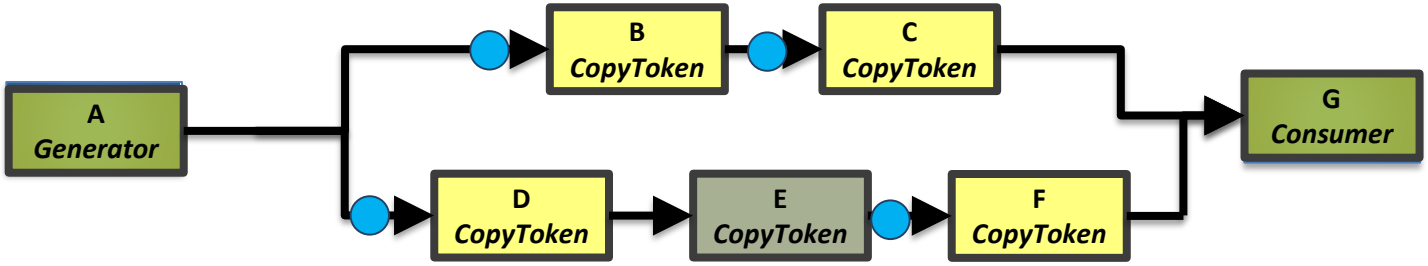
Execution Trace Graph



Execution Gantt Chart

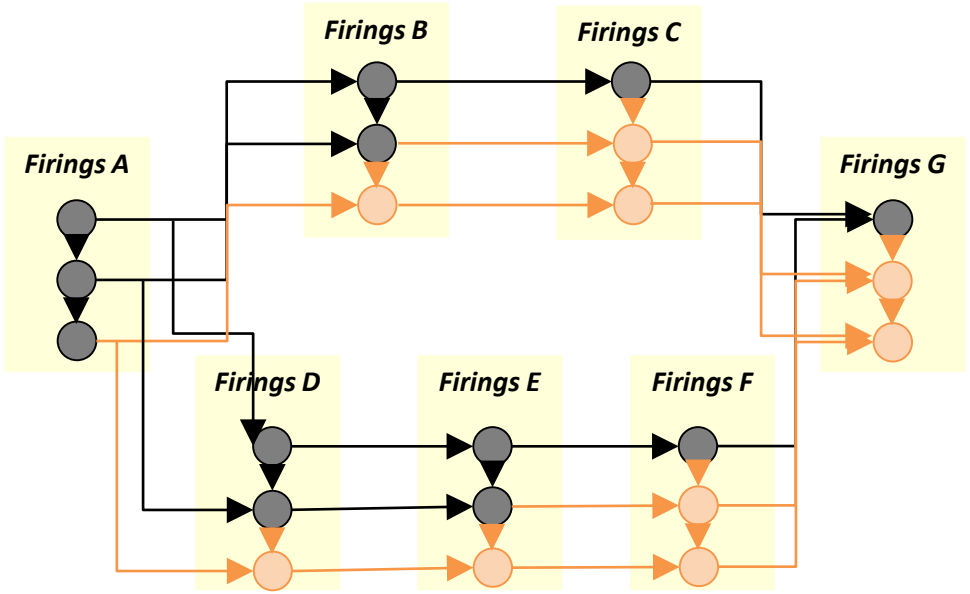


Firing Step: A:gen:3, E:copy:2, G:cons:1

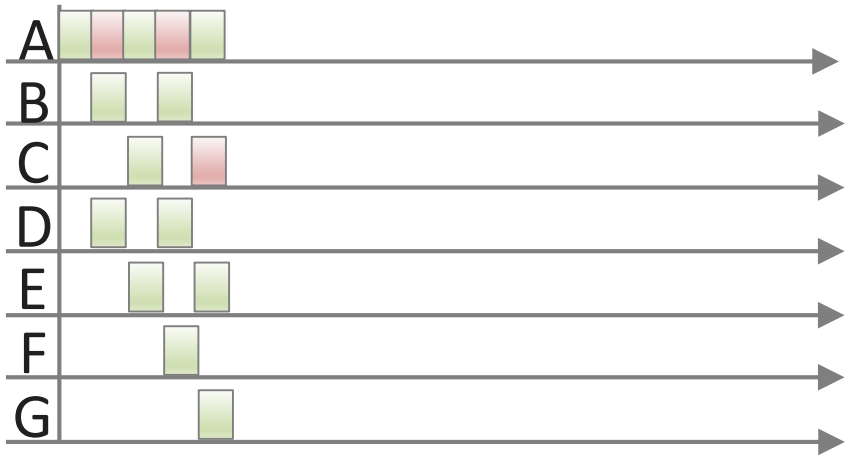


Example 3)

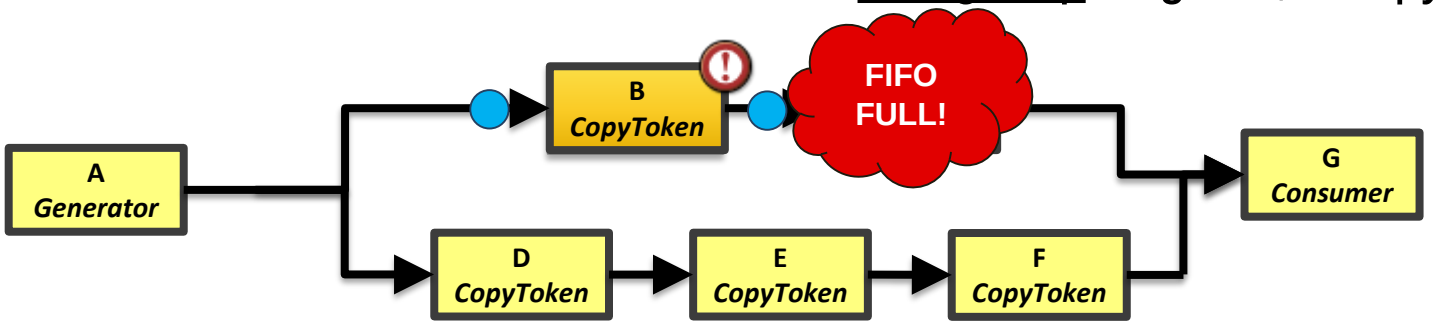
Execution Trace Graph



Execution Gantt Chart

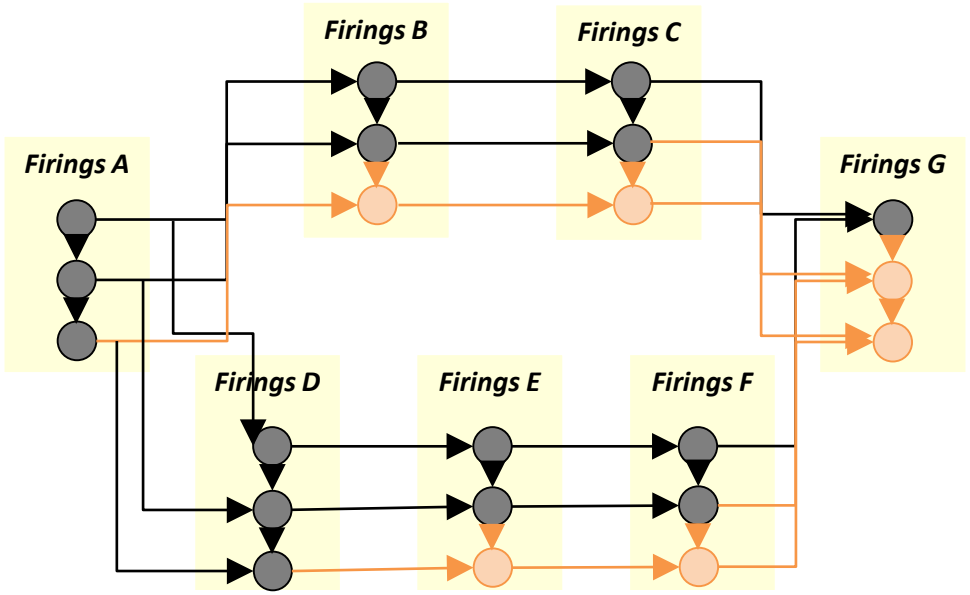


Firing Step: A:gen:3, E:copy:2, G:cons:1



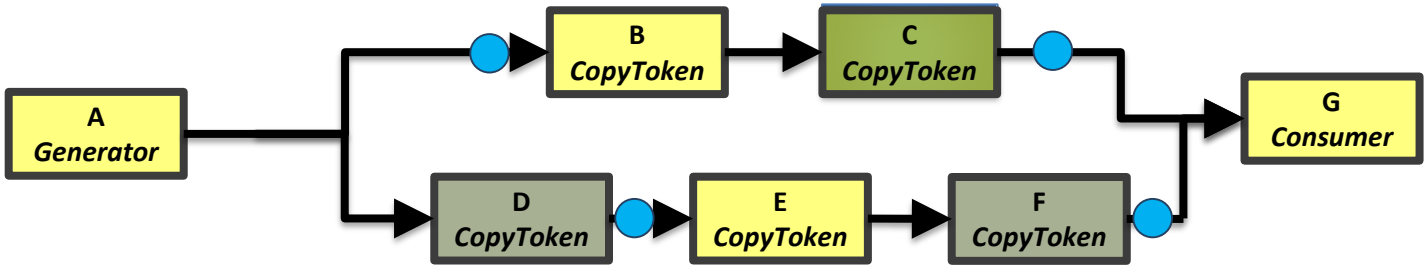
Example 3)

Execution Trace Graph



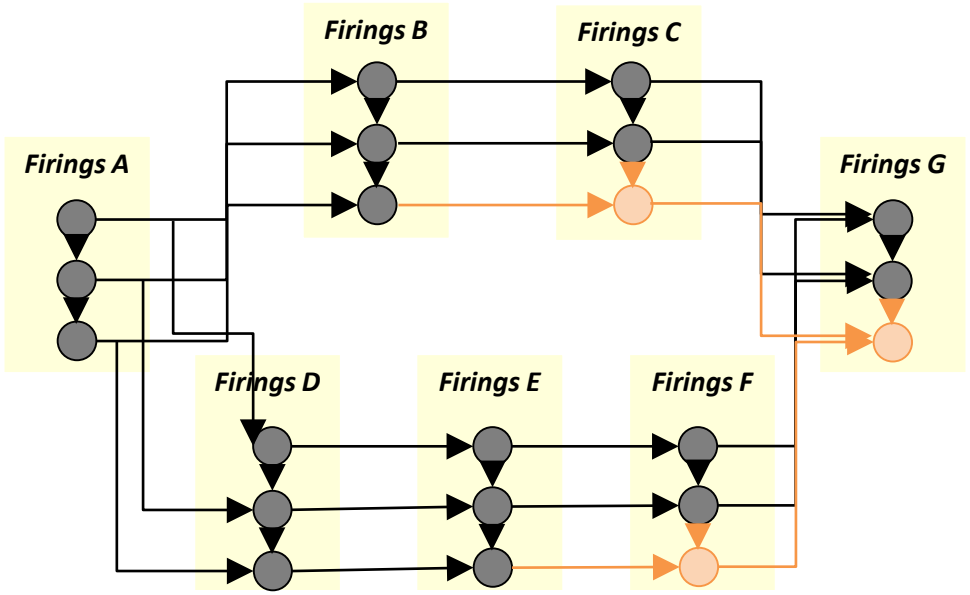
Execution Gantt Chart

Firing Step: C:copy:2, D:copy:3, F:copy:2

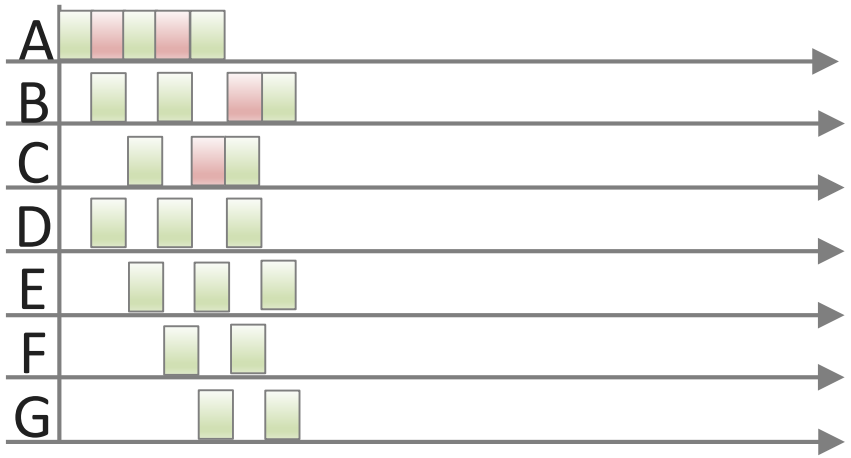


Example 3)

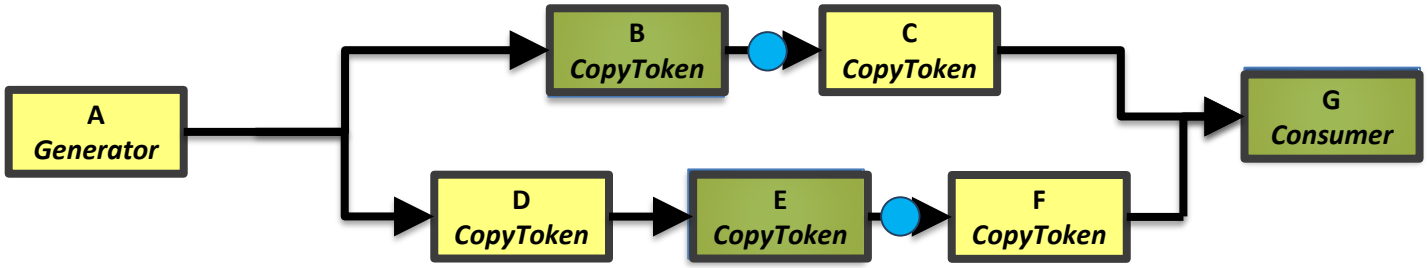
Execution Trace Graph



Execution Gantt Chart

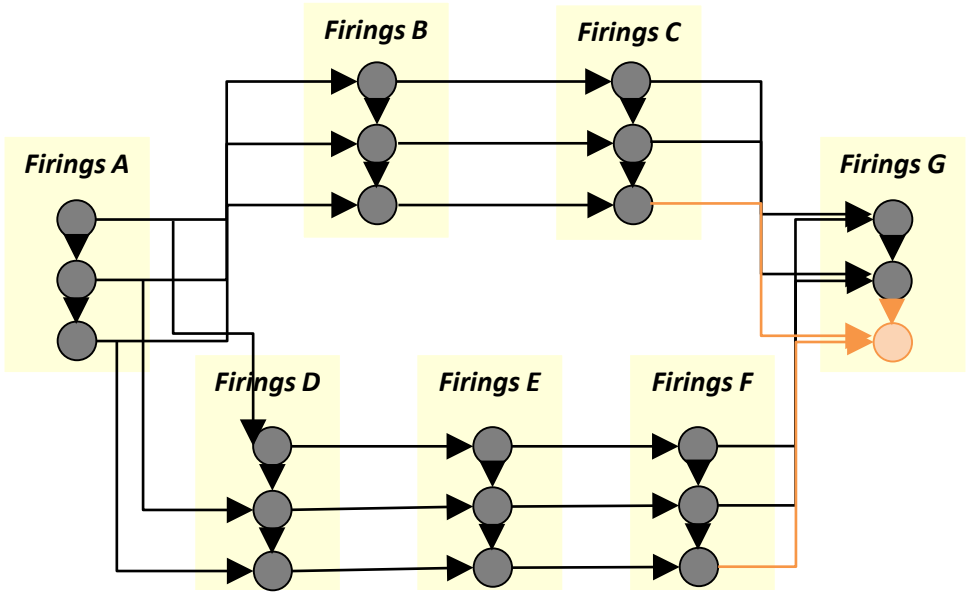


Firing Step: B:copy:3, E:copy:3, G:cons:2

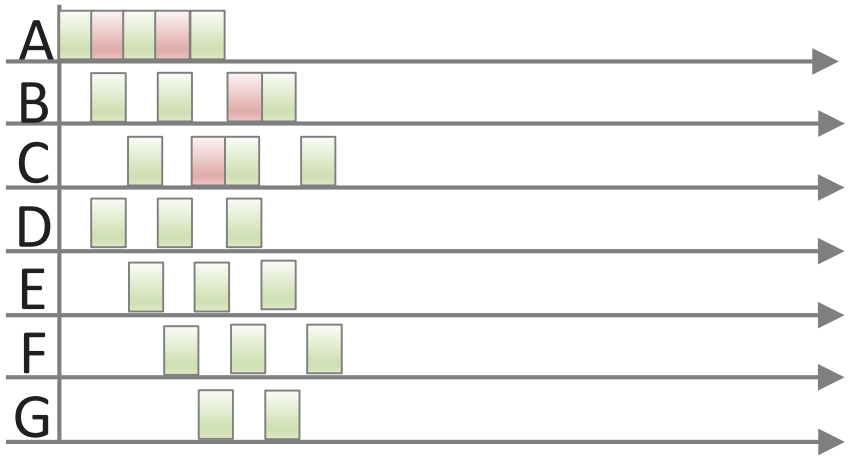


Example 3)

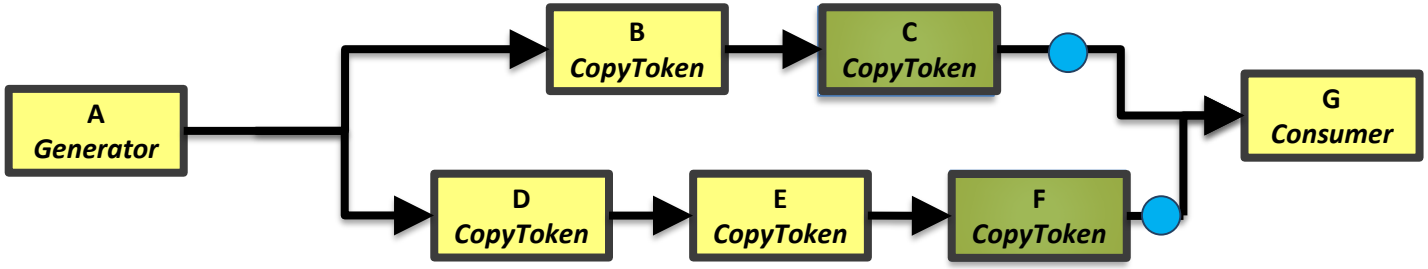
Execution Trace Graph



Execution Gantt Chart

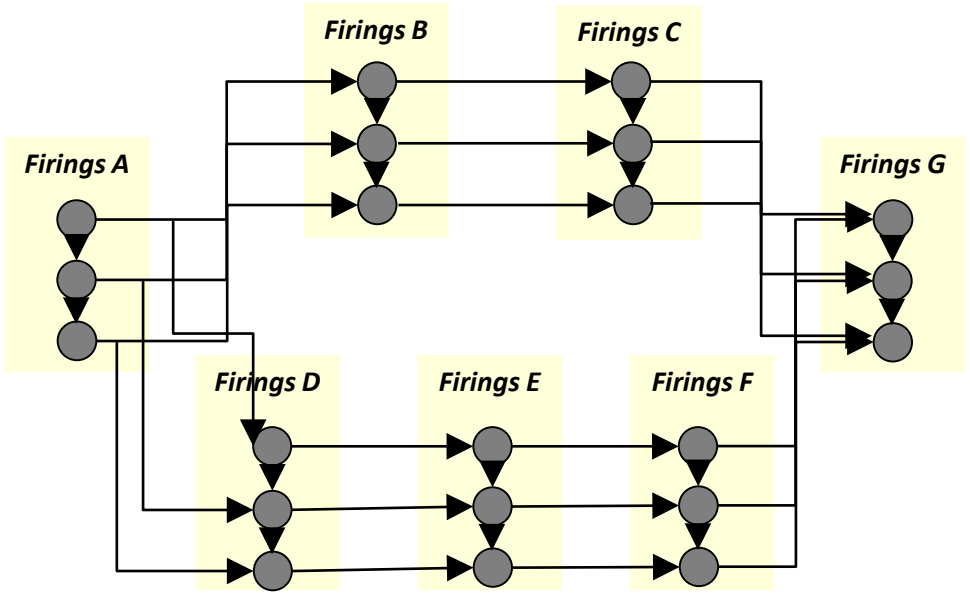


Firing Step: C:copy:3, F:copy:3

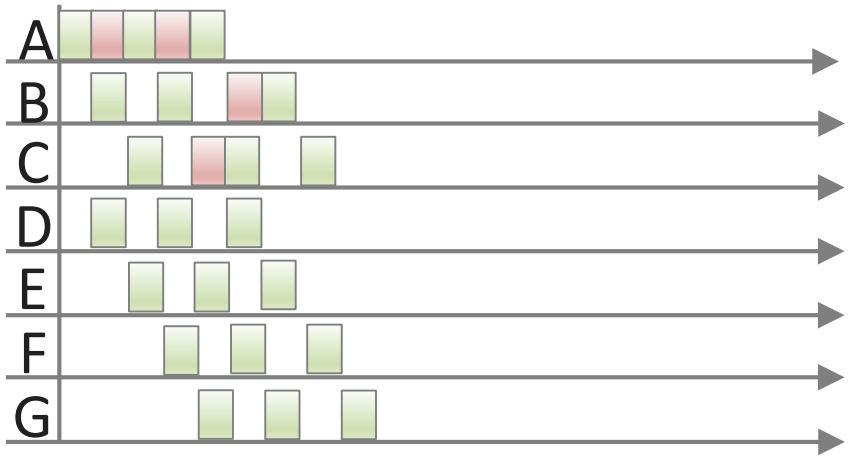


Example 3)

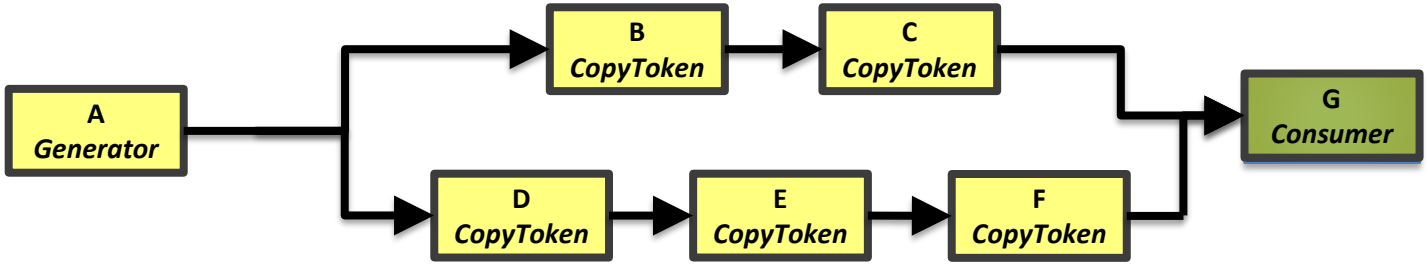
Execution Trace Graph



Execution Gantt Chart

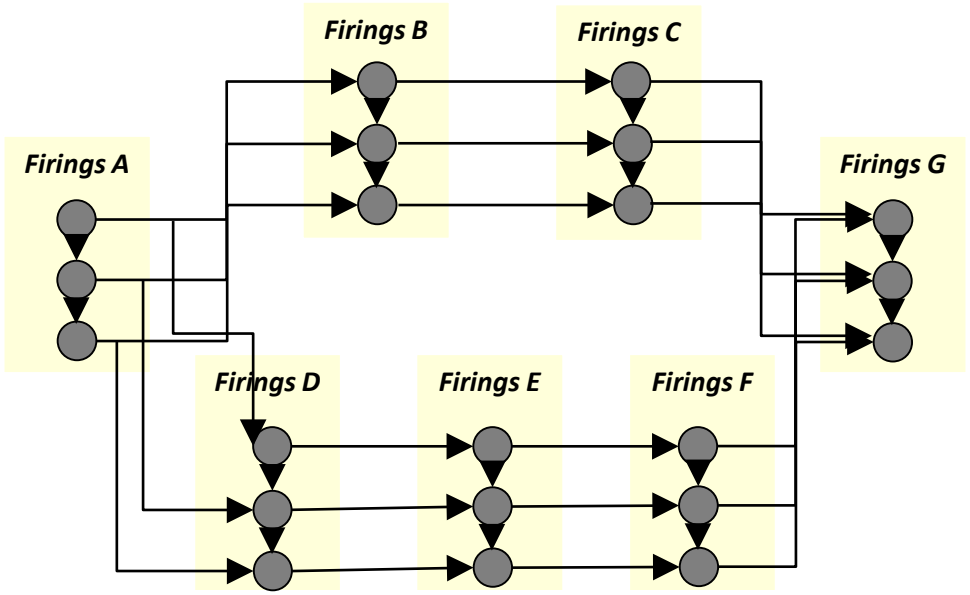


Firing Step: G:cons:3

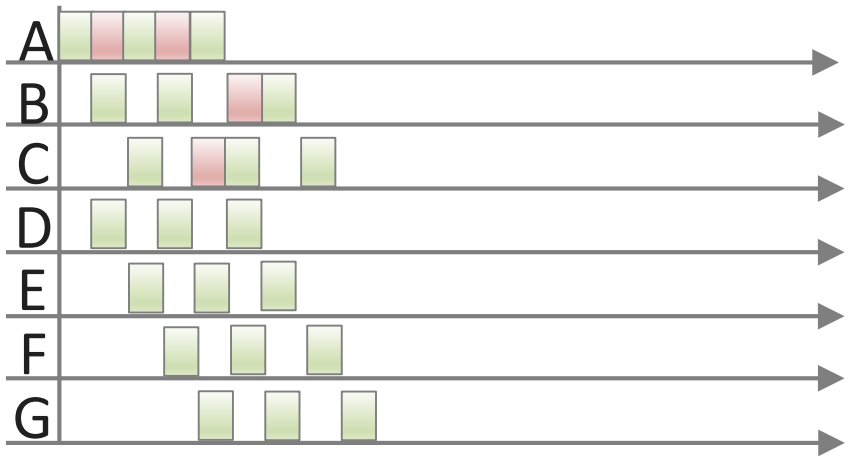


Example 3)

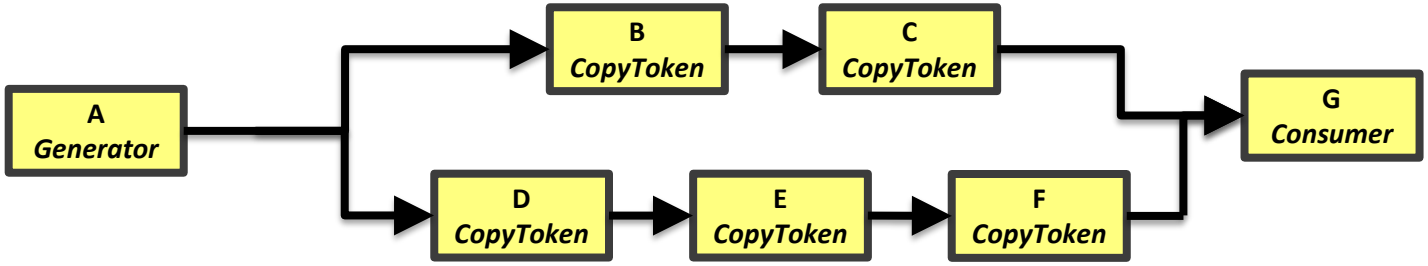
Execution Trace Graph



Execution Gantt Chart

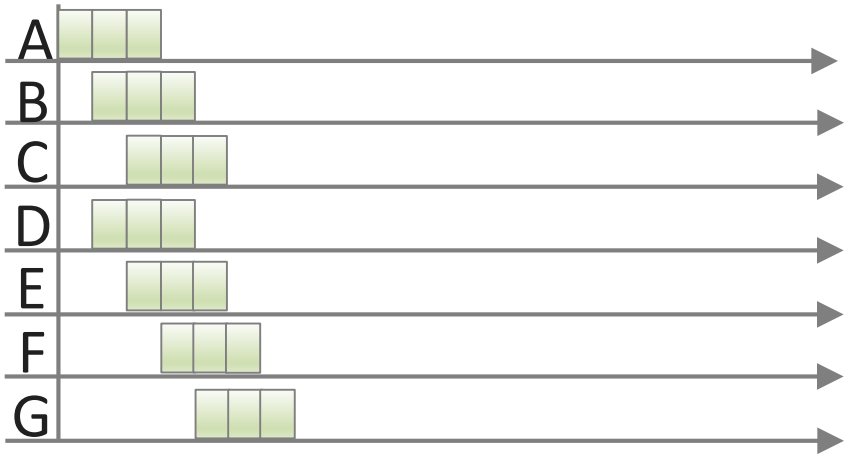


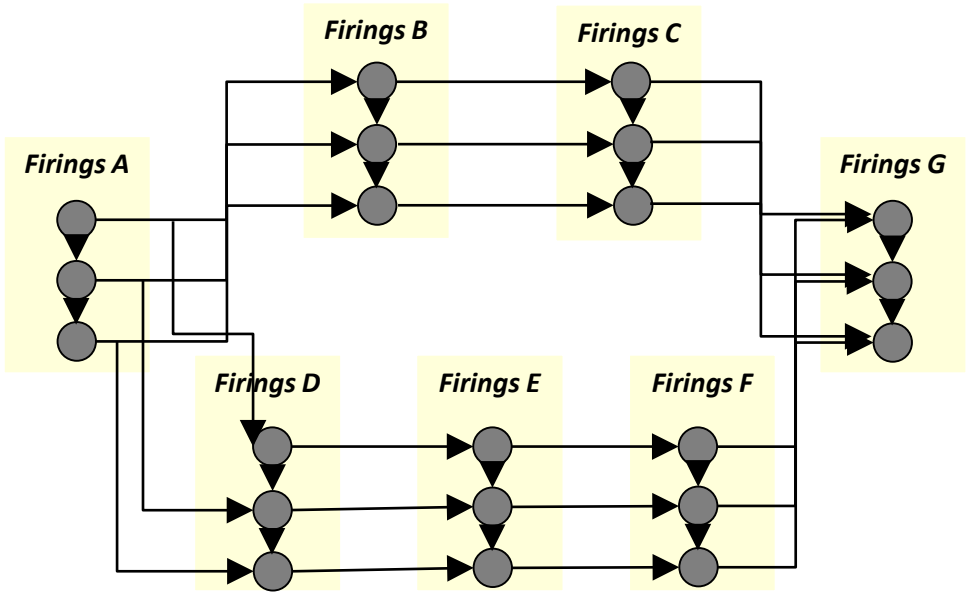
At the end of the execution



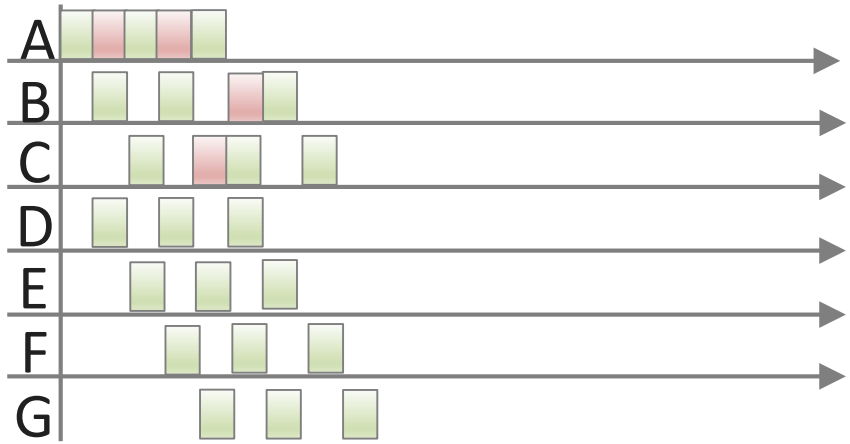
The same resulting trace, different execution times

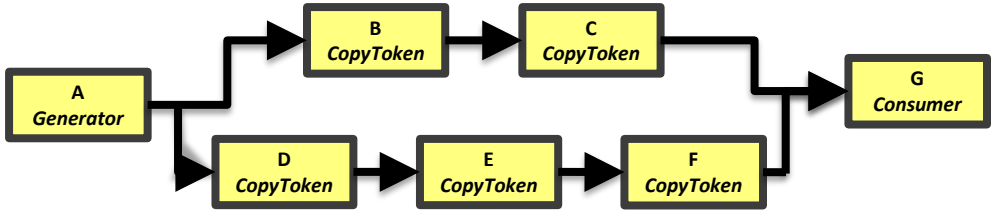
Example 2 (infinite buffer size)





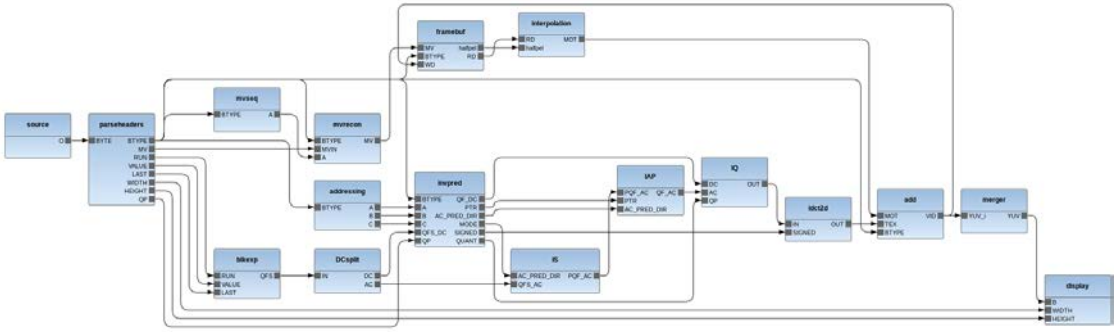
Example 3 (minimal buffer size)





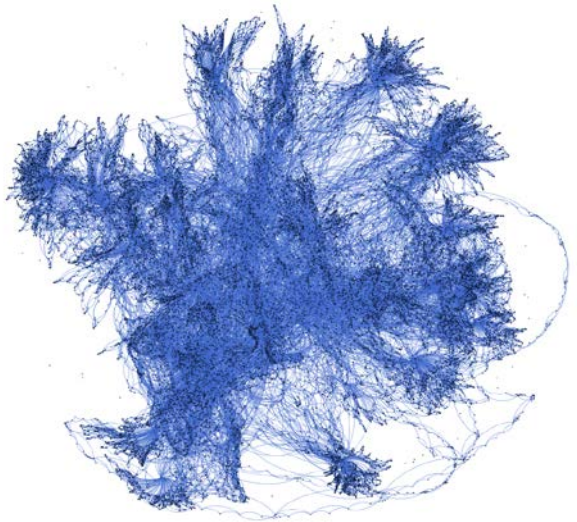
Execution trace graph

- Structural model of a dynamic execution
- Abstract from time
- Model for ***Design Space Exploration***
- Generated for a specific input stimulus
- Carries the information about the potential parallelism



dataflow network: MPEG4-SP decoder

17 actors, 38 buffers



ETG for a few frames of input stimulus

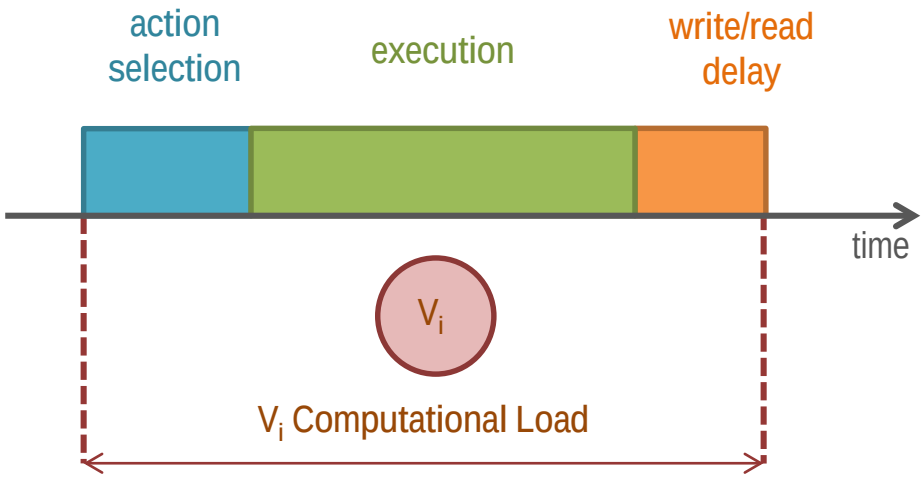
176 649 nodes, 1 609 543 arcs

Performance Estimation

Execution Trace Post-Processing

- For each fired action *(i.e. node of the execution trace)* its Computational Load is measured
- The dependencies set is extended according to the partitioning/scheduling configuration

→ Weighted Execution Trace



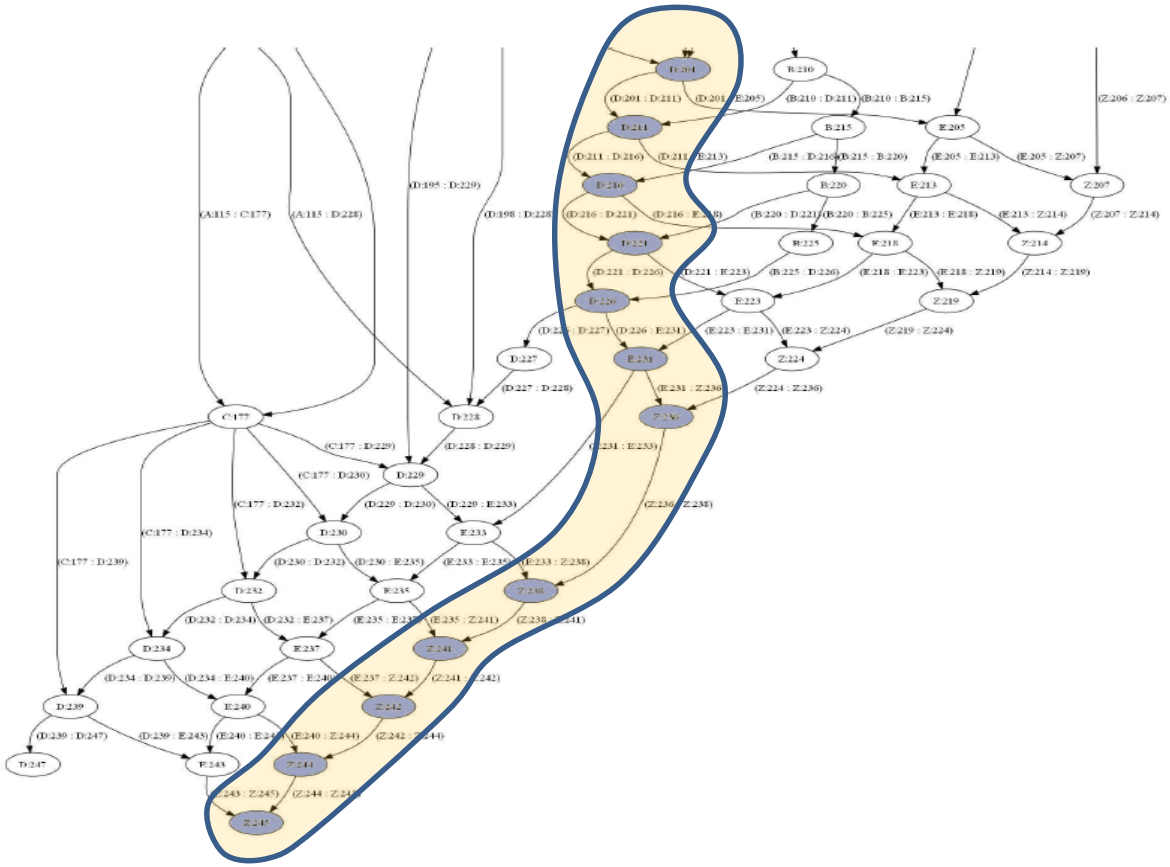
Execution Trace Critical Path

Critical Path Algorithm O(n)

Determine the design bottlenecks for improving the final throughput

- Most Critical Actors and Actions
- Most Critical Buffers *(only for bounded buffer configuration analysis)*

Definition – Critical path:
the longest time-weighted sequence of events from the start of the program to its termination.

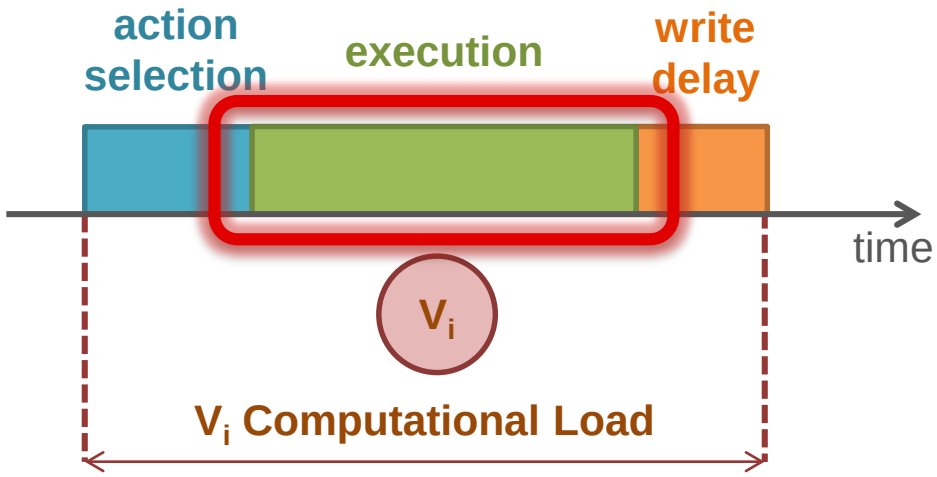


Critical Path Metrics

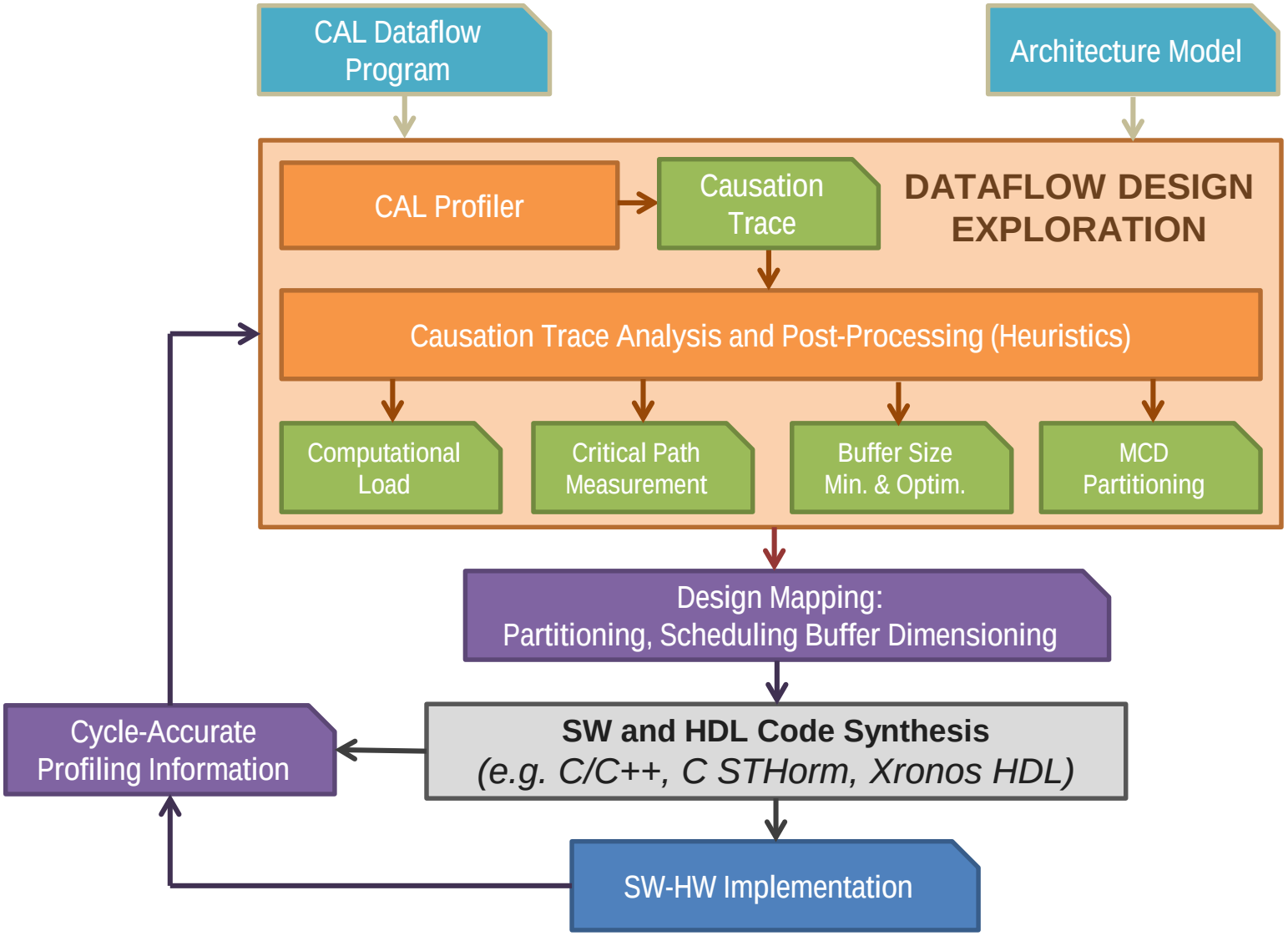
Critical Path Analysis Techniques:

- Critical Actions/Actors Ranking
- Impact Analysis
- Critical Path Gradient *(i.e. refactoring directions)*

NOTE: these metrics are used to estimate the possible design improvements obtained reducing the algorithmic complexity of an action.



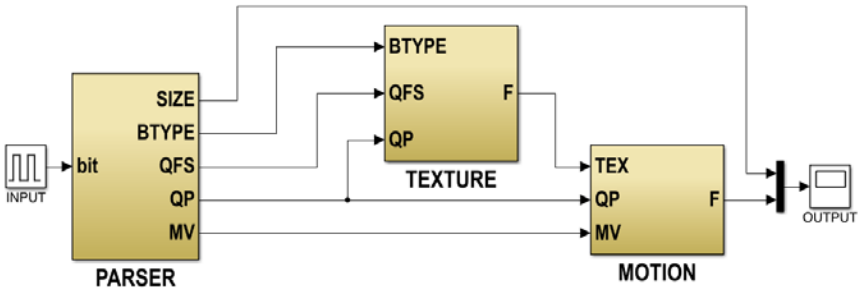
The Co-Exploration Framework



Experimental Results: Design Case

Application:
MPEG4-SP decoder, RVC-CAL

Platforms:
STMicroelectronics STHorm
Virtex-5 FPGA



Input Sequences:
4 different 10-frame QCIF bit streams *(Akyio, Foreman, Suzie, News)*

Execution Trace:
Steps: ≈ 10^6
Dependencies: ≈ 10^7

Critical Path Metrics

Critical Actions Ranking:

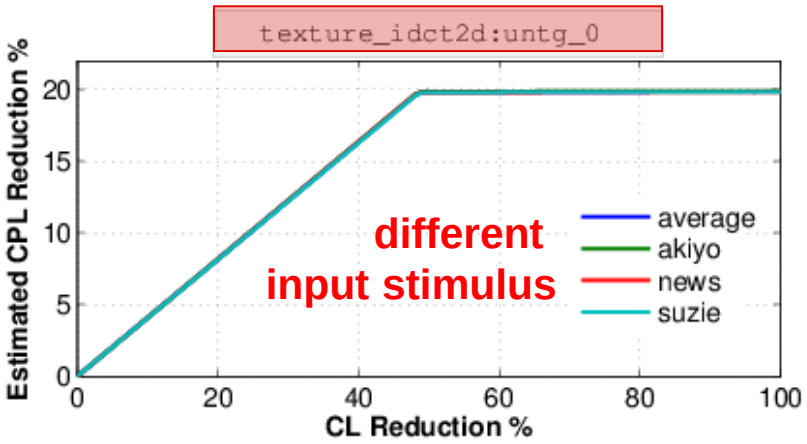
- Actions (and actors) are ranked by their Critical Path Contribution (CPC)

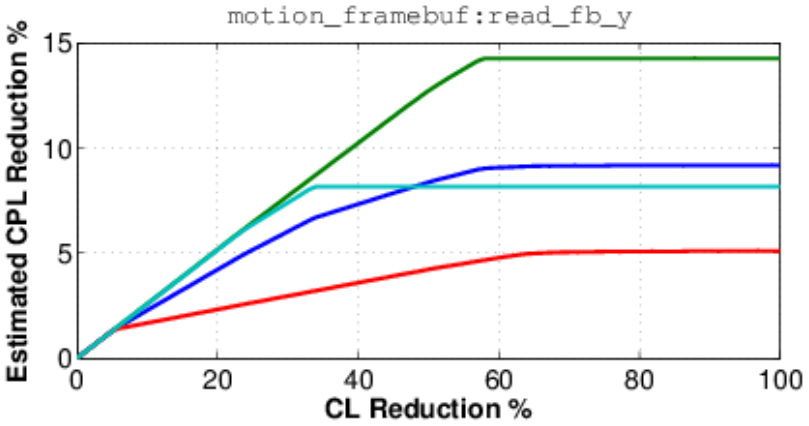
1	DESIGN Results								
2	Executions	Computational Load	CP Executions	CP Computational Load	Criticality Executions	Criticality Computational Load	Slack Max	Slack min	Slack Total
3	132230	20118185	2337	5728838	2337	5728838	5644629	17	2.75992E+11
4									
5	ACTORS Results								
6	Actor	Executions	Computational Load	CP Executions	CP Computational Load (CPP)	Criticality Executions	Criticality Computational Load	Read Tokens	CL / Read Tokens
7	decoder_textu	1712	5717476	1712	5717476	1712	5717476	111280	51.37919
8	decoder_parse	71498	737657	615	6016	615	6016	51800	14.24048
9	decoder_textu	1712	2939004	1	1765	1	1765	111280	26.41089
10	decoder_motic	8137	2723065	1	1666	1	1666	197637	13.77811
11	decoder_motic	5955	1121040	1	577	1	577	264617	4.23646
12	decoder_textu	1712	581628	1	336	1	336	118128	4.92371
13	decoder_textu	9374	647916	1	327	1	327	115523	5.60855
14	decoder_parse	9163	1140702	1	324	1	324	22350	51.03812
15	decoder_textu	1712	549552	1	321	1	321	111280	4.93846
16	source	6481	84209	3	30	3	30	0	-
17	display	1495	2541480	0	0	0	0	190090	13.36988
18	decoder_motic	2376	1221512	0	0	0	0	194832	6.26957
19	decoder_parse	10903	112944	0	0	0	0	4509	25.04857
20									
21	ACTIONS Results								
22	Actor	Action	Executions	Computational Load	CP Executions	CP Computational Load (CPP)	Criticality Executions	Criticality Computational Load	Read Tokens
23	decoder_textu	untagged_0	1712	5717476	1712	5717476	1712	5717476	111280
24	decoder_parse	untagged_0	2248	20232	395	3555	395	3555	2248
25	decoder_textu	untagged_0	1712	2939004	1	1765	1	1765	111280
26	decoder_motic	untagged_0	495	816120	1	1666	1	1666	190080
27	decoder_parse	untagged_1	41982	419820	150	1500	150	1500	41982
28	decoder_motic	combine	1118	645076	1	577	1	577	143104
29	decoder_parse	vld_code	6708	137573	24	496	24	496	6708
30	decoder_textu	untagged_0	1712	581628	1	336	1	336	118128
31	decoder_textu	getdc_intra	594	193364	1	327	1	327	38016
32	decoder_parse	last	1712	554688	1	324	1	324	5136
33	decoder_textu	untagged_0	1712	549552	1	321	1	321	111280
34	decoder_parse	dct_coeff	6856	47992	24	168	24	168	0
35	decoder_parse	vol_time_inc_res	1	77	1	77	1	77	0

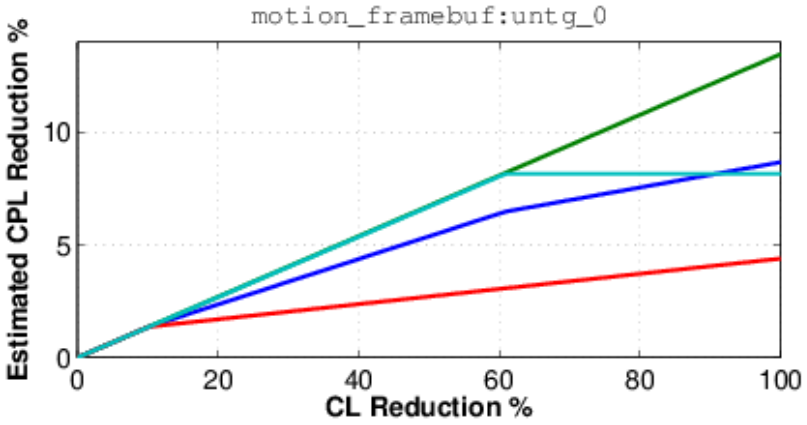
Critical Path Metrics

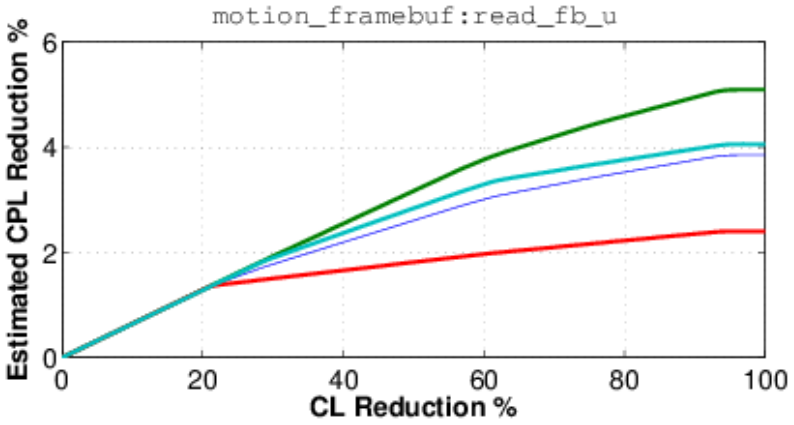
Impact Analysis: Which action need attention

Most Critical Action



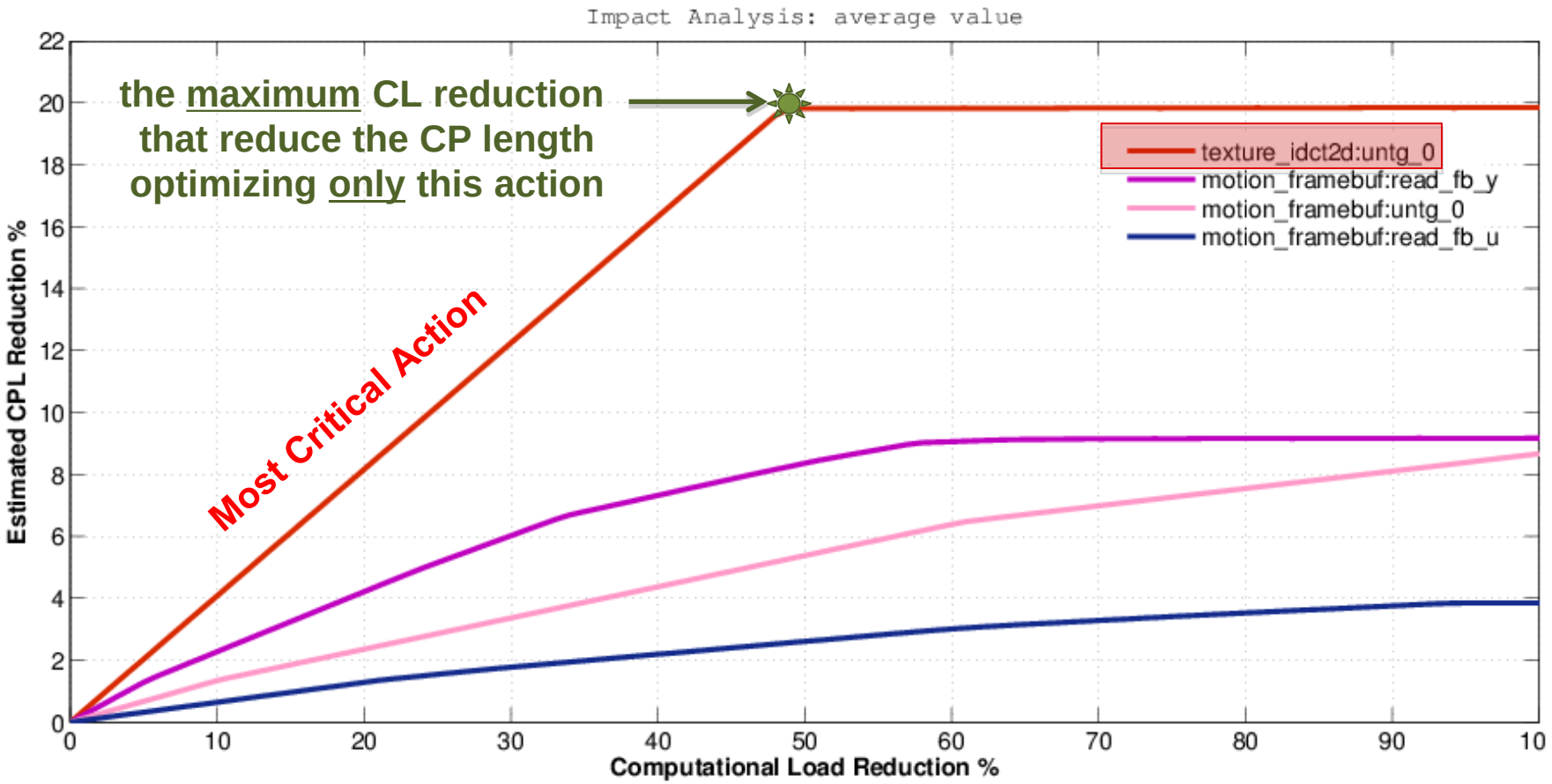






Critical Path Metrics

Impact Analysis: Which action need attention



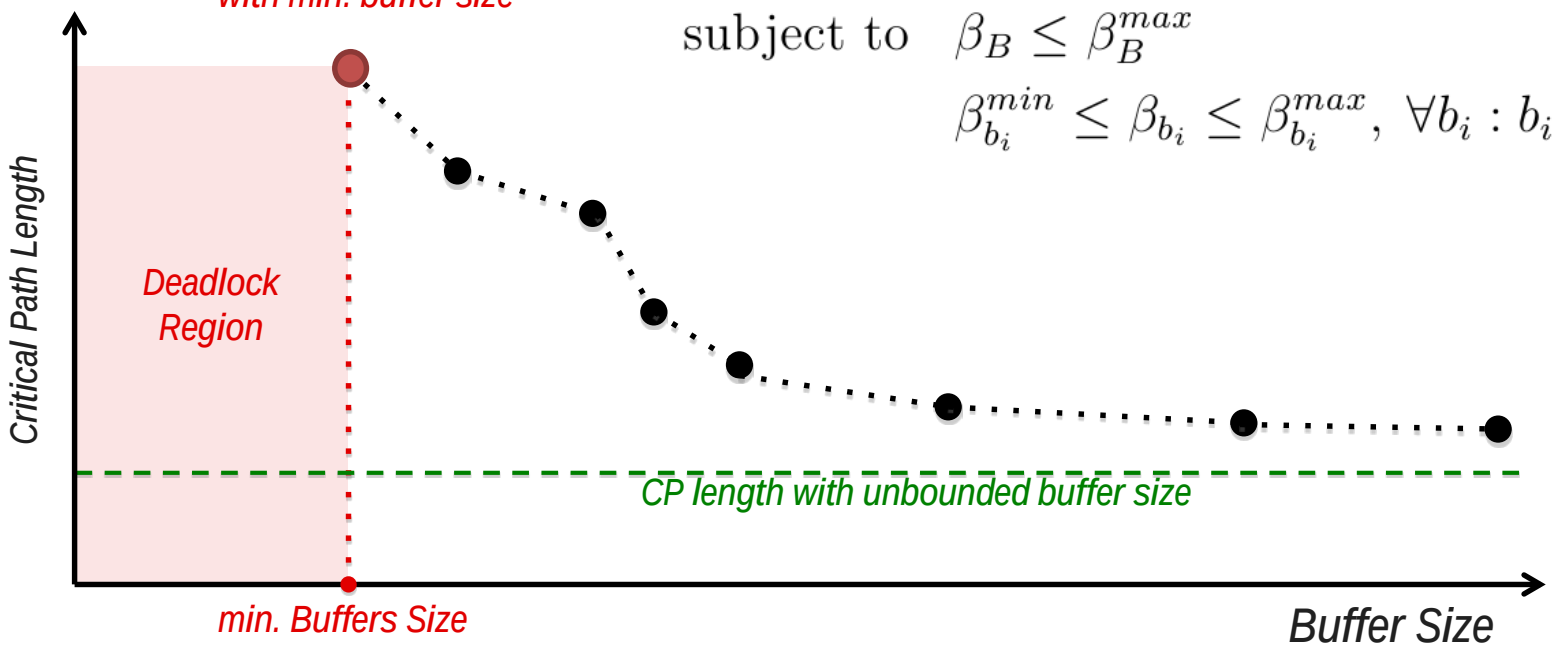
Buffer Dimensioning Problem: Formulation

Objective: find a trade-off between the buffer size and the critical path length
→ Minimum buffer size configuration for a given throughput

$$\underset{\beta_{b_i}}{\text{minimize}} \quad \begin{cases} CP(\beta_B, \rho, \sigma) \\ \beta_B = \sum \beta_{b_i}, \ \forall b_i : b_i \in B \end{cases}$$

$$\text{subject to} \quad \beta_B \leq \beta_B^{max}$$

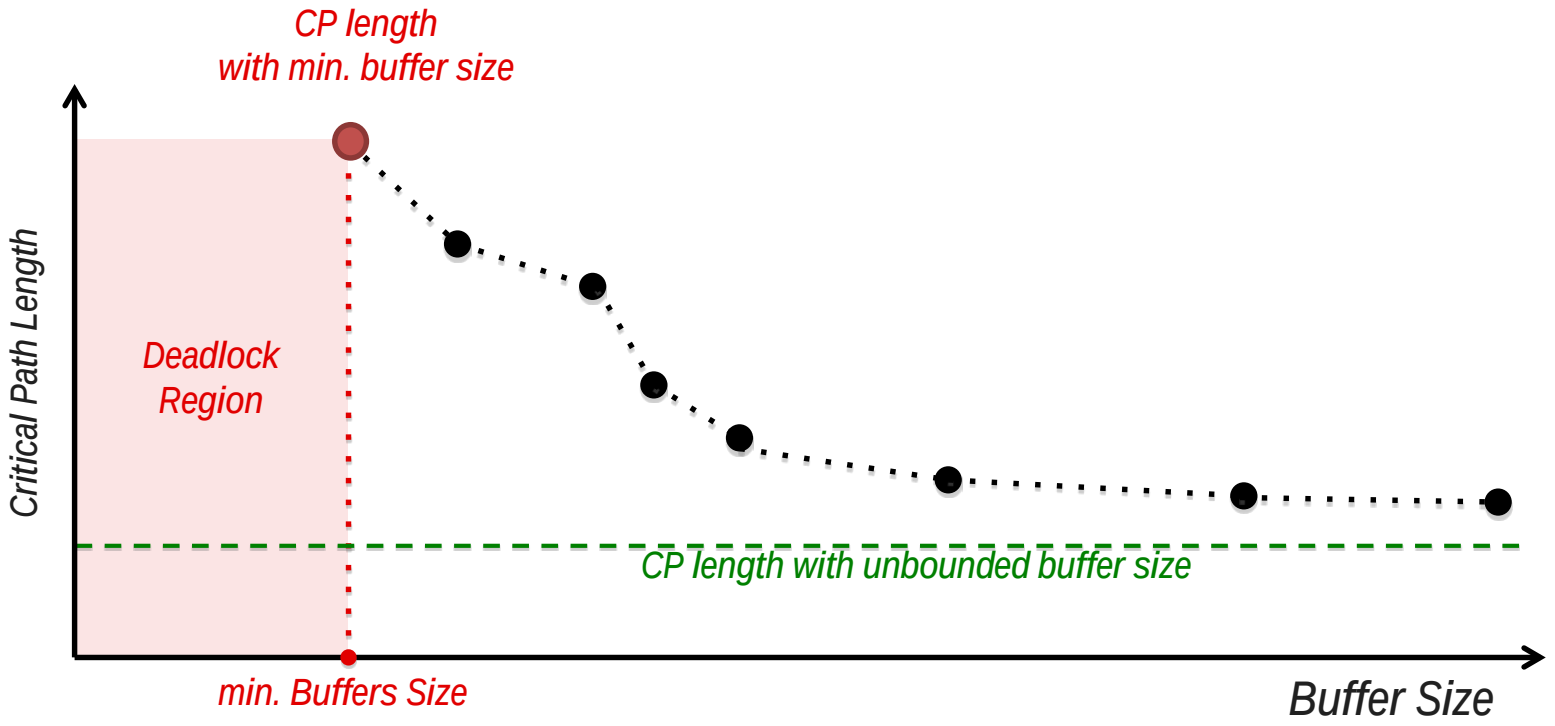
$$\beta_{b_i}^{min} \leq \beta_{b_i} \leq \beta_{b_i}^{max}, \ \forall b_i : b_i \in B$$



Heuristic Approach

Divide the problem onto two sub-problems:

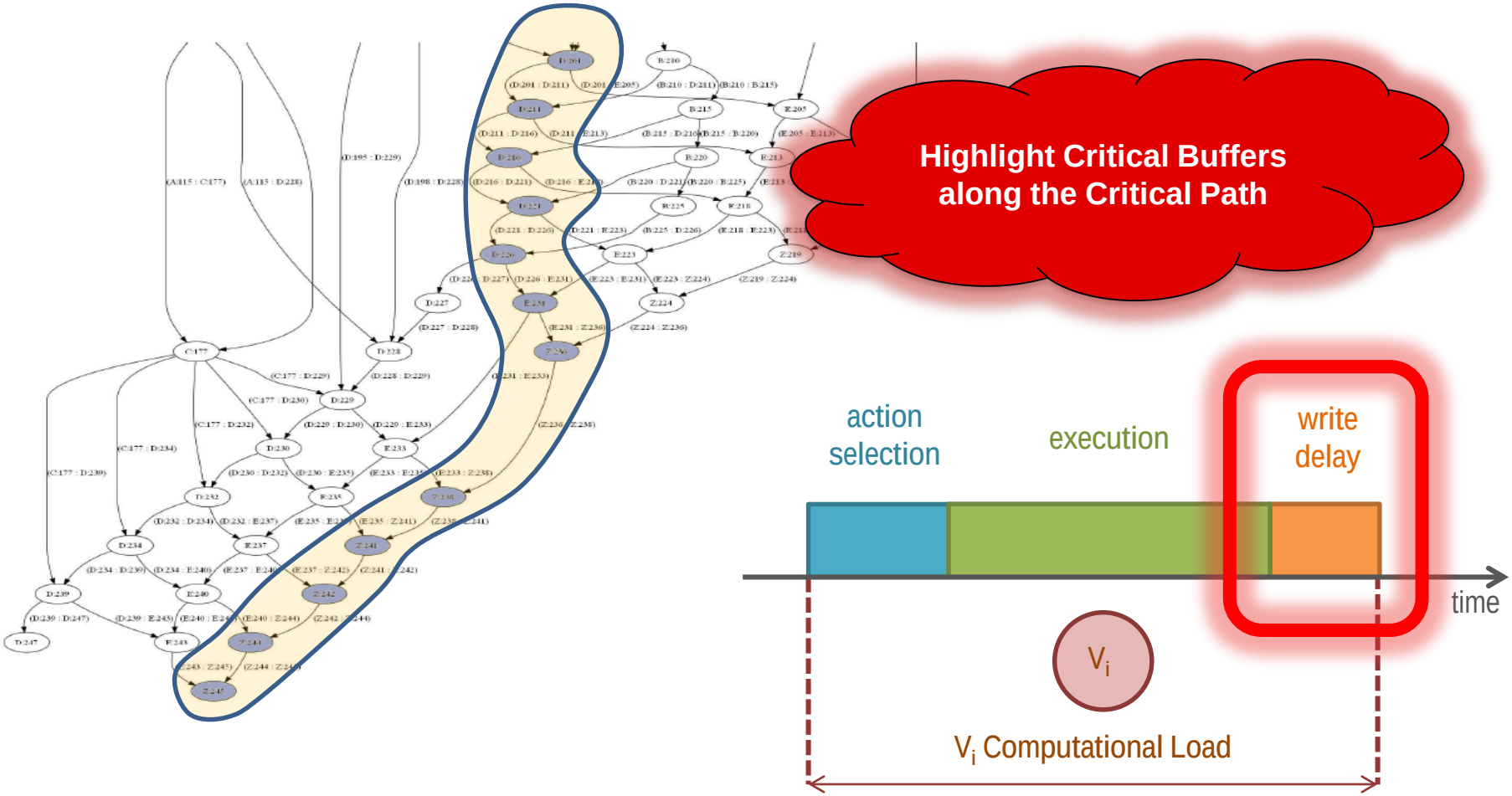
1. Compute the minimal buffer size
2. Optimize only critical buffers *(i.e. the buffers that blocks actions in the current CP)*



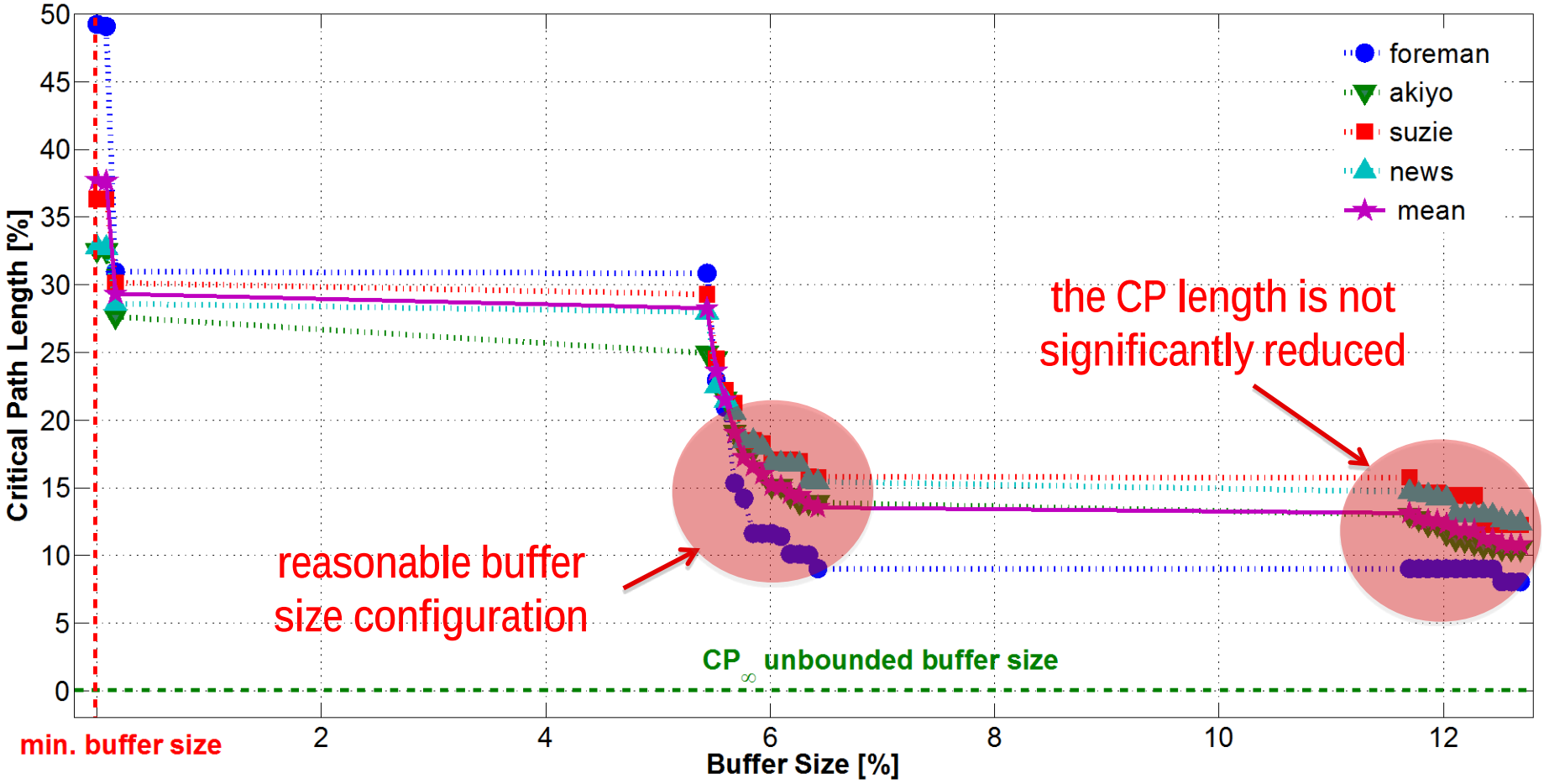
Critical Path Analysis

Highlight the Critical Buffer:

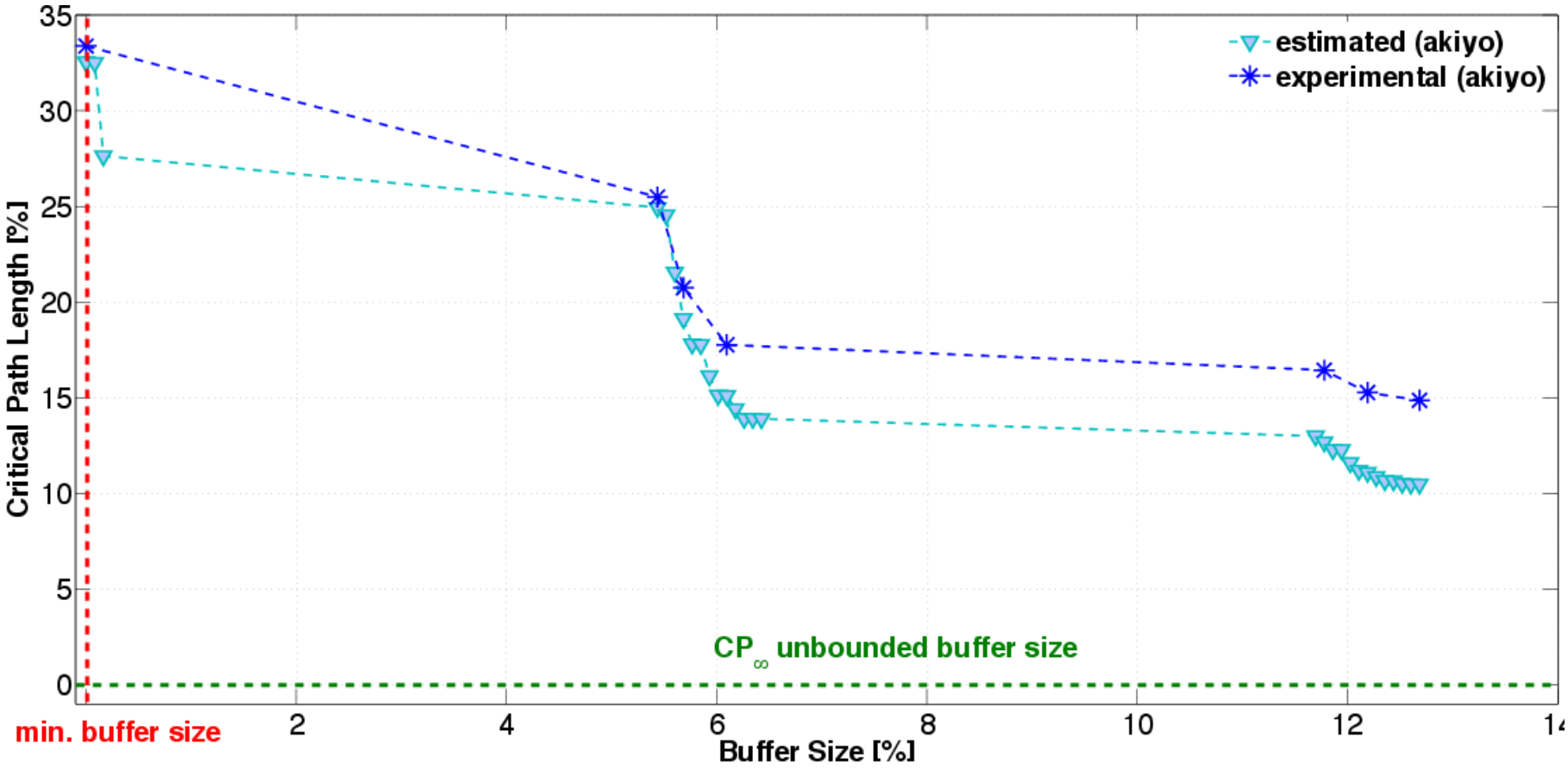
the one that introduces the highest total blocking latency time along the Critical Path



TURNUS Estimated Results

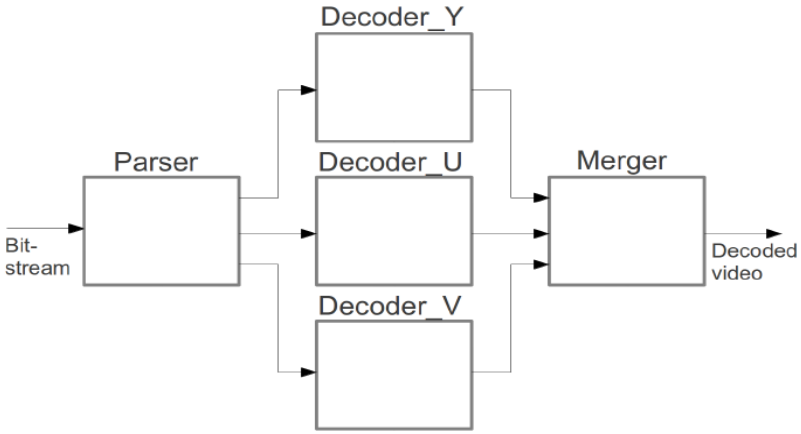


Estimated Results v.s. Implementation Results



Design case: MPEG-4 AVC/H.264 decoder

- Initial performance summary



Platform	Component	*LAT*	*FRE*	*THR*	*REG*	*LUT*	*BRM*	*SLI*
FPGA	Full decoder	660124	29	43	83804	809866	189	202467
	Decoder_Y	650710	56	86	24809	46288	77	11572
	Decoder_U/V	95100	79	831	12389	24471	78	6118
	Parser	134323	29	214	20301	50745	0	12686
CPU	Full decoder	-	2300	59	-	-	-	-
	Parser	-	2300	2327	-	-	-	-

MPEG-4 AVC/H.264 decoder

- Decoder Y design points summary

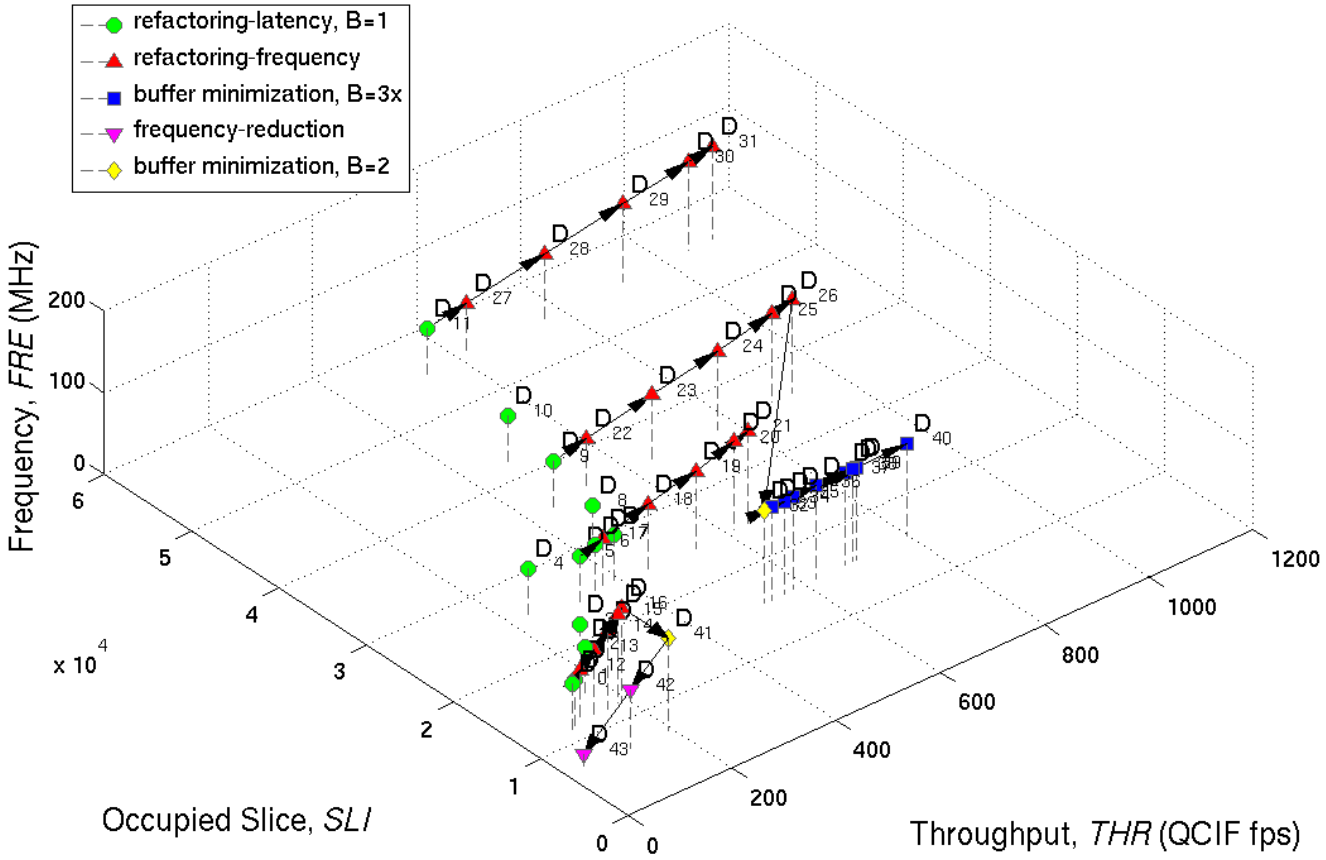
Design point	Technique	Actor	Latency (C.C./MB)
D_0	*original*	-	6573
D_1	Data-packing Redundancy-elimination	picture_buffer_y	5981
D_2	Data-parallelism(2x)	half_quarter_interpolation	3483
D_3	Data-parallelism(3x)	half_quarter_interpolation	2857
D_4	Data-parallelism(4x)	interp_reorder_y add_pix_sat demux_parser_info_y	2407
D_5	Redundancy-elimination	deblocking_filter_y	1797
D_6	Data-parallelism(4x)	picture_buffer_y deblocking_filter_y	1613
D_7	Data-packing	interp_reorder_y	1449
D_8	Task-parallelism(2x)	half_quarter_interpolation	1358
D_9	Task-parallelism(3x)	half_quarter_interpolation	1257
D_{10}	Task-parallelism(4x)	half_quarter_interpolation	1195
D_{11}	Task-parallelism(5x)	half_quarter_interpolation	1054

Design points	Technique
D_0 - D_{31}	Hardware Execution Optimization (HEO)
D_{32}, D_{41} - D_{43}	Trace Execution Minimization (TEM)
D_{33} - D_{40}	Trace Execution Optimization (TEO)

Design points	Actor(s)	Action(s)	# of pipeline stages	$\overline{f_{max}}$ (MHz)
$D_1/D_5/D_9, D_{11}$	*original*	-	-	56
$D_{12}/D_{17}/D_{22}/D_{27}$	half_quarter_interpolation	getPixVal_done	2	64
$D_{13}/D_{18}/D_{23}/D_{28}$	half_quarter_interpolation picture_buffer_y idct_scaler	getPixVal_done writeData_done read_coeff	3 3 4	80
$D_{14}/D_{19}/D_{24}/D_{29}$	picture_buffer_y intrapred_luma16x16	writeData_done write_mode	4 2	96
$D_{15}/D_{20}/D_{25}/D_{30}$	idct_scaler intrapred_luma16x16	read_coeff write_mode	5 4	109
$D_{16}/D_{21}/D_{26}/D_{31}$	idct_scaler intrapred_luma16x16	read_coeff write_mode	8 7	114

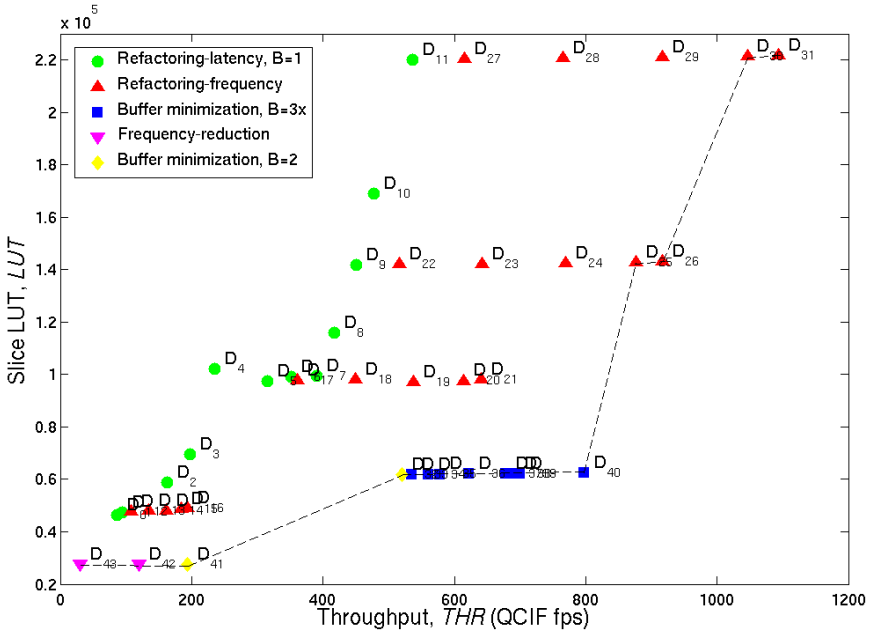
	Design Criteria							
	LAT	*FRE*	*THR*	*REG*	*LUT*	*BRM*	*SLI*	*DSP*
z^{nad}	104396	18	30	24427	27604	42	6901	62
z^{ideal}	650710	114	1093	37790	221781	124	55445	62

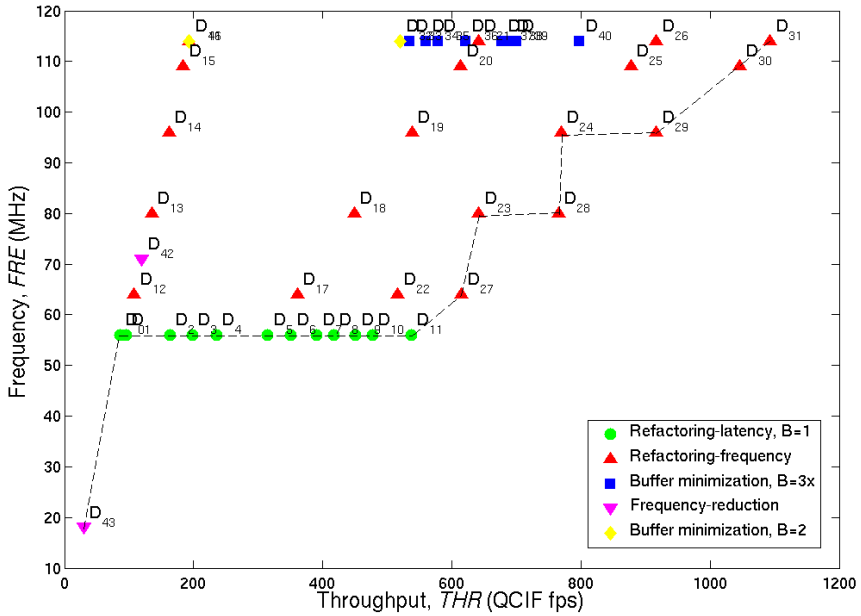
MPEG-4 AVC/H.264 decoder



Case study-2: MPEG-4 AVC/H.264 decoder (cont.)

- Decoder_Y Pareto analysis





- Throughput optimal design points for resource (slice LUT) and power (frequency)

Case study-2: MPEG-4 AVC/H.264 decoder (cont.)

- Design points comparison (original vs best)
  - ~18% code change for ~13x throughput gain
  - Less than ~5% code rewriting, essentially “cut and paste”

	D_0	D_{31}	Difference (%)
Throughput (QCIF fps)	86	1093	1271
Occupied slice	11572	55445	478
Frequency (MHz)	56	114	204
# of actor instances	33	47	142
# of actions	577	785	136
# of networks/sub-networks	9	10	111
# of buffer interconnections	86	170	198
# of code lines (CAL actors)	6657	7845	118
# of code lines (CAL networks)	1429	1681	118
# of code lines (generated C)	26195	38796	148
# of code lines (generated HDL)	137981	196727	143

Case study-2: MPEG-4 AVC/H.264 decoder (cont.)

- Comparison with related works

	Platform	Specification language	Frequency (MHz)	Throughput (QCIF fps)	Resource (slice/gates)
This work (THR_{min})	i7+Virtex-5	CAL	2300/18	30	18249
This work (THR_{max})	i7+Virtex-5	CAL	2300/114	1092	66833
[58]	Virtex-4	RTL	115	1092	44544
[59]	ASIC/FPGA	RTL	-	2045	-
[114]	Virtex-4	SystemC	110	30	23257
[119]	ARM+ASIC	C/RTL	140/10	7.4	28826

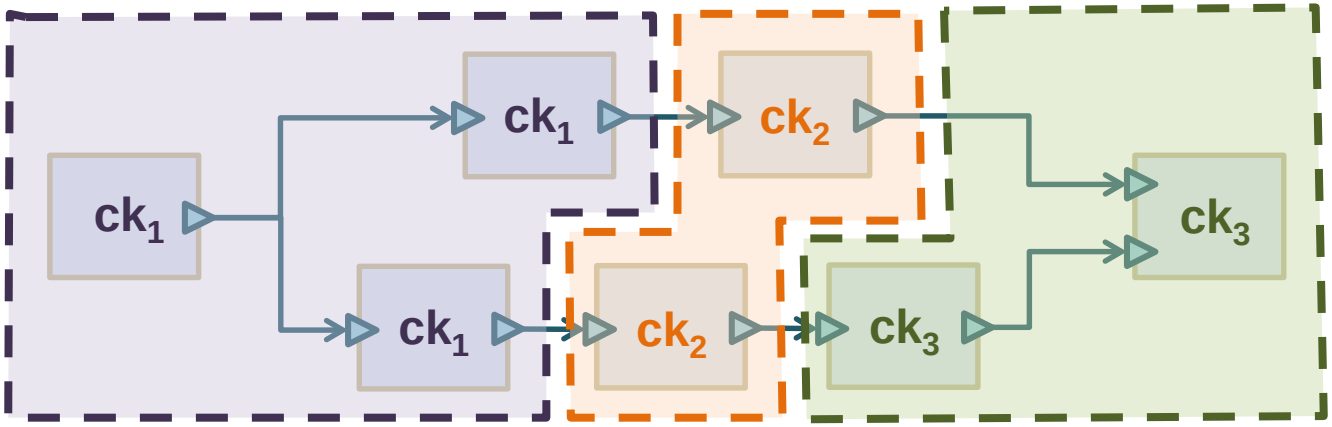
- Compares favorably with similar works
- Rapidly obtain design alternatives in exploration space

[58] FastVdo http://fastvdo.com/FV264. FV264-H.264/AVC ASIC IP CORE.
[59] CoreEL Technologies http://www.coreel.com/pages/productsDigitalVideoH264CBPDecode.aspx, H.264 CBP Decoder.
[114] M. Thadani et al. “ESL flow for a hardware H.264/AVC decoder using TLM-2.0 and high level synthesis: a quantitative
study.” *In Proceedings of SPIE - The International Society for Optical Engineering*, 2009.
[119] S. . Wang, et al. “A software-hardware co-implementation of MPEG-4 Advanced Video Coding (AVC) decoder with block level pipelining.” *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, 2005..

Multi Clock Domain Partitioning

Objective: partitioning the dataflow applications onto a MCD architectures

- Reduce the overall **power consumption**
- Without impacting the *(nominal)* **performances**



Partitioning Example

Multi Clock Domain Partitioning

Objective: partitioning the dataflow applications onto a MCD architectures

- Reduce the overall **power consumption**
- Without impacting the *(nominal)* **performances**

Find a mapping configuration Actor → Clock Domain *(i.e. clock frequency)*

$$\phi : \mathcal{A} \rightarrow \mathcal{F}$$

such that:

$$\text{minimize} \quad \sum_{a \in \mathcal{A}} c_a \, \phi(a)$$

$$\text{subject to} \quad T(\mathcal{A}, \mathcal{B}) \leq L$$

we simplify the relation **power - frequency** with a linear function

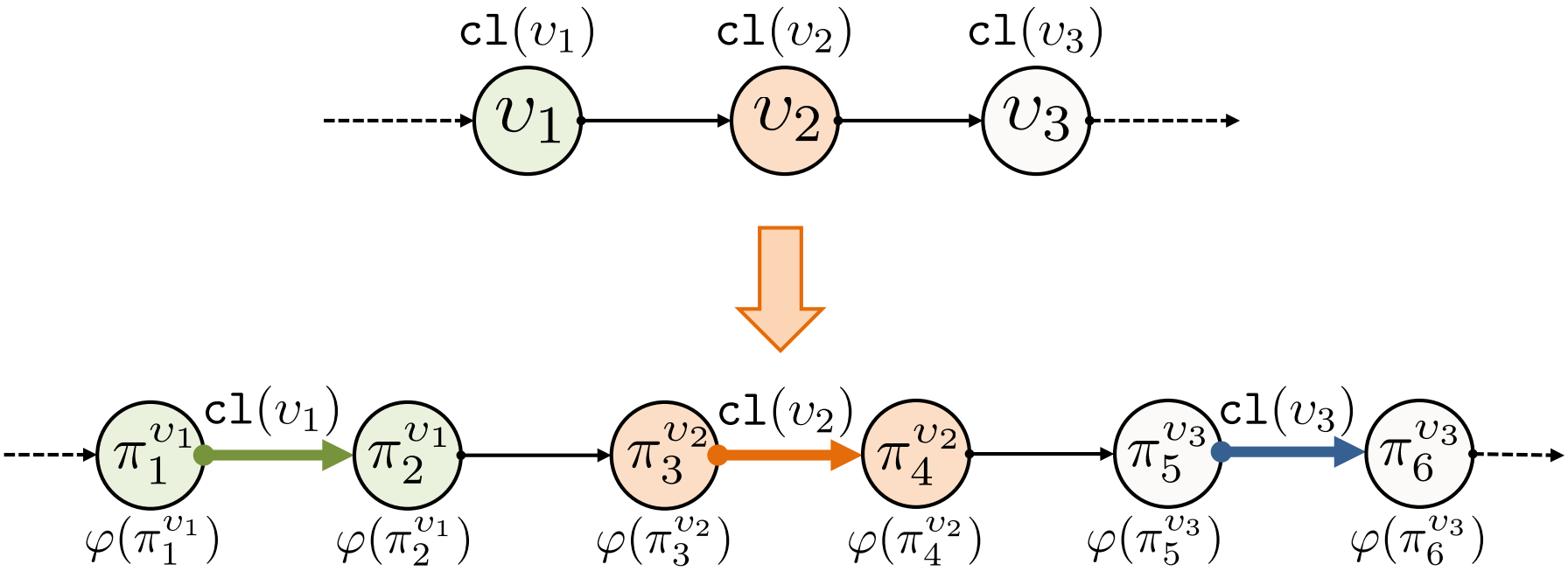
we define the **performance** constraint in terms of Critical Path Length

Multi Clock Domain Partitioning

Idea: slow down only actors outside the critical path

- The critical path length defines the longest serial part of the program execution
- The execution time of an action is directly related to its clock frequency

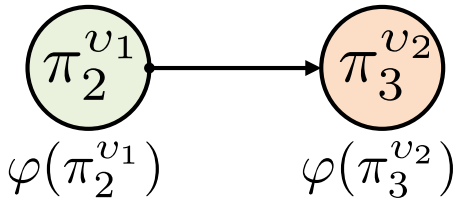
→ Stretch the graph in order to have all the parallel path with the same length



Multi Clock Domain Partitioning

Different kinds of Arcs:

- Action Firings Dependencies



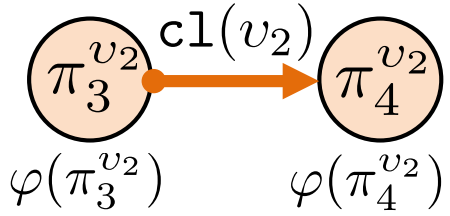
$$\varphi(\pi_3^{\upsilon_2}) - \varphi(\pi_2^{\upsilon_1}) \geq 0$$

Outside the CP

$$\varphi(\pi_3^{\upsilon_2}) - \varphi(\pi_2^{\upsilon_1}) = 0$$

Along the CP

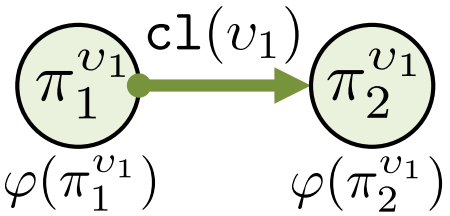
- Actions Firing inside the critical path



$$\varphi(\pi_4^{\upsilon_2}) - \varphi(\pi_3^{\upsilon_2}) = \mathtt{cl}(\upsilon_2)$$

The slow down factor is a free variable
(i.e. extend the arc's length)

- Actions Firing outside the critical path



$$\varphi(\pi_2^{\upsilon_1}) - \varphi(\pi_1^{\upsilon_1}) \geq \gamma\,\mathtt{cl}(\upsilon_2)$$

Multi Clock Domain Partitioning

Analytical Problem Formulation: ***(LP-formulation)***

$$\max. \quad \sum_{\mathrm{a} \notin \mathcal{A}_C} c_{\mathrm{a}}\, \gamma_{\mathrm{a}}$$

$$\text{s.t.} \quad \varphi(\pi_T) - \varphi(\pi_S) = \mathrm{CP}$$

Critical Path

$$\varphi(\pi_n^{v_i}) - \varphi(\pi_m^{v_h}) = w(e_{m,n}),\ \forall e_{m,n} \in \Xi_C$$

$$\varphi(\pi_n^{v_i}) - \varphi(\pi_m^{v_h}) \geq w(e_{m,n}),\ \forall e_{m,n} \notin \Xi_C, v_i \neq v_h$$

$$\varphi(\pi_n^{v_i}) - \varphi(\pi_m^{v_i}) \geq \gamma_a w(e_{m,n}),\ \forall e_{m,n} \notin \Xi_C$$

Outside the Critical Path

N

$$\gamma_{\mathrm{a}} \geq 1,\ \forall \mathrm{a} \notin \mathcal{A}_C$$

$$\phi(\mathrm{a}) = \begin{cases} \frac{\mathrm{f}_{max}}{\gamma_{\mathrm{a}}}, & \text{if a} \notin \mathcal{A}_C \\ \mathrm{f}_{max}, & \text{if a} \in \mathcal{A}_C \end{cases}$$

Multi Clock Domain Partitioning

Heuristic Approach: *(sub-optimal!)*

Try to slow down iteratively the clocks for actors outside the critical path

Require: $\Phi_1 \leftarrow \mathcal{A},\ \Phi_i = \emptyset,\ \forall i > 1$
Require: $\mathrm{CP} \leftarrow \mathrm{computeCP}(\Phi_1, \ldots, \Phi_N)$
Require: $k \leftarrow 1$
  while $k < N \wedge \Phi_k \neq \emptyset$ **do**
    $k \leftarrow k + 1$
    for all $\mathrm{a} \in \mathcal{R}^k$ **do**
      $\Phi^*_{k-1} \leftarrow \Phi_{k-1} \setminus \{\mathrm{a}\}$
      $\Phi^*_k \leftarrow \Phi_k \cup \{\mathrm{a}\}$
      $\mathrm{CP}^* \leftarrow \mathrm{computeCP}(\Phi_1, \ldots, \Phi^*_{k-1}, \Phi^*_{k-1}, \ldots)$
      if $\mathrm{CP}^* = \mathrm{CP}$ **then**
        $\phi(\mathrm{a}) \leftarrow \phi(\mathrm{a}) + 1$
        $\Phi_k \leftarrow \Phi^*_k$
      end if
    end for
  end while

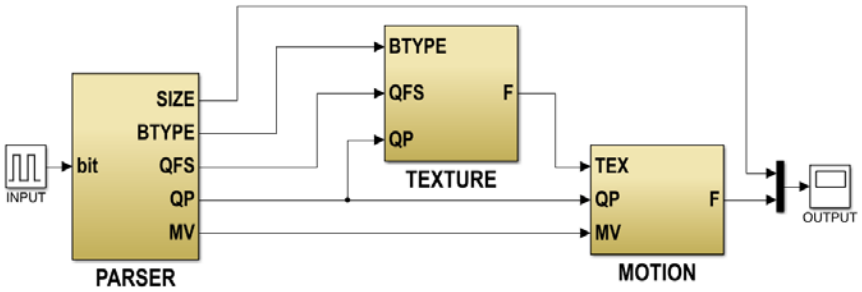
If slowing down a clock the clock of an actor outside the critical path does not change the performance, this is its new (temporarily) clock domain

Experimental Results: MPEG4-SP decoder

Application:
MPEG4-SP decoder
41 RVC-CAL actors

Platforms:
Virtex-5 FPGA

Execution Trace:
Steps: ≈ 10^6
Dependencies: ≈ 10^7



Experimental Results: MPEG4-SP decoder

Clock Domain	Actors
$f_1 = 50.0$ MHz	21
$f_2 = 6.25$ MHz	20

Clock Domain	Actors
$f_1 = 50.0$ MHz	12
$f_2 = 25.0$ MHz	5
$f_3 = 12.5$ MHz	8
$f_4 = 6.25$ MHz	16

Contribution	Nominal W	Optimized W	Δ%
Clocks	0.328	0.282	-14.0
Logic	0.069	0.06	-13.0
Signals	0.079	0.083	5.1
BRAMs	0.1	0.082	-18.0
Input/Output	0.005	0.005	0.0
Leakage	1.051	1.05	-0.1
Total	**1.632**	**1.562**	**-4.3**

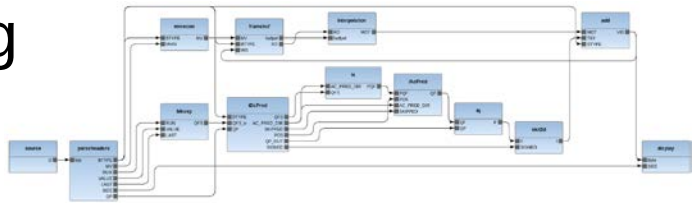
Contribution	Nominal W	Optimized W	Δ%
Clocks	0.436	0.357	-18.1
Logic	0.07	0.041	-41.4
Signals	0.095	0.053	-44.2
BRAMs	0.106	0.075	-29.2
Input/Output	0.005	0.004	-20.0
Leakage	1.053	1.05	-0.3
Total	**1.765**	**1.58**	**-10.5**

Power Reduction, compared to nominal performances

(i.e. all actors mapped in the clock domain with highest frequency)

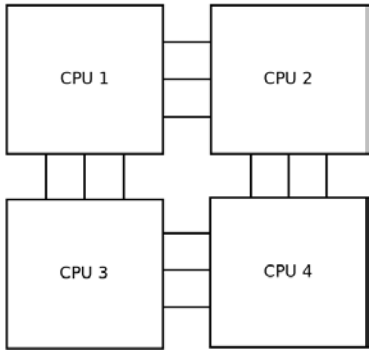
Dataflow programs: design exploration problem

- Partitioning



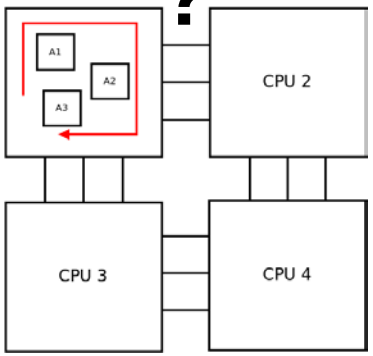
?

→

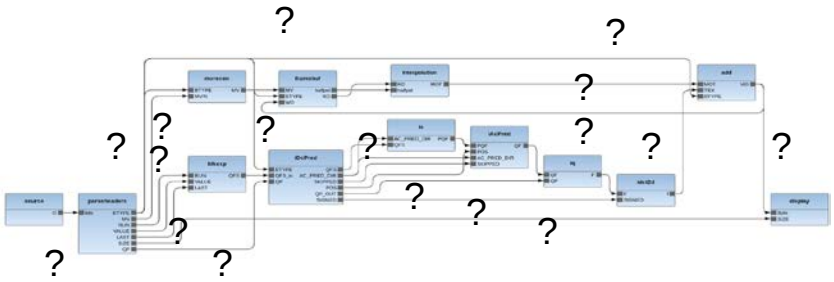


- Scheduling

?



- Buffer dimensioning



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Conclusions

Conclusions

- Dataflow (stream) programming provides the opportunity of a unified high level system design for both SW and HW implementations on heterogeneous platforms

- Exploration of the VERY LARGE design space (dataflow stream architecture, HW/SW partitioning, buffer dim, SW core partitioning, scheduling, clock assignment, ...) and optimization of parameters according to design criteria is a MUST!!

- Very interesting results and design optimizations can be obtained for dynamic dataflow programs by representing and exploring the execution trace

- Still progresses in tools support and further optimizations are under investigation: language extensions, design using mixed MoC, automatic support for dataflow refactoring, advanced scheduling, joint optimizations heuristics ……..

Bibliography (1)

Simone Casale-Brunet, Małgorzata Michalska, Endri Bezati, Junaid Jameel Ahmad, Marco Mattavelli: "Profiling and high-precision performance estimation of dynamic dataflow programs on multi-core platforms" in "Modeling, Optimization and Simulation for High Performance and Ultra-Low Power Circuits and Systems", Integration, the VLSI Journal.
[To appear in 2018].

Simone Casale-Brunet and M. Mattavelli: "Execution trace graph of dataflow process networks", IEEE Transactions on Multi-Scale Computing Systems, Volume: PP, Issue: 99, January 2018, Print ISSN: 2332-7766, DOI: 10.1109/TMSCS.2018.2790921

M. Michalska, S. Casale-Brunet , E. Bezati, and M. Mattavelli: "High-Precision Performance Estimation for the Design Space Exploration of Dynamic Dataflow Programs", IEEE Transactions on Multi-Scale Computing Systems, Volume: PP, Issue: 99, November 2017, Print ISSN: 2332-7766, Digital Object Identifier: 10.1109/TMSCS.2017.2774294.

Casale-Brunet, S; Mattavelli, M.; Janneck, J.W., "Buffer optimization based on critical path analysis of a dataflow program design," Circuits and Systems (ISCAS), 2013 IEEE International Symposium on, vol., no., pp.1384,1387, Beijing, China, 19-23 May 2013, doi: 10.1109/ISCAS.2013.6572113

Casale-Brunet, S.; Mattavelli, M.; Janneck, J.W., "TURNUS: A design exploration framework for dataflow system design," Circuits and Systems (ISCAS), 2013 IEEE International Symposium on, vol., no., pp.654,654, Beijing, China, 19-23 May 2013 doi: 10.1109/ISCAS.2013.6571927

- Małgorzata Michalska, Nicolas Zufferey and Marco Mattavelli: "Performance Estimation Based Multicriteria Partitioning Approach for Dynamic Dataflow Programs," Journal of Electrical and Computer Engineering, vol. 2016, Article ID 8536432, 15 pages, 2016. doi:10.1155/2016/8536432, http://www.hindawi.com/journals/jece/2016/8536432/ .

- M. Canale, S. Casale-Brunet, E. Bezati, M. Mattavelli, J. Janneck: "Dataflow Programs Analysis and Optimization Using Model Predictive Control Techniques", Journal of Signal Processing Systems, 2016, Vol: 84, No. 3, Pages 371--381, issn="1939-8115", Doi: 10.1007/s11265-015-1083-4, http://dx.doi.org/10.1007/s11265-015-1083-4 http://link.springer.com/article/10.1007/s11265-015-1083-4

- E. Bezati; S. Casale Brunet; M. Mattavelli; J. W. Janneck, "Clock-gating of streaming applications for energy efficient implementations on FPGAs," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems , vol. PP, Issue no.99, pp.1-1, Aug. 2016, doi: 10.1109/TCAD.2016.2597215.

- K. Jerbi, D. Renzi, D. De Saint Jorre, H. Yviquel, M. Raulet, C. Alberti, M. Mattavelli: "Development and optimization of high level dataflow programs: the HEVC decoder design case", Journal of Signal Processing Systems, 2016. http://link.springer.com/article/10.1007/s11265-016-1113-x

Bibliography (2)

Anatoly Prihozhy, Endri Bezati, Ab Al-Hadi Ab Rahman, Marco Mattavelli: "Synthesis and Optimization of Pipelines for HW Implementations of Dataflow Programs", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 10, October 2015, Pag. 1613-1626.

Jani Boutellier, Johan Ersfollk, Johan Lilius, Marco Mattavelli, Ghislain Roquier, Olli Silven: "Actor Merging for Dataflow Process Networks" IEEE Transactions on Signal Processing, Vol. 63, No. 10, May 15, 2015, Pag. 2496-2508.

C. Sau, P. Meloni, L. Raffo, F. Palumbo, E. Bezati, S. Casale Brunet and M. Mattavelli, "Automated Design Flow for Multi-Functional Dataflow-Based Platforms," in Journal of Signal Processing Systems -Signal Image and Video Technology-, p. 1-23, 2015. http://link.springer.com/article/10.1007/s11265-015-1026-0

C. Massimo, S. Casale Brunet, E. Bezati, M. Mattavelli and J. Janneck, "Dataflow Programs Analysis and Optimization Using Model Predictive Control Techniques. Two Examples of Bounded Buffer Scheduling: Deadlock Avoidance and Deadlock Recovery Strategies," in Journal of Signal Processing Systems, p. 1-11, 2015. Doi: 10.1007/s11265-015-1083-4. http://link.springer.com/article/10.1007%2Fs11265-015-1083-4.

M. Michalska, E. Bezati, S. Casale-Brunet and M. Mattavelli, "Buffer dimensioning for throughput improvement of dynamic dataflow signal processing applications on multi-core platforms," 2017 25th European Signal Processing Conference (EUSIPCO), Kos, 2017, pp. 1339-1343, DOI: 10.23919/EUSIPCO.2017.8081426

S. Casale-Brunet, E. Bezati and M. Mattavelli, "Design space exploration of dataflow-based Smith-Waterman FPGA implementations," 2017 IEEE International Workshop on Signal Processing Systems (SiPS), Lorient, France, 2017, pp. 1-6, DOI: 10.1109/SiPS.2017.8109982

S. Casale-Brunet, E. Bezati, A. Bloch and M. Mattavelli, "High-level system synthesis and optimization of dataflow programs for MPSoCs", "2017 51th Asilomar Conference on Signals, Systems and Computers", Pacific Grove, CA, 2017.

E. Bezati, S. Casale-Brunet and M. Mattavelli, "Execution Trace Graph based interface synthesis of signal processing dataflow programs for heterogeneous MPSoCs", "2017 51th Asilomar Conference on Signals, Systems and Computers", Pacific Grove, CA, 2017

E. Bezati, S. Casale Brunet, M. Mattavelli, J. W. Janneck: "High-level system synthesis and optimization of dataflow programs for MPSoCs", in 2016 50th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, USA, November 6-9, 2016.

E. Bezati; S. Casale Brunet; M. Mattavelli; J. W. Janneck, "High-Level Synthesis of Dynamic Dataflow Programs on heterogeneous MPSoC platforms", in 2016 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS).

Bibliography (3)

Simone Casale-Brunet, Endri Bezati, Marco Mattavelli: "Programming Models and Methods for Heterogeneous Parallel Embedded Systems", 2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Lyon, France, September 21-23 2016.

Simone Casale-Brunet, Endri Bezati, Marco Mattavelli: "High Level Synthesis of Smith-Waterman dataflow implementations", IEEE 42nd International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2016

M. Michalska, S. Casale Brunet, E. Bezati, M. Mattavelli, J. Janneck. Trace-Based Manycore Partitioning of Stream-Processing Applications. 50th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, USA, November 6-9, 2016.

M. Michalska, S. Casale Brunet, E. Bezati, M. Mattavelli: "High-precision performance estimation of dynamic dataflow programs". IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip, IEEE MCSoC 2016, Lyon, France, September 21-23 2016.

M. Michalska, N. Zufferey, E. Bezati, M. Mattavelli: "Design space exploration problem formulation for dataflow programs on heterogeneous architectures". IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip, IEEE MCSoC 2016, Lyon, France, September 21-23 2016.

C. Alberti, N. Daniels, M. Hernaez, J. Voges, R. L. Goldfeder, A. A. Hernandez-Lopez, M. Mattavelli, B. Berger: "An Evaluation Framework for Lossy Compression of Genome Sequencing Quality Values", IEEE 2016 Data Compression Conference, Snowbird, Utah, USA, March 27-29, 2016, DOI 10.1109/DCC.2016.39.

M. Michalska, J.J. Ahmad, E. Bezati, S. Casale-Brunet, M. Mattavelli: "Performance Estimation of Program Partitions on Multi-core Platforms". International Workshop on Power and Timing Modeling, Optimization and Simulation, Bremen, Germany, September 21-23 2016.

M. Michalska, N. Zufferey, M. Mattavelli: "Tabu Search for Partitioning Dynamic Dataflow Programs". International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA, Procedia Computer Science, Volume 80, Pages 1577-1588, 2016.

M. Michalska, E. Bezati, S. Casale-Brunet, M. Mattavelli: "A Partition Scheduler Model for Dynamic Dataflow Programs". International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA, Procedia Computer Science, Volume 80, Pages 2287–2291, 2016.

Bibliography (4)

M. Michalska, N. Zufferey, J. Boutellier, E. Bezati and M. Mattavelli. Efficient scheduling policies for dataflow programs executed on multi-core. Proceedings of the 9th International Workshop on Programmability and Architectures for Heterogeneous Multicore (MULTIPROG), Prague, Czech Republic, January 18, 2016.

M. Michalska, J. Boutellier and M. Mattavelli: "A methodology for Profiling and Partitioning Stream Programs on Many-core Architectures: "International Conference on Computational Science (ICCS), Reykjavik, Iceland, June 1-3, 2015, Procedia Computer Science, Vol 51, pag. 2962-2966. Doi: 10.1016/j.procs.2015.05.498.
http://www.sciencedirect.com/science/article/pii/S187705091501306X

M. Michalska, S. Casale-Brunet, E. Bezati and M. Mattavelli: "Execution Trace Graph Based Multi-criteria Partitioning of Stream Programs: "International Conference on Computational Science (ICCS), Reykjavik, Iceland, June 1-3, 2015, Procedia Computer Science, Vol 51, pag. 1443-1452. Doi: 10.1016/j.procs.2015.05.334
http://www.sciencedirect.com/science/article/pii/S1877050915011424

- Canale, M.; Casale-Brunet, S.; Bezati, E.; Mattavelli, M.; Janneck, J.W., "Dataflow Programs Analysis and Optimization using Model Predictive Control Techniques: an example of bounded buffer scheduling", SiPS, 2014, Belfast, North Ireland.
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6986054
DOI: 10.1109/SiPS.2014.6986054

- Casale-Brunet, S.; Bezati, E.; Mattavelli, M.; Canale, M.; Janneck, J.W., "Execution trace graph analysis of dataflow programs: bounded buffer scheduling and deadlock recovery using model predictive control", Conference on Design and Architectures for Signal and Image Processing (DASIP 2014), October 2014, Madrid, Spain.
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7115623
DOI: 10.1109/DASIP.2014.7115623

- S. Casale Brunet, M. Michalska, E. Bezati, M. Mattavelli, J. Janneck and M. Canale: "TURNUS: an open-source design space exploration framework for dynamic stream programs,", Conference on Design and Architectures for Signal and Image Processing (DASIP), October 2014, Madrid, Spain.
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7115614
DOI: 10.1109/DASIP.2014.7115614

- ICIP 2014 Damien de Saint Jorre, Claudio Alberti, Marco Mattavelli, Simone Casale-Brunet "Exploring MPEG HEVC decoder parallelism for the efficient porting onto many-core platforms" 2014 IEEE International Conference on Image Processing (ICIP), Paris, France, 27-30 October 2014, p. 2115-2119, doi:10.1109/ICIP.2014.7025424.

Bibliography (5)

E. Bezati, S. Casale-Brunet, M. Mattavelli, J.W. Janneck: " Coarse Grain Clock Gating of Streaming Applications In Programmable Logic Implementations," Proceedings Of The 2014 Electronic System Level Synthesis Conference (Eslsyn), 2014, San Francisco, CA, USA, Doi:10.1109/ESLsyn.2014.6850387, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6850387.

C. Sau, L. Raffo, F. Palumbo, E. Bezati, S. Casale-Brunet, M. Mattavelli: "Automated Design Flow for Coarse-Grained Reconfigurable Platforms: an RVC-CAL Multi-Standard Decoder Use-Case", IEEE International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014, Agios Konstantinos, Greece, DOI: 10.1109/SAMOS.2014.6893195 http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6893195.

J.W. Janneck., S. Casale-Brunet, M. Mattavelli: "Characterizing Communication Behaviour of Dataflow Programs Using Trace Analysis", IEEE International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014, Agios Konstantinos, Greece, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6893193 DOI: 10.1109/SAMOS.2014.6893193.

Rahman, A.A.A, Casale-Brunet S., Alberti C., Mattavelli M., "A Methodology for Optimizing Buffer Sizes of Dynamic Dataflow FPGAs Implementations", 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP2014), Firenze, Italy, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6854554 DOI: 10.1109/ICASSP.2014.6854554

Ersfolk, J.; Roquier, G.; Lund, W.; Mattavelli, M. & Lilius, J., "Static and Quasi-static Compositions of Stream Processing Applications from Dynamic Dataflow Programs", Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Vancouver, Canada, May 2013.

Bezati, E.; Casale-Brunet, S.; Mattavelli, M.; Janneck, J.W., "Synthesis and optimization of high-level stream programs," Electronic System Level Synthesis Conference (ESLsyn), 2013 , pp.1-6, Austin, Texas, USA, May 31-June 1, 2013.

Bezati, E., Thavot, R., Roquier, G., & Mattavelli, M. High-level dataflow design of signal processing systems for reconfigurable and multicore heterogeneous platforms. Journal of Real-Time Image Processing, 2013, doi:10.1007/s11554-013-0326-5.

Bezati, E.; Roquier, G.; Mattavelli, M., "Live demonstration: High level software and hardware synthesis of dataflow programs," Circuits and Systems (ISCAS), 2013 IEEE International Symposium on , vol., no., pp.660,660, Beijing, China, 19-23 May 2013, doi: 10.1109/ISCAS.2013.6571930.

Bibliography (6)

Casale-Brunet, S.; Mattavelli, M.; Alberti, C.; Janneck, J.W., "Representing Guard Dependencies in Dataflow Execution Traces," Computational Intelligence, Communication Systems and Networks (CICSyN), 2013 Fifth International Conference on , vol., no., pp.291,295, Madrid, Spain, 5-7 June 2013, doi: 10.1109/CICSYN.2013.26,

Casale-Brunet, S.; Mattavelli, M.; Alberti, C.; Janneck, J.W., "Design Space Exploration of High Level Stream Programs on Parallel Architectures: A focus on the Buffer Size Minimization and Optimization Problem," 8th International Symposium on Image and Signal Processing and Analysis (ISPA 2013), Trieste, Italy, 4-6 September 2013

Casale-Brunet, S.; Alberti, C.; Mattavelli, M.; Janneck, J.W., "TURNUS: a unified dataflow design space exploration framework for heterogeneous parallel systems," Conference on Design & Architectures for Signal & Image Processing (DASIP), Cagliari, Italy, 8-10 October 2013

Casale-Brunet, S.; Bezati, E.; Roquier, G., Alberti, C.; Mattavelli, M.; Janneck, J.W.; Boutellier, J., "Design Space Exploration and Implementation of RVC-CAL Applications using the TURNUS framework," Conference on Design & Architectures for Signal & Image Processing (DASIP), Cagliari, Italy, 8-10 October 2013

Casale-Brunet, S.; Bezati, E.; Alberti, C.; Mattavelli, M.; Amaldi, E.; Janneck, J.W., " Partitioning and Optimization of high level Stream applications for Multi Clock Domain Architectures," Signal Processing Systems (SiPS), 2013 IEEE Workshop on, Taipei, Taiwan, 16-18 October 2013

Casale-Brunet, S.; Bezati, E.; Alberti, C.; Mattavelli, M.; Amaldi, E.; Janneck, J.W., "Multi-clock domain optimization for reconfigurable architectures in high-level dataflow applications," Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 3-6 November 2013

Casale-Brunet, S.; Alberti, C.; Mattavelli, M.; Janneck, J.W., "Systems Design Space Exploration by Serial Dataflow Program Executions," Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 3-6 November 2013

Casale-Brunet, S.; Elguindy, A.; Bezati, E.; Thavot, R.; Roquier, G., Mattavelli, M.; Janneck, J.W., "Methods to explore design space for MPEG RMC codec specifications," Signal Processing: Image Communication, 2013, ISSN 0923-5965, DOI: 10.1016/j.image.2013.08.012.

Casale-Brunet, S.; Mattavelli, M.; Janneck, J.W., "Profiling of Dataflow Programs Using Post Mortem Causation Traces," Signal Processing Systems (SiPS), 2012 IEEE Workshop on , vol., no., pp.220,225, Québec City, Canada, 17-19 October 2012, doi: 10.1109/SiPS.2012.54,

Bibliography (7)

Bezati, E.; Casale-Brunet, S.; Mattavelli, M.; Janneck, J.W., "Synthesis and optimization of high-level stream programs," Electronic System Level Synthesis Conference (ESLsyn), 2013 , vol., no., pp.1,6, Austin, Texas, USA, May 31-June 1 2013

Bezati, E., Thavot, R., Roquier, G., & Mattavelli, M. High-level dataflow design of signal processing systems for reconfigurable and multicore heterogeneous platforms. Journal of Real-Time Image Processing, 2013, doi:10.1007/s11554-013-0326-5

Bezati, E.; Roquier, G.; Mattavelli, M., "Live demonstration: High level software and hardware synthesis of dataflow programs," Circuits and Systems (ISCAS), 2013 IEEE International Symposium on , vol., no., pp.660,660, Beijing, China, 19-23 May 2013, doi: 10.1109/ISCAS.2013.6571930

Bezati E.; Mattavelli M.; Janneck, J.W, "High-Level Synthesis of Dataflow Programs for Signal Processing Systems", 8th International Symposium on Image and Signal Processing and Analysis (ISPA 2013), Trieste, Italy, 4-6, September 2013

Rahman, A.A.A; Casale-Brunet, S.; Alberti, C.; Mattavelli, M., "Dataflow Program Analysis and Refactoring Techniques for Design Space Exploration: MPEG-4 AVC/H.264 Decoder Implementation Case Study," Conference on Design & Architectures for Signal & Image Processing (DASIP), Cagliari, Italy, 8-10 October 2013

Rahman, A.A.A.; Thavot, R.; Casale-Brunet, S.; Bezati, E.; Mattavelli, M., "Design space exploration strategies for FPGA implementation of signal processing systems using CAL dataflow program," Design and Architectures for Signal and Image Processing (DASIP), 2012 Conference on, vol., no., pp.1,8, Karlsruhe, Germany, 23-25 October 2012

Ahmad, J.J.; Li, S.; & Mattavelli, M., "Performance Benchmarking of RVC based Multimedia Specifications," Proceedings of the 20th IEEE International Conference on Image Processing (ICIP), Melbourne Australia, September 2013 © IEEE.

Ahmad, J.J.; Li, S.; Thavot, R.; & Mattavelli, M., "Secure Computing with the MPEG RVC Framework," Signal Processing: Image Communication, August 2013 © Elsevier Science B. V., DOI: 10.1016/j.image.2013.08.015

Ersfolk, J.; Roquier, G.; Lund, W.; Mattavelli, M. & Lilius, J., "Static and Quasi-static Compositions of Stream Processing Applications from Dynamic Dataflow Programs", Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Vancouver, Canada, May 2013.

M. Mattavelli: "MPEG Reconfigurable Video Representation", in "The MPEG Representation of Digital Media", Chapter 12, Springer Ed. 2011, DOI 10.1007/978-1-4419-6184-6_12. http://www.springer.com/engineering/signals/book/978-1-4419-6183-9 http://www.springerlink.com/content/q7728730l4215455/

M. Mattavelli, M. Raulet, J. Janneck: "MPEG Reconfigurable Video Coding" in "Handbook of Signal processing Systems", Pag. 43-68, S. S. Bhattacharyya, Ed F. Deprettere, R. Leupers and J. Takala Editors, Springer 2010. http://www.springer.com/engineering/signals/book/978-1-4419-6344-4

Bibliography (8)

Ab Al-Hadi Ab Rahman, Anatoly Prihozhy, Marco Mattavelli: "Pipeline Synthesis and Optimization of FPGA-based Video Processing Applications with CAL", Eurasip Journal on Image and Video Processing, 2011, 2011:19, http://jivp.eurasipjournals.com/content/2011/1/19.

Marco. Mattavelli, Ihab Amer and Mickael Raulet: "The reconfigurable video coding standard", IEEE Signal Processing Magazine, May 2010, Vol 27, No. 3, pag. 159-167.

Shuvra S. Bhattacharyya, Gordon Brebner, Johan Eker, Jorn W. Janneck, Marco Mattavelli, Carl von Platen, and Mickael Raulet: "OpenDF - A Dataflow Toolset for Reconfigurable Hardware and Multicore Systems", ACM SIGARCH Computer Architecture News, Special Issue: MCC08 – Multicore Computing 2008, Volume 36, Number 5, pp 29-35, December 2008.

Ihab Amer, Christophe Lucarz , Marco Mattavelli, Ghislain Roquier, Mickael Raulet, Olivier Deforges, Jean-Francois Nezan: "Reconfigurable Video Coding: the Video Coding Standard for Multi-core Platforms", IEEE Signal Processing Magazine, Special issue on Multicore Platforms, Vol 26, No. 6, November 2009, pag 113-123.

Mickael Raulet, Marco Mattavelli, Jorn Janneck: "Guest editorial: Special Issue on Reconfigurable Video Coding", Journal of Signal Processing Systems, 2009, DOI - 10.1007/s11265-009-0418-4, Link - http://www.springerlink.com/content/k0157333g2602l55/.

Christophe Lucarz, Jonathan Piat, Marco Mattavelli: "Automatic synthesis of parsers and validation of bitstreams within the MPEG Reconfigurable Video Coding Framework", Journal of Signal Processing Systems, 2009, DOI - 10.1007/s11265-009-0395-7 Link - http://www.springerlink.com/content/gg261p657x483m91.

Hussein K Aman-Allah, Karim M Maarouf, Ehab A Hanna, Ihab M Amer, Marco Mattavelli: "CAL Dataflow Components For an MPEG RVC AVC Baseline Encoder", Journal of Signal Processing Systems, 2009, DOI - 10.1007/s11265-009-0396-6 Link - http://www.springerlink.com/content/j487766m06306127.

Shuvra S. Bhattacharyya, Johan Eker, Jorn W. Janneck, Christophe Lucarz, Marco Mattavelli, Mickael Raulet: "Overview of the MPEG Reconfigurable Video Coding Framework", Journal of Signal Processing Systems, 2009, DOI - 10.1007/s11265-009-0399-3 Link - http://www.springerlink.com/content/24t3610261j1628j.

Credits and thanks to:

- Prof. Jorn Janneck, Prof. Edoardo Amaldi (Lund University, Politecnico di Milano)
- Simone Casale Bruner, Endri Bezati, Malgorzata Michalska, Claudio Alberti, Ghislain Roquier, Ab Al-Hadi Ab Rahman (EPFL)
- Dave Parlour, Ian Miller (Xilinx)
- Mickael Raulet, Mathieu Wipliez, JF Nezan (INSA)
- Johan Eker, Carl von Platen (Ericsson)
- And many others ……

Thanks for your attention!