Multicore Digital Signal Processing

Maxime Pelcat

2014

Slides from M. Pelcat, K. Desnos, J-F. Nezan, D. Ménard, M. Raulet, J Gorin

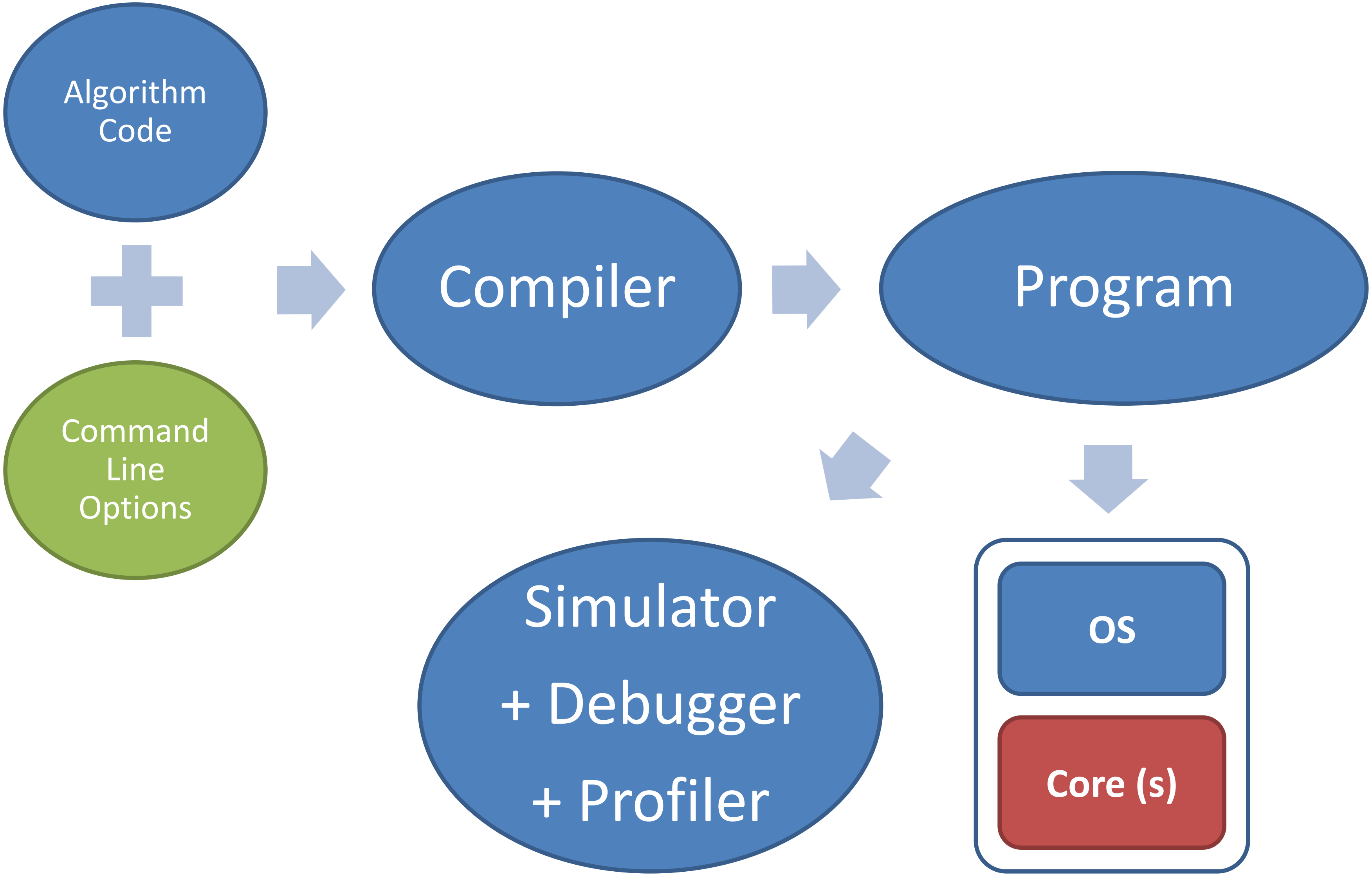
Institut National des Sciences Appliquées
Ecole publique d'ingénieurs

UNIVERSITÉ EUROPÉENNE DE BRETAGNE

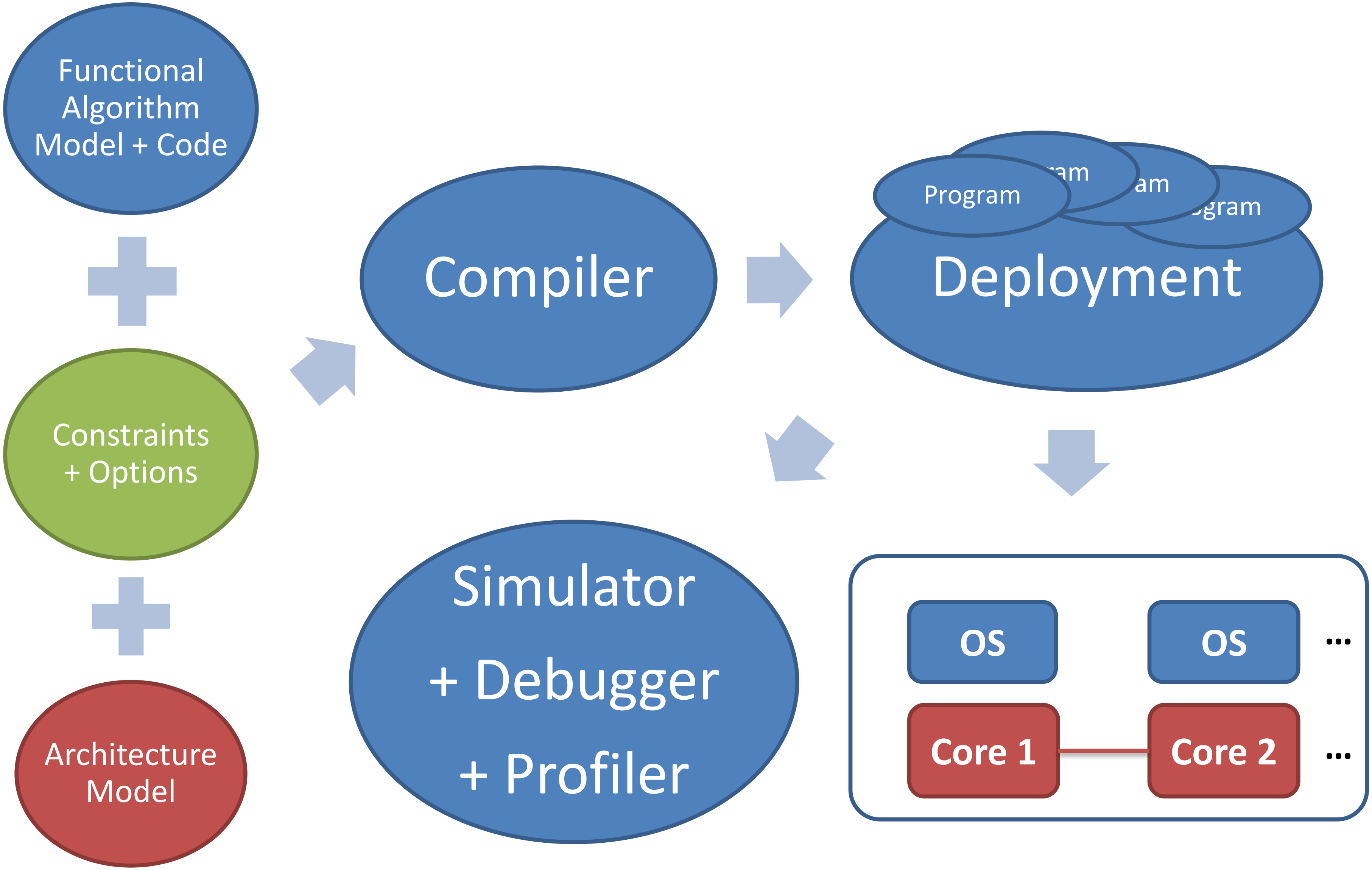
Institut National des Sciences Appliquées

Introduction: Porting Algorithms to Multicore DSPs

Typical Code Development Environment



Possible MPSoC Dataflow-based Development Environment



Addressed Problem

- **Distributed embedded systems are harder and harder to program**
- **Difficult Constraints**

  High computation requirements

  Low power consumption

  Many hardware and software choices: lack of information/metrics

  Real-time constraints (hard of soft real-time)

  Need to reuse legacy code

- **Difficult Goals**

  Design both hardware and software

  Balance loads

  Obtain the most from a given architecture

  Respect constraints

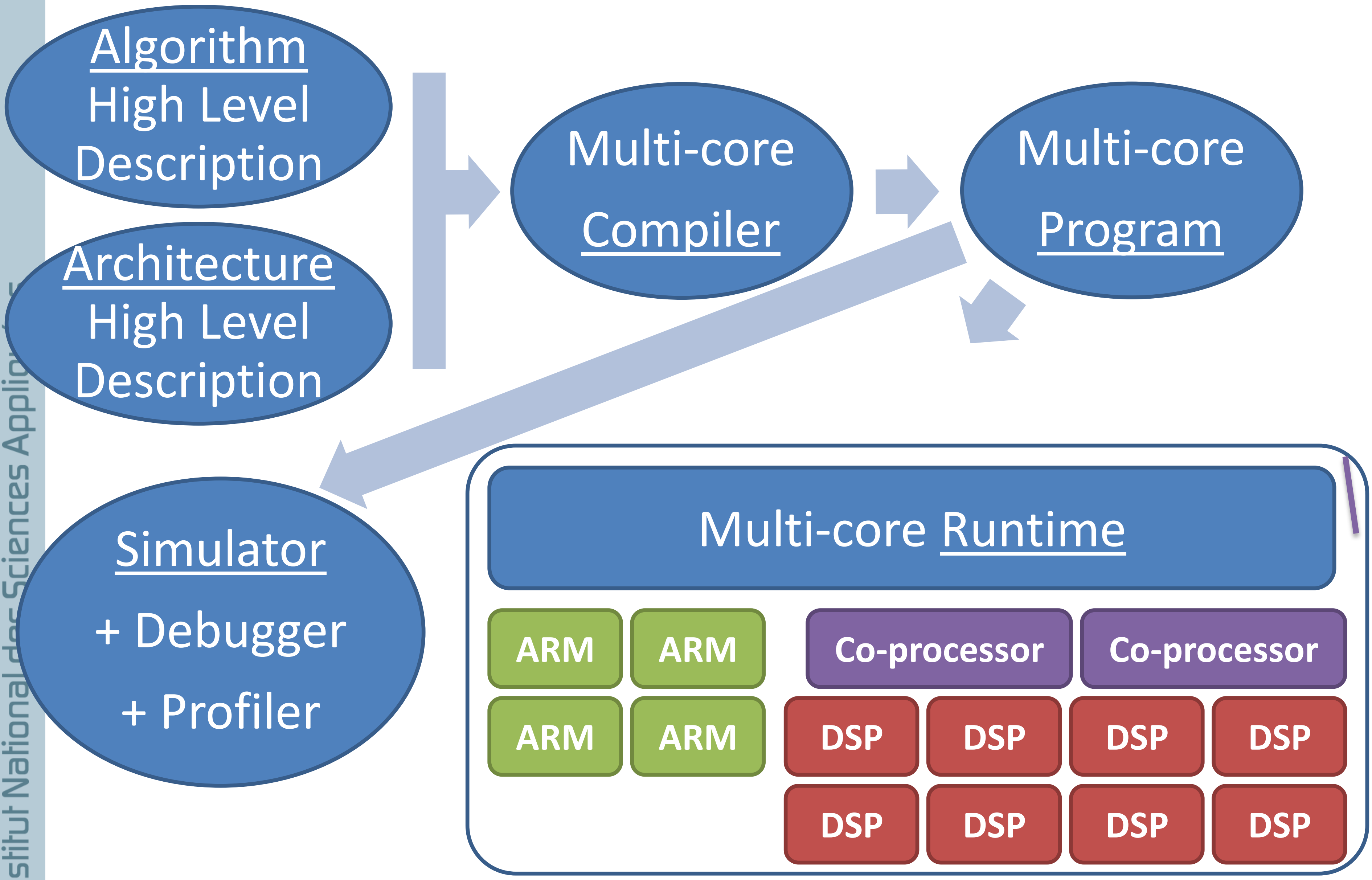
Addressed Problem

- **Traditional code in C abstracts core architecture**
  - Amount of registers
  - Number of pipeline stages
  - Instruction parallelism
  - Loop optimizations
  - Cache accesses
  - Data representation
  - …

- **C code can not be efficiently transformed into coarse grain parallel code**
  - Assumed global state in a program
  - Unique activity point
  - Inspired by the Turing machine

- **The solution may come from dataflow MoCs**

Grail of Multi-core DSP Programming



Algorithm High Level Description

Architecture High Level Description

Multi-core Compiler

Multi-core Program

Simulator + Debugger + Profiler

Multi-core Runtime

ARM ARM ARM ARM

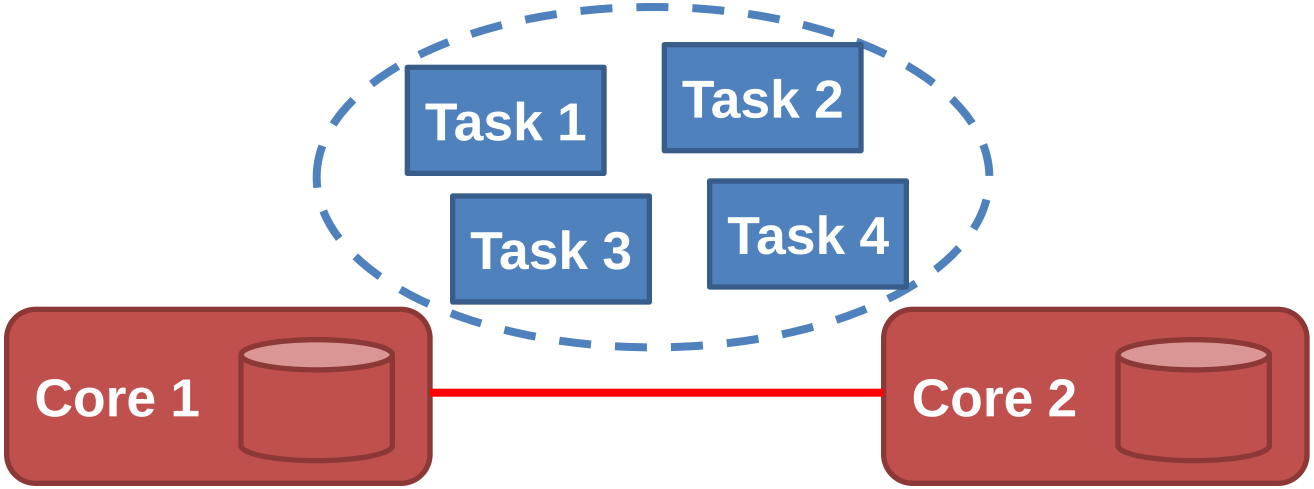
Co-processor Co-processor

DSP DSP DSP DSP

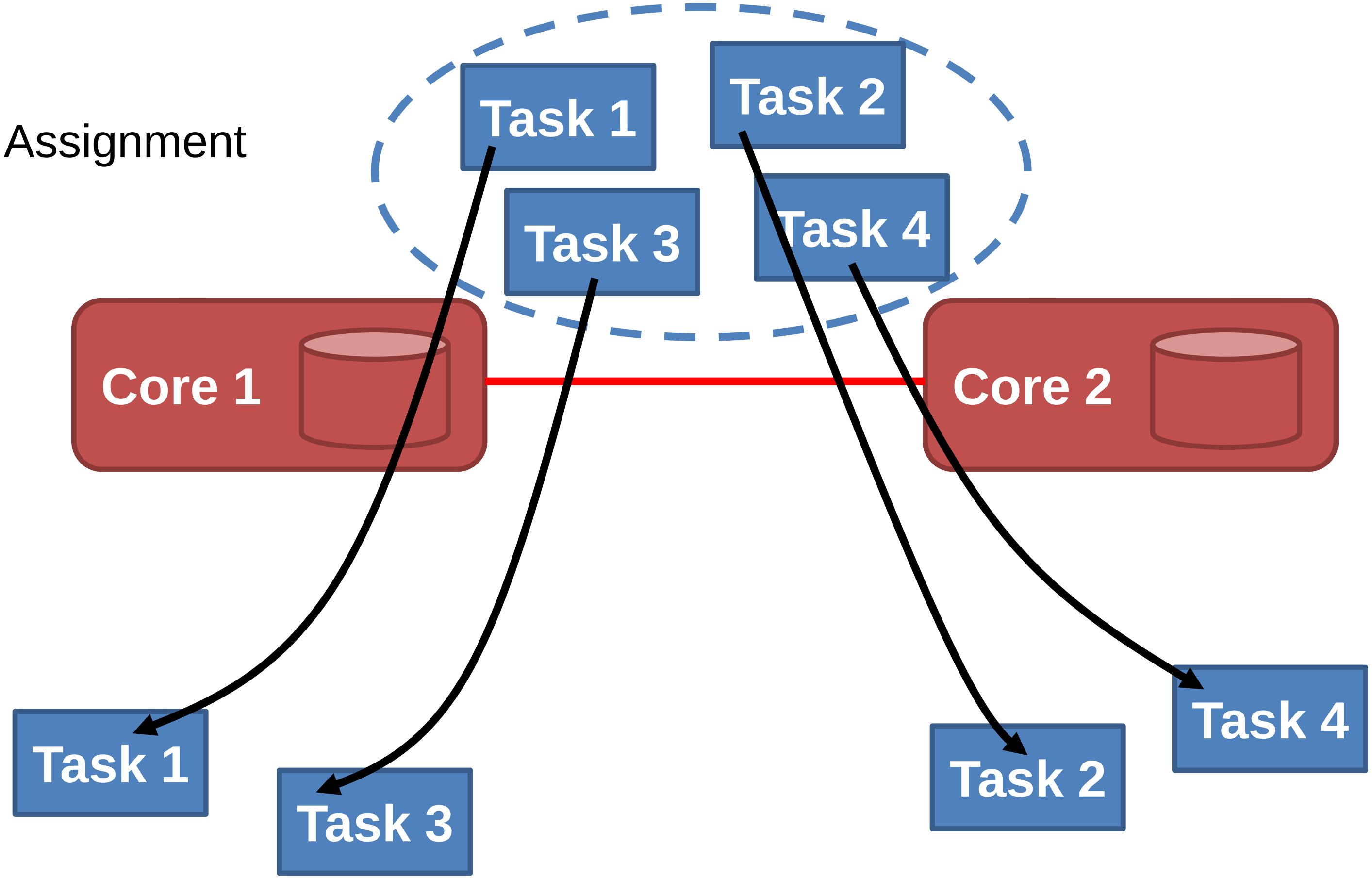
DSP DSP DSP DSP

Code Porting

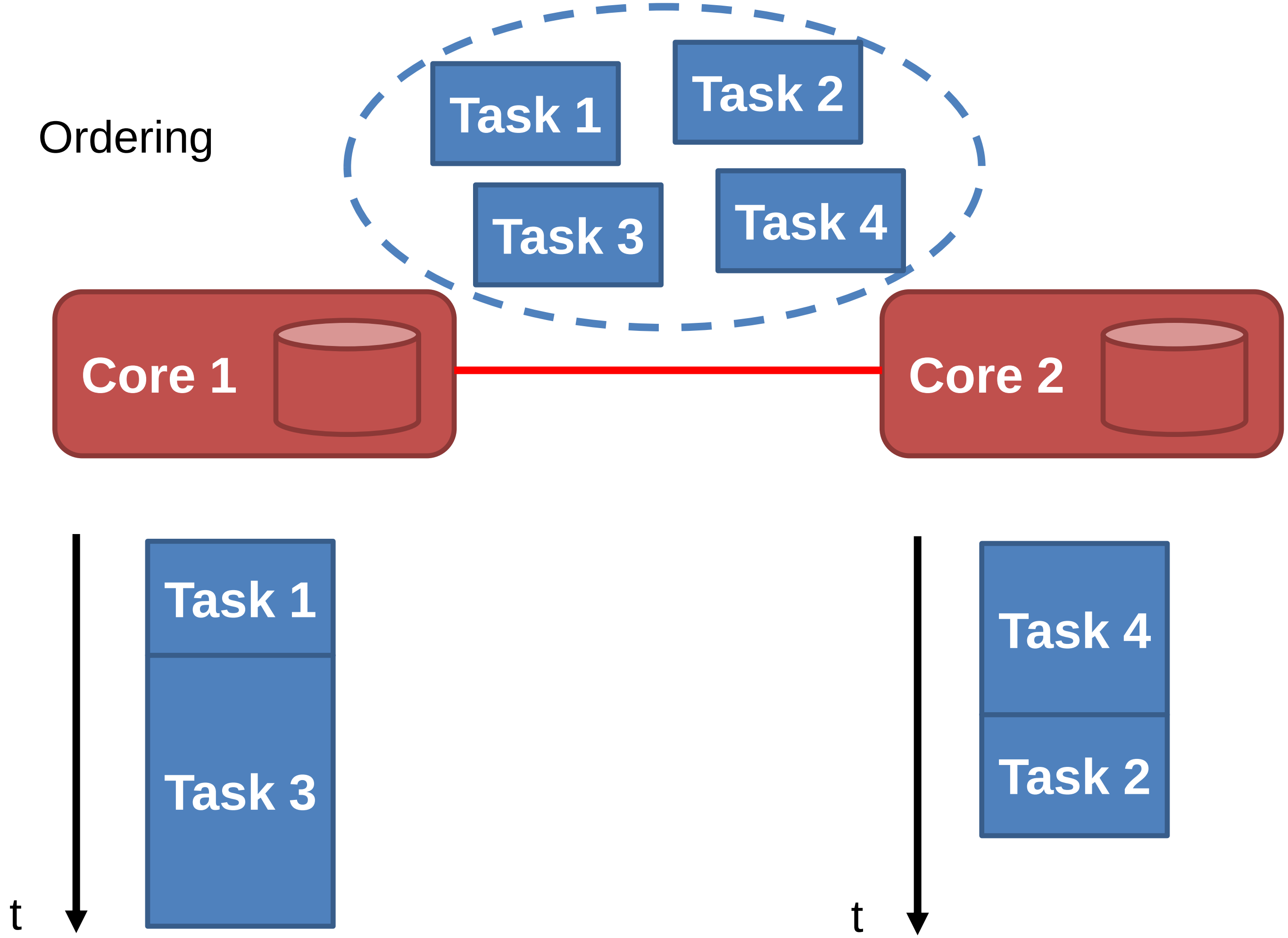
Multi-core code porting → assignment, ordering and timing



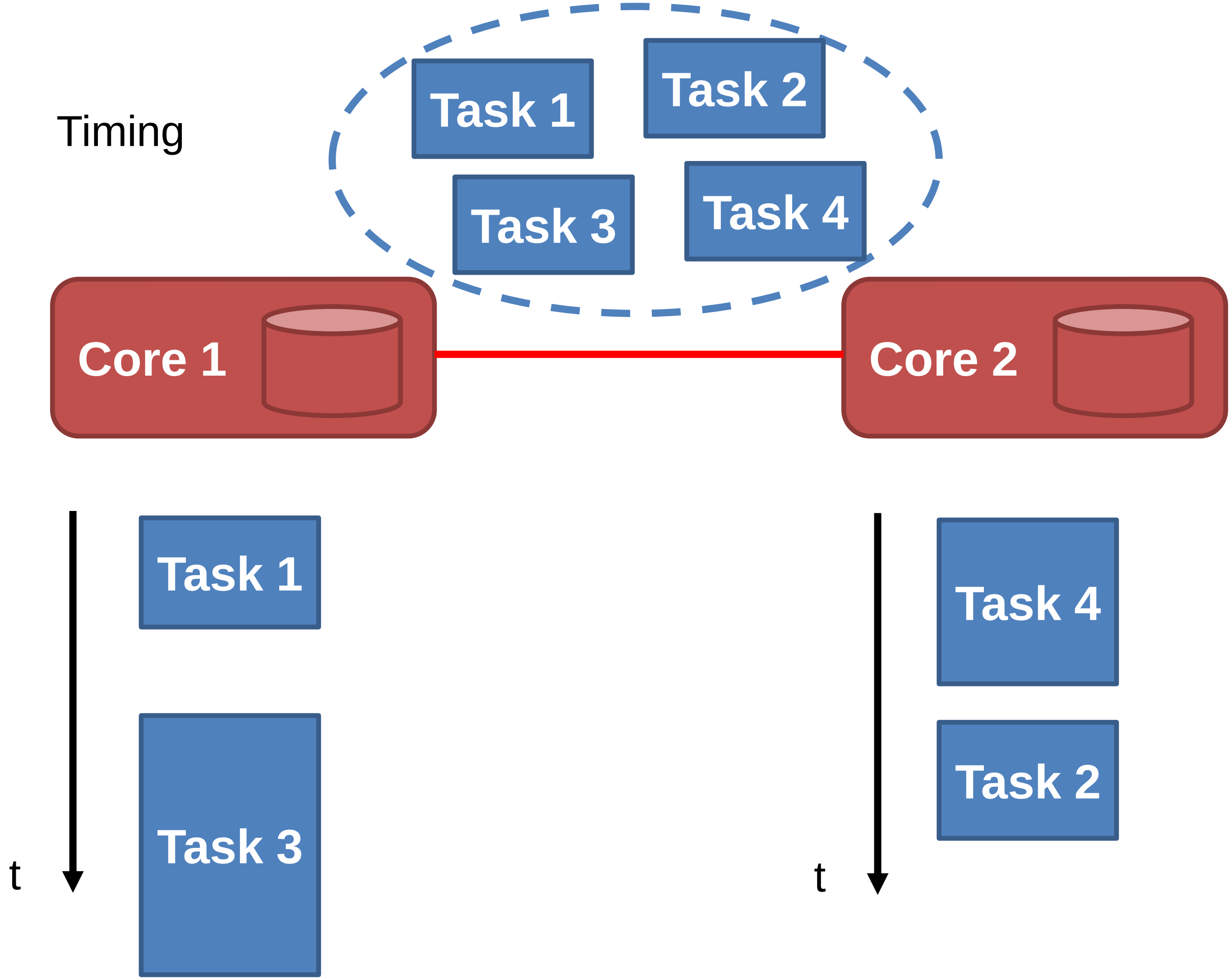
Code Porting



Code Porting



Code Porting



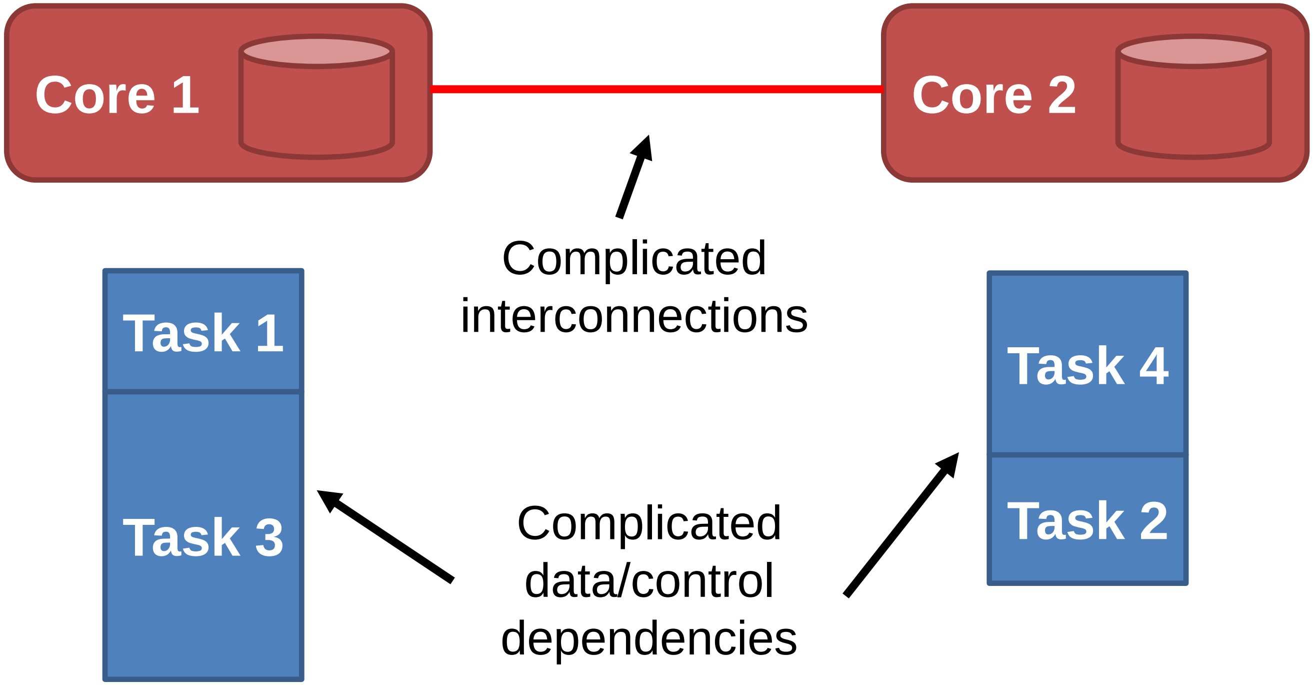
Code Porting

- **And other tasks:**
  - Choose communication (shared memory, DMA, direct copy)
  - Choose communication synchronization (polling or interrupts)
  - Allocate in memory, order and time communications

Code Deployment

Many possible assignments and orders

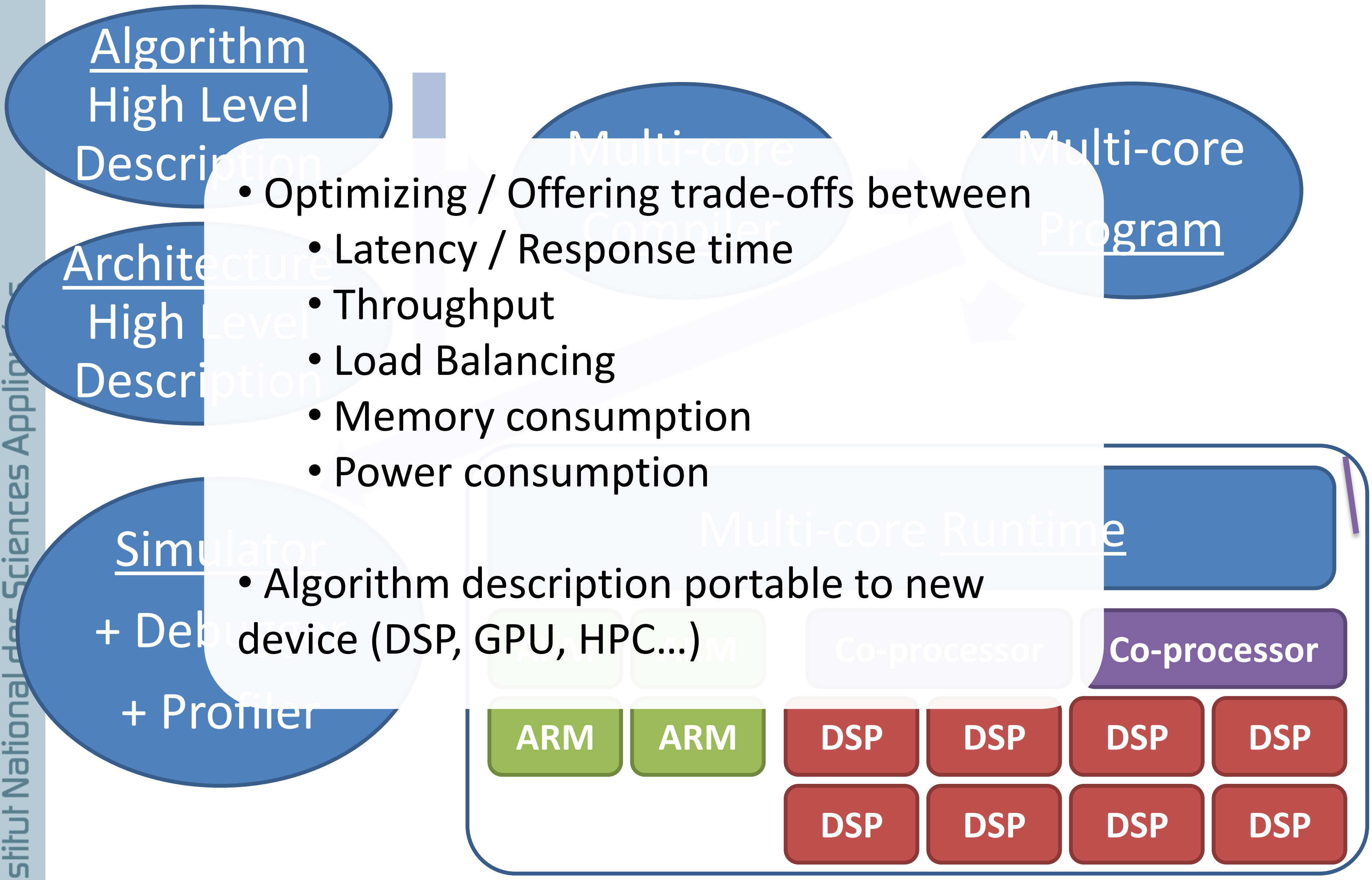
- Minimize latency/ response time
- Minimize execution time
- Minimize memory consumption
- Minimize power consumption
- …



Complicated interconnections

Complicated data/control dependencies

Grail of Multi-core DSP Programming

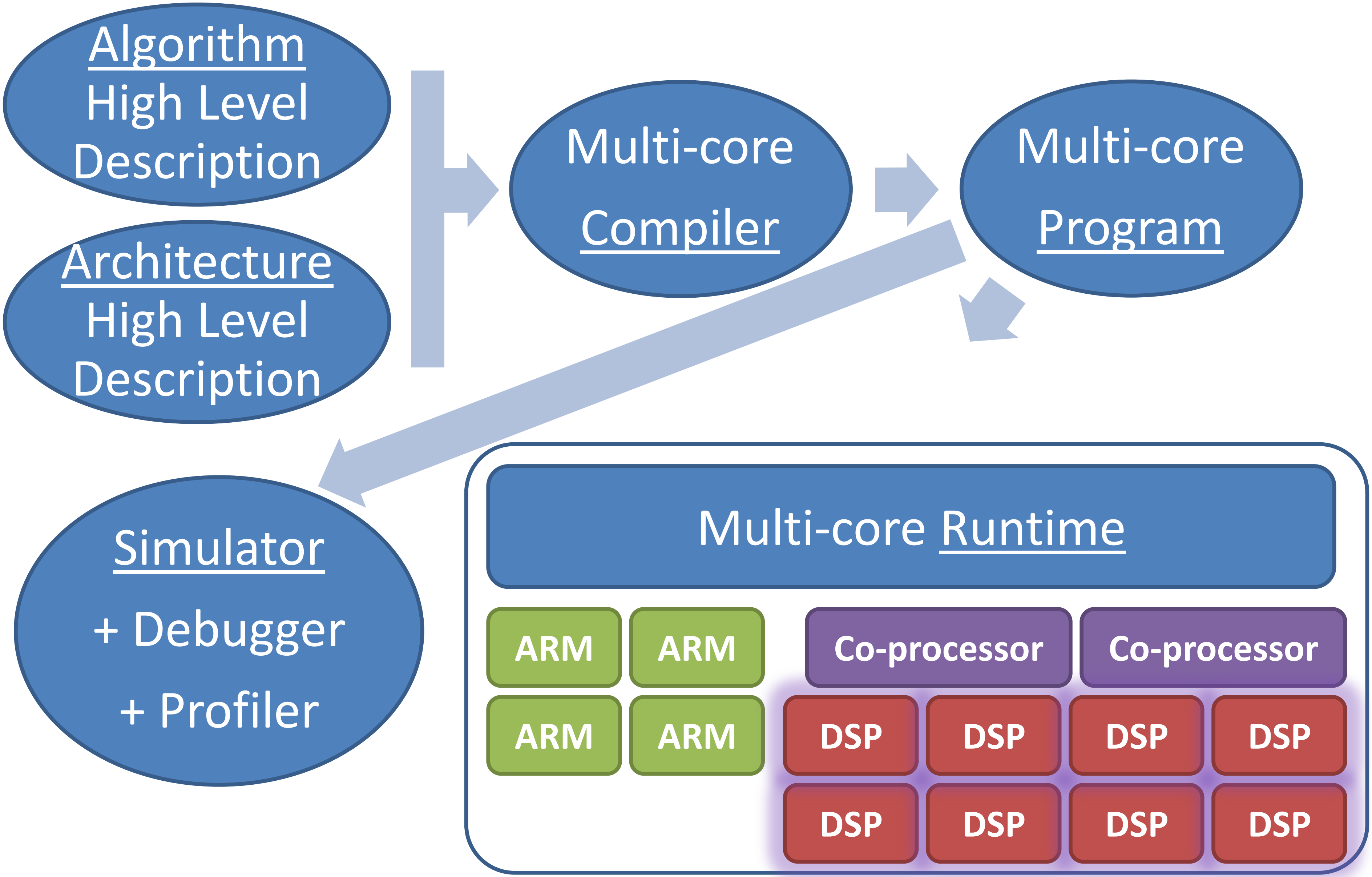


- Optimizing / Offering trade-offs between
  - Latency / Response time
  - Throughput
  - Load Balancing
  - Memory consumption
  - Power consumption
- Algorithm description portable to new device (DSP, GPU, HPC…)

General Outline

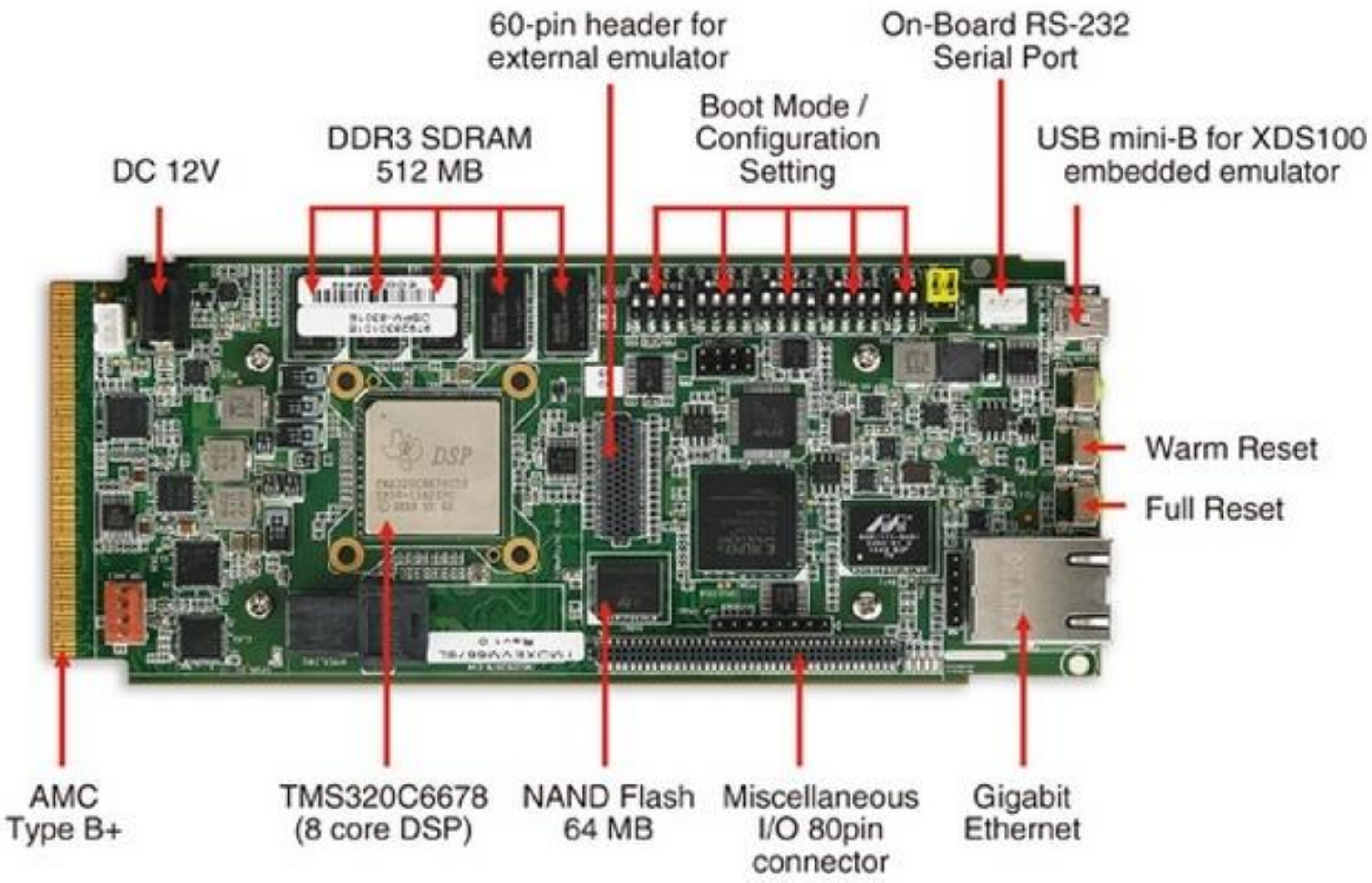
- Demo Platform
- Applications
- Models of Computation
- Architectures
- Models of Architecture
- Partitioning and Scheduling Problem
- Compile-time and Runtime Tools

Demo Platform



Advantech Board TDMSEVM6678L

- **EVM = Evaluation Module for theTMS320C6678**


60-pin header for external emulator
On-Board RS-232 Serial Port
Boot Mode / Configuration Setting
DDR3 SDRAM 512 MB
DC 12V
USB mini-B for XDS100 embedded emulator
Warm Reset
Full Reset
AMC Type B+
TMS320C6678 (8 core DSP)
NAND Flash 64 MB
Miscellaneous I/O 80pin connector
Gigabit Ethernet

TMS320C6678

- **8-Core DSP**

  8 C66x DSP Core Subsystems (C66x CorePacs), Each with:

  – 1.0 GHz or 1.25 GHz C66x Fixed/Floating-Point CPU Core

    40 GMAC/Core for Fixed Point @ 1.25 GHz

    20 GFLOP/Core for Floating Point @ 1.25 GHz

    Total: 320 GMACs + 160 Gflops: hard to reach!

  – 2 levels of Core Memory

    32K Byte L1P Per Core

    32K Byte L1D Per Core

    512K Byte Local L2 Per Core

- **4 MB of Internal Shared Memory**

  Multicore Shared Memory Controller (MSMC)

  L1D and L1P with automatic cache coherency in local

  Non coherent cache of the shared memory

- **Unified memory space for internal/external memory**

Demo Board

- **512 MB of Shared DDR3 on the emulation board**

  Any core can access DDR3, 8G Byte of DDR3 Addressable Memory

- **Hardware coprocessors**

  For repetitive common operations

  Reduced because multi-purpose processor

  Cryptography

  Network

- **XDS510 JTAG**

  via USB

  Possibility to extend to XDS560 via extension

- **Packaging**

  40nm technology, 841-Pin Flip-Chip Plastic BGA (CYP)

Inter-core Communication

- **KeyStoneTeraNet switch fabric (Network on Chip)**
- **Core Interrupt Controller**

- **Enhanced Direct Memory Access v3 (EDMA3)**
  - Data movement
  - Like a core with only MOV instructions

- **Multicore Navigator**
  - 8192 Multipurpose Hardware Queues with Queue Manager
  - Data movement or zero-copy

- **Shared MSMC and DDR3**
  - Data movement or zero-copy

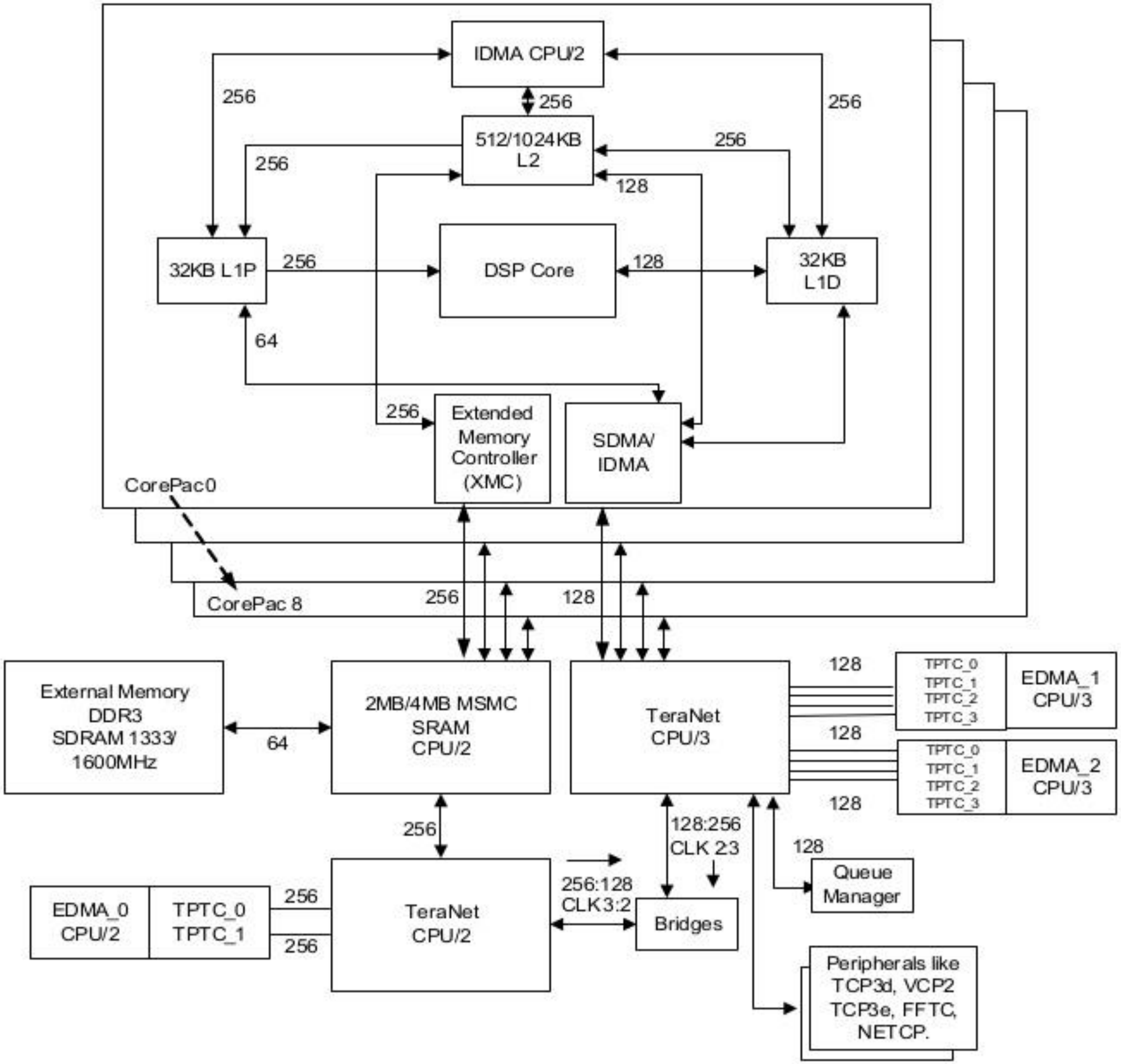
Inter-core Communication

- **Multicore Navigator**
  - Queue Manager Subsystem (QMSS)
  - Packet DMA (PKTDMA) for Zero-Overhead Transfers
  - Packet passing system between cores
  - Abstracts real data transfer

- **Open Event Machine**
  - Software runtime system provided by TI to offload code on cores
  - Event driven processing runtime for multicore

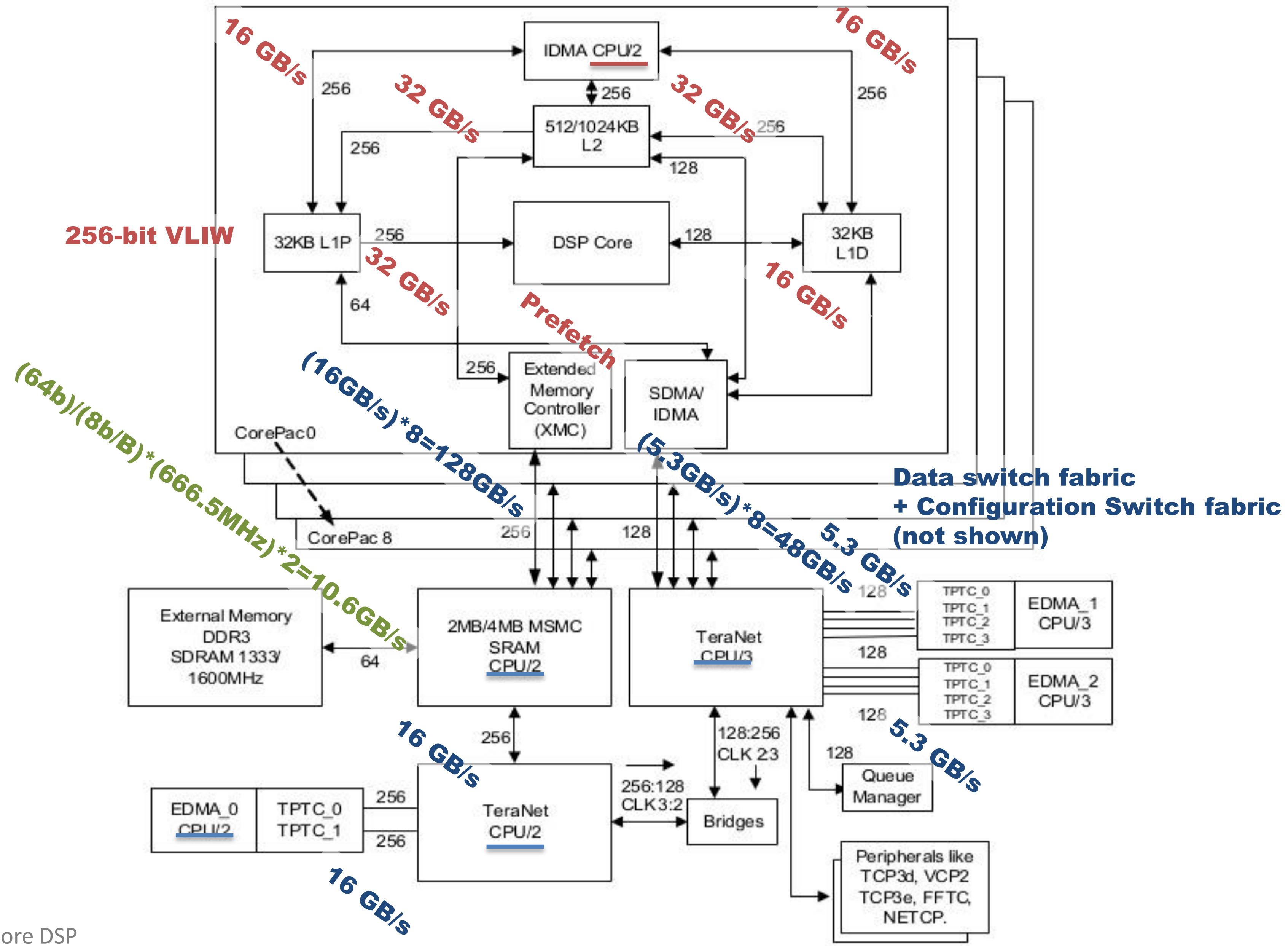
Source: sprabh2a TI

Inter-core Communication



Source: sprabh2a TI

Inter-core Communication Throughputs



16 GB/s

32 GB/s

16 GB/s

32 GB/s

256-bit VLIW

32 GB/s

16 GB/s

Prefetch

(16GB/s)*8=128GB/s

(5.3GB/s)*8=48GB/s

(64b)/(8b/B)*(666.5MHz)*2=10.6GB/s

5.3 GB/s

Data switch fabric
+ Configuration Switch fabric
(not shown)

16 GB/s

5.3 GB/s

16 GB/s

Cache Access Latencies

- **Cache operations**

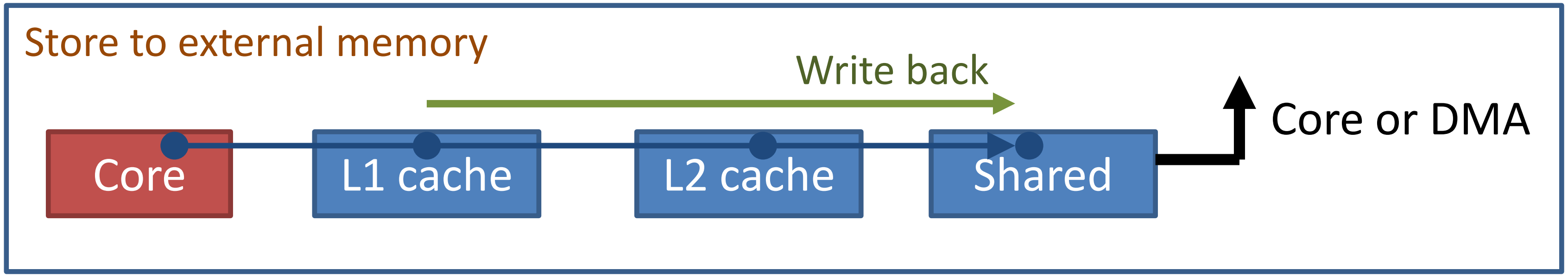
  The L1 has automatic cache coherence if local L2 is modified

  L1 has no automatic cache coherence if non local memory is modified

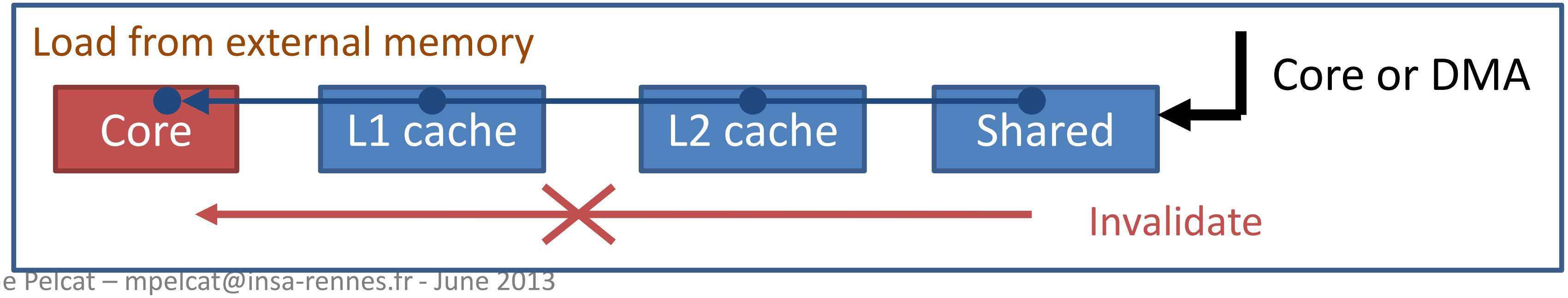
  L2 has no automatic cache coherency

  →L1 and L2 cache « write back » and « invalidate » must be called

  Up layer memory request: write back if modifications



  Low layer memory request: invalidate if modifications



Source: sprabh2a TI

Data Alignment and Performance

- **Caches have a strong effect on memory latency**

				Single Read	Single Read	Burst Read	Burst Read
	L1 Cache	**L2 Cache**	**XMC Prefetch**	**No Victim**	**Victim**	**No Victim**	**Victim**
All	**Hit**	NA	NA	0	NA	**0**	NA
Local L2	**Miss**	NA	NA	7	7	**3,5**	**10**
MSMC (SL2)	**Miss**	NA	**Hit**	7,5	7,5	7,4	11
MSMC (SL2)	**Miss**	NA	**Miss**	19,8	20,1	9,5	11,6
MSMC (SL3)	**Miss**	**Hit**	NA	9	9	4,5	4,5
MSMC (SL3)	**Miss**	**Miss**	**Hit**	10,6	15,6	9,7	129,6
MSMC (SL3)	**Miss**	**Miss**	**Miss**	22	28,1	**11**	**129,7**
DDR (SL2)	**Miss**	NA	**Hit**	9	9	23,2	59,8
DDR (SL2)	**Miss**	NA	**Miss**	84	113,6	41,5	113
DDR (SL3)	**Miss**	**Hit**	NA	9	9	4,5	4,5
DDR (SL3)	**Miss**	**Miss**	**Hit**	12,3	59,8	30,7	287
DDR (SL3)	**Miss**	**Miss**	**Miss**	89	123,8	**43,2**	**183**

Signal Input/Output

- **Four Lanes of SRIO 2.1**

  1.24 to 5 GBaud Operation Supported Per Lane → up to 20 Gbauds

- **PCIe Gen2**

  Single port supporting 1 or 2 lanes

  Supports Up To 5 GBaud Per Lane → up to 10 Gbauds

- **HyperLink**

  Supports Connections to Other KeyStone → up to 50 Gbauds

  Architecture Devices Providing Resource Scalability

- **Gigabit Ethernet (GbE) Switch Subsystem**

  Two SGMII Ports

  Supports 10/100/1000 Mbps operation → up to 2 Gbps

- **Other ports**

  UART Interface, I2C Interface, 16 GPIO Pins, SPI Interface

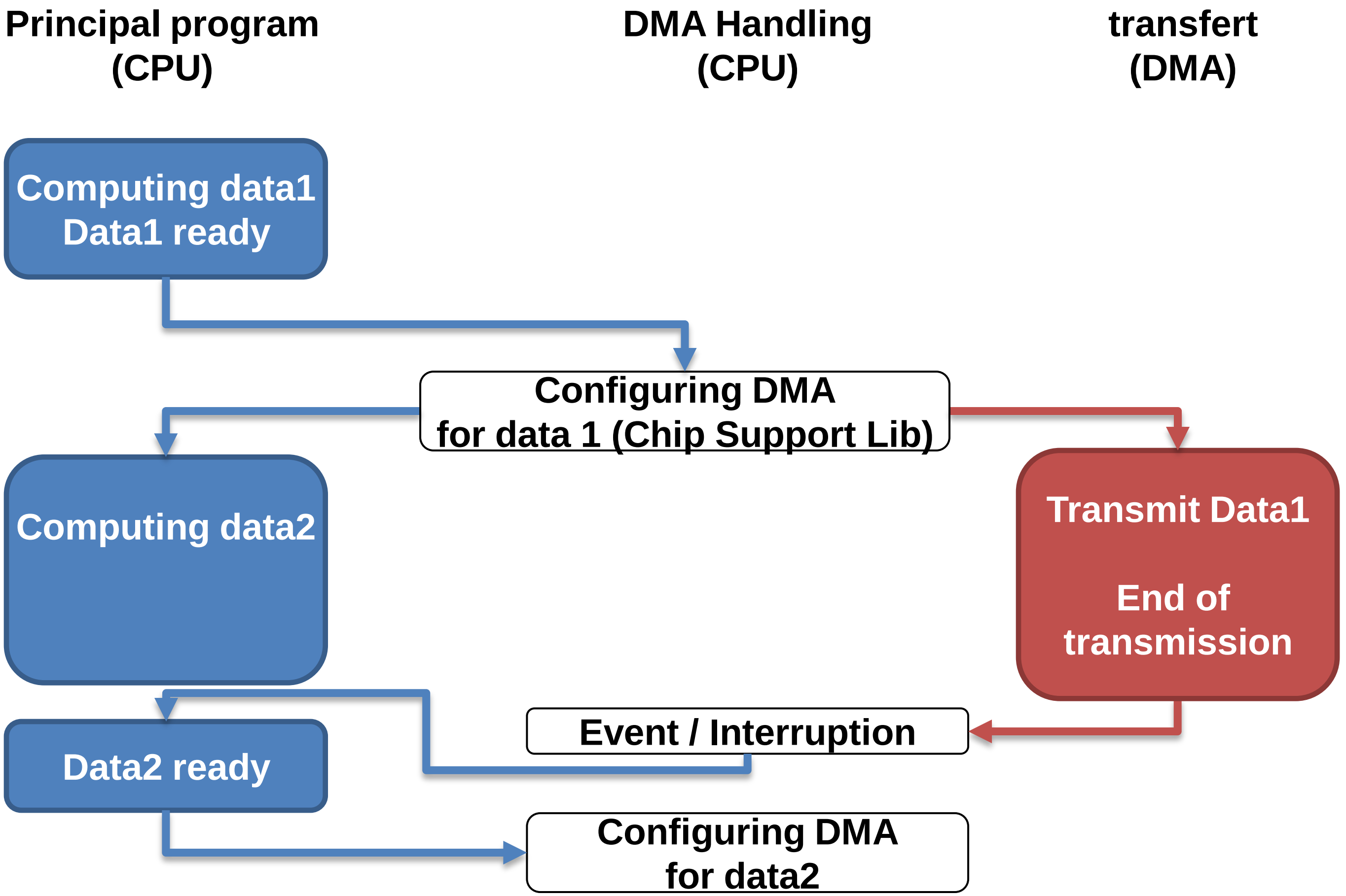
- **Remark**

  Uncompressed 1920x1080 4:2:0 video @ 60Hz = 1.5 Gbps

Code Composer Studio Software

- **Code Composer Studio v5 (CCS) IDE**
  - Based on Eclipse 3.7 Indigo
  - Runs under Windows and Linux
  - Integrable in an existing Eclipse
- **C66x compiler, linker, assembler, simulator…**
  - Delivered with CCS IDE
- **EVM Drivers**
  - Installed with CCS
  - Connect to the EVM JTAG (breakpoints…)

Using DMAs


Principal program
(CPU)
DMA Handling
(CPU)
transfert
(DMA)
Computing data1
Data1 ready
Configuring DMA
for data 1 (Chip Support Lib)
Computing data2
Transmit Data1
End of
transmission
Event / Interruption
Data2 ready
Configuring DMA
for data2

SYS/BIOS

SYS/BIOS

- **Multi-task OS :**

  enables sharing the DSP between several tasks

- **OS with Static and Dynamic configuration**

  Static : « configuration tool », .cfgfile

  Dynamic : specific functions to access interruptions, task creation, logs…

- **Gives many informations on the system for debug**

- **Preemptive RTOS**

  Scheduler

  task priorities

- **Alternative to DSP-BIOS:**

  Enea Solutions : other RTOS for C6x

DSP BIOS Limits

Memory cost

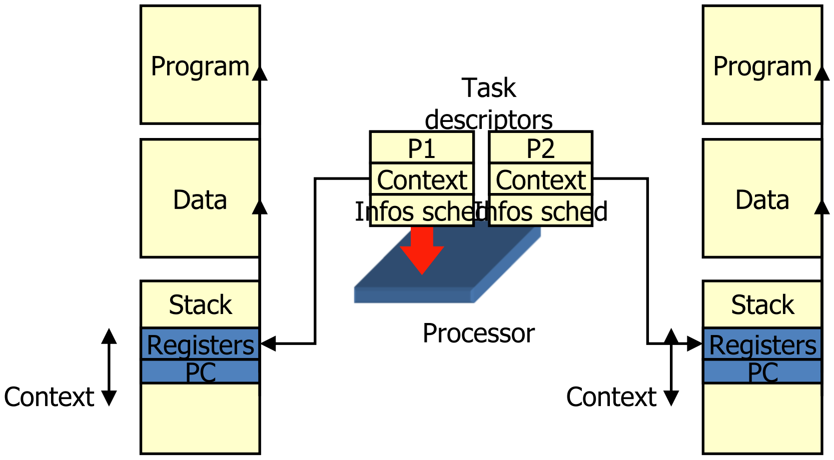
→ Modular but each module has a non negligible cost.

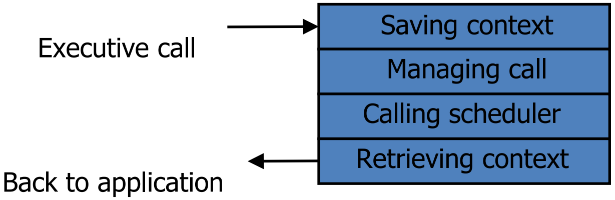
Mono-core system

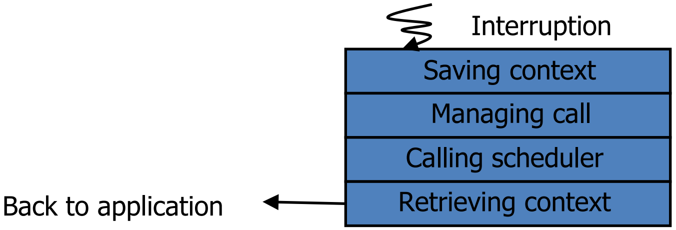
→No multicore OS

Time cost :

system calls
Interruption calls

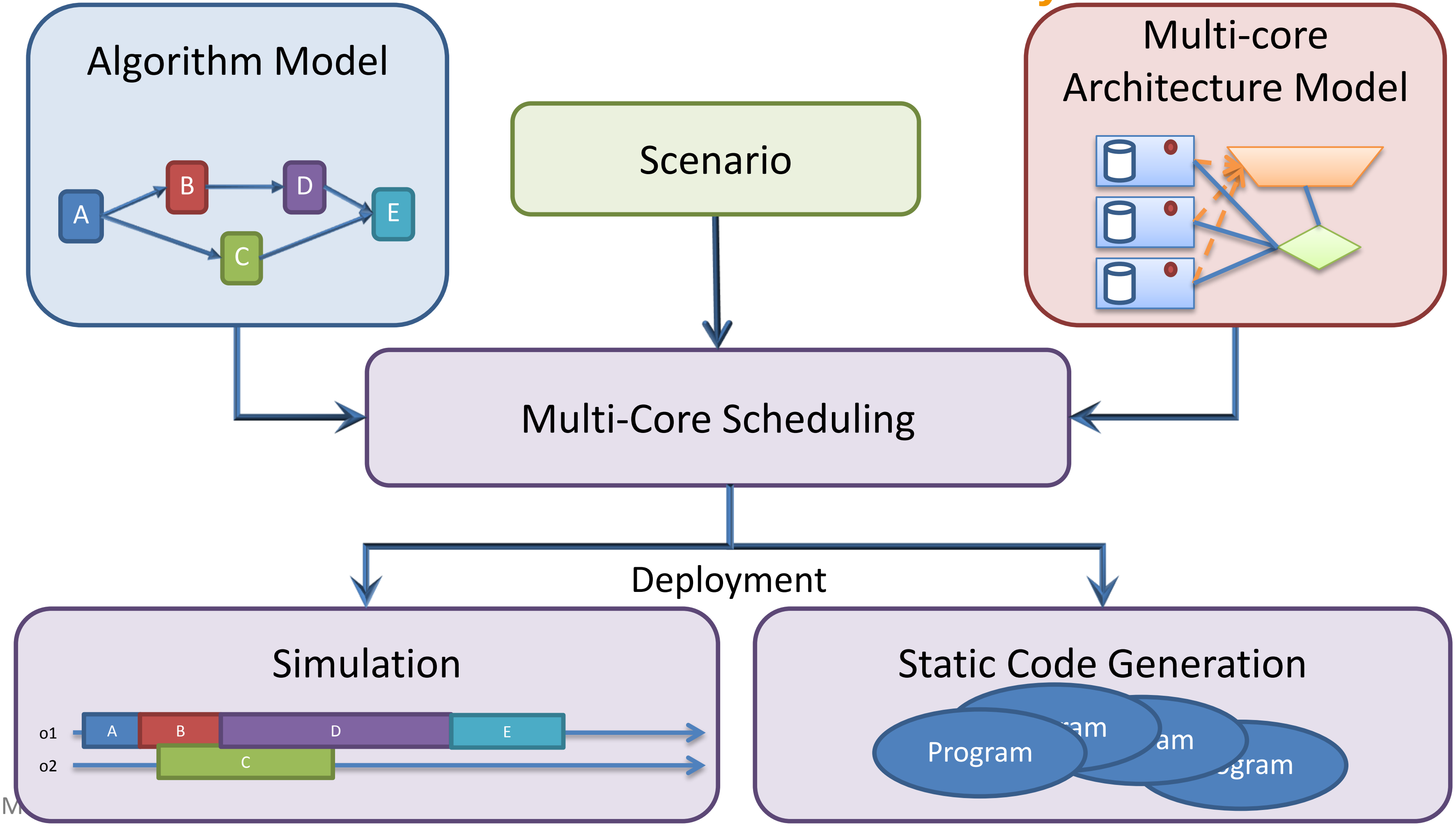






Preesm Software

- Developed at IETR under Eclipse
- From a dataflow graph to a multicore code execution
- Automates multicore communication/synchronization



Algorithm Model

Scenario

Multi-core Architecture Model

Multi-Core Scheduling

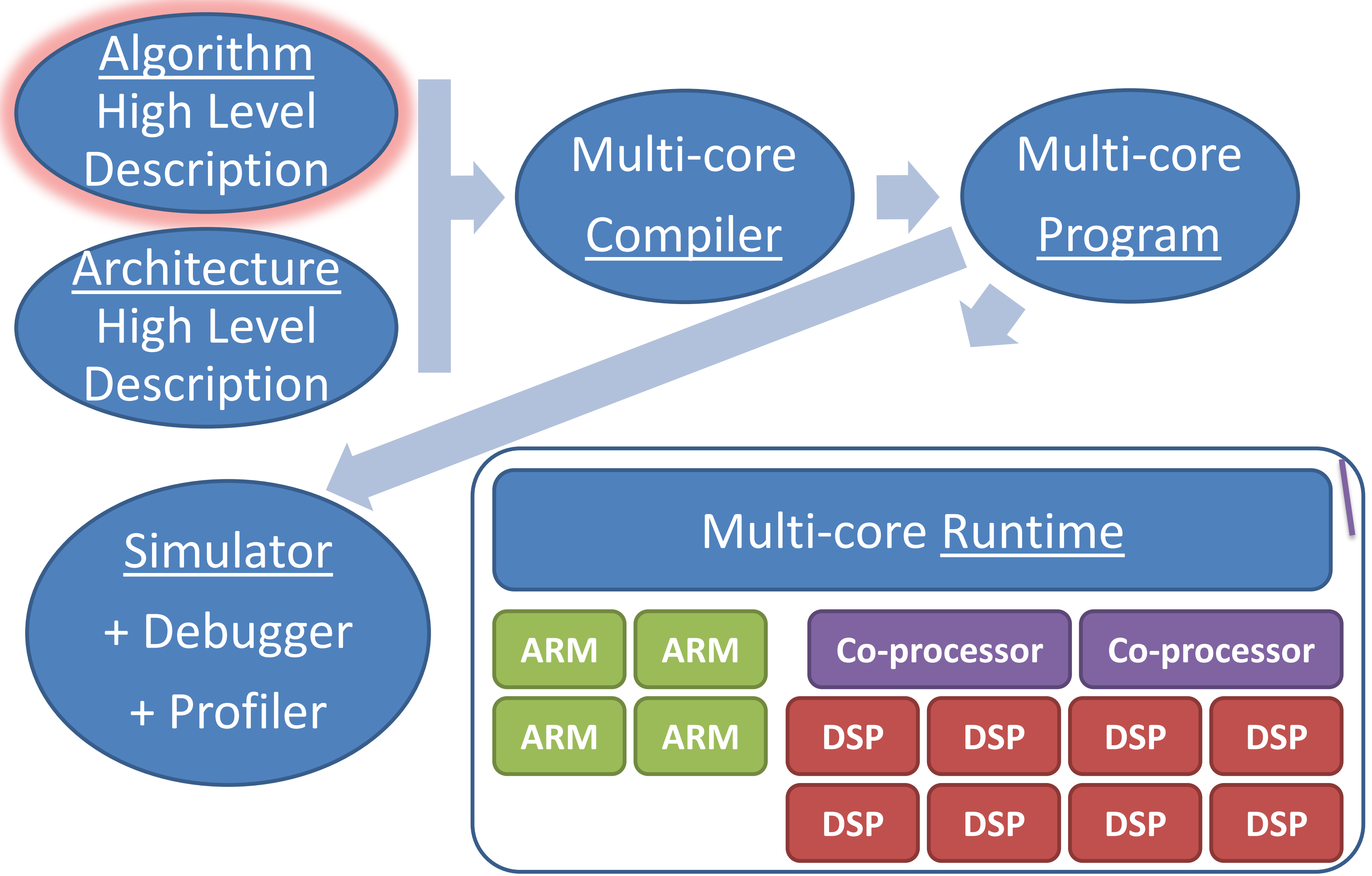
Deployment

Simulation

Static Code Generation

Institut National des Sciences Appliquées

Multi-core DSP Programming



High Performance DSP Applications

- Overview
- Standization Processes
- MPEG HEVC
- 4G

High Performance DSP Applications : Overview

- **Embedded system applications & High Performance computing applications**
  - Base stations & software-defined radio
  - Image and Audio processing
  - Industrial control systems
  - Aeronautics & transports
  - Radar / Sonar
  - Medical
  - Scientific computing & numerical simulation
    - High Performance Computing (HPC)
  - …

High Performance DSP Applications : Overview

- **Embedded system applications & High Performance computing applications**

	Digital Media Processors	OMAP Applications Processors	C6000 Digital Signal Processors	C5000 Digital Signal Processors	C2000 Microcontrollers	MSP430 Microcontrollers	Stellaris 32-Bit ARM Cortex-M3 MCUs
Audio	🟨	🟦	🟩	🟦			🟥
Automotive	🟨	🟦			🟪		
Communications	🟨		🟩	🟦	🟪		🟥
Industrial	🟨	🟦	🟩	🟦	🟪	🟧	🟥
Medical	🟨	🟦	🟩	🟦	🟪	🟧	🟥
Security	🟨	🟦				🟧	🟥
Video	🟨	🟦	🟩				
Wireless		🟦	🟩	🟦		🟧	🟥
Key Feature	Complete tailored video solution	Low power and high performance	High performance	Power-efficient performance	Performance, integration for greener industrial applications	Ultra-low power	Open architecture software, rich communications options

Source: TI

High Performance DSP Applications : Overview

- **Typical types of operations / tasks / actors**
  - low-pass, band-pass, high-pass and adaptive filtering (FIR and IIR filters)
  - cross/auto, linear/circular correlation (similarity between signals)
  - Convolution (equivalent to multiplication in Fourier domain)
  - transformations between domains (Fast Fourier, DCT, Hadamard, wavelet, Hilbert, Wigner-Ville...)
  - noise removal
  - power computation
  - independent component analysis
  - expected signal detection and extraction
  - data prediction (temporal, spatial)
  - entropy coding
  - complex, vector and matrix operations
  - forward error correction
  - ...

Standardization Processes

- **There are many standardization organizations**
- **Famous standardization organizations regarding signal processing include:**
  - ISO (International Organization for Standardization)
  - IEEE (Institute of Electrical and Electronics Engineers)
  - ITU (International Telecommunication Union)
  - 3GPP (Third Generation Partnership Project)

- **MPEG HEVC Video Compression Standard**
  - developed by the ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC JTC1 Moving Picture Experts Group (MPEG)
- **3GPP LTE Radio Telecommunication Standard**
  - developed by the 3GPP (3rd Generation Partnership Project)
  - Respecting (partially) the ITU-R organization IMT-Advanced specification

Institut National des Sciences Appliquées

DSP Application : MPEG AVC and HEVC

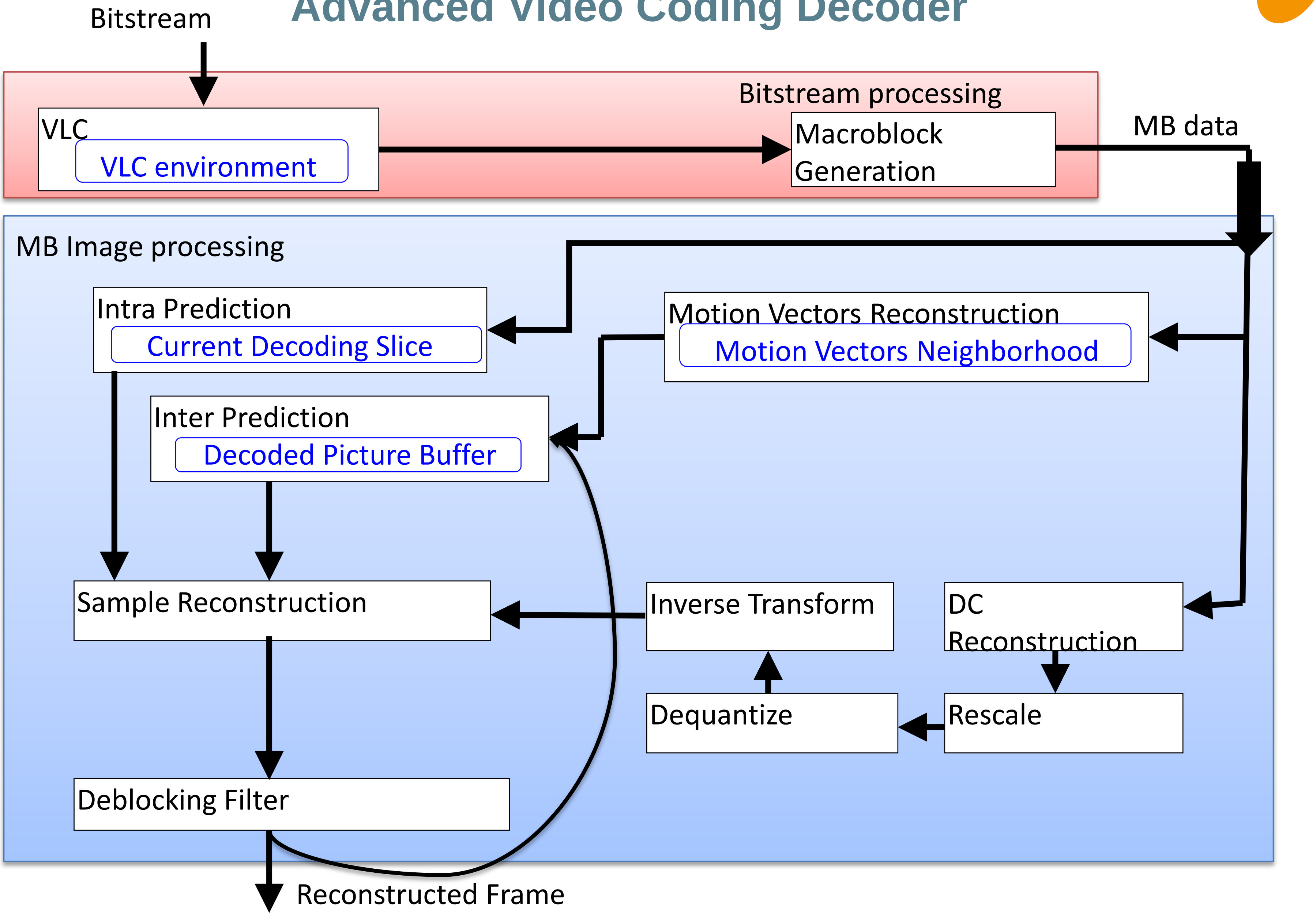
MPEG Standards

- **Mpeg-1 : (1992)**
- **Mpeg-2 : (1994)**
- **Mpeg-4 : (since 1998)**

  Example : Mpeg-4 Part 2 (DivX until v5,Xvid)

  Extension1: Mpeg-4 part 10 = H.264 (ITU-T) = AVC (Advanced Video Coding)
- **Each standard : better compression (HEVC: HD@4Mb/s)**

Advanced Video Coding Decoder



Bitstream

Bitstream processing

VLC

VLC environment

Macroblock Generation

MB data

MB Image processing

Intra Prediction

Current Decoding Slice

Motion Vectors Reconstruction

Motion Vectors Neighborhood

Inter Prediction

Decoded Picture Buffer

Sample Reconstruction

Inverse Transform

DC Reconstruction

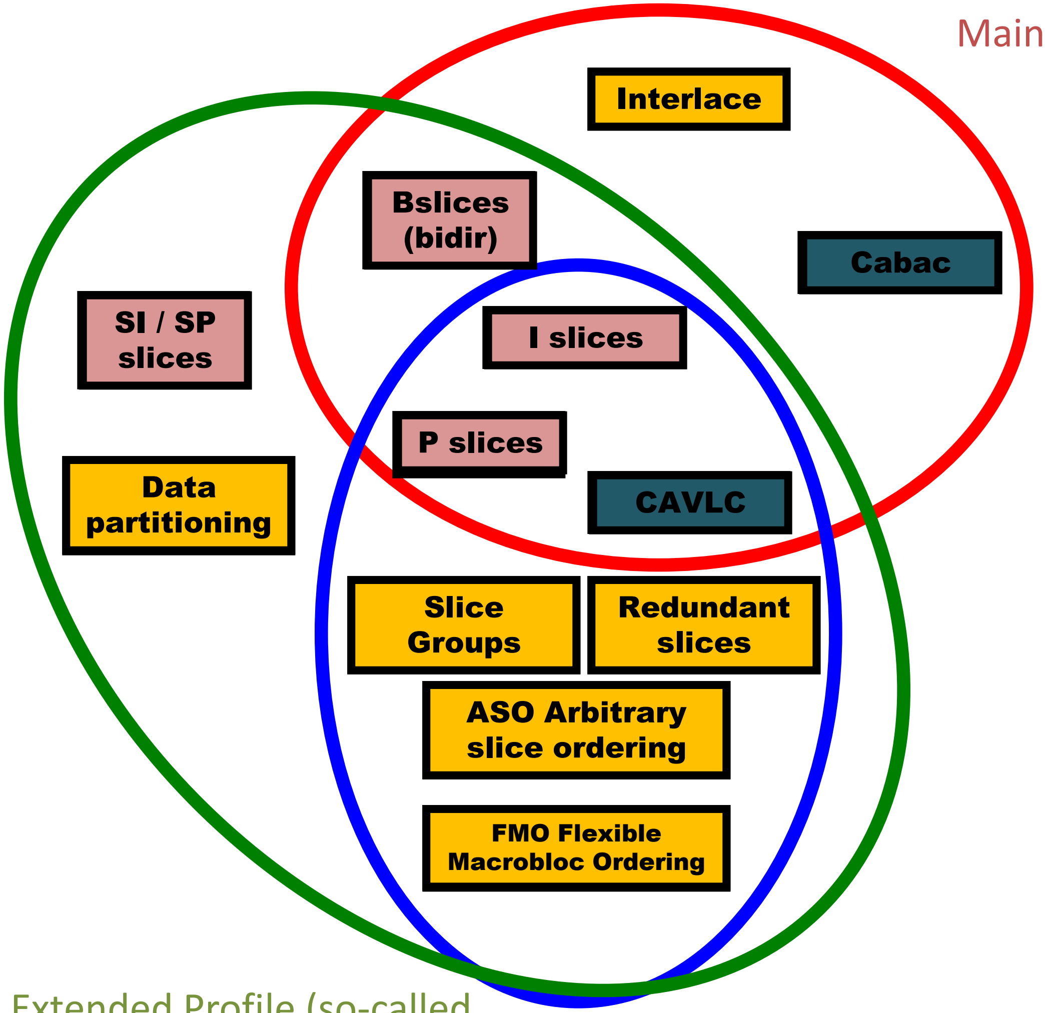
Dequantize

Rescale

Deblocking Filter

Reconstructed Frame

AVC Profiles



Main/High Profile

Interlace

Bslices (bidir)

Cabac

SI / SP slices

I slices

P slices

CAVLC

Data partitioning

Slice Groups

Redundant slices

ASO Arbitrary slice ordering

FMO Flexible Macrobloc Ordering

Extended Profile (so-called streaming profile)

Baseline (low latency)

AVC Baseline	Low Delay, Lower Processor Load
AVC Main/High	Supports Interlaced video, B-Frames

→ Predictions

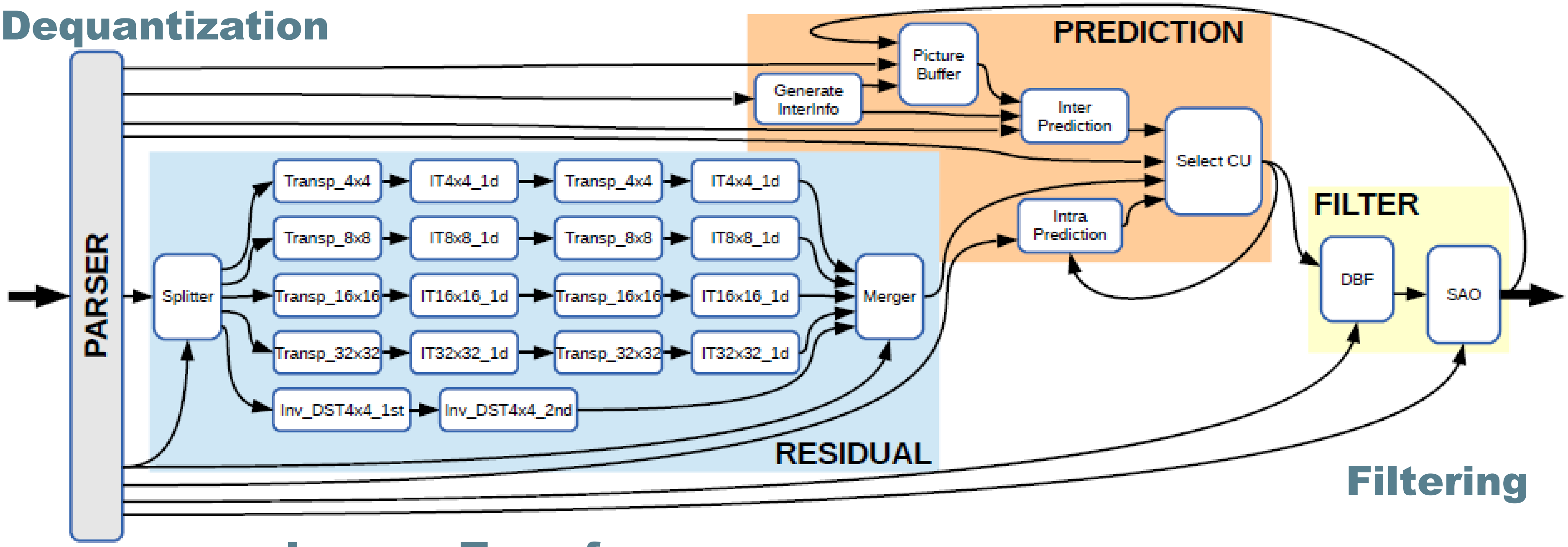
→ Entropy coding

→ Buffer management

HEVC Decoder

VLC Decoding

Dequantization



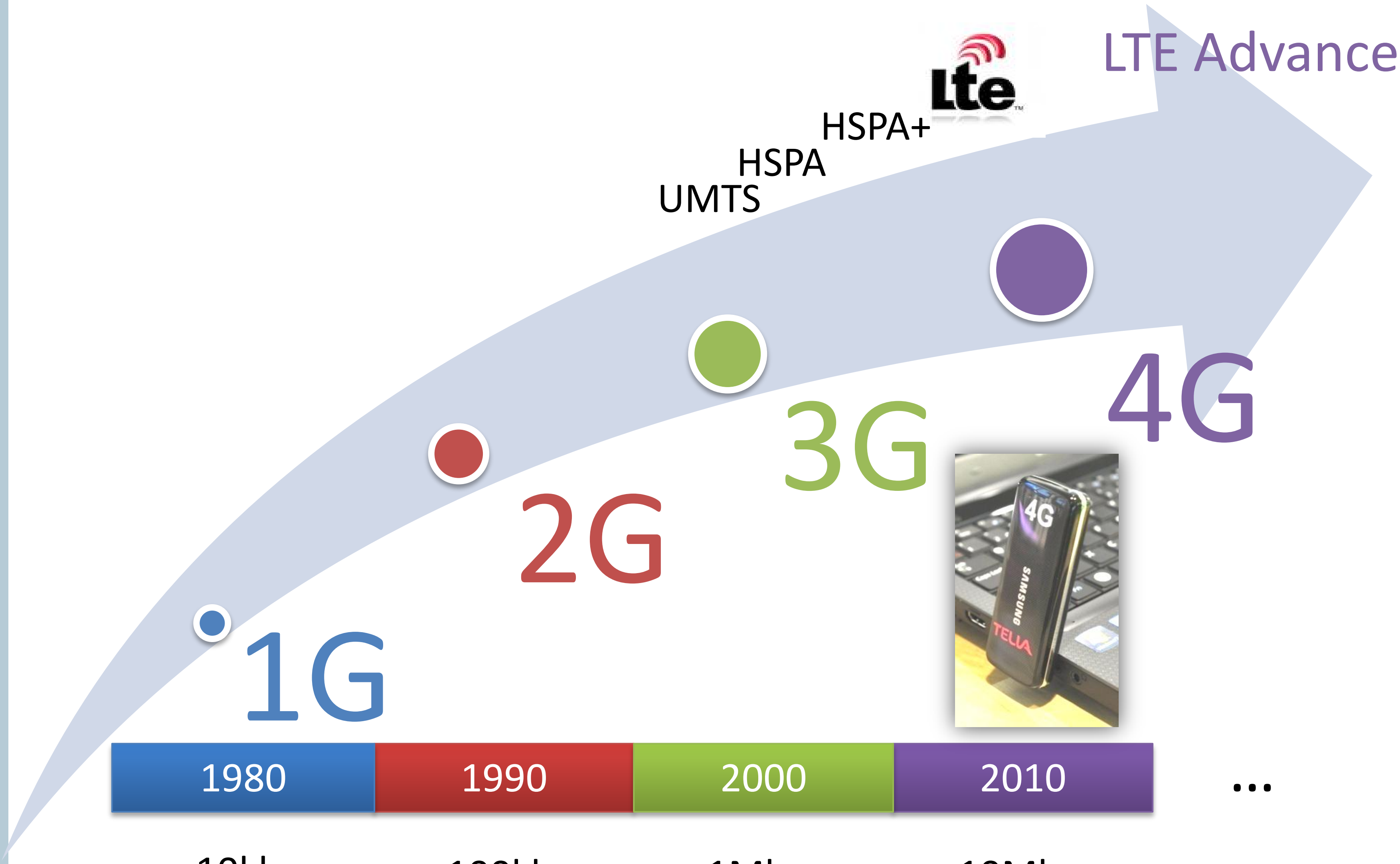
Filtering

Inverse Transform

Source: Hervé Yviquel

DSP Application : 4G

3GPP Standards



LTE Advanced

Lte

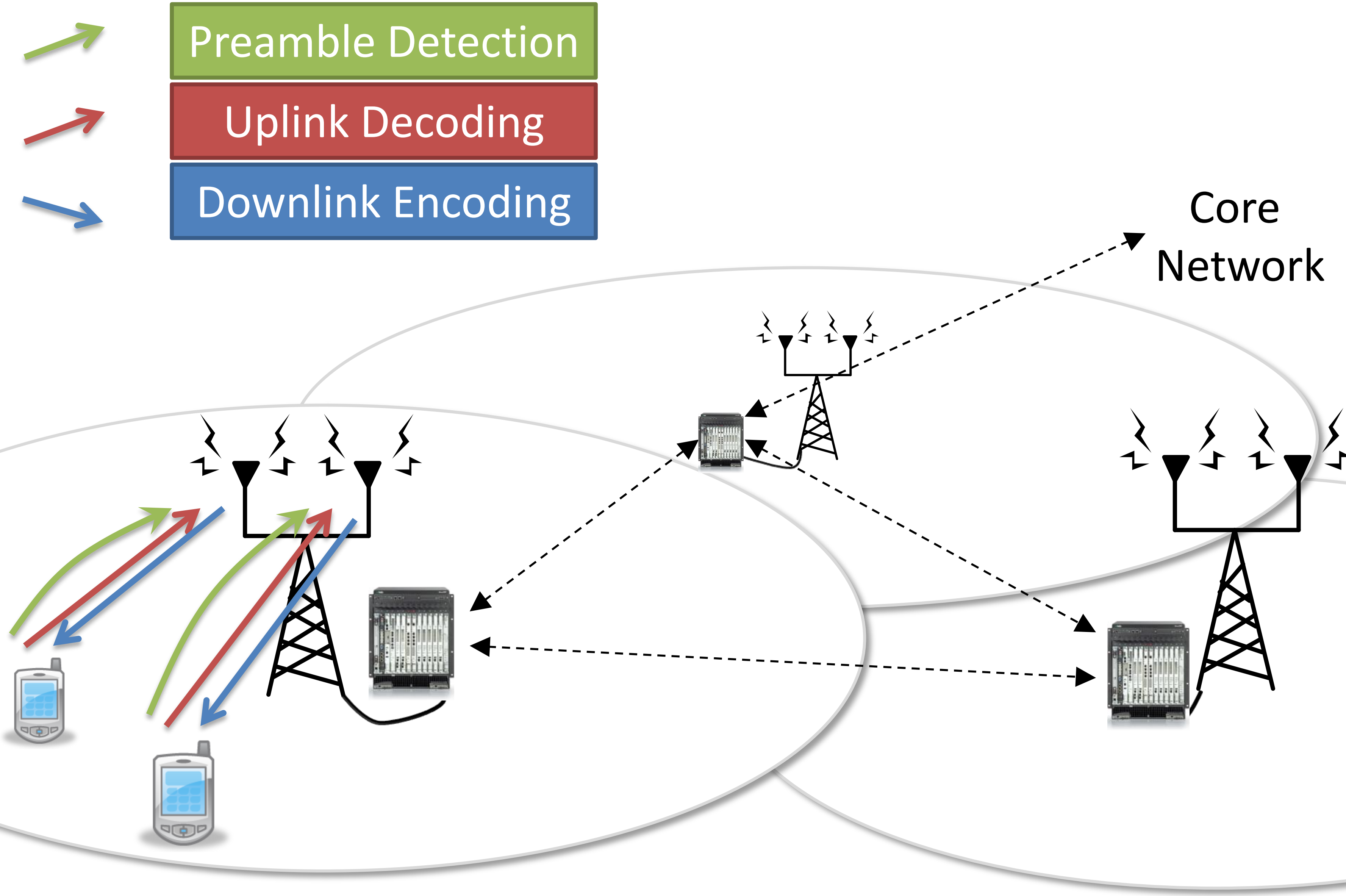
HSPA+

HSPA

UMTS

1G	2G	3G	4G	
1980	1990	2000	2010	…
10kbps	100kbps	1Mbps	10Mbps	

LTE Access Network


Preamble Detection

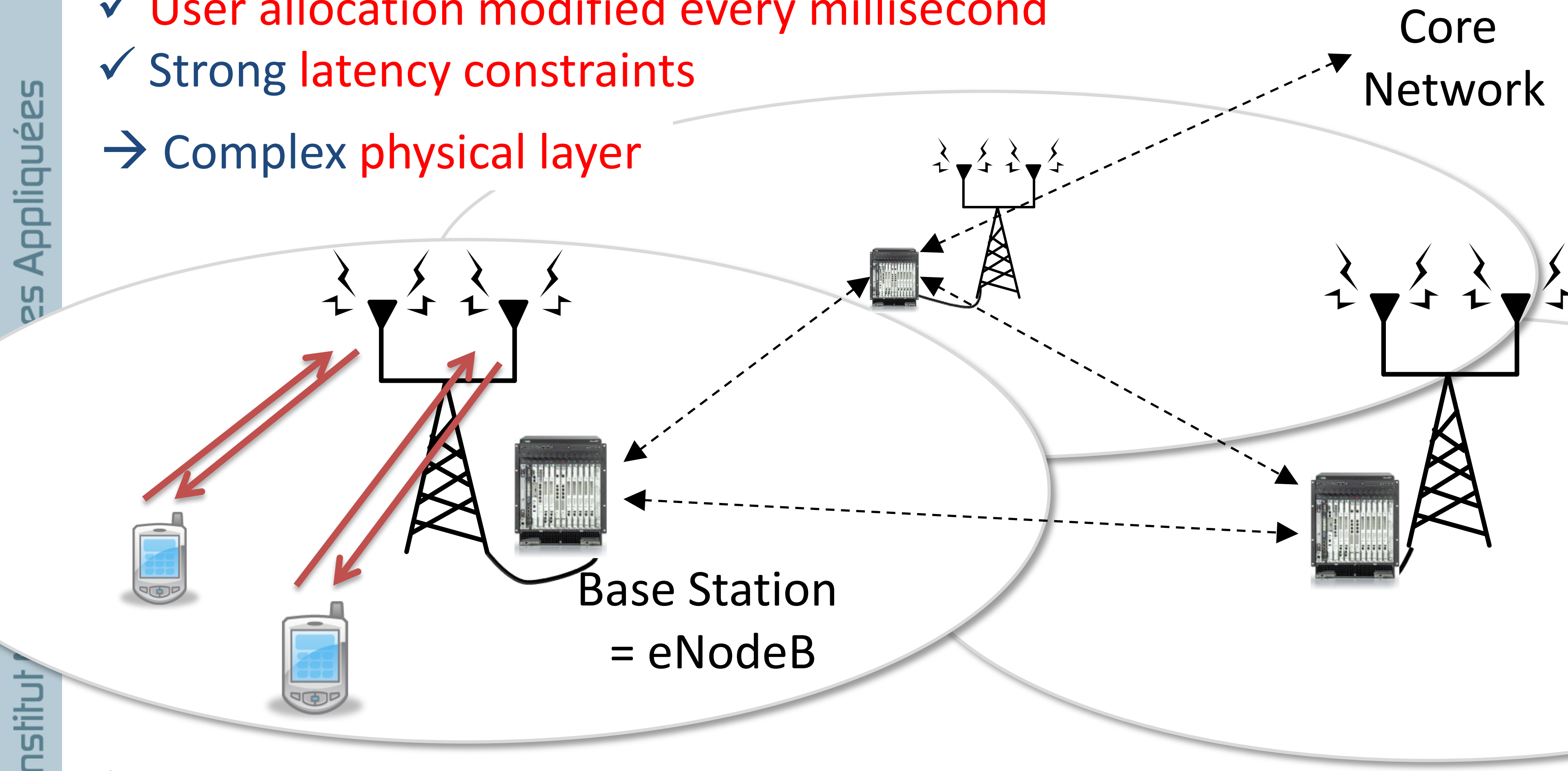
Uplink Decoding

Downlink Encoding

3GPP Long Term Evolution rel. 9

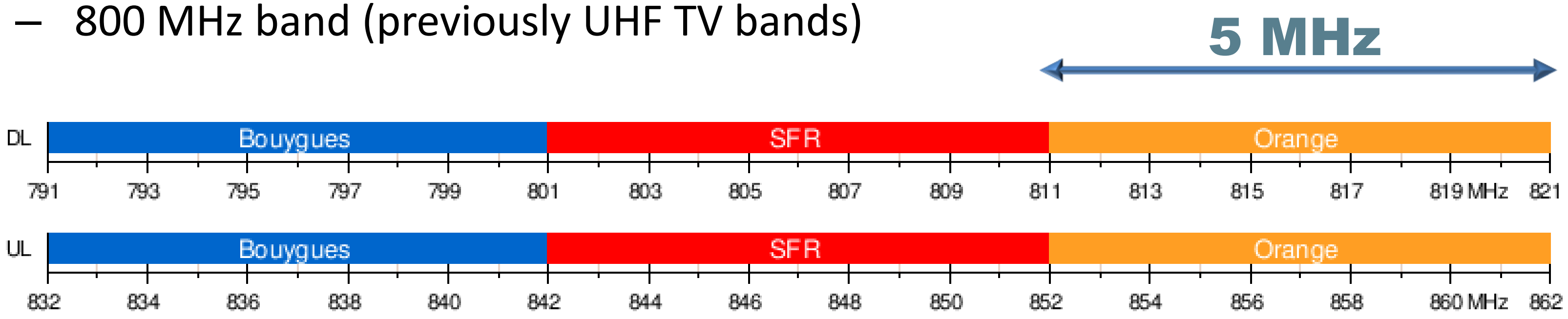
- ✓ Up to 100 Mbps in downlink, 50 Mbps in uplink
- ✓ Frequency Division Multiplexing, Multiple Input Multiple Output
- ✓ Dozens of Users communicating concurrently
- ✓ User allocation modified every millisecond
- ✓ Strong latency constraints

→ Complex physical layer

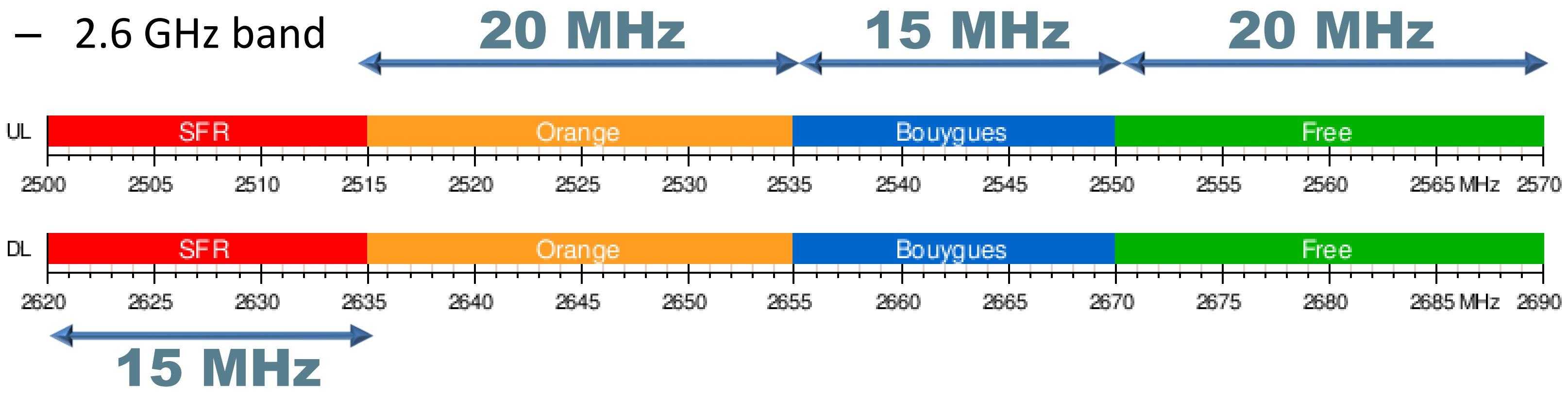


Frequency Allocations

- 3GPP LTE Frequency Allocations in France (ACERP october 2011)
  - 800 MHz band (previously UHF TV bands)



  - 2.6 GHz band



Source: Wikipedia

LTE Frequency division: subcarriers

- Spectrum flexibility
  - In both downlink and uplink

Bandwidth (MHz)	1.4	3	5	10	15	20
OFDM FFT size	128	256	512	1024	1536	2048
Number of available PRBs (downlink)	6	12	25	50	75	100
Number of available subcarriers	72	144	300	600	900	1200

Size (in complex values) of one symbol

3GPP LTE Rel. 9 Performance

- **High Data Rates**

  50Mbps(UpLink), 100Mbps (DownLink)
- **High User Equipment Speed**

  Walking to bullet-train (optimized up to 120 km/h)
- **Reduced Latency**

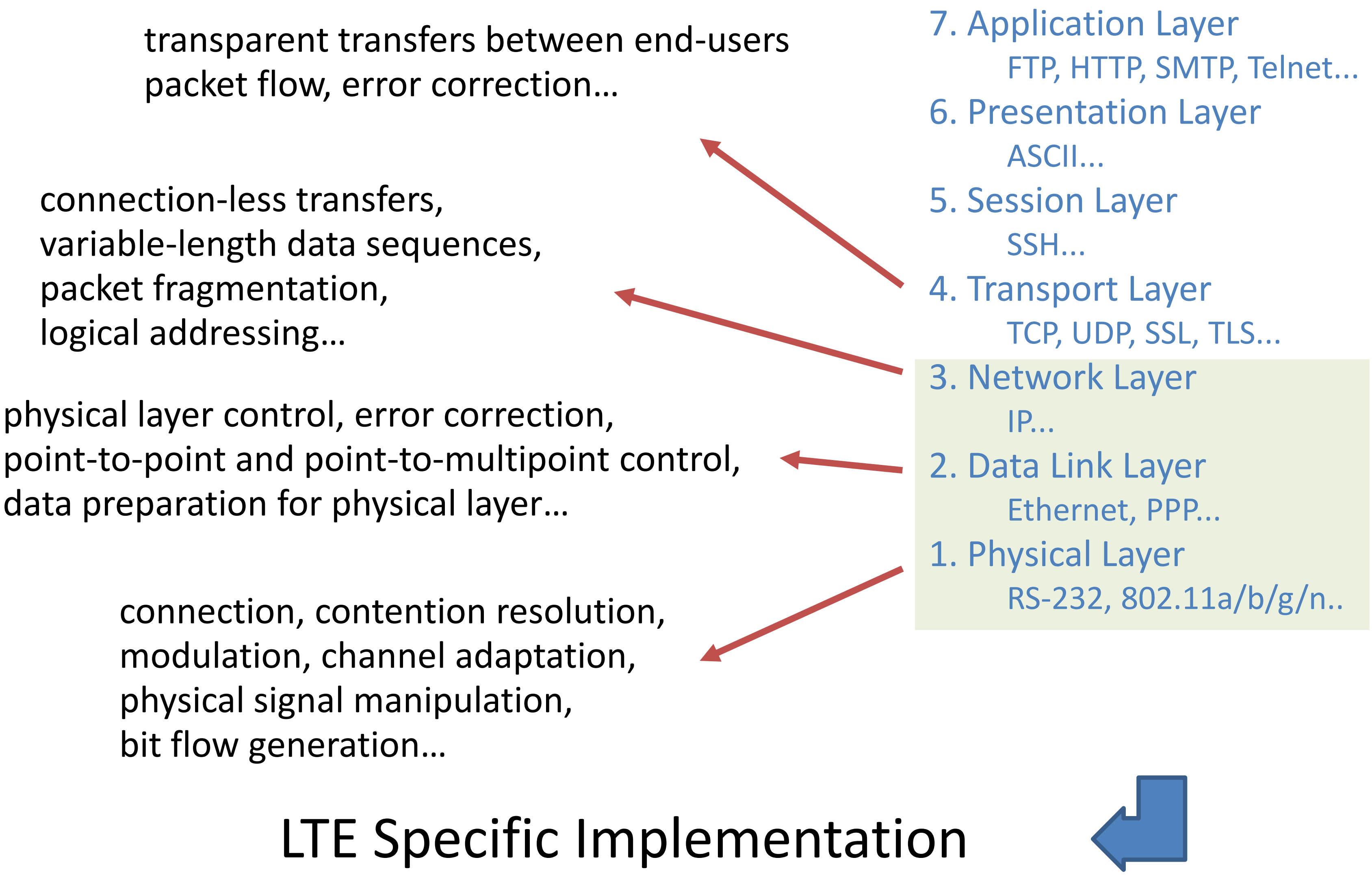
  Quick response time (under 5ms)
- **Cheap Roll-out**

  Bandwidth flexibility
- **Optimized for packet-switching**

  Good support for VoIP and data
- **Up to 100km radius cells (35km for GSM macrocells)**
- **Up to 100 user per cell**

- **Free to consult: search 36.211 and 36.212 in Google**

LTE and OSI Layers



7. Application Layer
   FTP, HTTP, SMTP, Telnet...
6. Presentation Layer
   ASCII...
5. Session Layer
   SSH...
4. Transport Layer
   TCP, UDP, SSL, TLS...
3. Network Layer
   IP...
2. Data Link Layer
   Ethernet, PPP...
1. Physical Layer
   RS-232, 802.11a/b/g/n..

transparent transfers between end-users
packet flow, error correction…

connection-less transfers,
variable-length data sequences,
packet fragmentation,
logical addressing…

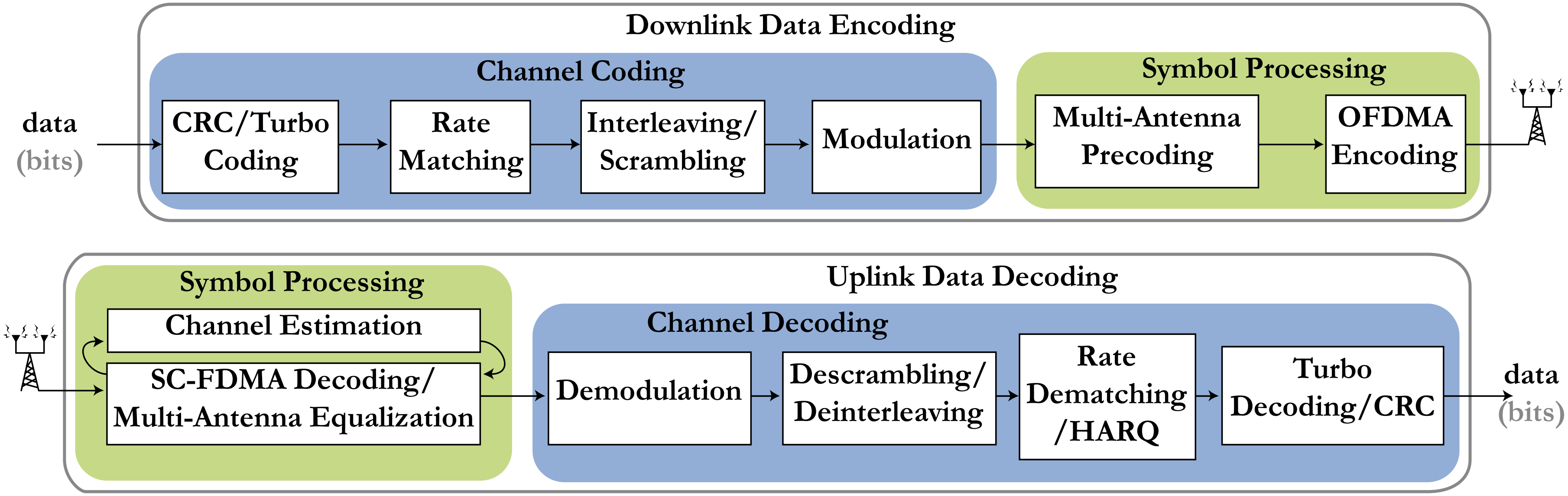
physical layer control, error correction,
point-to-point and point-to-multipoint control,
data preparation for physical layer…

connection, contention resolution,
modulation, channel adaptation,
physical signal manipulation,
bit flow generation…

LTE Specific Implementation

Downlink/Uplink in a Base Station

- Application is naturally described by a dataflow graph



Downlink Data Encoding

data (bits)

Channel Coding

CRC/Turbo Coding

Rate Matching

Interleaving/ Scrambling

Modulation

Symbol Processing

Multi-Antenna Precoding

OFDMA Encoding

Uplink Data Decoding

Symbol Processing

Channel Estimation

SC-FDMA Decoding/ Multi-Antenna Equalization

Channel Decoding

Demodulation

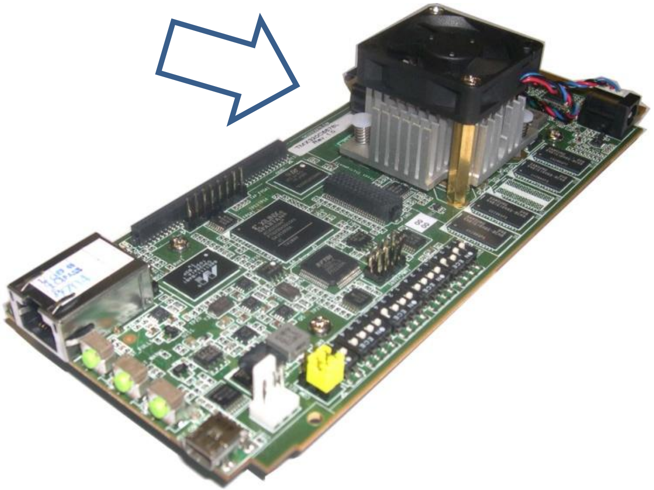
Descrambling/ Deinterleaving

Rate Dematching /HARQ

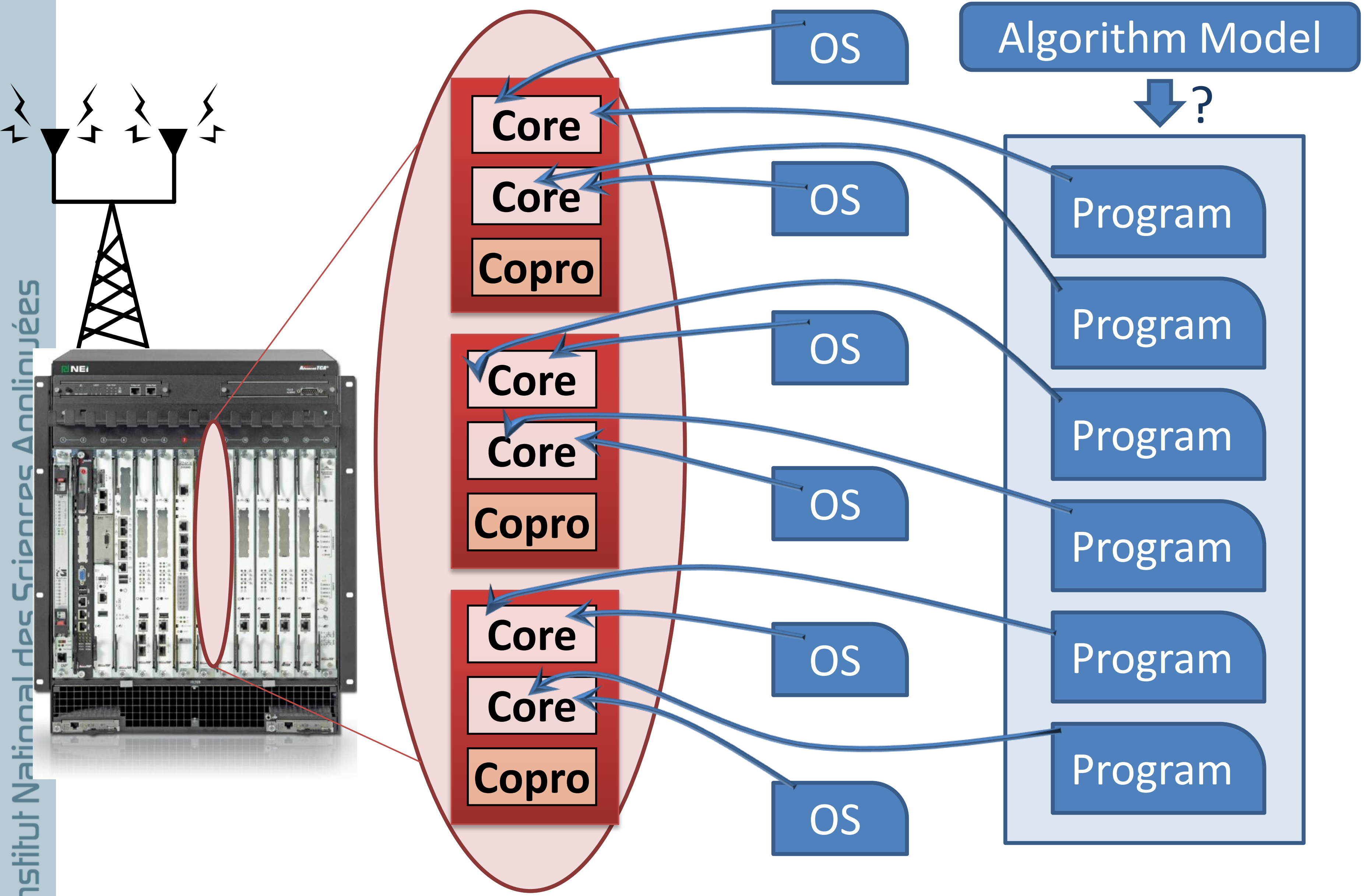
Turbo Decoding/CRC

data (bits)

- This application requires high DSP computing power

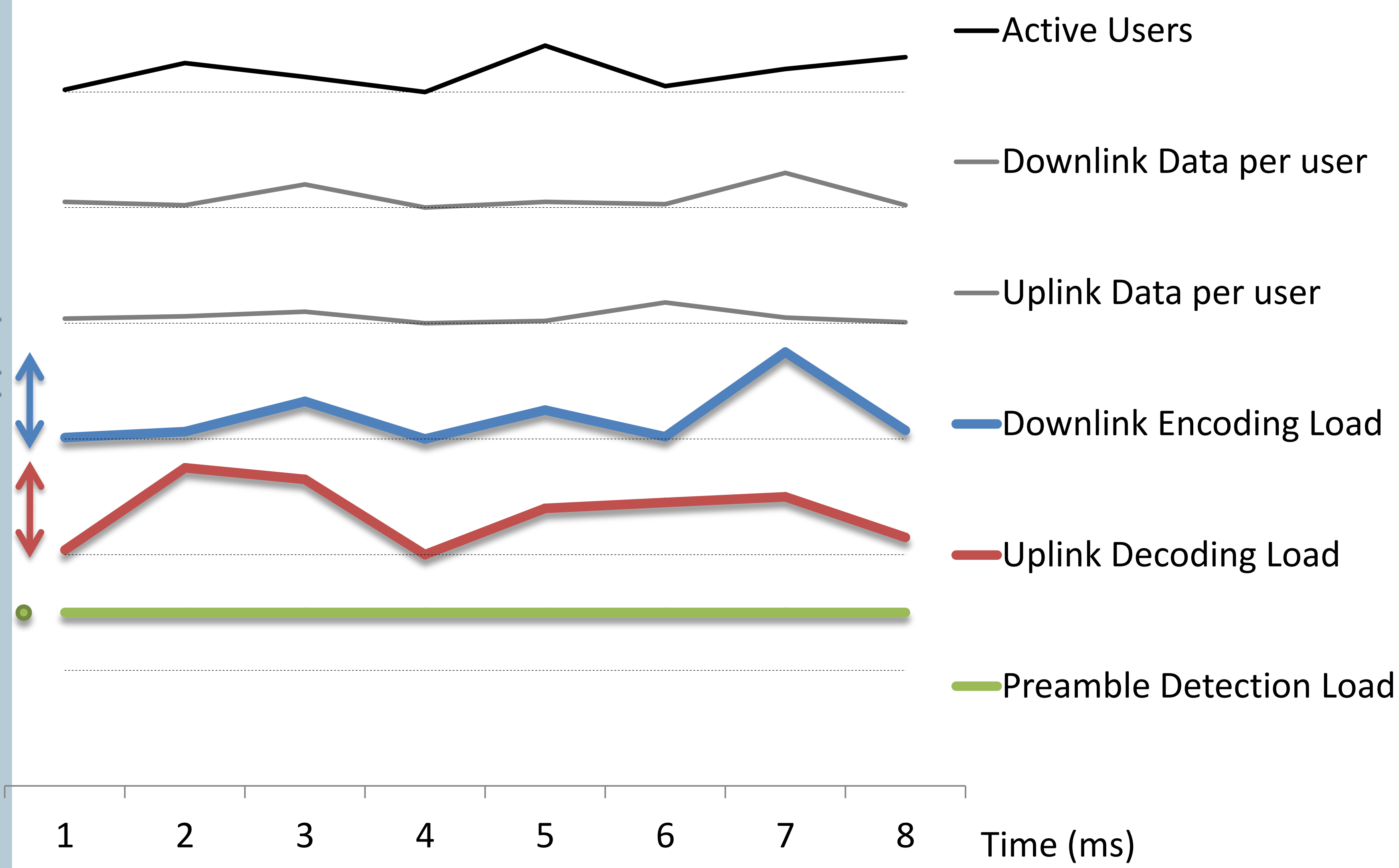


LTE eNodeB DSP Programming



Institut National des Sciences Appliquées

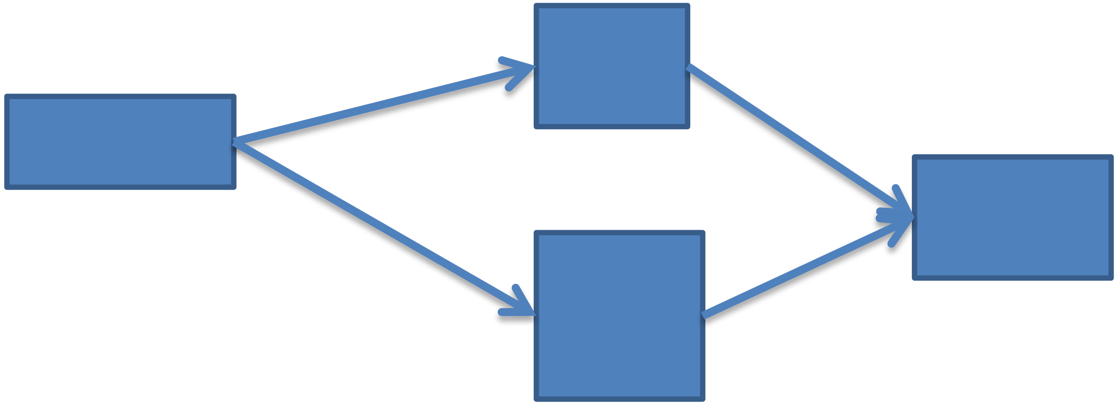
Static versus Variable Algorithms



Conclusion from Applications Part

- **Applications are complex!**
  - A designer should not need to be an expert in both application and architecture
  - Legacy code reuse between systems is absolutely needed
    - When a programmer has generated a functional efficient piece of code, he does want to reuse it
  - A designer should not need to tweak his code for his target architecture

- **Streaming applications are naturally broken down into dataflow actors**
  - When we analyze an application, it is natural to use dataflow graphs



Languages and Models of Computation (MoCs) for Application Description

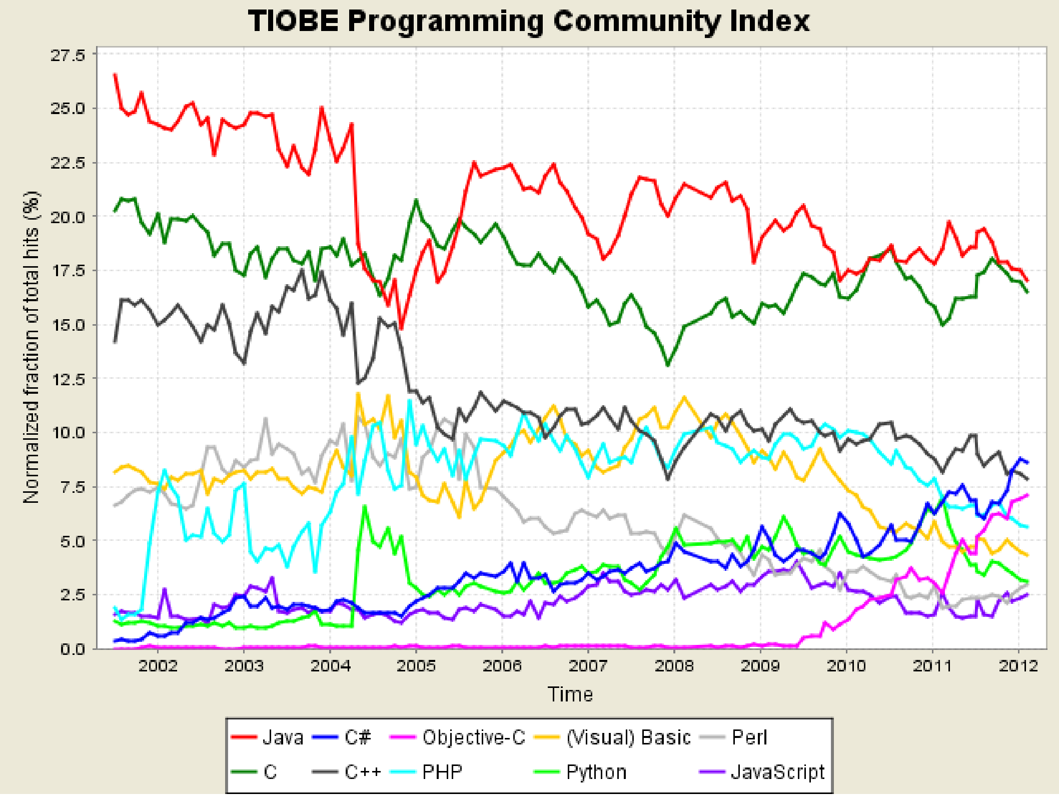
- Languages and MoCs
- Dataflow MoCs and Languages

  Focus on PiMM and πSDF

  Focus on CAL
- Practical Work Setup

Languages and Models of Computation

- **Among the 10 most used languages, all 10 are imperative, 7 are object-oriented. They share semantics.**
- **Other semantics exist. A MoC specifies semantics independently from a language syntax**



Semantics and Syntax

- UML implements object-oriented semantics
- C++ implements object-oriented semantics
- They share semantics but not syntax

Models of Computation for DSP

- **Useful for**

  Specification (especially for standards (cf. MPEG RVC))

  Simulation (performance metrics and constraints checking)

  Execution (functional description)

- **Families of MoCs :**

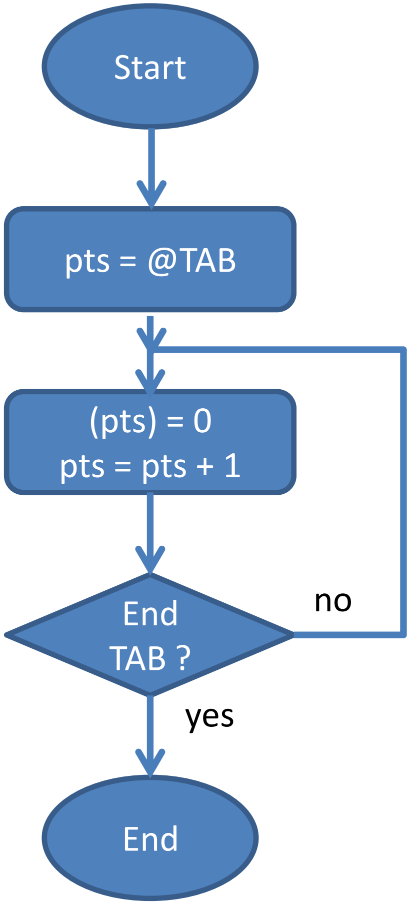
- **Finite State Machine MoCs**

  MoC of imperative languages (C/C++/Java…)

  States and transitions based on conditions

  Computation is executed on transitions

  Representing the behavior of a Turing machine



Institut National des Sciences Appliquées

Models of Computation for DSP

- **Discrete Event MoCs**
  - Modules react to events by producing events
  - Events tagged in time, i.e. the time at which events are consumed and produced is essential and is used to model system behavior
  - Modules share a clock
  - Used to model HDL behavior

- **Functional MoCs**
  - No state, a program returns the result of composed mathematical functions: result = f   g   h(inputs)
  - Based on lambda calculus
  - Haskell, Caml, Scheme, XSLT
  - Functional languages are examples of declarative languages

Institut National des Sciences Appliquées

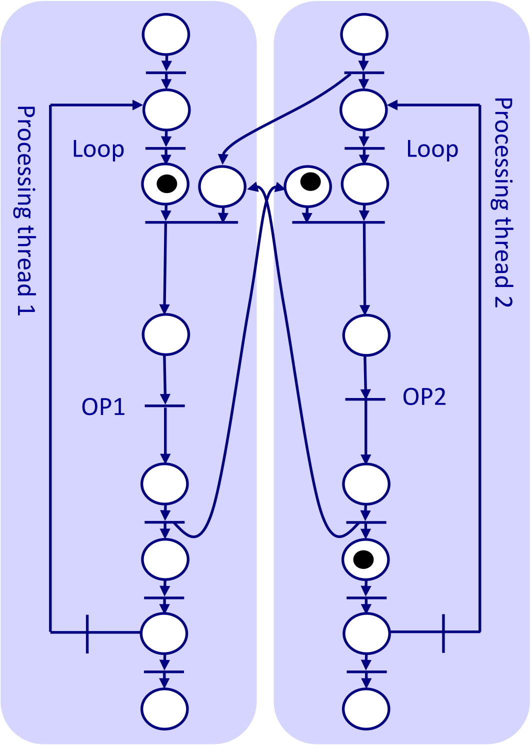
Models of Computation for DSP

- **Petri Nets**

  Close to state machines but

  able to represent parallelism

  Operations are done on transitions



- **Synchronous MoCs**

  Like in Discrete Events, modules react to events

  by producing new events

  Contrary to Discrete Events, time is not explicit and

  only the simultaneity of events and causality are important

  Language examples: Signal, Esterel, Lustre…

Models of Computation for DSP

- **Process Network MoCs**

  concurrent and independent modules (processes) communicate ordered tokens (data quanta) through First-InFirst-Out (FIFO) channels

  Include dataflow process networks

  Untimed models: the time is abstracted

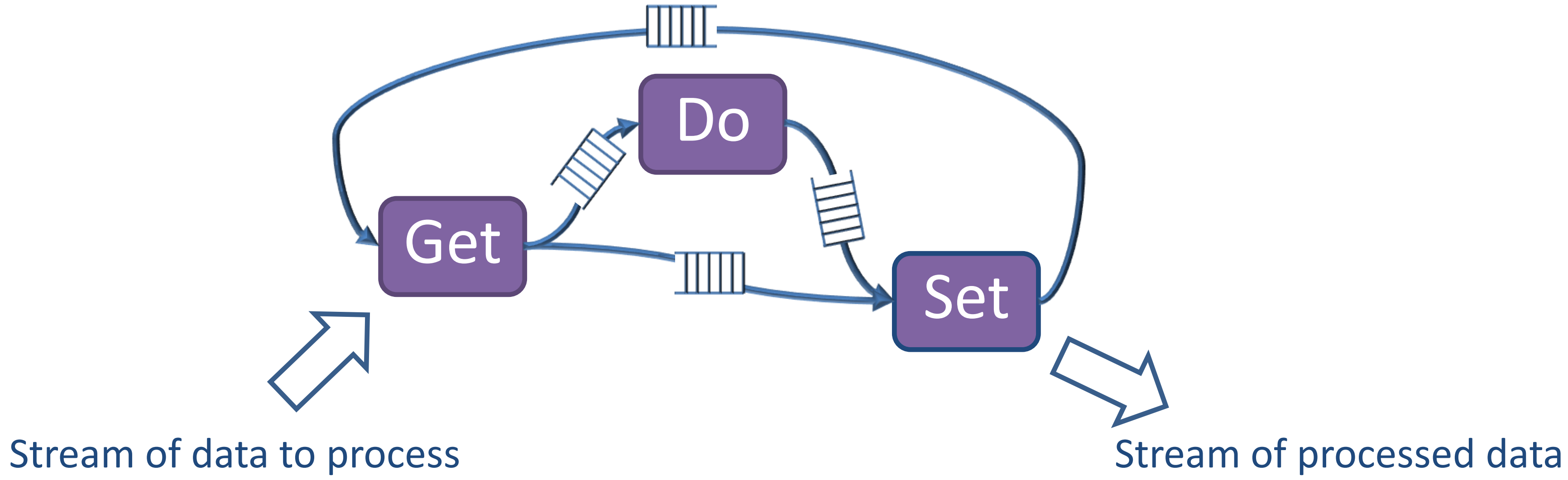
  →What we naturally used to describe MPEG HEVC and 3GPP LTE processing

  …

- **Any syntax can be used to express these semantics**

Kahn Process Networks

- **Computation is done by processes (= Actors)**
- **Actors communicate only though data infinite FIFOs**
  - Actors do not share a state and have no internal state
- **FIFO reading is blocked until a data arrives**

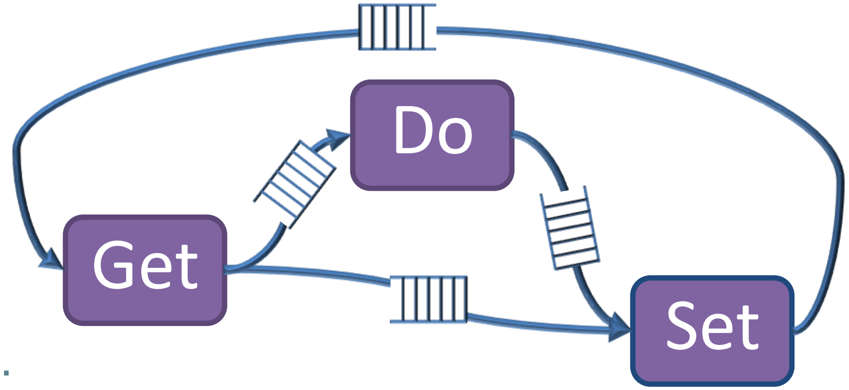


Dataflow Process Networks

- **DPN is a special case of Kahn Process Networks defining:**

  Firing rules: conditions on which an actor is ready to execute

  Actor « invocation » (=« firing »)



Example of firing rule for Set:

-Set consumes 3 tokens coming from Do and fires an action Set1, producing 1 token

-Set consumes 1 token on Get and one token on Do and fires action Set2 producing 2 tokens

Institut National des Sciences Appliquées

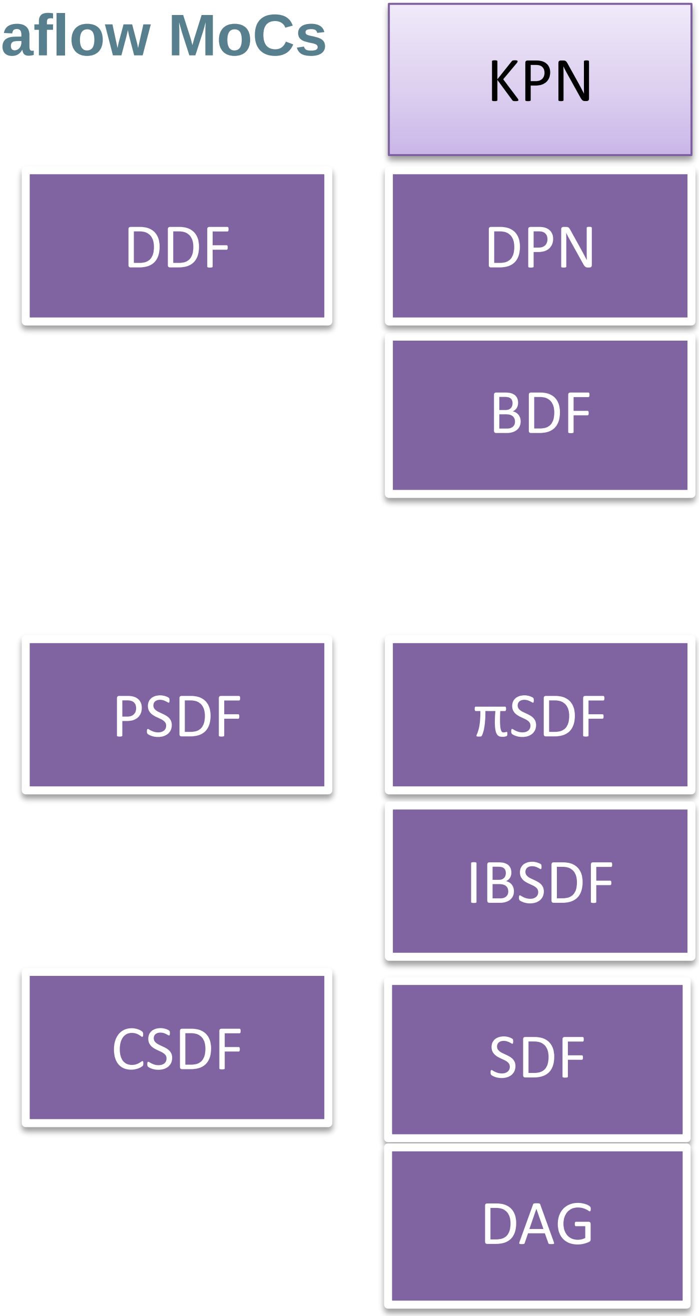
There Exist Many Dataflow MoCs

- **Differences**

  Is the number of exchanged tokens fixed/variable?
  Is it even specified?
  Does it depend of parameters?
  Is there an external control flow?
  Are there actor states

- **Devil is in the details**

  Dynamic Dataflow (DDF) is not a DPN because it can « peek » FIFOs (look inside The FIFO without popping the token)



KPN
DDF
DPN
BDF
PSDF
πSDF
IBSDF
CSDF
SDF
DAG

There Exist Many Dataflow MoCs

- **Expressiveness**

  How many different behavior types can the model specify?

  How « optimized » is the specification?

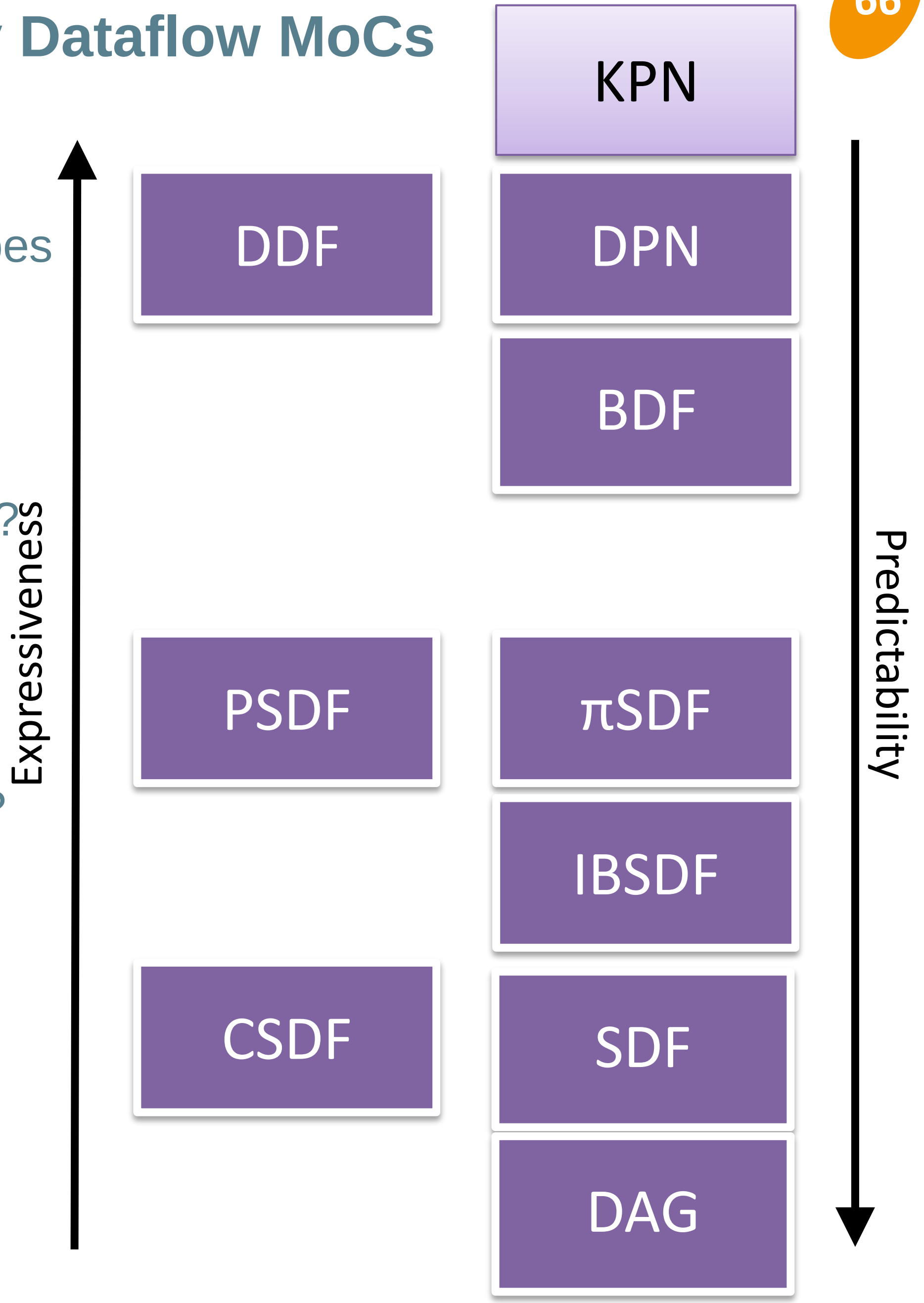
  How concise is the specification?

- **Predictability**

  Can the behavior (production And consumption) be predicted?

  at compile-time

  in advance at runtime



There Exist Many Dataflow MoCs

- **The dataflow MoC defines a coordination language**
- **Actors are implemented by a host language**
  - Any host language can be used
  - As long as the host language implements the firing rules
  - We can combine and make communicate
    - IPs coded in VHDL
    - High-level software actors written in Java
    - Low-level software actors written in C

Which MoC should we use to program Multicore DSPs?

- **Small grain, Locally to a core**

  Imperative languages such as C/C++ have shown their capacity to use cores efficiently, including with low-level parallelism (SIMD, VLIW…)

  Finite state machine MoC

- **Coarse grain, to combine the cores**

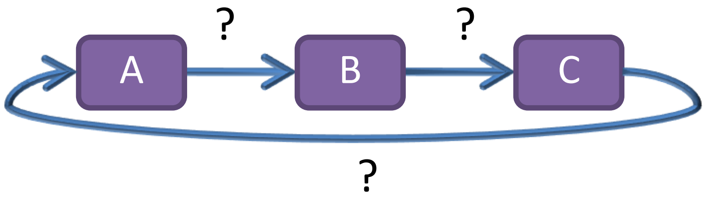
  Dataflow MoCs are good candidates because they decouple actors

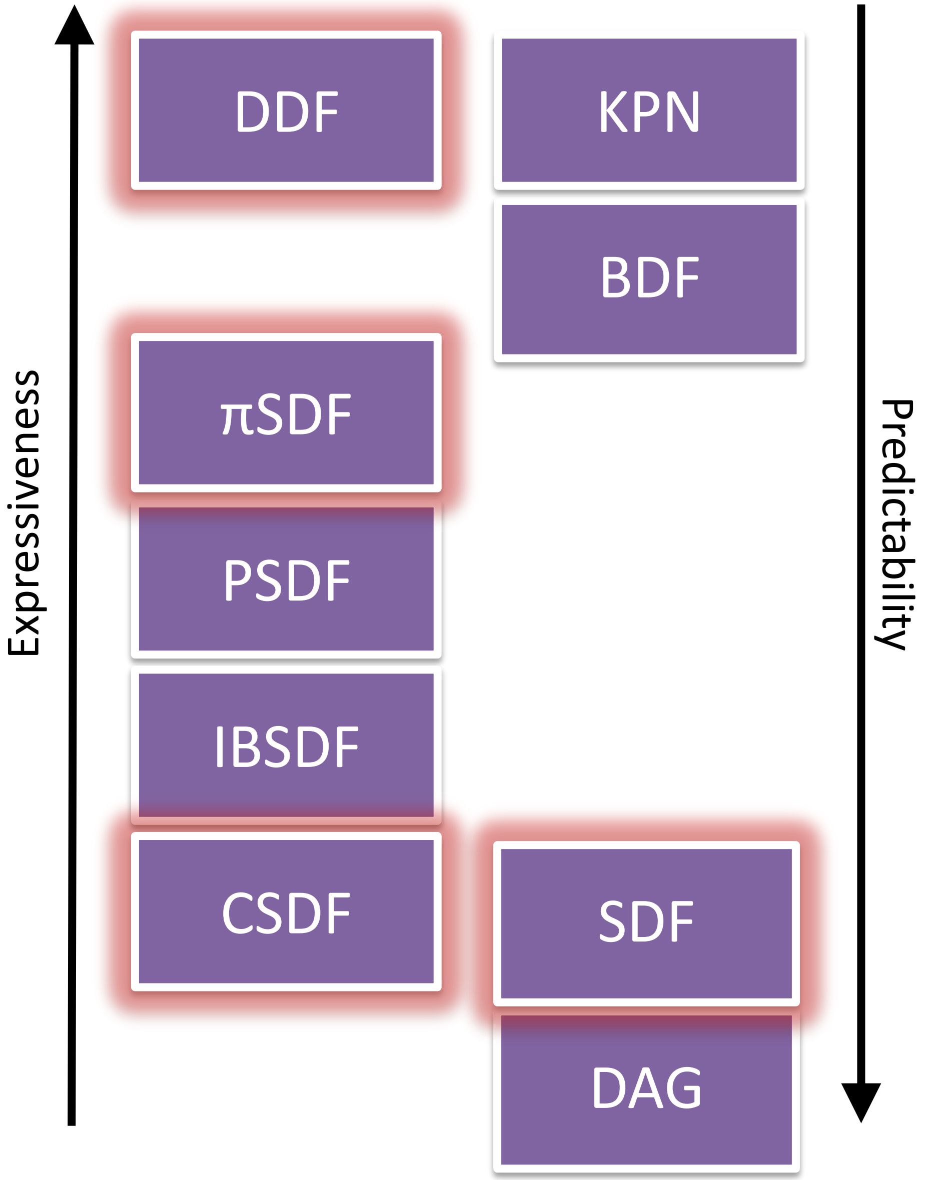
  All possible communications between actors are specified, so they can be used to organize data flows between cores

Which Dataflow Model for a Given Application ?

3GPP LTE
For example:

- Preamble Detection
- Uplink Decoding
- Downlink Encoding





Synchronous Dataflow

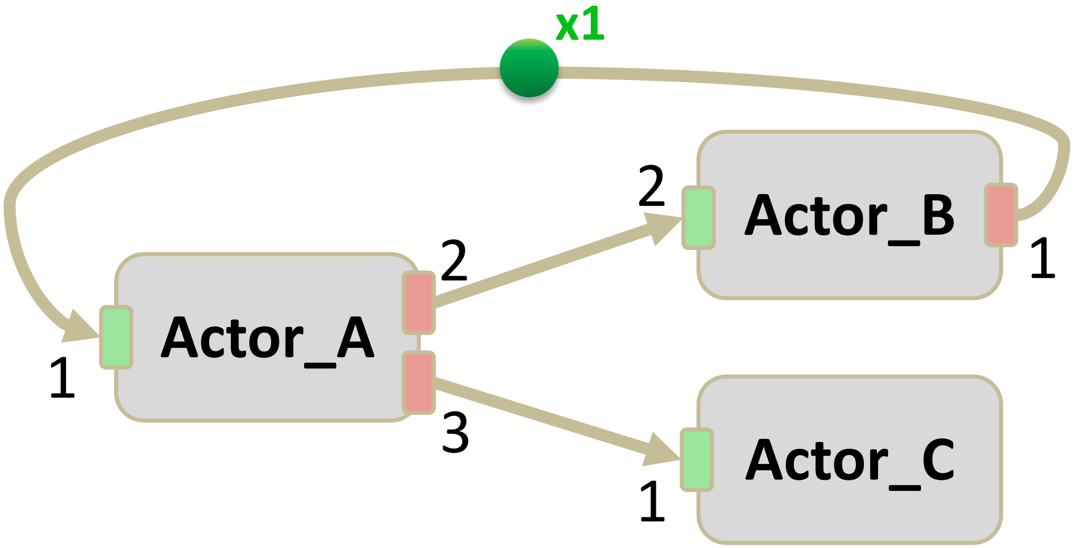
- **Actors and Data Ports**
- **FIFO Queues and Delays**

DDF

πSDF

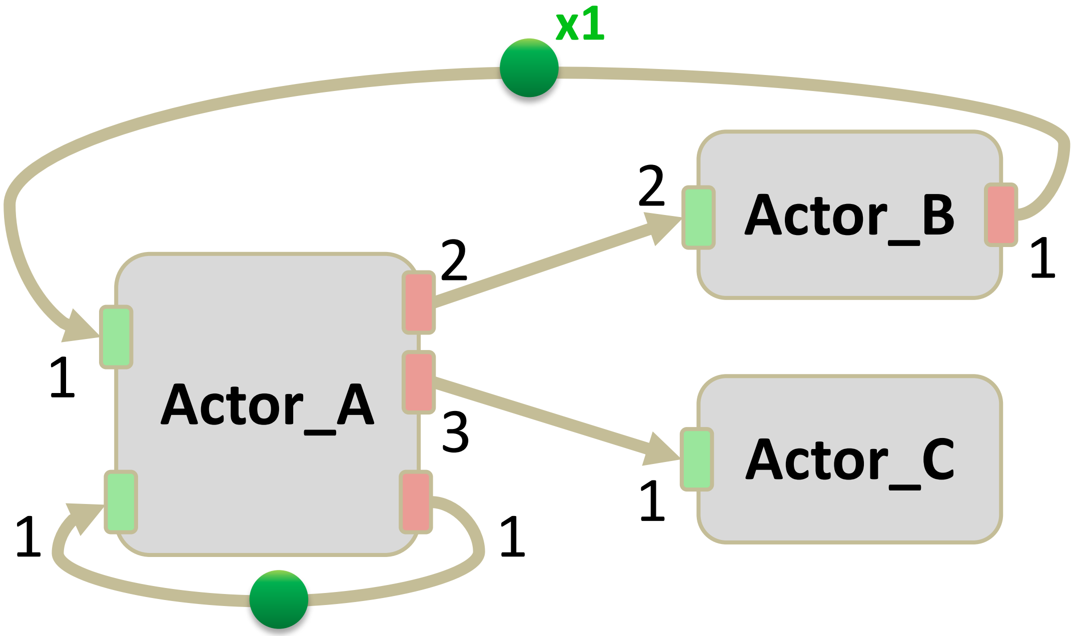
CSDF

SDF



E. Lee and D. Messerschmitt, "*Synchronous data flow*", Proceedings of the IEEE, 1987.

Synchronous Dataflow and Actor State



DDF

πSDF

CSDF

SDF

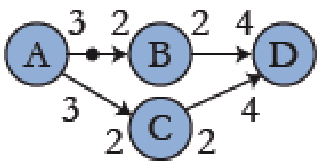
E. Lee and D. Messerschmitt, "*Synchronous data flow*", Proceedings of the IEEE, 1987.

SDF Predictability

- **By resolving the topology equation, we can check consistency**
- **By verifying that enough initial tokens have been set, we can check schedulability**

  Ability to come back to the initial FIFO states

SDF Graph



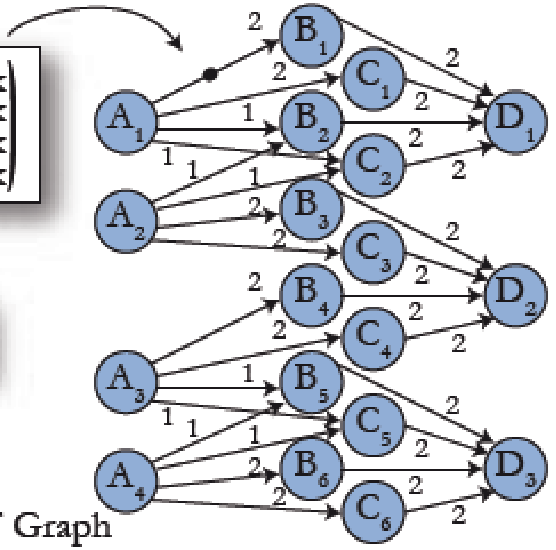
1) Topology Matrix of rank $nb_{actors} - 1$ => consistency

$$\begin{pmatrix} 3 & -2 & 0 & 0 \\ 3 & 0 & -2 & 0 \\ 0 & 2 & 0 & -4 \\ 0 & 0 & 2 & -4 \end{pmatrix} . q = 0 \Longleftrightarrow \begin{pmatrix} 3 & -2 & 0 & 0 \\ 0 & 2 & -2 & 0 \\ 0 & 0 & 2 & -4 \\ 0 & 0 & 0 & 0 \end{pmatrix} . q = 0 \Longleftrightarrow q = \begin{pmatrix} 4.x \\ 6.x \\ 6.x \\ 3.x \end{pmatrix}$$

2) A valid schedule, respecting initial tokens exists.

(4A)(6B)(6C)(3D)=> The graph is back to original state

=> The SDF graph is schedulable



Equivalent single rate SDF Graph

Institut National des Sciences Appliquées

Graph Scheduling and Transformation

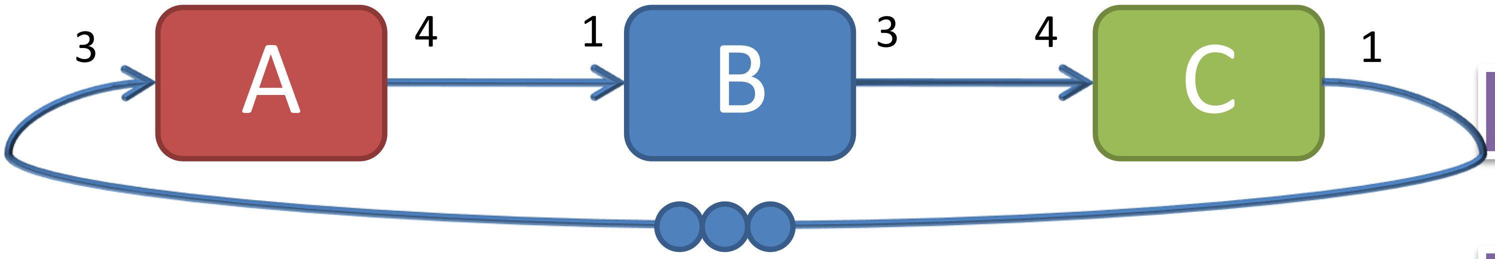
DDF

πSDF

CSDF

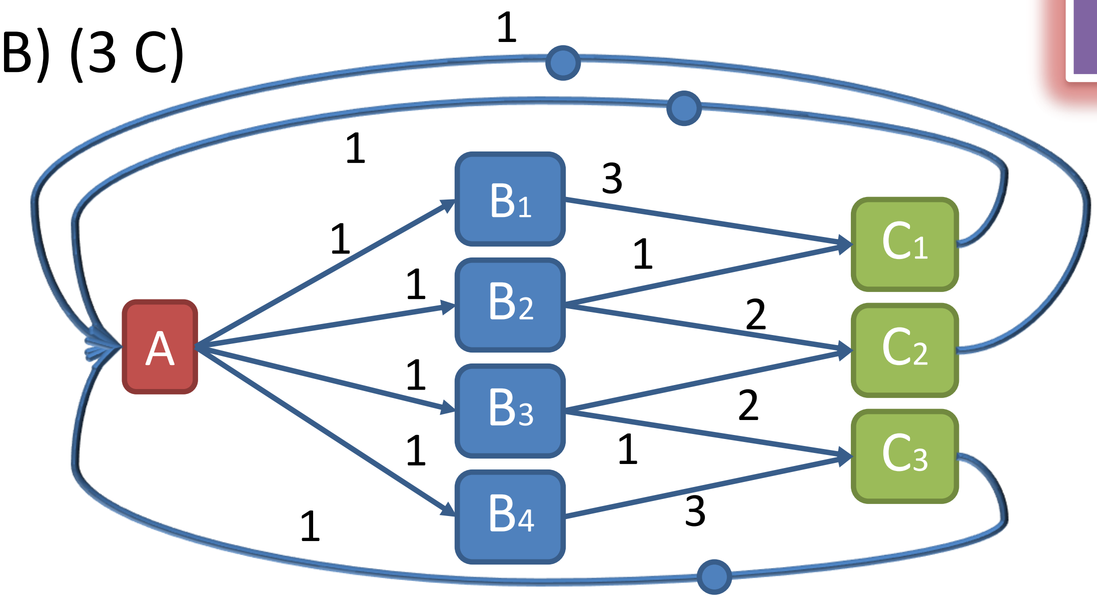
SDF

Synchronous Dataflow (SDF) graph:



Schedule: A (4 B) (3 C)

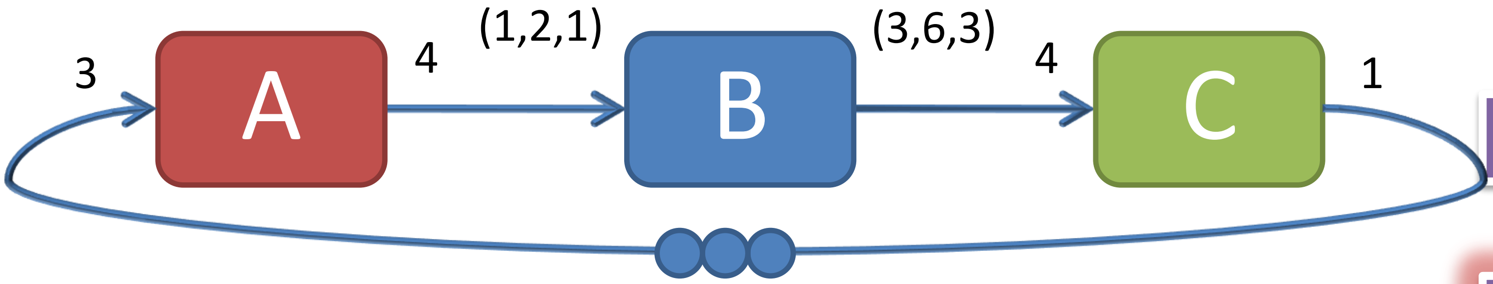
Single rate graph



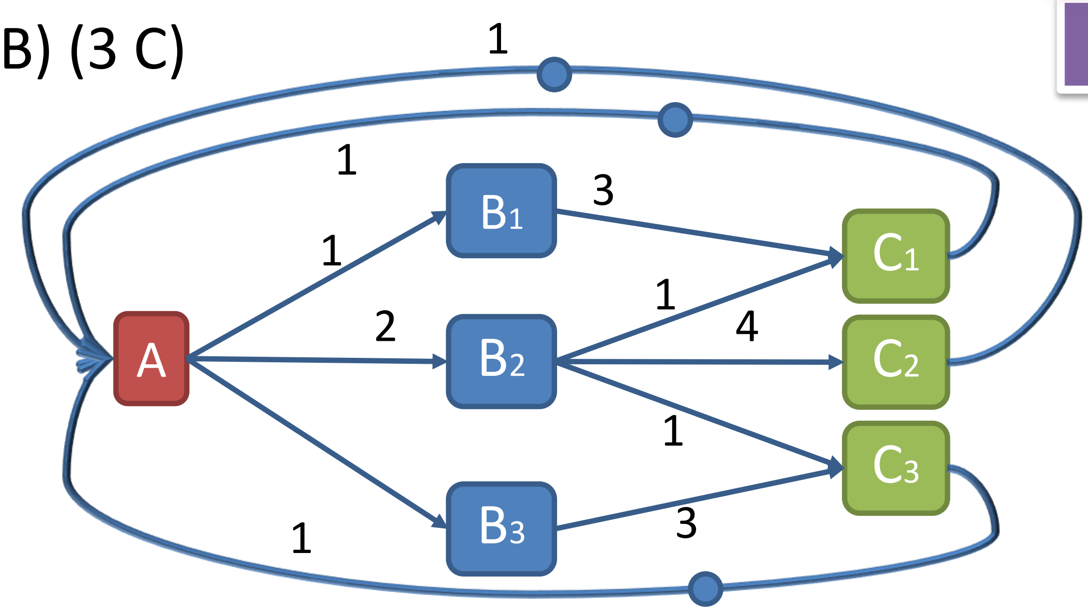
Graph Scheduling and Transformation

- DDF
- πSDF
- CSDF
- SDF

Cyclo Static Dataflow (CSDF) graph:



Schedule: A (3 B) (3 C)



Single rate
Graph = SDF!!

PiMM and πSDF

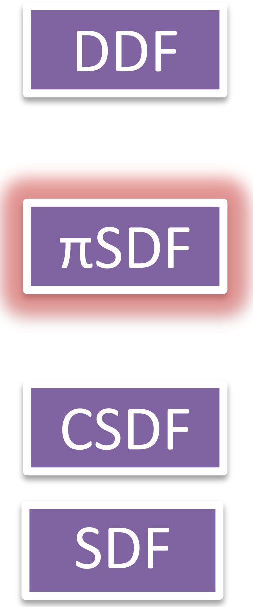
- **Meta-model developped at IETR**
- **in collaboration with UMD and TI**

- **Targeted Dataflow Model of Computation**
- **becomes:**

  Hierarchical & Compositional
  Statically parameterizable
  Dynamically reconfigurable

- **PiMM fosters:**

  Predictability
  Memory boundedness
  Parallelism
  Lightweight runtime overhead
  Developer-friendliness



DDF

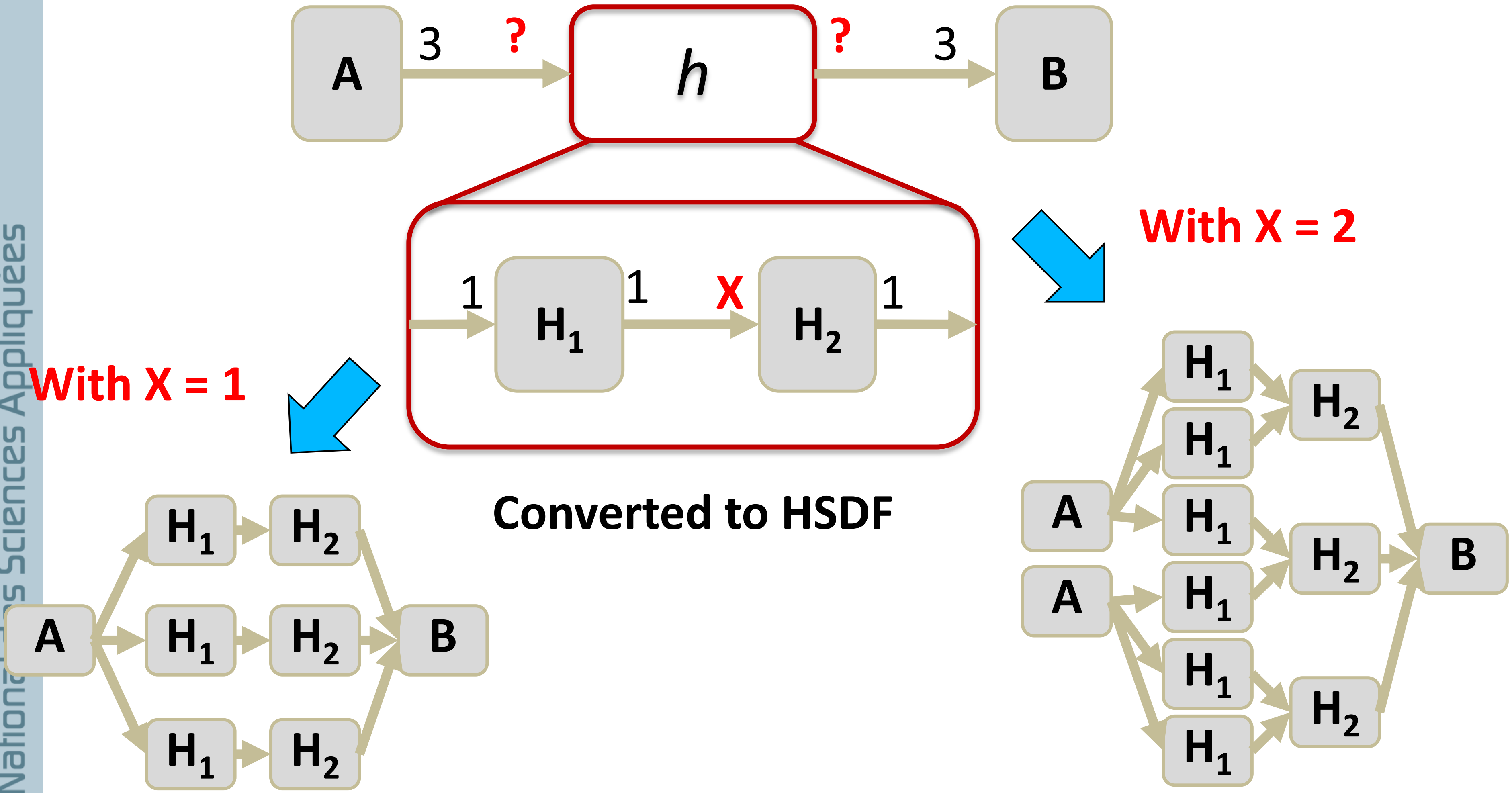
πSDF

CSDF

SDF

A Naive SDF Composition Mechanism

- Hierarchical SDF



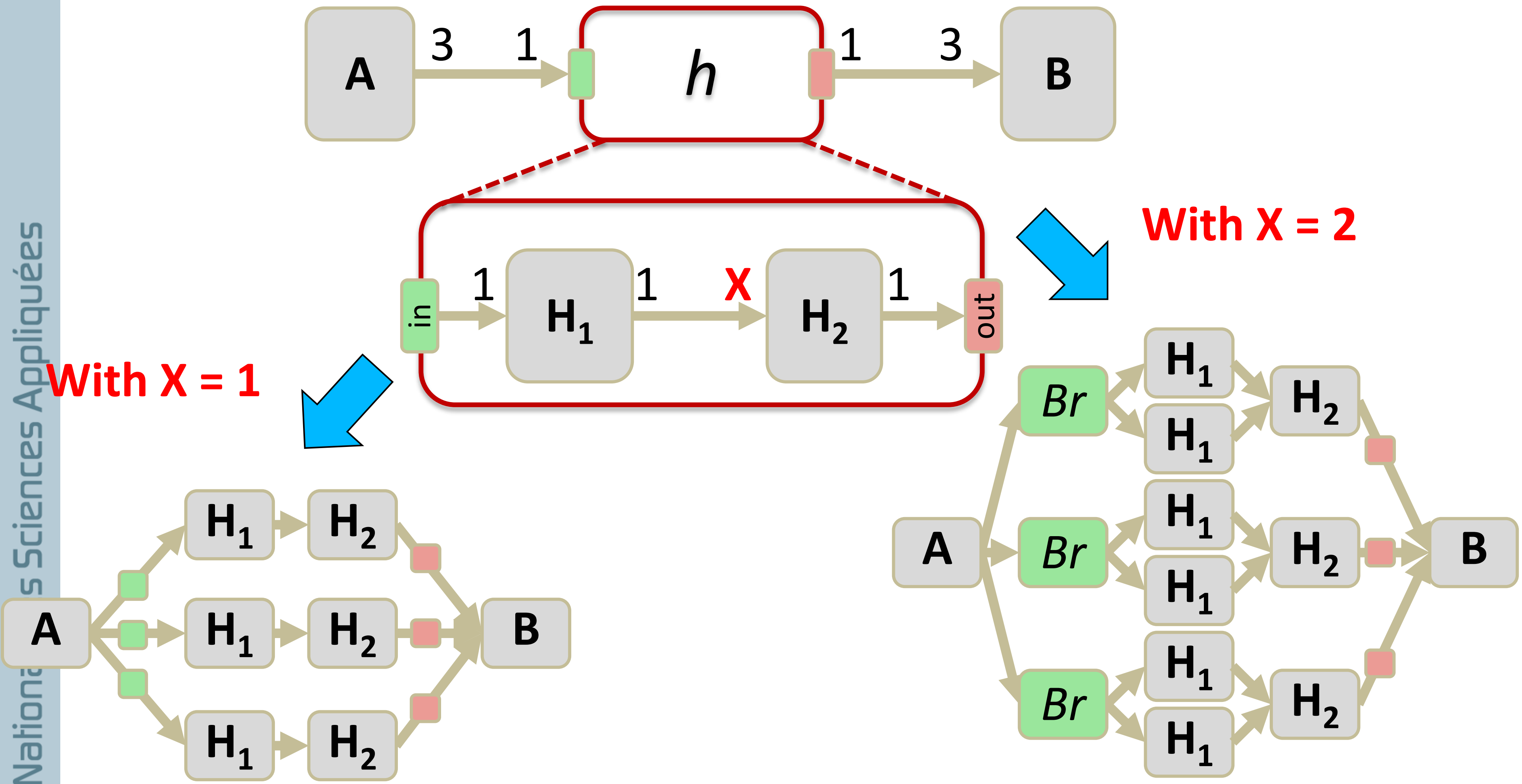
With X = 2

With X = 1

Converted to HSDF

PiMM Compositionality Mechanism

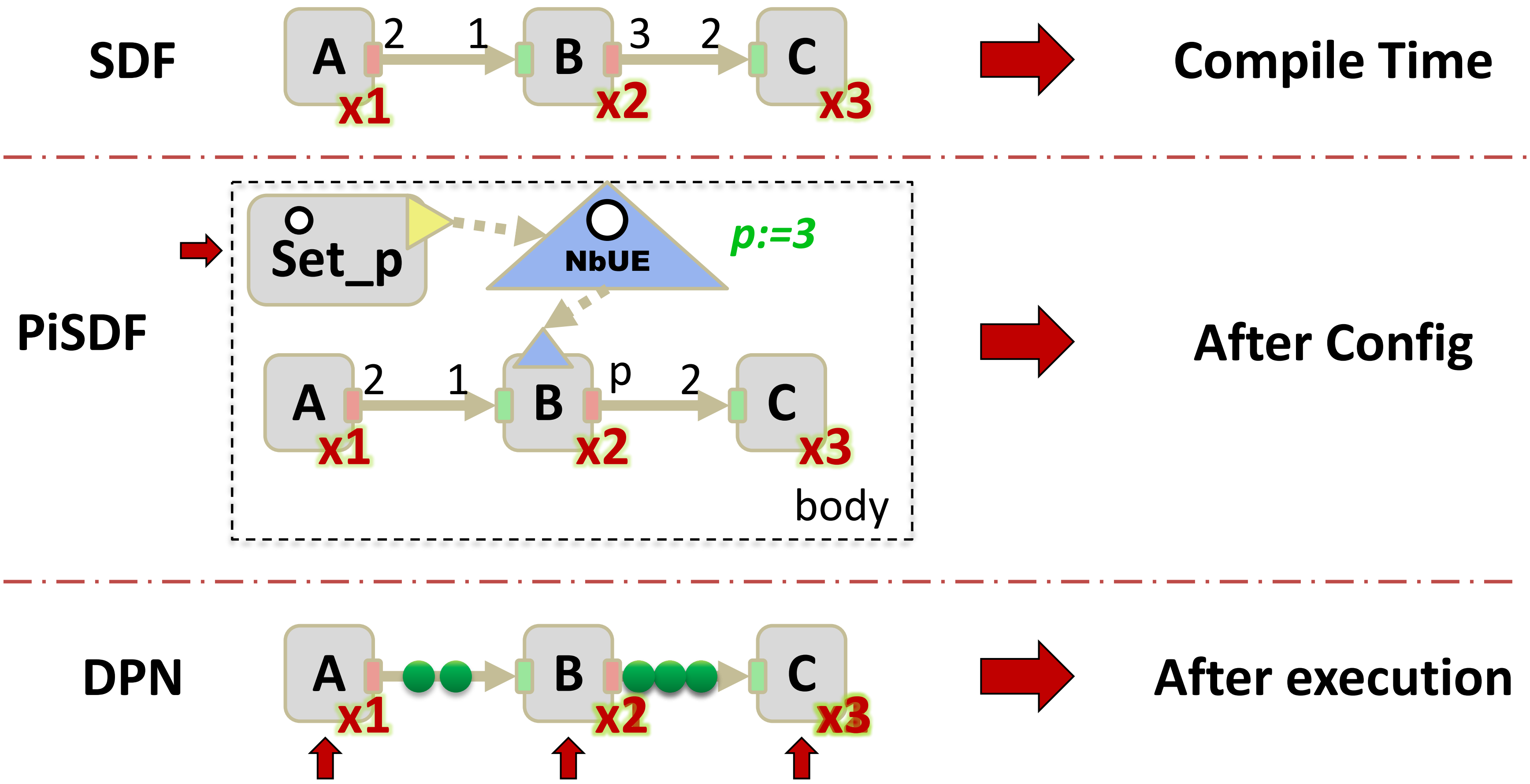
- **Hierarchical Interfaces**



J. Piat, S. Bhattacharyya, and M. Raulet, "*Interface-based hierarchy for synchronous data-flow graphs*" in SiPS Proceedings, 2009.

Trade-off Between Expressiveness and Analyzability

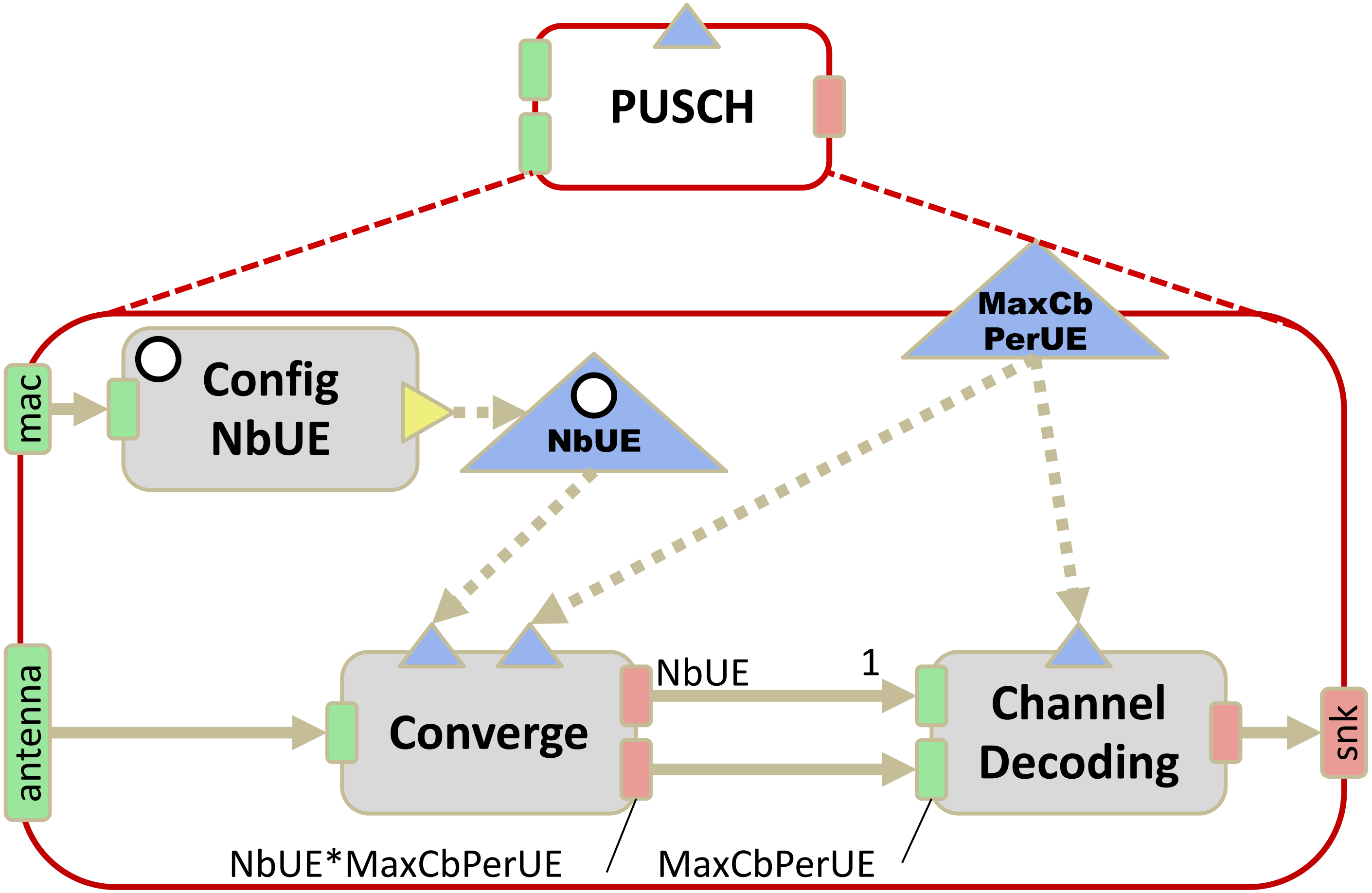
- The more dynamism, the less predictability



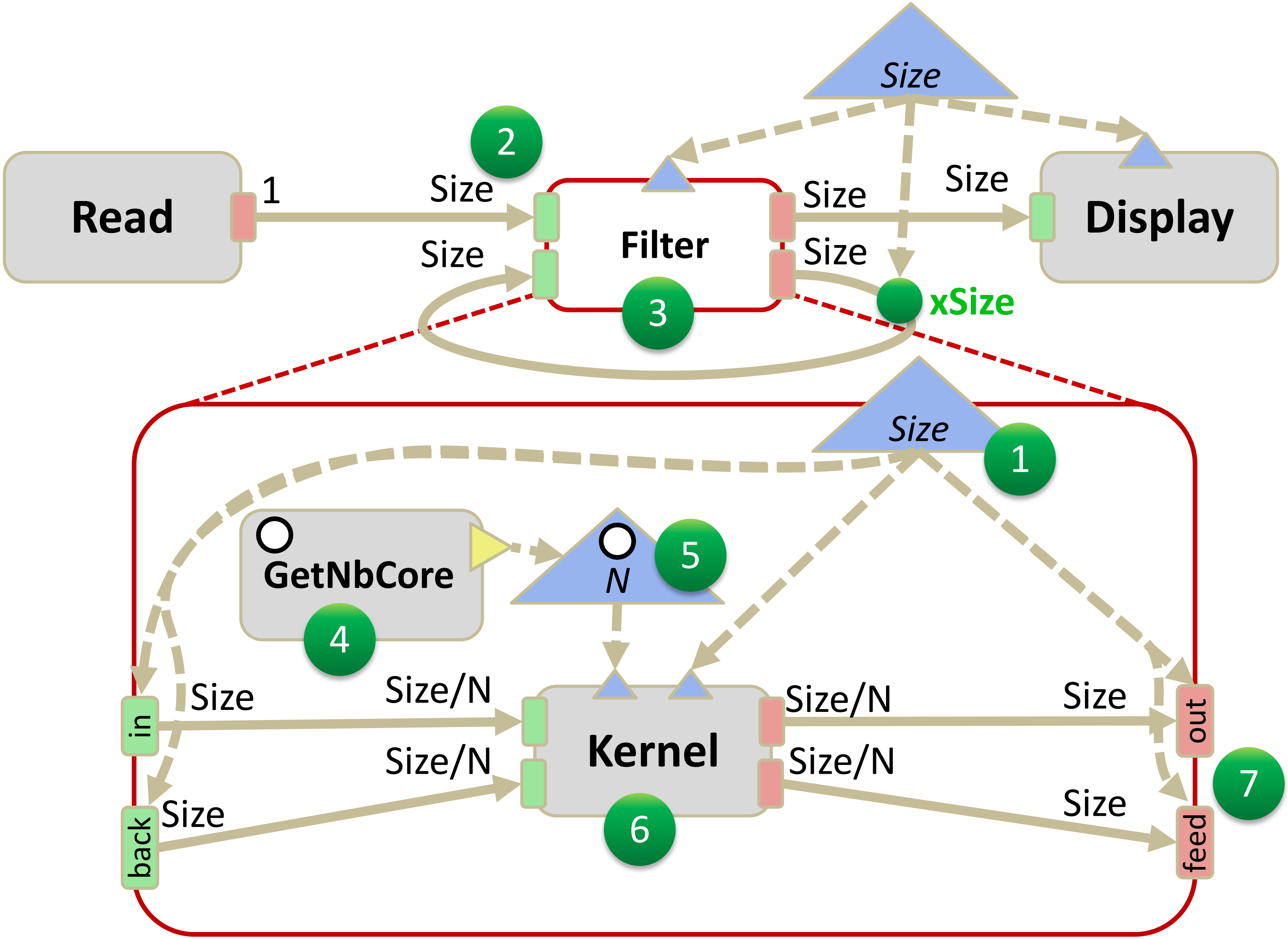
S. Neuendorffer and E. Lee, "*Hierarchical reconfiguration of dataflow models*" in MEMOCODE, 2004.

Configuring Parameters

- **Dynamic Parameters**
- **Configuration Actors**



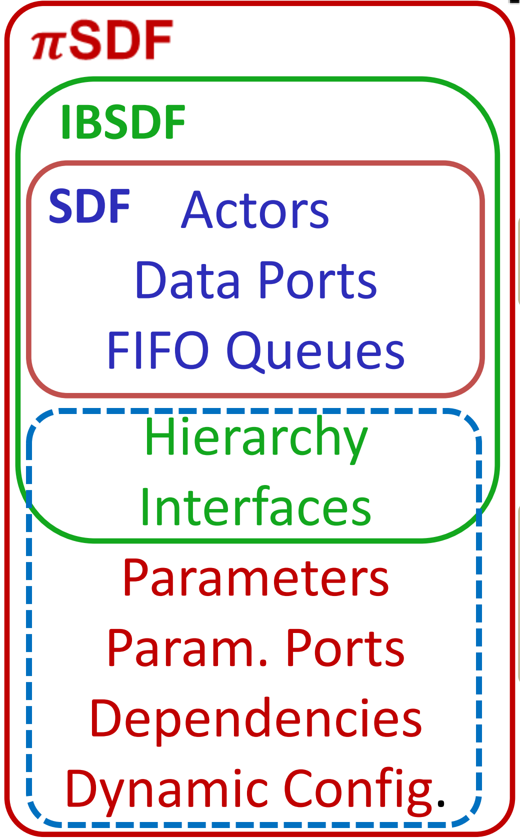
PiMM Operational Semantics



Institut National des Sciences Appliquées

PiMM Operational Semantics

- **PiMM** (Parameterized and Interfaced dataflow Meta-Model)
  - Evolution of IBSDF and PSDF
- πSDF Example:



πSDF

IBSDF

SDF Actors

Data Ports

FIFO Queues

Hierarchy

Interfaces

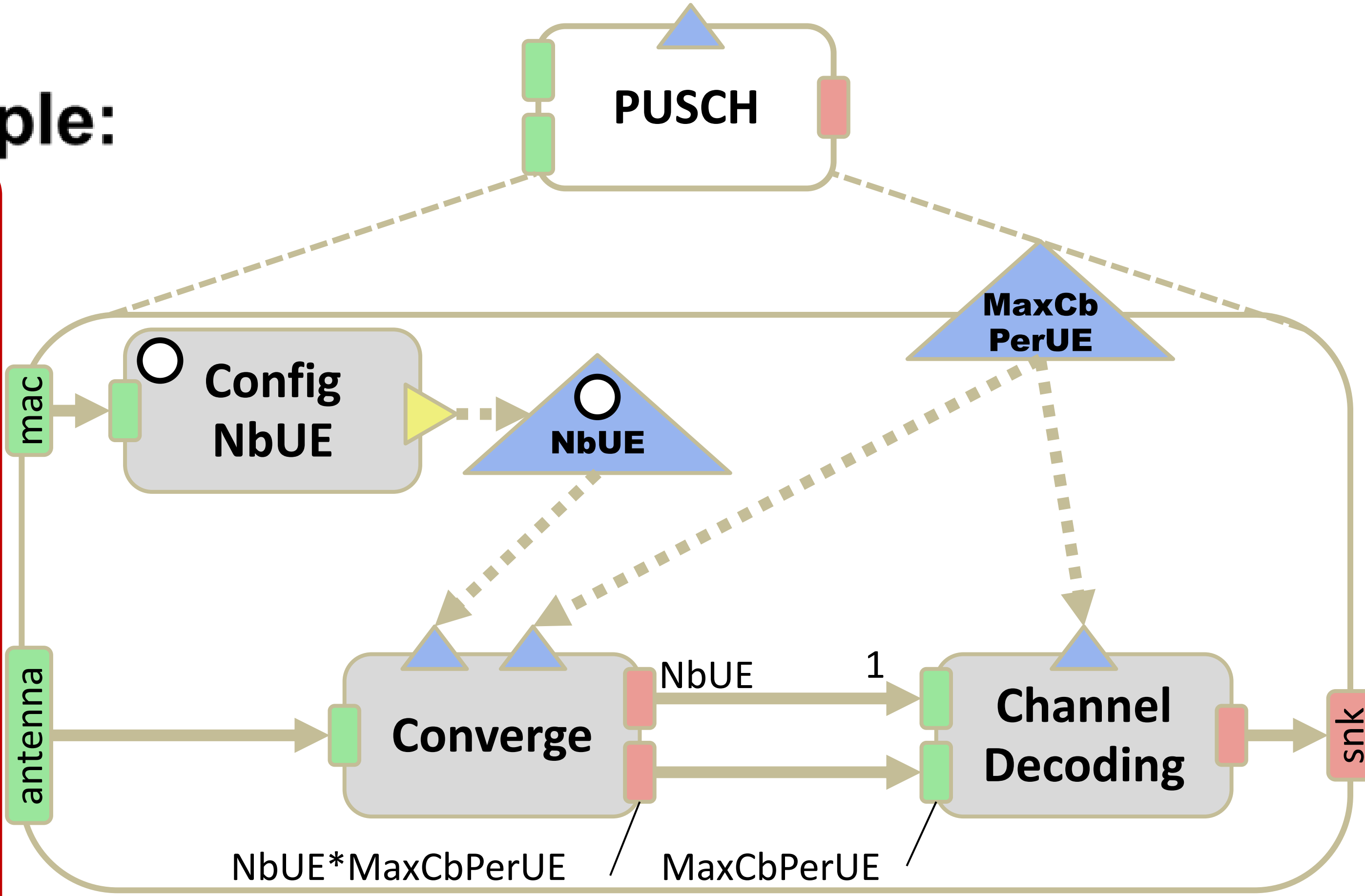
Parameters

Param. Ports

Dependencies

Dynamic Config.

PiMM



CAL and DDF

- **CAL = CAL Actor Language, J. Ecker and J. Janneck**
- **It implements**

  the Dynamic Dataflow (DDF) MoC

  and a host code named CAL specifying firing rules
- **DDF graph gives no information on token flow**

  Firing rules are specified in the CAL language

  consumed tokens, guards, finite state machine, priorities

DDF

πSDF

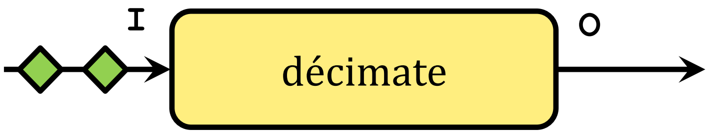
CSDF

SDF

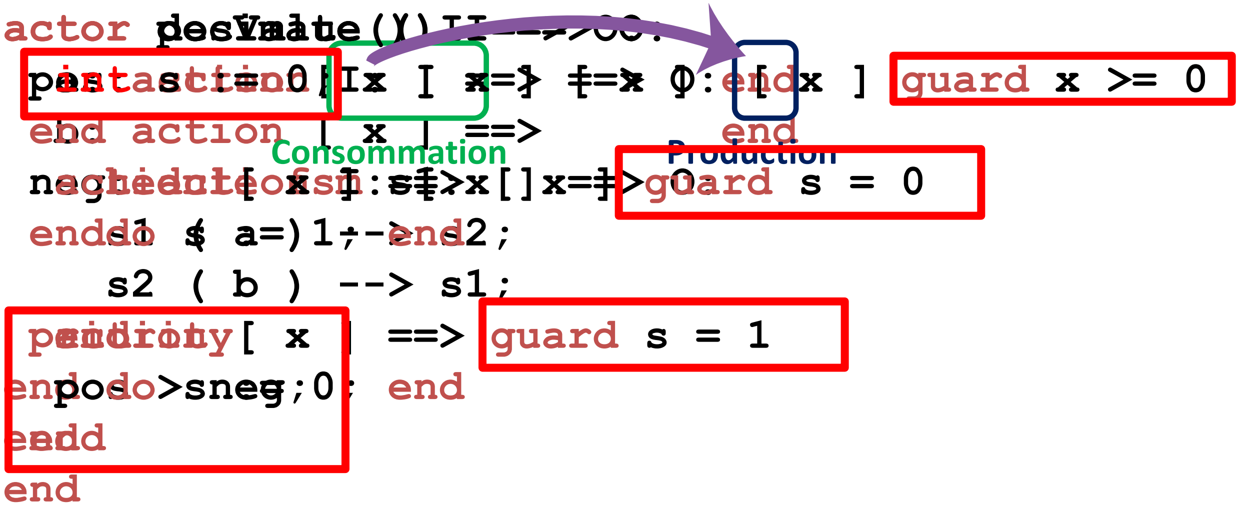
```
actor decimate () I ==> O :
  action I:[ a , b ] ==> O:[ a ] end
end
```

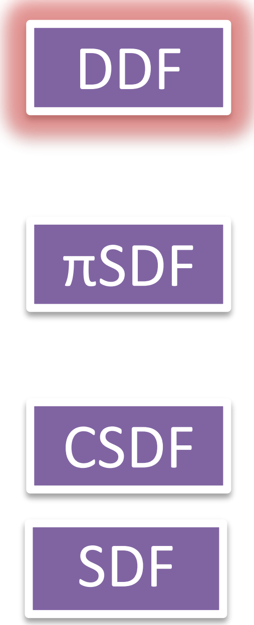
Consommation

Production



CAL and DDF





- **Via classification, a CAL actor can be transformed into SDF, CSDF, and soon πSDF**

  In order to make the model more predictable

- **The Orcc compiler, developed at IETR, is a compiler for CAL**

Institut National des Sciences Appliquées

CAL and DDF

- From CAL, both hardware and software can be generated
- CAL → Preesm is already working for static actors
- CAL → PiSDF is a future theme of research

DDF

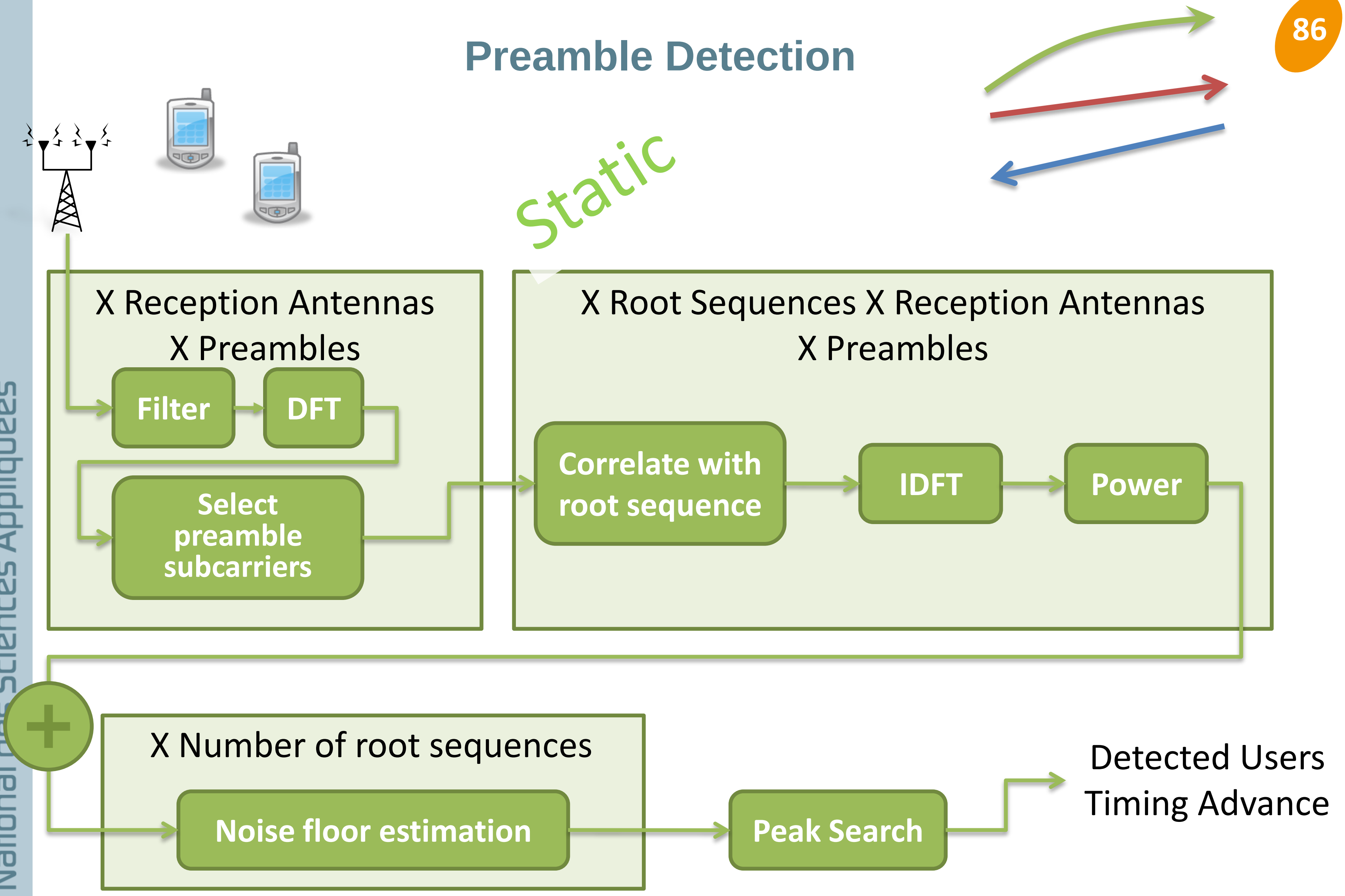
πSDF

CSDF

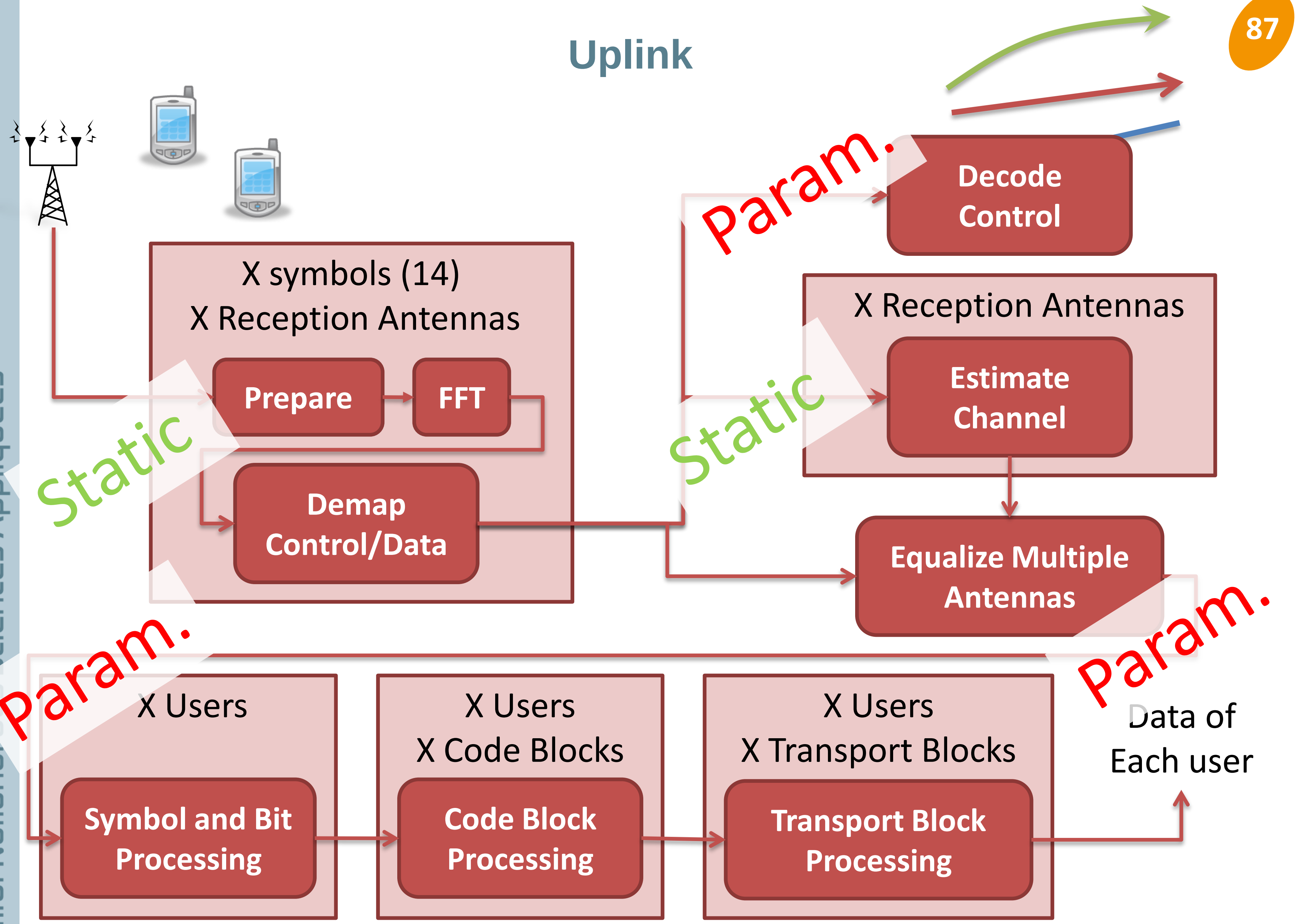
SDF

Example: Describing 3GPP LTE Base Station Physical Layer

Preamble Detection

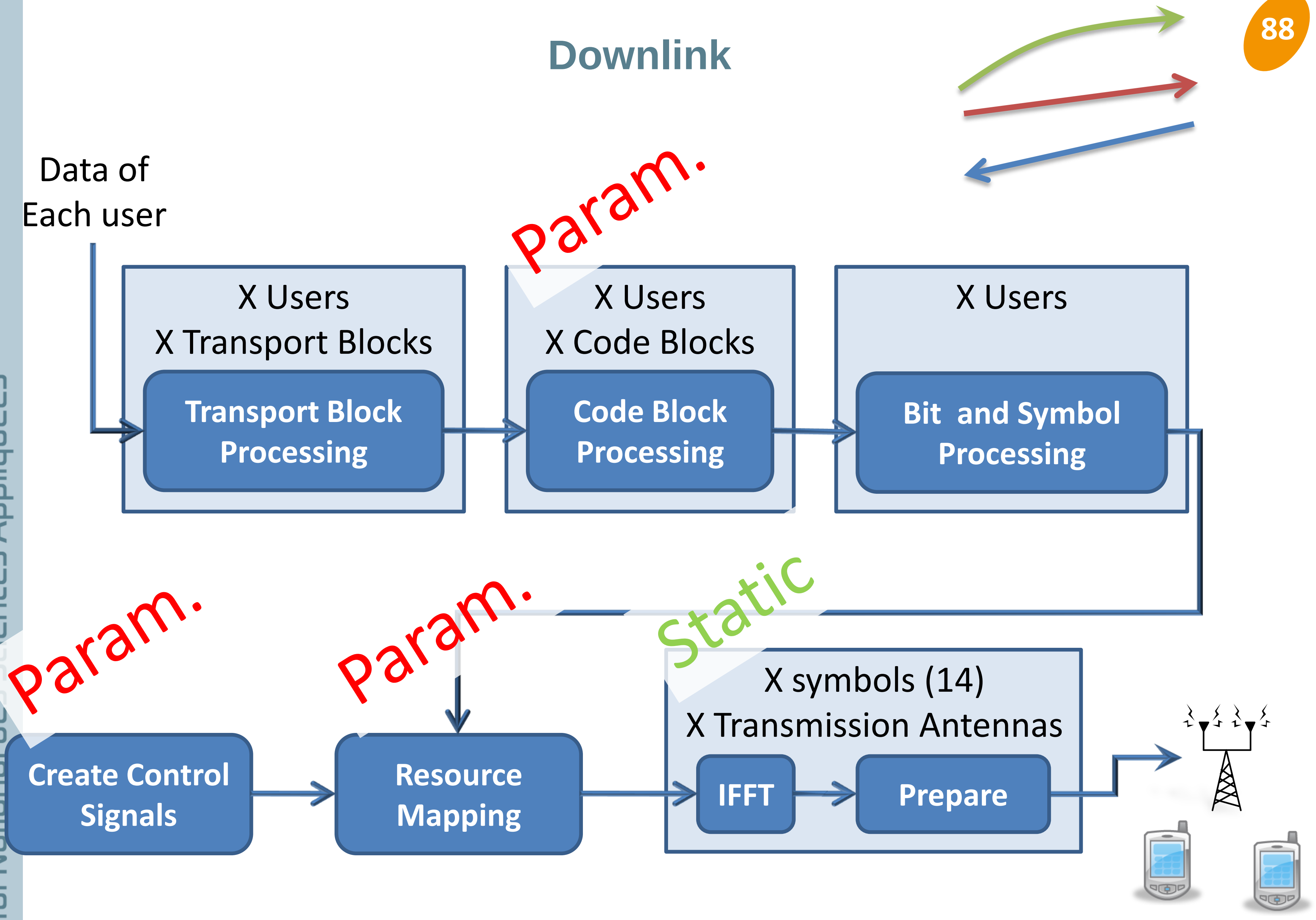


Uplink



Param.
Decode Control
X symbols (14)
X Reception Antennas
Prepare
FFT
Demap Control/Data
Static
X Reception Antennas
Estimate Channel
Static
Equalize Multiple Antennas
Param.
Param.
X Users
Symbol and Bit Processing
X Users
X Code Blocks
Code Block Processing
X Users
X Transport Blocks
Transport Block Processing
Data of Each user

Downlink



Data of Each user

X Users
X Transport Blocks

Transport Block Processing

X Users
X Code Blocks

Code Block Processing

X Users

Bit and Symbol Processing

Param.

Param.

Param.

Static

Create Control Signals

Resource Mapping

X symbols (14)
X Transmission Antennas

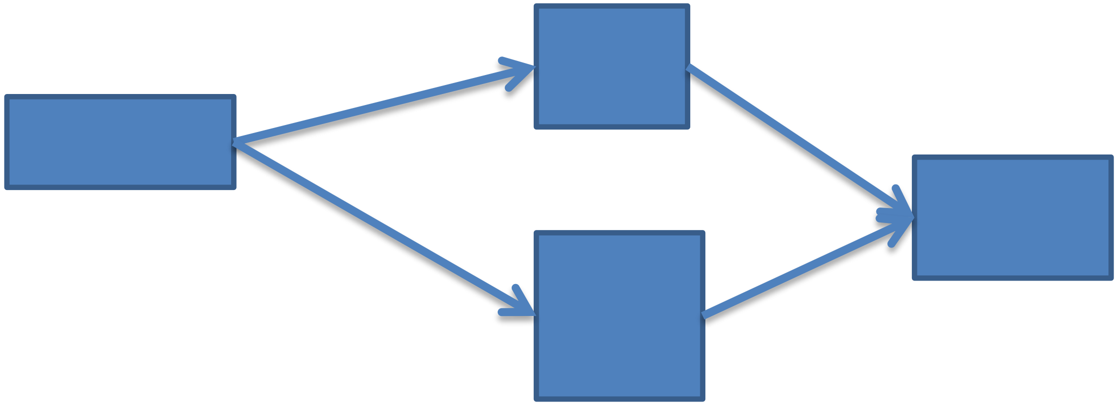
IFFT

Prepare

Conclusion from Models Part

- **Applications are naturally broken down into dataflow actors**

  When we analyze an application, it is natural to use dataflow graphs



  Many models exist with different semantics

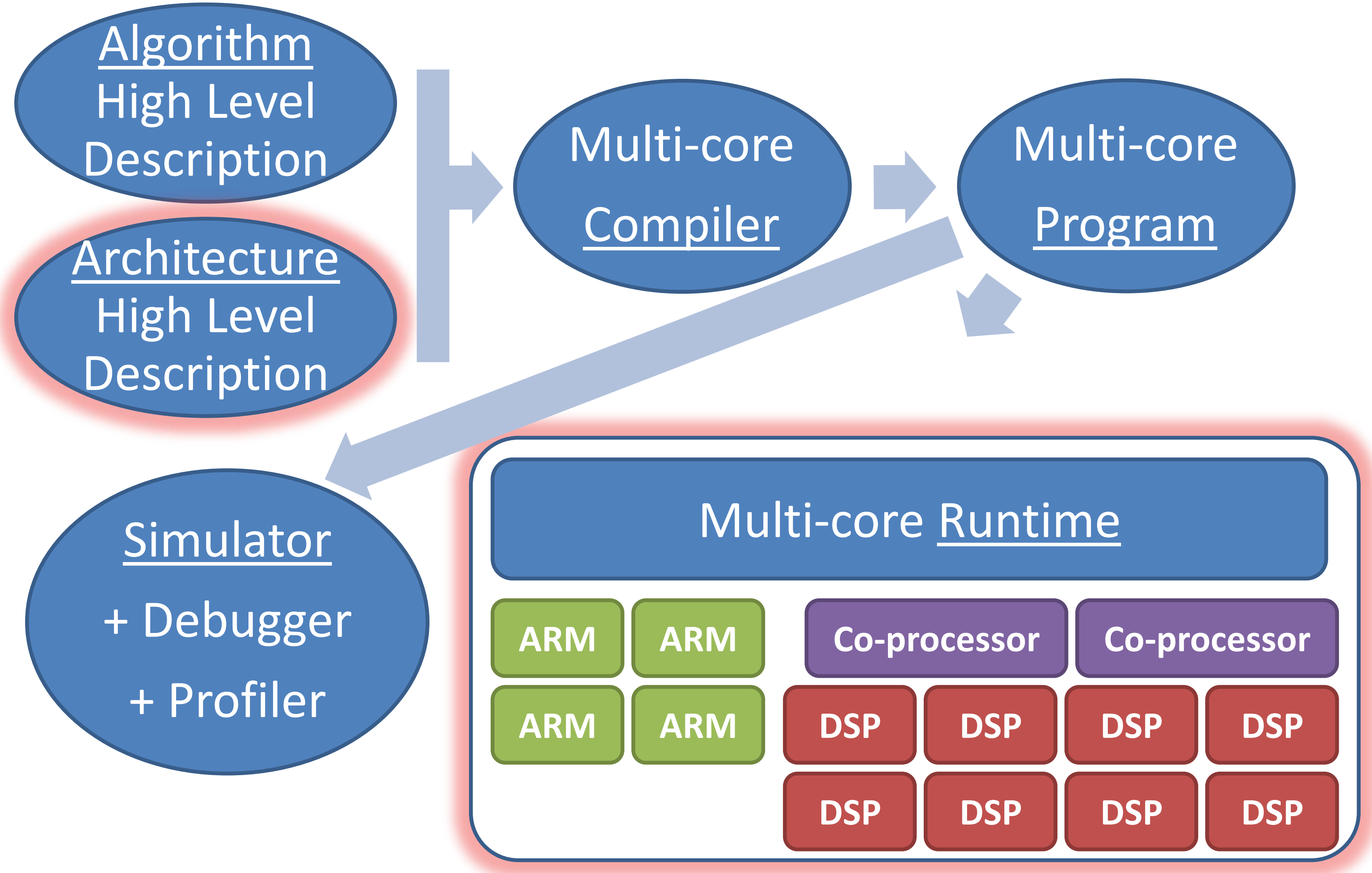
- **Dataflow models express parallelism in algorithms**

- **Syntax != Semantics**

  Semantics matter more than syntax

Institut National des Sciences Appliquées

Multi-core DSP Programming



Algorithm High Level Description

Architecture High Level Description

Multi-core Compiler

Multi-core Program

Simulator + Debugger + Profiler

Multi-core Runtime

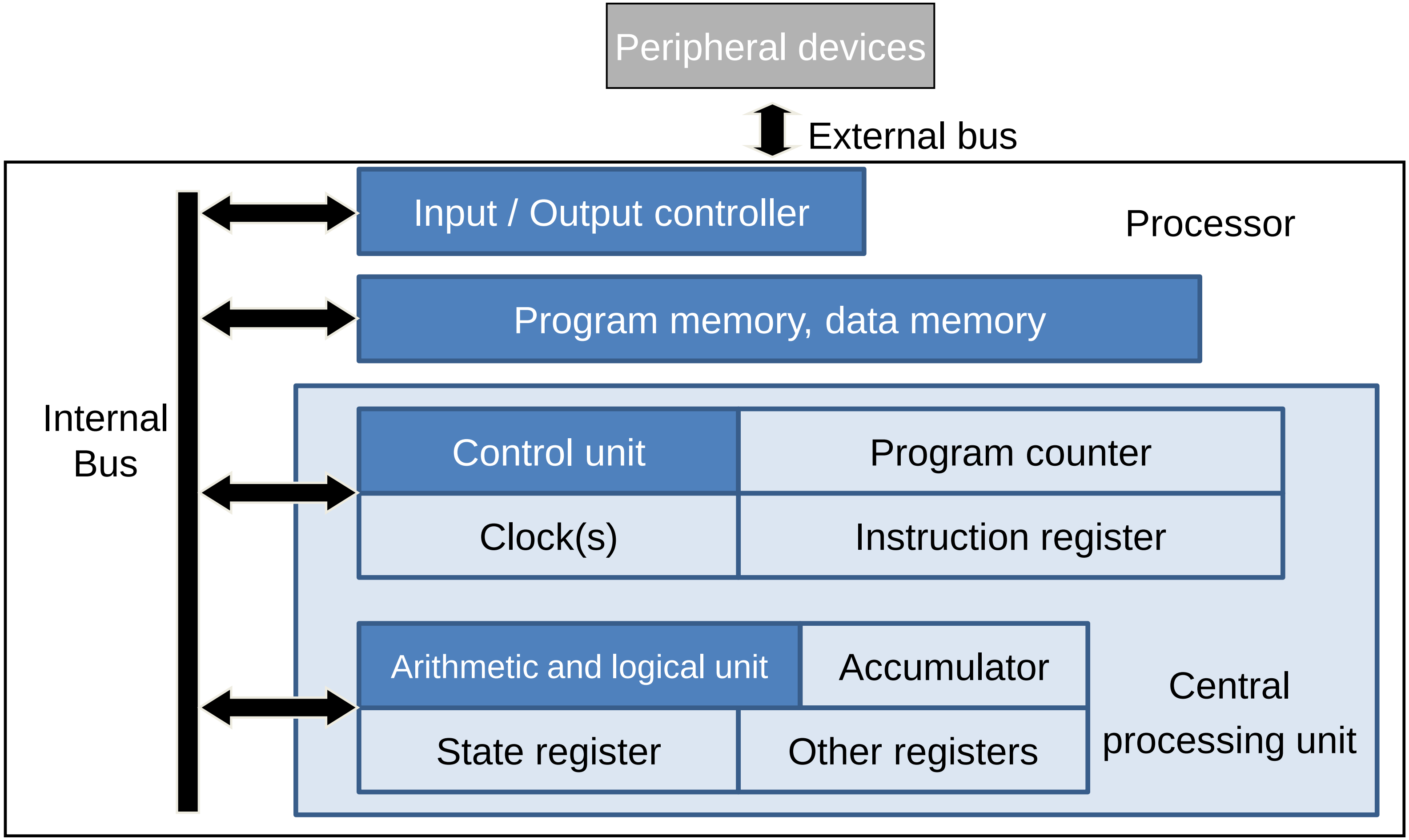
ARM	ARM	Co-processor		Co-processor	
ARM	ARM	DSP	DSP	DSP	DSP
		DSP	DSP	DSP	DSP

High Performance DSP Architectures

- Types of embedded processors
- DSP vs. GPP
- Multicore DSP architectures

Von Neumann Architecture

DSP : digital signal processors != GPP : General purpose processors
→ processors optimized and adapted to signal processing



Processor Generalities

- **Processor types :**
  - Microcontroller
  - Digital Signal Processors (DSP)
  - General purpose processors (GPP)
  - GP-GPU (General-Purpose Processing on Graphics Processing Units)
- **Choice factors:**
  - Price (architecture complexity, production technology)
  - Power consumption
  - CPU Performances
  - I/O performances
  - Memory capacity (data/code)

price
Control I/O
Engine control

Efficiency
Watt / price / size
Phones
Modem
Camera

Performance
Complex applications
Image
Video

Applications and Processor Types

TI Embedded Processors						
Microcontrollers (MCUs)		ARM®-Based Processors		Digital Signal Processors (DSPs)		
16-bit ultra-low power MCUs	32-bit real-time MCUs	32-bit ARM Cortex™-M3 MCUs	ARM Cortex-A8 & ARM9™ MPUs	DSP DSP+ARM	Multi-core DSP	Ultra Low power DSP
MSP430™	C2000™ Delfino™ Piccolo™	Stellaris® ARM Cortex-M3	Sitara™ ARM Cortex-A8 & ARM9	C6000™ DaVinci™ video processors OMAP™	C6000™	C5000™
Up to 25 MHz	40MHz to 300 MHz	Up to 100 MHz	375MHz to >1GHz	300MHz to >1Ghz +Accelerator	320 GMACS	Up to 300 MHz +Accelerator
Flash 1 KB to 256 KB	Flash, RAM 16 KB to 512 KB	Flash 8 KB to 256 KB	Cache, RAM, ROM	Cache RAM, ROM	Cache RAM, ROM	Up to 320KB RAM Up to 128KB ROM
Analog I/O, ADC LCD, USB, RF	PWM, ADC, CAN, SPI, I²C	USB, ENET MAC+PHY CAN, ADC, PWM, SPI	USB, CAN, SATA, SPI, PCIe, EMAC	USB, ENET, PCIe, SATA, SPI	SRIO, EMAC DMA, PCIe	USB, ADC McBSP, SPI, I²C
Measurement, Sensing, General Purpose	Motor Control, Digital Power, Lighting, Ren. Energy	Connectivity,Security, Motion Control, HMI, Industrial Automation	Industrial automation, POS & portable data terminals	Floating/Fixed Point Video, Audio, Voice, Security, Conferencing	Telecom test & meas media gateways, base stations	Audio, Voice Medical, Biometrics
$0.25 to $9.00	$1.50 to $20.00	$1.00 to $8.00	$5.00 to $25.00	$5.00 to $200.00	$40.00 to $200.00	$3.00 to $10.00

TI Processors and Applications

	Digital Media Processors	OMAP Applications Processors	C6000 Digital Signal Processors	C5000 Digital Signal Processors	C2000 Microcontrollers	MSP430 Microcontrollers	Stellaris 32-Bit ARM Cortex-M3 MCUs
Audio	🟫	🟦	🟩	🟦			🟥
Automotive	🟫	🟦			🟪		
Communications	🟫		🟩	🟦	🟪		🟥
Industrial	🟫	🟦	🟩	🟦	🟪	🟧	🟥
Medical	🟫	🟦	🟩	🟦	🟪	🟧	🟥
Security	🟫	🟦				🟧	🟥
Video	🟫	🟦	🟩				
Wireless		🟦	🟩	🟦		🟧	🟥
Key Feature	Complete tailored video solution	Low power and high performance	High performance	Power-efficient performance	Performance, integration for greener industrial applications	Ultra-low power	Open architecture software, rich communications options

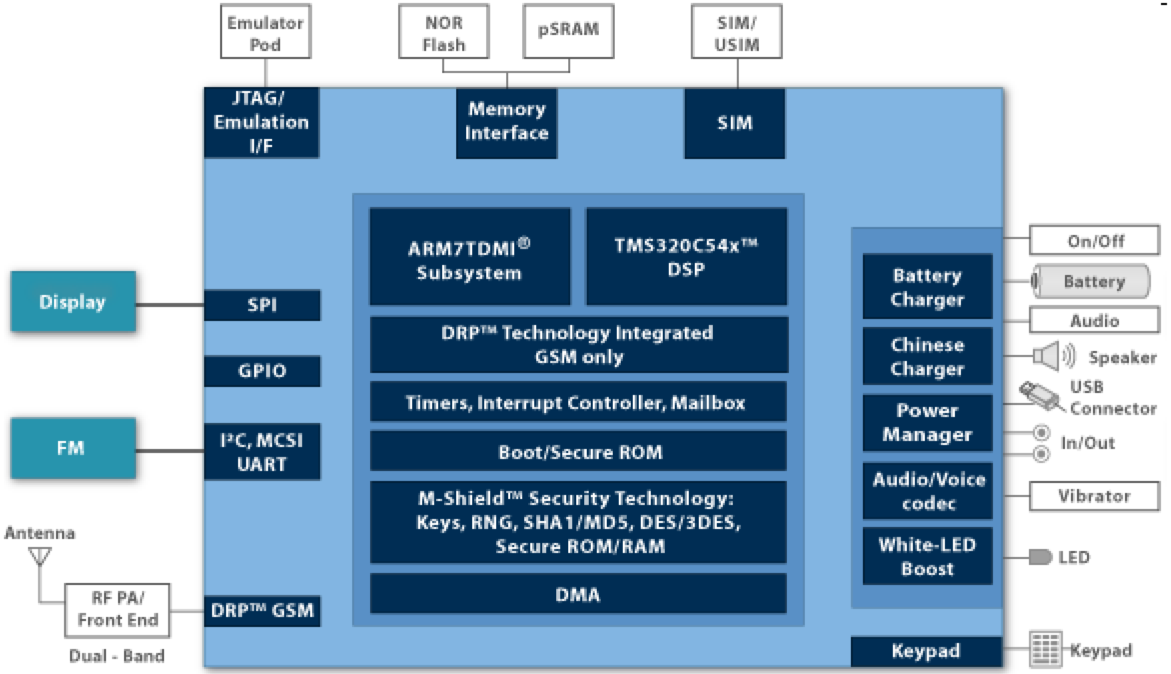
Source: TI

Low cost System-on chip

Locosto (TCS2305)

Very cheap
65-nm technology

ARM7 CPU + c54x
Minimal functionalities



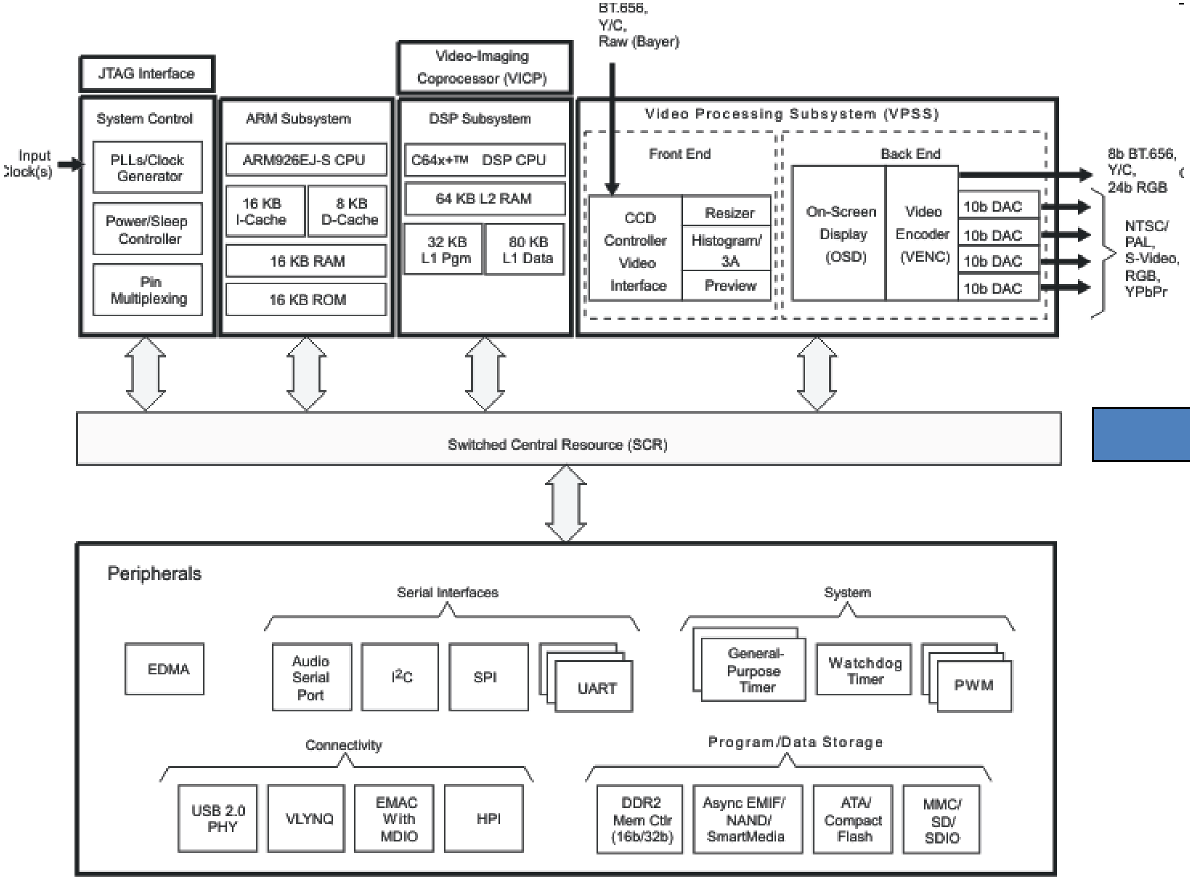
Video Processing DSP

Da Vinci TMS320DM644x

1 DSP tms320c64x+ (720MHz max) Video I/O for screen

1 ARM9

1 Video Accelerator

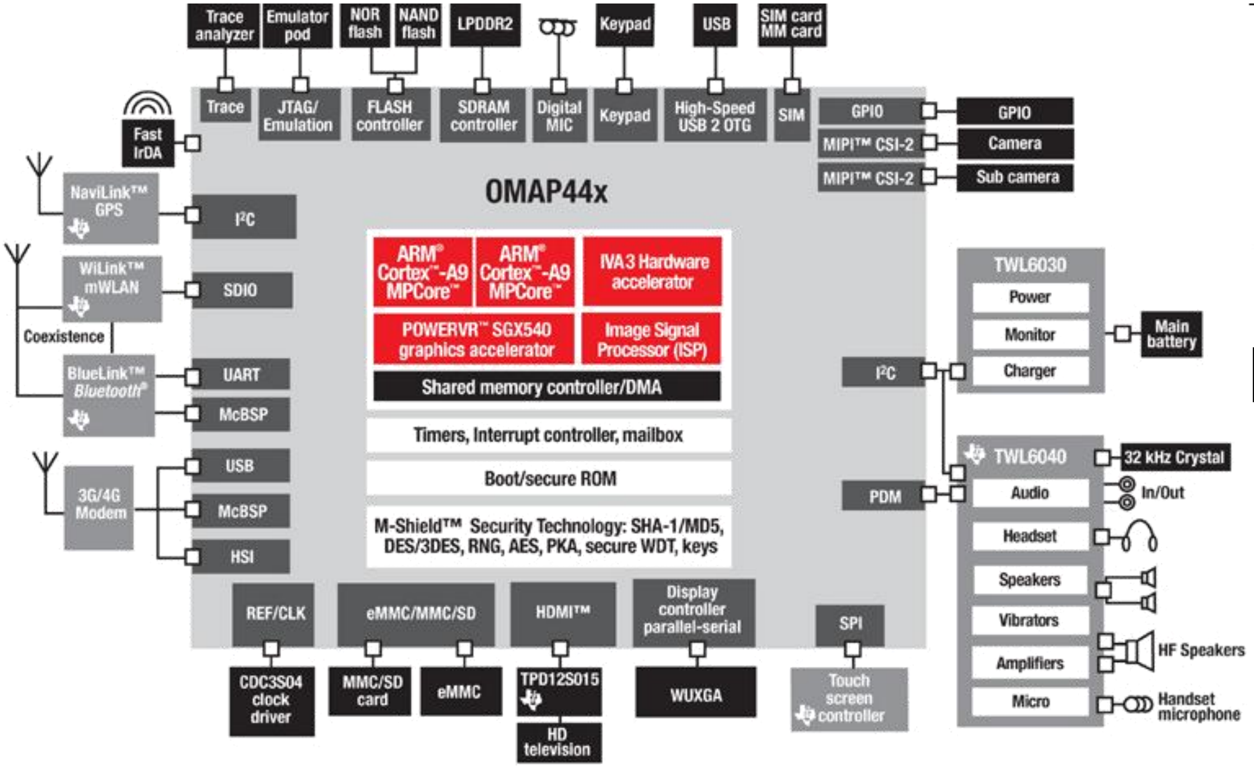


Advanced System-on chip

OMAP™ 4 Platform: OMAP4430/OMAP4440

High power of computation
Still low power consumption
Interfaces to modern peripherals

2 General purpose ARM Cortex A9 cores (1GHz)
IVA Supporting 1080p video encoding/decoding
3D graphics accelerator
No DSP!!!



DSPs vs. GPPs

- **Optimization of the compilation and instruction set for signal processing**
- **Reduced Power consumption in DSPs**
- **Memory Management Unit (MMU) in the general purpose processor :**

  In DSPs, any memory is accessed by addresses: registers, stack, heap, OS memory…

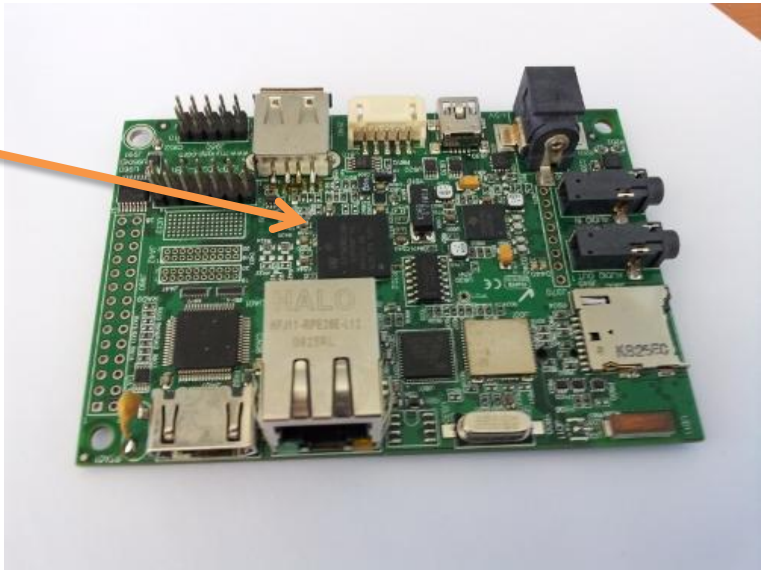
  Advanced OS (like Linux) need pagination: a virtual memory space in pages

  The MMU converts the virtual addresses into the physical addresses of the hardware

  The MMU can protect memory spaces from unwanted accesses

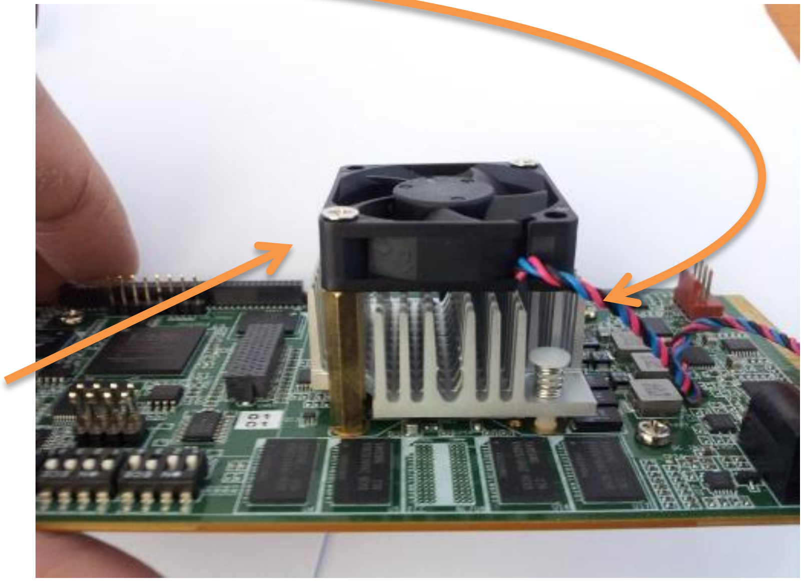
  DSPs use Real-Time OS: the c66x has SYS-BIOS
- **Kalray VLIW core is an exception: a DSP with MMU**

Power Dissipation



- Less than 2W: no specific dissipation problem

- 2 to 7W: heat sink and hot spot management



- 7W+: heat sink, fan and complex hot spot management

Why Go Multicore?

- **Frequency increase → power consumption → heat**

  heat → need for cooling, more faults, reduced longevity

  Dynamic Power = Cte x capacity x voltage^2 x frequency

  But augmenting frequency augments leakage so voltage must be higher

  Frequency x 1.5 → Power x2 on Freescale MPC8641 [1]

  Cores x2 → Power x1.3 on Freescale MPC8641 [1]

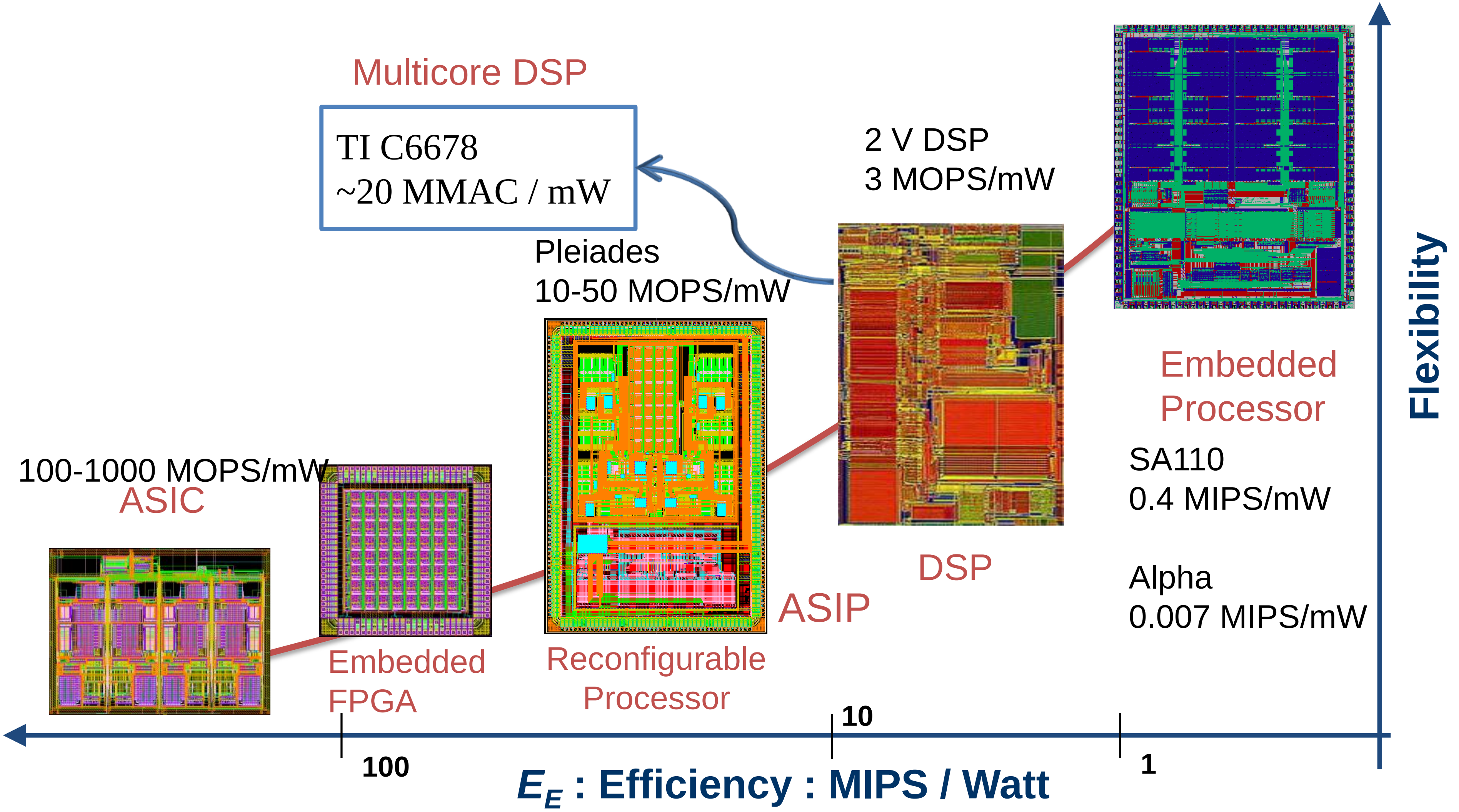
  Moreover, augmenting frequency may require longer pipeline

- **To increase MIPS and limit Watts**

  Increase number of core instead than frequency

Source:Freescale « Embedded Multicore: An Introduction»

Trade-off between Flexibility and Energy Efficiency



Multicore DSP

TI C6678
~20 MMAC / mW

2 V DSP
3 MOPS/mW

Pleiades
10-50 MOPS/mW

100-1000 MOPS/mW
ASIC

Embedded
FPGA

Reconfigurable
Processor

ASIP

DSP

Embedded
Processor

SA110
0.4 MIPS/mW

Alpha
0.007 MIPS/mW

Why Go Heterogeneous?

- **3 sources of heterogeneity**
  - Non uniform cores implementing different instruction sets
  - Combining software cores with hardware IPs
  - Non uniform communication performance

- **Heterogeneity improves performance**
  - Repetitive and costly actors can be efficiently computed by hardware logic (ASIC)
  - Actors with some control and reconfiguration needs are suited for DSPs
  - Control tasks with many conditions are suited to be run on GPP

Source:Freescale « Embedded Multicore: An Introduction»

Institut National des Sciences Appliquées

TMS320TCI6488 (2007)

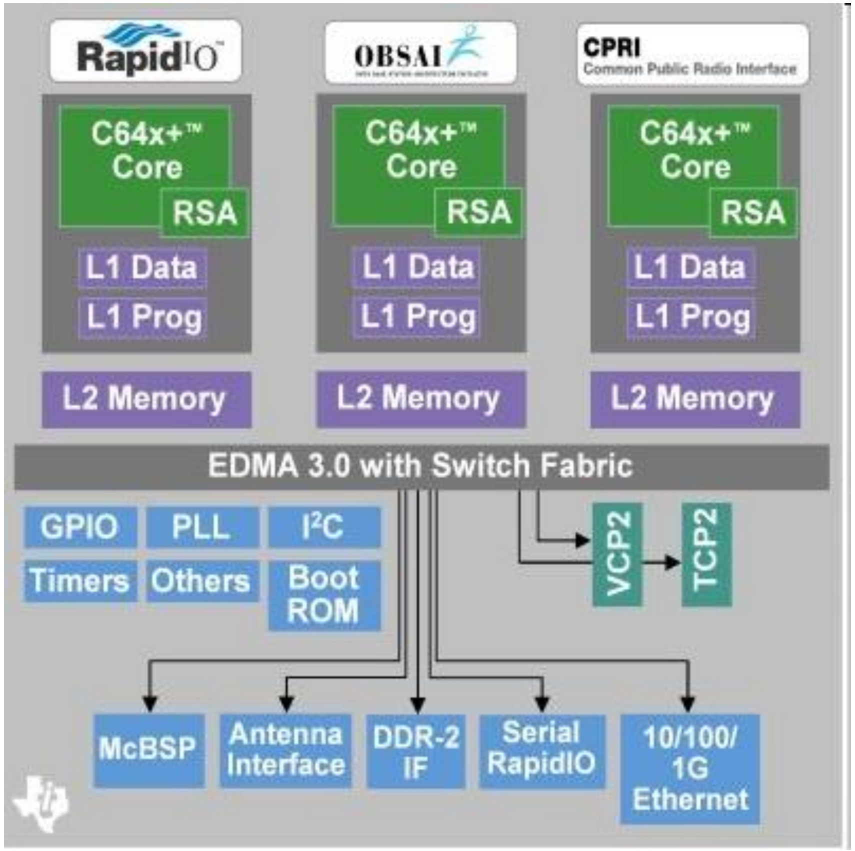
Towards multicore DSPs

TCI6488: Tri-core Telecommunication oriented DSP → Up to 1GHz/24Gips

→ Each core is programmed independantly → 3MB L2 memory

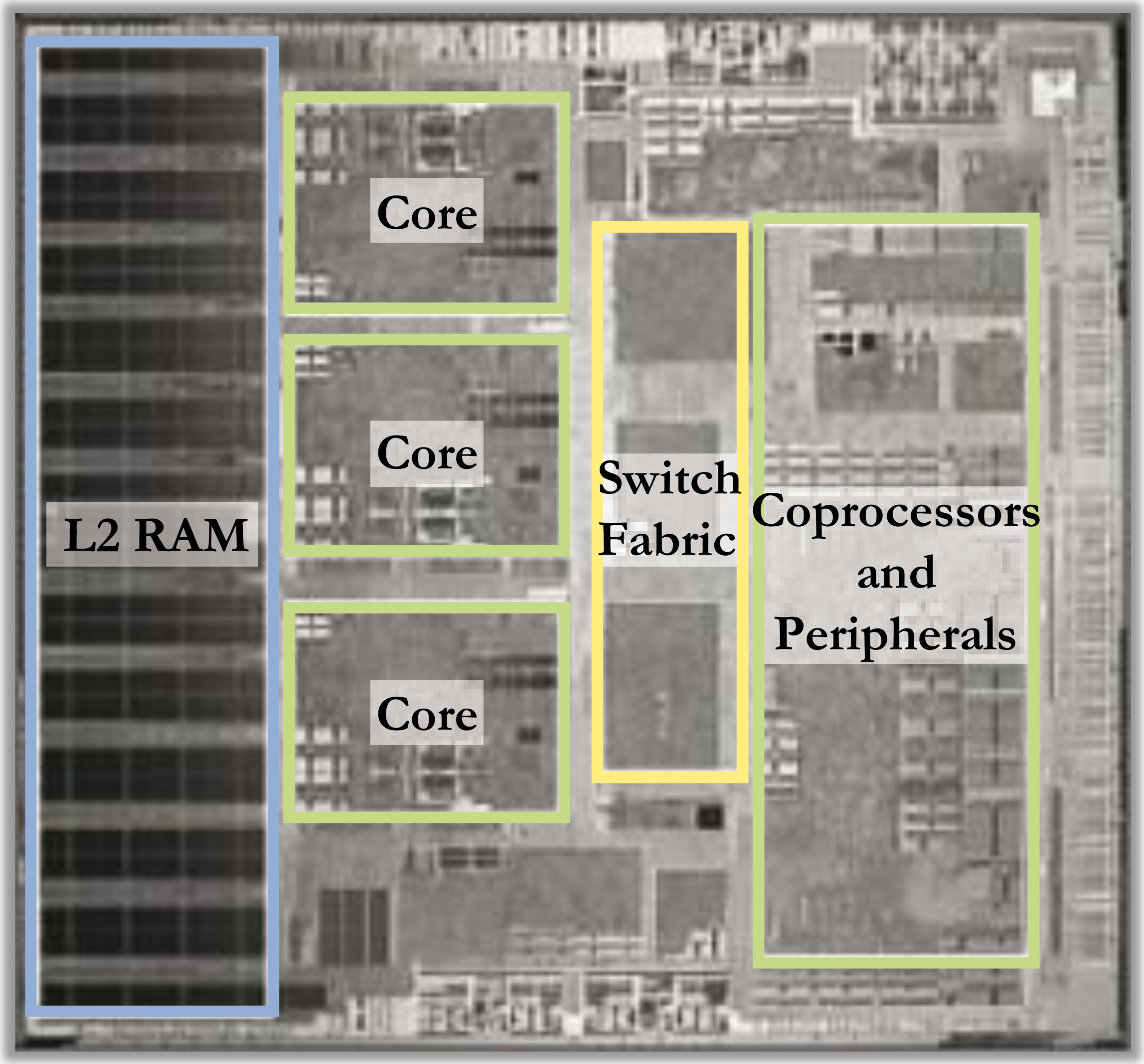
→ RSA: specific instruction set for CDMA operation (3G oriented)

→ I/O driven by EDMA

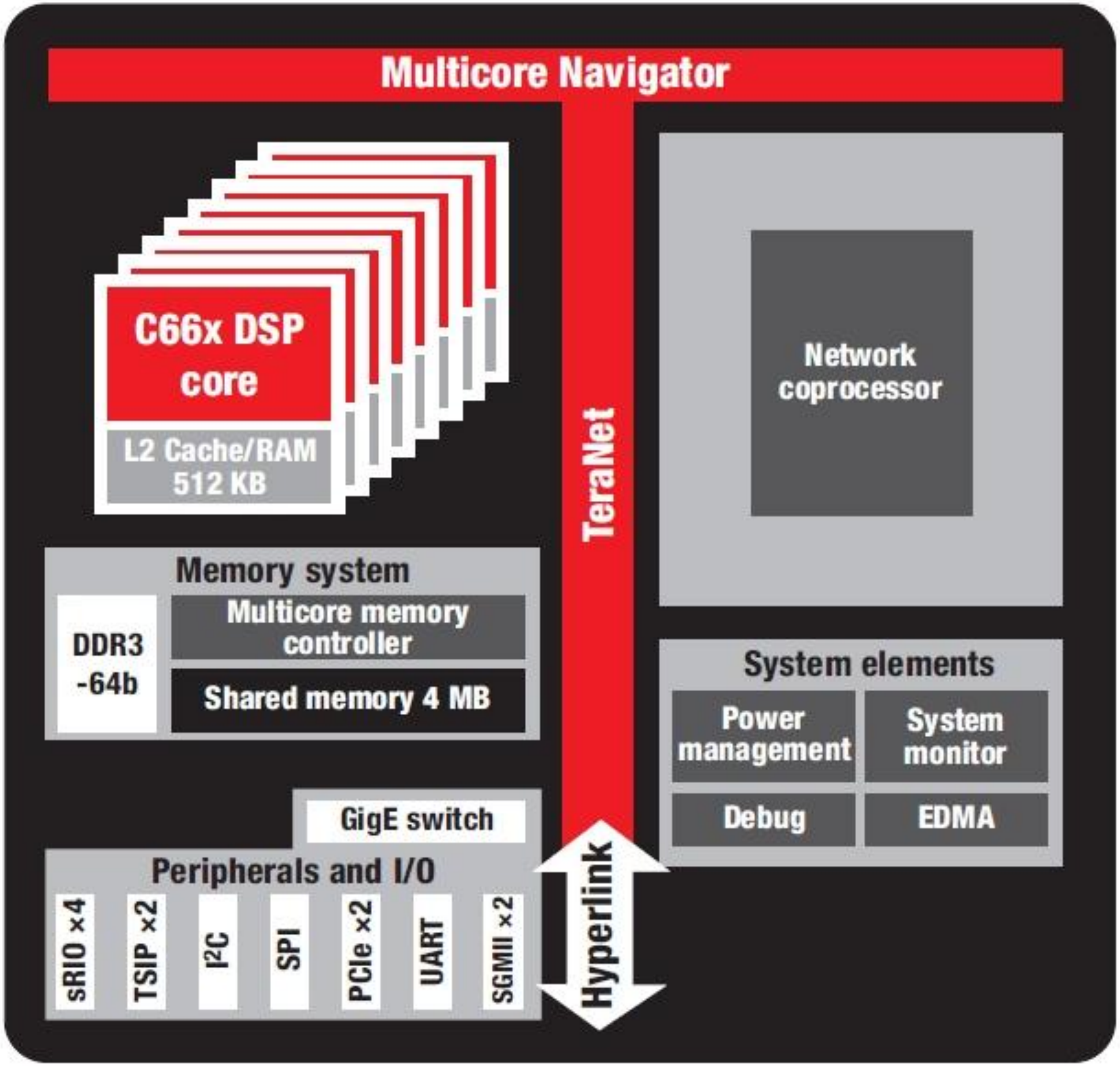


Source:TI

TMS320TCI6488 (2007)



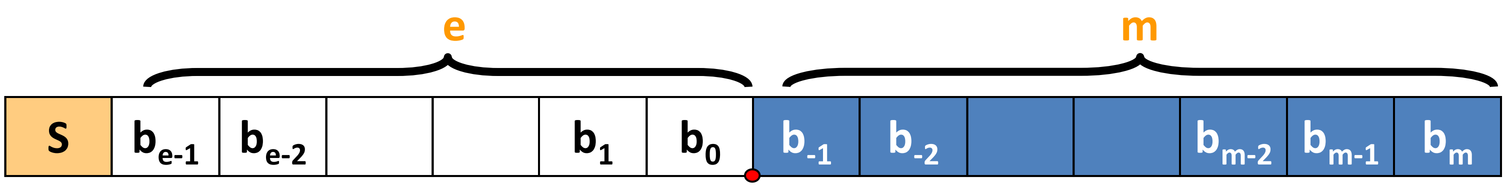
TMS320C6678 – Keystone I (2011)



New c66x core: floating point arithmetic

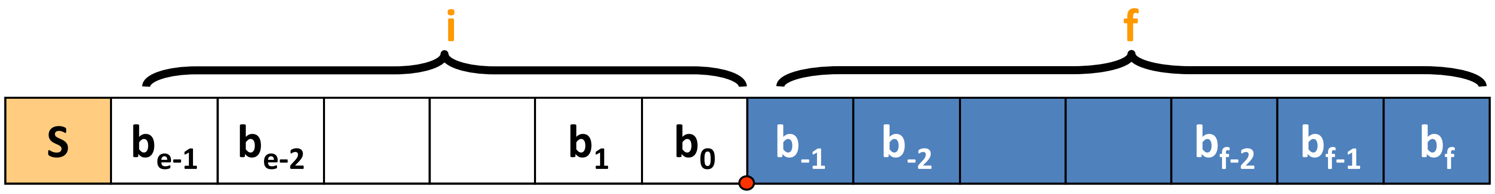
Representing real numbers

Floating-point arithmetics : Sign - exponent - mantissa



$(-1)^S. (1,m) . 2^e$

Fixed-point arithmetics : Sign – integer part – fractional part



$(-1)^S. e,f$

TMS320C6678 – Keystone I (2011)

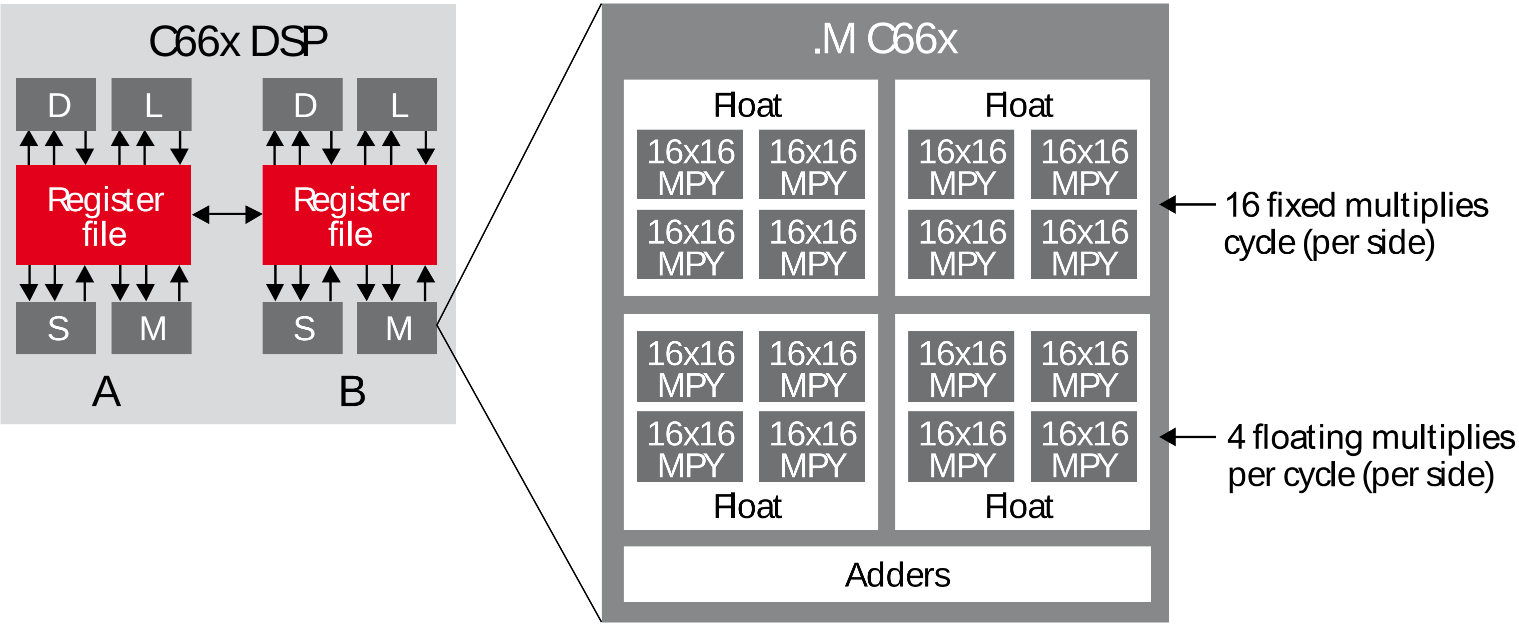
TCI6678: Octo-core DSP → Up to 1.25GHz/320 GMACs/160 GFlops

→ Each core is programmed independantly → 8MB L2 memory

→ I/O driven by Multi-core Navigator to automate transfers between cores

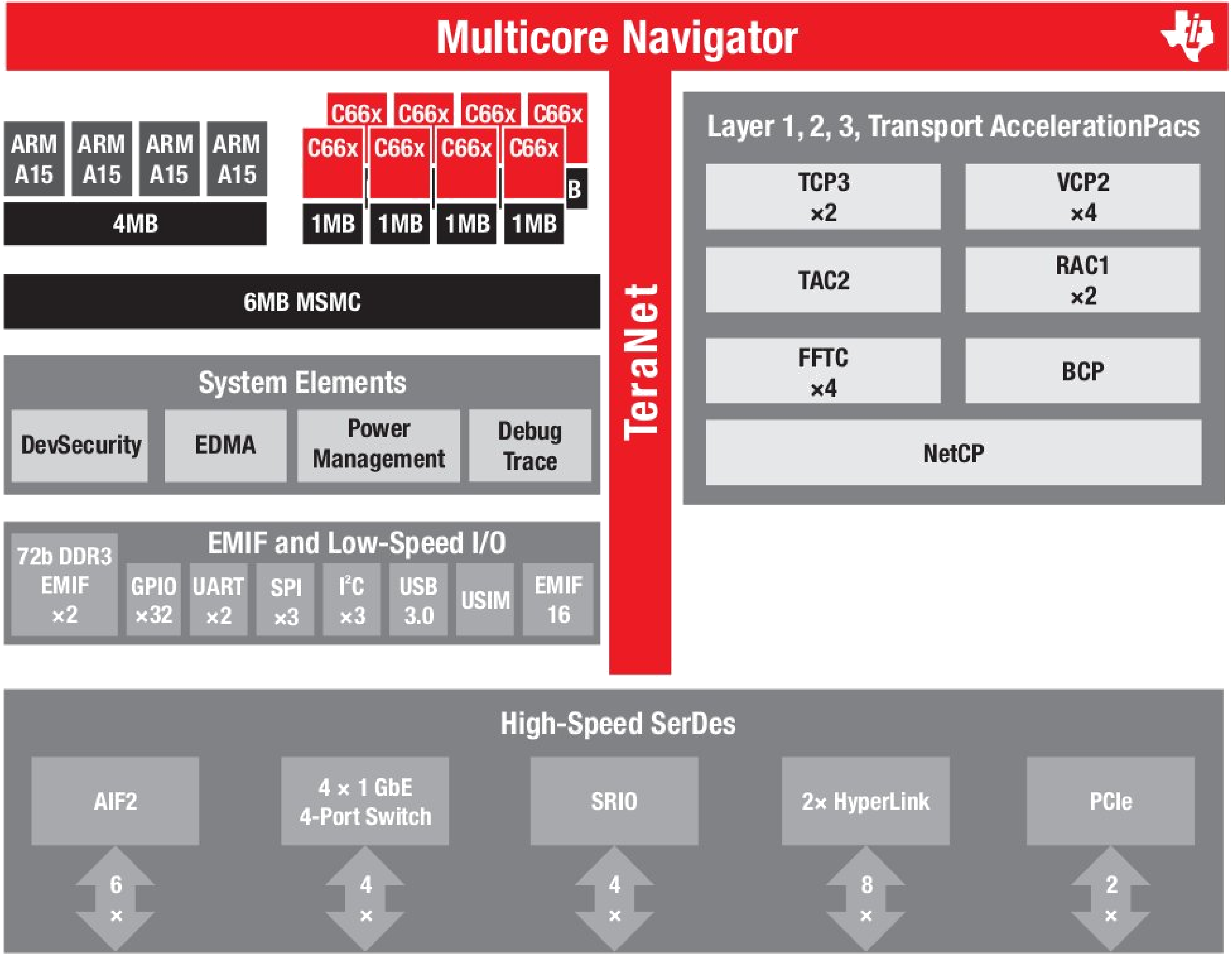
→ Fixed and floating-point ALUs

Core c66x = Arithmetic and logic units (L & S), Data management units (D), Multiplication units (M)



Source:TI

TI TMS320TCI6636 – Keystone II (2013)



Multicore Navigator

ARM A15 | ARM A15 | ARM A15 | ARM A15

4MB

C66x C66x C66x C66x

C66x C66x C66x C66x

1MB 1MB 1MB 1MB B

6MB MSMC

TeraNet

Layer 1, 2, 3, Transport AccelerationPacs

TCP3 ×2

VCP2 ×4

TAC2

RAC1 ×2

FFTC ×4

BCP

NetCP

System Elements

DevSecurity

EDMA

Power Management

Debug Trace

EMIF and Low-Speed I/O

72b DDR3 EMIF ×2

GPIO ×32

UART ×2

SPI ×3

I²C ×3

USB 3.0

USIM

EMIF 16

High-Speed SerDes

AIF2 — 6 ×

4 × 1 GbE 4-Port Switch — 4 ×

SRIO — 4 ×

2× HyperLink — 8 ×

PCIe — 2 ×

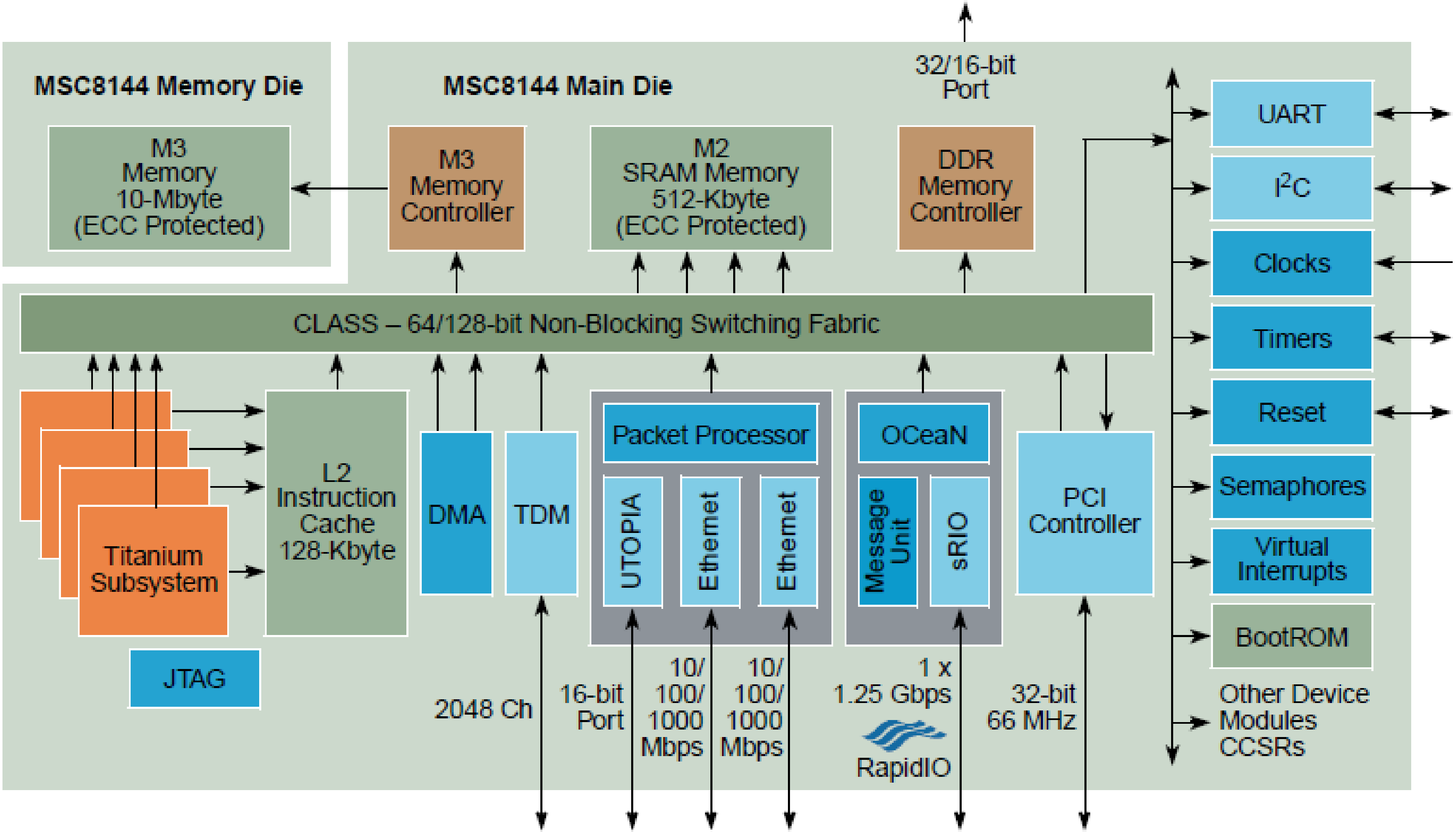
Source: TI

TI TMS320TCI6636 – Keystone II (2013)

- **8 C66x @ 1.2GHz**
  - 38.4 GMacs/Core for Fixed Point
  - 19.2 GFlops/Core for Floating Point
- **Memory**
  - 32KB L1P Per Core
  - 32KB L1D Per Core
  - 1MB Local L2 Per Core
  - 6 MB MSM SRAM Memory Shared by 8 DSPs
- **ARM Cortex A15 Quad Core Cluster @ 1.2GHz**
  - 32KB L1I Per Core
  - 32KB L1D Per Core
  - 4MB L2 Cache Memory Shared by Quad Core
  - AMBA 4.0 AXI Coherency Extension Master Port connected to MSMC
- **Multicore Navigator**
  - 16k Multi-purpose Hardware Queues with hardware queue manager
  - Packet-Based DMA for Zero-Overhead Transfers

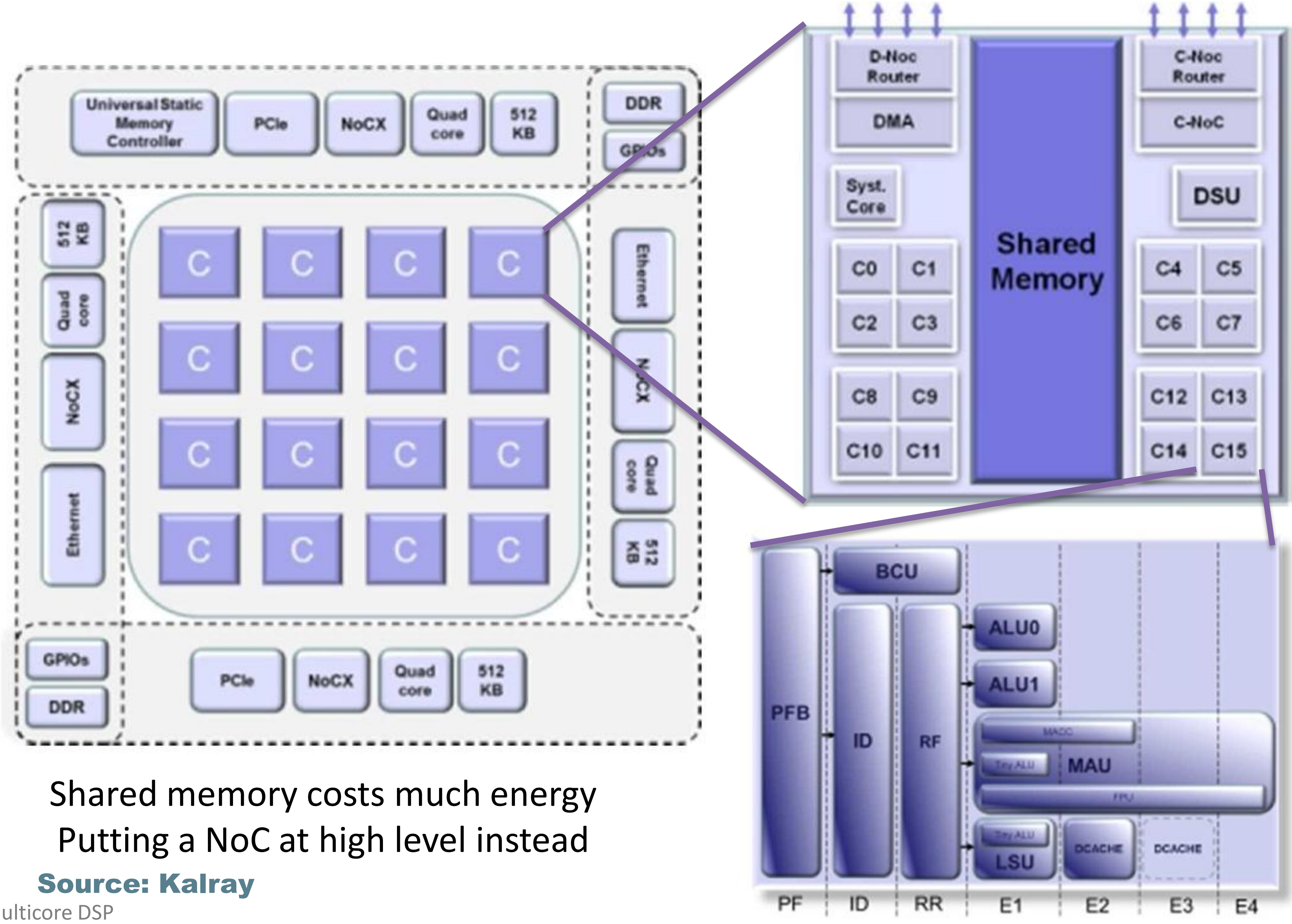
Source: TI

Freescale MSC8144 Starcore DSP



Source: Freescale

Kalray MPPA



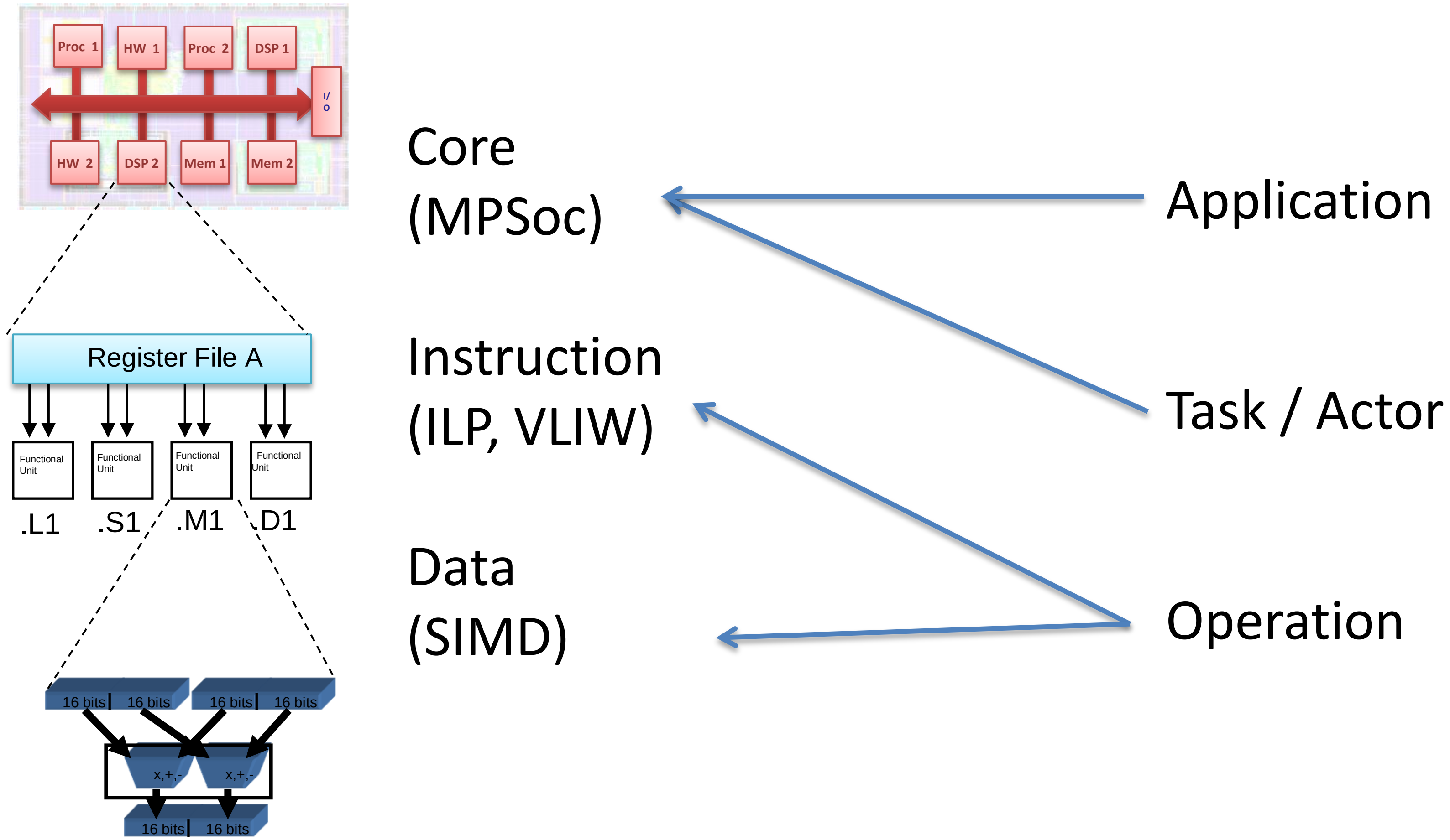
Shared memory costs much energy
Putting a NoC at high level instead

Source: Kalray

Institut National des Sciences Appliquées

Levels of parallelism in Multicore DSP Architectures

Architecture Levels of Parallelism



Core
(MPSoc)

Instruction
(ILP, VLIW)

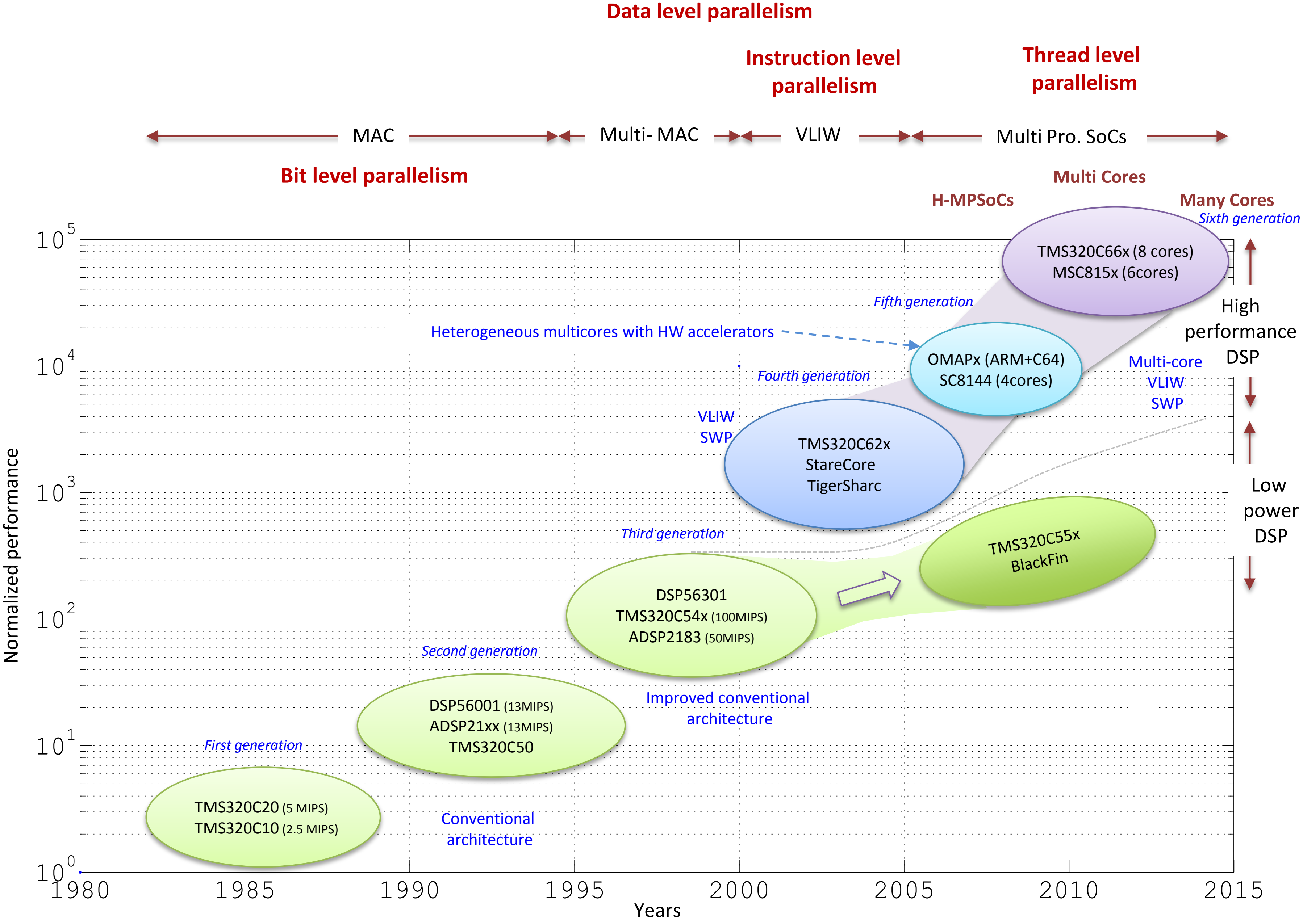
Data
(SIMD)

Application

Task / Actor

Operation

DSP Evolution



Data level parallelism

Instruction level parallelism

Thread level parallelism

MAC

Multi- MAC

VLIW

Multi Pro. SoCs

Bit level parallelism

H-MPSoCs

Multi Cores

Many Cores

First generation

TMS320C20 (5 MIPS)
TMS320C10 (2.5 MIPS)

Conventional architecture

Second generation

DSP56001 (13MIPS)
ADSP21xx (13MIPS)
TMS320C50

Third generation

DSP56301
TMS320C54x (100MIPS)
ADSP2183 (50MIPS)

Improved conventional architecture

TMS320C55x
BlackFin

VLIW
SWP

TMS320C62x
StareCore
TigerSharc

Fourth generation

Heterogeneous multicores with HW accelerators

OMAPx (ARM+C64)
SC8144 (4cores)

Fifth generation

Sixth generation

TMS320C66x (8 cores)
MSC815x (6cores)

Multi-core
VLIW
SWP

High performance DSP

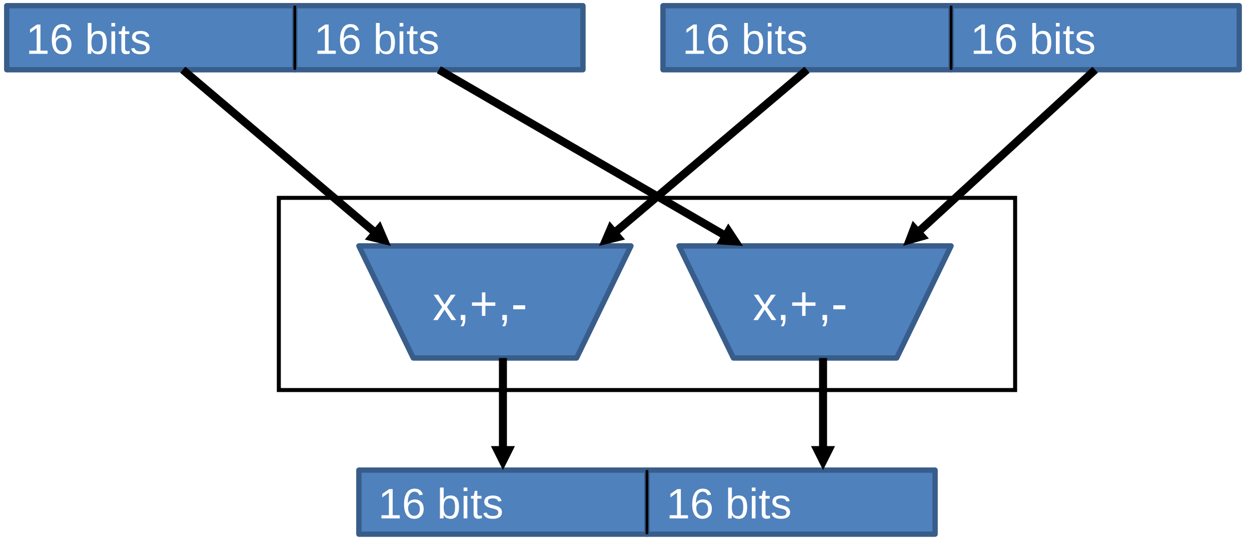
Low power DSP

Institut National des Sciences Appliquées

Single Instruction Multiple Data

- **Splitting ALU into subparts**

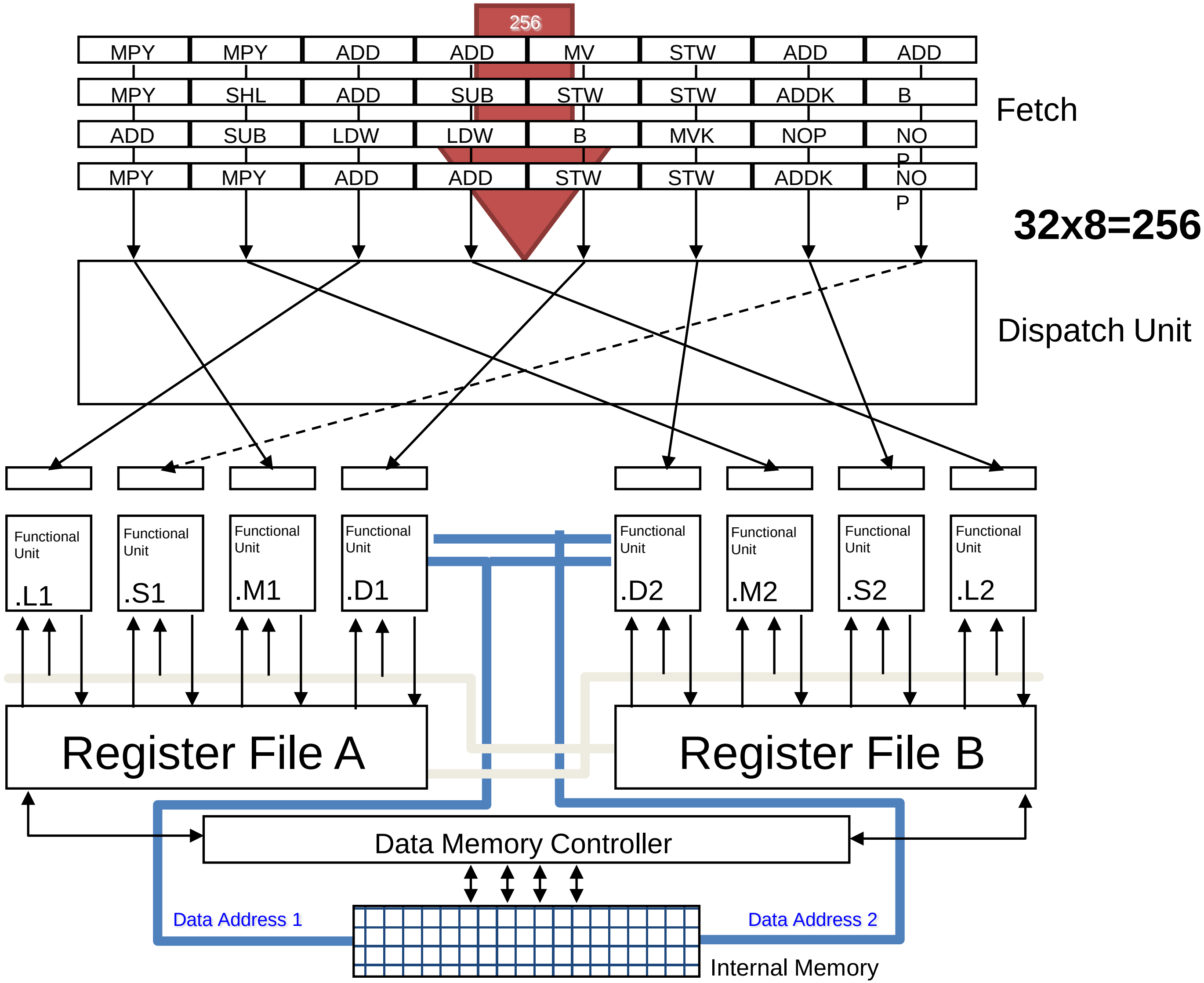
  Example: 2 16-bit additions on 1 32-bit adder



Very Long Instruction Word

- **VLIW characteristic**
  - Architecture made-up of parallel FU
  - Several instructions par cycle embedded in a macro-instruction
  - Homogeneous architecture : more orthogonal
    - Close to RISC processor
  - Uniform register file made-up of several registers
  - Load-Store architecture

VLIW Example : C64x



MPY	MPY	ADD	ADD	MV	STW	ADD	ADD
MPY	SHL	ADD	SUB	STW	STW	ADDK	B
ADD	SUB	LDW	LDW	B	MVK	NOP	NOP
MPY	MPY	ADD	ADD	STW	STW	ADDK	NOP

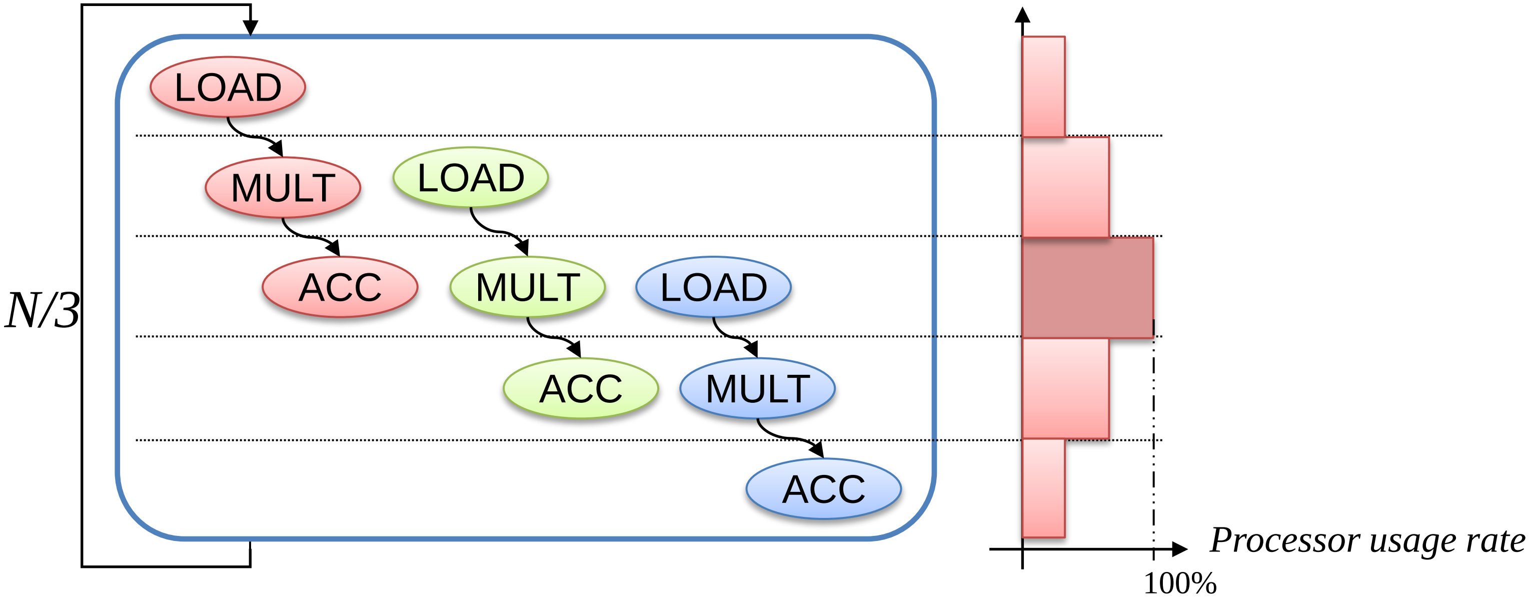
32x8=256 bits

L:ALU
S:Shift+ALU
M:Multplier
D:Address U

WLIV Compiler: Loop unrolling

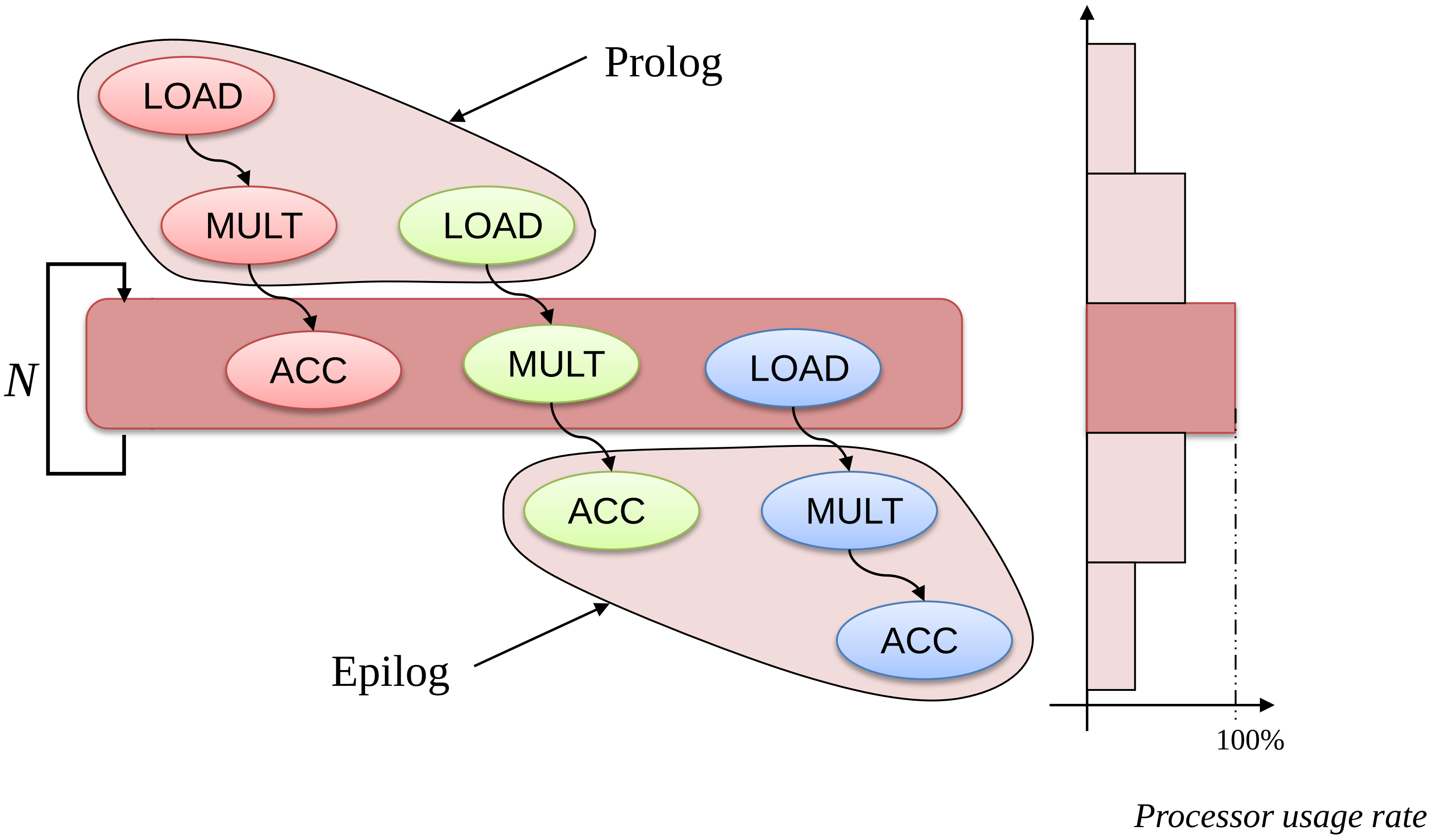
```
For(i=0;i<N;i++)
{
 ACC=ACC + x[i].h[i]
}
```

```
For(i=0;i<N;i+=3)
{
  ACC=ACC + x[i].h[i]
  ACC=ACC + x[i+1].h[i+1]
  ACC=ACC + x[i+2].h[i+2]
}
```



WLIV Compiler: Software pipelining

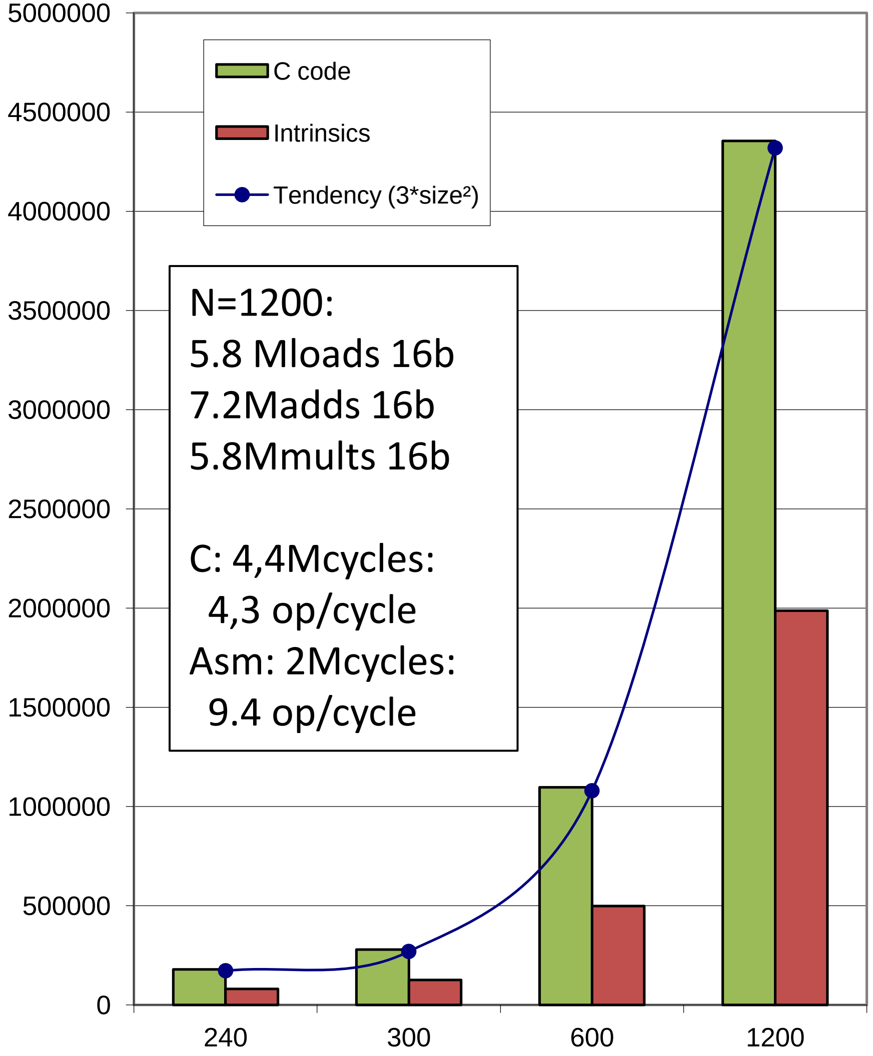
- **Improving the throughput at the cost of latency and memory**



Institut National des Sciences Appliquées

Convolution C code for VLIW and SIMD

$$Y(k) = \sum_{l=0}^{N-1} Z(l) C((l - k) \bmod N)$$



N=1200:
5.8 Mloads 16b
7.2Madds 16b
5.8Mmults 16b

C: 4,4Mcycles:
4,3 op/cycle
Asm: 2Mcycles:
9.4 op/cycle

```
void fir_circbuf_modified
(
    const RACH_cpx *restrict vectorb,
    const RACH_cpx *restrict vectora,
    RACH_cpx *restrict vectorc,
    short size,
    int type
)
{
    int i, j, xindex,imag, real, h1h0, x1x0;
    const short *restrict x = ((short*) vectorb);
    const short *restrict h = (short*) vectora;
    short *restrict r = (short*) vectorc;
    size = 2*size;

    #pragma MUST_ITERATE(4, ,4);
    #pragma UNROLL(4);
    for (i = 0; i < size; i += 2)
    {
        imag = 0;
        real = 0;

        #pragma MUST_ITERATE(2,,2)
        for (j = 0; j < size; j += 4)
        {
            xindex = j-i;
            xindex = xindex+size*(xindex<0);
            h1h0 = _mem4((void*)&h[j+0]);
            x1x0 = _mem4((void*)&x[xindex]);
            real += _dotpn2(h1h0,x1x0);
            imag += _dotp2(h1h0,_packlh2(x1x0,x1x0));

            xindex = j-i+2;
            xindex = xindex+size*(xindex<0);
            h1h0 = _mem4((void*)&h[j+2]);
            x1x0 = _mem4((void*)&x[xindex]);
            real += _dotpn2(h1h0,x1x0);
            imag += _dotp2(h1h0,_packlh2(x1x0,x1x0));
        }

        r[i  ] = (imag >> 15);
        r[i+1] = (real >> 15);
    }

}
```

Core-Level Parallelism

Inter-Processor Communication and Architecture Models

Types of Inter-Processor Communications

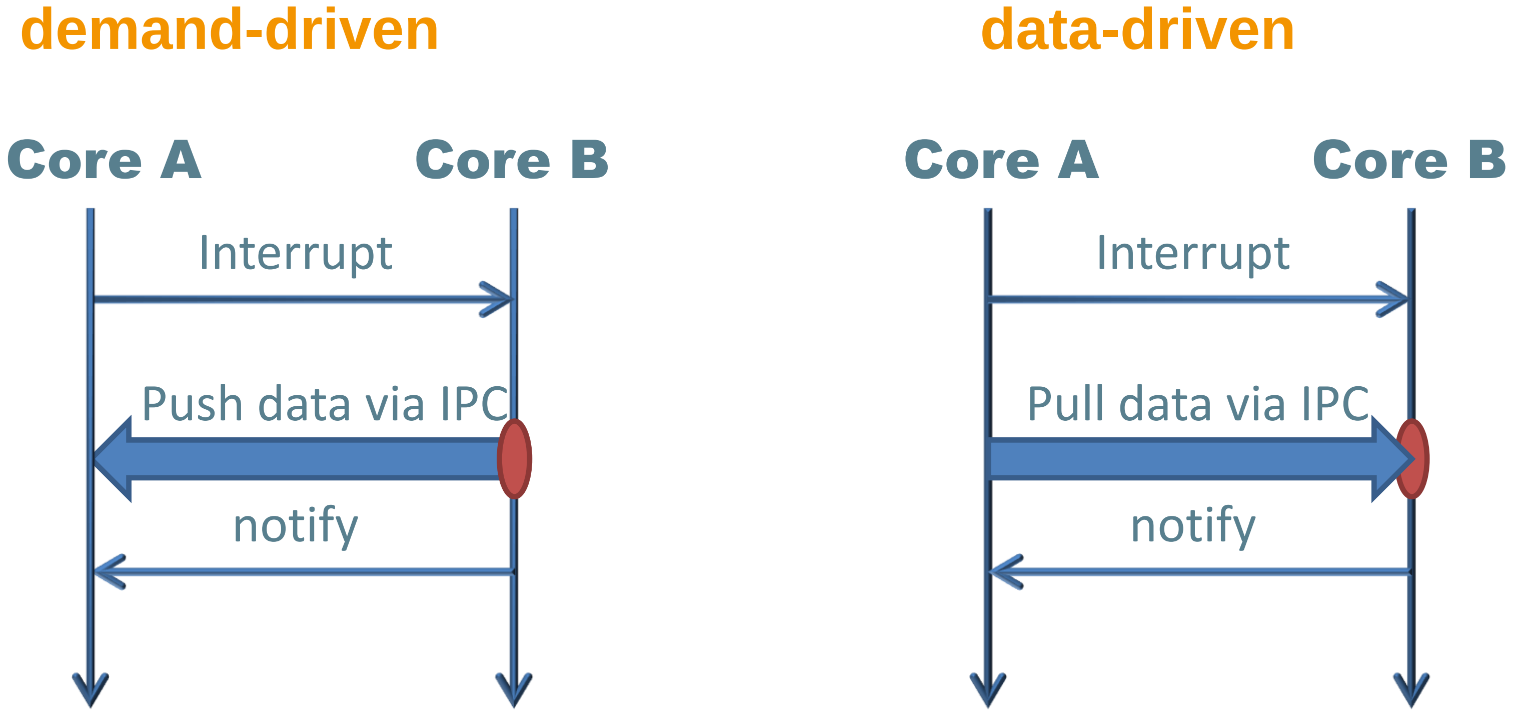
- **Transmitting one token via inter-processor communication or shared memory**
- **Direct Signaling**

  Interrupting a core from another core to either push or pull data
- **Indirect Signaling**

  Delegating the transmission to a DMA (Direct Memory Access) element
- **Atomic arbitration**

  Via hardware semaphores in case of shared memory

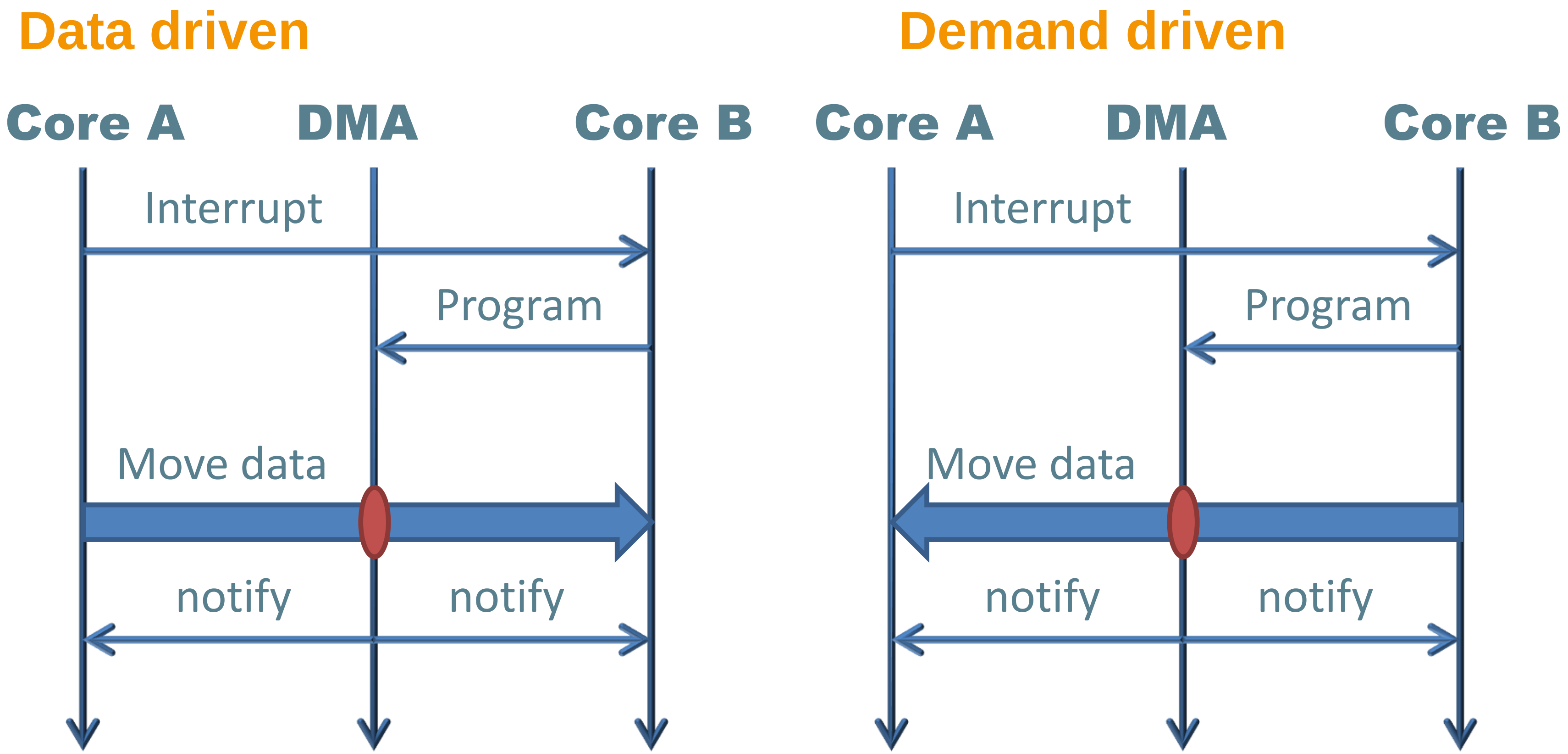
Direct Signaling Communication



- **Distributed or shared memory**
- **In demand-driven case, first interrupt can be avoided**

  if core A is constantly demanding data and core B cannot erase data befor e it is consumed (or if data can be discarded)
- **Inter-Processor communication can be ethernet, SRIO…**

Indirect Signaling Communication: delegating to a DMA



- **Distributed or shared memory**

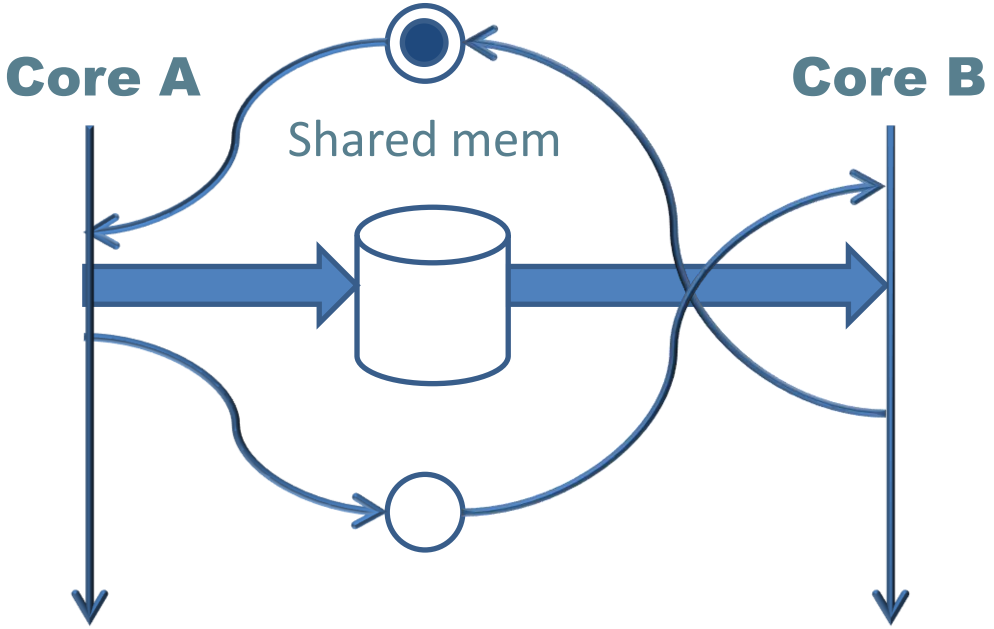
  DMA must be able to access the whole memory space

  DMA is another master on the memory bus

  Cores A and B are free to compute during DMA transfer

Atomic Arbitration: hardware or software semaphores

Protecting shared memory accesses by semaphores



- **1-place FIFO on Shared memory**

  Possibility of N-places FIFO with a round buffer and read and write indexes

  2-place FIFO = « ping-pong » buffer

- **More than a mutex**

  The 2 semaphores ensure alternate accesses from cores A and B

Communication Speed

- **On Keystone I, data transfers have a speed of about**

  L2 access: 5.3 GB/s

  DDR Access: 2.6 GB/s

  MSMC Access: 8 GB/s

  For comparison: Raw HD video 1080p60 4:2:0 = 0,19 GB/s

- **Inter-processor communication libs mask the complexity**

Modeling Architectures

Models of Architecture

- **SystemC**

  C++ templates and libraries used to simulate hardware modules

- **AADL**

  Separating hardware and software but specifying both and oriented towards threads and processes

- **IP-XACT**

  More a syntax for serialization than a real model

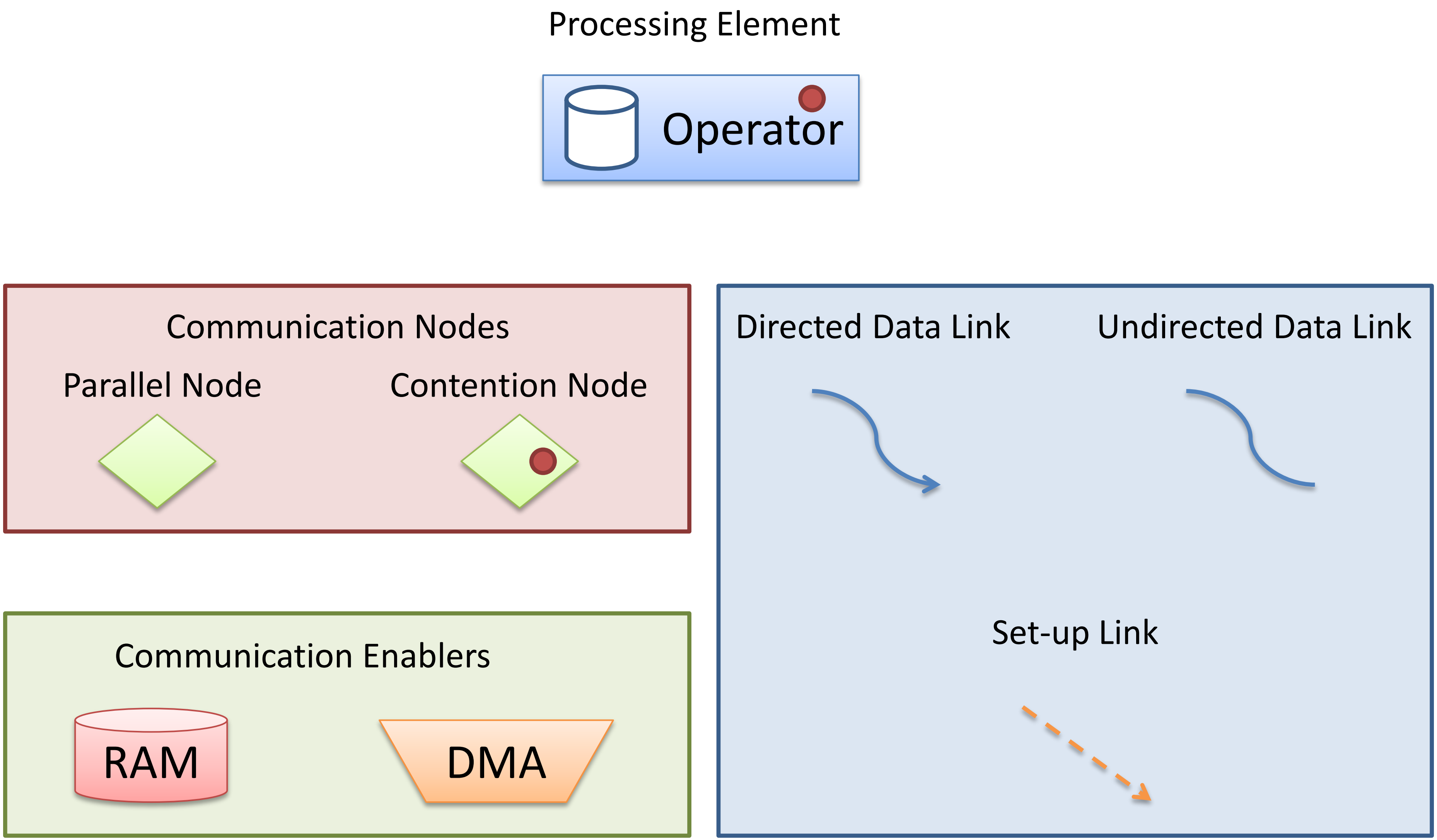
- **Custom models**

  In MAPS compiler, in SynDEx rapid prototyping tool, in PREESM…

Models of Architecture

- Often oriented towards hardware design debug
- Often custom and with no precise semantics
- Often no real separation between application and hardware

System-Level Architecture Model



Processing Element

Operator

Communication Nodes

Parallel Node

Contention Node

Communication Enablers

RAM

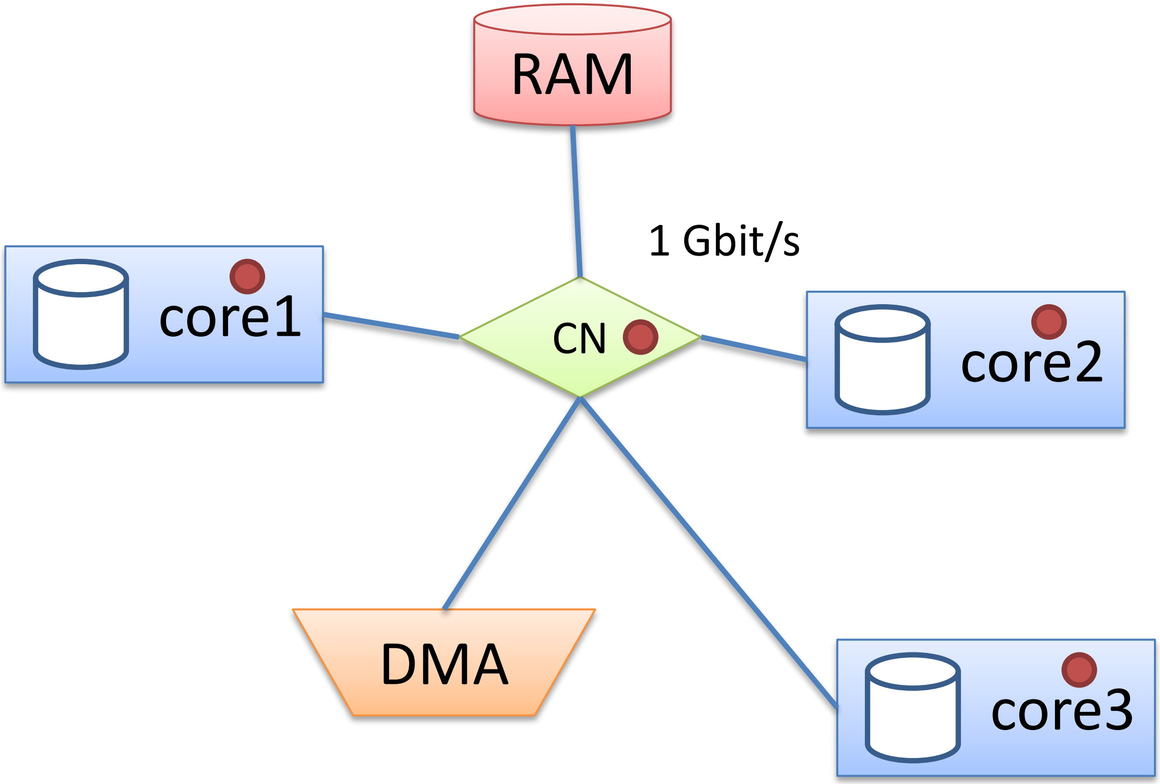
DMA

Directed Data Link

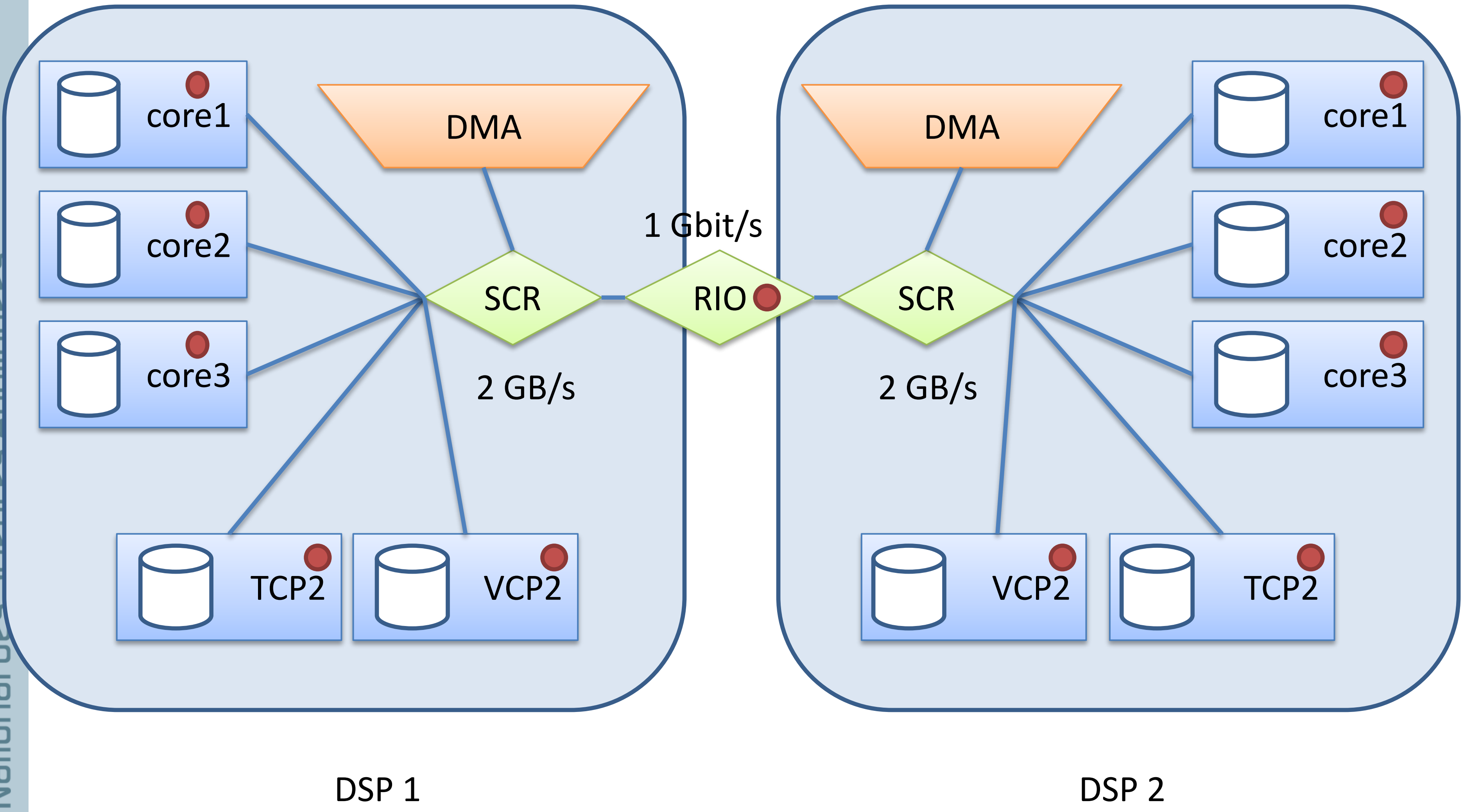
Undirected Data Link

Set-up Link

S-LAM Examples



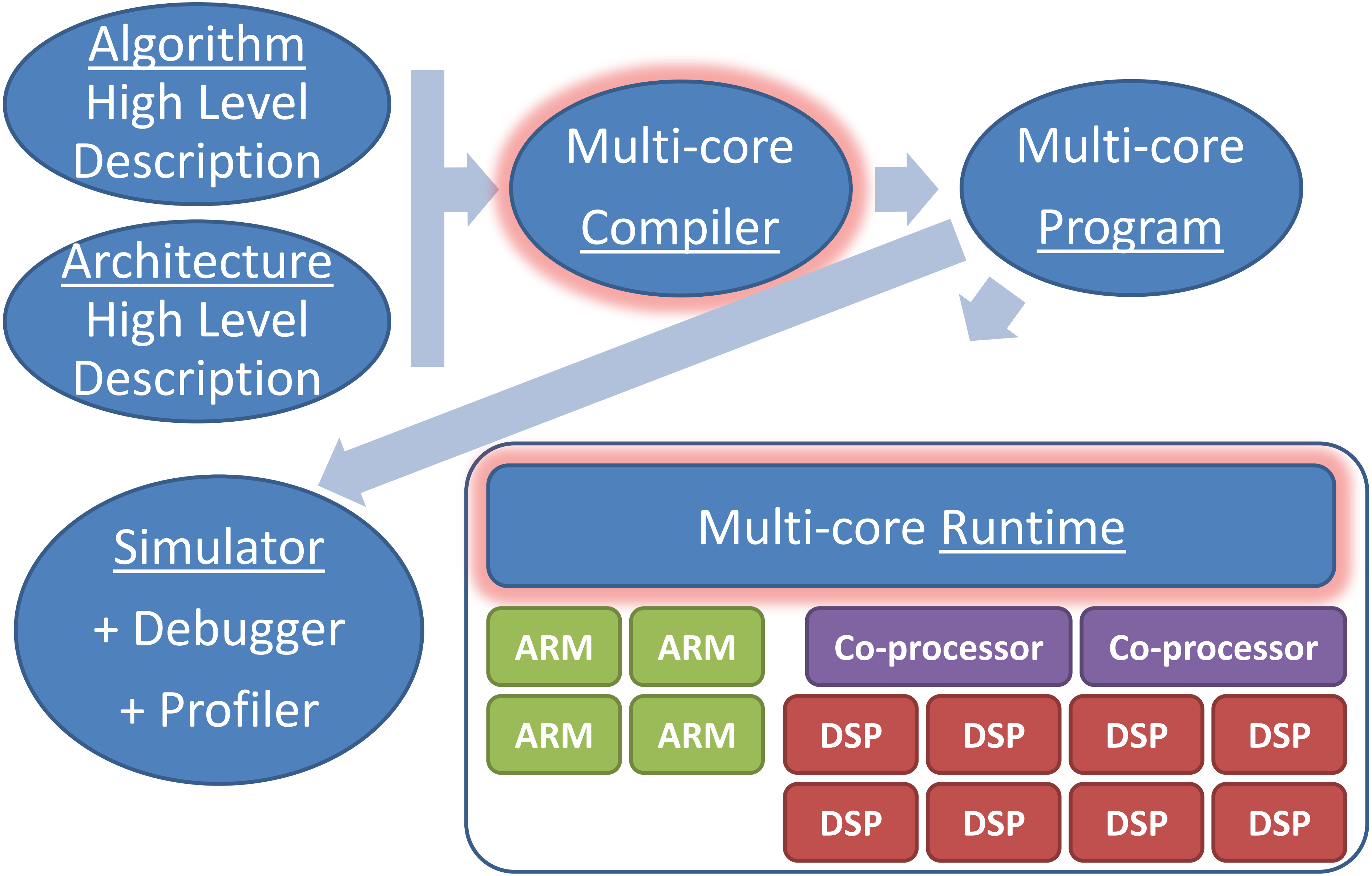
S-LAM of a Board with 2 TCI6488



Conclusion from Architecture Part

- **Architectures become more an more complex to program**
  - Parallelism rises at instruction and task level
  - Heterogeneity rises
  - Performance necessitates a correct use of DMA, cache, IPC

Multi-core DSP Programming



Automatic porting of Multicore DSP Applications

- Parallelism Laws
- Multicore Scheduling
- Multicore Tools

  Overview

  IETR Tools

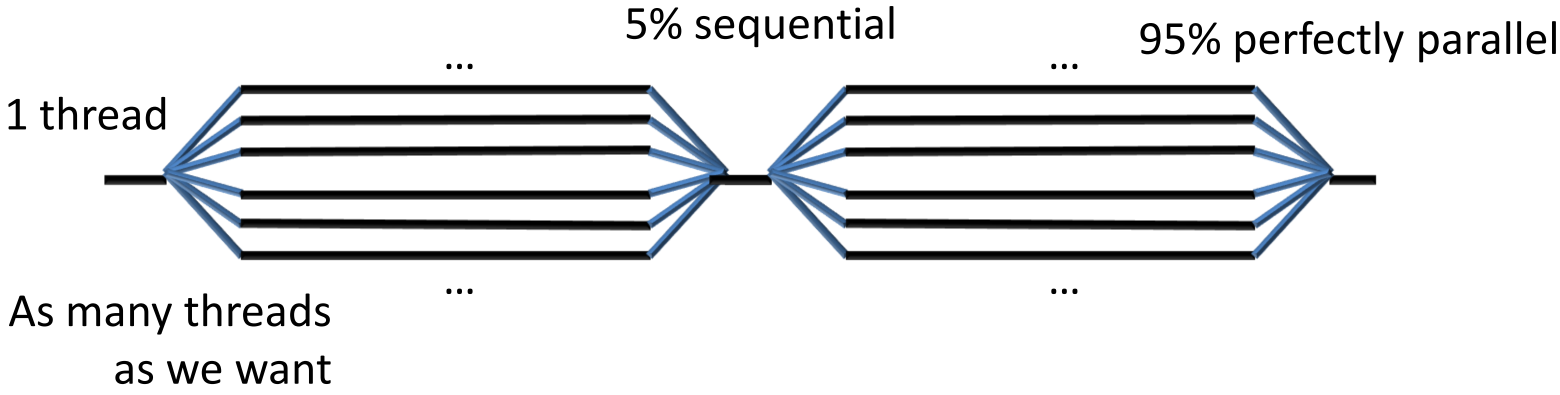
Institut National des Sciences Appliquées

Parallelism Laws

Amdahl’s Law

- **Developped in 1967 by Gene Amdahl**

- **It gives a generic performance metric for applications**
  - Simplifying problem assumption: x% of the code is sequential, the rest is perfectly parallel
  - With 5% of sequential code, speedup is limited to 7.5 on 12 cores
  - Speedup refers to the acceleration brought by adding cores



5% sequential

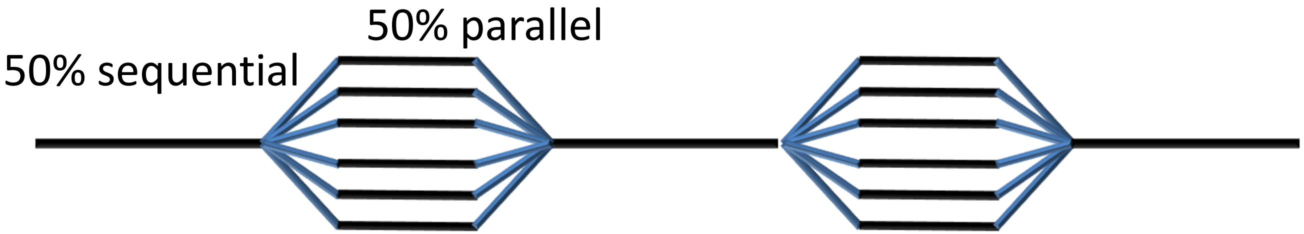
95% perfectly parallel

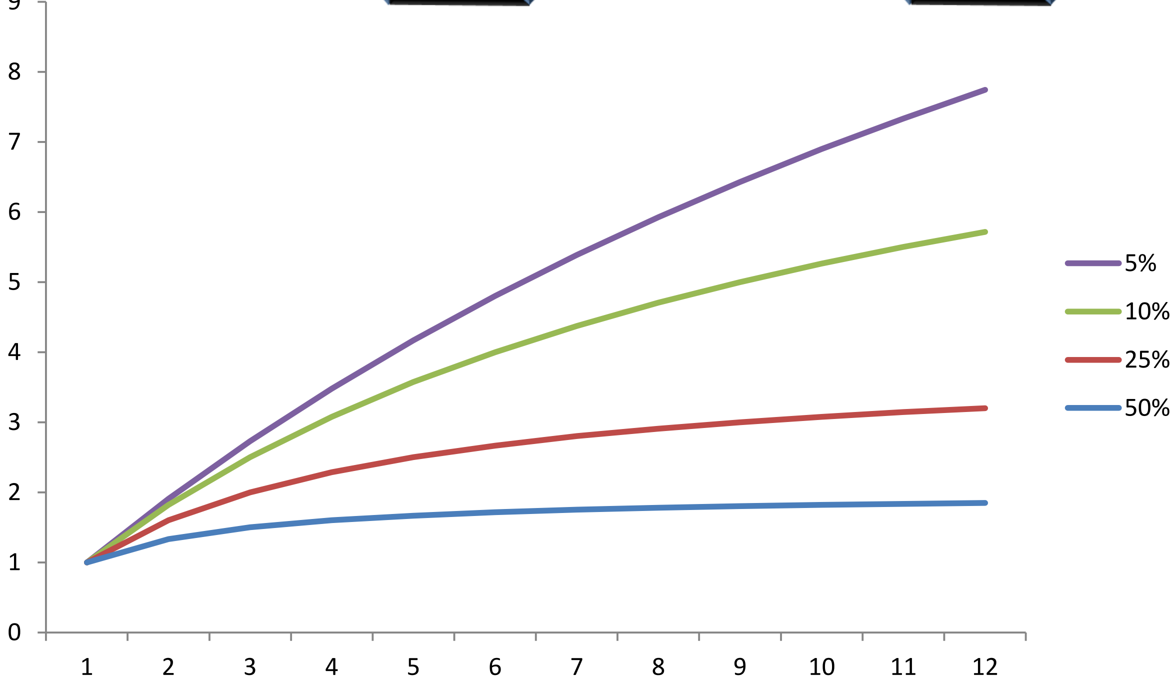
1 thread

As many threads
as we want

Amdahl's Law

- **For different parallel section percentages**
  - Simplifying problem assumption: x% of the code is sequential, the rest is perfectly parallel
  - With 50% of sequential code, speedup is limited to 1.84 on 12 cores





Amdahl’s Law

- **Amdahl’s law has brought many doubts on multicores**
  - It does not take into account inter-process communication that worsens the speedup

- **Why add more cores if the parallelism of applications limits speedups so much?**

Gustavson’s Law

- **Developed by John Gustavson in 1988**

- **With a different hypothesis, Gustafson has shown the limits of Amdahl’s law**

  Hypothesis: more cores imply more parallelism, the sequential section stays the same percentage of execution latency regardless the number of cores

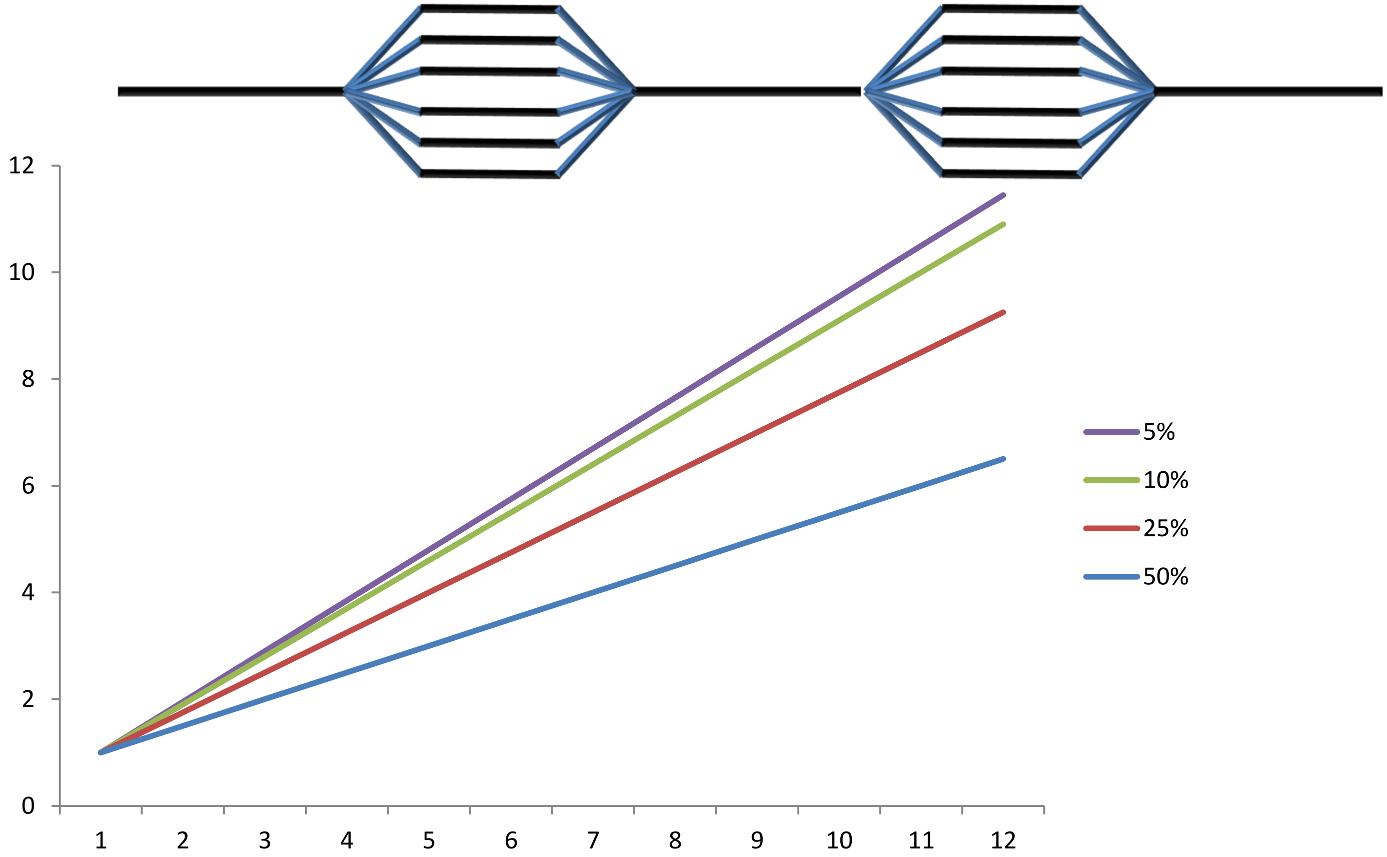
  in Amdahl’s law, the percentage tends to 100% because the parallel section time reduces and the sequential section stays unmodified

  With 5% of sequential code, speedup is limited to 11.89 on 12 cores

Gustavson’s Law (1988)

- **With a different hypothesis, Gustafson has shown the limits of Amdahl’s law**

Hypothesis: more cores imply more parallelism, the sequential section stays the same percentage of execution latency regardless the number of cores (in Amdahl’s law, it tends to 100%)

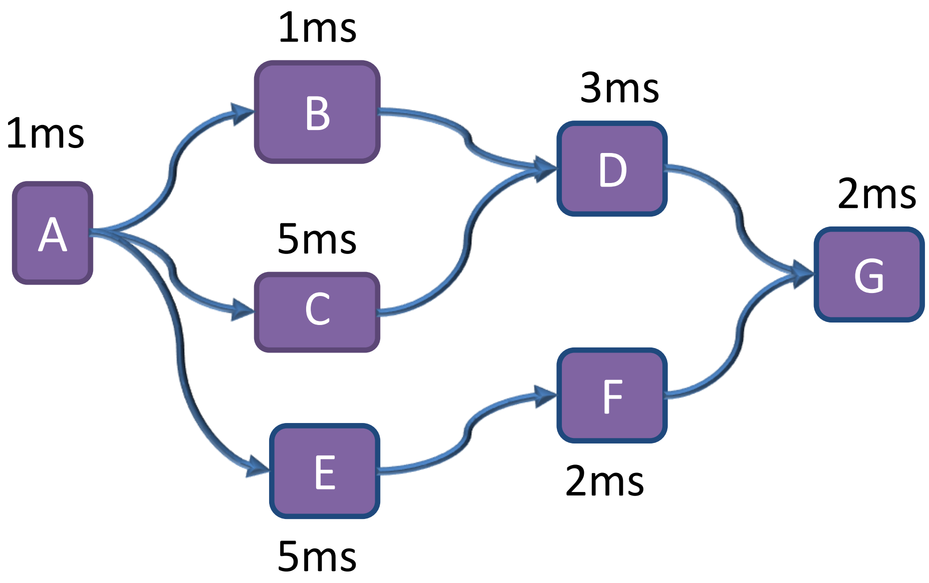


Dataflow Speedup

- **The maximum speedup of dataflow execution car be computed**

  It is limited by the critical path length

  Example: ignoring communication times



Institut National des Sciences Appliquées

Dataflow Speedup

- **The maximum speedup of dataflow execution car be computed**

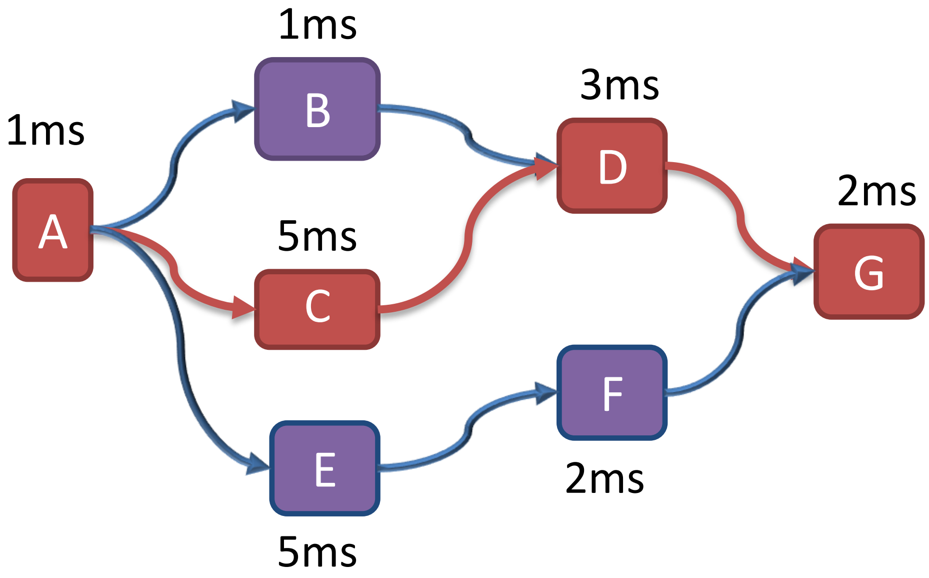
  It is limited by the critical path length

  Example: ignoring communication times

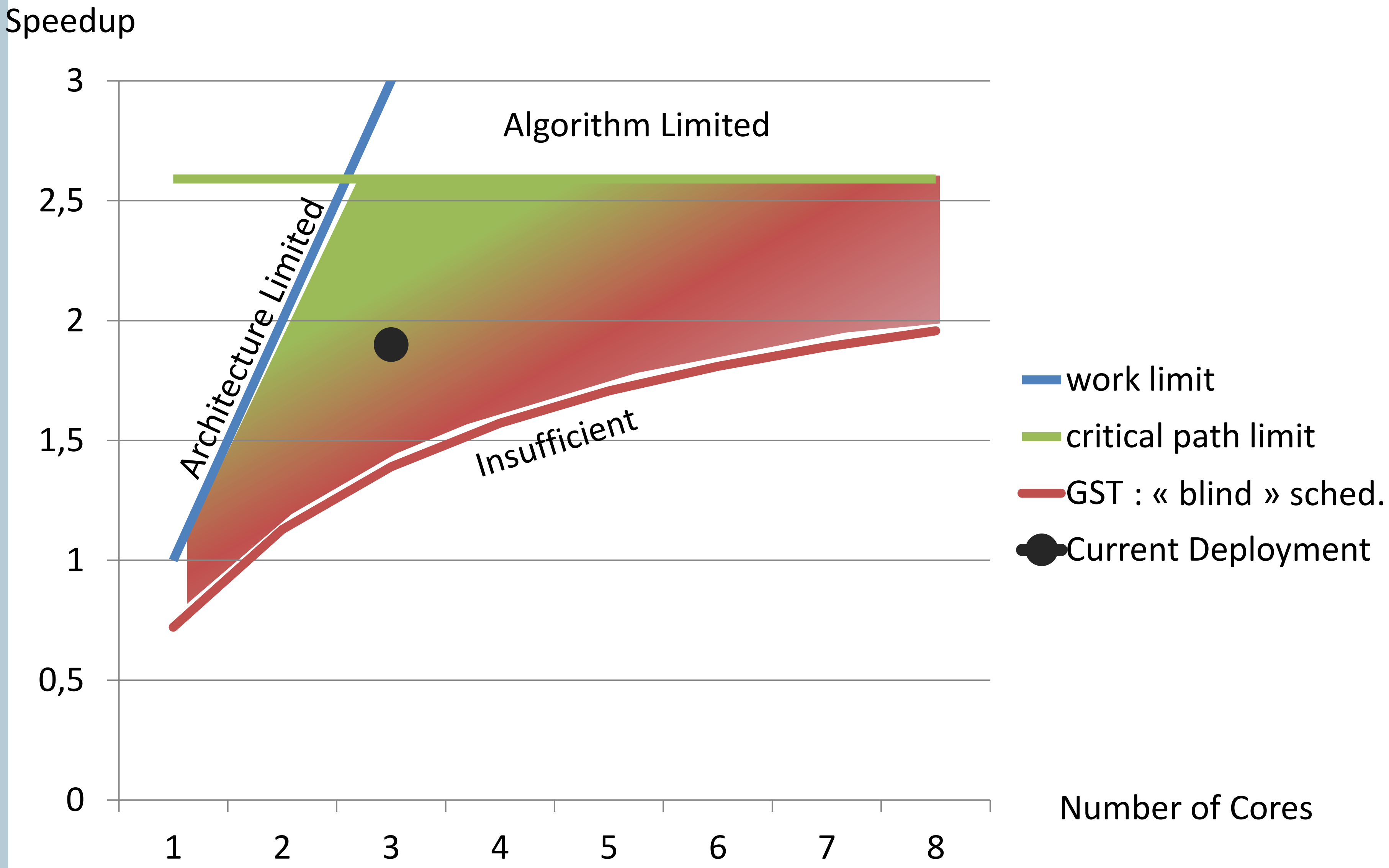
  critical path length = 1 + 5 + 3 + 2 + = 11ms

  work = 19 ms

  max speedup = 19 / 11 = 1.72



Preesm Speedup Assessment Chart



Preesm Speedup Assessment Chart

- **Speedup Assessment Chart Limitations**

  The speedup assessment chart considers only latency

  →No pipelining is taken into account

  All cores are considered identical in the chart (main operator)

  All communications have the same speed (main communication node)

- **How to add speedup**

  Redescribe the application to find more parallelism

  Add initial tokens (delays) to pipeline

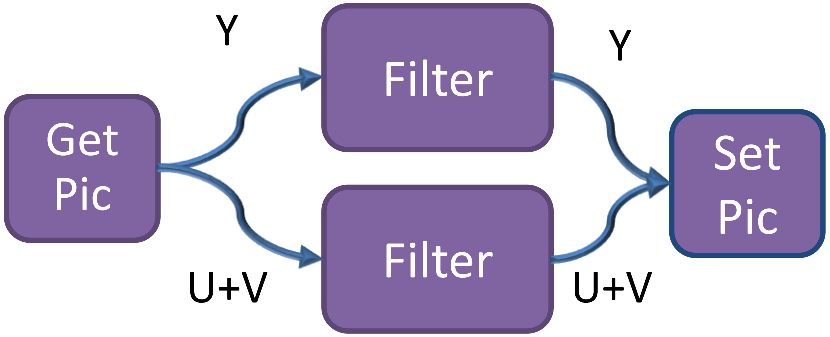
Task/Data/Pipeline Parallelism

- **Parallelism in a dataflow graph**

  There are 3 types of parallelism: task, data and pipeline parallelism

  Specific to stream processing applications

Data parallelism



Institut National des Sciences Appliquées

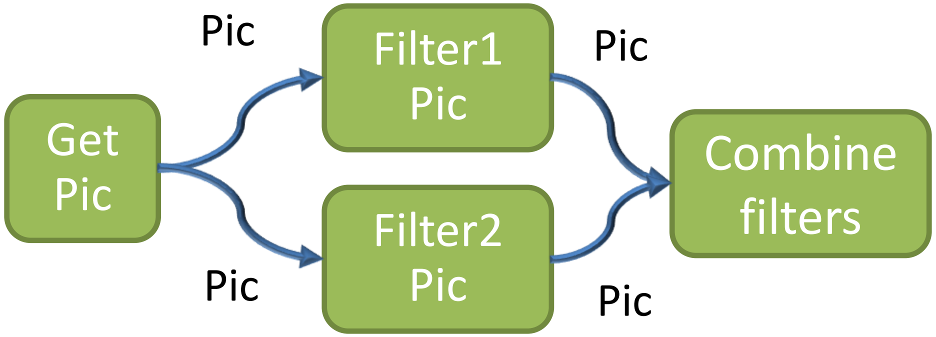
Task/Data/Pipeline Parallelism

- **Parallelism in a dataflow graph**

  There are 3 types of parallelism: task, data and pipeline parallelism

  Specific to stream processing applications

Task parallelism



Institut National des Sciences Appliquées

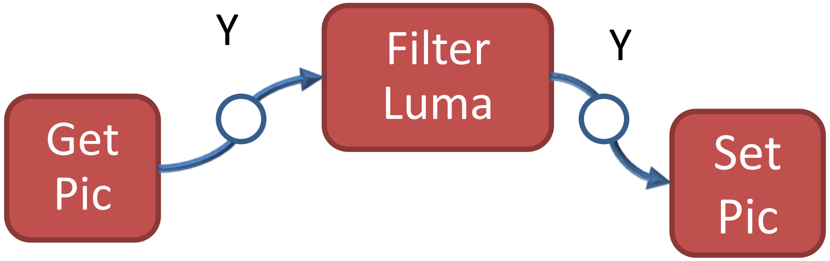
Task/Data/Pipeline Parallelism

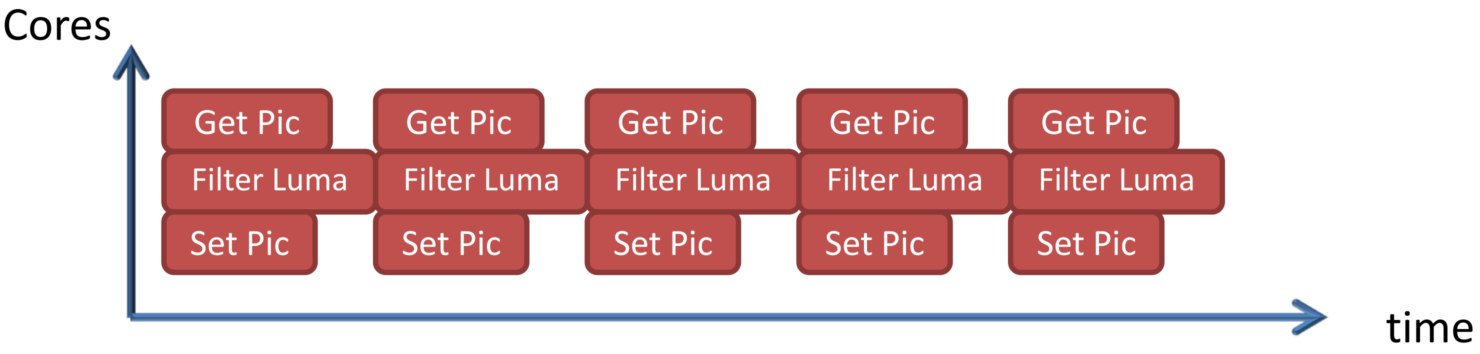
- **Parallelism in a dataflow graph**

  There are 3 types of parallelism: task, data and pipeline parallelism

  Specific to stream processing applications

Pipeline parallelism

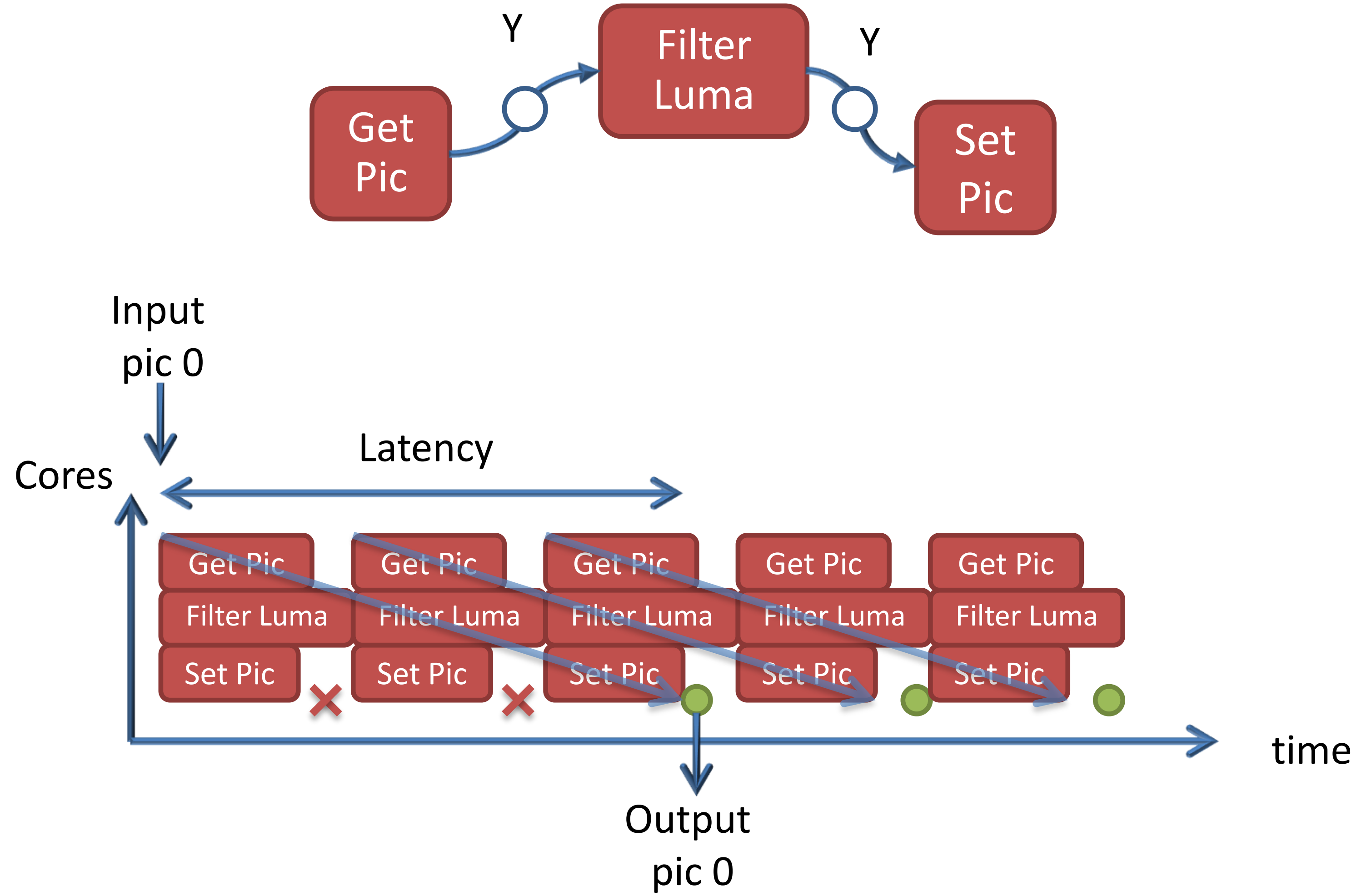




Task/Data/Pipeline Parallelism

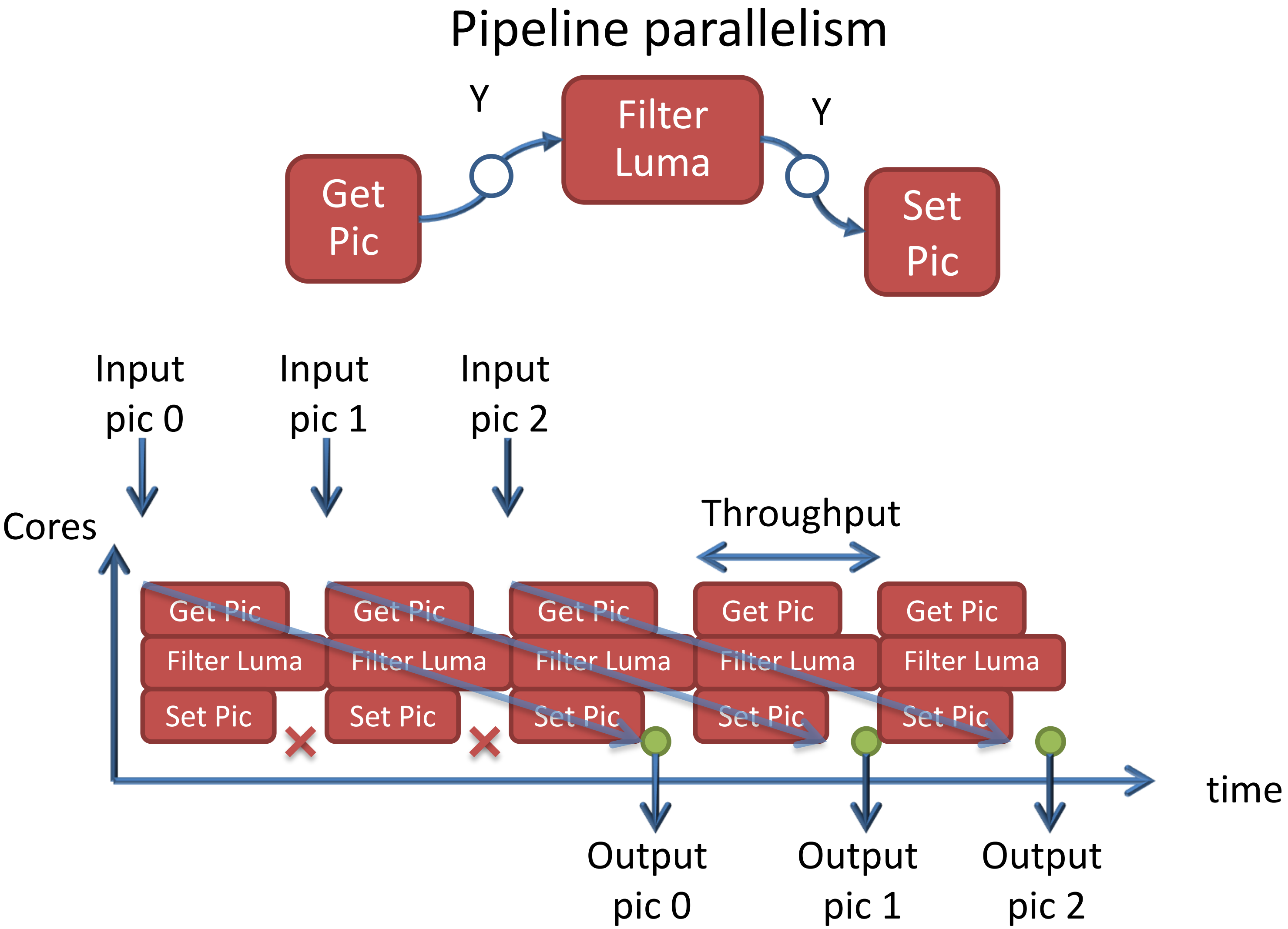
- **Parallelism in a dataflow graph**

Pipeline parallelism



Task/Data/Pipeline Parallelism

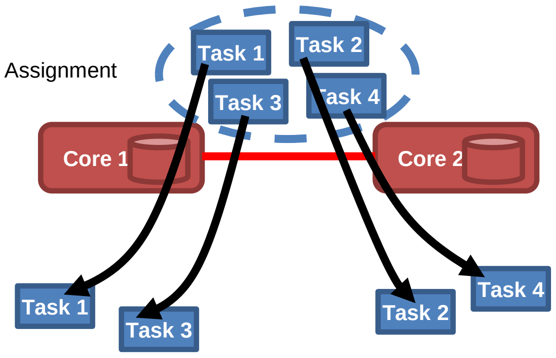
- **Parallelism in a dataflow graph**

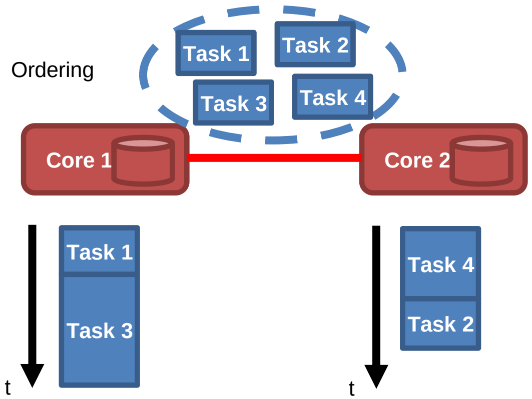


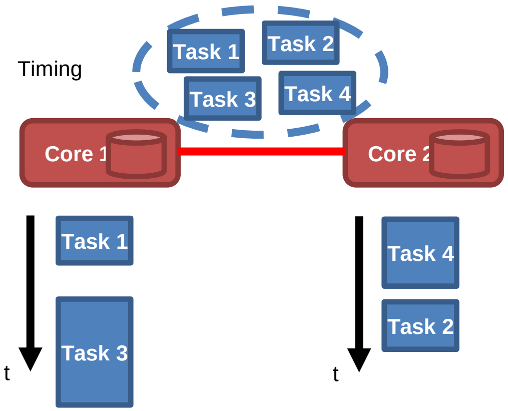
Institut National des Sciences Appliquées

Multicore Scheduling

Scheduling Strategies







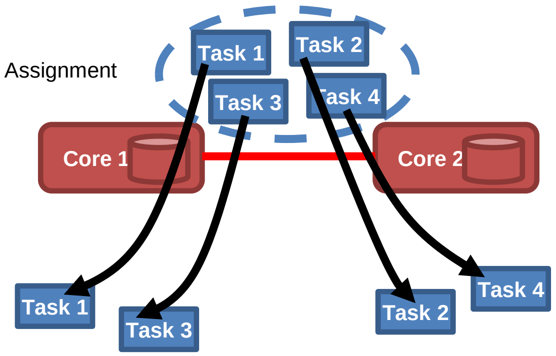
Scheduling Strategies

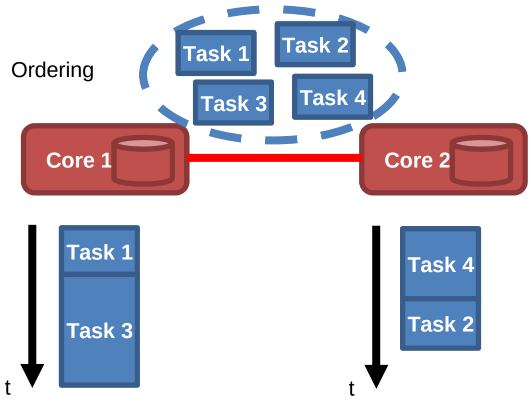
More adaptivity

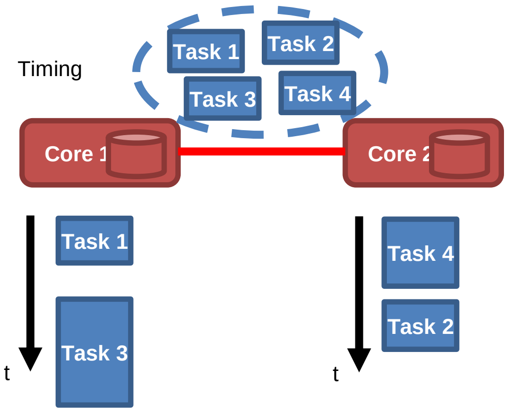
	assignment	ordering	Timing
fully dynamic	run	run	run
static-assignment	compile	run	run
self-timed	compile	compile	run
fully static	compile	compile	compile

EA Lee *Scheduling strategies for multiprocessor real-time DSP*

More performance







Scheduling Strategies

↑ More adaptivity

	assignment	ordering	Timing
fully dynamic	run	run	run
static-assignment	compile	run	run
self-timed	compile	compile	run
fully static	compile	compile	compile

EA Lee *Scheduling strategies for multiprocessor real-time DSP*

More performance ↓

Heterogeneous Multicore Scheduling

- **Assignment, ordering and timing**

  Part of « Operational Research »

  →How to organize a company

  →How to organize a project (Gantt chart…)

  →How to take decisions in general

- **NP hard problem**

  the verification that a possible solution of the problem is valid can be computed in polynomial time (verifying that a schedule is valid)

  no polynomial time algorithm for NP-complete problems is known and it is likely that none exists.

  When the problem grows (for example, the number of cores or actors), solving it is becoming more complex exponentially.

M. R Garey and D. S. Johnson “Computers and Intractability”

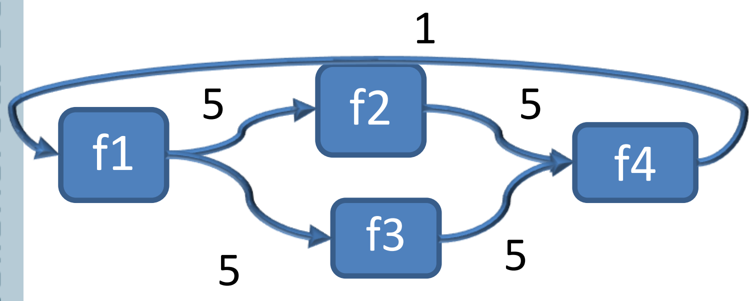
Heterogeneous Multicore Scheduling

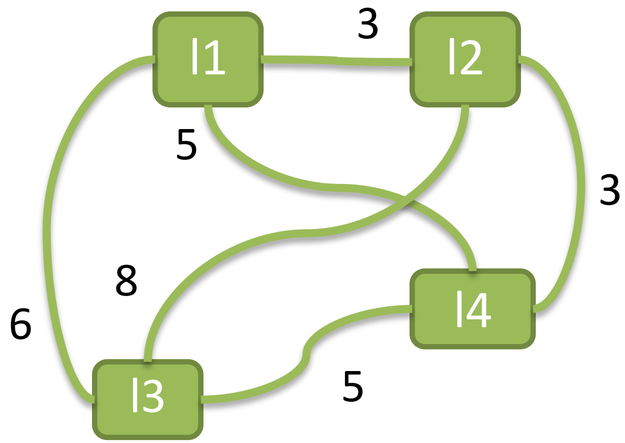
- **Multicore scheduling is equivalent to quadratic assignment NP hard problem**

  N facilities, each pair of facilities (f,g) associated to a flow of communication

  N locations to put the facilities, each pair of locations (l,m) associated to a distance

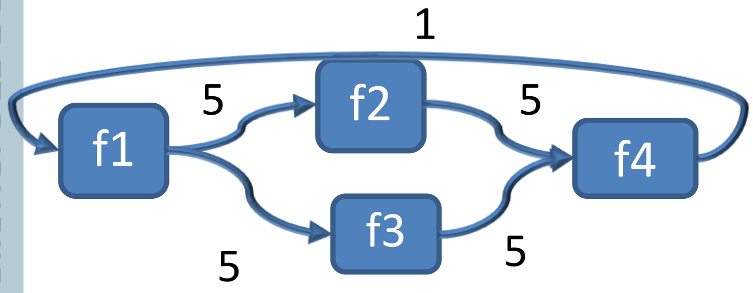
  → In which location (bijection) should we put each facility to minimize traffic (the sum of the distances multiplied by the corresponding flows)

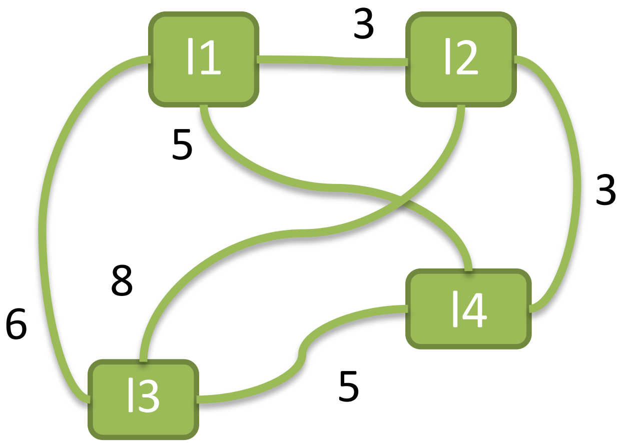




Heterogeneous Multicore Scheduling

- **The real problem is more complex**
  - M actors, N<M cores to put the facilities, each pair of locations (l,m) associated to a distance
  - Heterogeneity: actors have different costs on different cores
  - The objective function is not only communication minimization
    - latency, throughput…





Institut National des Sciences Appliquées

Heterogeneous Multicore Scheduling

- **The problems is solved by heuristics**

  Exhaustive methods are useless

  Heuristics explore only parts of the given problem

- **Many heuristics exist**

  list scheduling, greedy scheduling

  FAST scheduling (Y-K Kwok)

  Hybrid flow-shop scheduling (J. Boutellier)

  Meta-heuristics (genetic algorithms, ant colonies…)

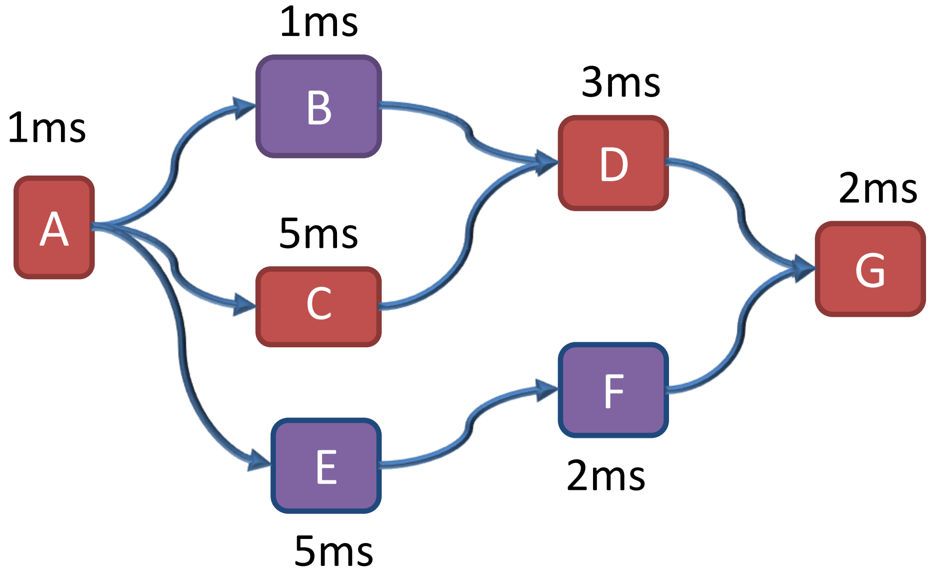
  …

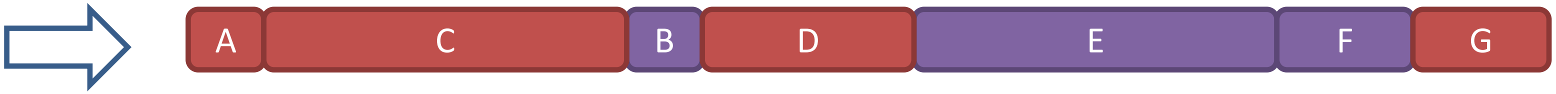
- **It is not possible to predict the quality of the result of a heuristic**

  But models should contain enough information to take decisions

Heterogeneous Multicore Scheduling

- **List scheduling**
  - Actors are scheduled in-order (topological order)
  - The core that can finish actor execution first wins





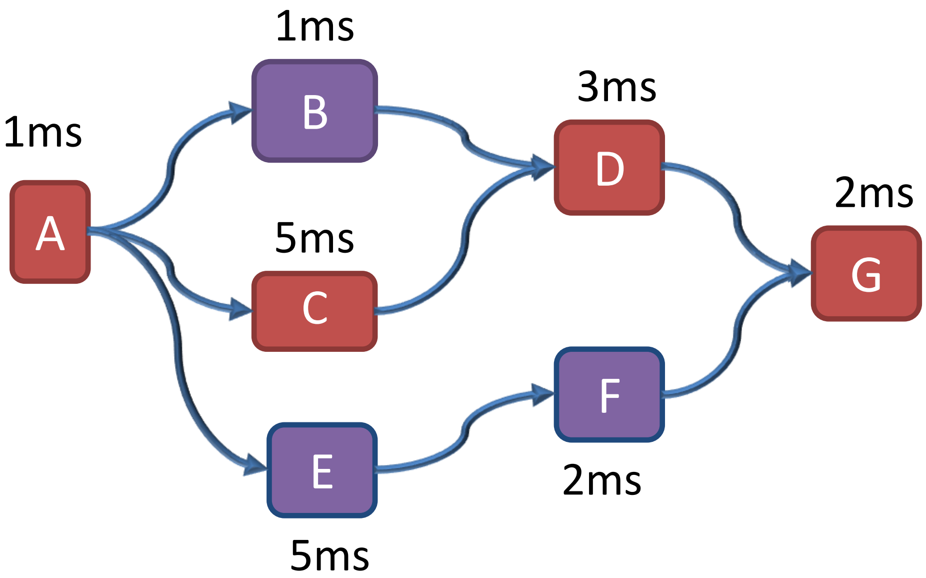
Institut National des Sciences Appliquées

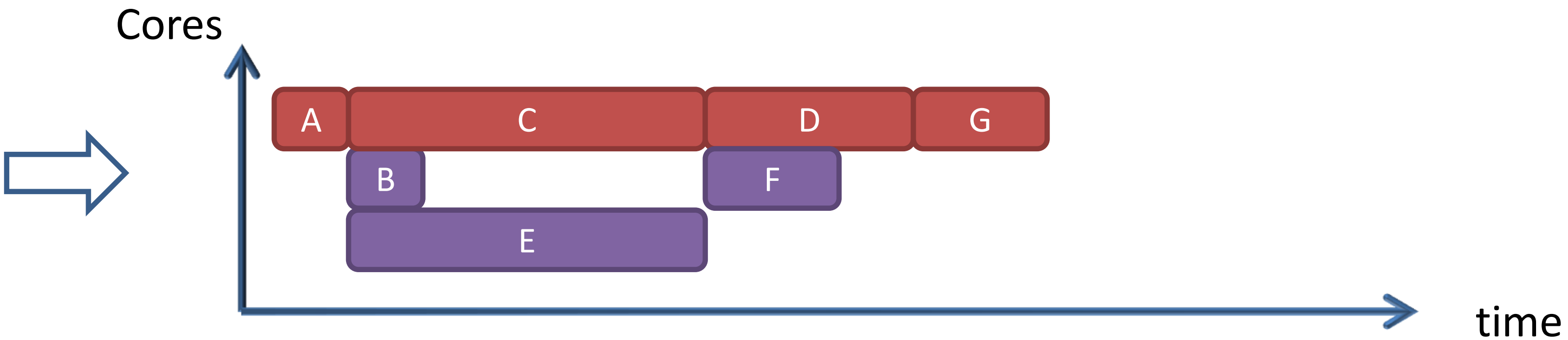
Heterogeneous Multicore Scheduling

- **List scheduling**

  Actors are scheduled in-order (topological order)

  The core that can finish actor execution first wins





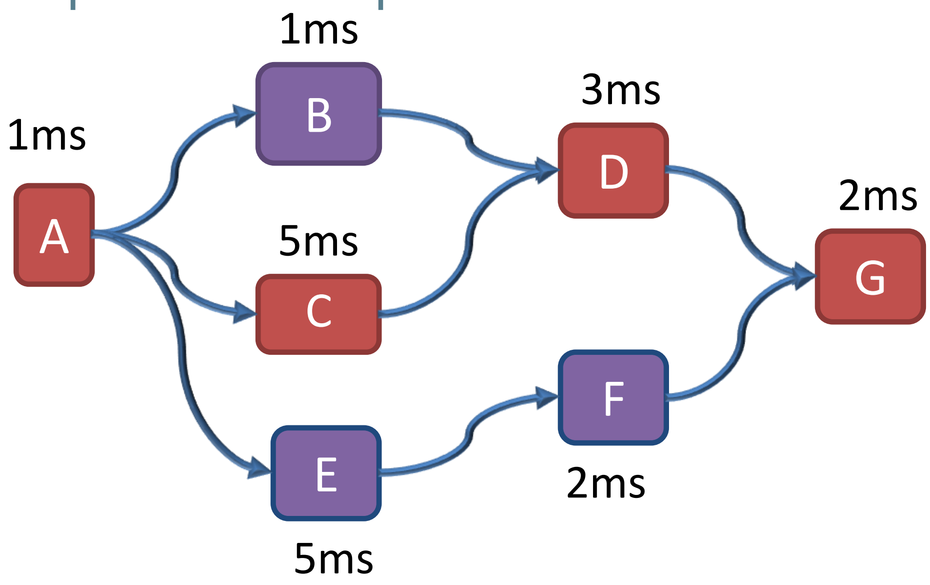
Institut National des Sciences Appliquées

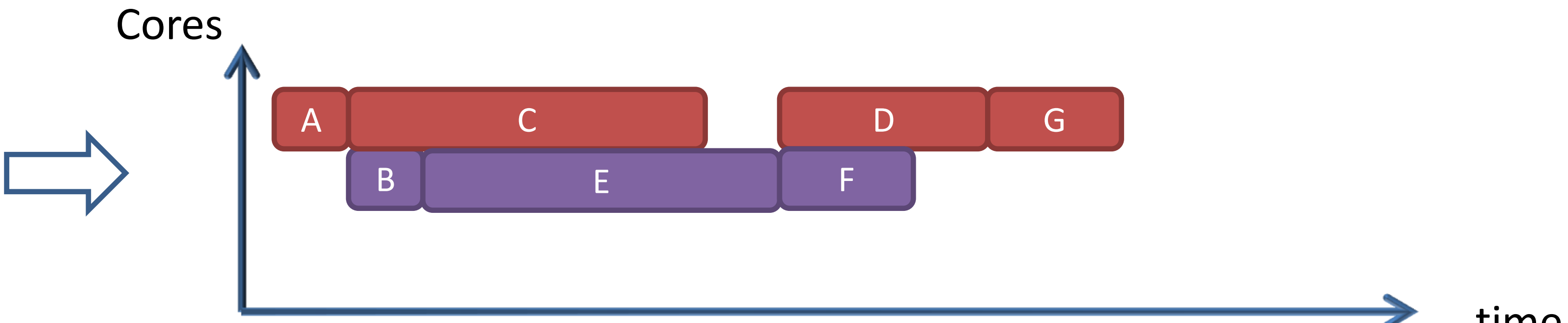
Heterogeneous Multicore Scheduling

- **Fast scheduling**

  Actors moved around to improve cost function (load balancing…)

  Critical path is more protected



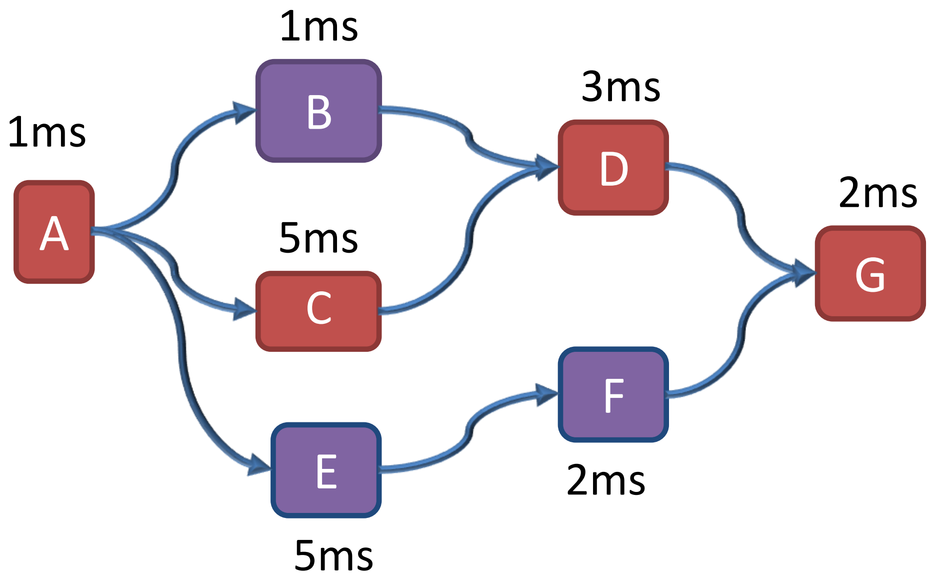


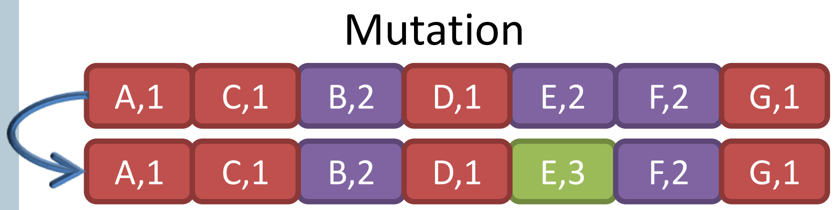
Institut National des Sciences Appliquées

Heterogeneous Multicore Scheduling

- **Genetic algorithm**

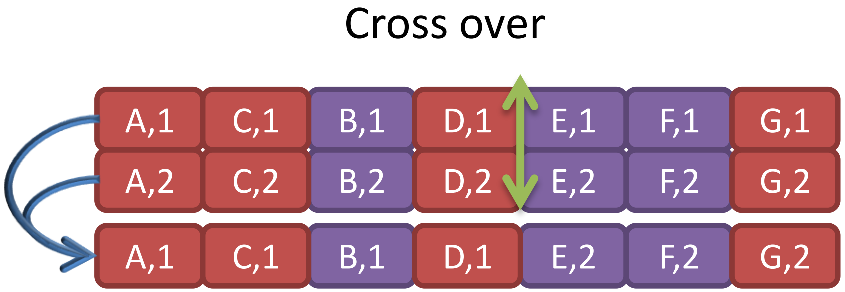
  Several mapping solutions are kept, they undergo mutations and cross-overs and worst solutions are regularly removed





Mutation

A,1	C,1	B,2	D,1	E,2	F,2	G,1
A,1	C,1	B,2	D,1	E,3	F,2	G,1



Cross over

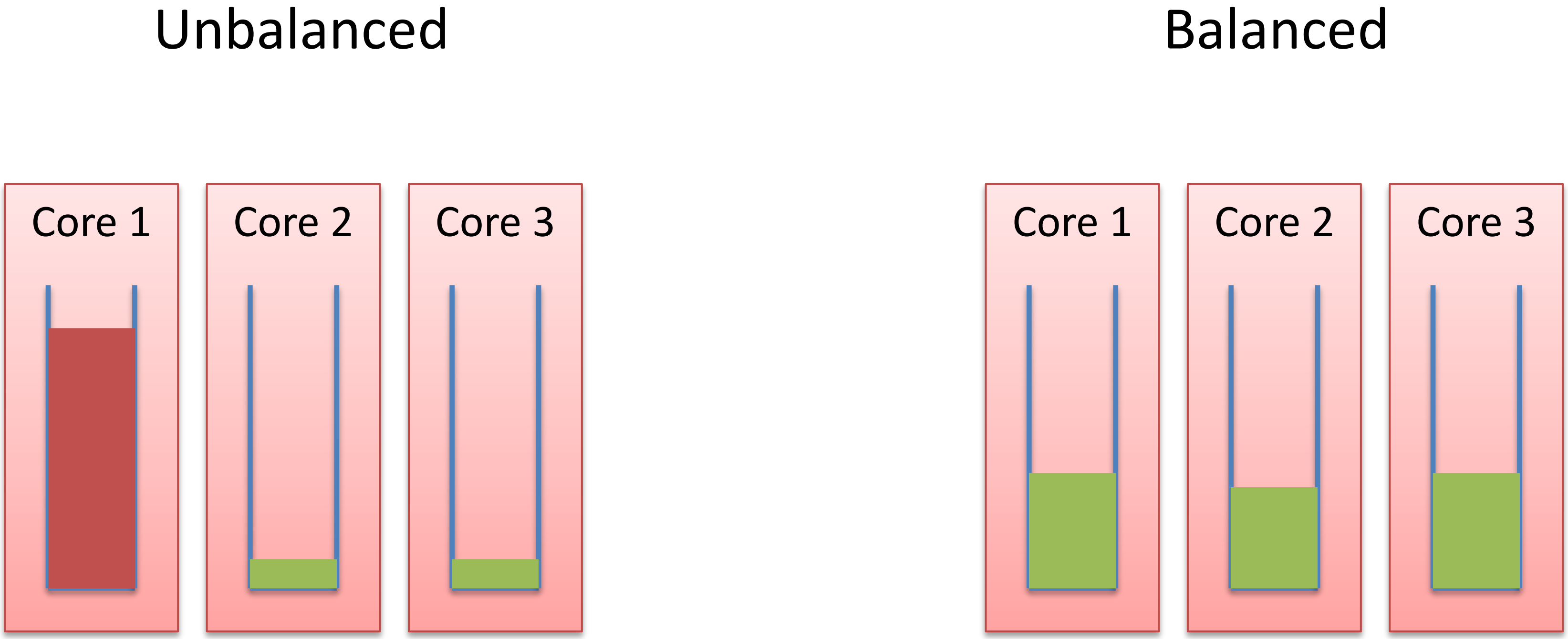
A,1	C,1	B,1	D,1	E,1	F,1	G,1
A,2	C,2	B,2	D,2	E,2	F,2	G,2
A,1	C,1	B,1	D,1	E,2	F,2	G,2

Load balancing

- **If no information is available on task / actor behavior**

  parallelism is brought by balancing the loads

  Execution is managed by task/job/actor queuing



Institut National des Sciences Appliquées

Execution Schemes

- **Load balancing without task behavior prediction**

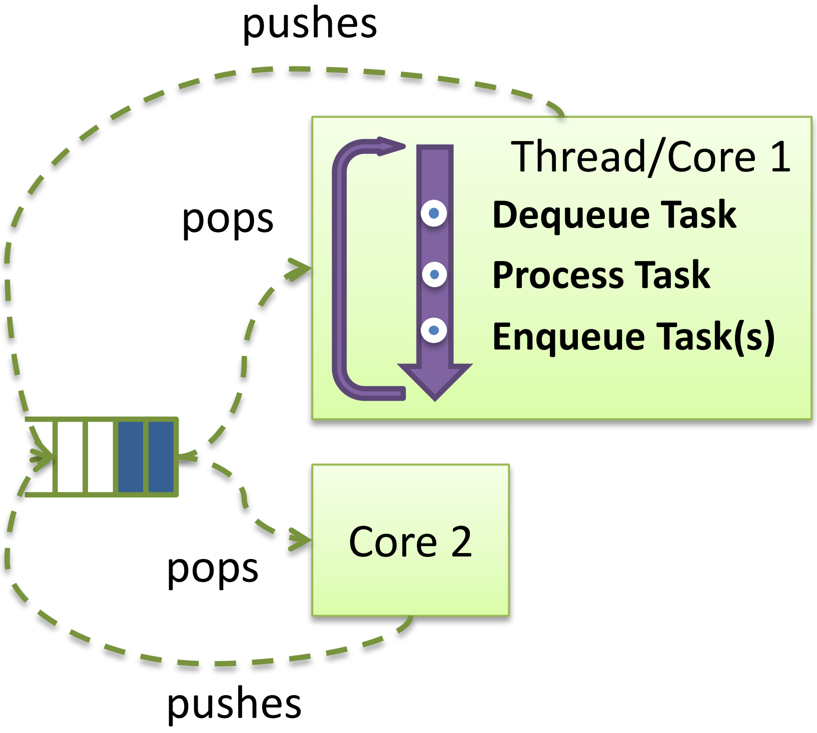
  Implemented over multi-threading

  Great freedom in thread creation

  The shared task queue becomes the bottleneck

Work-queueing

(Apple Grand Central Dispatch, OpenMP)



Institut National des Sciences Appliquées

Execution Schemes

- **Load balancing without task behavior prediction**

  Implemented over multi-threading

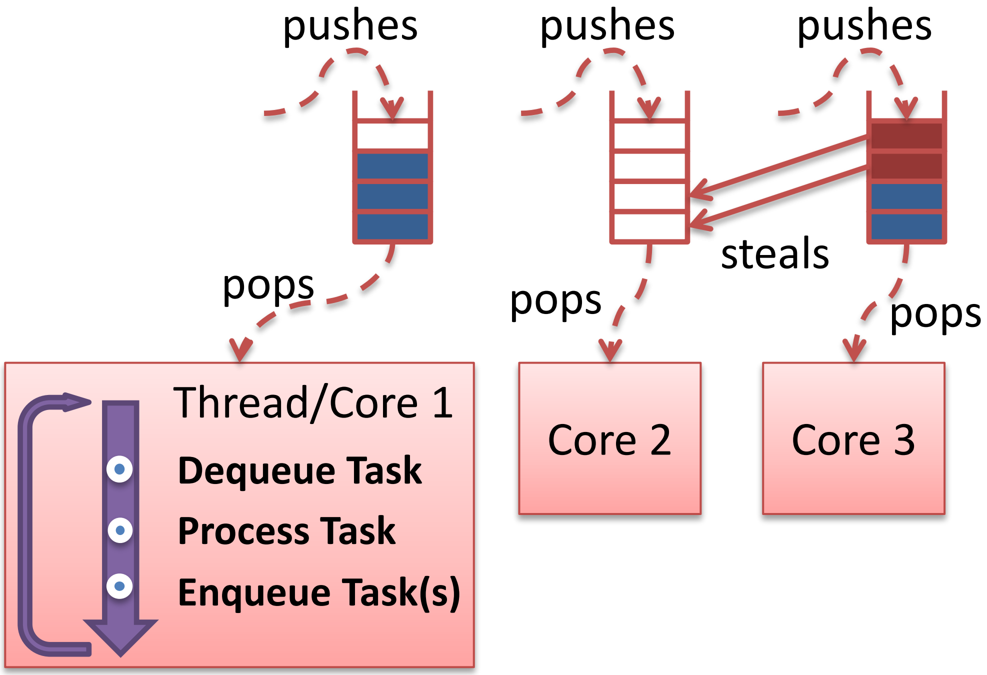
  One task queue per core:

  No more bottleneck

  Hard to predict performance

Job stealing

(Cilk, Intel Threading Building Blocks)



Execution Schemes

- **Load balancing without task behavior prediction**

  Implemented over multi-threading

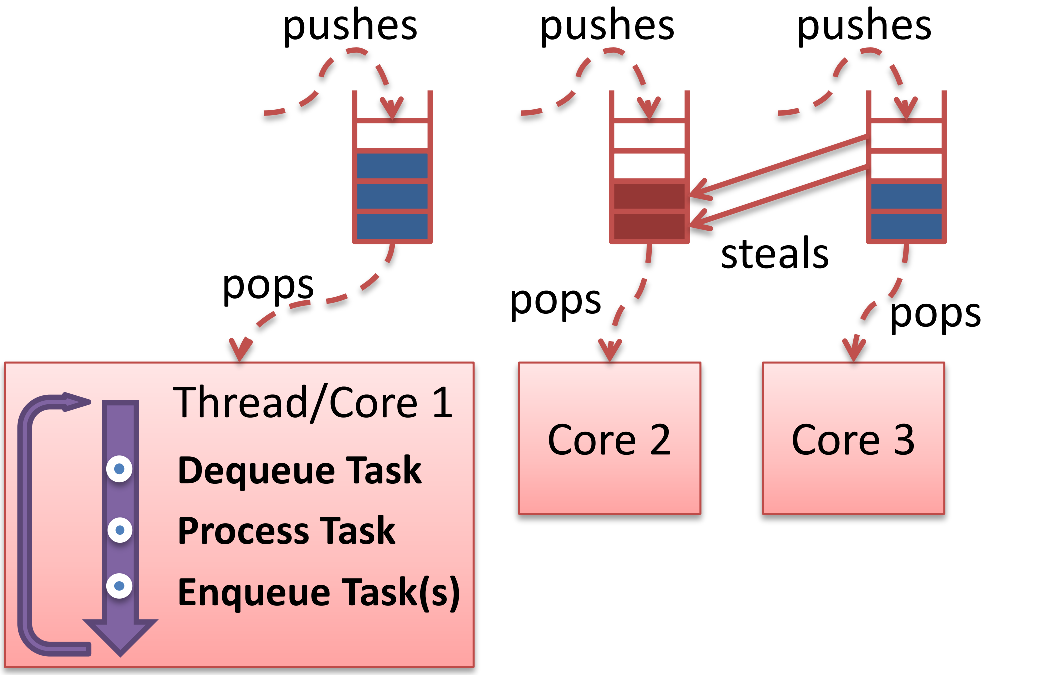
  One task queue per core:

  No more bottleneck

  Hard to predict performance

Job stealing

(Cilk, Intel Threading Building Blocks)



Thread/Core 1

Dequeue Task

Process Task

Enqueue Task(s)

Core 2

Core 3

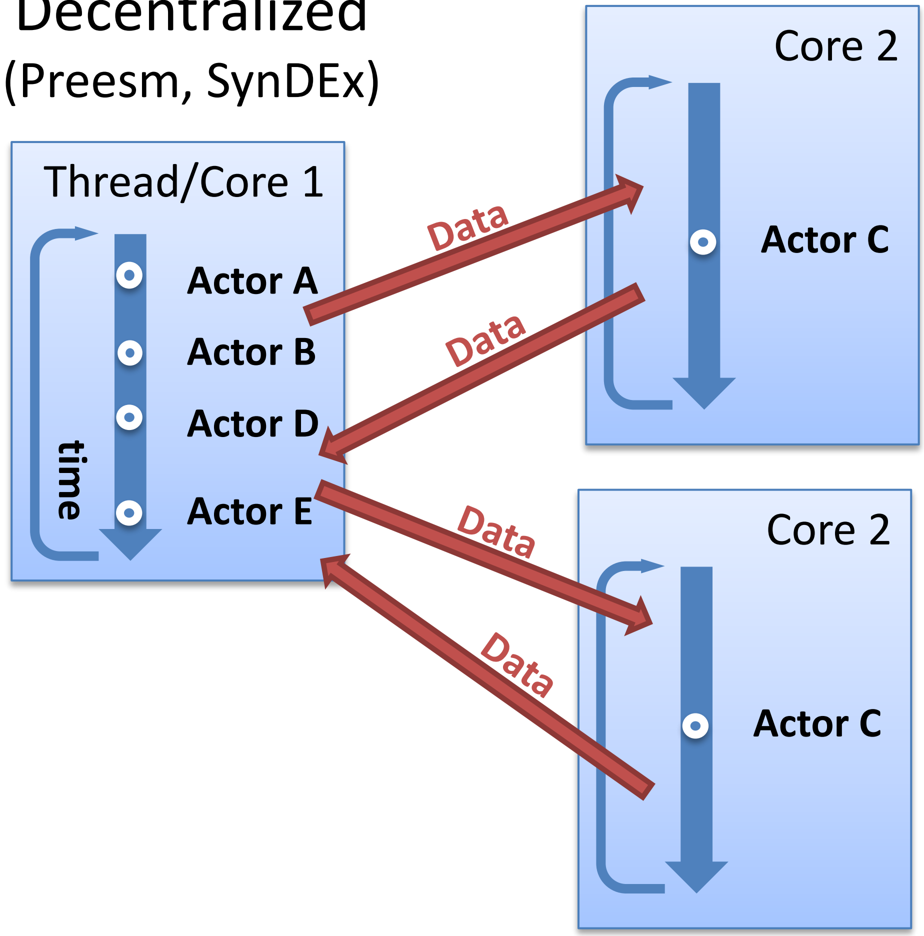
Execution Schemes

- **Scheduling with task behavior prediction**

  No adaptivity to algorithm modifications

  No decision overhead

Decentralized
(Preesm, SynDEx)



Institut National des Sciences Appliquées

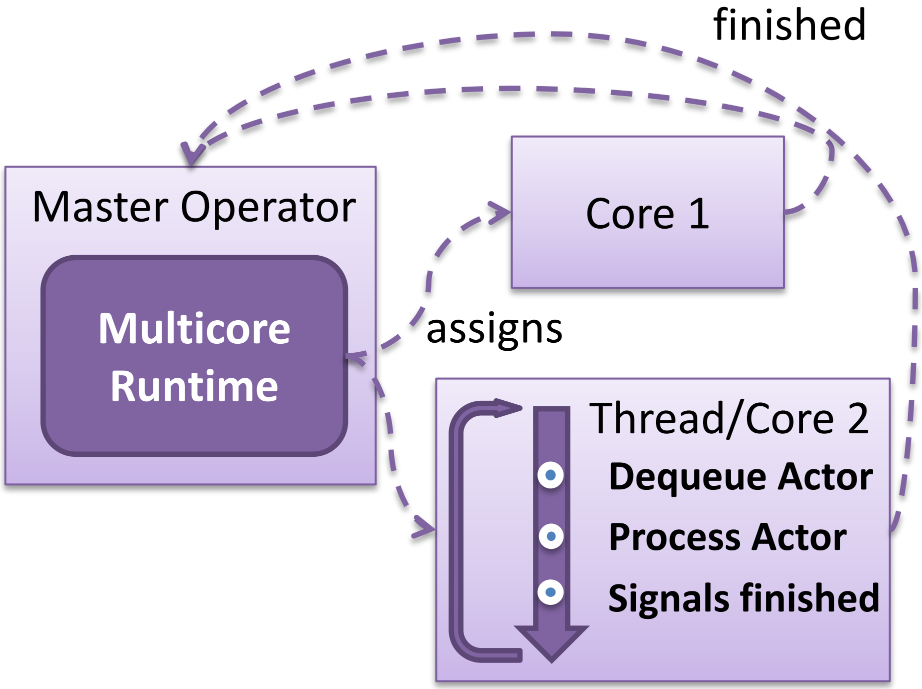
Execution Schemes

- **Scheduling with task behavior prediction**

Adaptivity to algorithm variations

Master core can become a bottleneck

Master/Slave
(Compa Runtime)



Actors versus threads

- **Why not use threads and processes instead of actors?**

- **Threads share memory within a process**

  Multicore thread execution necessitates shared memory between cores

  Shared memory is increasingly costly when number of cores grows

  This method of parallelizing is showing its limits already with 8 cores

- **Threads are designed for resource sharing**

  Cores, memory…

  What we want is more resource combining

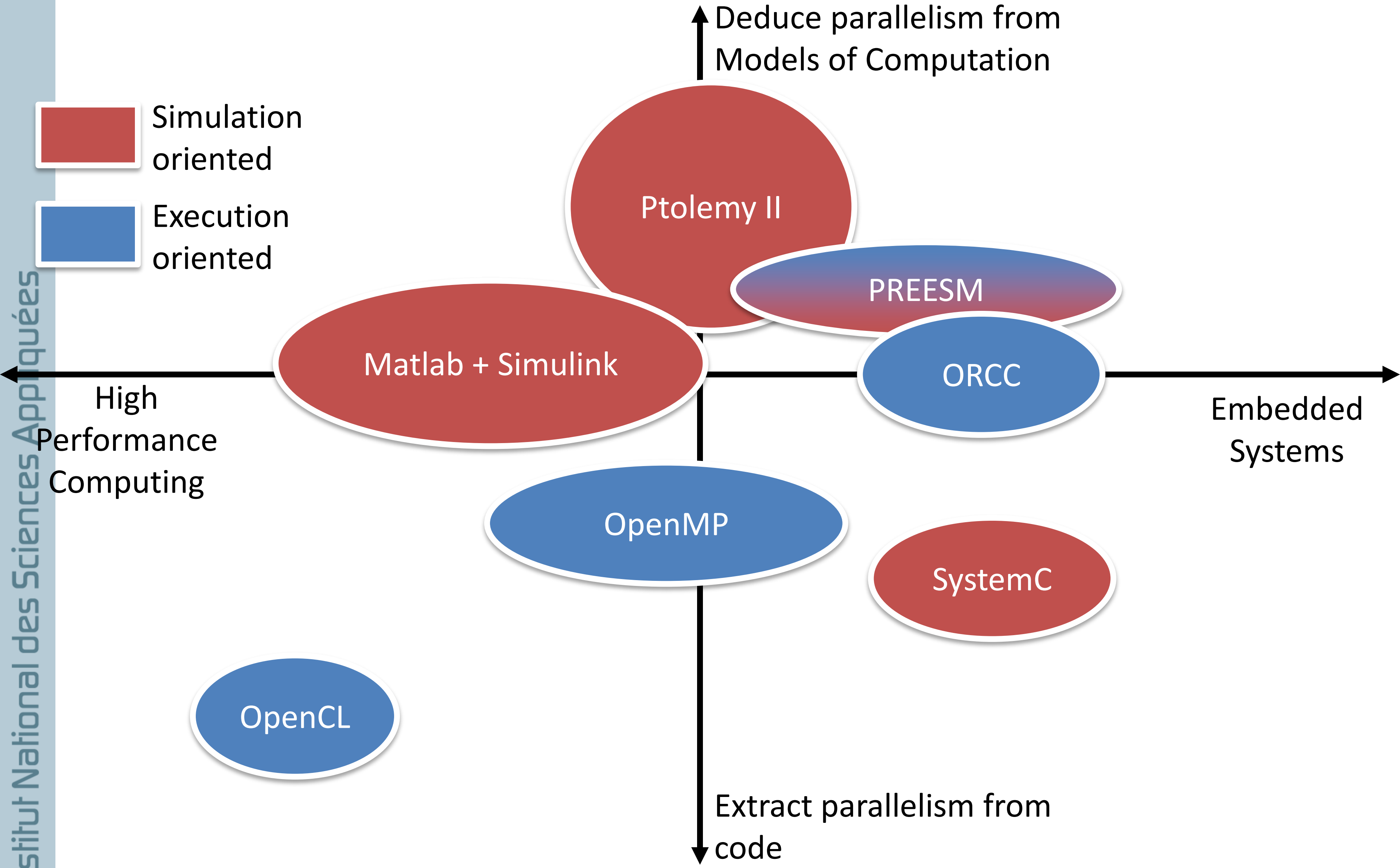
- **Actors are special types of processes**

  With firing rules, i.e. computation is triggered by available data

  Actors are dedicated to stream processing. Threads and processes can implement control-oriented code

Multicore Tools

Some Tools and Initiatives



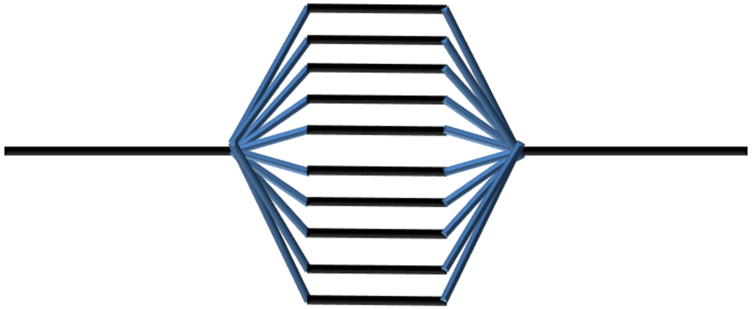
OpenMP

- **Implemented on GCC 4.2**
- **Implemented in TI compiler for Keystone I**

- **Adds pragmas (metadata) in C to tell the compiler which loops can be parallelized**

- **Example:**

```
#pragma omp parallel for
For(i=0; i<10; i++){
        T[i] = f(i);
}
```



The compiler knows that the iterations are independent (data parallelism in this case) and can be parallelized.

Institut National des Sciences Appliquées

Other Tools and Initiatives with Common Objectives

- **Model-Driven-Engineering (MDE)**
  - Starting from UML meta-model and meta-model transformation
  - Defines a whole methodology from specification to final system
  - UML profiles like UML Marte have been defined for that purpose
- **Synchronous languages**
  - Lustre, Signal, Esterel…
  - Specifying synchronizations between operators: close to dataflow
- **C extensions**
  - OpenMP, OpenCL, OpenACC, OpenHMPP, Cilk
  - Principal objective: a painless migration to coarse-grain parallelism
- **StreaMIT - MIT**
  - A language for stream processing, close to dataflow MoCs, and a compiler for massively parallel machiles

Other Tools and Initiatives with Common Objectives

- **Task queueing software frameworks**

  Intel Threading Building Blocks, Apple Grand Central Dispatch

  Not specifically oriented towards stream processing applications

- **Task queueing software/hardware frameworks**

  Open Event Machine and Multicore Navigator

  → Available on C6678

Preesm: Rapid Prototyping for Multi-core DSP

- **Eclipse-based Open Source Rapid Prototyping Tool**

- **Collaboration with Texas Instruments Nice**

  Communication Infrastructure Business Unit (CI BU)

  Rapid prototyping of base station algorithms (used for LTE)

- **Goals**

  Combine imperative actor (host) code and dataflow coordination code

  Latency, memory and energy study of embedded code execution

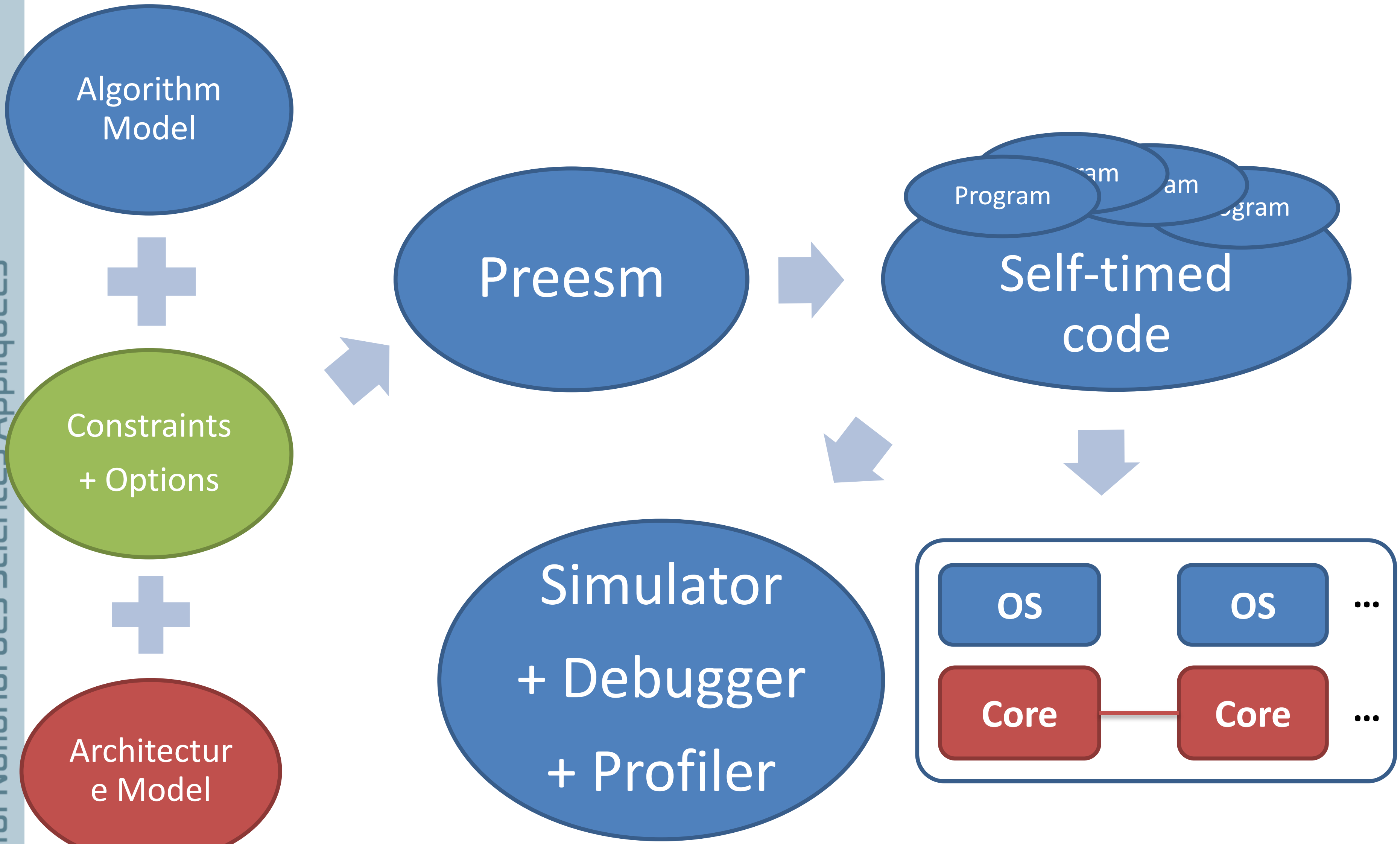
  Before target hardware is available

  With efficient automatic parallelization heuristics

  Generating static multi-core code

  Reuse legacy code

Rapid Prototyping using Preesm



Algorithm Model

Constraints

+ Options

Architectur e Model

Preesm

Program

ram

am

gram

Self-timed code

Simulator

+ Debugger

+ Profiler

OS

OS

…

Core

Core

…

Preesm: Rapid Prototyping for Multi-core DSP

- **Algorithm Model**

  Static Hierarchical SDF with C-like behaviour (IBSDF and PiSDF) Combined with C Actor Code

- **Architecture Model**

  System-Level Architecture Model (S-LAM) of heterogeneous architectures

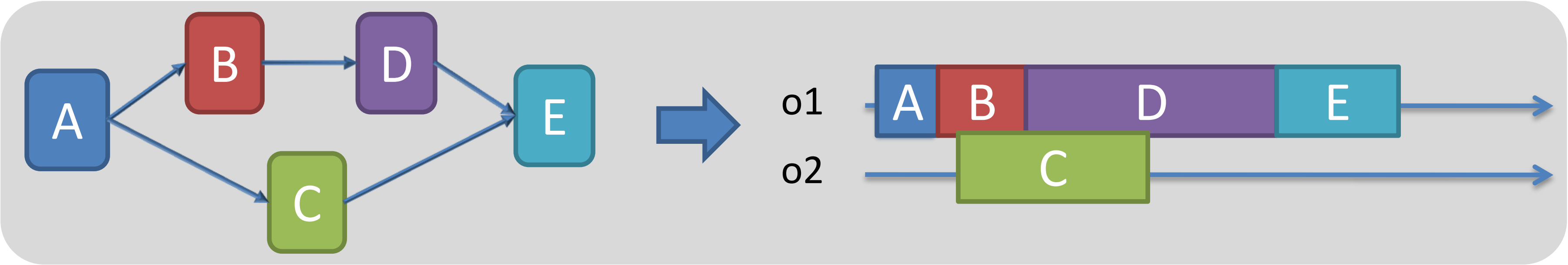
  Focuses on important contention points

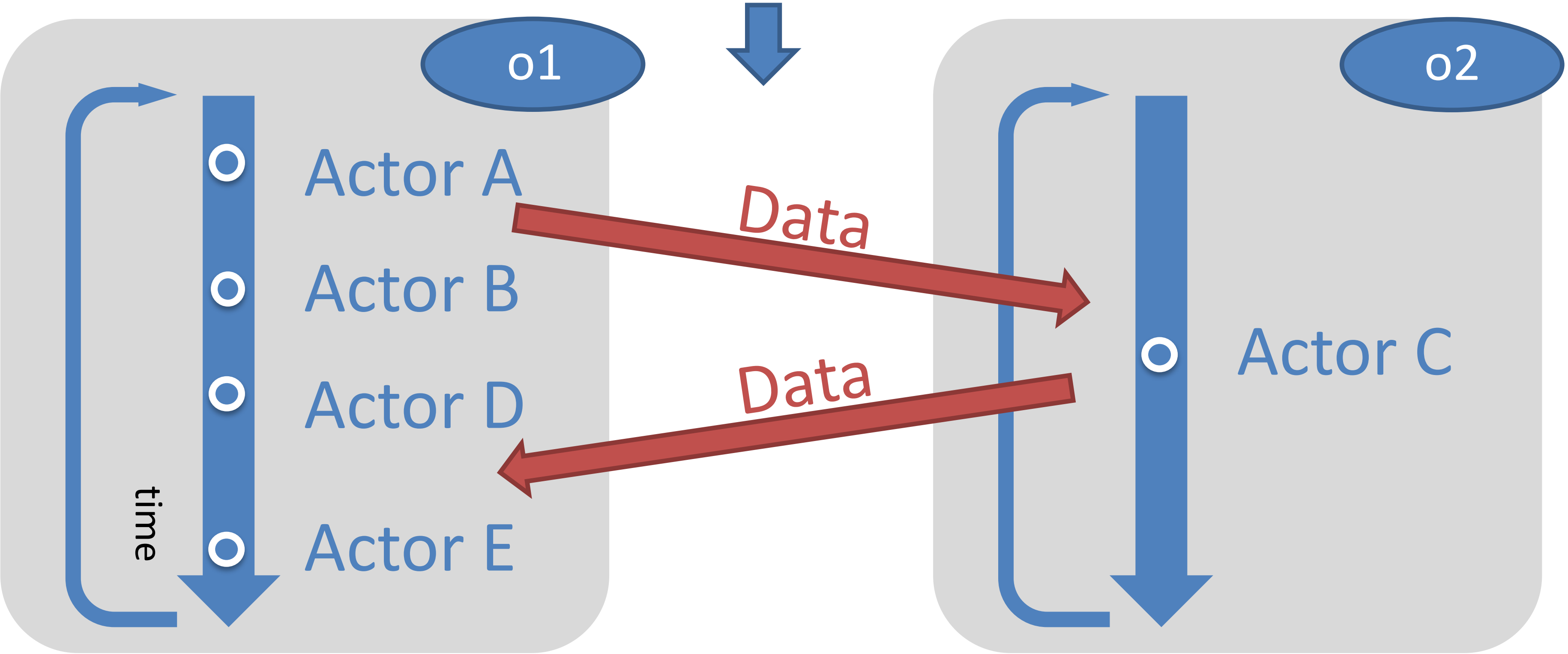
- **Prototyping**

  Shared memory / Message passing / DMA transfers /

  Load balancing enhancement

Static Code Generation : Self-Timed





General Conclusion

- **Applications and architectures are increasingly complex**

  Model-based system design helps at several design stages

- **To evaluate languages/models: focus on MoC**

  MoCs offer « pure » semantics, free of syntax

- **No one-fit-all solution to design Multicore DSP systems**

  Many solutions exist now, complex choices have to be made

Demo