

# NoC and Mixed-criticality Systems

---

**Mourad Dridi\*, Stéphane Rubini\*, Mounir Lallali\*, Frank Singhoff\*, Jean-Philippe Diguët+, Martha Johanna<sup>‡</sup>**

Lab-STICC, UMR CNRS 6285,

\*Université de Bretagne Occidentale

+Université de Bretagne Sud

<sup>‡</sup> Université Technique de Munich



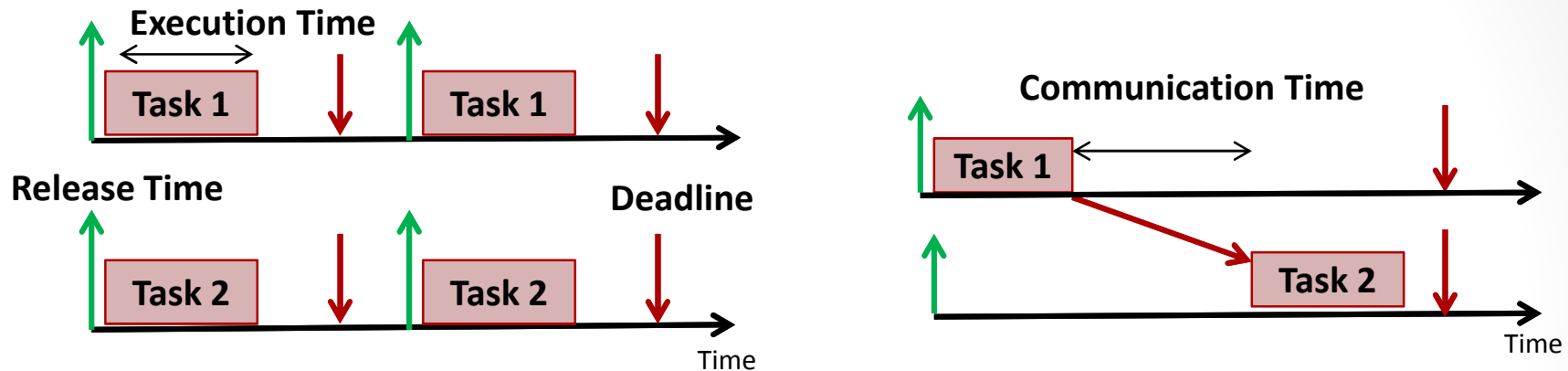
# Plan

- 1. Context**
2. DTFM : a Scheduling Analysis Model for NoC based Systems
3. DAS : a NoC Router for Mixed-Criticality Systems
4. Conclusion

# Context : Real-time System

- « *The correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced* » Stankovic, 1988.
  - Real Time applications are composed of software tasks
  - **Properties we look for:**
    - Timing constraints of tasks (e.g. deadline) must be enforced during execution
    - Timing behavior of tasks must be predictable: we must verify that temporal constraints are met prior execution.
- ➔ Schedulability analysis of the tasks

# Context : Mixed-Criticality Systems (MCS)



- High-Critical Real-Time applications
  - Have very strength task execution and communication requirements.
  - May have dramatic impact on human life, on the environment, ...
- Low-Critical Real-Time applications
  - Can tolerate some missed deadlines for the task execution and the communication.

# Context : Mixed-Criticality Systems (MCS)

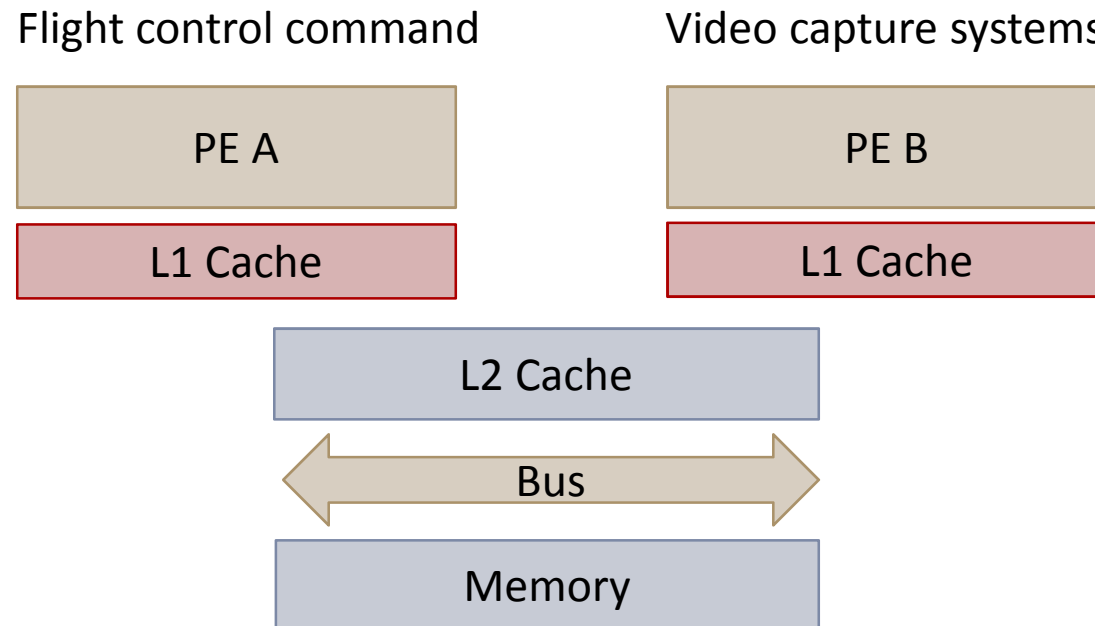
## Example of a Drone system :

- High-Critical Real-Time applications
  - ➔ Flight control command
  - Flows of small messages exchanged between tasks
  - Deadline must be met
- Low-Critical Real-Time applications
  - ➔ Video capture systems
  - Flows of large messages exchanged between tasks
  - May tolerate missed deadlines



➔ Assumptions of this work

# Context : Mixed-Criticality Systems (MCS)



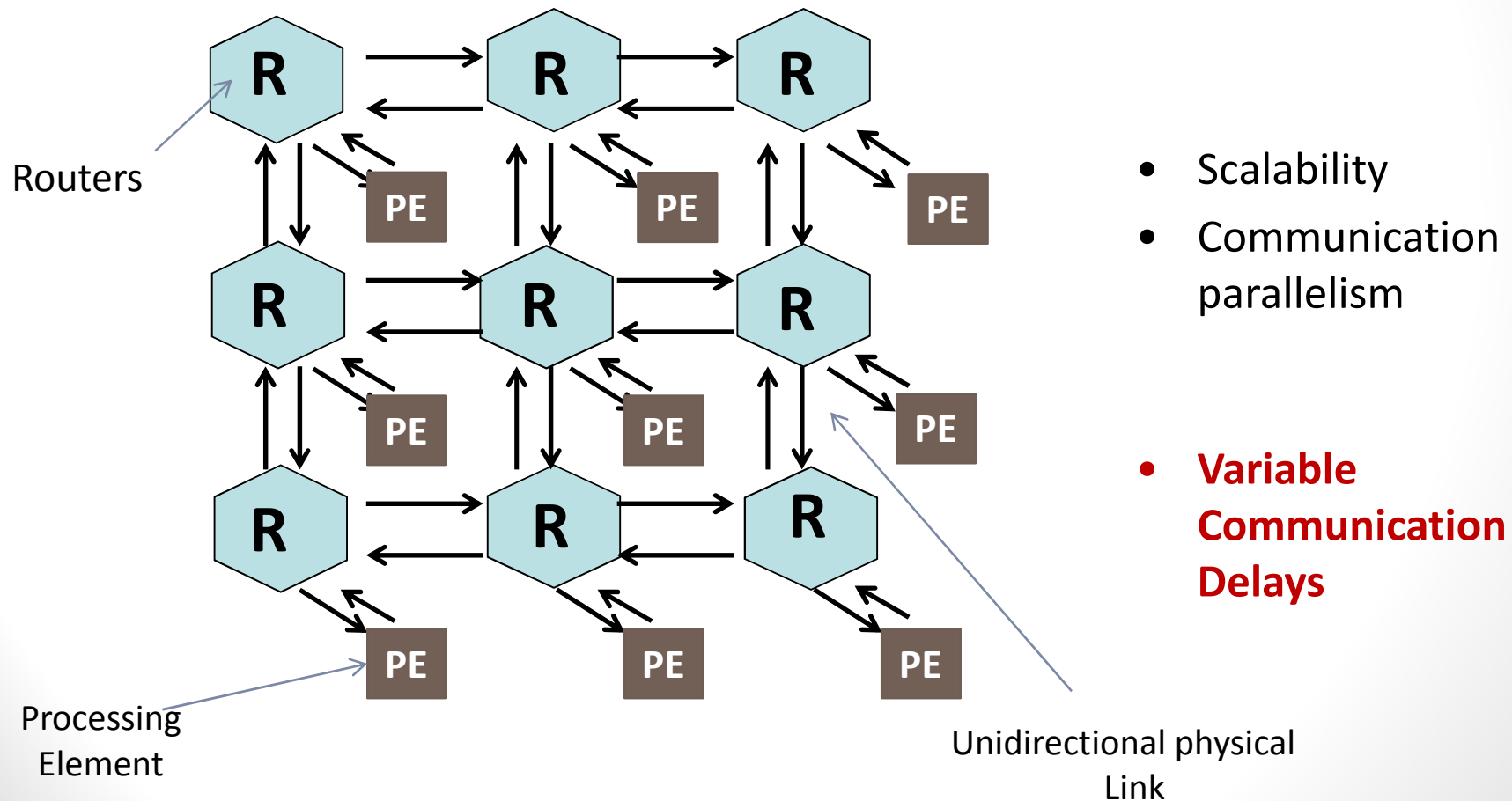
- Methods of schedulability analysis assume that the response time is predictable, while not
  - Two functionally independent tasks are not actually independent
    - Contention shared hardware resources
    - Caches L2, NoC, bus

# Context : NoC and MCS

- Problem Statement : **How to Deploy MCS over NoC ?**
- **Using NoC :**
  1. How we can :
    - Ensure the timing constraints for High-Critical flows
    - Minimizing the impact of resource sharing on Low-Critical flows
  2. How we can schedule Real-Time communications according to the tasks of applications ?

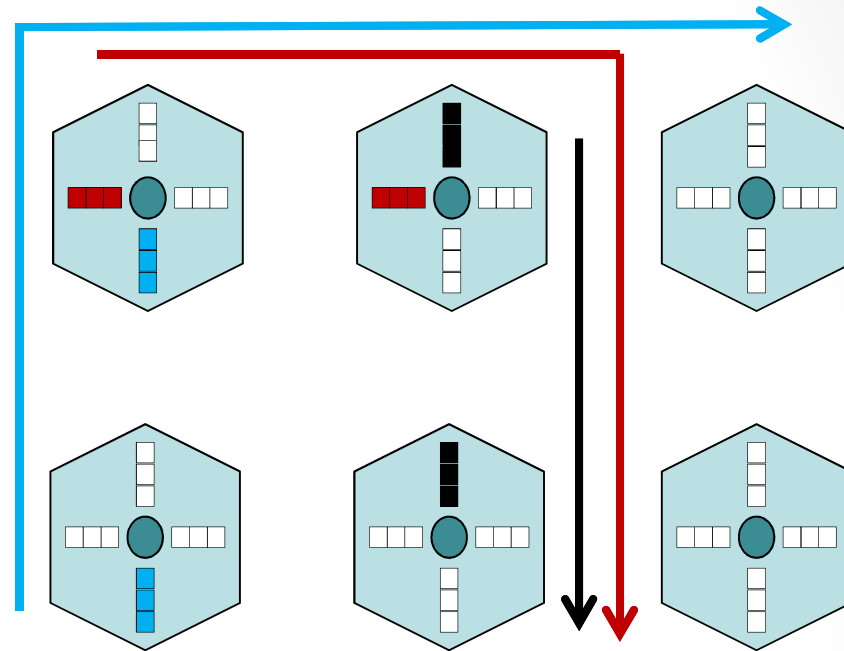
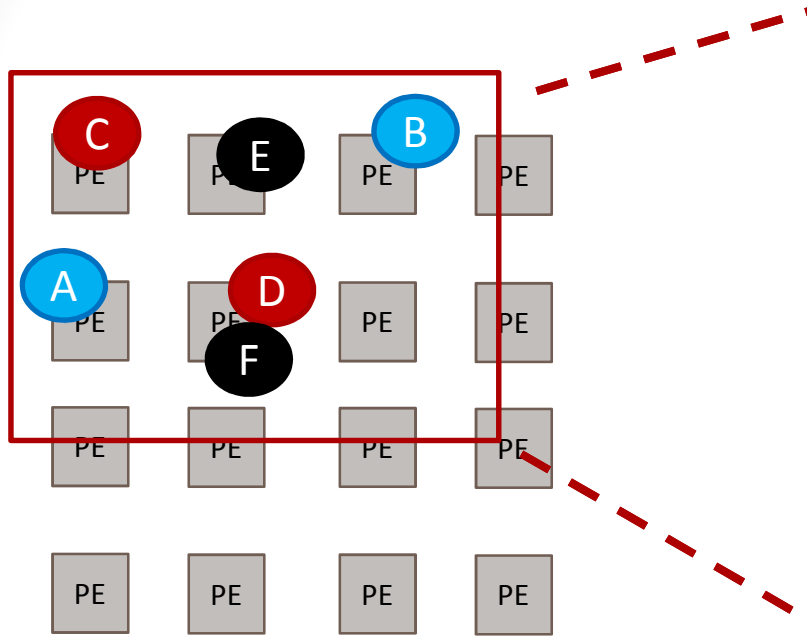
# Context : Network On Chip (NoC)

- Communication infrastructure based on links and routers that interconnect cores providing packet-based data transfer





# Examples of latencies with a NoC



## Assumptions:

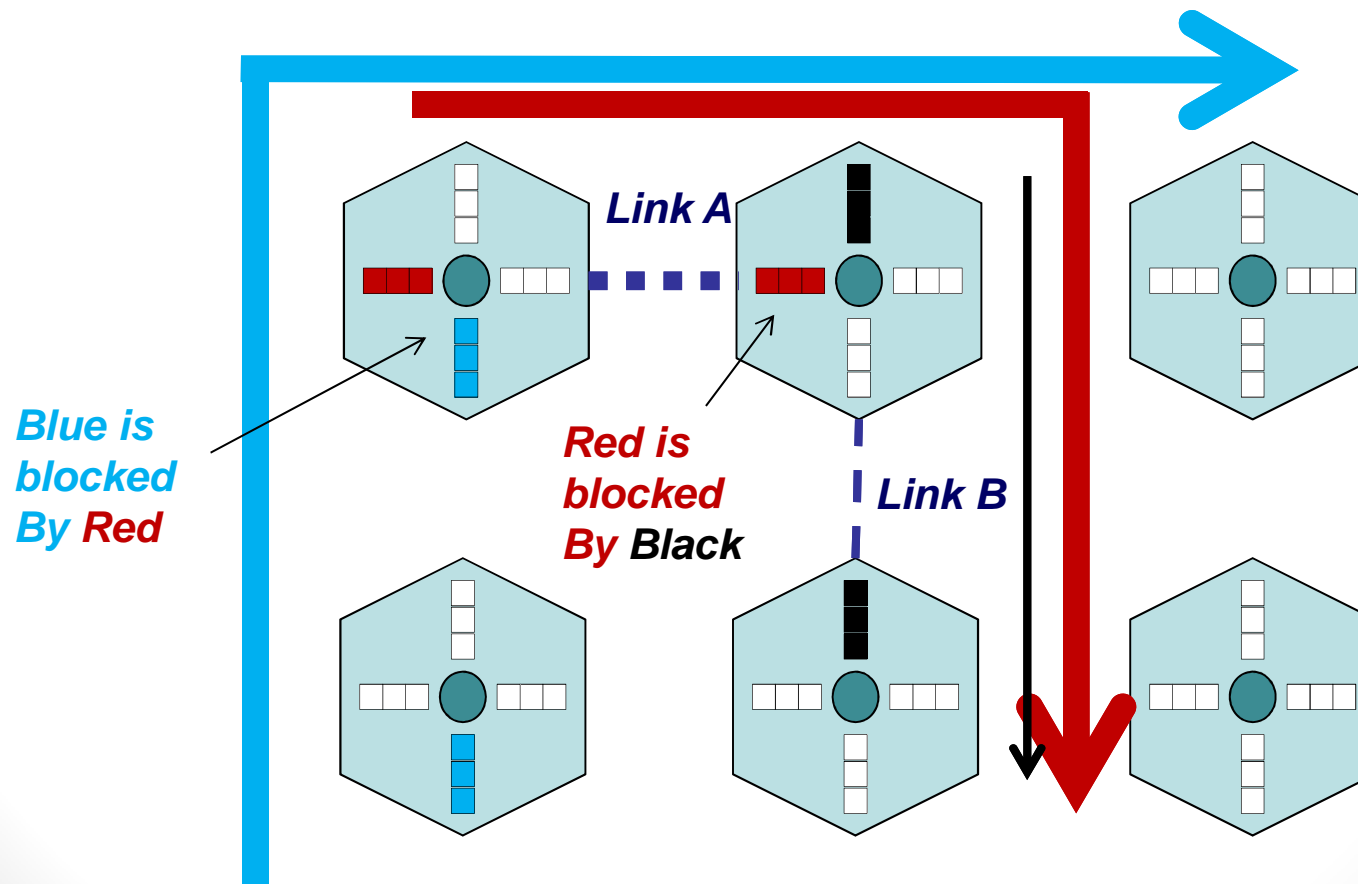
- 6 Dependent tasks
- 4\*4 NoC

## Assumptions:

- 3 Flows : Blue, Red and Black
- 6 Routers
- 4 Buffers in each Router

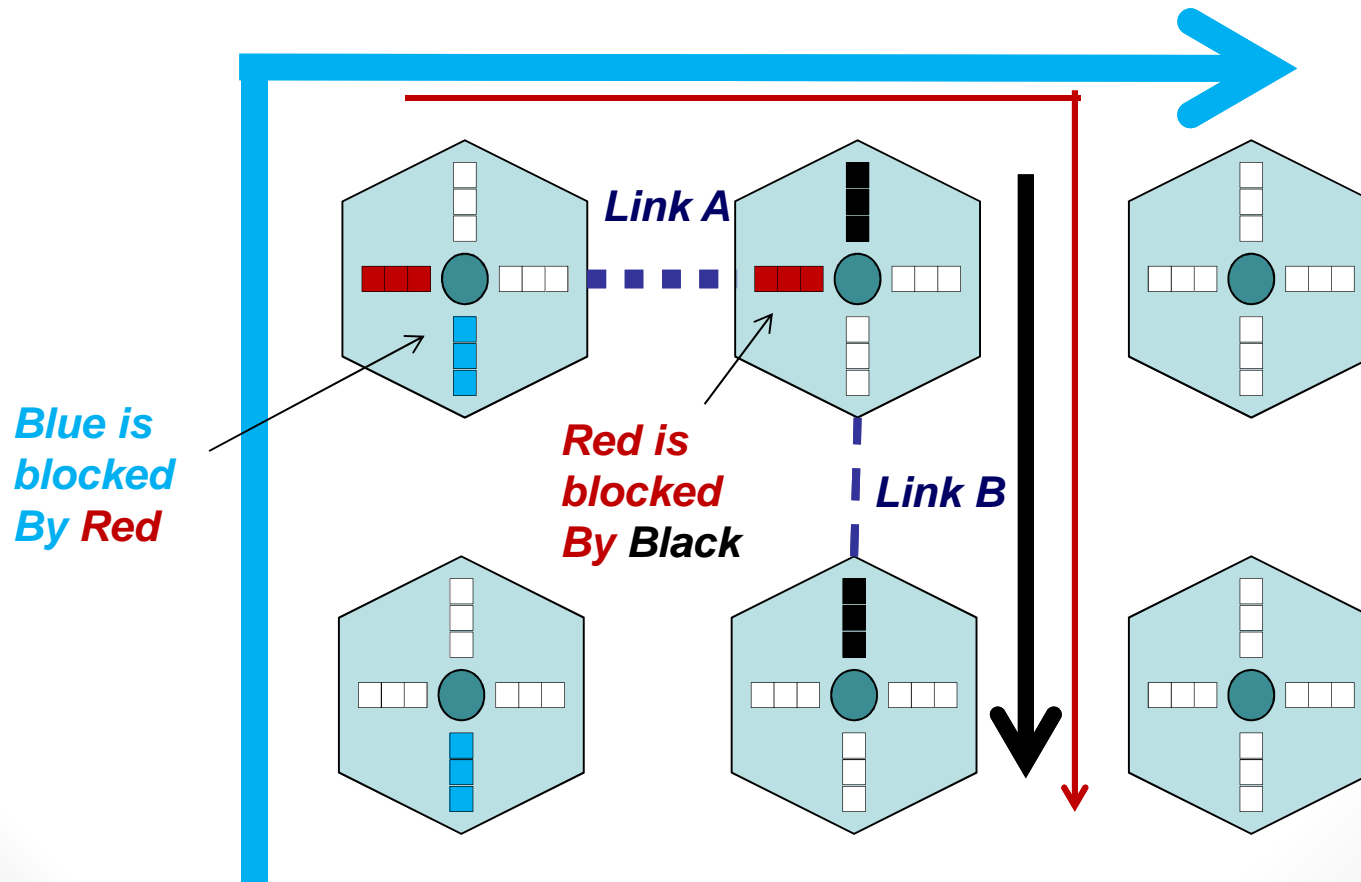
# Examples of latencies with a NoC

- Blue flow and Red flow share the same physical link (Link A)  
➔ Direct interference latency



# Examples of latencies with a NoC

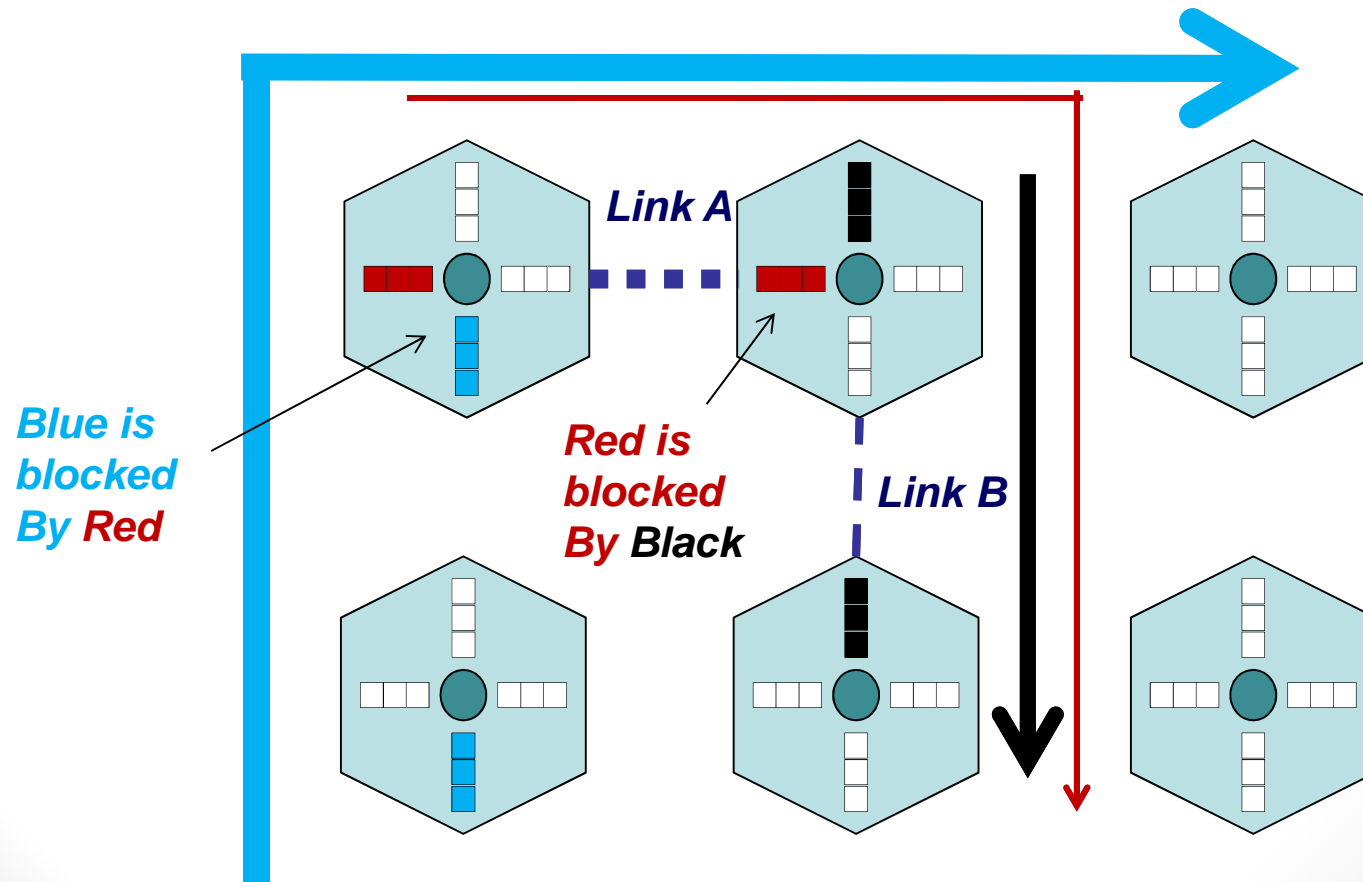
- Blue flow and Black flow do not share any physical link  
➔ Indirect interference latency



# Examples of latencies with a NoC

Different types of latencies introduced by the NoC:

- Path latency
- Direct interference latency
- Indirect interference latency



## Context : NoC and MCS

- **Contributions :**

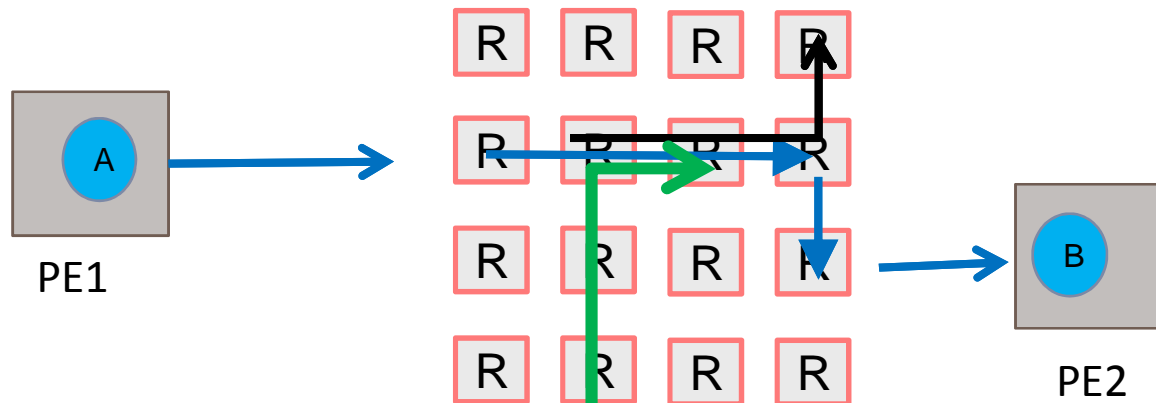
1. **DTFM** : a Dual Task and Flow Model to assess timing predictability of Real-Time applications over NoC architectures
2. **DAS** : A router architecture for On-Chip Network running Mixed-Criticality applications

# Plan

1. Context
2. **DTFM : a Scheduling Analysis Model for NoC based Systems**
3. DAS : a NoC Router for Mixed-Criticality Systems
4. Conclusion

# DTFM : Problem Statement

Periodic dependent tasks  
Periodic data flows



## ✘ Task models used by designers :

schedule the execution of many Real-Time tasks on processors

**Insufficient** : Do not take into account the delays introduced by NoC

## ✘ Flow models used by designers :

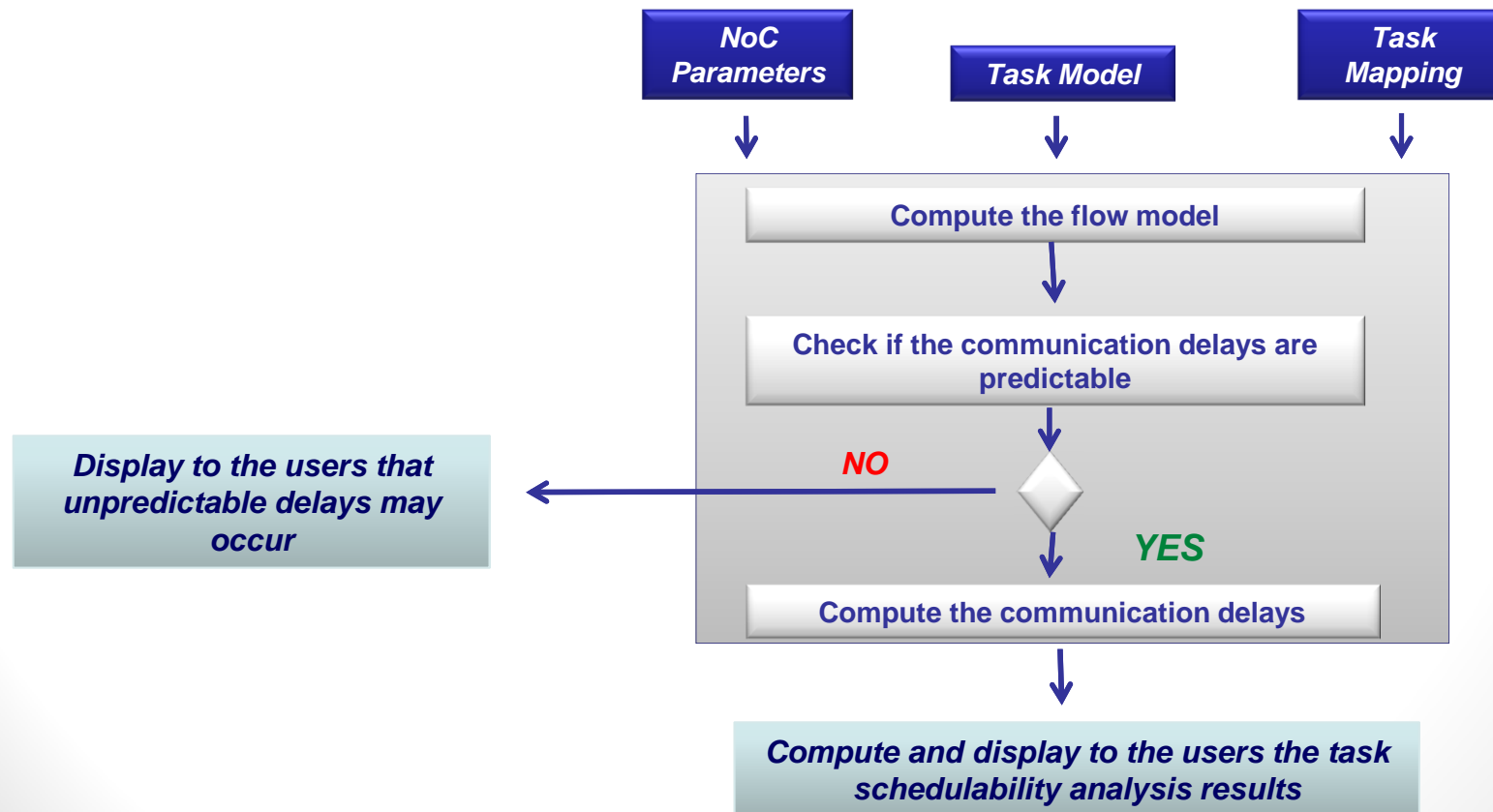
schedule Real-Time flows on NoC communication platform

**Insufficient** : Do not take into account the schedulability analysis for tasks

# DTFM : Proposition

DTFM, a Dual Task and Flow Model: assess timing predictability of real-time applications over NoC architectures.

- Take into account the communication delays and possible network conflicts
- Take into account the task schedulability





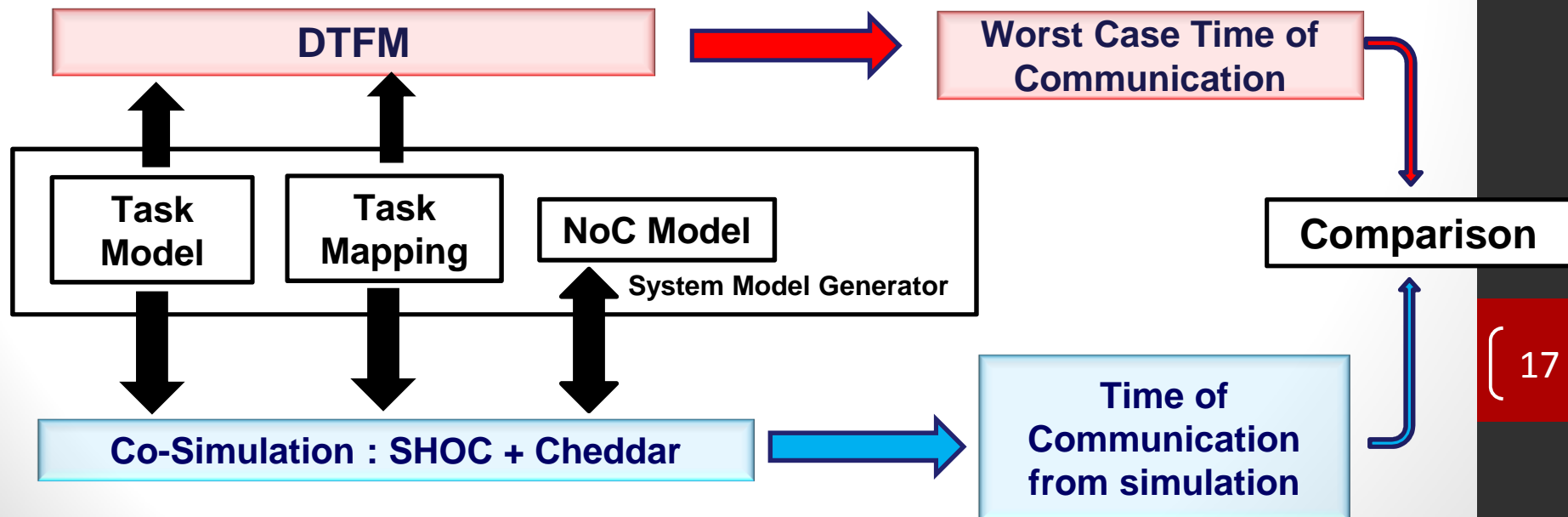
# DTFM : Evaluation with a multiscale toolset

- **Multiscale toolset**

- **DTFM :**

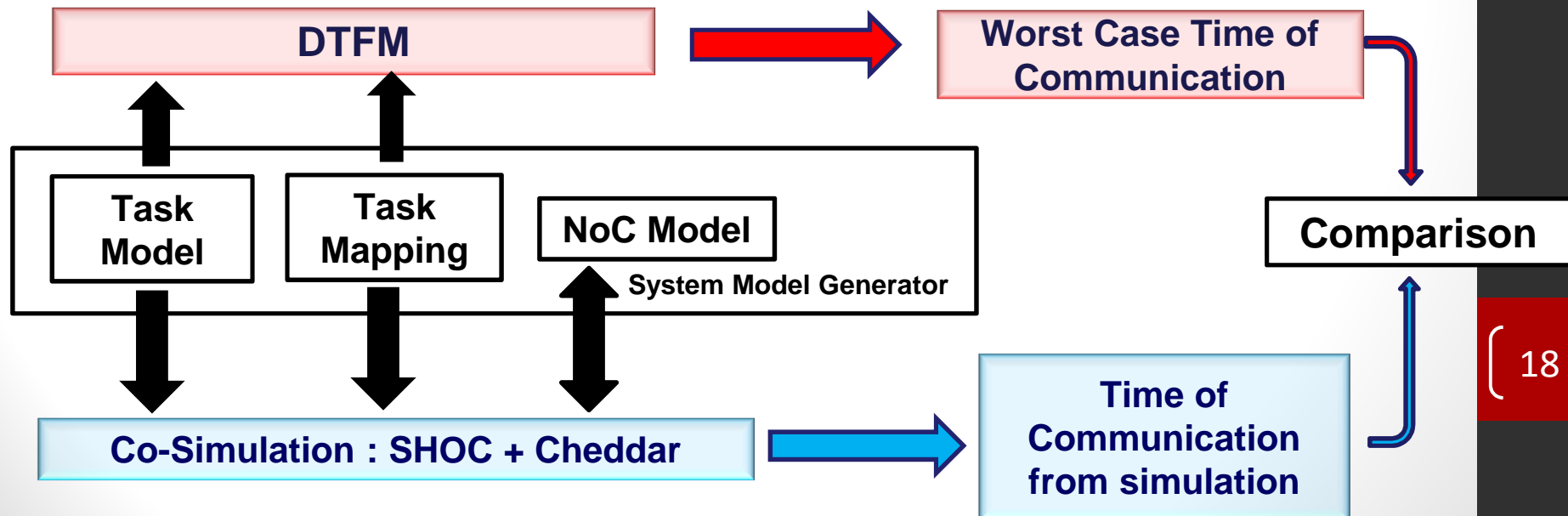
- produce the flow model from the task model
- Compute the communication delays

- **Co-Simulation with SHOC (TUM, Lab-STICC) and Cheddar (Lab-STICC)**



# DTFM : Evaluation with a multiscale toolset

- **Co-Simulation with SHOC and Cheddar :**
  - **SHOC simulator** : A cycle-accurate SystemC NoC simulator  
Simulate the communication time of each message
  - **Cheddar simulator** : A tick-accurate schedulability analysis tool  
Compute the task scheduling in order to predict when messages are sent to the NoC

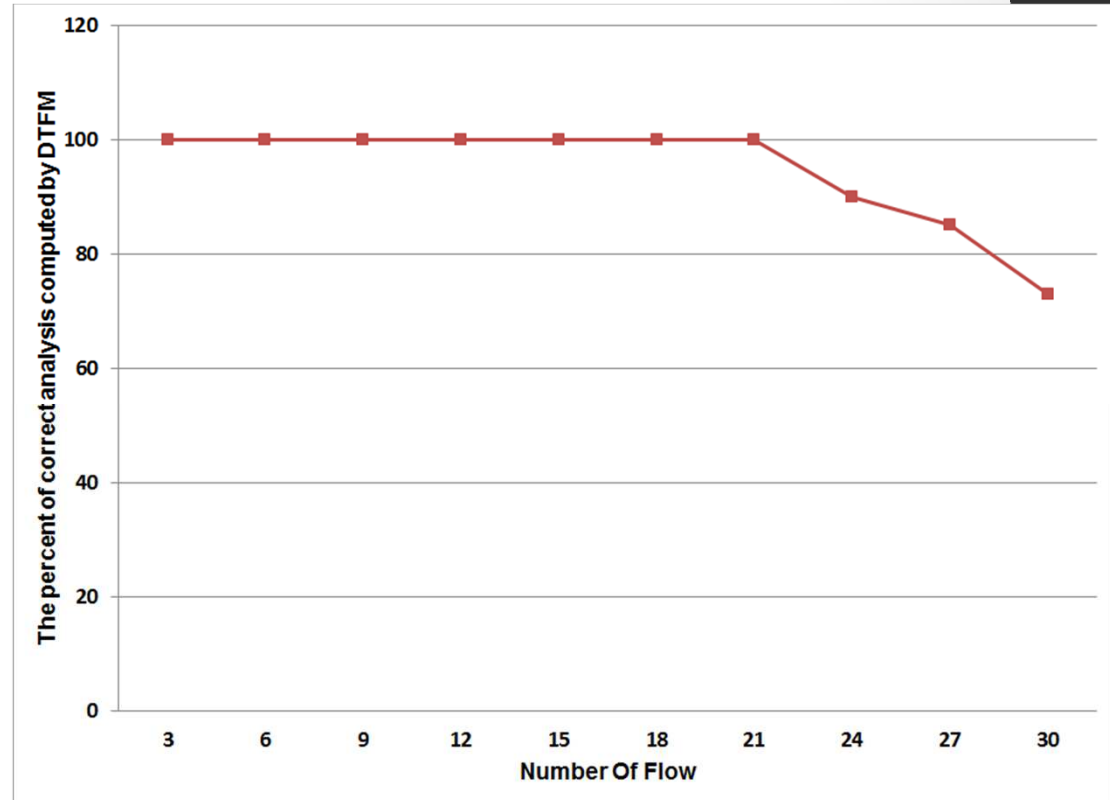


# DTFM : Evaluation with a multiscale toolset

- Validate the correctness of DTFM for the schedulability of systems

- Present the rate of correct analysis computed by DTFM

- For each configuration, vary the task mapping to change the number of flows
- Number of flows ranging from 3 to 30 with a step of 3



- **If there is no indirect delays, any task sets said schedulable by DTFM are also schedulable by SHOC :**
  - **DTFM is able to predict delays**
- **DTFM may be pessimistic**

# Plan

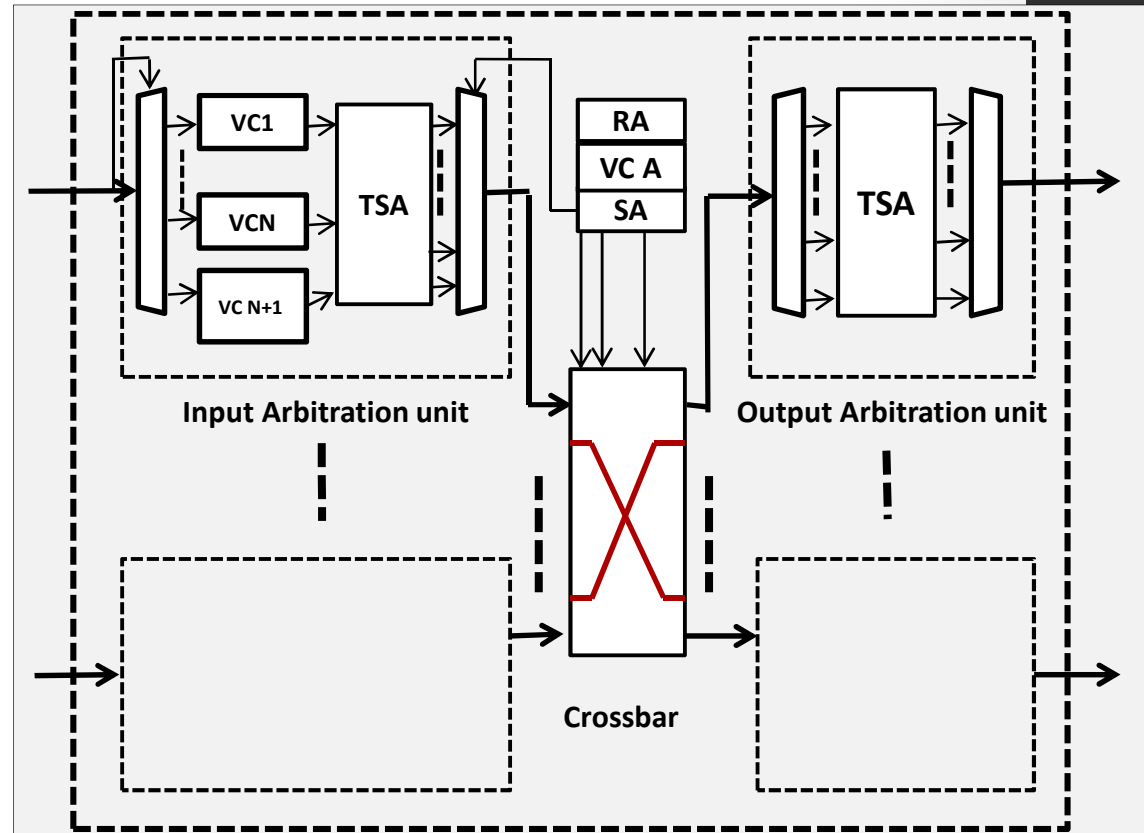
1. Context
2. DTFM : a Scheduling Analysis Model for NoC based Systems
3. **DAS : a NoC Router for Mixed-Criticality Systems**
4. Conclusion

# DAS : Problem Statement

- In order to deploy MCS over NoC, we need to :
  - **Ensure the timing constraints for High-Critical flows**
  - **Minimize the impact of resource sharing on Low-Critical flows**
  - **Provide an accurate communication time analysis**
- **Virtual Channel + Wormhole + Flit-Level Preemption**
  - ➔ **Too pessimistic Worst Case Communication Time of High-Critical flows**
- **TDM NoC Router**
  - ➔ **Low throughput for Low-Critical flows while providing highly deterministic communication time for High-Critical flows**

# DAS : Proposition

- **N+1 Virtual Channels**
- VCs 1 to N :  
High-Critical flows
  - Each VC is dedicated to only one given flow
- VC N+1 :  
Low-Critical flows
  - VC N + 1 can be shared by several low-critical flows



**Double Arbitrator and Switching (DAS)**

# DAS : Proposition

- **SAF for High-Critical flows**

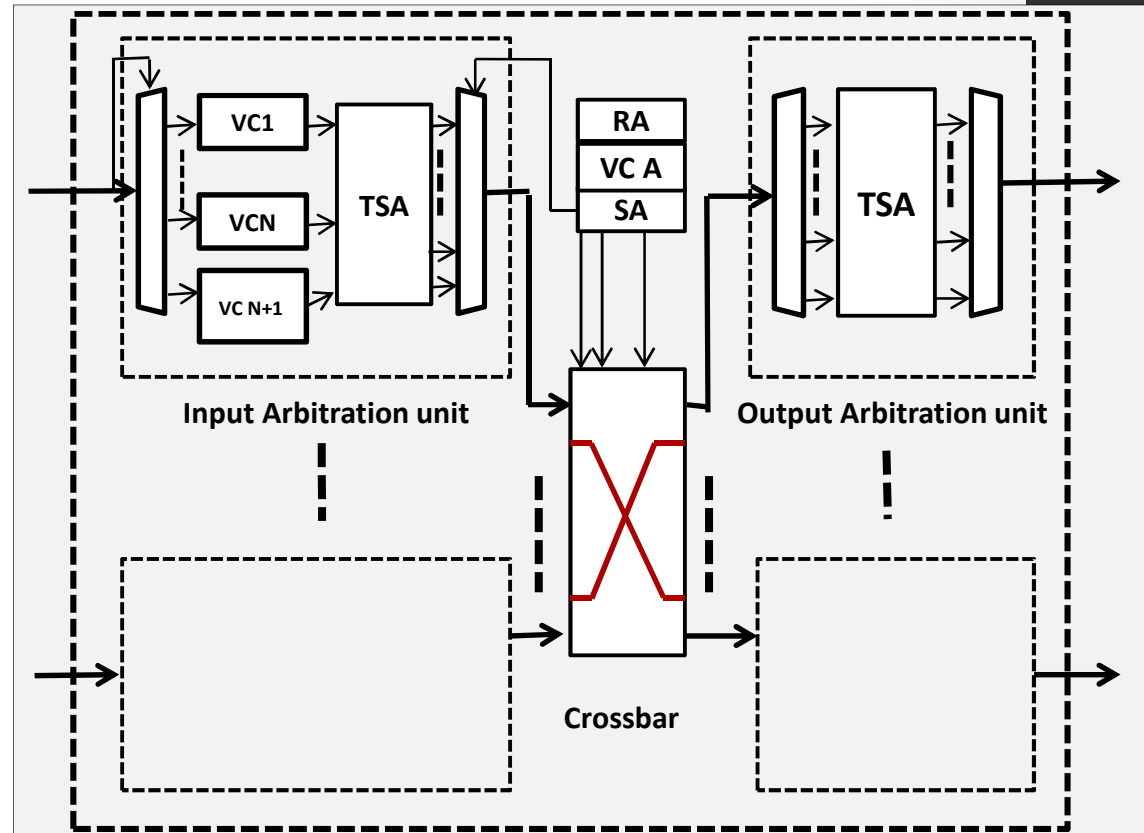
Each packet uses only one link at a time.



The congestion can be controlled with a reasonable cost considering small High-Critical packets.

- Why ?

- No indirect interference
- Less pessimistic worst case communication time

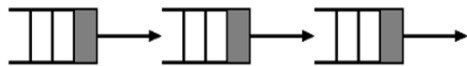


## Double Arbitrator and Switching (DAS)

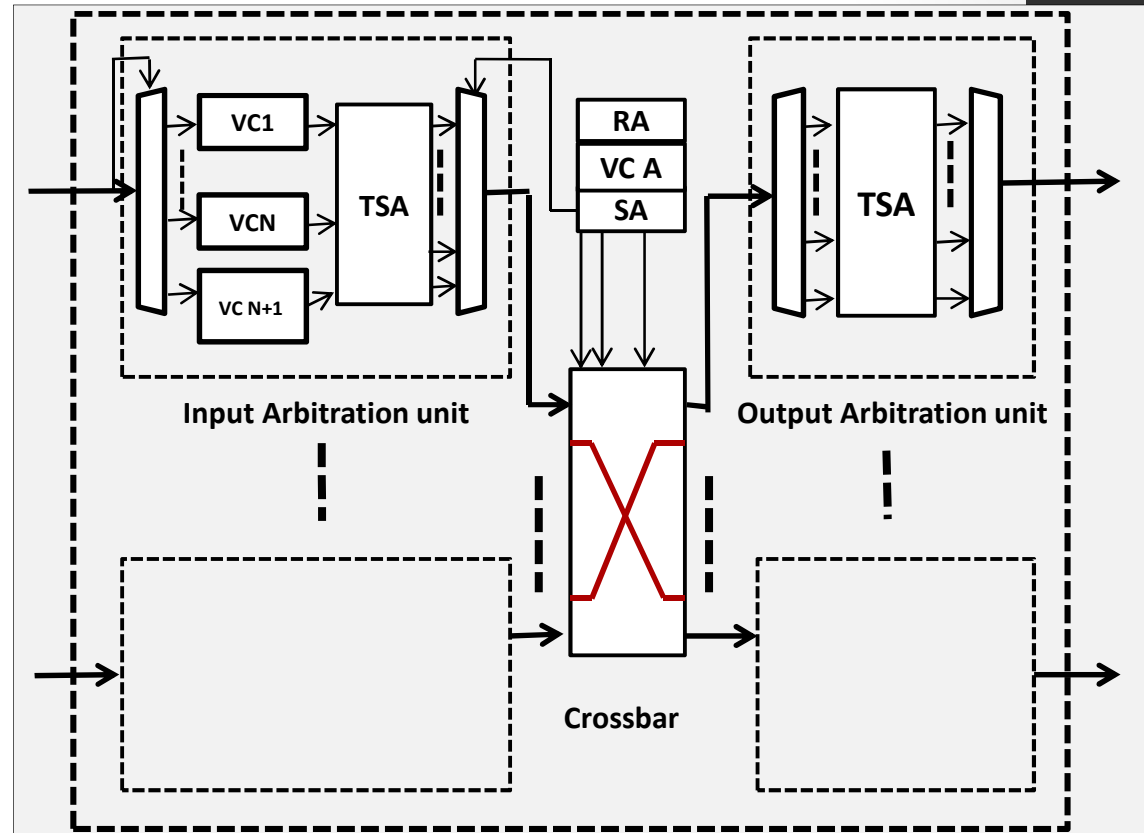
# DAS : Proposition

- **Wormhole for Low-Critical flows and Flit-level Preemption**

A packet can be stored over multiple routers and occupies several physical links at a time.



- Why ?
  - Buffer requirements are reduced to one flit, instead of an entire packet
  - Increase the network use rate by Low-Critical flows



**Double Arbitrator and Switching (DAS)**

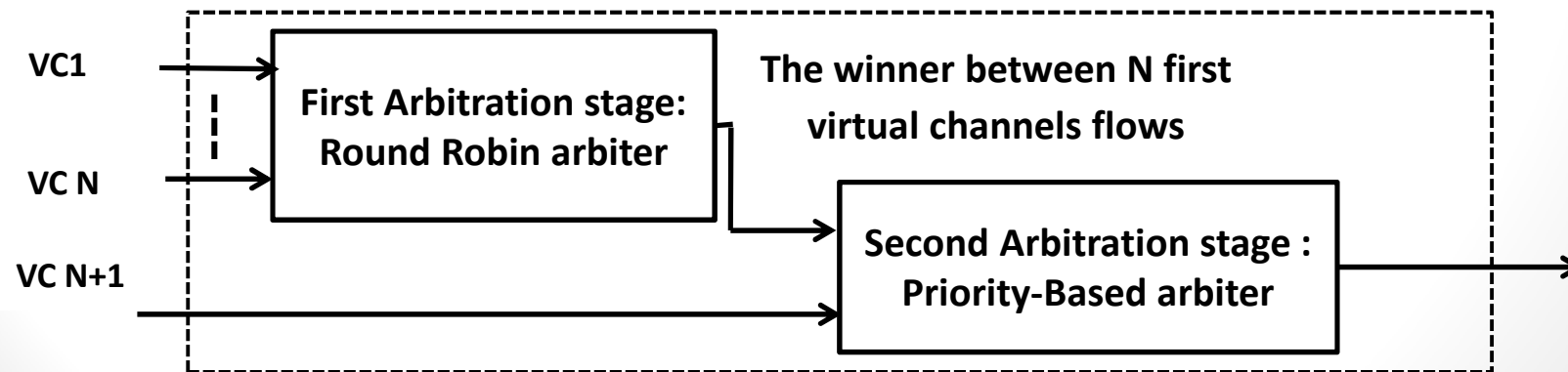


# DAS : Proposition

→ High-Critical flows always preempt Low-Critical flows in flit level

- **The Two Stages of Arbitration**

- Input and output arbitration units are based on two stages of arbitration :
  1. The first stage is a Round Robin Arbitration between the N first virtual channels
  2. The second stage is a Priority-Based Arbitration between the winner of the first stage and the last virtual channel.



# DAS : Implementation and Evaluation

- **SystemC and Verilog HDL implementation**
- **Integrated in SystemC TLM Simulator “SHOC”**
  - SHOC : A cycle accurate SystemC-TLM simulator
- **Evaluation :**
  - High-Critical flow latency evaluation
  - Low-Critical flow latency evaluation

	VC-Router	DAS
Dimension	4*4	4*4
Topology	2D-mesh	2D-mesh
Routing Algorithm	XY	XY
Switching Mode	Wormhole	Wormhole/SAF
Preemption Level	Packet	Flit / Packet
Arbitration Policy	Priority-Based Arbiter	Two Stages of Arbitration

# DAS : High-critical flow Latency Evaluation

Evaluate the impact of the resource sharing on the communication delay of High-Critical flows.

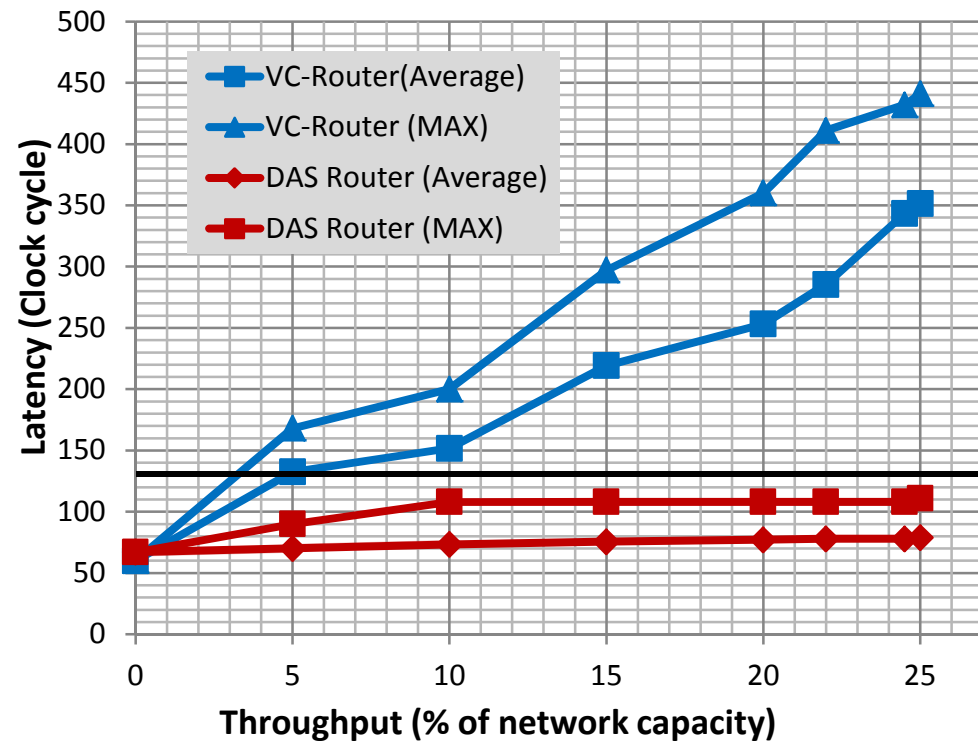
In every evaluation, one High-Critical flow is assigned to a randomly generated source and destination node.

For each simulation, we generate Low-Critical flow which share some physical links with the High-Critical flow.

High-Critical flow size = 2 flits  
Low-Critical flow size = 8 flits

Release time and the period of each flow are randomly generated.

3 physical links from the source to the destination node.



- Using DAS, High-Critical flows communication delays are bounded

# DAS : Low-Critical Flow Latency Evaluation

Evaluate the latency overhead on Low-Critical flows due to High-Critical flows resource reservation comparing to virtual channel routers.

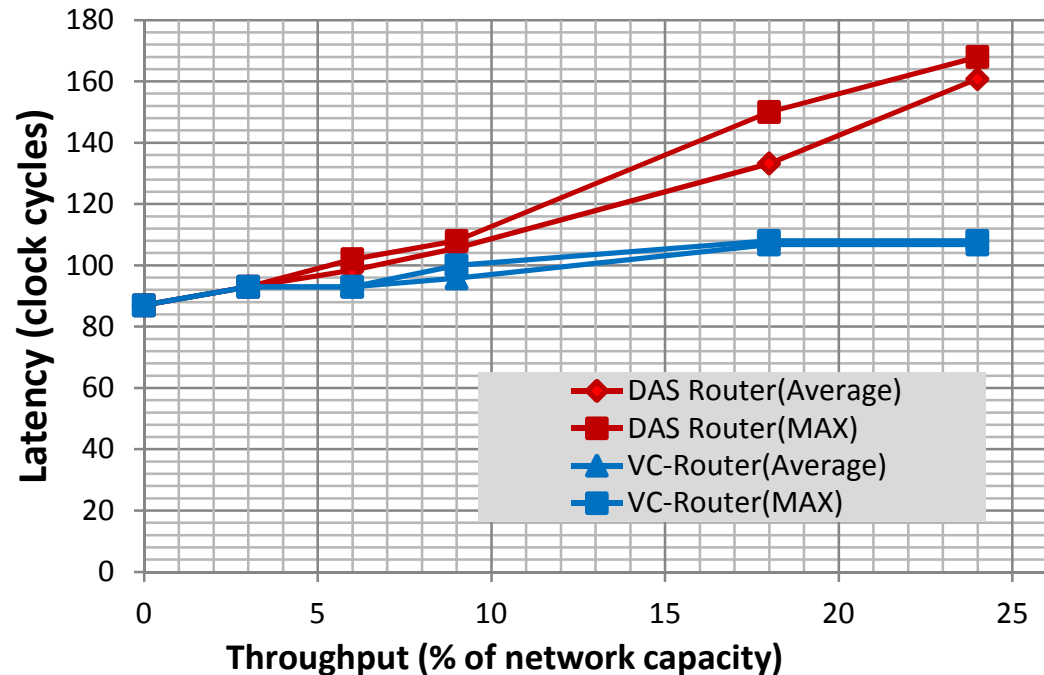
In every evaluation, we generate randomly one low-critical flow.

100 simulations by increasing the number of high-critical flows and by decreasing the high-critical flow period in order to increase the network use rate.

High-Critical flow size = 2 flits  
Low-Critical flow size = 8 flits

Release time of each flow are randomly generated.

3 physical links from the source to the destination node.



- DAS leads to larger latencies for Low-Critical flows compared to a Virtual Channel Router.
- Low-Critical flows in MCS may tolerate some additional delays without damaging the integrity of the whole system

# DAS : Cost evaluation

- **Typical NoC Router**, **Virtual Channel Router** and **DAS** synthesized with Synopsys DC using a 28nm ST SOI technology.
- Tools included in this technology generate reports describing the area of implementation

	Typical Router	Virtual Channel Router (A.Burns, 2009)	DAS
Number of ports	5	5	5
Data width	32-bits	32-bits	32-bits
Buffer size	16-flits	16-flits	16-flits
Total cell area	16046.31	18369.47	18831.32
Total area	1	+17%	+2,51%

# DAS : Validation with Model-Checking

- **Model Checking :**
  - Computes all the possible states of the system
  - Verify whether properties hold on all the possible states of the system
- **To validate the properties of DAS :**
  1. Formalize properties to be guaranteed by DAS
  2. Develop a model of DAS using state machines
  3. Express DAS using IF language and its tools
    - IF tool Provides an environment for modeling and validation of real-time systems described in IF language. (**M. Bozga, 2004**)
  4. Validate properties using IF-Observer

# DAS : Validation with Model-Checking

## An Example of an automaton of DAS

- The behavior of each of these entities is described using state machine.

### An automaton is :

- A set of states
- A set of transitions

### A transition :

- Can be fired when the guard is true
- When fired, an action is computed

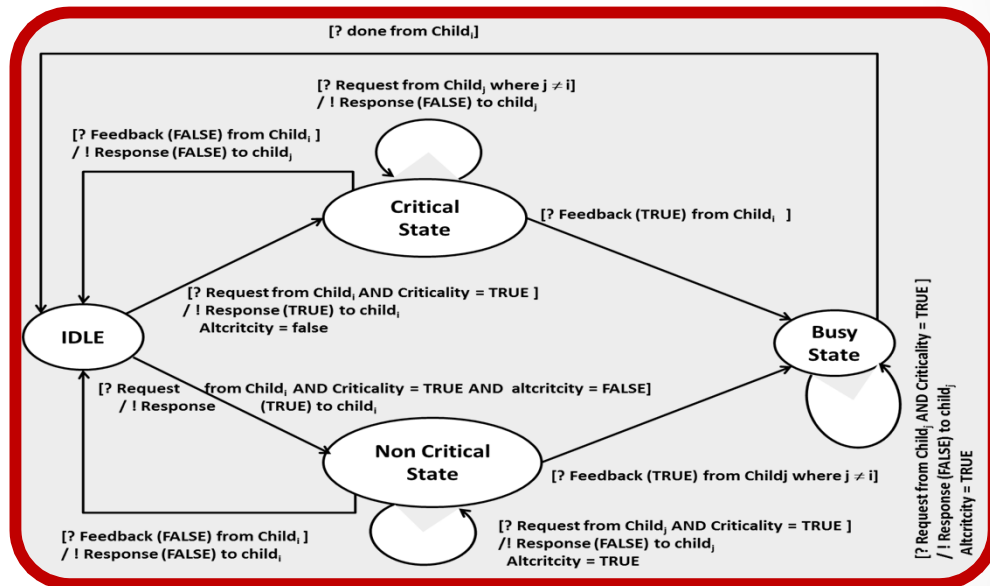
```

/*****
 * process Switch Declaration
 *****/
process Switch(SW);

    var index_i IndexType;
    var index_j IndexType;
    var childPid_i pid;
    var childPid_j pid;
    var doneBoolean boolean;

/*****
 * State idle
 *****/

state idle #start ;
    deadline lazy;
    input DAS_request_Switch(index_i);
        task childPid_i := ([DASMain]0).childInfoTable[index_i].childPid;
        output DAS_response_Switch(true) to [DASchild]childPid_i;
        nextstate busyState;
endstate;
    
```



**Example : Input Arbiter automaton**

- 15 Parameters
- 19 Automata
- 15 Signals
- 13 Buffers

# DAS : Validation with Model-Checking

- Exhaustive verification of the properties on any possible states of the system
- IF language provides observers to check properties
  - The IF observer is an extended timed automaton which is executed in parallel with the target system.
- **Properties :**
  - **P.1 High-Critical flows always preempt Low-Critical flows in flit level.**
  - **P.2 High-Critical flows always have a higher priority than Low-Critical flows.**

Properties	Number of States	Number of Fired Transitions	Time (hh:mm:ss)	Proof
P.1	378 452	858 546	00:00:20	Yes
P.2	386 684	811 734	00:00:18	Yes



# Plan

1. Context
2. DTFM : a Scheduling Analysis Model for NoC based Systems
3. DAS : a NoC Router for Mixed-Criticality Systems
4. **Conclusion**

# Conclusion

## Problem Statement

- Schedulability analysis of MCS over NoC architectures

## Contributions

- **DTFM : a Dual Task and Flow Model assess timing predictability of Real-Time applications over NoC architectures**
  - Implementation available in the Cheddar tool  
<http://beru.univ-brest.fr/svn/CHEDDAR/>
  - Co-simulation: combining a cycle-accurate SystemC simulator and a tick-accurate scheduling analysis tool
  - *“DTFM: a Flexible Model for Schedulability Analysis of Real-Time Applications on NoC-based Architectures”, **Reaction 2016, Porto, Portugal***

# Conclusion

- **DAS : A router architecture for On-Chip Network running Mixed-Criticality applications**
  - Evaluation : SystemC simulation with SHOK, Verilog HDL implementation, model checking with IF
  - *“DAS: An Efficient NoC Router for Mixed-Criticality Real-Time Systems”*, **ICCD2017, Boston, USA**
  - *“Modeling and Validation of a Mixed-Criticality NoC Router Using the IF Language”*, **NoCArc2017, Boston, USA**

## Future Works :

- Measure of the gain for low-critical flow with DAS
- Virtual Channel Manager : dynamic allocation of VCs



# Multi-Criticality Systems vs Mixed-Criticality Systems

## Multi-Criticality Systems

- Tasks with different level of criticality
- One Execution mode
- No mode change
- LO tasks are never stopped executing or changed of periods.

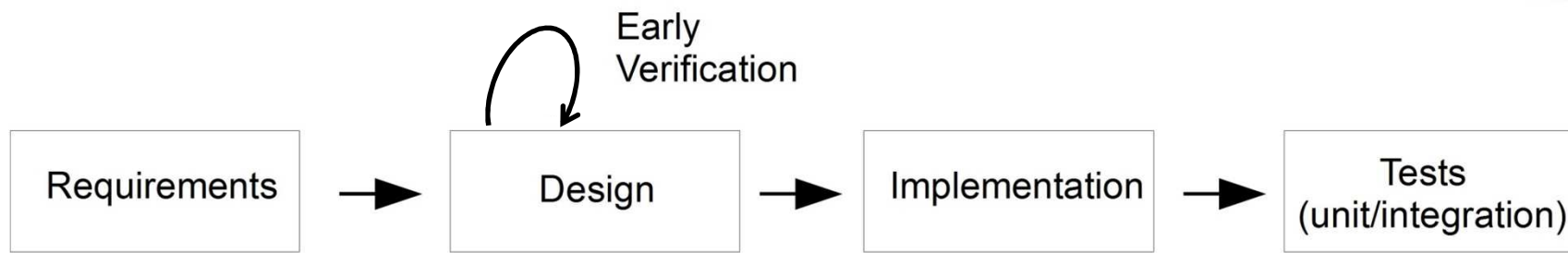
## Mixed-Criticality Systems

- Tasks with different level of criticality
- Several Execution mode:
  - Degradation Mode
  - Normal Mode
  - ...
- Mode change
- In degradation mode, LO tasks are stopped executing or changed of periods.

*“Systems with more than one criticality level but aim to only give complete isolation are called multiple-criticality systems; The use of mixed-criticality implies some tradeoff between isolation and integration that involves resource sharing.”*

*Alan Burns and Robert I. Davis, Mixed Criticality Systems - A Review, Jan 2017*

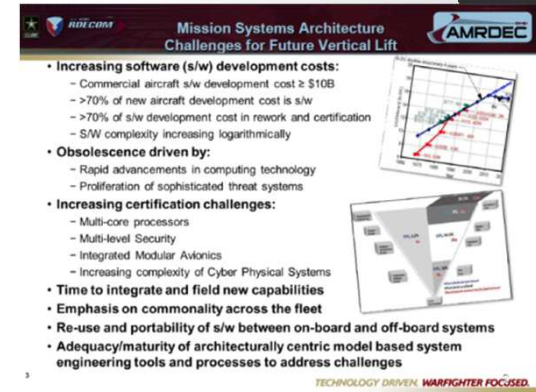
# Early Verification



- **Specific software engineering** methods/models/tools to master quality and cost
  - Example : early verifications at design step

# Motivation for early verification

- **From AMRDEC:**
  - 70% of fault are introduced during the design step ; Only 3% are found/solved. Cost : x1
  - Unit test step: 20% of fault are introduced ; 16% are found/solved. Cost : x5
  - Integration test step: 10% of fault are introduced ; 50% are found/solved. Cost : x16
- **Objective:** increase the number of faults found at design step!
- **Early verification:** multiple verifications, including expected performances, i.e deadlines can be met?



The slide is titled "Mission Systems Architecture Challenges for Future Vertical Lift" and features the AMRDEC logo. It lists several key challenges and trends in software development and system integration. The text is organized into bullet points with sub-bullets. To the right of the text, there are two small diagrams: a line graph showing an exponential increase in complexity and a block diagram of system architecture.

- **Increasing software (s/w) development costs:**
  - Commercial aircraft s/w development cost  $\geq$  \$10B
  - >70% of new aircraft development cost is s/w
  - >70% of s/w development cost in rework and certification
  - S/W complexity increasing logarithmically
- **Obsolescence driven by:**
  - Rapid advancements in computing technology
  - Proliferation of sophisticated threat systems
- **Increasing certification challenges:**
  - Multi-core processors
  - Multi-level Security
  - Integrated Modular Avionics
  - Increasing complexity of Cyber Physical Systems
- **Time to integrate and field new capabilities**
- **Emphasis on commonality across the fleet**
- **Re-use and portability of s/w between on-board and off-board systems**
- **Adequacy/maturity of architecturally centric model based system engineering tools and processes to address challenges**

TECHNOLOGY DRIVEN WARRIGHTER FOCUSED