FORMAL MODELING AND VERIFICATION FOR TIMING PREDICTABILITY

Mathieu Jan, Mihail Asavoae, Belgacem Ben Hedia

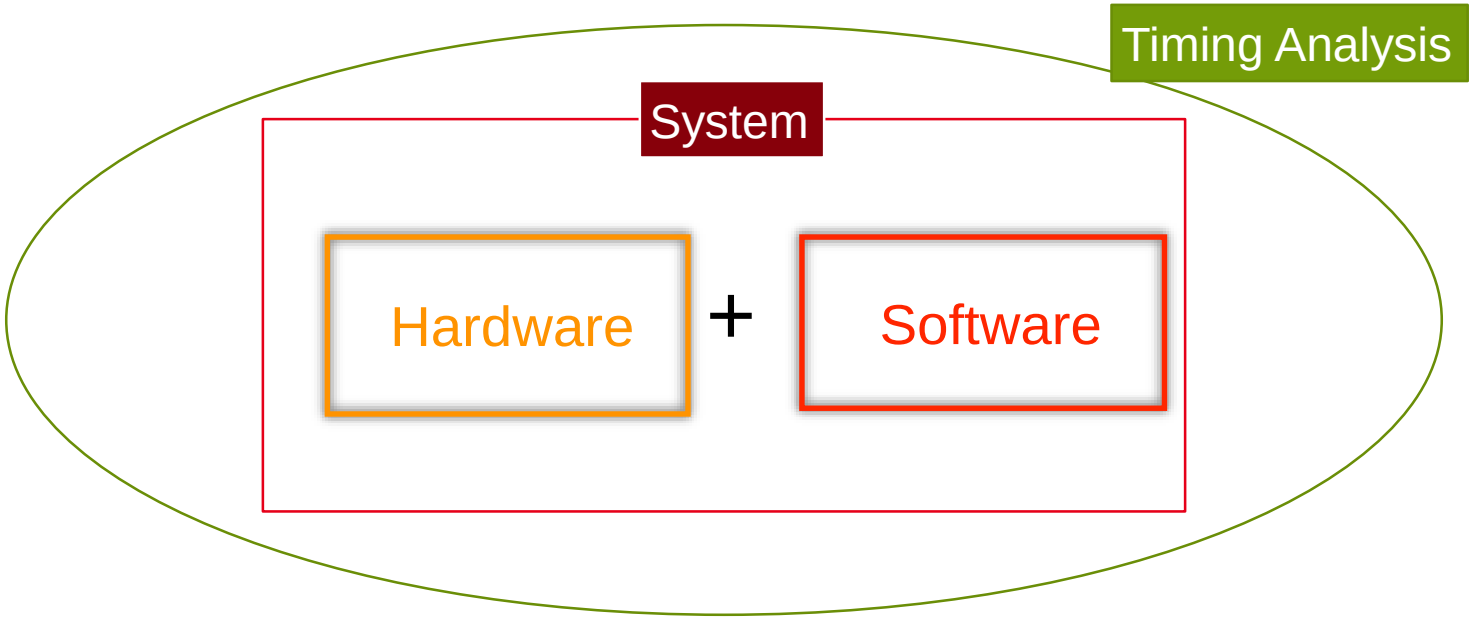
ERTS 2020
Toulouse

Outline of the talk

- **Motivations and goals**
  - Timing anomalies in Worst-Case Timing Reasoning
  - An example of a timing anomaly
  - Detection of timing anomalies and related work
- **Formal modeling language**
- **Case study: anomalies in out-of-order architectures**
  - Pipeline and software models
  - Acquire/release logic of functional units
  - Evaluation
- **Conclusions and future work**

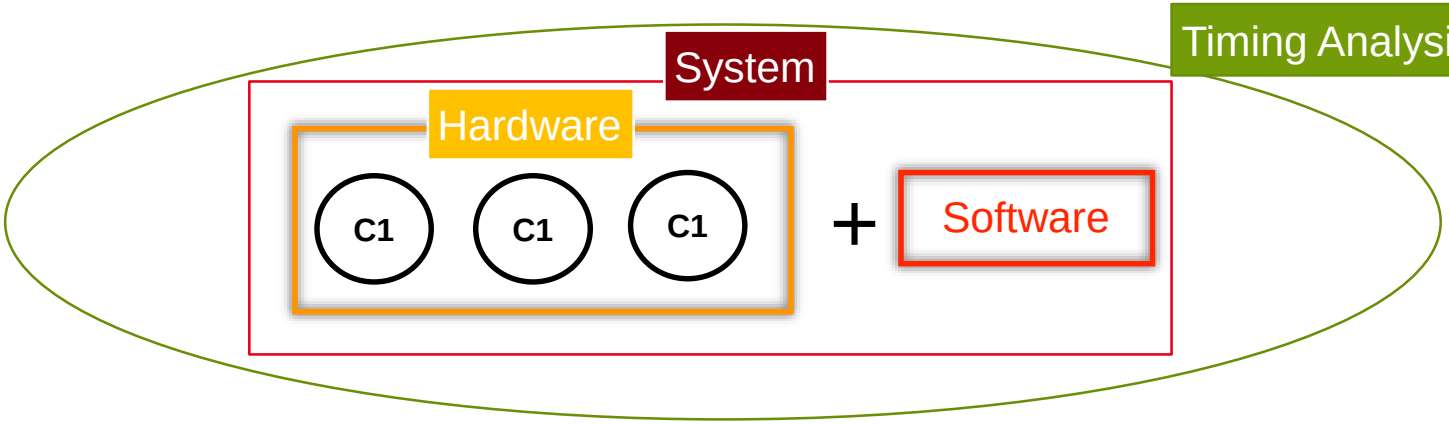
Timing Reasoning

- **Safety-critical systems need to satisfy strong timing constraints**
  - Timing reasoning in worst-case style in order to compute safe and tight timing bounds! ➔ WCET analysis
  - Requires inputs from Hardware + Software = System



WCET analysis assumptions

- **Properties to simplify WCET analysis**
  - Timing predictability: follow only local worst-case behaviors
  - Timing compositionality: compose local timing contributions



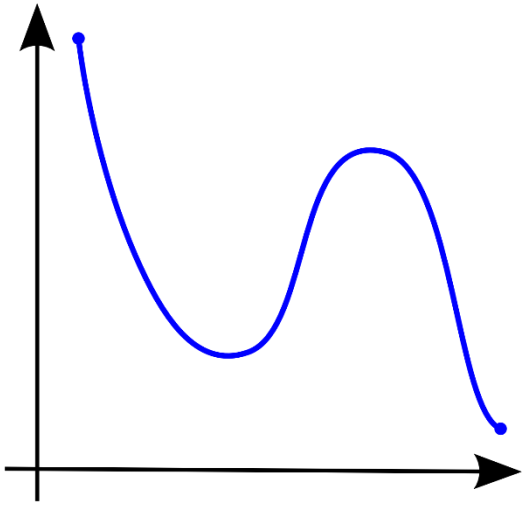
- **Most existing WCET analysis tools**
  - Simply assume both properties!
  - Known to be incorrect

ECRTS18, …

block, and thereby include the blocks before classified as *NC*. Note that we assume that the processor does not exhibit timing anomalies. We leave that investigation to future work.

Timing anomalies

- **What is a timing anomaly?**
  - A non-monotonic temporal behavior
  - Multiprocessor scheduling, Richard's anomalies (1966-76)

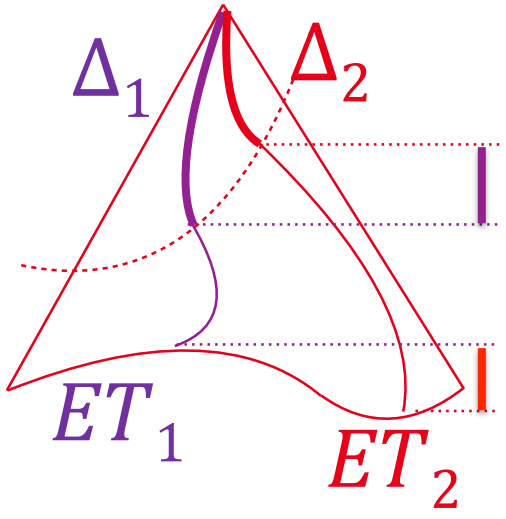


- **In the context of processor hardware (RTSS 1999)**
  - Supposed to be present only in out-of-order architectures
    - Shown to be present in in-order architectures (2002-2005)
    - Resource Allocation Condition (RAC): necessary condition for a timing anomaly
  - First formal definition in WCET 2006

Types of timing anomalies

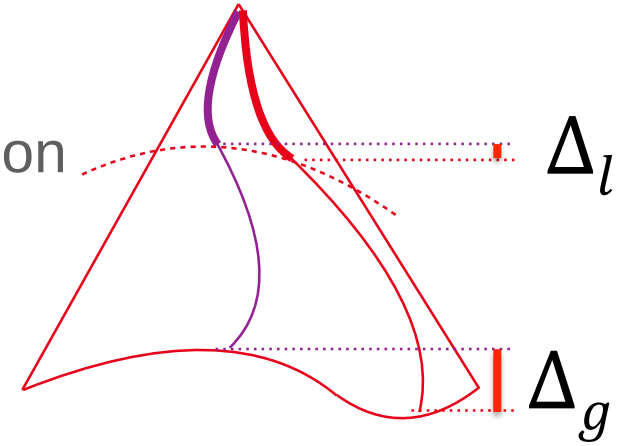
- **Counter-intuitive**
  - A local fast execution slows down an overall global execution
  - Available in the scheduling accesses of functional units

$$\Delta_1 > \Delta_2 \wedge ET_1 < ET_2$$



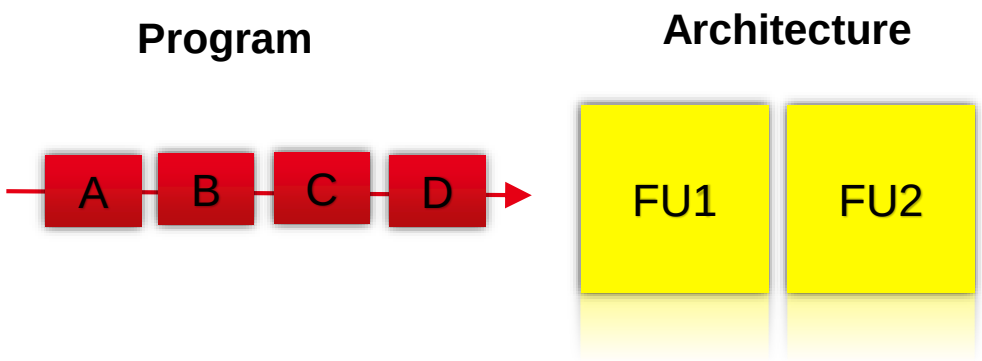
- **Amplification**
  - A local variation leads to a larger variation
  - Available in a in-order pipeline accessing in FCFS a bus arbiter

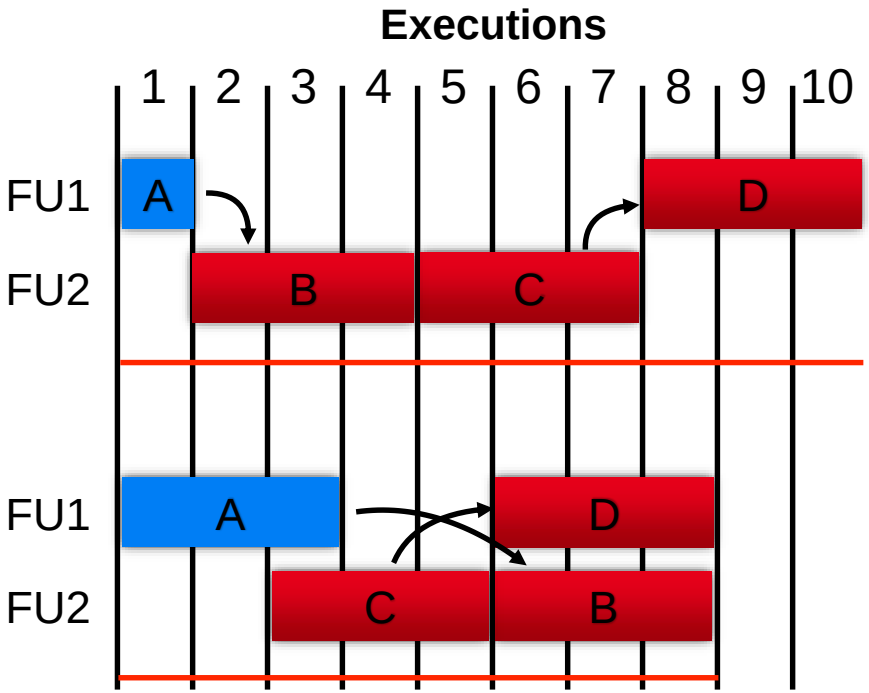
$$\Delta_g > \Delta_l$$



Timing anomalies: an example

- **Acquire/release of functional units in a pipeline**
  - Out-of-order
  - Scheduling timing anomalies



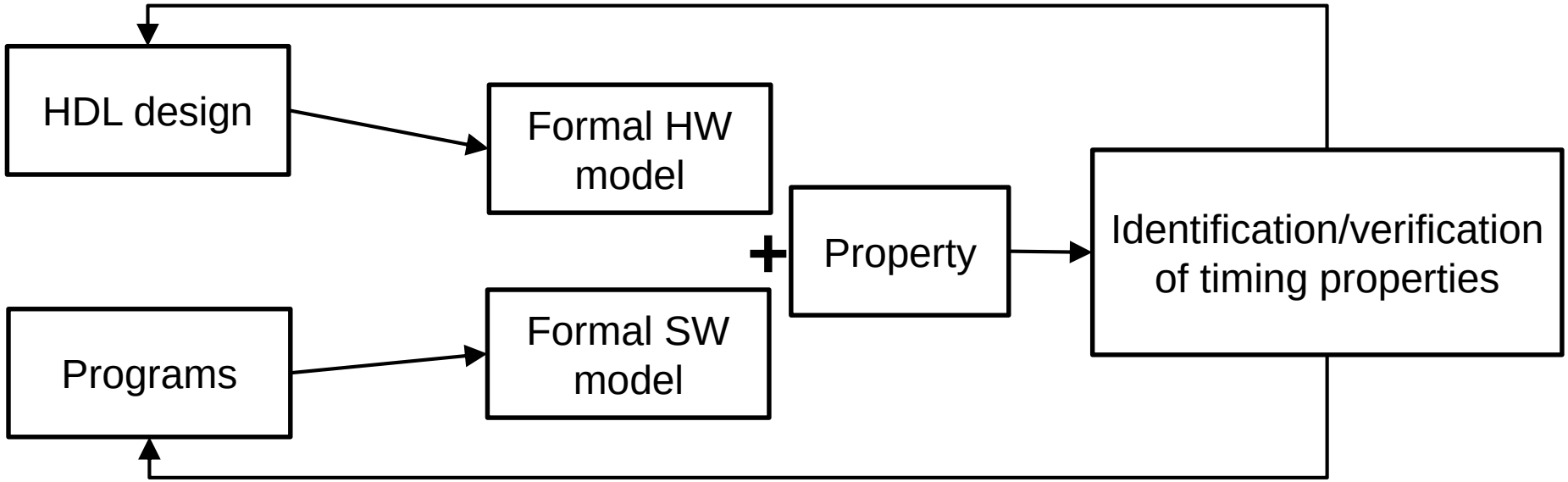


Execution constraints:

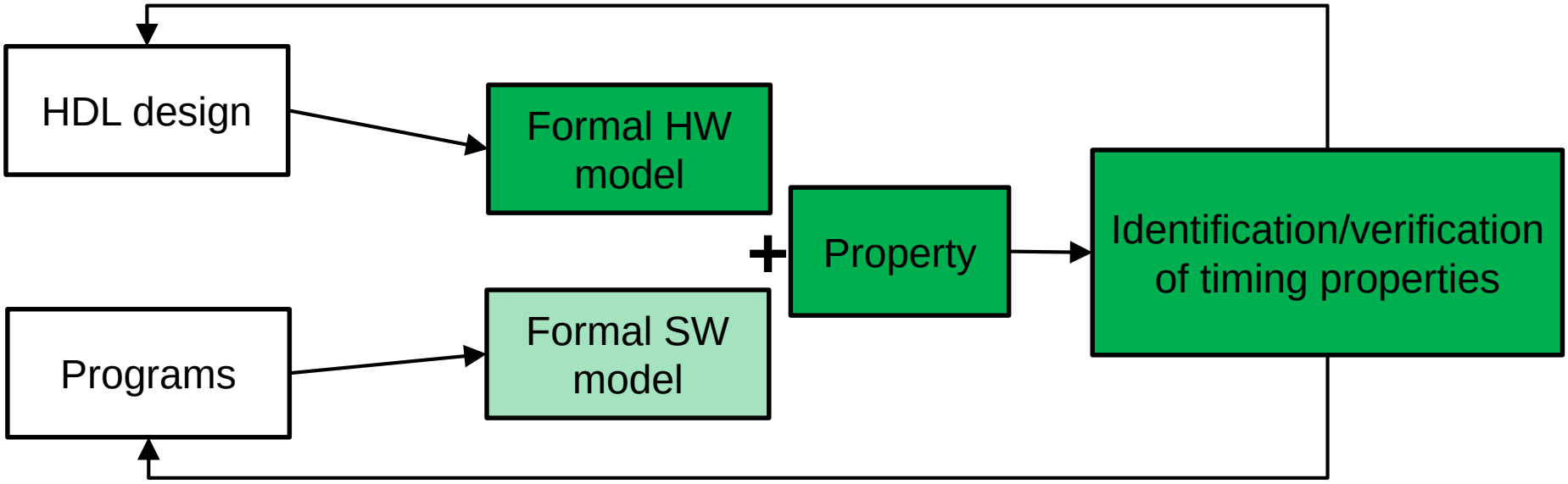
A - {FU1}	A - {1, 3}	A - {}
B - {FU2}	B - {3}	B - {A}
C - {FU2}	C - {3}	C - {}
D - {FU1}	D - {3}	D - {C}

Detection of anomalies: build or check?

- **Goal: code-specific detection of timing anomalies**
  - Build (code-independent) suffers from obvious complexity and scalability issues [DDECS 2006]
  - Mitigation possibilities (reduction)



Contributions



- **Counter-intuitive timing anomalies**
  - Over in-order pipeline [WCET2018] using TLA+
  - **Over out-of-order pipeline [ERTS2020] using TLA+**

Microsoft Research

- **Amplification timing anomalies**
  - A set of predictable pipelines [ASP-DAC2020] using UCLID

Berkeley UNIVERSITY OF CALIFORNIA

Formal modeling language: TLA+

- **“TLA+ is a language for modeling software above the code level and hardware above the circuit level” - L. Lamport**

- **TLA+ models a system as a set of behaviors (sequences of states) describing all the possible executions**
  - An initial condition Init, to specify the possible starting states
  - A next-state relation Trans, to specify all possible pairs of successive states (e.g. `x' = x + 1`)

```
Spec == Init /\ [] [Trans]_vars
```

  - Predicate logic
  - Modeling checking (TLC) support

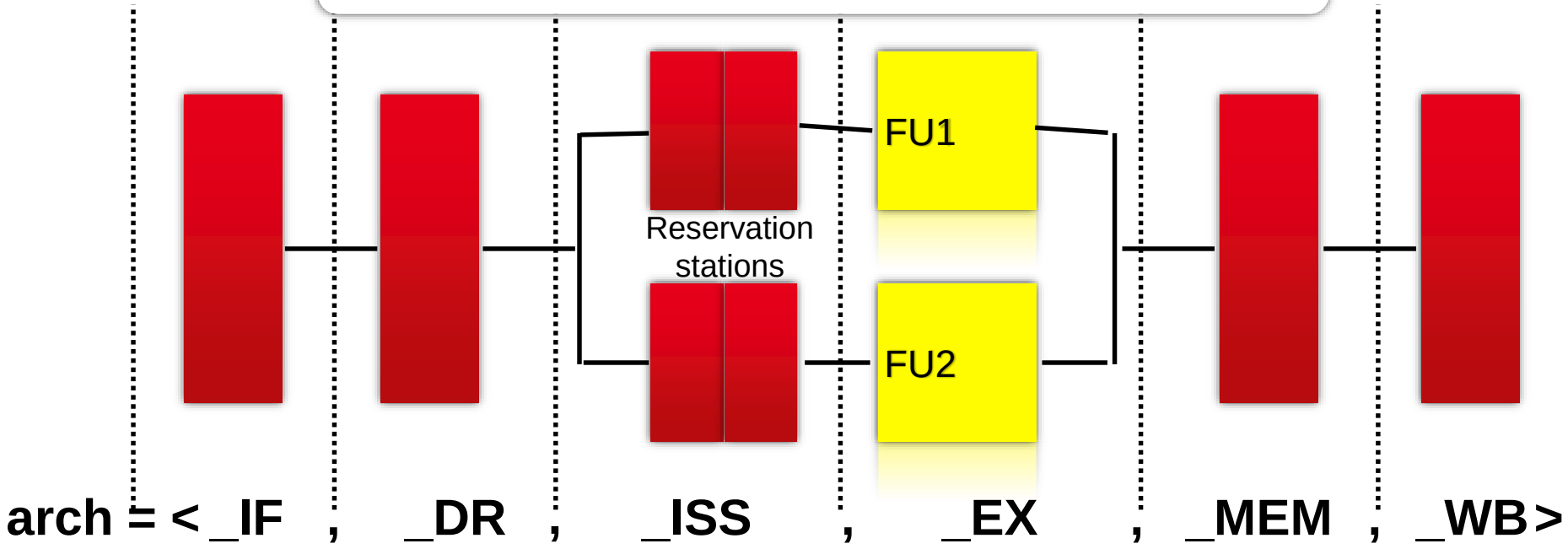
Pipeline model

Hardware

out-of-order
6-stages,
dual-issue
pipeline

MODULE BUFF with interface *Set* and *Reset*

MODULE FU with interface *Acquire* and *Release*

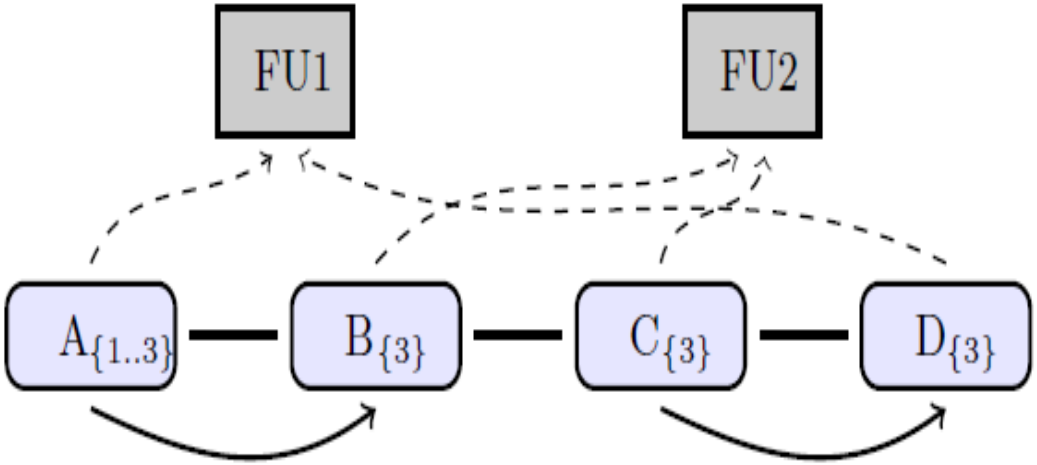


- **Full specification of the EX stage**
  - Acquire / release of functional units
  - Cycle-accurate and deterministic

Input program model

Software

- **Instruction representation**
  - Program counter
  - Set of functional units
  - Set of data dependencies
  - Set of instruction timing variation (latencies)



Acquire/release logic: acquire FU1

Hardware

- **Choose latencies**

$$\wedge\ \neg\, emptyISS\ \wedge\ emptyFU1$$

$$\wedge\ isInstrRdyFU1\ \wedge \neg\, isInstrRdyFU2$$

$\wedge\, condAcquireFU1$

$\wedge$ IF $twoIns$

THEN $\exists\ w1 \in code.currIns[1].lat$:

$\exists\ w2 \in code.currIns[2].lat$:

_IF' = $update2IF\ (\_IF,\ code.currIns[1],\ w1,\ code.currIns[2],\ w2)$

ELSE _IF' = $resetIF\ (\_IF)$

...

$\wedge$ _EX' = [_EX EXCEPT!.$fu1 = FU1!Acquire(\_ISS.rs\_f1[1].instr, currCycle)$]

Acquire/release logic: release FU1

Hardware

- **Tomasulo algorithm**
  - **Data dependencies resolution are sent over specific bus**

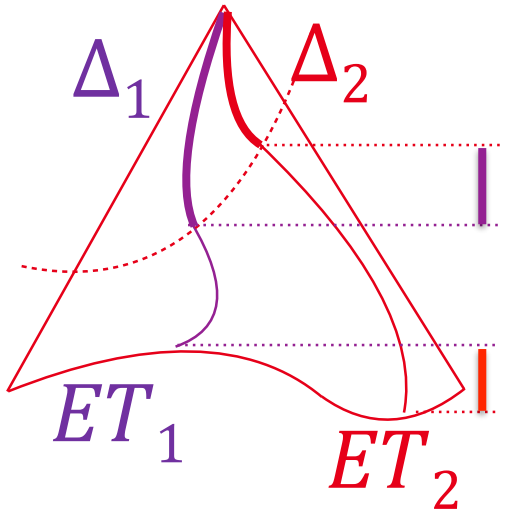
$$
\begin{array}{l}
\wedge\ \textbf{IF}\ Cardinality\ (allDependsF2\ (\_EX.fu1.instr.pc)) > 0 \\
\quad \textbf{THEN}\ \forall\ i \in allDependsF2\ (\_EX.fu1.instr.pc) : \\
\qquad \textbf{\_ISS'}\ =\ [\ \textbf{\_ISS}\ \textbf{EXCEPT}!.rs\_f2[i].instr.dep = @ \setminus \{\_EX.fu1.instr.pc\}\ ] \\
\quad \textbf{ELSE}\ \textbf{\_ISS'}\ =\ updateISS\ (\textbf{\_ISS}) \\
\hline
\wedge\ \textbf{\_EX'}\ =\ [\ \textbf{\_EX}\ \textbf{EXCEPT}!.fu1\ =\ FU1!Release\ ] \\
\hline
\wedge\ \textbf{\_MEM'}\ =\ [\ \textbf{\_MEM}\ \textbf{EXCEPT}!.buff[1] = MBUFF!Set(\_EX.fu1.instr, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \_EX.fu1.instr.lat)\ ] \\
\hline
\end{array}
$$

  - **When no more dependencies → update ISS**

Verification: property overview & results

$$Spec == Init \wedge \Box Trans_{\langle State \rangle}$$

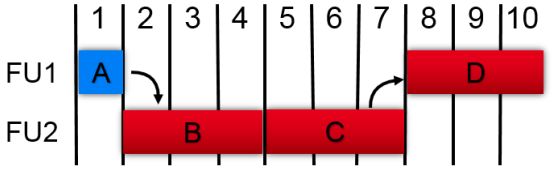
$$State = \langle \text{Hardware}, \text{Software} \rangle$$

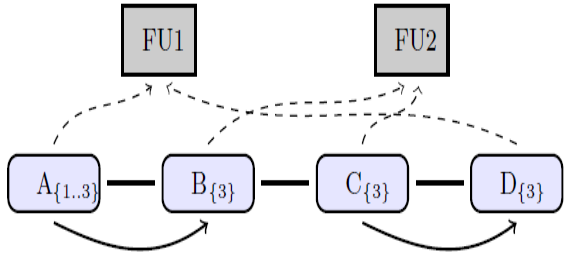


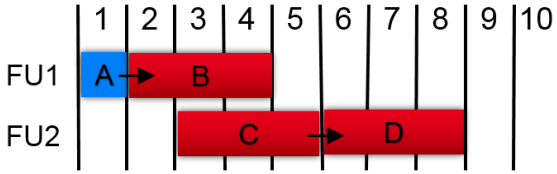
- **Detection of timing anomalies with TLC model-checker (LTL property)**

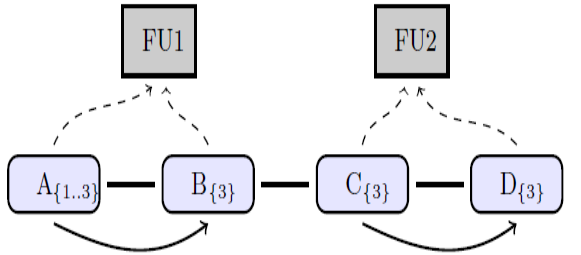
$$Prop == \Box \neg (\Delta_1 > \Delta_2 \wedge \mathrm{ET}_1 < \mathrm{ET}_2)$$

- **Modified program shown to be proved without timing anomalies**









Conclusion and on-going work

- **Extension of our automatic detection of timing anomalies for out-of-order architectures**
  - Counter-intuitive scheduling timing anomalies
  - Formal modeling in TLA+
  - Verification with TLC: LTL property
- **Combine with the detection of amplification timing anomalies**
- **Automatically build abstract formal HW models from HDL languages**