Modélisation des systèmes matériel/logiciel multiprocesseur, et méthodes d'annotations pour l'estimation de performance

Frédéric Pétrot

Avec l'aide de :
Aimen Bouchhima, Nicolas Fournel, Patrice Gerin et Marius Gligor

Laboratoire TIMA,
Grenoble

Outline

MPSoC Perspectives

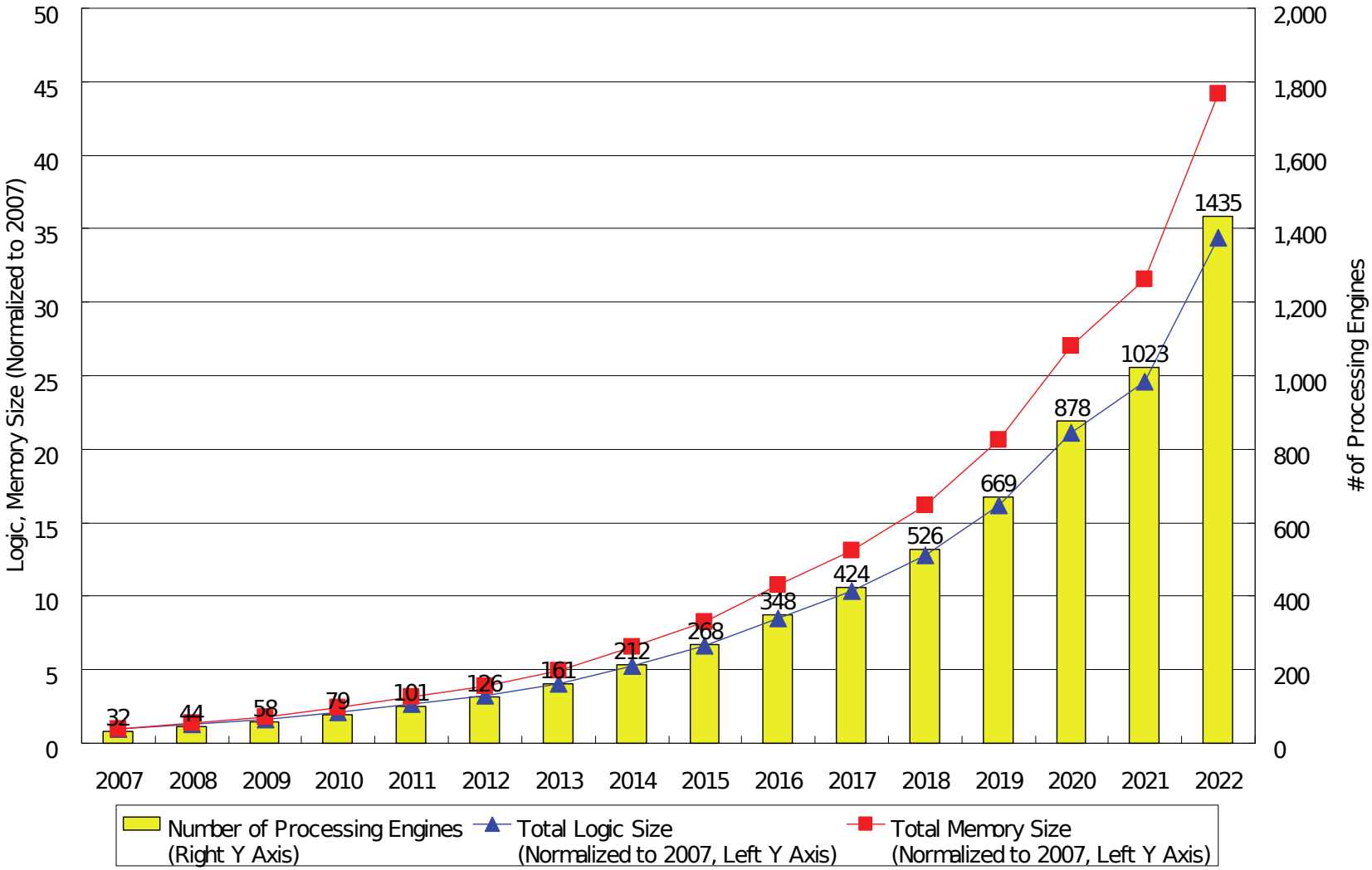


Figure: ITRS Roadmap for the number of cores in consumer devices

Example of Future SoCs Architecture

Characteristics

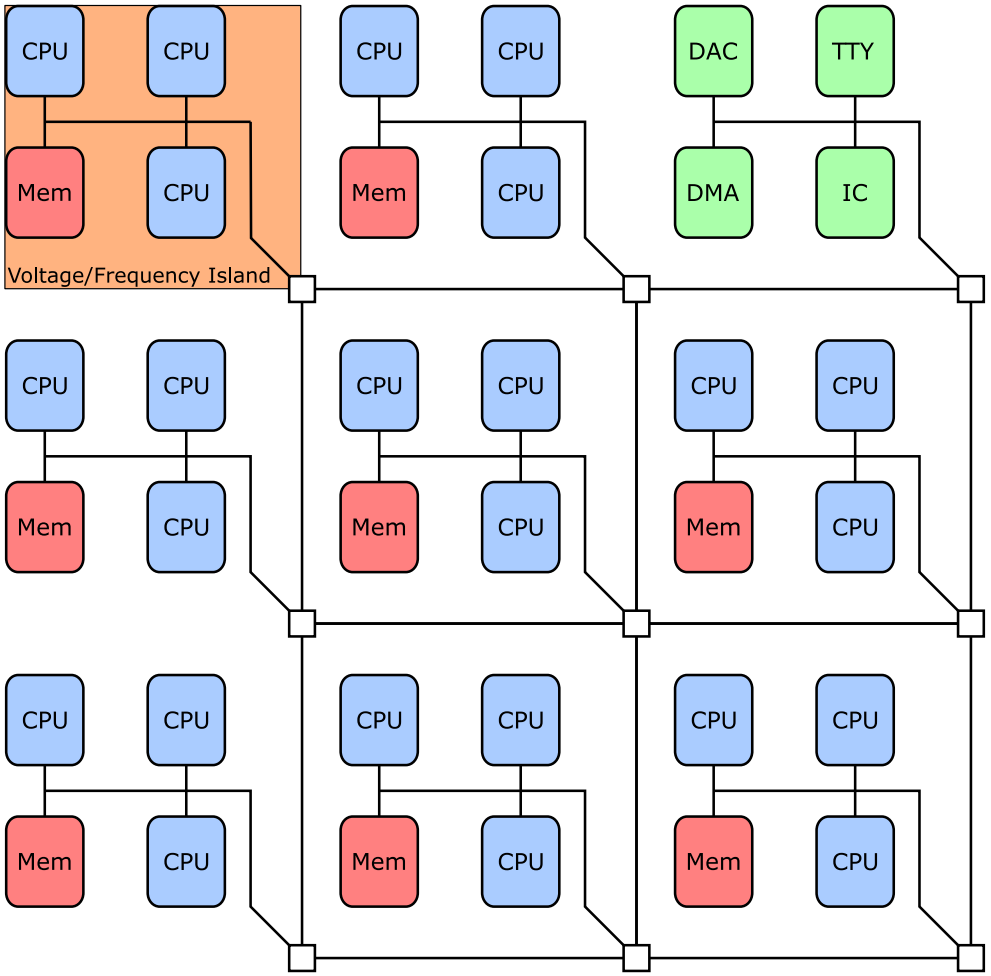
- Massively parallel
- Single address space
- More and more programmable

Properties

- Very redundant

$\Rightarrow$ Still usable with faults

$\Rightarrow$ Cost effective



Motivations

Design space exploration

- Goal: optimize chips for performances (time and energy)

Requirements

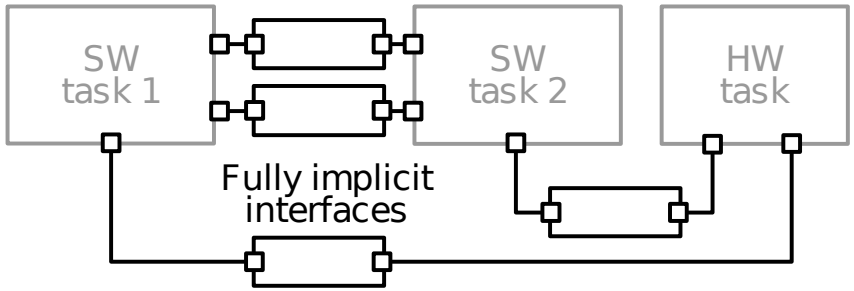
- Energy estimations: $E = \int_0^T Pdt$
- Good E estimations requires good T estimations
- Need of a precise simulation

Drawback

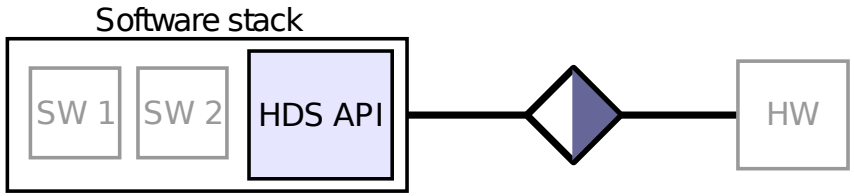
- Precision influences simulation speed (trade-off)
- Simulation speed is however a great concern

MPSoCs: Level of Abstractions

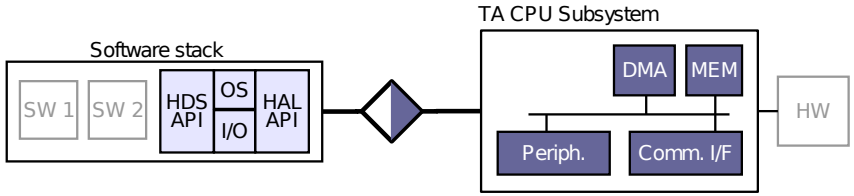
System



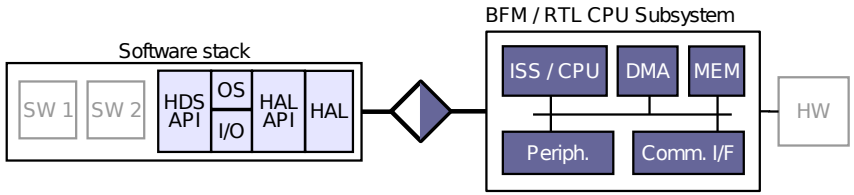
Virtual architecture



Transaction accurate



Cycle approximate / accurate



ISS Technologies

Current status:

Interpretive ISS

- Pros: flexible and precise
- Cons: low simulation speed

Binary translation based ISS

- Pros: fast
- Cons: problems with time precision !!!

Compiled simulation based ISS

- Pros: very fast (native execution)
- Cons: not flexible, no time precision

Hardware simulation technology

SystemC

- C++ classes library
- Support for modeling and simulation of electronic systems
- Different levels of abstraction: sytem level, TLM level, cycle accurate level, RTL
- Discrete event-driven simulation
  - Concurrency of processes
- Basic notions: modules, signals, ports, processes

Outline

Cycle Accurate Simulation

Targets

- Digital embedded systems
- Synchronous – possibly multiple clock domains – systems
- Rely on existing IPs: CPU cores, ASICs, RTK
- Large systems: from 10 to 100 cores

What is to be simulated?

- Hardware components
- Software programs
- Interconnect

Cycle Accurate Simulation

Bit true, cycle true and/or cycle approximate

- Exact interfaces
  Exact computations
  Direct link to physical implementation
  Direct comparison with physical implementation
- Interfaces must be fully specified
  Protocol must be fully defined
  More complex models
  Slower simulation: CPU is simulated, ASIC implement a bit true/cycle true communication protocol

Cycle Accurate Simulation

Simulation model

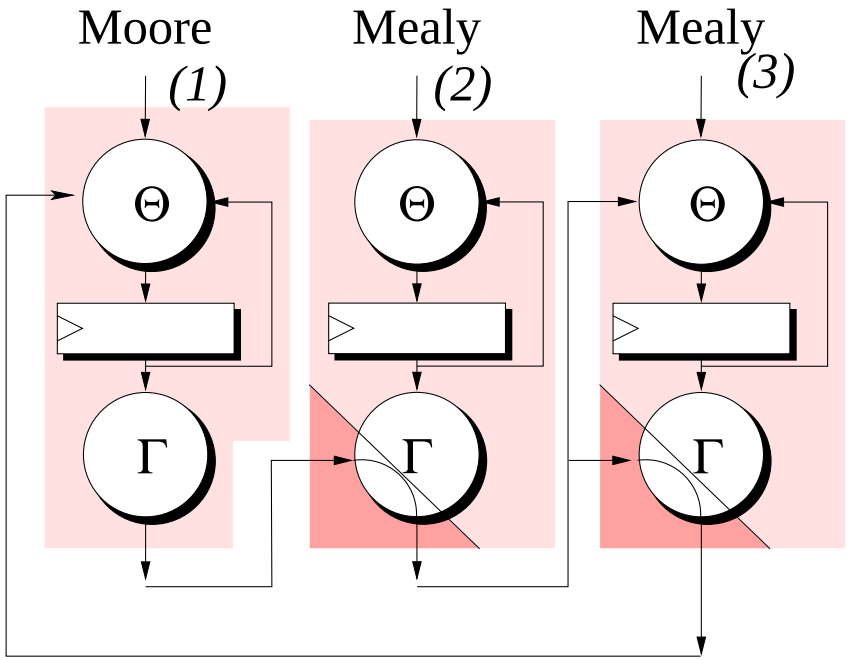
- System: Set of communicating Finite State Machines (FSMs)
- FSM :$\{I, O, S, \Theta, \Gamma\}$, $\{\mathcal{I}, \mathcal{O}, \mathcal{S}\}$ encodings of $\{I, O, S\}$

$$\underline{FSM_i} \begin{cases} \mathcal{I}_i^{t+1} \subseteq \mathcal{O}_1^t \cup \mathcal{O}_2^t \cup \cdots \cup \mathcal{O}_n^t & \\ \mathcal{S}_i^{t+1} \underset{ck}{\Leftarrow} \Theta_i(\mathcal{I}_i^t, \mathcal{S}_i^t) & \\ \mathcal{O}_i^{t+1} = \Gamma_i(\mathcal{S}_i^{t+1}) & \text{Moore} \\ \quad\quad = \Gamma_i^s(\mathcal{S}_i^{t+1}) & \\ \mathcal{O}_i^{t+1} = \Gamma_i(\mathcal{I}_i^{t+1}, \mathcal{S}_i^{t+1}) & \text{Mealy} \\ \quad\quad = \Gamma_i^s(\mathcal{S}_i^{t+1}) \cup \Gamma_i^c(\mathcal{I}_i^{t+1}, \mathcal{S}_i^{t+1}) & \end{cases}$$

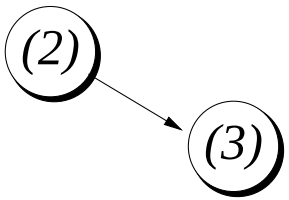
- To simulate such a system:
  - Moore $\Rightarrow$ run in parallel
  - Mealy $\Rightarrow$ event driven simulation or relaxation

Cycle Accurate Simulation

Static Ordering :

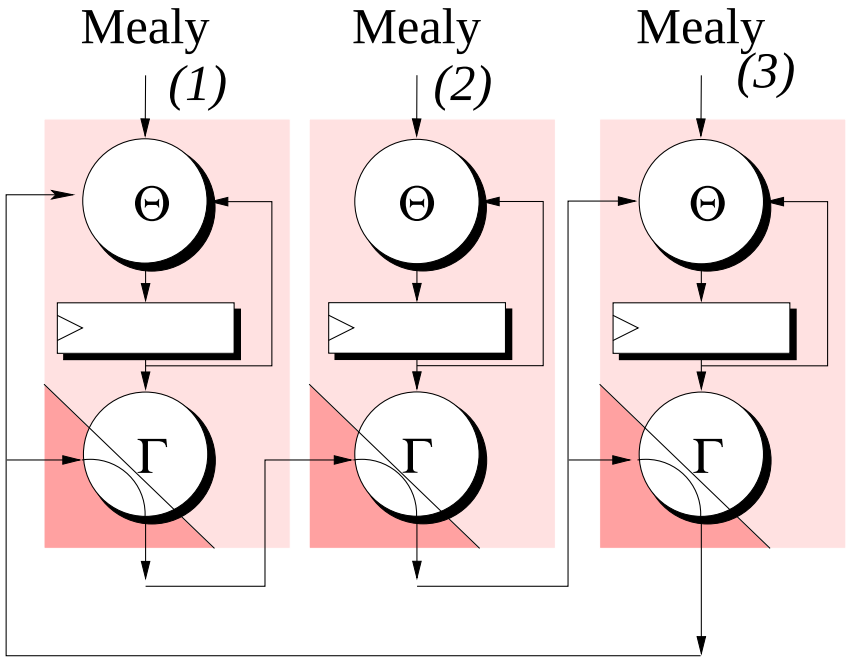


Corresponding CIC graph

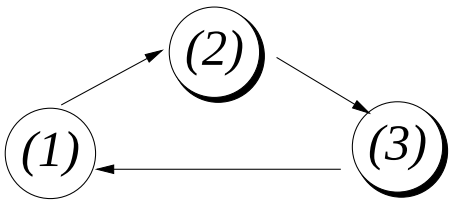


Cycle Accurate Simulation

Dynamic Evaluation :



Corresponding CIC graph



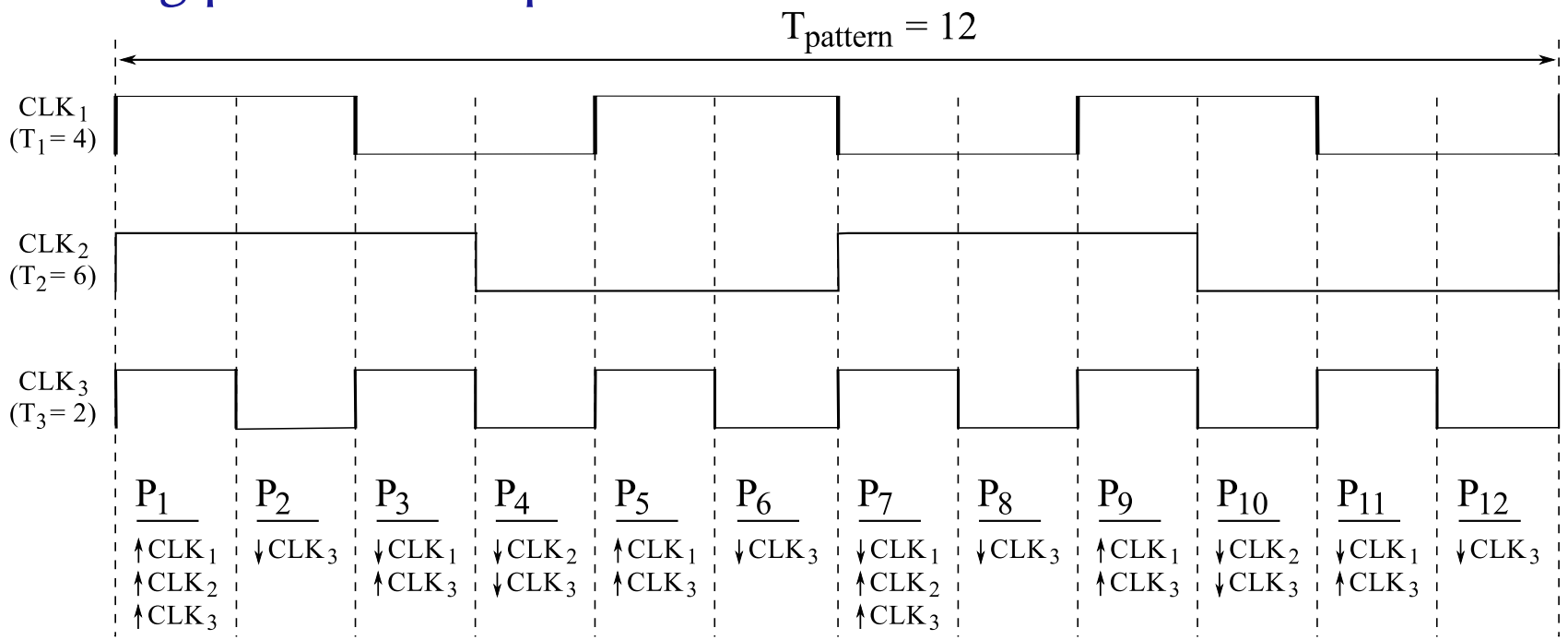
Uses bounded relaxation for combinational convergence

Multiple clocks Approach

Characteristics

- Uses multiple clock components
  - Dependency on non clock signals not allowed
- Main idea: scheduling pattern with the period equal to the least common multiple (LCM) of the clocks periods
- Simulation point: contains at least a clock edge

Scheduling pattern example



Details

Implementation & execution

- Processes are sorted
  - A list of Transition and a list of Moore functions for each clock
  - These lists don't change during the simulation
- A list of simulation points
  - A list of positive and a list of negative clock edges
  - Time offset to be added to pass to next simulation point
- Execution of a simulation point

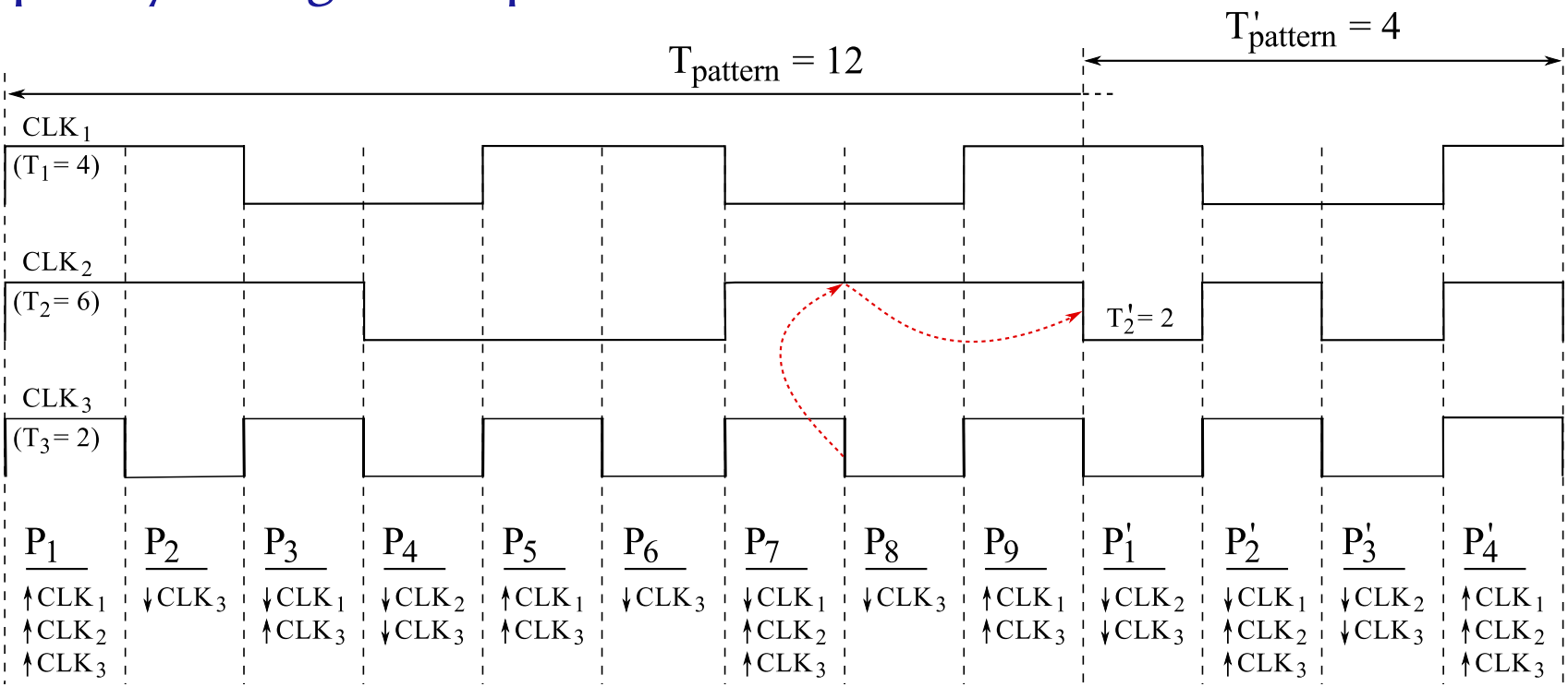
```
void simulation_point ()
{
    transition_functions_simpoint ();
    moore_functions_simpoint ();
    update_registers ();

    stab_all_mealy_functions ();
}
```

Run-time Change of the Frequencies

Characteristics

- A new API added to the clock component: change_period
- Scheduling pattern recomputed
- Frequency changes at the next edge of the target clock

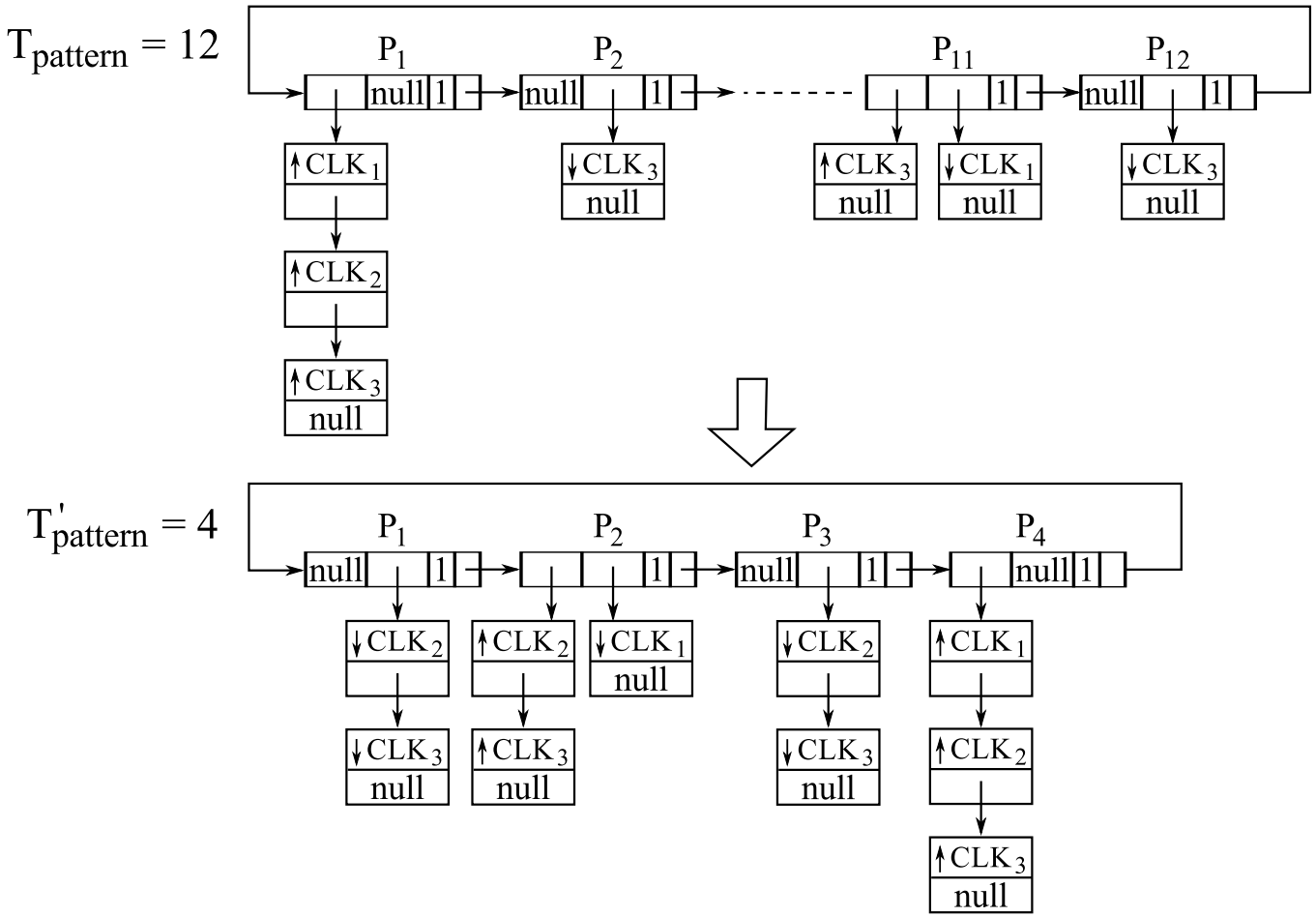
Frequency change example



Details

Run-time frequency change

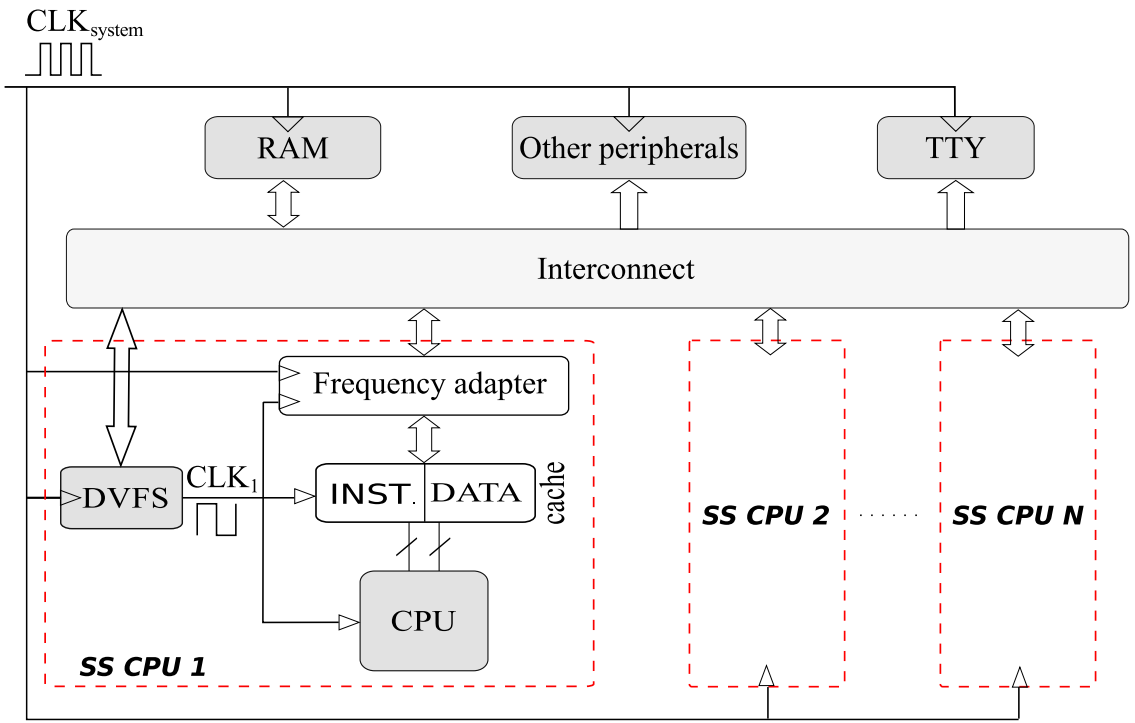
- Scheduling pattern recomputed
- Takes time
  - Number of simulation points in the new scheduling pattern



Experimental Results

Architecture example

- The Interconnect, RAM, TTY etc. use the frequency of the system clock
- The frequency of each processor and its caches generate by a clock component



Experimental Results

Hardware architecture

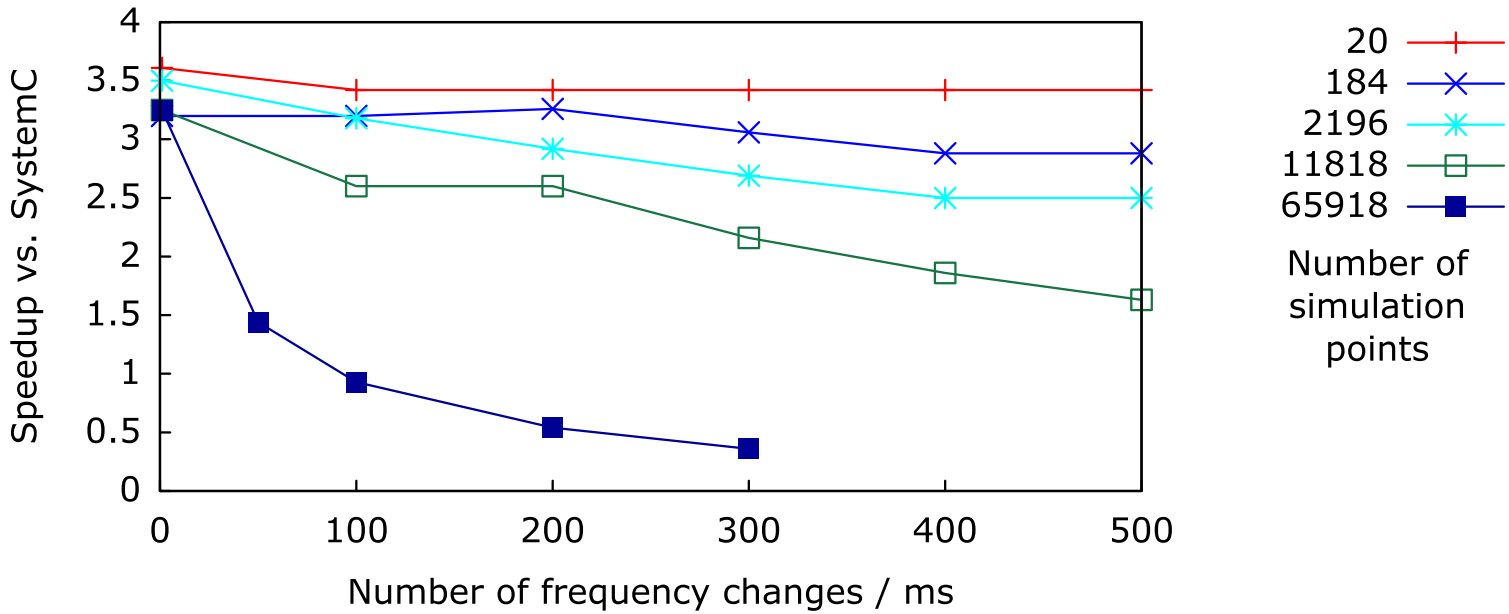
- SoCLib (`www.soclib.fr`) components
- Compiled and linked with 3 simulators
  - SystemC - reference OSCI implementation
  - SystemCASS
  - Multiple clock based static scheduling simulator

Simulator	Simulation speedup vs. SystemC	
	1 frequency	Multiple frequencies
SystemCass	3.50 X	NA
Multiple clocks	3.53 X	3.81 X

Experimental Results

Multiple clock simulator

- Simulation speedup (synthetic benchmarks)
  - Number of simulation points in scheduling pattern
  - Number of frequency changes



Outline

Simulation using Dynamic Binary Translation

Speed $\Rightarrow$ Simulation at transaction level

Promising Solution

Combining use of dynamic binary translation technology and event-driven simulation technology for MPSoC platform simulation

Possible candidates

- Binary translation based ISS: **QEMU**, Bochs, ...
- Event driven simulator: **SystemC** is the current solution for MPSoC simulation

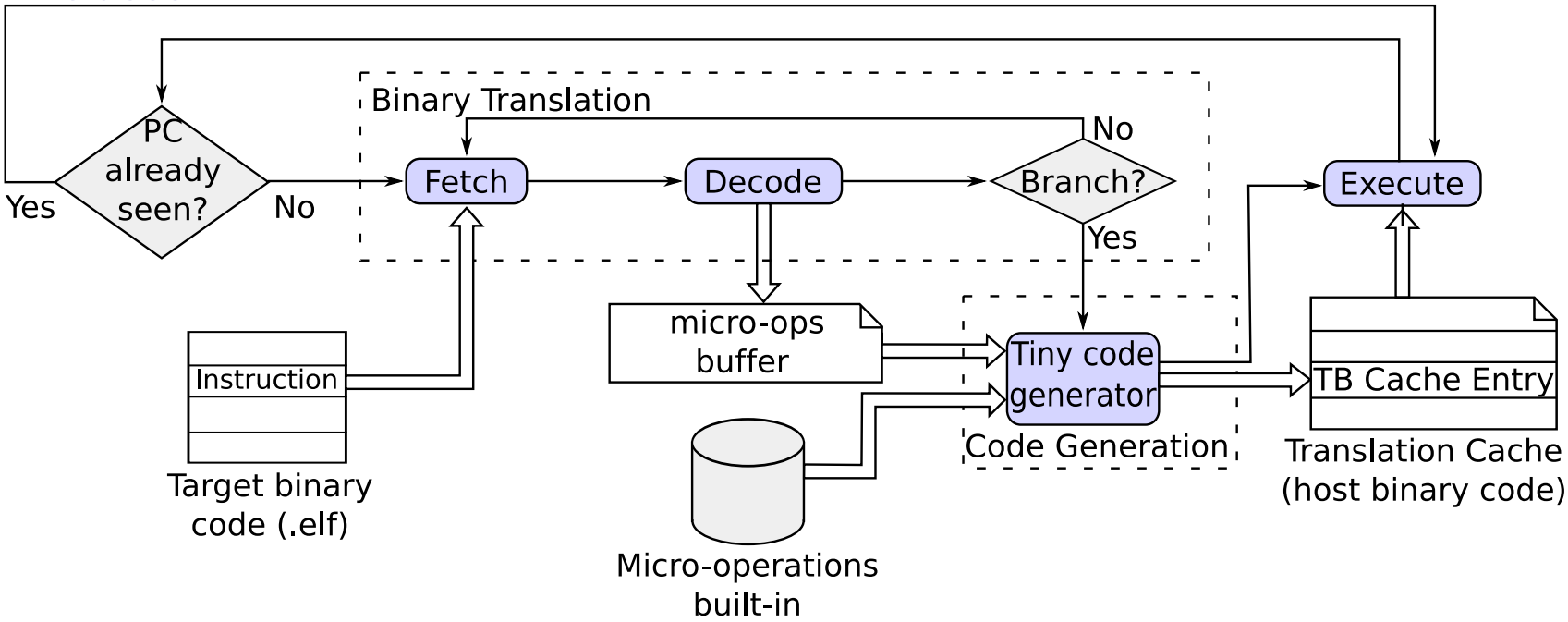
QEMU in a few words

QEMU

- Fast and portable emulator
- Emulates multiple architures: e.g. x86, ARM, SPARC, ...
- Based on dynamic binary translation and dynamic code generation
- 5 to 20 times slower than native code execution

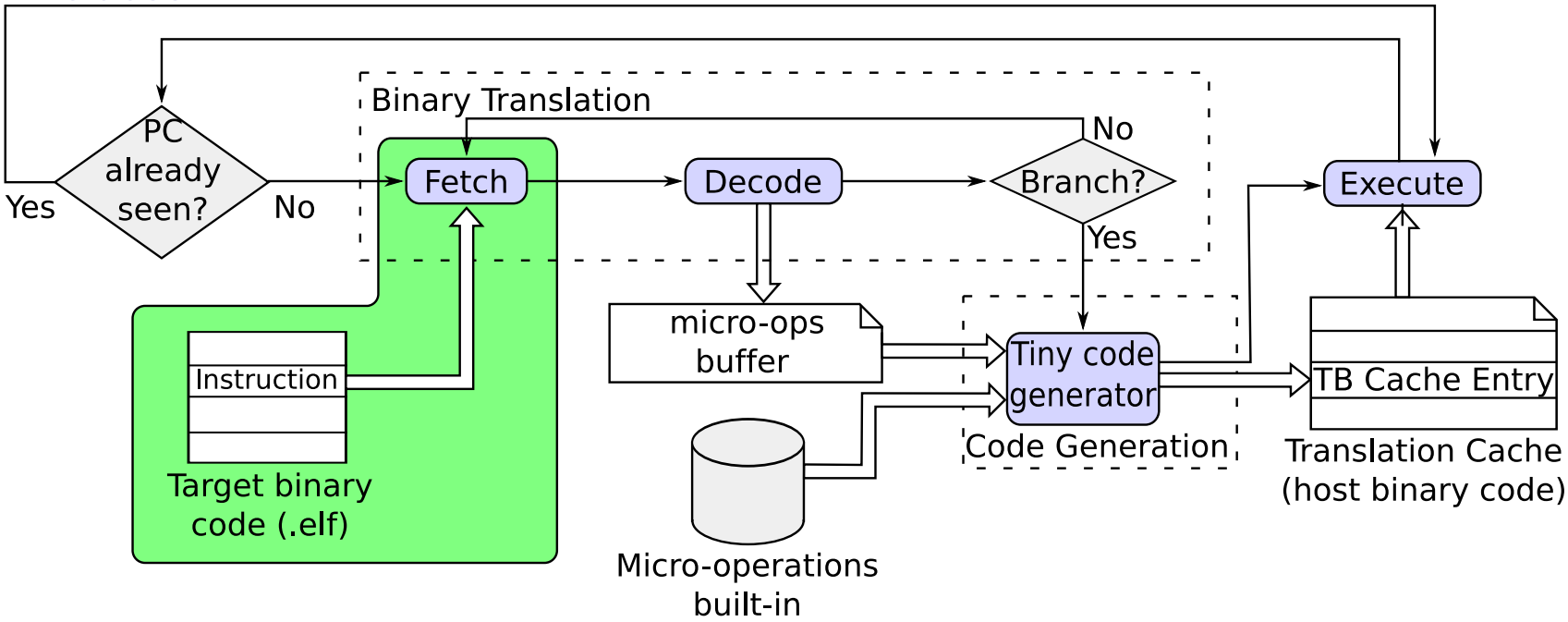
QEMU Emulation Process

Process



QEMU Emulation Process

Process

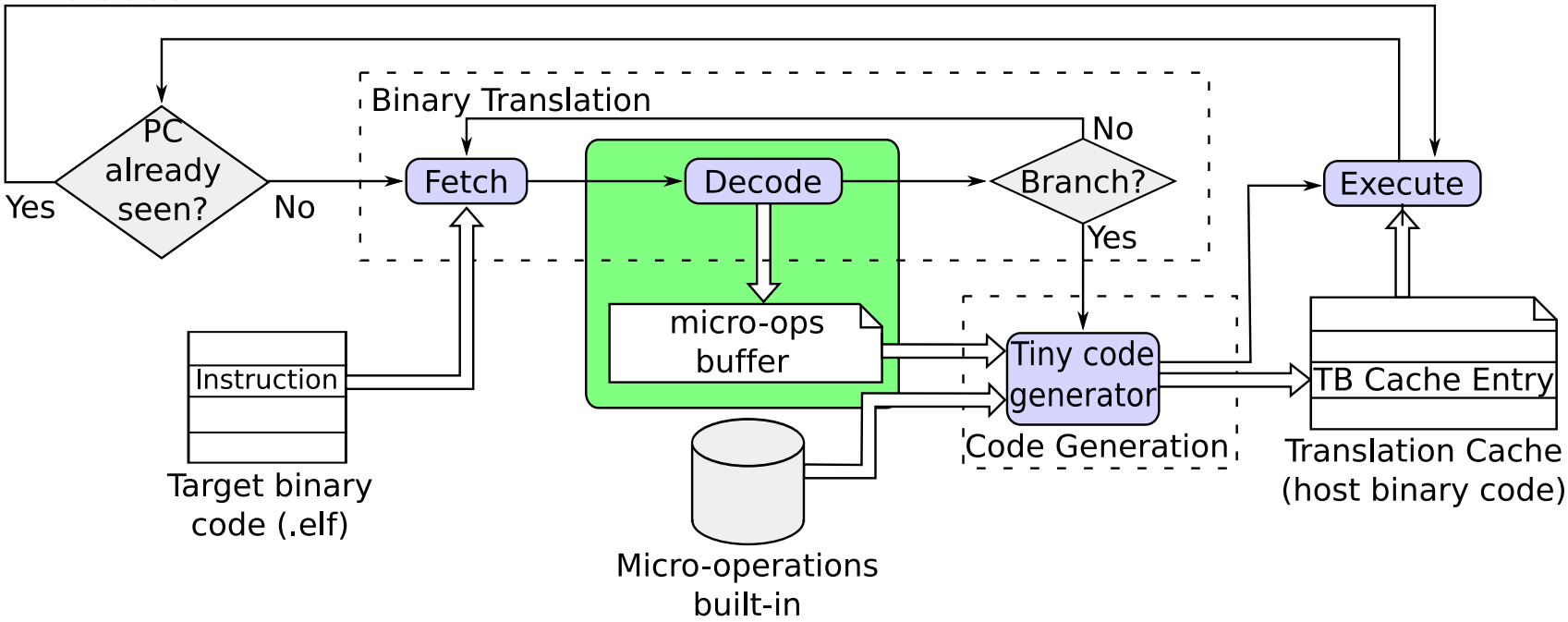


Code generation example

```
18    instrX_target
```

QEMU Emulation Process

Process



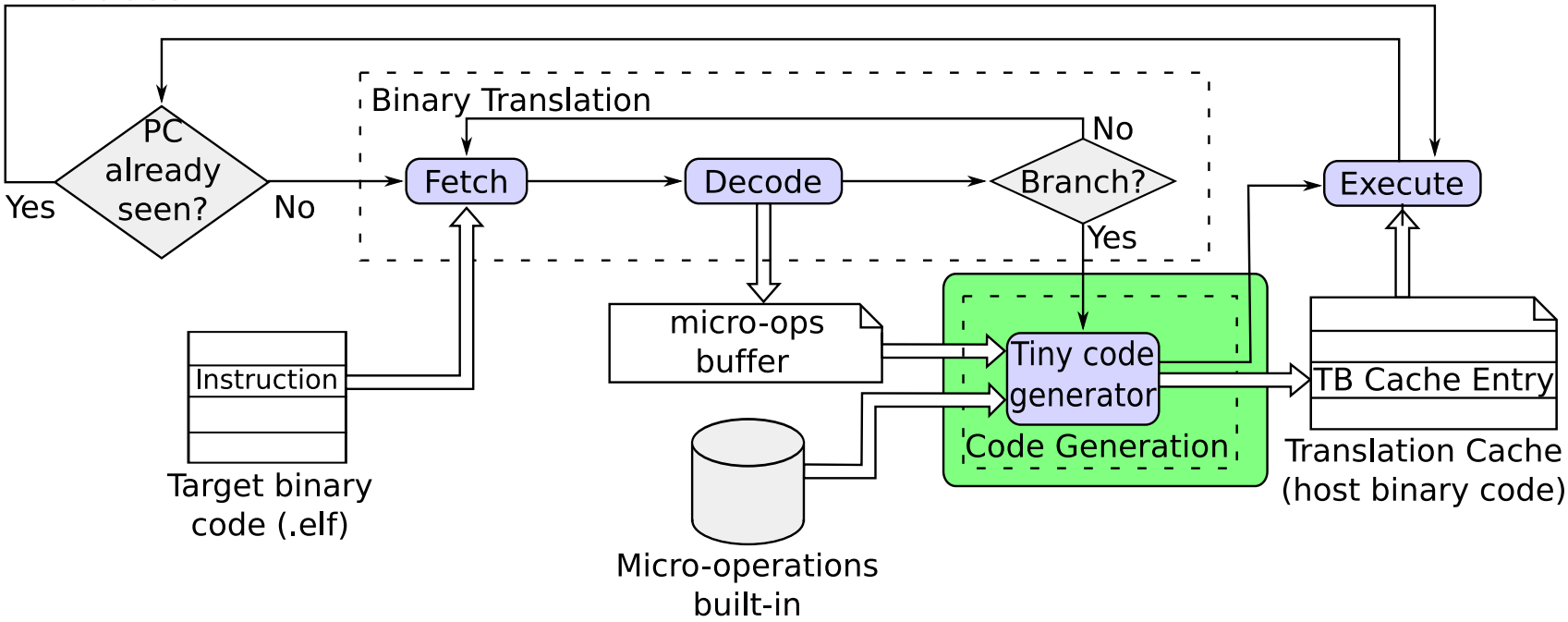
Code generation example

18 instrX_target micro-op1_instrX

micro-op2_instrX

QEMU Emulation Process

Process



Code generation example

18	instrX_target	micro-op1_instrX	host_instr1_micro-op1_instrX
			host_instr2_micro-op1_instrX
			host_instr3_micro-op1_instrX
		micro-op2_instrX	host_instr1_micro-op2_instrX

QEMU and QEMU-SystemC Platforms Comparison

QEMU platform

- Simulates the processors in a "circular" way
- Fast and functionally correct
- No time notion for the simulated platform
  - Uses the host time

QEMU-SystemC platform

- Simulates the processors concurrently
- Functionally correct, time approximative, slower simulation
- SystemC time
  - Enables DFS techniques
- Execution scheme
  - QEMU executes a generated code sequence (0 SystemC time)
  - Consumes the time due to this sequence execution with a SystemC `wait()`

QEMU-SystemC Platform Architecture

SystemC wrapper: QEMU platform

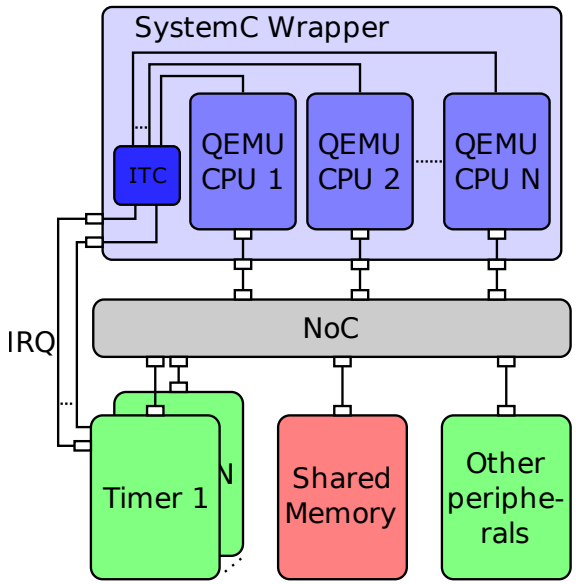
- Models the common aspects of the platform: e.g. interrupt controllers
- Contains a SystemC wrapper for each processor
- Connected to interconnect, through which communicate with other SystemC hardware components

SystemC wrapper: processors

- Simulates independently under the SystemC control
- Accesses the SystemC components by mapping ranges of physical addresses as I/O (except main memory)

SystemC TLM components

- Traffic generator, timers, main memory, spinlocks, interconnect, RAMDAC, TTYs



QEMU/SystemC Synchronization

Synchonization points

- Cache misses (instruction and data caches)
- I/O operations (uncached registers and memories accesses)
- QEMU normal processor simulation breaks e.g. interrupt handling
- Predefined period of time without synchronization

Interrupts

- Generated by hardware components during the SystemC activities of the processors
- Interrupt pending flags set during the SystemC activity of the processors
- Flags viewed by QEMU at its activity resume
- Treated at the beginning of the next translation block

Precision Levels

- Simulation speed/accuracy trade-off
- No caches
- Caches as pure directories
  - QEMU memory used
  - Two different possibilities varing on the time consuming scheme
    - Cache late: precomputed time consumed at the next synchronization
    - Cache wait: precomputed time consumed when a miss occurs
- Caches full
  - SystemC memory used
  - Search data and instructions over the interconnect
  - Ignore instructions from cache (instructions are simulated from QEMU translation cache)

Code Annotation: Principles

Motivation

Retrieve time precision on the the binary translated code

Insert micro-operations to

- Increment the number of cycles according to the datasheets. Need to take into account registers, data, branch prediction dependencies
- Emulate caches (instruction and data) and write buffer

Annotation example:

Instr address	Target code	Original translation	Annotated translation	Annotated generated code
addr_instr1	target_instrX	micro-op1_instrX	micro-op1_instrX	host_instr1_micro-op1_instrX
				host_instr2_micro-op1_instrX
				host_instr3_micro-op1_instrX
		micro-op2_instrX	micro-op_annotation	host_instr1_micro-op_annotation
				host_instr2_micro-op_annotation
			micro-op2_instrX	host_instr1_micro-op2_instrX

Code Annotation: Details

Instruction Cache

- Where?
  - At the beginning of each translation block
  - At the beginning of each cache line
- What?
  - Synchronize simulated cycles
  - Request over the interconnect

Data cache

- Where?
  - Before each data access (read and write)
- What?
  - On read miss: synchronize and fill the cache line using the interconnect
  - On write hit: update the value in cache
  - On write: update the value in memory through interconnect (write through policy)

Code Annotation: Example

- Assumptions: the cache lines have 8 words (32 bytes)

	Instr address	Target code	Original generated code	Annotated generated code
start_tb:	18	instr1_reg_operation	host_instr1_for_instr1 ..... host_instrN1_for_instr1	instr_cache_verify (18); nb_cycles += cpu_datasheet [instr1]; host_instr1_for_instr1 ..... host_instrN1_for_instr1
	1C	instr2_load_from_1000	host_instr1_for_instr2 ..... host_instrN2_for_instr2	nb_cycles += cpu_datasheet [instr2]; data_cache_verify (1000); host_instr1_for_instr2 ..... host_instrN2_for_instr2
	20	instr3_store_5_to_2000	host_instr1_for_instr3 ..... host_instrN3_for_instr3	instr_cache_verify (20); nb_cycles += cpu_datasheet [instr3]; write_access (2000, 5); host_instr1_for_instr3 ..... host_instrN3_for_instr3

Energy

At this point, we annotate the generated code to

- Account static instruction timings
- Generate dynamic timings at execution

Precise time estimation is possible

As for power estimation

- Static information is annotated in the same way than timing one.
- Dynamic information consists in: operating mode and frequency

Dynamic power estimations

- Mode is not a hard issue: resolved at execution
- DVFS is more complicated
  - DVS only change the power consumption base: resolved at execution
  - DFS interfere in timings estimation only

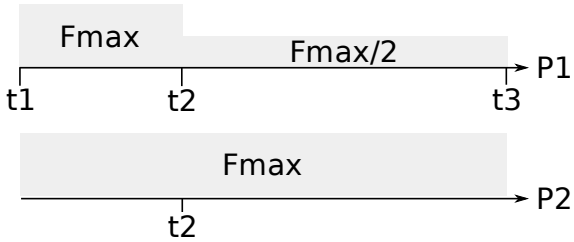
DFS Handling

2 types of times

- Dependent on the processor frequency (instruction cycles)
- Not dependent on the processor frequency (communications)

Characteristics

- Each processor has its DFS
- A processor can change the frequency of other processors
  - Frequency can change between two synchronization points



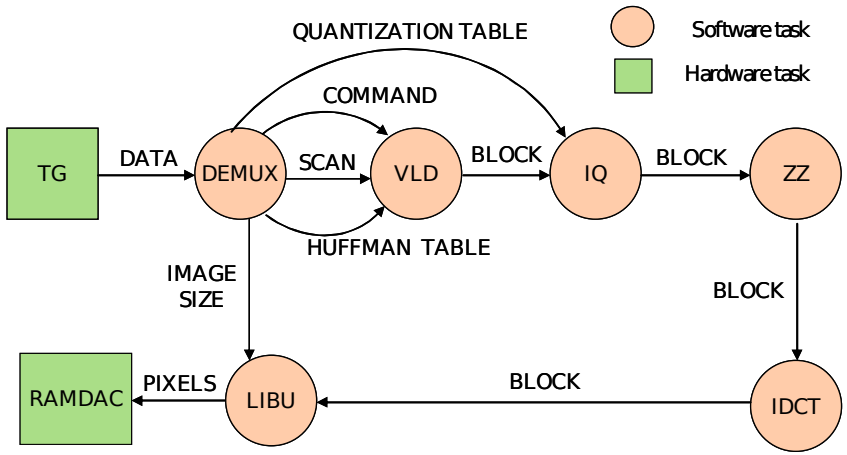
Synchronization implementation

- Based on the current frequency, compute the time corresponding to the number of cycles to synchronize (t = nb_cycles/fq)
- Call the SystemC wait function for waiting the computed time and the frequency change event
- If the frequency has changed, the remained cycles are computed and the algorithm repeats

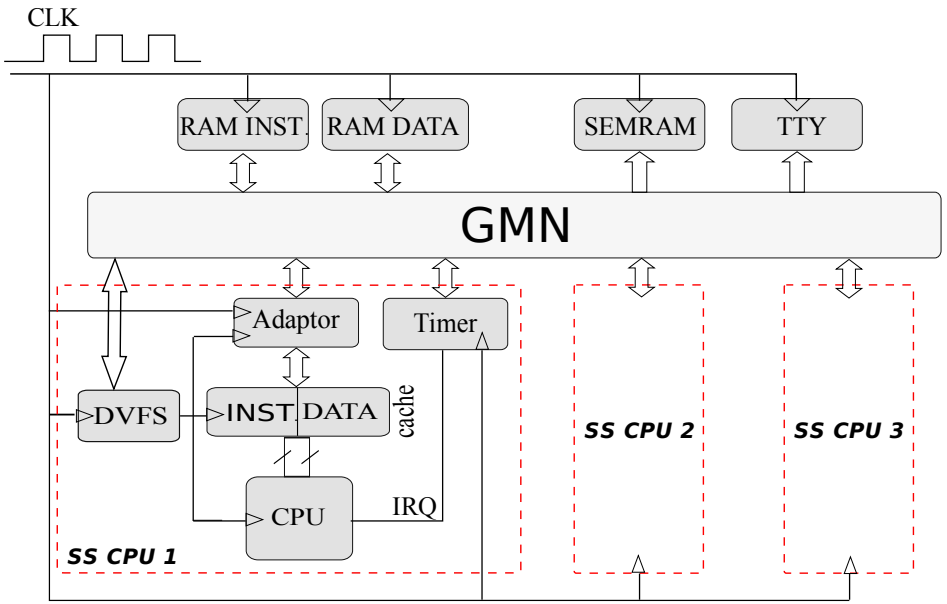
Motion-JPEG Application

Software stack

- Motion-JPEG decoding application
- Mutek operating system
  - POSIX compliant
  - SMP version



Hardware platform



Simulations

Virtual Platforms

- CABA Reference: modeled with ISS/SoClib/SystemCASS
- Transaction level: modeled with QEMU/SystemC

Different precision level

- No Cache: caches not implemented, QEMU memory used
- Cache late: directory caches and time consumed at next sync point
- Cache wait: directory caches and time consumed on miss
- Cache full: full caches and data fetched from the SystemC modeled memory

Accuracy

Table: Monoprocessor results

	SOCLIB	No cache (%)	Cache late (%)	Cache wait (%)	Cache full (%)
Instructions	24114066	-0.00	0.00	0.00	0.00
Cycles instr.	31303545	-0.00	0.00	0.00	0.00
Instr. cache misses	290428	-100.00	0.00	0.00	0.00
Data cache misses	240517	-100.00	-1.58	-1.58	-1.58
Write accesses	3895150	-100.00	-0.00	-0.00	-0.00
Uncached accesses	89820	-100.00	0.00	0.00	0.00
Cycles sim. ($*10^3$)	50635	-36.70	-0.04	-0.04	-0.04
Sim. speedup	1	553.10	356.13	55.39	28.55

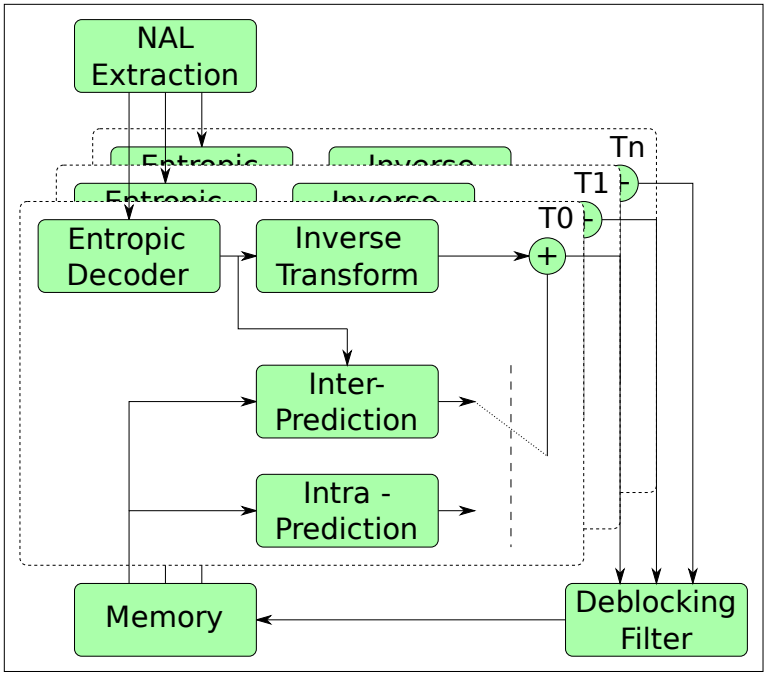
Table: Multiprocessor results

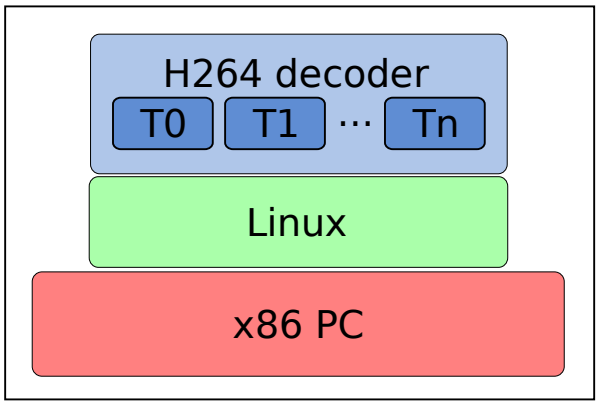
	SOCLIB	No cache (%)	Cache late (%)	Cache wait (%)	Cache full (%)
Instructions	25331336	35.13	22.31	5.24	6.28
Cycles instr.	32931244	34.53	22.01	5.44	6.45
Instr. cache misses	238916	-100.00	-1.42	2.30	2.09
Data cache misses	261273	-100.00	-32.94	-30.57	-31.05
Write accesses	4019613	-100.00	22.23	5.72	7.04
Uncached accesses	169614	-100.00	-9.80	-2.01	-10.54
Cycles sim. ($*10^3$)	19020	-21.07	1.34	-8.44	4.19
Sim. speedup	1	381.01	246.38	35.97	17.76

H264 Application

Software characteristics

- Parallelism exploitation: multithread implementation (SPMD)
- Standard operating system: Linux

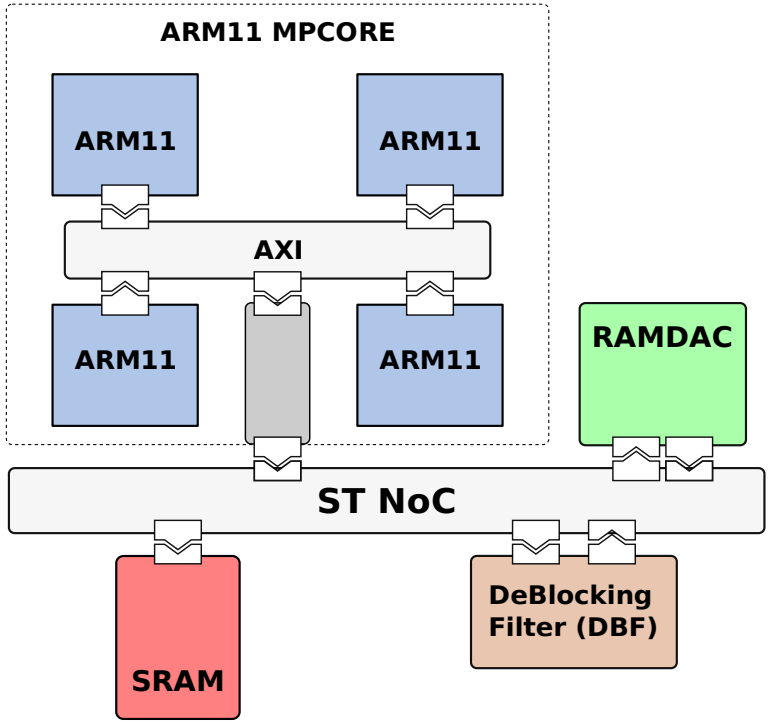




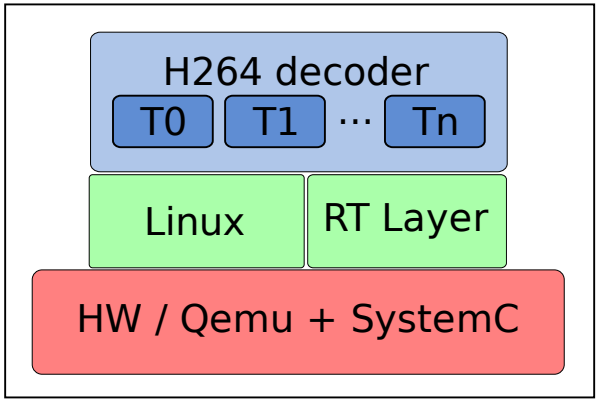
Implementation

Hardware architecture

- NoC based platform : Spidergon STNoC
- ARM 11MPCore processors: 4 Symmetric CPUs
- Specific hardware component: deblocking filter



Software architecture



Simulation results

- Utilization: percentage of frame rate period time needed to decode a frame in average
- Power: average power of the system during the processing

	1 thread		2 threads		3 threads		4 threads	
	Util.	Power	Util.	Power	Util.	Power	Util.	Power
Soft DBF	176.59	697	91.56	841	64.53	832	63.90	821
HW DBF	114.08	680	72.77	716	57.68	727	48.62	740
DVFS	114.08	681	80.22	628	73.17	507	72.62	509

Simulation speed

- Linux boot: **31s** (against **3.3h** estimated for CABA)
- Frame decoding: **2s** (against **12.7m** estimated for CABA)

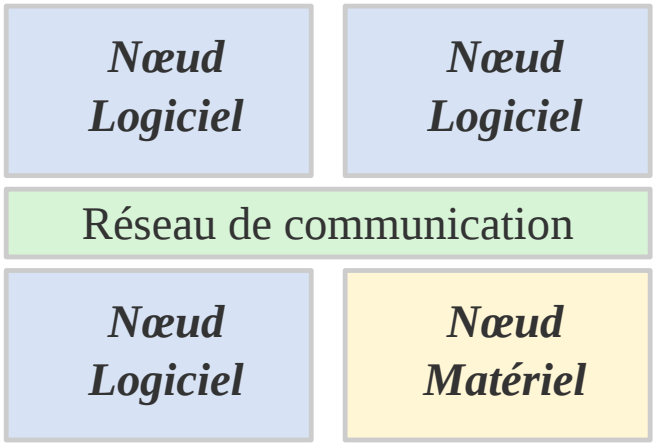
Outline

Simulation Native

Impose des contraintes de développement du logiciel

Définitions

- Nœuds matériels
- Nœuds logiciels
- Réseau de communication



Architecture d'un nœud logiciel

- Le sous-système processeur
  - Architecture SMP
    $\Rightarrow$ mêmes processeurs, même logiciel
- Le logiciel
  - Reposant sur la couche HAL
    $\Rightarrow$ Hardware Abstraction Layer
  - Unique moyen d'accéder au matériel

Simulation Native

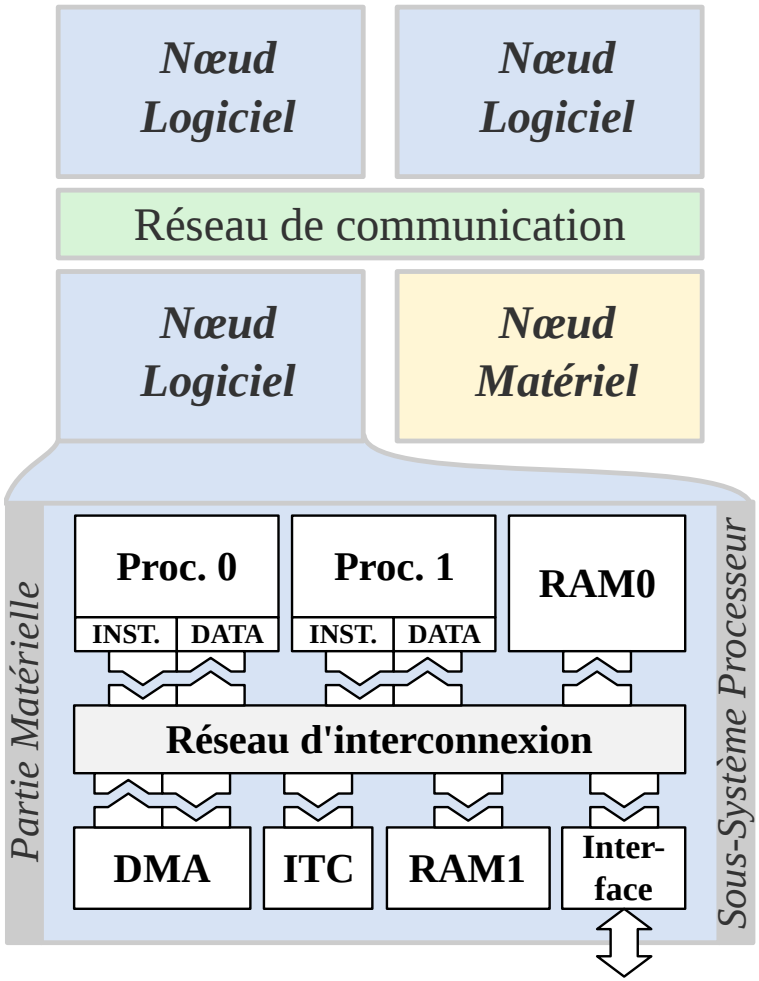
Impose des contraintes de développement du logiciel

Définitions

- Nœuds matériels
- Nœuds logiciels
- Réseau de communication

Architecture d'un nœud logiciel

- Le sous-système processeur
  - Architecture SMP
    $\Rightarrow$ mêmes processeurs, même logiciel
- Le logiciel
  - Reposant sur la couche HAL
    $\Rightarrow$ Hardware Abstraction Layer
  - Unique moyen d'accéder au matériel



Simulation Native

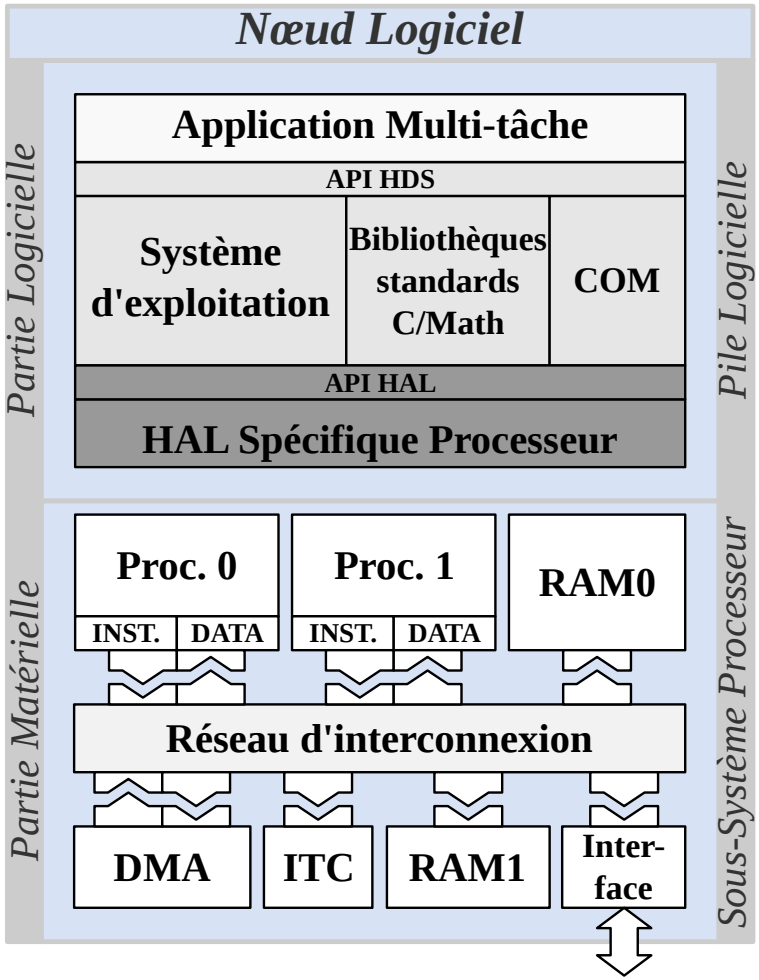
Impose des contraintes de développement du logiciel

Définitions

- Nœuds matériels
- Nœuds logiciels
- Réseau de communication

Architecture d'un nœud logiciel

- Le sous-système processeur
  - Architecture SMP
    ⇒ mêmes processeurs, même logiciel
- Le logiciel
  - Reposant sur la couche HAL
    ⇒ Hardware Abstraction Layer
  - Unique moyen d'accéder au matériel



Les niveaux CA et TLM classiques

Partie Matérielle

- Fidèle à l'architecture finale
- Plus abstraites en TLM
- Processeurs cibles modélisés par des ISS

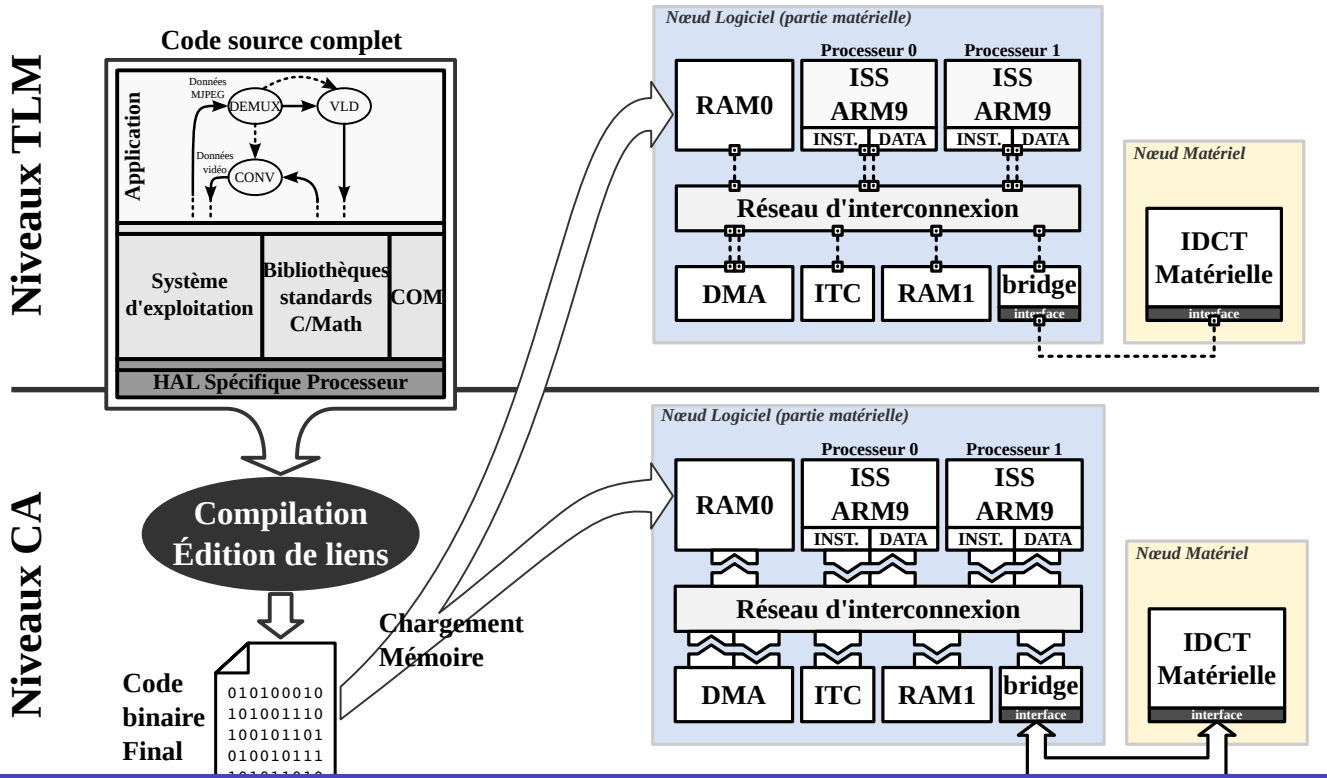
Partie Logicielle

- Fidèle au logiciel final
- Toute la pile logicielle est exécutée
- Binaire interprété par des ISS

✔ Bonne précision

✘ Simulation lente (nombreux ISS)

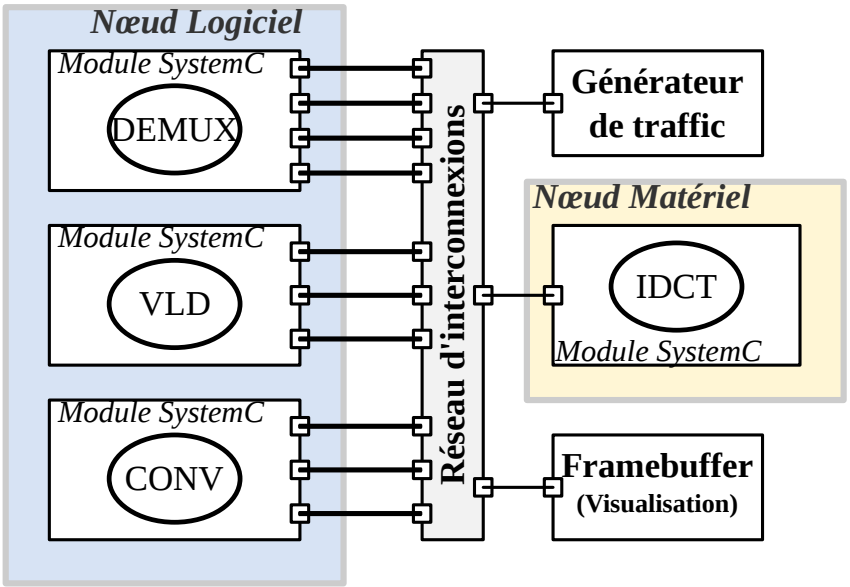
✘ Non adaptés à la validation logicielle



Le niveau TLM natif

Partie Matérielle

- Seulement certains aspects
- Pas d'ISS du processeur cible:
  Celui de la station de travail
  $\Rightarrow$ Le processeur hôte



Partie Logicielle

- Certaines tâches de l'application
- Encapsulées dans des modules matériels
- Fait partie du modèle matériel:
  Exécuté par le processeur hôte
  $\Rightarrow$ La simulation native

✔ Bonnes performances

✘ Trop proche d'un modèle fonctionnel

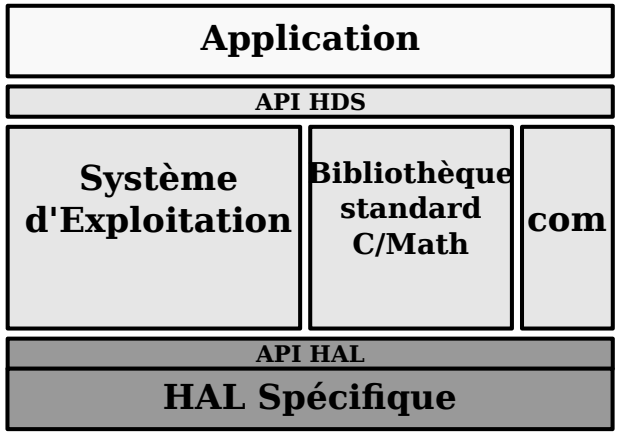
✘ Simulation matérielle

✘ Aucun aspect du logiciel

Le modèle Transaction Accurate

Dérivé des approches TLM natives

- Plus d'encapsulation du logiciel
- Repose sur les interfaces logicielles



Principe de base

- Le logiciel indépendant du processeur cible peut être compilé pour le processeur hôte

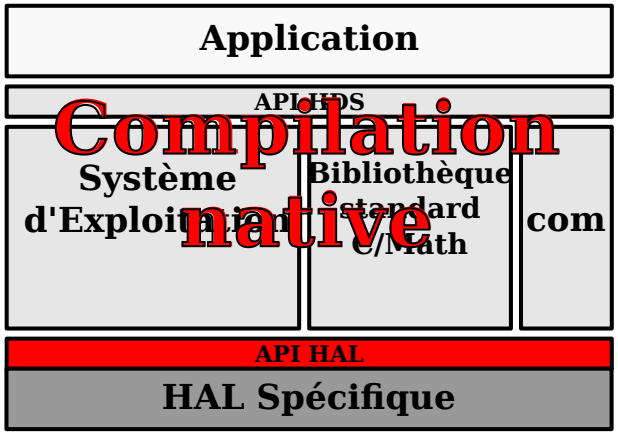
$\Rightarrow$ Tout le logiciel au dessus de l'API HAL

- La couche HAL est implémentée par le modèle de la plateforme matérielle

Le modèle Transaction Accurate

Dérivé des approches TLM natives

- Plus d'encapsulation du logiciel
- Repose sur les interfaces logicielles



Principe de base

- Le logiciel indépendant du processeur cible peut être compilé pour le processeur hôte

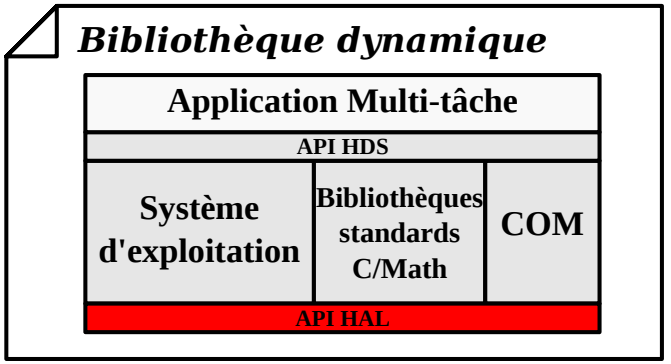
$\Rightarrow$ Tout le logiciel au dessus de l'API HAL

- La couche HAL est implémentée par le modèle de la plateforme matérielle

Le modèle Transaction Accurate

Dérivé des approches TLM natives

- Plus d'encapsulation du logiciel
- Repose sur les interfaces logicielles



Principe de base

- Le logiciel indépendant du processeur cible peut être compilé pour le processeur hôte

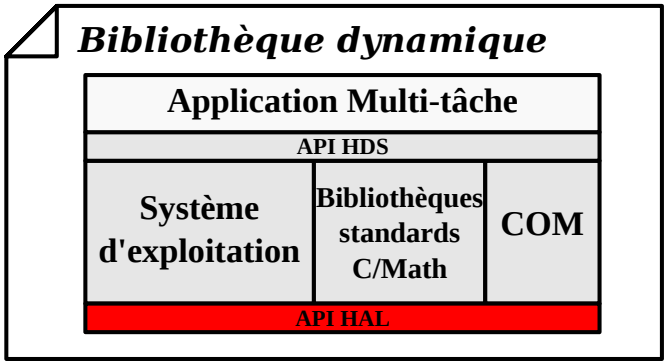
$\Rightarrow$ Tout le logiciel au dessus de l'API HAL

- La couche HAL est implémentée par le modèle de la plateforme matérielle

Le modèle Transaction Accurate

Dérivé des approches TLM natives

- Plus d'encapsulation du logiciel
- Repose sur les interfaces logicielles

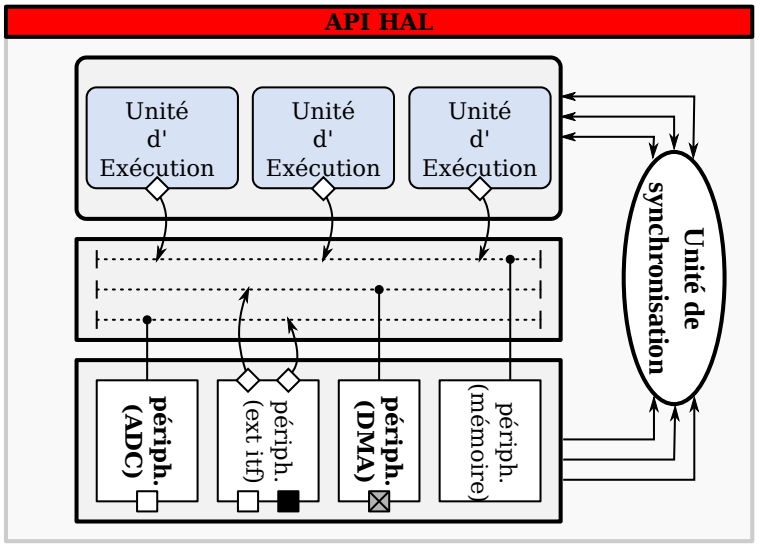


Principe de base

- Le logiciel indépendant du processeur cible peut être compilé pour le processeur hôte

⇒ Tout le logiciel au dessus de l'API HAL

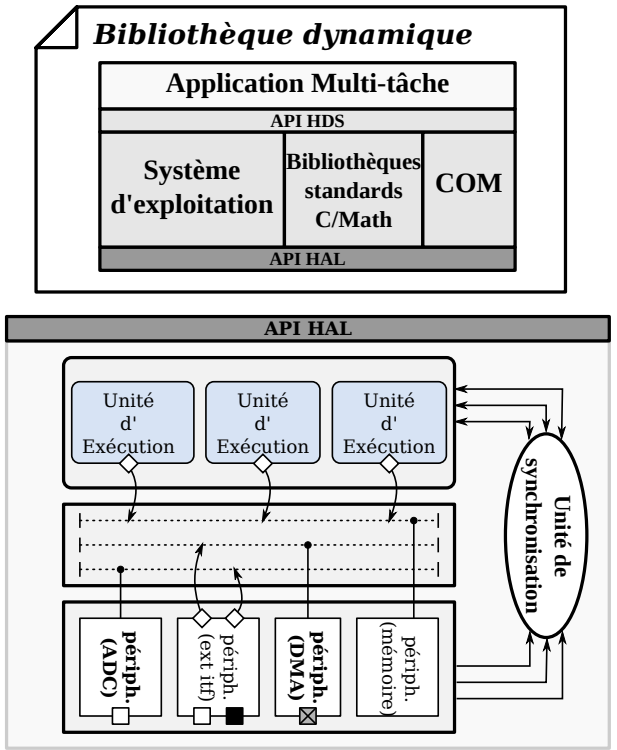
- La couche HAL est implémentée par le modèle de la plateforme matérielle



Problématiques liée à l'exécution native

Deux espaces mémoire

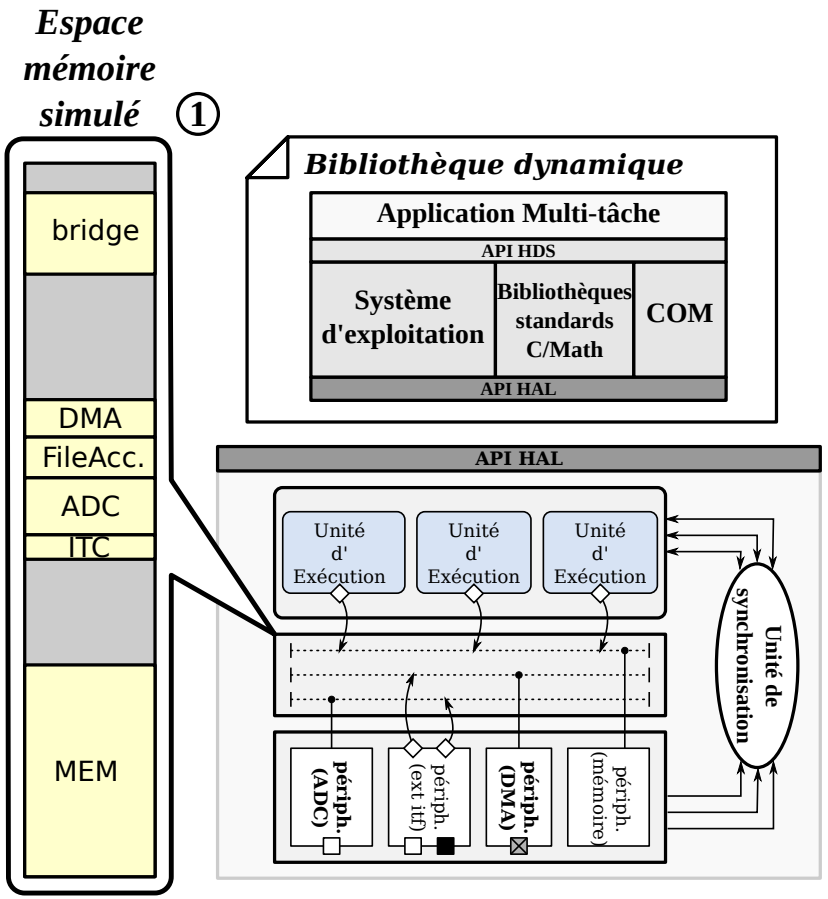
1. Espace mémoire simulé par la plateforme matérielle
   - Défini par les concepteurs
2. Espace mémoire hôte vu par le logiciel
   - Défini à la compilation



Problématiques liée à l'exécution native

Deux espaces mémoire

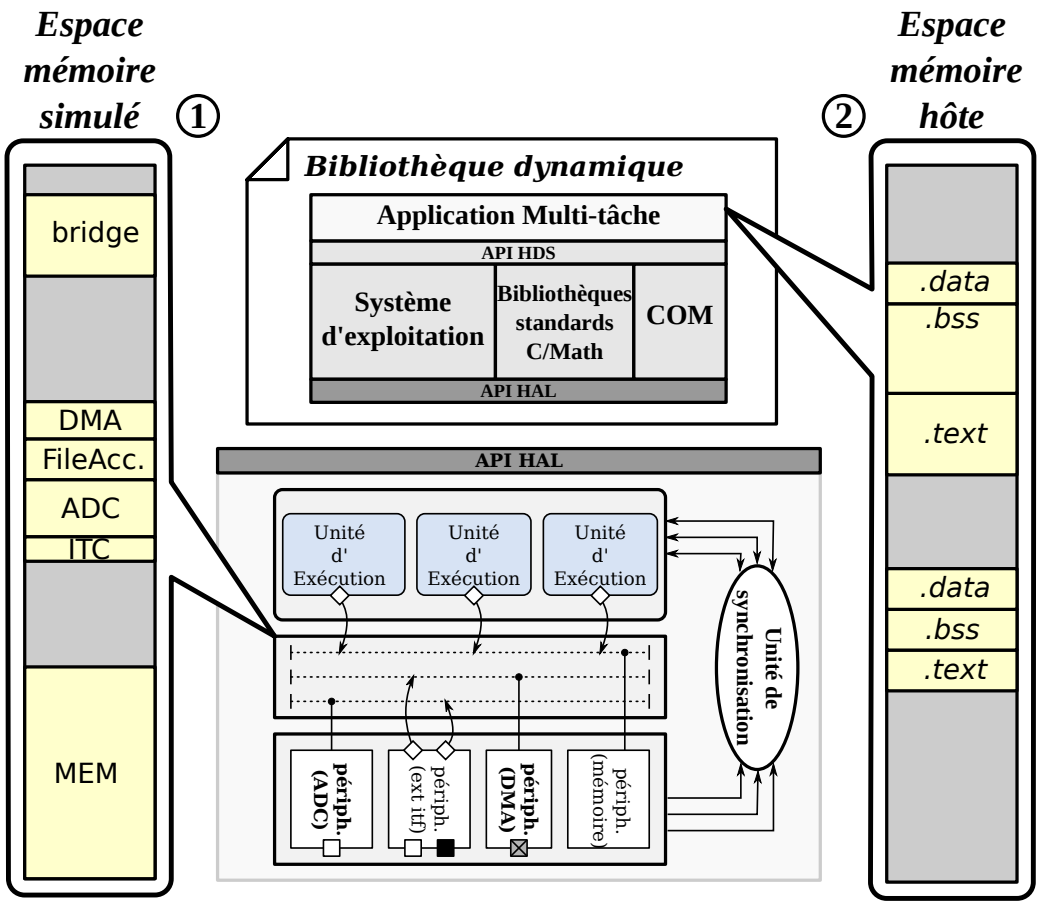
1. Espace mémoire simulé par la plateforme matérielle
   - Défini par les concepteurs
2. Espace mémoire hôte vu par le logiciel
   - Défini à la compilation



Problématiques liée à l'exécution native

Deux espaces mémoire

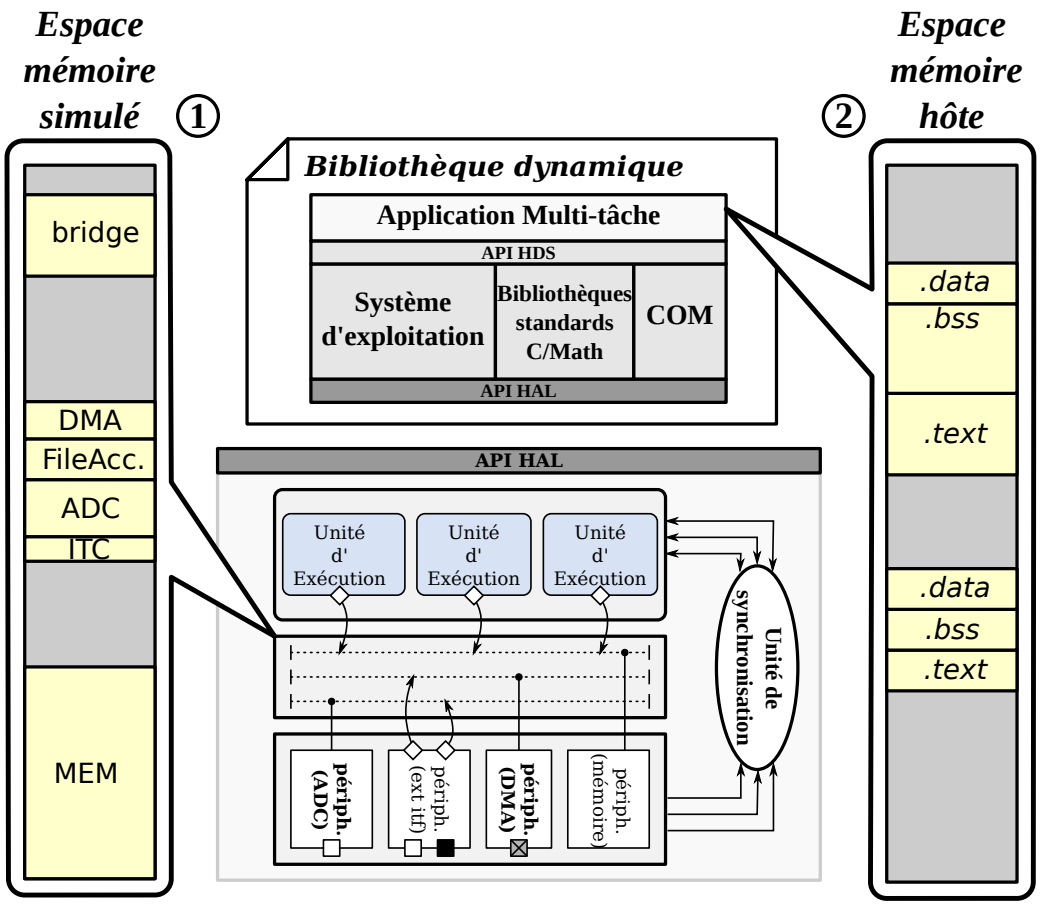
1. Espace mémoire simulé par la plateforme matérielle
   - Défini par les concepteurs
2. Espace mémoire hôte vu par le logiciel
   - Défini à la compilation



Problématiques liée à l'exécution native

Deux espaces mémoire

1. Espace mémoire simulé par la plateforme matérielle
   - Défini par les concepteurs
2. Espace mémoire hôte vu par le logiciel
   - Défini à la compilation



⇒ Difficultés à exécuter du logiciel sans le modifier

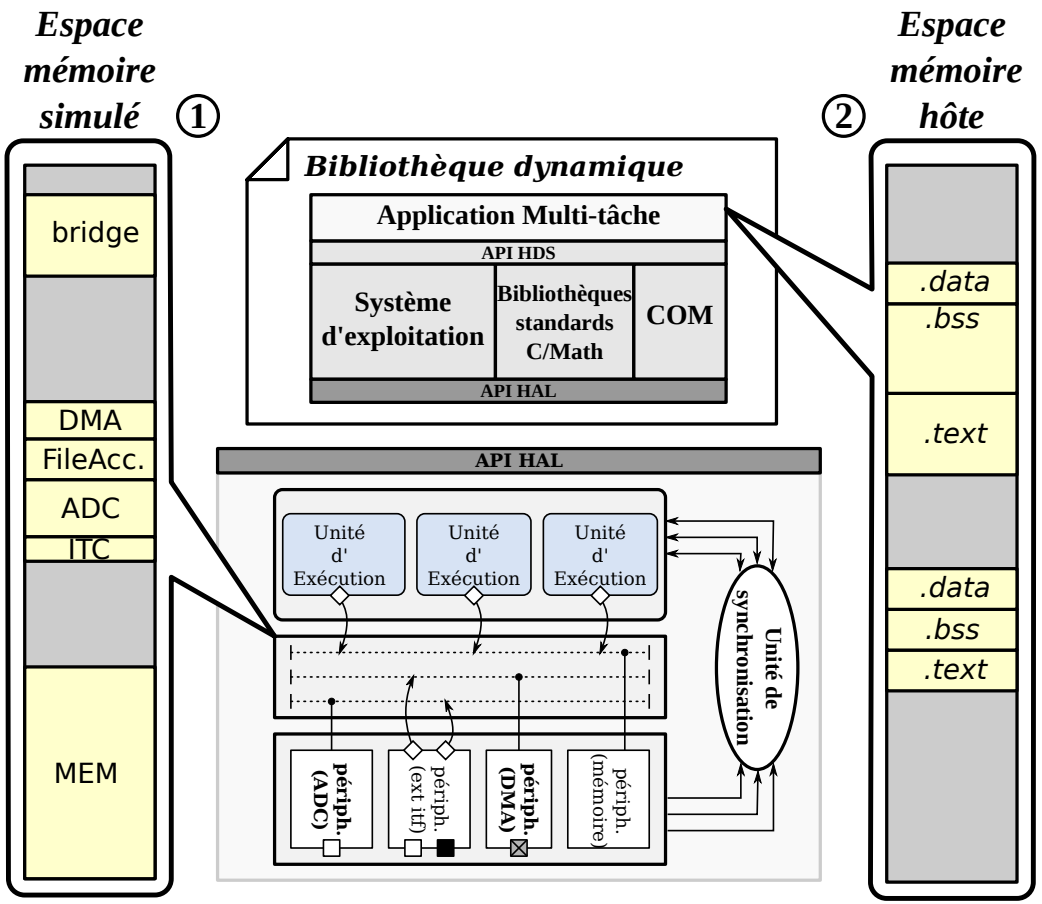
Problématiques liée à l'exécution native

Deux espaces mémoire

1. Espace mémoire simulé par la plateforme matérielle
   - Défini par les concepteurs
2. Espace mémoire hôte vu par le logiciel
   - Défini à la compilation

Exécution du logiciel

- En temps simulé nul
- Synchronisation explicite avec la fonction `wait` en SystemC



⇒ Difficultés à exécuter du logiciel sans le modifier

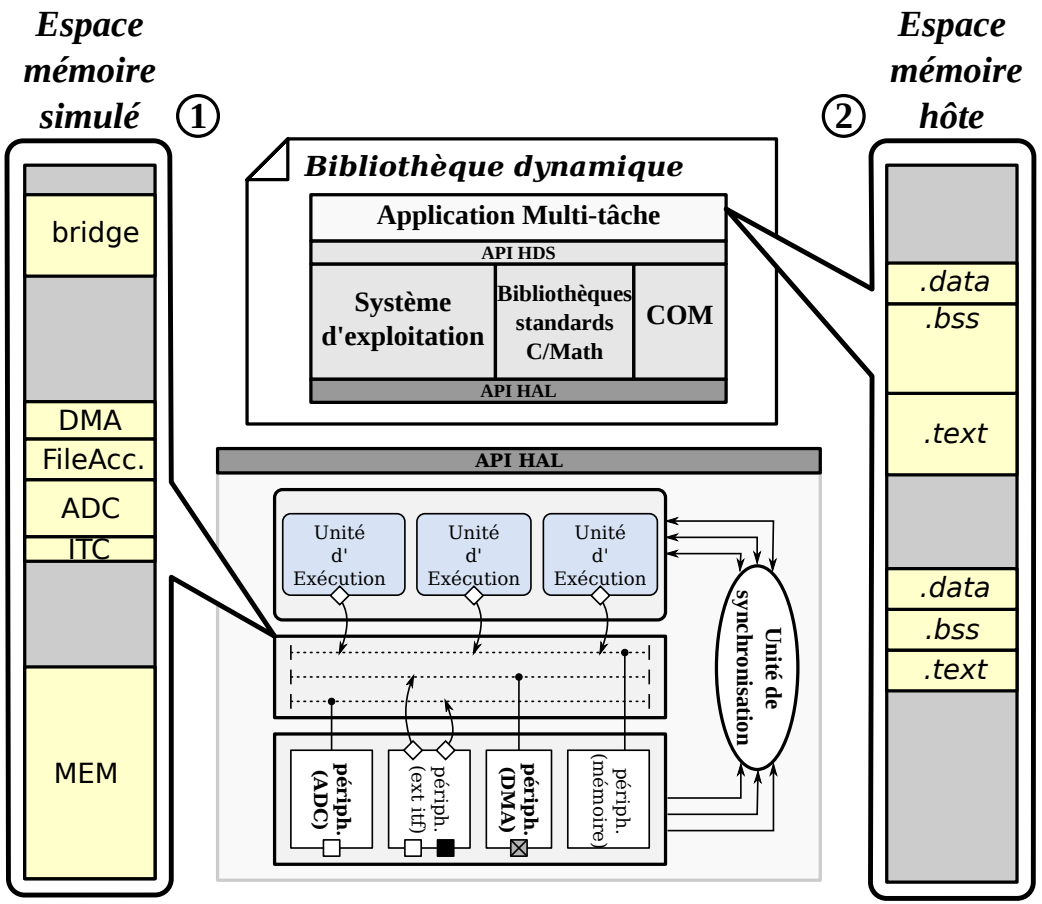
Problématiques liée à l'exécution native

Deux espaces mémoire

1. Espace mémoire simulé par la plateforme matérielle
   - Défini par les concepteurs
2. Espace mémoire hôte vu par le logiciel
   - Défini à la compilation

Exécution du logiciel

- En temps simulé nul
- Synchronisation explicite avec la fonction `wait` en SystemC



⇒ Difficultés à exécuter du logiciel sans le modifier

⇒ Difficultés à exécuter du logiciel dépendant des évènements matériels

Objectifs de ce travail

Exécuter tout le logiciel sans le modifier

- Dans quelle mesure cela est-il possible ?
- Comment maximiser la quantité de code source final réutilisé ?
- Comment prendre en compte les détails architecturaux dans le logiciel ?

Objectifs de ce travail

Exécuter tout le logiciel sans le modifier

- Dans quelle mesure cela est-il possible ?
- Comment maximiser la quantité de code source final réutilisé ?
- Comment prendre en compte les détails architecturaux dans le logiciel ?

Instrumentation du logiciel

- A quel niveau instrumenter le code ?
- Comment refléter l'exécution du logiciel sur le processeur cible ?
- Comment prendre en compte ces informations lors de la simulation ?

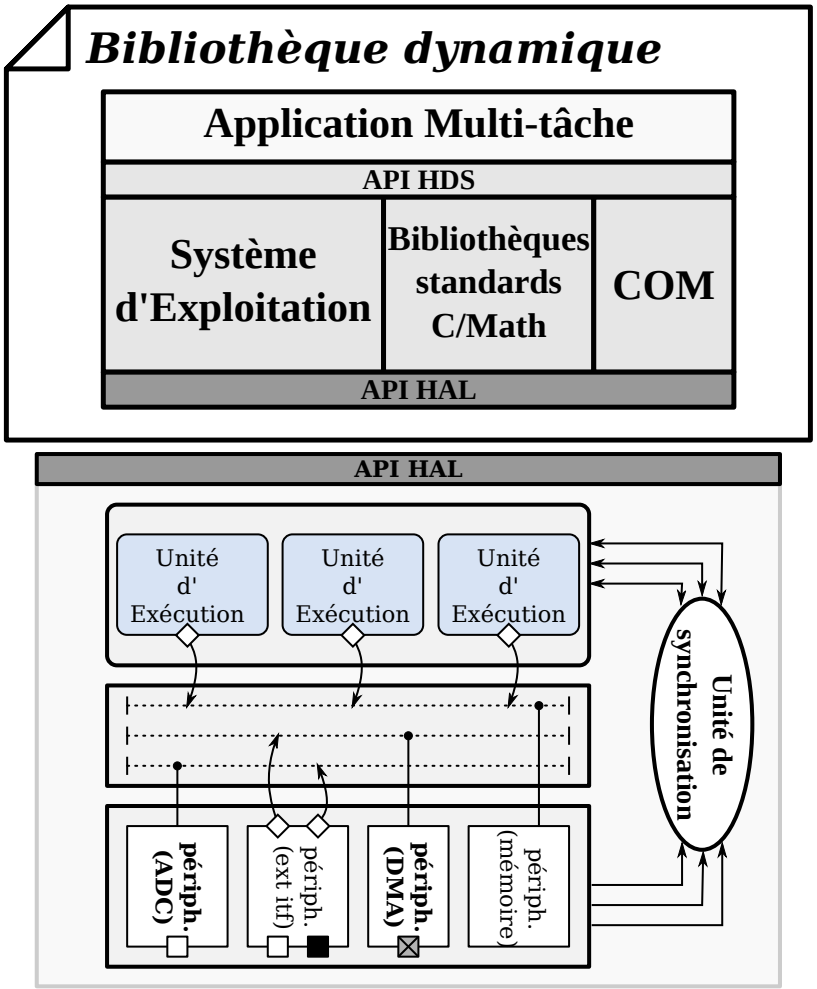
L'Unité d'Exécution

Interface logiciel/matériel unique

- Joue le rôle de processeur
- Unique support d'exécution du logiciel
- Permet d'utiliser des modèles TLM classiques

Contient

- Une partie logicielle
  - ⇒ Réalisation de la couche HAL
- Une partie matérielle
  - ⇒ Interface TLM
  - ⇒ Modélisation du processeur



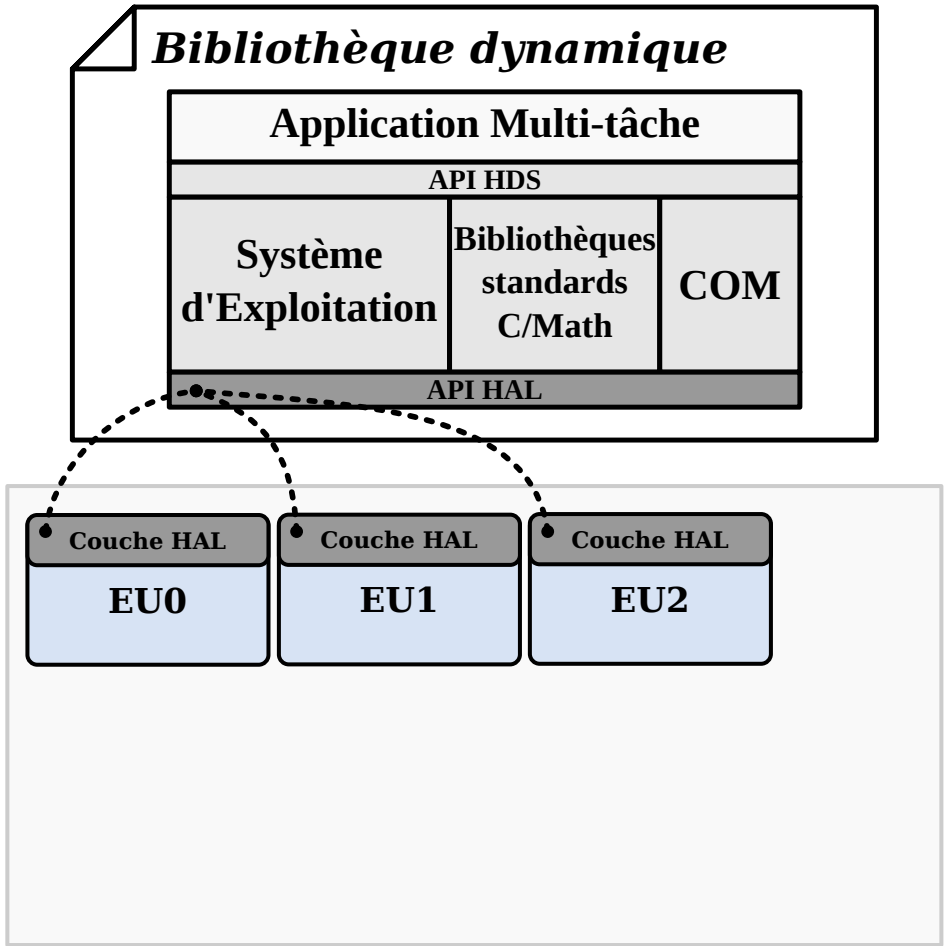
L'Unité d'Exécution

Interface logiciel/matériel unique

- Joue le rôle de processeur
- Unique support d'exécution du logiciel
- Permet d'utiliser des modèles TLM classiques

Contient

- Une partie logicielle
  - ⇒ Réalisation de la couche HAL
- Une partie matérielle
  - ⇒ Interface TLM
  - ⇒ Modélisation du processeur



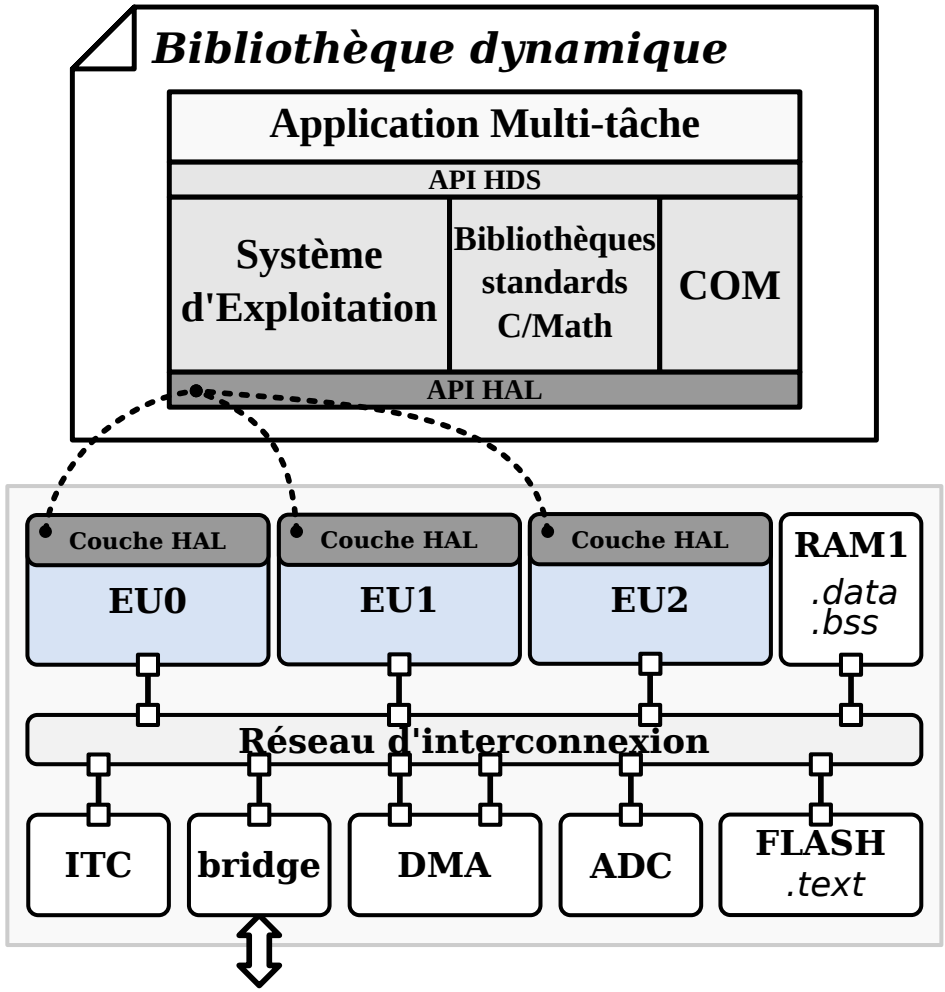
L'Unité d'Exécution

Interface logiciel/matériel unique

- Joue le rôle de processeur
- Unique support d'exécution du logiciel
- Permet d'utiliser des modèles TLM classiques

Contient

- Une partie logicielle
  - ⇒ Réalisation de la couche HAL
- Une partie matérielle
  - ⇒ Interface TLM
  - ⇒ Modélisation du processeur



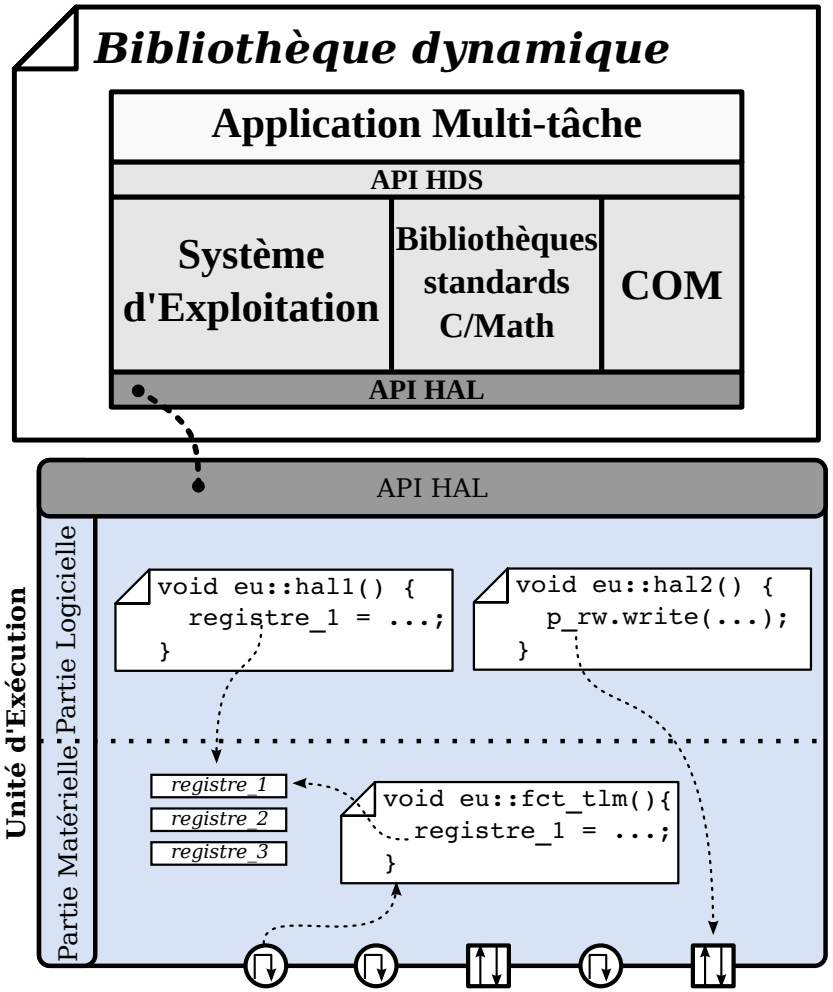
L'Unité d'Exécution

Interface logiciel/matériel unique

- Joue le rôle de processeur
- Unique support d'exécution du logiciel
- Permet d'utiliser des modèles TLM classiques

Contient

- Une partie logicielle
  - ⇒ Réalisation de la couche HAL
- Une partie matérielle
  - ⇒ Interface TLM
  - ⇒ Modélisation du processeur



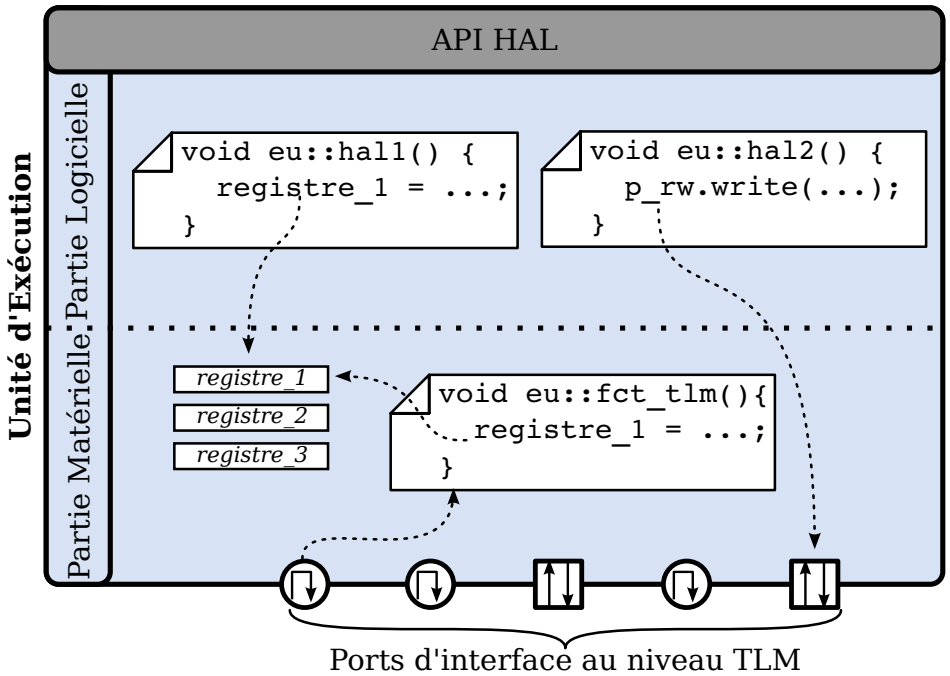
Implémentation de la couche HAL

Pas de règles précises

- Idée: utiliser toutes les ressources disponibles afin d'implémenter le comportement voulu
  - ⇒ Celles de la machine hôte
  - ⇒ Celles modélisées dans la partie matérielle

Exemples

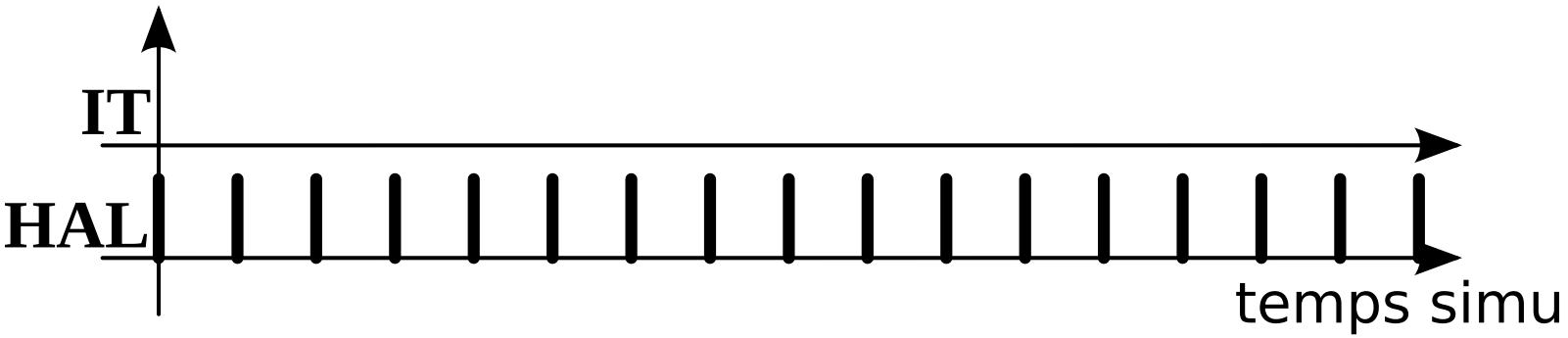
- Utilisation des ressources hôte
  - ⇒ Gestion de contexte
- Utilisation des ressources modélisées
  - ⇒ Identification du processeur
  - ⇒ Lecture/écriture en mémoire



L'API HAL comme point de synchronisation

Toujours consommer du temps

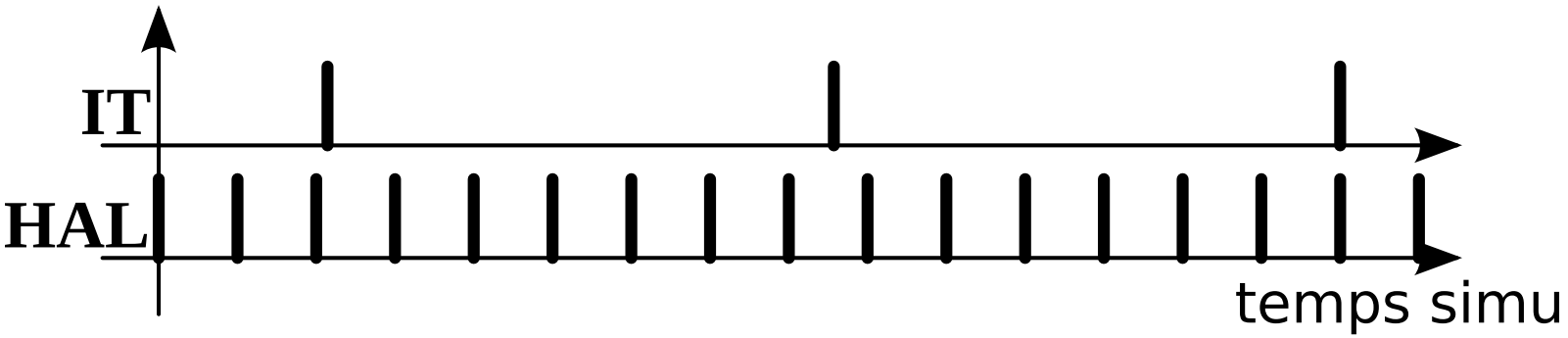
- Indispensable pour faire évoluer la simulation SystemC
- $\Rightarrow$ Temps de synchronisation $Tps_{synchro}$, consommé à chaque appel au HAL
- Quelle valeur choisir:
  - Simulation fonctionnelle $\Rightarrow$ Peu importe!
  - Simulation des évènements matériels $\Rightarrow$???



L'API HAL comme point de synchronisation

Toujours consommer du temps

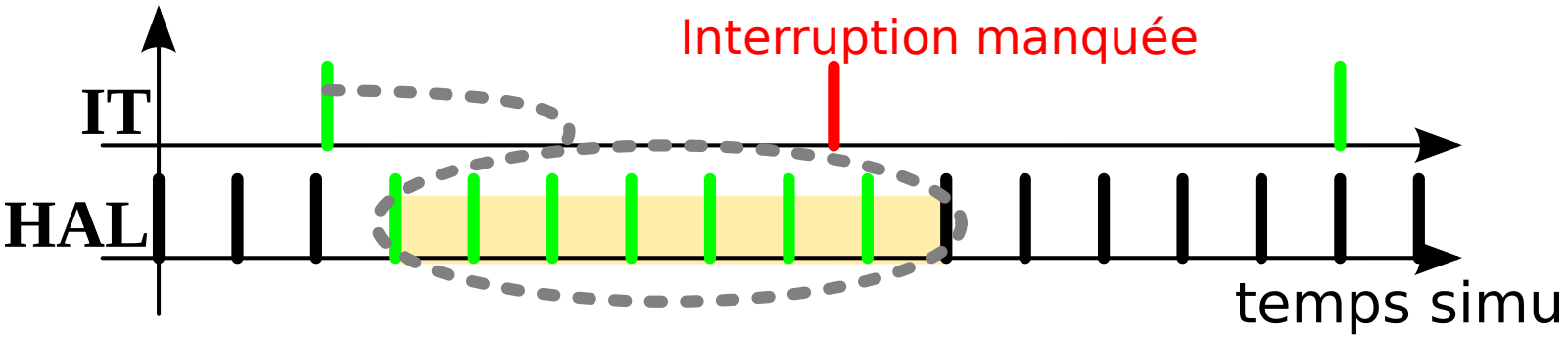
- Indispensable pour faire évoluer la simulation SystemC
- $\Rightarrow$ Temps de synchronisation $Tps_{synchro}$, consommé à chaque appel au HAL
- Quelle valeur choisir:
  - Simulation fonctionnelle $\Rightarrow$ Peu importe!
  - Simulation des évènements matériels $\Rightarrow$???



L'API HAL comme point de synchronisation

Toujours consommer du temps

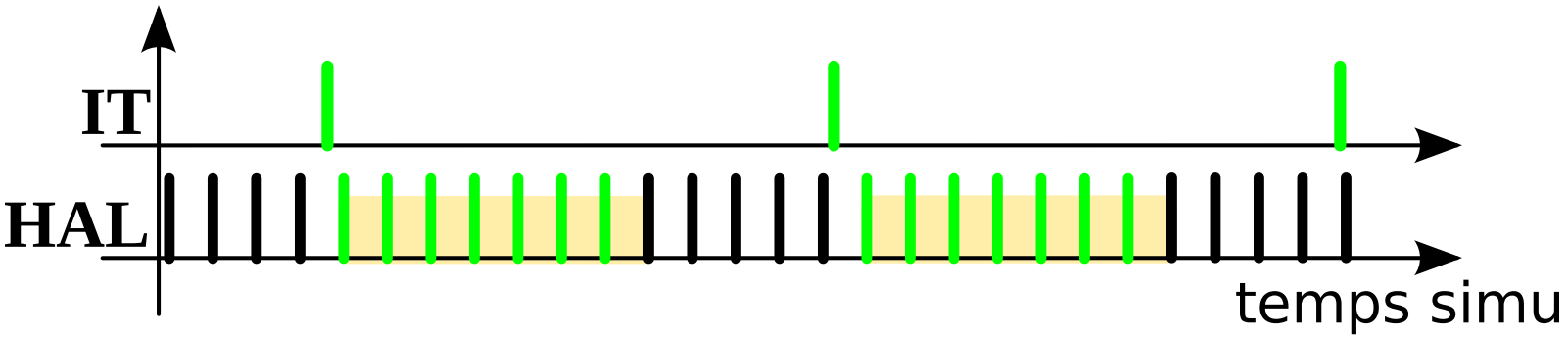
- Indispensable pour faire évoluer la simulation SystemC
- $\Rightarrow$ Temps de synchronisation $Tps_{synchro}$, consommé à chaque appel au HAL
- Quelle valeur choisir:
  - Simulation fonctionnelle $\Rightarrow$ Peu importe!
  - Simulation des évènements matériels $\Rightarrow$???



L'API HAL comme point de synchronisation

Toujours consommer du temps

- Indispensable pour faire évoluer la simulation SystemC
- $\Rightarrow$ Temps de synchronisation $Tps_{synchro}$, consommé à chaque appel au HAL
- Quelle valeur choisir:
  - Simulation fonctionnelle $\Rightarrow$ Peu importe!
  - Simulation des évènements matériels $\Rightarrow$???



L'API HAL comme point de synchronisation

Problème similaire à une condition de charge

Respecter l'équation suivante : $\sum_{i \in I} \frac{C_i}{T_i} \leq 1$

Transposée à l'exécution native:

$$\sum_{i \in I} \frac{nb\_appels\_hal_i \times Tps_{synchro_i}}{T_i} \leq 1 \boxed{\Rightarrow 0 < Tps_{synchro} \leq \frac{1}{\sum_{i \in I} \frac{nb\_appels\_hal_i}{T_i}}}$$

Choix de $Tps_{synchro}$ en pratique

- Éviter les valeurs trop petites
  - $\Rightarrow$ Diminution des performances de simulation
- Le nombre d'appels au HAL peut varier
  - $\Rightarrow$ Déterminer la valeur max manuellement ou par profilage du logiciel

$\Rightarrow$ Ce n'est pas une estimation du temps d'exécution

$\Rightarrow$ Uniquement le moyen d'exécuter nativement le logiciel de manière correcte

Implémentation de la partie matérielle

Modélisation de l'architecture interne du processeur

- Pas de règles précises
- Fonction des besoins ... et des capacités à modéliser le comportement attendu

Aspects de l'architecture à modéliser pour le logiciel

- Implicites: Pipelines, caches
  $\Rightarrow$ Ne peuvent être pris en compte que par l'instrumentation du logiciel.
- Explicites: ceux directement visibles par le logiciel
  Gestion des interruptions, Registres spécifiques, MMU ...
  $\Rightarrow$ Doivent impérativement être modélisés pour permettre l'exécution du logiciel sans modifications

Toute l'API du HAL doit être fournie par les EU, même si le comportement de certaines fonctions n'est pas modélisé

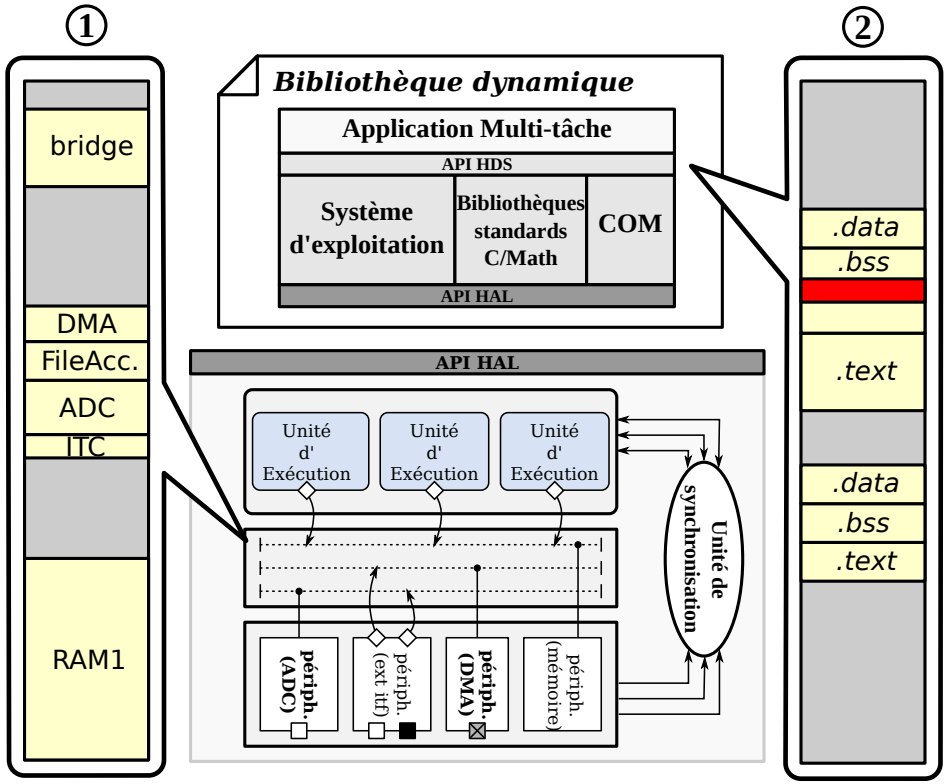
$\Rightarrow$ API spécifique aux mémoires caches par exemple

Incompatibilité des espaces mémoire

⇒ Impossible de modéliser des interactions entre le matériel et le logiciel

Exemple d'un transfert DMA

- Configuration du DMA
  - Lecture dans un périphérique
    Adresse cible 0xa0001004
  - Écriture en mémoire
    Adresse hôte 0xbf1ace00



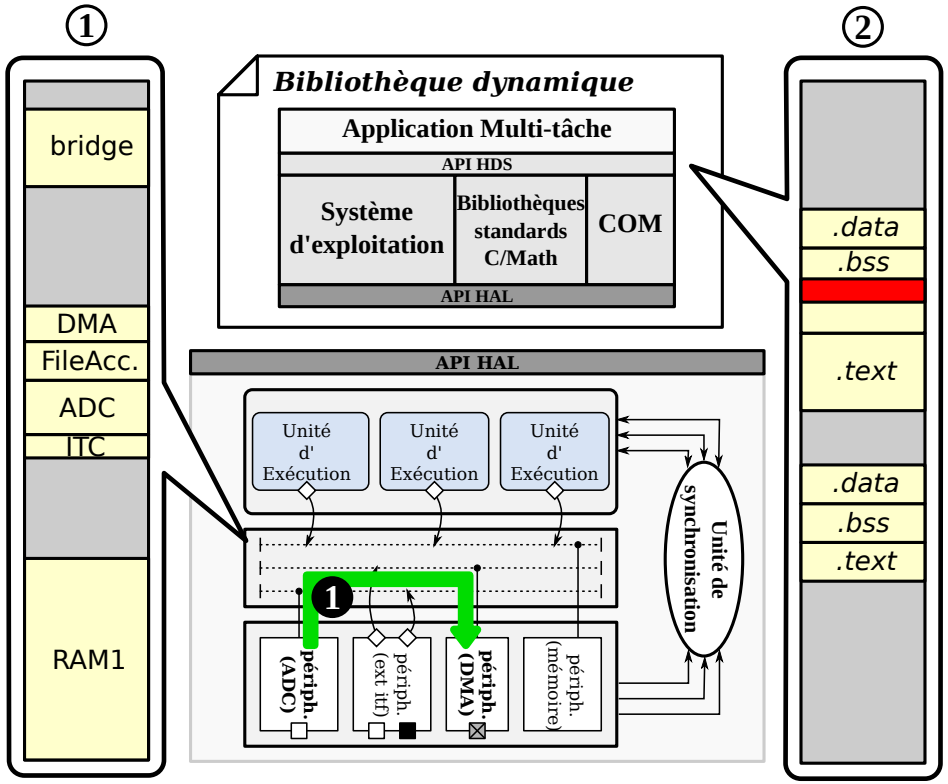
Incompatibilité des espaces mémoire

⇒ Impossible de modéliser des interactions entre le matériel et le logiciel

Exemple d'un transfert DMA

- Configuration du DMA
  - Lecture dans un périphérique
    Adresse cible 0xa0001004
  - Écriture en mémoire
    Adresse hôte 0xbf1ace00

1. Lecture des données
   - Accès DMA à l'adresse 0xa0001004
   - ⇒ Accès valide



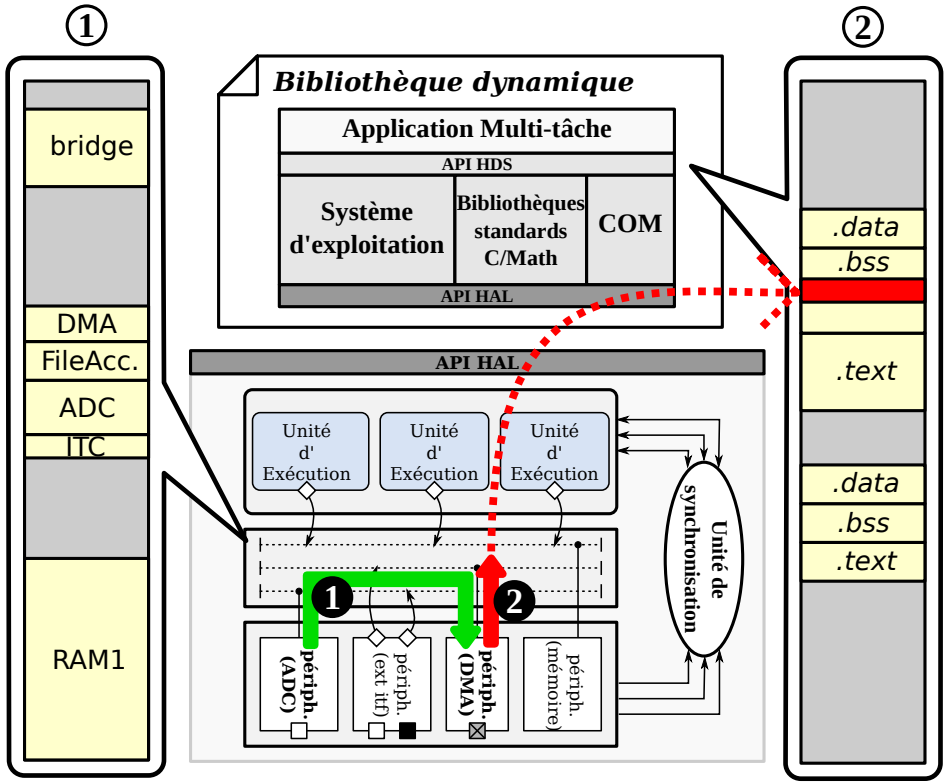
Incompatibilité des espaces mémoire

⇒ Impossible de modéliser des interactions entre le matériel et le logiciel

Exemple d'un transfert DMA

- Configuration du DMA
  - Lecture dans un périphérique
    Adresse cible 0xa0001004
  - Écriture en mémoire
    Adresse hôte 0xbf1ace00

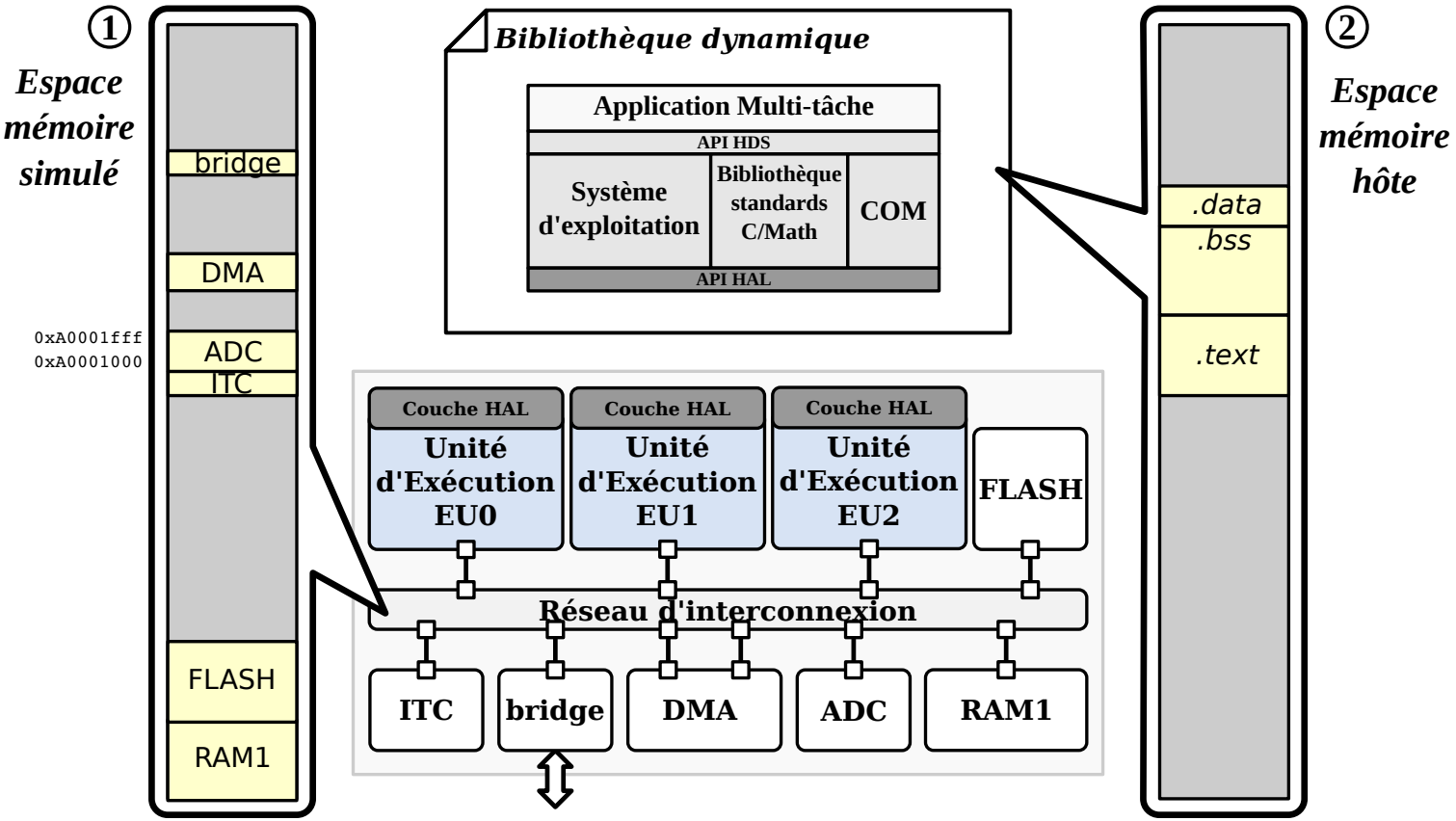
1. Lecture des données
   - Accès DMA à l'adresse 0xa0001004
   - ⇒ Accès valide
2. Écriture des données
   - Accès DMA à l'adresse 0xbf1ace00
   - ⇒ Accès non valide



Représentation uniforme de la mémoire

Utilisation du plan mémoire de l'exécutable SystemC

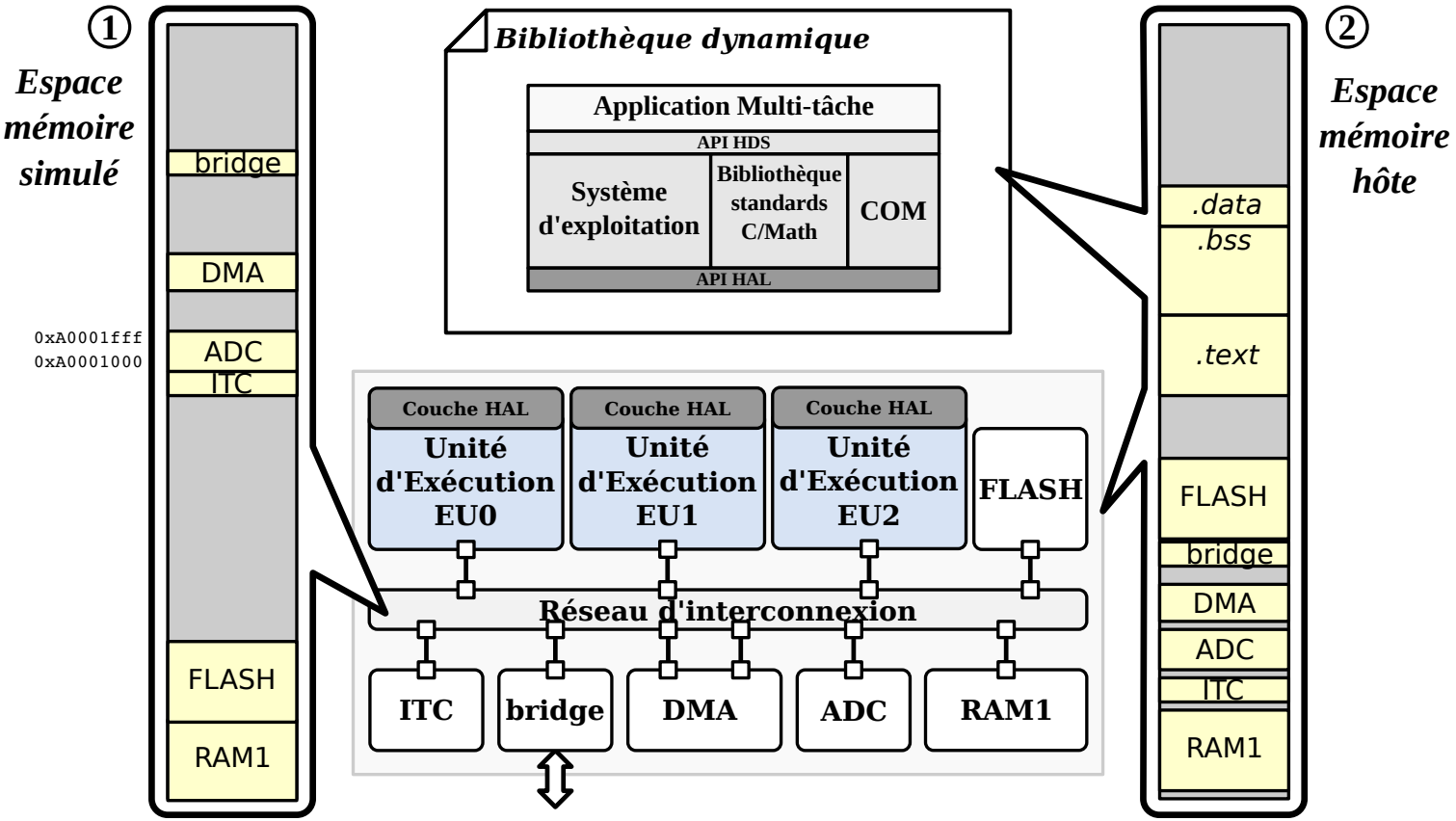
- Si tous les composants modélisent leurs registres et mémoires internes
  $\Rightarrow$ Pour chacune des adresses dans ① il existe des adresses équivalentes dans ②



Représentation uniforme de la mémoire

Utilisation du plan mémoire de l'exécutable SystemC

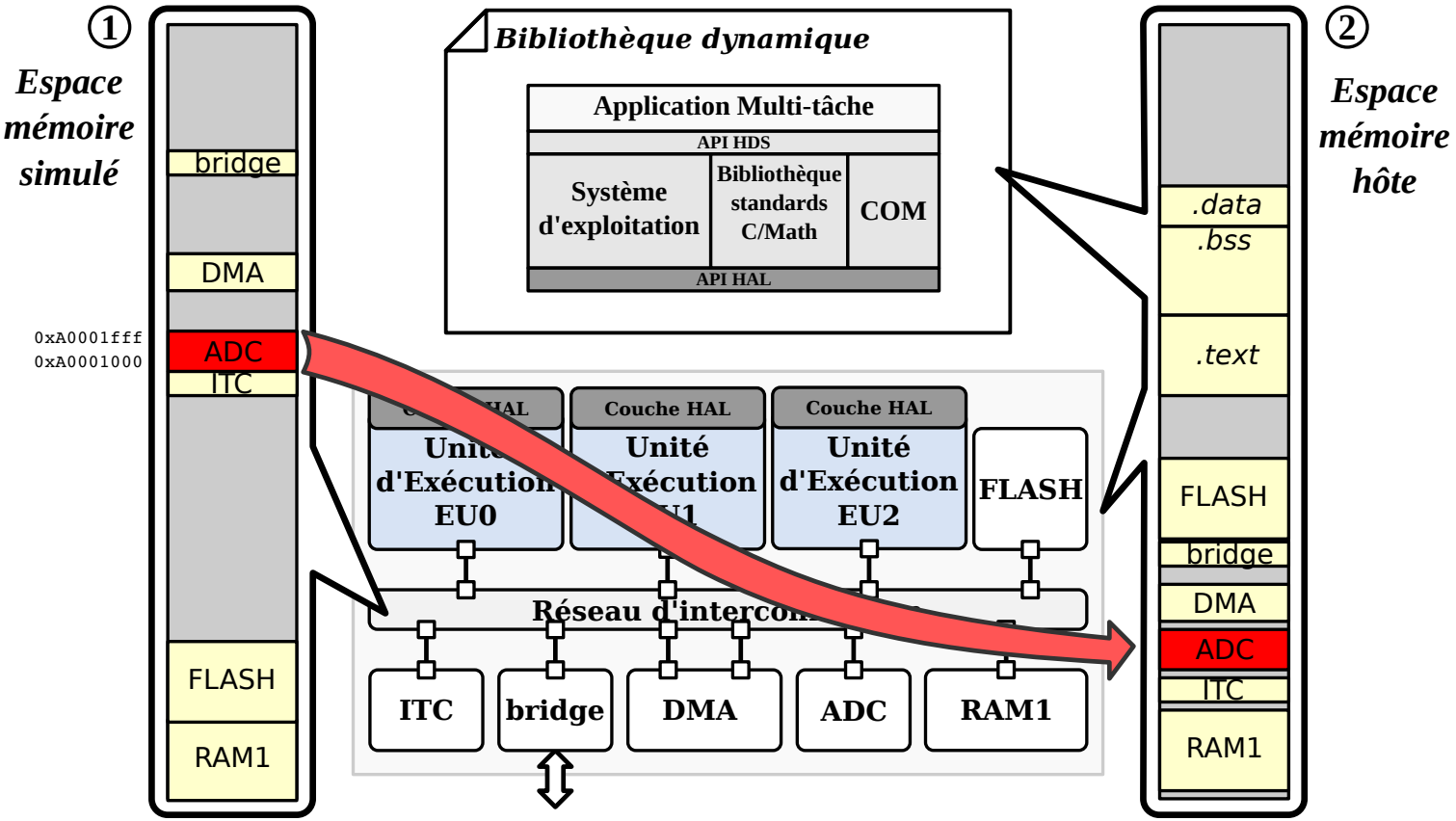
- Si tous les composants modélisent leurs registres et mémoires internes
  $\Rightarrow$ Pour chacune des adresses dans ① il existe des adresses équivalentes dans ②



Représentation uniforme de la mémoire

Utilisation du plan mémoire de l'exécutable SystemC

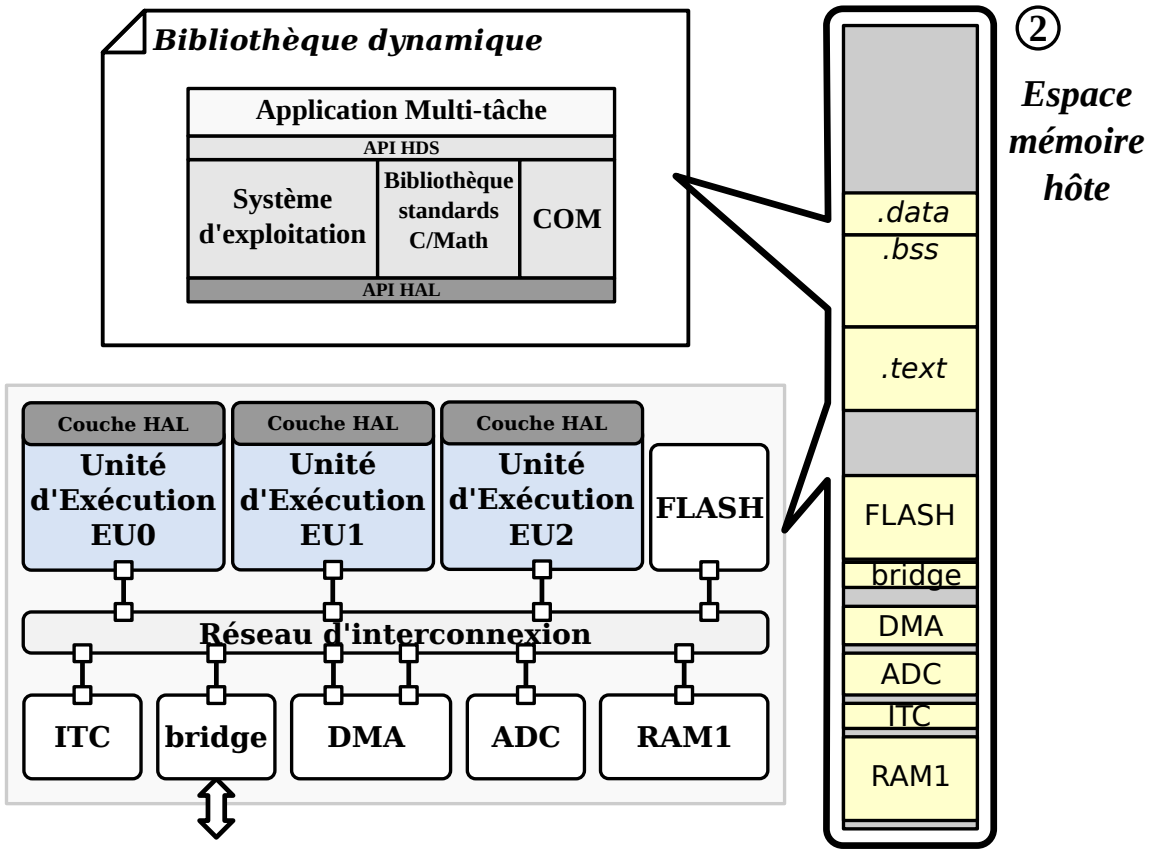
- Si tous les composants modélisent leurs registres et mémoires internes
  $\Rightarrow$ Pour chacune des adresses dans ① il existe des adresses équivalentes dans ②



Représentation uniforme de la mémoire

Construction dynamique du plan mémoire

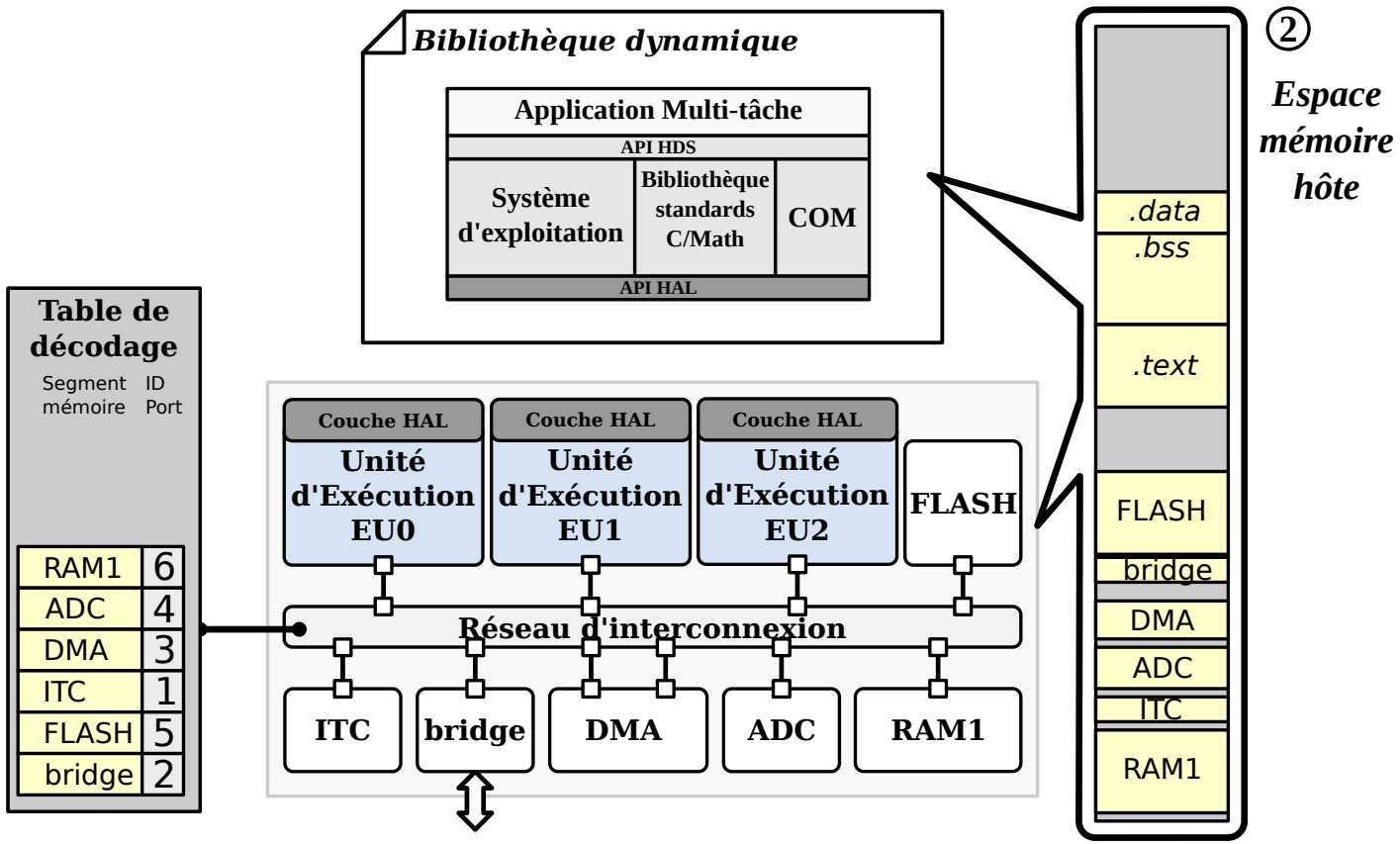
- Chaque composant fournit la description de son espace mémoire au réseau
- Le plan mémoire est construit à l'initialisation de la plateforme



Représentation uniforme de la mémoire

Construction dynamique du plan mémoire

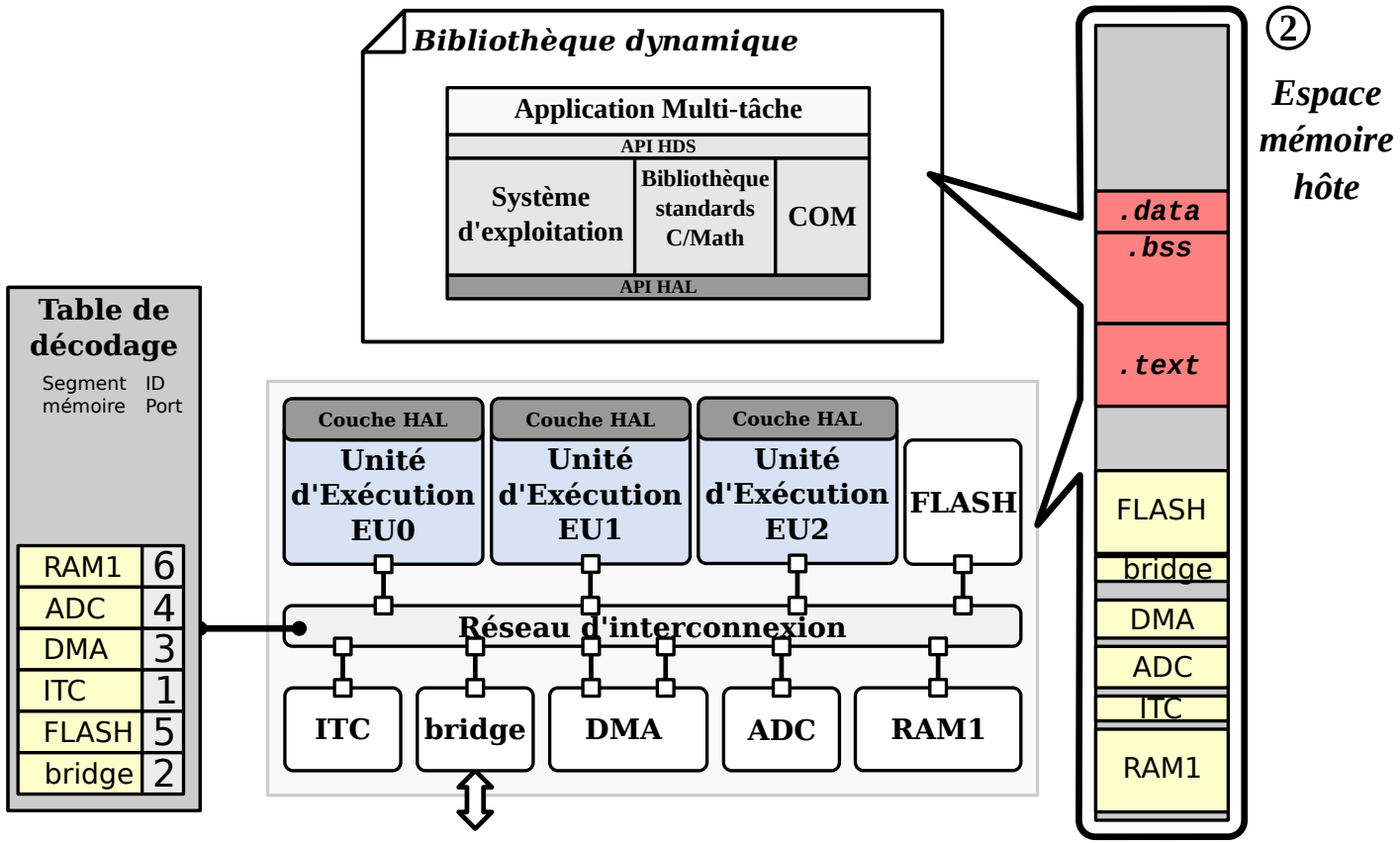
- Chaque composant fournit la description de son espace mémoire au réseau
- Le plan mémoire est construit à l'initialisation de la plateforme



Représentation uniforme de la mémoire

Les zones mémoire logicielles

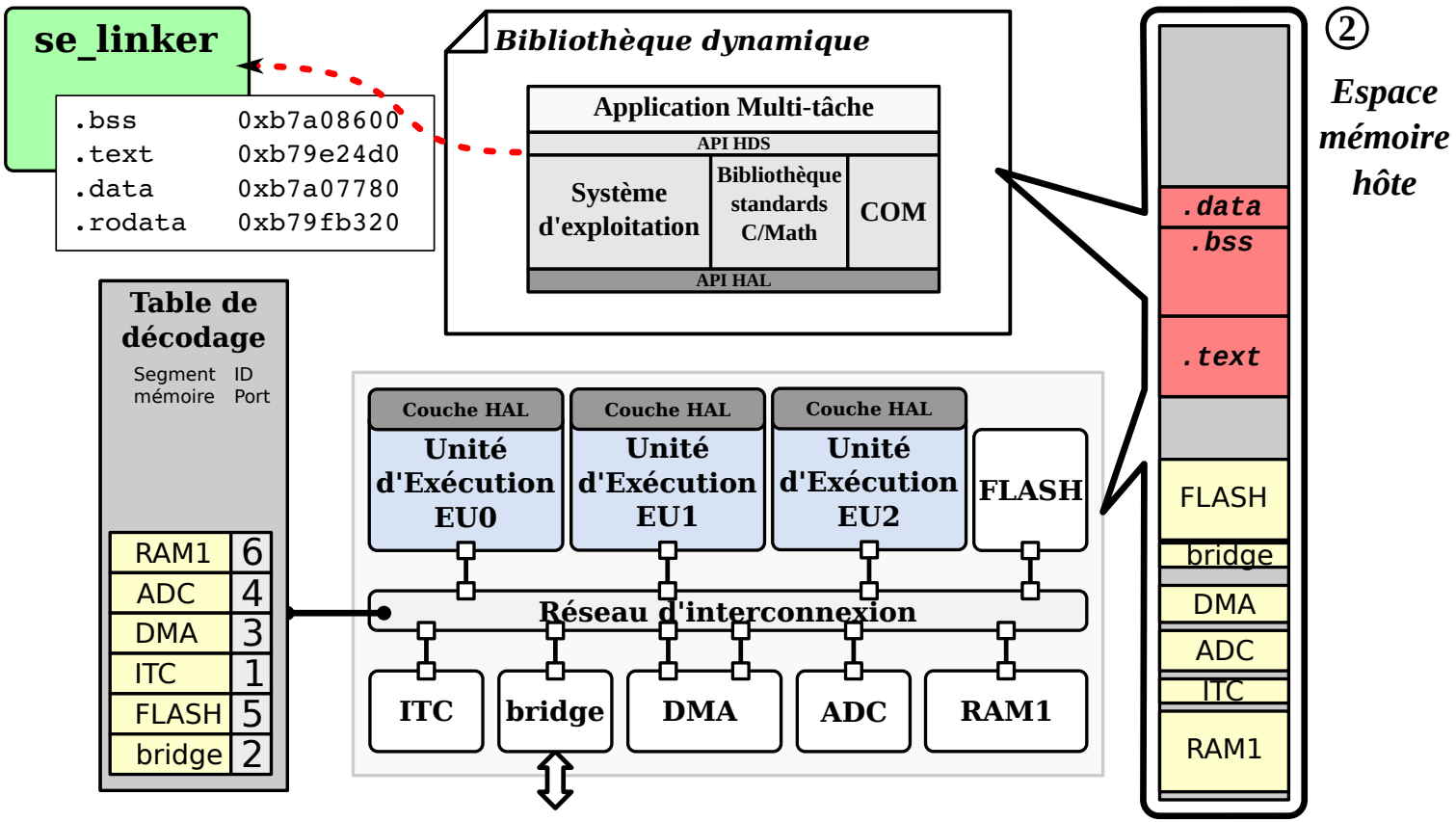
- Obtenues lors de la compilation du logiciel, `.text`, `.bss`, `.data` ..
- Doivent être accessibles par le matériel (exemple DMA)



Représentation uniforme de la mémoire

Les zones mémoire logicielles

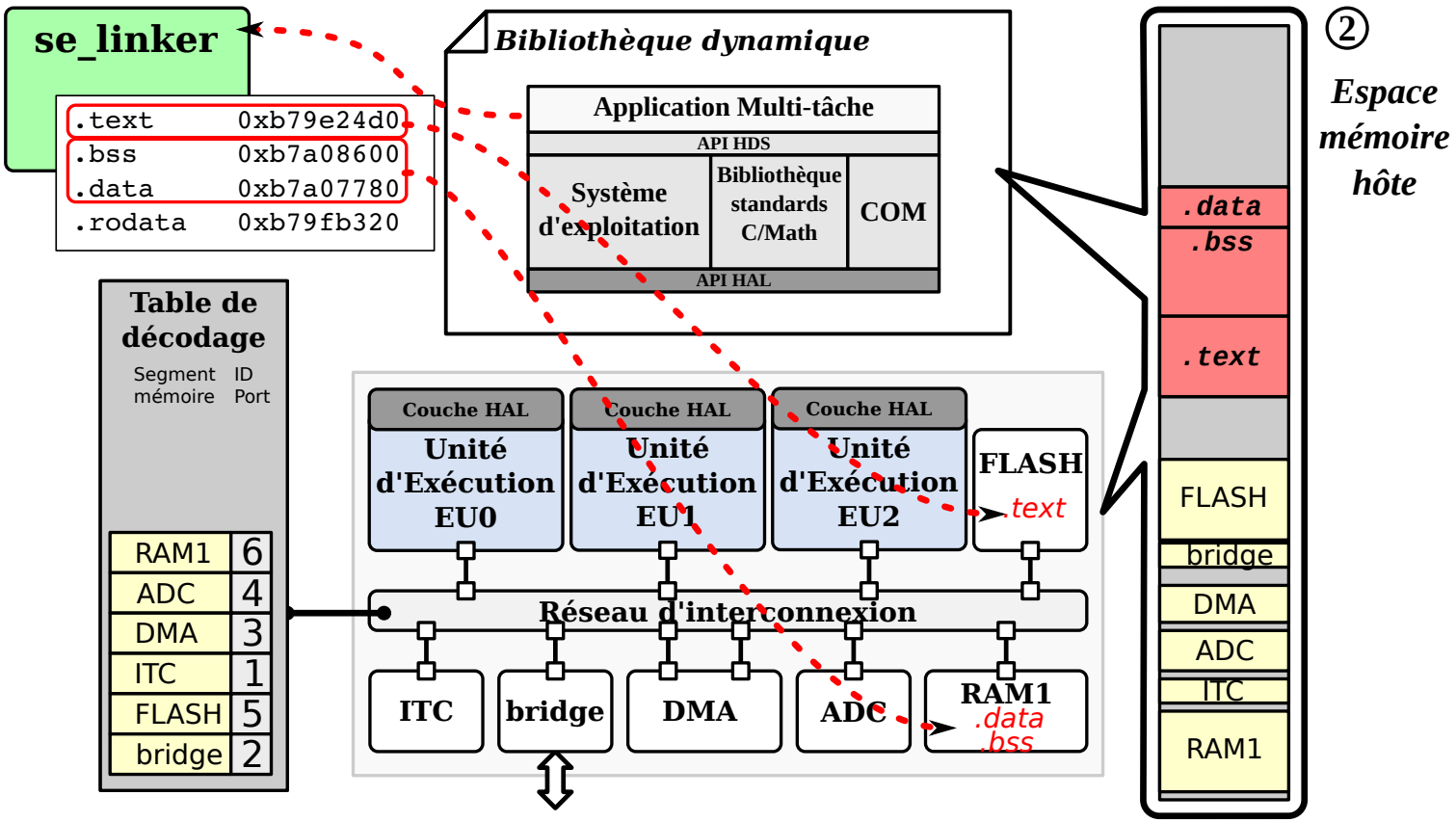
- Obtenues lors de la compilation du logiciel, `.text`, `.bss`, `.data` ..
- Doivent être accessibles par le matériel (exemple DMA)



Représentation uniforme de la mémoire

Les zones mémoire logicielles

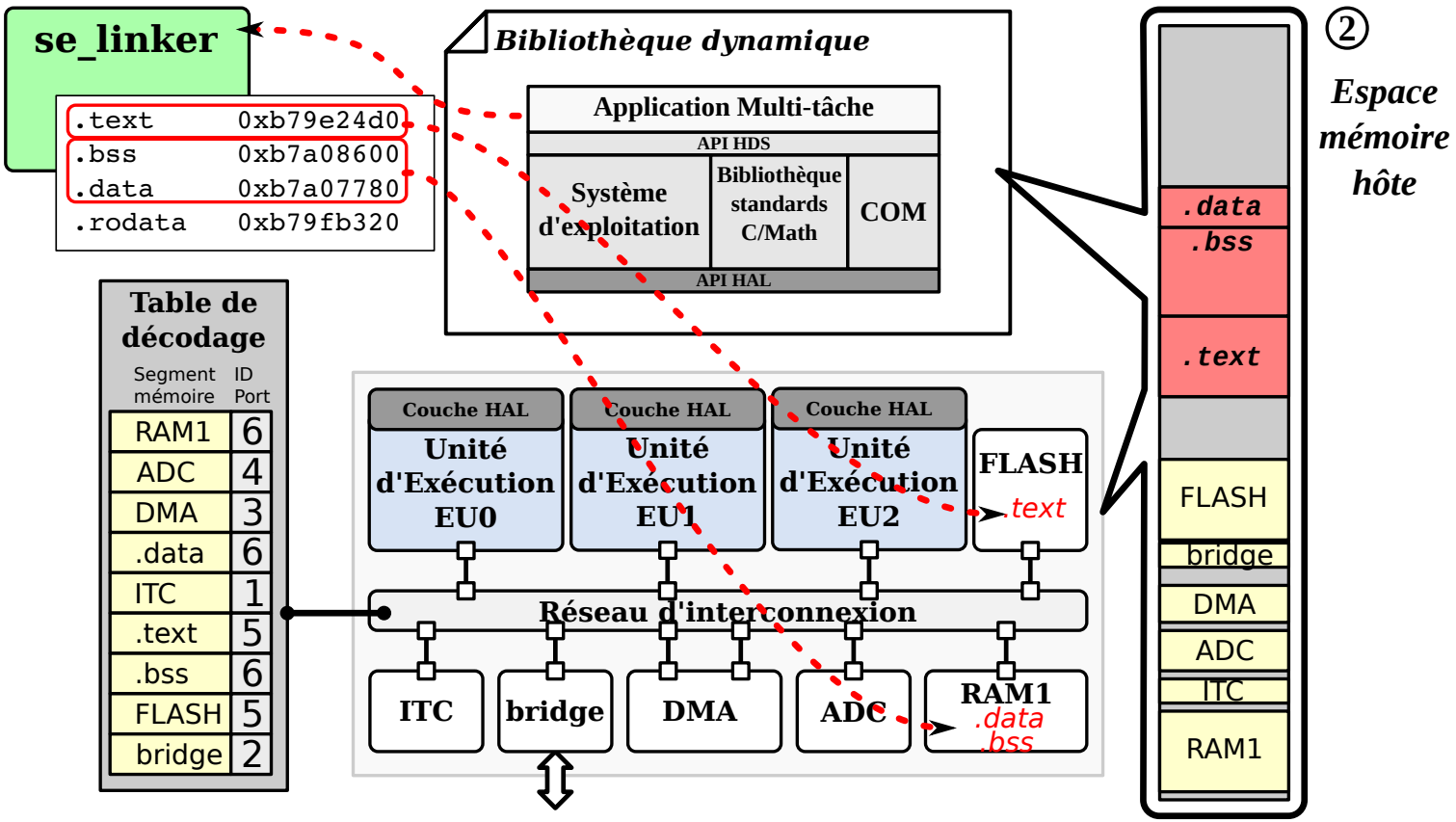
- Obtenues lors de la compilation du logiciel, `.text`, `.bss`, `.data` ..
- Doivent être accessibles par le matériel (exemple DMA)



Représentation uniforme de la mémoire

Les zones mémoire logicielles

- Obtenues lors de la compilation du logiciel, `.text`, `.bss`, `.data` ..
- Doivent être accessibles par le matériel (exemple DMA)



Édition dynamique de liens

Logiciel dépendant d'adresses physiques

- Pilotes de périphériques
  - ⇒ Adresses de bases des registres
- Système d'exploitation
  - ⇒ Adresses des zones mémoires (`.bss`, pile noyau, ...)

Solution proposée

- Utilisation de pointeurs intermédiaires

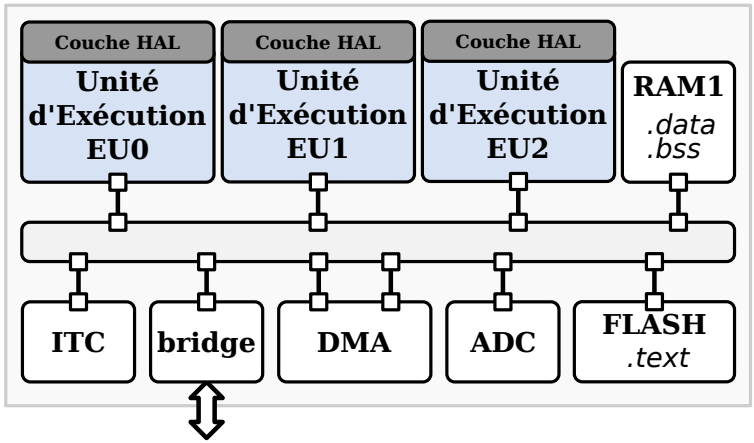
✘ ~~`#define adr_base ((volatile uint32_t *)0xA0001000)`~~

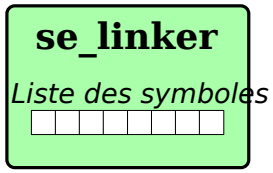
✔
```
volatile uint32_t *adr_base = (volatile uint32_t *)0xA0001000;
```

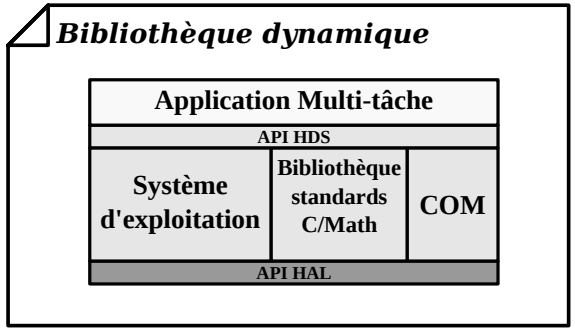
- Leurs valeurs pourront être modifiées à l'initialisation de la plateforme
- Également valable sur la plateforme finale sans modifier le code source

Édition dynamique de liens

Principe: Résolution des adresses à l'initialisation par le linker



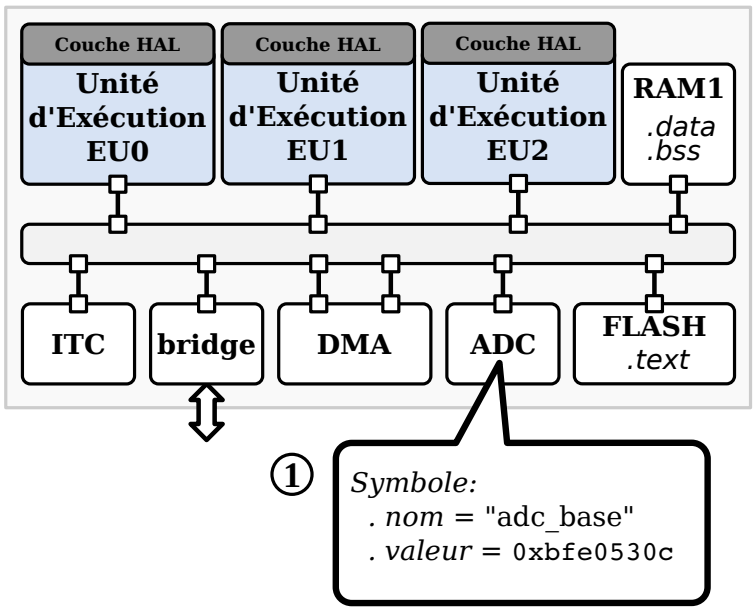


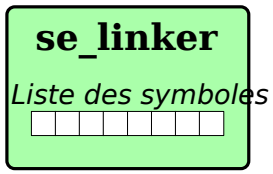


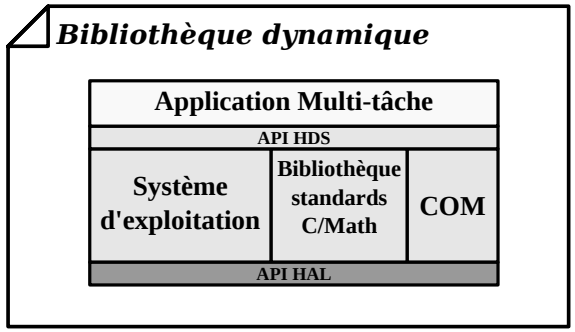
Édition dynamique de liens

Principe: Résolution des adresses à l'initialisation par le linker

1. Chaque composant définit les symboles (couple nom, valeur) attendus par le pilote logiciel correspondant



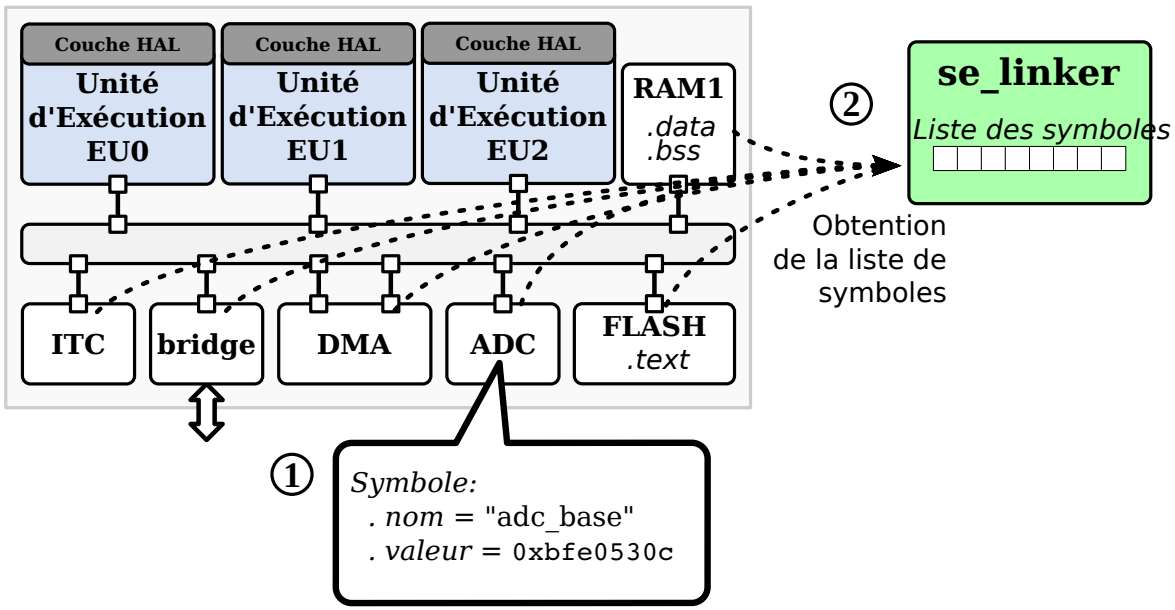


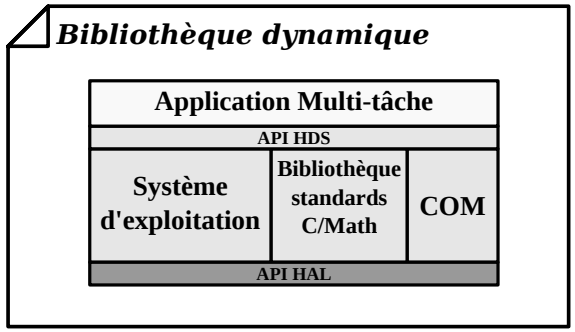


Édition dynamique de liens

Principe: Résolution des adresses à l'initialisation par le linker

1. Chaque composant définit les symboles (couple nom, valeur) attendus par le pilote logiciel correspondant
2. Le linker construit la liste des symboles de la plateforme

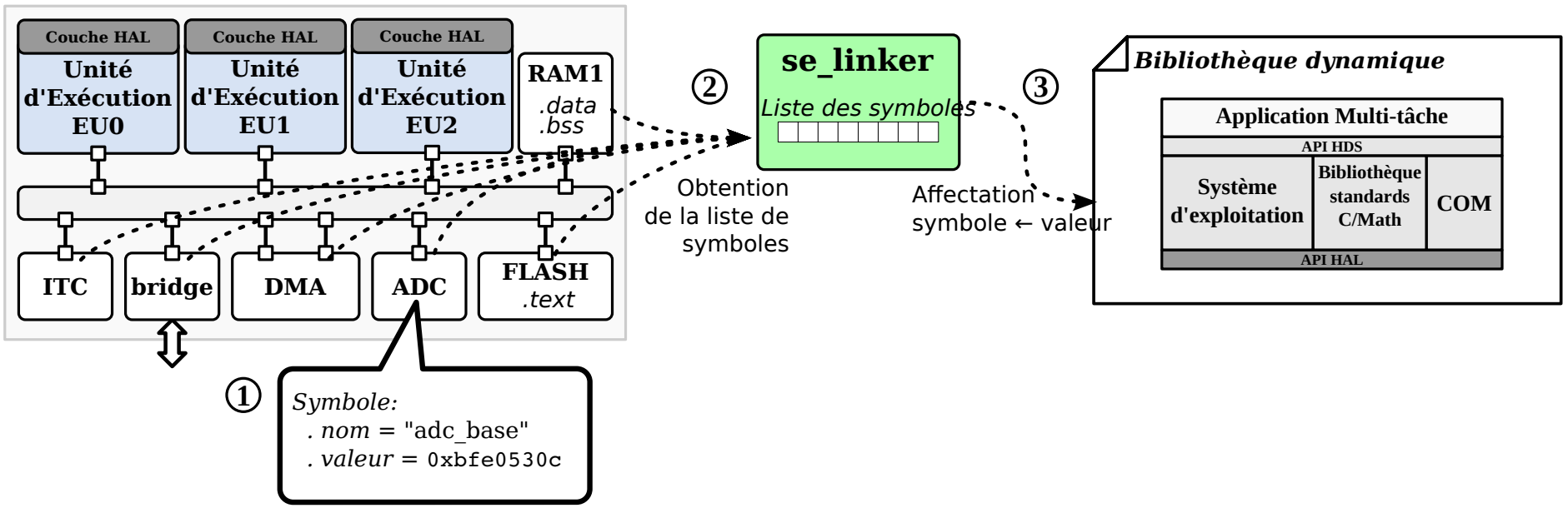




Édition dynamique de liens

Principe: Résolution des adresses à l'initialisation par le linker

1. Chaque composant définit les symboles (couple nom, valeur) attendus par le pilote logiciel correspondant
2. Le linker construit la liste des symboles de la plateforme
3. Les valeurs sont affectées directement aux symboles correspondants dans le code logiciel



Instrumentation automatique du logiciel embarqué

```
1 x = (y != 0) ? 23 : 1234567;
```

Concepts et problématiques

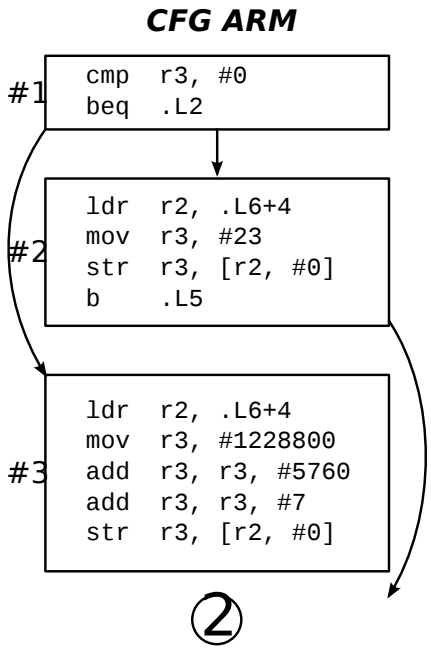
1. Code source à instrumenter

Instrumentation automatique du logiciel embarqué

① `x = (y != 0) ? 23 : 1234567;`

Concepts et problématiques

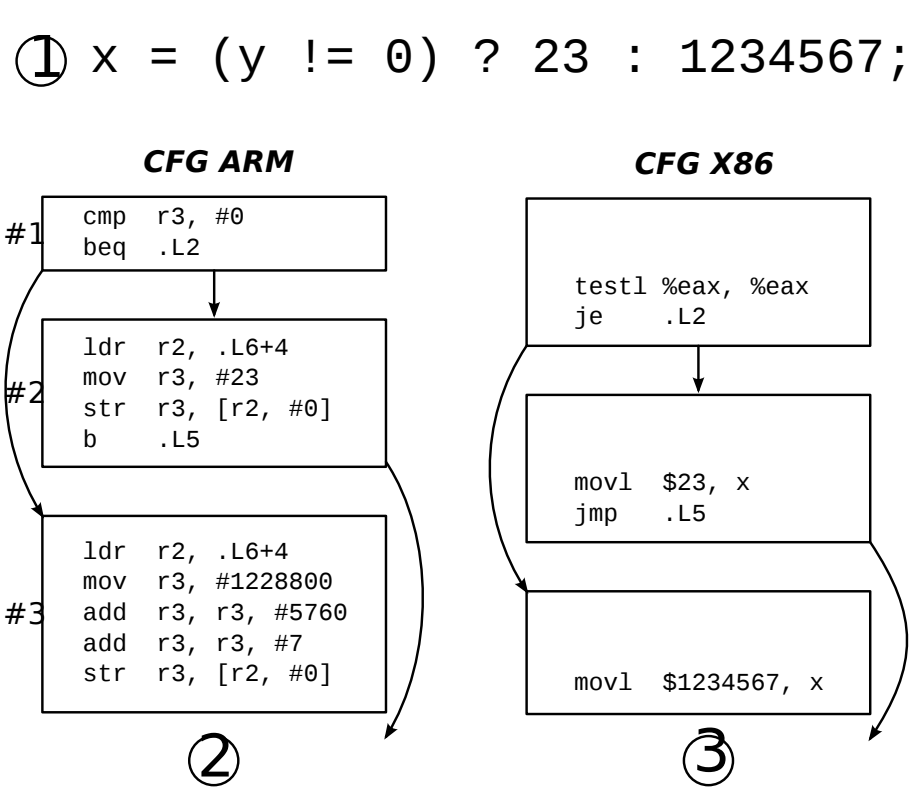
1. Code source à instrumenter
2. Code compilé pour le processeur cible (ARM)



Instrumentation automatique du logiciel embarqué

Concepts et problématiques

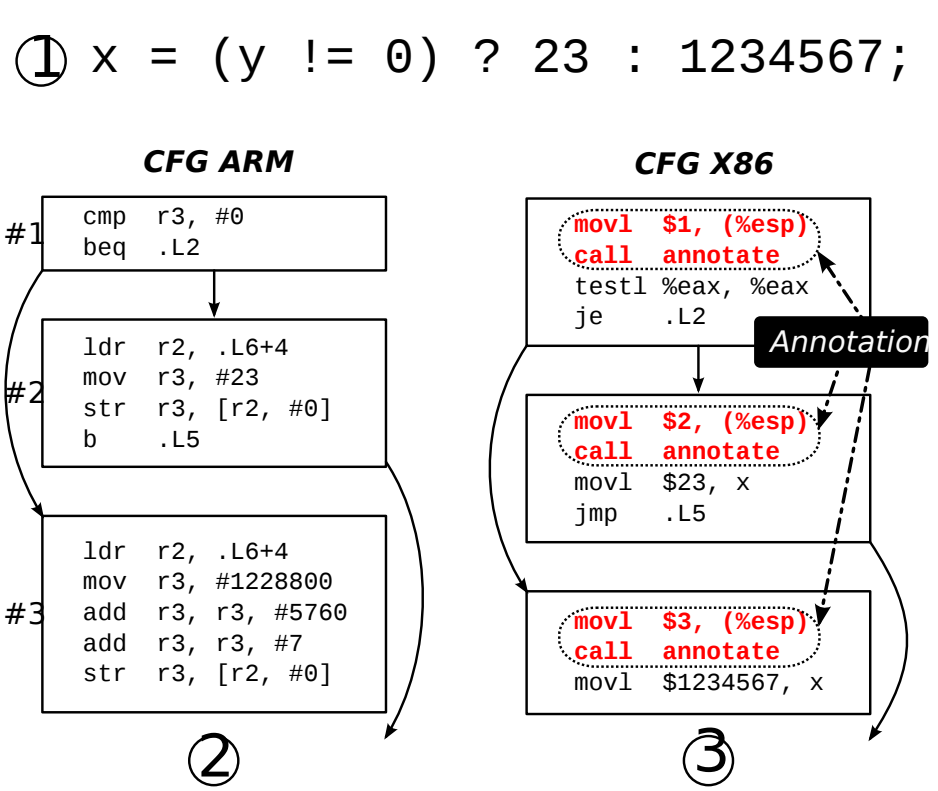
1. Code source à instrumenter
2. Code compilé pour le processeur cible (ARM)
3. Code compilé pour le processeur hôte (x86)
   - Annotations insérées en début de chaque bloc de base
   - L'argument est un identifiant du bloc cible correspondant



Instrumentation automatique du logiciel embarqué

Concepts et problématiques

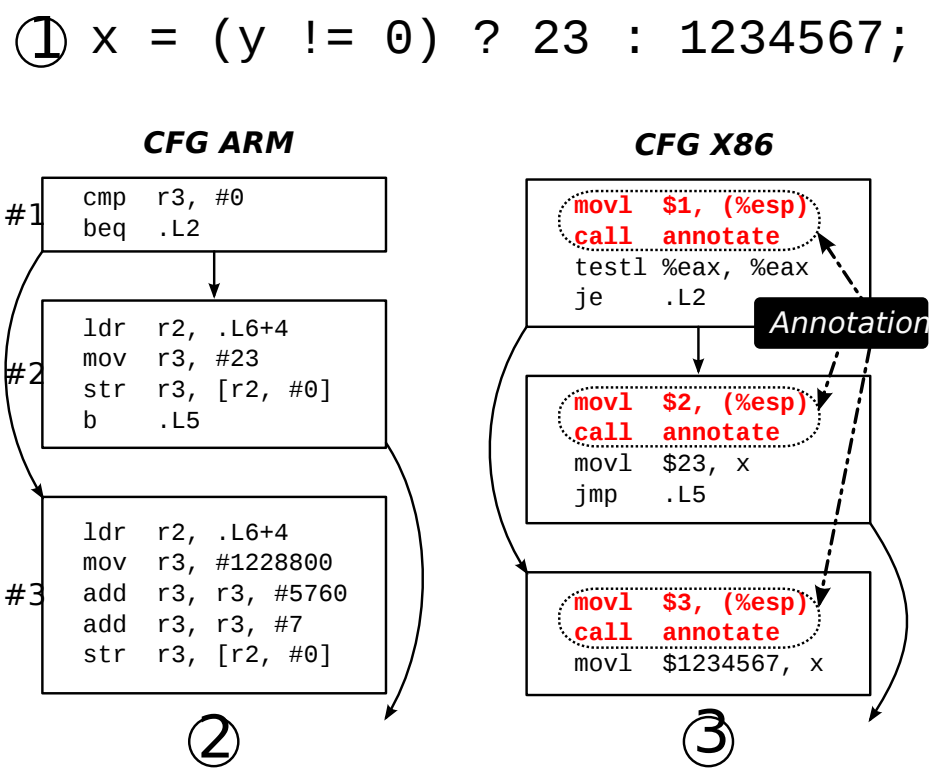
1. Code source à instrumenter
2. Code compilé pour le processeur cible (ARM)
3. Code compilé pour le processeur hôte (x86)
   - Annotations insérées en début de chaque bloc de base
   - L'argument est un identifiant du bloc cible correspondant



Instrumentation automatique du logiciel embarqué

Concepts et problématiques

1. Code source à instrumenter
2. Code compilé pour le processeur cible (ARM)
3. Code compilé pour le processeur hôte (x86)
   - Annotations insérées en début de chaque bloc de base
   - L'argument est un identifiant du bloc cible correspondant
4. Suppose que les CFG hôte et cible sont isomorphes:

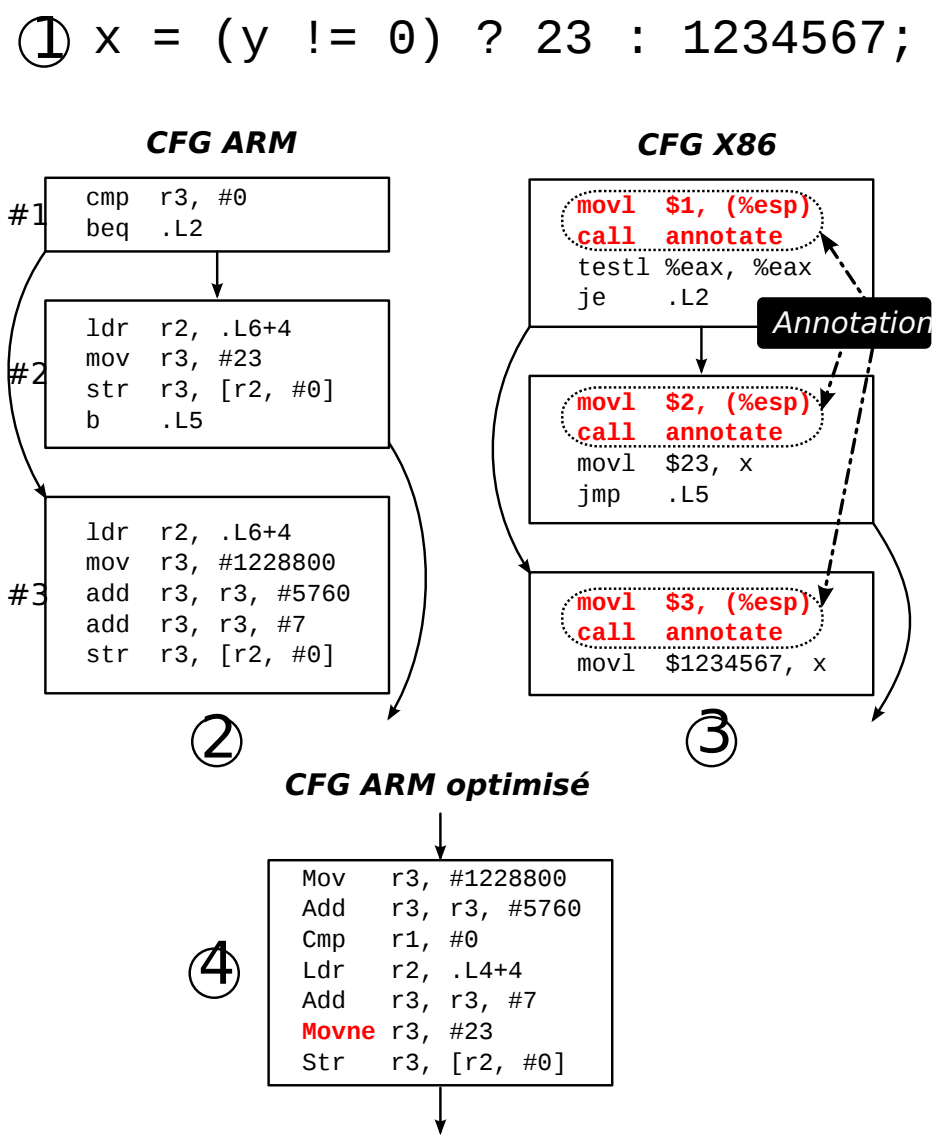


Instrumentation automatique du logiciel embarqué

Concepts et problématiques

1. Code source à instrumenter
2. Code compilé pour le processeur cible (ARM)
3. Code compilé pour le processeur hôte (x86)
   - Annotations insérées en début de chaque bloc de base
   - L'argument est un identifiant du bloc cible correspondant
4. Suppose que les CFG hôte et cible sont isomorphes:

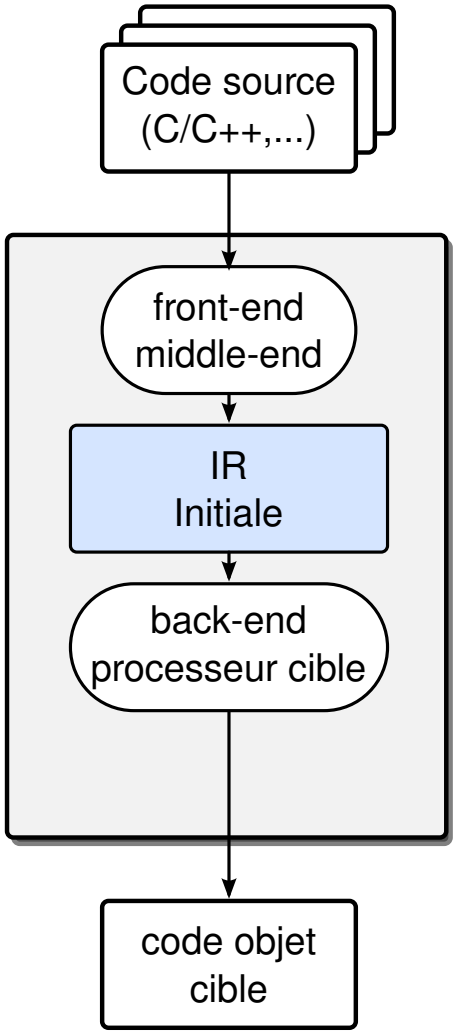
⇒ Ce n'est généralement pas le cas



Instrumentation à la compilation

Idée : utiliser la représentation intermédiaire (IR) du compilateur

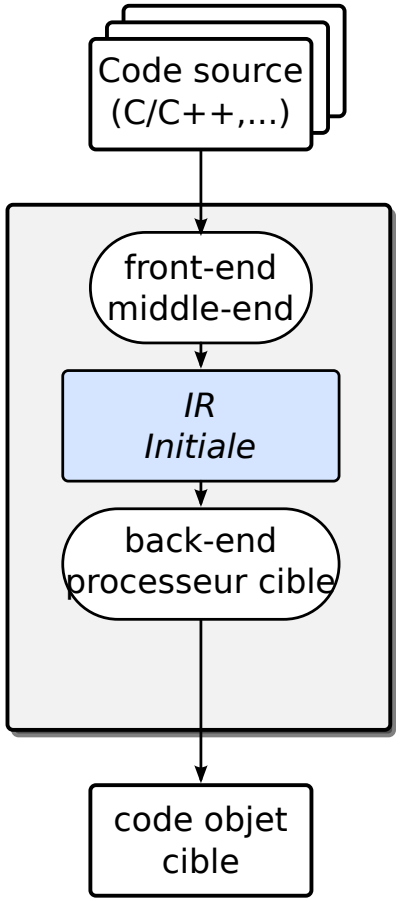
- Indépendante de la machine hôte
- Indépendante du langage de programmation utilisé
- Contient toutes les informations relatives aux CFGs



Instrumentation à la compilation

La IR étendue

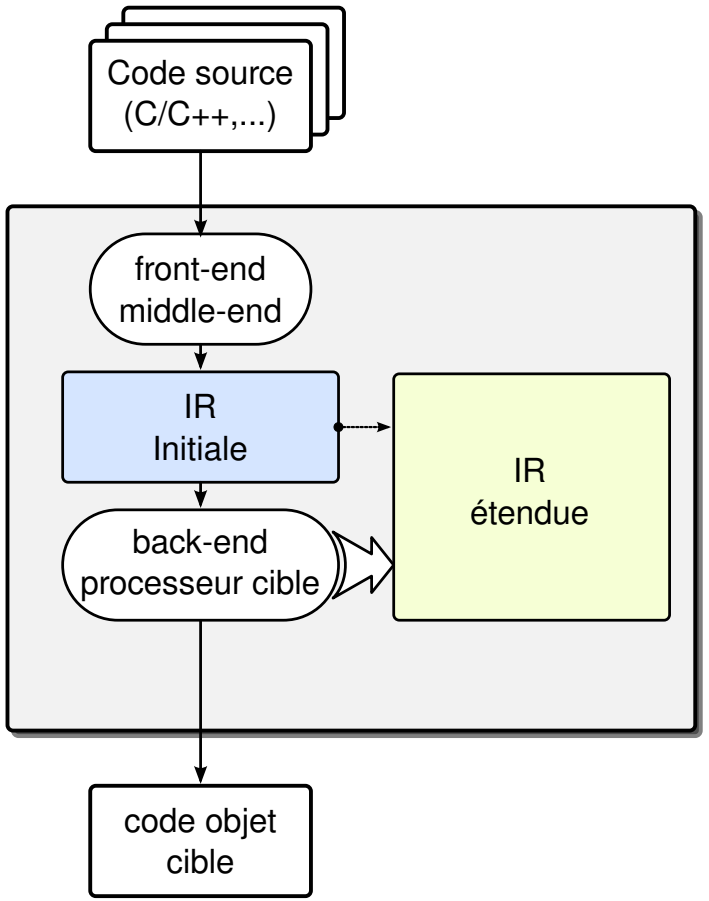
- Maintien des transformations de CFG
- Instrumentation en fin de back-end
- Compilation de la IR annotée pour le processeur hôte



Instrumentation à la compilation

La IR étendue

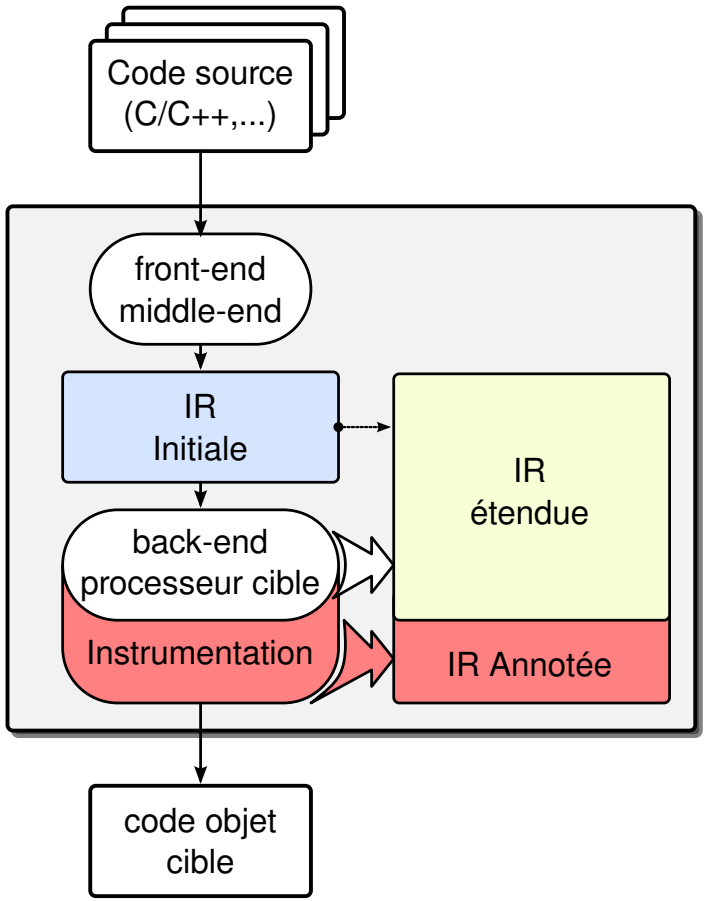
- Maintien des transformations de CFG
- Instrumentation en fin de back-end
- Compilation de la IR annotée pour le processeur hôte



Instrumentation à la compilation

La IR étendue

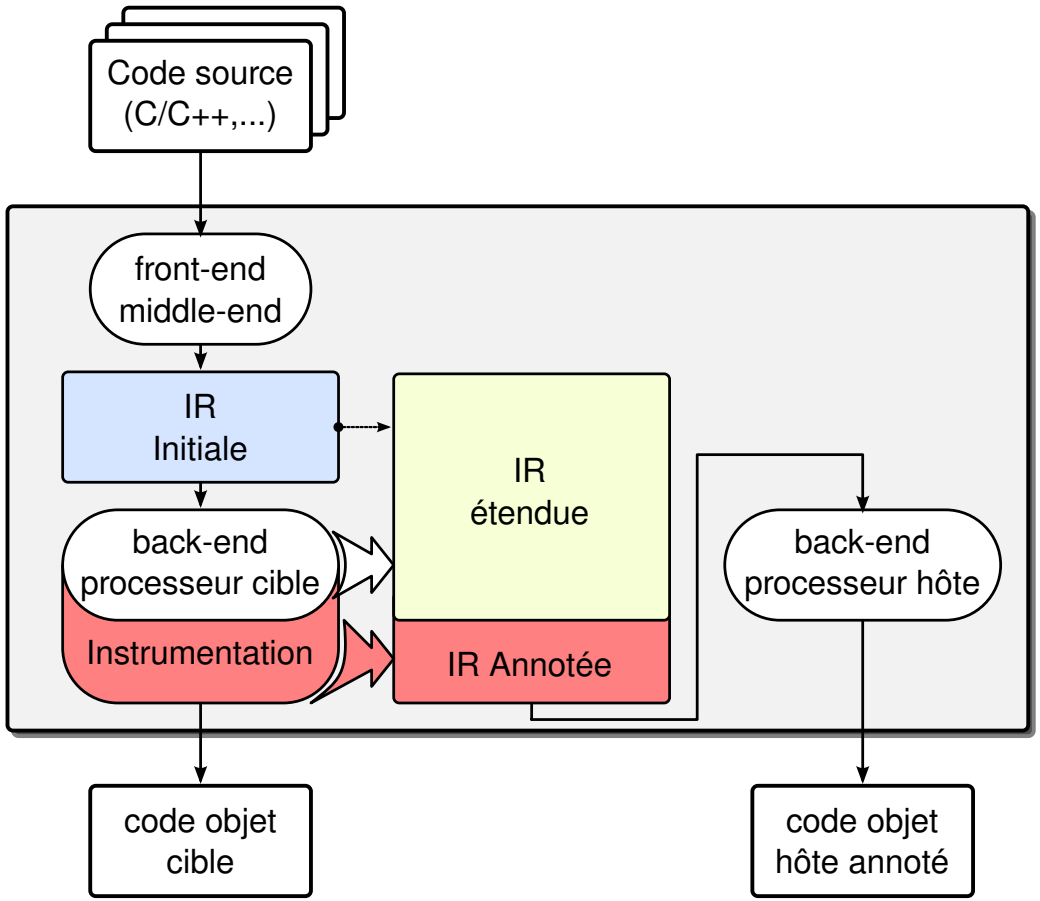
- Maintien des transformations de CFG
- Instrumentation en fin de back-end
- Compilation de la IR annotée pour le processeur hôte



Instrumentation à la compilation

La IR étendue

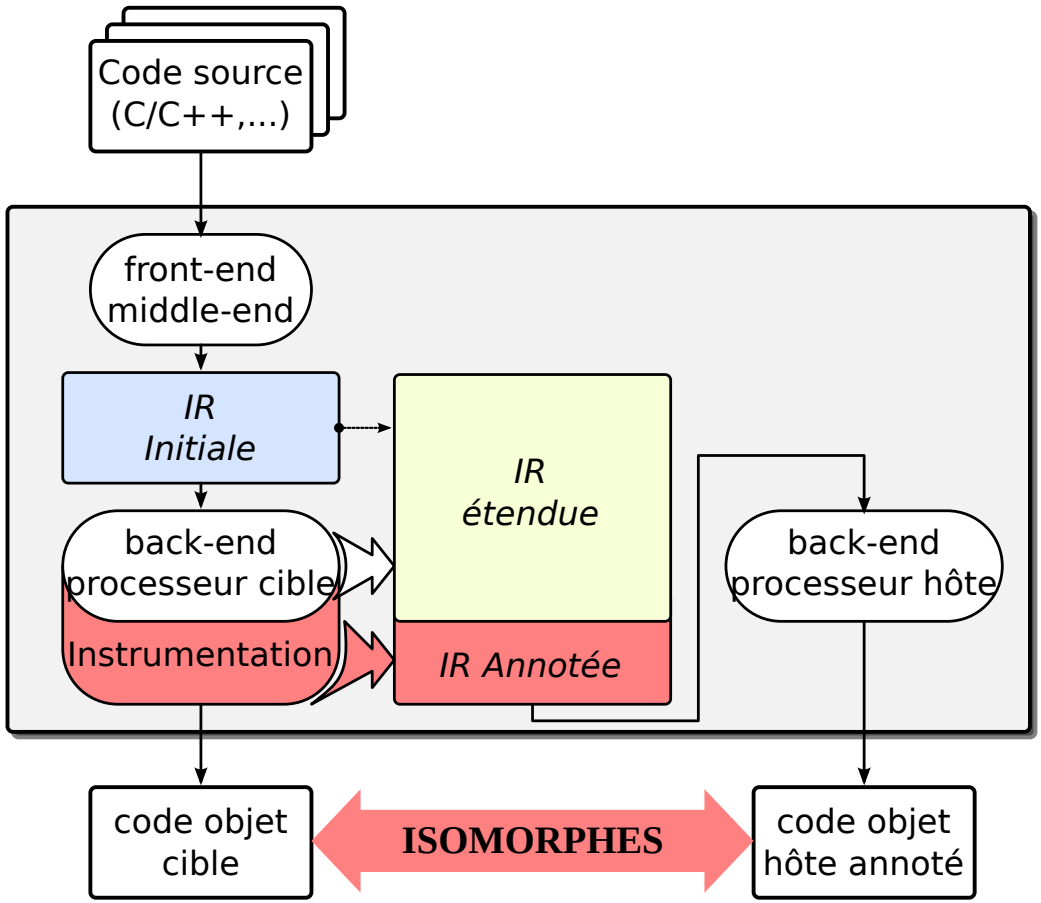
- Maintien des transformations de CFG
- Instrumentation en fin de back-end
- Compilation de la IR annotée pour le processeur hôte



Instrumentation à la compilation

La IR étendue

- Maintien des transformations de CFG
- Instrumentation en fin de back-end
- Compilation de la IR annotée pour le processeur hôte



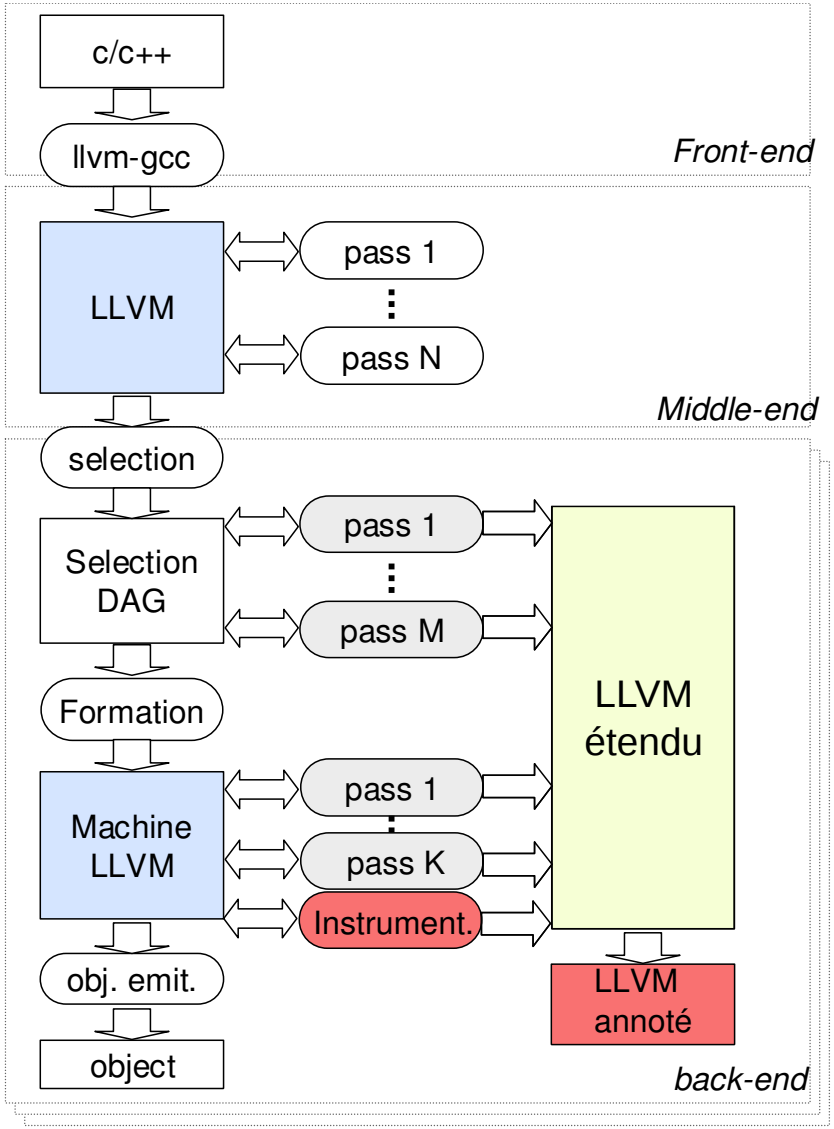
Instrumentation à la compilation

La IR étendue

- Maintien des transformations de CFG
- Instrumentation en fin de back-end
- Compilation de la IR annotée pour le processeur hôte

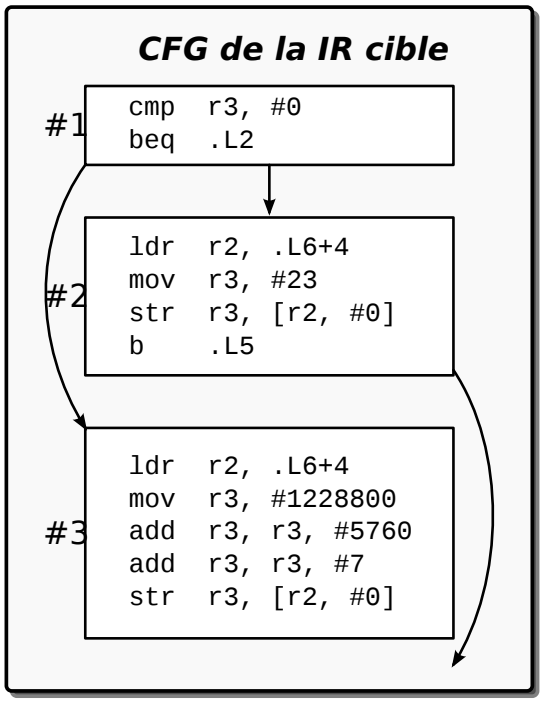
Réalisation dans LLVM

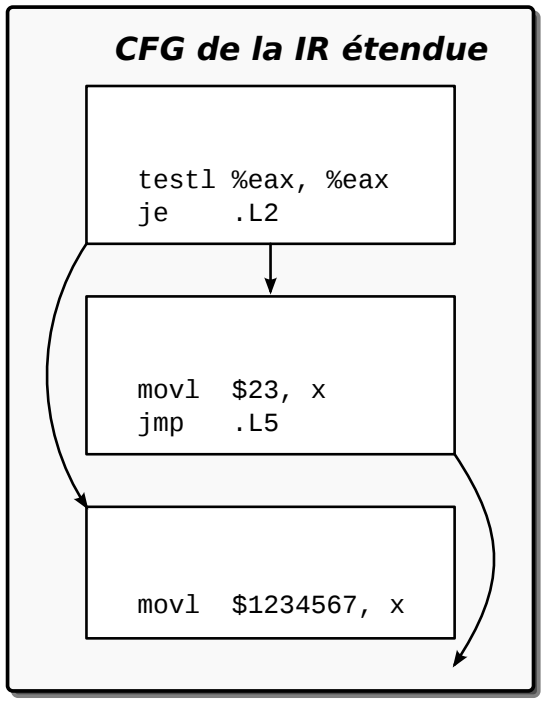
- Infrastructure de compilateur Open Source
- Une représentation intermédiaire



Instrumentation des blocs de base

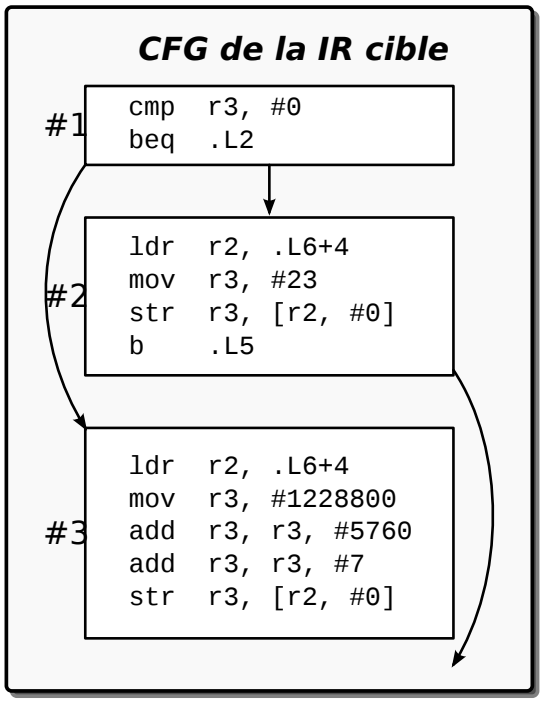
Pour chaque bloc de base du CFG du programme natif

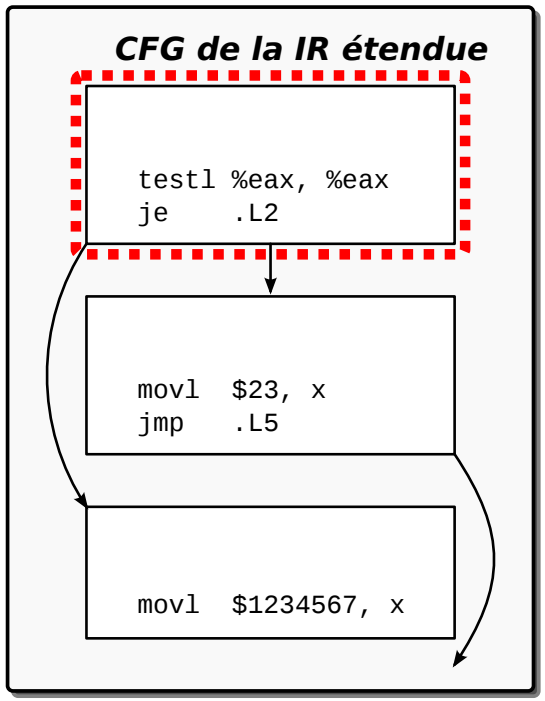




Instrumentation des blocs de base

Pour chaque bloc de base du CFG du programme natif

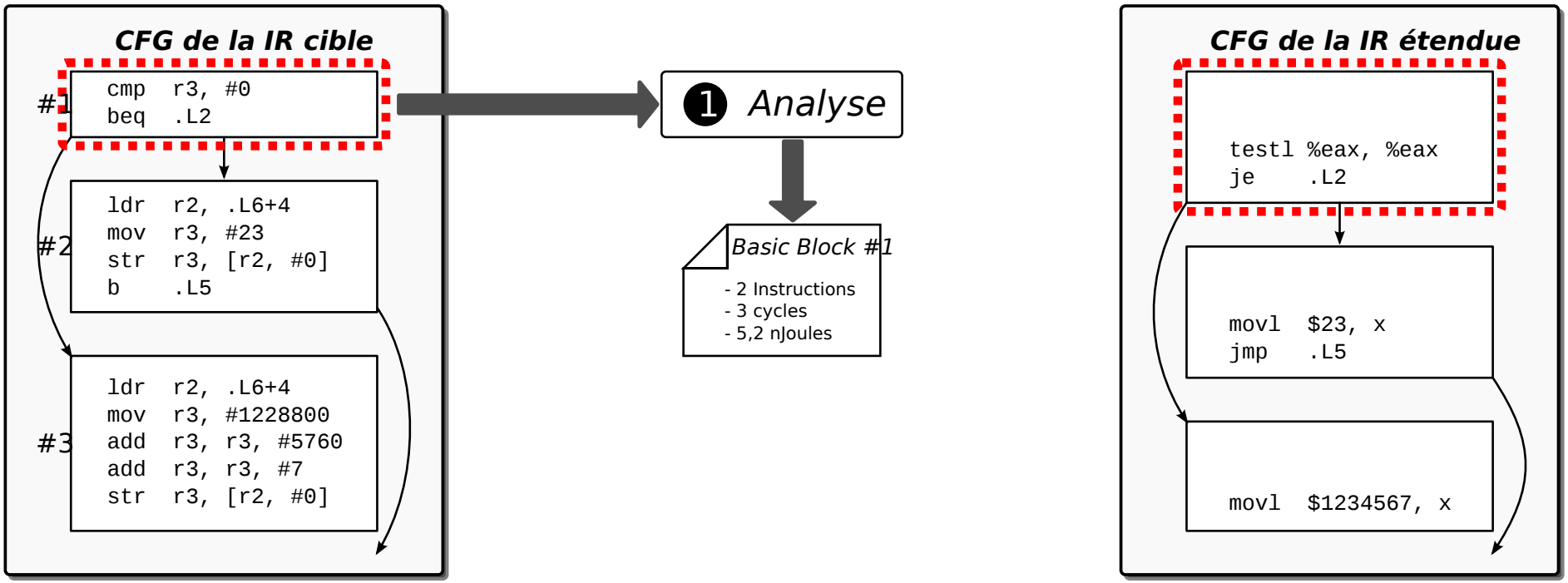




Instrumentation des blocs de base

Pour chaque bloc de base du CFG du programme natif

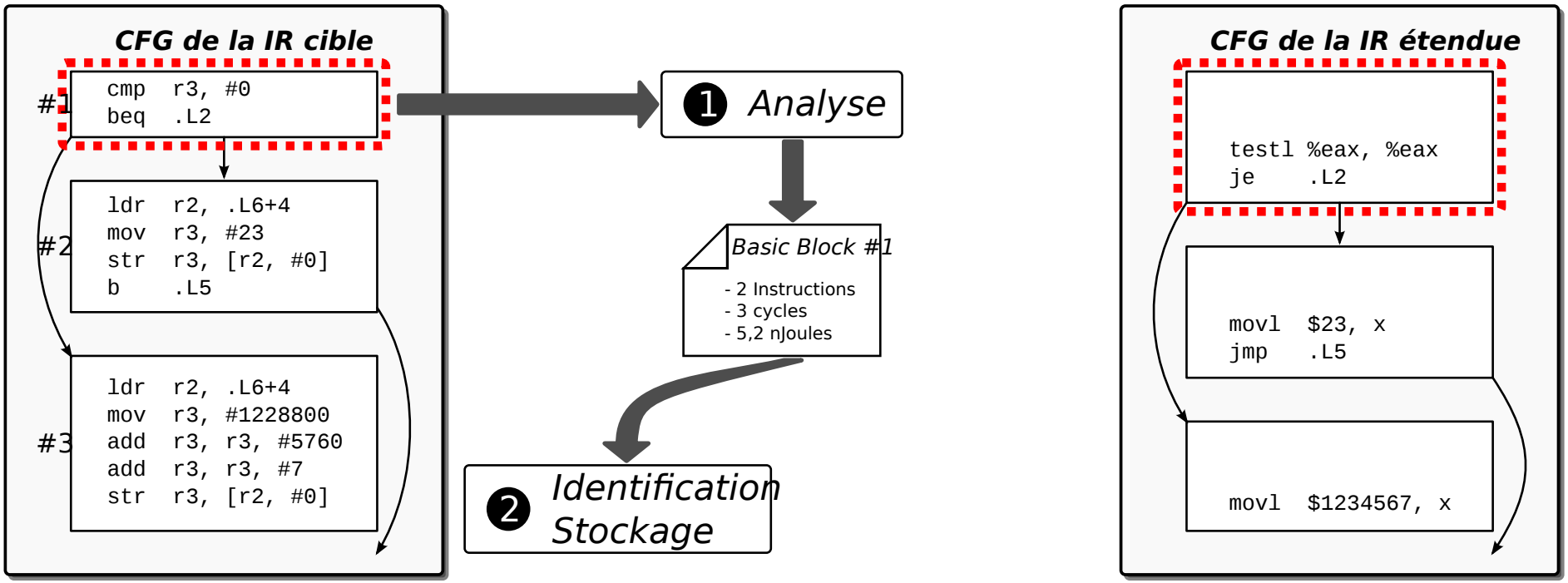
1. Analyse: extraire les informations d'annotation



Instrumentation des blocs de base

Pour chaque bloc de base du CFG du programme natif

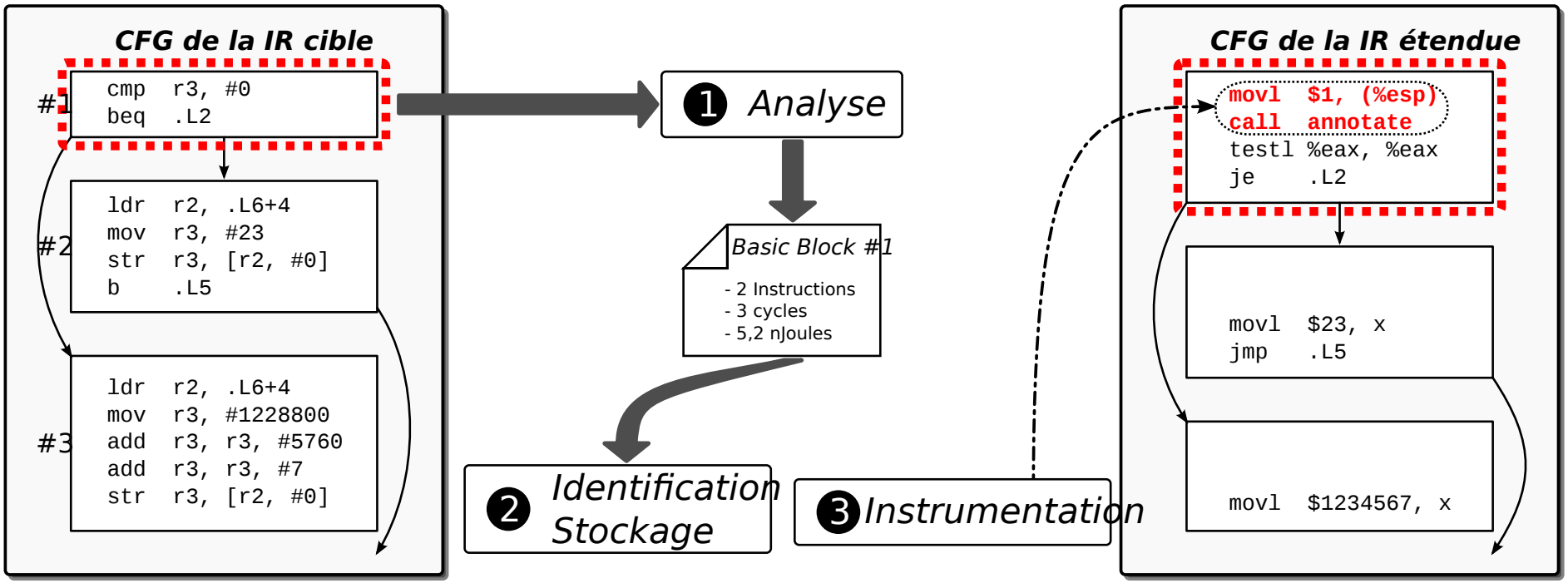
1. Analyse: extraire les informations d'annotation
2. Identification/Stockage: retrouver les informations lors de l'exécution



Instrumentation des blocs de base

Pour chaque bloc de base du CFG du programme natif

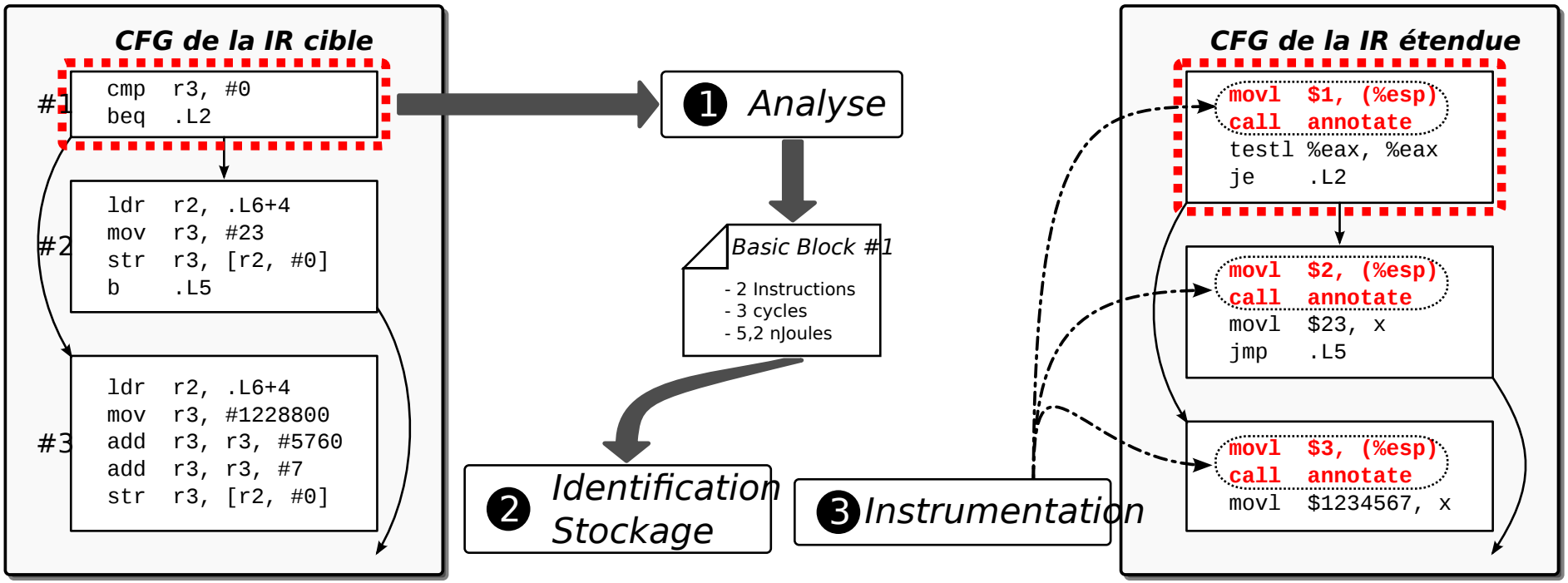
1. Analyse: extraire les informations d'annotation
2. Identification/Stockage: retrouver les informations lors de l'exécution
3. Instrumentation: insérer l'appel à la fonction d'annotation



Instrumentation des blocs de base

Pour chaque bloc de base du CFG du programme natif

1. Analyse: extraire les informations d'annotation
2. Identification/Stockage: retrouver les informations lors de l'exécution
3. Instrumentation: insérer l'appel à la fonction d'annotation



Instrumentation automatique du logiciel embarqué

Limitations

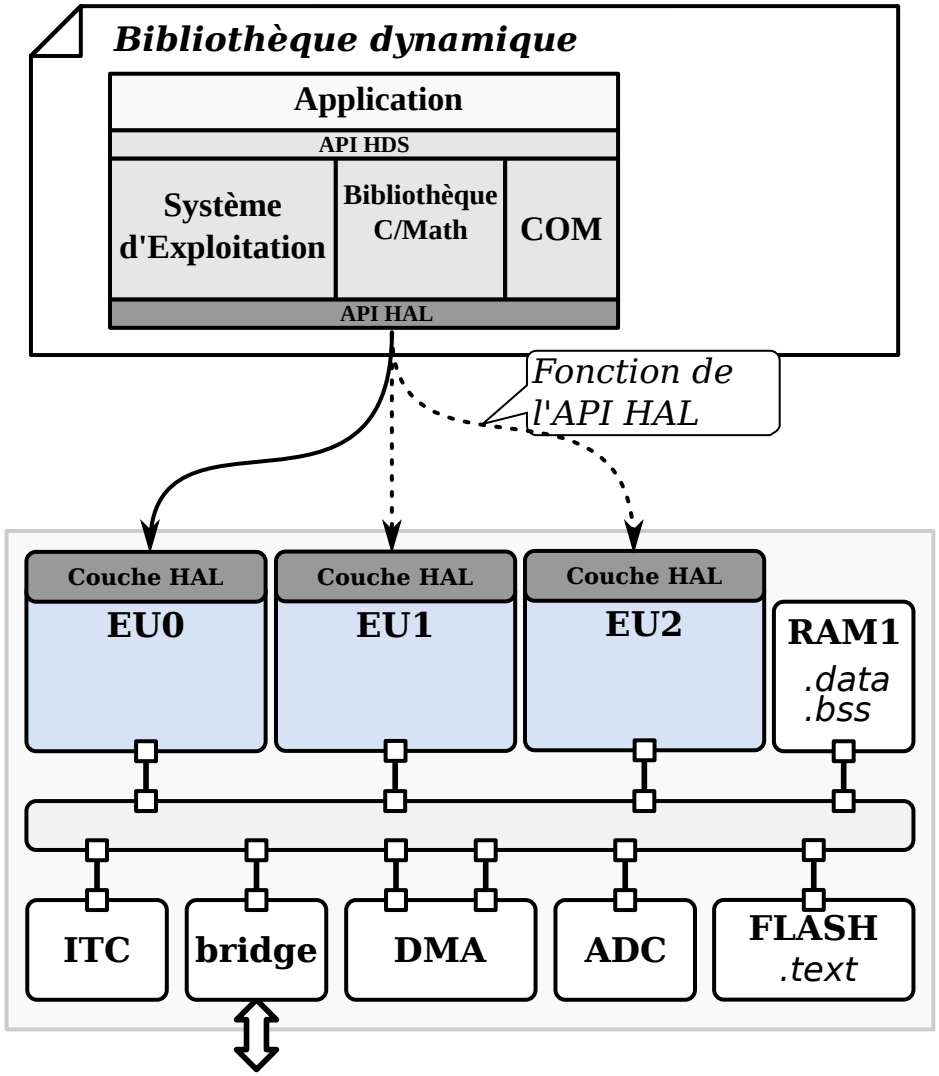
- Le compilateur doit supporter à la fois le processeur cible et le processeur hôte
- Implémentation de la technique proposée:
  - Difficulté à maintenir l'équivalence entre CFG hôte et CFG cible
  - Nécessite idéalement la maîtrise de toutes les passes de compilation
- Disposer impérativement du code source, sans instruction assembleur
- Code auto-modifiant non supporté
- Optimisation à l'édition de liens (LTO)

Logiciel instrumenté et plateforme TLM

Principe de base

- Annotations: avec le binaire du logiciel
- Interceptés par les EU

Utilisation dans le modèle TLM

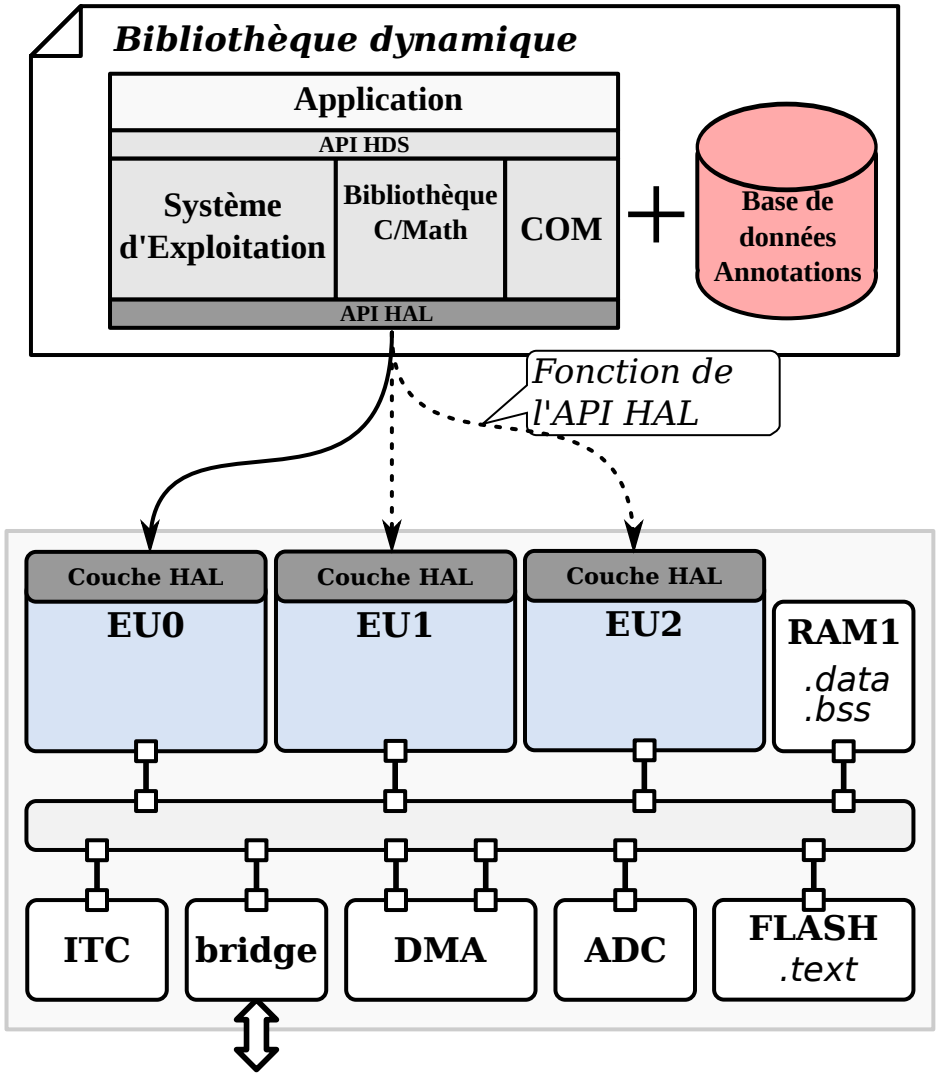


Logiciel instrumenté et plateforme TLM

Principe de base

- Annotations: avec le binaire du logiciel
- Interceptés par les EU

Utilisation dans le modèle TLM

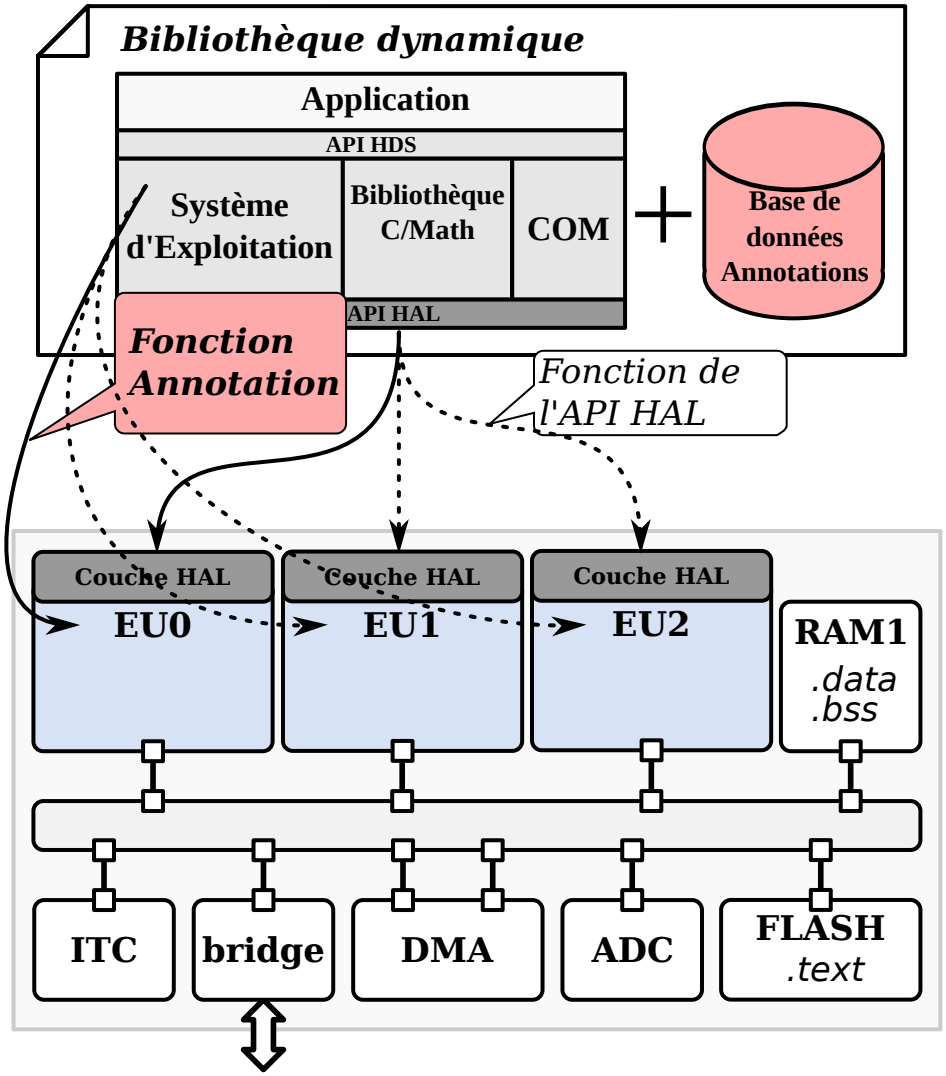


Logiciel instrumenté et plateforme TLM

Principe de base

- Annotations: avec le binaire du logiciel
- Interceptés par les EU

Utilisation dans le modèle TLM



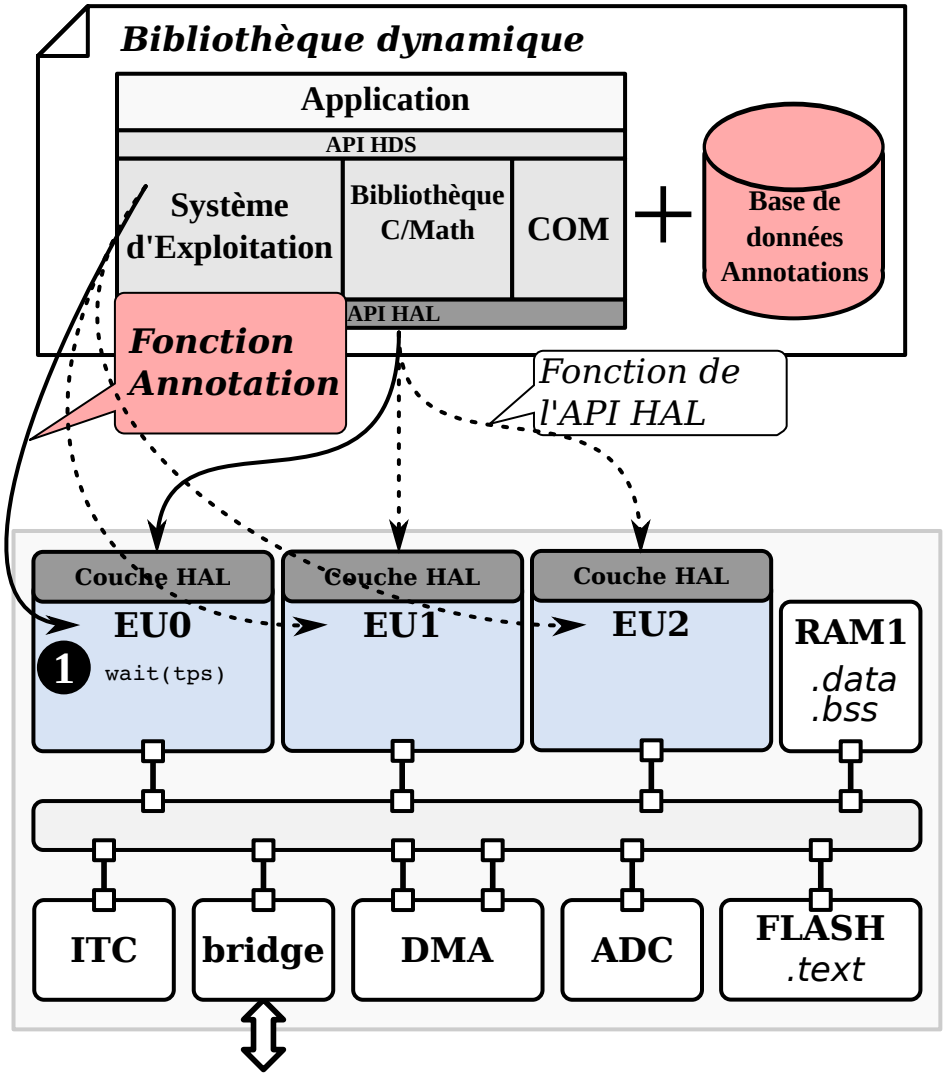
Logiciel instrumenté et plateforme TLM

Principe de base

- Annotations: avec le binaire du logiciel
- Interceptés par les EU

Utilisation dans le modèle TLM

1. Pour estimer des grandeurs physiques (cycles)
   ⇒ Estimation statique



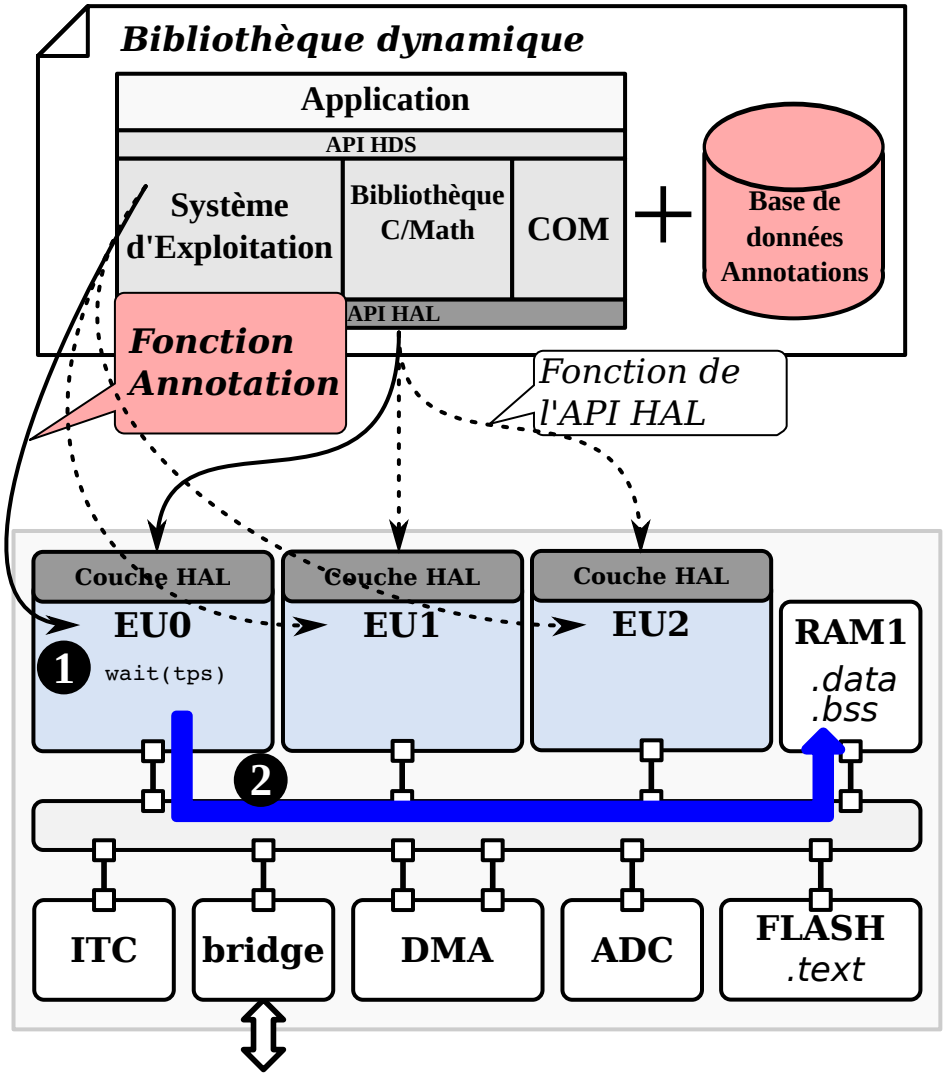
Logiciel instrumenté et plateforme TLM

Principe de base

- Annotations: avec le binaire du logiciel
- Interceptés par les EU

Utilisation dans le modèle TLM

1. Pour estimer des grandeurs physiques (cycles)
   ⇒ Estimation statique
2. Pour modéliser des accès mémoire (données/instructions)
   ⇒ Estimation dynamique



Estimation du temps d'exécution du logiciel

Analyse des blocs de base

- Indépendante du processus d'instrumentation
- Déterminer le nombre de cycles d'exécution d'un bloc de base

Le nombre de cycles exécutés dépend de

- L'architecture interne du processeur
  - Opérandes des instructions
  - Dépendances entre instructions
  - Pénalités de branchement
- L'architecture externe au processeur
  - Mémoire caches
  - Accès mémoires

Estimation du temps d'exécution du logiciel

Analyse des blocs de base

- Indépendante du processus d'instrumentation
- Déterminer le nombre de cycles d'exécution d'un bloc de base

Le nombre de cycles exécutés dépend de

- L'architecture interne du processeur
  - Opérandes des instructions
  - Dépendances entre instructions
  - Pénalités de branchement
- L'architecture externe au processeur
  - Mémoire caches
  - Accès mémoires

Assembleur ARM9

```
#1  .LBB3_0: @bb12
 1  stmfd     sp!, {r4,r5,r6,r7,lr}
 2  add       sp,sp,#100
 3  ldmfd     sp!, {r4,r5,r6, pc}
 4  mul       r0,r4,r5
 5  add       r1,r0,r6
 6  cmp       r1,#2, 24 @ 256
 7  bge       .LBB3_2 @ return
```

Estimation du temps d'exécution du logiciel

Analyse des blocs de base

- Indépendante du processus d'instrumentation
- Déterminer le nombre de cycles d'exécution d'un bloc de base

Le nombre de cycles exécutés dépend de

- L'architecture interne du processeur
  - Opérandes des instructions
  - Dépendances entre instructions
  - Pénalités de branchement
- L'architecture externe au processeur
  - Mémoire caches
  - Accès mémoires

```
    .LBB2_20: @bb33
1   stmfb sp!,{r4,r5,r6,lr}
2   ldrb r4, [sp, #+1280]
3   orr r3, r5, r4, lsl #8
4   str r3, [r6, #+4]
5   blt .LBB1_8 @ bb9
```

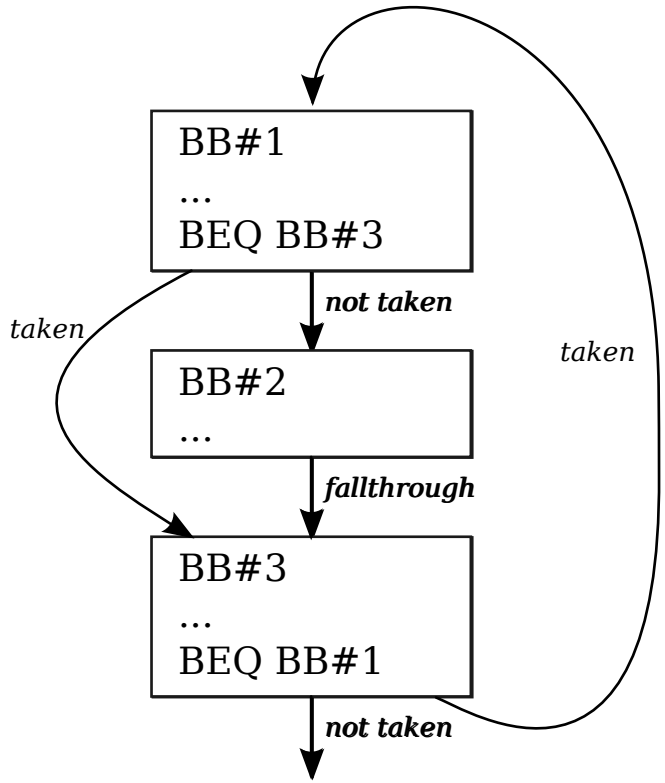
Estimation du temps d'exécution du logiciel

Analyse des blocs de base

- Indépendante du processus d'instrumentation
- Déterminer le nombre de cycles d'exécution d'un bloc de base

Le nombre de cycles exécutés dépend de

- L'architecture interne du processeur
  - Opérandes des instructions
  - Dépendances entre instructions
  - Pénalités de branchement
- L'architecture externe au processeur
  - Mémoire caches
  - Accès mémoires



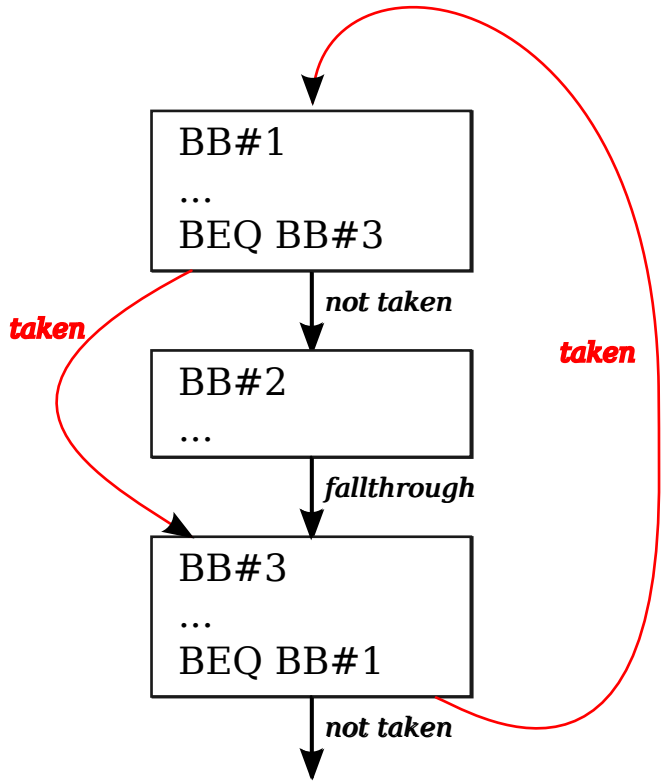
Estimation du temps d'exécution du logiciel

Analyse des blocs de base

- Indépendante du processus d'instrumentation
- Déterminer le nombre de cycles d'exécution d'un bloc de base

Le nombre de cycles exécutés dépend de

- L'architecture interne du processeur
  - Opérandes des instructions
  - Dépendances entre instructions
  - Pénalités de branchement
- L'architecture externe au processeur
  - Mémoire caches
  - Accès mémoires



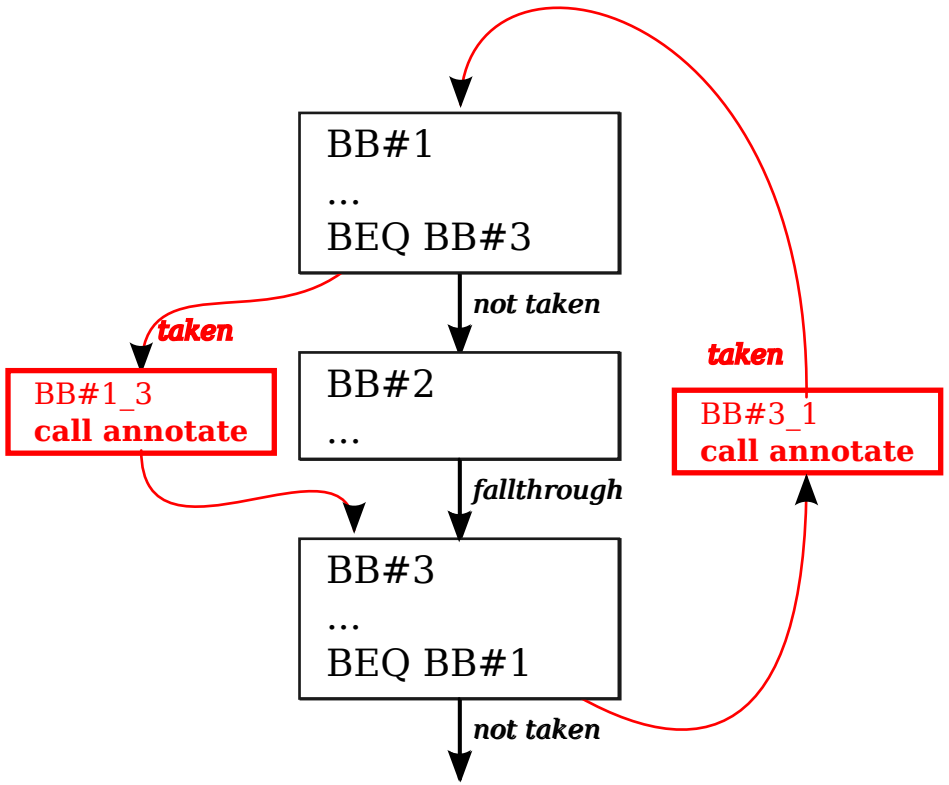
Estimation du temps d'exécution du logiciel

Analyse des blocs de base

- Indépendante du processus d'instrumentation
- Déterminer le nombre de cycles d'exécution d'un bloc de base

Le nombre de cycles exécutés dépend de

- L'architecture interne du processeur
  - Opérandes des instructions
  - Dépendances entre instructions
  - Pénalités de branchement
- L'architecture externe au processeur
  - Mémoire caches
  - Accès mémoires



Profilage logiciel

Indispensable

- Interpréter les temps d'exécution
- Affecter des coûts à chaque fonction du logiciel

Utilisation de l'instrumentation

- Insertion d'informations structurelles sur le logiciel
  - ⇒ Entrées/sorties de fonctions
- Construction de fichiers de profilage Valgrind
  - ⇒ Répercutions des coûts sur le graphe d'appel des fonctions
  - ⇒ Support multi-tâches et multiprocesseur

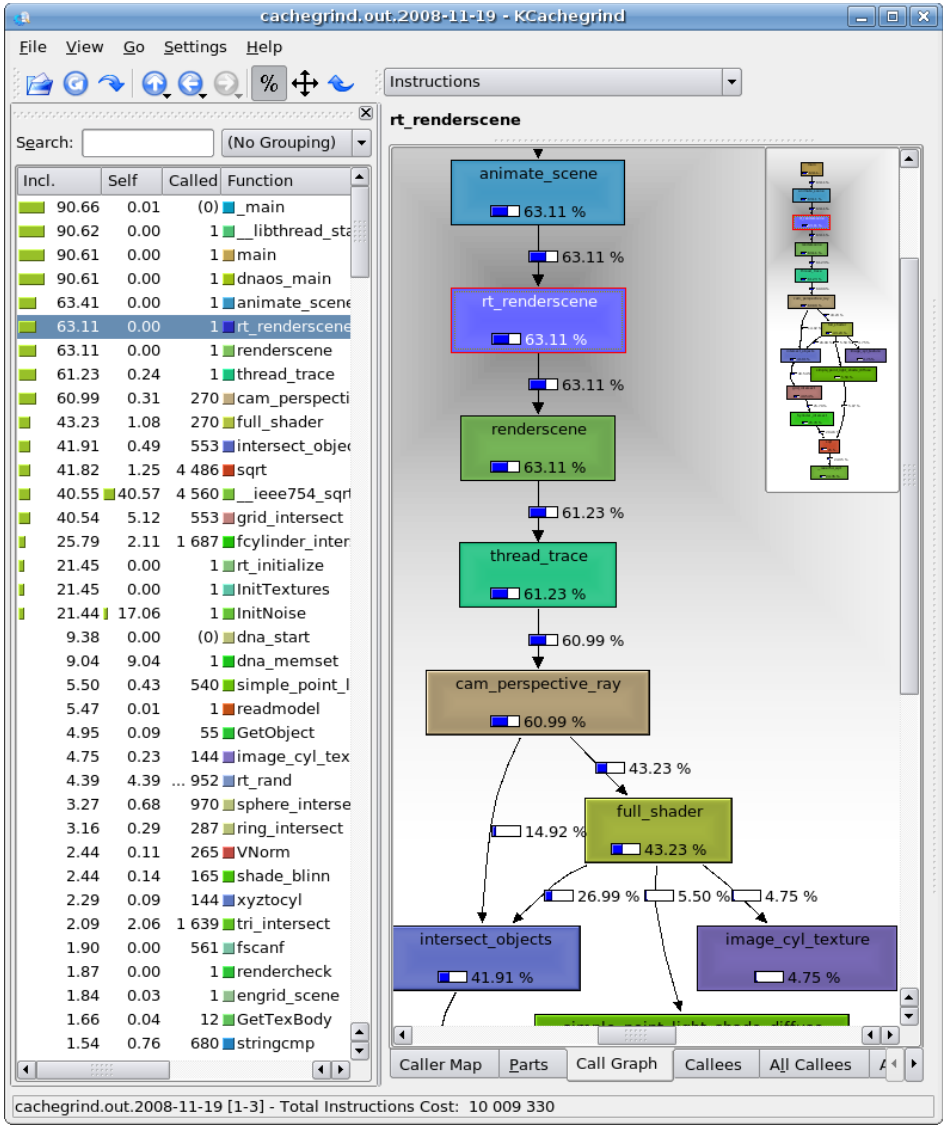
Profilage logiciel

Indispensable

- Interpréter les temps d'exécution
- Affecter des coûts à chaque fonction du logiciel

Utilisation de l'instrumentation

- Insertion d'informations structurelles sur le logiciel
  - ⇒ Entrées/sorties de fonctions
- Construction de fichiers de profilage Valgrind
  - ⇒ Répercutions des coûts sur le graphe d'appel des fonctions
  - ⇒ Support multi-tâches et multiprocesseur



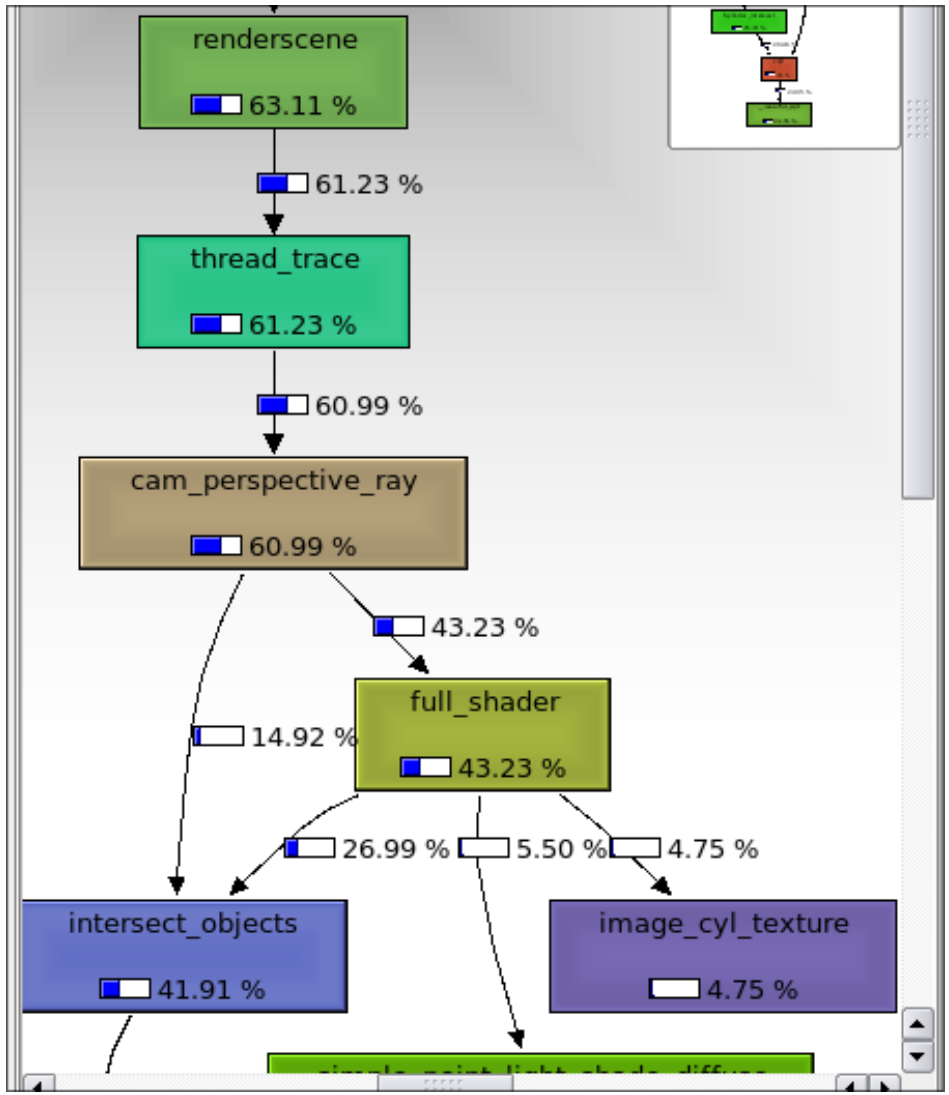
Profilage logiciel

Indispensable

- Interpréter les temps d'exécution
- Affecter des coûts à chaque fonction du logiciel

Utilisation de l'instrumentation

- Insertion d'informations structurelles sur le logiciel
  - ⇒ Entrées/sorties de fonctions
- Construction de fichiers de profilage Valgrind
  - ⇒ Répercutions des coûts sur le graphe d'appel des fonctions
  - ⇒ Support multi-tâches et multiprocesseur



Expérimentations

Objectifs

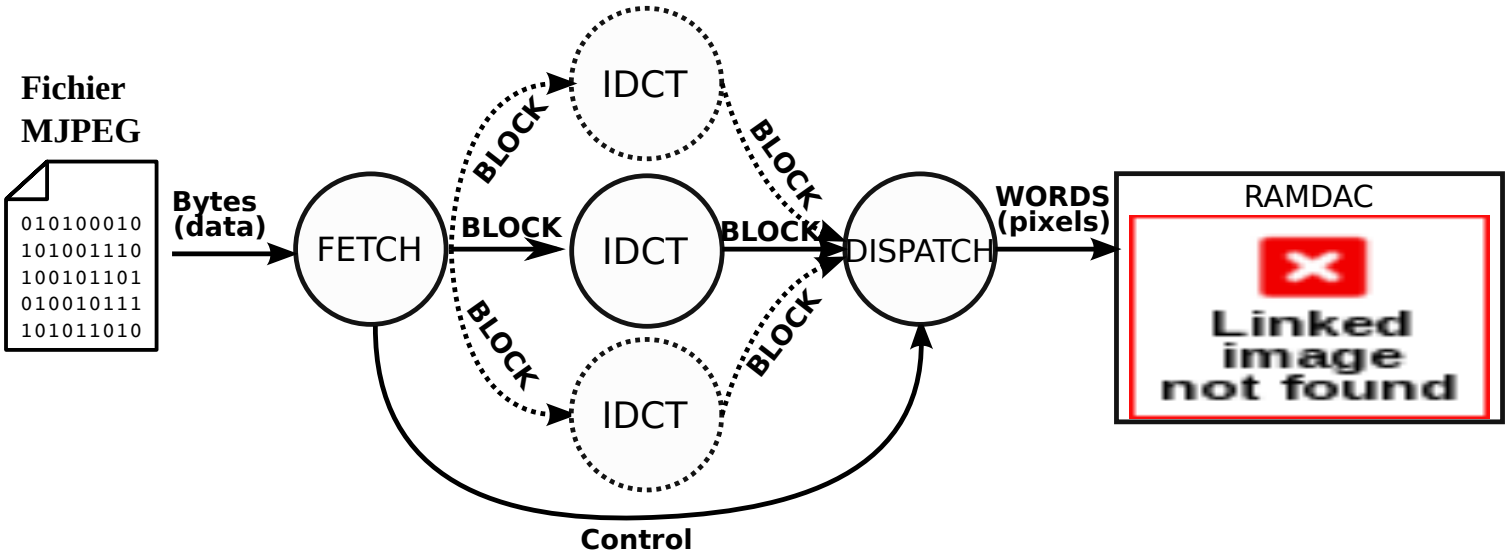
1. Montrer les capacités de la plateforme de simulation
   - $\Rightarrow$ À modéliser des architectures complexes et réalistes
   - $\Rightarrow$ À permettre la validation et le débogage du logiciel
2. Valider la technique d'instrumentation
   - $\Rightarrow$ Sa capacité à refléter l'exécution qu'aurait eu un programme sur le processeur cible

L'application Motion-JPEG

Le Motion-JPEG

- Nombre de tâches logicielles configurables facilement
- Système d'exploitation DNA et pilotes de périphériques utilisant l'API HAL
- NewlibC RedHat

Tout ce code est directement compilable pour la plateforme cible

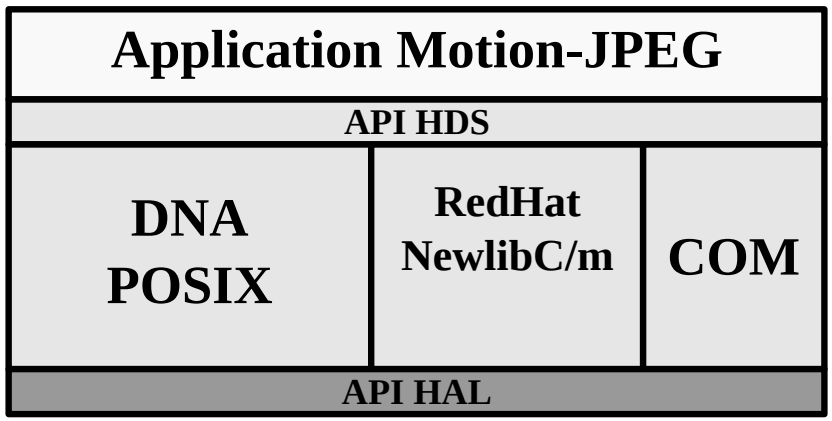


L'application Motion-JPEG

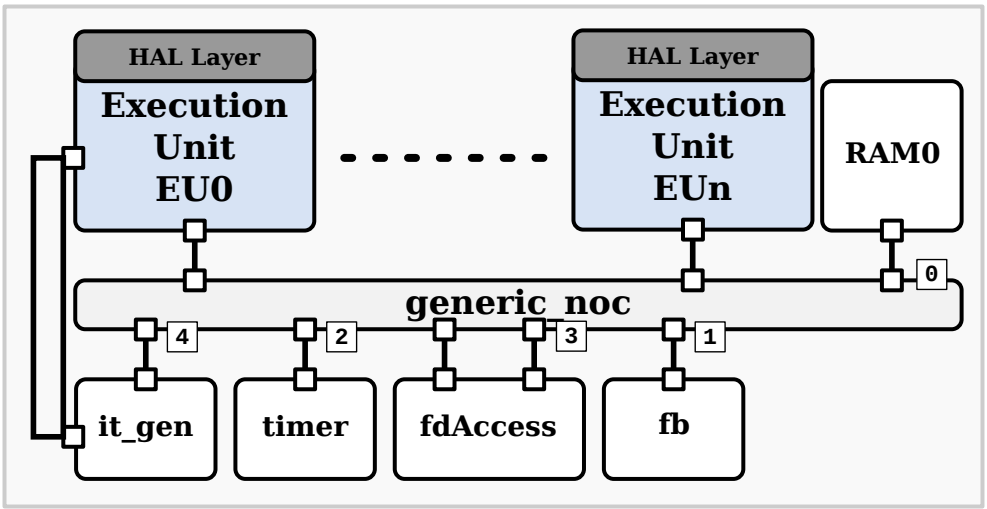
Le Motion-JPEG

- Nombre de tâches logicielles configurables facilement
- Système d'exploitation DNA et pilotes de périphériques utilisant l'API HAL
- NewlibC RedHat

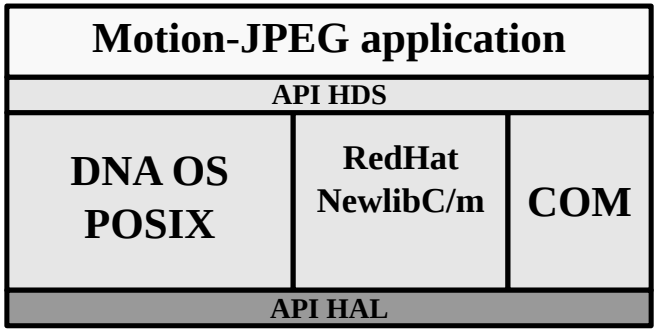
Tout ce code est directement compilable pour la plateforme cible

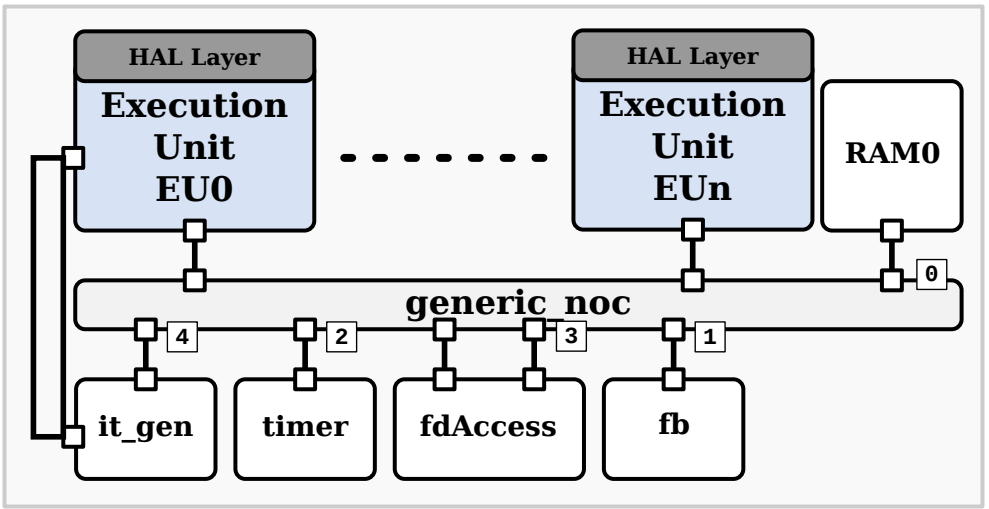


Modélisation d'une plateforme matérielle

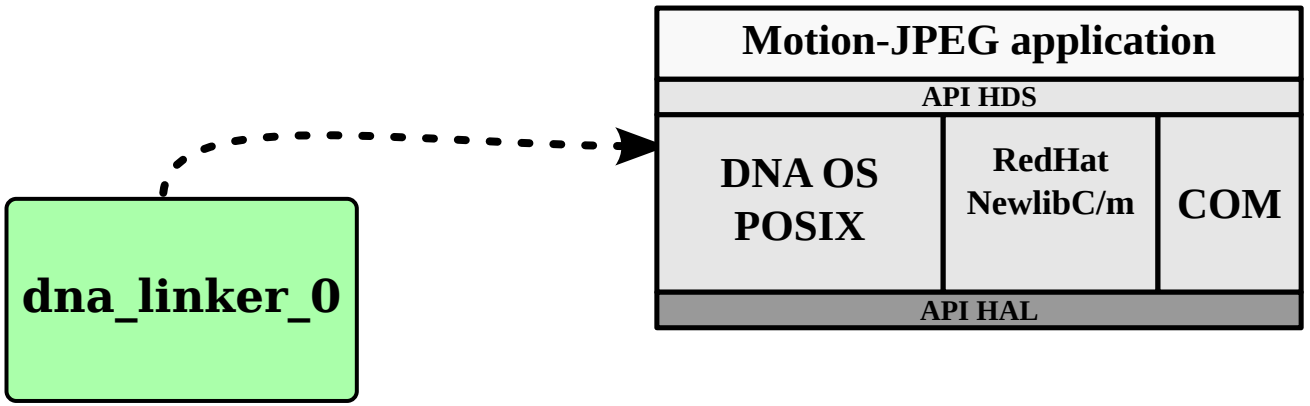


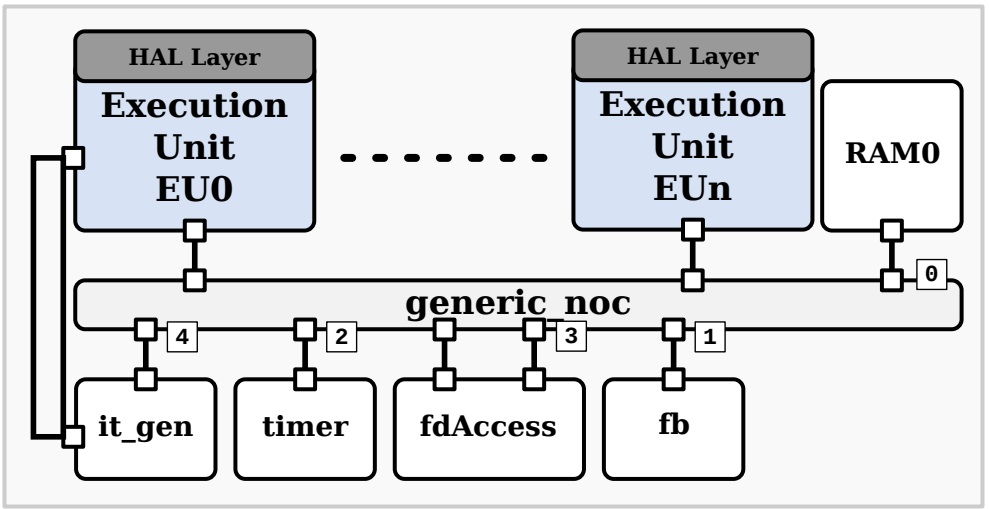
Modélisation d'une plateforme matérielle



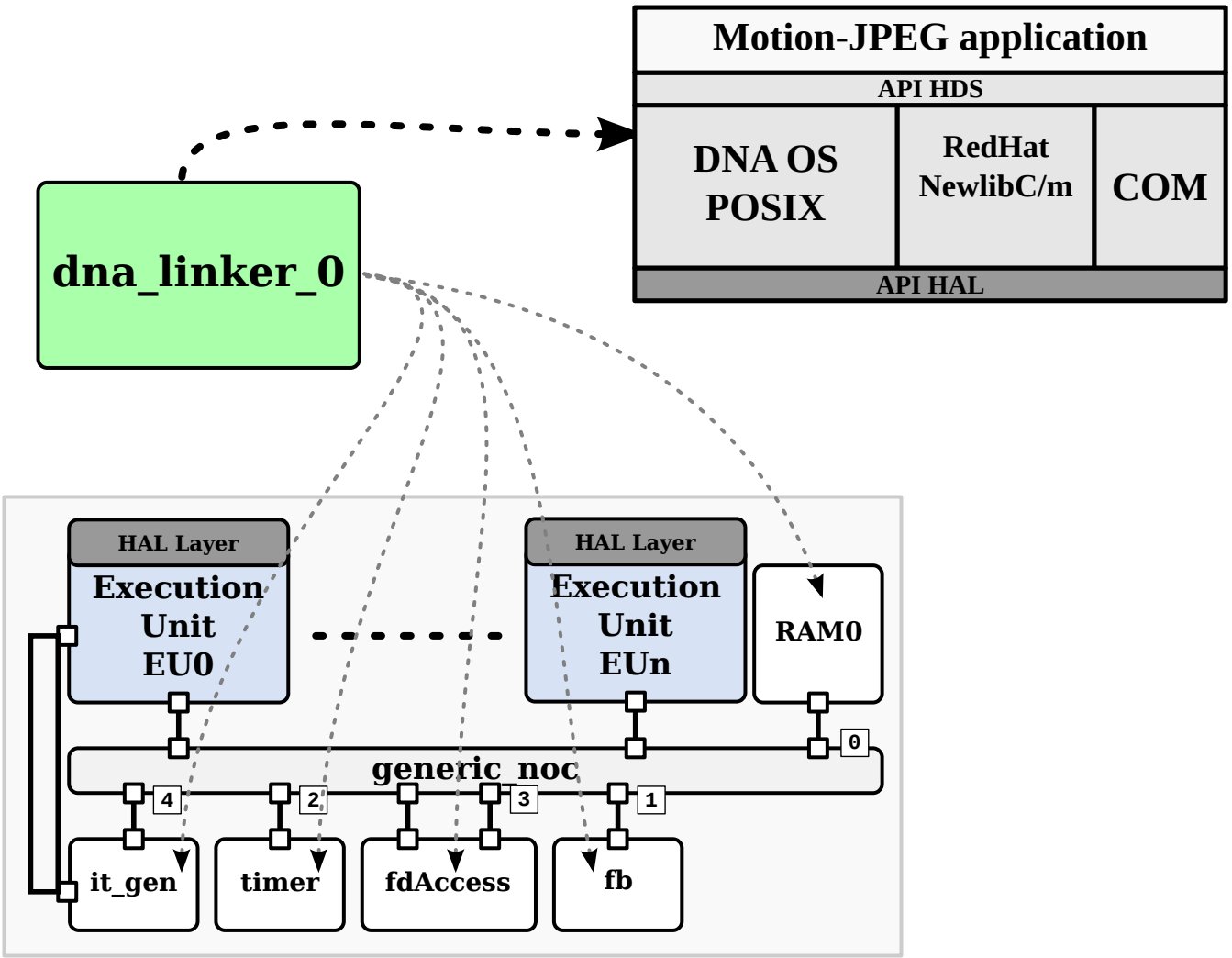


Modélisation d'une plateforme matérielle

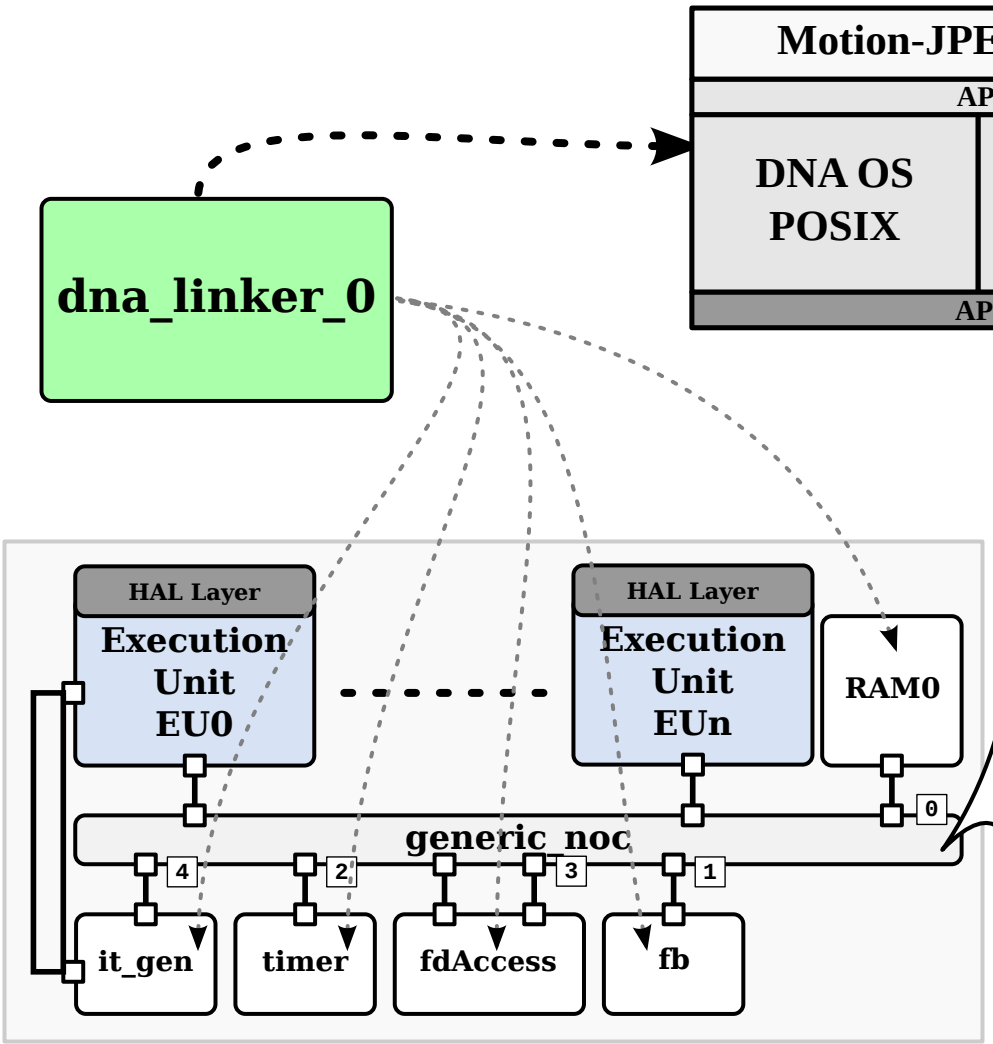




Modélisation d'une plateforme matérielle



Modélisation d'une plateforme matérielle



Section name	addresses	Port
fb_0:	0x08191948 - 0x081a3947	1
timer_0:	0x081d1650 - 0x081d165f	2
fdaccess_0:	0x081d4c28 - 0x081d4c4b	3
it_generator_0:	0x081d4cb0 - 0x081d4cb3	4
.sysstack:	0xb58dd008 - 0xb59dd007	0
.uheap:	0xb59de008 - 0xb69de007	0
.kheap:	0xb69df008 - 0xb79df007	0
.text:	0xb79ea2f0 - 0xb79fe7e6	0
.rodata:	0xb79fe800 - 0xb79ffed3	0
.data:	0xb7a00fc0 - 0xb7a01dd7	0
.bss:	0xb7a01de0 - 0xb7a01fe3	0

Performances de simulation

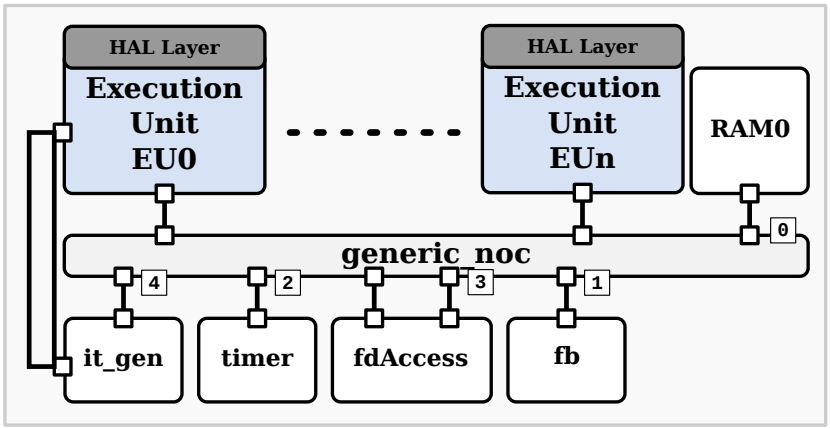
Fortement dépendantes

- Des annotations
- De l'application
- De la précision de la plateforme matérielle
- Du profilage du logiciel

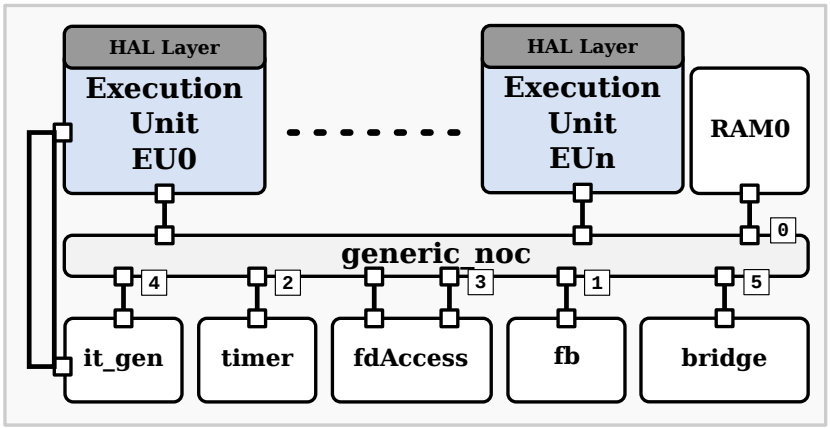
	Nombre de processeurs/EU					
Plateforme	1	2	3	4	6	8
Native	1.9	3.2	4.3	5.5	8.1	10.6
CABA	359.8	251.0	264.0	346.5	480.5	618.1
Accélération	×181	×78	×60	×62	×58	×58

Temps de simulation (secondes)

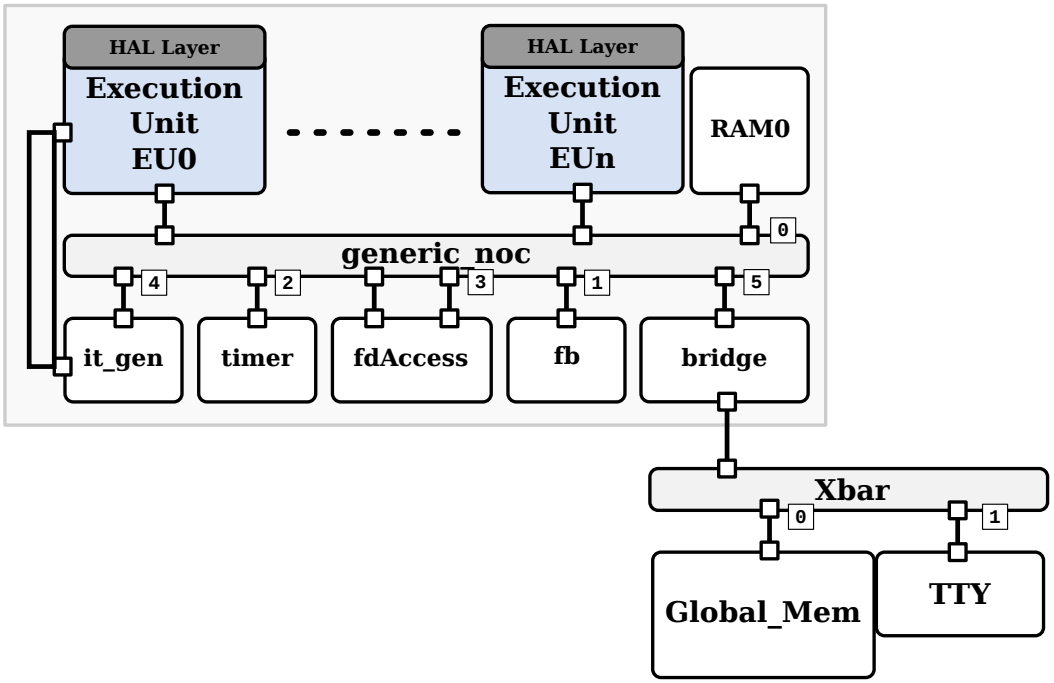
Modélisation de plusieurs nœuds logiciels



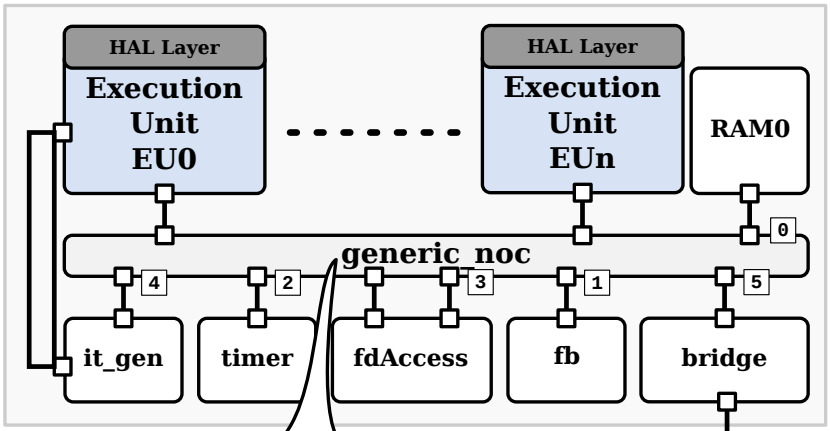
Modélisation de plusieurs nœuds logiciels



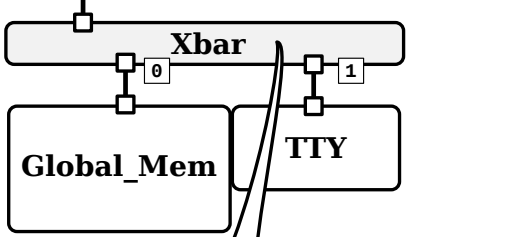
Modélisation de plusieurs nœuds logiciels



Modélisation de plusieurs nœuds logiciels

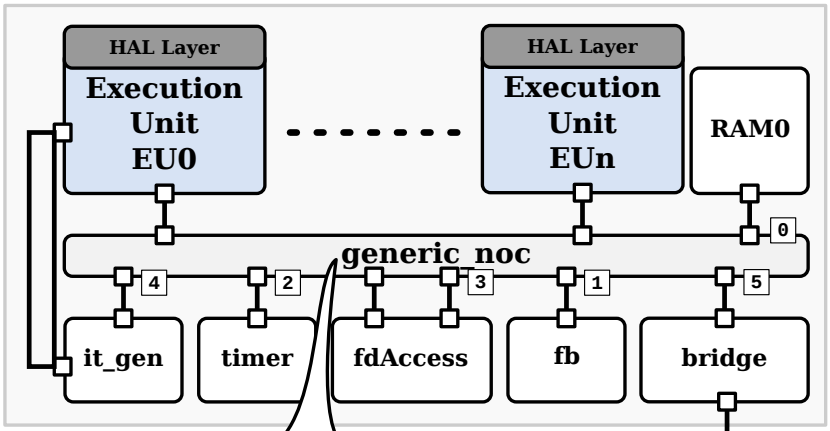


Section Name	Addresses	Port
.global_mem:	0x08181940 - 0x0819193f	5
fb_0:	0x08191948 - 0x081a3947	1
timer_0:	0x081d1650 - 0x081d165f	2
fdaccess_0:	0x081d4c28 - 0x081d4c4b	3
it_generator_0:	0x081d4cb0 - 0x081d4cb3	4
tty_pool_main:	0x081d56a8 - 0x081d56b7	5
.sysstack:	0xb58dd008 - 0xb59dd007	0
.uheap:	0xb59de008 - 0xb69de007	0
.kheap:	0xb69df008 - 0xb79df007	0
.text:	0xb79ea2f0 - 0xb79fe7e6	0
.rodata:	0xb79fe800 - 0xb79ffed3	0
.data:	0xb7a00fc0 - 0xb7a01dd7	0
.bss:	0xb7a01de0 - 0xb7a01fe3	0

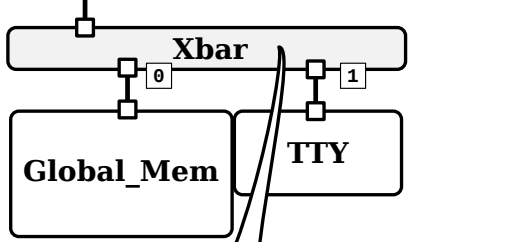


Section Name	Addresses	Port
.global_mem	0x08181940 - 0x0819193f	0
tty_pool_main	0x081d54a8 - 0x081d56b7	1

Modélisation de plusieurs nœuds logiciels

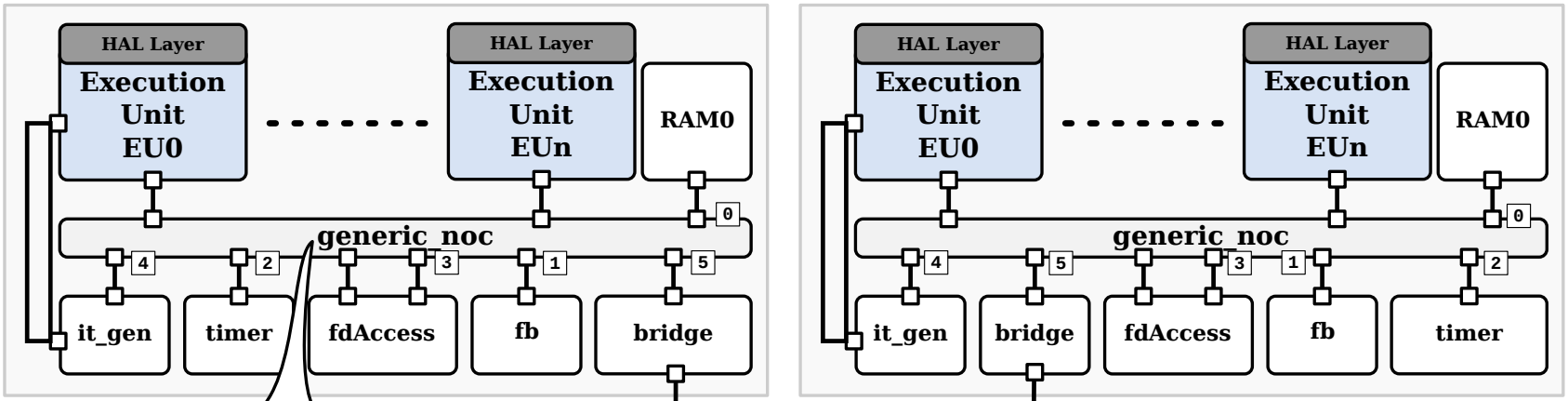


Section Name	Addresses	Port
.global_mem:	0x08181940 - 0x0819193f	
fb_0:	0x08191948 - 0x081a3947	1
timer_0:	0x081d1650 - 0x081d165f	2
fdaccess_0:	0x081d4c28 - 0x081d4c4b	3
it_generator_0:	0x081d4cb0 - 0x081d4cb3	4
tty_pool_main:	0x081d54a8 - 0x081d56b7	
.sysstack:	0xb58dd008 - 0xb59dd007	0
.uheap:	0xb59de008 - 0xb69de007	0
.kheap:	0xb69df008 - 0xb79df007	0
.text:	0xb79ea2f0 - 0xb79fe7e6	0
.rodata:	0xb79fe800 - 0xb79ffed3	0
.data:	0xb7a00fc0 - 0xb7a01dd7	0
.bss:	0xb7a01de0 - 0xb7a01fe3	0



Section Name	Addresses	Port
.global_mem:	0x08181940 - 0x0819193f	
tty_pool_main:	0x081d54a8 - 0x081d56b7	

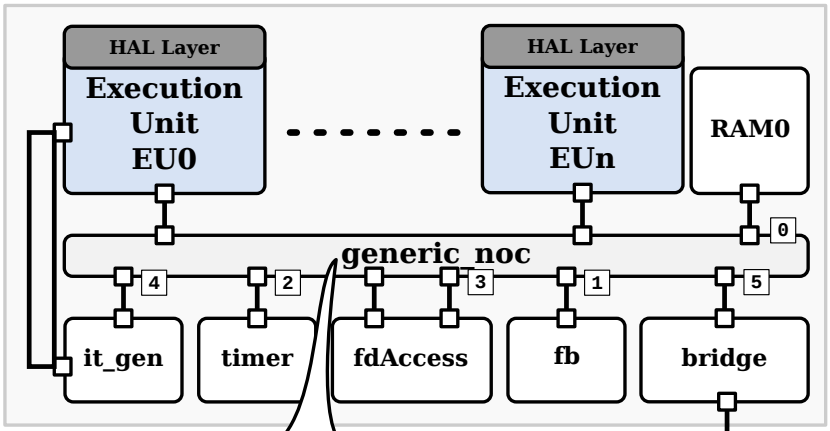
Modélisation de plusieurs nœuds logiciels

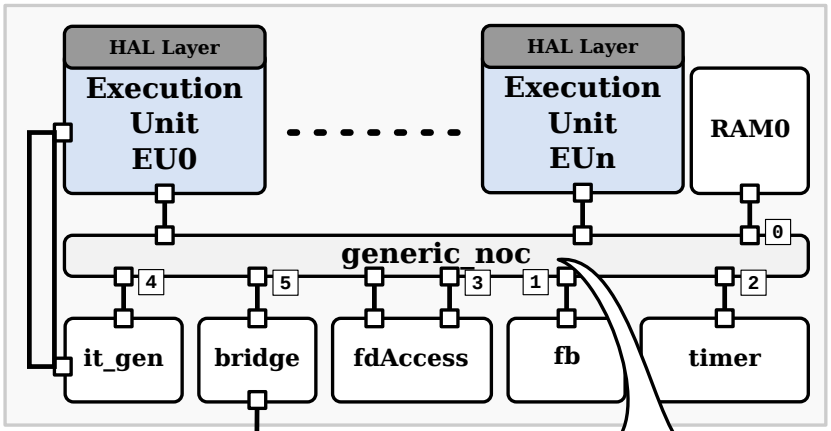


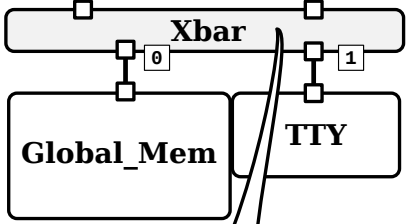
Section Name	Addresses	Port
.global_mem:	0x08181940 - 0x0819193f	5
fb_0:	0x08191948 - 0x081a3947	1
timer_0:	0x081d1650 - 0x081d165f	2
fdaccess_0:	0x081d4c28 - 0x081d4c4b	3
it_generator_0:	0x081d4cb0 - 0x081d4cb3	4
tty_pool_main:	0x081d56a8 - 0x081d56b7	5
.sysstack:	0xb58dd008 - 0xb59dd007	0
.uheap:	0xb59de008 - 0xb69de007	0
.kheap:	0xb69df008 - 0xb79df007	0
.text:	0xb79ea2f0 - 0xb79fe7e6	0
.rodata:	0xb79fe800 - 0xb79ffed3	0
.data:	0xb7a00fc0 - 0xb7a01dd7	0
.bss:	0xb7a01de0 - 0xb7a01fe3	0

Section Name	Addresses	Port
.global_mem	0x08181940 - 0x0819193f	0
tty_pool_main	0x081d54a8 - 0x081d56b7	1

Modélisation de plusieurs nœuds logiciels





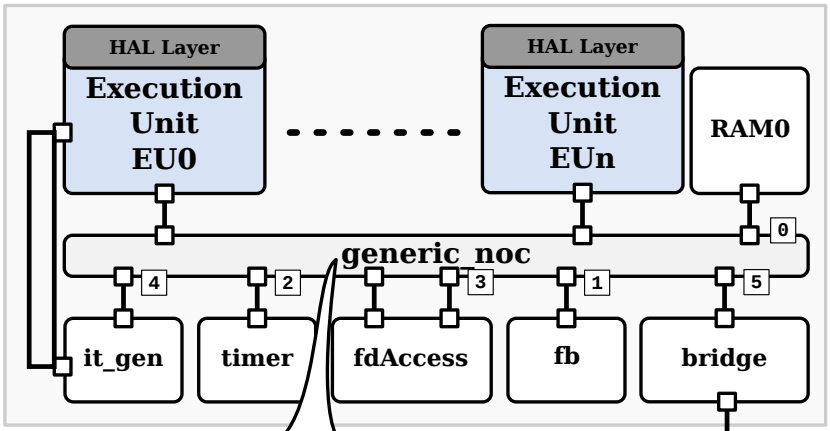


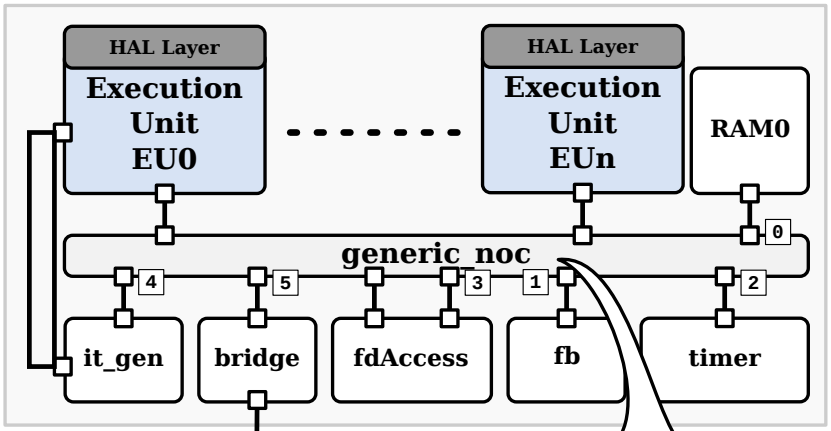
Section Name	Addresses	Port
.global_mem:	0x08181940 - 0x0819193f	5
fb_0:	0x08191948 - 0x081a3947	1
timer_0:	0x081d1650 - 0x081d165f	2
fdaccess_0:	0x081d4c28 - 0x081d4c4b	3
it_generator_0:	0x081d4cb0 - 0x081d4cb3	4
tty_pool_main:	0x081d56a8 - 0x081d56b7	5
.sysstack:	0xb58dd008 - 0xb59dd007	0
.uheap:	0xb59de008 - 0xb69de007	0
.kheap:	0xb69df008 - 0xb79df007	0
.text:	0xb79ea2f0 - 0xb79fe7e6	0
.rodata:	0xb79fe800 - 0xb79ffed3	0
.data:	0xb7a00fc0 - 0xb7a01dd7	0
.bss:	0xb7a01de0 - 0xb7a01fe3	0

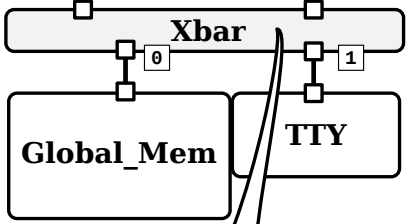
Section Name	Addresses	Port
.global_mem	0x08181940 - 0x0819193f	0
tty_pool_main	0x081d54a8 - 0x081d56b7	1

Section Name	Addresses	Port
.global_mem	0x08181940 - 0x0819193f	5
timer_1	0x081d5380 - 0x081d538f	2
fdaccess_1	0x081d54a8 - 0x081d54cb	3
it_generator	0x081d5530 - 0x081d5533	4
tty_pool_main	0x081d54a8 - 0x081d56b7	5
fb_1	0x081f98c0 - 0x0820b8bf	1
.sysstack:	0xb34e3008 - 0xb35e3007	0
.uheap:	0xb35e4008 - 0xb45e4007	0
ap:	0xb45e5008 - 0xb55e5007	0
t:	0xb563a2f0 - 0xb564e7e6	0
ata:	0xb564e800 - 0xb564fed3	0
.data:	0xb5650fc0 - 0xb5651dd7	0
.bss:	0xb5651de0 - 0xb5651fe3	0

Modélisation de plusieurs nœuds logiciels







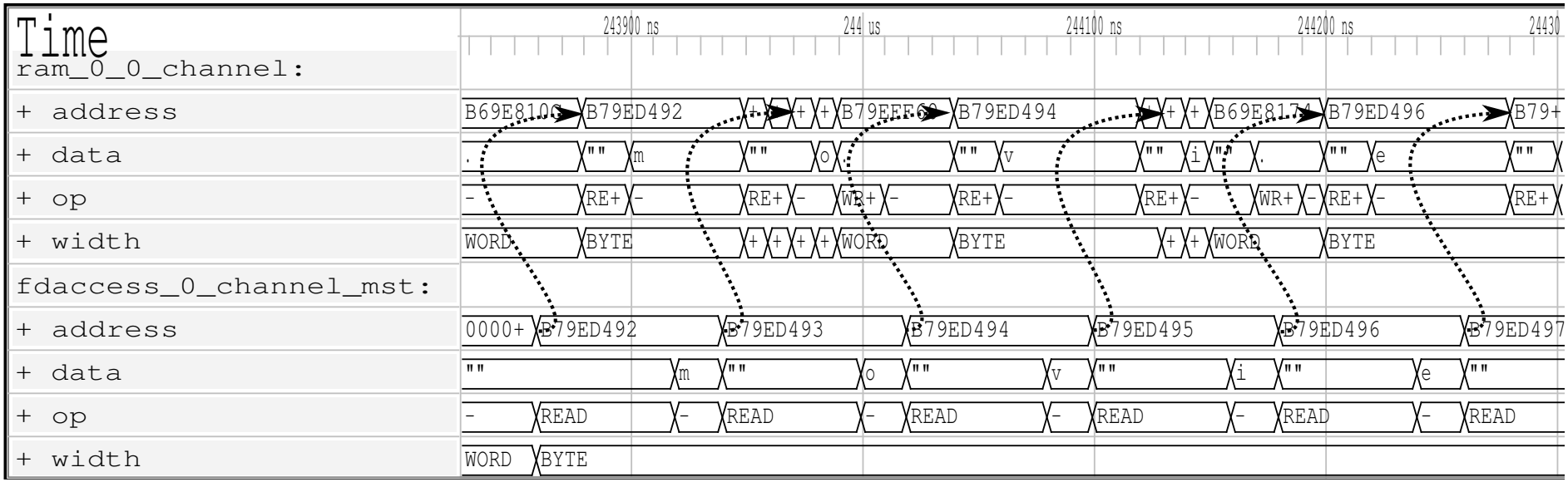
Section Name	Addresses	Port
.global_mem:	0x08181940 - 0x0819193f	5
fb_0:	0x08191948 - 0x081a3947	1
timer_0:	0x081d1650 - 0x081d165f	2
fdaccess_0:	0x081d4c28 - 0x081d4c4b	3
it_generator_0:	0x081d4cb0 - 0x081d4cb3	4
tty_pool_main:	0x081d56a8 - 0x081d56b7	5
.sysstack:	0xb58dd008 - 0xb59dd007	0
.uheap:	0xb59de008 - 0xb69de007	0
.kheap:	0xb69df008 - 0xb79df007	0
.text:	0xb79ea2f0 - 0xb79fe7e6	0
.rodata:	0xb79fe800 - 0xb79ffed3	0
.data:	0xb7a00fc0 - 0xb7a01dd7	0
.bss:	0xb7a01de0 - 0xb7a01fe3	0

Section Name	Addresses	Port
.global_mem	0x08181940 - 0x0819193f	2
timer_1	0x081d5380 - 0x081d538f	3
fdaccess_1	0x081d54a8 - 0x081d54cb	4
it_generator	0x081d5530 - 0x081d5533	5
tty_pool_main	0x081d54a8 - 0x081d56b7	1
fb_1	0x081f98c0 - 0x0820d8df	0
.sysstack:	0xb34e3008 - 0xb35e3007	0
.uheap:	0xb35e4008 - 0xb45e4007	0
.kheap:	0xb45e5008 - 0xb55e5007	0
.text:	0xb563a2f0 - 0xb564e7e6	0
.rodata:	0xb564e800 - 0xb564fed3	0
.data:	0xb5650fc0 - 0xb5651dd7	0
.bss:	0xb5651de0 - 0xb5651fe3	0

Capacités de validation

Validation du matériel

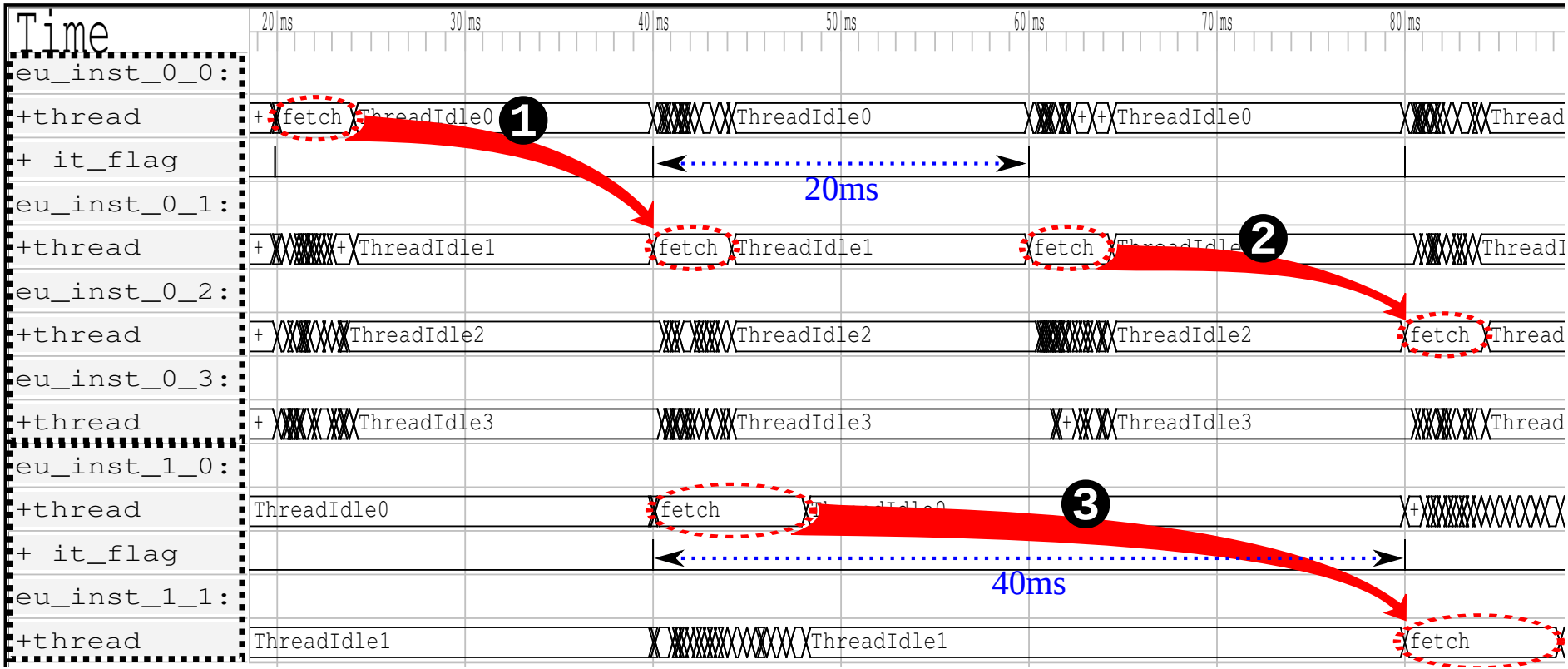
- Visualisation des chronogrammes
- Outils de débogage logiciel pour les composants matériels



Capacités de validation

Validation du logiciel

- Outils de débogage classiques (même pour du code instrumenté)
- Réutilisation des chronogrammes pour la validation du logiciel
  - ⇒ Suivi de migrations des tâches entre processeur



Précision de l'instrumentation

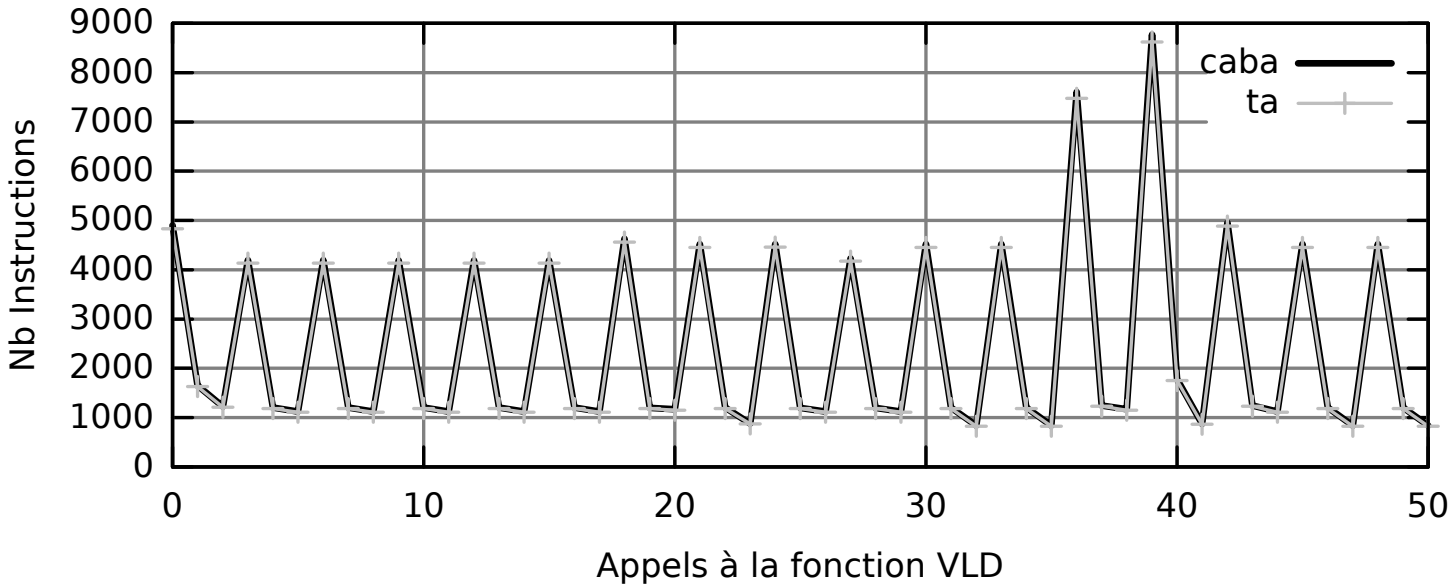
Comment mesurer cette précision

- Ne prendre en compte que la capacité à refléter le programme cible
  - $\Rightarrow$ Estimer le nombre d'instructions exécutées
- Besoin d'une fonction avec une dynamique d'exécution importante
  - $\Rightarrow$ La fonction de décodage VLD du format JPEG
    Très dépendante des données en entrée
- Ce n'est pas une estimation de performances

Précision de l'instrumentation

Nombre d'instructions exécutées à chaque appel à la fonction VLD

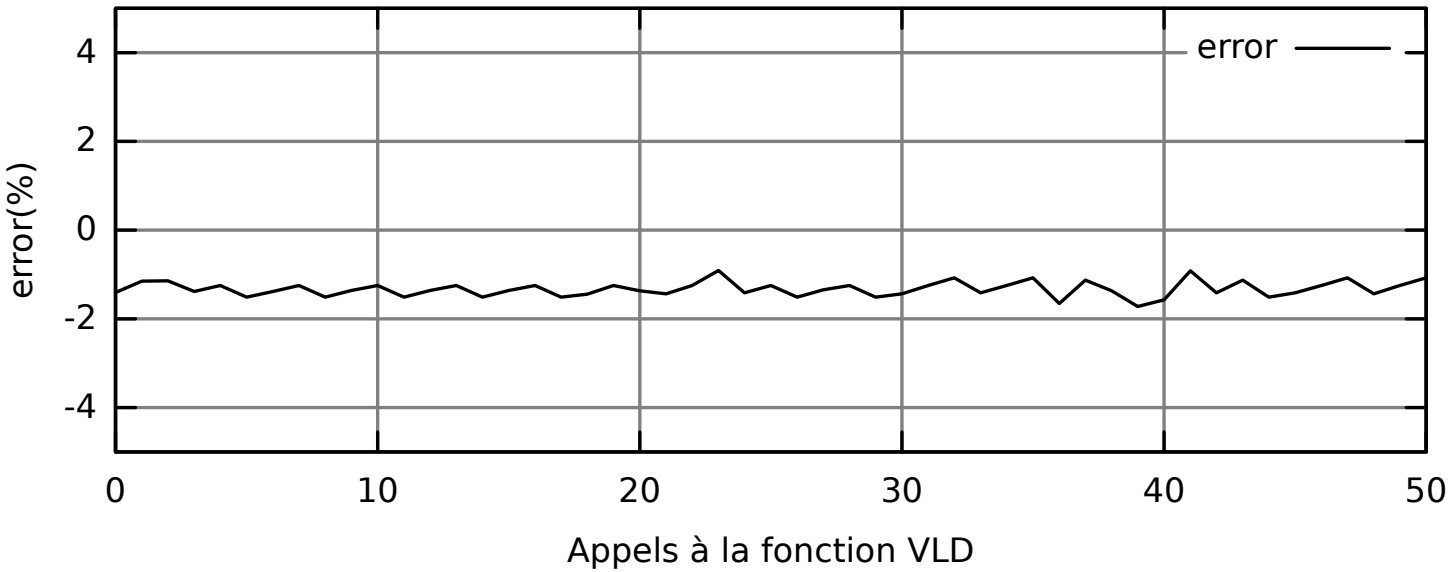
- Une plateforme CA fournit la mesure de référence.
- Erreur inférieure à 2%
  - Toujours négative (optimiste)
  - Correspond au code non instrumenté: HAL, builtins, ...



Précision de l'instrumentation

Nombre d'instructions exécutées à chaque appel à la fonction VLD

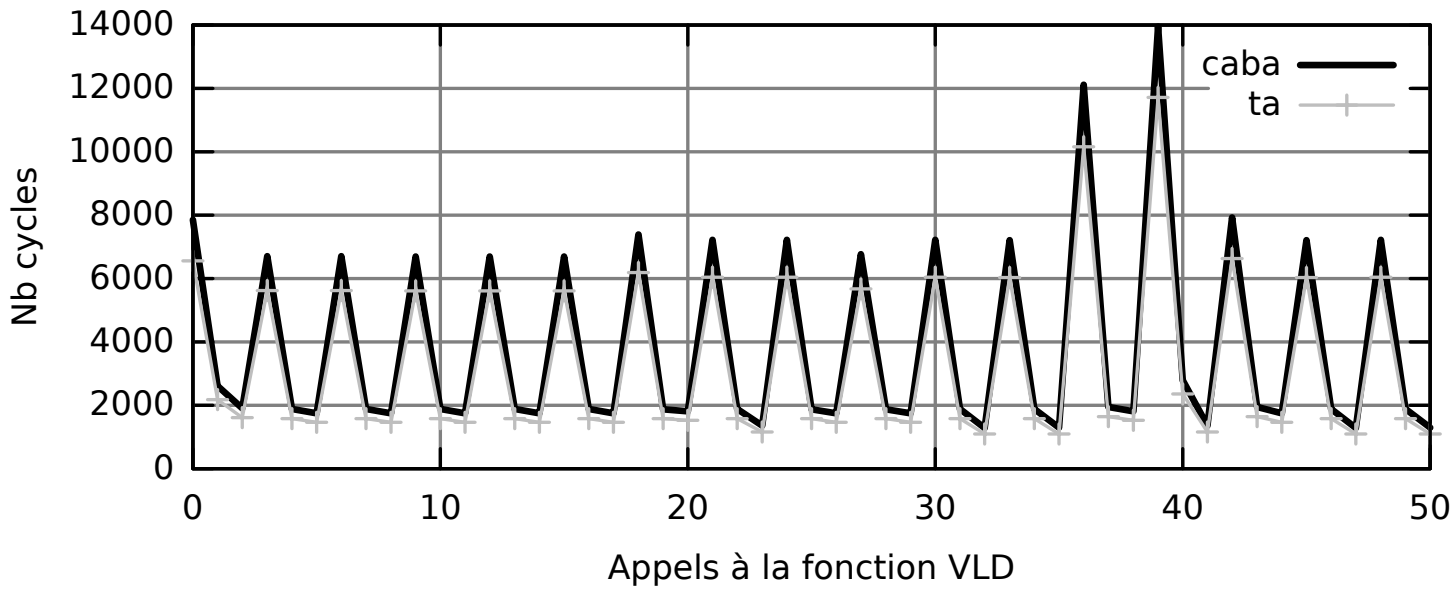
- Une plateforme CA fournit la mesure de référence.
- Erreur inférieure à 2%
  - Toujours négative (optimiste)
  - Correspond au code non instrumenté: HAL, builtins, ...



Estimation des performances

Précision d'analyse de blocs de base

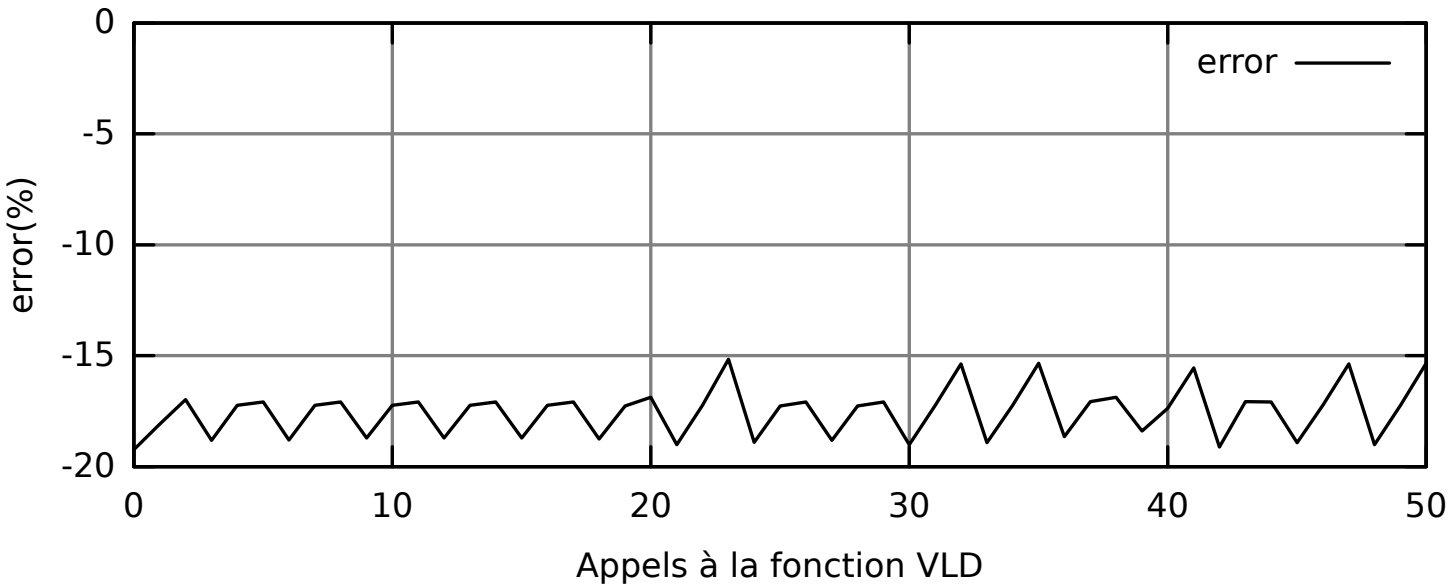
- Pénalités de branchement non instrumentées
  - $\Rightarrow$ 15% à 20% d'erreur
- Précision de l'analyse des instructions
  - $\Rightarrow$ $\approx$ 2% d'erreur



Estimation des performances

Précision d'analyse de blocs de base

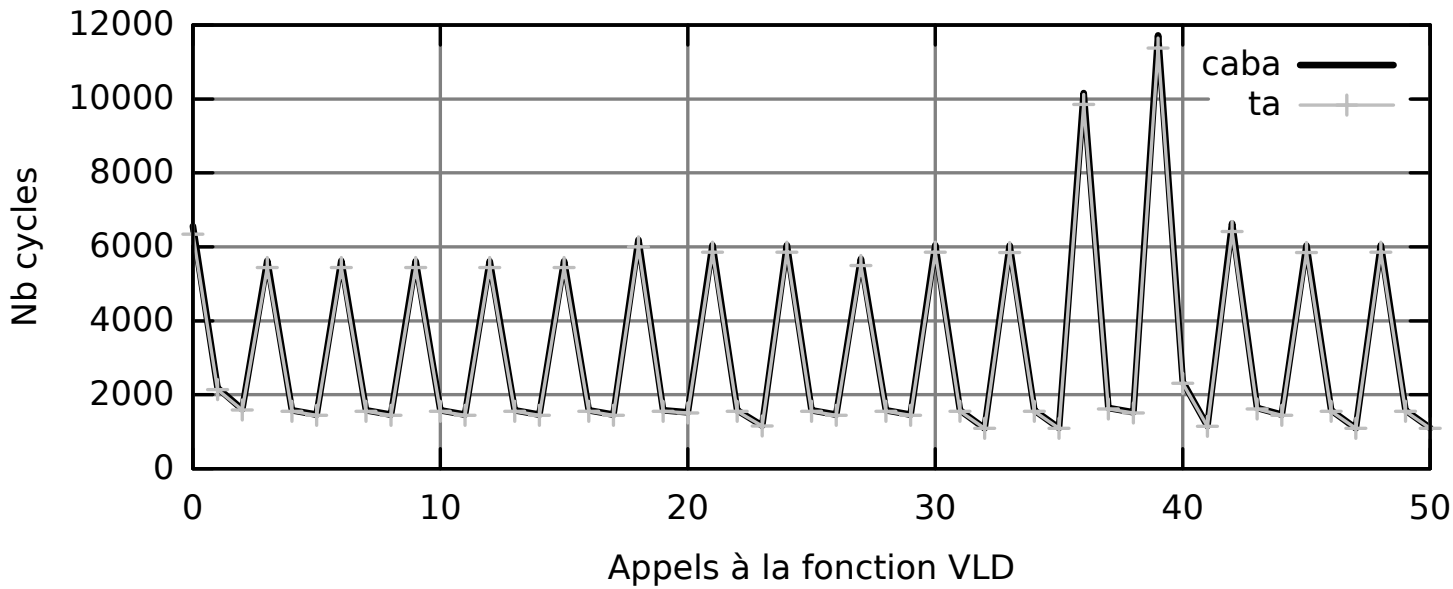
- Pénalités de branchement non instrumentées
  - $\Rightarrow$ 15% à 20% d'erreur
- Précision de l'analyse des instructions
  - $\Rightarrow$ $\approx$ 2% d'erreur



Estimation des performances

Précision d'analyse de blocs de base

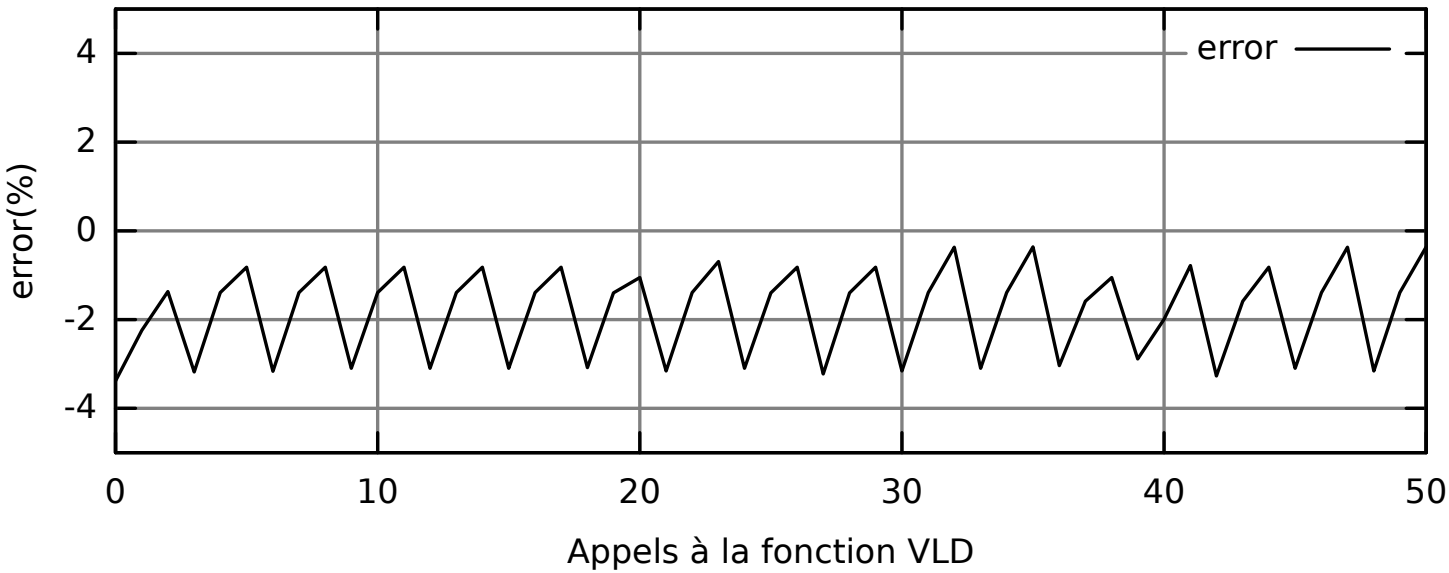
- Pénalités de branchement non instrumentées
  - $\Rightarrow$ 15% à 20% d'erreur
- Précision de l'analyse des instructions
  - $\Rightarrow$ $\approx$ 2% d'erreur



Estimation des performances

Précision d'analyse de blocs de base

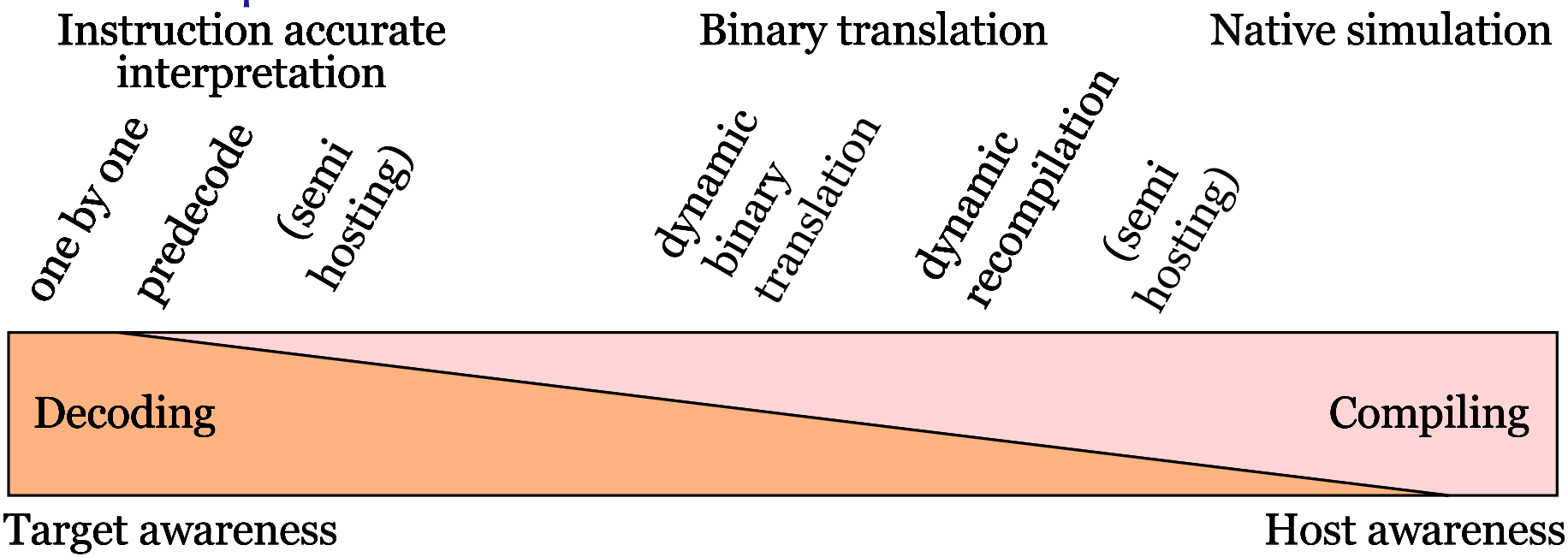
- Pénalités de branchement non instrumentées
  - $\Rightarrow$ 15% à 20% d'erreur
- Précision de l'analyse des instructions
  - $\Rightarrow$ $\approx$ 2% d'erreur



Outline

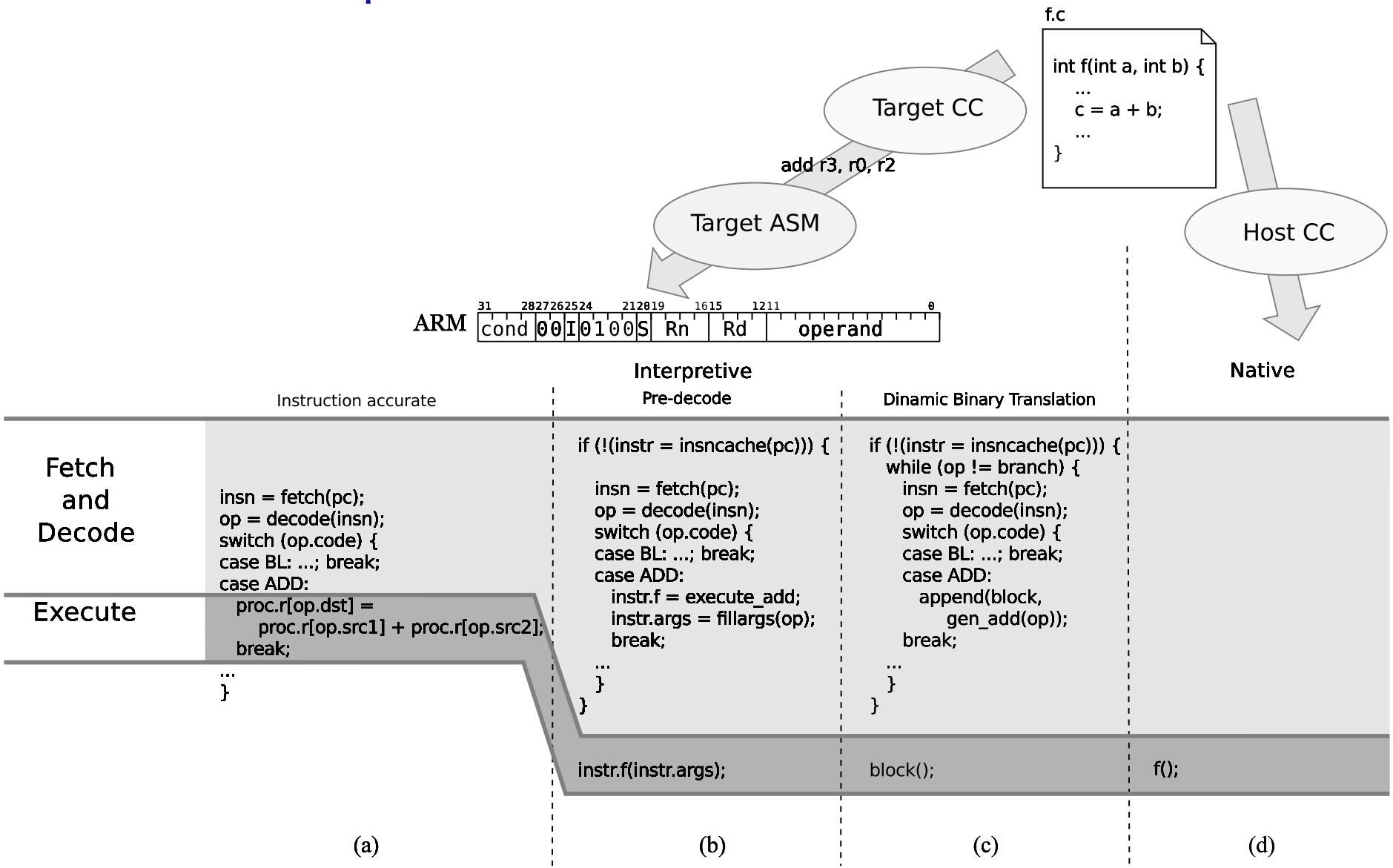
Résumé et conclusion

Simulation points



Résumé et conclusion

Simulation techniques



Résumé et conclusion

Table: Rough characteristics of the software simulation approaches

	sim. speed	simulation accuracy		development time		full soft. exec.
		ins. count	time	first	reuse	
IA	−−	++	++	+	+	yes
Predecode	−	++	++	−	+	yes
BT	+	−−	−−	−−	−	yes
Native	++	−−	−−	−−	++	no

Quelques ressources

Logiciel libres

- SoCLib : `www.soclib.fr`, cycle accurate, TLM-DT model, MutekH and DNA/OS OSes
- Rabbits : `tima-sls.imag.fr/www/research/rabbits`, DBT based simulation, Linux port, DNA/OS port

Papier

Frédéric Pétrot, Nicolas Fournel, Patrice Gerin, Marius Gligor, Mian Muhammed Hamayun, Hao Shen: On MPSoC Software Execution at the Transaction Level. IEEE Design & Test of Computers 28(3): 32-43 (2011)

Conférences

- DATE, DAC, ASP-DAC, CODES+ISSS
- ISCA, MICRO
- SIGMETRICS, SIGARCH