Preventing timing information leakages from the microarchitecture

By **Mathieu Escouteloup** (Inria)
Advisors: **Ronan Lashermes** (Inria)
Christophe Bidan (CentraleSupélec)
Jacques Fournier (CEA-Leti)

March 30, 2020

Table of contents

1. Shared resources
2. ISA contextualization
3. Timing evaluation
4. Conclusion

Table of contents

1 Shared resources

2 ISA contextualization

3 Timing evaluation

4 Conclusion

Microarchitectural sharing

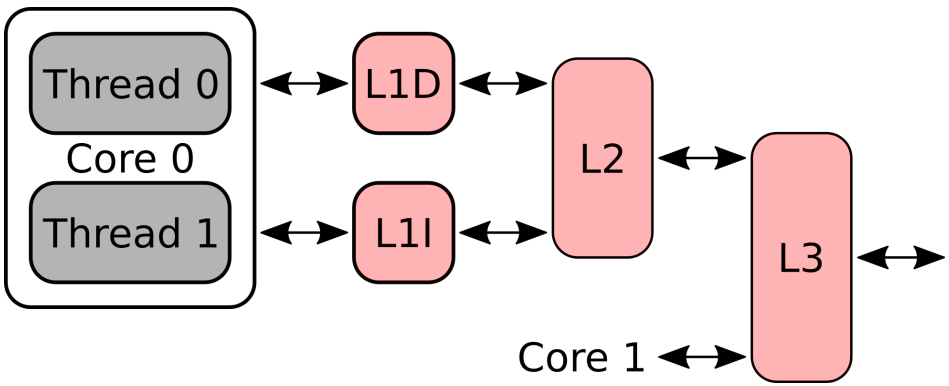
Definition

- Multiple entities can request the same resource.
- Entities : hardware threads, cores, processes ...
- Resources : cache memories, prediction tables, FSMs, buffers ...

Two kinds of sharing

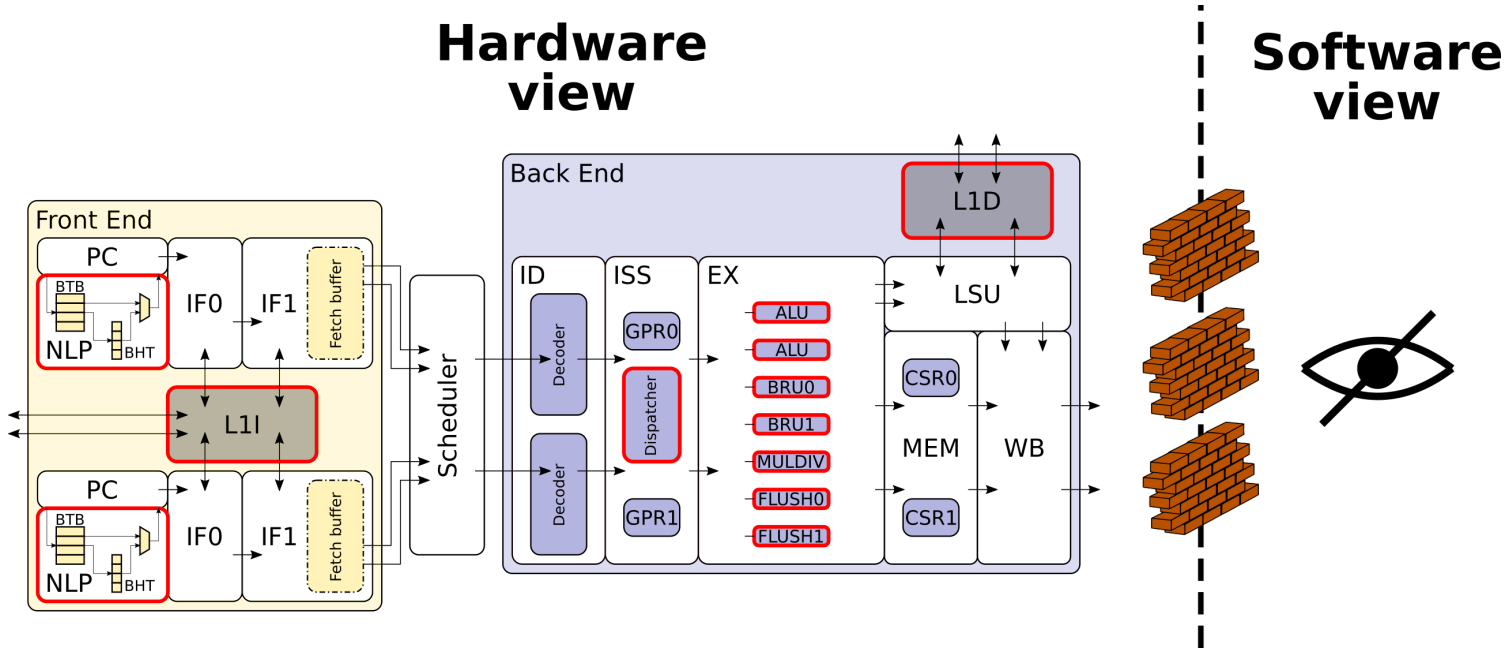
- Temporal : use the same resource but at different points in time.
- Spatial : use the same resource at the same time.
- Both can be combined.

The cache memory example



Data but also timing informations are shared between the entities.

An implementation issue ...



- Targeted shared resources are in the microarchitecture.
- Leakages depend on the implementation.
- Microarchitecture cannot be controlled by the software.

... but not only !

Table of contents

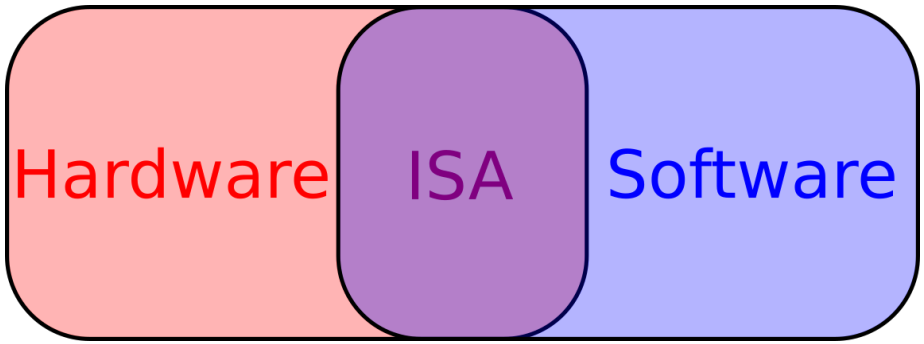
1 Shared resources

2 ISA contextualization

3 Timing evaluation

4 Conclusion

A global issue



- **Which part knows the application logic ?**
- **Which part can efficiently make the isolation ?**
- **How can they exchange information ?**

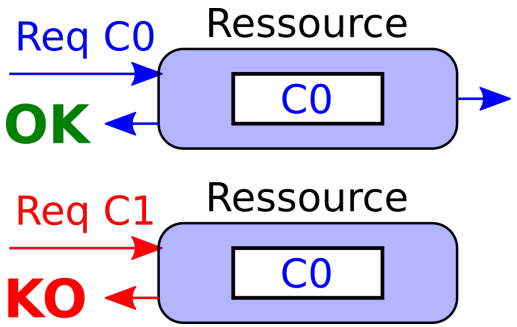
The whole system is concerned !

How to modify the ISA ?

Constraints :

1. Consider the whole isolation issue : temporal **and** spatial sharing.
2. Create custom security domains.
3. Scalability to multiple systems.
4. Preserve the architecture abstraction

Contextualization : associate a domain to each data and resource.



Our security domain model.

New dedicated register :

- identifier : an unique number for each security domain.

New instruction : `CONTEXT.SWITCH`.

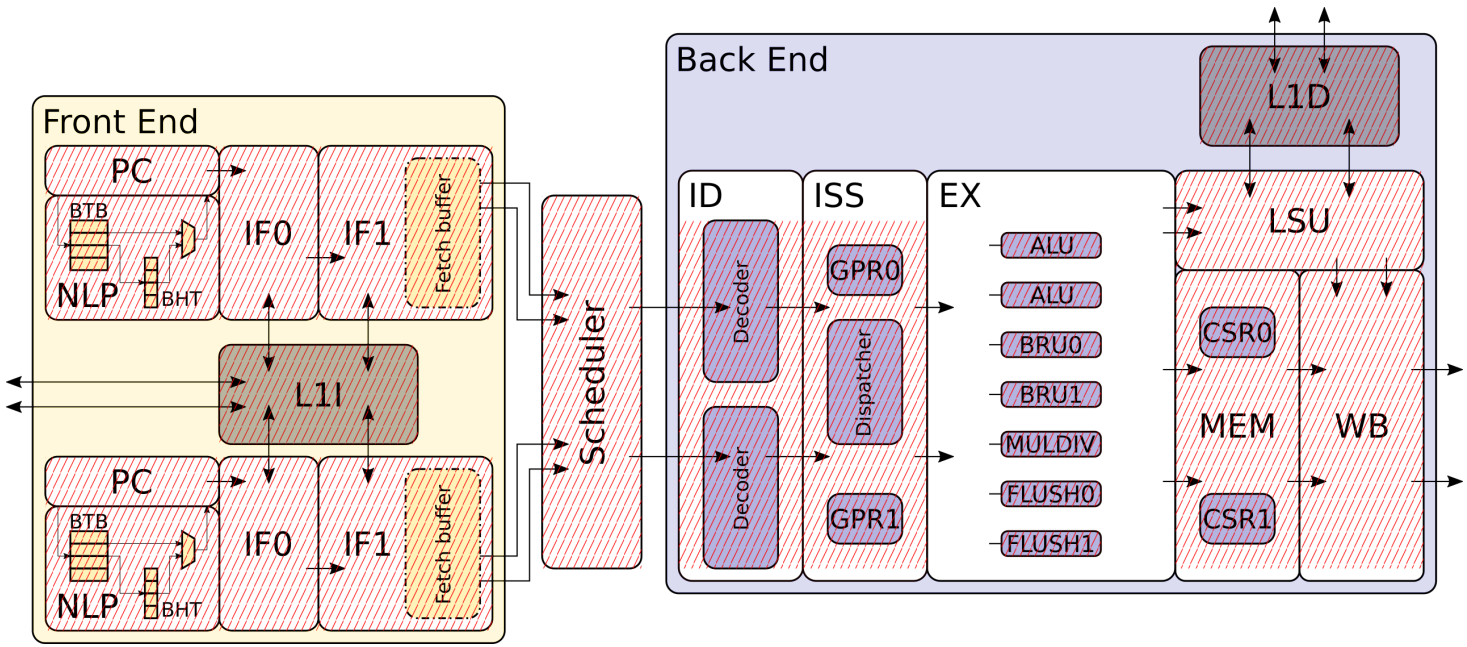
- Indicates a domain change.
- Some actions must be done :
  1. **Flush** traces from the old domain.
  2. **Split** resources if needed.
  3. **Lock** resources if needed.
- Successful → a new domain can be safely executed.

Software view.

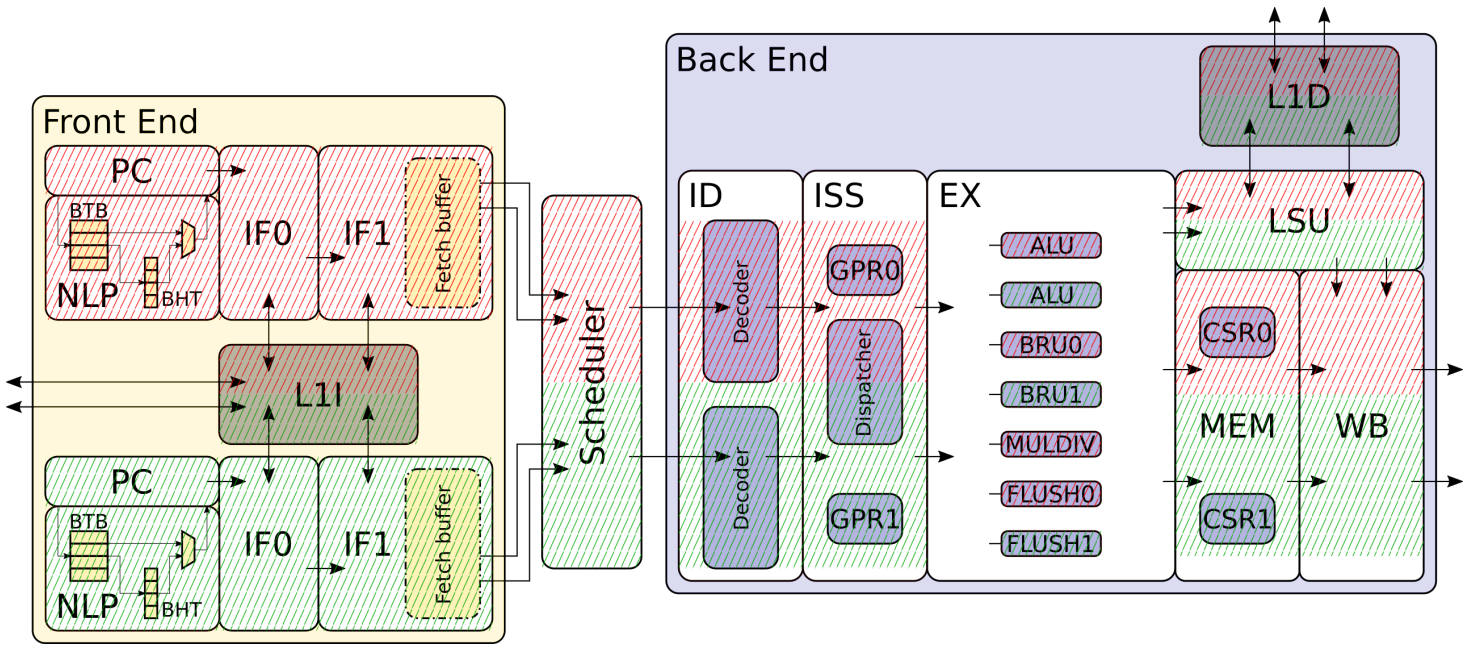
```
1.  # OLD CONTEXT
2.  old-app:
3.    ...
4.    ...
10. switch-code:
11.   csrw nextid,a0    # config
12.   switch a0         # switch


13. # NEW CONTEXT
14. new-app:
15.   ...
16.   ...
```

Hardware view : before switch.



Hardware view : after switch.



Successfully implemented in two cores, one with SMT.

Table of contents

1 Shared resources

2 ISA contextualization

3 Timing evaluation

4 Conclusion

An implementation agnostic benchmark

Goal :

to quantitatively evaluate information leakages in the microarchitecture.

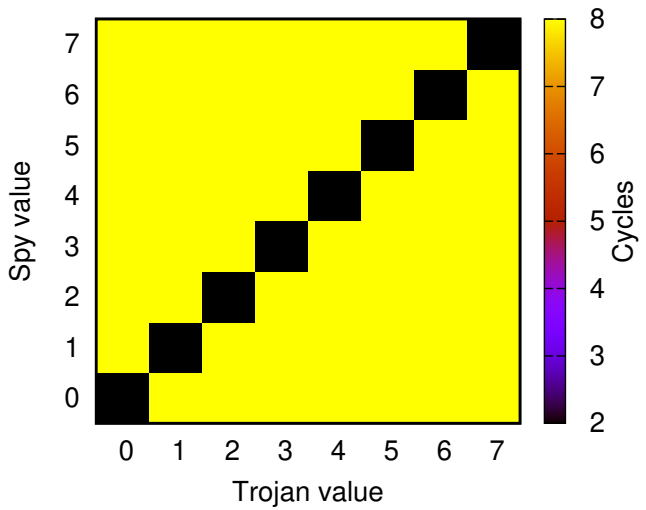
Constraints :

- focus on timing information leakages in the design,
- consider common shared resources,
- focus on vulnerability, not exploitability.

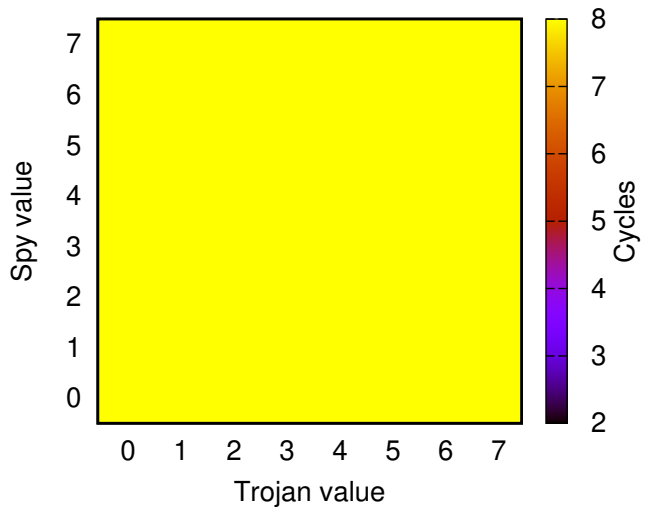
Scenario :

- a trojan encodes a value in a shared resource state,
- a spy tries to recover the value.

The cache example : temporal sharing

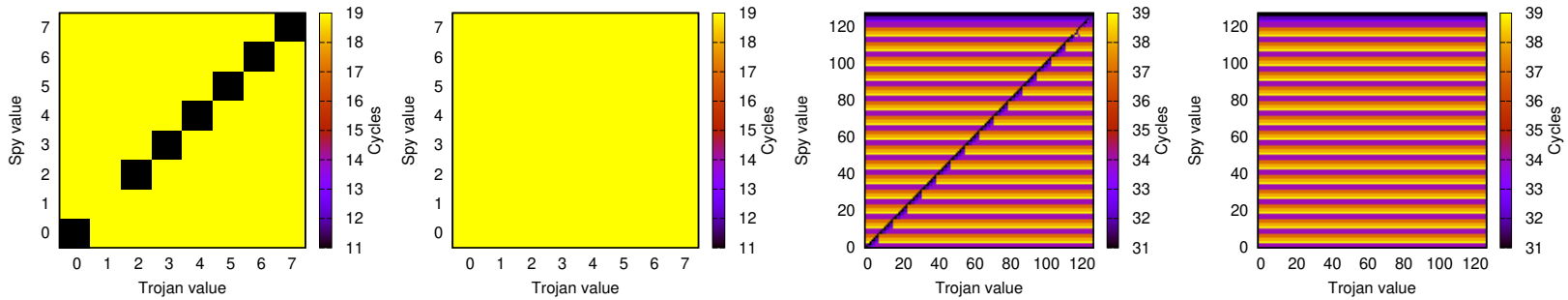


(a) Unprotected L1D



(b) Protected L1D

Other benchmarks



(a) **L1I** (b) **L1I** (c) **BHT** (d) **BHT**

(e) **BTB** (f) **BTB** (g) **Cross L1D** (h) **Cross L1D**

More under development ...

Table of contents

1 Shared resources

2 ISA contextualization

3 Timing evaluation

4 Conclusion

Conclusion

- Shared resources are sources of vulnerability.
- The ISA must be modified to give security information to the hardware.
- Software indicates its constraints, hardware applies them.
- A new security benchmark to evaluate the implementations.

Timesecbench : https ://gitlab.inria.fr/rlasherm/timesecbench