A Hardware Approach for Detecting, Tolerating and Exposing High Level Atomicity Violations.

Lois Orosa
João Lourenço

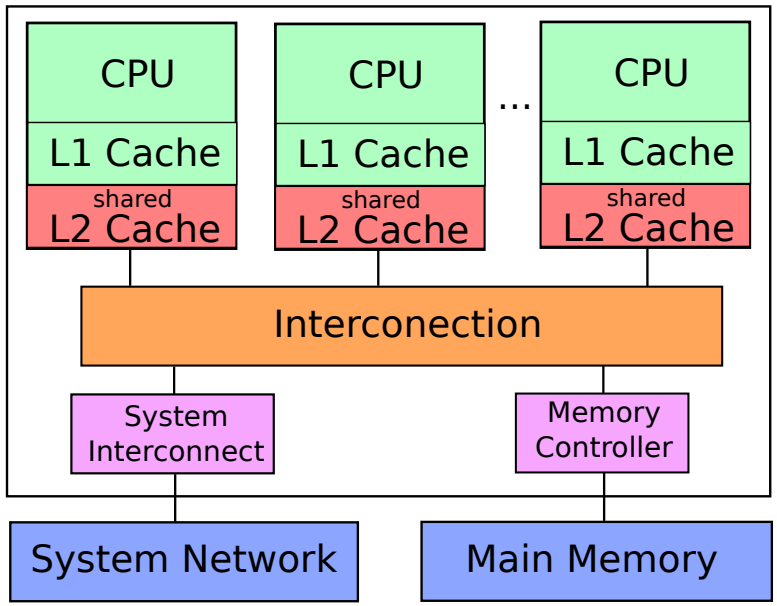
Euro-TM Short Term Scientific Mission
Departamento de informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa

DMTM - January 22, 2014

Contents

1. Introduction
2. Idea
   - Detecting HLAVs
   - Exposing HLAVs
   - Tolerating HLAVs
3. Conclusions

Introduction - Parallel Programming



- Multicore processors are mainstream (Intel, AMD, IBM, etc.)
- Shared memory model (more intuitive than message passing)
- Difficult to program and debug
- Synchronization mechanisms can lead to concurrent bugs

Introduction - Concurrent Bugs

- Concurrent bugs are difficult to detect and reproduce
- It takes long time to debug them
- Several types:
  - Dealocks, Livelocks ...
  - Data Races
  - Order violations
  - Asymmetric (or low level) data races
  - High Level Atomicity Violations
- Other Tools (Deterministic replay, tools for code analisys, etc.)
- Hardware and Software approaches

Introduction - High Level Atomicity Violations (HLAV)

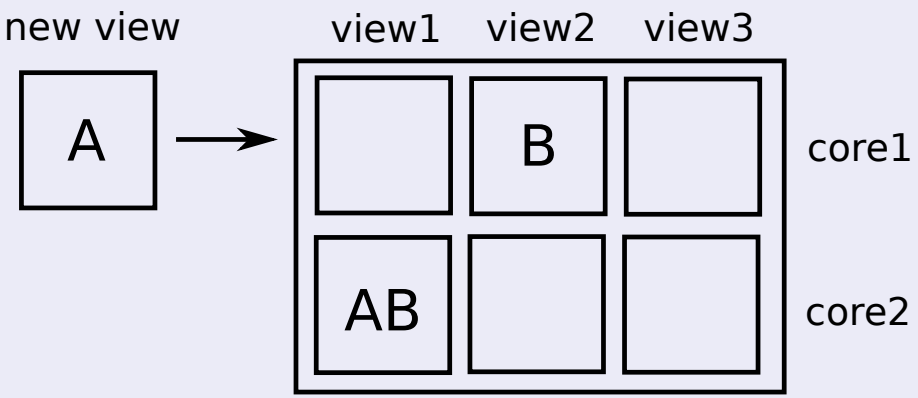
- All shared variables are protected by synchronization mechanisms (no data races).
- If one thread accesses data together in the atomic section and another thread accesses the same data separately in different atomic sections, a HLAV is reported

```
1  atomic void getA() {
2     return pair.a;
3  }
4  atomic void getB() {
5     return pair.b;
6  }
7  atomic void setPair (int a, int b) {
8     pair.a=a;
9     pair.b=b;
10 }
11 boolean areEqual(){
12    int a = getA();
13    int b = getB();
14    return a==b;
15 }
```

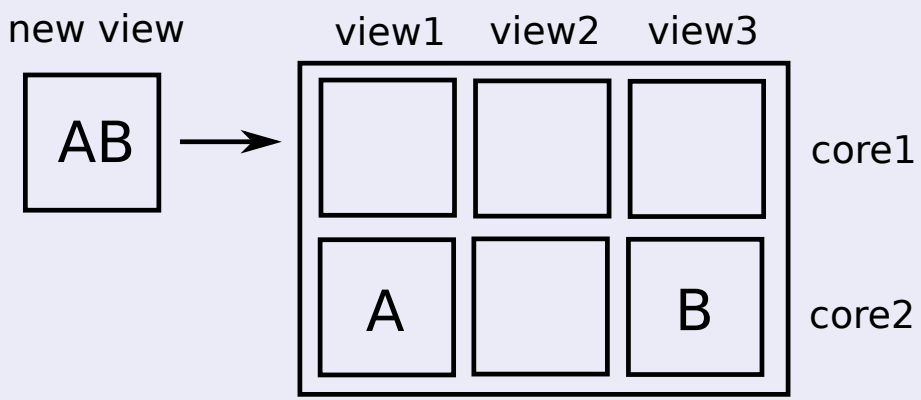
Introduction - High Level Atomicity Violations (II)

- View: addresses of memory locations accessed in an atomic region
- Maximal view: is not a subset of any views of that thread/core

real HLAV



potential HLAV



HLAV: definition

There is an HLAV if the intersections of two views of one thread with the maximal view of another thread do not form an inclusion chain.

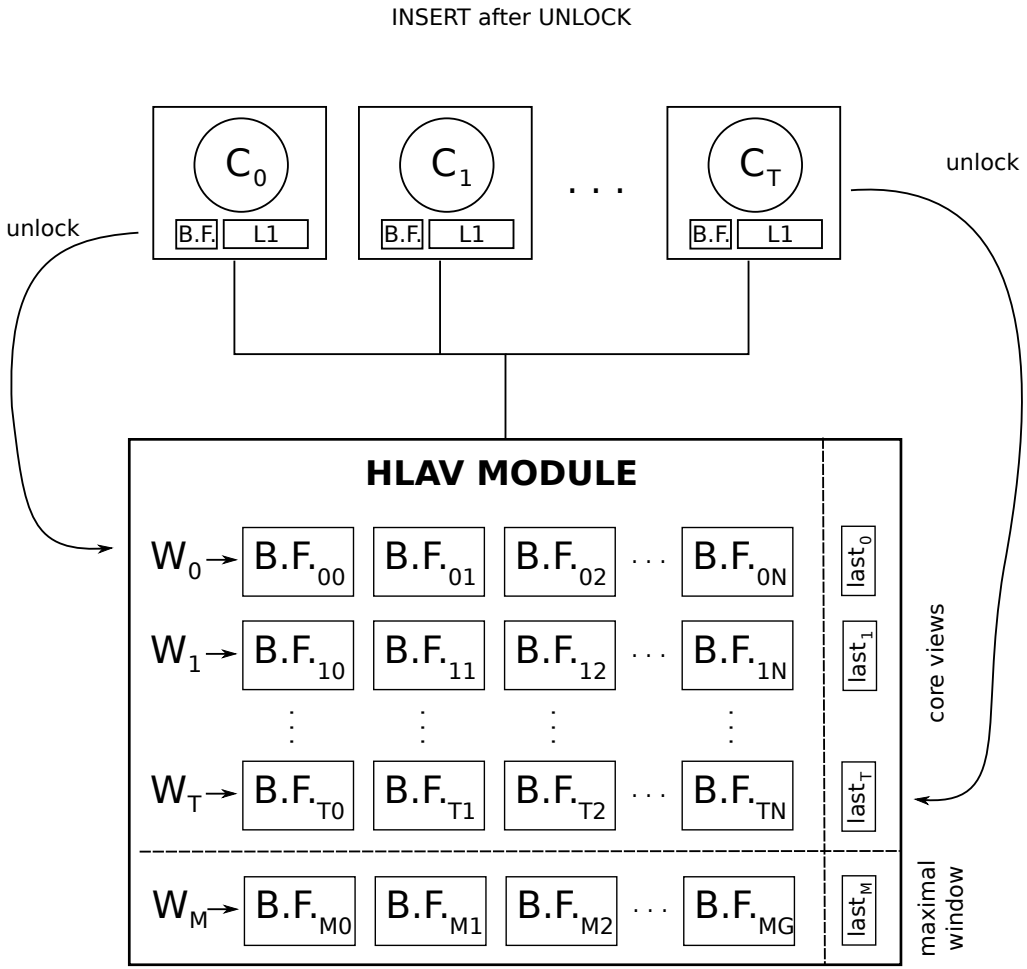
Contents

1. Introduction

2. Idea
   - Detecting HLAVs
   - Exposing HLAVs
   - Tolerating HLAVs

3. Conclusions

Idea - Module of Bloom filters



- Windows of *N* views per core
- The new view replaces the older view in the window
- Maximal window to maintain maximal views from all cores

Idea - Description

In each core

- Collection of the addresses accessed in the atomic sections.
- Additional Bloom filter per core.
- At unlocks, the core sends the content of the Bloom filter to the module.

Module

- Receives the views of all cores.
- Core windows keep the last views of the cores
- Maximal window keeps the maximal views of all cores
  - Also maintains additional information from cores that can have a HLAV with this maximal view

Contents

1. Introduction

2. Idea
   - Detecting HLAVs
   - Exposing HLAVs
   - Tolerating HLAVs

3. Conclusions

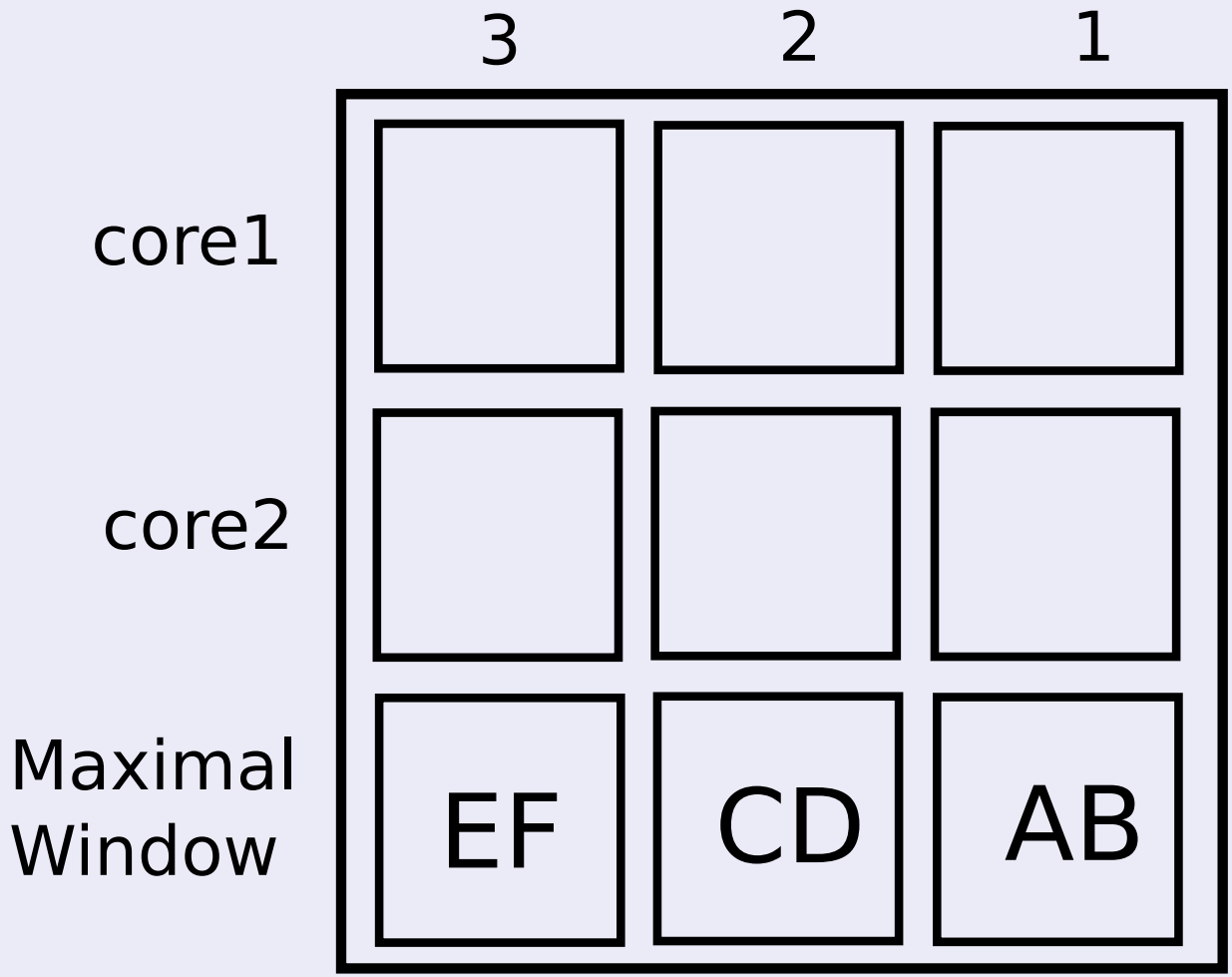
Idea - Detecting HLAVs (I)

New View

1. If a view is maximal, is inserted in the maximal window
2. If the intersection of the new view with a MV is subset of that MV, the new view is set as possible causative of a HLAV
3. If (2) and if there is another view in the same window that the new view that also matches (2), and this 2 views do not form a chain, HLAV is detected

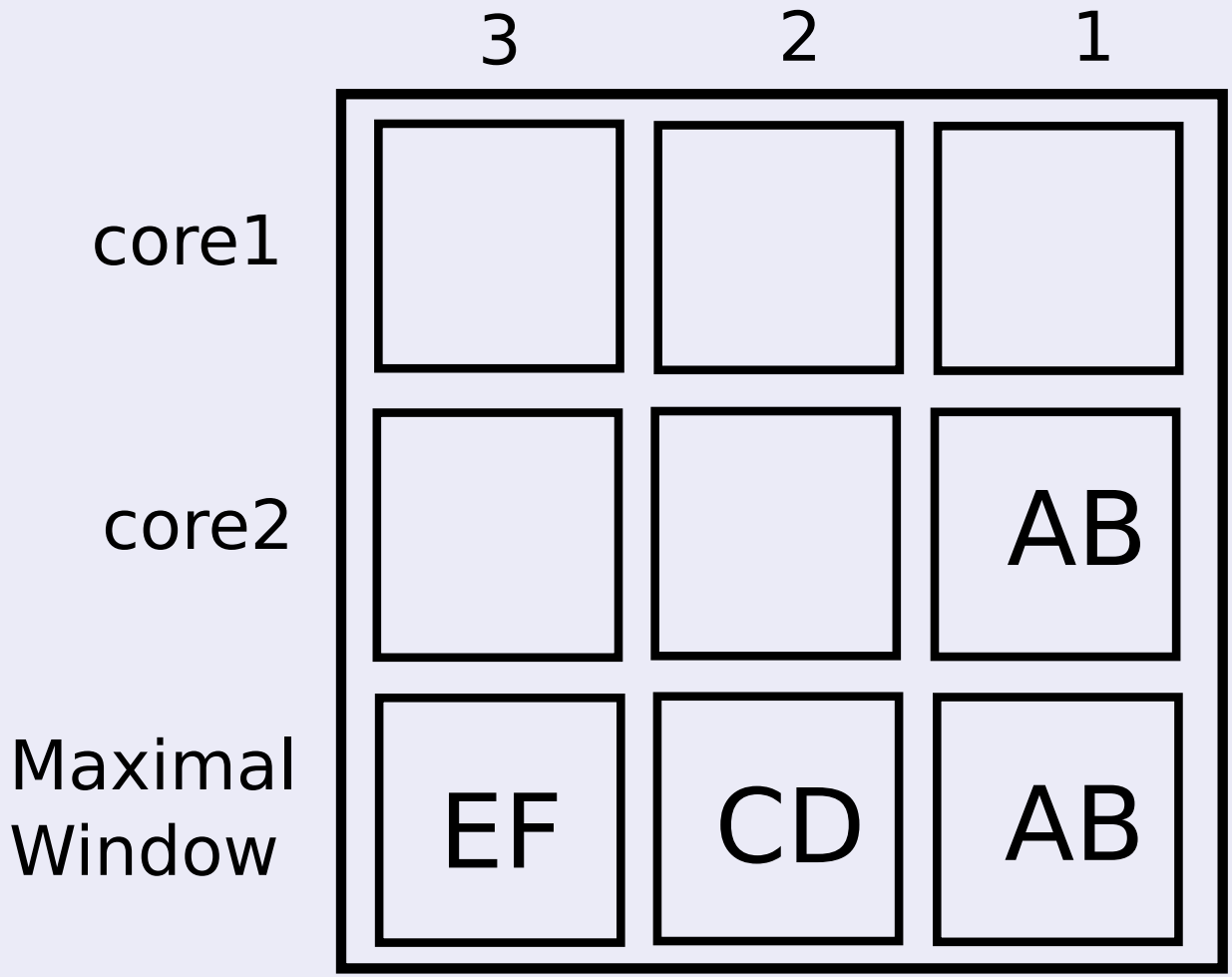
Idea - Detecting HLAVs (II)

Example



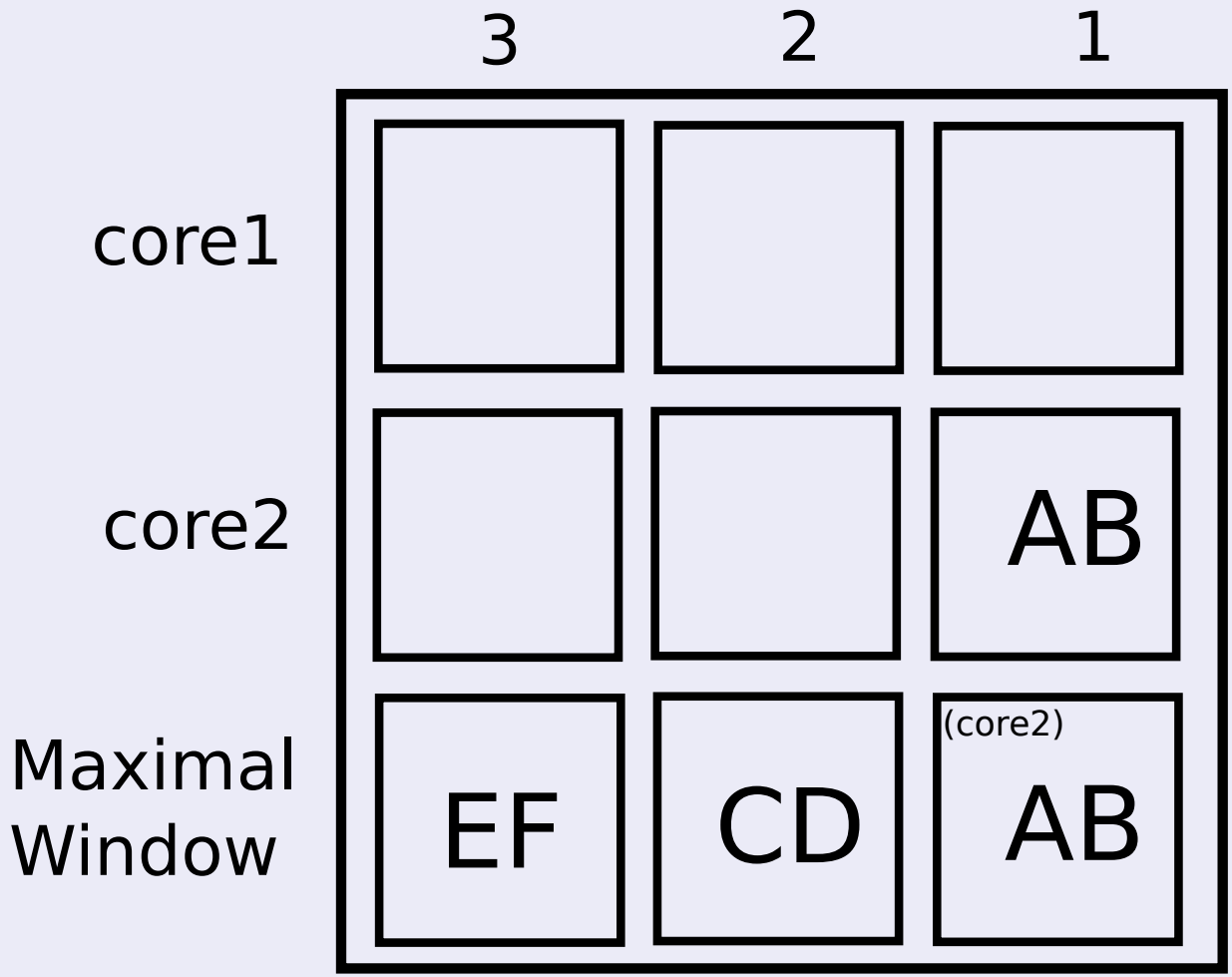
Idea - Detecting HLAVs (II)

Example



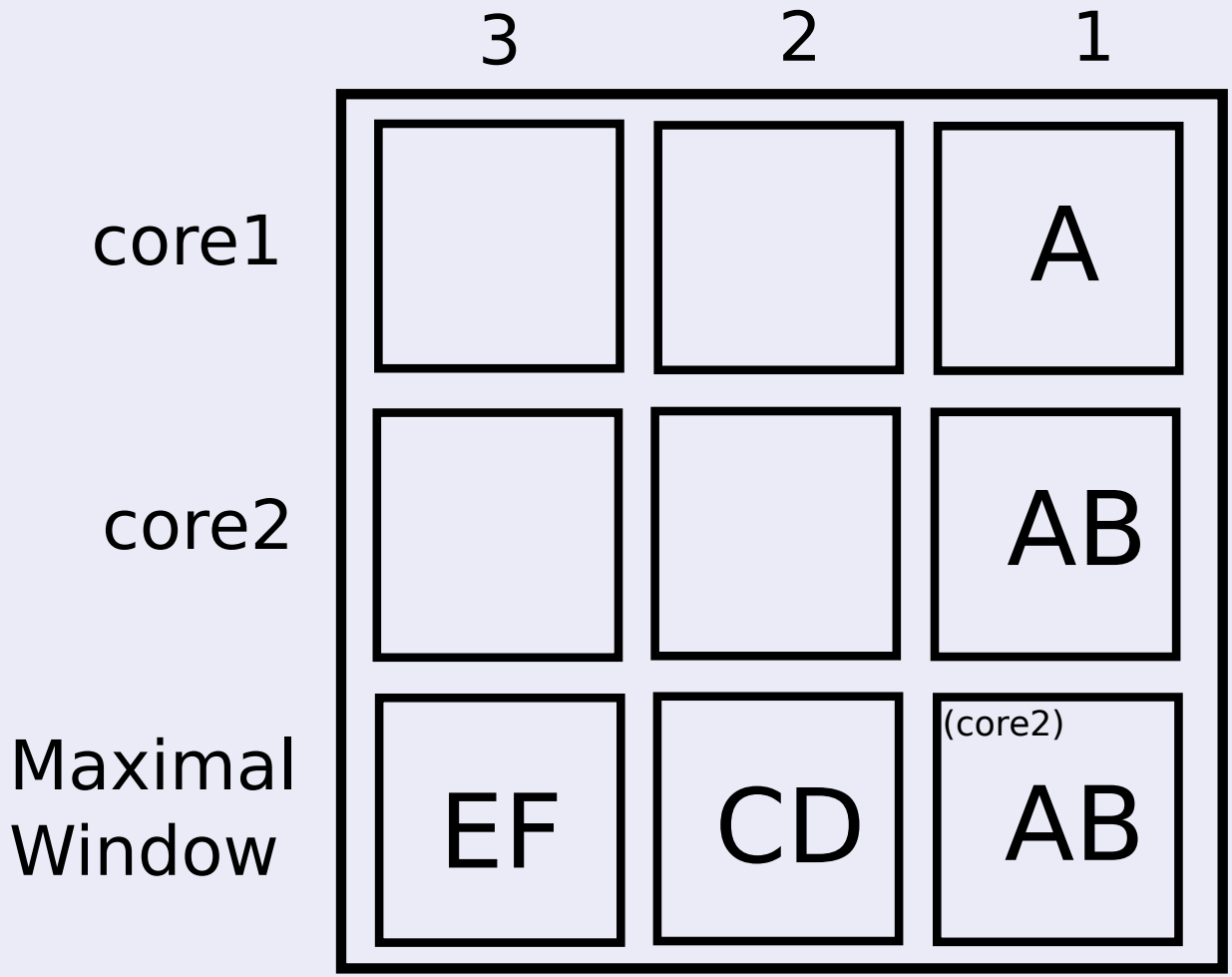
Idea - Detecting HLAVs (II)

Example



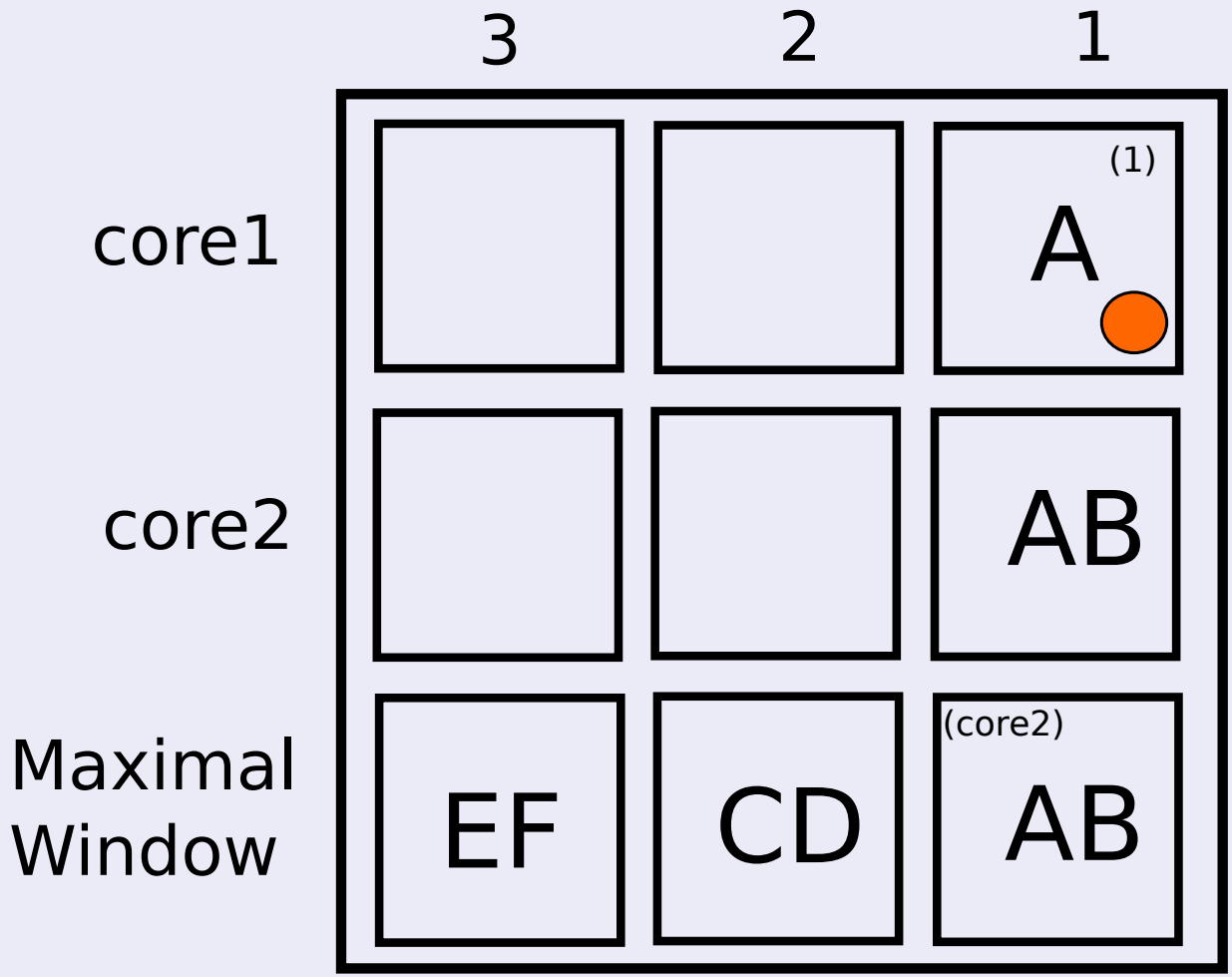
Idea - Detecting HLAVs (II)

Example



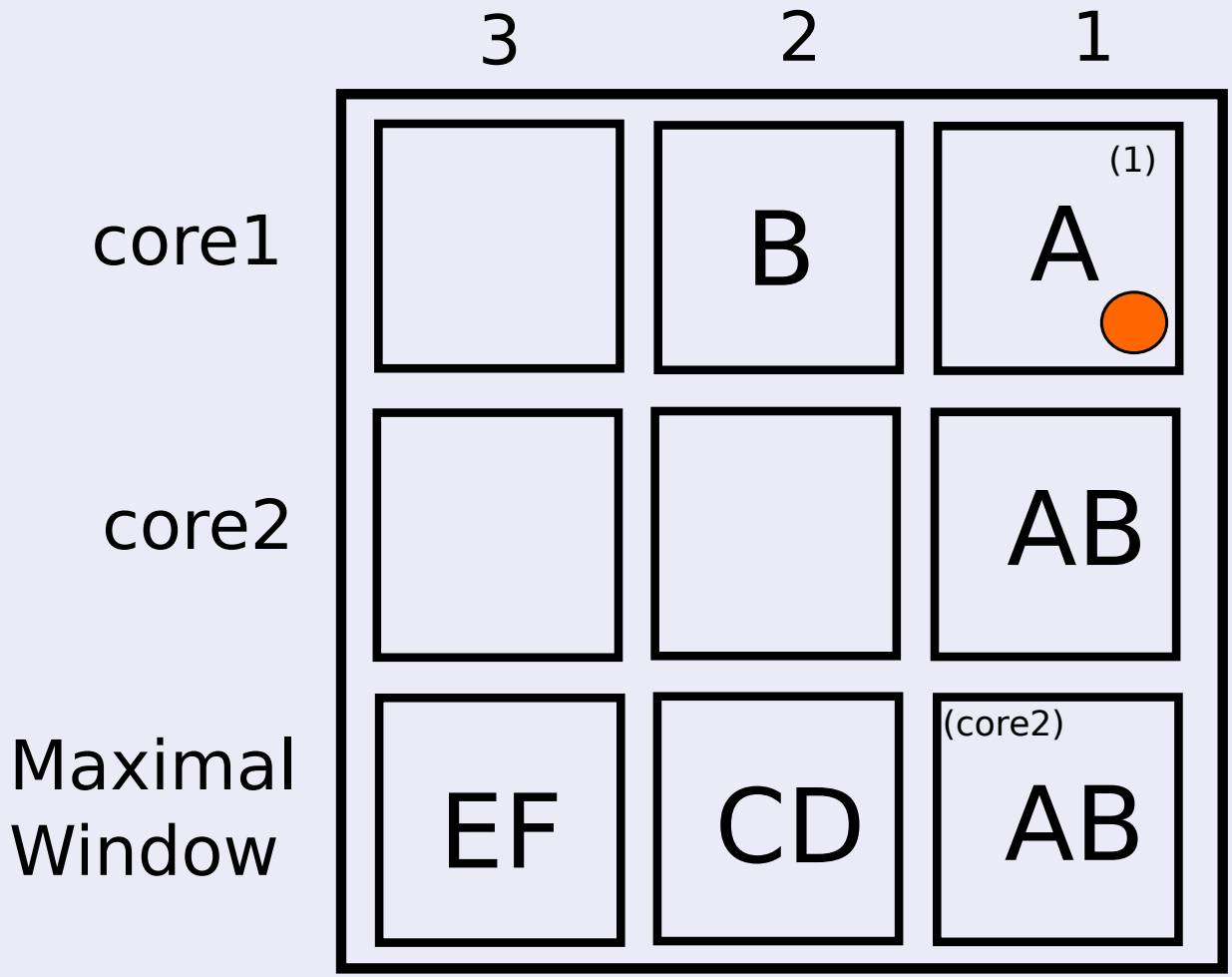
Idea - Detecting HLAVs (II)

Example



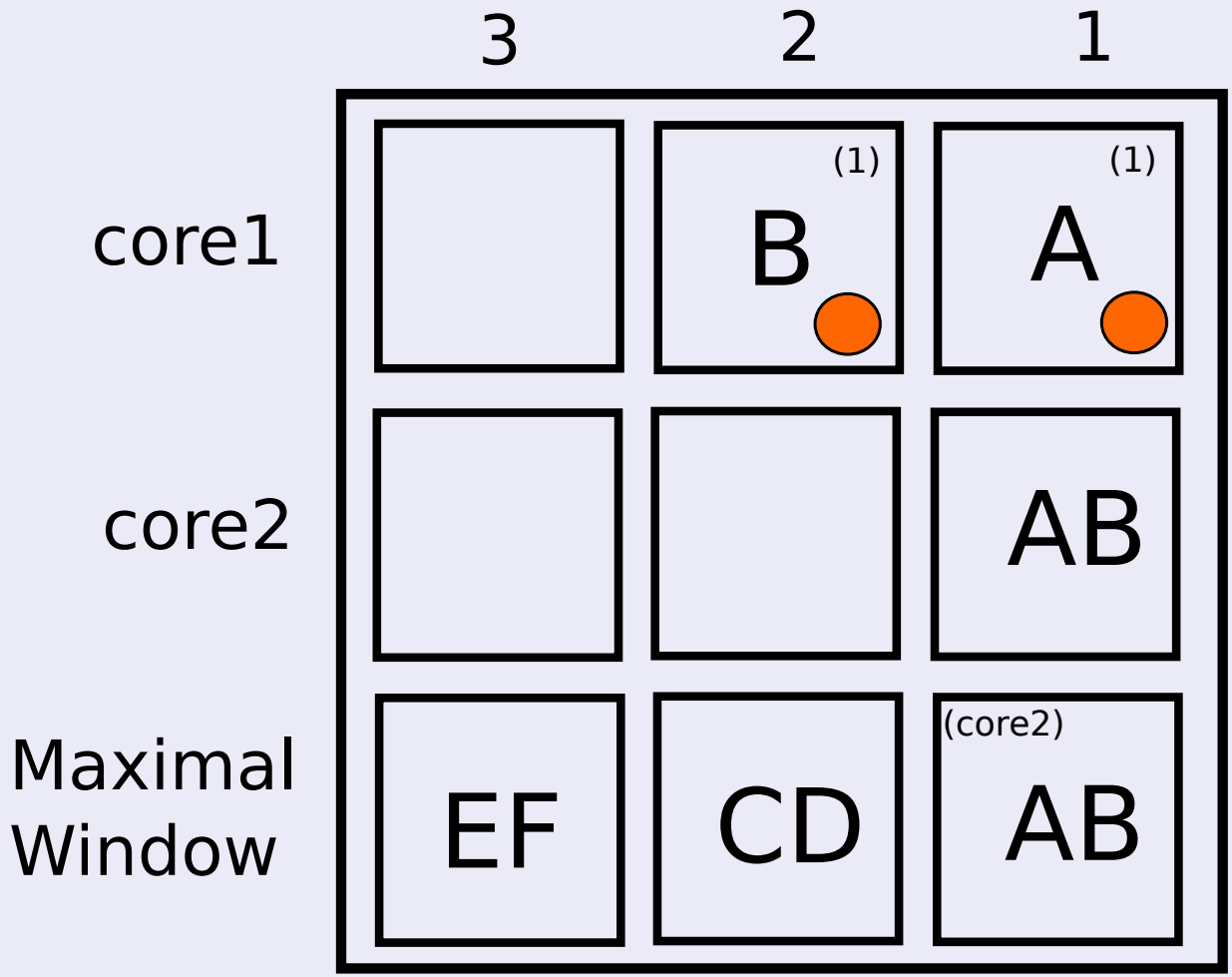
Idea - Detecting HLAVs (II)

Example



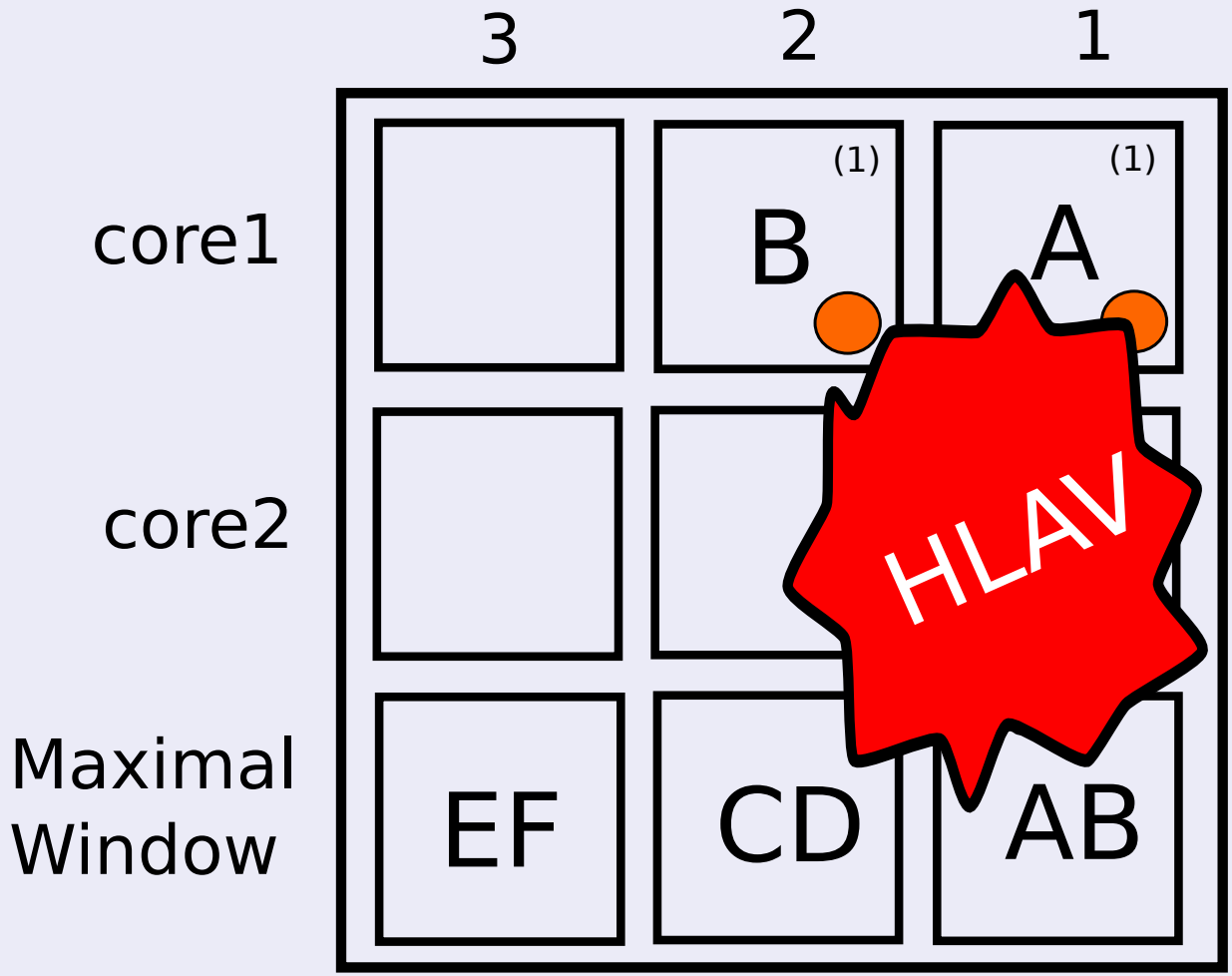
Idea - Detecting HLAVs (II)

Example



Idea - Detecting HLAVs (II)

Example



Contents

1. Introduction

2. Idea
   - Detecting HLAVs
   - Exposing HLAVs
   - Tolerating HLAVs

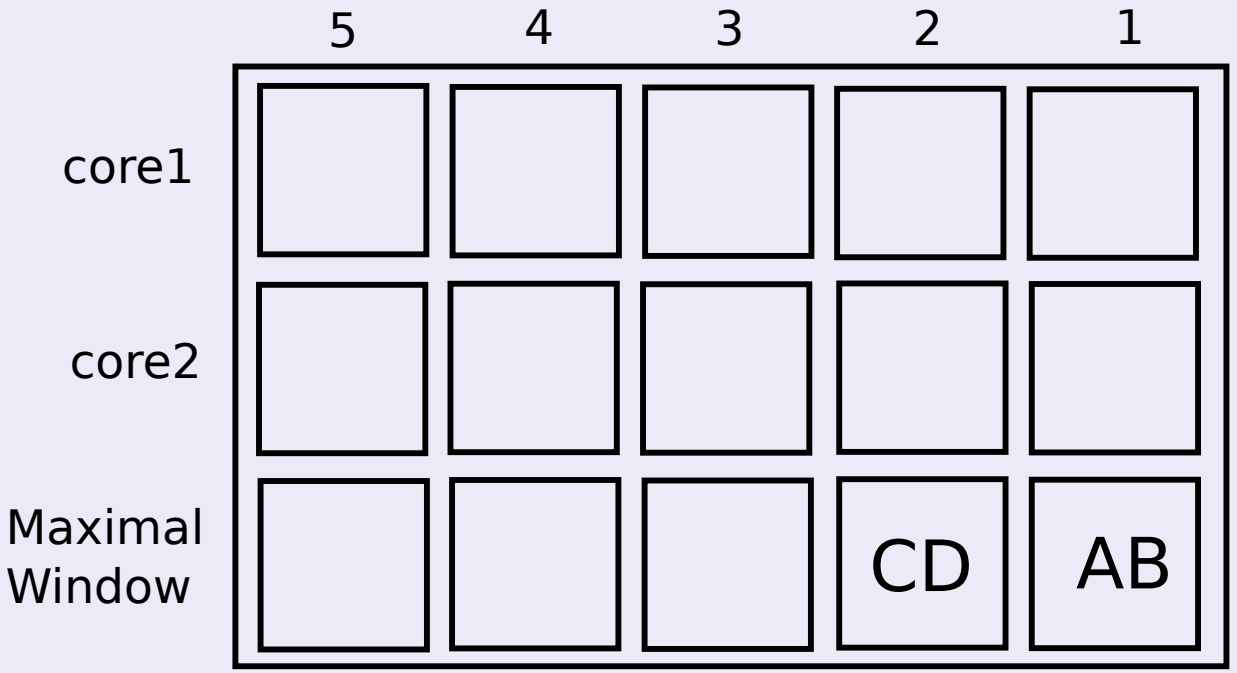
3. Conclusions

Idea - Exposing HLAVs (I)

- Tries to become the potential HLAVs into real HLAVs
- Forces the interleving by stalling cores
- Timeout to avoid deadlocks
- Minimal hardware additions to support this feature
- Practical in the debugging process

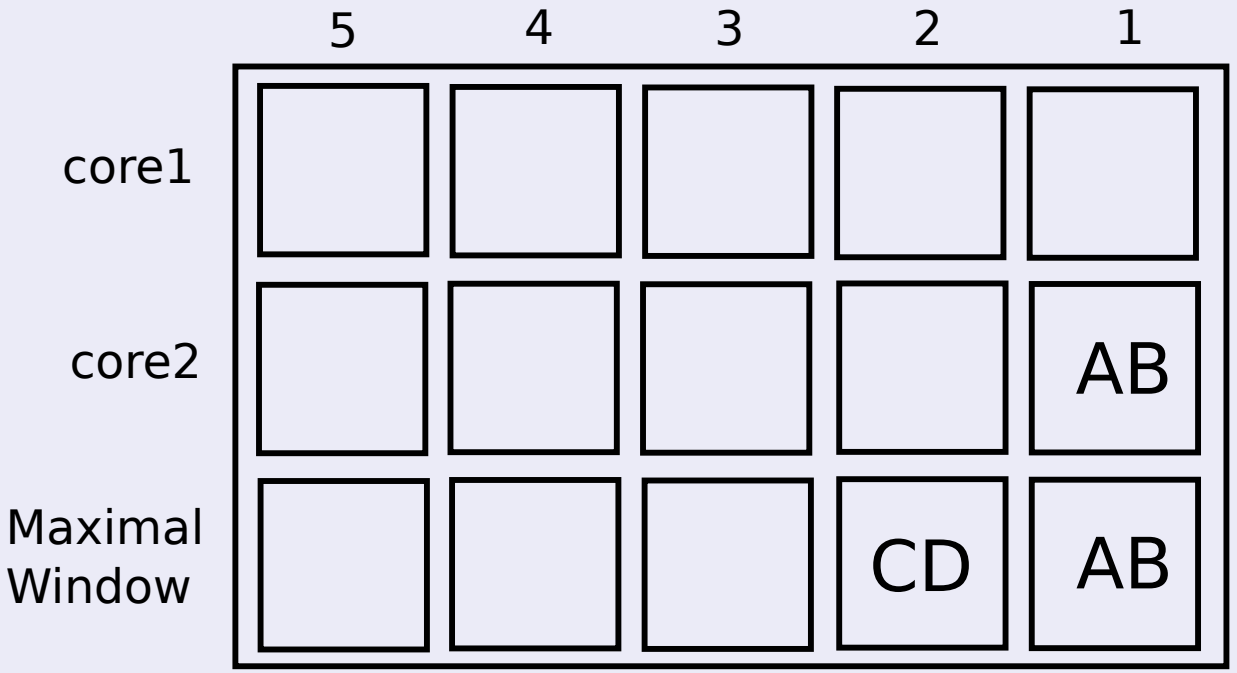
Idea - Exposing HLAVs (II)

Example



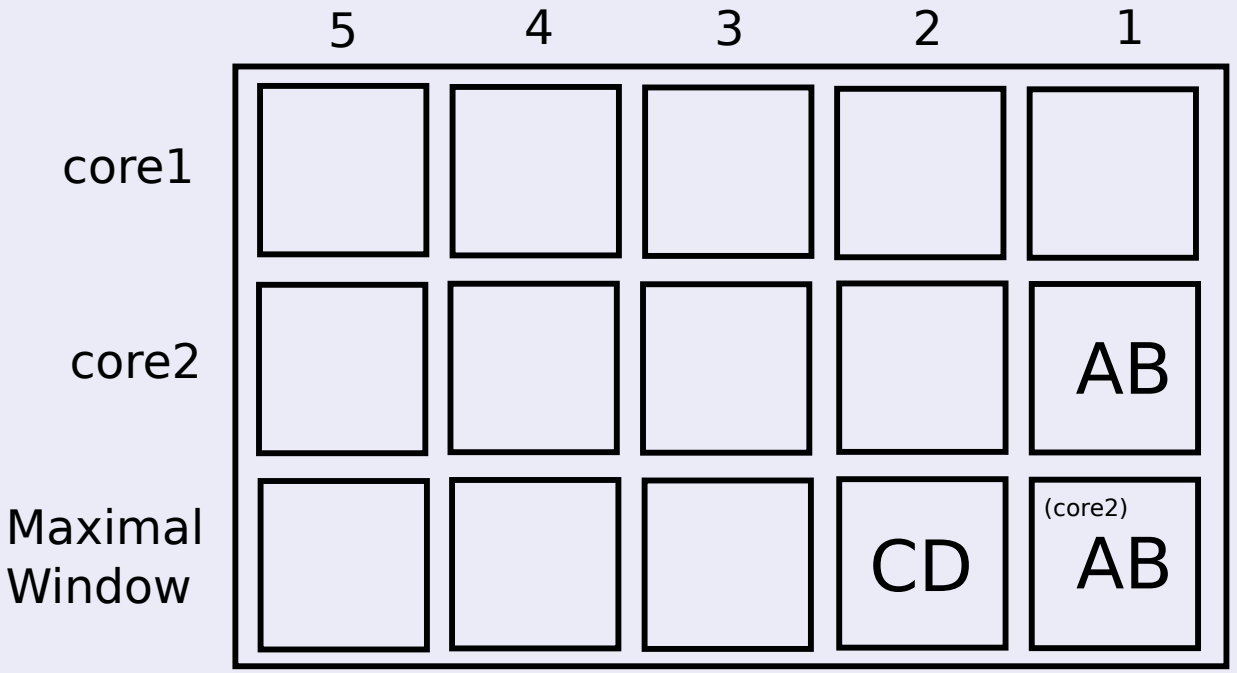
Idea - Exposing HLAVs (II)

Example



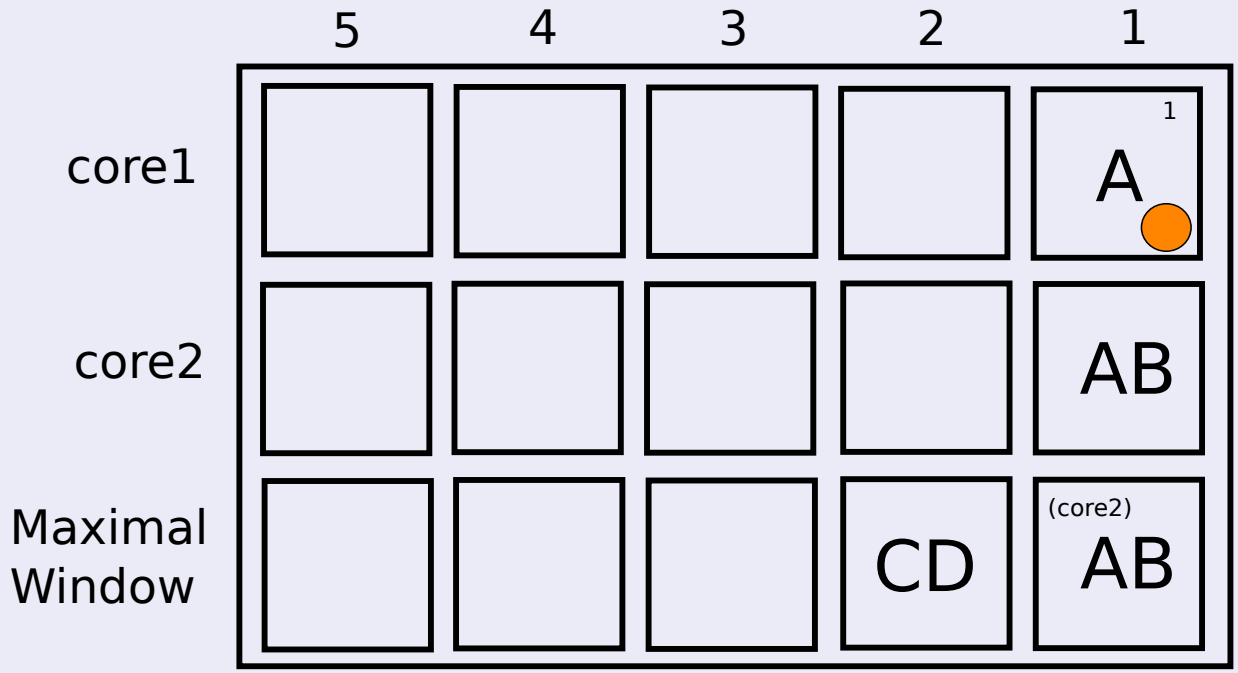
Idea - Exposing HLAVs (II)

Example



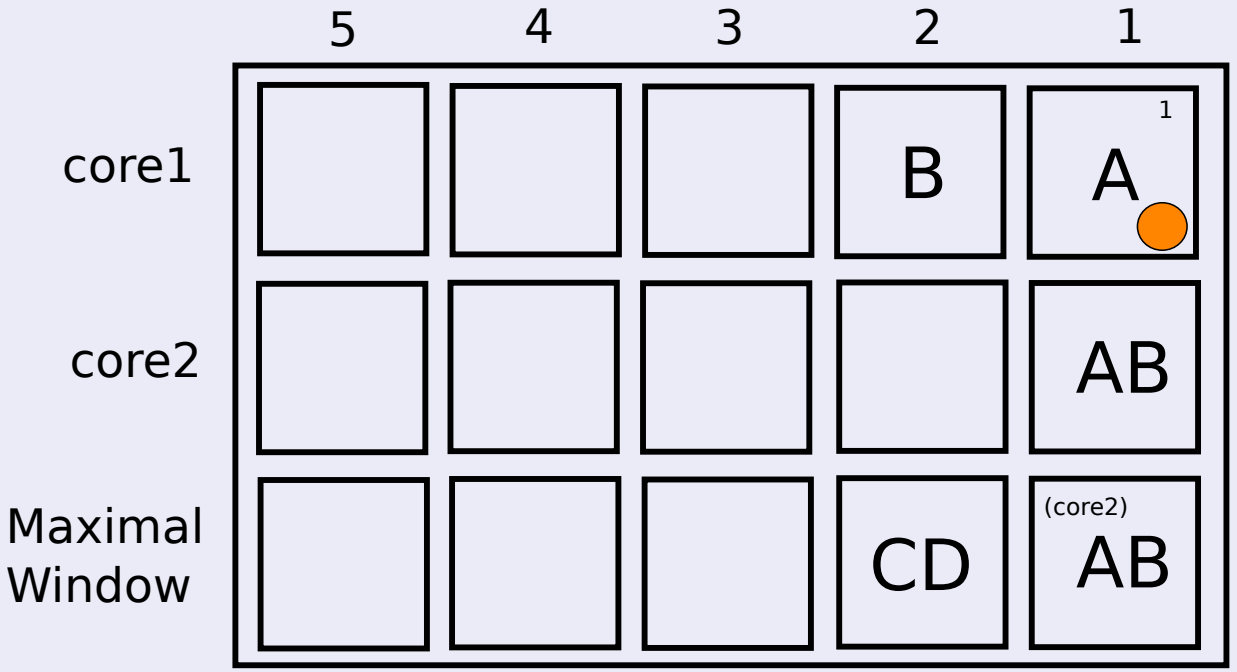
Idea - Exposing HLAVs (II)

Example



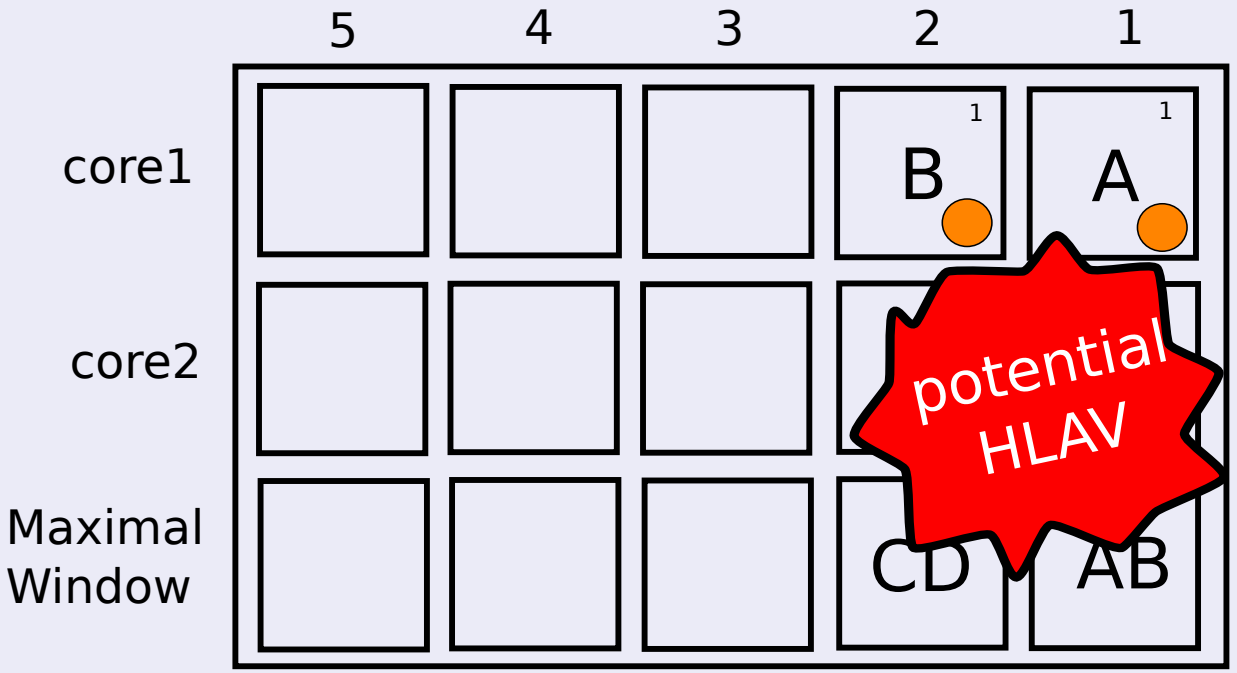
Idea - Exposing HLAVs (II)

Example



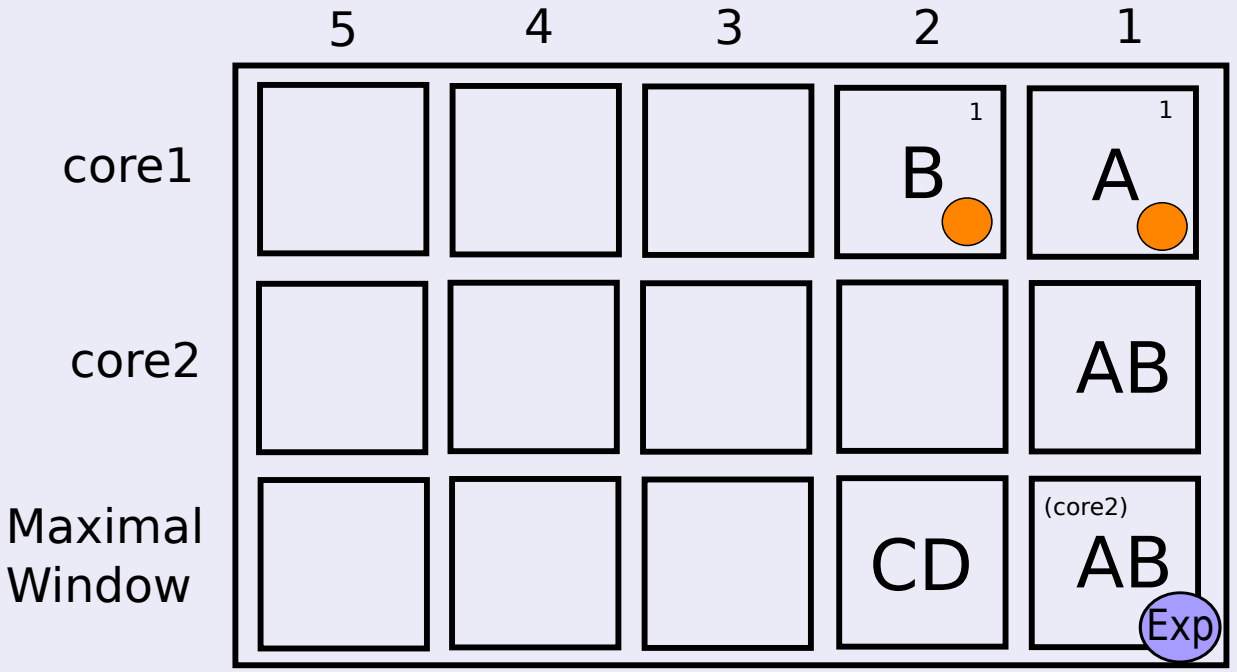
Idea - Exposing HLAVs (II)

Example



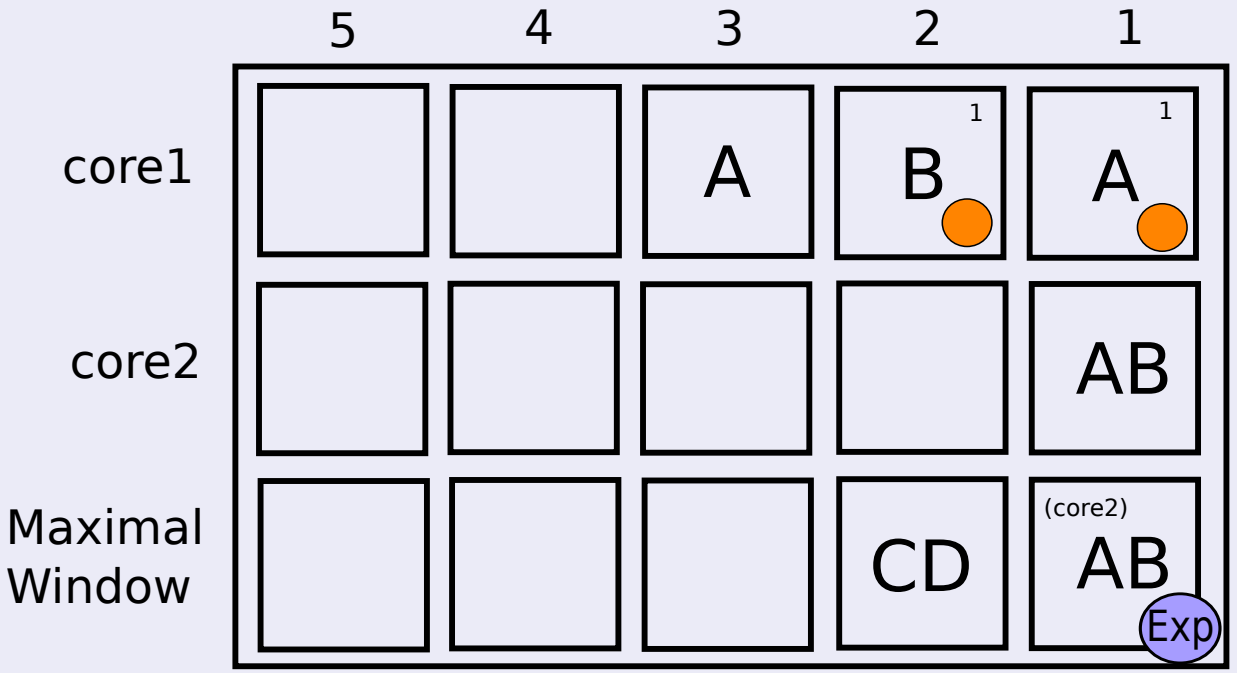
Idea - Exposing HLAVs (II)

Example



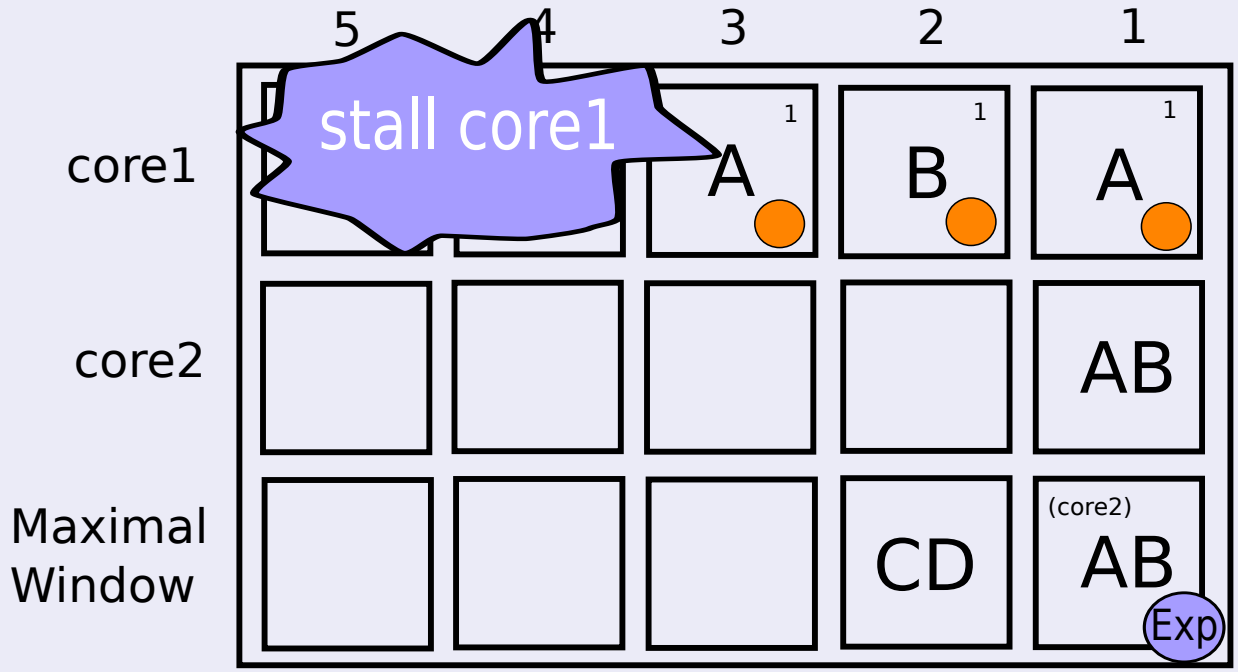
Idea - Exposing HLAVs (II)

Example



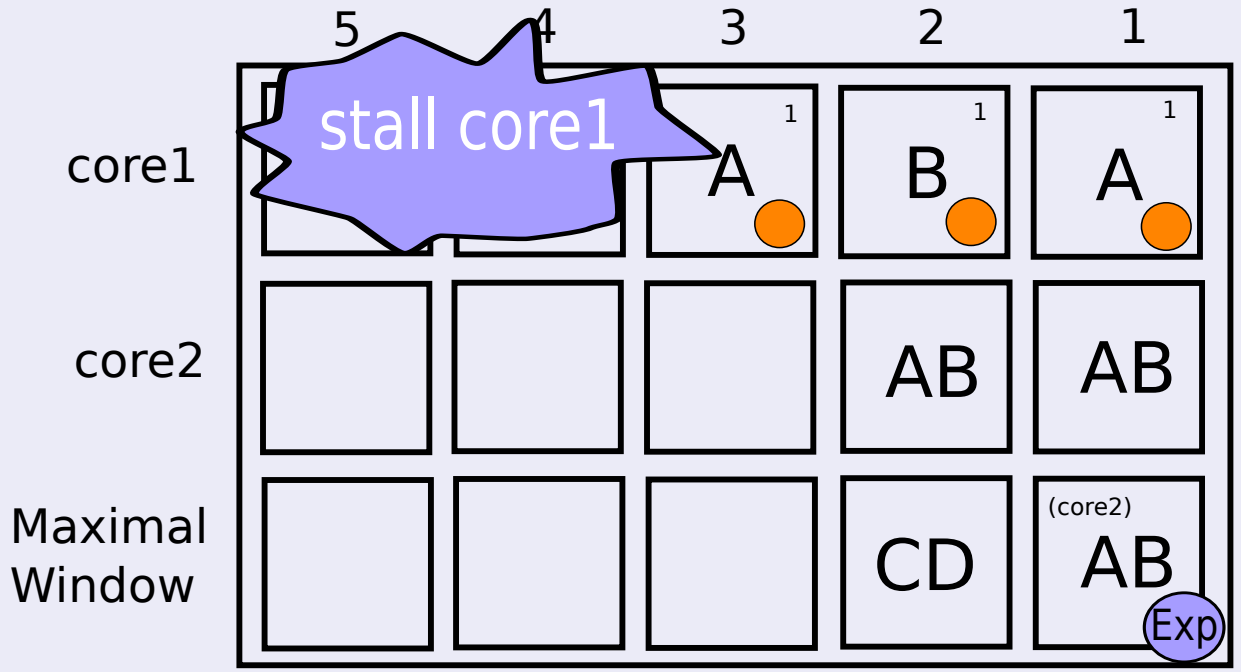
Idea - Exposing HLAVs (II)

Example



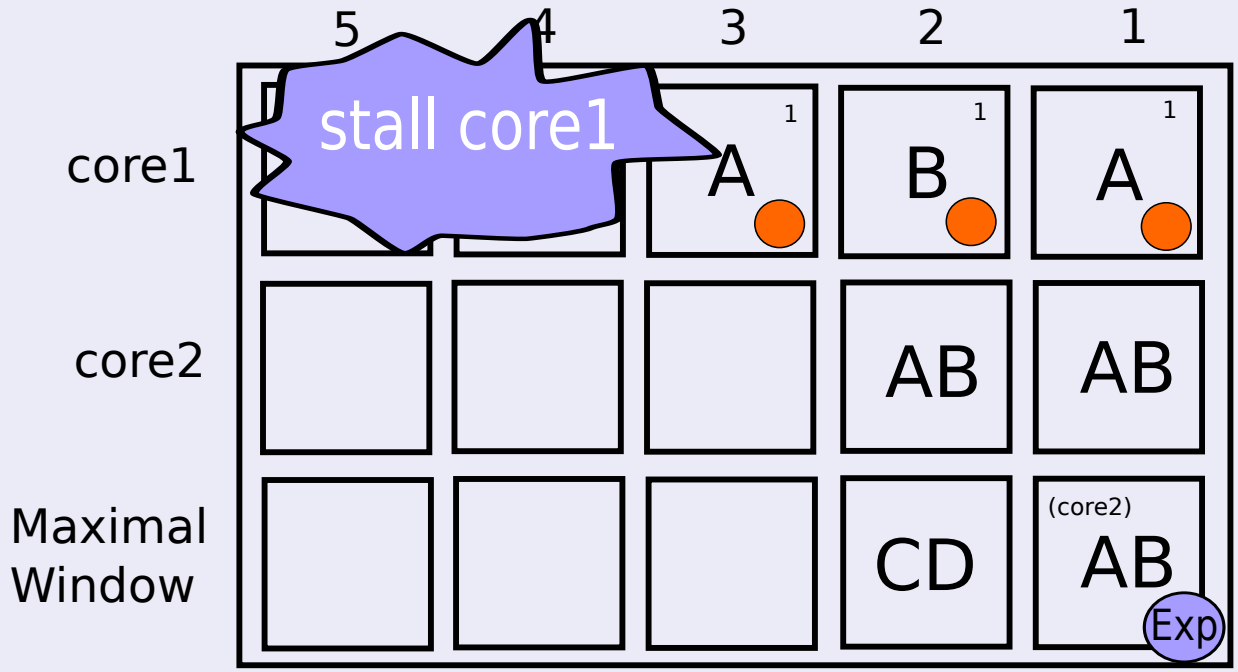
Idea - Exposing HLAVs (II)

Example



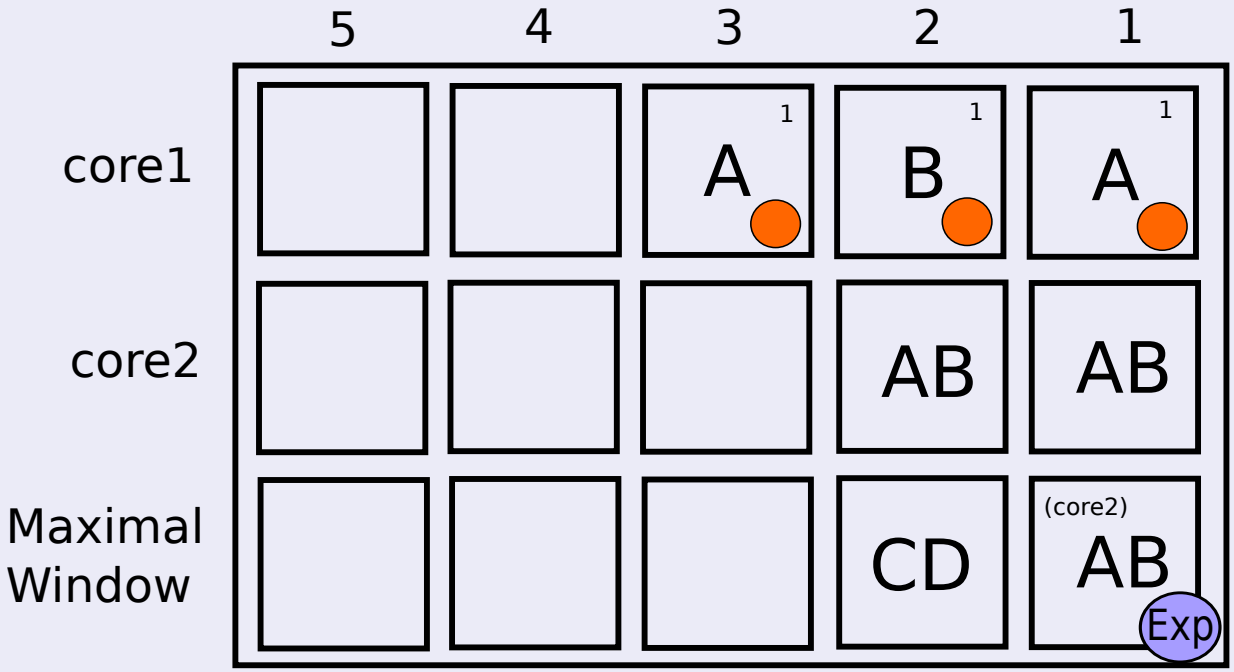
Idea - Exposing HLAVs (II)

Example



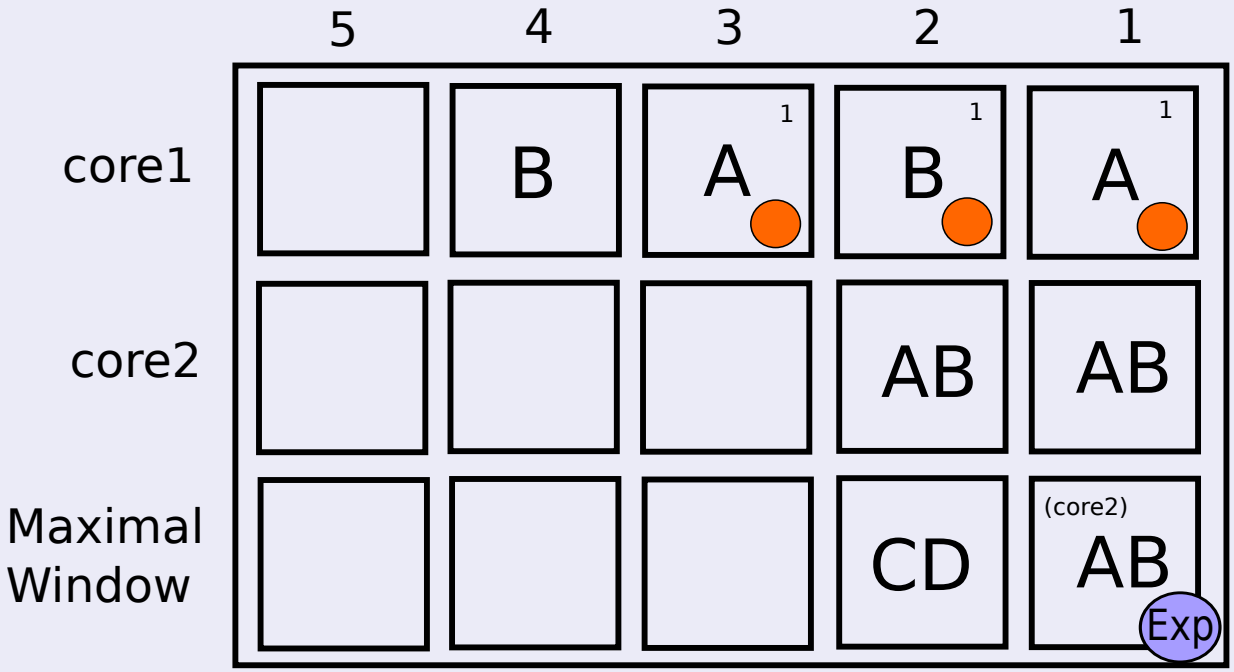
Idea - Exposing HLAVs (II)

Example



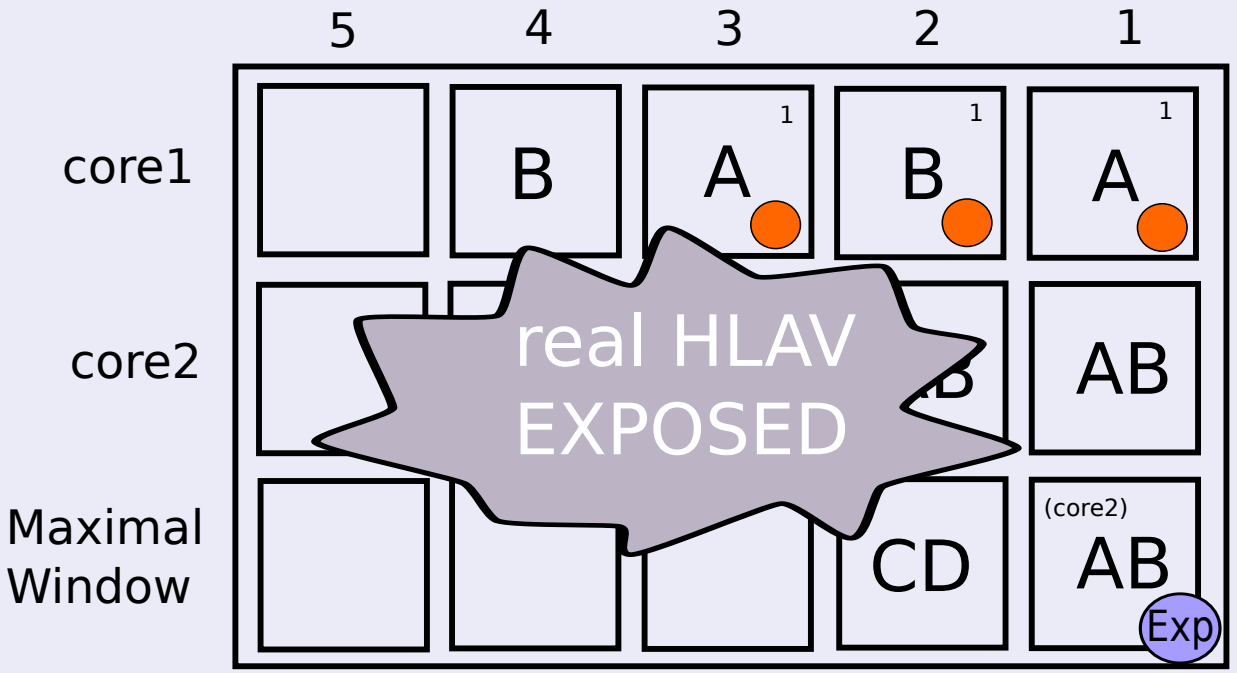
Idea - Exposing HLAVs (II)

Example



Idea - Exposing HLAVs (II)

Example



Contents

1. Introduction

2. Idea
   - Detecting HLAVs
   - Exposing HLAVs
   - Tolerating HLAVs

3. Conclusions

Idea - Tolerating HLAVs (I)

General Idea

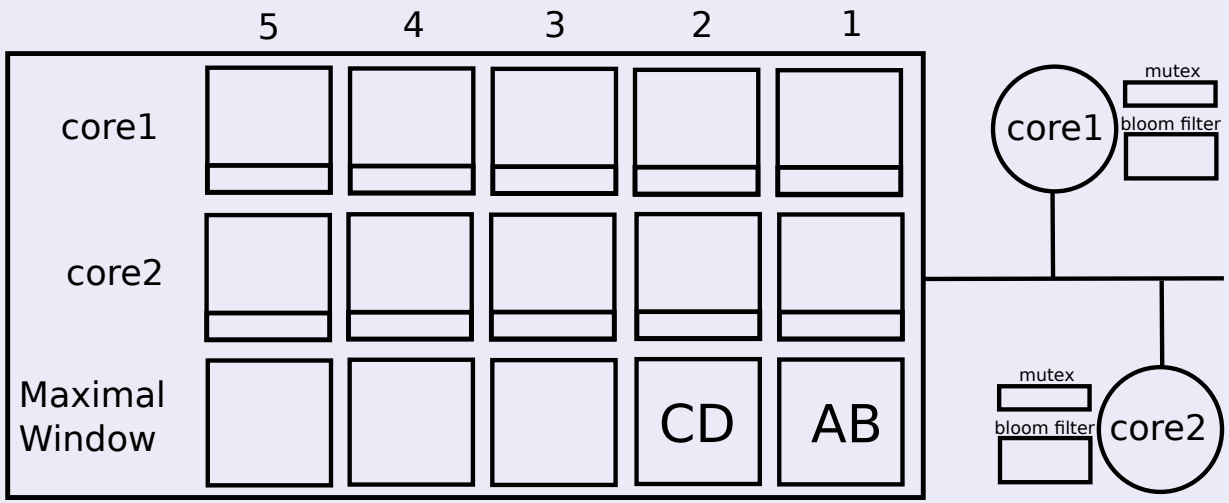
- Useful in production runs
- Tries to avoid real HLAVs
- Uses information based on previous potential HLAV
- Uses transactions to protect regions that are suspect to be buggy

Additions

- Lock addresses are tracked, and used by the module
- Cores have a list of suspicious locks (to start Tx)
- The views in the module could be now in speculative state

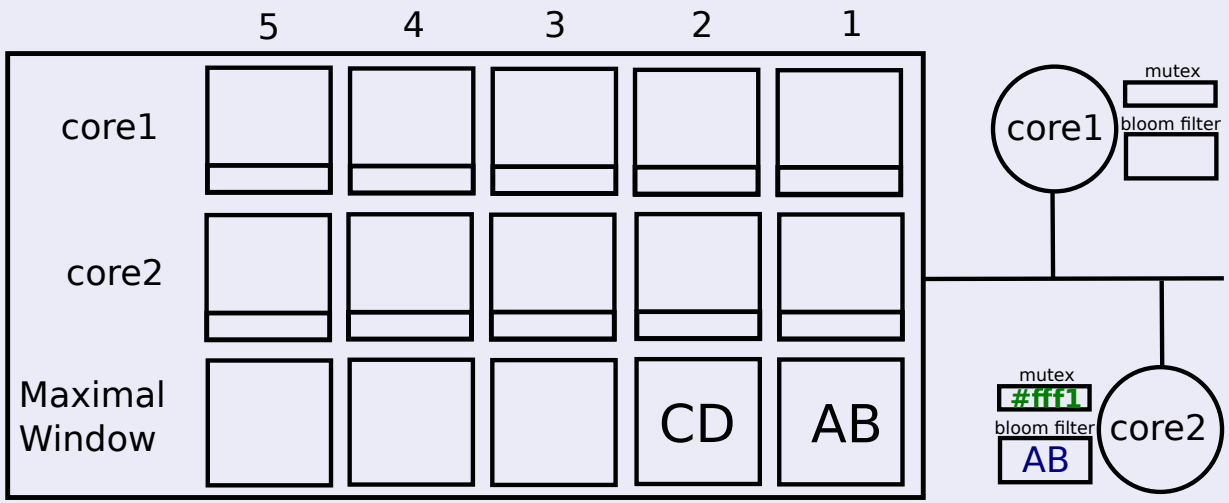
Idea - Tolerating HLAVs (II)

Example



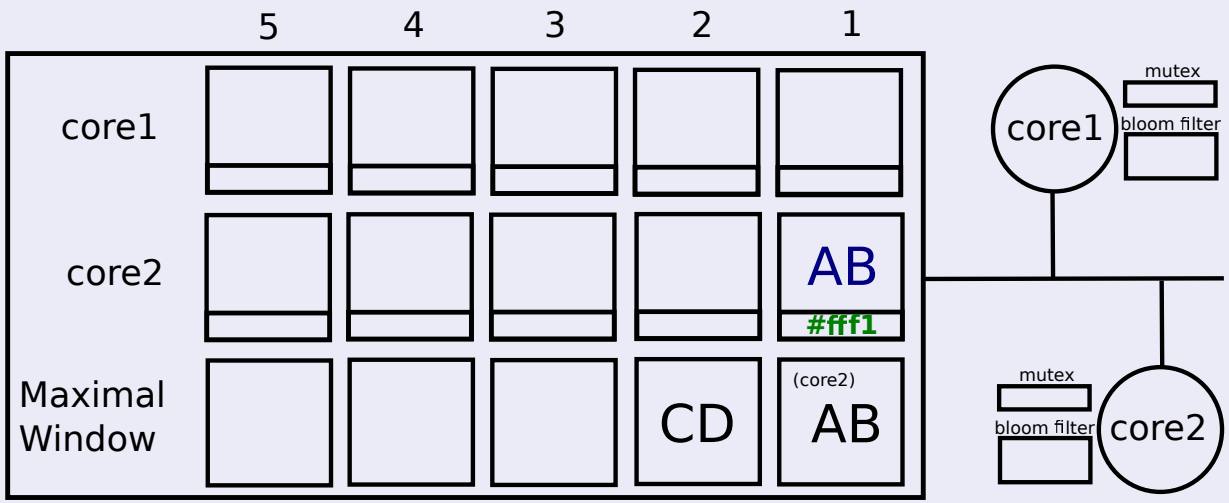
Idea - Tolerating HLAVs (II)

Example



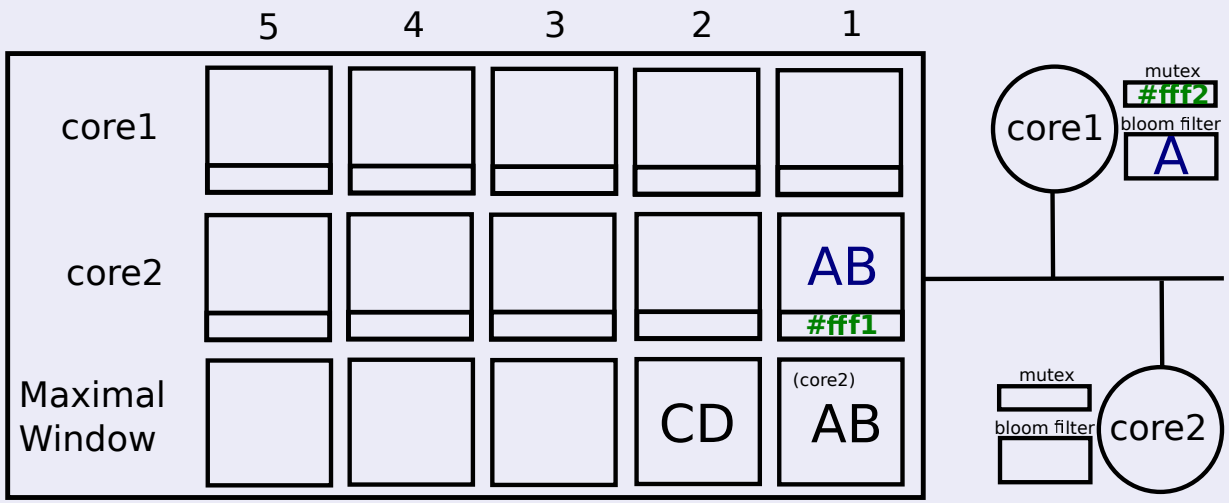
Idea - Tolerating HLAVs (II)

Example



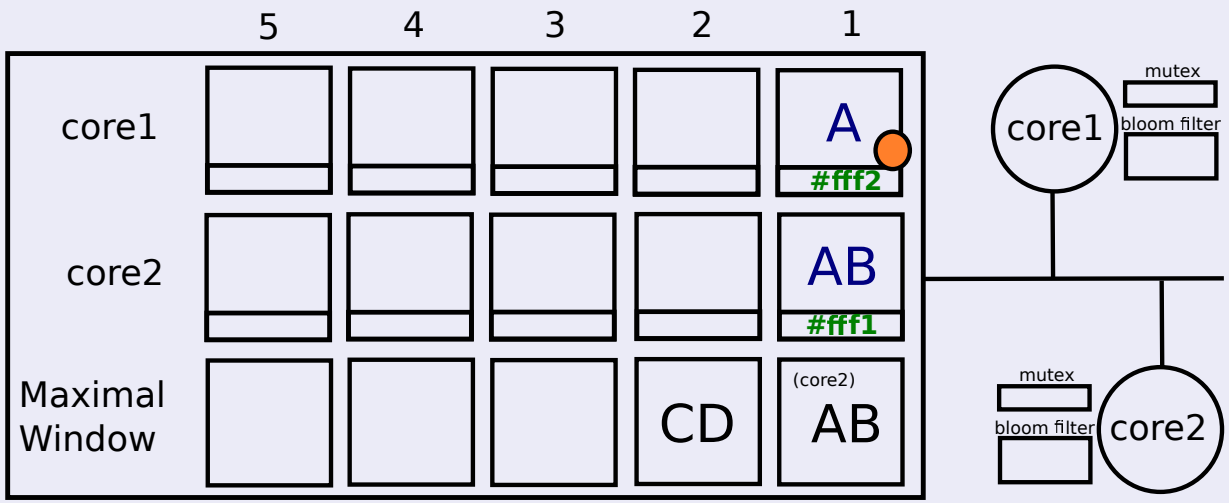
Idea - Tolerating HLAVs (II)

Example



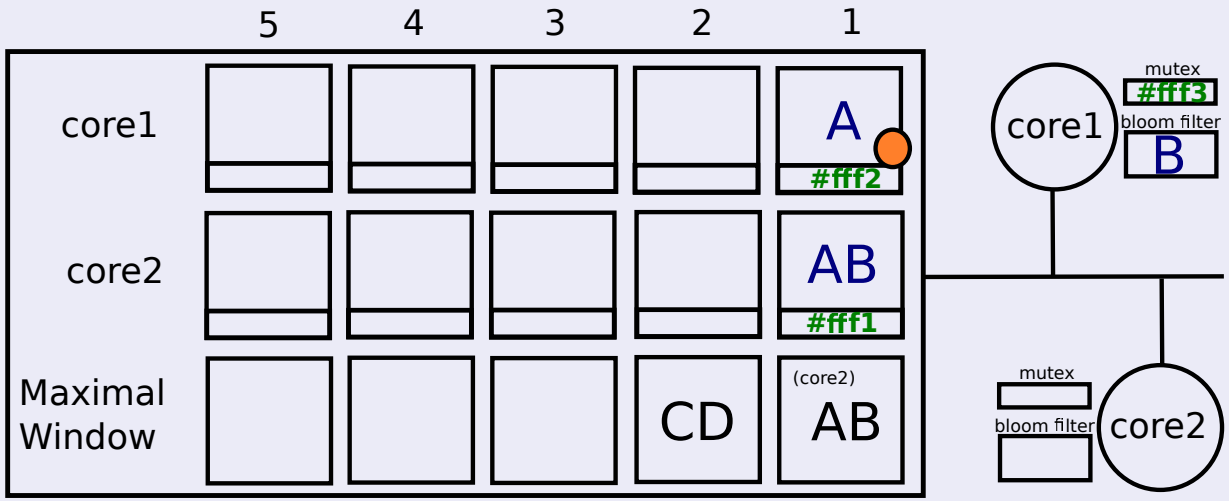
Idea - Tolerating HLAVs (II)

Example



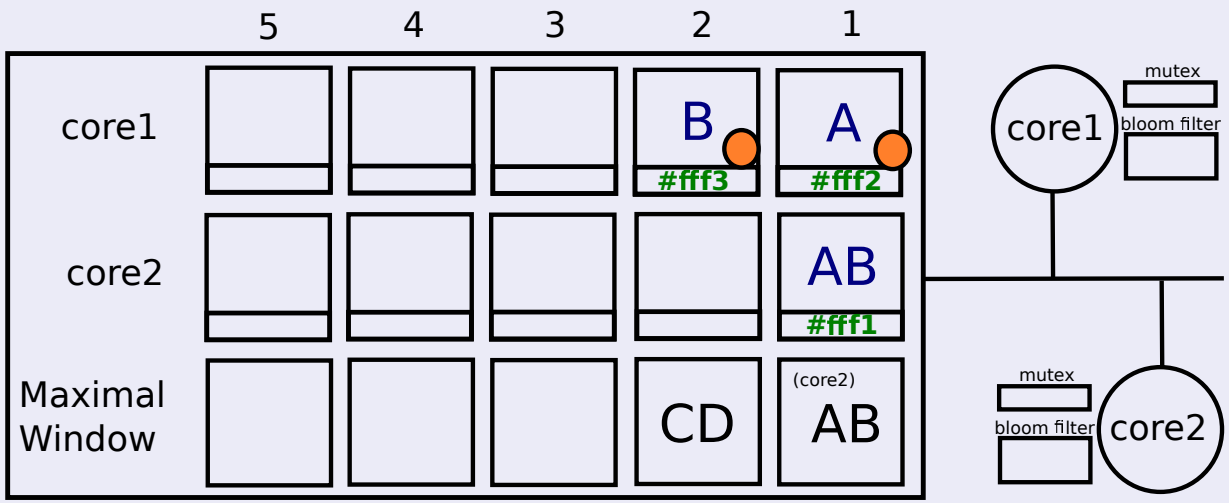
Idea - Tolerating HLAVs (II)

Example



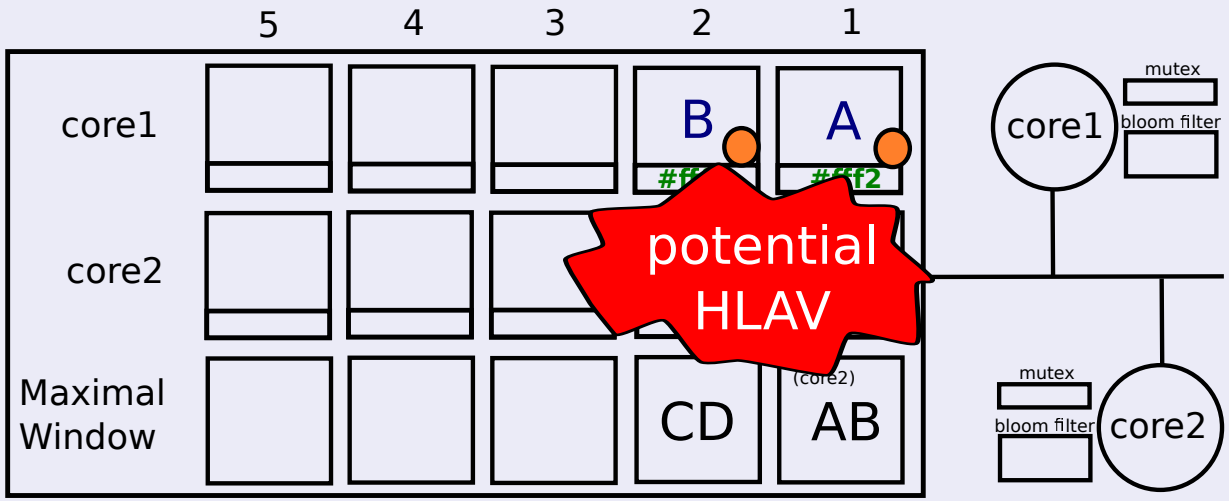
Idea - Tolerating HLAVs (II)

Example



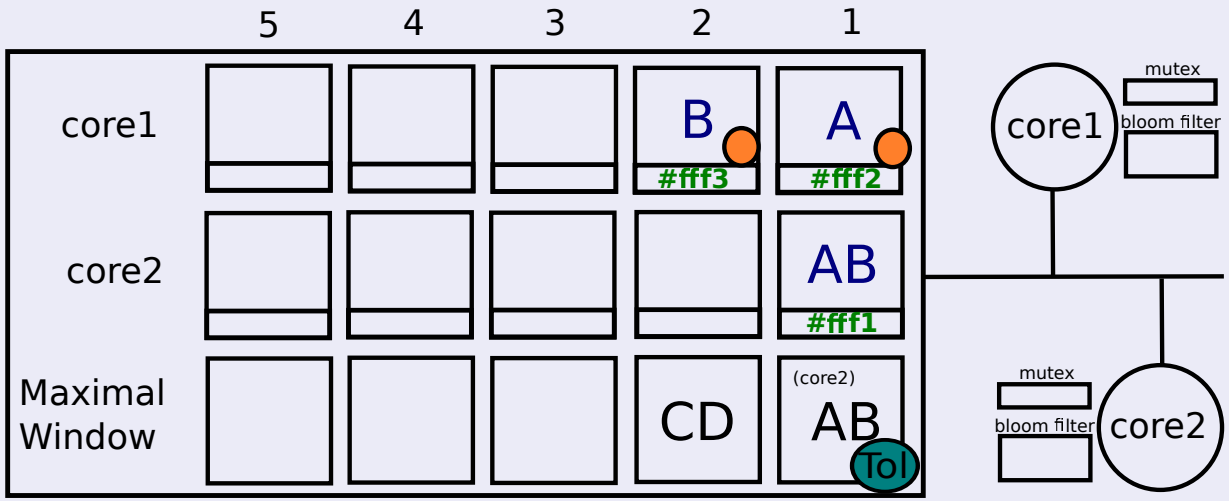
Idea - Tolerating HLAVs (II)

Example



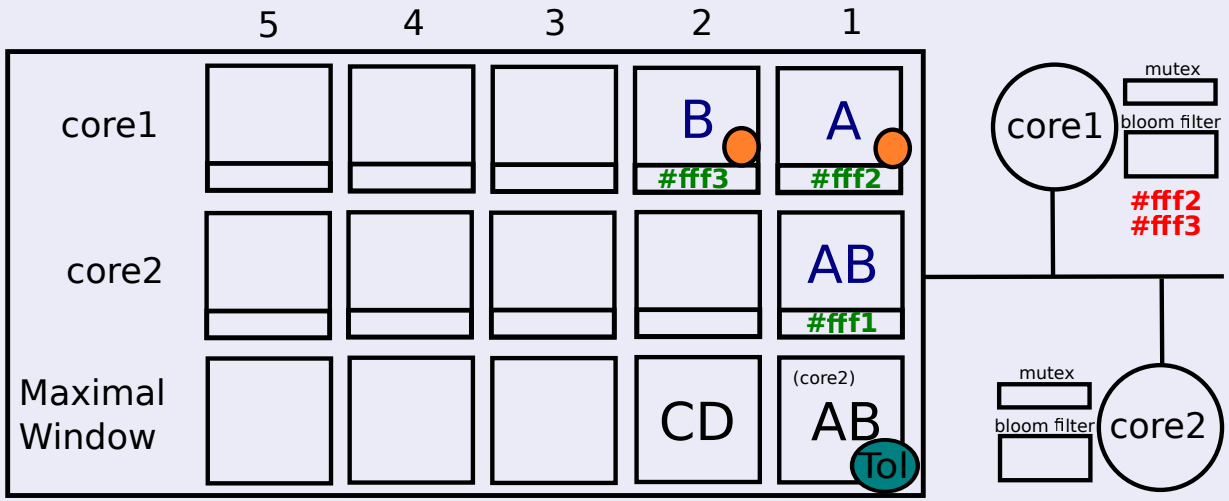
Idea - Tolerating HLAVs (II)

Example



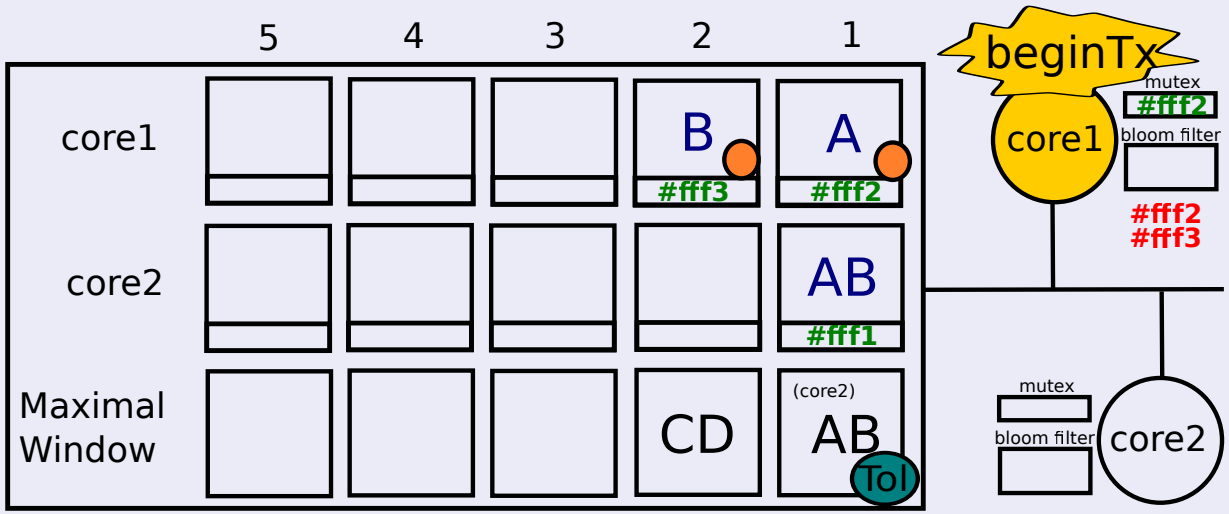
Idea - Tolerating HLAVs (II)

Example



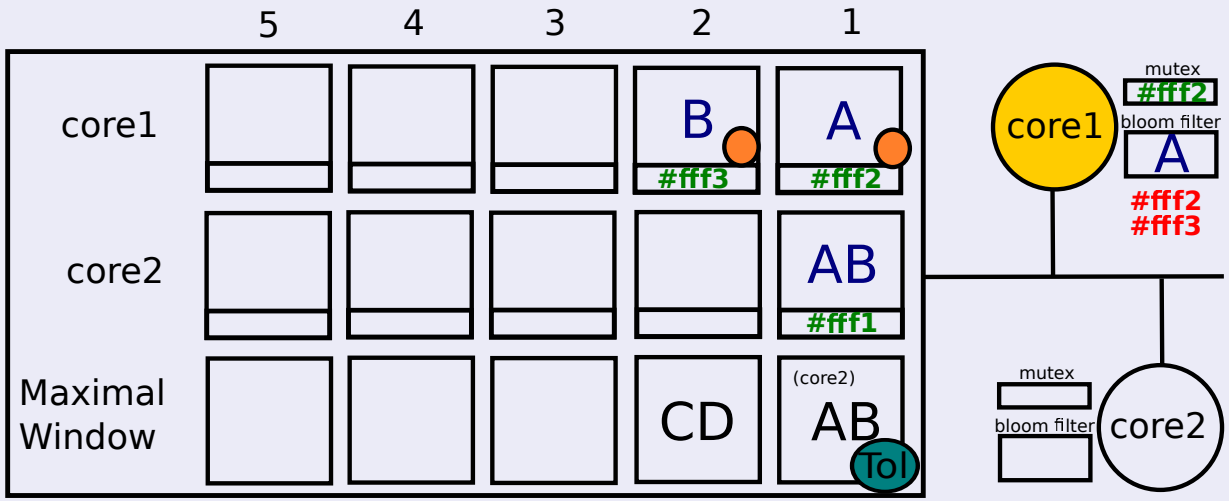
Idea - Tolerating HLAVs (II)

Example



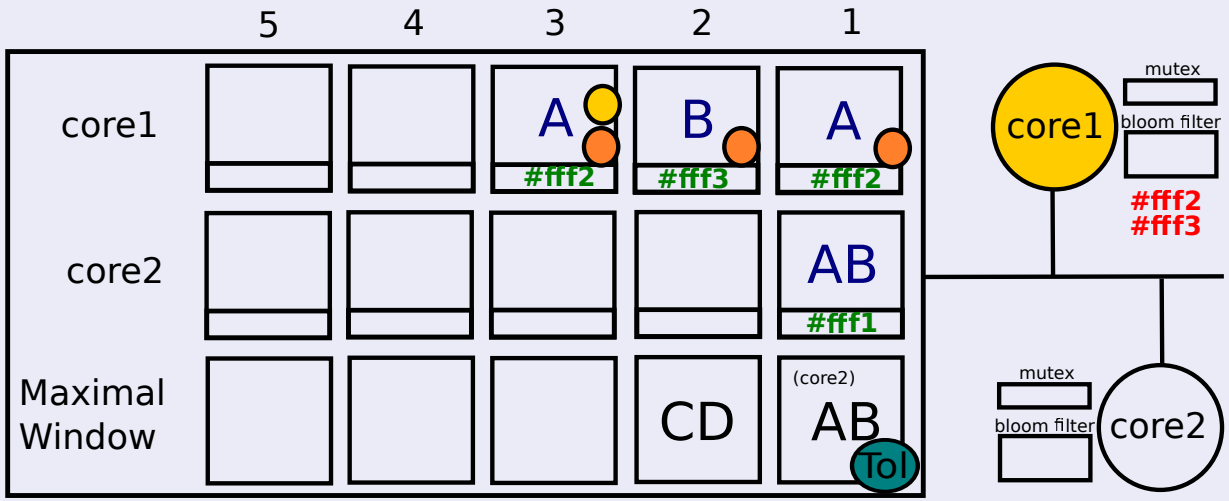
Idea - Tolerating HLAVs (II)

Example



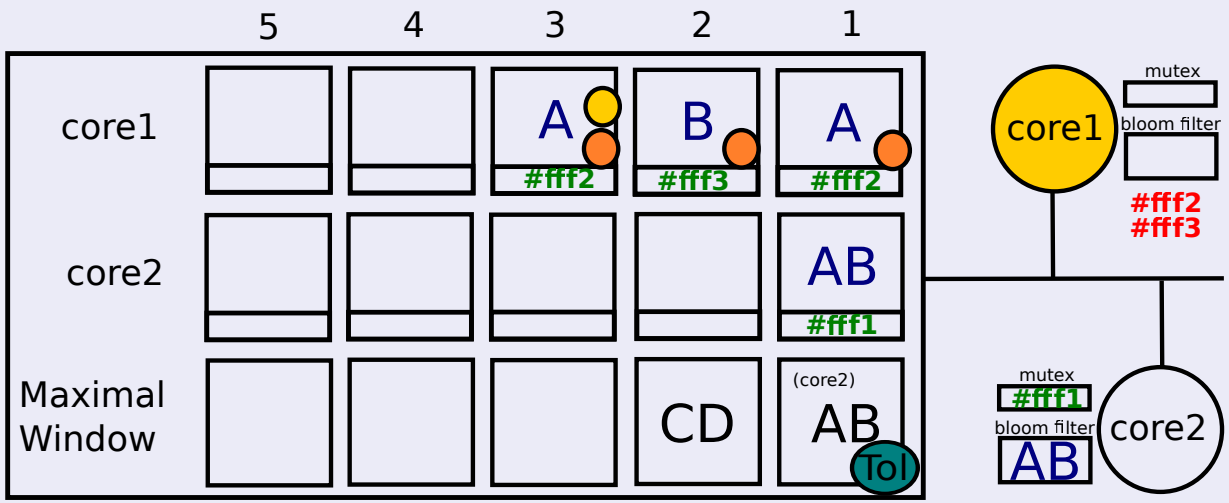
Idea - Tolerating HLAVs (II)

Example



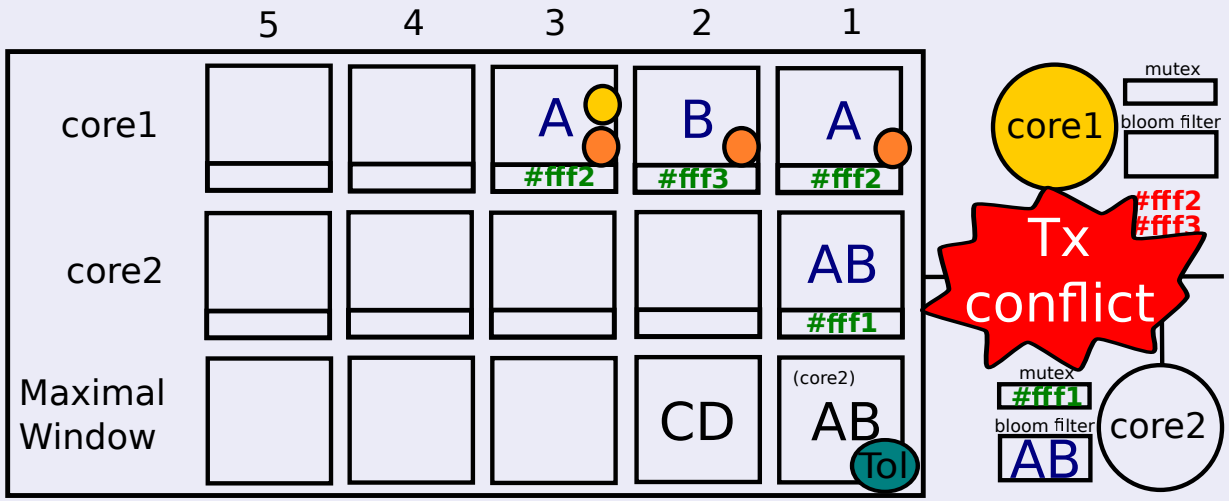
Idea - Tolerating HLAVs (II)

Example



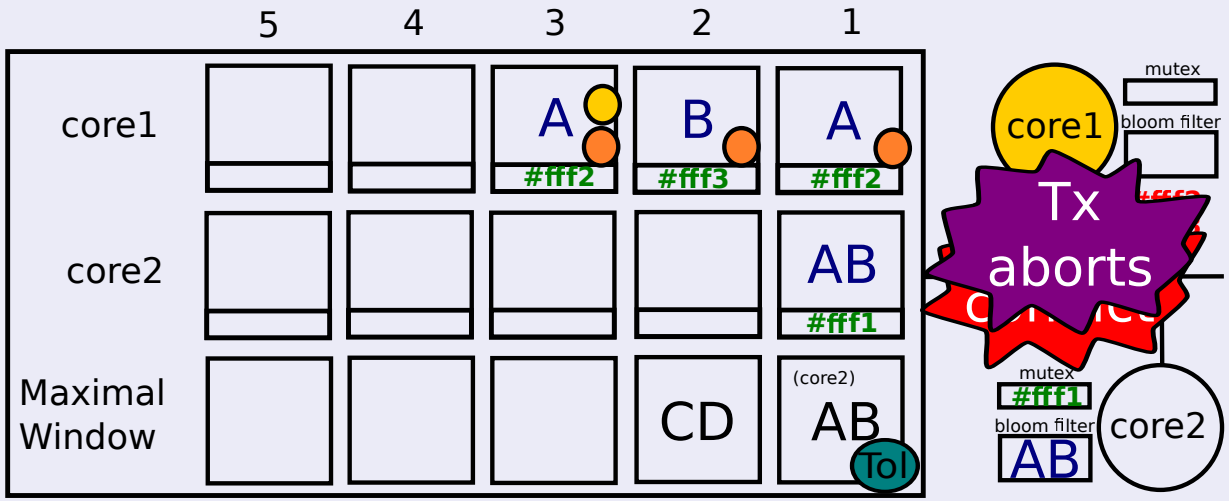
Idea - Tolerating HLAVs (II)

Example



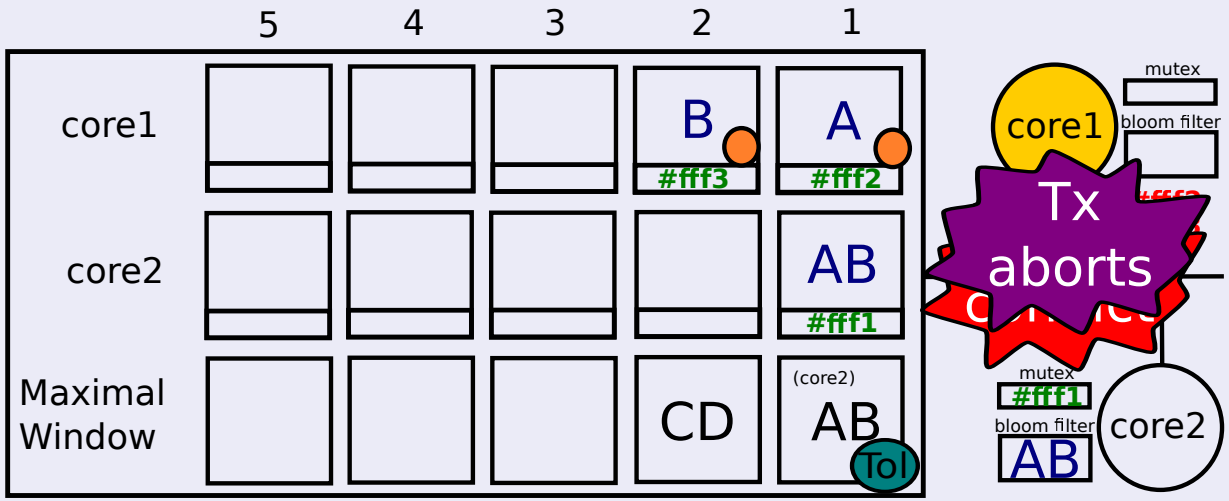
Idea - Tolerating HLAVs (II)

Example



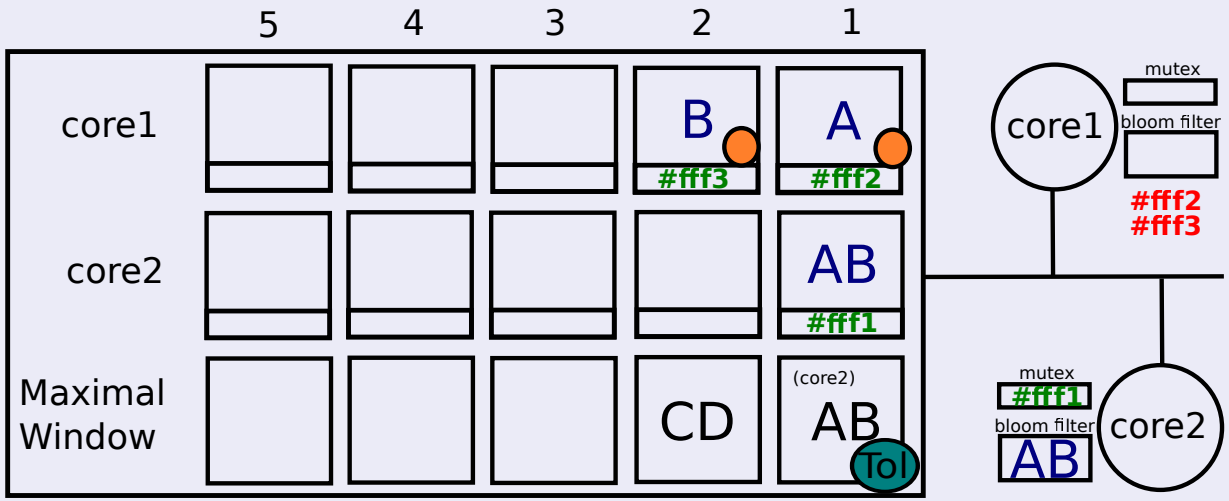
Idea - Tolerating HLAVs (II)

Example



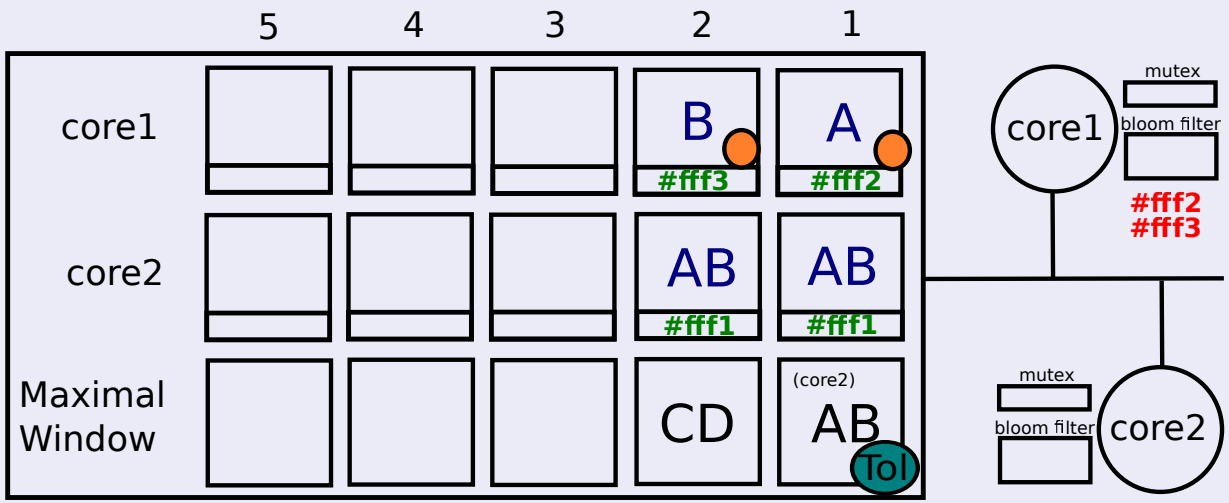
Idea - Tolerating HLAVs (II)

Example



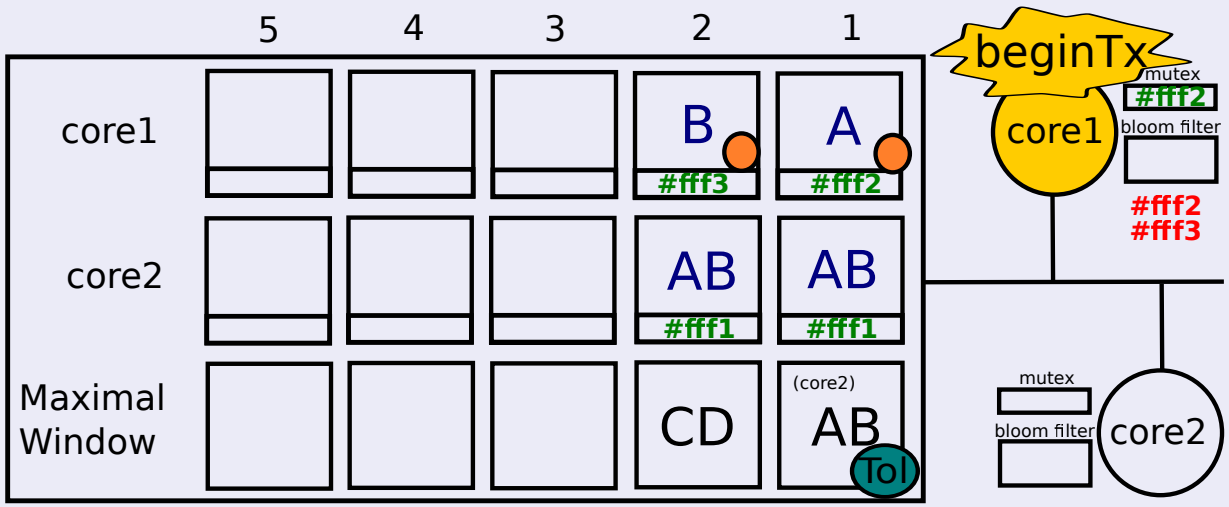
Idea - Tolerating HLAVs (II)

Example



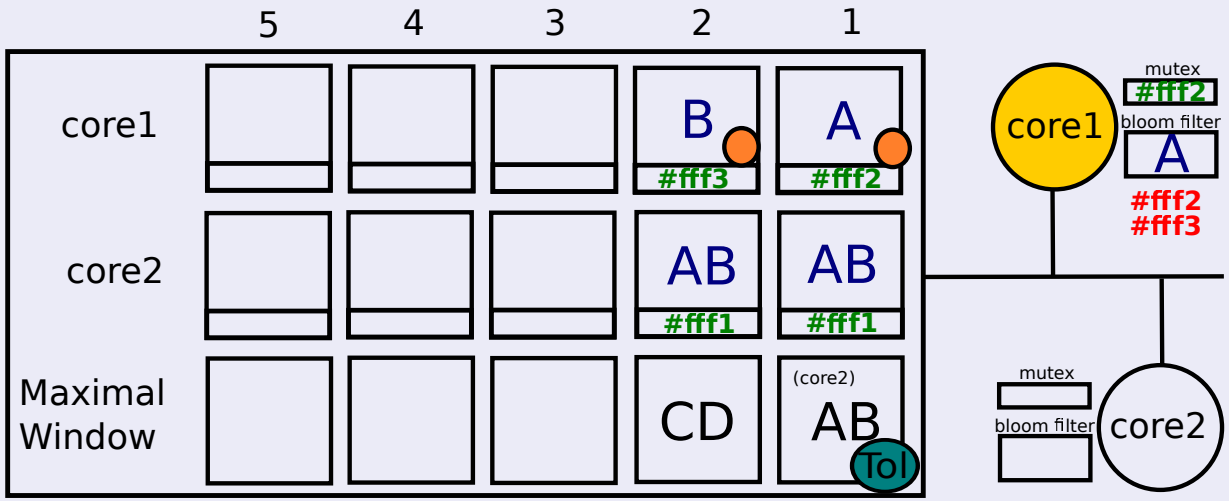
Idea - Tolerating HLAVs (II)

Example



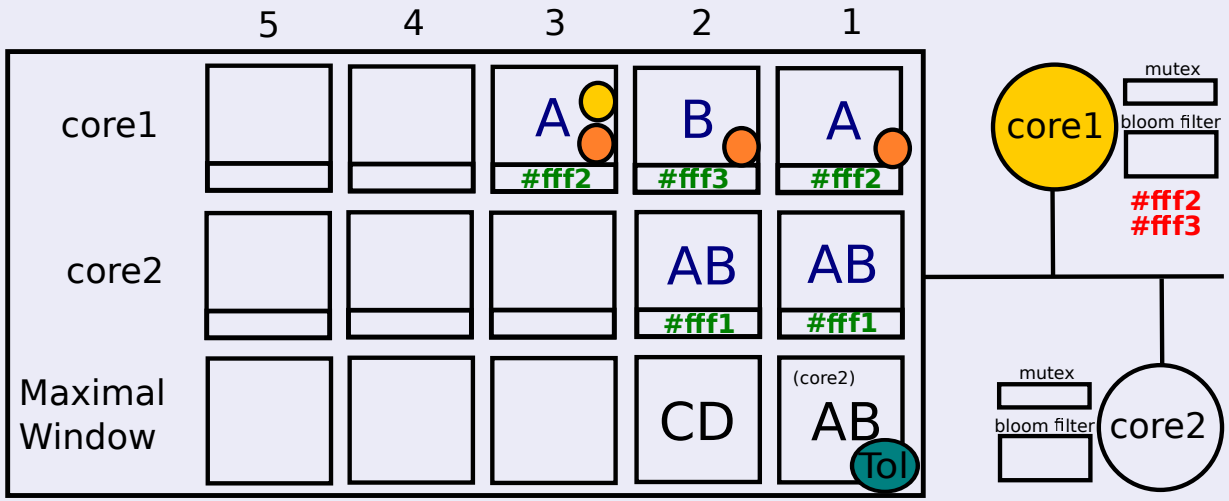
Idea - Tolerating HLAVs (II)

Example



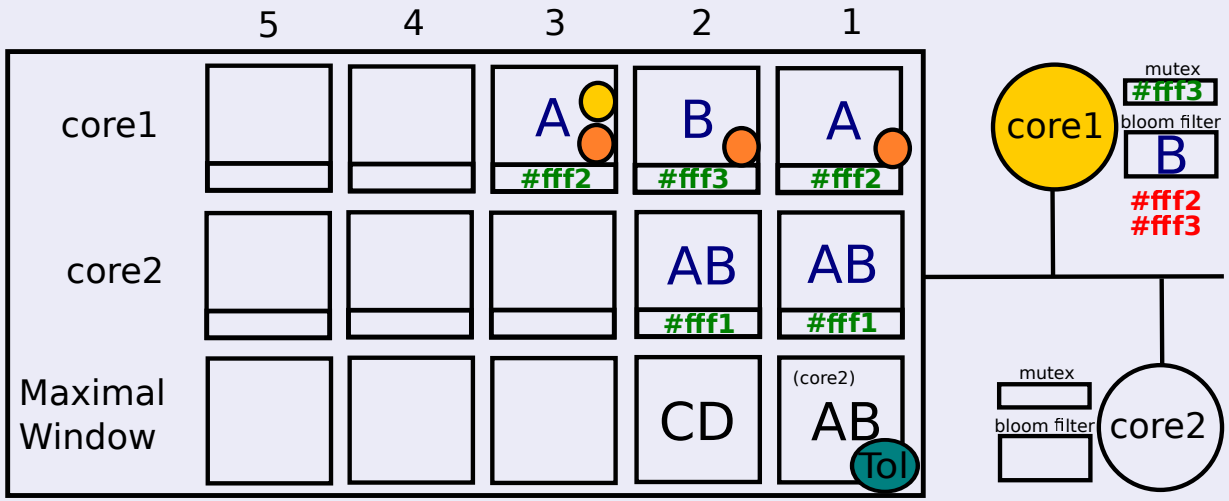
Idea - Tolerating HLAVs (II)

Example



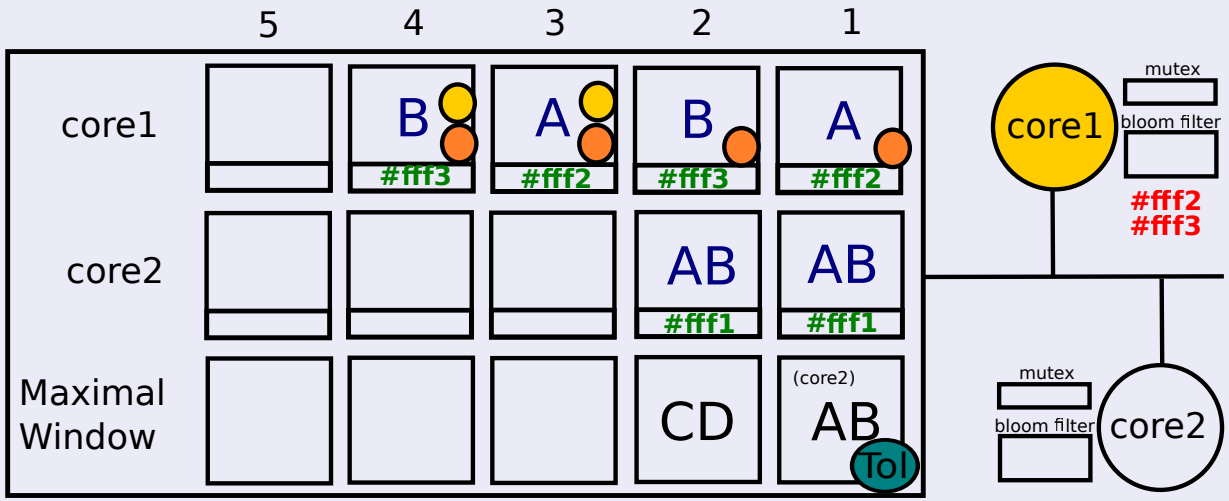
Idea - Tolerating HLAVs (II)

Example



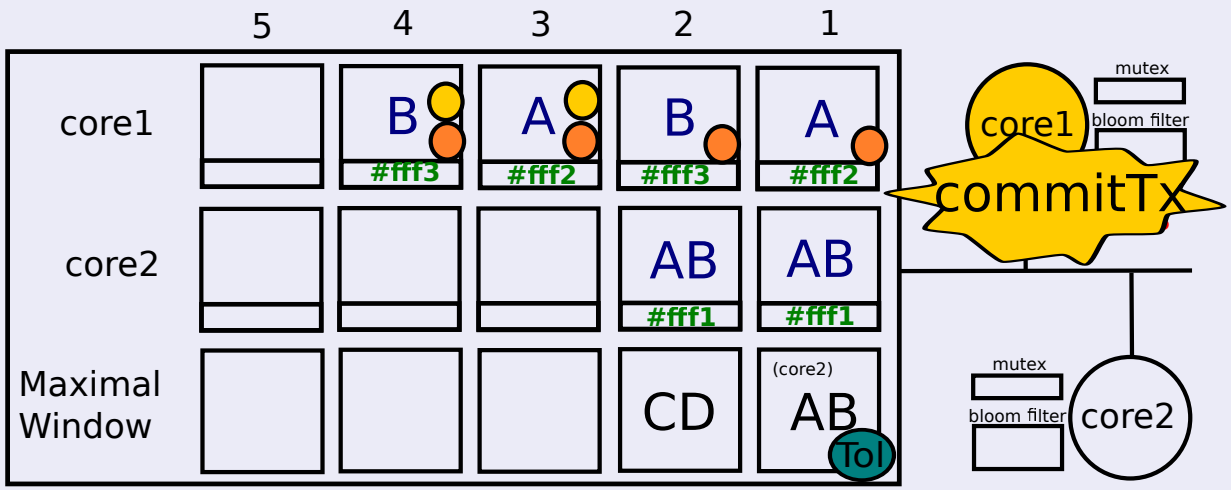
Idea - Tolerating HLAVs (II)

Example



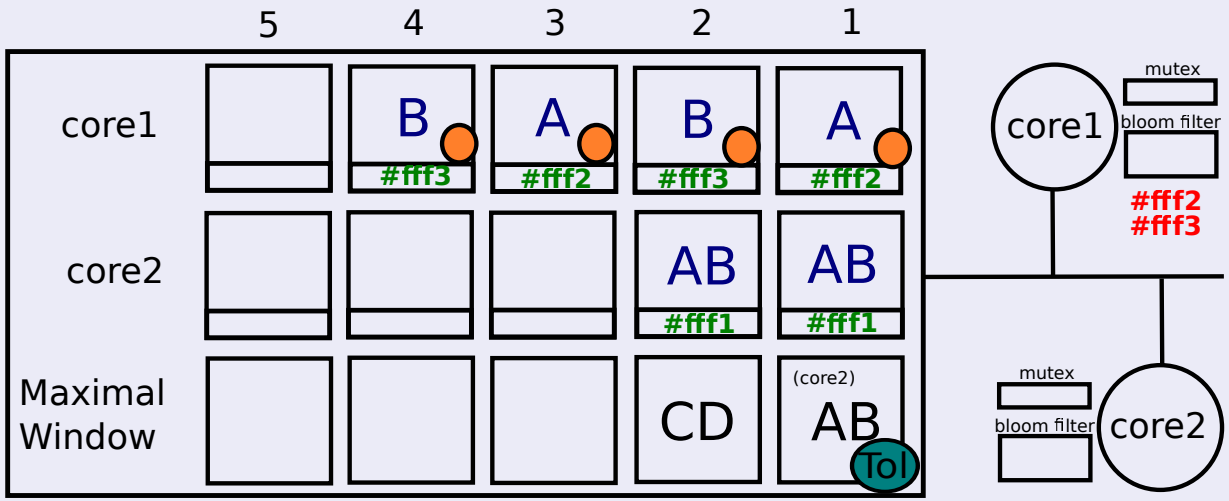
Idea - Tolerating HLAVs (II)

Example



Idea - Tolerating HLAVs (II)

Example



Contents

1. Introduction
2. Idea
   - Detecting HLAVs
   - Exposing HLAVs
   - Tolerating HLAVs
3. Conclusions

Conclusions

- First hardware approach addressing the detection of HLAVs
- The module Exposes bugs
- The module Tolerates bugs in production runs
- Very low performance overhead

Questions

Thank you very much!

Questions?