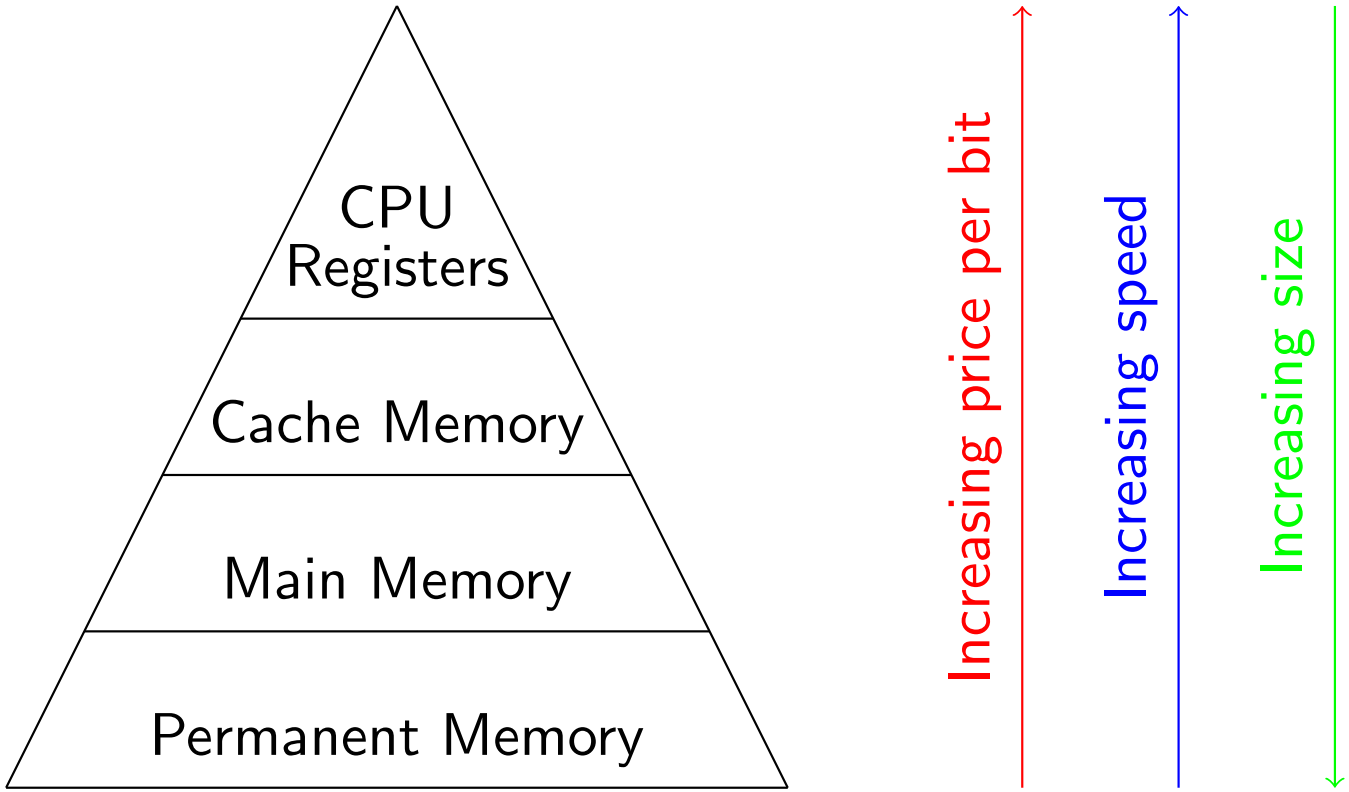
Learning Cache Replacement Policies using Register Automata

Guillem Rueda Cebollero

Uppsala Universitet

What is the Cache Memory? (Memory Hierarchy)



Hits / Misses

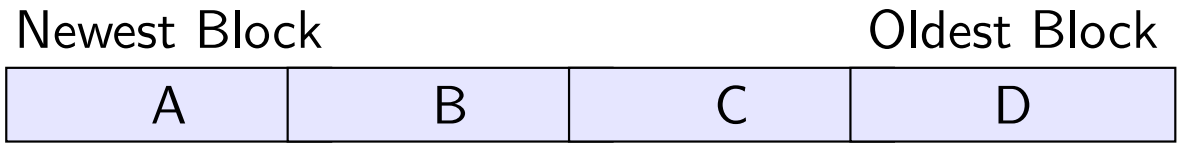
A block replacement tries to have minimum misses possible:

Hit:
When a block is inside of the cache memory.

Miss:
When a block is not in the cache memory. It means more processing time, because this block must be requested at the next memory level.

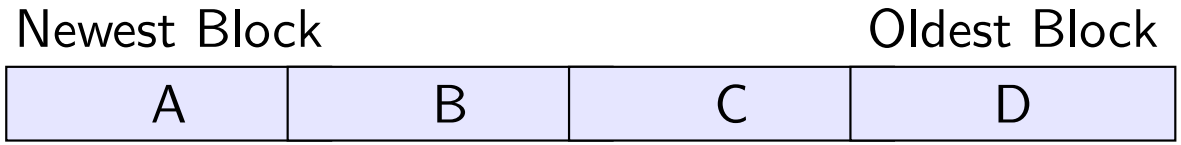
What is the Cache Memory? (Replacement Policy, LRU)

Initial conditions:

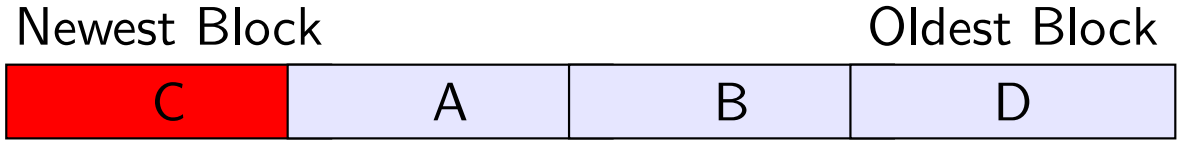


What is the Cache Memory? (Replacement Policy, LRU)

Initial conditions:

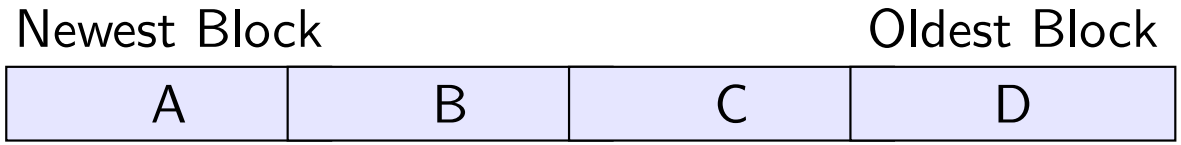


After hit:

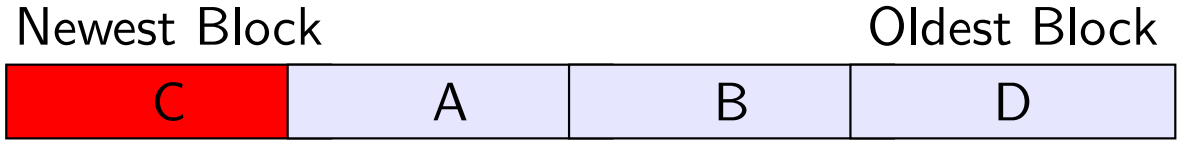


What is the Cache Memory? (Replacement Policy, LRU)

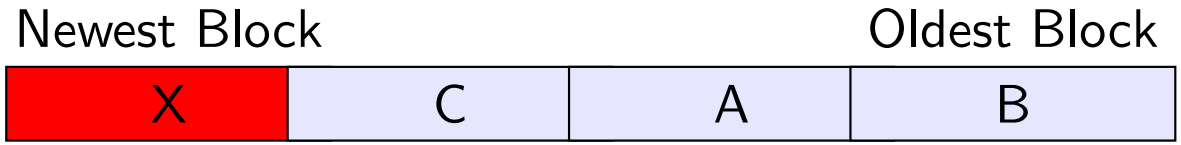
Initial conditions:



After hit:

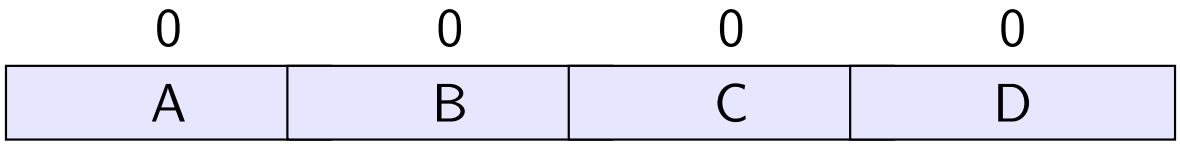


After miss:



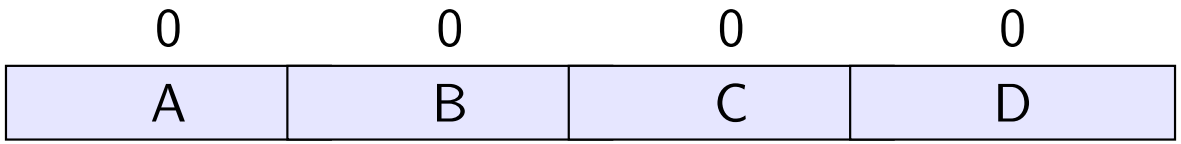
What is the Cache Memory? (Replacement Policy, MRU)

Initial conditions:

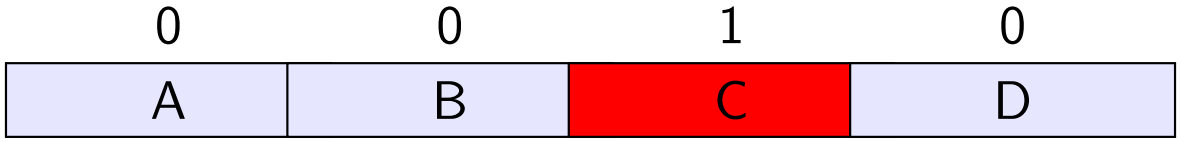


What is the Cache Memory? (Replacement Policy, MRU)

Initial conditions:

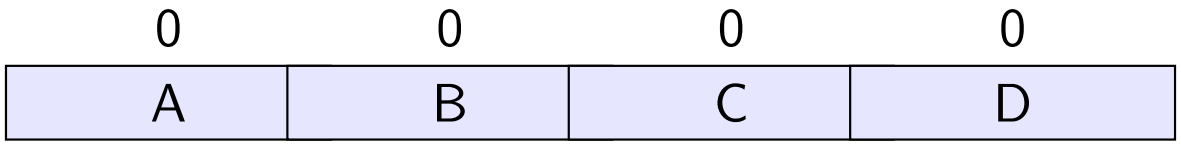


After hit:

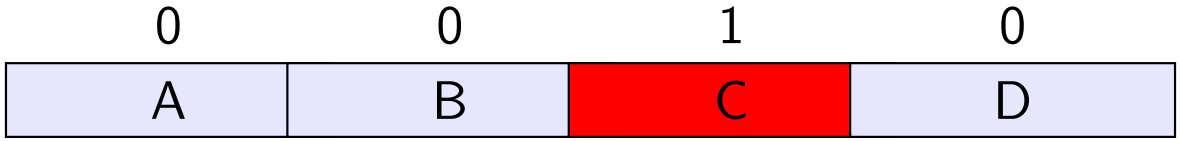


What is the Cache Memory? (Replacement Policy, MRU)

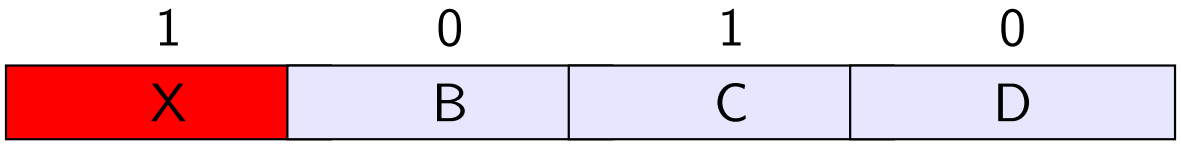
Initial conditions:



After hit:



After miss:



Motivations

The motivations are:

- Which block replacement policy does the system use?
- If the policy is unknown, how does it work?
- System Manufacturer specifications may not report about the cache memory policy

Goals

The goals of the project are:

- Represent using an Automaton a replacement policy.
- Simulate replacement policies and interact with Learning process
- Possible interactions with a real hardware.

ChiPC

- Program for inferring block replacment policy
- Gets all characteristics of cache: size, number of blocks, replacment policy
- Uses Permutation Vectors (PV) to infer the replacment policy
- Developed in C/C++
- Uses the Performance Counters (PC) furnished in the CPU

Permutation Vectors

Where will be the element on position 7 after a hit?

A	B	C	D	E	F	G	H
0	1	2	3	4	5	6	7

D	A	B	C	E	F	G	H
3	0	1	2	4	5	6	7

Permutation Vectors

It's the Fingerprint of the permutation policies. For example, LRU:

$$
\begin{aligned}
\prod_0^{LRU} &= (\boxed{0}; 1; 2; 3; 4; 5; 6; 7) \\
\prod_1^{LRU} &= (\boxed{1}; 0; 2; 3; 4; 5; 6; 7) \\
\prod_2^{LRU} &= (\boxed{2}; 0; 1; 3; 4; 5; 6; 7) \\
\rightarrow \prod_3^{LRU} &= (\boxed{3}; 0; 1; 2; 4; 5; 6; 7) \leftarrow \\
\prod_4^{LRU} &= (\boxed{4}; 0; 1; 2; 3; 5; 6; 7) \\
\prod_5^{LRU} &= (\boxed{5}; 0; 1; 2; 3; 4; 6; 7) \\
\prod_6^{LRU} &= (\boxed{6}; 0; 1; 2; 3; 4; 5; 7) \\
\prod_7^{LRU} &= (\boxed{7}; 0; 1; 2; 3; 4; 5; 6)
\end{aligned}
$$

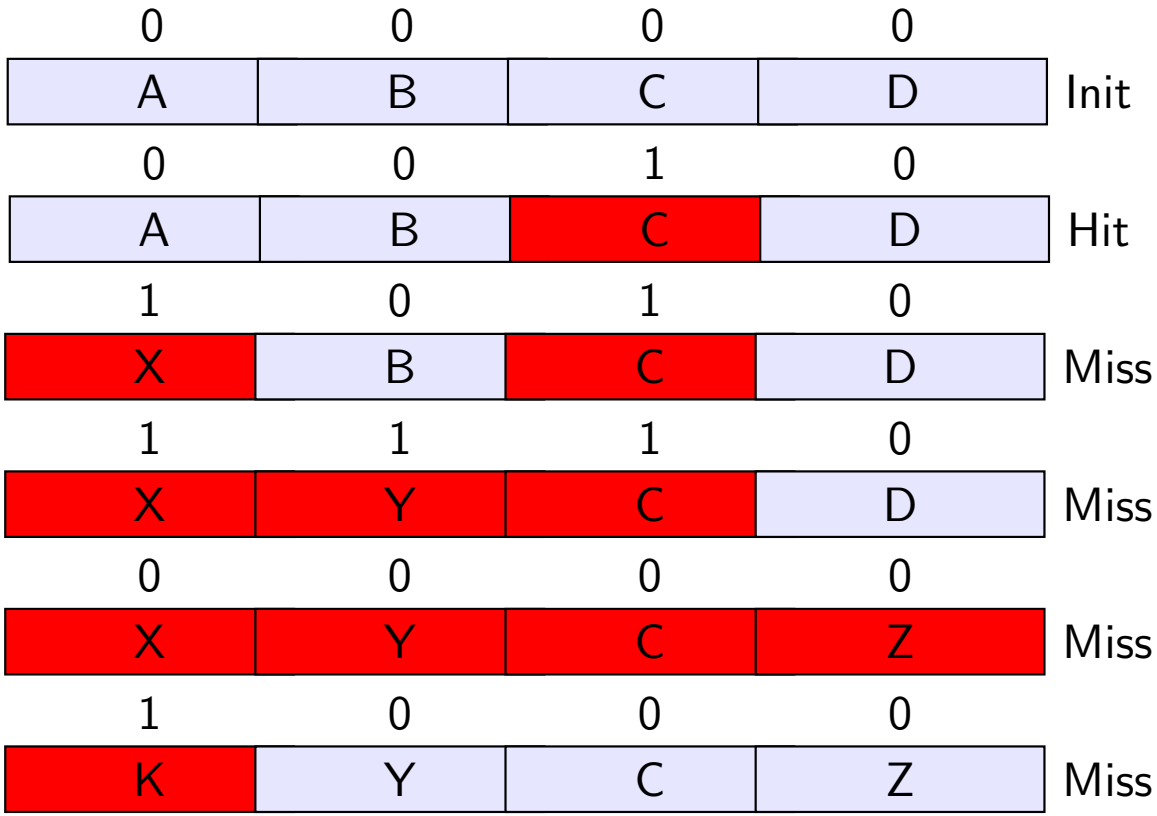
$$
\prod_{miss}^{LRU} = (\boxed{0}; 1; 2; 3; 4; 5; 6; 7)
$$

This vectors are the result of a process

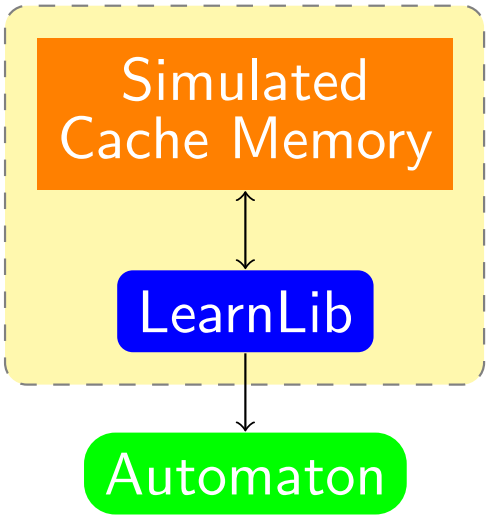
Permutation Vectors: Limitations

- PV offers a limited modelling of the block replacement policies
- For example, MRU can't be modelled with PV
- This Learning process must be changed for another more general

Permutation Vectors: Limitations with MRU

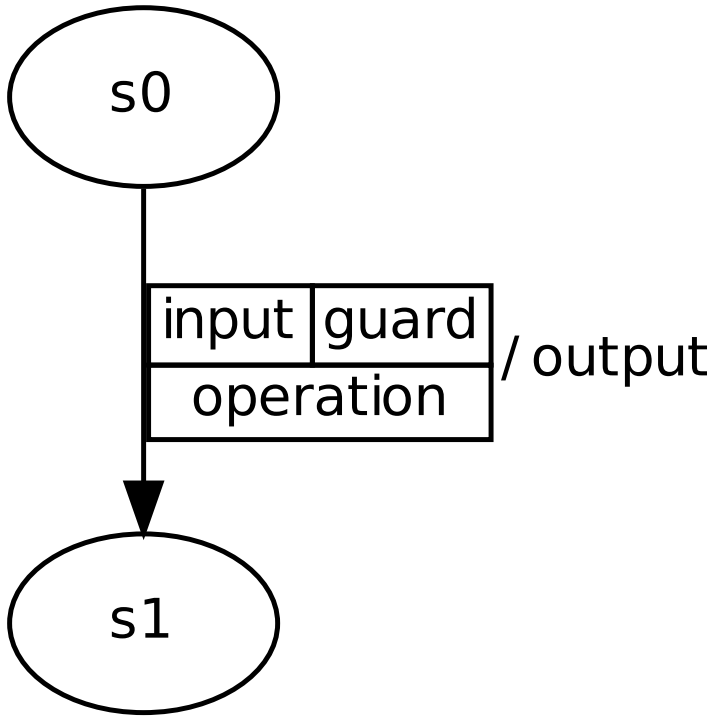


Introduction

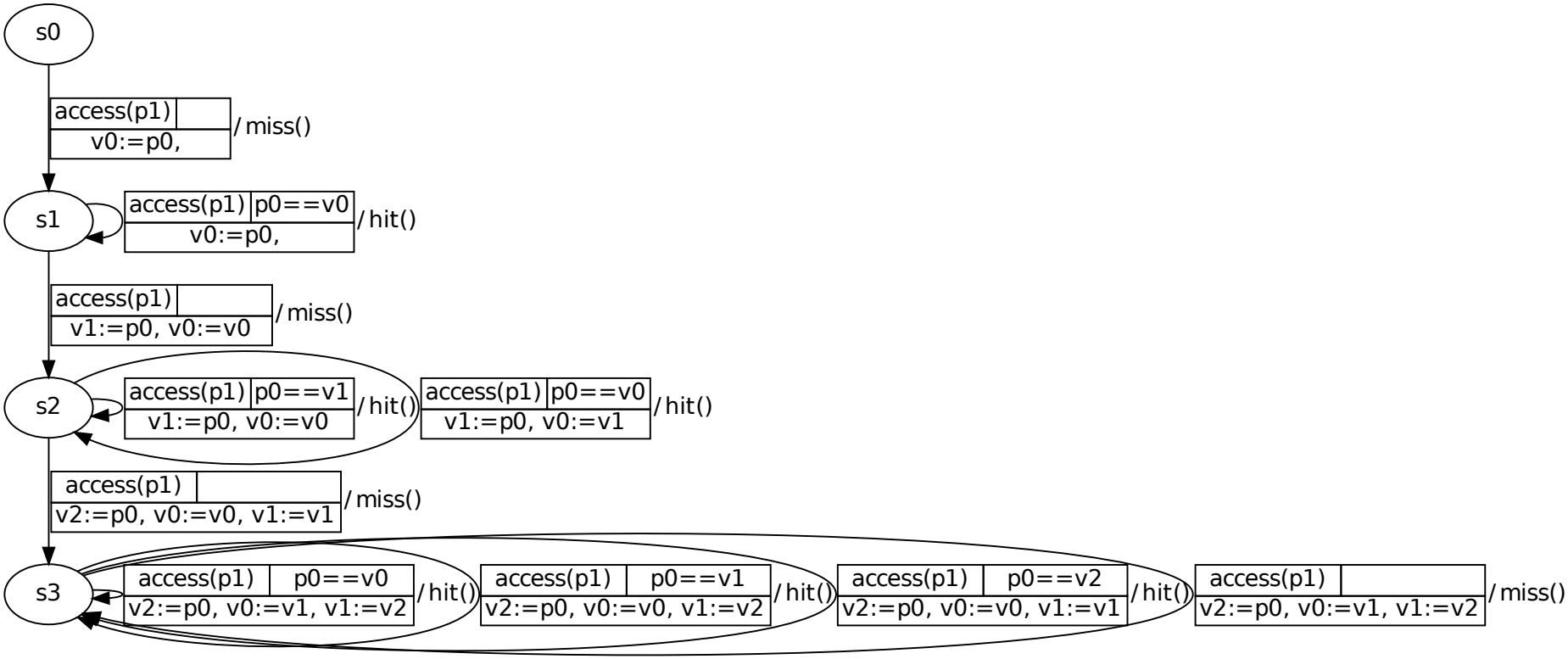


- LearnLib is a JAVA Framework with a Learning process
- It can be used to infer data policies
- The result of inferring is an automaton
- **First part of the project:** simulate different replacement policies

Automaton structure



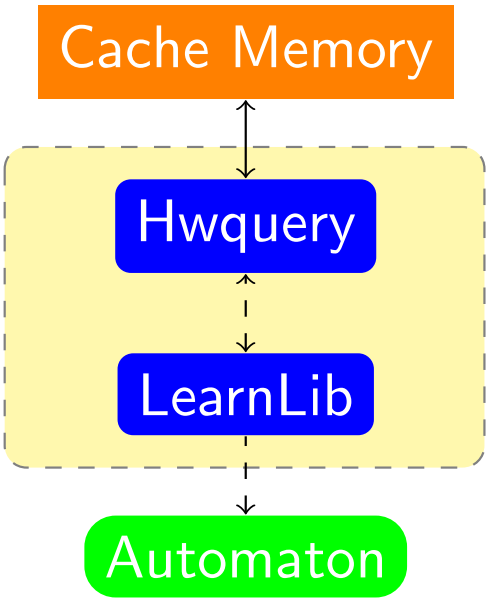
Example of LearnLib output: LRU size 3



Runtimes of simulated policies (in seconds)

Replacment Policy	2 Blocks	3 Blocks	4 Blocks	5 Blocks
FIFO	10	14	25	195
LRU	9	11	17	236
PLRU	9	X	25	X
MRU	9	50	426	Out of Memory

Introduction

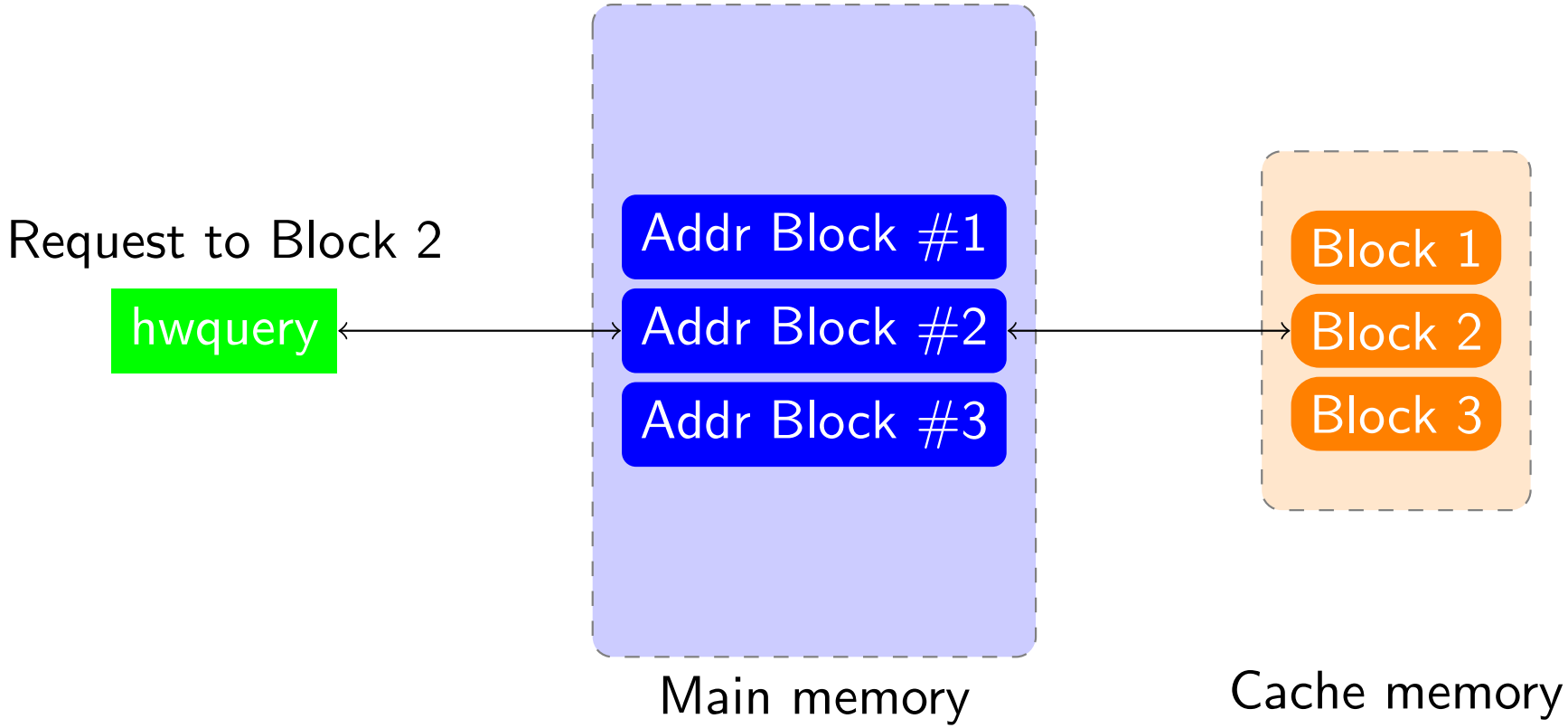


- LearnLib can not interact directly with hardware
- **Second part of the project:** propose to interface queries
- Hwquery is a propose to interact with the cache memory

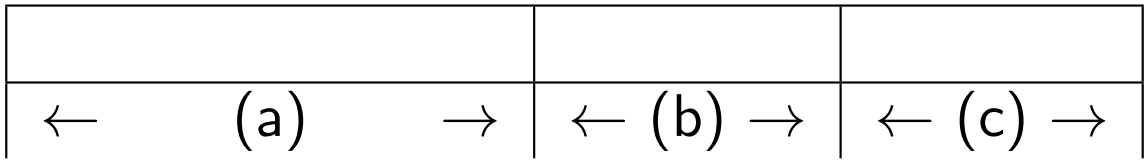
Characteristics

- Input: words containing request to a cache memory blocks
- Developed with C, using malloc to allocate all (main) memory
- The requests to a cache blocks are masked by a main memory address
- Reading of Performance Counter status to know the amount of misses
- Output: word with Hit or Miss of each block request

How it works


Request to Block 2
hwquery
Addr Block #1
Addr Block #2
Addr Block #3
Block 1
Block 2
Block 3
Main memory
Cache memory

Generation of valid cache address



(a) Tag: bits pointing for a block in a set

(b) Index: required bits for addressing the total number of sets

(c) Offset: required bits for addressing a word in the block

Generation of ϕ_{words}

Input word (4th iteration)

2	0	1	0	2	100	3

ϕ_{hit}

2	0	1	0	0

ϕ_{miss}

2	0	1	0	110

ϕ_{test}

2	0	1	0	2

Conclusions

LearnLib:

- Learnlib can represent with an automaton a replacement policy
- MRU cost is exponential

Hwquery:

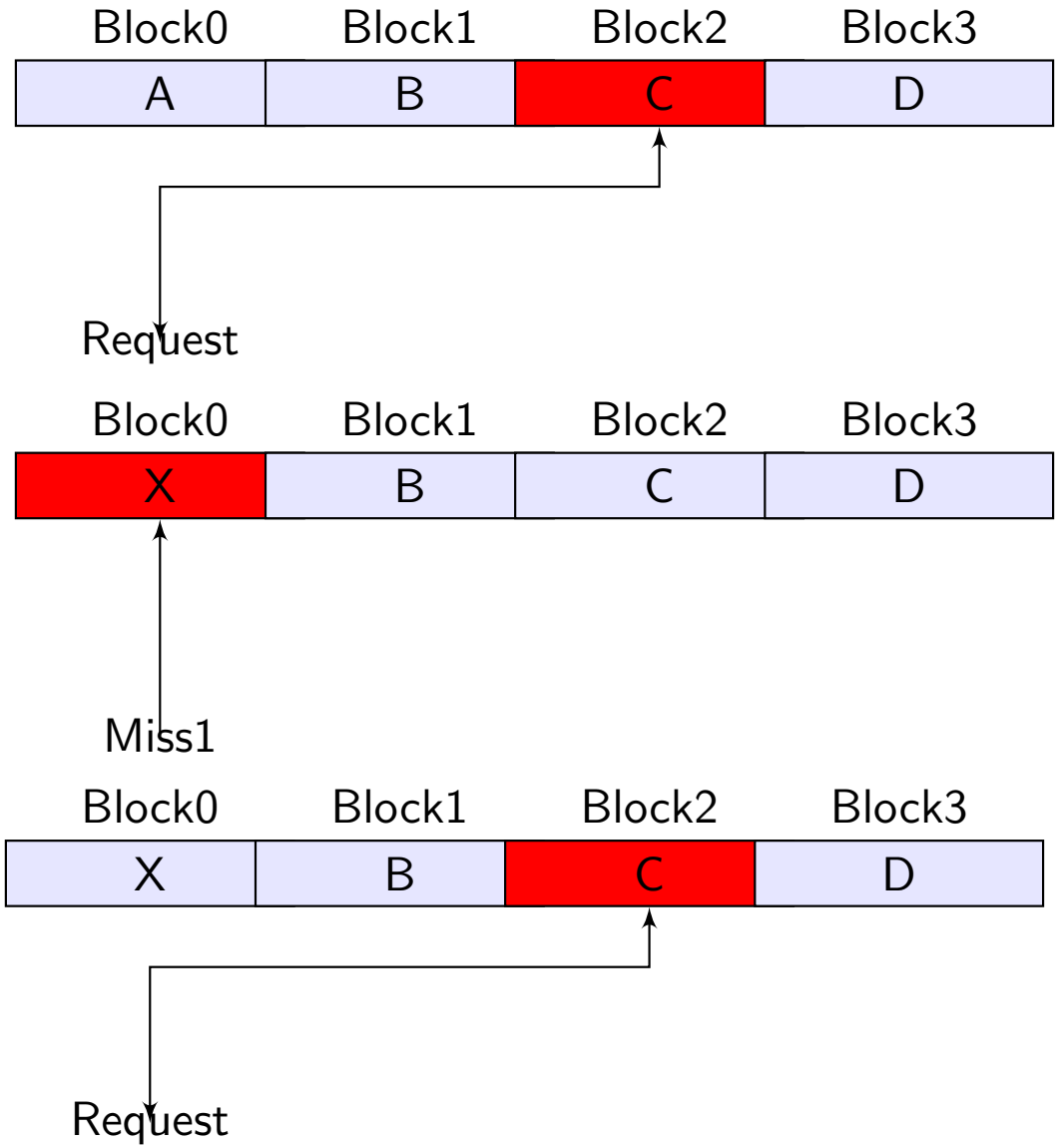
- Hwquery works but must be improved
- Hwquery alternative design to be architecture-independent (set-associative cache memory)

Future work

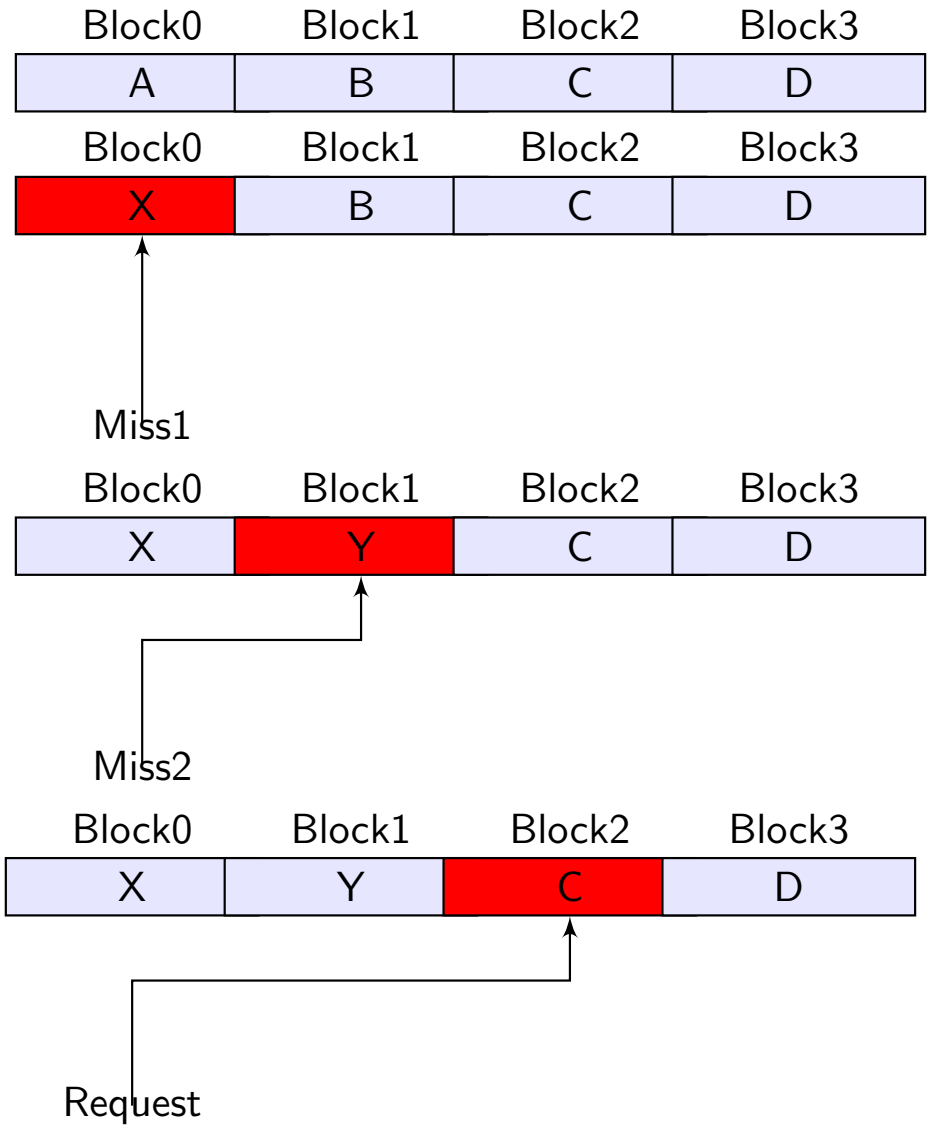
- Interconnect hwquery with LearnLib
- Improve the performance of LearnLib
- Match the replacement policy with a known policies

Thanks for your attention!

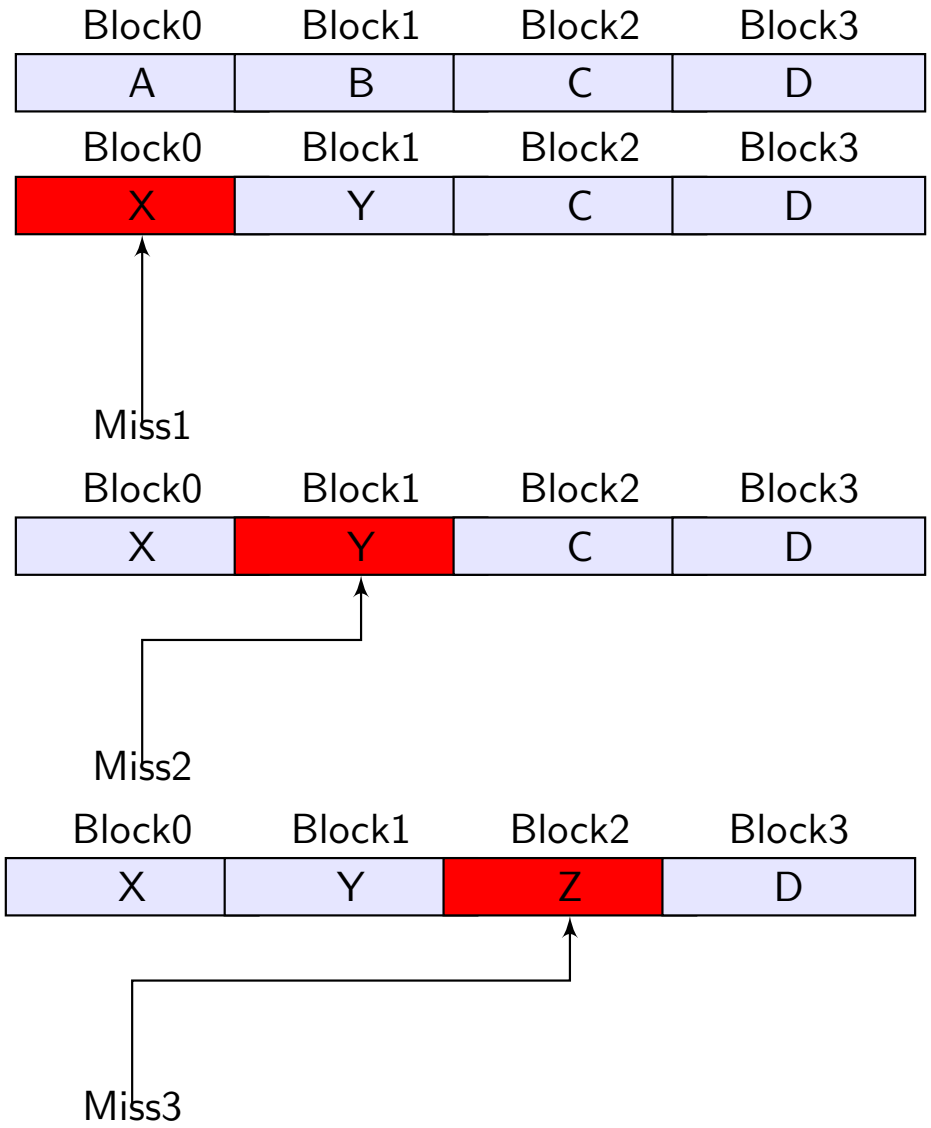
Learning PV


Block0
Block1
Block2
Block3
A
B
C
D
Request
Block0
Block1
Block2
Block3
X
B
C
D
Miss1
Block0
Block1
Block2
Block3
X
B
C
D
Request

Learning PV


Block0
Block1
Block2
Block3
A
B
C
D
Block0
Block1
Block2
Block3
X
B
C
D
Miss1
Block0
Block1
Block2
Block3
X
Y
C
D
Miss2
Block0
Block1
Block2
Block3
X
Y
C
D
Request

Learning PV


Block0
Block1
Block2
Block3
A
B
C
D
Block0
Block1
Block2
Block3
X
Y
C
D
Miss1
Block0
Block1
Block2
Block3
X
Y
C
D
Miss2
Block0
Block1
Block2
Block3
X
Y
Z
D
Miss3