RELOAD+REFRESH: Abusing Cache Replacement Policies to Perform Stealthy Cache Attacks

Samira Briongos[1], Pedro Malagón Marzo[1], José M. Moya Fernández[1], Thomas Eisenbarth[2]

[1]Universidad Politécnica de Madrid, [2]University of Lübeck

UNIVERSIDAD POLITÉCNICA DE MADRID

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR IT SECURITY

Motivation

- **Can we trust existing countermeasures against cache attacks?**
  - Hardware countermeasures
  - Preemptive countermeasures
  - Detection countermeasures

Motivation

- **Can we trust existing countermeasures against cache attacks?**
  - Hardware countermeasures
  - Preemptive countermeasures
  - **Detection countermeasures**

Motivation

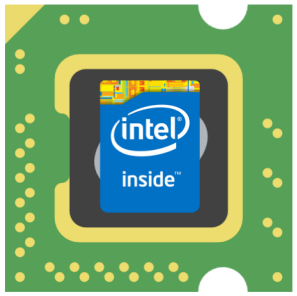
- **Can we trust existing countermeasures against cache attacks?**
  - Hardware countermeasures
  - Preemptive countermeasures
  - **Detection countermeasures**
- **How do they detect cache attacks?**
  - Collect information from performance counters (cache misses, cache accesses…)
  - Compute behavioral models of the process

Motivation

- **Can we trust existing countermeasures against cache attacks?**
  - Hardware countermeasures
  - Preemptive countermeasures
  - **Detection countermeasures**
- **How do they detect cache attacks?**
  - Collect information from performance counters (cache misses, cache accesses…)
  - Compute behavioral models of the process
- **Could an attacker gain information without forcing misses?**
  - It is possible if the eviction order depends on the actual order utilization of the data

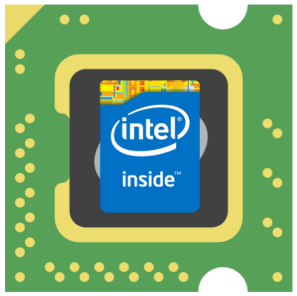
LLC Replacement policy

- The algorithm that decides which element has to be evicted from the cache in case of conflict.
- We focus on the L3 cache
  - Shared
  - Undocumented (Quad-Age LRU)



LLC Replacement policy

- **The algorithm that decides which element has to be evicted from the cache in case of conflict.**
- **We focus on the L3 cache**
  - Shared
  - Undocumented
- **Idea:**
  - We know the location of each block of data in the cache
  - We can enforce the order of the accesses to the data in the cache
  - We can measure access times, i.e. cache hits and misses



LLC Replacement policy: Test

- Manually define a replacement policy to test.
- Generate two eviction sets.
- Fill the cache set with the data in one of the eviction sets.

LLC Replacement policy: Test

- Manually define a replacement policy to test.
- Generate two eviction sets.
- Fill the cache set with the data in one of the eviction sets.

- Randomly access the data previously placed into the cache.

LLC Replacement policy: Test

- Manually define a replacement policy to test.
- Generate two eviction sets.
- Fill the cache set with the data in one of the eviction sets.

- Randomly access the data previously placed into the cache.

- Force a miss by accessing one element in the conflicting set.
- Check if the evicted block corresponds with the block predicted by the test policy.

LLC Replacement policy

- Data is linearly inserted in the cache set.

LLC Replacement policy

- Data is linearly inserted in the cache set.
- Only access to LLC update the ages of the blocks.

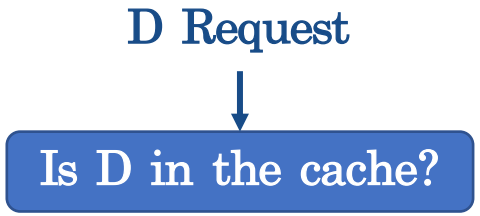
LLC Replacement policy

- Data is linearly inserted in the cache set.
- Only access to LLC update the ages of the blocks.
- Two different policies.
  - Adaptive Replacement Policy (4th and 5th generations).
  - Apparently fixed Replacement Policy (6th ,7th and 8th generations).

LLC Replacement policy

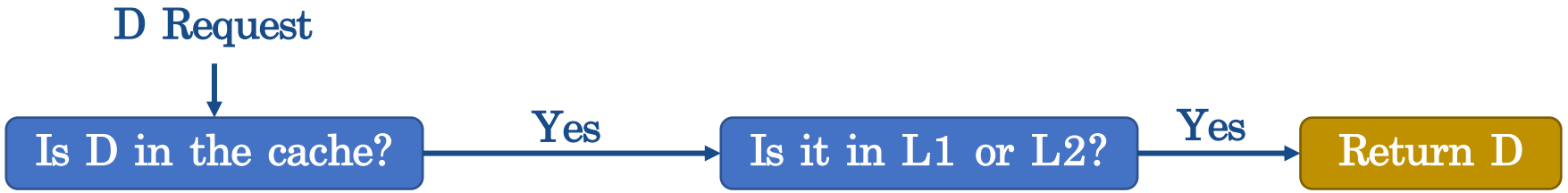
- **Data is linearly inserted in the cache set.**
- **Only access to LLC update the ages of the blocks.**
- **Two different policies.**
  - Adaptive Replacement Policy (4th and 5th generations).
  - Apparently fixed Replacement Policy (6th ,7th and 8th generations).
- **The replacement policy is the same, evict blocks with age 3.**
  - Differ in the insertion age (2 or 3).

LLC Replacement policy

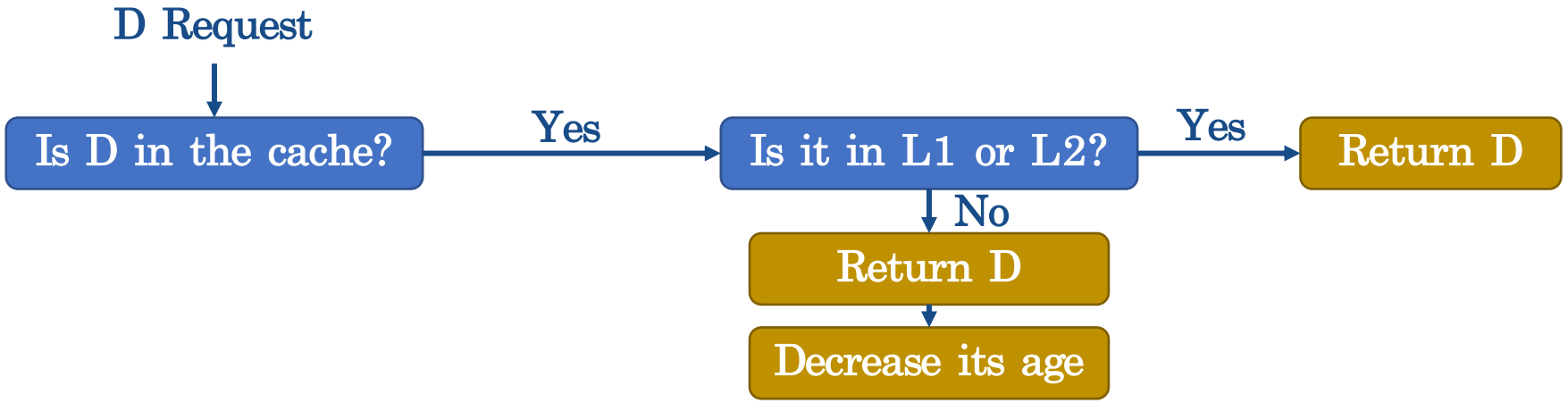


LLC Replacement policy

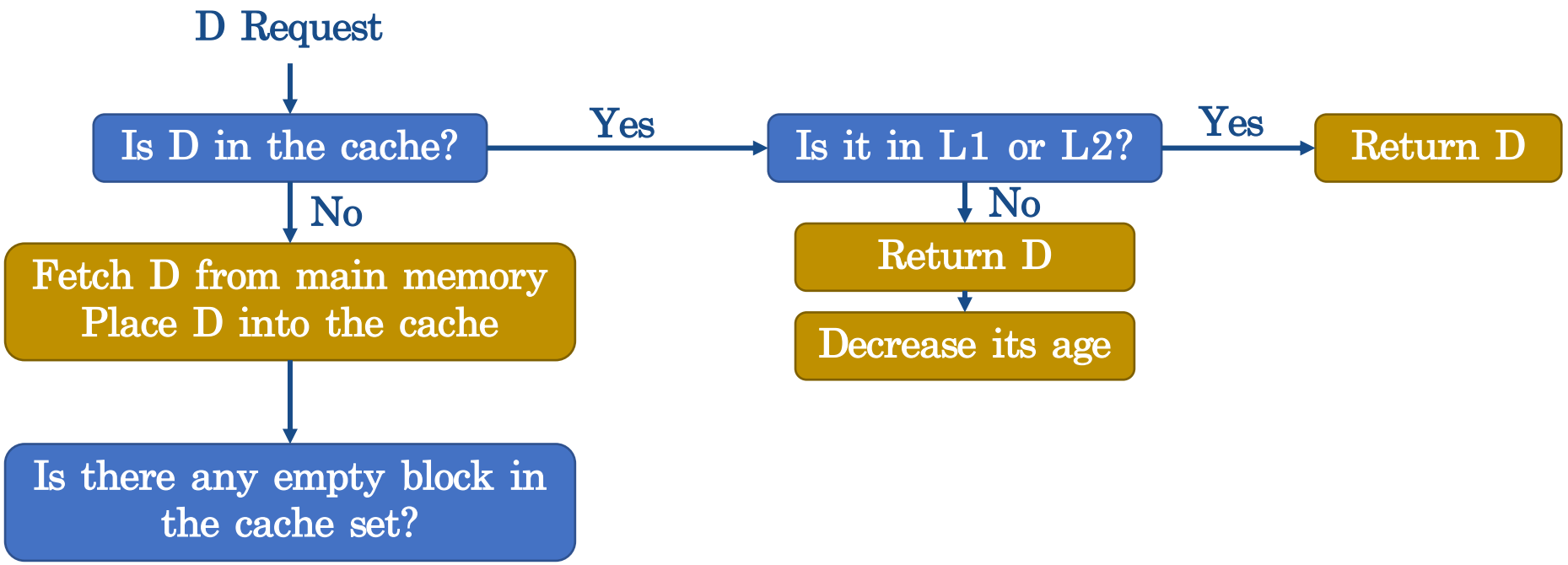
LLC Replacement policy



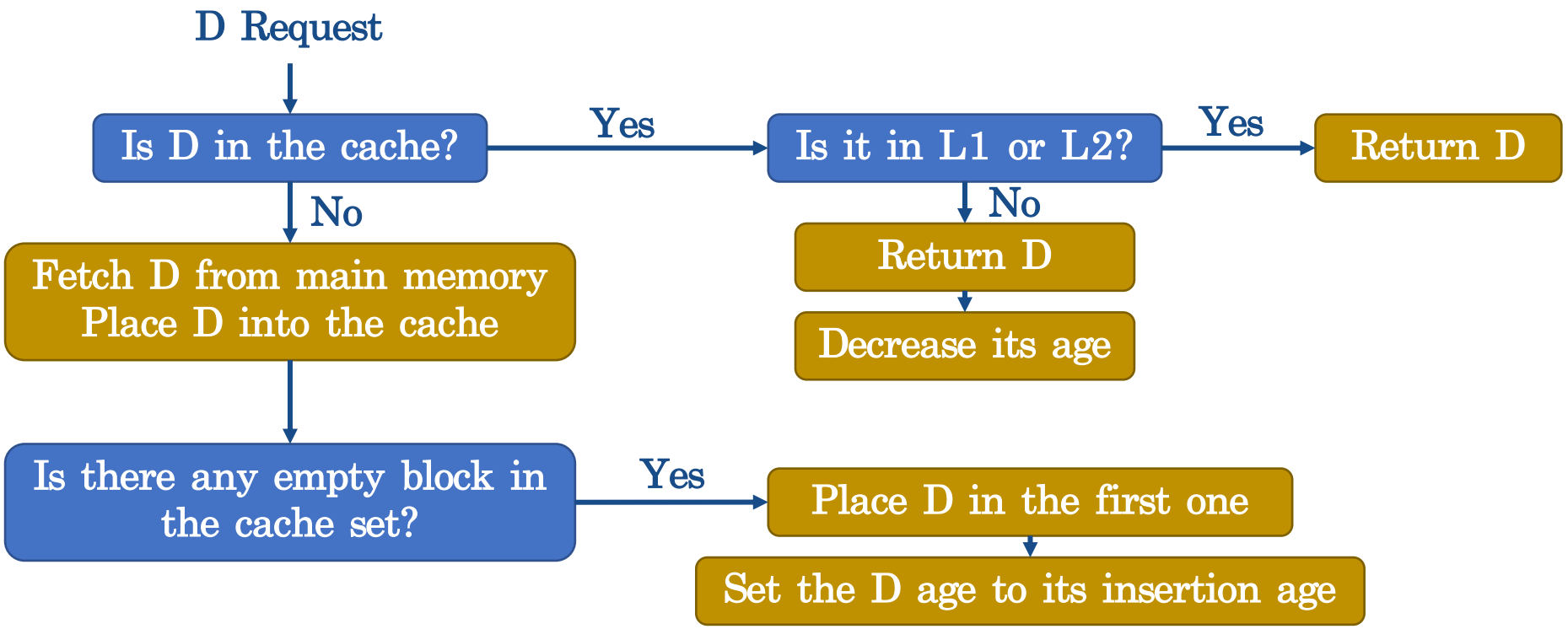
LLC Replacement policy



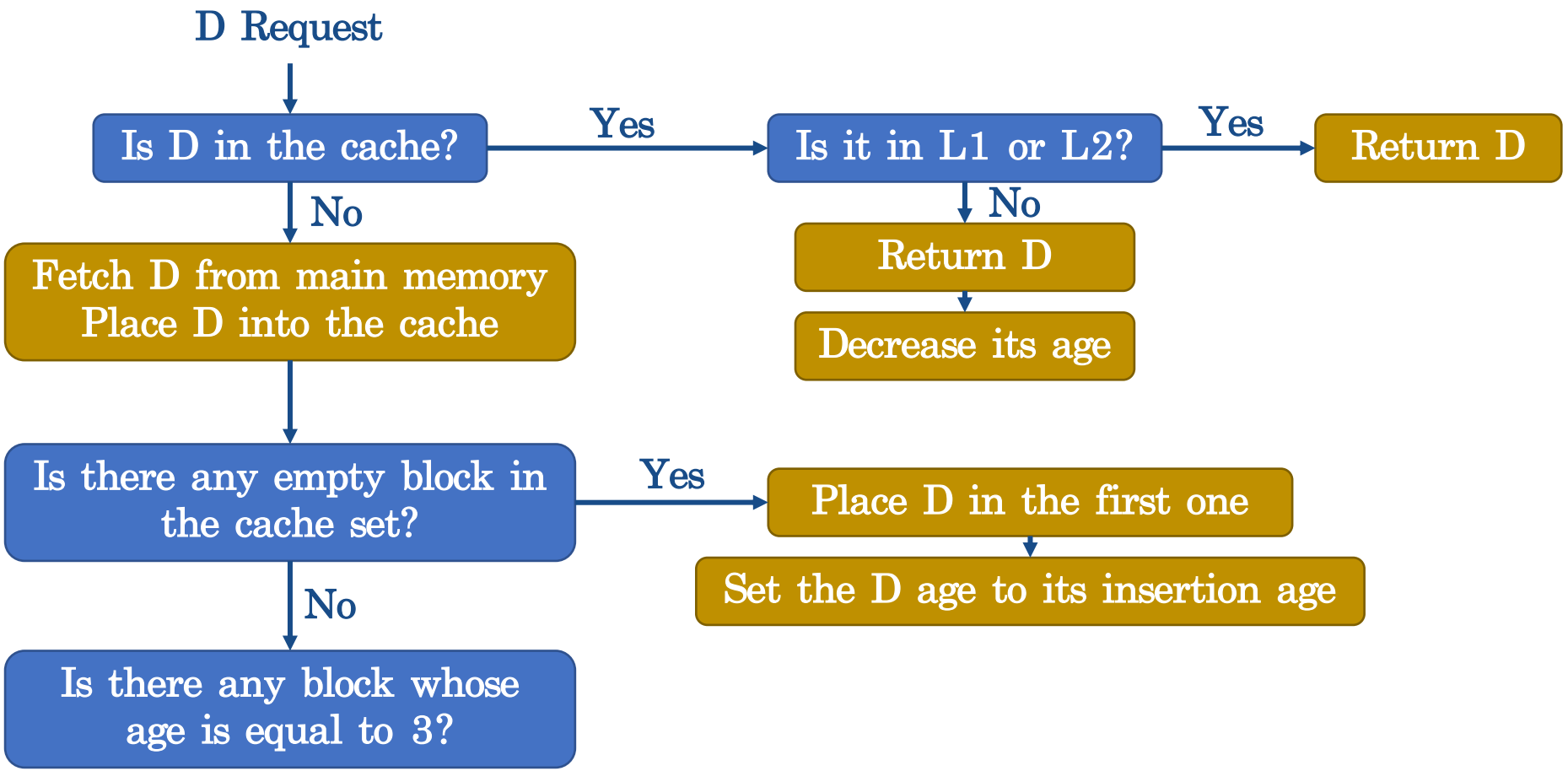
LLC Replacement policy



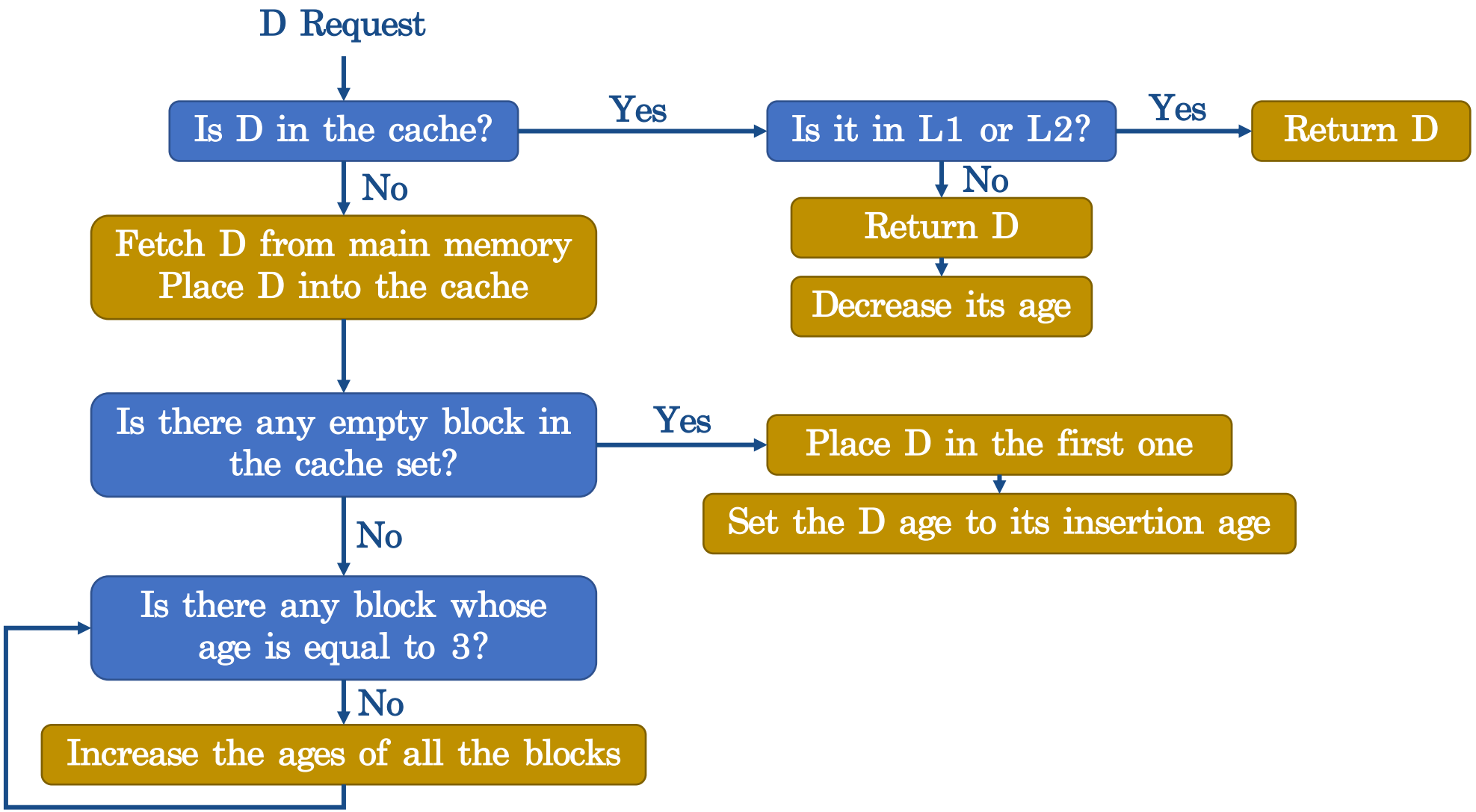
LLC Replacement policy



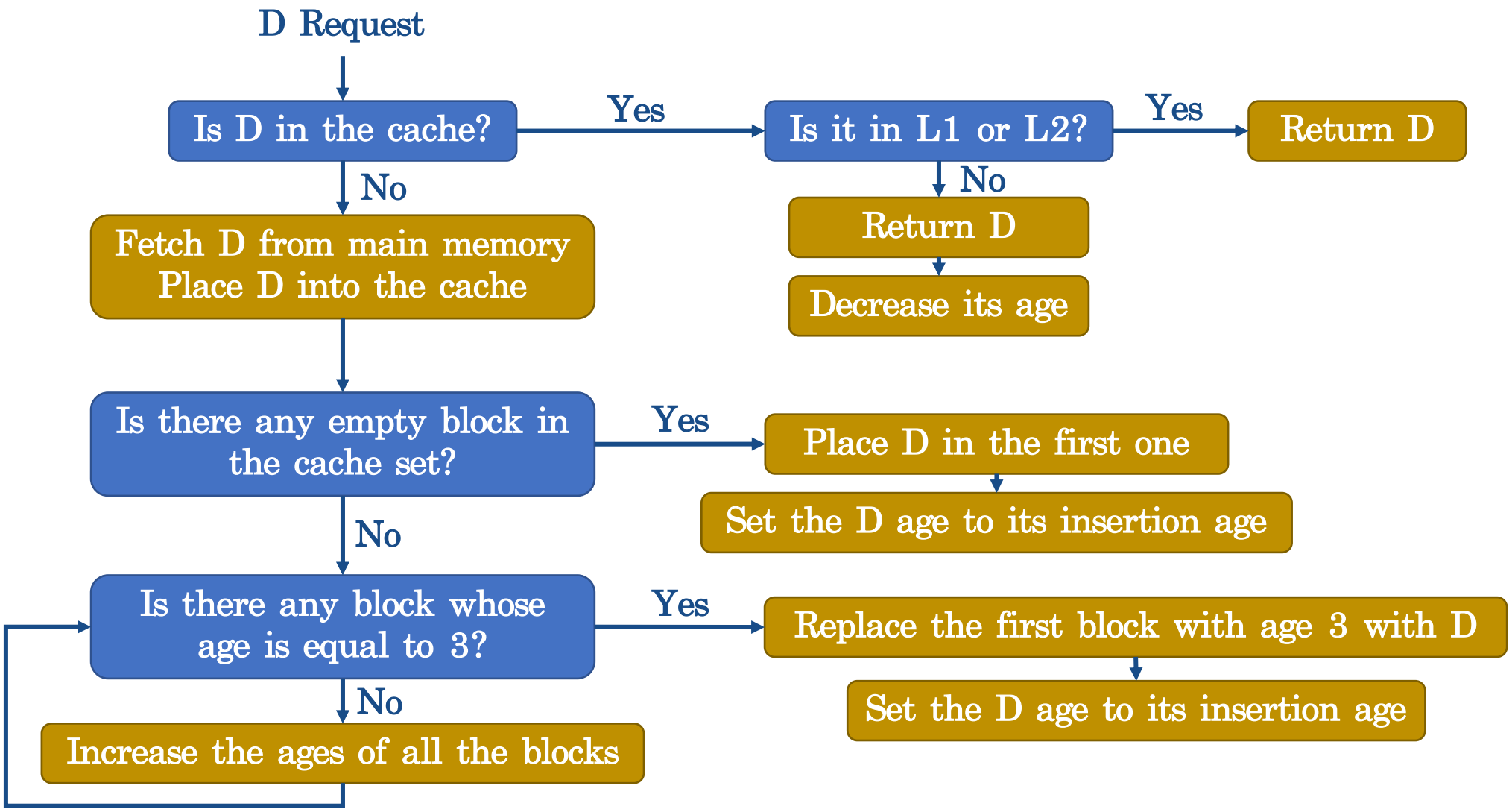
LLC Replacement policy



LLC Replacement policy



LLC Replacement policy



RELOAD+REFRESH

- Retrieve information without forcing misses on the victim

RELOAD+REFRESH

- **Retrieve information without forcing misses on the victim**
- **Abuse the replacement policy:**
  - Place data in the L3 cache in the desired position and set its age
  - Observe changes in the eviction candidate.

RELOAD+REFRESH

- **Retrieve information without forcing misses on the victim**
- **Abuse the replacement policy:**
  - Place data in the L3 cache in the desired position and set its age
  - Observe changes in the eviction candidate.
- **Assume:**
  - There is shared memory
  - The attacker can construct an eviction set
  - The attacker can ensure the order of the accesses

RELOAD+REFRESH: stages

1. Prepare the cache

Candidate

Content	Target							
Age	2							

RELOAD+REFRESH: stages

1. Prepare the cache

Candidate

Content	Target	A	B	C	D	E	F	G
Age	2	2	2	2	2	2	2	2

RELOAD+REFRESH: stages

2. Wait for the victim to access the data

Candidate

Content	Target	A	B	C	D	E	F	G
Age	2	2	2	2	2	2	2	2

RELOAD+REFRESH: stages

2. Wait for the victim to access the data

Candidate

Content	Target	A	B	C	D	E	F	G
Age	1	2	2	2	2	2	2	2

ACCESS

RELOAD+REFRESH: stages

2. Wait for the victim to access the data

Candidate

Content	Target	A	B	C	D	E	F	G
Age	1	2	2	2	2	2	2	2

ACCESS

RELOAD+REFRESH: stages

2. Wait for the victim to access the data

Candidate

Content	Target	A	B	C	D	E	F	G
Age	1	2	2	2	2	2	2	2

ACCESS

Candidate

Content	Target	A	B	C	D	E	F	G
Age	2	2	2	2	2	2	2	2

NO ACCESS

RELOAD+REFRESH: stages

3. Retrieve the information (Force miss + Reload)

Candidate

Content	Target	A	B	C	D	E	F	G
Age	1	2	2	2	2	2	2	2

ACCESS

Candidate

Content	Target	A	B	C	D	E	F	G
Age	2	2	2	2	2	2	2	2

NO ACCESS

RELOAD+REFRESH: stages

3. Retrieve the information (Force miss + Reload)

Candidate

Content	Target	A	B	C	D	E	F	G	ACCESS
Age	2	3	3	3	3	3	3	3	

Candidate

Content	Target	A	B	C	D	E	F	G	NO ACCESS
Age	3	3	3	3	3	3	3	3	

RELOAD+REFRESH: stages

3. Retrieve the information (Force miss + Reload)

Candidate

Content	Target	H	B	C	D	E	F	G	ACCESS
Age	2	2	3	3	3	3	3	3	

Candidate

Content	H	A	B	C	D	E	F	G	NO ACCESS
Age	2	3	3	3	3	3	3	3	

RELOAD+REFRESH: stages

3. Retrieve the information (Force miss + Reload)

Candidate

Content	Target	H	B	C	D	E	F	G	ACCESS
Age	2	2	3	3	3	3	3	3	

Candidate

Content	H	A	B	C	D	E	F	G	NO ACCESS
Age	2	3	3	3	3	3	3	3	

RELOAD+REFRESH: stages

3. Retrieve the information (Force miss + Reload)

Candidate

Content	Target	H	B	C	D	E	F	G	ACCESS
Age	1	2	3	3	3	3	3	3	

Candidate

Content	H	Target	B	C	D	E	F	G	NO ACCESS
Age	2	2	3	3	3	3	3	3	

RELOAD+REFRESH: stages

3. Retrieve the information (Force miss + Reload)

Candidate

Content	Target	H	B	C	D	E	F	G	ACCESS
Age	1	2	3	3	3	3	3	3	

Candidate

Content	H	Target	B	C	D	E	F	G	NO ACCESS
Age	2	2	3	3	3	3	3	3	

RELOAD+REFRESH: stages

4. Refresh the cache set state

Candidate

Content	Target	H	B	C	D	E	F	G
Age	1	2	3	3	3	3	3	3

ACCESS

Candidate

Content	H	Target	B	C	D	E	F	G
Age	2	2	3	3	3	3	3	3

NO ACCESS

RELOAD+REFRESH: stages

4. Refresh the cache set state

Candidate

Content			B	C	D	E	F	G	ACCESS
Age			3	3	3	3	3	3	

Candidate

Content			B	C	D	E	F	G	NO ACCESS
Age			3	3	3	3	3	3	

RELOAD+REFRESH: stages

4. Refresh the cache set state

Candidate

Content	Target		B	C	D	E	F	G	ACCESS
Age	2		3	3	3	3	3	3	

Candidate

Content	Target		B	C	D	E	F	G	NO ACCESS
Age	2		3	3	3	3	3	3	

RELOAD+REFRESH: stages

4. Refresh the cache set state

Candidate

Content	Target	A	B	C	D	E	F	G
Age	2	2	3	3	3	3	3	3

ACCESS

Candidate

Content	Target	A	B	C	D	E	F	G
Age	2	2	3	3	3	3	3	3

NO ACCESS

RELOAD+REFRESH: stages

4. Refresh the cache set state

Candidate

Content	Target	A	B	C	D	E	F	G
Age	2	2	2	2	2	2	2	2

ACCESS

Candidate

Content	Target	A	B	C	D	E	F	G
Age	2	2	2	2	2	2	2	2

NO ACCESS

RELOAD+REFRESH: stages

4. Refresh the cache set state

Candidate

Content	Target	A	B	C	D	E	F	G
Age	2	2	2	2	2	2	2	2

ACCESS

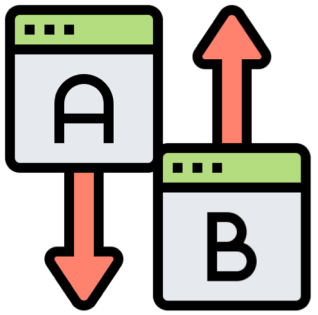
Candidate

Content	Target	A	B	C	D	E	F	G
Age	2	2	2	2	2	2	2	2

NO ACCESS

RELOAD+REFRESH: Evaluation

- **Covert channel**
  - Similar performance to existing approaches.
  - Slightly lower resolution



F-Score for the different approaches: Reload+Refresh(R+R), Flush+Reload(F+R) and Prime+Probe (P+P).

	Window length (ns)					
	50000	10000	1000	750	500	250
R+R	0.988	0.975	0.925	0.684	-	-
F+R	0.999	0.995	0.996	0.991	0.989	0.981
P+P	0.934	0.934	0873	0.716	0.548	-

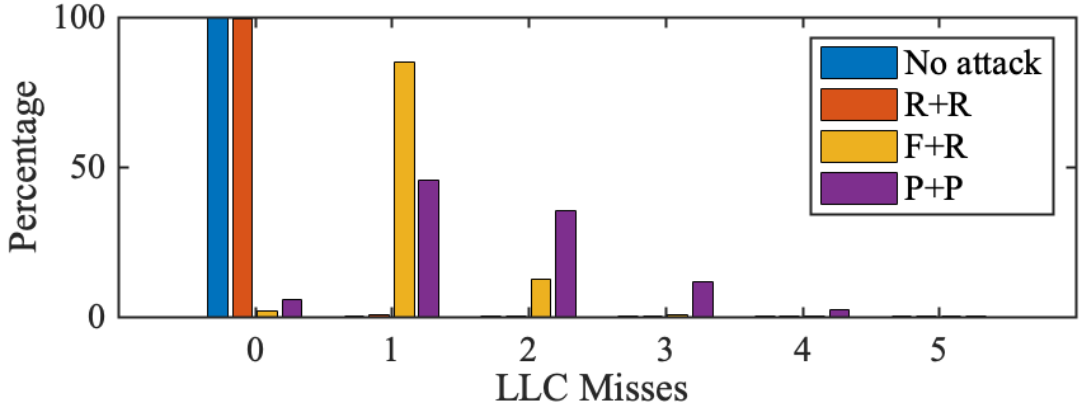
RELOAD+REFRESH: Evaluation

- AES
  - Retrieve the secret key

Attack	R+R	F+R	P+P
Samples	3800	3500	3900
F-Score	0.98	0.99	0.97

Reload+Refresh (R+R)
Flush+Reload (F+R)
Prime+Probe (P+P)

  - No cache misses



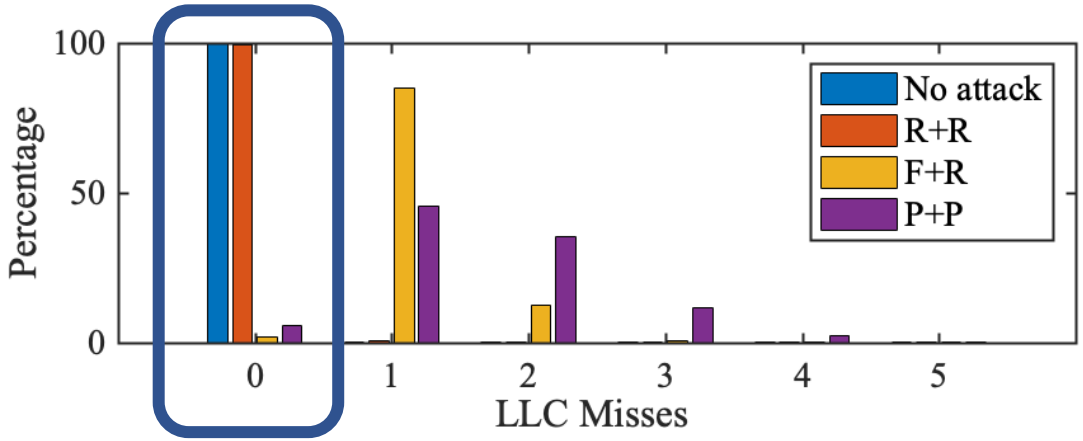
RELOAD+REFRESH: Evaluation

- AES
  - Retrieve the secret key

Attack	R+R	F+R	P+P
Samples	3800	3500	3900
F-Score	0.98	0.99	0.97

Reload+Refresh (R+R)
Flush+Reload (F+R)
Prime+Probe (P+P)

  - No cache misses



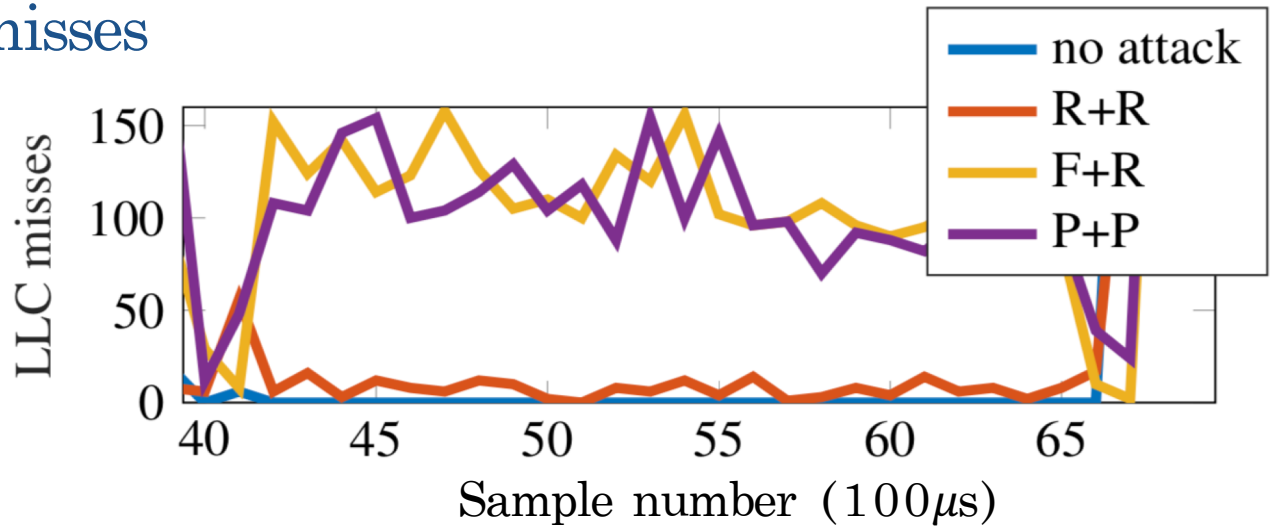
RELOAD+REFRESH: Evaluation

- RSA
  - Multiply operations

Attack	R+R	F+R	P+P
Accuracy	96.1%	98.6%	95.4%
F-Score	0.952	0.99	0.945

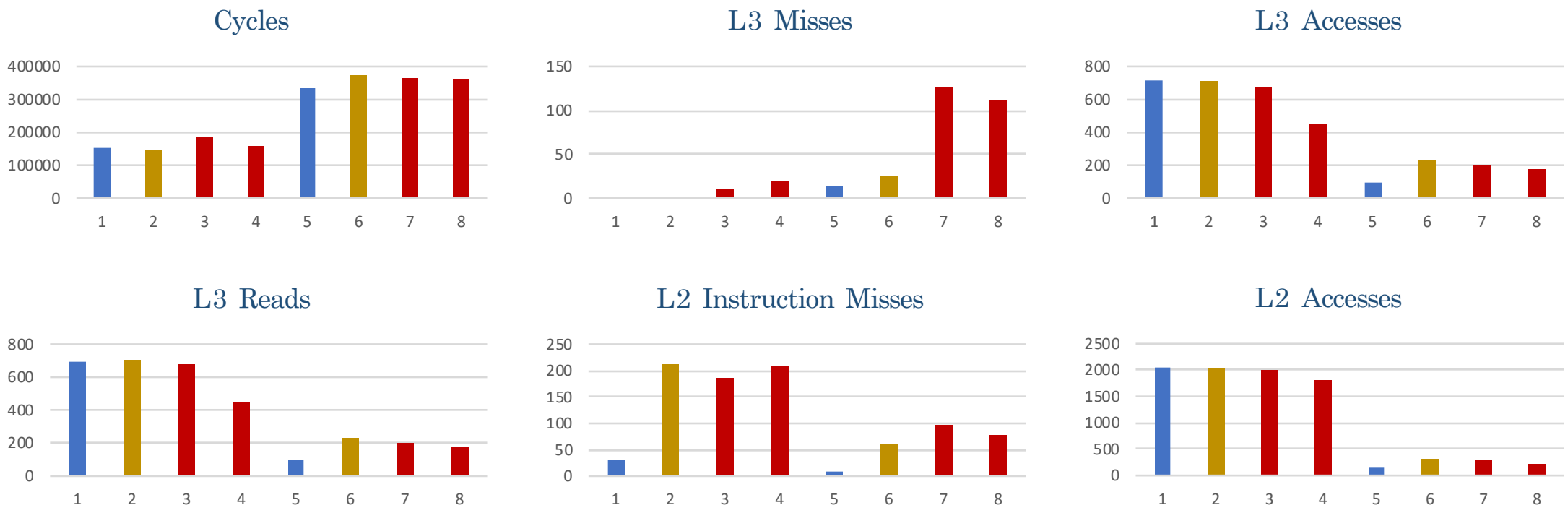
Reload+Refresh (R+R)
Flush+Reload (F+R)
Prime+Probe (P+P)

  - No cache misses



RELOAD+REFRESH: Detection

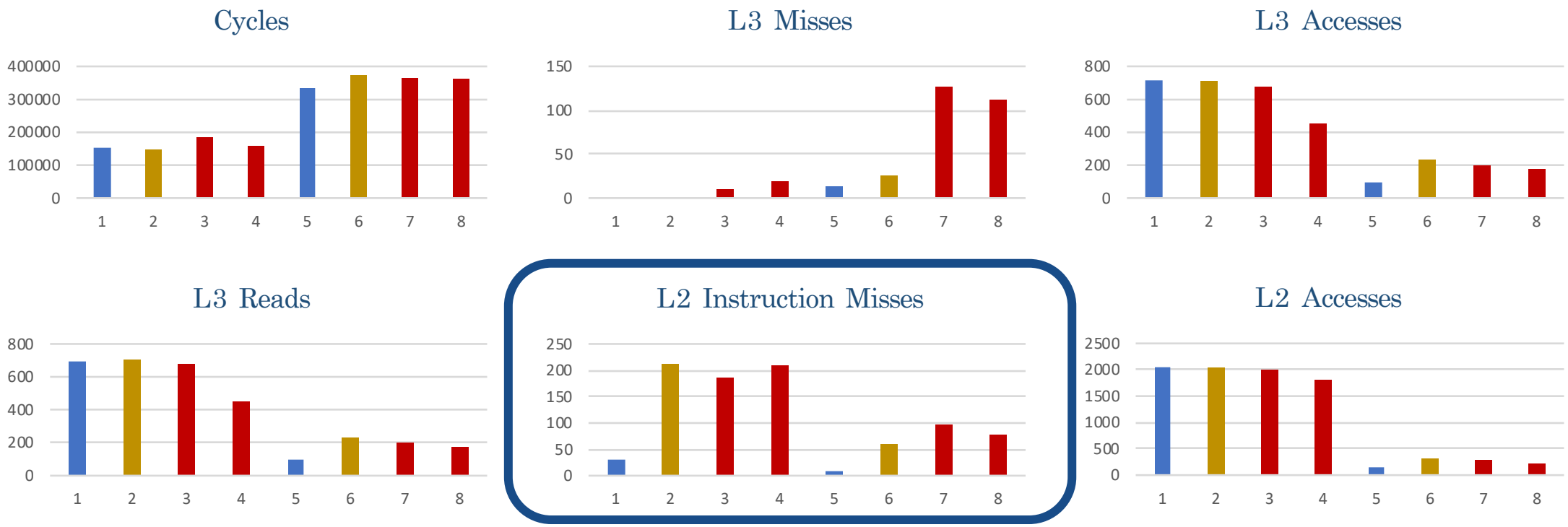
- Results of monitoring different performance counters



1-AES Normal 2-AES R+R 3-AES F+R 4-AES P+P 5-RSA Normal 6-RSA R+R 7-RSA F+R 8-RSA P+P

RELOAD+REFRESH: Detection

- Results of monitoring different performance counters



1-AES Normal 2-AES R+R 3-AES F+R 4-AES P+P 5-RSA Normal 6-RSA R+R 7-RSA F+R 8-RSA P+P

Conclusions

- **Show an important limitation in detection algorithms**
  - The number of counters is limited
  - RELOAD+REFRESH hides misses from the monitor
  - Cat and mouse game

Conclusions

- **Show an important limitation in detection algorithms**
  - The number of counters is limited
  - RELOAD+REFRESH hides misses from the monitor
  - Cat and mouse game
- **Provide further insights into the cache architecture**
  - Reverse-engineer the replacement policy
  - Other cache attacks could benefit from this knowledge

Thank you for your attention

Q&A

Contact information: samirabriongos@die.upm.es
Sample code: https://github.com/greenlsi/reload_refresh