Eviction Set-Finding on Randomized Caches with Reinforcement Learning

Mulong Luo and Mohit Tiwari
The University of Texas at Austin
mulong@utexas.edu

Outline

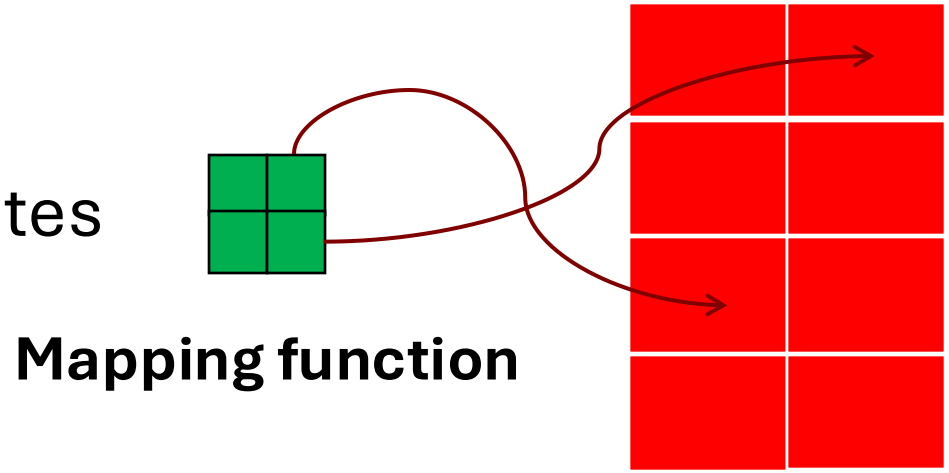
- Background and Motivation
- Challenges
- RL Formulation
- Evaluation
- Conclusion and Future Work

Contention-based Cache Timing Channel and Cache Randomization

- Eviction set and contention-based timing channel
  - A collection of addresses map to the same cache set
  - Accesses some of these addresses may evict a victim address that maps to the same cache set
  - Access pattern leaked by the timing measurement
- Cache randomization
  - There exist a mapping function: addresses → cache set/locations
    - for non-randomized caches, mapping function is trivial
    - For randomized caches, mapping function is not visible to the programs

Cache Timing Attacks Arms Race

- Randomization makes it hard to find eviction set (more steps needed)
  - CEASE(R-s)
  - ScatterCache
  - RPCache
  - NewCache
- More powerful eviction set-finding invalidates randomization defense



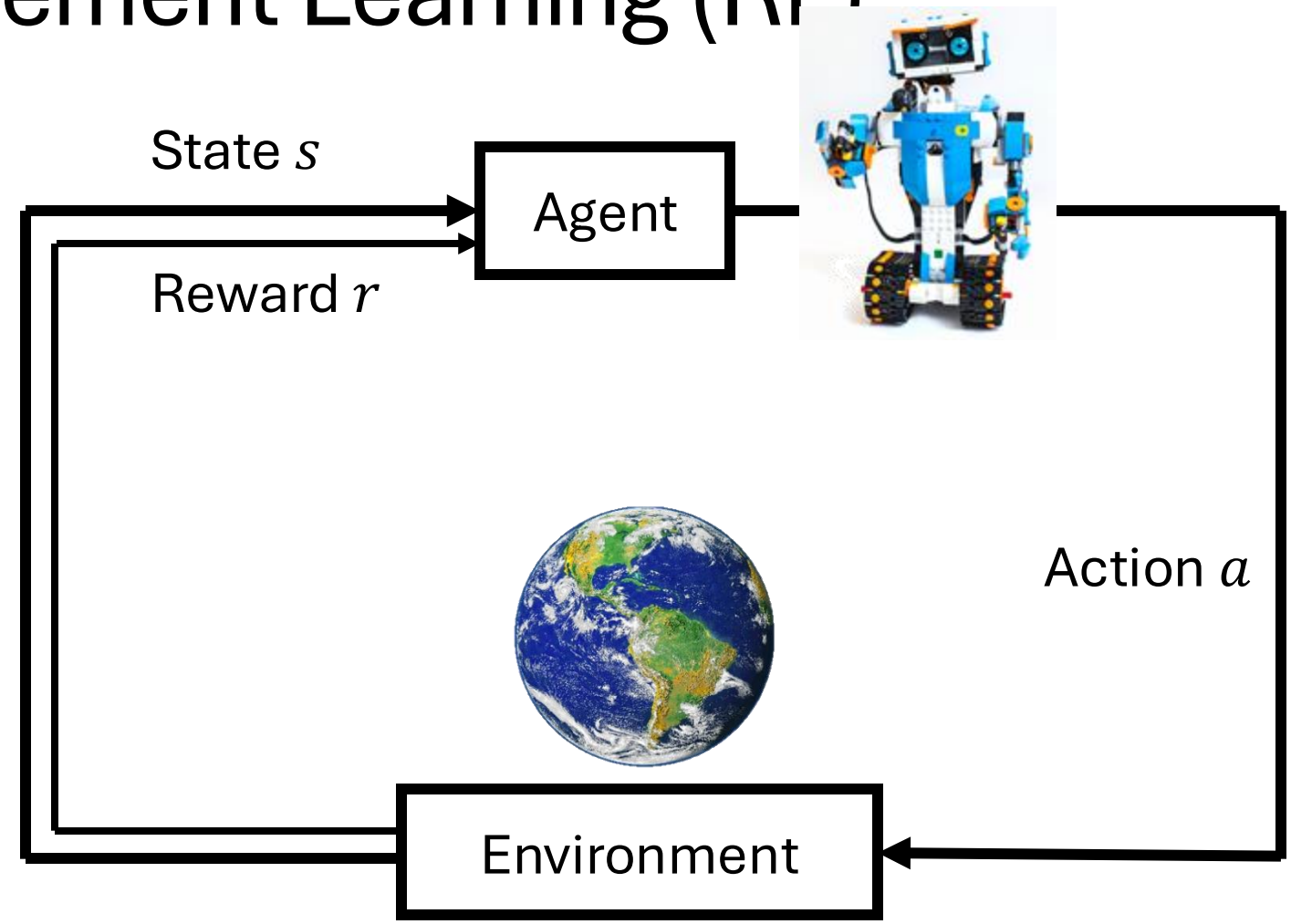
The Evolution of Eviction Set-Finding Algorithms

- Finding S addresses out of A addresses that cause eviction
  - Single holdout method[1] (SHM)
  - Group elimination[2,3] (GEM)
  - Prime+Prune+Probe[4] (PPP)
  - Exploiting cache hierarchy[5]

Remark: Eviction Set-Finding Algorithms are key to enable stronger cache timing attacks

1.Last-Level Cache Side-Channel Attacks are Practical, Liu, Oakland, 2015 2. New Attacks and Defense for Encrypted-Address Cache, Qureshi, ISCA, 2019 3. Theory and Practice of Finding Eviction Sets, Vila, Oakland, 2019 4. Systematic Analysis of Randomization-Based Protected Cache Architectures. Purnal, et. al. 2021 5. Last-Level Cache Side-Channel Attacks Are Feasible in the Modern Public Cloud, Zhao, et al., ASPLOS, 2024

Reinforcement Learning (RL)



RL for Games

Go

Chess

Shogi

Poker

DoTA 2

StarCraft II

Big Success in Games

(courtesy of Dr. Yuandong Tian)

RL for Algorithm Development

AlphaTensor (DeepMind)

$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$

$m_1 = (a_1 + a_4)(b_1 + b_4)$

$m_2 = (a_3 + a_4)b_1$

$m_3 = a_1(b_2 - b_4)$

$m_4 = a_4(b_3 - b_1)$

$m_5 = (a_1 + a_2)b_4$

$m_6 = (a_3 - a_1)(b_1 + b_2)$

$m_7 = (a_2 - a_4)(b_3 + b_4)$

$c_1 = m_1 + m_4 - m_5 + m_7$ $\quad c_3 = m_2 + m_4$

$c_2 = m_3 + m_5$ $\quad c_4 = m_1 - m_2 + m_3 + m_6$

AlphaDev (DeepMind)

mov	R	S
cmp	P	R
cmovl	P	S
~~mov~~	~~S~~	~~P~~
cmp	S	Q
cmovg	Q	P
cmovg	S	Q

A. Fawzi et al, *Discovering faster matrix multiplication algorithms with reinforcement learning*, Nature'22

(Adding one low-rank approximation at a time)

D. Mankowit et al, *Faster sorting algorithms discovered using deep reinforcement learning*, Nature'23

(Adding one instruction at a time)

(courtesy of Dr. Yuandong Tian)

Motivation

- For better/stronger attacks, we need efficient algorithms for eviction set-finding
- RL is capable of finding more efficient algorithms

Research Question: Can we use RL to find algorithms for eviction set finding?

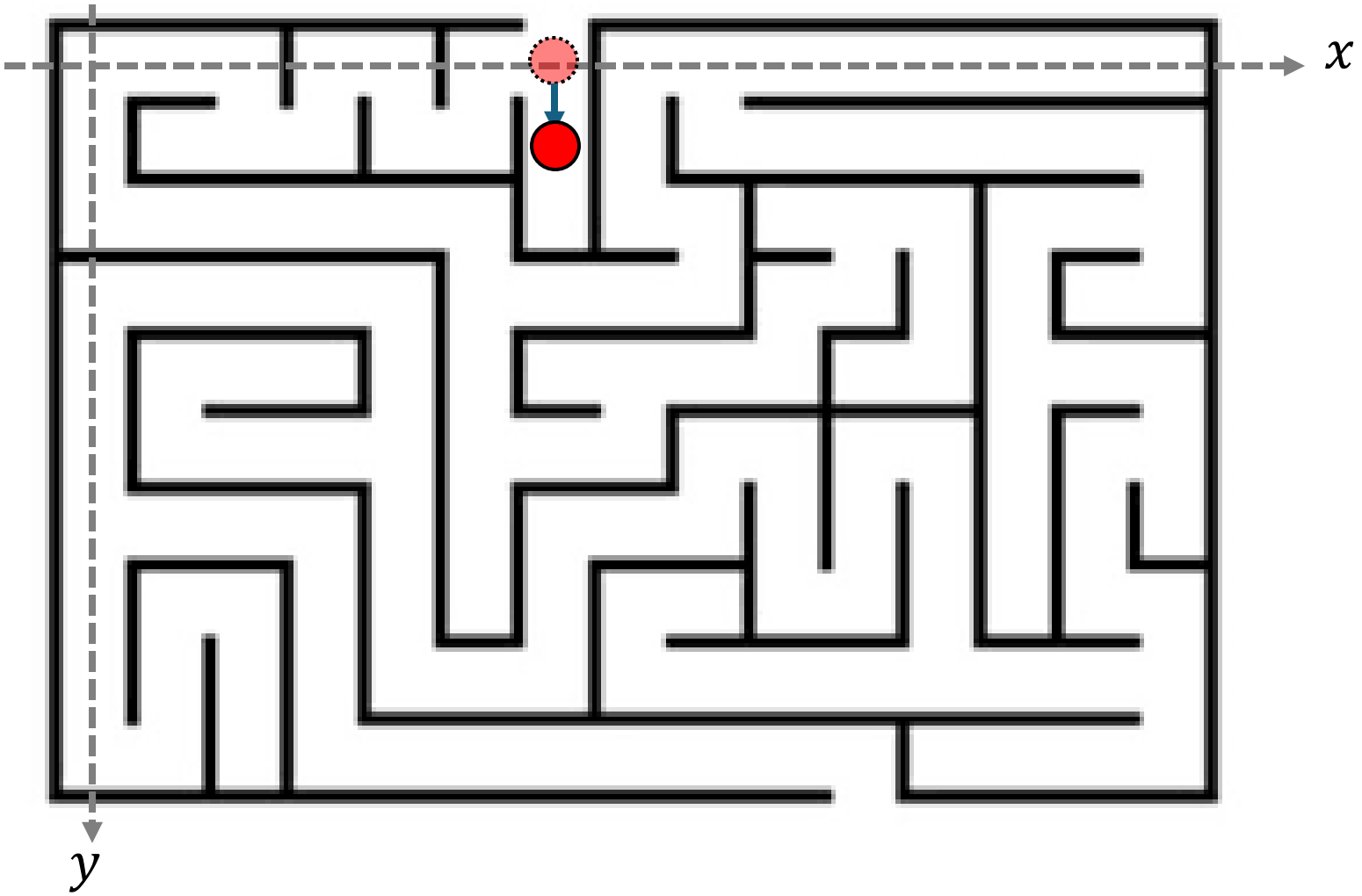
Outline

- Background and Motivation
- Challenges
- RL Formulation
- Evaluation
- Conclusion and Future Work

Example Problems RL Solves

- Maze Solving
- Finding Cache Timing Attacks

Maze Solving with RL



State:

$s = (x, y) = (6,0)$

Actions:

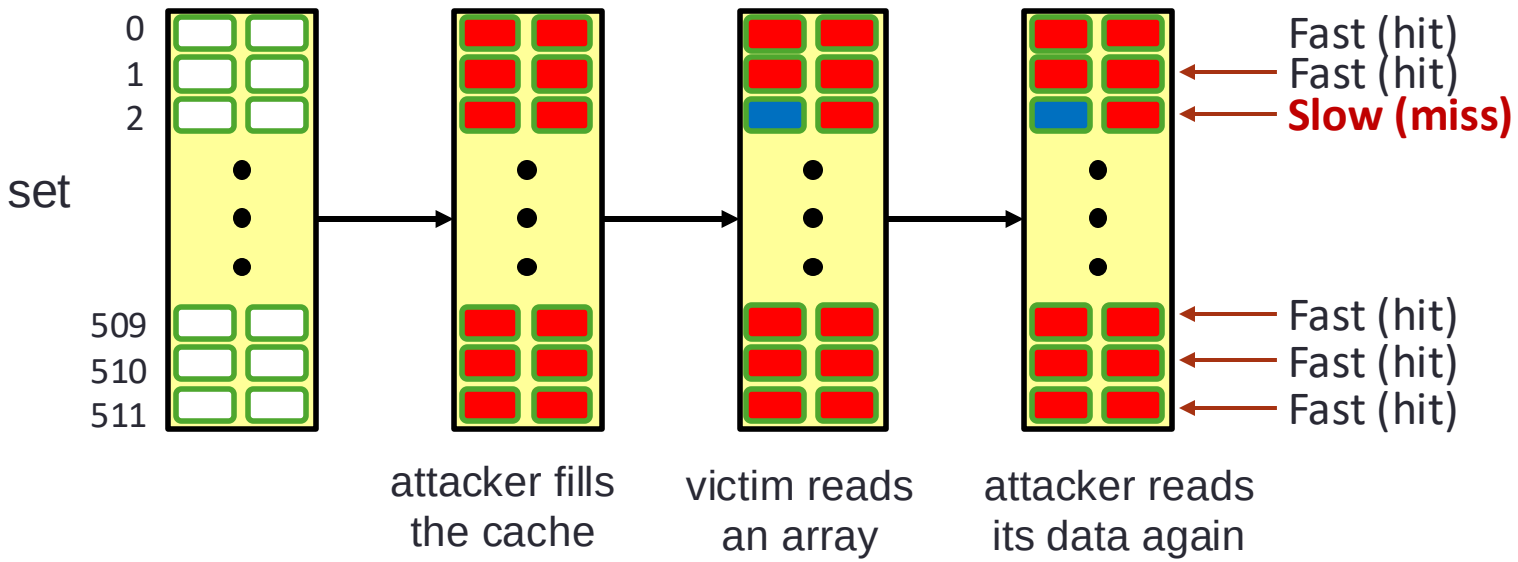
Left: $x \leftarrow x - 1$

Right: $x \leftarrow x + 1$

Up: $y \leftarrow y - 1$

Down: $y \leftarrow y + 1$

AutoCAT: RL for Attack on Non-Randomized Cache



- Agent: Attacker
- Environment: Cache
- Actions
  - attacker makes an access
  - **attacker waits for victim access**
  - attacker guesses the secret
- Observation
  - **latency of attacker accesses**
- Reward
  - guess correct: positive reward
  - guess wrong: negative reward
  - **each step: small negative reward**

AutoCAT Limitation

- Only works for non-randomized caches
  - Agent trained for one-randomization mapping does not work for other mapping functions

Challenge : needs a way to incorporate different mapping functions in a randomized cache

Outline

- Background and Motivation
- Challenges
- RL Formulation
- Evaluation
- Conclusion and Future Work

RL Formulation: Evicting N Bits

- **Given**
  - a randomized cache with unknown mapping function
  - A victim address to be evicted
  - A constant N
- **Find**
  - A RL policy that generates a sequence of memory accesses
- **Objective**
  - **minimizes** the number of memory accesses (cost)
- **Constraint**
  - the victim address is evicted N times

Outline

- Background and Motivation
- Challenges
- RL Formulation
- Evaluation
- Conclusion and Future Work

Evaluation Goals

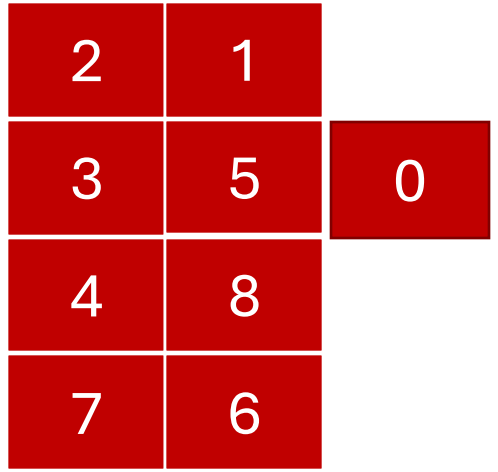
- **Train an RL agent that minimizes the number of steps to find minimal eviction set on randomized caches**
  - Q1: Is the agent **able to evict victim address** on randomized caches?
  - Q2: Are the addresses in the eviction sequence forming an **eviction set**?
  - Q3: Are the eviction set found by the agent a **minimal eviction set**?
  - Q4: how **many steps** does it take to find a minimal eviction set?

Evaluation Example

- Cache setting
  - A 4-set 2-way cache example
  - Address used: 0-8
  - Address 0-8 is randomly mapped to different cache locations
  - Victim address is 0

- RL setting
  - Evict victim address N times (N=1 , 5)
  - Episode length L (number of memory accesses) indicates the complexity

- Ideal case analysis
  - N =1, no need to actually “figure out” the eviction set of 0, just occupancy channel style accessing all addresses, L= 8
  - N=5, there is a need to reduce the number of steps to cause one eviction (figuring out a eviction set), L >= N * size(min_evset) + cost(evset_finding)
    - Size(min_evset) = 2, N = 5
    - L >= 2 *5 + cost(evset_finding)



4-set 2-way cache

address 0-8 randomly mapped to different set

Evaluated Cases

Cache Configuration	Epochs Trained	Episode Length	Victim Evicted	Eviction Set size	Cache ways	Steps taken
2 set 2 way	52	41	yes	2	2	29
4 set 2 way	10	60.35	yes	2	2	48
2 set 4 way	114	38.82	yes	4	4	19

Q1: Is the RL agent **able to evict victim address** on randomized caches?
Q2: Are the addresses in the eviction sequence forming an **eviction set**?
Q3: Are the eviction set found by the RL agent a **minimal eviction set**?
Q4: how **many steps** does it take to find a minimal eviction set?

Outline

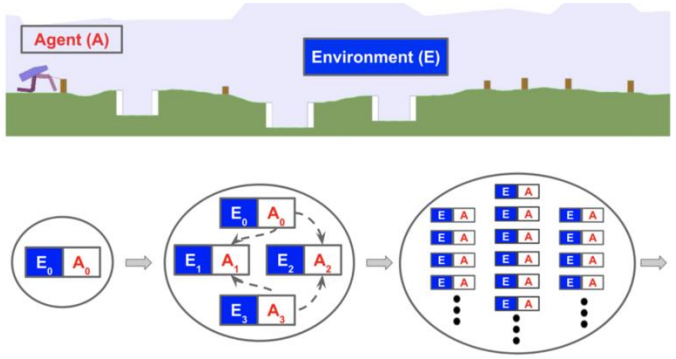
- Background and Motivation
- Challenges
- RL Formulation
- Evaluation
- Conclusion and Future Work

Conclusion and Future Work

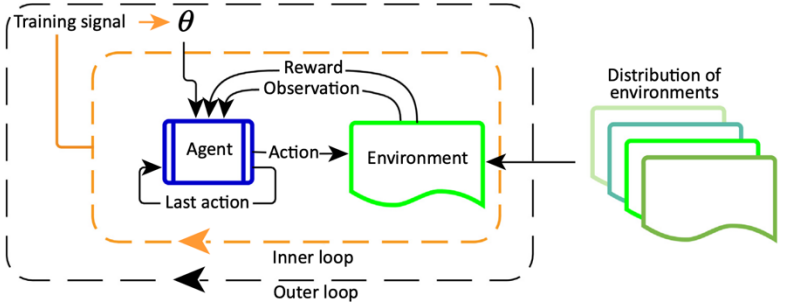
- Conclusion
  - randomized cache increases the difficulty of eviction set-finding
  - Reinforcement learning is a useful tool for effective eviction set finding on randomized caches
- Future Work
  - Scale to larger cache sizes and address space
  - Complexity analysis and comparison with existing eviction-set finding algorithms
  - Evaluate on partial congruent scheme, e.g., ScatterCache

Meta RL[1]

- Solving a class of problems rather than a single instance
  - Examples
    - E.g., solving any maze
    - E.g., finding out eviction sequence of any mapping function
  - Input: a meta parameter (may not be in the training set)
  - Output: a policy corresponding to that parameter
- Using Meta RL, a super agent (policy generator) learns to solve a class of problems
  - In general, an algorithm solves a class of problems
  - Thus, this super agent from Meta RL represents an algorithm
    - E.g., an algorithm that given the description of the maze, generate a policy that solves the maze
    - E.g., an algorithm that given the mapping function of a cache, find eviction set for particular address



A class of problems in Meta RL[3]



Meta RL [2]

1. Meta-Reinforcement Learning of Structured Exploration Strategies, Gupta et al, NIPS, 2018.
2. Reinforcement learning, fast and slow,Botvinick, 2018
3. https://www.uber.com/blog/poet-open-ended-deep-learning/

Hidden Meta Parameter

- **Problem:** For Randomized caches, the attacker does not see mapping function directly
  - The meta parameter is not visible to the agent
  - The input to the **super agent** is the same
- **Solution:** random sampling hidden meta parameter from a distribution during training
  - i.e., use different mapping functions (invisible to the agent) during training, so that the **super agent** learns the mapping and then find ways to evict the target address
- **Super Agent = An algorithm that evict an address for any mapping (randomization) function =/= an algorithm that finds eviction sets**