CENTER FOR EVOLVABLE COMPUTING

SPARK RESEARCH LAB
SECURITY, PRIVACY AND COMPUTER ARCHITECTURE

TEXAS
The University of Texas at Austin

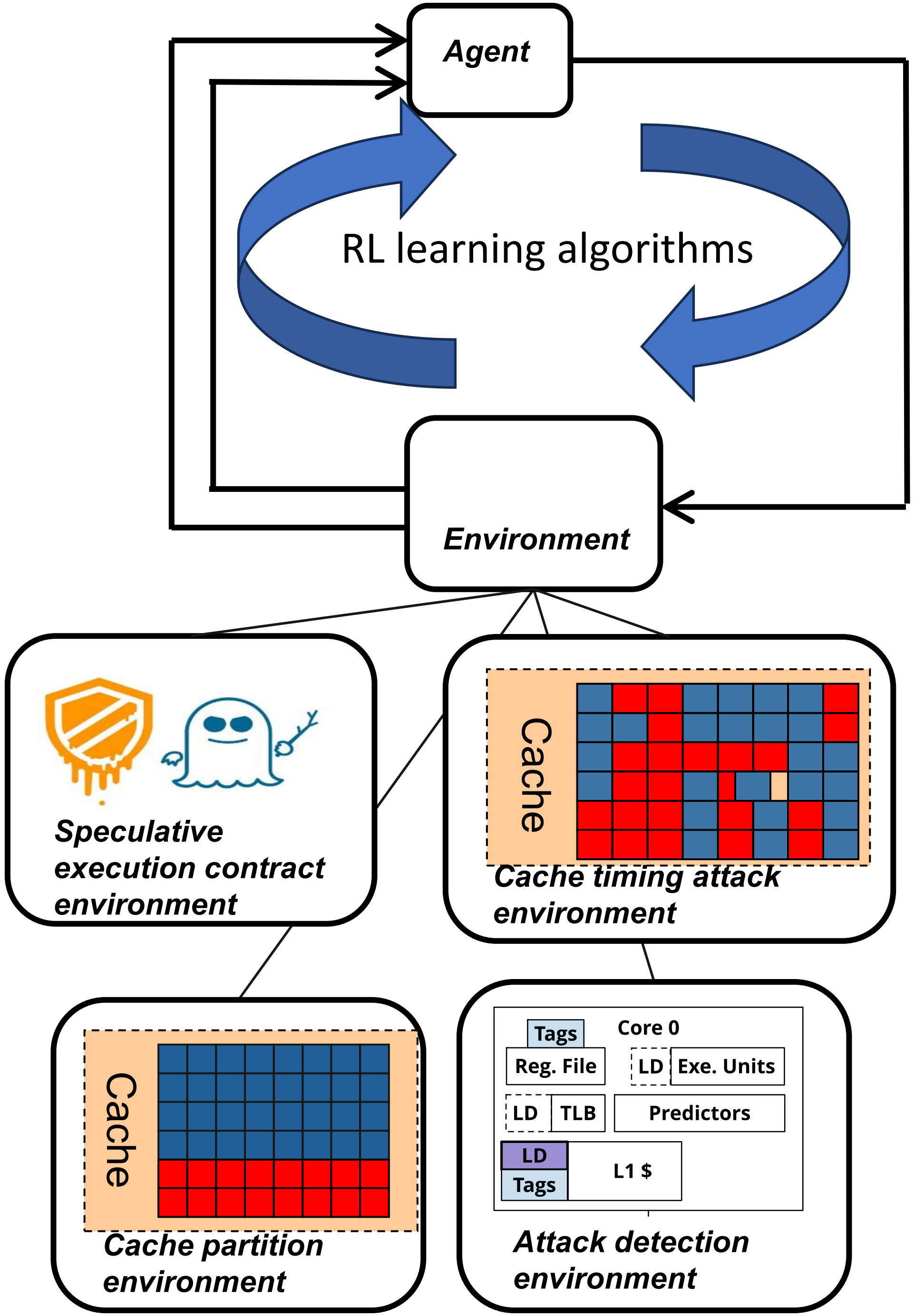
Reinforcement learning for microarchitectural security: cache timing channel, speculative execution, and defense

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

HASP workshop 2024
Nov 2, 2024

Executive Summary

- Microarchitectural security problems pose risks for information security in distributed systems
- Microarchitectural security analysis is laborious and error-prone
- Reinforcement learning is a useful tool that can achieve super-human performance
- We use reinforcement learning to address a variety of microarchitectural security problems

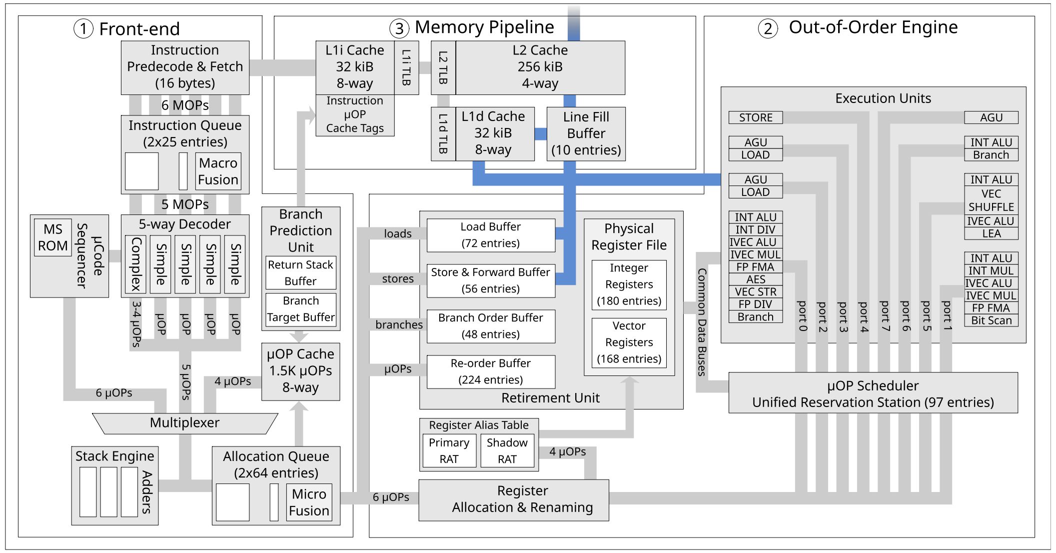


Outline

- Microarchitectural attacks and defenses
- RL methods
- Case studies
- Conclusions

Microarchitectural Attacks

- Adversaries exploit the microarchitecture vulnerabilities in microprocessors
  - steal information
  - damage the information integrity
  - makes the processor unavailable
- Examples:
  - Cache timing channel attacks
  - Speculative execution attacks
- Challenges for hardware:
  - Design time evaluation
  - Run time detection/defense

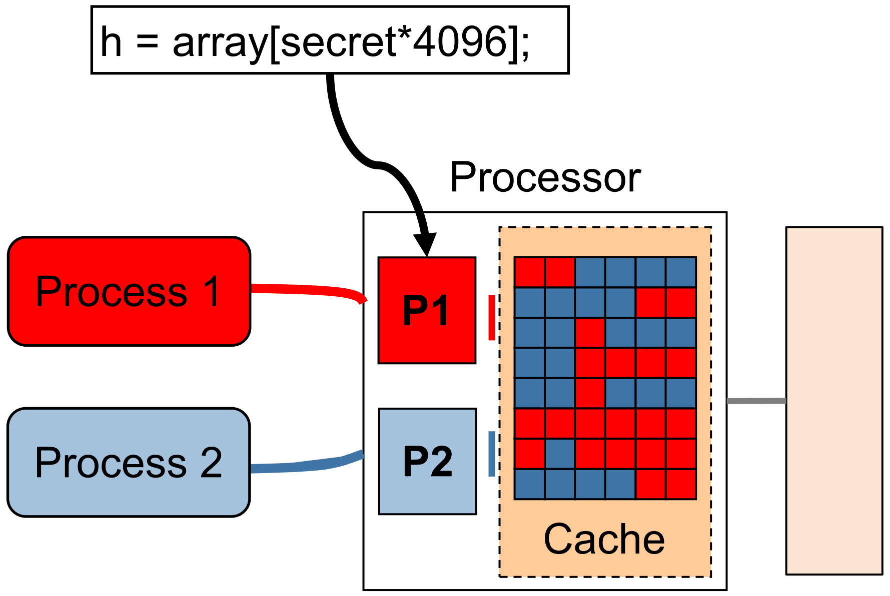


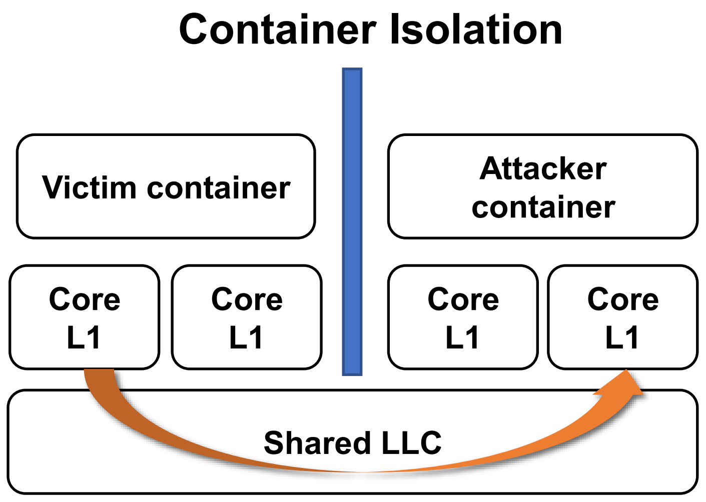
OOO architecture

Meltdown/Spectre

Cache-Timing Attack

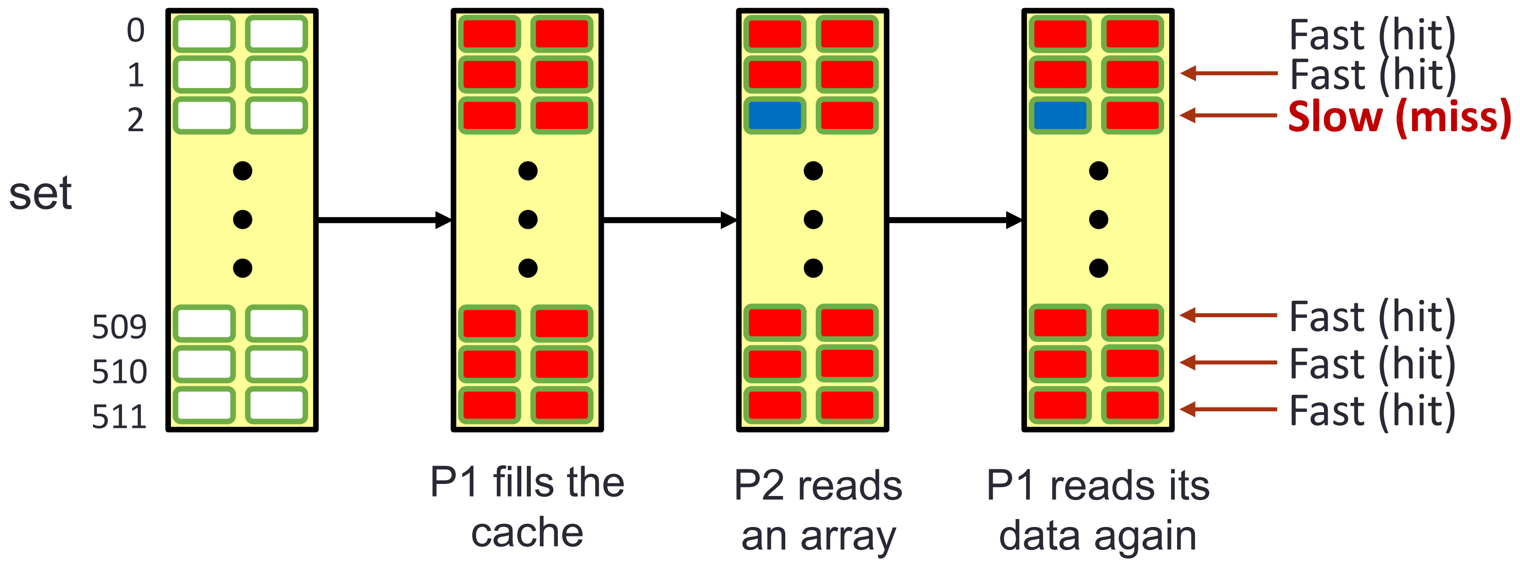
- Mechanism
  - sharing of caches by different processes
  - infer secret by observing cache timing
- Advantages
  - attacker is just a program, no physical access
  - does not violate any OS-level access control
- Leak important assets
  - cryptographic keys
  - Container/browser isolation
  - building blocks for Spectre/Meltdown





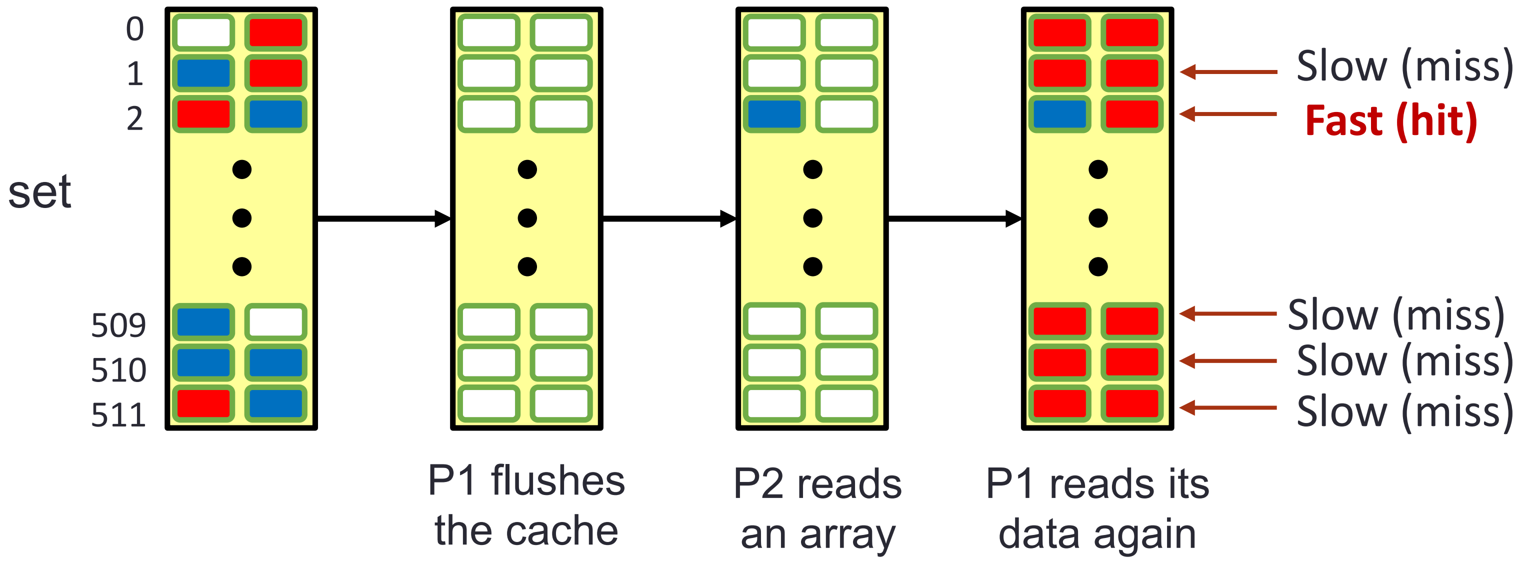
Prime+Probe Attacks

- P1 and P2 have different address space



Flush+Reload Attacks

- P1 and P2 share the same address space



Speculative Execution Attacks

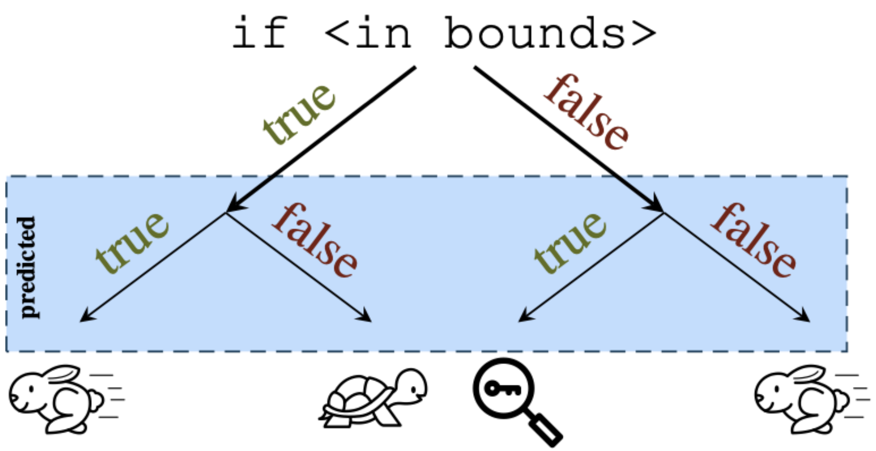
- Speculative execution may access secret by passing the domain isolation

- Speculative execution does not change architectural states represented on contract traces, CTrace)

- Speculative execution changes microarchitecture states (represented on Hardware traces, Htrace)

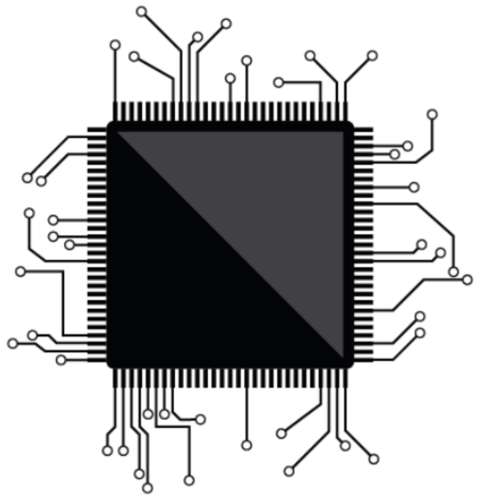
```
if (x < array1_size)
  y = array2[array1[x] * 4096];
```

Listing 1: Conditional Branch Example



Secure Design Challenges

- System is too complex
  - laptop processors have ~ 20,000,000,000 transistors



A microprocessor

- Undefined system behavior
  - timing of a memory read is unspecified
  - speculative execution that are not committed

- New architecture designs and optimizations create new vulnerabilities
  - E.g., the prefetcher in Apple Silicon

Meltdown/Spectre

RL's superhuman performance is a useful tool

Microarchitectural Defenses

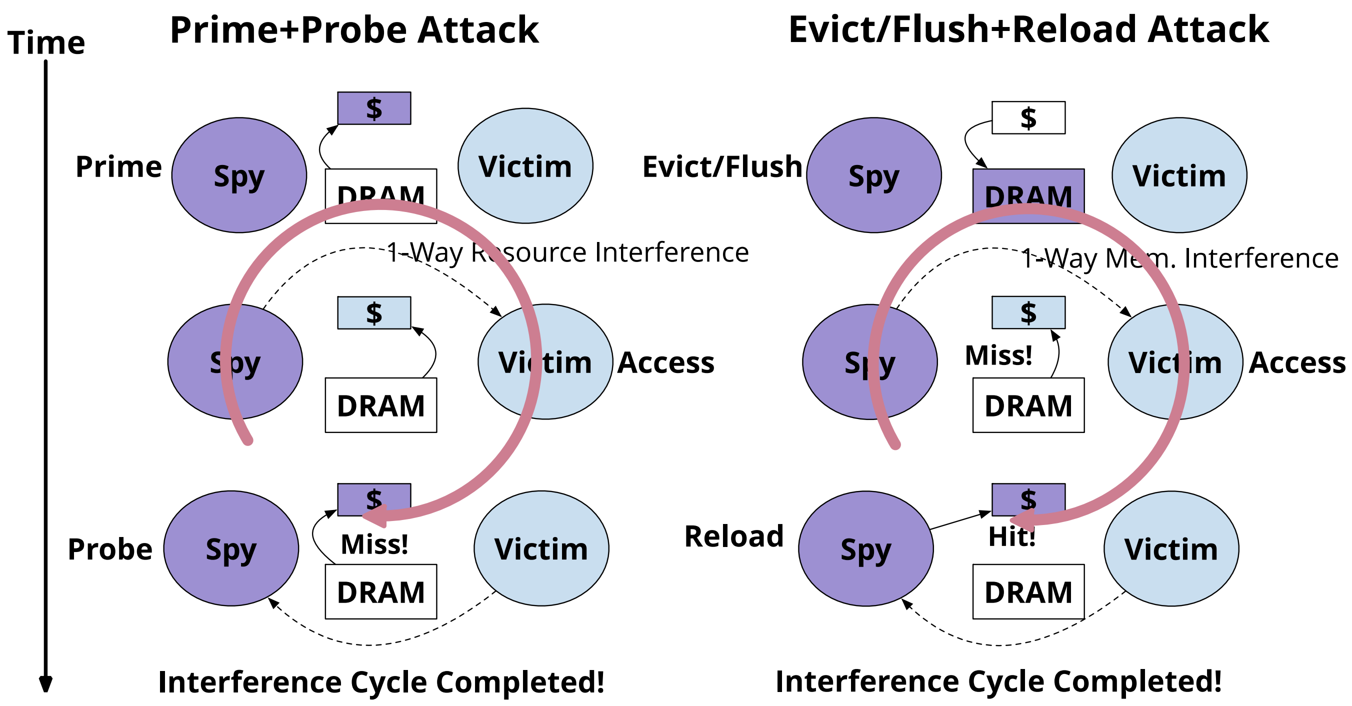
- Detection and response
  - Determine whether attacks exist at run time and response correspondingly

- Isolation
  - Separate different domains, eliminate interference

- Randomization
  - Randomize the actual interference, making it hard to guess the secret based on interference

Detection

- Example: Cyclone detector for cache timing attacks



- Cyclic Interference is a robust feature
  - Opportunity: detect attacks as anomalous cyclic interference

Harris, Austin, et al. "Cyclone: Detecting contention-based cache information leaks through cyclic interference." *MICRO 2019.*

Isolation

- Static isolation: eliminate the interference
  - Inflexible to the workload performance needs

- Dynamic isolation: SecDCP
  - Adjustable to performance needs of the applications
  - Potential leakage

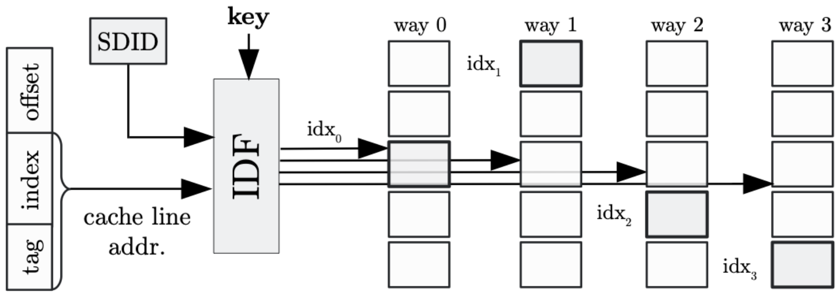
L: 2 cache ways
H: 2 cache ways

L: 3 cache ways
H: 1 cache way

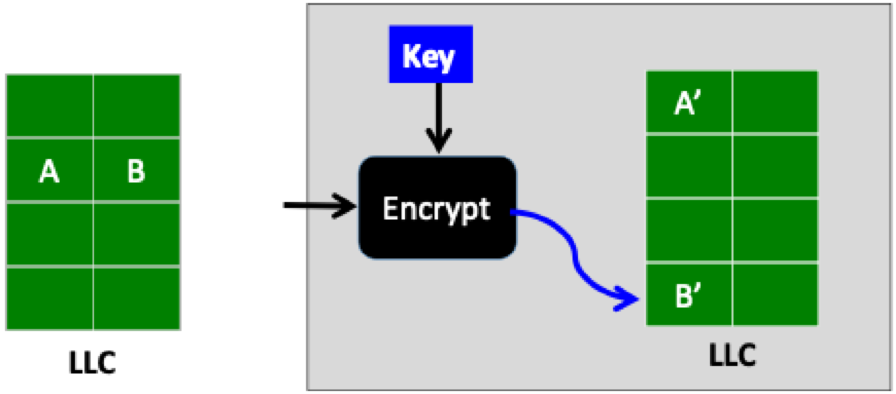
Wang, Yao, et al. "SecDCP: secure dynamic cache partitioning for efficient timing channel protection." *DAC,* 2016.

Randomization

- Forms
  - Static randomization
  - Dynamic randomization
- Implementation
  - Table-based
  - Cipher-based



ScatterCache



CEASER

1. Werner, Mario, et al. "{ScatterCache}: thwarting cache attacks via cache set randomization." *28th USENIX Security Symposium (USENIX Security 19)*. 2019.
2. Qureshi, Moinuddin K. "CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping." *2018 MICRO 2019.*

Runtime Defense Challenges

- Adaptive attackers
  - Attackers that dynamically adapts to existing (public) defense/detect mechanisms

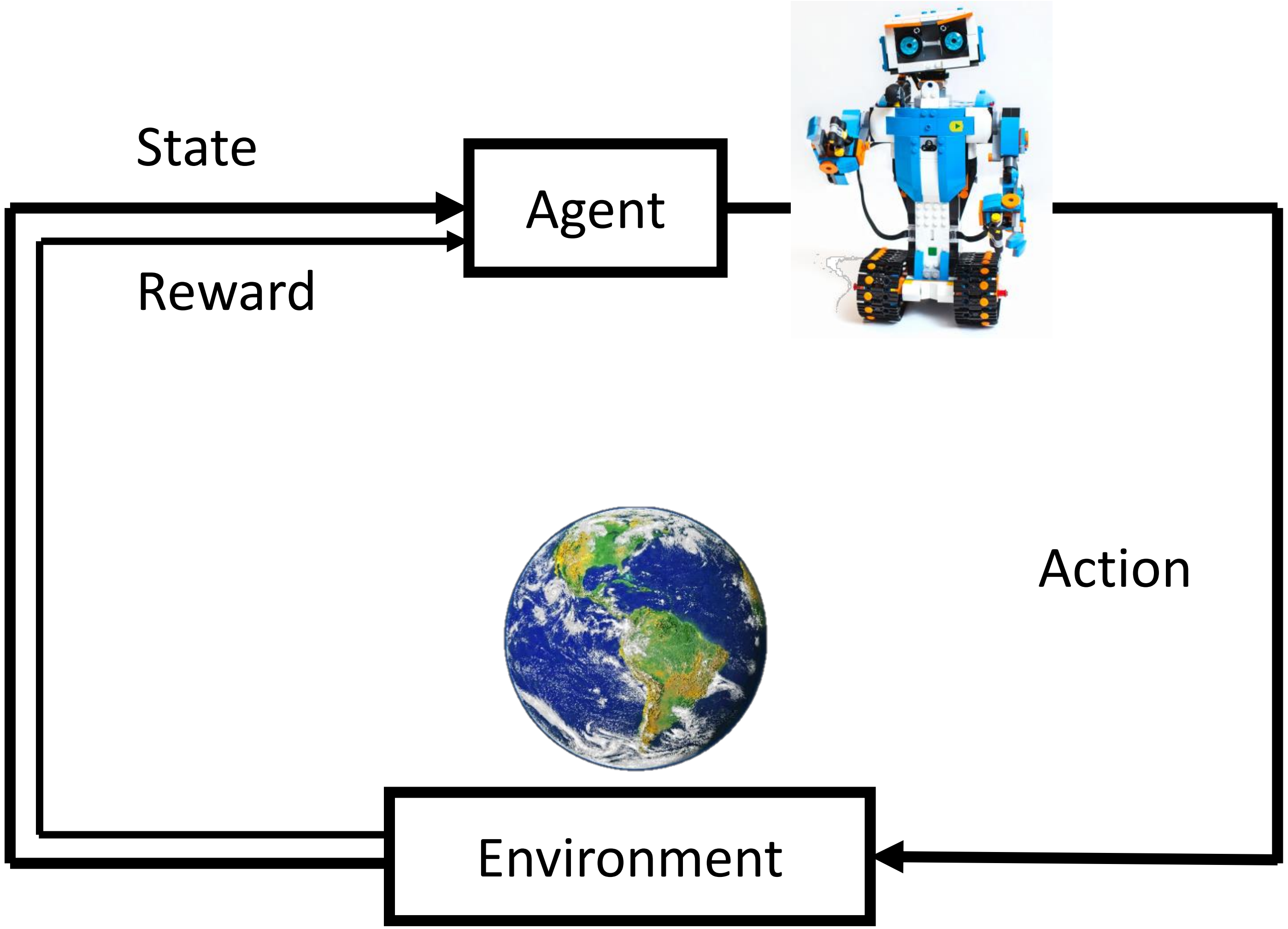
- Unseen attackers
  - Attackers whose attack strategies are unknown

Advanced RLs can be used to address these attacker challenges.

Outline

- Microarchitectural attacks
- RL methods
  - Single-agent RL
  - Multi-agent RL
  - Meta RL
- Case studies
- Conclusions

Reinforcement Learning (RL)



RL for Games

Go

Chess

Shogi

Poker

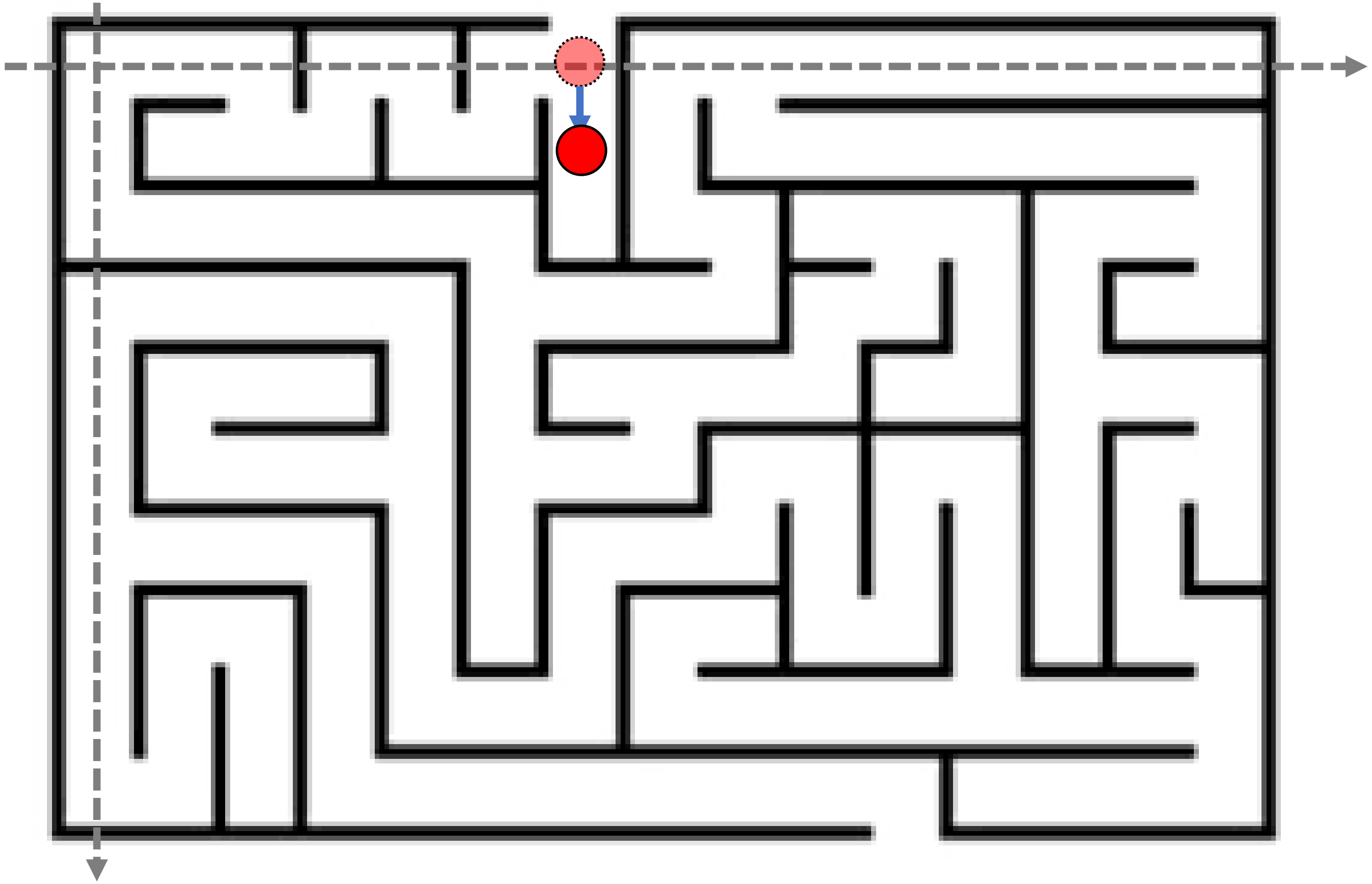
DoTA 2

StarCraft II

Big Success in Games

(courtesy of Dr. Yuandong Tian)

Maze Solving with RL



State:

= (,) = (6,0)

Actions:

Left: ← − 1

Right: ← + 1

Up: ← − 1

Down: ← + 1

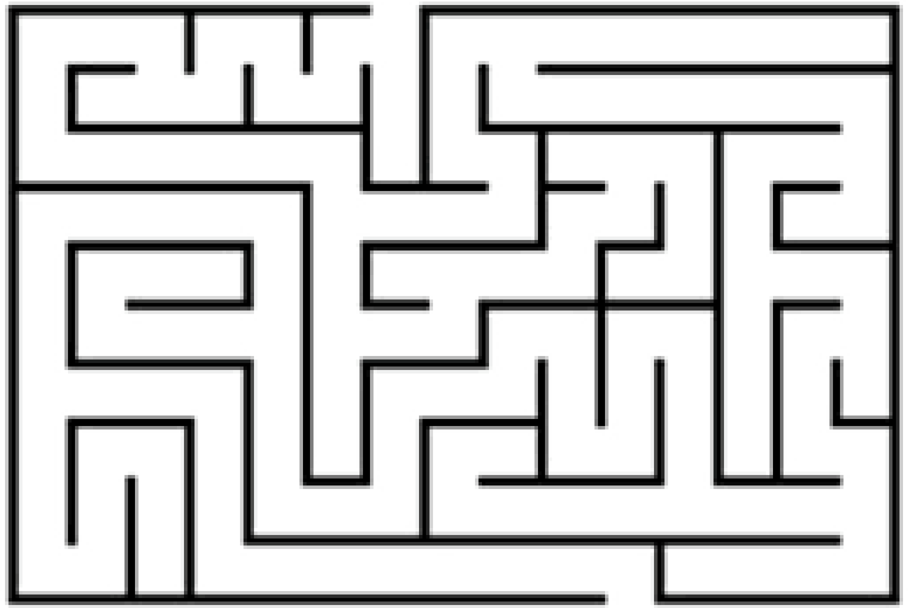
RL Advantages

- No dataset needed
  - data is generated by the environment

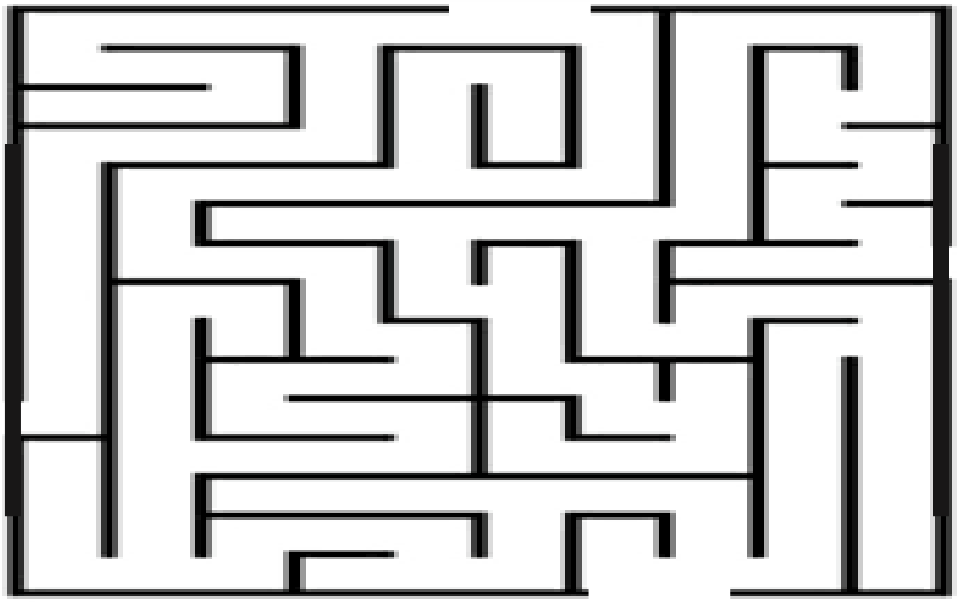
- Learning from feedback (reward)
  - Efficient use of data
  - Many fuzzing method do not use any feedback or use it insufficiently

Generalization Issue: A Different Maze

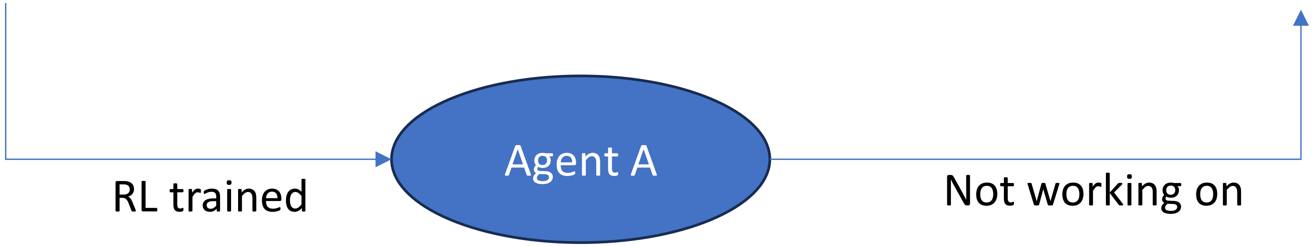
- An agent trained on one environment does not work on the other environment



Maze A

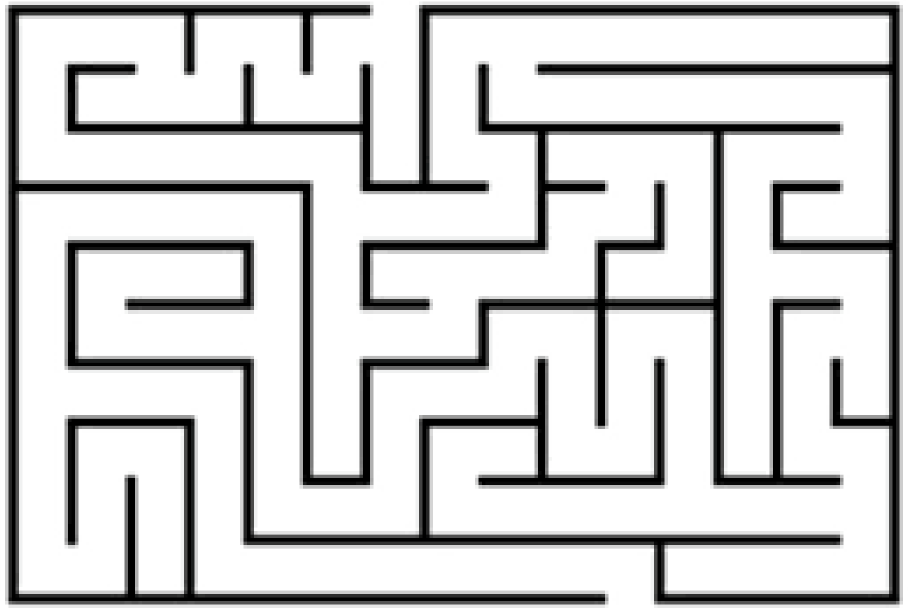


Maze B

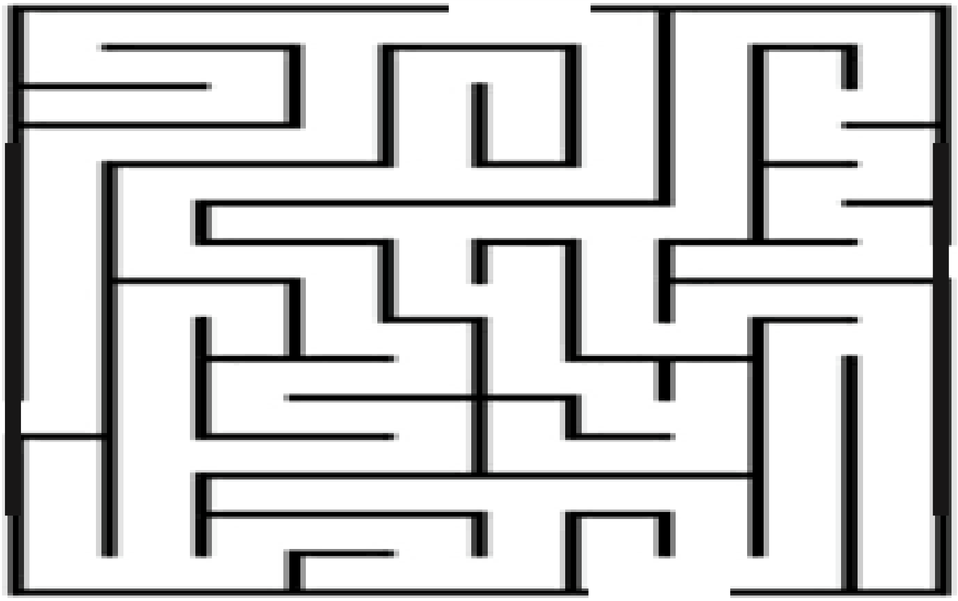


Generalization Issue: Dynamic Changing Maze

- An agent trained on a static environment does not work on a dynamic environment



Gradually change one wall at a time



Maze A

Maze A'

RL trained

Agent A

Not working on Task 3134.013

21

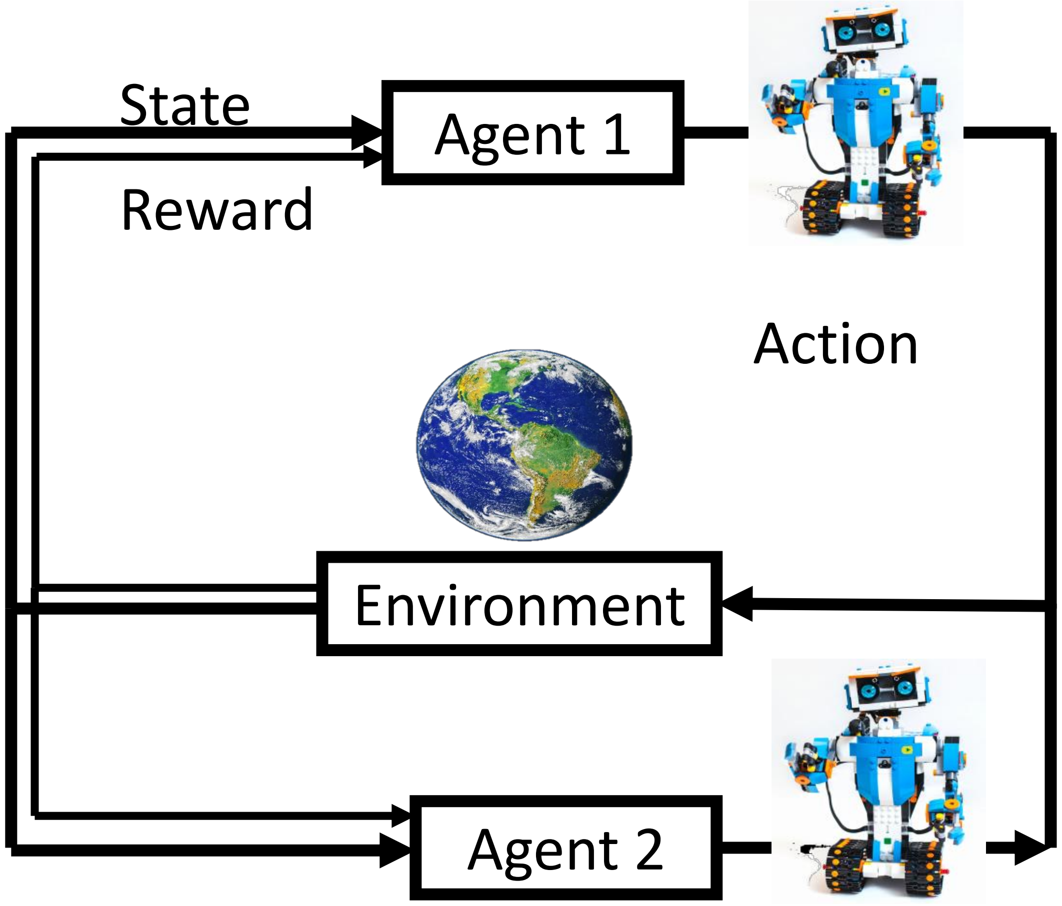
RL Generalization Issues

- Difficult to adapt to different (dynamic) environment
  - E.g., an adaptive attacker who changes attack strategy based on the detector, making it difficult to detect
  - Solution: multi-agent RL

- Difficult to adapt to different (static) environment
  - E.g., a randomized cache whose randomization is different for different machine instance
  - Solution: Meta RL

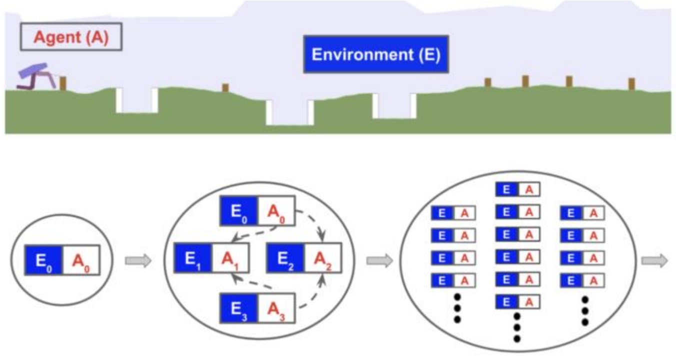
Multi-agent RL

- An RL that has more than one agent
  - One agent is used for the original purpose (detection/defense)
  - The other agent is used for modeling the adaptive behavior of the adversary



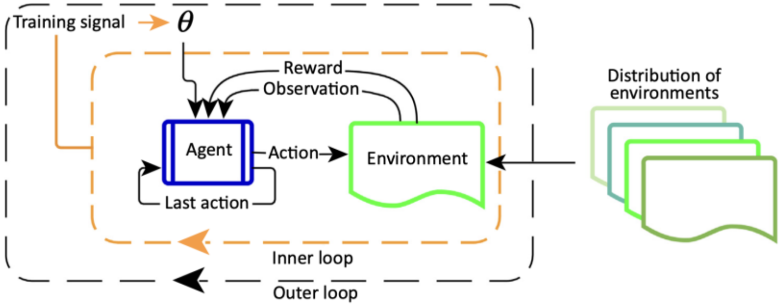
Meta RL

- 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



A class of problems in Meta RL[3]

- 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, generates a policy that solves the maze
    - E.g., an algorithm that given the mapping function of a cache, finds eviction set for particular address



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/

Task 3134.013

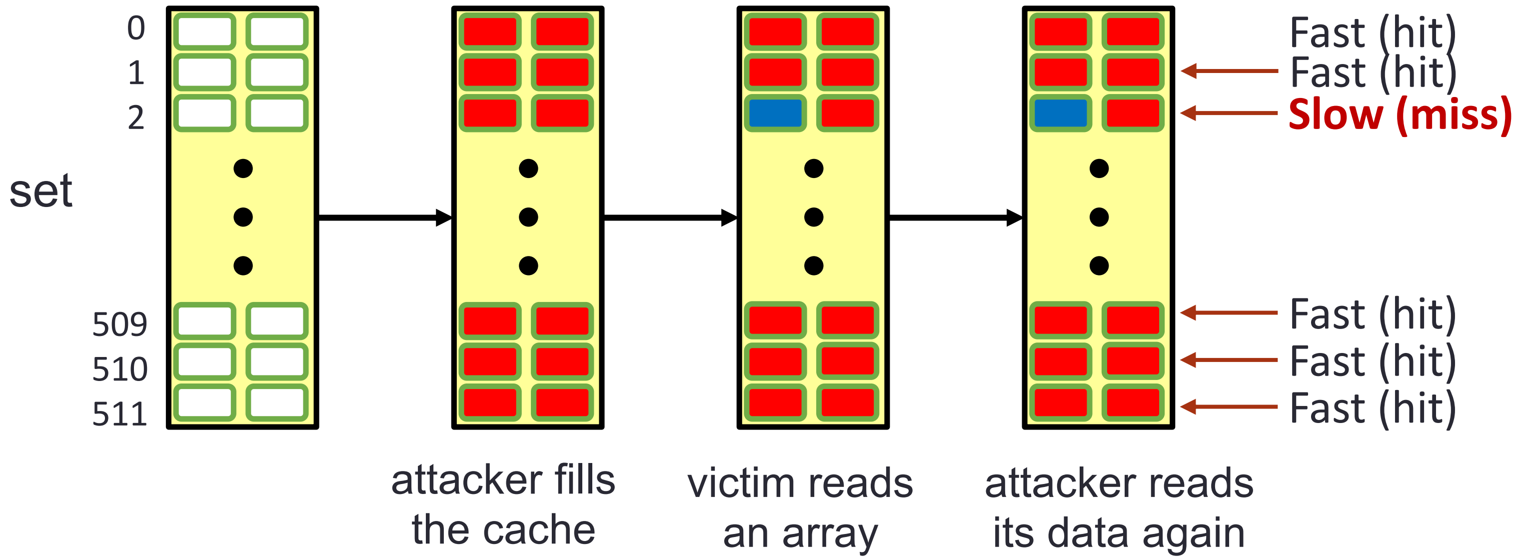
RL Methods Summary

RL Method	pros	cons	Microachitectural security Use cases
Single-agent RL	Simple Quick converge	Instance specific, not generalizable Cannot foresee unseen scenarios	Attack based on single set vulnerabilities
Multi-agent RL	Good for adversarial scenarios Robust against unseen scenarios Generalization for dynamic scenarios	long training time	detection training
Meta RL	Generalization for multiple environment	Long training time	Eviction set finding

Outline

- Microarchitectural attacks
- RL methods
- Case studies
  - AutoCAT: RL for cache timing attacks
  - SpecRL: RL for speculative contract detection
  - MACTA: multi-agent RL for cache timing attack detection
  - RLdefender: RL-based cache partition for security
  - AlphaEvict: Eviction set finding with RL
- Conclusions

Case 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: Reinforcement Learning for Automated Exploration of Cache-Timing Attacks, M. Luo, et al, HPCA, 2023

Case 2: SpecRL - Speculative Contract Violation Detection

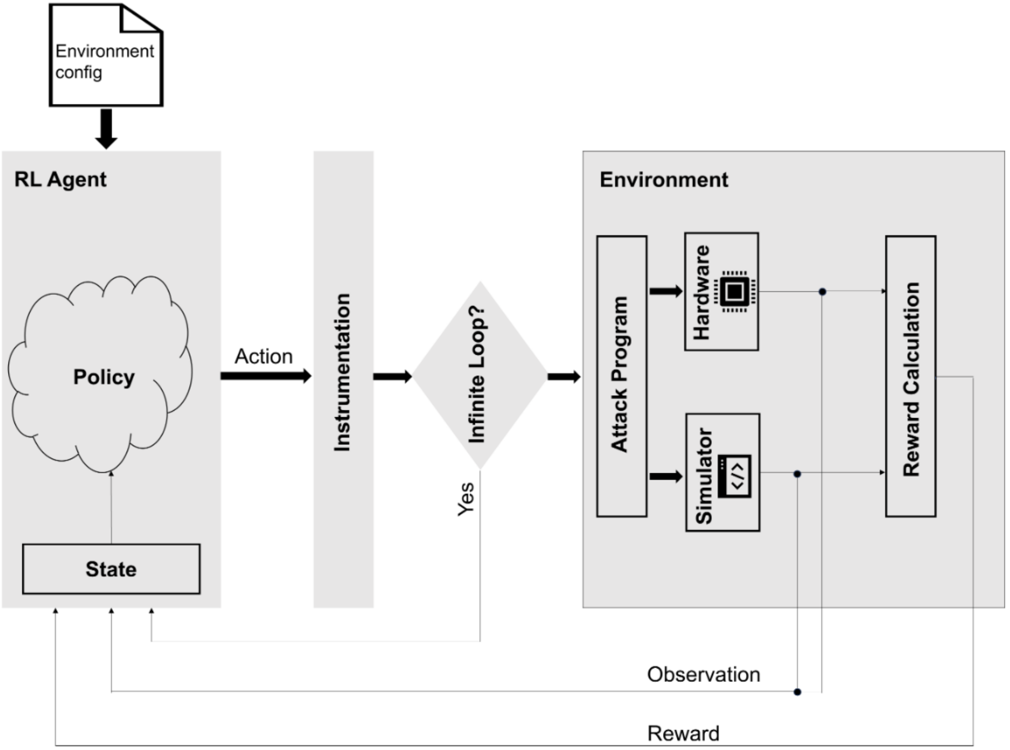


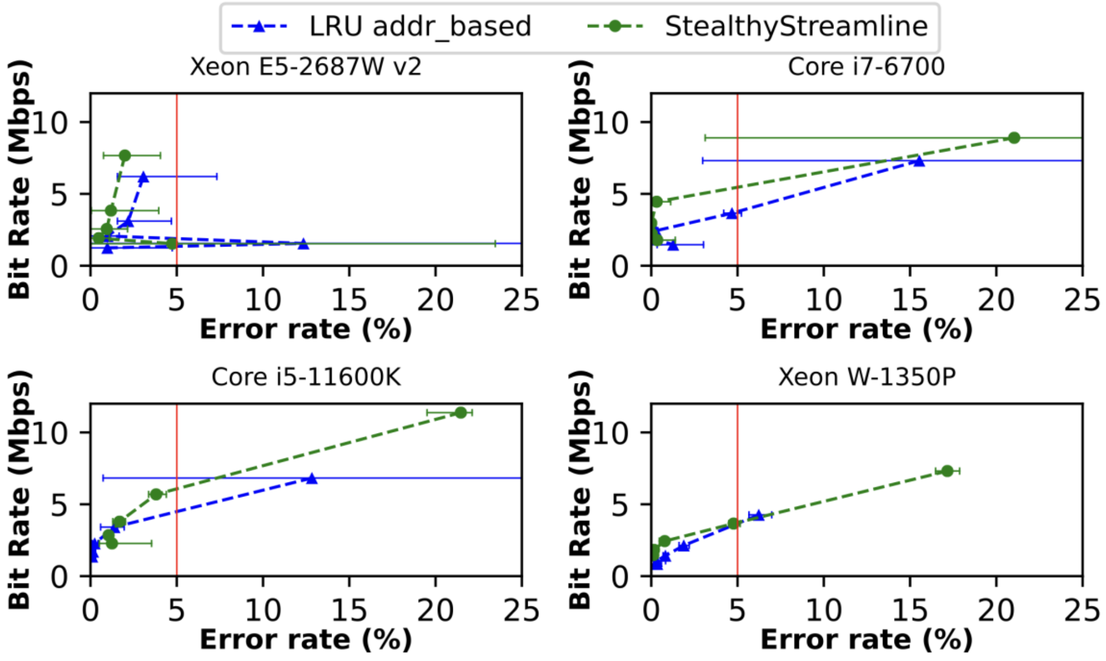
Fig. 1: SpecRL's training flow

- Agent: Attacker
- Environment: Processor
- Actions
  - Adding one assembly instruction
- Observation
  - Htraces (Hardware trace) of two inputs
  - Ctraces (contract trace) of two inputs
- Reward
  - 0, Htraces of two inputs are the same
  - Positive, Htraces of two inputs are different

SpecRL: Reinforcement Learning for Speculative Execution Vulnerability Exploration, E. Lai, M. Luo, M. Tiwari, manuscript in preparation.

Case 1 & 2: AutoCAT and SpecRL Results

AutoCAT can find high bandwidth cache timing channel attack – StealthyStreamline tested on 4 different processors

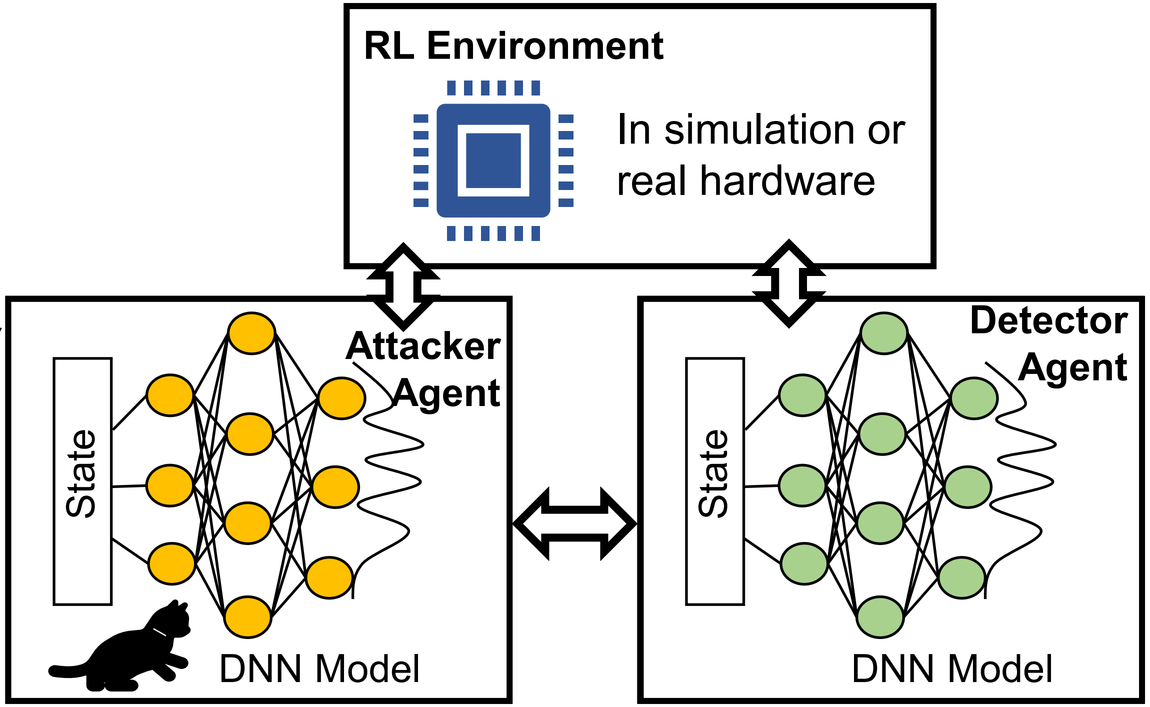


SpecRL can detect Spectre-V0 attack in a few training iterations on i7-6700

```
.test_case_enter:
.line_1:
SBB qword ptr [R14 + RBX], 35
.line_2:
JNS .line_4
.line_3:
JMP .line_5
.line_4:
IMUL byte ptr [R14 + RCX]
.line_5:
.line_6:
.line_7:
.line_8:
.line_9:
.line_10:
.test_case_exit:
```

Case 3: MACTA- A Multi-agent RL for Detection of Cache Timing Attacks

- Approach:
  - Multi-agent reinforcement learning (RL) for automatically exploring cache-timing attacks and detection schemes together.
- Key Findings:
  - Without any manual input from security experts,
    - the trained attacker is able to act more stealthily
    - the trained detector can generalize to unseen attacks
    - the trained detector is less exploitable to high-bandwidth attacks.

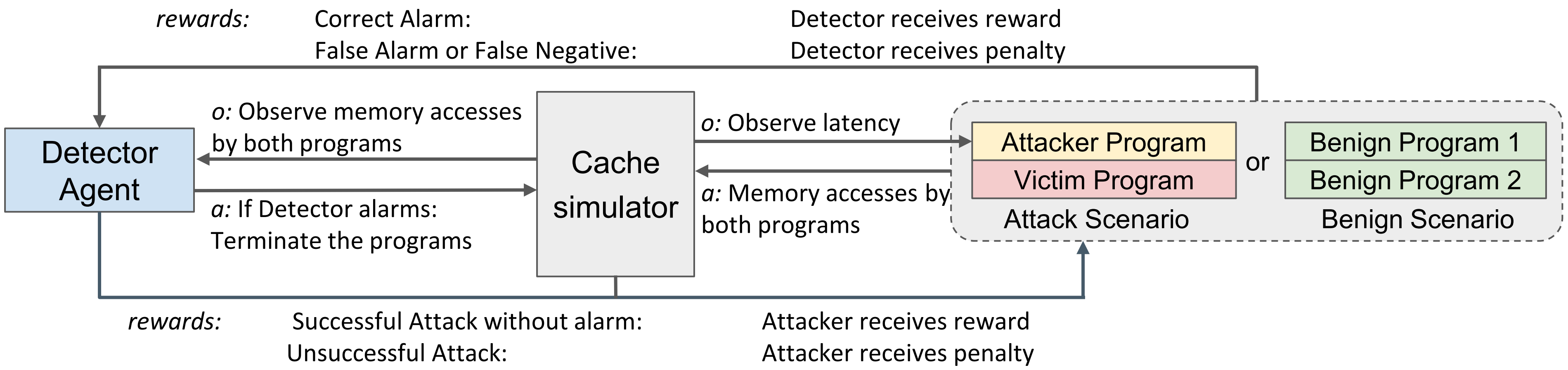


MACTA: A multi-agent Reinforcement Learning Approach for Cache Timing Attacks and Detection, J. Cui, X. Yang. M. Luo, et. Al., ICLR 2023.

Case 3: MACTA Formulation

- Agent 1: AutoCAT attacker
- Agent 2: RL detector
- Environment: Cache
- Actions: raise alarm

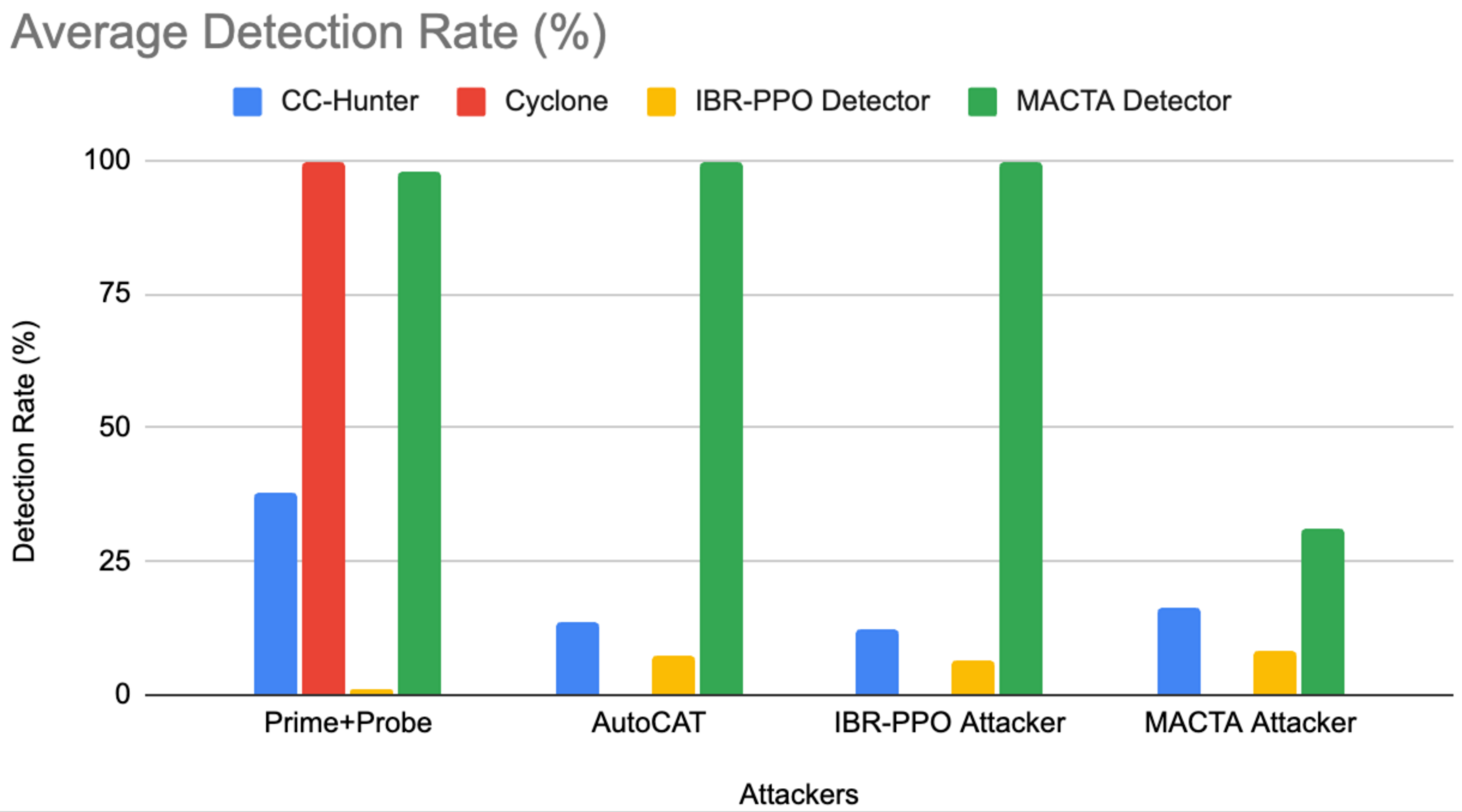
- Reward
  - Correct detect: positive reward
  - False positive: low negative reward
  - False negative: high penalty
  - each step: small negative reward

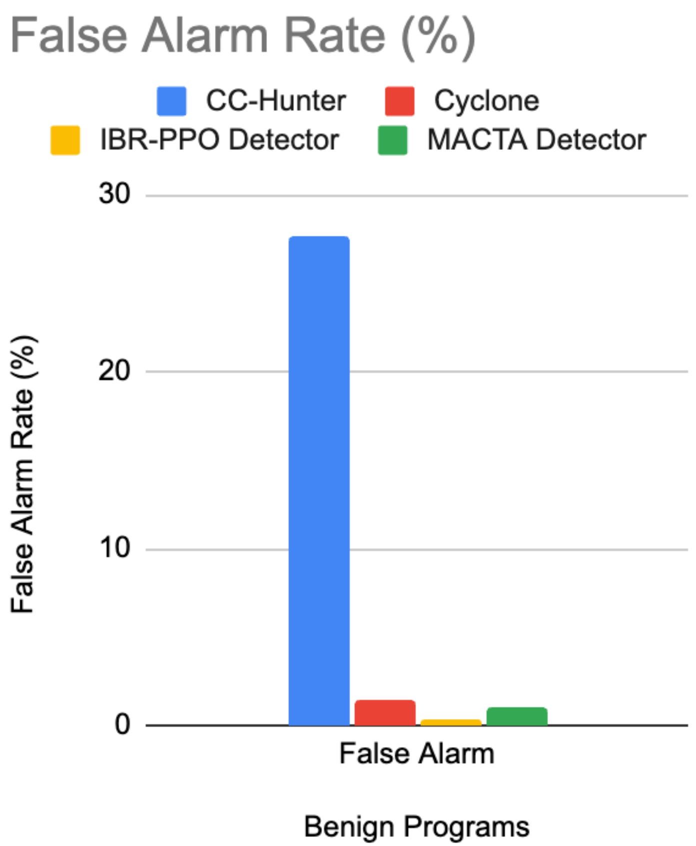


MACTA: A multi-agent Reinforcement Learning Approach for Cache Timing Attacks and Detection, J. Cui, X. Yang. M. Luo, et. Al., ICLR 2023.

Case 3: MACTA Results

- Without any manual input from security experts,
  - the trained MACTA detector can generalize to unseen attacks

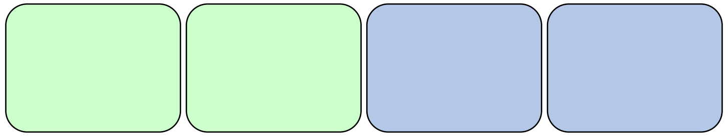




Case 4: RL defender –Multi-agent RL for Cache Set Partitioning

- Cache set partition limits the cache locations attacker can use
  - Reducing interference, making it difficult for an adaptive attacker to guess secret correctly
  - May negatively impacts the cache performance (e.g. miss rate)
- We use RL to dynamically partition each cache set that
  - Reduce attacker guess correct rate
  - Improve cache miss rate

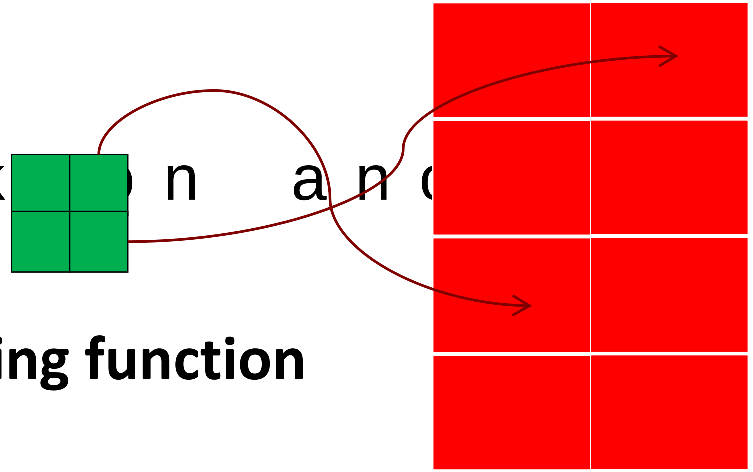
- Agent 1: AutoCAT attacker
- Agent 2: RL defender
- Environment: Cache
- Actions
  - Lock specific lines in a cache set (the locked cache line cannot be used by a different domain)
- Reward
  - guess correct: positive reward
  - guess wrong: negative reward
  - **each step: small negative reward**



Case 5: Meta RL for Eviction Set Finding

- Cache randomization makes it difficult to find eviction set for specific addresses
  - “one agent does not work on and

- We use Meta RL Techniques to find eviction sets
  - Each mapping function is one RL instance
  - Train one RL agent with changing RL mapping functions



Cache Address Randomization

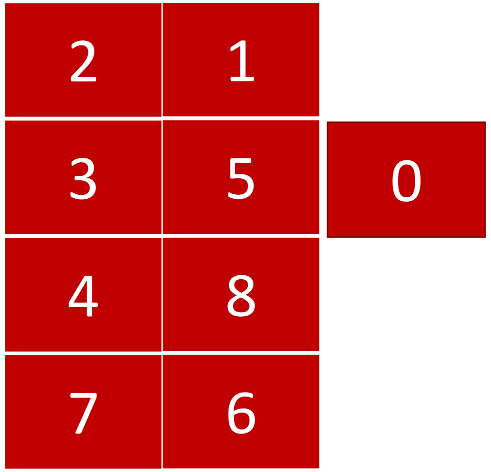
Eviction Set-Finding on Randomized Caches with Reinforcement Learning, M. Luo, M. Tiwari, SRC TECHCON 2024.

Case 5: 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 s 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

Case 5: 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

Meta RL can find eviction sets for any randomized mapping function in these scenarios!

Collaborators

- Mohit Tiwari, UT Austin
- Edward Suh, Nvidia / Cornell University
- Wenjie Xiong, Virginia Tech
- Hsien Hsin Lee, Intel
- Yuandong Tian, Meta AI
- Amy Zhang, UT Austin
- Benjamin Lee, University of Pennsylvania
- Jiaxun Cui, UT Austin
- Xiaomeng Yang, Google
- Erfan Iravani, Virginia Tech
- Evan Lai, UT Austin
- Mahir Kaya, UT Austin

TEXAS
The University of Texas at Austin

VIRGINIA TECH

Penn
UNIVERSITY of PENNSYLVANIA

Meta

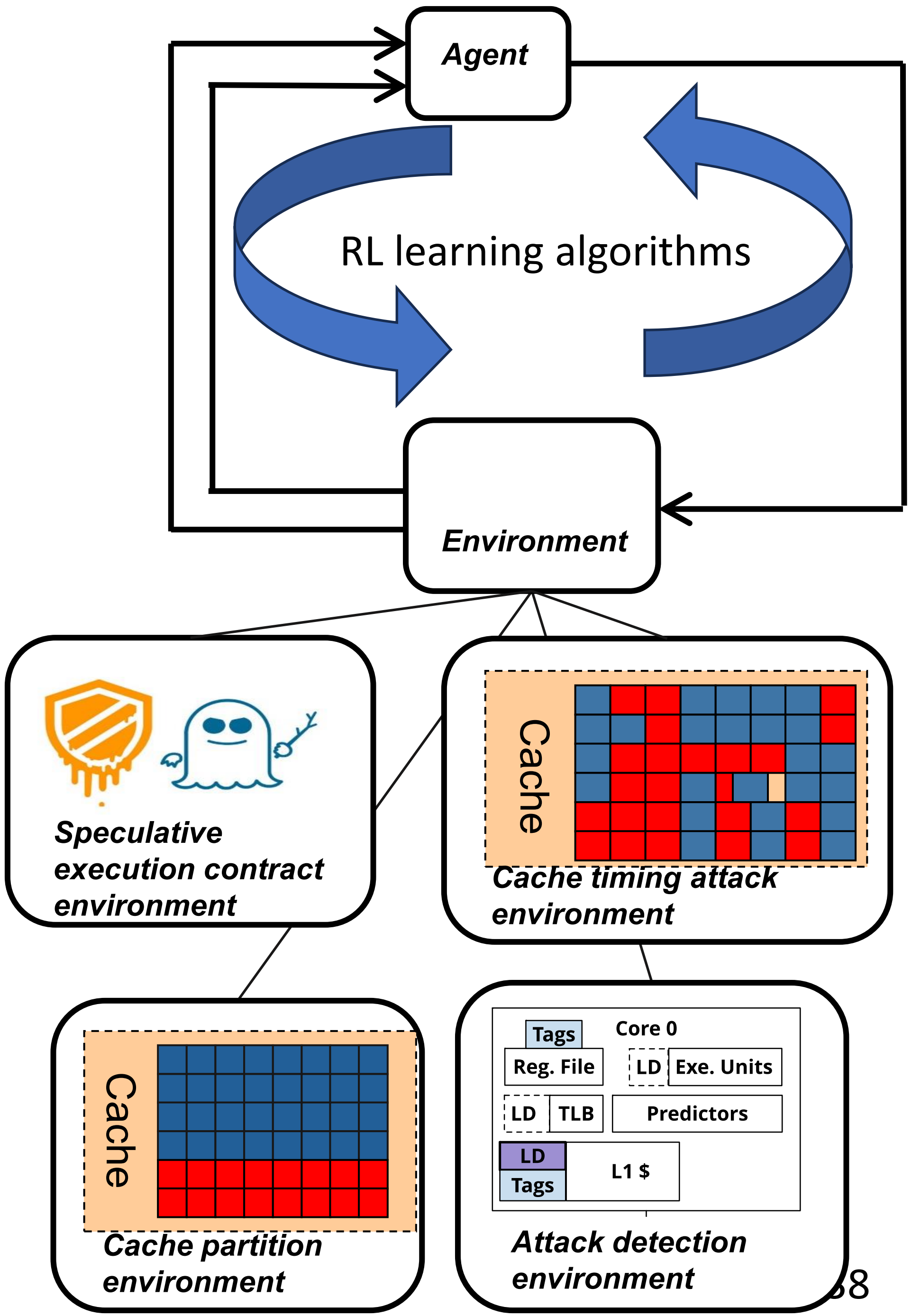
Cornell University

CENTER FOR EVOLVABLE COMPUTING

Conclusion and Future Work

- Summary
  - Microarchitectural security analysis is laborious and error-prone
  - We use reinforcement learning to address a variety of microarchitectural security problems

- Future Work
  - Explainable reinforcement learning for interpretable results



Acknowledgement

- This work is partially funded by SRC Jump 2.0 ACE Center for Evolvable Computing under task 3134.013

ACADEMIA • INDUSTRY • GOVERNMENT

ACE CENTER FOR EVOLVABLE COMPUTING