DAGguise

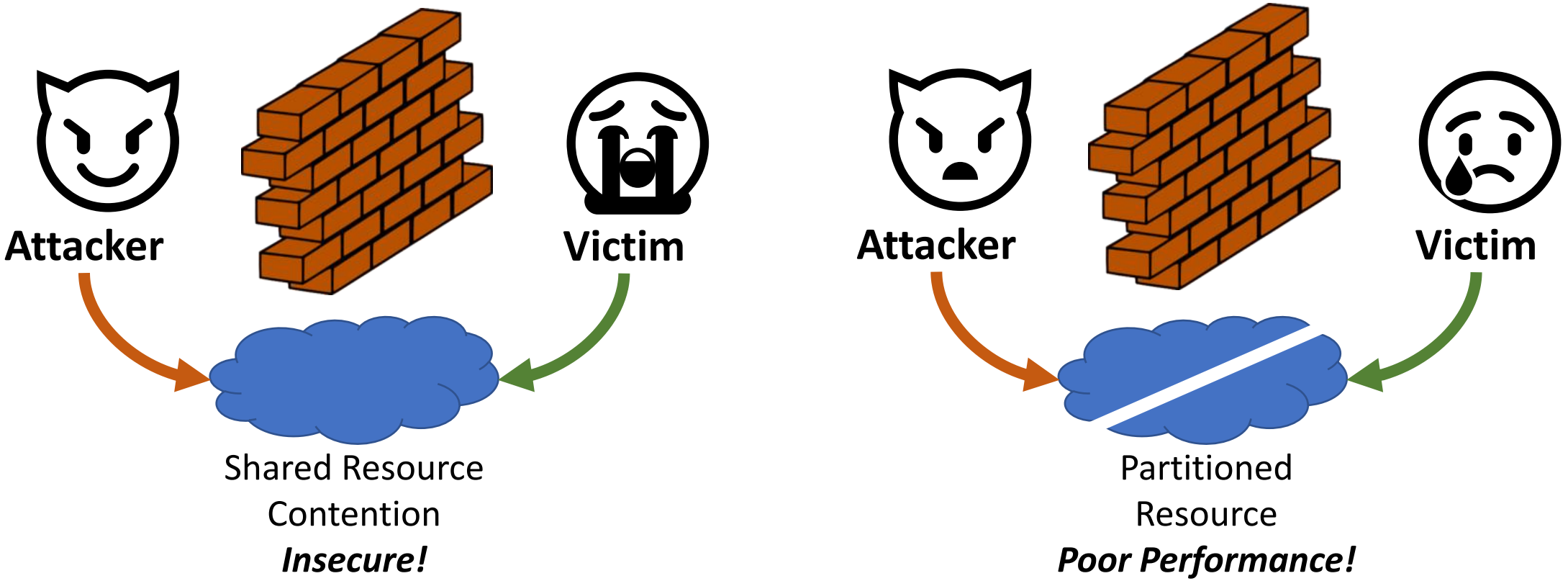
Mitigating Memory Controller Side Channels

Peter W. Deutsch*, Yuheng Yang*, *Thomas Bourgeat,*

Jules Drean, Joel Emer, and Mengjia Yan

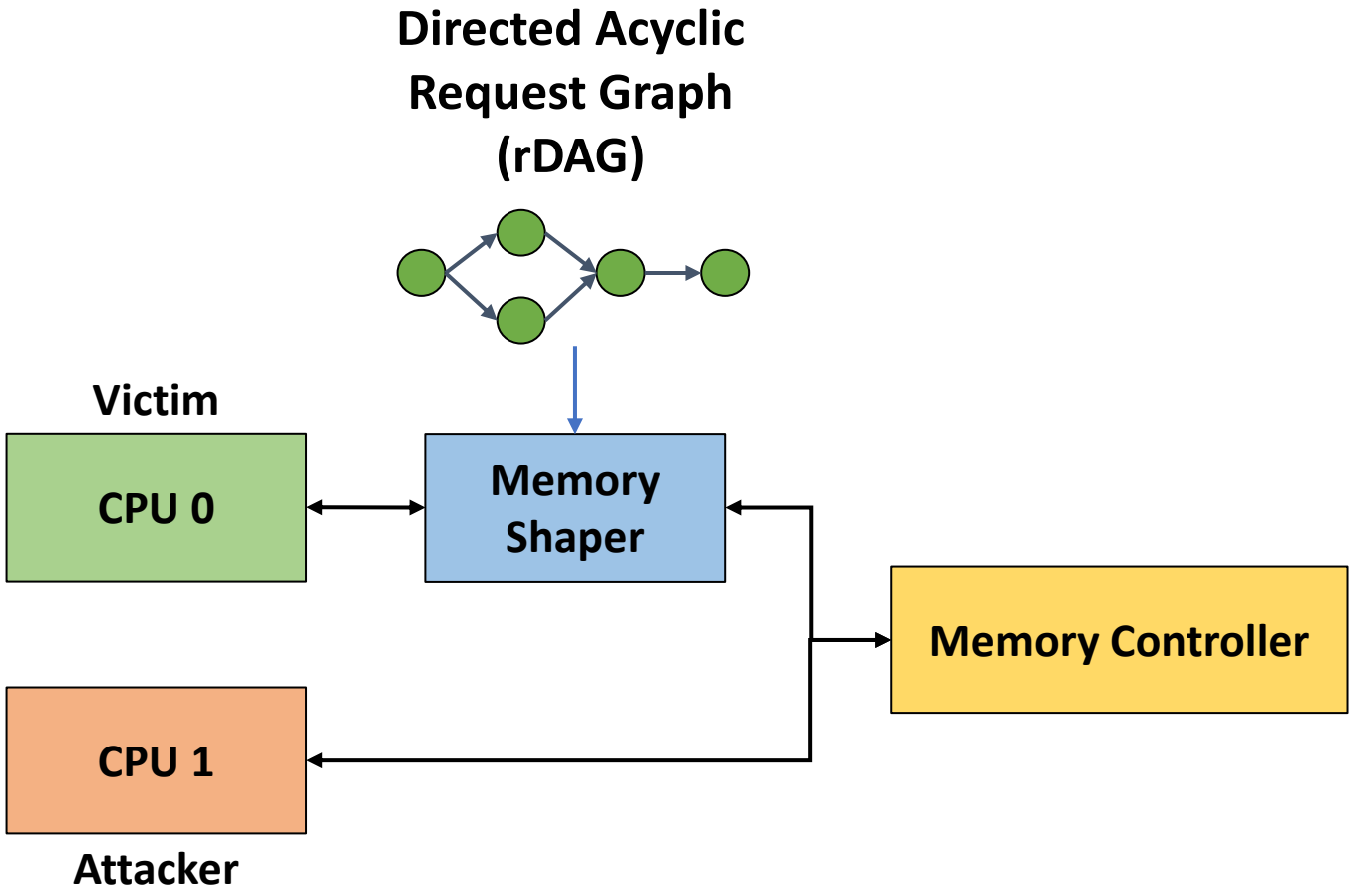
ASPLOS 2022 (Session 3A)

Microarchitectural Side-Channels



<u>Key Defense Tradeoff:</u> Security vs. Performance

DAGguise Key Idea



DAGguise achieves:

✓ **Formally-Verified Security**

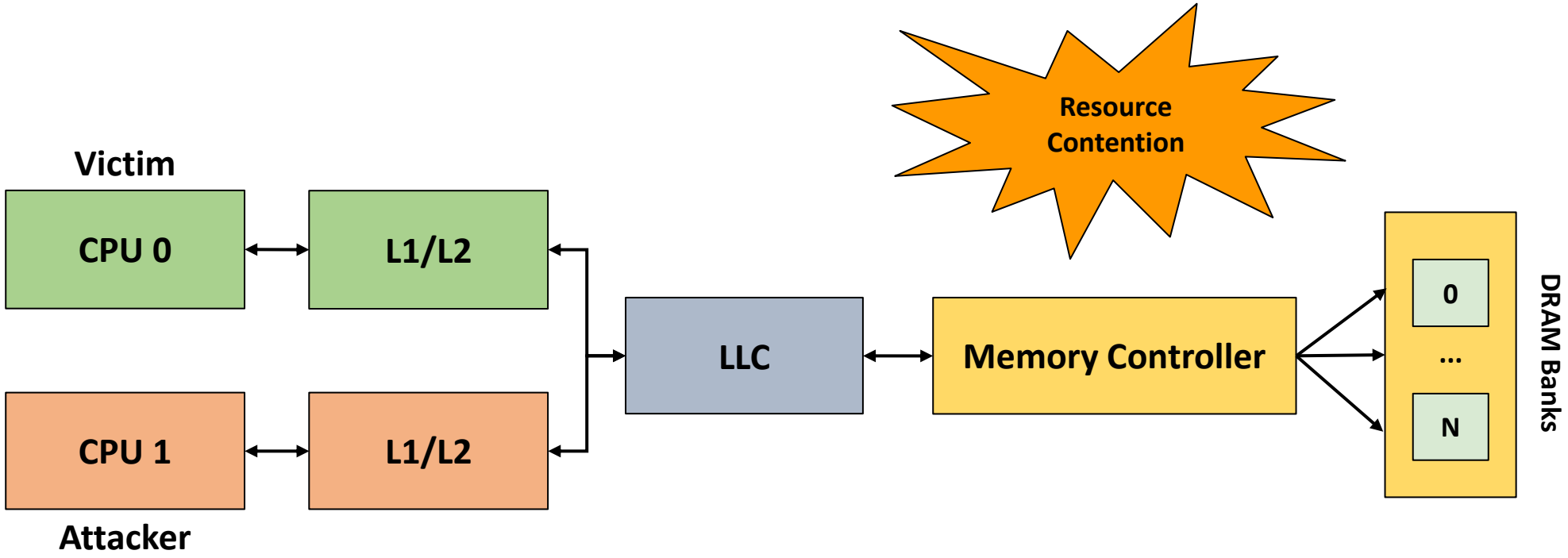
and

✓ **Good Performance**

Outline

- Memory Controller + Scheduler-based Side Channels
- Existing Approaches
  - Static Partitioning
  - Traffic Shaping
- DAGguise
  - Directed Acyclic Request Graphs (rDAGs)
- Security + Performance Evaluation
- Generalizability

Memory Controller Side Channels



This is a class of "*scheduler-based*" side channels!

Scheduler-Based Side Channels

This is the extended version of a paper that appears in USENIX Security 2021

ARTIFACT EVALUATED usenix PASSED

Lord of the Ring(s): Side Channel Attacks on the CPU On-Chip Ring Interconnect Are Practical

Riccardo Paccagnella Licheng Luo Christopher W. Fletcher
University of Illinois at Urbana-Champaign

DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks

Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz *and* Stefan Mangard
Graz University of Technology, Austria

Bandwidth Utilization Side-Channel on ML Inference Accelerators

Sarbartha Banerjee
The University of Texas at Austin
sarbartha@utexas.edu

Shijia Wei
The University of Texas at Austin
shijiawei@utexas.edu

Prakash Ramrakhyani
ARM Research
prakash.ramrakhyani@arm.com

Mohit Tiwari
The University of Texas at Austin
tiwari@austin.utexas.edu

arXiv:2110.07157v1 [cs.CR] 14 Oct 2021

SMoTherSpectre: Exploiting Speculative Execution through Port Contention

Atri Bhattacharyya
EPFL

Alexandra Sandulescu
IBM Research – Zurich

Matthias Neugschwandtner
IBM Research – Zurich

Alessandro Sorniotti
IBM Research – Zurich

Babak Falsafi
EPFL

Mathias Payer
EPFL

Anil Kurmus
IBM Research – Zurich

arXiv:1903.01843v3 [cs.CR] 26 Sep 2019

2019 IEEE Symposium on Security and Privacy

Port Contention for Fun and Profit

Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, Nicola Tuveri
Universidad Tecnológica de la Habana (CUJAE), Habana, Cuba
Tampere University, Tampere, Finland

Session 10D: VulnDet 2 + Side Channels 2 CCS'18, October 15-19, 2018, Toronto, ON, Canada

Rendered Insecure: GPU Side Channel Attacks are Practical

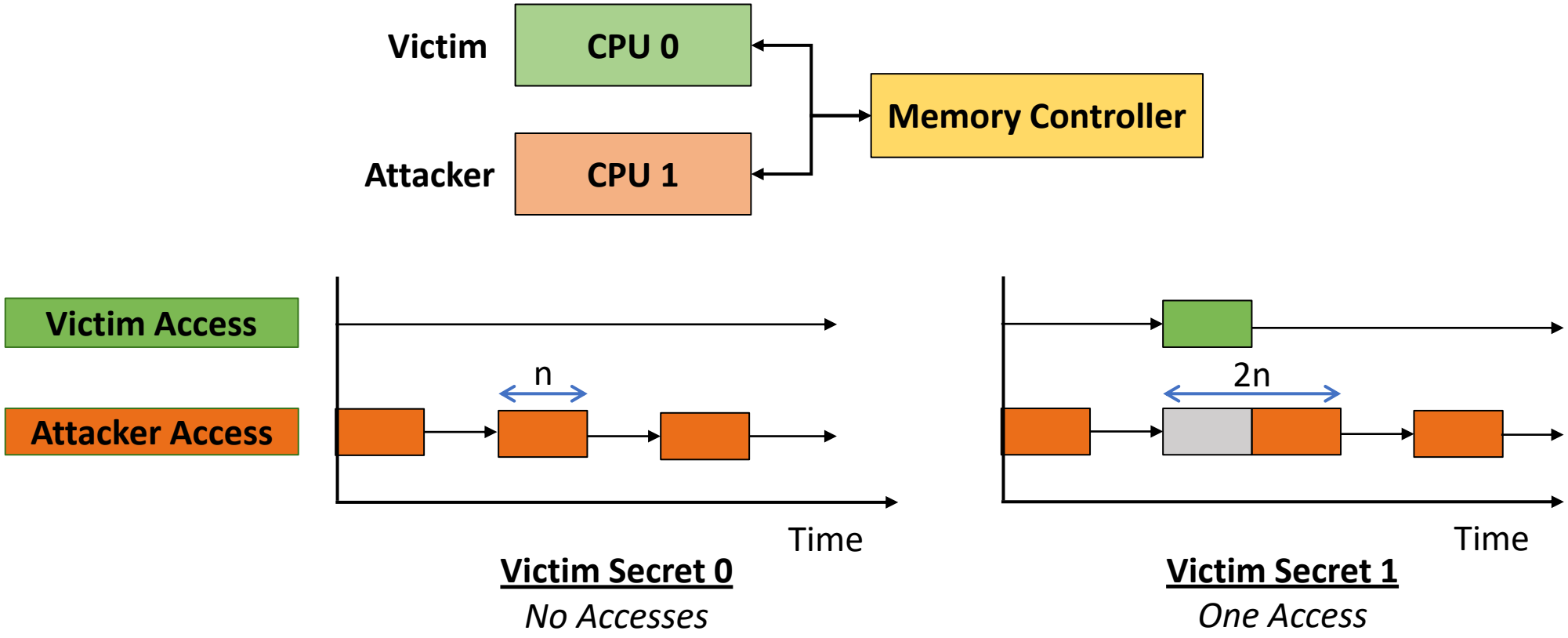
Hoda Naghibijouybari
University of California, Riverside
hnagh001@ucr.edu

Ajaya Neupane
University of California, Riverside
ajaya@ucr.edu

Zhiyun Qian
University of California, Riverside
zhiyunq@cs.ucr.edu

Nael Abu-Ghazaleh
University of California, Riverside
nael@cs.ucr.edu

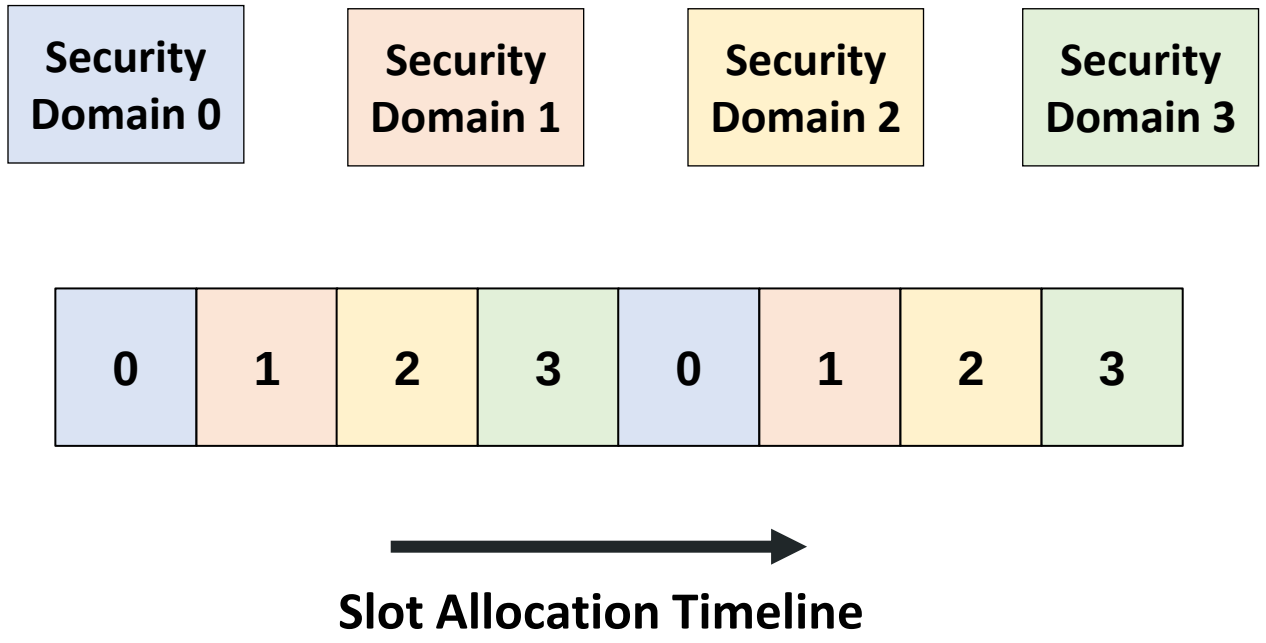
Timing Attack Example



The attacker uses its *own* latencies to leak information!

Static Partitioning in Time

Use a Round Robin, No-Skip Arbitration Policy



✓ **Secure**

Static partitioning, no leakage

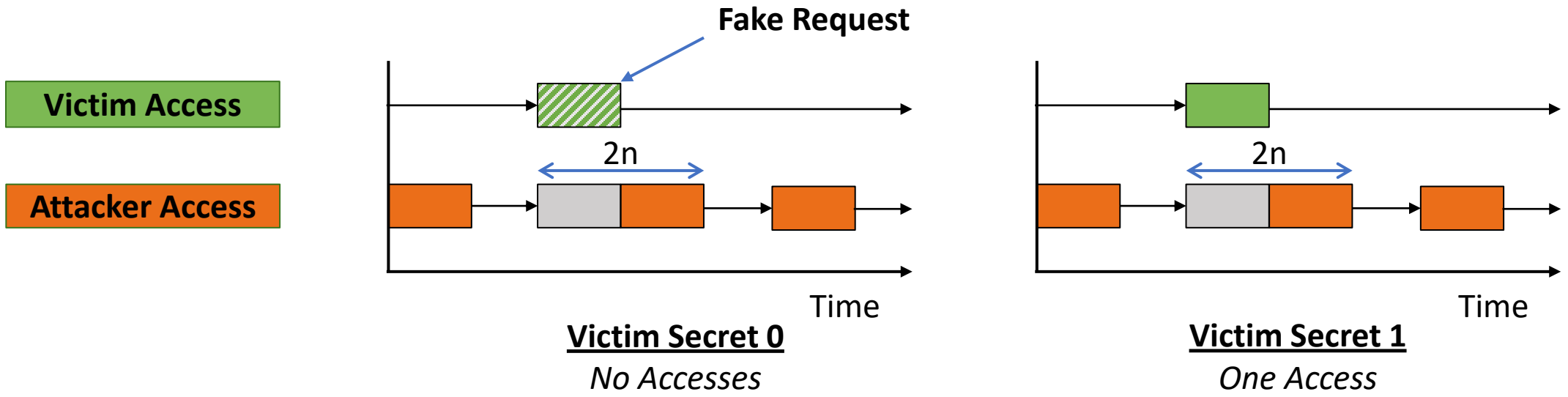
✗ **Bad Performance**

Poor bandwidth utilization!

Avoiding Information Leakage in the Memory Controller with Fixed Service Policies (Shafiee et al., Utah, ISCA 2015)

Traffic Shaping

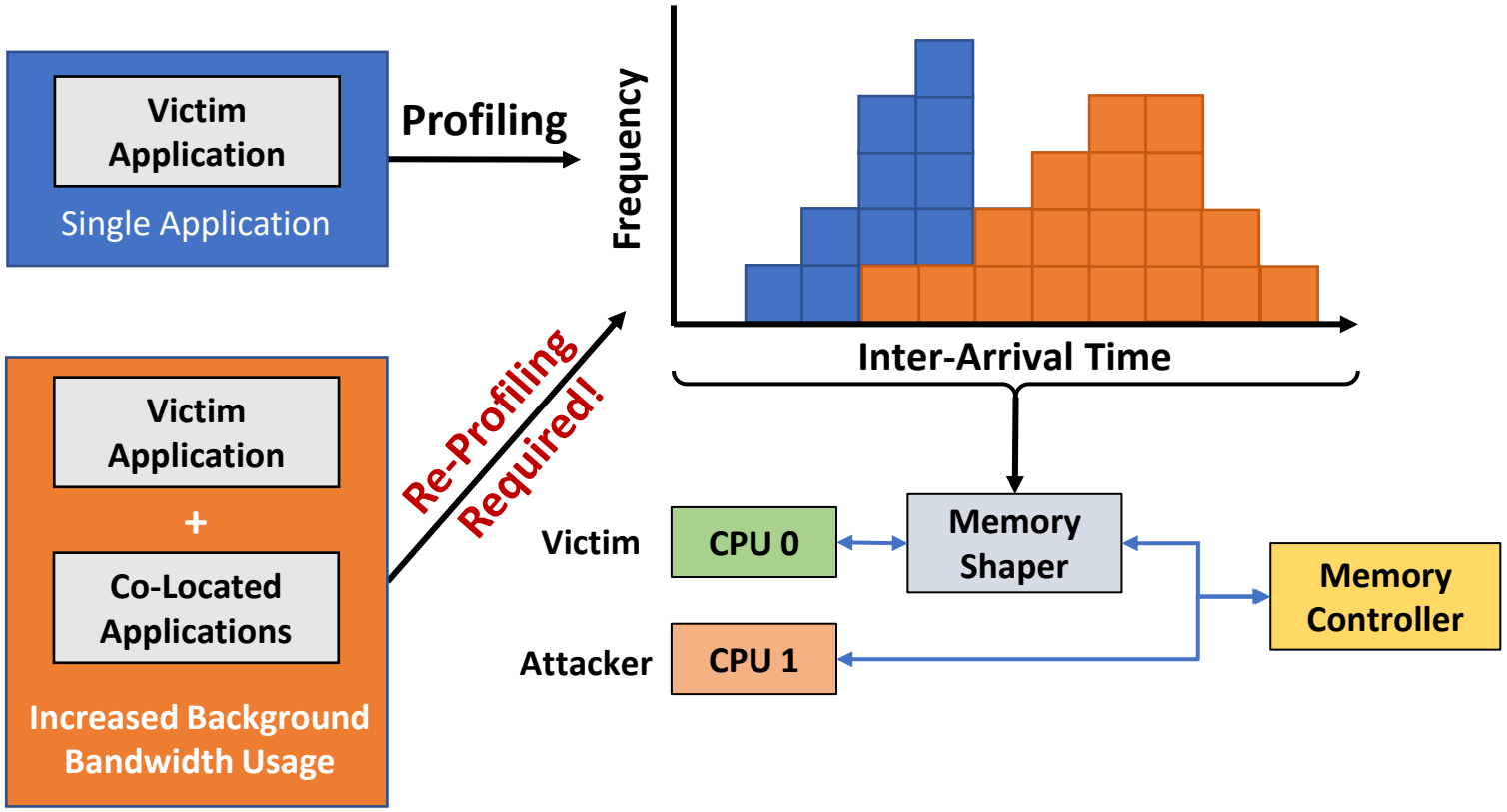
Shaping Strategy: Delay victim's existing requests and add fake requests



How do we do this for real applications without significant costs?

Camouflage's Traffic Shaping Strategy

Shape memory requests to a secret-independent *timing distribution*



✓ Good Performance

Dynamic sharing of the memory controller

✗ Insecure

Ordering or bank information can reveal the secret

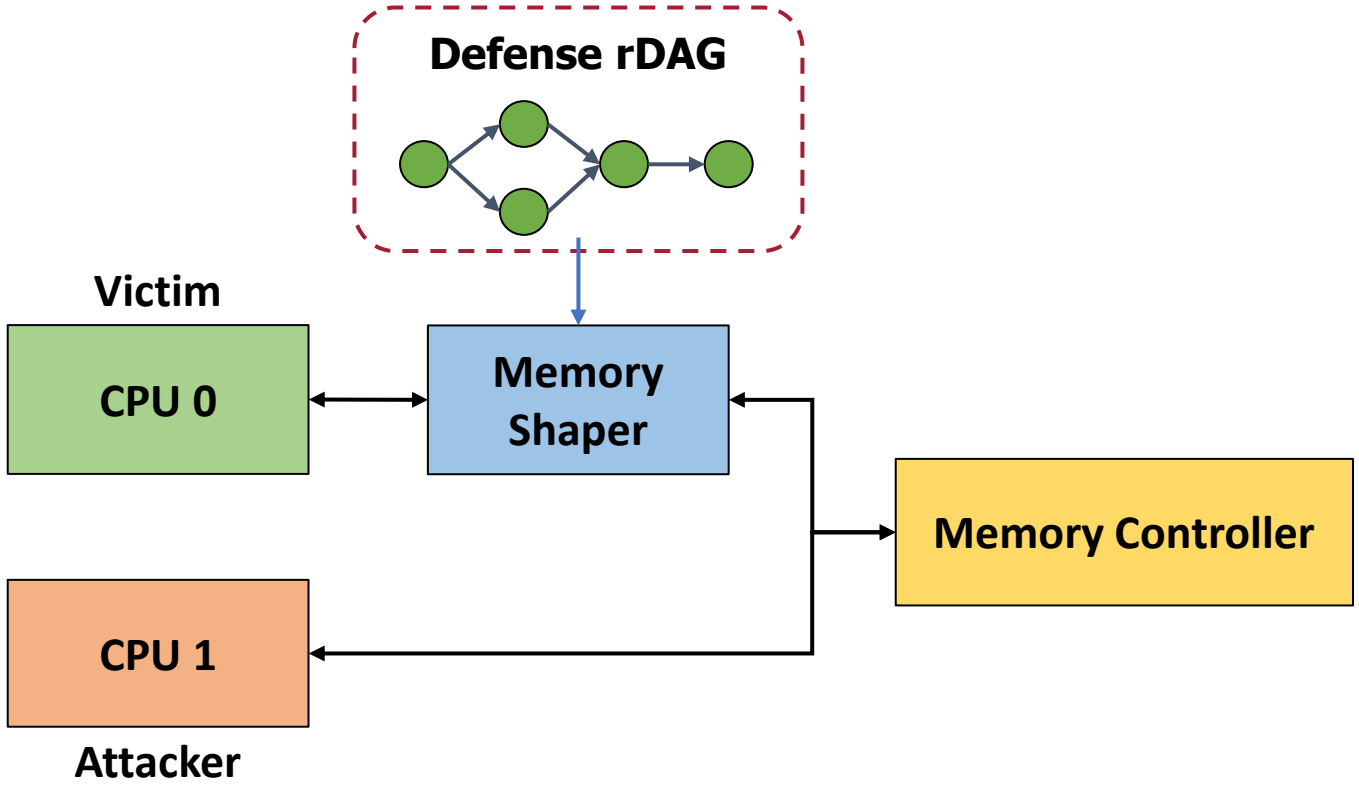
✗ Expensive Profiling

Ideal shaping distribution depends on co-running applications

Camouflage: Memory Traffic Shaping to Mitigate Timing Attacks (Zhou et al., Princeton, HPCA 2017)

DAGguise's Traffic Shaping Strategy

Shape memory requests to a secret-independent
Directed Acyclic Request Graph (rDAG)

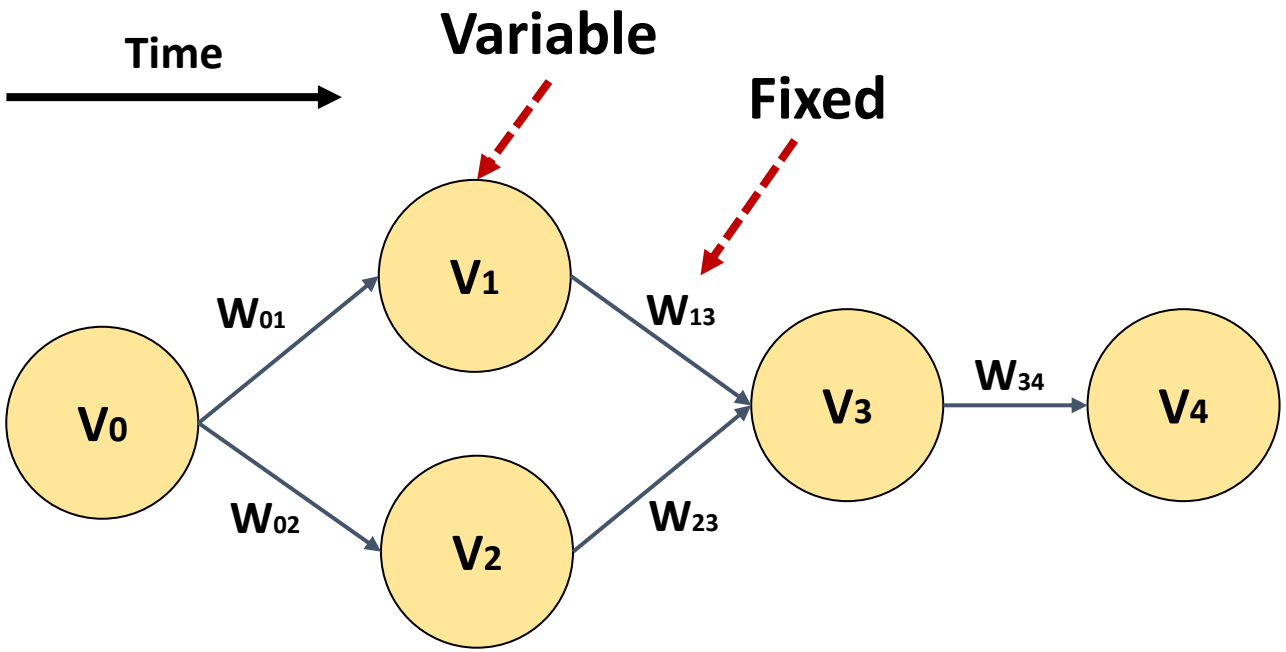


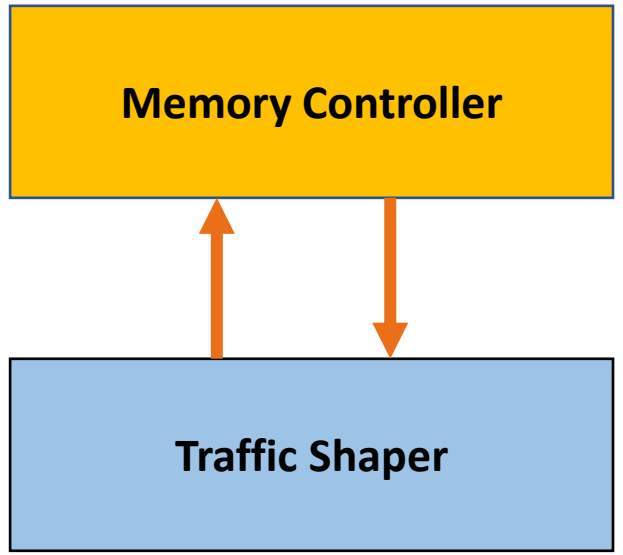
✓ Secure

✓ Good Performance

✓ Profile Victim Alone

Directed Acyclic Request Graphs





Vertices

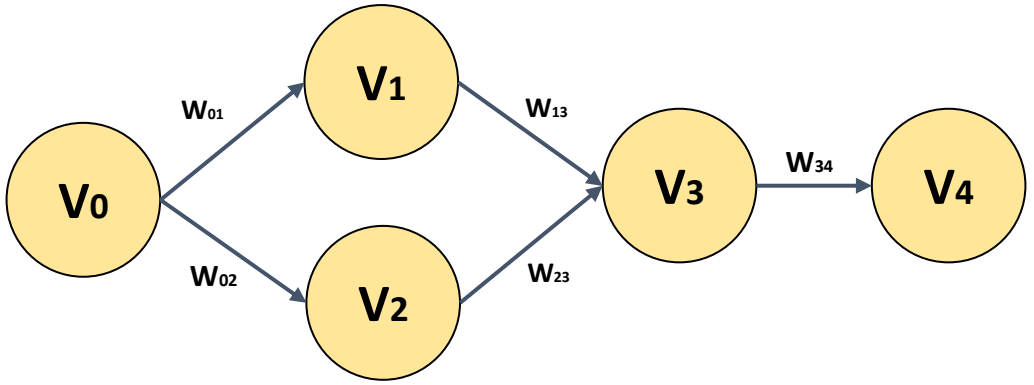
Memory requests with ***variable*** latency

Edges

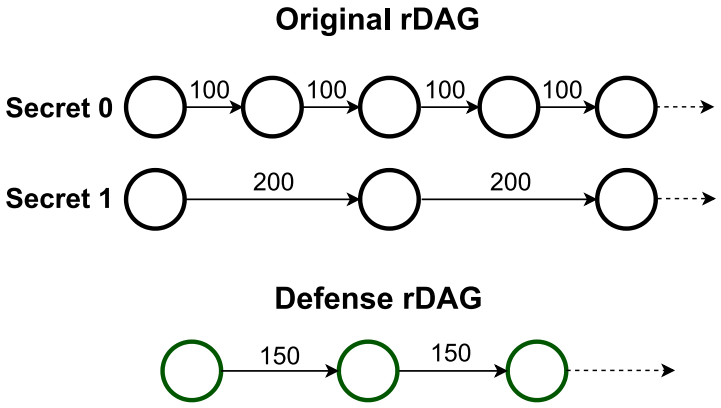
Dependencies between memory requests with ***fixed*** latency

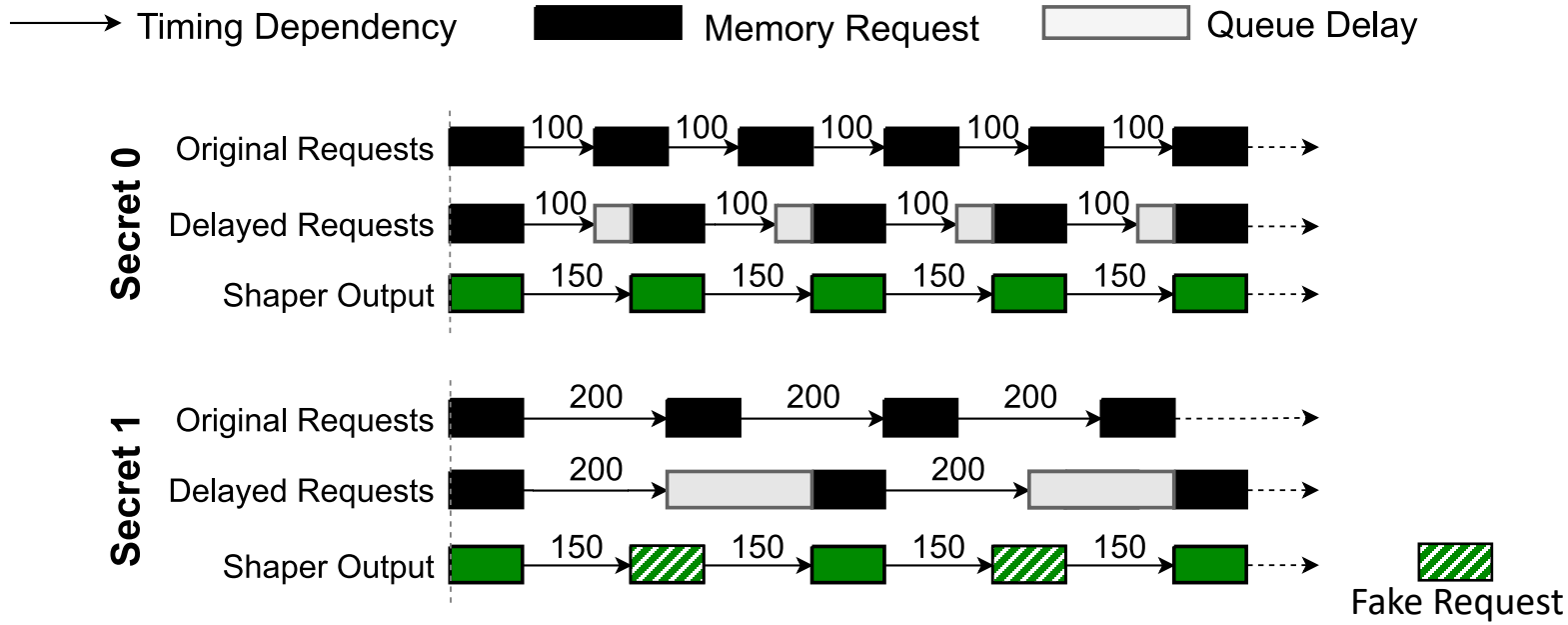
Why shape requests to an rDAG?

- ✓ **Security**
  - Shaping to a secret-independent defense rDAG makes victim request patterns *indistinguishable*
  - Defense rDAGs are public and are the only thing an attacker can recover
- ✓ **Performance**
  - Allows for *dynamic* sharing of memory resources in the memory controller
- ✓ **Profiling Cost**
  - Does not require knowledge of co-located applications



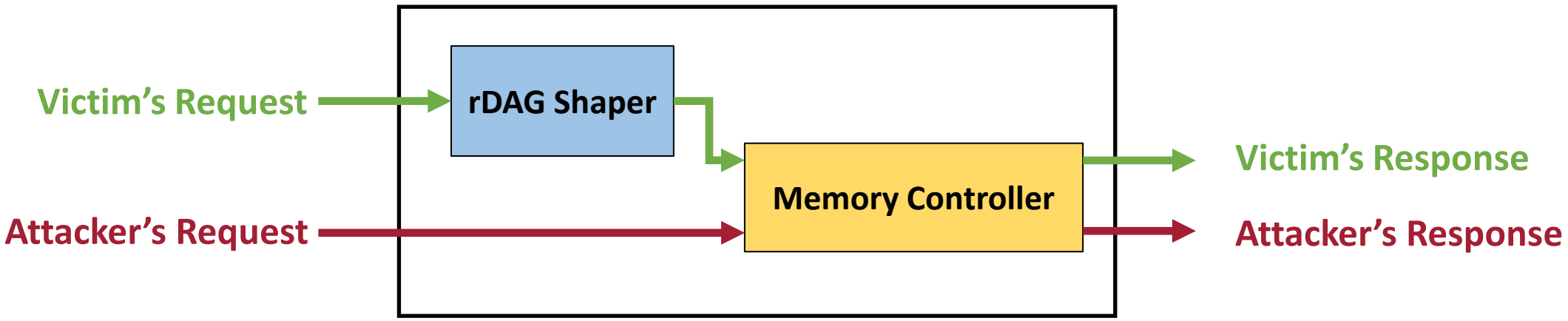
Simple Shaping Example





The shaper output is always the same, *no matter the secret*!

Indistinguishability Property



The attacker's observations should be *independent* from victim's request pattern

Indistinguishability Property

- Attacker's observation is independent from victim's request pattern
  - Given an attacker's request pattern, the attacker has an identical observation when contending with **ANY** victim's request pattern
  - This holds for **ANY** attacker's request pattern

Attacker's Observations when Contending with Victim

Victim Request Patterns / Attacker Request Patterns	A	B	C	… …
X	Attacker's Response Pattern X			

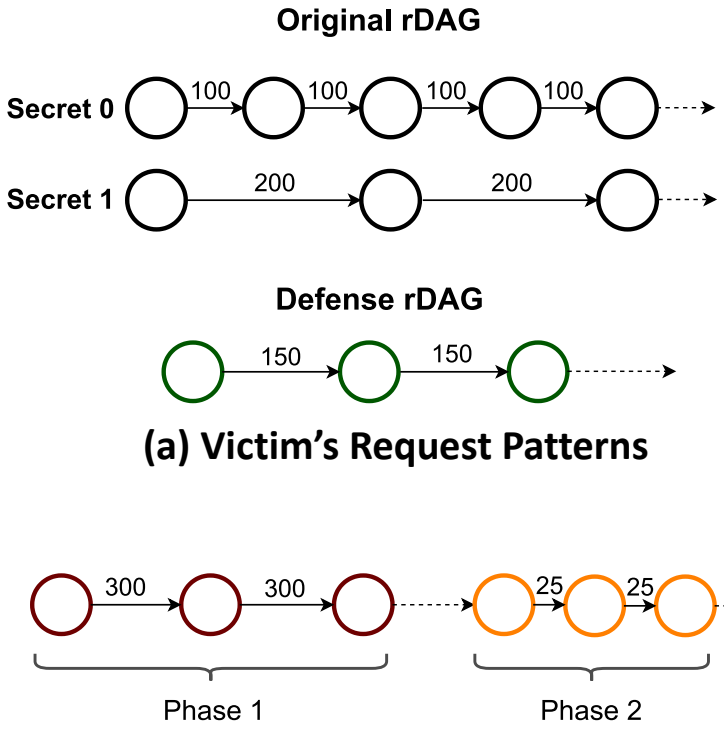
Formalization & Verification

- Formalize the indistinguishability property using state transitions

$$\mathrm{P}(S_0, n) := \ \forall\ \mathrm{Req}_{Tx}, \mathrm{Req}'_{Tx}, \ \forall\ \mathrm{Req}_{Rx}$$
$$\textbf{if}\ \ S_0 \xrightarrow[\mathrm{Req}_{Tx},\mathrm{Req}_{Rx}]{\mathrm{Resp}_{Tx},\mathrm{Resp}_{Rx}} S_n \ \ \text{and} \ \ S_0 \xrightarrow[\mathrm{Req}'_{Tx},\mathrm{Req}_{Rx}]{\mathrm{Resp}'_{Tx},\mathrm{Resp}'_{Rx}} S'_n$$
$$\textbf{then}\ \ \mathrm{Resp}_{Rx} = \mathrm{Resp}'_{Rx}$$

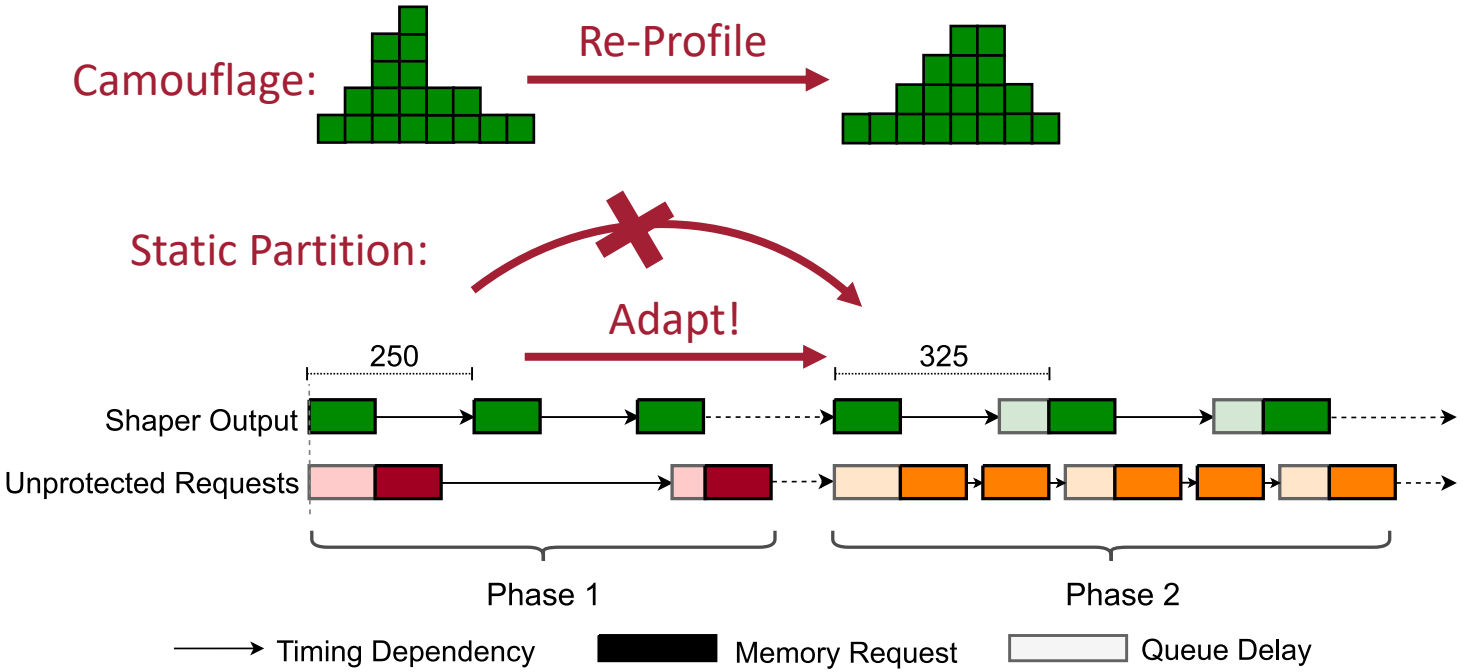
- Verification with *Rosette*:
  - First k cycles: symbolic execution
  - Arbitrary cycles: k-induction

rDAG Adaptivity



(a) Victim's Request Patterns

(b) Unprotected Program's Request Patterns

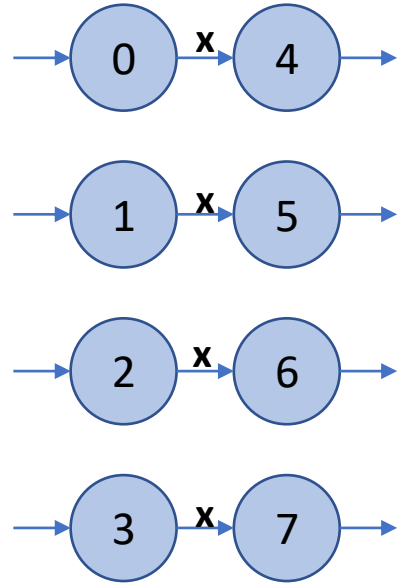


(c) Contention between Victim and Unprotected Program on Memory Controller

rDAG's adaptivity allows for better bandwidth utilization!

Offline Profiling Step

- Not for security, ***any*** secret-independent rDAG ensures security
- Low profiling cost
  - Victim is profiled ***alone***
  - Reduce search space by finding parameters for an rDAG *template*

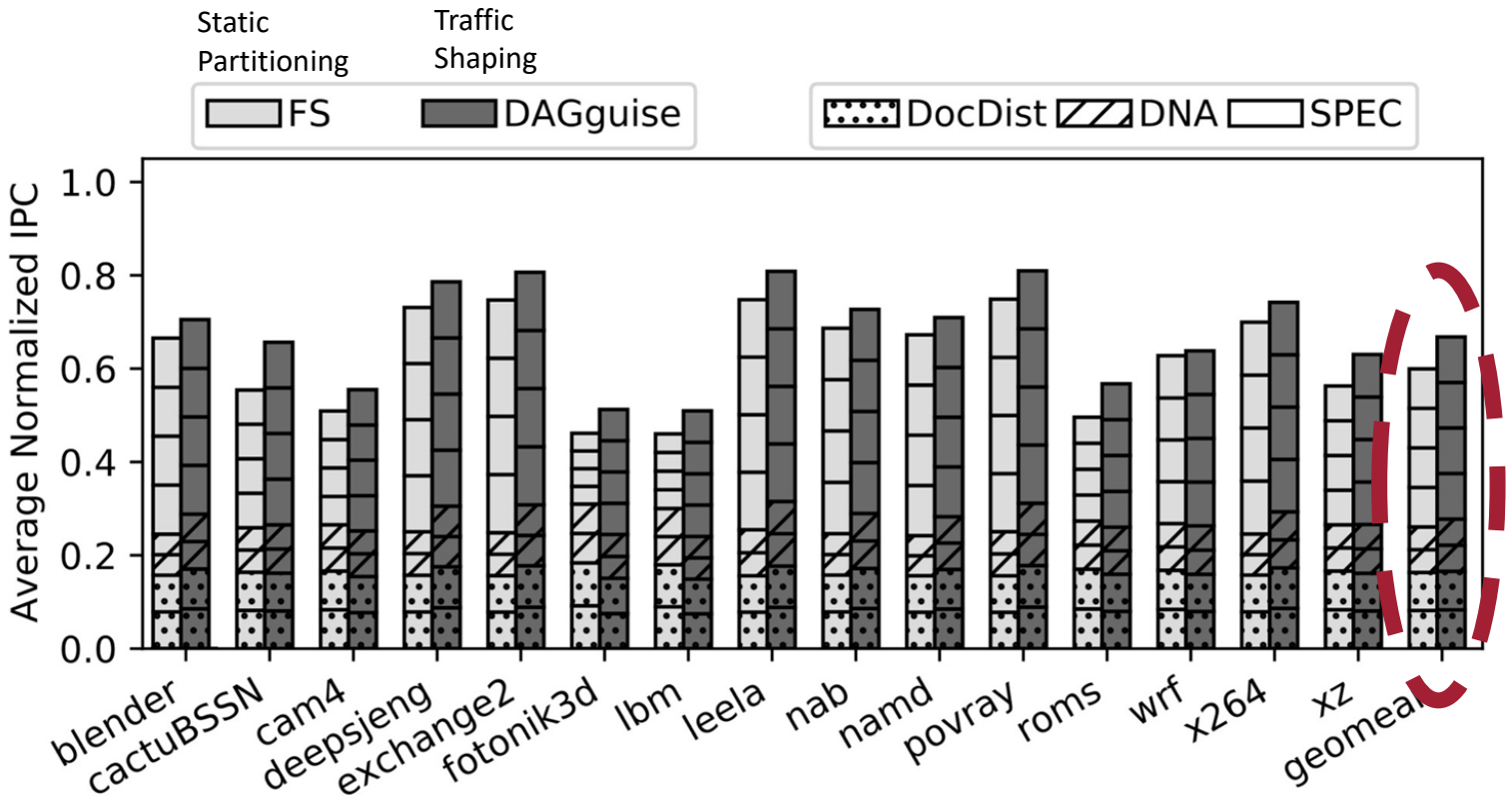


4-Parallel rDAG Template

Experimental Setup

- **Simulator:** gem5 and DRAMSim2
- **Architectural Specifications:**
  - 2 and 8 out-of-order CPU cores
  - 32KB L1i/d, 256kB L2, 1MB/core L3
- **Evaluated Configurations:**
  - DAGguise
  - Fixed Service (Bank Triple Alternation)
  - Baseline
- **Evaluated Applications:**
  - Unprotected SPEC benchmark(s) co-running alongside DAGguise protected application(s)

Experimental Results

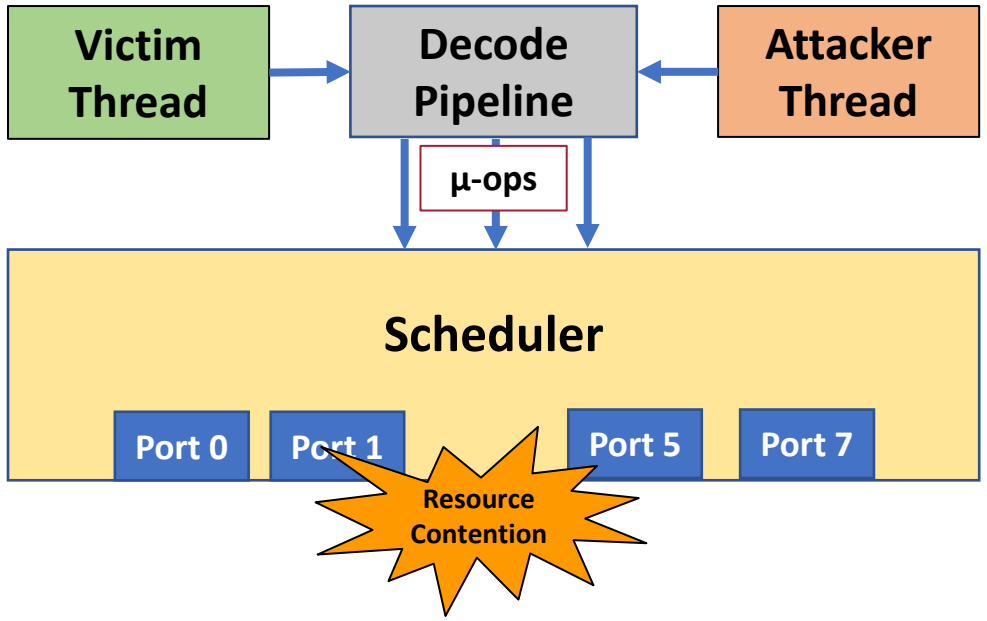


DAGguise's improves performance for both protected *and* unprotected applications!

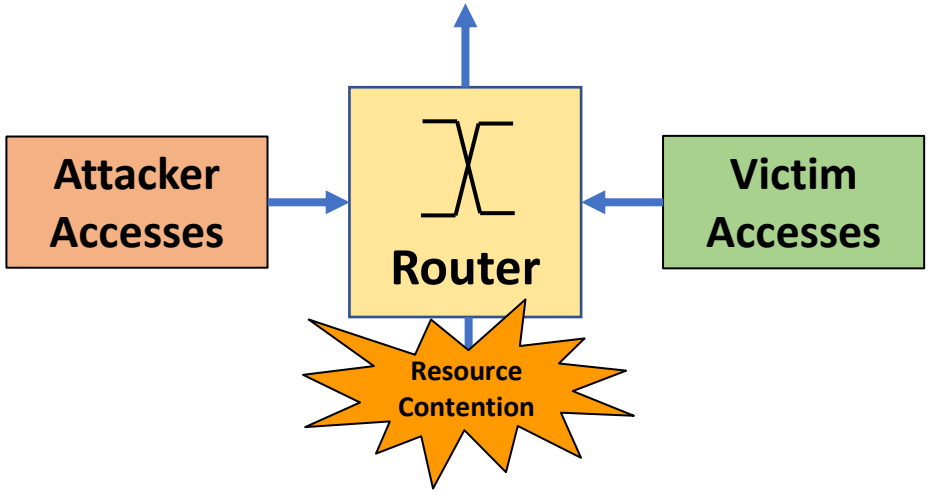
DAGguise achieves a 12% performance improvement over Fixed Service in an 8-CPU system

DAGguise Generalization

SMT Contention



Network on Chip Contention

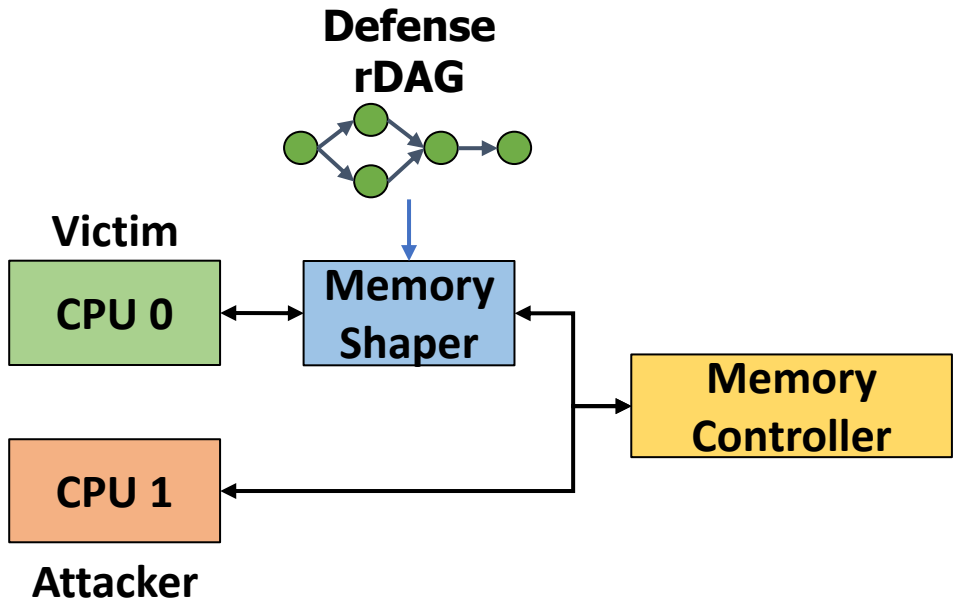


More in the Paper

- Implementation details of DAGguise shaper
- Formal security verification using symbolic execution and k-induction
- Detailed rDAG offline profiling process
- More performance and area overhead evaluation
- Generalizations to other scheduler-based side channels (e.g. port contention)

Conclusion

- DAGguise
  - A memory traffic shaper which:
    - Completely eliminates data leakage
    - Allows for dynamic contention
    - Requires only simple profiling
- rDAGs
  - A *general* and *adaptive* request representation
- A formal model of correctness using *Rosette*
- A generalized scheduler-based attack mitigation framework



DAGguise

Mitigating Memory Controller Side Channels

Peter W. Deutsch
pwd@mit.edu

Yuheng Yang
yuhengy@mit.edu

Thomas Bourgeat
bthom@mit.edu

Jules Drean
drean@mit.edu

Joel S. Emer
jsemer@mit.edu

Mengjia Yan
mengjiay@mit.edu