

# DAGguise

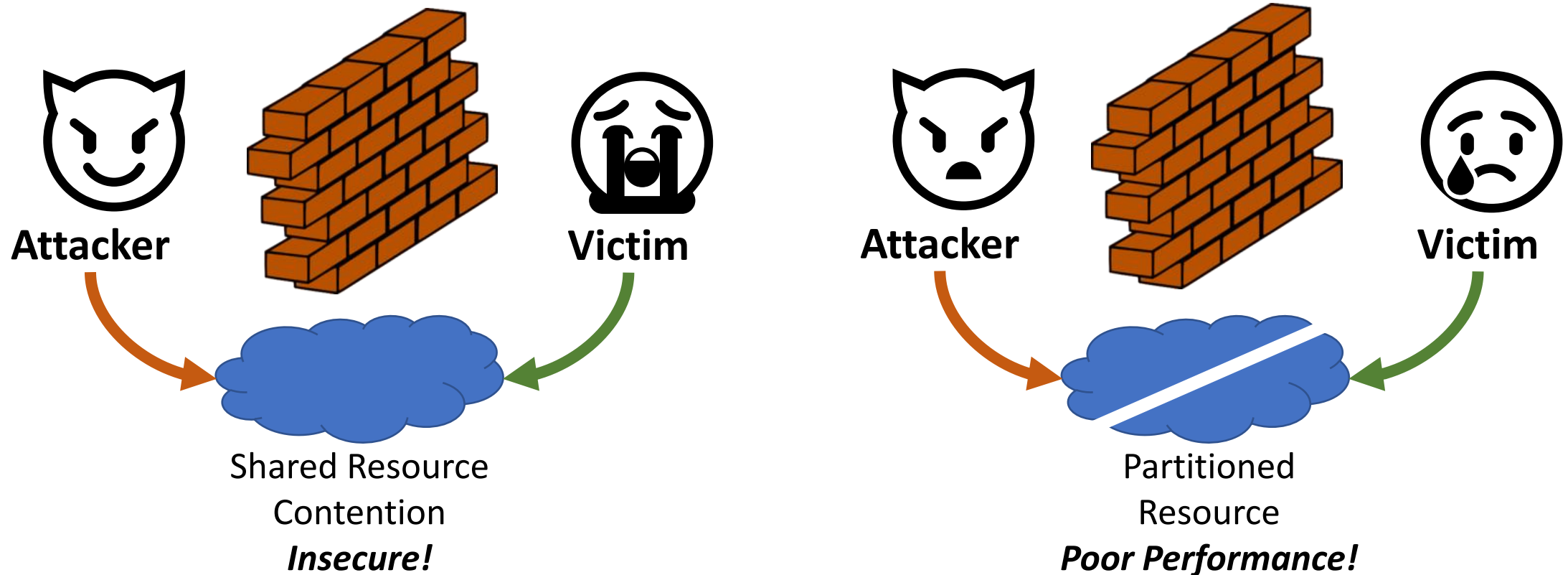
## Mitigating Memory Controller Side Channels

*Peter W. Deutsch\*, Yuheng Yang\*, Thomas Bourgeat,  
Jules Drean, Joel Emer, and Mengjia Yan*

ASPLOS 2022 (Session 3A)

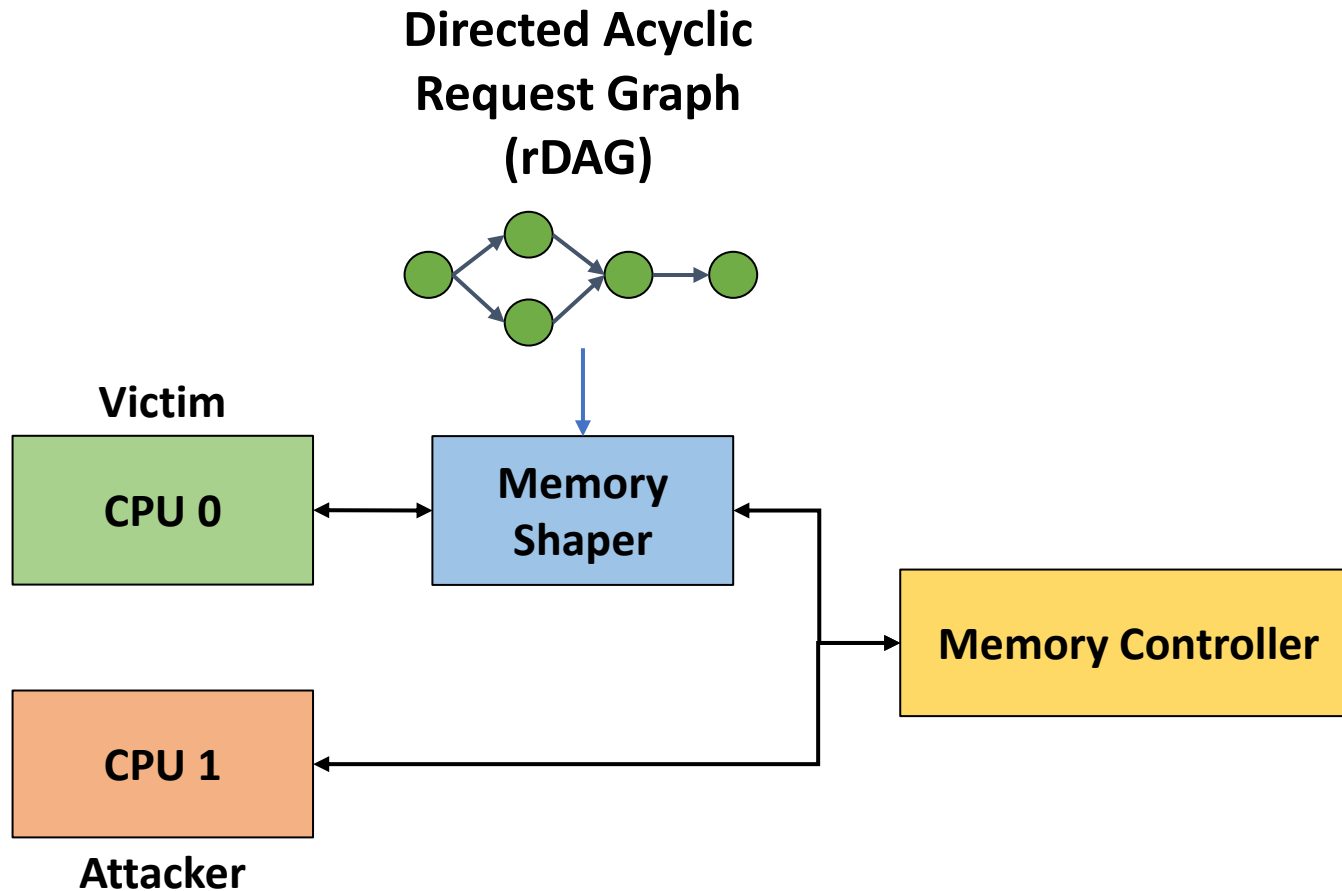


# Microarchitectural Side-Channels



**Key Defense Tradeoff: Security vs. Performance**

# DAGguise Key Idea



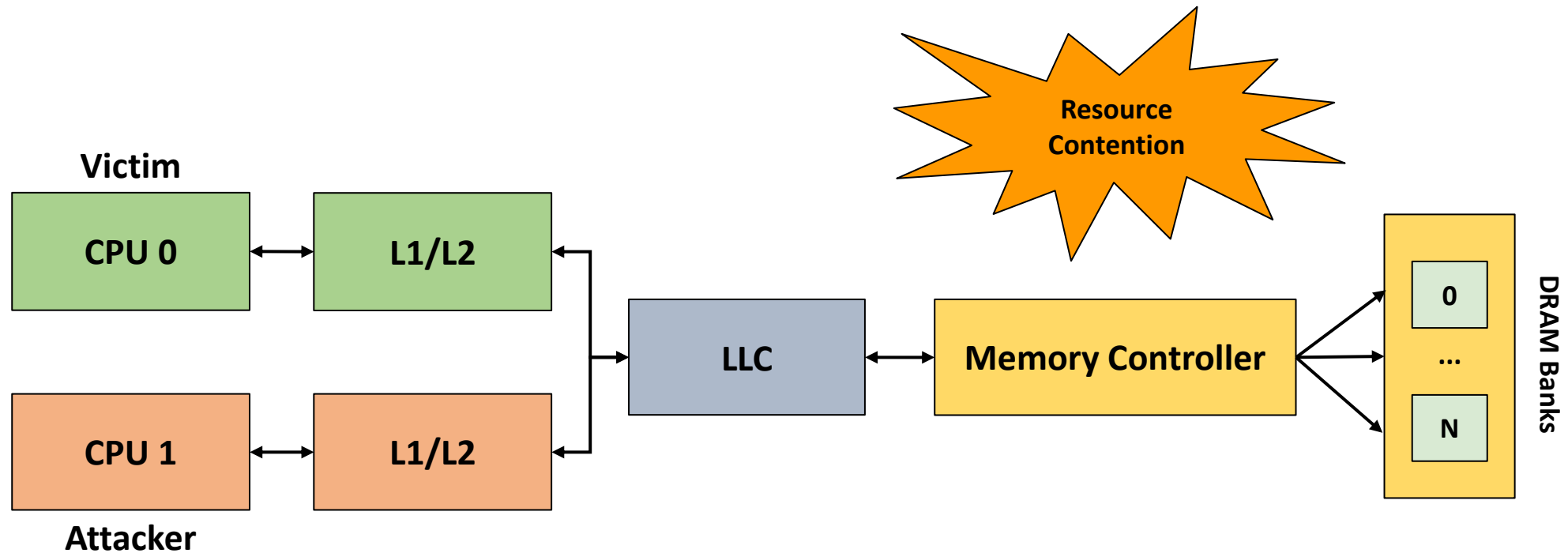
**DAGguise achieves:**

- ✓ 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

## 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



We introduce the first attacks that leverage cont... There are two challenge... exploit this channel. Fir... connect's functioning a... that can be learned by a... noisy by nature and has... the first challenge, we pe... of the sophisticated prot... the ring interconnect. W... core covert channel over... of over 4 Mbps from a s... cross-core channel not... the second challenge, w... patterns of ring conten... We demonstrate our atta... able EdDSA and RSA i... the precise timing of ke...

### 1 Introduction

Modern computers use... eral heterogeneous, inte... across computing units... fered significant benefi... created an opportunity... croarchitectural features... of software-based cover... Through these attacks, i... fects (e.g., timing variat... nel case) or infer a vic... channel case). These at... ample, many cache-ba... demonstrated that can l... trographic keys) in clo... web browsers [40, 61, 77]

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

## 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 Ramrakhyan  
ARM Research  
prakash.ramrakhyan@arm.com

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

Abstract—Accelerators used for machine learning (ML) infer...

ence provide great perf... onfidential model in i... attacks is critical in har... practice. Data and memo... proposed to defend agai...

In this paper, we demo... interface between accel... a side-channel for leaki... This side channel is ind... even in the presence of d... can be monitored throug... contention from an on-...

1.

Deep learning model... domain-specific comput... inference accelerators i... (NPU) are being develop... academic [8], [17], [19], [20]. system-on-chip (SoC) | Inference-as-a-service | providers like Amazon | on ML accelerators. Th... host trained models on... confidential user data li... data like disease classifi...

From the security pe... the cloud provider to... parameters as well as t... [18], [20] show how ke... used to steal a victim's | similar accuracy. An at...

2019 IEEE Symposium on Security and Privacy

## Port Contention for Fun and Profit

Alejandro Cabrera Adaya\*, Billy Bob Brumley†, Sobaib ul Hassan†, Cesar Pereira Garcia†, Nicola Turveri†  
\*Universidad Tecnológica de la Habana (CUJAE), Habana, Cuba  
†Tampere University, Tampere, Finland

Abstract—Simultaneous Multithreading (SMT) architectures are attractive targets for side-channel enabled attackers, with their inherently broader attack surface that exposes more per-physical core microarchitectural components than cross-core attacks. In this work, we explore SMT execution engine sharing as a side-channel leakage source. We target ports to stacks of execution units to create a high-resolution timing side-channel due to port contention, inherently stealthy since it does not depend on the memory subsystem like other cache or TLB based attacks. Implementing our channel on Intel Skylake and Katy Lake architectures featuring Hyper-Threading, we mount an end-to-end attack that recovers a P-384 private key from an OpenSSL-powered TLS server using a small number of repeated TLS handshake attempts. Furthermore, we show that traces targeting shared libraries, static builds, and SGX enclaves are essentially identical, hence our channel has wide target application.

sets, machine learning techniques, nor reverse engineering techniques.

To demonstrate PORTSMASH in action, we present a complete end-to-end attack in a real-world setting attacking the NIST P-384 curve during signature generation in a TLS server compiled against OpenSSL 1.1.0h for crypto functionality. Our Spy program measures the port contention delay while executing in parallel to ECDSA P-384 signature generation, creating a timing signal trace containing a noisy sequence of add and double operations during scalar multiplication. We then process the signal using various techniques to clean the signal and reduce errors in the information extracted from each trace. We then pass this partial key information to a recovery phase, creating lattice problem instances which ultimately yield the TLS server's ECDSA private key.

We extend our analysis to SGX, showing it is possible to

1. INTRODUCTION

## DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks

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

In cloud computi... often co-located at... Thus, preventing in... crucial. While the... tion, shared hardwa... ory bus, can leak se... sons, shared memo... abled. Furthermore... CPU. In this setting... a slow cross-CPU o... known. In contrast... channel as well as | across processors a... build these attacks... address mappings.

present two i... if memory ad... links. One u... be other runs... id. Using this... a novel class... that is share... our attacks v... First, we be... to 2 Mbps, w... faster than m... ld a side-chat... locate and m... how using the... s and in parti... on DDR4.

### Introduction

the populari... the same p... machines (VM

Association

Association

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

## 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

Amil Kurmus†  
IBM Research - Zurich

### ABSTRACT

Spectre, Meltdown, hypervisor are prone to i... weaknesses, i... applications, i... code, may be... these attacks | We introdu... that leverages... processors (SM... victim proces... contention by... world gadget... locates SMO... glite, we fou... mation. Final... the (OpenSSL... bits, and agai... OpenSSL libe... the plaintext |

### CCS CON

Security an... measures.

### KEYWORD

side-channel;... attack; micro

To whom corres...

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

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

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

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

### ABSTRACT

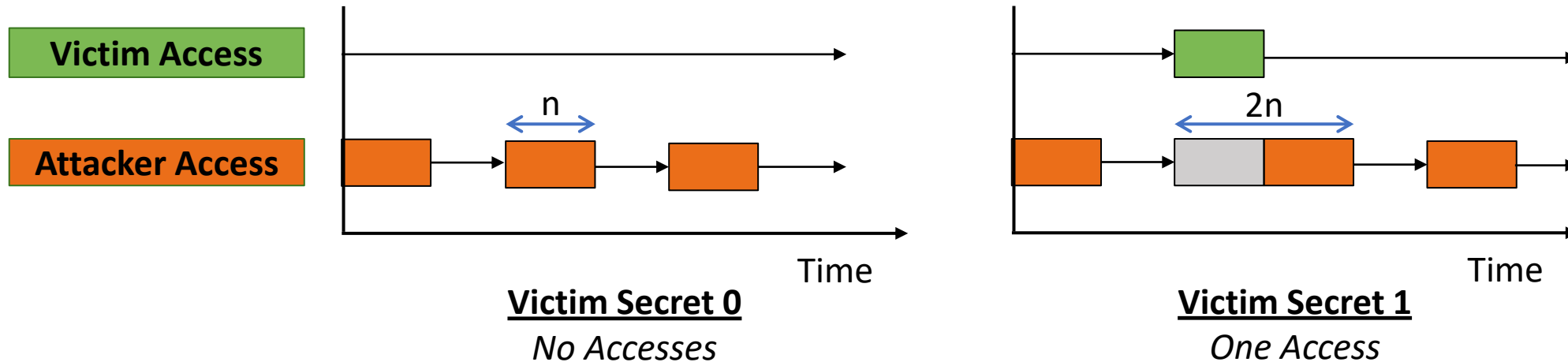
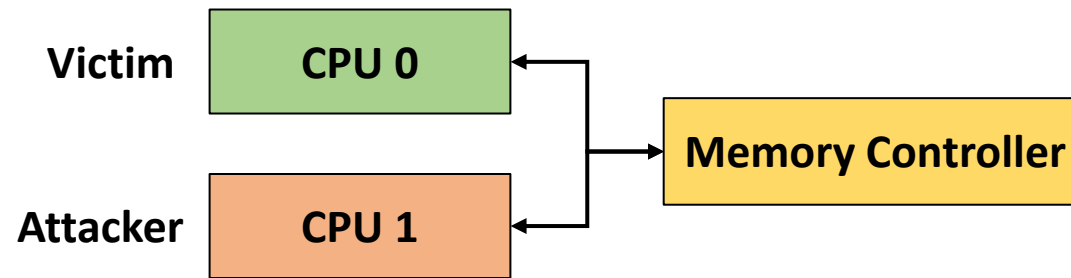
Graphics Processing Units (GPUs) are commonly integrated with computing devices to enhance the performance and capabilities of graphical workloads. In addition, they are increasingly being integrated in data centers and clouds such that they can be used to accelerate data intensive workloads. Under a number of scenarios the GPU can be shared between multiple applications at a fine granularity allowing a spy application to monitor side channels and attempt to infer the behavior of the victim. For example, OpenGL and WebGL send workloads to the GPU at the granularity of a frame, allowing an attacker to interleave the use of the GPU to measure the side-effects of the victim computation through performance counters or other resource tracking APIs. We demonstrate

### 1 INTRODUCTION

Graphics Processing Units (GPUs) are integral components to most modern computing devices, used to optimize the performance of today's graphics and multi-media heavy workloads. They are also increasingly integrated on computing servers to accelerate a range of applications from domains including security, computer vision, computational finance, bio-informatics and many others [52]. Both these classes of applications can operate on sensitive data [25, 31, 57] which can be compromised by security vulnerabilities in the GPU stack.

Although the security of GPUs is only starting to be explored, several vulnerabilities have already been demonstrated [46, 49, 55, 58, 63, 71, 74]. Most related to this paper, Luo et al. demonstrated a

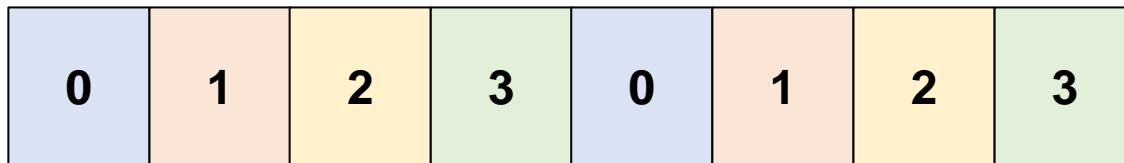
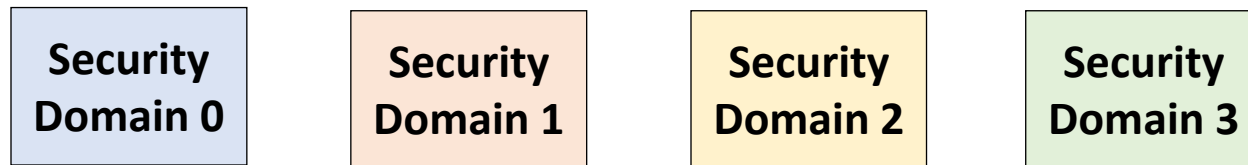
# Timing Attack Example



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

# Static Partitioning in Time

Use a Round Robin, No-Skip Arbitration Policy



→  
Slot Allocation Timeline

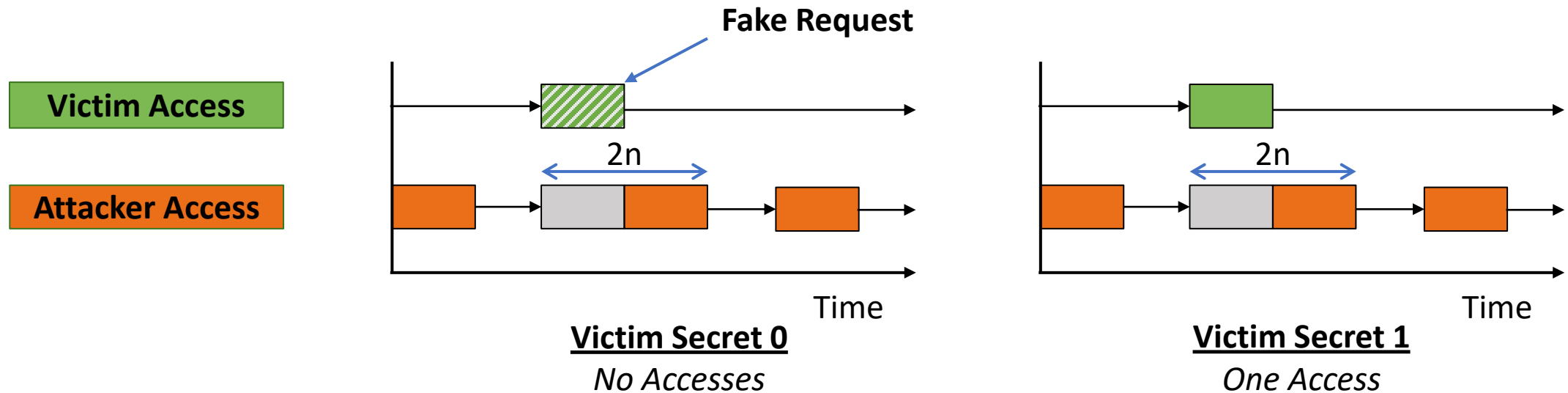
✓ Secure  
Static partitioning, no leakage

✗ Bad Performance  
Poor bandwidth utilization!



# Traffic Shaping

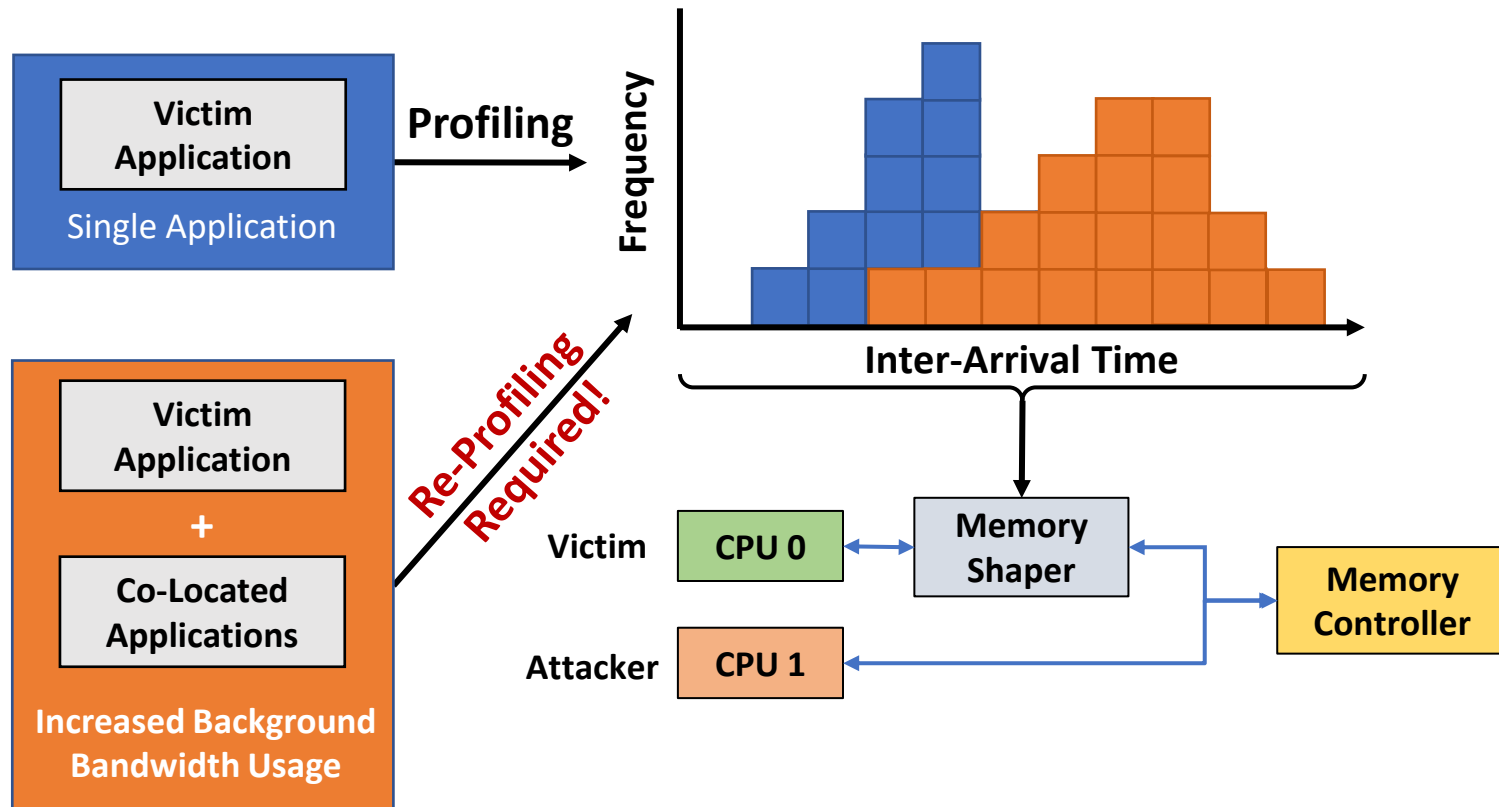
**Shaping Strategy:** Delay victim's existing requests and add fake requests



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

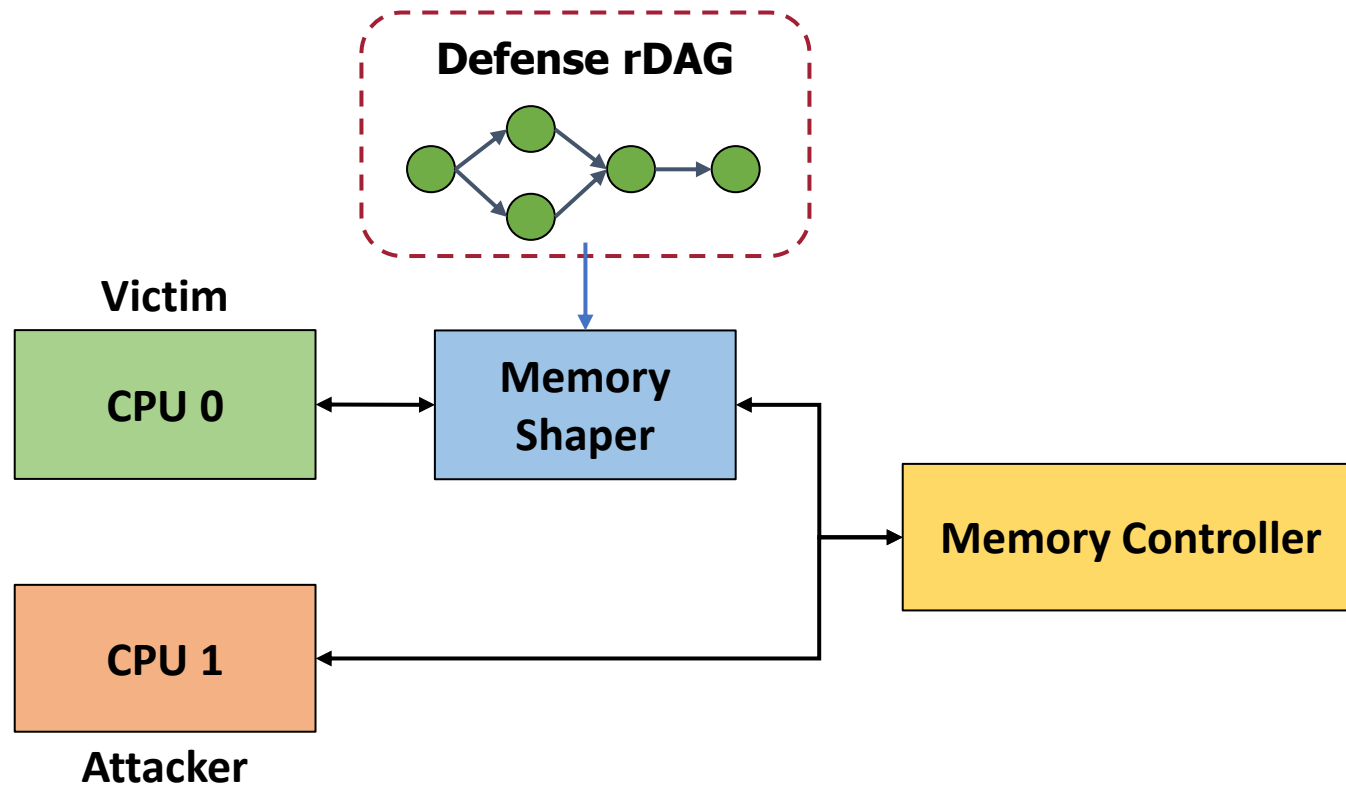
Ordering or bank information can reveal the secret

## ✗ Expensive Profiling

Ideal shaping distribution depends on co-running applications

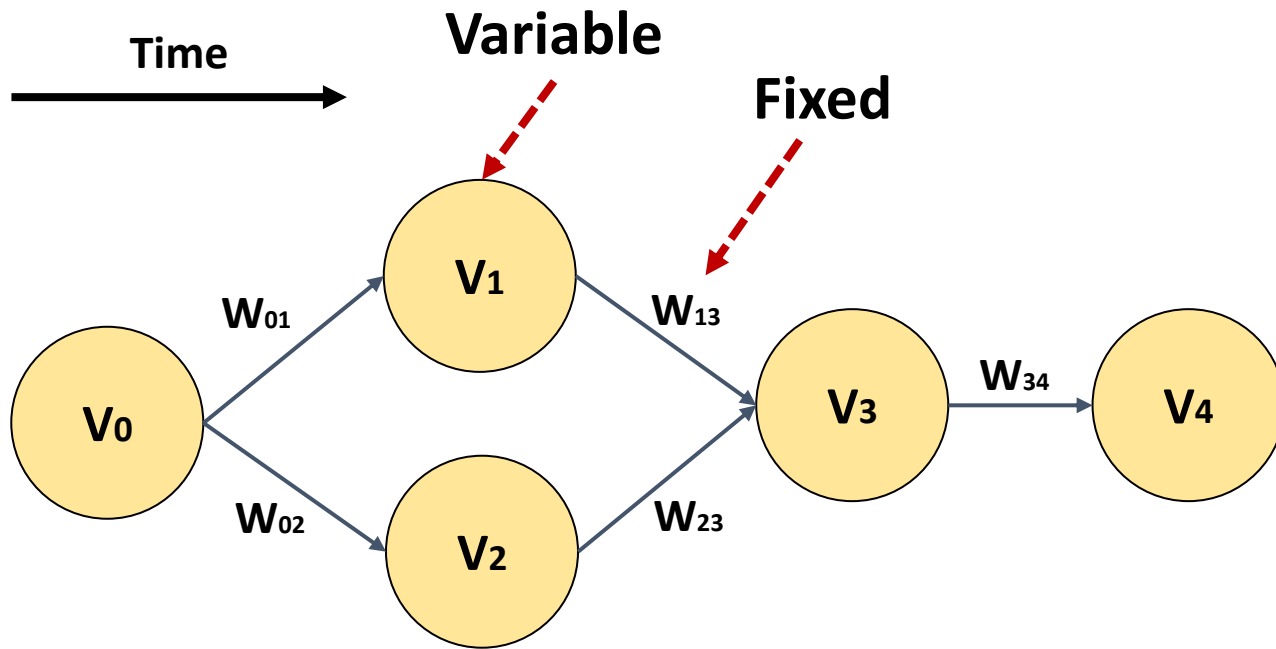
# DAGguise's Traffic Shaping Strategy

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



- ✓ Secure
- ✓ Good Performance
- ✓ Profile Victim Alone

# Directed Acyclic Request Graphs

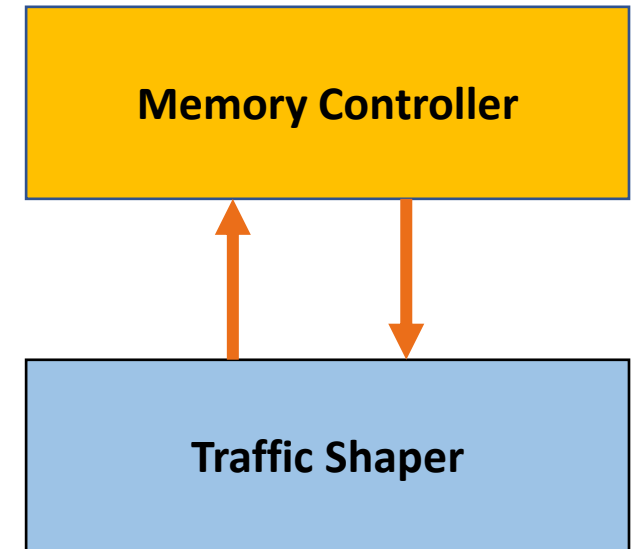


## Vertices

Memory requests with *variable* latency

## Edges

Dependencies between memory requests with *fixed* latency



# Why shape requests to an rDAG?

## ✓ Security

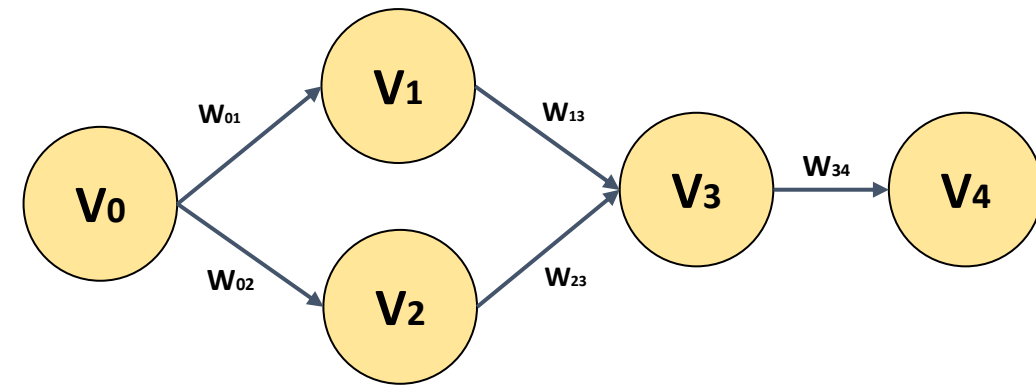
- Shaping to a secret-independent defense rDAG makes victim request patterns *indistinguishable*
- Defense rDAGs are public and are the only thing an attacker can recover

## ✓ Performance

- Allows for *dynamic* sharing of memory resources in the memory controller

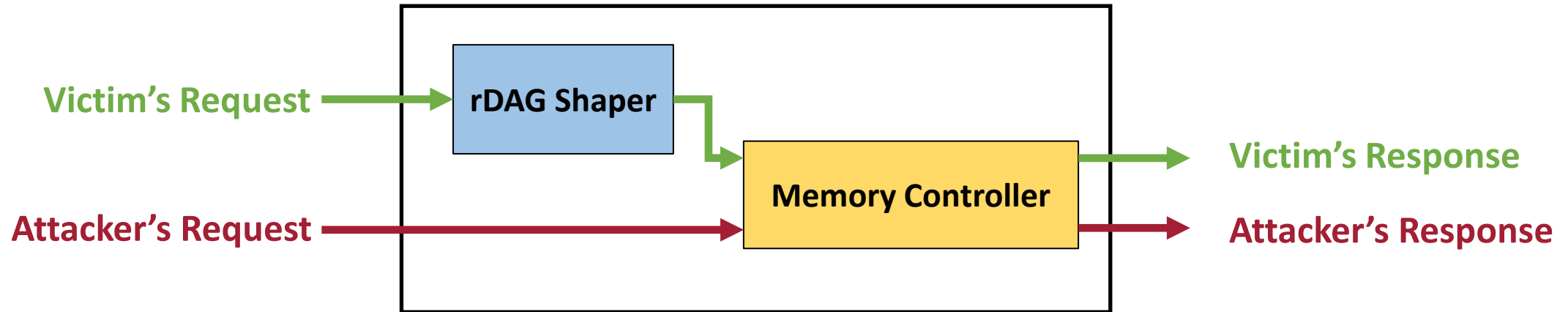
## ✓ Profiling Cost

- Does not require knowledge of co-located applications





# 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	... ..
	Attacker's Response Pattern X			
X				



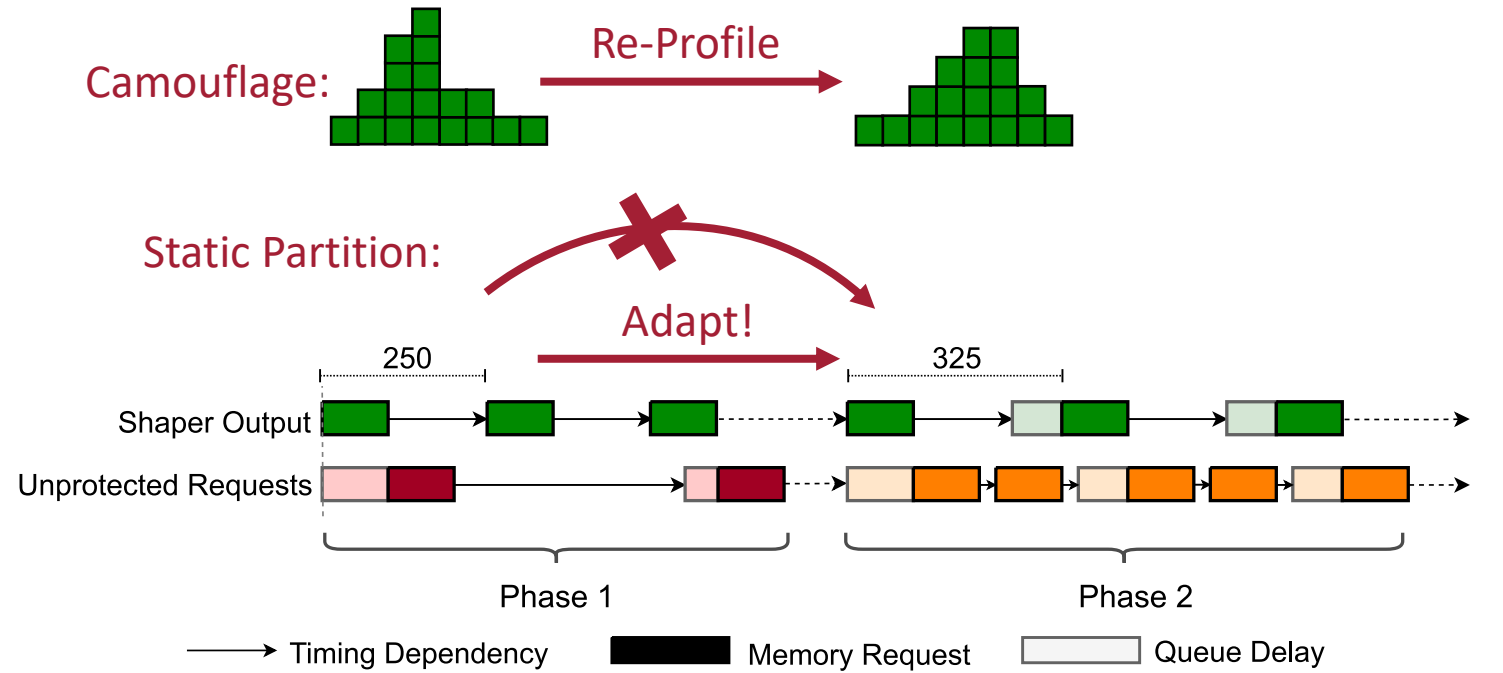
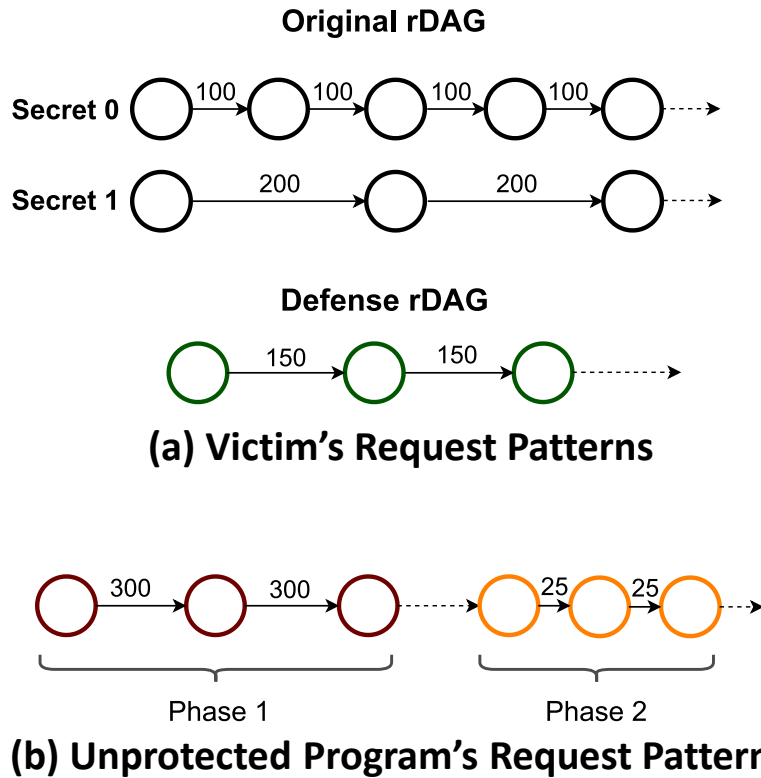
# Formalization & Verification

- Formalize the indistinguishability property using state transitions

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

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

# rDAG Adaptivity

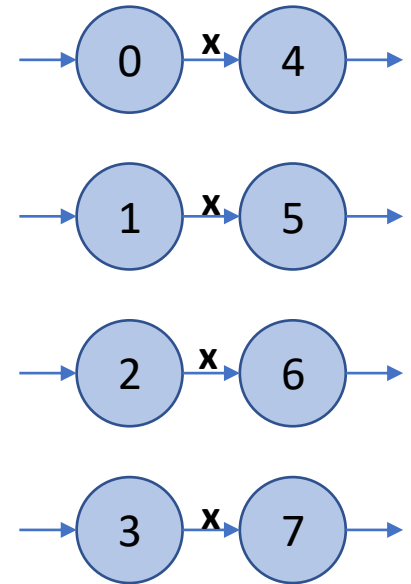


**(c) Contention between Victim and Unprotected Program on Memory Controller**

**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*



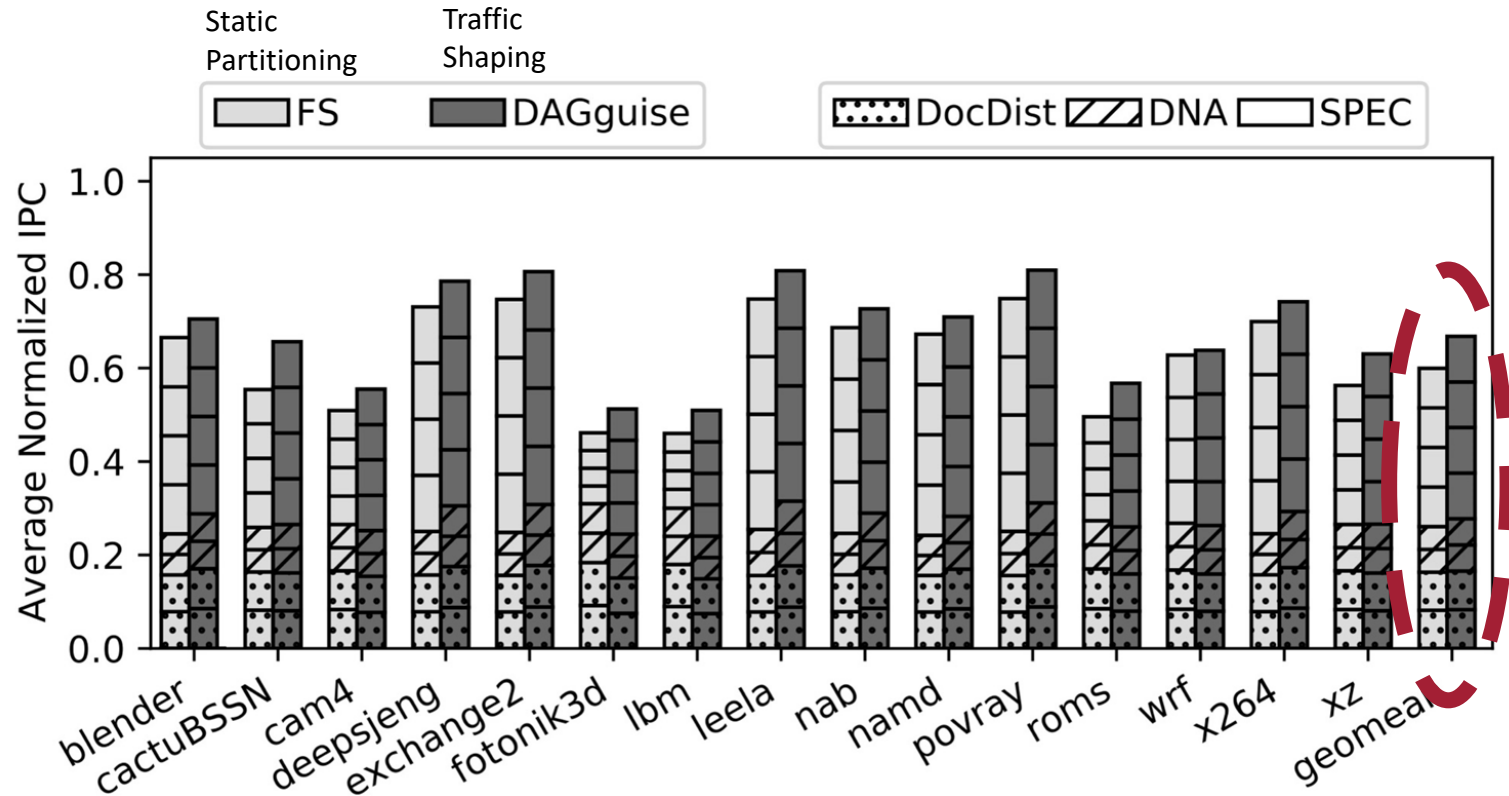
**4-Parallel rDAG Template**

# 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

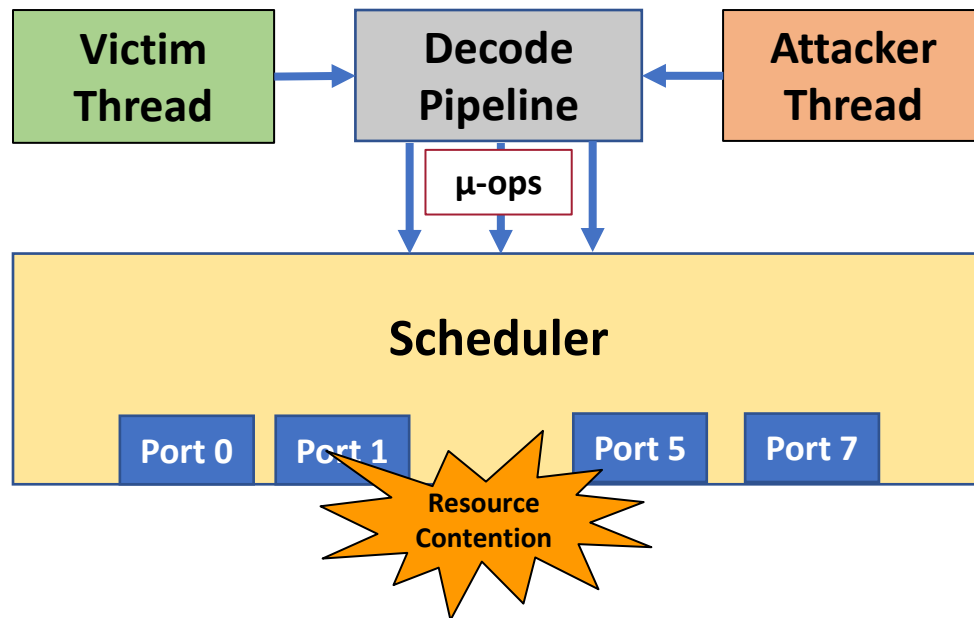


**DAGguise's improves performance for both protected *and* unprotected applications!**

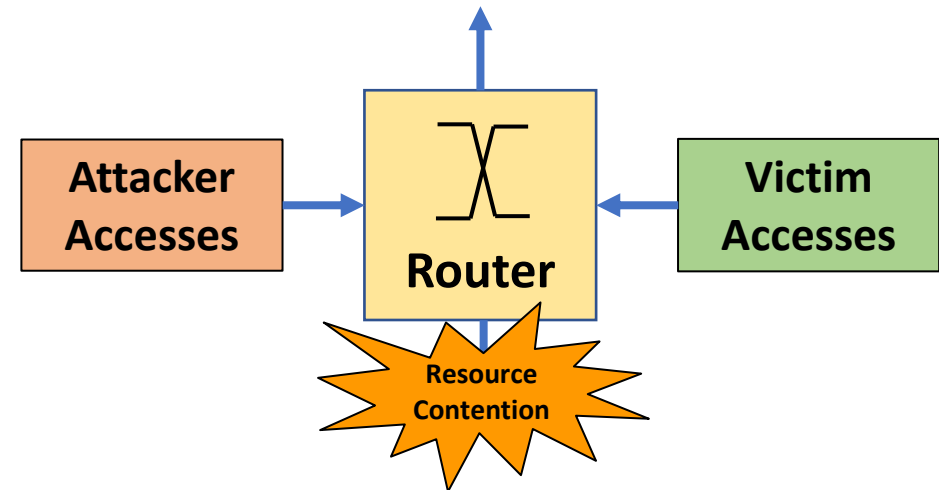
**DAGguise achieves a 12% performance improvement over Fixed Service in an 8-CPU system**

# DAGguise Generalization

## SMT Contention



## Network on Chip Contention

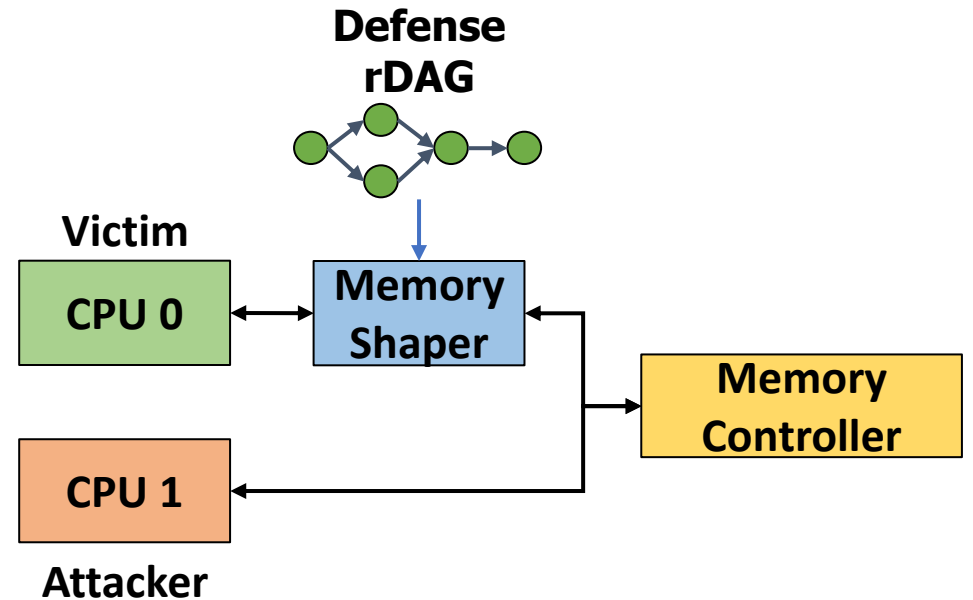


# More in the Paper

- 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](mailto:pwd@mit.edu)

**Yuheng Yang**

[yuhengy@mit.edu](mailto:yuhengy@mit.edu)

Thomas Bourgeat

[bthom@mit.edu](mailto:bthom@mit.edu)

Jules Drean

[drean@mit.edu](mailto:drean@mit.edu)

Joel S. Emer

[jsemer@mit.edu](mailto:jsemer@mit.edu)

Mengjia Yan

[mengjiay@mit.edu](mailto:mengjiay@mit.edu)

