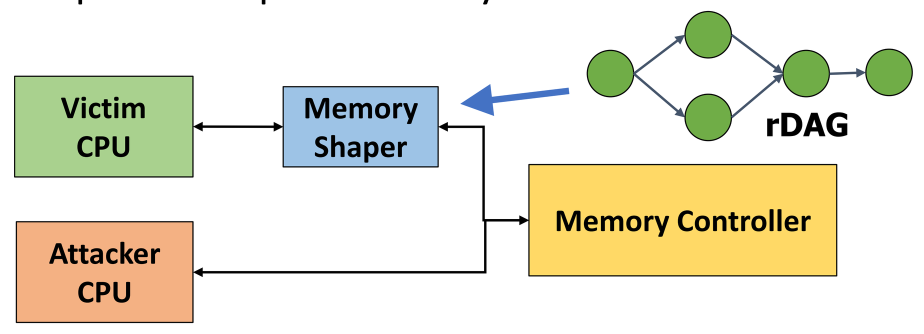
DAGguise: Mitigating Memory Controller Side Channels

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

1. SUMMARY

Problem: Contention in the memory controller can cause information leakage from a victim to an attacker

Our Solution: Shape the victim's memory traffic into a secret-independent pattern represented by an *rDAG*



Evaluation: Compared to the state-of-the-art, DAGguise achieves better security, performance, and has a lower profiling cost

	Previous Solutions		
	Fixed Service	Camouflage	DAGguise
Security	✓	✗	✓
Performance	✗	✓	✓
Profiling Cost	✓	✗	✓

Formally Verified

Profile Without Co-located Apps

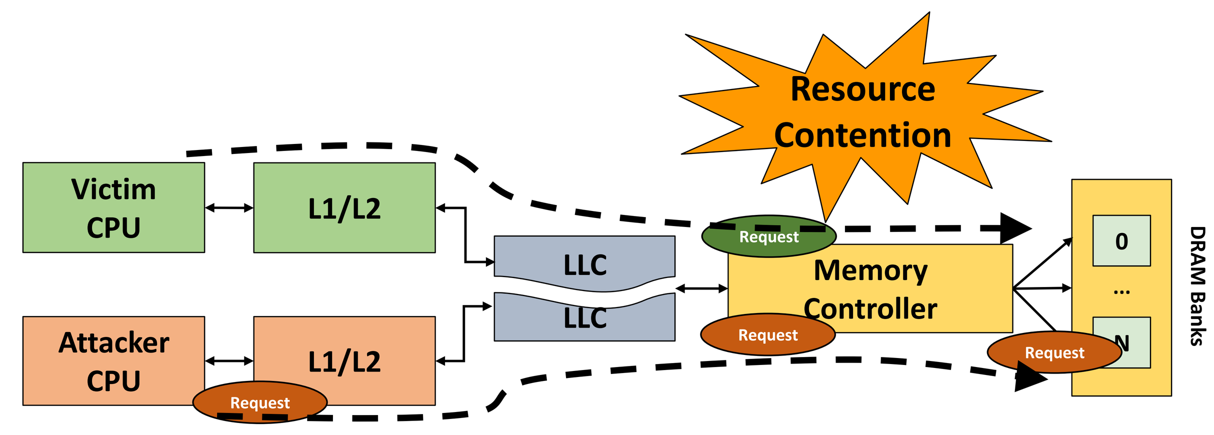
12% Speedup

Generalization: Can be extended to other scheduler-based side channels, ex:
- SMT Port Contention
- Network on Chip Contention

2. PROBLEM

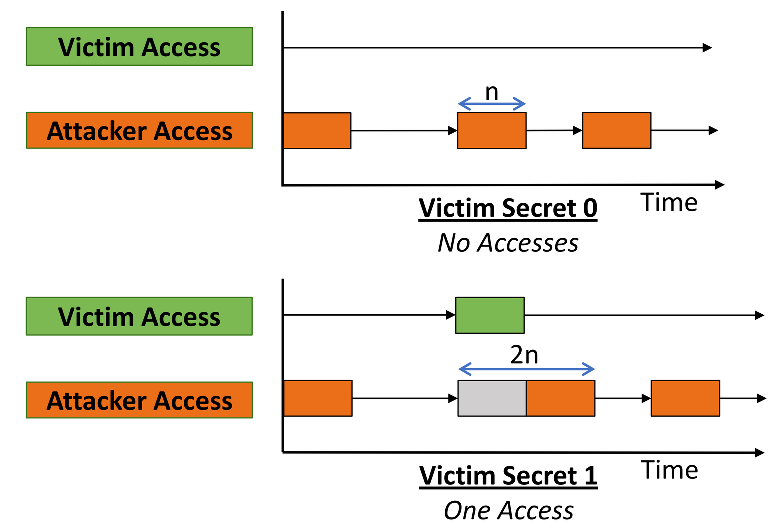
Memory Controller Side Channel

Victim's and attacker's memory requests contend with each other



Attack Example

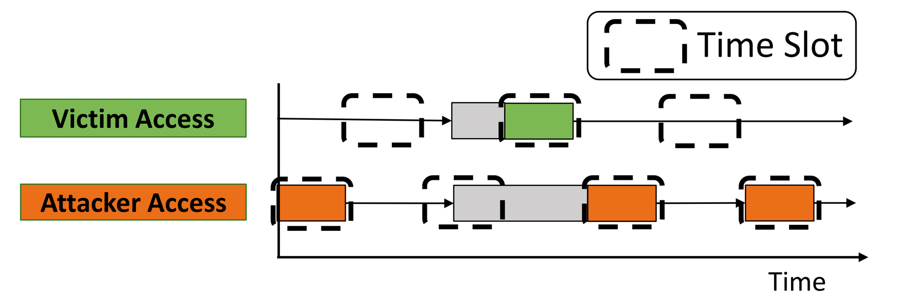
Attacker can use its own memory delays to reveal a boolean secret



3. PREVIOUS SOLUTIONS

Static Partitioning (e.g. Fixed Service):

Time slots are divided amongst CPUs/security domains in a round robin, no skip fashion

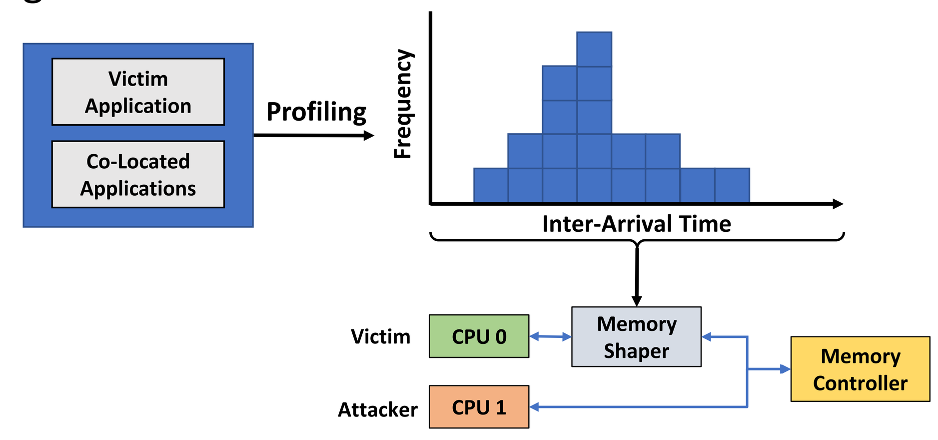


✓ **Secure**
Static partitioning, no leakage

✗ **Bad Performance**
Poor bandwidth utilization!

Traffic Shaping (e.g. Camouflage):

Shape memory requests to a secret-independent timing distribution



✗ **Insecure**
Ordering or bank information can reveal the secret

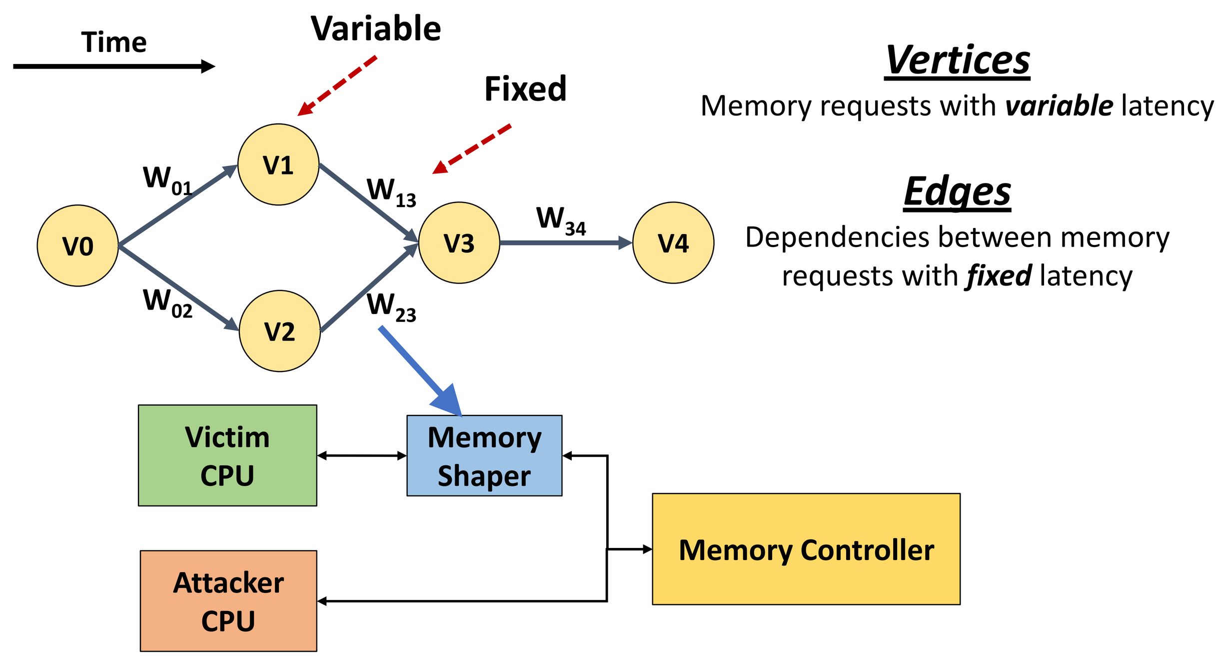
✓ **Good Performance**
Dynamic sharing of the memory controller

✗ **Expensive Profiling**
Ideal shaping distribution depends on co-running applications

4. OUR SOLUTION: DAGguise

DAGguise:

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



Vertices
Memory requests with ***variable*** latency

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

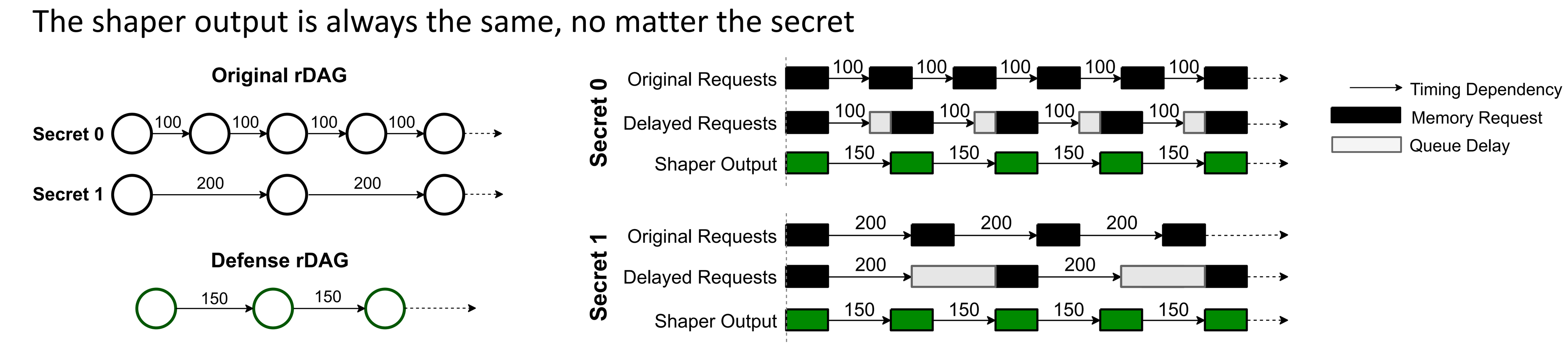
✓ **Security**
- Shaping to a secret-independent defense rDAG makes victim request patterns *indistinguishable*
- Defense rDAGs are public and are the only thing an attacker can recover

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

✓ **Profiling Cost**
- Does not require knowledge of co-located applications

5. SECURITY

Simple Shaping Example
- Different victim request patterns are shaped to the same defense rDAG
- The shaper output is always the same, no matter the secret

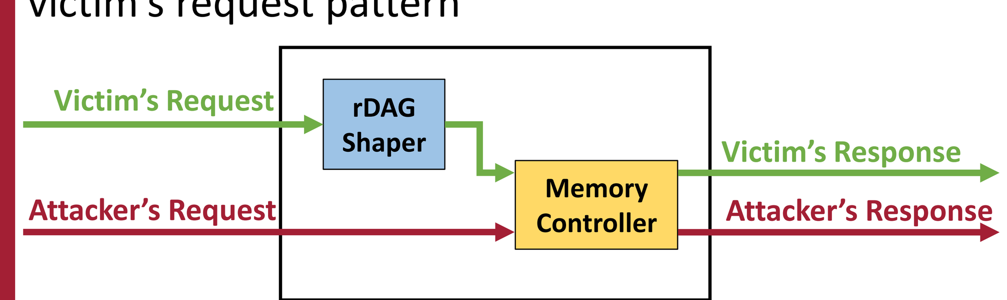


(a) Victim's Request Patterns

(b) Shaping Victim's Request Patterns to the Same rDAG

Indistinguishability Property

Attacker's response is independent from the victim's request pattern



Formalize Using State Transitions

$$\mathrm{P}(S_0, n) := \forall\ \mathrm{Req}_{Tx}, \mathrm{Req}'_{Tx},\ \forall\ \mathrm{Req}_{Rx}$$

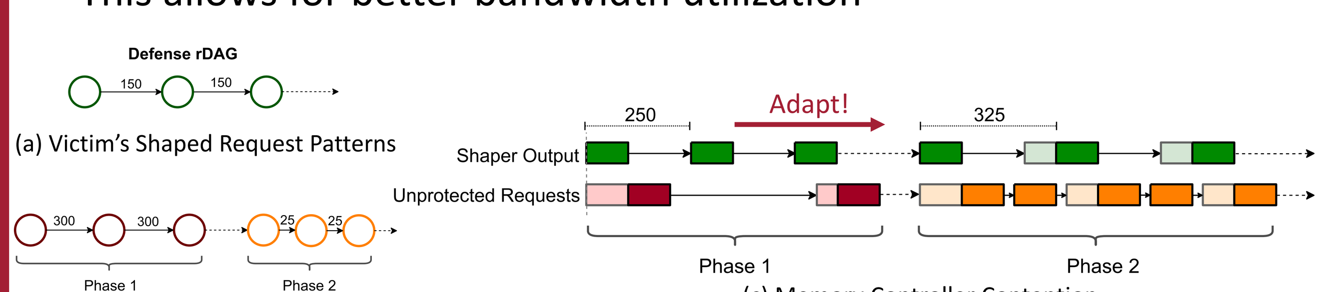
$$\textbf{if}\ S_0 \xrightarrow[\mathrm{Req}_{Tx},\mathrm{Req}_{Rx}]{\mathrm{Resp}_{Tx},\mathrm{Resp}_{Rx}} S_n\ \text{ and }\ S_0 \xrightarrow[\mathrm{Req}'_{Tx},\mathrm{Req}_{Rx}]{\mathrm{Resp}'_{Tx},\mathrm{Resp}'_{Rx}} S'_n$$

$$\textbf{then}\ \mathrm{Resp}_{Rx} = \mathrm{Resp}'_{Rx}$$

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

6. PERFORMANCE

Example: rDAG Adaptivity
- Shaper output can adapt to observed contention
- This allows for better bandwidth utilization

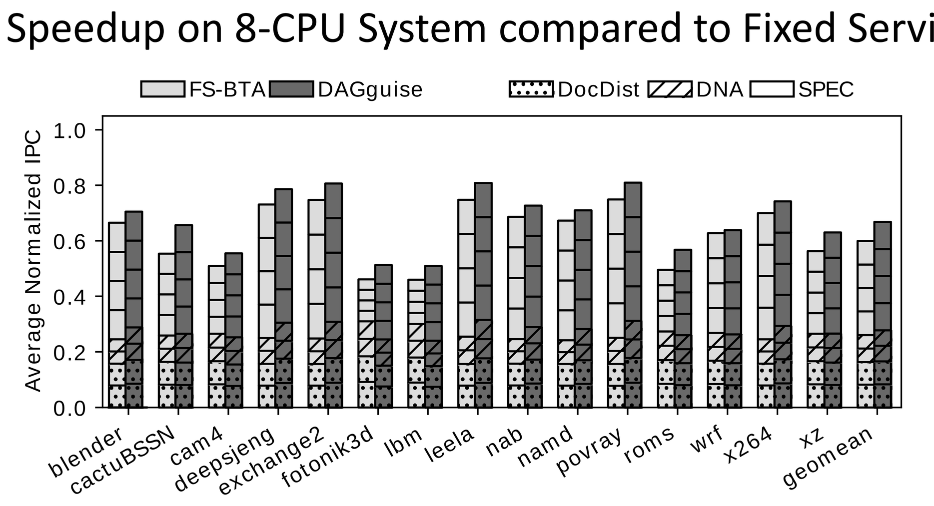


(a) Victim's Shaped Request Patterns

(b) Unprotected Program's Request Patterns

(c) Memory Controller Contention

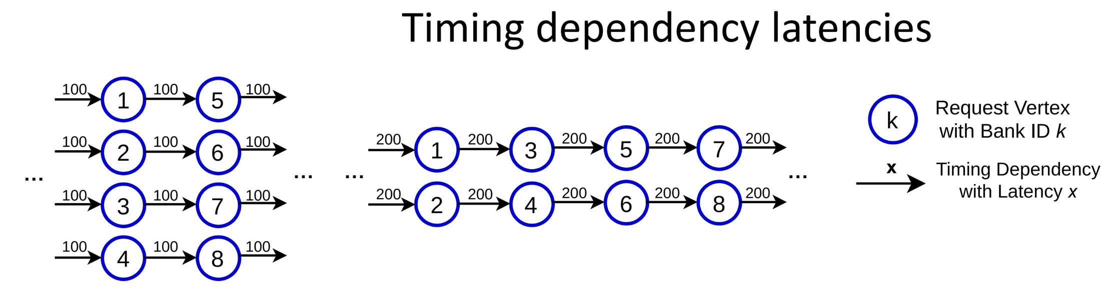
Evaluation
- Setup:
  - gem5 Out-of-Order CPU & DRAMSim2
  - 32KB L1i/d, 256kB L2, 1MB/core L3
  - Unprotected SPEC benchmark(s) co-running alongside DAGguise/Fixed Service protected application(s)
- 12% Speedup on 8-CPU System compared to Fixed Service



7. PROFILING

Goal: A defense rDAG should closely encapsulate the memory requirements of the victim

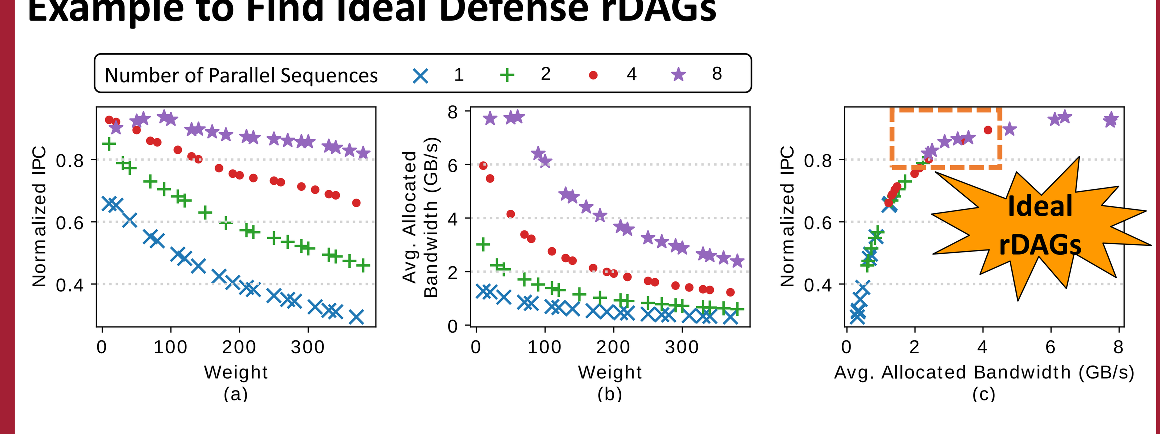
Low Profiling Cost
- Victim is profiled alone (since rDAGs can adapt to contention from co-running applications!)
- Reduce search space by finding parameters for an rDAG *template*
  - 2 Parameters: Number of parallel sequences; Timing dependency latencies



(a) 4 Parallel Sequences

(b) 2 Parallel Sequences

Example to Find Ideal Defense rDAGs

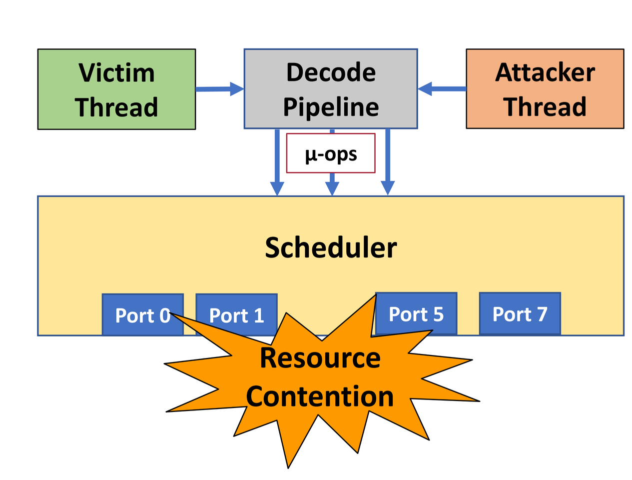


8. GENERALIZATION

Scheduler-based Side Channels:
- Requests from different security domains enter a scheduler to access shared resources
- Shape the request pattern before entering the scheduler

Examples

SMT Contention



Network on Chip Contention