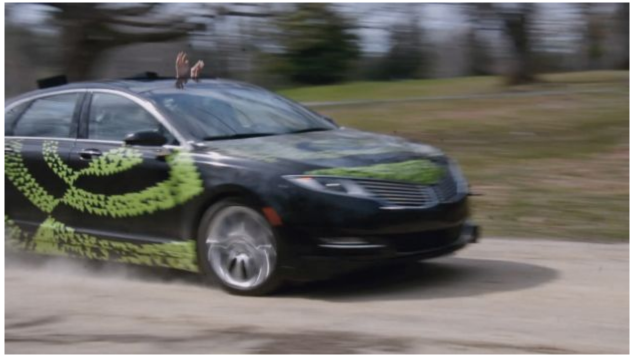
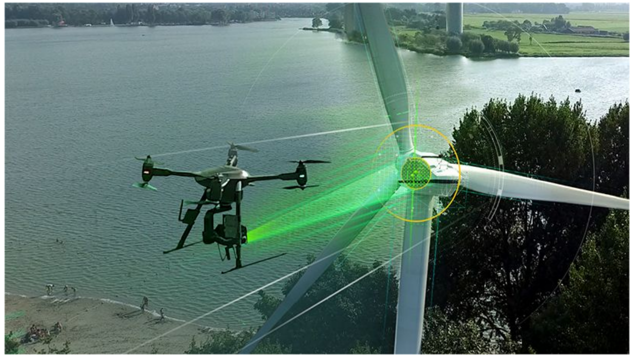
Cache Bank-Aware Denial-of-Service Attacks on Multicore ARM Processors

Michael Bechtel and Heechul Yun
University of Kansas, USA

Cyber Physical Systems (CPS)

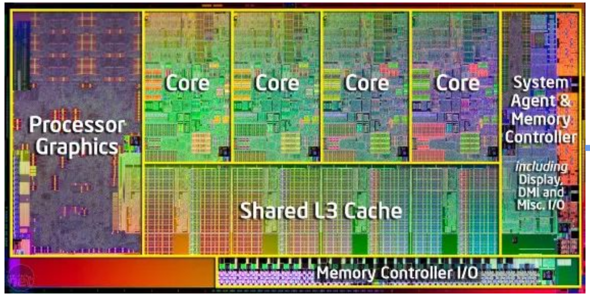
- Deployed in many different areas
  - Automotive, avionics, healthcare, etc.
- **Modern CPS require high performance platforms**

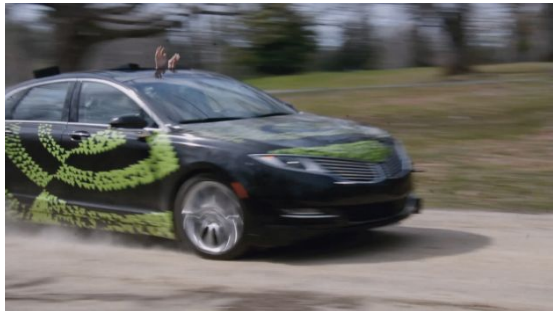

Autonomous Vehicles

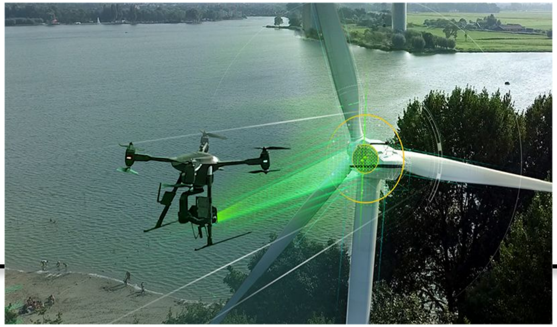

Autonomous UAVs

Multicore Platforms

- Increasingly demanded for modern CPS
  - Better performance than unicore
  - Better satisfy size, weight and power (SWaP) constraints
- **Problem: They are less predictable**

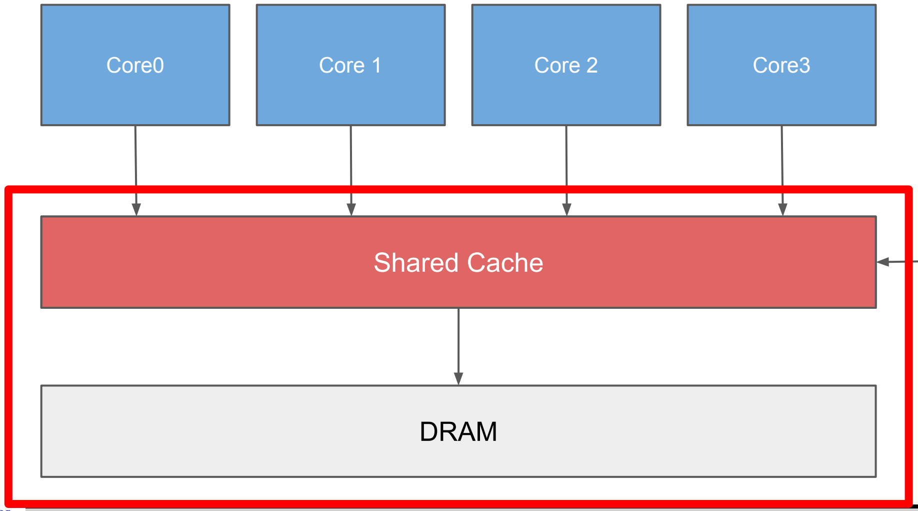






Shared Resource Contention

- Many resources are shared by all cores
- Worst case performance is unpredictable



Shared caches are important resources.

Cache Partitioning

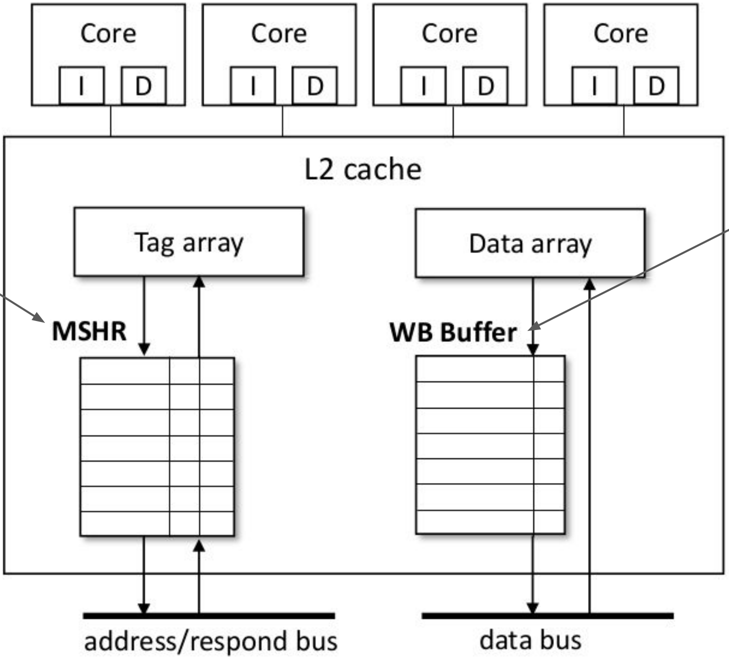
- Common technique used for isolation in shared caches
- Give each core its own dedicated slice of the cache
- Prevents unwanted cross-core LLC evictions
- **Does not prevent other forms of contention in the cache**

Contention on Shared Internal Buffers in LLC

- Modern caches use internal buffers to manage memory accesses
- **LLC partitioning can not mitigate contention on these buffers**

Miss Status Holding Registers[1]

- Track outstanding cache misses.



Writeback Buffer[2]

- Holds evicted dirty lines (writebacks).

[1] Prathap Kumar Valsan, Heechul Yun, Farzad Farshchi. Taming Non-blocking Caches to Improve Isolation in Multicore Real-Time Systems. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, 2016.
[2] Michael Bechtel and Heechul Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2019.

Memory Bandwidth Throttling

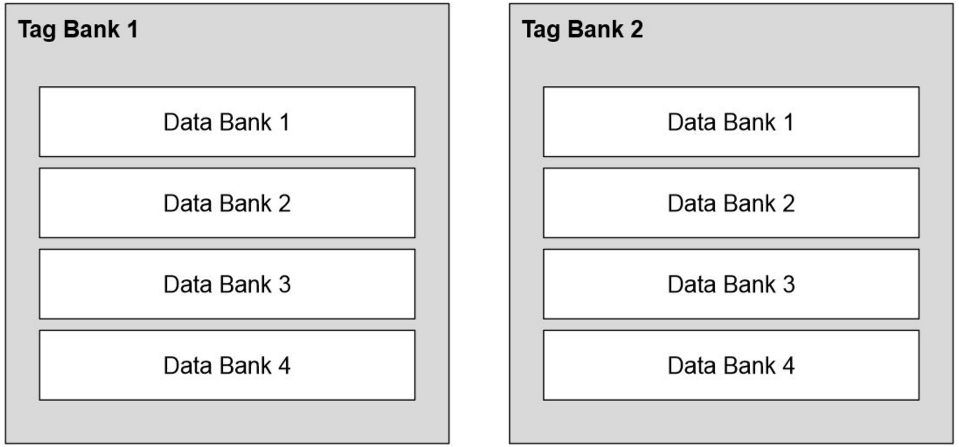
- Another common approach for mitigating contention
- Limit access of non-critical tasks to memory resources
- **Can effectively mitigate internal LLC buffer contention**[1]

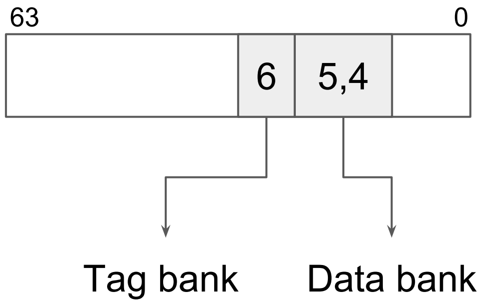
- **In this work, we show that memory bandwidth throttling is not sufficient**

[1] Michael Bechtel and Heechul Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2019.

Cache Bank Organization

- Modern caches employ bank architectures to improves MLP
- **Each bank can only service one request at a time**
- All banks can be accessed simultaneously

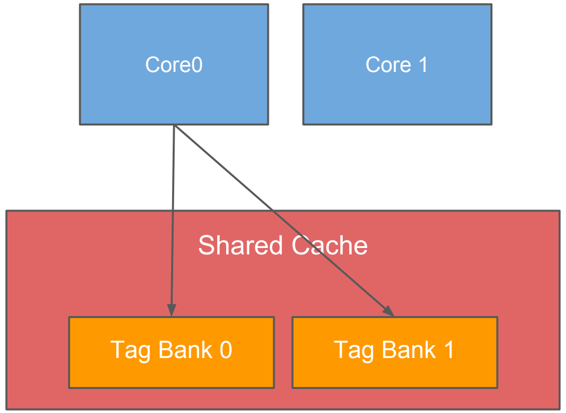




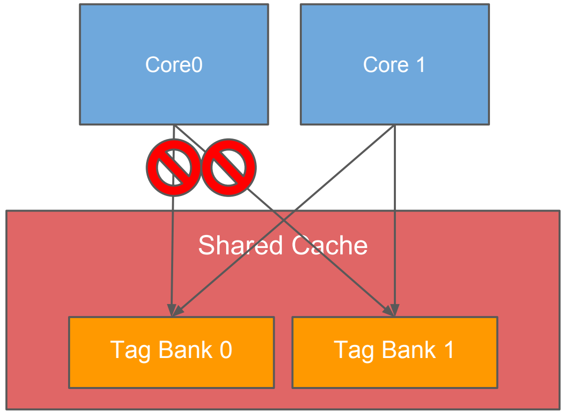
ARM Cortex A57/A72 LLC Bank Organization

Cache Bank Contention

- Cache banks also suffer from contention
- Occurs when multiple accesses are made to the same bank



Nominal Case

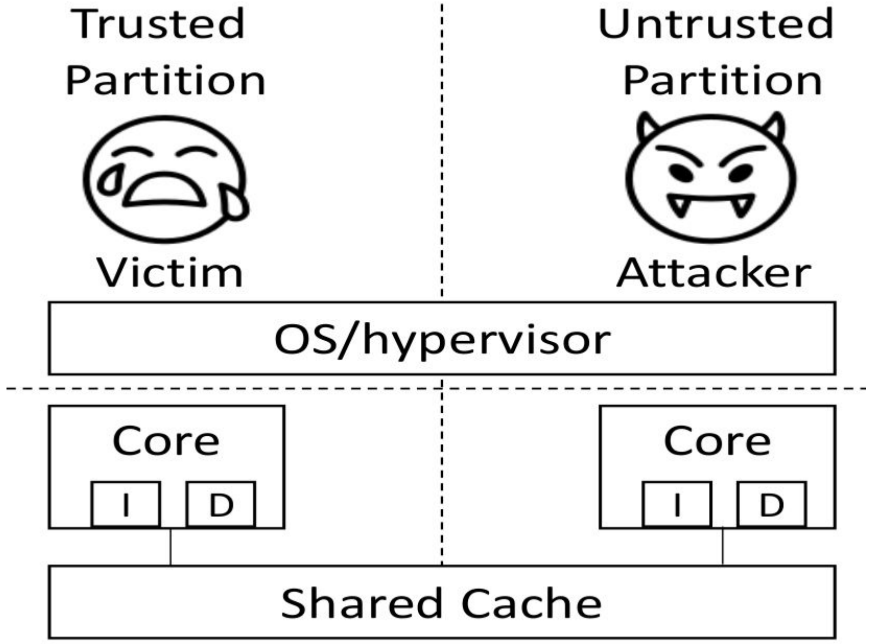


Contention Case

Outline

- Background
- **Cache Bank-Aware DoS Attacks**
- Evaluation
- Mitigation
- Conclusion

Threat Model



- Attackers can not directly affect the victim
- Attackers can not run privileged code
- **System has a shared cache**

Baseline DoS Attacks

```
for (int64_t i = 0; i<mem_size; i += LINE_SIZE)
{
    ptr[i] = 0xff;
}
```

Sequential Attacker
(BwWrite)

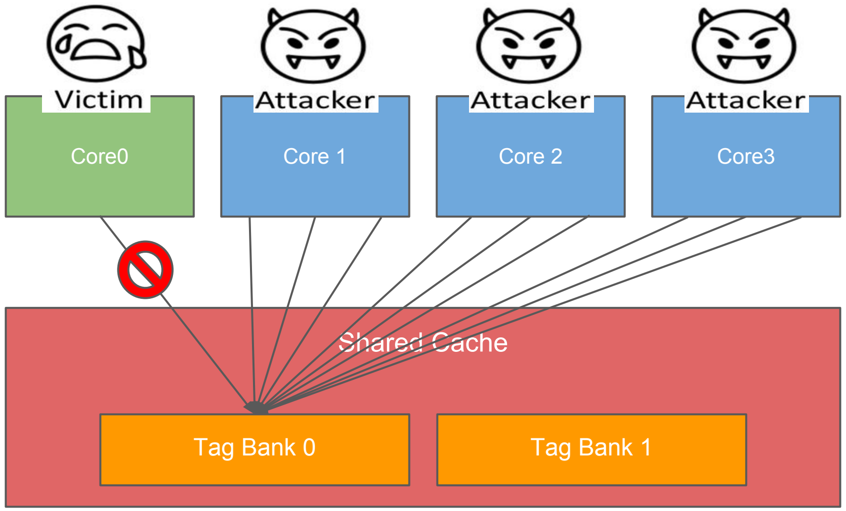
```
static int* list[MAX_MLP];
static int next[MAX_MLP];

for (int64_t i = 0; i < iter; i++) {
    switch (mlp) {
    case MAX_MLP:
    .
    .
    case 2:
        list[1][next[1]+1] = 0xff;
        next[1] = list[1][next[1]];
        /* fall-through */
    case 1:
        list[0][next[0]+1] = 0xff;
        next[0] = list[0][next[0]];
    }
}
```

Random Attacker
(PLLWrite)

- Both perform continuous write accesses
- Can be configured to access LLC or DRAM
    - E.g. BwWrite(LLC), BwWrite(DRAM), etc.

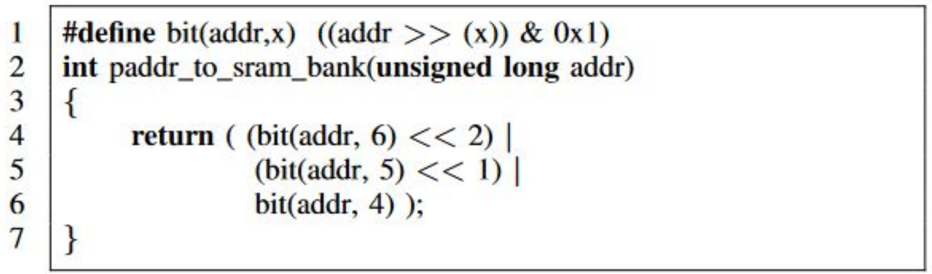
Cache Bank-Aware DoS Attack

- **Target and generate contention in a single cache bank**
  - Slows down accesses made by victim tasks to that bank
- All accesses are LLC hits ⇒ **Can still impact cross-core victim**



Cache Bank-Aware DoS Attack

- Allocate a large memory chunk
- Iterate over the memory addresses at a cache line granularity
  - **Only keep addresses that map to a target LLC data bank**



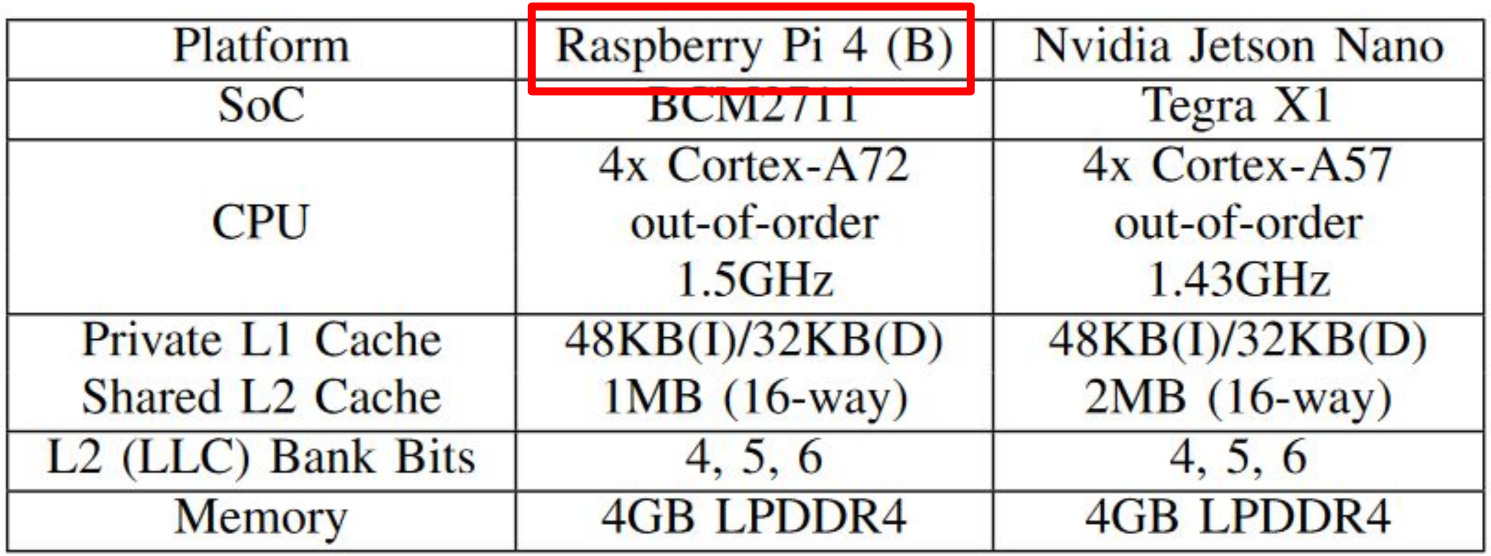
```
#define bit(addr,x)  ((addr >> (x)) & 0x1)
int paddr_to_sram_bank(unsigned long addr)
{
        return ( (bit(addr, 6) << 2) |
                 (bit(addr, 5) << 1) |
                 bit(addr, 4) );
}
```

- Split and access addresses across multiple parallel linked lists
- We refer to this attack as **BkPLLWrite(LLC)**

Outline

- Background
- Cache Bank-Aware DoS Attacks
- **Evaluation**
- Mitigation
- Conclusion

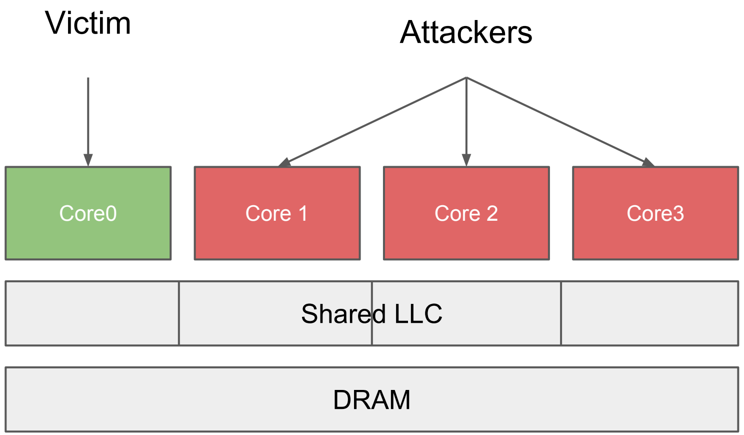
Tested Platforms



Platform	Raspberry Pi 4 (B)	Nvidia Jetson Nano
SoC	BCM2711	Tegra X1
CPU	4x Cortex-A72 out-of-order 1.5GHz	4x Cortex-A57 out-of-order 1.43GHz
Private L1 Cache Shared L2 Cache	48KB(I)/32KB(D) 1MB (16-way)	48KB(I)/32KB(D) 2MB (16-way)
L2 (LLC) Bank Bits	4, 5, 6	4, 5, 6
Memory	4GB LPDDR4	4GB LPDDR4

- Both platforms have the same LLC bank architectures
  - 2 tag banks X 4 data banks ⇒ 8 total banks

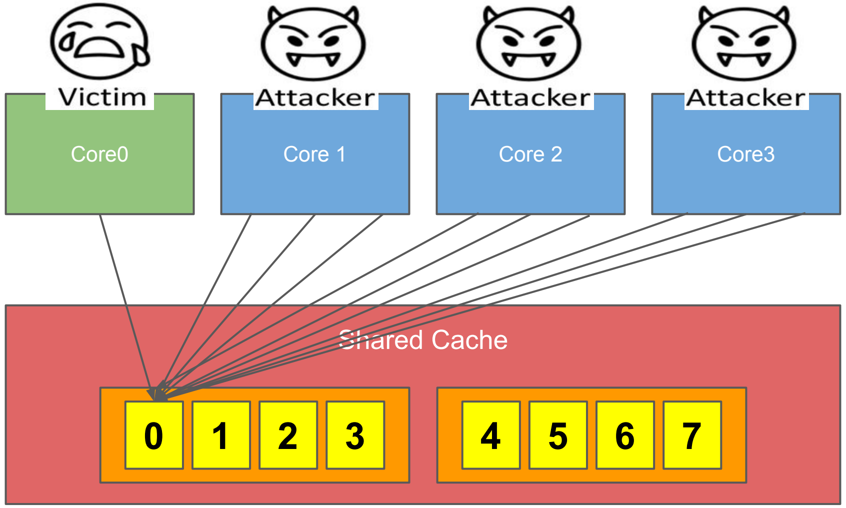
Methodology



- Run victim alone and with DoS attackers
  - Measure victim slowdown and LLC miss rates
- LLC set partitioning (PALLOC)
  - Split LLC into 4 partitions, **2/2** between victim and attackers

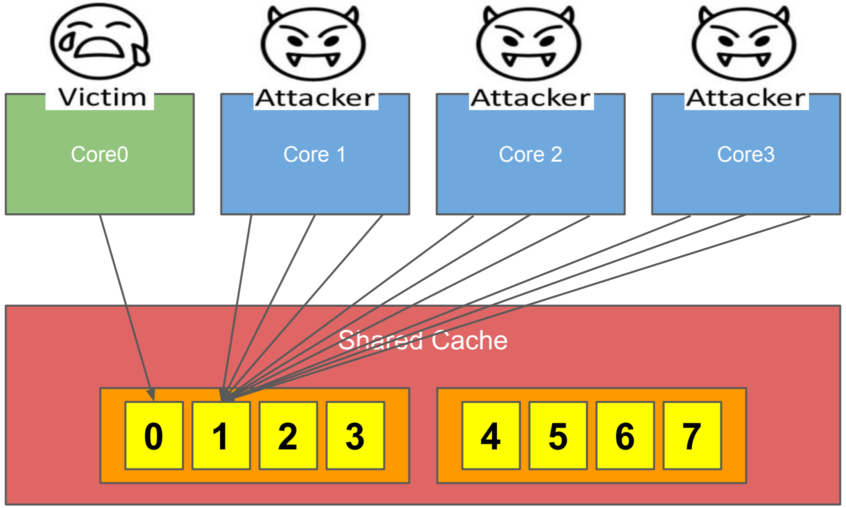
Cache Bank Partitioning Experiment

- Simulate LLC bank partitioning
- Victim - BkPLLRead(LLC) ⇒ Data bank 0
- Vary data bank accessed by attackers



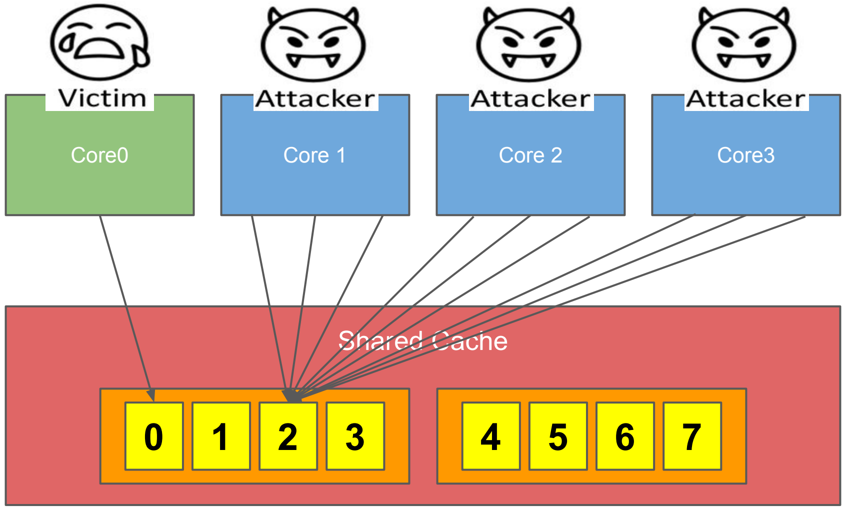
Cache Bank Partitioning Experiment

- Simulate LLC bank partitioning
- Victim - BkPLLRead(LLC) ⇒ Data bank 0
- Vary data bank accessed by co-runners



Cache Bank Partitioning Experiment

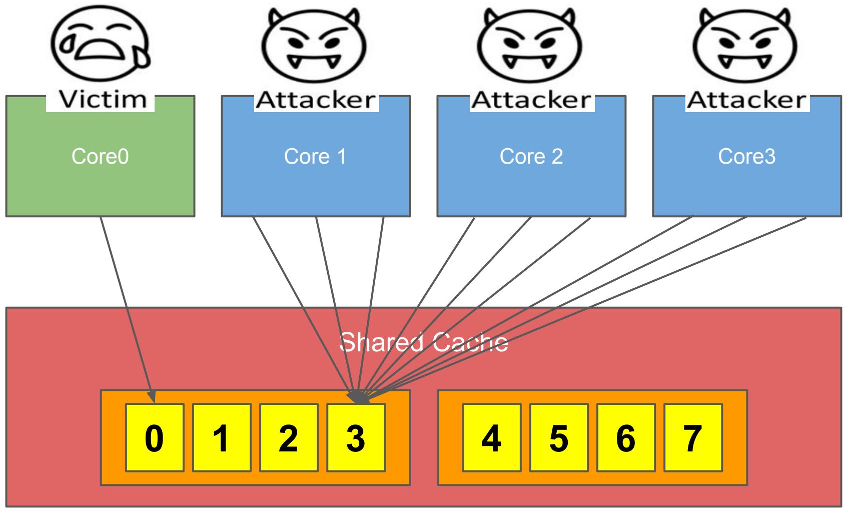
- Simulate LLC bank partitioning
- Victim - BkPLLRead(LLC) ⇒ Data bank 0
- Vary data bank accessed by co-runners



THE UNIVERSITY OF KANSAS

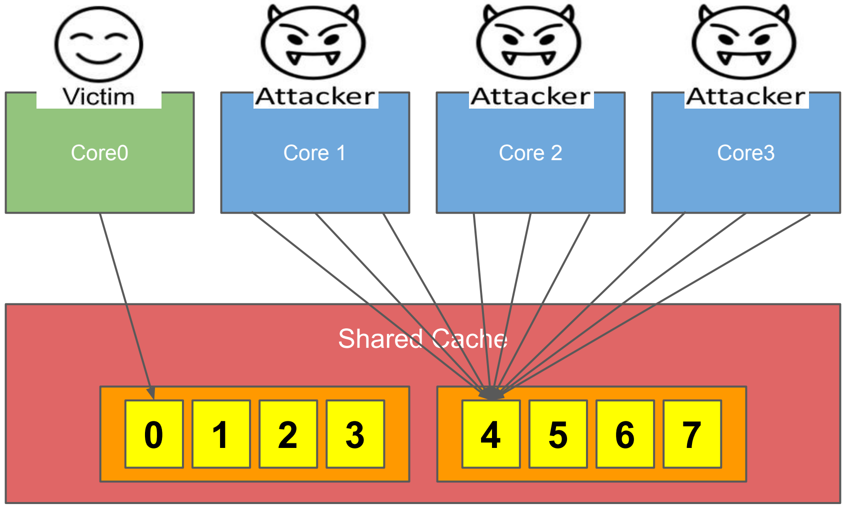
Cache Bank Partitioning Experiment

- Simulate LLC bank partitioning
- Victim - BkPLLRead(LLC) ⇒ Data bank 0
- Vary data bank accessed by co-runners



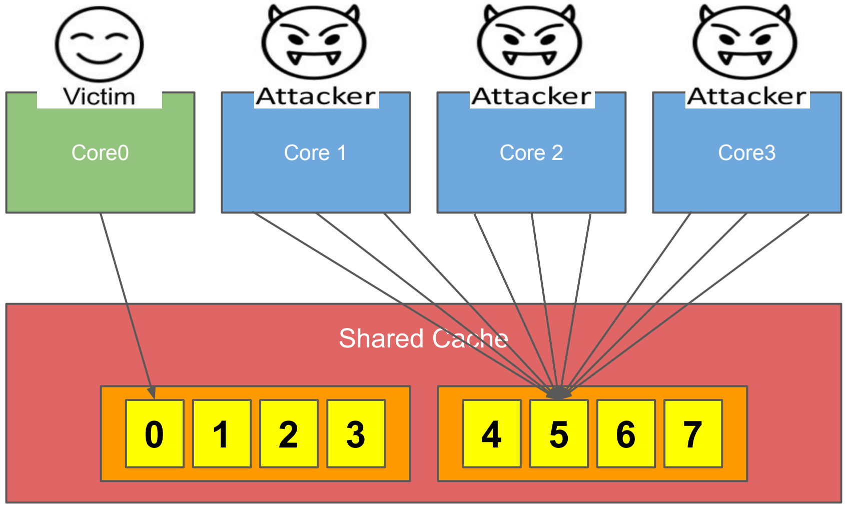
Cache Bank Partitioning Experiment

- Simulate LLC bank partitioning
- Victim - BkPLLRead(LLC) ⇒ Data bank 0
- Vary data bank accessed by co-runners



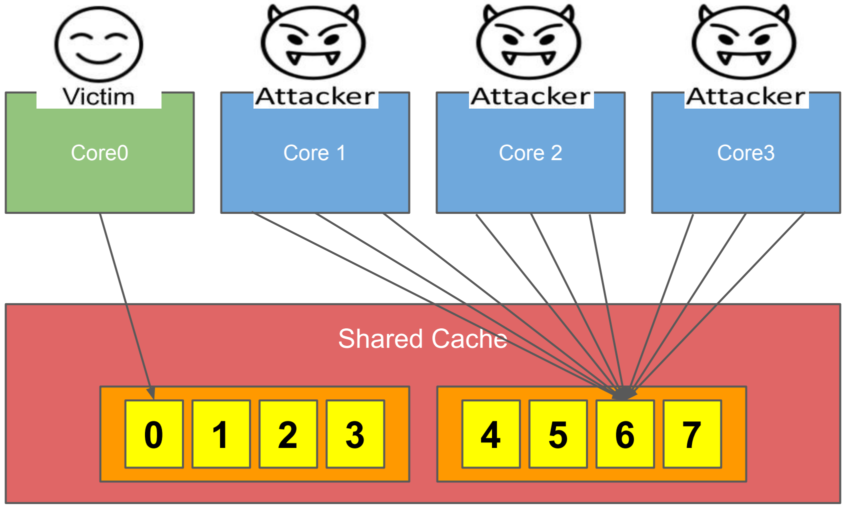
Cache Bank Partitioning Experiment

- Simulate LLC bank partitioning
- Victim - BkPLLRead(LLC) ⇒ Data bank 0
- Vary data bank accessed by co-runners



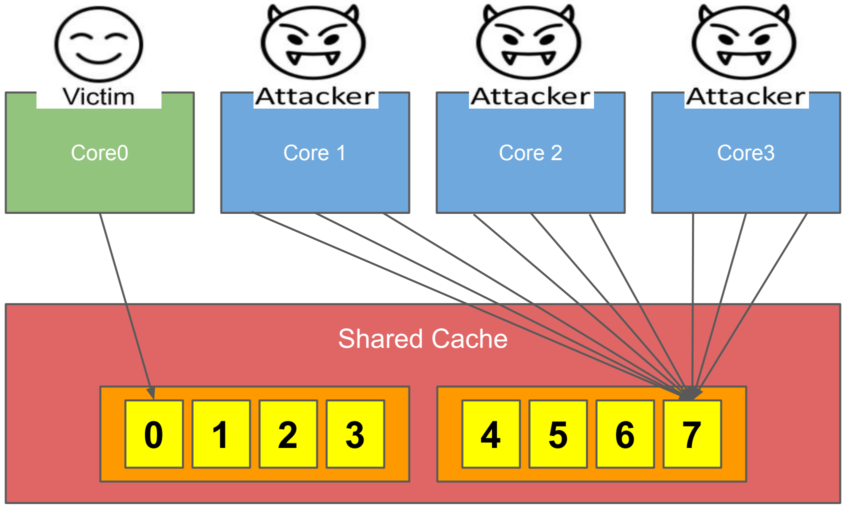
Cache Bank Partitioning Experiment

- Simulate LLC bank partitioning
- Victim - BkPLLRead(LLC) ⇒ Data bank 0
- Vary data bank accessed by co-runners

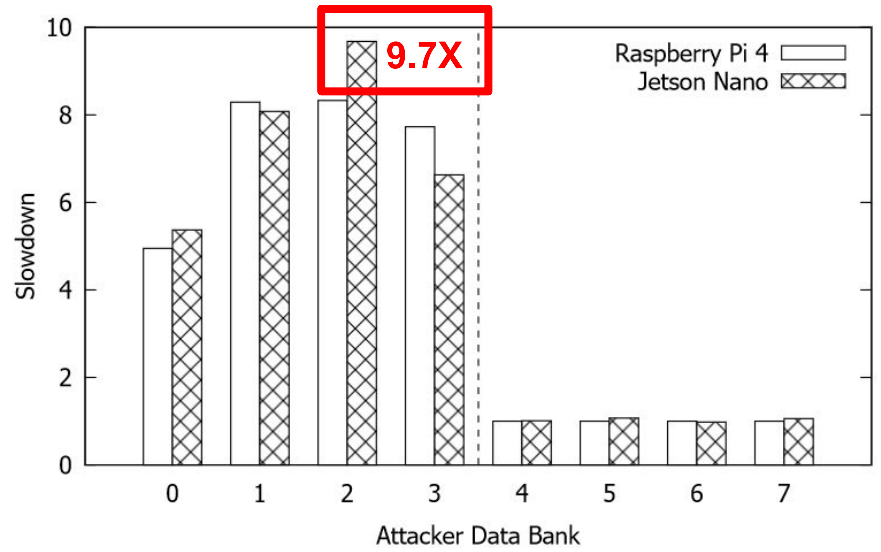


Cache Bank Partitioning Experiment

- Simulate LLC bank partitioning
- Victim - BkPLLRead(LLC) ⇒ Data bank 0
- Vary data bank accessed by co-runners



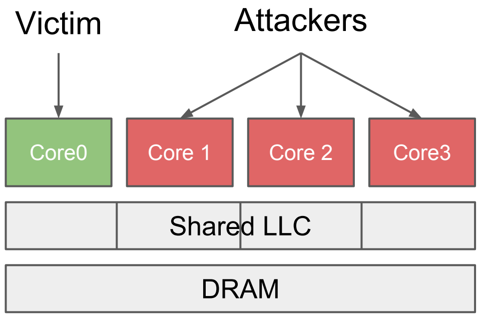
LLC Bank Partitioning Results



- Up to **9.7X** slowdown when same tag bank is accessed
- No impact when victim and attackers access different tag banks

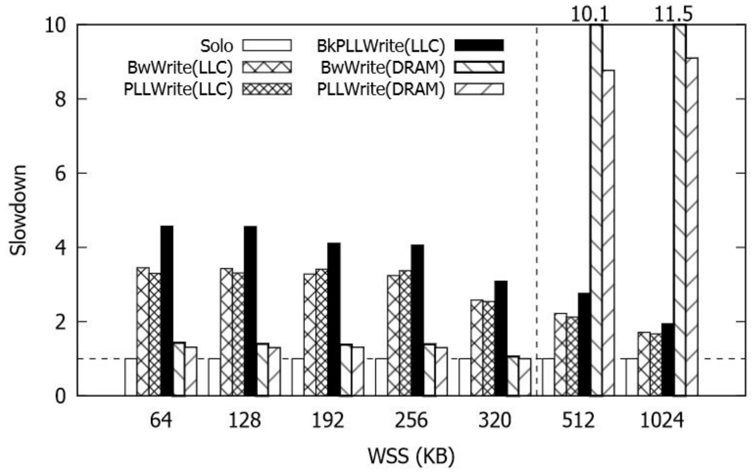
Synthetic Workload Experiment

- Victim task ⇒ BwRead
  - Read intensive benchmark that is cache bank-oblivious
  - Vary the working set size (WSS) to be LLC and DRAM-fitting
- Attacker tasks
  - DoS attacks on DRAM ⇒ BwWrite(DRAM), PLLWrite(DRAM)
  - Bank-oblivious cache attacks ⇒ BwWrite(LLC), PLLWrite(LLC)
  - Cache Bank-Aware attack ⇒ BkPLLWrite(LLC)

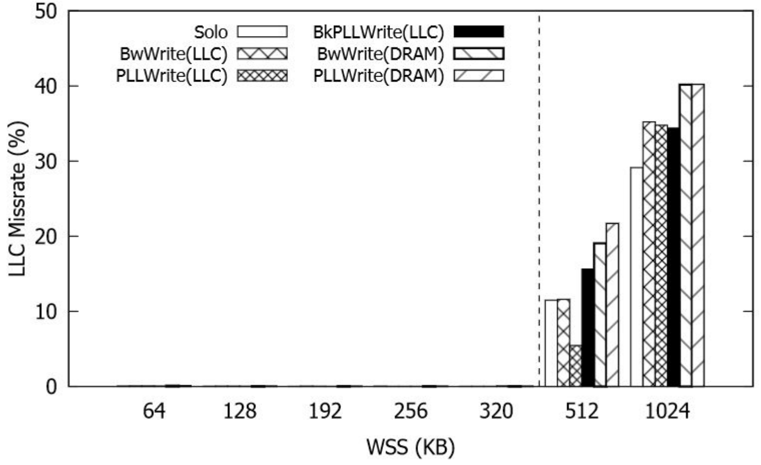


Impact to Synthetic Workloads

Slowdown



LLC Missrate



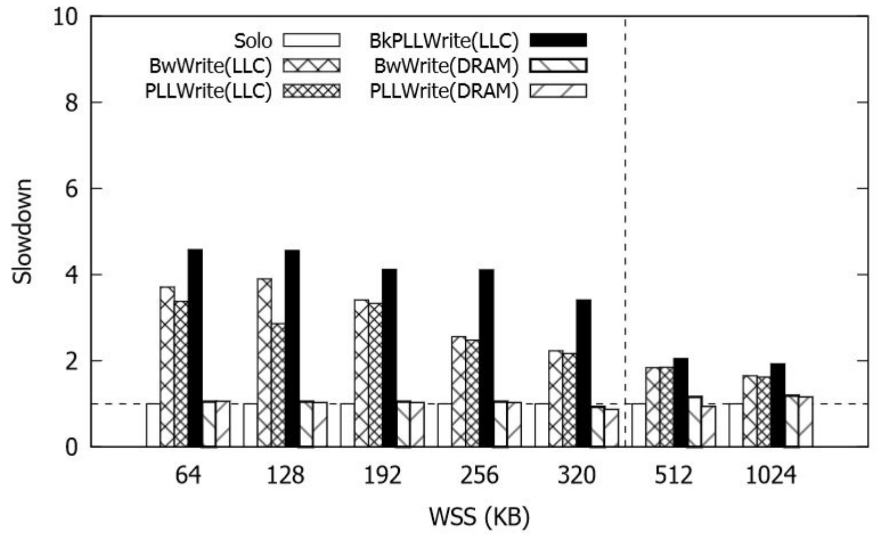
- **Bank-aware attacks are superior to bank-oblivious attacks**
  - When victim's working set size is LLC-fitting
  - No slowdown is due to LLC evictions

Memory Bandwidth Throttling Experiment

- Run the synthetic victim tests with memory bandwidth throttling enabled
- We use MemGuard, a per-core bandwidth regulator
  - Sets a bandwidth budget for each core over a period (e.g. 1 ms)
  - Throttles cores that exceed their budget until the next period
- Victim gets full bandwidth access (i.e. no throttling)
- Attackers are assigned a budget of 100 MB/s

- **Does MemGuard protect against cache DoS attacks?**

Impact of Memory Bandwidth Throttling

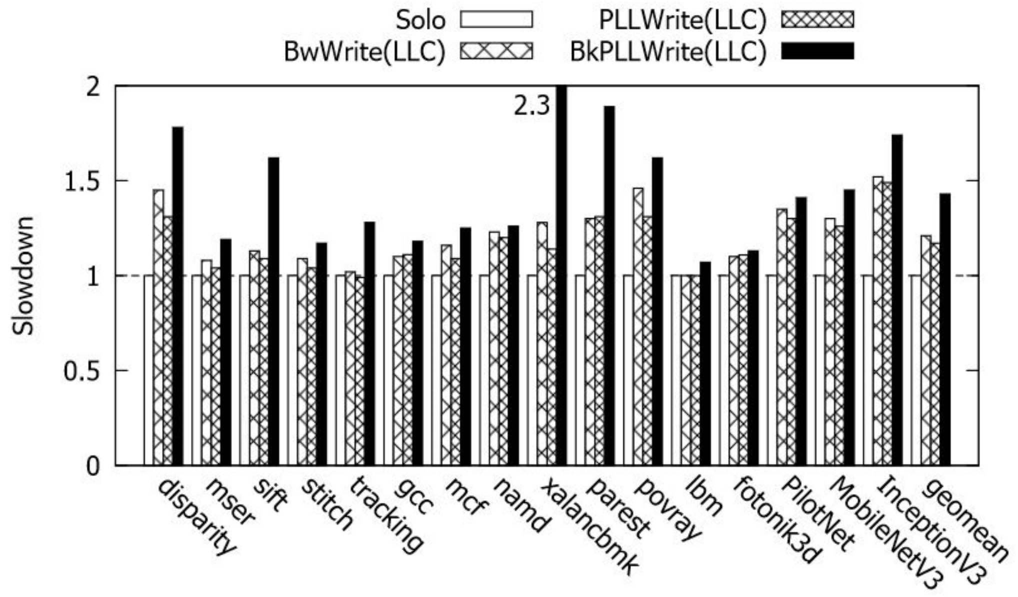


- Throttling does protect against DRAM-fitting attacks
- **None of the cache DoS attacks are affected**
    - Especially the Cache Bank-Aware attacks

Real-World Workload Experiment

- We run the same DoS attacks against real-world benchmarks:
  - Five benchmarks from the **SD-VBS** suite (input size = fullhd)
  - Eight benchmarks from the **SPEC2017** suite (input size = ref)
  - Three representative DNN models: PilotNet, MobileNetV3 and InceptionV3
- Both cache partitioning and memory bandwidth throttling are enabled
- We only run cache DoS attacks in these tests
  - DRAM bandwidth contention can be mitigated with MemGuard (bandwidth throttling)

Impact to Real-World Benchmarks



- Cache DoS attacks are still effective
- **Cache Bank-Aware attacks again have the most impact**
  - More than 2X on average over bank-oblivious attacks

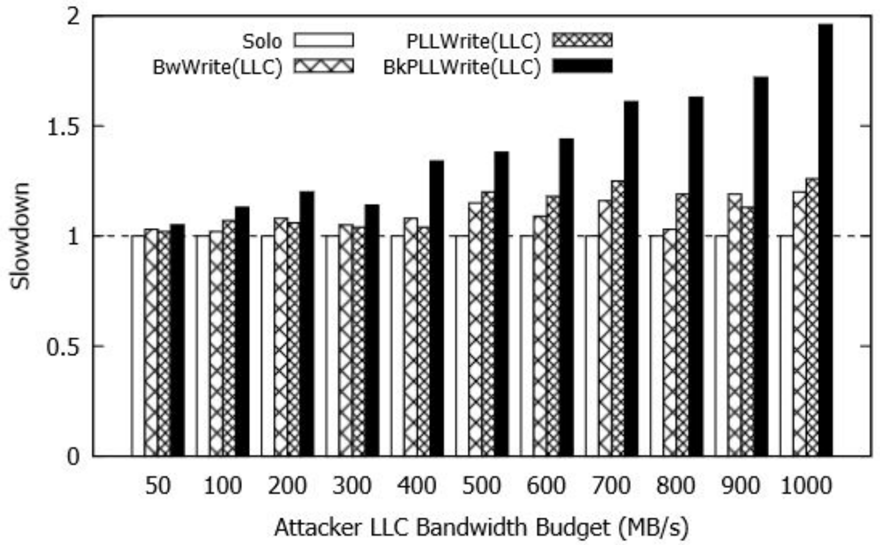
Outline

- Background
- Cache Bank-Aware DoS Attacks
- Evaluation
- **Mitigation**
- Conclusion

LLC Bandwidth Throttling

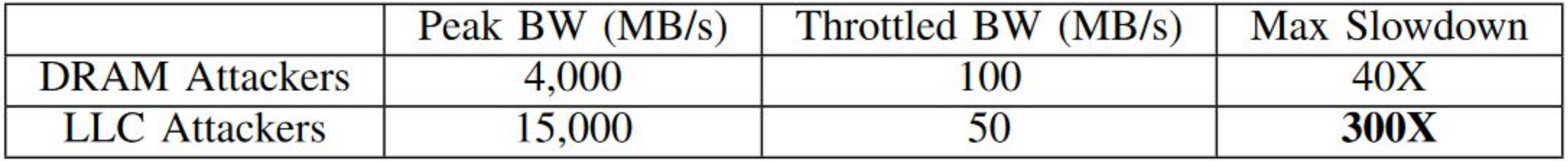
- Idea: Throttle LLC bandwidth instead of memory bandwidth
  - To mitigate cache bank contention
- By default, MemGuard tracks per-core LLC misses
  - Calculates memory bandwidth consumed by each core
- We modify MemGuard to instead track per-core L1 misses
  - Regulates LLC traffic instead of DRAM traffic
- Victim still has full LLC bandwidth access
- Attackers' LLC budget is varied from 50MB/s to 1GB/s

Impact of LLC Bandwidth Throttling



- LLC bandwidth throttling does protect against cache DoS attacks
  - When attack budgets are set low enough (e.g. 50MB/s)

LLC Bandwidth Throttling Tradeoff



	Peak BW (MB/s)	Throttled BW (MB/s)	Max Slowdown
DRAM Attackers	4,000	100	40X
LLC Attackers	15,000	50	**300X**

- LLC bandwidth throttling can provide isolation
- **But at a notable cost to system performance**
  - To achieve <10% slowdown, best-effort LLC bandwidth is throttled by **~300X**
- This tradeoff is not desirable for general system performance

Hardware-Based Solutions

- LLC bank partitioning
  - Could achieve isolation in a simulated test case
  - **Would require hardware modifications**
  - May require smaller cache space and bandwidth
- Complex bank address mappings
  - Difficult to reverse engineer
  - **Rendered ineffective if mapping is discovered**
- LLC bandwidth throttling
  - Software-based approach works, but at a cost
  - Hardware-based approaches may not have to pay such costs
  - **Our future work**

Outline

- Background
- Cache Bank-Aware DoS Attacks
- Evaluation
- Mitigation
- **Conclusion**

Conclusion

- We identify that cache bank contention as an important unaddressed problem
  - Can be exploited to delay cross-core victim tasks
- We develop a Cache Bank-Aware DoS attack
  - Highly effective at delaying cross-core victim tasks
- We show that our attack can bypass existing defense techniques
  - Both cache partitioning and memory bandwidth throttling
- We explore new mitigation mechanisms to address our attack
  - LLC bandwidth throttling is possible, but at a cost
  - Other mitigations require hardware support

Thank you!

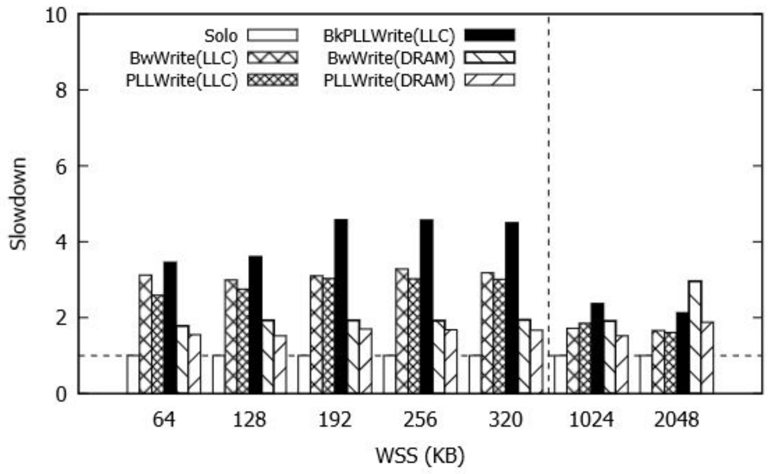
Disclaimer:

This research is supported by NSF grant CNS-1815959, CPS-2038923 and NSA Science of Security initiative contract no. H98230-18-D-000.

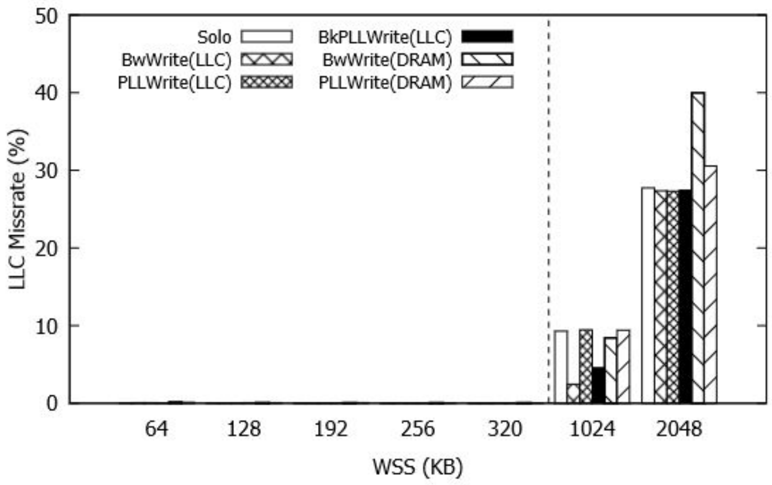
APPENDIX

Comparison to Existing DoS Attacks - Nano

Slowdown



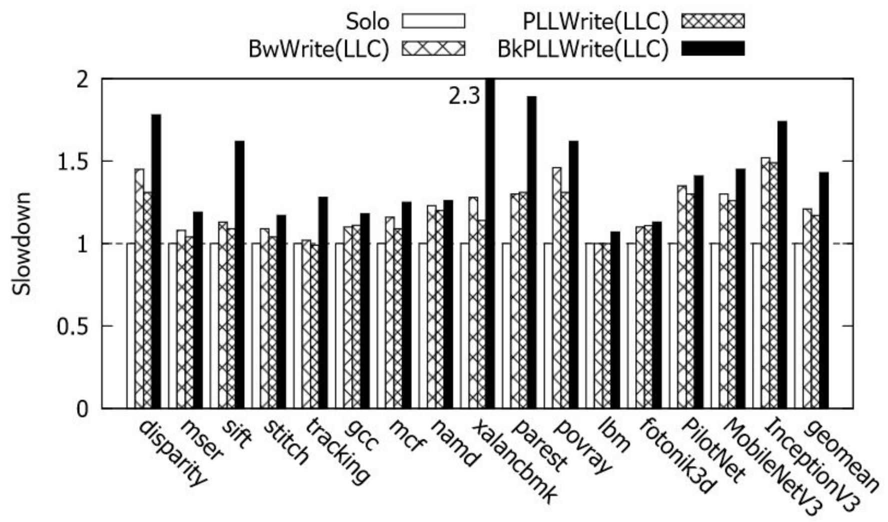
LLC Missrate



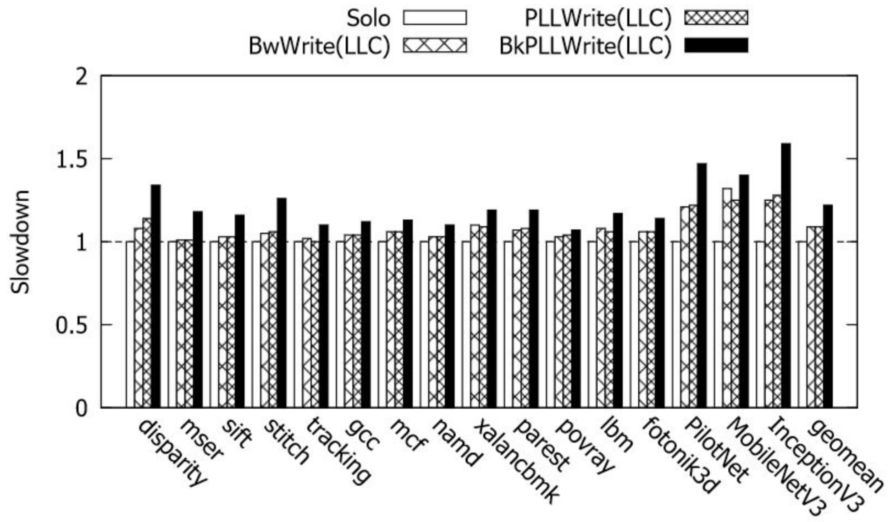
- **Like the Pi 4, Bank-Aware attacks have the most impact.**
- DRAM-fitting attacks are less effective.
  - Likely due to differences in the memory controller.

Impact to Real-World Benchmarks

Pi 4

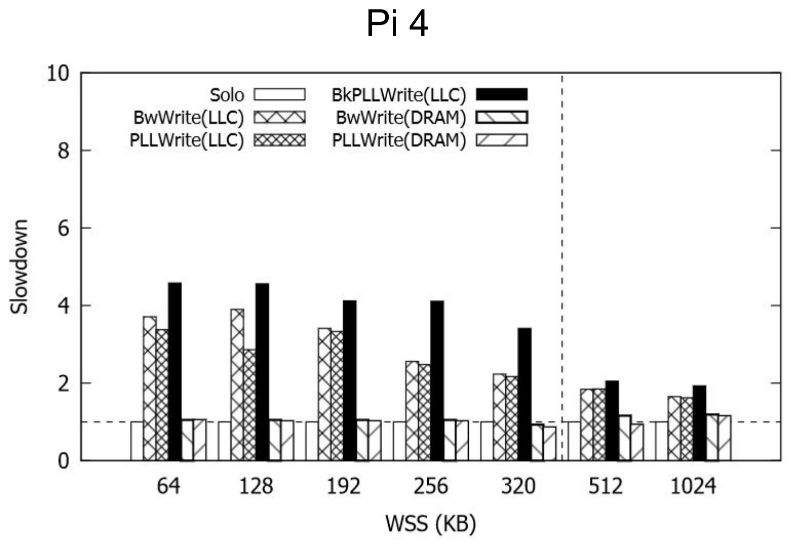


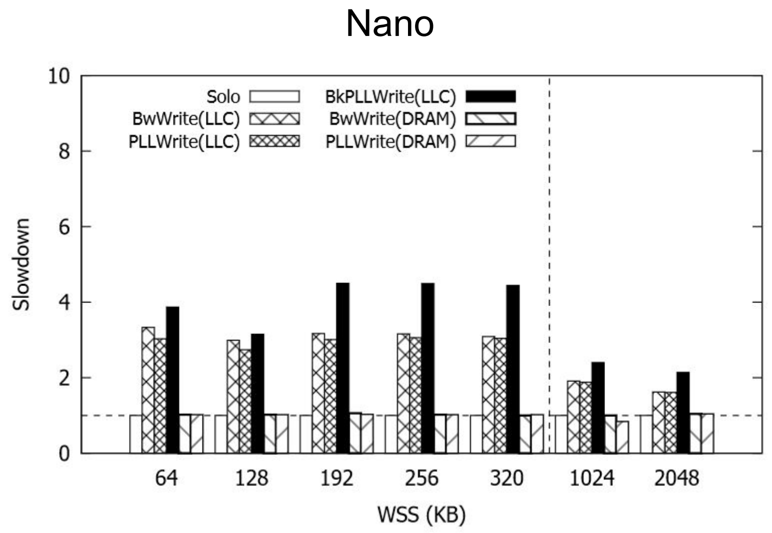
Nano



- LLC-fitting DoS attacks are still effective
- **Cache Bank-Aware attacks again have the most impact**
  - More than 2X on average over bank-oblivious attacks

Impact of DRAM Bandwidth Throttling

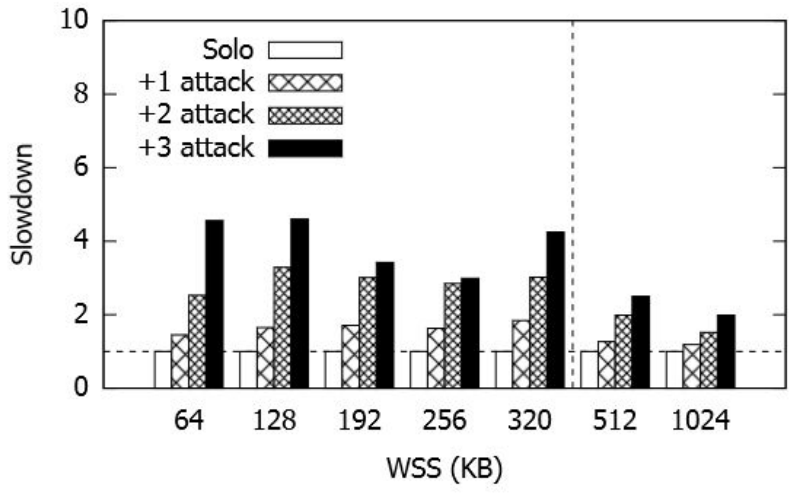




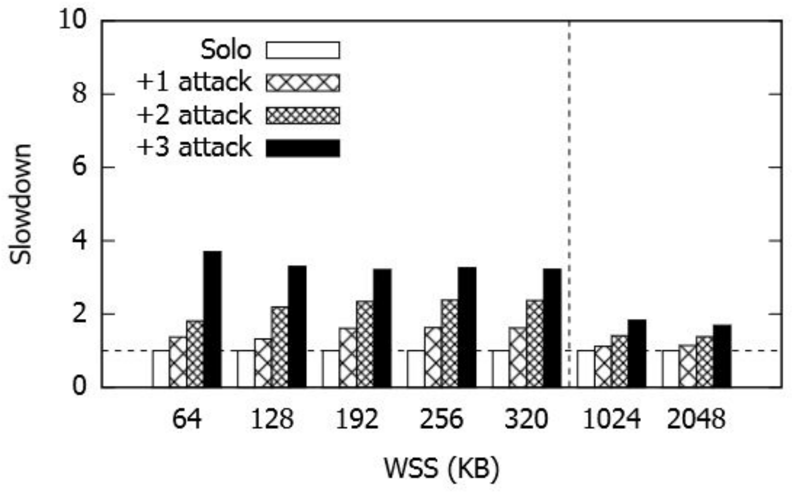
- Throttling does protect against DRAM-fitting attacks
- **None of the LLC-fitting DoS attacks are affected**
    - Especially the Bank-Aware DoS attacks

Impact of Varying # of Attackers

Pi 4



Nano



- As expected, more attackers have greater impact
- **Notable slowdown still happens with few attackers**