MI6: Secure enclaves in a speculative out-of-order processor

Arvind
Computer Science and Artificial Intellegence Lab
MIT

Barcelona Supercomputer Center
October 15, 2019

Security attacks

- Most security attacks exploit buggy systems software or poorly written applications codes
- Some security attacks result from poor or incomplete specifications
- Very few security attacks are a consequence of buggy hardware
- Some very targeted security attacks exploit processor *side channels*
  - Caches, branch predictors, ...

Spectre attack
[2018]

- Hardware is implemented correctly
- Software is implemented correctly
- And yet it has been shown that it is possible for one process to steal secrets from another using side-channels in out-of-order processors without passing any values explicitly!
- Many similar attacks are possible

It is impossible to write secure software under such conditions!

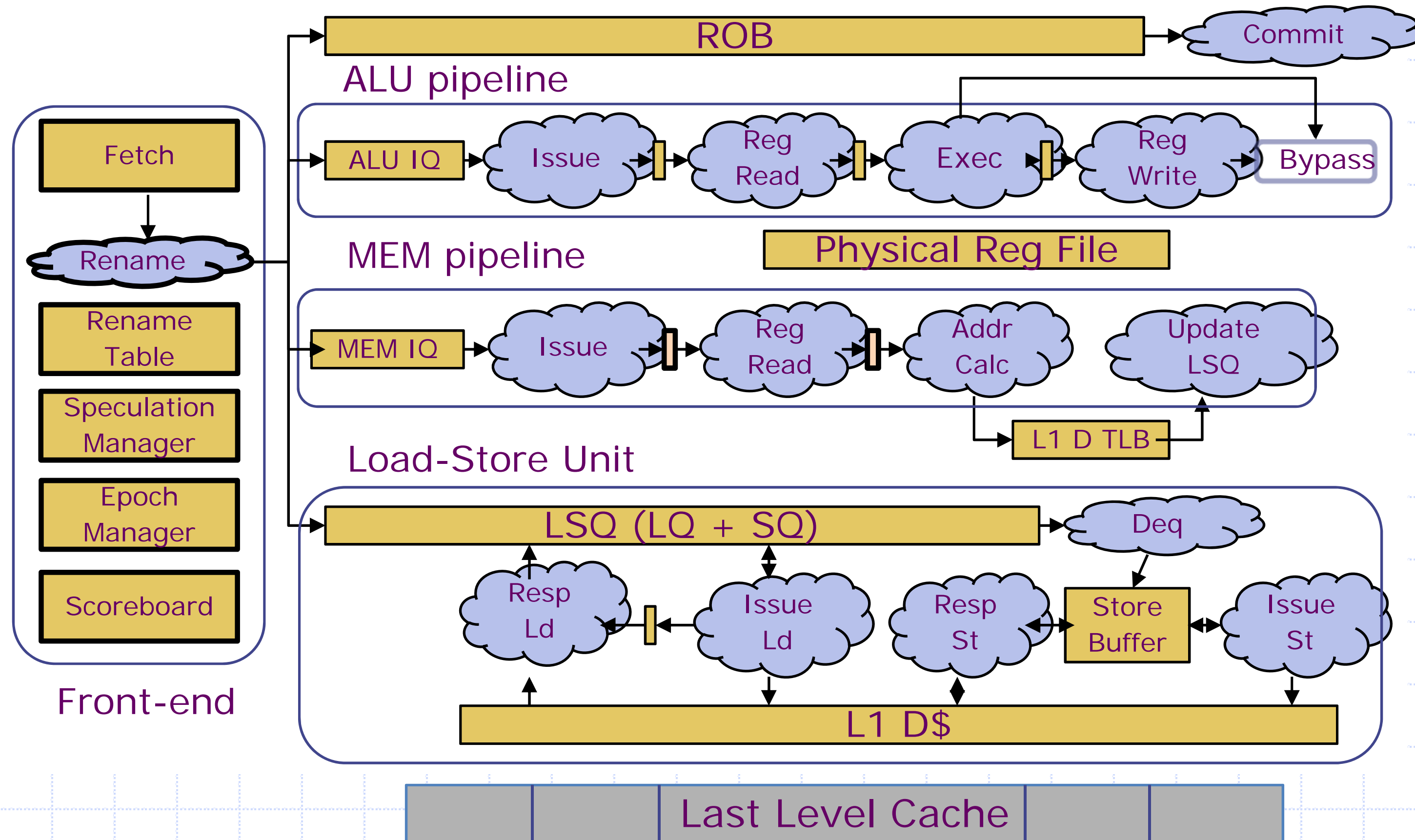
Hardware fix: Impose strong isolation between processes, i.e., secure enclaves

Intel XGS,

Sanctum[Costan, Lebedev, Devadas]

RiscyOO: A RISC V OOO Processor

S. Zhang, A. Wright, T. Bourgeat, Arvind [MICRO2018]

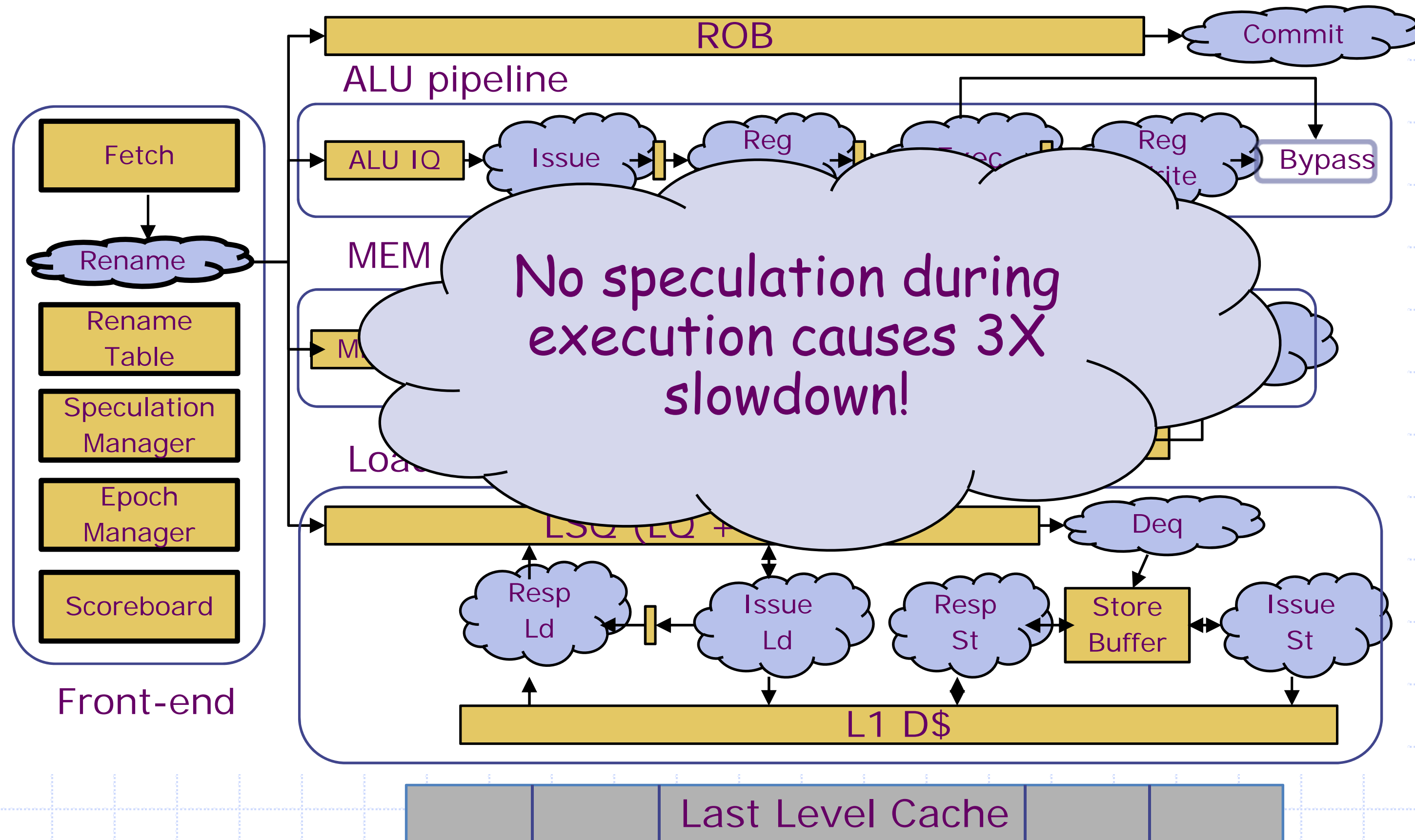


A sophisticated OOO processor designed for experimentation;

Boots linux, runs on FPGAs, synthesizes at 1GHz in 22nm

RiscyOO: A RISC V OOO Processor

S. Zhang, A. Wright, T. Bourgeat, Arvind [MICRO2018]



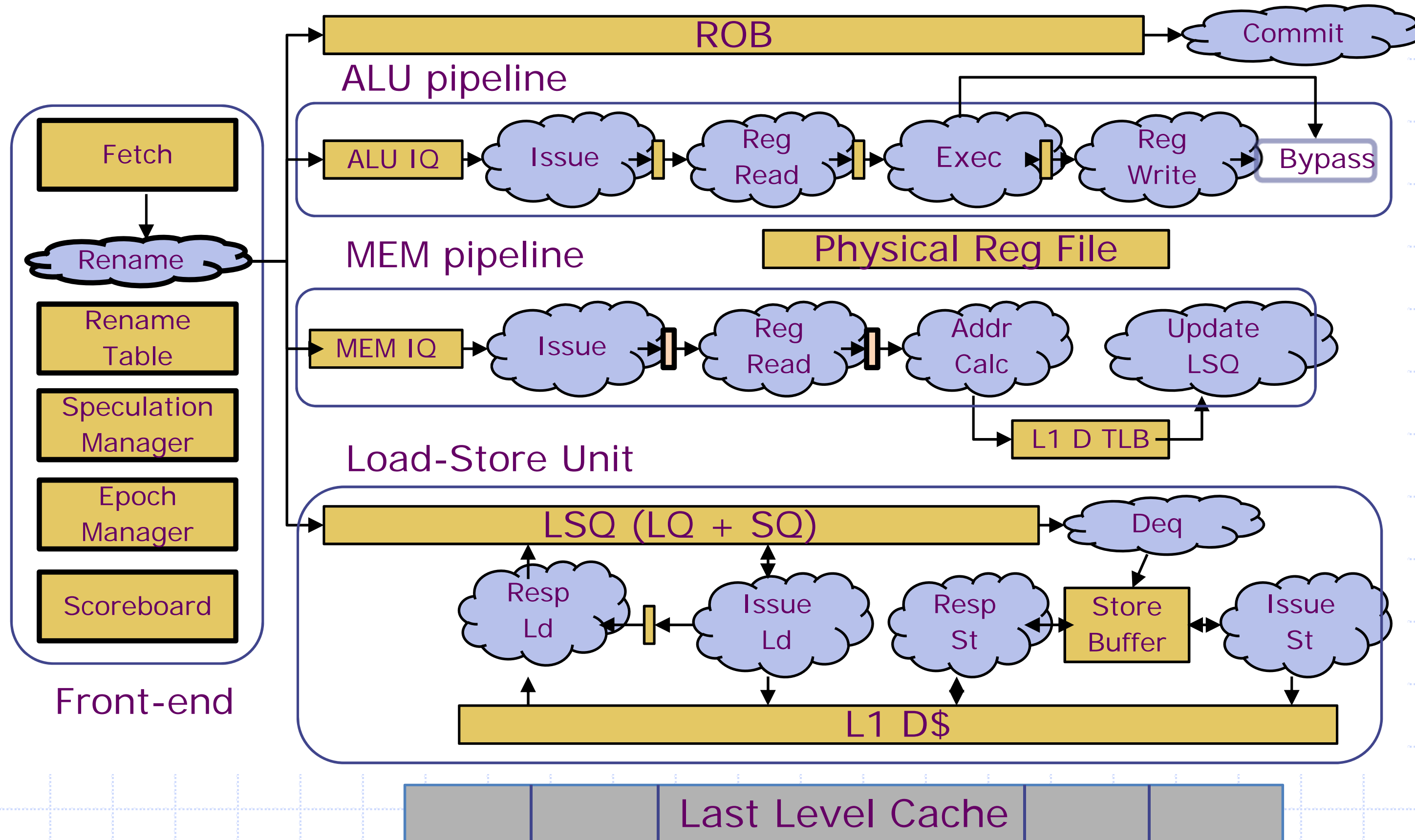
A sophisticated OOO processor designed for experimentation;

Boots linux, runs on FPGAs, synthesizes at 1GHz in 22nm

This is like throwing the baby out with the bath water!

MI6: Secure Enclaves in RiscyOO

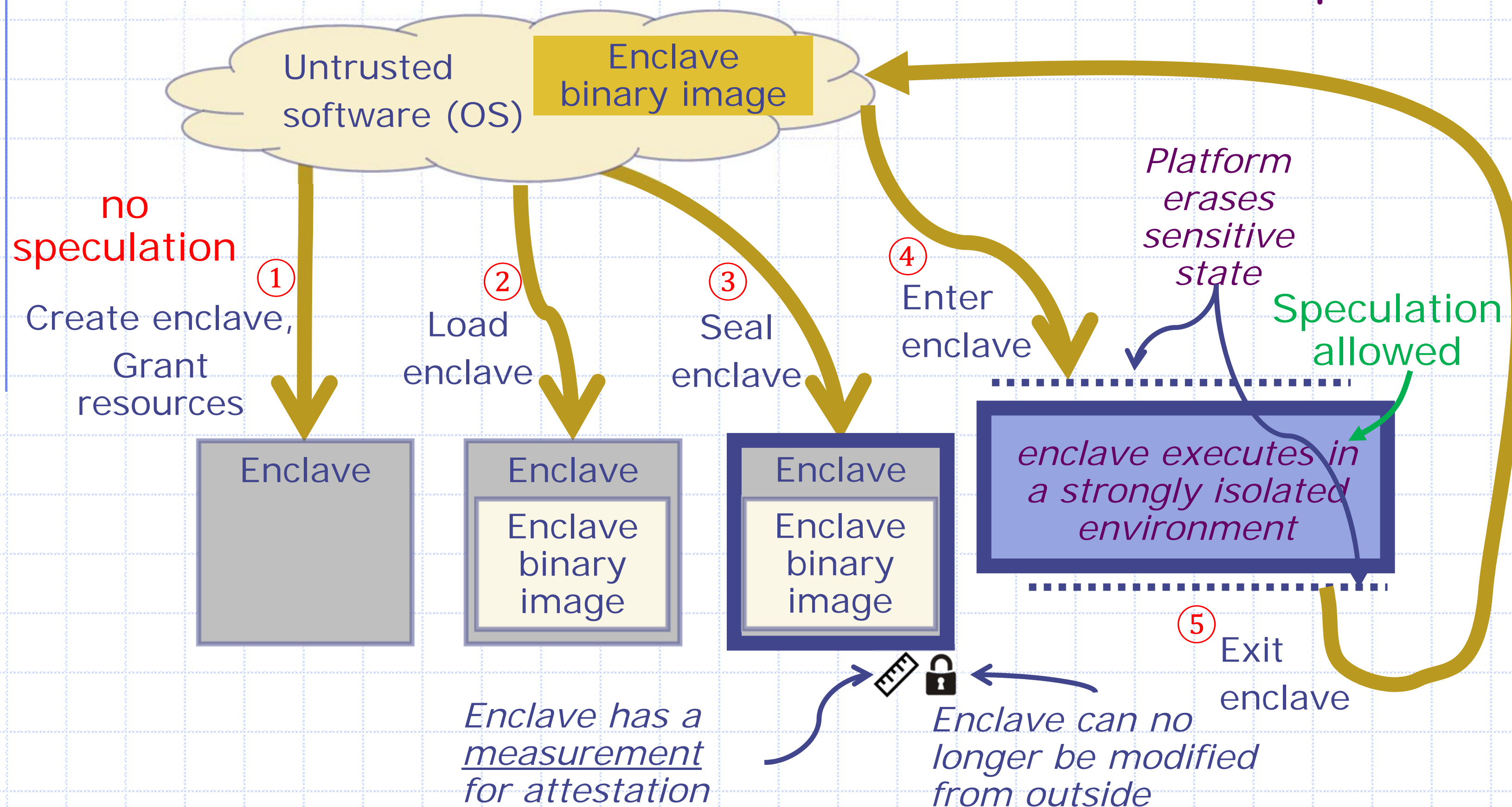
T. Bourgeat, I. Lebedev,, A. Wright, S. Zhang, Srini Devadas and Arvind [MICRO2019]



None of the blocks had to be redesigned but some required extra hardware checks.

Enclave Lifecycle

MI6 keeps this overall lifecycle of enclaves and enforces strong isolation in all phases



Sanctum[Costan, Lebedev, Devadas]

Isolation during the start and exit of enclave

- Threats to protect against
  - Enclave start: influence from the previous process on the same core
  - Enclave end: leakage to the next process on the same core
- Mechanisms to achieve isolation
  - Software can reset architectural states, i.e., contents of architectural registers and memory
  - Still need hardware support to reset microarchitectural states because of microarchitectural side channels!

Isolation during the start and exit of enclave

- Threats to protect against
  - Enclave start: influence from the previous process on the same core
  - Enclave end: leakage to the next process on the same core
- Mechanisms to achieve isolation
  - Software can reset architectural states, i.e., contents of architectural registers and memory
  - Still need hardware support to reset microarchitectural states because of microarchitectural side channels!

PURGE instruction: *ISA extension to reset microarchitectural state*

Semantics of the PURGE instruction

- After executing a PURGE instruction, the microarchitectural states in a core are reset to publicly known values. For example:
  - The core pipeline is empty
  - All buffers (e.g., ROB and LSQ) are empty, and pointers to the buffers are reset to a fixed position
  - States in branch predictors (e.g., branch history table) are reset
  - All TLBs are empty
  - L1 caches are empty
  - ...

Implementation of the PURGE instruction

Wait for older instructions to complete, including draining store buffer

Squash younger instructions

Wait for wrong-path activities (e.g., memory accesses) to complete

Reset microarchitectural states, e.g., flush L1 caches, TLBs, branch predictors, etc.

This idea works for both in-order and out-of-order processors

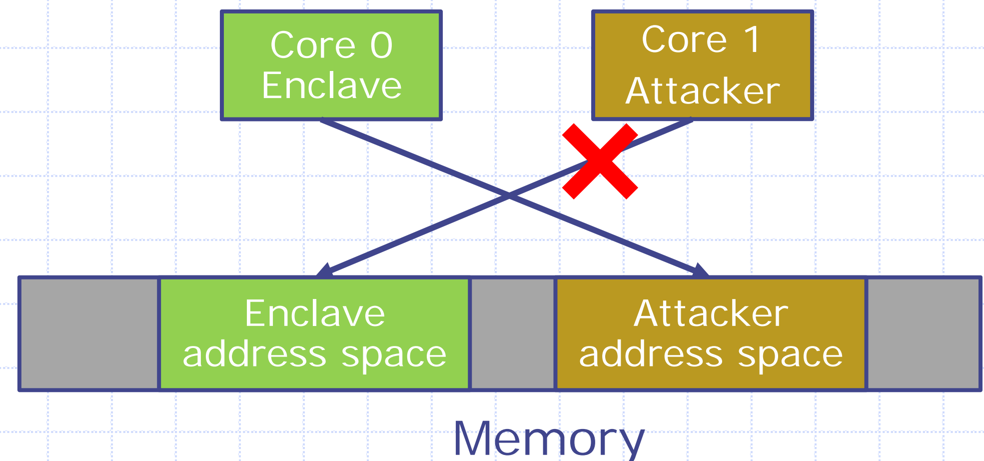
While an enclave is running

- Attacker can try to access memory of the enclave
- Enclave may accidently (speculatively) access memory of the attacker
- Through microarchitectural side channels,
  - Attacker can influence the execution of instructions or their timing in the enclave
  - Enclave can leak its secret inadvertently by affecting the execution or the timing of the attacker
    - Prime+Probe of shared cache set

While an enclave is running

Memory isolation

- Check the address of any memory access, including speculative accesses, instruction fetches, and page walks
  - Prevent attacker from issuing memory accesses to the address space of enclave



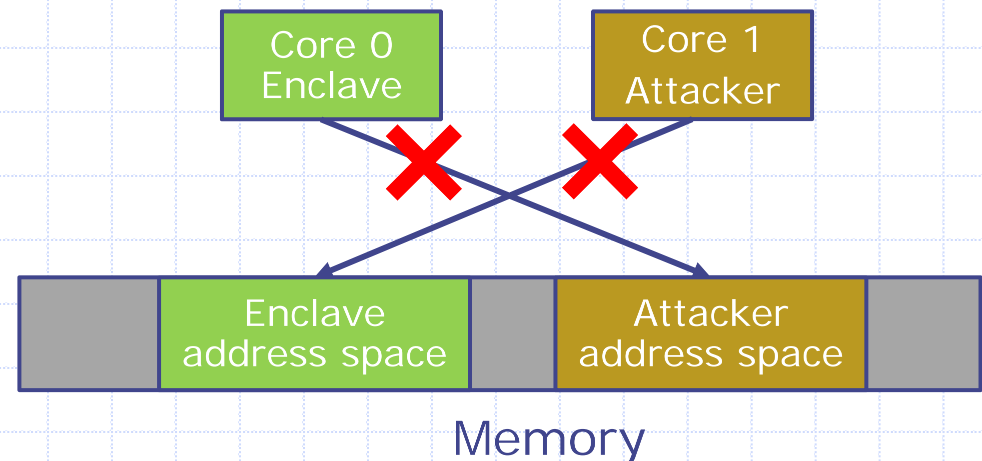
Mechanism borrowed from Sanctum [Ilia, S. Devadas]

While an enclave is running

Memory isolation

- Check the address of any memory access, including speculative accesses, instruction fetches, and page walks
  - Prevent attacker from issuing memory accesses to the address space of enclave
  - Prevent enclave from issuing memory accesses outside of the its own address space

It does not matter whether the core is speculative/out-of-order or not

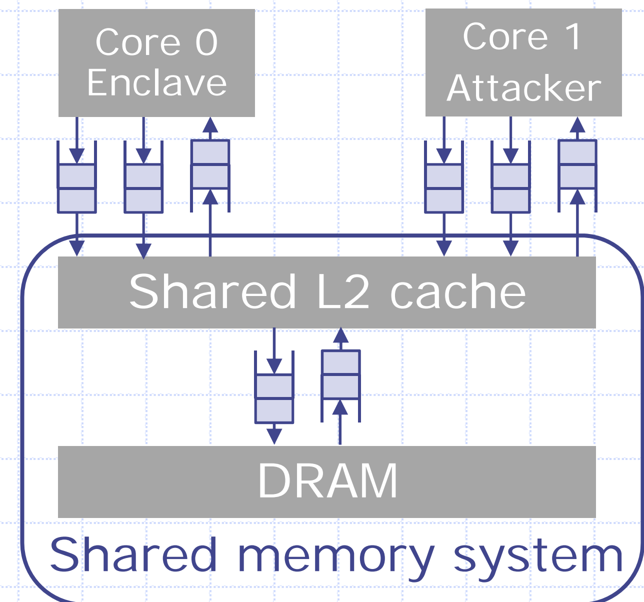


Mechanism borrowed from Sanctum [Ilia, S. Devadas]

While an enclave is running

Block memory side channels

- Any microarchitectural resource shared by the enclave and the attacker can become a side channel
- Resources in the core are owned exclusively by the enclave but everything in the memory system is shared
  - L2 cache slots
  - L2 cache miss status handling registers (MSHRs)
  - L2 cache internal bandwidth
  - DRAM access bandwidth



Microarchitectural isolation:
Partition every shared resource to isolate enclave from attackers

While an enclave is running

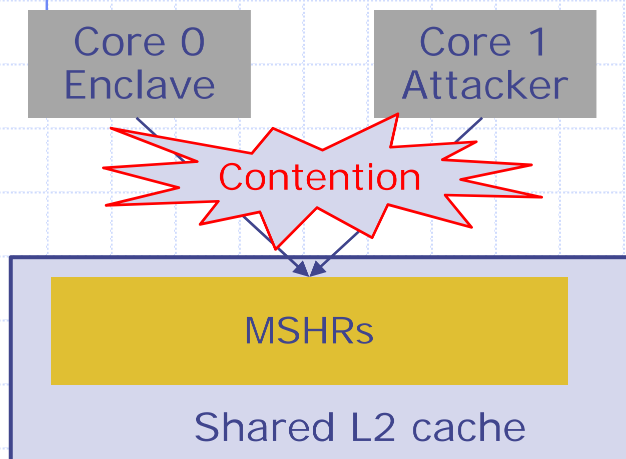
Partition L2 cache slots

- Reserve a fixed set of L2 cache sets for an enclave
- Borrowed the idea from Sanctum

While an enclave is running

Partition L2 MSHRs

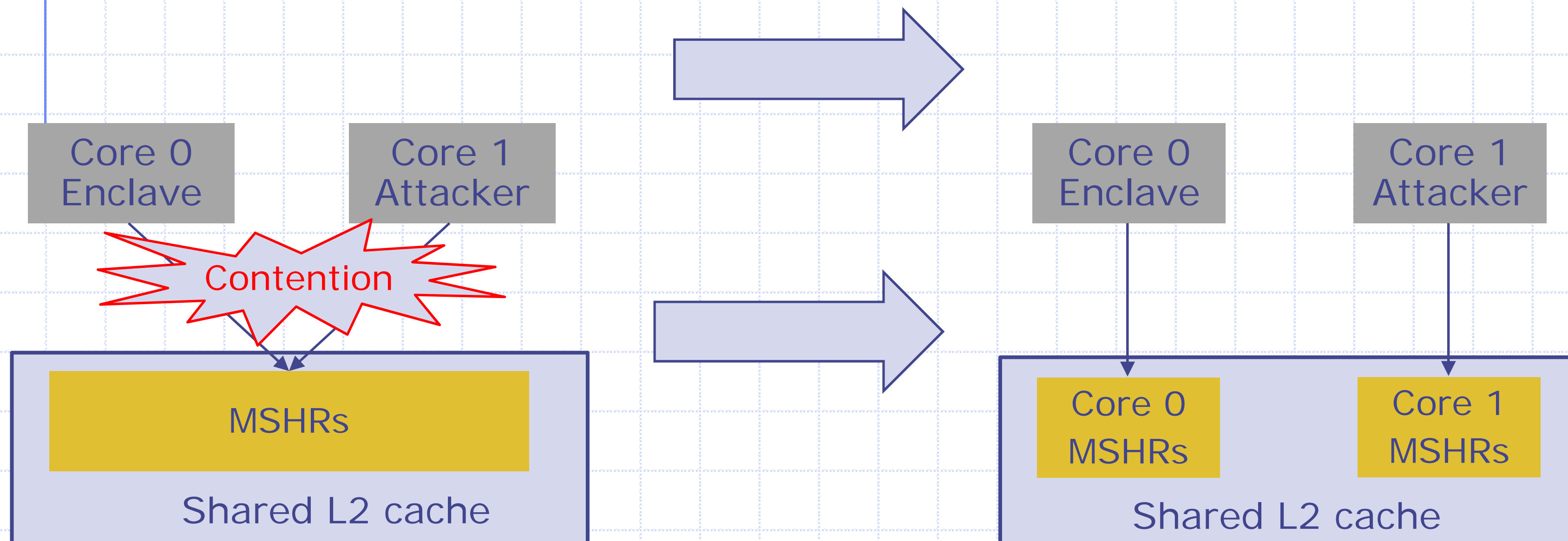
- Problem: requests from different cores are contending for the limited MSHR entries



While an enclave is running

Partition L2 MSHRs

- Problem: requests from different cores are contending for the limited MSHR entries
- Solution: divide the MSHR entries equally across cores



While an enclave is running

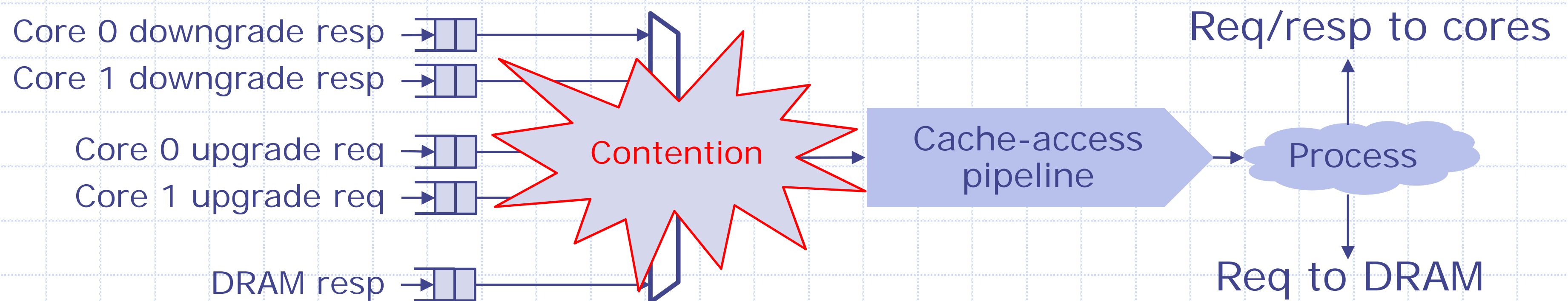
Partition DRAM access bandwidth

- Each missing request in MSHR can generate at most 2 DRAM requests (writeback + refill)
  - No contention if DRAM bandwidth > 2X(# of MSHRs)
- In reality, this problem is much harder, because DRAM bandwidth and access latency depends on access pattern (e.g., random vs. sequential), and DRAM controllers rearrange requests
  - In MI6, we assume a DRAM model with constant latency and bandwidth, so we only need to limit the number of MSHRs < DRAM bandwidth / 2

While an enclave is running

Partitioning L2 internal bandwidth

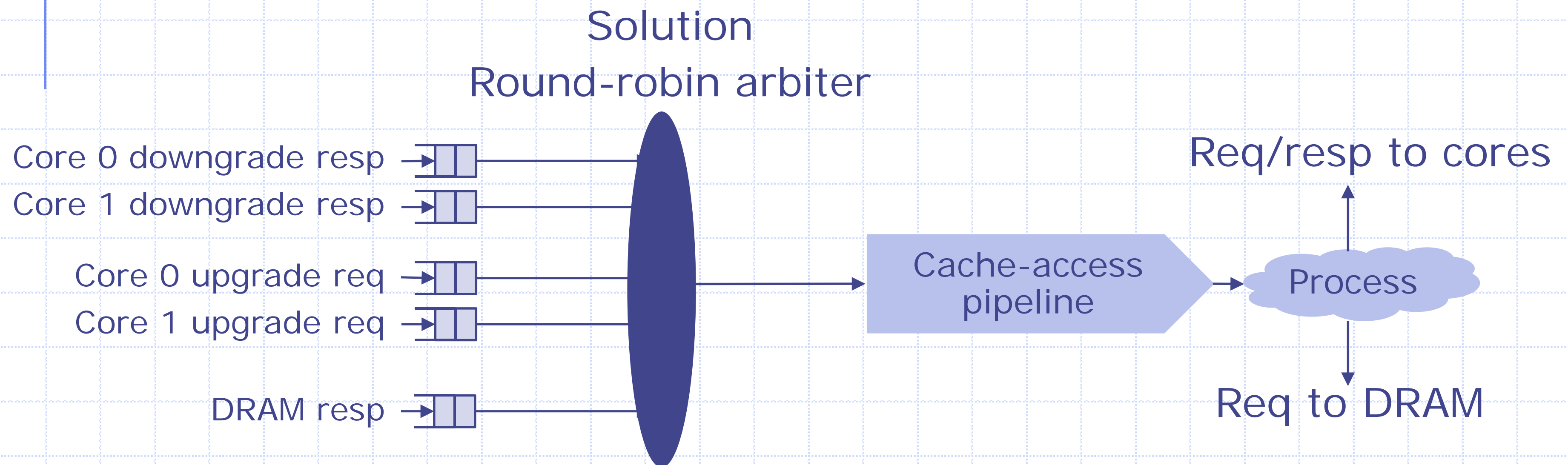
- Problem: messages from different cores are contending for accessing the cache arrays



While an enclave is running

Partitioning L2 internal bandwidth

- Problem: messages from different cores are contending for accessing the cache arrays
- Solution: use a round-robin arbiter to fairly arbitrate access to cache arrays across different cores



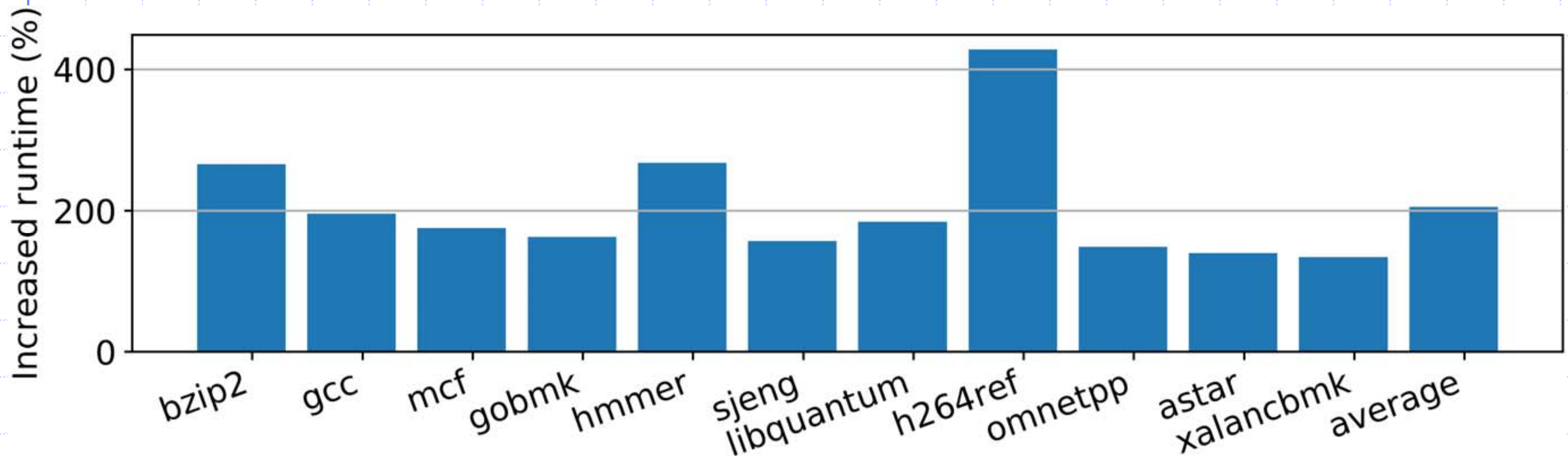
How much does this strong isolation of processes cost?

Performance evaluation:

The naïve solution - turning off speculation

Not our solution

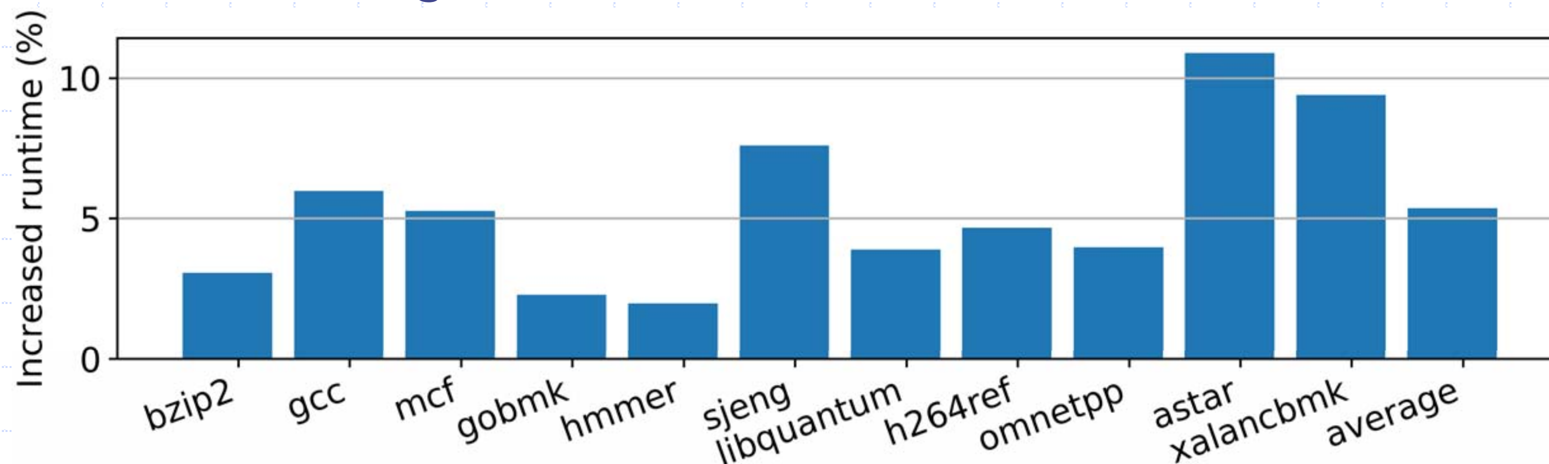
- Strictly less secure than our solution
- Overhead: average 205%, maximum 427%



Performance evaluation:

PURGE instruction

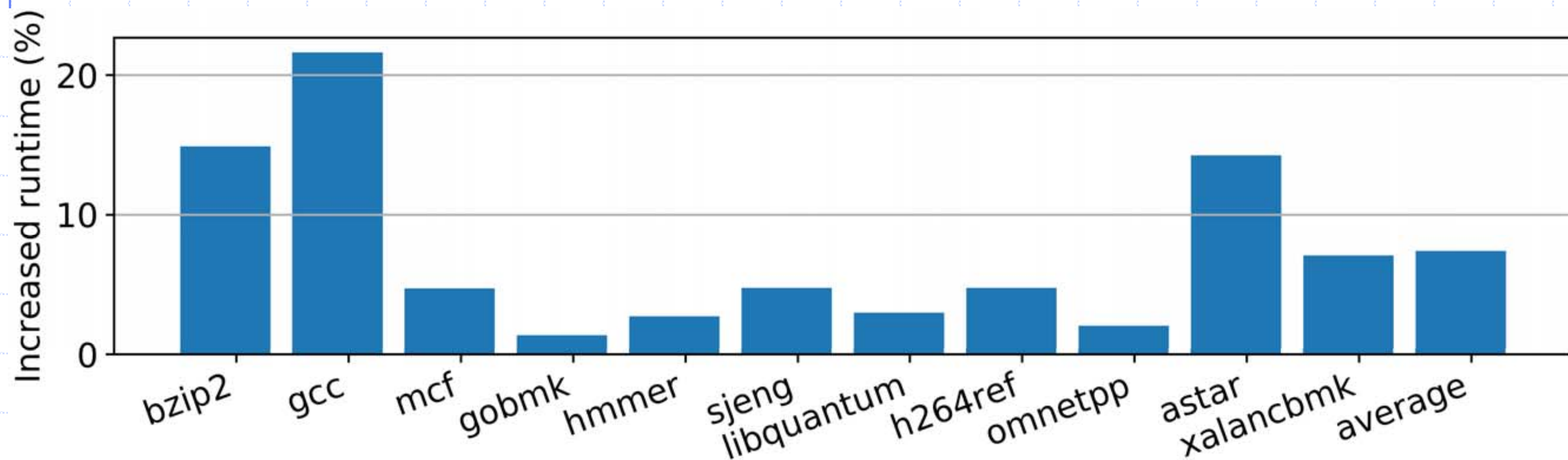
- Execute a PURGE instruction whenever the processor context is changed
  - exceptions, interrupts, system calls, return from trap handling
- Overhead: average 5.4%, maximum 10.9%
  - Mostly caused by the cold branch-predictor after executing PURGE



Performance evaluation:

Partitioning L2 cache slots

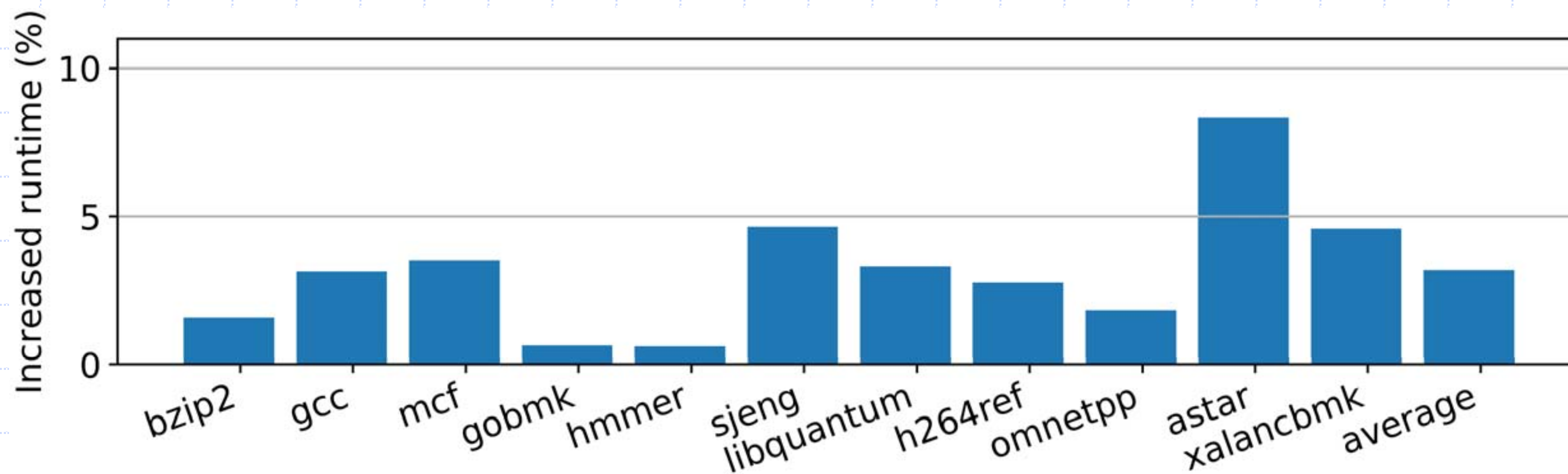
- Cache partitioning is commonly used to achieve quality of service
- Assuming the partition size is fixed, our partition scheme may increase cache conflicts compared to an ideal scheme
- Overhead: average 7.4%, maximum 21.6%



Performance evaluation:

Partitioning MSHR and DRAM bandwidth

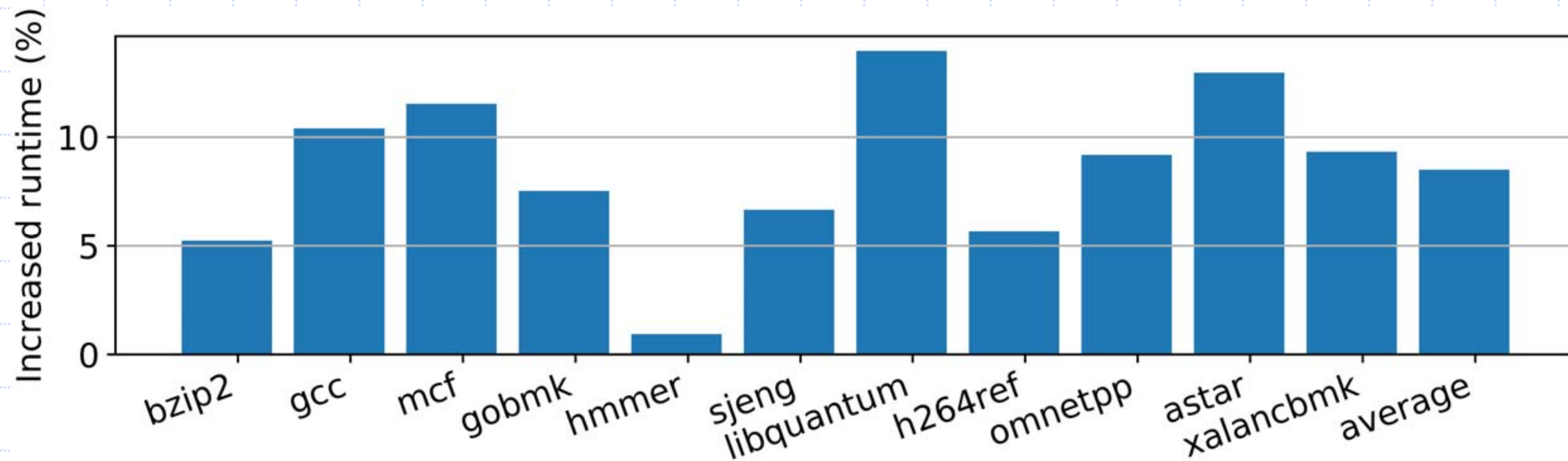
- Number of MSHRs is reduced
- Associativity of MSHR is reduced in case MSHRs are banked
- Worst case analysis:
  - 16 fully associative → 4 Banks x 3 per bank
- Overhead: average 3.2%, maximum 8.3%



Performance evaluation:

Partitioning L2 internal bandwidth

- Increased latency in accessing L2 cache arrays
  - For 16 cores, increase by 8 cycles on average
- Overhead: average 8.5%, maximum 14%

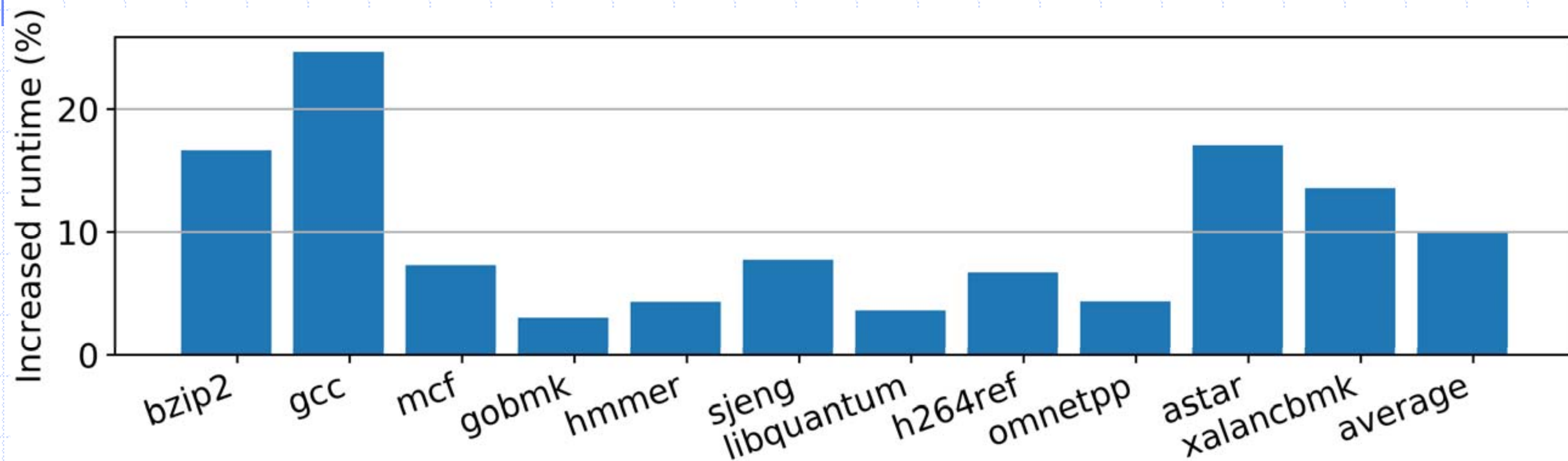


Performance evaluation

putting it all together

- Overhead: average 16.4%, maximum 34.8%

Much faster and more secure than turning off speculative execution (average overhead 205%, maximum overhead 427%)



Conclusion

- Speculation is not harmful in itself, if done within bounds
- The cost of checking at the boundaries is significantly less than the cost of turning off speculation
- We reduce the performance cost of security from 200% to ~20%
- PURGE instruction and memory isolations have been implemented and tested
- Shared state in the L2 cache are a work in progress

We can build high performance microprocessors on which it is possible to write secure software