Ghosting the Spectre

(what you don’t know can hurt you)

Allison Randal
Rivos / University of Cambridge

Transient Execution Vulnerabilities

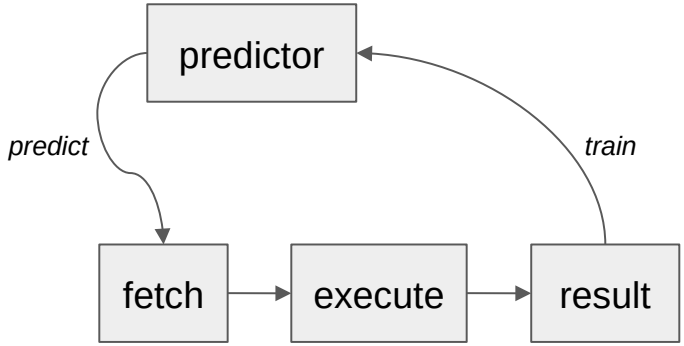
- Root cause overview
- PhD work
- RISC-V working group

Root causes and mitigation options

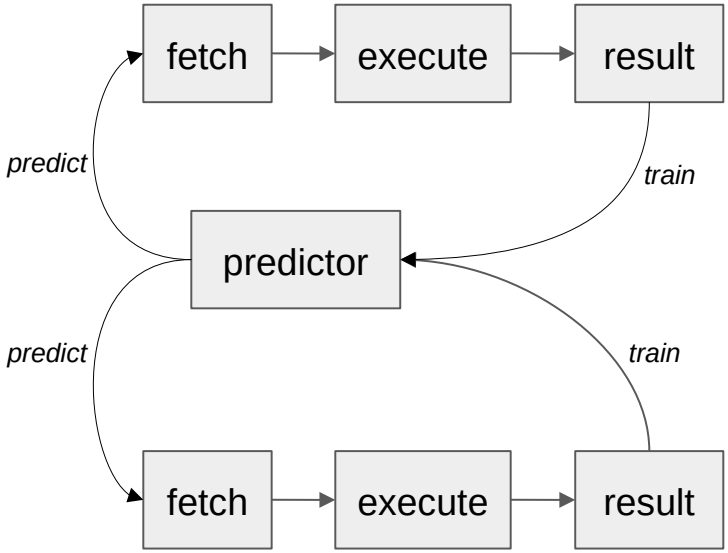
Vulnerability Class	Isolate	Flush	Disable
Spectre variants mistrain speculative predictors to leak privileged data or manipulate control flow.	Isolate predictors to limit the impact of cross-context attacks (provides no protection against same-context attacks). Isolate caches to limit the impact of cross-context attacks. Partitioning, tagging, and buffering are some options for implementing isolation.	Flush predictor state to temporarily disrupt the effect of mistrained predictors. Invalidate L1 data cache (and other caches) to temporarily disrupt the effect of leaked data.	Disable predictions and training to temporarily fully protect against Spectre class vulnerabilities.
Meltdown variants take advantage of exceptions that are temporarily suspended during transient execution (speculative or out-of-order), to leak privileged data or overwrite privileged data.	Isolate shared microarchitectural state to limit the impact of cross-context attacks. Avoid transient updates to shared microarchitectural state, such as L1 data cache and other caches.	Transient exceptions immediately clear transient microarchitectural state invalidated by the exception (don’t wait until the exception is raised on commit).	Temporarily prevent transient execution (speculative or out-of-order) of any instructions that depend on a load/read/write until permission checks are complete.

Naive mental model

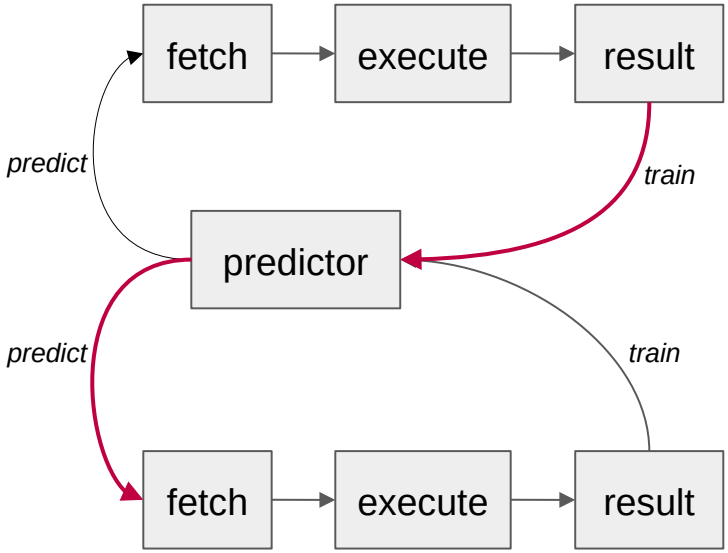
Slightly more realistic mental model



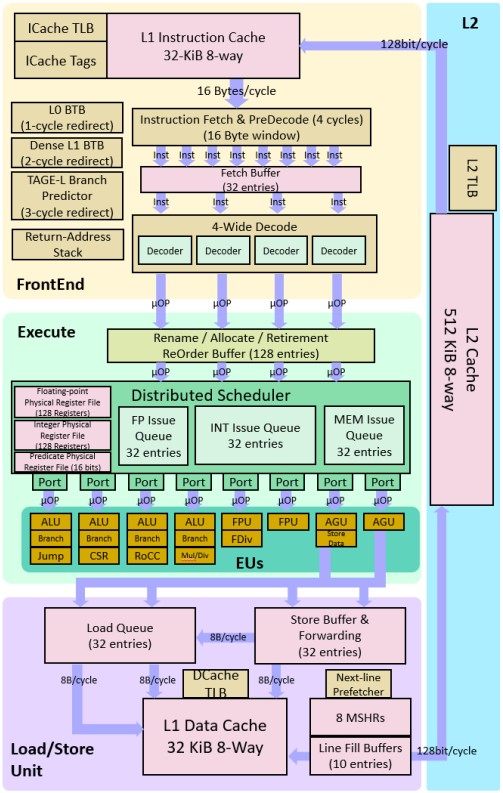
Slightly more realistic mental model



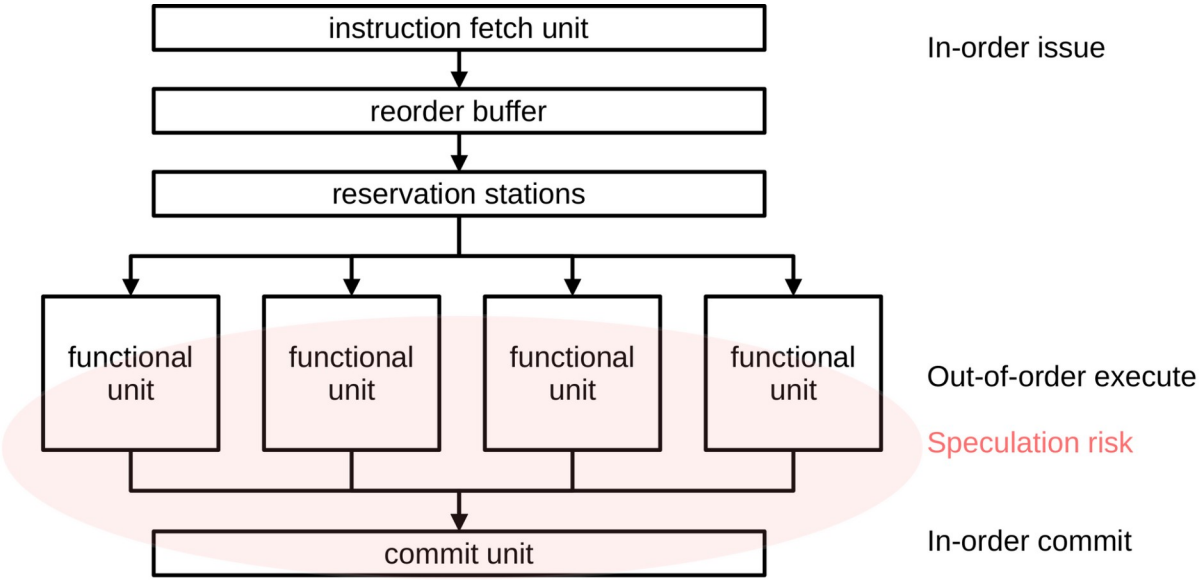
Slightly more realistic mental model



Too much information



Zone of risk


instruction fetch unit
reorder buffer
reservation stations
functional unit
functional unit
functional unit
functional unit
commit unit
In-order issue
Out-of-order execute
Speculation risk
In-order commit

Root causes and mitigation options

Vulnerability Class	Isolate	Flush	Disable
Spectre variants mistrain speculative predictors to leak privileged data or manipulate control flow.	Isolate predictors to limit the impact of cross-context attacks (provides no protection against same-context attacks). Isolate caches to limit the impact of cross-context attacks. Partitioning, tagging, and buffering are some options for implementing isolation.	Flush predictor state to temporarily disrupt the effect of mistrained predictors. Invalidate L1 data cache (and other caches) to temporarily disrupt the effect of leaked data.	Disable predictions and training to temporarily fully protect against Spectre class vulnerabilities.
Meltdown variants take advantage of exceptions that are temporarily suspended during transient execution (speculative or out-of-order), to leak privileged data or overwrite privileged data.	Isolate shared microarchitectural state to limit the impact of cross-context attacks. Avoid transient updates to shared microarchitectural state, such as L1 data cache and other caches.	Transient exceptions immediately clear transient microarchitectural state invalidated by the exception (don’t wait until the exception is raised on commit).	Temporarily prevent transient execution (speculative or out-of-order) of any instructions that depend on a load/read/write until permission checks are complete.

Predictors used in attacks

- Branch Target Buffer (BTB), direct and indirect branches
- Branch History Buffer (BHB), used by BTB
- Pattern History Table (PHT), conditional branches
- Return Stack Buffer (RSB) / Return Address Stack (RAS), returns
- Memory Disambiguator, memory loads (and stores)

Example: Branch Target Buffer (BTB)

- Spectre-BTB[1] (Spectre variant 2) mistrains direct or indirect branch predictions
  - Target: redirect transient control flow to an arbitrary destination (Branch Target Injection)
  - Attacks succeed cross/same-address-space, in-place & out-of-place
- SgxPectre[2] exposes TEE secret data (provisioning keys, seal keys, attestation keys) from outside the TEE.
- Spectre-BTB-SA-IP[3] bypasses mitigations that flush or partition the BTB
- Spectre-BHB[4] bypasses mitigations that isolate the BTB, demonstrating that BTB attacks can succeed by mistraining only the BHB

[1]P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom (2018) “Spectre Attacks: Exploiting Speculative Execution,” *arXiv:1801.01203*.
[2]G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai (2019) “SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution,” in 2019 IEEE European Symposium on Security and Privacy, pp. 142–157.
[3]C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss (2019) “A Systematic Evaluation of Transient Execution Attacks and Defenses,” *arXiv:1811.05441*.
[4]E. Barberis, P. Frigo, M. Muench, H. Bos, and C. Giuffrida (2022) "Branch History Injection: On the Effectiveness of Hardware Mitigations Against Cross-Privilege Spectre-v2 Attacks", In USENIX Security.

In-place & out-of place

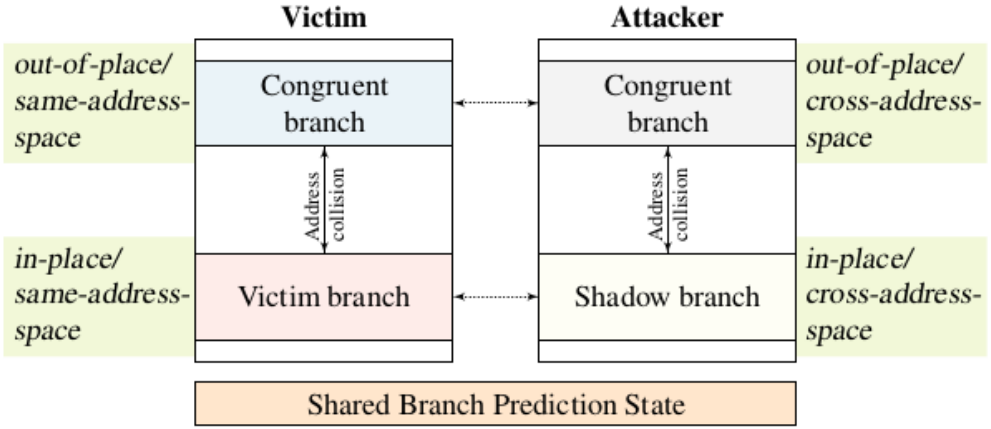


Figure 3: A branch can be mistrained either by the victim process (*same-address-space*) or by an attacker-controlled process (*cross-address-space*). Mistraining can be achieved either using the vulnerable branch itself (*in-place*) or a branch at a congruent virtual address (*out-of-place*).

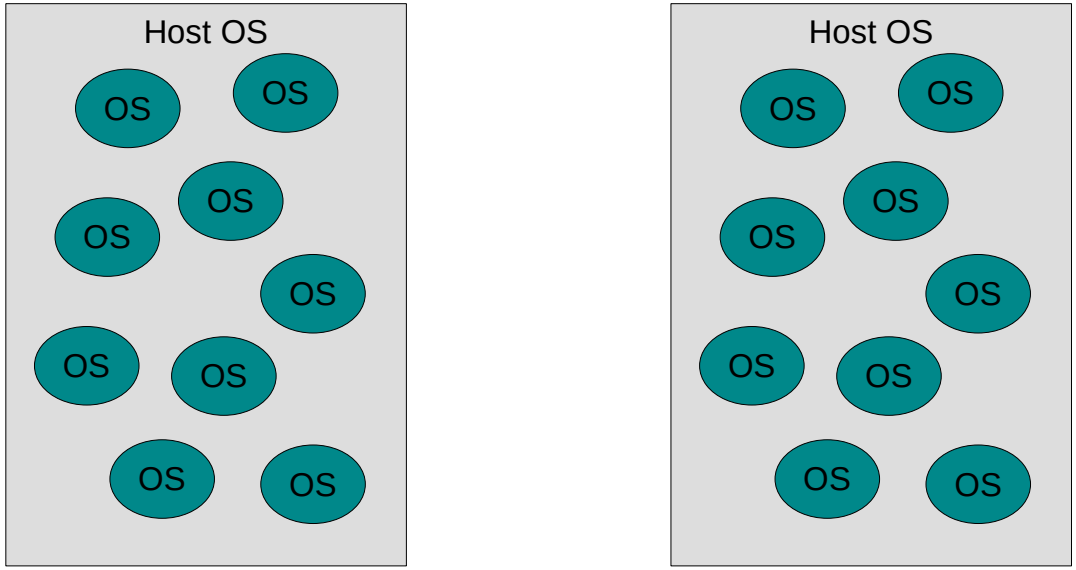
C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss (2019) “A Systematic Evaluation of Transient Execution Attacks and Defenses,” *arXiv:1811.05441*.

PhD work[1]

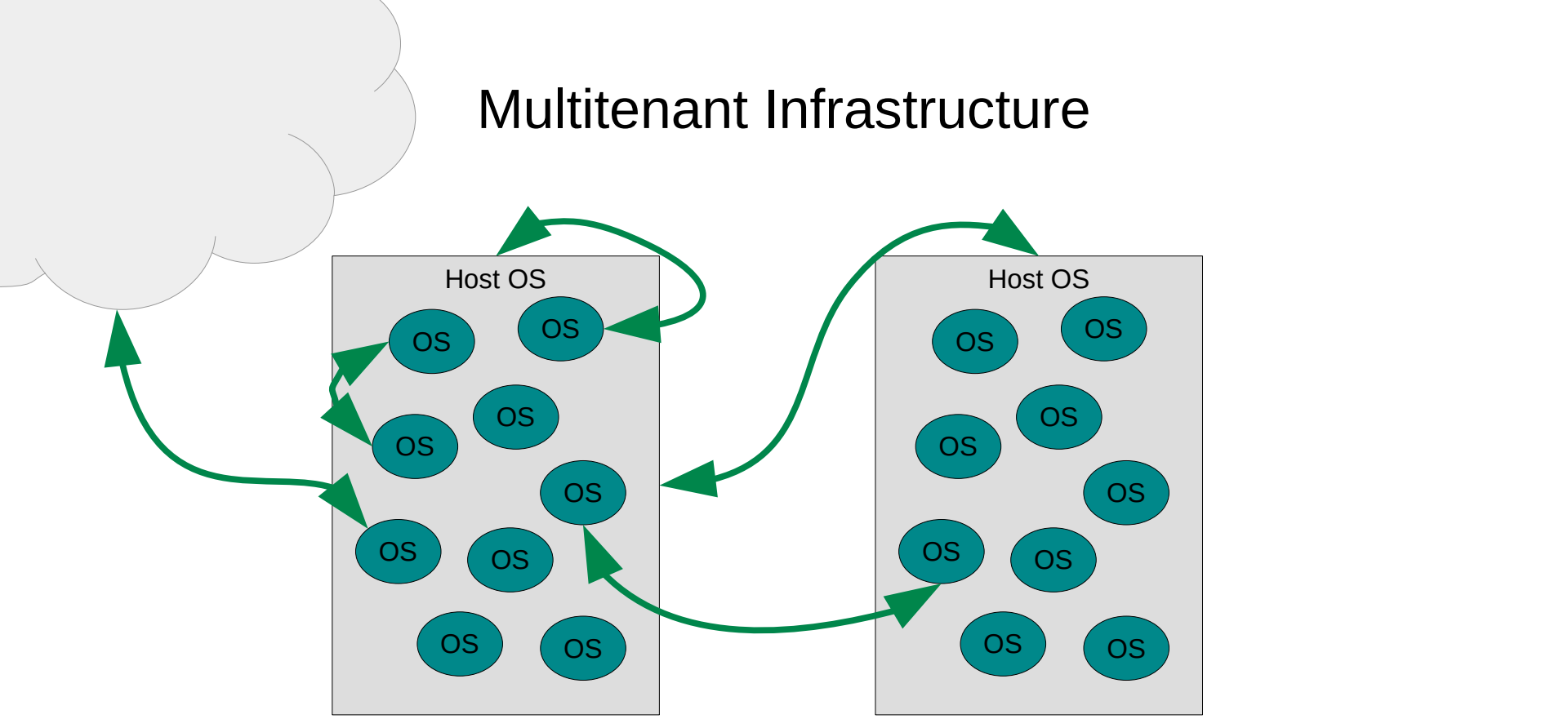
- How did we get here?
- Is there any way I'll ever trust multitenant computing again?
- Design space exploration
- "disable" mitigation techniques
- Prototype 3 variations on BOOMv3 RISC-V core
- Simulation on Amazon FPGA (with FireMarshal and FireSim)

[1]A. Randal (2021) *Ghosting the Spectre: fine-grained control over speculative execution*, University of Cambridge. https://cam.lohutok.net/publication/2021-ghosting-the-spectre/ghosting_the_spectre.pdf

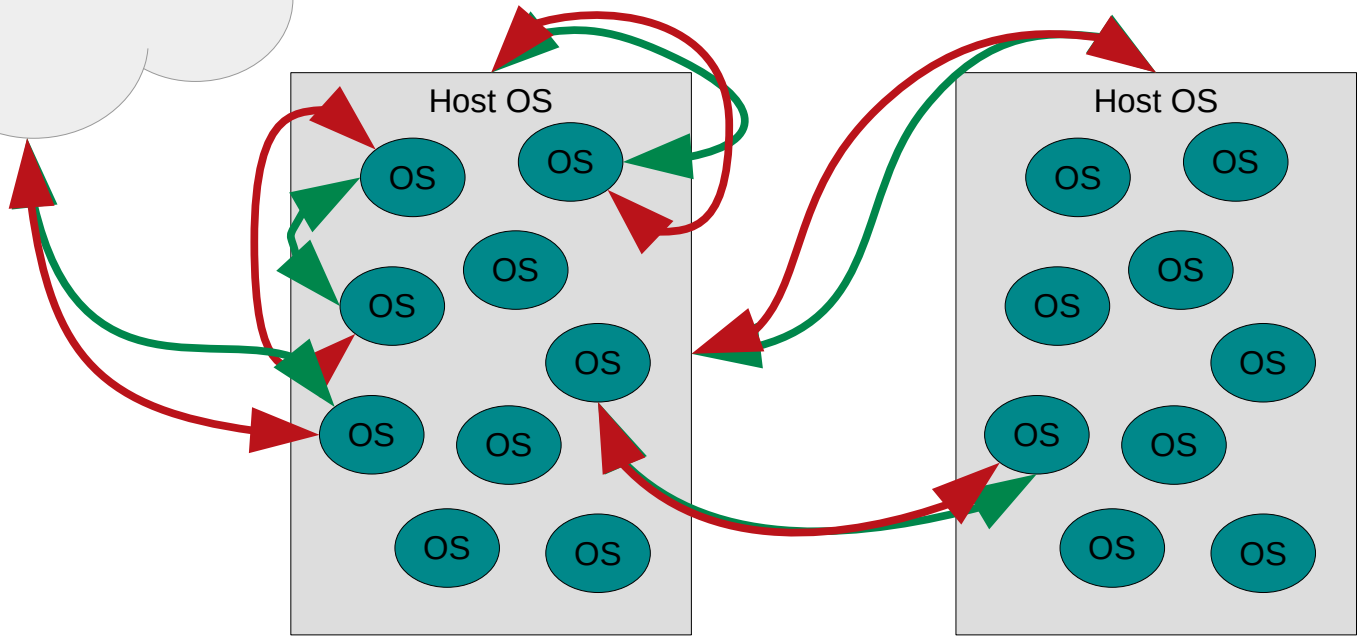
Multitenant Infrastructure



Multitenant Infrastructure



Secure Isolation



How did we get here?[1]

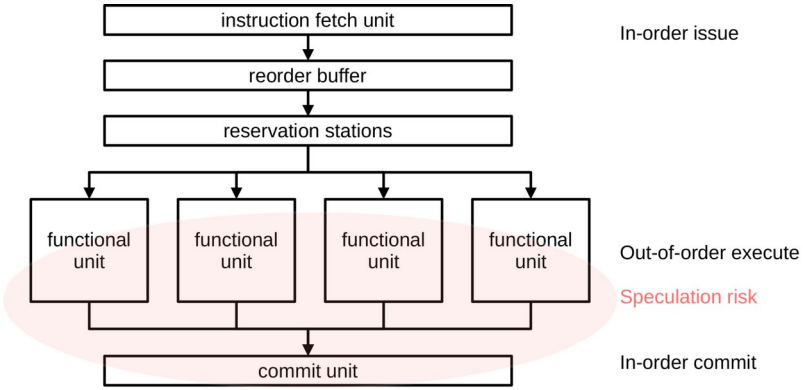
- Early co-design of hardware/software
- Increasing architectural stratification and standardization in hardware and software
- Modular and recombinable:
  - CPUs, memory, storage, etc
  - Kernels, system utilities, operating systems, applications, etc.
- Improved ease of development and maintenance
- Harder to reason about security properties across abstraction layers
- Few engineers work across the full stack from microarchitecture to applications

CAP[2], (C) 2004, Daderot, CC BY-SA 3.0

[1]A. Randal (2020) “The Ideal Versus the Real: Revisiting the History of Virtual Machines and Containers,” ACM Computing Surveys, vol. 53, no. 1, 5:1–5:31.
[2]R. M. Needham and R. D. H. Walker (1977) “The Cambridge CAP Computer and its protection system”, *In Proceedings of the Sixth ACM Symposium on Operating Systems Principles*, 1–10, ACM.

How did we get here?

- Assumption: speculative execution can safely create “transient” microarchitectural state, as long as it’s cleaned up on commit, not architecturally visible
- Reality: transient microarchitectural state can leak secret information, run arbitrary code gadgets



How did we get here?

- Percival[1] identified risk combining speculative execution, simultaneous multithreading, dynamic pipeline scheduling, multilevel memory caches, and hardware prefetching
- Spectre[2] and Meltdown[3] realize the full extent of the security impact, more sophisticated and severe than previously thought possible
- Many variants followed[4], continue to discover new variants

[1]C. Percival (2005) “Cache Missing for Fun and Profit,” in *Proceedings of BSDCan 2005*, Ottowa, Canada, p. 13.
[2]P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom (2018) “Spectre Attacks: Exploiting Speculative Execution,” *arXiv:1801.01203*.
[3]M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg (2018) “Meltdown,” *arXiv:1801.01207*.
[4]C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss (2019) “A Systematic Evaluation of Transient Execution Attacks and Defenses,” *arXiv:1811.05441*.

Trust in Computing

- Cloud, multitenant infrastructures
- “Trusted Computing”: hardware enhancements and software improvements to improve computer security
  - attestation, cryptographically signed software
- “Confidential Computing”: protects data in use
  - encrypted memory, secure enclave/trusted execution environment
- Rely on hardware isolation features, undermined by speculative execution

Trust in Computing

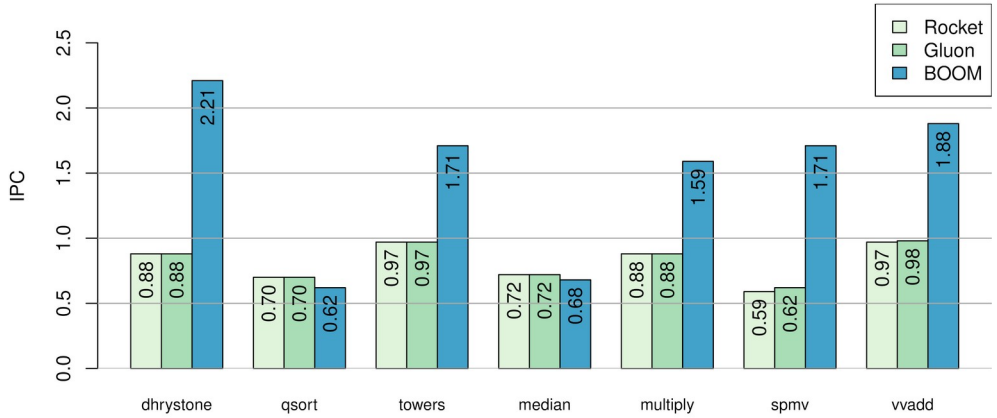
- Speculation has performance benefits
- Restricting speculation has security benefits
  - Share less, improve security
  - Share nothing, better security
- Can we combine speculation and no speculation?
- Can we give systems software developers the ability to choose?
- How would that work?

Gluon: heterogeneous multicore

- Pair a “big” speculative core with a “little” non-speculative core
- Analogous to ARM big.LITTLE architecture, for security (rather than performance)
- Comprehensive protection for workloads on non-spec core
  - Cannot mistrain other workloads (restrict malicious)
  - Cannot be mistrained by other workloads (protect confidential)
- Still need to mitigate the speculative core

Gluon: heterogeneous multicore

- Learned: performance determined by which core the workload runs on
- Learned: not viable for large-scale servers, because of inflexible resource allocation

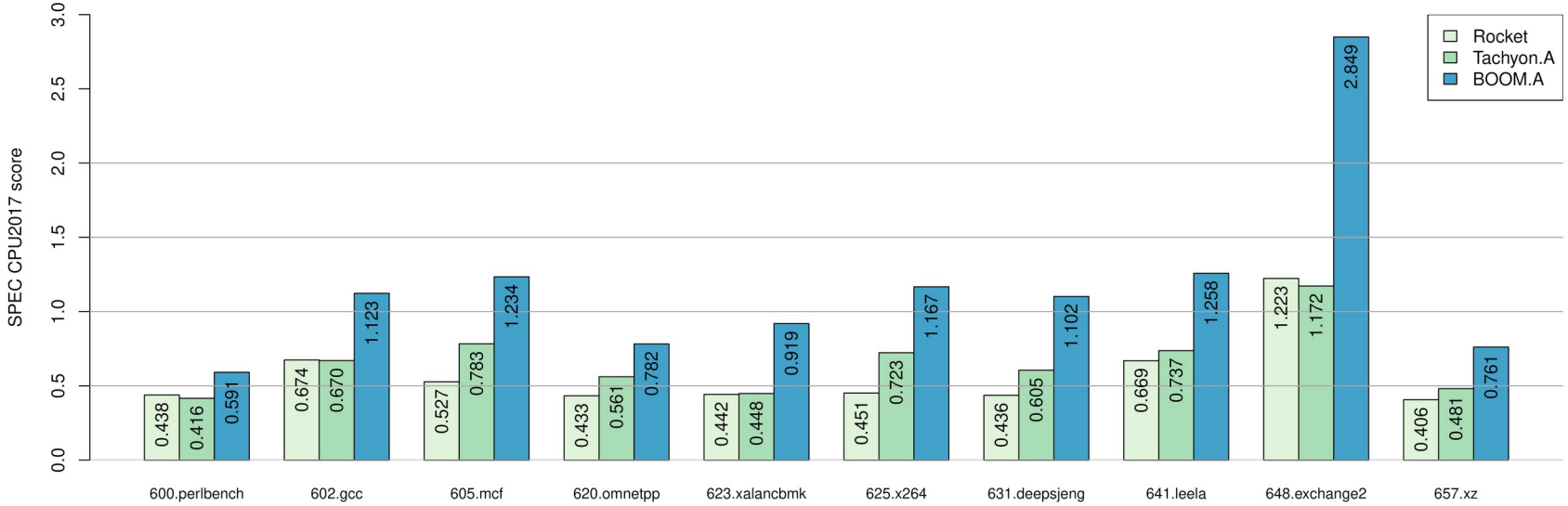


Tachyon: non-speculative

- Baseline for performance comparison
- Eliminate speculation entirely
- Protects against all known variants, and unknown future variants
- Performance always non-speculative

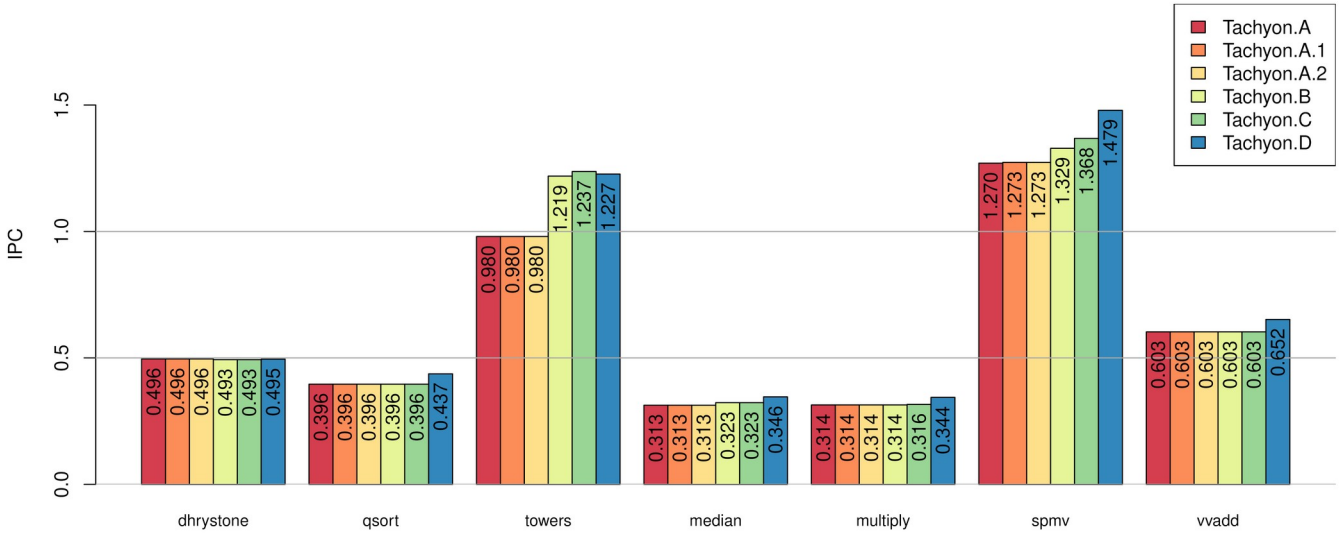
Tachyon: non-speculative

- Learned: performance is better than expected, on par with other comprehensive mitigations



Tachyon: non-speculative

- Learned: Can improve performance by increasing microarchitectural parallelism (fetch width, ROB, reservation stations, and execution units)

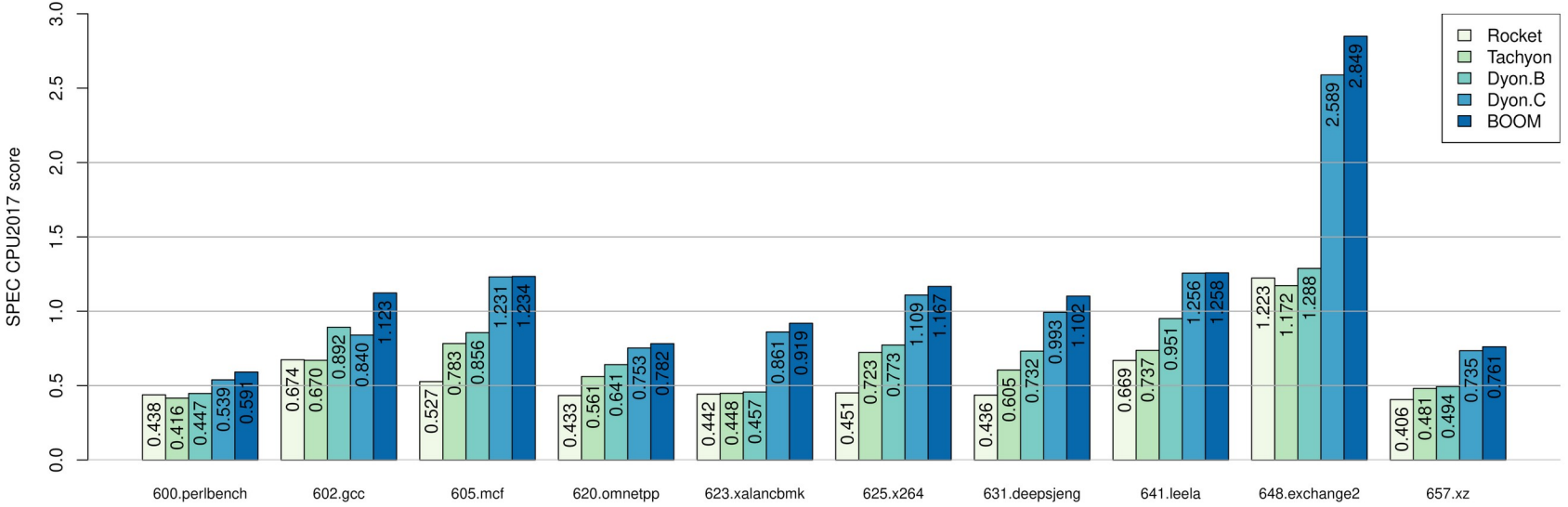


Dyon: demi-speculative

- Combines speculative and non-speculative execution in a single core
- Prototyped as an ISA extension
- Alternative: enabled for specific security domain (e.g. TEE or VM)
- Protects against known and future variants

Dyon: demi-speculative

- Learned: Performance determined by use of non-speculative regions
- Learned: combining speculative and non-speculative features in a single core is feasible



(food for thought)

Would you use it?

RISC-V working groups

- Microarchitecture Side Channels (Security) SIG
  - Upcoming: Dominic Rizzo, OpenTitan transient execution mitigation choices, June 27th
- Trusted Computing SIG & Trusted Execution Environments
- Reliability, Availability, Serviceability (RAS)
- Quality of Service (QoS)
- (also hiring)

Bonus Material[1]

[1]A. Randal (2022) *Transient Execution - Implementer's Security Guide (DRAFT)*, RISC-V International.
https://github.com/riscv-admin/uarch-side-channels/blob/main/docs/transient_implementer_guide.adoc

Tagged predictions

- Isolation by tagging, privilege mode and VM
- Protects against cross-domain (U-mode/S-mode/M-mode & VM) attacks
- No protection against same-domain attacks
- Performance is good, but high area cost (duplicate predictions)

BTB	BHB	PHT	RSB	MD
Tagged predictions	Tagged predictions	Tagged predictions	Tagged predictions	Tagged predictions

Flush/invalidate predictions

- Protection by flushing or invalidating, e.g. on privilege mode or address space change, or manually
- Protects against cross-domain attacks
- No protection against same-domain attacks
- Performance is poor, e.g. 90% increase in PHT mispredictions after flush[1]

BTB	BHB	PHT	RSB	MD
Flush predictions	Flush predictions	Flush predictions	Flush predictions	Flush predictions

[1]I. Vougioukas, N. Nikoleris, A. Sandberg, S. Diestelhorst, B. M. Al-Hashimi, and G. V. Merrett (2019) "BRB: Mitigating Branch Predictor Side-Channels," 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 466-477.

Buffer predictions

- Isolation by buffering, e.g. by privilege mode, address space, hart, VM, process, sandbox or manually
- Protects against cross-domain attacks
- No protection against same-domain attacks
- Performance depends on implementation, penalty higher for large predictors (like BTB or MD), lower for small predictors (PHT or RSB)

BTB	BHB	PHT	RSB	MD
Buffer on domain change	Buffer on domain change	Buffer on domain change	Buffer on domain change	Buffer on domain change

Untrusted predictions

- Option for TEE/VM
- Lightweight disabling technique, don’t need tagging or flushing
- Predictions made as usual, but execution of dependent instructions held by a “control dependency” tag until branch/return resolves
- Some similarity to short forward branch optimizations

BTB	BHB	PHT	RSB	MD
Normal predictions, hold execution	Normal (BTB makes predictions)	Normal predictions, hold execution	Normal predictions, hold execution	Normal predictions, hold execution

Performance baseline: no speculation

- For the sake of comparison
- Out-of-order, but not speculative
- Mitigation alternatives that perform worse than non-speculative are ruled out
- Other alternatives measured on performance gains over the baseline

BTB	BHB	PHT	RSB	MD
Removed	Removed	Removed	Removed	L0 speculation cache