

# Introducing Speculative Execution Attacks

No HMC colloquium today;  
HW3 Autograder to be  
released this afternoon;  
No class next Friday

For fun: <https://leaky.page/>

# About speculative execution vulnerabilities in ARM-based and Intel CPUs

- Apple has released security updates for macOS Sierra and El Capitan with mitigations for Meltdown.
- Apple has released updates for iOS, macOS High Sierra, and Safari on Sierra and El Capitan to help defend against Spectre.
- Apple Watch is unaffected by both Meltdown and Spectre.

Security researchers have recently uncovered security issues known by two names, Meltdown and Spectre. These issues apply to all modern processors and affect nearly all computing devices and operating systems. All Mac systems and iOS devices are affected, but there are no known exploits

Image Credit: <https://support.apple.com/en-us/101886>

DOI:10.1145/3399742

## Spectre Attacks: Exploiting Speculative Execution

By Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom

### Abstract

Modern processors use branch prediction and speculative execution to maximize performance. For example, if the destination of a branch depends on a memory value that is in the process of being read, CPUs will try to guess the destination and attempt to execute ahead. When the memory value finally arrives, the CPU either discards or commits the speculative computation. Speculative logic is unfaithful in how it executes, can access the victim's memory and registers, and can perform operations with measurable side effects.

Spectre attacks involve inducing a victim to speculatively perform operations that would not occur during correct

attacks, which do not require external measurement equipment. Although some attacks exploit software logic errors, other software attacks leverage hardware properties to infer sensitive information. Attacks of the latter type include microarchitectural attacks exploiting cache timing<sup>3, 6, 17</sup> and branch prediction history.<sup>1</sup> Software-based techniques have also been used to induce computation errors, such as fault attacks that alter physical memory<sup>11</sup> or internal CPU values.<sup>25</sup>

Several microarchitectural design techniques have facilitated the increase in processor speed over the past decades. One such advancement is speculative execution, which is widely used to increase performance and involves having

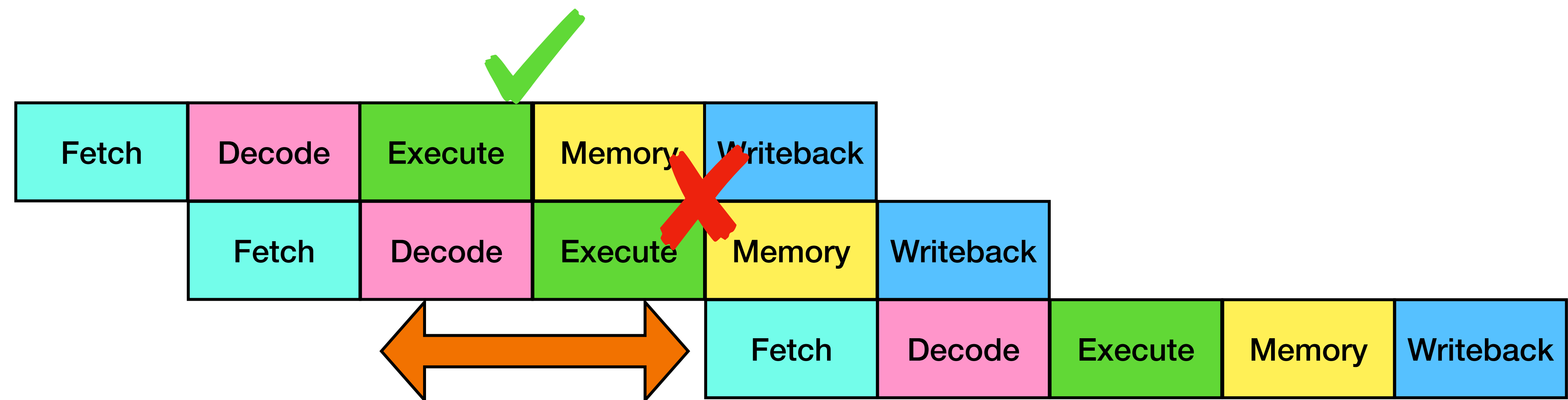
Image Credit: <https://dl.acm.org/doi/pdf/10.1145/3399742>

# Outline

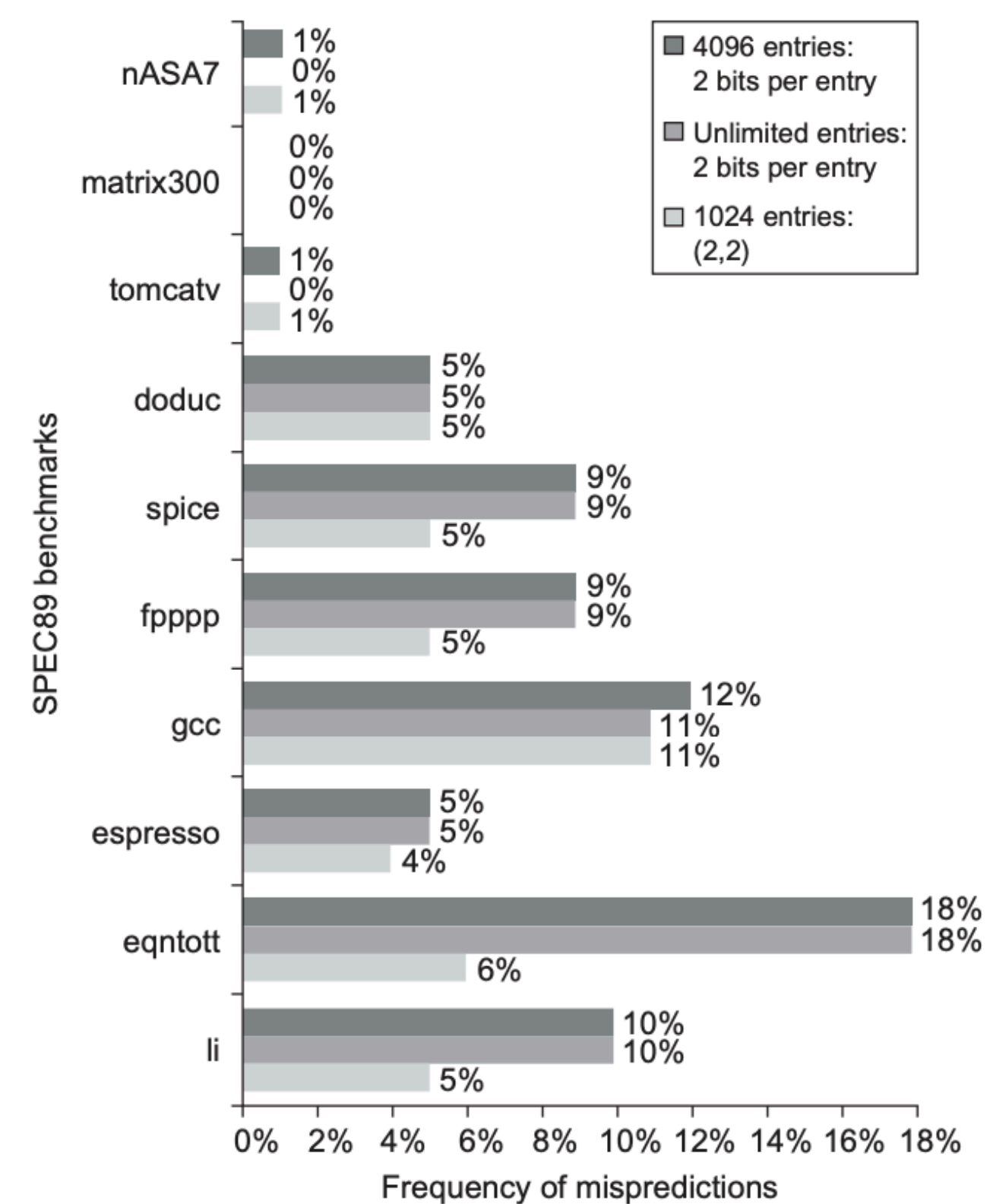
- Revisiting branch prediction
- Introducing speculative execution attacks
- Feedback form



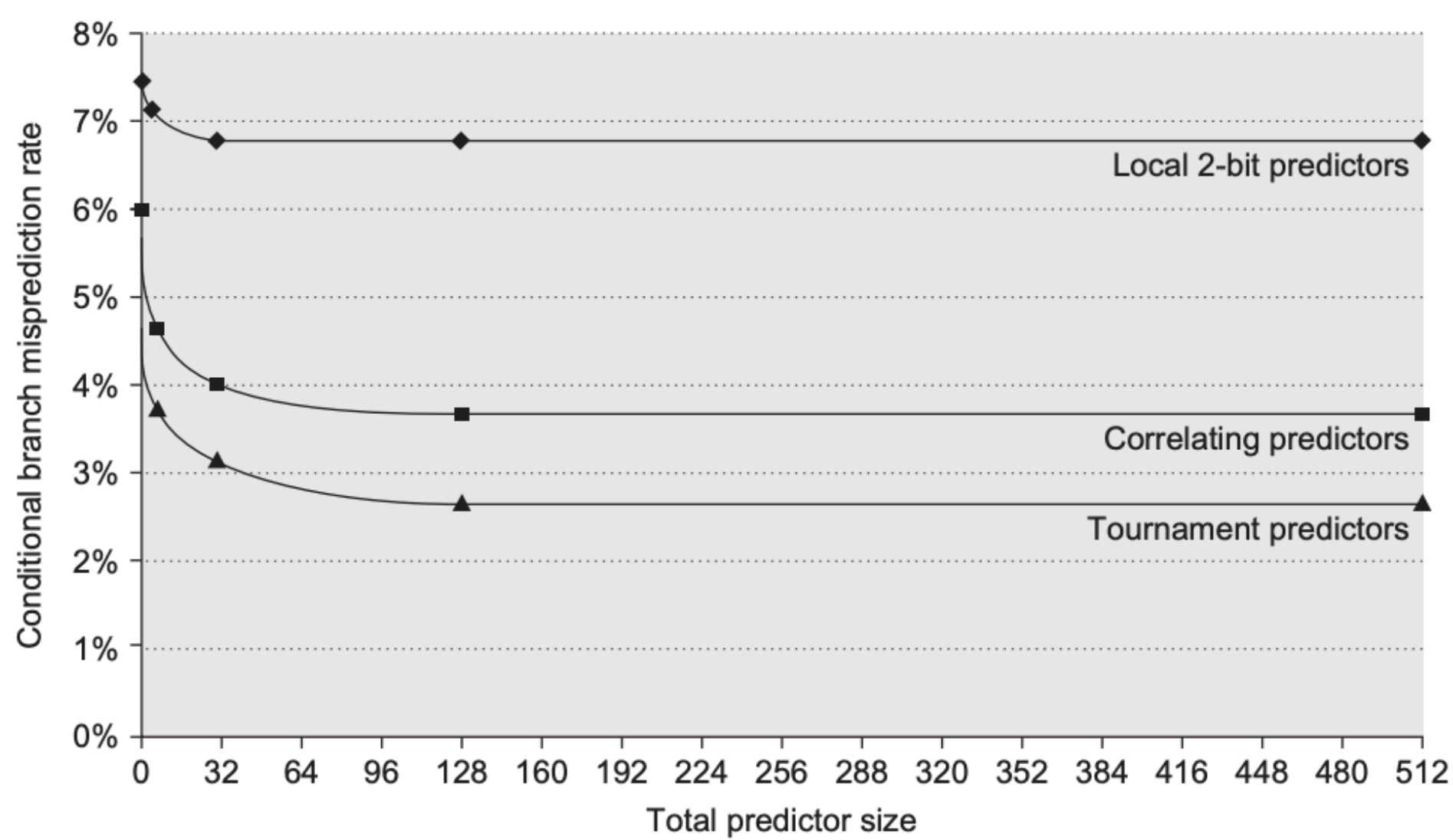
# Execution with Branch Prediction



# From Wednesday: Effectiveness of Branch Prediction



**Figure 3.3 Comparison of 2-bit predictors.** A noncorrelating predictor for 4096 bits is first, followed by a noncorrelating 2-bit predictor with unlimited entries and a 2-bit predictor with 2 bits of global history and a total of 1024 entries. Although these data are for an older version of SPEC, data for more recent SPEC benchmarks would show similar differences in accuracy.



**Figure 3.6 The misprediction rate for three different predictors on SPEC89 versus the size of the predictor in kilobits.** The predictors are a local 2-bit predictor, a correlating predictor that is optimally structured in its use of global and local information at each point in the graph, and a tournament predictor. Although these data are for an older version of SPEC, data for more recent SPEC benchmarks show similar behavior, perhaps converging to the asymptotic limit at slightly larger predictor sizes.

Image Credit: CA: AQA  
(course textbook)



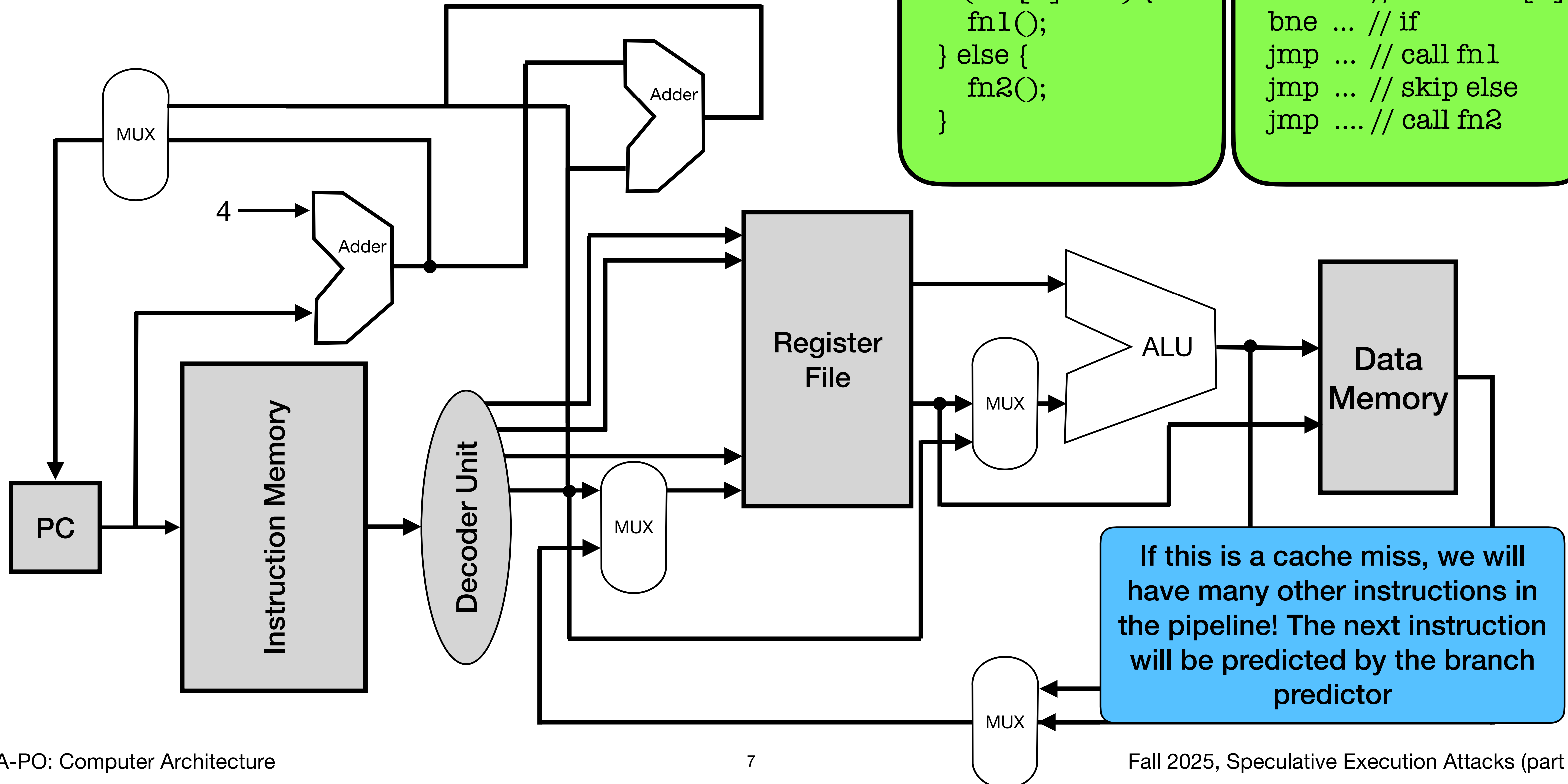
# Chat with your neighbor(s)!

Think about the instructions executed *before* we detect the true outcome of the branch. What are these instructions? What are the implications of having them in the pipeline?

# Slow to Resolve Branches

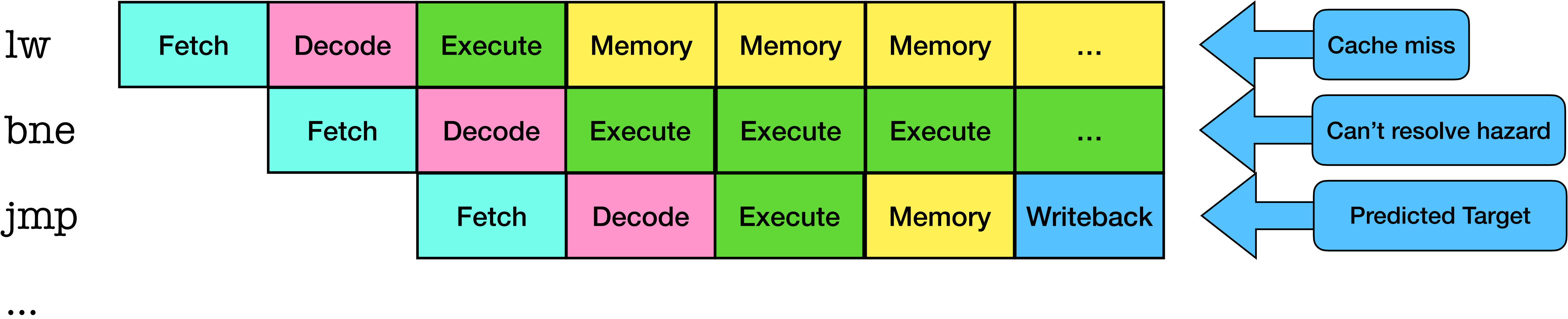
```
if (arr[0] == 1) {  
    fn1();  
} else {  
    fn2();  
}
```

```
lw    ... // fetch arr[0]  
bne   ... // if  
jmp   ... // call fn1  
jmp   ... // skip else  
jmp   ... // call fn2
```



If this is a cache miss, we will have many other instructions in the pipeline! The next instruction will be predicted by the branch predictor

# Slow to Resolve Branches



Problem! We don't want to writeback the state of an instruction if we don't know that this instruction was the right one to execute!

Speculative processors have a temporary buffer where data is written back to called the "Reorder Buffer" (ROB), which decouples instruction *writeback* from *commit*



# The Pitfalls of a Speculative Processor

- Branch predictors work from dynamic application behavior, as a result an adversary can *train* a branch predictor to guess the wrong side of an if-statement
- When executing a mispredicted branch in a transient state (e.g., when the processor is unsure which side of the branch is the true outcome), there are still implications on the *memory system*
- Often times, conditional branches are used for safety checks!

# Chat with your neighbor(s)!

The below code is vulnerable to a Spectre variant 1 attack. Think about what is happening in the processor to execute this program. What features might an adversary exploit and what information can they leak?

Basically a buffer  
overflow vulnerability!

```
void fn(x) {  
    if (x < size) {  
        y = array[x];  
    } else ...  
}
```

# The Premise of a Spectre Attack

- Adversary Goal: use the underlying microarchitectural state as a *covert channel* to get unintended code to leak secret information to an adversary
- The adversary will look for *gadgets* in the program state that will allow the processor to execute unintended instructions in a transient state
- Even if these instructions are not *committed*, accessed data will appear in the shared memory system
- For now, we should still be safe because the side channel attacks in the memory system only allow us to see the *addresses* in the memory system rather than the data contents itself... so no worries?



# Takeaways

- When executing our programs, even code that we do not intend to execute may still be in the microarchitectural state
- Adversaries can exploit this behavior to expose secrets via the memory system or the ROB
- When designing for efficiency (e.g., speculation), security often falls by the wayside
- We cannot leak very much yet unless there is something meaningful to learn about the address being in the cache state from a speculative execution...

<https://forms.cloud.microsoft/r/WFV2mjE3ih>



Exit Ticket