Security Compromise *via* speculation, caches & page tables

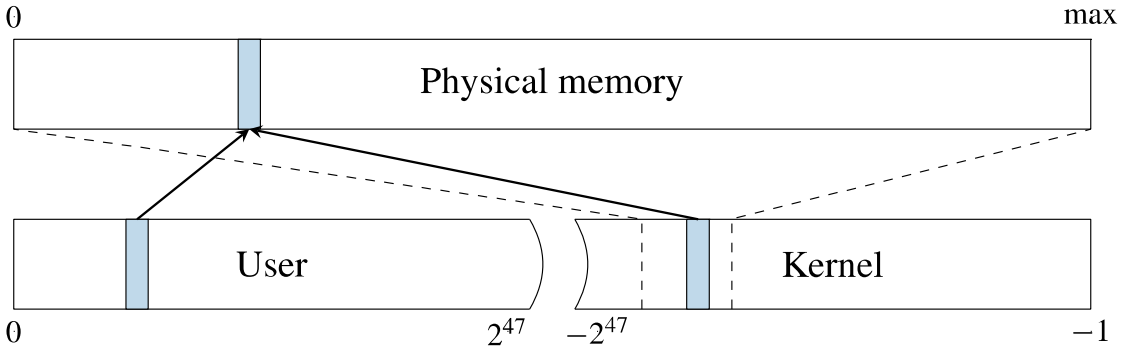
These slides adapted from materials provided by the textbook authors.

Meltdown & Spectre

- **Meltdown lets (unpriv'd) attacker read all of memory :**
  - Includes *all* of physical memory at ~500KBytes/s – ~5 hours to dump 8GB of memory
  - Possible to target specific data structures, passwords, etc
- **Spectre lets attacker read any memory in same process**
  - For example, JavaScript code in your browser can read any memory in the same process
- **Both attacks use same basic tools**
  - *Shared address space* – either shared O/S memory or same process
  - *Timers* – the ability to measure cache access times
  - And, *speculation* to get items into the cache

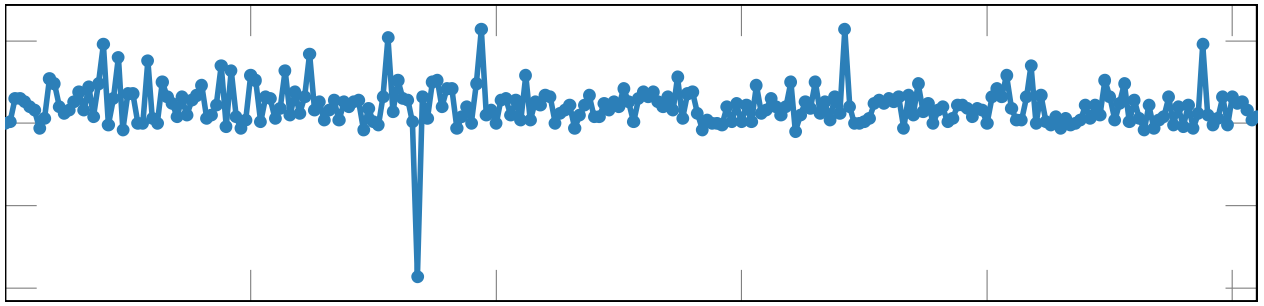
Shared Address Space

- **Software like Java, JavaScript, *etc* restricts the data that code can access by *sandboxing*, but that data is in a shared address space**
  - Don't let code read arbitrary memory (e.g. JavaScript)
  - Basis of security model for browser & other "extensible" interfaces
- **Most O/S's have physical memory in process address space, but inaccessible due to page table access control**
  - Reduces need to change page tables on entering O/S
  - Linux (used to) have all of physical memory accessible

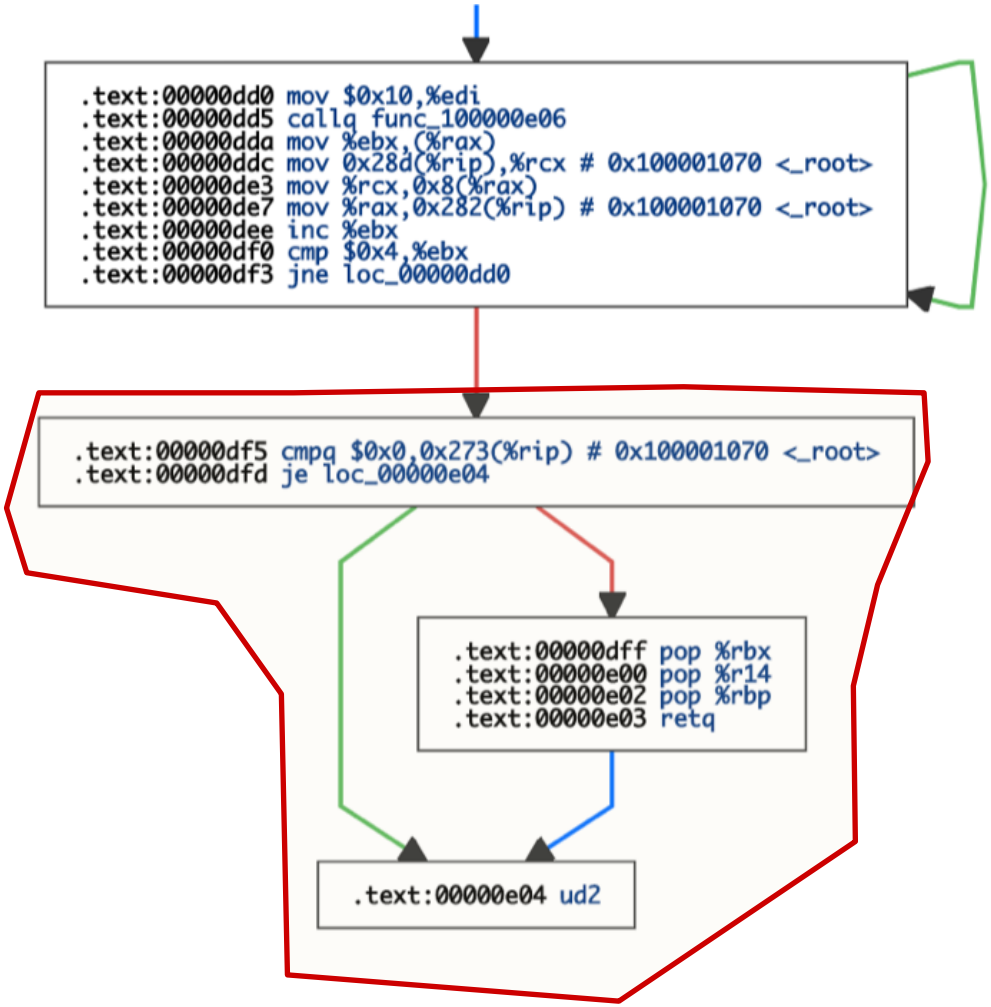


Detecting if data is in cache

- **We can tell if specific data is in the L1/L2 cache based on the time it takes to access that memory location**
  - Can time access using **rdtsc** or other methods
  - Will serve as a signal that we can exploit
- **It's also possible to "evict" items from cache**
  - **clflush** instruction
  - **Fill cache with other data using loops / memory loads**



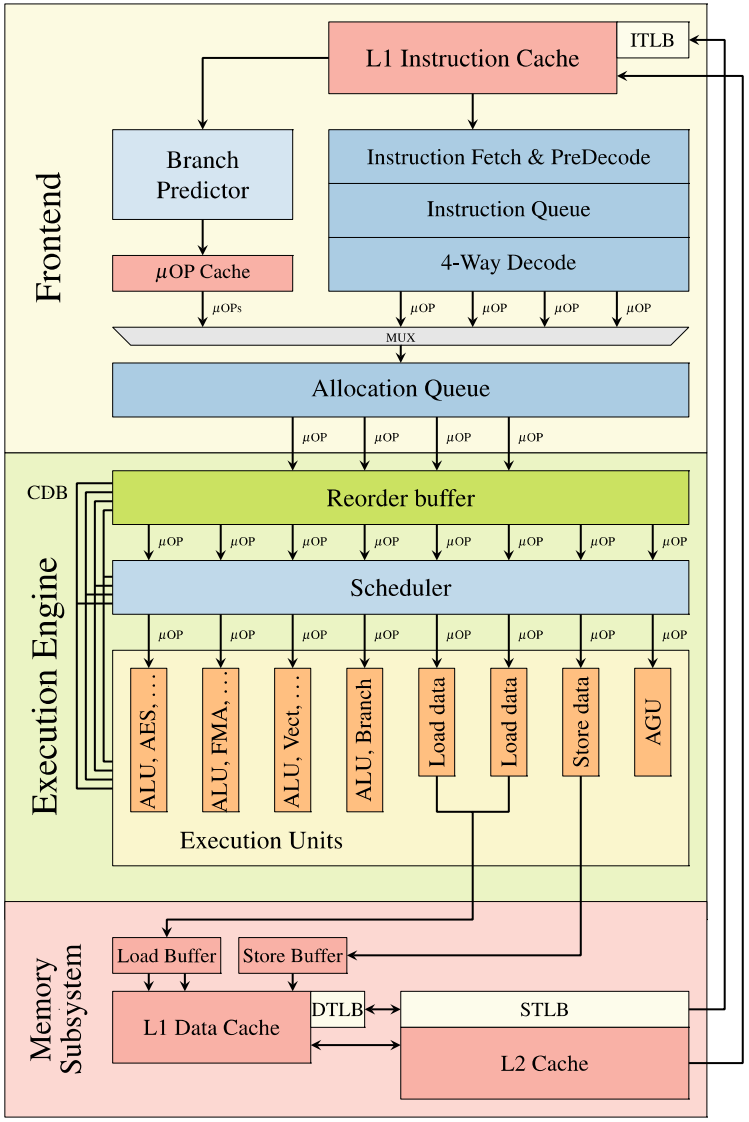
Speculative Execution



- Modern processors fetch & execute instructions ahead of the current IP
- Instructions execute, but hold results & exceptions in temporary registers until they "commit"
- But, memory loads need to be fetched and are cached

Memory Accesses During Speculation

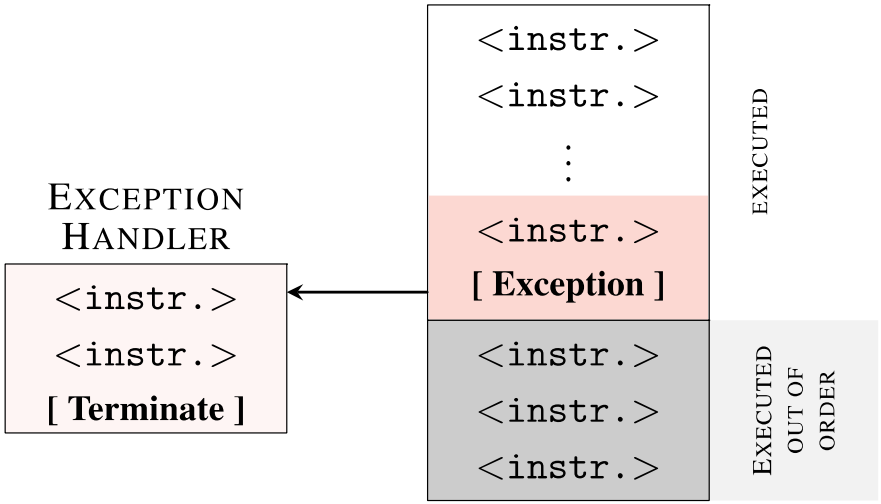
- During speculation the "frontend" fetches instructions
- Issues them to the execution engine, which executes any instructions that have data ready
- Also performs memory loads that are cached
- The output from "mis-speculated" instructions is discarded



Controlling Speculation

- We can cajole processor to speculate by training branch predictors (spectre)
- Or, can use exceptions to change execution flow (meltdown)

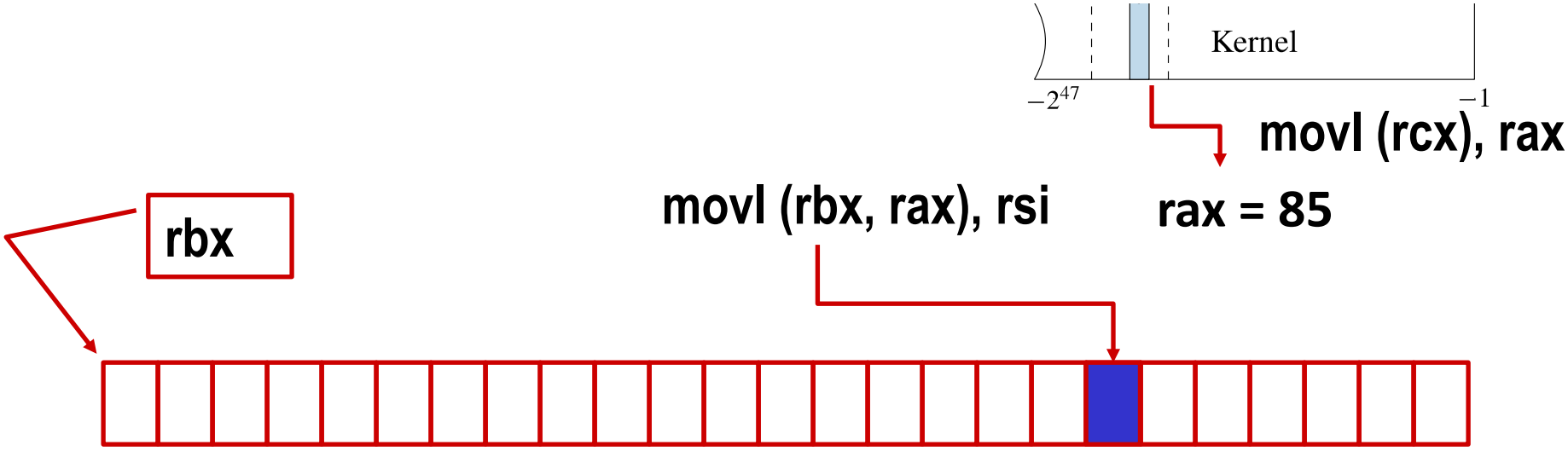
- Several methods, some are cheap (tsx)

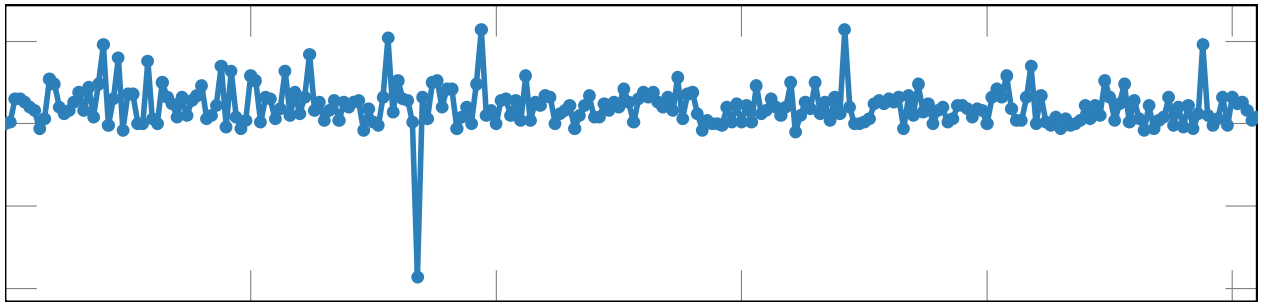


Now, the attack

- **Set up a buffer in user space, make certain it's not in cache**
  - lea $buffer,rbx
- **In code that is speculatively executed, fetch data value from system memory.**
  - movl $system_memory_address, rcx
  - movl (rcx), rax
- **Now use stolen data (rax) as an index into your big buffer**
  - movl (rbx, rax), rsi
- **When speculative instructions are killed, you no longer have value in 'rax', but you've fetched a memory address related to 'rax' and it's in cache**

The Attack – play by play

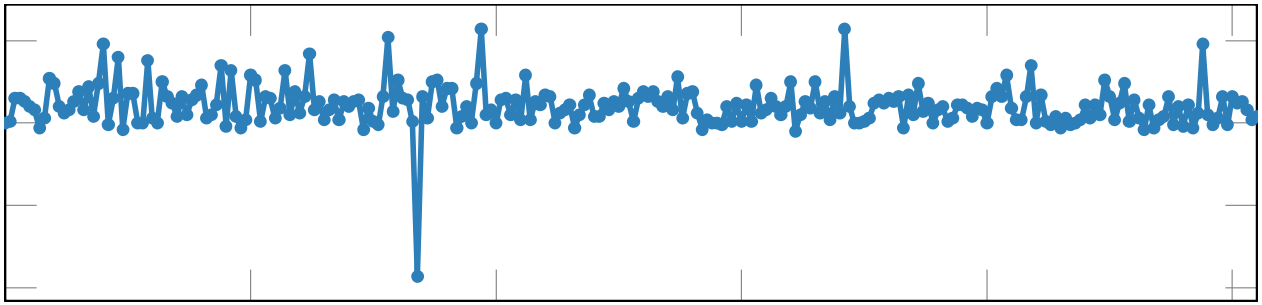




Attack code in Meltdown

- Meltdown uses the following speculative attack code
- The value from the kernel is multiplied by 4096

```
1 ; rcx = kernel address
2 ; rbx = probe array
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```



Attack code in Meltdown

- **Meltdown uses the following speculative attack code**
- **The value from the kernel is multiplied by 4096**

```
1 ; rcx = kernel address
2 ; rbx = probe array
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```

- **There's a loop checking if the value from kernel is zero**
  - The memory load takes time, and the hardware "guesses" the value is zero sometimes (1-2% of the time)
  - Eventually, the speculative code ends and either one line of data is in the cache or none is, indicating value is zero

Counter Measures & Implications

- **Kernel / physical memory attack (Meltdown) largely limited to Intel CPU's**
  - Problem addressed with KAISER patch that avoids having kernel in process address space. Similar patch in Windows.

- **Within-process attack (Spectre) works on Intel, AMD & ARM processors**
  - Firefox / Chrome disabled precise timers, but not sufficient fix
  - Processor vendors are added fixes to hardware, but not complete
  - Existing systems are compromised and working JavaScript exploit has been demonstrated

Lessons to take away from this

- As computer scientists and programmers, you're responsible for building secure and reliable systems
- People's lives and property depend on these systems
- Understanding *system wide* design lets you better understand threats and the necessity of counter measures