

4.22 (Understand computer)

Chip → core → thread



Multithreading

- o Different stacks
local variables
- o PCs
- o registers

- o Multiple threads
- o same memory (globals, heap)
- o same time

blocking semantic
key board interrupt functions

Multiple ALUs in
core someone
stalls

progress (main is thread)

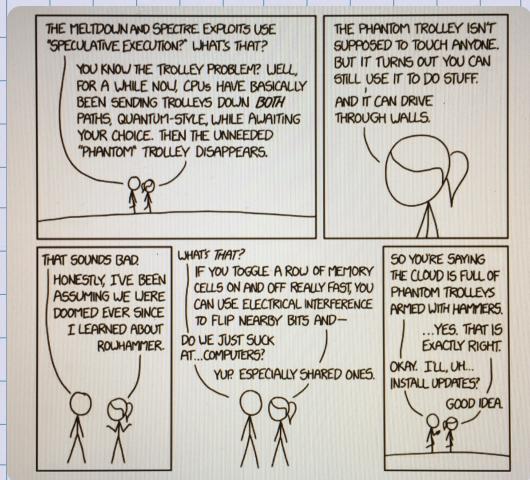
- thread
- thread
- thread

race condition
undefined sequence

LOCK - someone is writing back
don't interfere!

not operated instruction
non-unfill caches

Spectre & Meltdown



```
0x78          byte offset into cache line
0x1234_5600   index into cache table
0x123          tag

Our variable:
0x...4_56..
```

Pretty simple, right? Let's think about how to make this function very slow by making sure that every time the function is called, it hits a data-cache miss. Assume that:

- The cache lines are 256 bytes in size.
- There are exactly 4096 = 2^{12} cache lines in memory (only 1 line per set).
- You happen to know that the address of x is **0x1234_5678**.
- Your code (which messes things up) gets called every time, just before foo() is called.

Write a bit of C or MIPS which will put the cache in a state such that foo() will be slow. (NOTE: You don't care if your function is slow as well - so long as you reach your goal.)

```
addi    $s0, $zero, 0xffff45600
lw      $t0, 0($s0)
```

update cache
take the x's original
cache line possession

different cache lines

