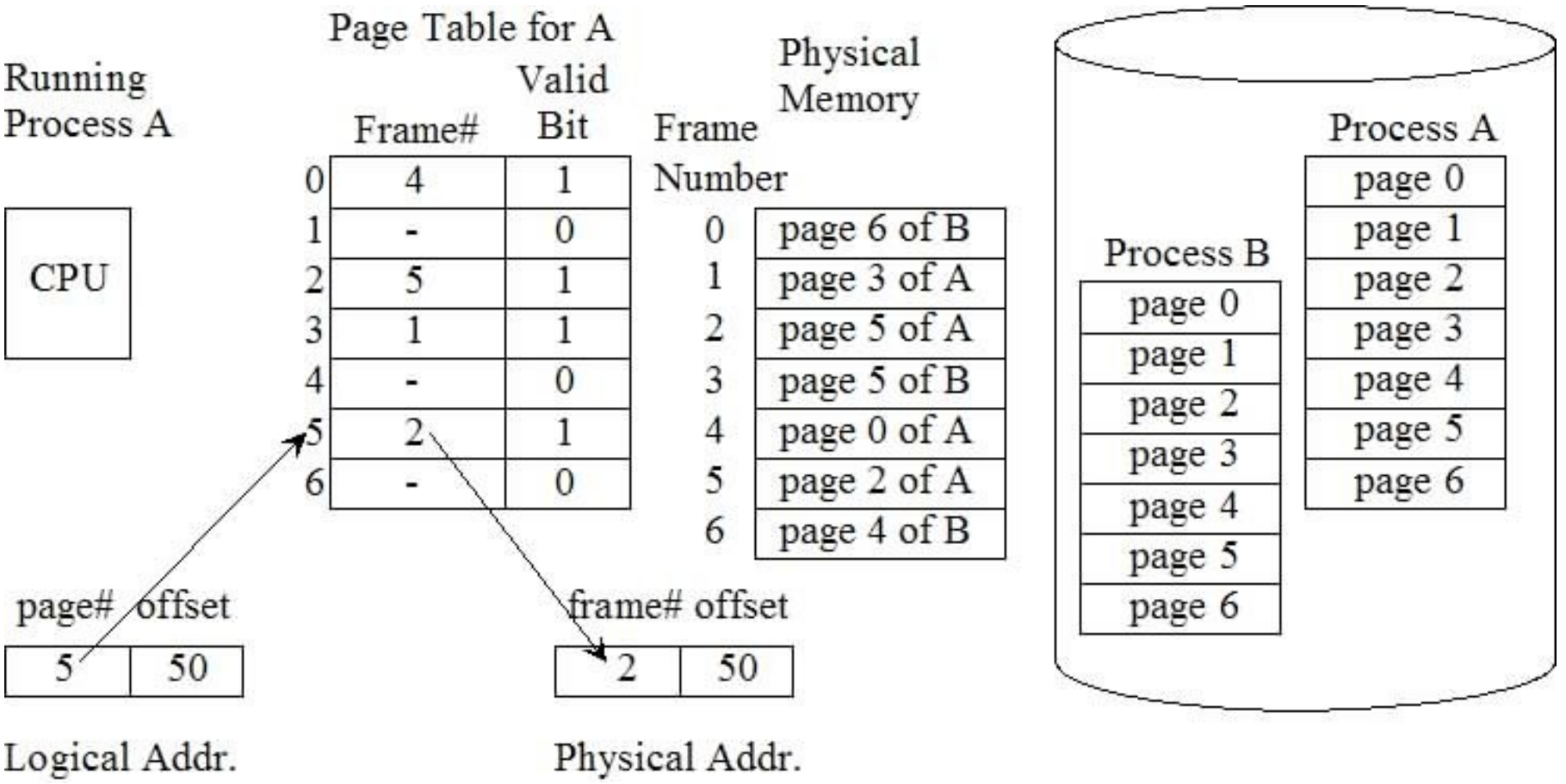
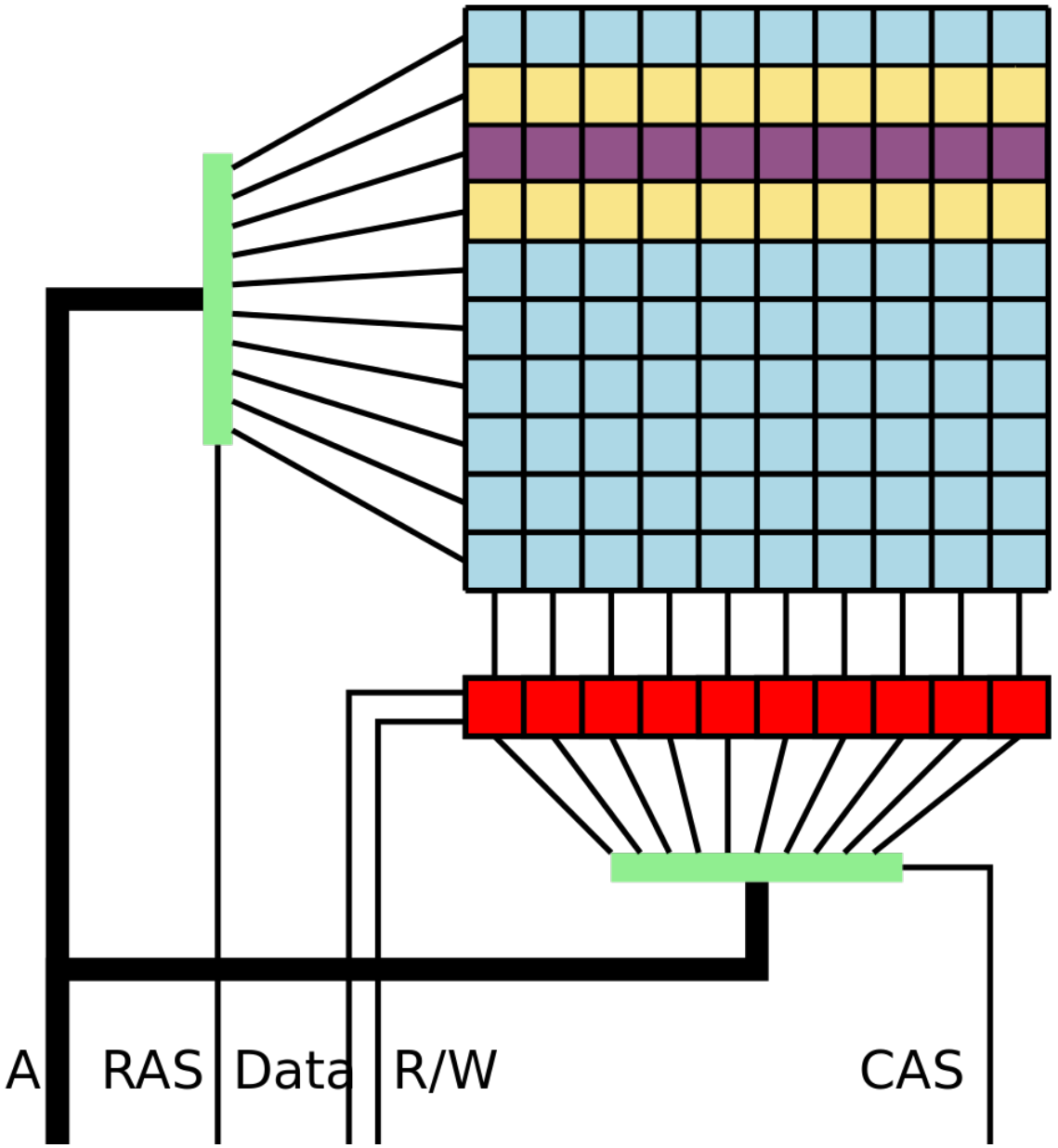
Rowhammer…

Food for thought

- Information is inherently physical
- Information only has meaning in that it is subject to interpretation
- Management information stored in-band with regular information
- Programming the weird machine



Plagiarized from http://www.cs.uni.edu/~fienup/cs142f05/lectures/lec20_OS_virtual_memory.HTM



Plagiarized from:
https://en.wikipedia.org/wiki/Row_hammer#/media/File:Row_hammer.svg

Step #1: Find aggressor and victim

- Allocate a large chunk of memory, like 1GB
- Aggressors X and Y must be different rows in the same bank
  - DRAM row is typically >4K and <2MB
  - Rows in a bank activated in lockstep
- Pick X and Y as random virtual addresses
  - Check if hammering X and Y flips a bit in Z
  - If you find that Z (have to check the whole block), that's your victim
- Hope that you can flip, e.g., the 12th bit in a 64-bit word rather than, e.g., the 51st
- munmap() all but these three pages (two aggressors, one victim)

Step #2: Randomize physical memory

- Why? So a small change in where a PTE points will not go from one data page to another.
- Allocate a huge chunk of memory with mmap() with MAP_POPULATE
- Throughout the exploit, release a random 4KB at a time with madvise + MADV_DONTNEED

Step #3: Spray physical memory with page tables

- Keep mmap()ing a file with markers in it, 2MB aligned
  - Why 2MB? One page table has 512 entries times 4K = 2MB
  - Try to have more page tables in memory than data
    - When victim is released it’s likely to be a page table
    - When bit is flipped new value is likely to point to a page table

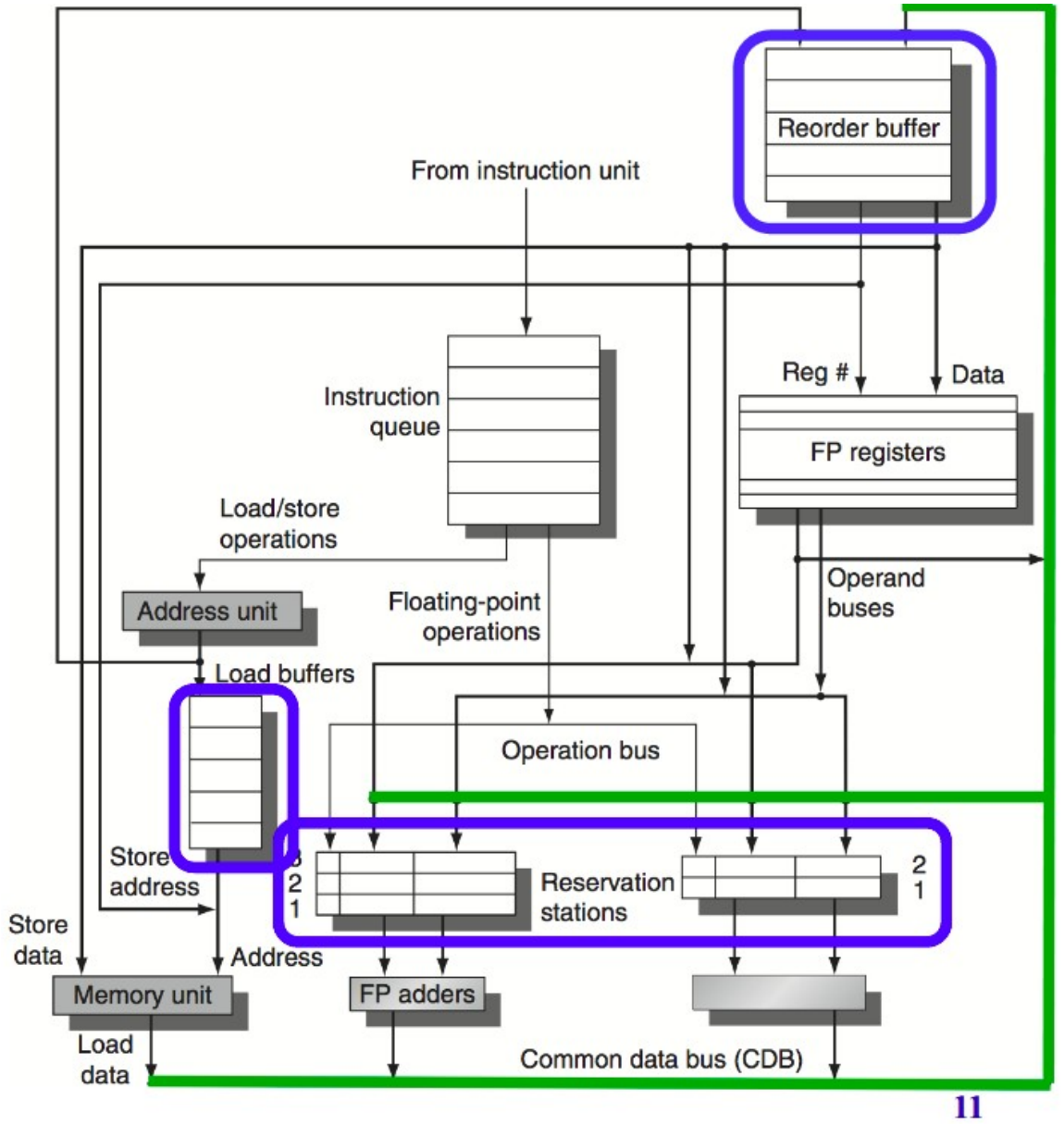
Step #4: Hammer time

- Check if bit flip changed a mapping in the page table to point to another page table
  - Only have to check the Nth page within each 2MB chunk
- If it’s not pointing to the file, then it’s likely pointing to another page table. Which one?
  - Can change it arbitrarily, then scan our virtual address space to fine another page that now doesn’t point to the file

Step #5: Exploit

- mmap() a setuid binary, like ping
  - Kernel won’t set write bit in your PTE for ping’s code section
  - Modify your writable page table to give yourself write permissions to the physical page where ping’s code section gets cached
  - Execute it as root

MELTDOWN...



Plagiarized from:
https://passlab.github.io/CSCE513/notes/lecture18_ILP_SuperscalarAdvancedARMIntel.pdf

Overly simplified MELTDOWN

```
int a[256 * cachelinesize] // cache aligned
char *p = &SomethingICantReadInKernel
int x = a[*p * cachelinesize]
```

- Side channel: whatever gets cached speculatively reveals *p

What does this mean?

- Supervisor bit is useless, because microarchitectural state can be visibly changed based on speculative execution that ignores the supervisor bit
- Can no longer put the kernel at the top of the virtual address space of every process

- https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html
- https://www.usenix.org/conference/usenixsecurity18/presentation/lipp
-