

# Virtual Memory III

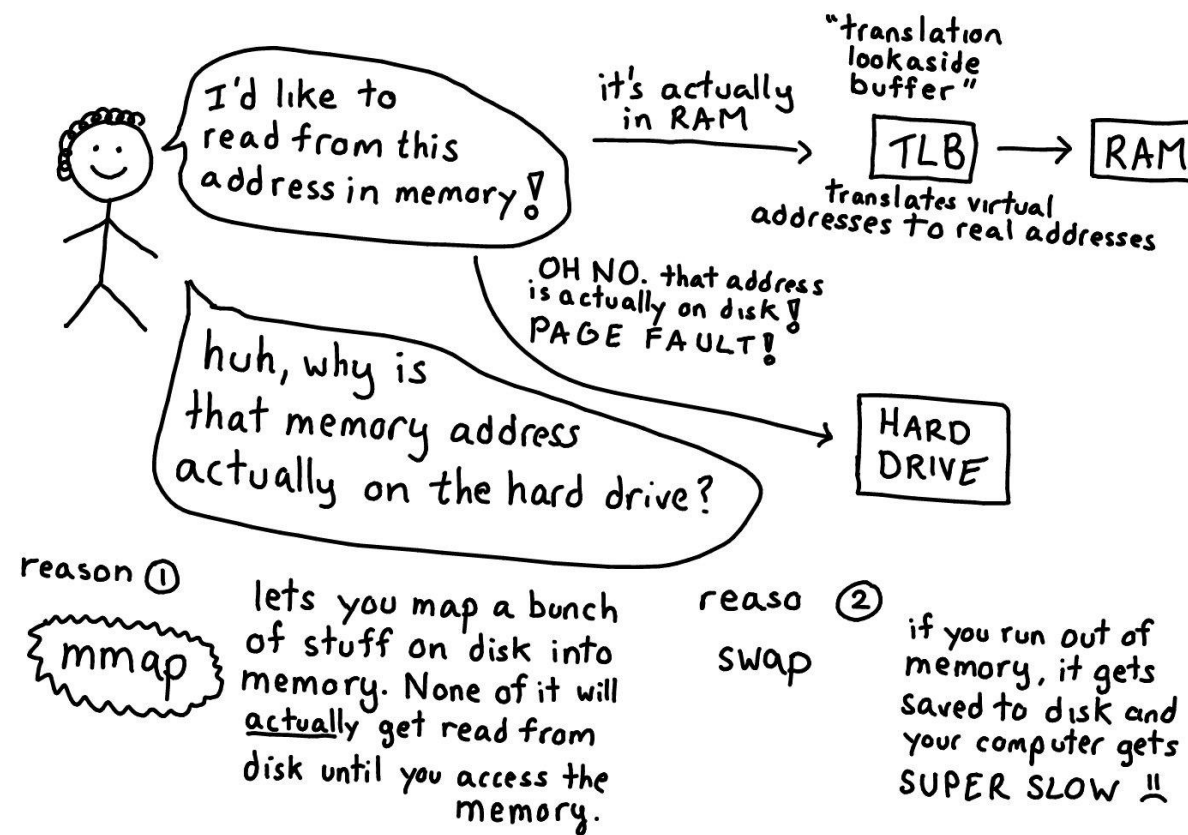
CSE 351 Autumn 2023

**Instructor:**  
Justin Hsia

**Teaching Assistants:**

- |               |                 |
|---------------|-----------------|
| Afifah Kashif | Malak Zaki      |
| Bhavik Soni   | Naama Amiel     |
| Cassandra Lam | Nayha Auradkar  |
| Connie Chen   | Nikolas McNamee |
| David Dai     | Pedro Amarante  |
| Dawit Hailu   | Renee Ruan      |
| Ellis Haker   | Simran Bagaria  |
| Eyoel Gebre   | Will Robertson  |
| Joshua Tan    |                 |

## ≡ VIRTUAL MEMORY ≡



# Relevant Course Information

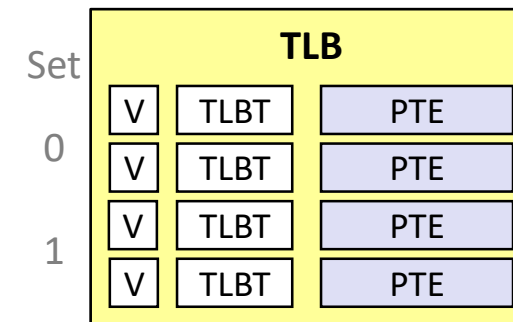
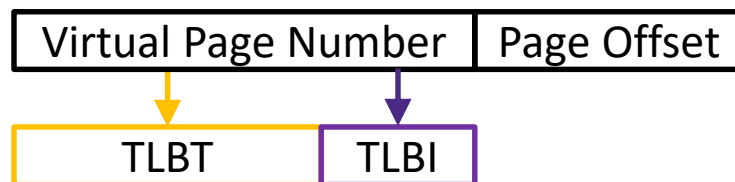
- ❖ HW24 due tonight; HW25 due Wednesday (12/6)
- ❖ Lab 5 due Thursday (12/7)
  - Recommended that you watch all of the Lab 5 helper videos
- ❖ No lessons in Week 11 – “normal” lectures
- ❖ **Final Exam: 12/11-13**
  - Similar to midterm; Gilligan’s Island Rule in effect
  - Final review section on 12/7
  - Review Session: Fri, 12/8, evening
    - More info to be released on Ed Discussion

A detailed, colorful micrograph of a microchip die, showing a complex grid of circuitry and various colored regions (purple, blue, yellow, green, red) representing different functional blocks.

# Virtual Memory III

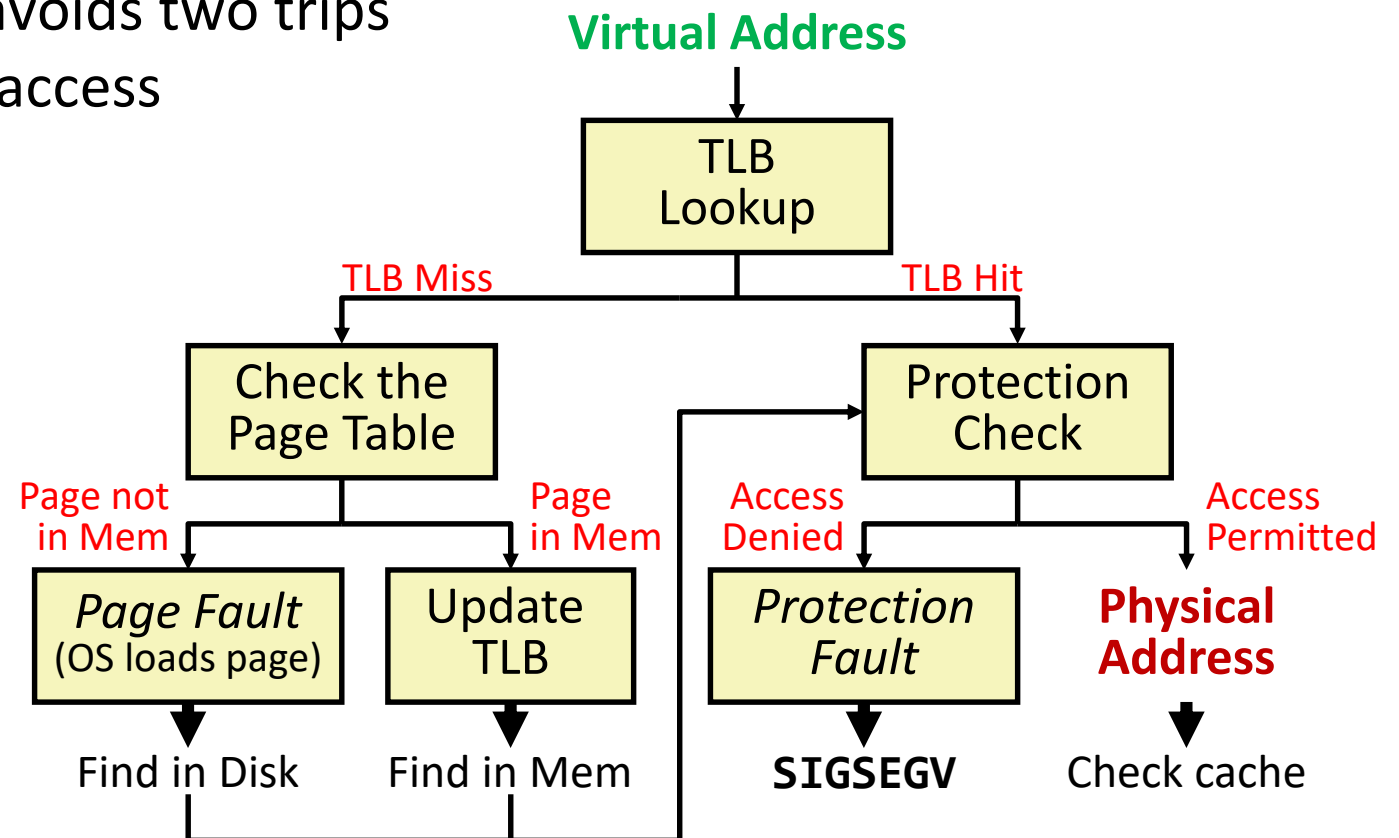
# Lesson Summary

- ❖ Introduced the *translation lookaside buffer* (TLB) as a cache for page table entries (PTEs)
  - Try to avoid accessing the page table in memory
  - Split VPN into **TLB Tag** and **TLB Index** based on # of sets in TLB
  - Management bits include valid (like \$, not PT) and TLB tag



# Address Translation

- ❖ VM is complicated, but also elegant and effective
  - Level of indirection to provide isolated memory & caching
  - TLB as a cache of page tables avoids two trips to memory for every memory access



# Fetching Data on a Memory Read

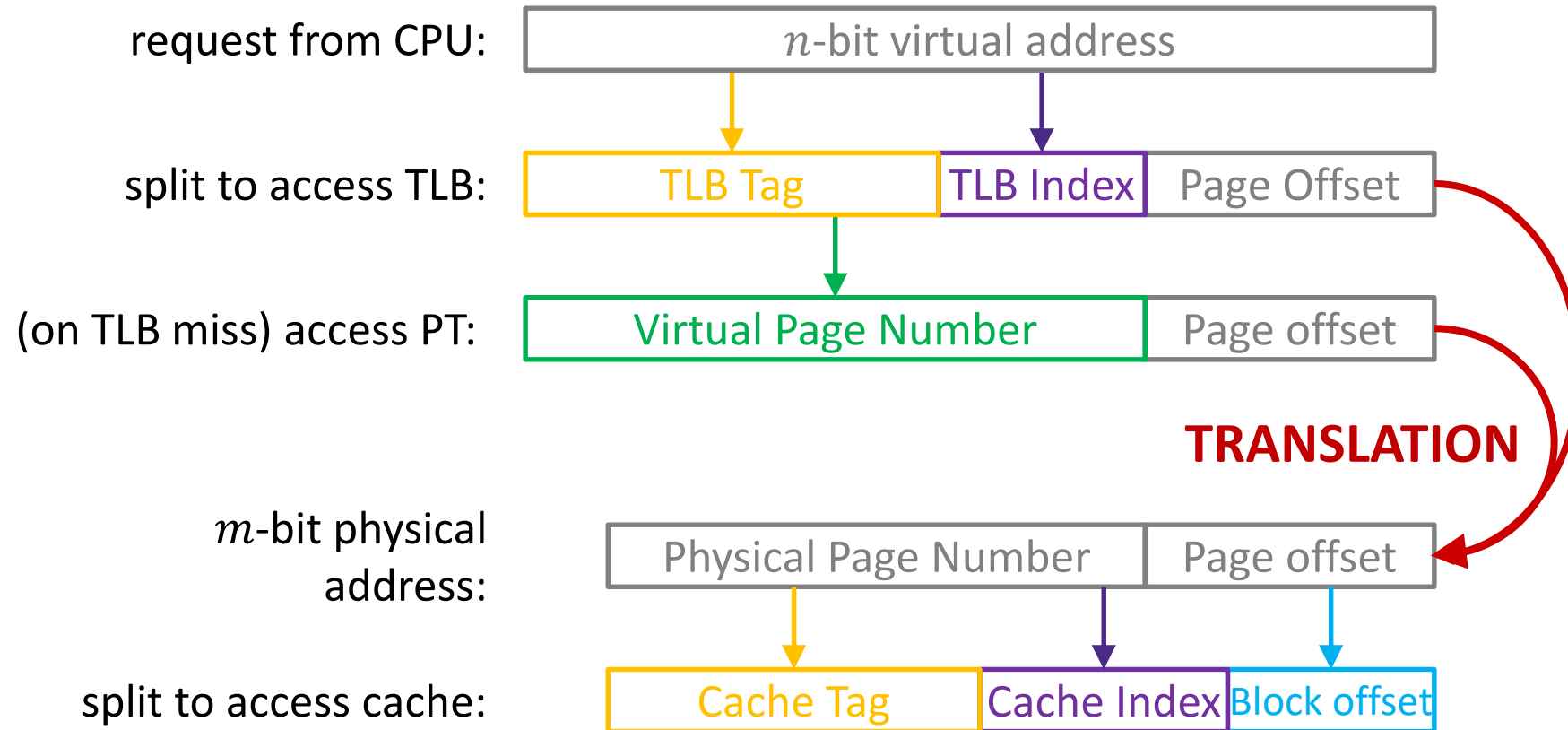
## 1) Address Translation (check TLB)

- Input: VPN, Output: PPN
- *TLB Hit*: Fetch translation, return PPN
- *TLB Miss*: Check page table (in memory)
  - *Page Table Hit*: Load page table entry into TLB
  - *Page Fault*: Fetch page from disk to memory, update corresponding page table entry, then load entry into TLB

## 2) Fetch Data (check cache)

- Input: physical address, Output: data
- *Cache Hit*: Return data value to processor
- *Cache Miss*: Fetch data value from memory, store it in cache, return it to processor

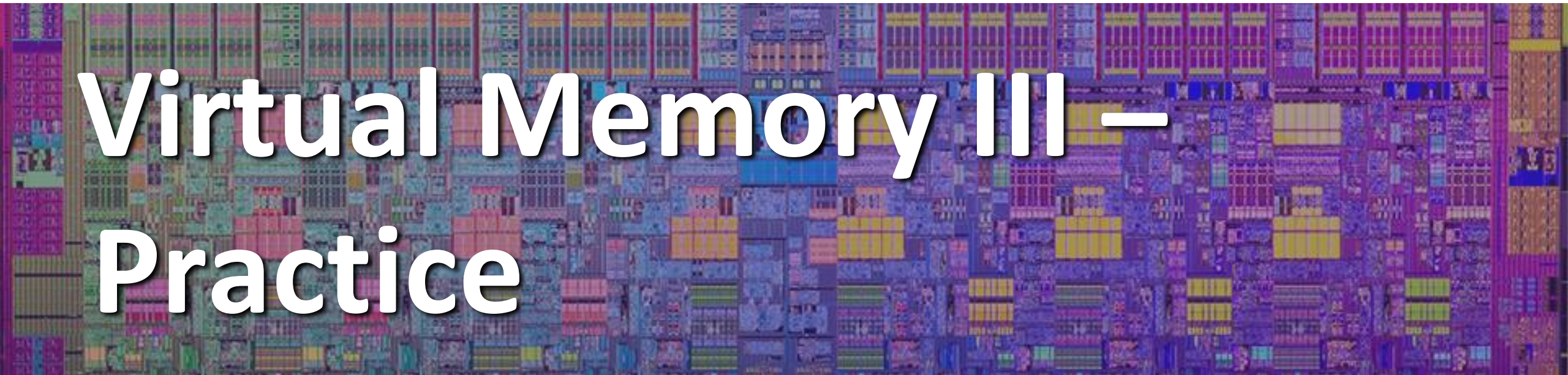
# Address Manipulation



# Lesson Q&A

- ❖ Terminology:
  - Translation Lookaside Buffer (TLB): TLB Hit, TLB Miss
  
- ❖ Learning Objectives:
  - Determine virtual memory parameters related to addresses, page tables, and TLBs.
  - Perform address translations (virtual address → physical address).
  - Describe the relationships between virtual memory parameters and policies.
  
- ❖ What lingering questions do you have from the lesson?



A detailed, colorful microchip die image serves as the background for the title. The chip is densely packed with various colored regions (purple, blue, yellow, green, red) representing different functional blocks and interconnects.

# Virtual Memory III – Practice

# Concept Questions

- ❖ What do Page Tables map?
- ❖ Where are Page Tables located?
- ❖ How many Page Tables are there?
- ❖ True / False: Virtual Addresses that are contiguous will always be contiguous in physical memory
- ❖ TLB stands for \_\_\_\_\_ and stores \_\_\_\_\_

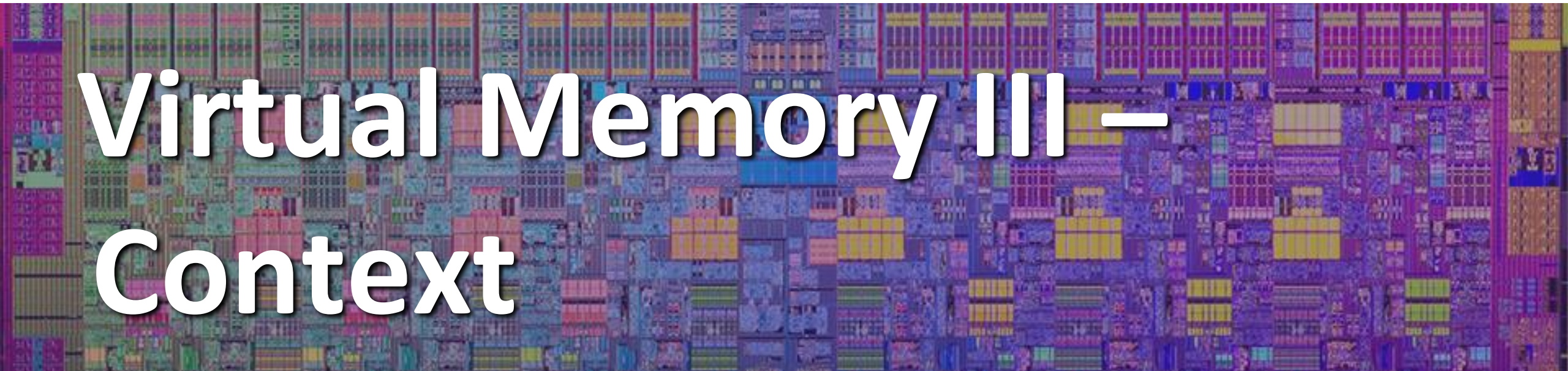
# VM Parameters Questions

- ❖ Our system has the following properties
  - 1 MiB of physical address space
  - 4 GiB of virtual address space
  - 32 KiB page size
  - 4-entry fully associative TLB with LRU replacement

a) Fill in the following blanks:

\_\_\_\_\_ Entries in a page table      \_\_\_\_\_ Minimum bit-width of PTBR

\_\_\_\_\_ TLBT bits      \_\_\_\_\_ Max # of valid entries in a page table

A detailed, colorful micrograph of a microchip die, showing a complex grid of circuitry and various colored regions (purple, blue, yellow, green, red) representing different functional blocks.

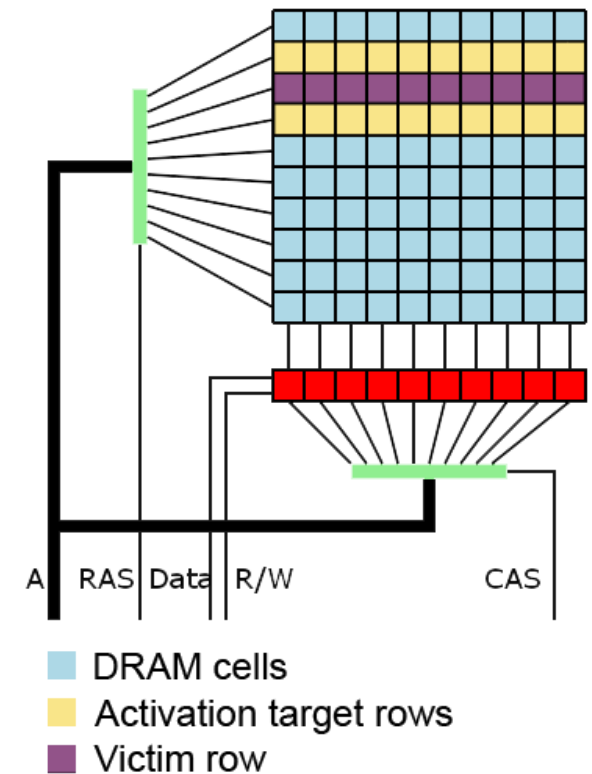
# Virtual Memory III – Context

# DRAMMER Security Attack

- ❖ **Relevance:** Uses your system's memory setup to gain elevated privileges
  - Ties together some of what we've learned about virtual memory and processes
- ❖ **Interest:** It's a software attack that uses *only hardware vulnerabilities* and requires *no user permissions*
- ❖ **Recent:** DRAMMER announced in October 2016; latest attack variant (Half Double) announced May 2021
  - Other recent variants of underlying vulnerability (row hammer) as recent as 2018

# Underlying Vulnerability: Row Hammer

- ❖ Dynamic RAM (DRAM) has gotten denser over time
  - DRAM cells physically closer and use smaller charges
  - More susceptible to “*disturbance errors*” (interference)
- ❖ DRAM capacitors need to be “refreshed” periodically (~64 ms)
  - Lose data when loss of power
  - Capacitors accessed in rows
- ❖ **Rapid accesses to one row can flip bits in an adjacent row!**
  - ~ 100K to 1M times



By Dsimic (modified), CC BY-SA 4.0,  
<https://commons.wikimedia.org/w/index.php?curid=38868341>

# Row Hammer Exploit

- ❖ Force constant memory access
  - Read then flush the cache
  - `clflush` – flush cache line
    - Invalidates cache line containing the specified address
    - Not available in all machines or environments
  - Want addresses X and Y to fall in activation target row(s)
    - Good to understand how *banks* of DRAM cells are laid out
- ❖ The row hammer effect was discovered in 2014
  - Only works on certain types of DRAM (2010 onwards)
  - These techniques target x86 machines

```
hammertime:  
    mov (X), %eax  
    mov (Y), %ebx  
    clflush (X)  
    clflush (Y)  
    jmp hammertime
```

# Consequences of Row Hammer

- ❖ Row hammering process can affect another process via memory
  - Circumvents virtual memory protection scheme
  - Memory needs to be in an adjacent row of DRAM
- ❖ Worse: privilege escalation
  - Page tables live in memory!
  - Hope to change PPN to access other parts of memory, or change permission bits
  - **Goal:** gain read/write access to a page containing a page table, hence granting process read/write access to *all of physical memory*



# Effectiveness?

- ❖ Doesn't seem so bad – random bit flip in a row of physical memory
  - Vulnerability affected by system setup and physical condition of memory cells
- ❖ **Improvements:**
  - Double-sided row hammering increases speed & chance
  - Do system identification first (*e.g.*, Lab 4)
    - Use timing to infer memory row layout & find “bad” rows
    - Allocate a huge chunk of memory and try many addresses, looking for a reliable/repeatable bit flip
  - Fill up memory with page tables first
    - fork extra processes; hope to elevate privileges in any page table

# What's DRAMMER?

- ❖ No one previously made a huge fuss
  - **Prevention:** error-correcting codes, target row refresh, higher DRAM refresh rates
  - Often relied on special memory management features
  - Often crashed system instead of gaining control
- ❖ Research group found a *deterministic* way to induce row hammer exploit in a non-x86 system (ARM)
  - Relies on predictable reuse patterns of standard physical memory allocators
  - Universiteit Amsterdam, Graz University of Technology, and University of California, Santa Barbara

# DRAMMER Demo Video

- ❖ <https://youtu.be/x6hL-obNhAw>
  - It's a shell, so not that sexy-looking, but still interesting
  - Apologies that the text is so small on the video



# How did we get here?

- ❖ Computing industry demands more and faster storage with lower power consumption
- ❖ Ability of user to circumvent the caching system
  - `clflush` is an unprivileged instruction in x86
  - Other commands exist that skip the cache
- ❖ Availability of virtual to physical address mapping
  - **Example:** `/proc/self/pagemap` on Linux (not human-readable)
- ❖ Google patch for Android (Nov. 8, 2016)
  - Patched the ION memory allocator

# More reading for those interested

- ❖ DRAMMER paper: <https://vvdveen.com/publications/drammer.pdf>
- ❖ Google Project Zero:  
<https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html>
- ❖ First rowhammer paper: <https://users.ece.cmu.edu/~yoonguk/papers/kim-isca14.pdf>
- ❖ Latest non-uniform, frequency-based exploit:  
<https://comsec.ethz.ch/research/dram/blacksmith/>
- ❖ Wikipedia: [https://en.wikipedia.org/wiki/Row\\_hammer](https://en.wikipedia.org/wiki/Row_hammer)

# Discussion Question

- ❖ Discuss the following question(s) in groups of 3-4 students
  - I will call on a few groups afterwards so please be prepared to share out
  - Be respectful of others' opinions and experiences
- ❖ We tend to think of software as existing in a realm abstracted away from hardware; however, this class has been about the interactions between hardware and software
  - Based on what we've learned in this class, what are some ways that your choice of hardware setup may affect your "day-to-day" as a software engineer?

# Group Work Time

- ❖ During this time, you are encouraged to work on the following:
  - 1) If desired, continue your discussion
  - 2) Work on the homework problems
  - 3) Work on the current lab
  
- ❖ Resources:
  - You can revisit the lesson material
  - Work together in groups and help each other out
  - Course staff will circle around to provide support