6.S081: Virtual Memory

Adam Belay

abelay@mit.edu

Logistics update

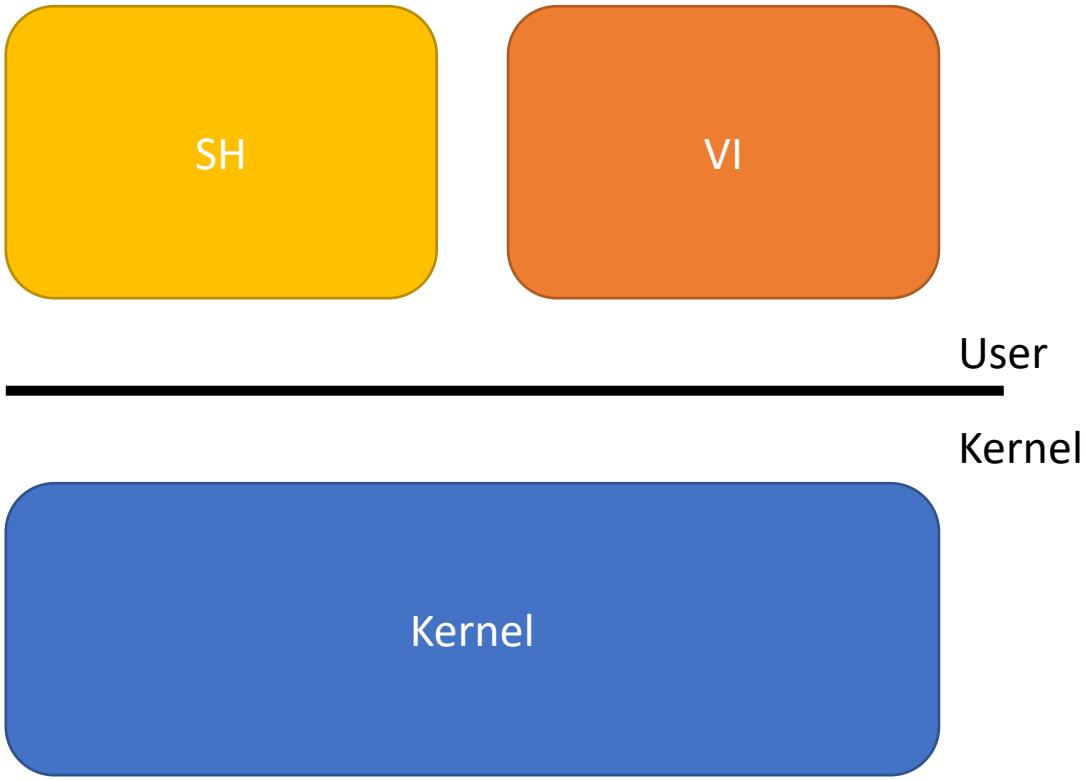
- Lab checkoff instructions posted on Piazza
  - You will receive an email if you have a checkoff for the first lab
  - Attend office hours (virtually or physically) for checkoff
- Syscall lab due this Thursday
- Pgtbl lab assigned this Wednesday

Outline

- Address spaces
- Risc-V paging hardware
- Case study: xv6 VM code

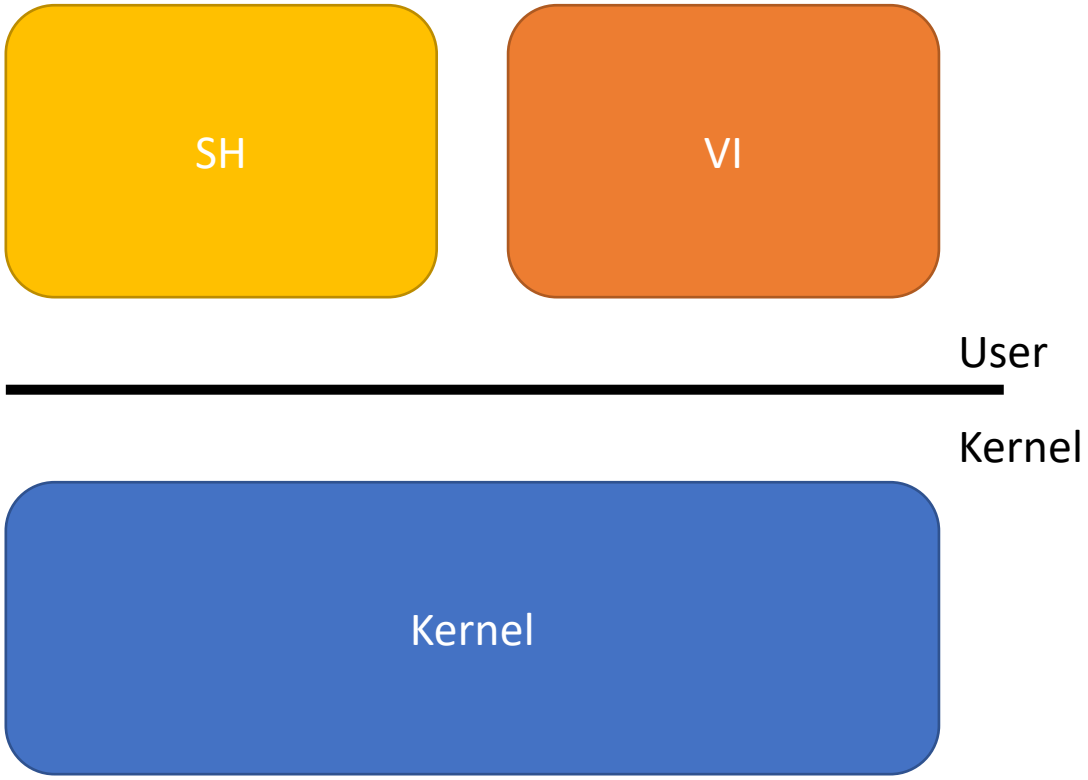
Today's problem

Protection View:



Today's problem

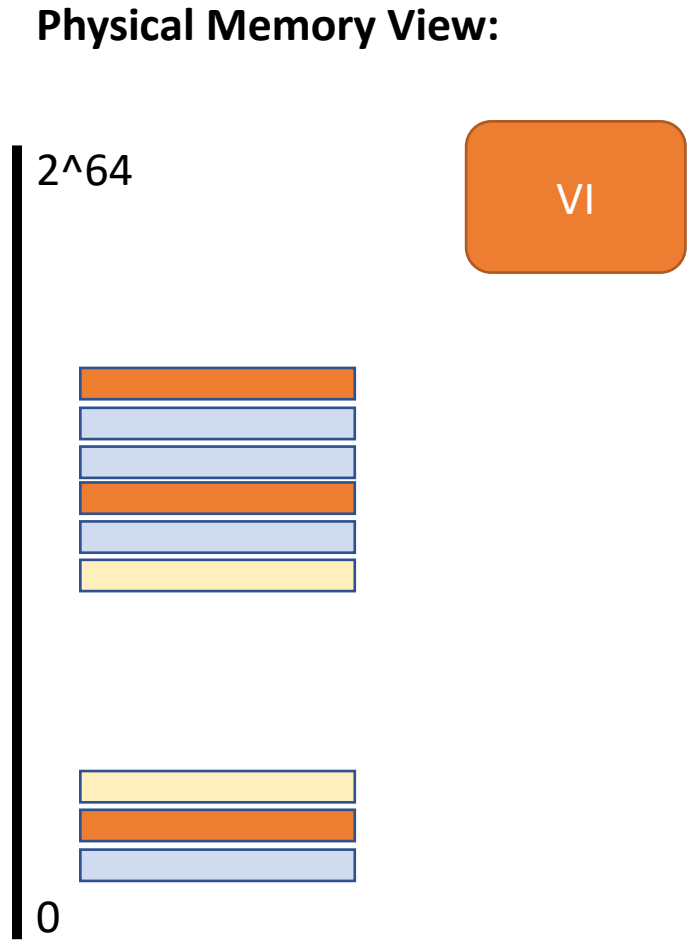
Protection View:



Physical Memory View:

Goal: Isolation

- Each process has its own memory
- Can read and write its own memory
- But cannot read or write the kernel's memory or another process' memory

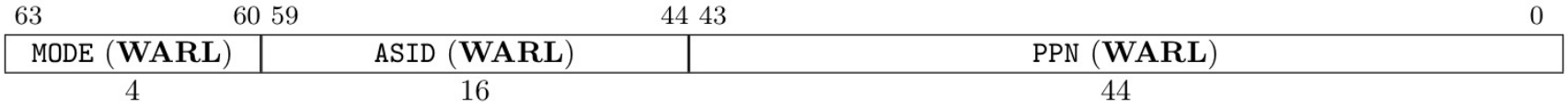


Solution: Introduce a level of indirection

- Plan: Software can only read and write to virtual memory
- Only kernel can program MMU
- MMU has a **page table** that maps virtual addresses to physical
- Some virtual addresses restricted to kernel-only

Virtual memory in Risc-V

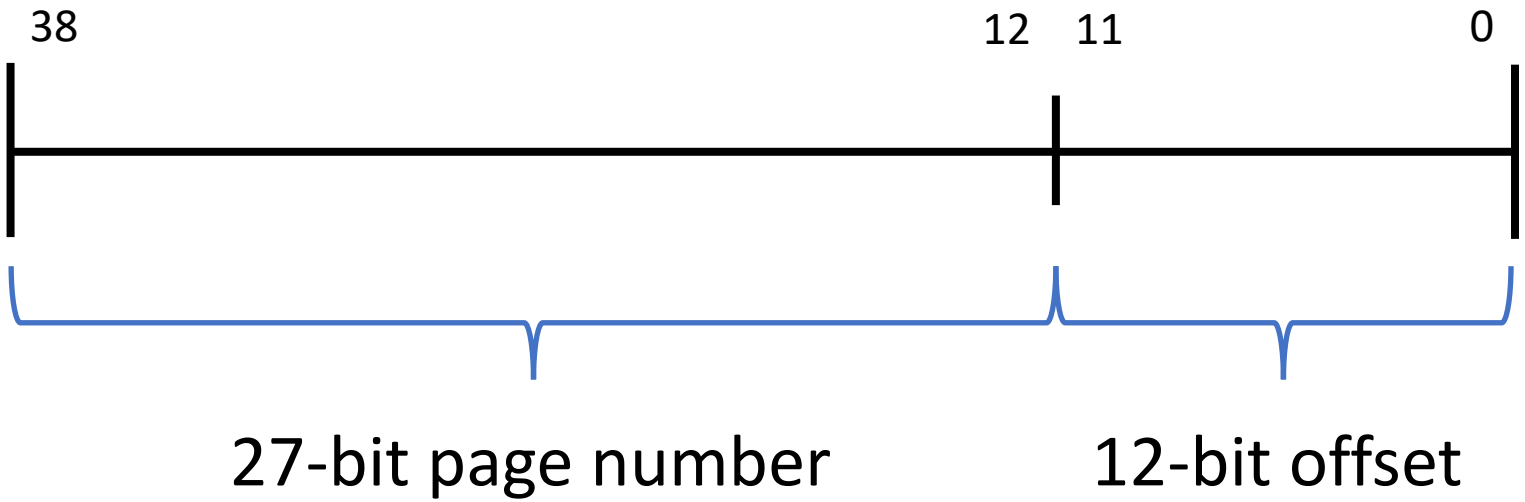
- Supports different addressing modes:
  - Sv32, Sv39, Sv48 -> number of virtual addr bits
  - This class will use Sv39 (3-level page table)
- **SFENCE.VMA** tells CPU to check page tbl updates
- **satp** register points to page root (set w/**CSRW**)



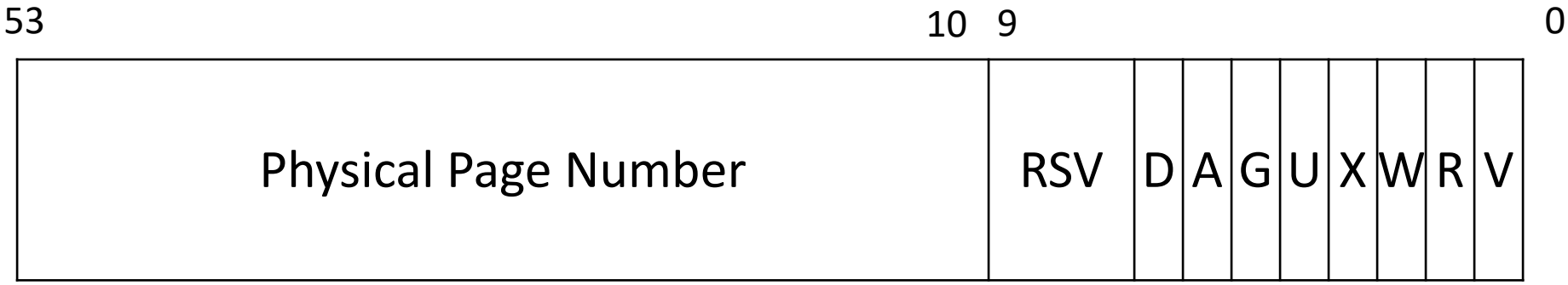
Virtual memory in Risc-V (Sv39)

Virtual addresses are divided into 4-KB “**pages**”

Virtual Address:



Page table entries (PTE)



Some important bits:

- **Physical page number**: Identifies 44-bit physical page location; MMU replaces virtual bits with these physical bits
- **U**: If set, userspace can access this virtual address
- **W**: If set, the CPU can write to this virtual address
- **V**: If set, an entry for this virtual address exists
- **RSV**: Ignored by MMU

Strawman: Store PTEs in an array

GET_PTE(va) = &ptes[va >> 12]

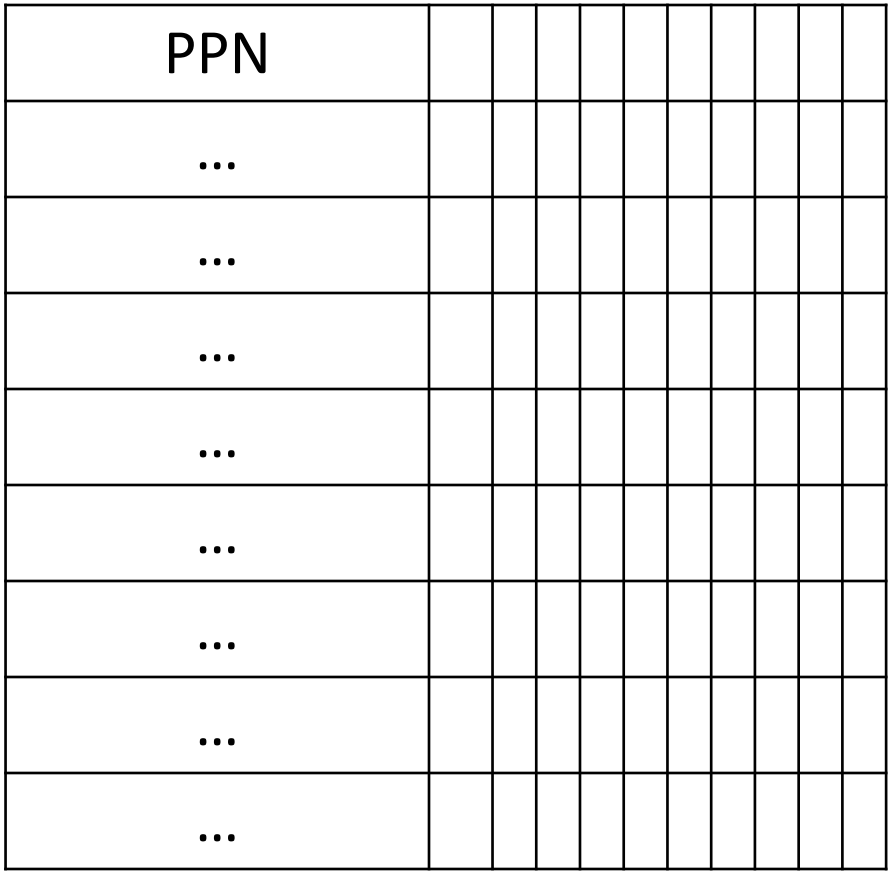
PPN										
…										
…										
…										
…										
…										
…										
…										
…										
…										

How large is the array?

…

Strawman: Store PTEs in an array

GET_PTE(va) = &ptes[va >> 12]



How large is the array?

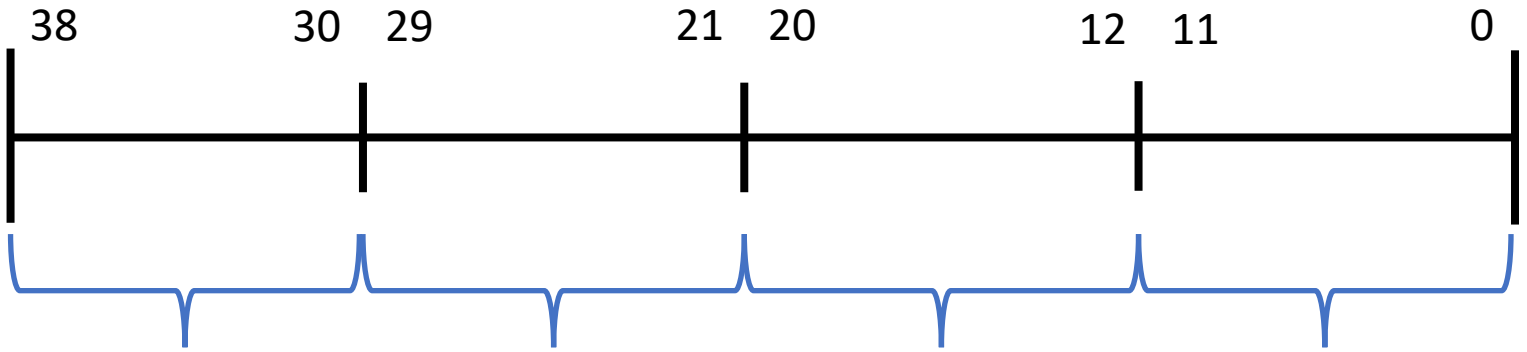
2^27 * 64 bits

2^27 * 8 bytes

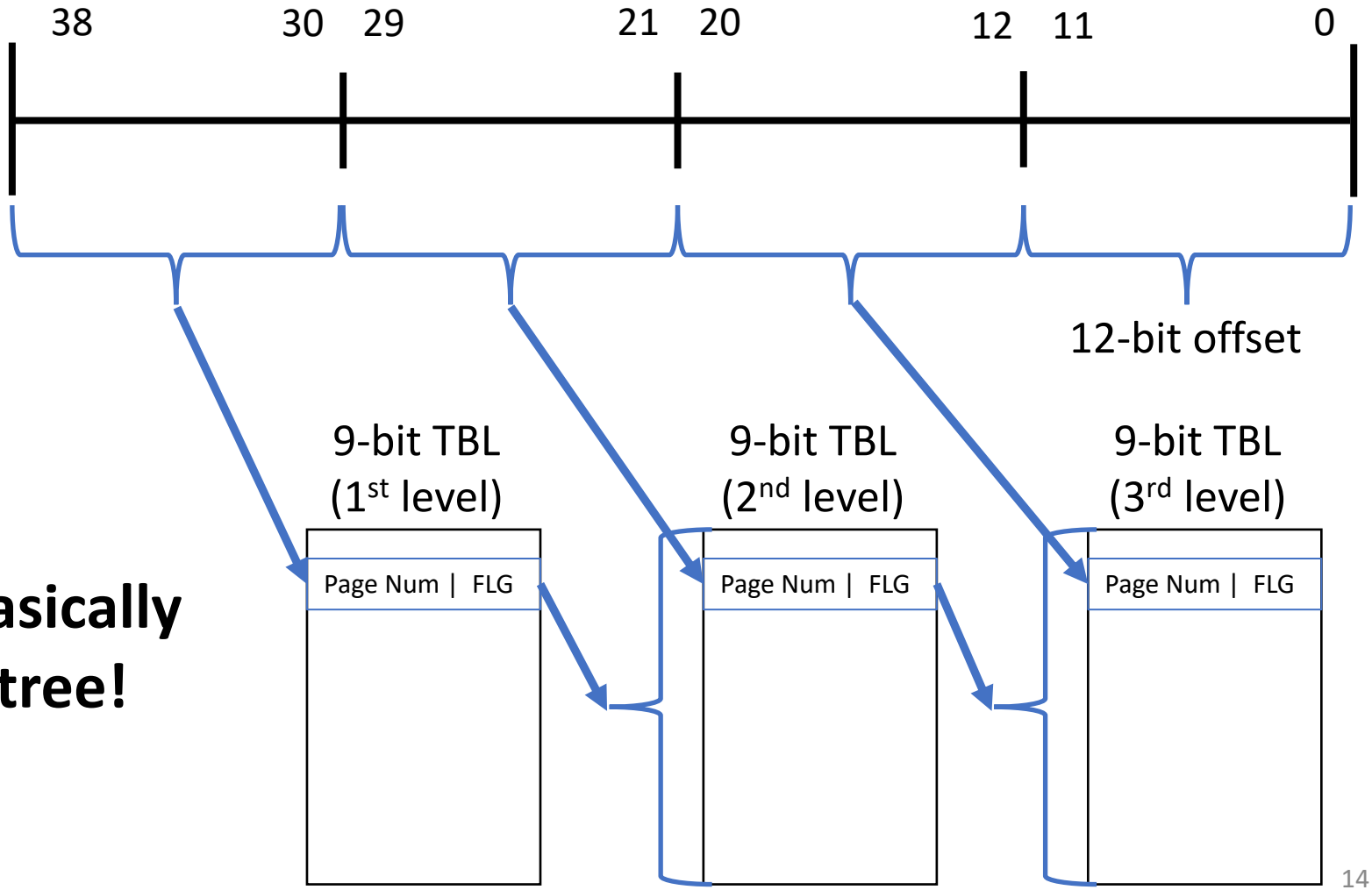
~ 1 Gigabyte!

...

Risc-V solution: Use three levels to save space

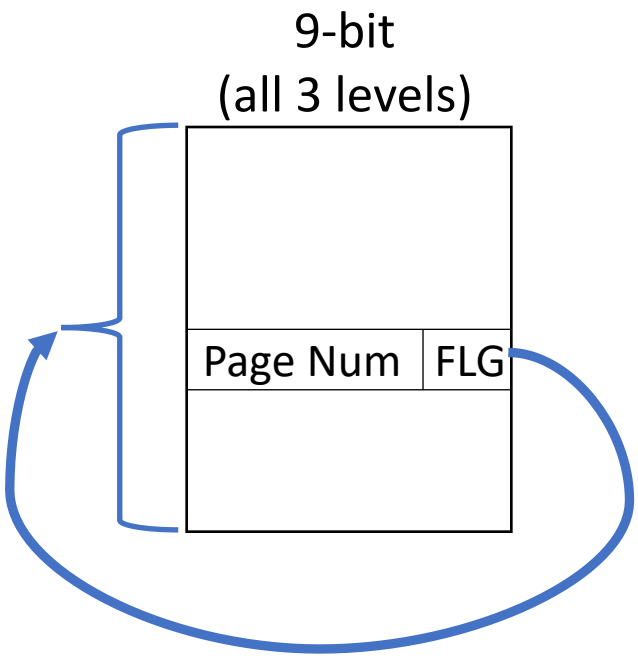


Risc-V solution: Use three levels to save space (512 entries each)

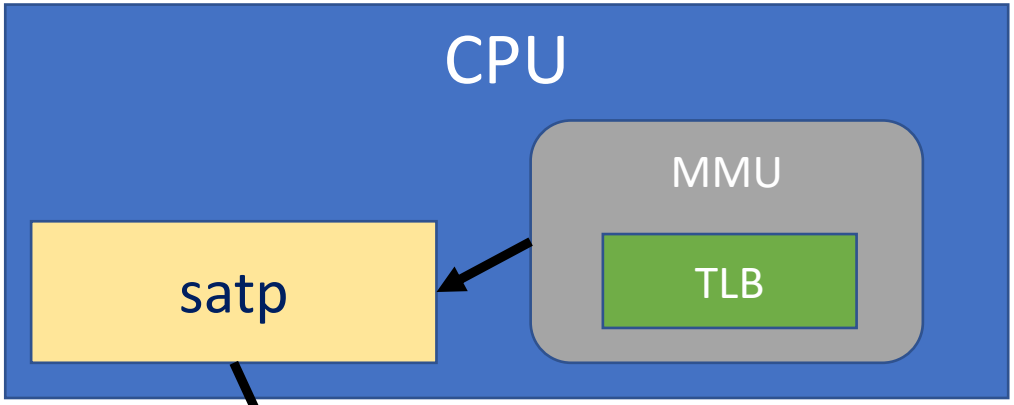


Basically a tree!

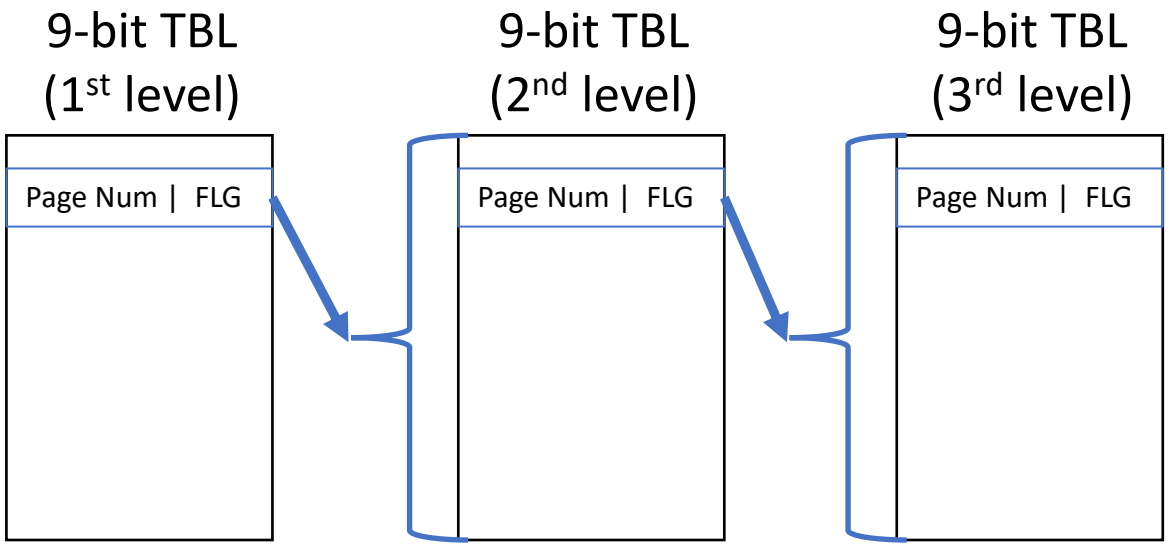
What about a recursive mapping?



How do we program the MMU?



- satp register is a pointer to current page table
- Hardware walks page table tree to find PTEs
- Recently used PTEs cached in TLB



More about flags

X	W	R	Meaning
0	0	0	Pointer to next level of page table.
0	0	1	Read-only page.
0	1	0	*Reserved for future use.*
0	1	1	Read-write page.
1	0	0	Execute-only page.
1	0	1	Read-execute page.
1	1	0	*Reserved for future use.*
1	1	1	Read-write-execute page.

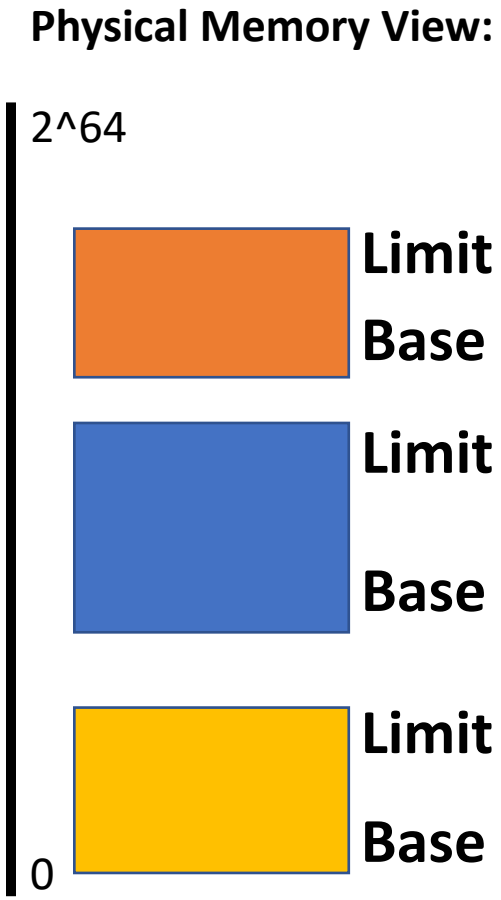
- If **U** is cleared, only the kernel can access
  - Why is this needed?
- What happens if flag permission is violated?
  - We get a page fault!
  - Then what happens?

Handling stale entries in the TLB

- Since TLB is a cache, might contain stale entries:
  - When PTEs are removed
  - When PTE flags change
  - When switching page roots (satp)
- Risc-V provides instruction to flush TLB
  - **LFENCE.VMA**: flushes entire TLB or specific VA
- **G** flag prevents TLB flushes of a PTE
  - Why is this needed?

What about segmentation?

- Base and bounds…
- Why not use this instead of paging?
- Paging seems to be favored but debate is still ongoing
- Really powerful features enabled by paging

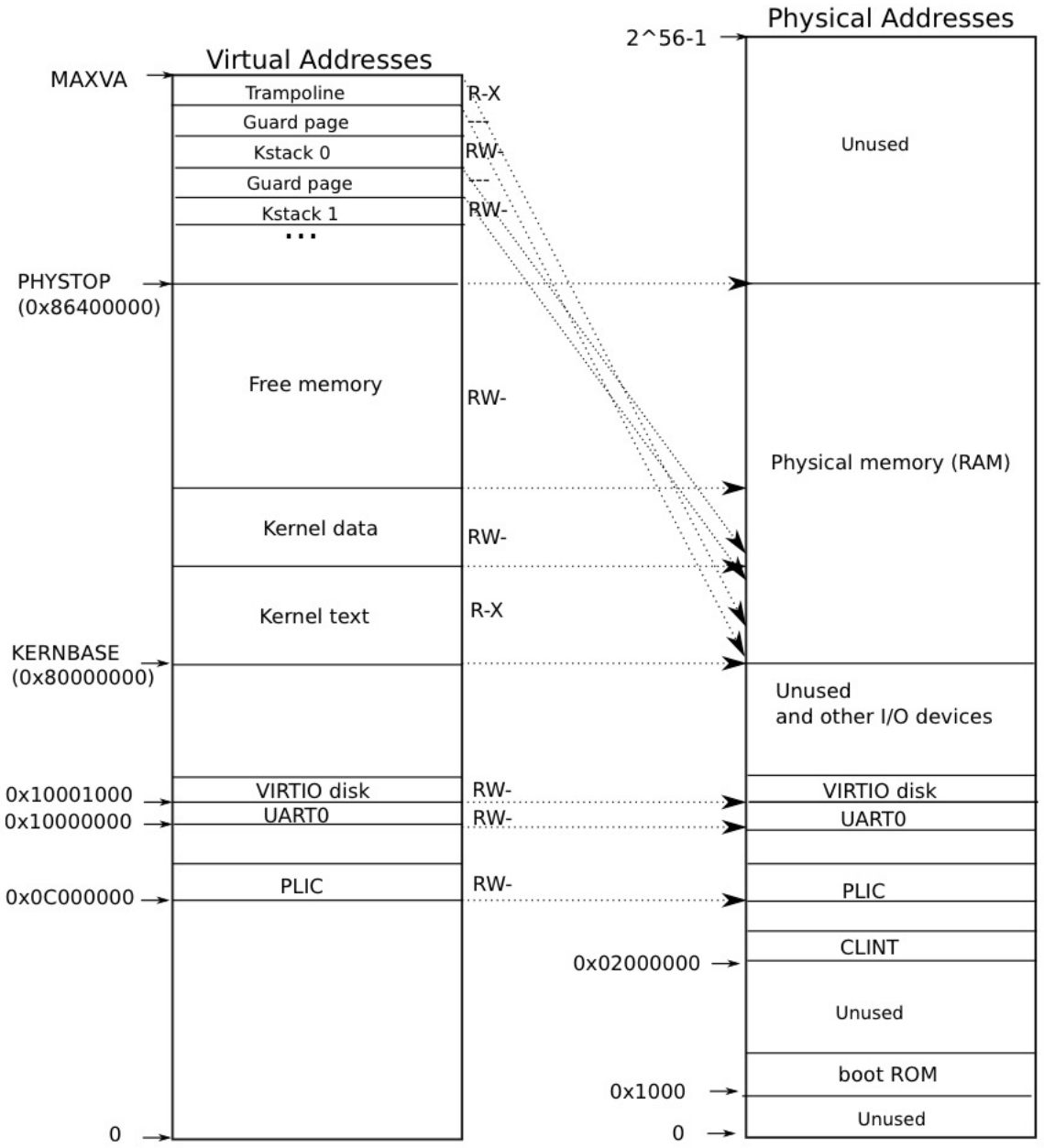


Why use virtual memory while in kernel?

- Isolation benefits for userspace are clear
- Practical reasons
  - Hard (expensive) to turn off paging for each system call
  - Hard to deal with system call arguments that straddle page boundaries
  - Difficult for kernel to support many different hardware physical address layouts
- Reducing fragmentation
  - The kernel needs to allocate memory too

Paging is powerful

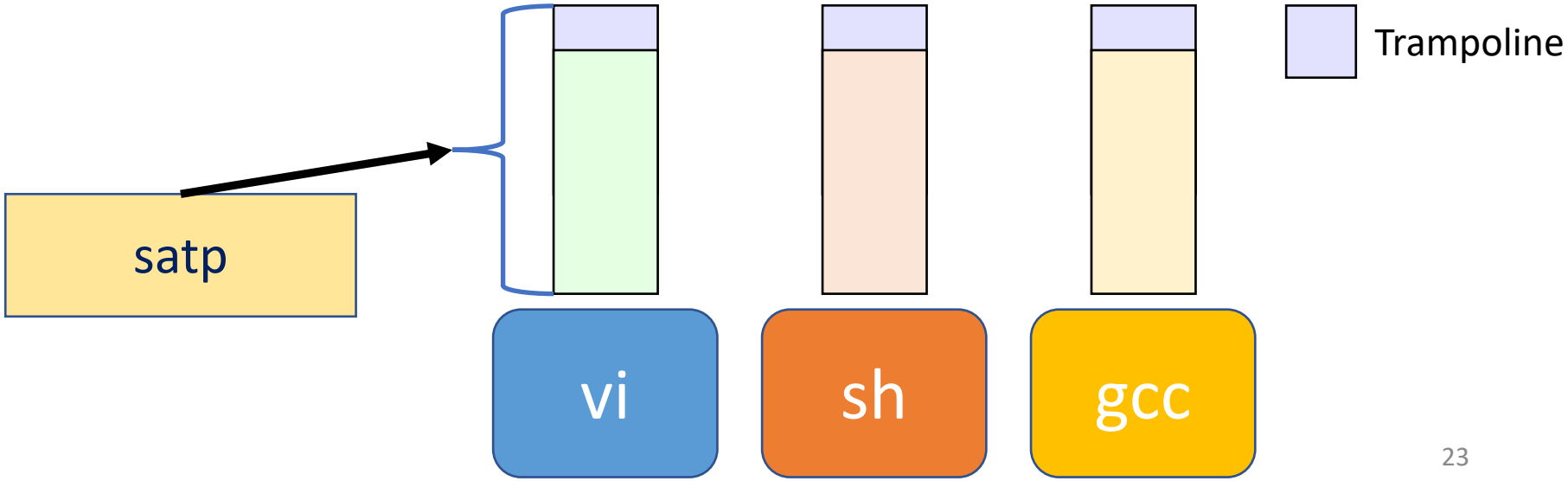
- Copy-on-write: Focus of upcoming lab
- Enables many use cases
  - Lazy memory allocation
  - Runtime system optimizations
  - Topic of upcoming lecture



Kernel memory layout

Processes in xv6

- Each process has its own page table
- Set satp to new page table root when switching processes
- Kernel's trampoline mapping exists in same location in each page table



Q: What permissions for trampoline?

Q: What permissions for trampoline?

- Must be X, but not W
- U and R don’t technically matter

How do processes allocate memory?

void *sbrk(int n)

- n == 0: get current position
- n > 0: allocate memory
- n < 0: free memory