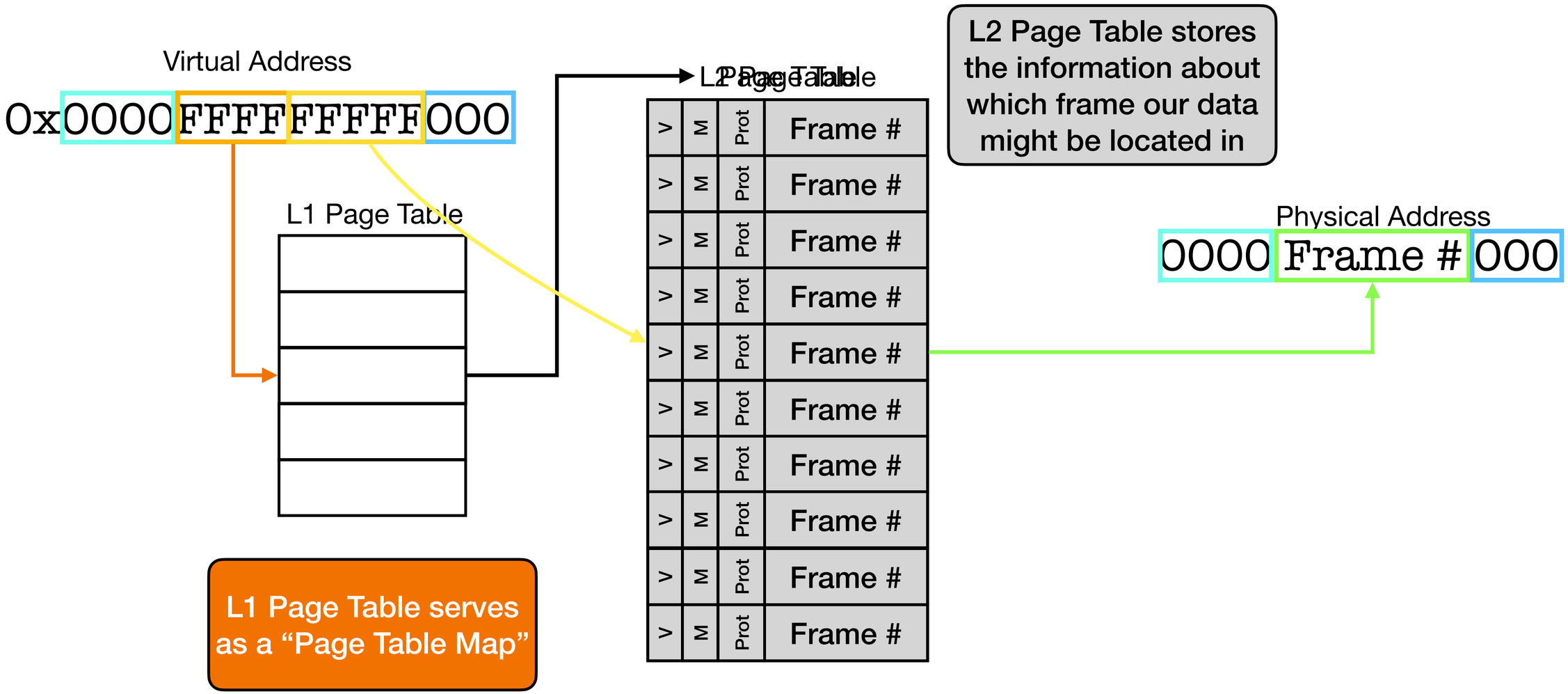
VM + Procs and TLB Shootdowns

FS Code Review in class today!
HW4 (mm) released!

Outline

- Virtual memory and processes
- Cross-process TLB considerations
- File Systems code review

Review: Multi-Level Page Tables



Chat with your neighbor(s)!

In order to do per-process translation, we need to ensure that all processes have a way to translate its virtual address space. What changes should we make to our process struct as we have defined it thus far to implement virtual memory?

Extending the Process Structure

Every process has its own instance of a page table, which means that the OS's VM subsystem needs to track all page tables!

```c
/* Process descriptor */
typedef struct proc {
    pid_t pid;                      // Process ID
    char p_name[MAX_STRING_LEN];    // Process name

    list_t p_threads;               // Threads list
    spinlock_t p_threads_lock;      // lock for threads list
    list_t p_children;              // Children list
    spinlock_t p_children_lock;     // lock for children list
    struct *p_pproc;                // Parent process

    list_link_t p_list_link;        // Link on list of all processes
    list_link_t p_child_link;       // Link on parent's list of children

    vnode_t *p_cwd;                 // The current working directory

    long p_status;                  // Exit status
    proc_state_t p_state;           // Process state
} proc_t;
```

```c
/* Process descriptor */
typedef struct proc {
    pid_t pid;                      // Process ID
    char p_name[MAX_STRING_LEN];    // Process name

    list_t p_threads;               // Threads list
    spinlock_t p_threads_lock;      // lock for threads list
    list_t p_children;              // Children list
    spinlock_t p_children_lock;     // lock for children list
    struct *p_pproc;                // Parent process

    list_link_t p_list_link;        // Link on list of all processes
    list_link_t p_child_link;       // Link on parent's list of children

    vnode_t *p_cwd;                 // The current working directory

    page_table_t *p_page_table;     // Page table

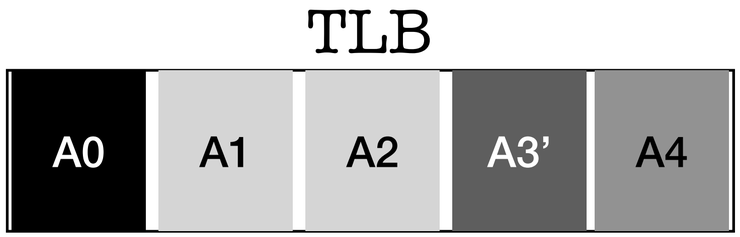
    long p_status;                  // Exit status
    proc_state_t p_state;           // Process state
} proc_t;
```

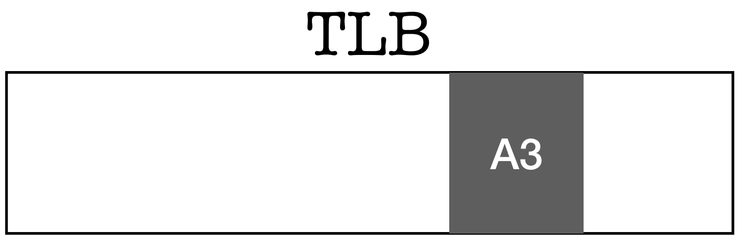
Extending the Process Structure

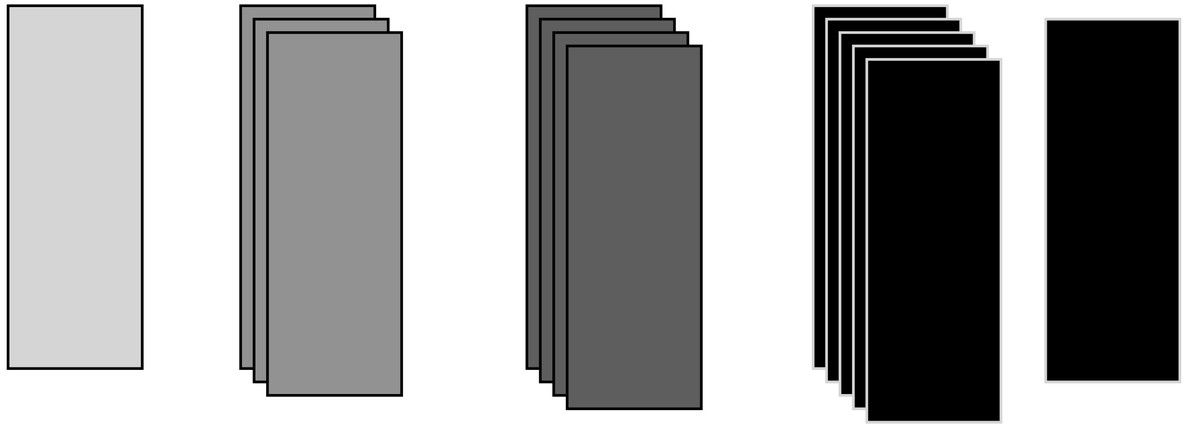
- In our OS kernel, there needs to be a subsystem tasked with tracking all of the fields required to implement virtual memory (e.g., address translation and address allocation) ➡️ in your OS kernel, you are just tasked with address allocation
- The virtual memory subsystem needs to include the state required to track the page tables associated with each process as an attribute of the subsystem
- If our page table sizes are lazily allocated, then the subsystem itself will need to have a mechanism for dynamically growing itself (we will come back to this on Wednesday!)

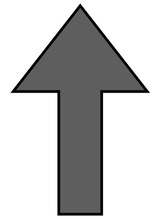
TLB Synchronization

We have a synchronization issue between cached versions of the TLBs!









Update PTE

Page fault! Map a new frame to the page table

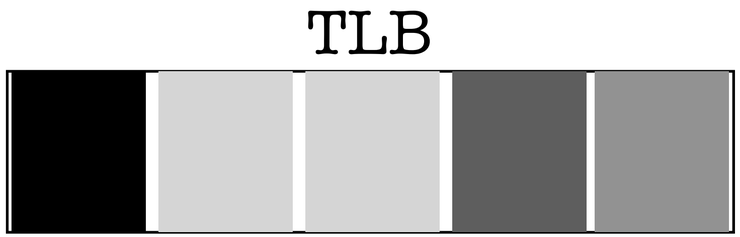
To handle this case, we need a *TLB shootdown* operation to invalidate cached copies of PTEs in other TLBs… updates need to be written through to copy of PTE in memory as well!

Chat with your neighbor(s)!

Consider an instance when the OS performs a context switch. What can go wrong with the TLB?

TLB and Context Switches

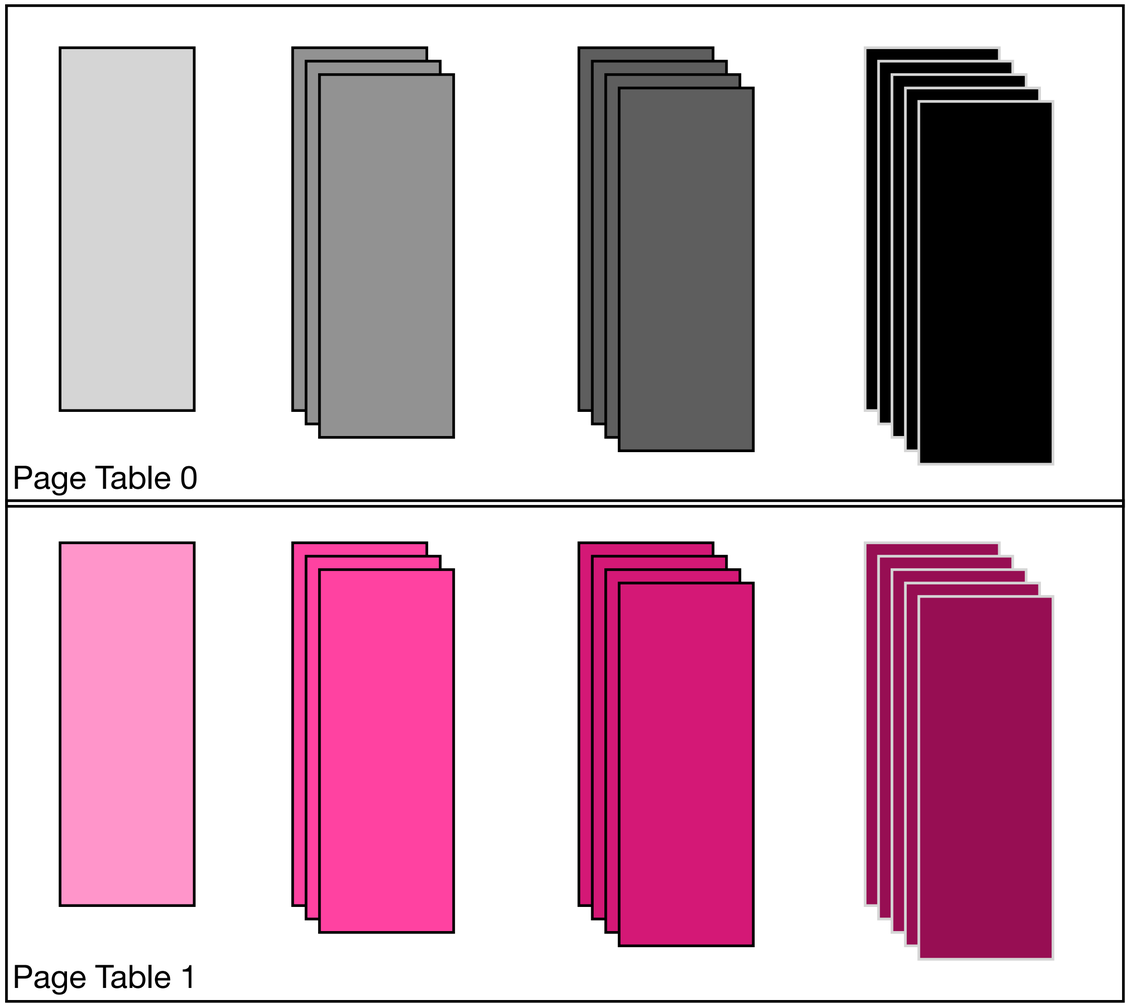
Although we can place the PTE in the TLB without conflict, the remnants of the prior process' TLB behavior is observable… shootdown context switch!



Where to place this PTE?

Leakage of which pages were used by other processes!

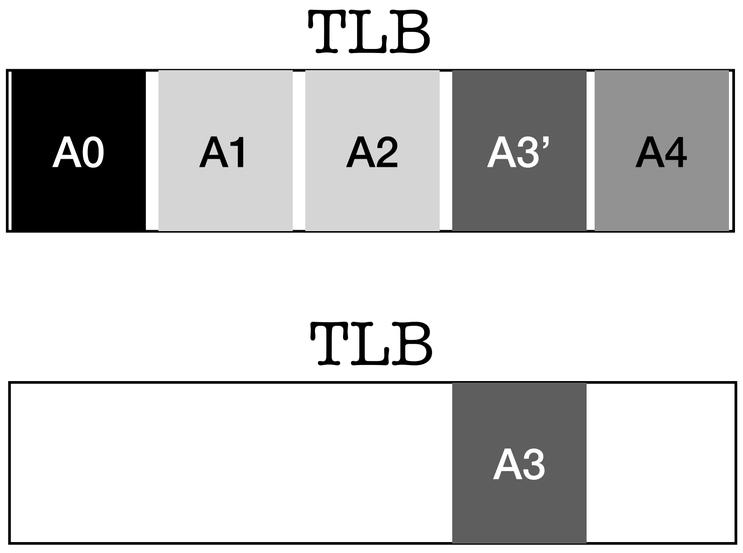
Context switch! Import new page table pointer to the MMU



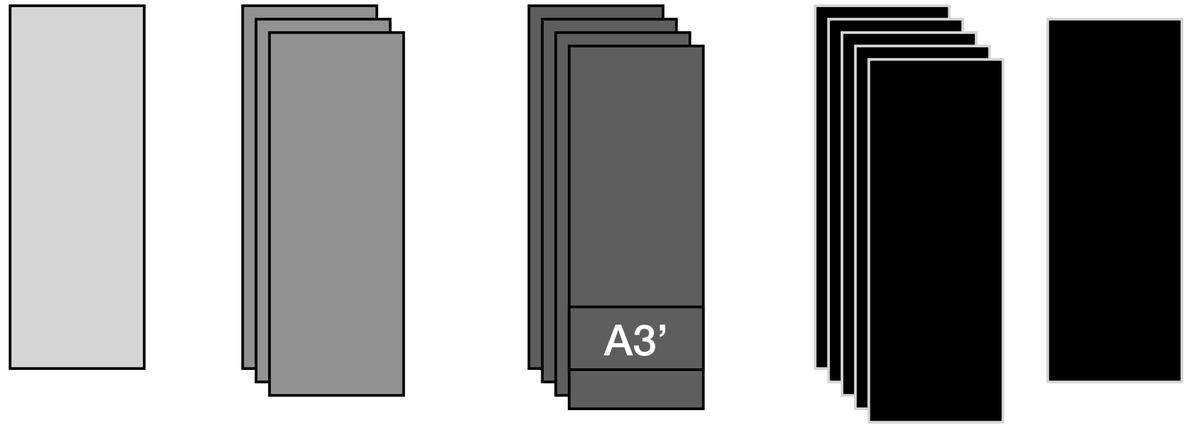
TLB Synchronization and Context Switches

- When running a multithreaded process on two or more processors, page table entries from the page table associated with a process will be cached in multiple TLBs across the system
- If a page fault occurs on one processor, then the update to the associated PTEs should propagate to all locations where the PTE is visible (e.g., the other TLBs)
- The OS achieves this by performing a *TLB shootdown*, a hardware operation in which all cached copies are invalidated, before updating the PTE
- The same mechanism is useful to avoid information leakage via TLB side channel across processes when implementing a context switch

Implementing TLB Shootdowns



Page fault! Map a new frame to the page table



Copy of the page table in memory serves as the true state of the data structure

Cached copy can never be out of sync with the version in memory!

The implementation of this coordination is described as a *TLB shutdown algorithm*

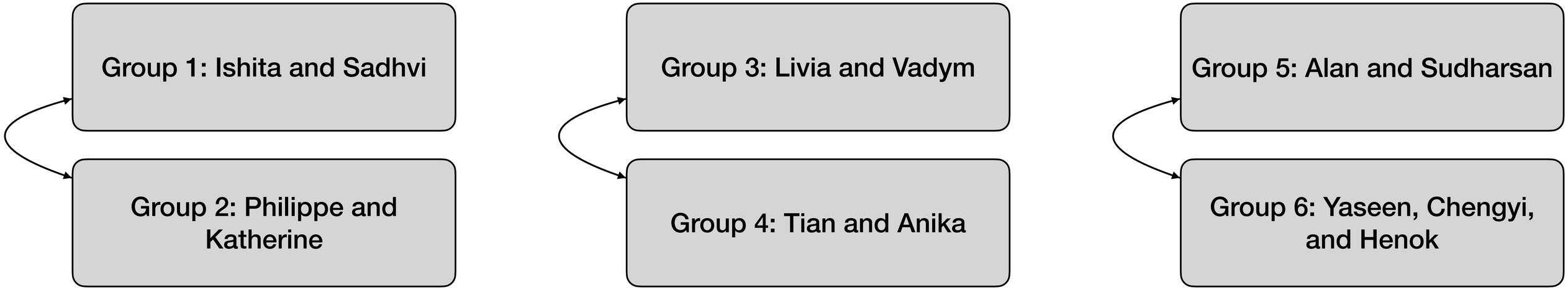
Implementing TLB Shootdowns

- The processor modifying the page table needs to force an interrupt on every other processor in the system that is currently using the address space to perform a *TLB shootdown* on their version of the TLB to ensure that they do not have a cached copy of the page table to update!

- If such a processor exists, then the updating processor needs to wait for confirmation that any cached versions of the corresponding PTE have been invalidate ➡️ only then can the processor write to the PTE

- When performing the write (or any write) updates to the PTE are written *through* the TLB to memory to ensure that all processors have visibility of the change

File Systems Code Review



FS: Road Map

- Programming component is done ✅
- Code Review has been completed ✅
- Code Review Response due by Friday ☑️
- If you would like to submit changes before I look at your code (for thoughts and feedback), you can continue to do so through Friday ☑️
- Homework 4 released! Details on the course page (programming due November 13)

Memory Management Partners

Group 1: Sadhvi and Vadym

Group 2: Alan and Philippe

Group 3: Katherine and Henok

Group 4: Tian and Chengyi

Group 5: Livia and Anika

Group 6: Ishita, Sudharsan, and Yaseen

Summary

- Seeing as address translation is implemented at the process granularity, page tables must be implemented as process-specific structs
- When thinking about the coordination with the TLB, we need to implement TLB shootdowns to ensure consistency and leakage freedom
- To implement TLB shootdowns, processors need to interrupt one another and ensure that updates to TLB entries are written through to memory, which serves as the true version of the translation entry