What makes paging slower?

- Consider the steps performed by paging MMU while moving a value from a memory address to a register (movl 0x0(%eax), %eax)
  - [Bit] Extract VPN and offset from VA
  - [Add] Calculate address of PTE (PTBR + VPN)
  - [Mem] Fetch PTE
  - [Bit] Extract PFN from PTBR and calculate PA ((PFN * Page Size) + offset)
  - [Mem] Fetch value from PA

- Imagine making two memory references for every instruction execution (instruction fetch, load or store)

How can paging performance be improved?

- Adding a limited size hardware cache (part of MMU) that stores the VA to PA translation
- This cache is called as TLB (Translation lookaside buffer) or address translation cache
- This reduces the number of memory references made in the above case by one (step 'c')
- The last step 'e' is removed by provisioning for a hardware processor cache that stores a minimal subset of memory

New address translation mechanism (done by hardware)

- After extracting VPN from VA, check if TLB contains VPN to PFN translation
- If yes, it’s a TLB Hit
- Else, **TLB miss is handled in normal way by referring the page table**

Why does TLB work?

- TLB cannot hold all VPN to PFN translation and the number of entries it could hold is very small
- Spatial Locality: Data close to each other are located within the same page (e.g. scanning an array)
- Temporal Locality: Recently accessed data could be re-accessed soon in the future (e.g. loop)
- Other techniques like prefetching (anticipate the page that will be used next and populate the VPN <-> PFN translation in TLB) are usually employed by processors

Content of TLB

- TLB is like a hash table where VPN is the key to the table and the value is the PTE entry
- The mapping contains PFN and the protection bits

TLB entry validation

- TLB entry contains a valid bit that signifies if the entry is valid or invalid
- Invalid **does not mean** the virtual page does not contain a mapping to a physical page. The hardware now has to scan the page table to find the right entry (TLB miss)

What happens during context switch?

- Note the translations contained in TLB are only valid for a particular process
- They have to flushed during every context switch or invalidated (set valid bit to zero for all entries)
Recent processors store the process id (or address space identifier) along with every TLB entry avoiding the invalidation during context switches

Questions

- Apart from context switch, are there reasons to invalidate a TLB entry?
- If a context switch happens between a parent and a child process, do we need to invalidate TLB?
- If you are given a task of designing a prefetcher for TLB, how would you do it?
- How would you design an experiment to measure the TLB size? (Assume the TLB includes a smart prefetcher)