Lecture 18: Translation Lookaside Buffer (TLB)

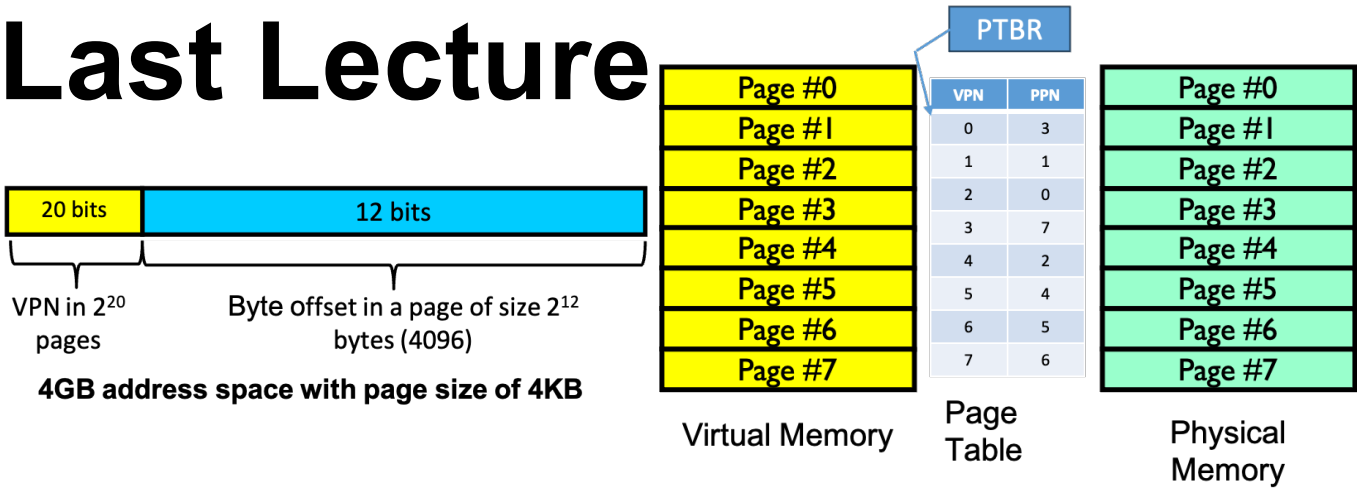
Vivek Kumar

Computer Science and Engineering

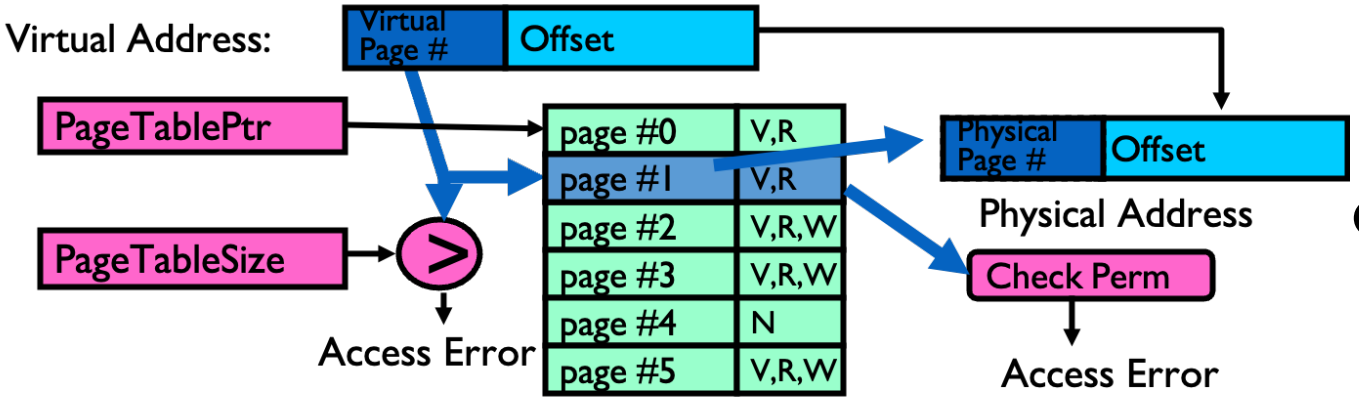
IIIT Delhi

vivekk@iiitd.ac.in

Last Lecture



How to Implement Simple Paging?



- Paging divides the available memory into small and fixed size blocks (4KB in Linux)
- Each VA or PA belongs to a VP or PP, respectively, and the corresponding numbers are VPN and PPN, respectively
- Linux uses a mechanism known as **Page Fault** for lazy page allocation

Today's Class

- TLB
- Locality
- Quiz-3

Mapping Virtual to Physical Address

- Approaches
  1. Virtual address is same as the physical address
  2. Virtual address is not the same as physical address
     a) Using base & bound for VA to PA mapping
     b) Using segmentation for VA to PA mapping
     c) Paging
        a) Using Page Table (PT) in RAM and having a base register for PT
        b) Using Page Table (PT) in RAM and having a base register for PT as well as a TLB cache

Page Table Entry

- Recall, a page table is a mapping between VPN to PPN
- What kind of data structures we can use for implementing a page table?
  - Arrays?
    - VPN will be the index in an array of PPN
  - Hash table?
    - VPN will be the key with the value as PPN

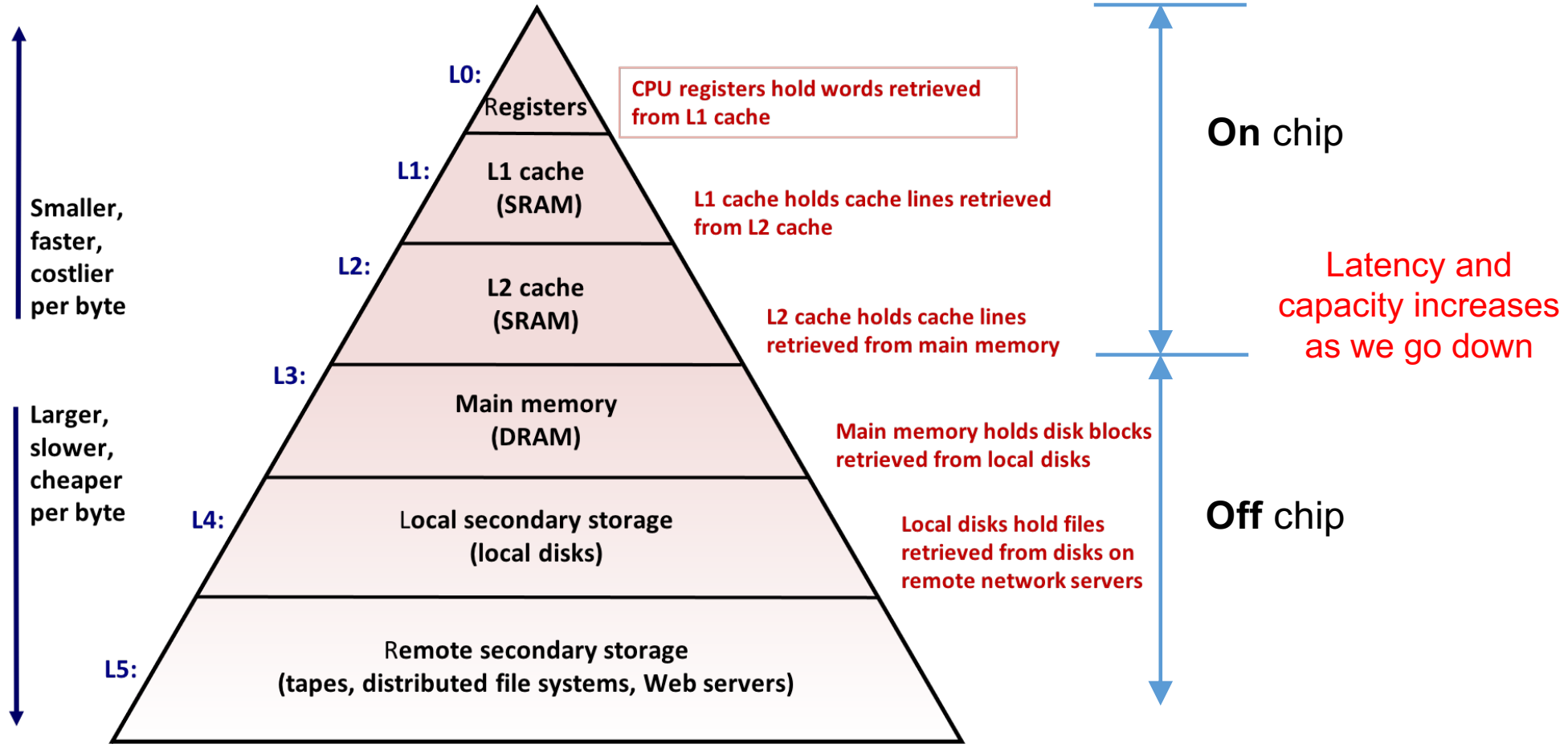
Overheads with the Page Table

- **Time overhead**

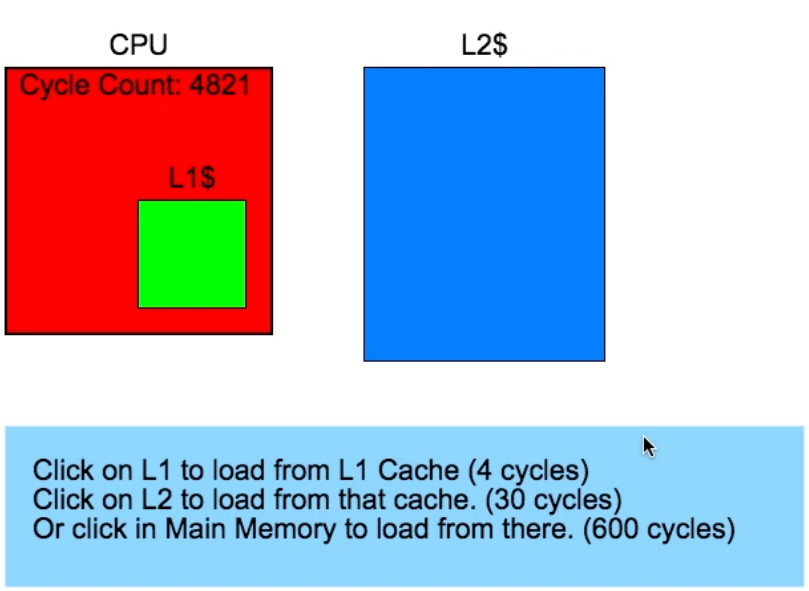
Discussion for today

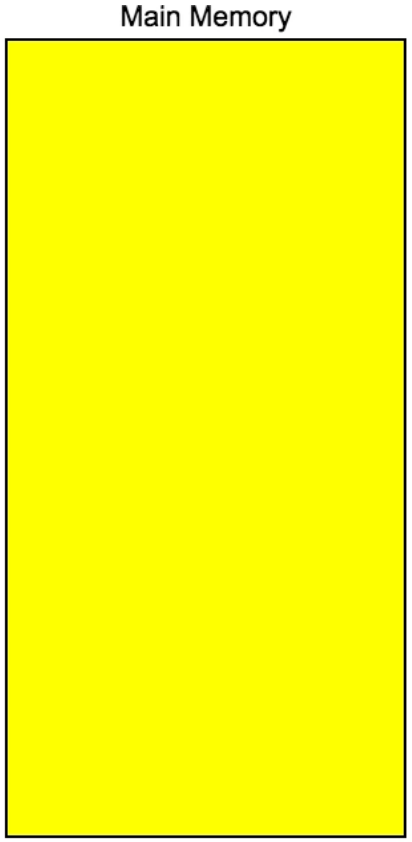
  - Accessing any virtual address require one extra memory access
    - Step-1 is to fetch the page table base address from register (PTBR)
    - Step-2 is to index into the PTE and fetch the PPN
- Space overhead
  - Next lecture

Detour: Memory Hierarchy



Detour: How Bad is Latency as we go Down?

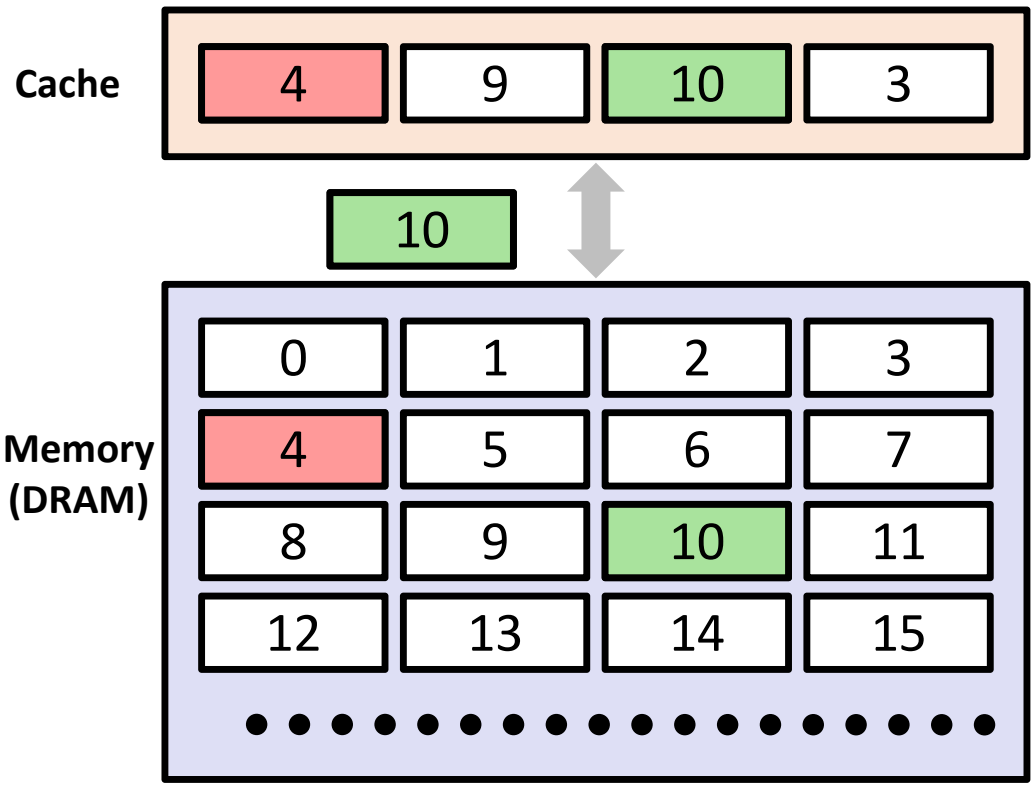




- Another analogy
  - Normalizing with L1 latency, and assuming one seconds is equal to 4 cycles
    - L1 = one second
    - L2 = 7.5 seconds
    - Main memory = 2.5 minutes
    - Hard drive = in several days!

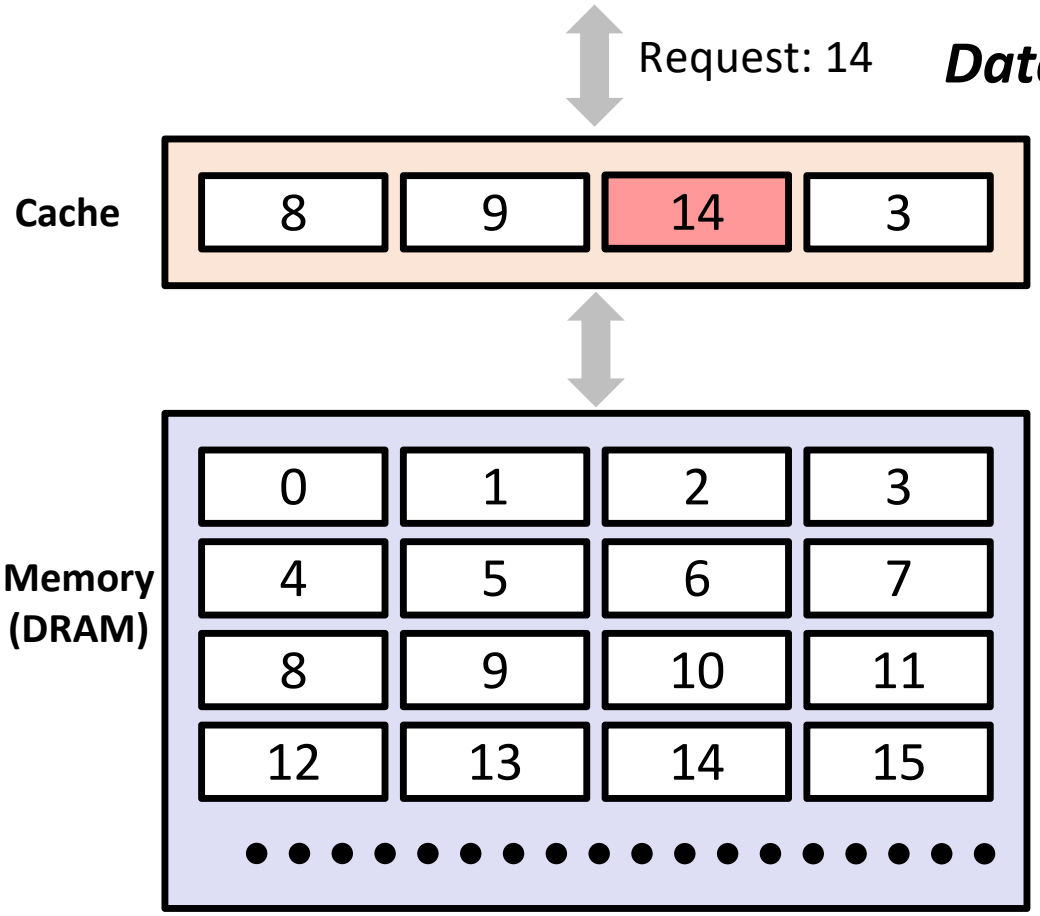
Animation source: https://overbyte.com.au/misc/Lesson3/CacheFun.html

Detour: General Cache Concepts



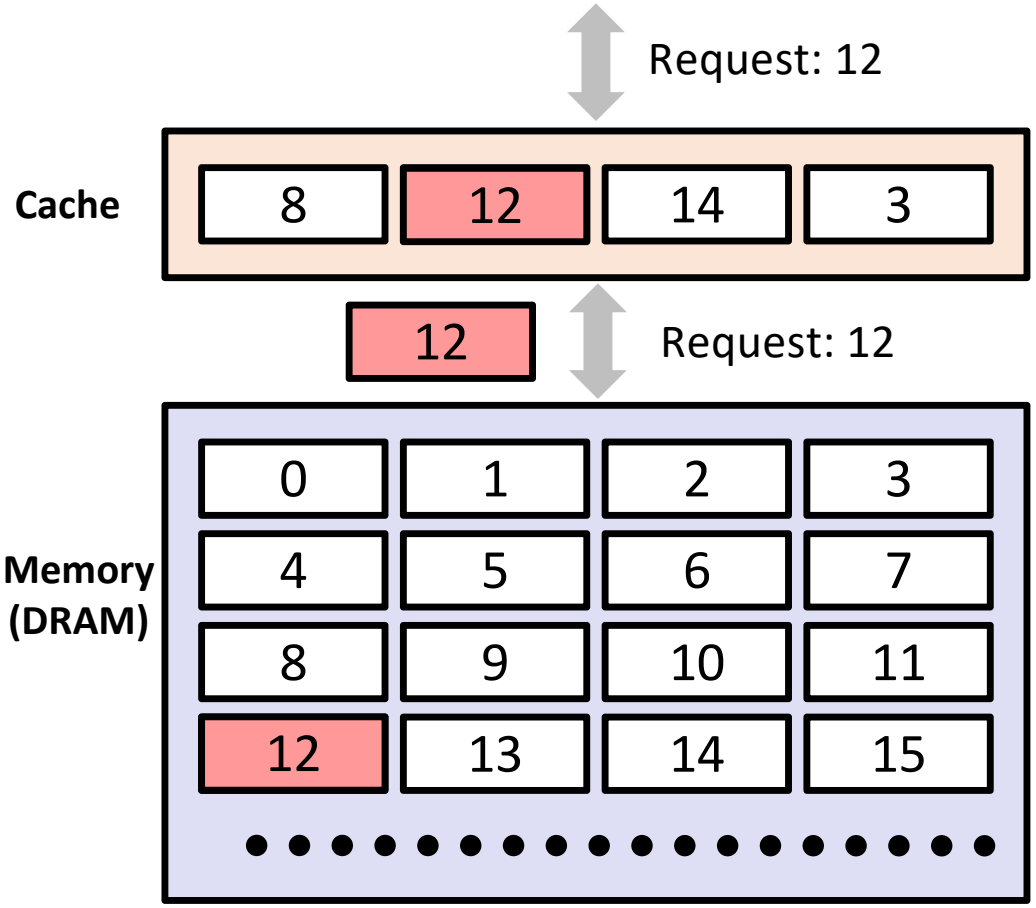
- Cache: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device

Detour: General Cache Concepts



- Cache hit
  - Data is already in the cache

Detour: General Cache Concepts



Data needed: 12

Data is not in cache:
Miss!

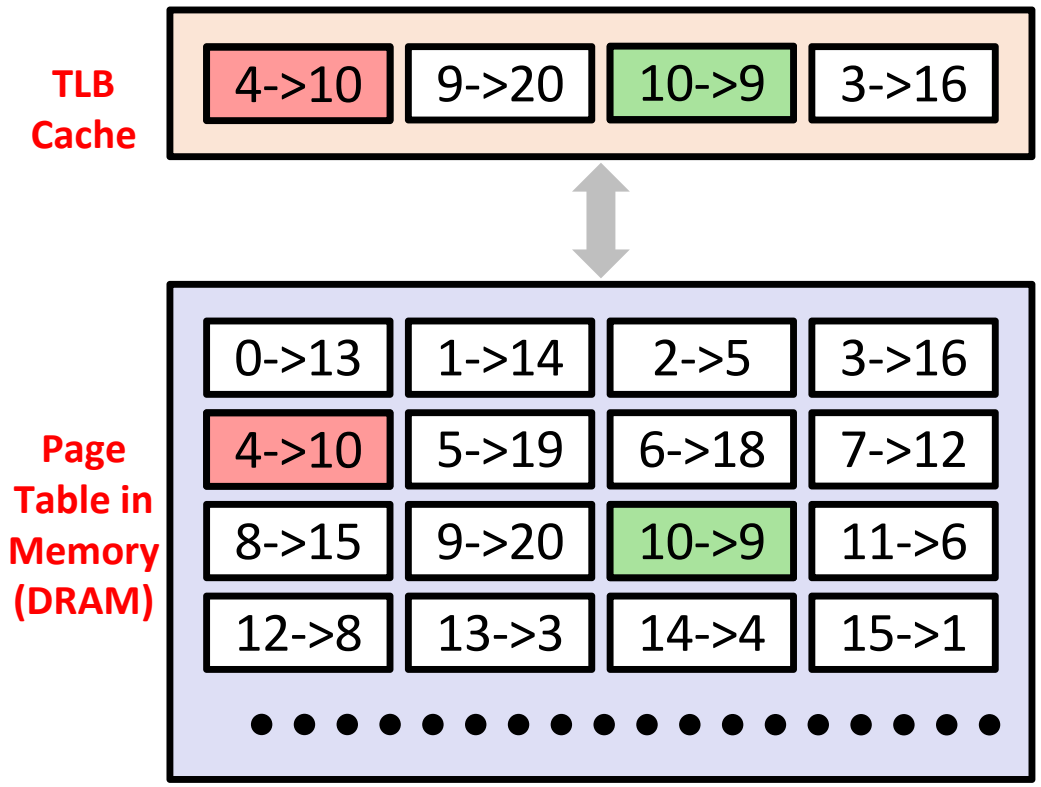
Data is fetched from *memory*

Data is stored in cache
By evicting some old data

So, How to Solve Slow Access of PT?

- Provide a hardware cache per CPU for saving frequently accessed PTE entries (different than the CPU cache L1, L2, etc.)
  - Translation Lookaside Buffer (TLB) cache that is a part of MMU
  - Typically accessed in a single cycle
- Save most recent VPN->PPN mappings on TLB (plus some more info, but not PT entry)

TLB Hit and Miss

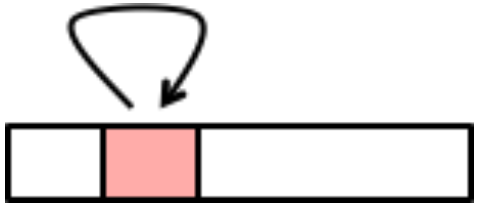


- TLB hit
  - If entry already in TLB then have the physical address without reading the page table
  - VPN(4)->PPN(10) is already on TLB. Hence, no need to access PT
- TLB miss
  - VPN(10)->PPN(9) mapping is not found on TBL
  - Locate PT using PTBR
  - Access page table to find the mapping and save the mapping on cache by evicting an earlier entry

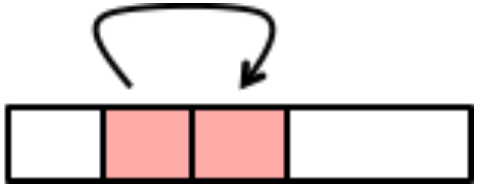
Locality Helps in Caching

- Principal of Locality
  - Empirical observation: Processors tend to access same set or nearby memory locations repetitively over a short period of time

- Temporal locality:
  - Recently referenced items are likely to be referenced again in the near future



- Spatial locality:
  - Items with nearby addresses tend to be referenced close together in time



Locality Helps in Caching

- Caching helps only if application supports **locality**
  - Let us understand the concept of **locality** using a simple application called as **iterative averaging**
  - Iterative averaging is the process of updating an array to so that each index becomes the average of the indices one before and one after it
  - After repeating this for many iterations, the array may converge to one set of numbers
    - Repetitive pattern in several scientific applications

Iterative Averaging

```
//all elements zeroed
float A[SIZE+2], B[SIZE+2];

void iterative_averaging() {
  A[SIZE+1] = B[SIZE+1] = 1;
  for (int iter=0; iter<ITERATIONS; iter++) {
    for (int j=1; j<=SIZE; j++) {
      B[j] = (A[j-1] + A[j+1])/2.0;
    }
    double* temp = B;
    B = A;
    A = temp;
  }
}
```

https://classes.engineering.wustl.edu/cse231/core/index.php/Iterative_Averaging

- For SIZE=9, the array A will eventually converge with values as 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9

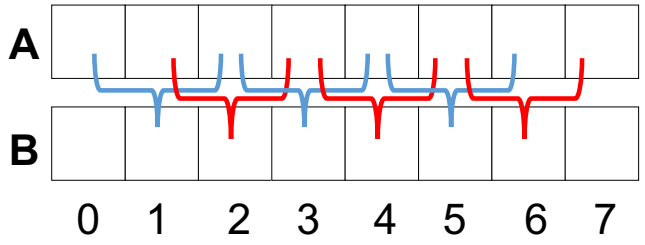
Spot the Locality

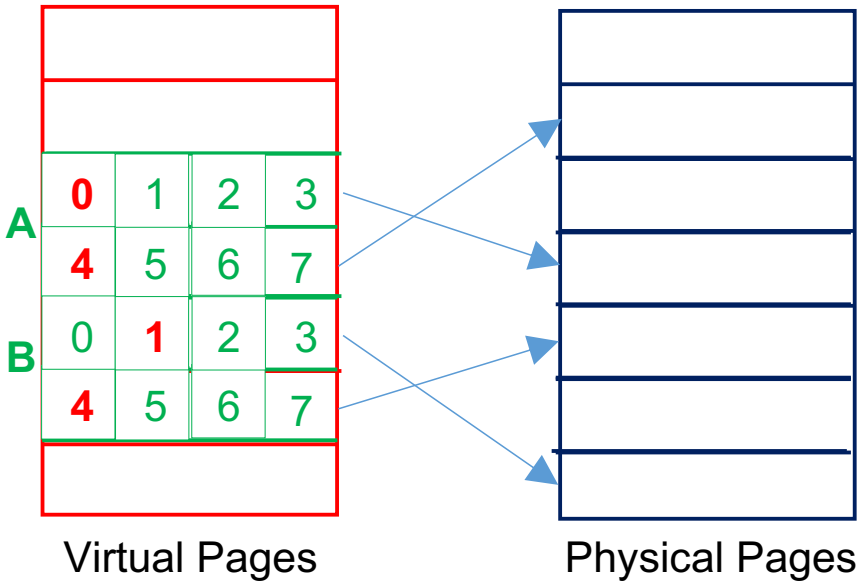
```
//all elements zeroed
float A[SIZE+2], B[SIZE+2];

void iterative_averaging() {
  A[SIZE+1] = B[SIZE+1] = 1;
  for (int iter=0; iter<ITERATIONS; iter++) {
    for (int j=1; j<=SIZE; j++) {
      B[j] = (A[j-1] + A[j+1])/2.0;
    }
    double* temp = B;
    B = A;
    A = temp;
  }
}
```

https://classes.engineering.wustl.edu/cse231/core/index.php/Iterative_Averaging

- Let us only consider the locality in array accesses
- Spatial locality
  - Inner for-loop
    - Reference array elements in succession (stride-1 reference pattern)
- Temporal locality
  - Outer for-loop
    - Accessing each arrays elements repeatedly

Spatial Locality

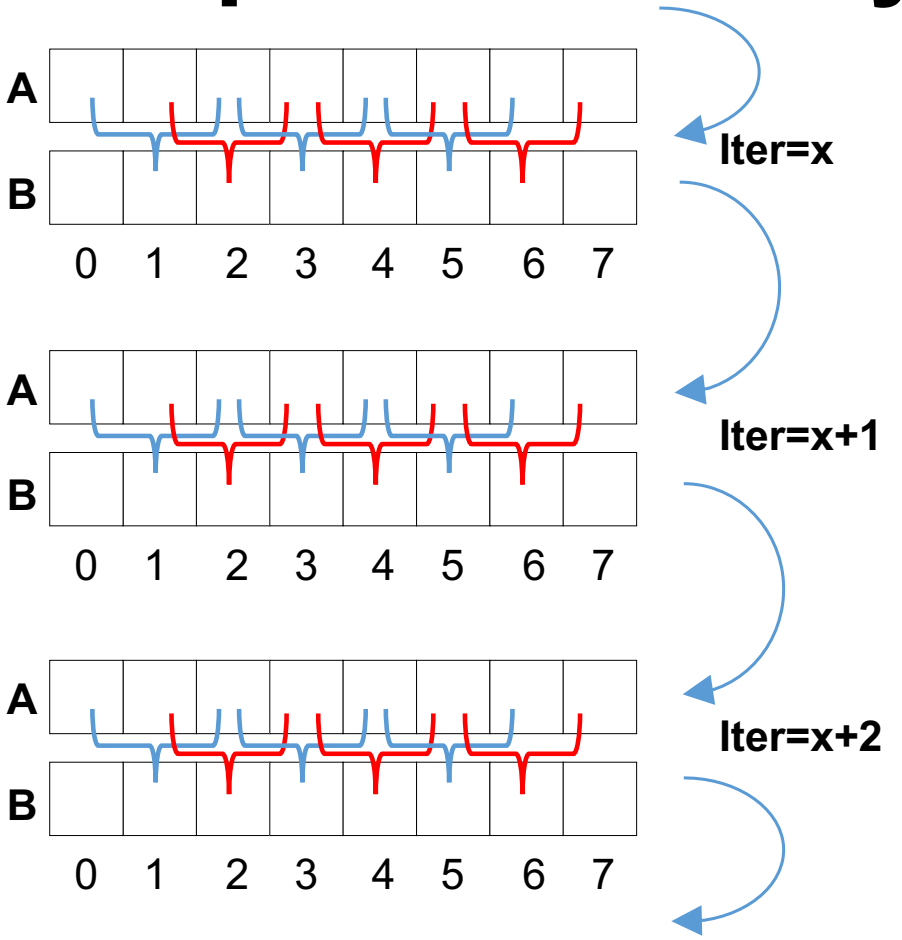




- Assumptions
  - The only memory access required are for accessing elements of arrays A and B
  - SIZE=6
  - Page size (PS) of 16 bytes
    - A requires 2 pages
    - B requires 2 pages
  - **TLB can hold 4 entries**
- Inside each iteration of outer loop
  - B is accessed **6** times and A is accessed **12** times (spatial locality)
  - Total memory accesses = 12+6=**18**
  - TBL misses
    - **2+2 = 4**
  - TLB hits
    - **10+4 = 14**

```
for (int j=1; j<=SIZE; j++) {
  B[j] = (A[j-1] + A[j+1])/2.0;
}
```

Temporal Locality

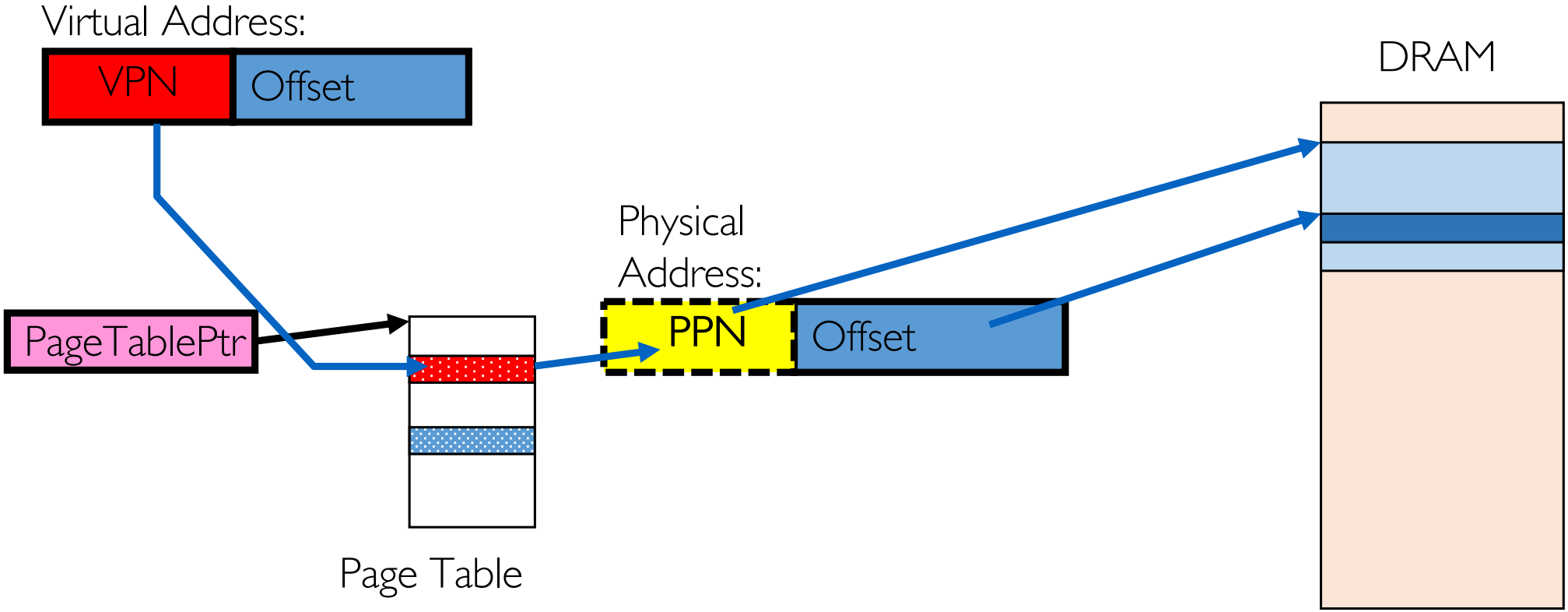


- Same indices are repeatedly accessed across the iterations of the outer for-loop
  - **Temporal locality**
- Total TLB misses for N number of iterations of outer for-loop?
  - 4 only!
- Total page table accesses with TLB?
  - 4 only!
- Total page table access without TLB?
  - 18N

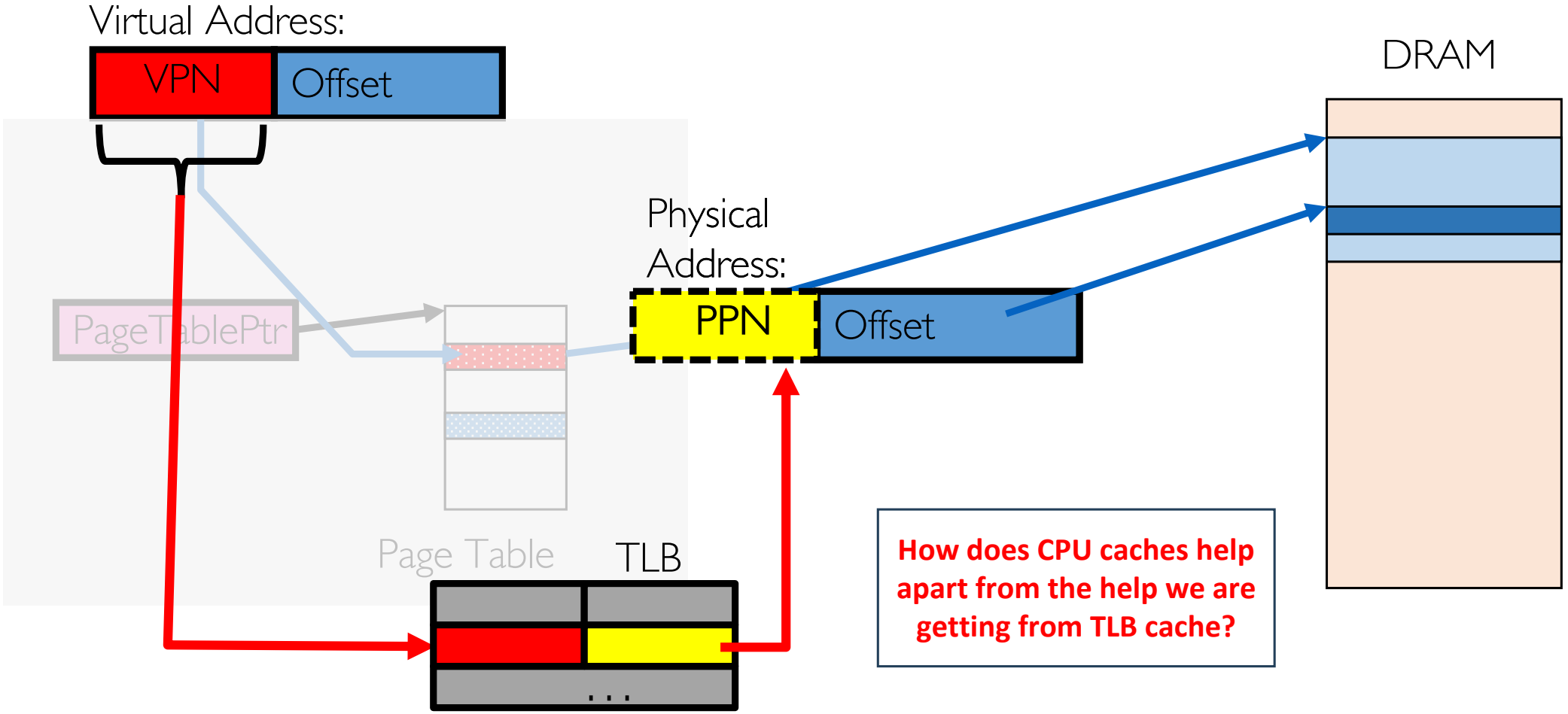
What to do at Context Switch?

- Need to do something, since TLBs map virtual addresses to physical addresses
  - Address Space just changed, so TLB entries no longer valid!
- Options?
  - Invalidate ("Flush") TLB: simple but might be expensive
    - What if switching frequently between processes?
  - Include ProcessID in TLB
    - This is an architectural solution: needs hardware

Putting Everything Together: Address Translation



Putting Everything Together: TLB



Next Lecture

- Space overheads with the page table