Introduction to Algorithm Engineering

Spring 2022

Lecture 2

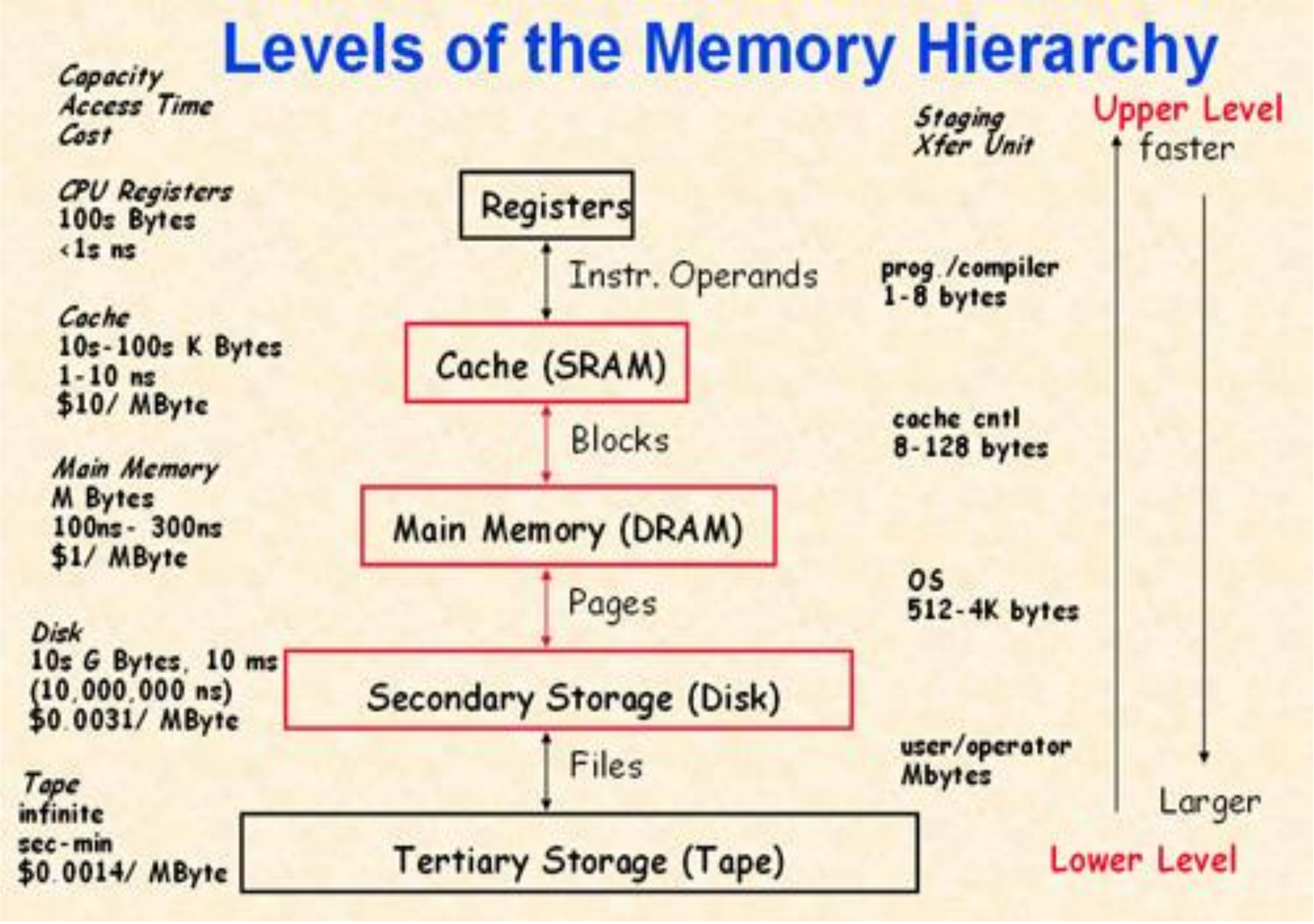
This Lecture

- In this lecture, we will study algorithm implementation from an architecture friendliness point of view.
- The central question to answer is: How Far Should Algorithms Consider the Architecture?
- There are predominantly two schools of thought
  - Very much
  - Should never
- Both sides have good arguments.

Cache Aware Algorithm Design

- A model that is relevant to efficient implementations of algorithms.
- Models the twin challenges of algorithm design for big data and a deep memory hierarchy.
- For instance, consider the case that the entire data does not fit in the memory of a computer.
- Then, the assumption that all memory references require the same access time is not truly valid.
- How to design and implement algorithms when the above assumption does not hold?

Example Hierarchy



Some Definitions

- Working Set: A collection of data items recently accessed by a program.
- A small working set is good. Can fit in the cache, and a prefetcher can help in most cases minimize cache misses.
- However, prefetch works best for computations with very strong locality in the computation.
  - Example: Add elements of two arrays index-wise.
- But, there are other computations that have strong non-local access patterns.
- In such cases, explicit memory management helps.

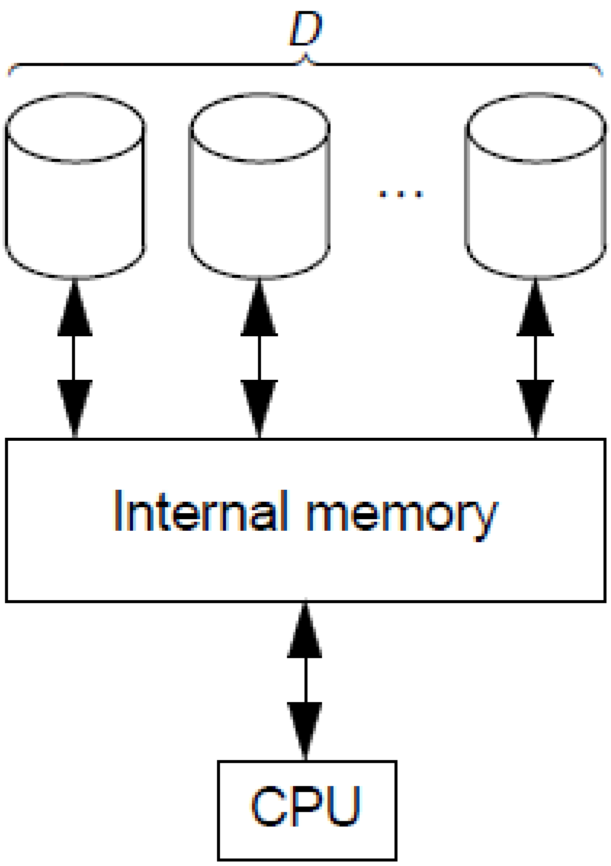
Performance Parameters

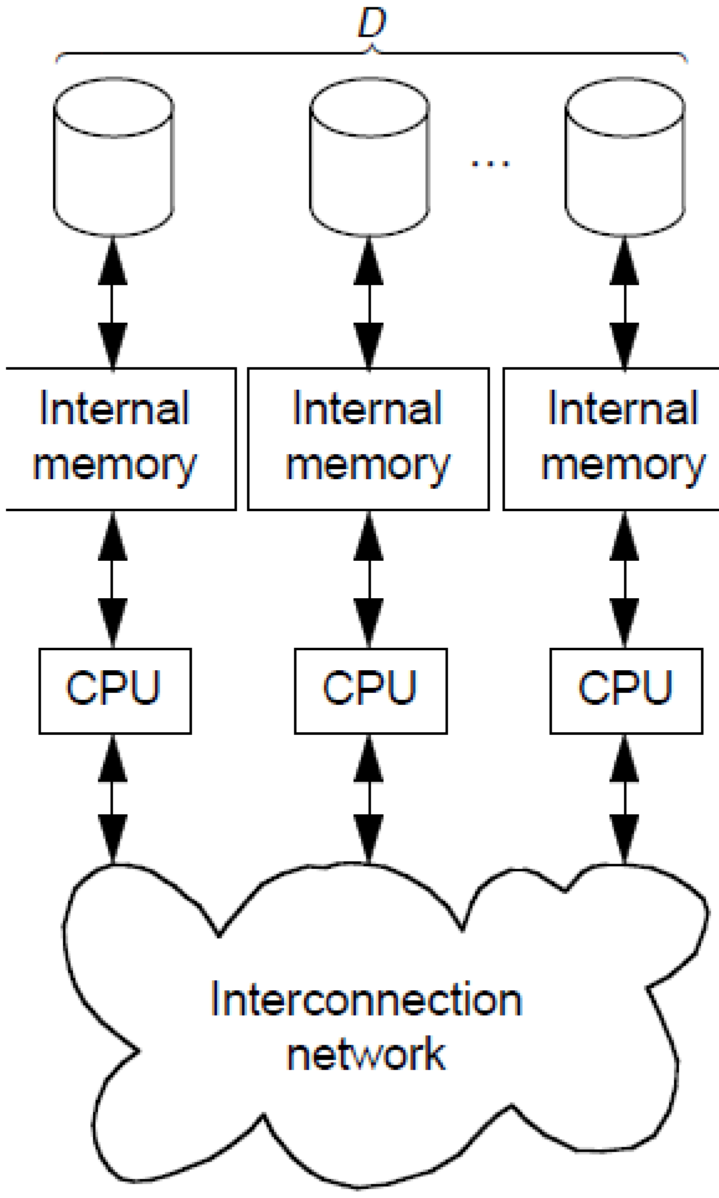
» The number of I/O operations,
» The disk space usage, and
» The CPU time.

» N = problem size (in units of data items);
» M = internal memory size (in units of data items);
» B = block transfer size (in units of data items);
» D = number of independent disk drives;
» P = number of CPUs

» Notice that $M < N$, and $1 \leq DB \leq M/2$

Model in Pictures





Four Fundamental Operations

- » Scanning (a.k.a. streaming or touching) a file of N data items, which involves reading or writing the items in the file sequentially.
- » Sorting a file of N data items, which puts the items into sorted order.
- » Searching online through N sorted data items.
- » The fourth one, is on batched query style operations. Read from Vitter's survery.

Scanning

» The I/O bound Scan(N) = O(N/B).

» Relatively straight-forward to understand.

» Sorting requires some additional tricks.

Sorting

- We will show that sorting N items with a buffer ratio B requires $O(N/B \log_{(M/B)} N/B)$ disk I/O operations.
- Two main techniques
  - Sample (Distribution) sort
  - Sorting by merging

Sample Sort

- First the basic idea
- Imagine an extension of quick sort.
- Choose s – 1 pivots. (For quick sort, s = 2).
- Use these pivots to divide the elements into s buckets.
- At this point, elements in bucket i are all smaller than those in bucket i+1, for i=1,2,..., s-1.
- Sort each bucket recursively.

Example Sample Sort

Sample Sort

- Just as in quick sort, need good pivots so that each bucket has roughly the same number of elements.
- Suppose we can find such pivots.
- Let us analyze the number of I/O operations.

Sample Sort

- For equal sized buckets, at every recursive level, the size of the bucket drops by O(s).
- So, there are O($\log_s$ N/M) levels of recursion.
  - Why N/M?
- In each level of the recursion, we have to scan the entire data across all the subproblems.
- At s = O(M/B), we get the required bound on the number of I/Os.

How to Find Good Pivots Quickly?

» Several techniques exist.

» One is find the kth quantiles.

  - Guaranteed to be good

» Randomized methods also are possible.

» More techniques for the multiple disk model.

» Read from the survey of Vitter.

Sample Sort – Multiple Disks

- With multiple disks, the algorithm gets more complicated.
- Need to ensure that buckets are written into all disks in near equal quantities.
  - Otherwise, cannot use multiple reads at the same time.