External Memory and B-Trees

Memory Limits: Directory of Telephone Numbers

- Suppose we have to implement a **Directory Assistance System** for Bell Canada
- Let's say that Canada has 100 million telephone numbers
- Even if we reserve only 100 bytes for each directory entry, the directory would have 10GB
- 10GB of data don't fit into the main memory (RAM) of even the largest servers.
  → Our directory operations **`search`**, **`insert`**, **`remove`** will have to **access external memory** (e.g., hard disk drive) while they execute

Memory Limits: Other Scenarios

- For other examples of data that won't fit into the internal memory of the CPU that manipulates it, consider:
  - The index of a web search engine
  - The data base of a credit card company
  - The inventory of amazon.com
  - A library information system
  - An English-German dictionary on a palm pilot
  - “Wearable maintenance computers” for air planes

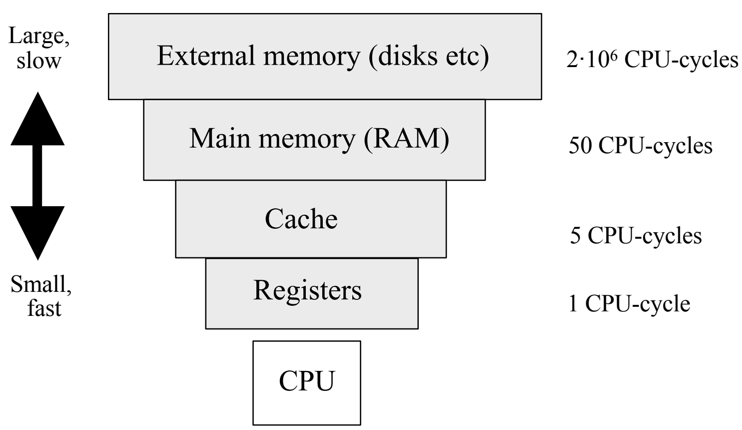
Memory Limits: Internal versus External

- Unfortunately, accesses to external memory (e.g., disk, CD-ROM, tape) are **much slower** than accesses to internal memory (e.g., registers, cache, RAM)
- To optimize run-time performance, algorithms need to **minimize external memory accesses**
- Up until now, only **logical view** of memory: **uniform** (doesn’t matter if data is in memory, on disk etc)
- However, **performance view** of memory: **not uniform** (registers, cache, RAM, hard disks, CDs, floppy disks all have different performance characteristics)
- Whenever algorithms work on data that does not fit into internal memory, performance difference between internal and external memory has to be taken into account

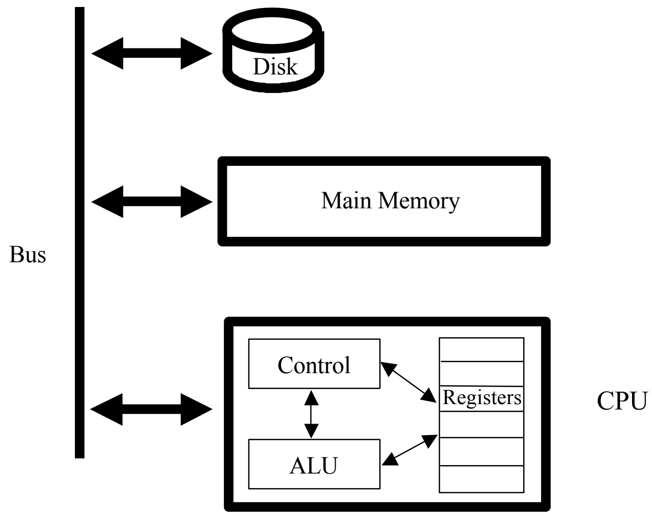
Memory Hierarchy

- Many problems that modern computers are given to solve (analyzing scientific data, running Win95, etc.) require **large amounts of storage**
- **Ideally**: all necessary information could be stored on chip in processor's **registers**, but that's **not feasible**
- **In reality**: computers use a **memory hierarchy** wih trade-off between speed and volume
- The hierarchy consists of four layers:
  - Registers
  - Cache
  - Internal memory (RAM)
  - External memory (Disk)

Memory Hierarchy



Memory Hierarchy (cont'd)

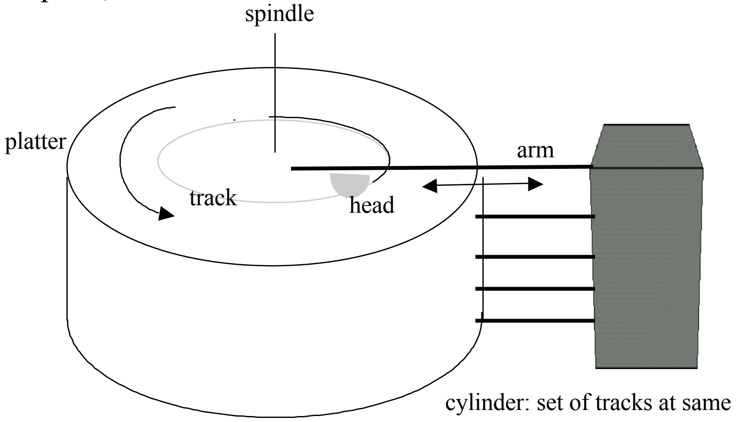


Caching and Blocking

- To minimize access to external memory, two assumptions about use of data are helpful:
  - **Temporal Locality**: If data is used once, it will probably be needed again soon after
  - **Spatial Locality**: If data is used once, the data next to it will probably be needed soon after
- Each assumption gives rise to a different **optimization technique**:
  - **Caching** (based on temporal locality and virtual memory):
    - Provide address space that is as large as secondary storage (virtual memory)
    - When data is requested from secondary storage, it is transferred to primary storage (cached)
  - **Blocking** (based on spatial locality):
    - When address A is requested from secondary storage, a large contiguous block (page) of data containing A is transferred into primary storage

Why Is External Memory So Slow?

- Because it requires the **mechanical movement** of disk parts, rather than the movement of electrons!



Why Is External Memory So Slow?

- External memory is large, slow, cheap
- In fact, external accesses are so slow that **many internal accesses are still faster than a single external access**
- It's slow, because of **mechanical positioning of the disk head** at the beginning of a block involved in a memory access
- Once block is found, actual read/write of block is pretty fast
  - → Our goal is thus to **minimize number of accesses**, not the number of bytes read or written
  - → Once the head is positioned, we might as well **read the entire disk block**
- For problem of implementing a large dictionary: minimize number of times we transfer a block between secondary and primary memory (**disk transfer**) during queries and updates.

How To Store Canada's Telephone Directory?

- As **sequence**:
  - O(N) time and O(N) disk accesses

  → Really, really, really bad!
- As **balanced, binary search tree**:
  - About $\log_2$ N time and $\log_2$ N disk accesses

  → Good, but can do better (by a constant factor)
- Consider the search algorithm for (2,4)-trees:
  - Every node on search path may have to be read from disk
  - Since a node contains at most 3 items, a node typically won't fill a block

  → If nodes contain more items, can reduce the height of the tree and make better use of a single disk access

(*a*,*b*)-Trees: Definition

- Generalize (2,4)-trees to allow for bigger nodes
- An (*a*,*b*)-tree is a multi-way search tree such that
  1. $2 \le a \le (b+1)/2$
  2. **Size property:** each internal node has
     - at least *a* children, unless it's the root, and
     - at most *b* children
  3. **Depth property:** all external nodes have the same depth
- Why $a \le (b+1)/2$?

(a,b)-Trees: Definition

- **Why $a \le (b+1)/2$?**
- Because it guarantees that
  - An **overflow with split** results in legal node:
    - Break (b+1)-node into $\lceil (b+1)/2 \rceil$-node and $\lfloor (b+1)/2 \rfloor$-node
    - $a \le (b+1)/2$ implies $a \le \lceil (b+1)/2 \rceil$ and $a \le \lfloor (b+1)/2 \rfloor$

(a,b)-Trees: Definition

- An **underflow with fusion** results in legal node:
  - Merge an (a – 1)-node with a sibling that is an a-node to get a new (2a – 1)-node.
  - $a \le (b+1)/2$ implies $2a - 1 \le b$

(a,b)-Trees: Height

- **Theorem:** The height h of an (a,b)-tree T is $\Omega(\log_b n)$ and $O(\log_a n)$
- For a balanced binary tree with 100 million entries the height is approximately 18.
- For example suppose that a = 16 and b = 32. That would yield a height of between approximately approximately 5 and 6.
- In practical terms this would reduce the number of disc accesses in a search from 18 to 5 or 6, a huge savings.