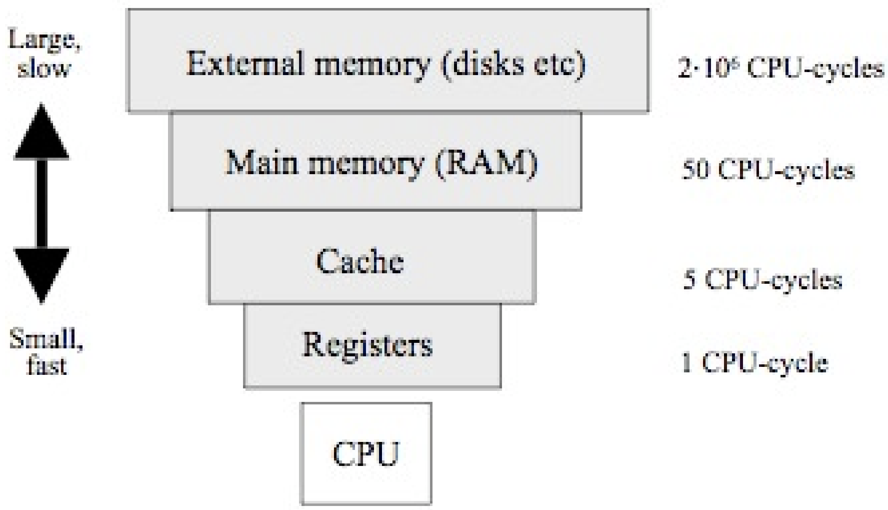
External Memory and B-Trees

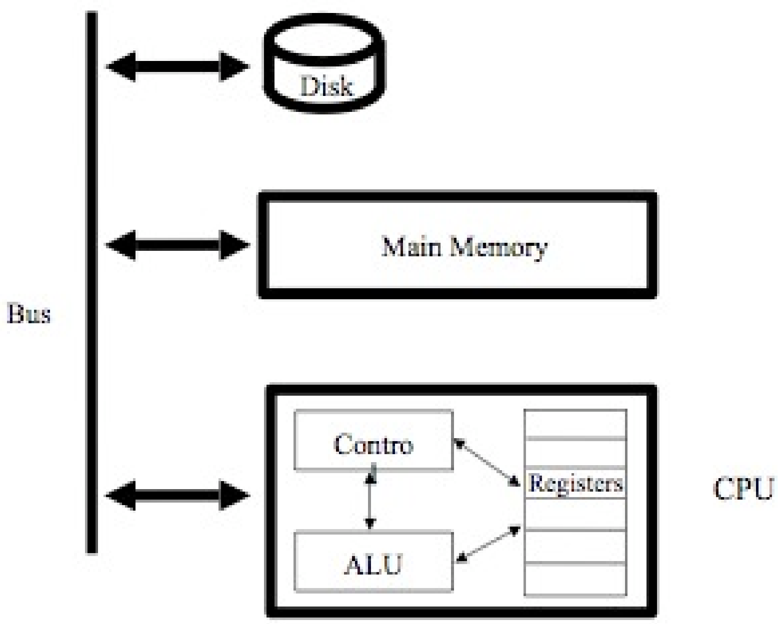
Memory Limits: Internal versus External

- 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



Memory Hierarchy (cont'd)

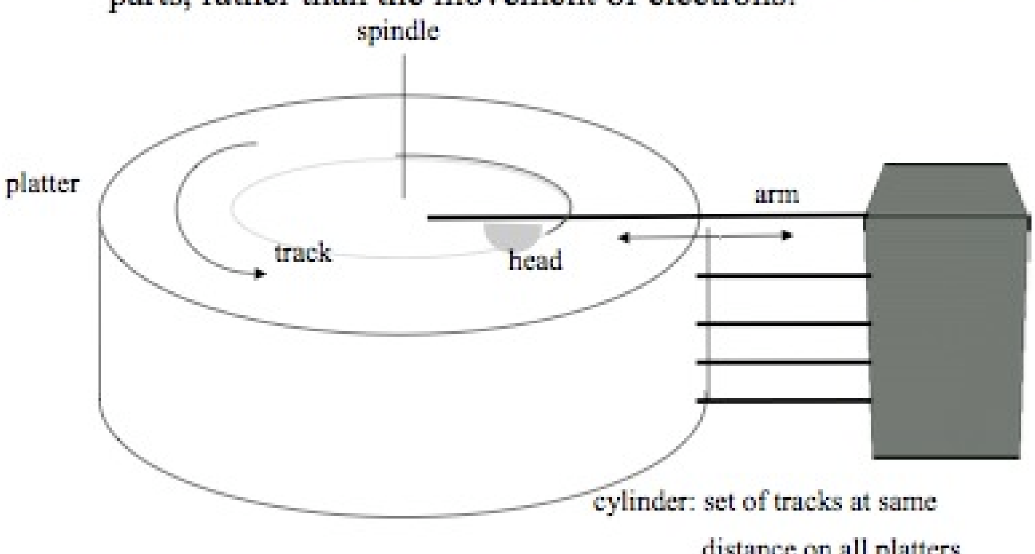


Caching and Blocking

- 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? ~ 80,000,000 phone numbers.

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

  → Really, really, really bad!
- As **balanced, binary search tree**:
  - $O(\log_2 n)$ time and $O(\log_2 n)$ 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: How to Minimize Disk Accesses

- Find **upper bound** b on node size:

  A node should not be larger than a block on disk

  → New node can always be read in one access
- Find **lower bound** a on node size:

  The lower bound should be as large as possible (remember that the fusion of two nodes to resolve underflow must result in a legal node)

  → Make best possible use of one access

  → $a = \lceil b/2 \rceil$
- This brings us to a special case of (a,b)-trees called **B-trees**

B-Trees: Definition

- **Definition:** A **B-tree** of order d is an (a,b)-tree such that

  $$a = \lceil d/2 \rceil \text{ and } b = d$$

  Examples:
  - A (2,3)-tree is a B-tree of order 3,
  - A (2,4)-tree is a B-tree of order 4,
  - A (3,5)-tree is a B-tree of order 5, etc.
- **Definition of** d**:** Given a disk, choose d such that

  d-1 key-item pairs together with d references to children

  fit into a single disk block
- **Observations:**
  - Every node in a B-tree is at least half full
  - Underflow with fusion in a B-tree always creates a node that is full and thus occupies exactly one block

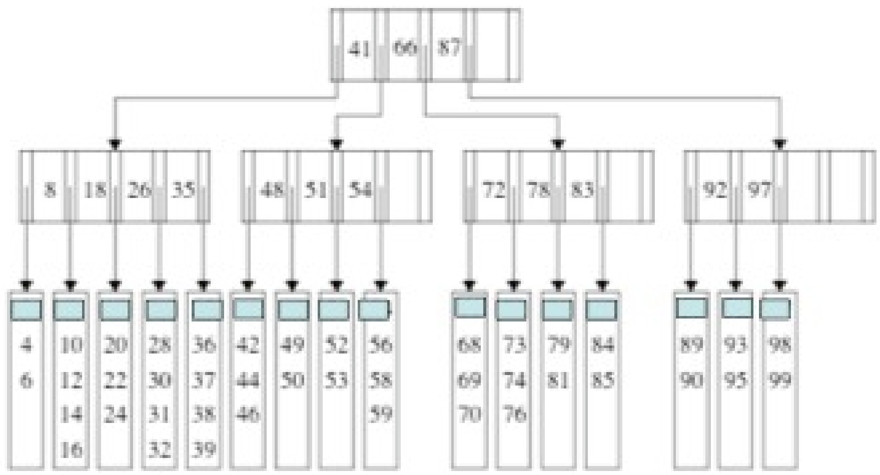
B-Trees: Complexity

- Let n be number of items stored
- Let d-1 be the number of items that fit into a disk block
- **Height:**
  $O(\log_{d/2} n)$, because height of (a,b)-tree is $O(\log_a n)$
- **Run-time of each operation:**
  $O(f(d) \cdot \log_{d/2} n)$, where $f(d)$ is the time to process a single node.
- **Number of disk accesses needed for each operation:**
  $O(\log_{d/2} n)$, at most one access for each level in tree
- Thus, the larger d is, the better the constant by which we reduce the number of disk accesses compared to the (2,4)-tree implementation
- Remember that disk accesses typically are very costly, so even improvements by constant factors are a big win
- With d = 200 and n = 80,000,000 the number of disk accesses is at most 6.

B-Tree

- **Definition:** A **B-tree** of order d is an (a,b)-tree such that
  $$a = \lceil d/2 \rceil \text{ and } b = d$$
  Examples:
  - A (2,3)-tree is a B-tree of order 3,
  - A (2,4)-tree is a B-tree of order 4,
  - A (3,5)-tree is a B-tree of order 5, etc.
- **Definition of** d**:** Given a disk, choose d such that
  d-1 key-item pairs together with d references to children
  fit into a single disk block
- **Observations:**
  - Every node in a B-tree is at least half full
  - Underflow with fusion in a B-tree always creates a node that is full and thus occupies exactly one block

B Tree



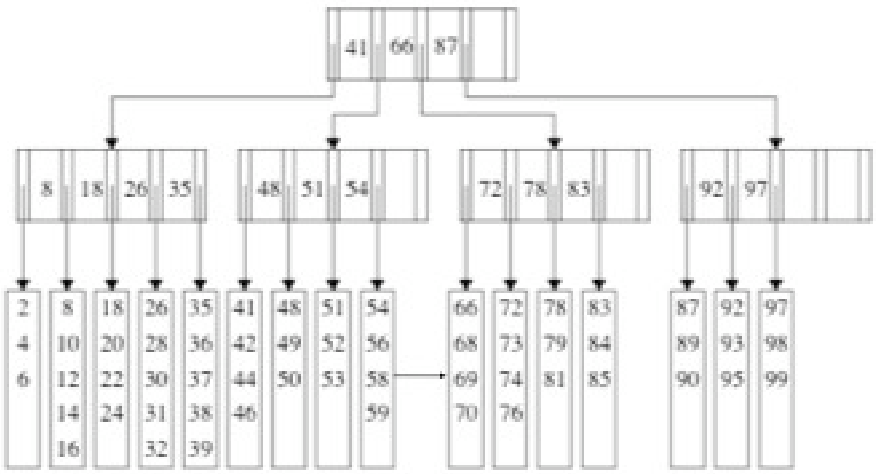
B$^+$ Tree (from text)

A B-tree of order M is an M-ary tree with the following properties:

1. The data items are stored at leaves.
2. The nonleaf nodes store up to $M - 1$ keys to guide the searching; key i represents the smallest key in subtree $i + 1$.
3. The root is either a leaf or has between 2 and M children.
4. All nonleaf nodes (except the root) have between $\lceil M/2 \rceil$ and M children.
5. All leaves are at the same depth and have between $\lceil L/2 \rceil$ and L children, for some L.

Note: B+ trees are often defined with the leaves linked
So that the entire Dictionary can be accessed
easily in order.

B^+ Tree



Note: Although all leaves are linked only one arrow is shown.