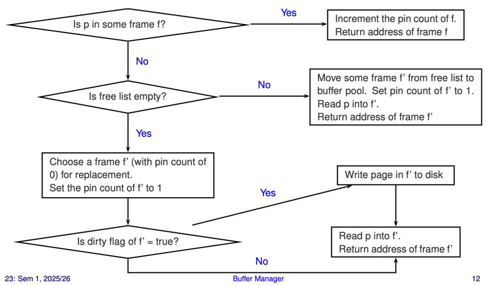
Storage

- Parts of disk
  - Platter has 2 surfaces
  - Surface has many tracks
  - Each track is broken up into sectors
  - Cylinder is the same tracks across all surfaces
  - Block comprises of multiple sectors
- **Disk Access Time** - Seek time + Rotational Delay + Transfer Time
  - **Seek Time** - Move arms to position disk head
  - **Rotational Delay** - $\frac{1}{2}\frac{60}{\text{RPM}}$
  - **Transfer time**(for n sectors) - $n \times \frac{\text{time for 1 revolution}}{\text{sectors per track}}$
    - n is requested sectors on track
- Access Order
  1. Contiguous Blocks within same track (same surface)
  2. Cylinder track within same cylinder
  3. next cylinder

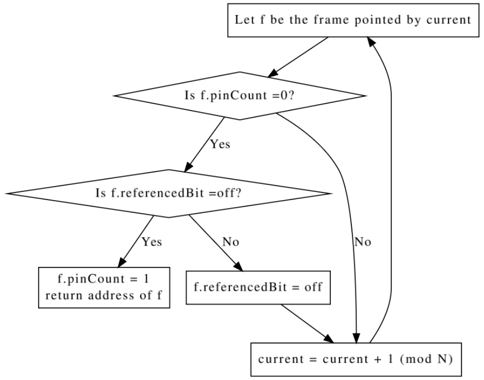
Buffer Manager



- Data stored in block sized pages called frames
- Each frame maintains pin count(PC) and dirty flag

Replacement Policies

- Decide which unpinned page to replace
- **LRU** - queue of pointers to frames with PC = 0
- **clock** - LRU variant
  - **Reference bit** - turns on when PC = 0
  - Replace a page when ref bit off and PC = 0



Files

- Heap File Implementation
  - Linked List
    - 2 linked lists, 1 of free pages, 1 of data pages
  - Page Directory Implementation
    - Directory structure, 1 entry per page.
    - to insert, scan directory to find page with space to store record

Page Formats

- **RID** = (page id, slot number)
- Fixed Length records
  - Packed Organization: Store records in contiguous slots (requires swapping last item to deleted location during deletion)
  - Unpacked organization: Use bit array to maintain free slots
- **Variable Length Records**: Slotted page organization

Record Formats

- Fixed Length Records: Stored consecutively
- Variable length Records
  - Delimit fields with special symbols (F1, $, F2 $, F3)
  - Array of field offsets $(o_1, o_2, o_3, F1, F2, F3)$

Data Entry Formats

1. $k*$ is an actual data record (with search key value k)
2. $k*$ is of the form **(k, rid)**
3. $k*$ is of the form **(k, rid-list)** list of rids of data with key k

B+ Tree index

- **Search key** is sequence of k data attributes $k \geq 1$
- **Composite search key** if $k > 1$
- **unique key** if search key contains *candidate* key of table
- index is stored as file
- **Clustered index** - Ordering of data is same as data entries
  - key is known as **clustering key**
  - Format 1 index is clustered index (Assume format 2 and 3 to be unclustered)

Tree based Index

- **root node** at level 0
- Height of tree = no of levels of internal node
- **Leaf nodes**
  - level h, where h is height of tree
- **internal nodes** store entries in form $(p_0, k_1, p_1, k_2, p_2, \ldots, p_n)$
  - $k_1 < k_2 < \ldots < k_n$
  - p_i = disk page address
- **Order** of index tree
  - Each non-root node has $m \in [d, 2d]$ entries
  - Root node has $m \in [1, 2d]$ entries
- **Equality search**: At each *internal* node N, find largest key k_i in N, such that $k_i \leq k$
  - if k_i exists, go subtree p_i, else p_0
- **Range search**: First matching record, and traverse doubly linked list
- **Min nodes at level** i is $2 \times (d+1)^{i-1}, i \geq 1$
- **Max nodes at level** i is $(2d+1)^i$

Operations (Right sibling first, then left)

Insertion

1. **Leaf node Overflow**
   - Redistribute and then split
   - **Split** - Create a new leaf N with $d+1$ entries. Create a new index entry $(k, \blacksquare)$ where k is smallest key in N
   - **Redistribute** - If sibling is not full, take from it. If given right, update right's parent pointer, else current node's parent pointer
2. **Internal node Overflow**

- Node has $2d+1$ keys.
- Push middle $(d+1)$-th key up to parent.

Deletion

1. **Leaf node**
   - Redistribute then merge
   - **Redistribution**
     - Sibling must have $> d$ recordsto borrow
     - Update parent pointers to right sibling's smallest key)
   - **Merge**
     - If sibling has d entries, then merge
     - Combine with sibling, and then remove parent node
2. **Internal Node Underflow**
   - Let N' be adjacent *sibling* node of N with $l, l > d$ entries
   - Insert $(K, N'.p_i)$ into N, where i is the leftmost(0) or rightmost entry(l)
   - Replace K in parent node with $N'.k_i$
   - Remove (p_i, k_i) entry from N'

Bulk Loading

1. Sort entries by search keys.
2. Load leaf pages with $2d$ entries
3. For each leaf page, insert index entry to rightmost parent page

Hash based Index

Static Hashing

Linear Hashing

Extensible Hashing