Example 1: Calculating Disk Addressing Requirements

A disk system has the following characteristics:

- **Surfaces** = 16, **Cylinders** = 10240, **Blocks per Track** = 64, **Block Size** = 8 KB

(a) How many bytes are required to store a block address?

- Assume a **fixed** number of blocks per track.
- Identify the number of bits required to address a **cylinder**, **surface**, and **block within a track**.

(b) How many bytes are required for a record address?

- The record address includes a **block address** and a **byte offset within a block**.
- Given the **block size of 8 KB**, determine how many bits are needed for the offset.

Example 2: Disk Addressing with Variable Sectors Per Track

A disk is organized as follows:

- **Number of Platters** = 10, **Number of Cylinders** = 5000, **Tracks per Surface** = 5000
- **Blocks per Track** = Variable (range: 20 - 40), **Block Size** = 2 KB

(a) What is the minimum and maximum number of bytes needed for a block address?

- Consider the **worst case** (maximum blocks per track) and **best case** (minimum blocks per track).
- Assume blocks are evenly distributed across all platters.

(b) How many bytes are required for a record address?

- Compute the **byte offset** based on the **block size of 2 KB**.
- Determine the **total number of bytes** required.

Example 3: Large-Scale Storage Disk Addressing

A high-capacity disk has the following parameters:

- **Surfaces** = 32, **Cylinders** = 65536, **Blocks per Track** = 128, **Block Size** = 16 KB

(a) How many bytes are required for a block address?

- Calculate the **number of bits needed** for each of the three components:
  1. **Cylinder selection**
  2. **Surface selection**
  3. **Block within the track**

(b) How many bytes are required for a record address?

- Determine the **byte offset** required within a **16 KB block**.
- Compute the **total size of the record address**.

Question 2
Suppose that if we swizzle all pointers automatically, we can perform the swizzling in half the time it would take to swizzle each one separately. If the probability that a pointer in main memory will be followed at least once is p, for what values of p is it more efficient to swizzle automatically than on demand?

Question 1: Pointer Swizzling Efficiency
Suppose that automatic pointer swizzling takes 60% of the time required for manual swizzling. If the probability that a pointer in main memory will be followed at least once is **p**, for what values of **p** is automatic swizzling more efficient than on-demand swizzling?

Question 2: Trade-off in Pointer Swizzling
Assume that when a pointer is swizzled on demand, it incurs an overhead of **T_d**, whereas automatic swizzling incurs an overhead of **T_a**, where **T_a = 0.7 * T_d**. If the probability that a pointer will be followed is **p**, determine the condition for which automatic swizzling is more efficient than on-demand swizzling.

Question 3: Optimizing Pointer Swizzling Strategy
In a database system, automatic pointer swizzling is performed at a **fixed cost** per batch, whereas on-demand swizzling incurs a **variable cost** depending on how often pointers are accessed. If a pointer is accessed with probability **p**, derive the inequality that determines when automatic swizzling is the better choice.

📘 DBMS 5 – Addressing, TID, and Pointer Swizzling

1. Bits and Addressing Basics

- **n bits** can represent 2^n values.
- To represent MM elements, need at least $\lceil \log_2(M) \rceil$.
- Example: $M=1321 \Rightarrow \lceil \log_2(1321) \rceil = 11$.
- Used to calculate storage needed for **disk addresses, block IDs, record IDs**.

---

2. Disk Addressing

- To uniquely identify a block, need:
  1. **Surface** (disk side)
  2. **Cylinder/track**
  3. **Block within track**
- Example (Disk 1):
  - Surfaces = 8 → need 3 bits
  - Cylinders = 8192 → need 13 bits
  - Blocks/track = 32 → need 5 bits
  - Total = 21 bits → rounded to **4 bytes** for block address.
- To form a **record address**, also include byte offset within block.
  - For 4096-byte block: need 12 bits → rounded to 2 bytes.
  - Total record address size = **6 bytes**.

---

3. Addressing Methods

- **Physical Addressing:**
  - Directly stores record location (fast, 1 I/O).
  - Problem: if record moves/deletes → dangling pointers. Needs *tombstones*.
- **Logical Addressing:**
  - Each record has logical ID; a **map table** translates it to physical address.
  - Pro: only map table updated if record moves/deletes.
  - Con: slower (extra lookup → 2 I/Os).

---

4. Deletions & Insertions

- **Deletion:**
  - Physical → mark with tombstone in record header (space reusable, but tombstone stays).
  - Logical → map entry updated to tombstone; record space can be reclaimed.
- **Insertion:**
  - If records not ordered: append at end or reuse deleted slots.
  - If ordered: try nearby free space; else use overflow blocks.

---

5. Tuple Identifier (TID)

- A compromise between physical and logical.
- **TID = (Block number, Tuple index within block).**
- **Block header** contains an offset table mapping tuple indexes → record positions.
- Benefits:
  - Stable within block: only offset table updated if record moves.
  - If record moves across blocks: original block holds a *forwarding address* with new TID.
- Performance: access usually in 1 I/O; at most 2 if record relocated.

---

6. Pointer Swizzling

- Problem: records in memory vs. disk have different addresses.
- **Swizzling = translating DB address → in-memory pointer** when loading a block.

Options:

1. **Never swizzle:** keep translation table; always look up. (Slow pointer chasing).
2. **Automatic swizzle:** translate all pointers immediately when block is loaded. (Fast later, but high upfront cost).
3. **On-demand swizzle:** translate only when pointer followed first time. (Balanced, but needs extra check per pointer).

- **Efficiency condition:**
  - If probability a pointer will be used ≥ 0.5, automatic swizzling is cheaper.
  - Otherwise, on-demand is better.
- **Note:** Pointers must be **unswizzled** before writing block back to disk.

✅ Key Takeaways

- **Bit representation** determines how many bytes needed for addresses.
- **Physical vs. logical addressing:** speed vs. flexibility trade-off.
- **TID**: hybrid addressing ensuring stable tuple IDs.
- **Swizzling:** improves in-memory pointer efficiency; choice depends on usage probability.
- DBMS carefully balances **I/O cost, flexibility, and pointer stability** when managing record addresses.
