RAID 0: No Redundancy

RAID 1: MIRRORED (2 IDENTICAL COPIES)
- Parallel reads
- Writes to 2 disks
- Maximum transfer rate = transfer rate of 1 disk.

RAID 0+1: STRIPING AND MIRRERING
- Parallel reads
- Writes to two disks
- Maximum transfer = aggregate bandwidth (for several blocks)

RAID 2: ERROR CORRECTING CODES
- Striping unit is a single bit

IDEA: 1001
\[ \text{0101} \quad e \text{-bit check} \]

REALLY: HAMMING CODES
- Hamming (7,4) has 3 parity bits for 4 data bits
  \( \rightarrow \) detect and correct 1 bit error
- 4 DATA DISKS, 3 CHECK DISKS

RAID 3: BIT INTERLEAVED PAIRITY
- Single check disk with parity
- Check scheme can detect which failed, but not which
- STRIPING UNIT IS 1 BIT
- READ, WRITE INVOLVES ALL DISKS
- CAN PROCESS ONLY REQUEST AT A TIME
RAID 4: BLOCK INTERLEAVED PARITY
- STRIPING UNIT: ONE DISK BLOCK
  ONE CHECK DISK
- LARGE REQUEST CAN UTILIZE FULL AGGREGATE BANDWIDTH
- WRITES TO MODIFIED DISK AND CHECK DISK

RAID 5: BLOCK INTERLEAVED, DISTRIBUTED PARITY
- SIMILAR TO LEVEL 4, BUT PARITY BLOCKS ARE DISTRIBUTED
- BEST PERFORMANCE OF ALL RAID FOR SMALL/LARGE READ & LARGE WRITE
- SMALL WRITES REQUIRE READ-MODIFY-WRITE CYCLES

WHICH RAID?
- DEPENDS ON CONCERN FOR DATA LOSS AND WORKLOAD.
  RAID 5 IS GOOD "GENERIC"
  RAID 0 IS GOOD IF NOT WORRIED FOR LOSS
  \[ 2 < 3, 4 < 5 \]

III DISK SPACE MANAGEMENT
- ALLOCATE/DEALLOCATE PAGE
- READ/WRITE PAGE
- DISK SPACE MANAGER

\[ \text{Page} = \text{UNIT OF DATA} \]
\[ \text{A BLOCK SIZE (ASSUMED)} \]
- REQUESTS FOR SEQUENCIAL PAGES (E.G. TO EXPLOIT SEQUENTIAL DISK BLOCK ACCESS)
  MUST BE HANDLED BY MANAGER

TRACK FREE BLOCKS
- MAINTAIN LIST OF "FREE" BLOCKS
- MAINTAIN BIT MAP OF FREE BLOCKS

WHY NOT OS?
- FOR OS, FILE IS SEQUENCE OF BYTES
  - DB MAY WANT DIFFERENT FEATURES
  - OS FILE SIZE MAY BE TOO SMALL
    - E.G. 52 BIT SYSTEM, LOGGED FILE IS 4 KB
  - OS FILES CAN'T SPAN MULTIPLE DISK DEVICES

IV BUFFER MANAGER

TABLE OF <FRAME#, PAGE#> PAIR IS MAINTAINED

WHEN A PAGE IS REQUESTED:
- IF PAGE IS NOT IN THE POOL:
  - CHOOSE A FRAME FOR REPLACEMENT
    - IF FRAME IS DIRTY, WRITE TO DISK
    - READ REQUESTED PAGE INTO FRAME
- PIN THE PAGE AND RETURN ADDRESS

\[ \text{pin count} = \text{how many times the page has been requested, not released} \]
\[ \text{dirty} = \text{if the page has been modified} \]
- PAGES CAN BE PRE-FETCHED IF YOU CAN PREDICT WORK LOAD
- Requester must "unpin"
- Concurrency control and recovery adds additional steps (Write Ahead Log)
- Replacement policies:
  LRU - Least Recently Used
  FIFO - First in, first out
  MRU - Most Recently Used
  Clock - Variation of LRU, see homework

Big impact:
- Example LRU and sequential scans

Why not O.S.?
- Portability
- Limitations
- Access to workload
- Need to pin and force page to disk for CC and recovery

II. Record Formats

II.1 Fixed length records

- "Packed"

- "Unpacked"

Record ID = \( <\text{page id}, \text{slot id}> \)
First approach can change RIP.

II.2 Variable length

VII. Heap Files

- FILE = COLLECTION OF PAGES = COLLEZION OF RECORDS
- DBMS operates on files and records
- Need to:
  - Insert/delete/modify record
  - Read a particular record
  - Scan all records
Unordered (Heap) Files
- simplest file structure, no order
  - as file grows, shrink, disk pages are allocated, de-allocated
- Need to:
  - track pages in a file
  - track free space on pages
  - track records on a page
- Many approaches/alternatives

Heap file is linked list!
- Store header page id & heap file name somewhere
- Each page has 2 pointers
- Problem: for variable length records, every page will have free space

Heap file as Page Directory
- Each directory entry = sequence of pages
- Use bit for free space and count of available space
- Can quickly search for pages with enough free space