Database Systems

File Organization1

A.R. Hurson

323 CS Building

Missouri Science and Technology

Database Systems

- Storage Medium
  - The collection of digital information must be physically stored on some storage medium, either during the course of operations or for the future use.
  - The database management system (DBMS) can then retrieve, update, and process the stored information.

Database Systems

- Storage Medium
  - Storage medium: Physical entity to be used to store and retrieve digital information.
  - Memory Requirements for a Computer
    - An internal storage medium to store the intermediate as well as the final results,
    - An external storage medium to store input information, and
    - An external storage medium to store permanent results for future

Database Systems

- Storage Medium
  - Different parameters can be used in order to classify the memory systems.
  - In the following we will use the access mode in order to classify memory systems
  - Access mode is defined as the way the information stored in the memory is accessed.

Database Systems

- Storage Medium — Access Mode
  - Address Accessible Memory: Where information is accessed by its address in the memory space.
  - Content Addressable Memory: Where information is accessed by its contents (or partial contents).

Database Systems

- Storage Medium — Access Mode
  - Within the scope of address accessible memory we can distinguish several sub-classes;
    - Random Access Memory (RAM): Access time is independent of the location of the information.
    - Sequential Access Memory (SAM): Access time is a function of the location of the information.
    - Direct Access Memory (DAM): Access time is partially independent of and partially dependent on the location of the information.

- Storage Medium
  - Even within each subclass, we can distinguish several sub subclasses.

Database Systems

- Storage Medium — RAM
  - Random access memory can also be grouped into different classes
    - Read Only Memory (ROM)
      - Programmable ROM
      - Erasable Programmable ROM (EPROM)
      - Electrically Alterable ROM (EAROM)
      - Flash Memory

Database Systems

- Storage Medium — RAM
  - Read/Write Memory (RWM)
    - Static RAM (SRAM)
    - Dynamic RAM (DRAM)
      - Synchronous DRAM
      - Double-Data-Rate SDRAM
  - Volatile/Non-Volatile Memory
  - Destructive/Non-Destructive Read Memory

Database Systems

- Storage Medium — DAM
  - Within the scope of Direct Access Memory we can recognize different groups:
    - Movable head disk,
    - Fixed head disk,
    - Parallel disk

Database Systems

- Storage Medium — DAM
  - Movable head disk: Each surface has just one read/write head. To initiate a read or write, the read/write head should be positioned on the right track first — seek time.
  - Seek time is a mechanical movement and hence, relatively, very slow and time consuming.

Database Systems

- Storage Medium — DAM
  - Fixed head disk: Each track has its own read/write head. This eliminates the seek time. However, this performance improvement comes at the expense of cost.

Database Systems

- Storage Medium — DAM
  - Parallel disk: To respond the growth in performance and capacity of semiconductor, secondary storage technology, introduced RAID — Redundant Array of Inexpensive Disks.
  - In short RAID is a large array of small independent disks acting as a single high performance logical disk.

Database Systems

- Storage Medium
  - Within the scope of the memory system, the goal is to design and build a system with low cost per bit, high speed, and high capacity. In other words, in the design of a memory system we want to:
    - Match the rate of the information access with the processor speed.
    - Attain adequate performance at a reasonable cost.

Database Systems

- Storage Medium
  - The appearance of a variety of hardware as well as software solutions represents the fact that in the worst cases the trade-off between cost, speed, and capacity can be made more attractive by combining different hardware systems coupled with special features — memory hierarchy.

Database Systems

- We will distinguish three levels in this hierarchical organization;
  - Primary Storage: This include storage medium that is directly accessible by the central processing unit (CPU) — Main memory, cache, …
  - Secondary Storage: This includes disk, flash memory, I/O caches, … This class of storage medium are mainly larger in capacity, lower in cost per bit, and slower than primary memories.
  - Tertiary Storage: Optical disks, tapes, … are removable offline storage for archiving information.

Database Systems

- Databases typically store large amount of persistent data over long period of time. This is in contrast with the notion of data structures that persist for a short period of time.

Database Systems

- Most databases are stored permanently on disk;
  - Generally, databases are too large to fit entirely in main memory (size gap) — Main memory database projects violate this.
  - Disk (non-volatile storage) are more reliable than main memory (volatile storage).
  - The cost of storage per unit of data is an order of magnitude less for disk than for primary storage .

Database Systems

- Hardware Description of Disk devices
  - Bit: Smallest granule of data.
  - Byte: Groups of bits, typically 4 to 8 bits — Character.
  - Capacity: Number of bytes.
  - Single-sided disk: A disk that stores information only on one of its surface.
  - Double-sided disk: A disk that stores information on both surfaces.
  - Disk pack: An assembly of several disks — as many as 30 surfaces.

Database Systems

- Hardware Description of Disk devices
  - Information on disk is stored in concentric circles called track. Tracks, regardless of their diameters, store the same amount of information — density increases for inner tracks than outer tracks.
  - In a disk pack, tracks on different surfaces with the same diameter are called a cylinder.

Database Systems

- Hardware Description of Disk devices
  - RAID increases the performance and reliability.
    - Data Striping
    - Redundancy

Database Systems

- RAID
  - Concept of data striping (distributing data transparently over multiple disks) is used to allow parallel access to the data and hence to improve disk performance.
  - In data striping, the data set is partitioned into equal size segments, and segments are distributed over multiple disks.
  - The size of segment is called the striping unit.

Database Systems

- RAID
  - Redundant information allows reconstruction of data if a disk fails. There are two choices to store redundant data:
    - Store redundant information on a small number of separate disks — check disks.
    - Distribute the redundant information uniformly over all disks.
  - Redundant information can be an exact duplicate of the data or we can use a Parity scheme — additional information that can be used to recover from failure of any one disk in the array.

Database Systems

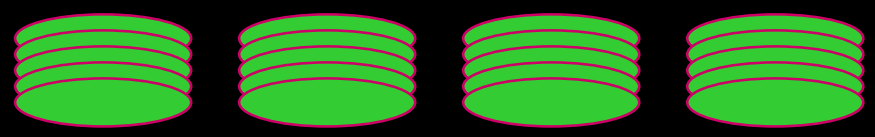
- RAID
  - The disk array can be partitioned into reliability group consists of a set of data disks and a set of check disks. Then a common redundancy scheme is applied to each group.
  - In the following discussion we assume a RAID of four disk packs.

Database Systems

- RAID Levels of Redundancy
  - Level 0: Striping without redundancy
    - Offers the best write performance — no redundant data is being updated.
    - Offers the highest Space utilization.
    - Does not offer the best read performance.

Database Systems

- RAID — Level 0



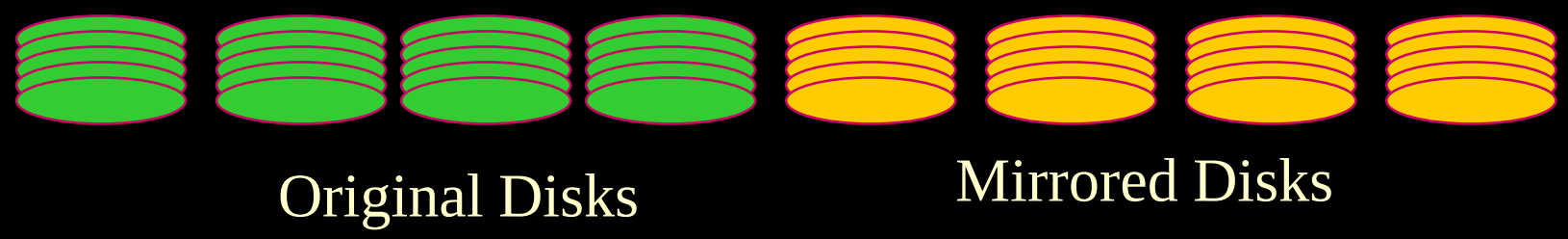
Non-redundant RAID

Database Systems

- RAID Levels of Redundancy
  - Level 1: Mirrored — Two identical copies
    - Each disk has a mirror image (Shadow).
    - Is the most expensive solution — space utilization is the lowest.
    - Parallel reads are allowed.
    - Write involves two disks, in some cases this will be done in sequence.
    - Maximum transfer rate is equal to the transfer rate of one disk — No striping.

Database Systems

- RAID — Level 1



Mirrored

Database Systems

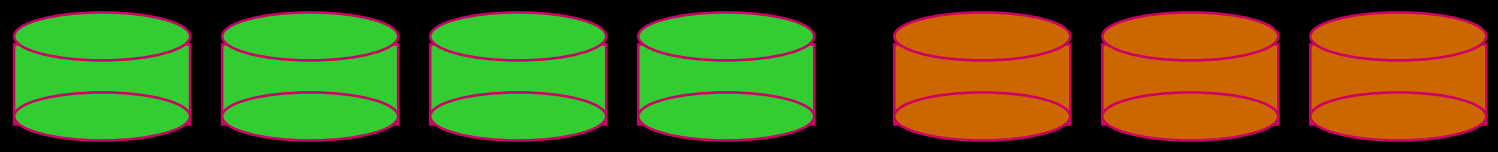
- RAID Levels of Redundancy
  - Level 0 + 1: Striping and Mirroring
    - Parallel reads are allowed.
    - Space utilization is the same as level 1.
    - Write involves two disks and cost of write is the same as Level 1.
    - Maximum transfer rate is equal to the aggregate bandwidth of the disks.

Database Systems

- RAID Levels of Redundancy
  - Level 2: Error Correcting Codes
    - The striping unit is a single bit.
    - Hamming coding is used as a redundancy scheme.
    - Space utilization increases as the number of data disks increases.
    - Maximum transfer rate is equal to the aggregate bandwidth of the disks — Read is very efficient for large requests and is bad for small requests of the size of an individual block.

Database Systems

- RAID — Level 2



Data Disks

Coded Information

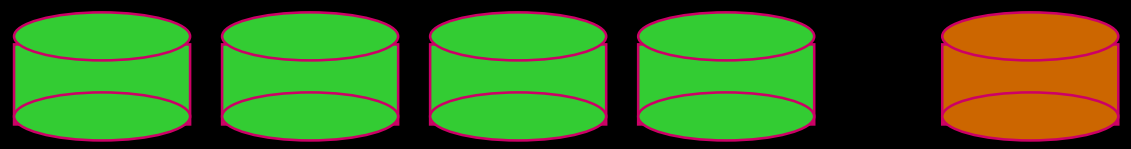
Error Correcting Codes

Database Systems

- RAID Levels of Redundancy
  - Level 3: Bit Interleaved Parity
    - The striping unit is a single bit.
    - Unlike the level 2, the check disk is just a single parity disk, and hence it offers a higher space utilization than level 2.
    - Write protocol is similar to level 2 — read-modify-write cycle.
    - Similar to level 2, can process one I/O at a time — each read and write request involves all disks.

Database Systems

- RAID — Level 3



Data Disks Parity Disk

Bit Interleaved Parity

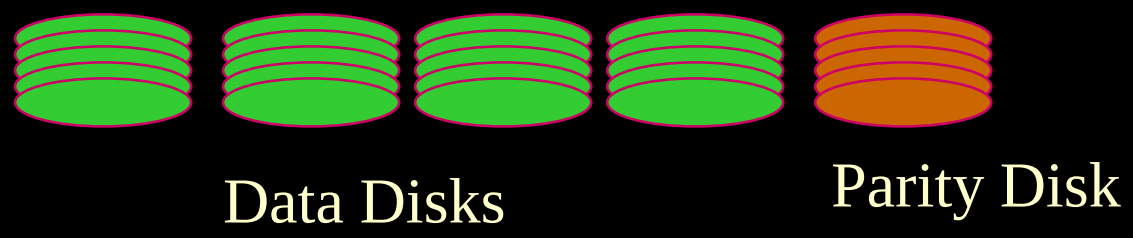
Database Systems

- RAID Levels of Redundancy
  - Level 4: Block Interleaved Parity
    - The striping unit is a disk block.
    - Read requests of the size of a single block can be served just by one disk.
    - Parallel reads are possible for small requests, large requests can utilize full bandwidth.
    - Write involves modified block and check disk

      NewParity = (OldData XOR NewData) XOR OldParity
    - Space utilization increases with the number of data disks.

Database Systems

- RAID — Level 4



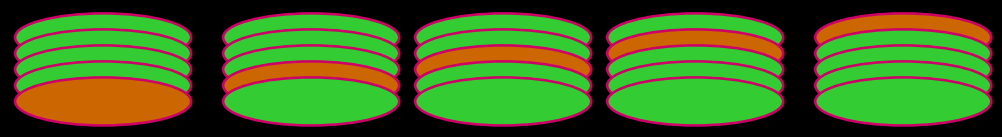
Block Interleaved Parity

Database Systems

- RAID Levels of Redundancy
  - Level 5: Block Interleaved Distributed Parity
    - Parity blocks are uniformly distributed among all disks. This eliminates the bottleneck at the check disk.
    - Several writes can be potentially done in parallel.
    - Read requests have a higher level of parallelism.

Database Systems

- RAID — Level 5



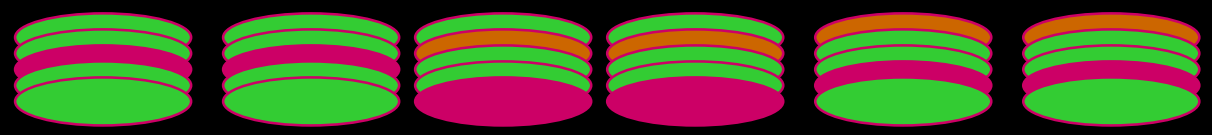
Block Interleaved Distributed Parity

Database Systems

- RAID Levels of Redundancy
  - Level 6: P+Q Redundancy
    - Can tolerate higher level of failure than level 2.
    - It requires two check disks and similar to level 5, redundant blocks are uniformly distributed at the block level over the disks.
    - For small and large read requests, and large write requests, the performance is similar to level 5.
    - For small write requests, it behaves the same as level 2.

Database Systems

- RAID — Level 6



P+Q Redundancy

Database Systems

Type	Striping	Redundancy	Space	Read	Write
RAID 0	Bit or block	No	100%	Parallel	Fastest
RAID 1	No	Mirroring	50%	Several reads	One write at a time
RAID 10	Bit or block	Mirroring	50%	Same as Raid 0	Same as Raid 1
RAID 2	Single Bit	Hamming code	57%	Bad for small #of blocks Good for large #of blocks	All disks are involved
RAID 3	Single Bit	Parity	80%	Same as Raid 2	Same as Raid 2
RAID 4	Block	Parity	80%	Overlapping several reads	No overlapping, one data and parity disks are involved
RAID 5	Block	Parity	80%	Higher than Raid 4	Overlapping several writes
RAID 6	Block	Reed-Solomon	66%	Higher than Raid 5	Same as Raid 5

Database Systems

- Hardware Description of Disk devices
  - A block is the unit of information communicated between disk and I/O buffer.
  - A block address is a combination of surface number, track number, and block number (within the track).

Database Systems

Time to have the beginning of the block under read/write head

- Access time = Seek time + rotational time (latency) + transfer time

Time to position read/write head on the current track

Time to transfer data

Database Systems

- Access time
  - Seek time and rotational time are the dominating factors:
    - Seek time varies from 1 to 20 msec.
    - Rotational time varies from 0 to 10 msec
    - Transfer time is about 1 msec for 4KB page.
  - Hence, reducing the seek time and rotational time are the key to lower I/O cost.

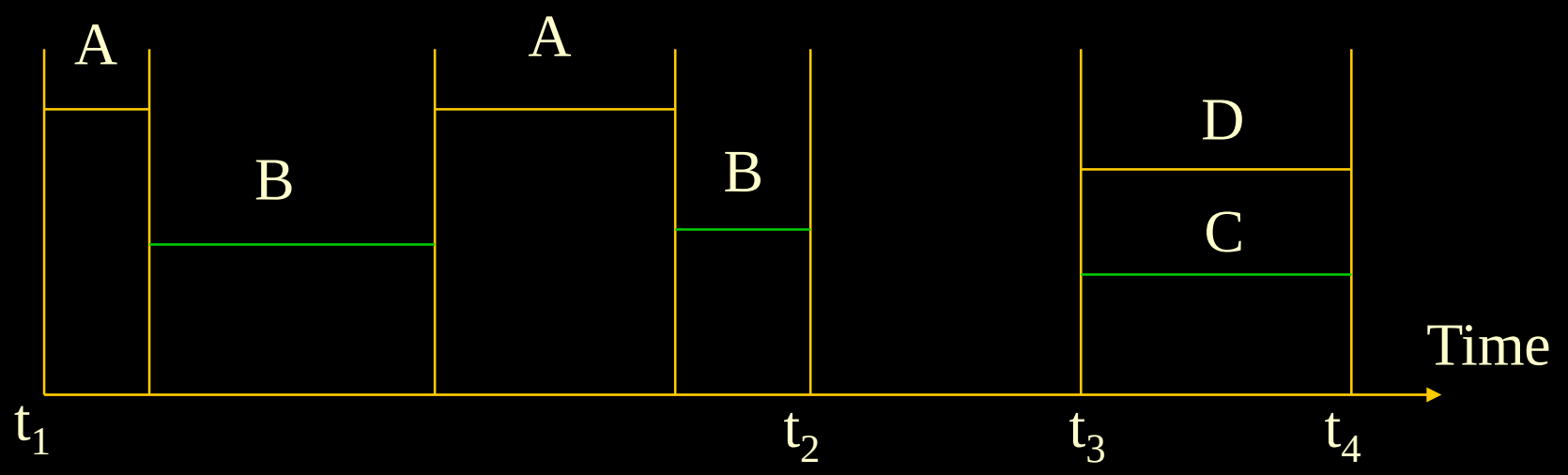
Database Systems

- Access time — How to reduce it?
  - ‘Next’ block concept:
    - blocks on the same track, followed by
    - blocks on same cylinder, followed by
    - blocks on adjacent cylinder
  - Blocks in a file should be arranged sequentially on disk based on the ‘Next’ concept, to reduce seek and rotational time.
  - In case of sequential scan, pre-fetch several blocks at a time.

Database Systems

- Hardware Description of Disk devices
  - Buffering of Blocks: When several blocks need to be transferred from disk to main memory and we know all the data addresses beforehand, several buffers in main memory can be used to speed up the transfer.
  - This allows either interleaving I/O operations with CPU operations, or simultaneous execution of I/O operations with CPU operations.

Database Systems



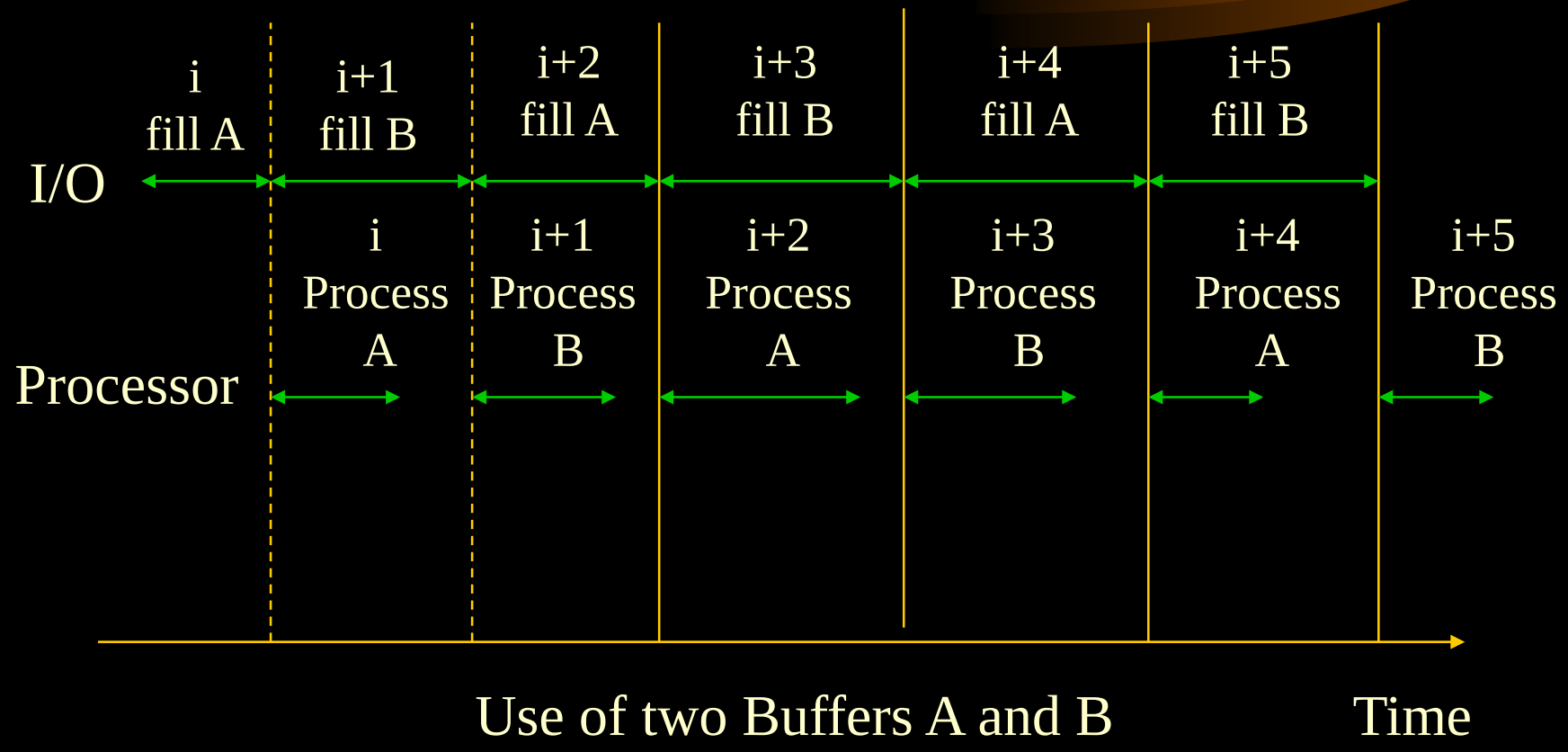
Interleaving process A with process B.

Parallel execution of processes C and D.

Database Systems

- Hardware Description of Disk devices
  - Technique such as double buffering can also be used to overlap I/O operations with CPU operations.
  - Double buffering can also be used to write a continues stream of blocks from memory to disk.

Database Systems



Database Systems

- Hardware Description of Disk devices
  - In general buffering allows us to take advantage of continuously reading or writing data on consecutive disk blocks. This reduces the seek time and rotational time, as a result, data will be available to CPU faster. Consequently, the waiting time in the program is reduced.

Database Systems

- Placement of Files on Disk
  - A file is a collection of records of the same record type (same format and definition). Data, on disk, is usually stored in the form of records.
  - A record is a collection of fields. It represents a collection of related data values identifying an entity. Each field then represents an attribute of an entity.
  - The collection of field names and their corresponding data types constitutes the record type or record format definition.

Database Systems

- In comparison to our earlier discussion about relation, then:
  - A file is equivalent to a relation,
  - A record is equivalent to a tuple,
  - A field is equivalent to an attribute,
  - Data type of each field is equivalent to the domain of each attribute.

Database Systems

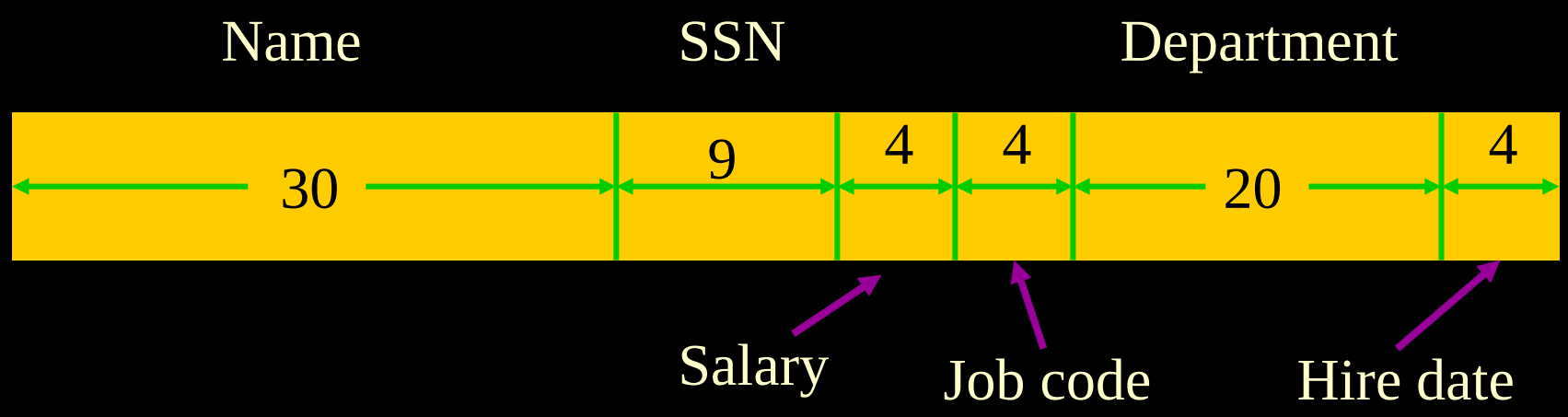
- Files could be of two types:
  - File with fixed length records: Each record in the file has exactly the same size.
  - File with variable length records: Records in the file are of different size.

Database Systems

- A file is of variable-length records, because:
  - Records are of the same record type but fields are of varying size.
  - Records are of the same record type but fields may have multiple values (repeating fields).
  - Records are of the same record type but one or more fields are optional (optional fields).
  - The file contains records of different record types (mixed file).

Database Systems

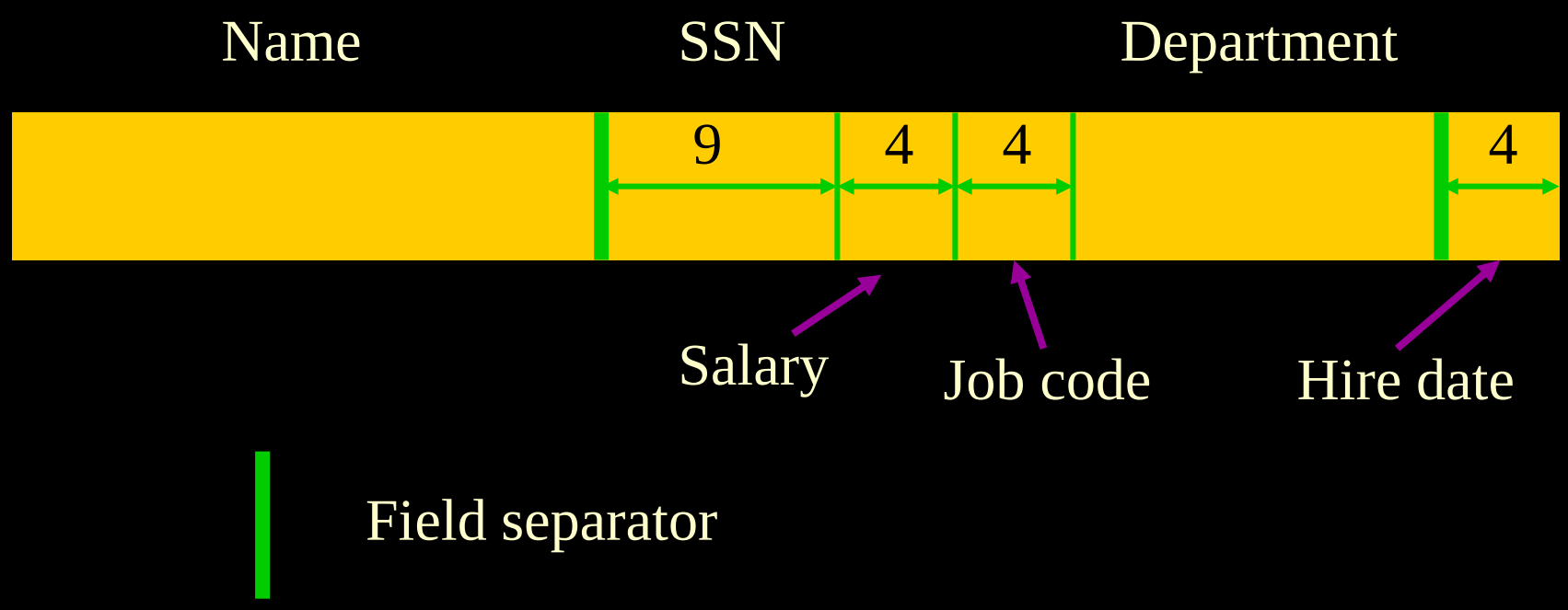
- Fixed-length record



- In a file of fixed-length records, all the records have the same fields and fields are of the same length.

Database Systems

- Variable-length record



Database Systems

- In case of files of records with optional fields when the total number of fields for the record type is large but the number of fields in each record is typically small, each field can be represented as a pair of:

<field-name, field value>

Database Systems

Name		Smith, John	SSN		123456789	Department		CSE

Field-name, field value separator

Field Separator

Record terminator

Database Systems

- A repeating field needs one separator to separate the repeating values of the field and another separator to indicate termination of the field.

- In files with different record types, each record must be preceded by a record type indicator.

Database Systems

- Record blocking, spanned and un-spanned records
  - As noted before, a block is the unit of information transferred between disk and memory. Naturally, when the block size is larger than the record size, each block may contains several records.
  - Let *B* be the block size, *R* be the record size, and B $\geq$ R, then bfr = $\lfloor B/R \rfloor$ records can be fit in a block.
  - bfr is called blocking factor.

Database Systems

- In case that B is not divisible by R then each block will have B - (bfr * R) unused bytes.
- This unused space can be used by a technique called spanning — breaking a record into two blocks — a part of a record along with a pointer is put in the unused part of one block and the rest of the record is put on the other block. The pointer points to the block that contains the rest of the record.

Database Systems

- Spanning can be used frequently when dealing with variable-length records.
- Un-spanned organization refers to the case where we do not allow records to cross block boundaries.
- For variable-length records or spanned organization, each block may contains different number of records. In this case bfr represents the average number of records per block for a file.

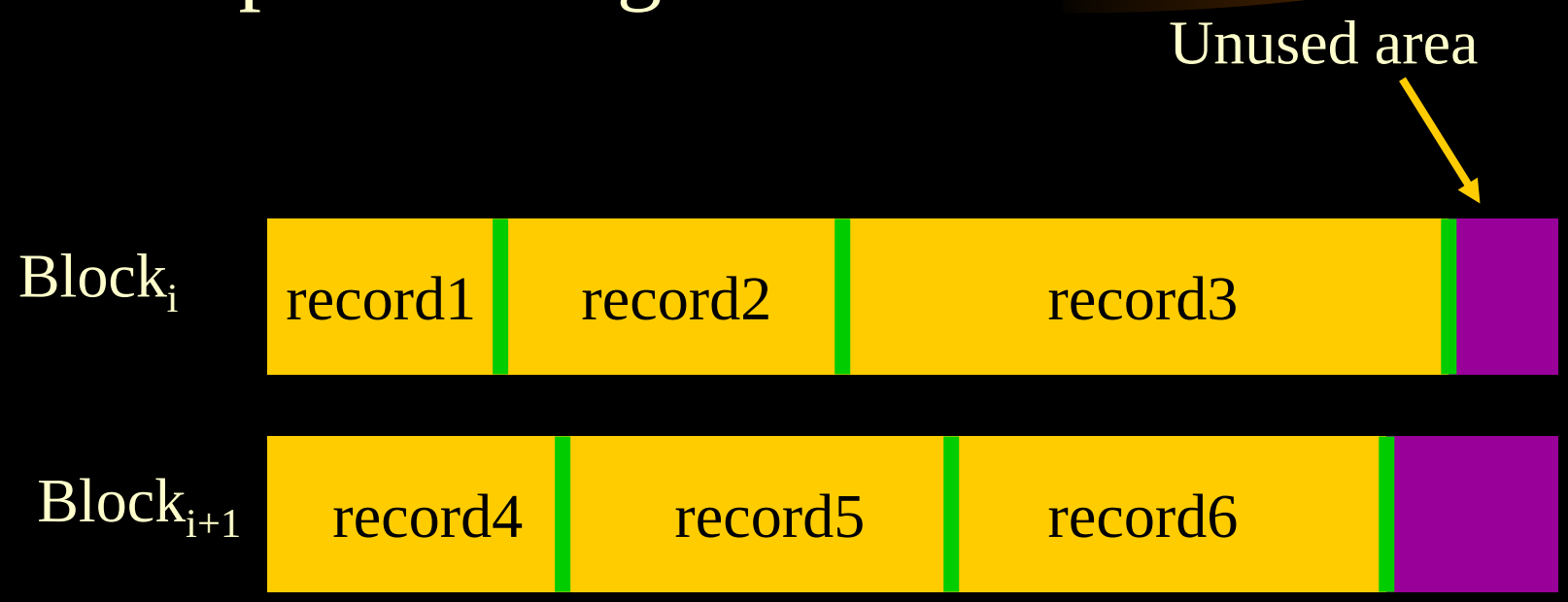
Database Systems

- Note bfr can be used to calculate the number of blocks *b* for a file of *r* records:

$$b = \lceil r / bfr \rceil$$

Database Systems

- Un-spanned Organization



Database Systems

- Spanned Organization

Database Systems

- In general, blocking is used for the following reasons:
  - Higher performance — blocking reduces the number of data transfer operations.
  - Higher storage utilization — number of inter-block gaps is reduces.

Database Systems

- Allocating File blocks on Disk
  - Contiguous allocation — file blocks are allocated to consecutive disk blocks:
    - Fast reading the whole file, specially when double buffering is used.
    - Difficult to expand the file.

Database Systems

- Allocating File blocks on Disk
  - Linked allocation — each file block contains a pointer to the next file block.
    - Easy to expand.
    - slow in reading the whole file.

Database Systems

- Allocating File blocks on Disk
  - Cluster allocation — a combination of the aforementioned schemes. Each cluster is a collection of consecutive disk blocks while clusters are linked together.
  - Indexed allocation — index blocks contain pointer to the file blocks.
  - Note, as will be discussed later, it is possible to use combinations of these schemes in an organization.

Database Systems

- Operations on files
  - Two major operations are performed on files:
    - Retrieval — contents of the file does not change,
    - Update — contents of file changes by insertion or deletion of record(s), or modification of field values.
  - For either cases, we have to use a criterion (search criterion) to distinguish the target record(s).

Database Systems

- Files of Unordered Records
  - Records are inserted in the file in the order they are inserted — new records are inserted at the end of the file. In this organization:
    - Insertion is very efficient — the last disk block is read into a buffer, the new record is added, and the block is rewritten back to disk, order of operation is O(1).
    - Search is expensive — any simple search forces a linear (exhaustive) search. On average half of the blocks need to be read in, order of operation is O(n).

Database Systems

- Files of Unordered Records
  - To delete a record, we must first search for it and locate its block. Bring the block containing the record into the buffer, delete the record, and rewrite the block back to disk. This creates unused spaces in the disk blocks — waste of space. Order of operation is still O(n). Usually, a marker (delete marker) is used to mark deleted record. Later on, we may run an algorithm (compaction algorithm) to reorganize the file by packing the records and hence reclaiming unused spaces in each block.

Database Systems

- Files of Unordered Records
  - Files could be of fixed-length records or variable-length records. Both spanned and un-spanned organization can be adopted. However, modifying a variable-length record may be problematic (why)?
  - Record *i* in an unordered, fixed-length un-spanned, contiguous file (relative file) can be accessed very easily. Say how?

Database Systems

- Files of Ordered Records
  - Records of a file on disk can be ordered based on the contents of one or several field(s) — Ordering field(s).
  - In case that the ordering field is also a key field, then the field is called the ordering key for the file.

Database Systems

- Files of Ordered Records
  - The major advantage of an ordered file comes from searching the file based on the ordering field. Binary search technique can be used to locate an specific record with a time complexity of $O(\log_2 b)$ where *b* is the number of blocks in the file.
  - Ordering does not provide any advantage for random or ordered access of the record based on the contents of non ordering field of the file.

Database Systems

- Files of Ordered Records
  - Insertion and deletion are expensive.
  - To insert a record, we first need to find its correct position in the field, then make space in the file to insert the record in that position. For a large file, this can be very time consuming. On the average, half the records of the file must be moved around.
  - Deletion is less problematic if we use deletion marker and reorganize the file periodically.

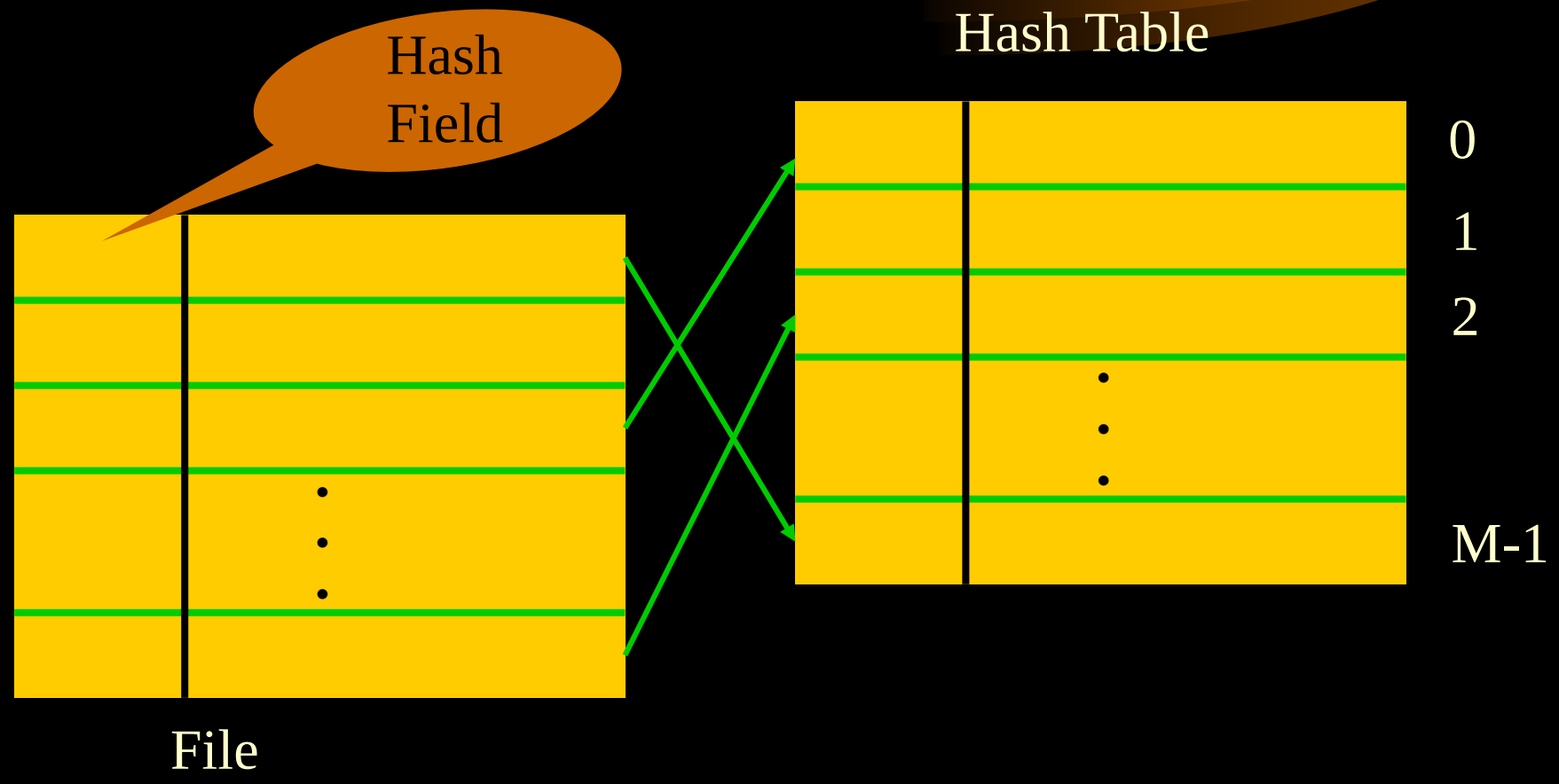
Database Systems

- Hash File
  - Hash file is a file organization that provides fast access to records based on certain search conditions — search must be on equality on a single field (hash field) of the file. In case where hash field is a key, then it is called hash key.
  - The basic idea is to map the contents of the hash field into a disk block address using a function called hash function. Naturally, a search for a record within a block will be carried out in a memory buffer. For most cases, this organization requires one block access to the disk.

Database Systems

- Internal Hashing
  - In case of internal hashing, hashing is typically implemented as a table — table of M entries whose addresses correspond to the array indexes. Hash function is applied to the hash field of each individual record, this will map the contents of the hash field into an address in the hash table.

Database Systems



Database Systems

- Internal Hashing
  - A problem with hashing is due to the fact that two different hash field values may hash to the same slot in the hash table — collision. In this case the second record needs to be stored somewhere else — collision resolution.
  - Several algorithms could be used to overcome this problem.

Database Systems

- Collision resolution
  - Open Hashing: After collision, proceed from the occupied position, specified by the hash address, to the next available slot and store the record in that empty slot.
  - Chaining: Each hash slot is augmented by a pointer. The collided record is stored in the overflow area and a pointer to that location is maintained in the hash table.
  - Multiple hashing: A second hash function is applied if the first resulted in a collision.

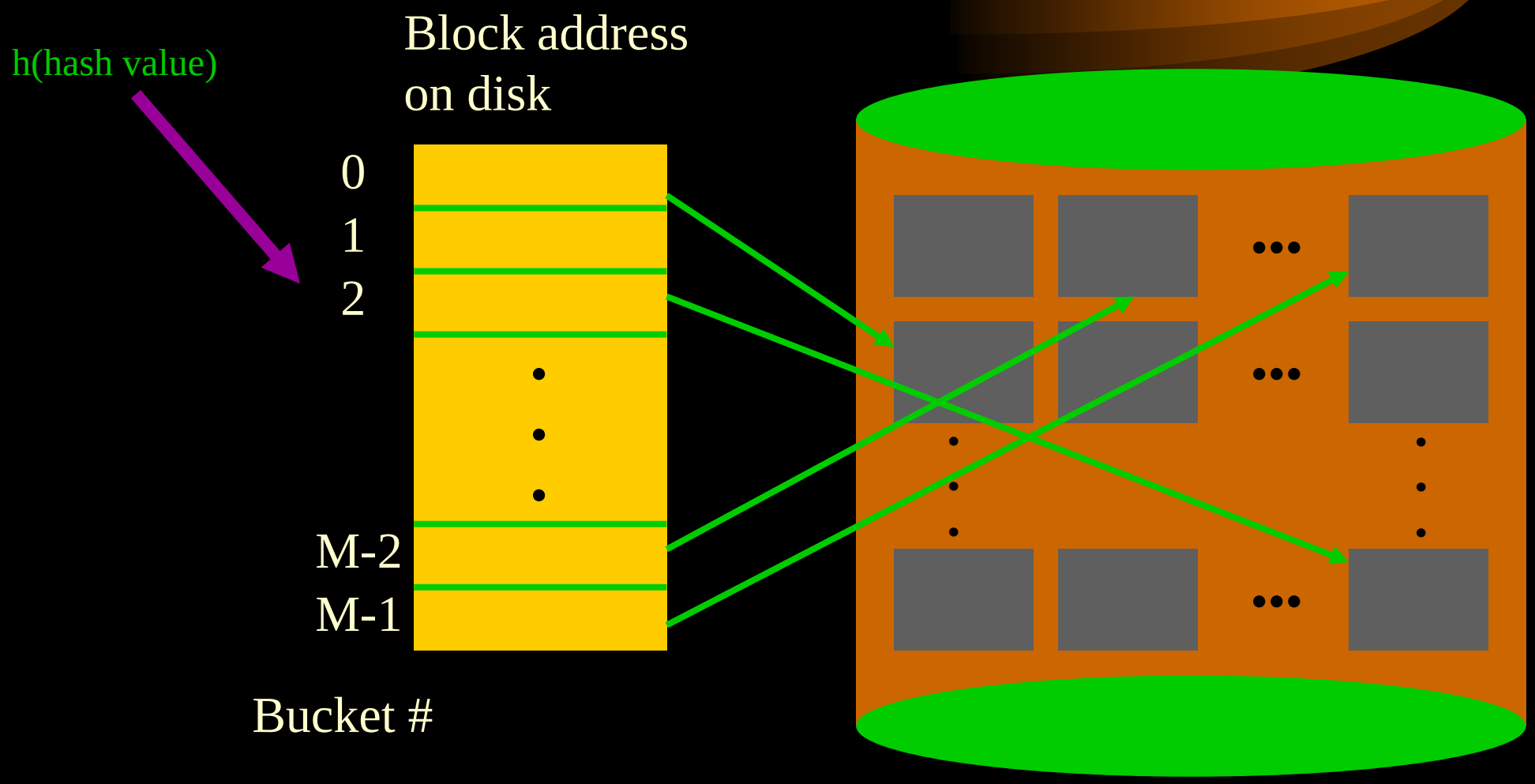
Database Systems

- Collision resolution
  - Insertion, deletion, and retrieval algorithms vary based on the collision resolution method.
  - In general, algorithms for chaining is easier and simpler. Deletion algorithm for open addressing is rather tricky.
  - A good hashing function should distribute the information uniformly over the address space in order to minimize collisions and utilize space efficiently.

Database Systems

- External Hashing
  - Hashing for disk files is called external hashing. Hashing issues as discussed before can be modified to fit the characteristics of disk — records are stored in blocks rather than individually, each block is called a bucket.
  - In case of external hashing, the hash function provides a relative bucket number and a table is used to correspond each relative bucket number to the disk block address.

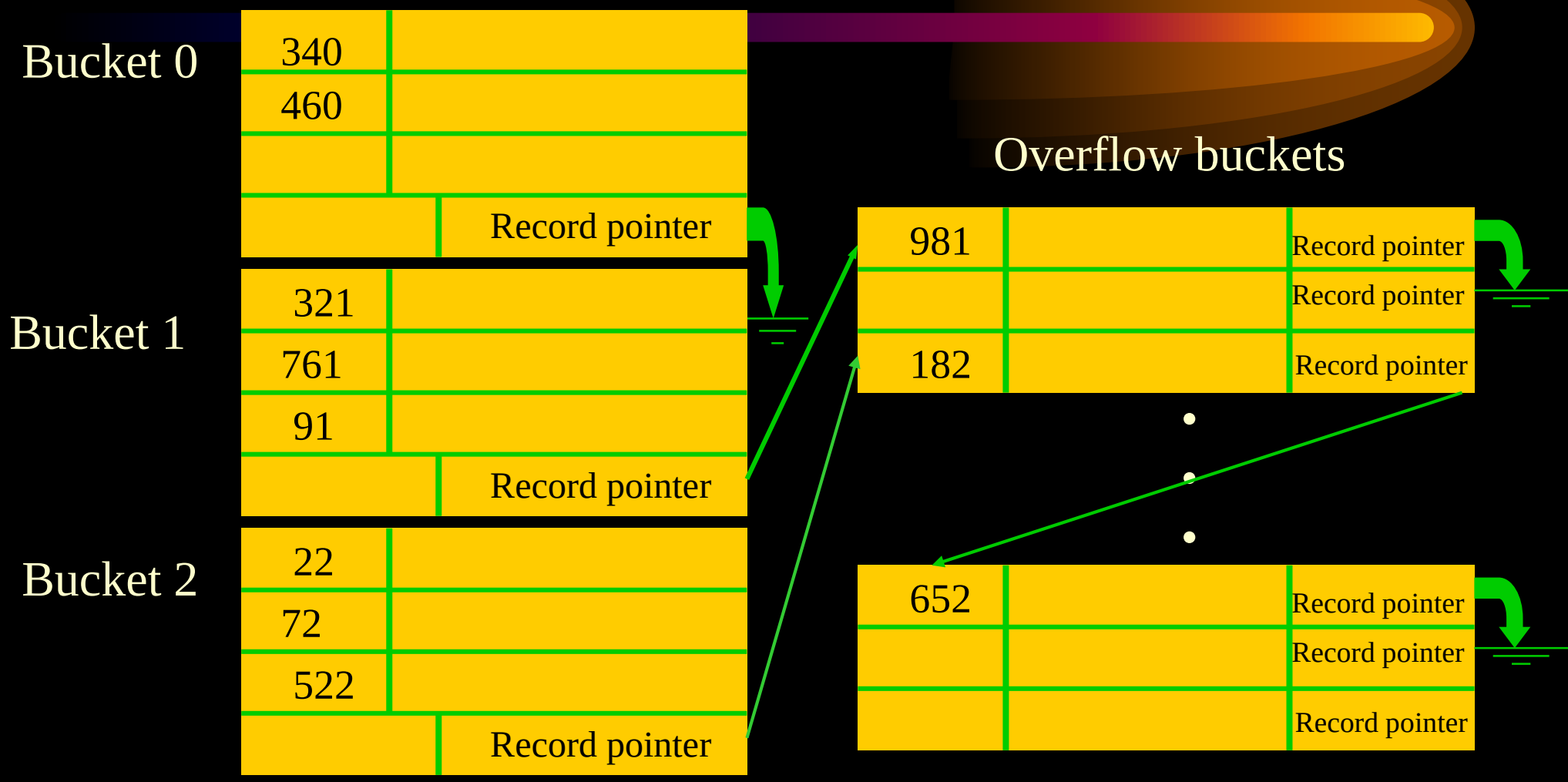
Database Systems



Database Systems

- External Hashing
  - In case of external hashing, the collision problem is less severe with buckets since each bucket holds several records. However, still provisions should be made in case the buckets get full. A variation of chaining can be used by making a linked list of overflow records for the bucket — note that in this case, the record pointer includes both the bucket# and a relative record position within the bucket.

Database Systems



Database Systems

- External Hashing
  - Hashing provides a fast access for retrieving an arbitrary record based on the contents of the hash field. However, it is showing a poor performance for other applications. For example, if we required the retrieval of records in order of their hash field values, hashing is not very suitable — majority of hash functions do not maintain records in order of hash field values.

Database Systems

- External Hashing
  - Another drawback is based on static size of address space. If we allocate *M* buckets and each bucket is as large as *m* records, then at most we have $m * M$ records in the allocated space. Now, if the number of records is significantly less than $m * M$ a lot of space will be unused, or if the number of records exceeds drastically from $m * M$ we will end up with a lot of collisions and hence retrieval operations will be degraded. As a result, we have to rearrange the file, by changing the address space and using different hashing function.

Database Systems

- External Hashing
  - Searching for a record based on the contents of some fields other than the hash field is as slow as unordered file — Linear search.
  - Record deletion can be implemented by removing deleted record from its bucket and moving a record from the overflow area into the bucket (in case the bucket has an overflow).

Database Systems

- External Hashing
  - Modifying a field value depends on two factors:
    - The search criterion to locate the record
    - The field to be modified
  - If the search criterion is an equality comparison on the hash field, the record can be located efficiently, otherwise we need to perform a linear search.

Database Systems

- External Hashing
  - A non hash field can be modified by changing the record and rewriting it in the same physical location on disk — assuming fixed length record.
  - Modifying the hash field requires a deletion of the old record followed by insertion of the modified record.

Database Systems

- Hashing schemes to allow Dynamic File expansion
  - Hashing schemes discussed so far assume a fixed address space! So the question is how to allow a file to expand or shrink dynamically.
  - For this purpose, we create an access structure in addition to the file (similar to indexing). Access structure is created as a result of applying the hash function to the hash field, however, in indexing, index is set just based on the value of the index field.

Database Systems

- Hashing schemes to allow Dynamic File expansion
  - The techniques we will discuss get advantage of the fact that applying the hash function to the hash field results in a non negative number represented in binary.
  - The access structure is built on the binary representation of the result of applying hash function to the hash field.

Database Systems

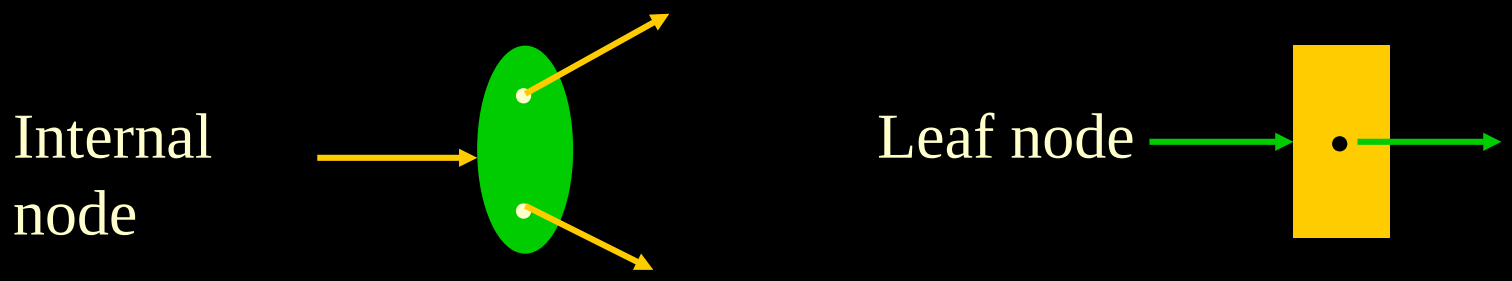
- Dynamic Hashing
  - In dynamic hashing, the number of buckets is not fixed — grows and diminished during the course of operations.
  - The file starts with a single bucket, once the bucket is full and a new record is being inserted, the bucket is split into two buckets and records are distributed among the buckets based on the value of the first bit of the hash value.

Database Systems

- Dynamic Hashing
  - All records with a hash field starting with a 0 are stored in one bucket and all the others in the other bucket.
  - This results in a binary tree (directory).
  - The directory is composed of two types of nodes:
    - Internal Nodes,
    - Leaf Nodes.

Database Systems

- Dynamic Hashing
  - Internal Nodes: each internal node points to its left child (corresponding to a 0 bit) and right child (corresponding to 1 bit).
  - Leaf nodes: each leaf node points to a bucket holding the records.

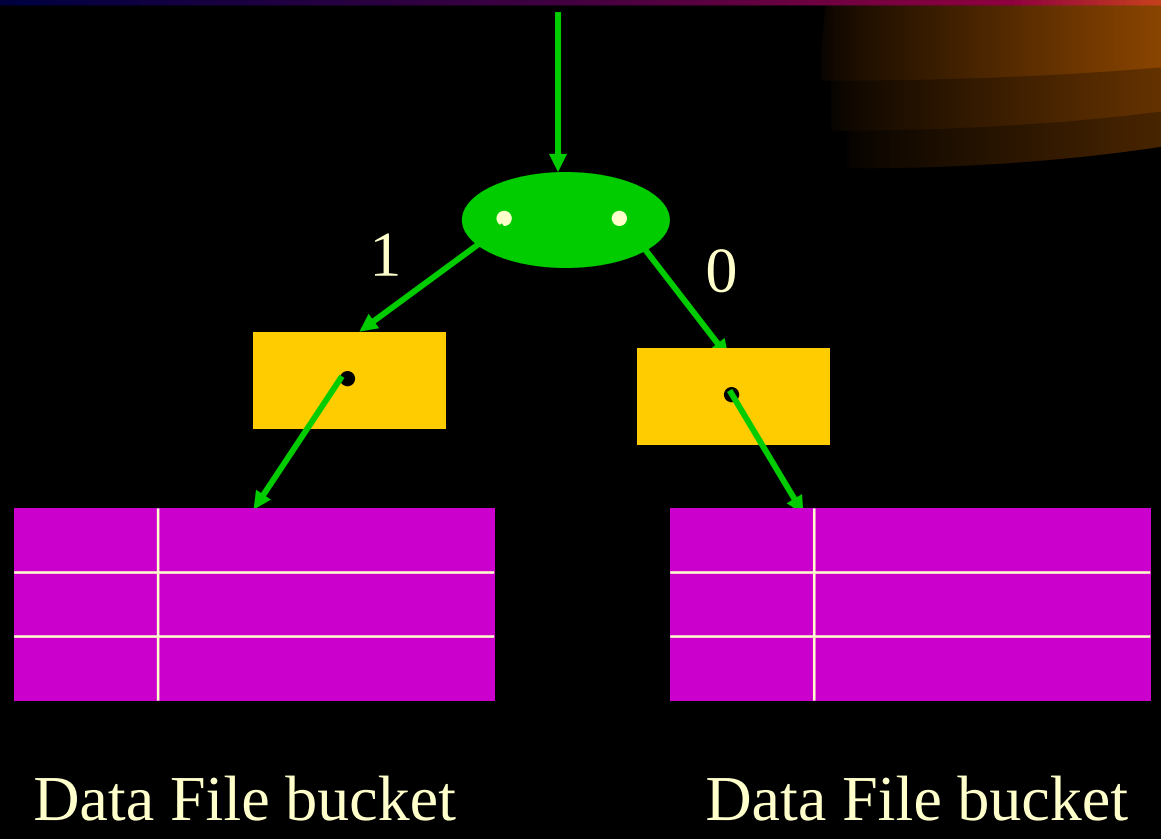


Database Systems

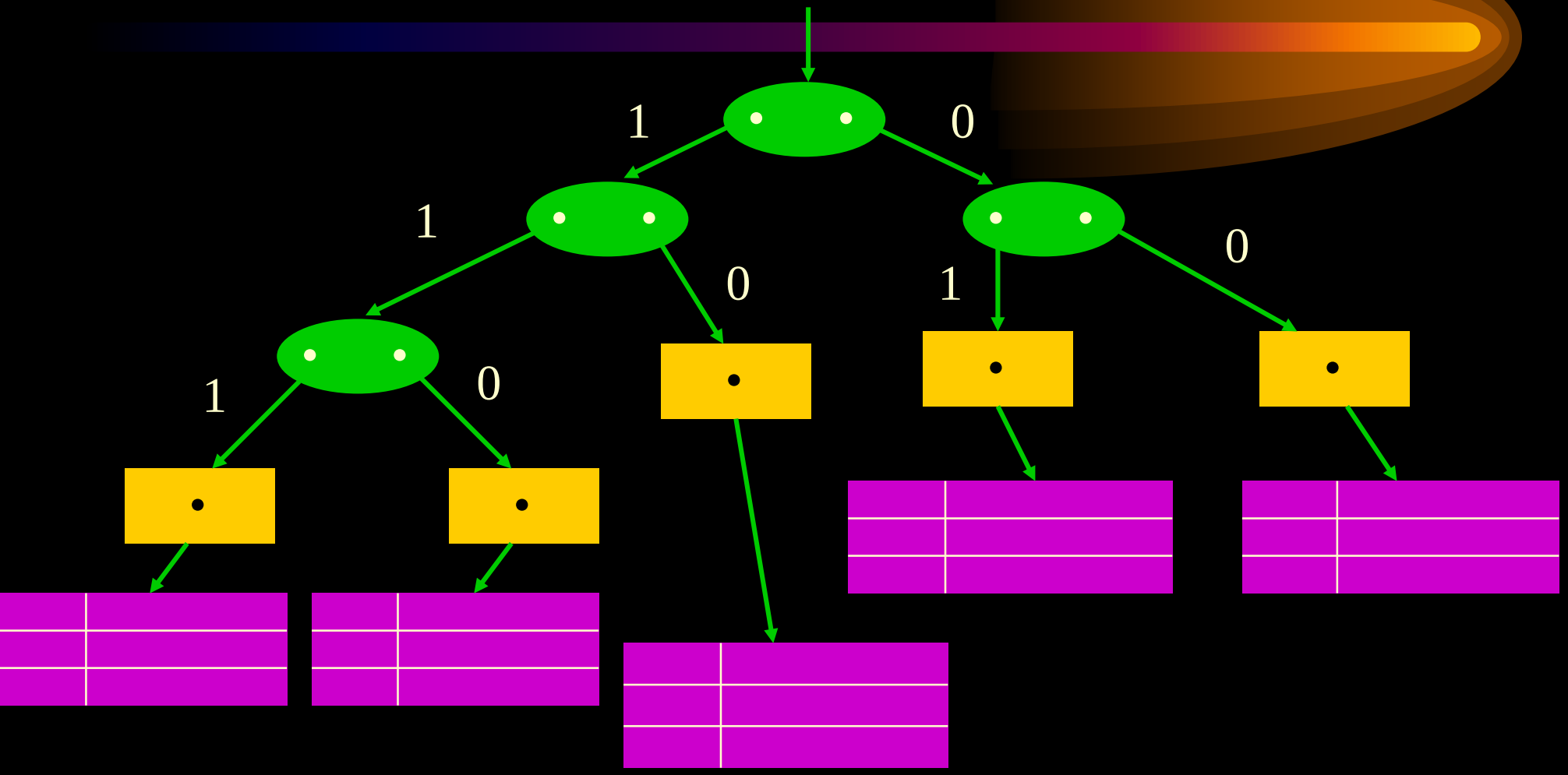
- Dynamic Hashing

```
H ← h(hash field)
t ← root of directory
i ← 1
while t is an internal node
    begin
        if i^th bit in H is a 0
            then t ← t(left)
            else t ← t(right)
        i ← i + 1
    end
search the bucket whose address in in t;
```

Database Systems



Database Systems



Data File bucket

Data File bucket

Database Systems

- Dynamic Hashing
  - If the hash function distributes the records uniformly, the directory will be a balanced tree.
  - Buckets can collapse together if a bucket become empty or if two neighboring buckets can fit in a single bucket.

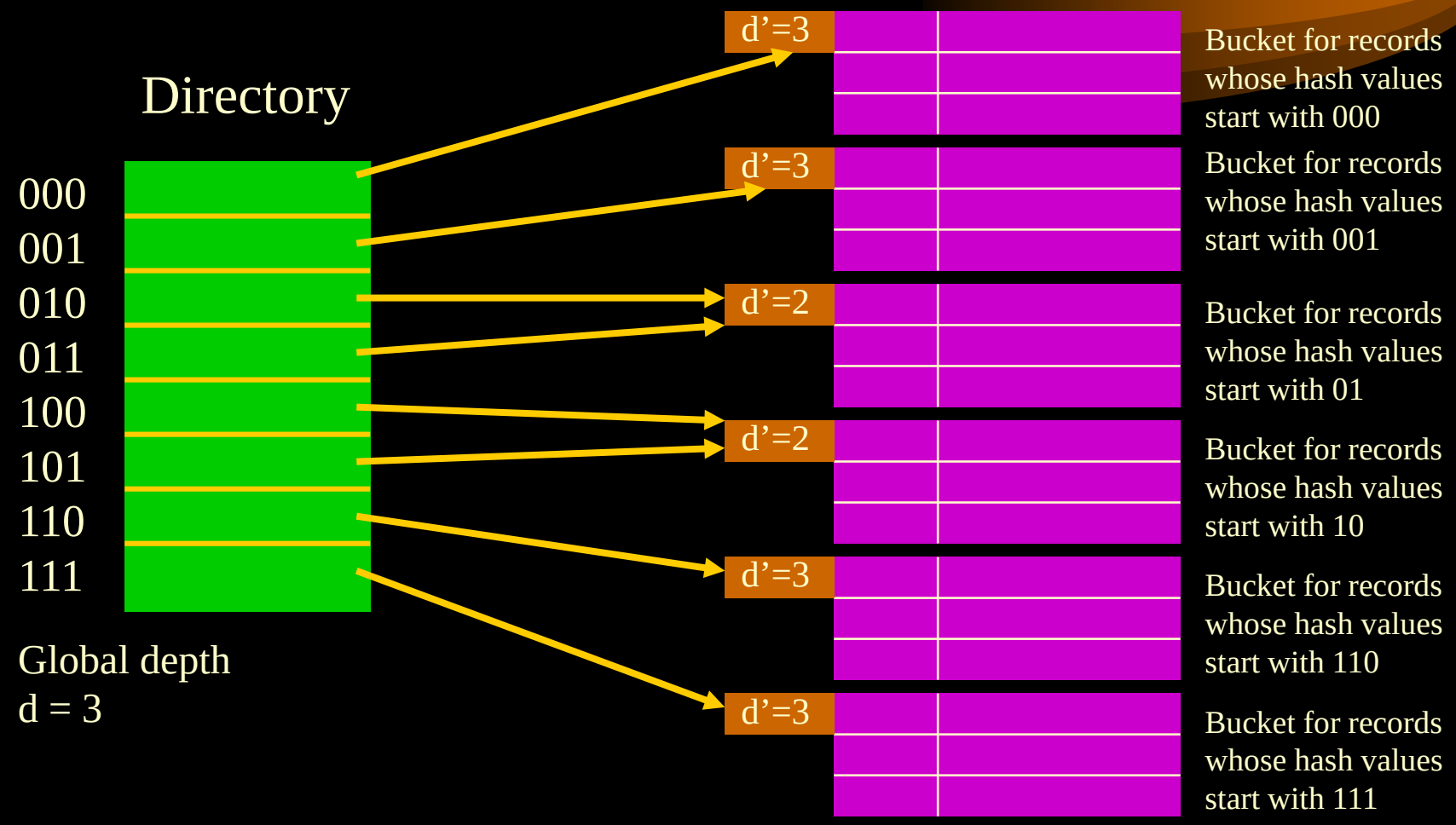
Database Systems

- Extendible Hashing
  - In extendible hashing an array of 2^d bucket addresses is maintained — d is called the global depth of the directory.
  - The first d bits (high order bits) of a hash value determine an entry in the directory where the contents of that entry points to the bucket that hold the record.
  - Note that entries in the directory necessarily are not pointing to different buckets.

Database Systems

- Extendible Hashing
  - A local depth (*d'*) is stored with each bucket. A bucket will split if we want to insert a record to a full bucket.
  - Directory can also expand (double) if we make an attempt to insert a record to a full bucket whose local depth is equal to the global depth.

Database Systems



Database Systems

- Extendible Hashing
  - An attempt to insert a record into the third bucket causing an overflow. As a result, records are distributed between two buckets of local depth 3. The first bucket contains all records whose hash value starts with 010 and the second bucket contains those records whose hash value starts with 011. The directory positions 010 and 011 then, will point to two distinct buckets.

Database Systems

- Extendible Hashing
  - Insertion of a record with the hash value starting with 111 causes an overflow in the corresponding bucket. Since the local depth is equal to the global depth, the directory must be expanded.
  - Each of the entry in the original directory splits into two locations, both of them having the same pointer value.