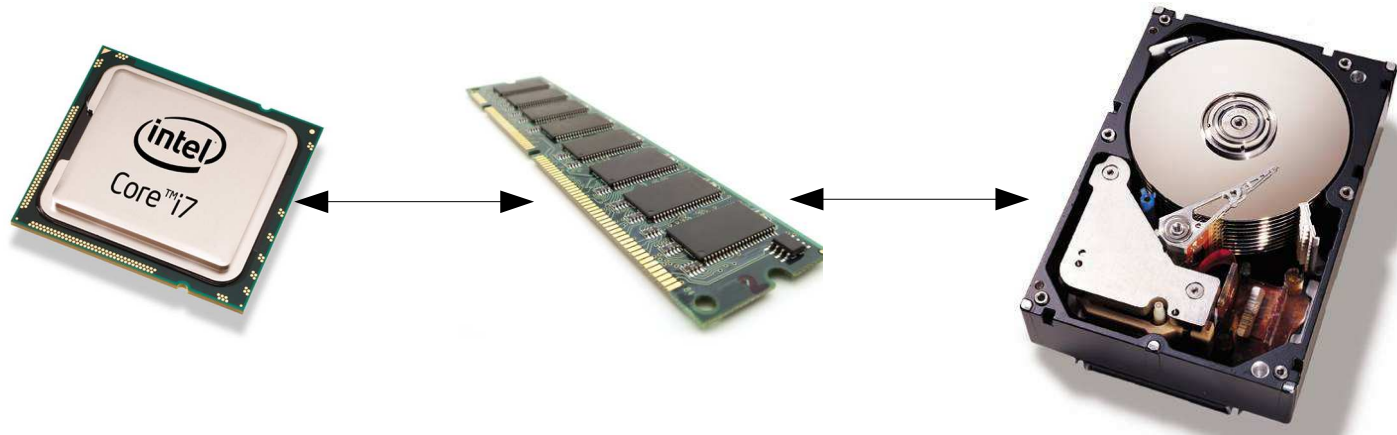


# One-dimensional index structures

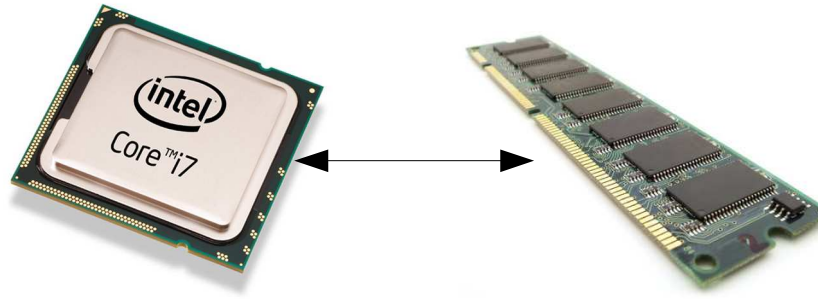
## Motivation: The I/O model of computation



### The I/O model

- Data is stored on disk, which is divided into **blocks** of bytes (typically 4 kilobytes) (each block can contain many data items)
- The CPU can only work on data items that are in memory, not on items on disk
- Therefore, data must first be transferred from disk to memory
- Data is transferred from disk to memory (and back) in whole blocks at the time
- The disk can hold  $D$  blocks, at most  $M$  blocks can be in memory at the same time (with  $M \ll D$ ).

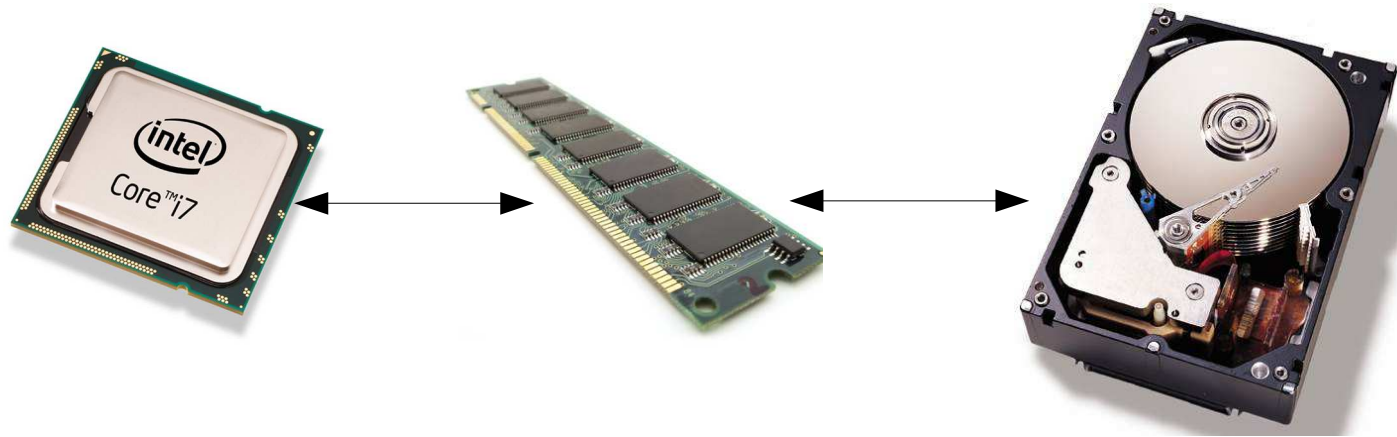
## Motivation: The I/O model of computation



**However: complexity of algorithms is traditionally analyzed in the RAM model of computation**

- Data is stored in an (infinite) memory
- The CPU works on data items in memory
- Complexity is measured in terms of the number of memory accesses and CPU operations.

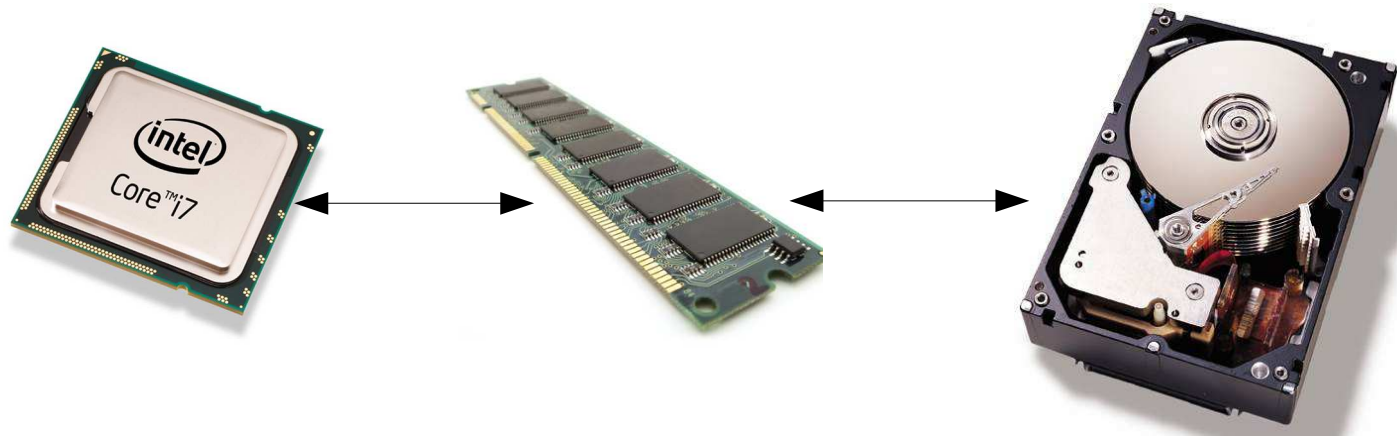
## Motivation: The I/O model of computation



“The difference in speed between modern CPU and disk technologies is analogous to the difference in speed in sharpening a pencil using a sharpener on ones desk or by taking an airplane to the other side of the world and using a sharpener on someone elses desk.”

(D. Comer)

## Motivation: The I/O model of computation



- In-memory computation is fast (memory access  $\approx 10^{-8}s$  )
- Disk-access is slow (disk access:  $\approx 10^{-3}s$  )
- Hence: execution time is dominated by disk I/O

**We will use the number of I/O operations required as cost metric**

## Motivation: searching in a database

### A hypothetical database

- A relation  $R(A, B, C, D)$ . Each tuple comprises 32 bytes.
- Attribute  $C$  is a (secondary) key for  $R$ .
- There are  $128 \cdot 10^6$  tuples in the relation. The block size  $B = 4096$  bytes.
- Hence there are 128 tuples per block, or  $10^6$  blocks in total.

### Searching for record with $C = 10$ in case $R$ is arbitrary

- For every block  $X$  in  $R$ :
  - Load  $X$  from disk in memory
  - Check whether there is a tuple with  $A = 10$  in  $X$ ;
  - If so output record and terminate loop; otherwise continue
  - Release  $X$  from memory
- Worst case I/O Cost: the total number of blocks in  $R$ , or  $10^6$  I/O's.
- At  $10^{-3}$  s per IO this takes 16.6 minutes.  $\Rightarrow$  **Can we do better?**

# Index structures

See corresponding slides

# Searching in a database with a index (1/2)

## The database

- A relation  $R(A, B, C, D)$ . Each tuple comprises 32 bytes.
- Attribute  $C$  is a (secondary) key for  $R$ .
- There are  $128 \cdot 10^6$  tuples in the relation. The block size  $B = 4096$  bytes.
- Hence there are 128 tuples per block, or  $10^6$  blocks in total.

## The index

- There is a secondary index on attribute  $C$ .
- A (key value, ptr) pair in the index takes 16 bytes.
- **Question:** How many (key, ptr) pairs fit in a block?
- **Question:** How many blocks does the dense 1st level index take?
- **Question:** How many blocks does the sparse 2nd level index take?



# Searching in a database with a index (1/2)

## The database

- A relation  $R(A, B, C, D)$ . Each tuple comprises 32 bytes.
- Attribute  $C$  is a (secondary) key for  $R$ .
- There are  $128 \cdot 10^6$  tuples in the relation. The block size  $B = 4096$  bytes.
- Hence there are 128 tuples per block, or  $10^6$  blocks in total.

## The index

- There is a secondary index on attribute  $C$ .
- A (key value, ptr) pair in the index takes 16 bytes.
- **Question:** How many (key, ptr) pairs fit in a block? 256
- **Question:** How many blocks does the dense 1st level index take?  $5 \cdot 10^5$
- **Question:** How many blocks does the sparse 2nd level index take? 1954

## Searching in a database with a index (2/2)

### Searching for records with $C = 10$ using the index

- Algorithm:
  - Loop through all of the blocks in  $X$ , one, by one, and find the (key, ptr) pair in  $X$  with the largest key value satisfying  $\text{key} \leq 10$ .
  - Follow ptr to dense index block, and use the information in this block to locate the block in  $R$  containing the record with  $C = 10$  (if it exists).
- Worst case I/O Cost: loading of all blocks of sparse index + 1 block of dense index + 1 block of  $R$ , or  $1954 + 1 + 1 = 1956$  I/Os.
- At  $10^{-3}$  s per I/O this takes 2 seconds.

**Since the sparse index is sorted, we could perform binary search on it.**

- I/O Cost: binary search in sparse index + 1 block of dense index + 1 block of  $R$ , or  $\log_2(1954) + 1 + 1 = 14$  I/Os  $\rightarrow 0.014$  seconds.

# Searching in a database with a BTree index (1/6)

## The database

- A relation  $R(A, B, C, D)$ . Attribute  $C$  is a (secondary) key for  $R$ .
- There are  $128 \cdot 10^6$  tuples in the relation. The block size  $B = 4096$  bytes.

## The index

- There is a BTree index on attribute  $C$ .
- A key value takes 8 bytes, a ptr also 8 bytes.
- **Question:** What is the maximum order  $n$  of the BTree, taking into account that blocks are 4096 bytes large?

## Searching in a database with a BTree index (2/6)

### The database

- A relation  $R(A, B, C, D)$ . Attribute  $C$  is a (secondary) key for  $R$ .
- There are  $128 \cdot 10^6$  tuples in the relation. The block size  $B = 4096$  bytes.

### The index

- There is a BTree index on attribute  $C$ .
- A key value takes 8 bytes, a ptr also 8 bytes.
- **Question:** What is the maximum order  $n$  of the BTree, taking into account that blocks are 4096 bytes large?
- **Answer:** A BTree of degree  $n$  stores  $n + 1$  pointers and  $n$  key values in each block. We are hence looking for the largest integer value of  $n$  satisfying:

$$(n + 1) \text{ ptrs} \times 8 \text{ bytes/ptr} + n \text{ keys} \times 8 \text{ bytes/ptr} \leq 4096 \text{ bytes}$$

As such,  $n = 255$ : we store 256 pointers and 255 keys in a block.

# Searching in a database with a BTree index (3/6)

## The database

- A relation  $R(A, B, C, D)$ . Attribute  $C$  is a (secondary) key for  $R$ .
- There are  $128 \cdot 10^6$  tuples in the relation. The block size  $B = 4096$  bytes.

## The index

- There is a BTree index of order 255 on attribute  $C$ .
- **Question:** What is the height of the BTree assuming that leaf blocks are full and internal blocks contain 255 pointers?

# Searching in a database with a BTree index (4/6)

## The database

- A relation  $R(A, B, C, D)$ . Attribute  $C$  is a (secondary) key for  $R$ .
- There are  $128 \cdot 10^6$  tuples in the relation. The block size  $B = 4096$  bytes.

## The index

- There is a BTree index of order 255 on attribute  $C$ .
- **Question:** What is the height of the BTree assuming that leaf blocks are full and internal blocks contain 255 pointers?
- **Answer:** : Observe:
  - there are  $\left\lceil \frac{128 \cdot 10^6}{255} \right\rceil$  leaf blocks (at level 1)
  - there are  $\left\lceil \frac{128 \cdot 10^6}{(255)^2} \right\rceil$  blocks at level 2
  - there are  $\left\lceil \frac{128 \cdot 10^6}{(255)^3} \right\rceil$  blocks at level 3
  - ...

○ there are  $\left\lceil \frac{128 \cdot 10^6}{(255)^h} \right\rceil$  blocks at level  $h$

Since the root is at the level where there is only one block, we are looking for the smallest value of  $h$  such that  $\left\lceil \frac{128 \cdot 10^6}{(255)^h} \right\rceil = 1$ .

So,  $h = \lceil \log_{255} 128 \cdot 10^6 \rceil = 4$ .

# Searching in a database with a BTree index (5/6)

## The database

- A relation  $R(A, B, C, D)$ . Attribute  $C$  is a (secondary) key for  $R$ .
- There are  $128 \cdot 10^6$  tuples in the relation. The block size  $B = 4096$  bytes.

## The index

- There is a BTree index of order 255 on attribute  $C$ .
- Hence we can store at most 256 pointers; and 255 key values in a block.
- **Question:** What is the cost of searching for the record with  $C = 10$  using this BTree, assuming the worst-case scenario that each block in the BTree is half full?



# Searching in a database with a BTree index (6/6)

## The database

- A relation  $R(A, B, C, D)$ . Attribute  $C$  is a (secondary) key for  $R$ .
- There are  $128 \cdot 10^6$  tuples in the relation. The block size  $B = 4096$  bytes.

## The index

- There is a BTree index of order 255 on attribute  $C$ .
- Hence we can store at most 256 pointers; and 255 key values in a block.
- **Question:** What is the cost of searching for the record with  $C = 10$  using this BTree, assuming the worst-case scenario that each block in the BTree is half full?

**Answer:** height of the Bree in which blocks are half full + 1 I/O to access main file

$$= \lceil \log_{128} 128 \cdot 10^6 \rceil + 1 = 5 \rightarrow 0.005 \text{ seconds.}$$

# Inserting in a BTree index

## The database

- A relation  $R(A, B, C, D)$ . Attribute  $C$  is a (secondary) key for  $R$ .
- There are  $128 \cdot 10^6$  tuples in the relation.

## The index

- There is a BTree index of order 255 on attribute  $C$ .
- Hence we can store at most 256 pointers; and 255 key values.
- **Question:** What is the cost of inserting a new record in this BTree, assuming the record is already in the main file, and assuming the worst-case scenario where each block in the BTree is full?

# Inserting in a BTree index

## The database

- A relation  $R(A, B, C, D)$ . Attribute  $C$  is a (secondary) key for  $R$ .
- There are  $128 \cdot 10^6$  tuples in the relation.

## The index

- There is a BTree index on attribute  $C$ .
- Hence we can store at most 256 pointers; and 255 key values.
- **Question:** What is the cost of inserting a new record in this BTree, assuming the record is already in the main file, and assuming the worst-case scenario where each block in the BTree is full?

**Answer:** cost of a search + 2 I/O's per level of the BTree

$$= \lceil \log_{255} 128 \cdot 10^6 \rceil + 2 \lceil \log_{255} 128 \cdot 10^6 \rceil + 1 = 3 \lceil \log_{255} 128 \cdot 10^6 \rceil + 1 = 13 \rightarrow 0.013s$$