# External Memory Algorithms:

# Computing on MASSIVE Data

*Jeff Vitter*

Duke University

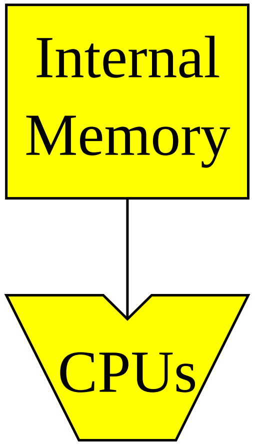
Department of Computer Science

Center for Geometric & Biological Computing

http://www.cs.duke.edu/CGBC/

EEF Summer School—July 2002

# The RAM Model of Computation

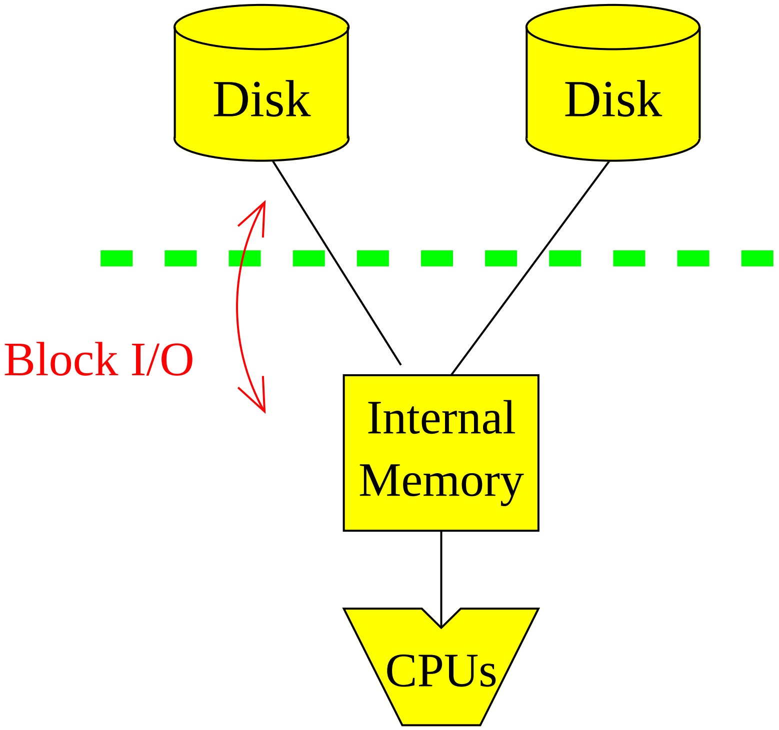


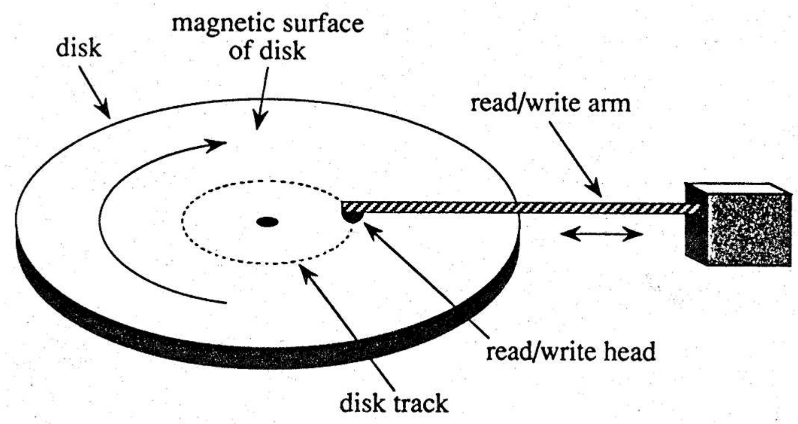

- ☆ The simple uniform memory model—unit time per memory access–enabled the development of compilers and high-level languages.
- ☆ Crucial for success of the computer industry.

*BUT . . .*

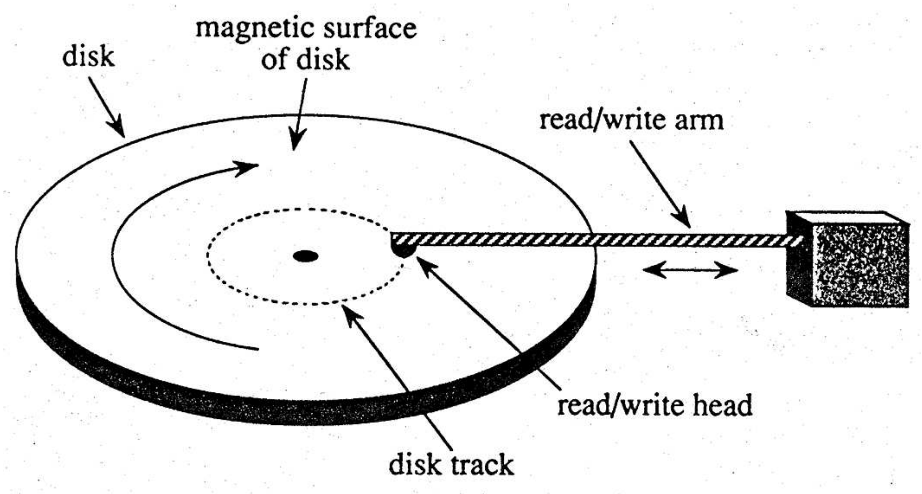
## What is Realistic

Doug Comer: *The difference in speed between modern CPU and disk technologies is analogous to the difference in speed in sharpening a pencil by using a sharpener on one's desk or by taking an airplane to the other side of the world and using a sharpener on someone else's desk.*



# Magnetic Disk Drives as Secondary Memory




- ☆ I/O Crisis! Disk access is 1,000,000 times slower!
  More like *Around the World in Eighty Days*!
- ☆ Time for rotation ≈ Time for seek.
- ☆ Amortize search time by transferring large blocks so that
  Time for rotation ≈ Time for seek ≈ Time to transfer data.
- ☆ Solution 1: Exploit locality: take advantage of block transfer.
- ☆ Solution 2: Use disks in parallel.

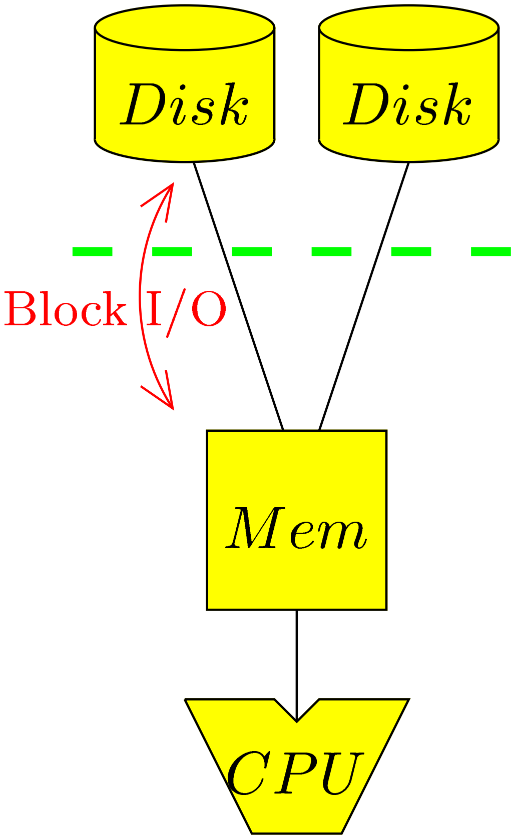
## Trends

| Parameter | Yearly Improvement Rate |
|---|---|
| Disk Latency | 10% |
| Disk Bandwidth | 20% |
| Processor Speed | 55% |
| RAM Bandwidth | 40% |
| RAM Capacity/Cost | 45% |

- ☆ Performance gap is increasing.
- ☆ RAM Capacity/Cost doubling every 22–23 months, but users doubling data storage every 5 months.
- ☆ Users frequently reprocess data in entirety.
- ☆ Result is an I/O Bottleneck.

# Parallel Disk Model

[Aggarwal & Vitter 88], [Vitter & Shriver 90, 94], ...




$N$ = problem data size.

$M$ = size of internal memory.

$B$ = size of disk block.

$D$ = number of independent disks.

$P$ = number of CPUs.

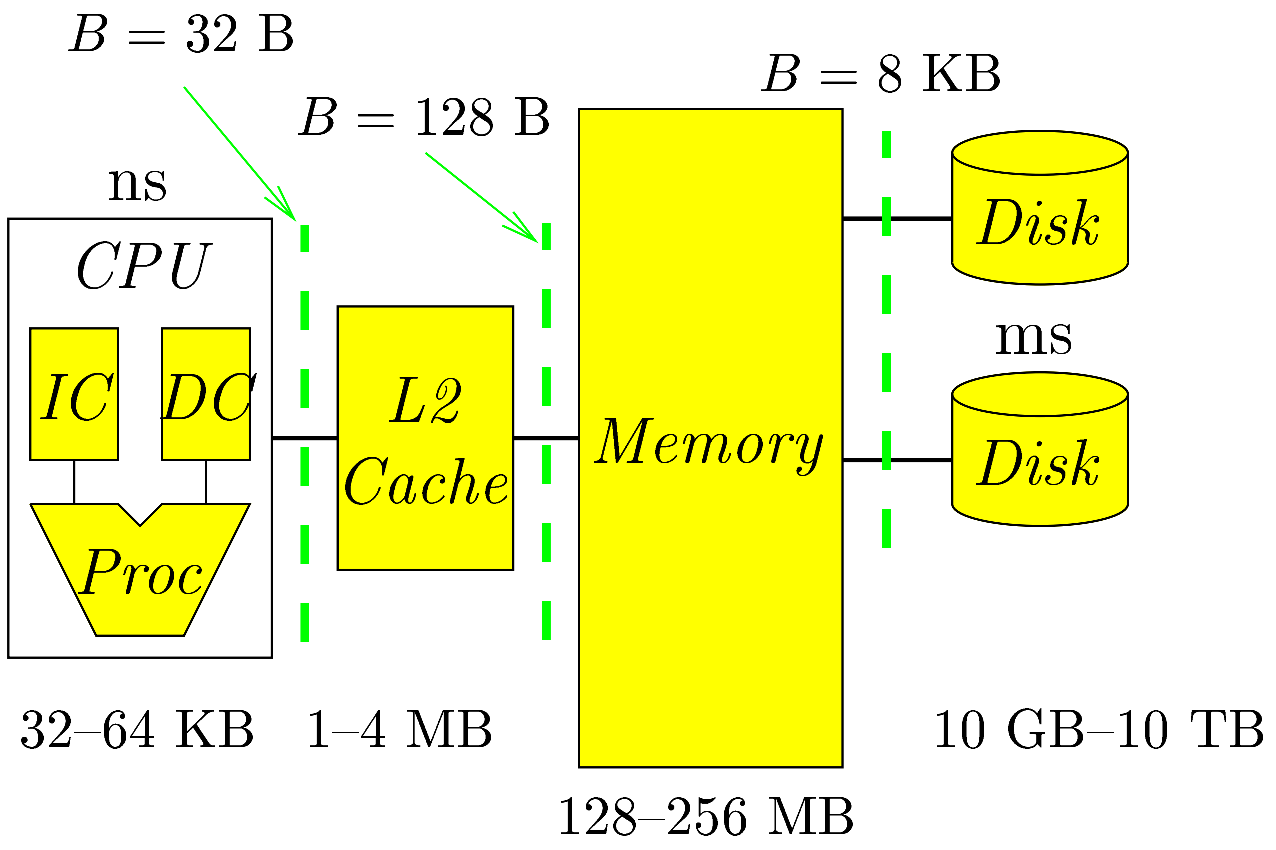
$Q$ = number of queries.

$Z$ = problem output size.

Notational convenience (in units of blocks):

$$n = \frac{N}{B}, \quad m = \frac{M}{B}, \quad q = \frac{Q}{B}, \quad z = \frac{Z}{B}.$$

# A “Real” Machine



# Outline

1. Fundamental Techniques for batched problems.
   - ⋆ Merge sort, distribution sort.
2. Techniques for solving batched geometric problems.
   - ⋆ Distribution sweeping, batched filtering, randomized incremental construction.
   - ⋆ Red-blue orthogonal rectangle intersection, convex hull, range search, nearest neighbors.
   - ⋆ Empirical results (via TPIE programming environment).
3. Fundamental lower bounds.
   - ⋆ Sorting, permuting, FFT, matrix transposition, bundle sort.
   - ⋆ Dynamic memory allocation
   - ⋆ Hierarchical memory.
4. Parallel disks.
   - ⋆ Load balancing among disks is key issue.
   - ⋆ Duality: reading (prefetching) $\longleftrightarrow$ writing, merging $\longleftrightarrow$ distribution

# Fundamental I/O Bounds (with $D = 1$ disk)

⋆ Batched problems [AV88], [VS90], [VS94]:

- Scanning (touch problem): $\Theta\left(\frac{N}{B}\right) = \Theta(n)$
- Sorting:

$$\Theta\left(\frac{N}{B}\frac{\log\frac{N}{B}}{\log\frac{M}{B}}\right) = \Theta\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right) = \Theta\left(n\log_m n\right)$$

- Permuting: $\Theta\left(\min\left\{N,\ n\log_m n\right\}\right)$

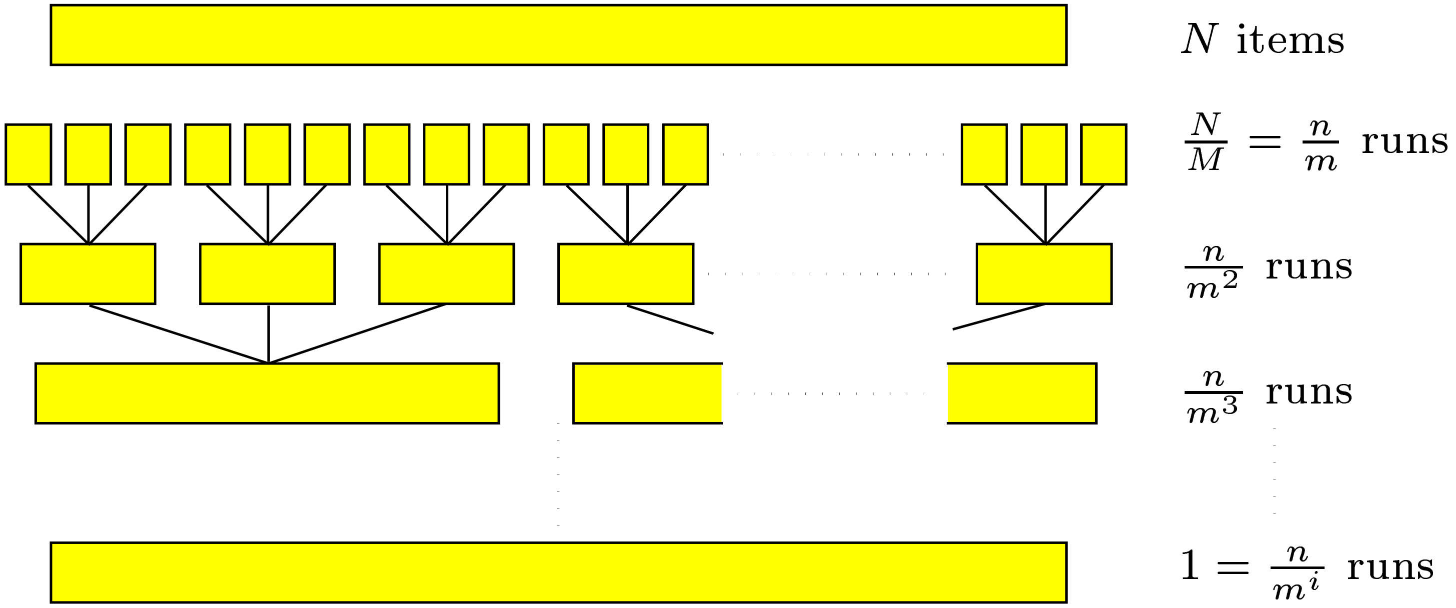
⋆ For other problems [CGGTVV95], [AKL95], . . .

- Graph problems $\asymp$ Permutation
- Computational Geometry $\asymp$ Sorting

⋆ Online problems:

- Searching and Querying: $\Theta\left(\log_B N + \frac{Z}{B}\right) = \Theta(\log_B N + z)$

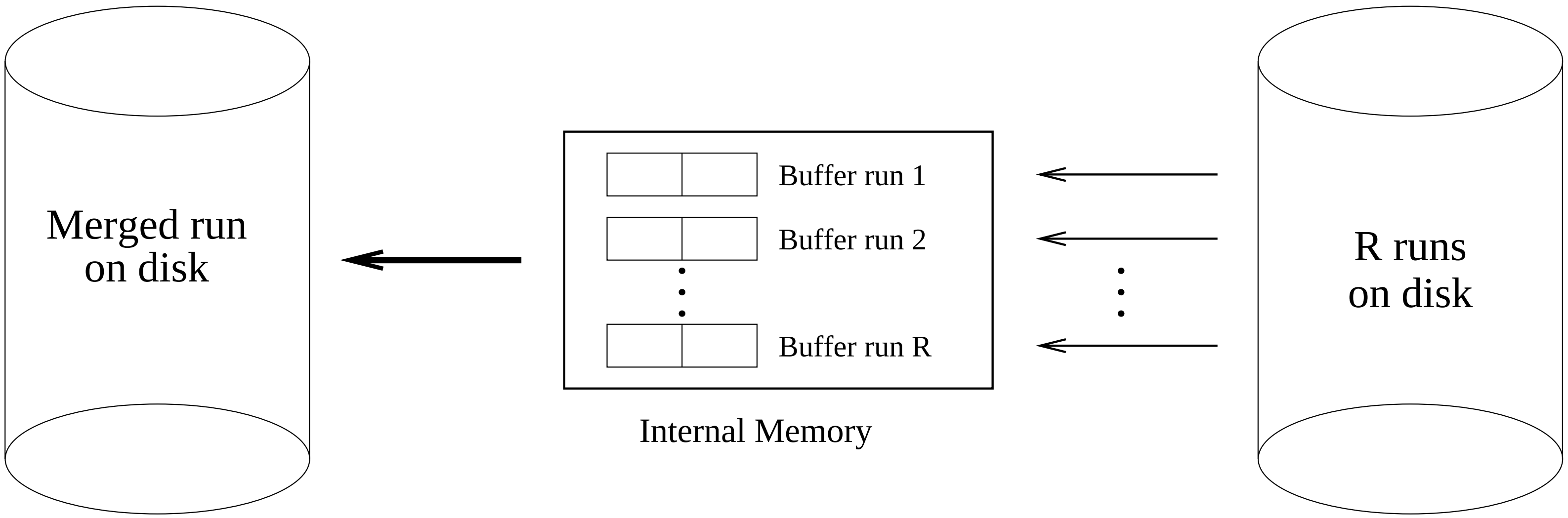
# Design Techniques — Merging




☆ # passes $= \left\lceil \log_m \frac{n}{m} \right\rceil + 1 = \lceil \log_m n \rceil \approx \log_m n.$

☆ $2n$ I/Os used in each pass $\Longrightarrow O(n \log_m n)$-I/O solution.

# Merge Sort



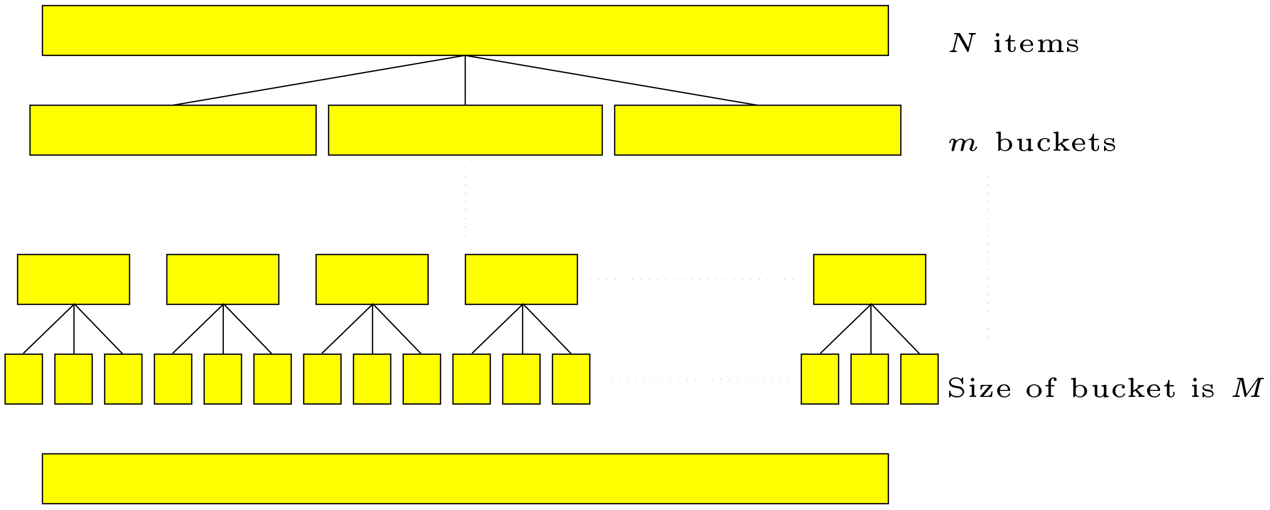

- ☆ Form $2n/m$ initially sorted runs, each consisting of $m/2$ blocks (i.e., one half-memoryload). Sort one half-memoryload while outputting the previously sorted half-memoryload, so that I/O is overlapped with computation. Big win!

- ☆ Merging phase: Use $R = m/2 - 2$ double buffers, each size $B$, in internal memory. Repeatedly merge together $R$ runs at a time. For each run, one buffer is used for the merge while the other is being filled, which overlaps I/O with computation.

## Merge Sort

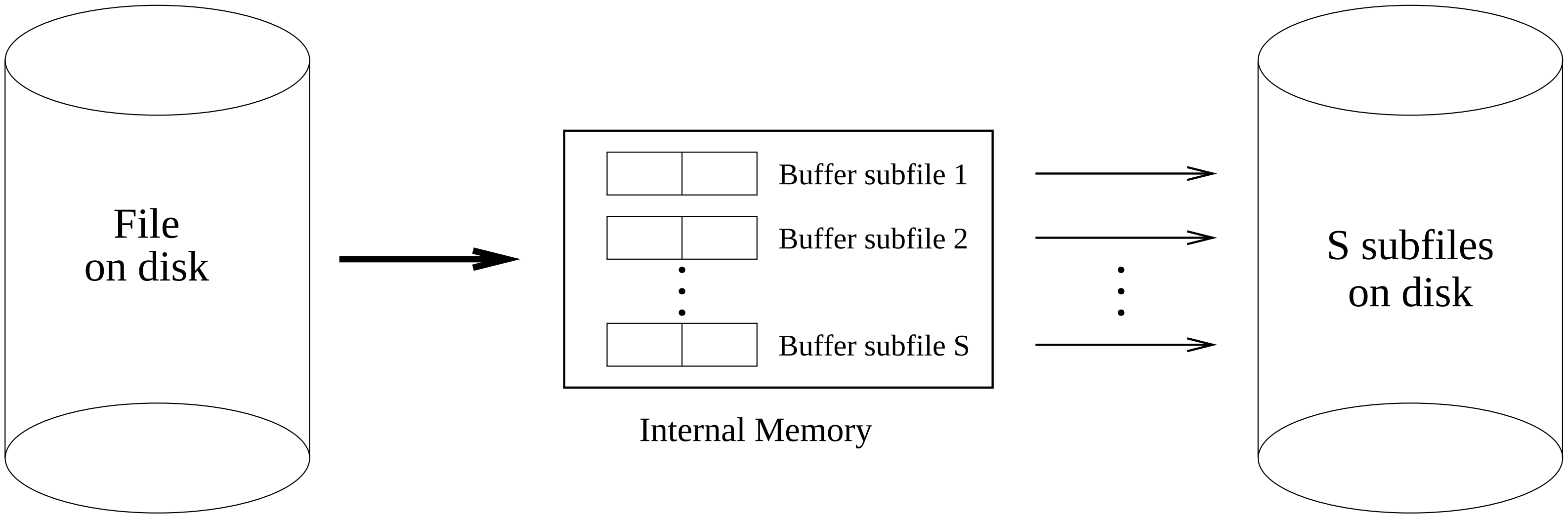
- $\implies \quad \#\ \text{passes} = 1 + \log_R \frac{2n}{m} \approx 1 + \log_{m/2} n - 1 \approx \log_m n.$
- Double buffers keep # passes roughly the same, but overlap I/O with processing. Big win!
- Each pass uses $\Theta(n)$ I/Os
  $\implies \quad \#\ \text{I/Os} = O\left(n \log_m n\right).$

# Design Techniques—Distribution




- ☆ # levels of recursion is roughly $\log_m \frac{N}{M} = \log_m n - 1$.
- ☆ If distribution factor is reduced to $\sqrt{m}$, the pivot elements can be found deterministically in $O(n)$ I/Os [AV 88].
- ☆ Constant factor increase in I/Os:
  # levels of recursion $= \log_{\sqrt{m}} n = 2 \log_m n$.
- ☆ $O(n)$ I/Os in each pass $\Longrightarrow O(n \log_m n)$ solution.

# Distribution (Bucket) Sort




- ☆ Select $S = \Theta(m)$ or $\Theta(\sqrt{m}\,)$ partition elements that divide the file evenly into buckets. (See [AV88, §5].)
- ☆ Stream the file through internal memory, forming $S$ subfiles.
- ☆ Independently sort the $S$ subfiles recursively.

$\Longrightarrow$ # levels of recursion $= \log_S n = O(\log_m n)$.
each level of recursion uses $O(n)$ I/Os

$\Longrightarrow$ # I/Os $= O(n \log_m n)$.

# Finding $S = \sqrt{m}$ Partition Elements Deterministically

First we describe a subroutine that finds the $k$th smallest key, for any $k$, in $O(n)$ I/Os:

- ☆ Form $N/M$ sorted memoryloads of $M$ records each.
- ☆ Extract the median key from each memoryload.
  Total of $N/M$ medians.
- ☆ Find the median of the $N/M$ medians using the linear-time RAM algorithm. # I/Os $= O(N/M)$.
- ☆ This median of medians partitions the file into two parts, each with most $3N/4$ records.
- ☆ Recursively apply algorithm to appropriate part.
- ☆ # I/Os geometrically decreases with each level of recursion, $= O(n + N/M) = O(n)$.

## Finding $S = \sqrt{m}$ Partition Elements $b_1$, $b_2$, ...

- ☆ Form $N/M$ sorted memoryloads of $M$ records each.
- ☆ Let $U_j$ be the $j$th sorted memoryload.
  $U' := \bigcup_j \{\text{every other } \frac{S}{4}\text{th key from } U_j\}; \quad |U'| = \frac{4N}{S}$.
- ☆ For each $i = 1, 2, \ldots, S$, use the subroutine to compute
  $$b_i := \text{ the } i(4N/S^2)\text{th key from } U'.$$
- ☆ # I/Os $= O(S|U'|/B) = O(N/B) = O(n)$.
- ☆ $(\#\text{ keys} \le b_{i-1}) \ge (i-1)\frac{4N}{S^2} \cdot \frac{S}{4} = (i-1)\frac{N}{S}$.
- ☆ $(\#\text{ keys} \le b_i) \le i\frac{4N}{S^2} \cdot \frac{S}{4} + \frac{N}{M} \cdot \frac{S}{4} = \frac{iN}{S} + \frac{NS}{4M}$.
- ☆ $(\#\text{ keys between } b_{i-1} \text{ and } b_i) \le \frac{N}{S} + \frac{NS}{4M} \le \frac{5}{4}\frac{N}{S}$
  if $\frac{NS}{M} \le \frac{N}{S} \iff S^2 \le M$. $S = \sqrt{m}$ will work.
- ☆ Similarly, $(\#\text{ keys between } b_{i-1} \text{ and } b_i) \ge \frac{3}{4}\frac{N}{S}$