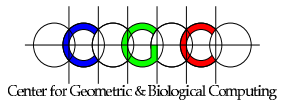
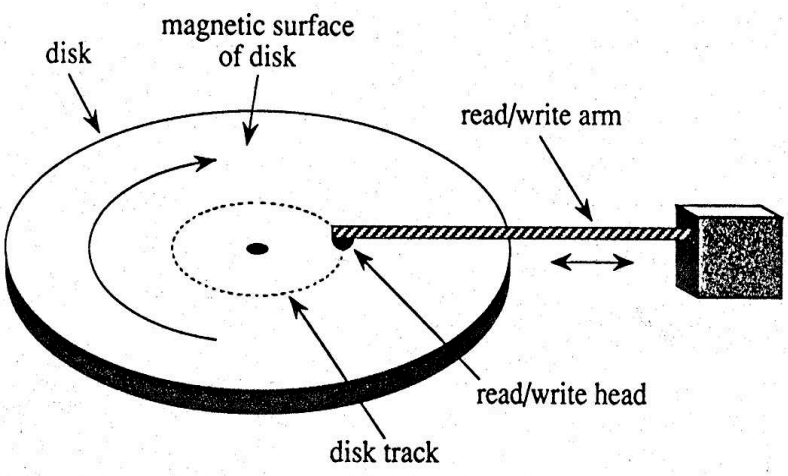
Computing on MASSIVE Data

Jeff Vitter

Duke University



Magnetic Disk Drives as Secondary Memory



- ☆ I/O Crisis! Disk access is 1,000,000 times slower!
- ☆ 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 and take advantage of block transfer. *Focus of this talk!*
- ☆ Solution 2: Parallel disks. (Not our focus today.)

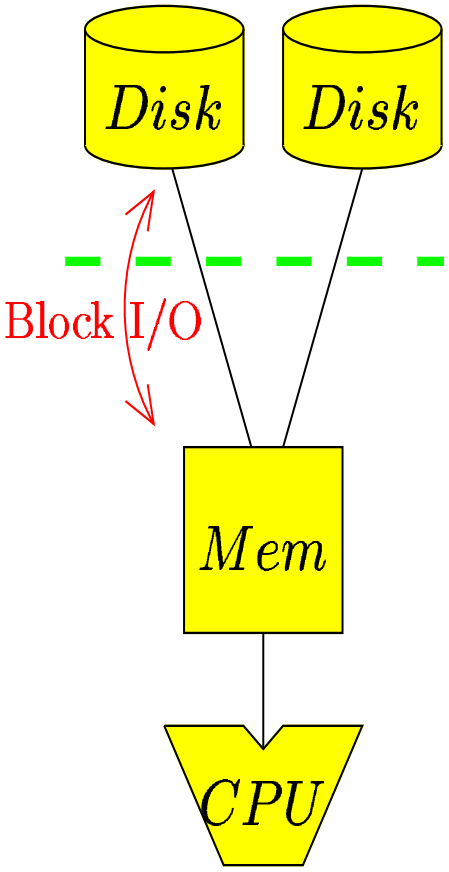
Parallel Disk Model

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

N = problem data size.

M = size of internal memory.

Z = problem output size.

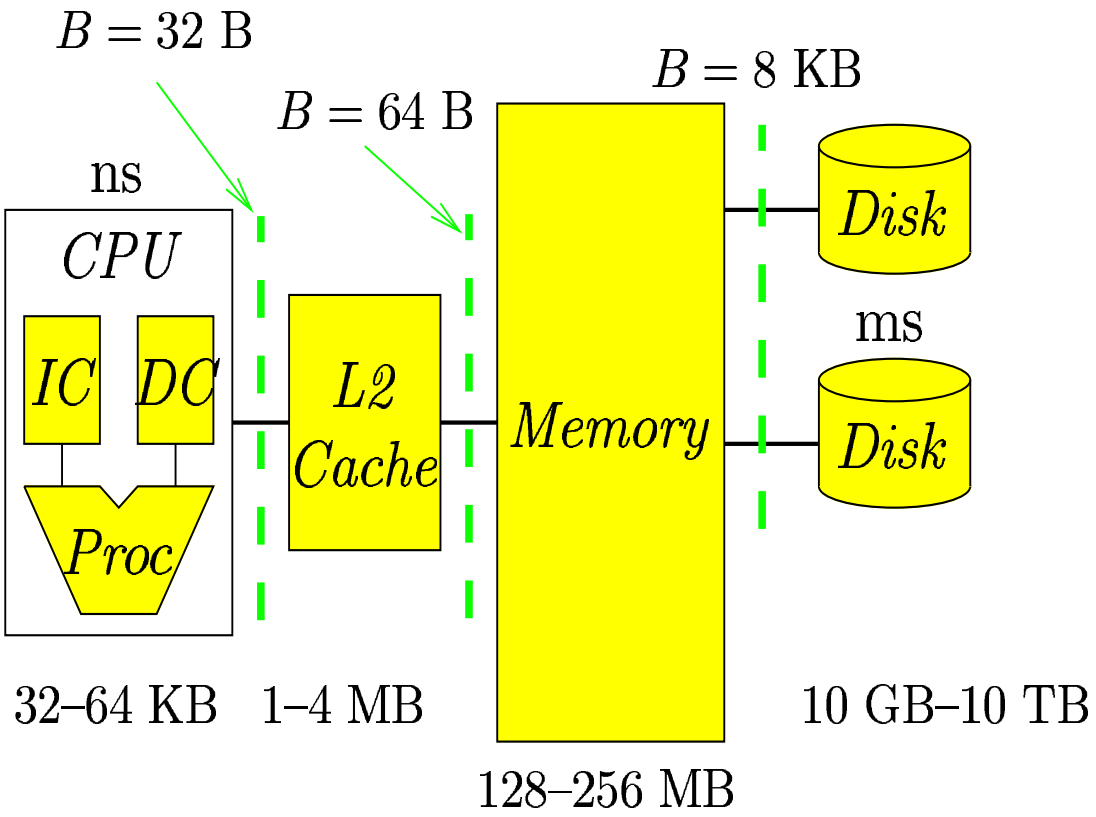


B = size of disk block.

Notational convenience (in units of blocks):

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

A "Real" Machine



Outline

1. Fundamental I/O Bounds.
   - ☆ Merge sort.
   - ☆ Distribution sort.
   - ☆ Searching.
   - ☆ Lower Bounds.
2. Techniques for solving batched geometric problems.
   - ☆ Distribution sweeping.
   - ☆ Red-blue orthogonal rectangle intersection.
   - ☆ Water flow routing and accumulation in terrains.
   - ☆ Empirical results (via TPIE programming environment).
3. Online data structures.
   - ☆ B-trees, buffer trees, R-trees, etc.
   - ☆ Range searching.

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

I/O Lower Bound for Permuting

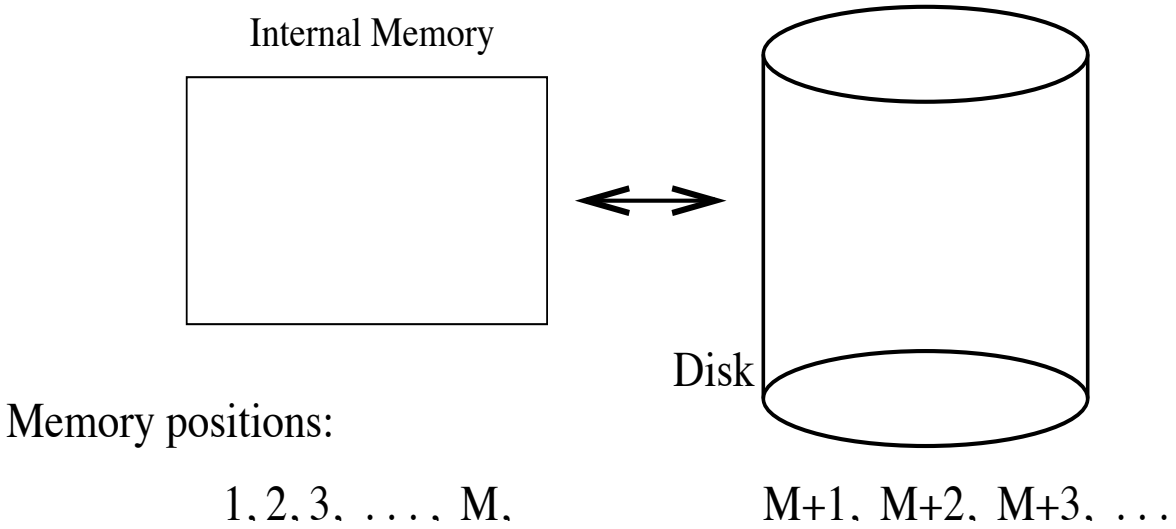
Permuting problem: Given N distinct items from $\{1, 2, \ldots, N\}$, rearrange the N items into sorted order.

- ☆ We will show the lower bound that permuting requires $\Omega\big(\min\{N,\ n \log_m n\}\big)$ I/Os.
- ☆ Typically the min term is $n \log_m n$.
- ☆ Permuting is a special case of sorting.
- ☆ I/O lower bound also applies to sorting. *It is based only upon routing considerations, since the order is already known.*
- ☆ For the pathological case when $N < n \log_m n$, we can show that sorting requires $\Omega(n \log_m n)$ I/Os in comparison model.
- ☆ In the RAM model, permutation takes only $O(N)$ time. But in I/O model, it (and most interesting problems) require sorting complexity (except for pathological case)!

I/O Lower Bound for Permuting

Goal: See how many I/O steps T are needed so that any of the $N!$ permutations of the N items can be realized.

We say that a permutation is realizable if it appears in extended memory in the required order.



Tactic: Determine how much the tth I/O step can increase the number of possible realizable permutations.

Lower Bound on External Sorting

Theorem 1 *External sorting requires $\Omega(n \log_m n)$ I/Os in the comparison I/O model (comparisons only allowed operations in internal memory).*

Proof: We have N records to sort, therefore there are $N!$ possibilities for the correct ordering that are consistent with the information we have from the start (which is none). The idea is now to see how much we can narrow down this number using one input operation and whatever number of comparisons we want, under the assumption that an adversary chooses the worst possible outcome of the comparisons we perform. Needless to say, output operations cannot contribute to narrowing down the possibilities because any information we can get after the output we could have obtained before the output.

Consider an input of B records into internal memory. Assuming that we know the order of the records already in internal memory, but not the order of the B newly read records, there are at most $\binom{M}{B}(B!)$ possible orderings of the records in internal memory. If S denotes the number of possible orderings before the input, there exists at least one of the $\binom{M}{B}(B!)$ orderings of records in internal memory, such that the number of total orderings (of the initial N records) consistent with this ordering, is at least $\frac{S}{\binom{M}{B}(B!)}$. The adversary always chooses one such ordering. It follows that after t inputs, the number of possible orderings is at least $\frac{N!}{\left(\binom{M}{B}(B!)\right)^t}$.

The above was under the assumption that we did not know the order of the B records read into internal memory. This is not the case if the B records have been together in internal memory previously, because we always determine the order of the records in internal memory after an input. The number of times we can read B records that have not previously been together in internal memory cannot exceed $\frac{N}{B}$. It follows that after t input operations there are at least $\frac{N!}{\binom{M}{B}^t (B!)^{\frac{N}{B}}}$ orderings consistent with the information obtained from the adversary.

We want to narrow the possible orderings down to 1, and the number of I/O-operations needed to do this must therefore be the least t such that $\frac{N!}{\binom{M}{B}^t (B!)^{\frac{N}{B}}} \leq 1$. Using the (rough) assumptions that $\log x! = x \log x$ (Stirlings formula) and $\log \binom{M}{B} = B \log \frac{M}{B}$ we then get the following.

$$
\begin{array}{rcll}
\frac{N!}{\binom{M}{B}^t (B!)^{\frac{N}{B}}} & \leq & 1 & \Downarrow \\
\binom{M}{B}^t (B!)^{\frac{N}{B}} & \geq & N! & \Downarrow \\
t \log \binom{M}{B} + \frac{N}{B} \log(B!) & \geq & \log(N!) & \Downarrow \\
tB \log(\frac{M}{B}) + \frac{N}{B} B \log B & \geq & N \log N & \Downarrow \\
tB \log(\frac{M}{B}) & \geq & N \log(\frac{N}{B}) & \Downarrow \\
t & \geq & \frac{N}{B} \frac{\log(\frac{N}{B})}{\log(\frac{M}{B})} & \Downarrow \\
t & \geq & n \log_m n &
\end{array}
$$

□

I/O Lower Bound for Permuting

Assumption: the N items to permute are indivisible.

Number of ways to choose B blocks out of M total memory blocks = $\binom{M}{B}$

realizable permutations after tth I/O

$$= \begin{cases} \binom{M}{B} \times (\text{\# realizable permutations after } (t-1)\text{st I/O}) \\ \qquad\qquad \text{if block was previously accessed} \\ B! \times \binom{M}{B} \times (\text{\# realizable permutations after } (t-1)\text{st I/O}) \\ \qquad\qquad \text{if this is first access to block} \end{cases}$$

Number of ways to permute B blocks: B!

There are N/B blocks initially unaccessed.

$$\text{\# choices for block accessed in } t\text{th I/O} = \left(\frac{N}{B} + t\right) \leq N^2.$$

Notes:

$\binom{M}{B}$ = Number of ways to choose B memory locations out of M total memory

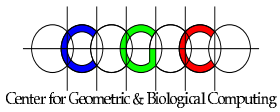
B! = Number of ways to permute B elements of a block

Number T of required I/Os for Permuting

$$(B!)^{N/B}\left(\binom{M}{B}N^2\right)^T \geq N!$$

Taking logs and applying Stirling's approximation:

$$\begin{aligned}
\frac{N}{B}\log B! + T\left(\log\binom{M}{B} + \log N\right) &= \Omega(\log N!) \\
\frac{N}{B}(B\log B) + T\left(B\log\frac{M}{B} + \log N\right) &= N\log N \\
T\left(B\log\frac{M}{B} + \log N\right) &= N\log N - N\log B \\
&= N\log\frac{N}{B} \\
T &= \Omega\left(\min\left\{N,\ \frac{N}{B}\frac{\log(N/B)}{\log(M/B)}\right\}\right) \\
&= \Omega\left(\min\left\{N,\ n\log_m n\right\}\right)
\end{aligned}$$



Notes: $n = \frac{N}{B},\ m = \frac{M}{B}$.

Stirling's Bounds for Factorial N! : Number of ways to permute N keys:

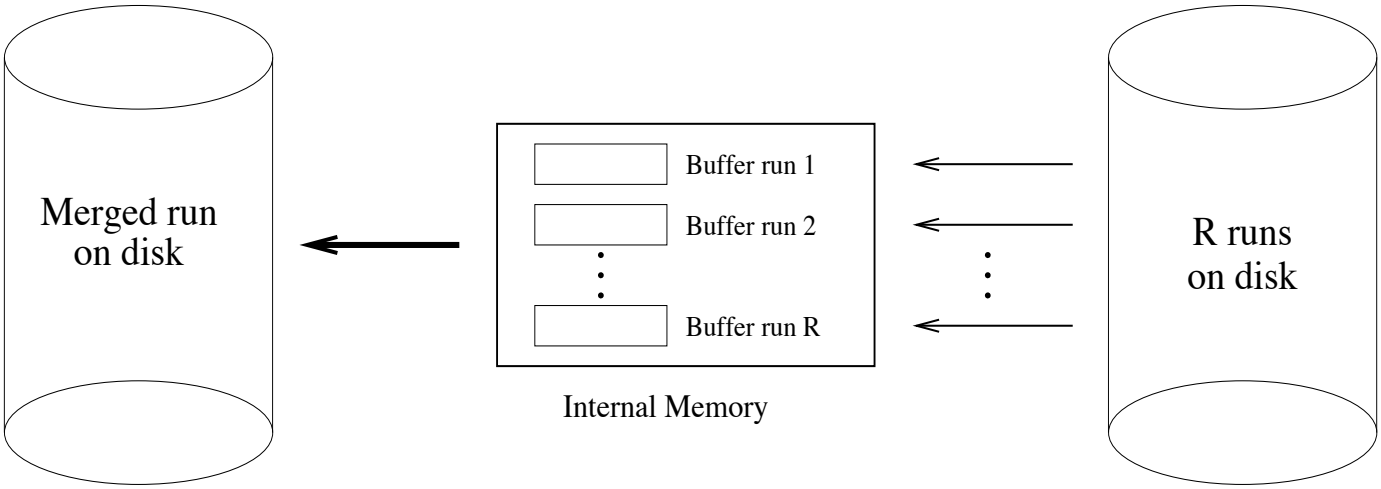
$N! \sim \sqrt{2\pi N}\left(\frac{N}{e}\right)^N$ and taking logs gives: $\log N! \sim N\log N$

Bounds for Factorial B! : Number of ways to permute B elements of a block: log B! $\sim$ B log B

Number of ways to choose B memory locations out of M total memory:

$\binom{M}{B} \sim \left(\frac{M}{B}\right)^B$ and taking logs gives: $\log\binom{M}{B} \sim B\log\frac{M}{B}$

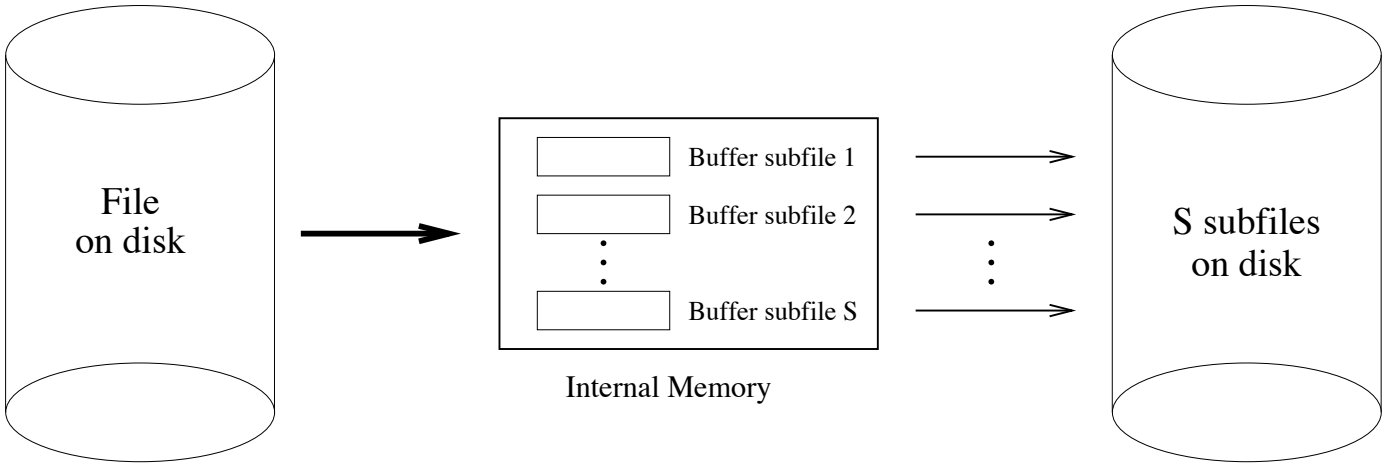
Merge Sort



- ☆ Form n/m initially sorted runs,
  each consisting of m blocks (i.e., one memoryload).
- ☆ Reserve $R = \Theta(m)$ buffers, each of size B, in internal memory.
  Repeatedly merge together R runs at a time
  $$\Longrightarrow \quad \#\text{ passes} = \log_R \frac{n}{m} = \log_m n - 1$$
- ☆ Each pass uses $\Theta(n)$ I/Os
  $$\Longrightarrow \quad \#\text{ I/Os} = O\left(n \log_m n\right).$$

Notes: $n = \dfrac{N}{B}, \quad m = \dfrac{M}{B}$.

Distribution (Bucket) Sort



☆ Select $S = \Theta(m)$ or $\Theta(\sqrt{m}\,)$ partitioning 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.

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

Batched Problems in Geometry

[GTVV93], [AVV95], [APRSV98a], [APRSV98b], [CFMMR98]

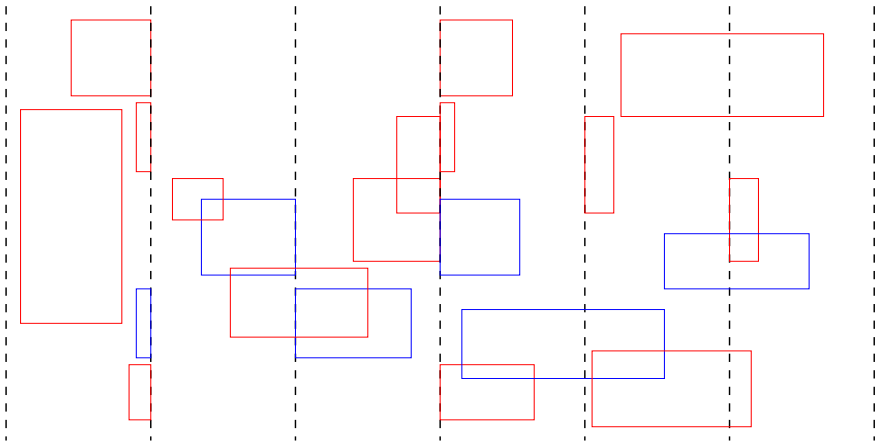
- ☆ Orthogonal rectangle intersection.
- ☆ Red-blue line segment intersection.
- ☆ General line segment intersection.
- ☆ All nearest neighbors.
- ☆ 2-D and 3-D convex hulls.
- ☆ Batched range queries.
- ☆ Trapezoid decomposition
- ☆ Batched planar point location.
- ☆ Triangulation.

Use of virtual memory $\Longrightarrow$ $\Omega\left(N \log_B N + Z\right)$ I/Os. *Bad !!!*

We can improve this to $O\left(n \log_m n + z\right)$ I/Os using

- ☆ Distribution sweep.
- ☆ Batched filtering.
- ☆ Random incremental construction.
- ☆ Parallel simulation.

Sketch of External Solution [APRSV98]:



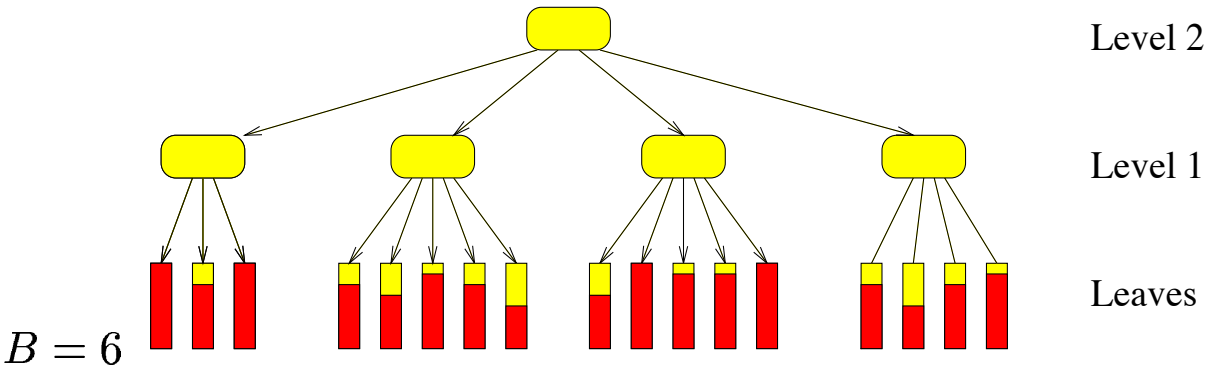
1. Divide plane into $\sqrt{m}$ slabs, each with $O(N/\sqrt{m})$ endpoints.
2. Find Z' intersections involving the part of a rectangle completely spanning slabs.
3. Recursively solve problem in each slab.

☆ $O(\log_{\sqrt{m}} n) = O(\log_m n)$ levels of recursion.

☆ Performing Step 2 in $O\left(n + \frac{Z'}{B}\right)$ I/Os

$\Longrightarrow$ $O(n \log_m n + z)$ I/Os total.

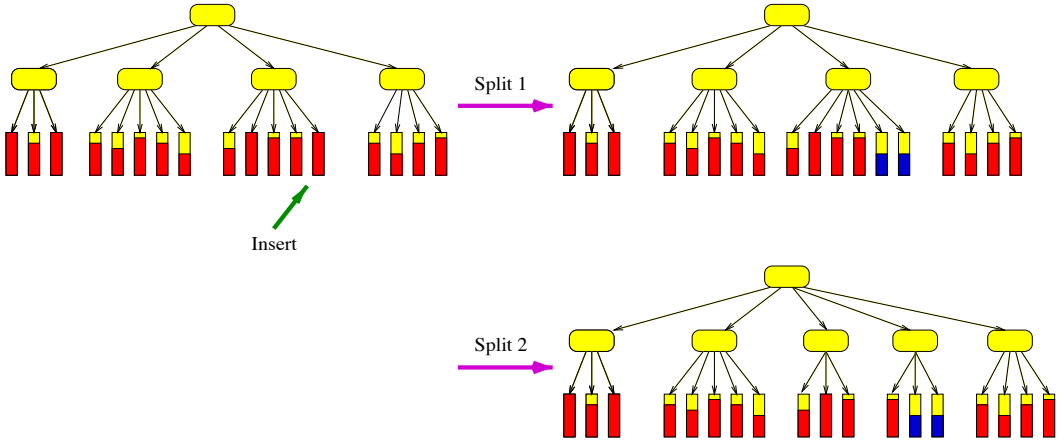
B-Trees [Bayer-McCreight72]



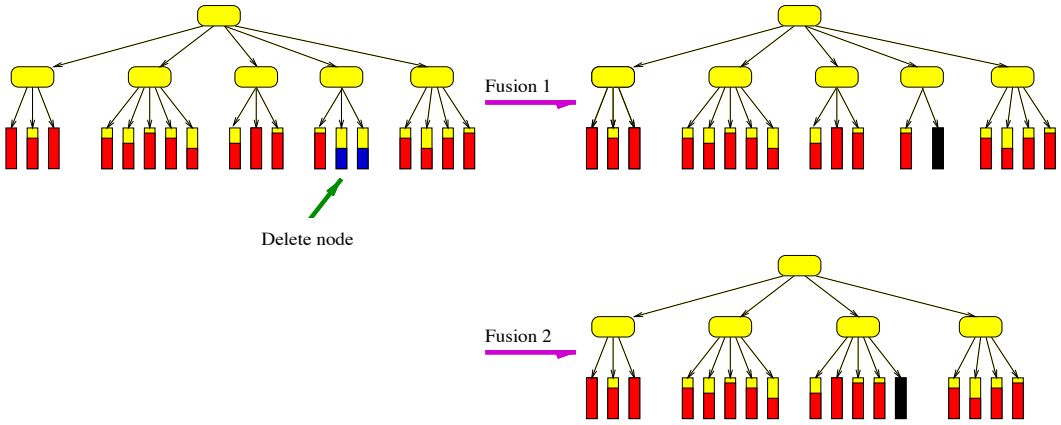
- ☆ Each node is stored in one disk block.
- ☆ Leaf nodes store all the data items.
- ☆ Internal nodes store keys and pointers to guide the searching.
- ☆ $\frac{B}{2} \leq$ node degree $\leq B - 1$, except for the root.
- ☆ One I/O per level
  $\Longrightarrow \quad O(\log_B N + z)$ I/Os for a 1-D range query.

B-Trees—Updates in $O(\log_B N)$ I/Os

Insertion may cause splits:



Deletions may cause nodes to combine:



Sharing also possible (good packing).

Batched Dynamic Processing

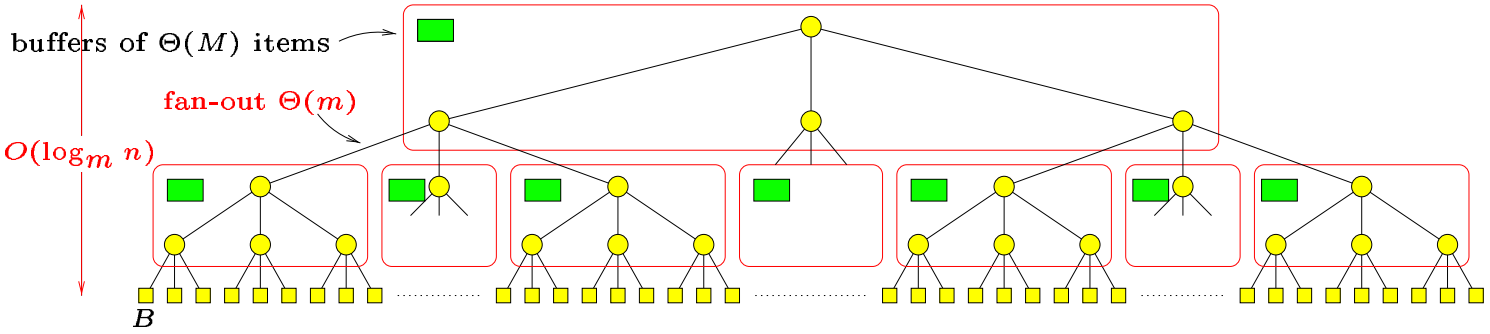
Motivation: Sometimes we may want to handle inserts, deletes, and queries in batches. In RAM algorithms, we can use an optimal dynamic data structure and get optimal results.
But NOT in external memory!!!

Example: Sorting by inserting and deleting into a priority queue.

- ☆ Insert items one at a time into priority queue.
- ☆ Repeat: Find item in priority queue with minimum key. Delete it from priority queue (*delete_min* operation) and output it.
- ☆ N inserts into priority queue
  $\Longrightarrow$ $O(N \log_B N)$ I/Os. *Bad !!!*
- ☆ We want $O(n \log_m n)$ I/Os. *Good !!!*

Another example: Sweep-line algorithm for finding rectangle intersections.

The Buffer Tree [Arge95]



Main idea: Combine degree-B nodes to form supernodes of degree $m = M/B$. Each supernode gets a buffer of size M.

- ☆ Insertions are done "lazily": Items are inserted into buffers. When a buffer runs full, its items are pushed one level down.
- ☆ Buffer-emptying of M items takes $O(m)$ I/Os
  $\Longrightarrow$ $O(m/M) = O(1/B)$ I/Os per item per level
  $\Longrightarrow$ N items inserted in $O\left(\frac{N}{B}\log_m n\right) = O(n \log_m n)$ I/Os.

Online Search

B-trees are an optimal data structure for the dictionary problem and one-dimensional range queries:

- ☆ $O(\log_B N + z)$ I/Os per query.
- ☆ $O(\log_B N)$ I/Os per update.
- ☆ $O(n)$ disk blocks of space.

We want similar performance for harder problems:

- ☆ Proximity queries, nearest neighbor, clustering
- ☆ Point location, ray shooting
- ☆ Range searching in 2-D, 3-D, orthogonal, halfspace

Two very useful paradigms:

1. Filtering [Chazelle86]: You can afford to do an extra I/O during search if you can charge it to $\Theta(B)$ query outputs.
2. Bootstrapping [AV96, ASV99]
   - ☆ Use an external version of an efficient internal-memory data structure as the global search structure.
   - ☆ Within the structure, use efficient static data structures on smaller instances (of size $O(B^2)$) of the problem.