I/O-Algorithms

Lars Arge

Spring 2012

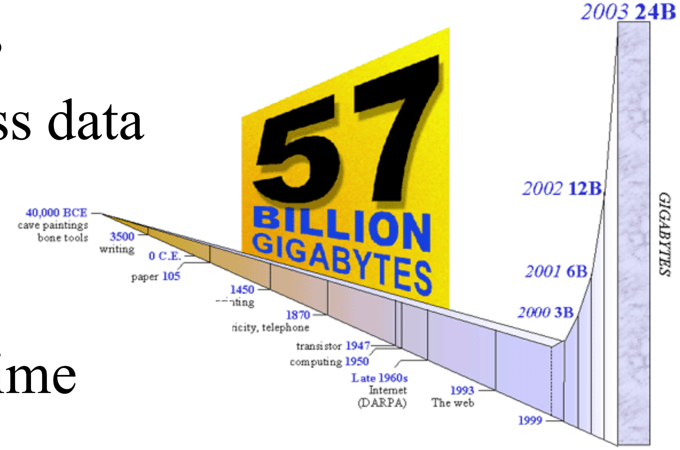
January 31, 2012

Massive Data

- Pervasive use of computers and sensors
- Increased ability to acquire/store/process data

  → Massive data collected everywhere
- Society increasingly "data driven"

  → Access/process data anywhere any time



Nature/Science special issues:

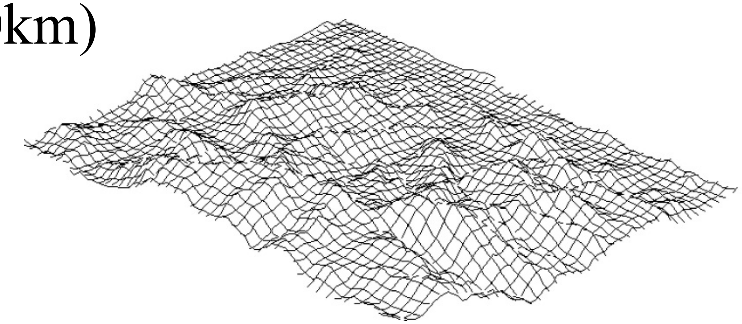
- 2/06, 9/08, 2/11
- Scientific data size growing exponentially, while quality and availability improving
- Paradigm shift: *Science will be about mining data*

Obviously recently in sciences:

- Economist 02/10:
  - From 150 Billion Gigabytes five years ago to 1200 Billion today
  - Managing data deluge difficult; doing so will transform business/public life

Example: Grid Terrain Data

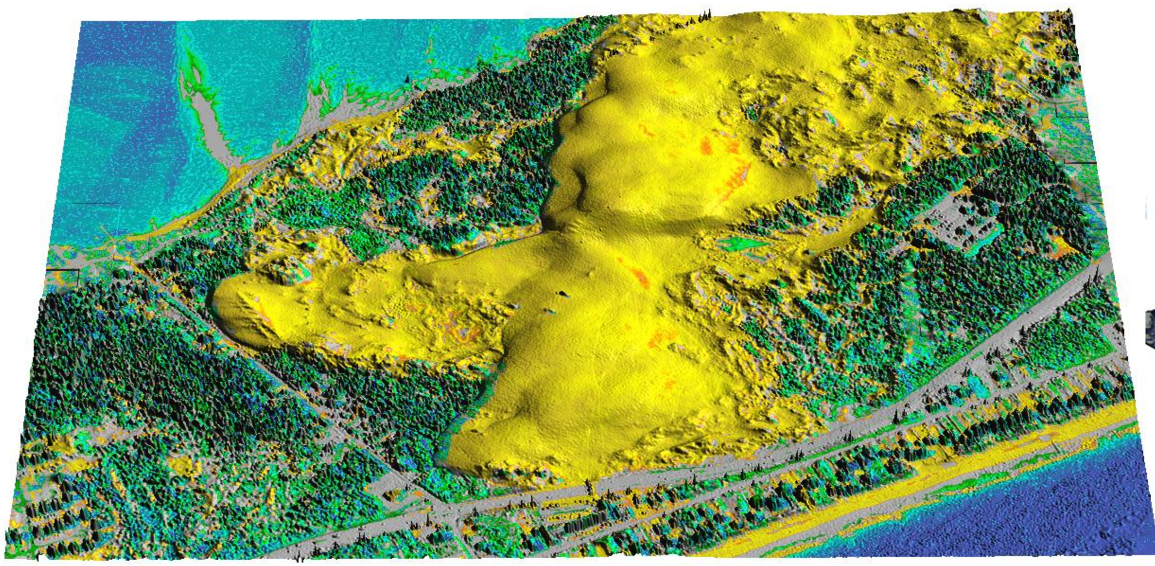
- Appalachian Mountains (800km x 800km)

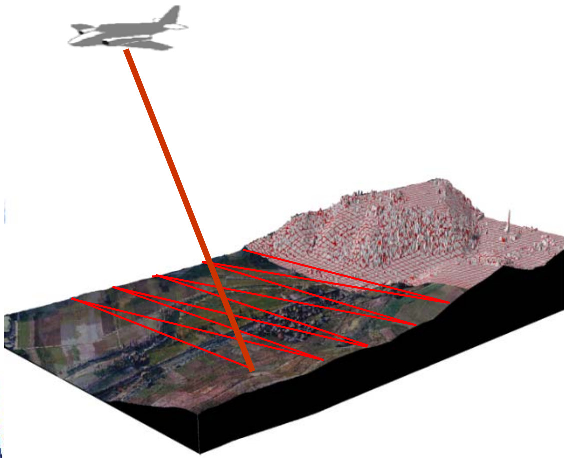


  - 100m resolution ⇒ ~ 64M cells
    ⇒ ~128MB raw data (~500MB when processing)
  - ~ 1.2GB at 30m resolution
    NASA SRTM mission acquired 30m
    data for 80% of the earth land mass
  - ~ 12GB at 10m resolution (much of US available from USGS)
  - ~ 1.2TB at 1m resolution (quickly becoming available)
  - …. 10-100 points per m^2 already possible

Example: LIDAR Terrain Data

- Massive (irregular) point sets (~1m resolution)
  - Becoming relatively cheap and easy to collect

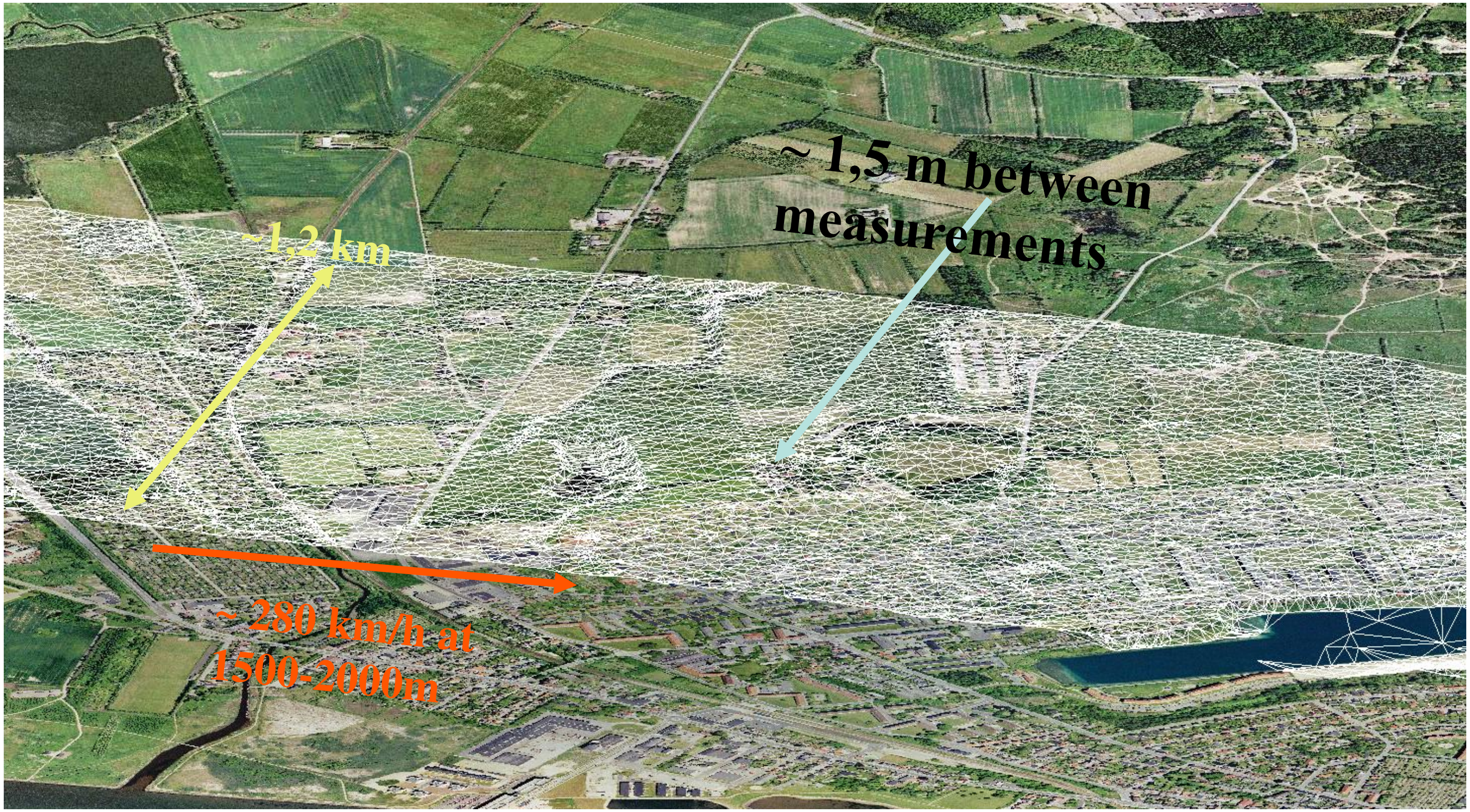




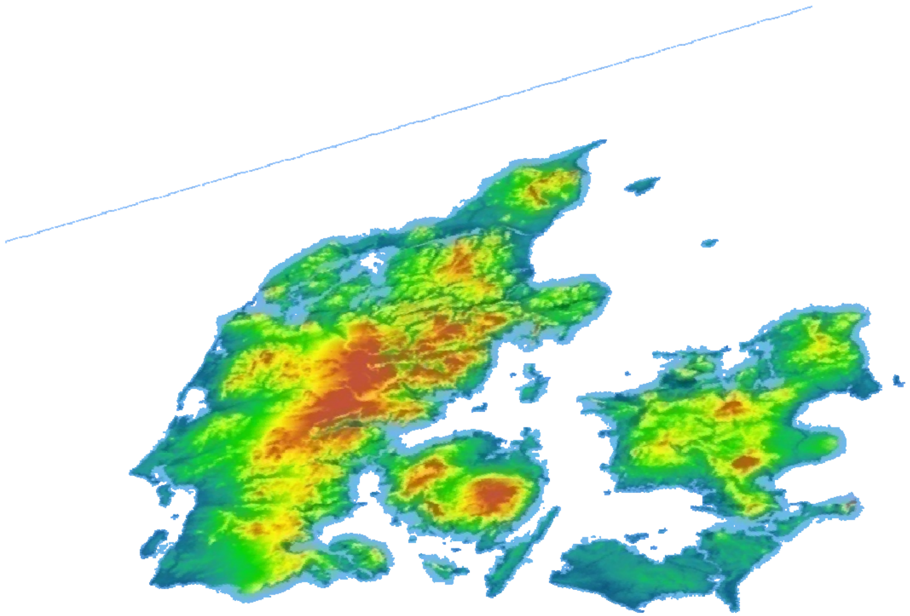
- Sub-meter resolution using mobile mapping

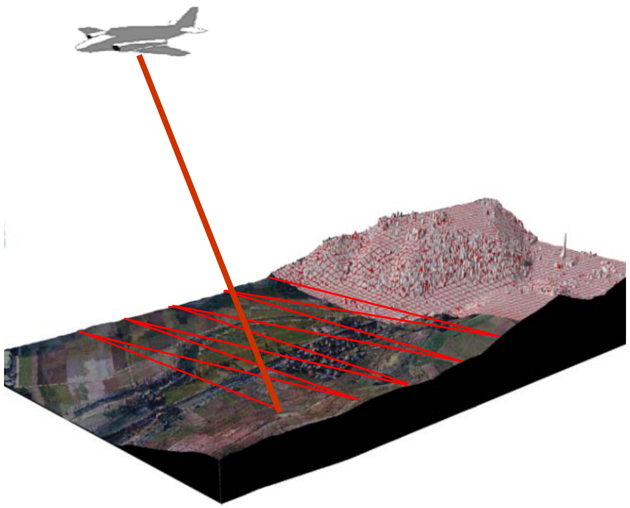
Example: LIDAR Terrain Data

- COWI A/S (and others) have scanned Denmark



Example: LIDAR Terrain Data



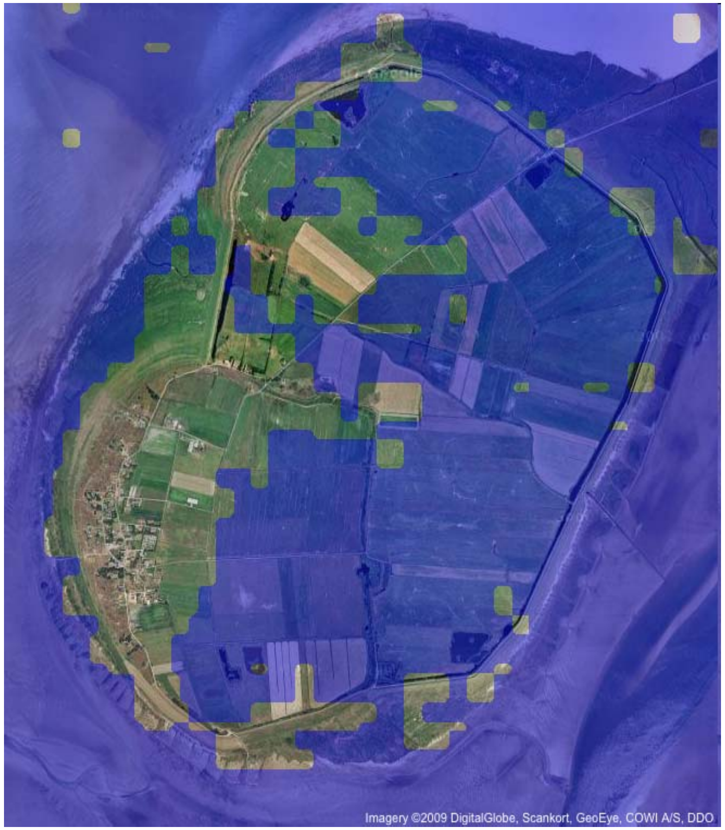


- ~2 million points at 30 meter (<1GB)
- ~18 billion points at 1 meter (>1TB)

Application Example: Flooding Prediction

Example: Detailed Data Essential

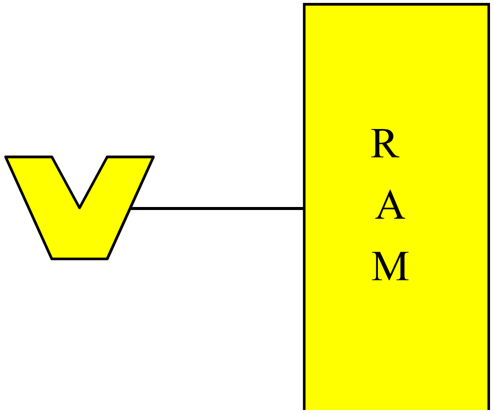
- Mandø with 2 meter sea-level raise



80 meter terrain model

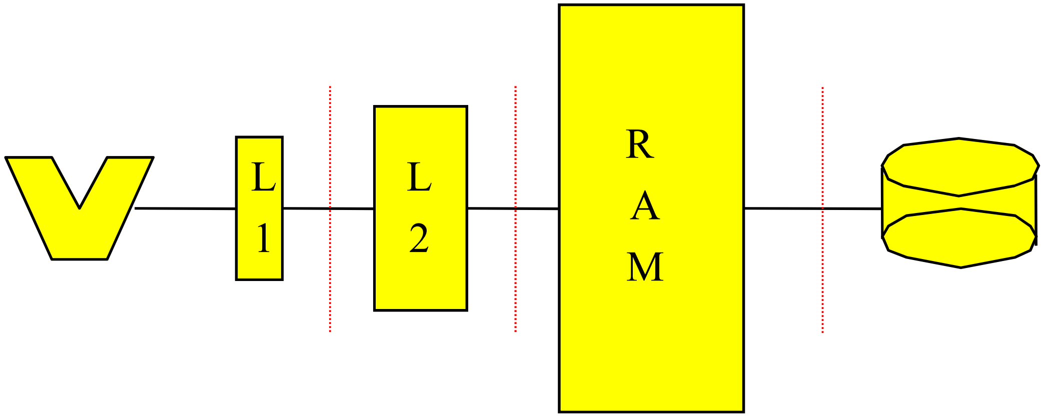
2 meter terrain model

Random Access Machine Model



- Standard theoretical model of computation:
  - Infinite memory
  - Uniform access cost
- Simple model crucial for success of computer industry

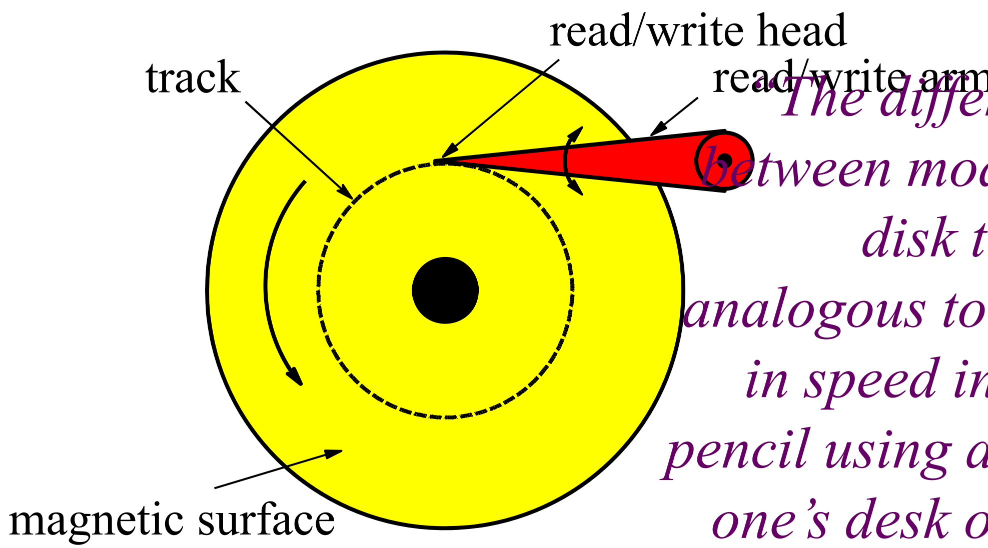
Hierarchical Memory



- Modern machines have complicated memory hierarchy
  - Levels get larger and slower further away from CPU
  - Data moved between levels using large blocks

Slow I/O

- Disk access is 10^6 times slower than main memory access



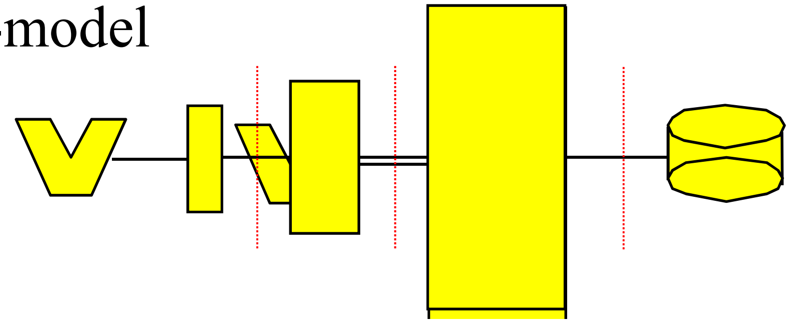
The difference in speed between modern CPU and disk technologies is analogous to the difference in speed in sharpening a pencil using a sharpener on one's desk or by taking an airplane to the other side of the world and using a sharpener on someone else's desk. (D. Comer)

  – Disk systems try to amortize large access time transferring large contiguous blocks of data (8-16Kbytes)

- Important to store/access data to take advantage of blocks (locality)

Scalability Problems

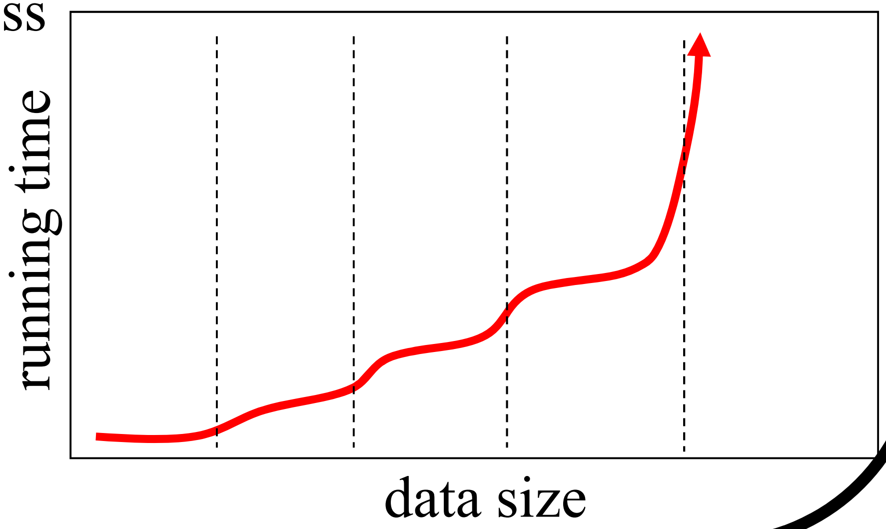
- Most programs developed in RAM-model
  - Run on large datasets because OS moves blocks as needed



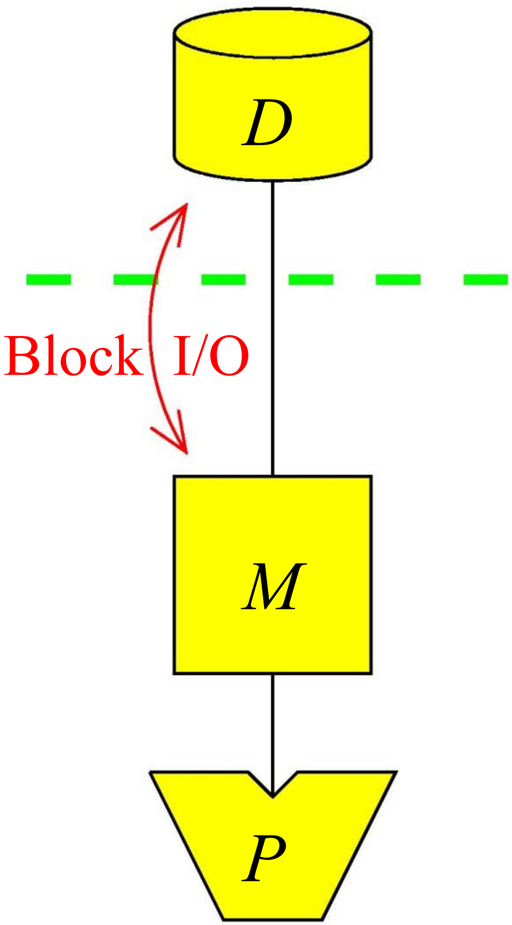
- Moderns OS utilizes sophisticated paging and prefetching strategies
  - But if program makes scattered accesses even good OS cannot take advantage of block access

⇓

Scalability problems!



External Memory Model



$N =$ # of items in the problem instance

$B =$ # of items per disk block

$M =$ # of items that fit in main memory

$T =$ # of items in output

I/O: Move block between memory and disk

We assume (for convenience) that $M > B^2$

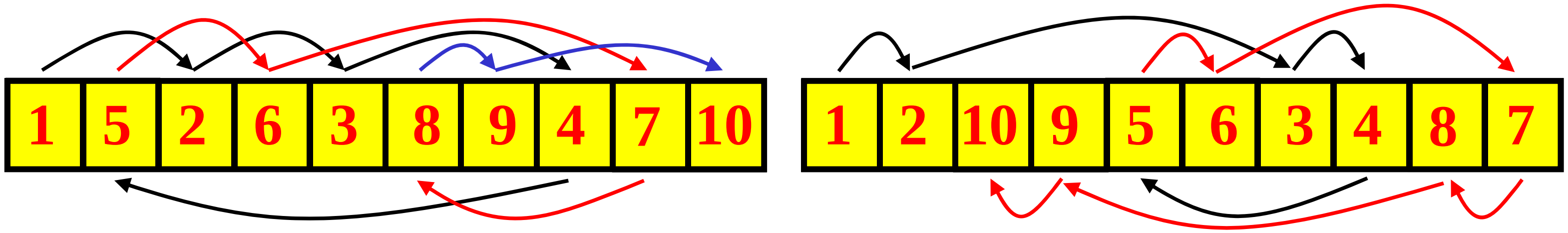
Fundamental Bounds

	Internal	External
• Scanning:	N	$\frac{N}{B}$
• Sorting:	$N \log N$	$\frac{N}{B} \log_{M/B} \frac{N}{B}$
• Permuting	N	$\min\{N, \frac{N}{B} \log_{M/B} \frac{N}{B}\}$
• Searching:	$\log_2 N$	$\log_B N$

- Note:
  - Linear I/O: $O(N/B)$
  - Permuting not linear
  - Permuting and sorting bounds are equal in all practical cases
  - B factor VERY important: $\frac{N}{B} < \frac{N}{B} \log_{M/B} \frac{N}{B} << N$
  - Cannot sort optimally with search tree

Scalability Problems: Block Access Matters

- Example: Traversing linked list (List ranking)
  - Array size N = 10 elements
  - Disk block size B = 2 elements
  - Main memory size M = 4 elements (2 blocks)



Algorithm 1: N=10 I/Os

Algorithm 2: N/B=5 I/Os

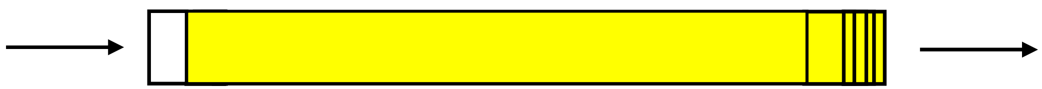
- Large difference between N and N/B large since block size is large
  - Example: $N = 256 \times 10^6$, $B = 8000$, $1ms$ disk access time

    $\Rightarrow$ N I/Os take 256×10^3 $sec = 4266$ $min = 71$ hr

    $\Rightarrow$ N/B I/Os take $256/8$ $sec = 32$ sec

Queues and Stacks

- Queue:
  – Maintain push and pop blocks in main memory

Push  Pop

$\Downarrow$

$O(1/B)$ Push/Pop operations

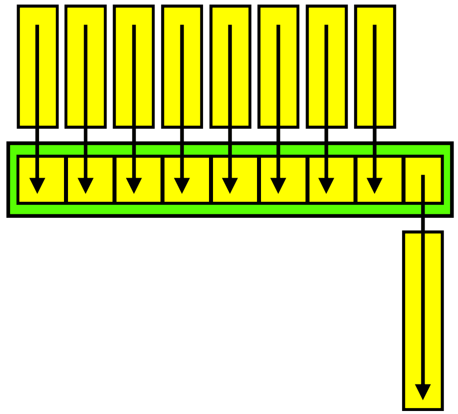
- Stack:
  – Maintain push/pop blocks in main memory

$\Downarrow$

$O(1/B)$ Push/Pop operations

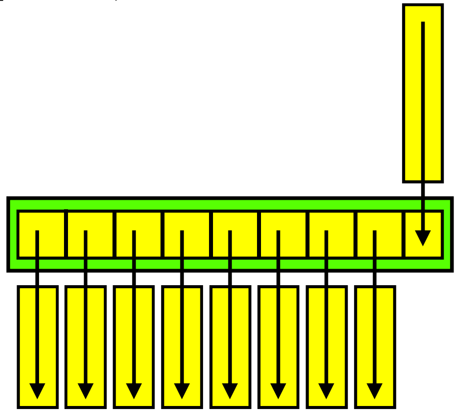
Sorting

- $<M/B$ sorted lists (queues) can be merged in $O(N/B)$ I/Os



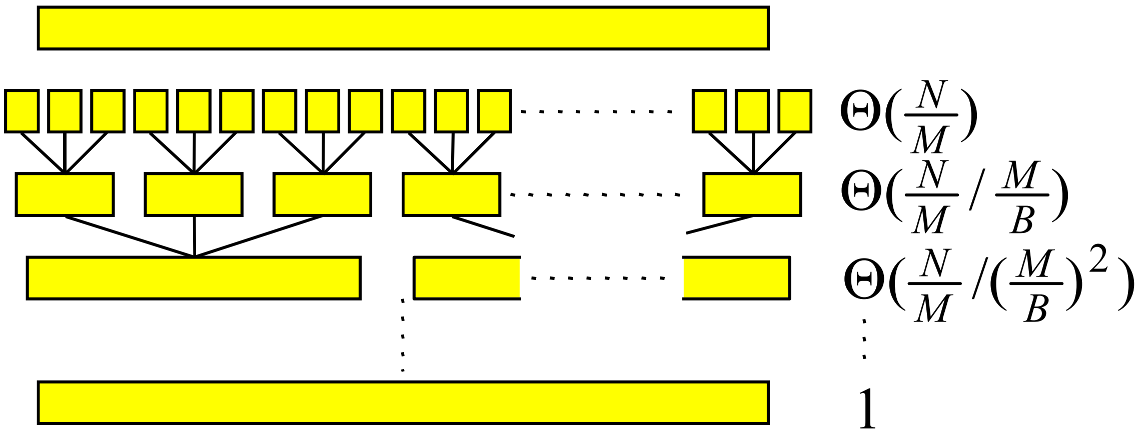
M/B blocks in main memory

- Unsorted list (queue) can be distributed using $<M/B$ split elements in $O(N/B)$ I/Os



Sorting

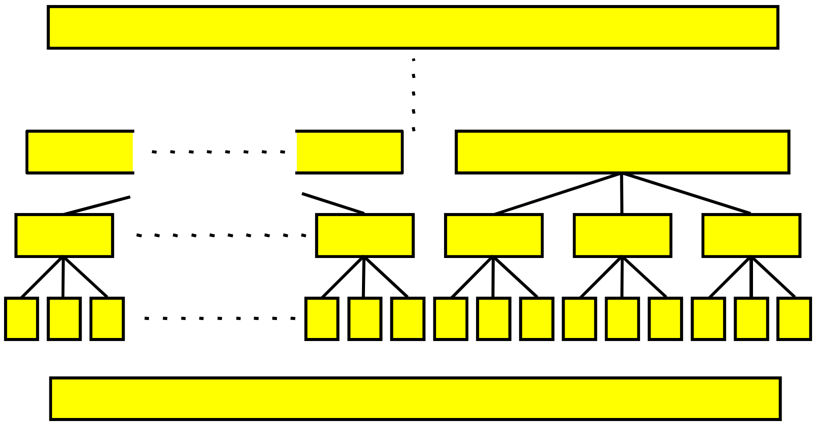
- Merge sort:
  - Create N/M memory sized sorted lists
  - Repeatedly merge lists together $\Theta(M/B)$ at a time



$\Rightarrow O(\log_{M/B} \frac{N}{M})$ phases using $O(N/B)$ I/Os each $\Rightarrow O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/Os

Sorting

- Distribution sort (multiway quicksort):
  - Compute $\Theta(M/B)$ splitting elements
  - Distribute unsorted list into $\Theta(M/B)$ unsorted lists of equal size
  - Recursively split lists until fit in memory



$\Rightarrow O(\log_{M/B} \frac{N}{M})$ phases

$\Rightarrow O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/Os if splitting elements computed in $O(N/B)$ I/Os

Computing Splitting Elements

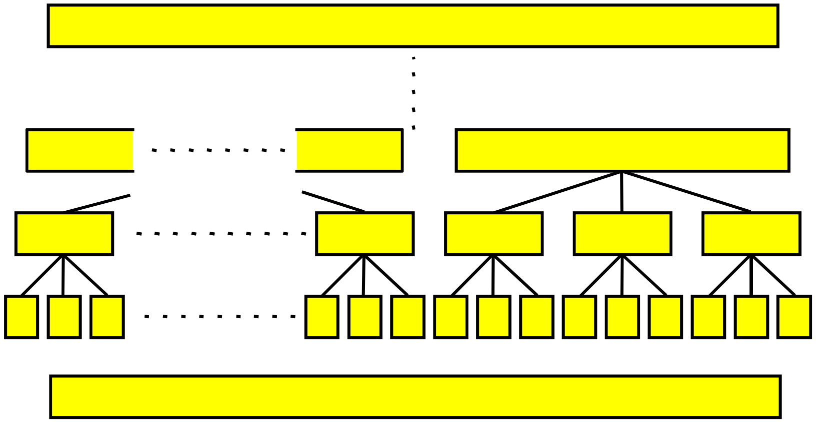
- In internal memory (deterministic) quicksort split element (median) found using linear time selection

- Selection algorithm: Finding *i*'th element in sorted order
  1) Select median of every group of *5* elements
  2) Recursively select median of ~ *N/5* selected elements
  3) Distribute elements into two lists using computed median
  4) Recursively select in one of two lists
- Analysis:
  - Step 1 and 3 performed in $O(N/B)$ I/Os.
  - Step 4 recursion on at most $\sim\frac{7}{10}N$ elements

$$\Rightarrow T(N) = O(N/B) + T(N/5) + T(7N/10) = O(N/B) \text{ I/Os}$$

Sorting

- Distribution sort (multiway quicksort):



- Computing splitting elements:
  - $\Theta(M/B)$ times linear I/O selection $\Rightarrow O(NM/B^2)$ I/O algorithm
  - But can use selection algorithm to compute $\sqrt{M/B}$ splitting elements in $O(N/B)$ I/Os, partitioning into lists of size $< \frac{3}{2}\frac{N}{\sqrt{M/B}}$

$\Rightarrow O(\log_{\sqrt{M/B}} \frac{N}{M}) = O(\log_{M/B} \frac{N}{M})$ phases $\Rightarrow O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ algorithm

Computing Splitting Elements

1) Sample $\frac{4N}{\sqrt{M/B}}$ elements:
   - Create N/M memory sized sorted lists
   - Pick every $\frac{1}{4}\sqrt{M/B}$'th element from each sorted list

2) Choose $\sqrt{M/B}$ split elements from sample:
   - Use selection algorithm $\sqrt{M/B}$ times to find every $\frac{4N}{\sqrt{M/B}} \Big/ \sqrt{M/B} = \frac{4N}{M/B}$'th element

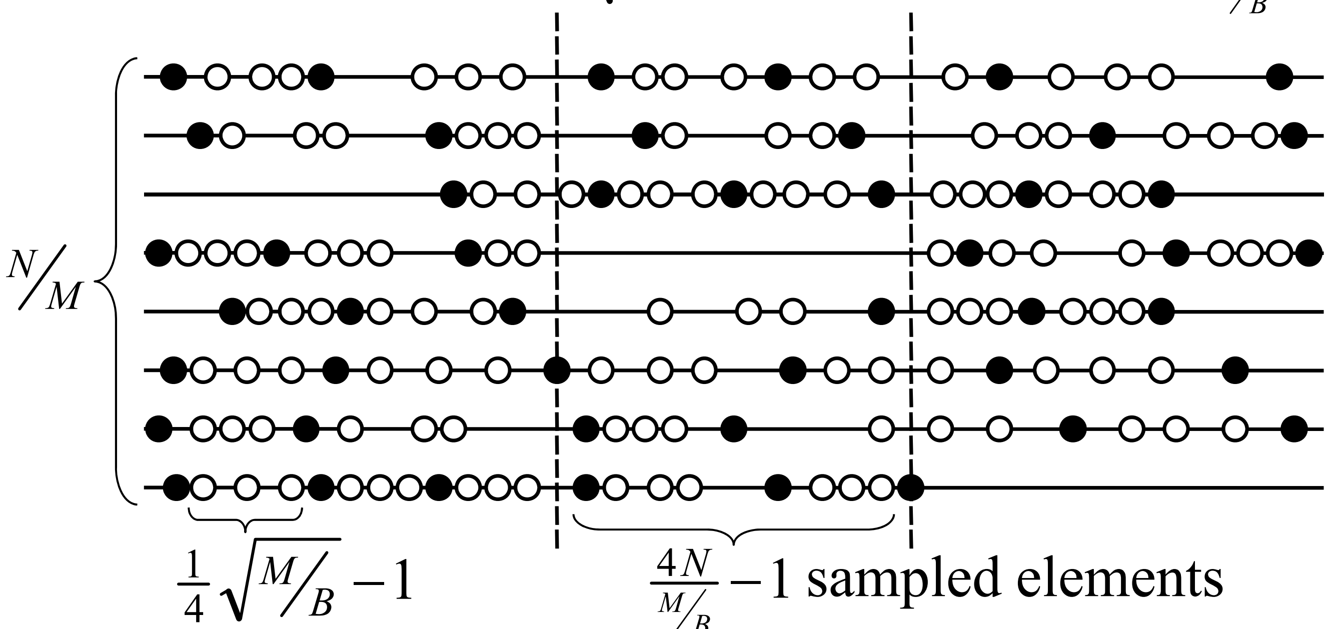
- Analysis:
  - Step 1 performed in $O(N/B)$ I/Os
  - Step 2 performed in $\sqrt{M/B} \cdot O(\frac{N}{\sqrt{M/B}B}) = O(N/B)$ I/Os

  $\Rightarrow O(N/B)$ I/Os

Computing Splitting Elements

1) Sample $\frac{4N}{\sqrt{M/B}}$ elements:
- Create N/M memory sized sorted lists
- Pick every $\frac{1}{4}\sqrt{M/B}$'th element from each sorted list

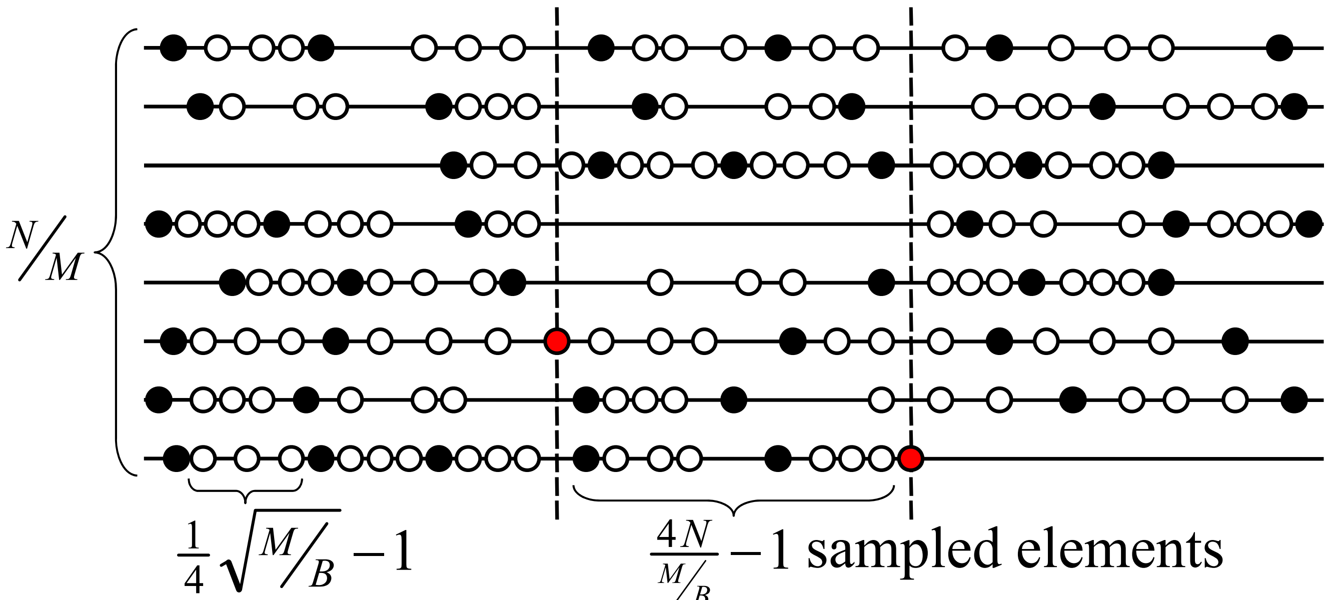
2) Choose $\sqrt{M/B}$ split elements from sample:
- Use selection algorithm $\sqrt{M/B}$ times to find every $\frac{4N}{M/B}$'th element



Computing Splitting Elements

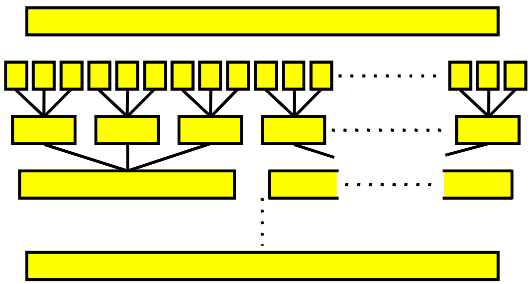
- Elements in range R defined by consecutive split elements
  - Sampled elements in R: $\frac{4N}{M/B} - 1$
  - Between sampled elements in R: $(\frac{4N}{M/B} - 1) \cdot (\frac{1}{4}\sqrt{M/B} - 1)$
  - Between sampled element in R and outside R: $2\frac{N}{M} \cdot (\frac{1}{4}\sqrt{M/B} - 1)$

$$\Rightarrow \; < \frac{4N}{M/B} + \left(\frac{N}{\sqrt{M/B}} - \frac{4N}{M/B}\right) + \frac{N}{2B\sqrt{M/B}} < \frac{3}{2}\frac{N}{\sqrt{M/B}}$$



Summary/Conclusion: Sorting

- External merge or distribution sort takes $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/Os
  - Merge-sort based on M/B-way merging
  - Distribution sort based on $\sqrt{M/B}$-way distribution and partition elements finding



- Optimal
  - As we will prove next time

References

- **Input/Output Complexity of Sorting and Related Problems**
  A. Aggarwal and J.S. Vitter. *CACM* 31(9), 1998

- **External partition element finding**
  Lecture notes by L. Arge and M. G. Lagoudakis.

Project 1: Implementation of Merge Sort