Improved Visibility Computation on Massive Grid Terrains

Jeremy Fishman
Bowdoin College
USA

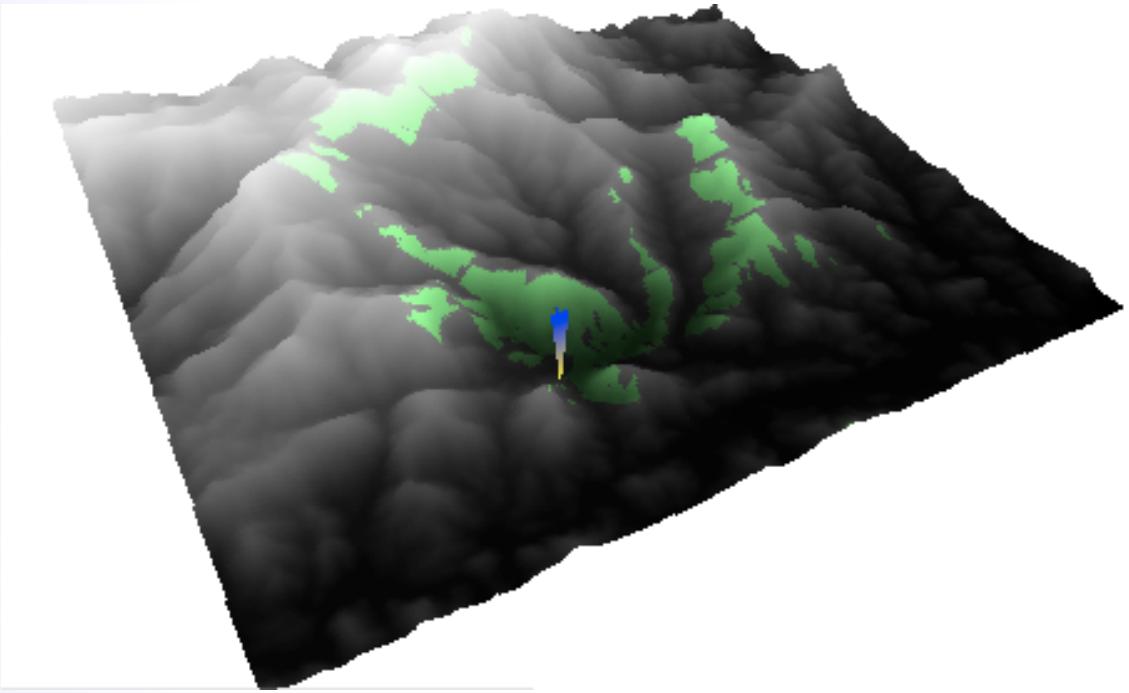
Herman Haverkort
Eindhoven University
The Netherlands

Laura Toma
Bowdoin College
USA

Laura Toma
ACM GIS 2009

Visibility

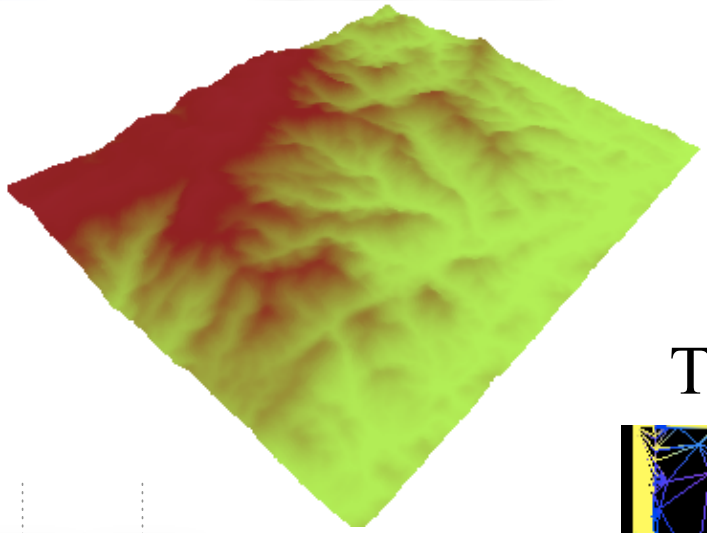
- Problem
  - Terrain T + viewpoint v
  - Compute visibility map or viewshed of v: set of points in T visible from v



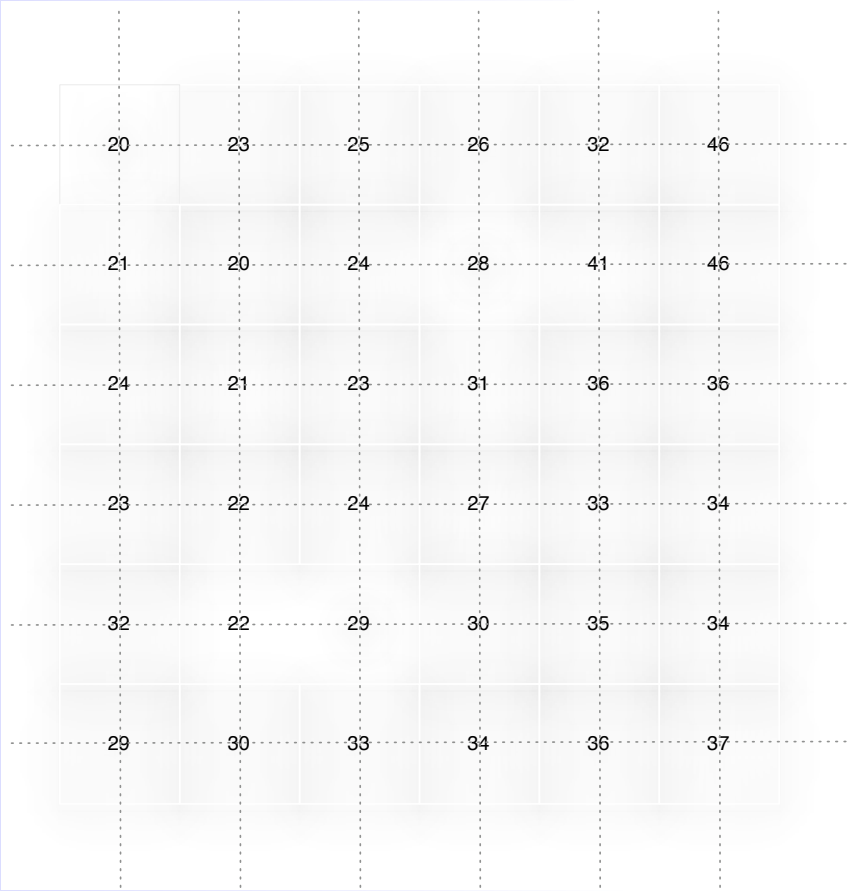
Sierra Nevada, 30m resolution

- Applications:
  - path planning, navigation, placement of fire towers, radar sites, cell phone towers, ...

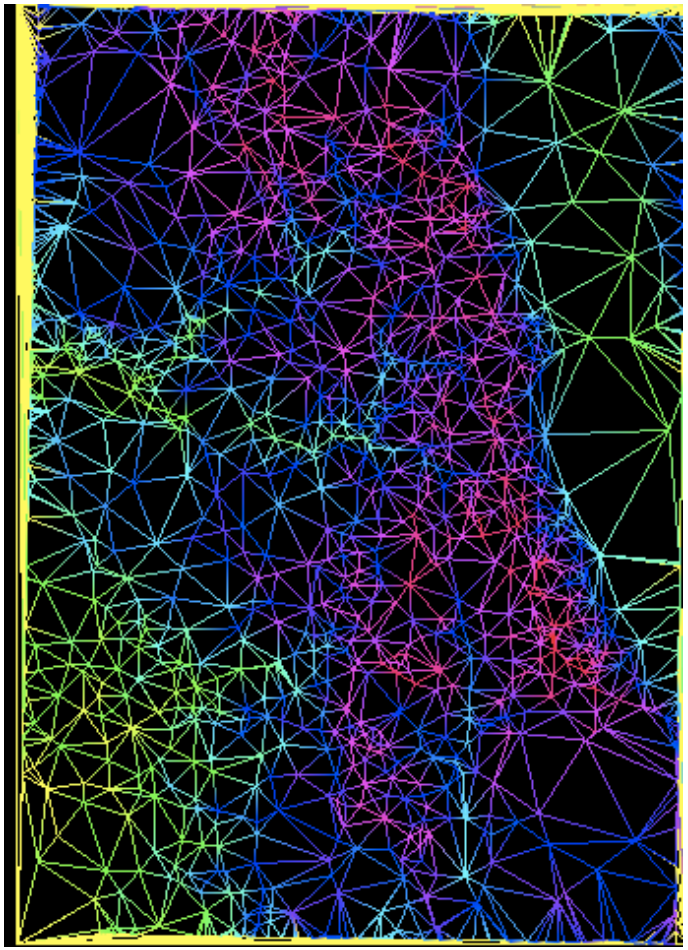
Terrain data



Grid



TIN (triangulated irregular network)



Massive Terrains

- Large amounts of data have become available
  - NASA SRTM project: 30m resolution over the entire globe (~10TB)
  - LIDAR data: sub-meter resolution
  - E.g.: Washington State, 1m grid: 689 GB

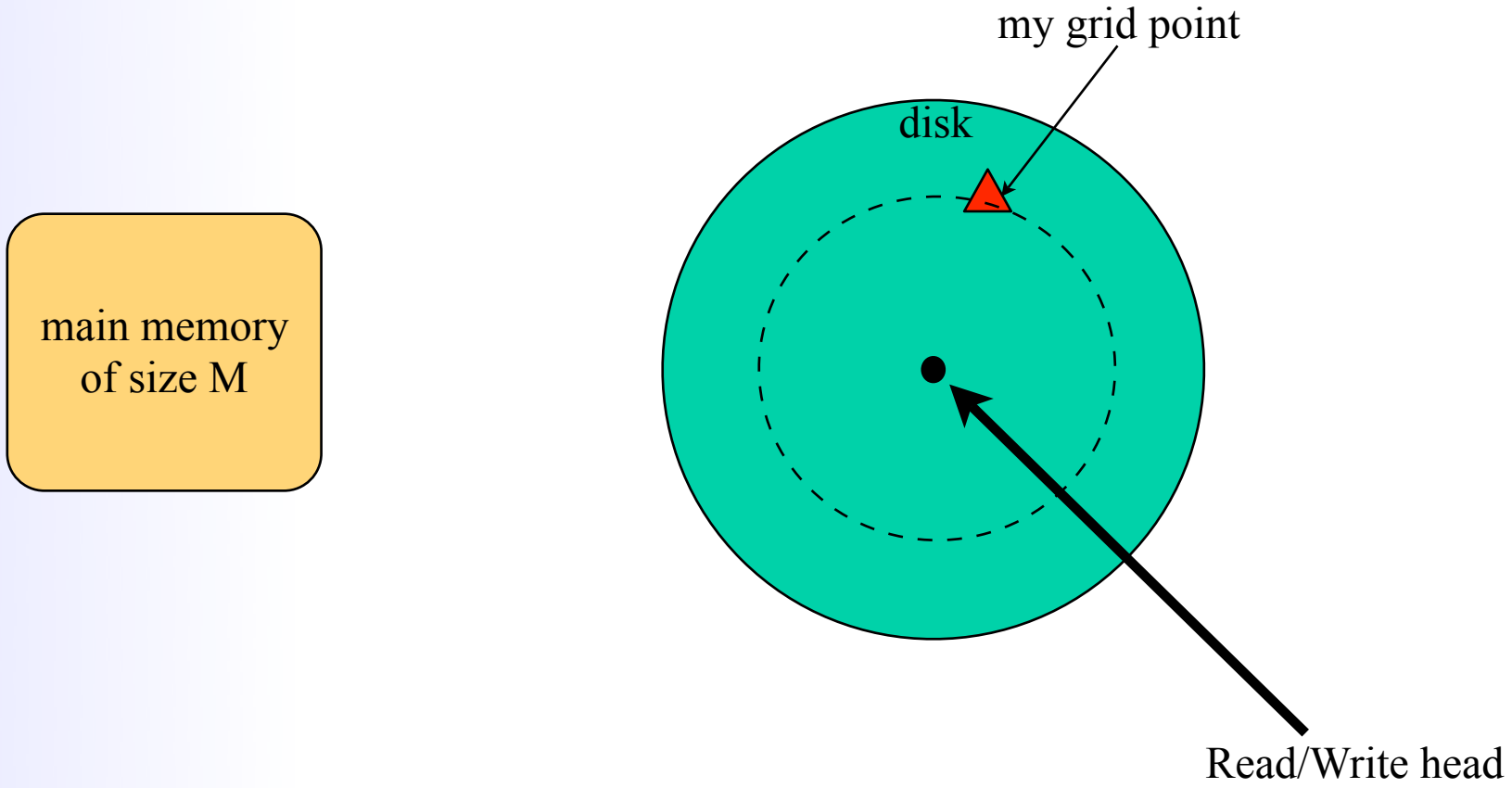
- Traditional (internal memory) algorithms
  - assume all data fits in memory

- In external memory
  - main memory too small to hold all data
  - data on disk
  - disks 1,000,000 slower than memory

→ IO bottleneck

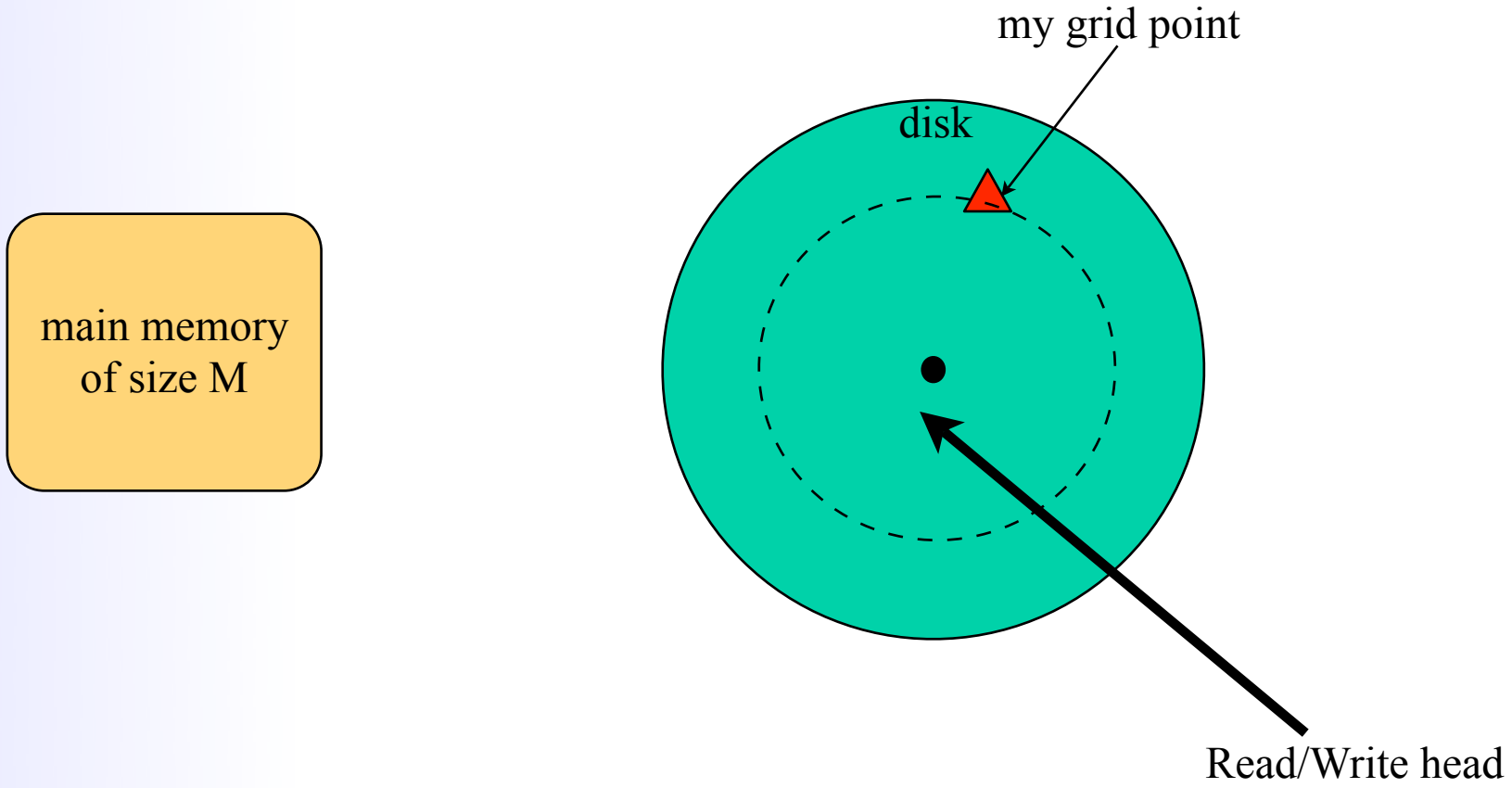
In External Memory

- If main memory is too small to hold all data



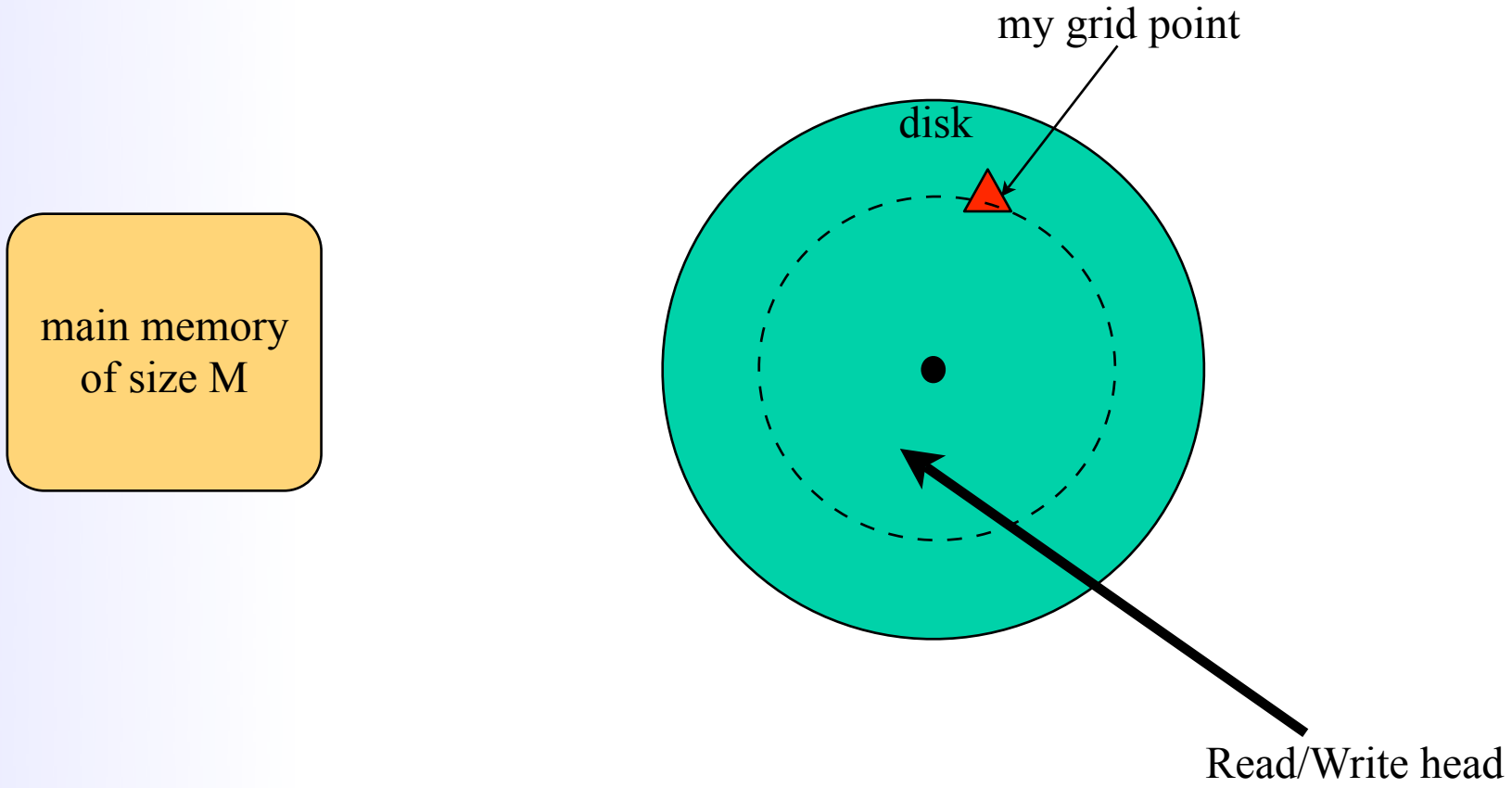
In External Memory

- If main memory is too small to hold all data



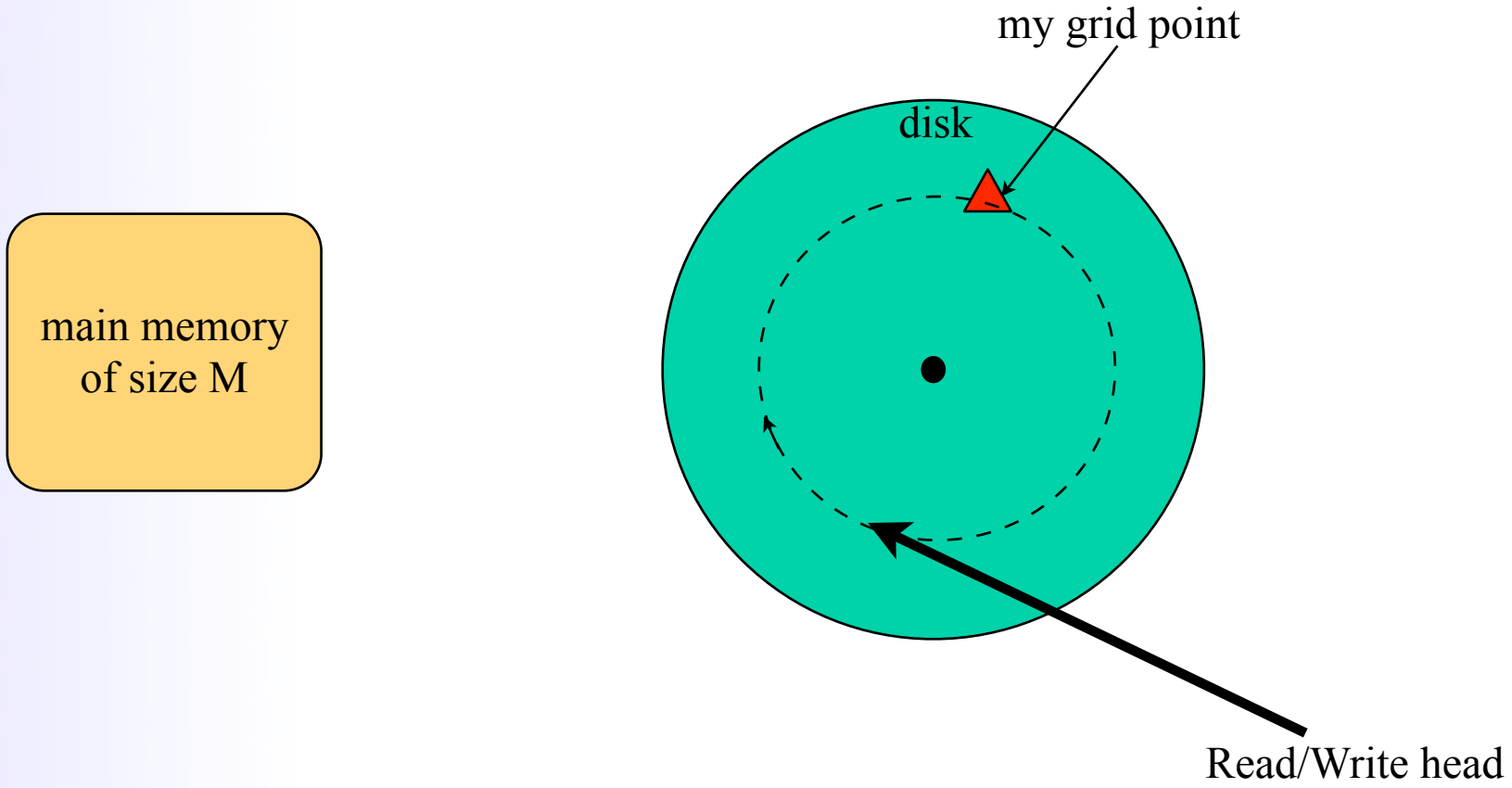
In External Memory

- If main memory is too small to hold all data



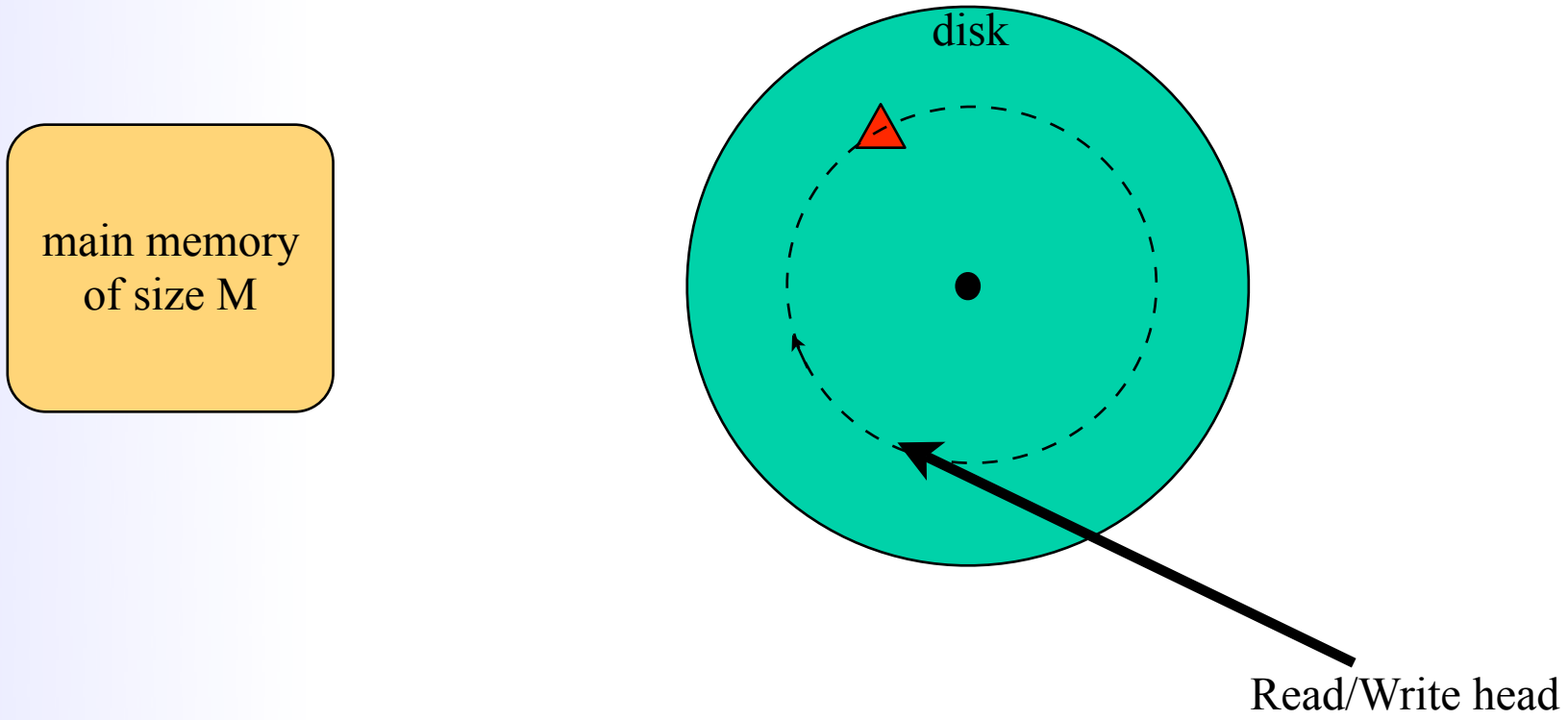
In External Memory

- If main memory is too small to hold all data



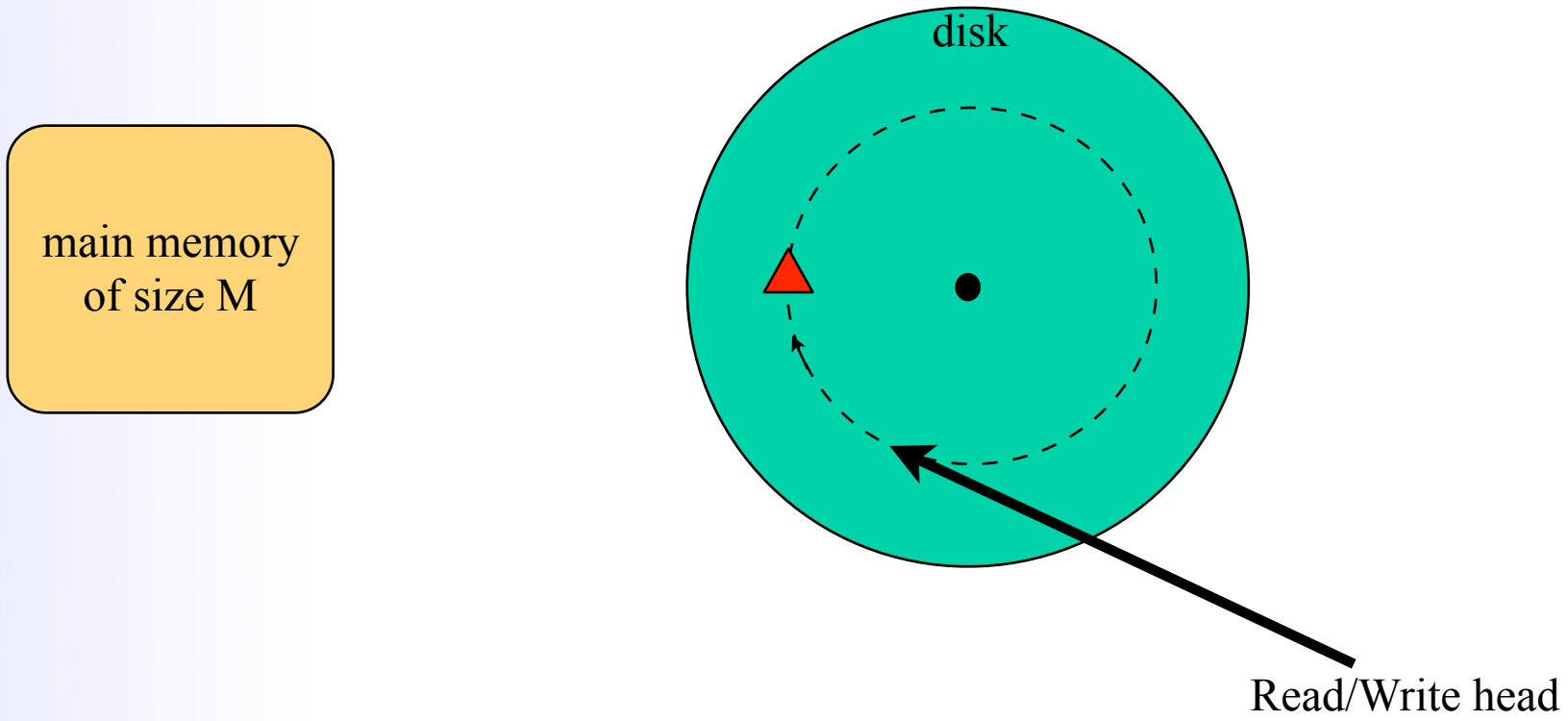
In External Memory

- If main memory is too small to hold all data



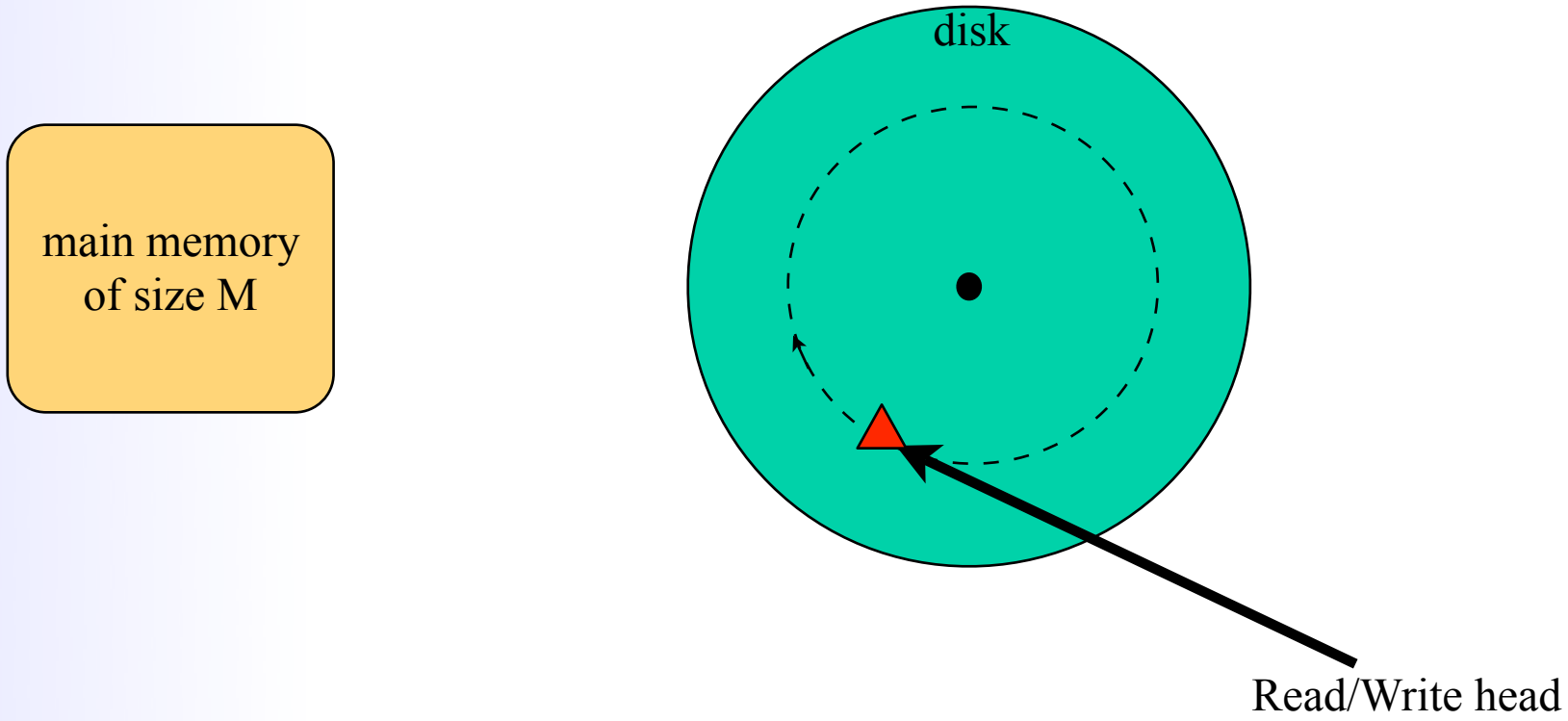
In External Memory

- If main memory is too small to hold all data



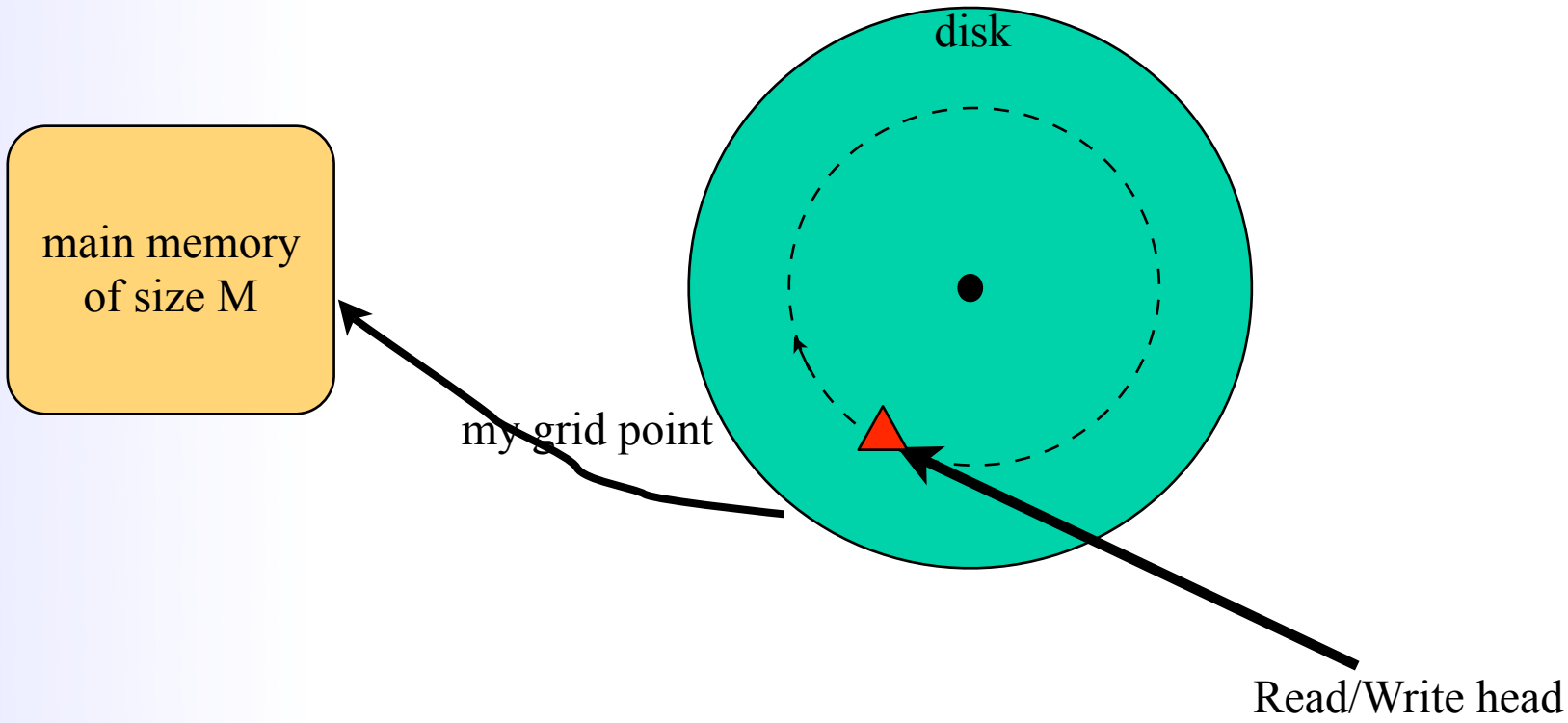
In External Memory

- If main memory is too small to hold all data



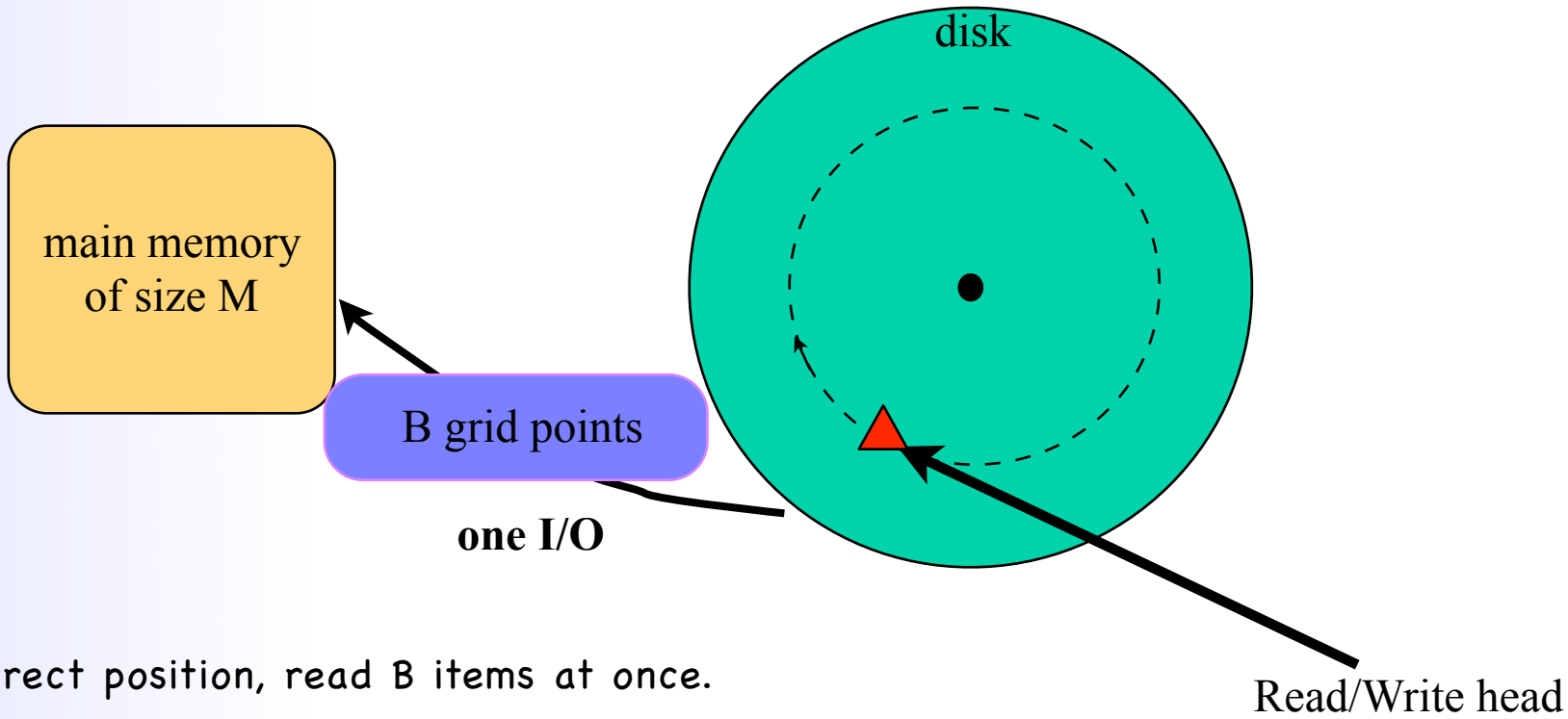
In External Memory

- If main memory is too small to hold all data



In External Memory

- If main memory is too small to hold all data

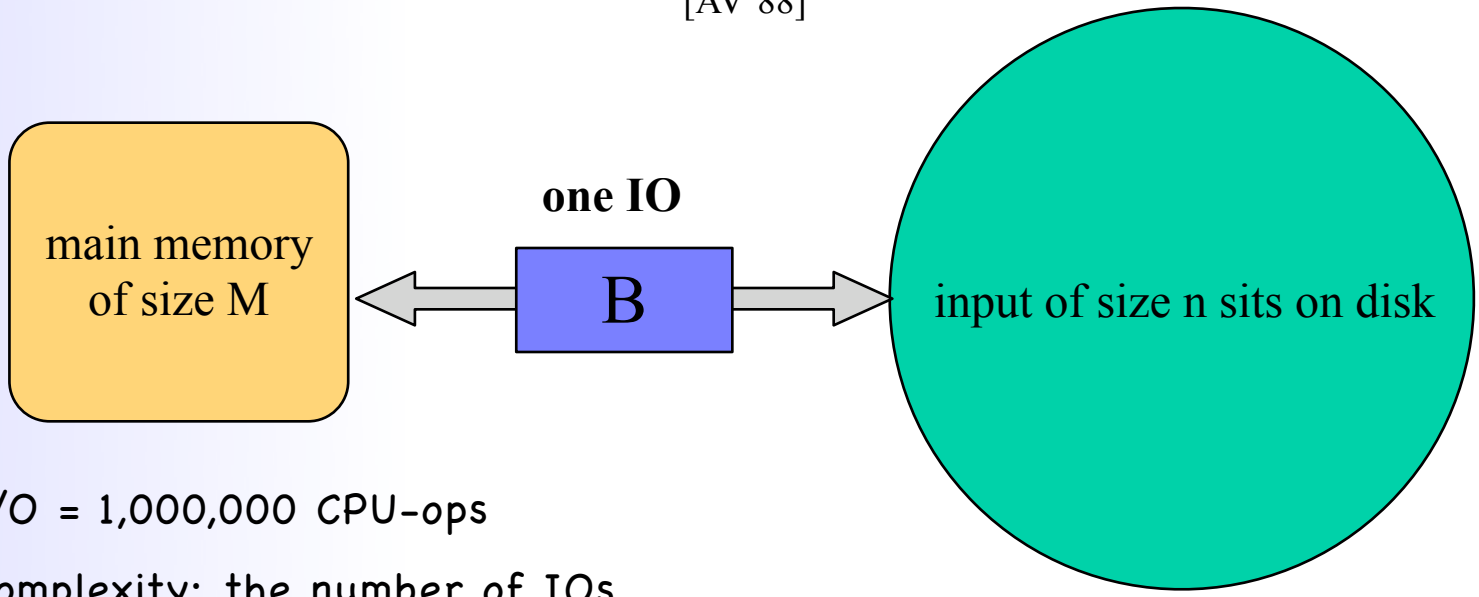


- Once in correct position, read B items at once.
  (hope you can keep them in memory until you need them)

- When working with large data, I/Os dominate.
- Traditional (internal memory) algorithms usually do not scale

I/O-Model

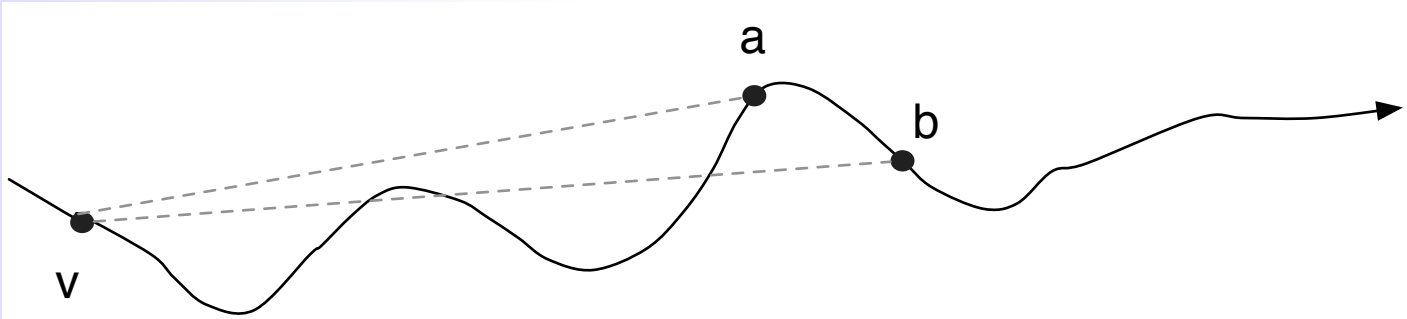
[AV'88]

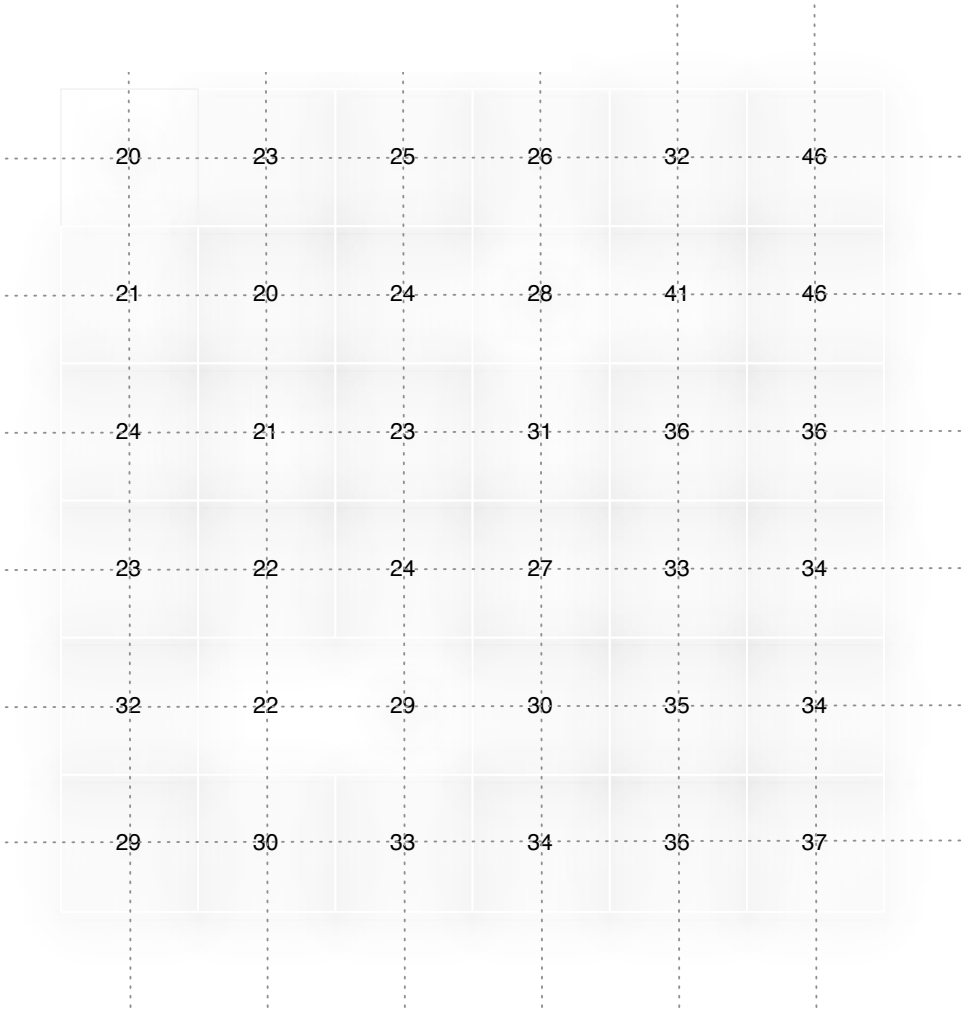


- one I/O = 1,000,000 CPU-ops
- I/O-complexity: the number of IOs
- Goal: minimize (CPU and) I/O-complexity

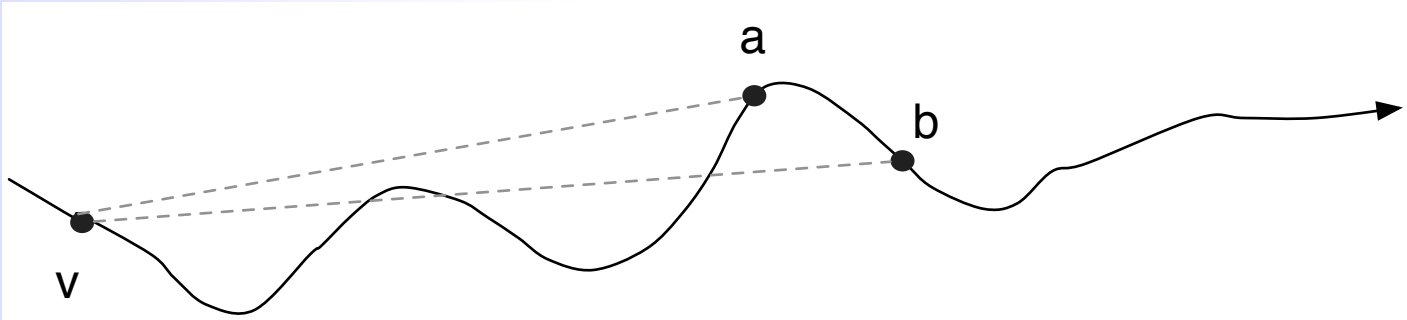
- Basic building blocks and bounds:
  - scanning : $\mathtt{scan}(n) = \frac{n}{B}$ IOs
  - sorting: $\mathtt{sort}(n) = \Theta(\frac{n}{B} \log_{M/B} \frac{n}{B})$ IOs [AV'88]

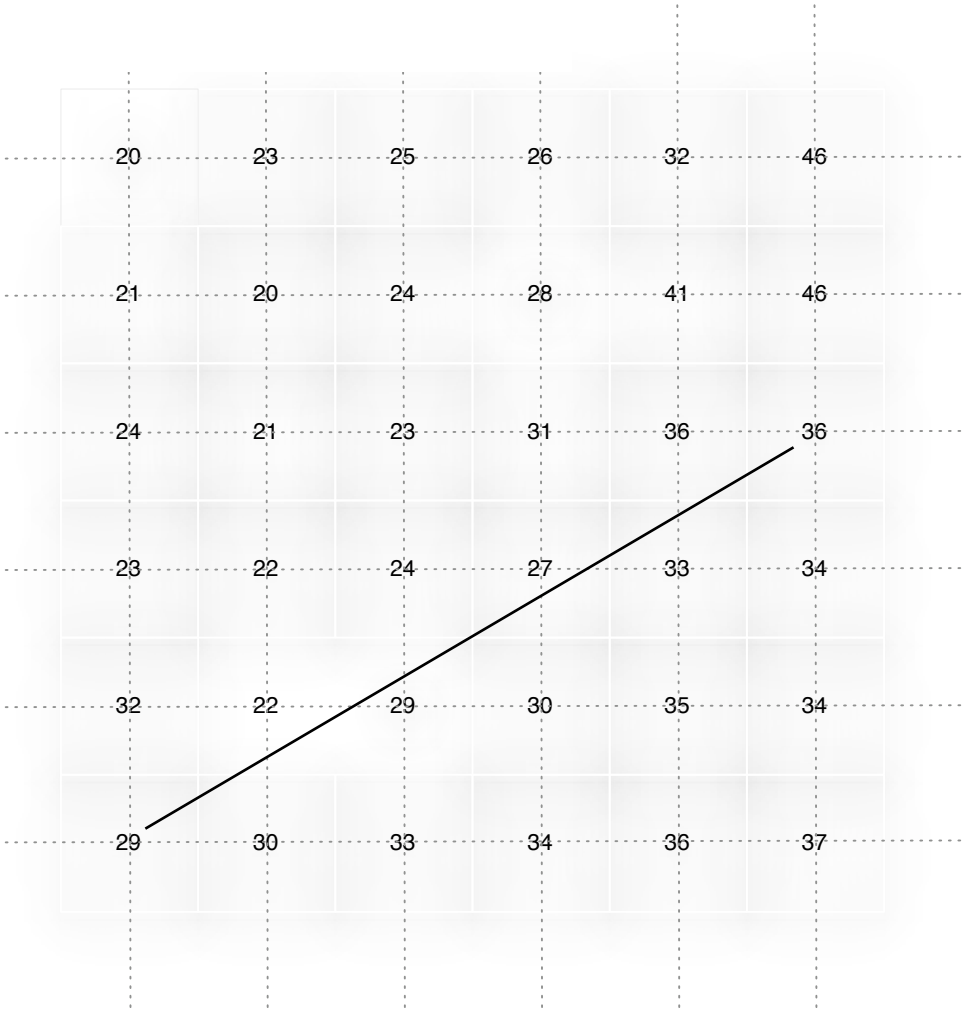
Visibility model



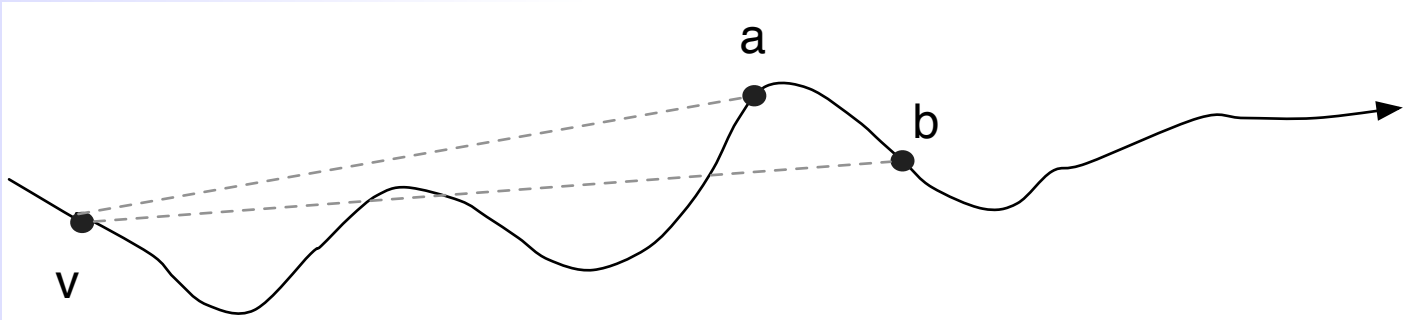


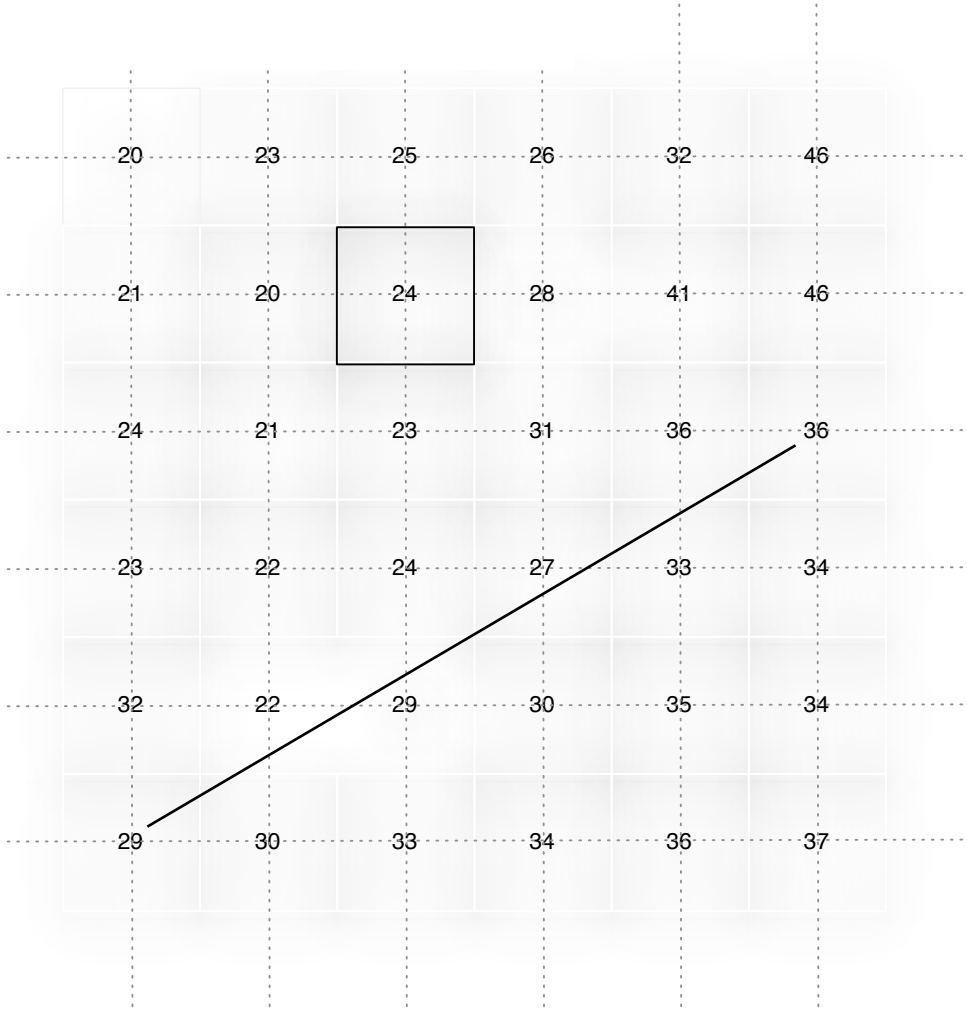
Visibility model





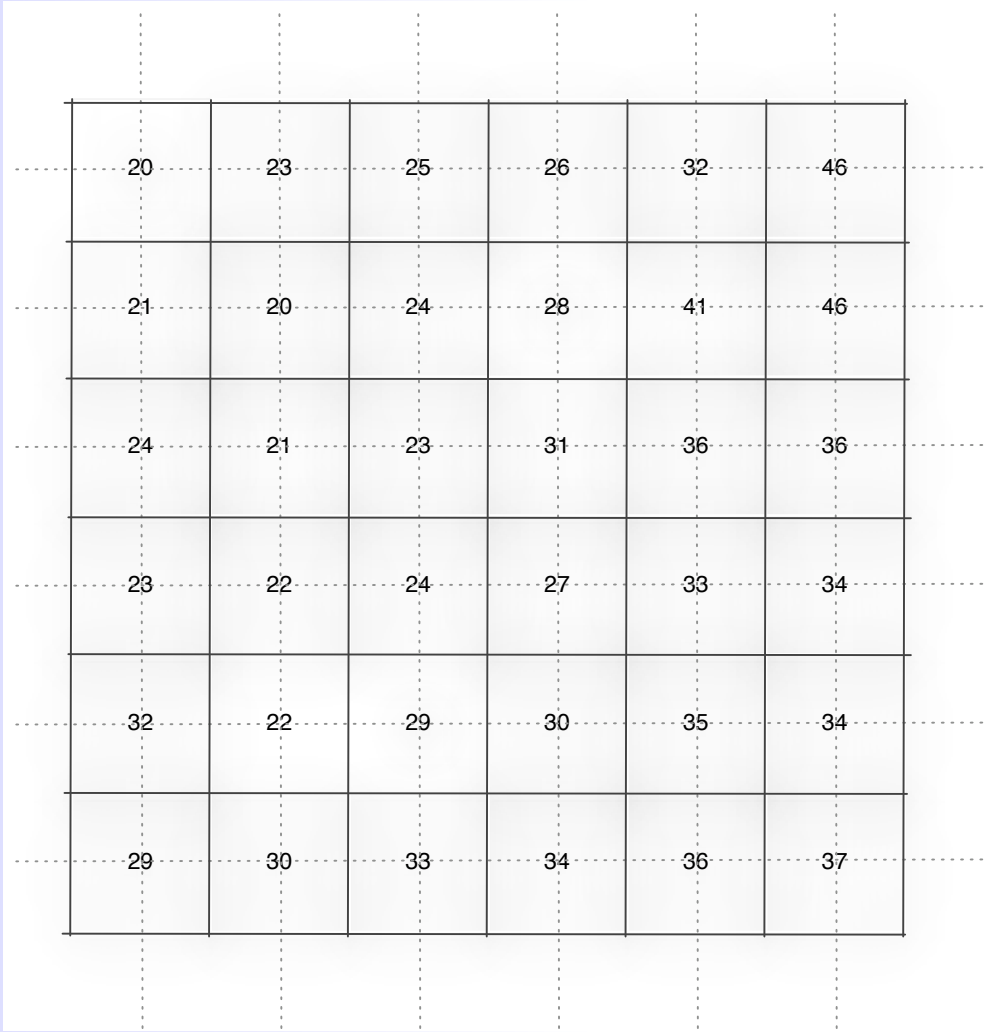
Visibility model

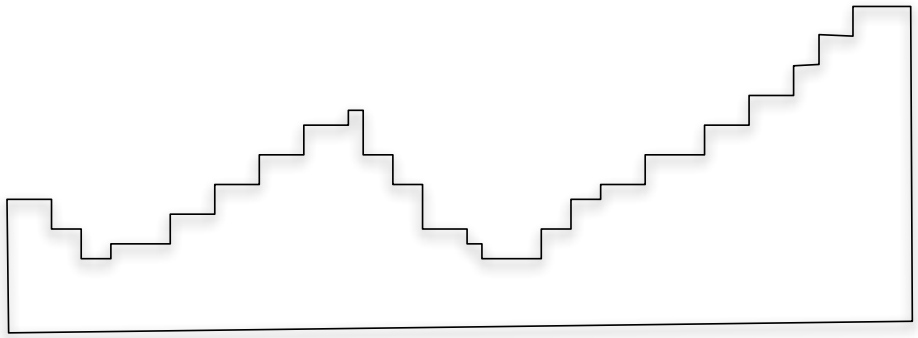




Visibility model

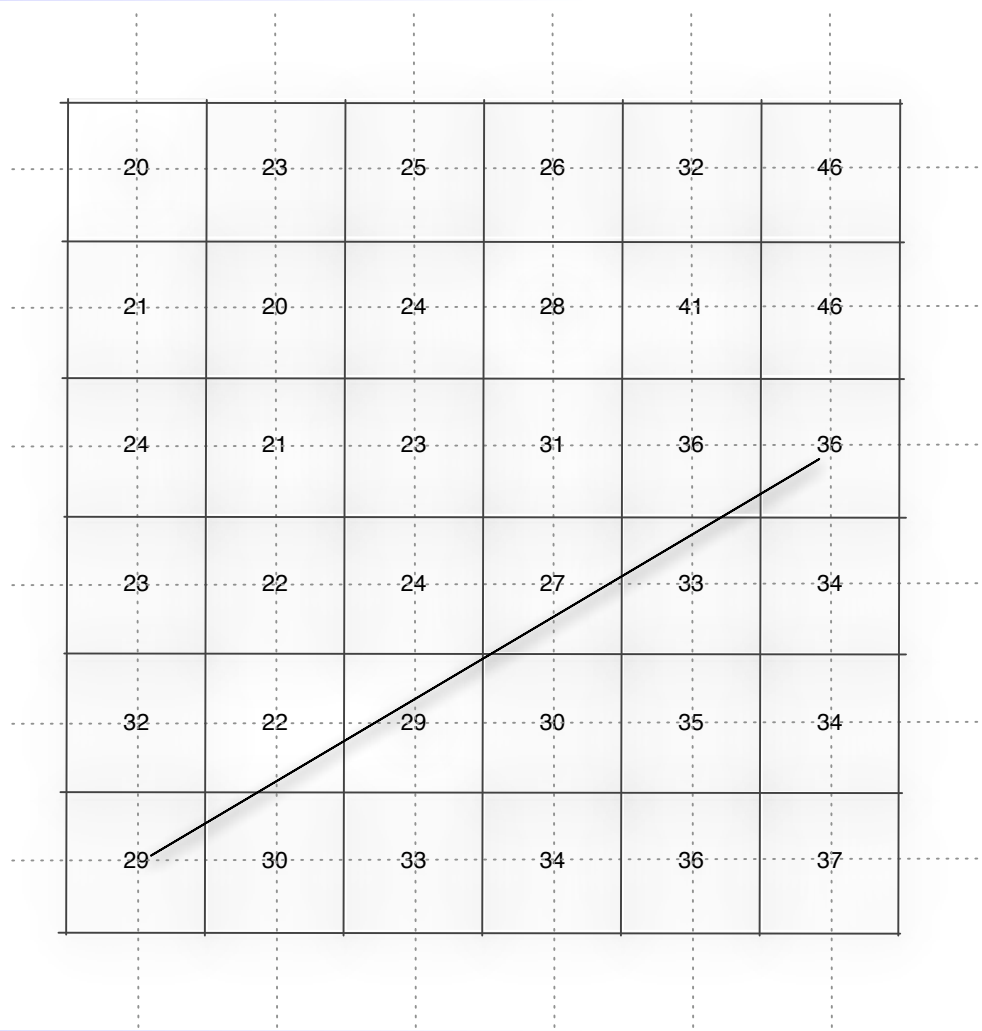
- Nearest-grid-neighbor interpolation

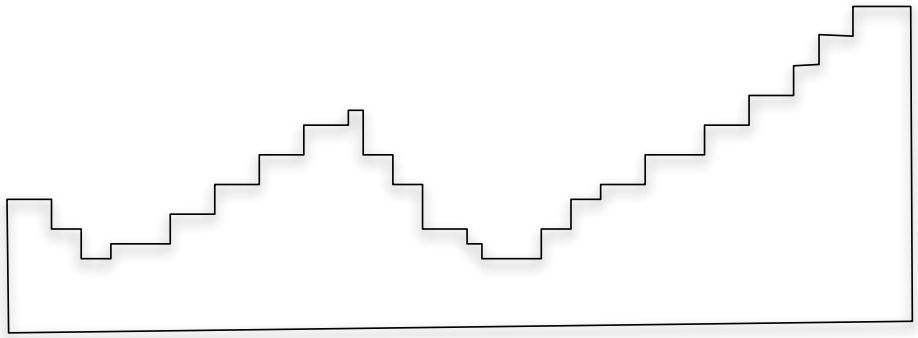




Visibility model

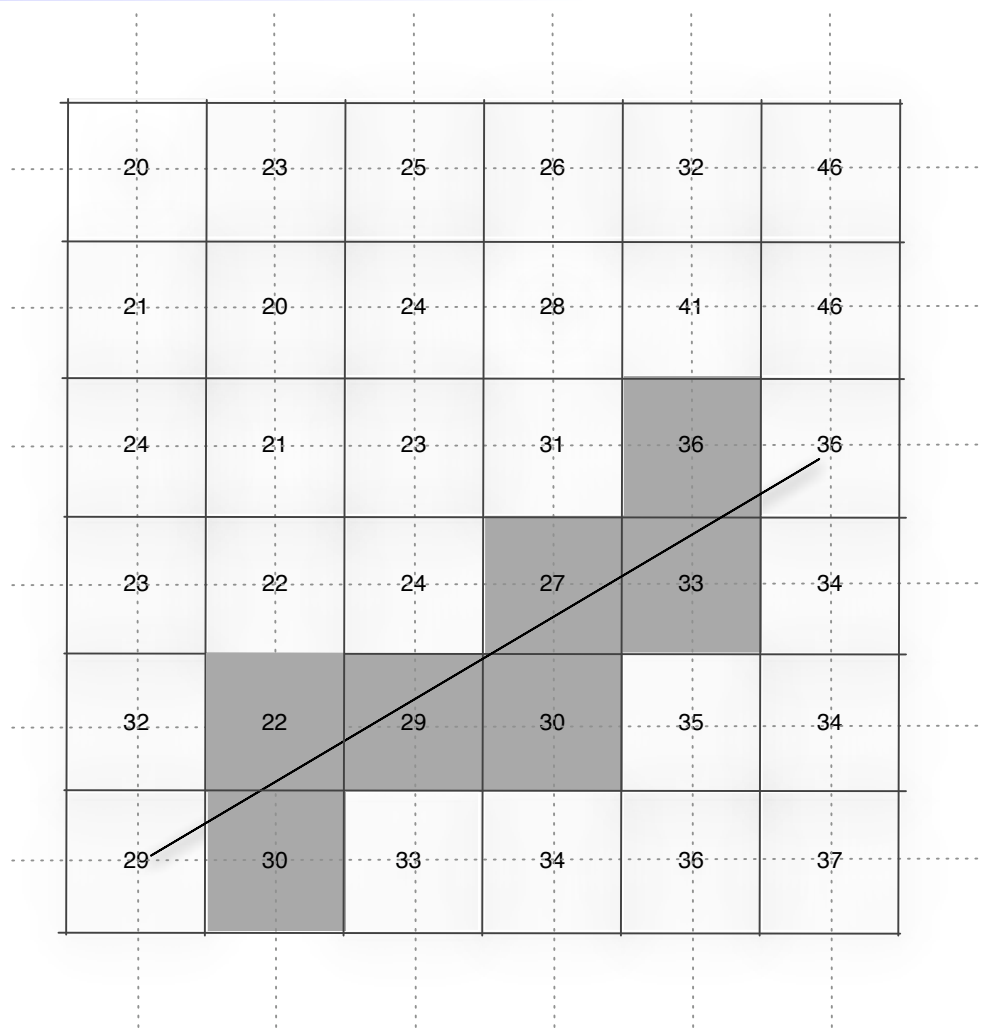
- Nearest-grid-neighbor interpolation

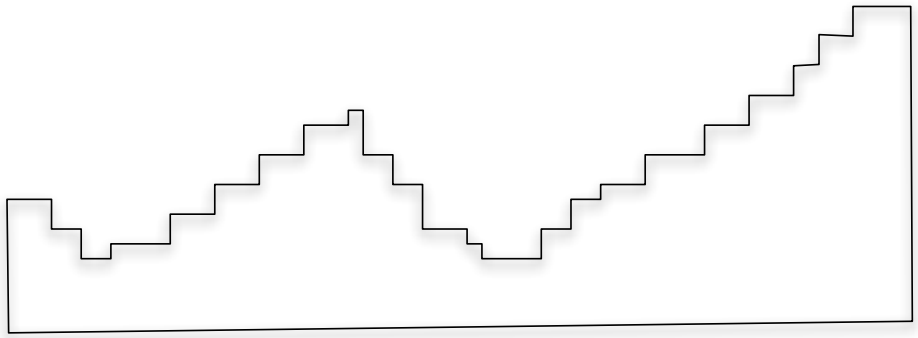




The model

- Nearest-grid-neighbor interpolation





Related work

- Van Kreveld [1996]
  - O(n lg n) radial sweep
  - nearest-grid-neighbor interpolation
- TINs: Cole and Sharir [1989]
- De Floriani and Magillo [1994, 1999, 2003,...]
- Fisher [1993, 1994], Franklin et al [1994, 2002,...],..

- External memory
  - Haverkort, Toma & Zhuang [2006] :
    - O(sort(n)) IOs, good results in practice
    - based on Van Kreveld's model and sweep algorithm
  - Magalhaes, Andrade et al [2007]: O(sort(n)) IOs, good results in practice
    - based on algorithm/model by Franklin
    - LOS to the points on the boundary of the grid used to infer visibility for all inner points

Our contributions

- n = grid size $(\sqrt{n} \text{ by } \sqrt{n})$
- B = disk block size
- M = main memory size

New algorithms for computing viewsheds on grids IO-efficiently

- radial sweep maintaining the terrain profile along the ray
  - O(sort(n)) IOs, O(n lg n) CPU
- centrifugal sweep updating horizons around the viewpoint
  - O(scan(n)) IOs, cache-oblivious, O(n) CPU

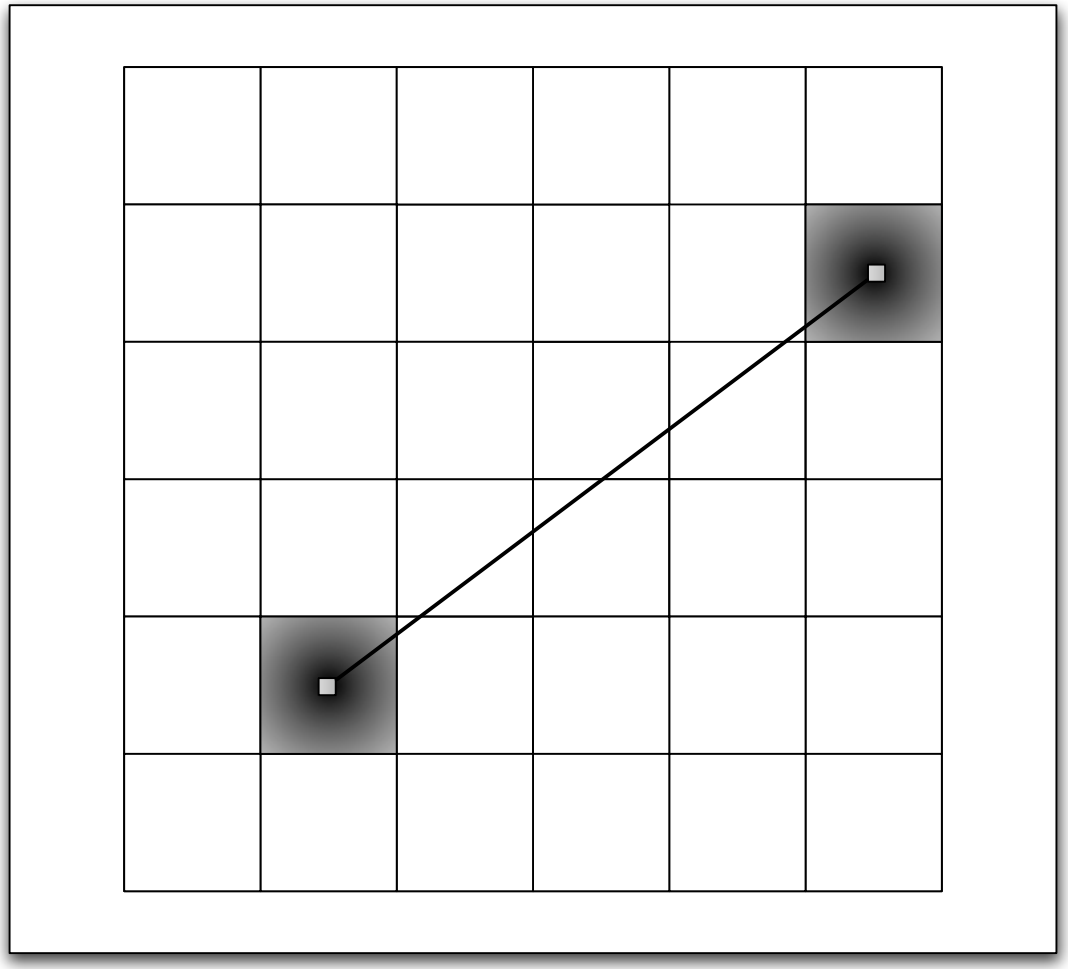
Fast and scalable in practice

- e.g: grid 2.1 billion pts (8.1 GB): 47 minutes -- 110 minutes with .5 GB RAM

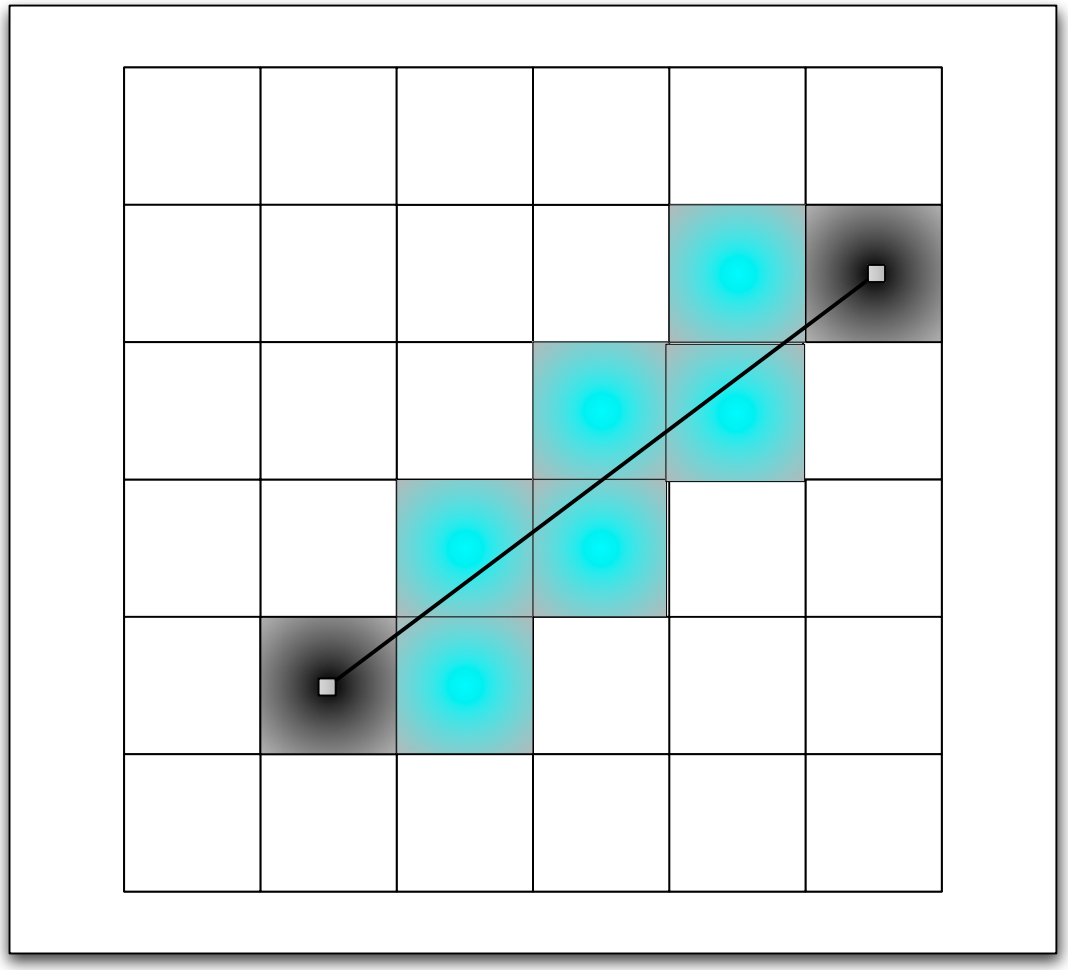
Outline

- Introduction
- Visibility definition
- Model
- Related work
- IO model
- Our results
- Van kreveld's radial sweep algorithm in internal memory
- An IO-efficient radial sweep in layers
- An IO-efficient radial sweep in sectors
- An IO-efficient, cache-oblivious concentric sweep
- Experimental analysis

Ingredients: Van Kreveld's radial sweep algorithm

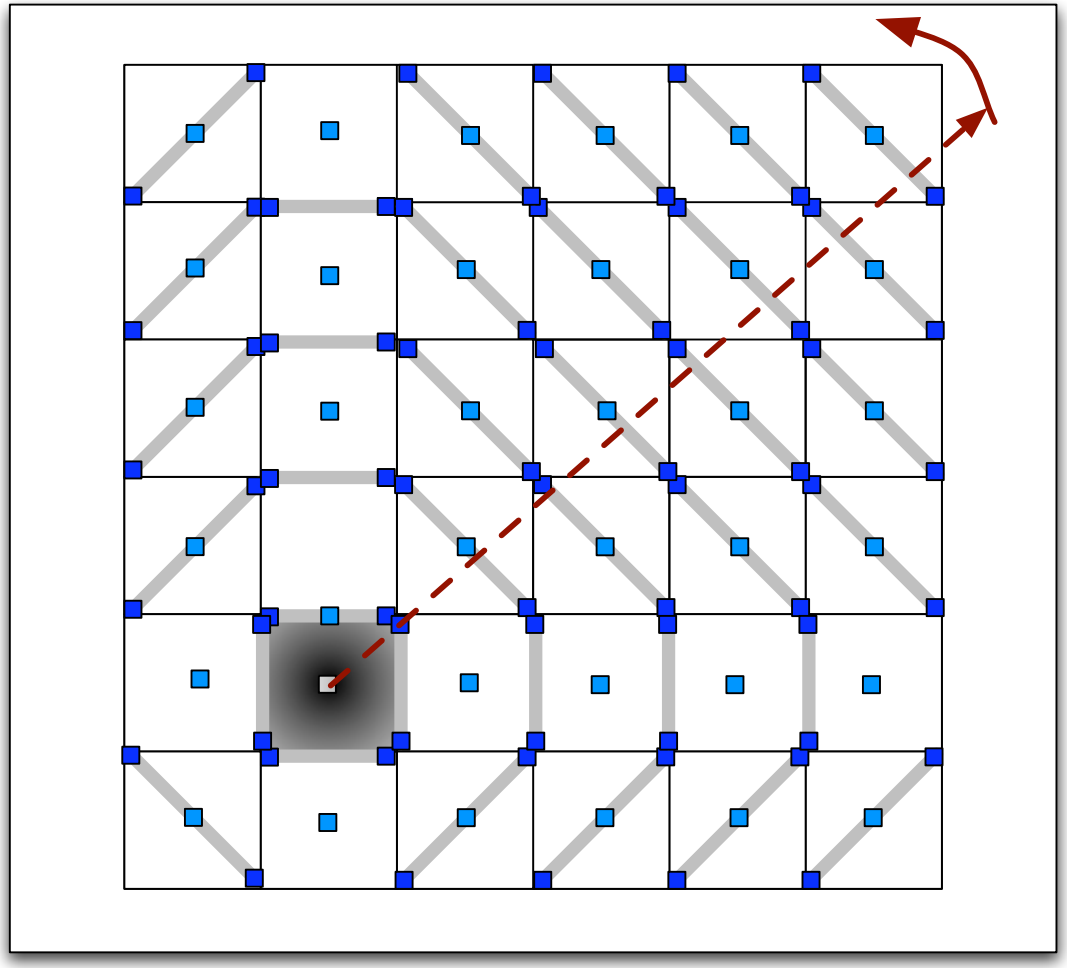


Ingredients: Van Kreveld's radial sweep algorithm



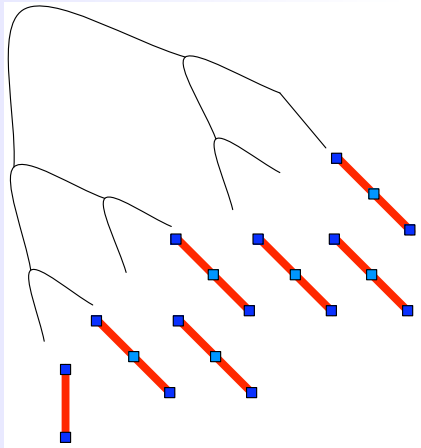
Ingredients: Van Kreveld's radial sweep algorithm

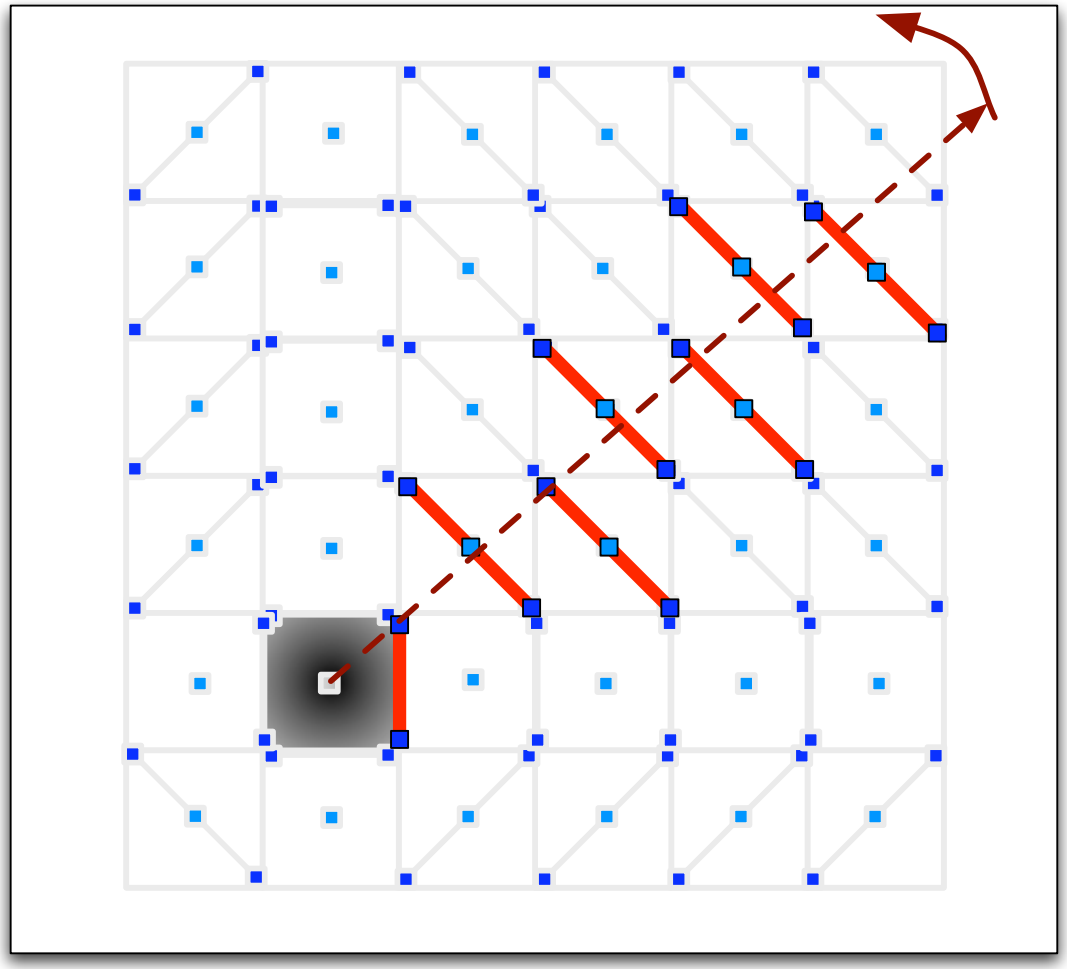
- 3 events per cell:
  - ENTER: becomes active
  - EXIT: stops being active
  - CENTER: query visibility



Ingredients: Van Kreveld's radial sweep algorithm

- ENTER: insert cell in active structure
- EXIT: delete from active structure
- CENTER: find the largest viewangle of all active cells closer to the viewpoint

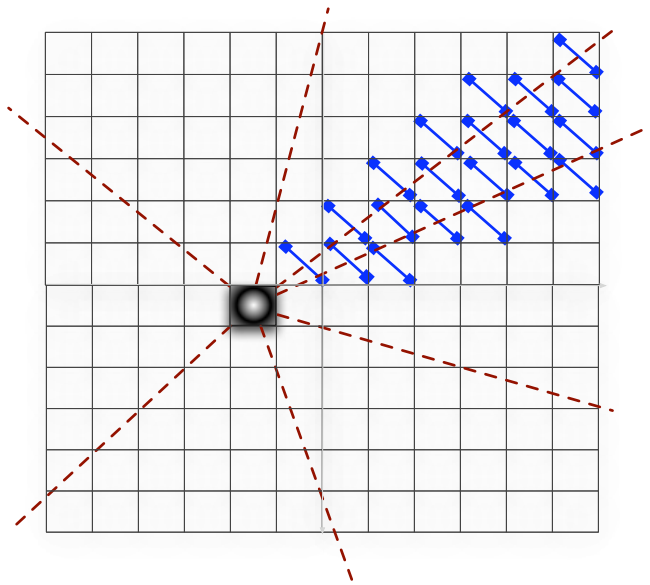




- 3n events, O(lg n) per event ==> Total: O(n lg n)

An IO-efficient radial sweep [HTZ'07]

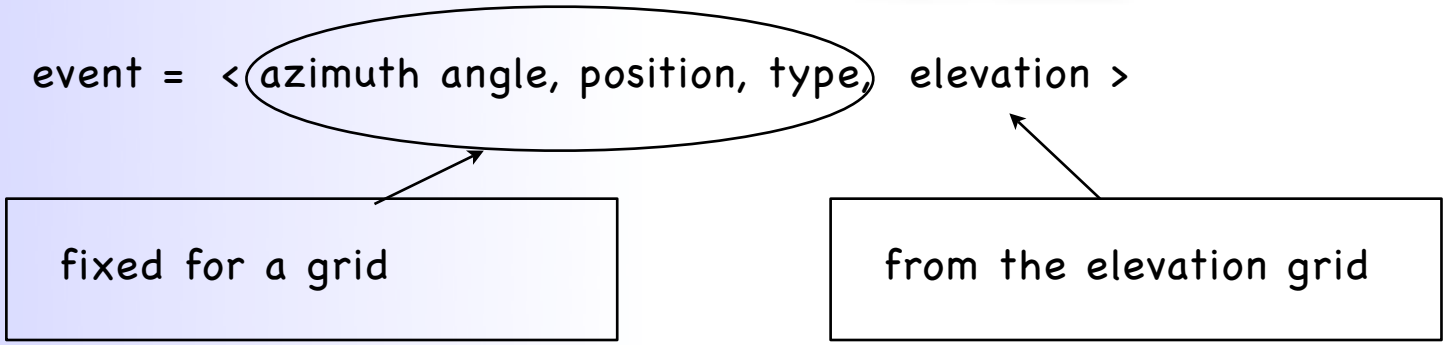
- generate 3n events ==> event stream
  - sort event stream by azimuth angle



Lessons:

- bottleneck: event stream
  - total size: 36n bytes (event = < azimuth angle, i, j, type, elevation >)
  - e.g.: $n=10^9$, event stream=36GB
- in practice $M < n < M^2$

New ideas


event = <azimuth angle, position, type, elevation >
fixed for a grid
from the elevation grid

New ideas

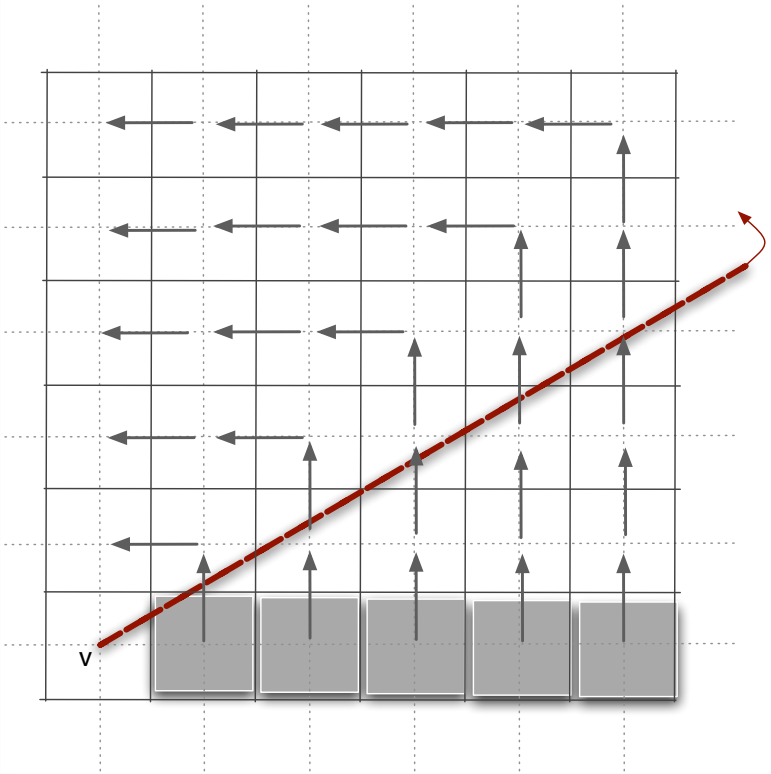
event = < azimuth angle, position, type, elevation >

fixed for a grid

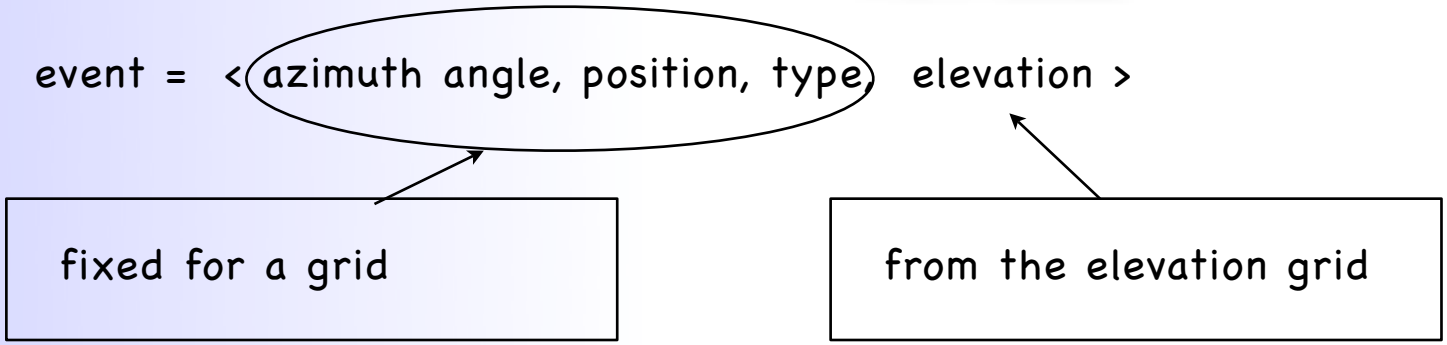
from the elevation grid

(1) the events can be generated, in order, on the fly

- if $n < M^2$, the queue fits in main memory.



New ideas



(2) CENTER and EXIT events do not need to know the elevation

(3) need a list of the elevation of the grid cells in the order of their ENTER event

An IO-efficient radial sweep in layers

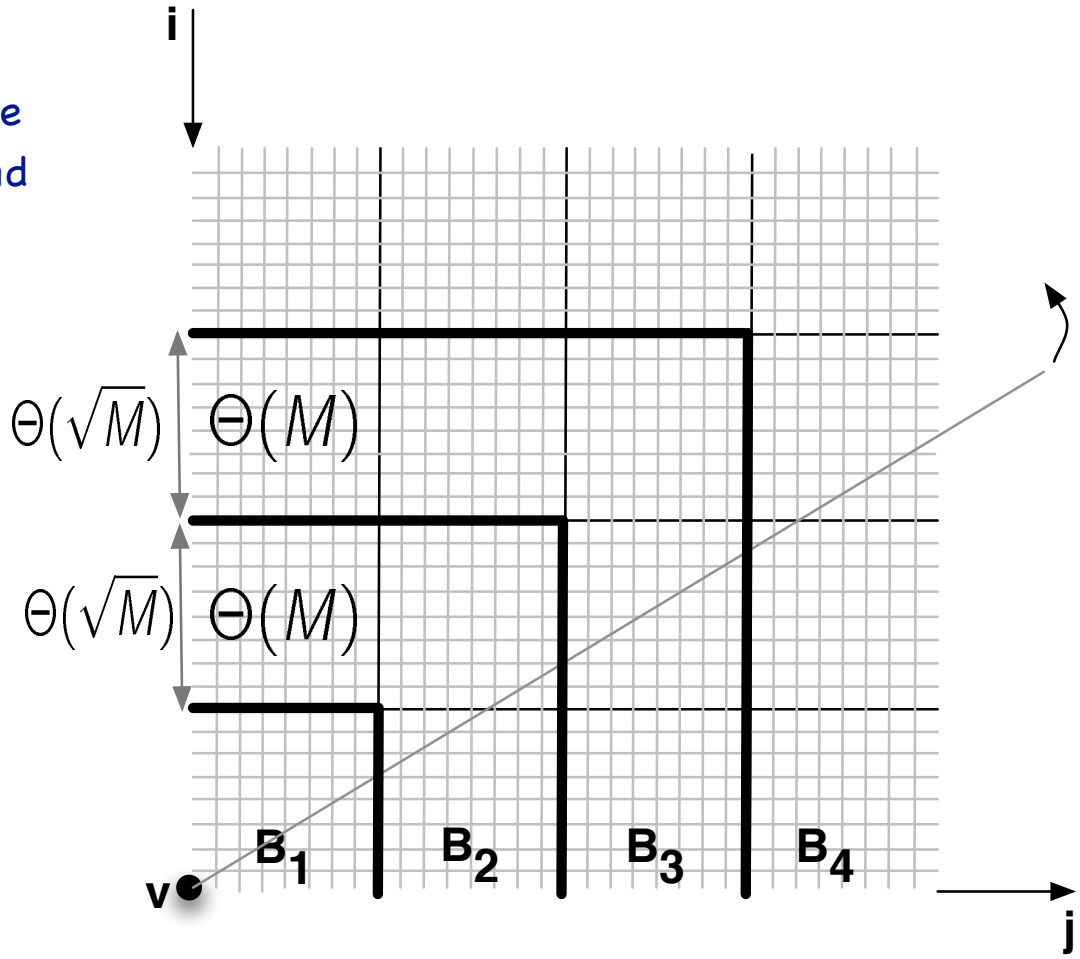
An IO-efficient radial sweep in layers

Algorithm:

1. BUILD-BANDS: for each band B_k, generate a list E_k containing the elevations in the band in order of their ENTER event.

2. SWEEP: compute and output list V_{ij}

3. Sort V_{ij} to create visibility grid



An IO-efficient radial sweep in layers

Algorithm:

1. BUILD-BANDS: for each B_k, generate a list E_k containing the elevation in the band in order of their ENTER event.

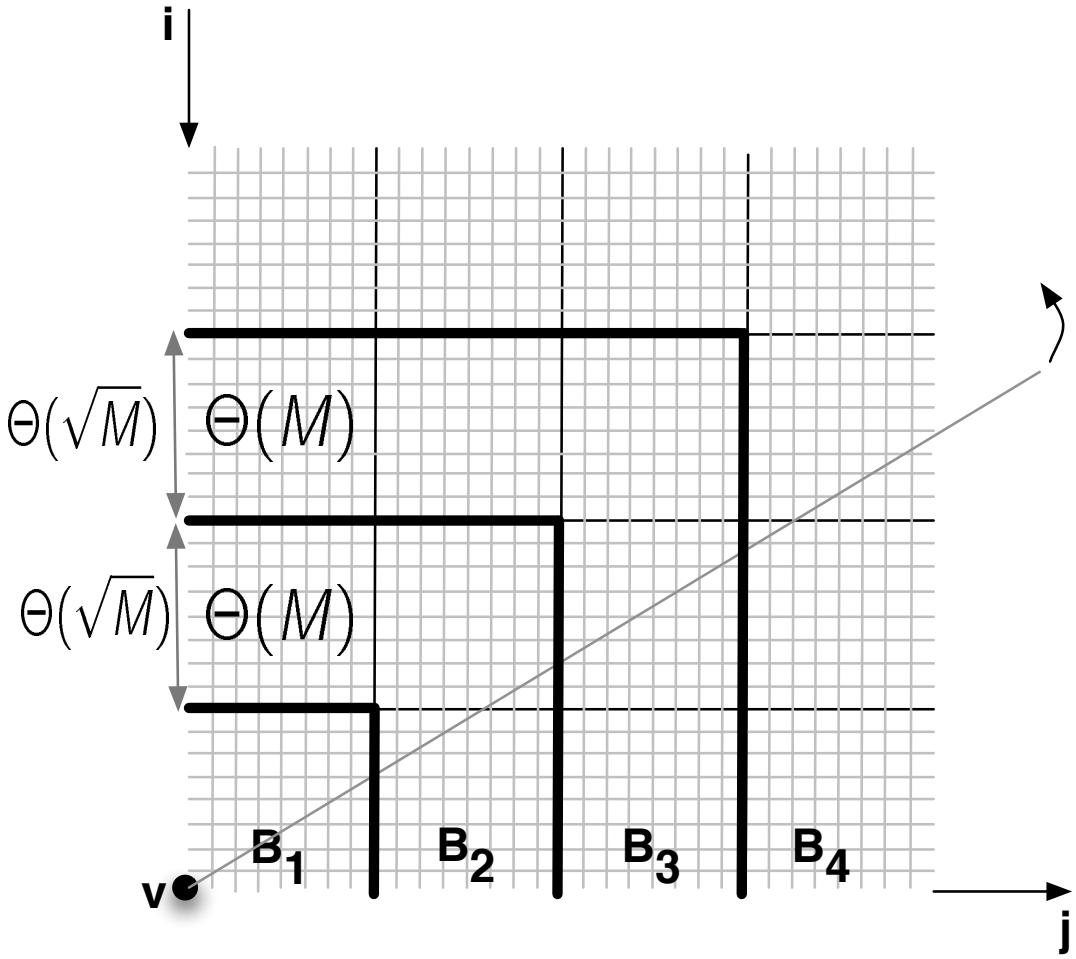
   IO: one scan pass

2. SWEEP: compute and output list V_{ij}

   IO: one scan pass

3. Sort V_{ij} to create visibility grid

   IO: one scan pass



Analysis:

- Total: 3 scan passes = O(scan(n)) IOs
  - assume $M > B^2$, $n < M^2$
- without any assumptions: O(sort(n)) IOs

An IO-efficient radial sweep in sectors

An IO-efficient radial sweep in sectors

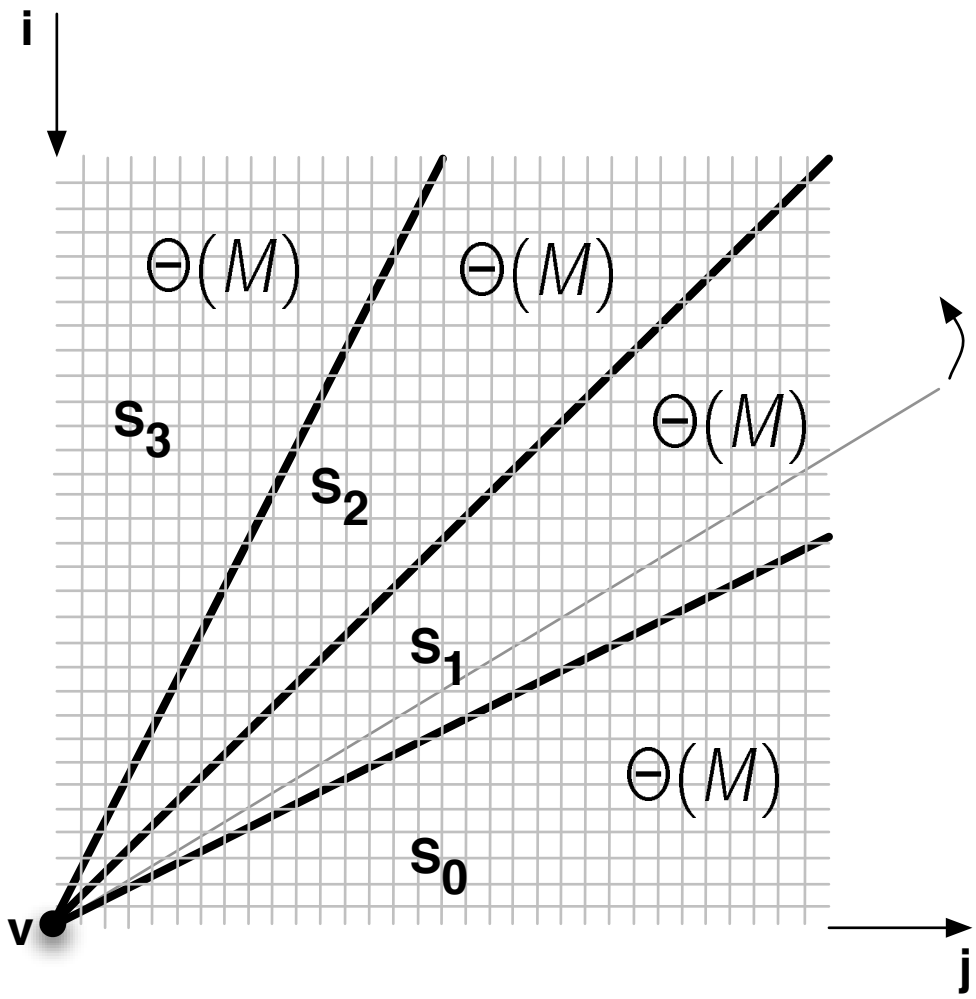
Idea: divide grid in sectors instead of bands

Algorithm:

1. BUILD-SECTORS: for each S_k, generate a list E_k containing the elevation in the sector

2. SWEEP: Compute and output V_{ij}

3. Sort V_{ij} to create visibility grid



An IO-efficient radial sweep in sectors

Idea: divide grid in sectors instead of bands

Algorithm:

1. BUILD-SECTORS:

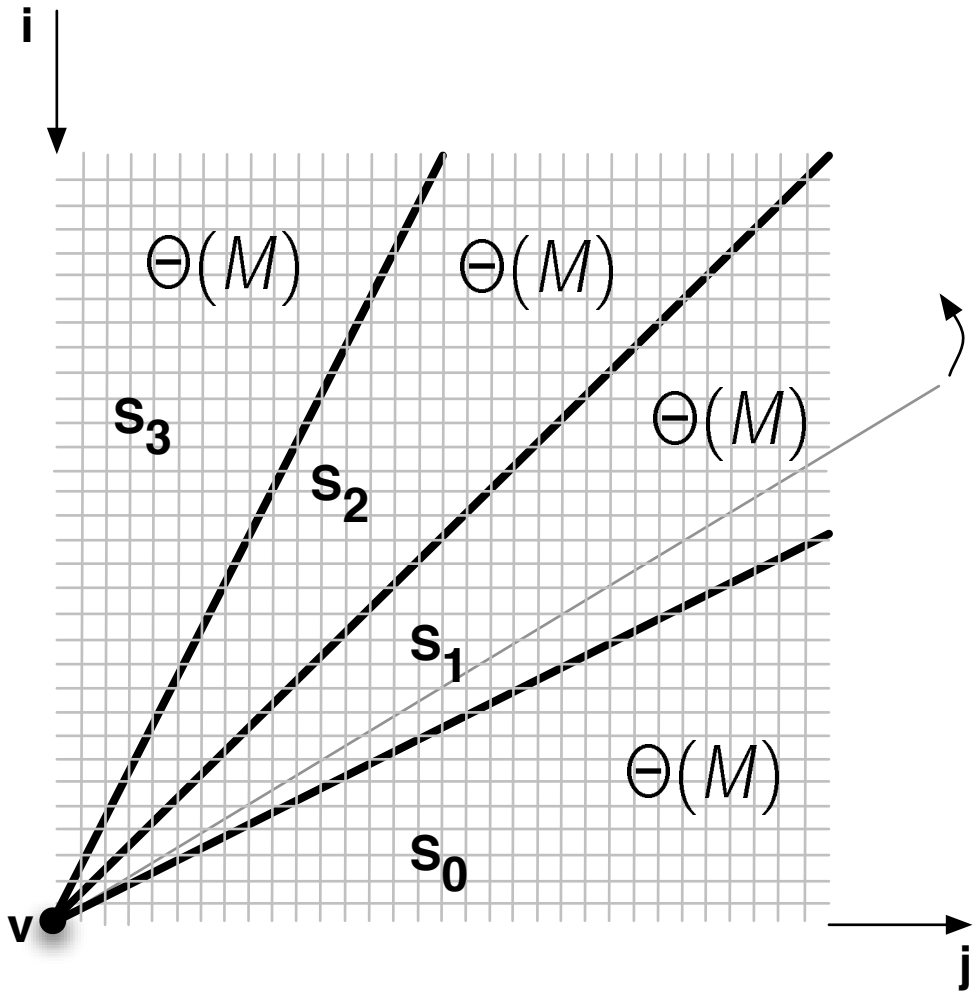
   IO: scan(n) IOs

2. SWEEP: Compute and output V_{ij}

   IO: scan(n) IOs

3. Sort V_{ij} to create visibility grid

   IO: scan(n)



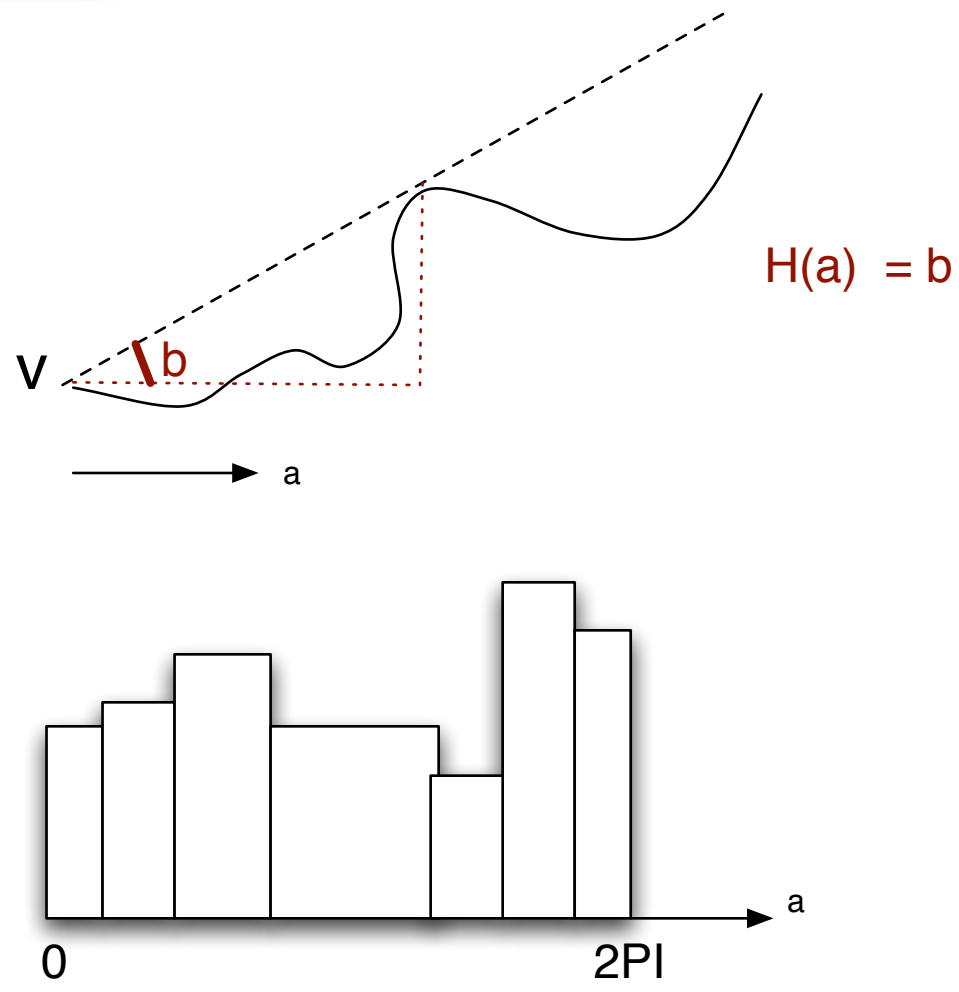
Analysis:

- Total: 3 scan passes = O(scan(n)) IOs, assuming $n < M^2/B$
- Without any assumptions: O(sort(n)) IOs

An IO-efficient, cache-oblivious concentric sweep

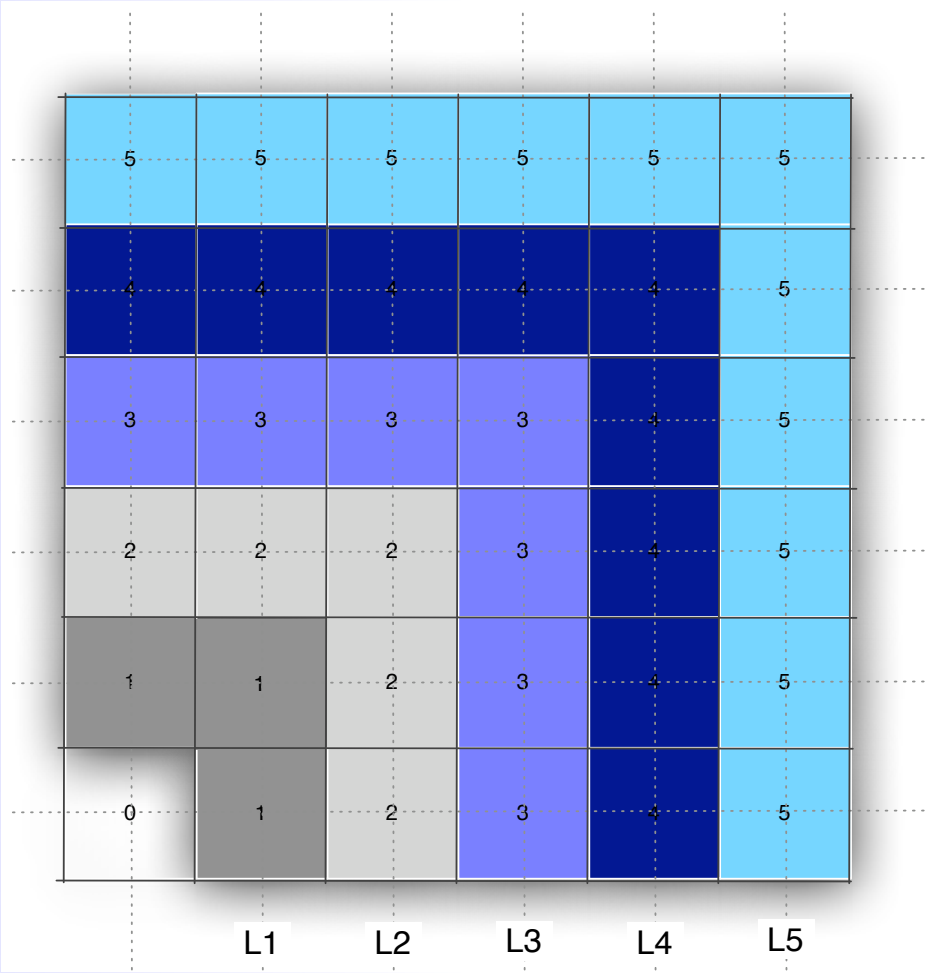
An IO-efficient, cache-oblivious concentric sweep

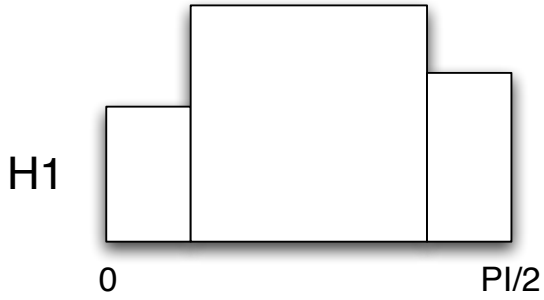
- Idea: expand horizons around the viewpoint

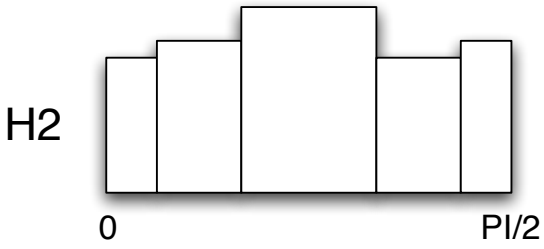


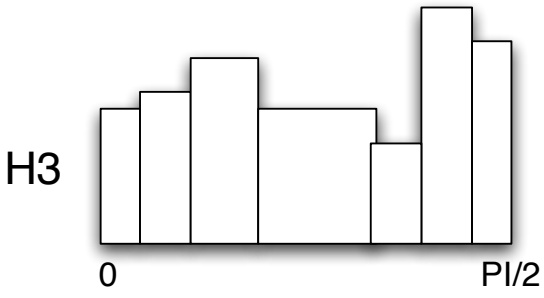
An IO-efficient, cache-oblivious concentric sweep

- Idea: compute horizons in layers around the viewpoint, similar to TIN algorithms
  - compute H1, H2, H3, ...
  - merge



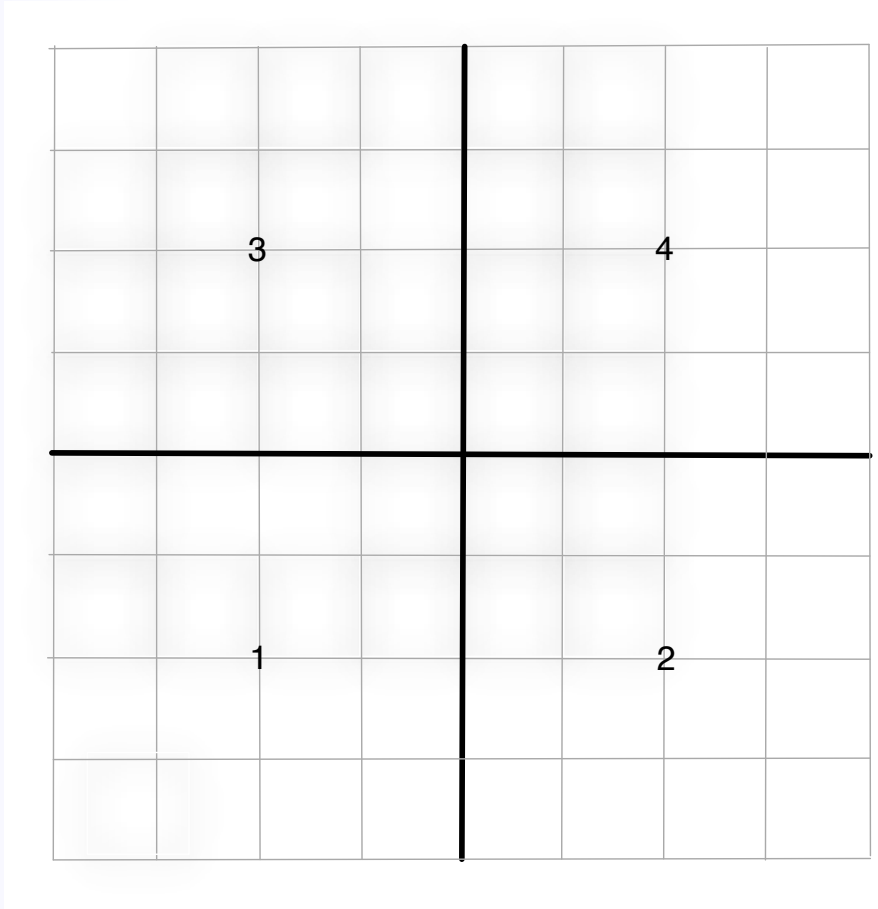






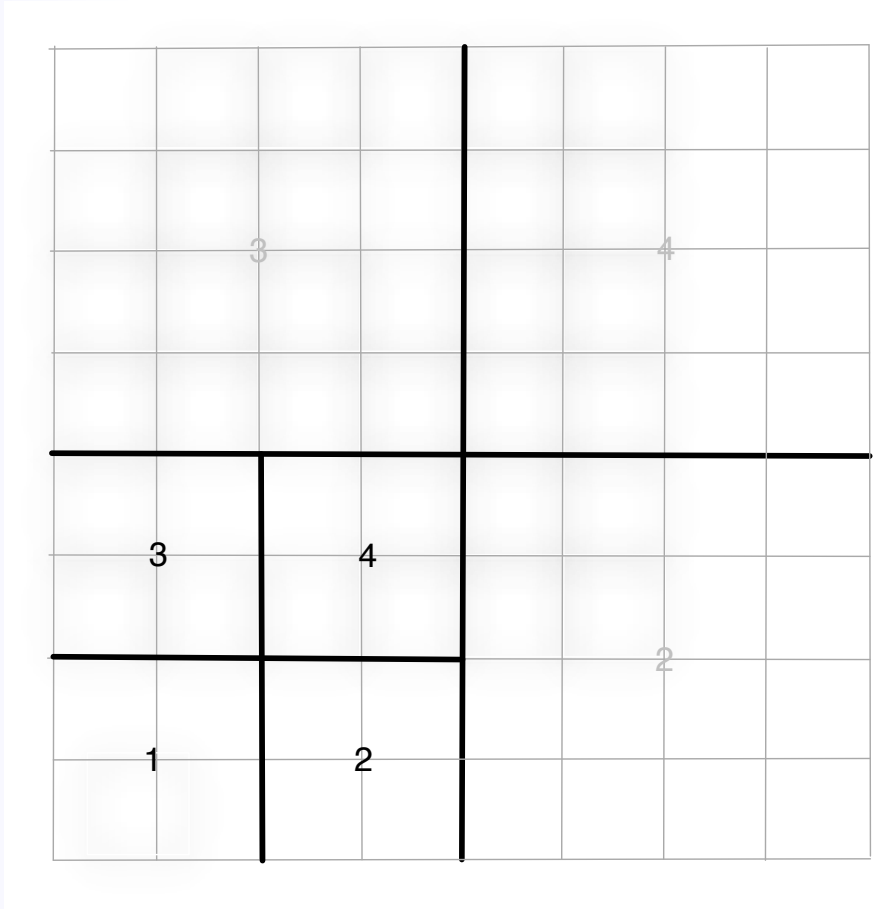
An IO-efficient, cache-oblivious concentric sweep

- New ideas:
  - recursive, block-based approach ==> IO-efficient, cache-oblivious
  - store the horizon of a layer as an array with resolution similar to the grid



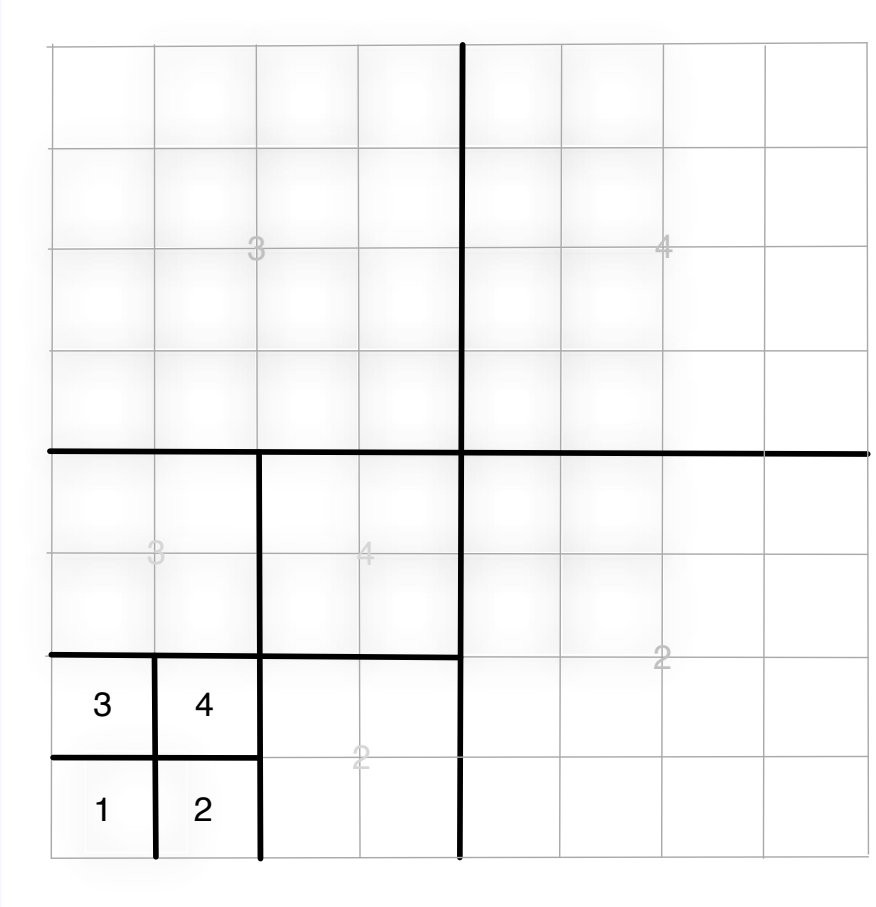
An IO-efficient, cache-oblivious concentric sweep

- New ideas:
  - recursive, block-based approach ==> IO-efficient, cache-oblivious
  - store the horizon of a layer as an array with resolution similar to the grid



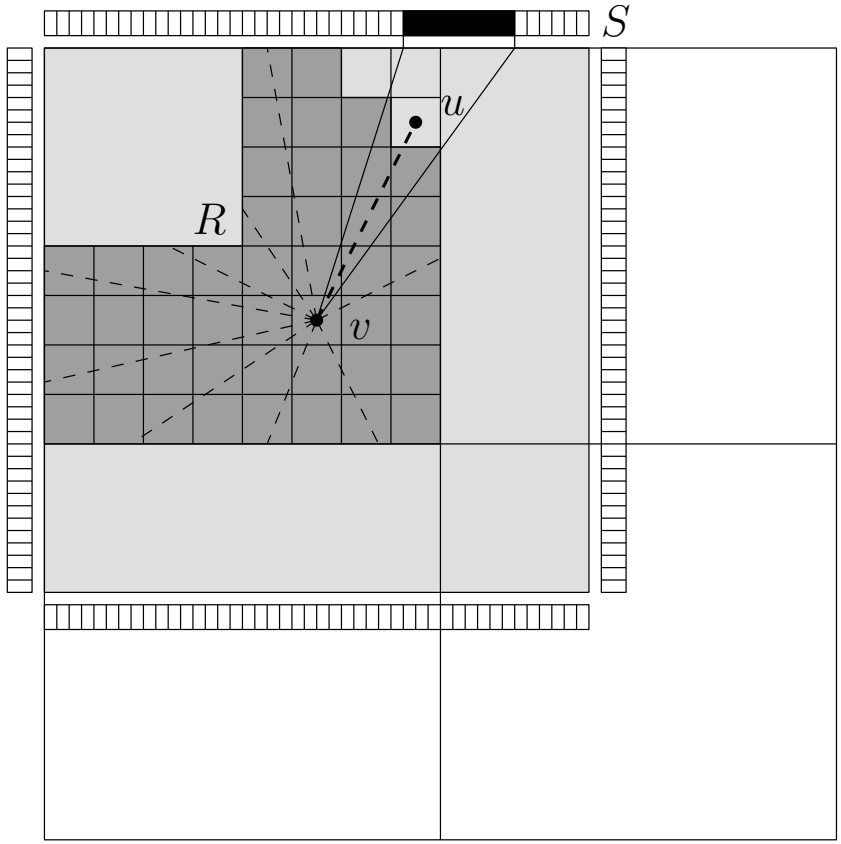
An IO-efficient, cache-oblivious concentric sweep

- New ideas:
  - recursive, block-based approach ==> IO-efficient, cache-oblivious
  - store the horizon of a layer as an array with resolution similar to the grid



An IO-efficient, cache-oblivious concentric sweep

- New ideas:
  - recursive approach ==> IO-efficient, cache-oblivious
  - store the horizon as an array with resolution similar to the grid $\Theta(\sqrt{n})$



- Analysis:
  - accesses to elevation grid: loading a tile of size O(M) takes $O(\sqrt{M} + M/B) = O(M/B)$)IOs (assume $M > B^2$)
  - access to horizon array: O(n) CPU, O(n/B) IOs
    - $\Theta(\sqrt{n})$ layers
    - H has size $\Theta(\sqrt{n})$
- TOTAL: O(n/B) = scan(n) IOs, O(n) CPU

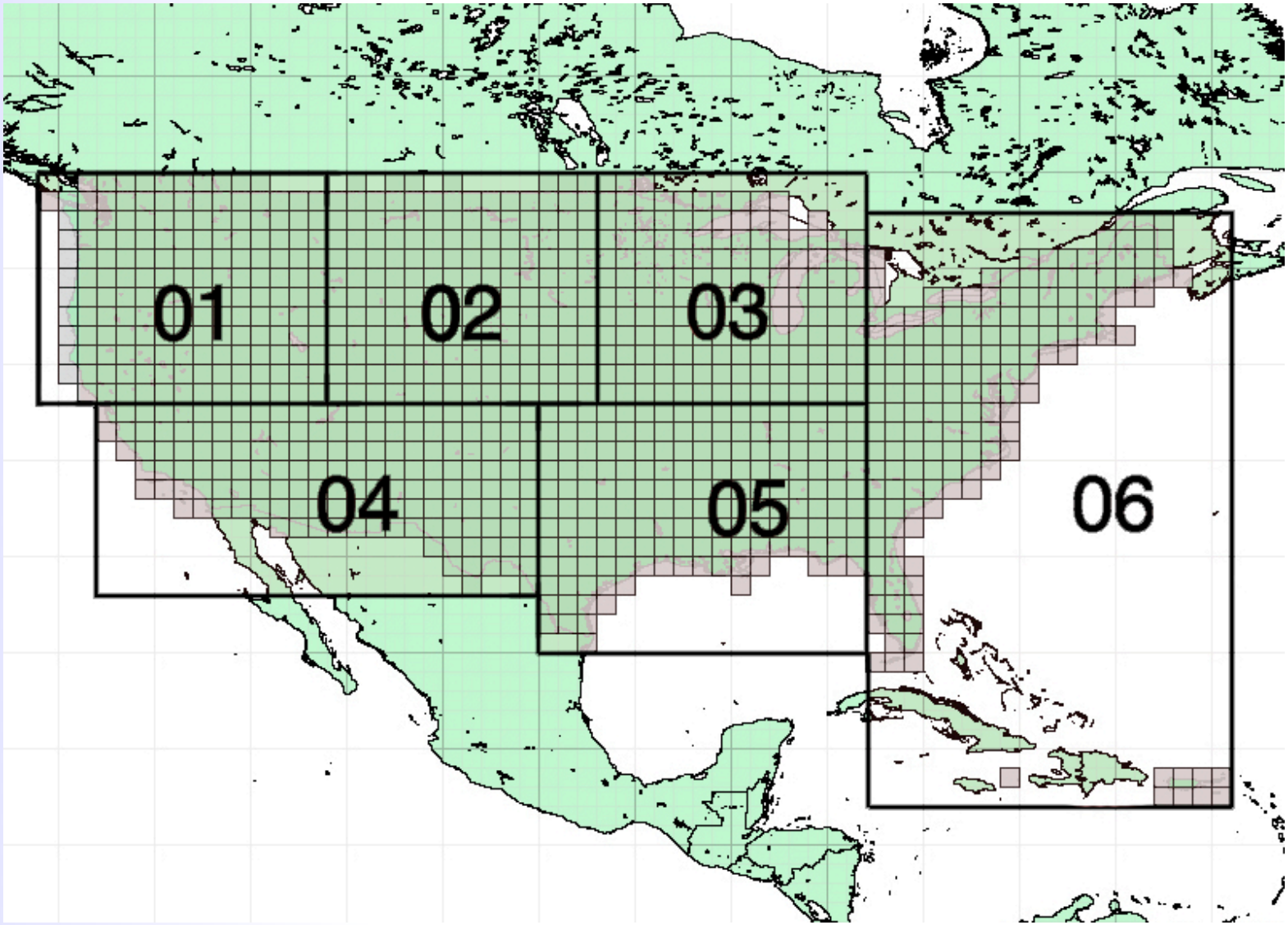
Experimental results

Experimental results

- Implemented algorithms
  - io-radial1 (previous work)
  - io-radial2 (bands)
  - io-radial3 (sectors)
  - io-centrifugal
- Platform:
  - C, gcc 4.1.2
  - HP 200 blade servers
  - 512MB RAM
  - 5400 rpm SATA disks
- Datasets

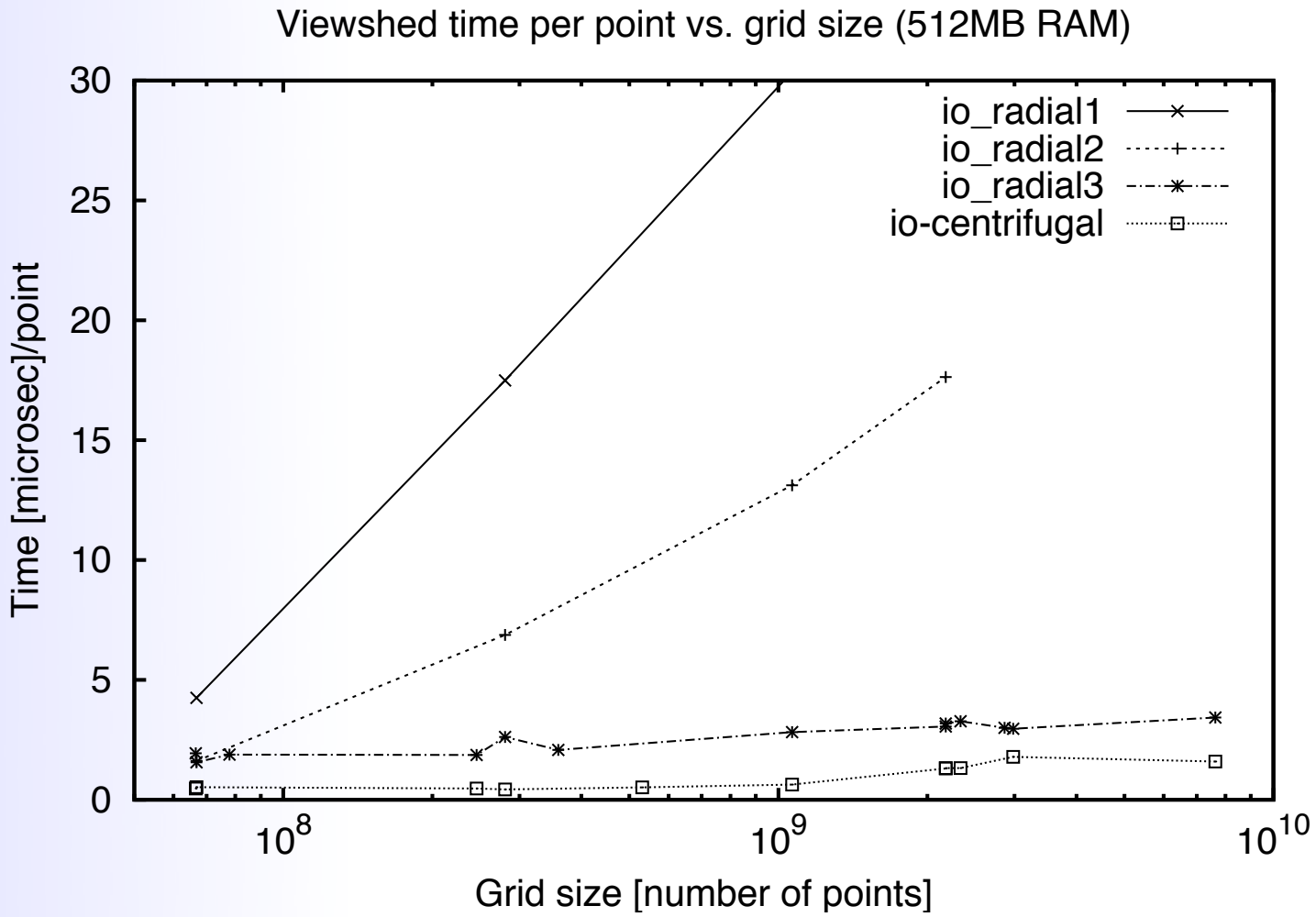
Dataset	Grid Size (rows x columns)	GB
Cumberlands	8,704 x 7,673	.25
USA DEM 6	13,500 x 18,200	.92
USA DEM 2	11,000 x 25,500	1.04
Washington	31,886 x 33,454	3.97
SRTM1-region03	50,401 x 43,201	8.11
SRTM1-region04	82,801 x 36,001	11.10
SRTM1-region06	68,401 x 111,601	28.44

SRTM1 data



30m resolution, available at: ftp://e0srp01.ecs.nasa.gov/srtm

Running times



Running times

Dataset	Grid Size (rows x columns)	GB	io-radial2 (minutes)	io-radial3 (minutes)	io-centrifugal (minutes)
Cumberlands	8,704 x 7,673	.25	1 (84%)	2 (91%)	.5 (49%)
USA DEM 6	13,500 x 18,200	.92	46 (13%)	8 (84%)	2 (57%)
USA DEM 2	11,000 x 25,500	1.04	31 (34%)	12 (87%)	2 (54%)
Washington	31,886 x 33,454	3.97	229 (22%)	50 (89%)	11 (41%)
SRTM1-region03	50,401 x 43,201	8.11	633 (16%)	111 (81%)	47 (17%)
SRTM1-region04	82,801 x 36,001	11.10	-	147 (79%)	89 (11%)
SRTM1-region06	68,401 x 111,601	28.44	-	437 (61%)	203 (18%)

Improved visibility computation on massive grid terrains

Jeremy Fishman, Herman Haverkort and Laura Toma

- New algorithms for computing the visibility map on a grid terrain
- Complexity: sort(n)/scan(n) IOs and O(n lg n) / O(n) CPU
- Efficient in practice
  - 2 passes through the input grid, 1 pass through the output grid
  - e.g. : 28.8GB grid in 200-400 minutes
  - significantly faster than related work
- Grid model: nearest-neighbor interpolation
  - O(\sqrt n) obstacles along any LOS
- Extensions to other models possible

Thank you