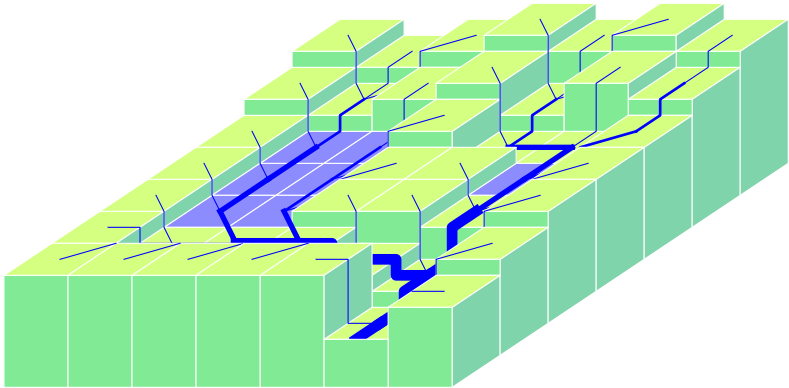
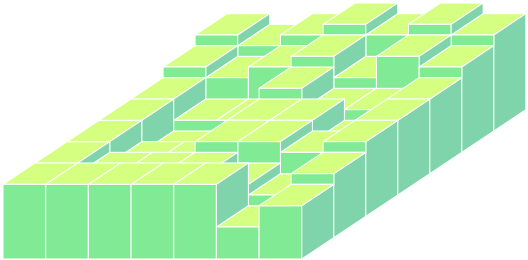
Simple I/O-efficient flow accumulation on grid terrains



Herman Haverkort
TU Eindhoven

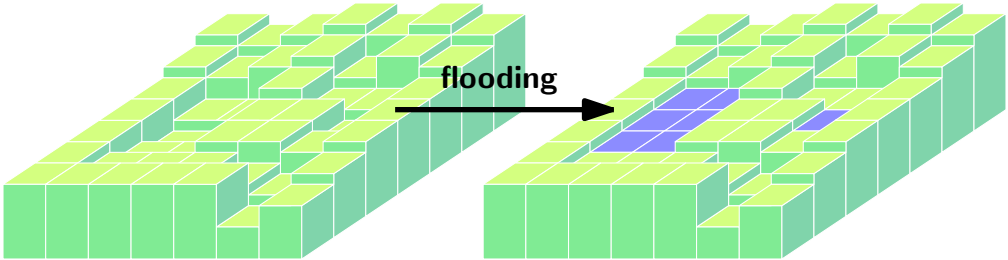
Jeffrey Janssen
Realworld Systems

Drainage network analysis



grid elevation model

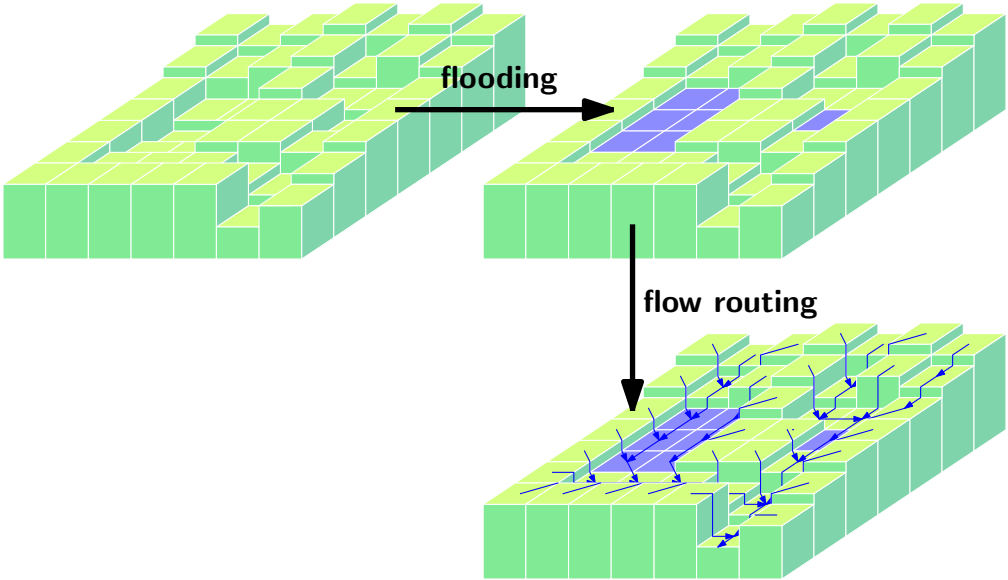
Drainage network analysis



grid elevation model

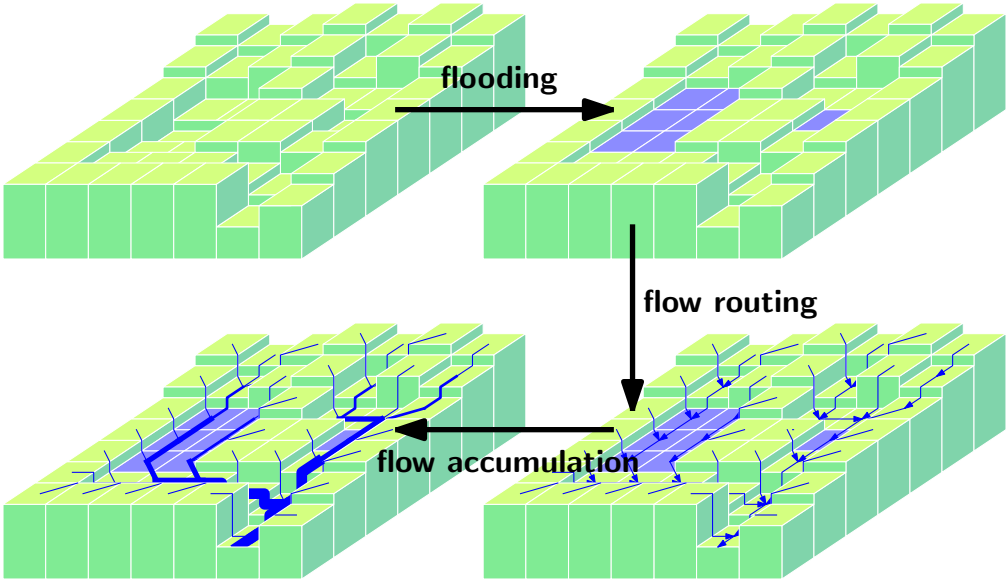
each cell has non-ascending path to boundary

Drainage network analysis



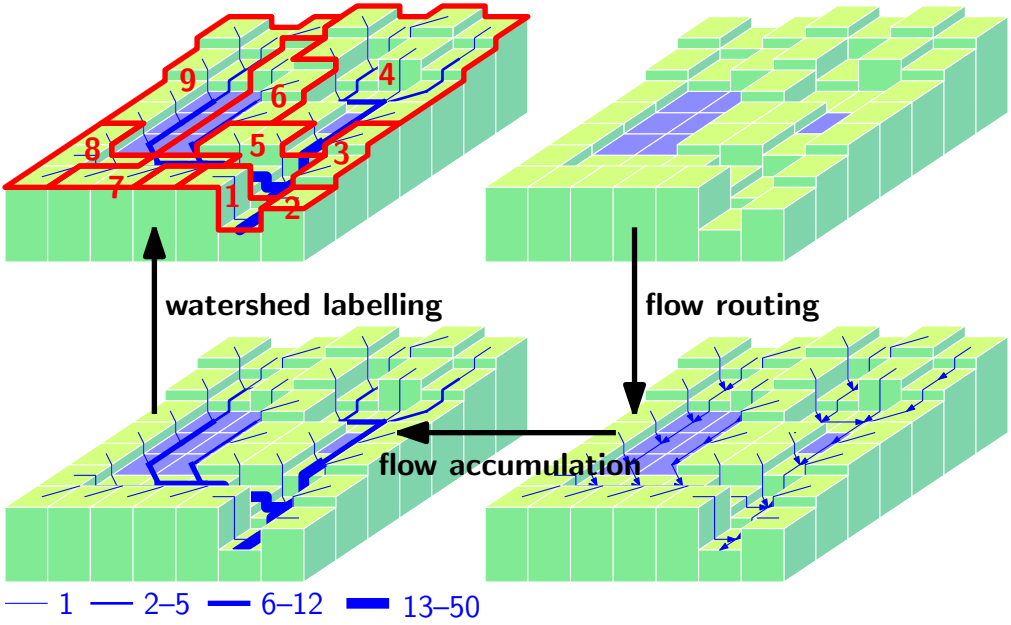
each cell has a flow direction; following flow directions leads to boundary

Drainage network analysis

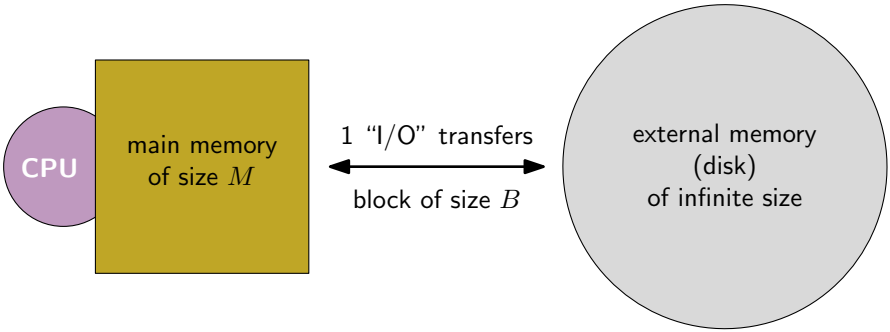


compute for each cell c, from how many cells water passes through c

Drainage network analysis



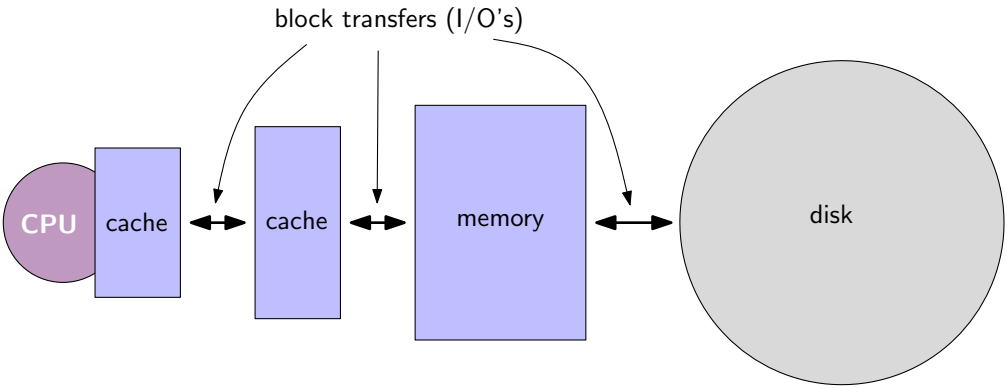
Analysing I/O-efficiency



CPU only operates on data in main memory (for free)

I/O-efficiency = number of I/O's as function of M, B, and grid size N
(sometimes assume $M \geq c \cdot B^2$)

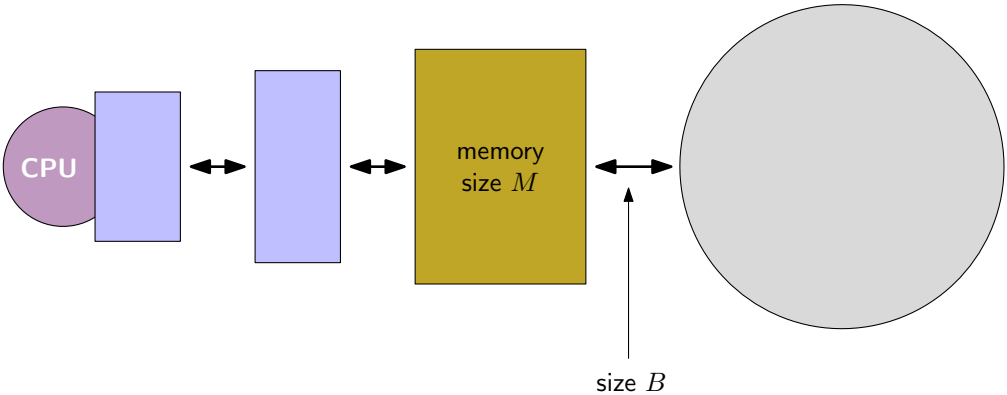
Analysing I/O-efficiency



CPU only operates on data in main memory (for free)

I/O-efficiency = number of I/O's as function of M, B, and grid size N
(sometimes assume $M \geq c \cdot B^2$)

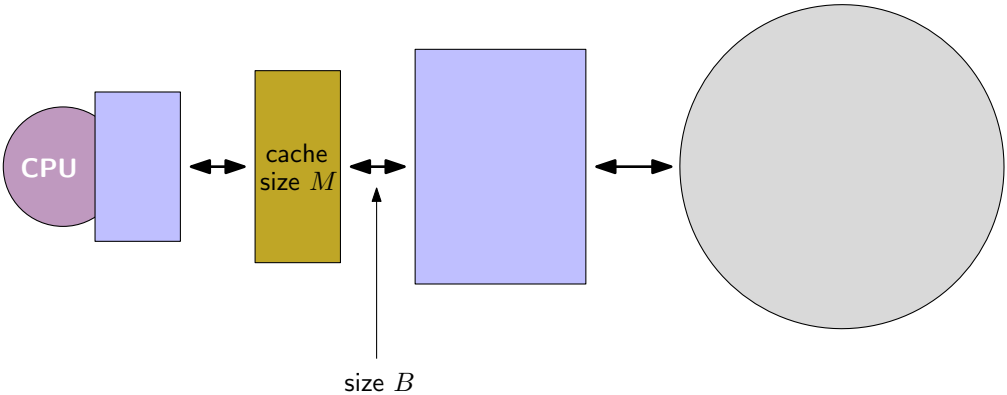
Analysing I/O-efficiency



CPU only operates on data in main memory (for free)

I/O-efficiency = number of I/O's as function of M, B, and grid size N
(sometimes assume $M \geq c \cdot B^2$)

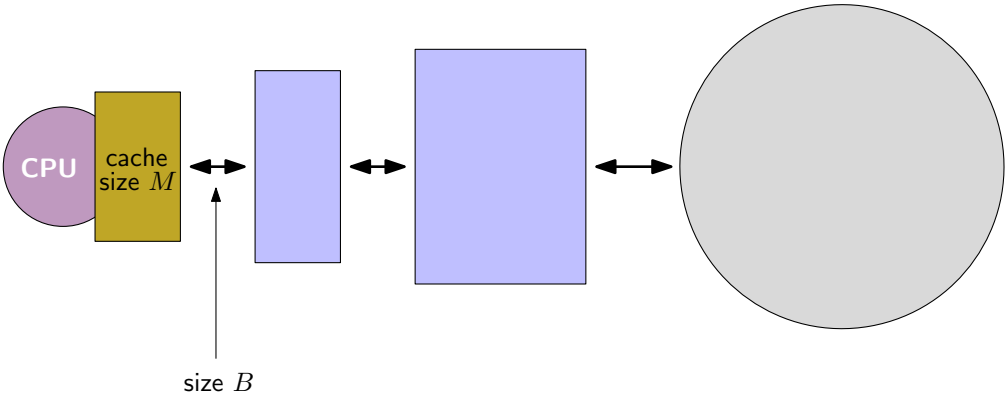
Analysing I/O-efficiency



CPU only operates on data in main memory (for free)

I/O-efficiency = number of I/O's as function of M, B, and grid size N
(sometimes assume $M \geq c \cdot B^2$)

Analysing I/O-efficiency

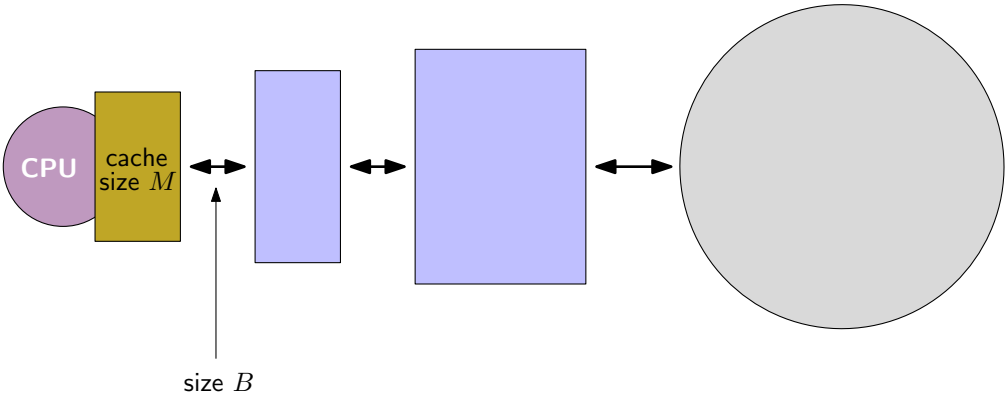


CPU only operates on data in main memory (for free)

I/O-efficiency = number of I/O's as function of M, B, and grid size N
(sometimes assume $M \geq c \cdot B^2$)

Analysing I/O-efficiency

cache-aware algorithms: know M and B, control caching
cache-oblivious algorithms: do not know M and B, caching left to system

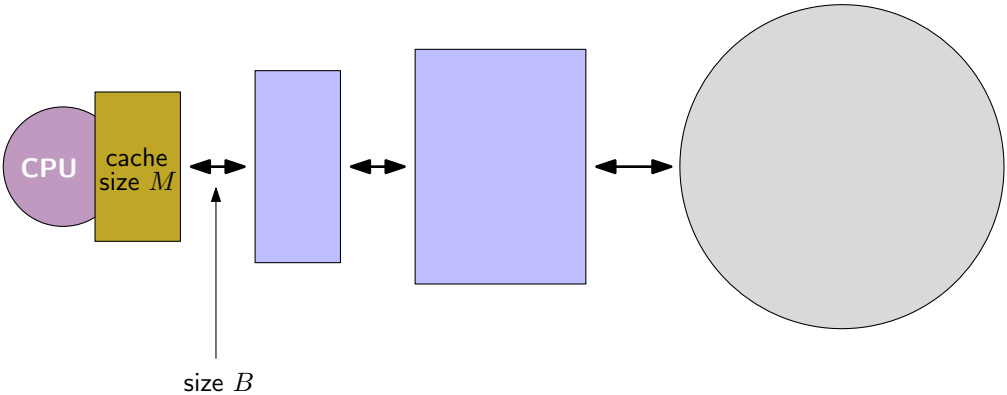


CPU only operates on data in main memory (for free)

I/O-efficiency = number of I/O's as function of M, B, and grid size N
(sometimes assume $M \geq c \cdot B^2$)

Analysing I/O-efficiency

cache-aware algorithms: know M and B, control caching
cache-oblivious algorithms: do not know M and B, caching left to system

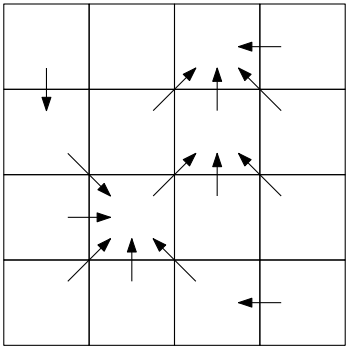


CPU only operates on data in main memory (for free)

scanning N cells takes $\Theta(\mathrm{scan}(N)) = \Theta(N/B)$ I/O's
sorting N cells takes $\Theta(\mathrm{sort}(N)) = \Theta(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/O's

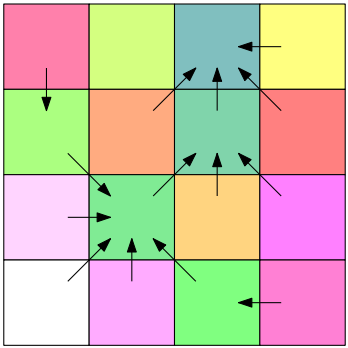
Time-forward processing (Arge et al.)

Goal: compute flow accumulation for each cell c
= #cells from which water passes through c
= size of tree rooted at c



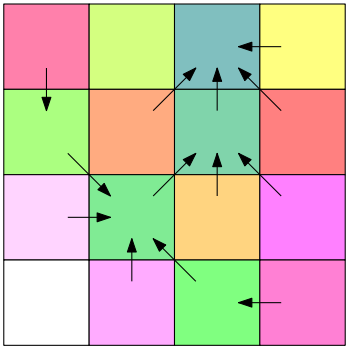
Time-forward processing (Arge et al.)

Goal: compute flow accumulation for each cell c
= #cells from which water passes through c
= size of tree rooted at c



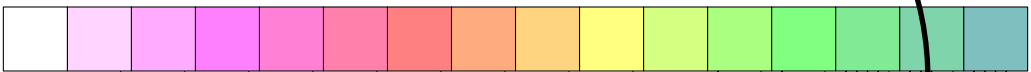
Time-forward processing (Arge et al.)

Goal: compute flow accumulation for each cell c
= #cells from which water passes through c
= size of tree rooted at c



sort

output

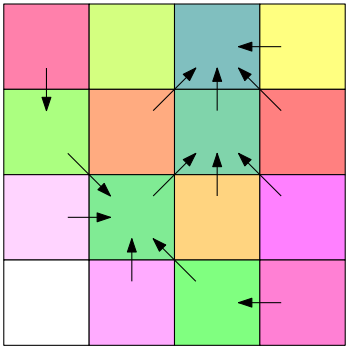


input

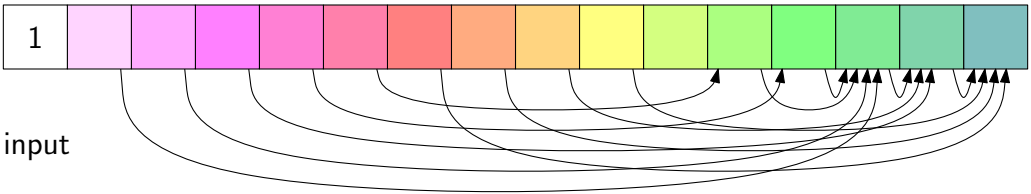
pqueue

Time-forward processing (Arge et al.)

Goal: compute flow accumulation for each cell c
= #cells from which water passes through c
= size of tree rooted at c



output

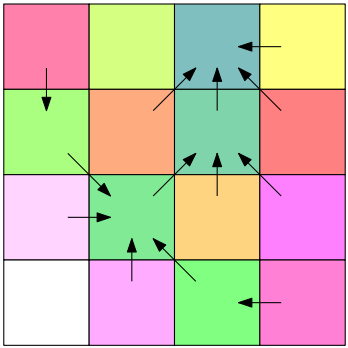


input

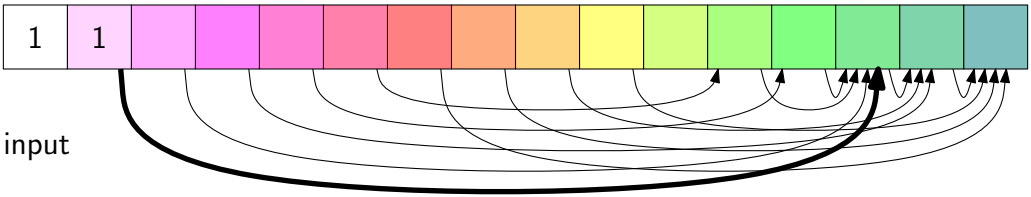
pqueue

Time-forward processing (Arge et al.)

Goal: compute flow accumulation for each cell c
= #cells from which water passes through c
= size of tree rooted at c



output

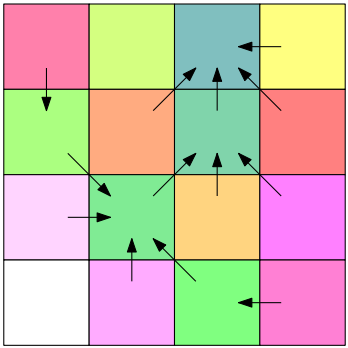


input

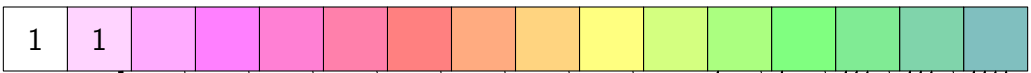
pqueue

Time-forward processing (Arge et al.)

Goal: compute flow accumulation for each cell c
= #cells from which water passes through c
= size of tree rooted at c



output

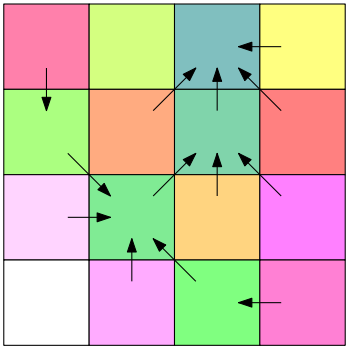


input

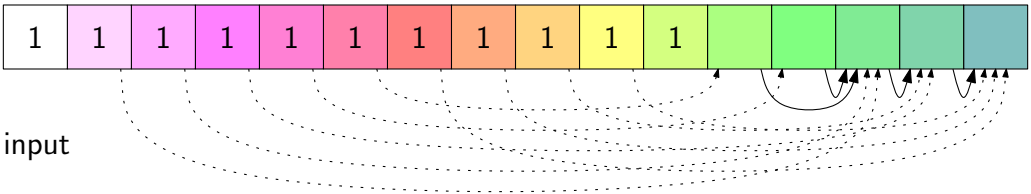
pqueue

Time-forward processing (Arge et al.)

Goal: compute flow accumulation for each cell c
= #cells from which water passes through c
= size of tree rooted at c

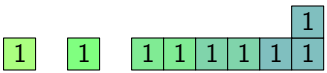


output



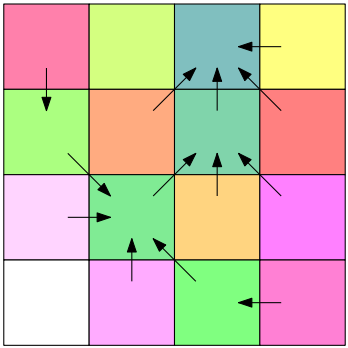
input

pqueue

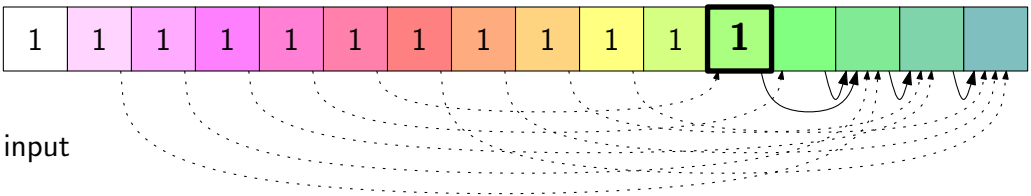


Time-forward processing (Arge et al.)

Goal: compute flow accumulation for each cell c
= #cells from which water passes through c
= size of tree rooted at c

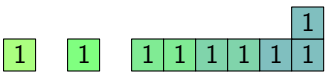


output



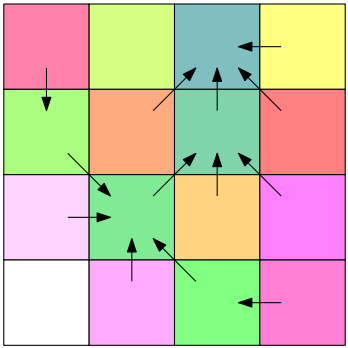
input

pqueue

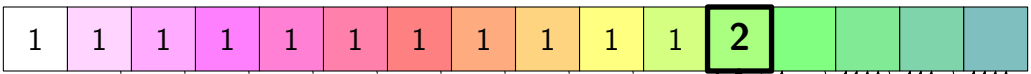


Time-forward processing (Arge et al.)

Goal: compute flow accumulation for each cell c
= #cells from which water passes through c
= size of tree rooted at c

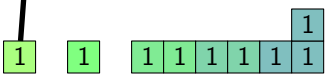


output



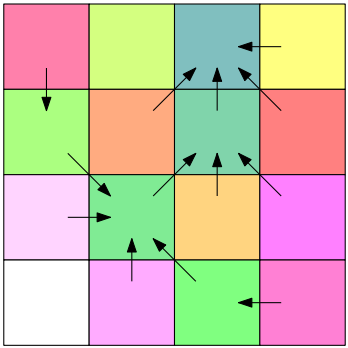
input

pqueue

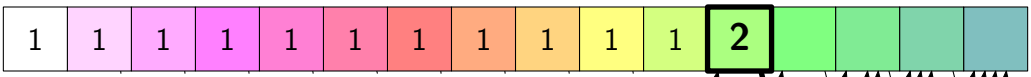


Time-forward processing (Arge et al.)

Goal: compute flow accumulation for each cell c
= #cells from which water passes through c
= size of tree rooted at c

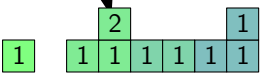


output



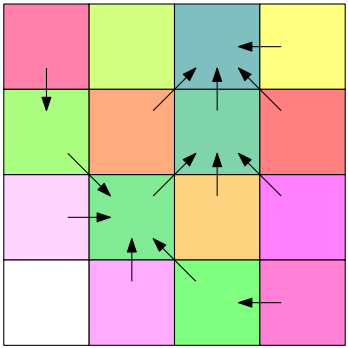
input

pqueue

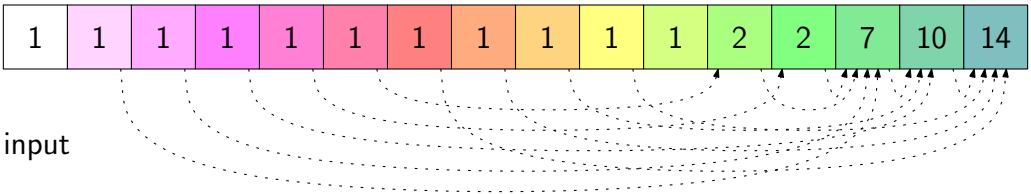


Time-forward processing (Arge et al.)

Goal: compute flow accumulation for each cell c
= #cells from which water passes through c
= size of tree rooted at c



output

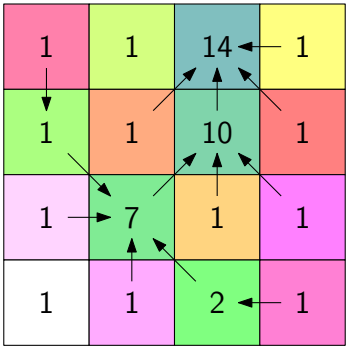


input

pqueue

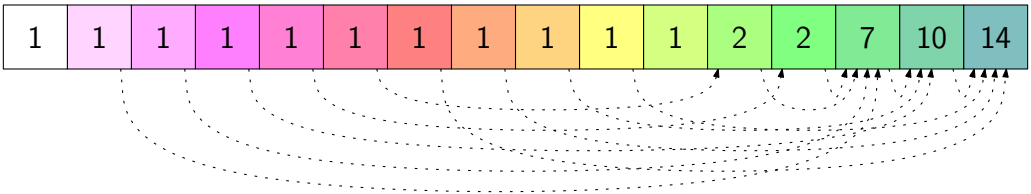
Time-forward processing (Arge et al.)

Goal: compute flow accumulation for each cell c
= #cells from which water passes through c
= size of tree rooted at c



sort

output



Time-forward processing

Worst-case I/O's: $\Theta(\mathrm{sort}(N))$ (Arge et al.)

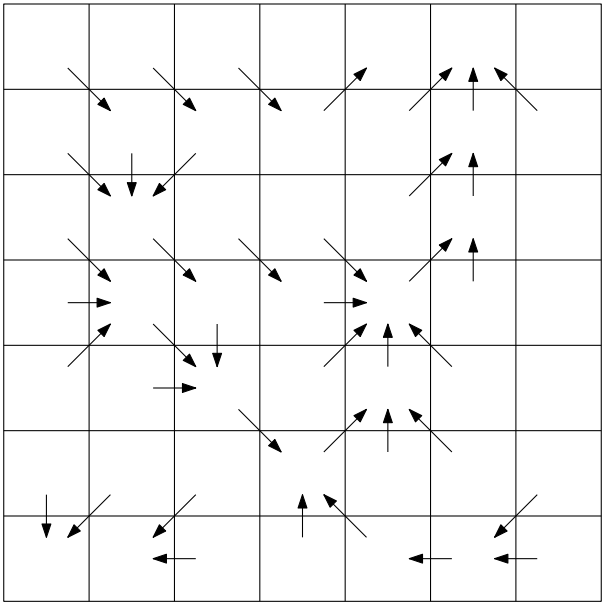
I/O-volume per grid cell (optimistic):

Sorting grid into list of (xy-location, topological nr., out-neighbour top. nr.)	$2 \times 2 \times 24 = 96$ bytes
Flow accumulation, input: Flow accumulation, output: (xy-location, flow)	24 bytes 16 bytes
Sorting output into grid	$2 \times 2 \times 16 = 64$ bytes
Total:	**200 bytes**

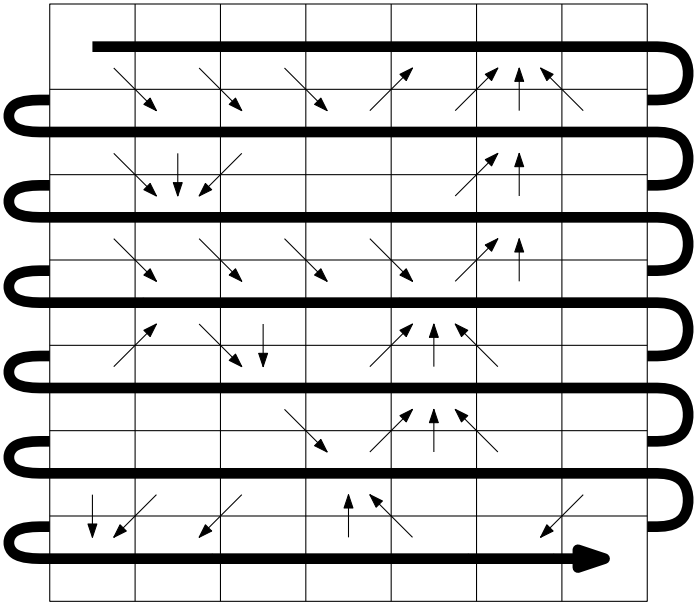
I/O-volume per grid cell (pessimistic):

Sorting grid into list of (xy-location, topological nr., out-neighbour top. nr.)	$3 \times 2 \times 24 = 144$ bytes
Flow accumulation, input: Flow accumulation, priority queue: Flow accumulation, output: (xy-location, flow)	24 bytes $2 \times 16 = 32$ bytes 16 bytes
Sorting output into grid	$3 \times 2 \times 16 = 96$ bytes
Total:	**312 bytes**

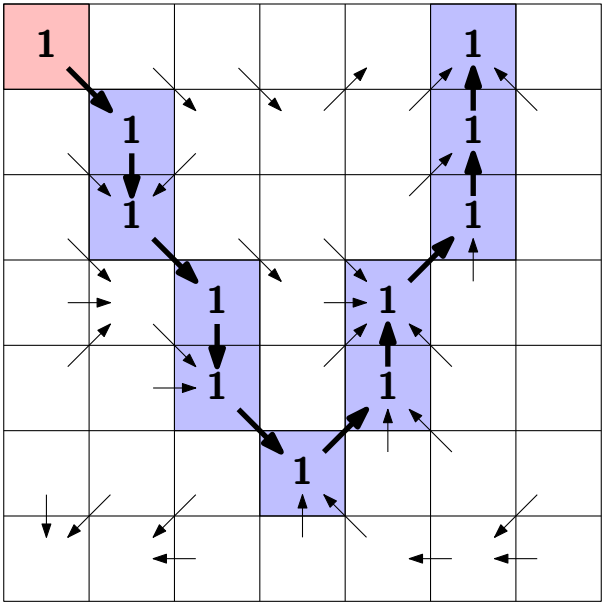
Naïve algorithm



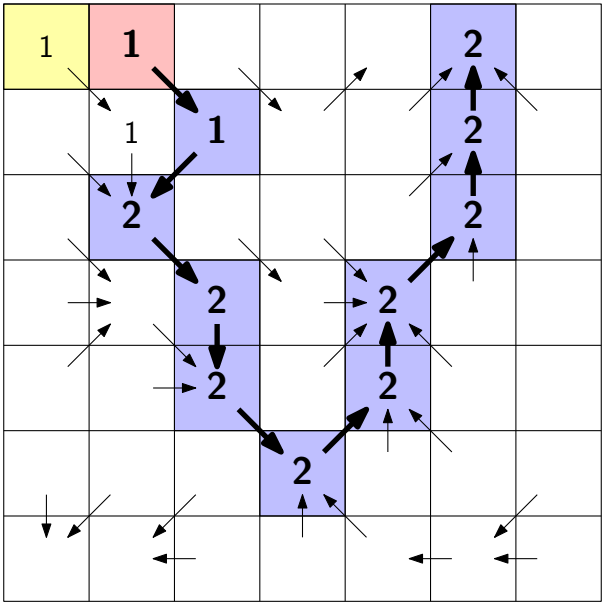
Naïve algorithm



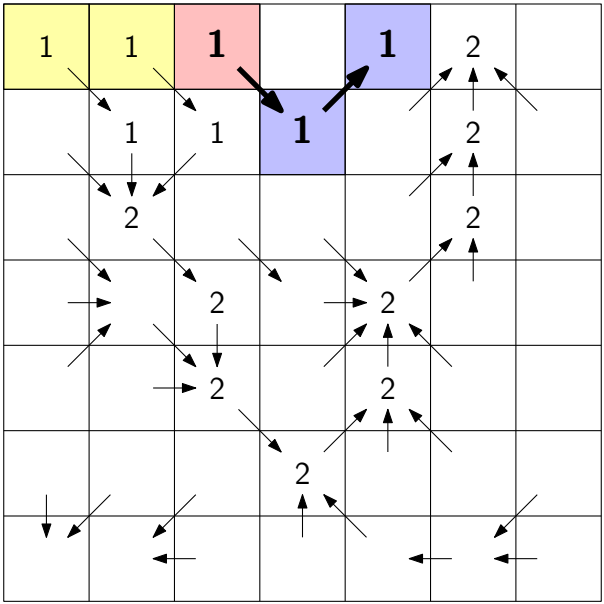
Naïve algorithm



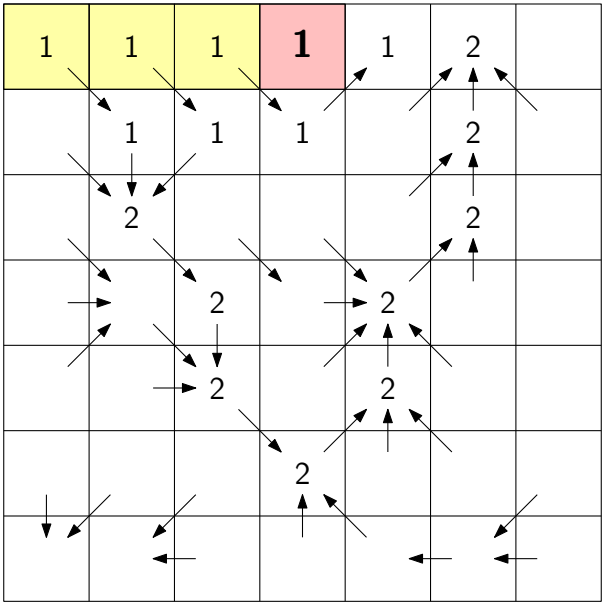
Naïve algorithm



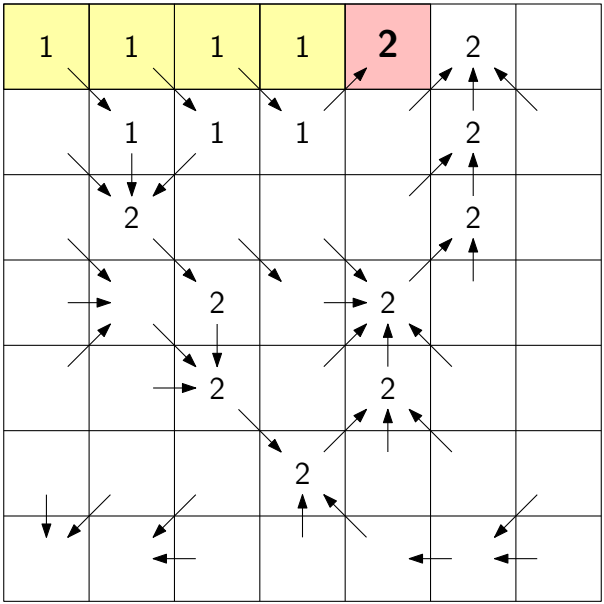
Naïve algorithm



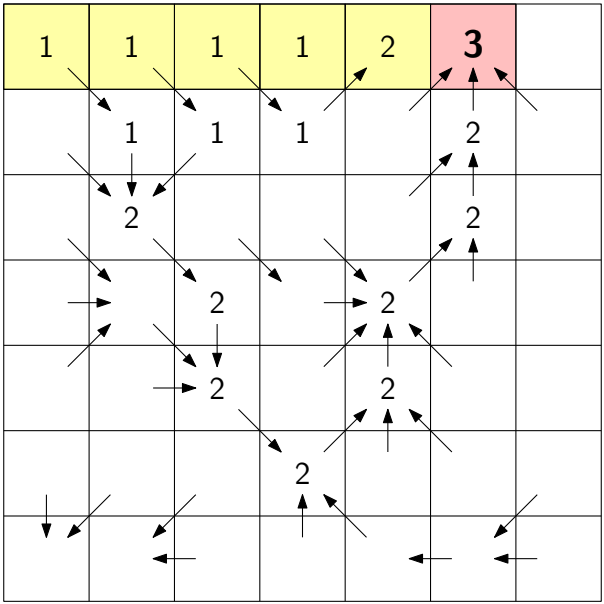
Naïve algorithm



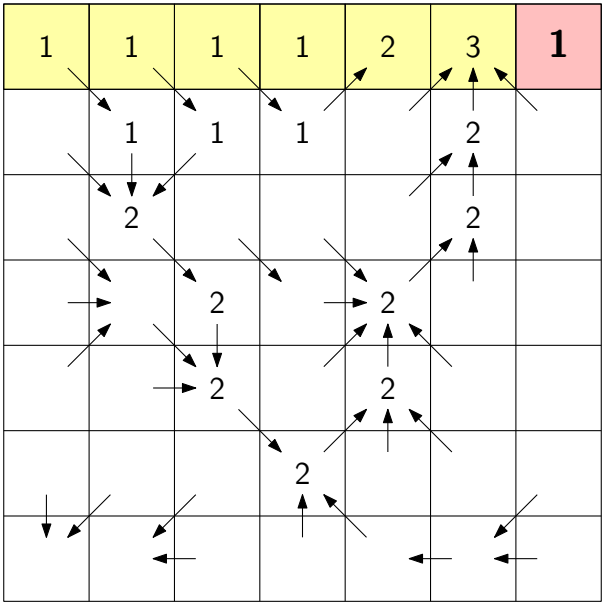
Naïve algorithm



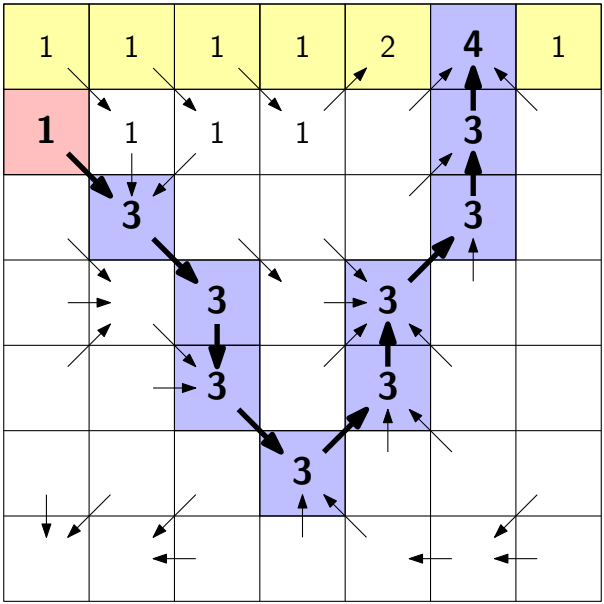
Naïve algorithm



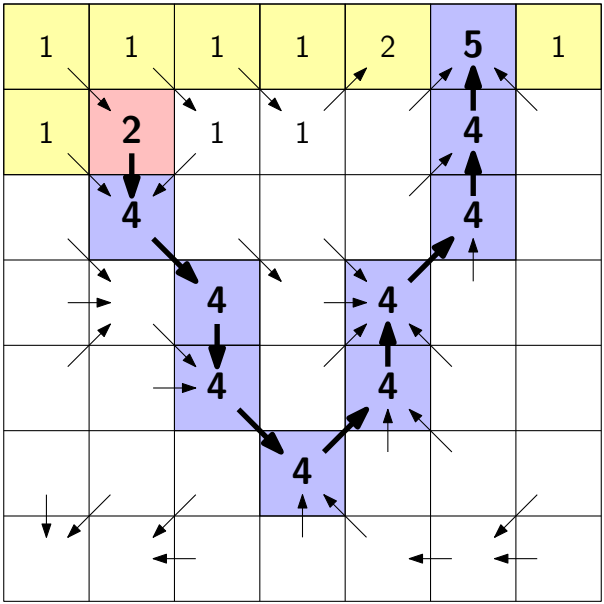
Naïve algorithm



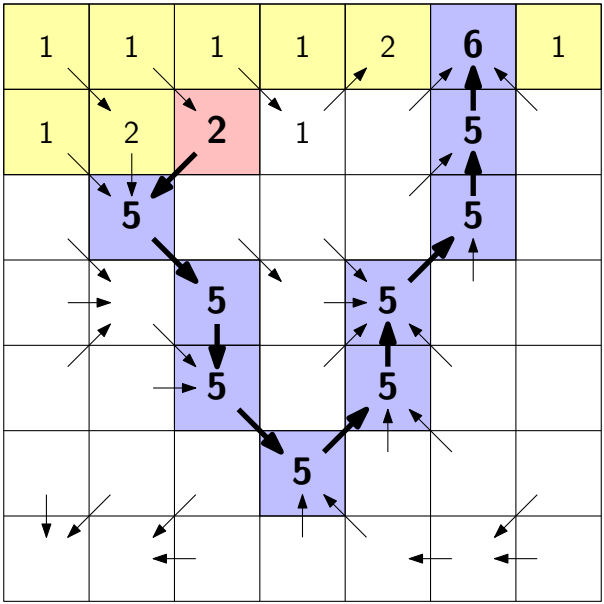
Naïve algorithm



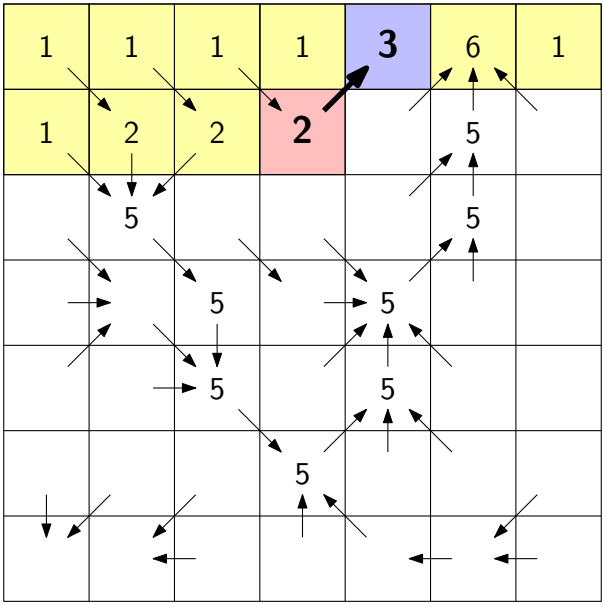
Naïve algorithm



Naïve algorithm

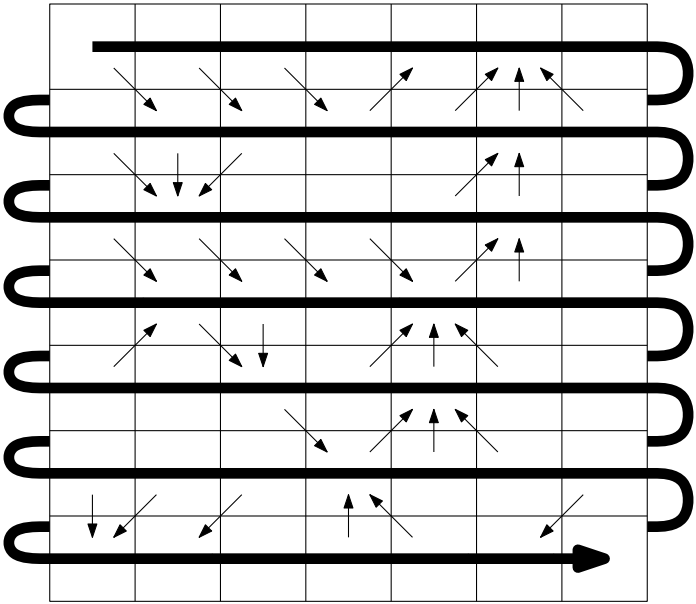


Naïve algorithm

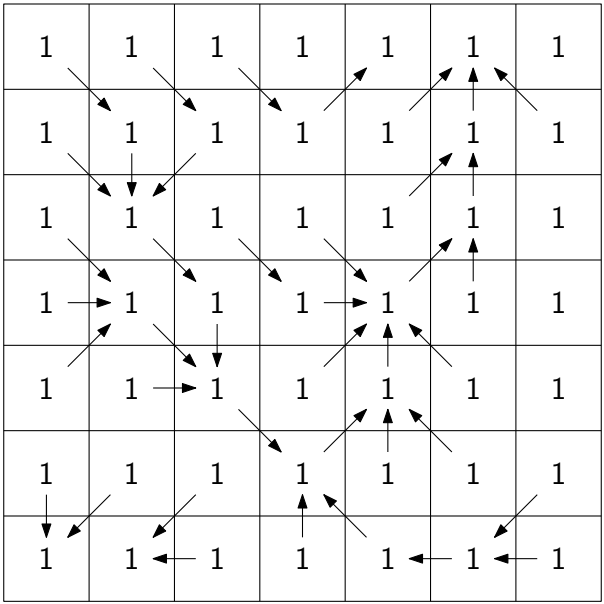


Worst-case running time: $\Theta(n^2)$

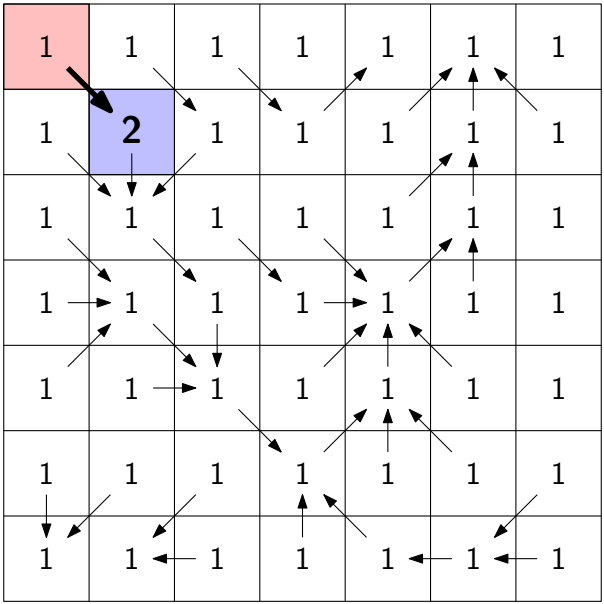
Row-by-row scan



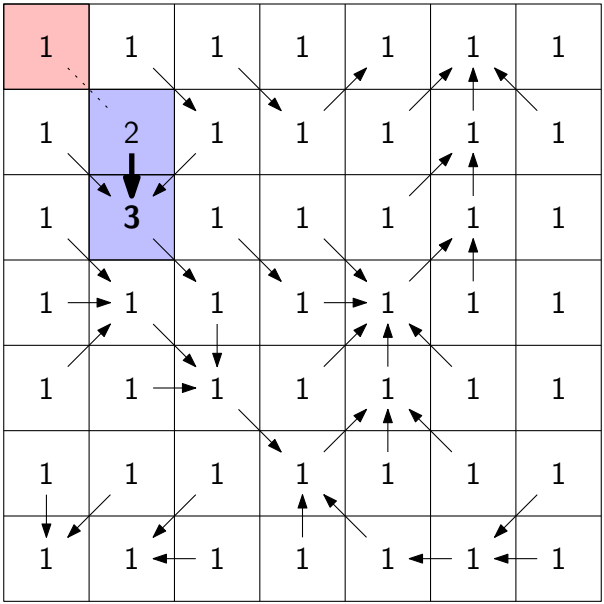
Row-by-row scan



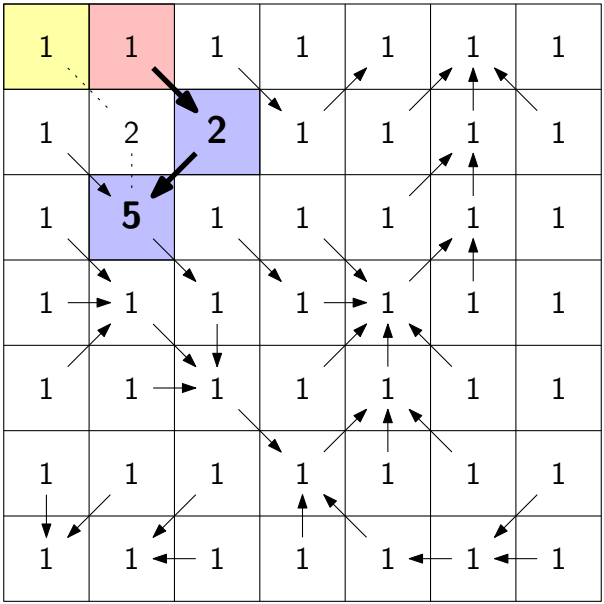
Row-by-row scan



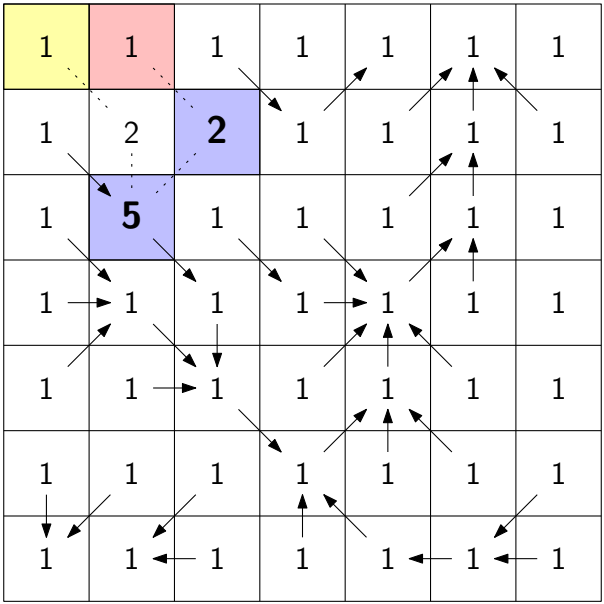
Row-by-row scan



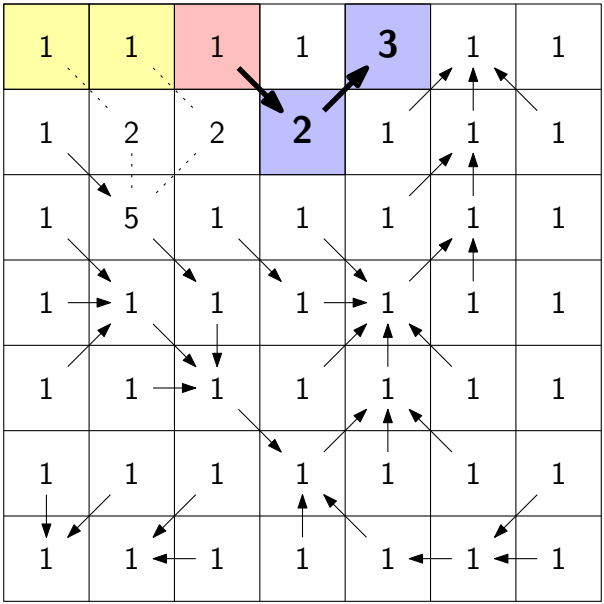
Row-by-row scan



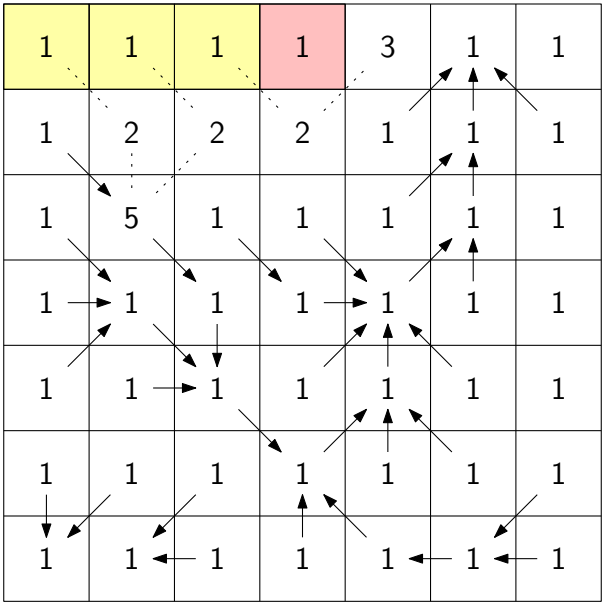
Row-by-row scan



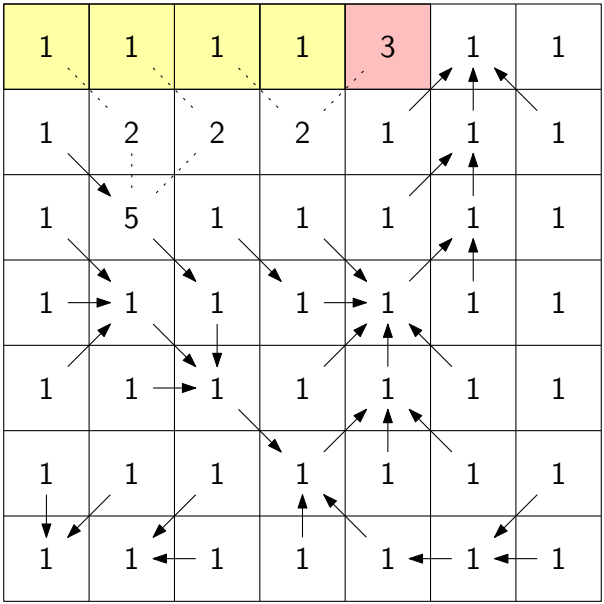
Row-by-row scan



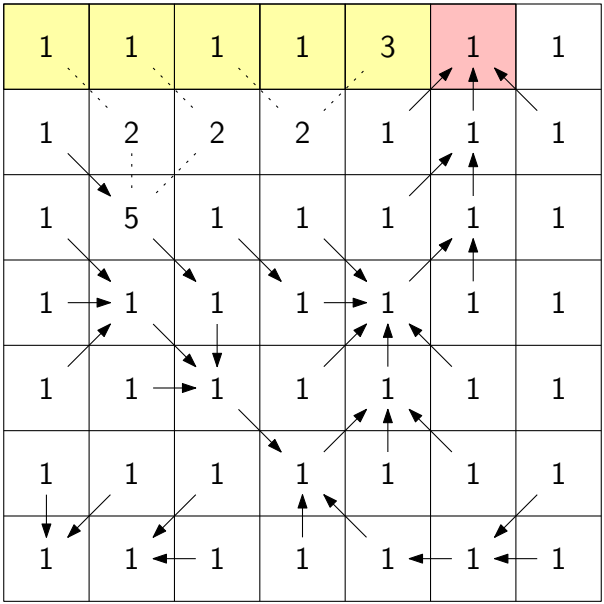
Row-by-row scan



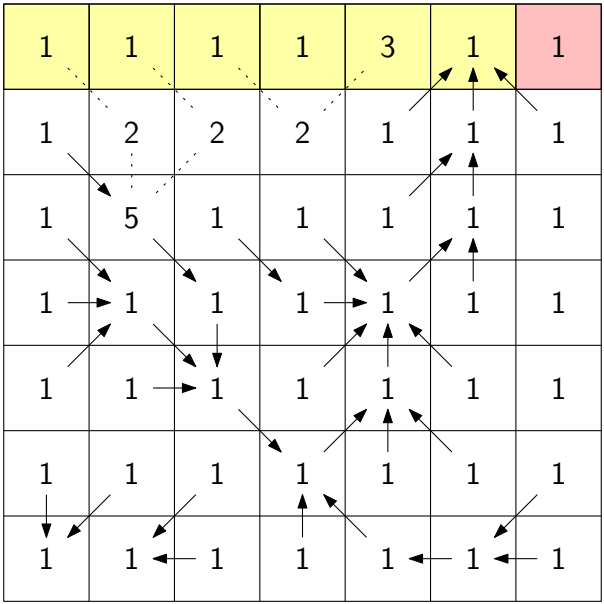
Row-by-row scan



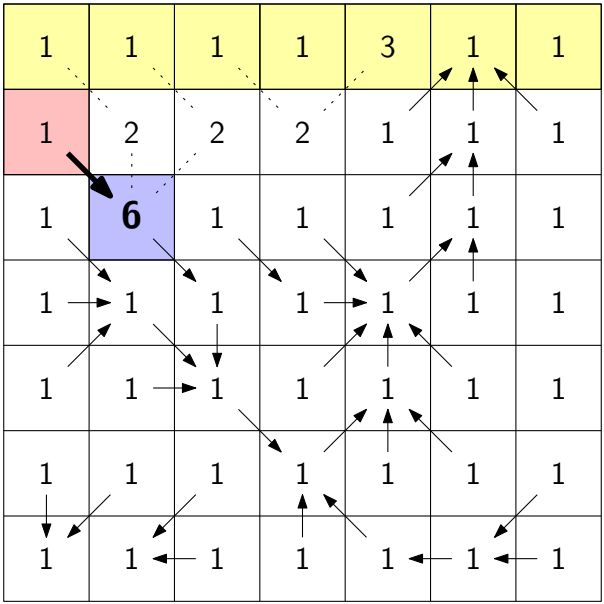
Row-by-row scan



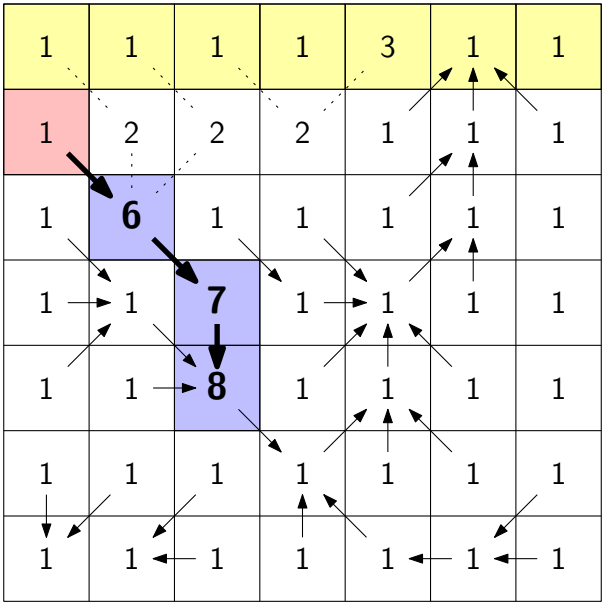
Row-by-row scan



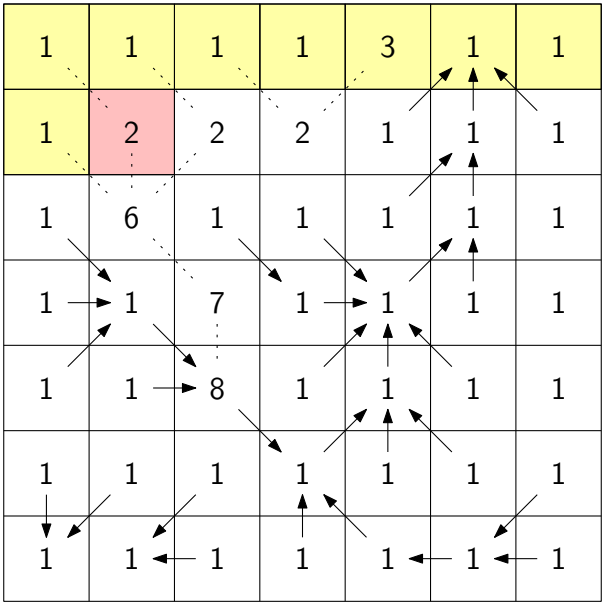
Row-by-row scan



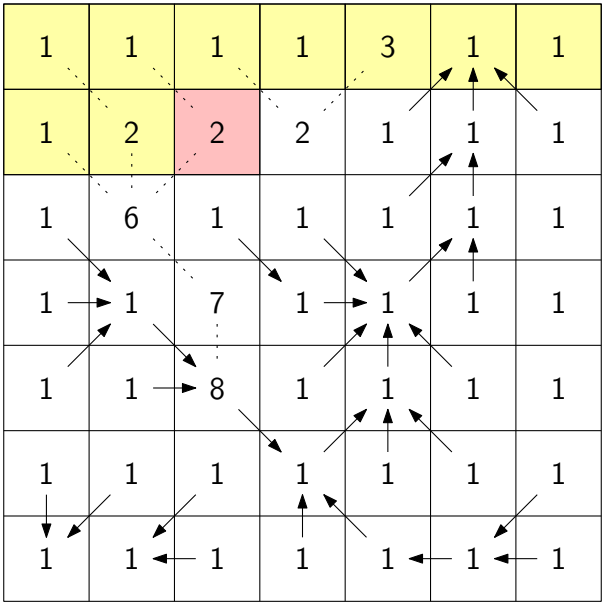
Row-by-row scan



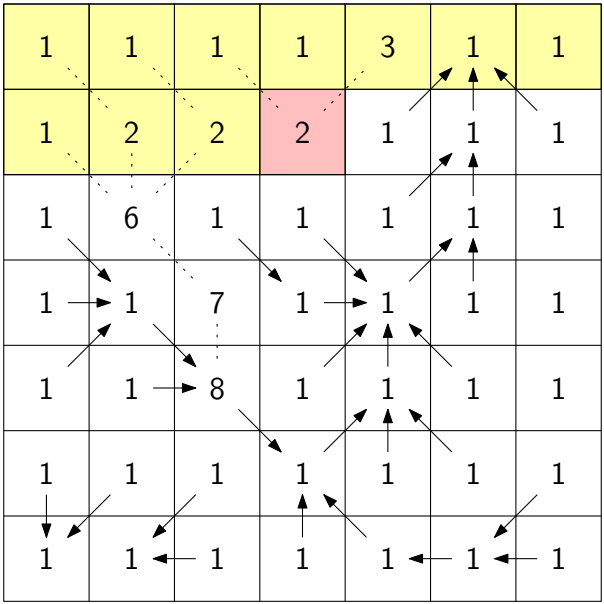
Row-by-row scan



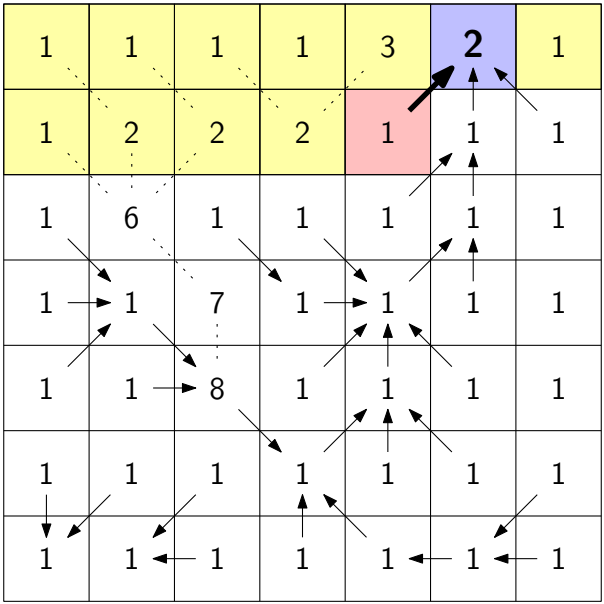
Row-by-row scan



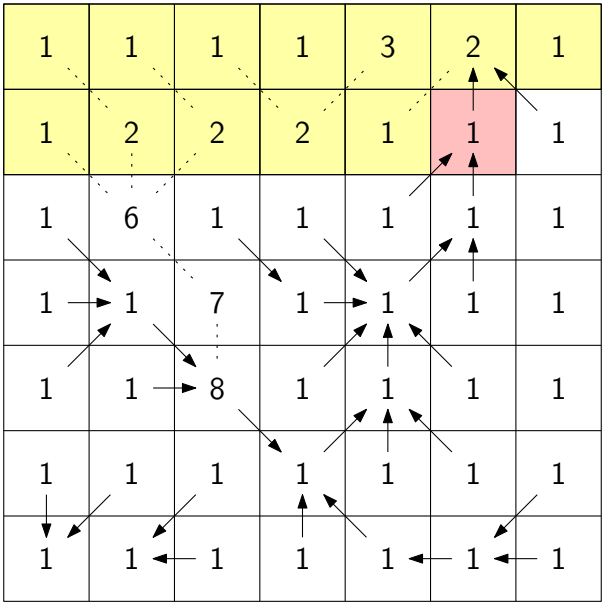
Row-by-row scan



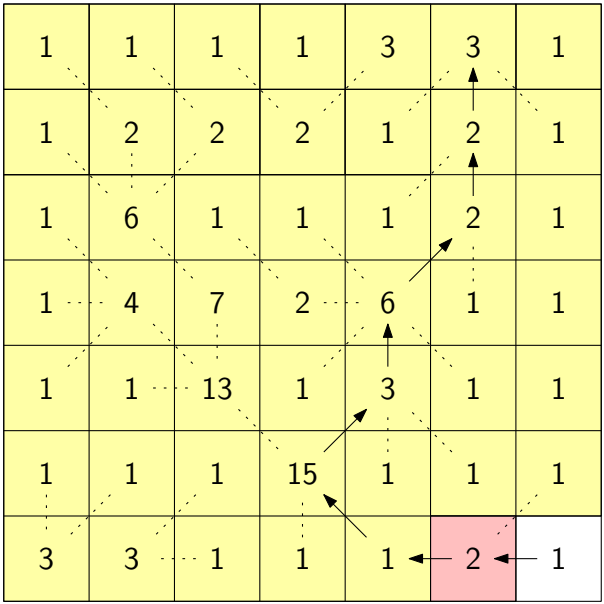
Row-by-row scan



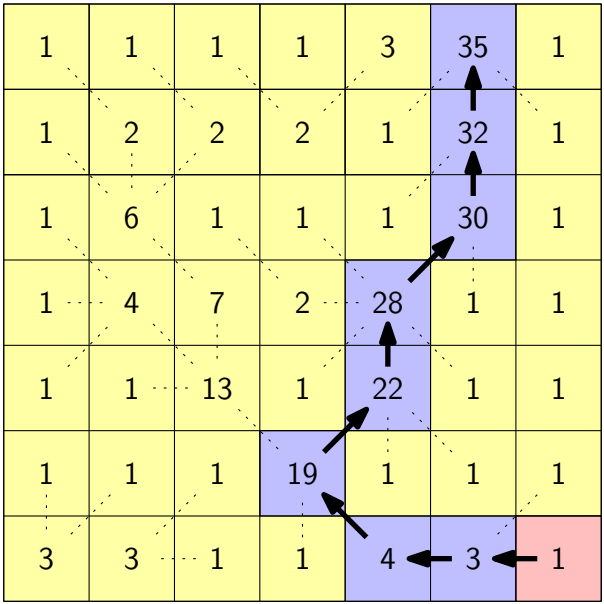
Row-by-row scan



Row-by-row scan



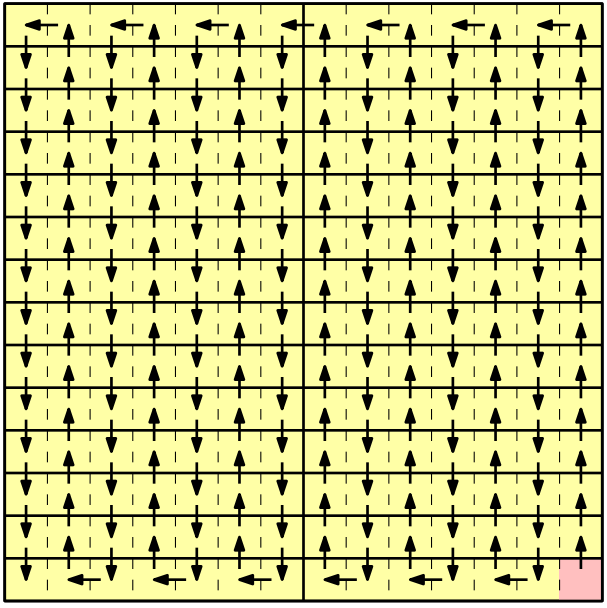
Row-by-row scan



Running time: $\Theta(n)$

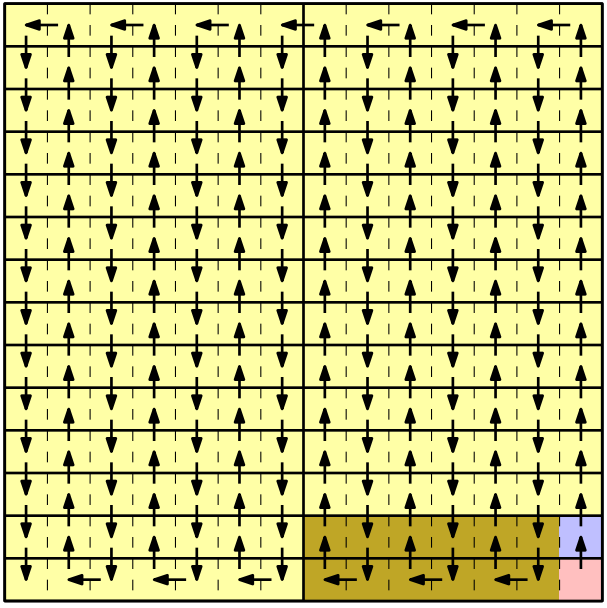
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



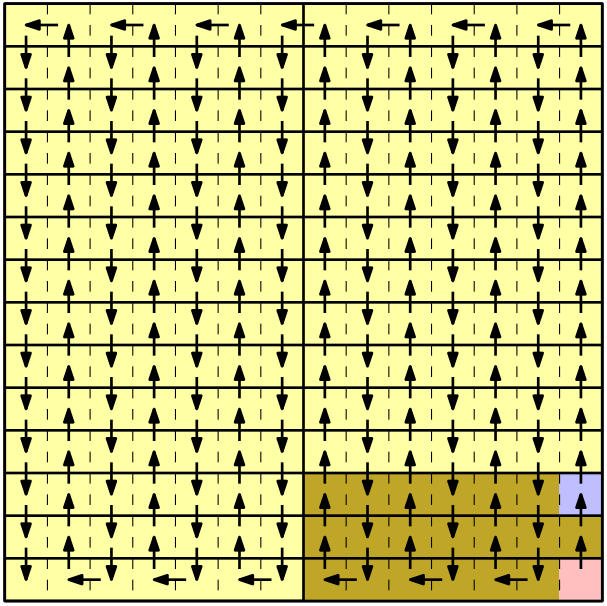
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



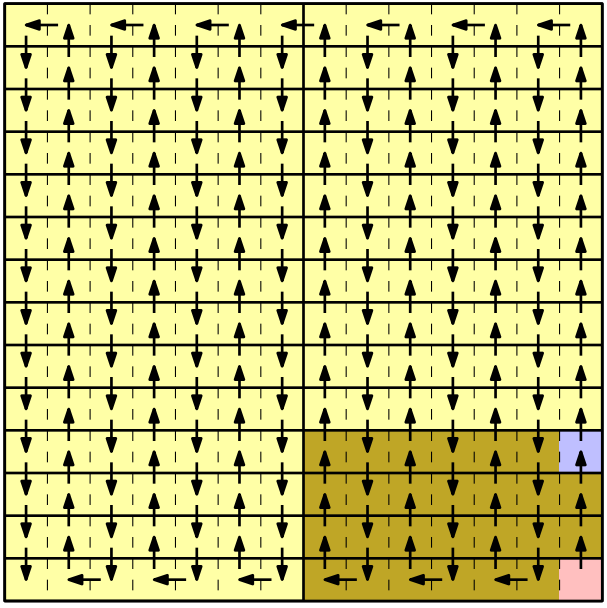
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



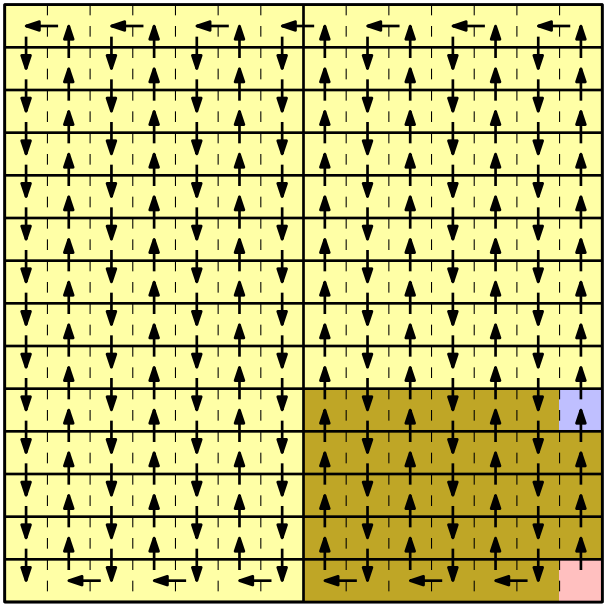
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



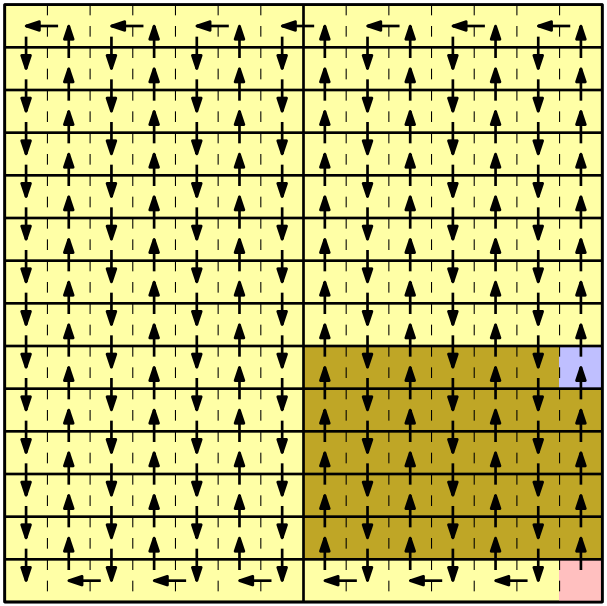
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



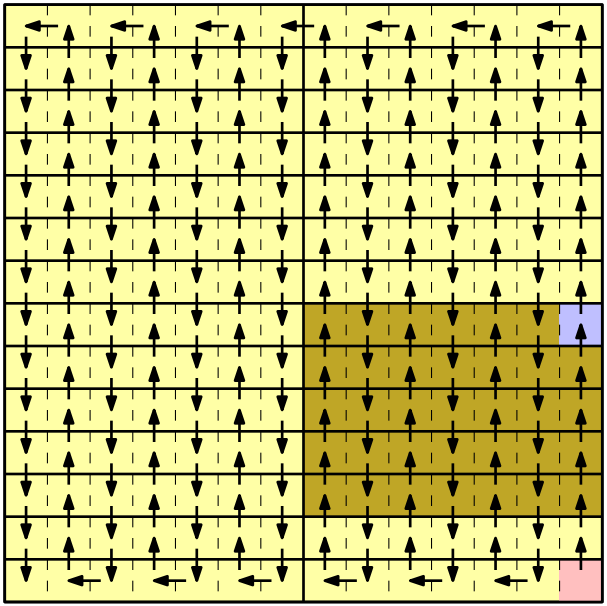
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



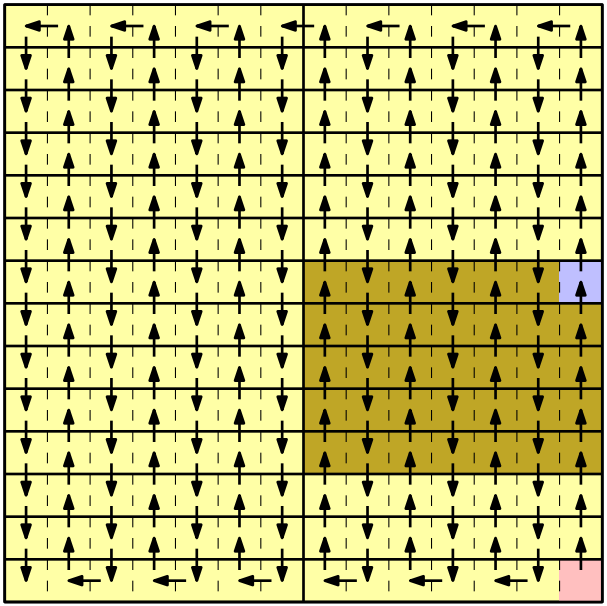
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



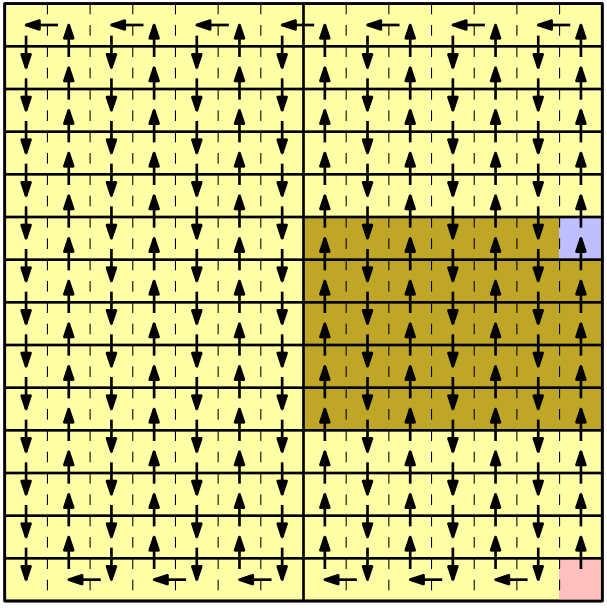
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



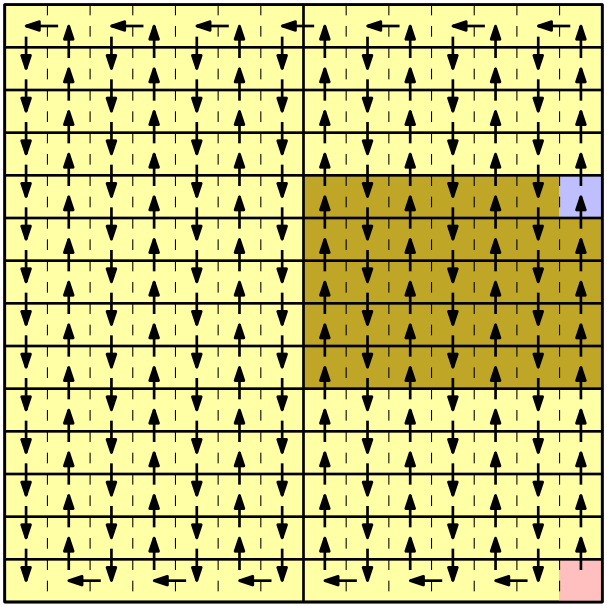
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



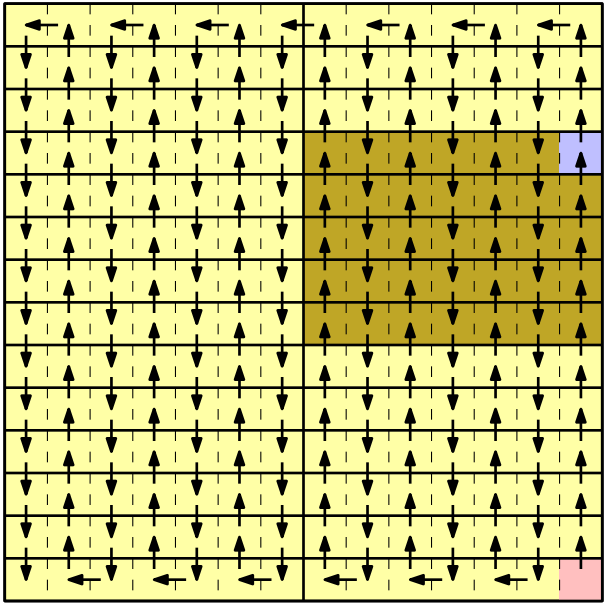
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



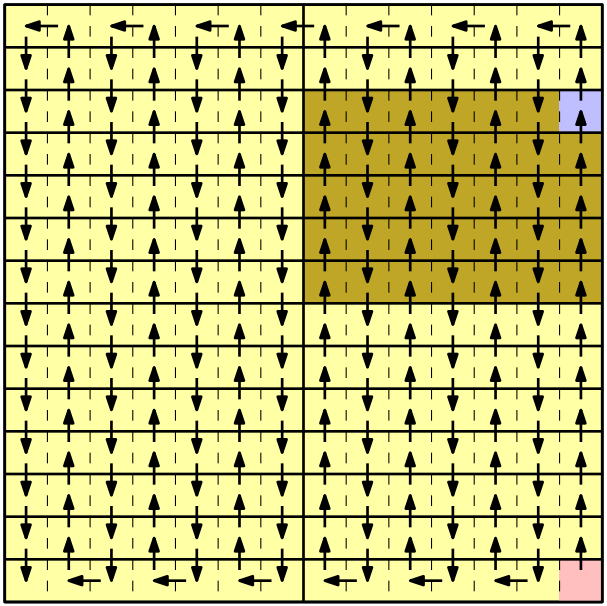
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



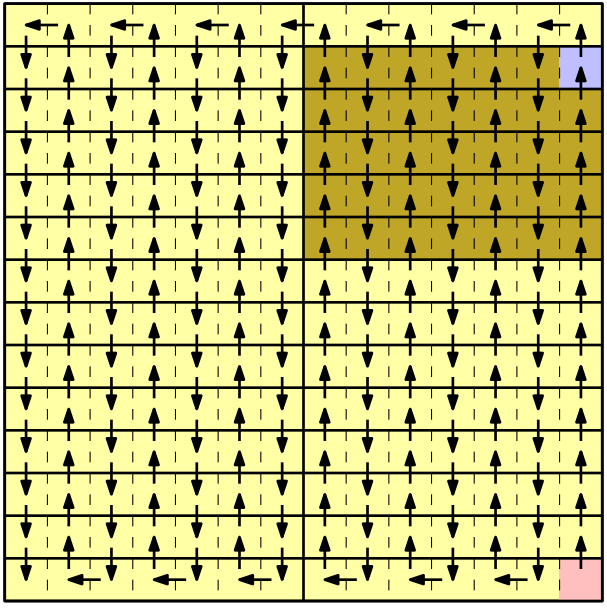
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



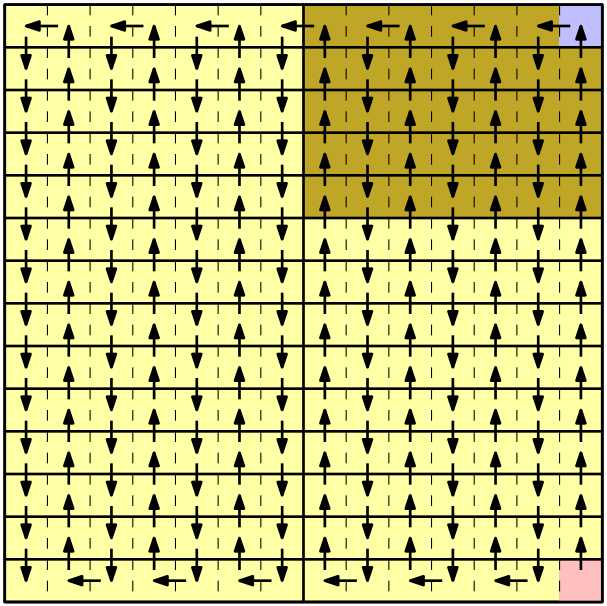
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



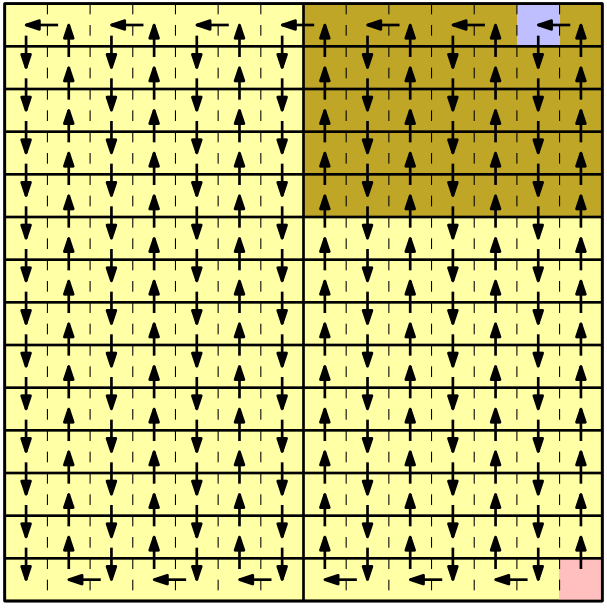
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



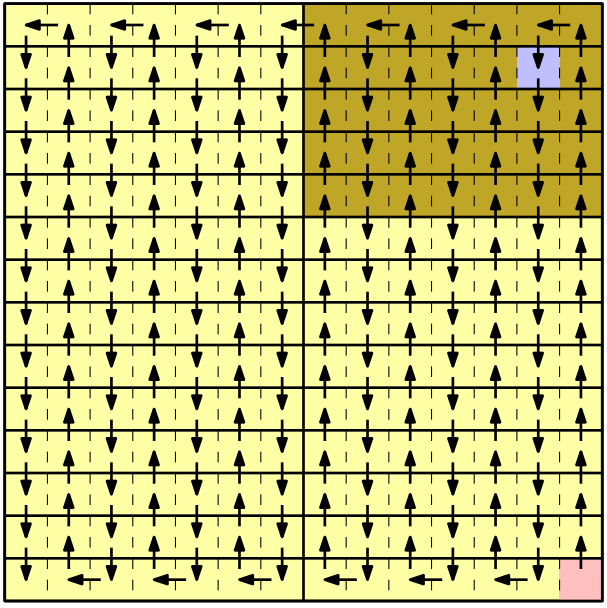
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



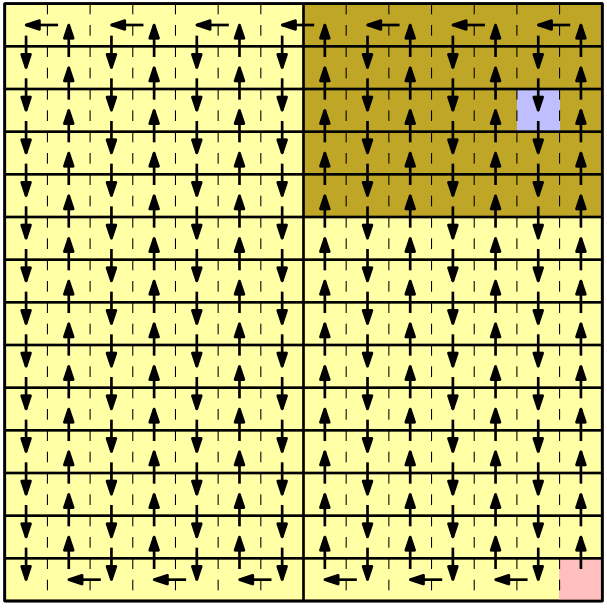
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



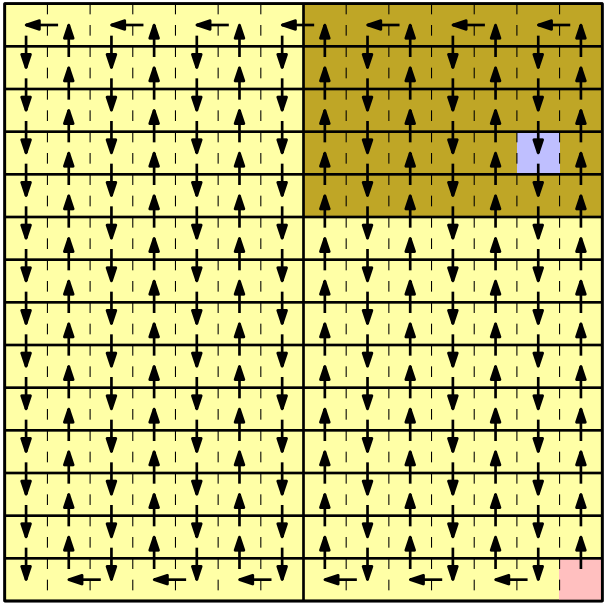
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



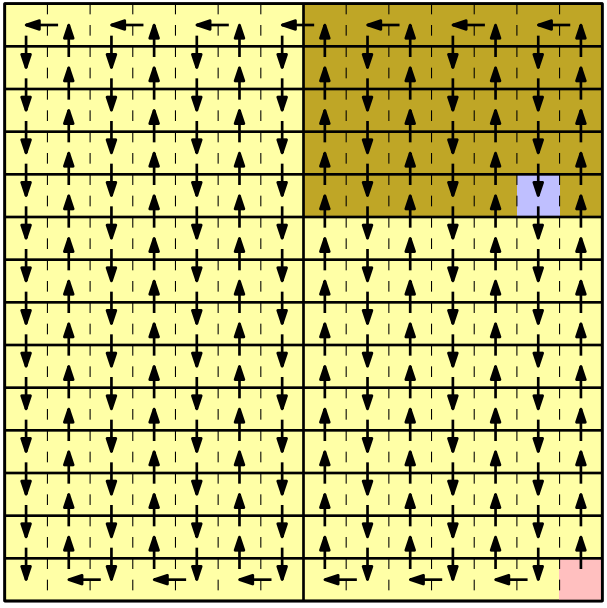
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



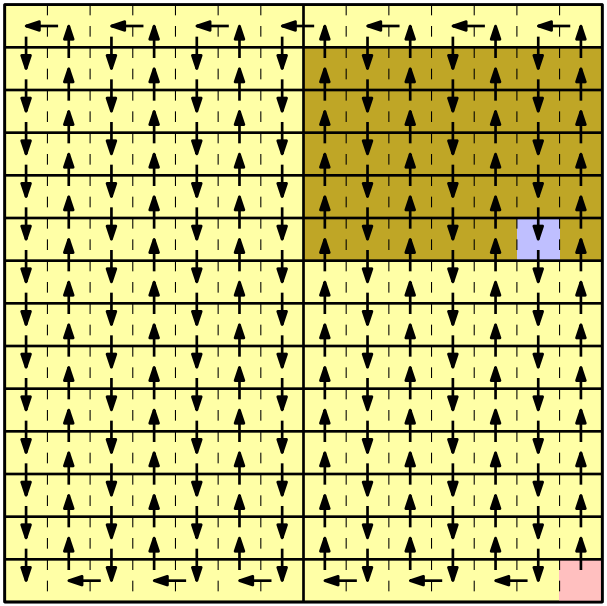
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



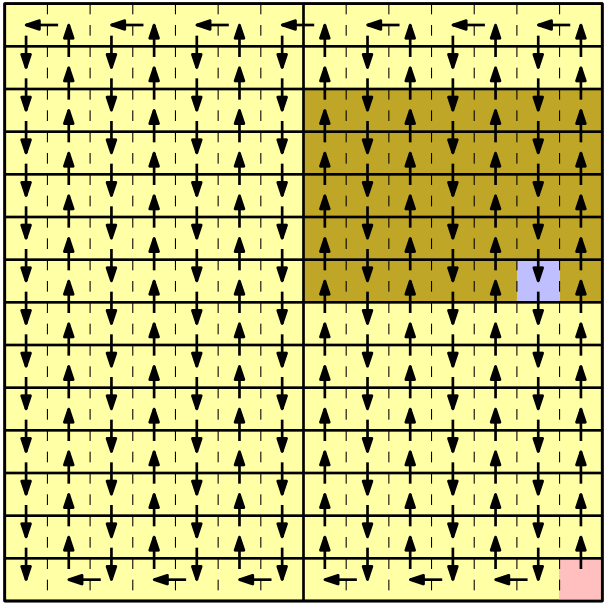
Row-by-row scan

$\Theta(N)$ I/O's in the worst case



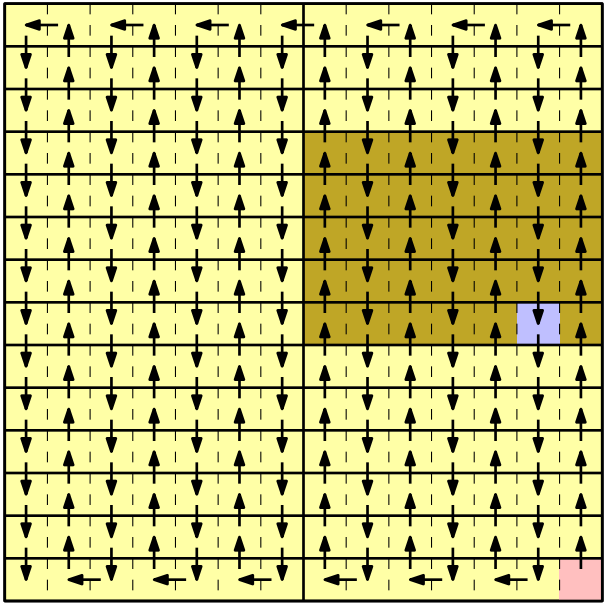
Row-by-row scan

$\Theta(N)$ I/O's in the worst case

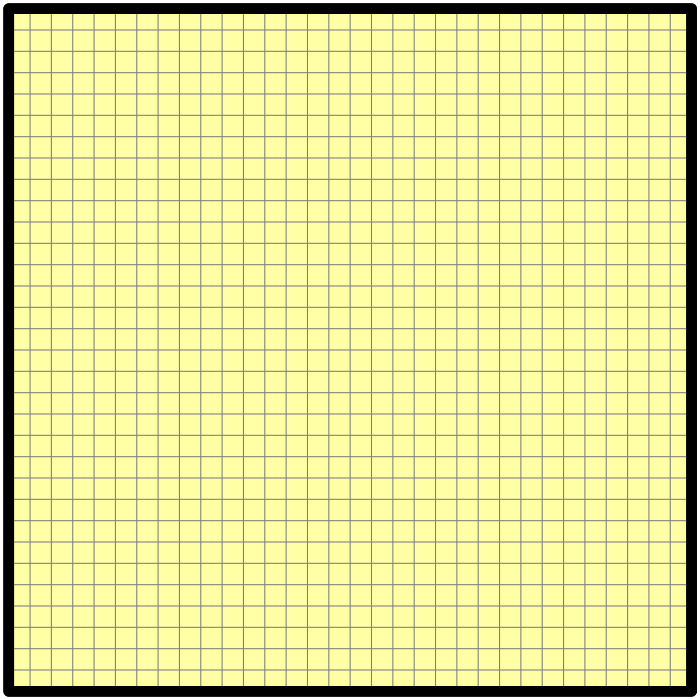


Row-by-row scan

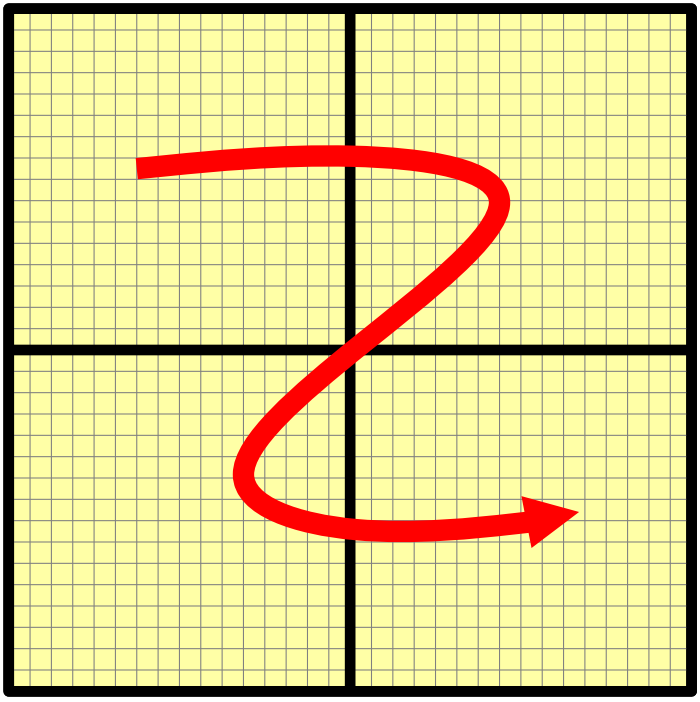
$\Theta(N)$ I/O's in the worst case



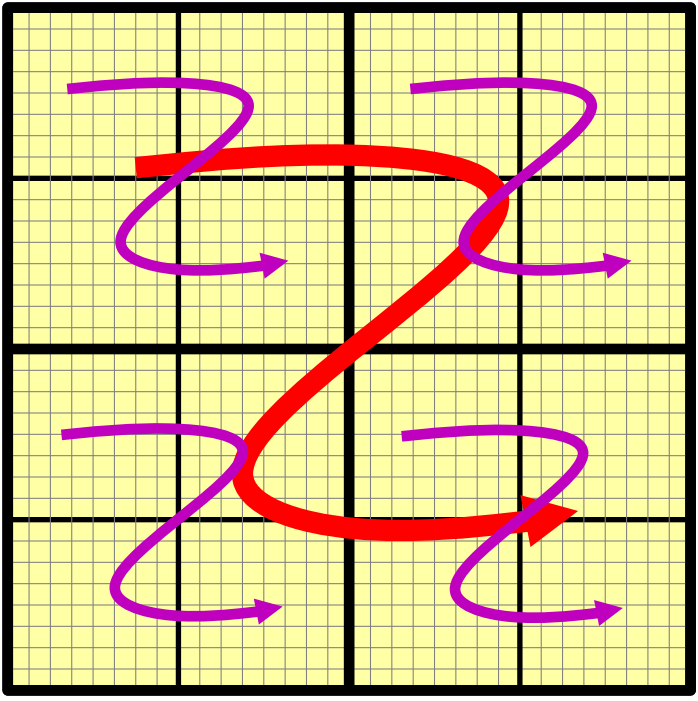
Z-order scan



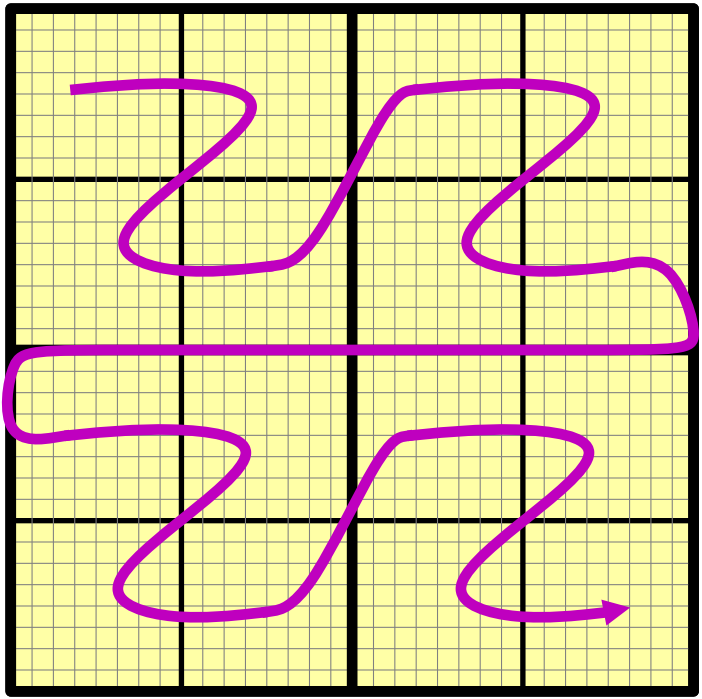
Z-order scan



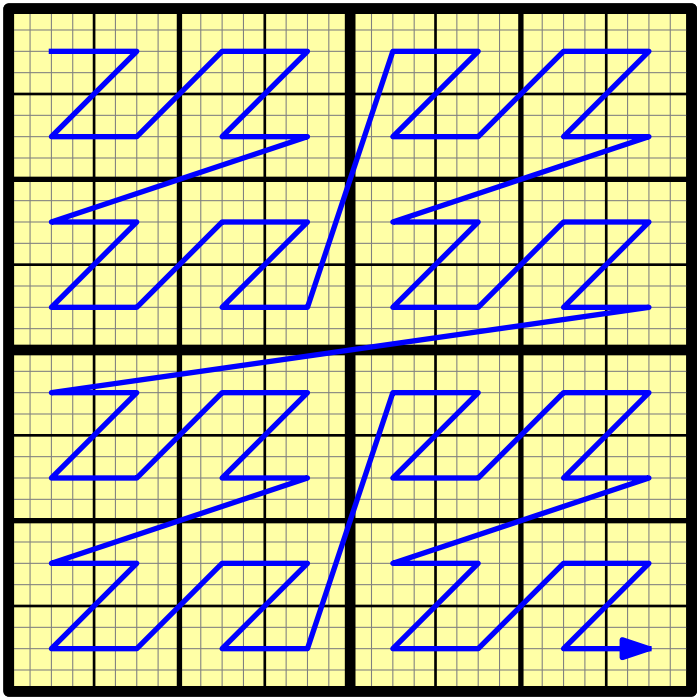
Z-order scan



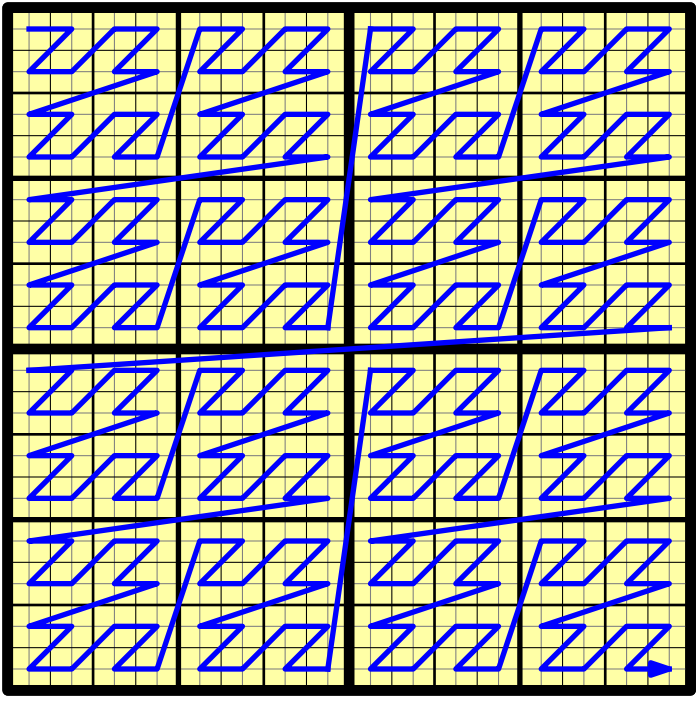
Z-order scan



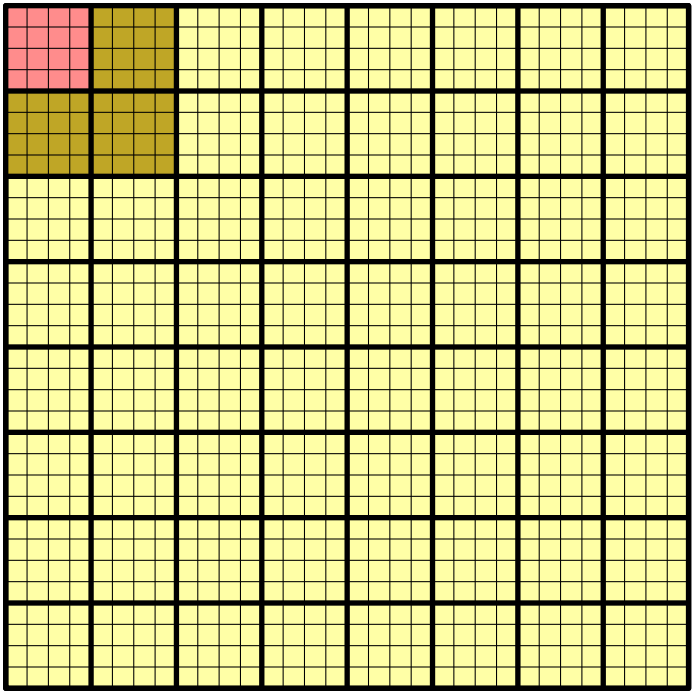
Z-order scan



Z-order scan



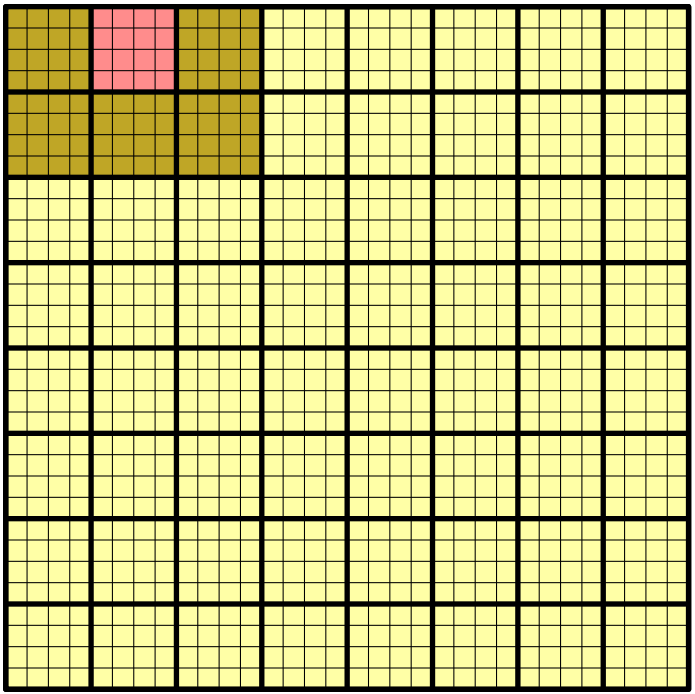
Z-order scan on Z-order file



$\sqrt{B}$

While working on a block, have its neighbours in memory too

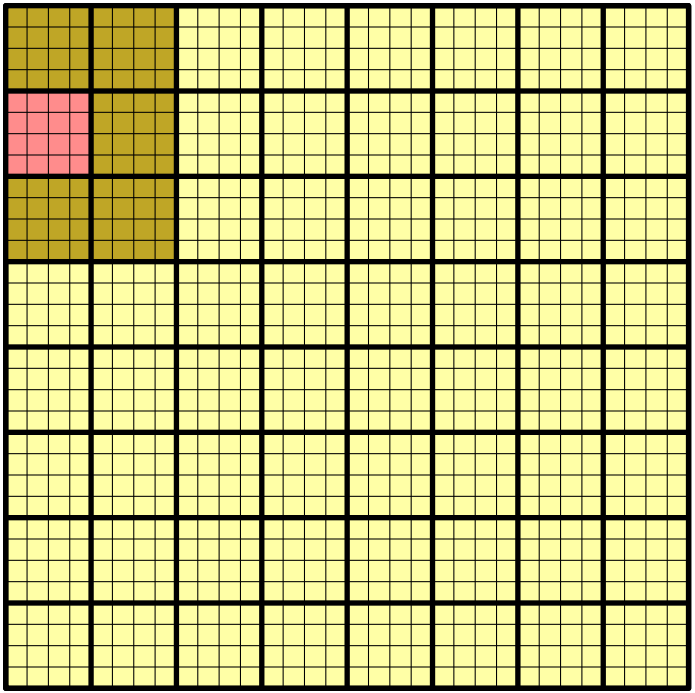
Z-order scan on Z-order file



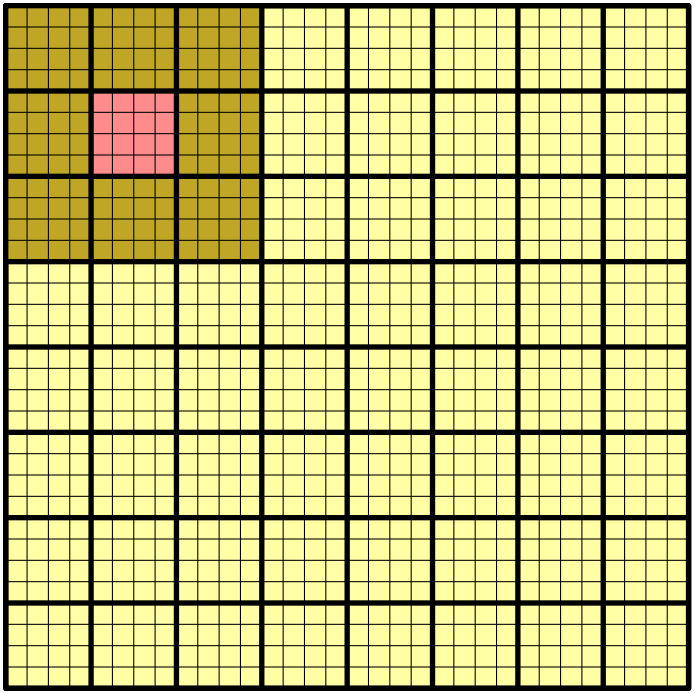
$\sqrt{B}$

While working on a block, have its neighbours in memory too

Z-order scan on Z-order file



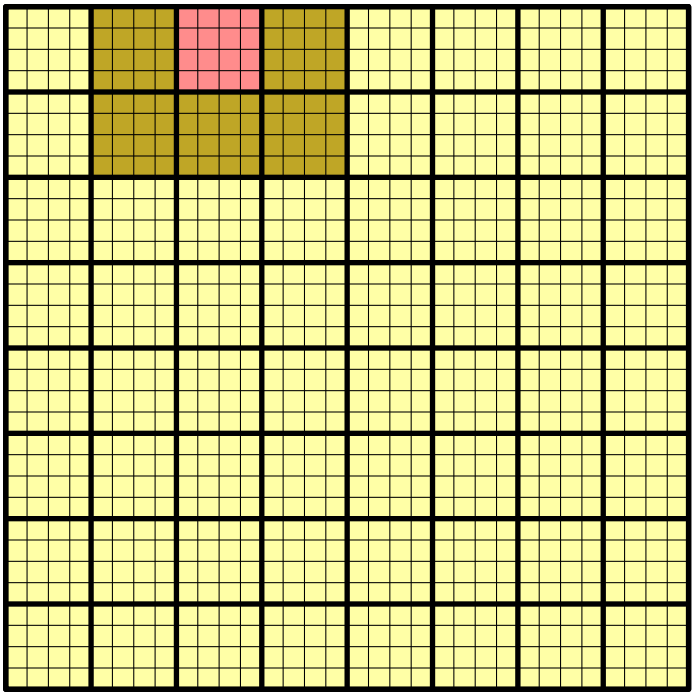
Z-order scan on Z-order file



$\sqrt{B}$

While working on a block, have its neighbours in memory too

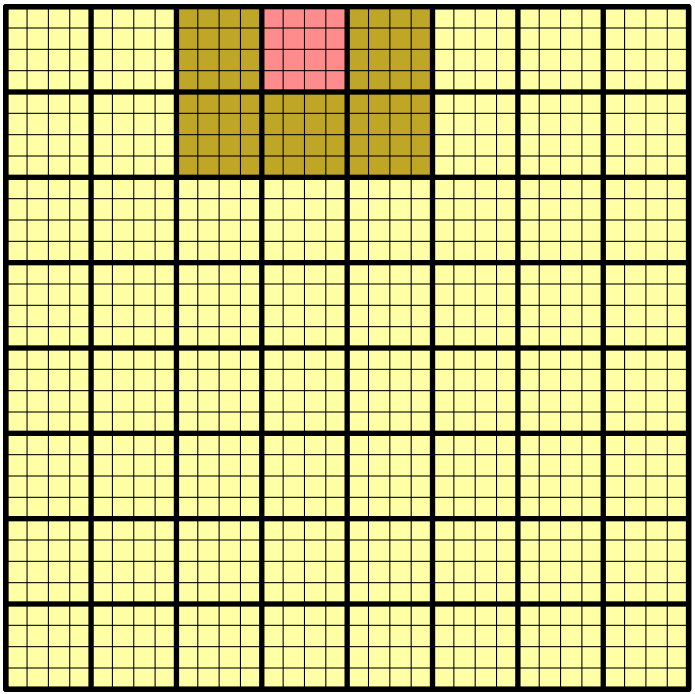
Z-order scan on Z-order file



$\sqrt{B}$

While working on a block, have its neighbours in memory too

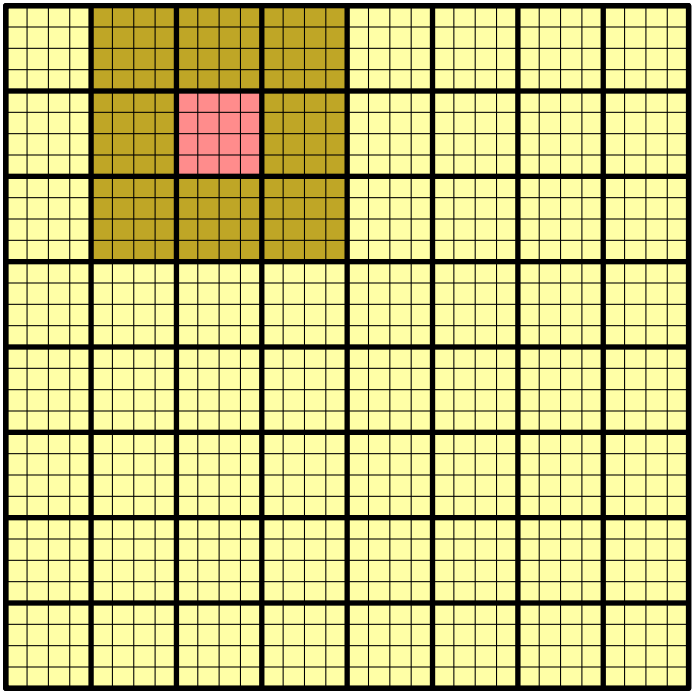
Z-order scan on Z-order file



$\sqrt{B}$

While working on a block, have its neighbours in memory too

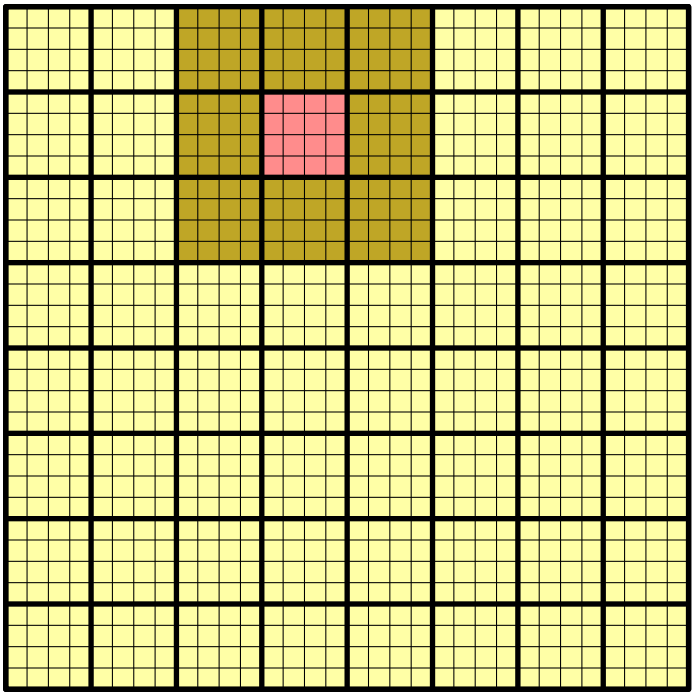
Z-order scan on Z-order file



$\sqrt{B}$

While working on a block, have its neighbours in memory too

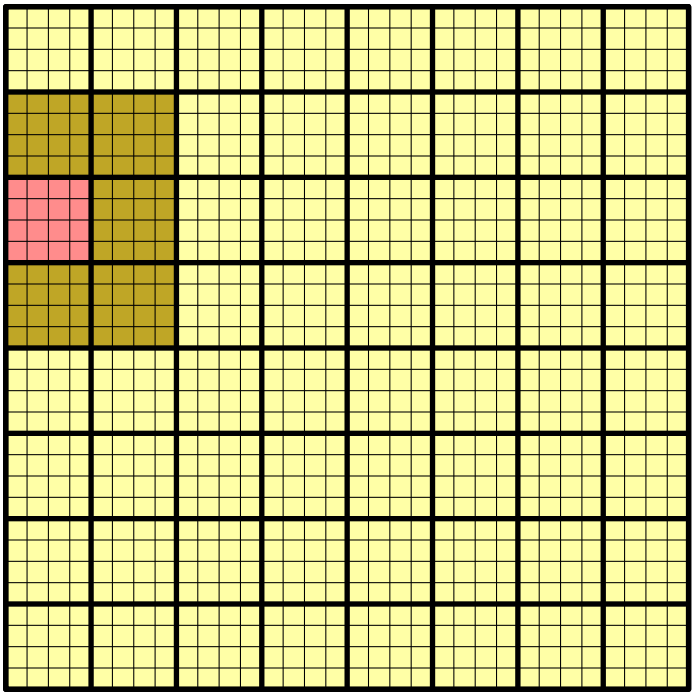
Z-order scan on Z-order file



$\sqrt{B}$

While working on a block, have its neighbours in memory too

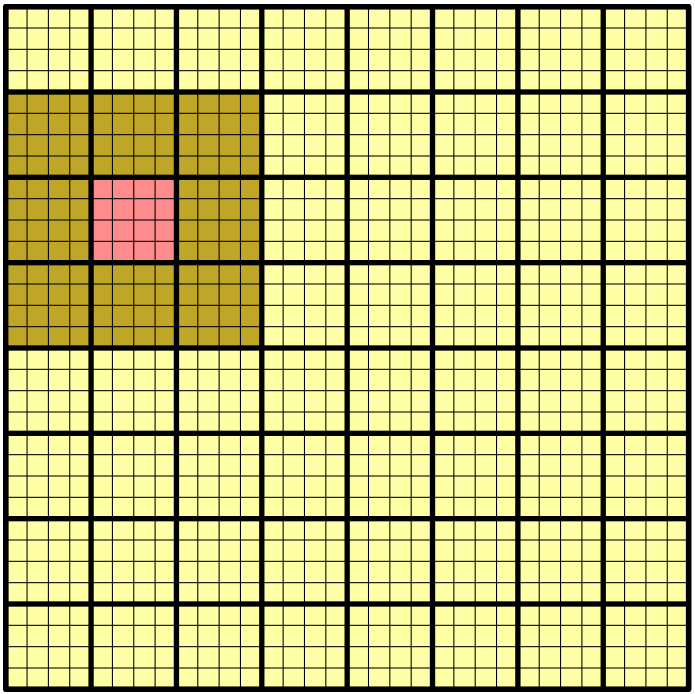
Z-order scan on Z-order file



$\sqrt{B}$

While working on a block, have its neighbours in memory too

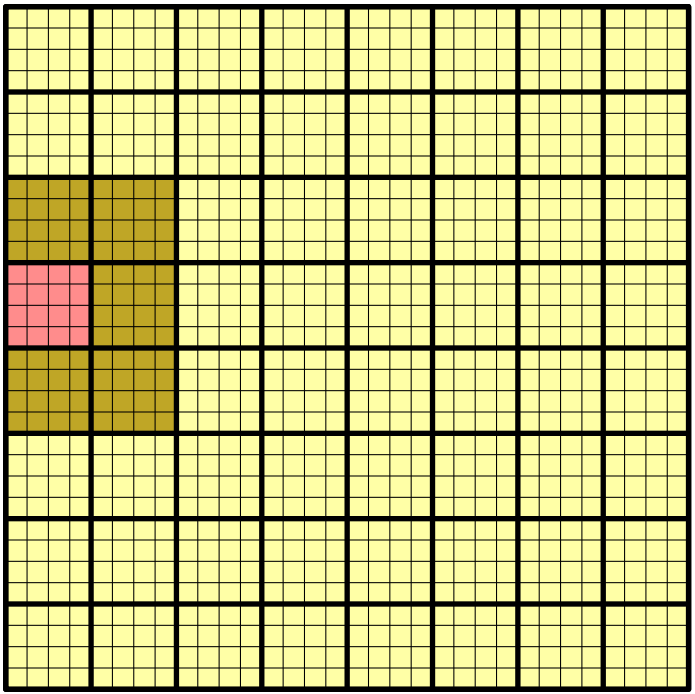
Z-order scan on Z-order file



$\sqrt{B}$

While working on a block, have its neighbours in memory too

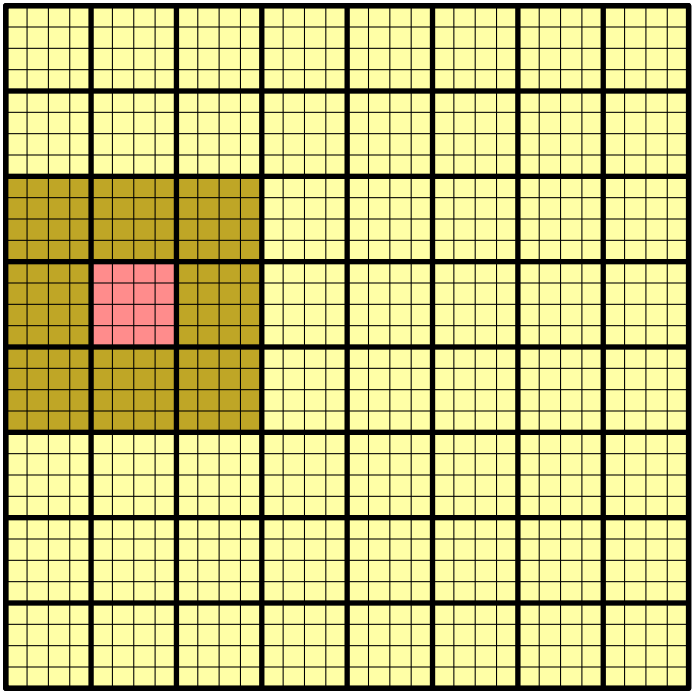
Z-order scan on Z-order file



$\sqrt{B}$

While working on a block, have its neighbours in memory too

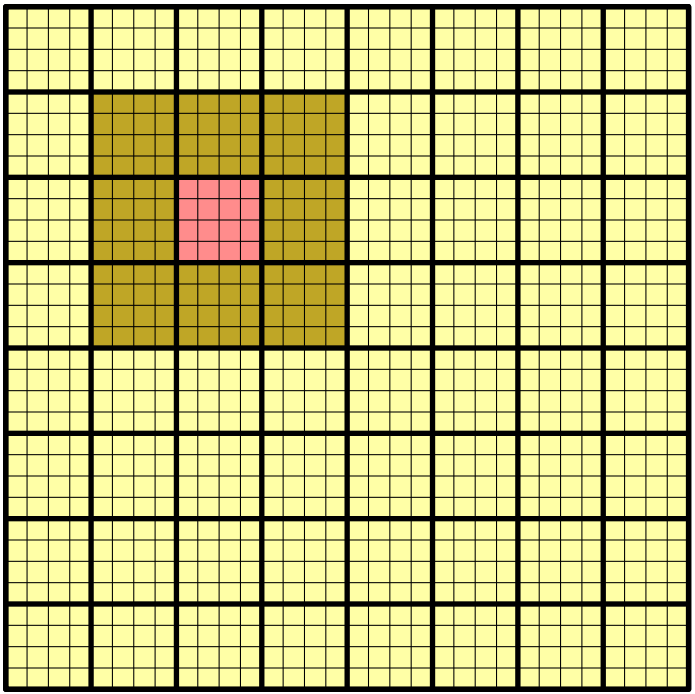
Z-order scan on Z-order file



$\sqrt{B}$

While working on a block, have its neighbours in memory too

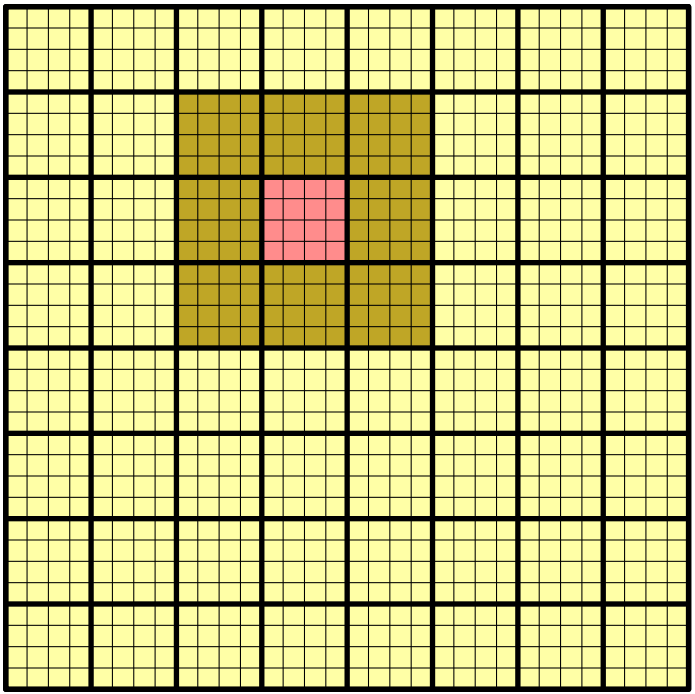
Z-order scan on Z-order file



$\sqrt{B}$

While working on a block, have its neighbours in memory too

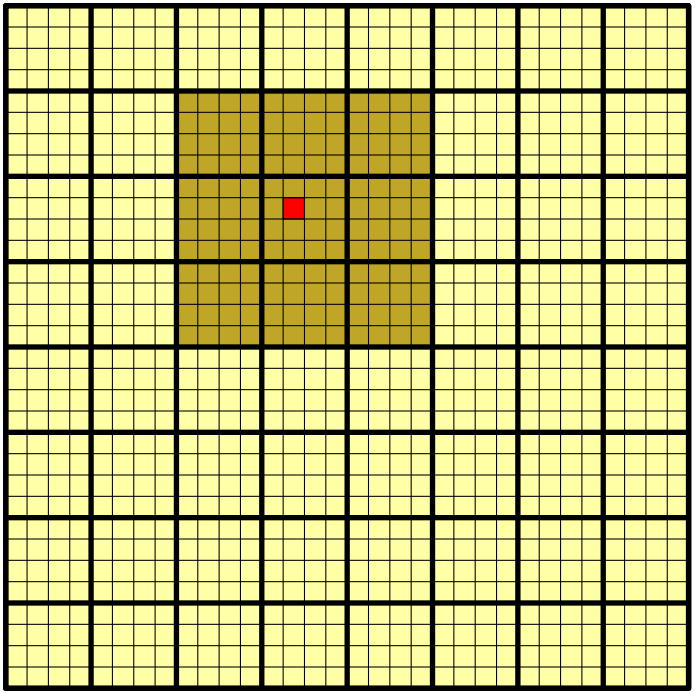
Z-order scan on Z-order file



$\sqrt{B}$

While working on a block, have its neighbours in memory too

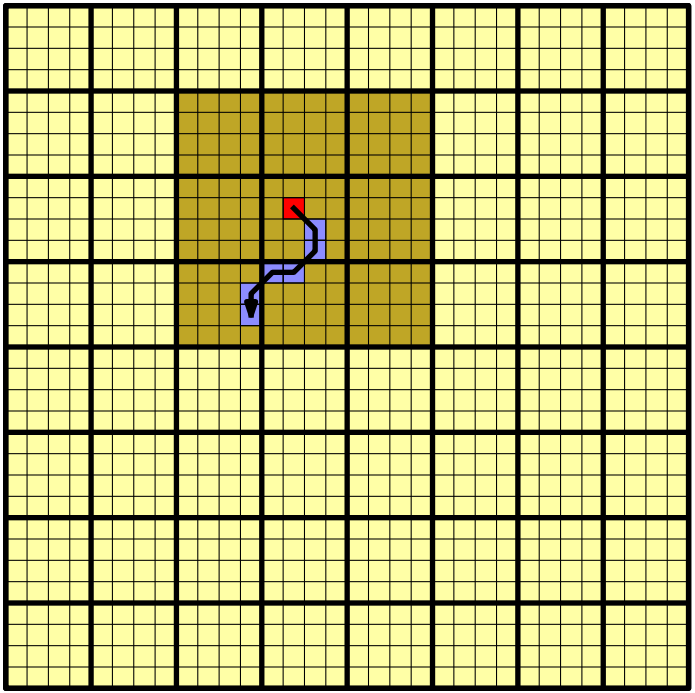
Z-order scan on Z-order file



$\sqrt{B}$

While working on a block, have its neighbours in memory too

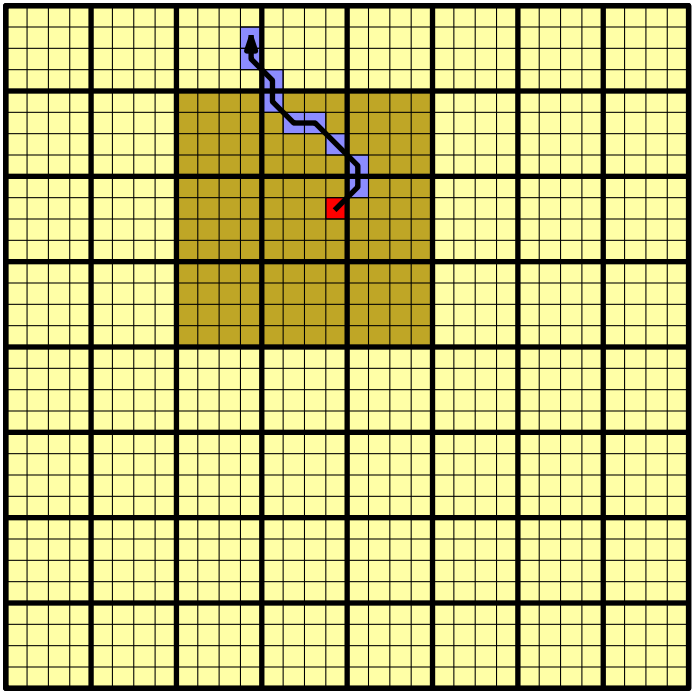
Z-order scan on Z-order file



$\sqrt{B}$

While working on a block, have its neighbours in memory too

Z-order scan on Z-order file



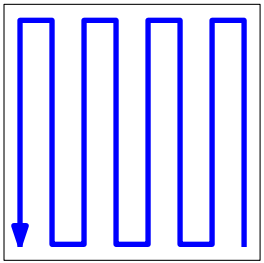
$\sqrt{B}$

While working on a block, have its neighbours in memory too

Only long paths require additional swapping

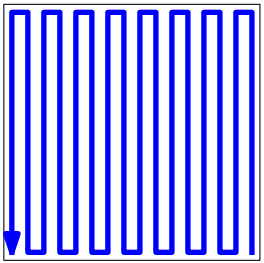
Worst-case terrains vs. real terrains

Worst-case, size $n = 8^2$

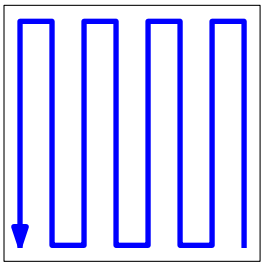


$\Omega(\sqrt{N})$ big turns

Worst-case, size $4n$

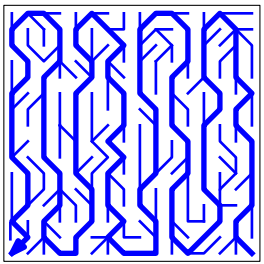


Realistic, size $n = 8^2$



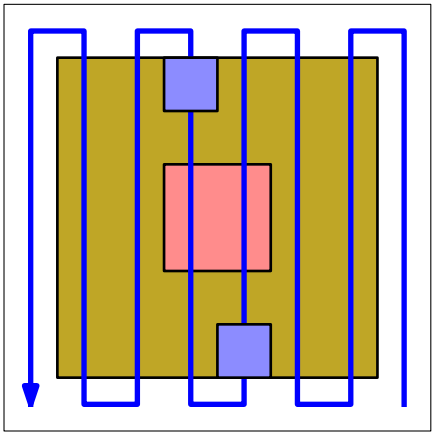
$\Theta(1)$ big turns

Realistic, size $4n$

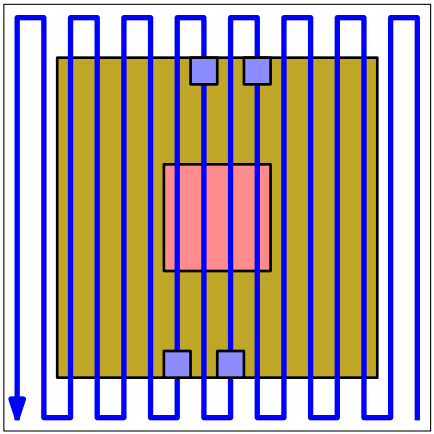


Worst-case terrains vs. real terrains

Worst-case, size $n = 8^2$



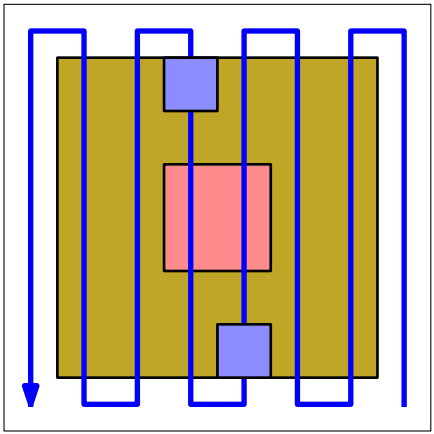
Worst-case, size $4n$



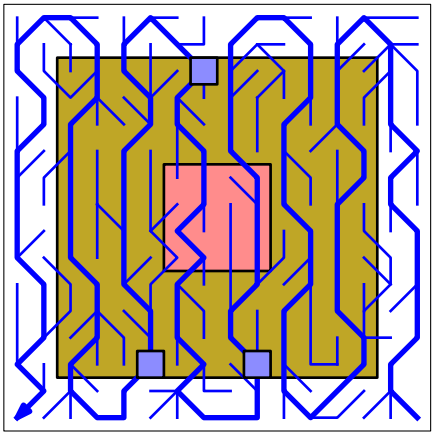
$Q' = Q$ scaled by factor 3.
Far cells of Q: cells on boundary of Q' where water from Q collects.
In the worst case, maximum number of far cells grows with resolution.

Worst-case terrains vs. real terrains

Realistic, size $n = 8^2$

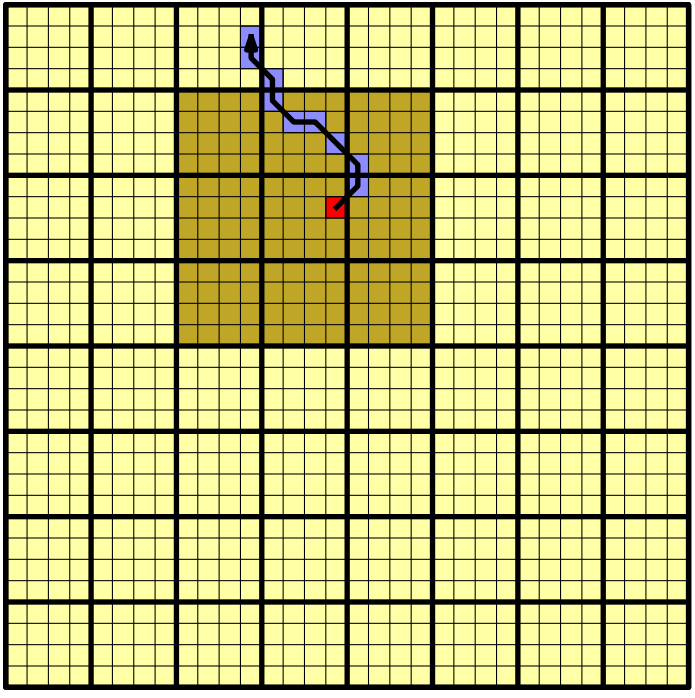


Realistic, size $4n$



$Q' = Q$ scaled by factor 3.
Far cells of Q: cells on boundary of Q' where water from Q collects.
Confluence assumption: number of far cells for any square $Q \leq$ constant γ

Z-order scan on Z-order file



$\sqrt{B}$

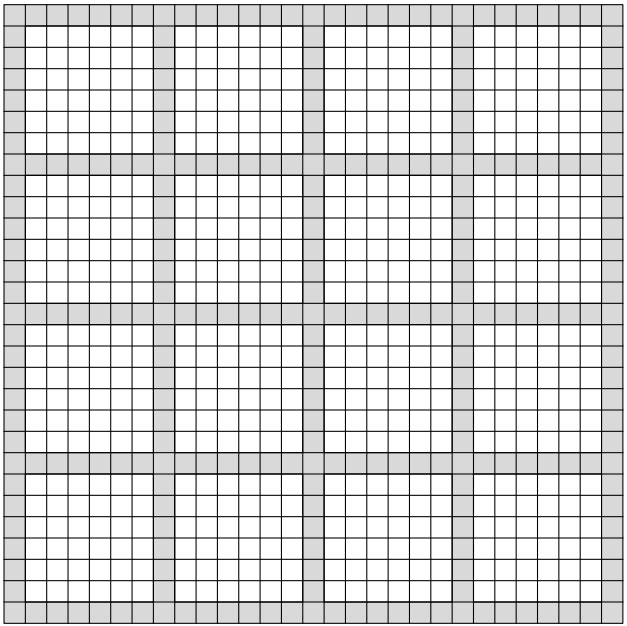
While working on a block, have its neighbours in memory too

$\Theta(\mathrm{scan}(N))$I/Os

Only long paths require additional swapping

N/B blocks $\times$ γ swaps $\times$ 9-block window $=$ $\Theta(\mathrm{scan}(N))$I/Os

Cache-aware separator-based algorithm



Separator cells: each $\Theta(\sqrt{M})$-th row/column

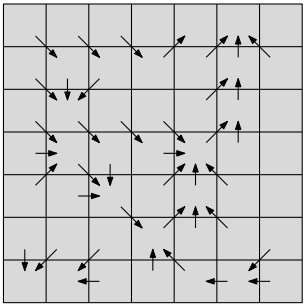
$\rightarrow$

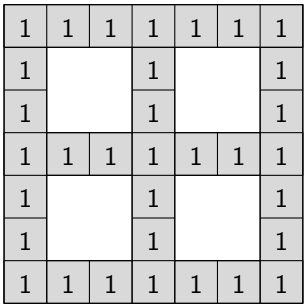
divide grid into $\Theta(N/M)$ subgrids of size $\Theta(M)$

Cache-aware separator-based algorithm

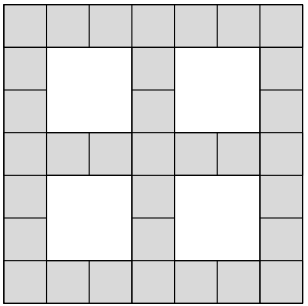
input

1. move flow from interior of subgrids to separators





separator flow accumul.

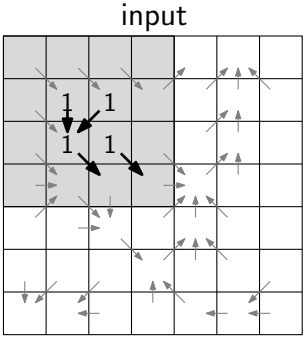


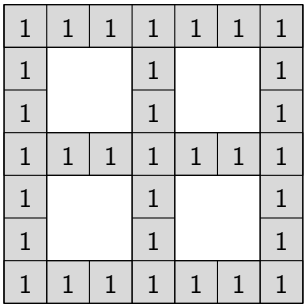
$\Theta(\sqrt{M})$

separator flow directions

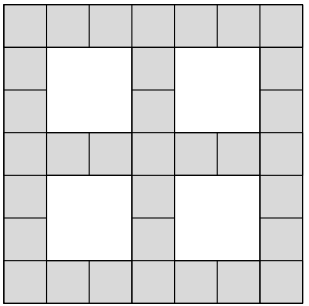
Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators





separator flow accumul.

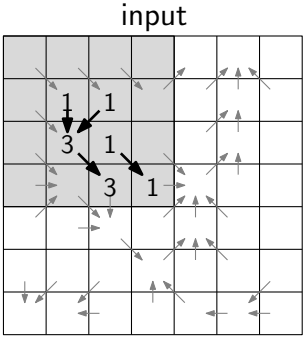


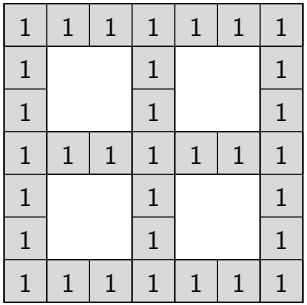
$\Theta(\sqrt{M})$

separator flow directions

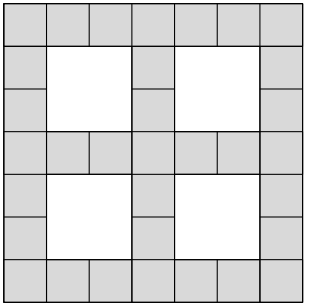
Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators





separator flow accumul.



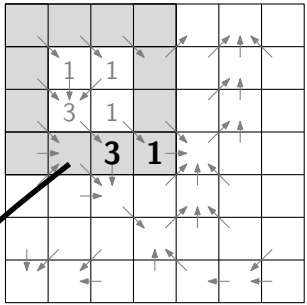
$\Theta(\sqrt{M})$

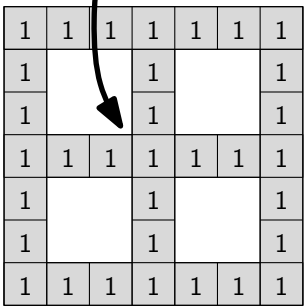
separator flow directions

Cache-aware separator-based algorithm

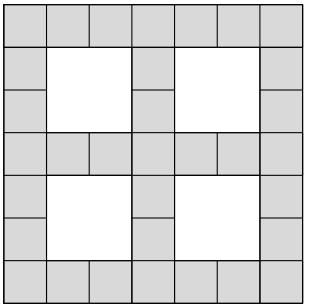
input

1. move flow from interior of subgrids to separators





separator flow accumul.



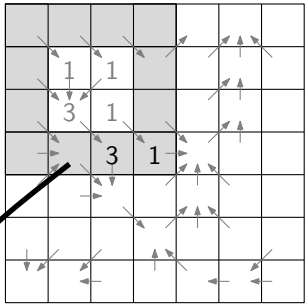
$\Theta(\sqrt{M})$

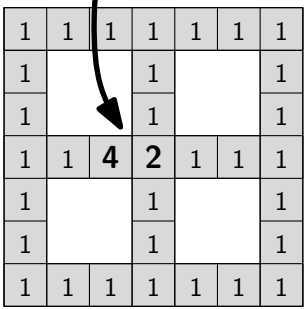
separator flow directions

Cache-aware separator-based algorithm

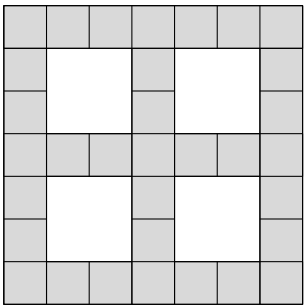
input

1. move flow from interior of subgrids to separators





separator flow accumul.



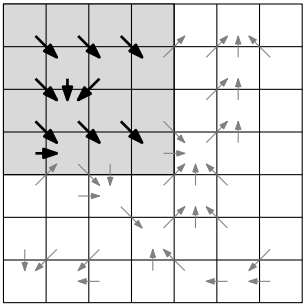
$\Theta(\sqrt{M})$

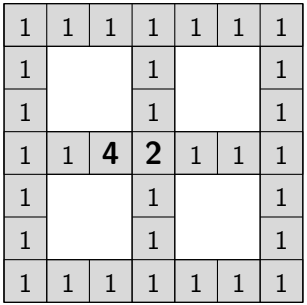
separator flow directions

Cache-aware separator-based algorithm

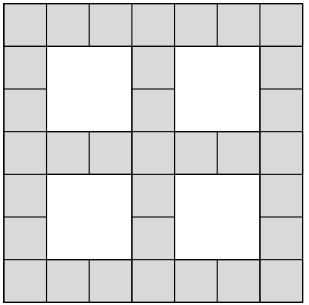
input

1. move flow from interior of subgrids to separators and compute flow connections between separators





separator flow accumul.

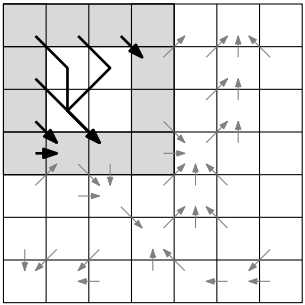


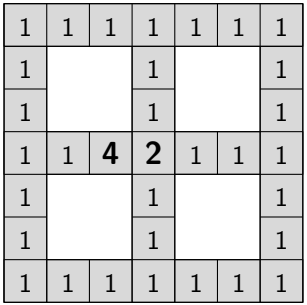
separator flow directions

Cache-aware separator-based algorithm

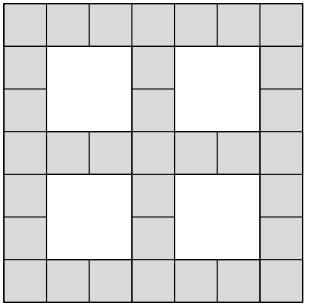
input

1. move flow from interior of subgrids to separators and compute flow connections between separators





separator flow accumul.



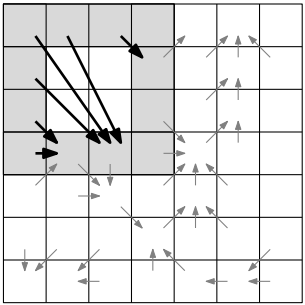
$\Theta(\sqrt{M})$

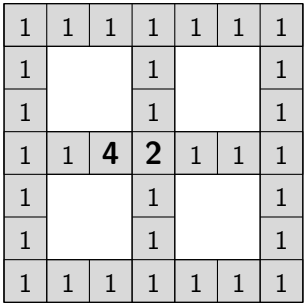
separator flow directions

Cache-aware separator-based algorithm

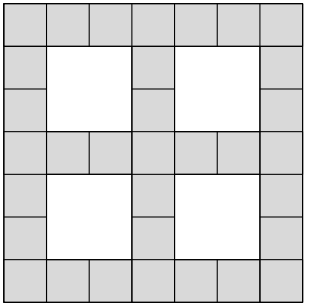
input

1. move flow from interior of subgrids to separators and compute flow connections between separators





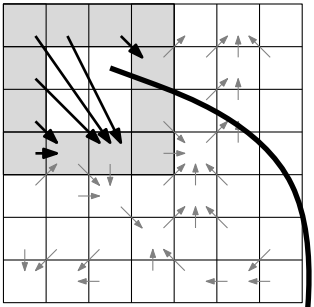
separator flow accumul.

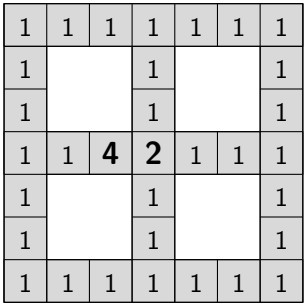


separator flow directions

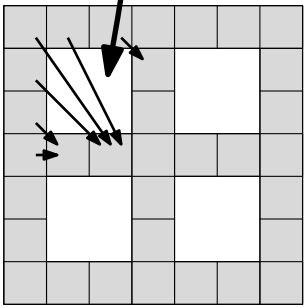
Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators and compute flow connections between separators





separator flow accumul.

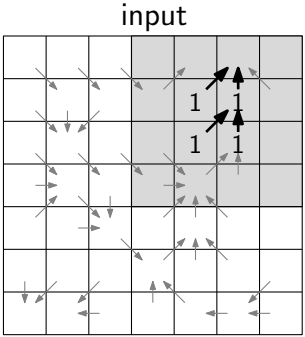


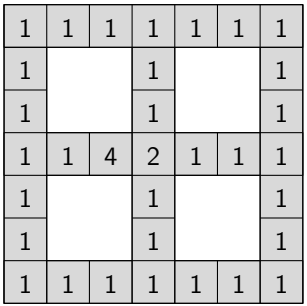
$\Theta(\sqrt{M})$

separator flow directions

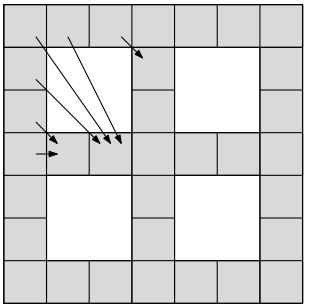
Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators and compute flow connections between separators





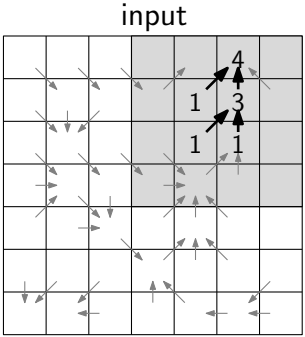
separator flow accumul.

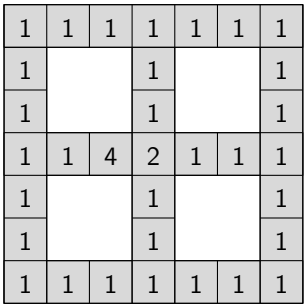


separator flow directions

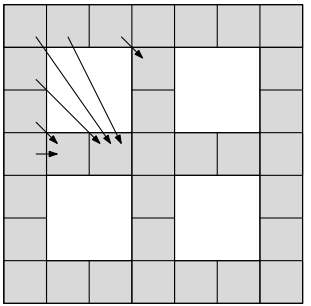
Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators and compute flow connections between separators





separator flow accumul.

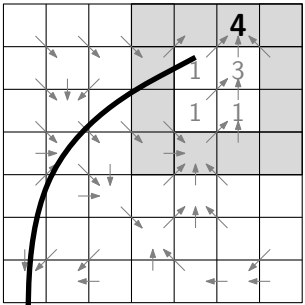


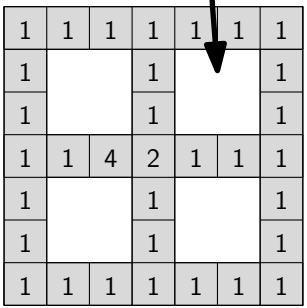
separator flow directions

Cache-aware separator-based algorithm

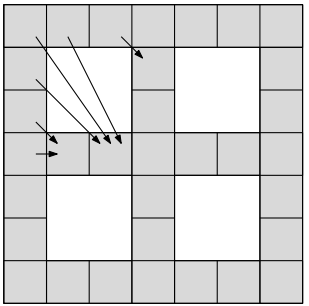
input

1. move flow from interior of subgrids to separators and compute flow connections between separators





separator flow accumul.



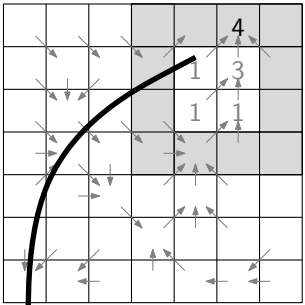
$\Theta(\sqrt{M})$

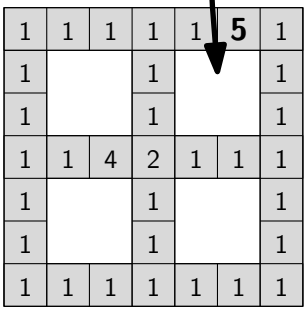
separator flow directions

Cache-aware separator-based algorithm

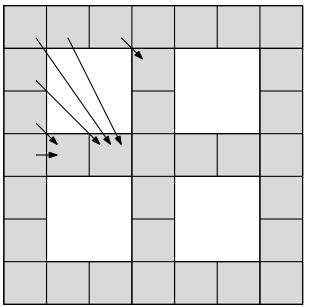
input

1. move flow from interior of subgrids to separators and compute flow connections between separators





separator flow accumul.

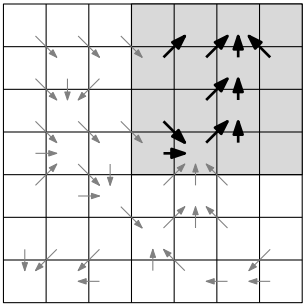


$\Theta(\sqrt{M})$

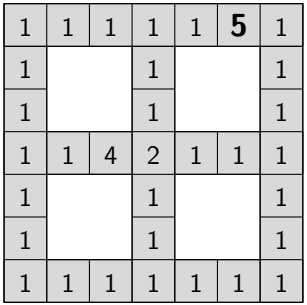
separator flow directions

Cache-aware separator-based algorithm

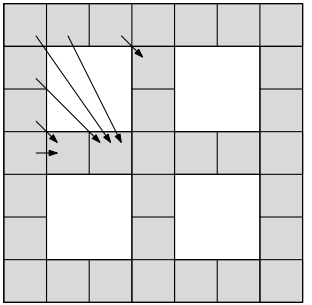
input



1. move flow from interior of subgrids to separators and compute flow connections between separators



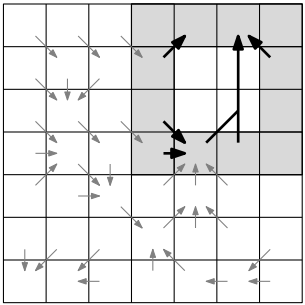
separator flow accumul.



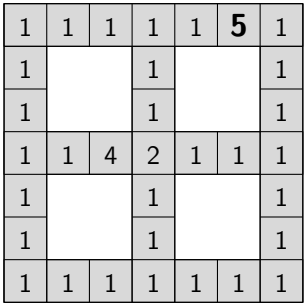
separator flow directions

Cache-aware separator-based algorithm

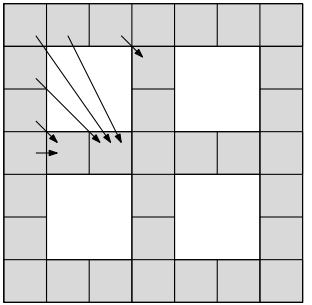
input



1. move flow from interior of subgrids to separators and compute flow connections between separators



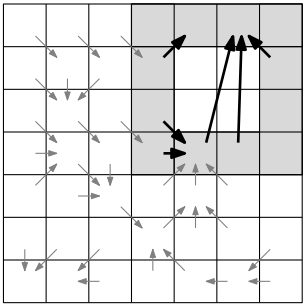
separator flow accumul.



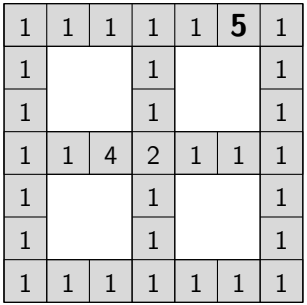
separator flow directions

Cache-aware separator-based algorithm

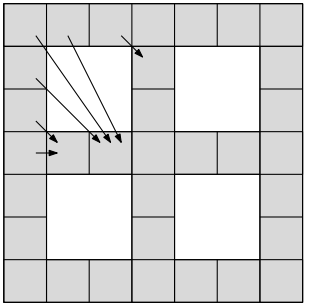
input



1. move flow from interior of subgrids to separators and compute flow connections between separators



separator flow accumul.

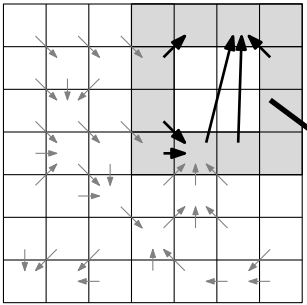


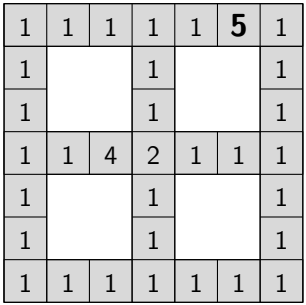
separator flow directions

Cache-aware separator-based algorithm

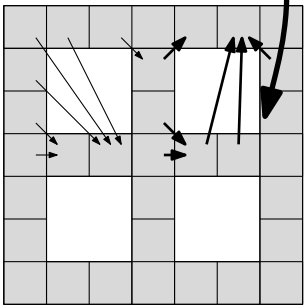
input

1. move flow from interior of subgrids to separators and compute flow connections between separators





separator flow accumul.

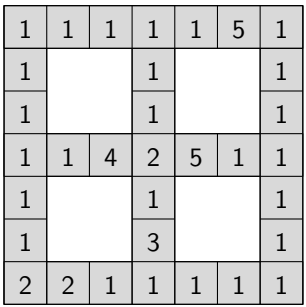


$\Theta(\sqrt{M})$

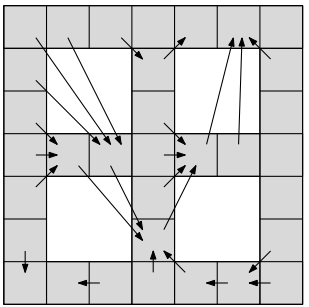
separator flow directions

Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators and compute flow connections between separators



separator flow accumul.

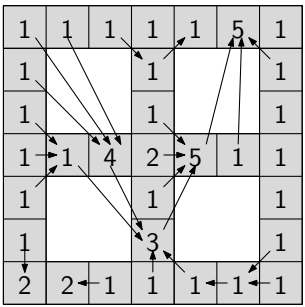


separator flow directions

Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators and compute flow connections between separators

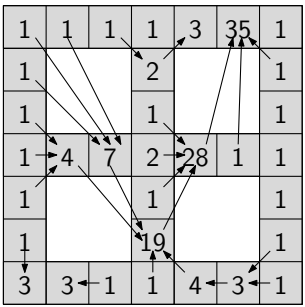
2. compute flow accumulation of separators



Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators and compute flow connections between separators

2. compute flow accumulation of separators



Cache-aware separator-based algorithm

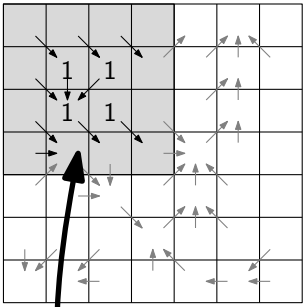
1. move flow from interior of subgrids to separators and compute flow connections between separators

2. compute flow accumulation of separators

1	1	1	1	3	35	1
1			2			1
1			1			1
1	4	7	2	28	1	1
1			1			1
1			19			1
3	3	1	1	4	3	1

Cache-aware separator-based algorithm

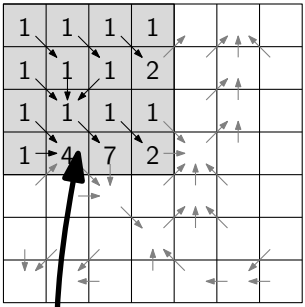
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids

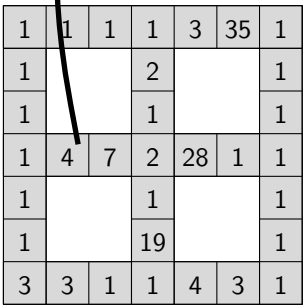


1	1	1	1	3	35	1
1			2			1
1			1			1
1	4	7	2	28	1	1
1			1			1
1			19			1
3	3	1	1	4	3	1

Cache-aware separator-based algorithm

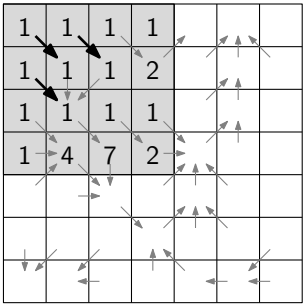
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids





Cache-aware separator-based algorithm

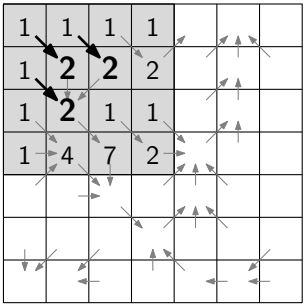
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids



1	1	1	1	3	35	1
1			2			1
1			1			1
1	4	7	2	28	1	1
1			1			1
1			19			1
3	3	1	1	4	3	1

Cache-aware separator-based algorithm

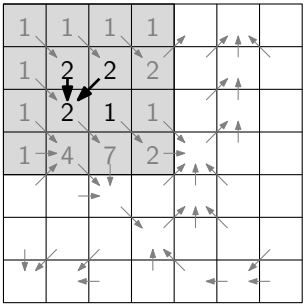
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids



1	1	1	1	3	35	1
1			2			1
1			1			1
1	4	7	2	28	1	1
1			1			1
1			19			1
3	3	1	1	4	3	1

Cache-aware separator-based algorithm

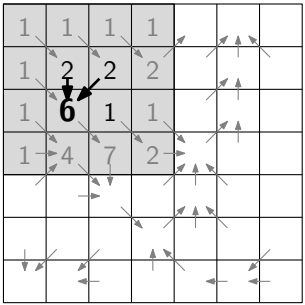
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids



1	1	1	1	3	35	1
1			2			1
1			1			1
1	4	7	2	28	1	1
1			1			1
1			19			1
3	3	1	1	4	3	1

Cache-aware separator-based algorithm

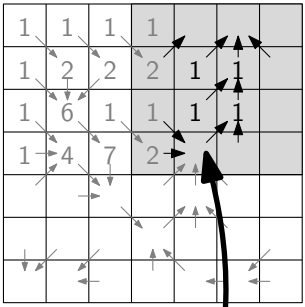
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids



1	1	1	1	3	35	1
1			2			1
1			1			1
1	4	7	2	28	1	1
1			1			1
1			19			1
3	3	1	1	4	3	1

Cache-aware separator-based algorithm

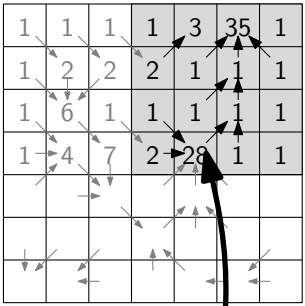
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids

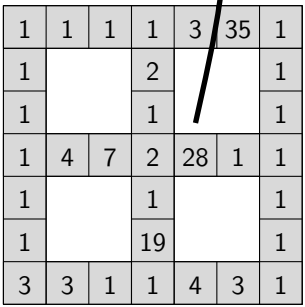


1	1	1	1	3	35	1
1			2			1
1			1			1
1	4	7	2	28	1	1
1			1			1
1			19			1
3	3	1	1	4	3	1

Cache-aware separator-based algorithm

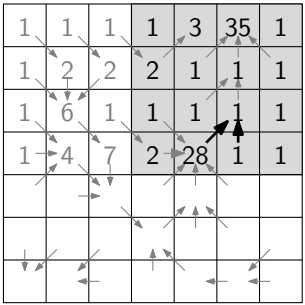
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids





Cache-aware separator-based algorithm

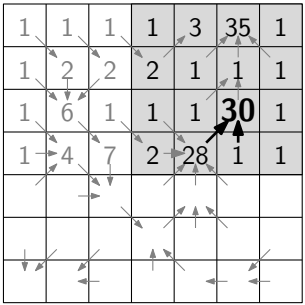
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids



1	1	1	1	3	35	1
1			2			1
1			1			1
1	4	7	2	28	1	1
1			1			1
1			19			1
3	3	1	1	4	3	1

Cache-aware separator-based algorithm

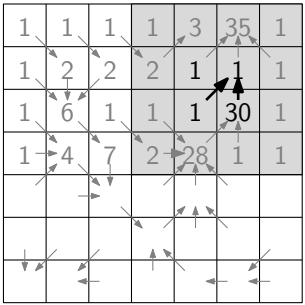
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids



1	1	1	1	3	35	1
1			2			1
1			1			1
1	4	7	2	28	1	1
1			1			1
1			19			1
3	3	1	1	4	3	1

Cache-aware separator-based algorithm

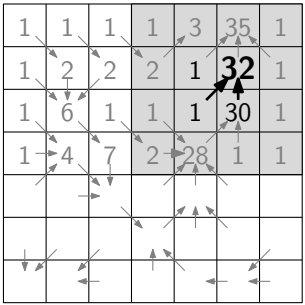
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids



1	1	1	1	3	35	1
1			2			1
1			1			1
1	4	7	2	28	1	1
1			1			1
1			19			1
3	3	1	1	4	3	1

Cache-aware separator-based algorithm

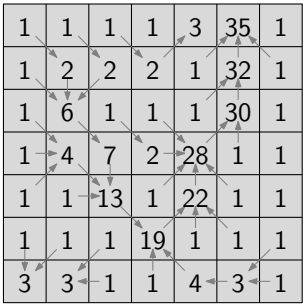
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids

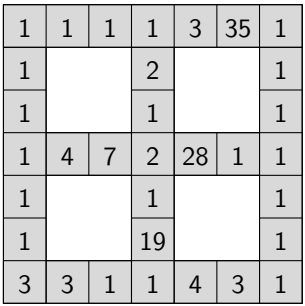


1	1	1	1	3	35	1
1			2			1
1			1			1
1	4	7	2	28	1	1
1			1			1
1			19			1
3	3	1	1	4	3	1

Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids





Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators and compute flow connections between separators	1. $\Theta(N/M)$ subgrids $\times$ $\Theta(M/B + \sqrt{M}) =$ $\Theta(M/B) =$ $\Theta(\mathrm{scan}(N))$ I/O's	1. 1 byte of I/O per cell
2. compute flow accumulation of separators	2. linear-time algo, input $\Theta(N/M) = \Theta(\mathrm{scan}(N))$	2. no I/O in practice
3. move flow from separators into subgrids	3. $\Theta(\mathrm{scan}(N))$ I/O's (like phase 1)	3. 9 bytes of I/O per cell
		Total: 10 bytes per cell if grid stored in Z-order
		Total: 20 to 60 bytes if grid stored row by row (for $1/4 \leq M/B^2 \leq 4$)

1	1	1	1	3	35	1
1			2			1
1			1			1
1	4	7	2	28	1	1
1			1			1
1			19			1
3	3	1	1	4	3	1

Results on flow accumulation

algorithm	file order	worst case	'realistic'	bytes per cell	time (mins)
row-by-row scan	row by row	$O(N)$	$O(N/\sqrt{B})$	tenthousands	111
Z-order scan	row by row	$O(N)$	$O(\mathrm{scan}(N))^*$	thousands	
Z-order scan	Z-order	$O(N/\sqrt{B})$	$O(\mathrm{scan}(N))$	hundreds	41
$-aware separ.	row by row	$O(\mathrm{scan}(N))^*$		20 to 60	39
$-aware separ.	Z-order	$O(\mathrm{scan}(N))$		10	run this!
$-obliv. separ.	row by row	$O(\mathrm{scan}(N))^*$		>100	
$-obliv. separ.	Z-order	$O(\mathrm{scan}(N))^*$		>100	118
time-fwd proc.	any	$O(\mathrm{sort}(N))$		70 to 300	sev. hundred
row $\leftrightarrow$ Z-order		$O(\mathrm{scan}(N))^*$		16	88

I/O-volume: $N = 2^{32}$, $M = 1$ GB, $B = 16$ to 64KB

time: 3 GHz Pentium, one disk for data + scratch, $N = 3.5 \cdot 10^9$ (Neuse), $M = 1$ GB

*) needs tall cache

Further research

- separator-based flooding works well too (146 minutes), but do not know how to do controlled/partial flooding
- grid techniques also seem applicable to Pfafstetter watershed labelling
- how about flow routing?
- how about multiple flow direction models? (time-forward processing can do that, can we?)
- does the confluence assumption make sense?
  - is γ indeed constant for realistic terrains under increasing resolution?
  - what are typical values for γ?
  - can we come up with an algorithm to compute γ for a given terrain?
  - can we refine the analysis of the scanning algorithms to explain their good performance?