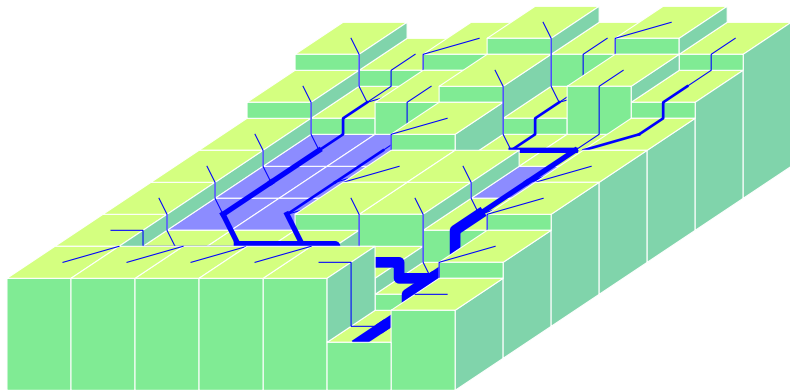


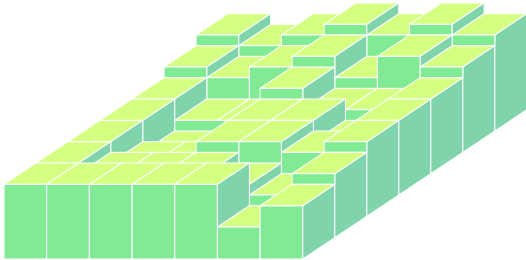
Simple I/O-efficient flow accumulation on grid terrains



Herman Haverkort
TU Eindhoven

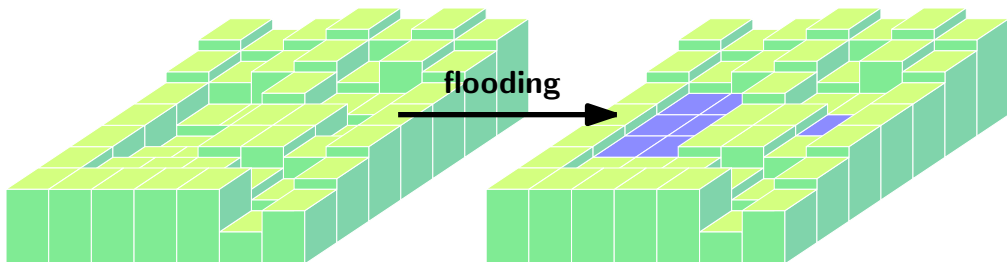
Jeffrey Janssen
Realworld Systems

Drainage network analysis



grid elevation model

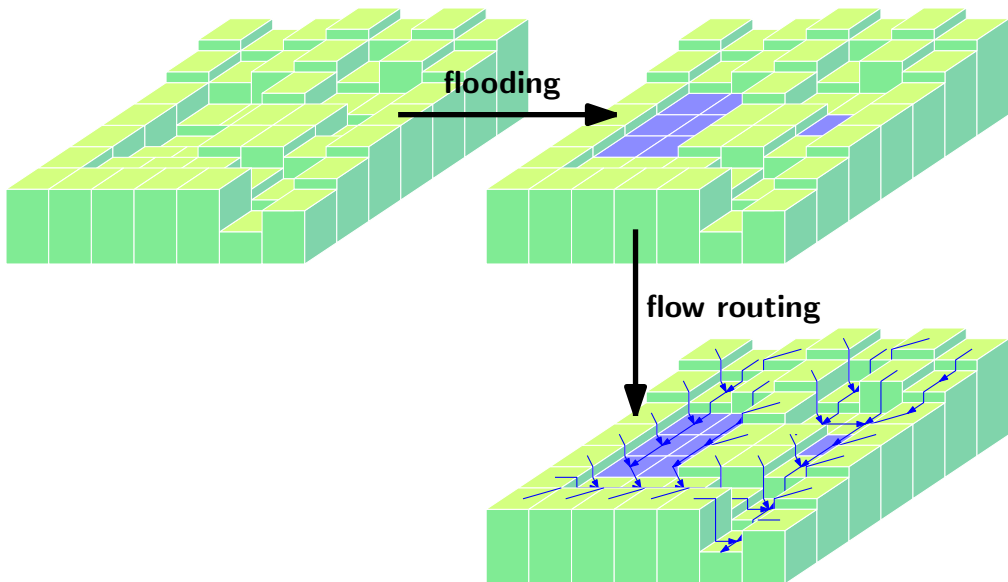
Drainage network analysis



grid elevation model

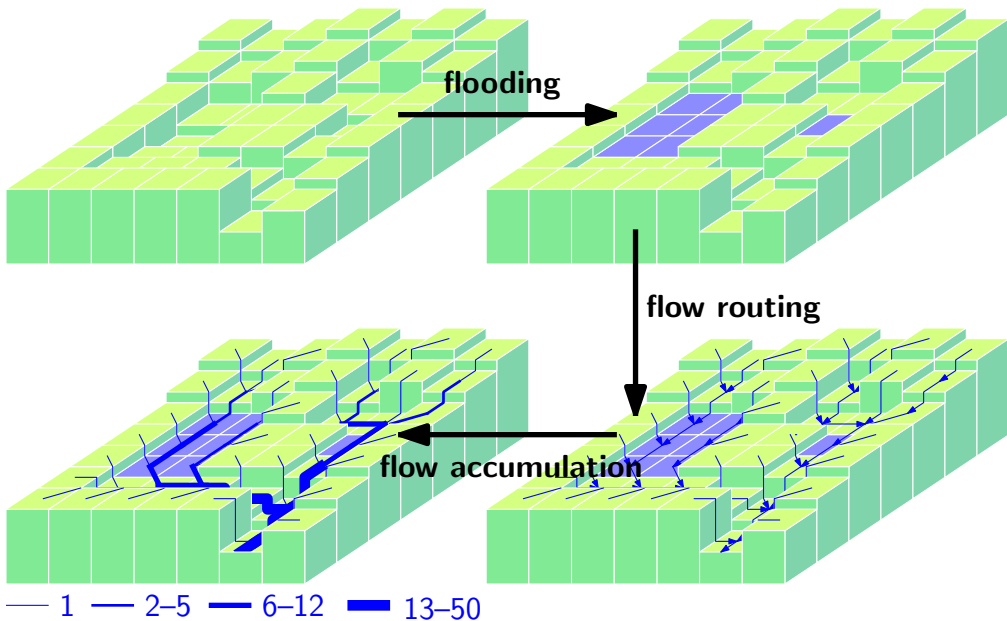
each cell has non-ascending
path to boundary

Drainage network analysis



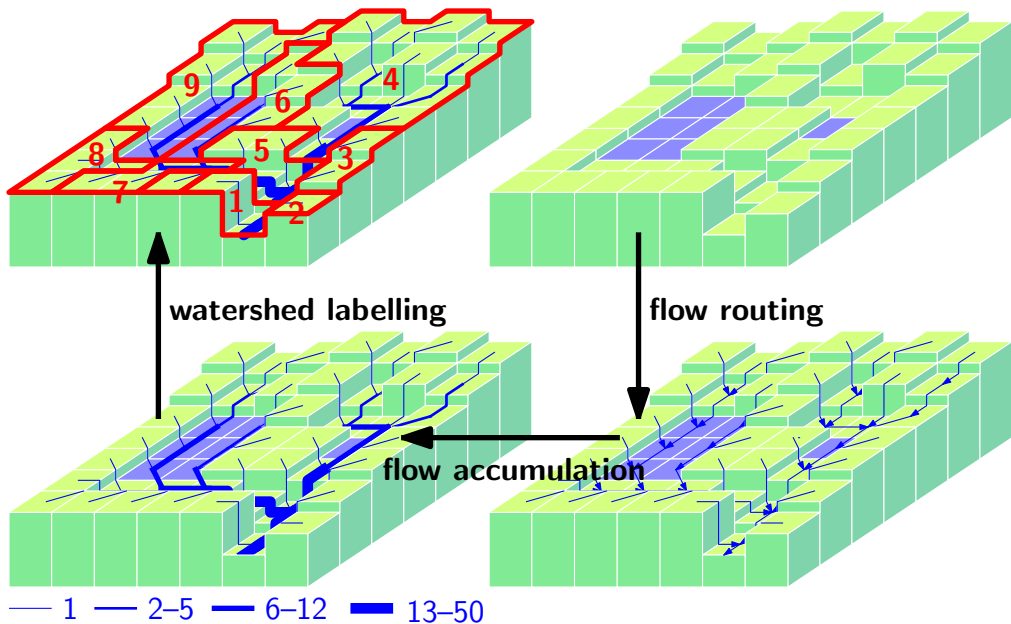
each cell has a flow direction;
following flow directions leads
to boundary

Drainage network analysis

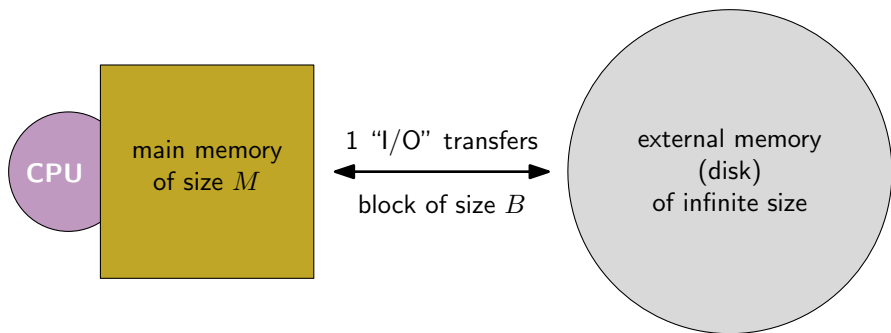


compute for each cell c , from how many cells water passes through c

Drainage network analysis



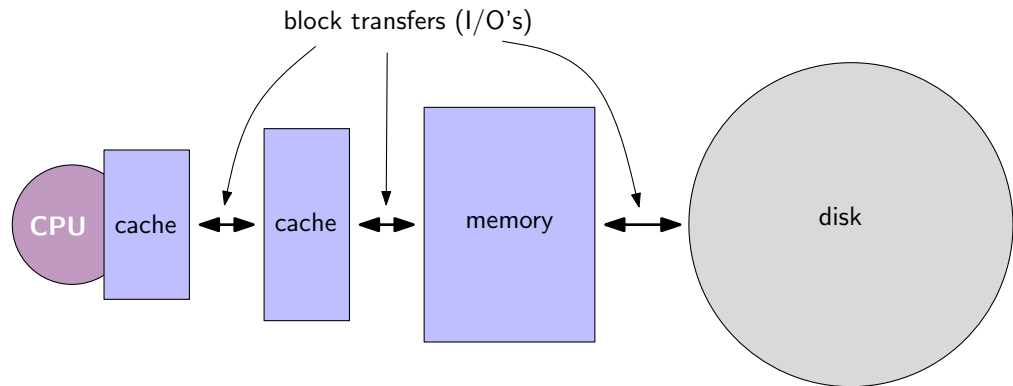
Analysing I/O-efficiency



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 $M \geq c \cdot B^2$)

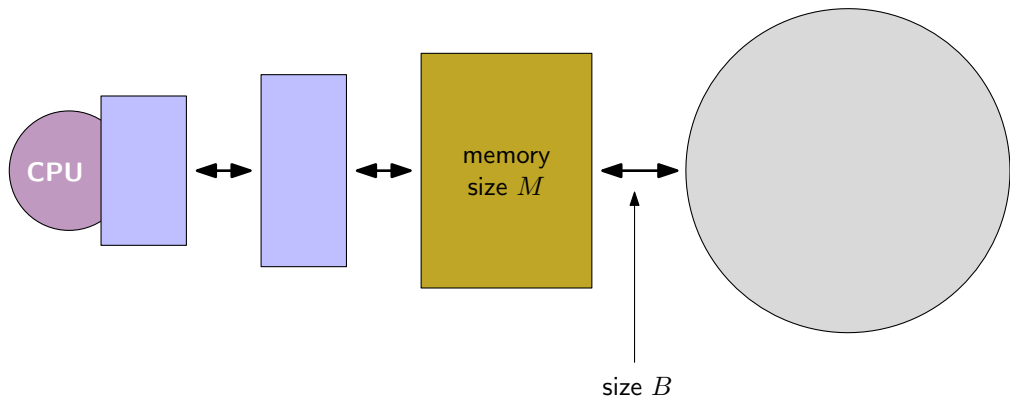
Analysing I/O-efficiency



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 $M \geq c \cdot B^2$)

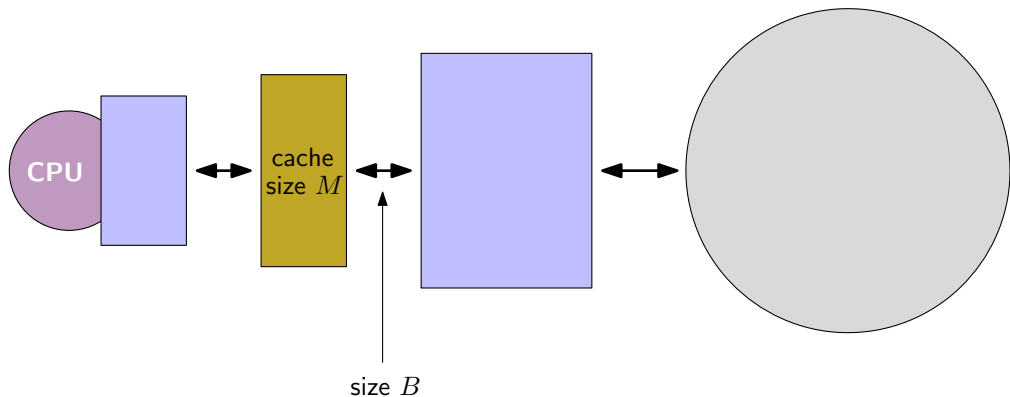
Analysing I/O-efficiency



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 $M \geq c \cdot B^2$)

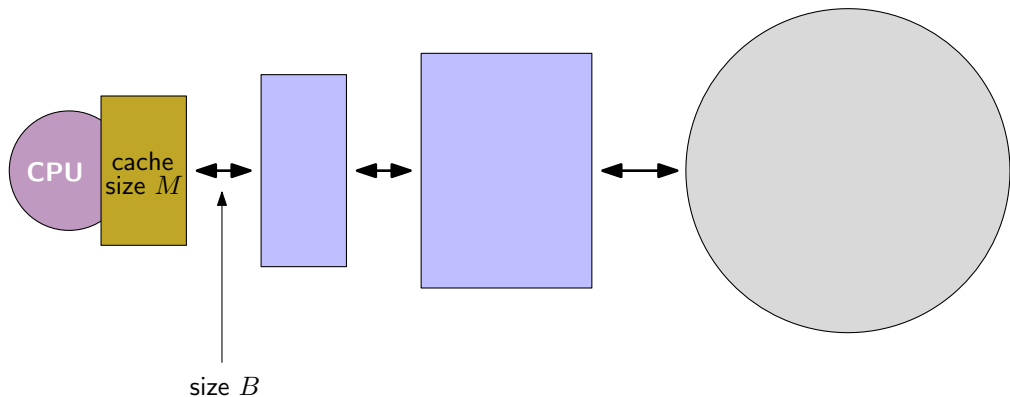
Analysing I/O-efficiency



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



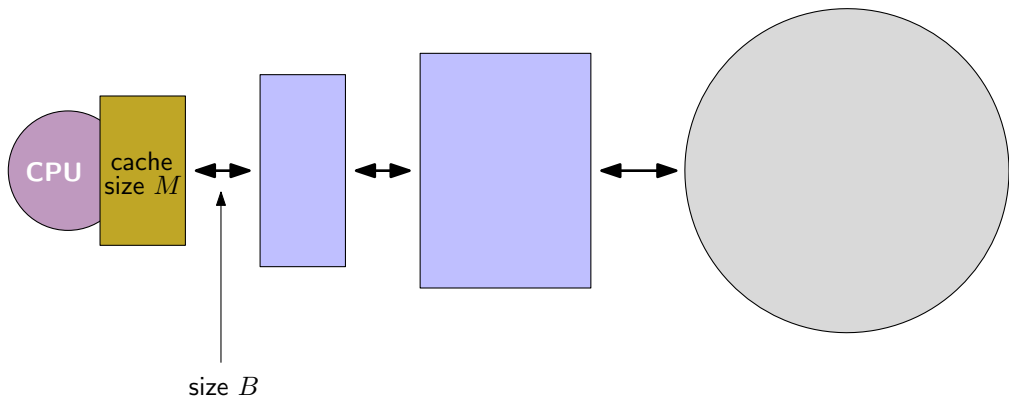
CPU only operates on data in **main memory** (for free)

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



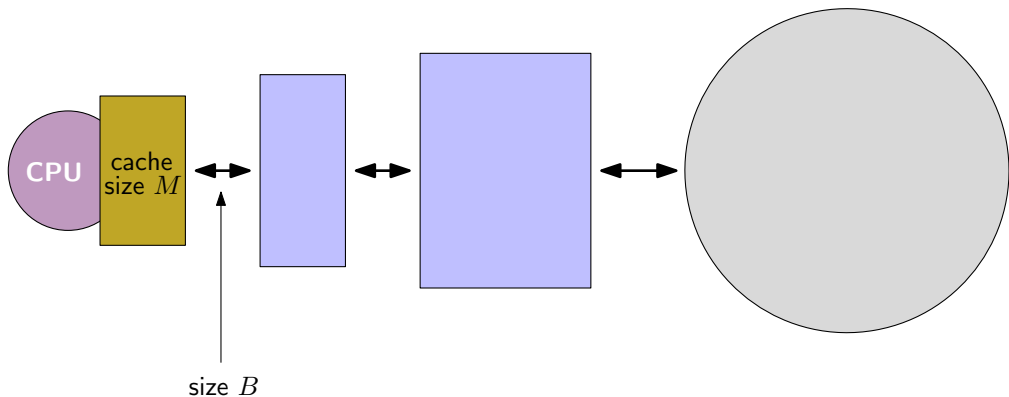
CPU only operates on data in **main memory** (for free)

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 $\Theta(\text{scan}(N)) = \Theta(N/B)$ I/O's

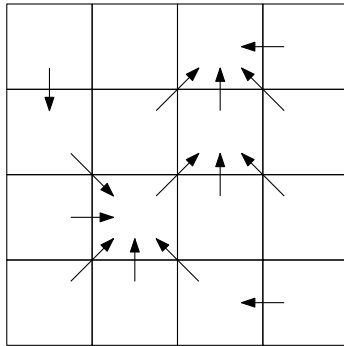
sorting N cells takes $\Theta(\text{sort}(N)) = \Theta(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/O's

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

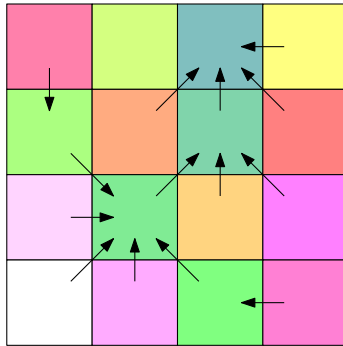


Time-forward processing (Arge et al.)

Goal: compute flow accumulation for each cell c

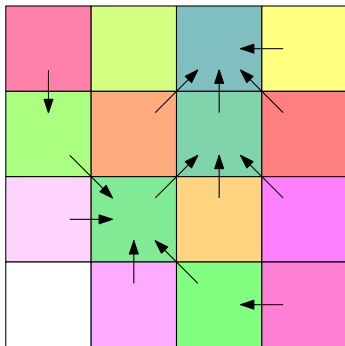
= #cells from which water passes through c

= size of tree rooted at c

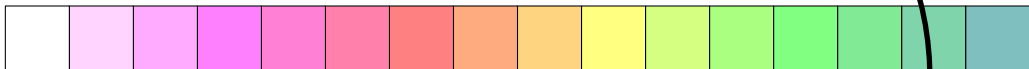


Time-forward processing (Arge et al.)

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



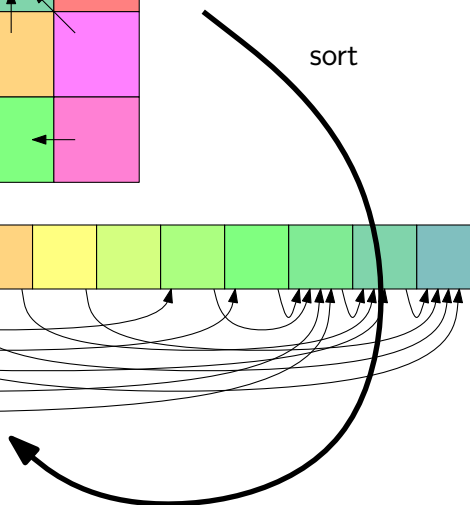
output



input

pqueue

sort

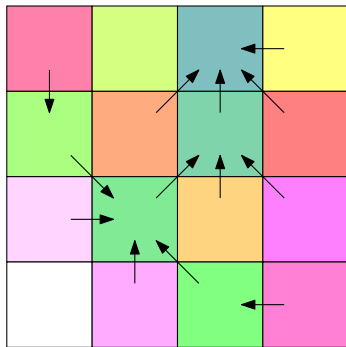


Time-forward processing (Arge et al.)

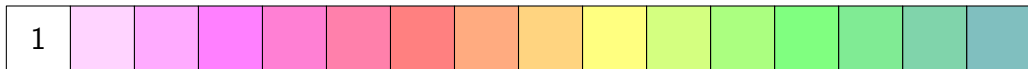
Goal: compute flow accumulation for each cell c

= #cells from which water passes through c

= size of tree rooted at c



output



input

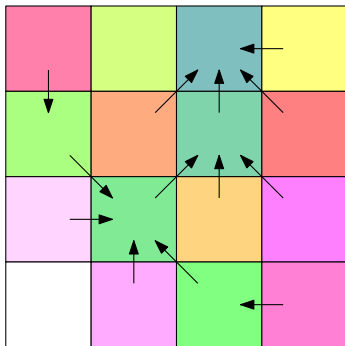
pqueue

Time-forward processing (Arge et al.)

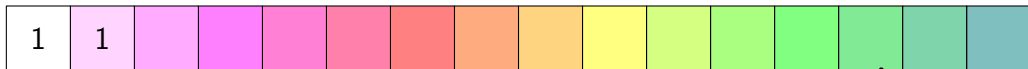
Goal: compute flow accumulation for each cell c

= #cells from which water passes through c

= size of tree rooted at c



output



input

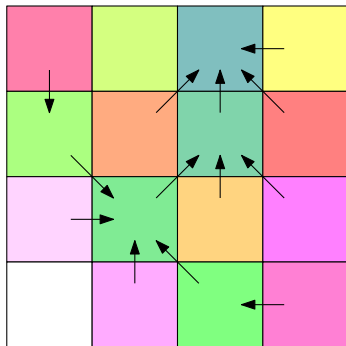
pqueue

Time-forward processing (Arge et al.)

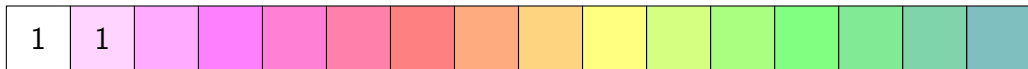
Goal: compute flow accumulation for each cell c

= #cells from which water passes through c

= size of tree rooted at c



output



input

queue

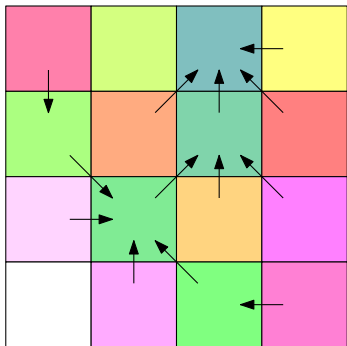
1

Time-forward processing (Arge et al.)

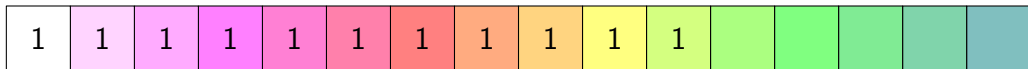
Goal: compute flow accumulation for each cell c

= #cells from which water passes through c

= size of tree rooted at c

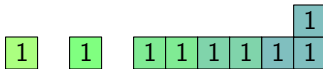


output



input

queue

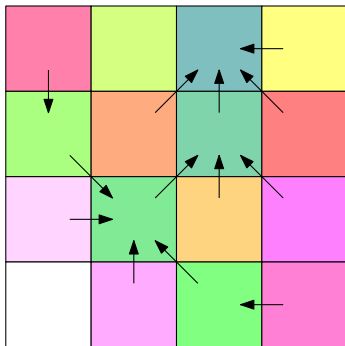


Time-forward processing (Arge et al.)

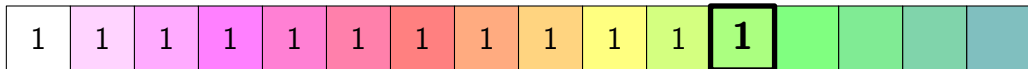
Goal: compute flow accumulation for each cell c

= #cells from which water passes through c

= size of tree rooted at c

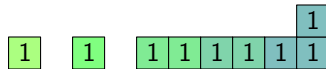


output



input

pqueue

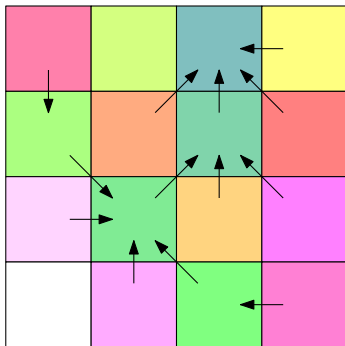


Time-forward processing (Arge et al.)

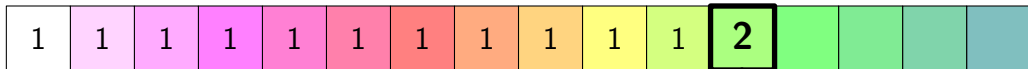
Goal: compute flow accumulation for each cell c

= #cells from which water passes through c

= size of tree rooted at c



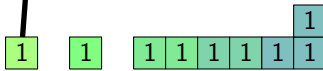
output



input



pqueue

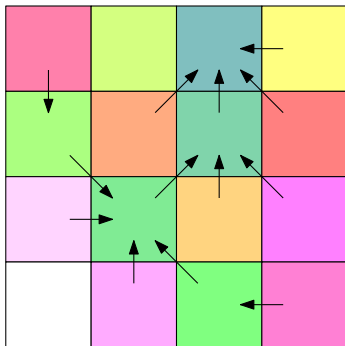


Time-forward processing (Arge et al.)

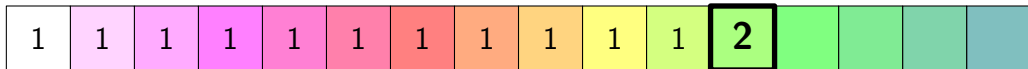
Goal: compute flow accumulation for each cell c

= #cells from which water passes through c

= size of tree rooted at c



output



input

pqueue

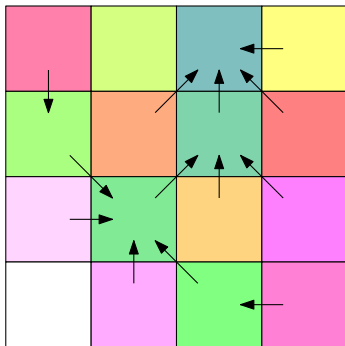


Time-forward processing (Arge et al.)

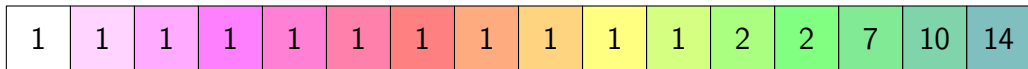
Goal: compute flow accumulation for each cell c

= #cells from which water passes through c

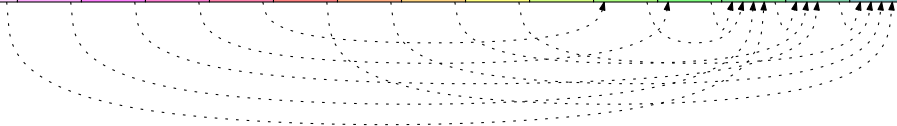
= size of tree rooted at c



output



input



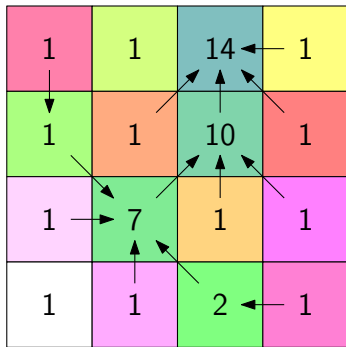
pqueue

Time-forward processing (Arge et al.)

Goal: compute flow accumulation for each cell c

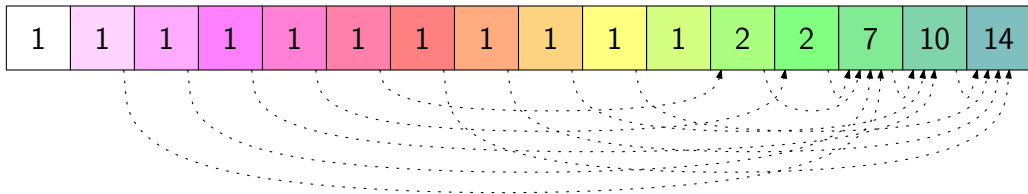
= #cells from which water passes through c

= size of tree rooted at c



sort

output



Time-forward processing

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

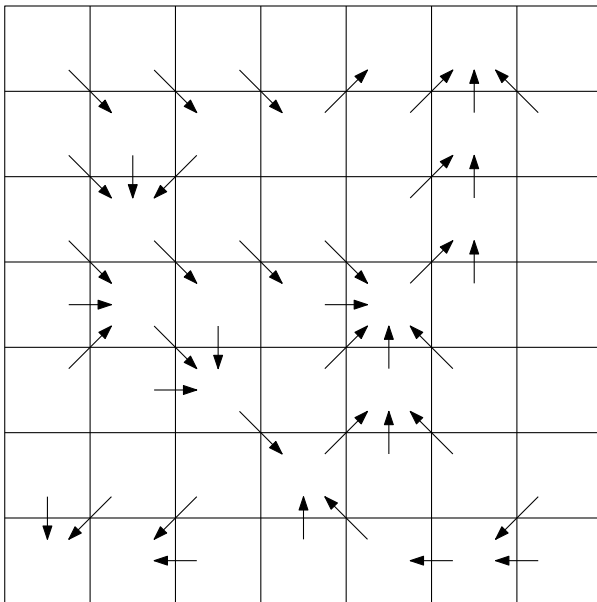
I/O-volume per grid cell (optimistic):

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

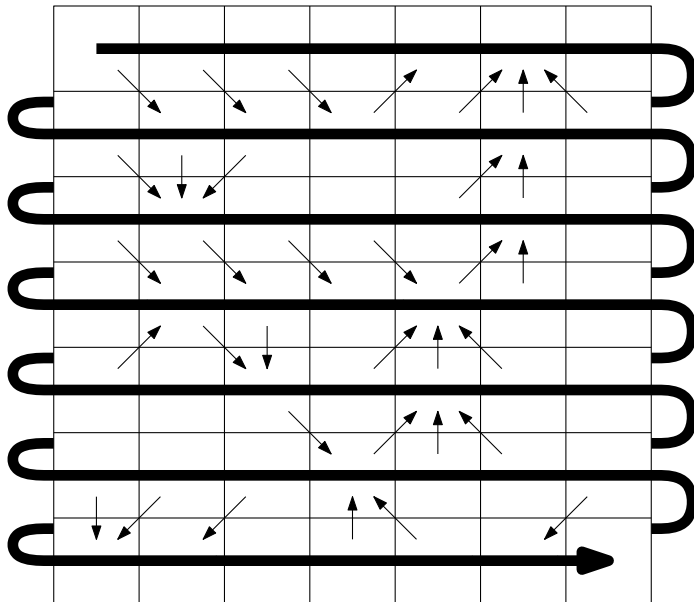
I/O-volume per grid cell (pessimistic):

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

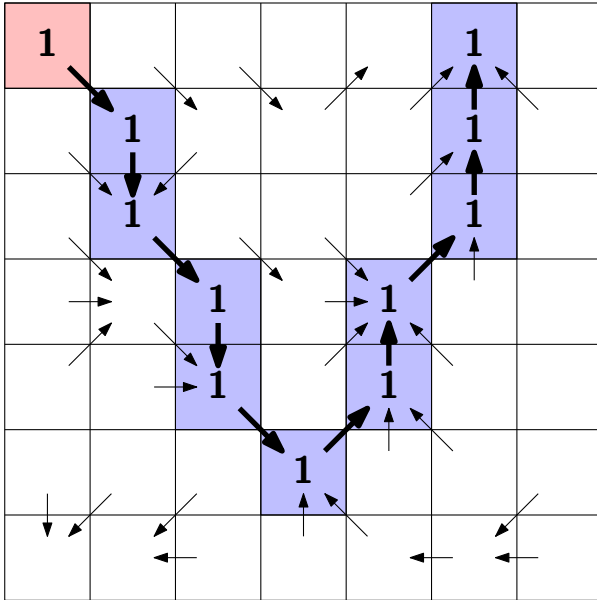
Naïve algorithm



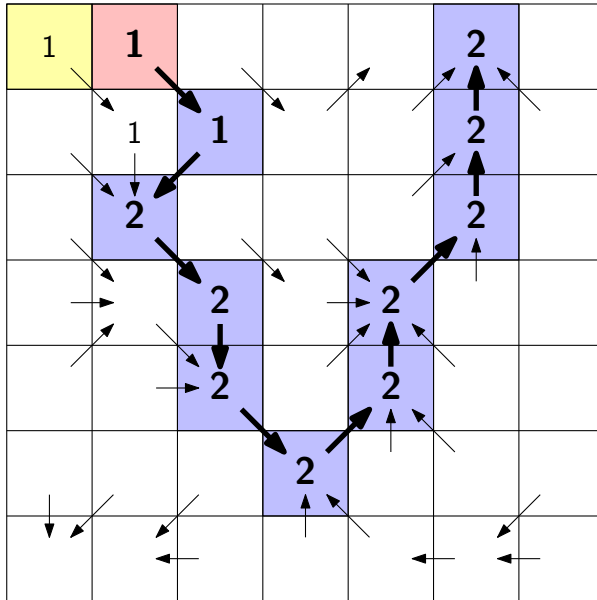
Naïve algorithm



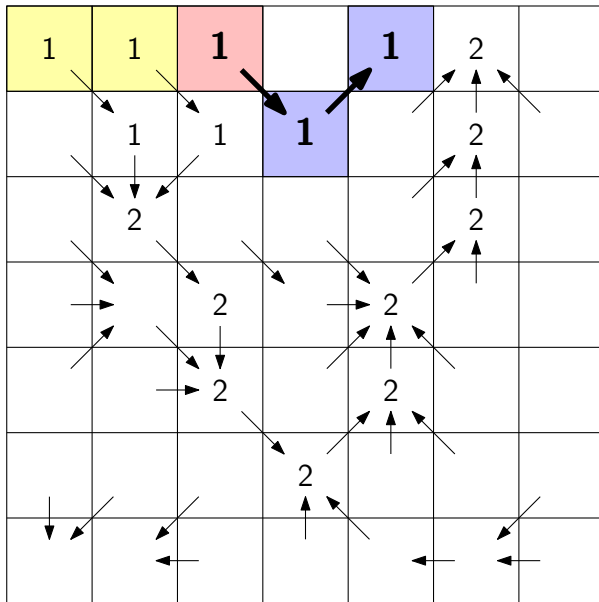
Naïve algorithm



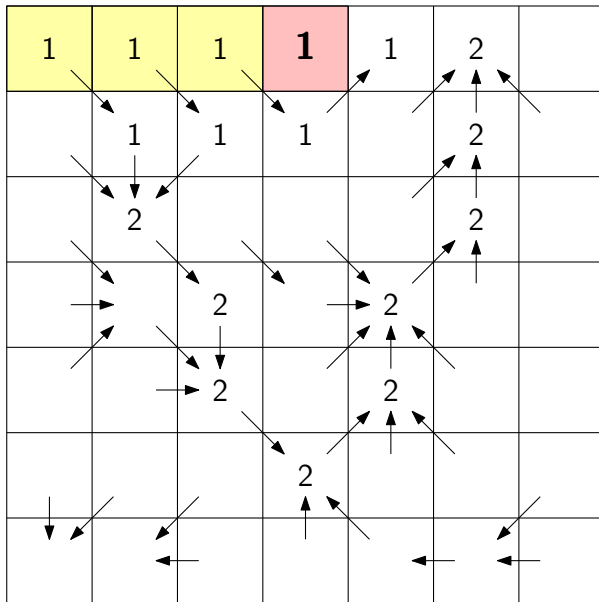
Naïve algorithm



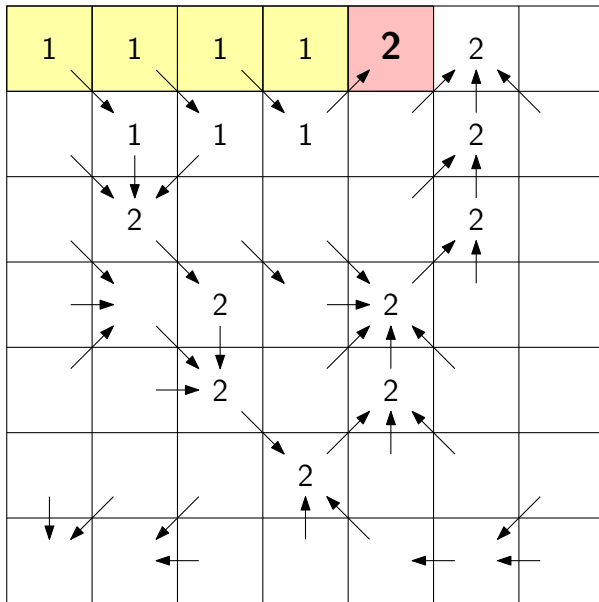
Naïve algorithm



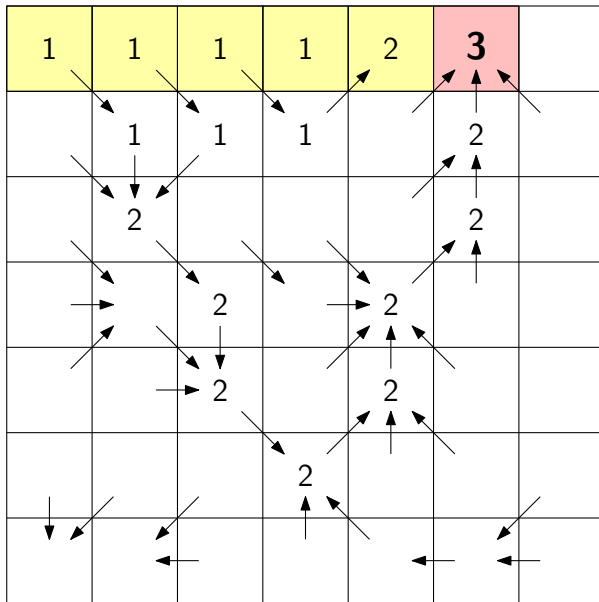
Naïve algorithm



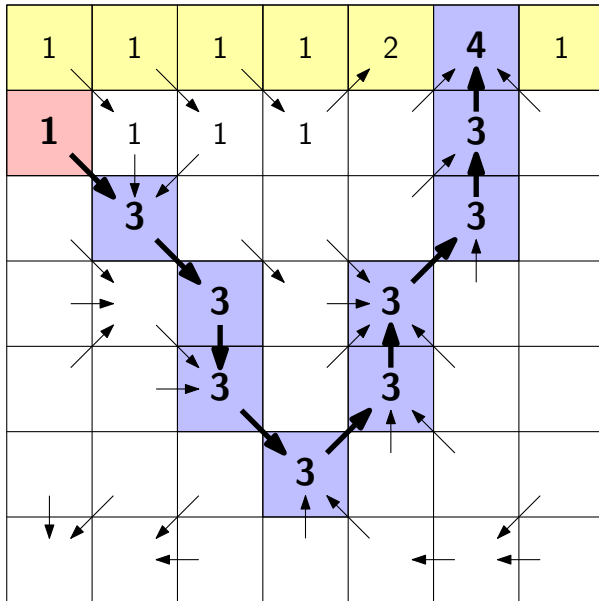
Naïve algorithm



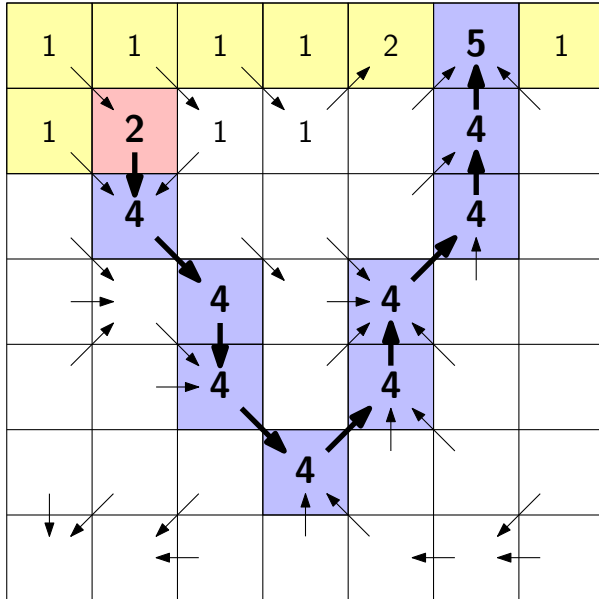
Naïve algorithm



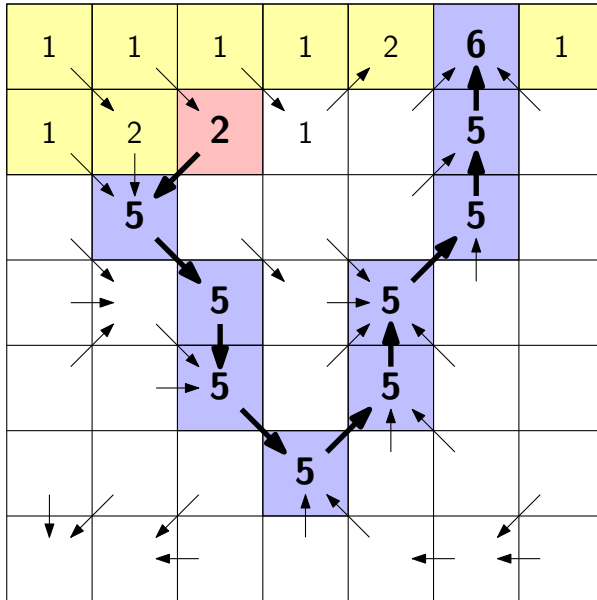
Naïve algorithm



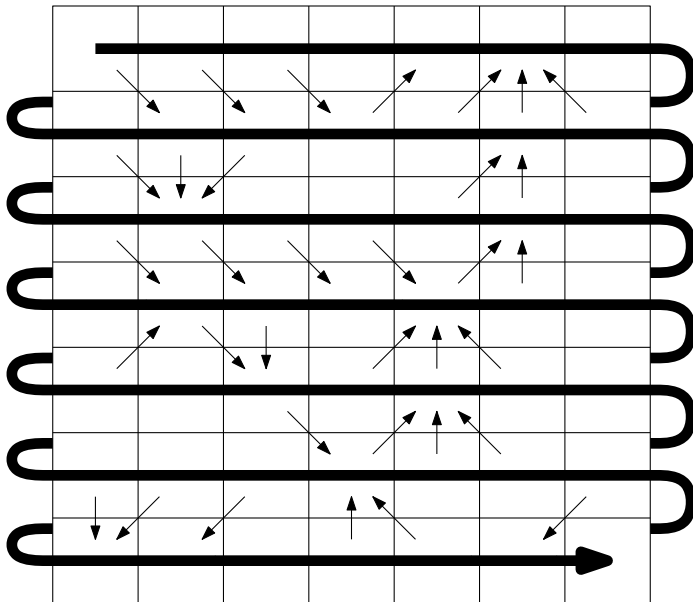
Naïve algorithm



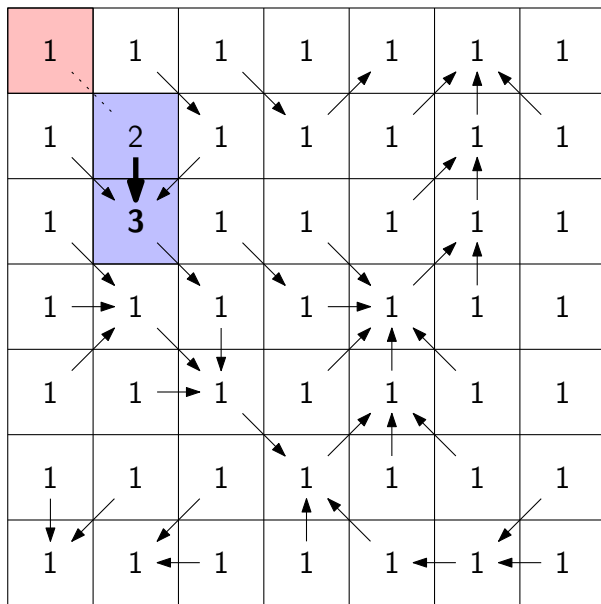
Naïve algorithm



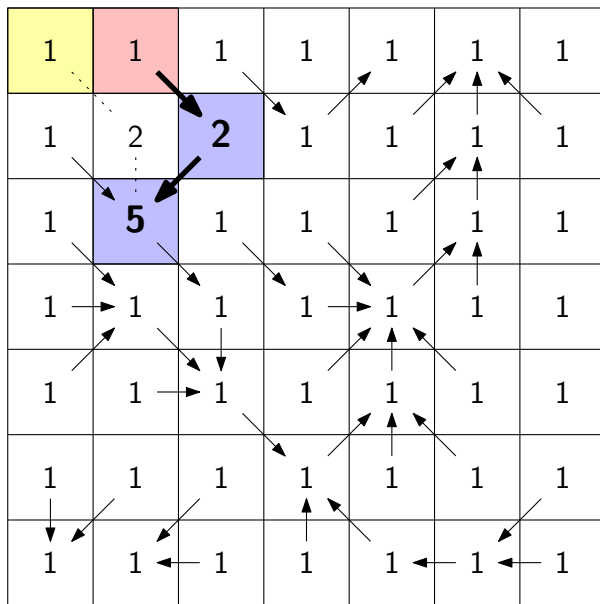
Row-by-row scan



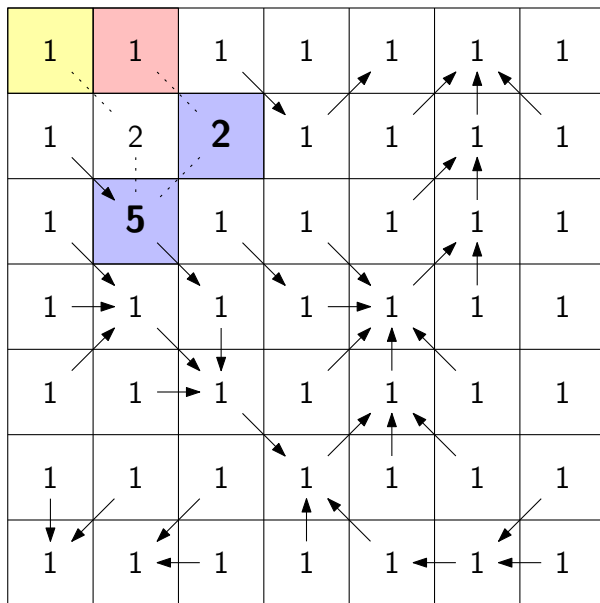
Row-by-row scan



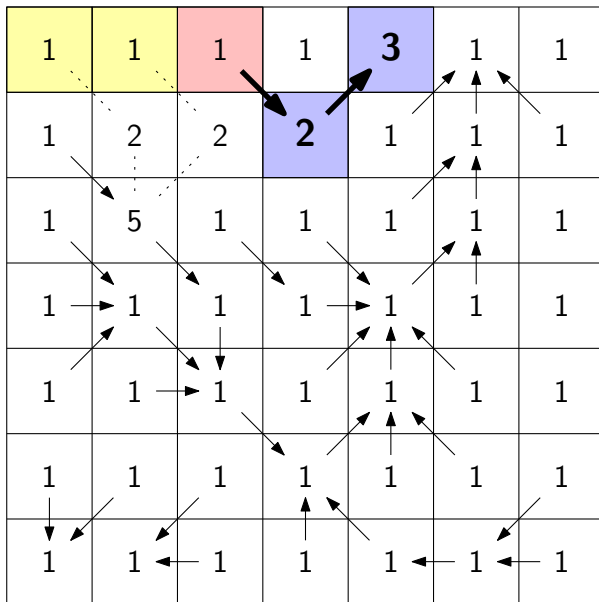
Row-by-row scan



Row-by-row scan



Row-by-row scan



Row-by-row scan

1	1	1	1	3	1	1
1	2	2	2	1	1	1
1	5	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

Row-by-row scan

1	1	1	1	3	1	1
1	2	2	2	1	1	1
1	5	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

Row-by-row scan

1	1	1	1	3	1	1
1	2	2	2	1	1	1
1	5	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

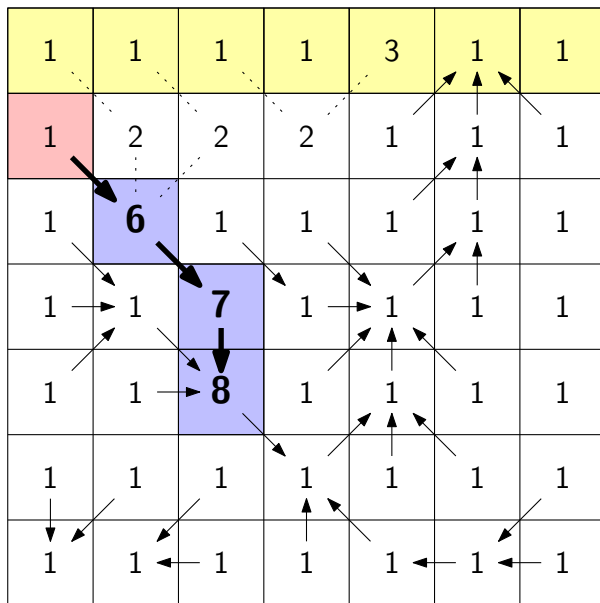
Row-by-row scan

1	1	1	1	3	1	1
1	2	2	2	1	1	1
1	5	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

Row-by-row scan

1	1	1	1	3	1	1
1	2	2	2	1	1	1
1	6	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

Row-by-row scan



Row-by-row scan

1	1	1	1	3	1	1
1	2	2	2	1	1	1
1	6	1	1	1	1	1
1	1	7	1	1	1	1
1	1	8	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

Row-by-row scan

1	1	1	1	3	1	1
1	2	2	2	1	1	1
1	6	1	1	1	1	1
1	1	7	1	1	1	1
1	1	8	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

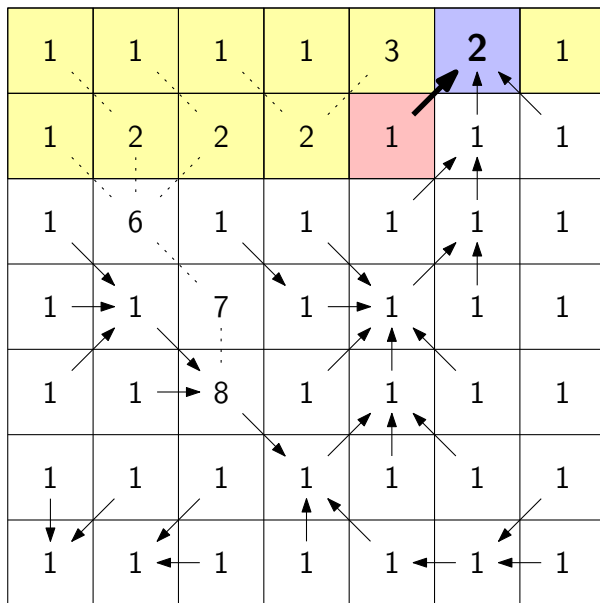
The diagram illustrates a row-by-row scan of a 7x7 grid. The grid contains numerical values and arrows indicating the scan path. The top row is highlighted in yellow, and the second row is highlighted in pink. Dotted lines show the path of the scan, and solid arrows show the direction of the scan at each step.

The grid values are:

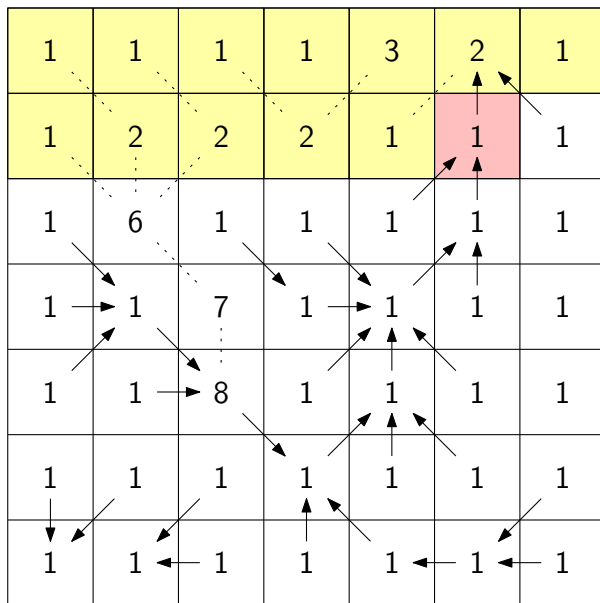
1	1	1	1	3	1	1
1	2	2	2	1	1	1
1	6	1	1	1	1	1
1	1	7	1	1	1	1
1	1	8	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

The scan path is indicated by dotted lines and arrows. The path starts at the top-left cell (1,1) and moves horizontally across the first row. From the top-right cell (1,7), it moves diagonally down to (2,6), then diagonally down to (3,5), then diagonally down to (4,4), then diagonally down to (5,3), then diagonally down to (6,2), and finally diagonally down to (7,1). From (7,1), it moves horizontally across the bottom row. From (7,7), it moves diagonally up to (6,6), then diagonally up to (5,5), then diagonally up to (4,4), then diagonally up to (3,3), then diagonally up to (2,2), and finally diagonally up to (1,1).

Row-by-row scan



Row-by-row scan



Row-by-row scan

The diagram shows a 7x7 grid with numerical values and arrows indicating dependencies. The grid is as follows:

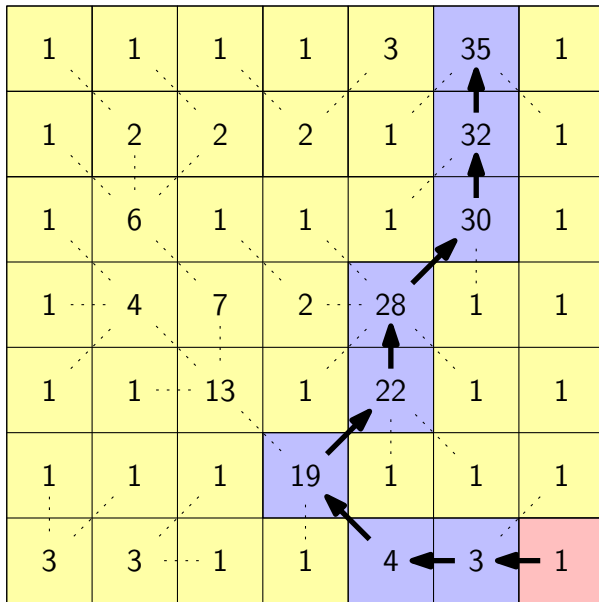
1	1	1	1	3	3	1
1	2	2	2	1	2	1
1	6	1	1	1	2	1
1	4	7	2	6	1	1
1	1	13	1	3	1	1
1	1	1	15	1	1	1
3	3	1	1	1	2	1

Arrows indicate dependencies:

- Vertical arrows pointing up: (1,6) ← (2,6), (2,6) ← (3,6), (4,6) ← (5,6), (5,6) ← (6,6).
- Diagonal arrows pointing up-right: (2,6) ← (3,5), (3,6) ← (4,5), (4,6) ← (5,5), (5,6) ← (6,5), (6,6) ← (7,5).
- Diagonal arrows pointing down-left: (1,6) ← (2,5), (2,5) ← (3,4), (3,4) ← (4,3), (4,3) ← (5,2), (5,2) ← (6,1), (6,1) ← (7,2).
- Horizontal arrows pointing left: (7,6) ← (7,7), (7,5) ← (7,6).
- Diagonal arrows pointing down-right: (7,5) ← (6,4), (6,4) ← (5,3), (5,3) ← (4,2), (4,2) ← (3,1).

The cell at row 7, column 6 (value 2) is highlighted in pink.

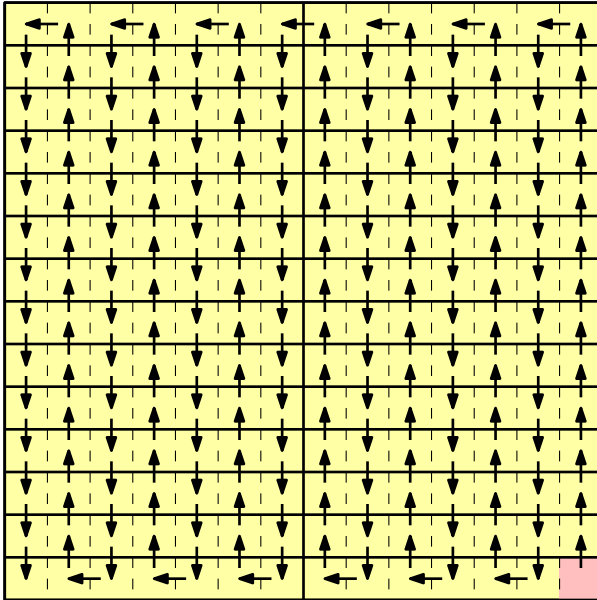
Row-by-row scan



Running time:
 $\Theta(n)$

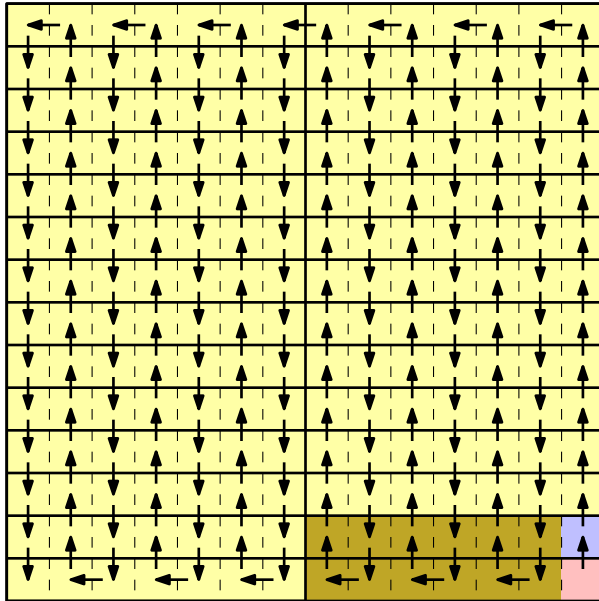
Row-by-row scan

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



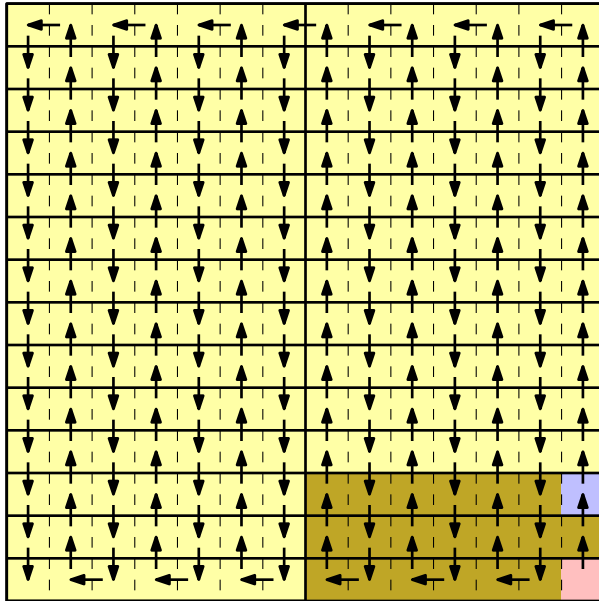
Row-by-row scan

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



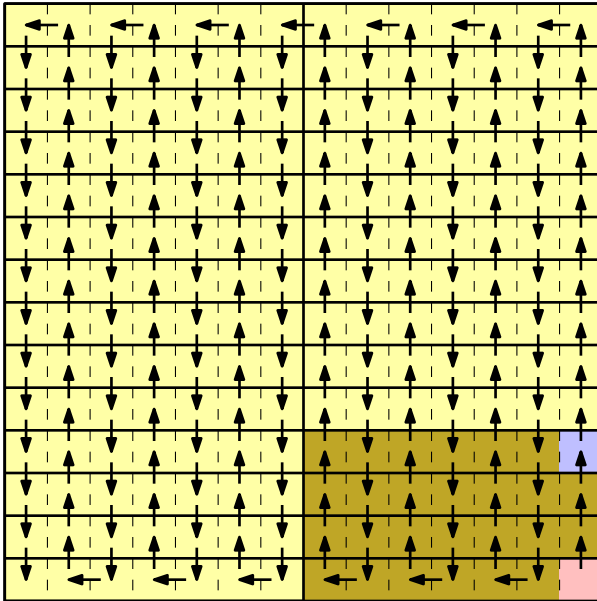
Row-by-row scan

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



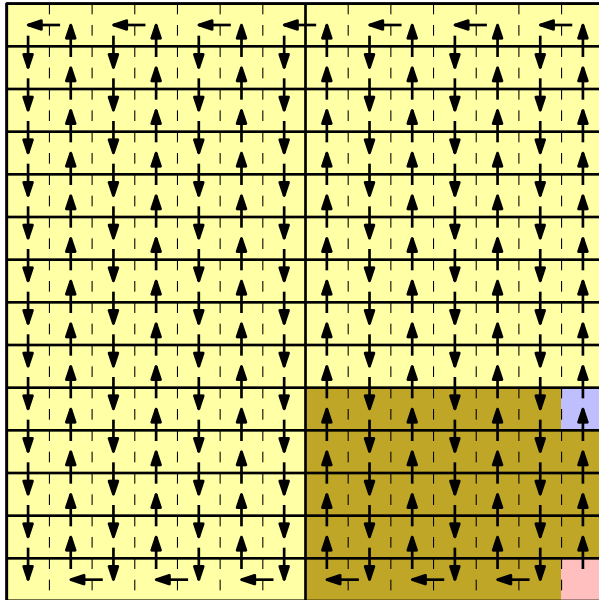
Row-by-row scan

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



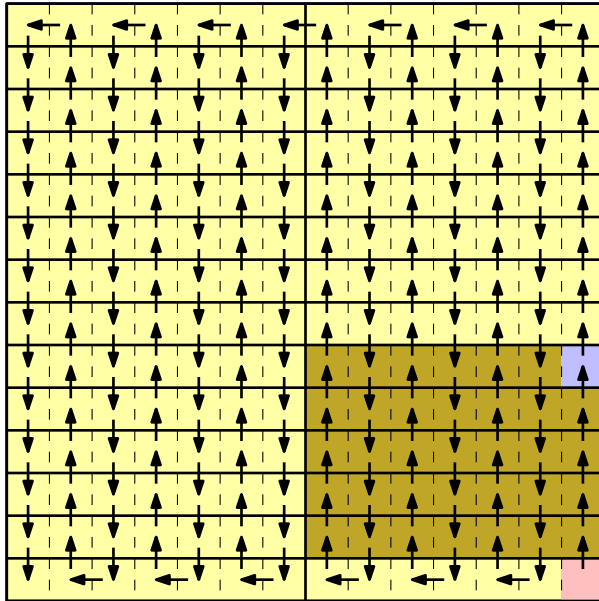
Row-by-row scan

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



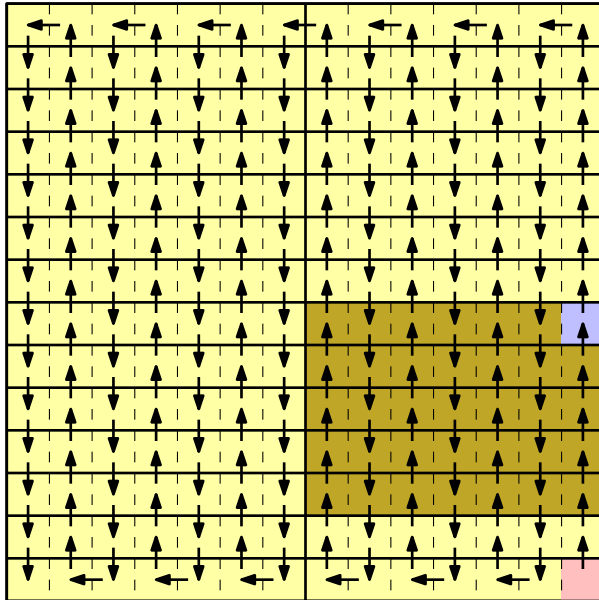
Row-by-row scan

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



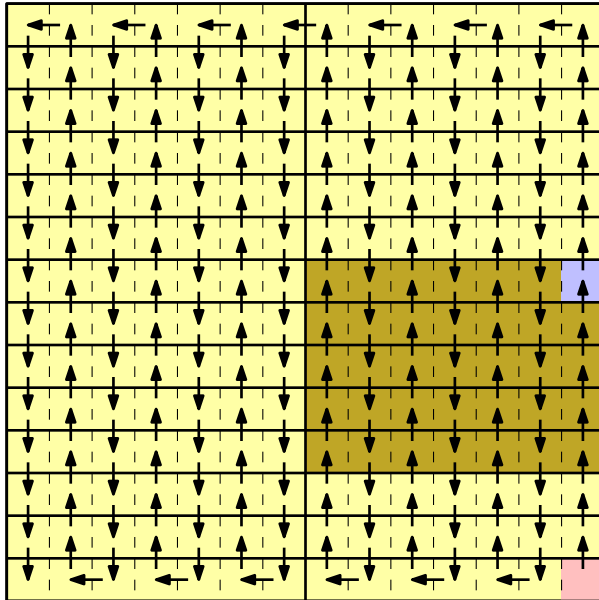
Row-by-row scan

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



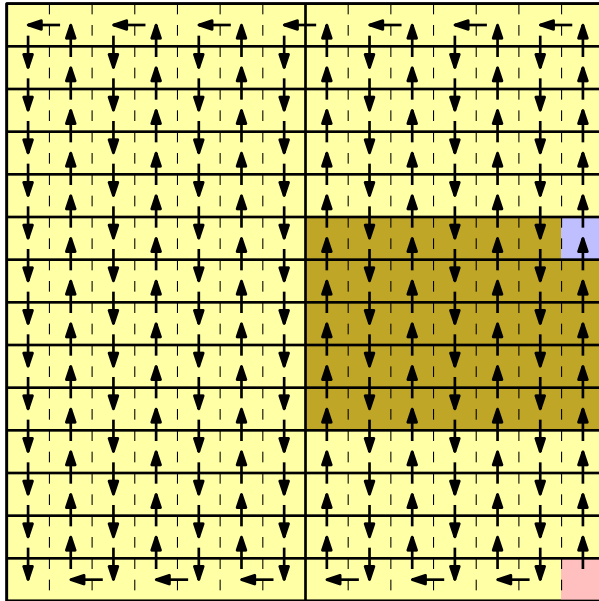
Row-by-row scan

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



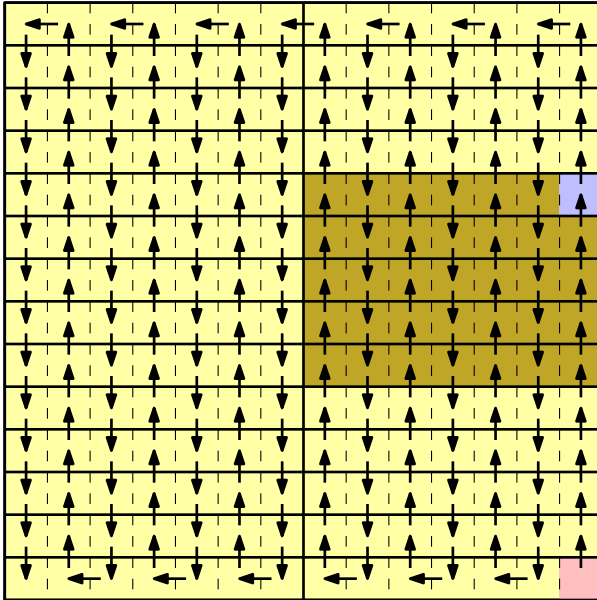
Row-by-row scan

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



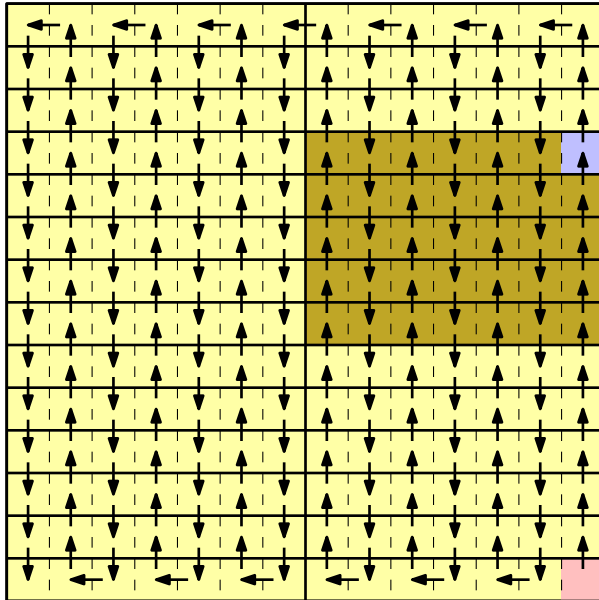
Row-by-row scan

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



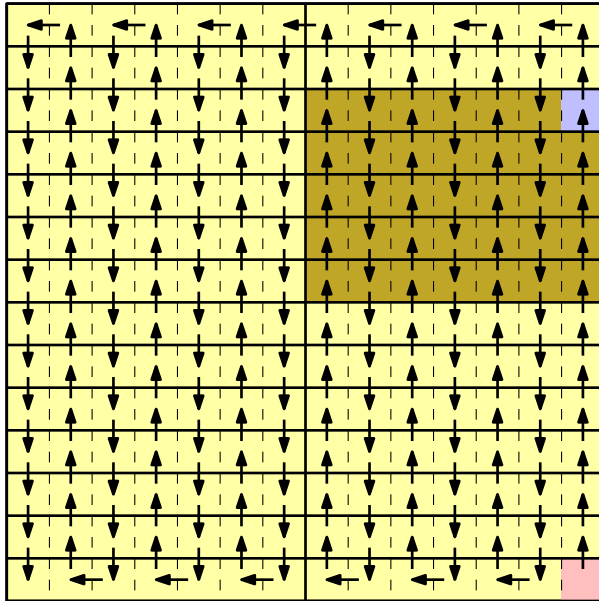
Row-by-row scan

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



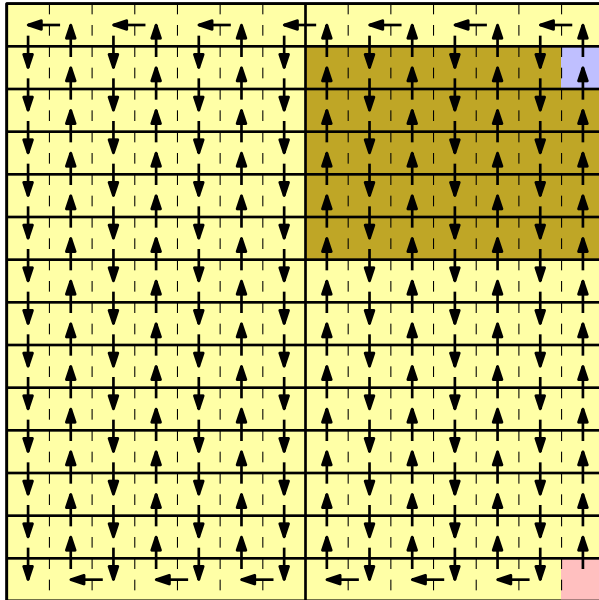
Row-by-row scan

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



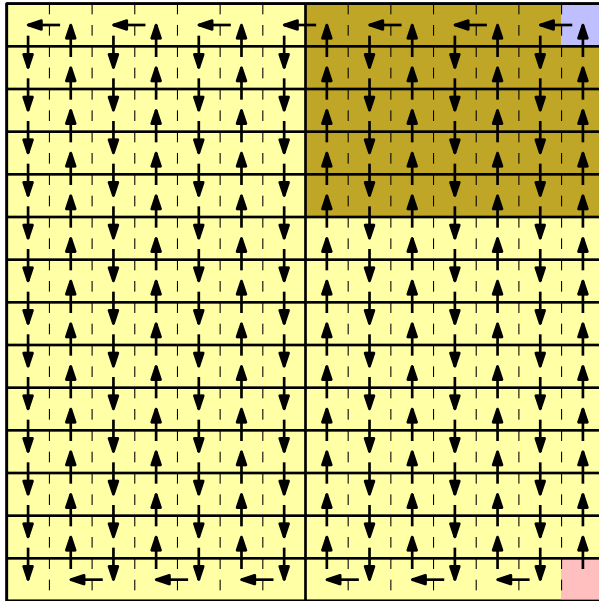
Row-by-row scan

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



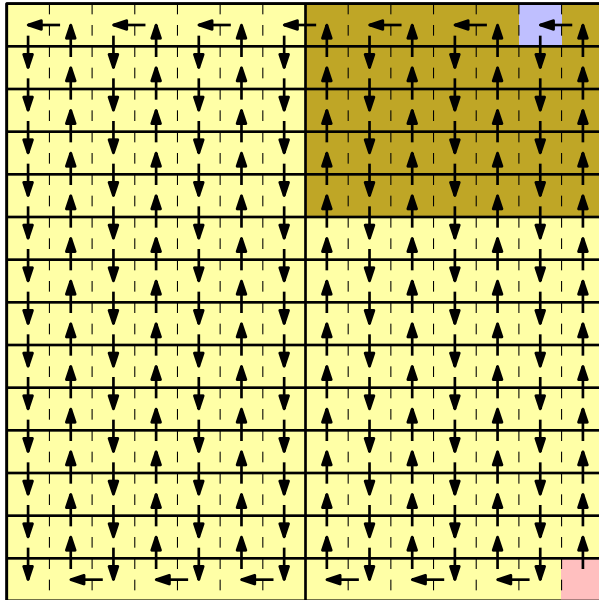
Row-by-row scan

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



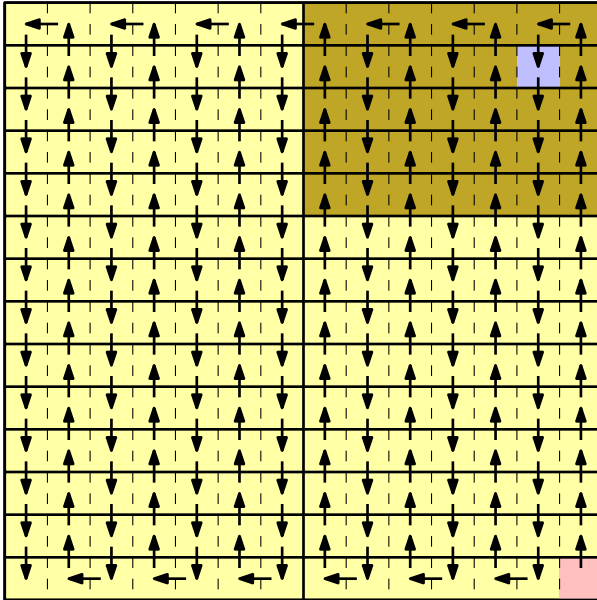
Row-by-row scan

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



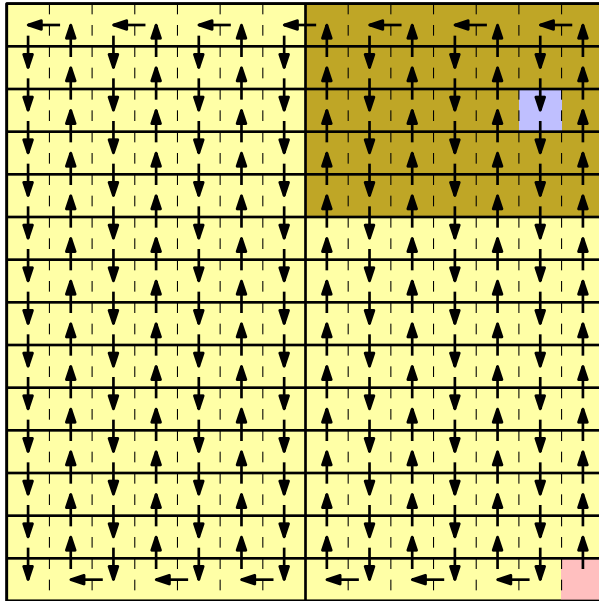
Row-by-row scan

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



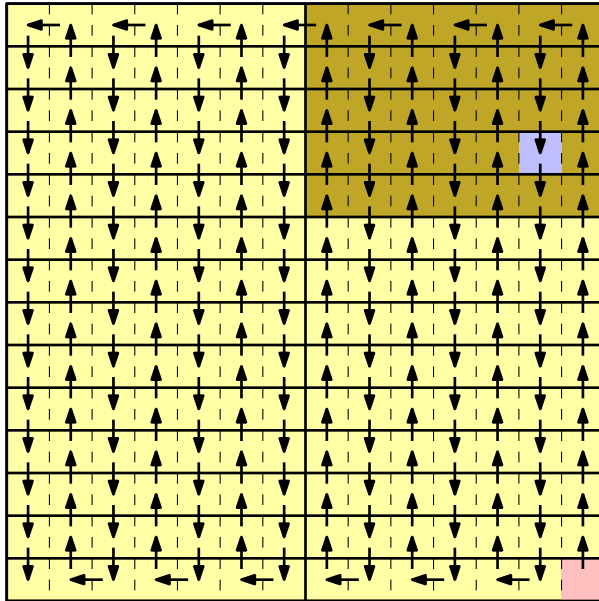
Row-by-row scan

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



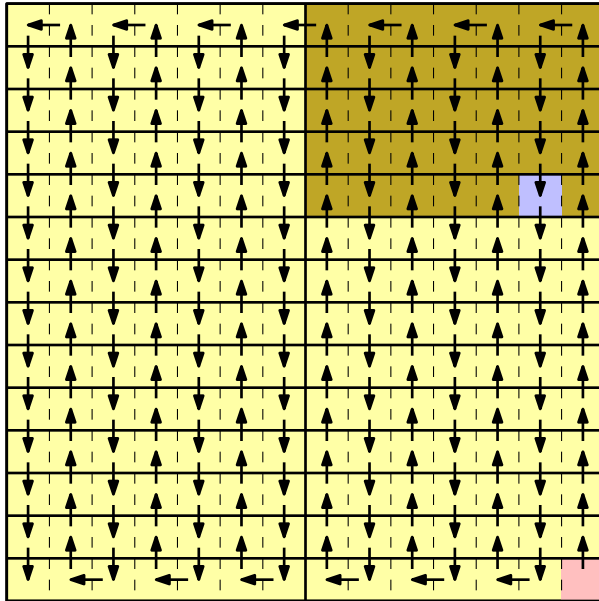
Row-by-row scan

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



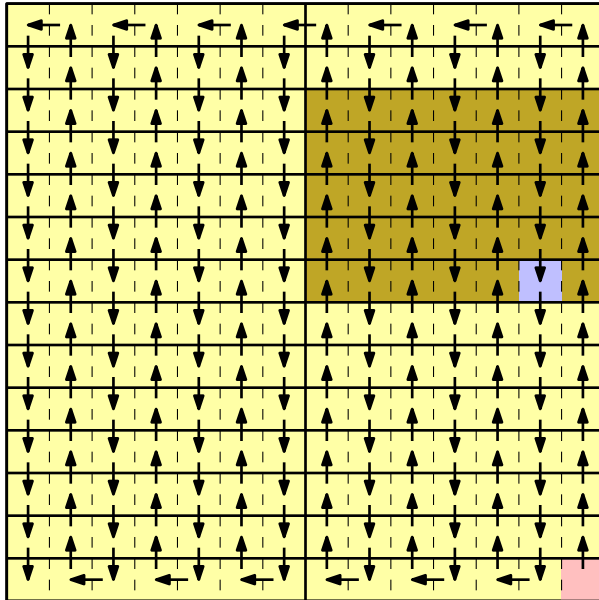
Row-by-row scan

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



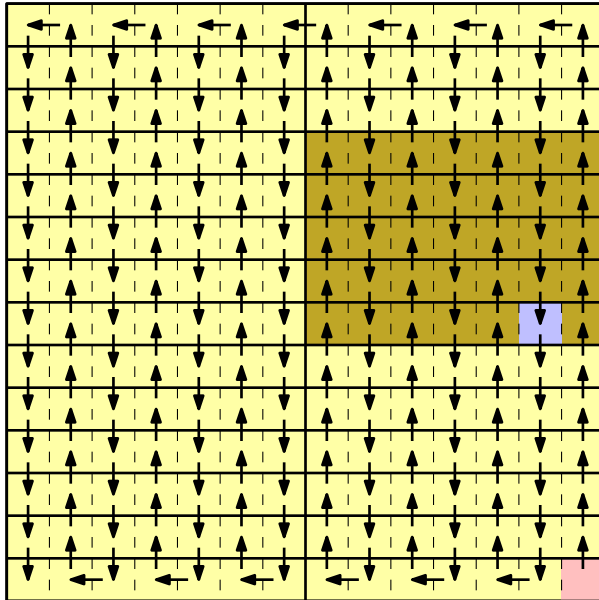
Row-by-row scan

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

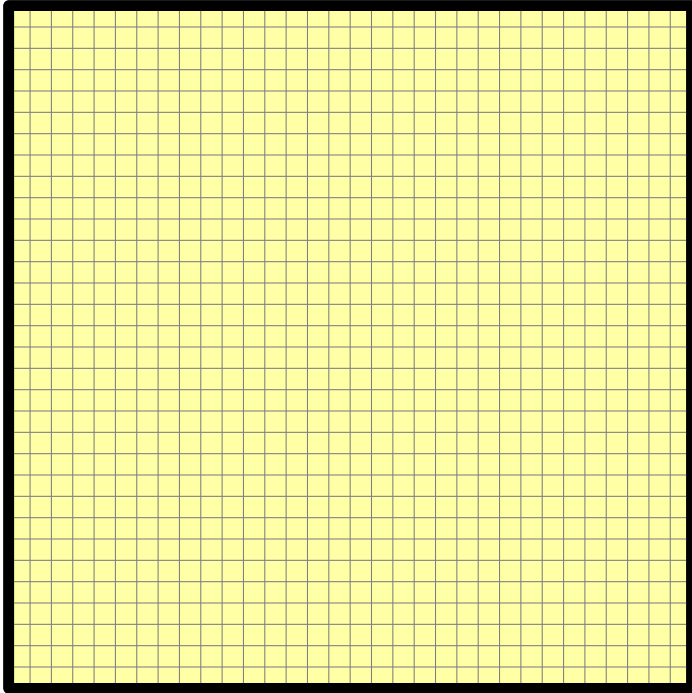


Row-by-row scan

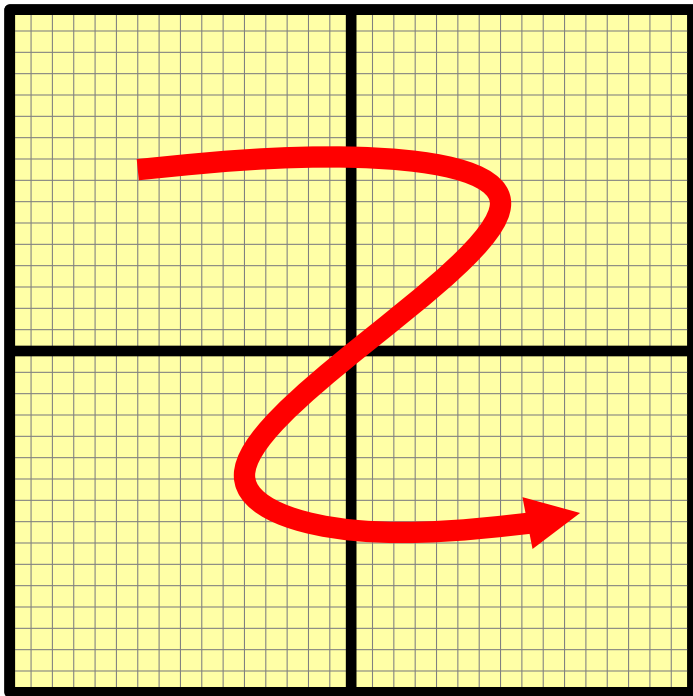
$\Theta(N)$ I/O's in the worst case



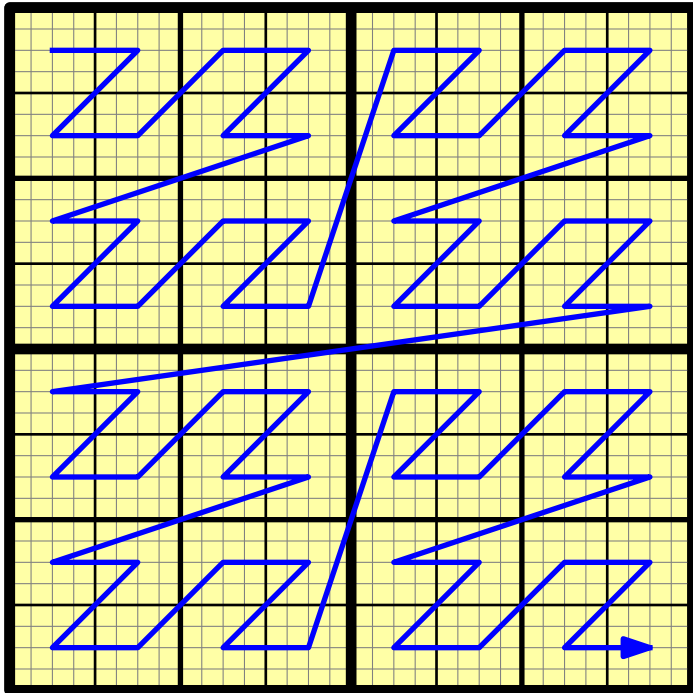
Z-order scan



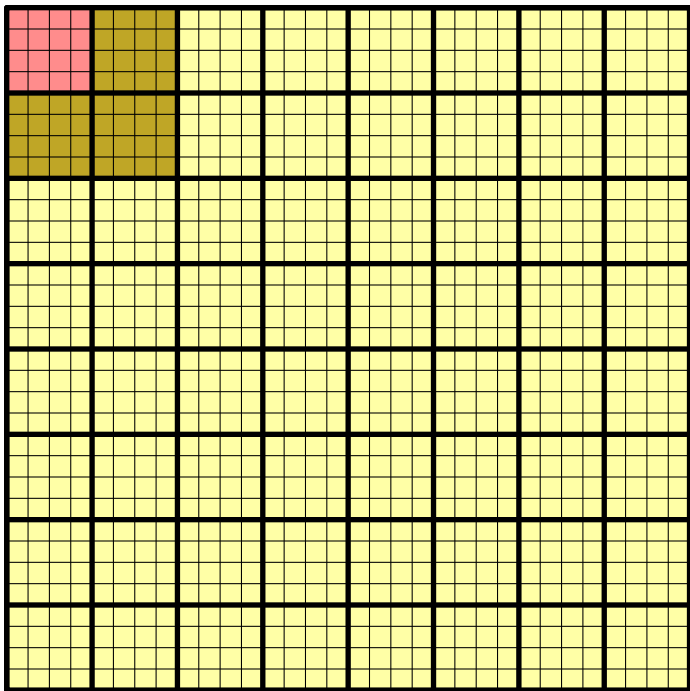
Z-order scan



Z-order scan

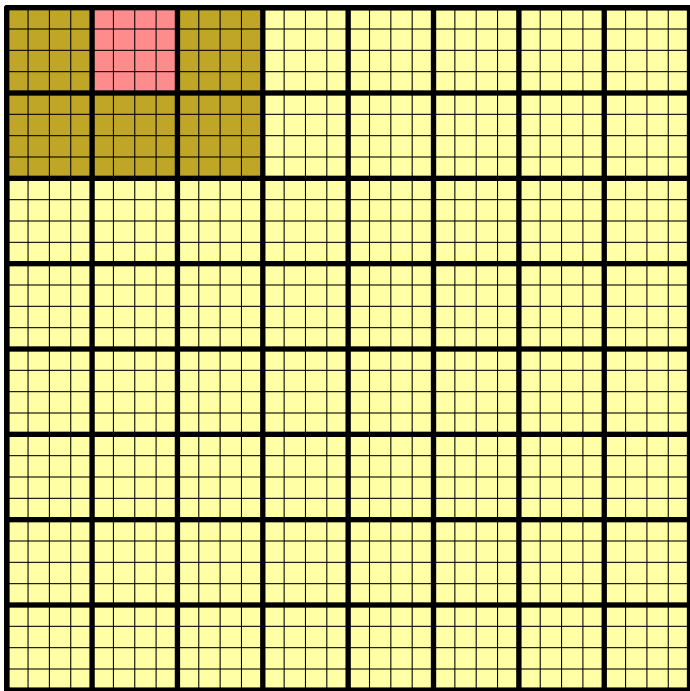


Z-order scan on Z-order file



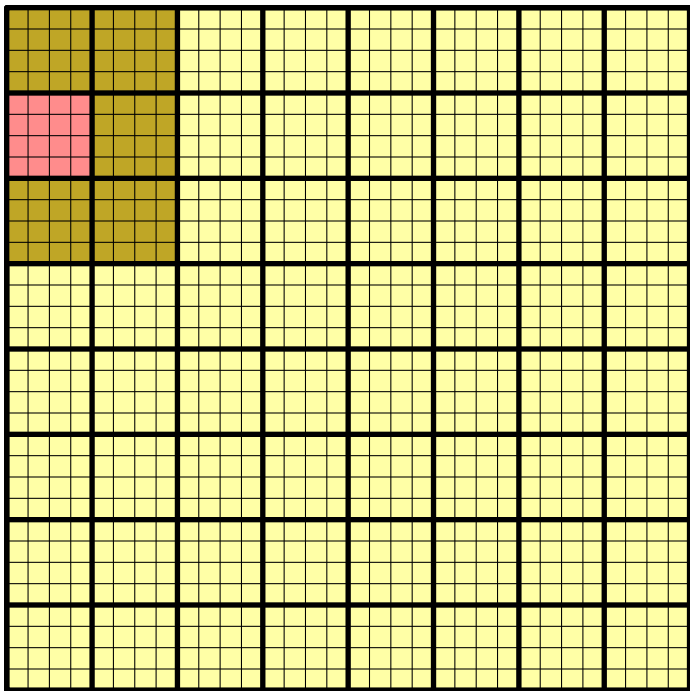
While working on a **block**, have its neighbours in **memory** too

Z-order scan on Z-order file



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

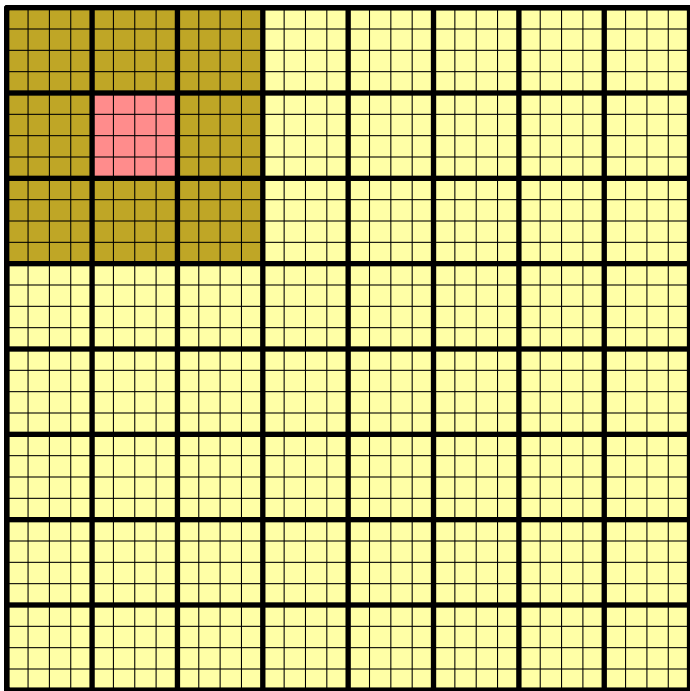
Z-order scan on Z-order file



} \sqrt{B}

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

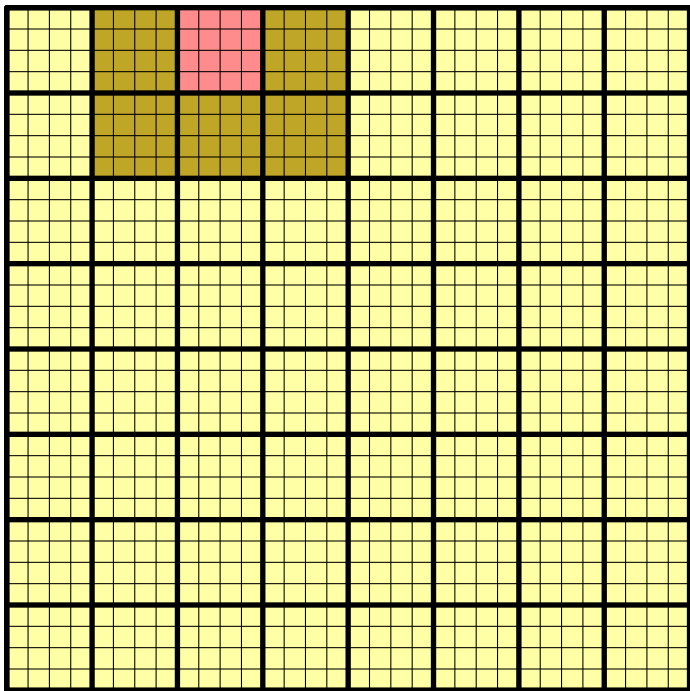
Z-order scan on Z-order file



} \sqrt{B}

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

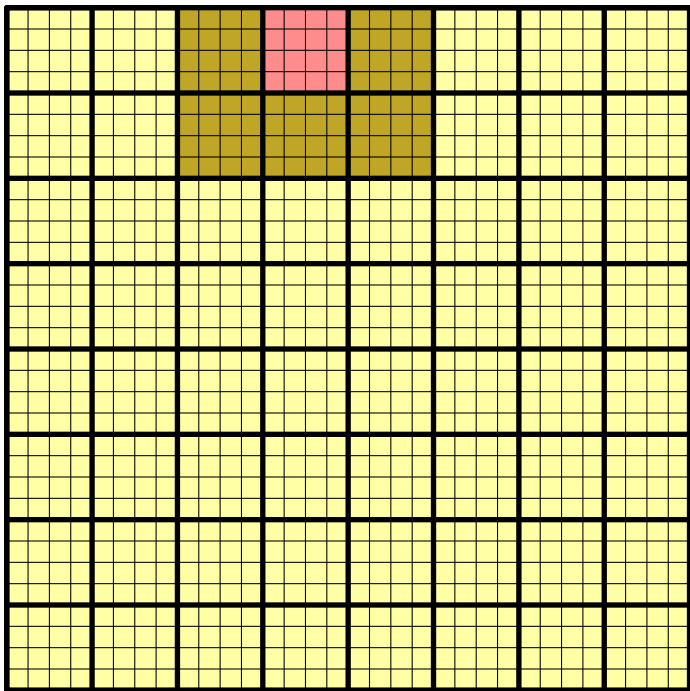
Z-order scan on Z-order file



} \sqrt{B}

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

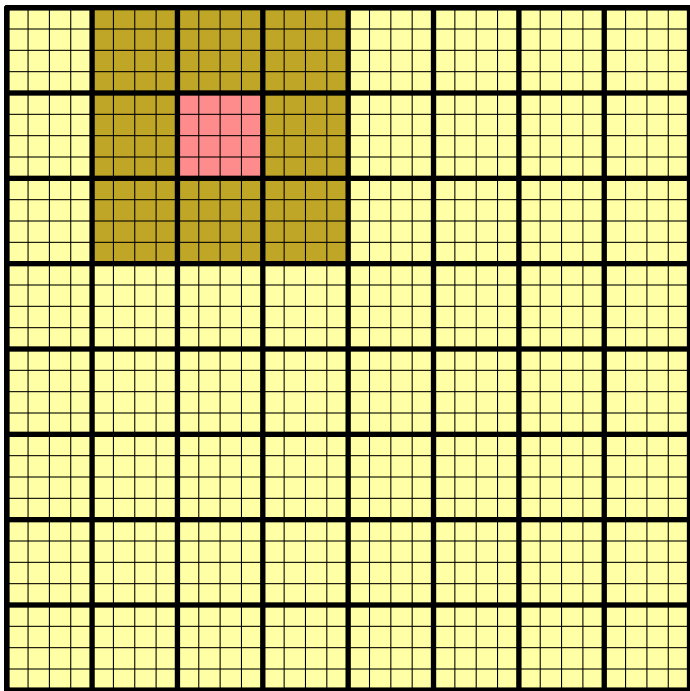
Z-order scan on Z-order file



} \sqrt{B}

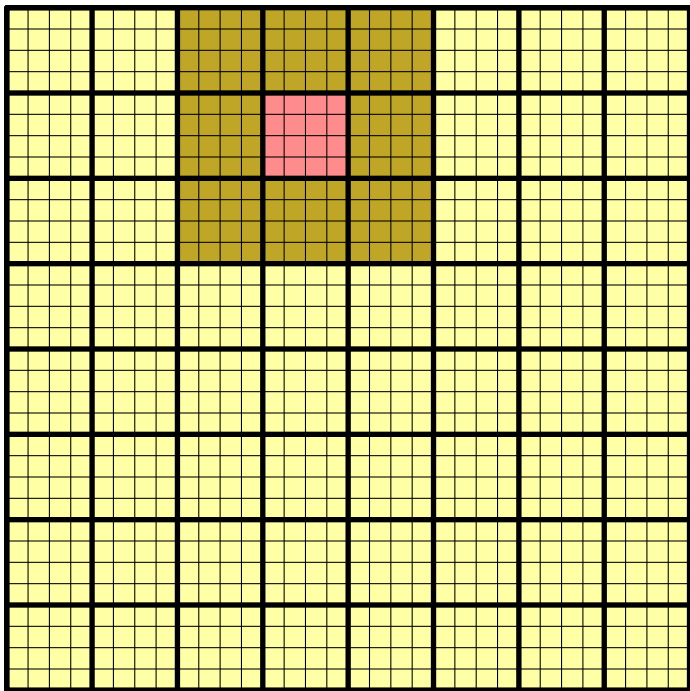
While working on a block, have its neighbours in memory too

Z-order scan on Z-order file



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

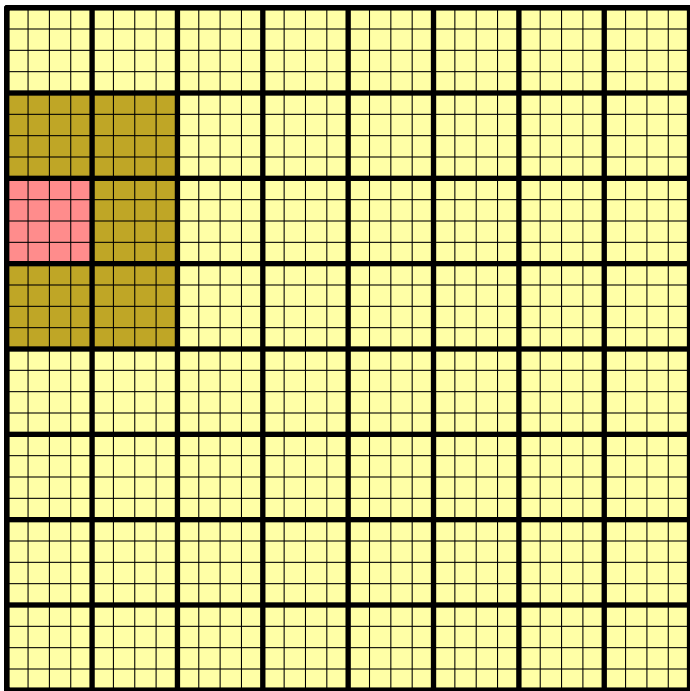
Z-order scan on Z-order file



} \sqrt{B}

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

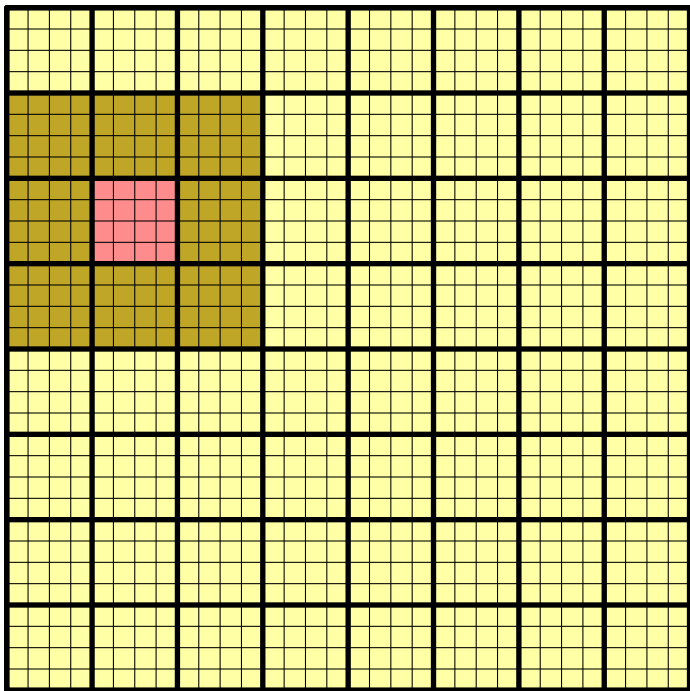
Z-order scan on Z-order file



} \sqrt{B}

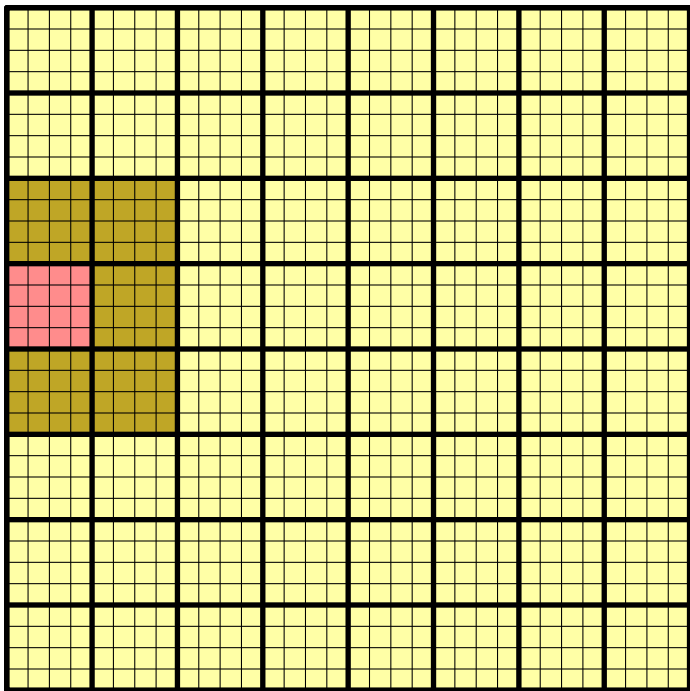
While working on a block, have its neighbours in memory too

Z-order scan on Z-order file



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

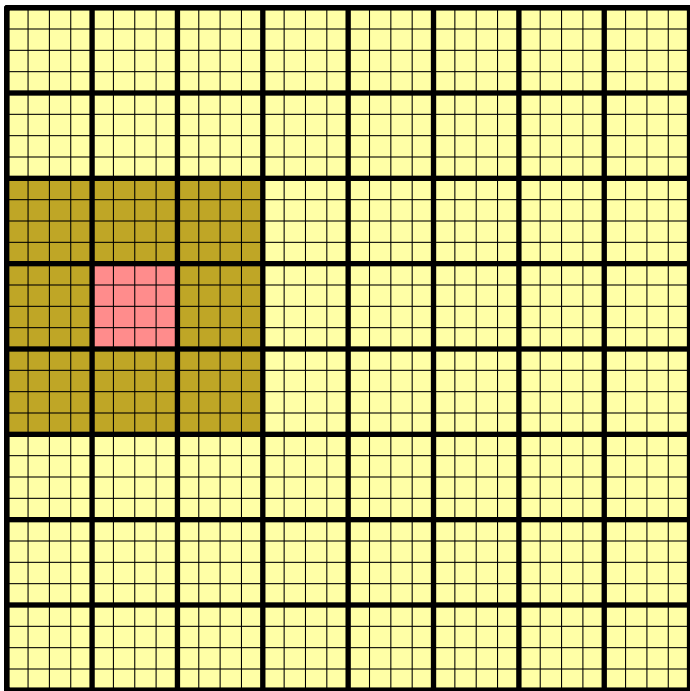
Z-order scan on Z-order file



} \sqrt{B}

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

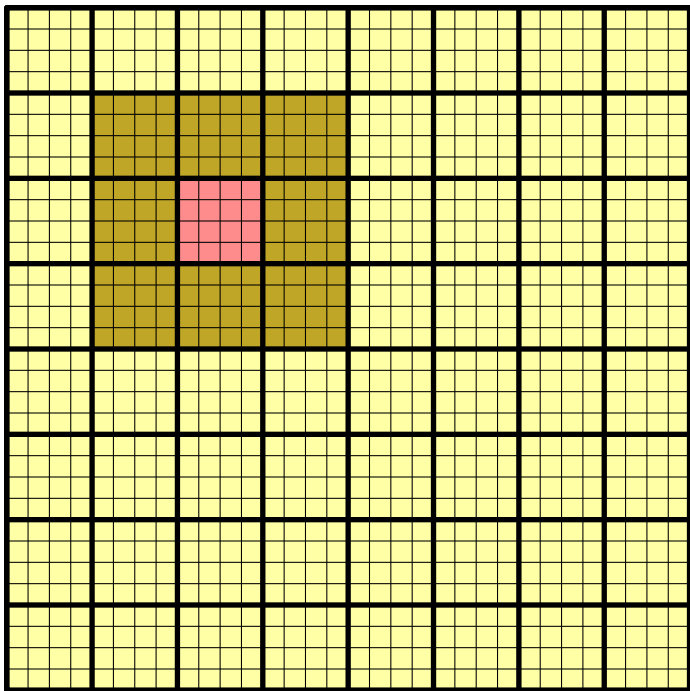
Z-order scan on Z-order file



} \sqrt{B}

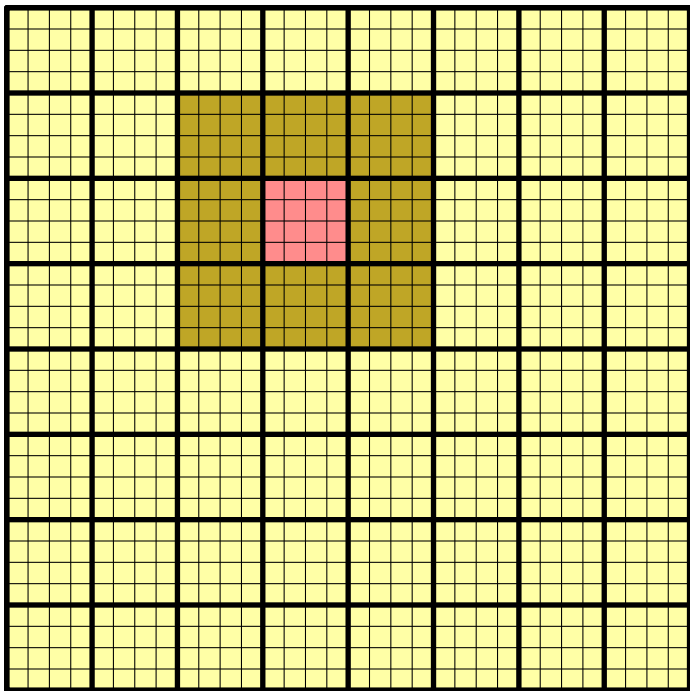
While working
on a **block**,
have its
neighbours in
memory too

Z-order scan on Z-order file



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

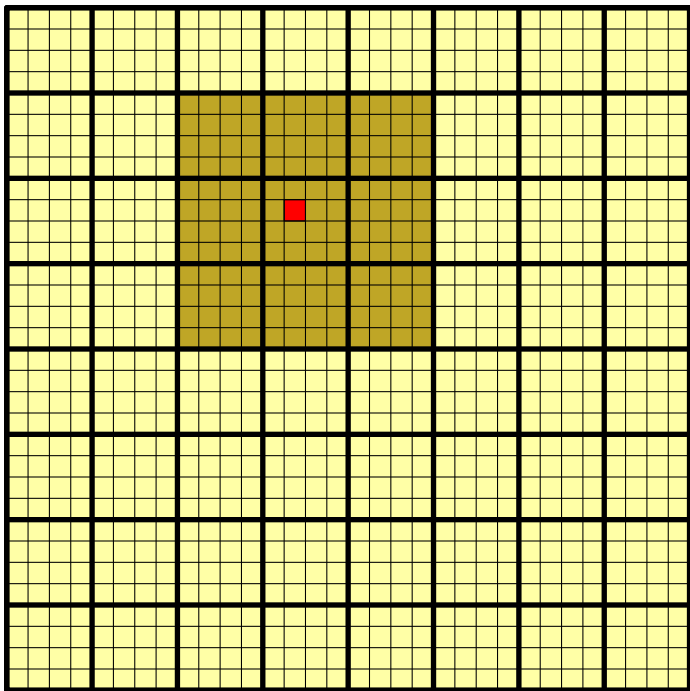
Z-order scan on Z-order file



} \sqrt{B}

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

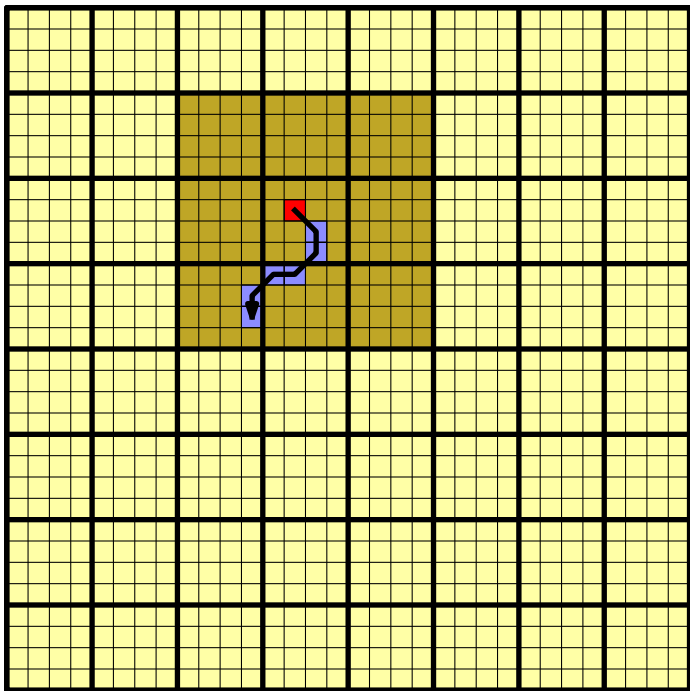
Z-order scan on Z-order file



} \sqrt{B}

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

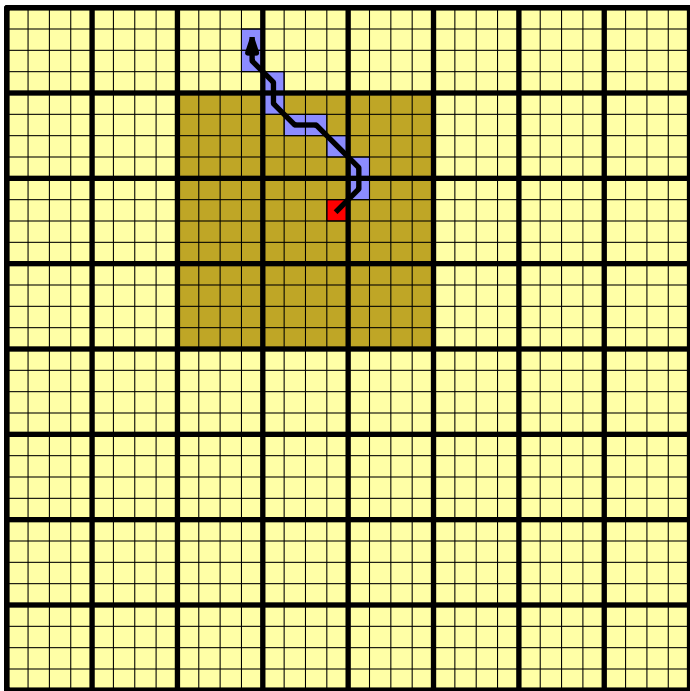
Z-order scan on Z-order file



} \sqrt{B}

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

Z-order scan on Z-order file



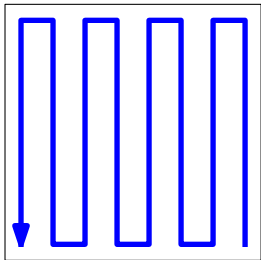
} \sqrt{B}

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

Only long paths require additional swapping

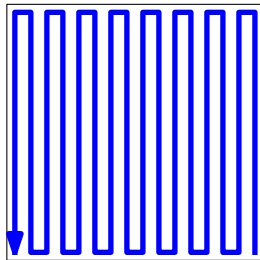
Worst-case terrains vs. real terrains

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

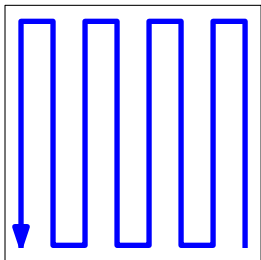


$\Omega(\sqrt{N})$ big turns

Worst-case, size $4n$

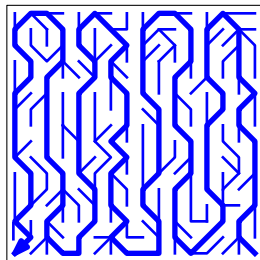


Realistic, size $n = 8^2$



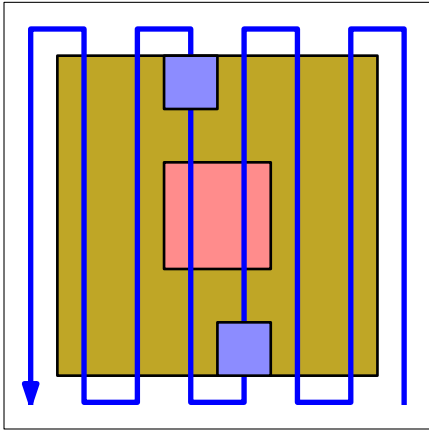
$\Theta(1)$ big turns

Realistic, size $4n$

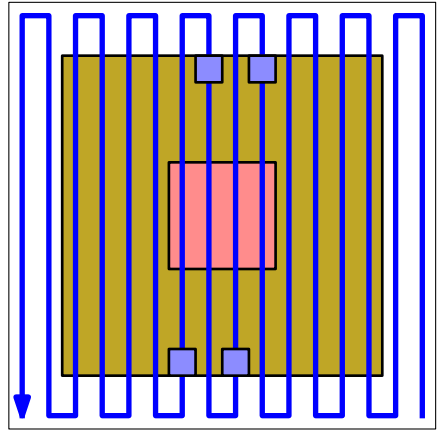


Worst-case terrains vs. real terrains

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



Worst-case, size $4n$

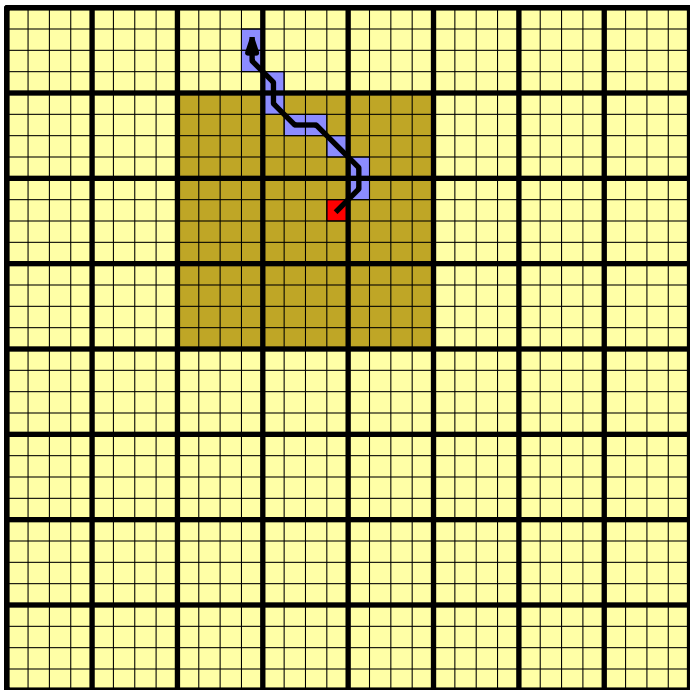


$Q' = Q$ scaled by factor 3.

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.

Z-order scan on Z-order file



} \sqrt{B}

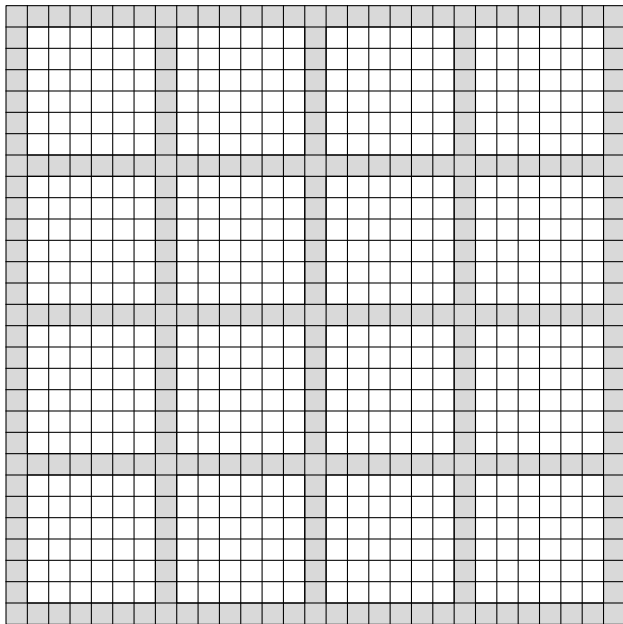
While working on a block, have its neighbours in memory too

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

Only long paths require additional swapping

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

Cache-aware separator-based algorithm

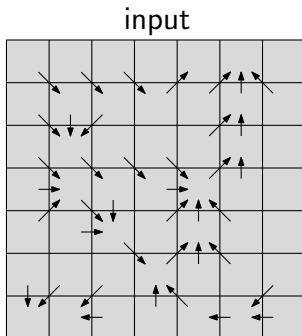


Separator cells: each
 $\Theta(\sqrt{M})$ -th row/column
→
divide grid into $\Theta(N/M)$
subgrids of size $\Theta(M)$

$\Theta(\sqrt{M})$

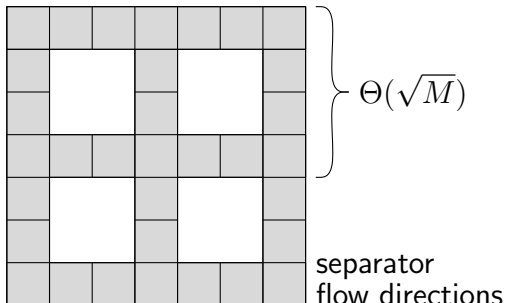
Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators



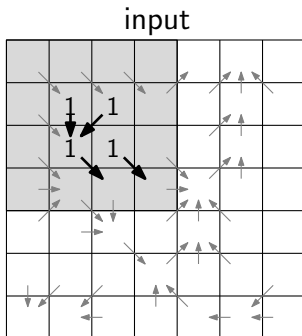
1	1	1	1	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1

separator
flow accumul.



Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators



1	1	1	1	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1

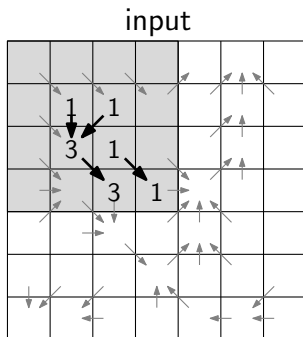
separator
flow accumul.

} $\Theta(\sqrt{M})$

separator
flow directions

Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators



1	1	1	1	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1

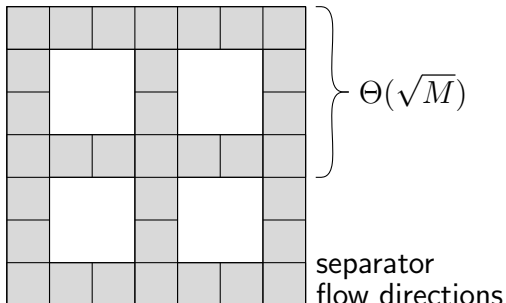
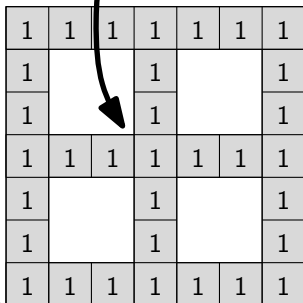
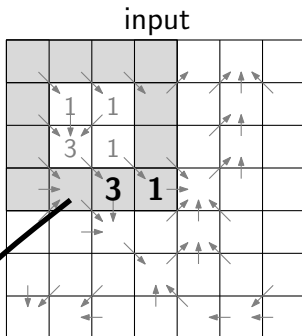
separator
flow accumul.

} $\Theta(\sqrt{M})$

separator
flow directions

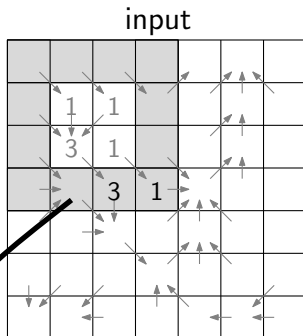
Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators



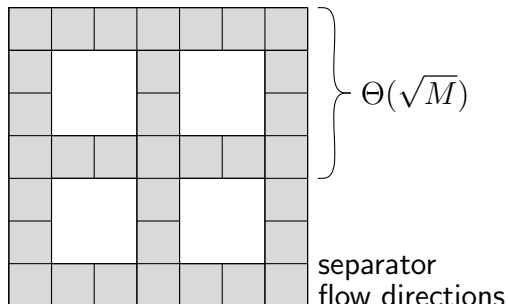
Cache-aware separator-based algorithm

1. move flow from interior of subgrids to separators



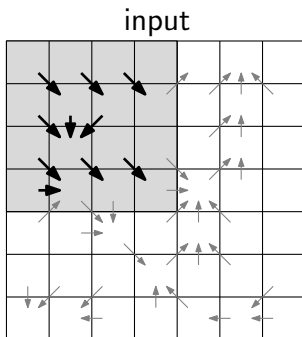
separator flow accumul.

1	1	1	1	1	1	1
1			1			1
1			1			1
1	1	4	2	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1



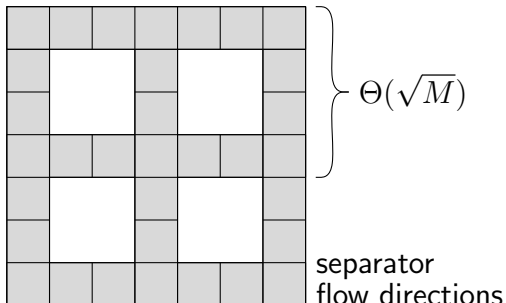
Cache-aware separator-based algorithm

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



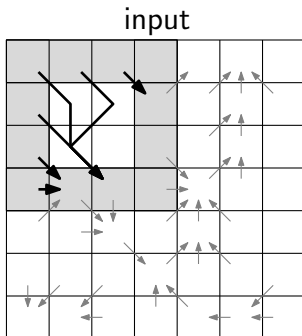
1	1	1	1	1	1	1
1			1			1
1			1			1
1	1	4	2	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1

separator
flow accumul.



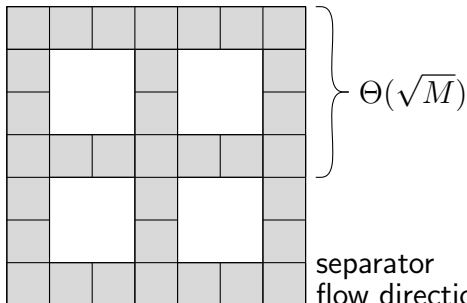
Cache-aware separator-based algorithm

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



1	1	1	1	1	1	1
1			1			1
1			1			1
1	1	4	2	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1

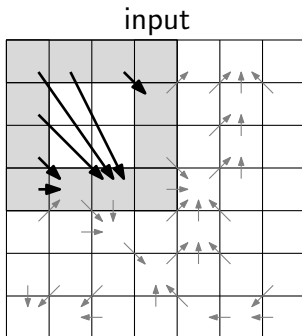
separator
flow accumul.



separator
flow directions

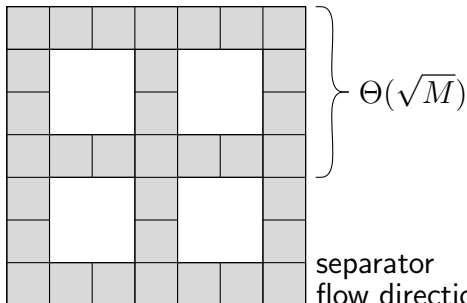
Cache-aware separator-based algorithm

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



1	1	1	1	1	1	1
1			1			1
1			1			1
1	1	4	2	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1

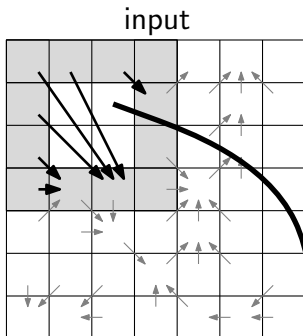
separator
flow accumul.



separator
flow directions

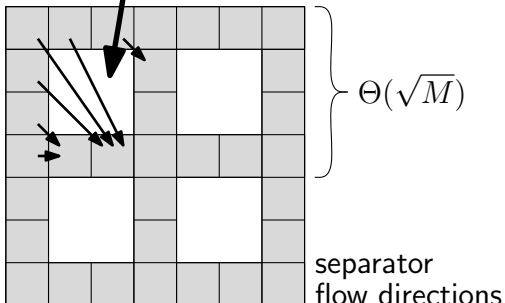
Cache-aware separator-based algorithm

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



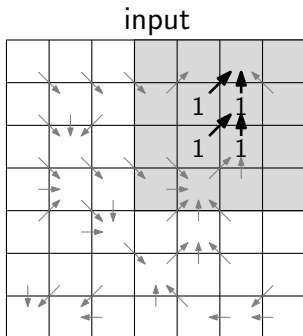
1	1	1	1	1	1	1
1			1			1
1			1			1
1	1	4	2	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1

separator
flow accumul.



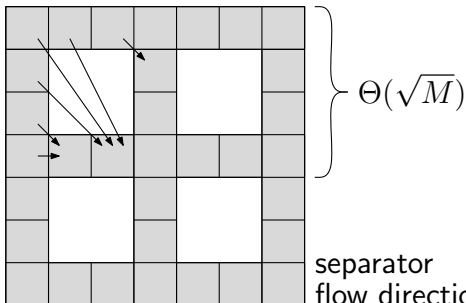
Cache-aware separator-based algorithm

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



1	1	1	1	1	1	1
1			1			1
1			1			1
1	1	4	2	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1

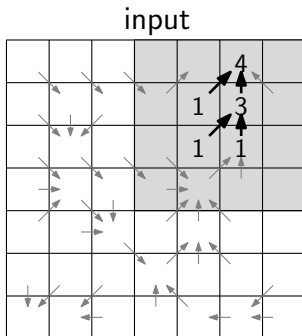
separator
flow accumul.



separator
flow directions

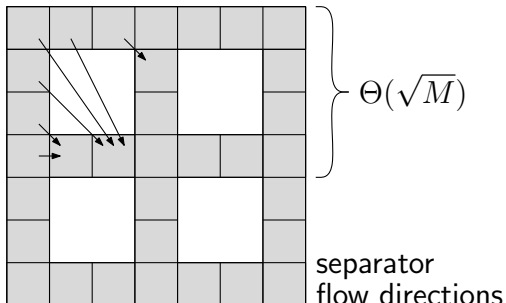
Cache-aware separator-based algorithm

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



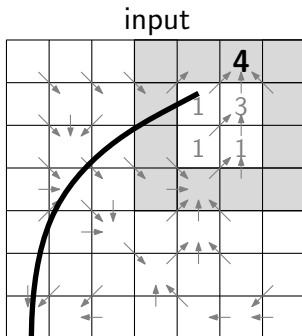
1	1	1	1	1	1	1
1			1			1
1			1			1
1	1	4	2	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1

separator
flow accumul.



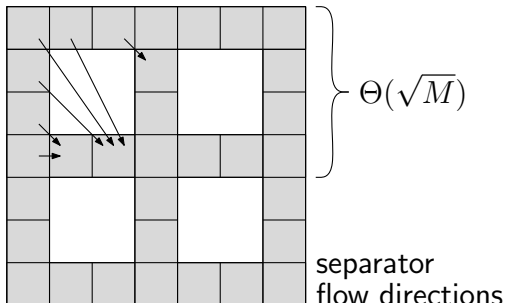
Cache-aware separator-based algorithm

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



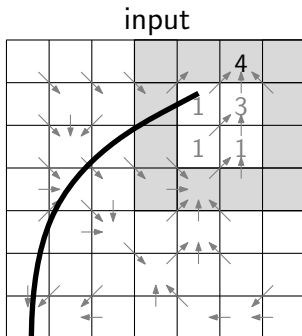
separator flow accumul.

1	1	1	1	1	1	1
1			1			1
1			1			1
1	1	4	2	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1



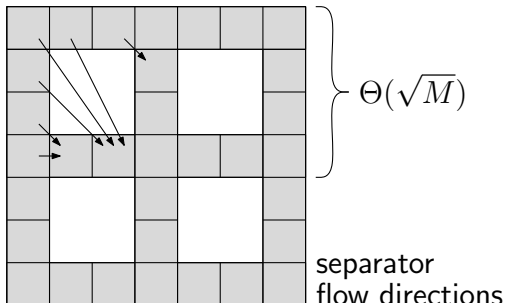
Cache-aware separator-based algorithm

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



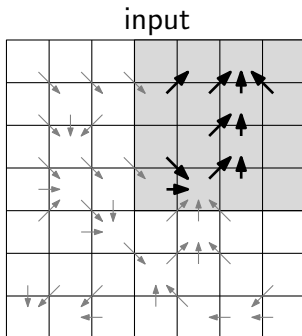
separator flow accumul.

1	1	1	1	1	5	1
1			1			1
1			1			1
1	1	4	2	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1



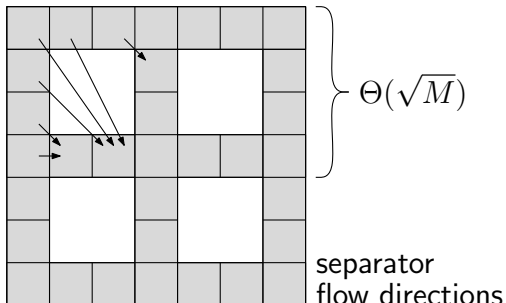
Cache-aware separator-based algorithm

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



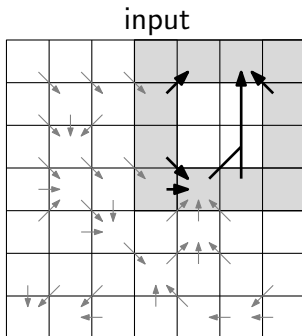
1	1	1	1	1	5	1
1			1			1
1			1			1
1	1	4	2	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1

separator
flow accumul.



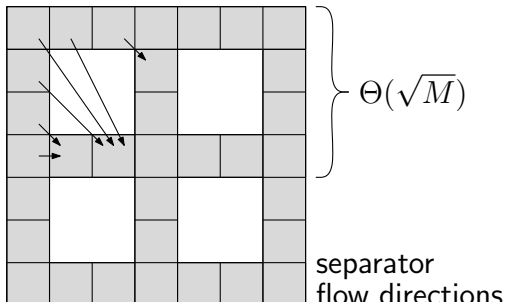
Cache-aware separator-based algorithm

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



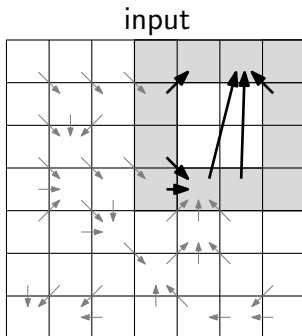
1	1	1	1	1	5	1
1			1			1
1			1			1
1	1	4	2	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1

separator
flow accumul.



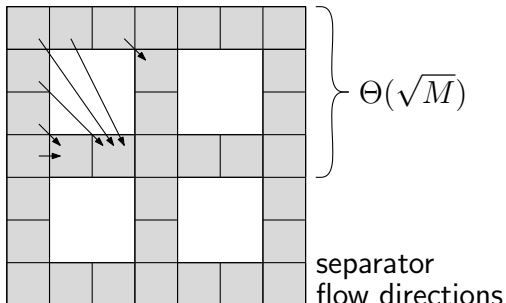
Cache-aware separator-based algorithm

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



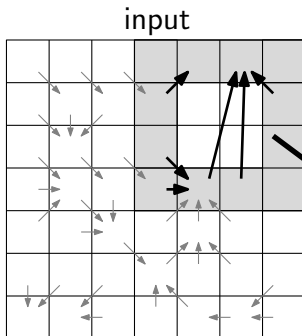
1	1	1	1	1	5	1
1			1			1
1			1			1
1	1	4	2	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1

separator
flow accumul.



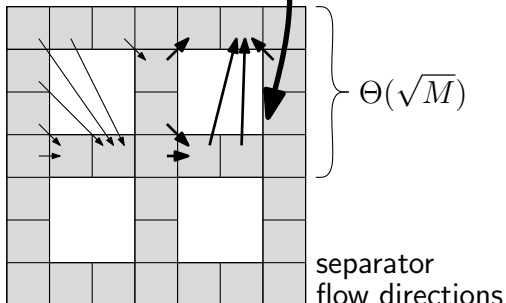
Cache-aware separator-based algorithm

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



1	1	1	1	1	5	1
1			1			1
1			1			1
1	1	4	2	1	1	1
1			1			1
1			1			1
1	1	1	1	1	1	1

separator
flow accumul.

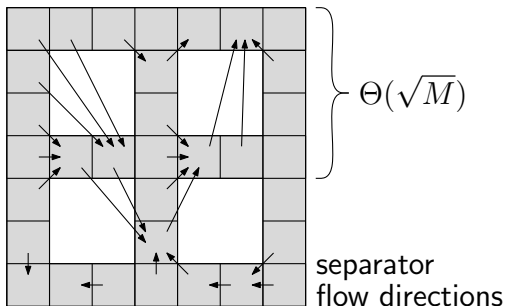


Cache-aware separator-based algorithm

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

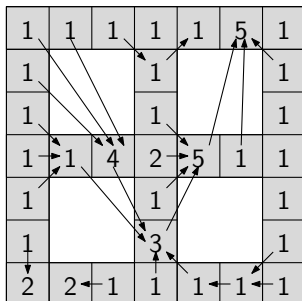
separator flow accumul.

1	1	1	1	1	5	1
1			1			1
1			1			1
1	1	4	2	5	1	1
1			1			1
1			3			1
2	2	1	1	1	1	1



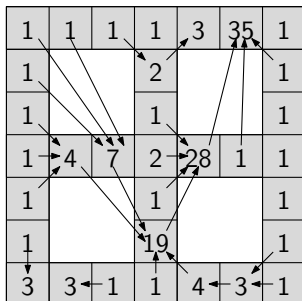
Cache-aware separator-based algorithm

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



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



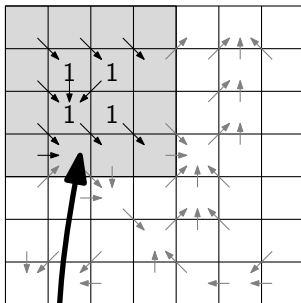
Cache-aware separator-based algorithm

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

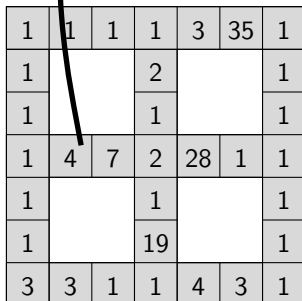
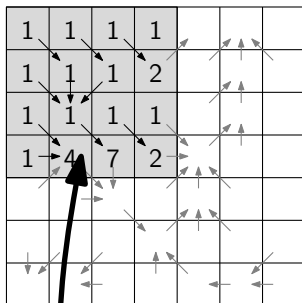
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids



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

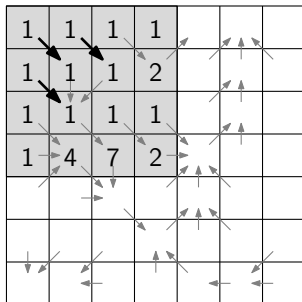
Cache-aware separator-based algorithm

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

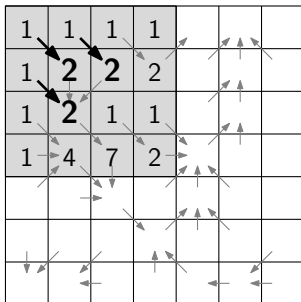
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids



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

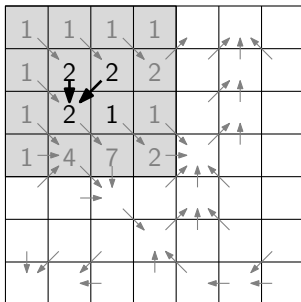
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids



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

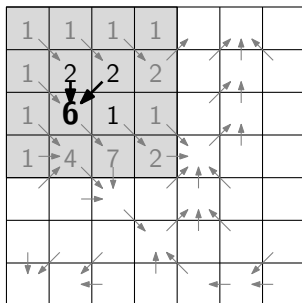
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids



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

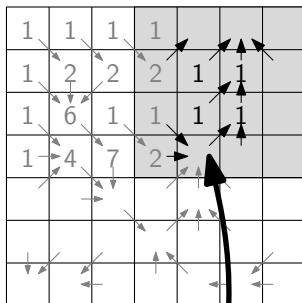
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids



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

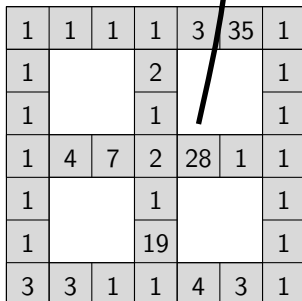
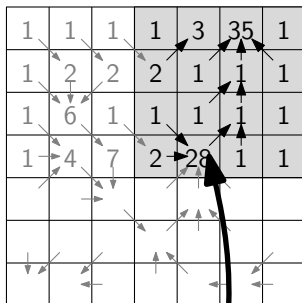
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids



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

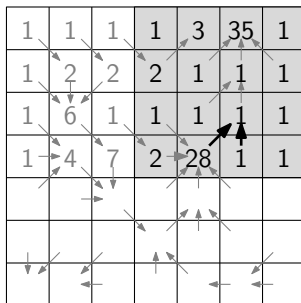
Cache-aware separator-based algorithm

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

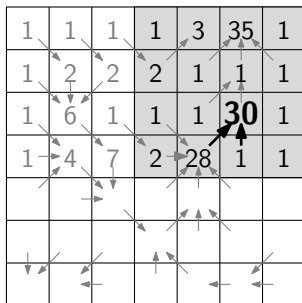
1. move flow from interior of subgrids to separators and compute flow connections between separators
2. compute flow accumulation of separators
3. move flow from separators into subgrids



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



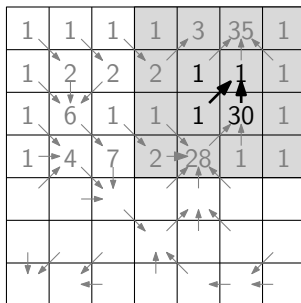
1	1	1	1	3	35	1
1			2			1
1			1			1
1	4	7	2	28	1	1
1			1			1
1			19			1
3	3	1	1	4	3	1

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

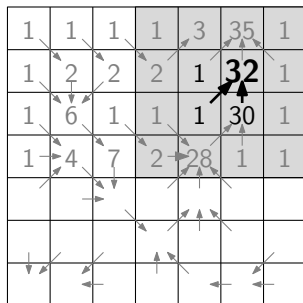
3. move flow from separators into subgrids



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

1	1	1	1	3	35	1
1	2	2	2	1	32	1
1	6	1	1	1	30	1
1	4	7	2	28	1	1
1	1	13	1	22	1	1
1	1	1	19	1	1	1
3	3	1	1	4	3	1

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

1. $\Theta(N/M)$ subgrids \times
 $\Theta(M/B + \sqrt{M}) =$
 $\Theta(M/B) =$
 $\Theta(\text{scan}(N))$ I/O's

1. 1 byte of I/O per cell

2. compute flow accumulation of separators

2. linear-time algo, input
 $\Theta(N/M) = \Theta(\text{scan}(N))$

2. no I/O in practice

3. move flow from separators into subgrids

3. $\Theta(\text{scan}(N))$ 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	$O(N)$	$O(\text{scan}(N))^*$	thousands	
Z-order scan	Z-order	$O(N/\sqrt{B})$	$O(\text{scan}(N))$	hundreds	41
\$-aware separ.	row by row	$O(\text{scan}(N))^*$		20 to 60	39
\$-aware separ.	Z-order	$O(\text{scan}(N))$		10	run this!
\$-obliv. separ.	row by row	$O(\text{scan}(N))^*$		>100	
\$-obliv. separ.	Z-order	$O(\text{scan}(N))^*$		>100	118
time-fwd proc.	any	$O(\text{sort}(N))$		70 to 300	sev. hundred
row \leftrightarrow Z-order		$O(\text{scan}(N))^*$		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?