Utrecht University

B3CC: Concurrency

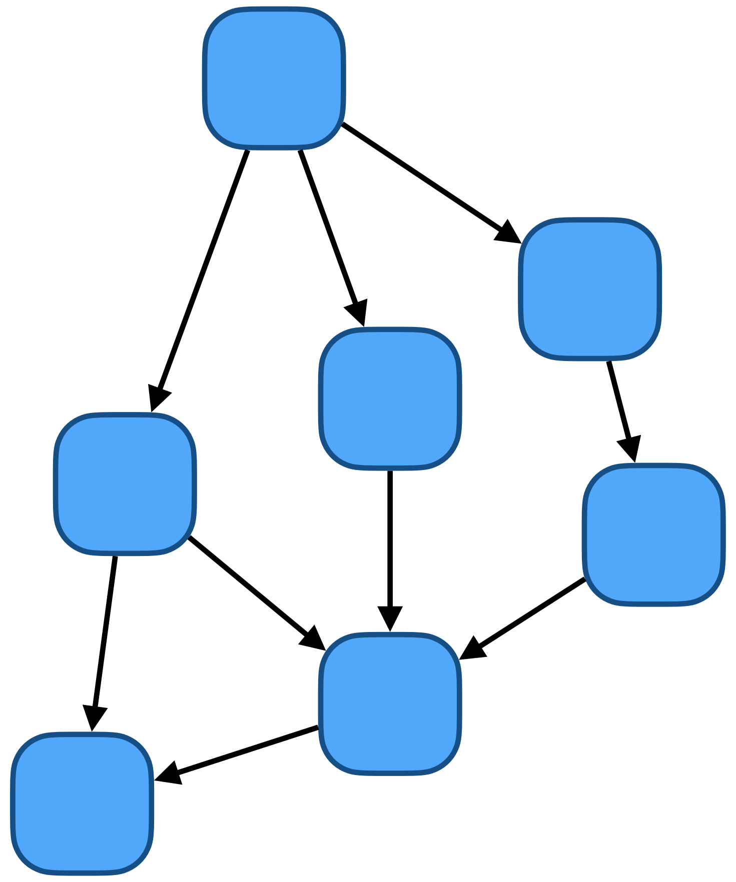
12: Data Parallelism (1)

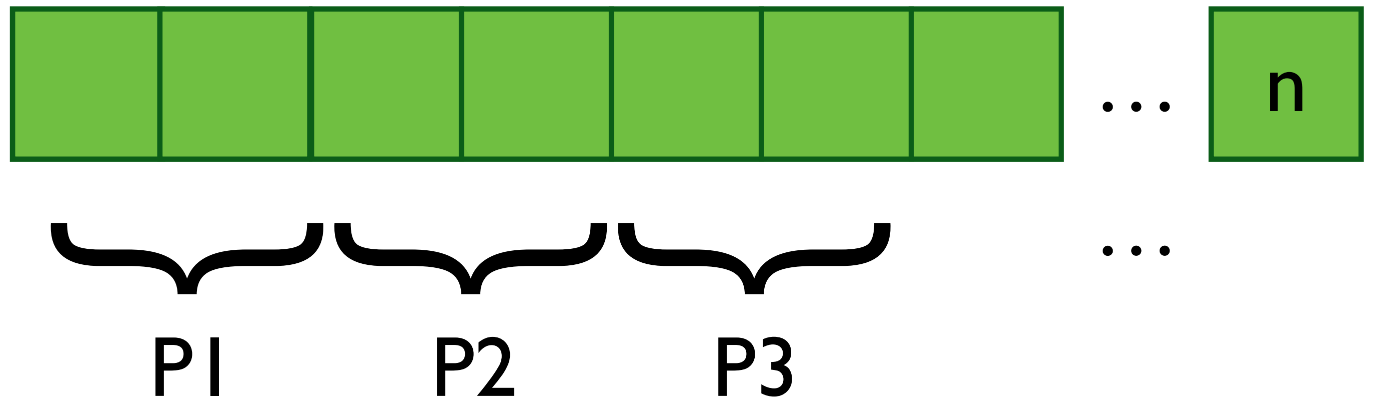
Ivo Gabe de Wolff

Recap

- **Concurrency**: dealing with lots of things at once
- **Parallelism:** doing lots of things at once
  - Processors are no longer getting faster: limitations in power consumption, memory speed, and instruction-level parallelism
  - Adding more processor cores has been the dominant method for improving processor performance for the last decade

Recap





Task parallelism

- Explicit threads
- Synchronise via locks, messages, or STM
- Modest parallelism
- Hard to program

Data parallelism

- Operate simultaneously on bulk data
- Implicit synchronisation
- Massive parallelism
- Easy to program

Goals

- Large applications use a mix of task- and data-parallelism
  - There is a difference in how to make use of 2-4 cores vs. 32+ cores
- In the application of parallelism, we would like to achieve:
  - *Performance*: ease of use, scalability, and predictability
  - *Productivity*: expressivity, correctness, clarity, and maintainability
  - *Portability*: between different machines, compilers, or architectures

Applications

- Games
  - Probably the primary consumer market for teraflop computing applications
- Image and video editing
- Scientific computing
  - Numeric simulations, modelling, etc.
- Machine learning

Patterns

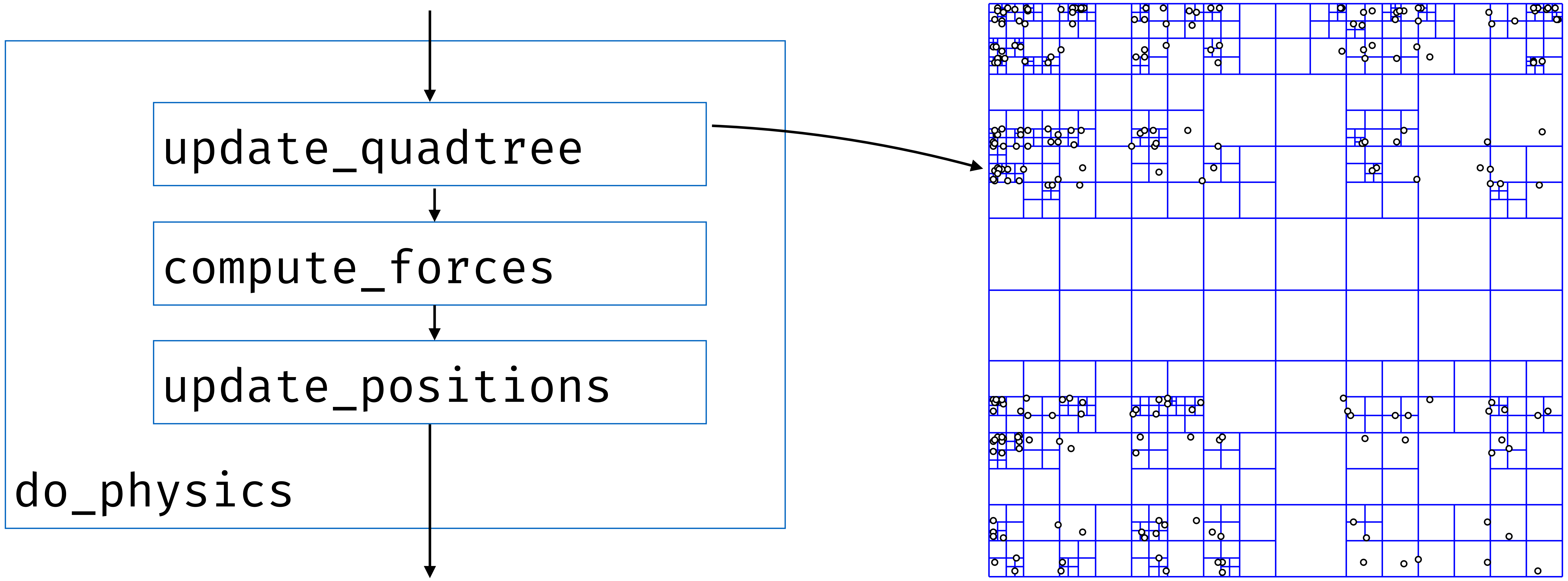
- Patterns, or *algorithmic skeletons*
  - A pattern is a recurring combination of task distribution and data access
  - Patterns provide a vocabulary for [parallel] algorithm design
  - These ideas are not tied to a particular hardware architecture
- This distinguishes two important aspects:
  - Semantics: what we want to achieve
  - Implementation: how to achieve this on a given architecture

Patterns

- Patterns also exist in serial code
  - We often don't think of serial code in this way, however it helps to name these patterns in order to talk about these ideas in a parallel context
  - Compositional patterns: nesting
  - Control-flow patterns: sequence, selection, repetition, and recursion

Patterns: nesting

- Nesting simply refers to the ability to hierarchically compose patterns
  - Including recursive functions



`update_quadtree`

`compute_forces`

`update_positions`

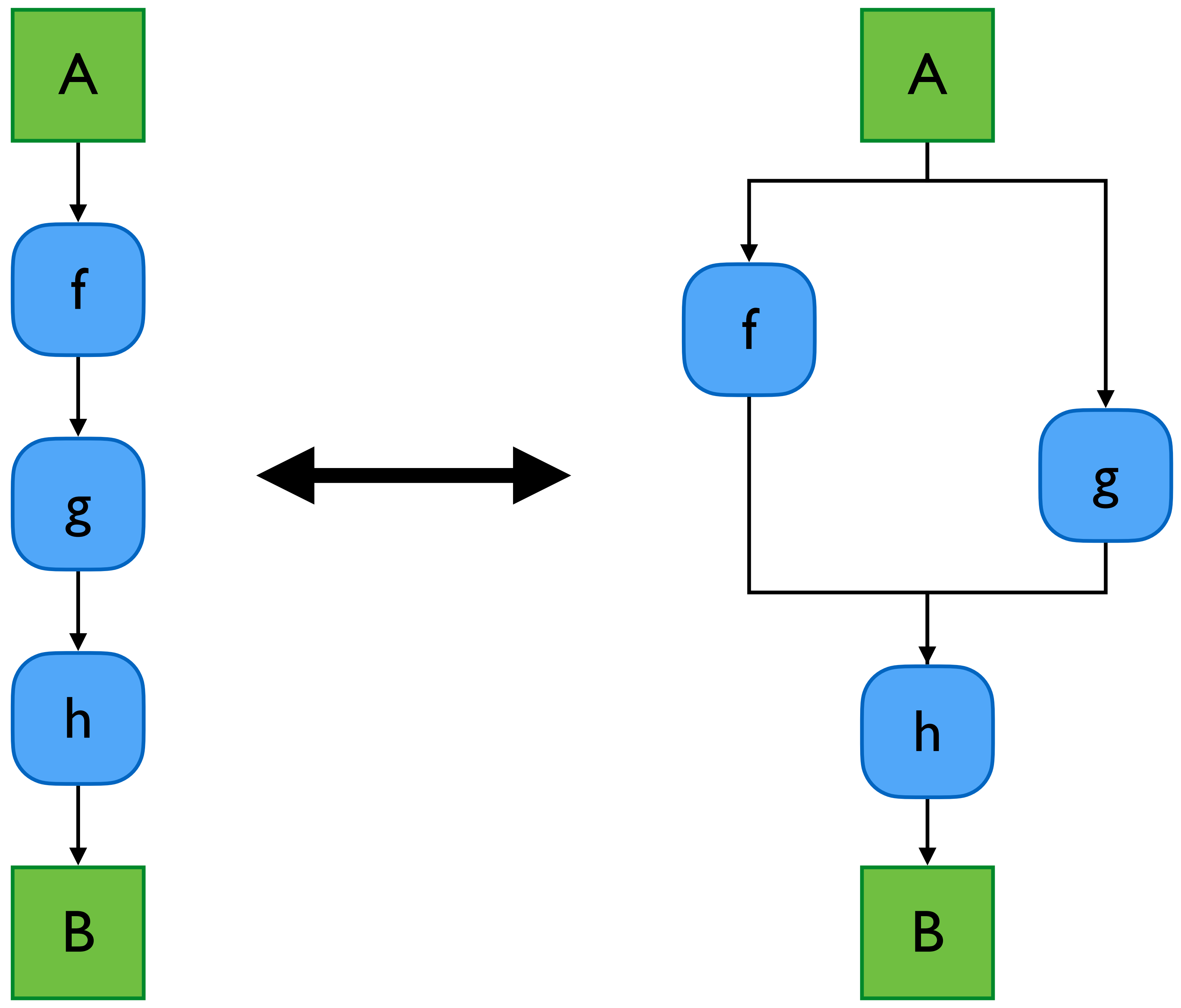
`do_physics`

recursive

Patterns: sequence

- Tasks executed in a specified order
  - We generally assume that the program is executed in the text order
  - Modern CPUs violate this (out-of-order execution (instructions & memory))
  - Programmer or language specifies if/how memory operations may be reordered (memory_order in c++)

```
f = a * a
g = a + a
h = f - g
```

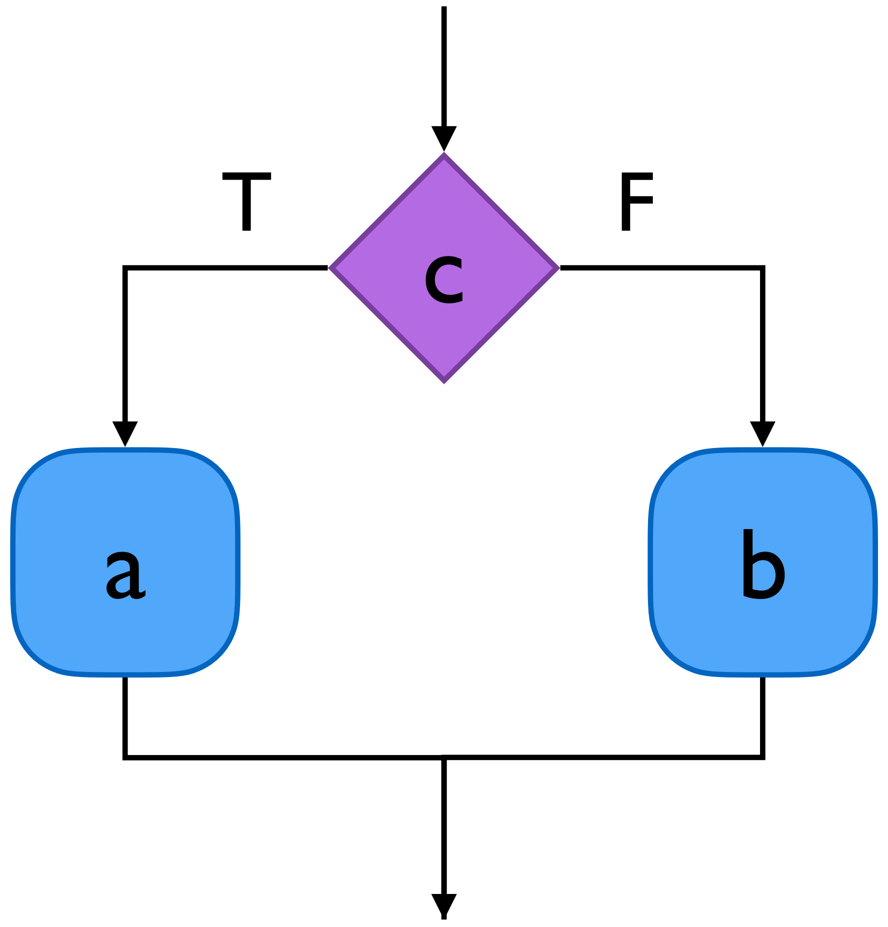


https://en.wikipedia.org/wiki/Out-of-order_execution
https://en.wikipedia.org/wiki/Memory_ordering#Runtime_memory_ordering
https://en.cppreference.com/w/cpp/atomic/memory_order

Patterns: selection

- Conditionals are pervasive in serial code
  - On average one branch every five instructions
  - Modern CPUs speculatively execute (far) ahead of when `c` is known
  - TensorFlow (google deep learning framework) always evaluates both branches of conditionals

```
if (c) {
  a;
} else {
  b;
}
```

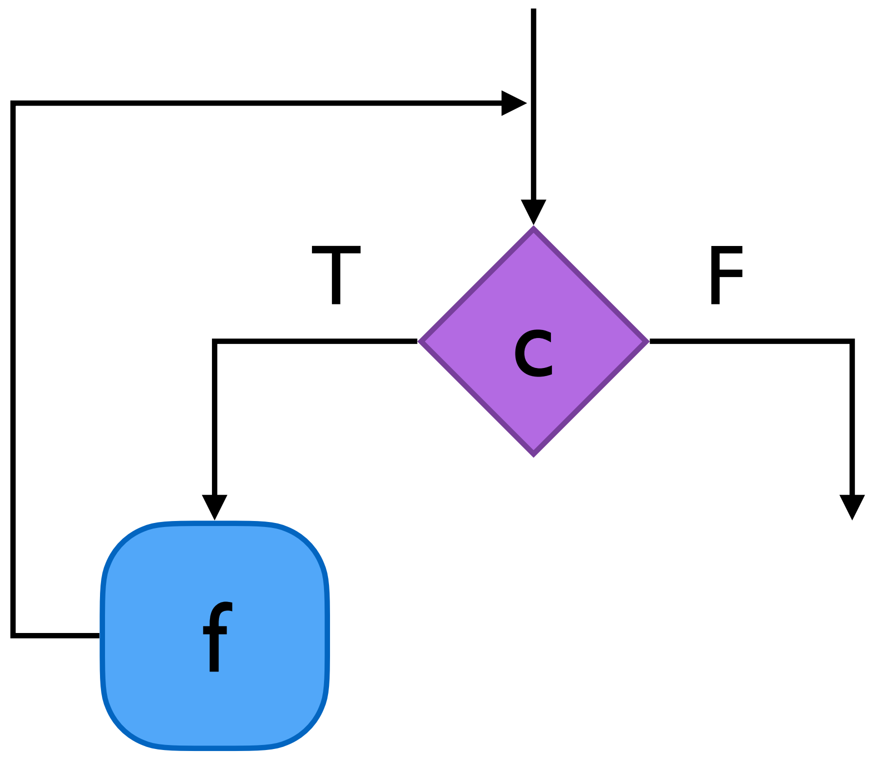


https://en.wikipedia.org/wiki/Speculative_execution

Patterns: iteration

- Continually execute a task while some condition is true
    - Parallelisation of loops is complicated due to *loop-carried dependencies*
    - There is a lot of research in this area (polyhedral models, loop nests)
    - Instead, several *parallel* patterns exist for specific loop forms: map, reduce, scan, scatter, gather…

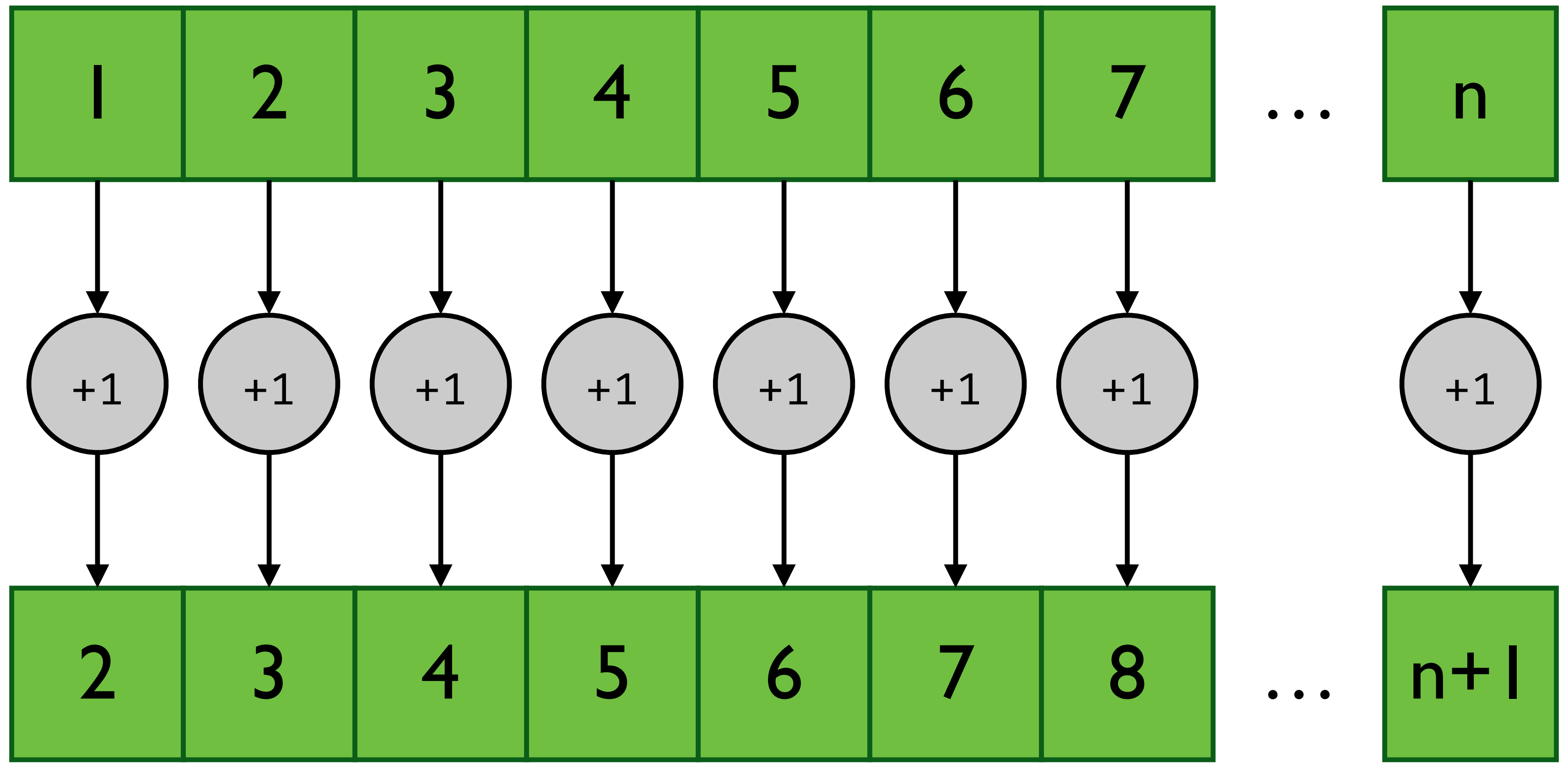
```
while (c) {
  f;
}
```



https://en.wikipedia.org/wiki/Polytope_model

Map

- The `map` operation applies the *same* function to each element of a set
    - This is a parallelisation of a loop with a *fixed* number of iterations
    - There must not be any dependencies between loop iterations: the function uses only the input element value and/or index

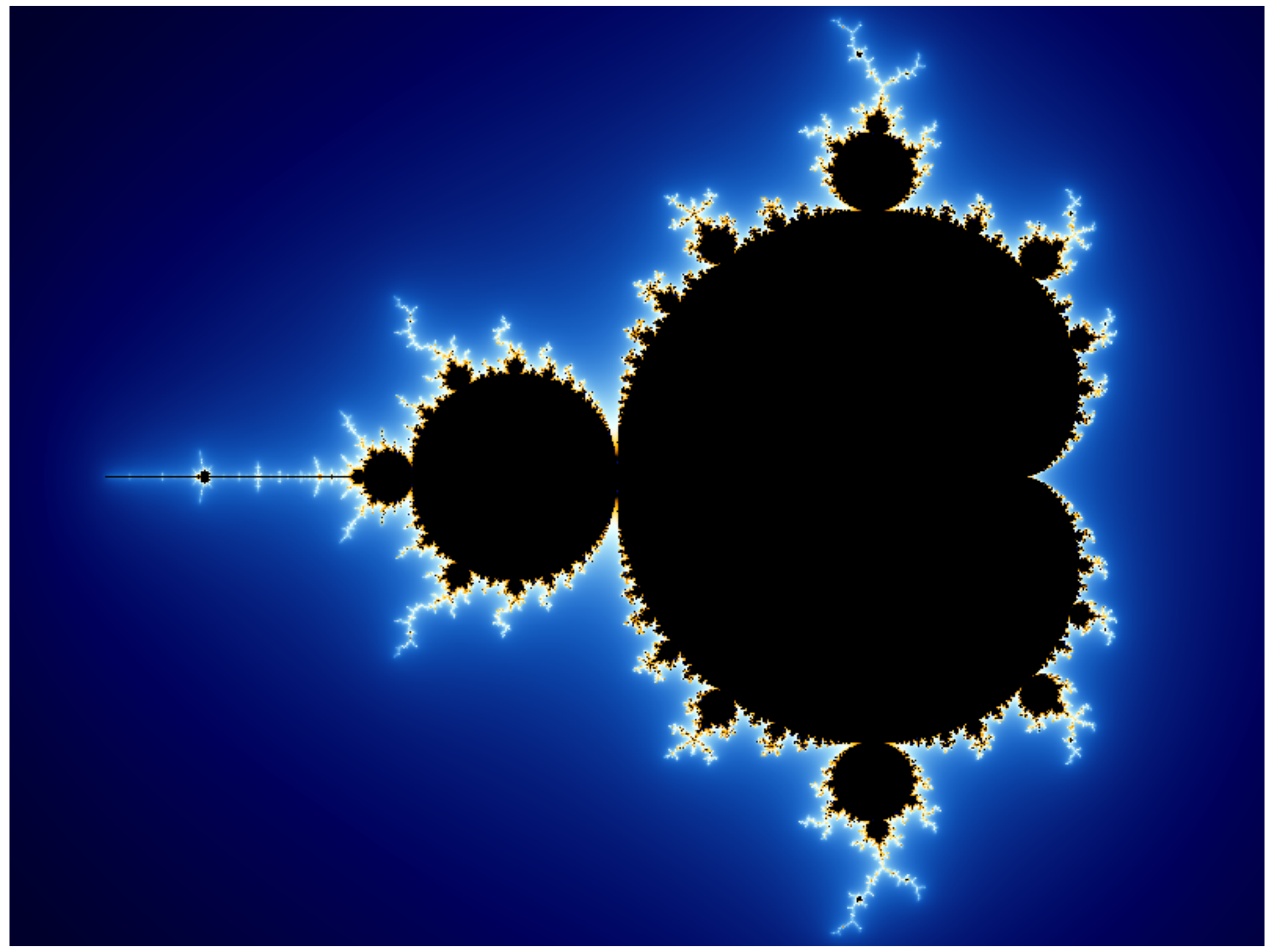


```
for (i = 0; i < len; ++i)
{
    x = xs[i];
    y = f(x);
    ys[i] = y;
}
```

https://hackage.haskell.org/package/accelerate-1.3.0.0/docs/Data-Array-Accelerate.html#g:18

Map

- The map operation applies the *same* function to each element of a set
  - The function only has access to a *single value*
  - The number of iterations is dynamic (e.g. size of the array) but fixed at the start of the `map`: does not vary based on the loop body
  - Note that the order of operations is not specified



$$z_{n+1} = z_n^2 + c$$

Map

- The map operation applies the *same* function to each element of a set
  - On the GPU this corresponds to one thread per element
  - Number of loop iterations is controlled by how many threads the kernel is launched with
  - Host code:

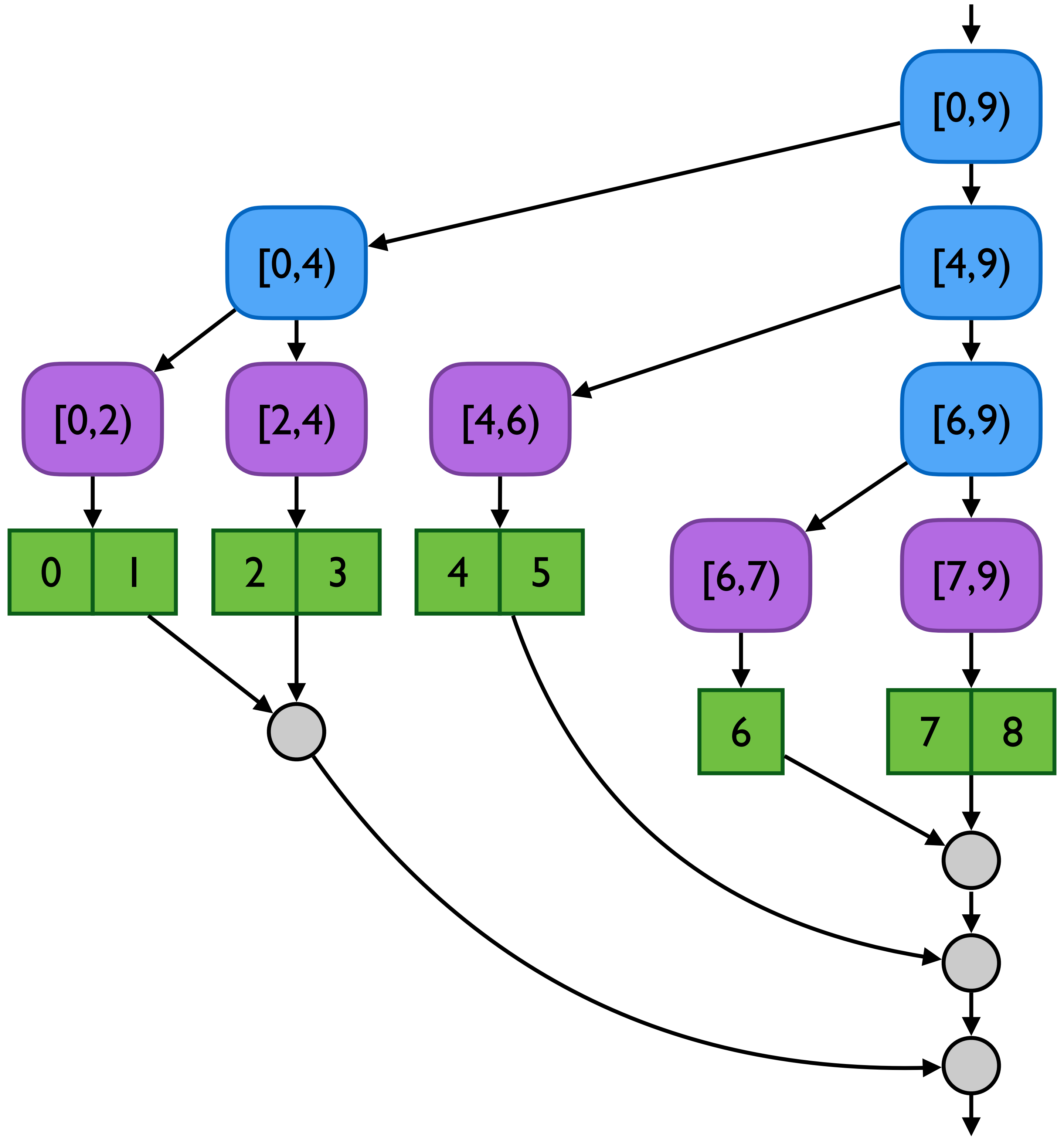
```
map<<<4, 1024>>>( h_xs, h_ys, 4000 );
```

  - GPU code:

```
__global__ void map( float* d_xs, float* d_ys, int len )
{
   int i = blockDim.x * blockIdx.x + threadIdx.x;
   if ( i < len ) {
      d_ys[i] = f ( d_xs[i] );
   }
}
```

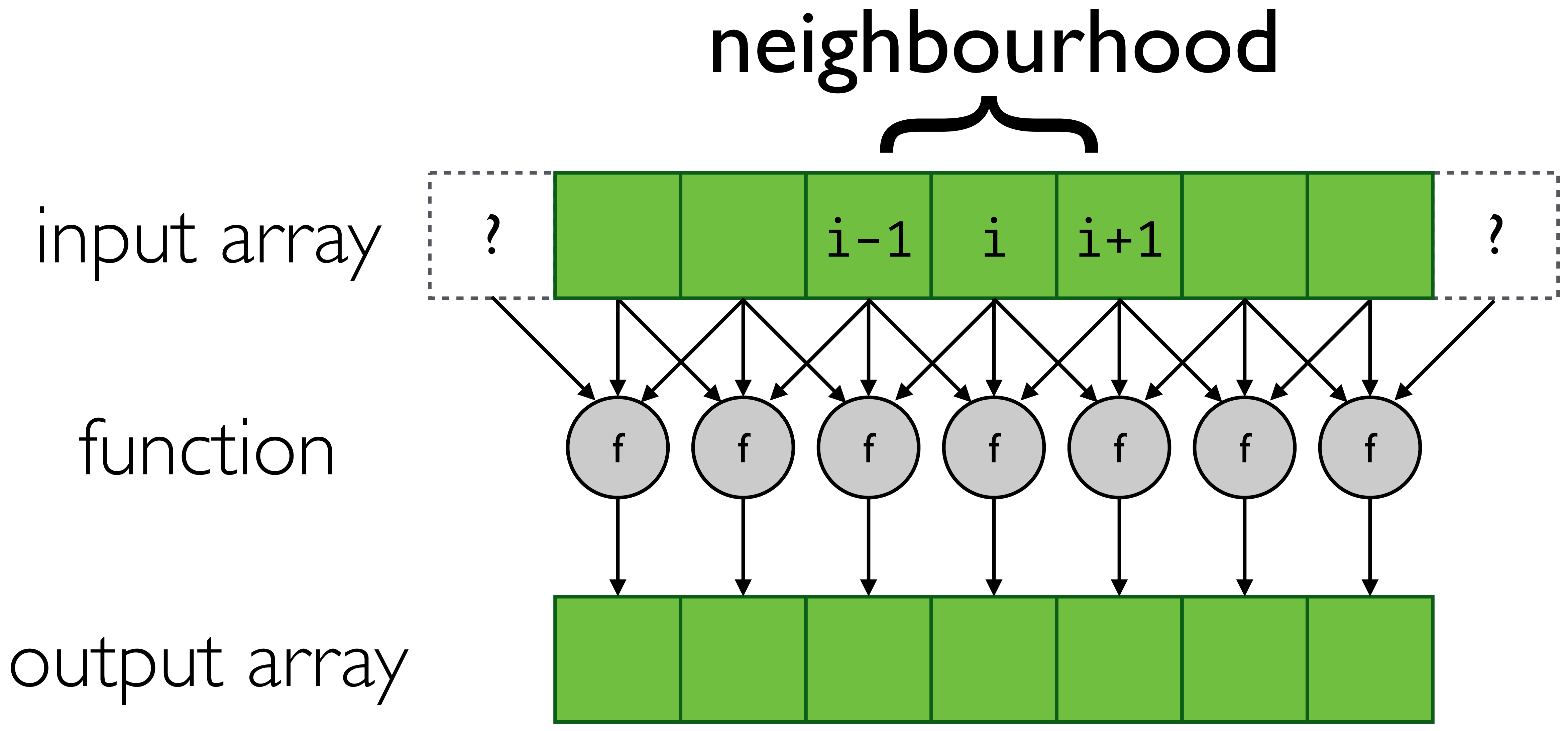
Map

- In the graphics pipeline, vertex and fragment shaders are a parallel map
  - Each shader outputs a single pixel or vertex; no other side effects
  - Shaders are also examples of *streaming algorithms*: data is used exactly once, so no caching is necessary
- On the CPU, can be implemented via
  - Static schedule (like count & list mode of IBAN)
  - fork-join
  - divide-and-conquer (like search mode of IBAN)
  - …



Stencil

- A `map` with access to the neighbourhood around each element
  - The set of neighbours is fixed, and relative to the element
  - Ubiquitous in scientific, engineering, and image processing algorithms



https://hackage.haskell.org/package/accelerate-1.3.0.0/docs/Data-Array-Accelerate.html#g:37

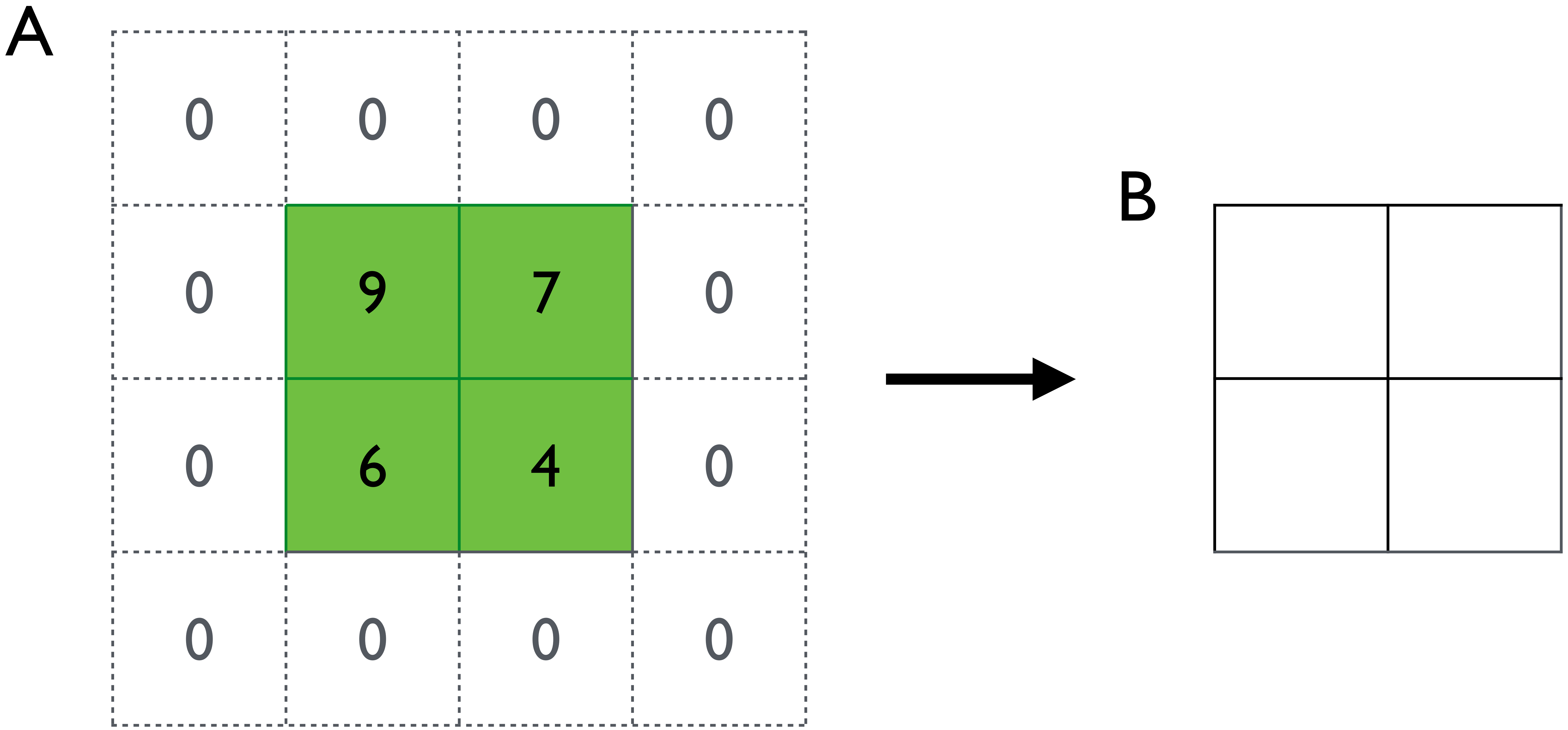
Stencil

- The stencil pattern
  - The set of neighbouring elements used by the stencil function
  - The shape of the stencil pattern can be anything: sparse, non-symmetric, etc.
  - The pattern of the stencil determines how the stencil operates in an application

```
__global__ void stencil( float* xs, float* ys, int width, int height )
{
   int i = blockDim.x * blockIdx.x + threadIdx.x;
   ys[i] = f (          xs[i-width]
             , xs[i-1], xs[i],       xs[i+1]
             ,          xs[i+width]
             );
}
```

Example

- Apply a stencil operation to the inner square
  - Treat out-of-bounds elements are zero (we'll come back to this later)



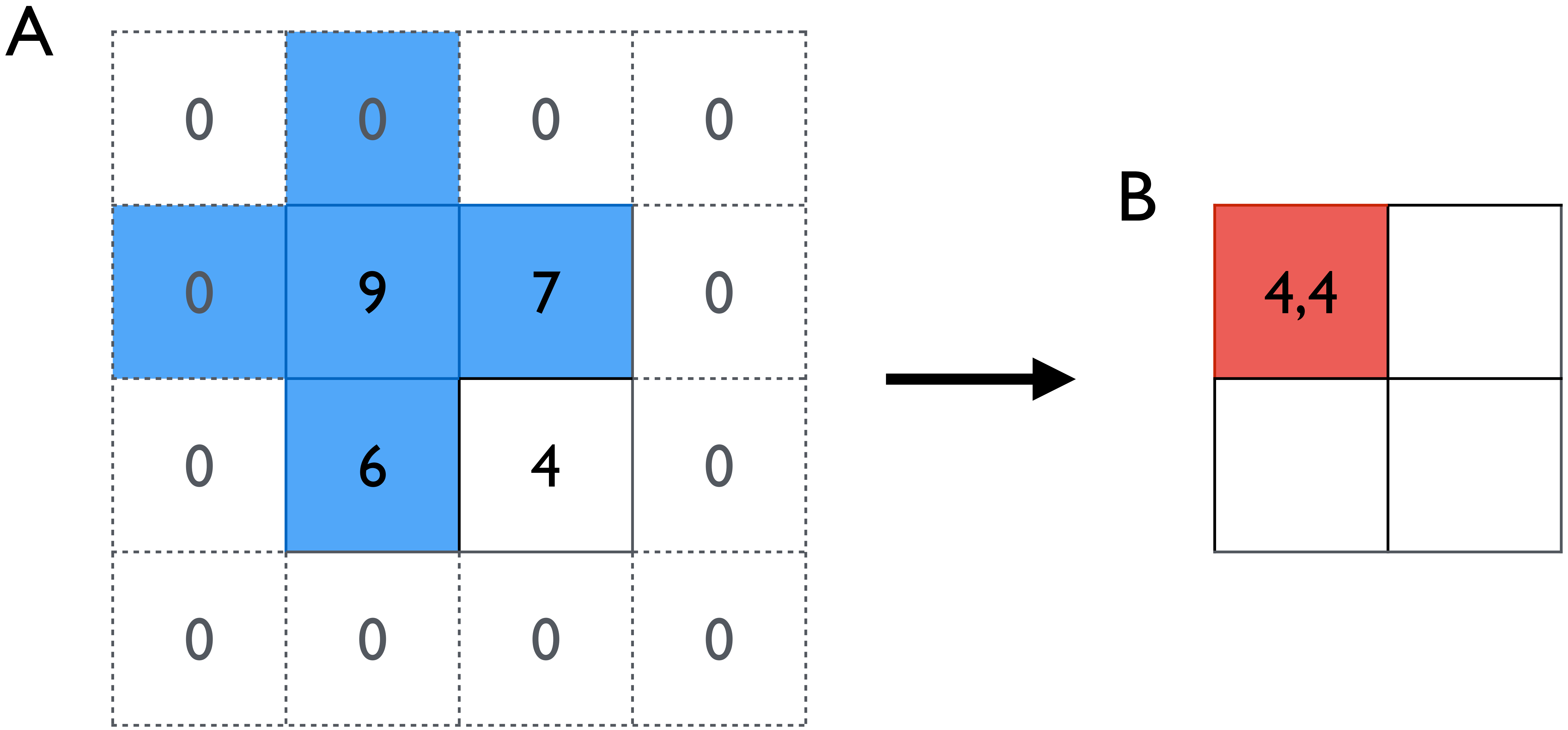
A

0	0	0	0
0	9	7	0
0	6	4	0
0	0	0	0

B

Example

- Apply a stencil operation to the inner square
  - Treat out-of-bounds elements are zero
  - Stencil function: average of the blue squares

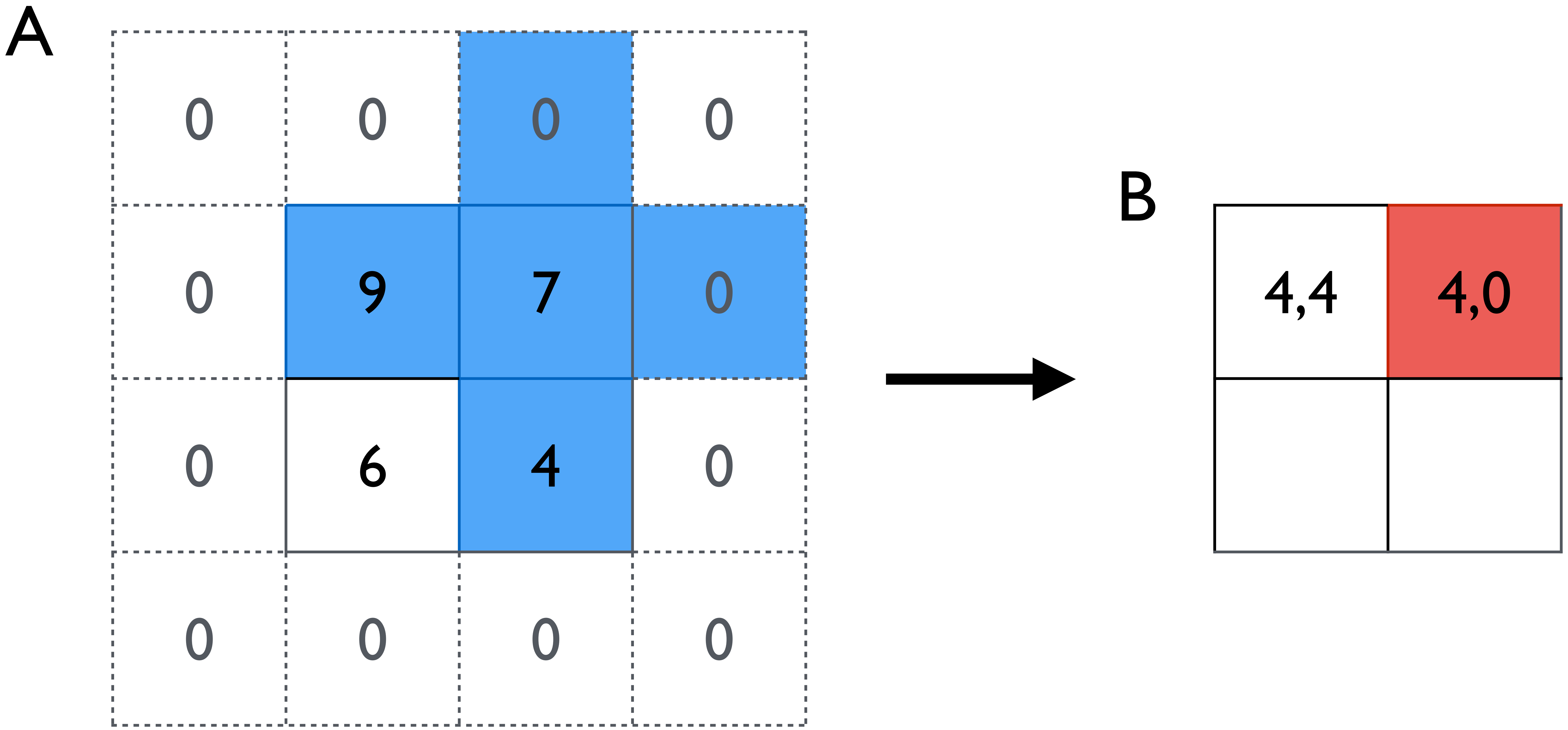


0	0	0	0
0	9	7	0
0	6	4	0
0	0	0	0

4,4	

Example

- Apply a stencil operation to the inner square
  - Treat out-of-bounds elements are zero
  - Stencil function: average of the blue squares



A

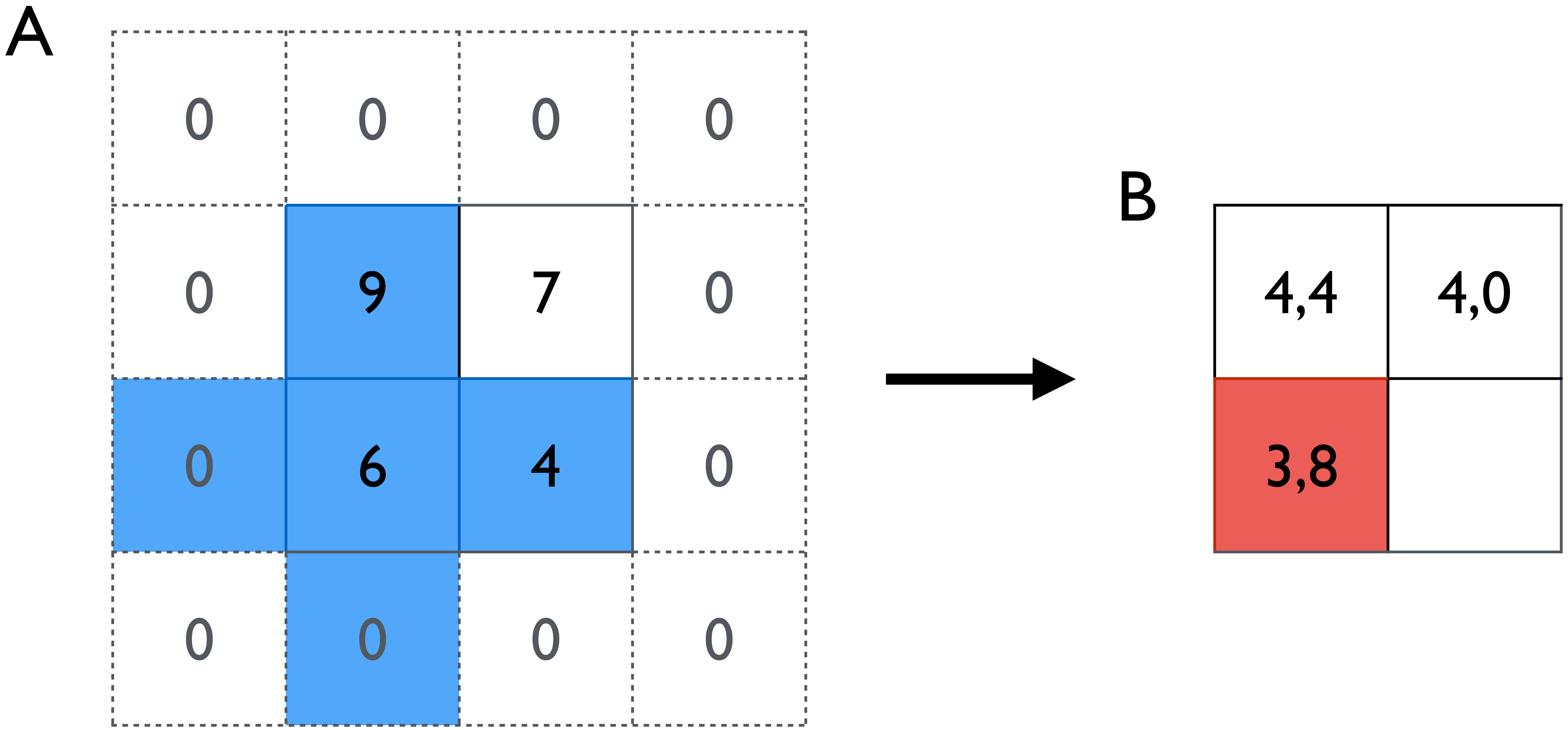
0	0	0	0
0	9	7	0
0	6	4	0
0	0	0	0

B

4,4	4,0

Example

- Apply a stencil operation to the inner square
  - Treat out-of-bounds elements are zero
  - Stencil function: average of the blue squares



A

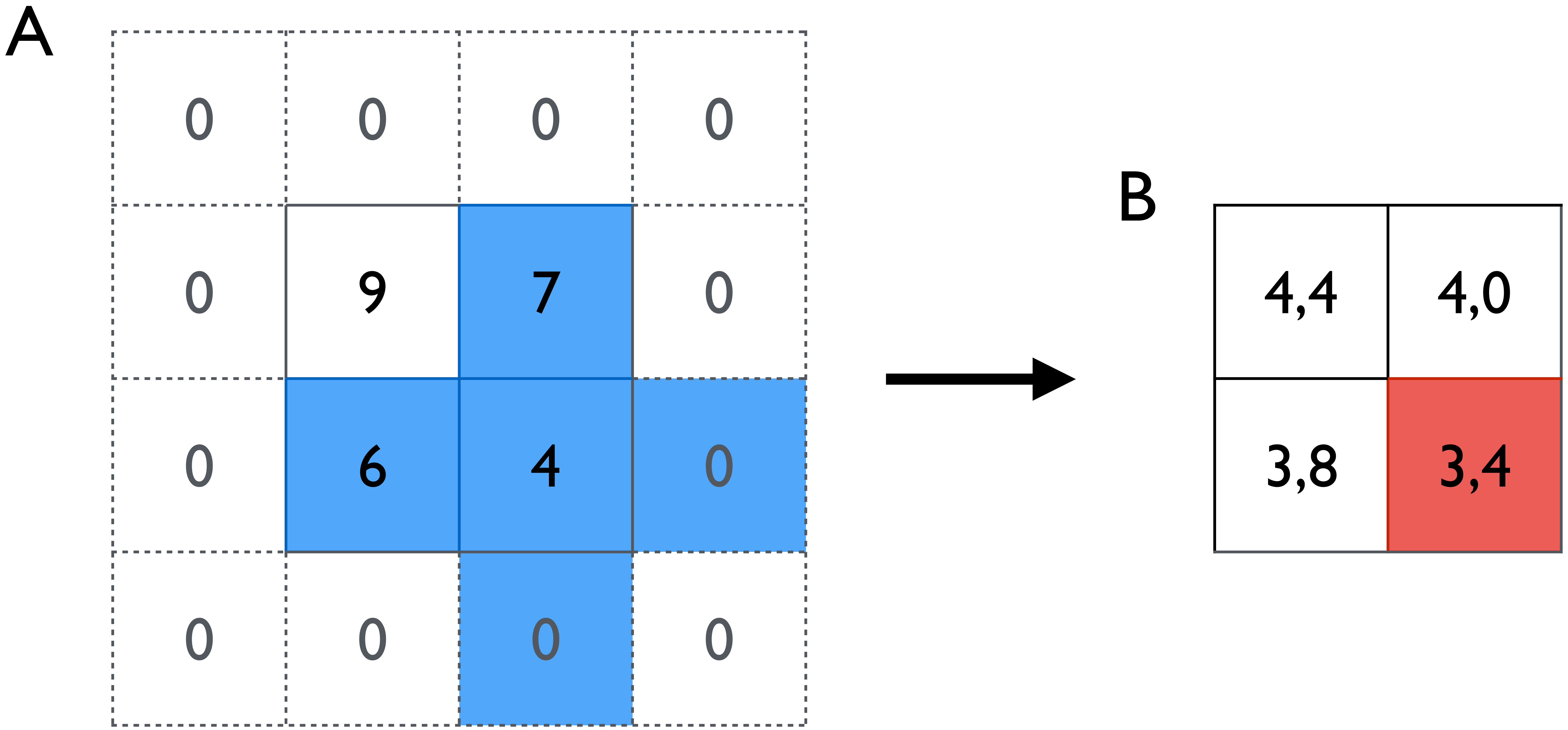
0	0	0	0
0	9	7	0
0	6	4	0
0	0	0	0

B

4,4	4,0
3,8	

Example

- Apply a stencil operation to the inner square
  - Treat out-of-bounds elements are zero
  - Stencil function: average of the blue squares



A

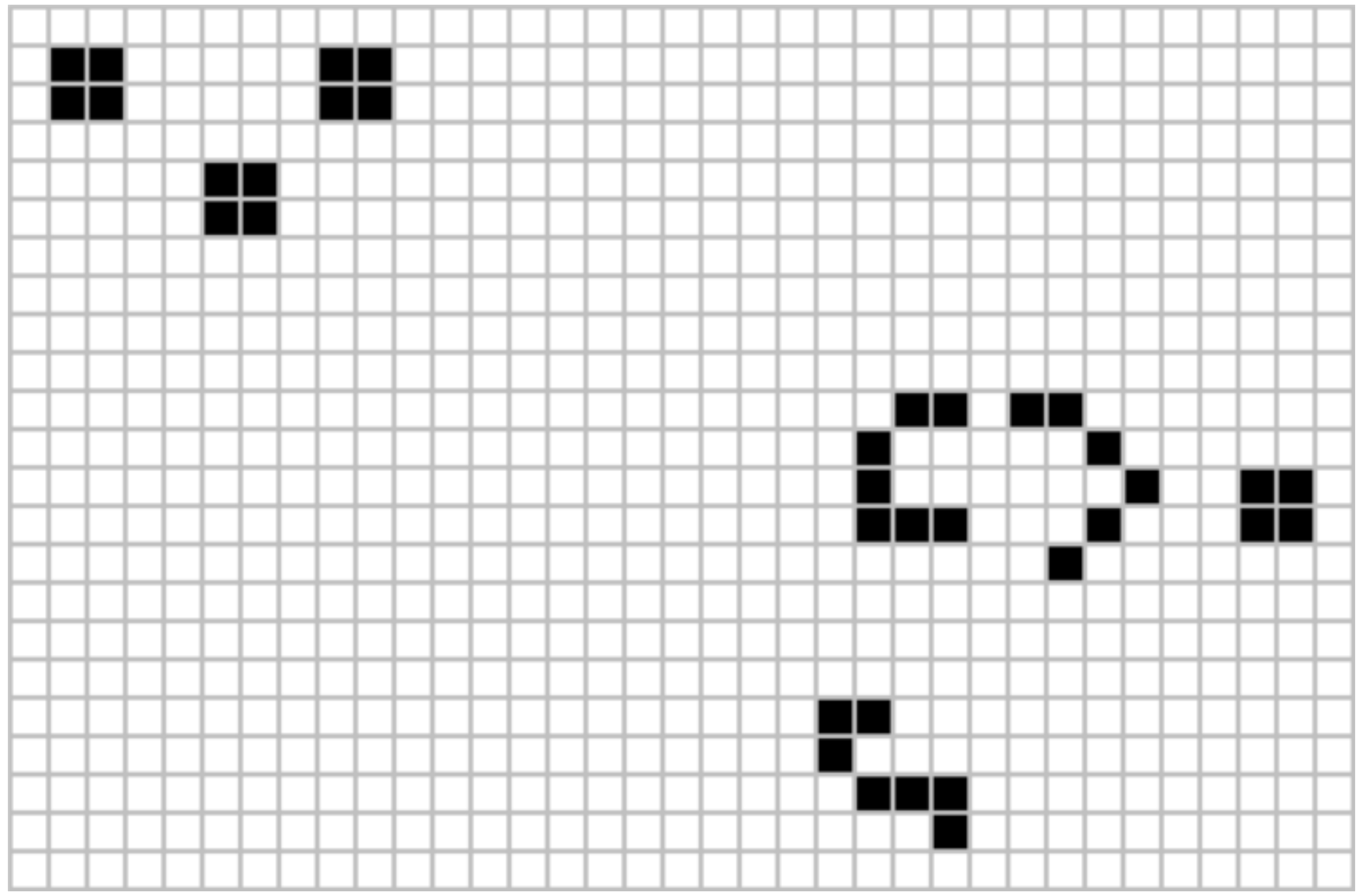
0	0	0	0
0	9	7	0
0	6	4	0
0	0	0	0

B

4,4	4,0
3,8	3,4

Example: Conway's game of life

- Cellula automaton developed in 1970
  - Evolution of the system is determined from an initial state
  - Cells live or die based on the population of their surrounding neighbours
  - Turing complete!

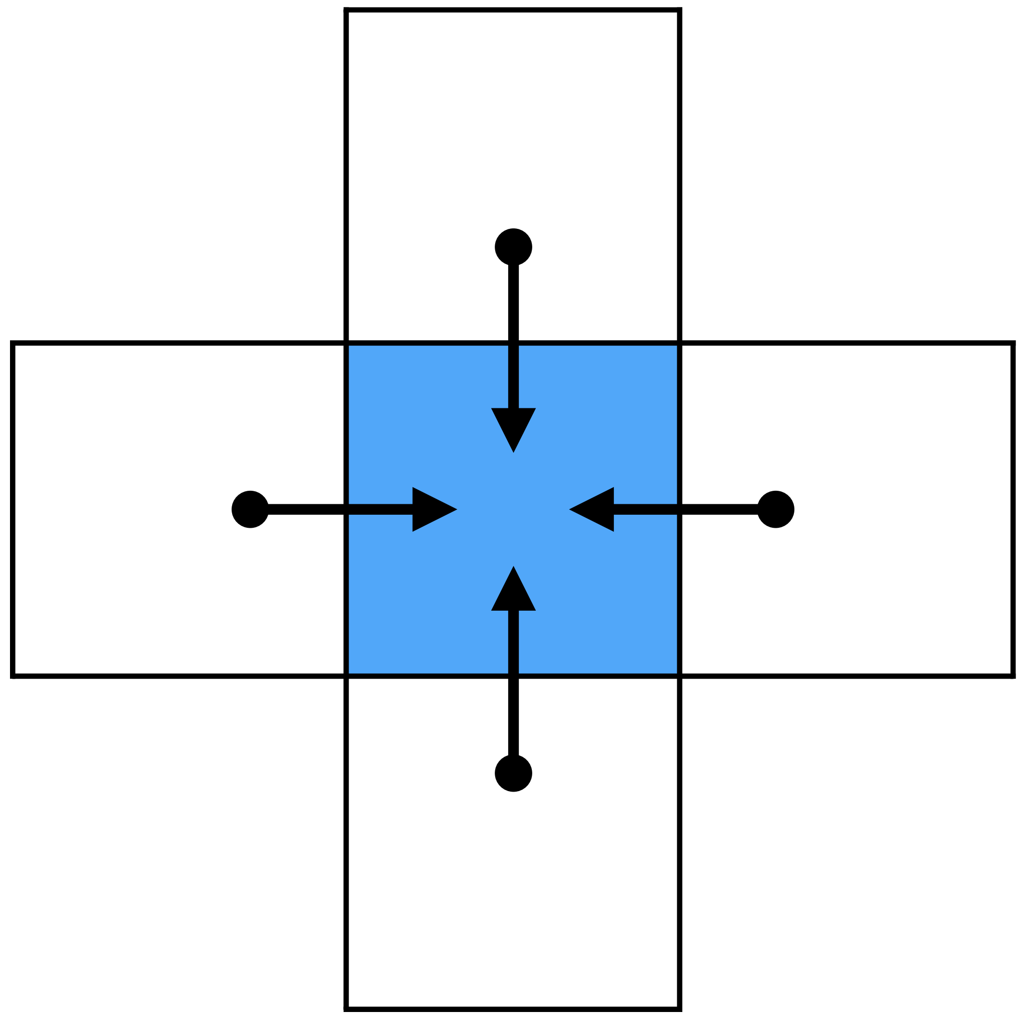


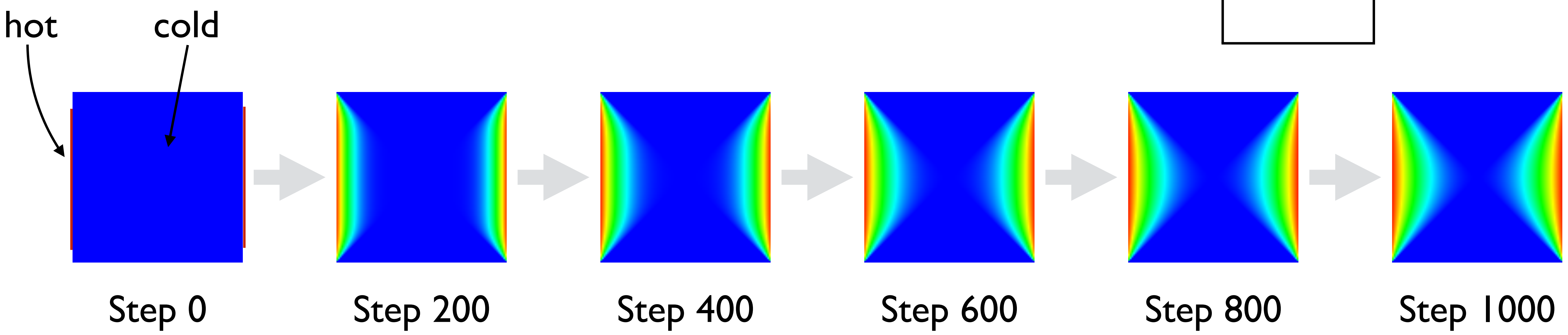
https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
https://github.com/tmcdonell/gameoflife-accelerate

Example: heat equation

- Iterative codes are ones that update their data in steps
- Most commonly found in simulations for scientific & engineering applications
  - Often used to solve partial differential equations

$$\begin{aligned}\nabla^2 u &= 0 \\ &= \frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}}{4}\end{aligned}$$





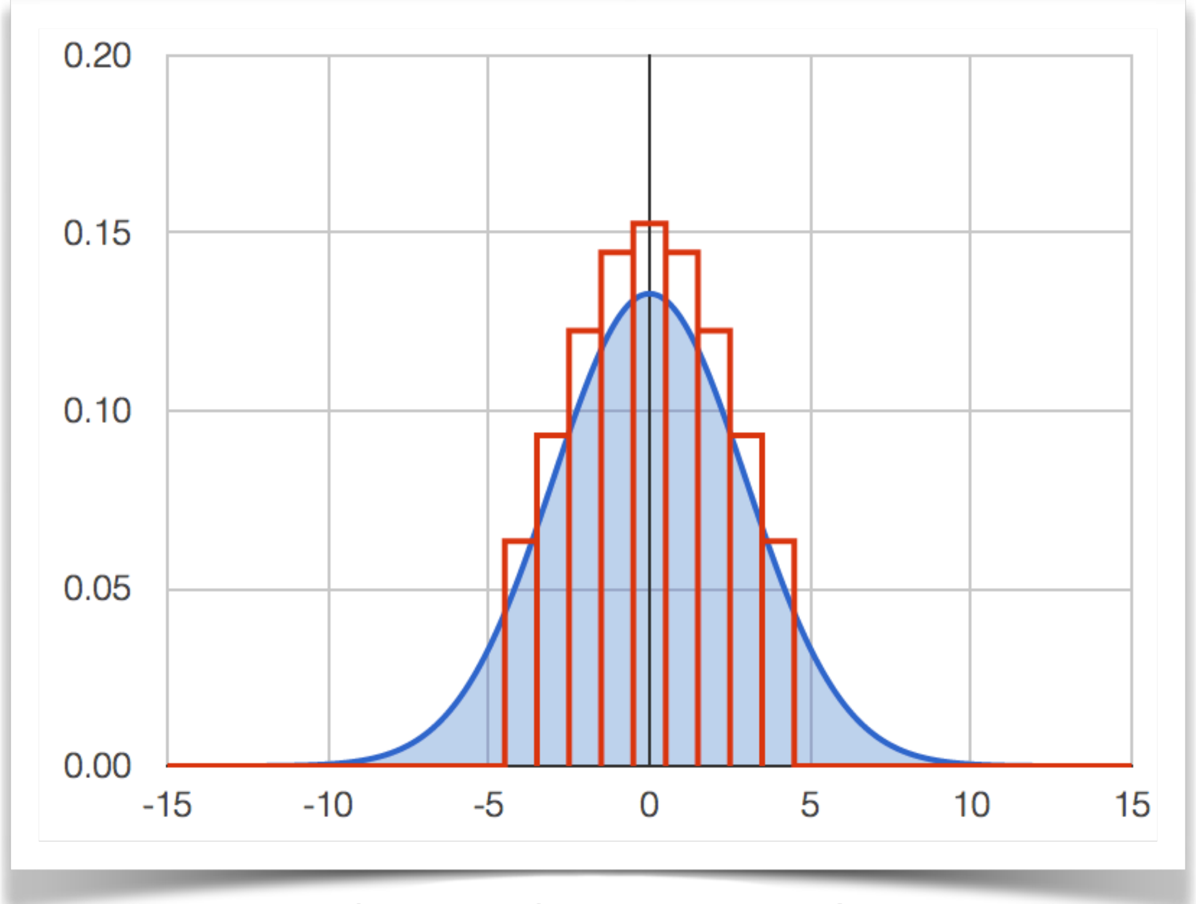
https://en.wikipedia.org/wiki/Stencil_code
https://github.com/tmcdonell/fisher-accelerate

Example: gaussian blur

- Convolution with a Gaussian function
  - Typically used to reduce image noise
  - Each pixel becomes the weighted sum of the surrounding pixels

$$(I \otimes K)(x, y) = \sum_i \sum_j I(x + i, y + j) K(i, j)$$

$\otimes$



$9 \times 9, \sigma = 3$

$=$

https://en.wikipedia.org/wiki/Gaussian_blur
https://github.com/tmcdonell/accelerate-examples/blob/master/examples/canny/src-acc/Canny.hs#L82

Example: gaussian blur

- Gaussian function
  - This is a *separable* convolution:
    instead of a single $n \times n$ stencil,
    it can be implemented as an $1 \times n$ stencil after a $n \times 1$ stencil
  - This is significant for large n
  - Example: 3×3 stencil

$$\begin{bmatrix} 0.077847 & 0.123317 & 0.077847 \\ 0.123317 & 0.195346 & 0.123317 \\ 0.077847 & 0.123317 & 0.077847 \end{bmatrix} = \begin{bmatrix} 0.27901 \\ 0.44198 \\ 0.27901 \end{bmatrix} \times \begin{bmatrix} 0.27901 & 0.44198 & 0.27901 \end{bmatrix}$$

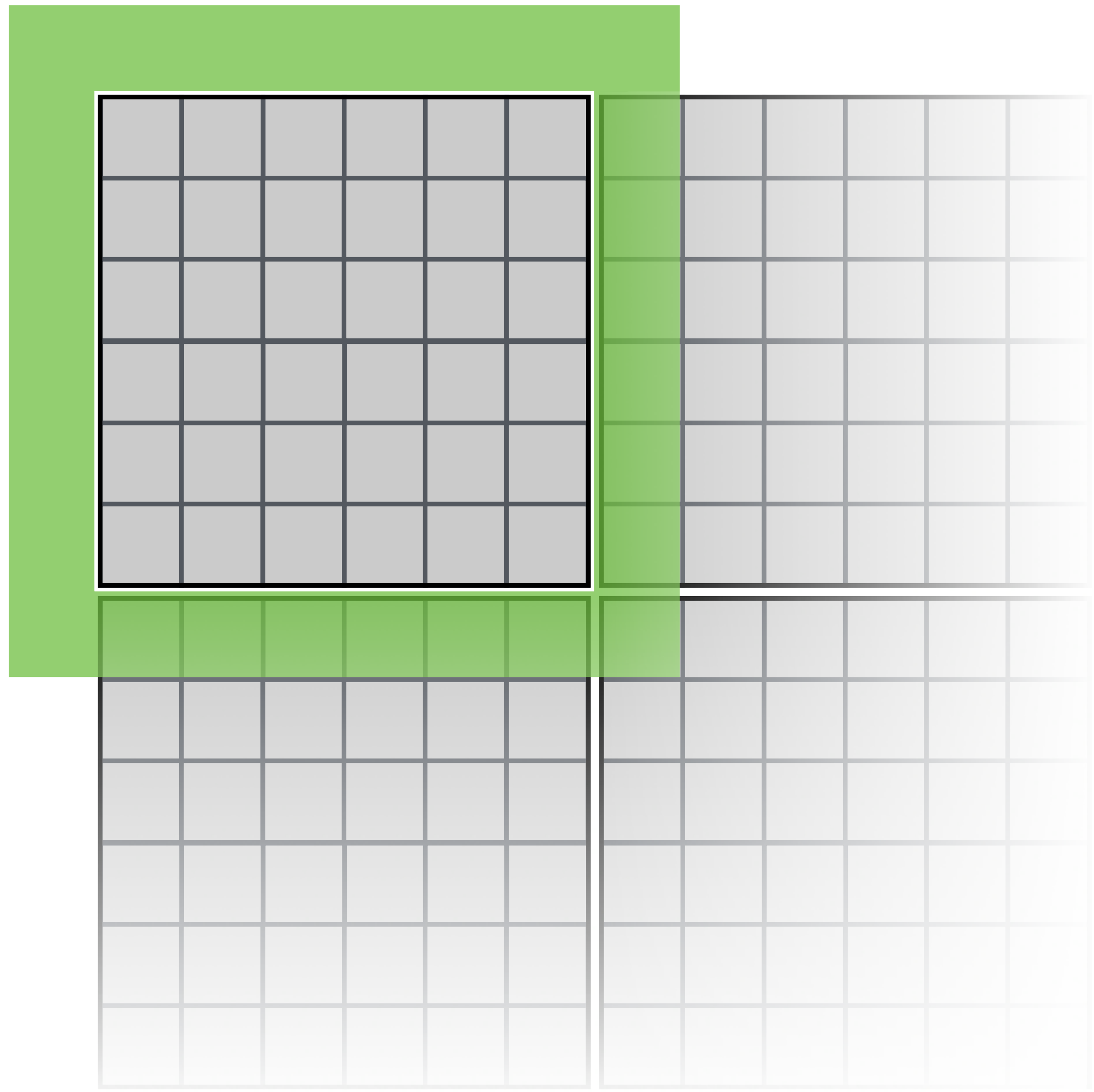
http://dev.theomader.com/gaussian-kernel-calculator

Stencil boundary

- What to do when the stencil pattern falls outside the bounds of the array?
  - At the edges of a simulation, we may need to impose boundary conditions
    - choose a fixed value or derivative (e.g. to impose symmetry)
    - many options are possible…
- What about between processors?

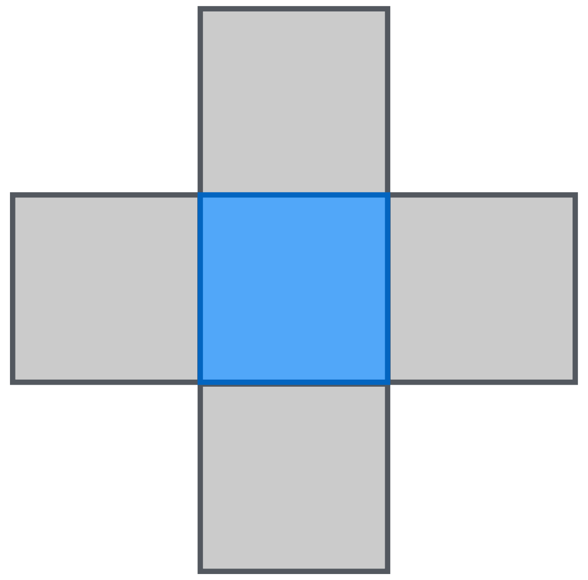
Stencil boundary

- What happens at the boundary of the computation?
  - Each larger box is owned by a thread / processor
- *Ghost cells* are one solution to the boundary and update issues of a stencil computation
  - Each thread keeps a copy of the neighbour's data to use in local computations
  - The ghost cells must be updated after each iteration of the stencil
  - The set of ghost cells is called the *halo*
    - A deeper halo can be used to reduce communication for some redundant work

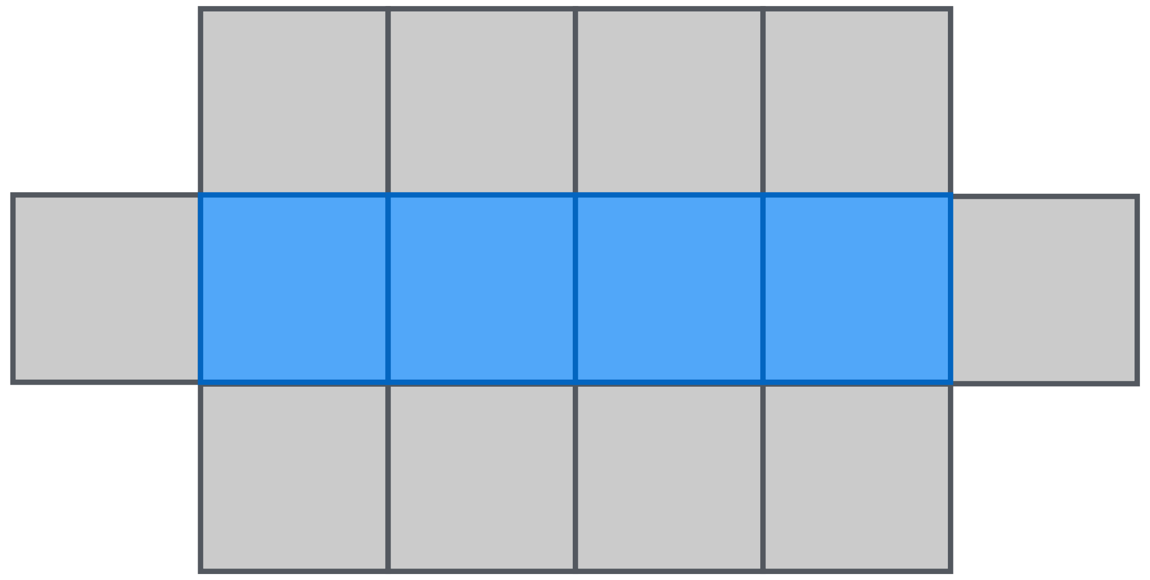


Stencil optimisations

- Use a different kernel for the interior and border regions
  - In the gaussian blur example of a 512x512 pixel image, 98% of the pixels do not require in-bounds checks
- Optimise data locality & reuse through tiling
  - *Strip mining* is an optimisation that groups elements in a way that avoids redundant memory access and aligns accesses with cache lines



4 x (5 reads + 1 write)

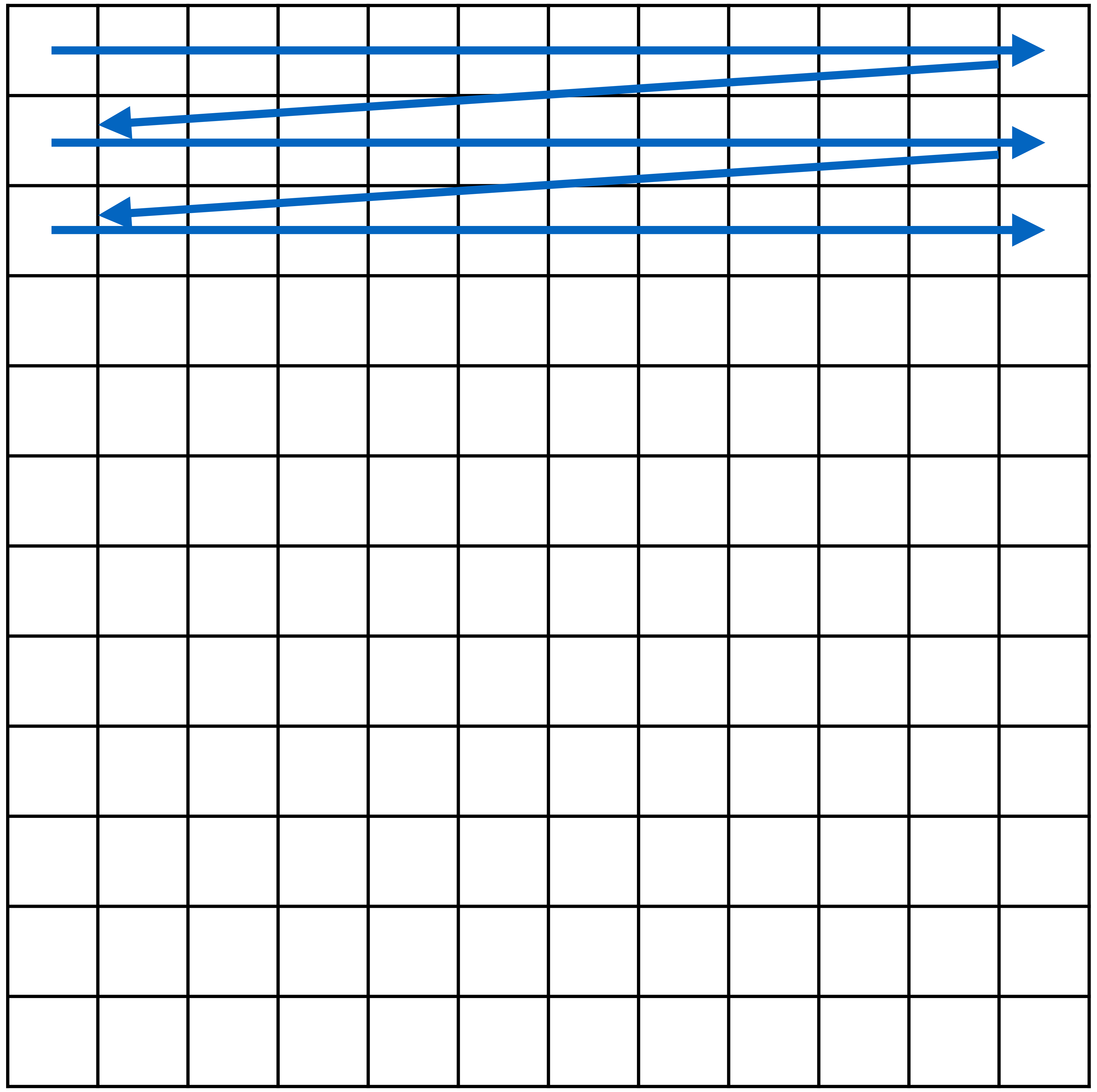


14 reads + 4 writes

Stencil optimisations

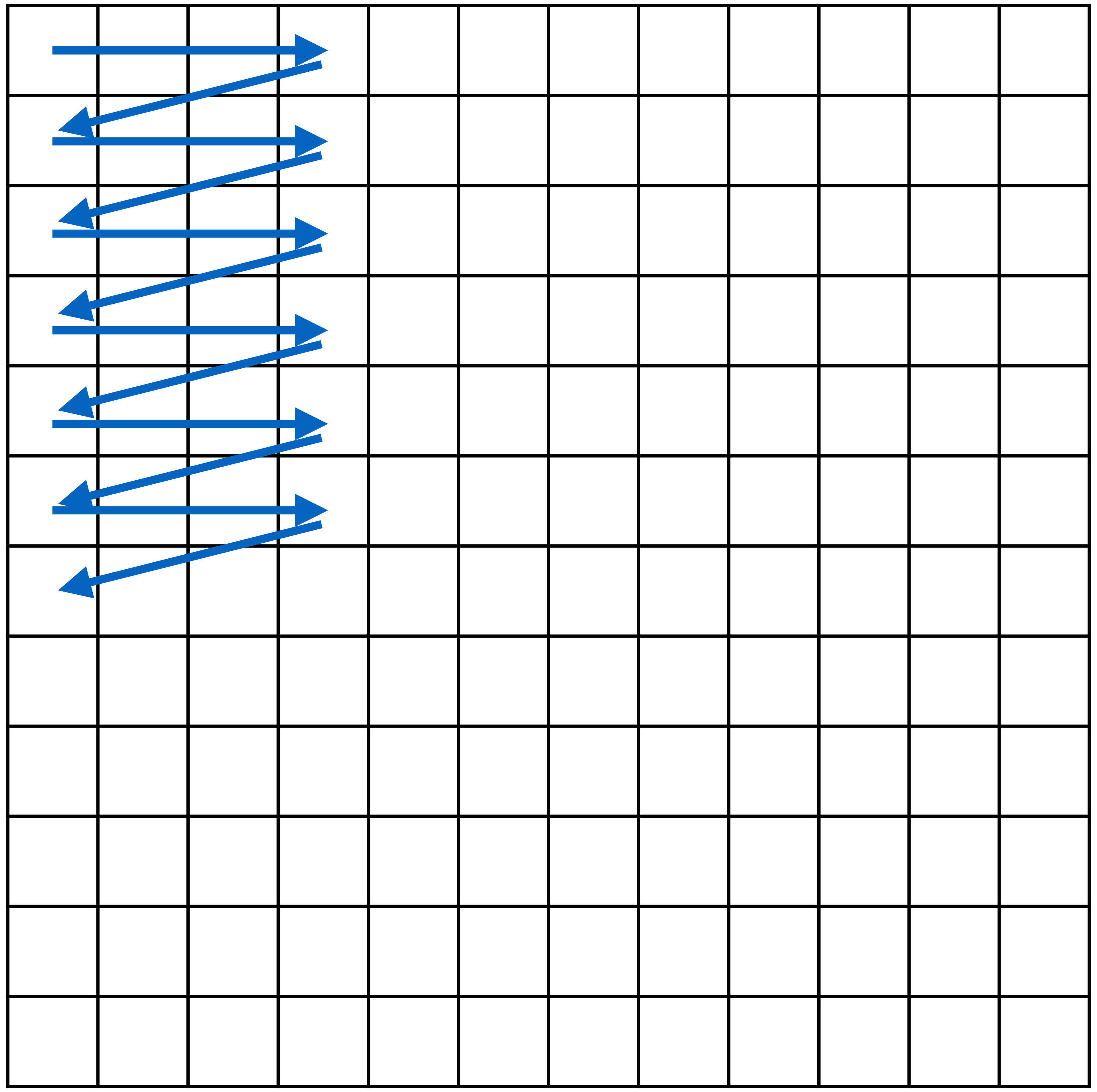
Without tiling

- When handling row 0, row 1 is loaded in cache.
- First values of row 1 may already be out of cache, when handling row 1

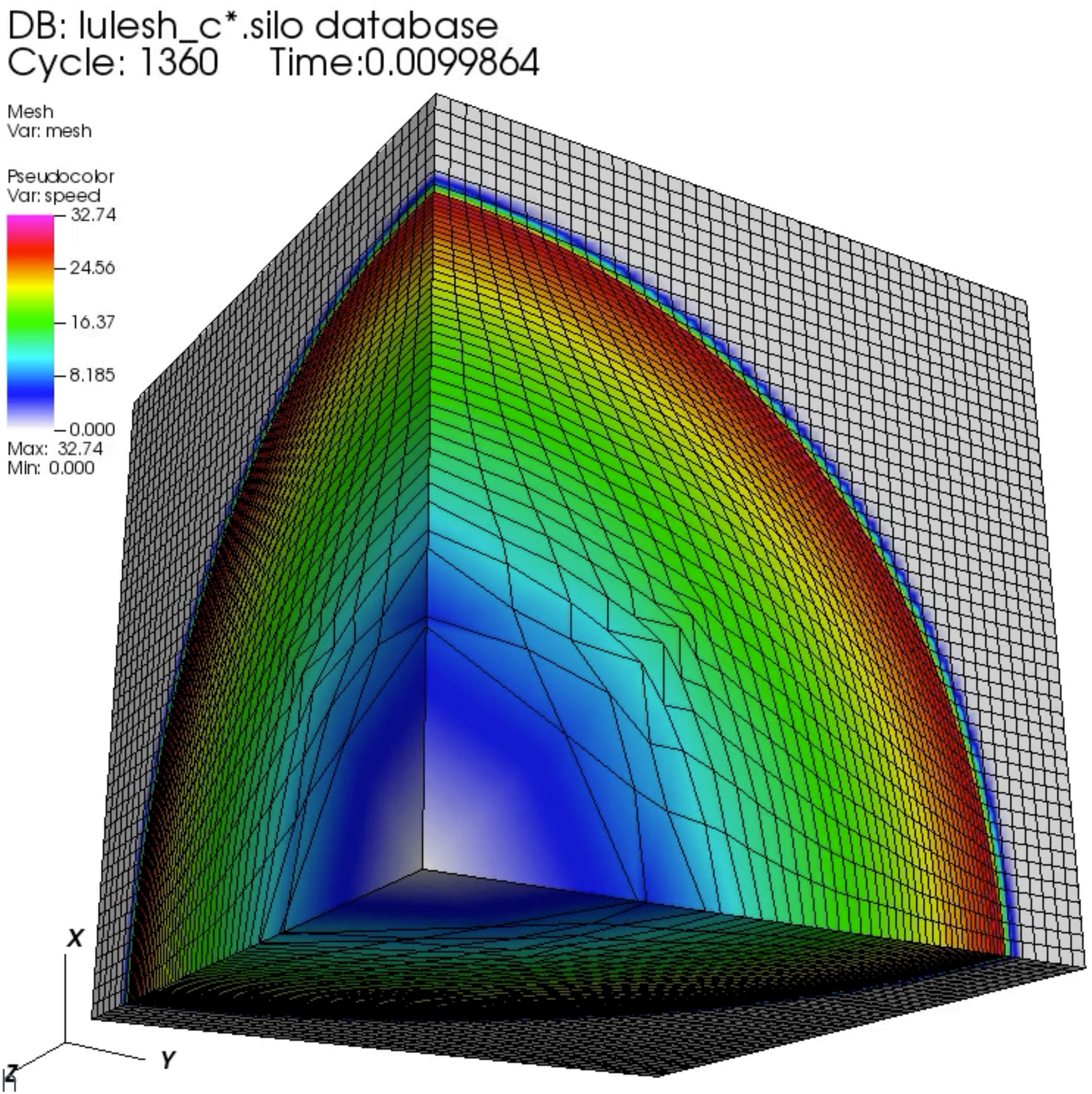


With tiling

- Previously loaded row is still in cache
- Tile width is usually a power of 2, on GPUs often the warp size (32)



Example: LULESH



https://computing.llnl.gov/projects/co-design/lulesh
https://github.com/tmcdonell/lulesh-accelerate

Summary

- Data-parallelism is a good fit for parallel computing
  - Conceptually simple programming model: single logical thread of control
  - Separate the pattern (what you want to do) from the implementation (how to do it: optimisations, target hardware, etc.)

tot ziens