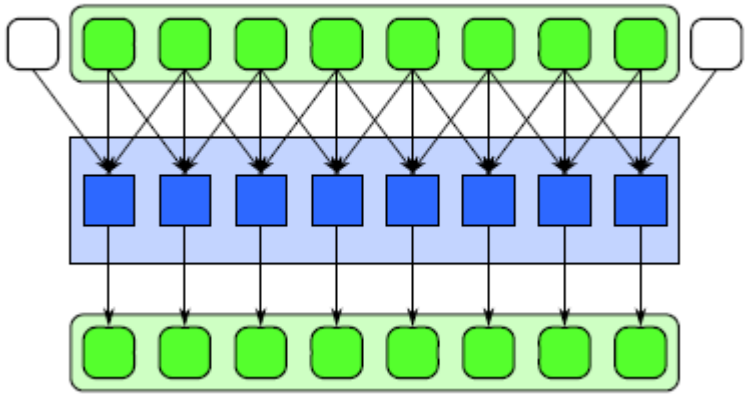
Patterns: stencil

UROŠ LOTRIČ

Stencil

Special case of map

- 1D or multiple dimensions



Has regular data access pattern

- Each output depends on a neighbourhood of inputs
- Inputs have fixed offsets relative to the output
- Can be implemented as
  - Set of random reads for each output
  - Shifts

Applications

- Image and signal processing (convolution)
- Physics, mechanical engineering, CFD (PDE solvers over regular grids)

Stencil

Different neighbourhoods

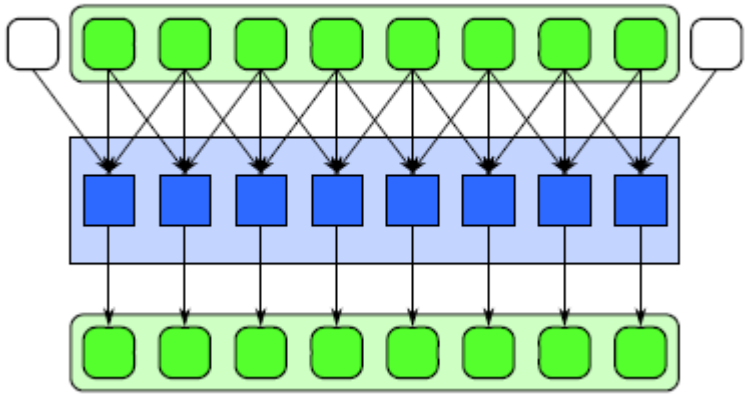
- Square compact, …, sparse

Cache optimizations

- Stencils reuse samples required for neighbouring elements

Boundaries of grids given to a processor

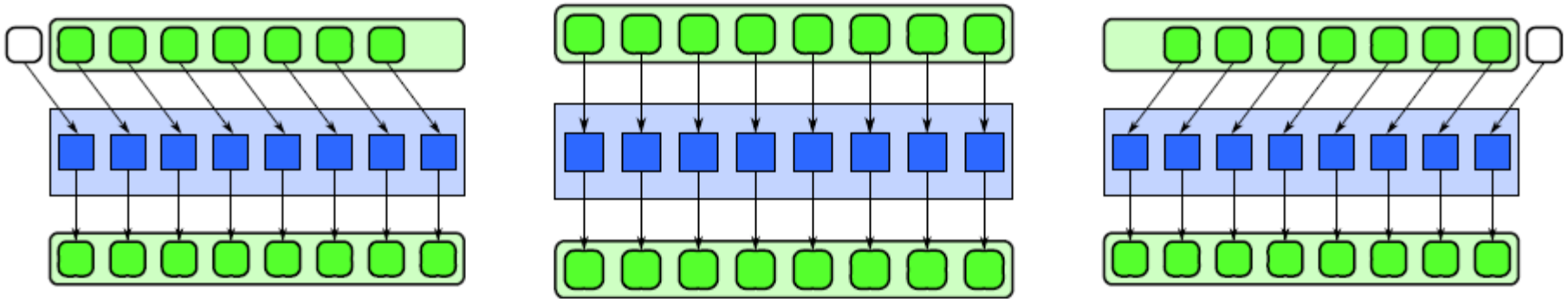
- Exchange data with other processors
- Additional communication costs



Stencil

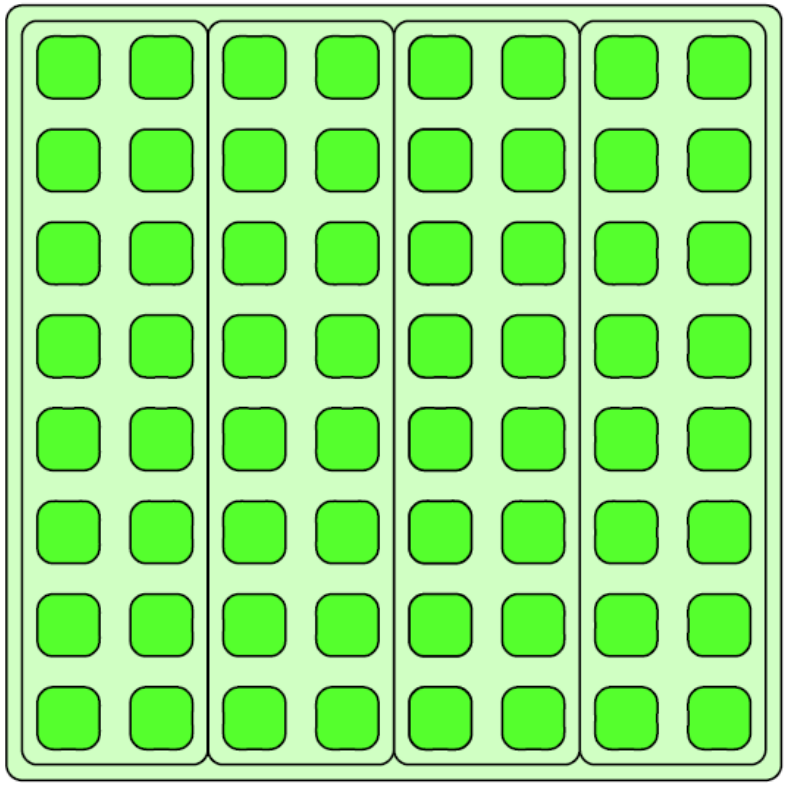
Implementation with shift operation

- Beneficial for 1D stencils
- Allow vectorization of data reads
- Does not reduce memory traffic



Stencil

Implementation with tiles



- Multidimensional stencils
- Strip-mining (optimized for cache)
- Example
  - Two dimensional array organized in row-by-row fashion with many vertical offsets
    - Horizontal data in the same cache line, vertical far away
  - Horizontal split
    - whole line does not fit cache, a lot of cache misses when accessing adjacent rows
  - Vertical split
    - processors redundantly read the same cache line
  - Strips
    - Each processor gets its strip of width equal to a multiple of cache line size
    - Processing goes sequentially from top to bottom to maximize cache reuse
    - Multiple of cache line size prevents false sharing between adjacent strips on output

Stencil

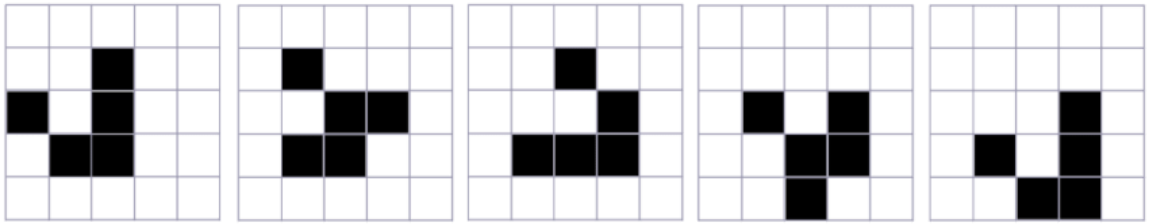
Communication

- Commonly the output of stencil is used as the input for the next iteration
  - Double buffering
  - Pointers to buffers are interchanged between iterations
- Need for synchronization
- Boundary regions (halo) of the grid may need explicit communication with neighbouring processors
  - Halo can be exchanged each iteration
  - Data exchange can take place on each *k*-th iteration when halo is increased and some redundant computation takes place on each processor
  - Latency hiding (update of internal grid cells when waiting for halo exchange)

Stencil

Example

- Conway's Game of Life
  - Zero player game played on a board of cells
  - A cell can be dead or alive, has 8 neighbours
  - Rules
    - A live cell with < 2 neighbours dies
    - A live cell with 2 or 3 neighbours lives on
    - A live cell with > 3 neighbours dies
    - A dead cell with 3 neighbours becomes alive
  - Iterations are not independent
    - Status of all cells must be computed first
    - All cell statuses are refreshed at once



Patterns: Reduce

GPU reduce

Example: dot product

- One thread, sequential
  - Problem size and number of threads
  - Shared memory
  - Summation on host
- Tree-like
  - Sum neighbours: stride is increasing with iterations
  - Warp-optimized solution: stride is decreasing with iterations
  - Non-power-of-two

Patterns: Scan

Scan

Also prefix scan

Produces all partial reductions of an input sequence

- exclusive and inclusive scan

Operation

- Input sequence:
  - $[a_0, a_1, a_2, \ldots, a_{n-1}]$
- Output:
  - exclusive scan: $[I, a_0, a_0 \circ a_1, a_0 \circ a_1 \circ a_2, \ldots, a_0 \circ \cdots \circ a_{n-2}]$
  - inclusive scan: $[a_0, a_0 \circ a_1, a_0 \circ a_1 \circ a_2, \ldots, a_0 \circ \cdots \circ a_{n-1}]$
- Example: summation

X	3	4	6	3	8	7	5	4	
Y	0	3	7	13	16	24	31	36	Exclusive Scan
Y	3	7	13	16	24	31	36	40	Inclusive Scan

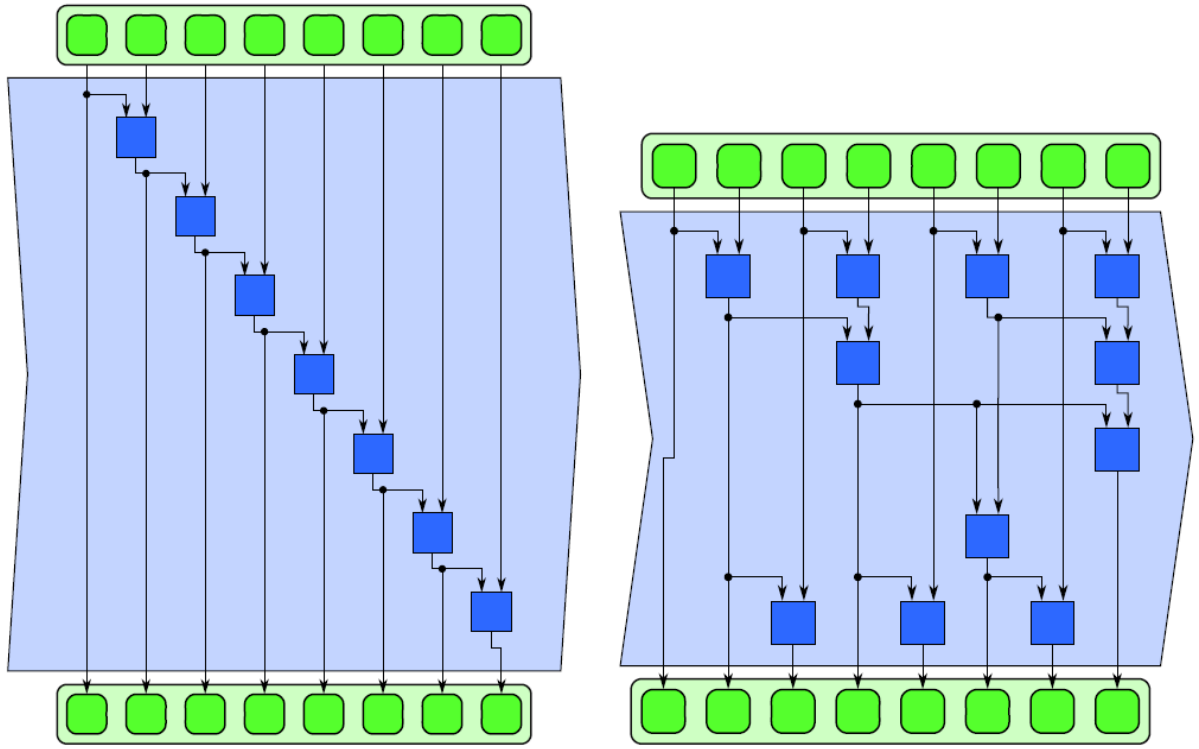
Scan patterns

Sequential approach

- Loop-carried dependence

Parallel approach

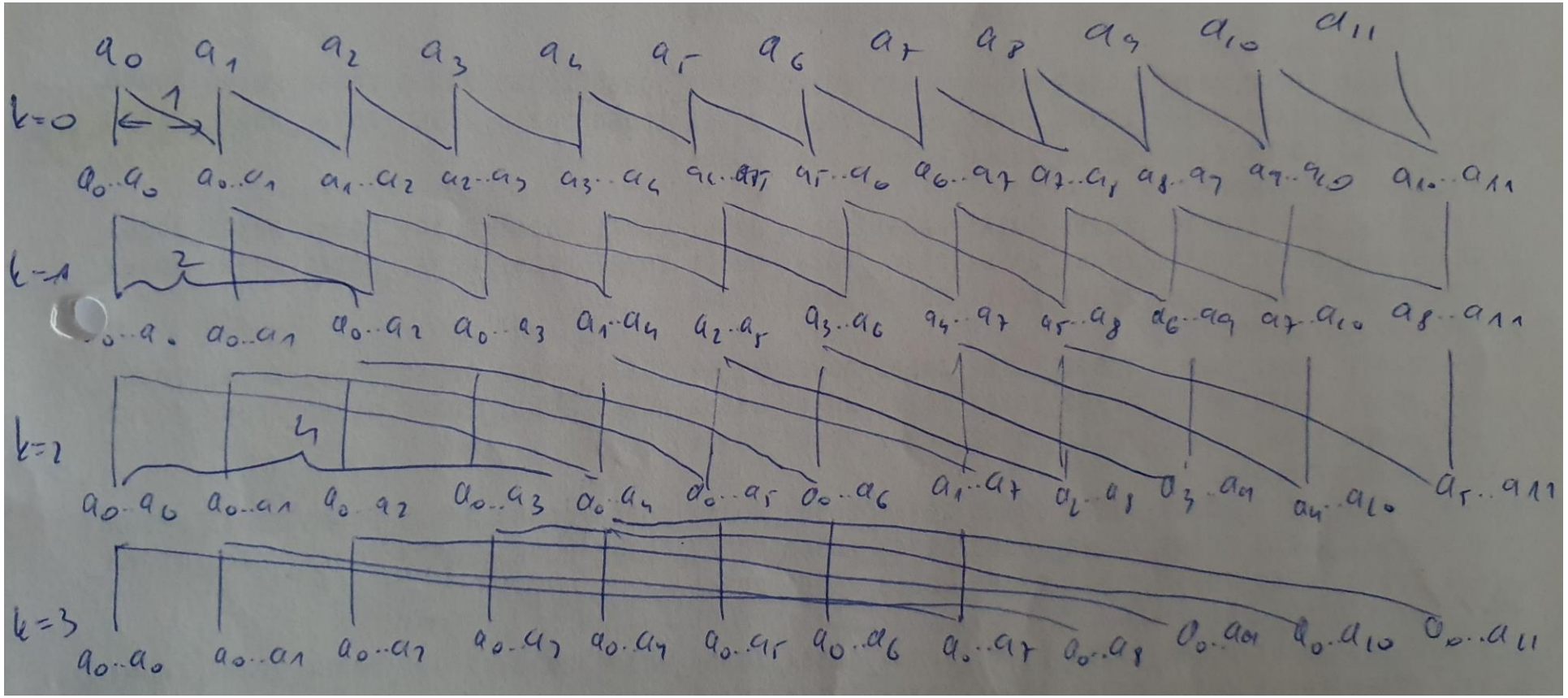
- Loop-carried dependence
- Similar to reduce
- Two solutions
- Count on associativity of combiner function (∘)



Scan, version 1

Hillis and Steele, 1986

Solution requires two buffers of length n



Scan, version 1

Number of synchronization steps: $\lceil \log_2 n \rceil$

Number of operations when n is power of 2: $n(\log_2 n - 1) + 1$

Sequential time

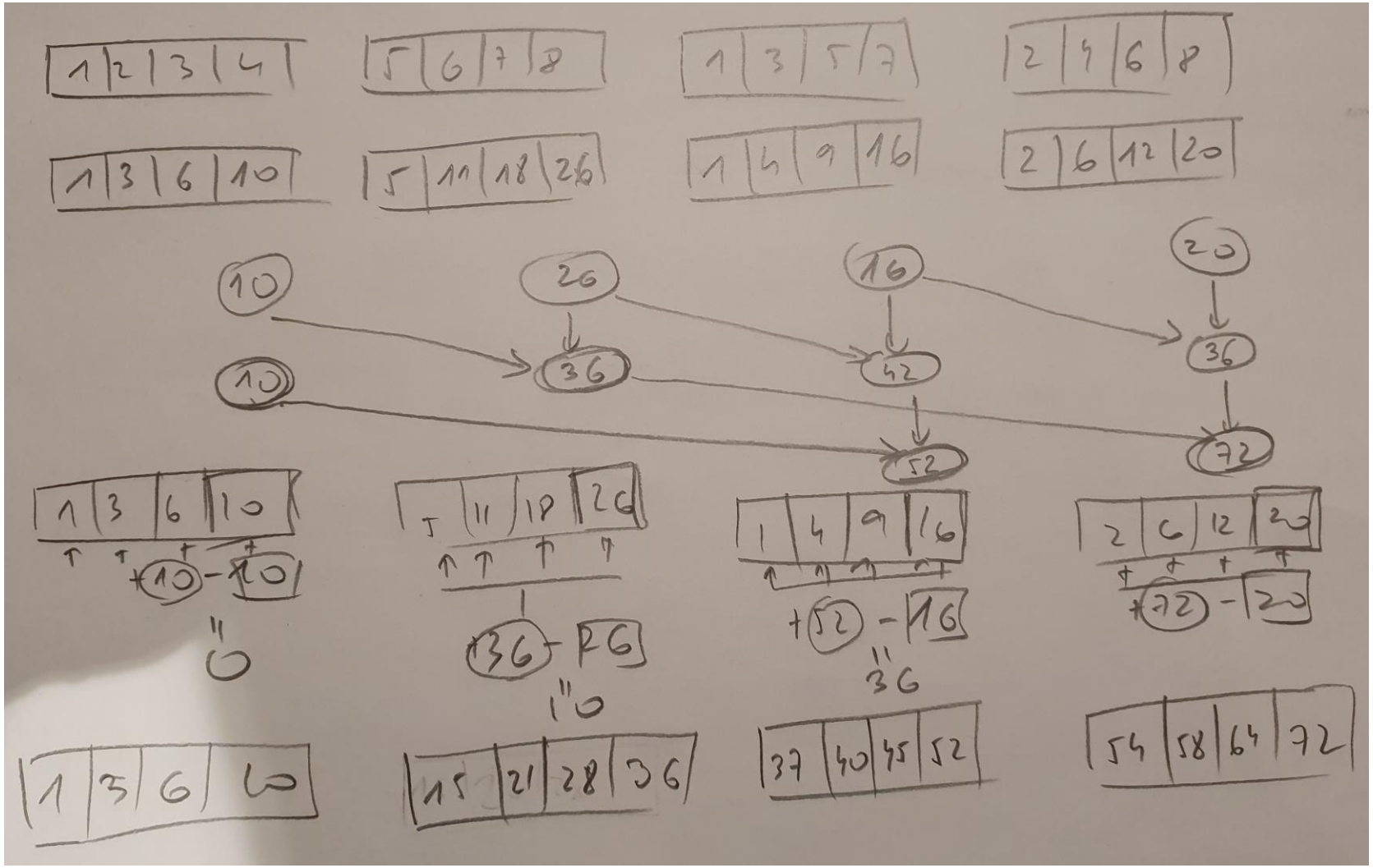
- $t_s(n) = \chi(n-1) = O(n)$

Scan, version 1

Tiled computation

Example

- n elements
- p processes



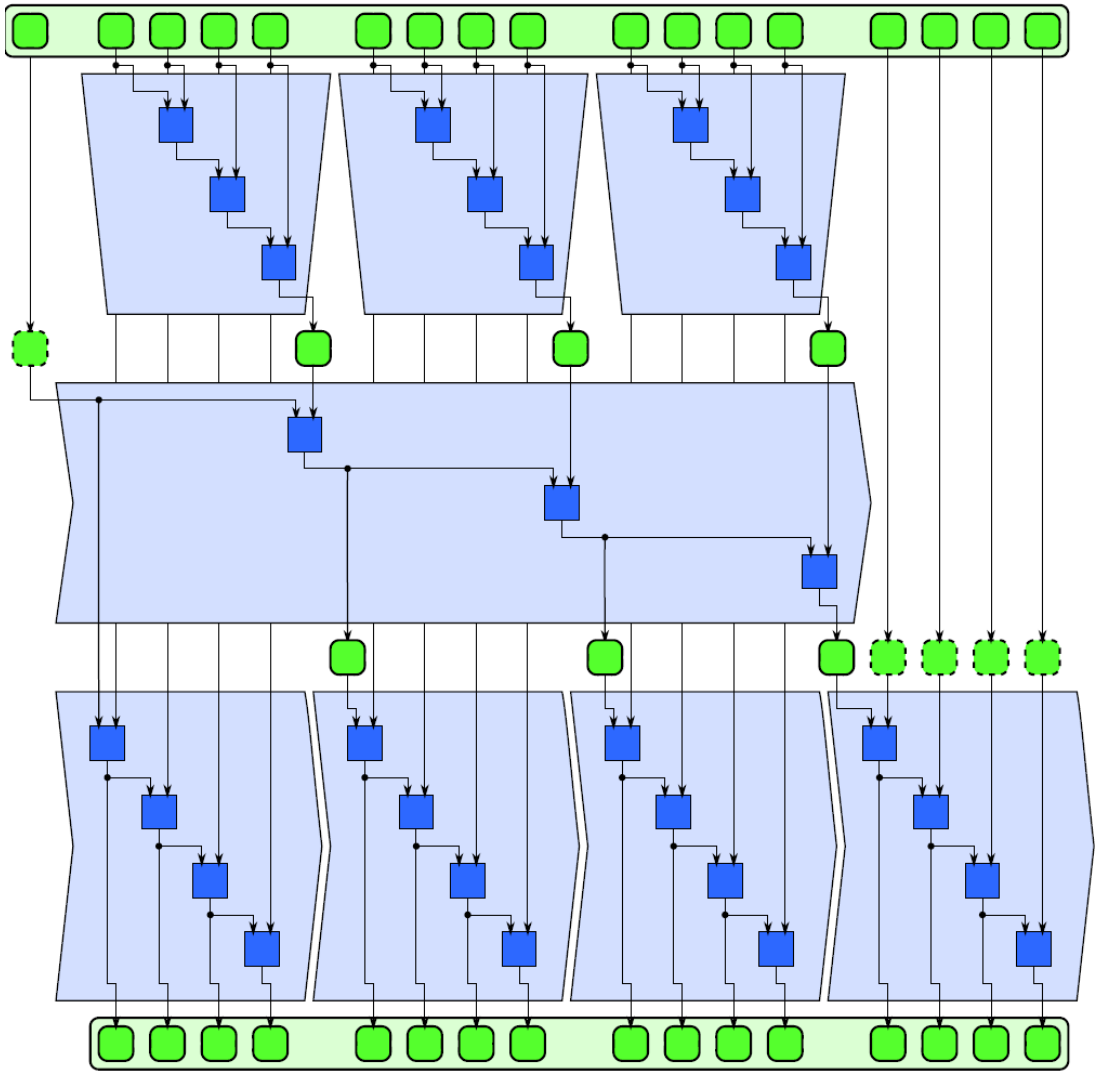
Scan, version 1

Tiled computation

Pattern

Example

- Tiled computation in OpenCL: solution with one work-item
- Tiled computation in OpenCL: Hillis and Steele



Scan, version 2

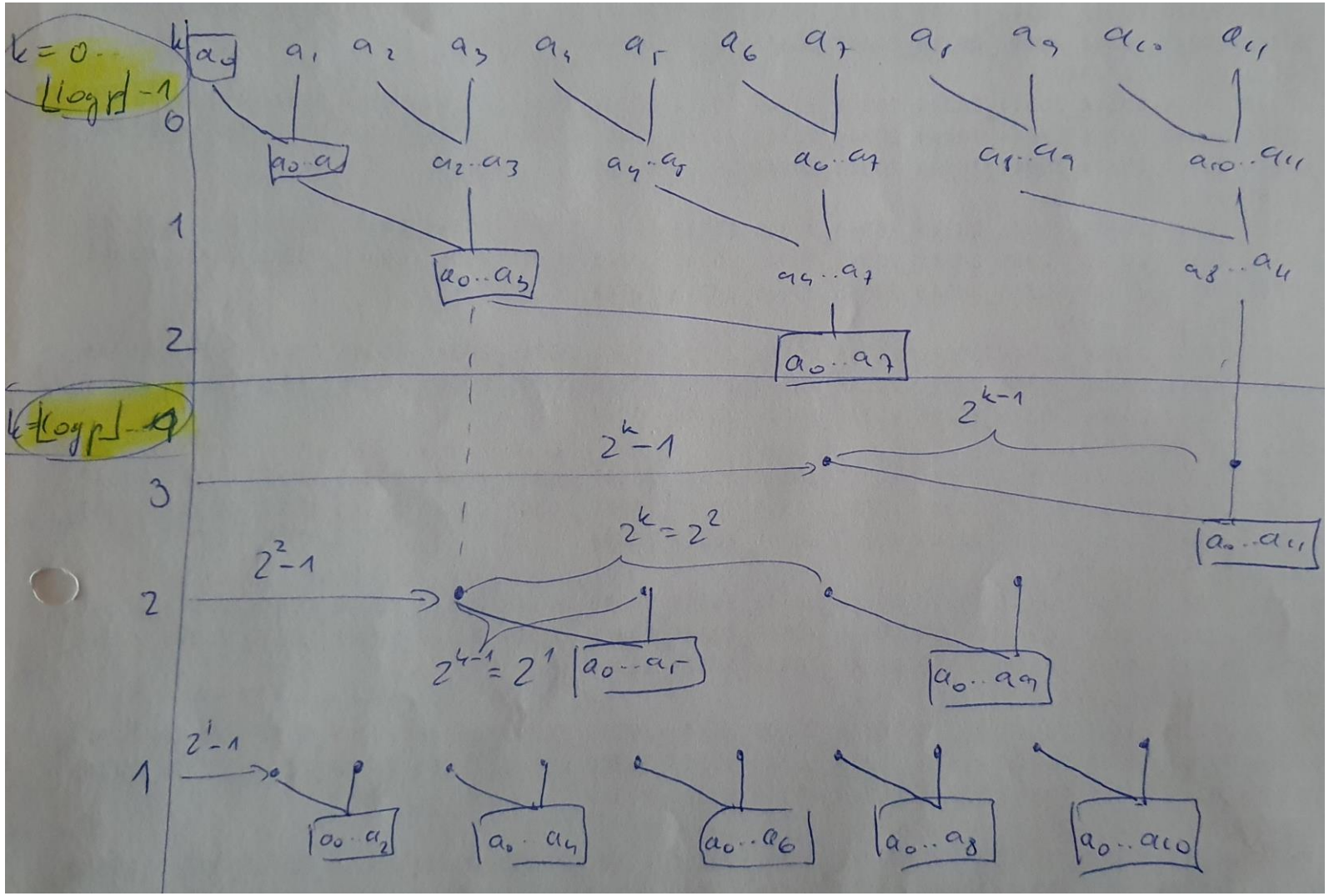
Work efficient solution

One buffer suffices

Figure

- n = 12
- rounded values are final

Similar to reduction with increasing stride



Combining scan

Map

- Tiled scan: map can be applied before the first stage and/or after the last stage
- Reduces data transfers

Reduce

- Similar scheme, can do both with little extra work