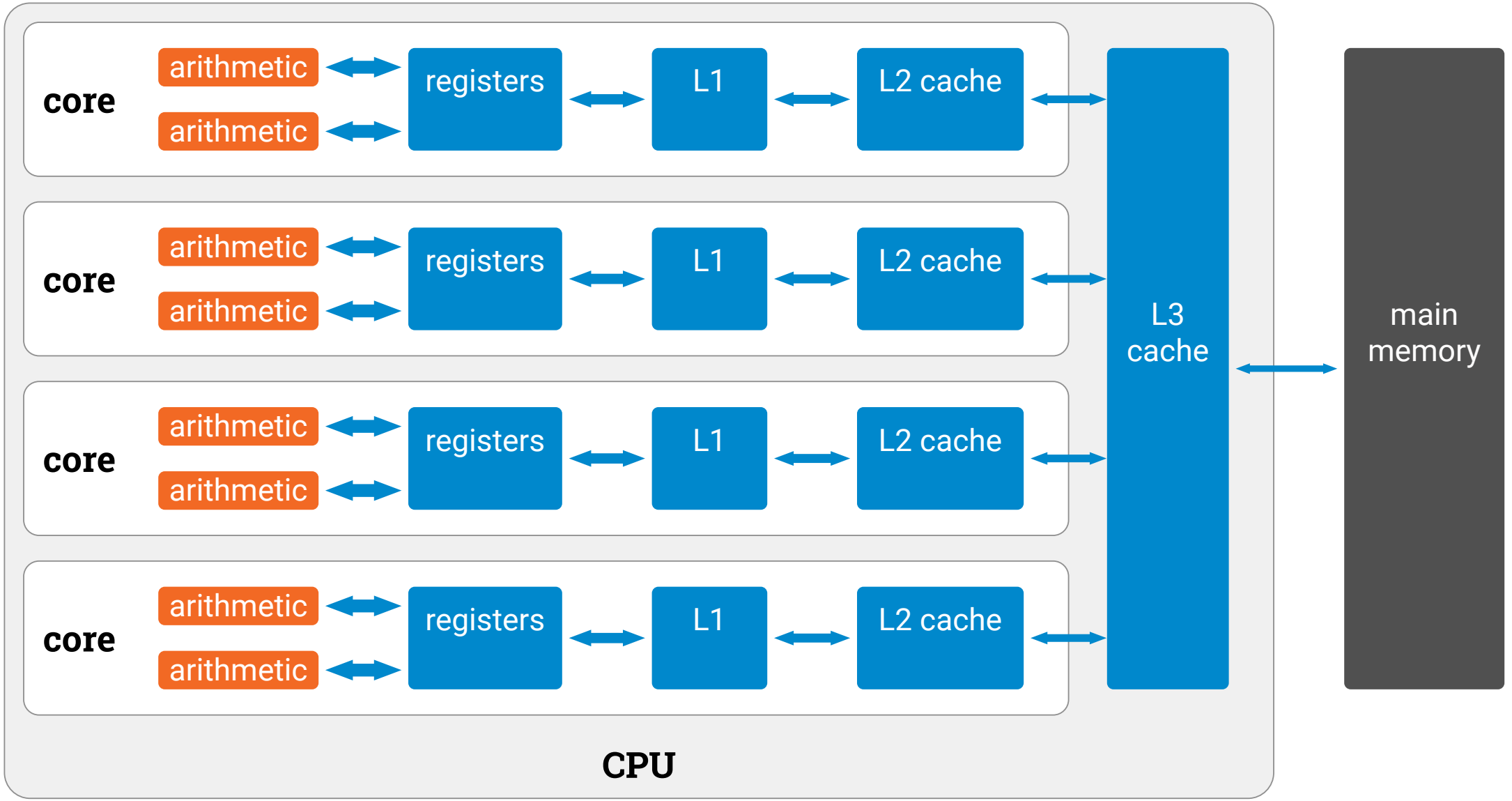
Programming Parallel Computers

Jukka Suomela · Aalto University · ppc.cs.aalto.fi

Part 3C: Reusing data in cache

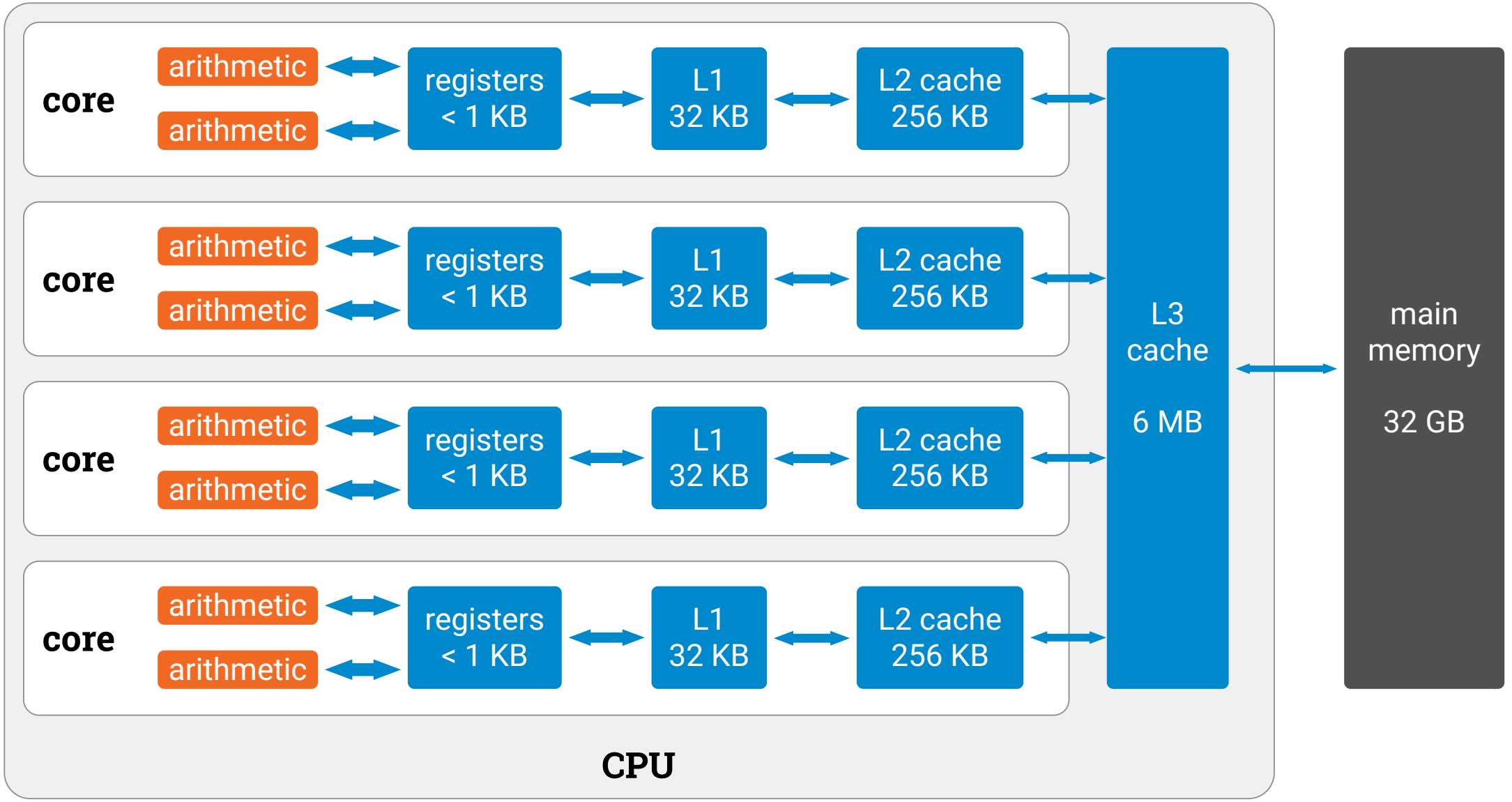
Reading little vs. reading efficiently

- **Previous part:**
  - how to organize code so that you ***read as little as possible***
  - read once to **registers**, use it many times
  - accessing registers is free
  - this should be always your **plan A**
- **This part:**
  - how to make sure that you can ***read as fast as possible***
  - organize memory access so that you benefit from **cache memory**
  - accessing cache is not as fast as registers
  - worry about this **when your plan A failed**…


core
arithmetic
arithmetic
registers
L1
L2 cache
core
arithmetic
arithmetic
registers
L1
L2 cache
core
arithmetic
arithmetic
registers
L1
L2 cache
core
arithmetic
arithmetic
registers
L1
L2 cache
L3 cache
main memory
CPU

How do caches work?

- When your program reads **anything** that is in the main memory:
  - ***CPU tries to get it from L1 cache***
  - if not there, try L2 cache
  - if not there, try L3 cache
  - if not there, get from main memory
  - ***CPU automatically stores it in caches*** if it was not there yet
  - makes space by throwing away some not-so-recently-used values

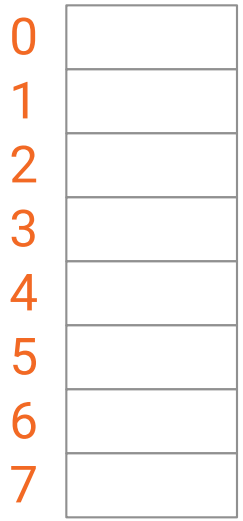

core
arithmetic
arithmetic
registers
< 1 KB
L1
32 KB
L2 cache
256 KB
core
arithmetic
arithmetic
registers
< 1 KB
L1
32 KB
L2 cache
256 KB
core
arithmetic
arithmetic
registers
< 1 KB
L1
32 KB
L2 cache
256 KB
core
arithmetic
arithmetic
registers
< 1 KB
L1
32 KB
L2 cache
256 KB
L3
cache
6 MB
main
memory
32 GB
CPU

How do caches work?

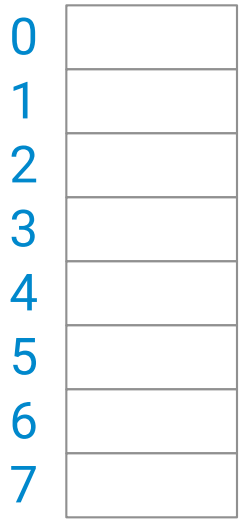
- Smallest meaningful unit of data: ***cache line = 64 bytes***
  - data in caches is organized in cache lines
  - data is transmitted between main memory and caches in cache lines
  - **you need just 1 byte — you will also get 63 other bytes around it**
  - you waste bandwidth if you don't take this into account
- It makes sense to access e.g. consecutive array elements
  - the first memory reference brings the whole cache line to caches
  - the next memory references get data from cache

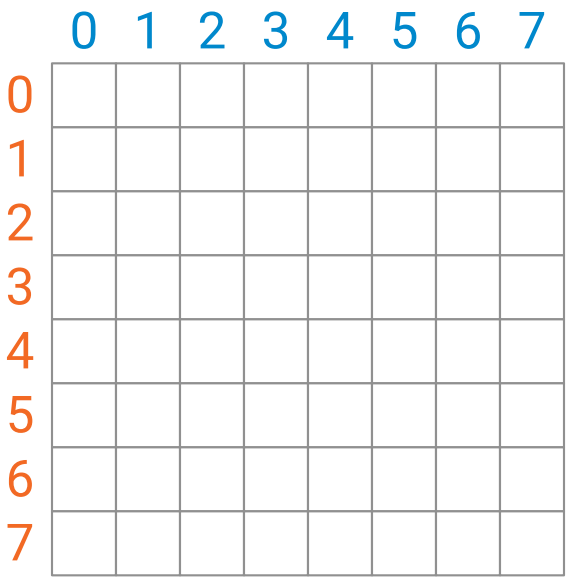
How to benefit from cache memory?

- You need to ***design the memory access pattern*** in your code so that you benefit from cache memory as much as possible
- Most of your memory reads should ***refer to elements that you have recently read***
  - or at least are in the same cache line as those that you have recently read
- Some examples of what this might mean in practice…

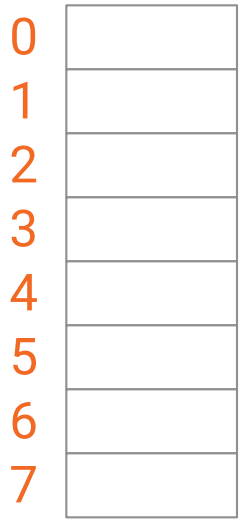


Input elements

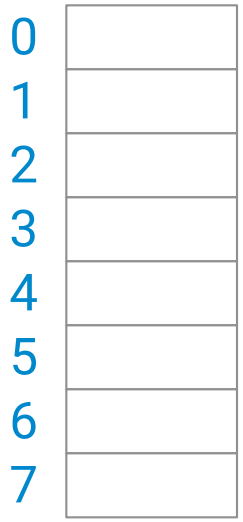


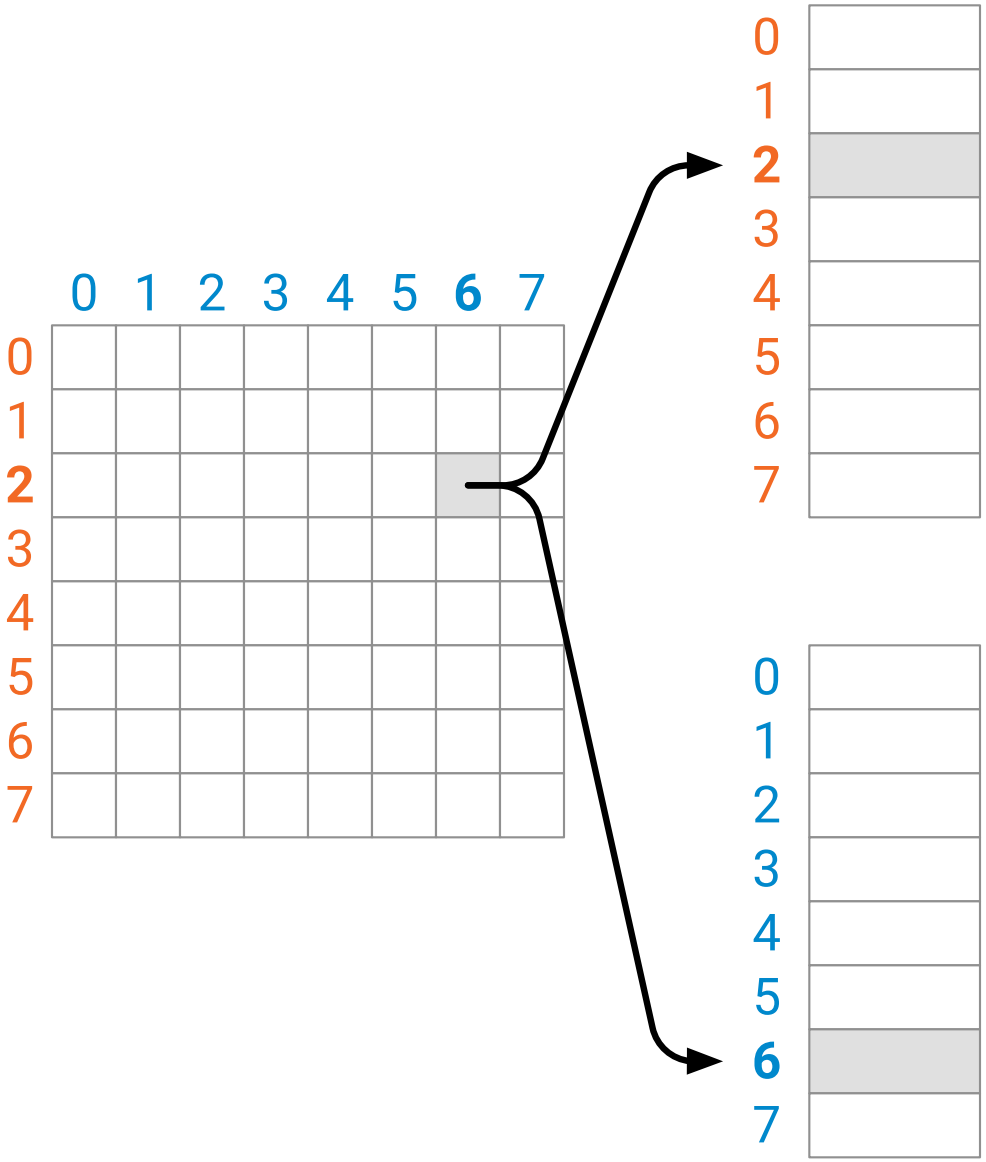


Output array

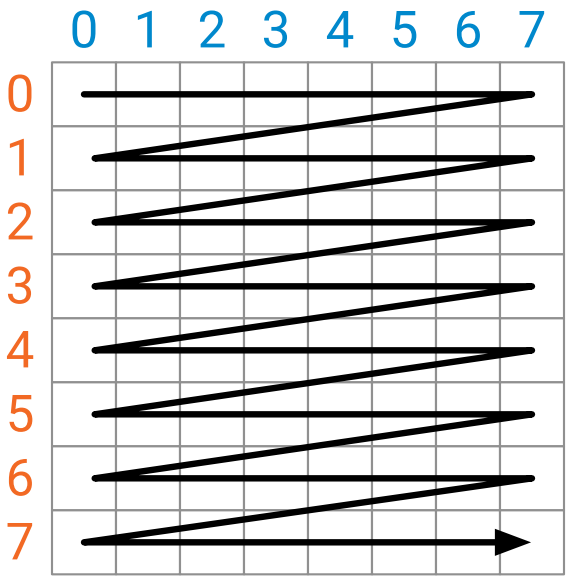


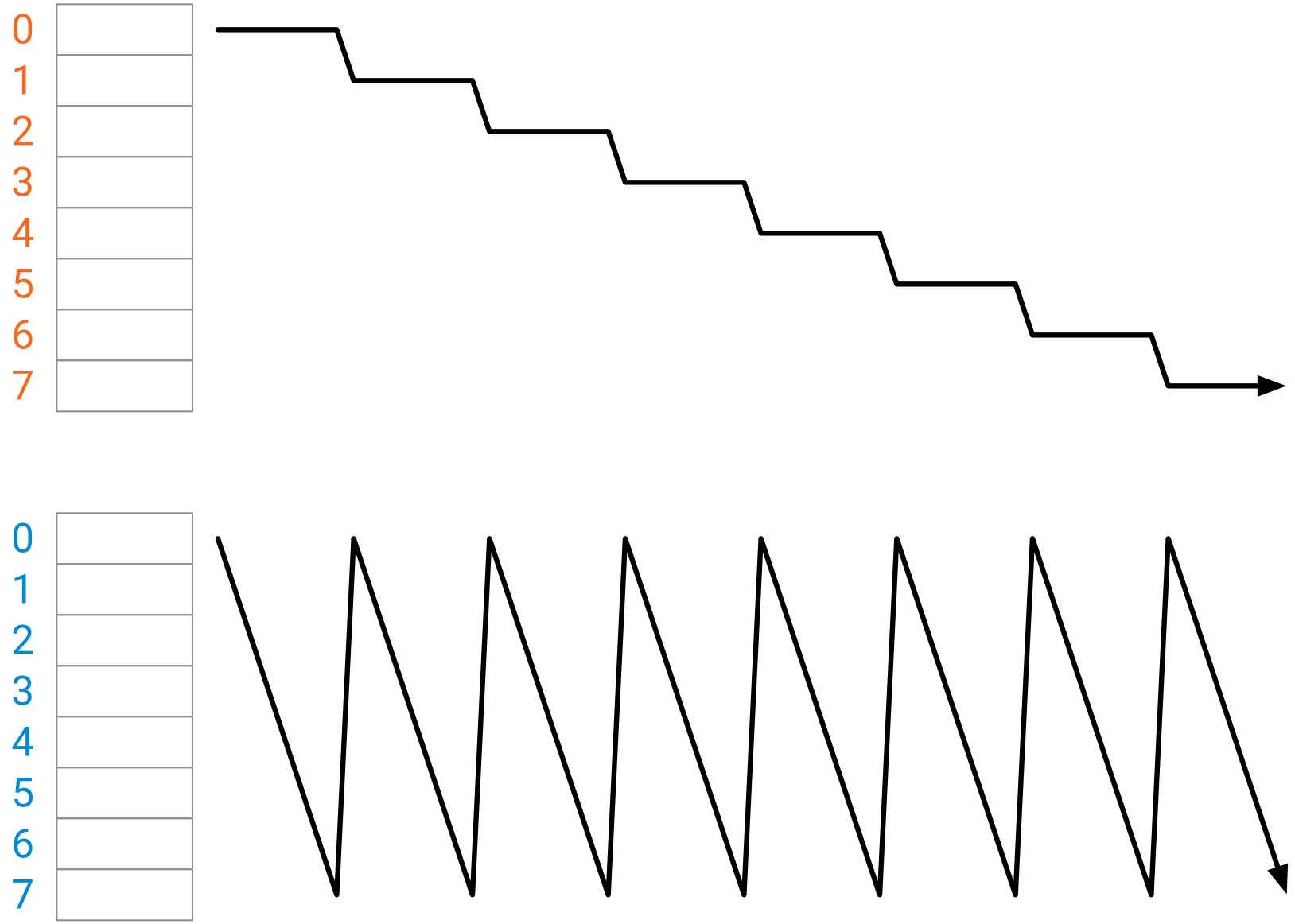
Input elements

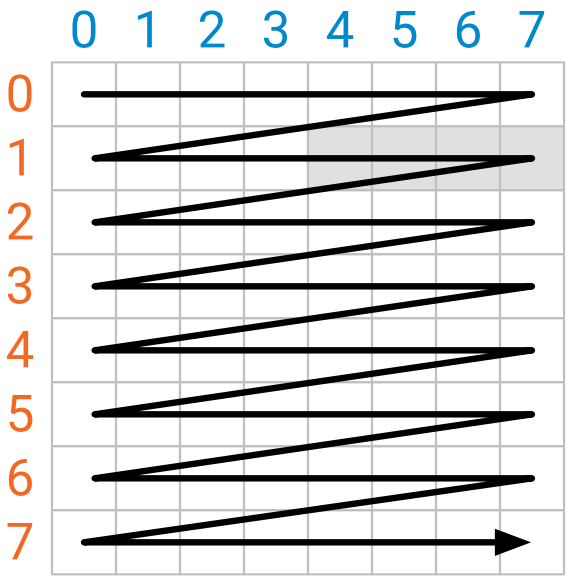


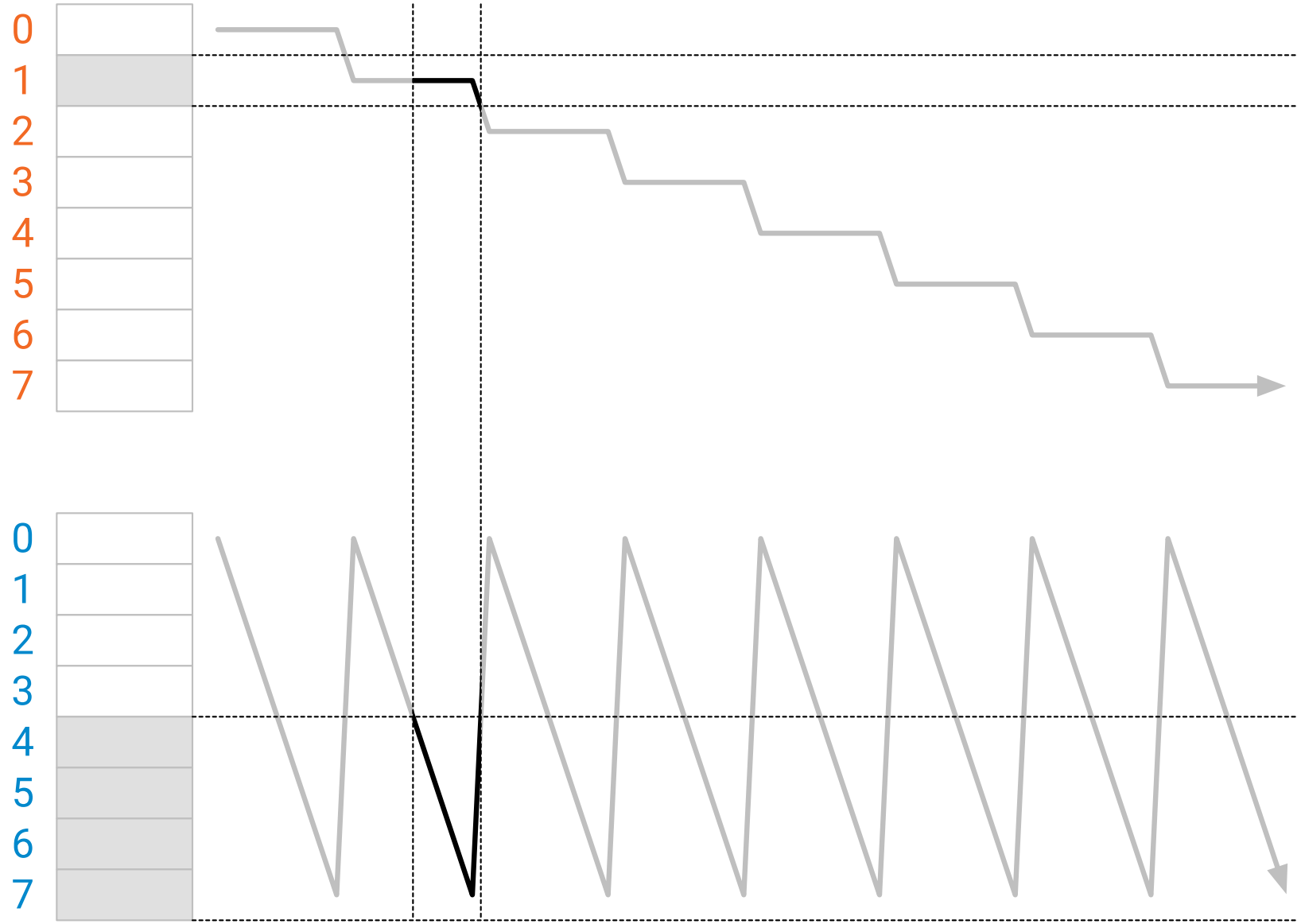


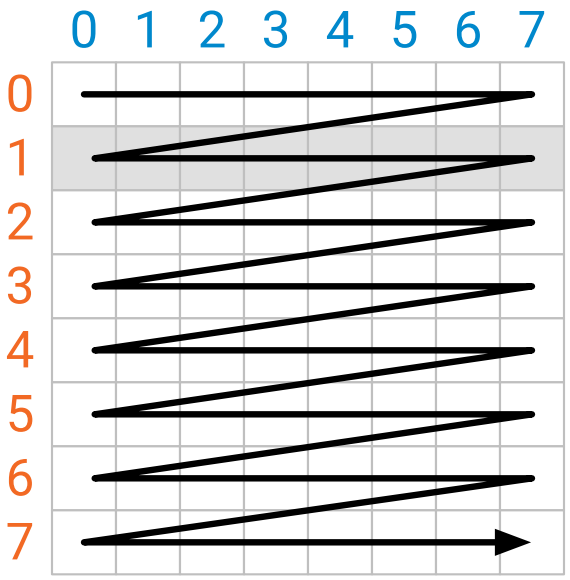
Output value (i, j) is computed from orange element i and blue element j

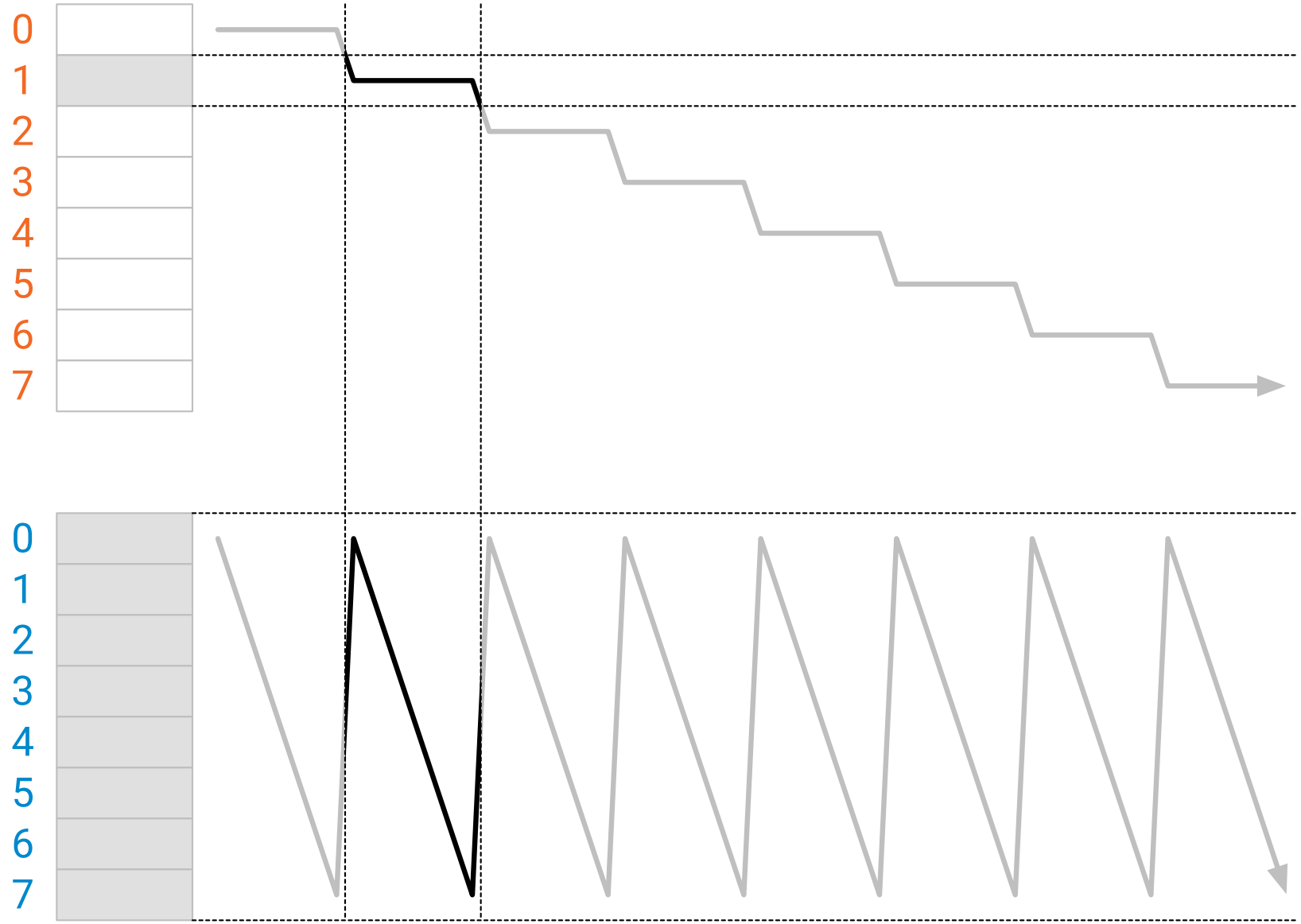

0 1 2 3 4 5 6 7
0
1
2
3
4
5
6
7

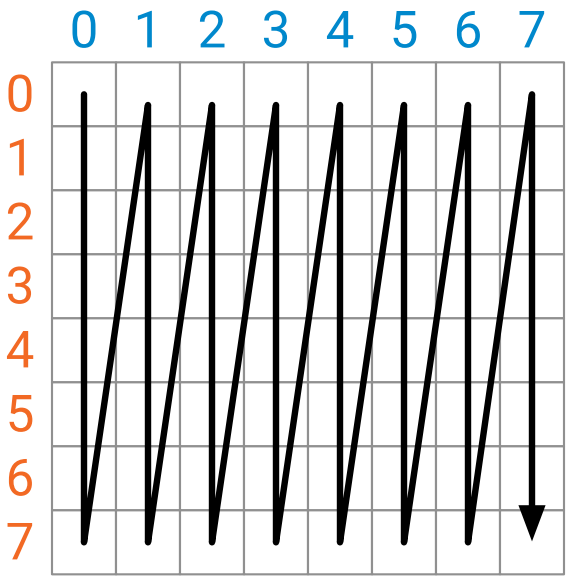

0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7

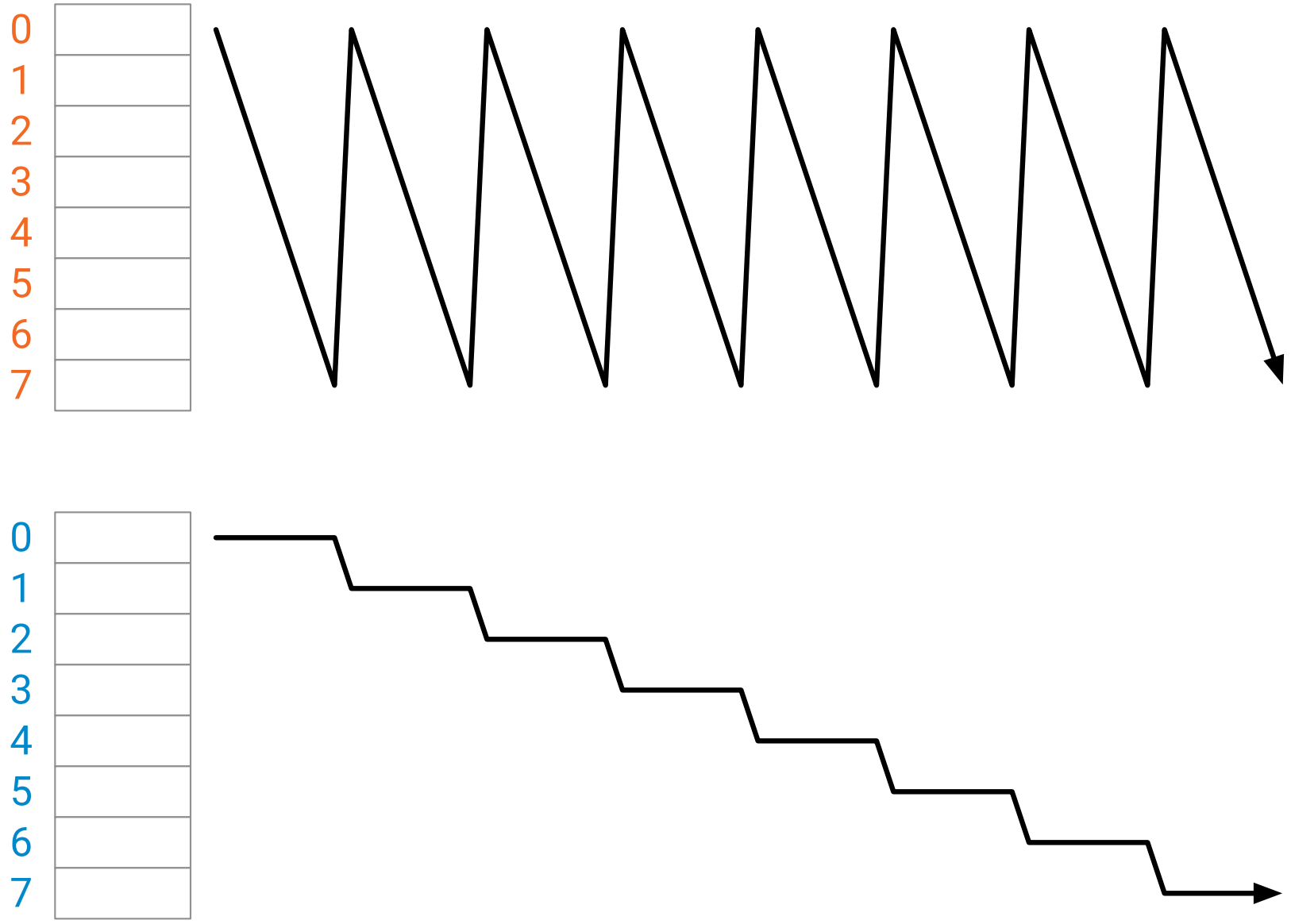

0 1 2 3 4 5 6 7
0
1
2
3
4
5
6
7

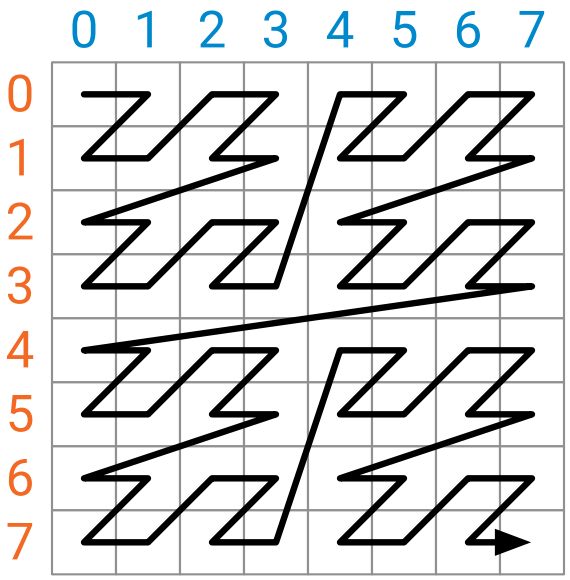

0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7

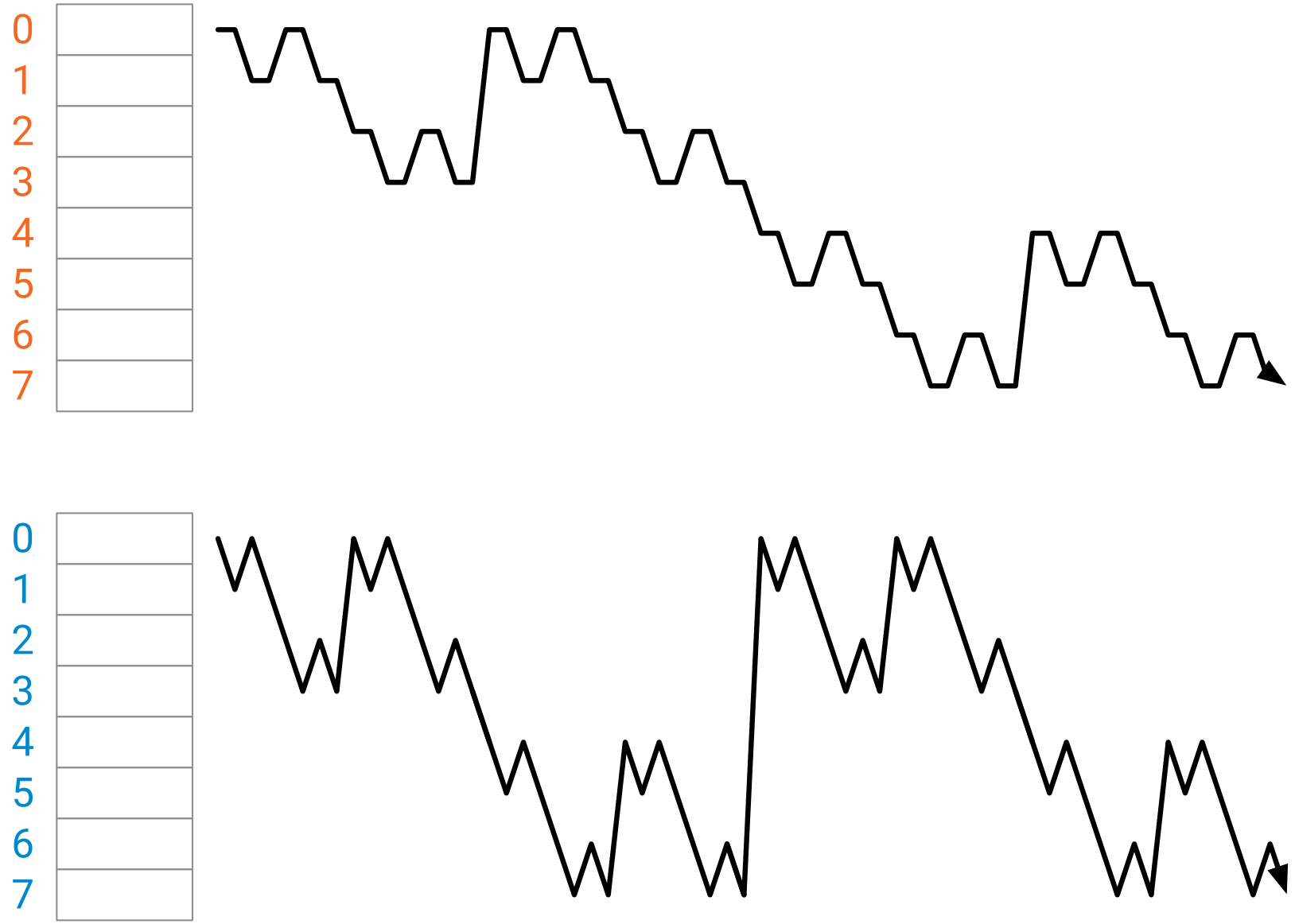

0 1 2 3 4 5 6 7
0
1
2
3
4
5
6
7

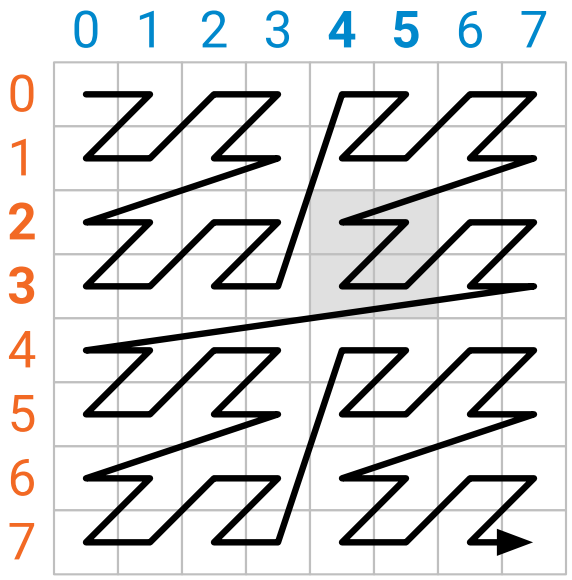

0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7

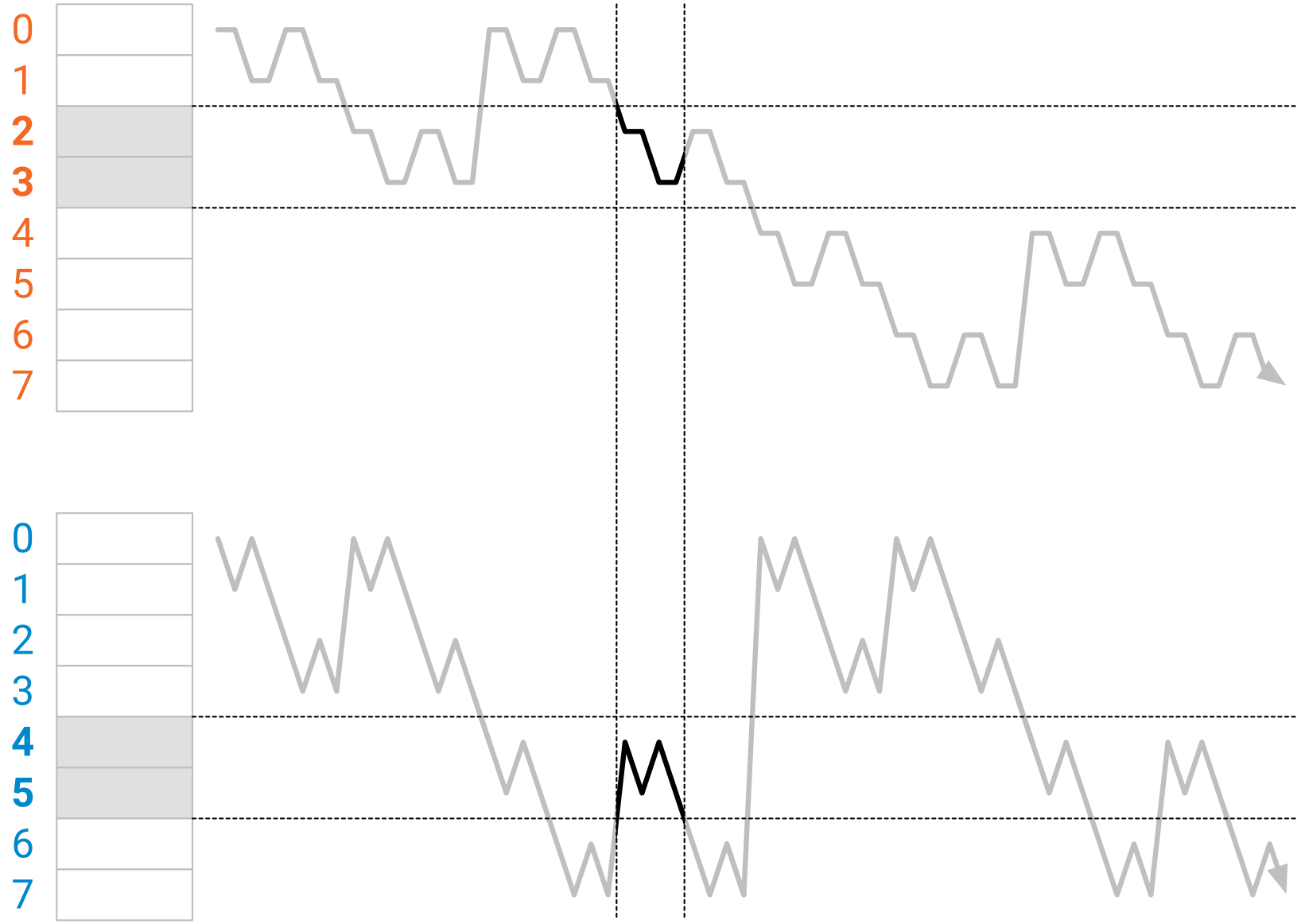

0 1 2 3 4 5 6 7
0
1
2
3
4
5
6
7

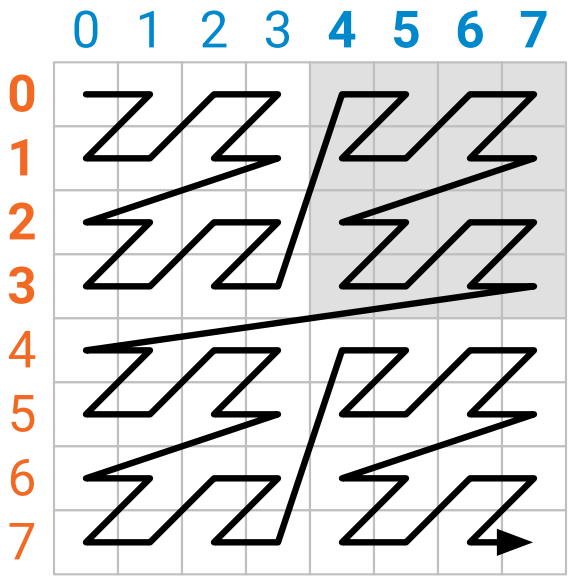

0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7

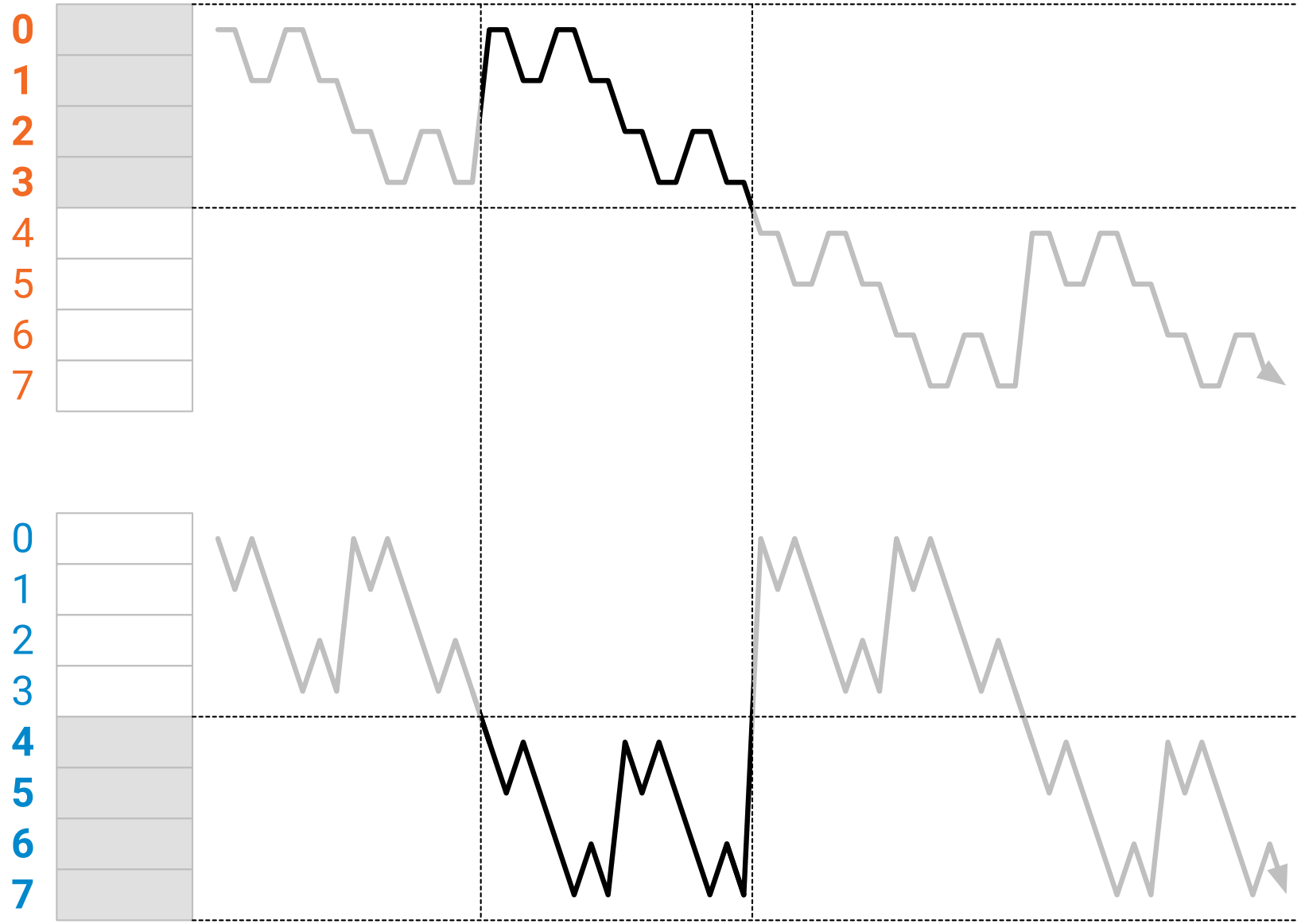

0 1 2 3 4 5 6 7
0
1
2
3
4
5
6
7

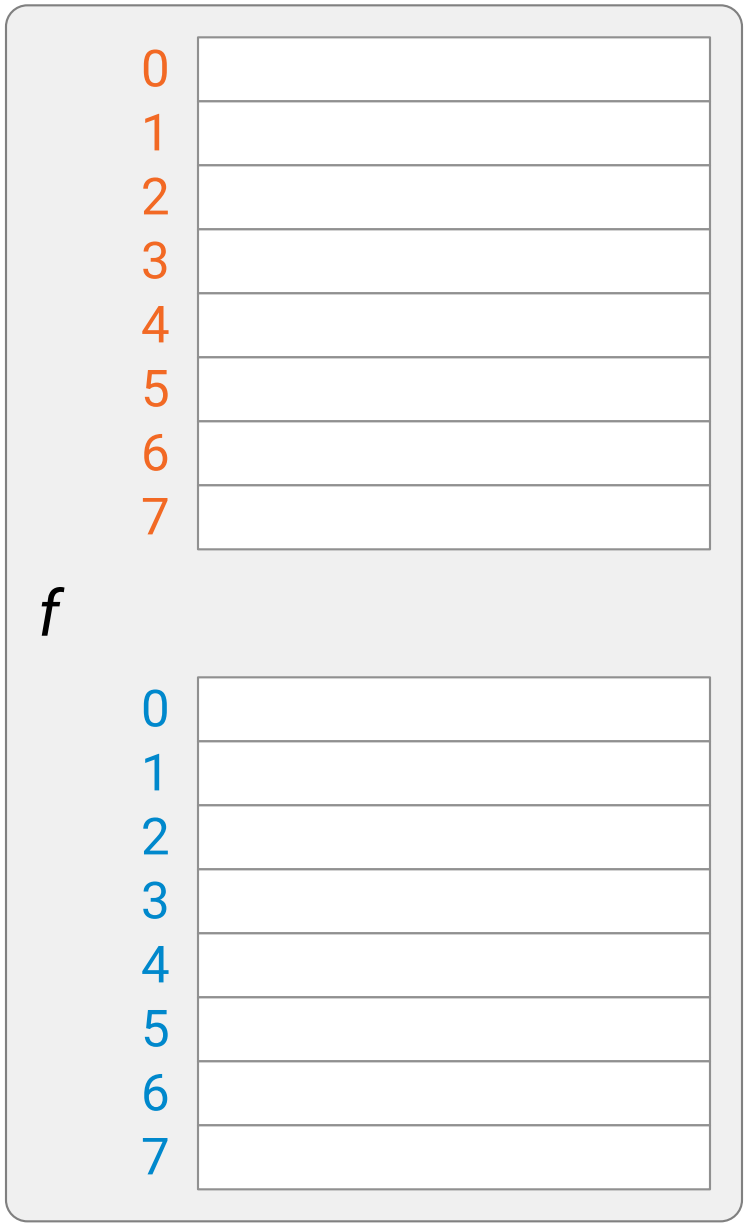

0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7

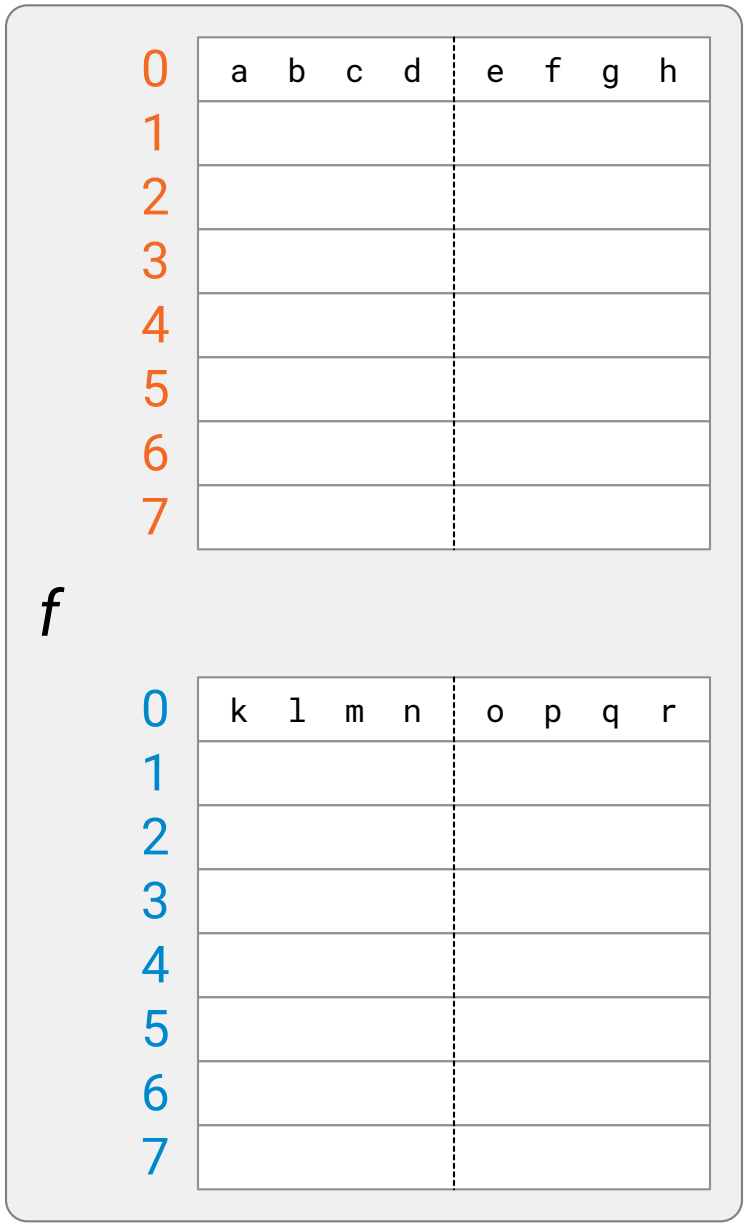

0 1 2 3 4 5 6 7
0
1
2
3
4
5
6
7

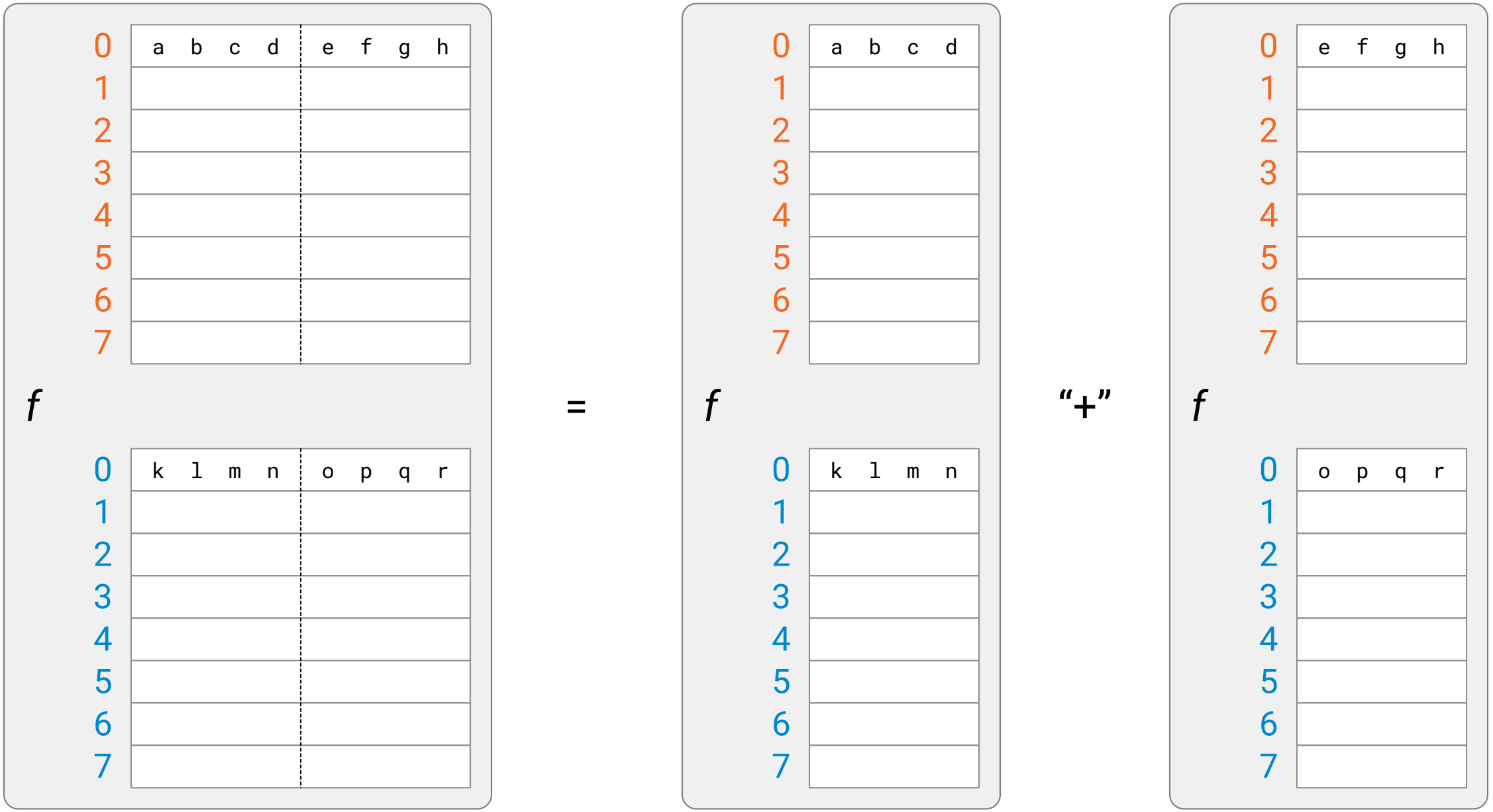

0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7


0 1 2 3 4 5 6 7
0
1
2
3
4
5
6
7


0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7


0
1
2
3
4
5
6
7
f
0
1
2
3
4
5
6
7


0
a b c d e f g h
1
2
3
4
5
6
7
f
0
k l m n o p q r
1
2
3
4
5
6
7


0 a b c d e f g h
1
2
3
4
5
6
7
f
0 k l m n o p q r
1
2
3
4
5
6
7
=
0 a b c d
1
2
3
4
5
6
7
f
0 k l m n
1
2
3
4
5
6
7
“+”
0 e f g h
1
2
3
4
5
6
7
f
0 o p q r
1
2
3
4
5
6
7

Putting it together

Baseline: **99 s**

Final: **0.7 s**

Factor-151 speedup

93% of theoretical maximum