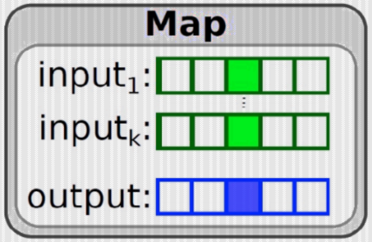
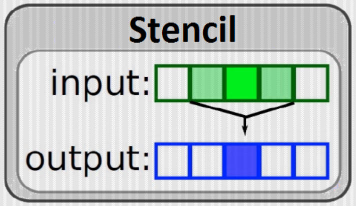
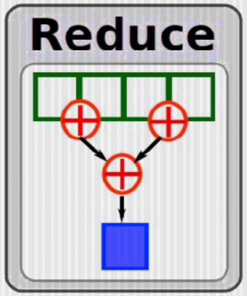
Parallel design patterns

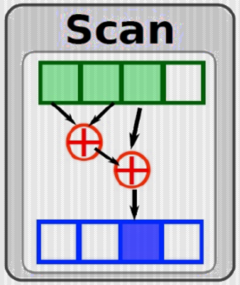
GPU programming







Parallel design patterns

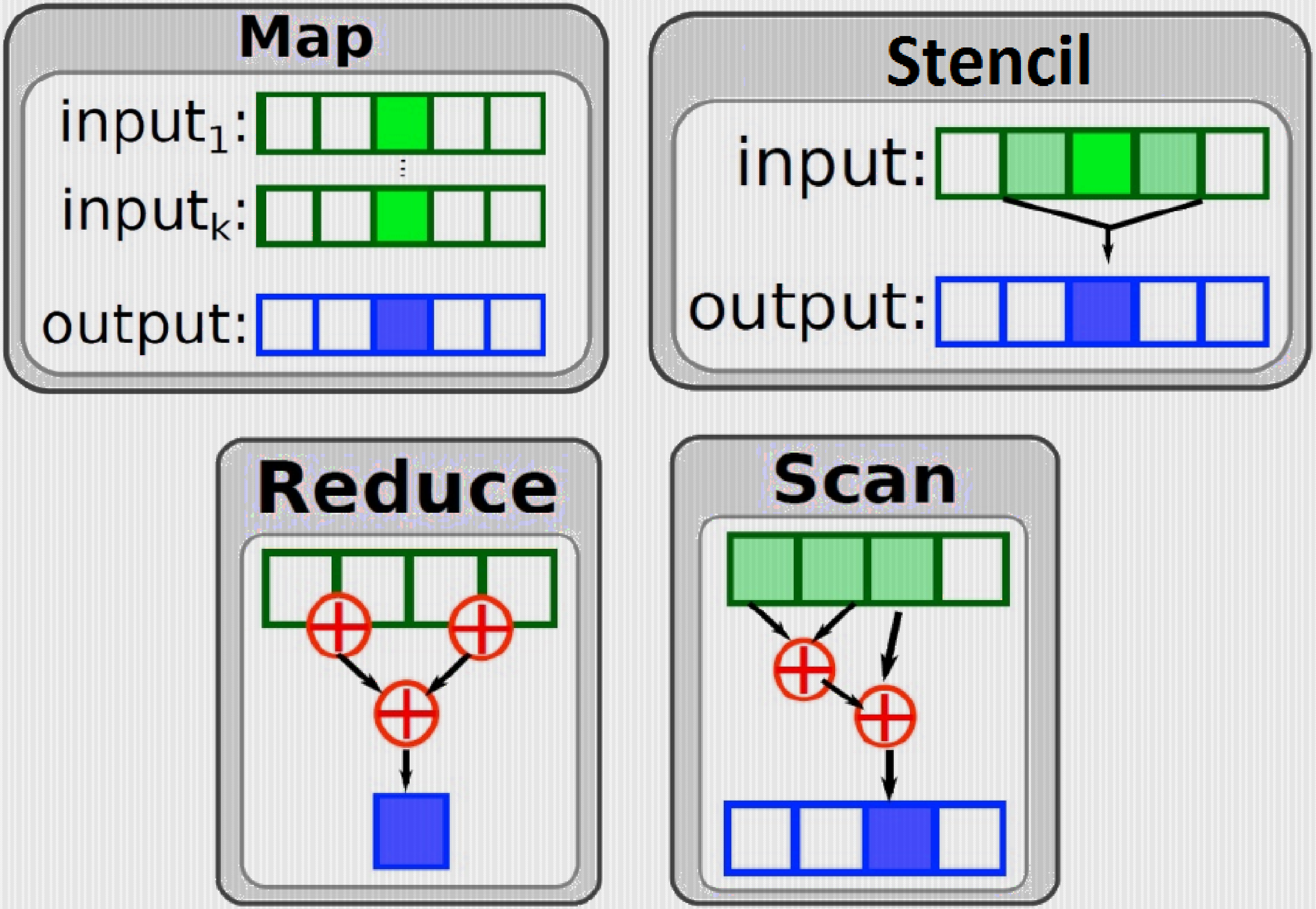


OpenCL

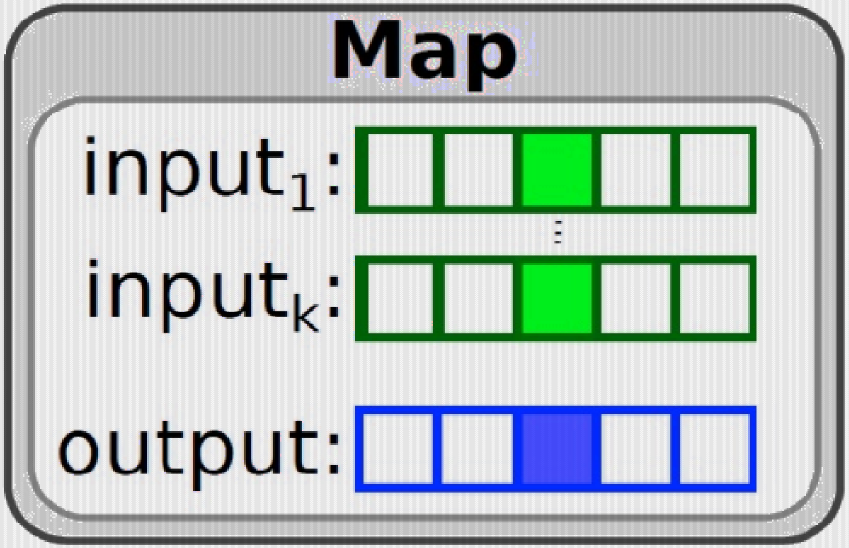
Outline

- Case studies on parallel design patterns
  - **Map**
  - (Gather)
  - (Scatter)
  - **Stencil**
  - **Reduce**
  - **Scan**

Parallel computing – Important design patterns

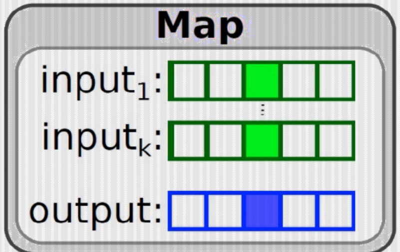


PP design pattern: Map

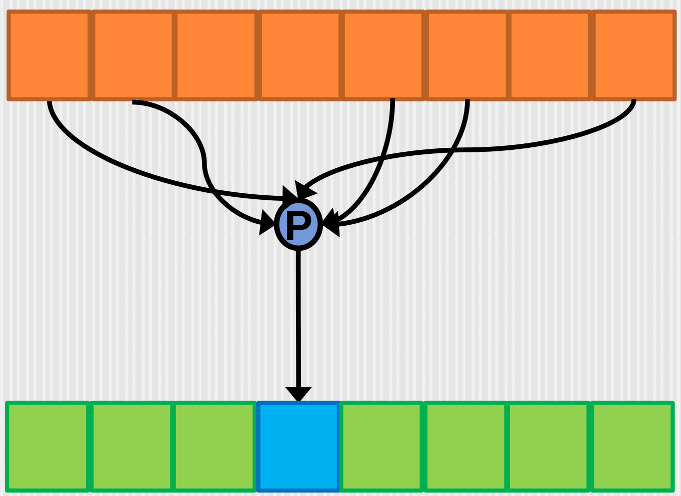


PP design pattern: Map

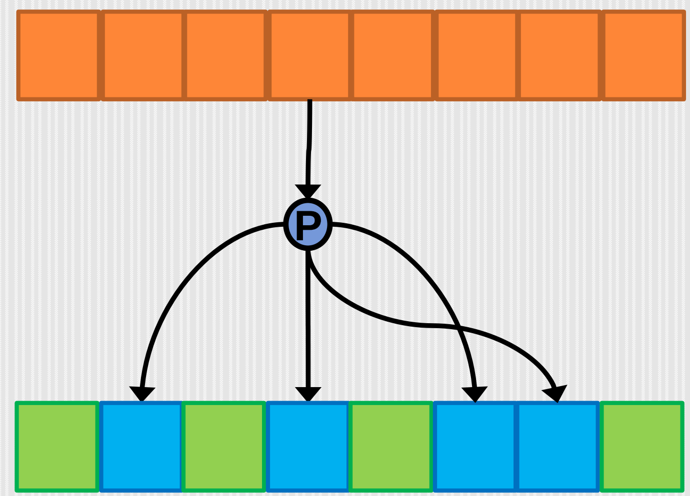
- Application of a multivariate function.
  - One-to-One mapping
- Typically combined with different patterns.
  - New patterns arise.
- About implementation
  - Could be carried in-place (on the input).



(Gather and Scatter)

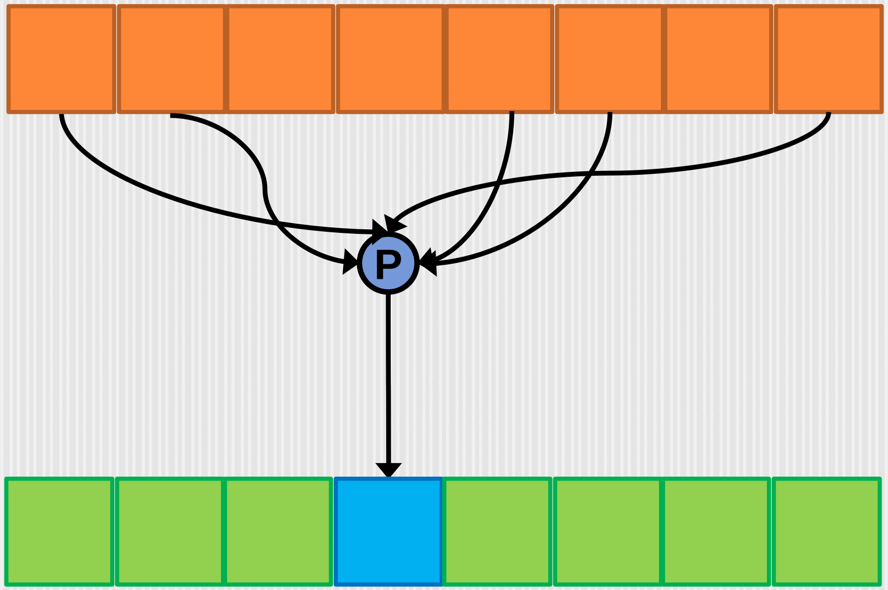


(Gather)



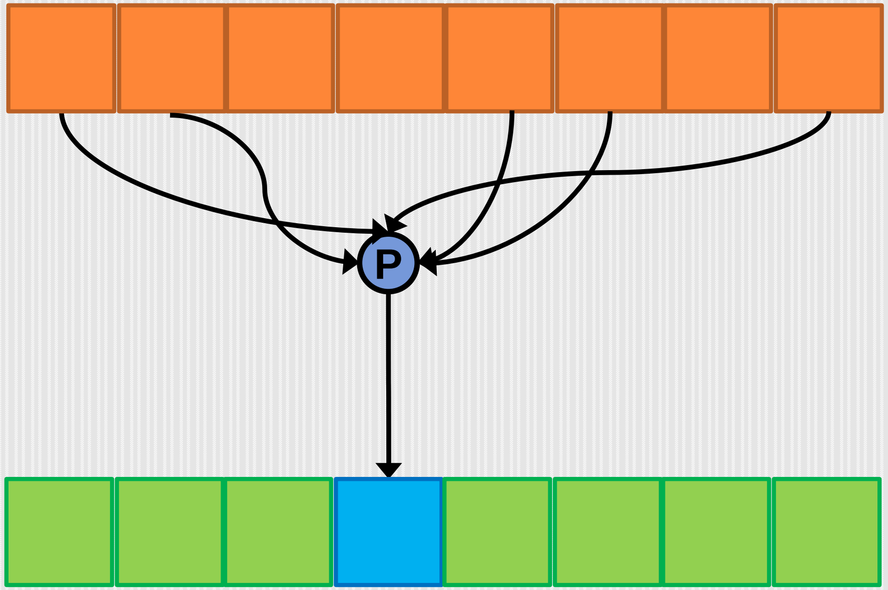
(Scatter)

(PP design pattern: Gather)

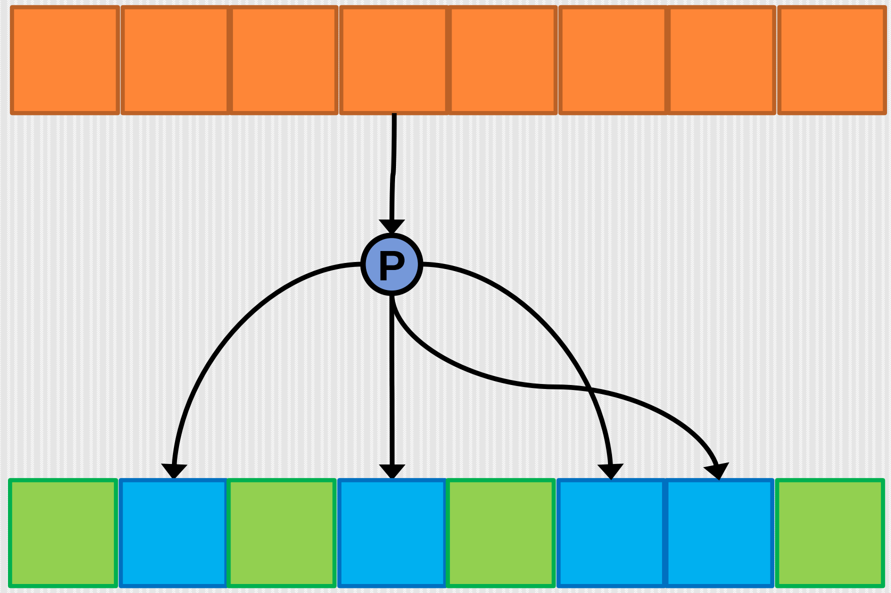


(PP design pattern: Gather)

Gather: Process „P" collects data from multiple locations and has a single unit of output.

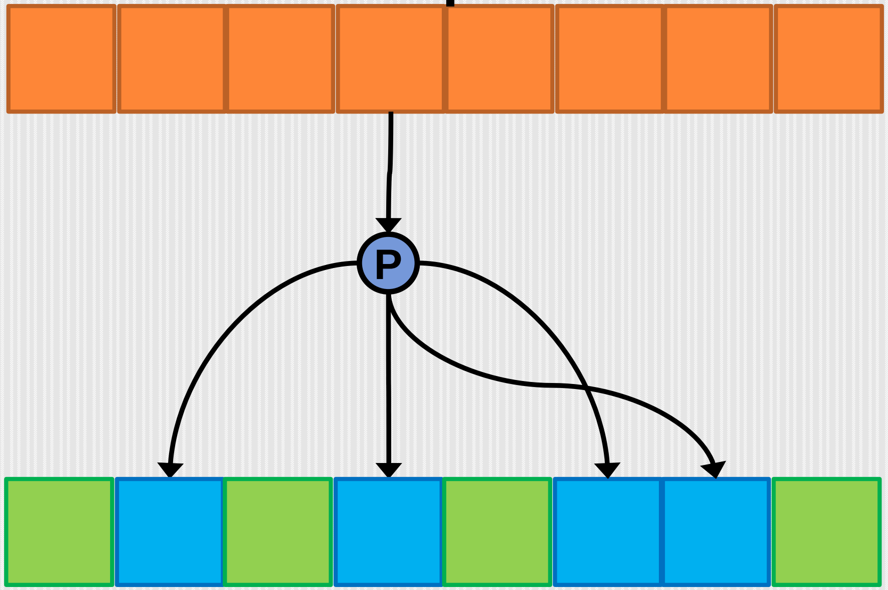


(PP design pattern: Scatter)

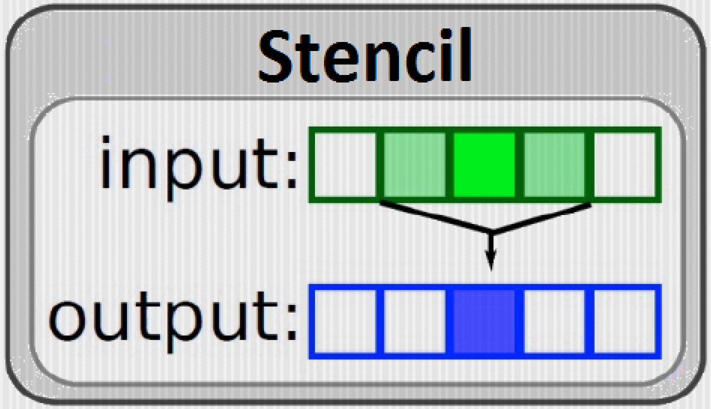


(PP design pattern: Scatter)

Scatter: Process „P" touches multiple output elements given a unit of input data.

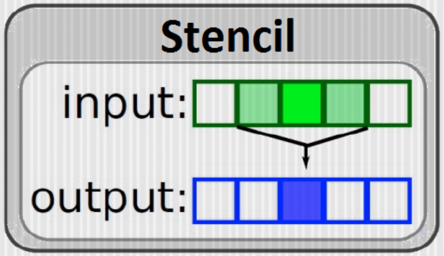


PP design pattern: Stencil (1)



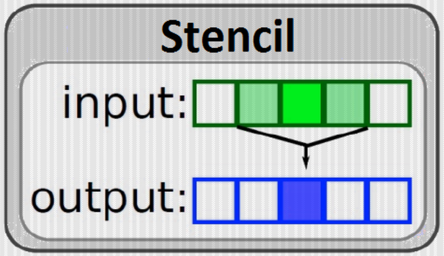
PP design pattern: Stencil (1)

- Overview
  - N input elements, N output elements.
  - Output is calculated based on some context of the element being processed.
  - 1D, 2D, 3D, ….
- Applications
  - E.g.: Running filter (**in-place**)
  - E.g.: Blurring (**not in-place**)
    - Box filter, Gaussian filter, …
- Techniques
  - Convolution, Median, Finite differences, Bilateral filtering, etc
- Implementation
  - Convolution: Separable? => Speedup!



PP design pattern: Stencil (2)

- Important aspects:
  - Stencil has a fixed pattern of input data access.
  - Stencil touches each output element.
- Problem of its naive implementation:
  - Data reuse. ☹
    - Q: A solution? E.g. for *runing mean*?
  - Task:
    - Q: How many times wouls stencil access the the same data?



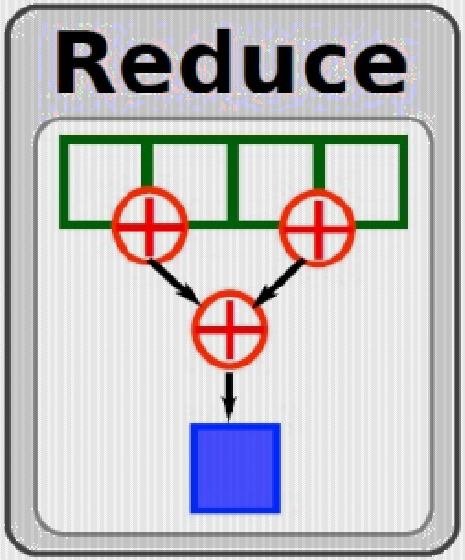
Task

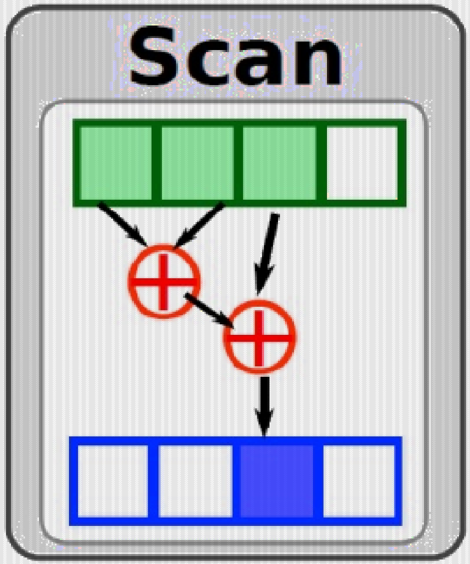
- Q: Based on what design pattern the following can work …
  - Summation?
  - Conditional assignment?
  - Sorting?

Overview

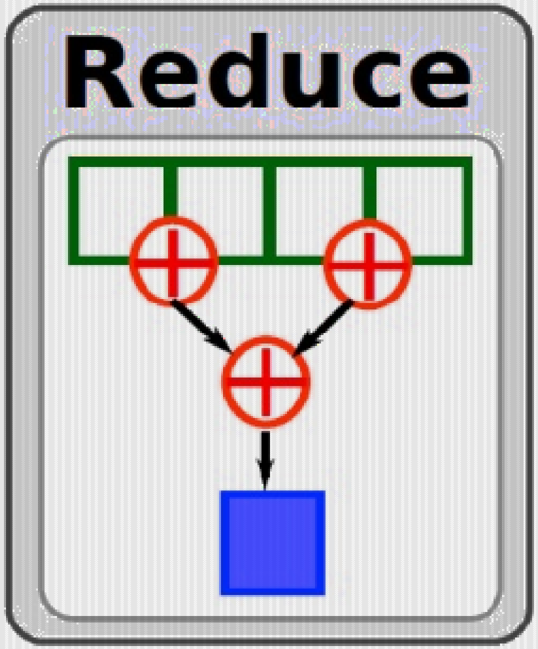
- **Map:**
  - Index space ←→ (Input, Output)
  - „One-to-One"
- **Gather:**
  - Index space ←→ Output
  - „Many-to-One"
- **Scatter:**
  - Index space ←→ Input
  - „One-to-Many"
- **Stencil:**
  - Index space ←→ (Input*, Output)
  - „Several-to-One"

Complex PP design patterns





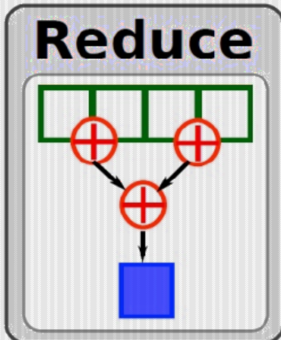
PP design pattern: Reduce (1)



PP design pattern: Reduce (1)

~"Compression"

- Given an operation: $\oplus$
  - Associative.
  - Commutative?
- Given Input where the operation can be used.
- The input is reduced **using the operation**:
  - E.g.: Sum (operation: +)
  - E.g.: Selection of maximum

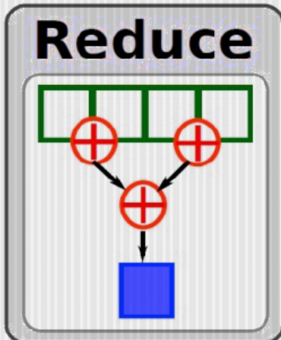


PP design pattern: Reduce (2)

Applications

- Summation, Maximum-selection ...

- **Only-Associative** case: A stage of the **Scan** design pattern! („Up-Sweep”)

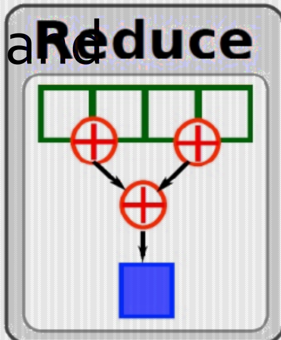


PP design pattern: Reduce (3)

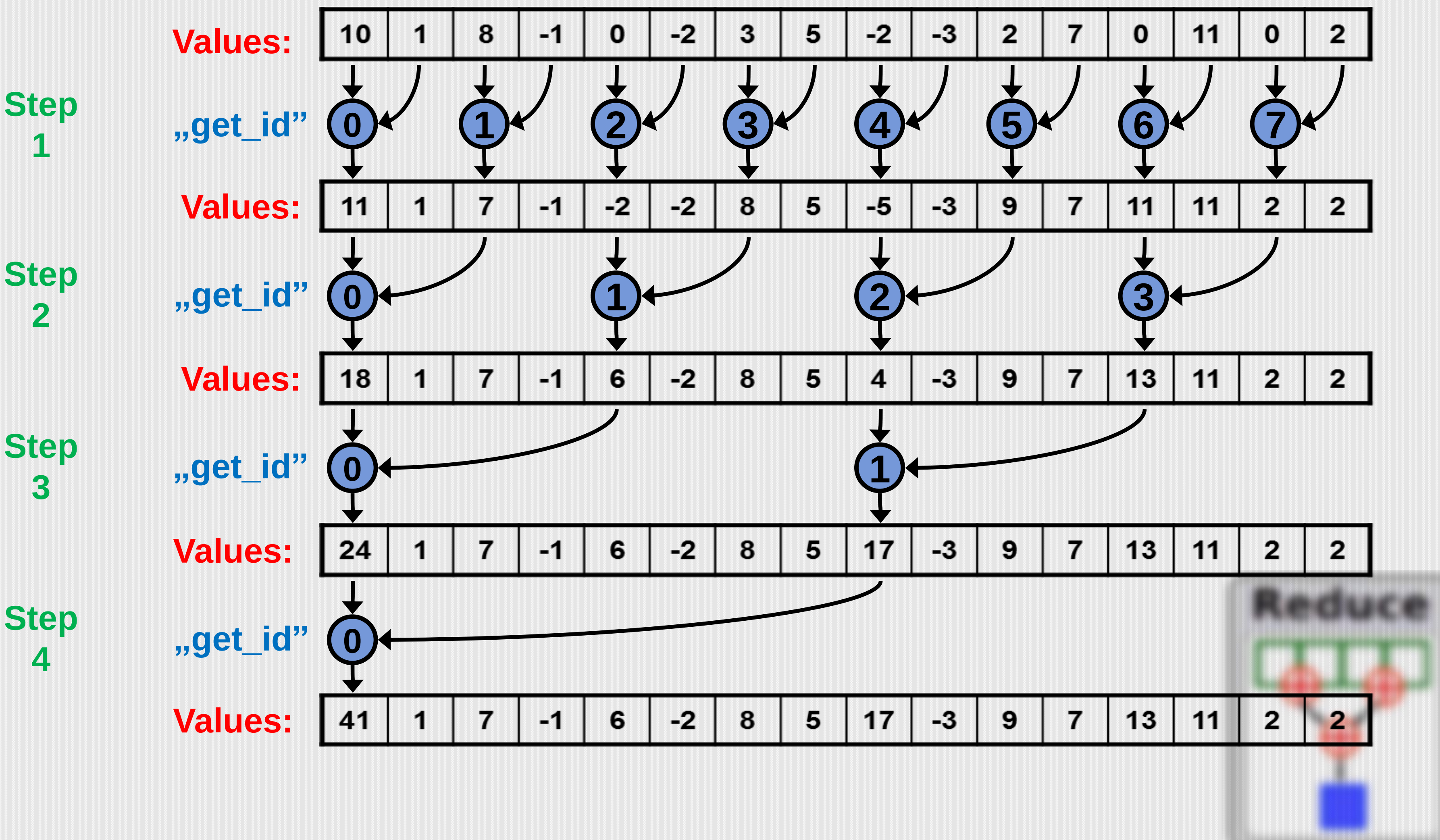
Implementation

- In-place, or using „alternated" Input-Output arrays.
- **Synchronization** is needed for efficiency.

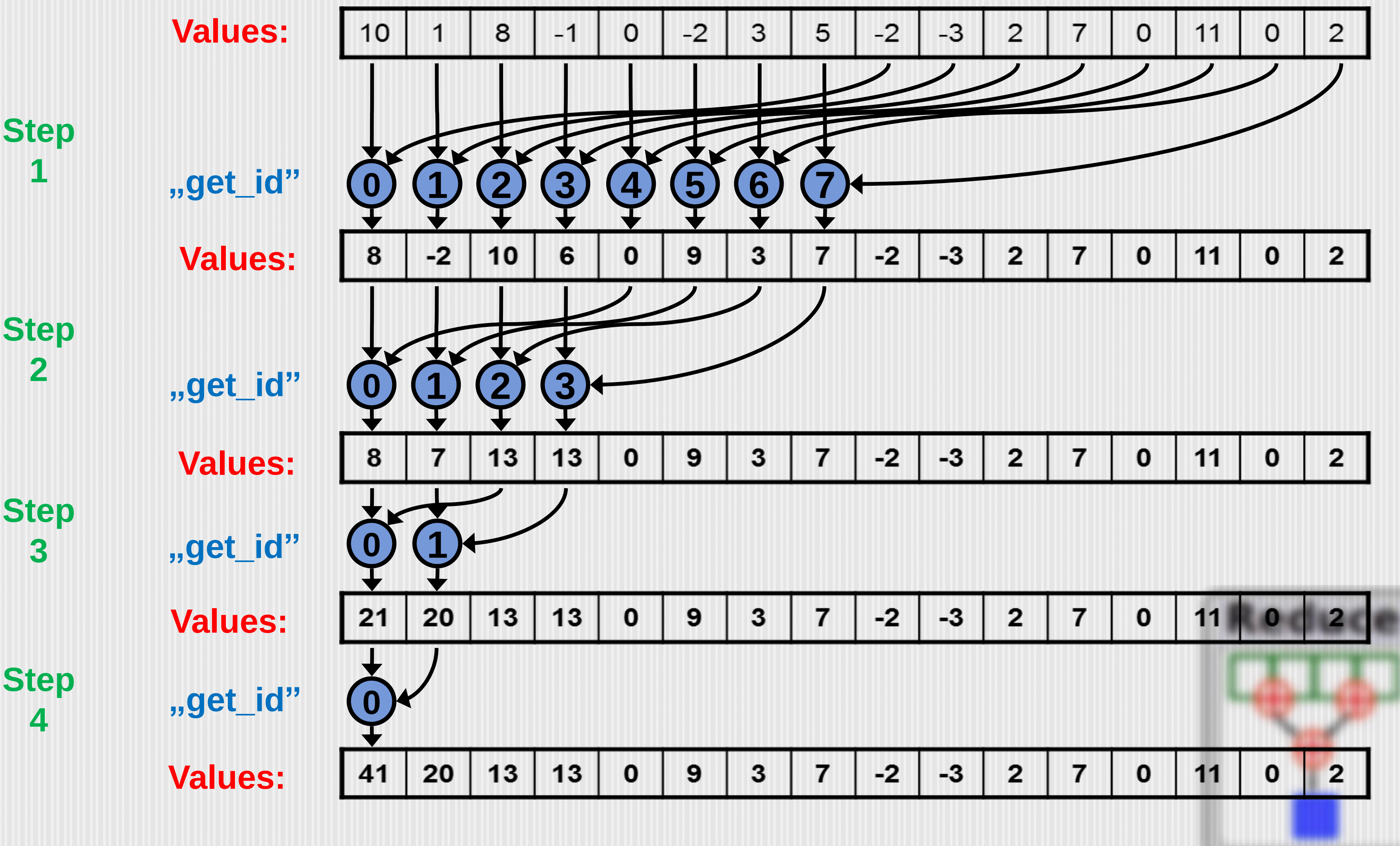
- Regarding the properties of the operation:
  - **Associative**: Base-case.
  - **Commutative**: Efficient memory and cache usage and other benefits. Con: order of data changes.



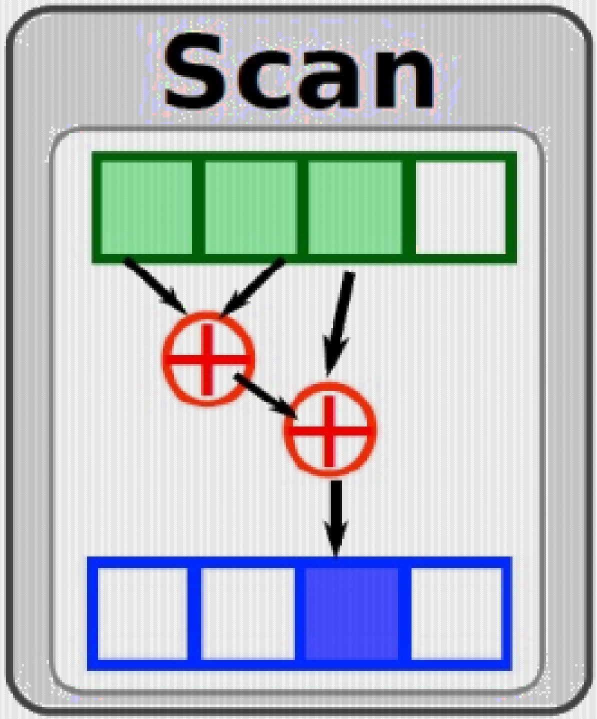
PP design pattern: Reduce (4)
Associative case



PP design pattern: Reduce (5)
Commutative case

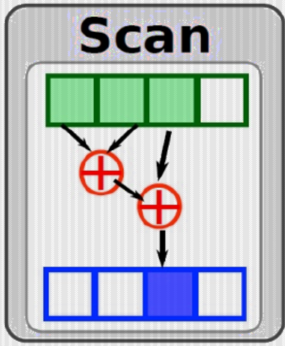


PP design pattern: Scan (1)



PP design pattern: Scan (1)

- Given operation: $\oplus$
  - Associative.
- The partial results are also computed.
  - Comparison: What „Reduce” is all about is the end result. But!



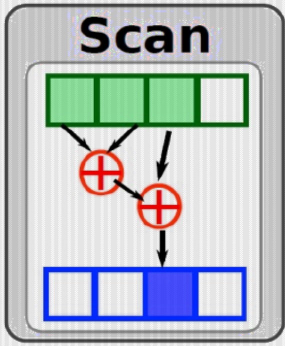
PP design pattern: Scan (2)

Applications

- A step of Radix sort.
- Variable width image filtering.
- Stream processing.

Read:

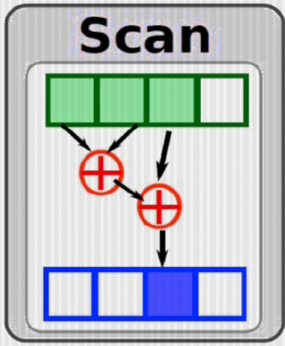
- Blelloch, Guy E. 1990. "Prefix Sums and Their Applications." Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University.
- http://people.inf.elte.hu/hz/parh/parhprg.html



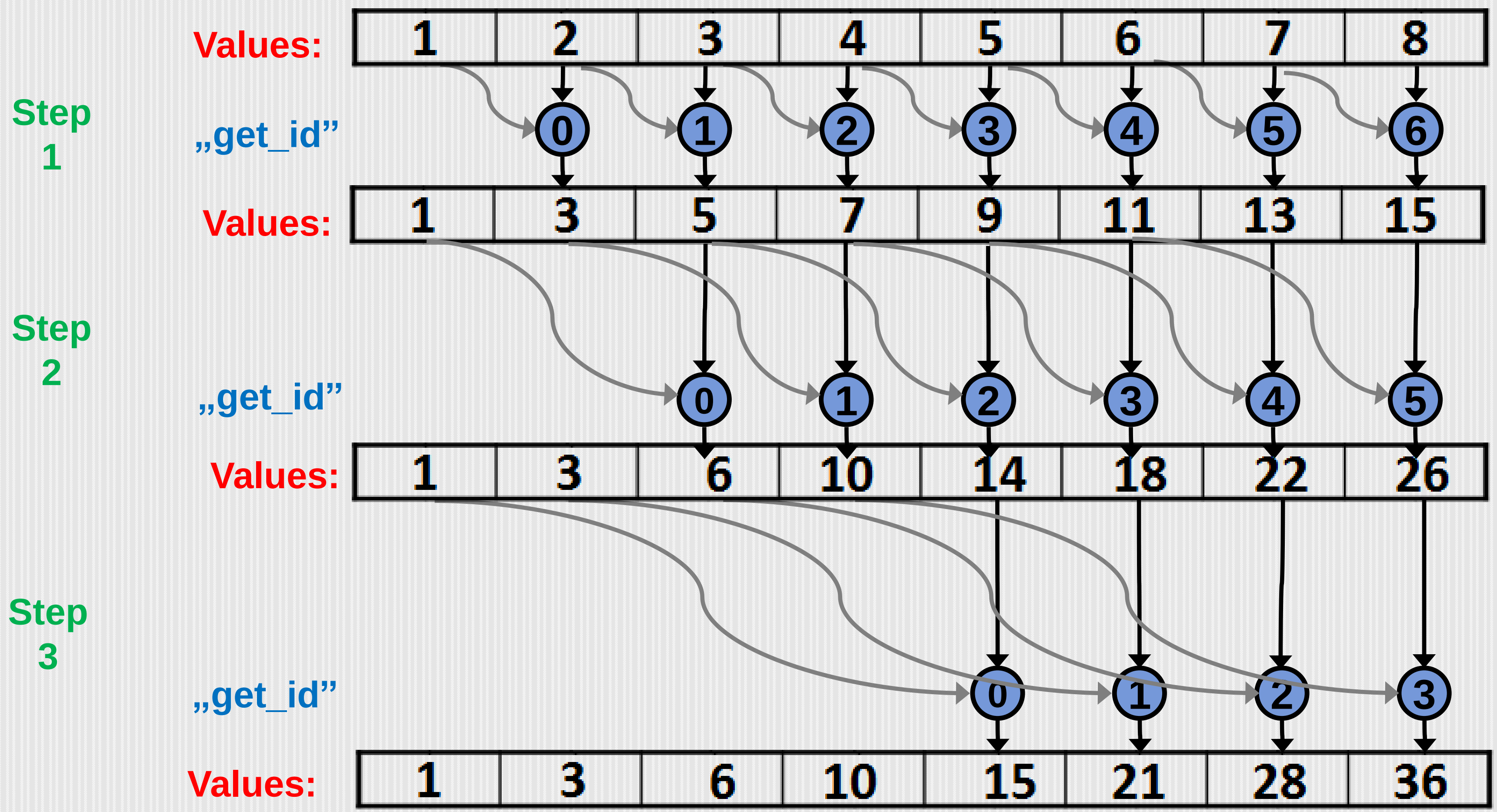
PP design pattern: Scan (3)

Implementation

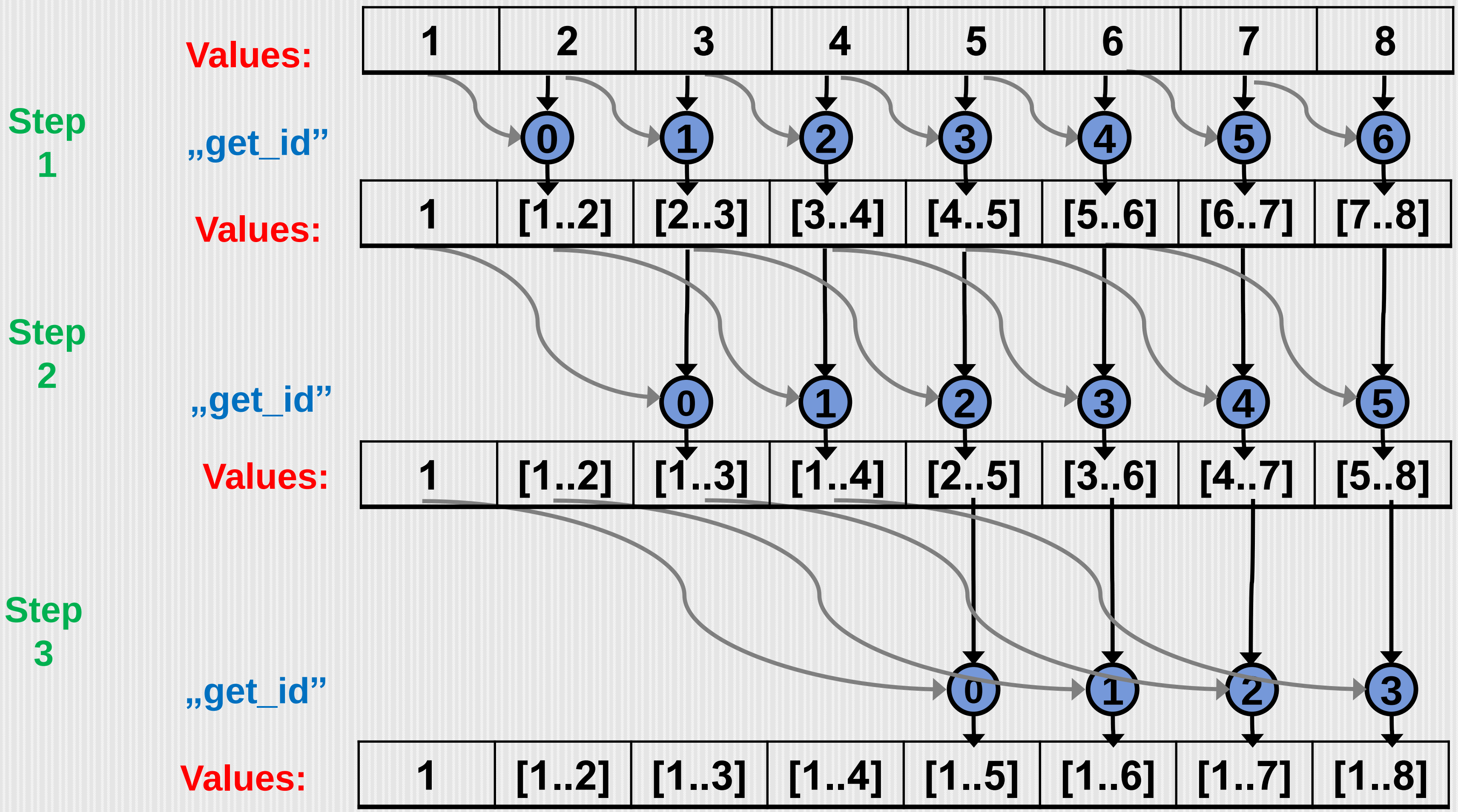
- Naive: not „work-efficient”: $O(n \log_2 n)$
- Efficient:
  - *balanced trees*
    - Really useful patterin in PP!!!
    - Here, binary tree.
  - Work-efficiency: $O(n)$.
  - **The binary tree is not stored, only the principle is used!**
  - Two Steps:
    - „Up-Sweep” (Reduce pattern)
    - „Down-Sweep”
  - Read: Belloch (1990)



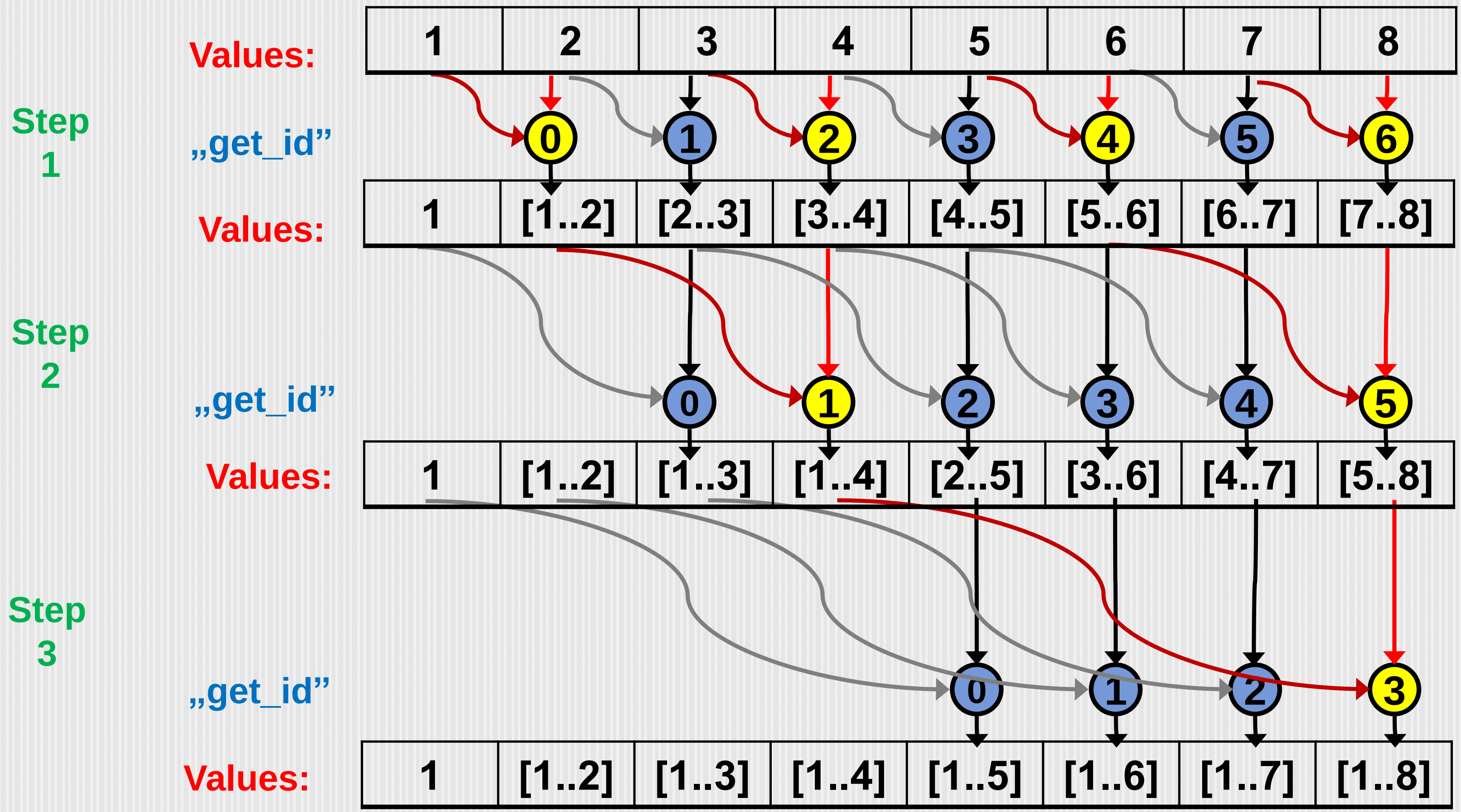
PP design pattern: Scan (4.1)
Naive approach– *O(n log(n))*



PP design pattern: Scan (4.1)
Naive approach– *O(n log(n))*



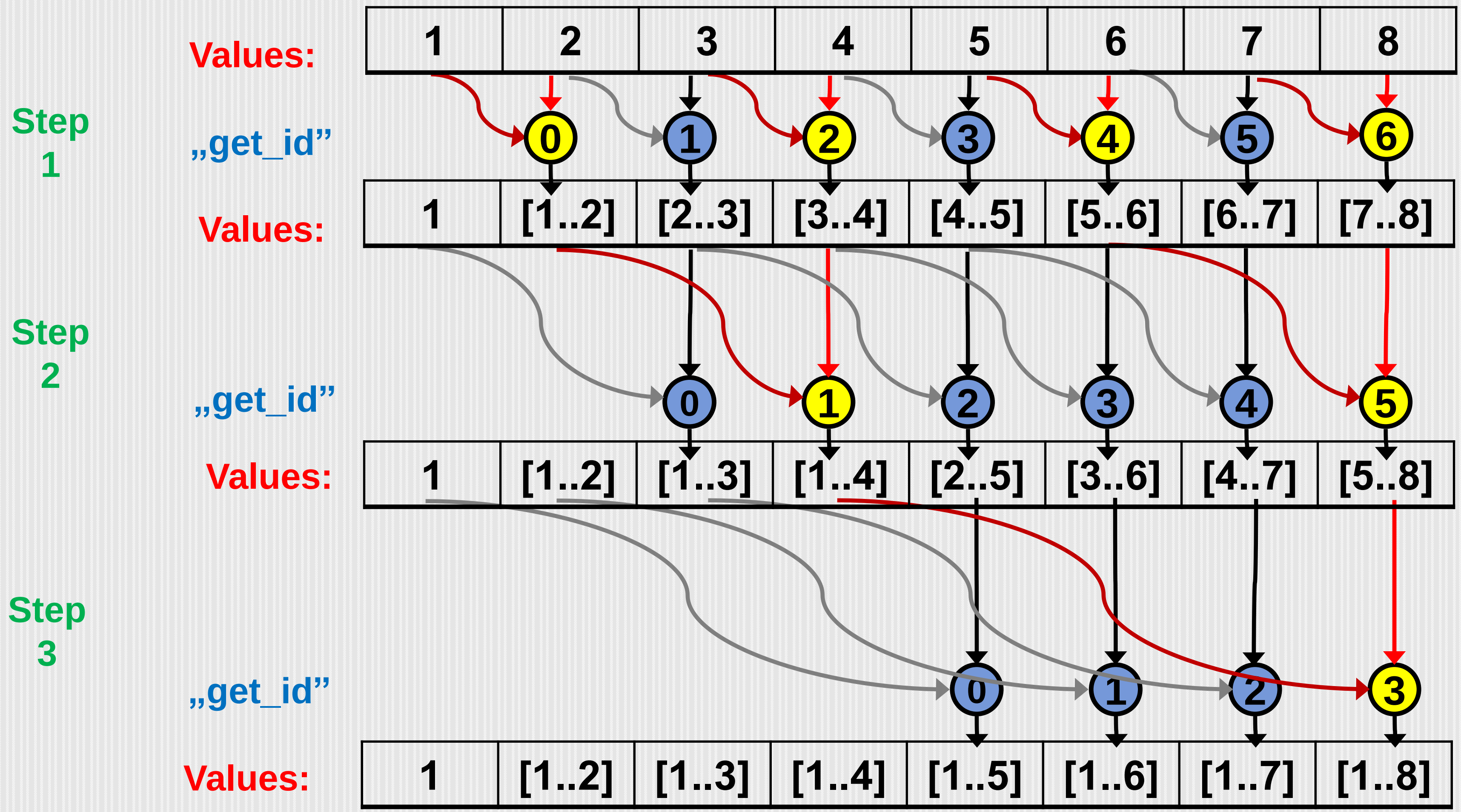
PP design pattern: Scan (4.1)
Naive approach– *O(n log(n))*



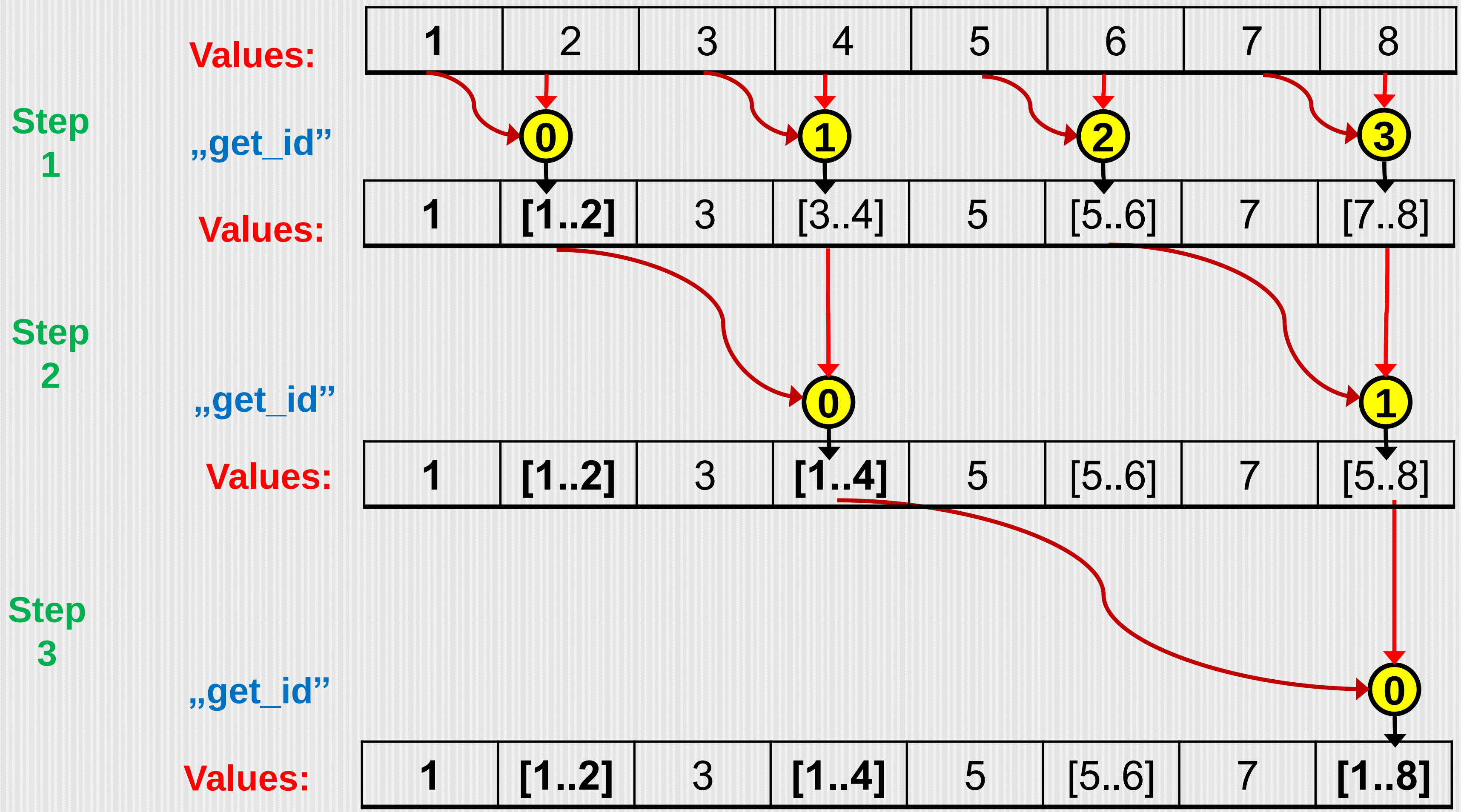
PP design pattern: Scan (4.2)
Efficient – $O(n)$

- Two steps:
  - Up-Sweep
  - Down-Sweep
- **Up-Sweep**
  - Associative! „Reduce”.
- **Down-Sweep**
  - Look at the next slides!

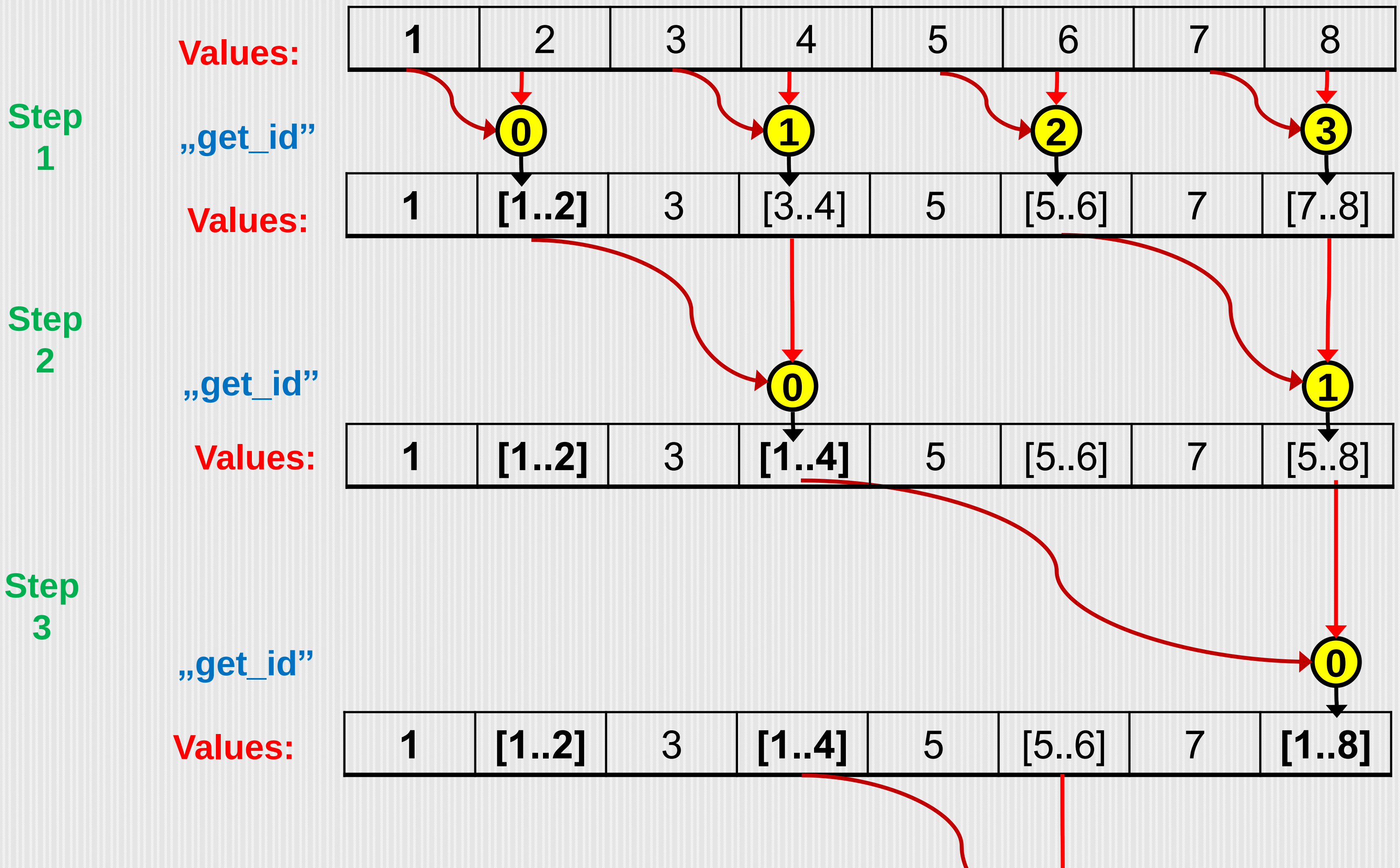
PP design pattern: Scan (4.1)
Naive approach– *O(n log(n))*



PP design pattern: Scan (4.1) Work-efficiency– *Down-sweep*



PP design pattern: Scan (4.1)
Work-efficiency– *Down-sweep*



PP design pattern: Scan (4.1)
Work-efficiency– „*Up*”-*sweep*