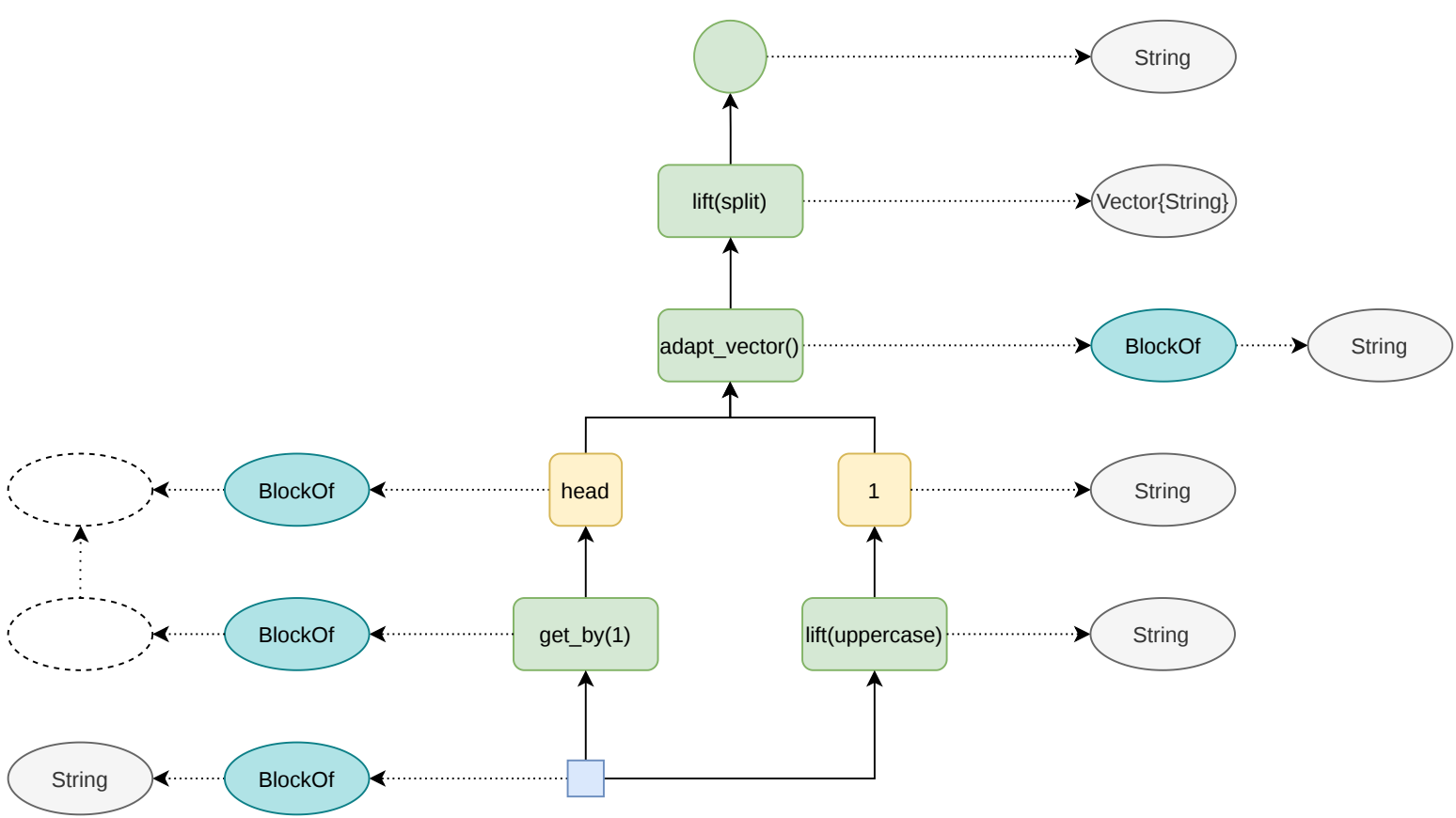




The diagram illustrates the transformation of a flat table into a hierarchical tree structure through a series of steps:

- Initial Flat Table:** A single table with two columns: an index (1) and a value ("Hello World").
- String Splitting:** The value "Hello World" is split into an array of strings: ["Hello", "World"].
- Indexing and Splitting:** The array is indexed (1: "Hello", 2: "World"). A new root node is created, branching into two child nodes corresponding to the indexed elements.
- Case Folding:** The string values are converted to uppercase: "HELLO" and "WORLD".
- Final Hierarchical Structure:** The final tree structure consists of a root node branching into two child nodes. The left child node has two indexed elements (1: "HELLO", 2: "WORLD"), and the right child node has one indexed element (1: "HELLO").



wrap()



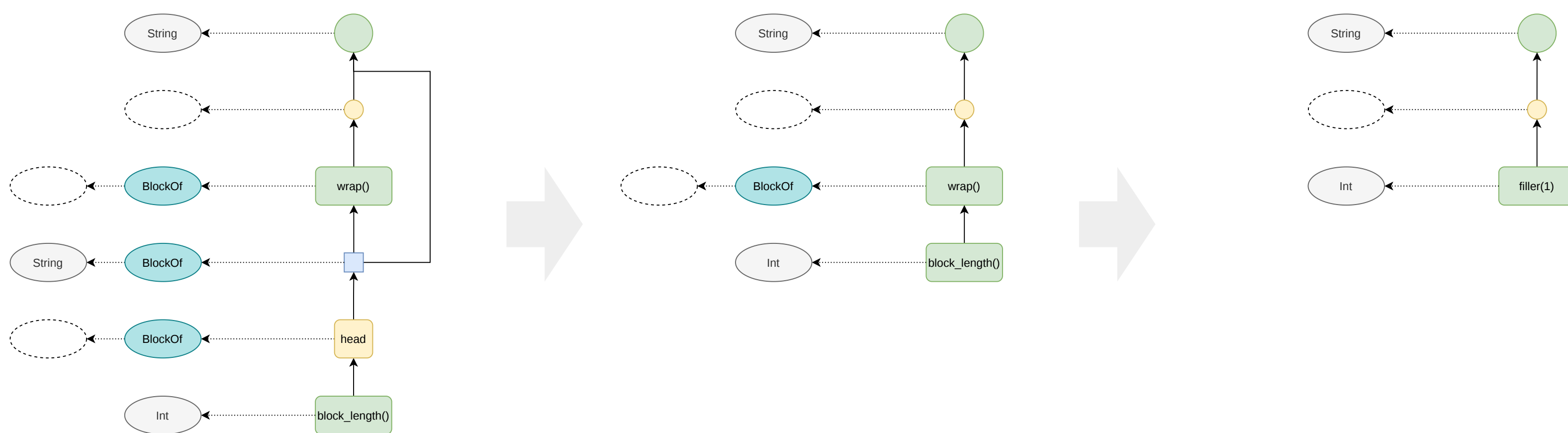
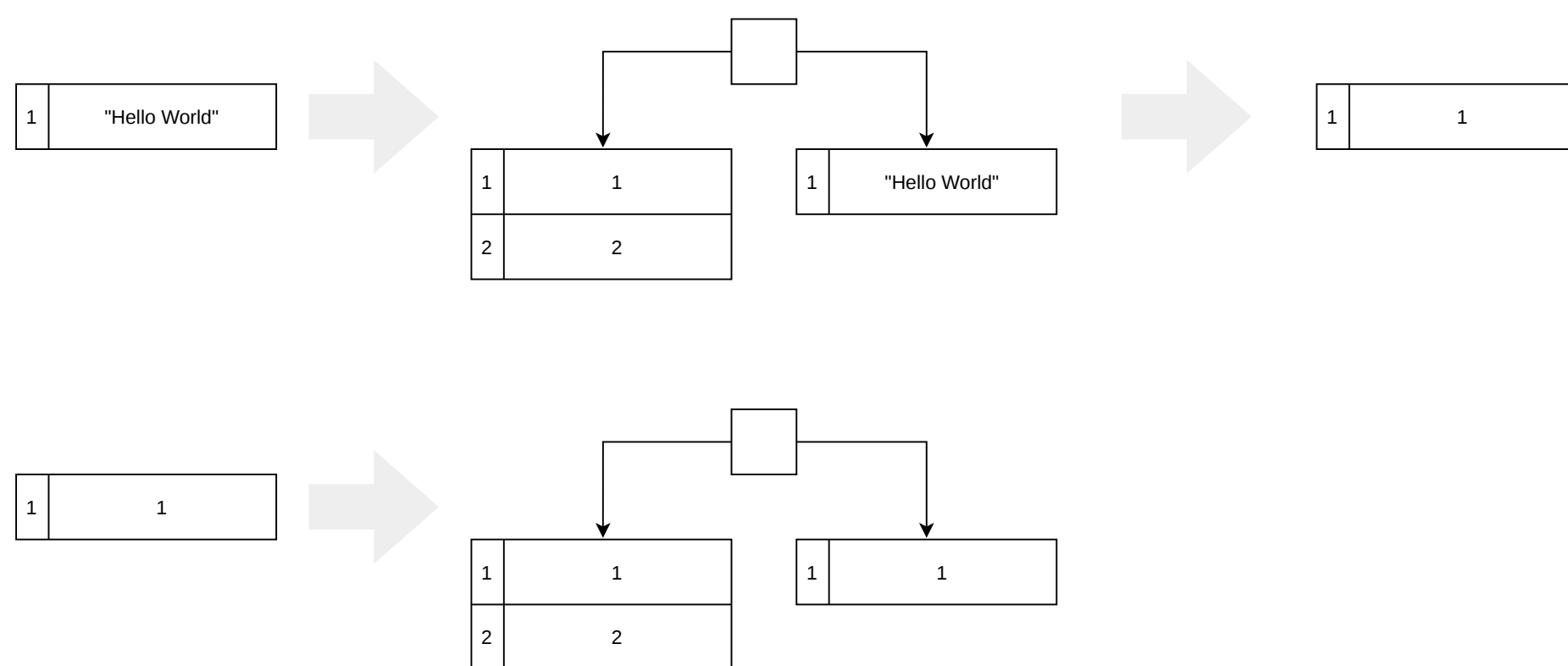
chain_of(tuple_of(2), column(1))



sieve_by0



```
chain_of(wrap(), block_length())
```



@query "Hello World" split(it)

```
chain_of(
  wrap(),
  with_elements(
    chain_of(
      wrap(),
      lift(split),
      adapt_vector()),
    flatten())
```



untrace(n::NodeRef, guard::NodeRef)::Tuple{Pipeline,Vector{NodeRef}}





@query "Hello World" split(it)

```
chain_of(
  wrap(),
  with_elements(
    chain_of(
      wrap(),
      lift(split),
      adapt_vector()),
    flatten())
```







`untrace(n::NodeRef, guard::NodeRef)::Tuple{Pipeline,Vector{NodeRef}}`

`{it ^ 2, (it ^ 2) ^ 3}`

`chain_of(f, tuple_of(g, h)) => tuple_of(chain_of(f, g), chain_of(f, h))`



`untrace(n::NodeRef, guard::NodeRef)::Tuple{Pipeline,Vector{NodeRef}}`

@query "Hello World" split(it)

```
chain_of(
  wrap(),
  with_elements(wrap()),
  with_elements(lift(split)),
  with_elements(adapt_vector()),
  flatten())
```

