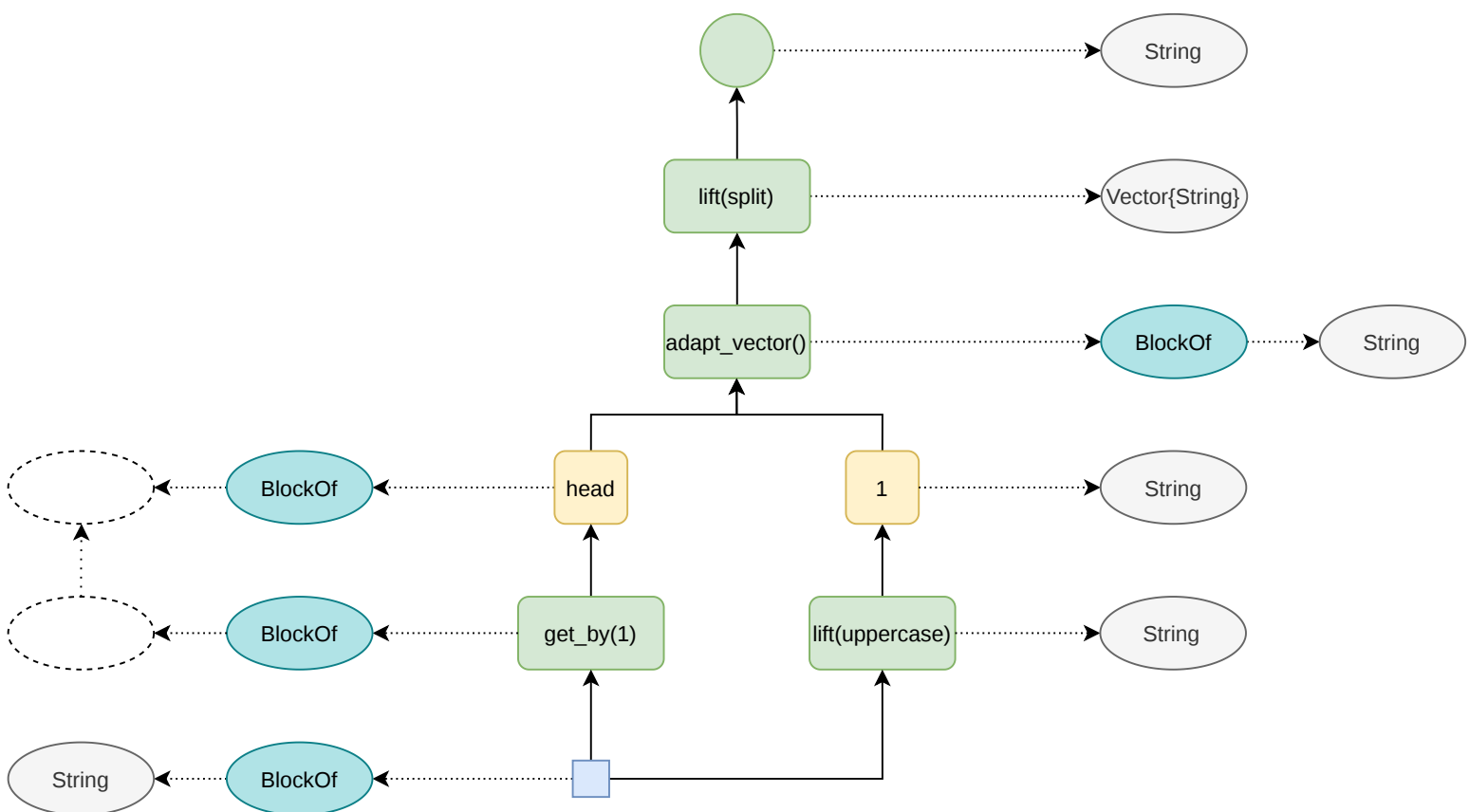


The diagram illustrates the transformation of a flat table into a hierarchical tree structure through four stages, connected by large grey arrows:

- Stage 1:** A flat table with two columns: an index column containing '1' and a data column containing 'Hello World'.
- Stage 2:** The data is split into two rows based on a hidden pivot point. The first row has index '1' and data '1'. The second row has index '2' and data '3'. A separate node contains the original data split into two rows: index '1' with data 'Hello' and index '2' with data 'World'.
- Stage 3:** The data is further processed. The first row now has index '1' and data '1'. The second row has index '2' and data '3'. The separate node now has index '1' with data 'HELLO' and index '2' with data 'WORLD'.
- Stage 4:** The final hierarchical structure. The first row has index '1' and data '1'. The second row has index '2' and data '2'. The separate node now has index '1' with data 'HELLO' and index '2' with data 'WORLD'.



wrap()



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



sieve_by()

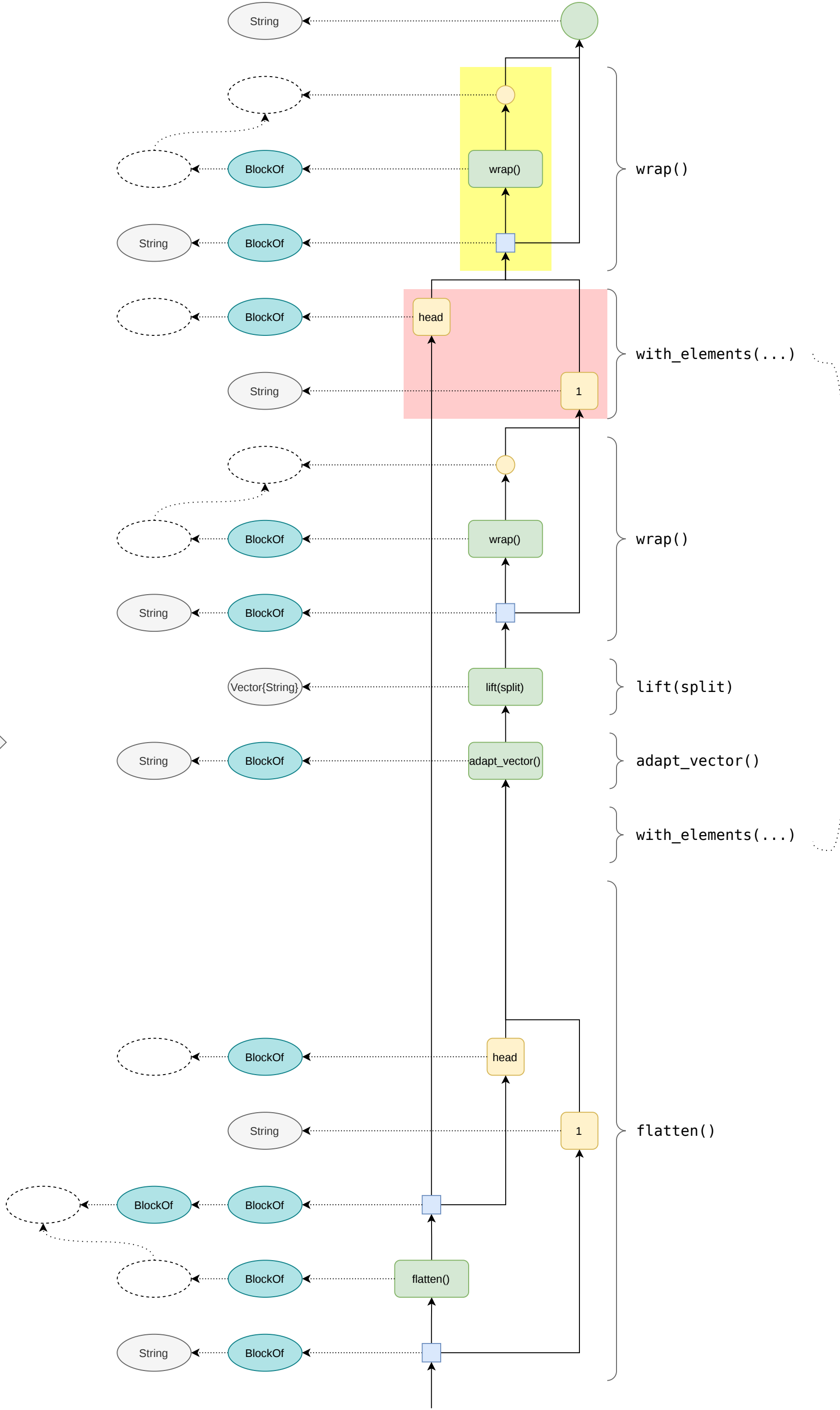


chain_of(wrap(), block_length())



@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())
```

