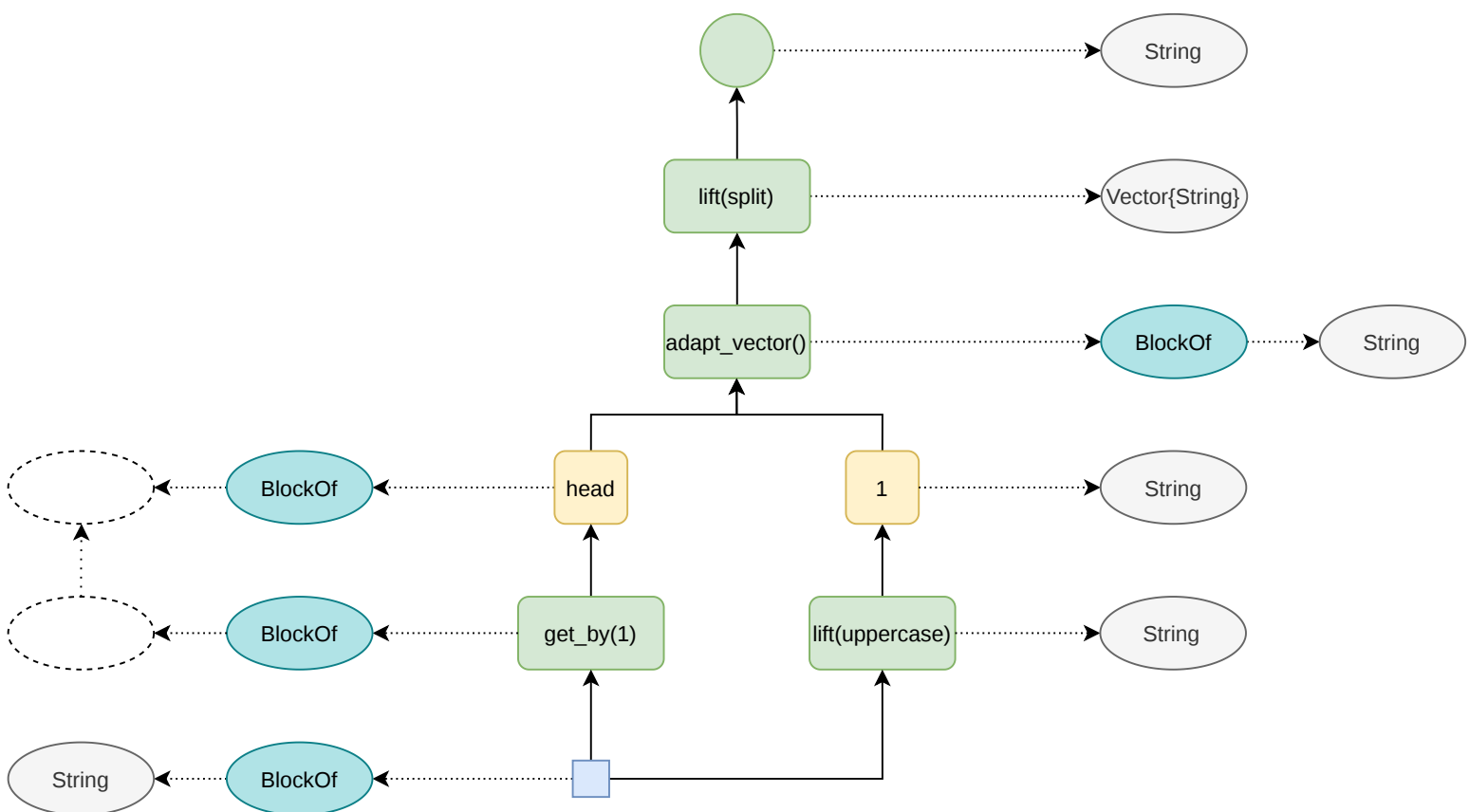




The diagram illustrates the transformation of a flat table into a hierarchical tree structure through four stages, connected by large grey arrows:

- Stage 1:** A flat table with two columns: an index column (1) and a value column ("Hello World").
- Stage 2:** The table is transformed into a hierarchical structure. The root node (empty box) branches into two child nodes. The left child is a table with index 1 and value 1. The right child is a table with index 1 and value "Hello", and index 2 and value "World".
- Stage 3:** The structure is further transformed. The root node branches into two child nodes. The left child is a table with index 1 and value 1, and index 2 and value 3. The right child is a table with index 1 and value "HELLO", and index 2 and value "WORLD".
- Stage 4:** The final structure is a hierarchical tree. The root node branches into two child nodes. The left child is a table with index 1 and value 1, and index 2 and value 2. The right child is a table with index 1 and value "HELLO".



wrap()



chain\_of(tuple\_of(2), column(1))



sieve\_by()

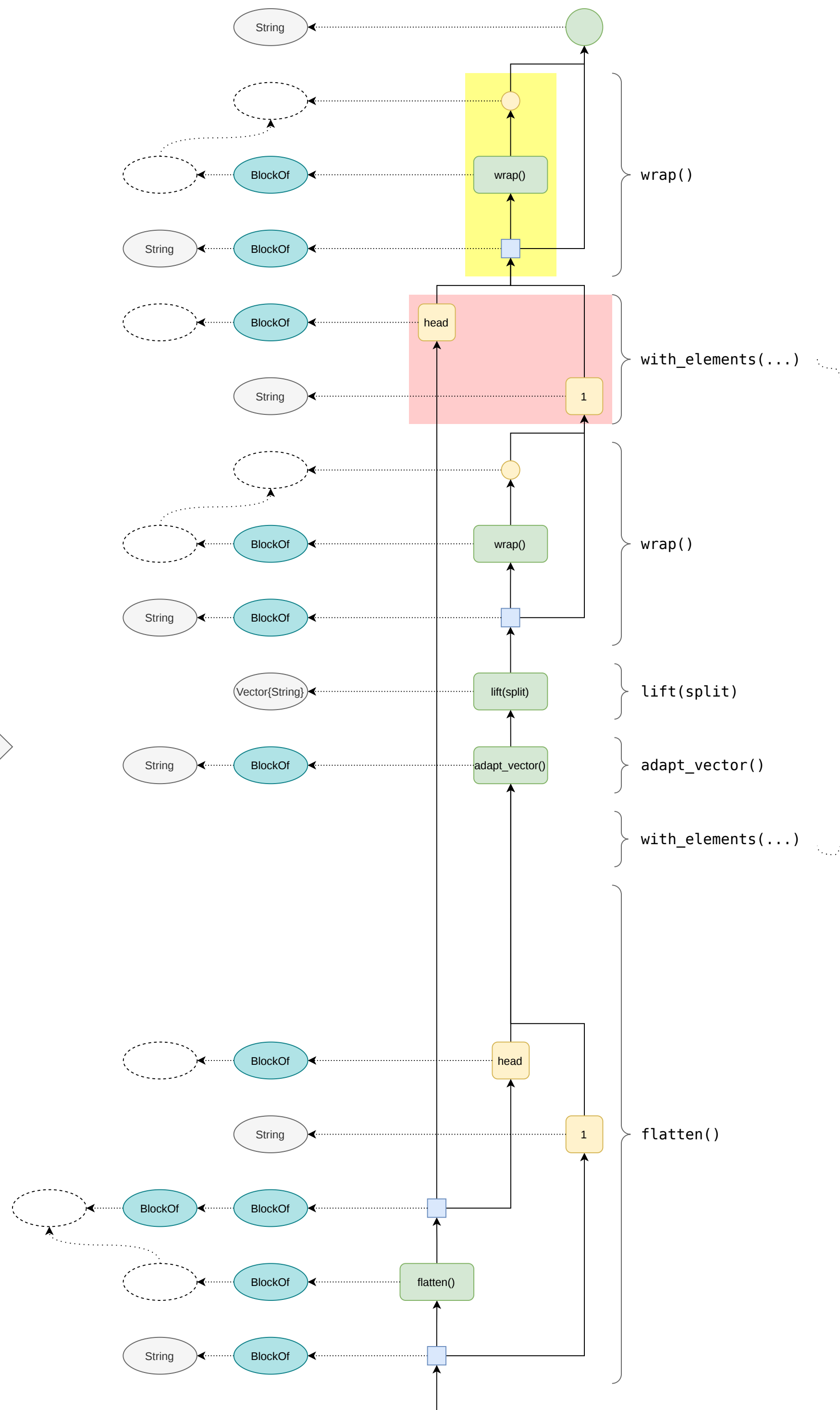
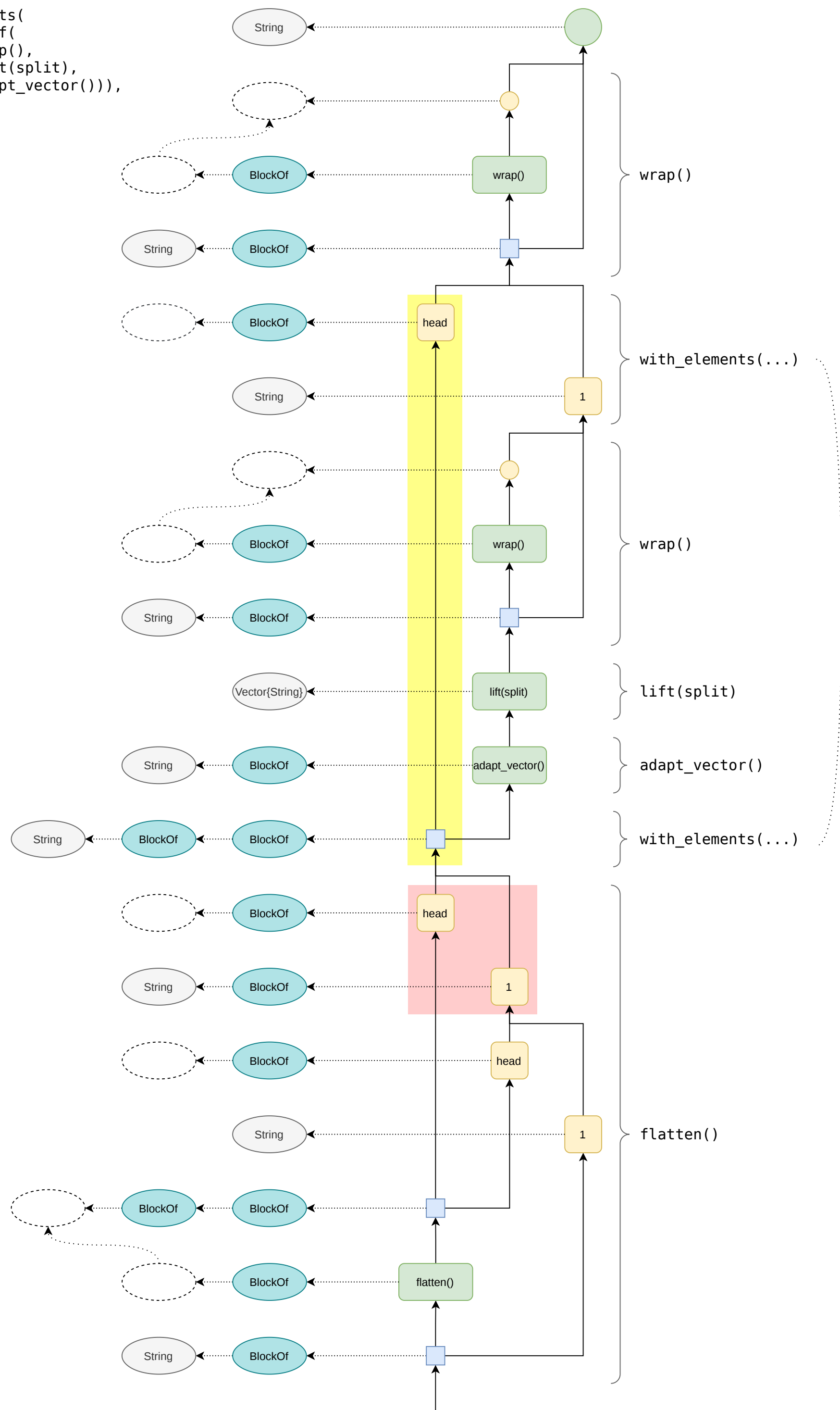


chain\_of(wrap(), block\_length())



```
@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())
```



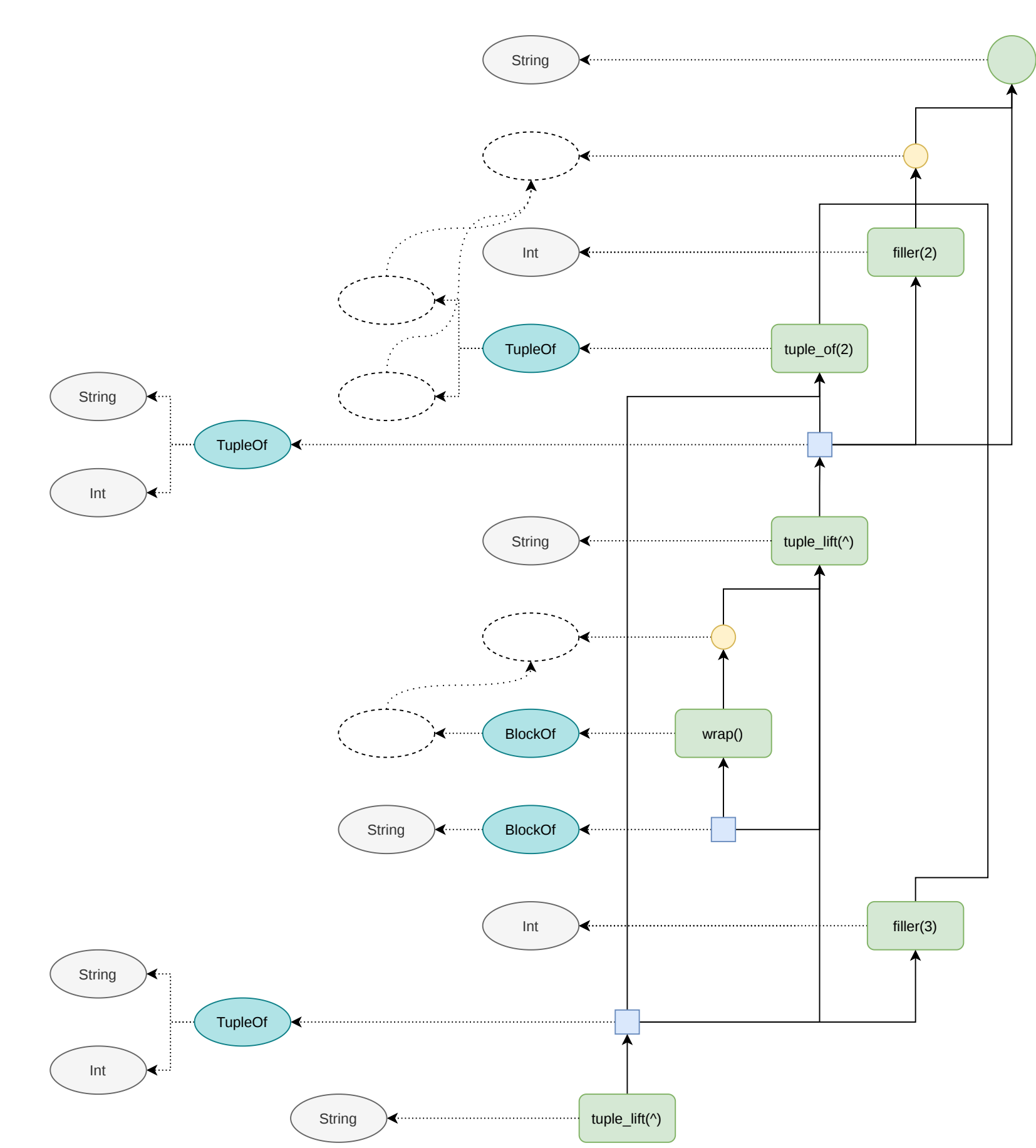




`untrace(n::NodeRef, guard::NodeRef)::Tuple{Pipeline,Vector{NodeRef}}`

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

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



`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())
```

