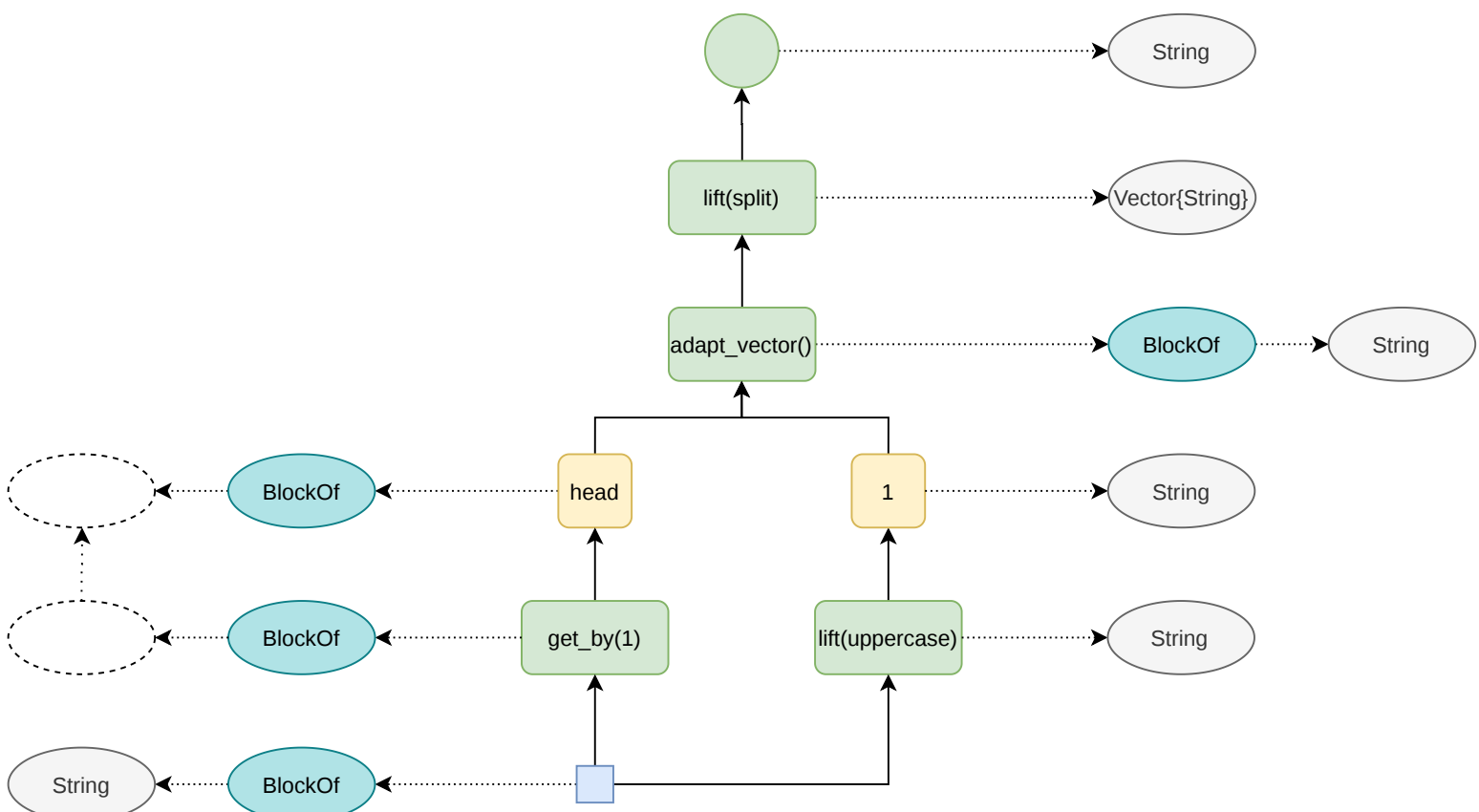
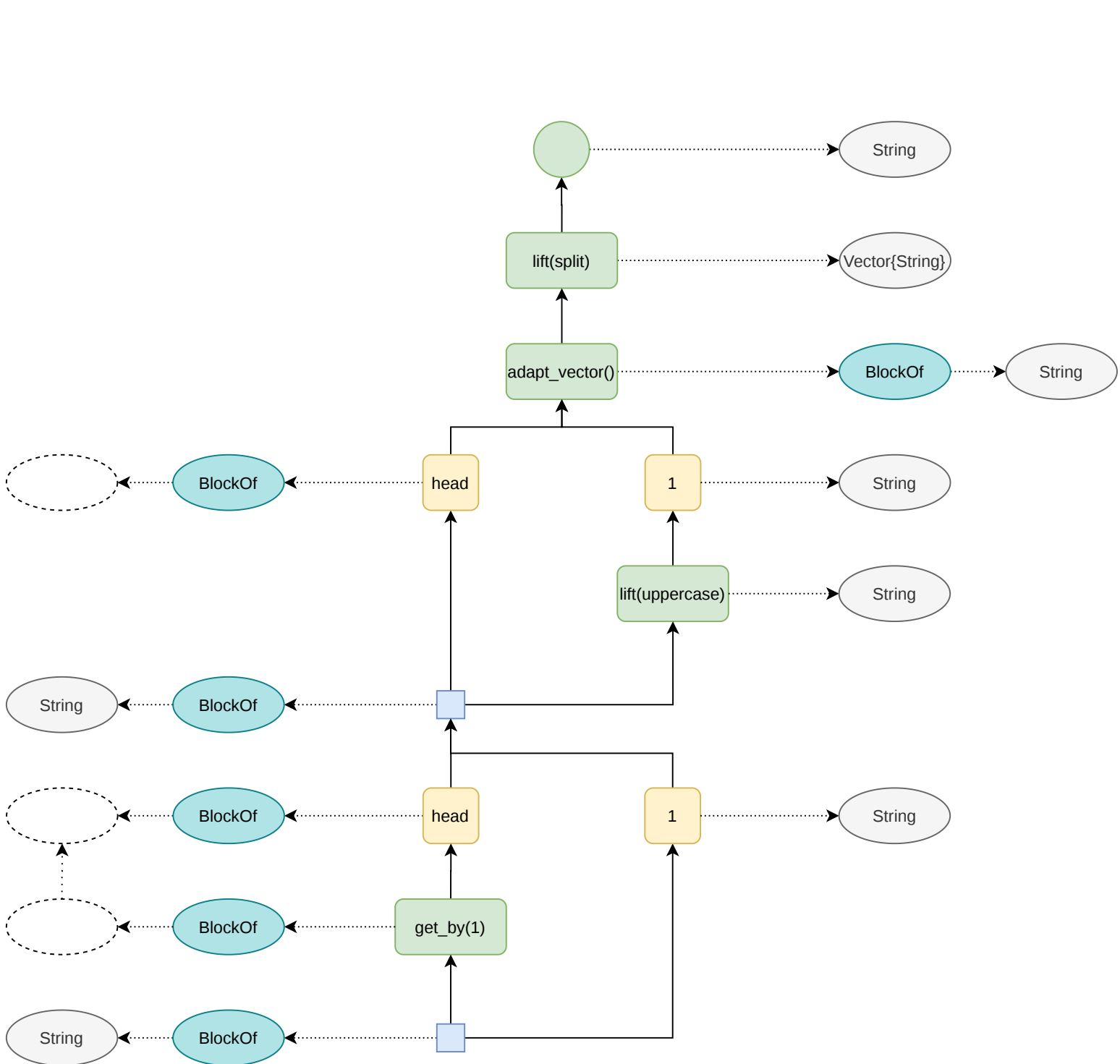
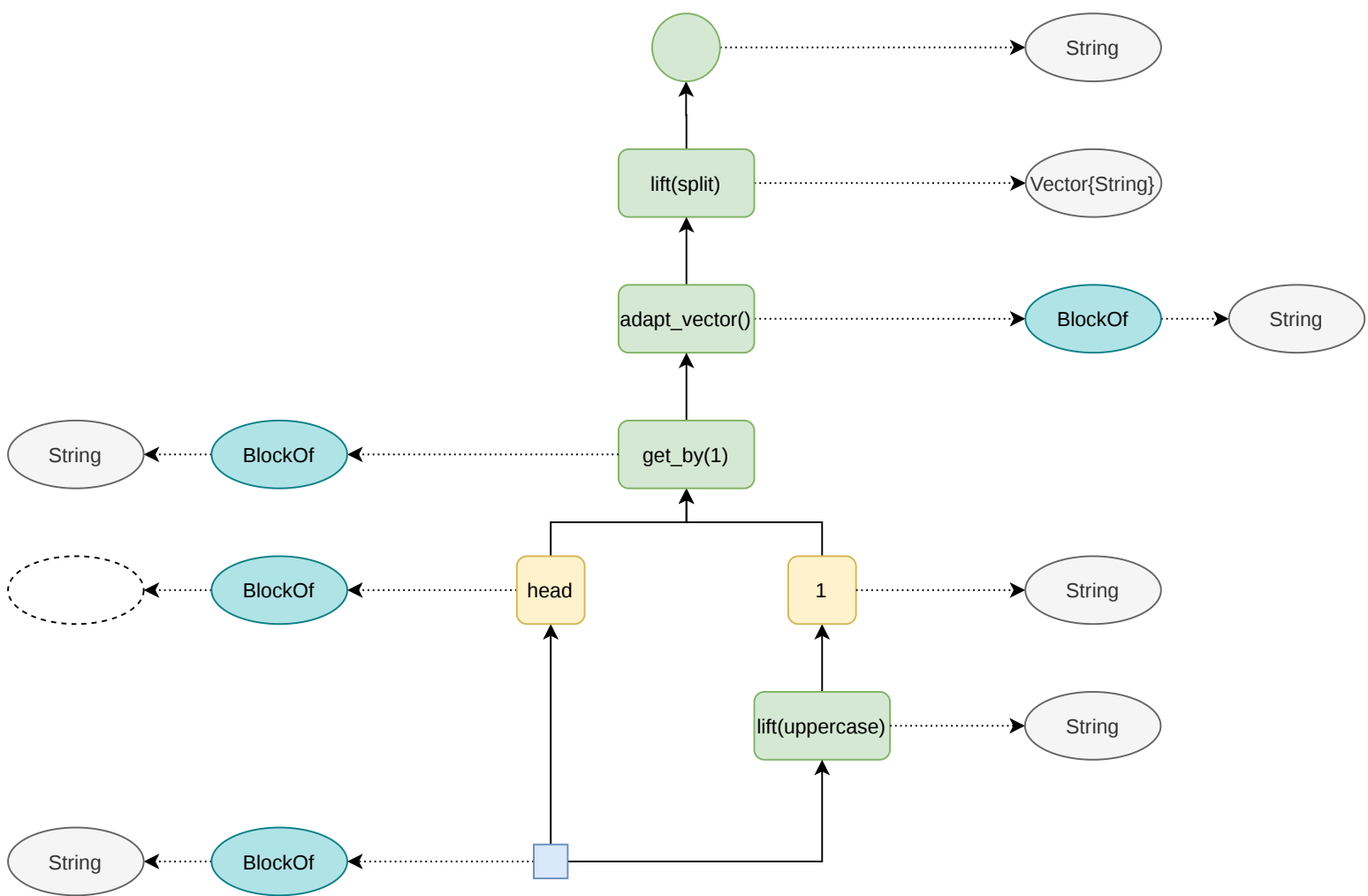


The diagram illustrates the transformation of a flat table into a hierarchical tree structure through four stages, connected by large gray arrows:

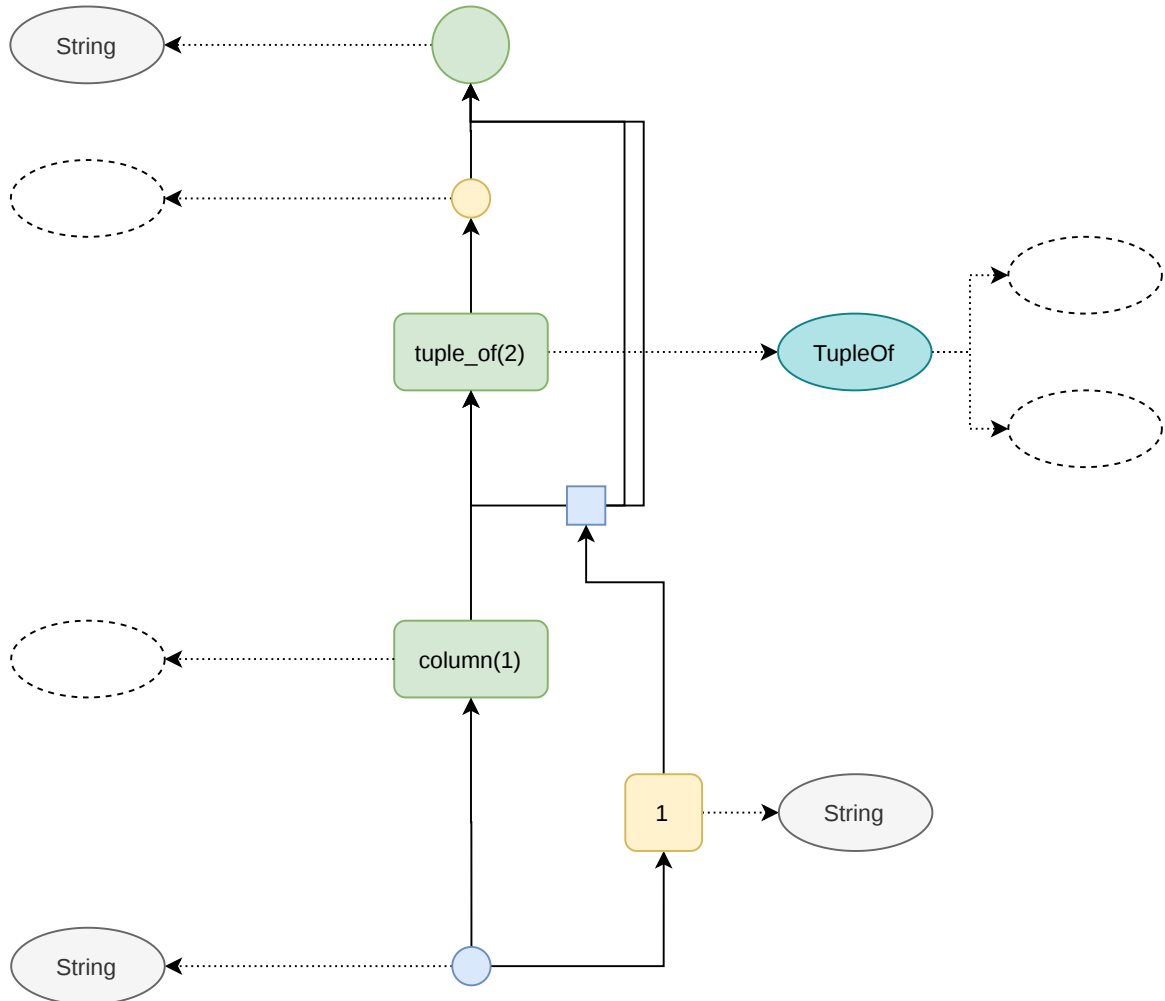
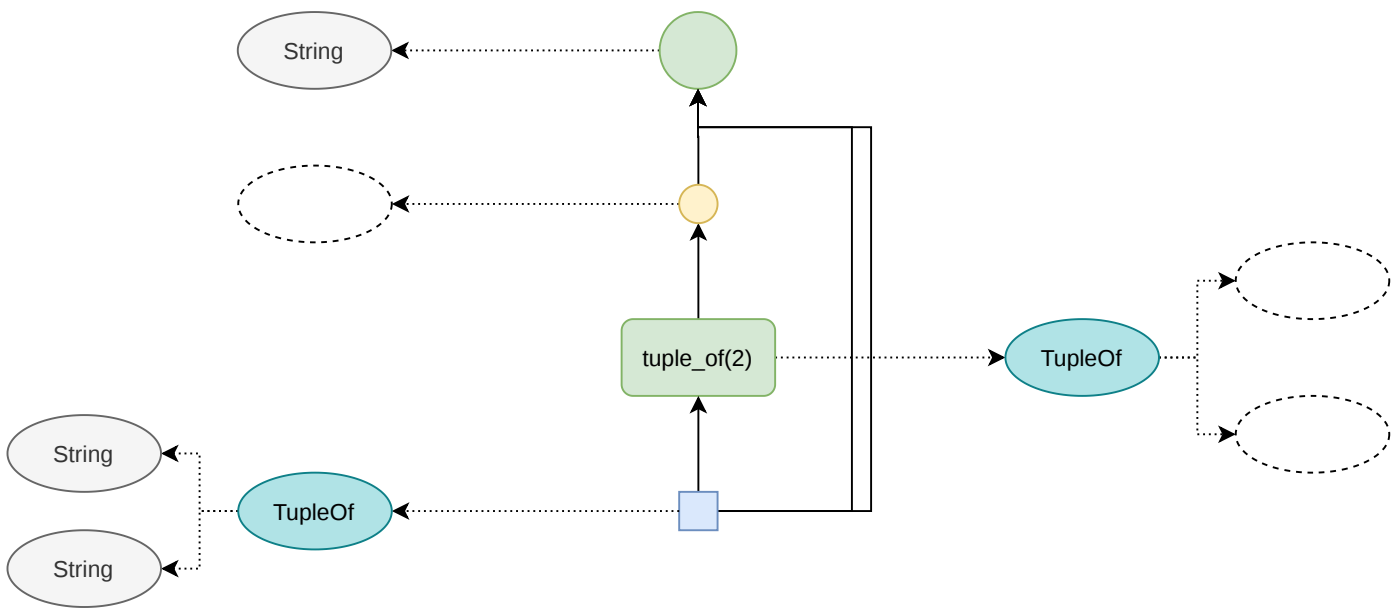
- Stage 1:** A flat table with two columns: an index column (1) and a value column ("Hello World").
- Stage 2:** The table is transformed into a hierarchical structure. The index column is split into two rows (1 and 2), and the value column is split into two columns ("Hello" and "World").
- Stage 3:** The hierarchical structure is further transformed. The index column is split into two rows (1 and 2), and the value column is split into two columns ("HELLO" and "WORLD").
- Stage 4:** The hierarchical structure is further transformed. The index column is split into two rows (1 and 2), and the value column is split into two columns (1 and 1).



wrap()



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



sieve\_by()



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

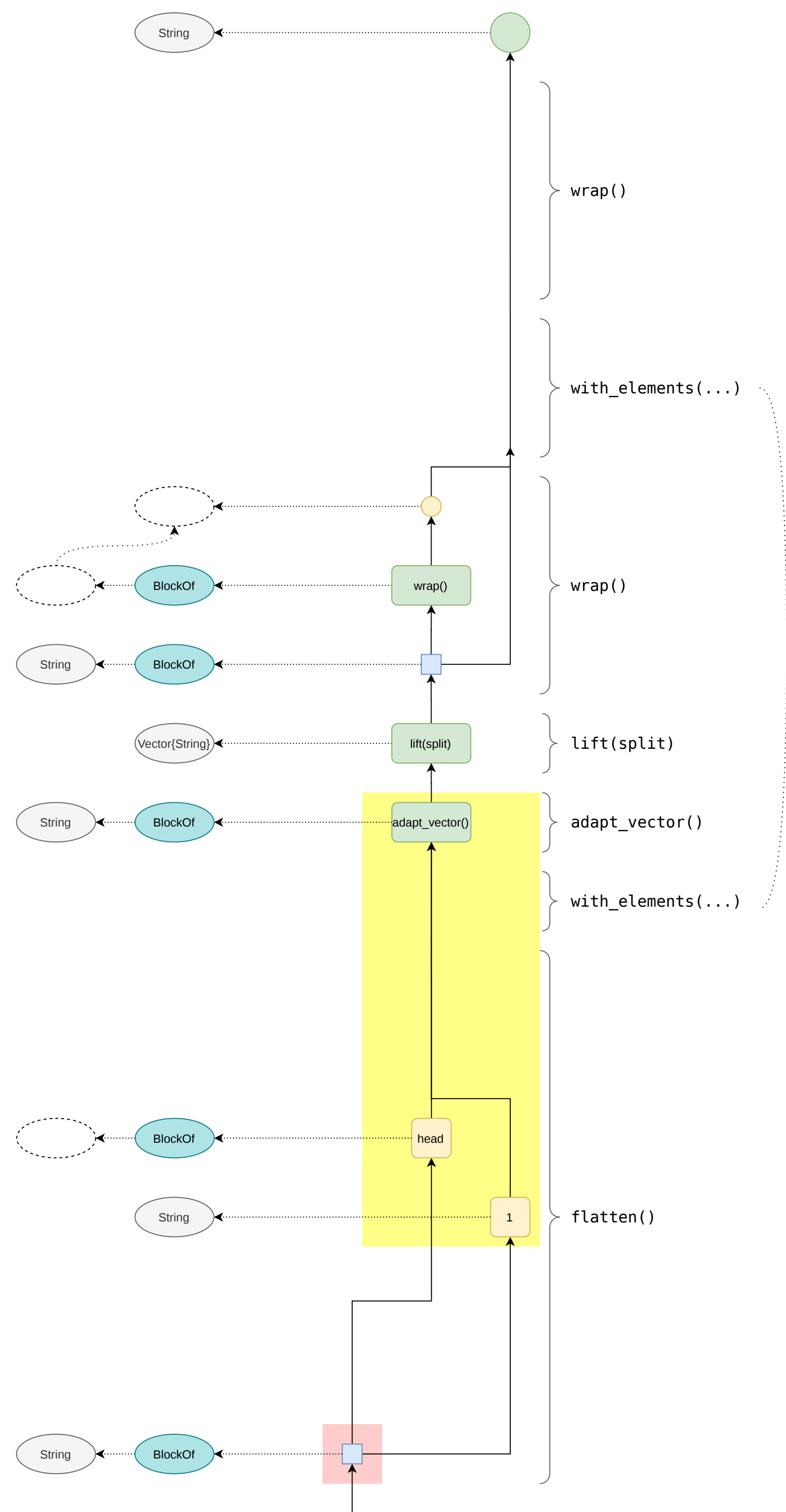
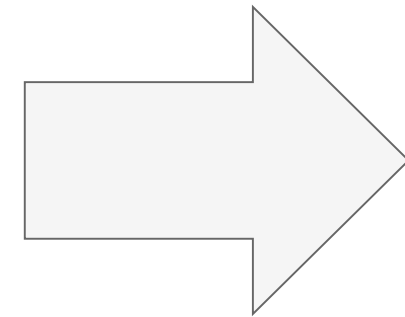


@query "Hello World" split(it)

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



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

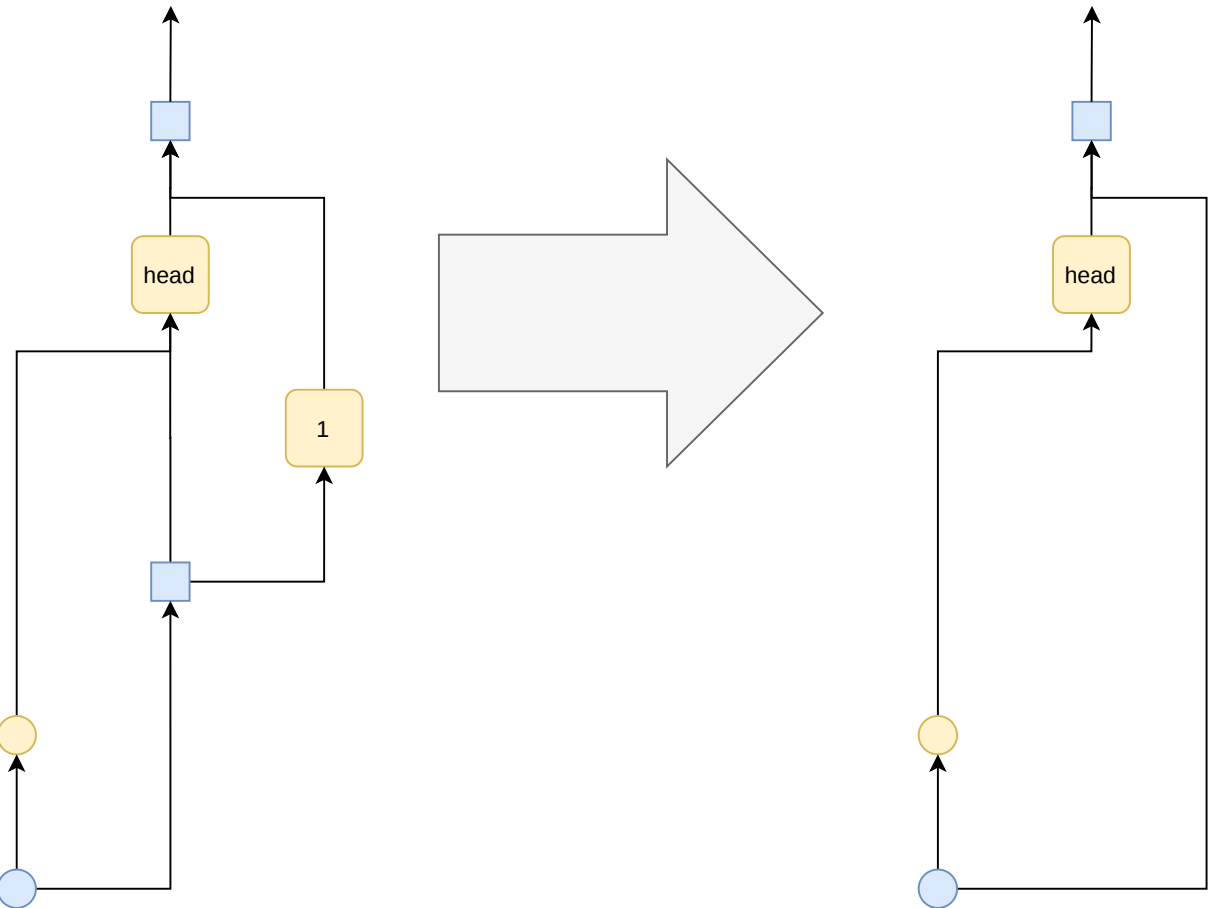






@query "Hello World" split(it)

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



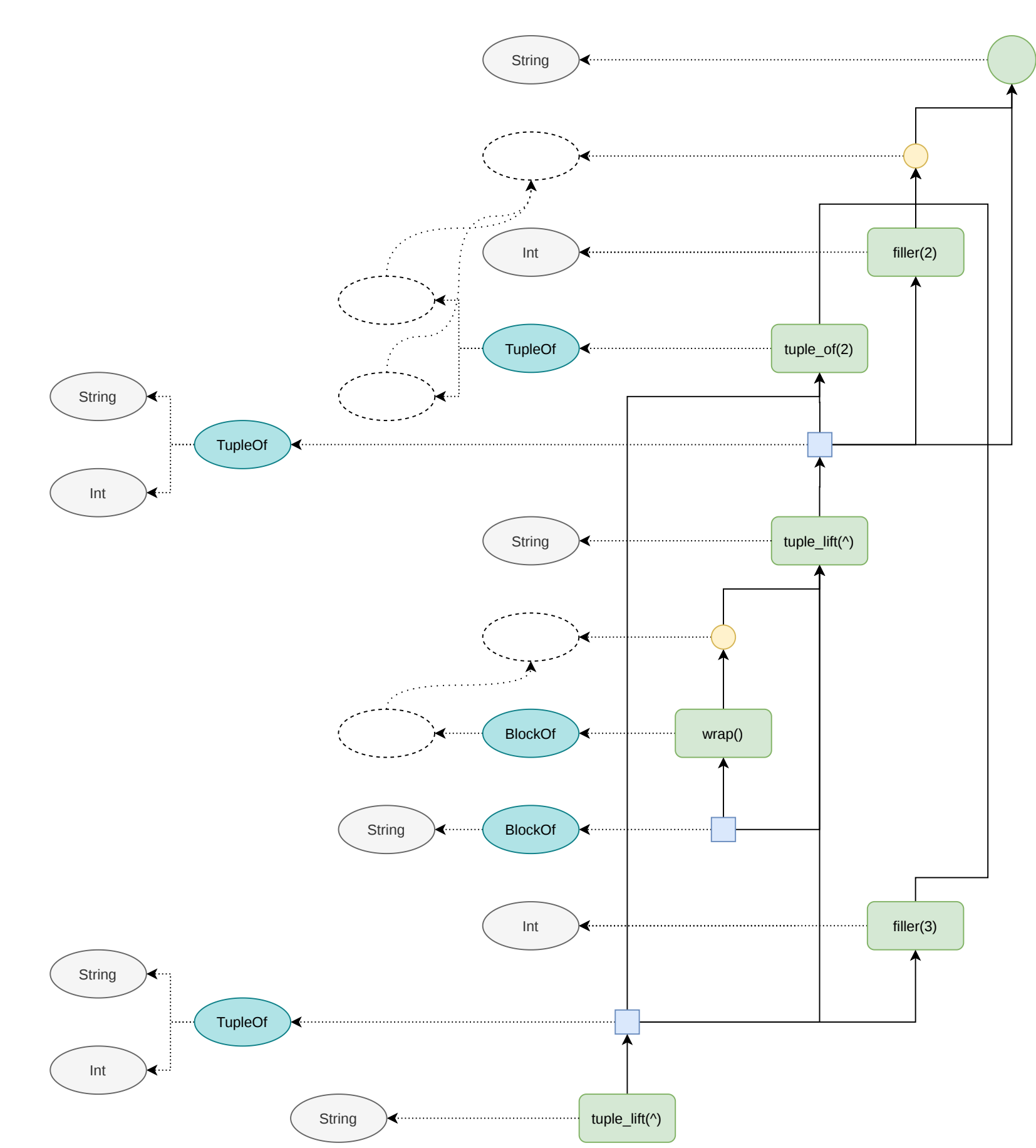




`untrace(n::NodeRef, guard::NodeRef)::Tuple{Pipeline,Vector{NodeRef}}`

`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())
```

