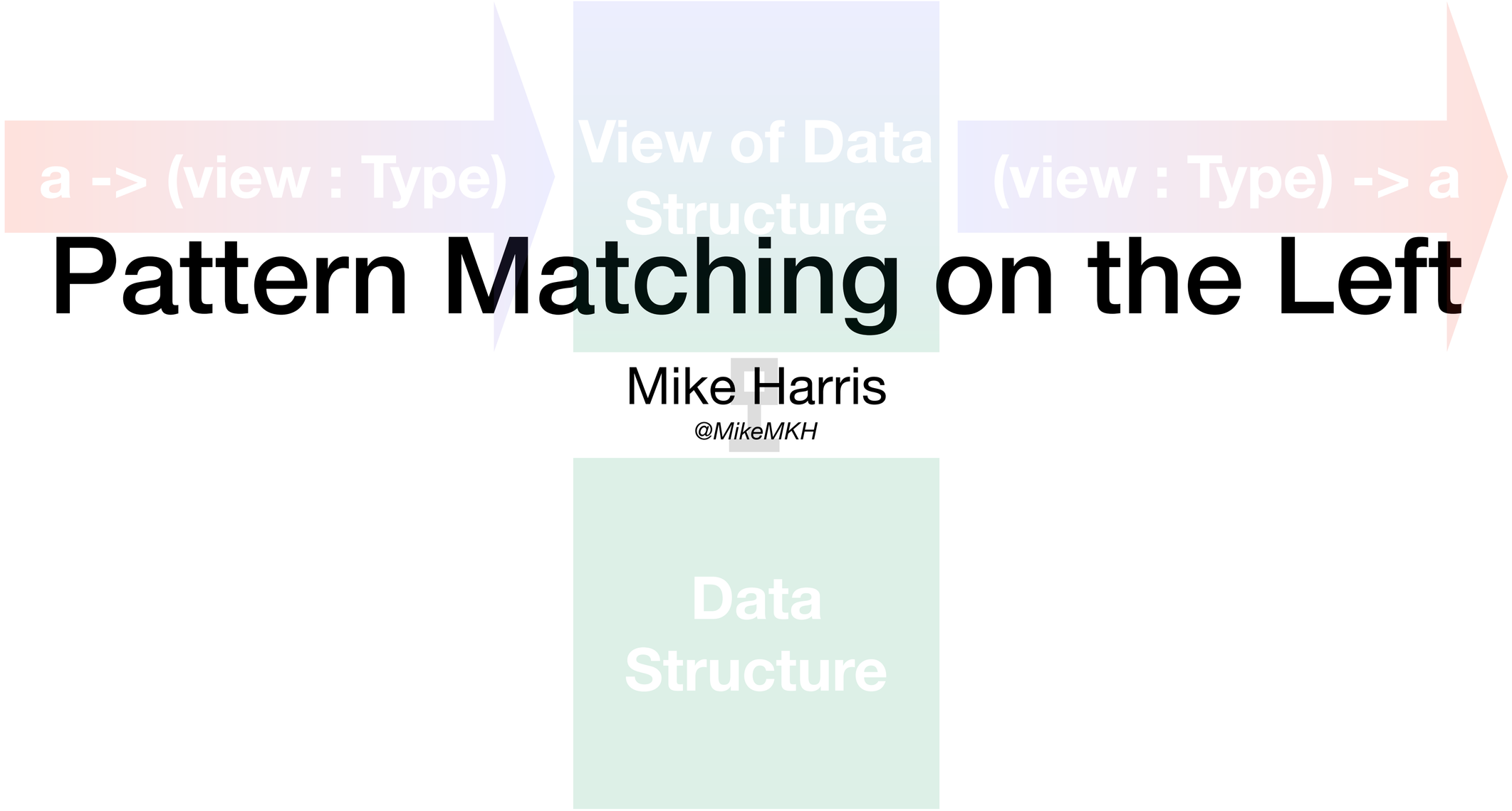
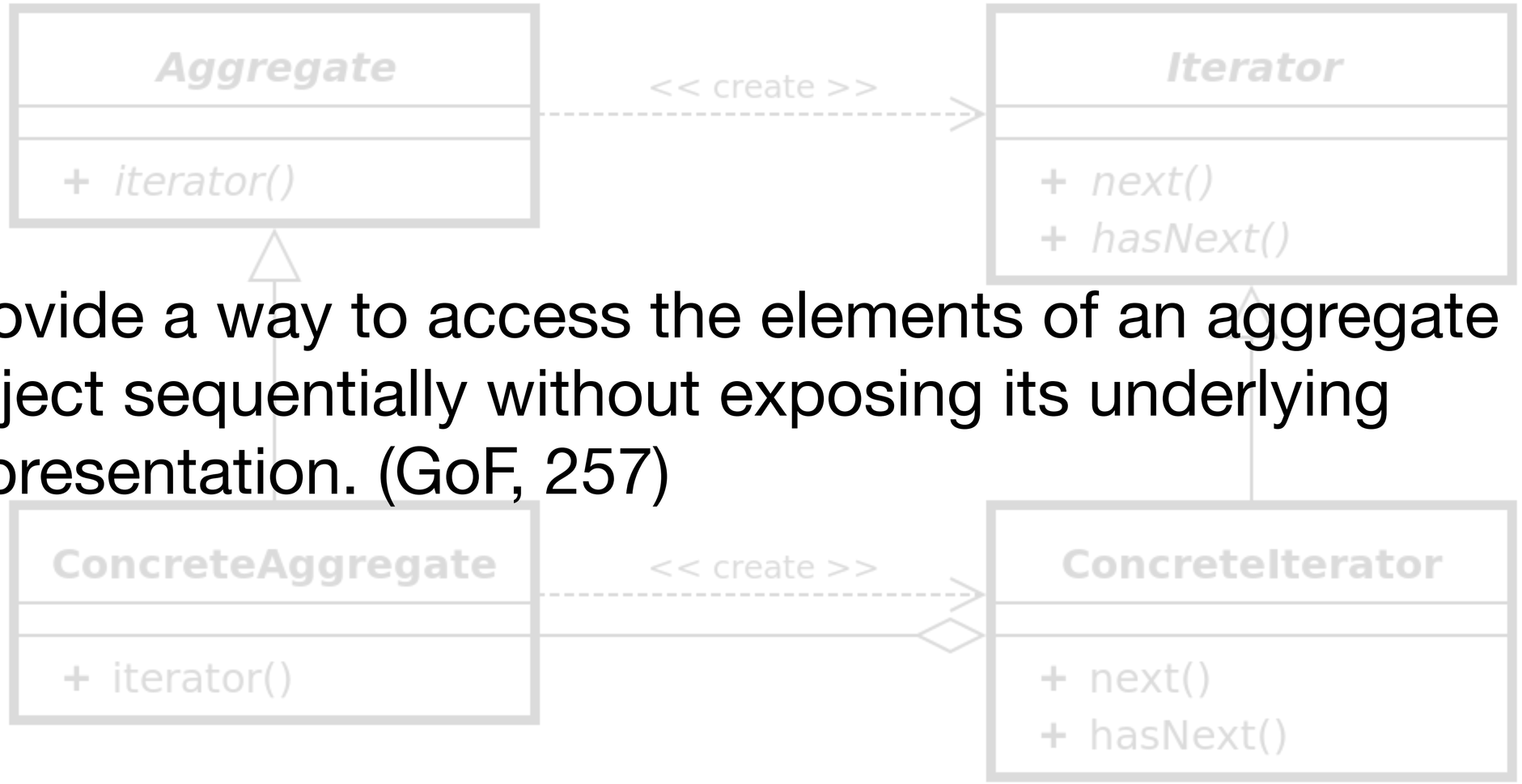


Pattern Matching on the Left

Mike Harris

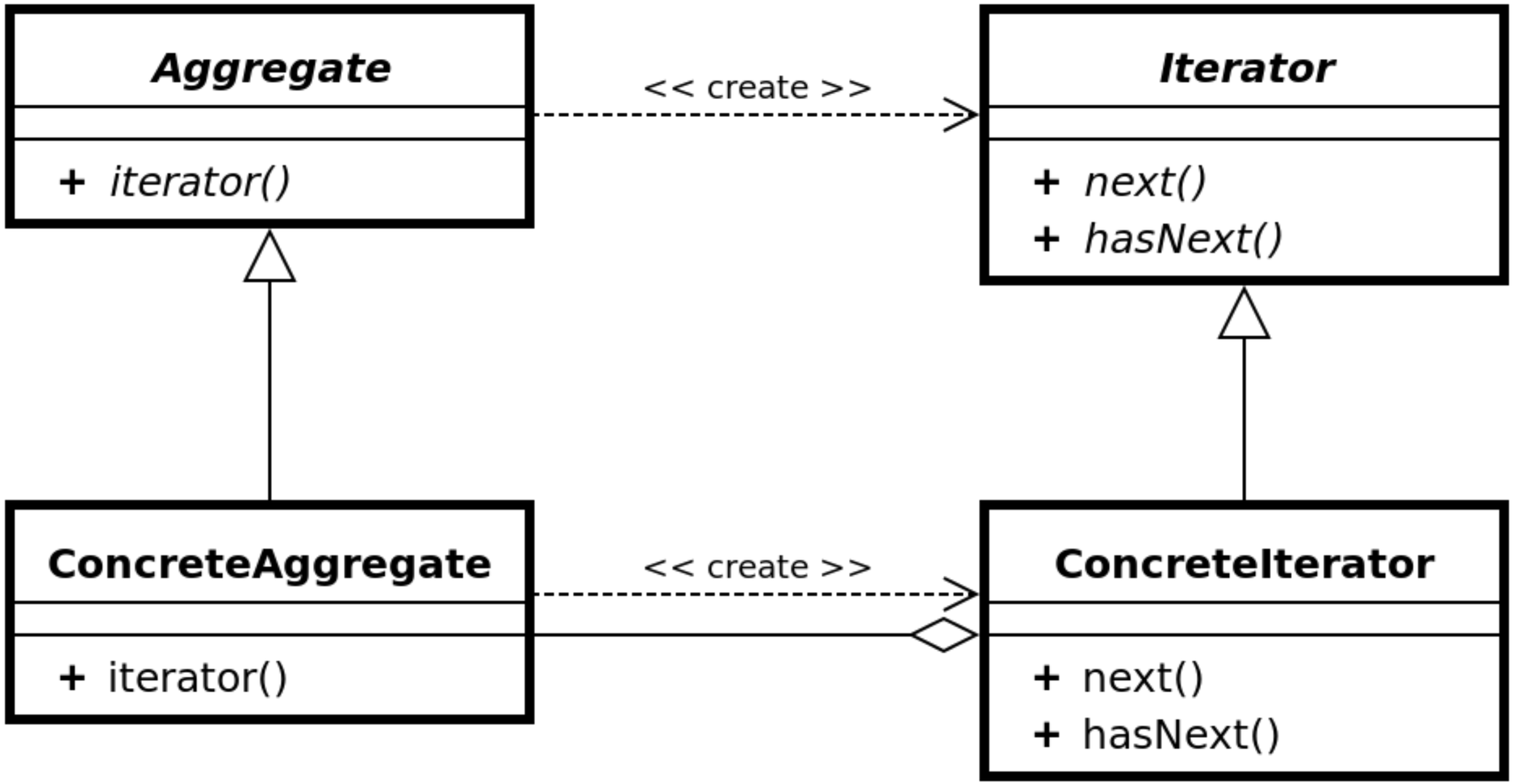
@MikeMKH

Iterator



- Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation. (GoF, 257)

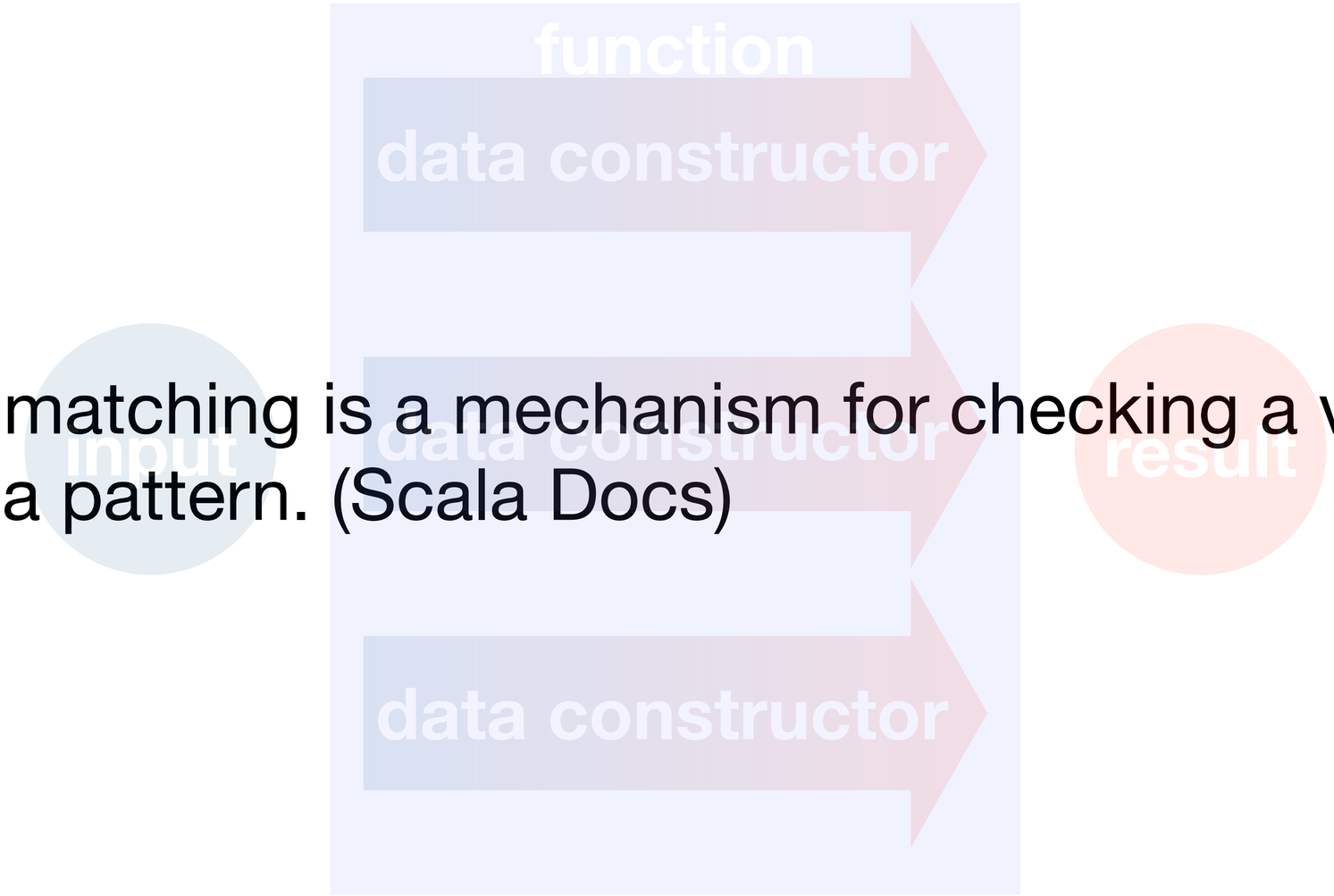
Iterator


Aggregate
+ iterator()
<< create >>
Iterator
+ next()
+ hasNext()
ConcreteAggregate
+ iterator()
<< create >>
ConcreteIterator
+ next()
+ hasNext()

Iterator Example

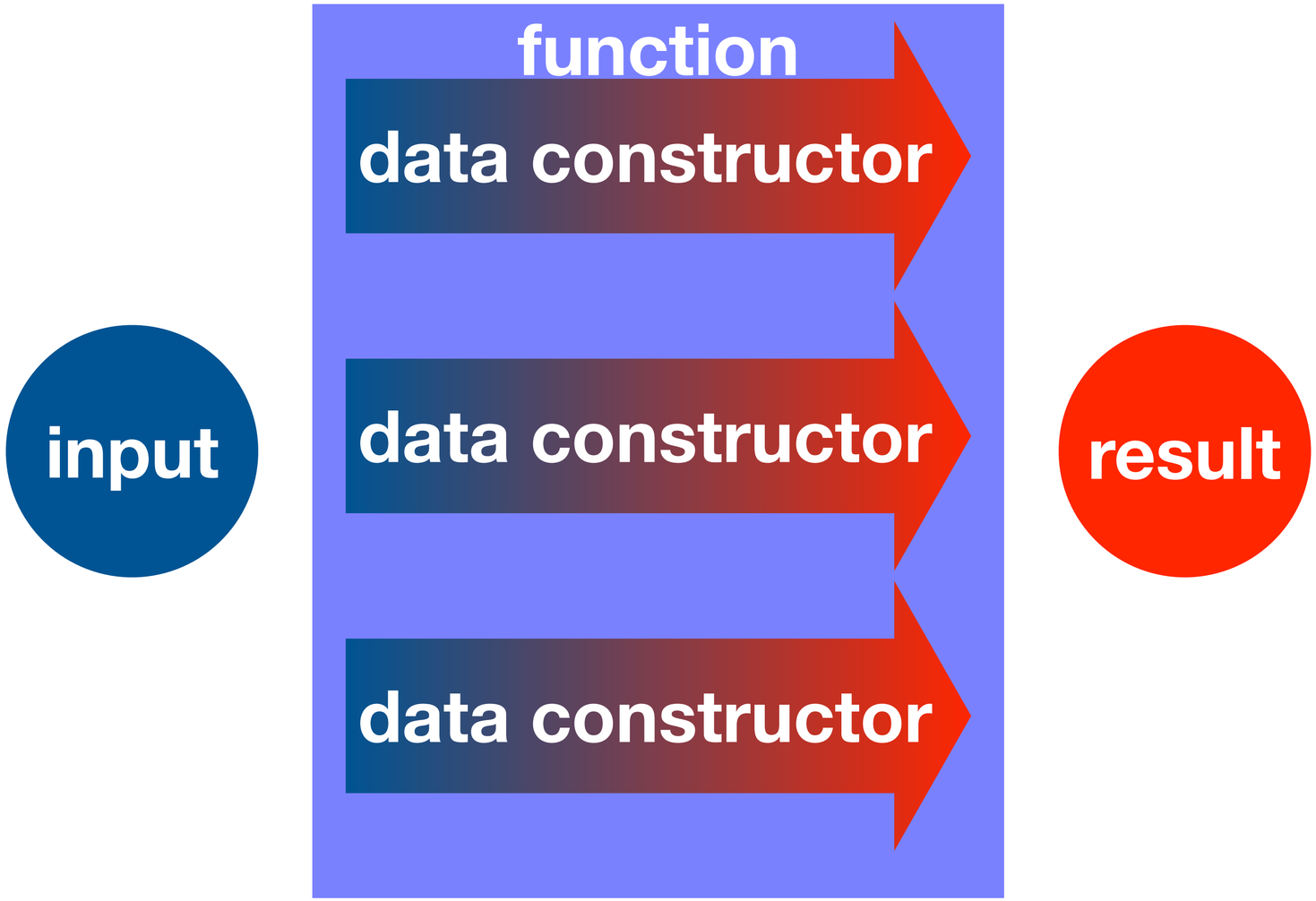
```
[Fact]
public void GivenCollectionWeCanIteratorThroughIt()
{
  IEnumerable<int> col = new HashSet<int>() { 1, 2, 3 };
  IEnumerator<int> iterator = col.GetEnumerator();

  var results = new LinkedList<int>();
  while (iterator.MoveNext())
  {
    results.AddLast(iterator.Current);
  }

  Assert.Equal(results, col);
}
```

Pattern Matching



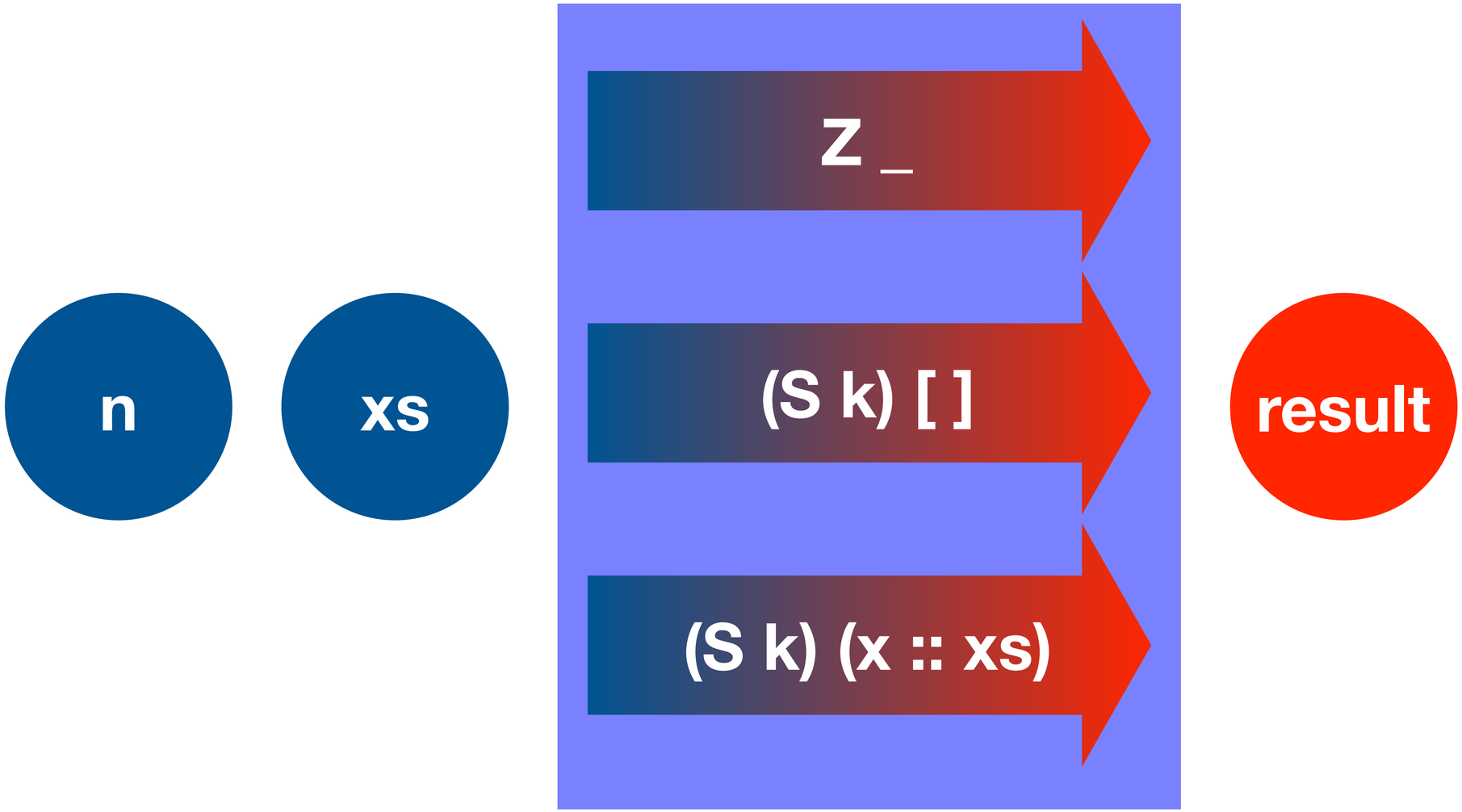
- Pattern matching is a mechanism for checking a value against a pattern. (Scala Docs)

Pattern Matching

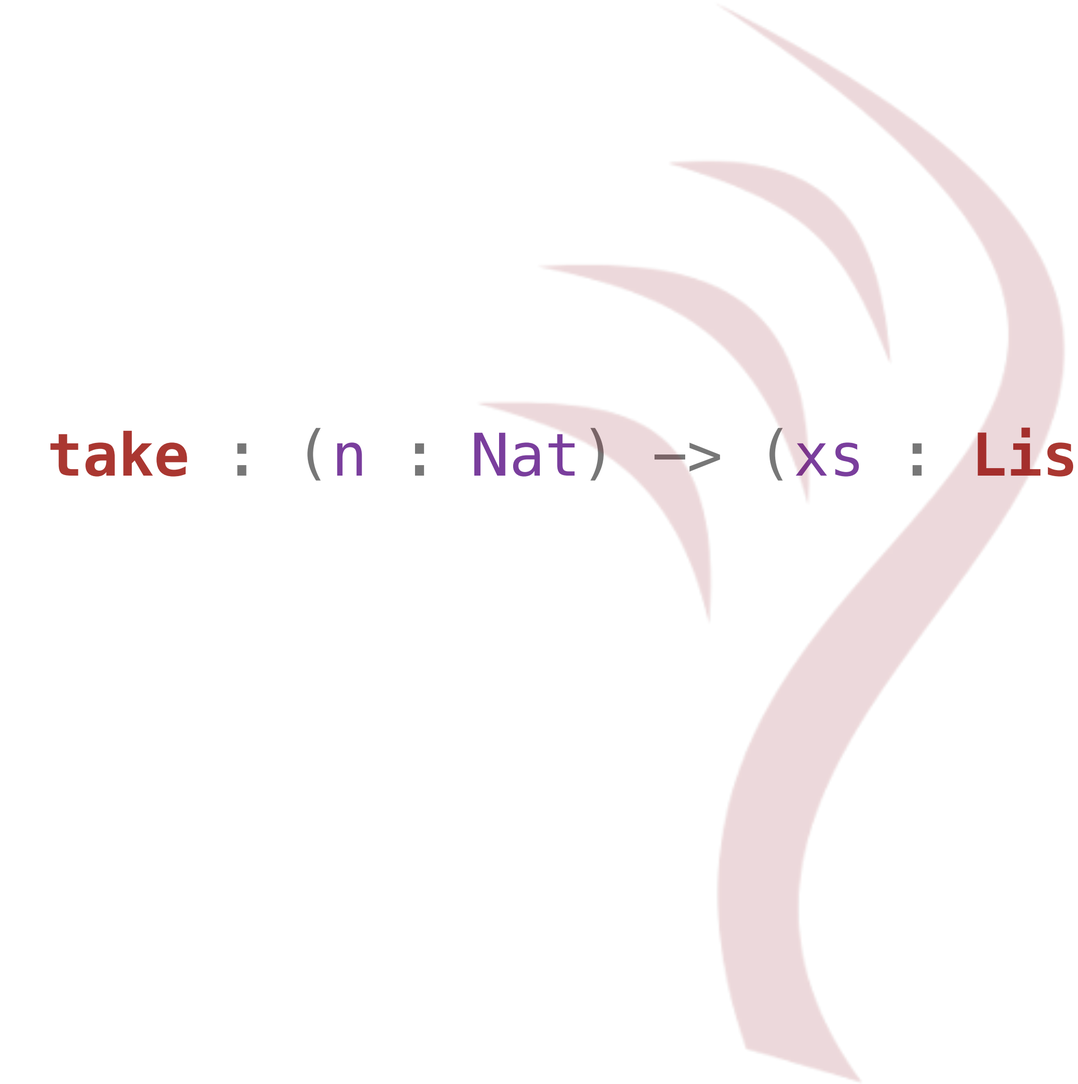


take

take



Example take



```
take : (n : Nat) -> (xs : List a) -> List a
```

Example take

```
take : (n : Nat) -> (xs : List a) -> List a
take n xs = ?take_rhs
```

Example take

```
take : (n : Nat) -> (xs : List a) -> List a
take Z xs = ?take_rhs_1
take (S k) xs = ?take_rhs_2
```

Example take

```
take : (n : Nat) -> (xs : List a) -> List a
take Z xs = []
take (S k) xs = ?take_rhs_2
```

Example take

```
take : (n : Nat) -> (xs : List a) -> List a
take Z _ = []
take (S k) xs = ?take_rhs_2
```

Example take

```
take : (n : Nat) -> (xs : List a) -> List a
take Z _ = []
take (S k) [] = ?take_rhs_1
take (S k) (x :: xs) = ?take_rhs_3
```

Example take

```
take : (n : Nat) -> (xs : List a) -> List a
take Z _ = []
take (S k) [] = []
take (S k) (x :: xs) = ?take_rhs_3
```

Example take

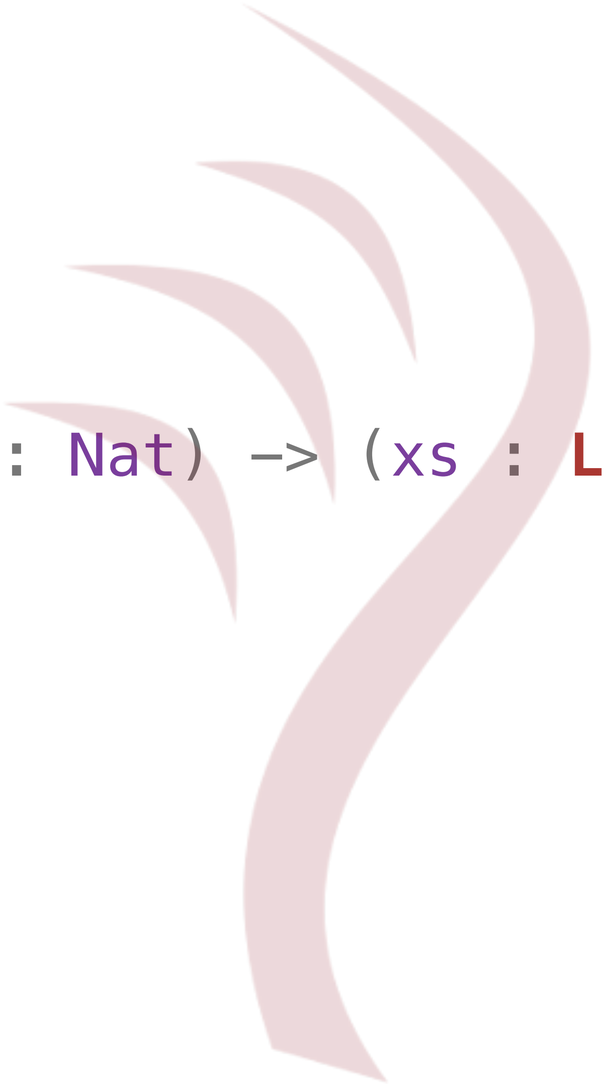
```
take : (n : Nat) -> (xs : List a) -> List a
take Z _ = []
take (S k) [] = []
take (S k) (x :: xs) = x :: take k xs
```

Example take

```
take : (n : Nat) -> (xs : List a) -> List a
take Z _ = []
take (S k) [] = []
take (S k) (x :: xs) = x :: take k xs
```

takeLast

Example takeLast



```
takeLast : (n : Nat) -> (xs : List a) -> List a
```

Example takeLast

```
takeLast : (n : Nat) -> (xs : List a) -> List a
takeLast Z xs = []
takeLast (S k) [] = []
takeLast (S k) (x :: []) = ?takeLast_rhs_1
takeLast (S k) (x :: (y :: xs)) = ?takeLast_rhs_2
```

Example takeLast

```
takeLast : (n : Nat) -> (xs : List a) -> List a
takeLast n xs = reverse $ take n $ reverse xs
```

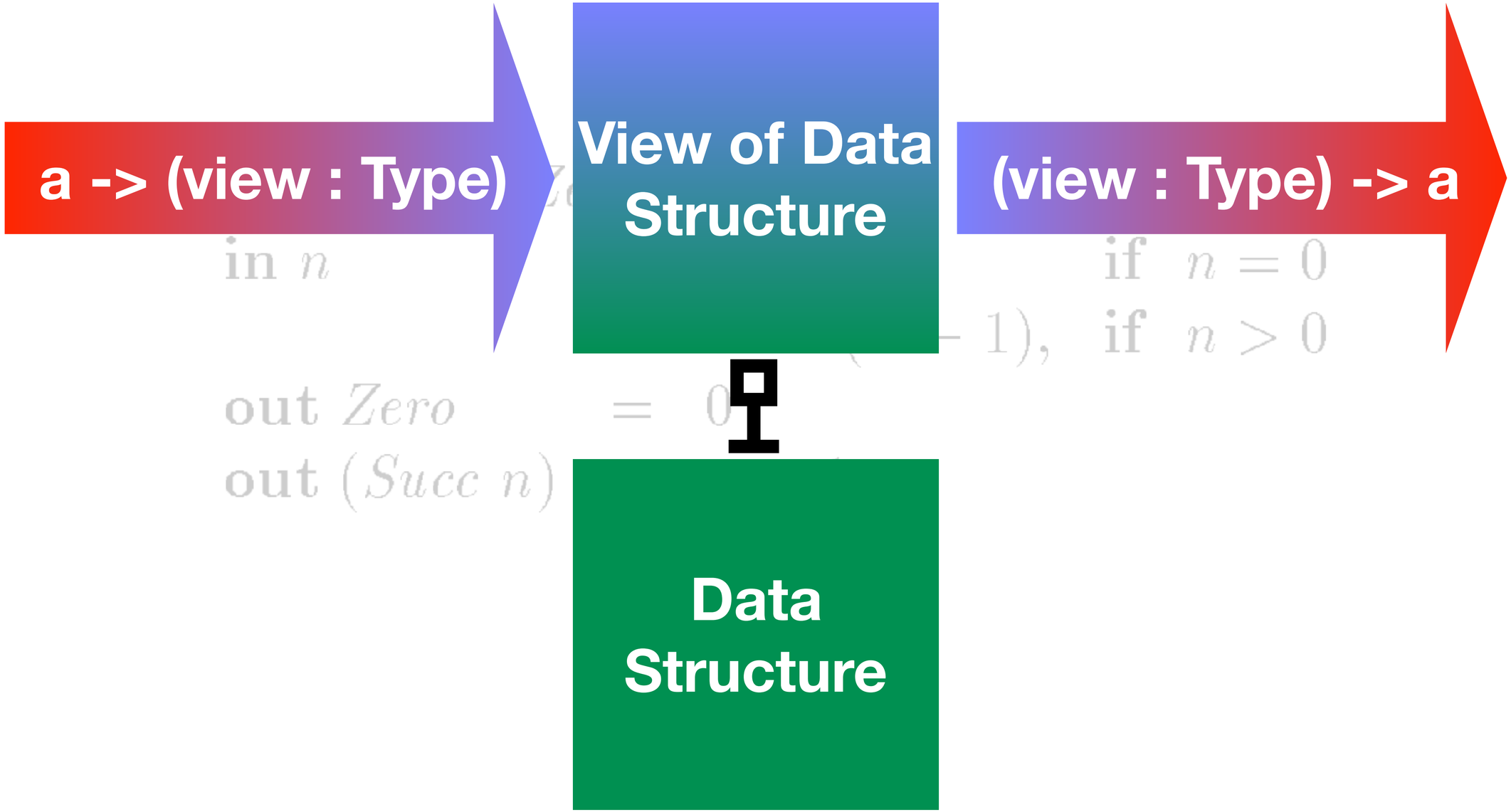
Views

Views

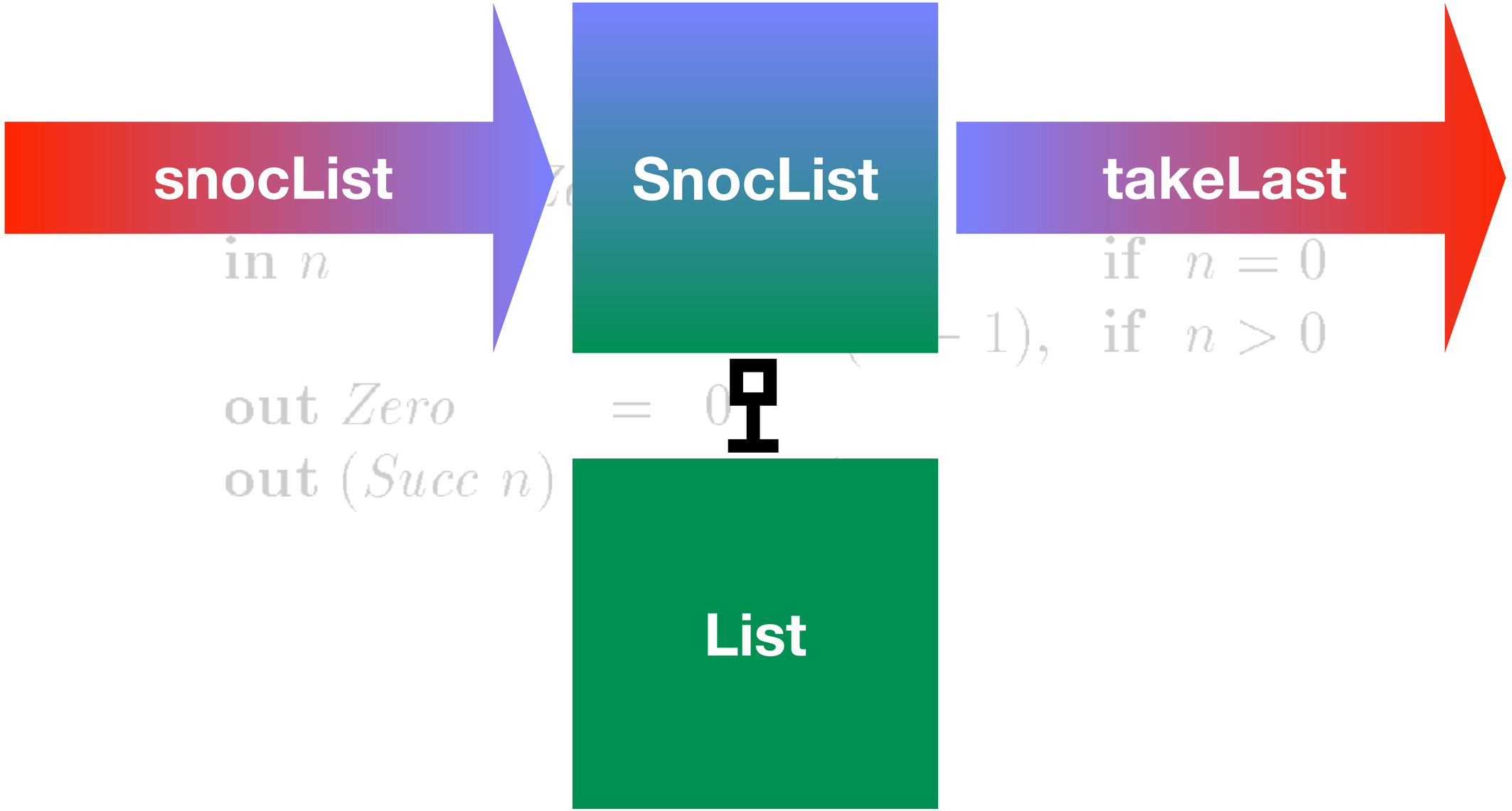
$$
\begin{array}{lcll}
\textbf{view}\ \ int ::= Zero \mid Succ\ int & & & \\
\quad \textbf{in}\ n & = & Zero, & \textbf{if}\ \ n = 0 \\
 & = & Succ\ (n-1), & \textbf{if}\ \ n > 0 \\
\quad \textbf{out}\ Zero & = & 0 & \\
\quad \textbf{out}\ (Succ\ n) & = & n+1 &
\end{array}
$$

(Wadler, 3)

Views


a -> (view : Type)
View of Data Structure
(view : Type) -> a
Data Structure

Views



SnocList view in

```
snocListHelp : SnocList xs -> (ys : List a) -> SnocList (xs ++ ys)
snocListHelp {xs} x [] = rewrite appendNilRightNeutral xs in x
snocListHelp {xs} x (y :: ys)
    = rewrite appendAssociative xs [y] ys in snocListHelp (Snoc x) ys

snocList : (xs : List a) -> SnocList xs
snocList xs = snocListHelp Empty xs
```

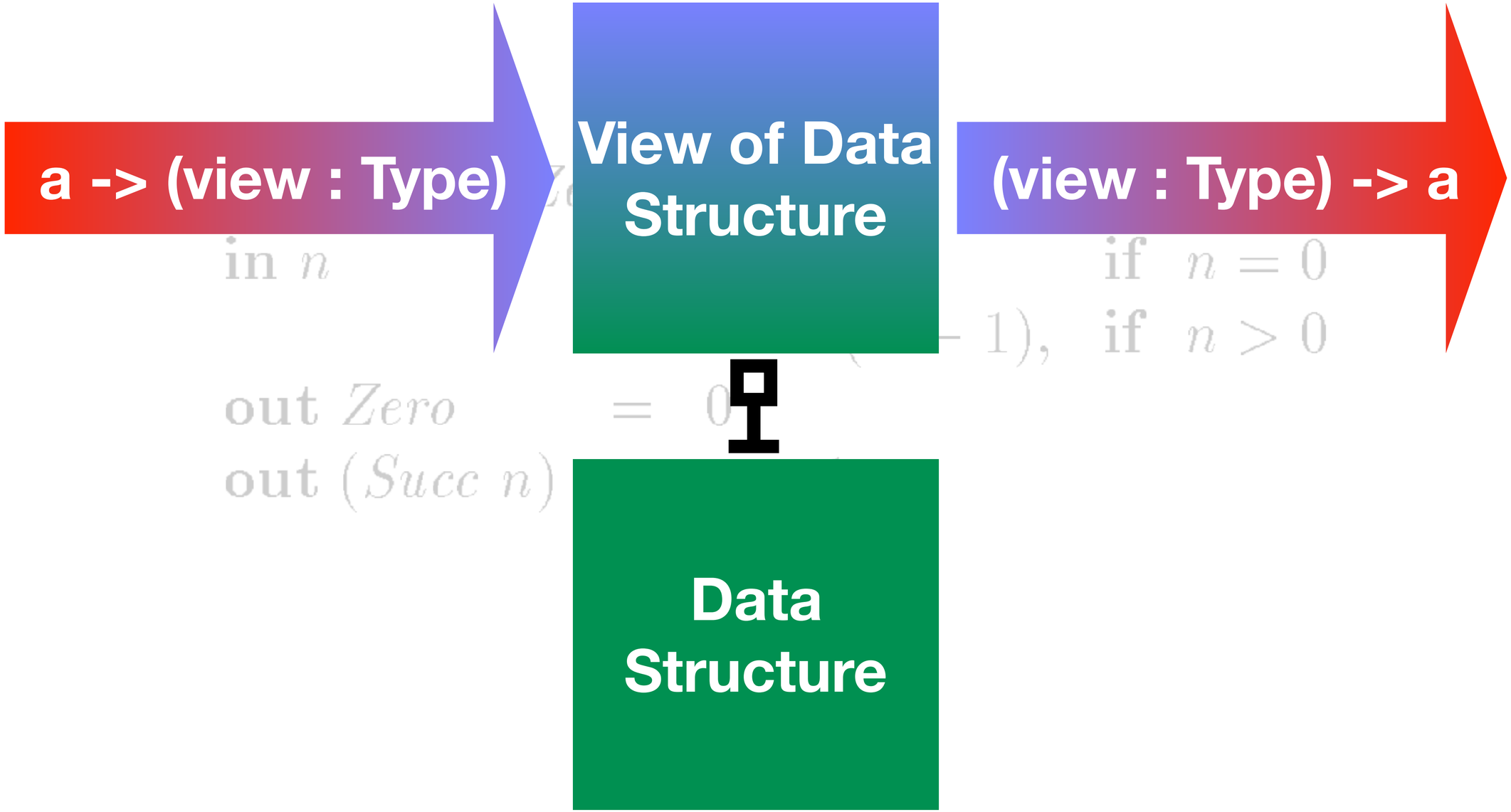
SnocList view data structure

```
data SnocList : List a -> Type where
  Empty : SnocList []
  Snoc : {x : a} -> {xs : List a} ->
         (rec : SnocList xs) -> SnocList (xs ++ [x])
```

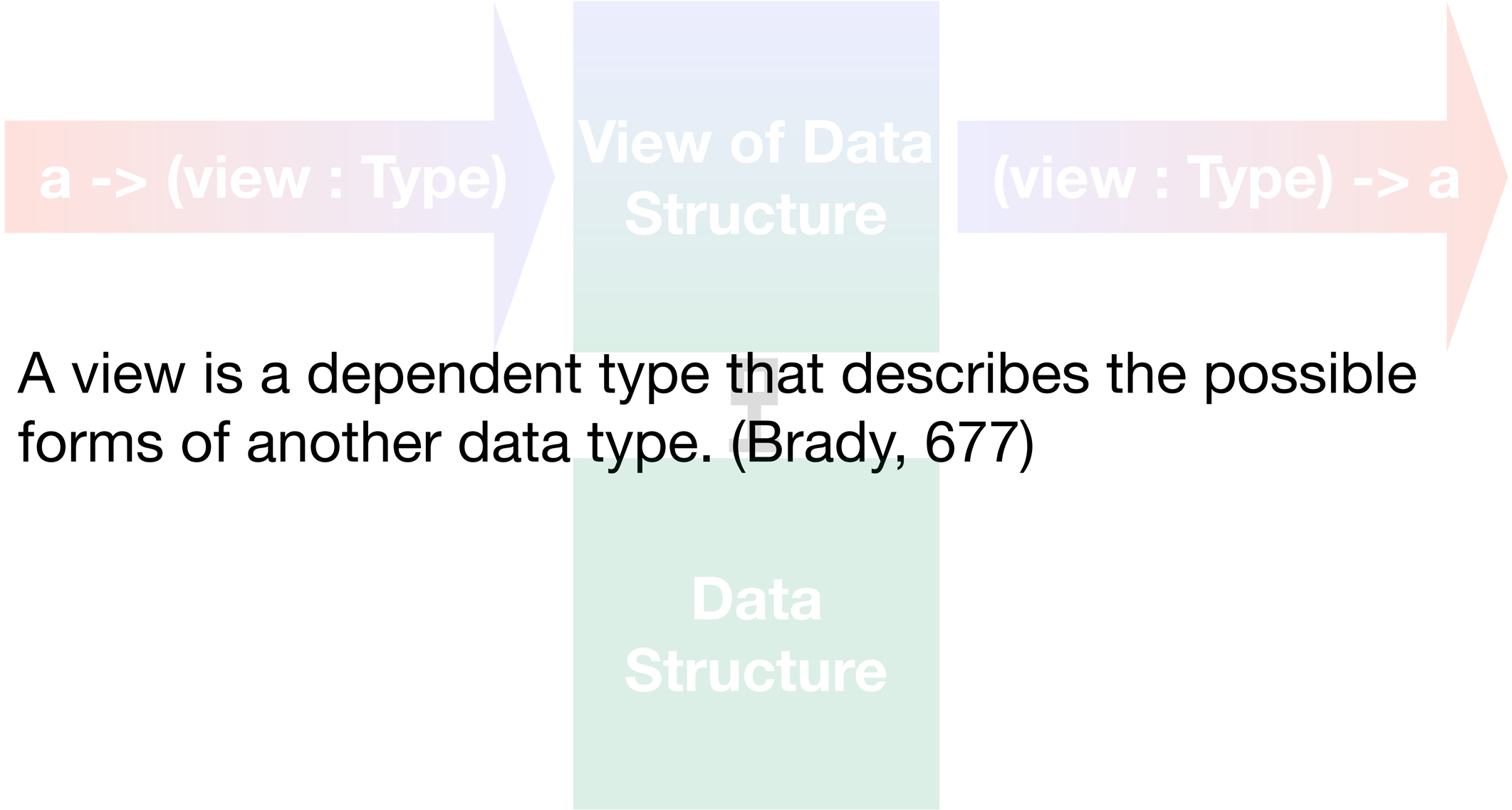
SnocList view out

```
takeLast : (n : Nat) -> List a -> List a
takeLast n xs with (snocList xs)
  takeLast Z [] | Empty = []
  takeLast Z (ys ++ [x]) | (Snoc rec) = []
  takeLast (S k) [] | Empty = []
  takeLast (S k) (ys ++ [x]) | (Snoc rec) = takeLast k ys ++ [x] | rec
```

Views


a -> (view : Type)
View of Data Structure
(view : Type) -> a
Data Structure

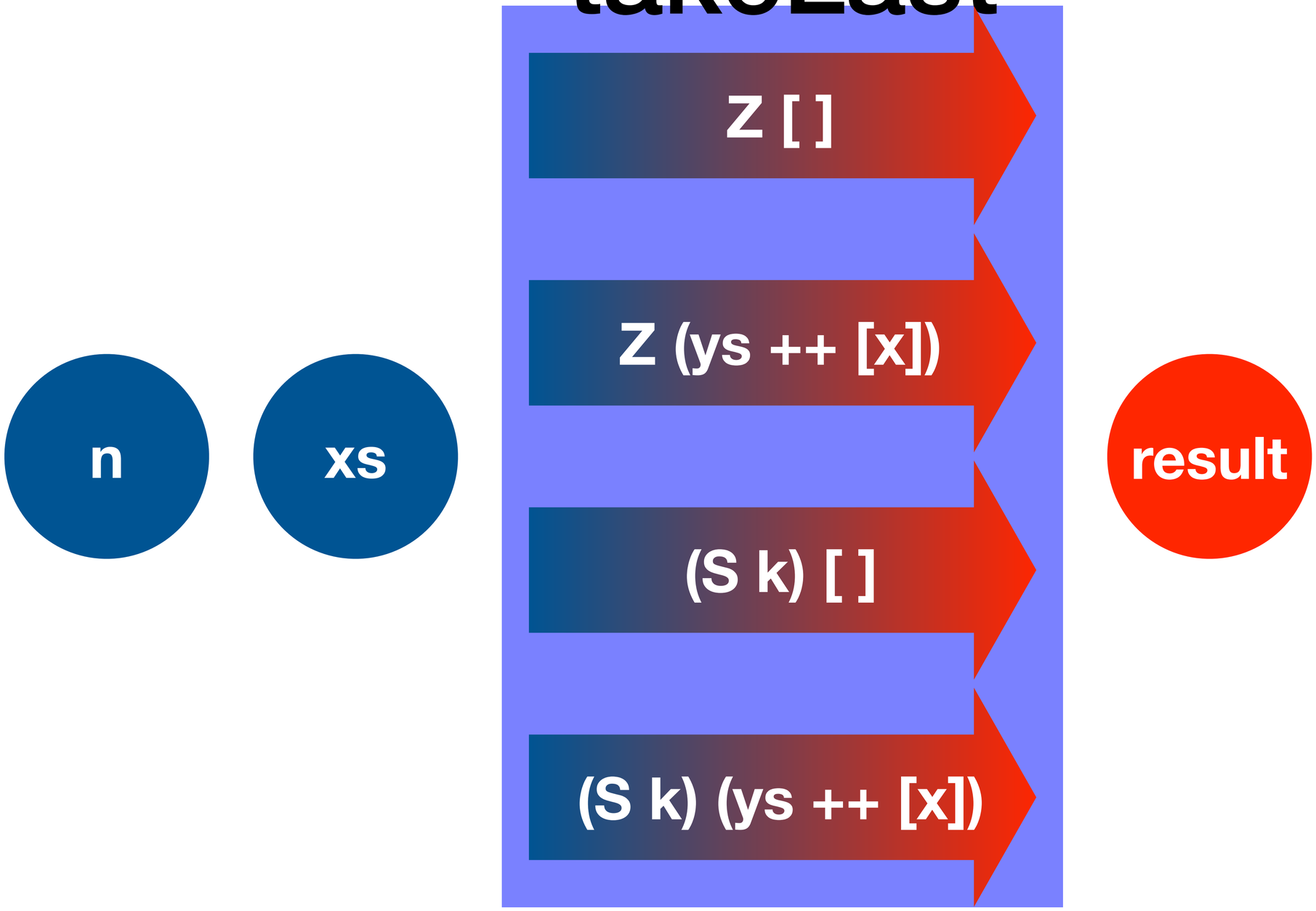
Views



- A view is a dependent type that describes the possible forms of another data type. (Brady, 677)

takeLast

takeLast



Example takeLast

```
takeLast : (n : Nat) -> List a -> List a
```

Example takeLast

```
takeLast : (n : Nat) -> List a -> List a
takeLast n xs = ?takeLast_rhs
```

Example takeLast

```
takeLast : (n : Nat) -> List a -> List a
takeLast n xs with (_)
  takeLast n xs | with_pat = ?takeLast_rhs_rhs
```

Example takeLast

```
takeLast : (n : Nat) -> List a -> List a
takeLast n xs with (snocList xs)
  takeLast n xs | with_pat = ?takeLast_rhs_rhs
```

Example takeLast

```
takeLast : (n : Nat) -> List a -> List a
takeLast n xs with (snocList xs)
  takeLast Z xs | with_pat = ?takeLast_rhs_rhs_1
  takeLast (S k) xs | with_pat = ?takeLast_rhs_rhs_2
```

Example takeLast

```
takeLast : (n : Nat) -> List a -> List a
takeLast n xs with (snocList xs)
  takeLast Z [] | Empty = ?takeLast_rhs_rhs_3
  takeLast Z (ys ++ [x]) | (Snoc rec) = ?takeLast_rhs_rhs_4
  takeLast (S k) xs | with_pat = ?takeLast_rhs_rhs_2
```

Example takeLast

```
takeLast : (n : Nat) -> List a -> List a
takeLast n xs with (snocList xs)
  takeLast Z [] | Empty = []
  takeLast Z (ys ++ [x]) | (Snoc rec) = ?takeLast_rhs_rhs_4
  takeLast (S k) xs | with_pat = ?takeLast_rhs_rhs_2
```

Example takeLast

```
takeLast : (n : Nat) -> List a -> List a
takeLast n xs with (snocList xs)
  takeLast Z [] | Empty = []
  takeLast Z (ys ++ [x]) | (Snoc rec) = []
  takeLast (S k) xs | with_pat = ?takeLast_rhs_rhs_2
```

Example takeLast

```
takeLast : (n : Nat) -> List a -> List a
takeLast n xs with (snocList xs)
  takeLast Z [] | Empty = []
  takeLast Z (ys ++ [x]) | (Snoc rec) = []
  takeLast (S k) [] | Empty = ?takeLast_rhs_rhs_1
  takeLast (S k) (ys ++ [x]) | (Snoc rec) = ?takeLast_rhs_rhs_3
```

Example takeLast

```
takeLast : (n : Nat) -> List a -> List a
takeLast n xs with (snocList xs)
  takeLast Z [] | Empty = []
  takeLast Z (ys ++ [x]) | (Snoc rec) = []
  takeLast (S k) [] | Empty = []
  takeLast (S k) (ys ++ [x]) | (Snoc rec) = ?takeLast_rhs_rhs_3
```

Example takeLast

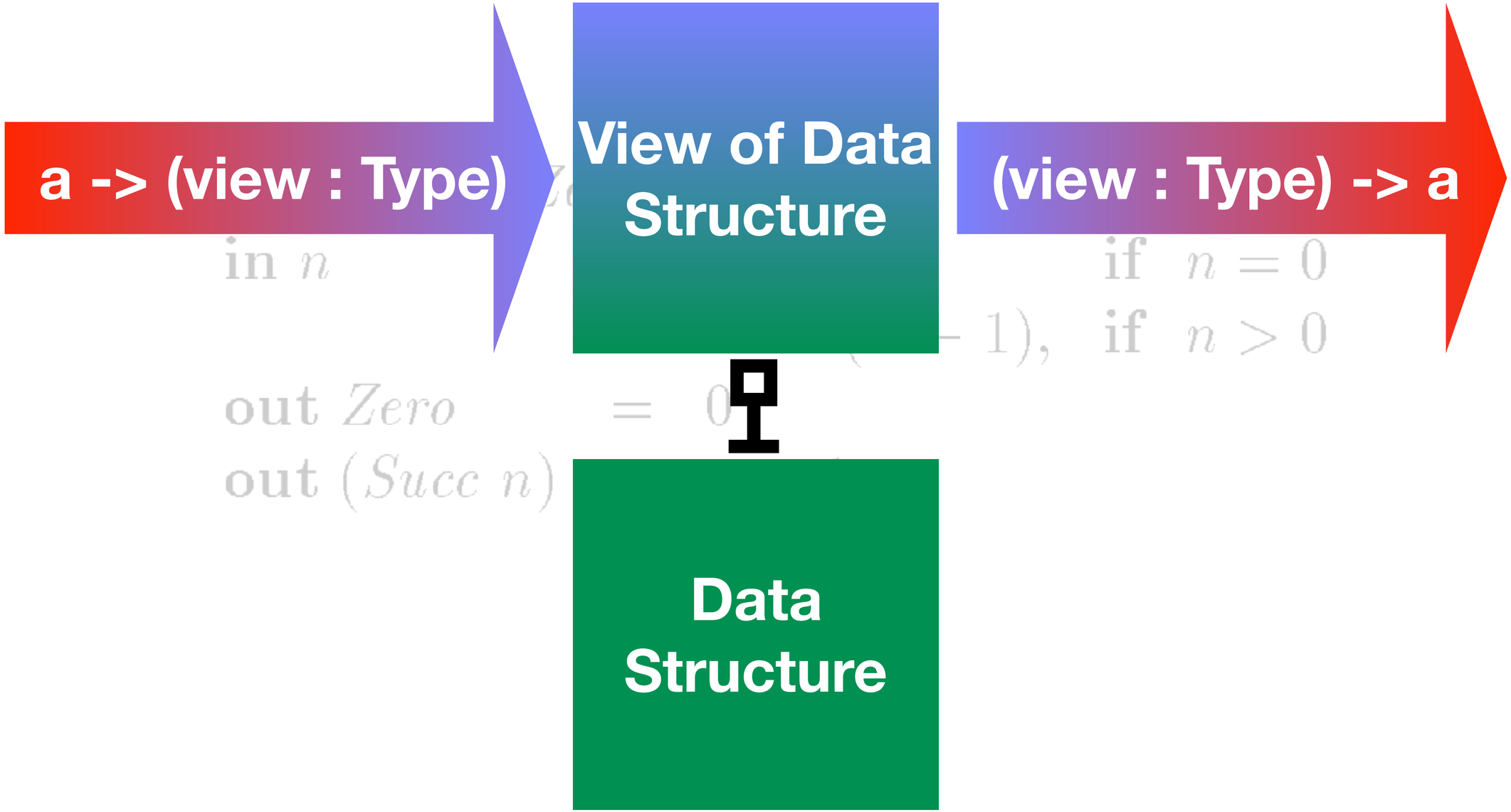
```
takeLast : (n : Nat) -> List a -> List a
takeLast n xs with (snocList xs)
  takeLast Z [] | Empty = []
  takeLast Z (ys ++ [x]) | (Snoc rec) = []
  takeLast (S k) [] | Empty = []
  takeLast (S k) (ys ++ [x]) | (Snoc rec) = takeLast k ys ++ [x] | rec
```

Example takeLast

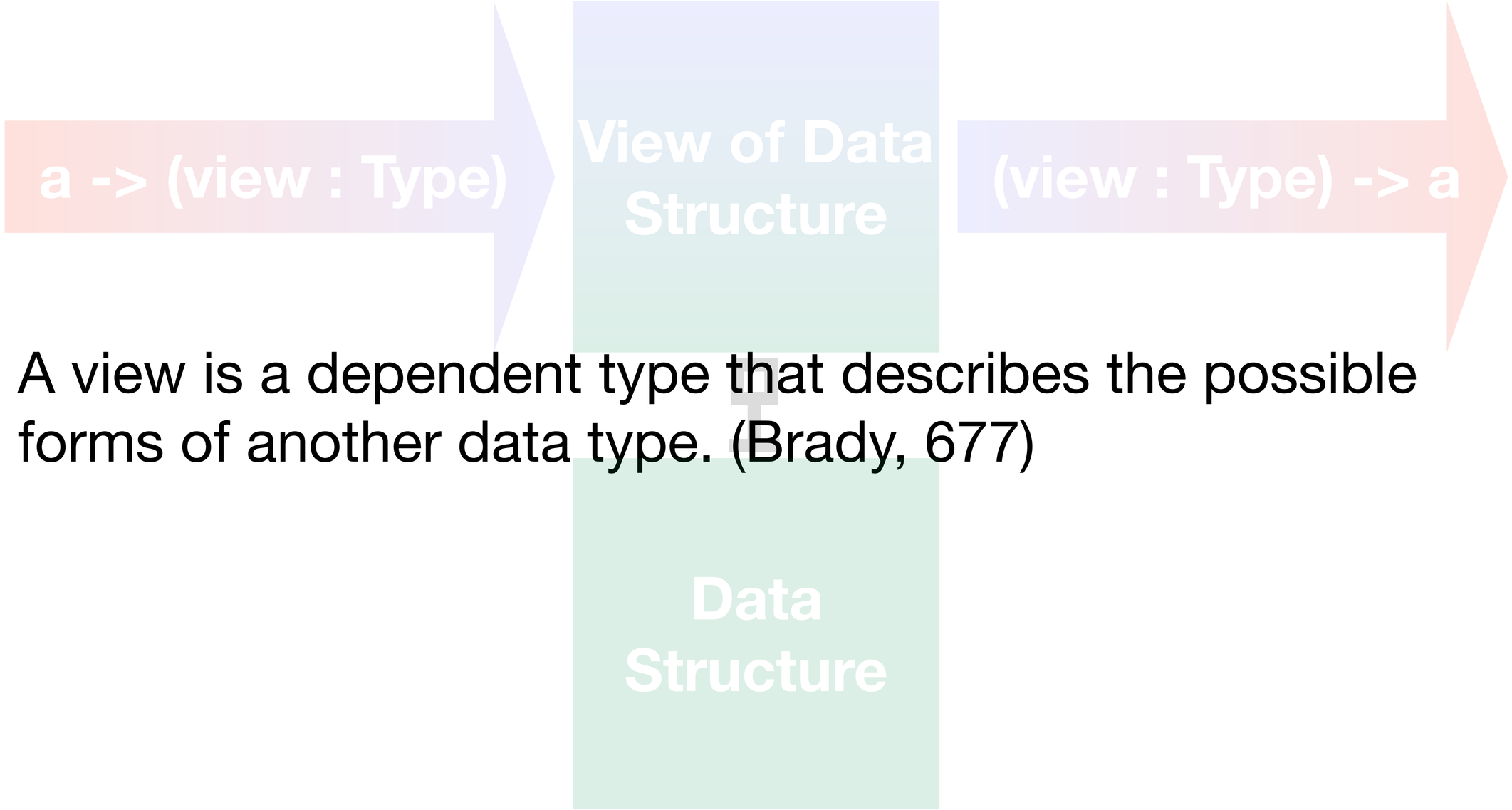
```
takeLast : (n : Nat) -> List a -> List a
takeLast n xs with (snocList xs)
  takeLast Z [] | Empty = []
  takeLast Z (ys ++ [x]) | (Snoc rec) = []
  takeLast (S k) [] | Empty = []
  takeLast (S k) (ys ++ [x]) | (Snoc rec) = takeLast k ys ++ [x] | rec
```

Views

Views


a -> (view : Type)
View of Data Structure
(view : Type) -> a
Data Structure

Views



- A view is a dependent type that describes the possible forms of another data type. (Brady, 677)

Thank you!

Mike Harris
@MikeMKH

https://github.com/MikeMKH/talks/tree/master/pattern-matching-on-the-left

Next Steps

- Philip Wadler, Views: A way for pattern matching to cohabit with data abstraction. http://cs.ru.nl/~freek/courses/tt-2010/tvftl/wadler-views.pdf
- Edwin Brady, Type-Driven Development with Idris, chapter 10. https://www.manning.com/books/type-driven-development-with-idris
- Idris docs, Views and the “with” rule. http://docs.idris-lang.org/en/latest/tutorial/views.html

Sources

- Brady, Edwin. Type-driven development with Idris. Shelter Island, NY: Manning Publications Co, 2017. Print.

- Gamma, Erich. Design patterns : elements of reusable object-oriented software. Reading, Mass: Addison-Wesley, 1995. Print.

- Scala Docs: Tour of Scala, Pattern Matching. https://docs.scala-lang.org/tour/pattern-matching.html. Online.

- Wadler, Philip. Views: a way for pattern matching to cohabit with data abstraction. In Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages (POPL '87). ACM, New York, NY, USA, 307-313. DOI=http://dx.doi.org/10.1145/41625.41653. Print.

Code

- Example C# code, https://github.com/MikeMKH/talks/blob/master/pattern-matching-on-the-left/Tests.cs
- Example Idris code, https://github.com/MikeMKH/talks/blob/master/pattern-matching-on-the-left/Example.idr
- SnocList from Idris source, https://github.com/idris-lang/Idris-dev/blob/8ab4dc878a2bac542ee8a817f0054b378d9dad8a/libs/base/Data/List/Views.idr#L77-L93

Images

- UML Iterator by Trashtoy - My own work written with text editor., Public Domain, https://commons.wikimedia.org/w/index.php?curid=1698830

- Idris logo - Created by Heath Johns (https://twitter.com/edwinbrady/status/566261662303653888) and added to GitHub by Jan de Muijnck-Hughes, Public Domain, https://github.com/idris-lang/Idris-dev/blob/master/icons/text-x-idris.svg