Records with Rank Polymorphism

Justin Slepak
jrslepak@ccs.neu.edu

Olin Shivers
shivers@ccs.neu.edu

Panagiotis Manolios
pete@ccs.neu.edu

Northeastern University

Boston, MA, USA

The Remora Project

Remora: Higher-order rank-polymorphic language

The Remora Project

Remora: Higher-order rank-polymorphic language

Functions work on arbitrarily high-dimensional data

The Remora Project

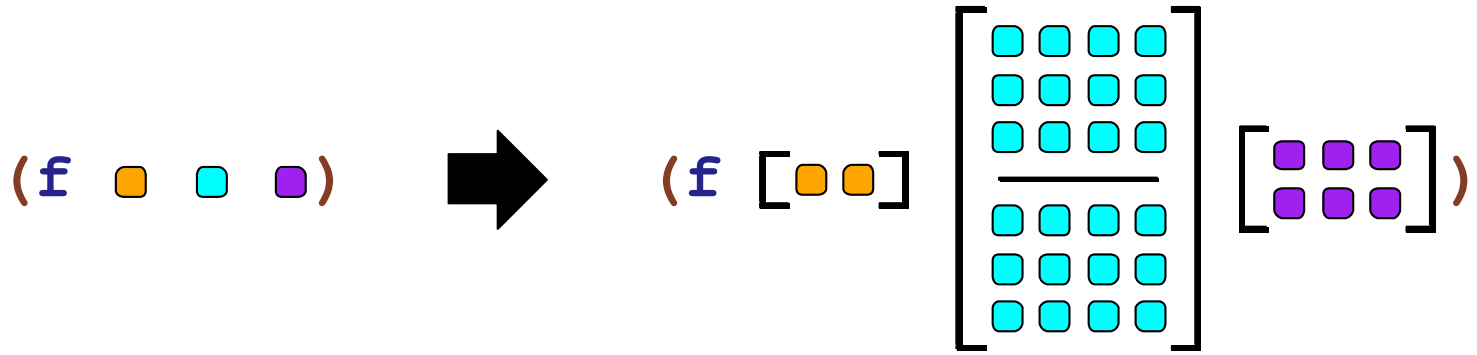
Remora: Higher-order rank-polymorphic language

Functions work on arbitrarily high-dimensional data

The Remora Project

Remora: Higher-order rank-polymorphic language

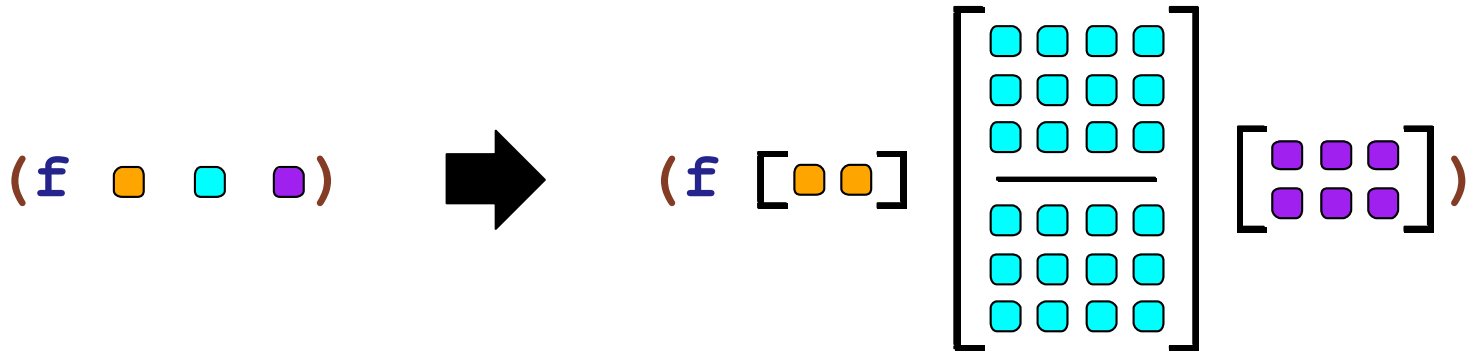
Functions work on arbitrarily high-dimensional data

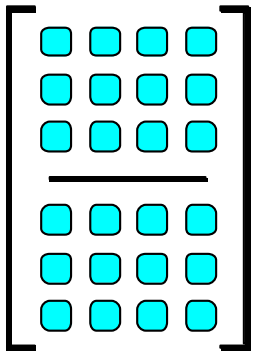


The Remora Project

Remora: Higher-order rank-polymorphic language

Functions work on arbitrarily high-dimensional data

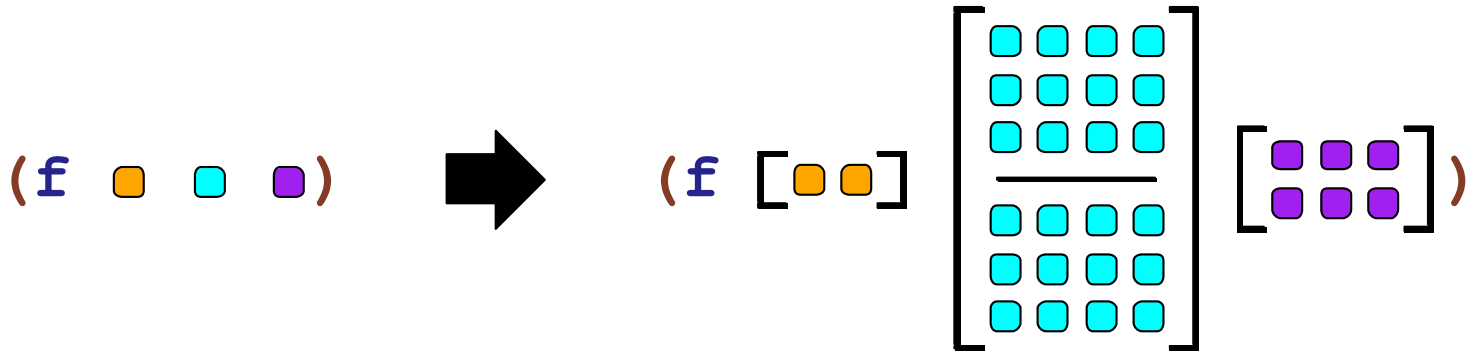


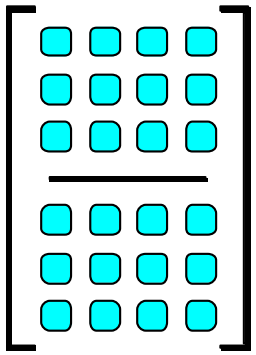


The Remora Project

Remora: Higher-order rank-polymorphic language

Functions work on arbitrarily high-dimensional data



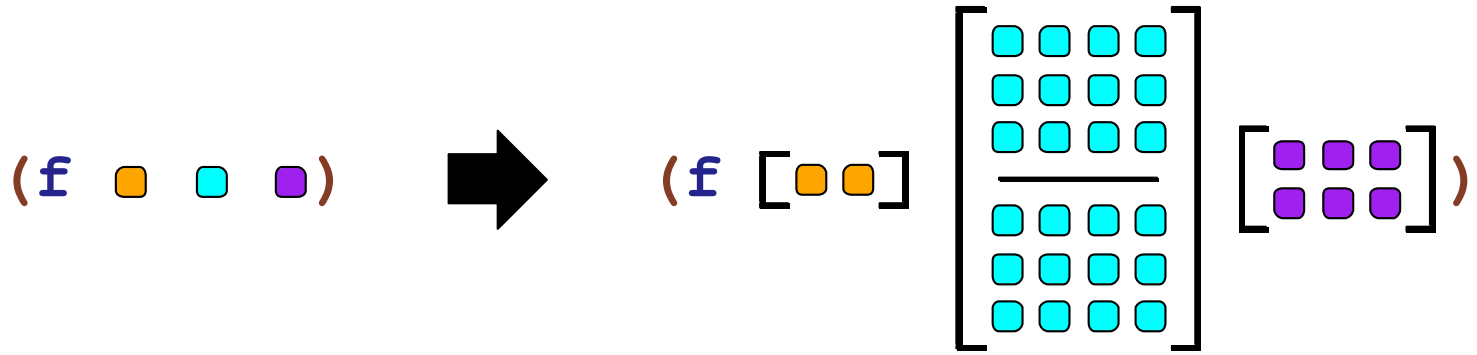


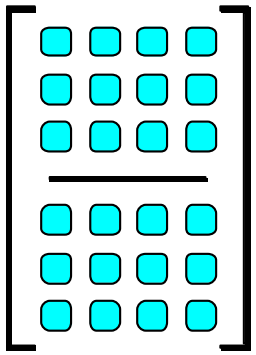
```
for (i = 0; i < 2; i++) {
    for (j = 0; j < 3; j++) {
        for (k = 0; k < 4; k++) {
            ...
        }
    }
}
```

The Remora Project

Remora: Higher-order rank-polymorphic language

Functions work on arbitrarily high-dimensional data





```
for (i = 0; i < 2; i++) {
    for (j = 0; j < 3; j++) {
        for (k = 0; k < 4; k++) {
            ...
        }
    }
}
```

Can we statically determine implicit control structure?

How This Happened

Visited TensorFlow team at Google

How This Happened

Visited TensorFlow team at Google

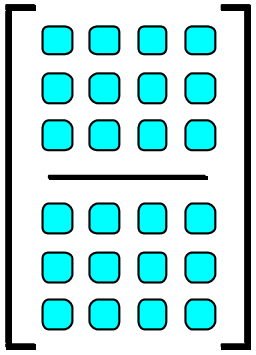
"Can you make something like Pandas data frames?"

How This Happened

Visited TensorFlow team at Google

"Can you make something like Pandas data frames?"

Remora: homogeneous data

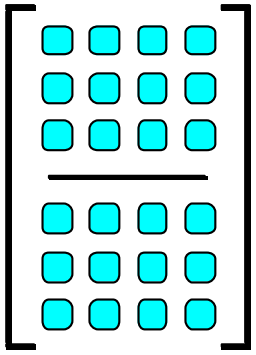


How This Happened

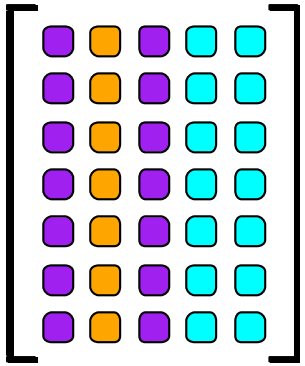
Visited TensorFlow team at Google

"Can you make something like Pandas data frames?"

Remora: homogeneous data



Data frame: columnar table



Two Kinds of Aggregate Data

Two Kinds of Aggregate Data

Arrays

Two Kinds of Aggregate Data

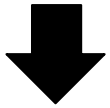
Arrays

Homogeneous data

Two Kinds of Aggregate Data

Arrays

Homogeneous data

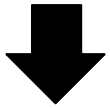


Consume uniformly

Two Kinds of Aggregate Data

Arrays

Homogeneous data



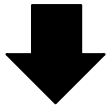
Consume uniformly

Rank polymorphism

Two Kinds of Aggregate Data

Arrays

Homogeneous data



Consume uniformly

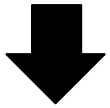
Rank polymorphism

Records

Two Kinds of Aggregate Data

Arrays

Homogeneous data



Consume uniformly

Rank polymorphism

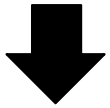
Records

Heterogeneous data

Two Kinds of Aggregate Data

Arrays

Homogeneous data



Consume uniformly

Rank polymorphism

Records

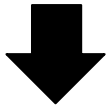
Heterogeneous data

Consume individually

Two Kinds of Aggregate Data

Arrays

Homogeneous data



Consume uniformly

Rank polymorphism

Records

Heterogeneous data

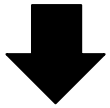
Consume individually

Field projection

Two Kinds of Aggregate Data

Arrays

Homogeneous data



Consume uniformly

Rank polymorphism

Records

Heterogeneous data

Consume individually

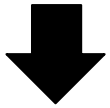
Field projection

Data Frames ≅ Arrays + Records

Two Kinds of Aggregate Data

Arrays

Homogeneous data



Consume uniformly

Rank polymorphism

Records

Heterogeneous data

Consume individually

Field projection

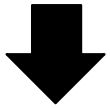
Data Frames ≅ Arrays + Records

How to design records?

Two Kinds of Aggregate Data

Arrays

Homogeneous data



Consume uniformly

Rank polymorphism

Records

Heterogeneous data

Consume individually

Field projection

Data Frames ≅ Arrays + Records

How to design records?

(to work with rank polymorphism)

Records with Rank Polymorphism

Small collection of
mutually composable
constructs

Records with Rank Polymorphism

Small collection of
mutually composable
constructs

1. Rank polymorphism in Remora
2. Data frames in Pandas
3. Synthesis design
4. Conclusion

Rank Polymorphism

Data Model

Data Model

Atoms

Data Model

Atoms

Arrays

Data Model

Atoms `0` `1` `9.25` `-4+i` `'s'` `#t` `#f`

Arrays

Data Model

Atoms `0  1  9.25  -4+i  's'  #t  #f`

Arrays

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}_{3,2}$$

Data Model

Atoms `0` `1` `9.25` `-4+i` `'s'` `#t` `#f`

Arrays

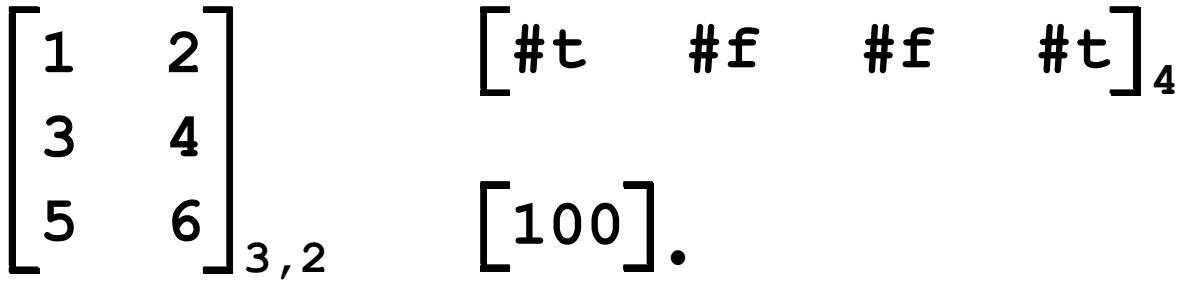
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}_{3,2}$$

$$\begin{bmatrix} \texttt{\#t} & \texttt{\#f} & \texttt{\#f} & \texttt{\#t} \end{bmatrix}_{4}$$

Data Model

Atoms `0` `1` `9.25` `-4+i` `'s'` `#t` `#f`

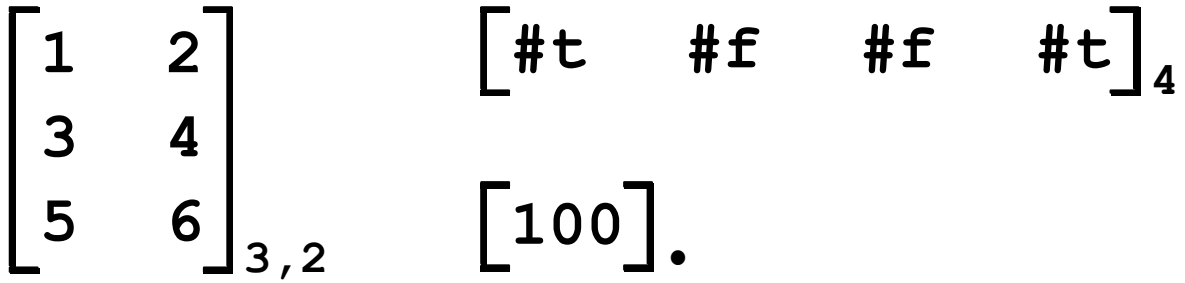
Arrays



Data Model

Atoms `0` `1` `9.25` `-4+i` `'s'` `#t` `#f`

Arrays

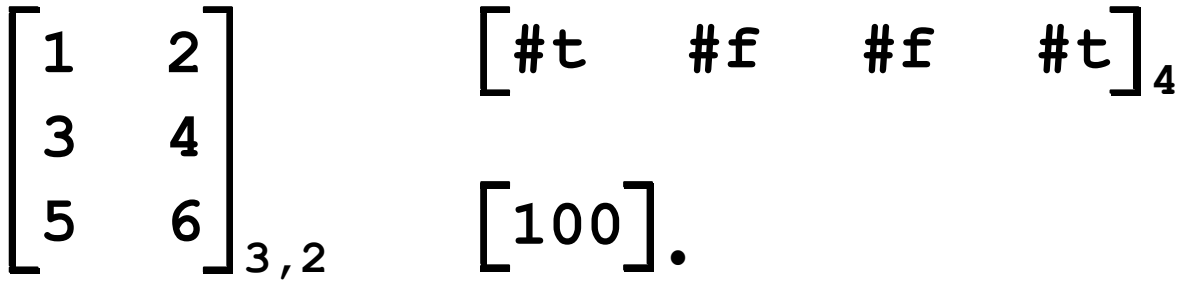


Shape Sequence of sizes in each dimension

Data Model

Atoms `0  1  9.25  -4+i  's'  #t  #f`

Arrays



Shape Sequence of sizes in each dimension

Rank Number of dimensions an array has

Data Model

Atoms `0` `1` `9.25` `-4+i` `'s'` `#t` `#f`

Arrays $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}_{3,2}$ $\begin{bmatrix} \texttt{\#t} & \texttt{\#f} & \texttt{\#f} & \texttt{\#t} \end{bmatrix}_4$ $\begin{bmatrix} 100 \end{bmatrix}.$

Shape Sequence of sizes in each dimension

Rank Number of dimensions an array has

Expressions only stand for arrays, not atoms

Decomposing an Array

Decomposing an Array

Cells Individual sub-arrays a function will consume

Decomposing an Array

Cells	Individual sub-arrays a function will consume
Frame	Aggregate structure around cells

Decomposing an Array

Cells Individual sub-arrays a function will consume

Frame Aggregate structure around cells

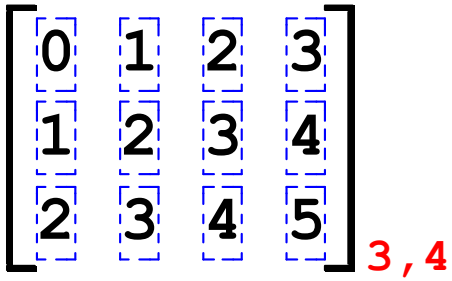
Rank n array can split n+1 ways

Decomposing an Array

Cells Individual sub-arrays a function will consume

Frame Aggregate structure around cells

Rank n array can split n+1 ways



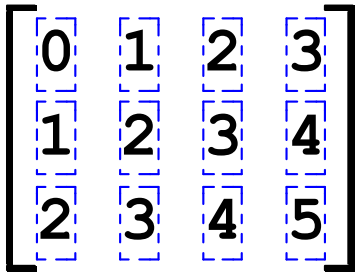
3×4 matrix frame,
twelve scalar cells

Decomposing an Array

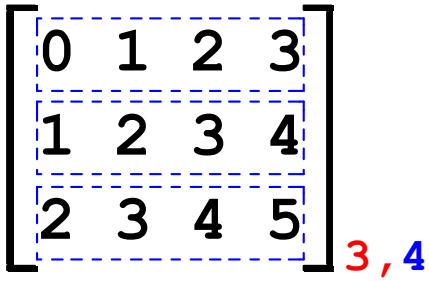
Cells Individual sub-arrays a function will consume

Frame Aggregate structure around cells

Rank n array can split n+1 ways



3×4 matrix frame,
twelve scalar cells



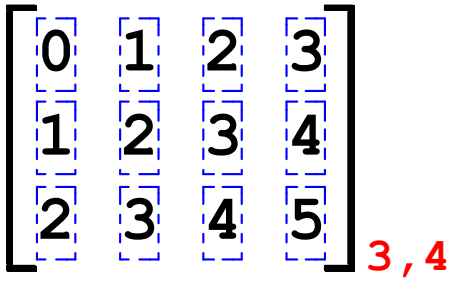
3-vector frame,
three 4-vector cells

Decomposing an Array

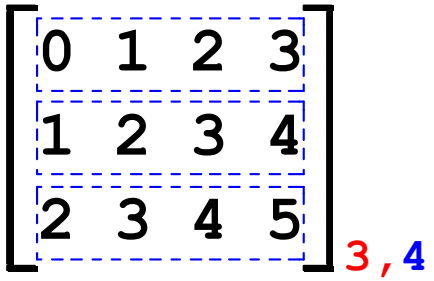
Cells Individual sub-arrays a function will consume

Frame Aggregate structure around cells

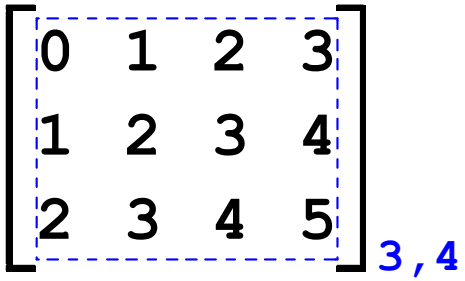
Rank n array can split n+1 ways



3×4 matrix frame,
twelve scalar cells



3-vector frame,
three 4-vector cells



scalar frame,
one 3×4 matrix cell

Arrays

Arrays

Abstract value

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

Arrays

Abstract value

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

$$\left[\begin{array}{cc} 0 & 1 \\ 2 & 3 \\ \hline 4 & 5 \\ 6 & 7 \end{array}\right]_{2,2,2}$$

Arrays

Abstract value

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

$$\left[\begin{array}{cc} 0 & 1 \\ 2 & 3 \\ \hline 4 & 5 \\ 6 & 7 \end{array}\right]_{2,2,2}$$

$$[0].$$

Arrays

Abstract value

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ \hline 4 & 5 \\ 6 & 7 \end{bmatrix}_{2,2,2}$$

$$[0].$$

Remora syntax

```
[[1 0 0 1 1 0 1]
 [0 1 0 1 0 1 1]
 [0 0 1 0 1 1 1]]
```

Arrays

Abstract value

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

$$\left[\begin{array}{cc} 0 & 1 \\ 2 & 3 \\ \hline 4 & 5 \\ 6 & 7 \end{array}\right]_{2,2,2}$$

$$[0].$$

Remora syntax

```
[[1 0 0 1 1 0 1]
 [0 1 0 1 0 1 1]
 [0 0 1 0 1 1 1]]
```

```
[[[0 1]
  [2 3]]
 [[4 5]
  [6 7]]]
```

Arrays

Abstract value

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

$$\left[\begin{array}{cc} 0 & 1 \\ 2 & 3 \\ \hline 4 & 5 \\ 6 & 7 \end{array}\right]_{2,2,2}$$

$$[0].$$

Remora syntax

```
[[1 0 0 1 1 0 1]
 [0 1 0 1 0 1 1]
 [0 0 1 0 1 1 1]]
```

```
[[[0 1]
  [2 3]]
 [[4 5]
  [6 7]]]
```

```
0
```

Function Application

```
(+ [10 20] [[1 2]
            [3 4]])
```

Function Application

```
(+ [10 20] [[1 2]
            [3 4]])
```

Cells: `10, 20` `1, 2, 3, 4`

Function Application

```
(+ [10 20] [[1 2]
            [3 4]])
```

Cells: `10, 20` `1, 2, 3, 4`

Frame: `[2]` `[2 2]`

Function Application

```
(+ [10 20] [[1 2]
            [3 4]])
```

Cells:	`10, 20`	`1, 2, 3, 4`
Frame:	`[2]`	`[2 2]`
Lifted:	`[[10 10]` `[20 20]]`	`[[1 2]` `[3 4]]`

Function Application

```
(+ [10 20] [[1 2]
            [3 4]])
```

Cells:	`10, 20`	`1, 2, 3, 4`
Frame:	`[2]`	`[2 2]`
Lifted:	`[[10 10]` `[20 20]]`	`[[1 2]` `[3 4]]`

```
(v+ [10 20] [[1 2]
             [3 4]])
```

Function Application

```
(+ [10 20] [[1 2]
            [3 4]])
```

Cells:	`10, 20`	`1, 2, 3, 4`
Frame:	`[2]`	`[2 2]`
Lifted:	`[[10 10]` `[20 20]]`	`[[1 2]` `[3 4]]`

```
(v+ [10 20] [[1 2]
             [3 4]])
```

Cells:	`[10 20]`	`[1 2], [3 4]`

Function Application

```
(+ [10 20] [[1 2]
            [3 4]])
```

Cells:	`10, 20`	`1, 2, 3, 4`
Frame:	`[2]`	`[2 2]`
Lifted:	`[[10 10]` `[20 20]]`	`[[1 2]` `[3 4]]`

```
(v+ [10 20] [[1 2]
             [3 4]])
```

Cells:	`[10 20]`	`[1 2], [3 4]`
Frame:	`[]`	`[2]`

Function Application

```
(+ [10 20] [[1 2]
            [3 4]])
```

Cells:	`10, 20`	`1, 2, 3, 4`
Frame:	`[2]`	`[2 2]`
Lifted:	`[[10 10]` `[20 20]]`	`[[1 2]` `[3 4]]`

```
(v+ [10 20] [[1 2]
             [3 4]])
```

Cells:	`[10 20]`	`[1 2], [3 4]`
Frame:	`[]`	`[2]`
Lifted:	`[[10 20]` `[10 20]]`	`[[1 2]` `[3 4]]`

Function Application

```
(+ [10 20] [[1 2]
            [3 4]])
```

Cells:	`+`	`10, 20`	`1, 2, 3, 4`
Frame:	`[]`	`[2]`	`[2 2]`
Lifted:	`[[+ +]` `[+ +]]`	`[[10 10]` `[20 20]]`	`[[1 2]` `[3 4]]`

```
(v+ [10 20] [[1 2]
             [3 4]])
```

Cells:		`[10 20]`	`[1 2], [3 4]`
Frame:		`[]`	`[2]`
Lifted:		`[[10 20]` `[10 20]]`	`[[1 2]` `[3 4]]`

Function Application

```
(+ [10 20] [[1 2]
            [3 4]])
```

Cells:	`+`	`10, 20`	`1, 2, 3, 4`
Frame:	`[]`	`[2]`	`[2 2]`
Lifted:	`[[+ +]` `[+ +]]`	`[[10 10]` `[20 20]]`	`[[1 2]` `[3 4]]`

```
(v+ [10 20] [[1 2]
             [3 4]])
```

Cells:	`v+`	`[10 20]`	`[1 2], [3 4]`
Frame:	`[]`	`[]`	`[2]`
Lifted:	`[v+ v+]`	`[[10 20]` `[10 20]]`	`[[1 2]` `[3 4]]`

Function Application

```
(+ [10 20] [[1 2]      ↦    [[11 12]
           [3 4]])           [23 24]]
```

Cells:	`+`	`10, 20`	`1, 2, 3, 4`
Frame:	`[]`	`[2]`	`[2 2]`
Lifted:	`[[+ +]` `[+ +]]`	`[[10 10]` `[20 20]]`	`[[1 2]` `[3 4]]`

```
(v+ [10 20] [[1 2]
            [3 4]])
```

Cells:	`v+`	`[10 20]`	`[1 2], [3 4]`
Frame:	`[]`	`[]`	`[2]`
Lifted:	`[v+ v+]`	`[[10 20]` `[10 20]]`	`[[1 2]` `[3 4]]`

Function Application

```
(+ [10 20] [[1 2]      ↦   [[11 12]
           [3 4]])            [23 24]]
```

Cells:	`+`	`10, 20`	`1, 2, 3, 4`
Frame:	`[]`	`[2]`	`[2 2]`
Lifted:	`[[+ +]` `[+ +]]`	`[[10 10]` `[20 20]]`	`[[1 2]` `[3 4]]`

```
(v+ [10 20] [[1 2]      ↦   [[11 22]
            [3 4]])            [13 24]]
```

Cells:	`v+`	`[10 20]`	`[1 2], [3 4]`
Frame:	`[]`	`[]`	`[2]`
Lifted:	`[v+ v+]`	`[[10 20]` `[10 20]]`	`[[1 2]` `[3 4]]`

Expected Argument Rank

Frame/cell split determined by function

Expected Argument Rank

Frame/cell split determined by function

```
        +    0, 0
 dot-prod    1, 1
     minv    2
     lerp    0, 0, 0
poly-eval    1, 0
```

Expected Argument Rank

Frame/cell split determined by function

```
          +    0, 0
  dot-prod     1, 1
       minv    2
       lerp    0, 0, 0
  poly-eval    1, 0
```

```
(define (lerp (lo 0)
              (hi 0)
              (α 0))
  (+ (* α hi)
     (* (- 1 α) lo)))
```

Expected Argument Rank

Frame/cell split determined by function

```
        +    0, 0
 dot-prod    1, 1
     minv    2
     lerp    0, 0, 0
poly-eval    1, 0
```

```
(define (lerp (lo 0)
              (hi 0)
              (α 0))
  (+ (* α hi)
     (* (- 1 α) lo)))
```

Change argument rank by η-expansion

Expected Argument Rank

Frame/cell split determined by function

```
        +    0, 0
 dot-prod    1, 1
     minv    2
     lerp    0, 0, 0
poly-eval    1, 0
```

```
(define (lerp (lo 0)
              (hi 0)
              (α 0))
  (+ (* α hi)
     (* (- 1 α) lo)))
```

Change argument rank by η-expansion

```
v+
```

Expected Argument Rank

Frame/cell split determined by function

```
        +    0, 0
dot-prod     1, 1
     minv    2
     lerp    0, 0, 0
poly-eval    1, 0
```

```
(define (lerp (lo 0)
              (hi 0)
              (α 0))
  (+ (* α hi)
     (* (- 1 α) lo)))
```

Change argument rank by η-expansion

```
v+  =  (λ ((a 1) (b 1))
          (+ a b))
```

Expected Argument Rank

Frame/cell split determined by function

```
          +    0, 0
  dot-prod     1, 1
       minv    2
       lerp    0, 0, 0
 poly-eval     1, 0
```

```
(define (lerp (lo 0)
              (hi 0)
              (α 0))
  (+ (* α hi)
     (* (- 1 α) lo)))
```

Change argument rank by η-expansion

```
v+  =  (λ ((a 1) (b 1))
          (+ a b))   =   ~(1 1)+
```

Ragged Data

`iota` **`input`** **`filter`** **`reshape`**

Ragged Data

`iota` **`input`** **`filter`** **`reshape`**

Output shape depends on input atoms

Ragged Data

iota input filter reshape

Output shape depends on input atoms

```
> (iota [3])
```

Ragged Data

iota input filter reshape

Output shape depends on input atoms

```
> (iota [3])

  [0 1 2]
```

Ragged Data

iota input filter reshape

Output shape depends on input atoms

```
> (iota [3])
```

```
[0 1 2]
```

```
> (iota [[3]
         [2]
         [4]])
```

Ragged Data

iota input filter reshape

Output shape depends on input atoms

```
> (iota [3])
[0 1 2]
```

```
> (iota [[3]
         [2]
         [4]])
[[0 1 2]
 [0 1]
 [0 1 2 3]]
```

Ragged Data

iota input filter reshape

Output shape depends on input atoms

```
> (iota [3])
[0 1 2]
```

```
> (iota [[3]
         [2]
         [4]])
[[0 1 2]
 [0 1]
 [0 1 2 3]]
```

$$\begin{bmatrix} 0 & 1 & 2 & \\ 0 & 1 & & \\ 0 & 1 & 2 & 3 \end{bmatrix}_{3,?}$$

Ragged Data

iota input filter reshape

Output shape depends on input atoms

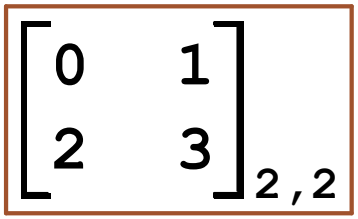
```
> (iota [3])
[0 1 2]
```

```
> (iota [[3]
         [2]
         [4]])
```

$$\begin{bmatrix} 0 & 1 & 2 & \\ 0 & 1 & & \\ 0 & 1 & 2 & 3 \end{bmatrix}_{3,?}$$

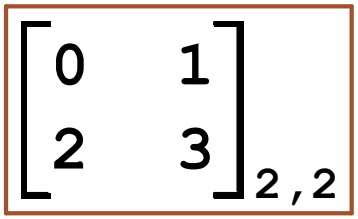
Ragged Data

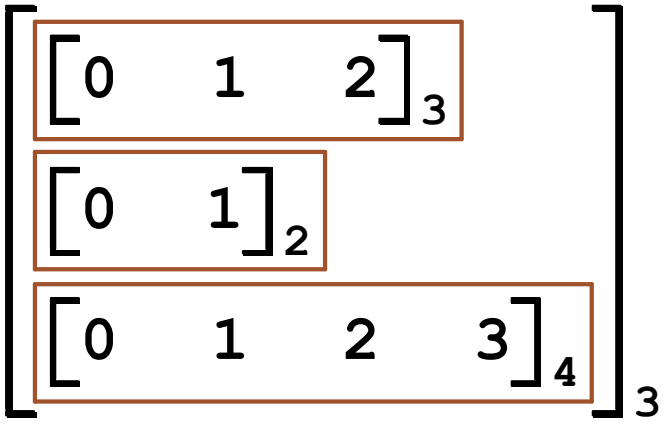
New type of atom: *boxed* array



Ragged Data

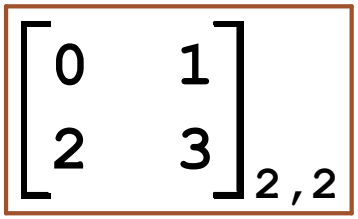
New type of atom: *boxed* array

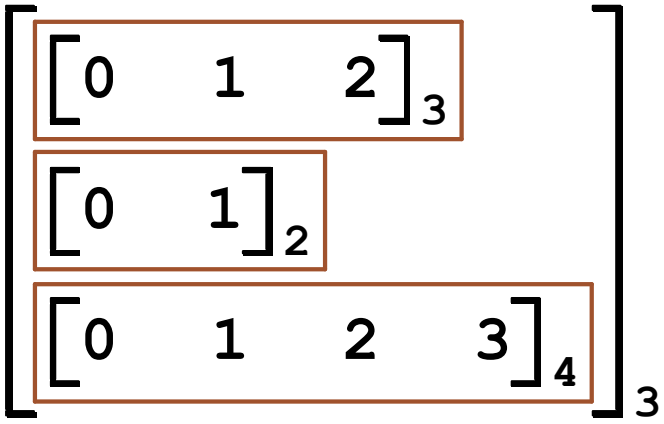




Ragged Data

New type of atom: *boxed* array

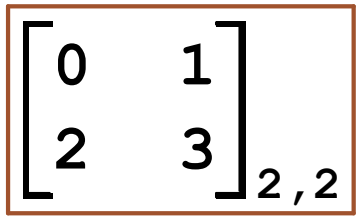


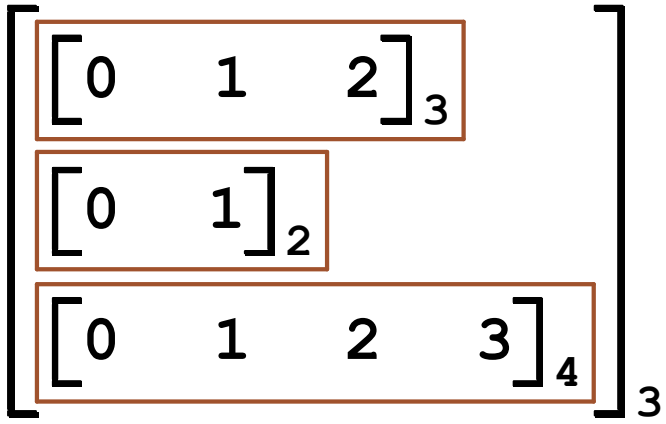


Lift-safe variant `iota*`

Ragged Data

New type of atom: *boxed* array



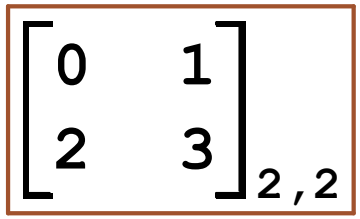


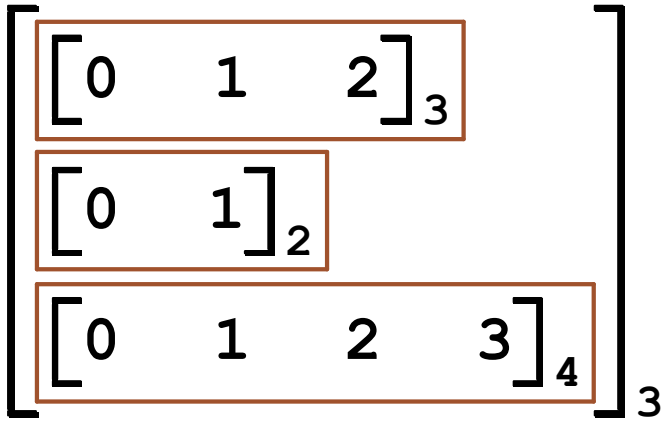
Lift-safe variant `iota*`

```
> (iota* [[3] [2] [4]])
```

Ragged Data

New type of atom: *boxed* array





Lift-safe variant `iota*`

```
> (iota* [[3] [2] [4]])

[(box [0 1 2])
 (box [0 1])
 (box [0 1 2 3])]
```

Pandas DataFrame

Python Dictionary

```
>>> dallas_temp = {'loc': 'Dallas', 'day': 28,
...                 'month': 3, 'year': 2019,
...                 'hi': 74, 'lo': 57}
```

Python Dictionary

```
>>> dallas_temp = {'loc': 'Dallas', 'day': 28,
...                 'month': 3, 'year': 2019,
...                 'hi': 74, 'lo': 57}

>>> temp_list = [dallas_temp,
...               {'loc': 'Dublin', 'day': 1,
...                'month': 4, 'year': 2019,
...                'hi': 74, 'lo': 57}
...               {'loc': 'Nome', 'day': 31,
...                'month': 3, 'year': 2019,
...                'hi': 31, 'lo': 26}
...               {'loc': 'Tunis', 'day': 31,
...                'month': 3, 'year': 2019,
...                'hi': 21, 'lo': 12}]
```

Pandas DataFrame

More sophisticated tool

Pandas DataFrame

More sophisticated tool

```
>>> by_rows = pd.DataFrame(temp_list)
```

Pandas DataFrame

More sophisticated tool

```
>>> by_rows = pd.DataFrame(temp_list)

>>> by_cols =
... pd.DataFrame({'loc': ['Dallas', 'Dublin',
...                       'Nome', 'Tunis'],
...                 'day': [28, 1, 31, 31],
...                 'month': [3, 4, 3, 3],
...                 'year': 2019,
...                 'hi': [74, 11, 31, 21],
...                 'lo': [57, 5, 26, 12]})
```

Column Subset

Dictionary of **`Series`** objects

Column Subset

Dictionary of **`Series`** objects

```
>>> by_cols['loc']
```

Column Subset

Dictionary of **`Series`** objects

```
>>> by_cols['loc']
0    Dallas
1    Dublin
2      Nome
3     Tunis
```

Column Subset

Dictionary of **`Series`** objects

```
>>> by_cols['loc']
0    Dallas
1    Dublin
2      Nome
3     Tunis
```

Ask for list of columns, get new **`DataFrame`**

Column Subset

Dictionary of **`Series`** objects

```
>>> by_cols['loc']
0    Dallas
1    Dublin
2      Nome
3     Tunis
```

Ask for list of columns, get new **`DataFrame`**

```
>>> by_cols[['loc', 'hi']]
```

Column Subset

Dictionary of **`Series`** objects

```
>>> by_cols['loc']
0    Dallas
1    Dublin
2      Nome
3     Tunis
```

Ask for list of columns, get new **`DataFrame`**

```
>>> by_cols[['loc', 'hi']]
        loc  hi
0    Dallas  74
1    Dublin  11
2      Nome  31
3     Tunis  21
```

Column Update

```
>>> def normalize_temp(r):
...     if in_usa(r['loc']):
...             r['lo'] = f2c(r['lo'])
...             r['hi'] = f2c(r['hi'])
...     return r
```

Column Update

```
>>> def normalize_temp(r):
...     if in_usa(r['loc']):
...             r['lo'] = f2c(r['lo'])
...             r['hi'] = f2c(r['hi'])
...     return r

>>> by_cols.apply(normalize_temp, axis=1)
```

Column Update

```
>>> def normalize_temp(r):
...     if in_usa(r['loc']):
...             r['lo'] = f2c(r['lo'])
...             r['hi'] = f2c(r['hi'])
...     return r

>>> by_cols.apply(normalize_temp, axis=1)
      loc  day  month  year     hi     lo
0 Dallas   28      3  2019  23.33  13.89
1 Dublin    1      4  2019  11.00   5.00
2   Nome   31      3  2019  -0.56  -3.33
3  Tunis   31      3  2019  21.00  12.00
```

Row Filter

```
>>> by_cols['loc'] == 'Dublin'
```

Row Filter

```
>>> by_cols['loc'] == 'Dublin'
0    False
1    True
2    False
3    False
```

Row Filter

```
>>> by_cols['loc'] == 'Dublin'
0    False
1    True
2    False
3    False

>>> by_cols[by_cols['loc'] == 'Dublin']
```

Row Filter

```
>>> by_cols['loc'] == 'Dublin'
0    False
1    True
2    False
3    False

>>> by_cols[by_cols['loc'] == 'Dublin']
      loc  day  month  year  hi  lo
1  Dublin    1      4  2019  11   5
```

Row Filter

Lifting user-defined functions?

Row Filter

Lifting user-defined functions?

```
>>> in_usa(by_cols['loc'])
```

Row Filter

Lifting user-defined functions?

```
>>> in_usa(by_cols['loc'])
ValueError: The truth value
of a Series is ambiguous.
```

Row Filter

Lifting user-defined functions?

```
>>> in_usa(by_cols['loc'])
ValueError: The truth value
of a Series is ambiguous.

>>> [in_usa(l) for l in by_cols['loc']]
```

Row Filter

Lifting user-defined functions?

```
>>> in_usa(by_cols['loc'])
ValueError: The truth value
of a Series is ambiguous.

>>> [in_usa(l) for l in by_cols['loc']]
[True, False, True, False]
```

Row Filter

Lifting user-defined functions?

```
>>> in_usa(by_cols['loc'])
ValueError: The truth value
of a Series is ambiguous.

>>> [in_usa(l) for l in by_cols['loc']]
[True, False, True, False]

>>> by_cols[[in_usa(l) for l in
             by_cols['loc']]]
```

Row Filter

Lifting user-defined functions?

```
>>> in_usa(by_cols['loc'])
ValueError: The truth value
of a Series is ambiguous.

>>> [in_usa(l) for l in by_cols['loc']]
[True, False, True, False]

>>> by_cols[[in_usa(l) for l in
             by_cols['loc']]]
      loc  day  month  year  hi  lo
0  Dallas   28      3  2019  74  57
2    Nome   31      3  2019  31  26
```

Row Partition

No lifting over masks

Row Partition

No lifting over masks

```
>>> us__non_us =
... by_cols.groupby([in_usa(l) for l in
...                  by_cols['loc']])
```

Row Partition

No lifting over masks

```
>>> us__non_us =
... by_cols.groupby([in_usa(l) for l in
...                  by_cols['loc']])

>>> us__non_us.get_group(True)
      loc  day  month  year  hi  lo
0  Dallas   28      3  2019  74  57
2    Nome   31      3  2019  31  26
```

Row Partition

No lifting over masks

```
>>> us__non_us =
... by_cols.groupby([in_usa(l) for l in
...                  by_cols['loc']])

>>> us__non_us.get_group(True)
      loc  day  month  year  hi  lo
0  Dallas   28      3  2019  74  57
2    Nome   31      3  2019  31  26

>>> us__non_us.get_group(False)
      loc  day  month  year  hi  lo
1  Dublin    1      4  2019  11   5
3   Tunis   31      3  2019  21  12
```

Ergonomics

- ✔ Wide range of functionality

Ergonomics

- ✔ Wide range of functionality
- ✔ Support for row-wise and column-wise operations

Ergonomics

- ✔ Wide range of functionality
- ✔ Support for row-wise and column-wise operations
- ✖ Many *ad hoc* structures and methods

Ergonomics

- ✓ Wide range of functionality
- ✓ Support for row-wise and column-wise operations
- ✗ Many *ad hoc* structures and methods
- ✗ Lifting rules do not generalize

Ergonomics

- ✔ Wide range of functionality
- ✔ Support for row-wise and column-wise operations
- ✖ Many *ad hoc* structures and methods
- ✖ Lifting rules do not generalize

"It is better to have 100 functions operate on one data structure than 10 functions on 10 data structures."
— Alan Perlis

Synthesis Design

Synthesis Design Goals

Create/project/update with *anonymous* functions

Synthesis Design Goals

Create/project/update with *anonymous* functions

Eventual possibility of static typing

Records

Constructor function

(record $fname_1$ **...** $fname_n$**)**

Records

Constructor function

(record $fname_1$ **...** $fname_n$**)**

Can still support "literal" syntax

{ ($fname_1$ arg_1**)** **...** **(**$fname_n$ arg_n**) }**

Records

Constructor function

(record $fname_1$ **...** $fname_n$**)**

Can still support "literal" syntax

{ ($fname_1$ arg_1**)** **...** **(**$fname_n$ arg_n**) }**

((record $fname_1$ **...** $fname_n$**)** arg_1 **...** arg_n**)**

Lenses

Function for focusing on piece of data structure

(lens *fname*)

Lenses

Function for focusing on piece of data structure

(lens *fname*)

Composition → focus on nested pieces

(compose (lens $fname_1$**)** ... **(lens** $fname_n$**))**

Lenses

Function for focusing on piece of data structure

(lens *fname*)

Composition → focus on nested pieces

(compose (lens $fname_1$**)** ... **(lens** $fname_n$**))**

Three operations on lenses

view **set** **over**

Lens Operations

`((view L) R)` ↦ Field `L` from within `R`

Lens Operations

`((view L) R)` ↦ Field `L` from within `R`

`(((set L) V) R)` ↦ `R` with field `L` changed to `V`

Lens Operations

`((view L) R)`	↦	Field `L` from within `R`
`(((set L) V) R)`	↦	`R` with field `L` changed to `V`
`(((over L) F) R)`	↦	`R` with `F` applied to field `L`

Lens Operations

`((view L) R)`	↦	Field `L` from within `R`
`(((set L) V) R)`	↦	`R` with field `L` changed to `V`
`(((over L) F) R)`	↦	`R` with `F` applied to field `L`

Syntactic sugar

Lens Operations

`((view L) R)`	↦	Field `L` from within `R`
`(((set L) V) R)`	↦	`R` with field `L` changed to `V`
`(((over L) F) R)`	↦	`R` with `F` applied to field `L`

Syntactic sugar

`#_(`*fname* `...)`

`(view (compose (lens` *fname*`) ...))`

Lens Operations

`((view L) R)`	↦	Field `L` from within `R`
`(((set L) V) R)`	↦	`R` with field `L` changed to `V`
`(((over L) F) R)`	↦	`R` with `F` applied to field `L`

Syntactic sugar

```
#_(fname ...)
(view (compose (lens fname) ...))
```

```
#=(fname ...)
(set (compose (lens fname) ...))
```

Lens Operations

`((view L) R)`	↦	Field `L` from within `R`
`(((set L) V) R)`	↦	`R` with field `L` changed to `V`
`(((over L) F) R)`	↦	`R` with `F` applied to field `L`

Syntactic sugar

```
#_(fname ...)
(view (compose (lens fname) ...))
```

```
#=(fname ...)
(set (compose (lens fname) ...))
```

```
#^(fname ...)
(over (compose (lens fname) ...))
```

Record Creation

```
> (define dallas-temp
    {(loc "Dallas") (day 28) (month 3)
     (year 2019) (hi 74) (lo 57)})
```

Record Creation

```
> (define dallas-temp
    {(loc "Dallas") (day 28) (month 3)
     (year 2019) (hi 74) (lo 57)})

> (#_(year) dallas-temp)
```

Record Creation

```
> (define dallas-temp
    {(loc "Dallas") (day 28) (month 3)
     (year 2019) (hi 74) (lo 57)})

> (#_(year) dallas-temp)
2019
```

Record Creation

```
> (define dallas-temp
    {(loc "Dallas") (day 28) (month 3)
     (year 2019) (hi 74) (lo 57)})

> (#_(year) dallas-temp)
2019
> ([#_(hi) #_(lo)] dallas-temp)
```

Record Creation

```
> (define dallas-temp
    {(loc "Dallas") (day 28) (month 3)
     (year 2019) (hi 74) (lo 57)})

> (#_(year) dallas-temp)
2019
> ([#_(hi) #_(lo)] dallas-temp)
[74 57]
```

Table Creation—Two Different Ways

```
> (define temp-readings ; by rows
    [dallas-temp
     {(loc "Dublin") (day 1) (month 4)
      (year 2019) (hi 11) (lo 5)}
     {(loc "Nome") (day 31) (month 3)
      (year 2019) (hi 31) (lo 26)}
     {(loc "Tunis") (day 31) (month 3)
      (year 2019) (hi 21) (lo 12)}])
```

Table Creation—Two Different Ways

```
> (define temp-readings ; by rows
    [dallas-temp
     {(loc "Dublin") (day 1) (month 4)
      (year 2019) (hi 11) (lo 5)}
     {(loc "Nome") (day 31) (month 3)
      (year 2019) (hi 31) (lo 26)}
     {(loc "Tunis") (day 31) (month 3)
      (year 2019) (hi 21) (lo 12)}])

> (define temp-readings ; by columns
    {(loc ["Dallas" "Dublin" "Nome" "Tunis"])
     (day [28 1 31 31])
     (month [3 4 3 3])
     (year 2019)
     (hi [74 11 31 21])
     (lo [57 5 26 12])})
```

Table Creaction

```
{(loc ["Dallas" "Dublin" "Nome" "Tunis"])
 (day [28 1 31 31])
 (month [3 4 3 3])
 (year 2019)
 (hi [74 11 31 21])
 (lo [57 5 26 12])}
```

Table Creaction

```
{(loc ["Dallas" "Dublin" "Nome" "Tunis"])
 (day [28 1 31 31])
 (month [3 4 3 3])
 (year 2019)
 (hi [74 11 31 21])
 (lo [57 5 26 12])}
```

```
((record loc day month year hi lo)
 ["Dallas" "Dublin" "Nome" "Tunis"]
 [28 1 31 31]
 [3 4 3 3]
 2019
 [74 11 31 21]
 [57 5 26 12])
```

Column Extraction

```
> (#_(loc) temp-readings)
```

Column Extraction

```
> (#_(loc) temp-readings)
["Dallas" "Dublin" "Nome" "Tunis"]
```

Column Extraction

```
> (#_(loc) temp-readings)
["Dallas" "Dublin" "Nome" "Tunis"]

> (define hi-only
    {(loc   (#_(loc)   temp-readings))
     (day   (#_(day)   temp-readings))
     (month (#_(month) temp-readings))
     (hi    (#_(hi)    temp-readings))})
```

Column Extraction

```
> (#_(loc) temp-readings)
["Dallas" "Dublin" "Nome" "Tunis"]

> (define hi-only
    {(loc   (#_(loc)   temp-readings))
     (day   (#_(day)   temp-readings))
     (month (#_(month) temp-readings))
     (hi    (#_(hi)    temp-readings))})
> hi-only
[{(loc "Dallas") (day 28) (month 3) (hi 74)}
 {(loc "Dublin") (day 1) (month 4) (hi 11)}
 {(loc "Nome") (day 31) (month 3) (hi 31)}
 {(loc "Tunis") (day 31) (month 3) (hi 21)}]
```

Column Update

```
> (define (normalize-temps (w 0))
    (define fix-temp
      (select (in-usa? (#_(loc) w)) f->c id))
    ((#^(lo) fix-temp) ((#^(hi) fix-temp) w)))
```

Column Update

```
> (define (normalize-temps (w 0))
    (define fix-temp
      (select (in-usa? (#_(loc) w)) f->c id))
    ((#^(lo) fix-temp) ((#^(hi) fix-temp) w)))

> (normalize-temps temp-readings)
```

Column Update

```
> (define (normalize-temps (w 0))
    (define fix-temp
      (select (in-usa? (#_(loc) w)) f->c id))
    ((#^(lo) fix-temp) ((#^(hi) fix-temp) w)))

> (normalize-temps temp-readings)
[{(loc "Dallas") (day 28) (month 3) (year 2019)
  (hi 23.33) (lo 13.89)}
 {(loc "Dublin") (day 1) (month 4) (year 2019)
  (hi 11) (lo 5)}
 {(loc "Nome") (day 31) (month 3) (year 2019)
  (hi -0.56) (lo -3.33)}
 {(loc "Tunis") (day 31) (month 3) (year 2019)
  (hi 21) (lo 12)}]
```

Row Filter

```
> (define usa-mask
    ((compose in-usa?  #_(loc))
     hi-only))
```

Row Filter

```
> (define usa-mask
    ((compose in-usa?  #_(loc))
     hi-only))
> usa-mask
[#t #f #t #f]
```

Row Filter

```
> (define usa-mask
    ((compose in-usa?  #_(loc))
     hi-only))
> usa-mask
[#t #f #t #f]

> (filter usa-mask hi-only)
```

Row Filter

```
> (define usa-mask
    ((compose in-usa?  #_(loc))
     hi-only))
> usa-mask
[#t #f #t #f]

> (filter usa-mask hi-only)
[{(loc "Dallas") (day 28) (month 3) (hi 74)}
 {(loc "Nome") (day 31) (month 3) (hi 31)}]
```

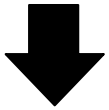
Filter → Partition

Filter → Partition

Row Filter

Filter → Partition

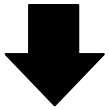
Row Filter



`filter` with one mask

Filter → Partition

Row Filter

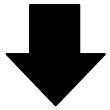


`filter` with one mask

Filter → Partition

Row Filter

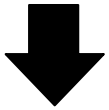
Row Partition



`filter` with one mask

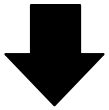
Filter → Partition

Row Filter



`filter` with one mask

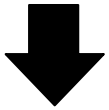
Row Partition



`filter` with multiple masks

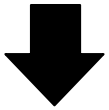
Filter → Partition

Row Filter

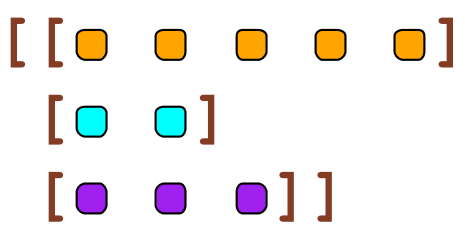


`filter` with one mask

Row Partition

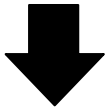


`filter` with multiple masks



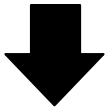
Filter → Partition

Row Filter

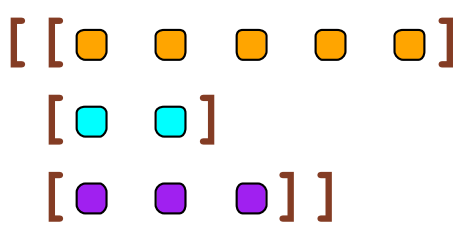


`filter` with one mask

Row Partition



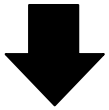
`filter` with multiple masks



Ragged data!

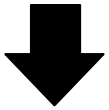
Filter → Partition

Row Filter

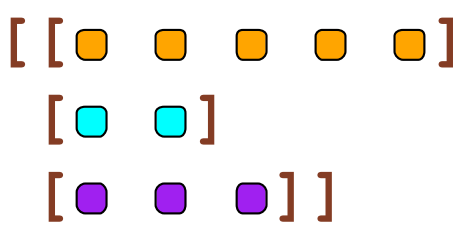


`filter*` with one mask

Row Partition



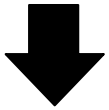
`filter` with multiple masks



Ragged data!

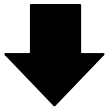
Filter → Partition

Row Filter

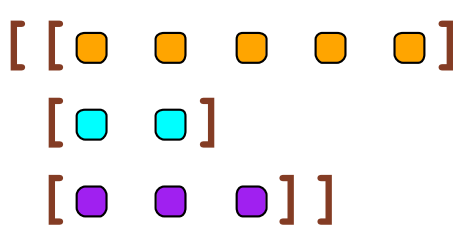


`filter*` with one mask

Row Partition



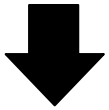
`filter` with multiple masks



Ragged data!

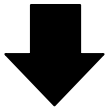
Filter → Partition

Row Filter

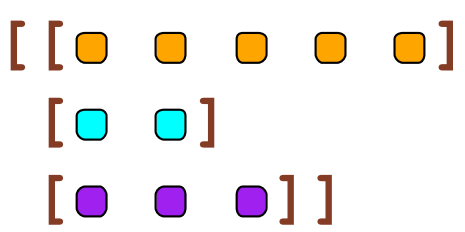


`filter*` with one mask

Row Partition



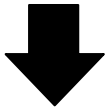
`filter*` with multiple masks



Ragged data!

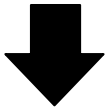
Filter → Partition

Row Filter

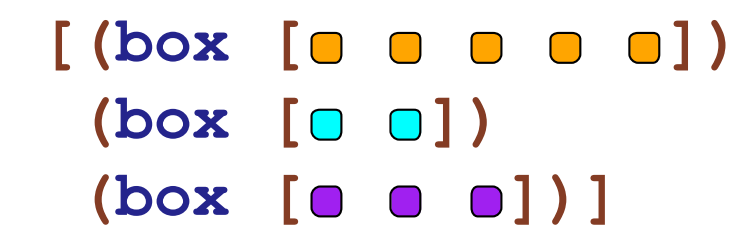


`filter*` with one mask

Row Partition



`filter*` with multiple masks



Ragged data!

Row Partition

```
> (filter* [usa-mask (not usa-mask)]
           hi-only)
```

Row Partition

```
> (filter* [usa-mask (not usa-mask)]
           hi-only)

[(box
  [{(loc "Dallas") (day 28) (month 3) (hi 74)}
   {(loc "Nome") (day 31) (month 3) (hi 31)}])
 (box
  [{(loc "Dublin") (day 1) (month 4) (hi 11)}
   {(loc "Tunis") (day 31) (month 3) (hi 21)}])]
```

How to design records?

How to design records?
(to work with rank polymorphism)

How to design records?
(to work with rank polymorphism)

Small collection of
mutually composable
constructs