Vectorization without Replication

Ben Lippmeier
University of New South Wales
WLDI 2011/08/25

Flat vs **N**ested **D**ata **P**arallelism

- ***Nested Parallelism:*** Worker function can be parallel.

```
thingo xss
  = mapP (\xs. zipWithP xs ys) xss
```

- ***Flat Parallelism:*** Worker function is sequential.

```
thingo xs
  = mapP (\x. x + 1) xs
```

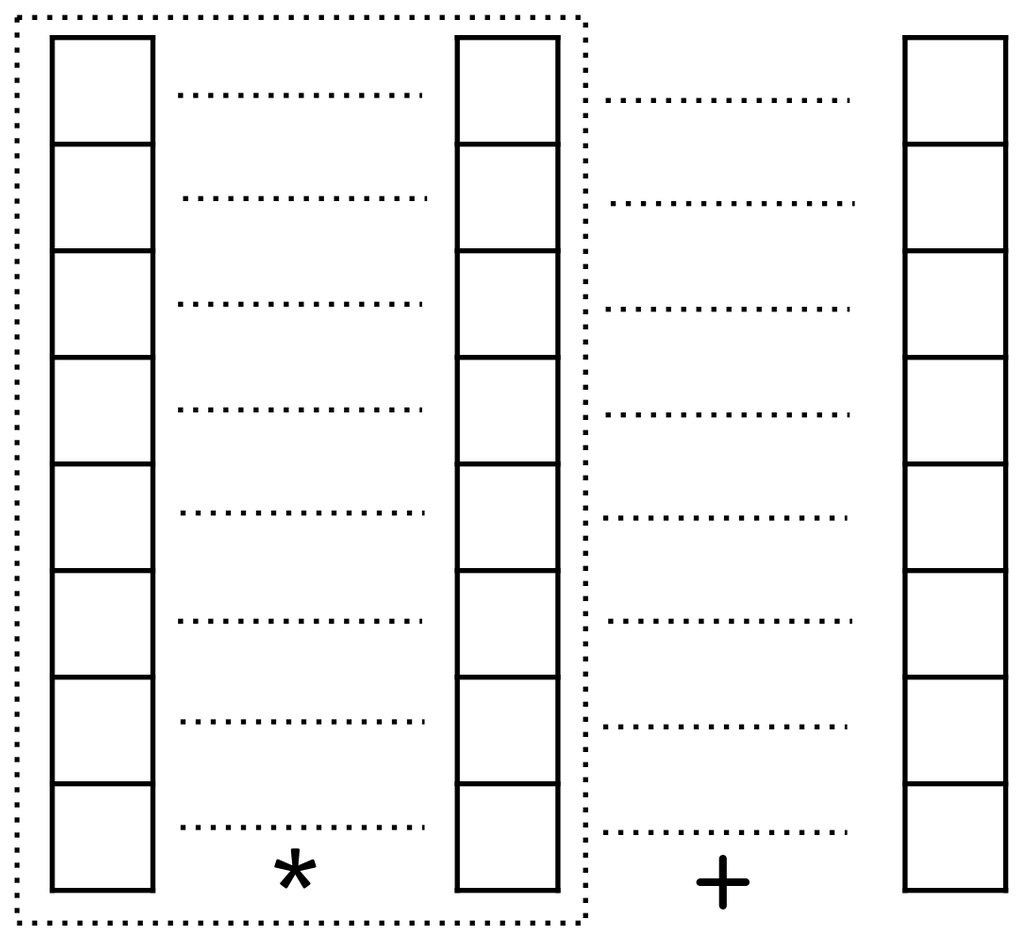
- The Flattening / Vectorization transform converts nested parallelism into flat parallelism.

Vectorization

```
f  x  y  z  = (x  *  y ) +  z
              □     □     □
```

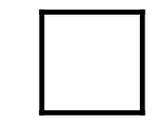
```
f^ c xs ys zs = (xs *^ ys) +^ zs
```

```
c = len xs
  = len ys
  = len zs
```

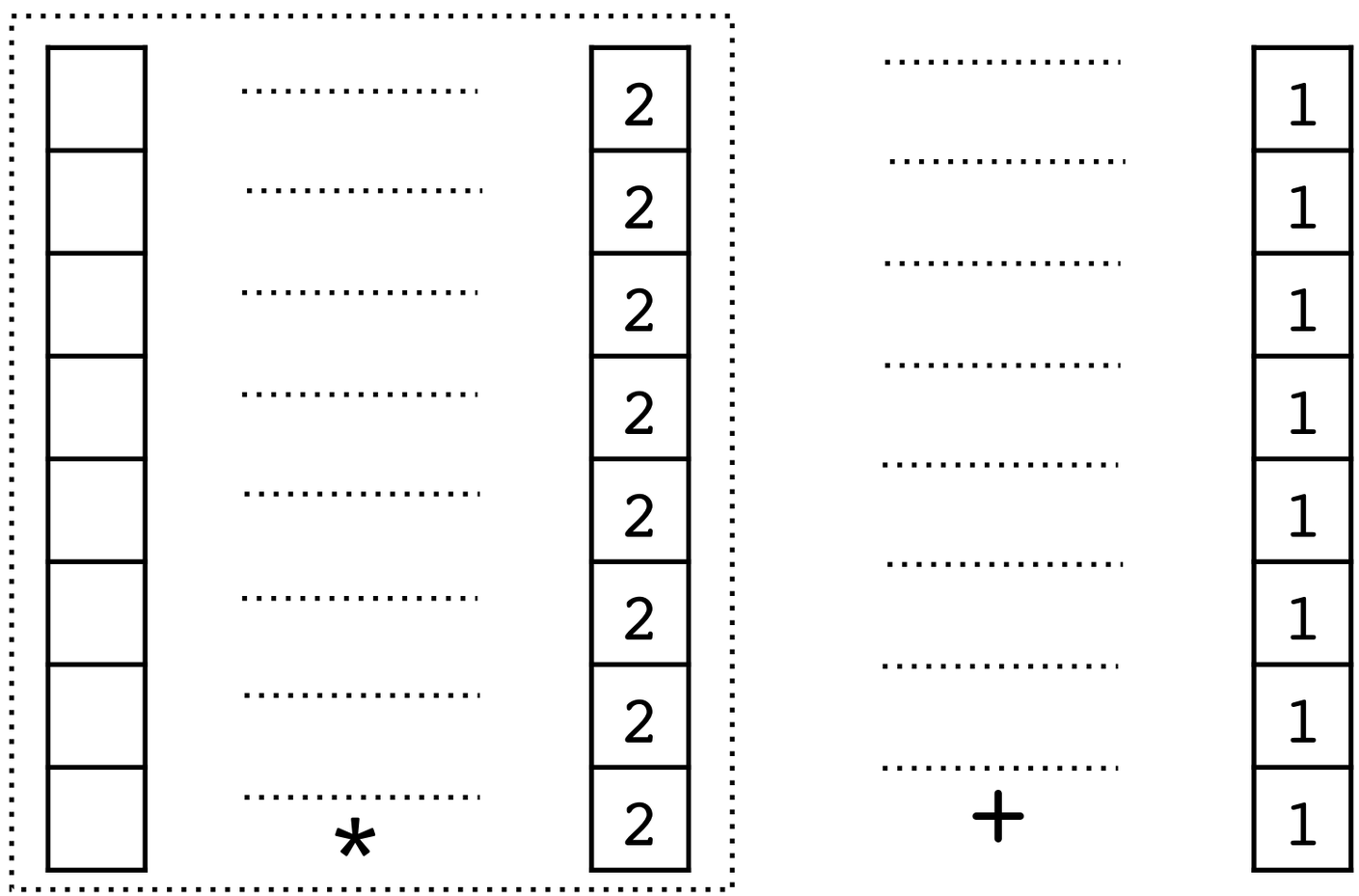


Vectorization with constants

```
f x   = (x * 2) + 1
```

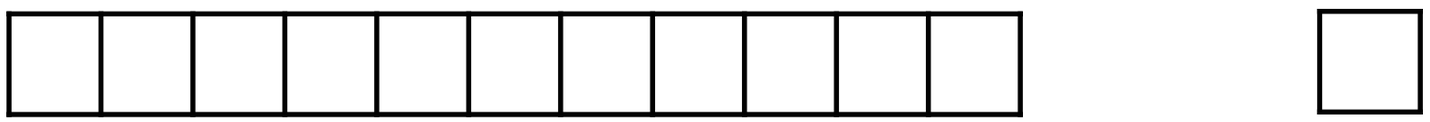


```
f^ c xs = (xs *^ rep c 2) +^ rep c 1
```

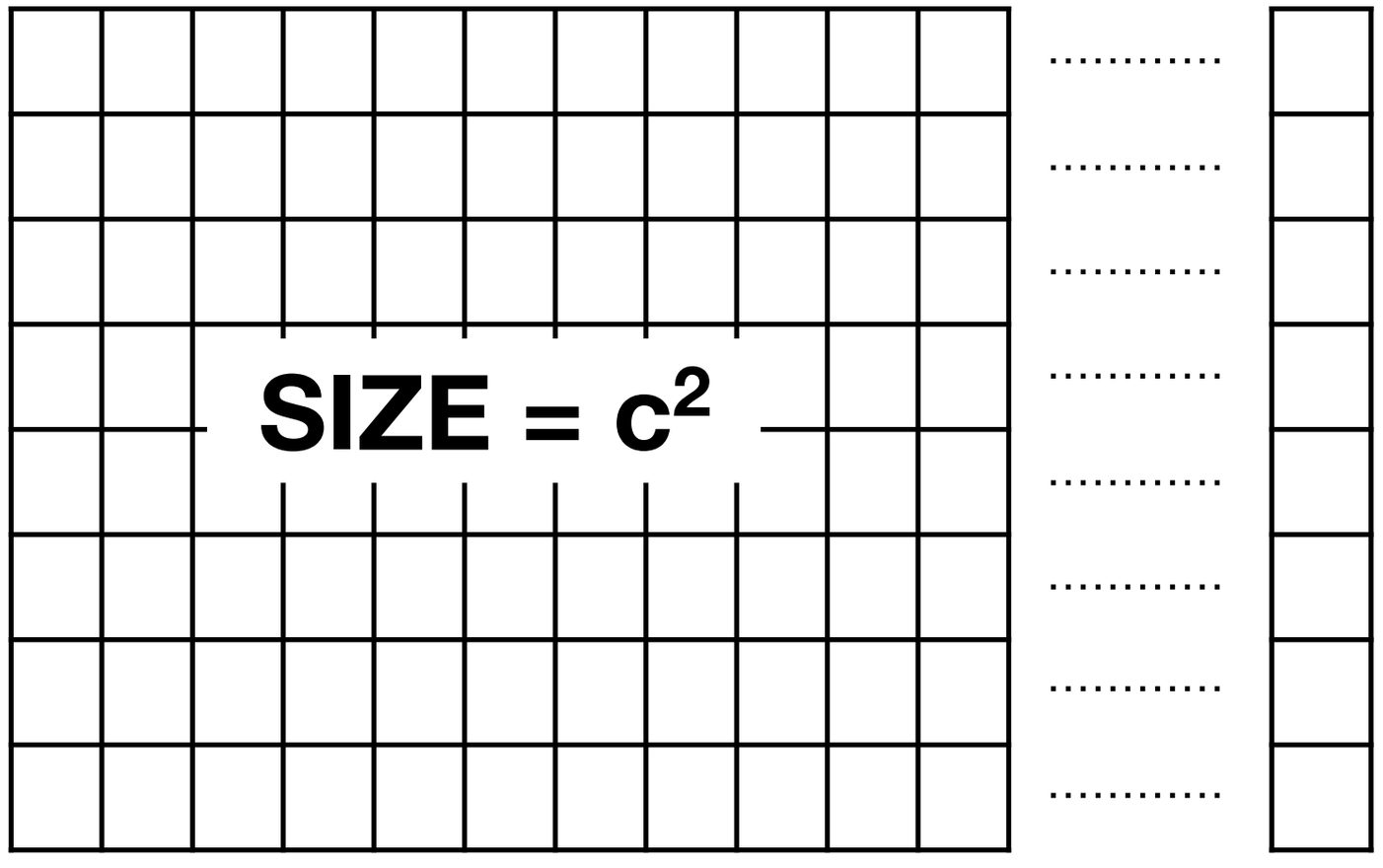


Vectorization with array constants

```
xs    = [...]
f i   = xs  !  i
```

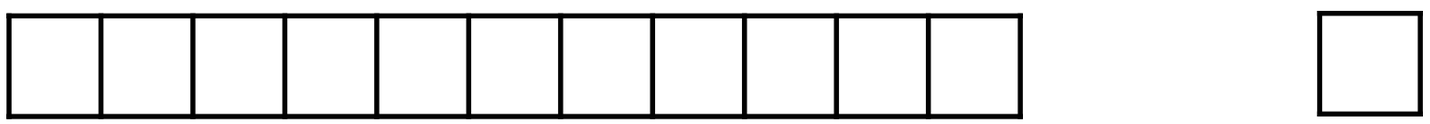


```
f^ c is = rep c xs !^ is
```

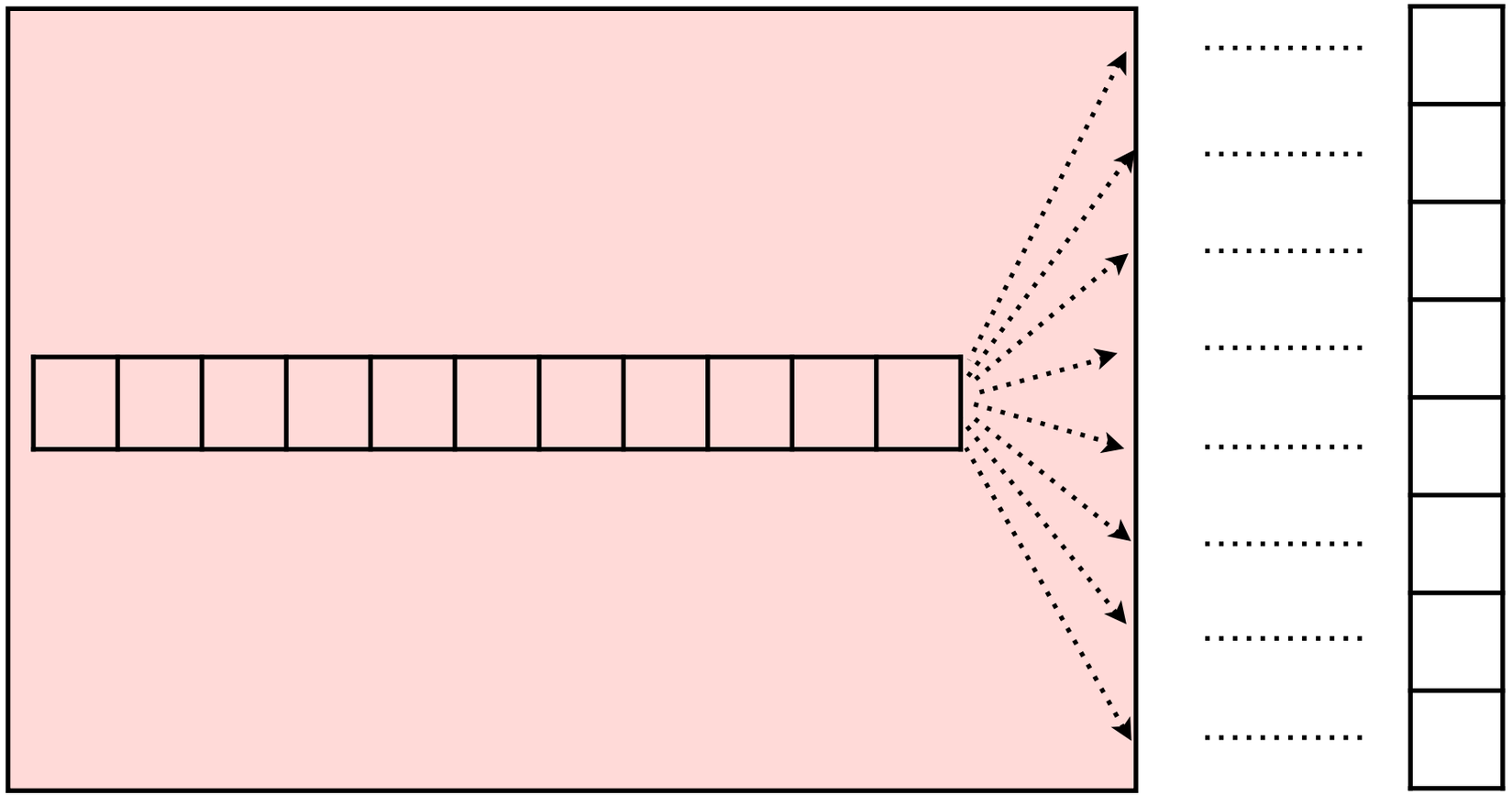


Store a single physical copy

```
xs    = [...]
f i   = xs  !  i
```



```
f^ c is = rep c xs !^ is
```



Array types

```
xs :: [Int]
xs = [...]

f ::    Int  -> [Int] -> [Int]
f i   = xs  !  i
```

```
f^ ::  Int
   -> [Int] -> [[Int]] -> [[Int]]

f^ c is = rep c xs !^ is
```

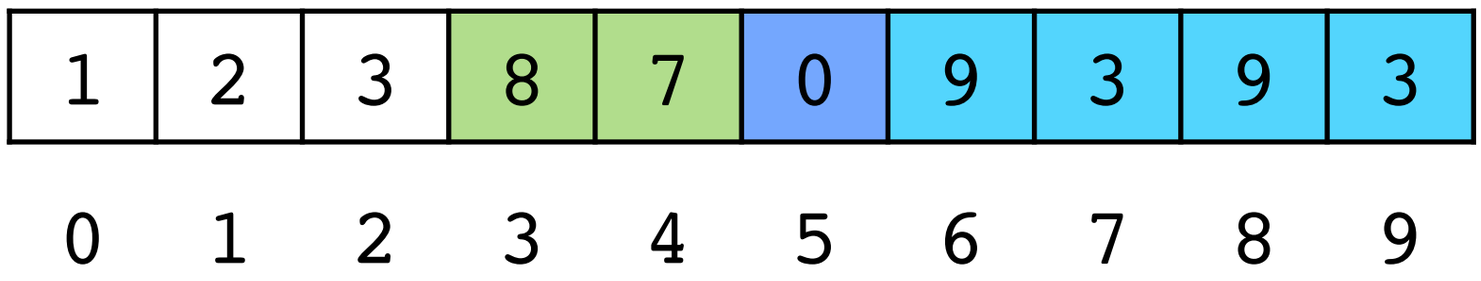
Nested array representation

```
data Array (Array a) = Nested Segd (Array a)

data Segd      = Segd (Array Int) (Array Int)
```

```
xs = Array (Array Int)
xs = [[1 2 3] [8 7] [0] [9 3 9 1]]
```

```
seg lens:    [ 3 2 1 4 ]
seg starts:  [ 0 3 5 6 ]
```

flat data:



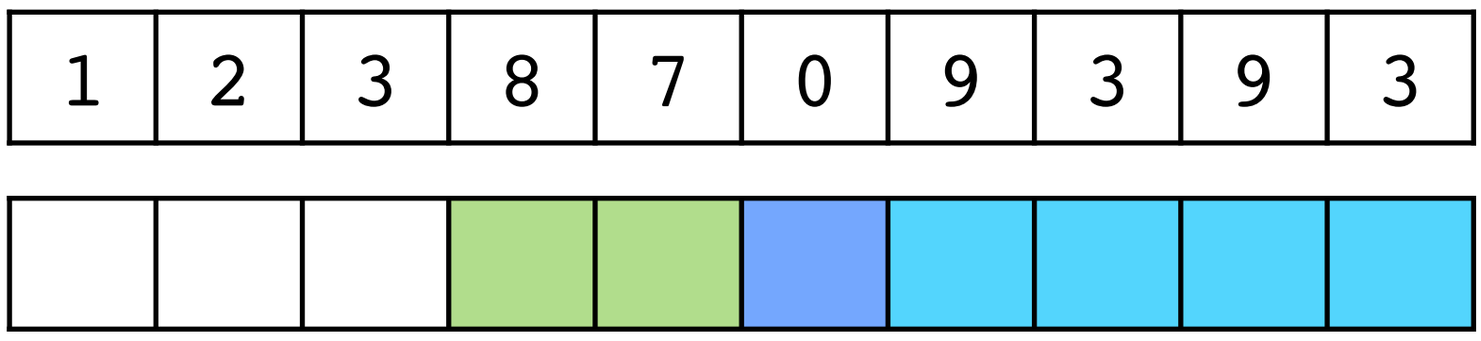
Nested array representation

```
data Array (Array a) = Nested Segd (Array a)

data Segd      = Segd (Array Int) (Array Int)
```

```
xs = Array (Array Int)
xs = [[1 2 3] [8 7] [0] [9 3 9 1]]
```

```
seg lens:    [ 3 2 1 4 ]

seg starts:  [ 0 3 5 6 ]
```

flat data:



Nested array representation

```
data Array (Array a) = Nested Segd (Array a)

data Segd            = Segd (Array Int) (Array Int)
```

```
xs = Array (Array Int)
xs = [[1 2 3] [8 7] [0] [9 3 9 1]]

seg lens:    [ 3 2 1 4 ]
seg starts:  [ 0 3 5 6 ]

flat data:     1  2  3  8  7  0  9  3  9  3
```

Nested array representation

```
data Array (Array a) = Nested Segd (Array a)

data Segd            = Segd (Array Int) (Array Int)
```

```
xs = Array (Array (Array Int))

xs = [[[1 2 3][8 7]] [] [[0][9 3 9 1]]]

seg lens:    [ 3 2 1 4 ]   seg lens:    [ 2 0 2 ]

seg starts:  [ 0 3 5 6 ]   seg starts:  [ 0 5 5 ]
```

flat data: 1 2 3 8 7 0 9 3 9 3

Replicated array representation

```
rep 3 [4 1 6] = [[4 1 6] [4 1 6] [4 1 6]]
```

```
seg lens:    [ 3 3 3 ]
seg starts:  [ 0 3 6 ]


flat data:   4   1   6   4   1   6   4   1   6
```

Replicated array representation

```
rep 3 [4 1 6] = [[4 1 6] [4 1 6] [4 1 6]]
```

```
seg lens:    [ 3 3 3 ]
seg starts:  [ 0 3 6 ]

rep count:   3
flat data:   4  1  6  4  1  6  4  1  6
```

Replicated array representation

```
rep 80000 [0..90000]
      = [[0..90000] [0..90000] ...]
```

virtual index overflow


```
seg lens:    [ 90000 ... 90000 ]
seg starts:  [ 0     ... 7,200,000,000]
```

rep count: 80000

flat data: 4 1 6 4 1 6 ... 4 1 6

Replicated array representation

```
rep 80000 [0..90000]
      = [[0..90000] [0..90000] ...]
```

```
seg lens:    [ 90000 ... 90000 ]
seg starts:  [ 0     ... 0 ]
```

```
flat data:  4  1  6
```

Replicated array representation

```
rep 80000 [0..90000]
      = [[0..90000] [0..90000] ...]
```

```
 rep count:   80000
 seg lens:    [ 90000 ]
 seg starts:  [ 0 ]
```

```
flat data:   4   1   6
```

Doubly lifted functions

```
zs       = [...]
f x      = x + 1
ys       = mapP f zs
```

```
f^ c xs = xs +^ 1
ys      = f^ (length zs) zs
```

Doubly lifted functions

```
zs       = [...]
f x      = x + 1
ys       = mapP f zs
```

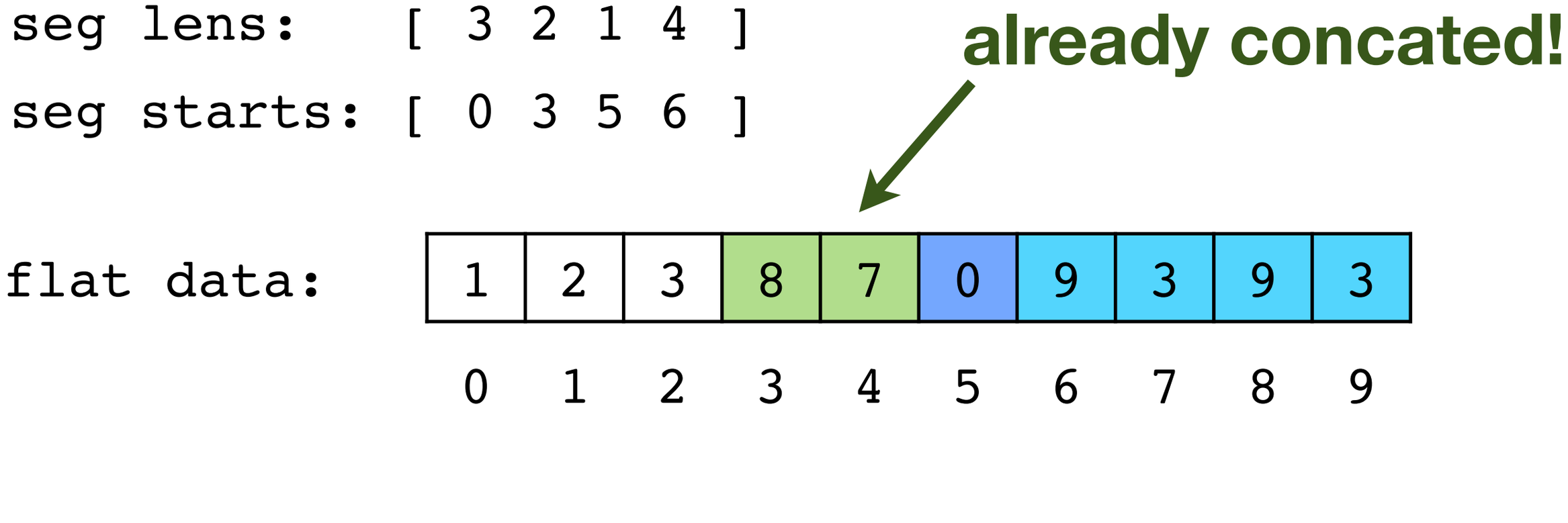
```
f^ c xs = xs +^ 1
ys      = f^ (length zs) zs
```

```
yss      = mapP (mapP f) xss
```

```
yss      = mapP f^ xss
yss      = f^^ xss     ???
yss      = unconcatP xss (f^ (concatP xss))
```

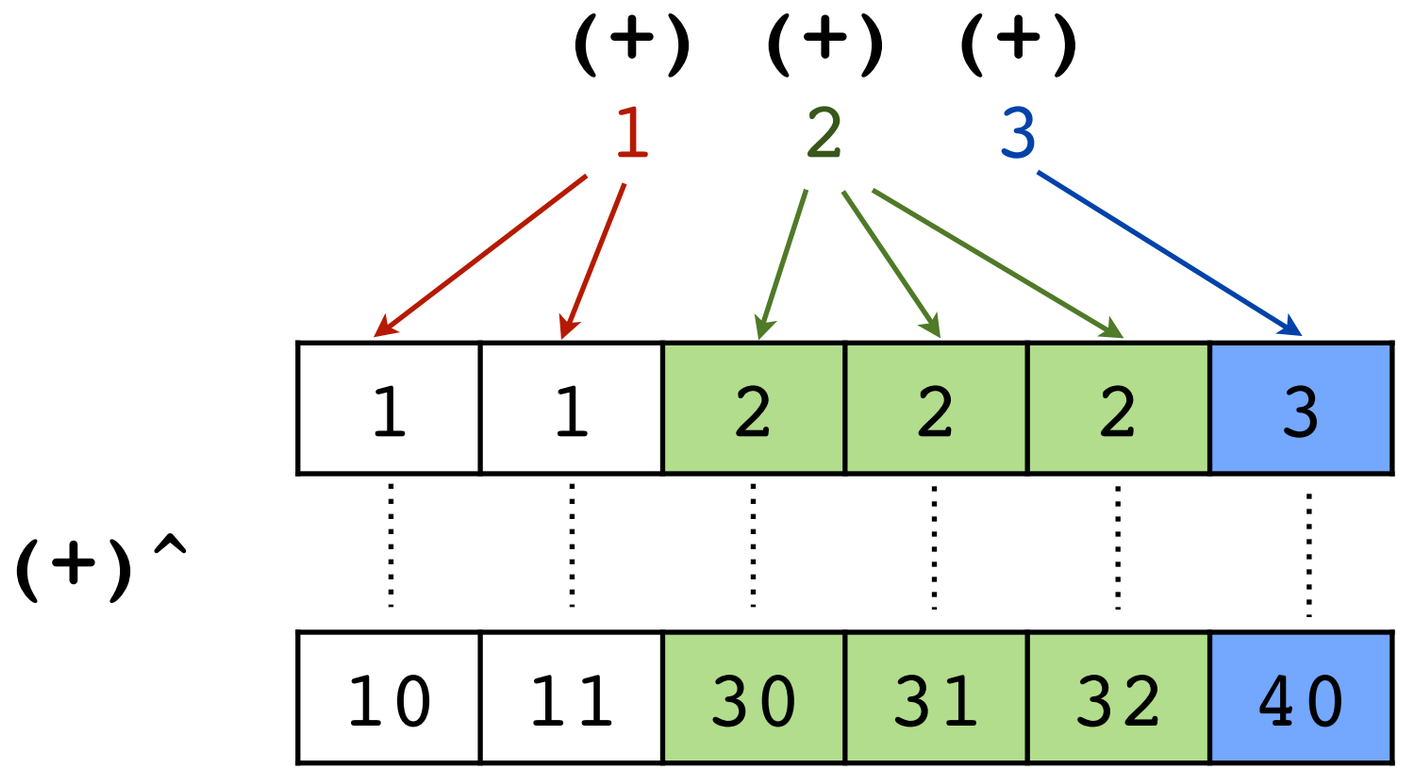
```
xs = Array (Array Int)
xs = [[1 2 3] [8 7] [0] [9 3 9 1]]
```

```
seg lens:    [ 3 2 1 4 ]
seg starts:  [ 0 3 5 6 ]
```

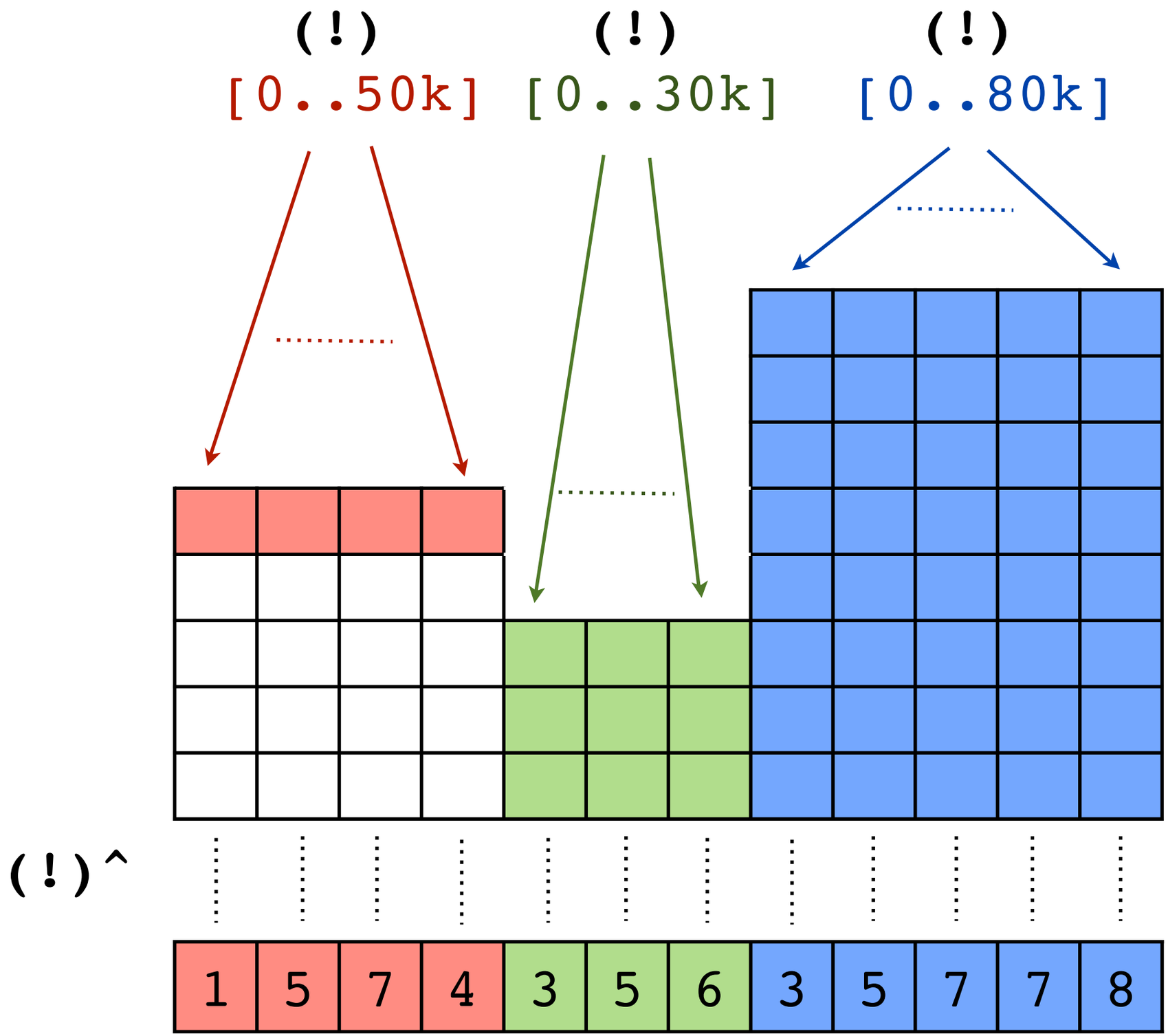


Functions with closures

```
zipWith map (map (+) [1 2 3])
            [[10 11] [30 31 32] [40]]

          = [[11 12] [32 33 34] [43]]
```



Lifted indexing



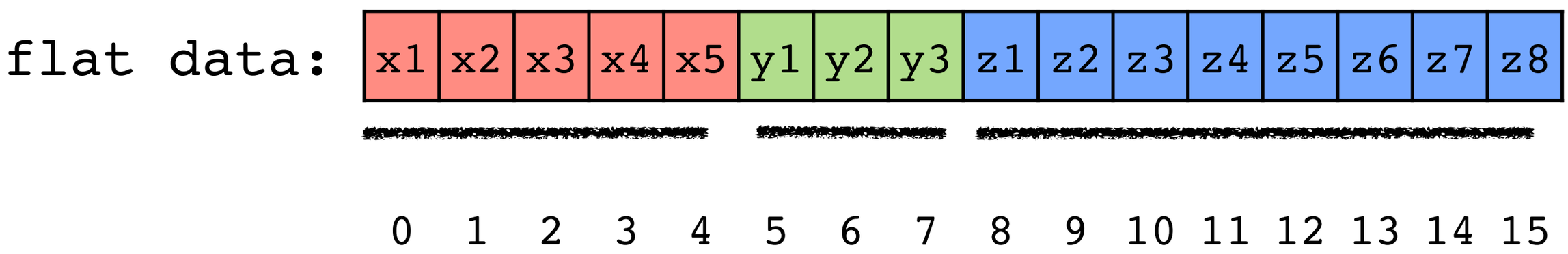
Segmented Replicate

```
reps :: [Int] -> [a] -> [a]

reps [2 3 1] [xs ys zs]
    = [xs xs ys ys ys zs]
```

rep counts: [2 3 1]

seg lens: [5 3 8]

seg starts: [0 5 8]

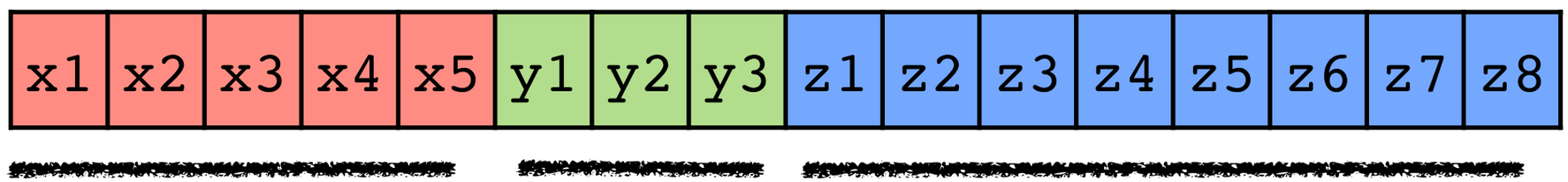


Segmented Replicate

```
reps :: [Int] -> [a] -> [a]

reps [2 3 1] [xs ys zs]
    = [xs xs ys ys ys zs]
```

```
virt seg ids:     [0 0 1 1 1 2]
phys seg lens:    [5 3 8]
phys seg starts:  [0 5 8]
```

flat data:

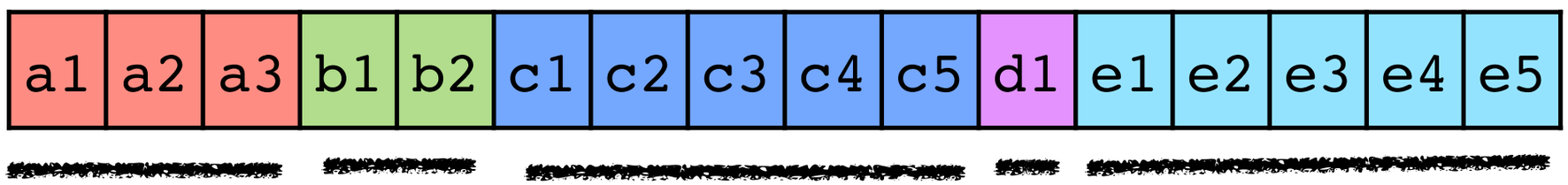


Pack

```
pack :: [Bool] -> [a] -> [a]
pack [T F T T F] [as bs cs ds es] = [as cs ds]
```

```
virt seg ids:     [0   1   2   3   4]
phys seg lens:    [3   2   5   1   5]
phys seg starts: [0   3   5   10  11]
```

flat data:

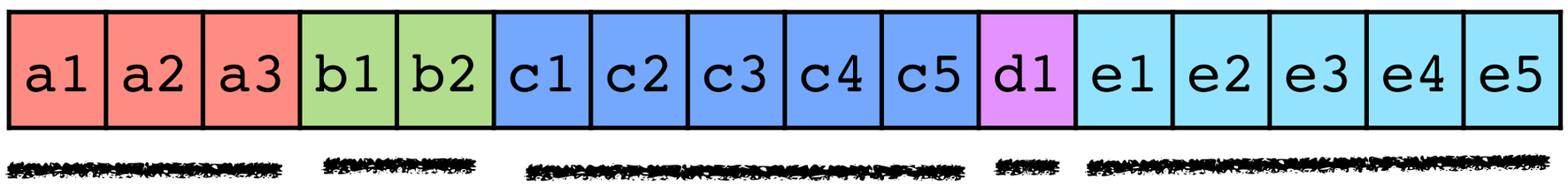


Pack

```
pack :: [Bool] -> [a] -> [a]
pack [T F T T F] [as bs cs ds es] = [as cs ds]
```

```
                    [T   F   T   T   F]
virt seg ids:       [0   1   2   3   4]
phys seg lens:      [3   2   5   1   5]
phys seg starts:    [0   3   5   10  11]
```

flat data:

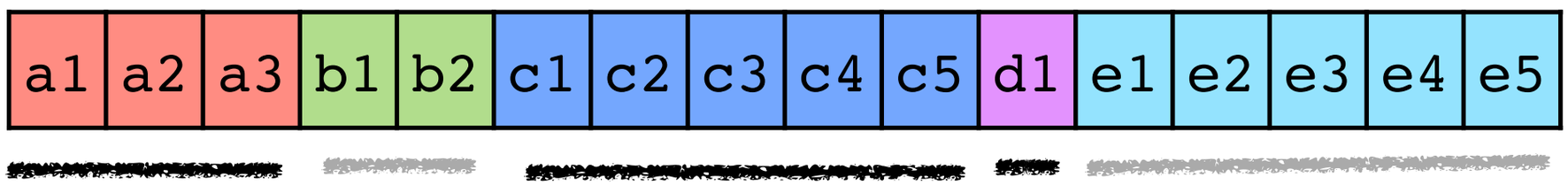


Pack

```
pack :: [Bool] -> [a] -> [a]
pack [T F T T F] [as bs cs ds es] = [as cs ds]
```

```
                    [T   T   T]
virt seg ids:       [0   2   3]
phys seg lens:      [3   2   5   1   5]
phys seg starts:    [0   3   5   10  11]
```

flat data:

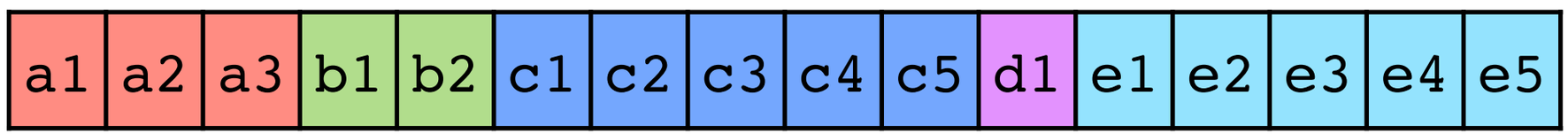


Pack

```
pack :: [Bool] -> [a] -> [a]
pack [T F T T F] [as bs cs ds es] = [as cs ds]
```

```
                    [T   T   T]
virt seg ids:       [0   1   2]
phys seg lens:      [3   5   1]
phys seg starts:    [0   5   10]
```

flat data:



Operators

```
rep       ::  Int    ->  a      -> [a]
reps      :: [Int]   -> [a]     -> [a]

concat    :: [[a]]   -> [a]
unconcat  :: [[a]]   -> [a]     -> [[a]]

pack      :: [Bool] -> [a]    -> [a]
combine2  :: [Bool] -> [a]    -> [a] -> [a]

index     :: Int     -> [a]    -> a
indexl    :: [Int]   -> [[a]] -> [a]

append    :: [a]     -> [a]    -> [a]
appendl   :: [[a]]   -> [[a]] -> [[a]]
```

Functions with closures

```
map^ :: Int -> [a -> b] -> [[a]] -> [[b]]
map^ c (AClo f f^ envs) xss
 = unconcat xss
     (fl (replicates (map length xss) envs)
         (concat xss))
```

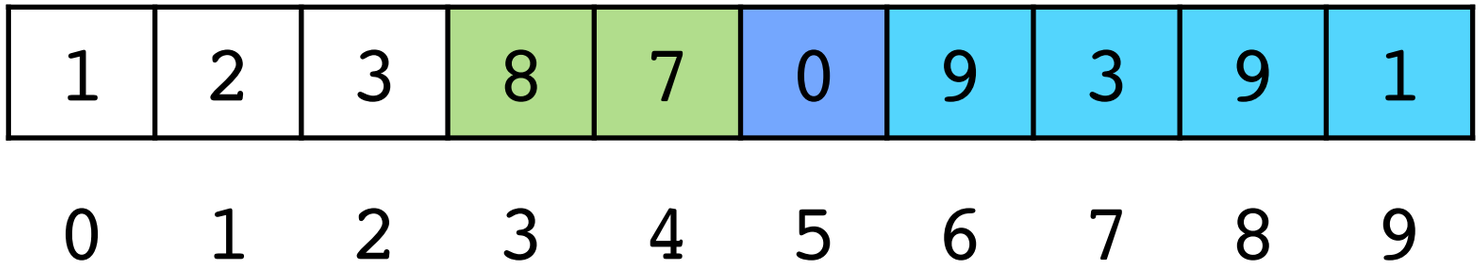
Nested Arrays

```
xs = [[1 2 3] [8 7] [0] [9 3 9 1]]
```

Nested Arrays

```
xs = [[1 2 3] [8 7] [0] [9 3 9 1]]

seg lens:  [ 3 2 1 4 ]

seg idxs:  [ 0 3 5 6 ]
```

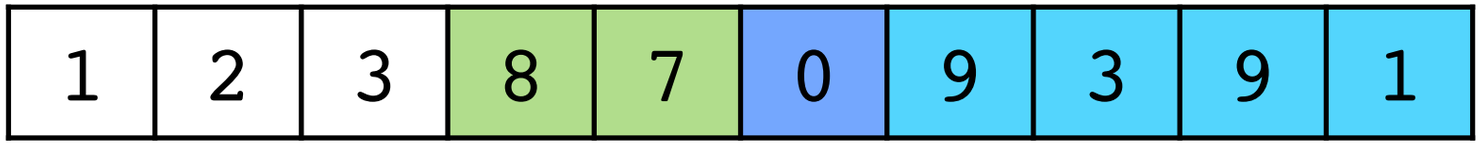
flat data:



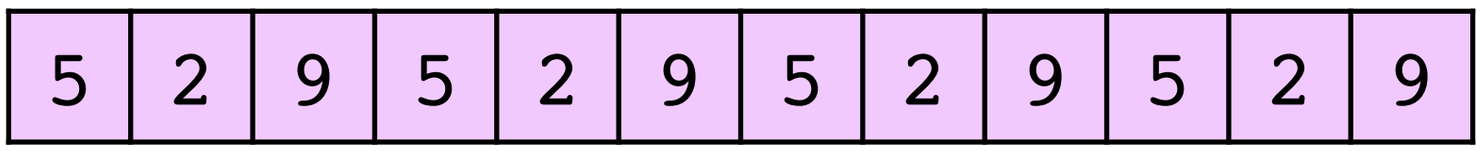
Nested Arrays

```
xs = [[1 2 3] [8 7] [0] [9 3 9 1]]

seg lens:  [ 3 2 1 4 ]

seg idxs:  [ 0 3 5 6 ]
```

flat data:



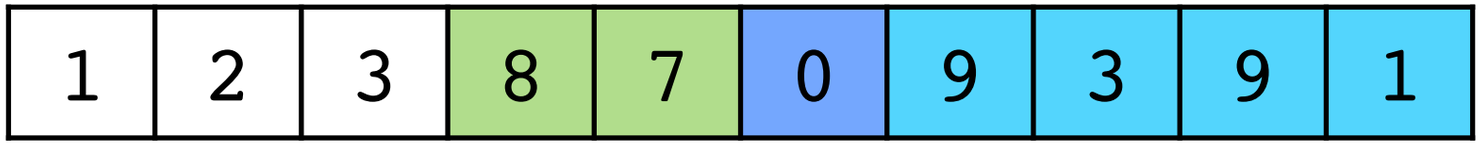

```
replicate 4 [5 2 9]
   = [[5 2 9] [5 2 9] [5 2 9] [5 2 9]]
```



Nested Arrays

```
xs = [[1 2 3] [8 7] [0] [9 3 9 1]]

seg lens:  [ 3 2 1 4 ]

seg idxs:  [ 0 3 5 6 ]
```

flat data:

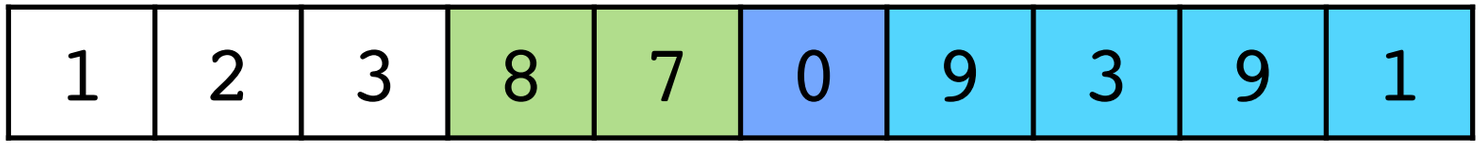



```
reps [1 3 2 1]
     [[1 2 3][8 7][0][9 3 9 1]]

   = [[1 2 3][8 7][8 7][8 7][0][0][9 3 9 1]]
```

Nested Arrays

```
      xs = [[1 2 3] [8 7] [0] [9 3 9 1]]

 seg lens:   [ 3 2 1 4 ]

 seg idxs:   [ 0 3 5 6 ]
```

flat data:




```
pack [T F T F]
      [[1 2 3] [8 7] [0] [9 3 9 1]]

   = [[1 2 3] [0]]
```

UVSegd

reps: [1 3 2 1]

vsegs: [0 1 1 1 2 2 3]

[[1 2 3][8 7][8 7][8 7][0][0][9 3 9 1]]

USegd

lens: [3 2 1 4]

idxs: [0 3 5 6]

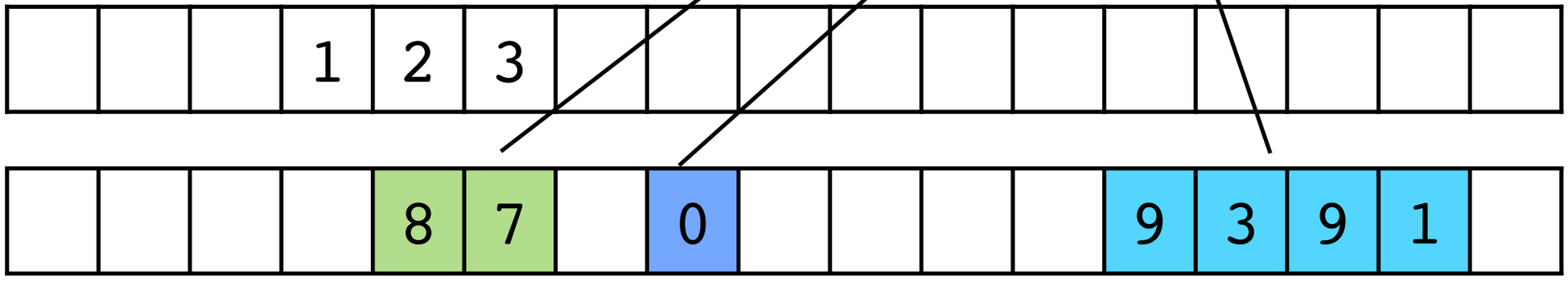
[[1 2 3][8 7][0][9 3 9 1]]

USSegd

srcs: [0 1 1 1]

starts: [3 4 7 12]

[# # # #]



Questions?