Union Find を永続にしたい

https://github.com/fumieval/persistent-union-find

Union Find

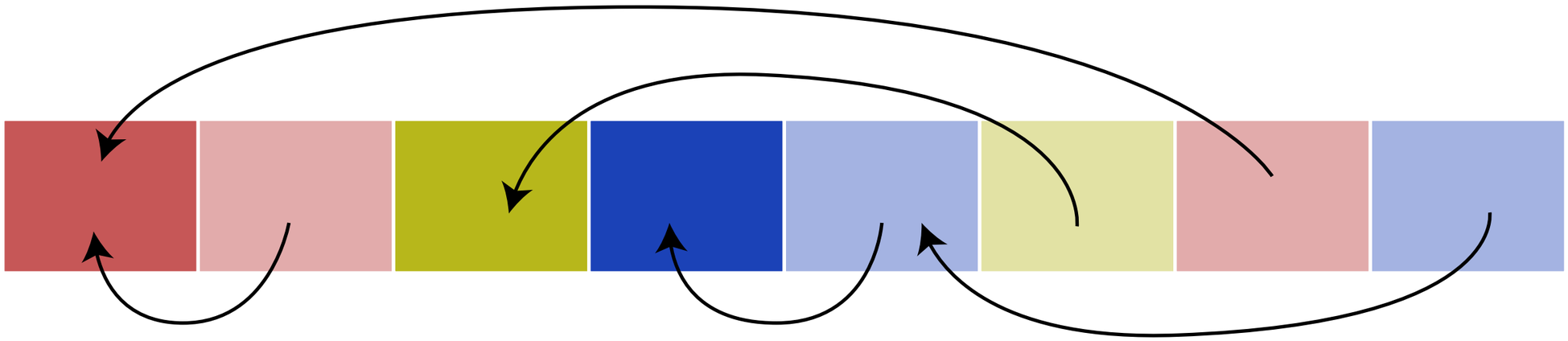
要素が属する集合を返す関数 find、

集合同士を統合する操作 union ができる構造

無向グラフの二頂点間に路が存在するかの判定に使える

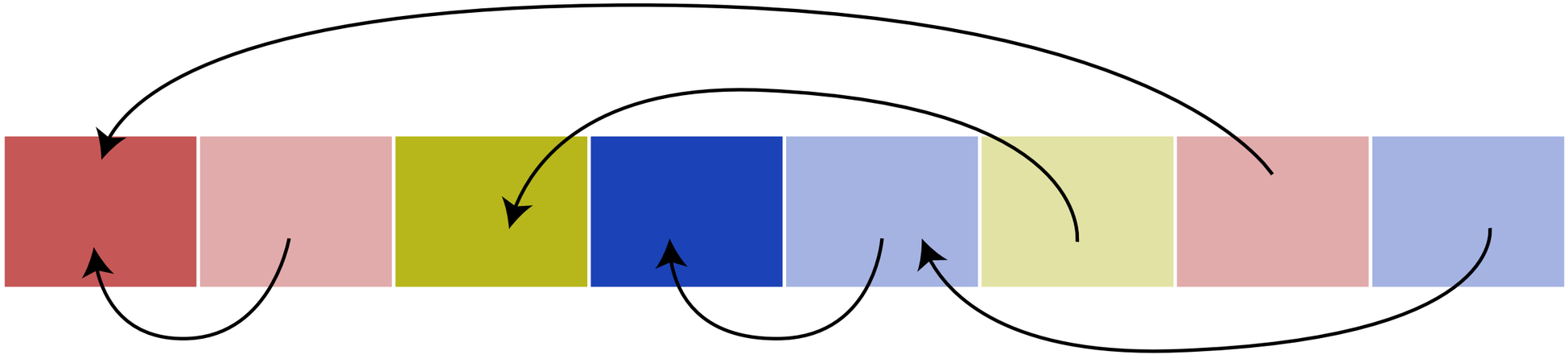
```
union p q
union q r
assert (find p == find r)
```

配列を用いた実装

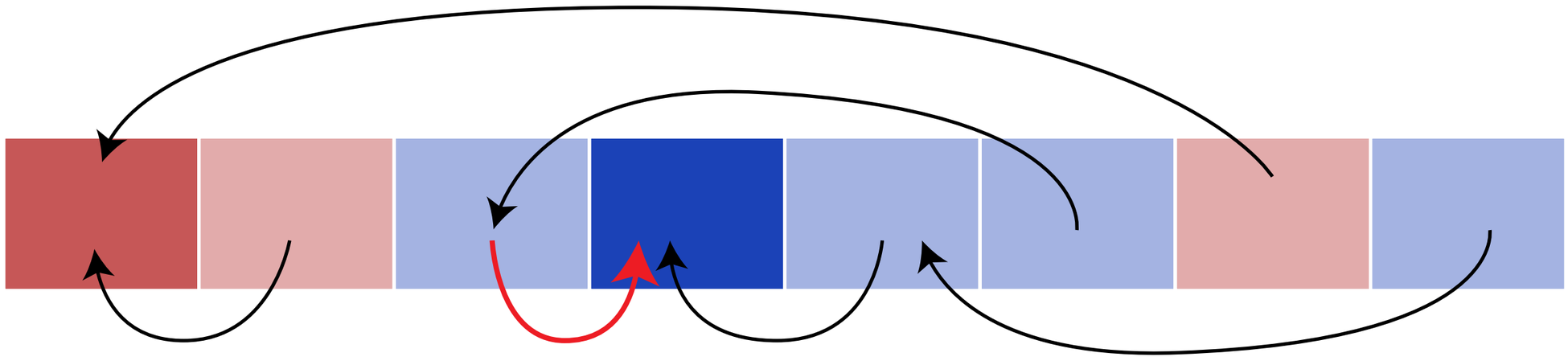


「自分自身のインデックスの配列」による木構造として表されることが多い

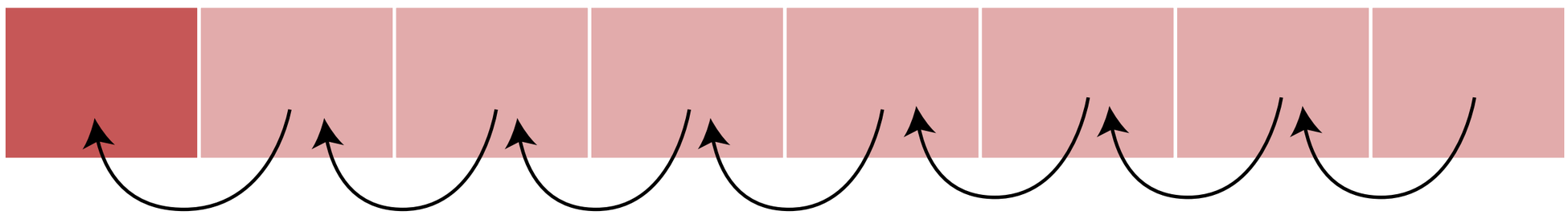
find: インデックスが指し示す要素がそれ自身になるまで辿ること



元の状態

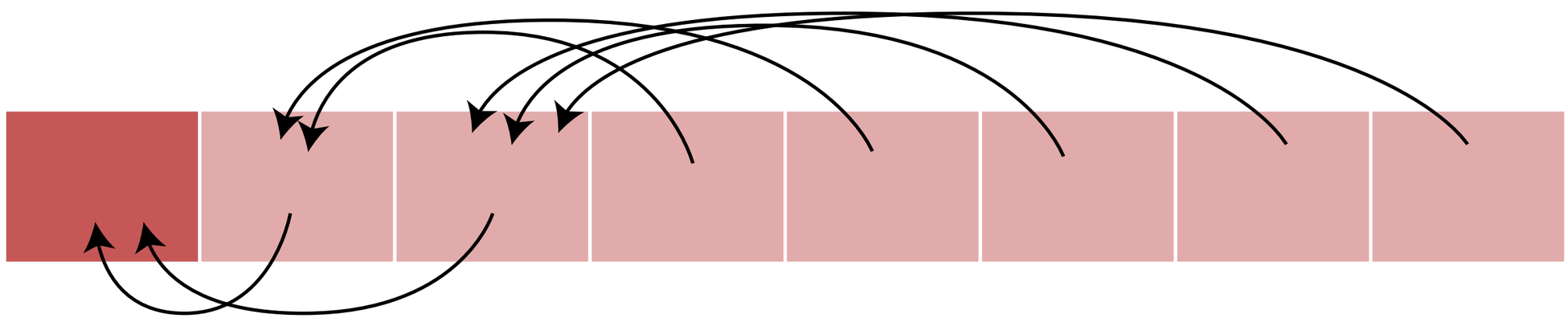


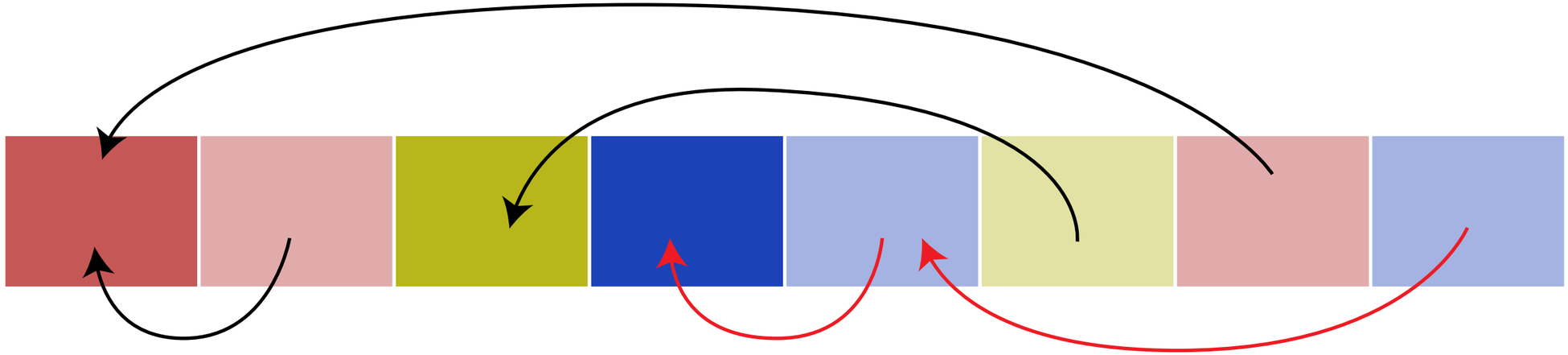
union（■, ■）した状態



find の計算量を抑えるため、上よりは下のようにしたい。

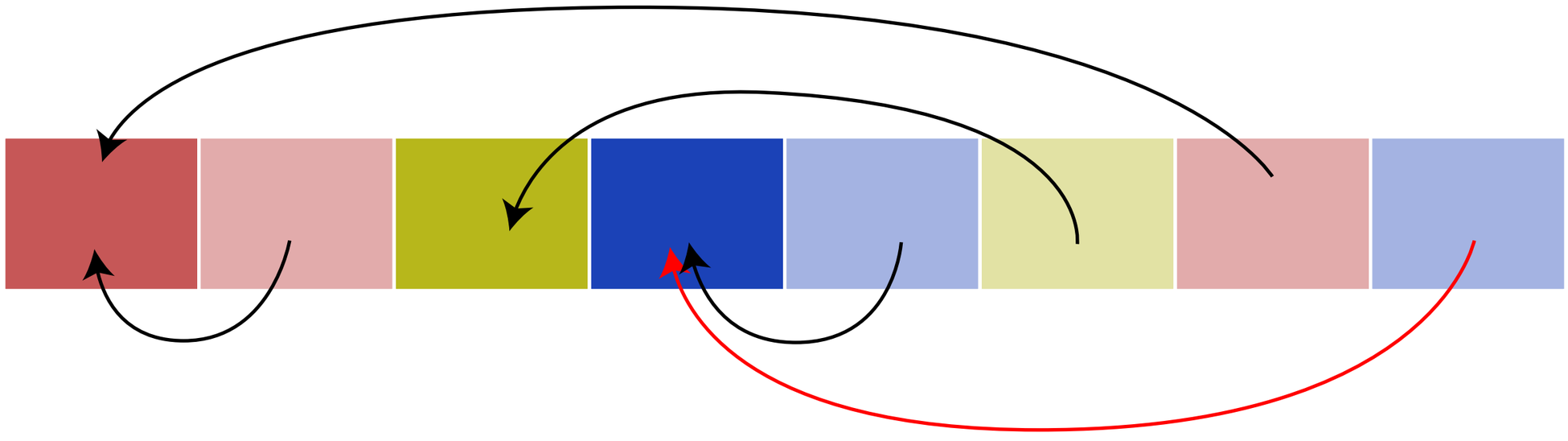
各要素に深さ（しばしば rank と呼ばれる）を割り当て、
rank が高いほうを親にすることで最適化できる





一度辿ったところを書き換えることで

次のアクセスを高速化する



大問題

典型的な Union-Find の表現は

ポインタを書き換える操作ありきであり、

ミュータブルな実装と非常に相性がよく、

イミュータブルな実装と相性が悪い

アプローチ　甲

IntMap などを配列の代わりにする

https://hackage.haskell.org/package/union-find

Data.UnionFind.IntMap で採用された方法

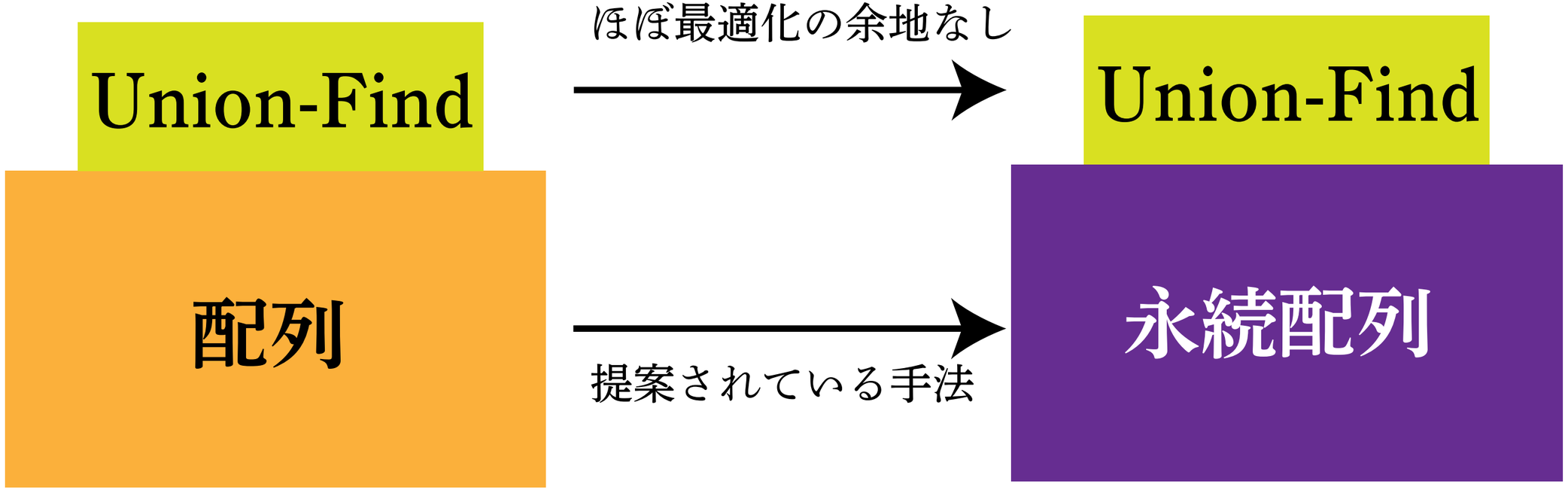
アプローチ　乙

破壊的変更を活用した永続配列を作る

Sylvain Conchon, Jean-Christophe Filliâtre

A persistent union-find data structure

https://www.lri.fr/~filliatr/ftp/publis/puf-wml07.pdf


Union-Find
配列
ほぼ最適化の余地なし
提案されている手法
Union-Find
永続配列

永続配列 の作り方

元の配列と、そこからの差分を表現する

```
newtype PArray a
      = PArray (IORef (PArrayData a))


data PArrayData a
  = Base (MV.IOVector a)
  | Diff !Int !a (PArray a)
```

永続配列の作り方

差分を辿っていくことで読み取りを行う

```
read :: PArray a -> Int -> IO a
read (PArray ref) i = readIORef ref >>= \case
  Base vec -> MV.read vec i
  Diff j v inner
    | i == j -> pure v
    | otherwise -> read inner i
```

永続配列 の作り方

```
set :: Int → a → Array a → Array a
set i v array =
```

インデックス i に v をセットした配列を作り、

array をその差分で置き換える ←── ！ ？

x = [0,1,2,3,4]

y = set 1 10 x

z = set 2 20 y

[0,1,2,3,4]

書き換え

(1,1) :

[0,10,2,3,4]

書き換え

(1,1) : (2,2):

[0,10,20,3,4]

禁術 unsafePerformIO を使えば Haskell でも可能

```
set :: Int -> a -> PArray a -> PArray a
set i v ref = unsafePerformIO $ do
  vec <- reroot ref                    生配列を取得する
  old <- MV.read vec i                 古い値を取得する
  MV.write vec i v                     新しい値を書き込む
  ref' <- newIORef (Base vec)
  writeIORef (unPArray ref)
    $ Diff i old (PArray ref')         差分を書き込む
  pure (PArray ref')
```

```
reroot :: PArray a -> IO (MV.IOVector a)
reroot ref = readIORef (unPArray ref) >>= \case
  Base vec -> pure vec
  Diff i v inner -> do
    vec <- reroot inner
    old <- MV.read vec i
    MV.write vec i v
    writeIORef (unPArray ref) (Base vec)
    writeIORef (unPArray inner) (Diff i old ref)
    pure vec
```