"collect"

list (key, value) pairs unsorted

goal: combine values by equal key

e.g. values: $\mathbb{R}$

combine with addition

$(1, 42.0)$ $(10, 1.0)$ $(1, 0.5)$ $(11, 0.0)$

⇓ sort

$\rightarrow$ $(1, 42.0)$ $(1, 0.5)$ $(10, 1.0)$ $(11, 0.0)$

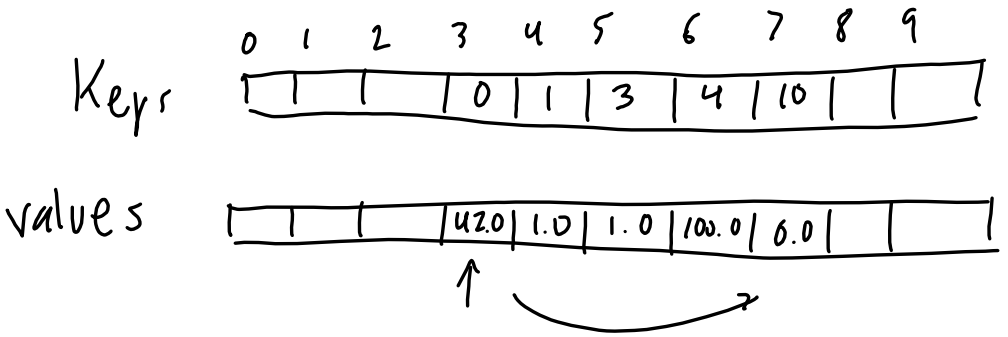
start index: 0 2 3

↑ number of unique keys

⇓

$(1, 42.5)$ $(10, 1.0)$ $(11, 0.0)$

"open addressing" "linear probing"



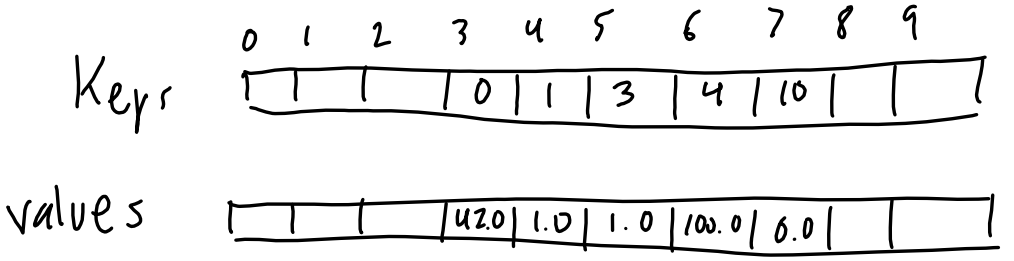
insert 3, 1.0 hash(3) = 5

insert 0, 42.0 hash(0) = 3

insert 4, 100.0 hash(4) = 5

insert 1, 1.0 hash(1) = 4

insert 10, 0.0 hash(10) = 3



not safe for concurrency! (yet)

```
insert(K, V) =
  let
    fun loop(i) =
      if Keys[i] = EMPTY then
        (Keys[i] := K;
         values[i] := V)
      else Keys[i] = K then
          values[i] := combine(values[i], V)
      else
          loop((i+1) % |K|)
  in
    loop(hash(K))
```

atomically in hardware

```
compare_and_swap (A: elem array, i, old, new)
    let x = A[i]
    if   x  = old then
       A[i] := new;
       x
    else
       x
```

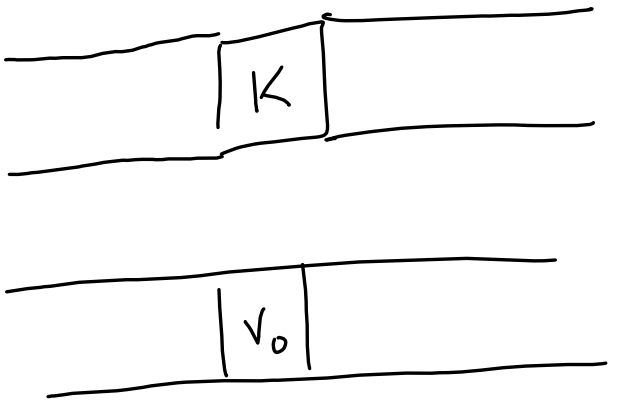
```
type lock = bool array (length 1)

fun lock(L) =                                   "spin lock"
   if L[0] then lock(L)
   else
   if false = compare_and_swap(L,0, false, true)
   then () ✓
   else lock(L)                                 "lock freedom"

fun unlock(L) =
   L(0) := false
```

```
insert(K,v) =
  let
    fun loop(i) =
      if Keys[i] = EMPTY then
        (Keys[i] := K ;
         values[i] := v)
      else Keys[i] = K then
        values[i] := combine(values[i], v)
      else
        loop((i+1) % |K|)
  in
    loop(hash(K))
```

INVARIANT:
if Keys[i] is EMPTY
then Values[i] = 0

```
fun insert(K,v) =
  let fun loop(i) =
      let curr = Keys[i]
      if curr ≠ EMPTY then
        if curr = K then
          update value
          atomic_combine(values, i, v)
        else
          loop((i+1) % |Keys|)
      else
        if EMPTY = compare_and_swp(Keys, i, EMPTY, K)
          insert value
          atomic_combine(values, i, v)
        else
          loop(i)
  in
    loop(hash(K))
```

"LINEARIZABILITY"



insert(K, v_1) || insert(K, v_2)

old = values[i]
new = combine(old, v_1)
values[i] := new

old = values[i]
new = combine(old, v_2)
values[i] := new

values[i] :=
combine(values[i], v_1)

values[i] :=
combine(values[i], v_2)

```
fun atomic_combine (A, i, v)
    let old = A[i]
        new = combine (old, v)
    if old = Compare_and_swap (A, i, old, new)
    then ()
    else atomic_combine (A, i, v)
```

LL/SC "load linked store conditional"

multi-word compare and swap

double-word compare and swap

STM HTM