"Collect"

similar to Map Reduce

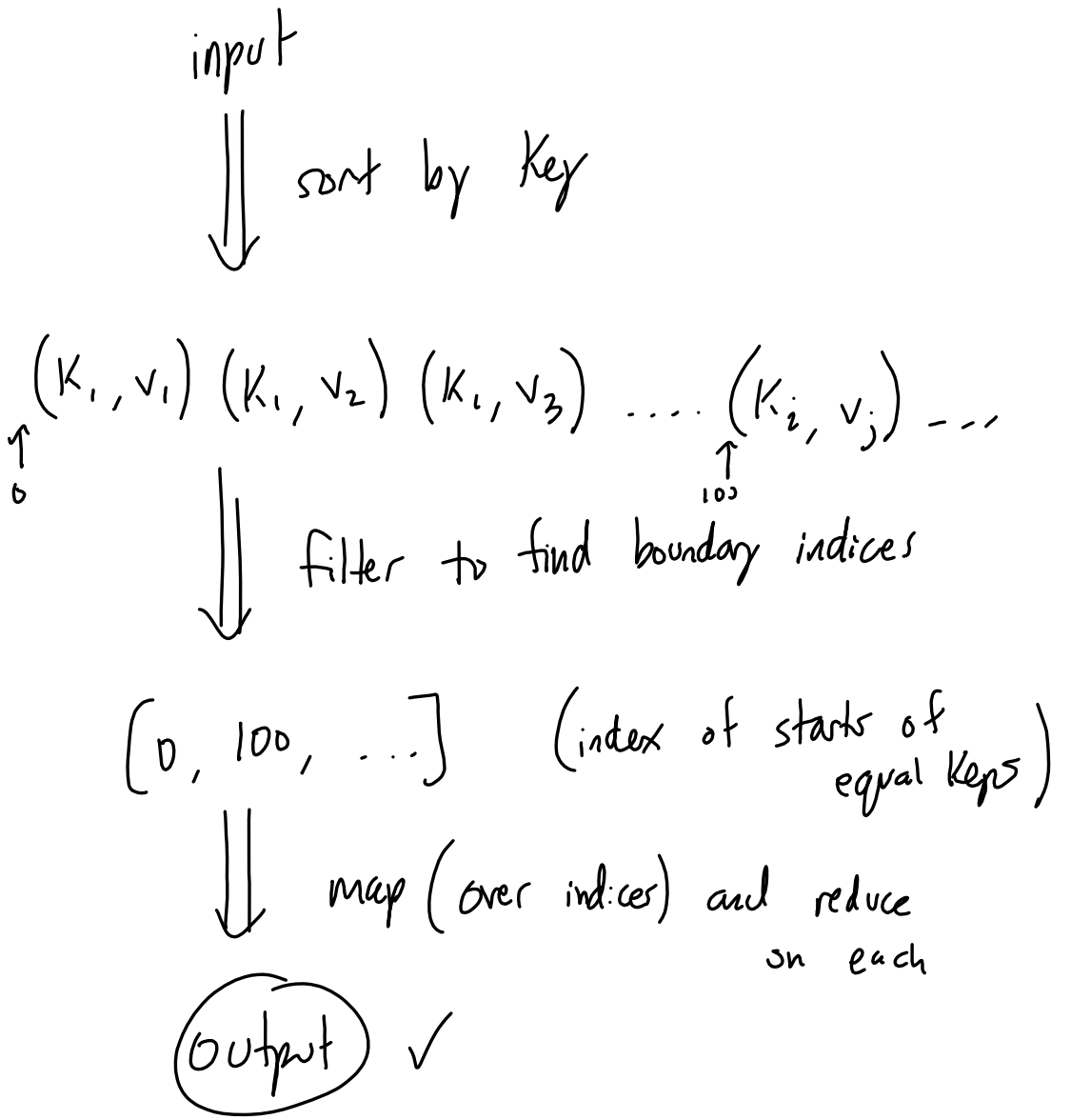
input: seq of (key, value) pairs
function to "aggregate" values

output: seq of (key, value) pairs:
unique keys
aggregated values per unique key
in order of sorted keys

example (adding values)

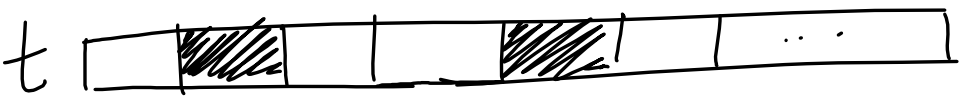
(1, 42.0) (10, 1.0) (1, 0.5) (11, 0.5)

(1, 42.5) (10, 1.0) (11, 0.5)


input
sort by key
(K_1, V_1) (K_1, V_2) (K_1, V_3) .... (K_i, V_j) ...
0
100
filter to find boundary indices
[0, 100, ...]
(index of starts of equal keys)
map (over indices) and reduce on each
output ✓

Open addressing

"hash set"



sequence of keys

"empty key"

hash (K) → int

```
insert (t, K) =
  let
    fun loop (i) =
      if t[i] = empty then
        (t[i] := K; ())
      else if t[i] = K then
        ()
      else
        loop ((i+1) mod |t|)
  in
    loop (hash (K) mod |t|)
  end
```

LINEAR PROBING

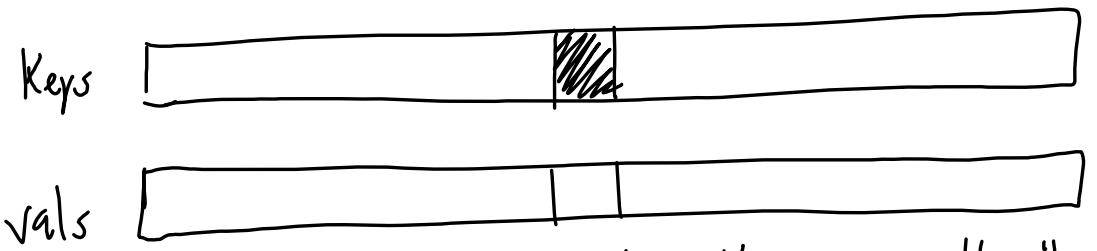
```
insert (t, K) =
  let
    fun loop(i) =
      if t[i] = empty then
        (t[i] := K; ())
      else if t[i] = K then
        ()
      else
        loop((i+1) mod |t|)
  in
    loop(hash(K) mod |t|)
  end
```

```
if CAS(t, i, empty, K)
then ()
else
  if t[i] = K then
    ()
  else
    loop(i+1 ...)
```

hash(K_1) = 1	hash(K_2) = 1
if t[1] = empty	if t[1] = empty
t[1] := K_1	t[1] := K_2

```
CAS(arr, i, old, new) =
  if arr[i] = old then (arr[i] := new; true)
  else false
```

ATOMICALLY IN HARDWARE



init: Keys are all "empty"
vals are all "zero"

```
insert (keys, vals) (K, v) =
  let
    fun loop(i) =
      if CAS(keys, i, empty, K) then
        atomic_combine(vals, i, v)
      else if keys[i] = K then
        atomic_combine(vals, i, v)
      else
        loop(i+1 mod ...)

  in
    loop(hash(K) mod |t|)
  end
```

combine : $V \times V \to V$

```
atomic_combine (vals, i, v) =
  let
    t = vals[i]
    t' = combine (t, v)
  in
    if CAS (vals, i, t, t') then
      ()
    else
      atomic_combine (vals, i, v)
  end
```

"CAS LOOP"

"READ-MODIFY-WRITE LOOP"

"RMW LOOP"