var-length
unordered

map : [key] ⟶ value

⇓ (hash code)

hash table [int (small) range] ⟶ value
array — bucket array

* $h(key) = compress(HashCode(key))$

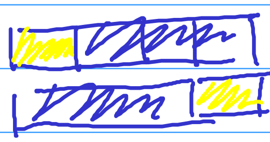
compress ↓ division (% K)

HashCode:

① Σ key[i] — sum — "abc", "acb"

② Σ (key[i] << i) %2^{32} — shift — a polynomial hash code

③ cyclic shift



* collision : $h(key_1) = h(key_2)$ but $key_1 \neq key_2$

handling

⓪ don't care < throw away / don't insert

(tolerate some errors)

bloom filter

hash-based map : set = { k_1, k_2, ⋯ , k_p }

key → { true, false }

bit bucket array

→ [0 0 1 0 0 1 0 0 1 0 0 0 1]

$a[h(k_1)] = 1$
$a[h(k_2)] = 1$
$a[h(k_p)] = 1$

$a[h(k_i)]$ < 1 — ? — correct if no collision / ? if collision
0 — no error

① fixed array per bucket (can still overflow)

② linked list per bucket (chaining) (hope short chain)

③ other container : secondary

④ "reuse" empty space

open addressing

insert

a[key] = value (abstract)

→ ① insert (key, value) to array location $h_0(key) = h(key)$ check

② if fail, re-insert to $h_1(key)$ check

③ if fail, re-insert to $h_2(key)$ check

⋮

(m) if fail, re-insert to $h_{m-1}(key)$ check

(m+1) declare failure (not found)

x = a[key] get

Ⓐ linear probing

$h_i = (h_{i-1} + 1) \% K$

$= (h_0 + i) \% K$

m at most K-1

clustering

Ⓑ quadratic probing

$h_i = (h_0 + i^2) \% K$

Ⓒ double hashing

$h_i = (h_0 + i \cdot \text{another hash } \hat{h}(key)) \% K$

hashcode ↓ array ↓

* hash table K entries

n keys

want $\underbrace{\frac{n}{K}}_{\text{load factor}}$ small

$\frac{n}{K}$ large → hash won't work

($K > n$) ↑

hash non-uniform → $\frac{n}{K_{eff}}$ large

* $\frac{n}{K}$ ↑ too big than K ↑

* naive

if $\frac{n}{K} > \theta$

① set $K^{new} = 2K$

② $h(key)$ change → range $\{0, 1, \cdots, 2K-1\}$

③ rebuild w/ $O(n)$ ✓ if each insert ≈ $O(1)$

full

long waiting

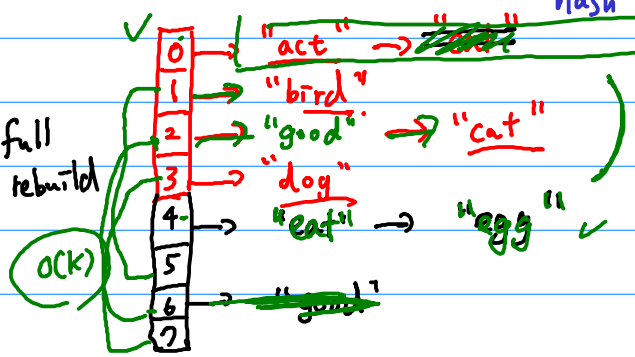
partial rebuild $O(K) + O(\frac{n}{K})$ ✓

$h(key) = (key[0] - 'a') \% K$

4

8

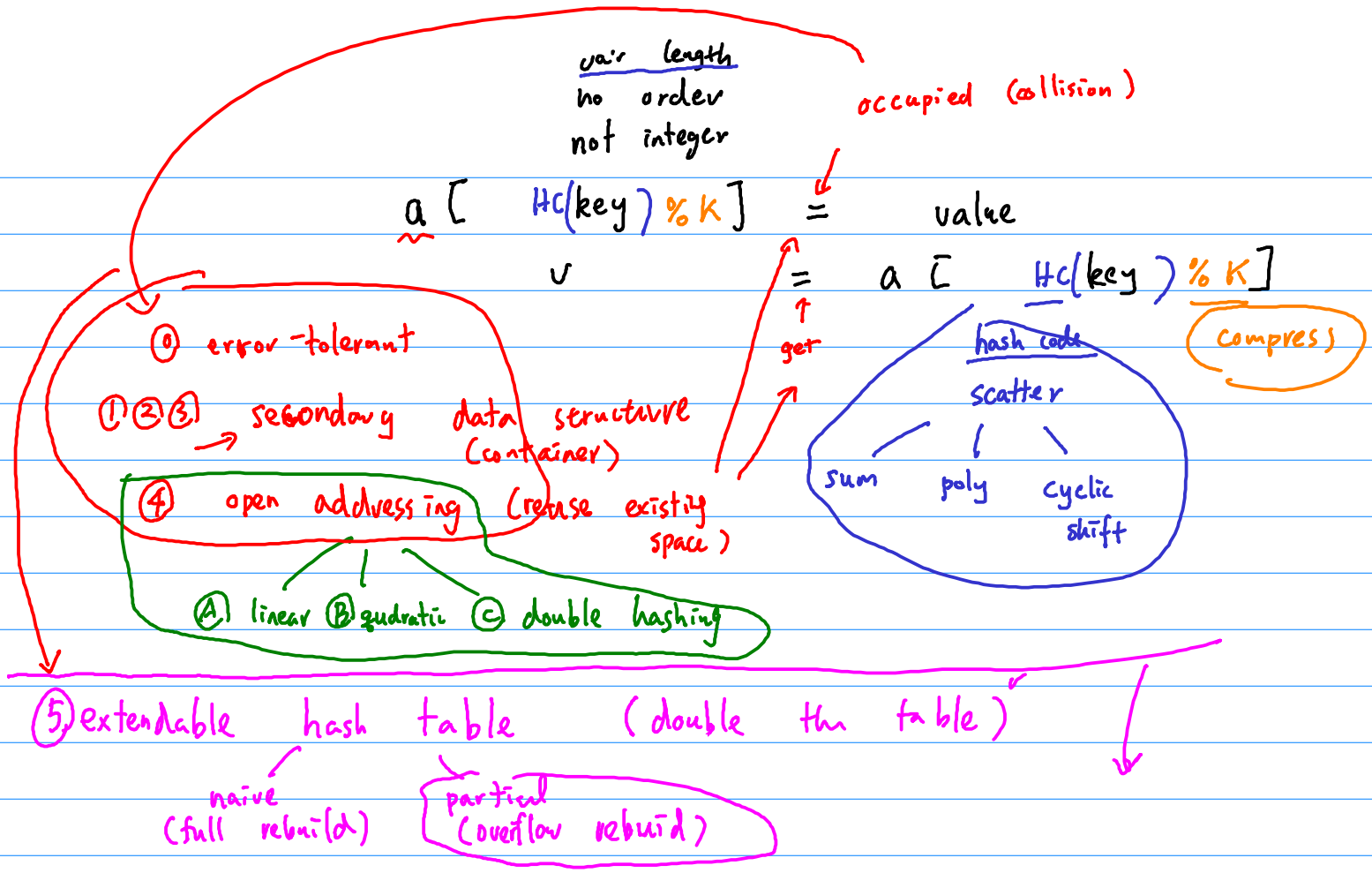
hash code



insert "egg"

partial rebuild

extendable Hash Table


var length
no order
not integer
occupied (collision)
a [HC(key) % K] = value
v
= a [HC(key) % K]
get
hash code
scatter
sum
poly
cyclic shift
compress
0 error-tolerant
1 2 3 secondary data structure (container)
4 open addressing (reuse existing space)
A linear B qudratic C double hashing
5 extendable hash table (double the table)
naive (full rebuild)
partial (overflow rebuild)