HARVARD
School of Engineering
and Applied Sciences

CS165
Data Systems

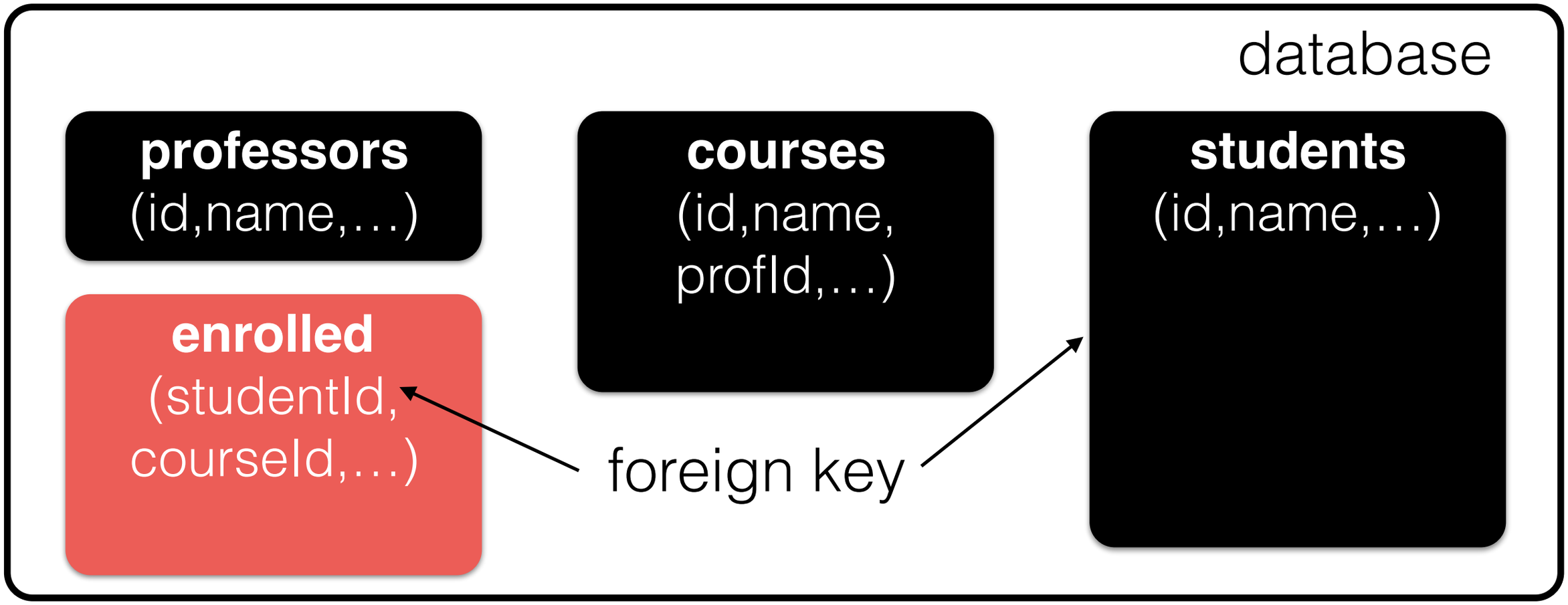
Have fun learning to design and build modern data systems

class 18

hashing

prof. Stratos Idreos

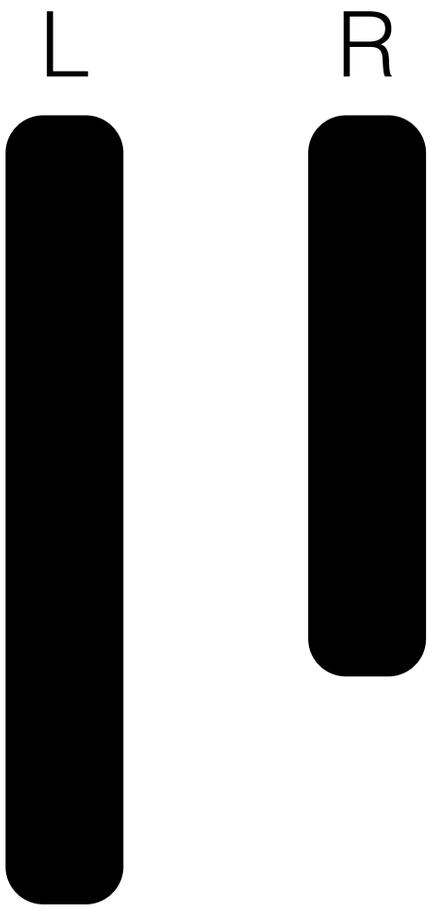
HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS165/



give me all students enrolled in cs165

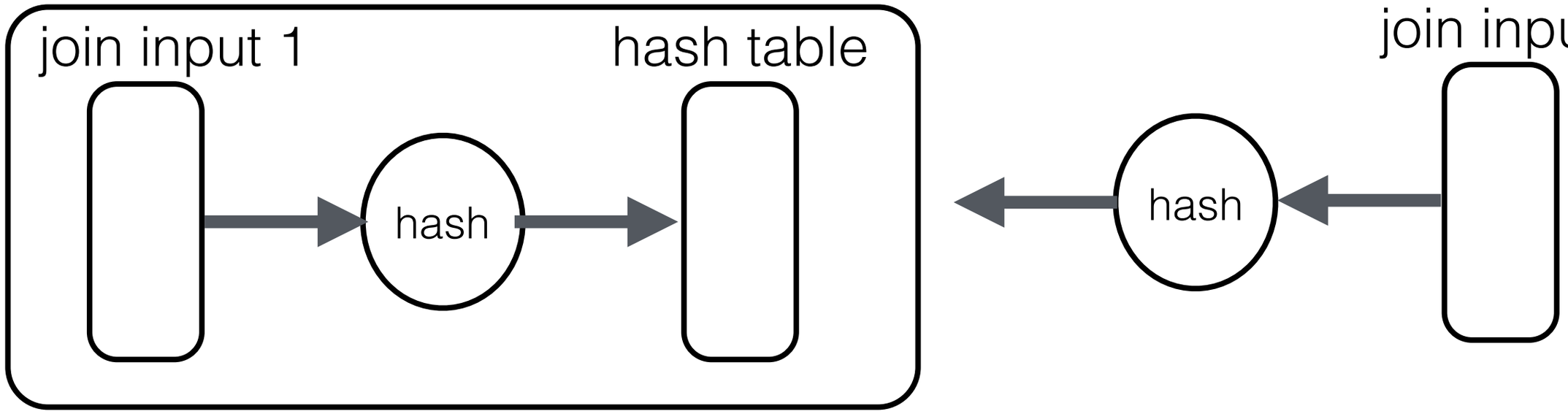
select *student.name* ***from*** *students, enrolled, courses* ***where*** *courses.name="cs165" and enrolled.courseId=course.id and student.id=enrolled.studentId* **join**

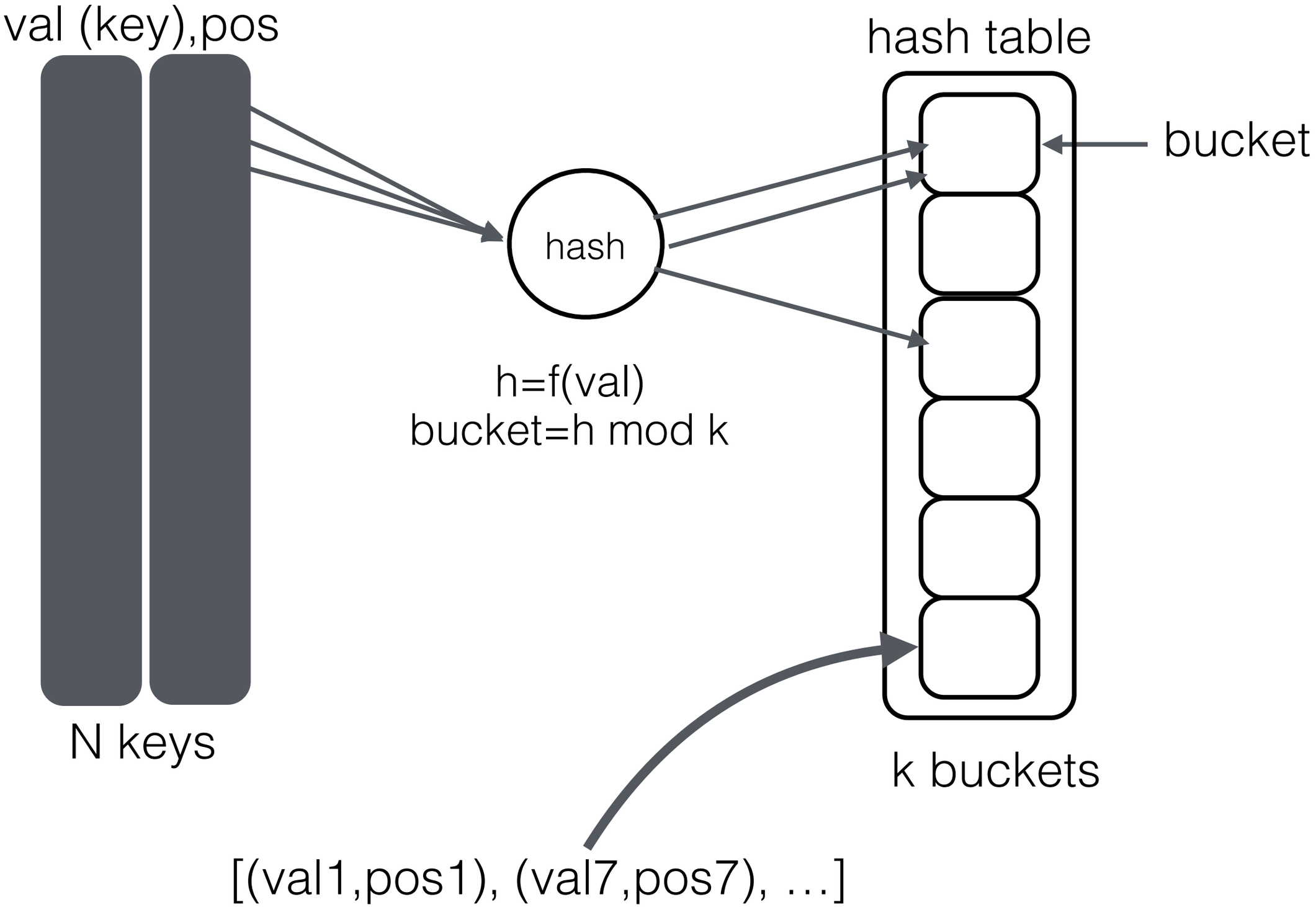
nested loops



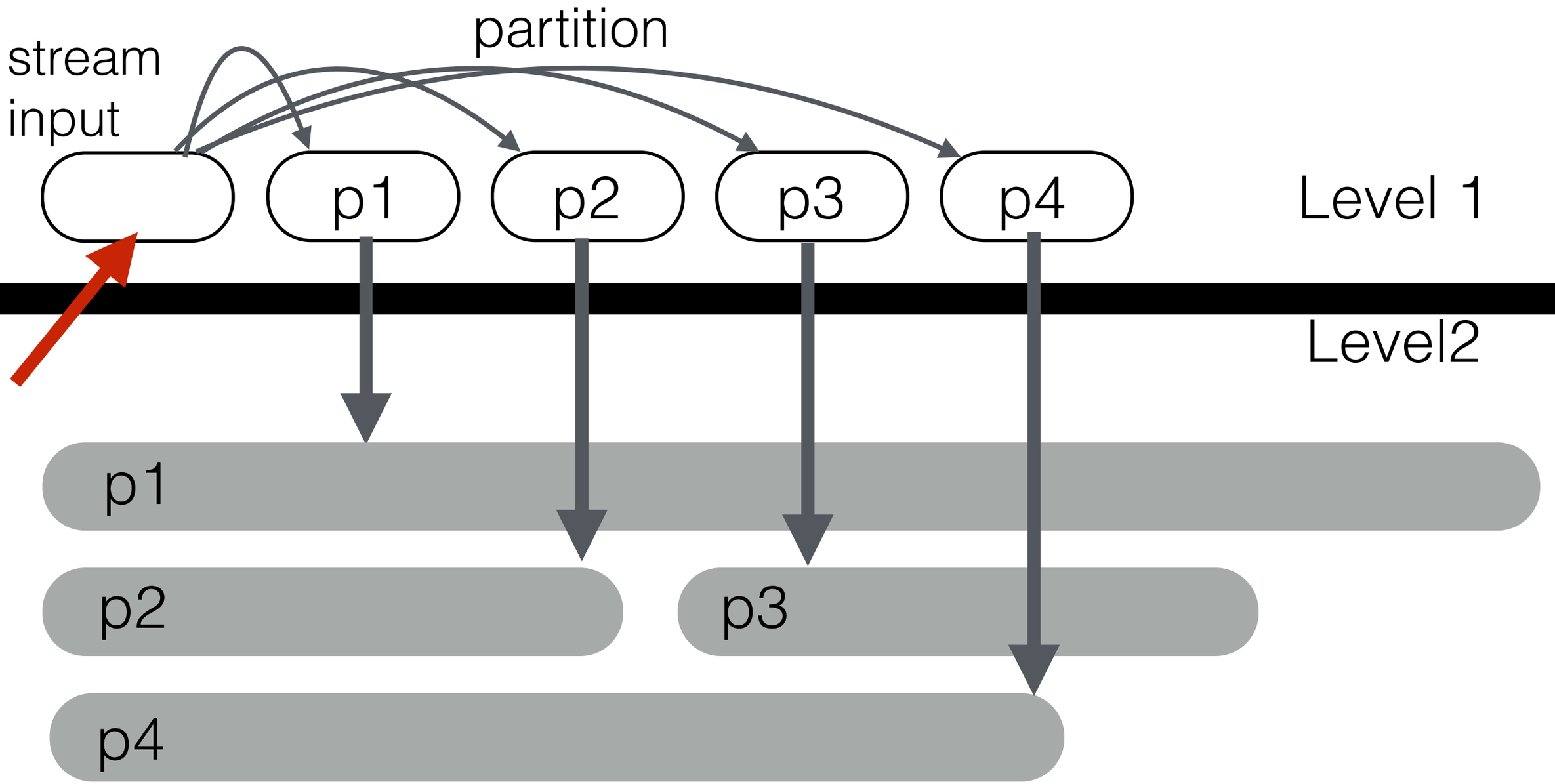

```
new resL[]; new resR[]; k=0
for (i=0;i<L.size;i=i++)
    for (j=0;j<R.size;j++)
        if L[i]==R[j]
            resL[k]=i
            resR[k++]=j
```

hash join



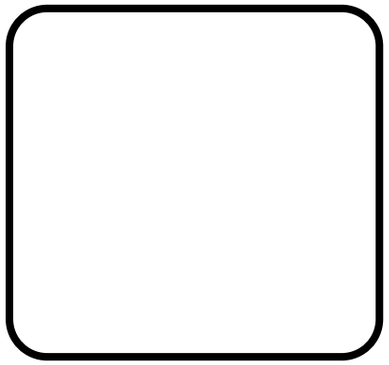

val (key),pos
hash table
bucket
hash
h=f(val)
bucket=h mod k
N keys
k buckets
[(val1,pos1), (val7,pos7), …]

1. read input into stream buffer, hash and write to respective partition buffer
2. when input buffer is consumed, bring the next one
3. when a partition buffer is full, write to L2
we can partition into L1-1 pieces in one pass

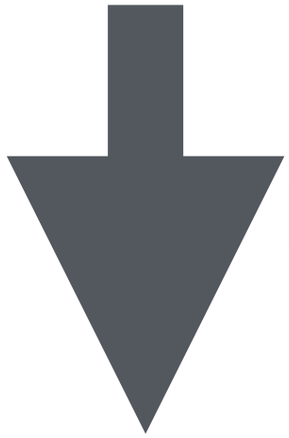


grace hash join

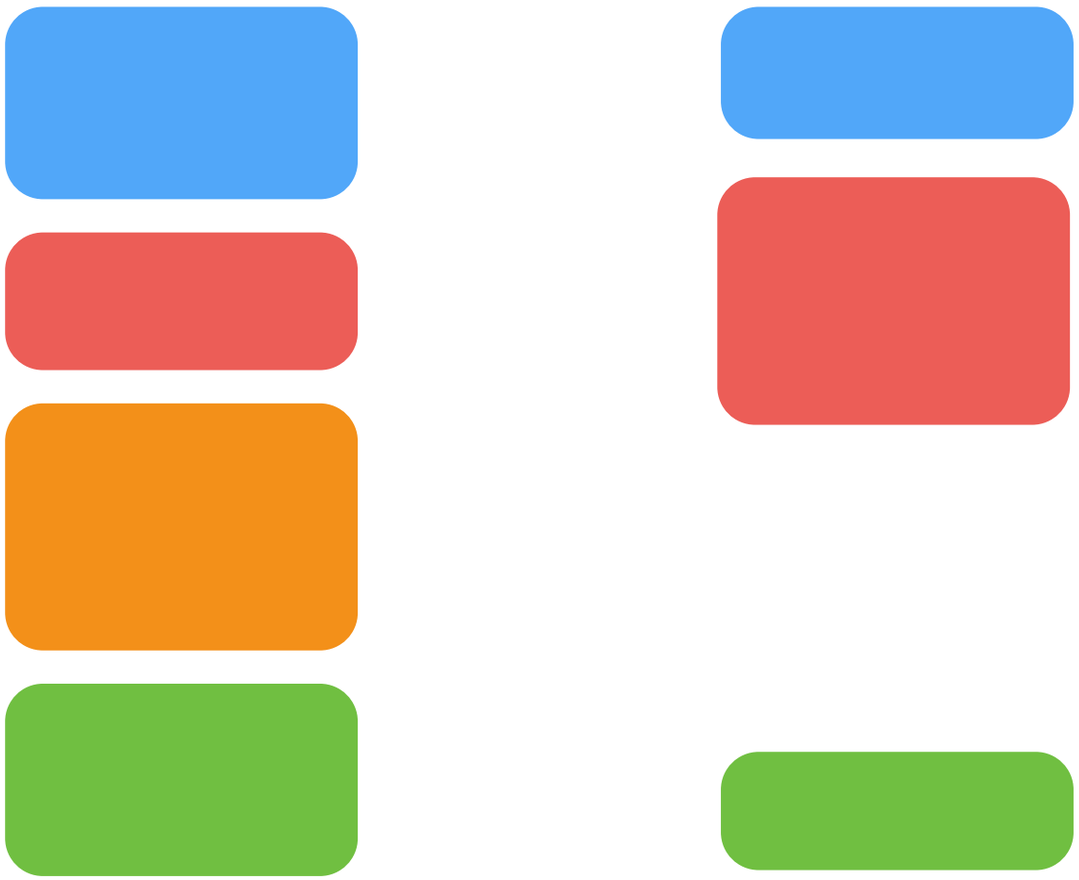
join input 1



join input 2



hash partitioning



then join each pair of partitions independently in memory

as long as at least one of the pieces <=L1-2

stream input left

probe

L1-2

p1 right hash table

result

Level 1

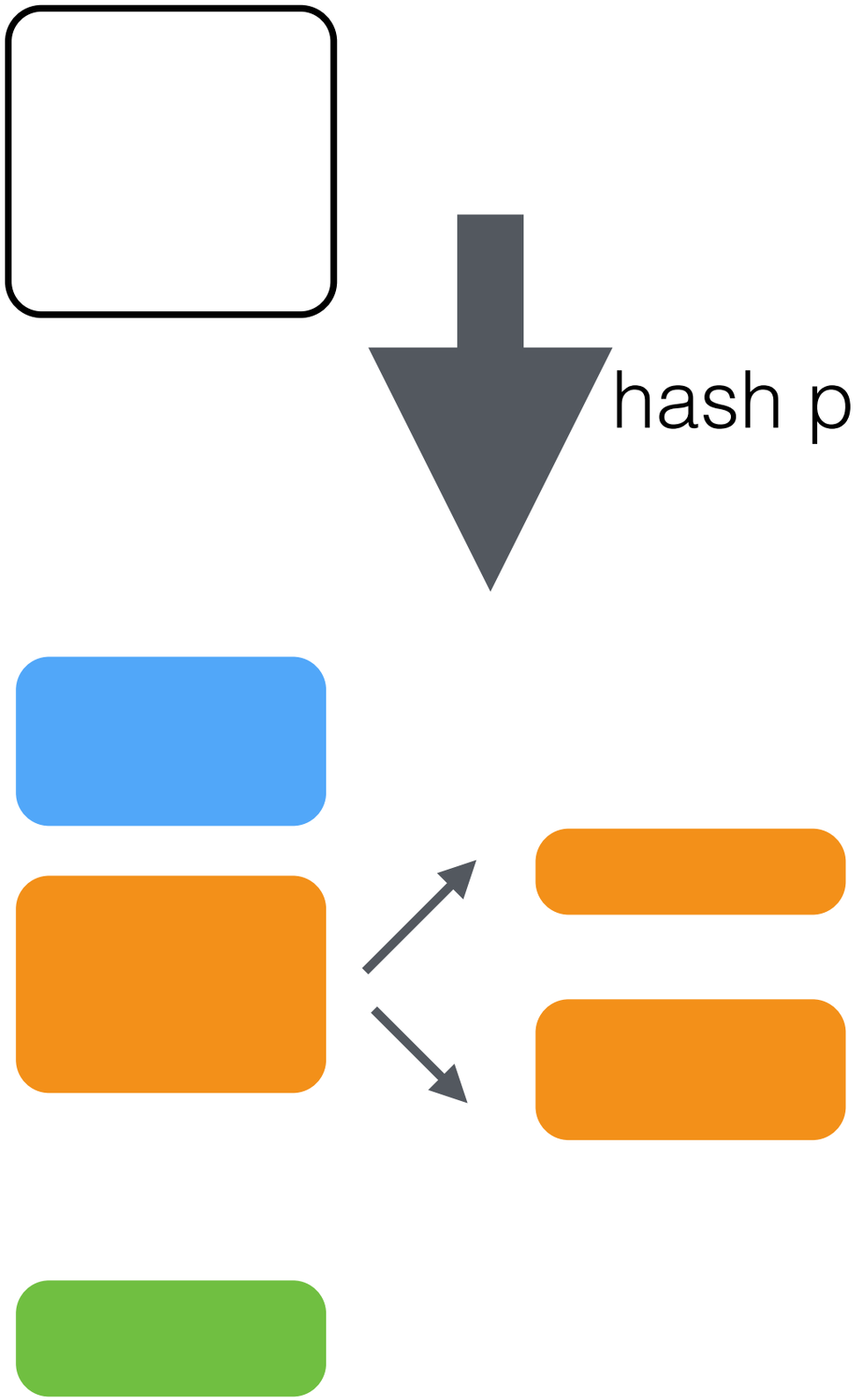
Level2

p1 left

p1 right

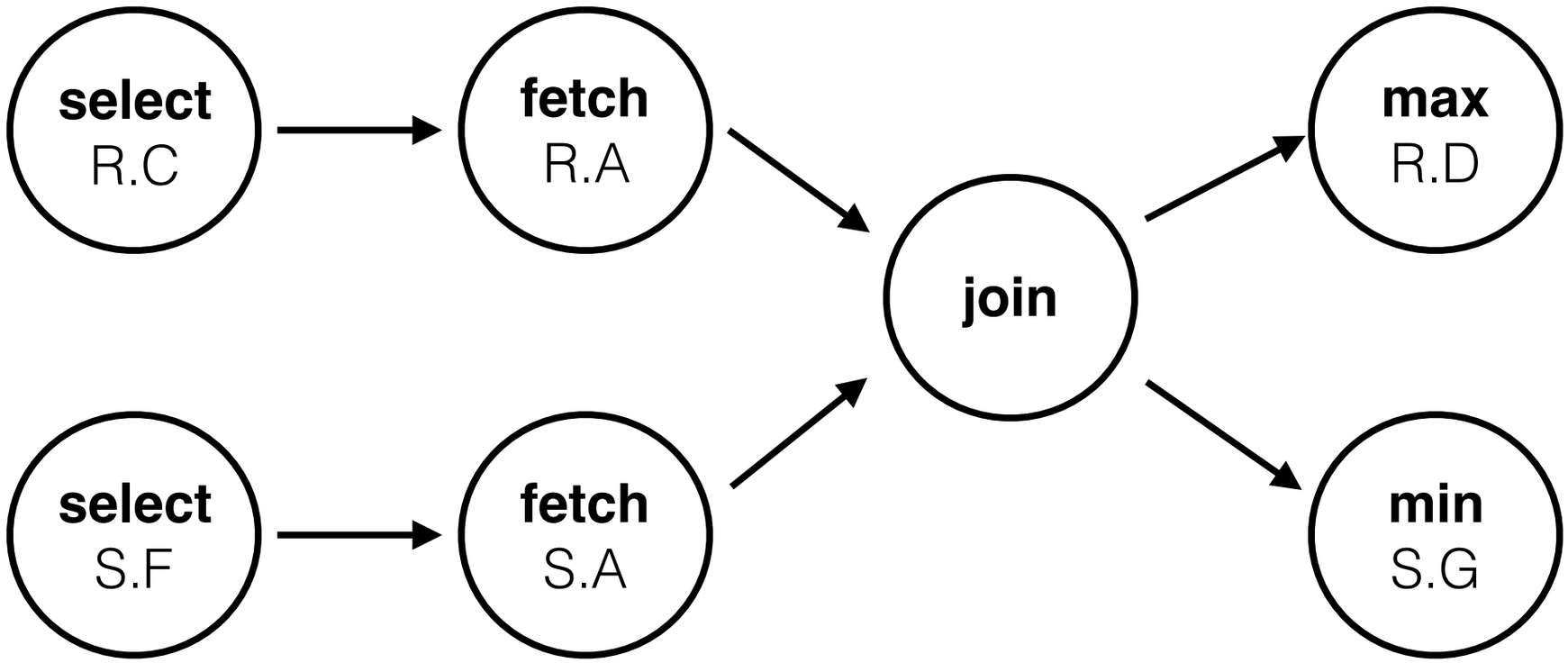
result

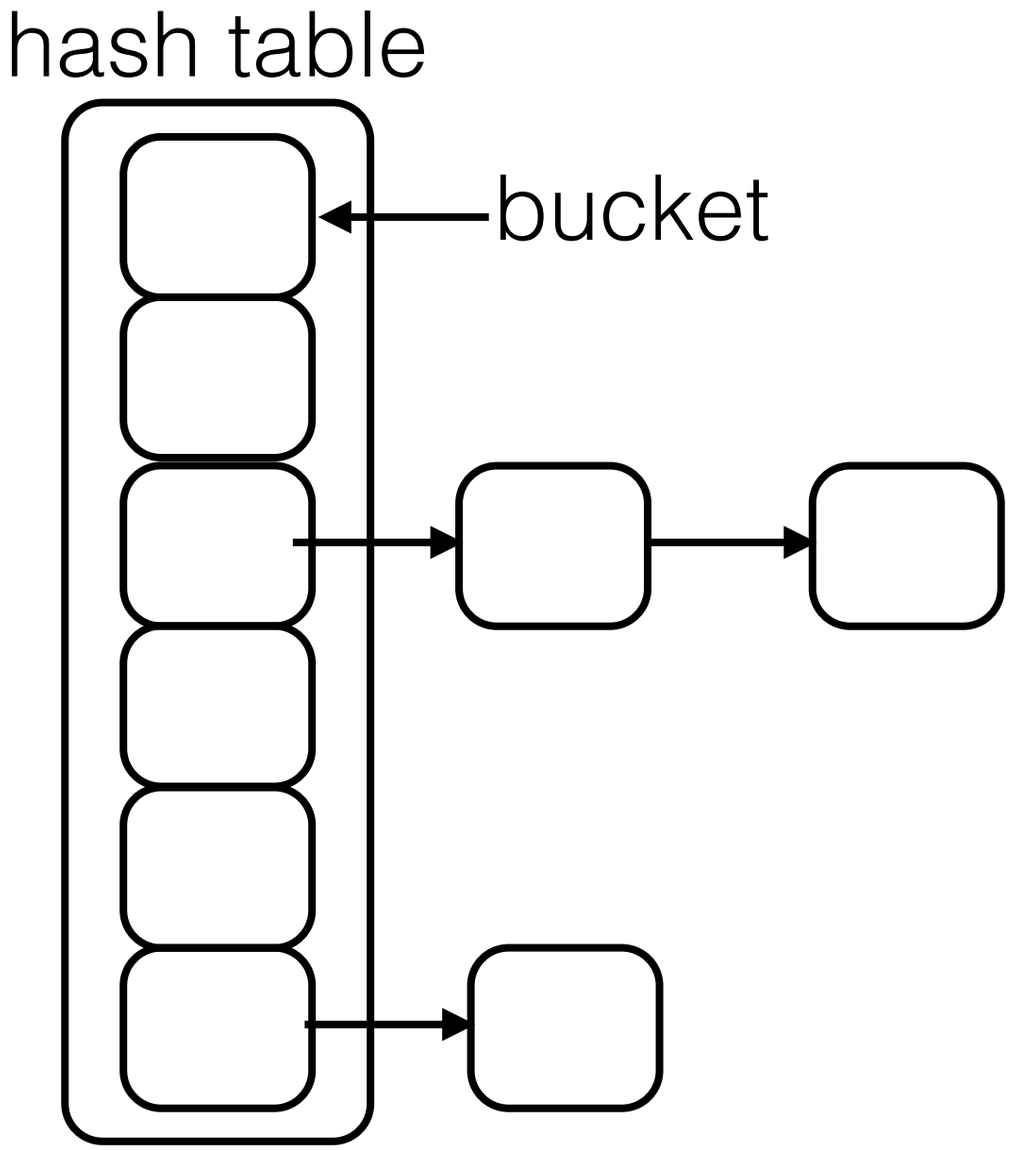
grace hash join



apply recursively if a partition does not fit in memory (L1-2)

today: how to create the hash table

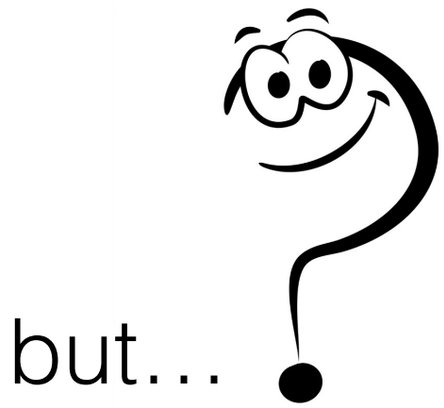

select
R.C
fetch
R.A
max
R.D
join
select
S.F
fetch
S.A
min
S.G



updates may create long lists

can this happen anyway,
i.e., even with no updates

simple solution would be to
double the hash table and rehash everything…

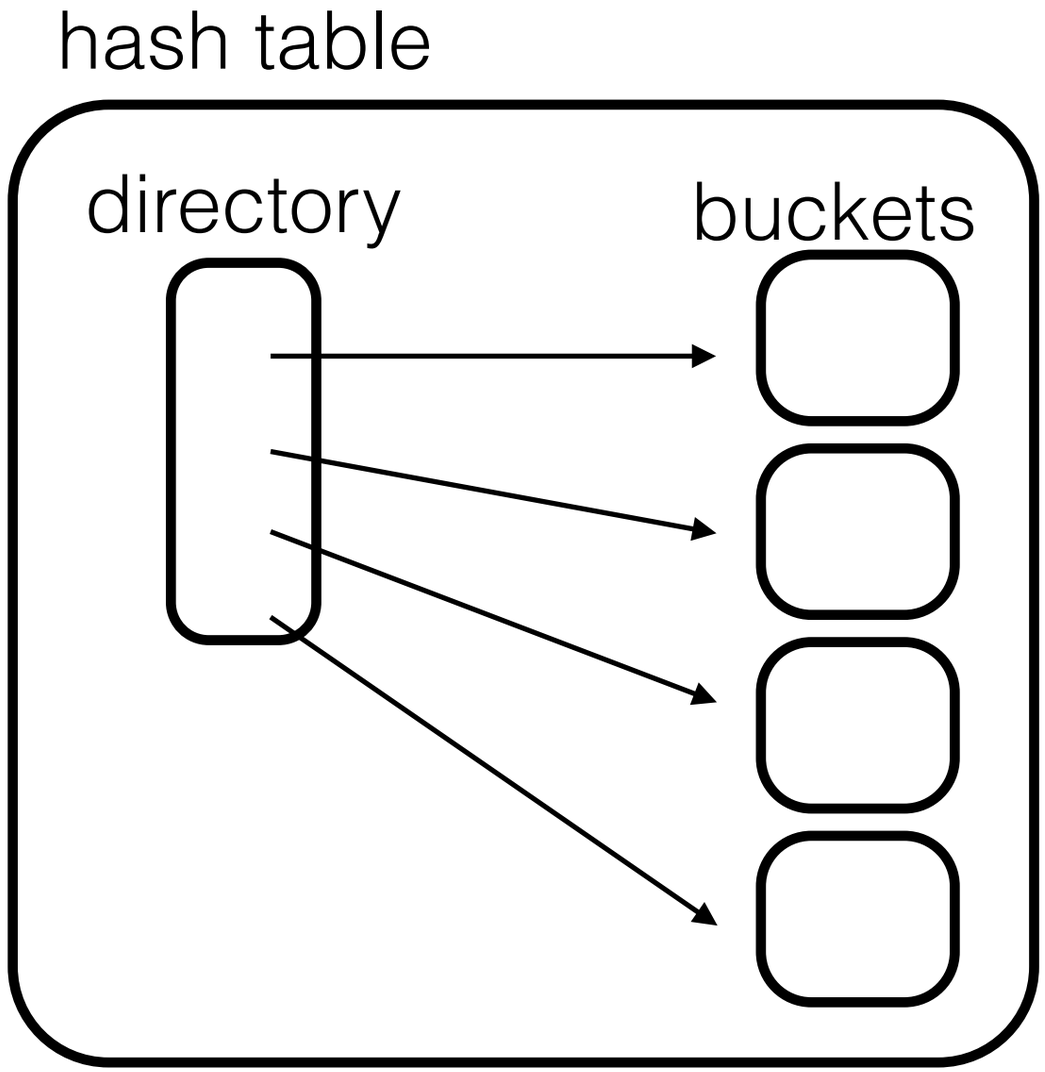


but…

dynamic hashing

extendible hashing - linear hashing

extendible hashing

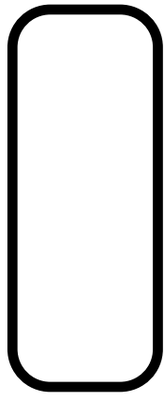


when there is no more space
split individual buckets
and if needed **double the directory**

with directory in memory
we need 1 I/O to fetch the bucket

not true from memory to caches in general

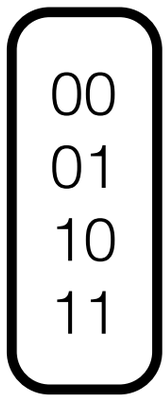
directory **maps hash values to buckets**



h=hash(key)

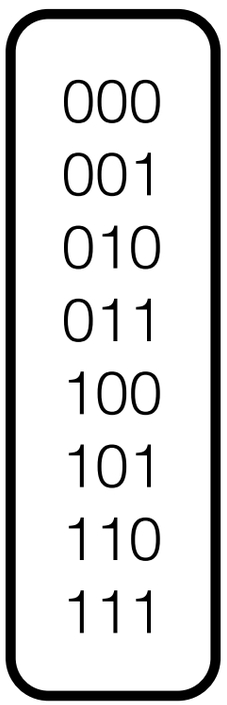
use binary form of h
use X last bits to map 2^x buckets

for X=2

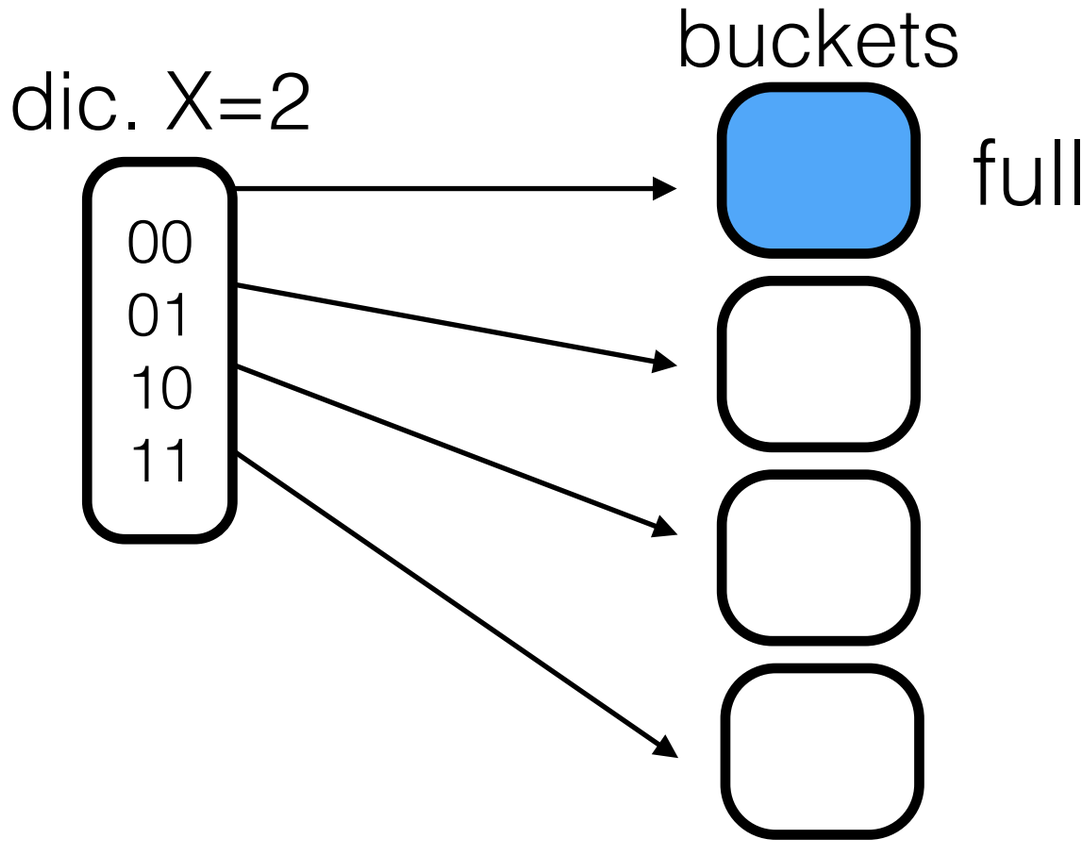


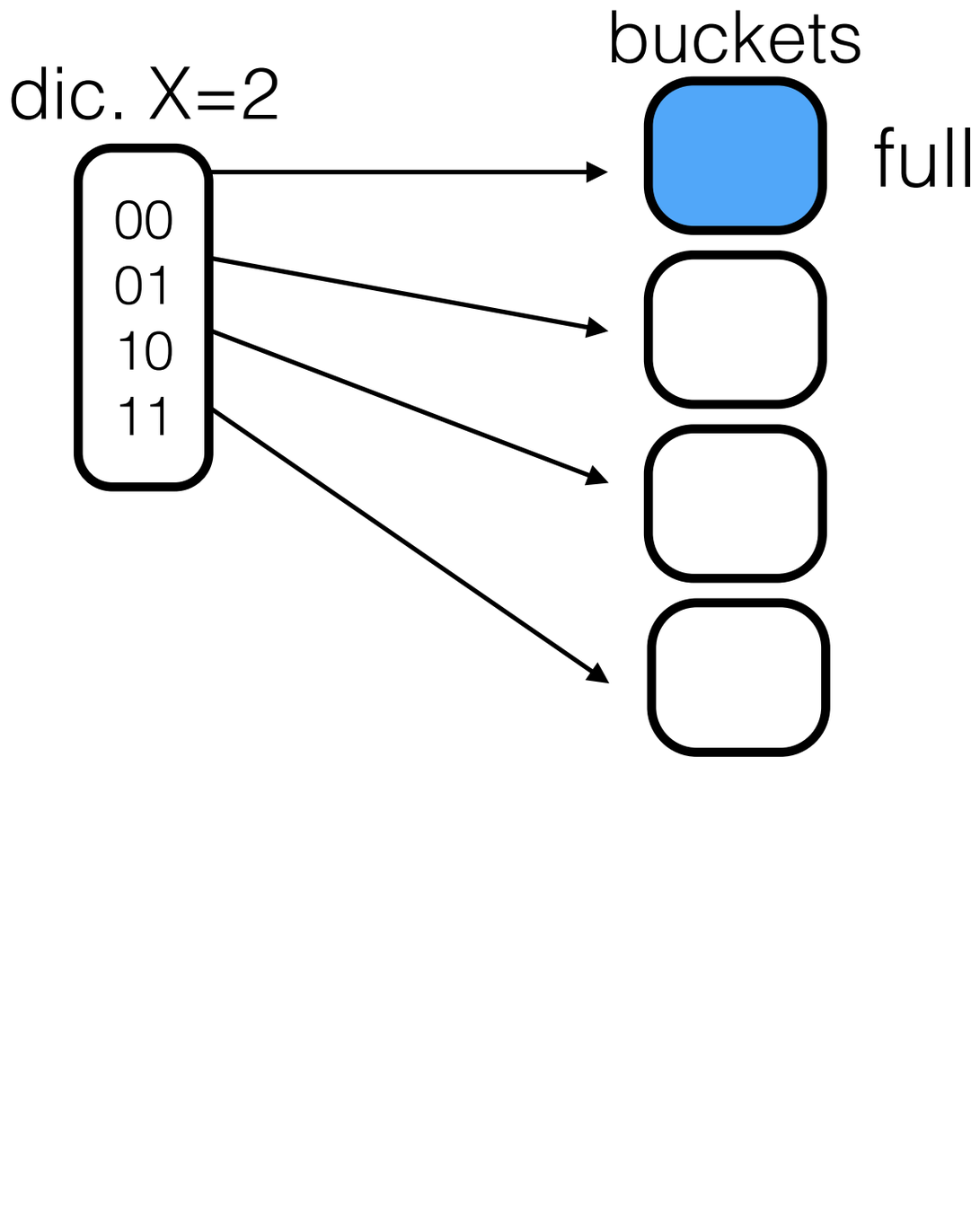
extending the directory

for X=3



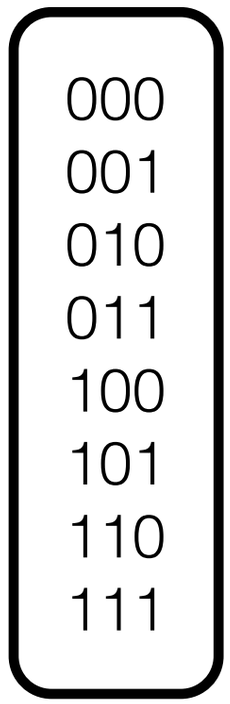
...

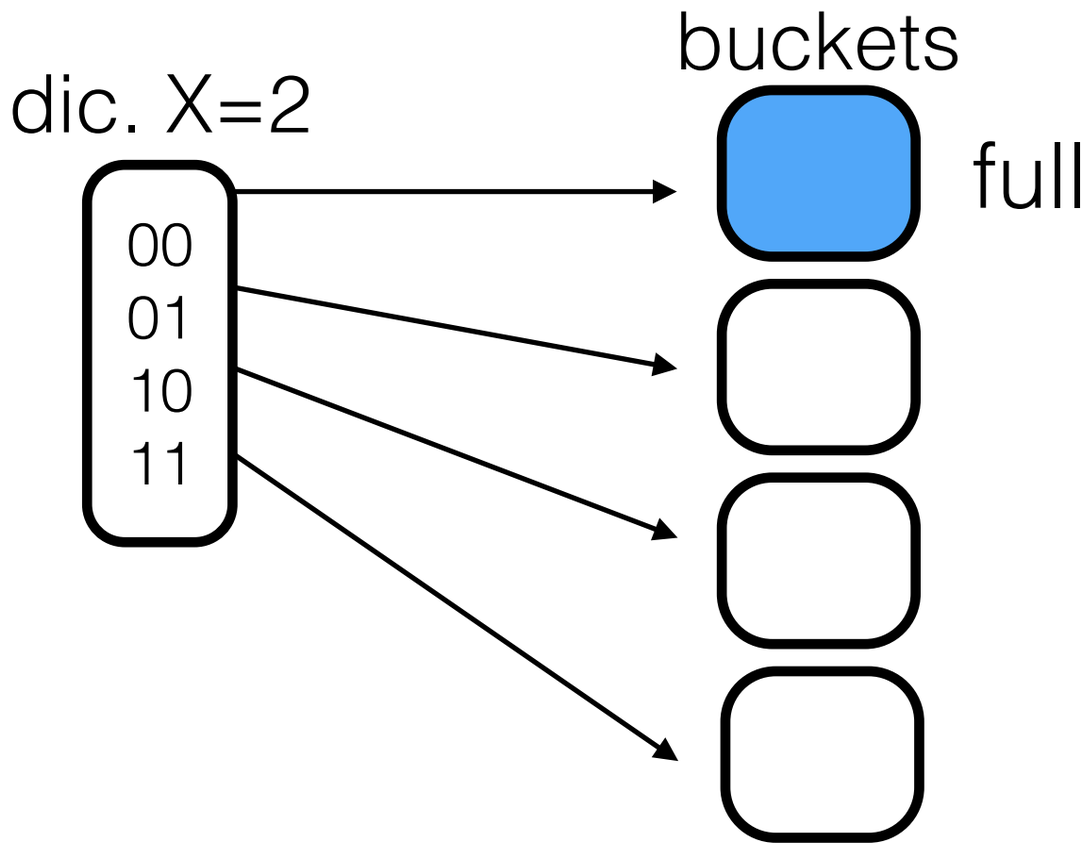

buckets
dic. X=2
full
00
01
10
11

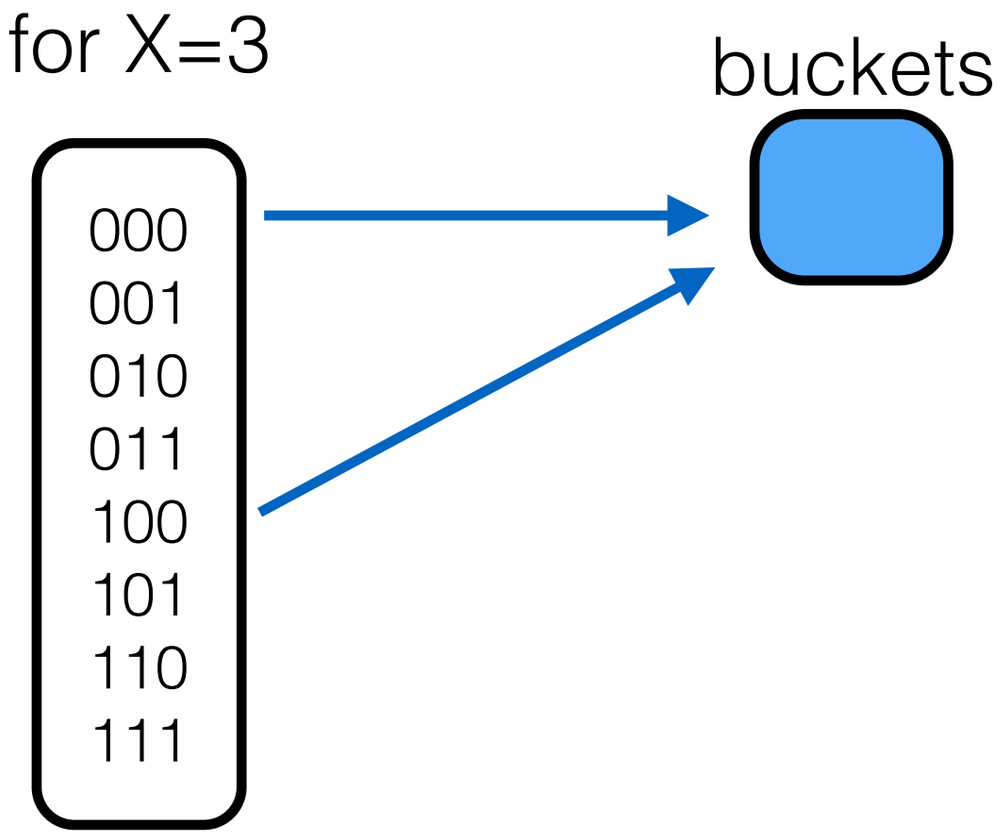


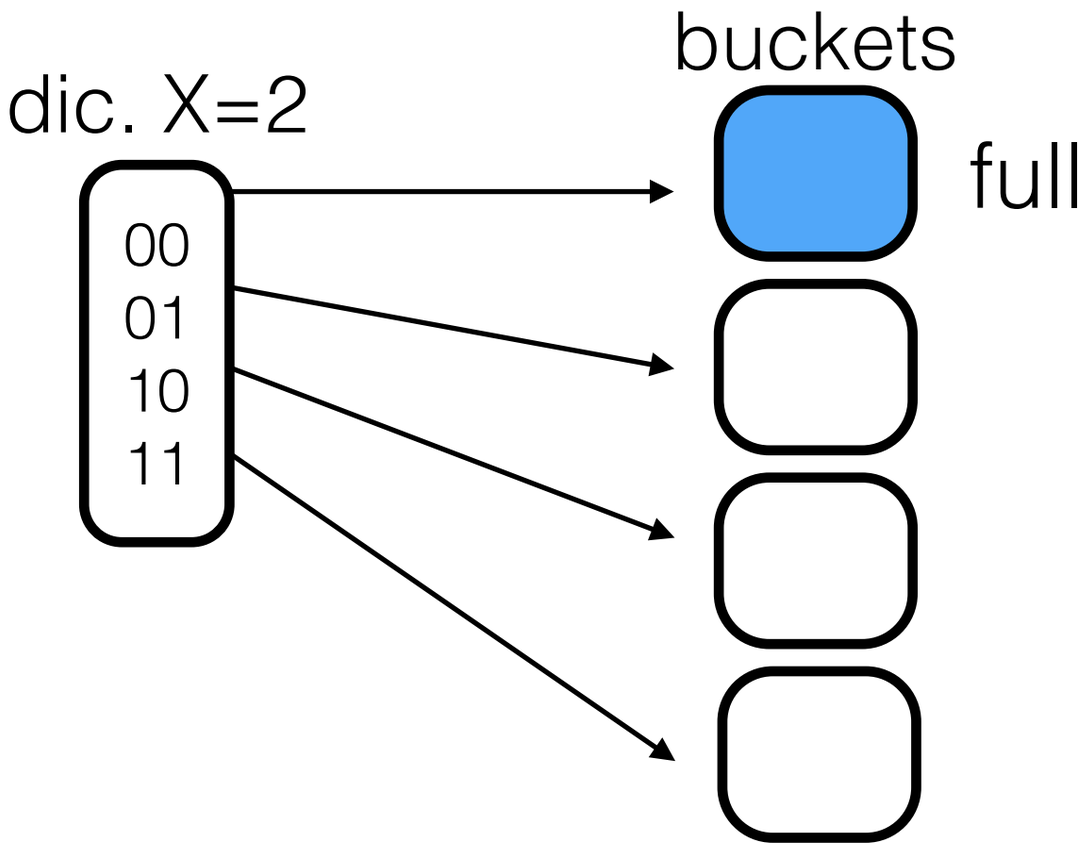
for X=3

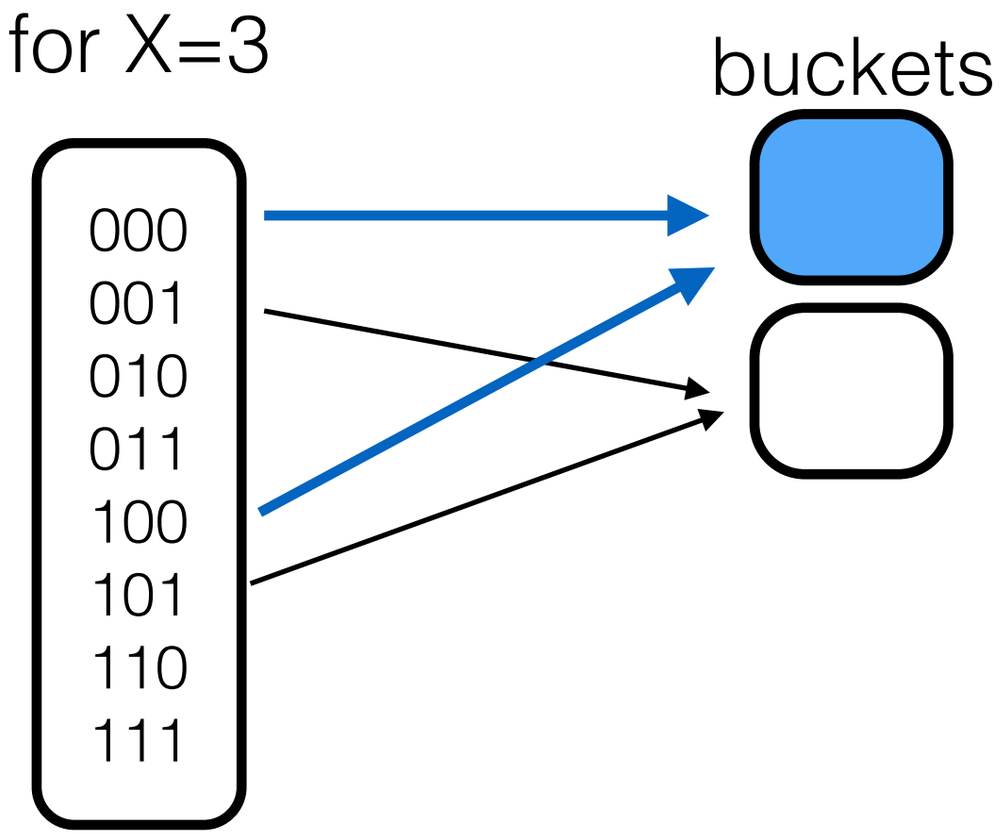
buckets

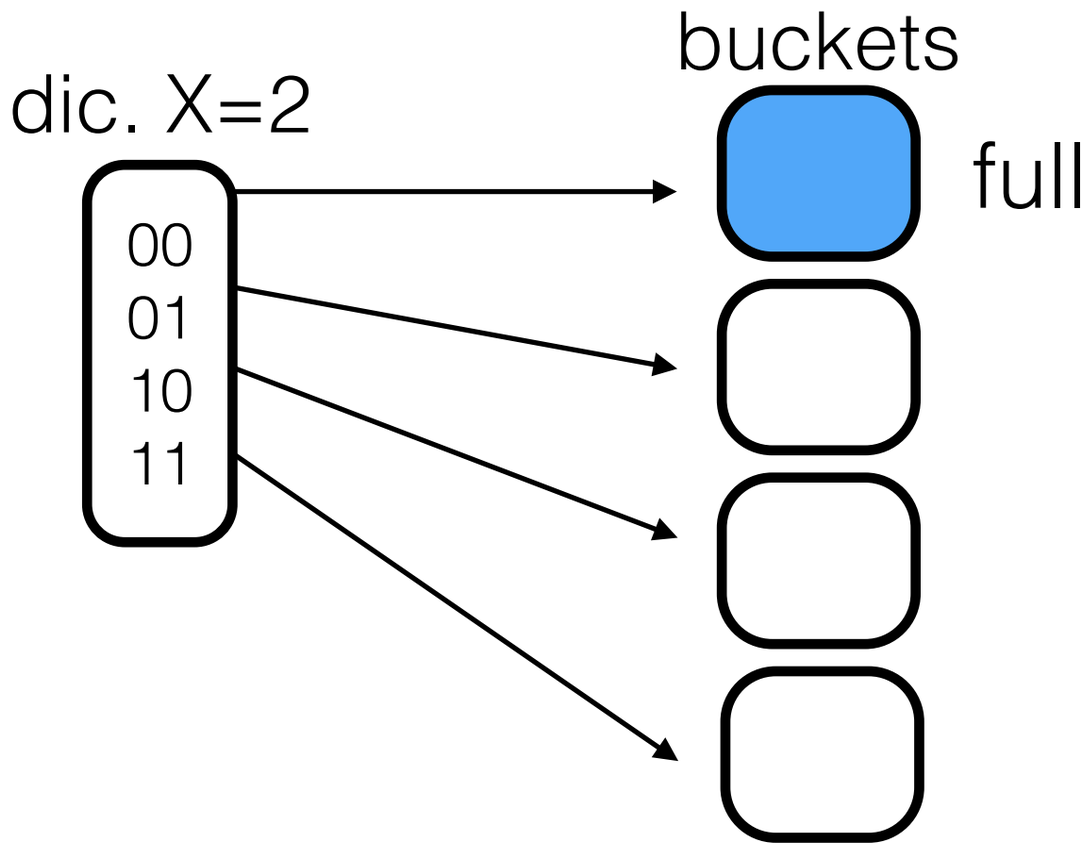


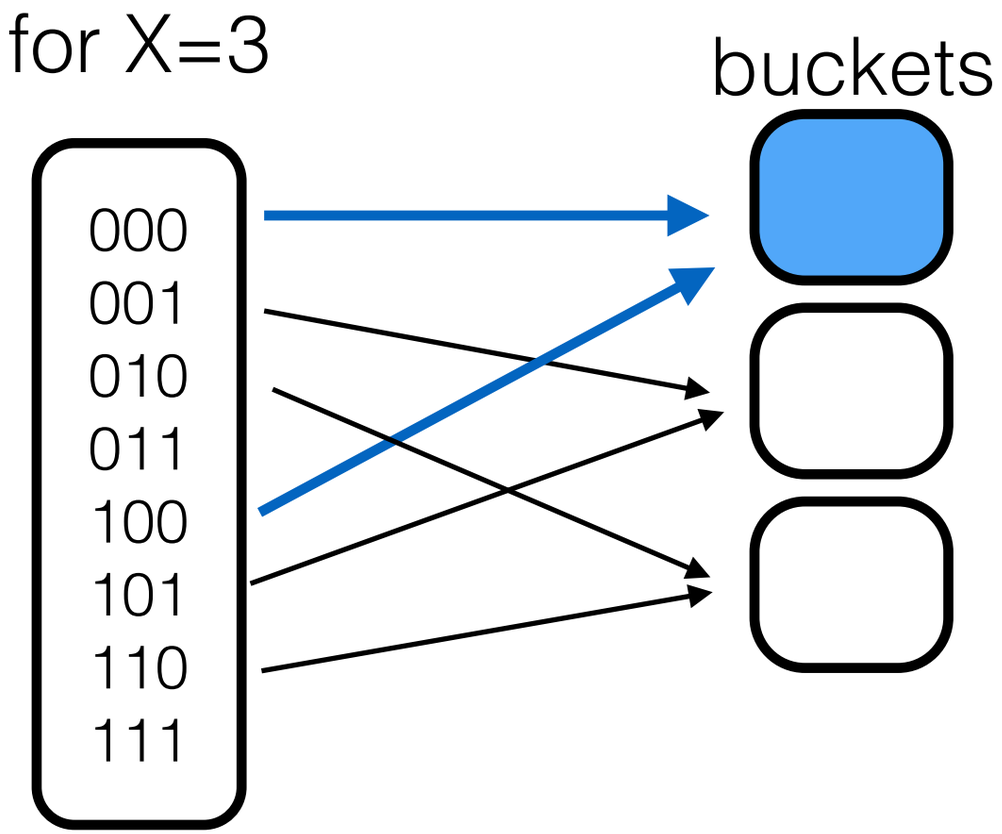

buckets
dic. X=2
full
00
01
10
11

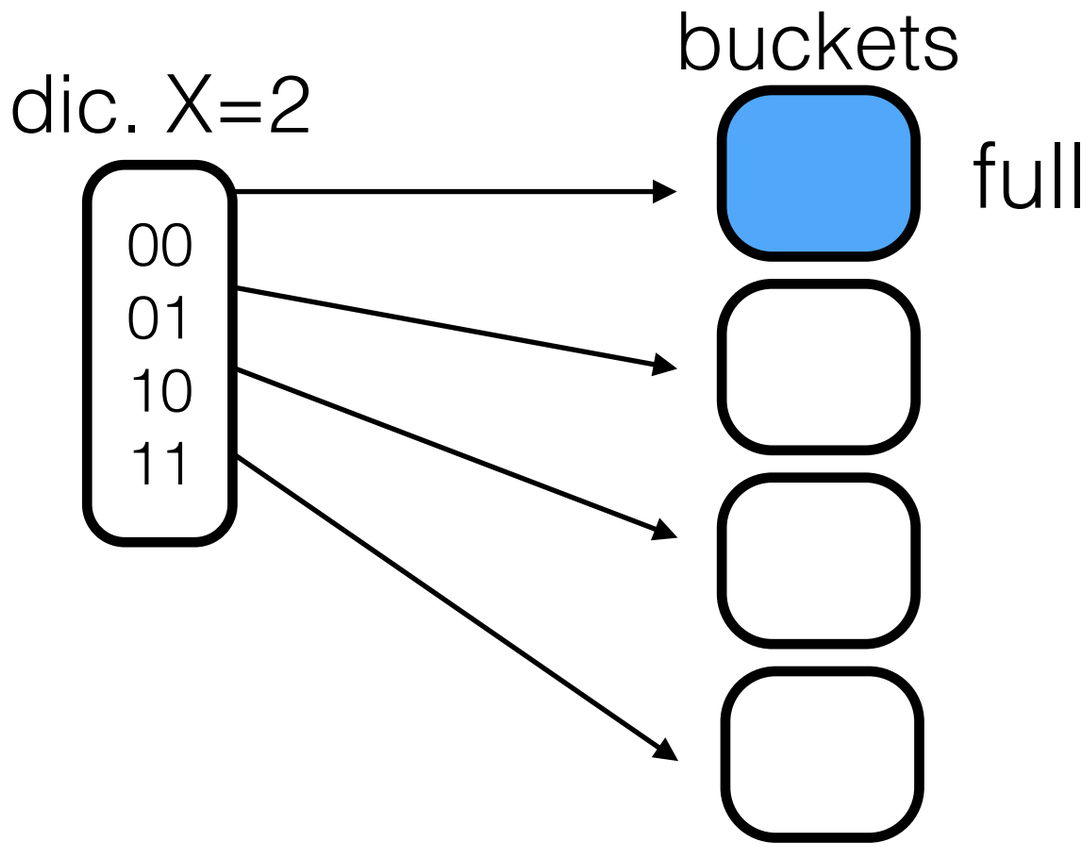

for X=3
buckets
000
001
010
011
100
101
110
111

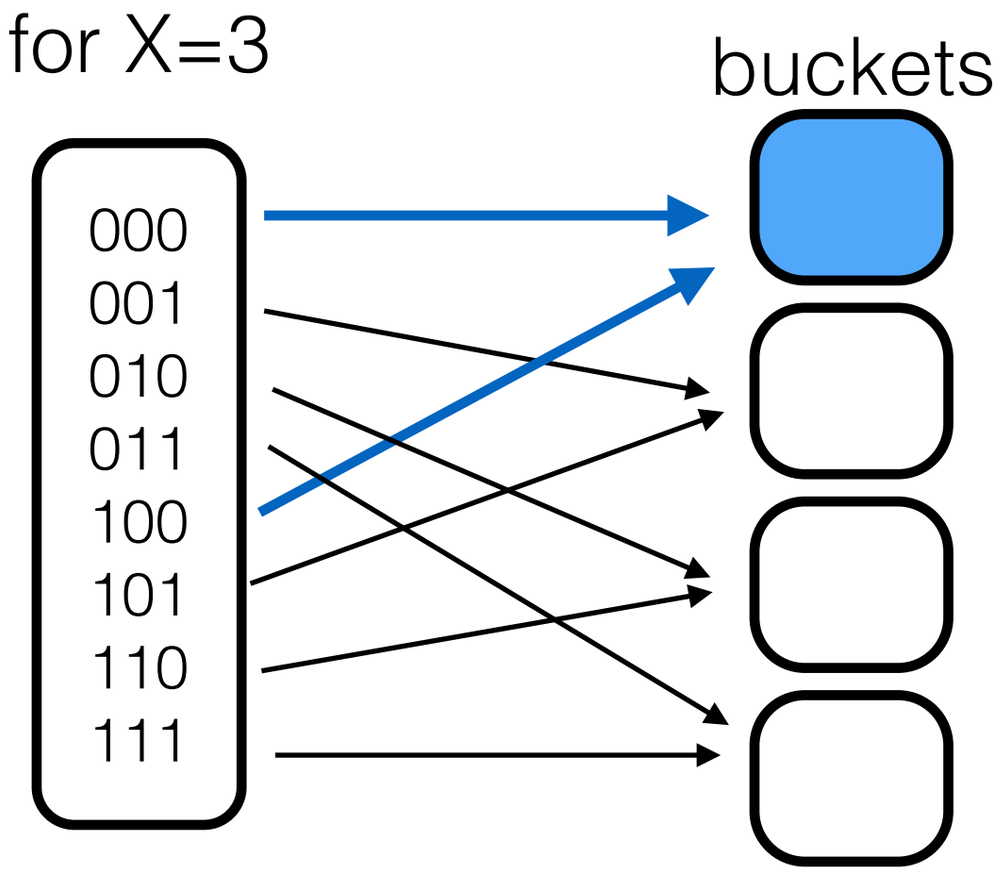

buckets
dic. X=2
full
00
01
10
11

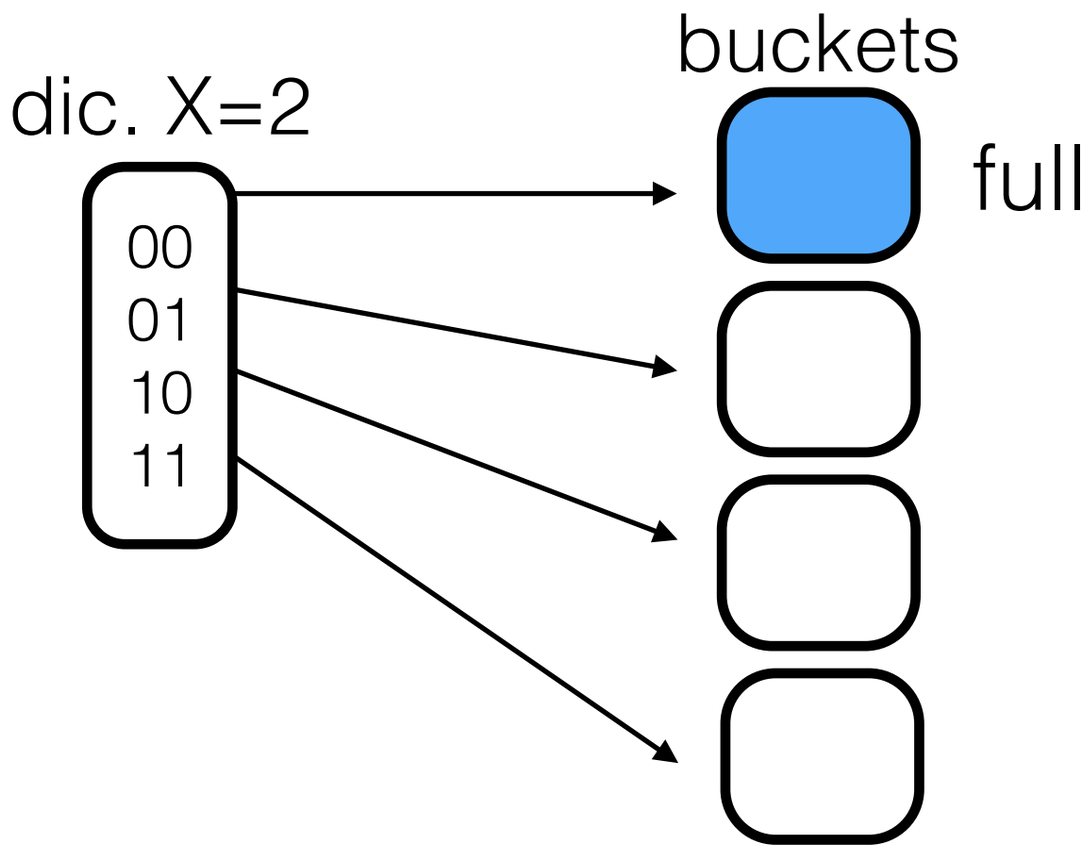

for X=3
buckets
000
001
010
011
100
101
110
111

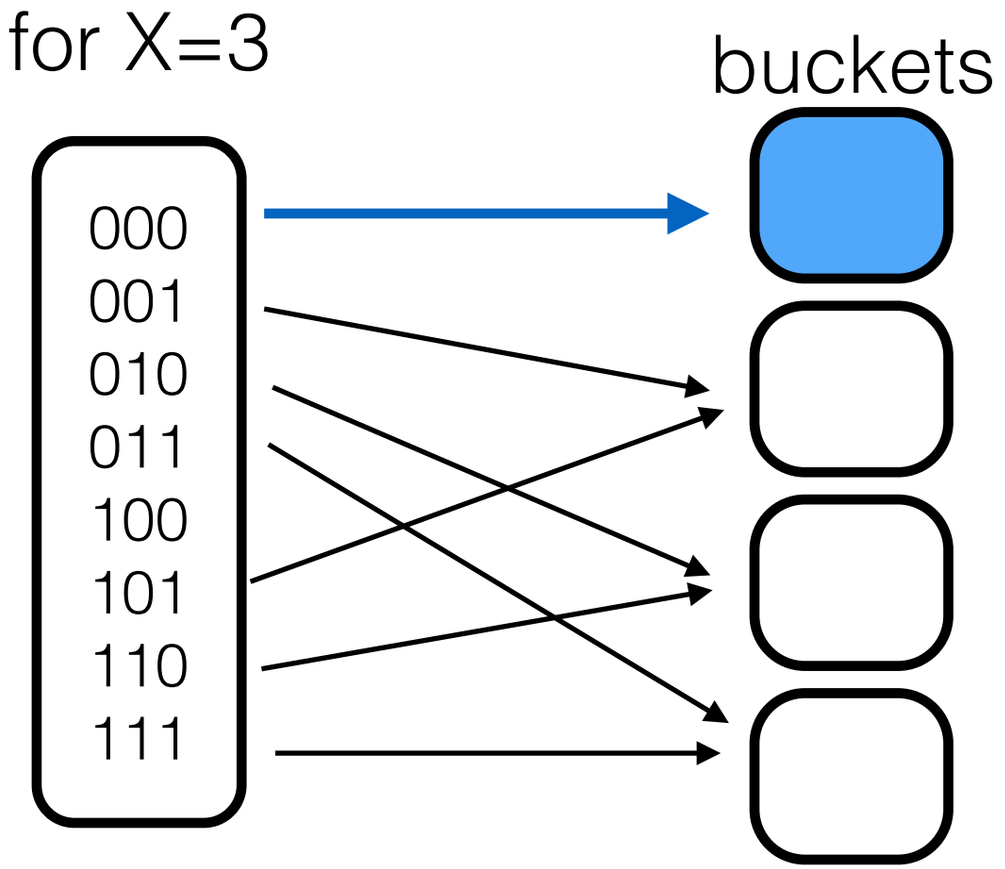

buckets
dic. X=2
full
00
01
10
11

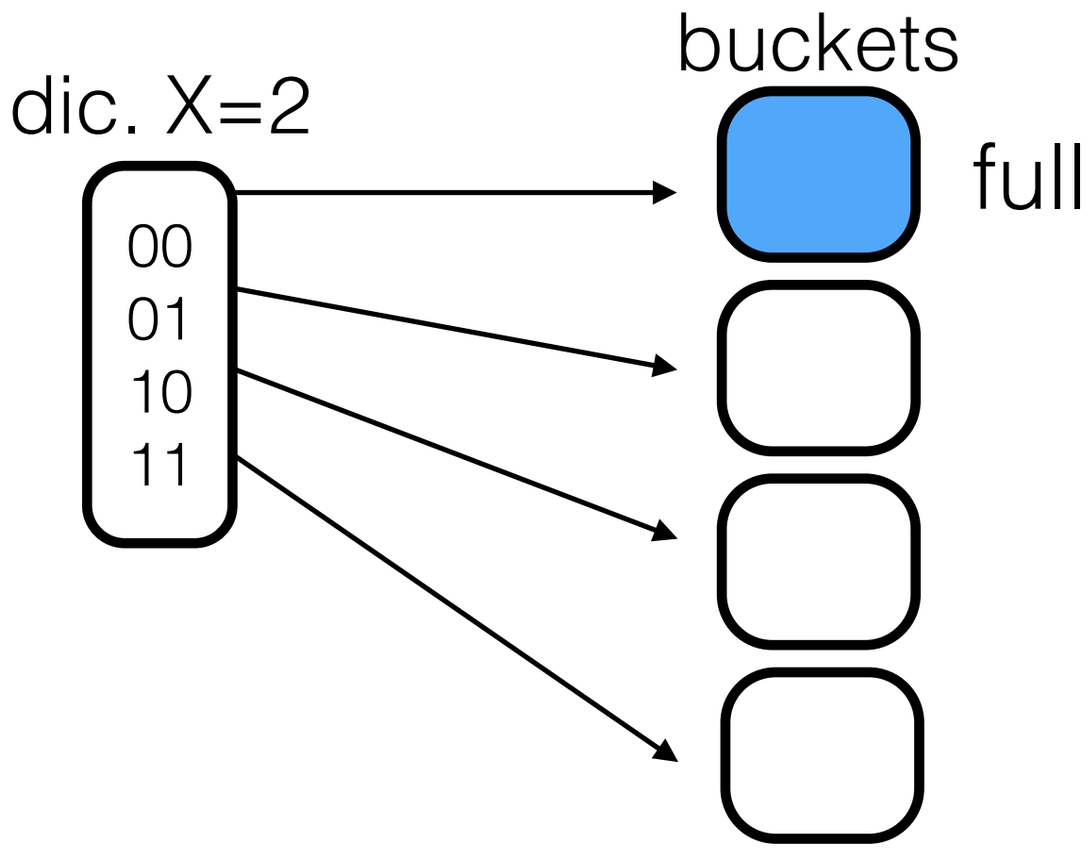

for X=3
buckets
000
001
010
011
100
101
110
111

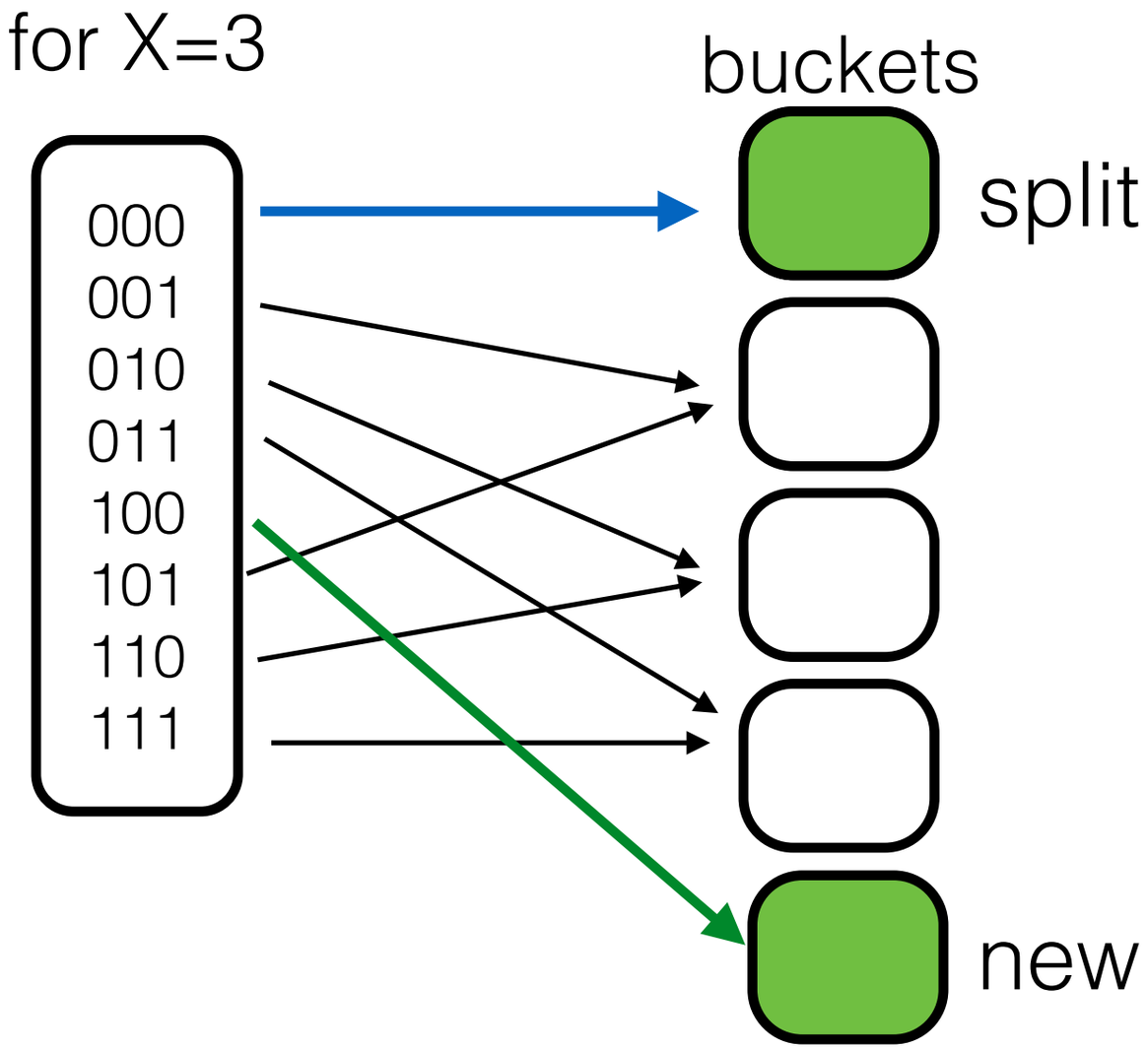

buckets
dic. X=2
full
00
01
10
11

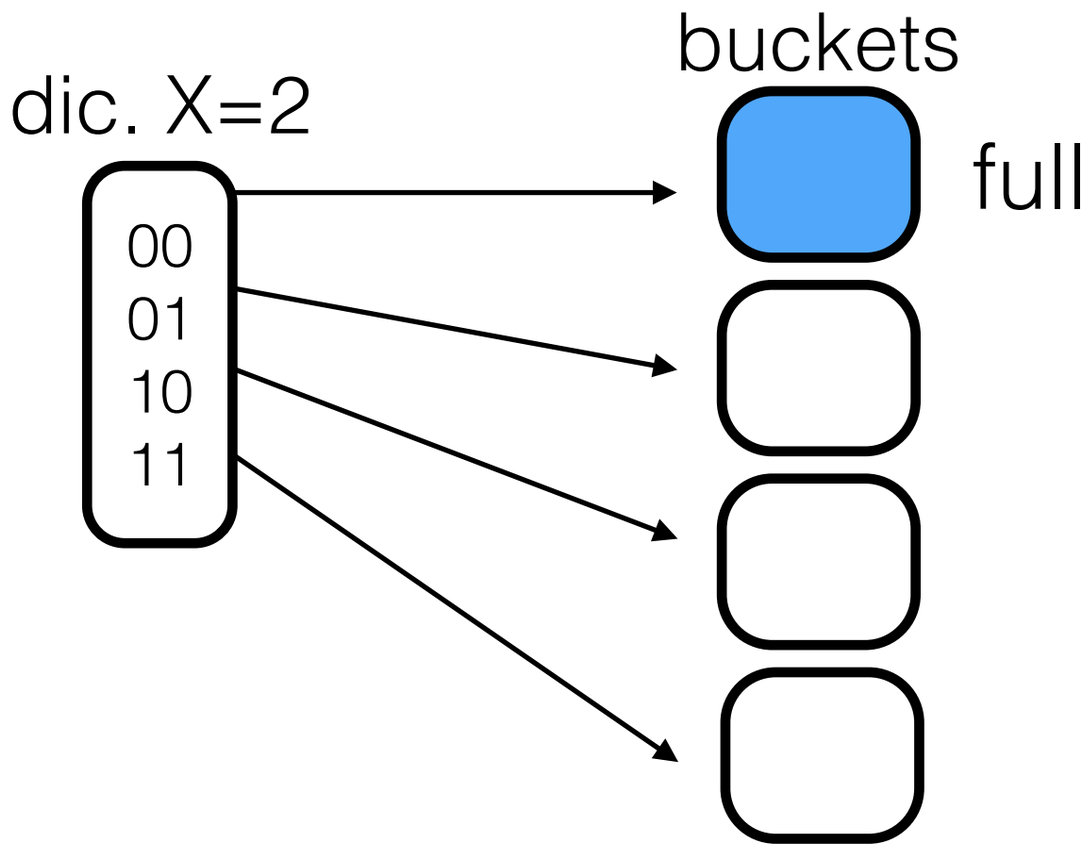

for X=3
buckets
000
001
010
011
100
101
110
111

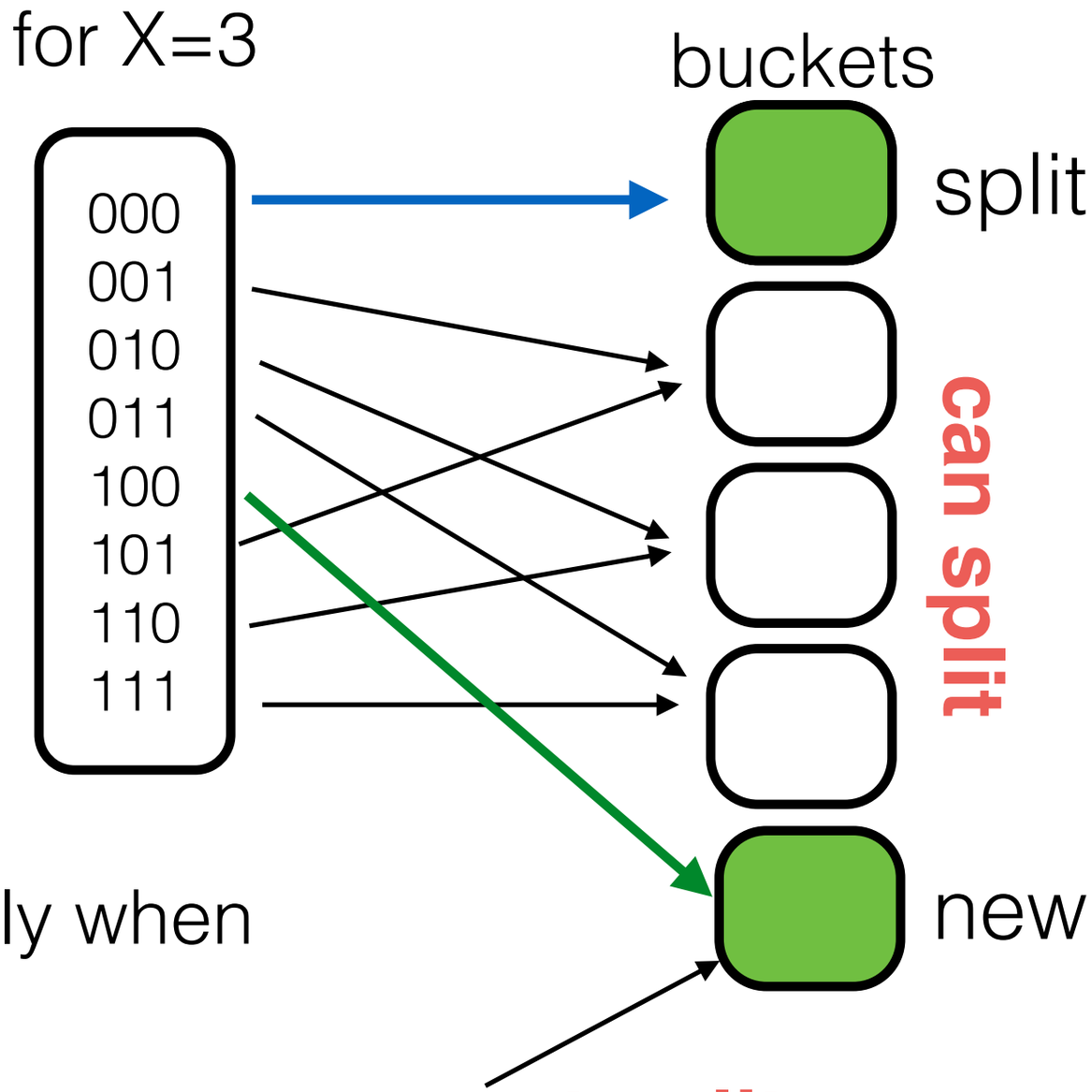

buckets
dic. X=2
full
00
01
10
11


for X=3
buckets
000
001
010
011
100
101
110
111


buckets
dic. X=2
00
01
10
11
full


for X=3
buckets
000
001
010
011
100
101
110
111
split
new





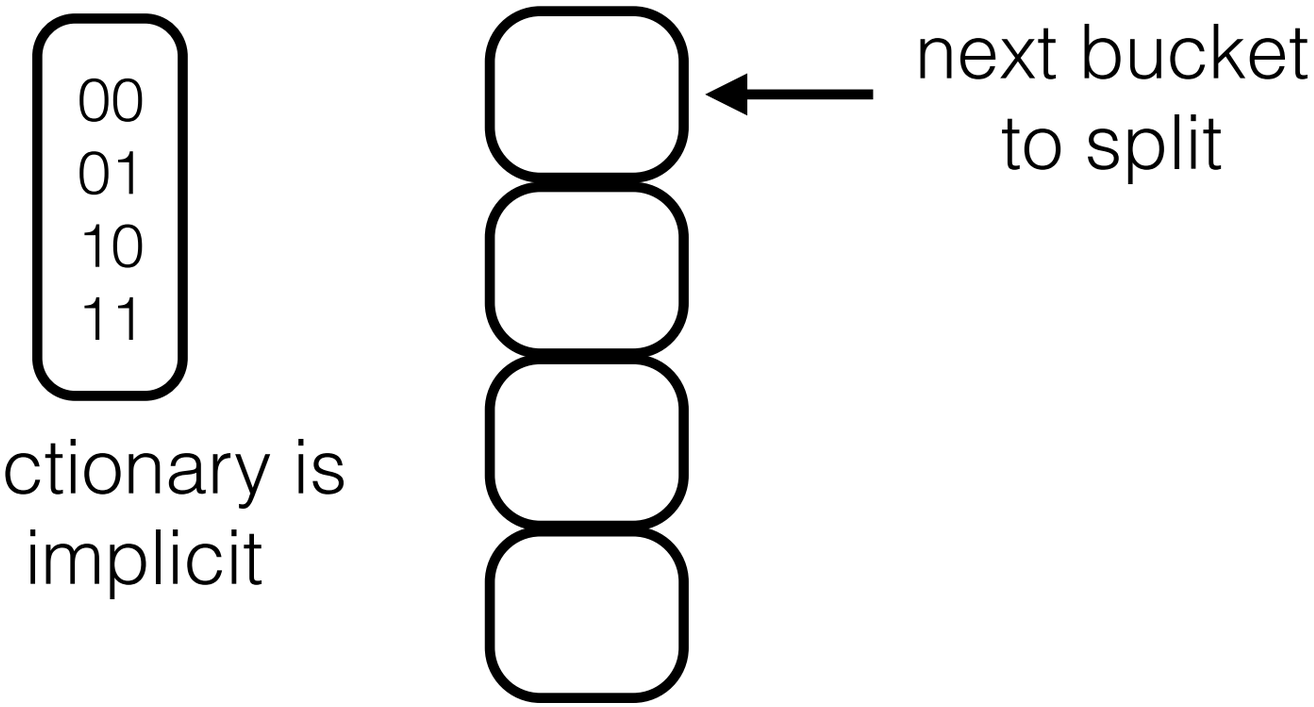
extend directory only when
no split is possible

linear hashing

do not use a directory (indirection cost)

instead incrementally extend
the hash table one bucket at a time

e.g.,use 2 bits of hash value (h0)

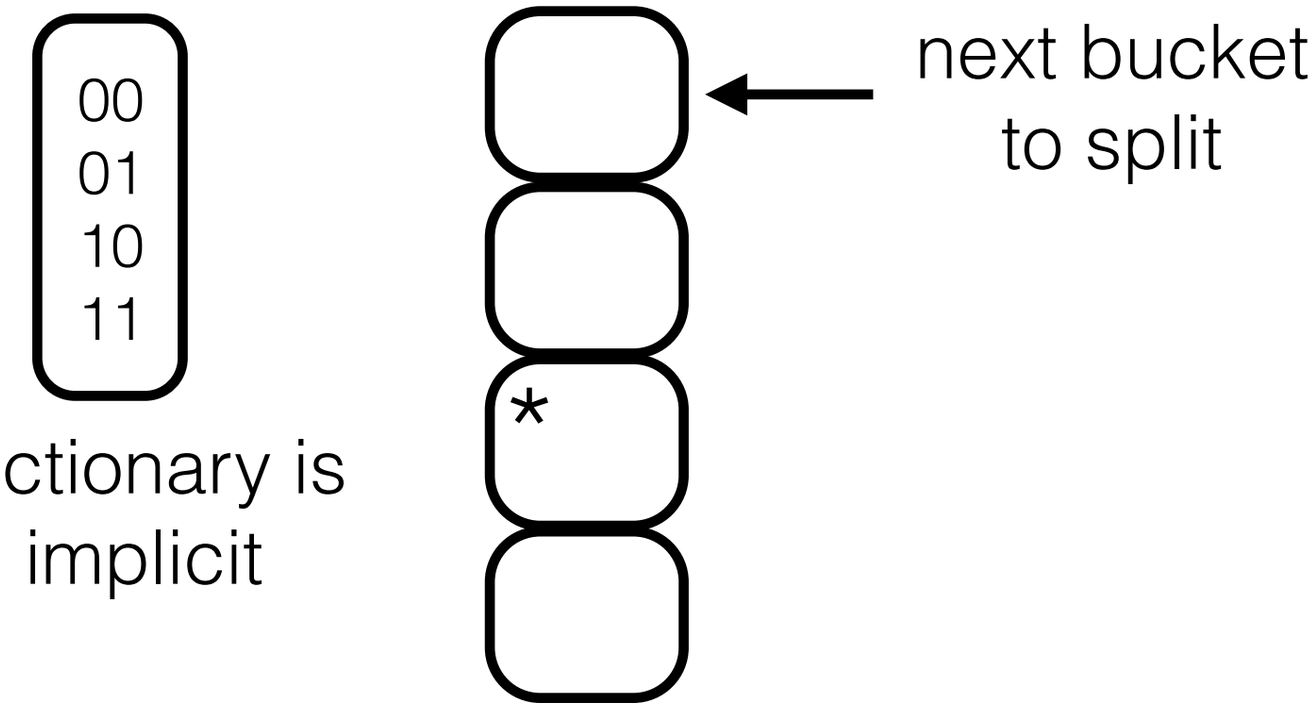


dictionary is implicit

when insert causes overflow somewhere split *next bucket*

use +1 bit of hash value

e.g.,use 2 bits of hash value (h0)

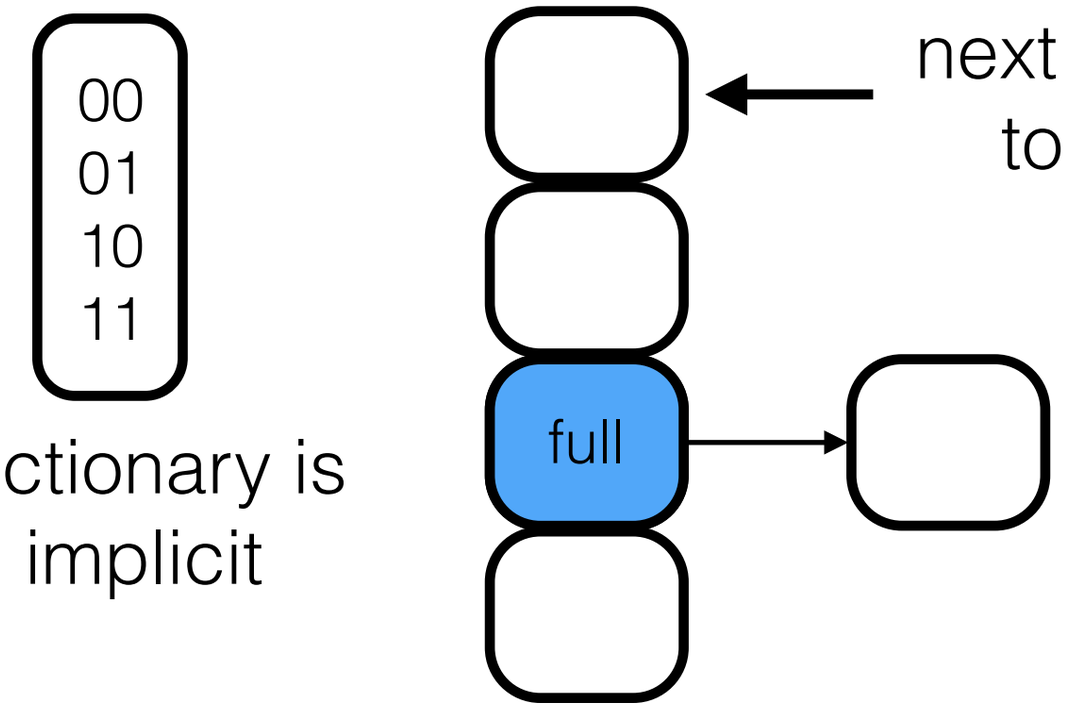


dictionary is implicit

when insert causes overflow somewhere split *next bucket*

use +1 bit of hash value

e.g.,use 2 bits of hash value (h0)



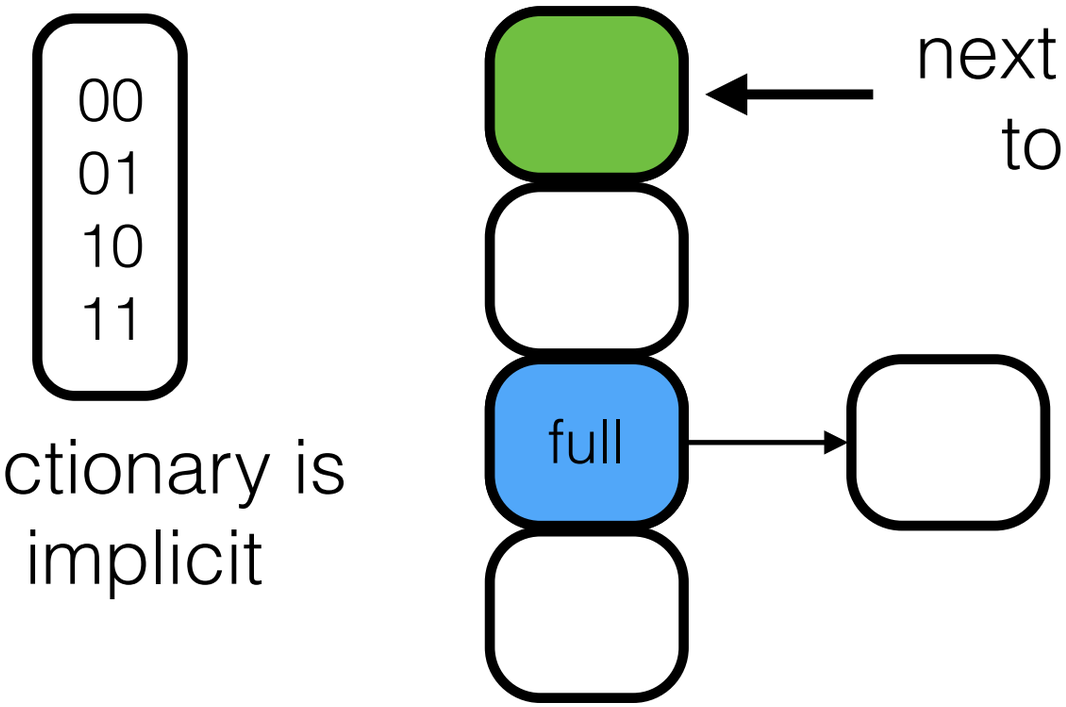
dictionary is implicit

next bucket to split

when insert causes overflow somewhere split *next bucket*

use +1 bit of hash value

e.g.,use 2 bits of hash value (h0)

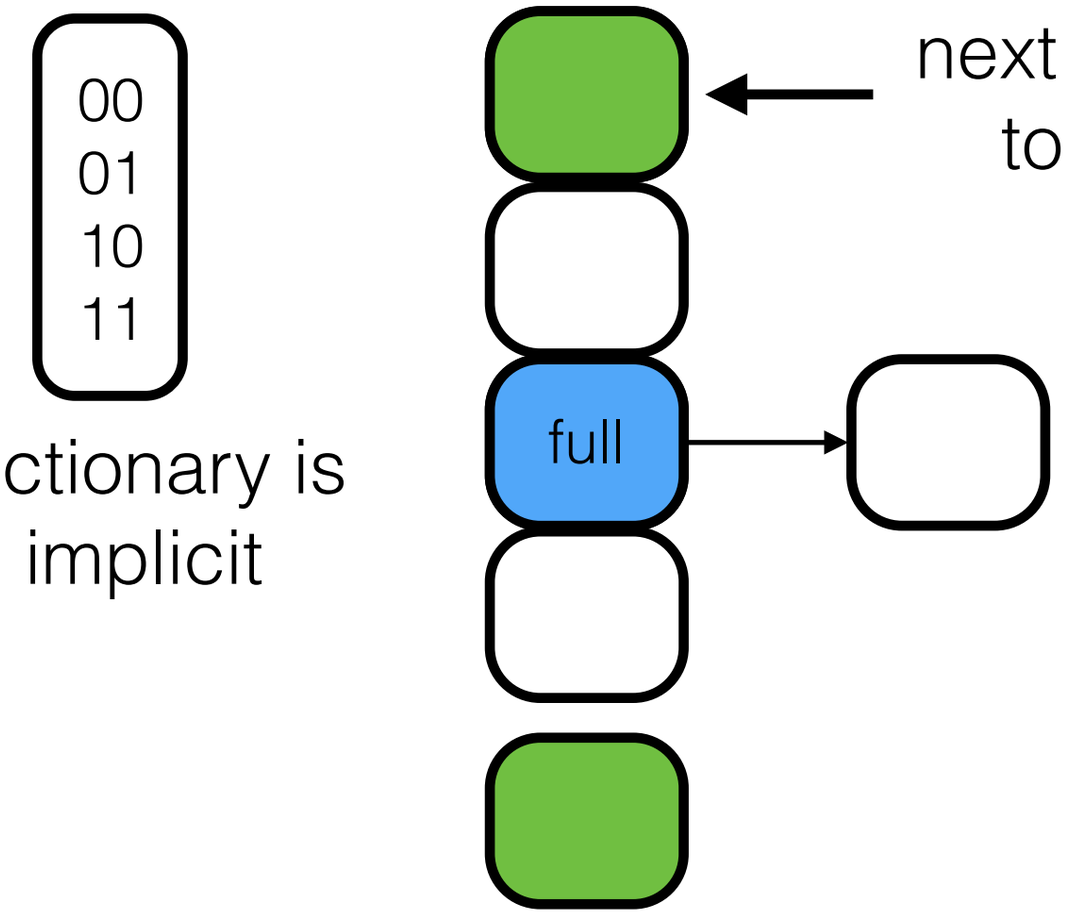


dictionary is implicit

when insert causes overflow somewhere split *next bucket*

use +1 bit of hash value

e.g.,use 2 bits of hash value (h0)

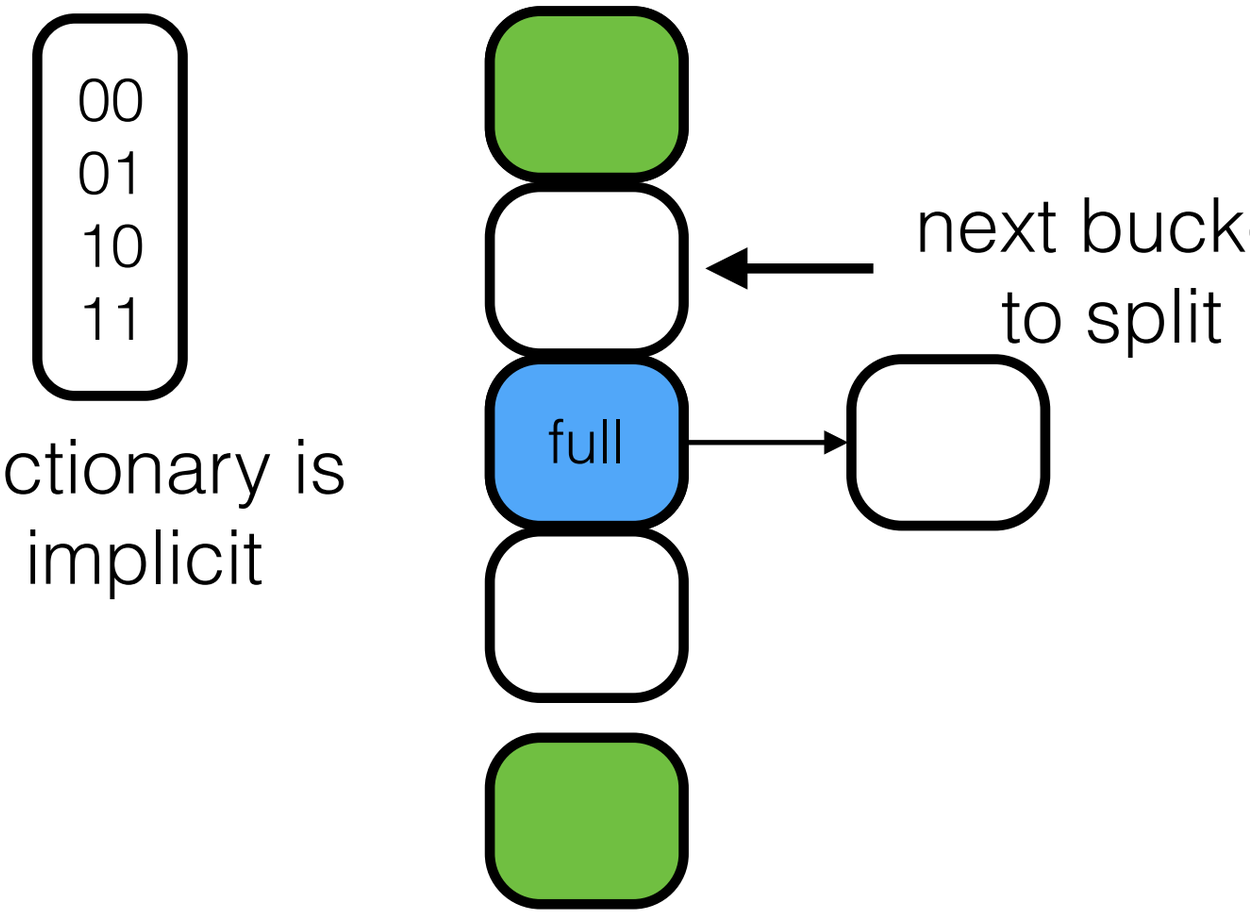


dictionary is implicit

when insert causes overflow somewhere split *next bucket*

use +1 bit of hash value

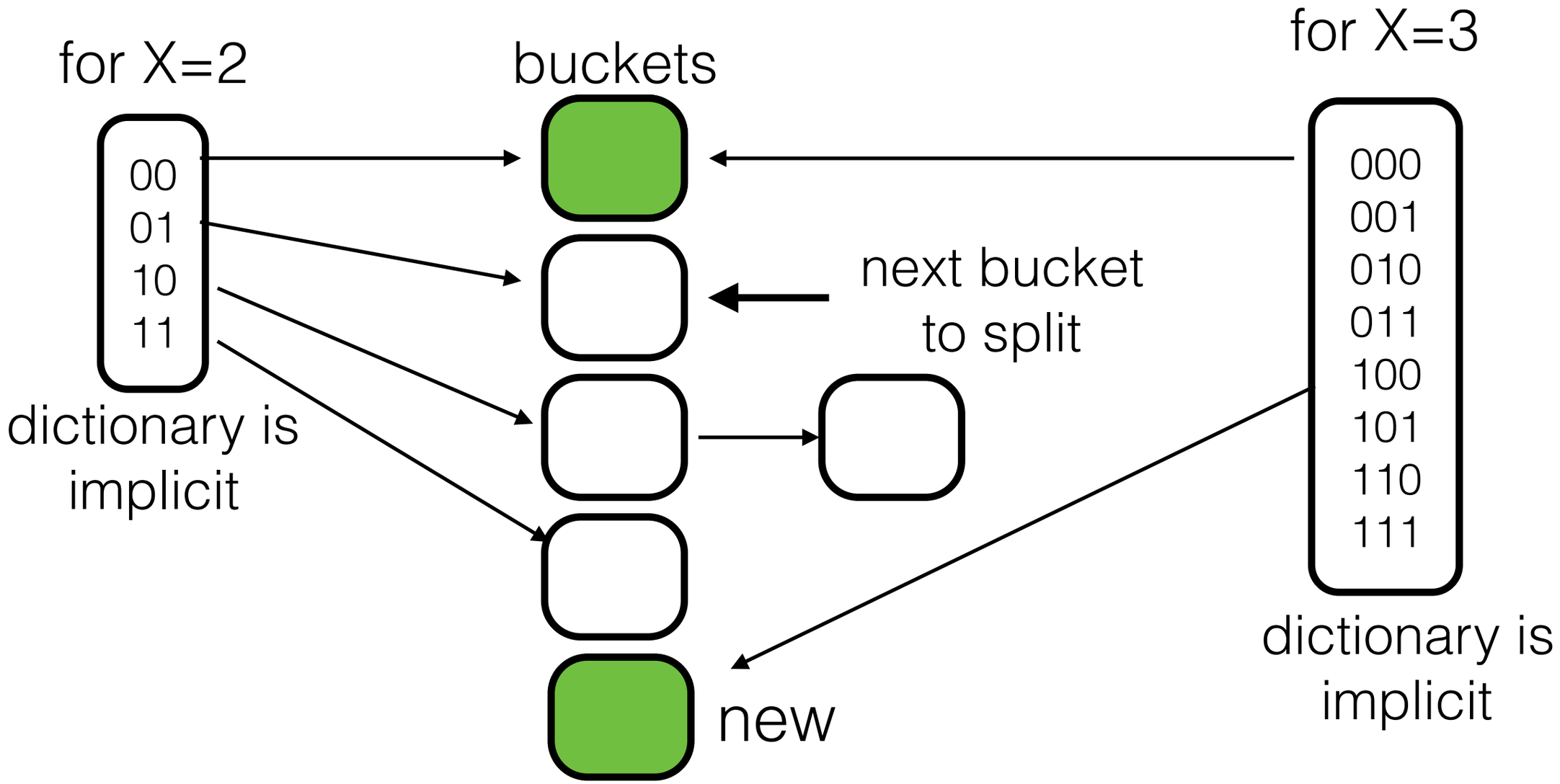
e.g.,use 2 bits of hash value (h0)



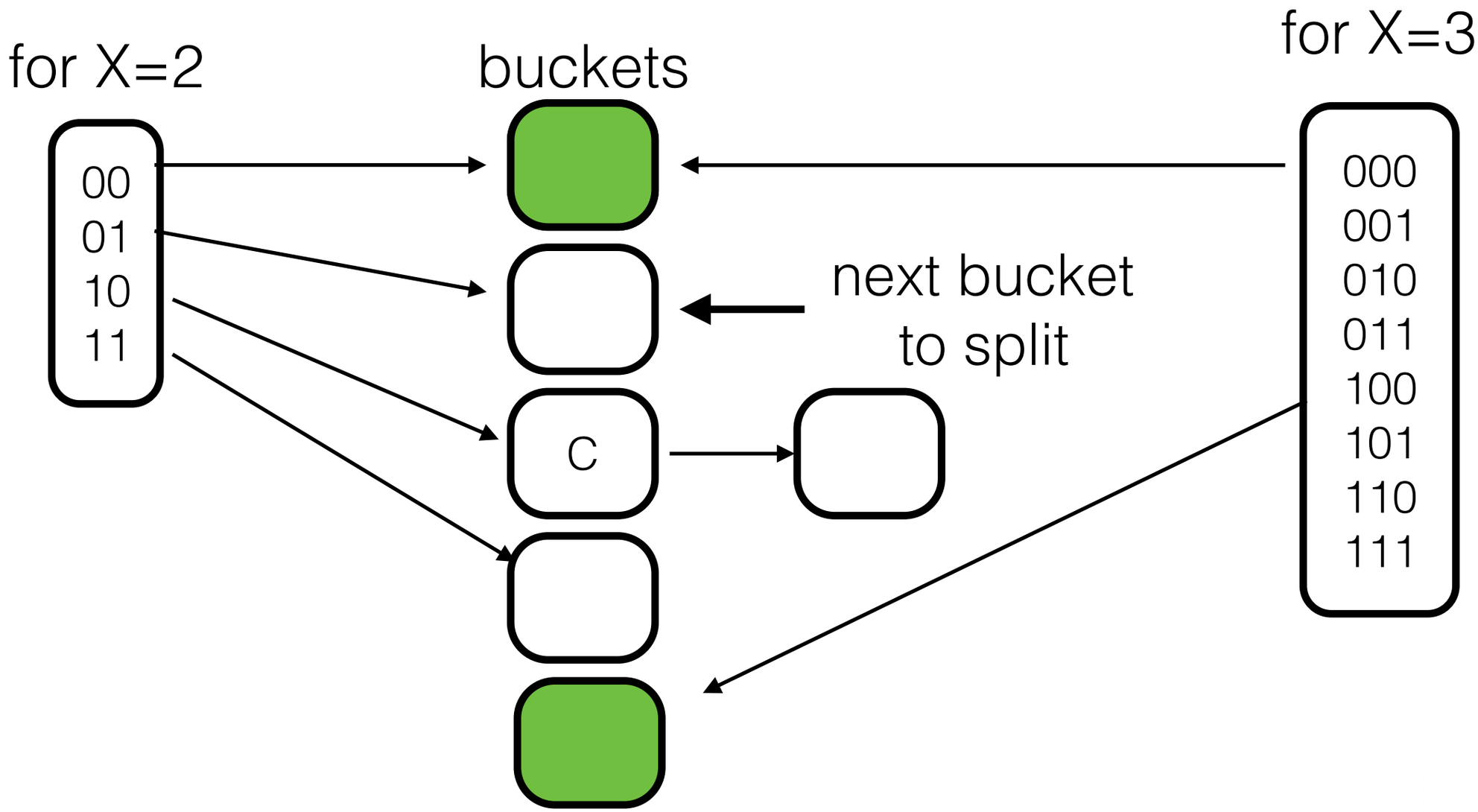
dictionary is implicit

when insert causes overflow somewhere split *next bucket*

use +1 bit of hash value

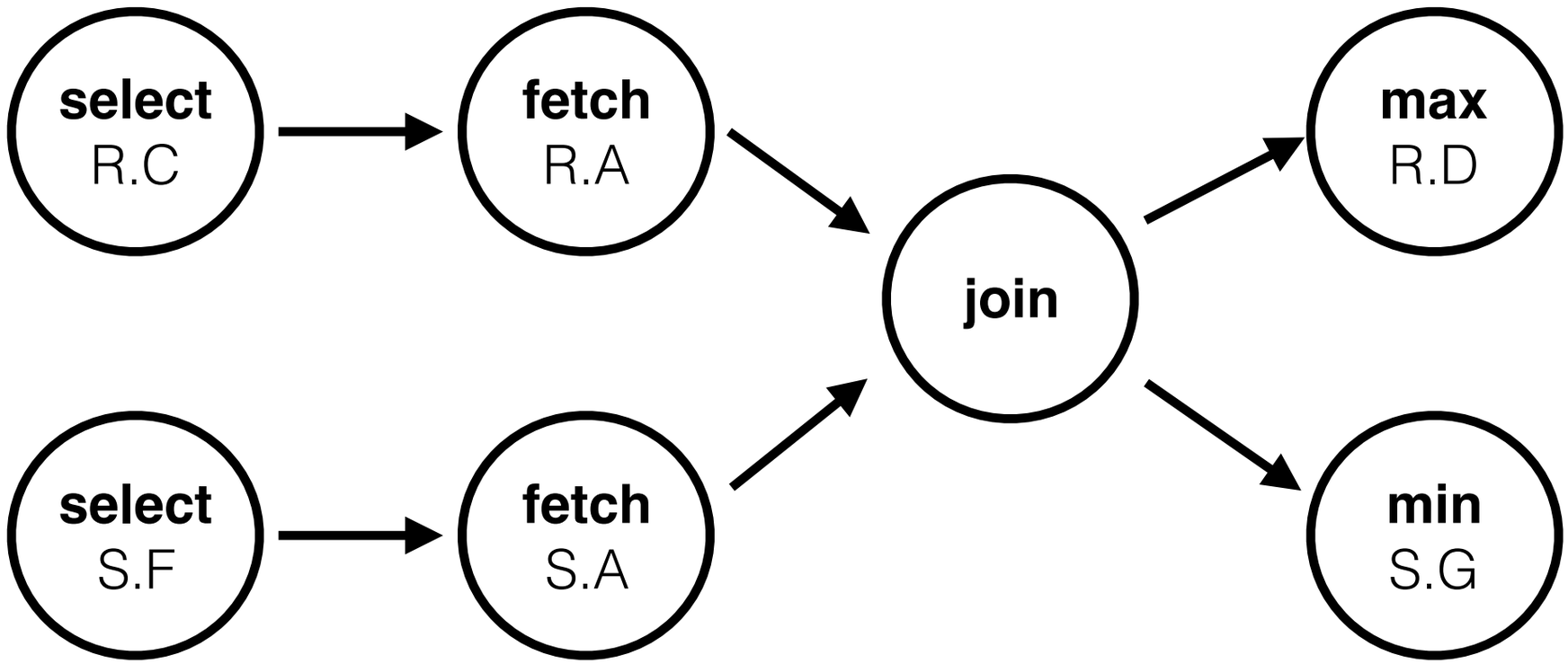


search: use current hash function (bits)
if after split bucket ok
else use next hash function (bits+1)



keep splitting next bucket with each overflow
until all buckets are split (for x=2)
then restart for x=3

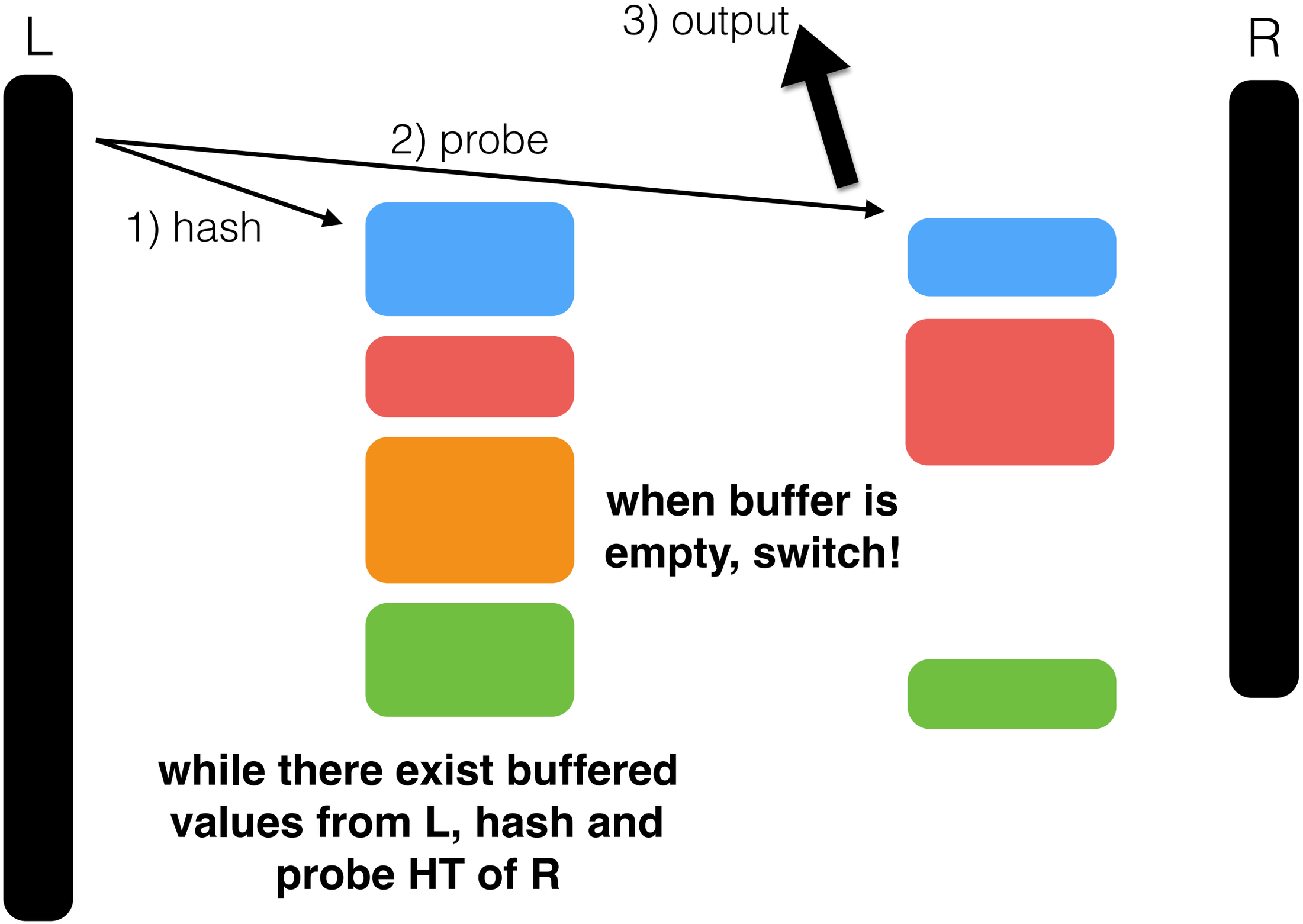
any problems
what would happen
when we split bucket C



What can we do to start working immediately?
(hint: vectorized processing)

What can we do if we wait for all data to arrive?
(hint: bulk processing)

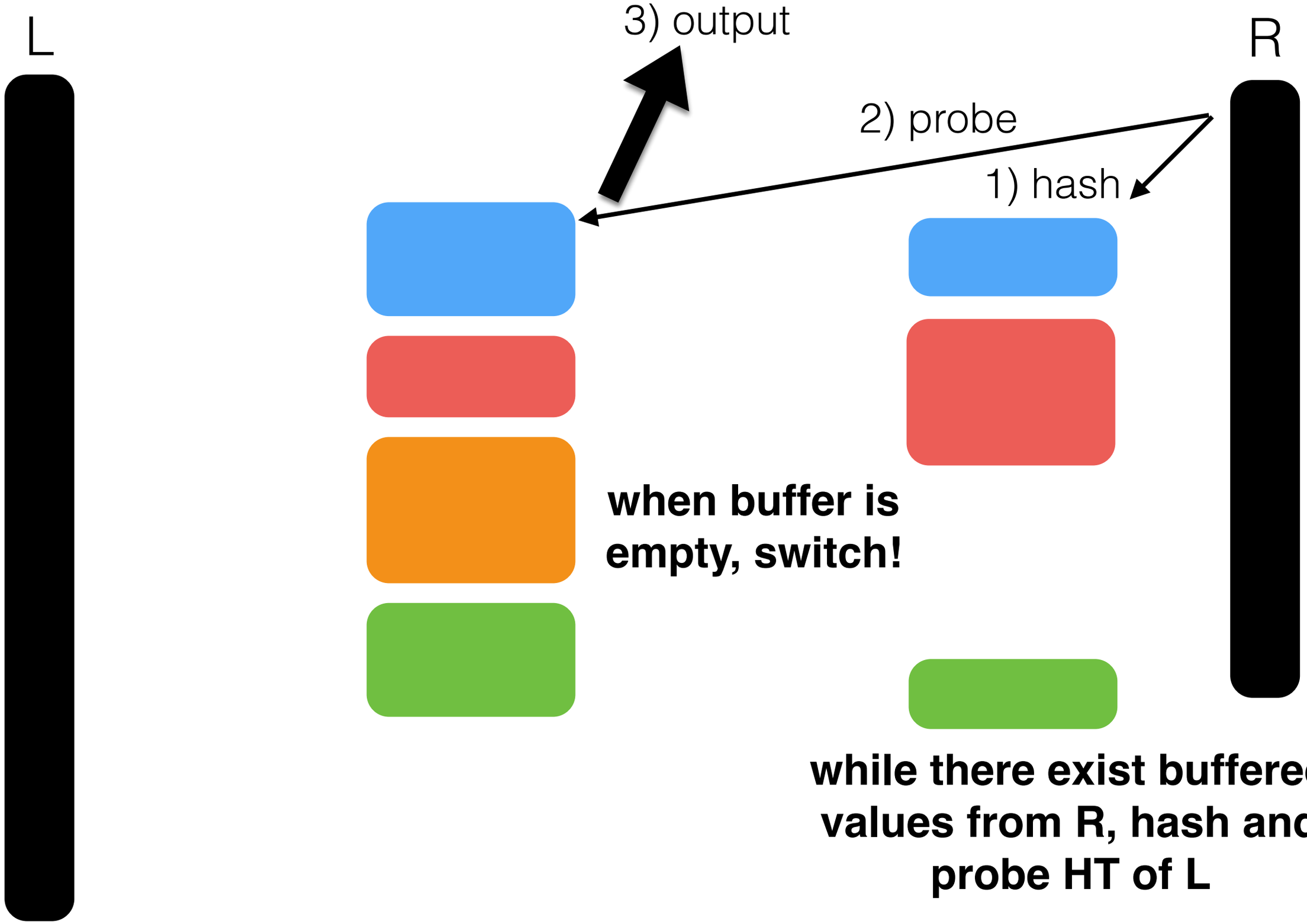
symmetric hash join



when buffer is empty, switch!

while there exist buffered values from L, hash and probe HT of R

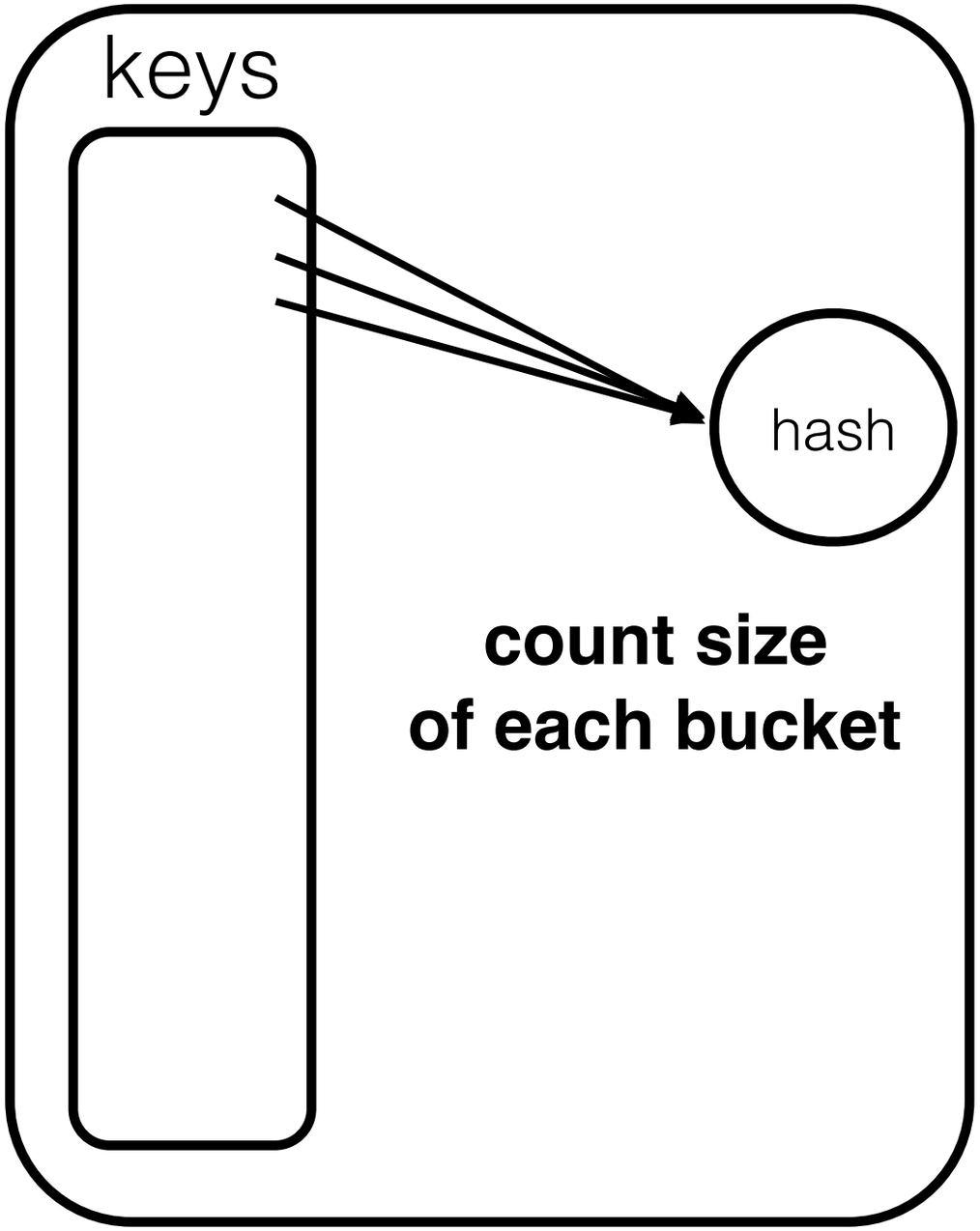
symmetric hash join

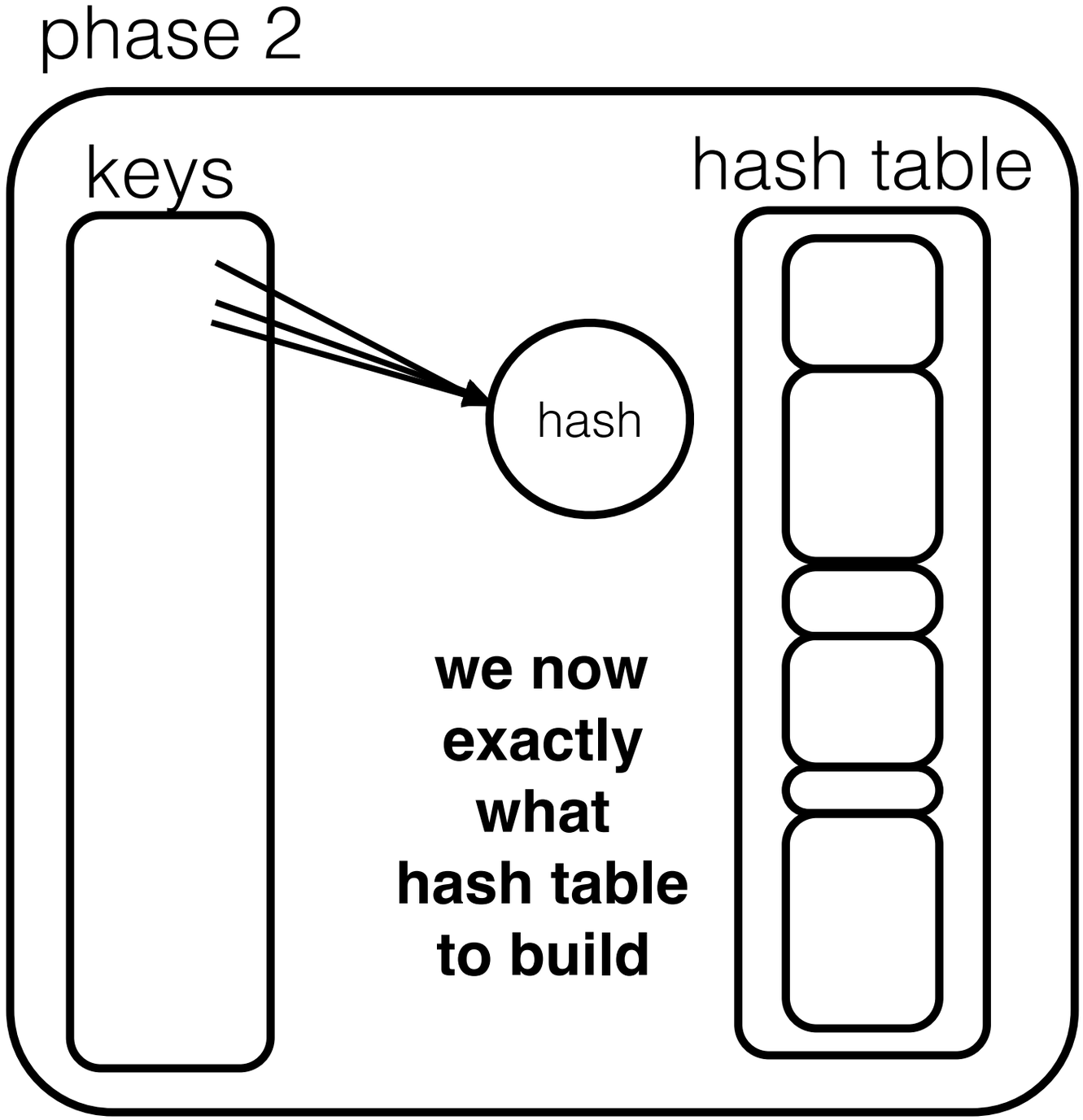


when buffer is empty, switch!

while there exist buffered values from R, hash and probe HT of L

static hashing with 2 passes





phase 1

keys

hash

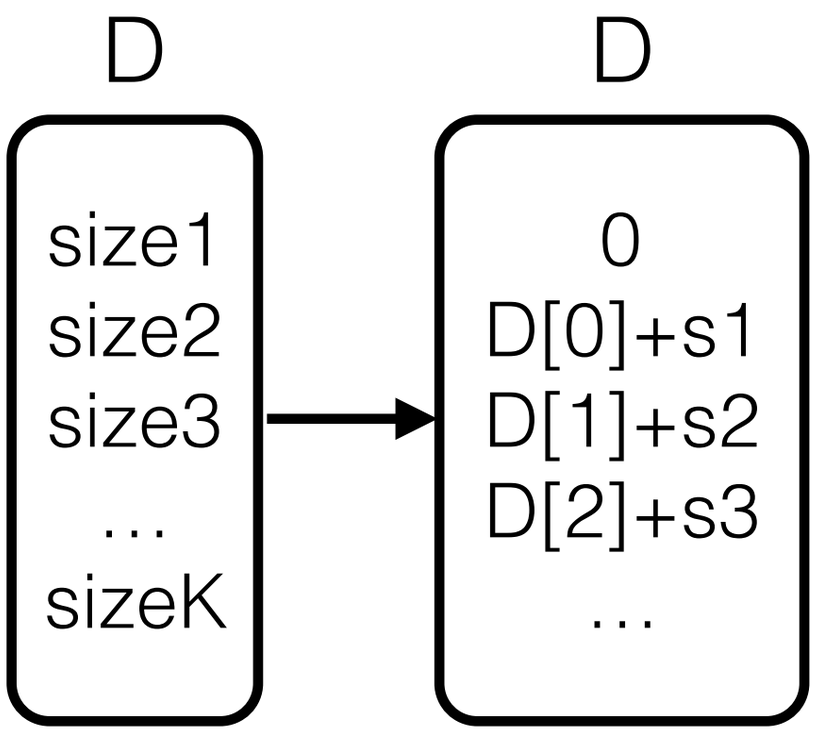
count size of each bucket

D

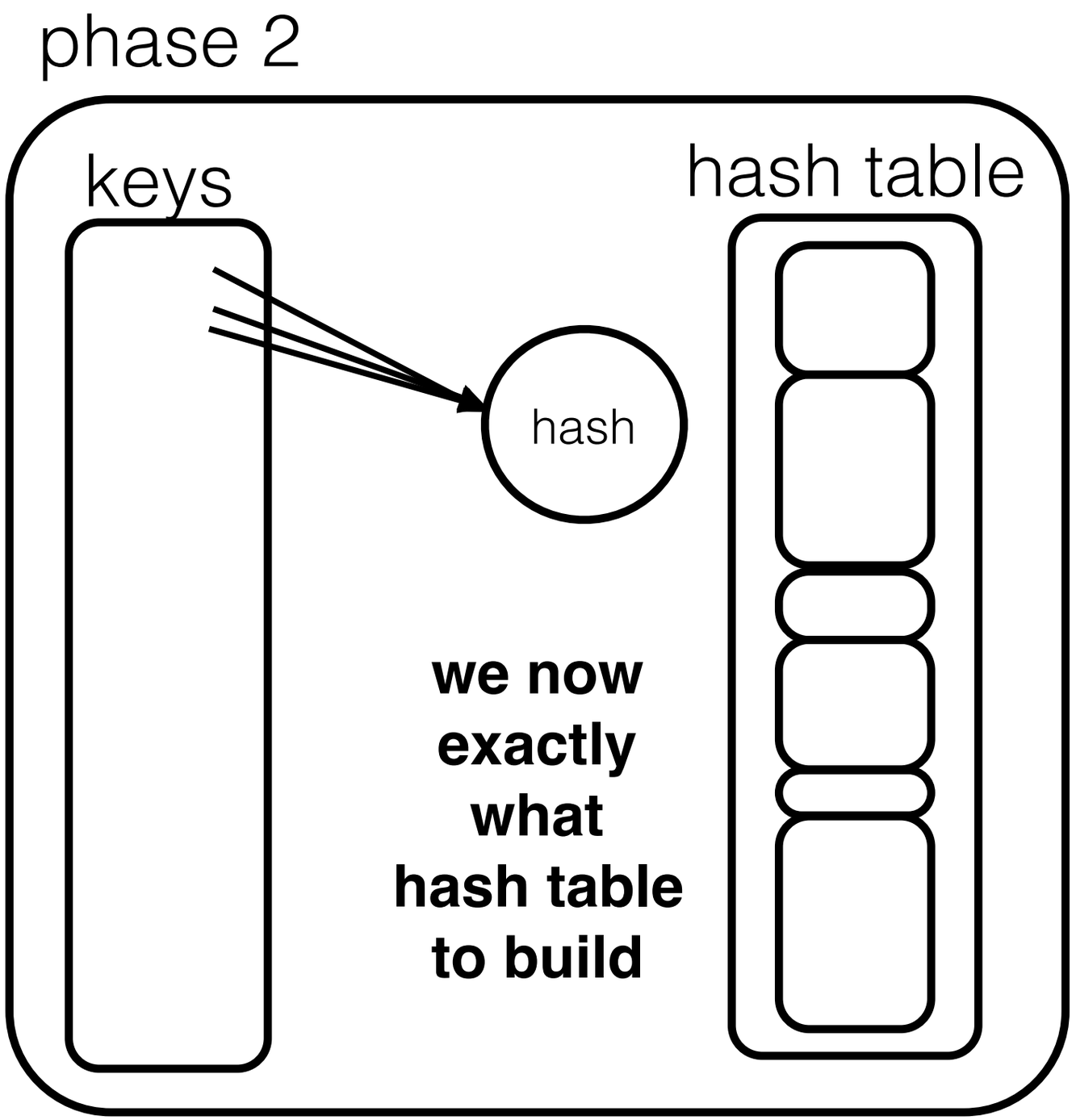
size1
size2
size3
…
sizeK

for each bucket
we know its size

array of k slots
where k the buckets
we want to have



pass D and sum all counts to get offsets in a sequentially stored hash table



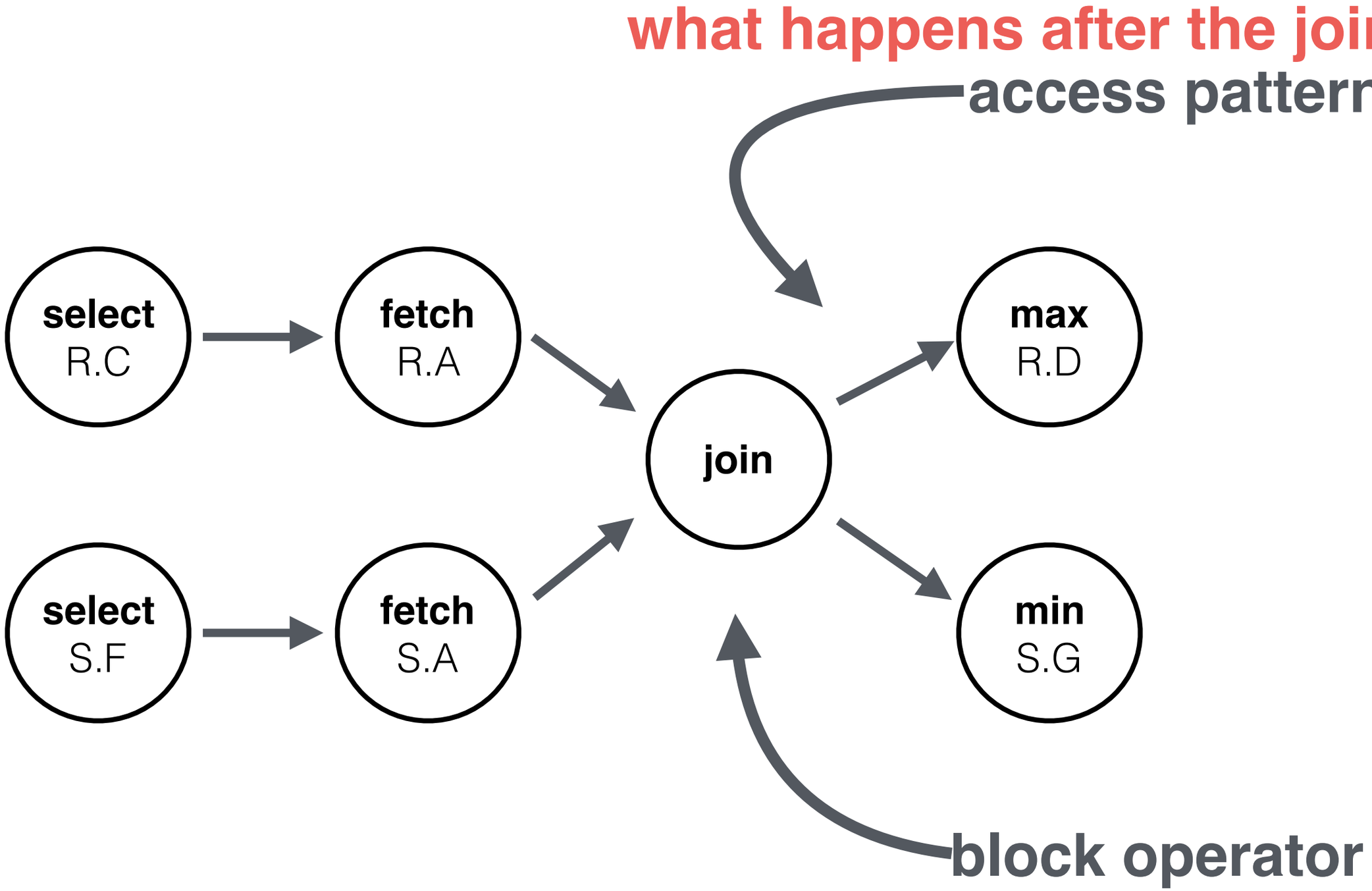
which hash function should we use?

h=a*key+b

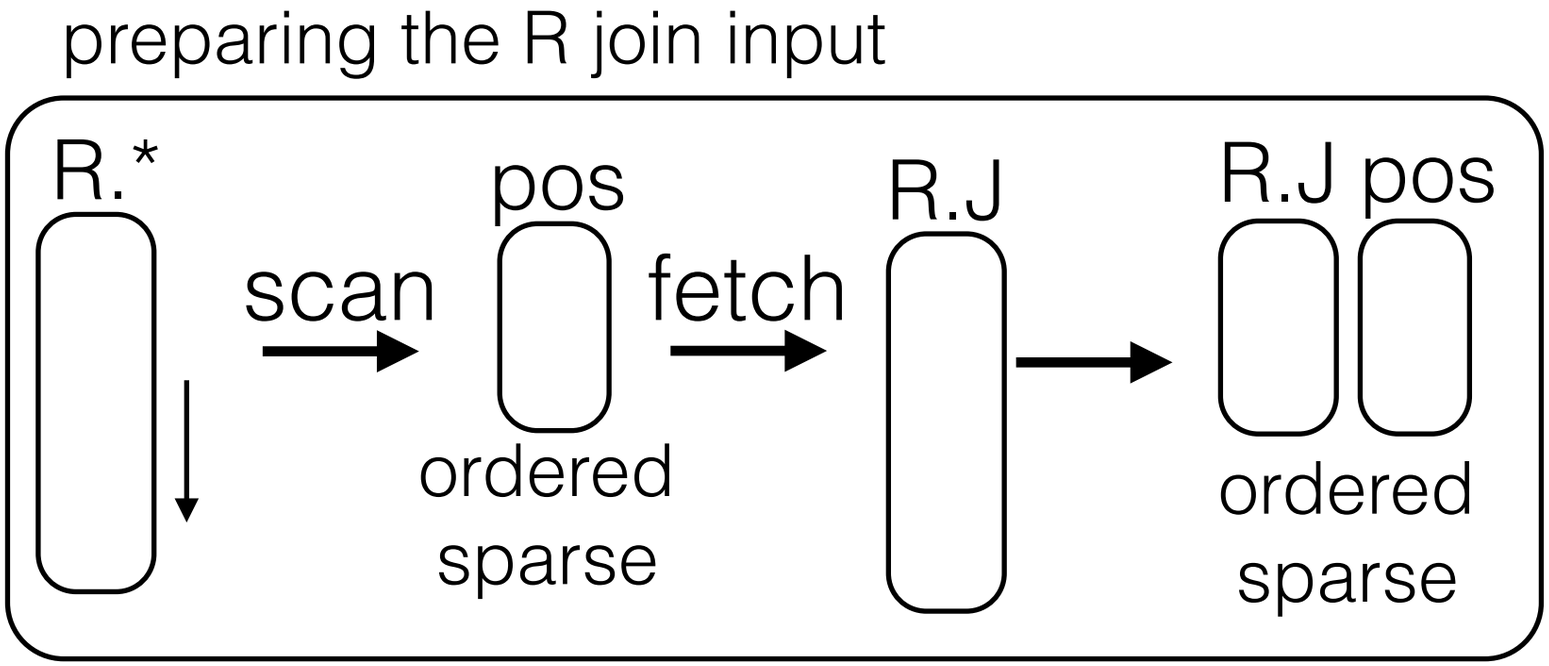
b=h mod K

can be done with binary ops only use X LSBs

select max(R.D),min(S.G)
from R,S
where R.A=S.A and R.C<10 and S.F>30



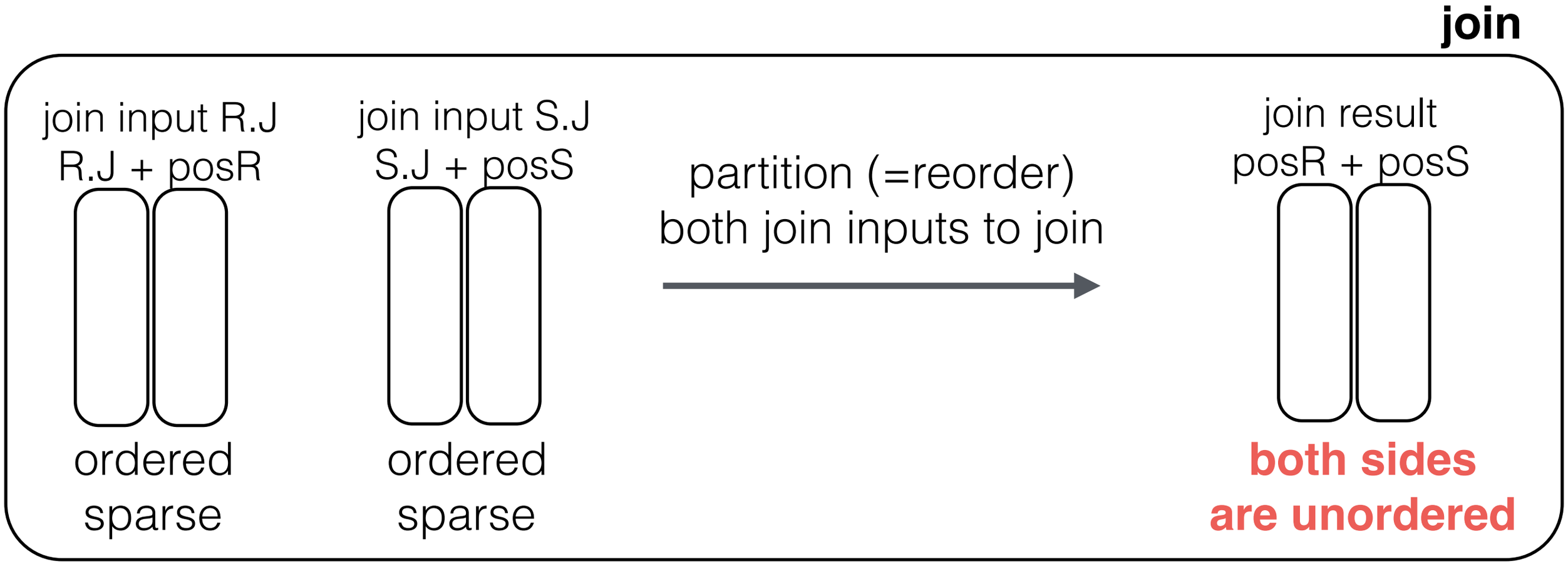
select R.A, R.B, R.C, S.A, S.B, S.C
from R, S
where R.J=S.J and …



we need the original positions
so we can fetch other R columns
after the join

same for the S join input

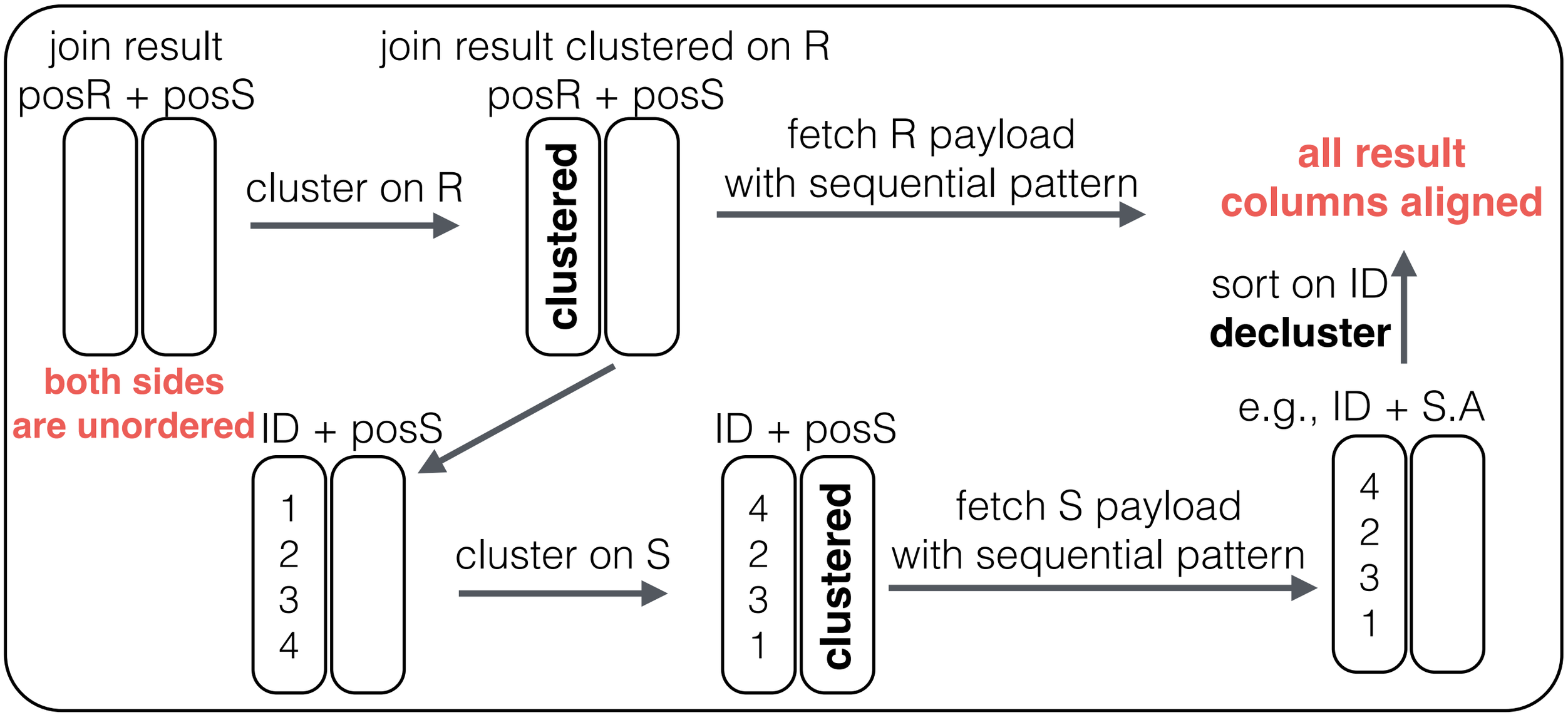
select R.A, R.B, R.C, S.A, S.B, S.C
from R, S
where R.J=S.J and …



select R.A, R.B, R.C, S.A, S.B, S.C
from R, S
where R.J=S.J and …

Cache-Conscious Radix Decluster Projections
By S. Manegold, P. Boncz, N. Nes, and M. Kersten
Very Large Databases Conference, 2004

radix declustering



Textbook Chapter 11

Cache-Conscious Radix Decluster Projections

S. Manegold, P. Boncz, N. Nes, and M. Kersten

Very Large Databases Conference, 2004

hashing

DATA SYSTEMS

prof. Stratos Idreos