# CSE 350

## Advanced Data Structures

### Topic 10: Hash-Based Storage 📦🥘#️⃣

# Review: Modular Arithmetic

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

## Review: Hash Functions

Examples
- SHA256
- MD5
- BCrypt
- MurmurHash3

$h(x)$ → Pseudorandom, uniform over $[0, Max]$
→ Deterministic: $h(x) = h(x) = h(x)$
→ Statistically independent $h(x) \neq h(x')$ for $x \neq x'$

↑ hash function

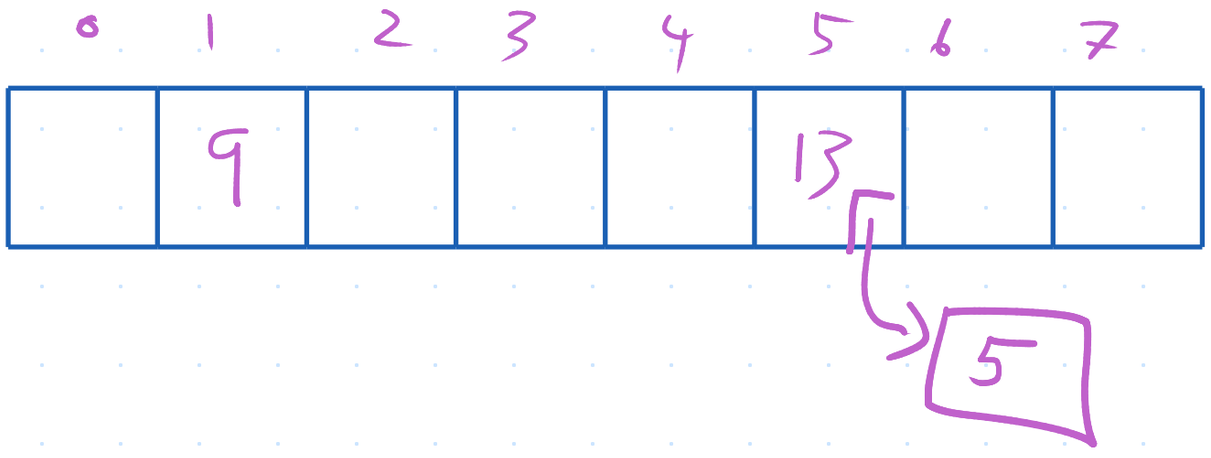
↑ any sequence of bits

$h(x) \bmod N$ "$h(x)\%N$" has the same properties (if $N \ll Max$)

# Review: Hash Tables

ex: $h(x) = x$

$h(x) \% 8$

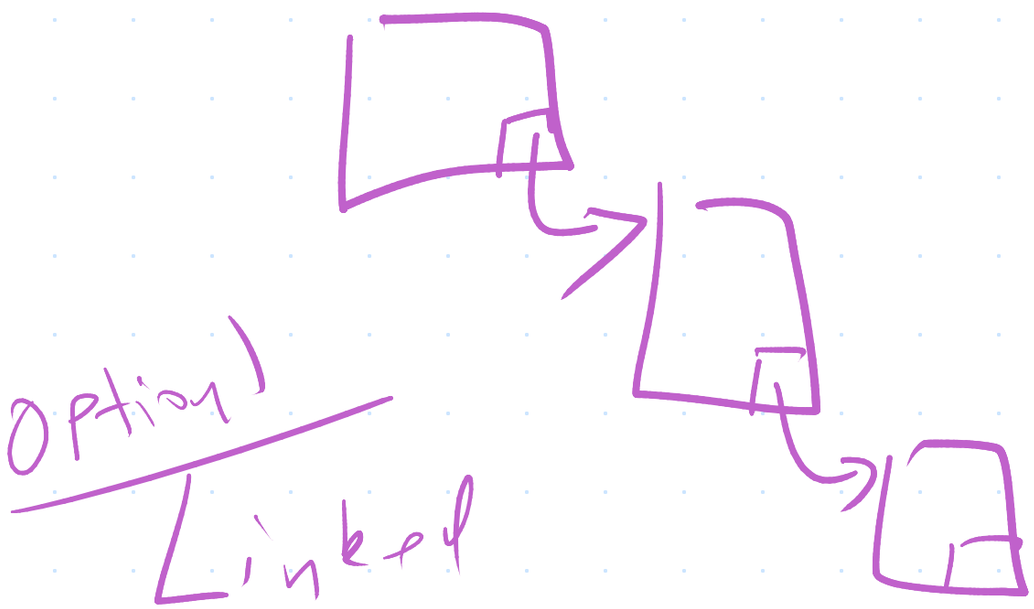



## Expected number of items in each bucket

N = Number of Records
B = Number of Buckets

$$E[\text{records}] = \frac{N}{B}$$

# What happens if the buckets overflow?



Option 1

Linked List

Option 2

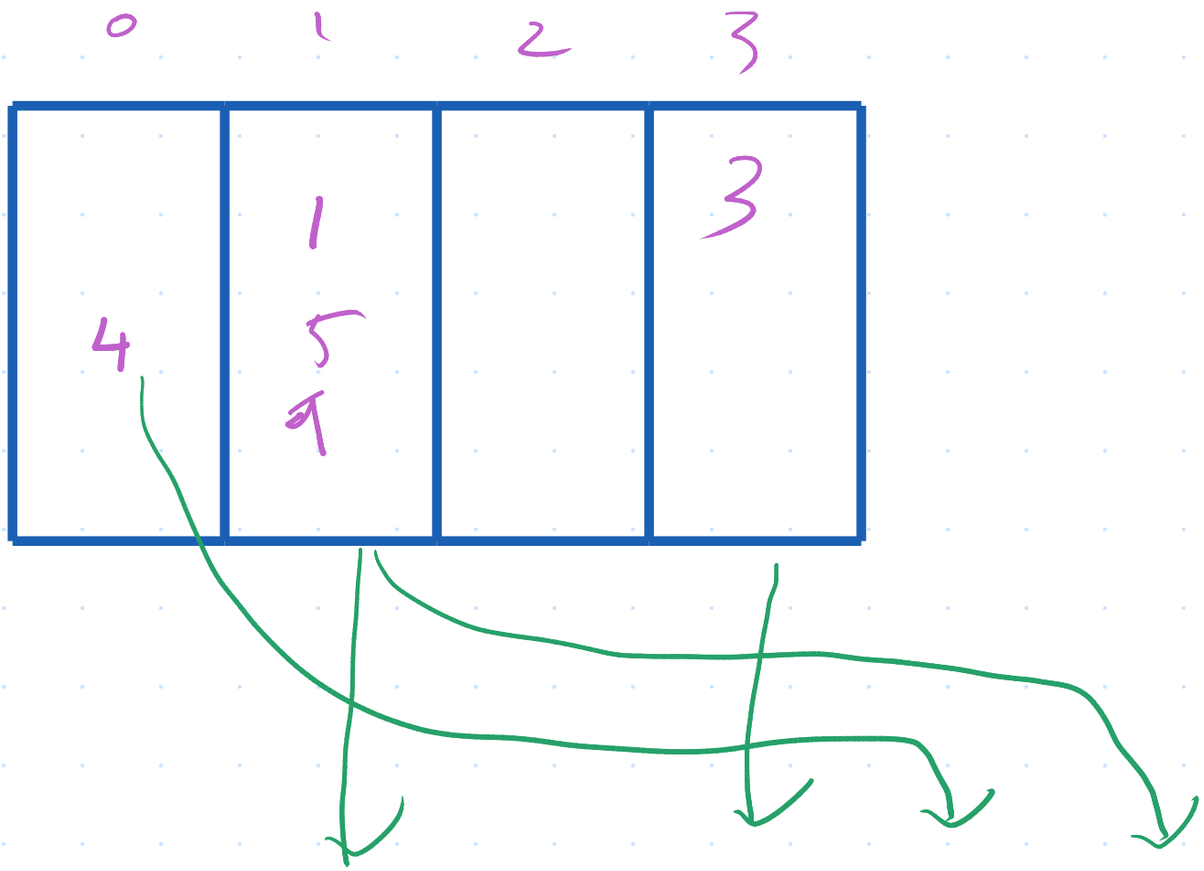
Resize to 2x original size

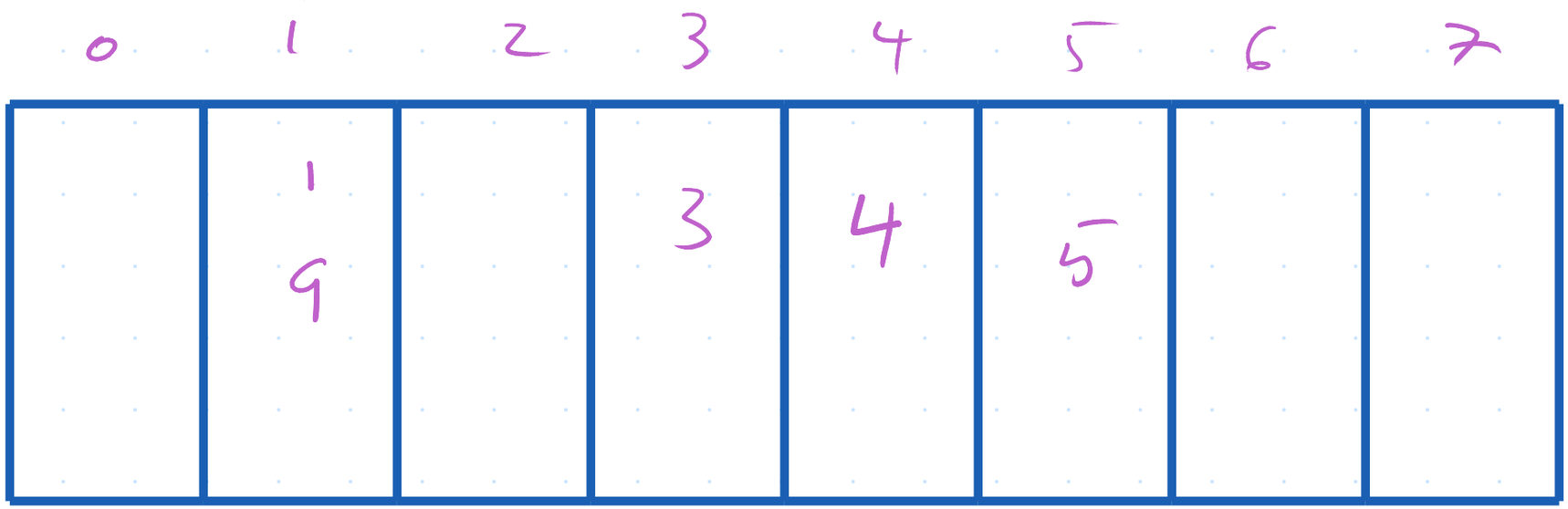
(Amortizes to O(1) per insert)

## What changes when we store buckets on disk?

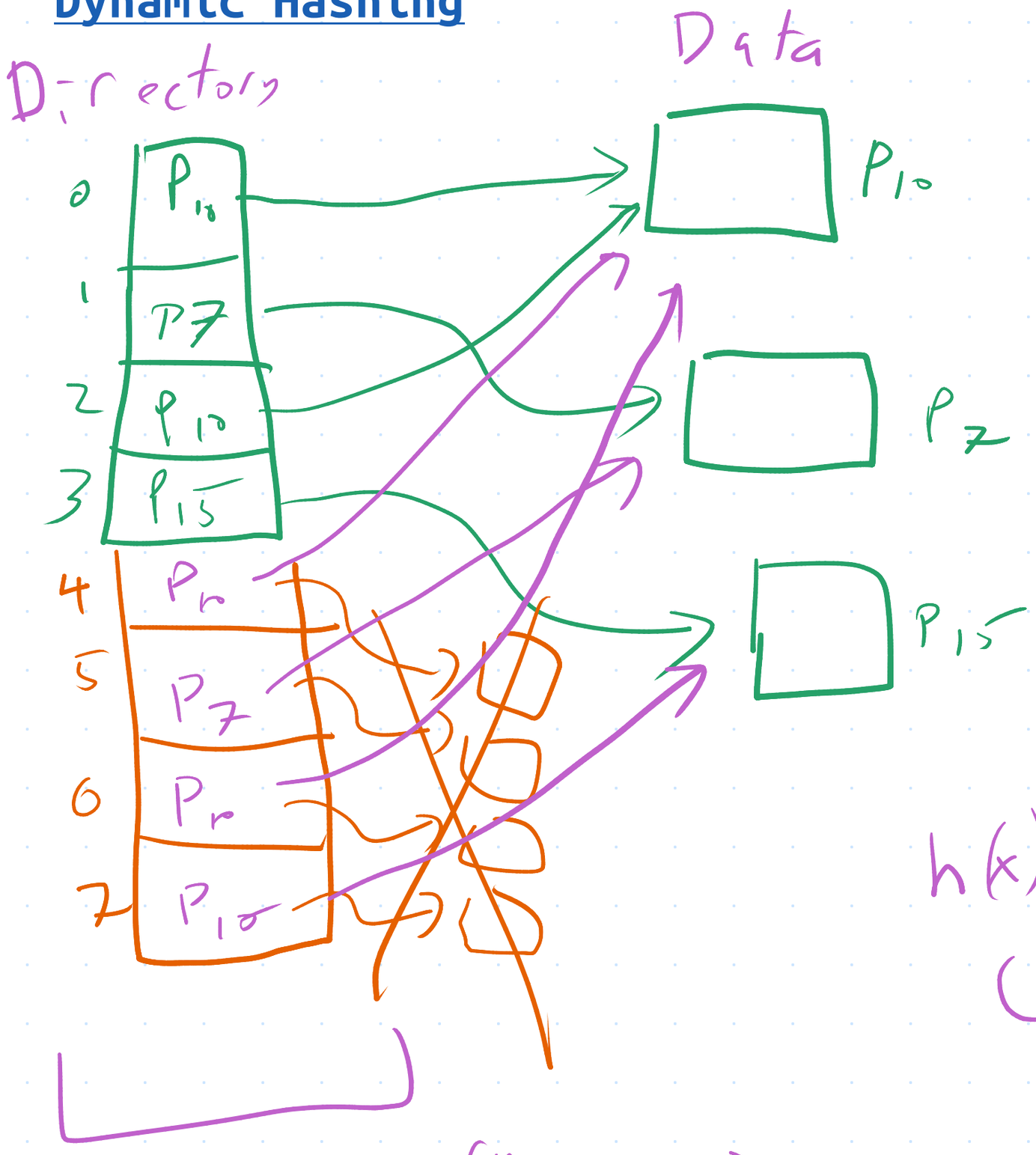
- We like spatial locality: Make $\frac{N}{B} \approx P$
  We want ~P records per bucket
  still one IO to fetch all
- Resizing Becomes Suuuper Extensive
  - Read Entire Hash Table & Write it Back

# Resizing the Hash Table



# Dynamic Hashing




To grow directory,
append a copy
of the existing
directory

$h(x) \% N = i$ → $h(x) \% 2N = i$

$h(x) \% N = i$ → $h(x) \% 2N = i + N$

↳ Each data page goes to one of two new data pages

# Extras

Track - directory depth (D) → Directory has $2^D$ entries

- data page depth (d) → Data page stores records that hash to $h(x)\%2^d = i$

If data page fills up

↳ d = D?

↳ Double directory size

↳ create new page for $h(x) = i + 2^d$ @ depth d+1

↳ increase page depth to d+1

↳ re hash the page

Con

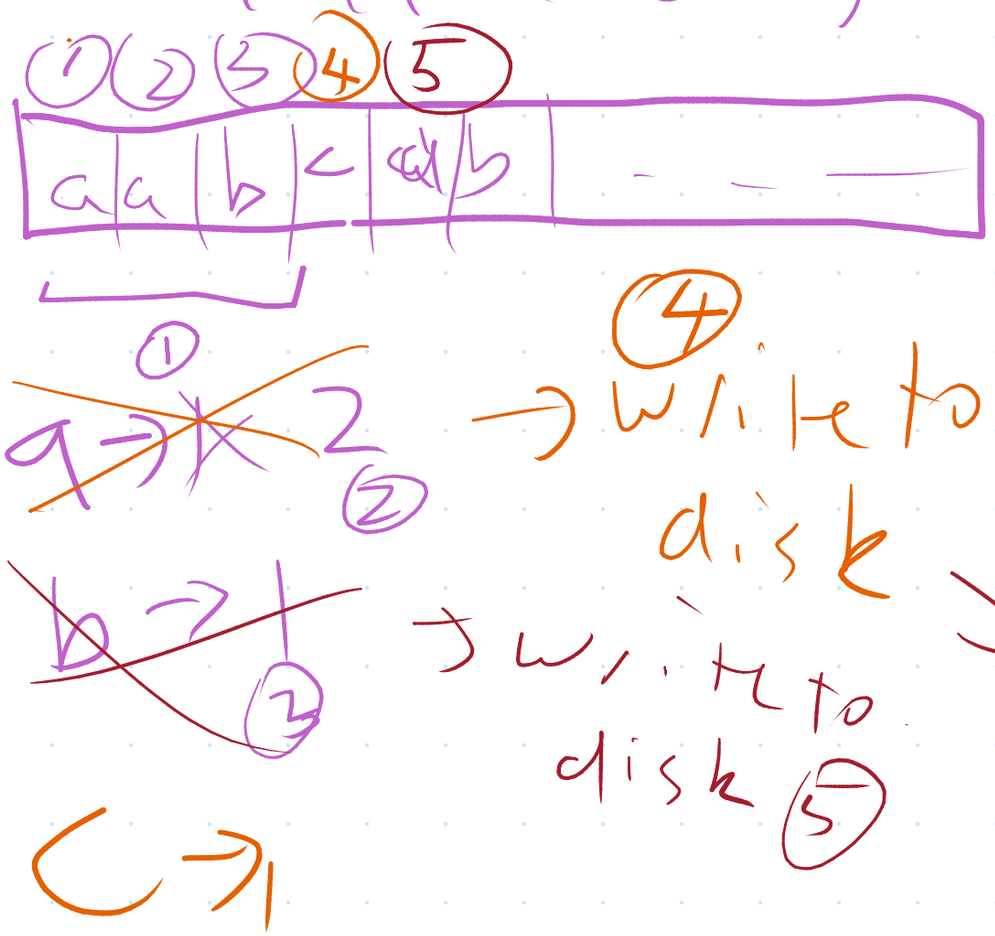
- More complicated
- Less spatial locality

Pro

- No "big" resize events
  - work more spread out
  - more resilient to bad high fps

# Hash Partitioning: Aggregation

group-by aggregation → Need memory to hold each group

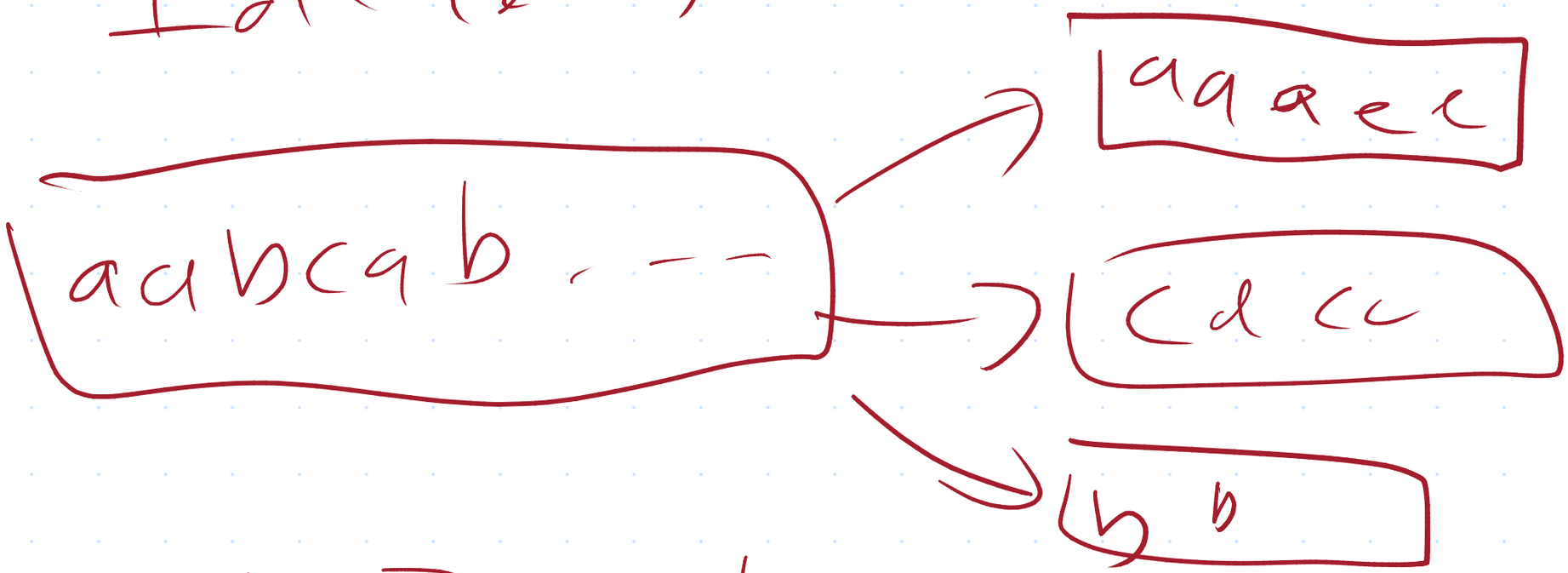



Example

2 groups
3 records
] memory size

If not enough space for all groups, you (usually) get random access

Idea: use hash to partition




Pick B so that each partition is very likely to fit in memory!

1. Read input a little at a time
(O(N) I Os)

2. Keep one file open per partition
append records to the right file
(O(N) I Os)

3. Reread each partition file
one at a time
O(N) IOs

~3N IOs