SELECT ——————

```
CREATE TABLE (
    Studio  string
    Title   string
    Lang
    Director
    Star
    Rating Int
);
```

A	B	C	D
1	2	3	4
5	6	7	8
⋮			

Movie	Studio	Title	Lang	Director	Star	Rating

- "Tidy Data" (Normal Forms)

Movies	Title	Lang	Runtime
	Princess Bride	English	114
	Pinncess Bride	Spanish	114
		English	~~108~~ 126
	Die Hard	English	128
	Die Hard	Polish	128
	Matrix	English	112

Movies	Title	Lang	Runtime
	PrincessBride	[Eng, Spa]	114
	Die Hard	[Eng, Pol]	126
	Matrix	[Eng]	112

document database

Lang	Title
Eng	PB
Spa	PB
Eng	DH
Pol	DH
Eng	M

Runtime	Title
114	PB
126	DH
112	M

Title	Runtime
PB	114
DH	126
M	112

Functional Dependency (FD)

R (A B C D E — —)

{A, B} → {D, E}

If I know A and B, I can uniquely obtain D, E

{Title} → {Runtime}

~~{Title} → {Lang}~~

{A} → {D E}

↓ implies

{AB} → {DE}

~~{A} → {D, E}~~

↓ implies

{A} → {D}

Rules For FDs

{A} → {D, E}

↓

{A} → {A, D, E}

$$R(A_1, A_2, A_3 \ldots, B_1, B_2, B_3 \ldots)$$

$$\{A_1, A_2, \ldots\} \rightarrow \{A_1, A_2, \ldots, B_1, B_2, B_3\}$$

(Super) Key

If I know these

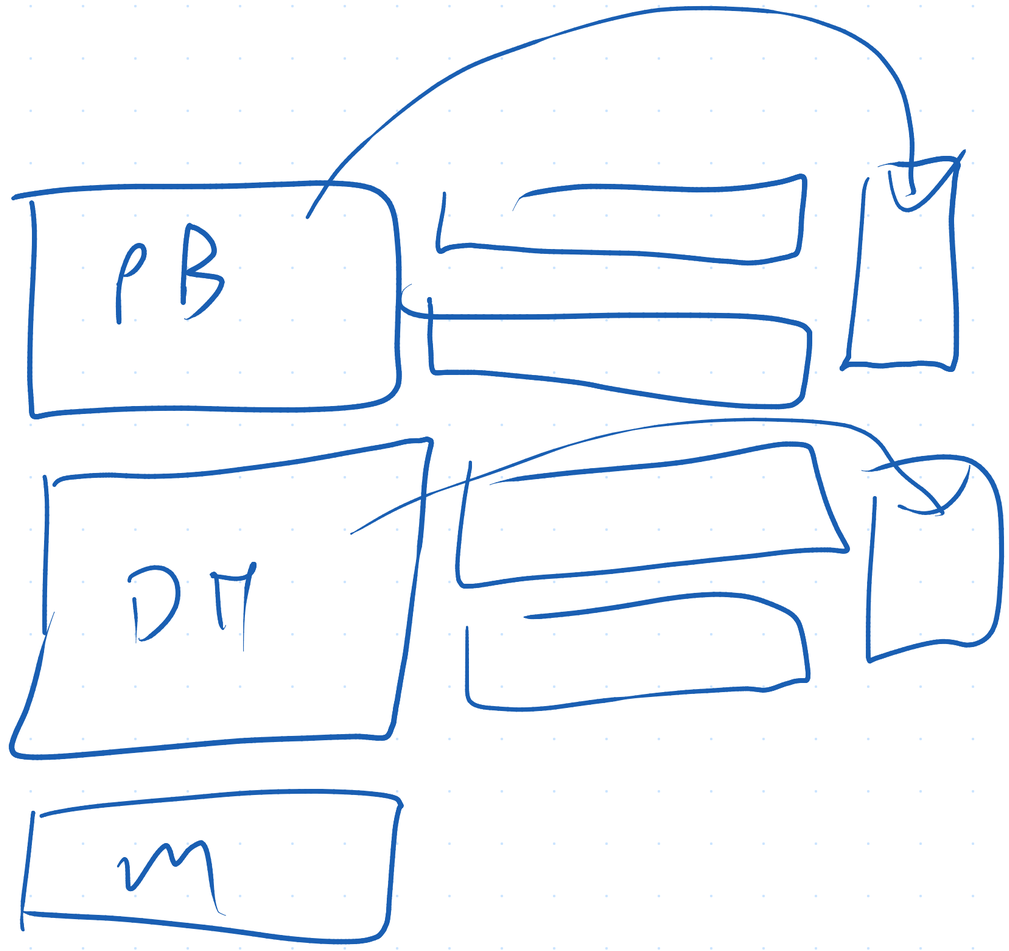
I can uniquely Id the entire record

Title, Lang] key

vs

Title, Lang, Runtime] superset of a minimal key

↖



FD from

{Title} → {Runtime}

↓

Not all attrs in table

CHS is not a key (or superkey)

↗

2nd Normal form Property

Datasets

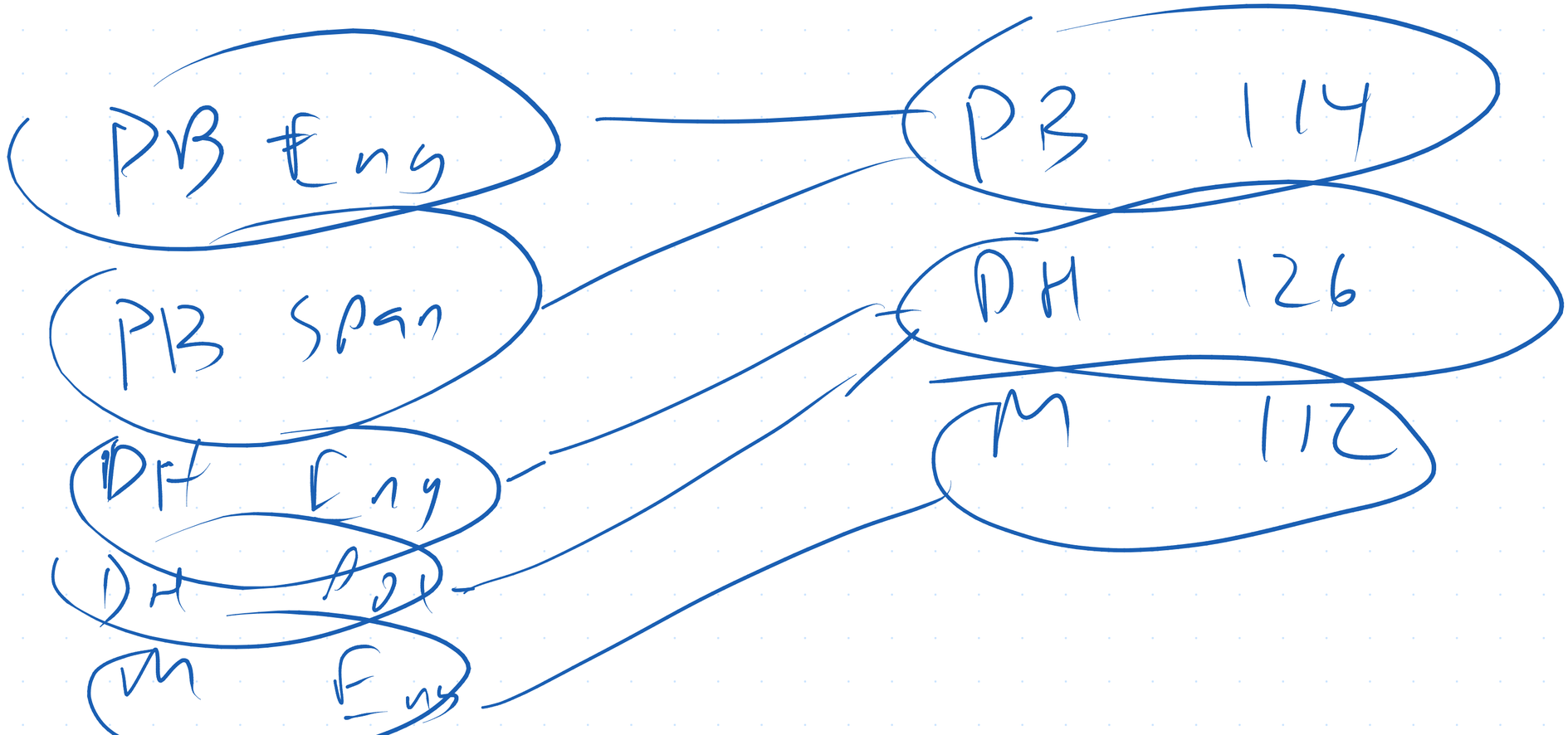
- 1 Normal Form → No nesting
  No lists, etc —
- 2nd Normal form → Any FD* must
  have a superkey
  on the left

Lang

NATURAL JOIN

Runtime

Money

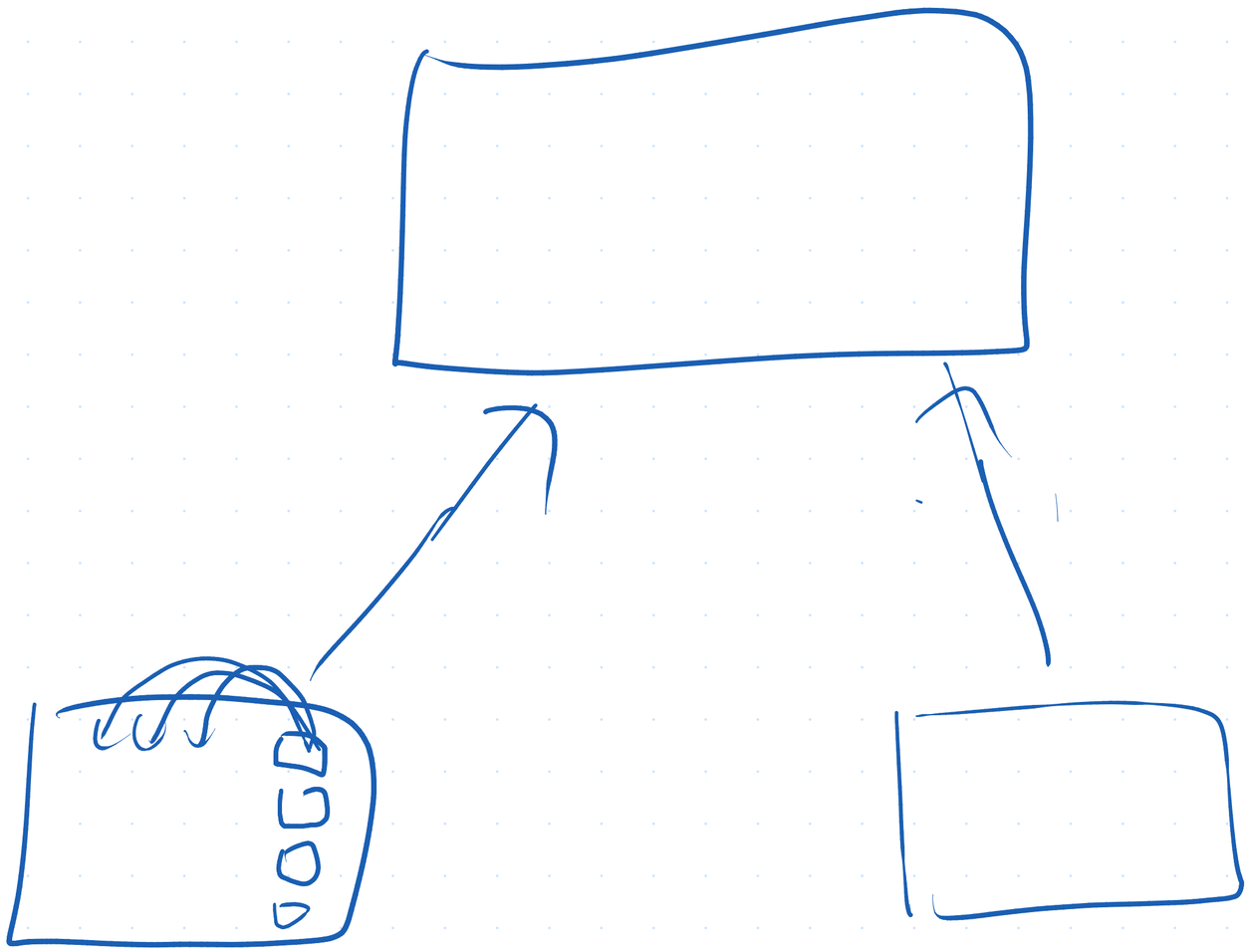


A Join B ON A_1, A_2, A_3 ...

↳ For each record in A pair it up with Every record B that has matching values of A_1, A_2, A_3 ...

NATURAL JOIN

↳ All common Attrs



Runtime	Title Runtime

Language	Title Language


Runtime
114
Language
English
Spanish

Cartesian Product

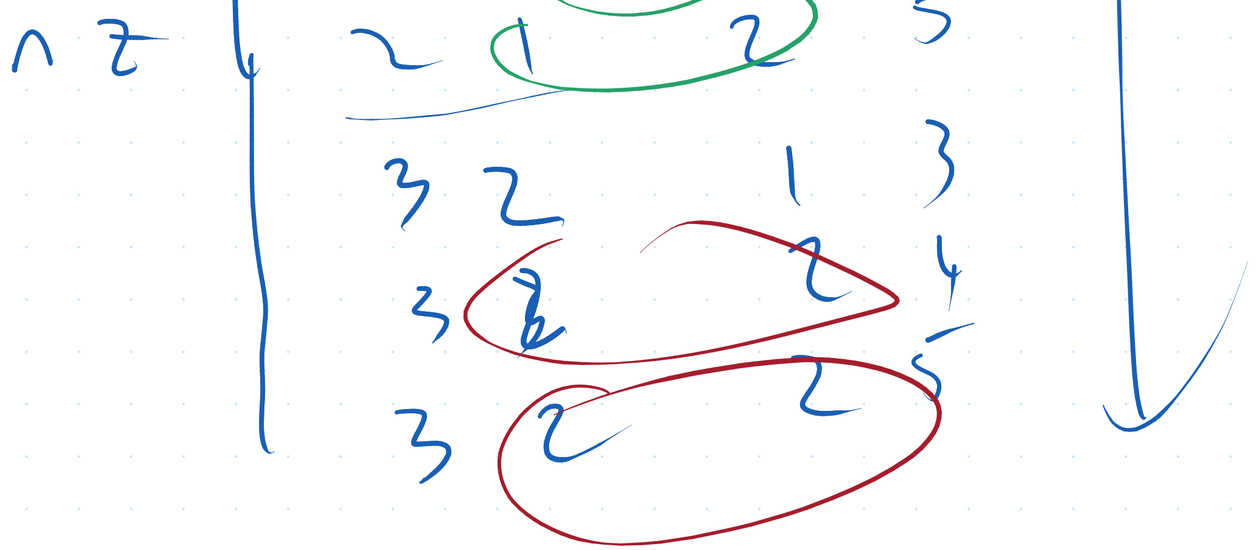
R	A	B
m	1	1
n	2	1
o	3	2

S	B	C
x	1	3
y	2	4
z	2	5

$R.B = S.B$

$R.B \subseteq S.B$

R x S	A	R.B	S.B	C
m x	1	1	1	3
m y	1	1	2	4
m z	1	1	2	5
n x	2	1	1	3
n y	2	1	2	4
n z	2	1	2	5
o x	3	2	1	3
o y	3	2	2	4
o z	3	2	2	5



R X S

↓

filter

↓

join ← Equijoin R.B = S.B

Natural join

attrs(R) ∩ attrs(S)

Movies = Runtime ⋈ Languages

≡ SELECT Title, Runtime, Language
From Runtimes NATURAL JOIN
Languages

≡ SELECT Title ——
FROM Runtimes, Languages
WHERE Runtimes.title = lang.Tit

filter (R x S)

RAS

(A B) · (

C·A + C·B

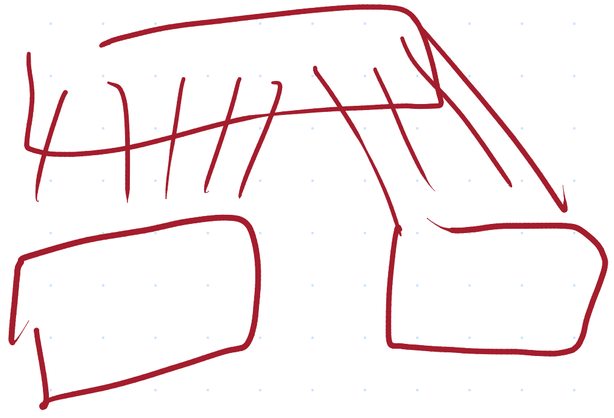
|R|

|S|

compute R x S ?

↳ O(|R| x |S|)

runtime complexity



R ⋈ S

filter on θ

for r in R:

for s in S:

|R| times

if θ(r, s):

output (r ∘ s)

Records per Page

→ ⌈ |R| / p_r ⌉ records from R

→ ⌈ |S| / p ⌉ |R| records from S

Runtime $= O(|R| \cdot |S|)$

I/O $= O\left(|R| + \frac{|S| \cdot |R|}{p}\right) = O\left(\frac{|S| \cdot |R|}{p}\right)$

Memory $= O(1)$

---

file file

$R' \in sort(R)$

$S' \in sort(S)$ ← e.g. 2Pass MergeSort on join key eg B in R.B=S.B

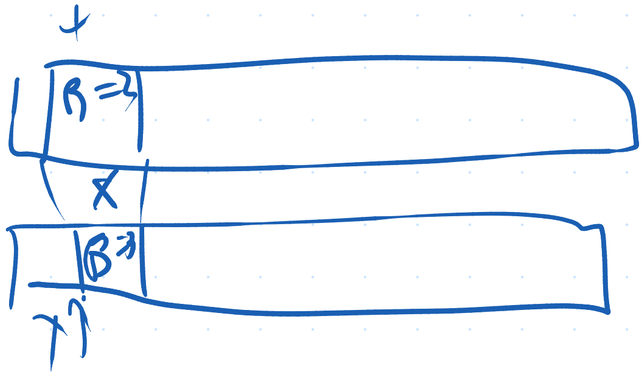


Table 1

Table 2

O(N)

key

q 1

b 2

c 2

d 3

1 n

1 n

2 0

2 1

1

1

2

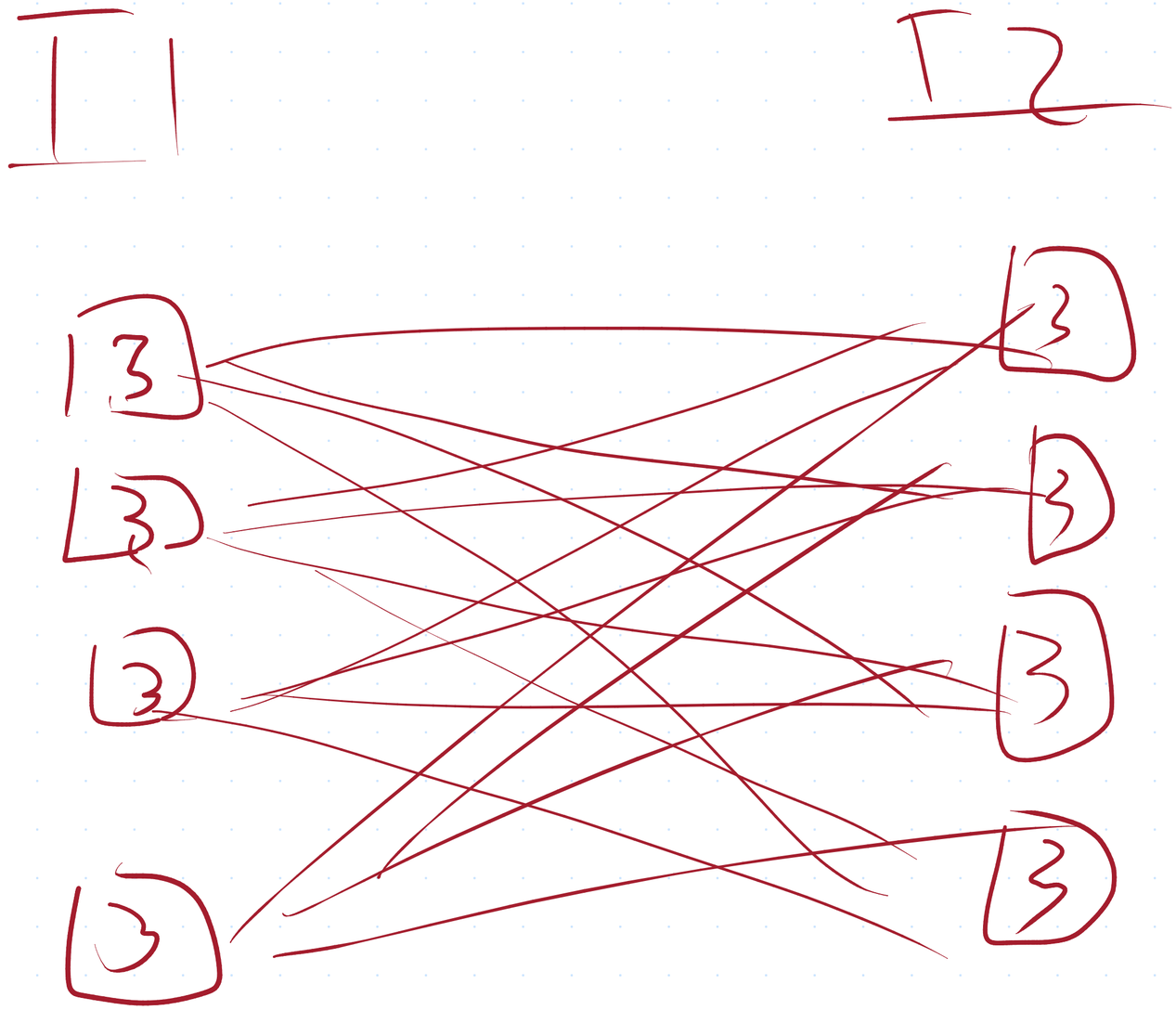
2

2

O(N)

key

Merge-Join



$O(N^2)$

Merge Join

Runtime:

- $O(N^2)$
- $(N+M)$
- $O(|T1| + |T2| + |T1 \bowtie T2|)$
  - if $|T1 \bowtie T2| = O(N)$ and $|T1| = O(N)$, $|T2| = O(N)$
    - $\hookrightarrow O(N)$

Assumes sorted tables

+ $O(N \log N)$ otherwise

IO complexity

$$O(|T_1| + |T_2| + |T_1 \bowtie T_2|)$$

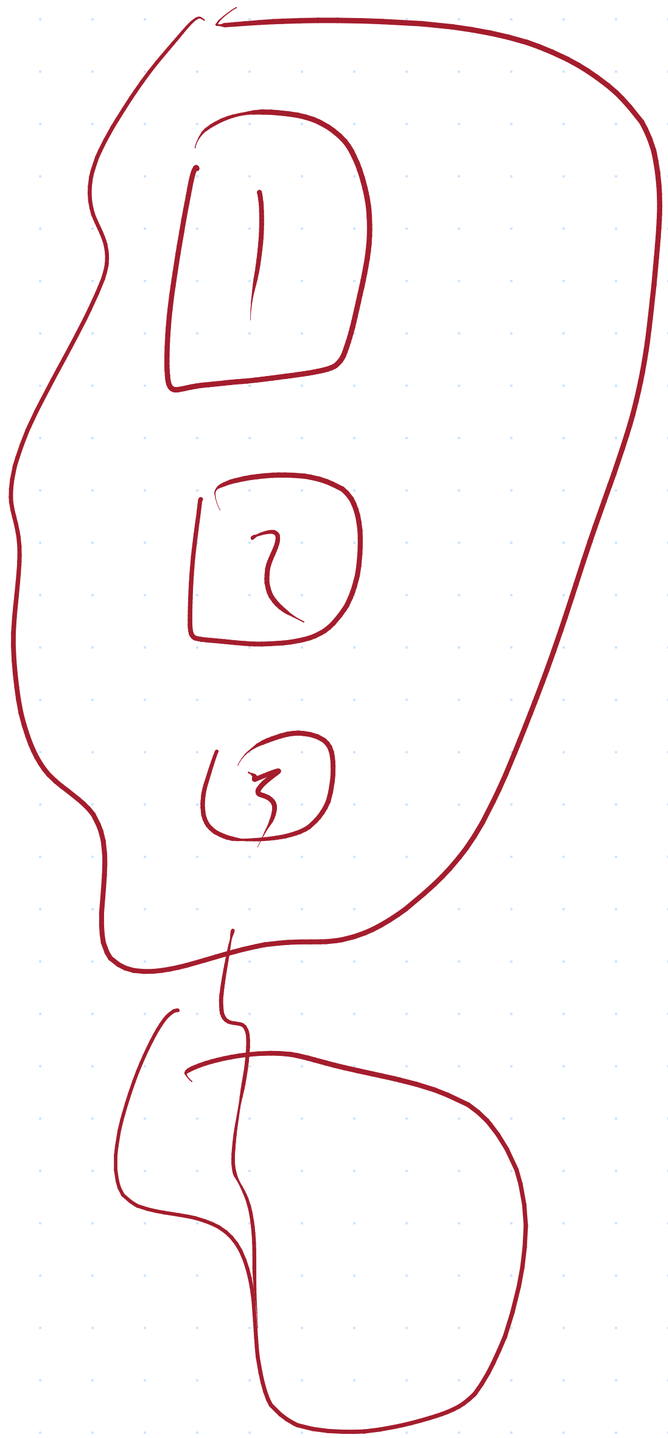
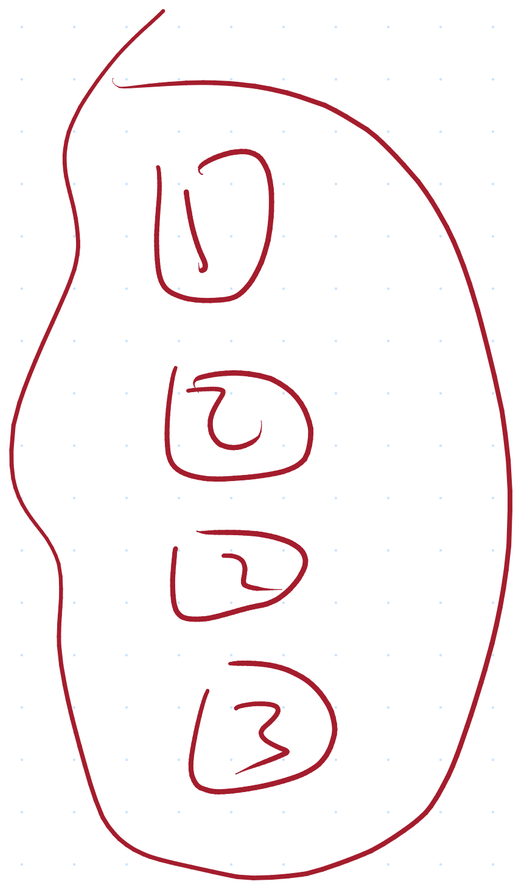
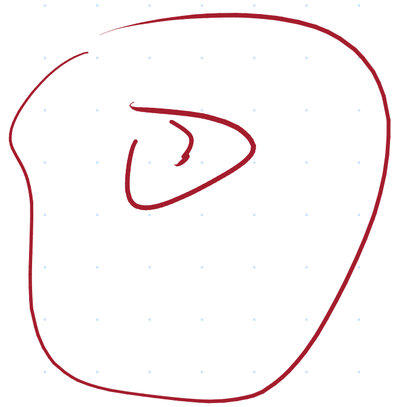
reads

writes

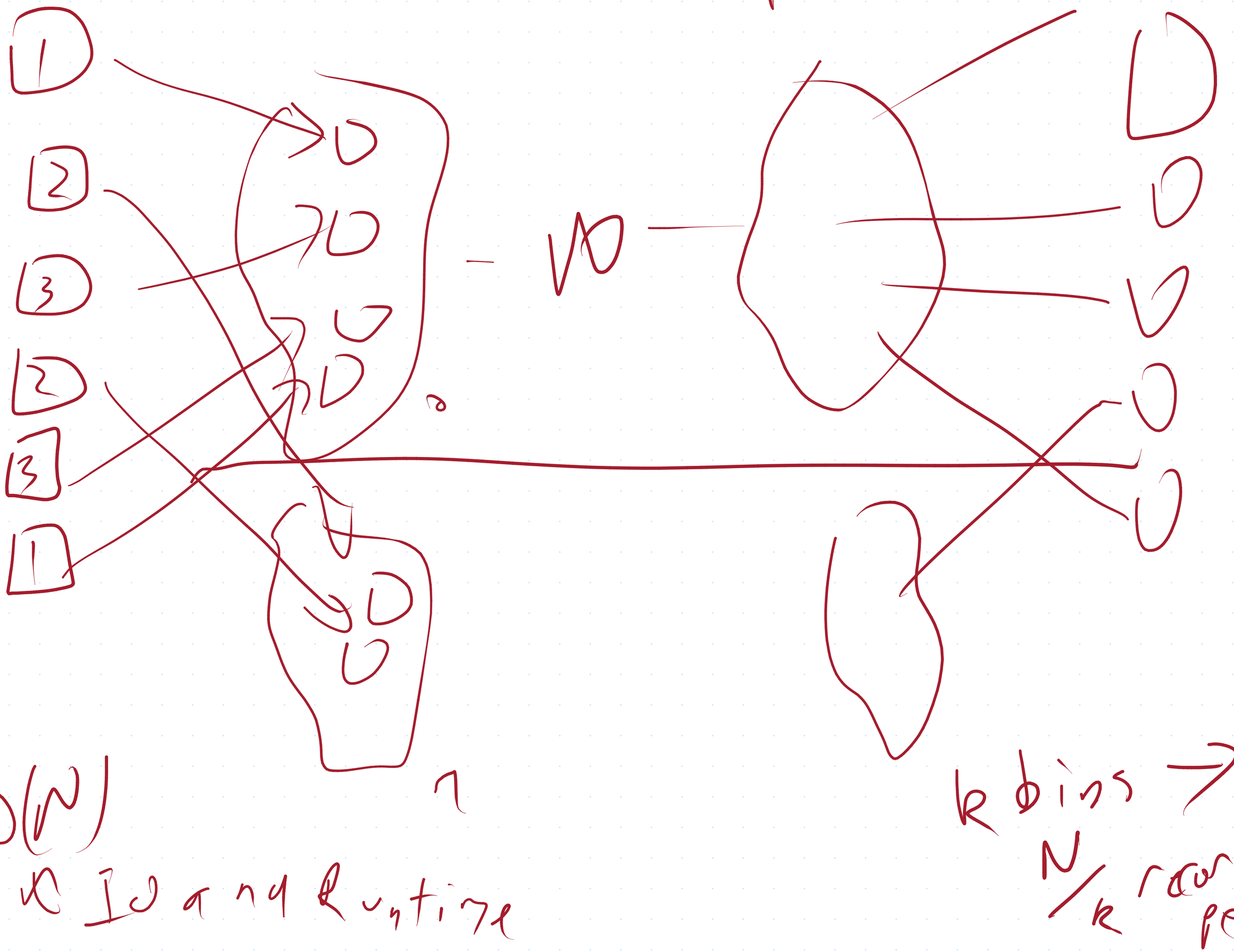
Memory complexity

$+ N \log(N)$ if unsorted

$O(\sum)$ Sum of fanout bounds of the two tables

Hash Join (2 pass)



O(N)

k Io and Runtime

k bins →

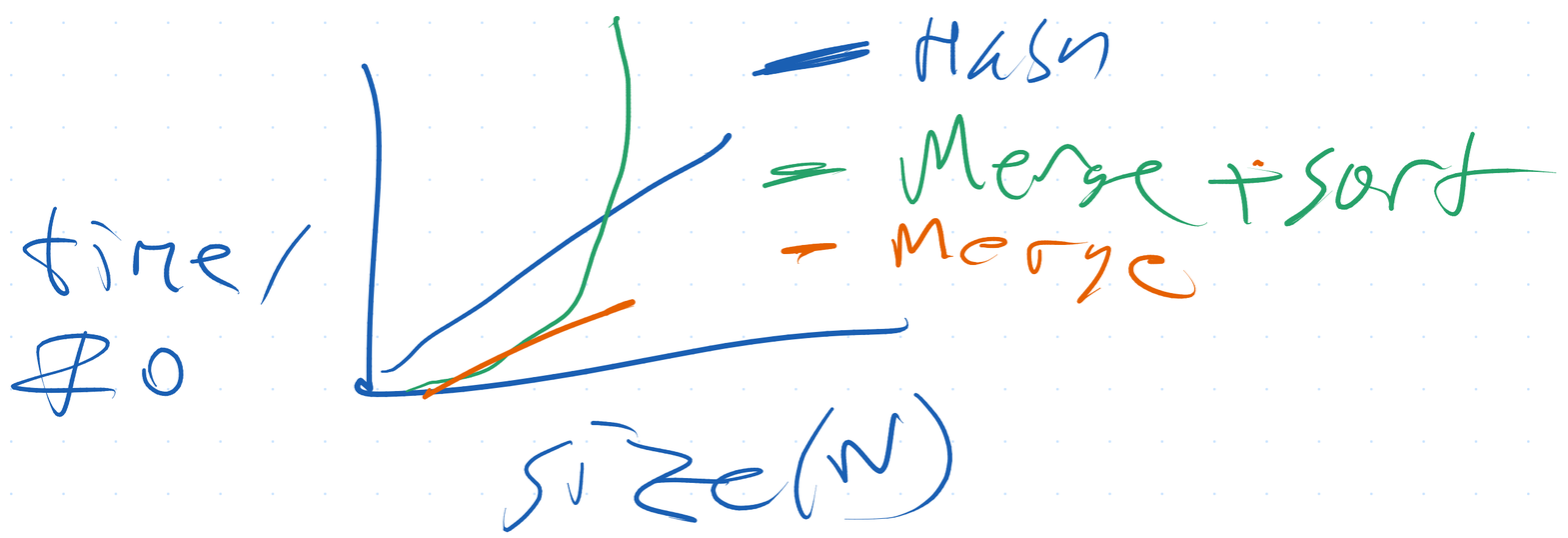
N/k records per page

Q) m ~ cn

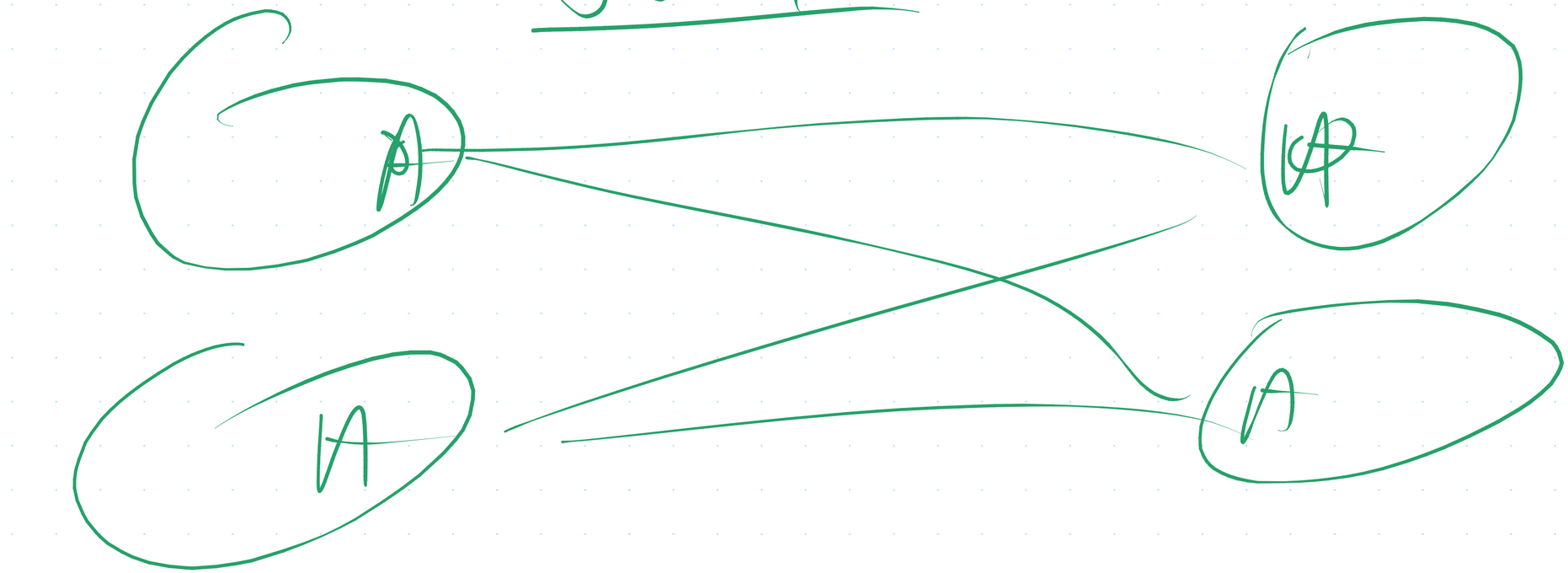
Total IO = O(|T1| + |T2| + |T1 ⋈ T2|) O(n/k | expected records per page

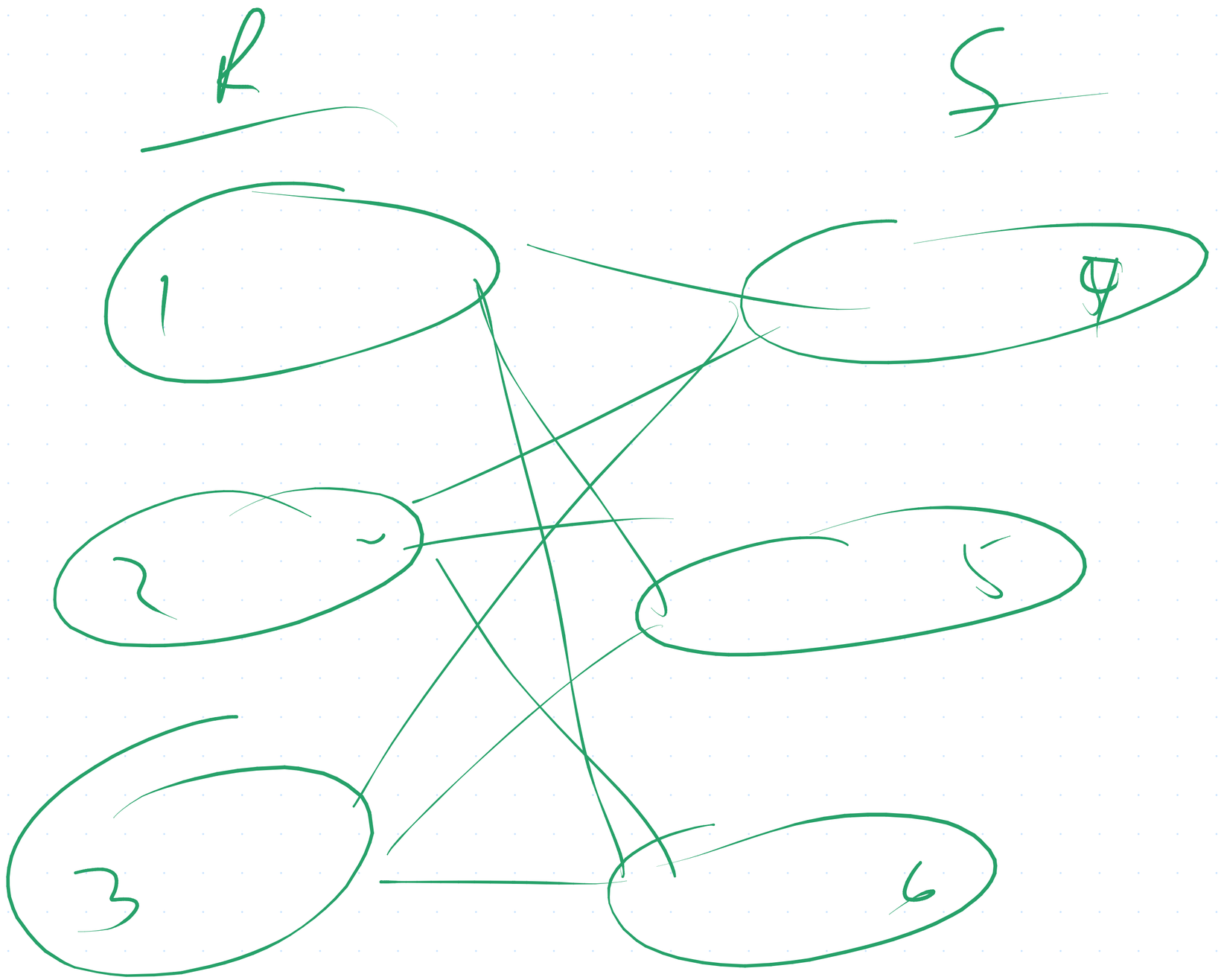
(and runtime)

one extra round of wire trans (higher constant)



Join

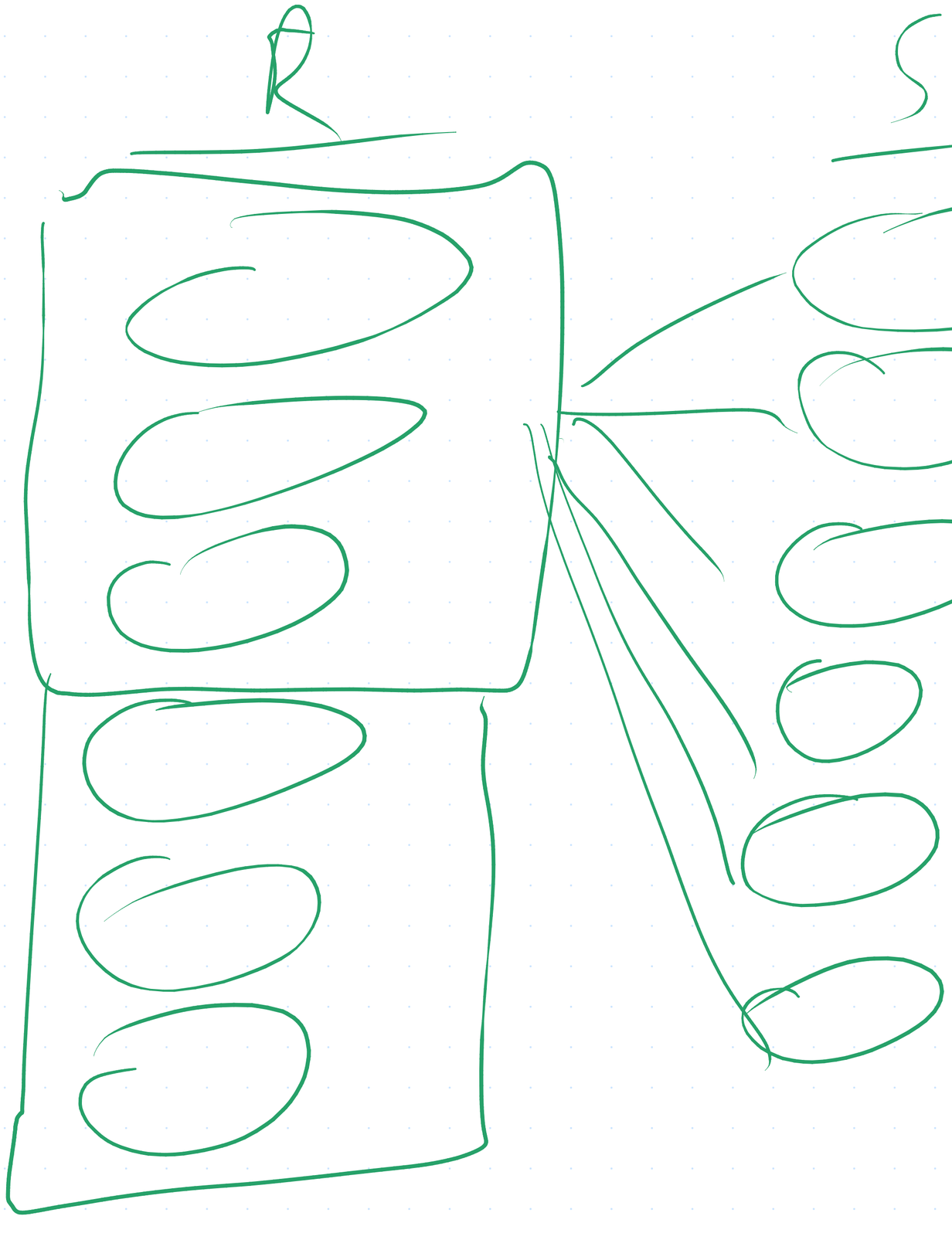




|R.key|
|=S.key|

Nested Loop Join

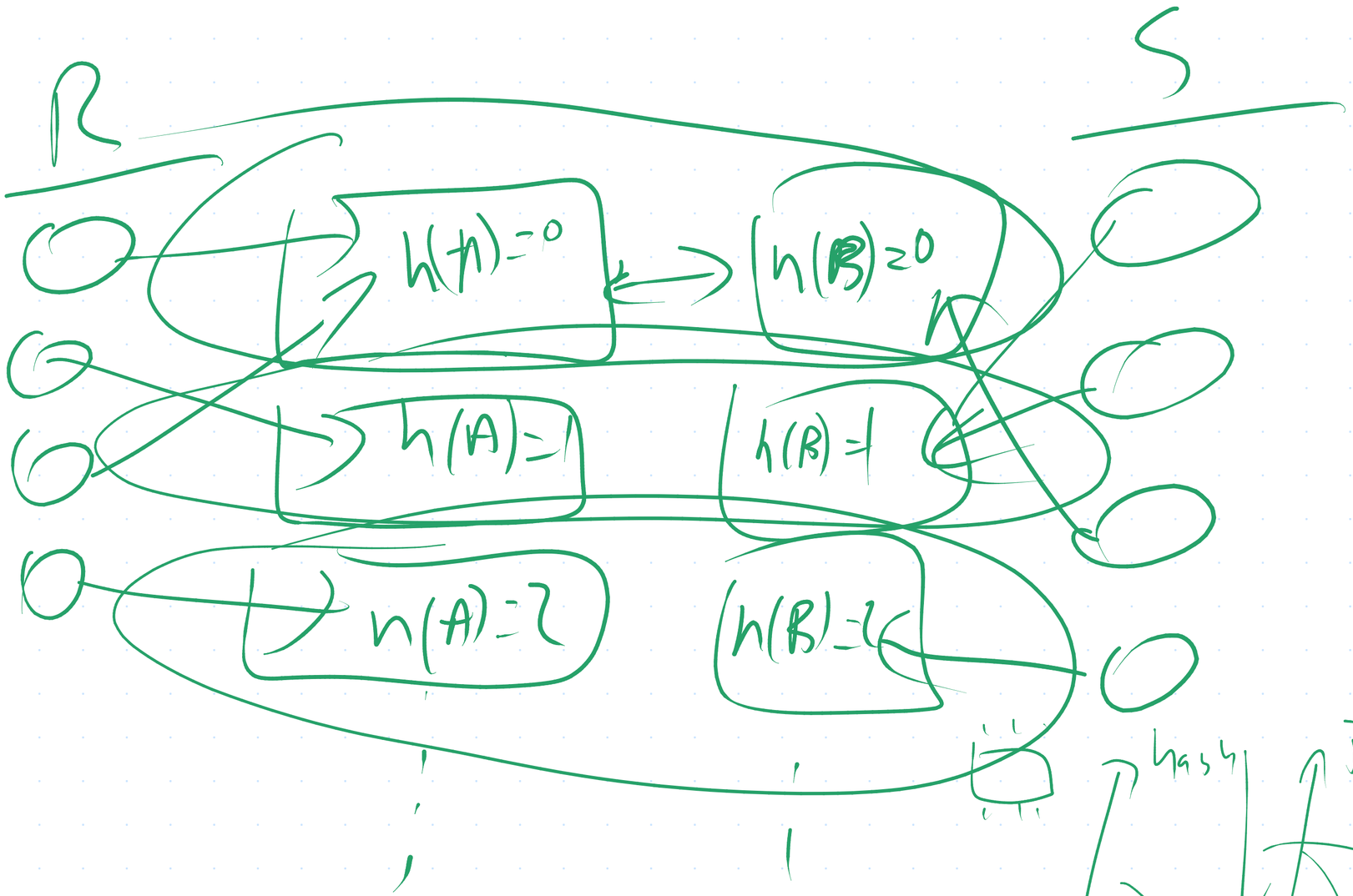
```
for r in R
  for s in S
    if test(r,s)
      emit r,s
```

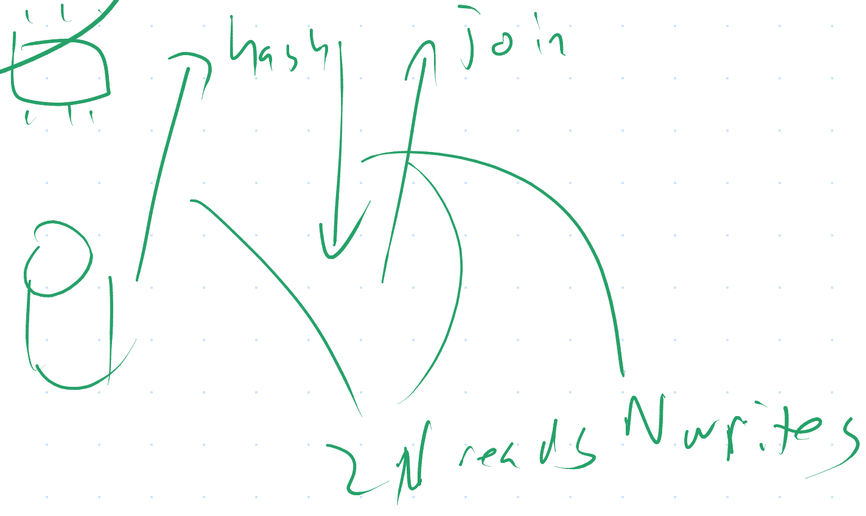


for block in R:
    for s in S:
        for r in block:
            if test r, s:
                emit r, s

Many
keys

$\rightarrow R.A = S.B$





2 Pass Hash

R.A=S.B

R

A=1

A=1

0

0

0

Sort increasing on A

S

B=1

0

0

6

6

Sort increasing on B

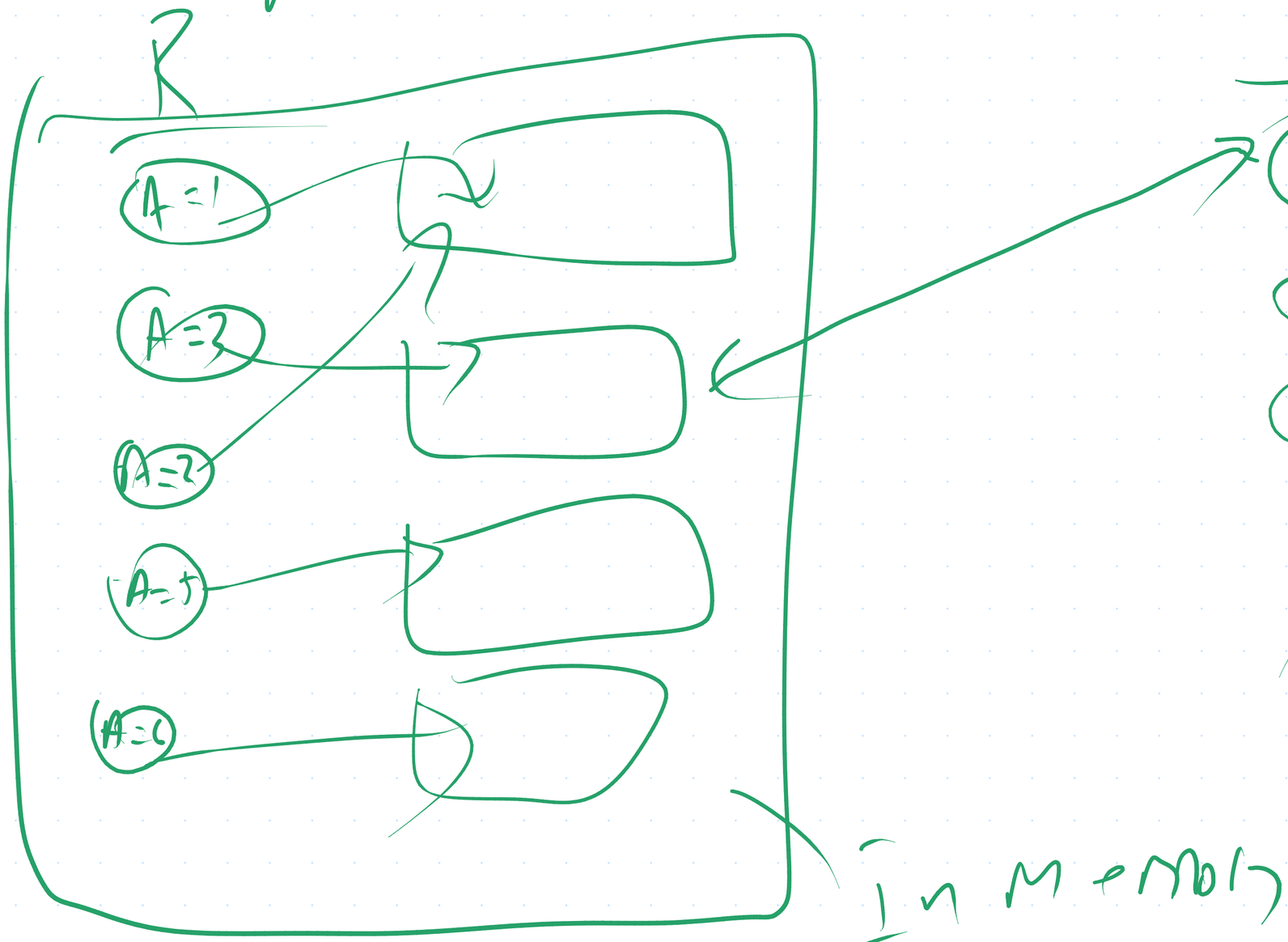
Similar to sort-merge

[Sort-] Merge Join

N records

R ⋈ S

M records



In memory

One-pass (Grace) Hash Join

	I/O	Mem
Nested Loop	$N \cdot M$	$N + M$	$O(1)$
Block Nested Loop	$N \cdot M$	$\frac{N \cdot M}{B}$	B
Sort Merge (w/o sort)	$O(1)$	$O(1)$	$O(1)$
2 Pass Hash	$O(N+M)$	$O(N+M)$	B
1 Pass Hash	$O(N+M)$	$O(N+M)$	$\|R\|$ or $\|S\|$

↑ smaller constant than 2 pass

Movies	Title	ID	Runtime	Gross	Genre
	A1	A2	A3	A4	A5

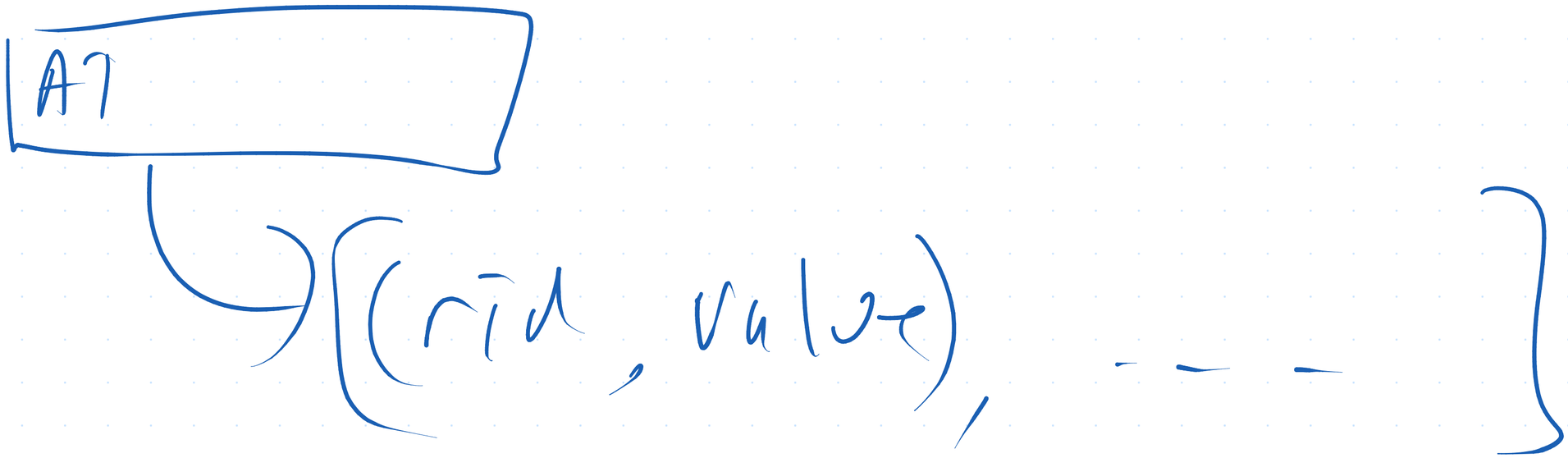
Movies	Title	ID	Runtime	Gross	Genre
	r1				
	r2				
	r3				

1 billion records @ 50Bytes/record

↳ 50GB

Q: Average Runtime of Action Marks

Executor memory: 12GB


A1
[(rid, value), ---]

Movies =

Title (RID, value)

ID (RID, value)

Runtime (RID, value)

Gross (RID, value)

Genre (RID, value)

Title ⋈ ID ⋈ Runtime

⋈ Gross ⋈ Genre

Runtime ⋈ Gross

Ideal

No RIDs

↳ Use position in Array implicitly

Pros - Less overhead (No RIDs)

Cons - Fixed sort order, Expensive to find RIDs for values

- Harder to use w/ variable-sized data

Example: Genre

Categorical Data

↳ "Sci-Fi" → 0

"Action" → 1

"Drama" → 2

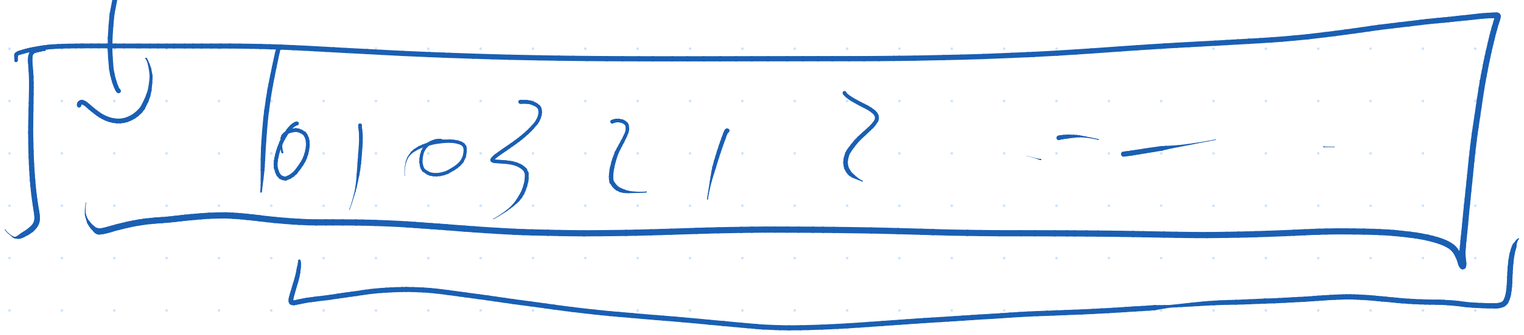
"Thriller" → 3

8 bytes

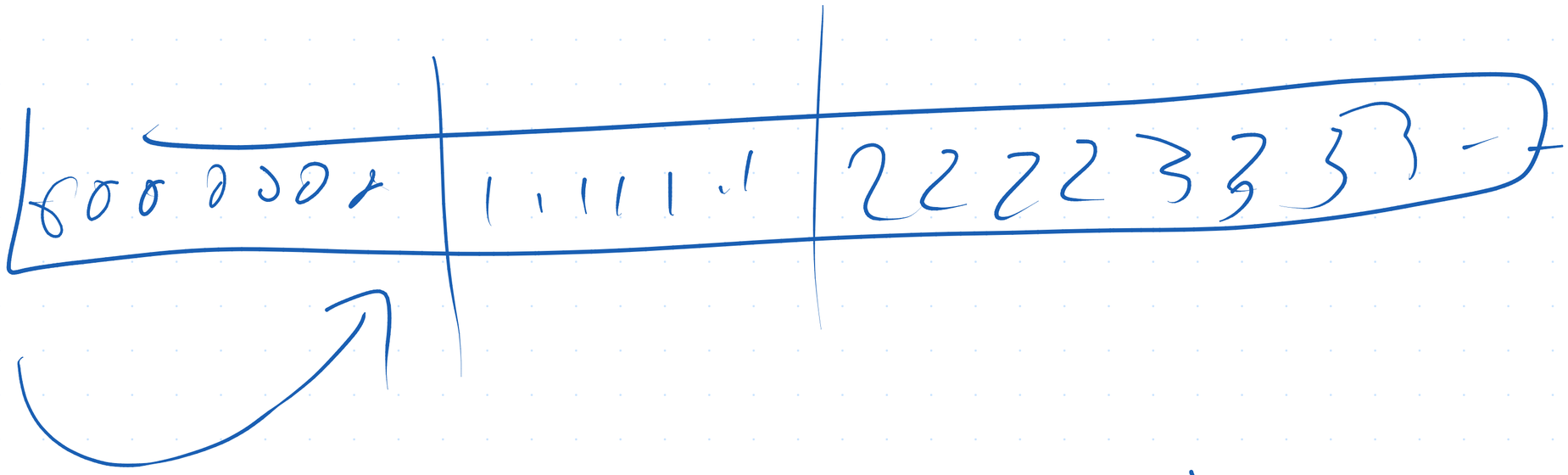
↓

1 byte

Dictionary



↑ Dictionary encoded data



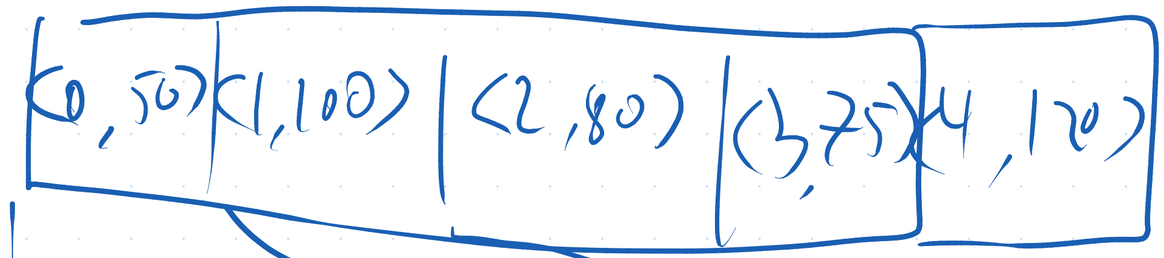
'0' for the next 50 entries

'1' for the next 100 entries

<0, 50>
<1, 100>
⋮

Run-length encoding

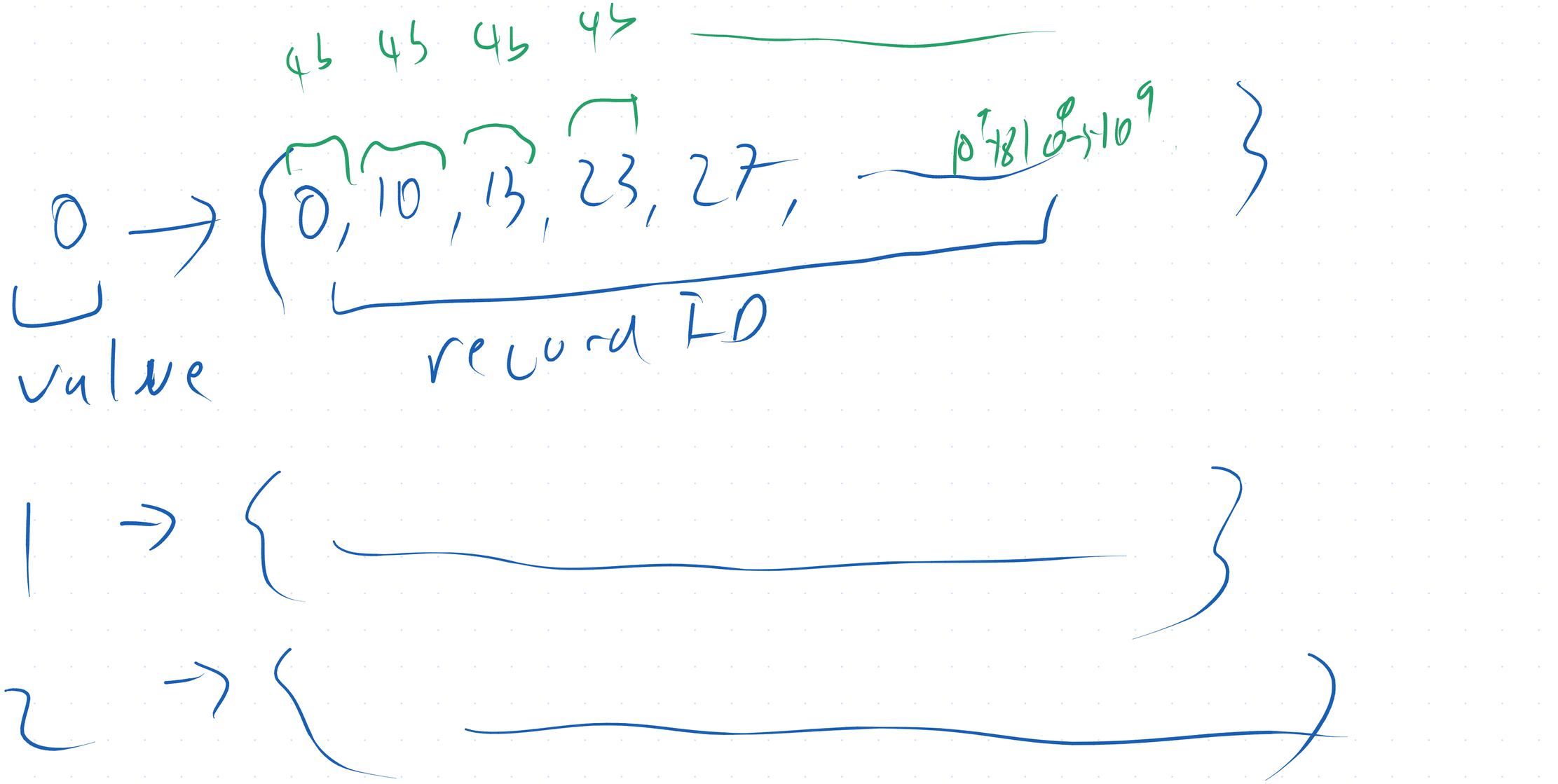
1837

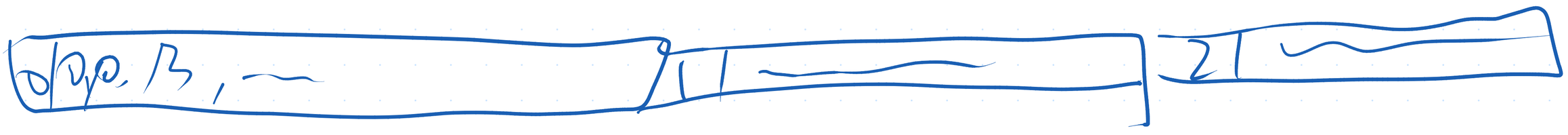


0→0 ... 49→0 50→1 ... 149→1, 150→2 ...

↳ pro/small

con/Requires sorted runs in the data


4b 4b 4b 4b
0 → { 0, 10, 13, 23, 27, 0~781 of ~10^9 }
value
record ID
1 → { }
2 → ( )


dpo.B, —
1 —
2 ~

0, 10, 13, 23, 27

10 3 10 4