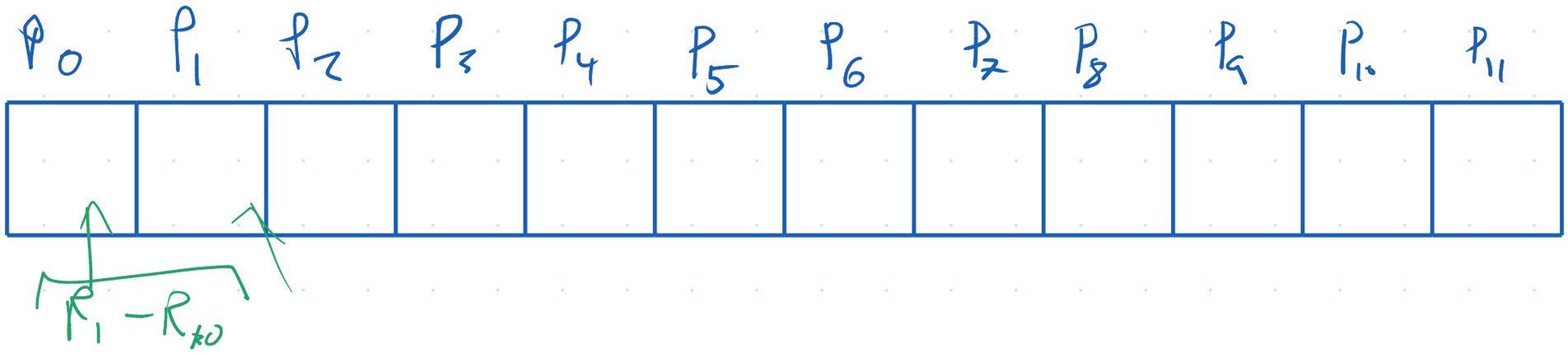
CSE 350

Advanced Data Structures

Topic 9: Storage in Trees □□

File Layout

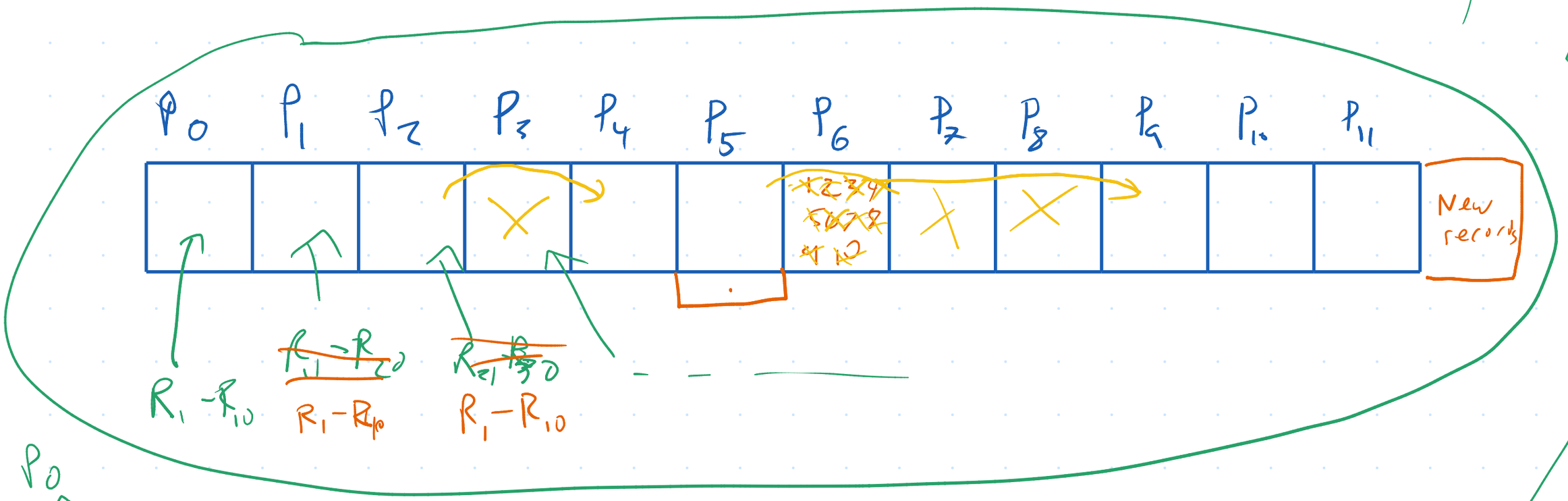


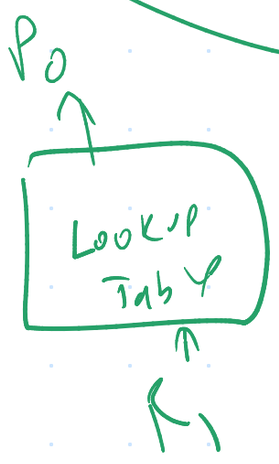
Goals

- Fast record lookups
  - ↳ CPU
  - ↳ IO
- Low wasted space → Fast scans
- Data mutability

Flat File

Table → Bag
→ Set
→ List
of records (Tuples)





Idea for name

Key

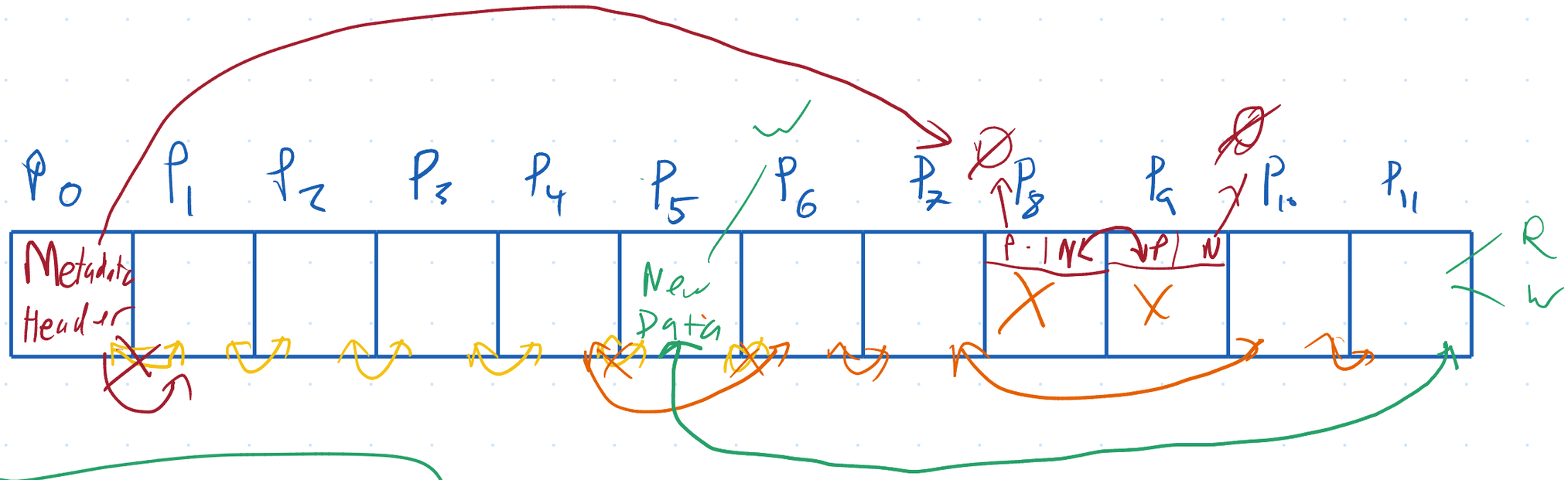
Permit#	City	Type	Address
→ 1	Buffalo	Electrical --	
2	Depew	Electrical --	
3	Buffalo	--	

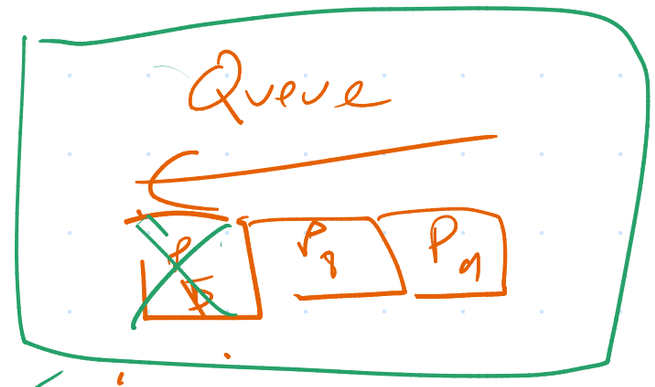
(P_0, R_1) R_1
(P_0, R_2) R_2
R_3

Load Page 5
↳ Find R_4

Find P_5 R_4

Linked List

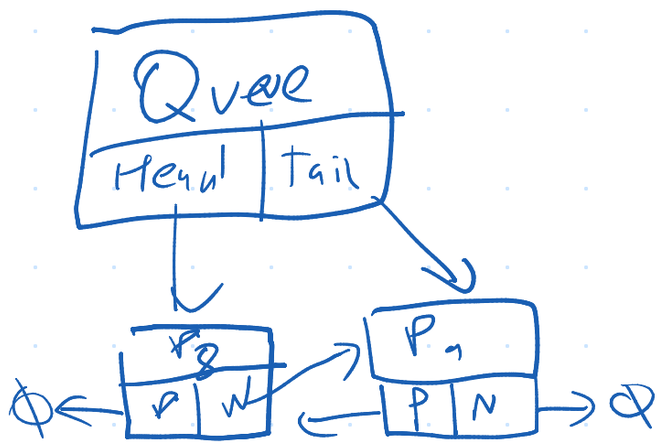


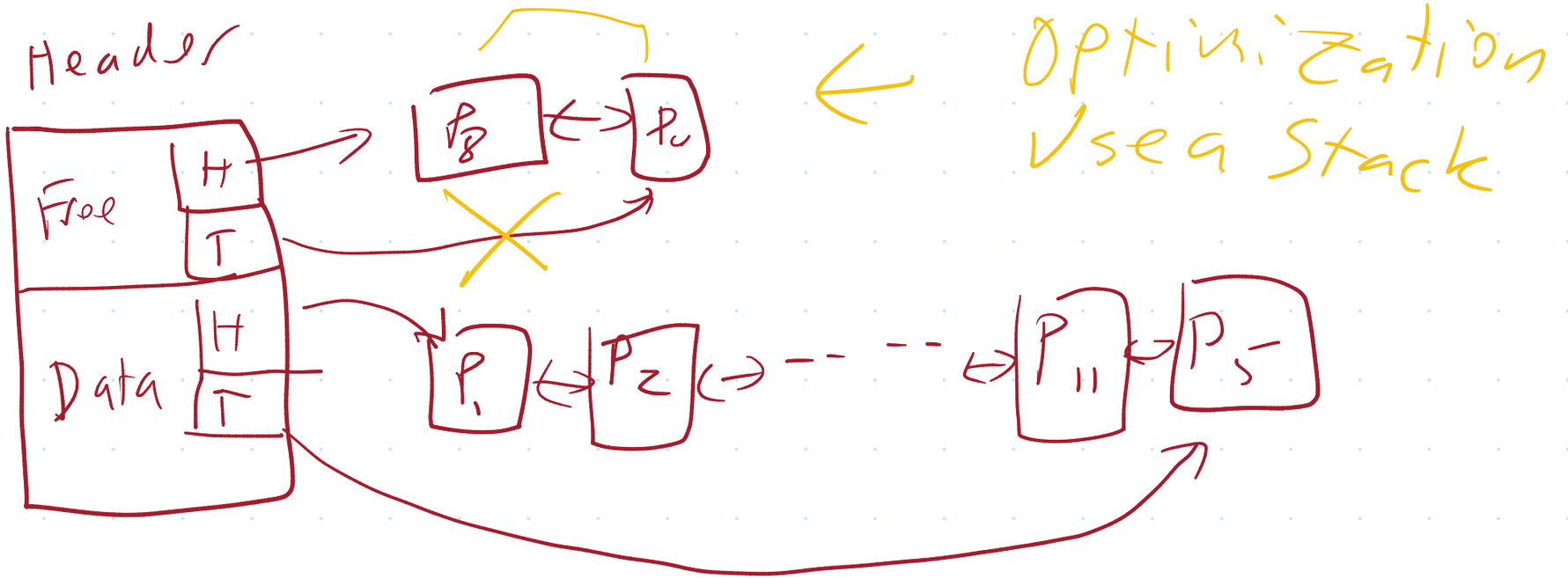


Store On Disk?

$P_0\ P_1\ P_2\ P_3\ P_4\ P_6\ P_7\ P_8\ P_{11}\ P_5$

3 IOs





← Optimization
Vsea Stack

Finding Records

How is a record identified?

↳ Pointer (Page, Record index)

↳ Key

(Any attribute to id a collection of records)

What are the tradeoffs of each?

Key

⊖ Table may not have one

⊕

⊖ Need to map key to location (mapping function)

Pointer

⊕

⊖ SQL only talks in attributes

⊕ Fast lookups

How can we change the file layout
to make key lookups faster?

Problem

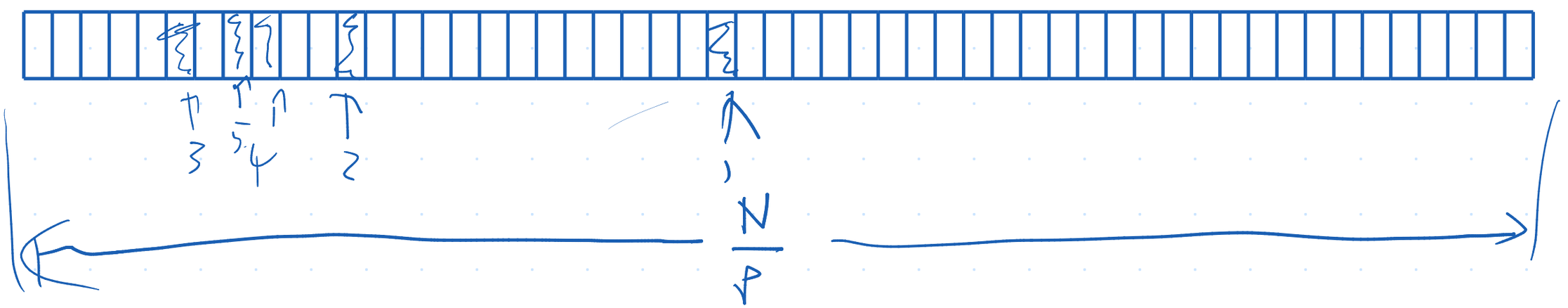
↳ No information relating key to location by default

↳ HashTable

↳ Sort it

Idea 1: Sort the Records

How do we find a record?



How Much IO?

N records
p records / page

IO	Search Space
0	N/p pages
1	$\frac{N}{p} \cdot \frac{1}{2}$ pages
2	$\frac{N}{p} \cdot \frac{1}{4}$
i	$\frac{N}{p} \cdot \frac{1}{2^i}$
$\log_2 \frac{N}{p}$	1 page

$= \log_2 N - \log_2 p$

What is the IO complexity of binary search?

On a sorted file

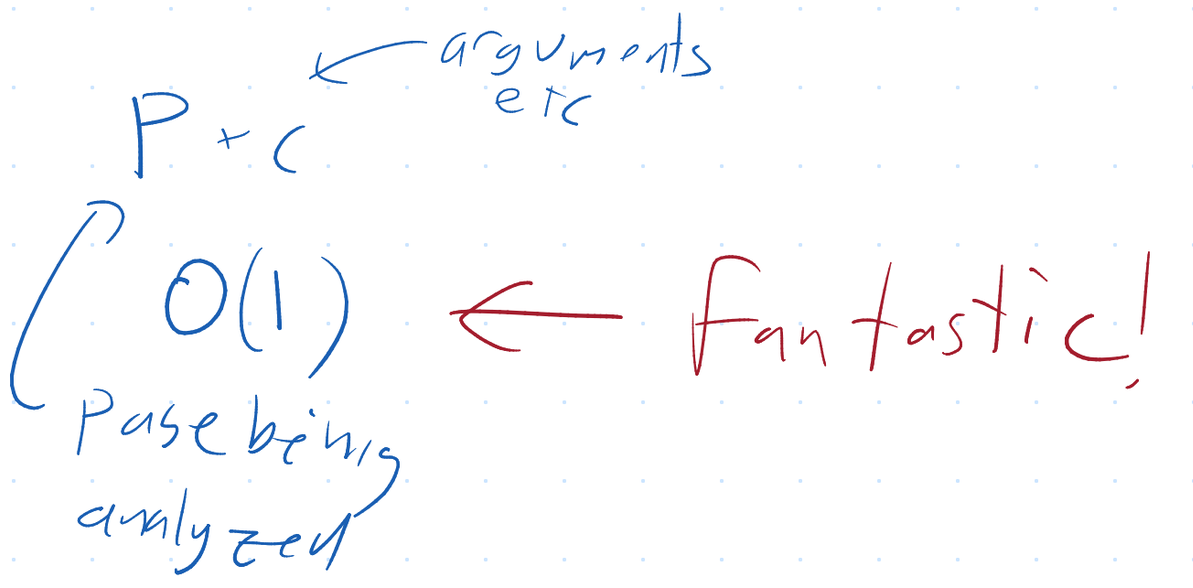
$\log_2 N - \log_2 P$ P pages read

$O((\log_2(N))$ IO

☹ Not great

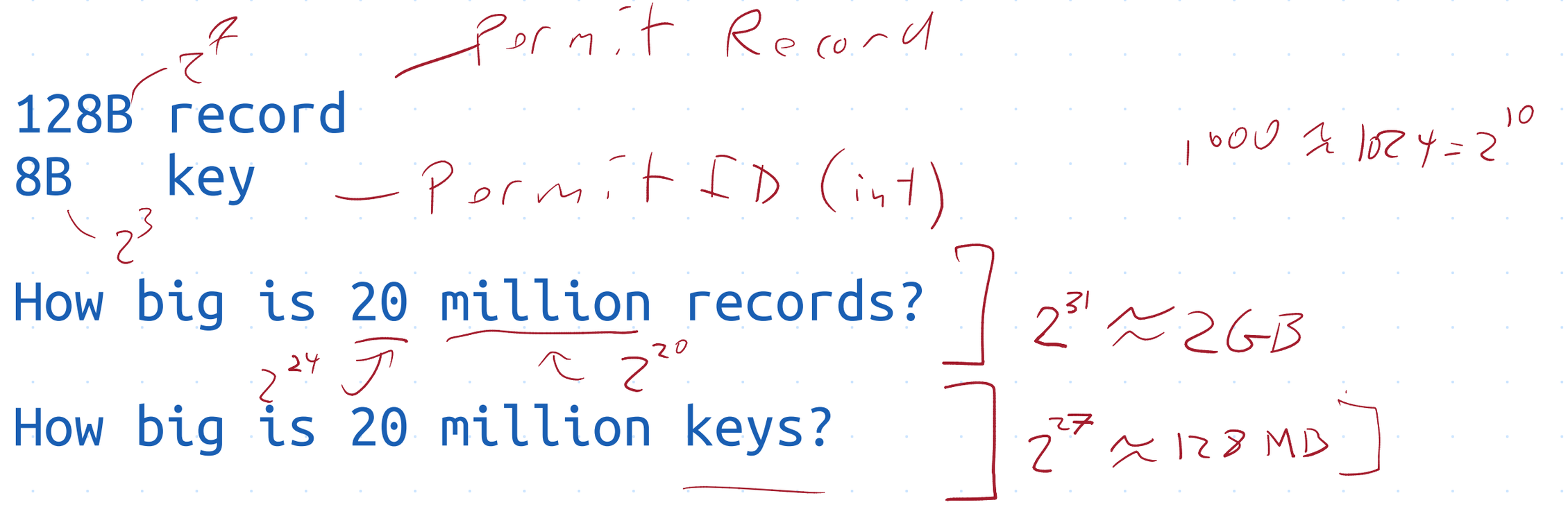
What is the Memory complexity of binary search?

on a sorted file

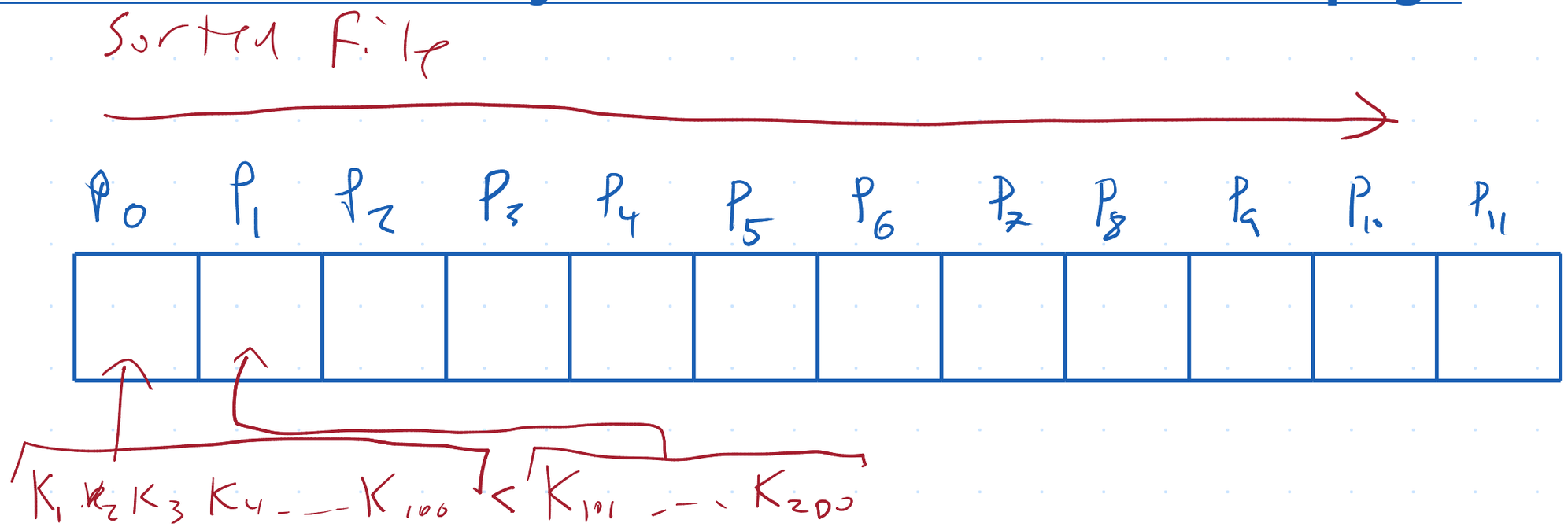


Relative Sizes

128B record
8B key
How big is 20 million records?
How big is 20 million keys?

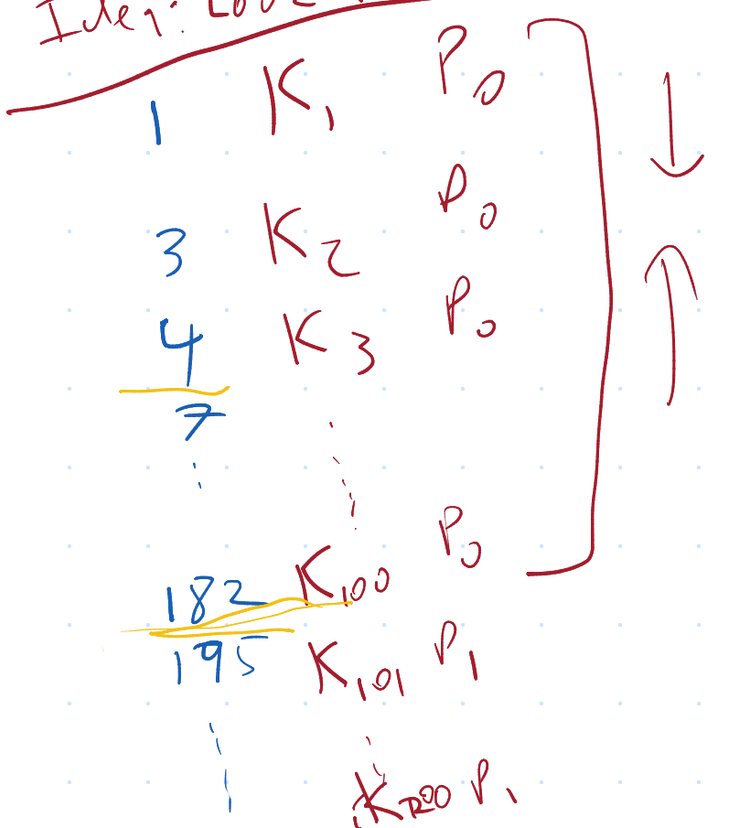


Idea 2: Track key information for each page



$K_1 < K_2 < K_3$ - - -

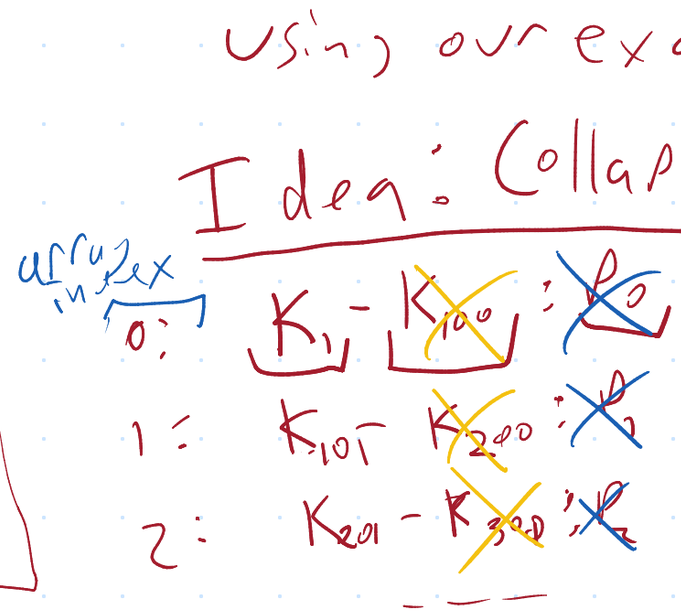
Idea: Lookup table



N Keys + pageids ⇒ much smaller than N records $\frac{1}{2^3}/2^7$

using our example $\approx \frac{1}{2^3} = \frac{1}{8}$ the size

Idea: Collapse Lookup Table



~~2^8~~ $2 \cdot 2^8$ bytes

$P = 100$

~~$2N$~~ $\frac{N}{P}$ keys vs N records

much much smaller

~~Y_{100}~~ the size

Find 312

I O

Fence Pointer Table

192	274	458	703	904	1089	128	1358	1584	1821
P_1	P_2	P_3	P_4	P_5	P_6	P_7			

← Array of keys

How big? $O(\frac{N}{P})$

One key per page $\approx O(N)$

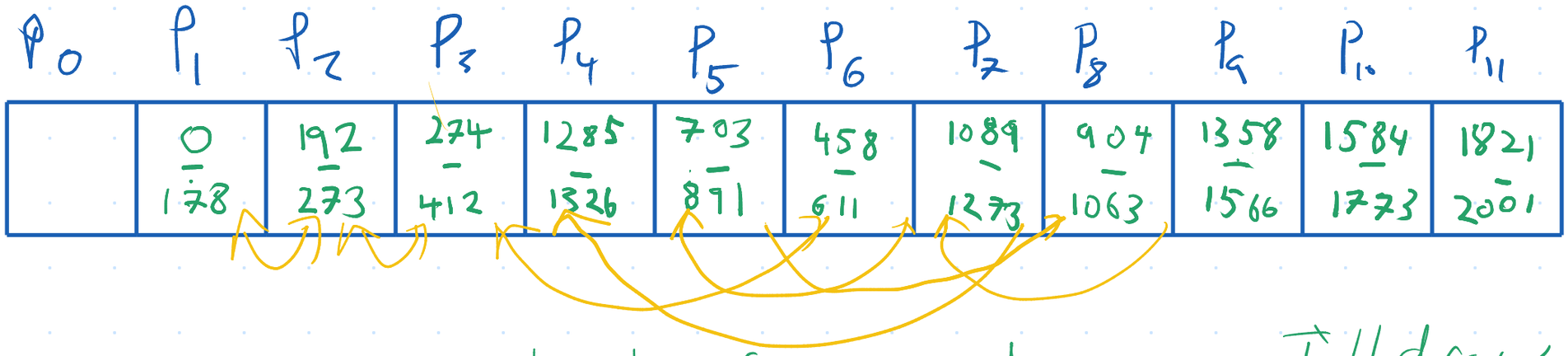
P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}
	0 – 128	192 – 273	274 – 412	458 – 611	703 – 891	904 – 1063	1089 – 1273	1285 – 1526	1358 – 1566	1584 – 1773	1821 – 2001

IO1 *

IO2

$O(\frac{K}{P^2})$ pages of fence pointer data

Variation: Unordered Data Pages

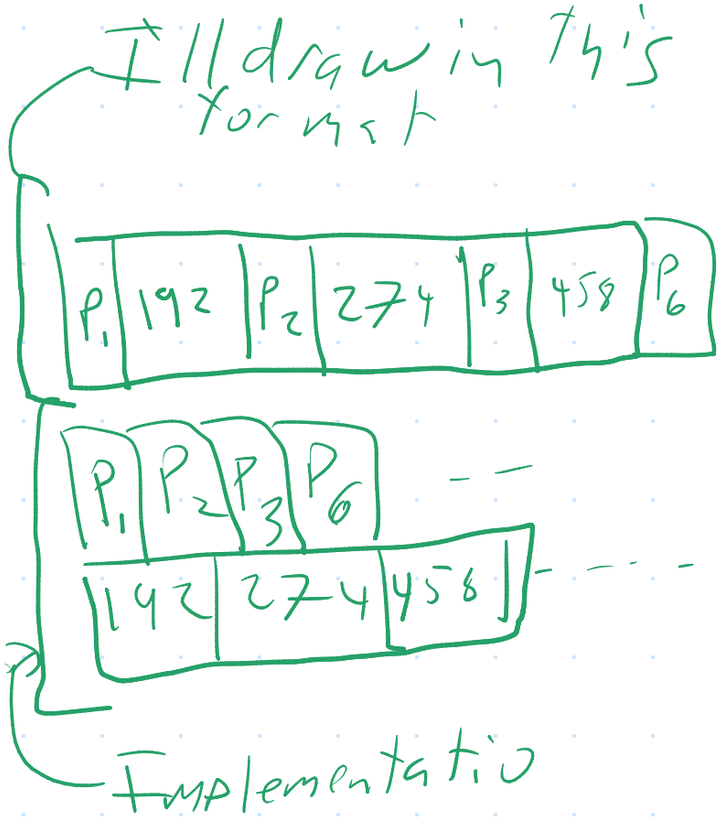


Linked list for order

[192 274 1285 703 458 1089

[192 P1
274 P2
458 P3
703 P6
904 P5
1084 P8
P7
]

I'll draw in this format



Implementatio

What is the IO complexity of binary search?

w/ FP Table

"2"

But really O(N) ← N divided by a BIG constant

Variation: Keep FP Table In memory

↳ IO Exactly

What is the Memory complexity of binary search?

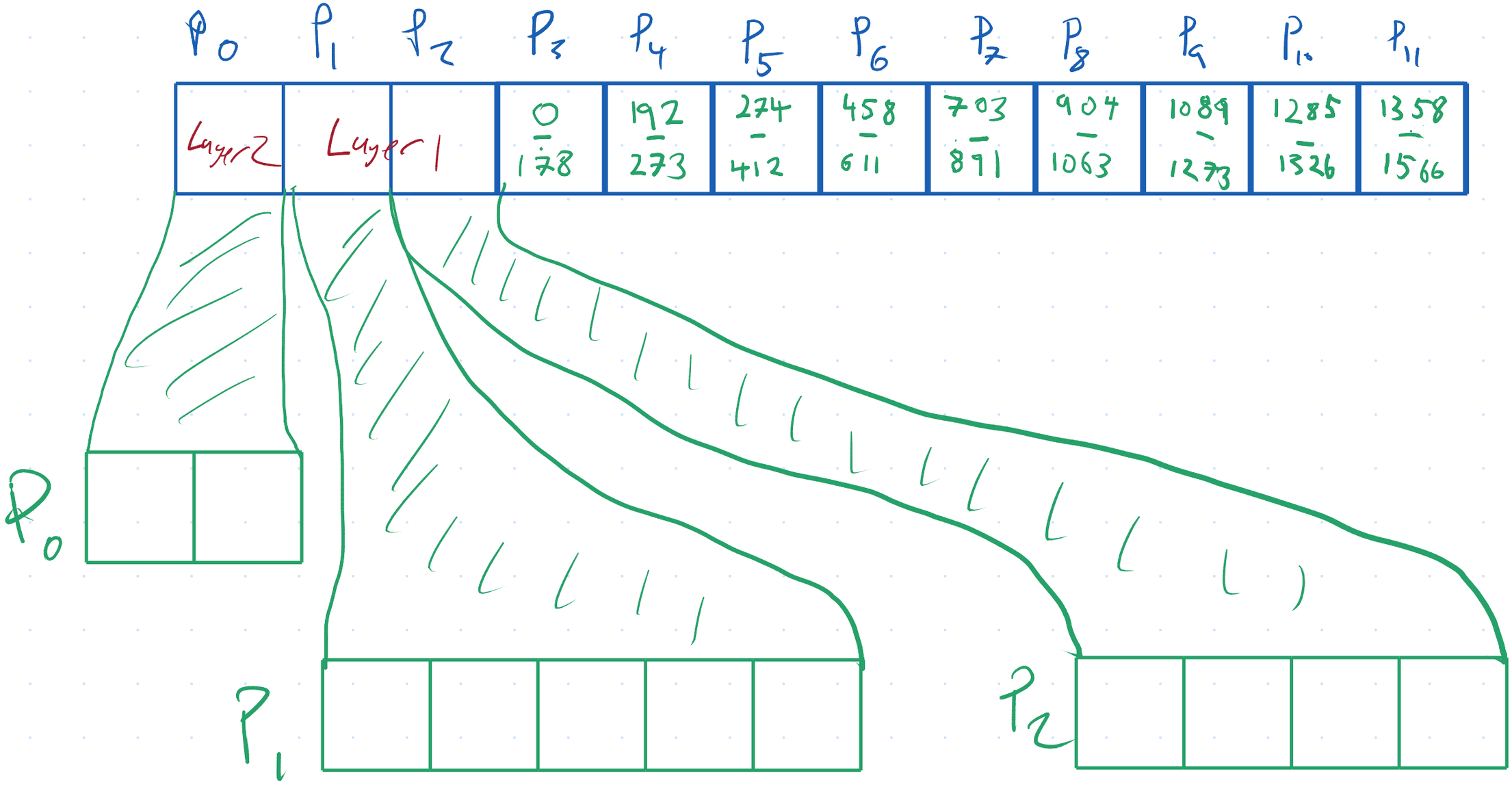
w/ FP Table

1 Page + small constant

Variation: Keep FPTable In Memory

$O(\frac{N}{P})$ memory

Idea 3: Layers



ISAM Tree

Index structure

Access method

Directory Pages

Layer 2

Layer 1

P_0 | 904 |

P_1 | 192 | 274 | 458 | 703 |

P_2 | 1089 | 1285 | 1358 | |

P_3 P_4 P_5 P_6 P_7

| 0 – 178 | 192 – 273 | 274 – 412 | 458 – 611 | 703 – 891 |

Data Pages

P_8 P_9 P_{10} P_{11}

| 904 – 1063 | 1089 – 1273 | 1285 – 1326 | 1358 – 1566 | |

How deep is the tree?

What is the IO complexity of binary search?

What is the Memory complexity of binary search?

What needs to change to allow changes?

Idea 4: B+ Trees