B+TREE

QuyenPT3-TranNDC

Agenda

01 B-TREE
B-Tree structure
Pros and cons

02 B+TREE
B+Tree structure
Why?
Applications

03 Implement Key Value Store with B+TREE
- FIles

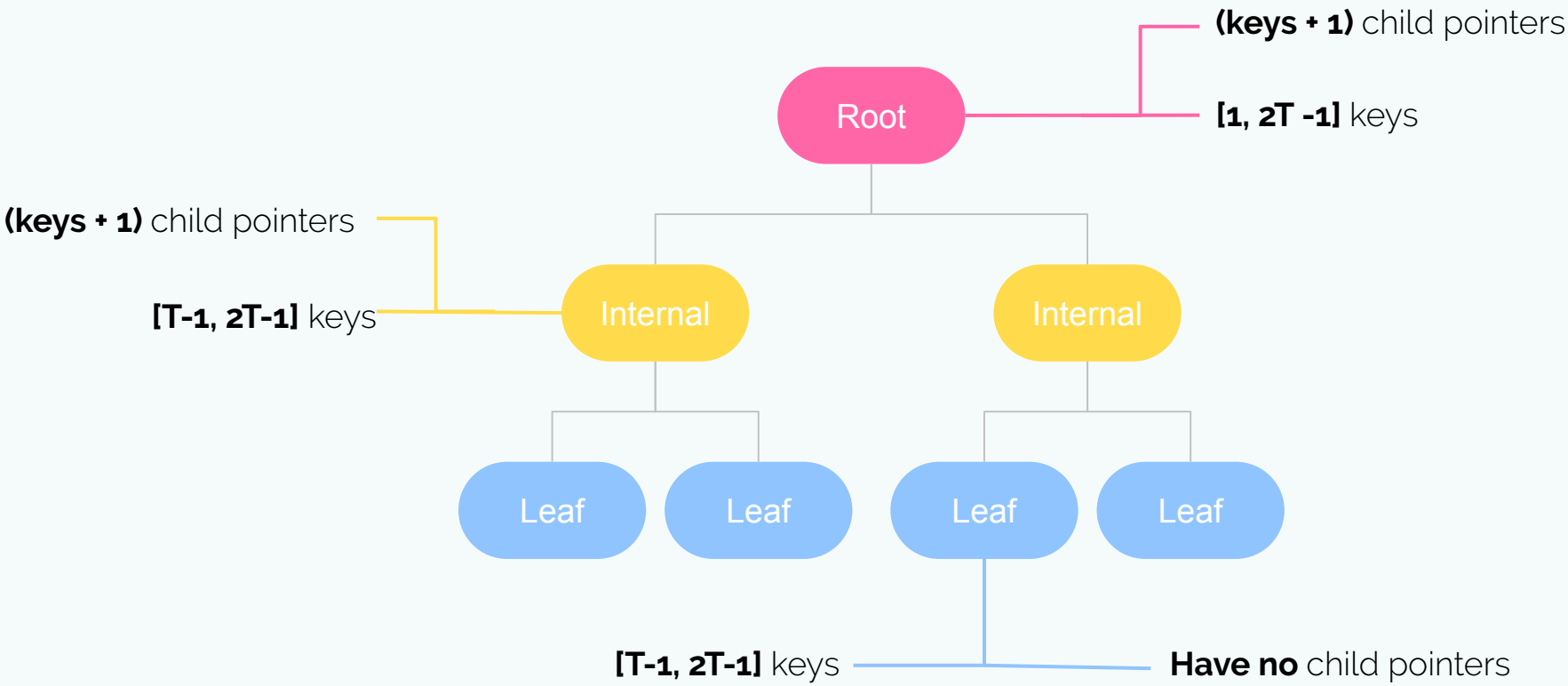
04 DEMO

01

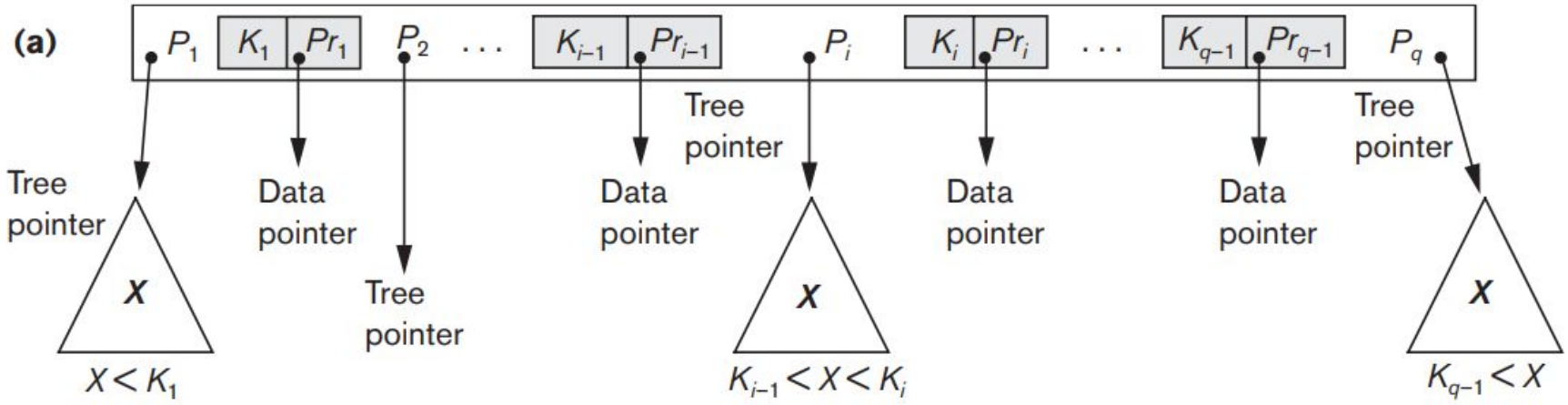
B-TREE

B-Tree Structure
Pros and cons

B-Tree Structure



B-Tree Structure



Pros and Cons

- **Height of the tree** is minimized
- Reduce disk latency: **O(h) disk access**
- Total CPU time used: **O(th)**

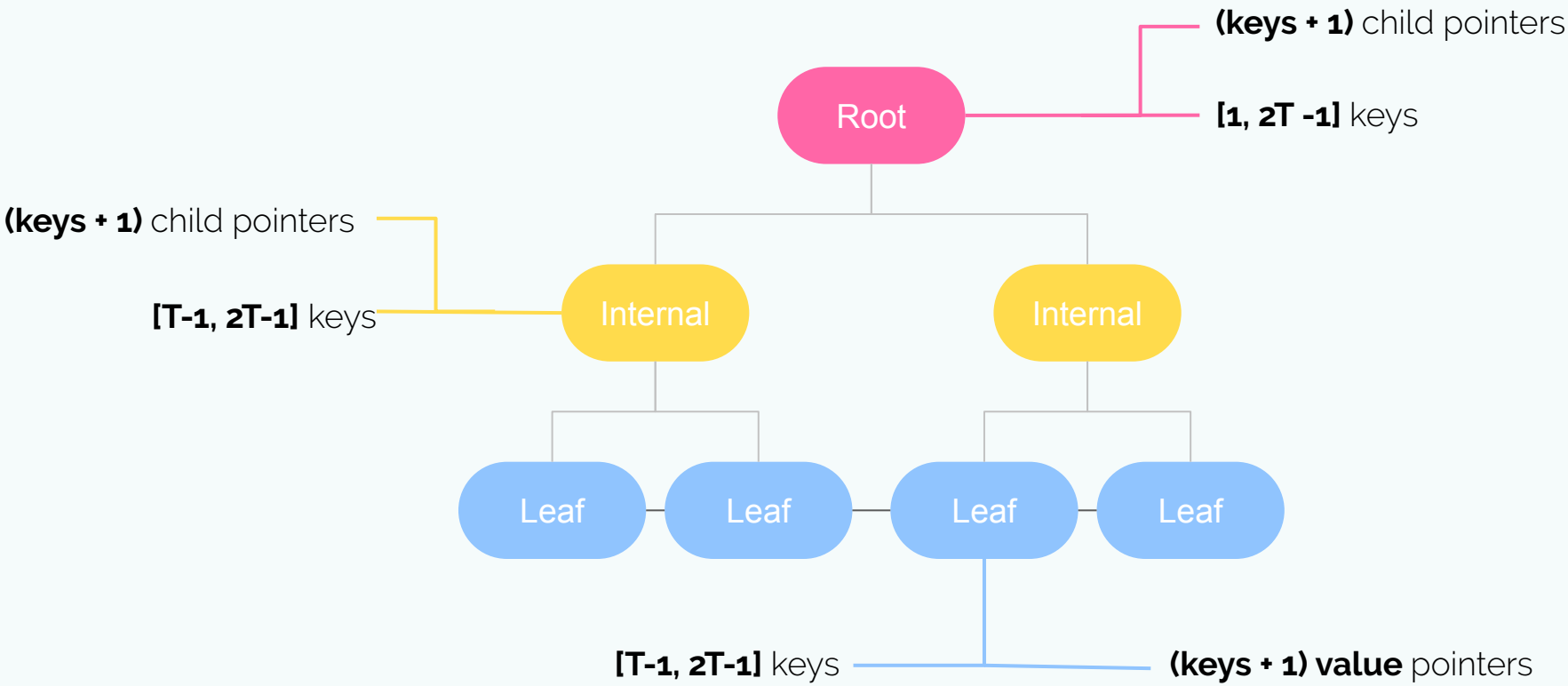
- The **number of keys** in one node is **limited** because keys are stored along with their values (or pointers of values)
- Poor write performance: Random writes and write-amplification
- Delete operation is complex when handling **deletion on internal nodes**

02

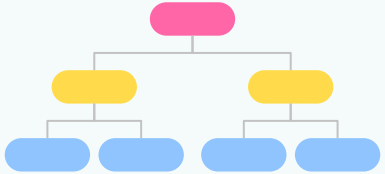
B+TREE

B+Tree Structure
Why?
Applications

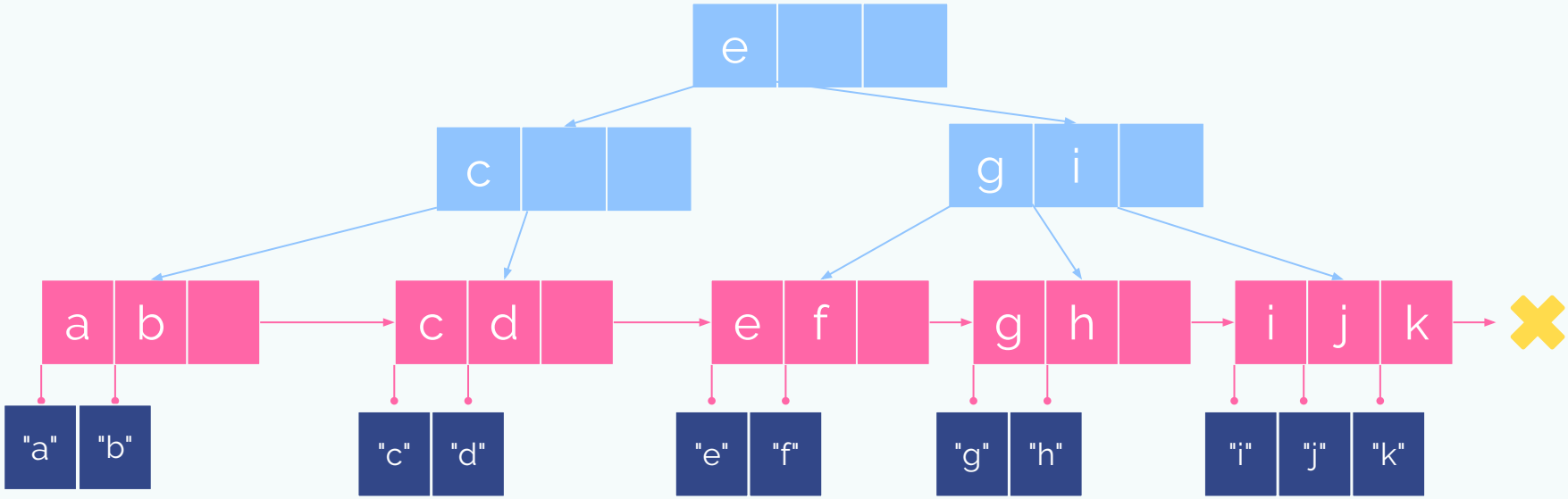
B+Tree Structure



B+Tree Structure



B+Tree Example



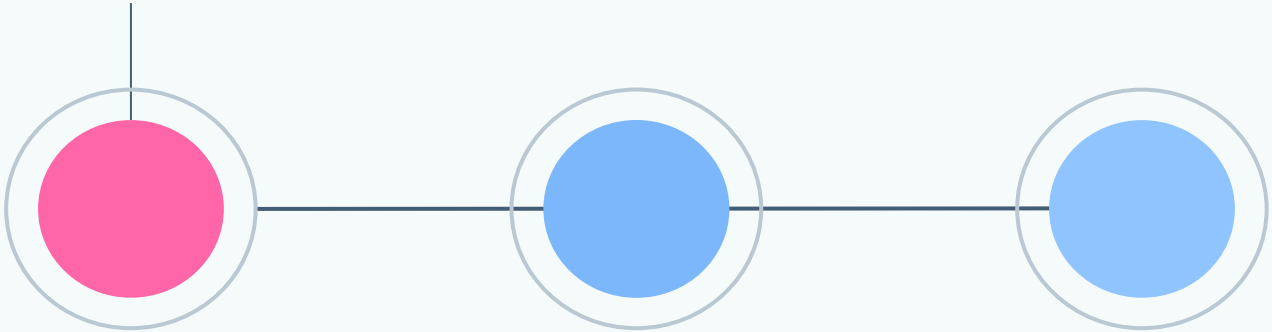
B+Tree operations complexity

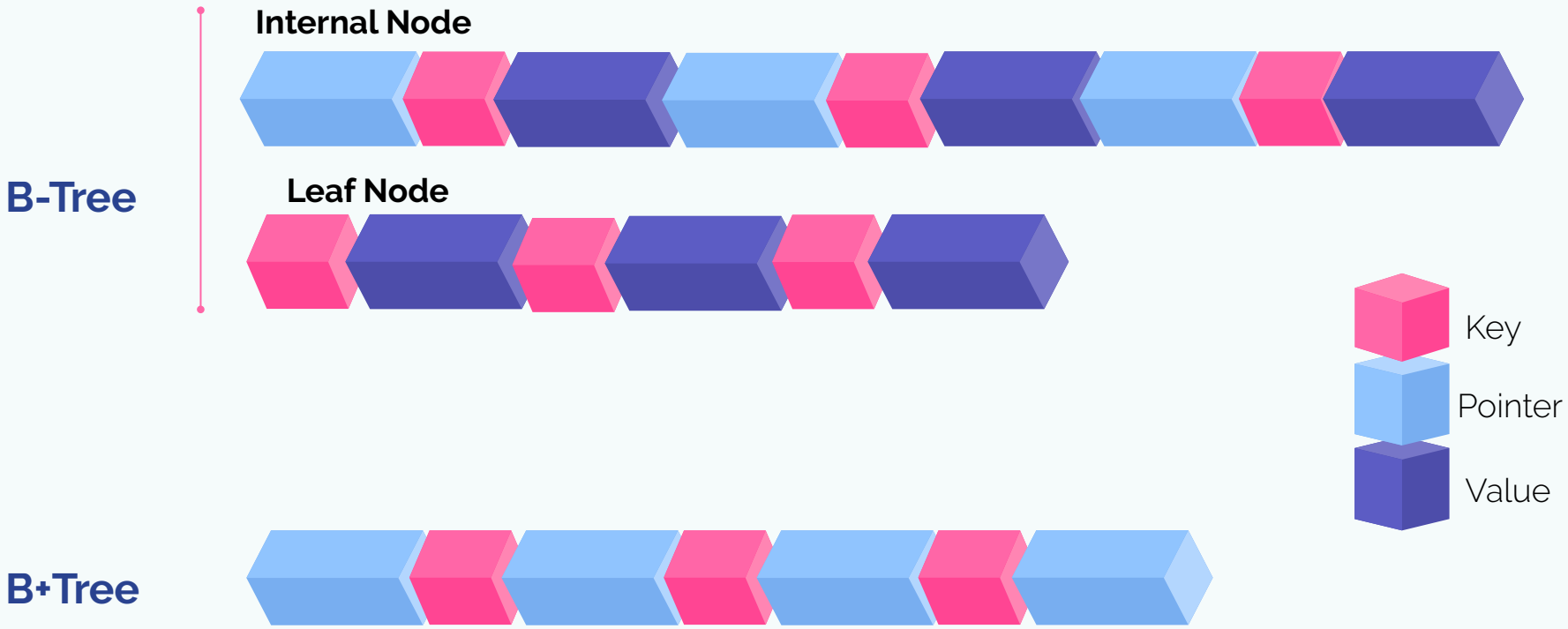
- **Insert**: *O(logN)*
- **Remove**: *O(logN)*
- **Get**: *O(logN)*
- **Exist**: *O(logN)*

Why B+Tree ?

Why B+Tree ?

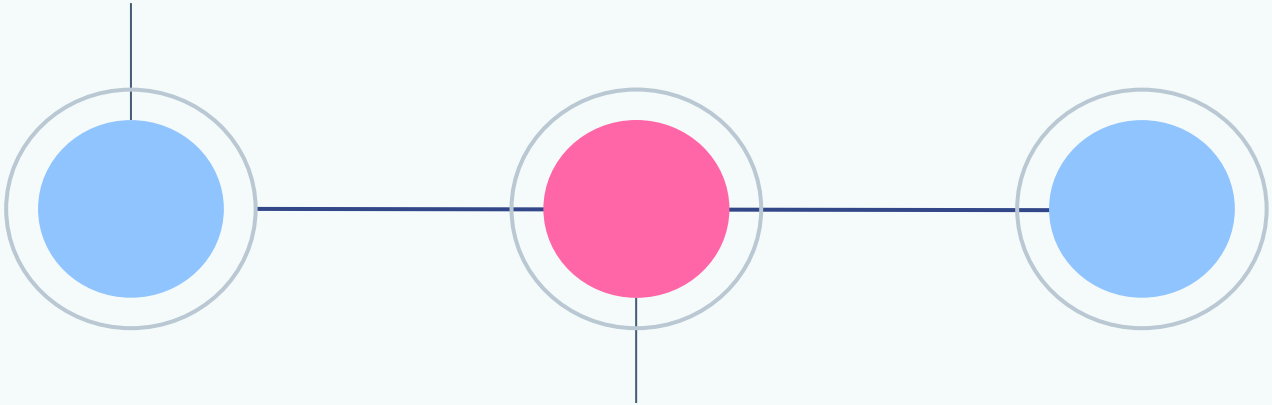
Structure of nodes
are **unified**




Internal Node
B-Tree
Leaf Node
Key
Pointer
Value
B+Tree

Why B+Tree ?

Structure of nodes
are **unified**



Increase branching factor:
reduce height of tree
comparing with B-Tree

- **Key's size**: 4 bytes
- **Value pointers**: 8 bytes
- **Node pointers**: 8 bytes

B-TREE

	NUM NODES	NUM KEYS	NUM VALUE POINTERS	NUM NODE POINTERS
ROOT	1	50	50	51
LEVEL 1	51	2550	2550	2601
LEVEL 2	132651	130050	130050	–

B+TREE

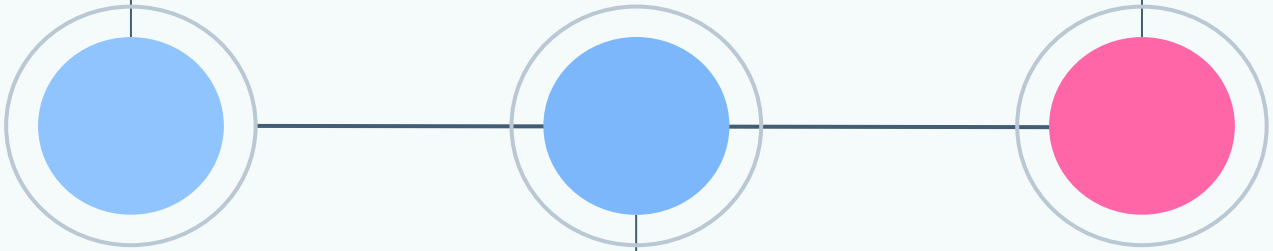
- **Key's size**: 4 bytes
- **Value pointers**: 8 bytes
- **Node pointers**: 8 bytes

	NUM NODES	NUM KEYS	NUM NODE POINTERS
ROOT	1	84	85
LEVEL 1	85	7140	7225
LEVEL 2	7225	599675	614125

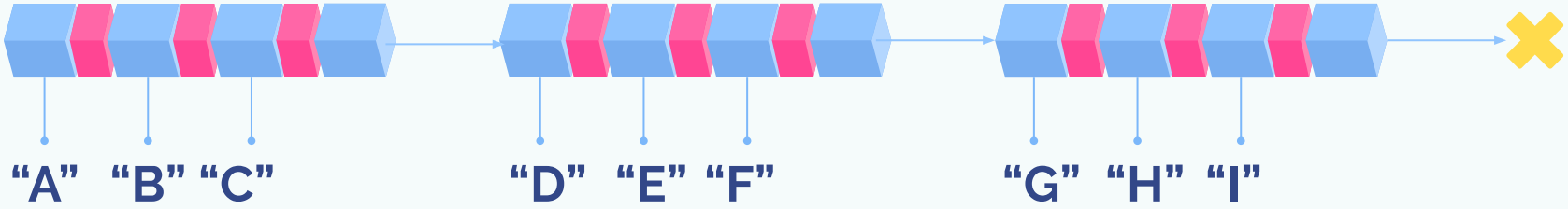
Why B+Tree ?

Structure of nodes are **unified**

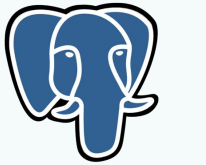
Increase **range queries performance** because leaf nodes are linked



Increase branching factor: reduce height of tree comparing with B-Tree


“A”
“B”
“C”
“D”
“E”
“F”
“G”
“H”
“I”

B+Tree's applications



Implement Key-Value Store with B+TREE

03

FILES

DATABASE

Store Key-Value entries

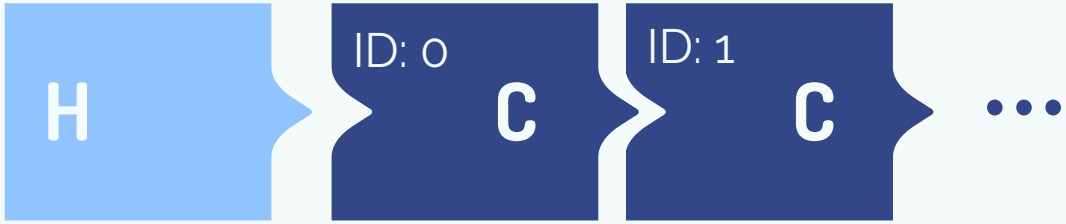
DELETED NODES

Offsets of deleted nodes

DELETED VALUES

Offsets of deleted values

DATABASE FILE STRUCTURE



HEADER

Metadata for
Key-Value storage

CONTENT PAGE

2 type: **NODE PAGE** | **VALUE PAGE**

Fixed size

Header

Root's Offset : 8 byte

Last Value Offset : 8 byte

Page Size: 2 byte

Key Size: 2 byte

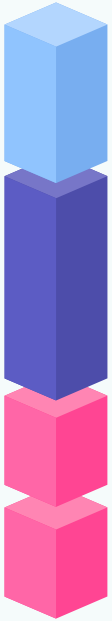
Value Size: 2 byte

Deleted Nodes File Name: 32 byte

Deleted Values File Name: 34 byte

88 bytes

Node page



Childs Offset: *8 x (2 x MaxDegree)* bytes

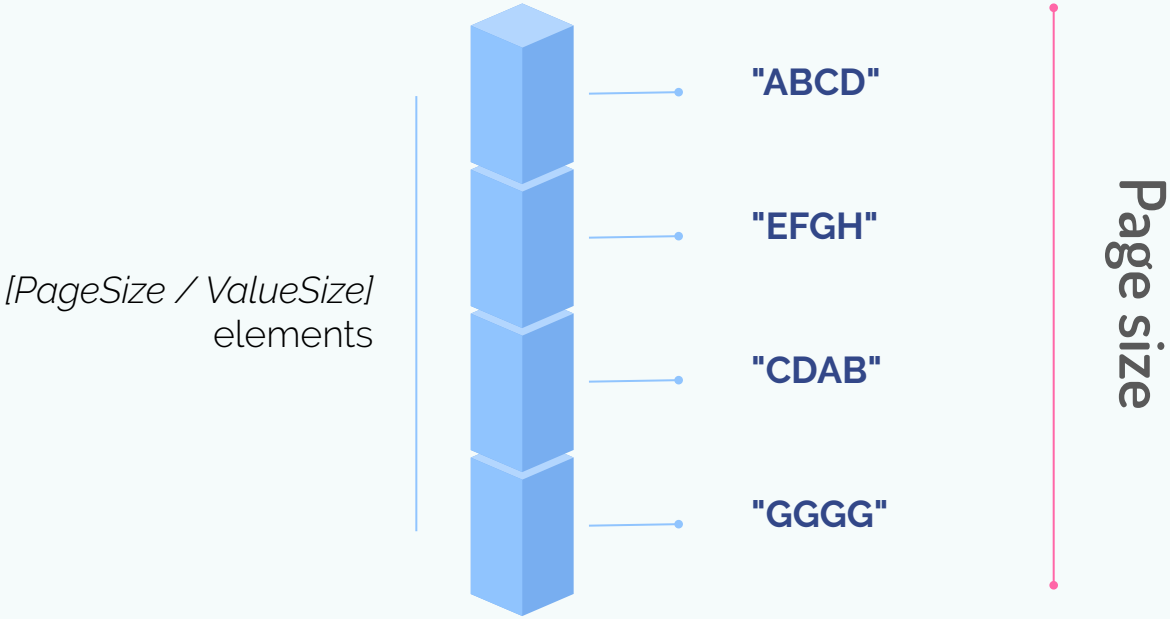
Keys: *KeySize x (2 x MaxDegree - 1)* bytes

Flag: 1 byte
LEAF | INTERNAL | REMOVED

Size: 2 bytes
Number of keys in node

Page size

Value page



Offset

INDEX OF VALUE IN NODE

bitSize(PageSize / ValueSize) bits

= 00 If it's NodePage

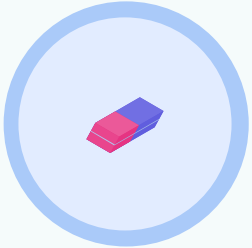
REAL NODE ID

64 - *bitSize(PageSize / ValueSize)* bits

FILES

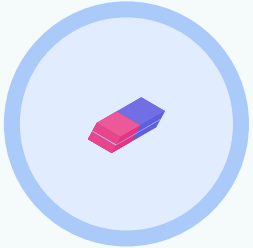
DATABASE

Store Key-Value entries



DELETED NODES

Offsets of deleted nodes



DELETED VALUES

Offsets of deleted values

DELETED FILE STRUCTURE

DEMO

04


Q/A

THANKS FOR LISTENING