



Lecture 13: Range Query & Ordered Index



Logistics

- Point Solutions App
 - Session ID: **database**
- Exercise Sheet 1 is due on Sep 30
- Exam on Oct 2

Recap

- Parallel Index Construction
- Fine-Grained Locking
- Shared Mutex
- Simulation Framework

Lecture Overview

- Range Query
- Ordered Index
- B+Tree Template

Range Query



Range Query vs Point Query

Range Query

Tuples where a
column is in a
Range of Values

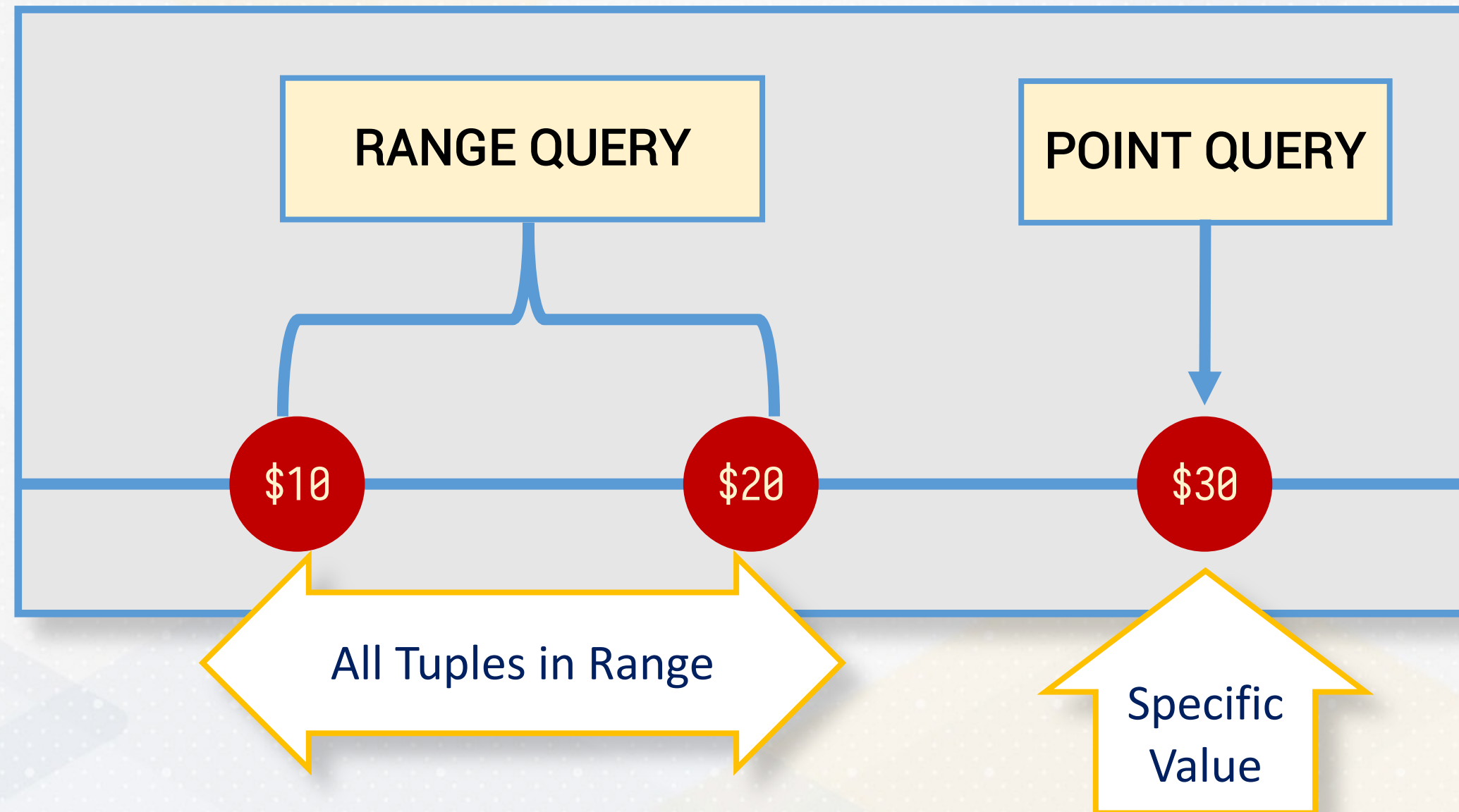
Products priced
between
\$10 and \$20

Point Query

Tuples where a
column is equal to
a Single Value

Products priced
\$30

Range Query vs Point Query



Point Query Using Hash Table Index

Look up a key in the hash table with $O(1)$ average complexity.

```
std::vector<int> tuple_ids = productIndex.getValue(30);
```

POINT QUERY



0	1	2	3	4	5	6	7	8	9
\$30		\$12			\$25		\$17		
[103, 105]		[104]			[101]		[106, 102]		

Point Query Using Hash Table Index

Hash Table

Not ordered based on key

Iterate through entire table

RANGE QUERY

0	1	2	3	4	5	6	7	8	9
\$30		\$12			\$25		\$17		
[103, 105]		[104]			[101]		[106, 102]		

Point Query Using Hash Table Index

```
std::vector<int> rangeQuery(int lowerBound, int upperBound) const {  
    std::vector<int> results;  
    for (size_t i = 0; i < capacity; ++i) {  
        if (hashTable[i].exists && hashTable[i].key >= lowerBound &&  
            hashTable[i].key <= upperBound) {  
            results.push_back(hashTable[i].value);  
        }  
    }  
    return results;  
}
```

Range Query in BuzzDB



buzzDB

```
void performRangeQueryWithHashIndex(int lowerBound, int upperBound) {  
    auto results = index.rangeQuery(lowerBound, upperBound);  
    std::cout << "Found " << results.size() << " records in the range.\n";  
}
```

Ordered Index



Ordered Index

RANGE QUERY **[\$10, \$20]**

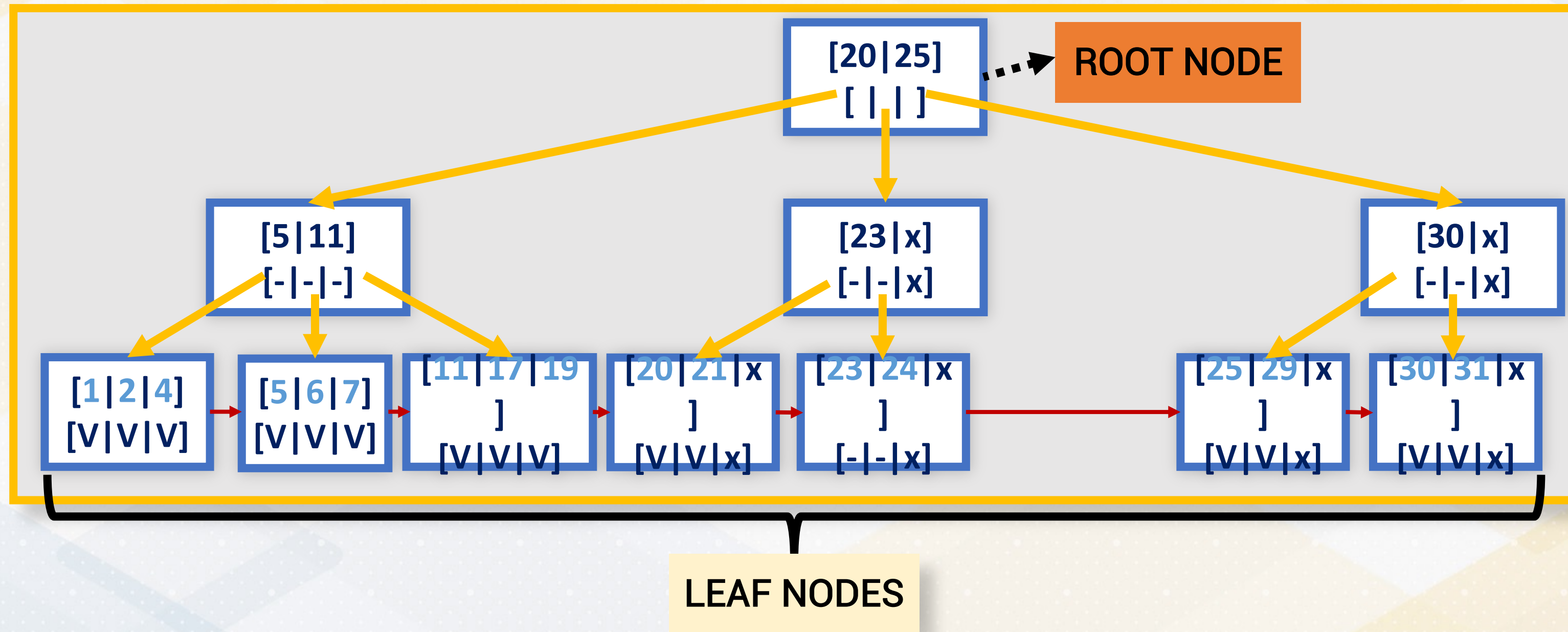
0	1	2	3	4	5	6	7	8	9
			\$12		\$17			\$25	\$30
			[104]		[106, 102]			[101]	[103, 105]

Unordered Index

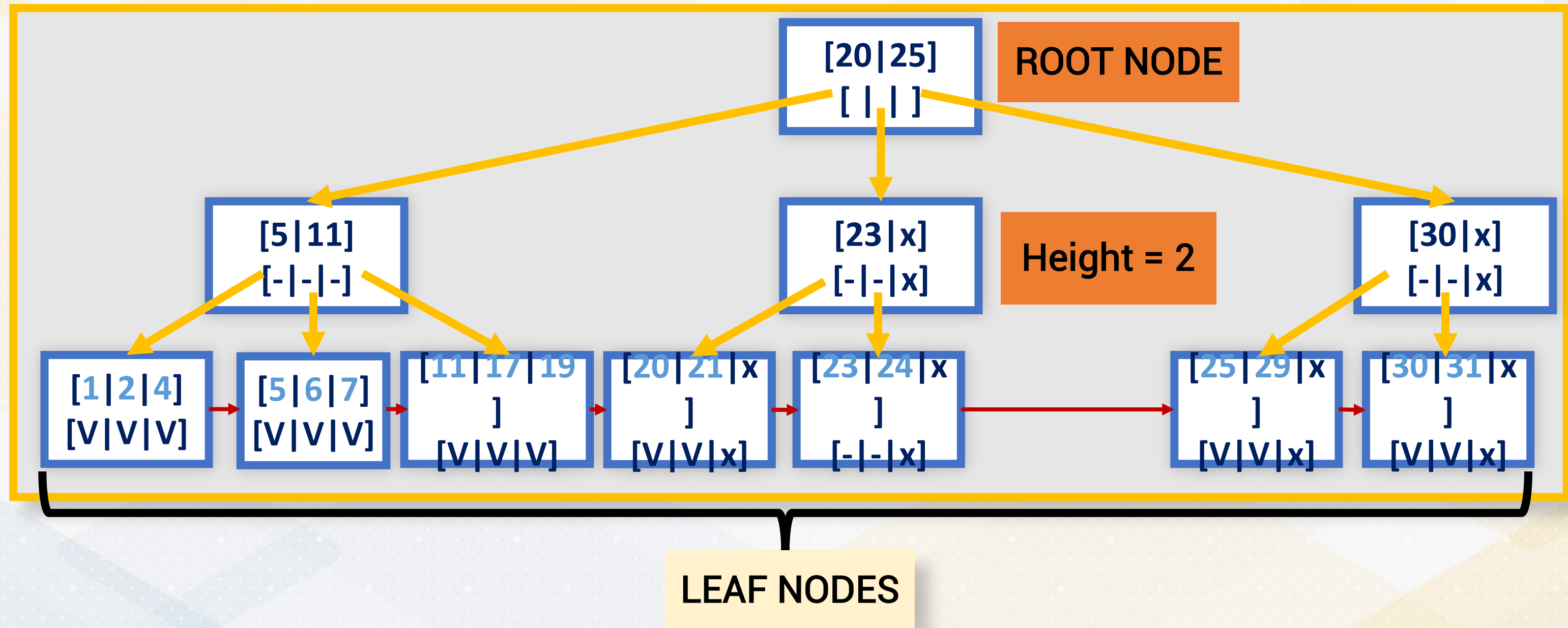
RANGE QUERY **[\$10, \$20]**

0	1	2	3	4	5	6	7	8	9
\$30		\$12			\$25		\$17		
[103, 105]		[104]			[101]		[106, 102]		

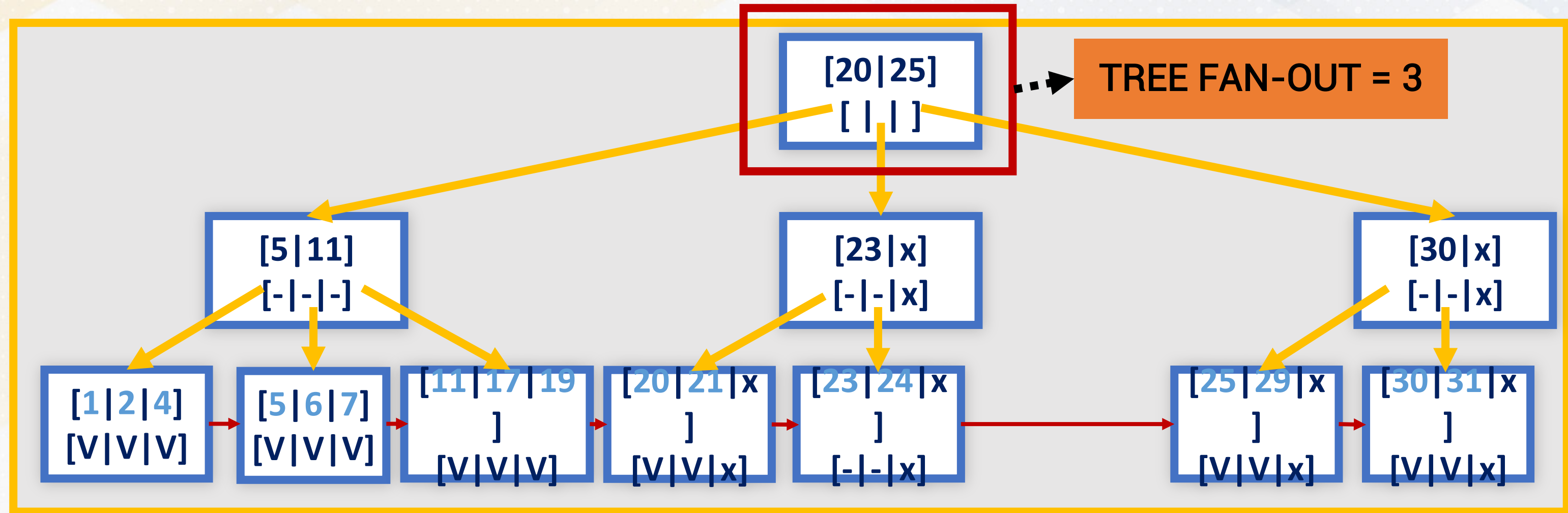
B+Tree Data Structure



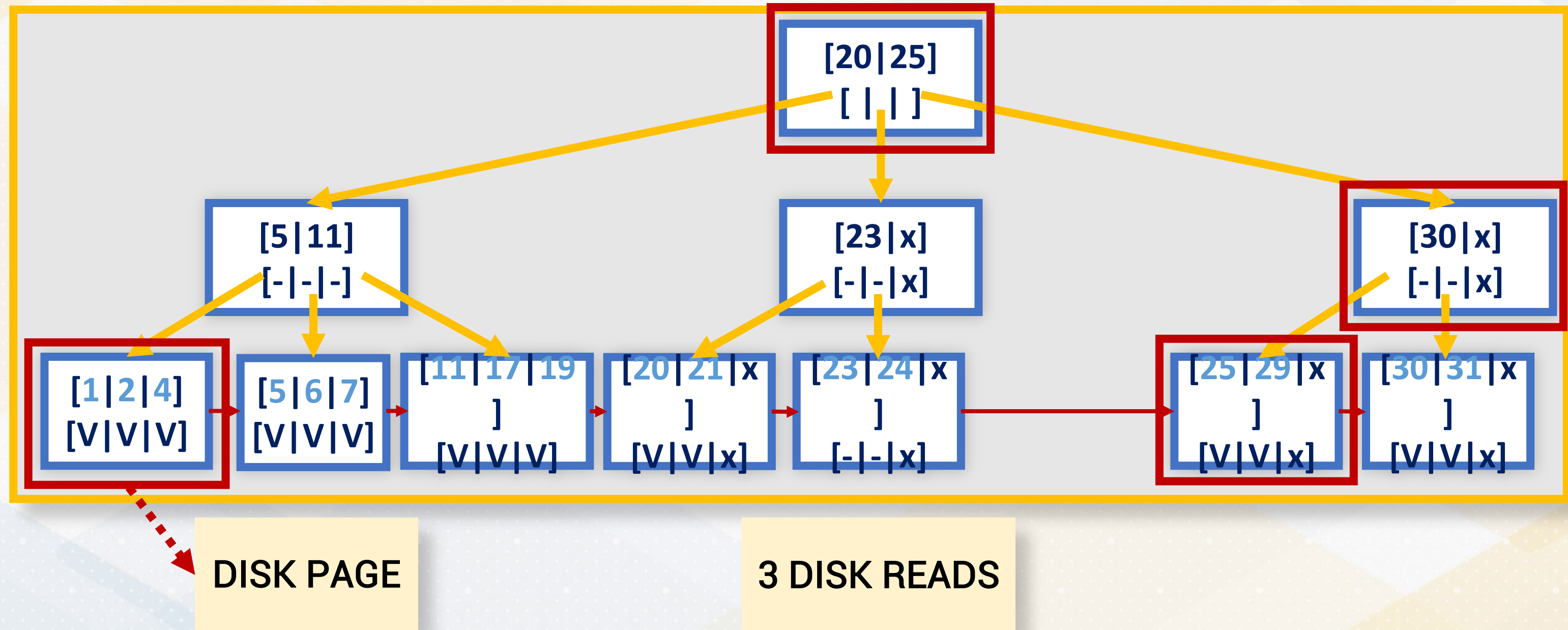
Height of the Tree



Fan-Out of the Tree

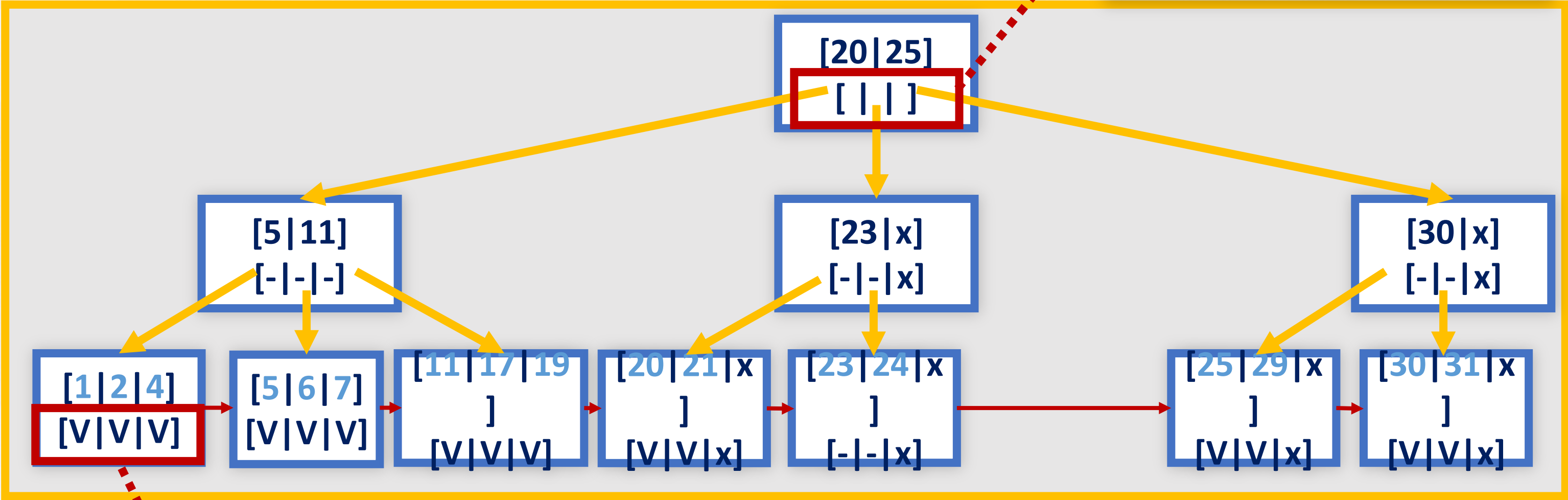


Disk Accesses



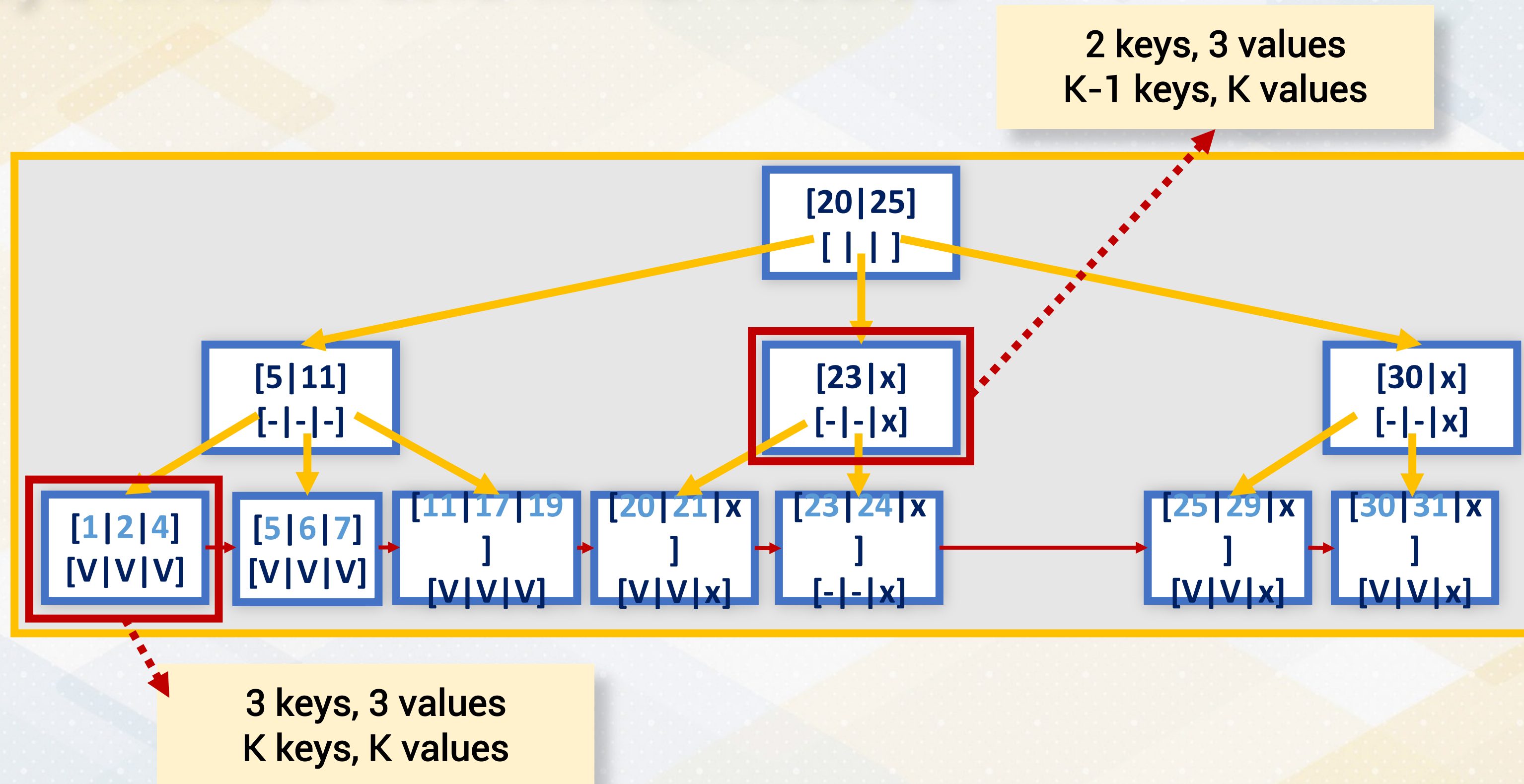
Values in Leaf and Inner Nodes

NODE POINTERS
Example: [Page 10]

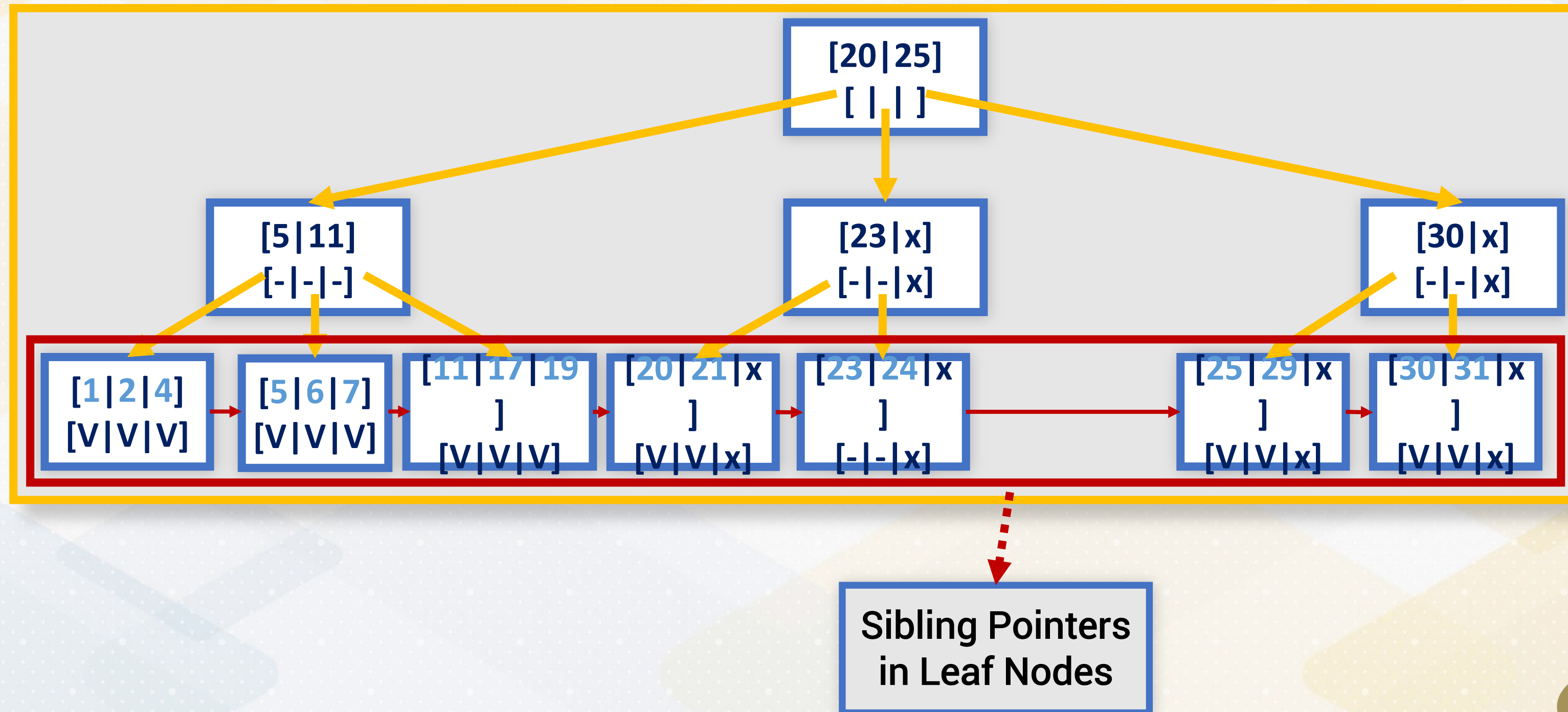


ACTUAL VALUES
Example: [Tuple 104]

Keys in Leaf and Inner Nodes



Leaf Sibling Pointers for Range Queries

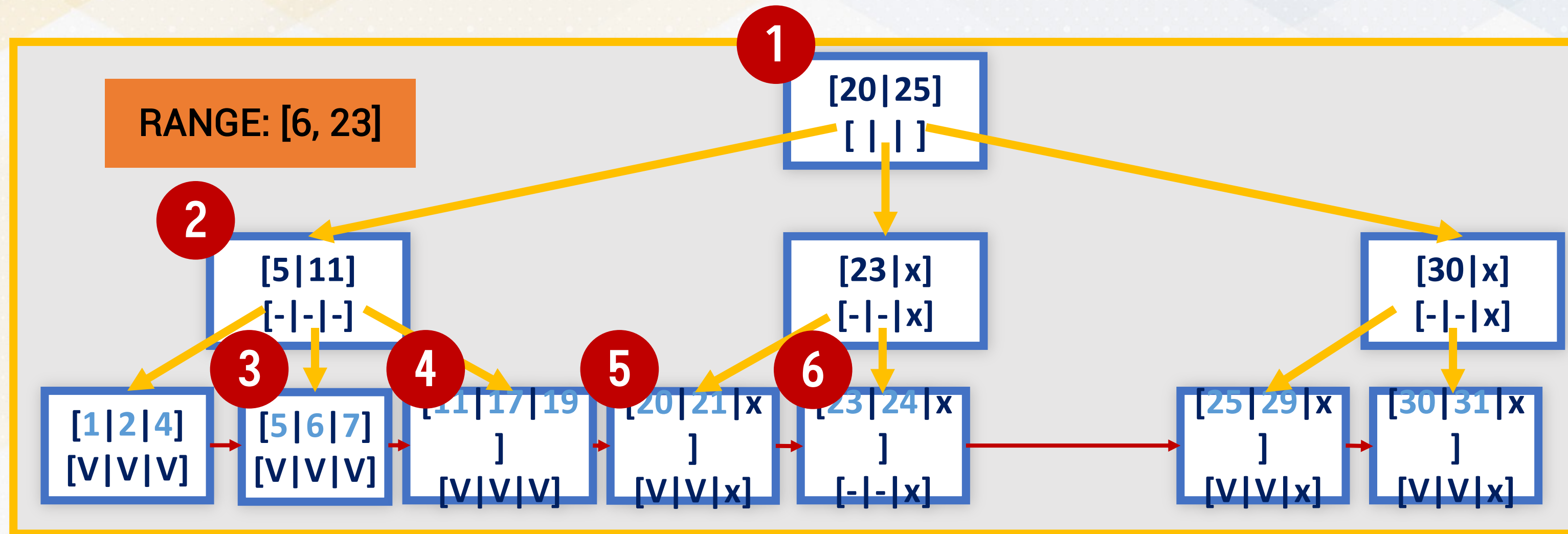


Fast Range Queries Using B+ Trees

The sorted nature and linked leaf nodes allow B+ trees to execute range queries by simply traversing the leaf nodes that fall within the range

```
std::vector<Value> rangeQuery(const Key &lowerBound, const Key &upperBound) const {  
    // Traverse to the correct leaf node and then follow the linked leaf nodes.  
    // While traversing the leaf nodes, collect all the key-value pairs in the range  
}
```

Fast Range Queries Using B+ Trees



B+Tree Template



Templates in C++

Templates

Functions & Classes
operate with
generic Types

```
template <typename T> T add(T a, T b) {  
    return a + b;  
}
```

Type Safety Ensured at Compile Time

Template Instantiation

Add Function Instantiated with Different Types

```
int result1 = add<int>(5, 3);  
// returns 8
```

```
std::string result3 = add<std::string>("Hello, ", "world!");  
// returns "Hello, world!"
```

B+Tree Template in C++

B+ Trees Use
C++
Templates

Placeholders
Replaced at
Compile Time

B+ Tree Handles
Any Key and
Value Types

Enhance Code
Reusability &
Maintainability

```
template <typename Key, typename Value> class BPlusTree {  
    struct Node; // Node definition  
    using NodePtr = std::unique_ptr<Node>;  
    // Node and Entry structures omitted for brevity  
};
```

B+Tree Template Instantiation #1

Example: Employee ID (int) → Salary (int)

```
BPlusTree<int, int> idToSalary;  
  
// Employee ID and salary are inserted into the tree.  
idToSalary.insertOrUpdate(1001, 50000);  
  
// Retrieve the salary of an employee by their ID quickly.  
int salary = idToSalary.getValue(1001);
```

B+Tree Template Instantiation #2

Example: Product Name (string) → Price (int)

```
BPlusTree<std::string, int> productToPrice;  
  
// Prices of products are stored in the B+ tree  
productToPrice.insertOrUpdate("Apple iPhone 12", 999);  
  
// Quickly find the price of a specific product.  
int price = productToPrice.getValue("Apple iPhone 12");
```

B+Tree Template Instantiation #3

Example: TicketKey (user-defined type) → Name (int)

```
struct TicketKey {  
    std::string eventDate;  
    int seatNumber;  
    // Overload the comparison operators for ordering in the B+ tree  
    bool operator<(const TicketKey &other) const {  
        if (eventDate == other.eventDate) {  
            return seatNumber < other.seatNumber;  
        }  
        return eventDate < other.eventDate;  
    }  
};
```

B+Tree Template Instantiation #3

Example: TicketKey (user-defined type) → Name (int)

```
BPlusTree<TicketKey, std::string> concertTickets;  
// When a ticket is booked, it is entered into B+ tree using its composite key.  
concertTickets.insertOrUpdate(TicketKey("2030-12-25", 101), "John Doe");  
concertTickets.insertOrUpdate(TicketKey("2030-12-25", 102), "Jane Smith");  
  
// Retrieve the booking information for a specific seat on a particular date.  
std::string ticketHolder =  
    concertTickets.getValue(TicketKey("2030-12-25", 101));  
std::cout << "Seat 101 is booked by: " << ticketHolder << std::endl;
```

Ordered Index Class in BuzzDB

We implement OrderedIndex class by encapsulating a BPlusTree

```
class OrderedIndex {
    BPlusTree<int, int> bptree;

public:
    void insertOrUpdate(int key, int value) {
        bptree.insertOrUpdate(key, value);
    }

    std::vector<int> rangeQuery(int lowerBound, int upperBound) const {
        return bptree.rangeQuery(lowerBound, upperBound);
    }
};
```

Conclusion

- Range Query
- Ordered Index
- B+Tree Template