Conf42 2024 | March 21 2024 | Online

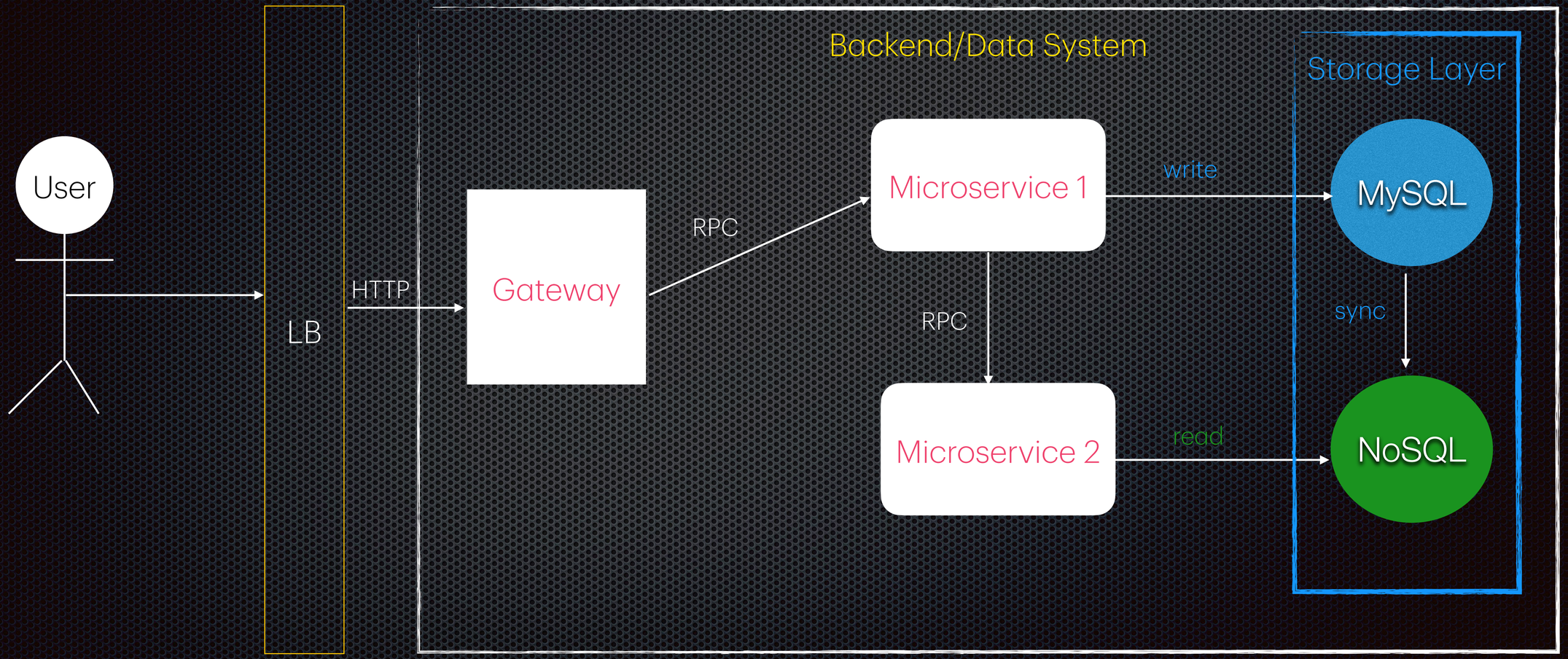
Architectural best practices for large-scale data systems

Santosh Nikhil Kumar Adireddy
Engineering Lead, ByteDance

Part 1

Storage Architecture & retrieval

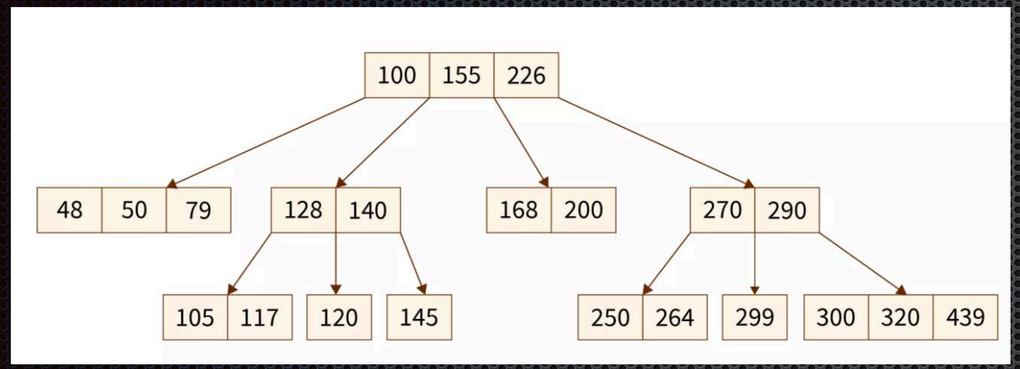
Where does Storage fit in the data systems



B-Trees vs QuadTrees vs LSM Trees vs R-Trees vs Inverted Index

- B-Trees:- Distributed email service like gmail, yahoo mail uses B-Trees for PostgreSQL (RDBMS)
- Quad-Trees:- Proximity service like yelp uses QuadTrees(MongoDB, PostGIS) for spacial indexing
- LSM-Trees (Log structured merge trees):- Services with write-heavy workload use databases with LSM-Trees (RocksDB). Digital wallet is an example. HDFS and Kafka are other examples
- Inverted index:- Information retrieval systems (search engines) like amazon.com elastic search service and Twitter's search

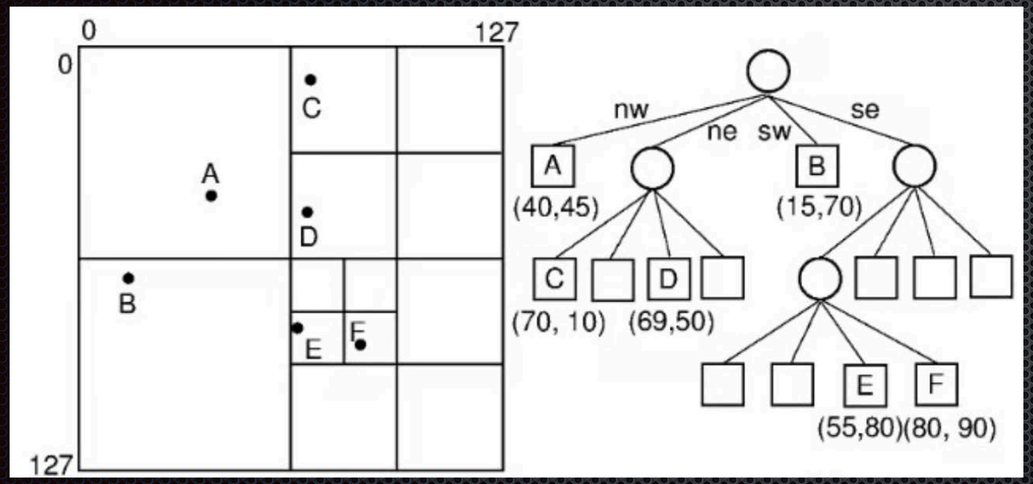
Why B-Trees in Distributed email service



- Efficient indexing: quick lookup, insertion, deletion. All have O(log n) time complexity
- Support for range queues: users often search for emails with a criteria (from a sender or time range)
- Disk-based storage optimization: data on secondary storage like hard drives(cheaper). B-Trees structure minimizes the disk I/O operations.

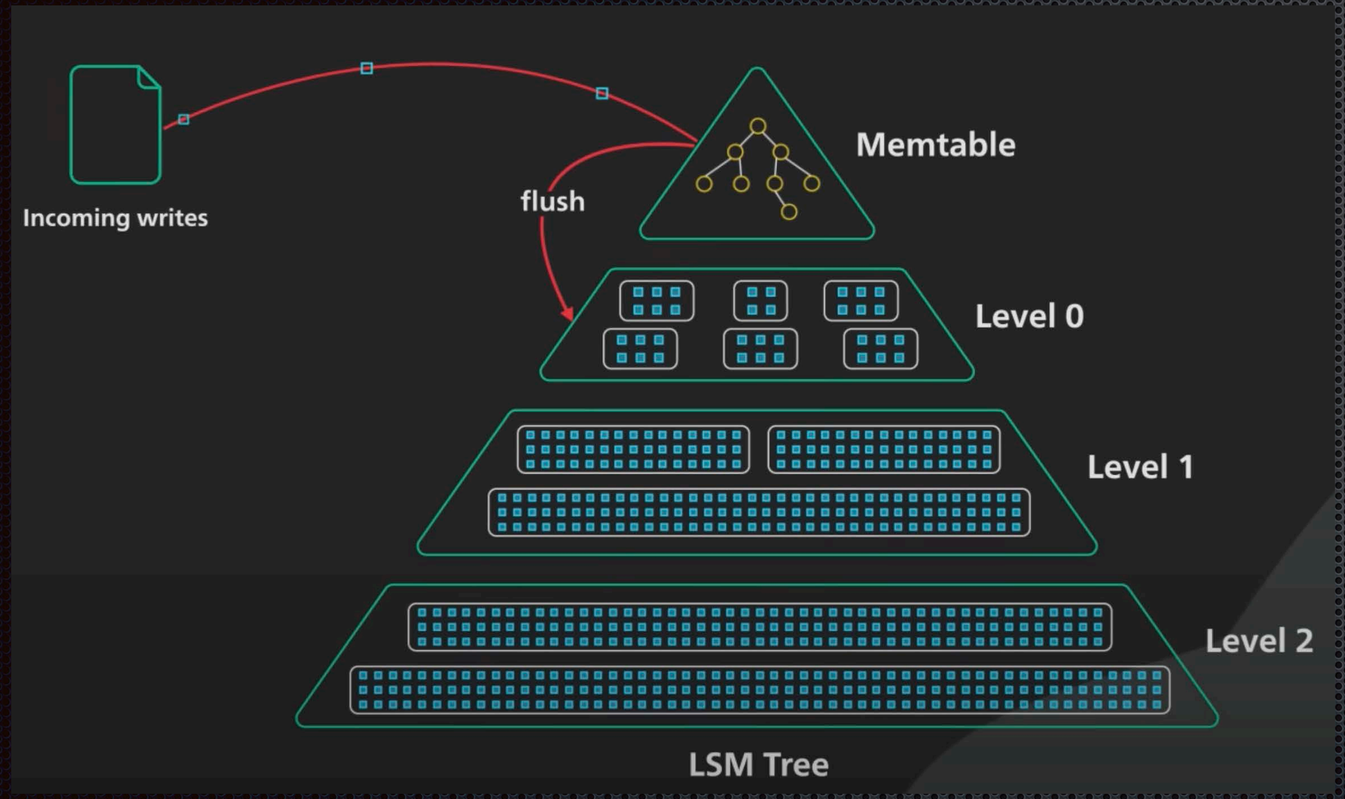
Why Quad-Trees in Proximity service

- Parent node = some region in the 2-D map
- 4 children of parent = 4 quadrants of parent region



- Spacial indexing: spacial data organized in hierarchical structure based on their coordinates
- Support for range queues: users search for nearby places. Prune irrelevant branches (O(log n))
- Adaptability to density: In high data density regions, quad-tree nodes are subdivided. In low data density regions, nodes are merged.

Why LSM-Trees in write-heavy system



- In-memory buffer: new data is added in in-memory buffer and later flushed to disk, minimizing disk I/O
- Compaction mechanism: periodic merging of data (immutable SS Tables) into single level, optimizing read performance (no need to lookup many SSTables)
- Tunable performance : buffer size, flush frequency, compaction thresholds

Why inverted index in search engines

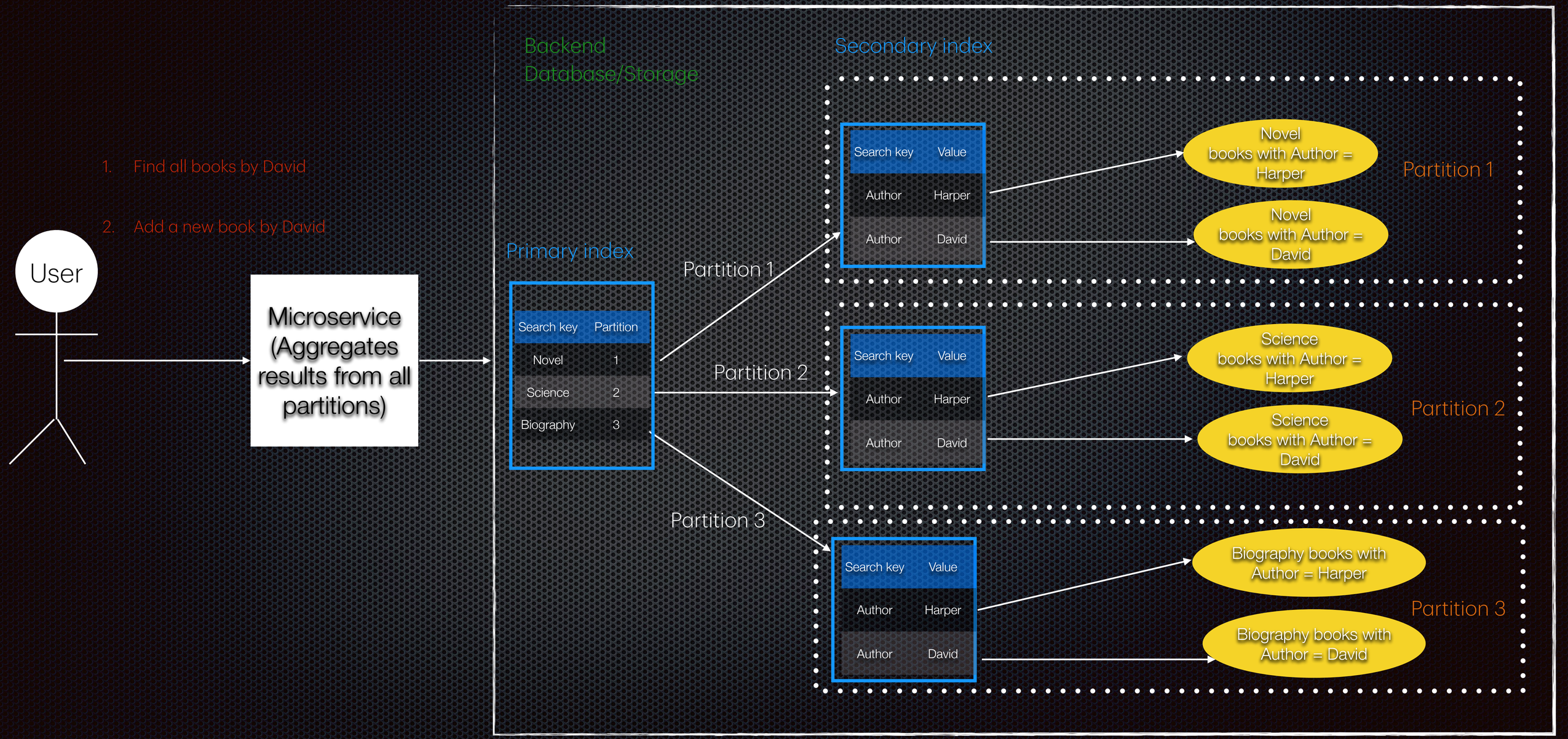
- We'll construct an inverted index for these documents :-
  - Document 1: "The quick brown fox jumps over the lazy dog."
  - Document 2: "A quick brown dog jumps over the lazy fox."
  - Document 3: "The lazy dog sleeps all day.”
- When a search query is submitted, the search engine can quickly look up the query terms in the inverted index to identify the relevant documents

Term	Documents
a	2
all	3
brown	1, 2
day	3
dog	1, 2, 3
fox	1, 2
jumps	1, 2
lazy	1, 2, 3
over	1, 2
quick	1, 2
sleeps	3
the	1, 2, 3

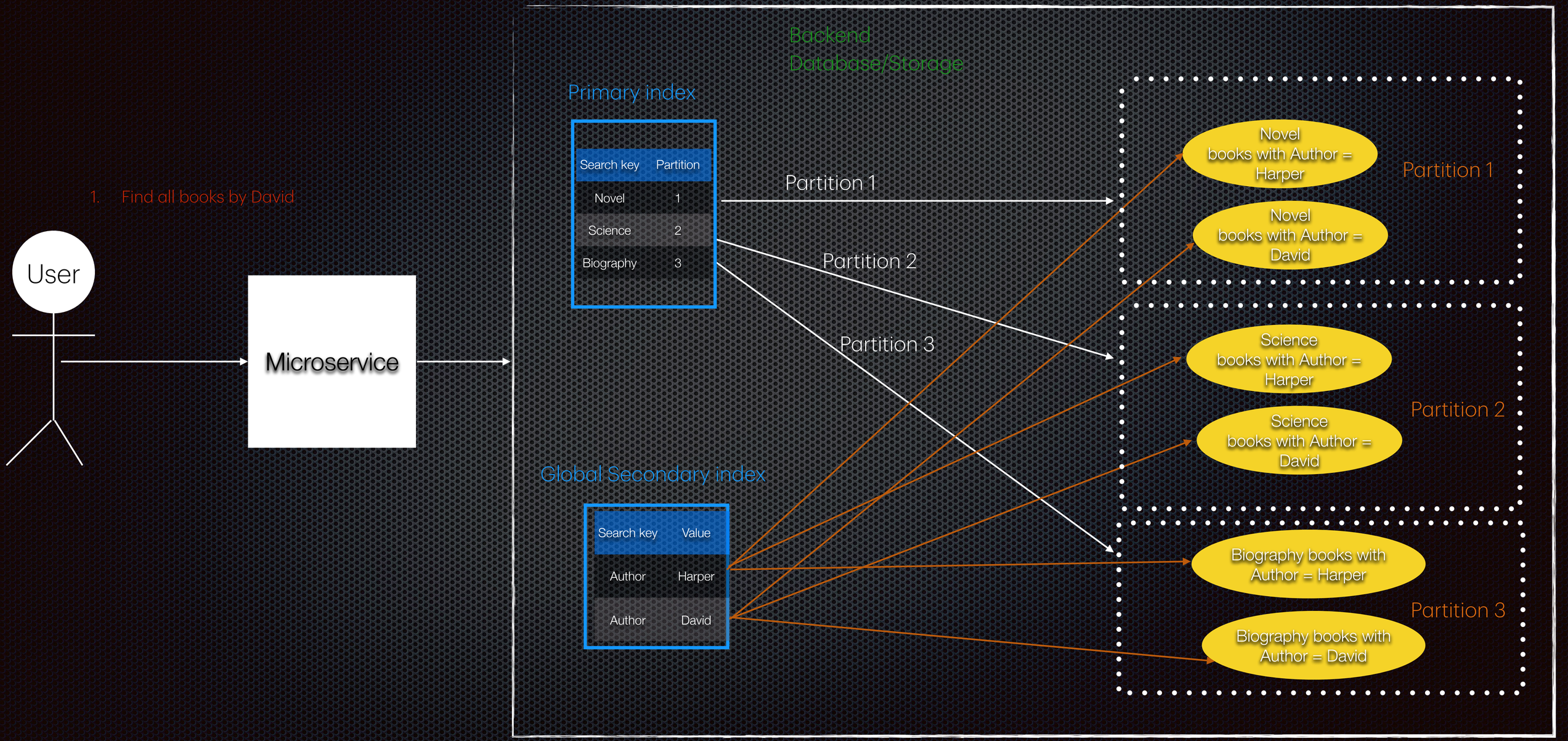
Part 2

Partitioning database:- Secondary indices

Partitioning Secondary index - Local



Partitioning Secondary index - Global



Local secondary index use case

- E-Commerce platform- DB partitioned by product categories like electronics, clothing, home goods
  - Customer wants to view all Sony products (local secondary index) in Electronics category
  - Data is needed from Electronics partition only

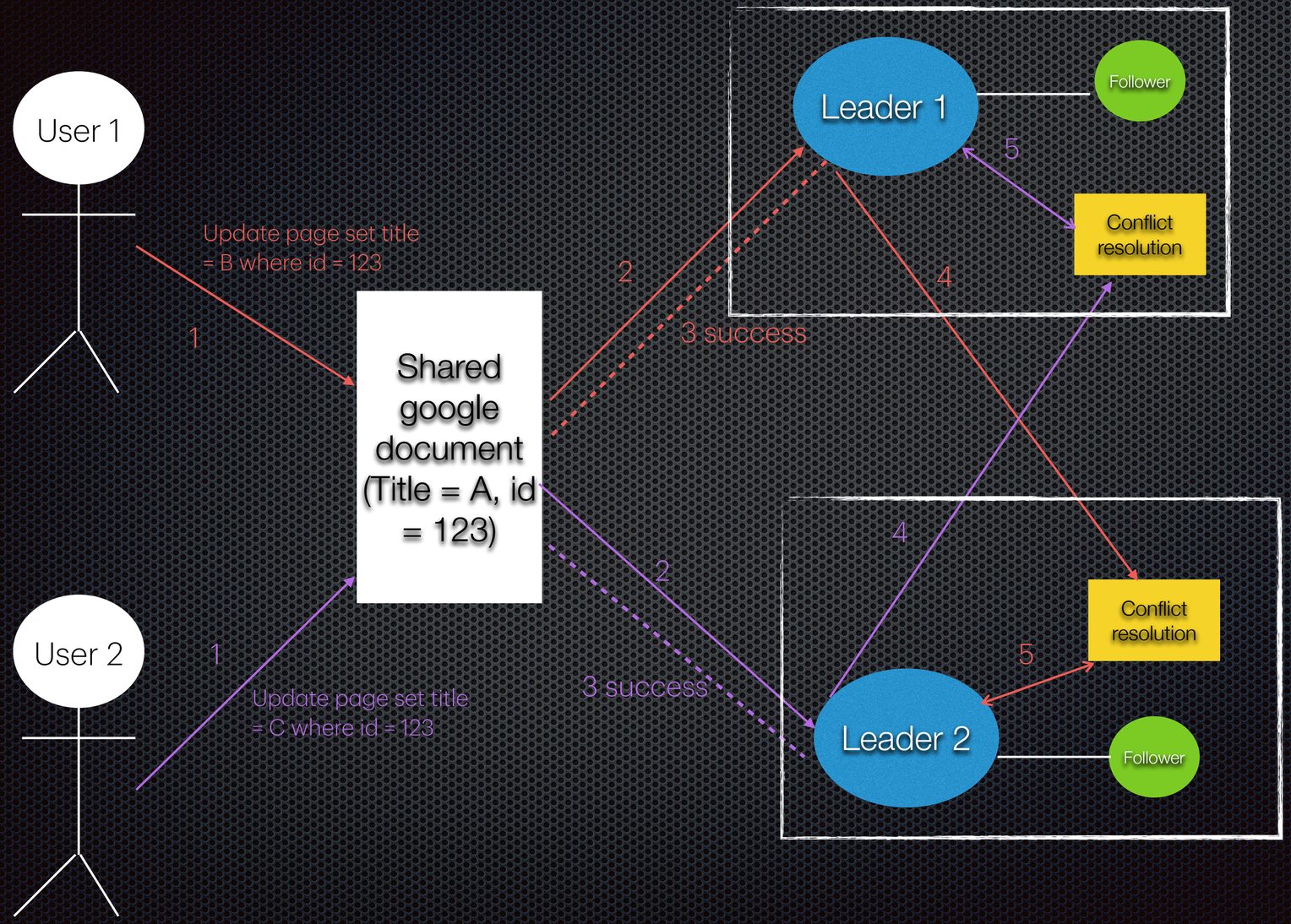
Global secondary index use case

- A multi-national company’s employee database - DB partitioned by country (USA, UK, Japan)
  - HR department wants to view all managers (global secondary index) across the company regardless of the country
  - Data is needed from all/multiple partitions

Part 3

Conflict free replicated datatypes (CRDTs)

Write conflict in Multi-leader replication



During steps 4 and 5 which are asynchronous replication, there are conflicts:-

Step 4, 5: Change id = 123, old = A, new = B . Can't change because title is now C

Step 4, 5: Change id = 123, old = A, new = C. Can't change because title is now B

Solution: Conflict free replicated datatypes

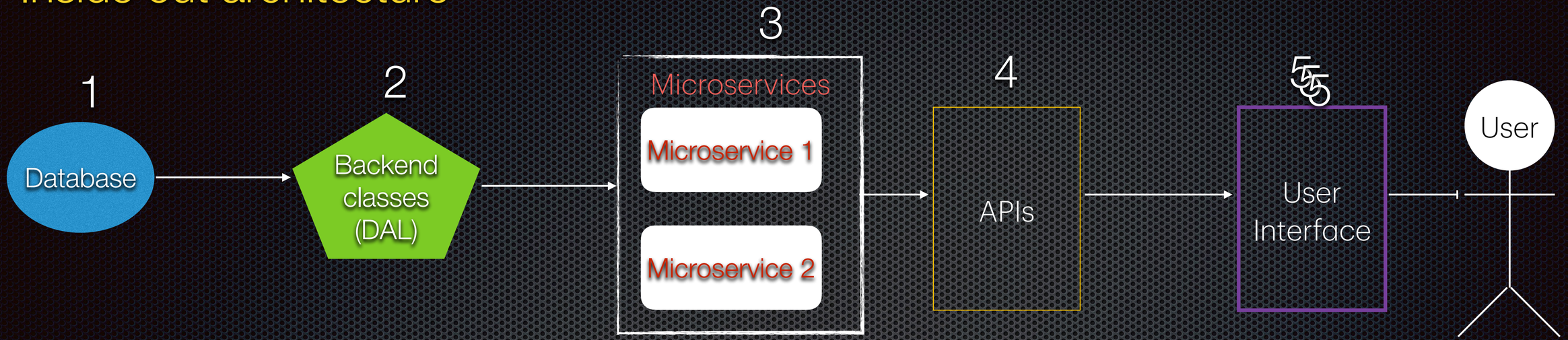
```java
import java.util.concurrent.atomic.AtomicInteger;

public class CounterCRDT {
    private AtomicInteger value;

    public CounterCRDT() {
        this.value = new AtomicInteger(0);
    }

    public void increment() {
        value.incrementAndGet();
    }

    public void decrement() {
        value.decrementAndGet();
    }

    public void merge(CounterCRDT other) {
        int otherValue = other.getValue();
        value.set(Math.max(value.get(), otherValue));
    }

    public int getValue() {
        return value.get();
    }

    public static void main(String[] args) {
        CounterCRDT counter1 = new CounterCRDT(); // Leader 1
        CounterCRDT counter2 = new CounterCRDT(); // Leader 2

        counter1.increment(); // Leader 1 increments by 1
        counter2.increment(); // Leader 2 increments by 1

        // Merge leaders
        counter1.merge(counter2); // Merge changes from leader 2 to leader 1

        System.out.println("Counter value after merging: " + counter1.getValue()); // Output: 2
    }
}
```

- A family of data structures for strings, sets, maps, ordered lists, counters, so on that can be concurrently edited by multiple users and they automatically resolve conflicts

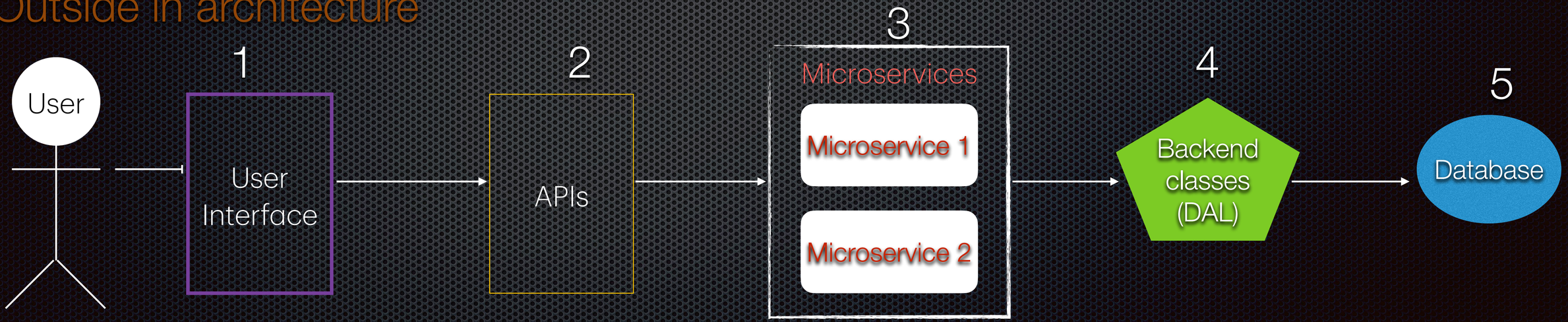
Part 4

Inside-out vs Outside-in architecture

Inside out architecture



Outside in architecture



Inside out

- Push strategy
- Forecast the demands of the UI needs
- Predictable and well-known problem
- Monolithic to MicroServices re-architecture , Domain driven like banking system

Outside in

- Pull strategy
- Generate demand through UI
- No historical data to predict the solution to the problem
- User-centric design, API driven development:- E-commerce platform