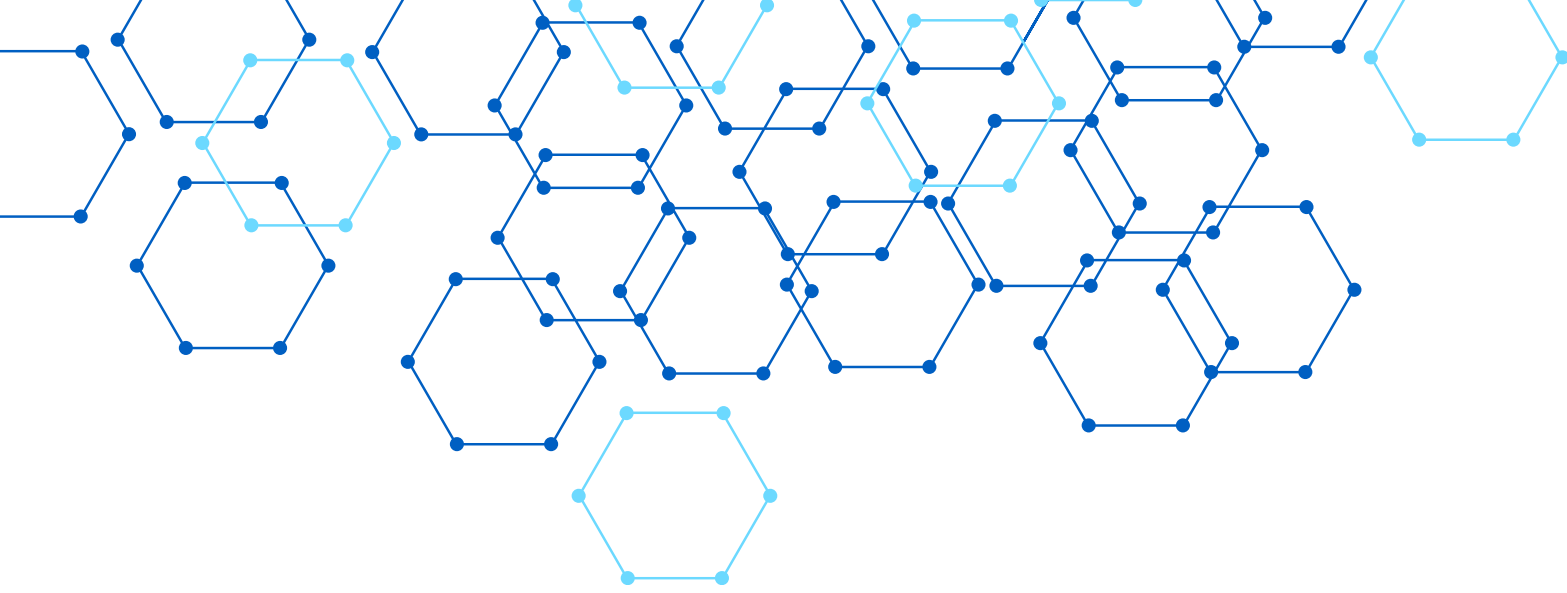


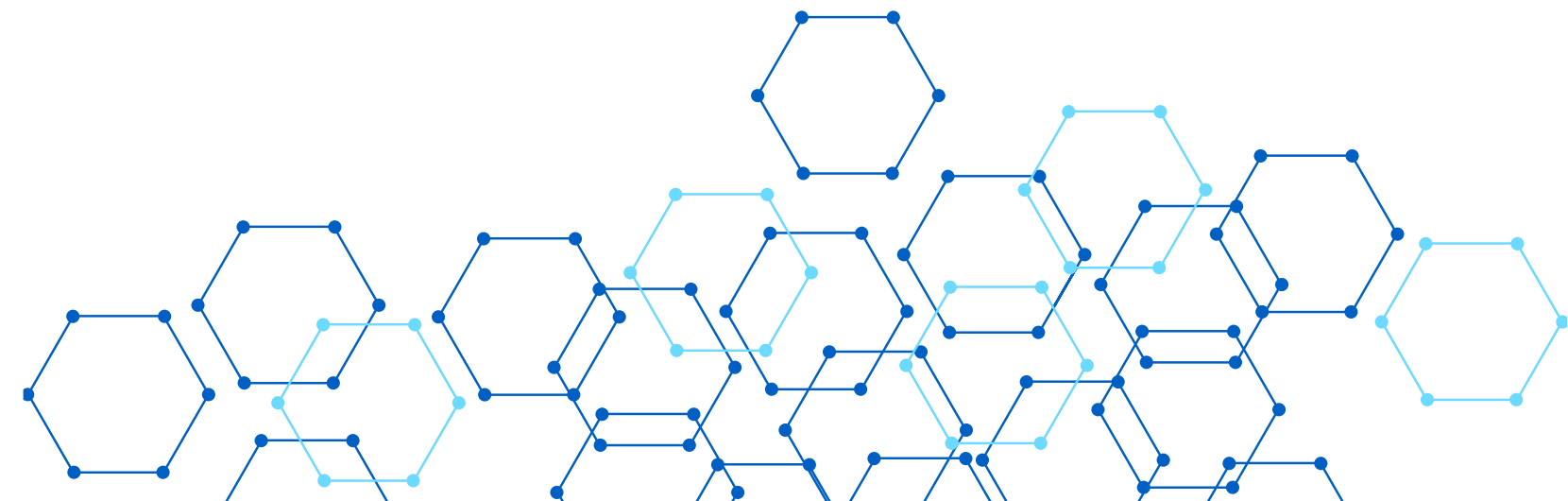


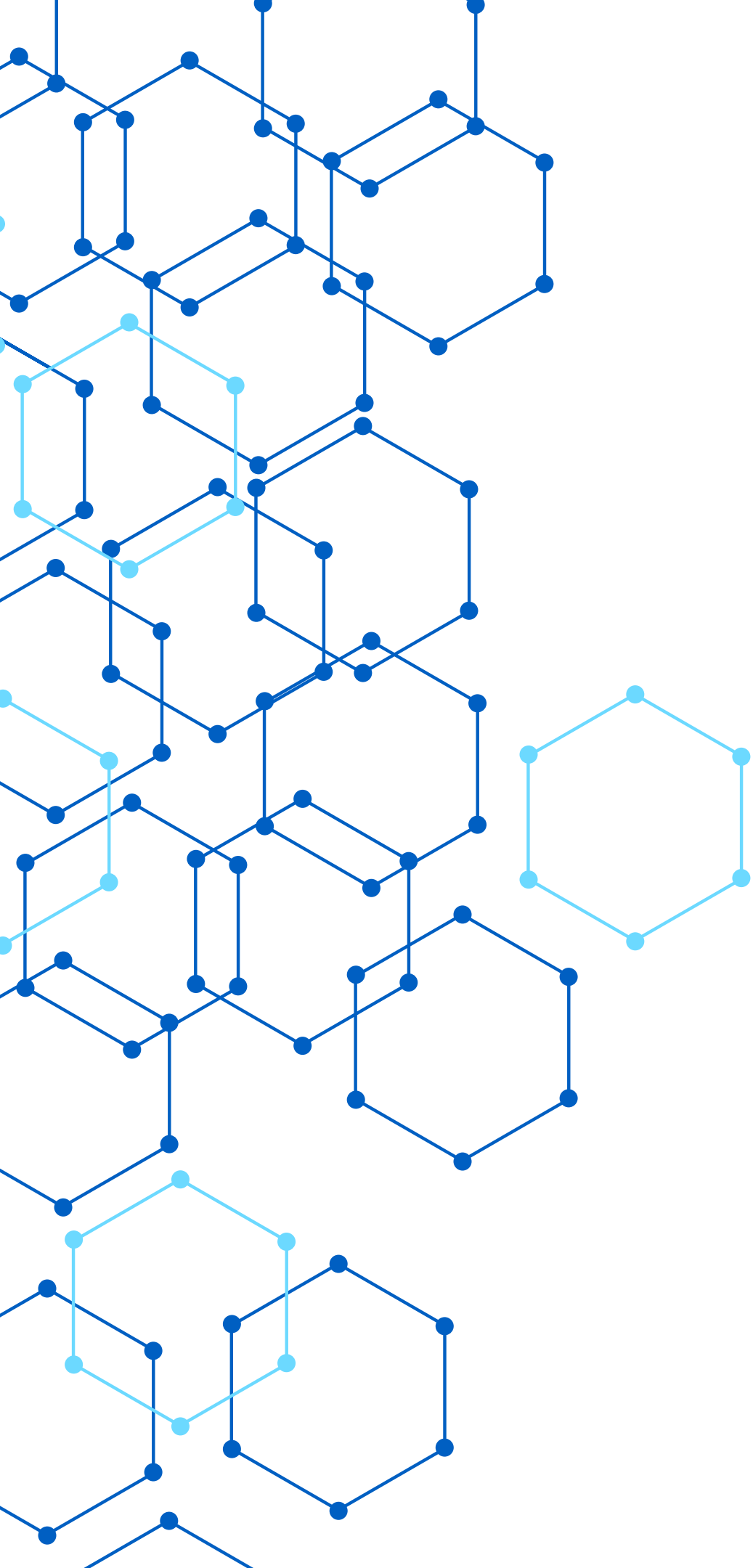
APACHE KAFKA SYSTEM DESIGN

High-Throughput Event Streaming with Scalability
& Partition Tolerance

Aqsa Israr | System Design - Capstone Project

[Google drive Capstone link](#)





PROJECT OVERVIEW

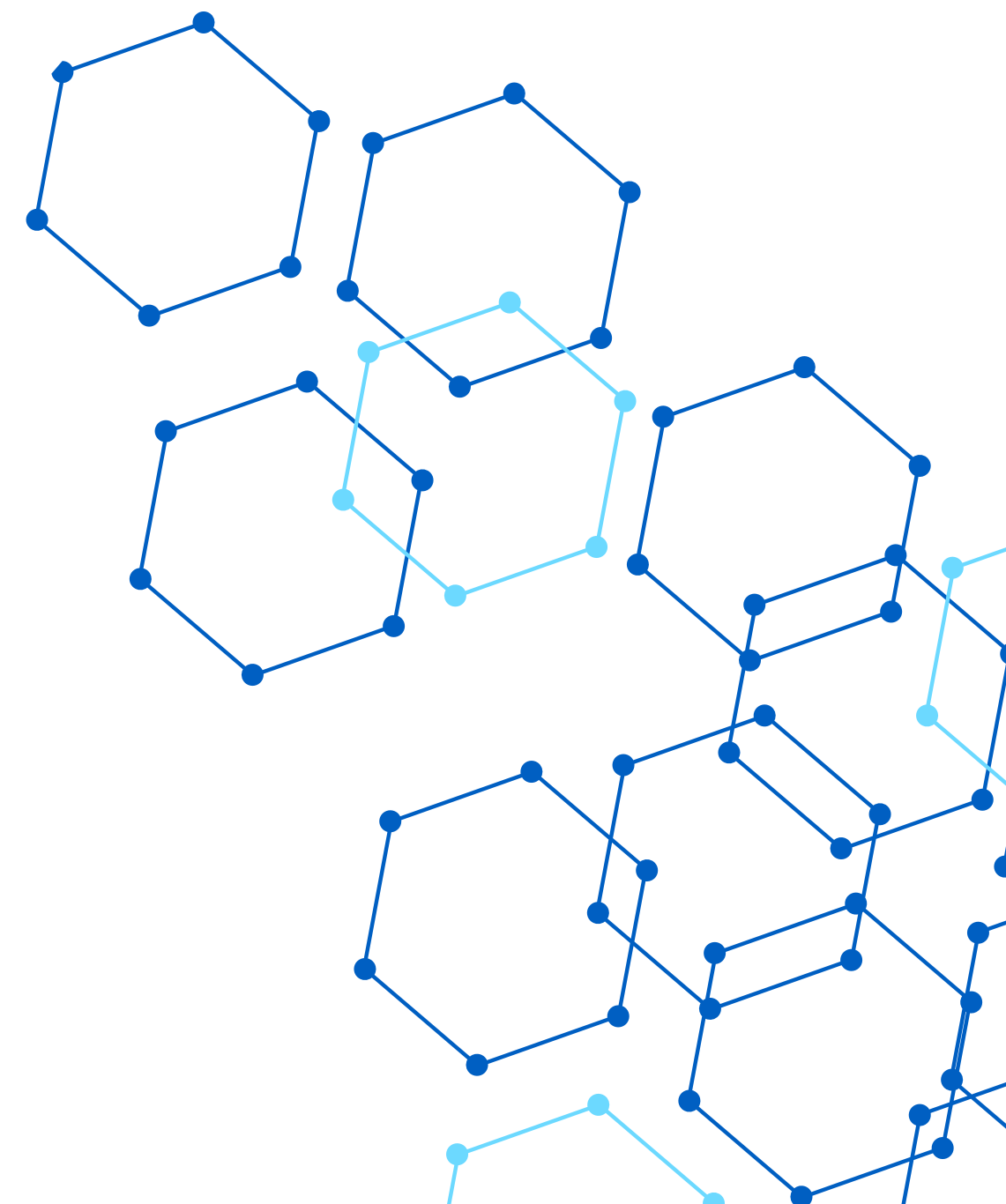
Apache Kafka is a distributed event streaming platform designed to move and retain large volumes of events between producers and consumers.

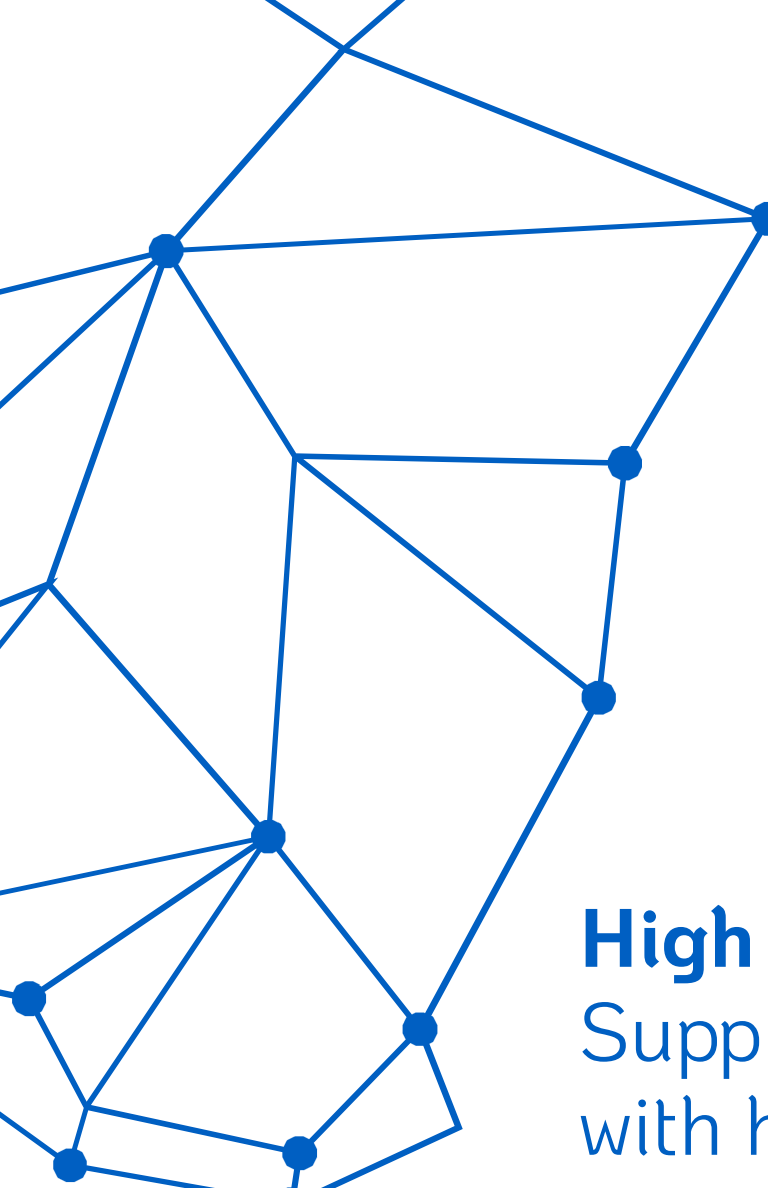
This capstone explores how Kafka achieves:

- High-throughput event ingestion and consumption
- Horizontal scaling using topics, partitions, and brokers
- Parallel processing through consumer groups
- Durability and fault tolerance through replication
- Ordered, replayable event streams

FUNCTIONAL REQUIREMENTS

- ✓ **Topic Management:** Administrators can create and configure topics, partitions, replication, and retention.
- ✓ **Publishing:** Producers can publish events to specific topics.
- ✓ **Partitioning & Ordering:** Events are distributed across partitions, with ordering guaranteed within each partition.
- ✓ **Consuming:** Consumers can subscribe to topics and fetch new events.
- ✓ **Consumer Groups & Offsets:** Consumers in a group share partition processing, while committed offsets track processing progress.
- ✓ **Retention & Replay:** Events remain stored for a configurable period and can be replayed when needed.





NON-FUNCTIONAL REQUIREMENTS



High Throughput & Scalability: Support 20K+ events/sec sustained, with horizontal scaling through brokers and partitions.



Low Latency: Target < 20 ms p95 publish acknowledgement under normal operating conditions.



High Availability & Durability: Tolerate broker failures without losing acknowledged events.



Delivery Semantics: At-least-once delivery by default, with stronger guarantees available.

ASSUMPTIONS & SCALE

Traffic Assumptions

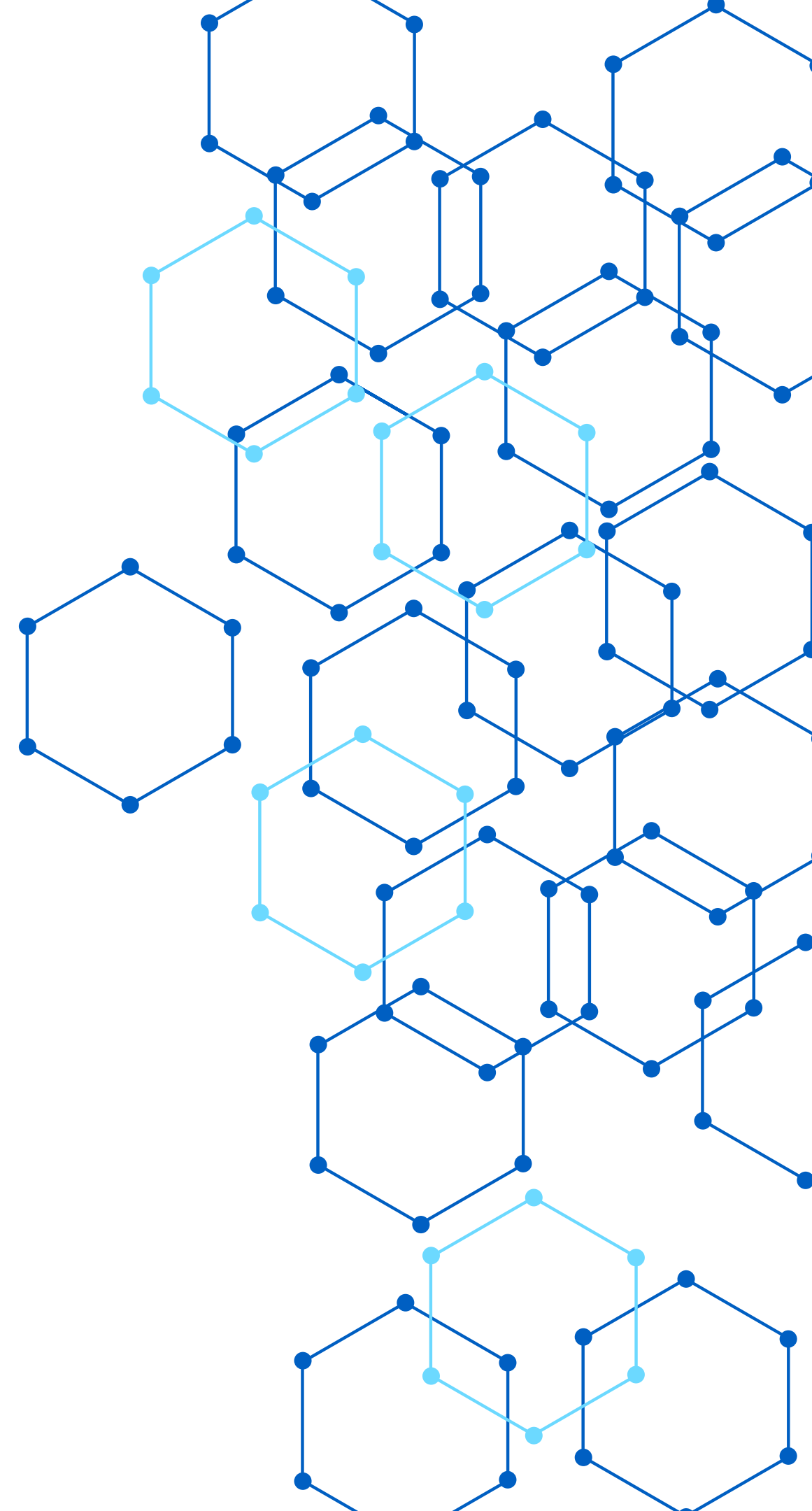
- 12M active event sources
- 144 events/source/day - approximately 1 event every 10 minutes
- $12\text{M sources} \times 144 \text{ events} = 1.728\text{B events/day}$
- $1.728\text{B events/day} \approx 20\text{K events/sec average}$
- 50% during the busiest 4 hours $\approx 60\text{K events/sec peak}$
- 20% annual traffic growth

Consumer Assumptions

- 3 independent consumer groups per event stream

Cluster/Data Assumptions

- 7-day default retention
- Replication factor = 3
- 3 Availability Zones



PRODUCER THROUGHPUT & SCALE



1. Average Throughput:

$1.728\text{B} \div 86,400$
 $\approx 20\text{K events/sec}$

$20\text{K} \times 1\text{ KB}$
 $\approx 20\text{ MB/sec ingress}$



2. Peak Throughput:

50% of 1.728B events in 4 hours
 $= 864\text{M events}$

$864\text{M} \div 14,400$
 $\approx 60\text{K events/sec}$

$60\text{K} \times 1\text{ KB}$
 $\approx 60\text{ MB/sec ingress}$



3. Daily Event Volume:

$12\text{M sources} \times 144\text{ events/source/day}$
 $= 1.728\text{B events/day}$

Design Target: Sustain 20K+ events/sec and absorb bursts up to 60K events/sec.

CONSUMER THROUGHPUT & FAN-OUT



1. Fan-Out Assumption

- 3 independent consumer groups per event stream
- Each group consumes the complete stream independently



2. Average Consumer Throughput

$20\text{K events/sec} \times 3 \text{ groups}$
 $= 60\text{K event deliveries/sec}$

$60\text{K} \times 1 \text{ KB}$
 $\approx 60 \text{ MB/sec egress}$



3. Peak Consumer Throughput

$60\text{K events/sec} \times 3 \text{ groups}$
 $= 180\text{K event deliveries/sec}$

$180\text{K} \times 1 \text{ KB}$
 $\approx 180 \text{ MB/sec egress}$

Consumer traffic can exceed producer traffic because the same event may be read by multiple consumer groups.

STORAGE REQUIREMENT ESTIMATE



1. Raw Event Data / Day

$1.728\text{B events/day} \times 1\text{ KB/event}$
 $\approx 1.728\text{ TB/day}$



2. With Replication

Replication Factor = 3
 $1.728\text{ TB} \times 3$
 $\approx 5.184\text{ TB/day}$



3. Seven-Day Retention

$5.184\text{ TB} \times 7$
 $\approx 36.3\text{ TB}$

Base Replicated Storage: ~36.3 TB

**Provisioned Capacity: ~47 TB with 30% operational headroom—
before compression**

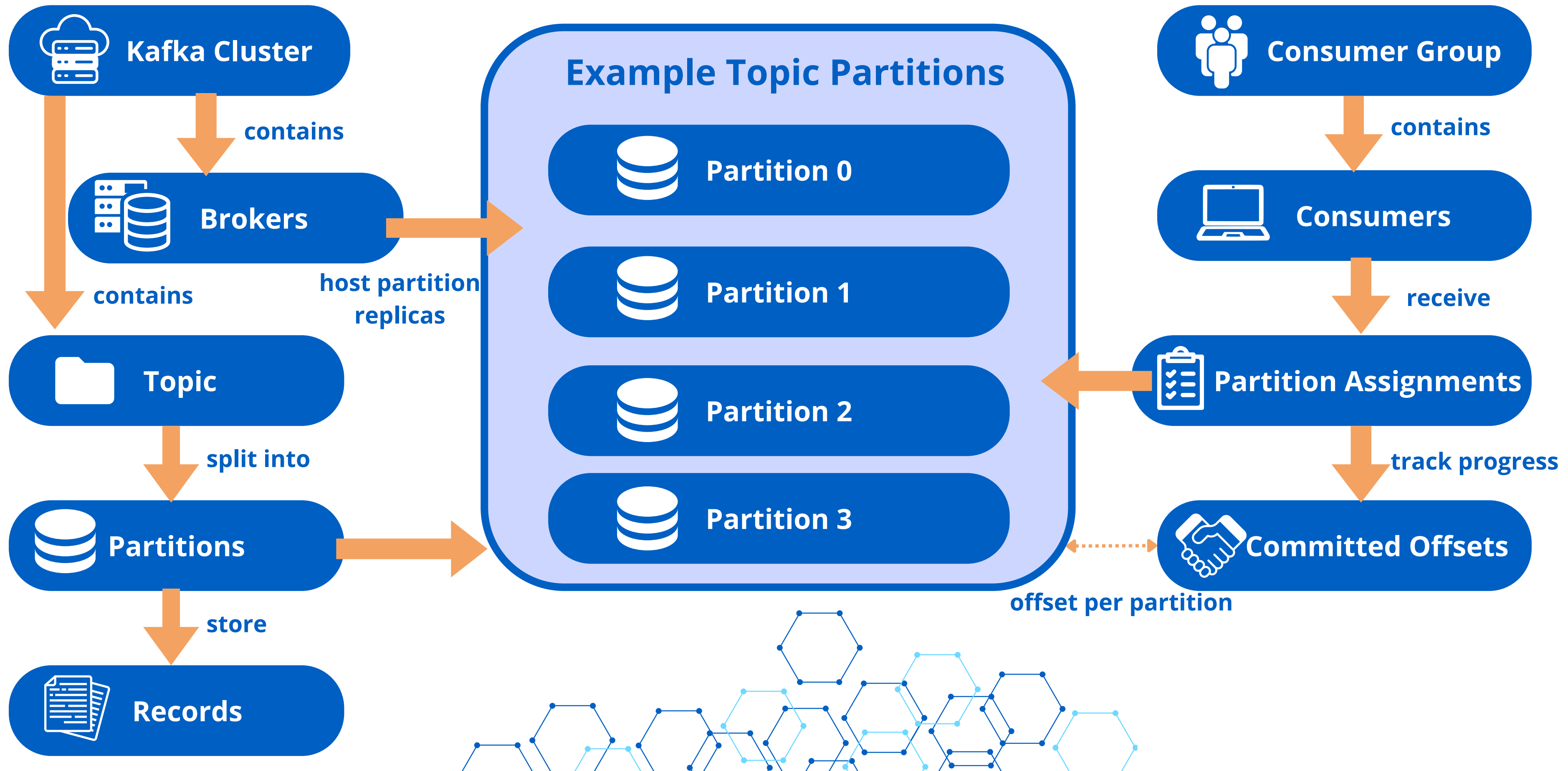


GROWTH ESTIMATES

Assumes 20% annual growth in active event sources and traffic

Year	Avg Producer Throughput	Peak Throughput	Base 7-Day Replicated Storage
Now	20K events/sec	60K events/sec	36 TB
1	24K	72K	44 TB
2	28.8K	86.4K	52 TB
3	34.6K	103.7K	63 TB
4	41.5K	124.4K	75 TB
5	49.8K	149.3K	90 TB

CORE KAFKA ENTITIES & RELATIONSHIPS



KAFKA ADMIN & PRODUCER OPERATIONS

1

CREATE TOPIC

CreateTopic(topicName,
partitionCount,
replicationFactor,
retention)

Request:

topic: "user_events"
partitions: 8
replication_factor: 3
retention: 7 days

Response:

status: CREATED
topic: "user_events"

2

PRODUCE EVENT

Produce(topicName,
key, payload)

Request:

topic: "user_events"
key: "user-123"
payload: {...}

Response:

partition: 4
offset: 1042
status: ACKNOWLEDGED



Simplified logical Kafka client operations.

KAFKA CONSUMER OPERATIONS

1

FETCH EVENTS

Fetch(topicName,
consumerGroupId,
maxBatchSize)

Request:

topic: "user_events"
group_id: "analytics"
max_batch_size: 500

Response:

partition: 4
records: [...]
next_offset: 1100

2

COMMIT OFFSET

CommitOffset(topicName,
consumerGroupId,
partitionId, offset)

Request:

group_id: "analytics"
topic: "user_events"
partition: 4
offset: 1100

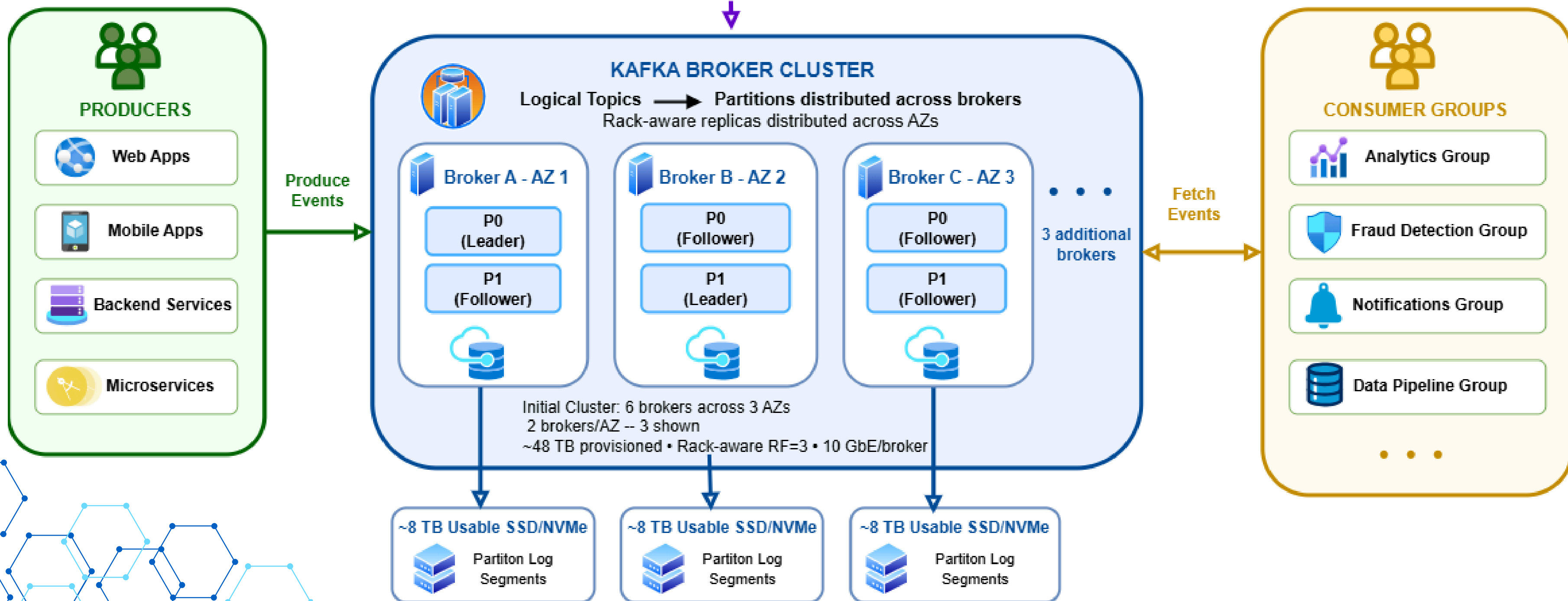
Response:

status: COMMITTED



Simplified logical Kafka client operations.

HLD



PARTITIONING & ORDERING STRATEGY



Partition Key: Related events with the same key, such as `user_id`, are routed to the same partition.



Ordering: Kafka preserves record order within a partition, not across the whole topic.



Parallelism: Multiple partitions allow producers and consumers to process events concurrently.



Trade-off: A poorly distributed key can create hot partitions and uneven load.



Partition 2
(Ordered log)

Offset

Event (key = user-123)

101

Login

102

View Product

103

Add to Cart

104

Purchase

105

Logout

...

PARTITION SIZING & CAPACITY PLANNING

Planning assumption: ~10 MB/sec per partition

Normal Load

20K events/sec × 1 KB event
≈ 20 MB/sec

20 MB/sec ÷ 10 MB/sec
= 2 partitions minimum for
average traffic

Peak Load

60K events/sec × 1 KB event
≈ 60 MB/sec

60 MB/sec ÷ 10 MB/sec
= 6 partitions minimum for
peak traffic

Minimum for current peak: 6 partitions

Initial provisioning decision: 8 partitions

Reason: Supports current 60 MB/sec peak + Year 1 peak of ~72 MB/sec

Scale trigger: Add capacity before peak traffic exceeds the initial 80 MB/sec partition capacity.



BROKER STORAGE DESIGN



Append-Only Log

Each partition is stored as an ordered log. New records are appended to the end instead of updating existing data.



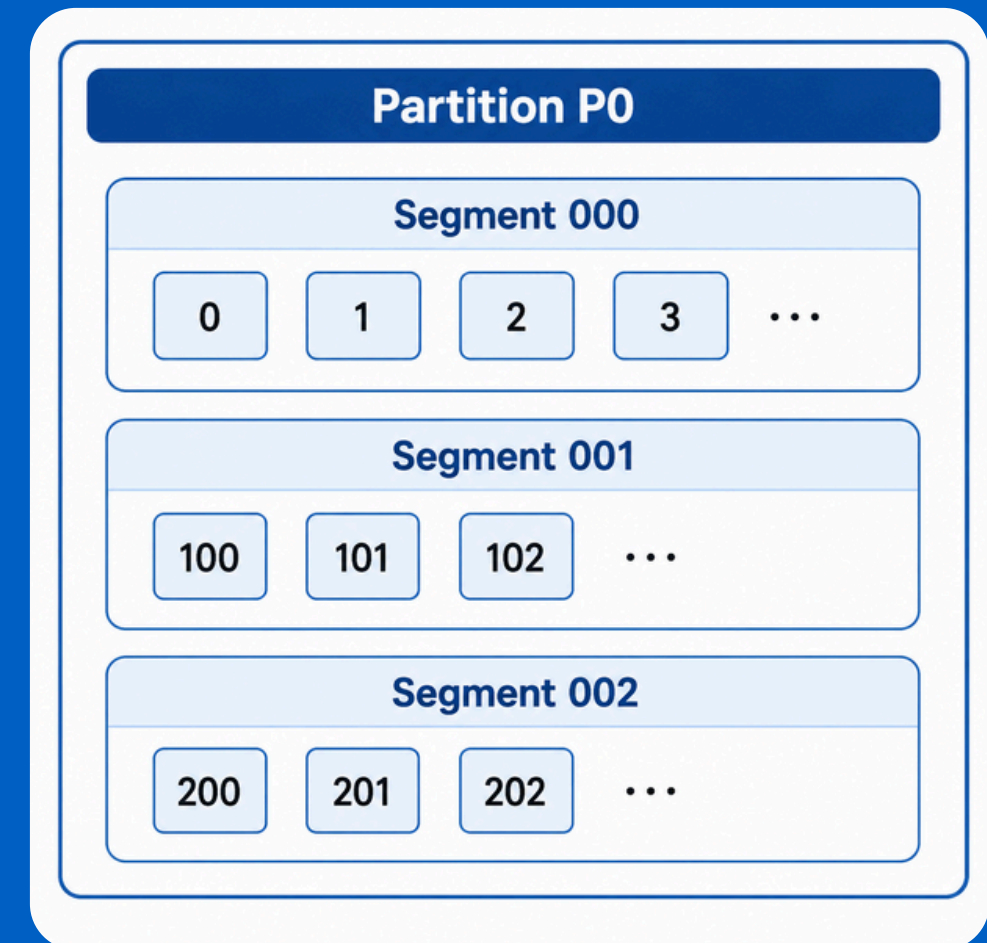
Log Segments

Large partition logs are split into smaller segment files, making retention and cleanup easier.



Offset Index

Each record receives a sequential offset. Each segment keeps a sparse index for faster record lookup.



HOW KAFKA ACHIEVES HIGH THROUGHPUT



OS Page Cache:

Broker writes first go to the Linux page cache in RAM. The OS flushes pages to disk asynchronously.



Replication

Leader waits for required in-sync replicas to pull the data into their page cache before acknowledging.



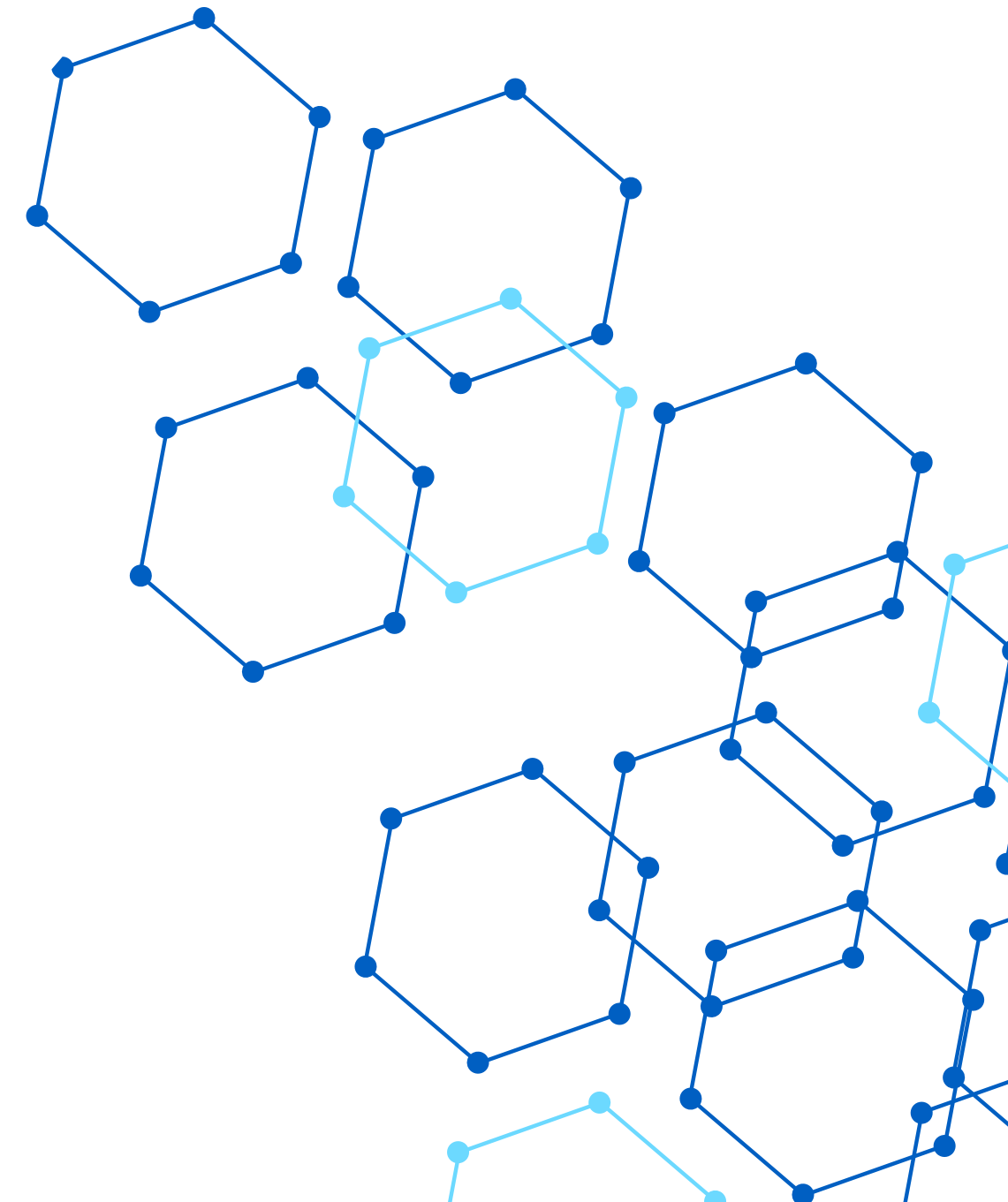
Batching & Compression

Records are grouped and compressed, reducing network and CPU overhead.



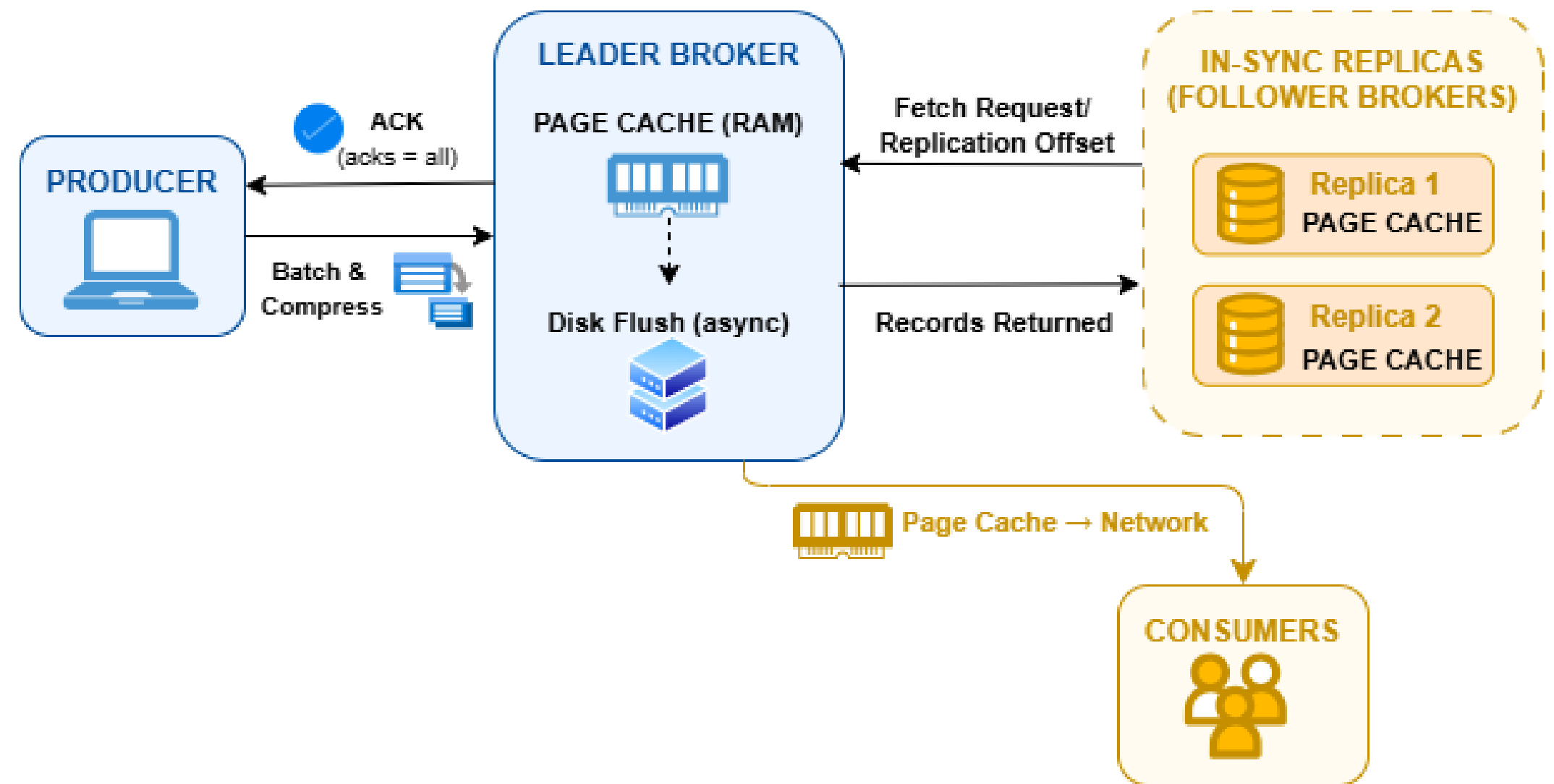
Efficient Transfer

Data can move from page cache to network with fewer application-level copies.



HIGH THROUGHPUT VISUALIZATION

- Producer batches & compresses records before sending.
- Leader writes to page cache; disk flush happens asynchronously.
- Followers fetch and replicate records, then report their progress.
- Leader ACKs after required ISR replication; consumers read efficiently from page cache.



REPLICATION, ISR & HIGH WATERMARK

Leader–Follower Replication

Each partition has one leader and replicated followers across brokers.

In-Sync Replicas (ISR)

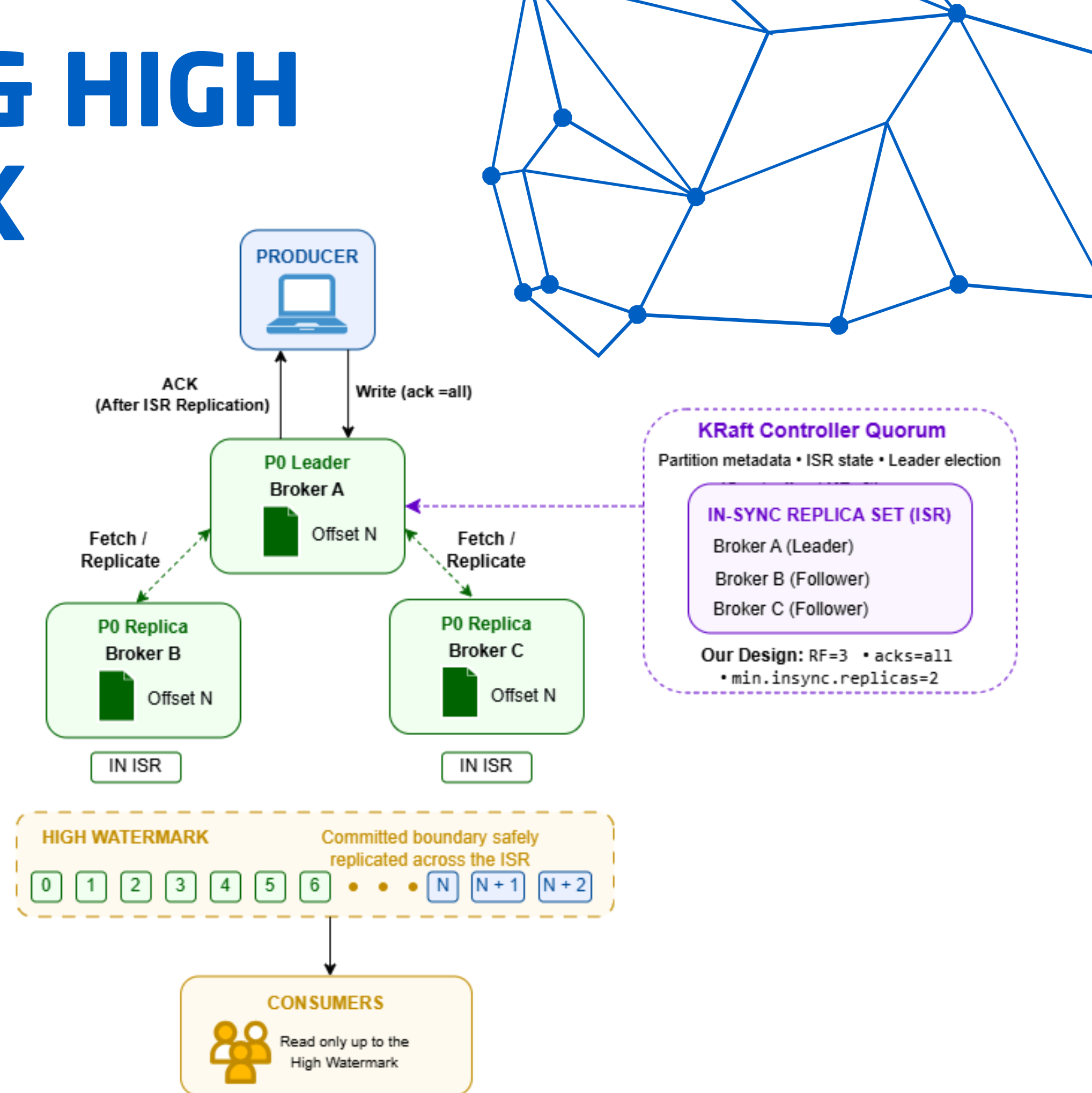
Followers that remain sufficiently caught up with the leader stay in the ISR.

Durable Acknowledgement

With `acks=all`, the leader waits for ISR replication before acknowledging.

High Watermark

Consumers read only committed records up to the High Watermark.



CONSUMER GROUPS, OFFSETS & REBALANCING

Partition Assignment

One partition is assigned to only one consumer per group at a time.

Consumer Parallelism

8 partitions + 4 consumers → about 2 partitions per consumer.
Consumers beyond the partition count remain idle.

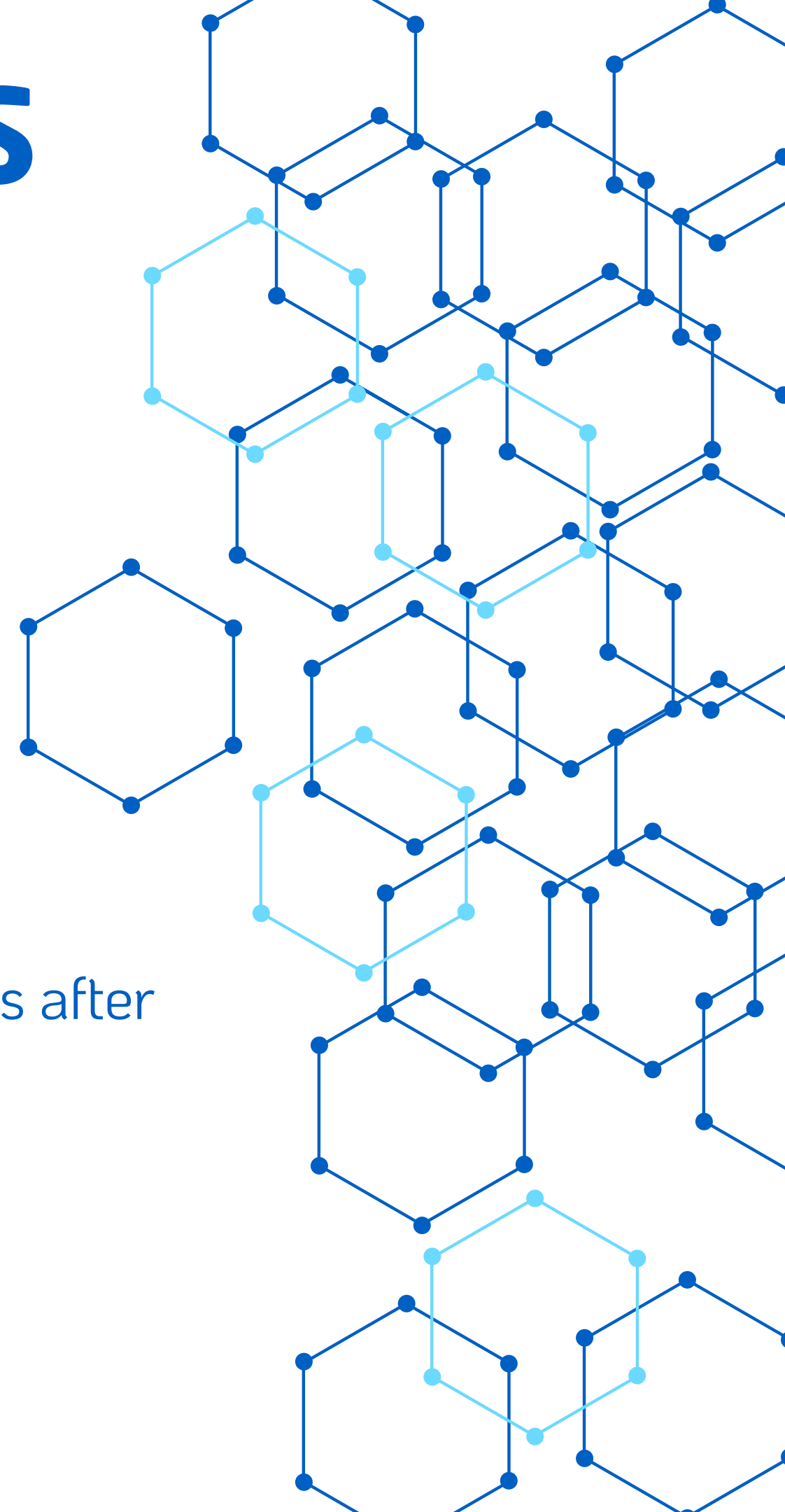
Heartbeats & Rebalancing

The Group Coordinator tracks consumer heartbeats and reassigns partitions after failures.

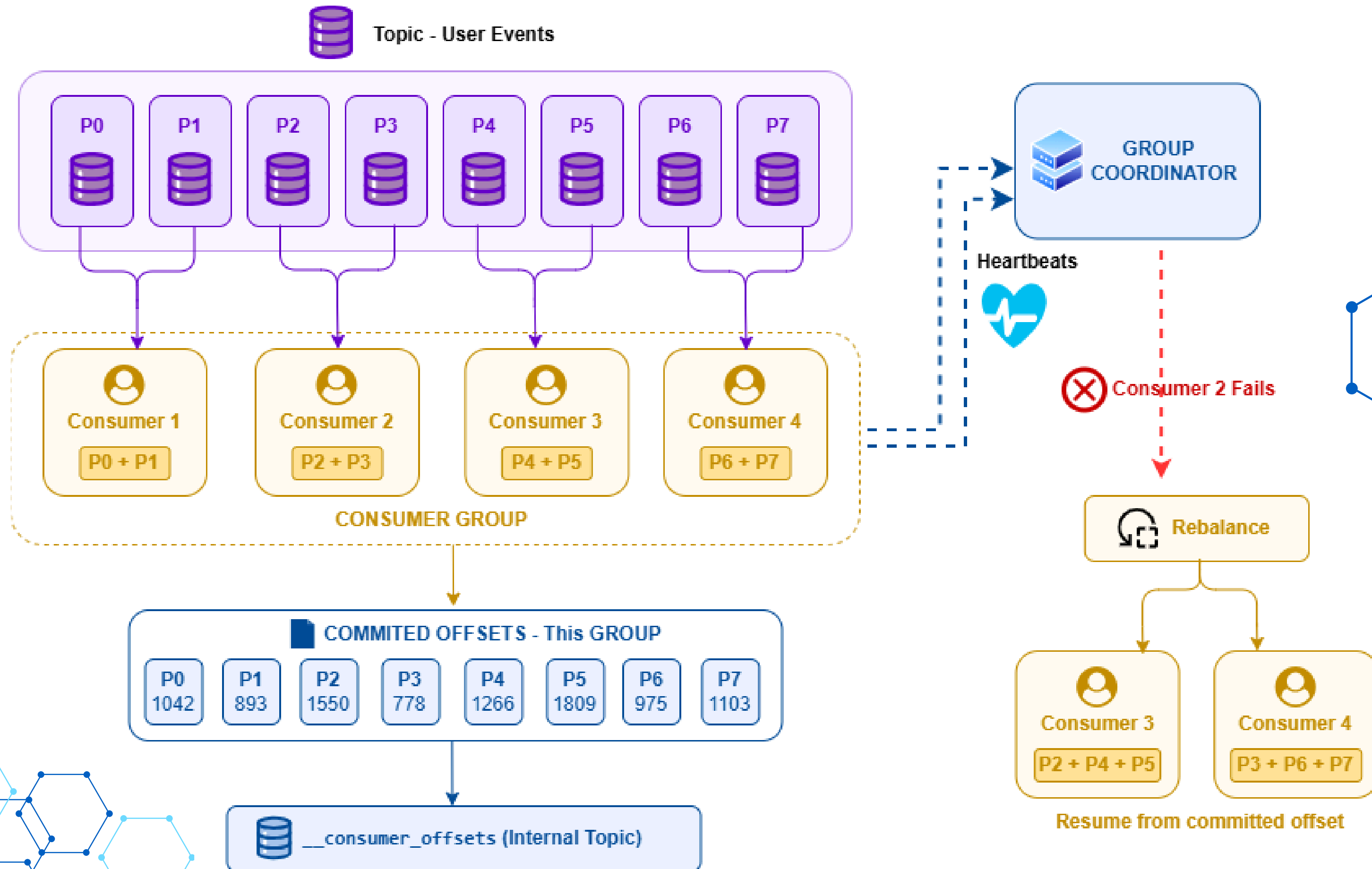
Offset Tracking

Consumers commit progress to Kafka's internal `__consumer_offsets` topic.

Maximum parallelism per consumer group = number of partitions



CONSUMER GROUPS & REBALANCING



BROKER FAILURE & PARTITION TOLERANCE



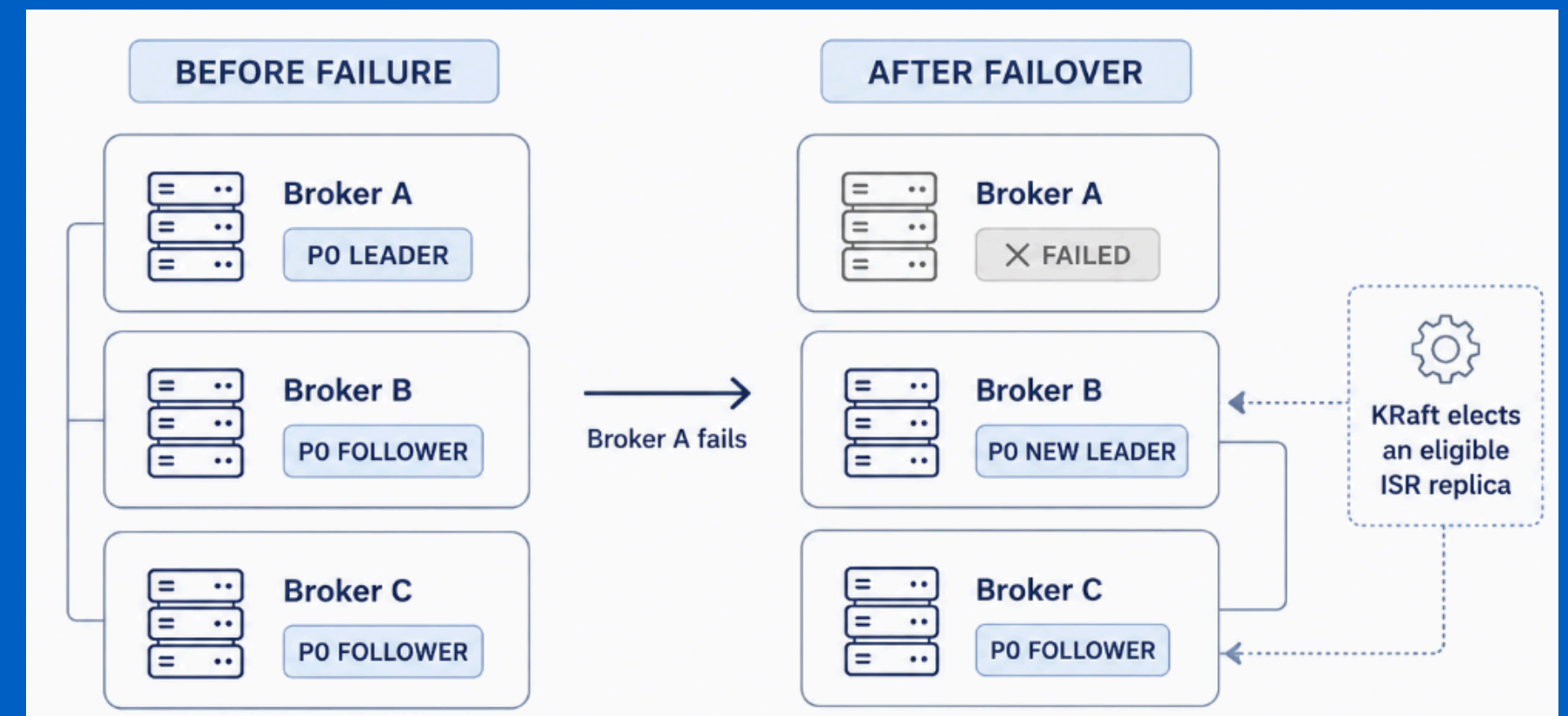
Broker Failure: KRaft detects an unavailable broker. If a partition leader fails, an eligible in-sync replica is promoted.



Network Partition: If the leader cannot maintain at least two in-sync replicas, acks=all writes fail. Kafka favors durability and consistency over write availability.



Recovery: Producers and consumers refresh metadata and reconnect to the new leader. Committed records remain available through replicated copies.



SCALING & BOTTLENECKS

Horizontal Scaling

- **Add Brokers:** More storage, network bandwidth, and partition capacity.
- **Add Partitions:** More producer and consumer parallelism.
- **Add Consumers:** More processing capacity, up to the partition count.

Vertical Scaling

- More CPU, RAM, network bandwidth and NVMe improve a broker's capacity.
- Useful short-term, but eventually limited by a single machine.

Key Bottlenecks

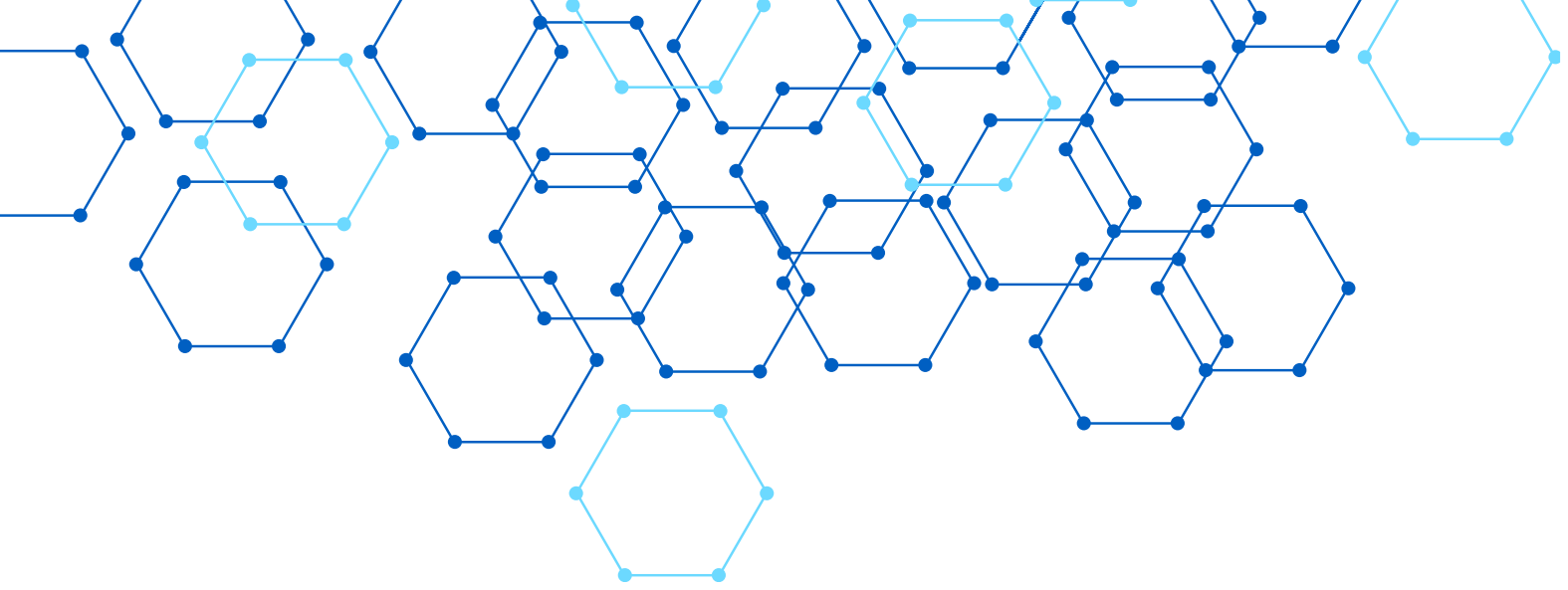
- **Hot Partitions:** Uneven keys concentrate traffic.
- **Consumer Lag:** Consumers process slower than producers.
- **Broker Saturation:** Disk or network reaches capacity.
- **Too Many Partitions:** Higher metadata and coordination overhead.



KEY DECISIONS & TRADE-OFFS

- ✓ **8 Initial Partitions:** Supports the current peak and Year-1 growth; scale before peak throughput exceeds approximately 80 MB/sec.
- ✓ **Key-Based Partitioning:** Preserves ordering for related events; uneven keys can create hot partitions.
- ✓ **RF=3 + acks=all + min.insync.replicas=2:** Protects acknowledged records during a single-broker failure; writes fail if fewer than two replicas remain in sync.
- ✓ **At-Least-Once Delivery:** Consumers may reprocess events after failure; use idempotent consumers, and Kafka transactions for Kafka-to-Kafka exactly-once workflows.
- ✓ **7-Day Local Retention:** Keeps recent data on fast NVMe/SSD for replay; older data can move to tiered/object storage if longer retention is needed
- ✓ **Why Kafka:** Retained partition logs, replay, independent consumer groups and horizontal throughput make Kafka more suitable than a traditional transient work queue.





THANK YOU

