Cockroach Labs

CockroachDB at the Edge of Resilience, Security, and AI

Alicia Lu, Product Manager
Biplav Saraf, Product Manager

PC Computing Era

1980s/90s

Traditional DB

Mobile Computing Era

2010s/20s

NoSQL/Distributed SQL DB

AI Integrated Computing Era

2020s/30s

Vector/Distributed SQL DB

Purpose of Databases has remained **largely the same**,
though the way they **function/operate has changed**

CockroachDB is expanding its deployment model options

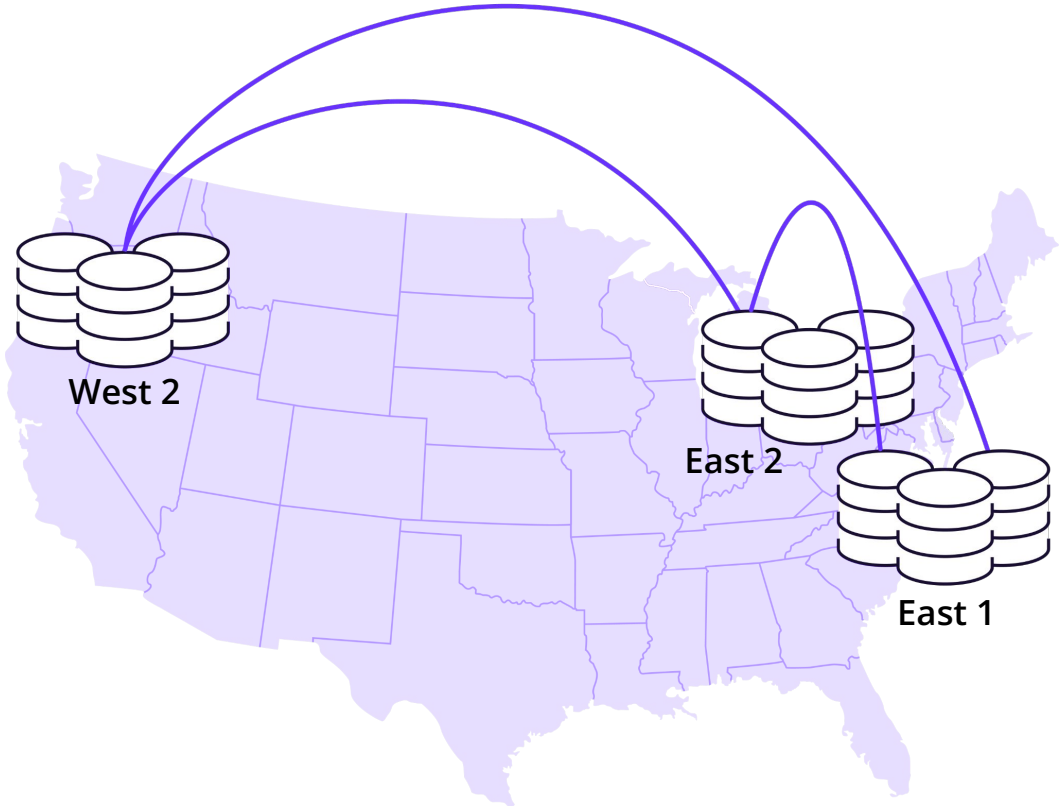
Today, we'll cover:

1) Cross-Cluster Replication in CockroachDB to meet two data center resiliency
2) CockroachDB Security at Scale
3) Architecturing CockroachDB for AI Workloads

Cockroach Labs

Resiliency beyond Raft: Cross-Cluster Replication in CockroachDB

CockroachDB enables resilient, fully consistent clusters



Running critical, distributed workloads with serializable isolation

Achieving zero downtime region resiliency with 3+ data centers

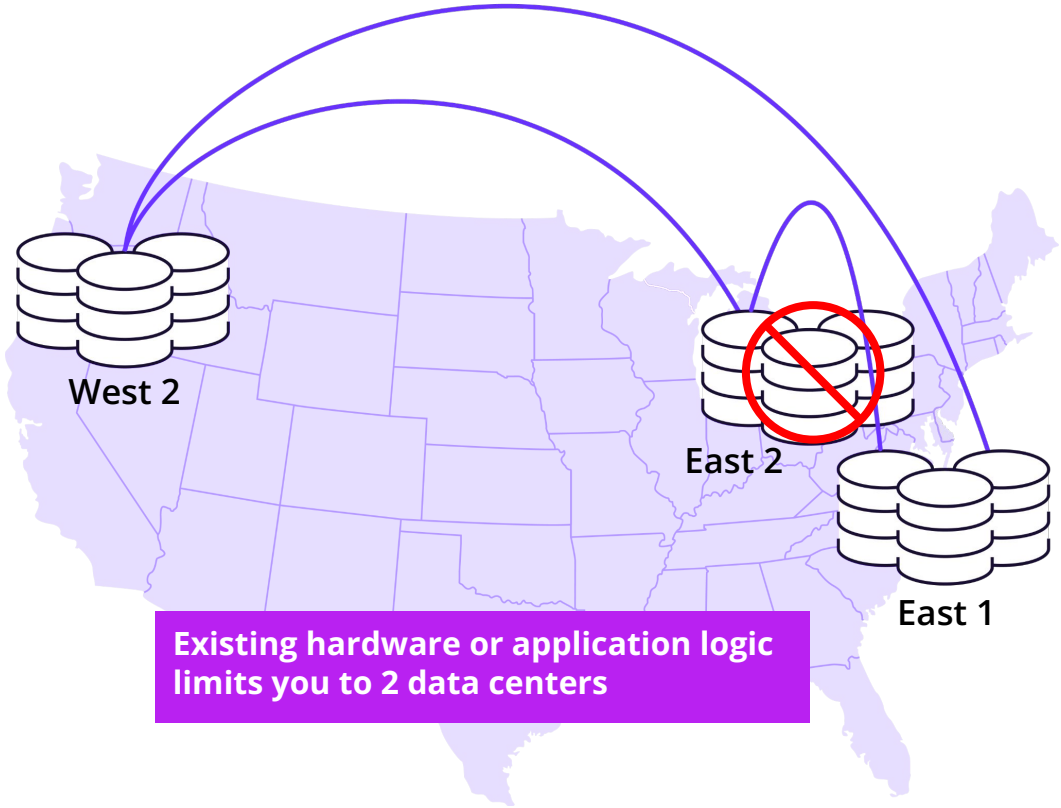
Resiliency powered by native Raft replication and distributed SQL

CockroachDB's native consensus-based Raft replication

- Full data consistency across nodes, zones & regions
- Zero data loss during failures within a cluster

Raft Replication

But what if you have other requirements?



- Running critical, distributed workloads with serializable isolation
- Achieving zero downtime region resiliency with 3+ data centers
- You want to survive a region failure with 2 regions
- You want to achieve low, single-region write latency
- You want to migrate data between clusters
- You want more flexibility with deployment topologies

Introducing new tools to meet custom requirements

CockroachDB's native consensus-based Raft replication

- Full data consistency across nodes, zones & regions
- Zero data loss during failures within a cluster

Raft Replication

+

[GA] Physical Cluster Replication

- Transactionally consistent, full cluster replication
- Ensures data protection and disaster recovery with a passive standby cluster

+

[GA] Logical Data Replication

- Eventually consistent, table and database level replication
- Enables high availability and real-time analysis with active clusters

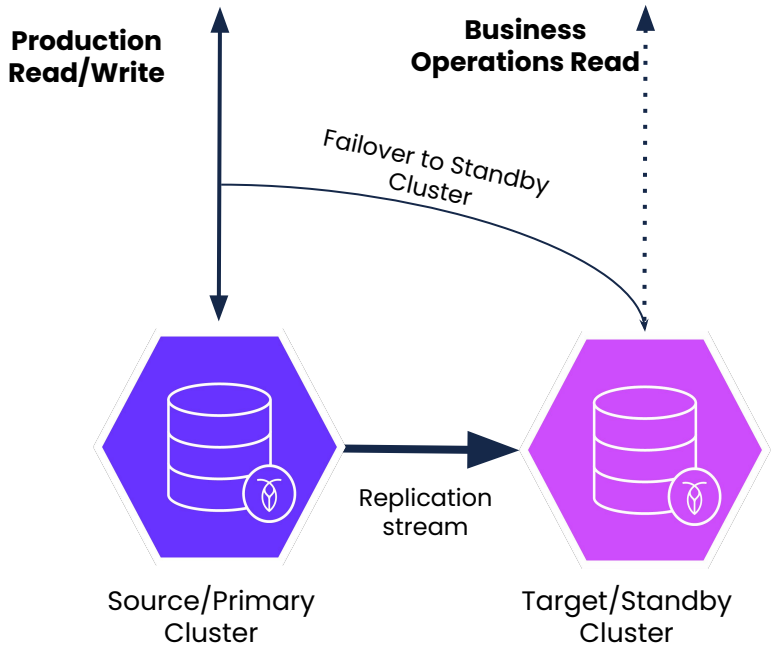
Cross-Cluster Replication Tools

Physical Cluster Replication

Generally Available in 24.1
Limited Access on Cockroach Cloud

What is Physical Cluster Replication (PCR)?

Physical Cluster Replication continuously replicates all data from a *primary* CockroachDB cluster to an independent *standby* CockroachDB cluster



Use cases

Region and Cloud Survivability

Use PCR to survive an regional, cluster, or cloud outage with low RPO and RTO by cutting over to the standby.

Workload Isolation

Use PCR to offload non-critical operations, such as running analytics queries, to the standby cluster (24.3)

Data Migration

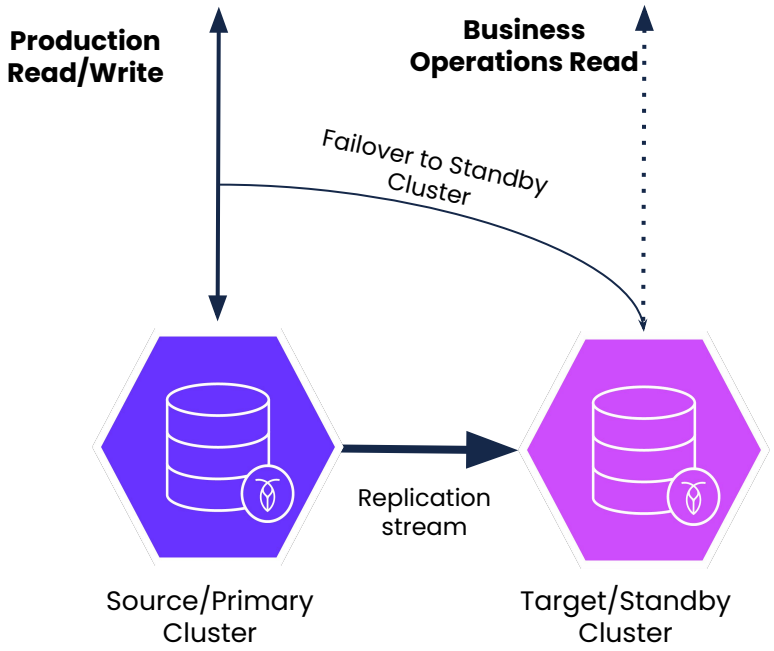
Use PCR to migrate data seamlessly between CockroachDB clusters.

- Cluster upgrade protection
- Populate dev clusters in real time

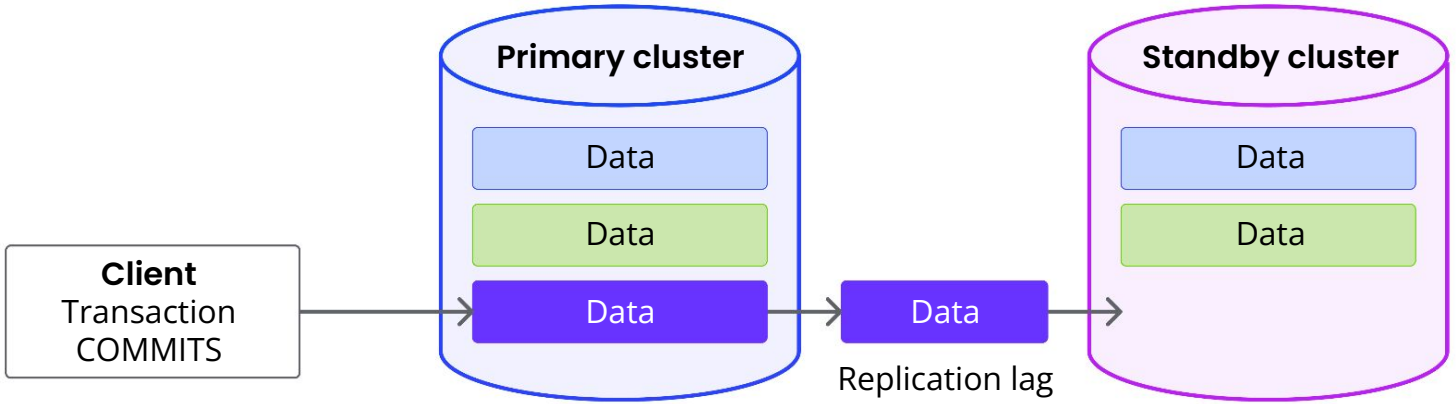
Physical Cluster Replication (PCR) overview

Cluster-level
Replication happens at cluster-level, i.e., it is unaware of database, table, row boundaries, etc.

Transactionally Consistent
You see a transactionally consistent state when you "failover" to the standby, including schema changes, zone configs, users, privileges, etc. No conflicts.

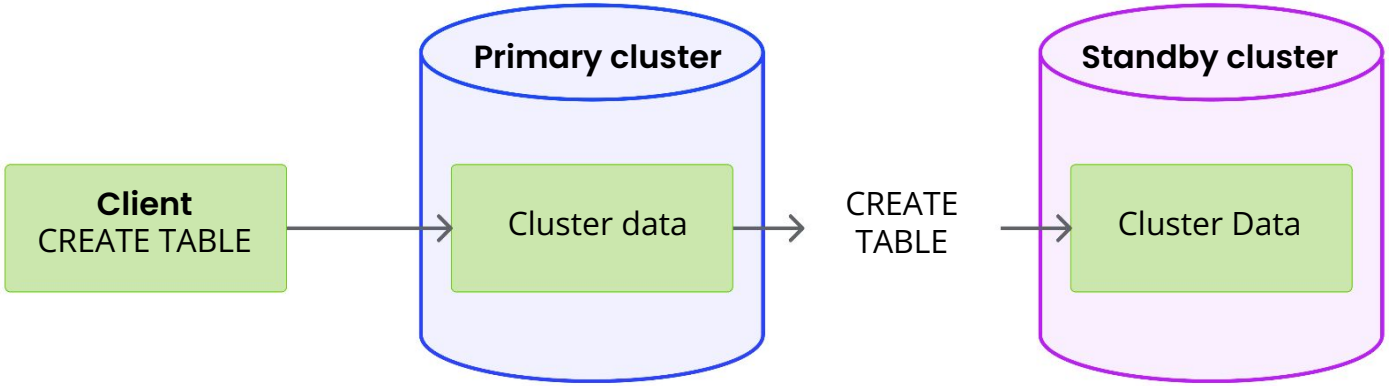


Life of a transaction in PCR

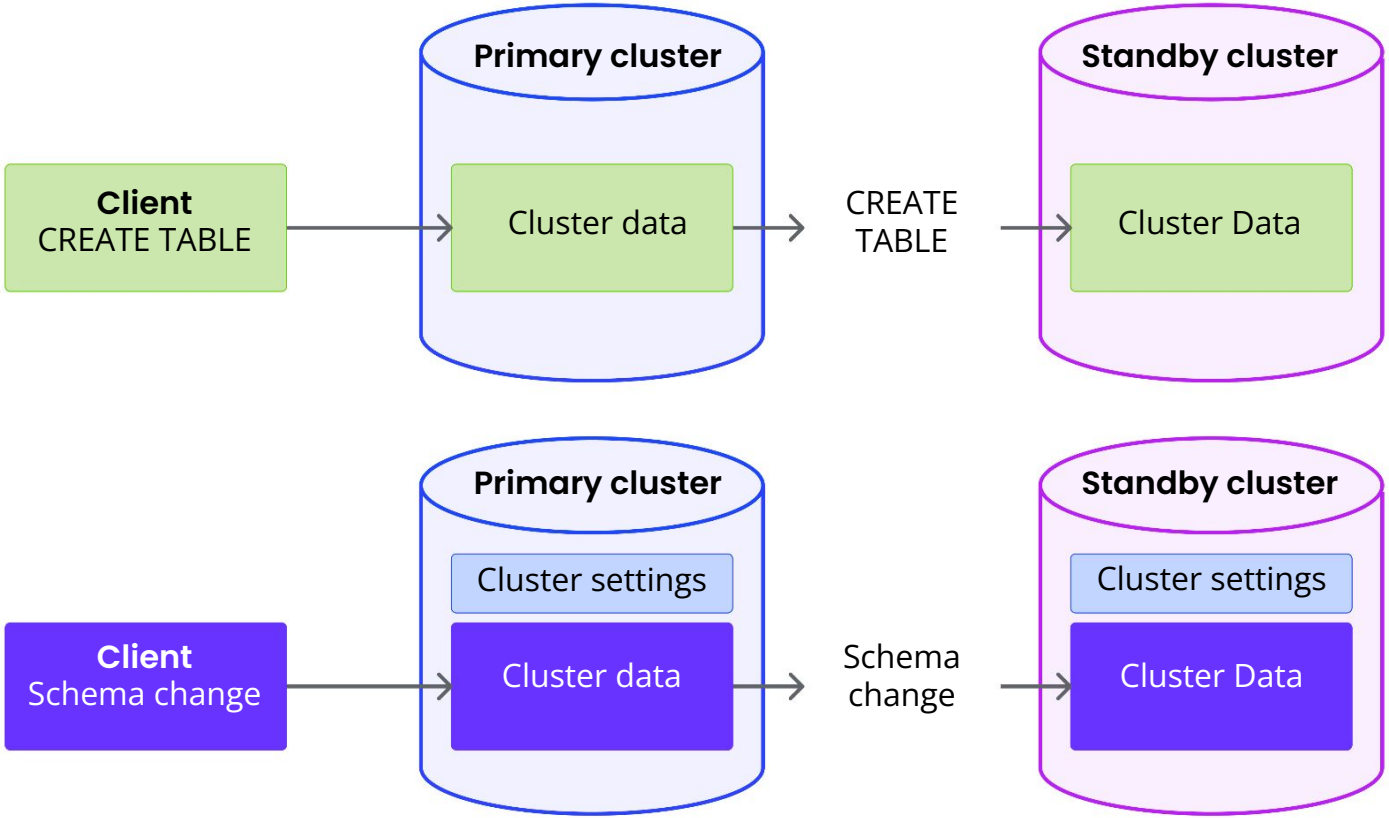


All data is accounted for in PCR during replication.

PCR replicates the entire cluster



PCR replicates the entire cluster



How does PCR handle disasters?

Node Outage, Network Partition

PCR will replan node connections between clusters and continue replication without any manual intervention required.

Cluster, Data-center, Region, Cloud Outage

Users can cutover to the standby cluster with a ~30 second RPO. Users will manually redirect application traffic to the standby cluster.

Human Error, dropping a table

PCR lets you cutover to a **time in the past,** ensuring that the target cluster is a consistent copy of the source cluster at the chosen timestamp.

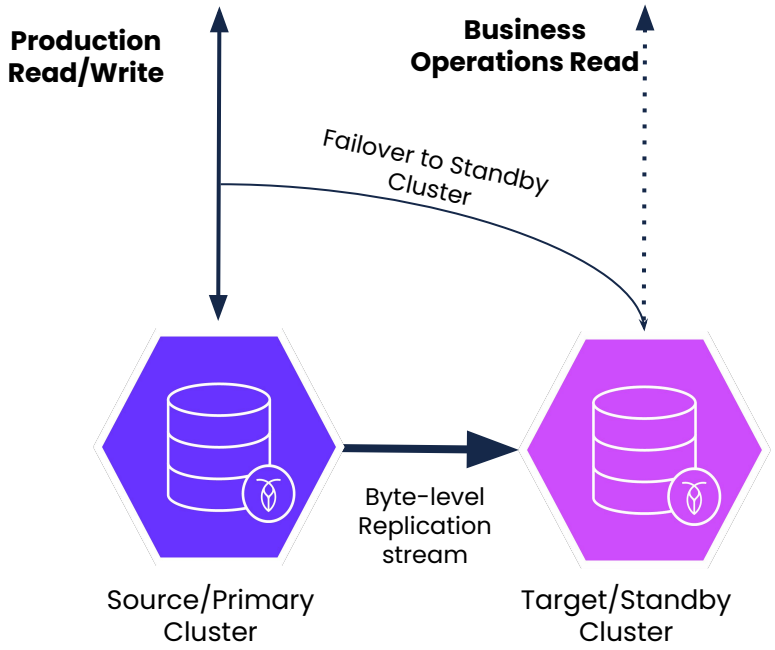
Cyber-attack

PCR lets you cutover to a **time in the past,** ensuring that the target cluster is a consistent copy of the source cluster at the chosen timestamp.

Physical Cluster Replication (PCR) overview

RPO is ~30 seconds for most applications, depending on the workload traffic and corresponding replication lag.

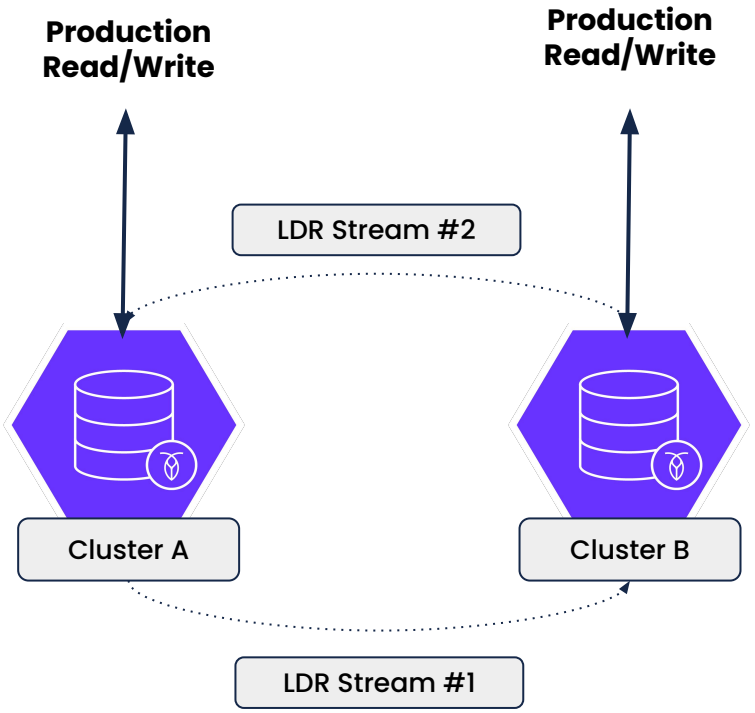
RTO is ~one minute limited by the time it takes to identify the incident, decide and complete the cutover from the primary to standby database, & promote the standby to primary within your load balancer.



Logical Data Replication

GA in 25.2

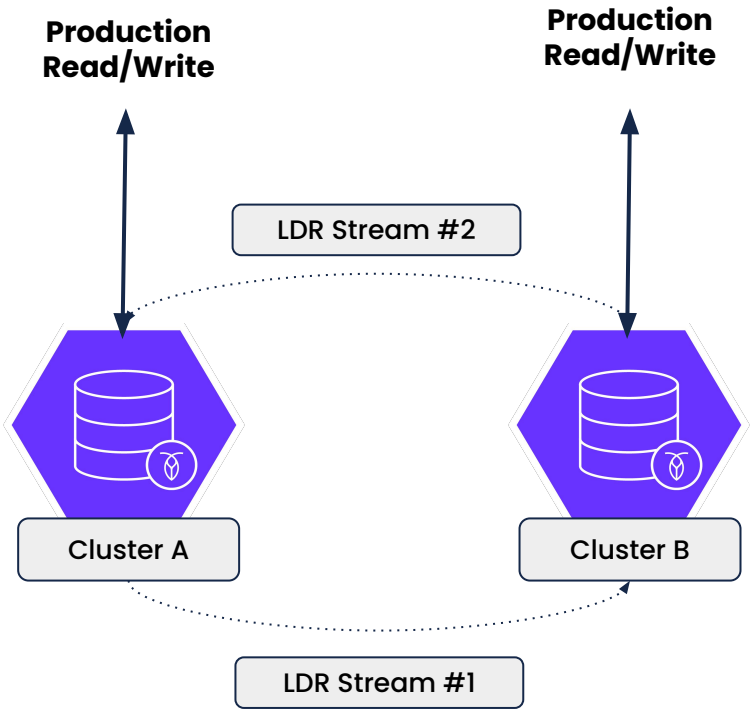
What is Logical Data Replication (LDR)?



LDR provides **ACID consistency on each cluster** and **asynchronous replication** between clusters so that users can achieve **eventual consistency** between 2+ clusters.

Reads and writes can occur concurrently on both clusters allowing customers to survive a region failure while providing low latency read and writes.

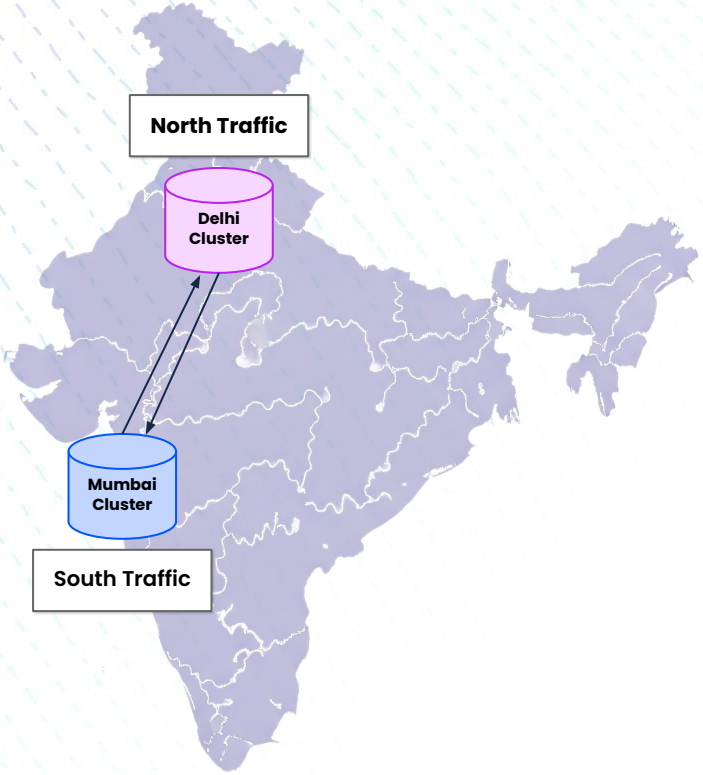
What is Logical Data Replication (LDR)?



LDR replicates **table subsets** of a cluster.

LDR uses **last writer wins** conflict resolution based on the MVCC timestamps of the replicating write.

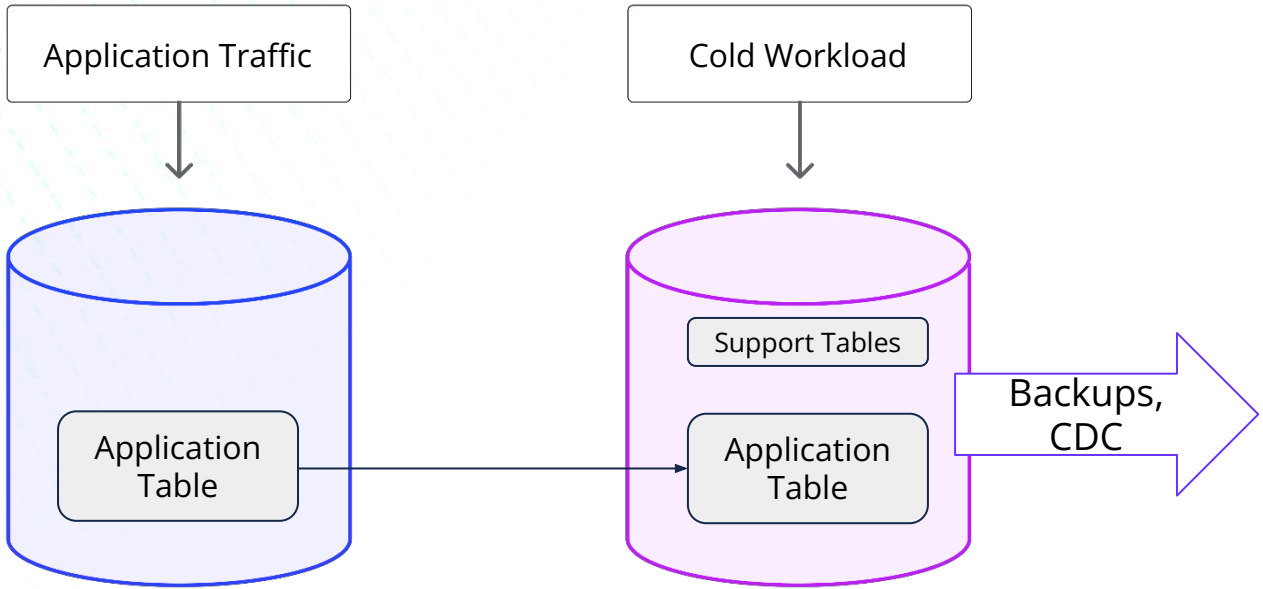
Achieving high availability and low latency with 2 data centers



- Both clusters can receive application reads and writes
- Both clusters have low, single-region write latency with transactionally consistent writes via Raft consensus replication
- In a data center or cluster outage, operators can redirect application traffic to the surviving cluster with little downtime

Workload isolation

LDR enables users to isolate critical application workloads from non-critical application workloads, enabling users to isolate resources and size clusters for certain workloads.



Cross-Cluster Replication in CockroachDB

	Physical Cluster Replication (PCR)	Logical Data Replication (LDR)
What is it?	• **One-way physical replication** • Replicates the entire source cluster	• **Bi-directional logical replication** • Replicate table subsets of a cluster
Why would you use it?	**Surviving region/DC outages with 2 DCs** **Surviving cluster control plane outages** **Data migration between clusters** **Create flexible deployment topologies**	
Operator Experience	**Started by SQL statements, in the DB** **Jobs infrastructure integration** **Admission Control for protecting foreground workloads**	

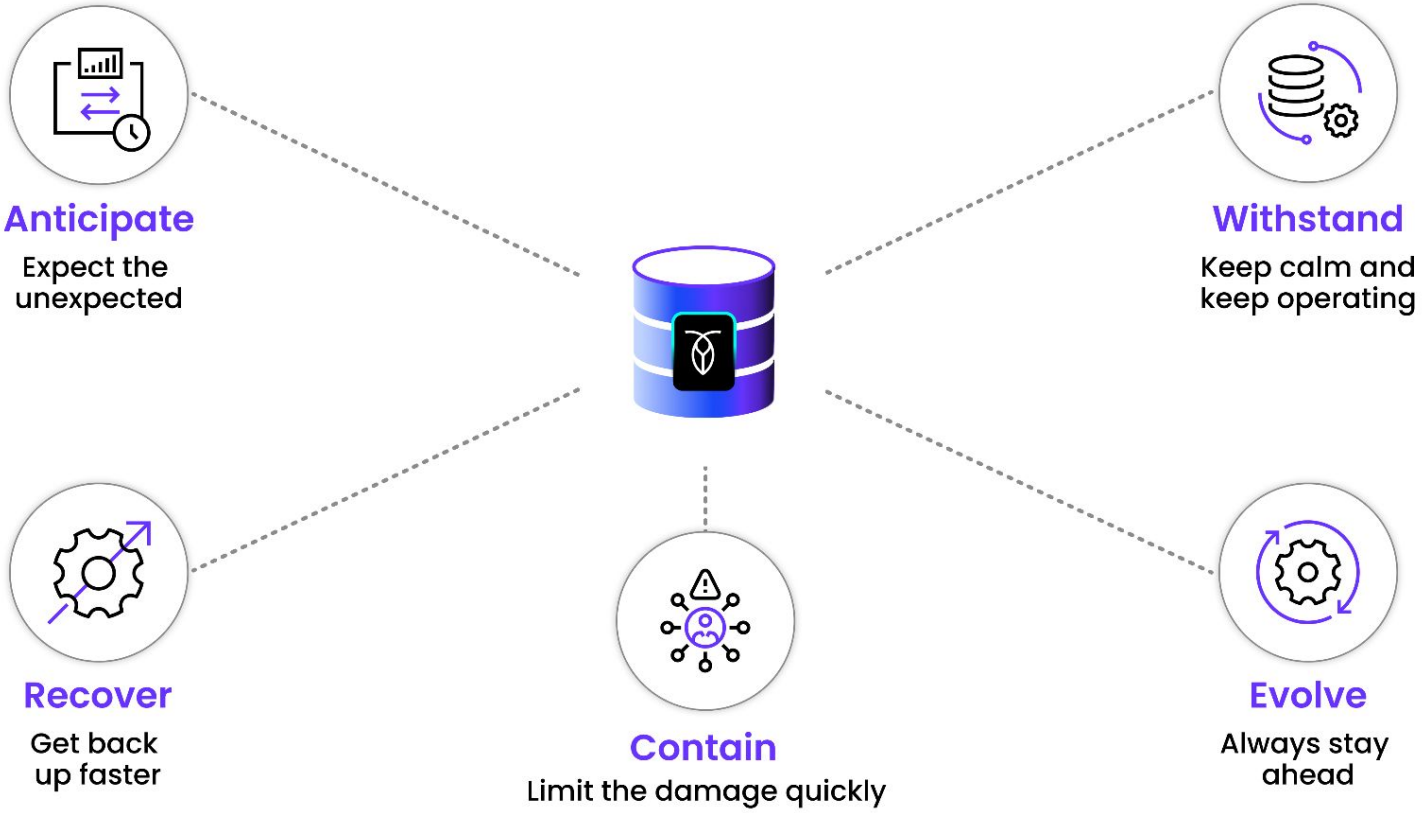
Cockroach Labs

Modern applications demand trust: secure everywhere and at scale

Security starts where the data lives

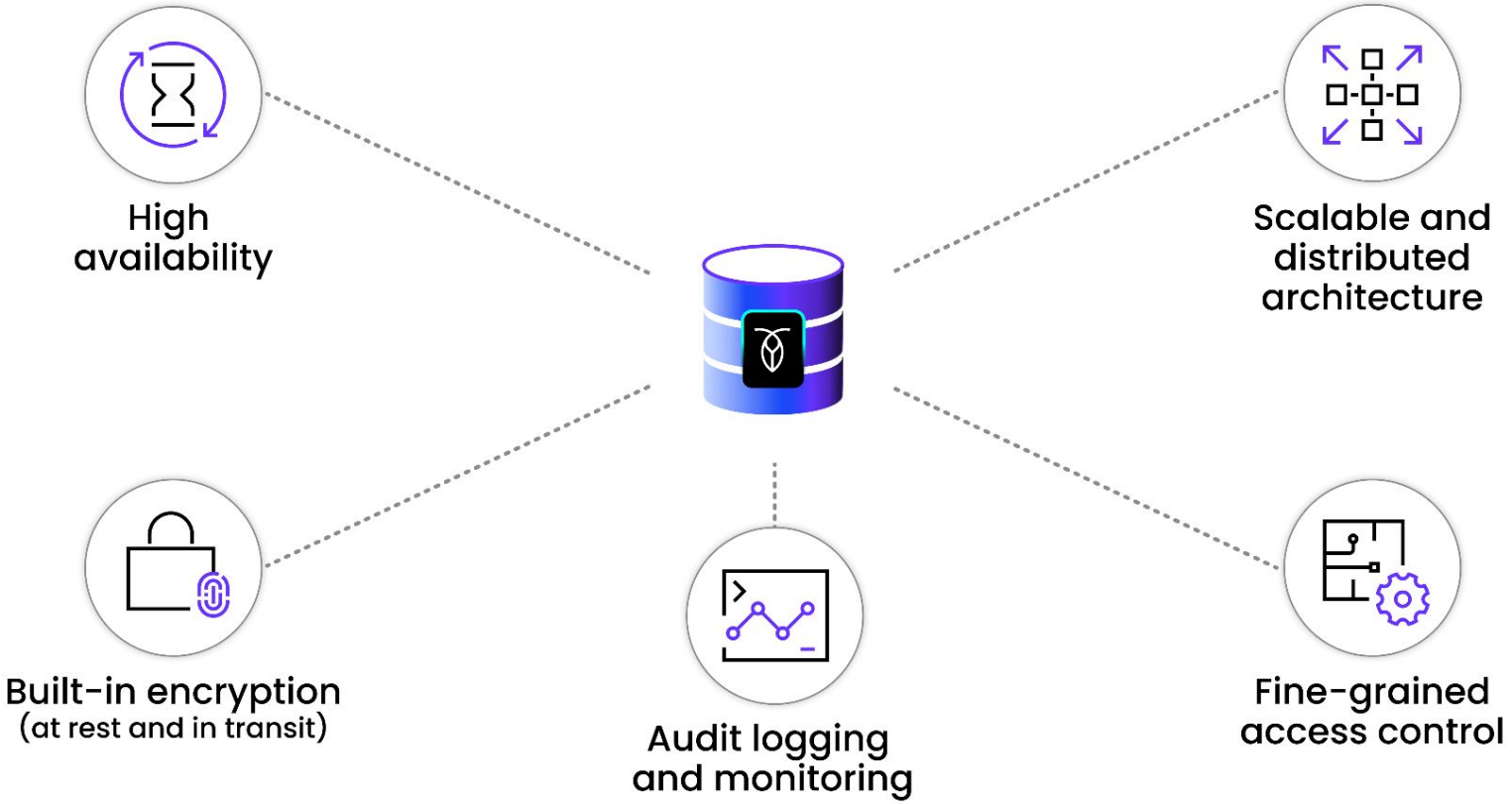
Contain, recover, evolve: resilience against threats

CockroachDB goals



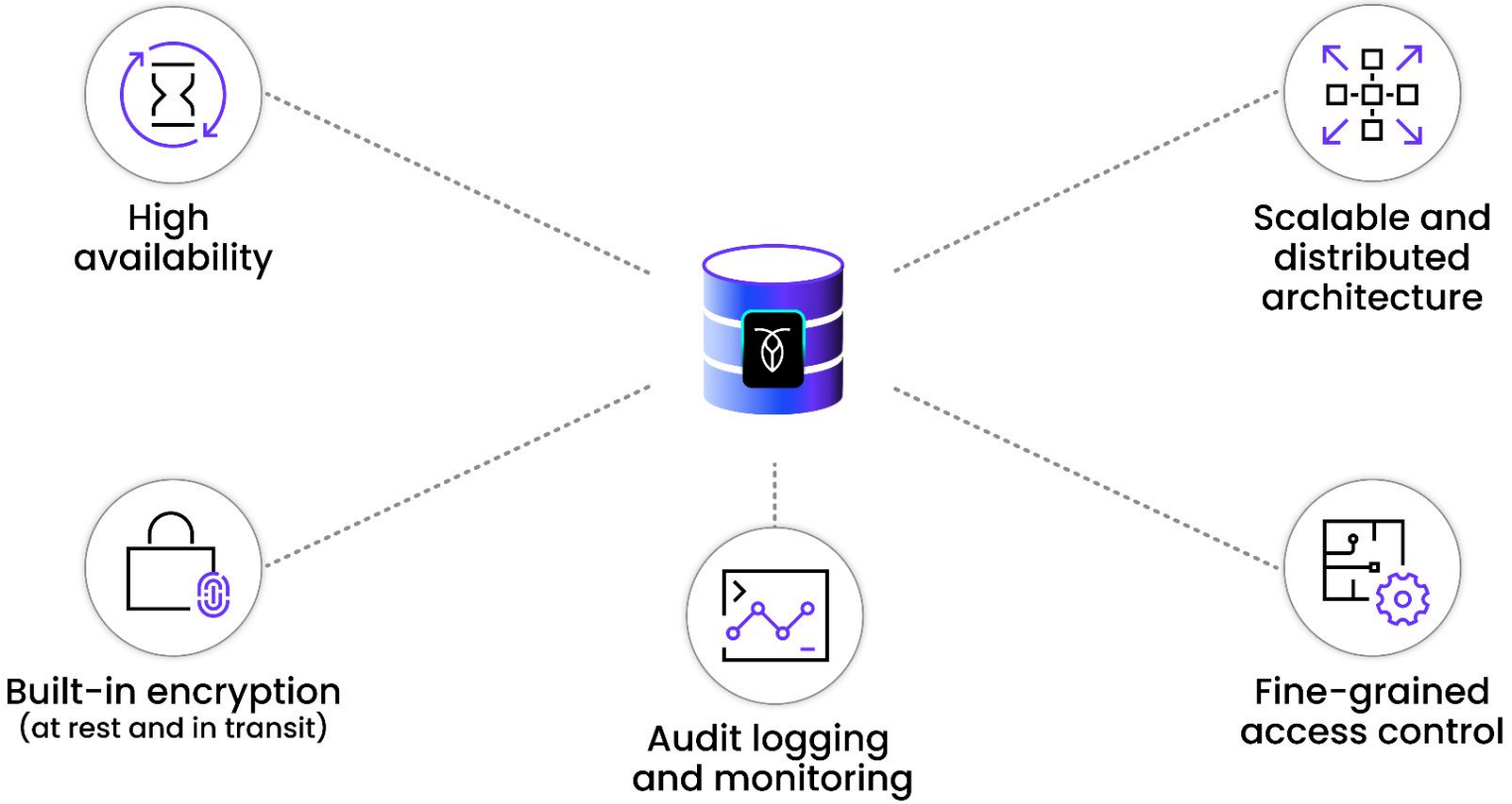
Security and resilience built into the database

CockroachDB features



Security and resilience built into the database

CockroachDB features



Compliance at scale: regional, tenant-aware, secure

Row Level Security provides database native advanced security and compliance controls

SHOW DON'T TELL

CockroachDB

Row-Level Security

Rob Reid

Technical Evangelist at Cockroach Labs

twitter.com/robreid_io

youtube.com/cockroachdb

Cockroach Labs

Architecting for AI: Evolving Database Design for the AI Era

AI memory: the agent's brain

Short-Term Memory (context window)

What it is: *The AI's "working memory" for the current conversation.*

Analogy: *An AI's immediate consciousness.*

Key Traits: *Volatile, limited capacity, and extremely fast.*

AI memory: the agent's brain

Short-Term Memory (context window)

What it is: *The AI's "working memory" for the current conversation.*

Analogy: *An AI's immediate consciousness.*

Key Traits: *Volatile, limited capacity, and extremely fast.*

Long-Term Memory (knowledge base)

What it is: *The permanent store for learning, personalization, and reasoning.*

Episodic: *Remembers specific past interactions. "Last time you asked about billing."*

Semantic: *General knowledge of facts and concepts. "What is a distributed database?"*

Procedural: *Knows how to perform tasks. "How do I book a flight?"*

AI memory: the agent's brain

Short-Term Memory (context window)

What it is: *The AI's "working memory" for the current conversation.*

Analogy: *An AI's immediate consciousness.*

Key Traits: *Volatile, limited capacity, and extremely fast.*

Long-Term Memory (knowledge base)

What it is: *The permanent store for learning, personalization, and reasoning.*

Episodic: *Remembers specific past interactions. "Last time you asked about billing."*

Semantic: *General knowledge of facts and concepts. "What is a distributed database?"*

Procedural: *Knows how to perform tasks. "How do I book a flight?"*

Incidental Memory (governance)

What it is: *The unintentional capture of irrelevant or sensitive information (e.g., personal details shared in a conversation).*

Analogy: *Remembering the color of a passing car you weren't trying to memorize.*

Key Challenge: *This is not a feature, but a governance and security problem. The goal is to prevent storing this data to protect user privacy.*

AI memory: the agent's brain

Short-Term Memory (context window)

What it is: *The AI's "working memory" for the current conversation.*

Analogy: *An AI's immediate consciousness.*

Key Traits: *Volatile, limited capacity, and extremely fast.*

Long-Term Memory (knowledge base)

What it is: *The permanent store for learning, personalization, and reasoning.*

Episodic: *Remembers specific past interactions. "Last time you asked about billing."*

Semantic: *General knowledge of facts and concepts. "What is a distributed database?"*

Procedural: *Knows how to perform tasks. "How do I book a flight?"*

Incidental Memory (governance)

What it is: *The unintentional capture of irrelevant or sensitive information (e.g., personal details shared in a conversation).*

Analogy: *Remembering the color of a passing car you weren't trying to memorize.*

Key Challenge: *This is not a feature, but a governance and security problem. The goal is to prevent storing this data to protect user privacy.*

Contextual Memory (orchestrator)

What it is: *The effective blending of short-term and long-term memory to maintain coherent understanding across interactions.*

Function: *Ensures the AI is consistent and applies past knowledge to current conversations.*

AI memory: the agent's brain

Short-Term Memory (context window)

What it is: *The AI's "working memory" for the current conversation.*

Analogy: *An AI's immediate consciousness.*

Key Traits: *Volatile, limited capacity, and extremely fast.*

Long-Term Memory (knowledge base)

What it is: *The permanent store for learning, personalization, and reasoning.*

Episodic: *Remembers specific past interactions. "Last time you asked about billing."*

Semantic: *General knowledge of facts and concepts. "What is a distributed database?"*

Procedural: *Knows how to perform tasks. "How do I book a flight?"*

Incidental Memory (governance)

What it is: *The unintentional capture of irrelevant or sensitive information (e.g., personal details shared in a conversation).*

Analogy: *Remembering the color of a passing car you weren't trying to memorize.*

Key Challenge: *This is not a feature, but a governance and security problem. The goal is to prevent storing this data to protect user privacy.*

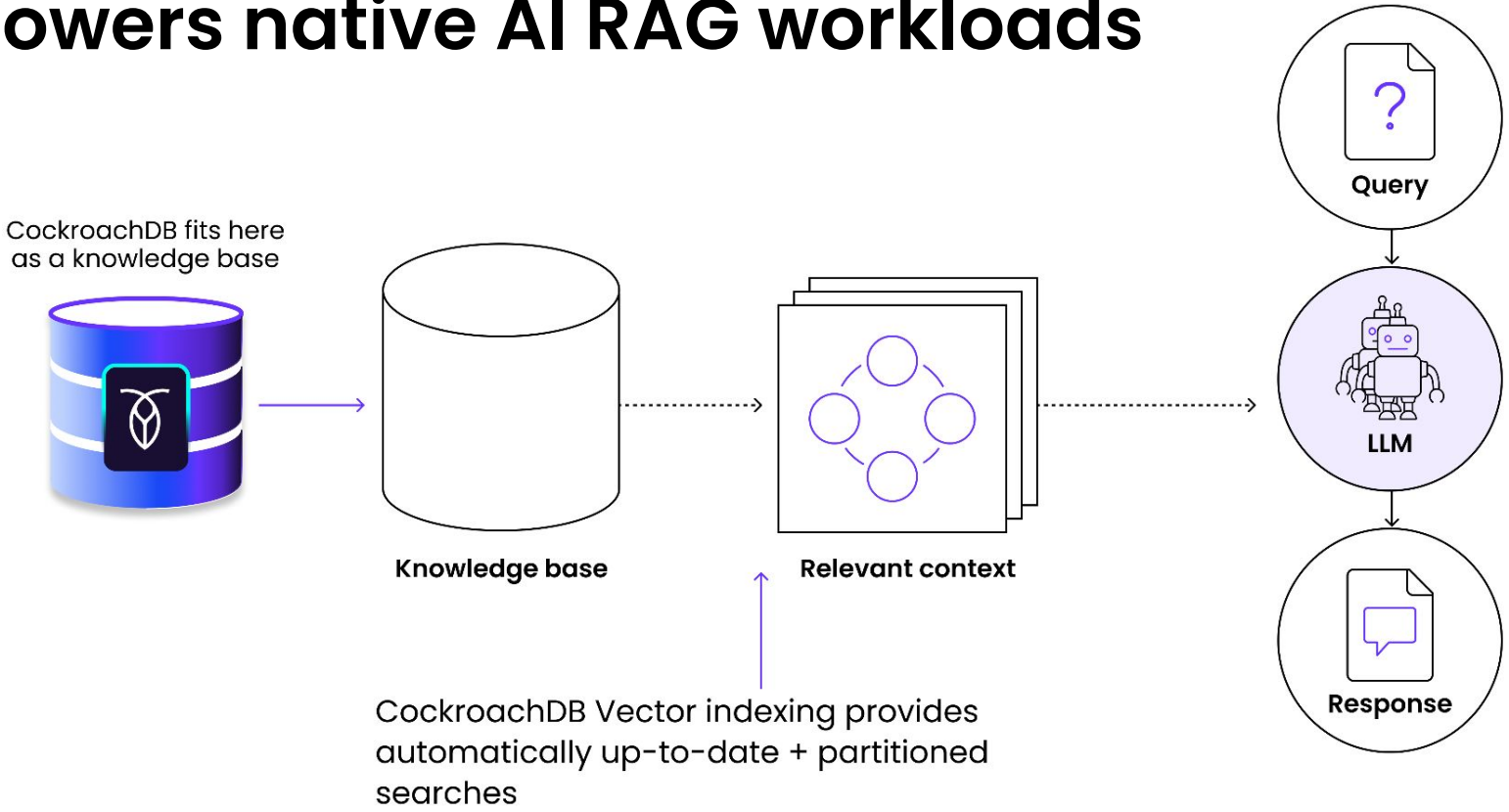
Contextual Memory (orchestrator)

What it is: *The effective blending of short-term and long-term memory to maintain coherent understanding across interactions.*

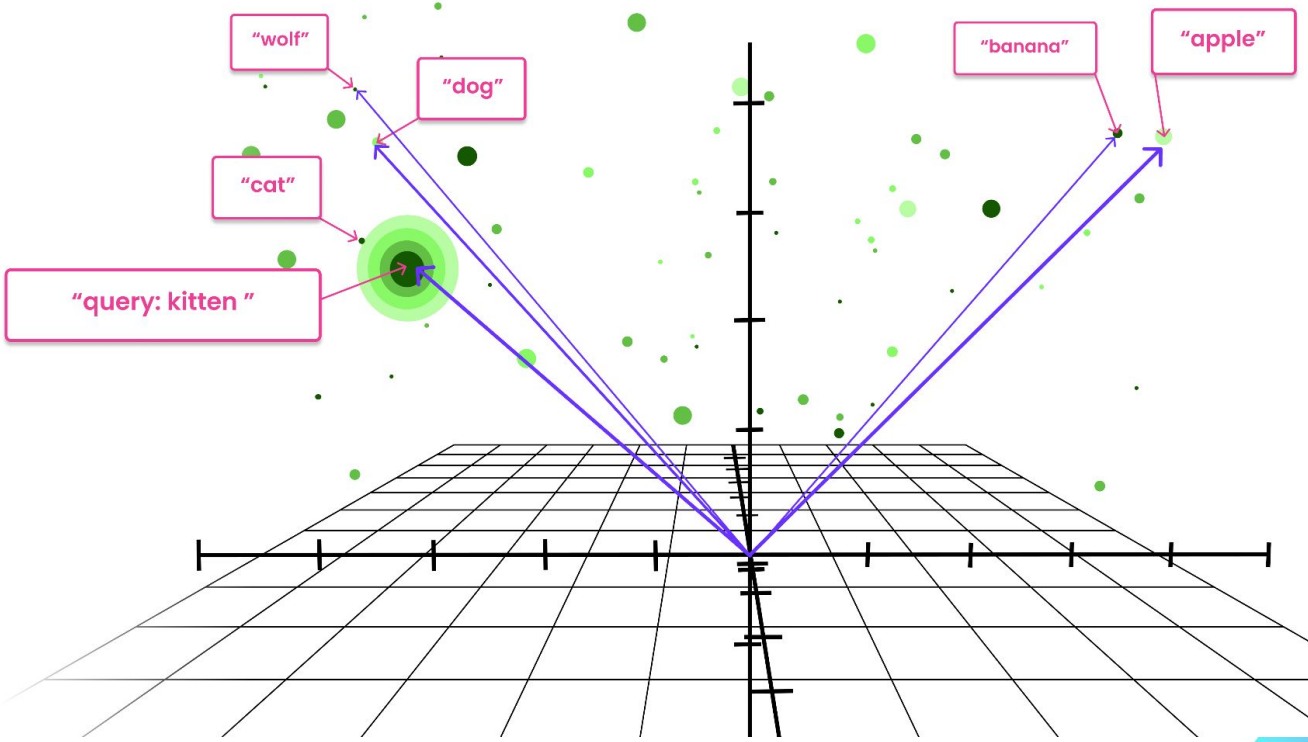
Function: *Ensures the AI is consistent and applies past knowledge to current conversations.*

LLMs are Dumb without it’s brain!

CockroachDB vector data and indexing powers native AI RAG workloads

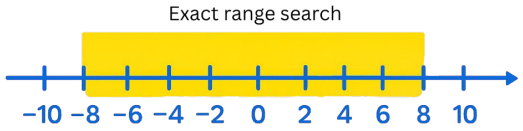


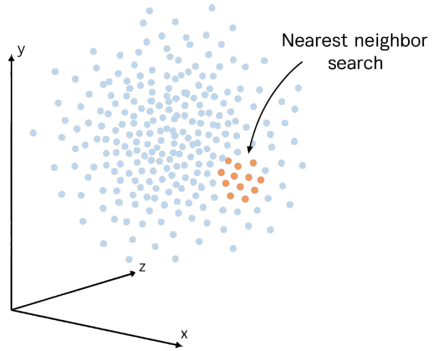
What is a vector?



Why you need a vector index

- Regular indexes work for single-dimensional, ordered data types like strings or numbers
- Embeddings are high-dimensional and don’t have a natural sort order
- We care about **semantic similarity**, not exact match
- Brute-force scanning works at small scale, but falls apart with millions of vectors
- Vector indexes enable fast, approximate nearest neighbor (ANN) search **at scale**





The challenge: distributed vector indexing

- No central coordinator
- No large in-memory structures
- Minimal network hops
- Sharding-compatible layout
- No hot-spots
- Incremental updates

The solution: introducing C-SPANN

- Vector indexing algorithm designed specifically for CockroachDB
- Based on **SPANN and SPFresh** papers from Microsoft
- **Adapted** for a distributed SQL architecture
- Data quantization done with **RaBitQ** to speed up performance

SPANN: Highly-efficient Billion-scale Approximate Nearest Neighbor Search

Qi Chen[1,*] Bing Zhao[1,2,†] Haidong Wang[1] Mingqin Li[1] Chuanjie Liu[1,3,†]
Zengzhong Li[1] Mao Yang[1] Jingdong Wang[1,4,=,†]
[1]Microsoft [2]Peking University [3]Tencent [4]Baidu
[1]{cheqi, haidwa, mingqili, jaoli, maoyang}@microsoft.com
[2]its.bingzhao@pku.edu.cn [3]liu.chuanjie@outlook.com
[4]wangjingdong@outlook.com

Abstract

The in-memory algorithms for approximate nearest neighbor search (ANNS) have achieved great success for fast high-recall search, but are extremely expensive when handling very large scale database. Thus, there is an increasing request for the hybrid ANNS solutions with small memory and inexpensive solid-state drive (SSD). In this paper, we present a simple but efficient memory-disk hybrid indexing and search system, named SPANN, that follows the inverted index methodology. It stores the centroid points of the posting lists in the memory and the large posting lists in the disk. We guarantee both disk-access efficiency (low latency) and high recall by effectively reducing the disk-access number and retrieving high-quality posting lists. In the index-building stage, we adopt a hierarchical balanced clustering algorithm to balance the length of posting lists and augment the posting list by adding the points in the closure of the corresponding clusters. In the search stage, we use a query-aware scheme to dynamically prune the access of unnecessary posting lists. Experiment results demonstrate that SPANN is 2× faster than the state-of-the-art ANNS solution DiskANN to reach the same recall quality 90% with same memory cost in three billion-scale datasets. It can reach 90% recall@1 and recall@10 in just around one millisecond with only 32GB memory cost. Code is available at: https://github.com/microsoft/SPTAG.

1 Introduction

Vector nearest neighbor search has played an important role in information retrieval area, such as multimedia search and web search, which provides relevant results by searching vectors with minimum distance to the query vector. Exact solutions for K-nearest neighbor search [49, 40] are not applicable in big data scenario due to substantial computation cost and high query latency. Therefore, researchers have proposed many kinds of approximate nearest neighbor search (ANNS) algorithms in the literature [11, 18, 38, 10, 14, 31, 34, 13, 29, 21, 16, 26, 42, 43, 33, 44, 37, 32, 19, 27, 9, 12, 39, 50, 20, 36]. However, most of the algorithms mainly focus on how to do low latency and high recall search all in memory with offline pre-built indexes. When targeting to the super large scale vector search scenarios, such as web search, the memory cost will become extremely expensive. There is an increasing request for the hybrid ANNS solutions that use small memory and inexpensive disk to serve the large scale datasets.

There are only a few approaches working on the hybrid ANNS solutions, including DiskANN [39] and HM-ANN [36]. Both of them are graph based solutions. DiskANN uses Product Quantization

*Corresponding author.
†Work done while at Microsoft.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

SPFresh: Incremental In-Place Update for Billion-Scale Vector Search

Yuming Xu[1,2] Hengyu Liang[1,2] Jin Li[1,2,*] Shuotao Xu[2] Qi Chen[2] Qianxi Zhang[2]
Cheng Li[2] Ziyue Yang[2] Fan Yang[2] Yuqing Yang[2] Peng Cheng[2] Mao Yang[2]
[1]University of Science and Technology of China [2]Microsoft Research Asia [*]Harvard University

Abstract

Approximate Nearest Neighbor Search (ANNS) on high-dimensional vector data is now widely used in various applications, including information retrieval, question answering, and recommendation. As the amount of vector data grows continuously, it becomes important to support updates to vector index, the enabling technique that allows for efficient and accurate ANNS on vectors.

Because of the curse of high dimensionality, it is often costly to identify the right neighbors of a new vector, a necessary process for index update. To amortize update costs, existing systems maintain a secondary index to accumulate updates, which are merged with the main index by globally rebuilding the entire index periodically. However, this approach has high fluctuations of search latency and accuracy, not to mention that it requires substantial resources and is extremely time-consuming to rebuild.

We introduce SPFresh, a system that supports in-place vector updates. At the heart of SPFresh is LIRE, a lightweight incremental rebalancing protocol to split vector partitions and reassign vectors in the nearby partitions to adapt to data distribution shifts. LIRE achieves low-overhead vector updates by only reassigning vectors at the boundary between partitions, where in a high-quality vector index the amount of such vectors is deemed small. With LIRE, SPFresh provides superior query latency and accuracy to solutions based on global rebuild, with only 1% of DRAM and less than 10% cores needed at the peak compared to the state-of-the-art, in a billion scale disk-based vector index with a 1% of daily vector update rate.

CCS Concepts: • **Information systems** → **Information storage systems**; **Information retrieval**.

Keywords: Vector Search, Incremental Update, Billion-scale

ACM Reference Format:
Yuming Xu, Hengyu Liang, Jin Li, Shuotao Xu, Qi Chen, Qianxi Zhang, Cheng Li, Ziyue Yang, Fan Yang, Yuqing Yang, Peng Cheng, Mao Yang. 2023. SPFresh: Incremental In-Place Update for Billion-Scale Vector Search. In *ACM SIGOPS 29th Symposium on Operating Systems Principles (SOSP '23), October 23–26, 2023, Koblenz, Germany.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3600006.3613166

1 Introduction

Today deep learning models can embed almost all types of data, including speech, vision, and text information, into multi-dimensional vectors with tens or even hundreds of dimensions. Such vectors are critical for complex semantic understanding tasks [41, 69]. To enable effective vector analysis, vector nearest neighbor search (NNS) systems have become critical system components for an increasing number of online services like search [35] and recommendation [57].

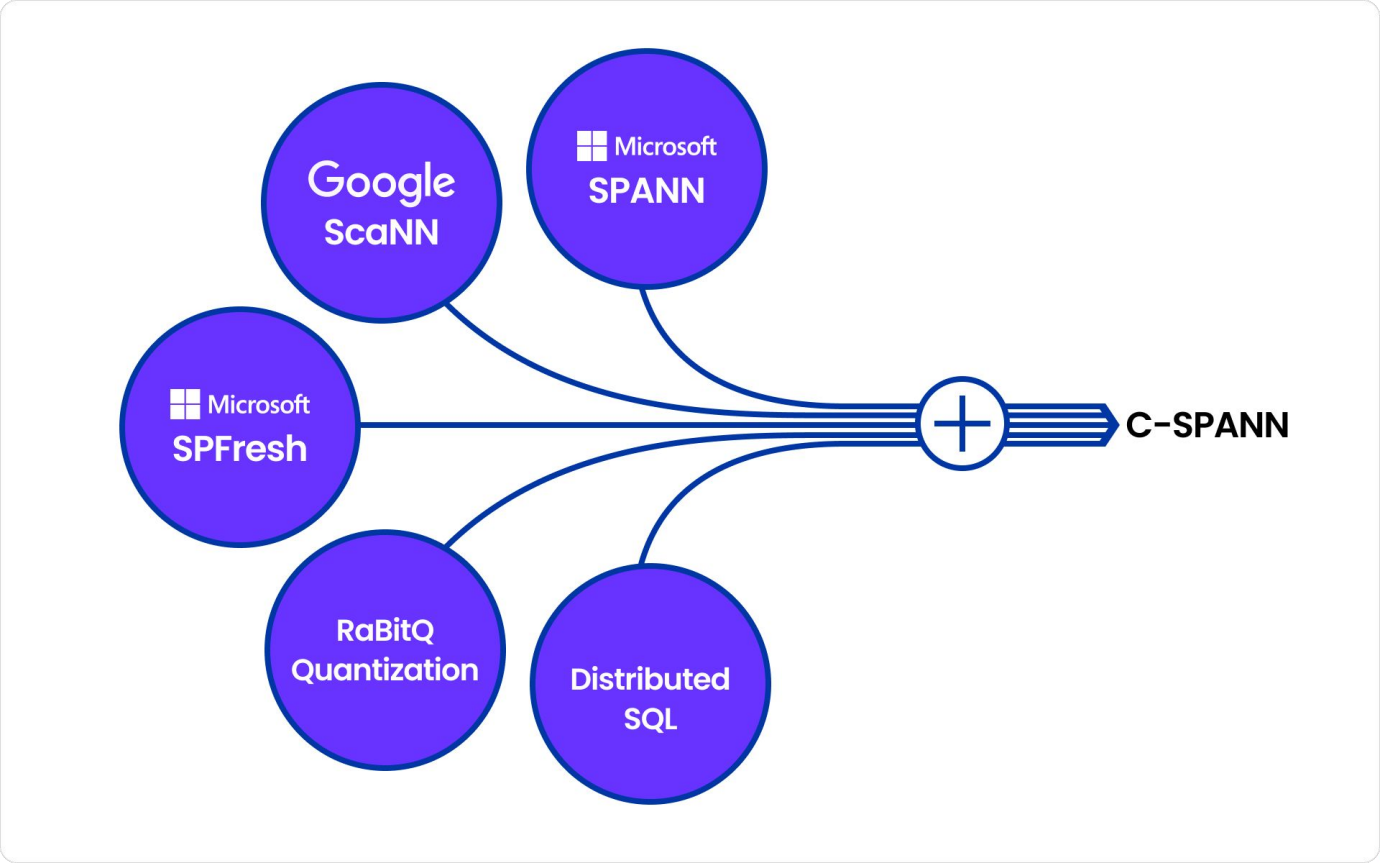
To satisfy the strict query latency requirement for these online services, vector search systems often resort to approximate *nearest neighbor search* (ANNS) [17, 22, 34, 51, 56, 59, 65, 69], to locate as many correct results as possible (i.e., query accuracy). At the heart of a large-scale NNS system is a vector index, a key data structure that organizes high-dimensional vectors efficiently for high-accuracy low-latency vector searches [3, 9, 14, 38, 58, 67].

Like traditional indices, a high-quality vector index organizes a quick "navigation map" of vectors based on the vector proximity in a high dimensional space. The proximity measurements are often implemented with "shortcuts", which only exist between a pair of vectors with a short distance. A search query traverses the datasets based on "shortcuts" the result set. The quality of the index for efficient traversal is highly dependent on the quality of shortcuts, where insufficient shortcuts miss relevant vectors, and extraneous shortcuts incur excessive traversal and storage costs. For high-dimensional data, vector indices require careful construction to produce a sufficient amount of high-quality "shortcuts", often as vector partitions [14, 67] or graphs of vector data [16, 61].

To add fuel to fire, there is a strong desire to support *fresh* update of vector indices because current systems generate a vast amount of vector data continuously in various settings. For example, 500+ hours of content are uploaded to

[*]Work done during his final-year study at USTC and his internship at MSRA.

The solution: introducing C-SPANN



Why Cockroach DB + vector search?

Scalability

- **Horizontal Scale**: Take advantage of Cockroach DB's horizontal scalability to store large amounts of Vector data.
- **Distributed Architecture**: Automatically scales out horizontally as Vector data grows, maintaining performance and reliability without complex manual sharding.

Why CRDB + vector search?

Scalability

- **Horizontal Scale**: Take advantage of Cockroach DB's horizontal scalability to store large amounts of Vector data.

- **Distributed Architecture**: Automatically scales out horizontally as Vector data grows, maintaining performance and reliability without complex manual sharding.

Resilience

- **High Availability**: Industry-standard 99.999% SLA on CockroachDB Cloud Dedicated ensures continuous operation and minimal downtime.
- **Self-healing infrastructure**: Vector data automatically rebalances when a node is lost and heals when you add back a node

Why CRDB + vector search?

Scalability

- **Horizontal Scale**: Take advantage of Cockroach DB's horizontal scalability to store large amounts of Vector data.

- **Distributed Architecture**: Automatically scales out horizontally as Vector data grows, maintaining performance and reliability without complex manual sharding.

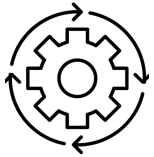
Resilience

- **High Availability**: Industry-standard 99.999% SLA on CockroachDB Cloud Dedicated ensures continuous operation and minimal downtime.
- **Self-healing infrastructure**: Vector data automatically rebalances when a node is lost and heals when you add back a node

Enterprise Features

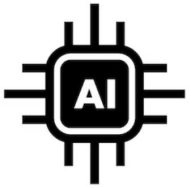
- **Integrated Vector and Operational Data**: Combines the strengths of a Vector database and an operational database into a single, resilient and scalable solution, simplifying architecture and reducing costs.
- **Future-Proof Investment**: Supports modernizing legacy applications and meets evolving demands of AI and machine learning workloads.

Redefining the limits of operational data using CockroachDB



Modernization

Cloud



AI

CockroachDB Distributed SQL

Delivers state of the art resilience	Scales effortlessly to meet demand	Expands easily everywhere	Simplifies operations, optimize TCO

Can be deployed anywhere and available as self-hosted software or SaaS managed service

Thank you!

Your feedback fuels our roadmap

RoachFest25