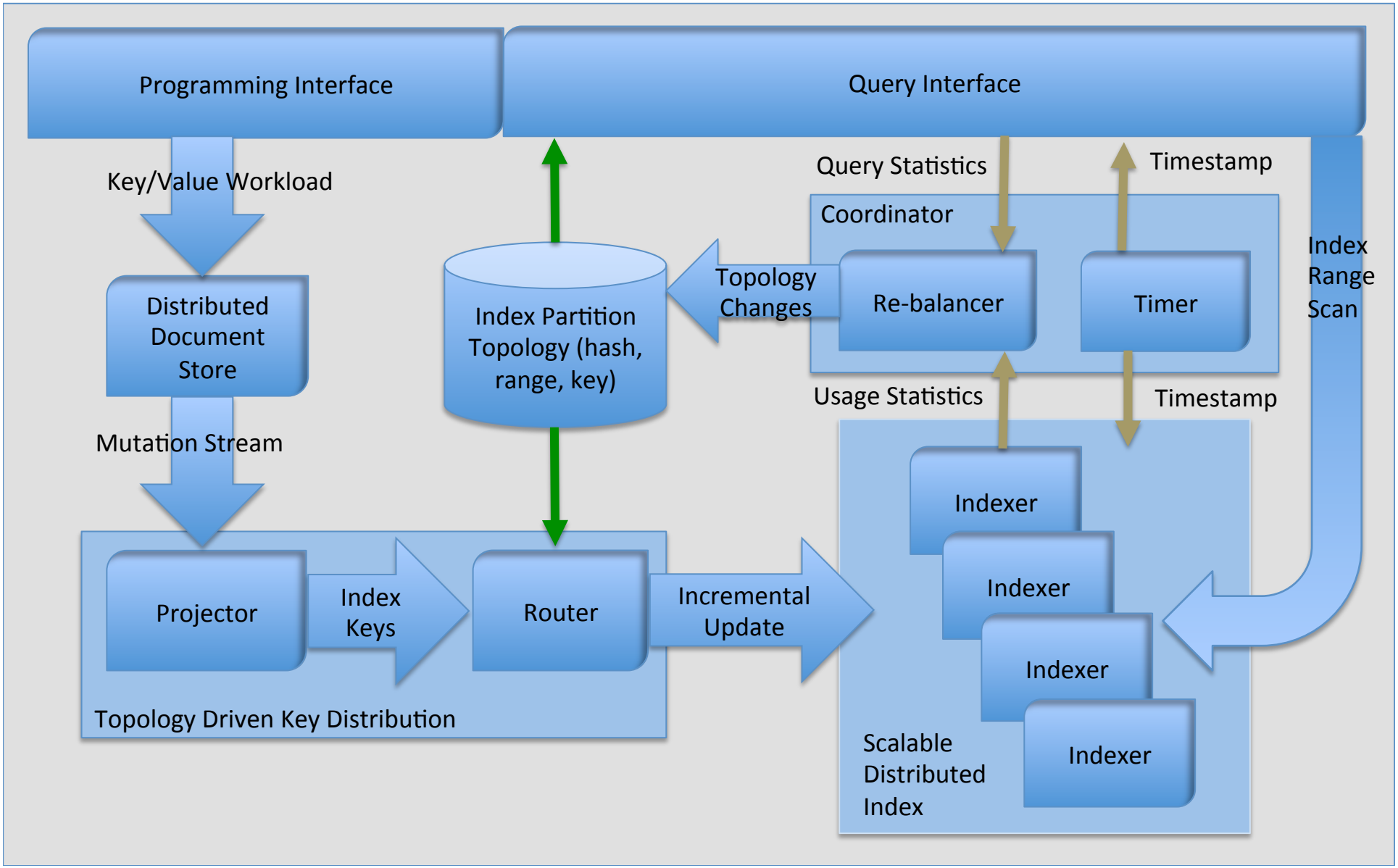
Secondary Indexes Design

Goals

- **Performance** – Support index clustering to avoid scatter/gather + index working set management
- **Independent Scaling** – Enable secondary index to scale independently from Key-value workload
- **Partitioning Scheme Flexibility** – Can support for different partitioning schemes for index data (hash, range, key)
- **Indexing Structure Flexibility** – Can support for different indexing structure (b-tree/trie, hash, R+ tree)
- **Scan Consistency** – Using timestamp to ensure query stability and consistency

Architectural Flow



Key Concepts

- Stability and Consistency
  - Timestamp
  - Scan Stability
  - Data Consistency
  - Stability Timestamp
- Partitioning
  - Index Topology
- Flexible Deployment

Timestamp

- Array of 1024 vbucket seqNo
- Cannot provide exact true wall clock time
- Can provide the following guarantee (if no data loss)
  - Two identical timestamps represents same ordering of events/mutations across documents
  - If timestamp T1 is greater than T2, it means T1 is more recent than T2 (T1 contains the events of T2)
- Can augment with UUID to compare timestamps "equality" across failover boundaries

Scan Stability

- Correctness requirements with respect to index data partitioning and isolation
  - No Tearing Read : A single KV mutation split into 3 secondary keys -> results be consistent across 3 separate key scans within the same query
  - Monotonic Read : Scans for two separate, serialized queries -> 2nd scan result must be at least as recent as the 1st scan
  - Stable Read : Incremental updates while iterating scan results -> Iteration of scan result at T1 must not contain updates occurred after T1
- Can bypass scan stability at query level for better latency
  - Still stable if single key query + partition key is provided

Data Consistency

- Behavioral requirements for observing document values with respect to (1) single document (2) multiple documents (as results of *some* global/ partial ordering of system events/mutations)
- Single Document
  - Default : eventual consistency
  - Timestamp can provide point-in-time consistency
- Multiple Documents
  - Default : eventual consistency
  - Timestamp can provide “bounded staleness” (e.g. at least as recent as T1, RYOW)

Stability Timestamp

- A timestamp to support scan stability and bounded staleness
- Generated periodically by a single authority (by observing seqNo from UPR stream)
- Stable snapshot – all index snapshots (across nodes) being taken at same stability timestamp (snapshots with matching seqNo as stability timestamp)
- Scan Stability – scan results from a stable snapshot
- Bounded Staleness – scan results using a stability snapshot satisfying a given time constraint (e.g. snapshot greater than timestamp T1)

Partitioning

- Index data is partitioned differently than document store in order to improve scan efficiency with data clustering
- Data clustering depends on query pattern and data statistics
  - Low selectivity -> low data clustering | low scalability -> higher cost
- Enable different partitioning scheme to cluster data based on query pattern
  - Partition by Key : Select * from orders where userId='Joe123' -> partitioned by hashing userId
  - Partition by Range : Select * from orders where createDate >= '2014-1-1' and createDate <= '2014-3-31' -> partitioned by range on createDate (orders based on financial quarters per year)
- Index DDL requires partition key (userId, createDate) and partition scheme (key, range)
- Use Scatter/Gather with no data locality on specific index scan

Partition Topology

- Metadata that captures distribution of index data across multiple nodes
- Index is physically partitioned into 'Slices' (similar to vbucket)
  - No fixed number of slices (unlike vbucket)
  - Slices can be put on any index nodes
  - A index logical partition (e.g. range) can be made up of multiple slices (for low selectivity data) for balancing large partition across node -> Handling hot spot (with reduced data locality)
- Partitioning Algorithm (Key/Range) responsible for mapping keys to Index Slices

Flexible Deployment

- Index-only nodes : Specialization of node for indexing workload
  - Workload Separation (from KV node)
  - Data Clustering for Index Scan (vs. document access)
- Should also support co-location of index data with KV node (deployment choice)

Key Modules

- Projector
- Router
- Indexer
- Index Coordinator

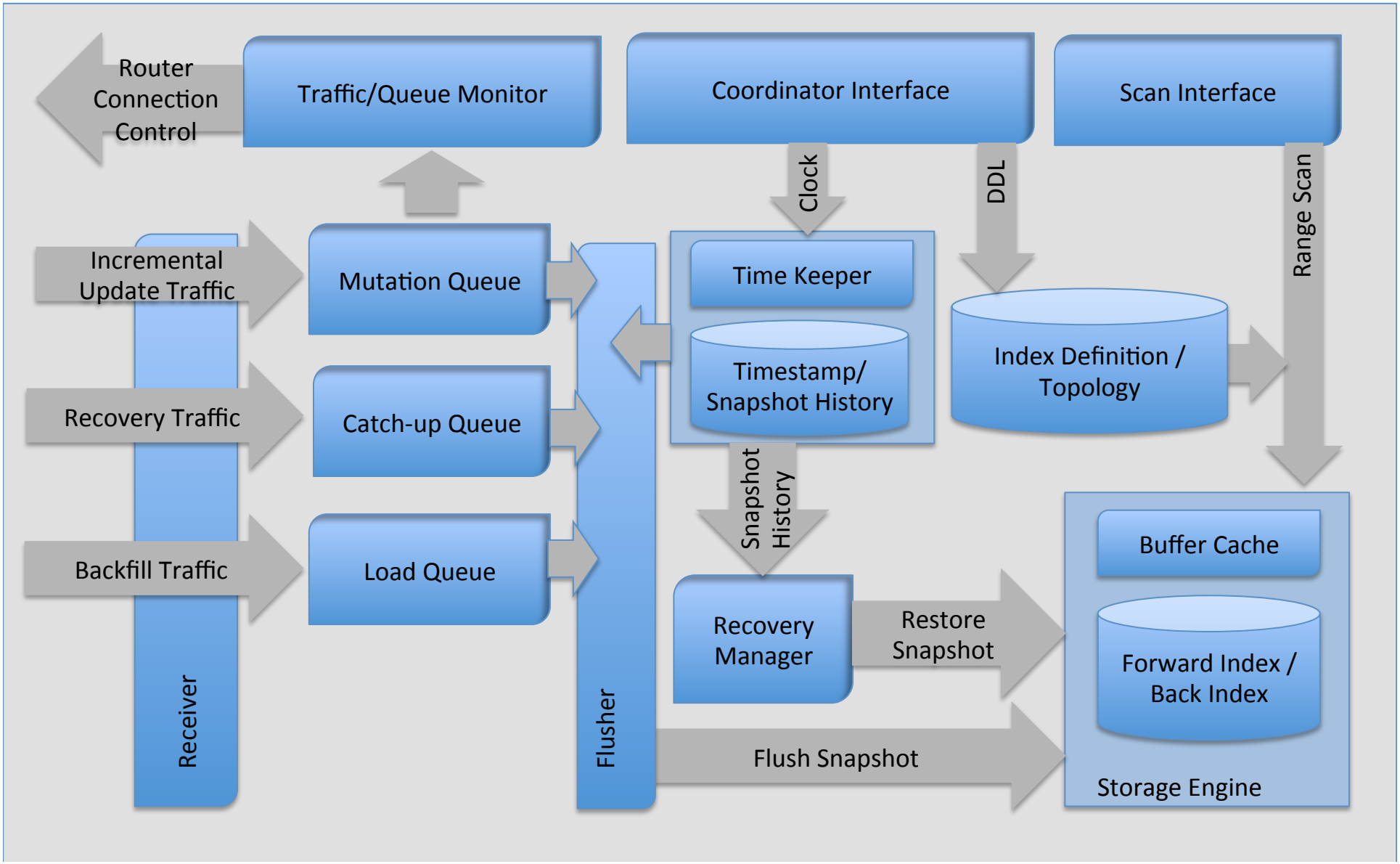
Projector

- Manage UPR connections for clients
  - Client (e.g. index coordinator) initiates connection creation
  - One UPR connection (on N vbuckets) shared with all indexer nodes for incremental update traffic
  - One UPR connection shared will all indexer nodes for backfill (initial index build)
  - Failed or Slow indexer node may drop off the shared connection -> require opening individual UPR connection for “catch-up”
  - Close catch-up connection if catch-up traffic/seqNo matches incremental traffic/seqNo
- Transform/Map UPR mutations into secondary keys based on index definition
- Forward secondary keys to Router for transmission
- Send Sync Messages periodically to recipients for broadcasting current progress (vbuckets’ hi-seq-no)
- Send Control Messages to recipients on topology changes (KV rebalance)

Router

- Route secondary keys to corresponding recipients (e.g. indexer)
  - Use Partition Topology + Partition Scheme to determine indexer nodes for receiving new key
  - Topology is provided by the client when the connection is established
  - Partition Scheme is determined by the index definition
  - For Upsert, broadcast to all indexer nodes for deleting message of old key
- Secondary keys generated from the same KV mutation (seqNo) will be packaged in one message for each recipient

Indexer


Router Connection Control
Traffic/Queue Monitor
Coordinator Interface
Scan Interface
Clock
DDL
Range Scan
Incremental Update Traffic
Mutation Queue
Time Keeper
Timestamp/ Snapshot History
Index Definition / Topology
Recovery Traffic
Catch-up Queue
Snapshot History
Backfill Traffic
Load Queue
Buffer Cache
Recovery Manager
Restore Snapshot
Forward Index / Back Index
Receiver
Flusher
Flush Snapshot
Storage Engine

Index Coordinator

- Manage Index Topology and Definition
  - Replicate Metadata across coordinator replica
- Coordinate indexers bootstrap
- Coordinate rollback due to KV failover
- Generate/Broadcast Stability Timestamp
- Create/Manage Incremental Update + Backfill connection with Projectors

Key Flows

- Overview
  - https://github.com/couchbase/indexing/blob/master/secondary/docs/design/overview.md
- Incremental Update
- Initial Index Load
- Query Scan
- Recovery

Incremental Update

- Design Flow / Detail Steps
  - https://github.com/couchbase/indexing/blob/master/secondary/docs/design/markdown/system.md
  - https://github.com/couchbase/indexing/blob/master/secondary/docs/design/markdown/mutation.md
- Key Points
  - Keys arriving at Indexer -> placed in Mutation Queue
  - Coordinator periodically issues stability timestamp to all Indexers (fixed rate)
    - Frequency can also depends on KV mutation rate, Indexer persistence drain rate, tolerance on staleness
  - Indexer flushes keys from mutation queue to storage upon receiving stability timestamp
    - If keys have seqNo smaller or equal to received stability timestamp
  - Indexer creates snapshot after keys (seqno smaller or equal to stability timestamp) are flushed

Initial Index Load

- Design Flow / Detail Steps
  - https://github.com/couchbase/indexing/blob/master/secondary/docs/design/markdown/initialbuild.md
- Key Points
  - 2 separate Phases : CreateIndex DDL + Index Loading
    - Amortize traffic by sharing same backfill stream for multiple indexes
  - Use backfill queue for index loading
  - Incremental Update is concurrently running for existing index (mutation queue)
  - Termination Criteria:
    - Load Timestamp -> vbucket hi seqNo @ initial load time
    - Backfill Timestamp -> seqNo @ tail of backfill queue
    - Low watermark Timestamp -> seqNo @ head of mutation queue
    - Index is ready for query => (backfill timestamp >= load timestamp)
    - Traffic switch-over to mutation queue => (backfill timestamp >= LW timestamp)

Query Scan

- Design Flow / Detail Steps
  - https://github.com/couchbase/indexing/blob/master/secondary/docs/design/markdown/query.md
- Key Points
  - Indexer serves scan request -> becomes scan coordinator (indexer client can also be coordinator)
  - Use Topology and stability timestamp to locate indexers that can participate in the scan
  - Option to scan on stability snapshot or Tip (no consistency)
  - Scatter/Gather results if needed

Recovery

- Design Flow / Detail Steps
  - https://docs.google.com/document/d/1_zWRJ_VmbZZ1x925lUqunipuTzCJJMsqejtUrbtmVTA/edit?usp=sharing
- Key Points
  - Coordinator responsible for verifying metadata + restart shared UPR connections (update, backfill)
  - Each indexer handles its own recovery -> at its own speed
    - Same steps for system bootstrap, indexer restart, KV rollback
    - Rollback to last good snapshot (from UPR point of view)
    - Start catch-up stream (one for each indexer)
    - Switch to incremental update stream when (catch-up queue timestamp > mutation queue timestamp)
  - Projector/router/KV failure -> client is responsible for detecting connection closure and restart connection
    - Handshake between projector/client will determine (1) re-starting seqNo (2) if rollback is needed

Key Dependencies

- ForestDB – Index Storage
  - Snapshot Support
- NS Server
  - Index Only Nodes
  - Master Election / Failover : Index Coordinator
  - Projector Bootstrap / Registration
- UPR
  - Send Old Mutation (if avail in hash table)

Key Backlogs

- Index Re-balancing
- Efficient catch-up for Slow Node / Index Build
- Administration UI
- Diagnostic Stat / Monitor
- Hot Spot Management (Post V1)

Key Issues

- Query Behavior during KV Rollback
- Enable Query with in-memory Snapshot
- Lack of System Dictionary
- Latency between Coordinator Failovers
- Index Notification - Observe

Backup Slides

Architectural Flow