DRILL EXCHANGE ENHANCEMENTS PROPOSAL

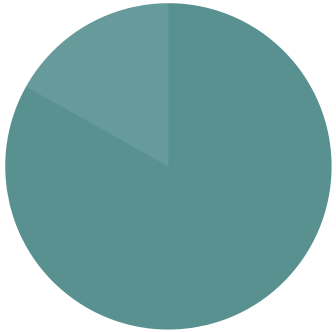
SALIM ACHOUCHE

AGENDA

1. CURRENT EXCHANGE IMPLEMENTATION
    - INTRODUCTION
    - COMMUNICATION INFRASTRUCTURE
    - ANALYSIS
2. EXCHANGE ENHANCEMENTS PROPOSAL

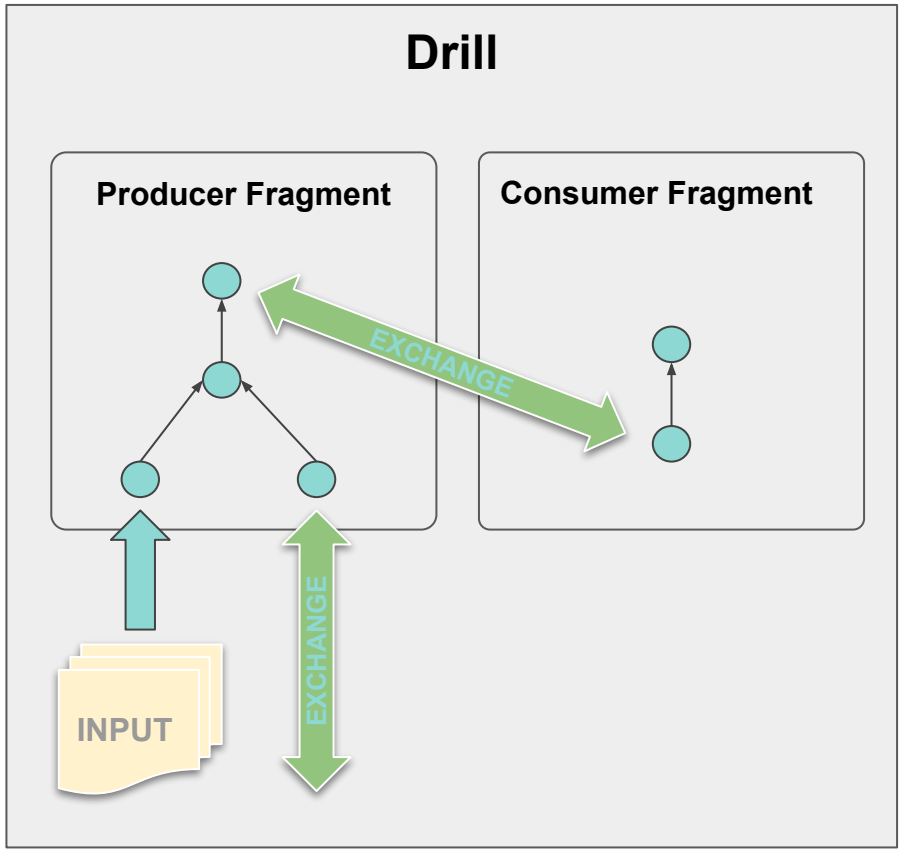
CURRENT EXCHANGE IMPLEMENTATION

INTRODUCTION TO THE EXCHANGE OPERATORS



DRILL EXCHANGE OPERATOR

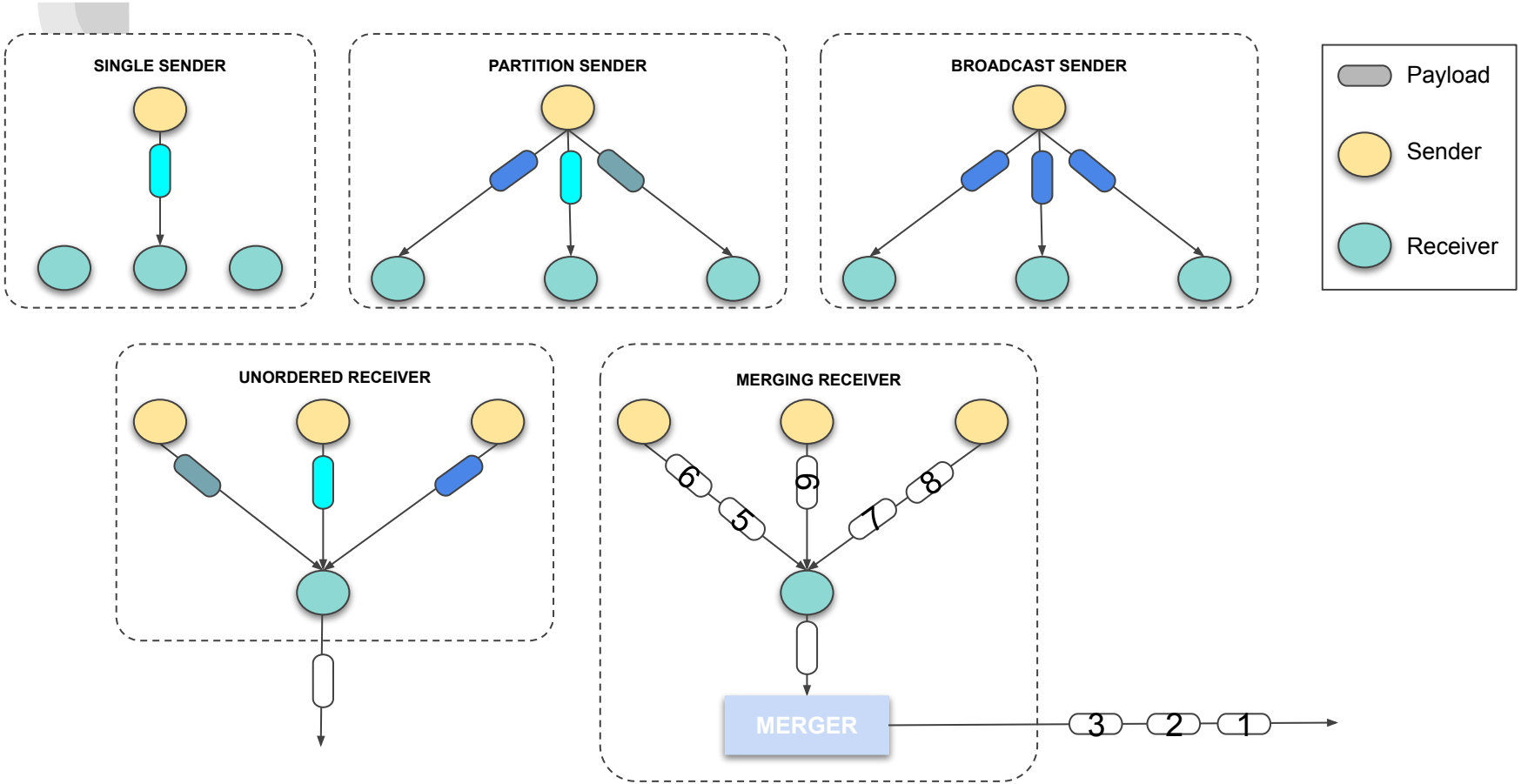
- Not related to Drill Bit Exchange 😀
- The Exchange Operator enables parallel query execution
  - Inter & Intra operator parallelism
  - Links two Drill fragments running concurrently
  - Data flows from the producer to consumer fragments
- Exchange Characteristics
  - Topology -
    - One-to-many (Partitioned / Broadcast)
    - Many-to-one (Union)
  - Supports Ordering
  - Partitioning Strategy
    - Hash
- Exchanges are logical operators
  - Composed of elementary physical exchange constructs
  - Sender & Receiver



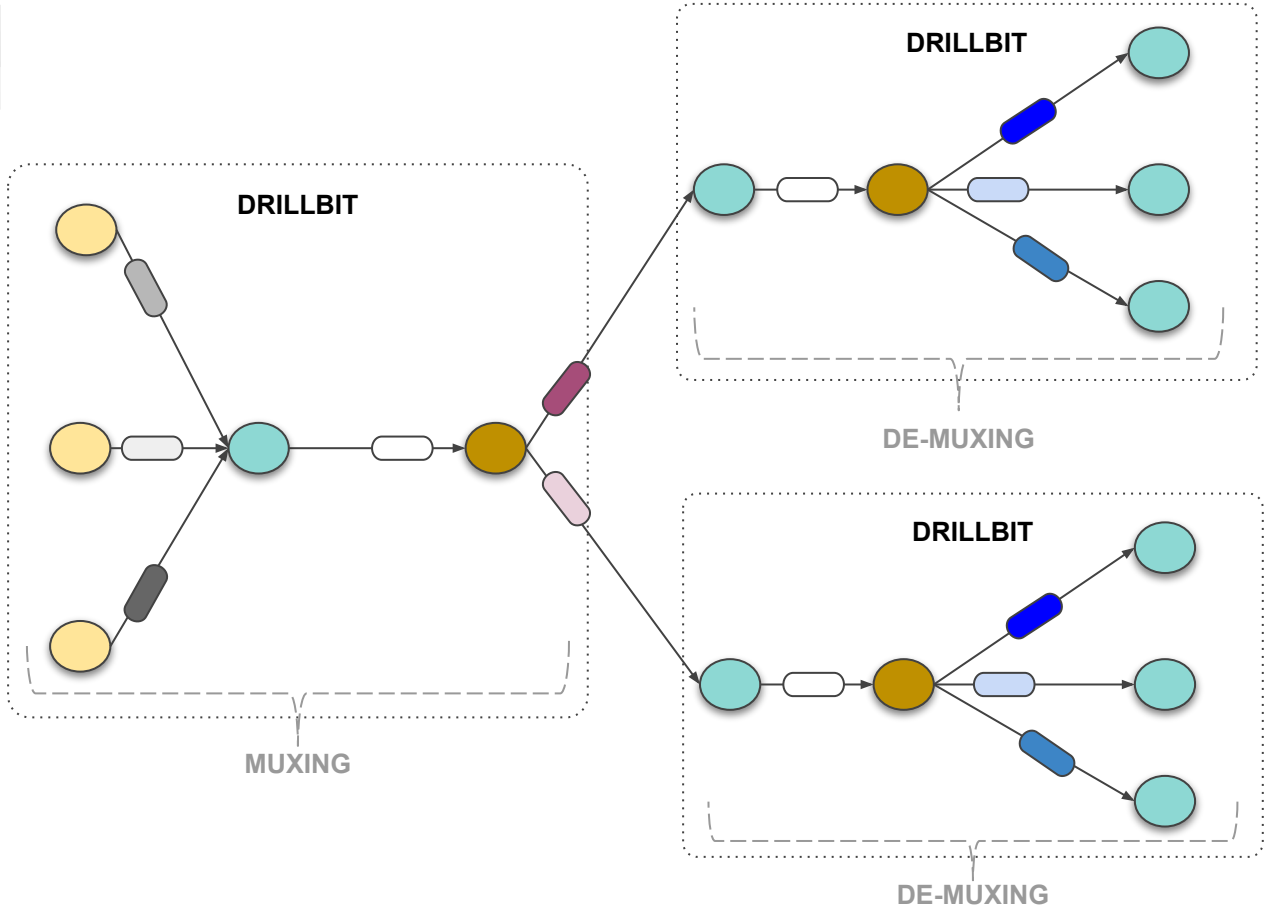
EXCHANGE OPERATORS - *Logical Exchanges*

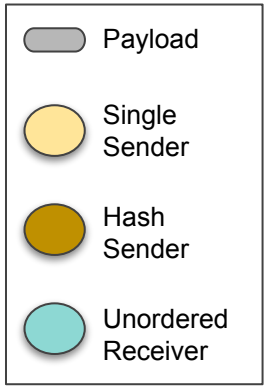
LOGICAL EXCHANGE	RECEIVER	SENDER
BROADCAST	Unordered Receiver	Broadcast Sender
HASH to MERGE	Merging Receiver	Hash Partition Sender
HASH to RANDOM	Unordered Receiver	Hash Partition Sender
ORDERED MUX	Merging Receiver	Single Sender
ORDERED PARTITION	Unordered Receiver	Ordered Partition Sender
SINGLE MERGE	Merging Receiver	Single Sender
UNION	Unordered Receiver	Single Sender
UNORDERED DEMUX	Unordered Receiver	Hash Partition Sender
UNORDERED MUX	Unordered Receiver	Single Sender

EXCHANGE OPERATORS - *Exchange Primitives*



EXCHANGE OPERATORS - *Multiplexing*

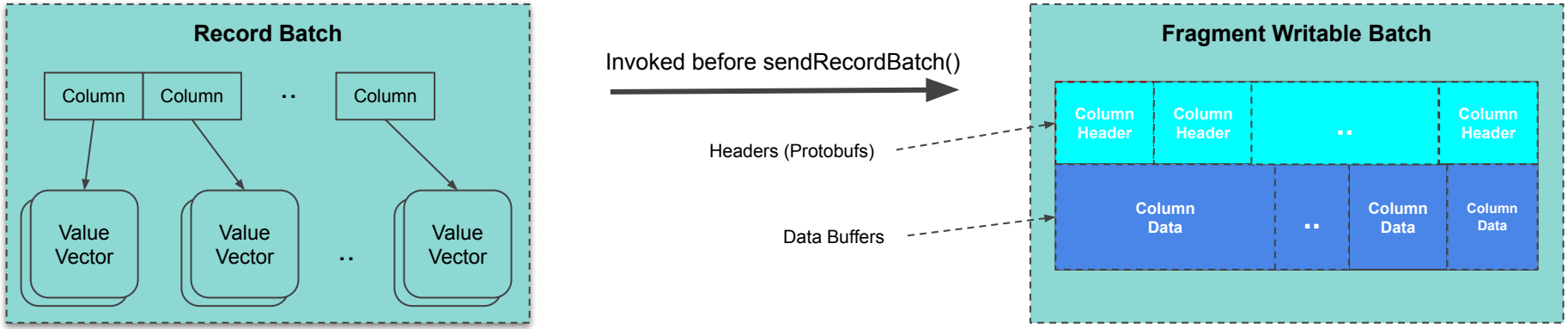




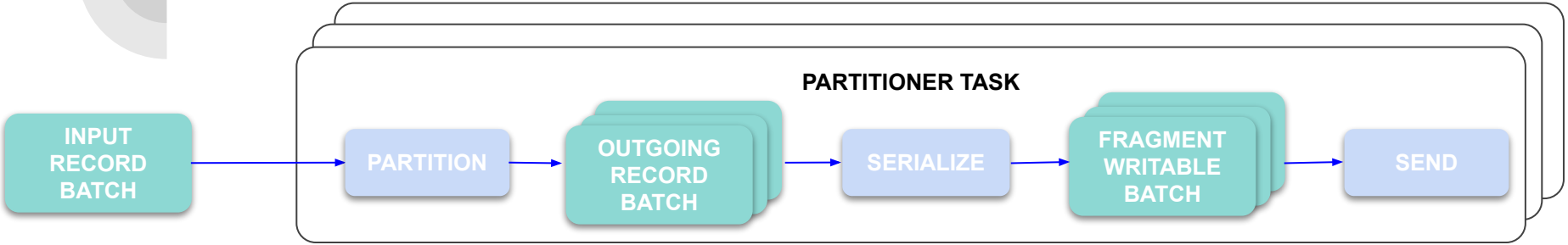
De-Mux phase is currently not enabled by default; in which case, Hash to Random is instead used.

EXCHANGE OPERATORS - *Unit of Exchange*

- RecordBatch is the unit of exchange
  - Holds a batch of rows
  - Input for Sender operators
  - Serialized into FragmentWritableBatch
- FragmentWritableBatch
  - Made up of protobuf headers (describes each column data)
  - One contiguous byte array to hold (nullable info, offsets, and data)



EXCHANGE OPERATORS - *Hash Partitioner Implementation*



IMPLEMENTATION NOTES

- Each Sender Partitioner task targets a different set of receivers
- Outgoing record batches are flushed when they reach 1k rows
- The number of Partitioner tasks
  - Computed based on cost and number of receivers
  - Configured by end user
- The Input Batch read repeatedly (once per Partitioner tasks)

EXCHANGE OPERATORS - *Batch Acknowledgment*

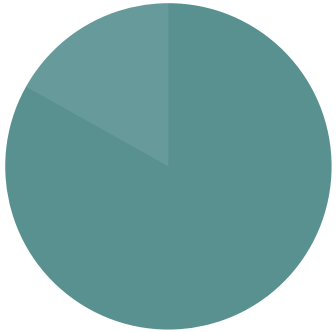
SENDER

- Each Sender is allowed to send upto three batches
- A semaphore of limit three used to implement this logic
- Acknowledgments
  - Received asynchronously
  - Cause the SEND semaphore to increment

RECEIVER

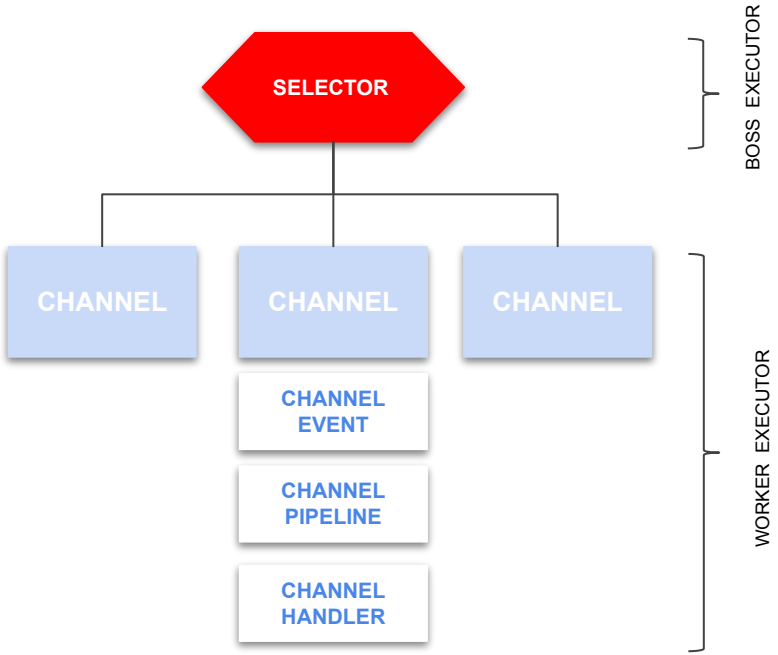
- Acknowledgment logic performed within each minor fragment
- Acknowledgment sent when
  - Number of enqueued batches is below soft limit
    - <buffer-size-per-socket> x <num-senders>
    - buffer-size-per-socket configurable (default: six)
  - Record batch is consumed
- Acknowledgment influenced by the Data Collector used
  - Partitioned → when ordering is used
  - Merged → when ordering is not needed

COMMUNICATION INFRASTRUCTURE

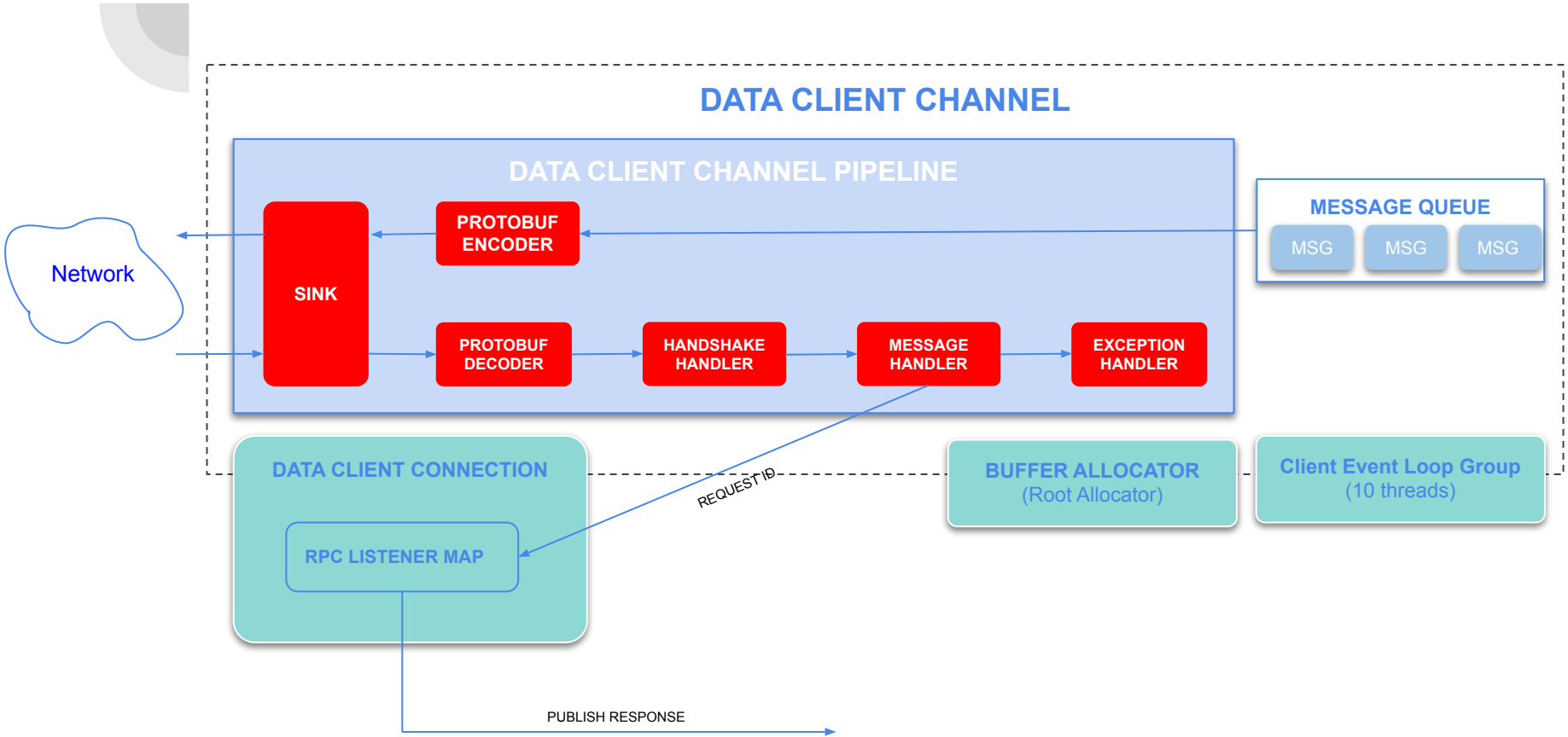


NETTY NIO TRANSPORT

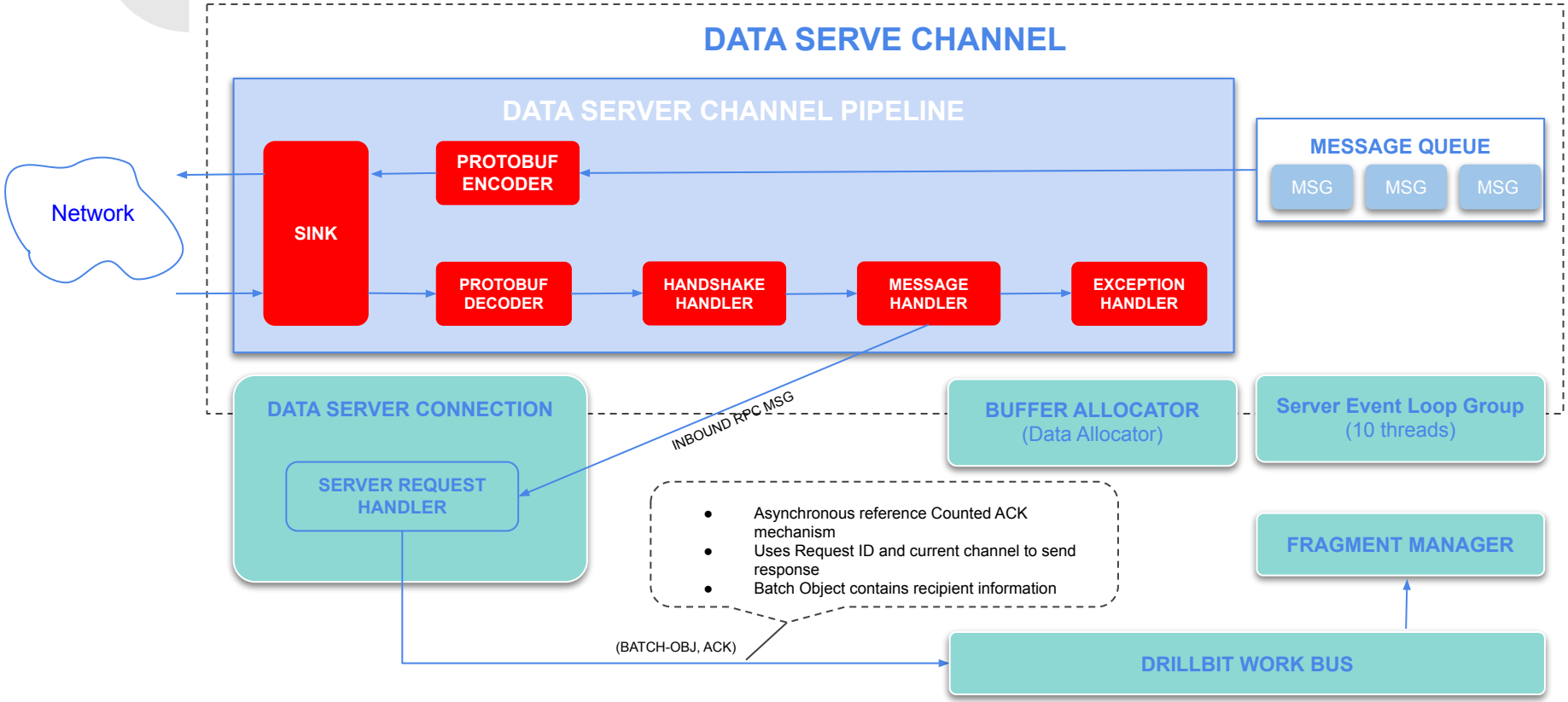
- Netty exposes easy to use NIO wrappers
- Main Concepts
  - Event Loop Group
  - Channel
  - Channel Event
  - Channel Pipeline
  - Channel Handler
- Pipelining makes it easy to support
  - Compression
  - Encryption
  - Authentication
  - Transport Protocols
  - RPC



DRILLBIT RPC - *Send Data*



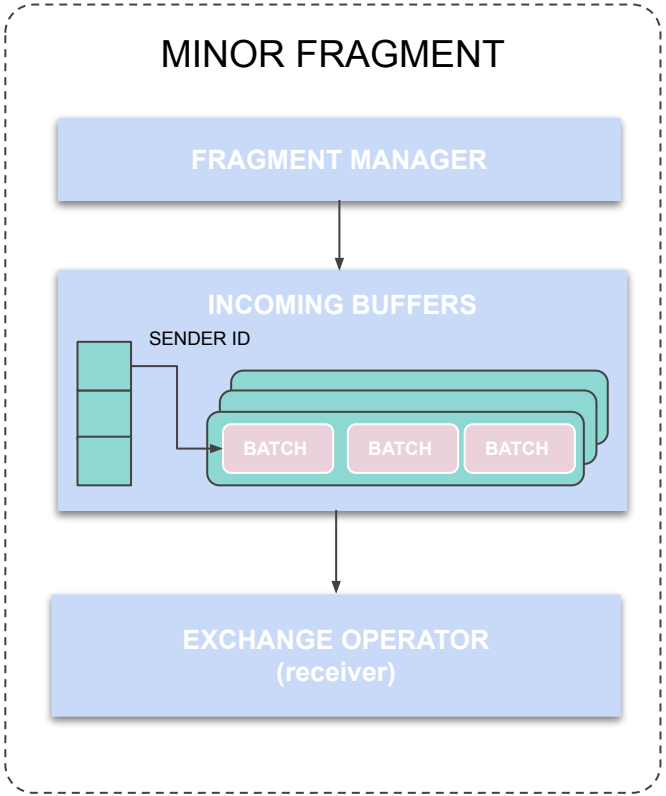
DRILLBIT RPC - *Receive Data*



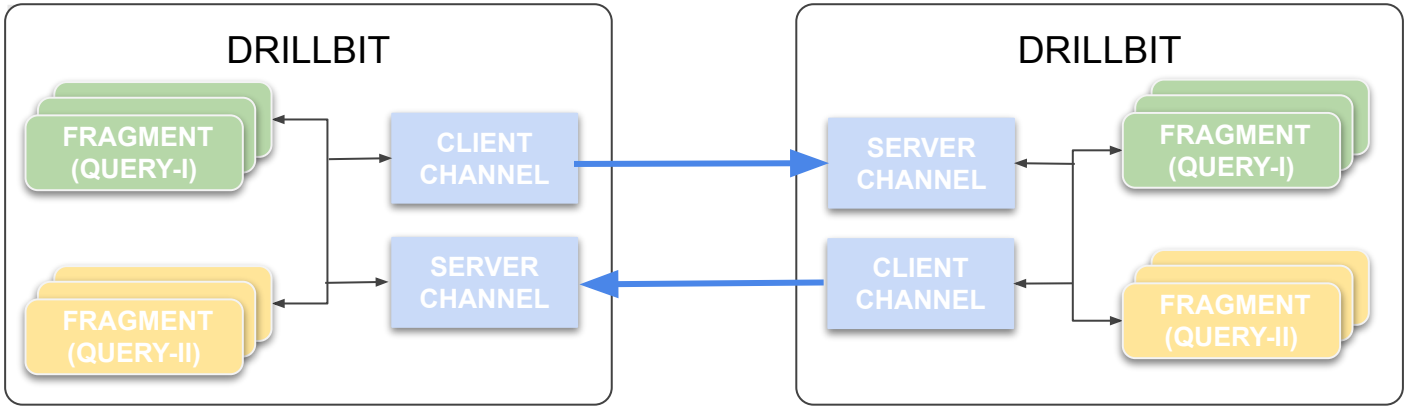
DRILLBIT RPC - *Fragment Batch Queue*

Incoming Record Batches

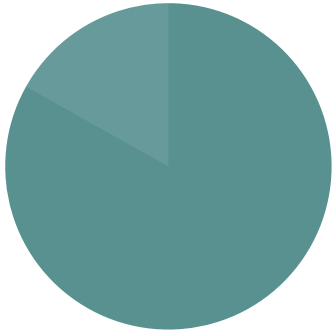
- Received asynchronously
- Enqueued within a single queue (unordered exchange)
- Enqueued in a per sender queues (ordered exchange)
- Acked when consumed by receiver exchange
- Memory ownership transferred to the exchange operator



DRILLBIT RPC - *Network Topology*



- Data Client Connections
  - Drillbit-A opens one data connection to another Drillbit-B
  - A global pool is maintained for connection sharing
- Data Server Connections
  - Drillbit-A creates a server side socket when Drillbit-B initiates a client connection
- Acknowledgment sent from server to client sockets
  - TCP connections are duplex
- Control & User requests use different Client / Server connections



ANALYSIS

ANALYSIS

KEY ASPECTS to CONSIDER

- Scalability
- Resource Management
- Performance

SCALABILITY

- Muxer & DeMuxer exchanges should allow Drillbits to scale
- Not sure why DeMuxer is disabled by default
  - Previous analysis only discussed Muxer issues
  - Need to use local Data Tunnel

RESOURCE MANAGEMENT

- Weak
  - Record batch constrained by number of rows
  - Number of prefetch record batches hardcoded
  - No quota for Send / Receive record batch queues

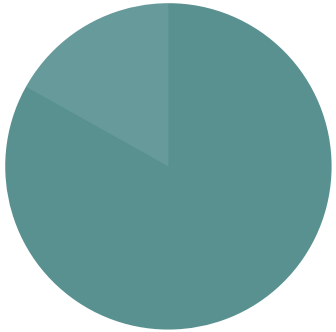
ANALYSIS - continued

PERFORMANCE

- Hash Partition Sender
  - Number of threads has no impact on memory usage
  - Repeated reading of input batch
  - Partitioner tasks can block
    - During flush when receiver(s) didn't ACK sent batches
    - Waiting on other tasks to finish processing the current document
  - Record batch flushed only when full
    - What about latency?
- Record batches hash-partitioned twice
- Demuxed record batches can be small
- No optimization for local record batches
- Shared Data Channels
  - Data tunnels backed by one shared data channel
  - Receivers must consume sent data to avoid blocking other queries

EXCHANGE ENHANCEMENTS PROPOSAL

EXCHANGE ENHANCEMENTS OVERVIEW



EXCHANGE ENHANCEMENTS

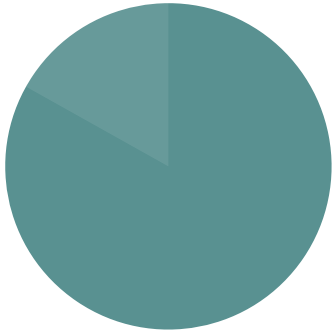
MULTIPLEXING

- Improve processing performance
- Manage resource utilization
- Dynamic control flow

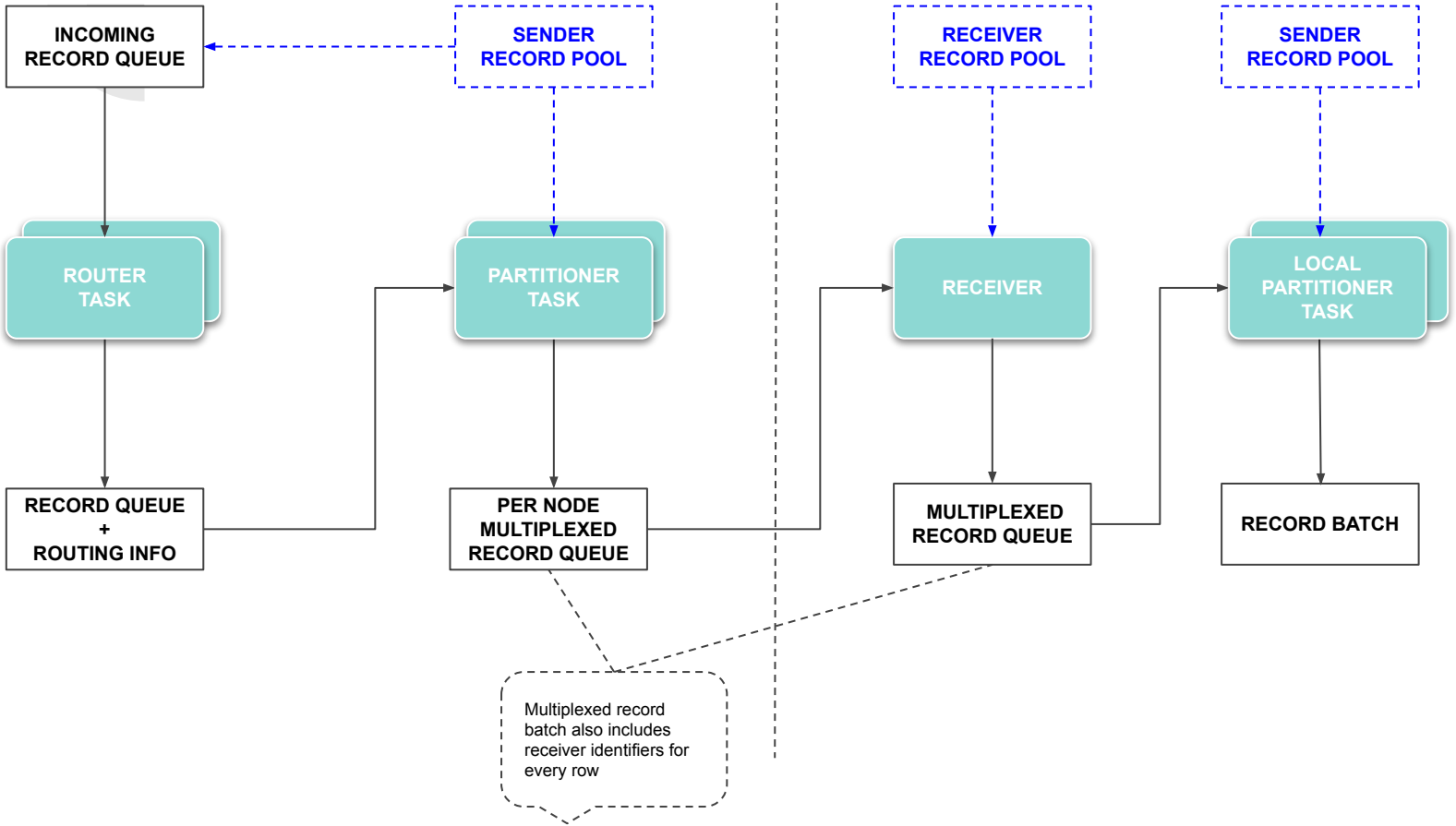
EXCHANGE OPERATORS

- Manage resource utilization
- Optimize local transfers

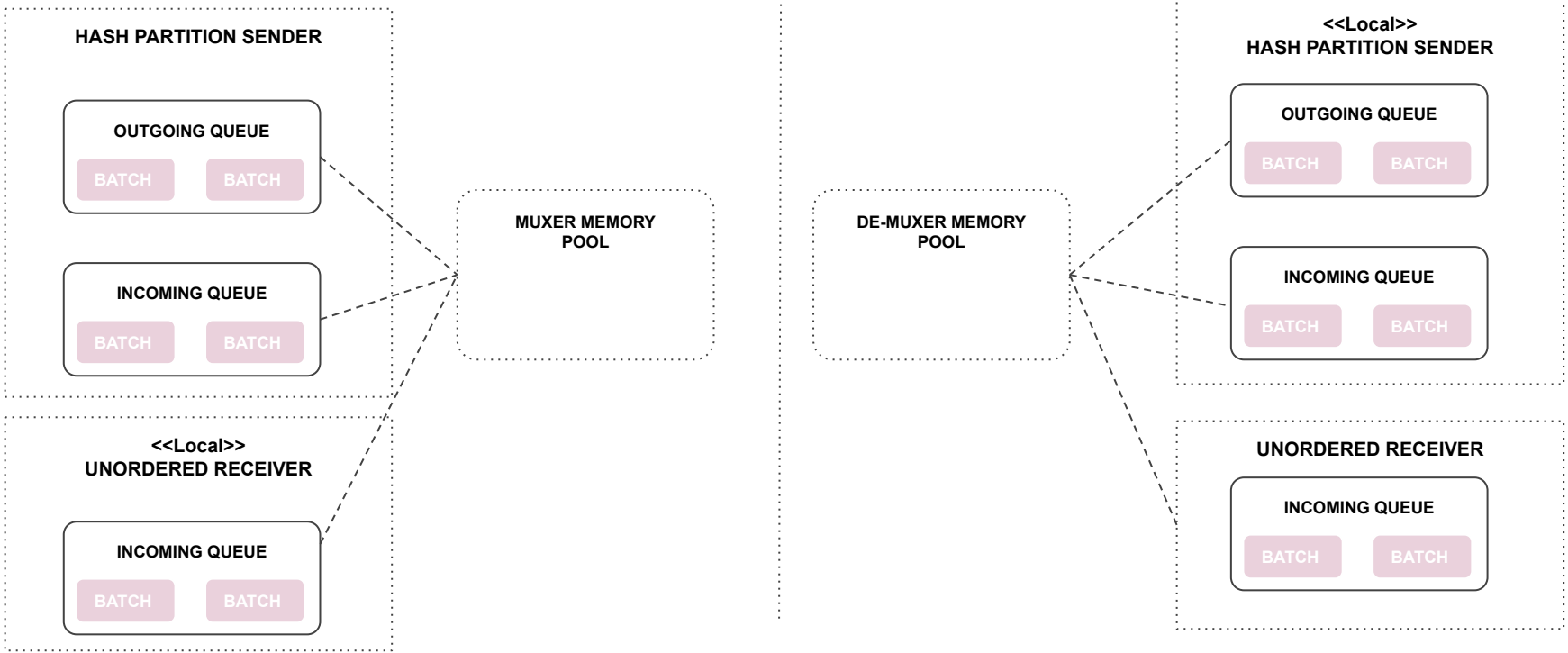
MULTIPLEXING ENHANCEMENTS



OVERVIEW


INCOMING RECORD QUEUE
SENDER RECORD POOL
RECEIVER RECORD POOL
SENDER RECORD POOL
ROUTER TASK
PARTITIONER TASK
RECEIVER
LOCAL PARTITIONER TASK
RECORD QUEUE + ROUTING INFO
PER NODE MULTIPLEXED RECORD QUEUE
MULTIPLEXED RECORD QUEUE
RECORD BATCH
Multiplexed record batch also includes receiver identifiers for every row

RESOURCE MANAGEMENT



RESOURCE MANAGEMENT - Continued

Sender given a fixed memory budget

- Incoming record batch pool
  - “incoming-pool-num-records” = “incoming-record-pool-size” / “record-batch-size”
  - Ideally
    - “incoming-record-pool-size” ~ [10% - 20%] of sender-memory-budget
    - 1Mb <= “record-batch-size” <= 4Mb; promotes concurrent processing
    - “incoming-record-queue-size” at least equal to the number of Router tasks
- Outgoing record batch pool
  - “outgoing-pool-num-records” = “outgoing-record-pool-size” / “outgoing-record-batch-size”
  - Ideally
    - “outgoing-record-pool-size” ~ [70% - 80%] of sender-memory-budget
    - 256Kb <= “outgoing-record-batch-size” <= 1Mb
    - Smaller size improves latency while larger size decreases receiver’s processing overhead

Receiver given a fixed memory budget

- “receiver-pool-num-records” = “receiver-incoming-record-pool-size” / “sender-outgoing-record-batch-size”
- “receiver-window-size” = “receiver-pool-num-records” / “number-of-senders”

The Receiver window size importance diminishes as the number of node receivers increase; the protocol should be able to handle a value of zero (currently, the lowest value is one)

RESOURCE MANAGEMENT - Continued

Local Sender

- Exchanges should use small record batches
  - When number of local receivers is large
- Midstream operators can increase the record batch size
  - Minimize the framework execution overhead

Local Receiver

- Incoming record queue can be smaller than number of senders
- The exchange protocol should automatically advertise the window size
  - A value of zero means the sender cannot send

RESOURCE MANAGEMENT - Continued

EXAMPLE -

- **Drill cluster made up of 100 nodes**
- **Aggregate memory budget 150Mb**
- **Hash Sender**
  - **Memory budget 66Mb**
  - **Incoming Pool**
    - “incoming-record-pool-size” = 16Mb
    - “record-batch-size” = 4MB
    - “incoming-record-queue-size” = 4
  - **Outgoing Pool**
    - “outgoing-record-pool-size” = 50Mb
    - “outgoing-record-batch-size” = 512kb
    - “outgoing-pool-num-records” = 100
- **Receiver**
  - “receiver-incoming-record-pool-size” = 50Mb
  - “receiver-pool-num-records” = 100
  - “receiver-window-size” = 1
- **Local Sender & Receiver**
  - 16Mb for the receiver (4 records)
  - 16Mb for the sender
    - Batch size will be adjusted based on the # of receiving minor fragments

MUXER CHANGES - *Hash Partition Sender*

Exchange Execution Service

- Shared thread pool
- Number of threads should be capped (4 threads ideal)
- Associated tasks never allowed to perform IO or block

Enqueue Task

- Executed by the fragment thread
- Enqueues new record batches
  - Blocks if the queue capacity reached
  - Submits a router task to process the enqueued record batch

Router Task

- Executed by the Exchange Execution Service
- Responsible for computing routing information
  - Computes (“node-id”, “receiver-id”*) for every row
  - Adds (“record-batch”, “routing-info”) into per Partitioner task table
    - Record batches are processed according to original enqueue order
    - The record batch is reference counted
  - “partitioner-id” = “node-id” % “num-partitioner-tasks”
  - Submits new partitioner tasks if they had no work

* Receiver identifier after demuxing

MUXER CHANGES - *Hash Partition Sender*

Partitioner Task

- Executed by the Exchange Execution Service
- Responsible for creating outgoing record batches for a subset of the receiver nodes
  - Picks oldest (“record-batch”, “routing-info”) entry
  - Constructs “outgoing-record-batch”; “receiver-id” column added to the record batch
  - “outgoing-record-batch” flushed when full or “max-wait-time” reached*
  - Decrement input record batch when fully processed
    - When value reaches zero execute Incoming Queue deque logic
  - Amount of execution should be capped
  - A new partitioner task should be submitted
    - Before exiting and there is remaining work

* Get time computed at start of loop and then compared to batch-creation-time

DE-MUXER CHANGES - *Hash Partition Sender*

Enqueue Task

- Executed by the fragment thread
- Enqueues new record batches
  - Blocks if the queue capacity reached
  - Adds “record-batch” into all Partitioners task table
    - Record batches are processed according to original enqueue order
    - The record batch is reference counted
  - Submits partitioner tasks if they had no work

Router Task

- None
- Incoming record batch already contains “receiver-id”

Local Partitioner Task

- Same as Muxer with the exception that Local Data Tunnels should be used