DML and Storage

Agenda

StorageSets

DML and Storage

Agenda

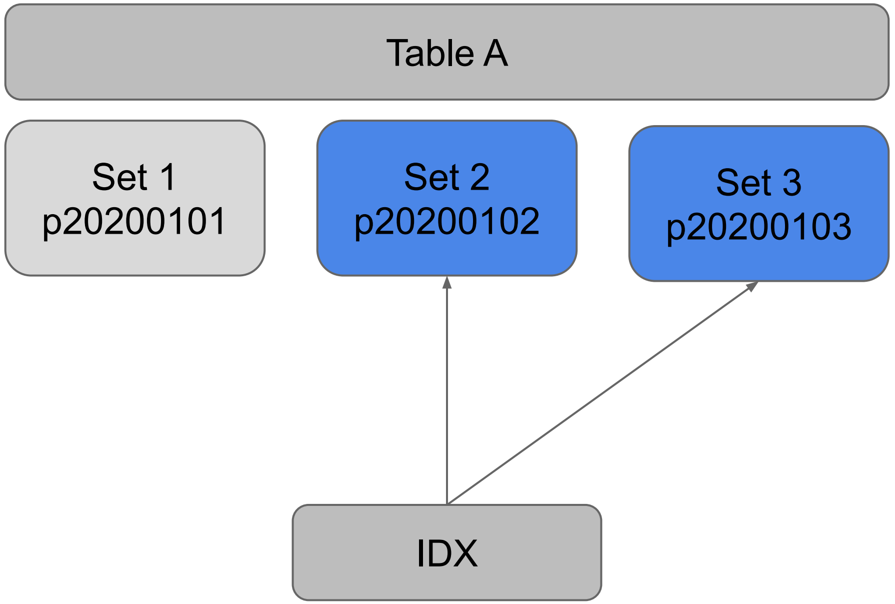
StorageSets

Table Metadata & StorageSets

StorageSet (data commit) metadata contains:

- Colossus path information
- Number of inputs (columnar files)
    - Not one file per column, files are sets of rows.
- Partition Key
    - Implication: one StorageSet per partition
- State
    - PENDING (Preparing to commit)
    - COMMITTED (Live)
    - GARBAGE (Deleted or superseded by newer data)
- Data Stats
    - Column info, sizes, data constraints/ranges

Partitioning



SELECT … WHERE eventDate >= “20200102”

Agenda

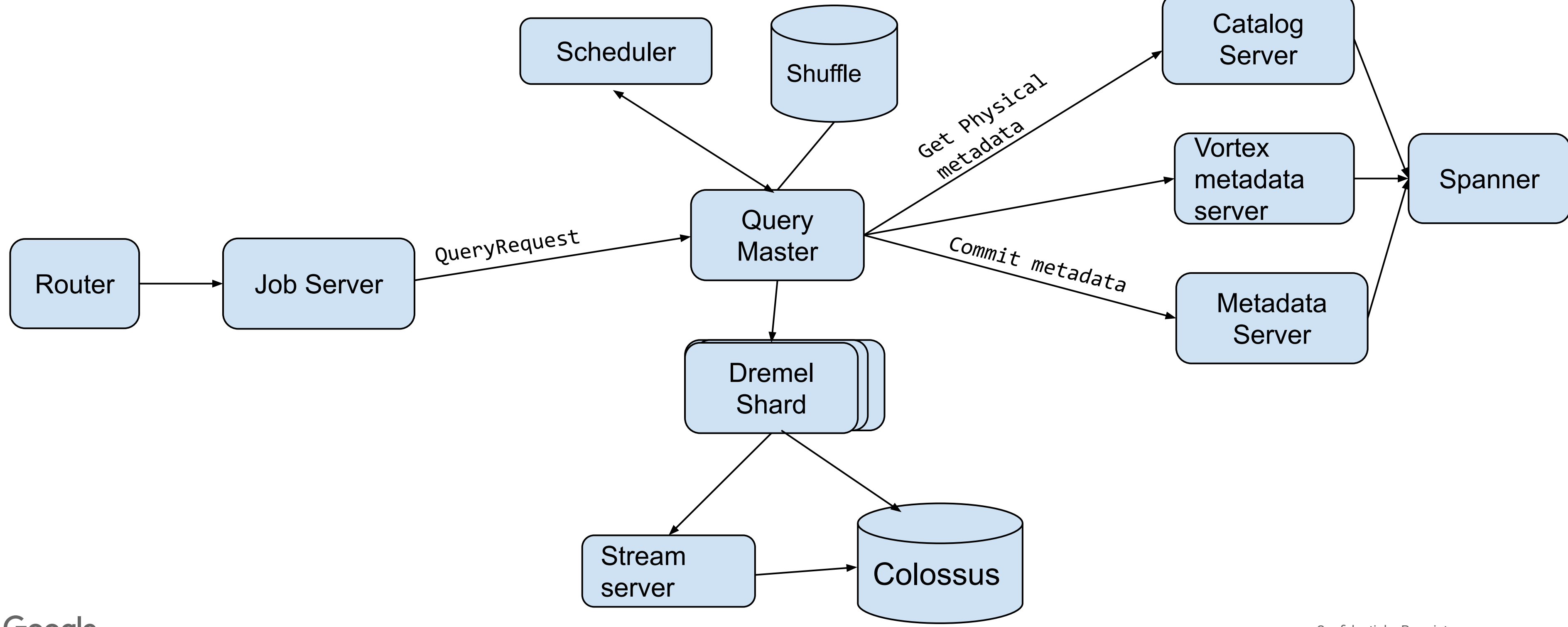
DML and Storage

API

- Jobs.Insert/ Jobs.query

```
Job {
   configuration {
       query {
         query: “DELETE T WHERE field = 1”
        }
    }
}
```

Query path



Scheduler
Shuffle
Catalog Server
Get Physical metadata
Vortex metadata server
Spanner
QueryRequest
Query Master
Router
Job Server
Commit metadata
Metadata Server
Dremel Shard
Stream server
Colossus

Query Startup [Job Server]

1. Authorize that job can be created
2. Extract list of referenced entities from the SQL
3. Authorize input tables and destination table
4. Read table schemas, resolve views transitively
5. Extract referenced fields
6. Perform billing quota checks
7. Persist job
8. Perform concurrency quota check
9. Send QueryRequest to Dremel with the table schemas and URIs

Query Startup Errors

1. Authorize that job can be created
   - Permission denied, bigquery.jobs.create IAM permission
2. Extract list of referenced entities from the SQL
   - Parses SQL, generally reports specific syntax errors
3. Authorize input tables and destination table
   - Permission denied on tables
4. Read table schemas, resolve views transitively
   - Logical view referencing deleted table
5. Extract referenced fields
6. Perform billing quota checks
   - Insufficient quota

Query Startup Errors

7. Persist job
    - Rare, massive job metadata, i.e. millions of load URLs
8. Perform concurrency quota check
    - Usually insufficient user quota
9. Send QueryRequest to Dremel with the table schemas and URIs
    - General query execution errors

If general Google inter-service backend issue

- Connection error

Questions?

Dremel [Query Master, Shards]

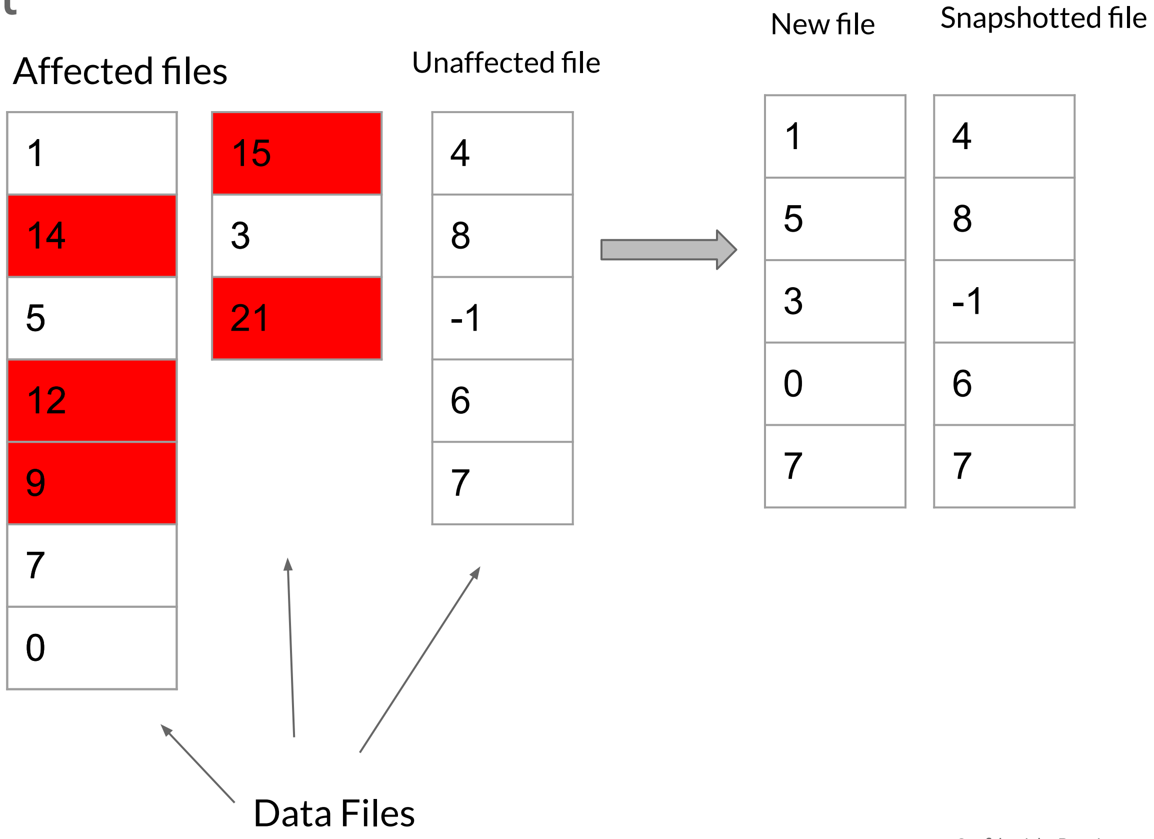
- Logical query planning (Query master)
  - Express in Query Algebra
- Table expansion, partition pruning #1 (Query master)
  - Convert table URIs to actual files, filters unneeded partitions
  - Determines upper bound on cost estimate in billing quota check
- Build distributed plan (Query master)
- Dispatching of partitions - unit of execution (Query master)
- Result file materialization (Shards + Query master)
- Calls MergeStorage to commit changes.

INSERT statement

- Same as EXPORT DATA statement.
- Auto-awesome shuffles data to produce right size files.
- Special kind of partitioning function for partitioned tables.

DELETE statement

DELETE T WHERE field >= 9



Google

Confidential + Proprietary

DELETE statement

Produce rows to re-insert for next stage, deleted file ids to querymaster.

DELETE table1_partitioned_explicit where MOD(employee_id, 2) = 0

Table is partitioned on orderdate TIMESTAMP column

Identify rows to be deleted

```
+-Distributed(SHUFFLE, HASH($2), @1000 AUTO)
  +-Filter mod($1, 2) = 0
    +-Scan(table1_partitioned_explicit)
        $1:employee_id[INT64],
        $2:$file_temp_id[UINT64],
        $3:$row_temp_id[UINT64]
```

```
+-Update(DELETE: table1_partitioned_explicit) file_id:
$2 row_id: $3
   | $40:employee_id[INT64],
   | $41:name[STRING],
   | $42:orderdate[TIMESTAMP],
     +-Sort($2, $3 DESC)
```

DELETE statement

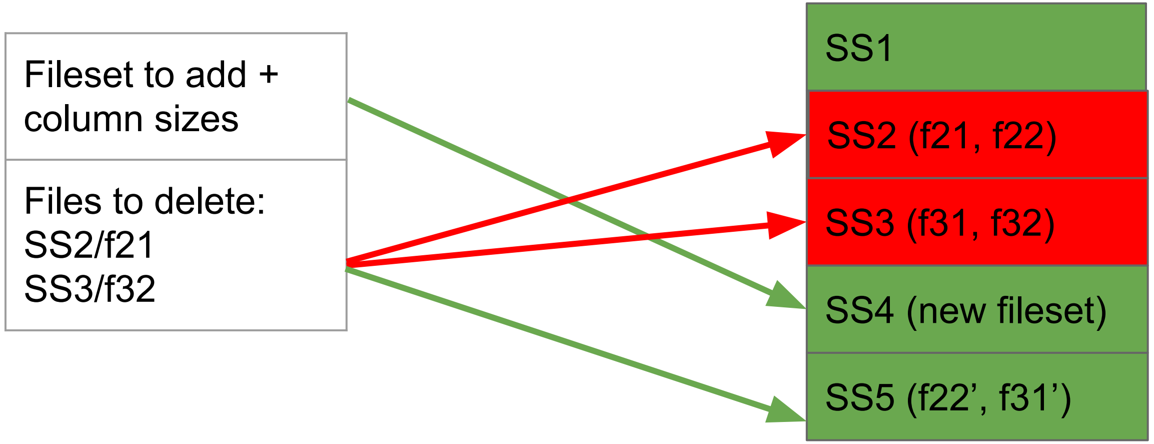
Generate output files, use shuffle for proper filesize

```
+-Distributed(UNION, @1)
    +-Materialize(path: "/path_to_analytical_table" auto_sharding: true
partitioning { time { } column: "orderdate" } max_num_partitions: 100 key(s):
$30)
      | $40 as employee_id,
      | $41 as name,
      | $42 as orderdate
      +-Distributed(SHUFFLE, IDENTITY($30),  AUTO_MATERIALIZE)
         +-Compute
           | $30[INT64]:=
           |   compute_partition_id($42, {time { } column: "orderdate"})
```

Storage Commit

Query master sends MergeStorage RPC to storage metadata server

Maps deleted file ids (8B) to file paths (<200B)



(File pointers, hard links on Colossus)

Questions?

UPDATE statement

Produce updated rows, rows to re-insert, deleted file ids.

Implemented as DELETE old row + INSERT updated row

```
UPDATE table1_partitioned_auto
set employee_id = employee_id + 1
where MOD(employee_id, 2) = 0
```

Identify affected rows

```
+-Distributed(SHUFFLE, HASH($2), @1000 AUTO)
  +-Filter mod($1, 2) = 0
    +-Scan(table1_partitioned_auto)
        $1:employee_id[INT64],
        $2:$file_temp_id[UINT64],
        $3:$row_temp_id[UINT64]
```

```
+-Update(UPDATE: table1_partitioned_auto file_id: $2
row_id: $3
   | $40:employee_id[INT64],
   | $41:name[STRING],
   | $42:_PARTITIONTIME[TIMESTAMP],
   | $43:$is_update[BOOL],
     +-Sort($2, $3 DESC)
```

UPDATE statement

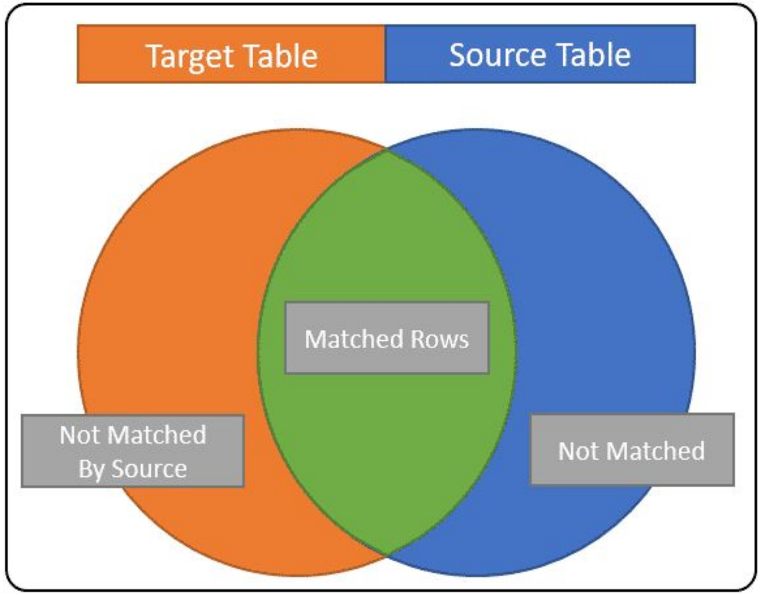
Generate output files

```
+-Distributed(UNION, @1)
  +-Materialize(path: "/path_to_analytical_table" auto_sharding: true partitioning
{ time { } column: "_PARTITIONTIME" key(s): $30)
    | $20 as employee_id,
    | $41 as name,
    +-Distributed(SHUFFLE, IDENTITY($30), AUTO_MATERIALIZE)
      +-Compute
        | $20[INT64]:= if($43, $1 + 1, $40),
        +-Compute
          | $30[INT64]:=
          |   compute_partition_id($42, {time { } column: "_PARTITIONTIME"})
```

MERGE Statement

INSERT, UPDATE, DELETE over single table in one statement

```
MERGE table2 T
USING table1 S
ON T.field2 = S.field2
WHEN MATCHED AND T.field2 > 100 THEN
  DELETE
WHEN NOT MATCHED THEN
  INSERT(field2) VALUES(field2)
```



Google

Confidential + Proprietary

Questions?

DML common issues

- Small number of concurrent non-insert DML jobs per table (2). This avoids too many conflicts.
    - Non-insert DML Statements ultimately cause garbage collection of entire Storage Sets (SSs) - there can be no overlap of these SS's in concurrent jobs
    - Without limit, many wasted slots in failed jobs
- Limited number of outstanding DML jobs (20). Having more concurrent usually doesn't help.
- Metadata update rate limit at job insert time.
- Unknown time after the job is done is very long. Usually caused by commit latency. Often increased by large number of partitions.
- Users doing single row inserts or single row updates. Generally undesirable pattern.

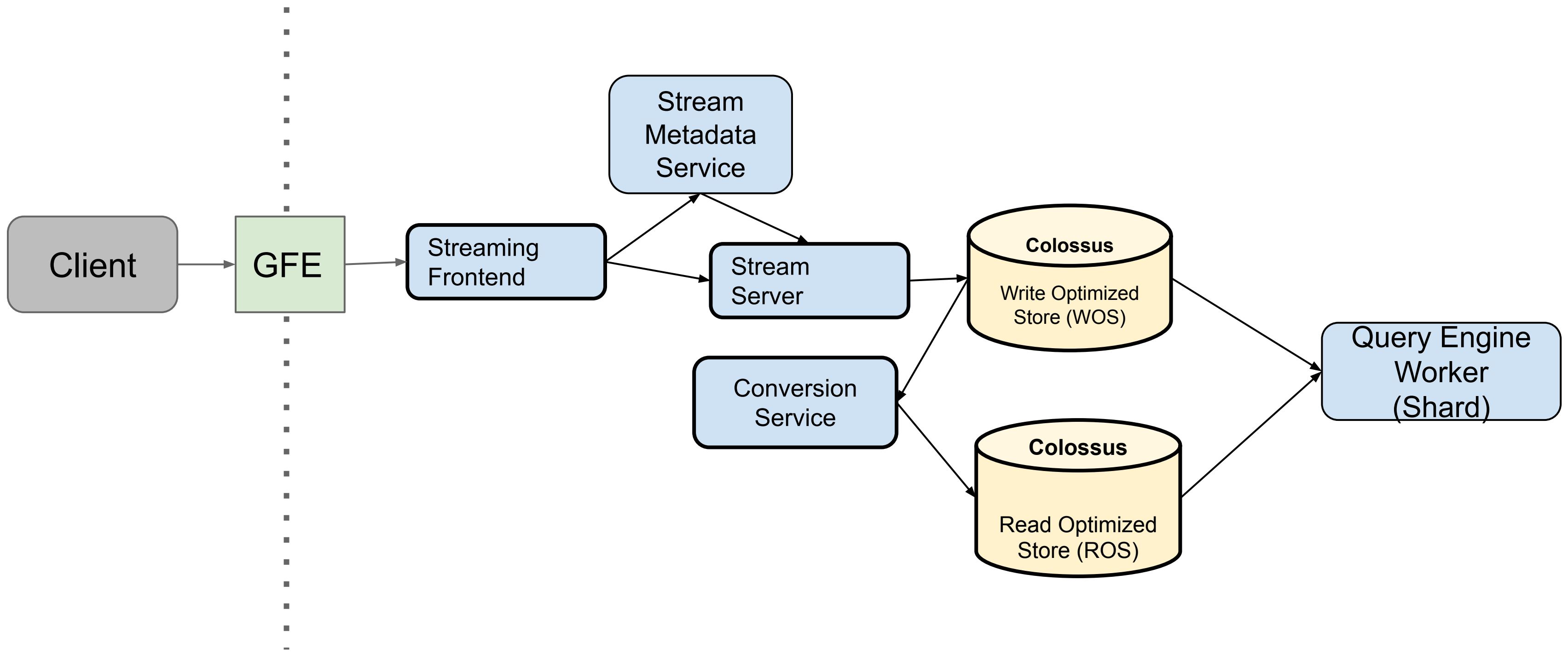
DML common performance bottlenecks

- Updating rows with no locality. For example, few rows from all files. Causes rewrite of entire table, even with partitions. Consider using clustering.
- Commit time sometimes longer than actual execution time. Runs several auxiliary queries.
    - SS causes “Byte Counting Query”, required for Billing Quota Check
- Narrow mutations (or deletions) with locality not as fast as they should be. May be the files are large.

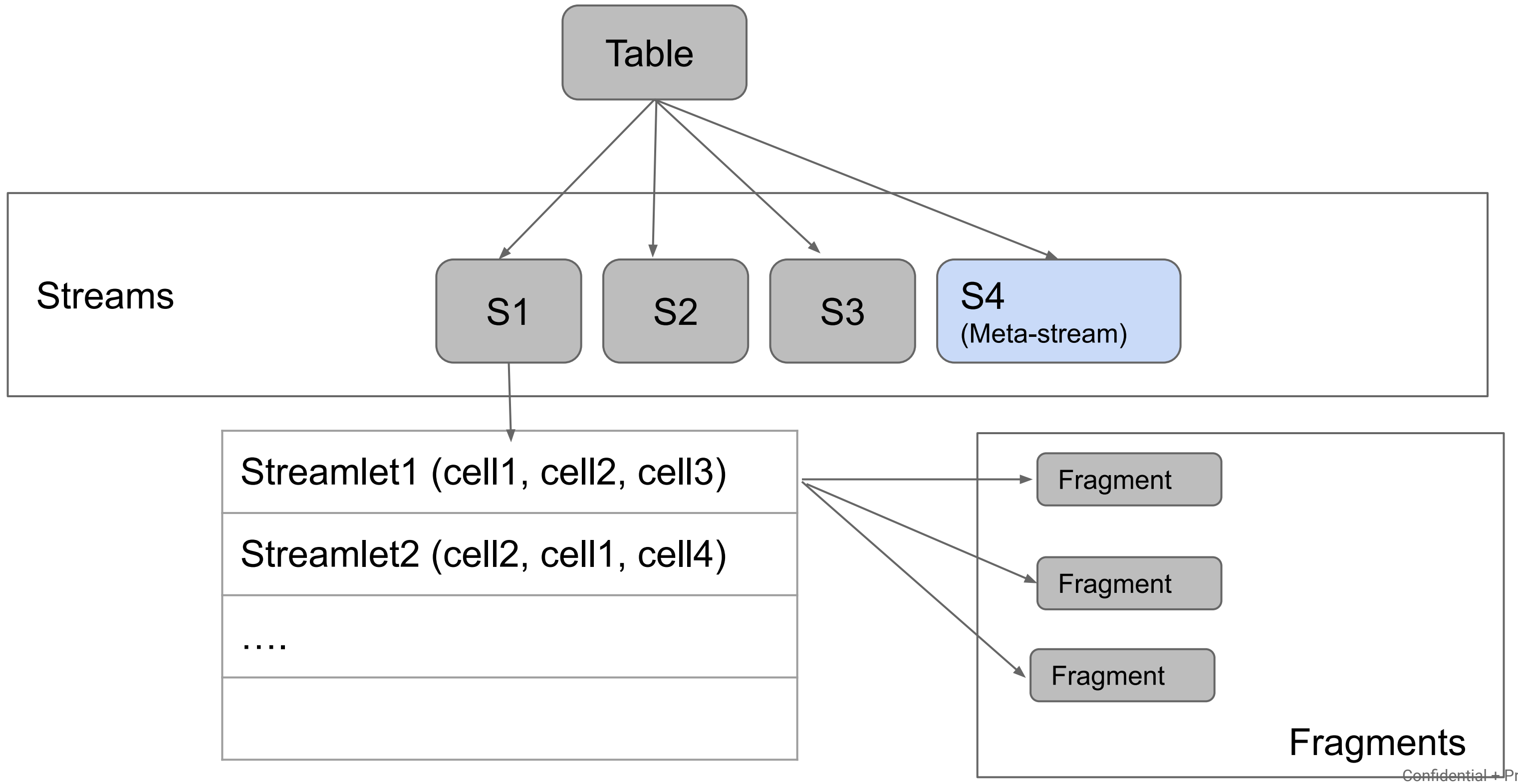
(Batch) Load jobs

- Supports loading from GCS. Parquet, AVRO, ORC, JSON, CSV formats are supported
- Run on Dremel just like a regular query that writes to a table.
- Performance can be managed through reservations.
- Per table limit of 1500 per day. Limited to prevent metadata blow up due to microbatching.

Streaming High Level Architecture



Physical metadata



Append API

- Create Stream against a table

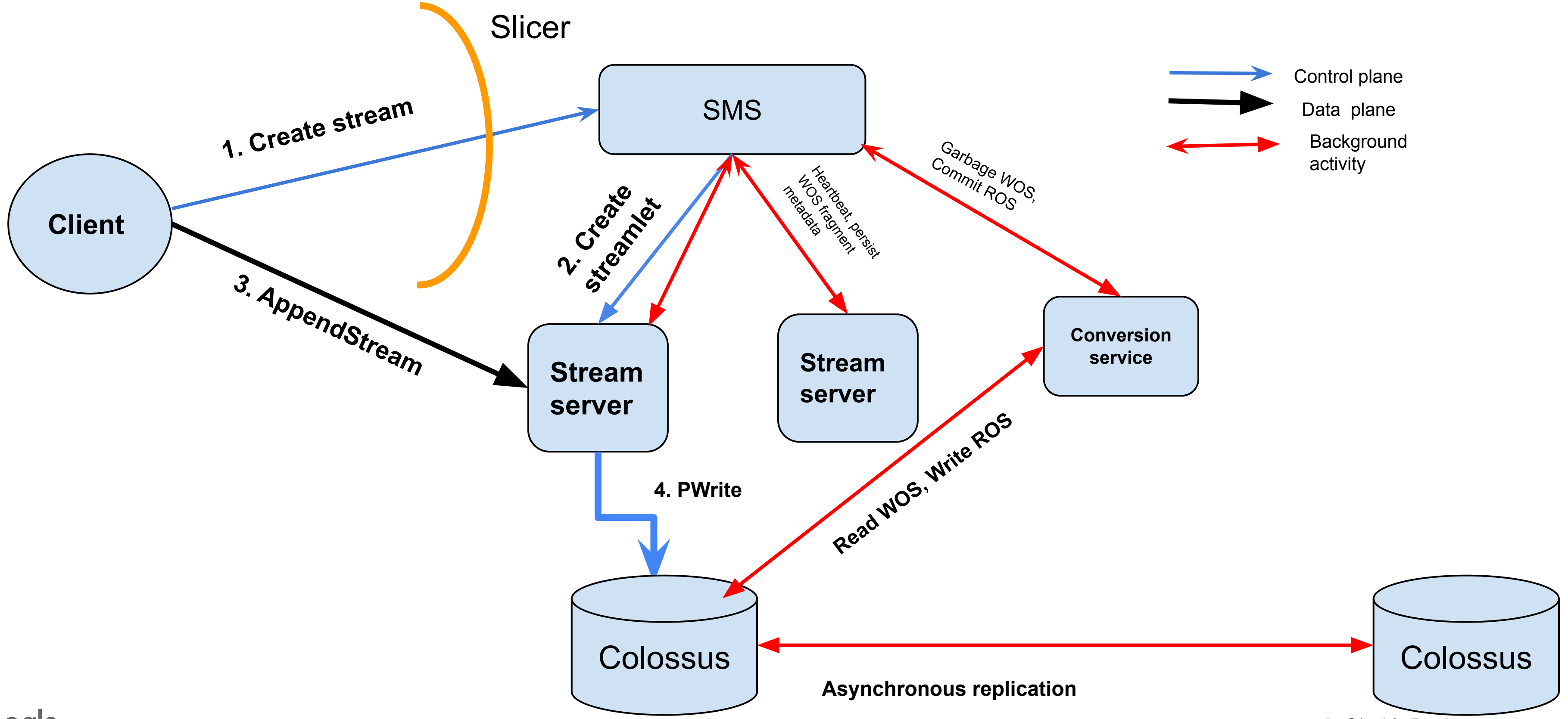
```
StreamReference s = CreateStream(TableReference, Options)
```

- Append rows to a stream

```
AppendResult result = Append(StreamReference s, RowSet rows, [uint64
sequence_number]);
AppendResult {
   ErrorCode err;
   Uint64 sequence_number;
}
```

- Flush stream
- Finalize stream

Life of an append



Slicer
SMS
Client
1. Create stream
2. Create streamlet
3. AppendStream
Stream server
Stream server
Conversion service
Heartbeat, persist WOS fragment metadata
Garbage WOS, Commit ROS
4. PWrite
Read WOS, Write ROS
Colossus
Colossus
Asynchronous replication
Control plane
Data plane
Background activity