Spanner: Becoming a SQL System

SIGMOD 2017, Chicago, May 16

D. Bacon, N. Bales, N. Bruno, B. Cooper,
A. Dickinson, A. Fikes, C. Fraser, A. Gubarev, M. Joshi,
E. Kogan, A. Lloyd, Sergey Melnik, C. Taylor, R. Rao,
D. Shue, M. van der Holst, D. Woodford

Google

What is Spanner

- Distributed transactional data management system
- Globally replicated, highly-available managed service
- Backs hundreds of mission-critical services at Google
    - AdWords, Google Play, Photos, etc.
    - 10s of millions QPS, 100s of petabytes, 5,000+ databases
- Publicly available on Google Cloud Platform (subset): http://cloud.google.com/spanner
- Builds on OSDI'12 paper
    - ACID transactions, replication, fault-tolerance
- This talk: making Spanner a SQL DBMS

Agenda

- Background
- SQL interface
- Distributed query processing
- Lessons learned

In the paper:

- Blockwise-columnar storage

Background

Logical data model

```
CREATE TABLE Singers (
  SingerId INT64 NOT NULL,
  SingerName STRING(MAX)
) PRIMARY KEY(SingerId);

CREATE TABLE Albums (
  SingerId INT64 NOT NULL,
  AlbumId INT64 NOT NULL,
  AlbumTitle STRING(MAX),
) PRIMARY KEY(SingerId, AlbumId),
  INTERLEAVE IN Singers;
```

SingerId	**SingerName**
1	Beatles
2	U2
3	Pink Floyd

SingerId	**AlbumId**	**AlbumTitle**
1	1	Help!
1	2	Abbey Road
3	1	The Wall

Database sharding

- Shard: horizontal slice of database, key-range partitioned
- Rows that agree on `SingerId` are co-located
- Can be physically interleaved

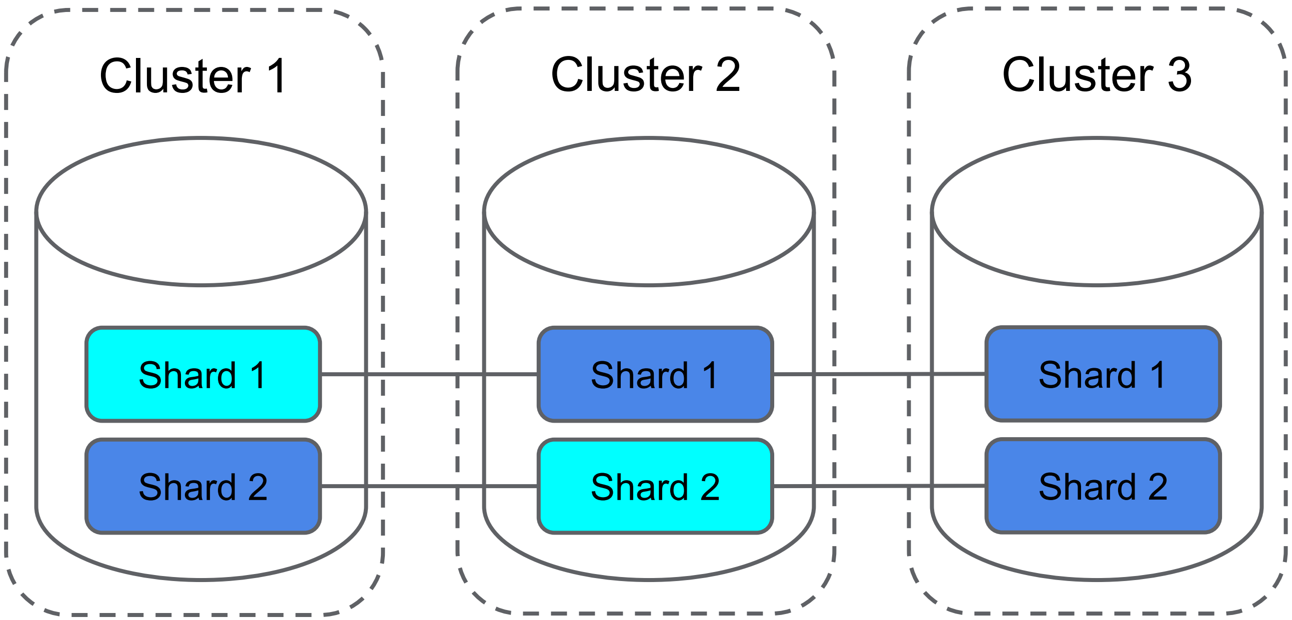
1	Beatles	
1	1	Help!
1	2	Abbey Road
2	U2	

Shard 1: SingerId ∈ (-INF, 3)

3	Pink Floyd	
3	1	The Wall

Shard 2: SingerId ∈ [3, +INF)

Shard replication



- Replication uses Paxos
- Sync and async replication protocols
- Leaders responsible for writes
- Non-leaders serve reads, may be behind

(cyan)	Leader
(blue)	Non-leader

Replica placement

Example: geo-replication of a mission-critical database

Google

Transactions

details in OSDI'12

- Pessimistic locking + timestamp versioning (MVCC)
- Externally consistent: respect wall-time order
  - Every Tx occurs at a timestamp T
  - Via atomic and GPS clocks
- Snapshot transactions: non-blocking
  - See consistent state of entire database at some timestamp T
  - Strong reads: effects of all Tx committed up to now
  - Stale reads: pick T in bounded past
- Read-Write transactions
  - All writes are buffered and committed at end of Tx
  - 2PL within a Paxos group
  - 2PC across Paxos groups
  - Tx use write-ahead redo log
  - Log entry is committed by replicating to a quorum of replicas

SQL interface

Common SQL dialect

- Standards-compliant
- Type system aligned with programming languages
  - INT64, FLOAT, STRING (UTF8), TIMESTAMP (nanoseconds)
  - Reduces impedance mismatch
- First-class support for nested data
  - ARRAY and STRUCT types
  - Protocol Buffers: schematized binary objects (currently internal only)

- Significant language design work across teams
- Shared with other Google systems: BigQuery/Dremel, F1 (Ads), etc.

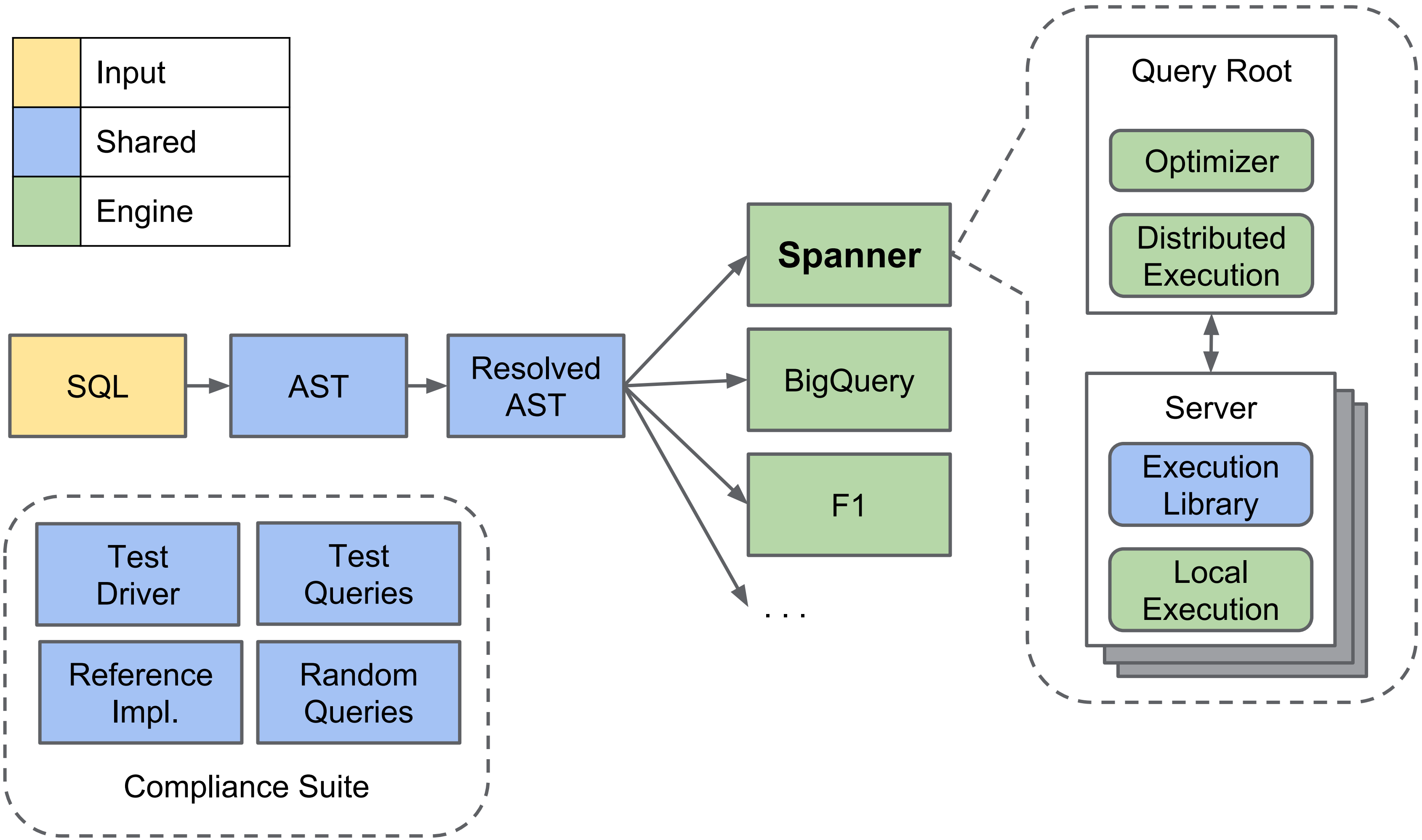
Sample query: name & titles

```
SELECT s.SingerName,
       ARRAY(SELECT a.AlbumTitle
             FROM Albums a
             WHERE a.SingerId = s.SingerId) titles
FROM Singers s
WHERE s.SingerId BETWEEN 1 AND 5
```

SingerName STRING	**titles** ARRAY<STRING>
Beatles	[Help!, Abbey Road]
U2	[ ]
Pink Floyd	[The Wall]

- Easier to use than outer joins or multiple roundtrips

Same query semantics across systems



Distributed Execution

Distributed query execution

- Tightly coupled architecture
  - Query processor inside the database server
  - Typical design for standalone DBMSes but less so for distributed systems
- Challenge of scale: data never sits still
  - Continuous resharding (due to load, capacity, config changes, ...)
  - Shard boundaries may change while query is running
  - Shards may become temporarily unavailable during query execution
  - Alternative replicas: near/far, loaded/idle, caught-up/behind
- Mechanisms used in Spanner
  - Query routing: key-range rpcs + range extraction
  - Parallelizing execution: partition work by shards, push it down
  - Dealing with failures: restartable query processing

Query routing: key-range rpcs

- Routes requests to row ranges
  - E.g., `WHERE SingerId BETWEEN @low AND @high`
- Hides complexity of locating data
- Finds nearest, sufficiently up-to-date replica for given concurrency mode
- Retries automatically
  - Unavailability, data movement, schema updates, ...
- Clients cache sharding information
- Clients cache "location hints" for queries
  - Send query to right server without extra hops or query analysis
  - E.g., `Singers/SingerId[@low]`

Query routing: range extraction

```
SELECT * FROM Albums
WHERE (SingerId = 1 AND AlbumId >= 10) OR
      (SingerId IN (2,3) AND AlbumId != 0)
```

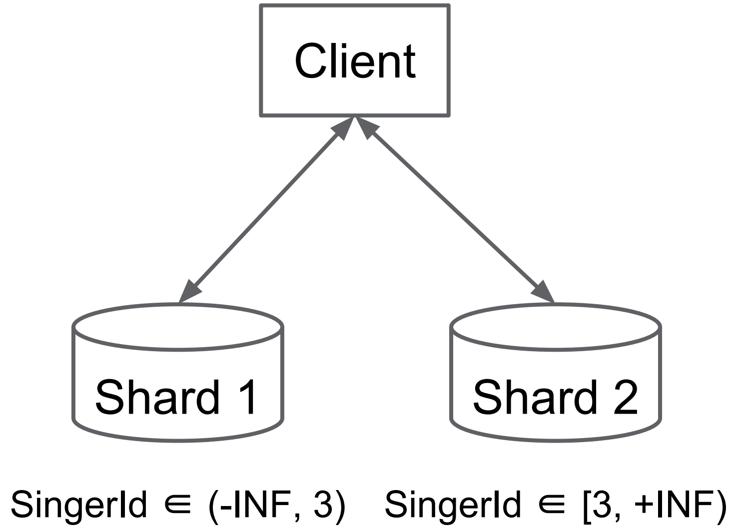
- Also used for restricting scan ranges
- Computed at runtime
  - May access data
- Uses efficient data structure
  - Filter tree (in the paper)

SingerId	AlbumId
[1..1]	[10, +INF)
[2..2]	(-INF, 0)
[2..2]	(0, +INF)
[3..3]	(-INF, 0)
[3..3]	(0, +INF)

Parallelizing Execution

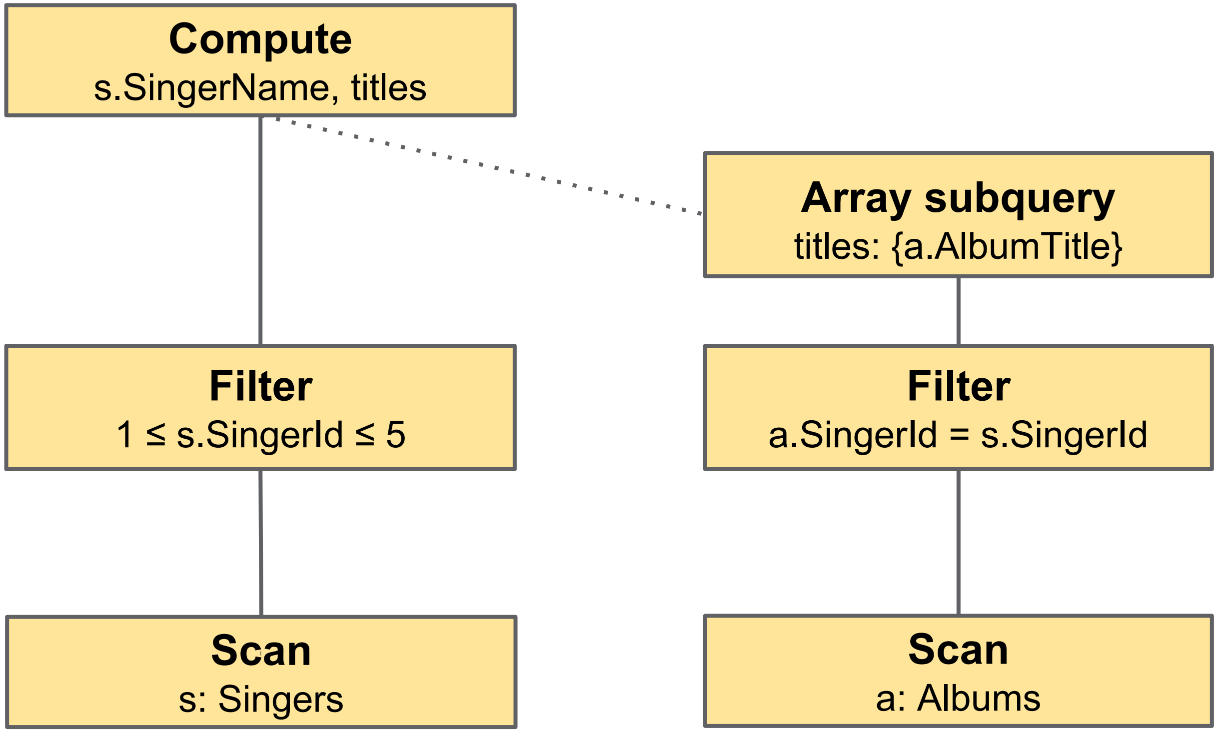
Parallelizing execution

```
SELECT SingerName, ARRAY(SELECT ...) titles
FROM Singers WHERE SingerId BETWEEN 1 AND 5
```

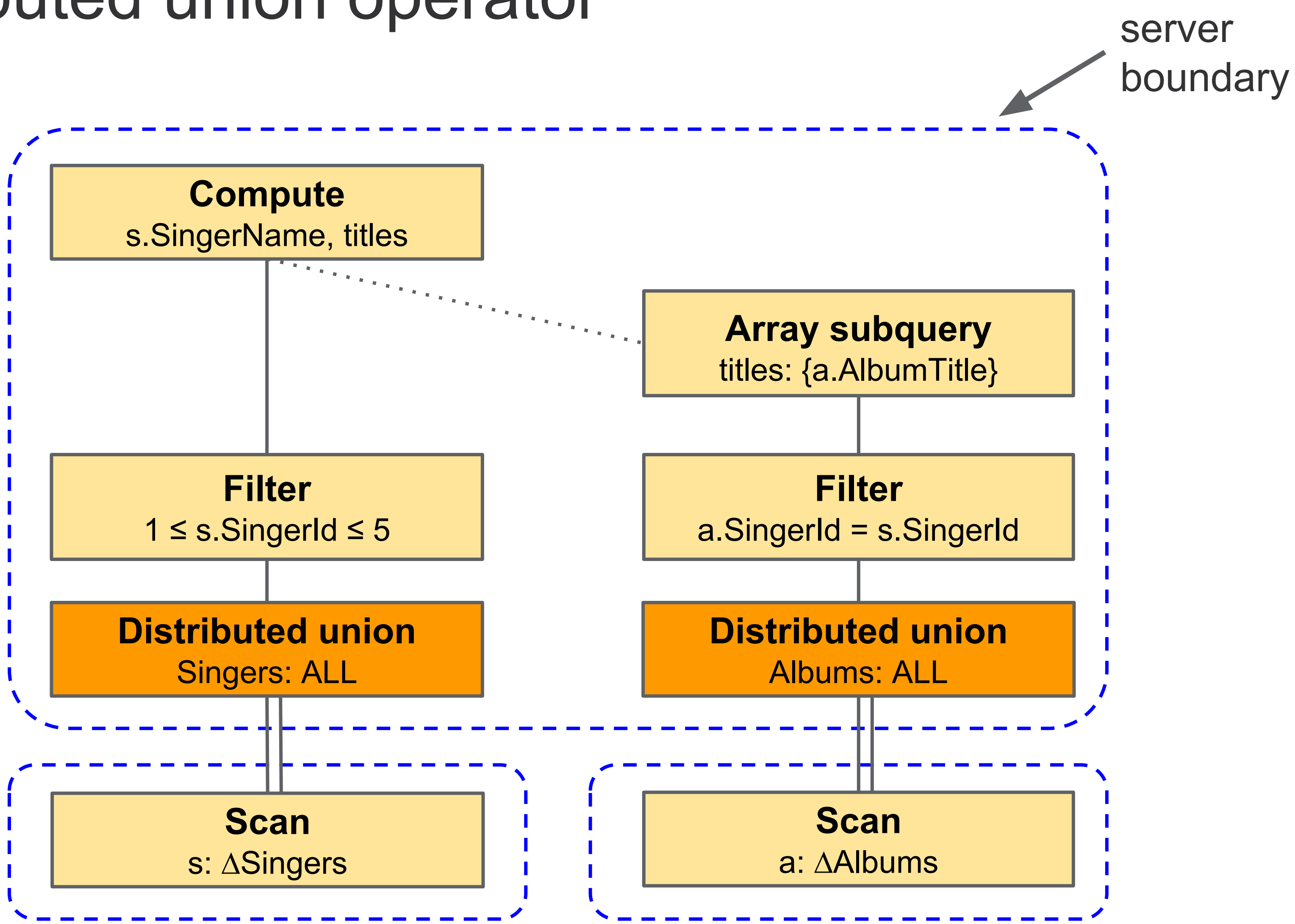


- Assume fixed shard boundaries for now

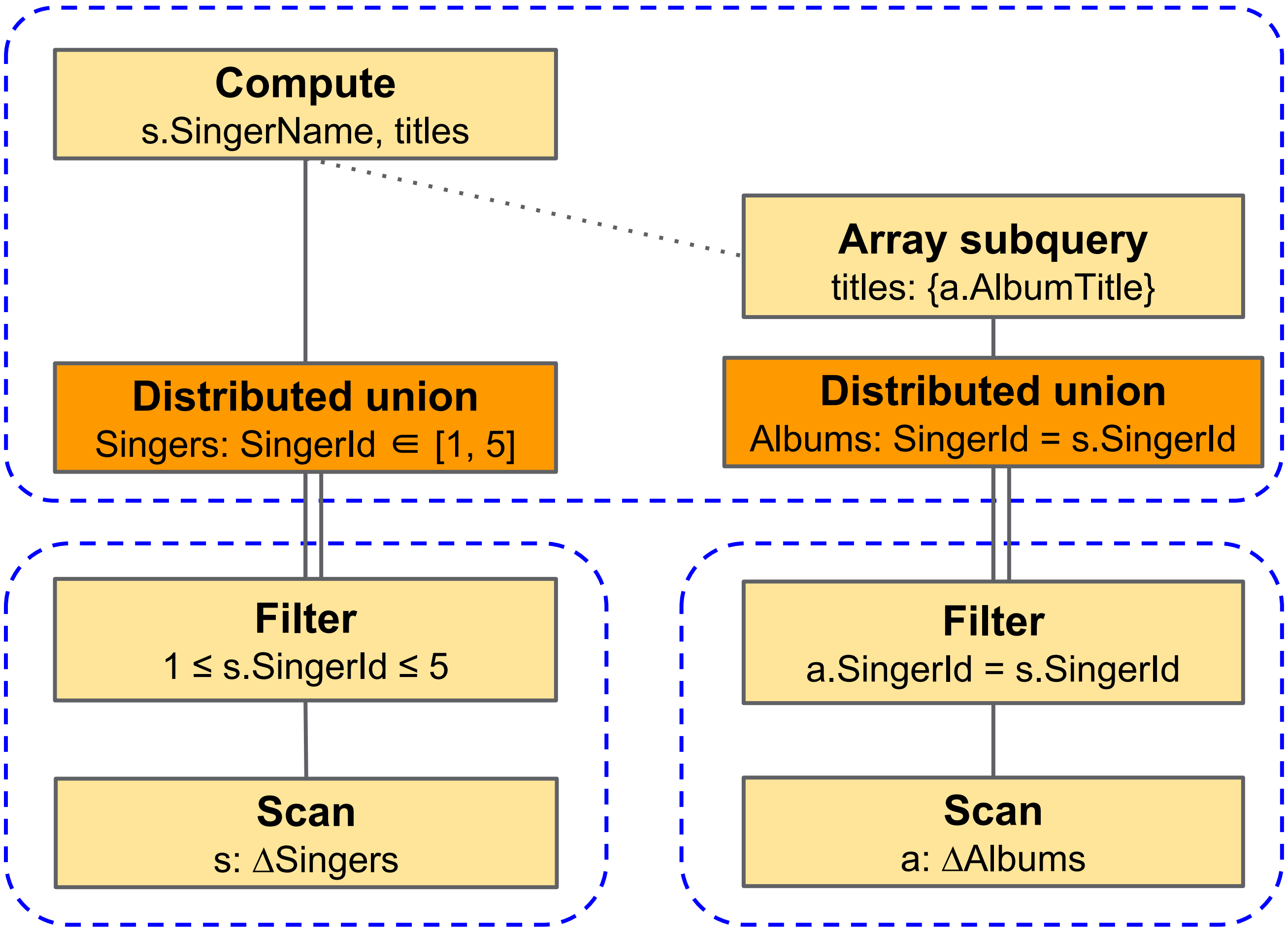
Initial logical plan



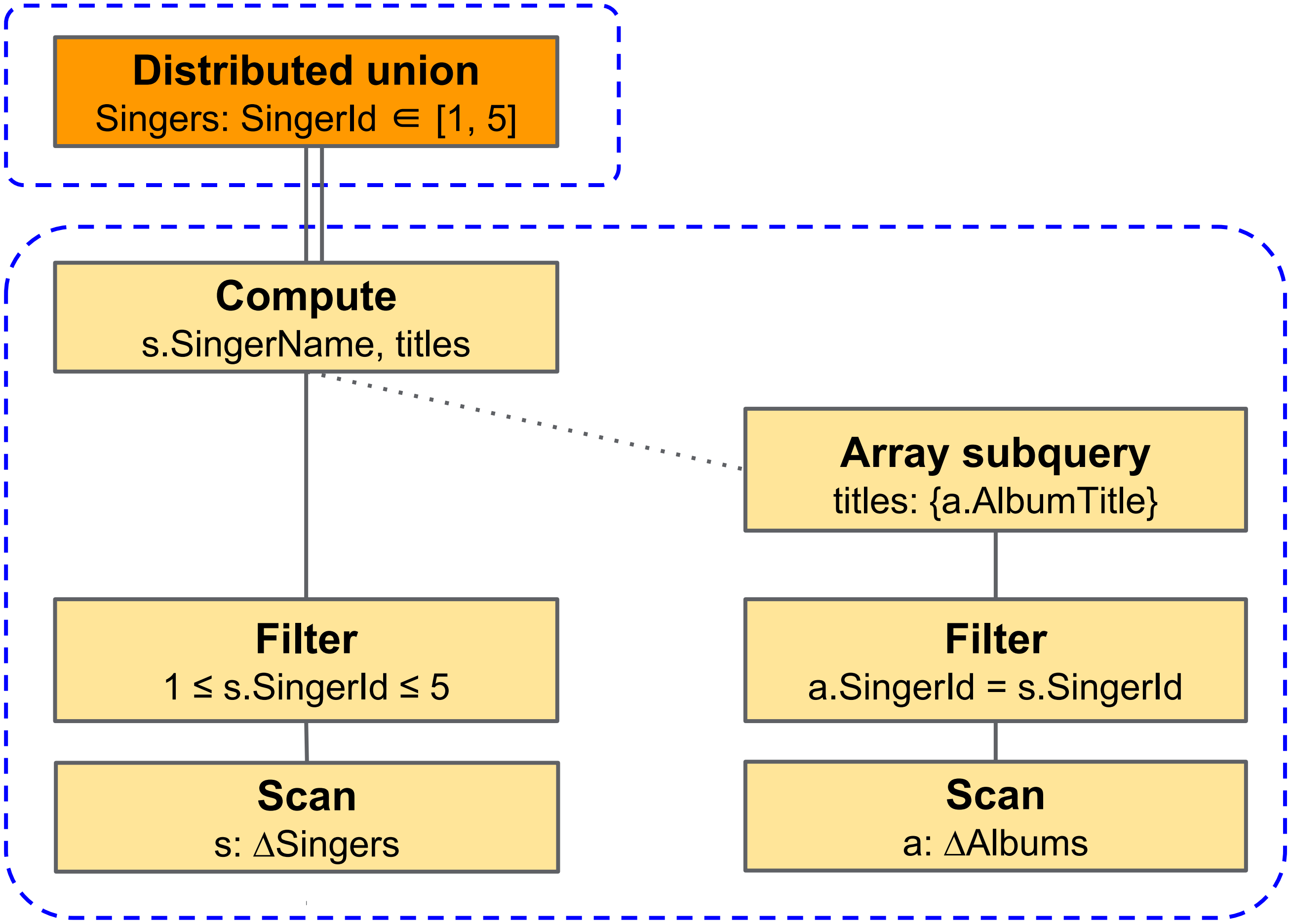
Distributed union operator



Push work to shards, extract distribution ranges



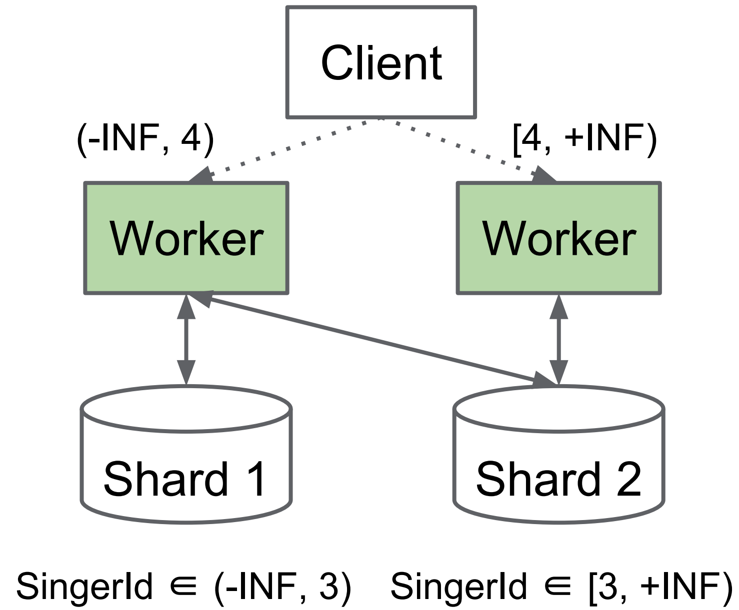
Exploiting co-location



Parallel-consumer API

```
SELECT SingerName, ARRAY(SELECT ...) titles
FROM Singers WHERE SingerId BETWEEN 1 AND 5
```

- Root-partitionable query:
  Q(Union of ∆T) = Union of Q(∆T)
- Same result up to order of rows
- Another main distribution operator: Distributed Cross Apply (in the paper)



Restartable snapshot queries

Google

Query restarts: overview

- Automatic compensation for failures
- For snapshot queries only
- Server yields "restart token" with each result batch
- Client can resume query execution after consuming partial results
- Contract: omit previously returned rows
  - No repeatability guarantee for subsequent rows

SingerName STRING	**titles** ARRAY<STRING>
Beatles	[Help!, Abbey Road]
U2	[ ]
Pink Floyd	[The Wall]

restart

Google

Query restarts: implementation challenges

- Naive solutions don't work well for "large" queries
  - Buffer final result, persist intermediate results, count rows, etc.
- Instead: efficiently capture distributed state of query execution
- Dynamic resharding
  - May restart on different row range
- Non-determinism
  - Memory size, parallelism, computer architecture, numerics, ...
- Restarts across server versions
  - Query plans, execution algorithms

Query restarts: hard but worth it

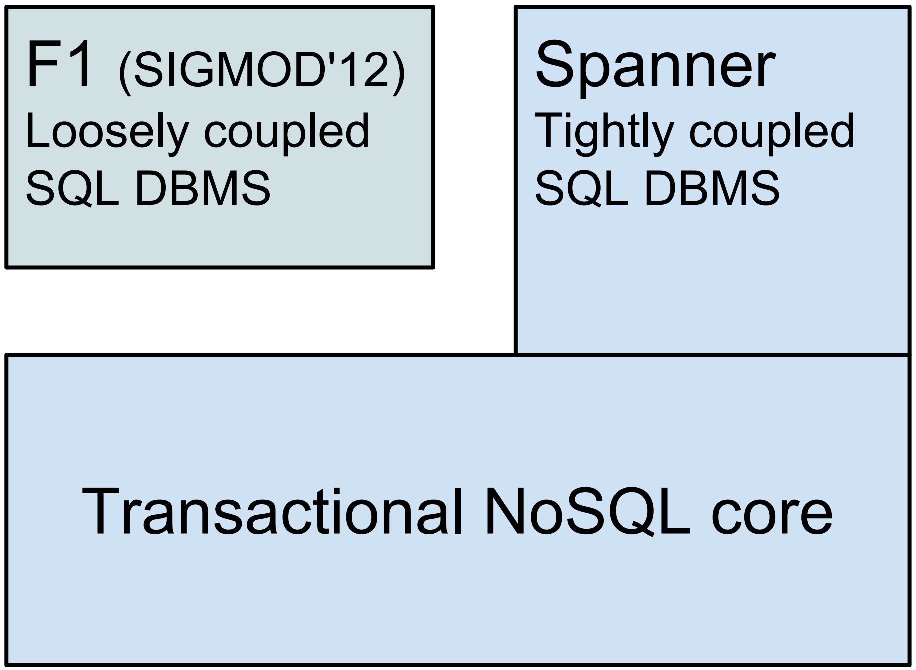
- Hide transient failures
- No retry loops: simpler programming model
- Streaming pagination
- Ensure forward progress for important class of long-running queries
- Improve tail latency of online requests
- Low-impact rolling server upgrades

Lessons learned

Rethinking DBMS architecture for scale

- Runs exclusively as a service
  - Huge economy of scale in centralizing & automating human load
  - Must never regress (query optimization is hard)
- Dynamic sharding
  - Essential for elasticity
- Requests to data ranges, not servers
  - Automatic replica selection
- Shard-level isolation
  - Shard can become unavailable without affecting workloads on other data in same table
- Restartable snapshot queries
  - Robustness & forward progress in presence of failures

System layering



- Relational model
- Schemas
- SQL
- Indexes

- Horizontal scalability
  - Web-scale systems
- Manageability
  - Transparent failover
  - Easy resharding
  - Control plane
- ACID transactions
  - Across arbitrary rows

Lessons learned

- Both loosely & tightly coupled SQL designs work well
  - Deployed simultaneously on same transactional NoSQL core
- Transactions are hugely helpful for system internals
  - Schema versioning, data movement/resharding, online index creation, backups, storage format changes, ...
- Relational model: better earlier than later
  - Well-known abstractions get developers on common page
  - Reduces cost of foreseeable future migration
- SQL vs. NoSQL dichotomy may no longer be relevant at Google

Questions?

http://cloud.google.com/spanner

Backup slides

Storage & Dist

Spanner: Becoming a SQL System

D. Bacon, N. Bales, N. Bruno, B. Cooper, A. Dickinson, A. Fikes, C. Fraser, A. Gubarev, M. Joshi, E. Kogan, A. Lloyd, Sergey Melnik, C. Taylor, R. Rao, D. Shue, M. van der Holst, D. Woodford (Google)

- Distributed transactional data management system
- Globally replicated, highly-available managed service
- Backs hundreds of mission-critical services at Google
  - AdWords, Google Play, Photos, etc.
  - 10s of millions QPS, 100s of petabytes, 5,000+ databases
- Publicly available on Google Cloud Platform: http://cloud.google.com/spanner
- This talk: making Spanner a SQL DBMS

Session 7: Storage and Distribution (2)
14:00-15:40 @ Continental C

Doh... Just implement the SQL standard

- NIST abandoned compliance testing in 1996
  - Before then, Fed Govt would only buy compliant DBMSes
- SQL:1999 specs onward are broad, imprecise
  - No implementation requirement (unlike W3C)
- Spec'ed features implemented differently by DBMSes
  - Many proprietary extensions

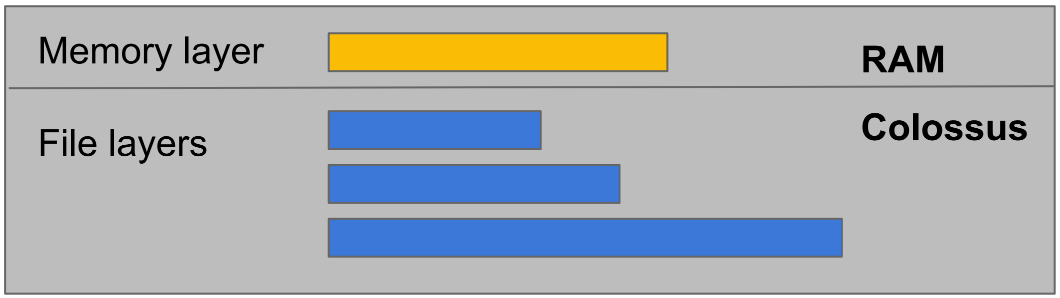
http://www.tdan.com/view-articles/4923/
Is SQL a real standard anymore?
by M. M. Gorman, ANSI/SQL committee secretary

Bottom line: substantial language design work

Blockwise-columnar storage

Persistent storage

- Log-structured merge tree:



- Original layer format: SSTables (from Bigtable)
  - Optimized for schema-less key/value pairs
- Improved format: Ressi (mid 2017)
  - Essentially, PAX layout (Ailamaki et al 2002)
  - For schematized data & hybrid OLTP/OLAP workloads

SSTables vs. Ressi

```
CREATE TABLE Singers (
   SingerId INT64 NOT NULL,
   SingerName STRING(MAX),
   URL STRING(MAX)
) PRIMARY KEY(SingerId);
```

Key	T	Value
1		Beatles
1		beatles.com
1		thebeatles.com
2		U2
2		u2.com
3		Pink Floyd
3		pinkfloyd.com

SingerId
1
2
3

URL
thebeatles.com
u2.com
pinkfloyd.com

SingerName
Beatles
U2
Pink Floyd

- Columnar within blocks
- Old versions & large values stored separately:

URL
beatles.com

Challenges

- Query optimization: never regress
- Transaction: read uncommitted results in active Tx
  - Essential for SQL DML
- Physical design
  - Wrong choices can kill performance
- Versatility
  - OLTP, OLAP, full-text, JSON, etc.