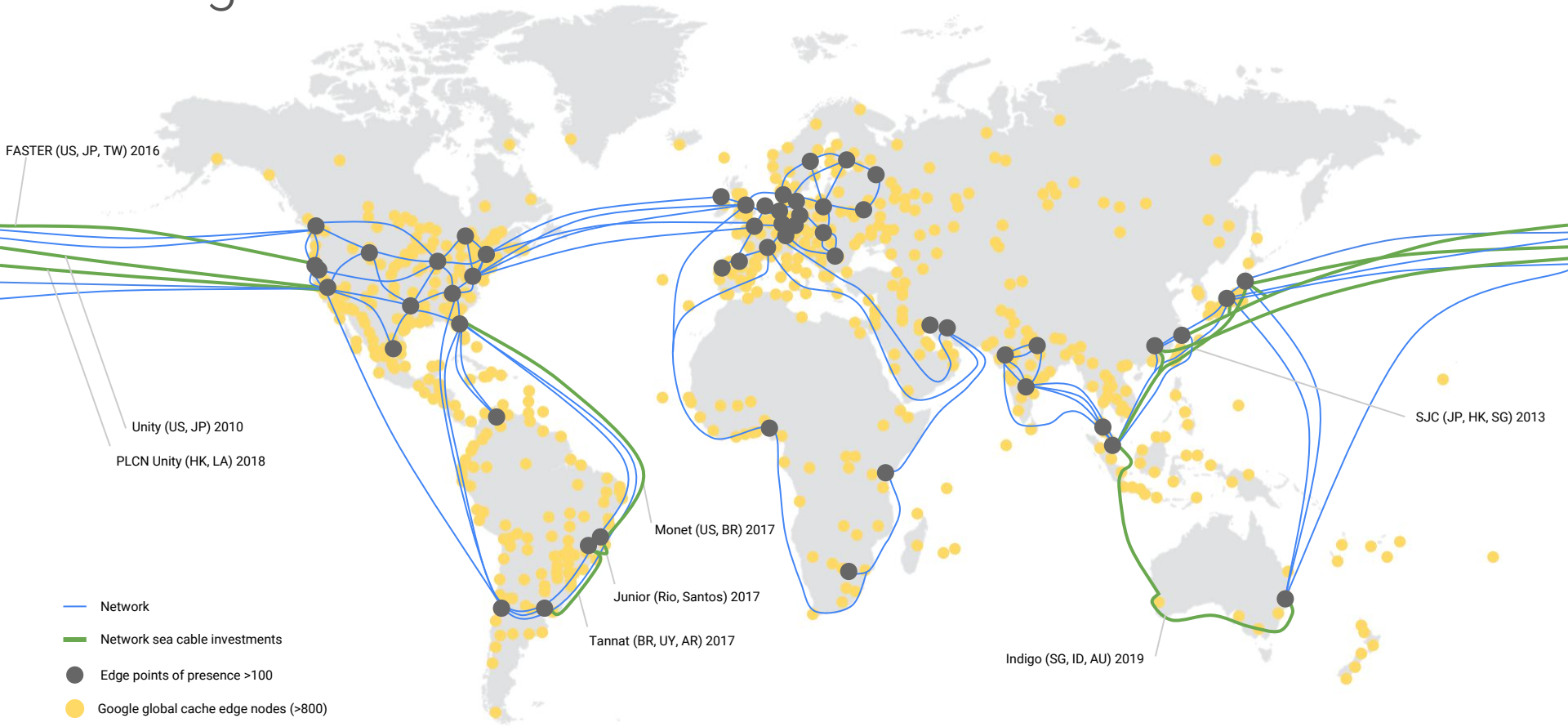
Cloud Spanner

Joe Intrakamhang
Solutions Engineer
@joe_int

$29.4 Billion

3 Year Trailing CAPEX Investment

Google Network



Background

Why build Spanner?

Technical infrastructure at Google

It’s 2005...

Google’s needs

- Horizontally Scaling Database
- ACID Transactions with global consistency
- No downtime!

Existing options

Nonrelational

Relational

Overview

What is Cloud Spanner?

What is Cloud Spanner?

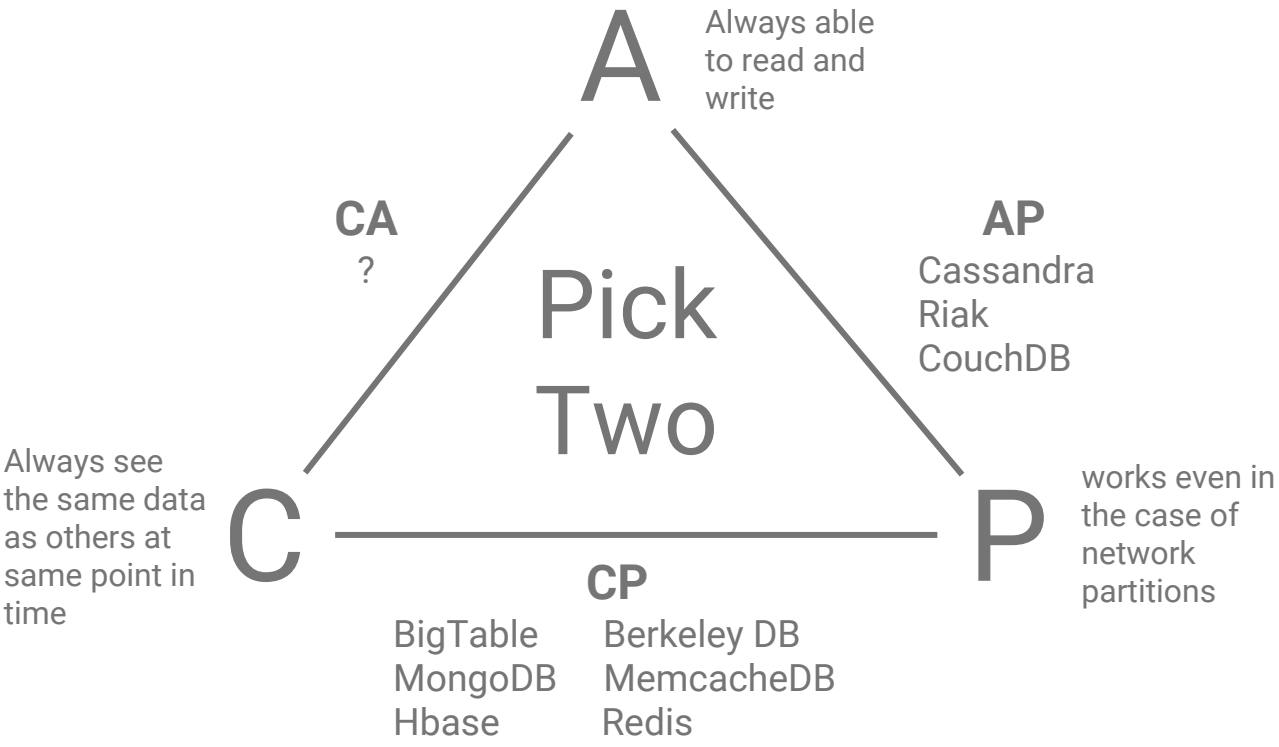
Google’s mission-critical scalable relational Database Service

- Fully managed, database service with global scale
- Traditional relational semantics: schemas, ACID transactions, SQL
- Automatic, synchronous replication within and across regions for availability
- Battle-tested within Google for 5+ yrs (AdWords, GooglePlay)

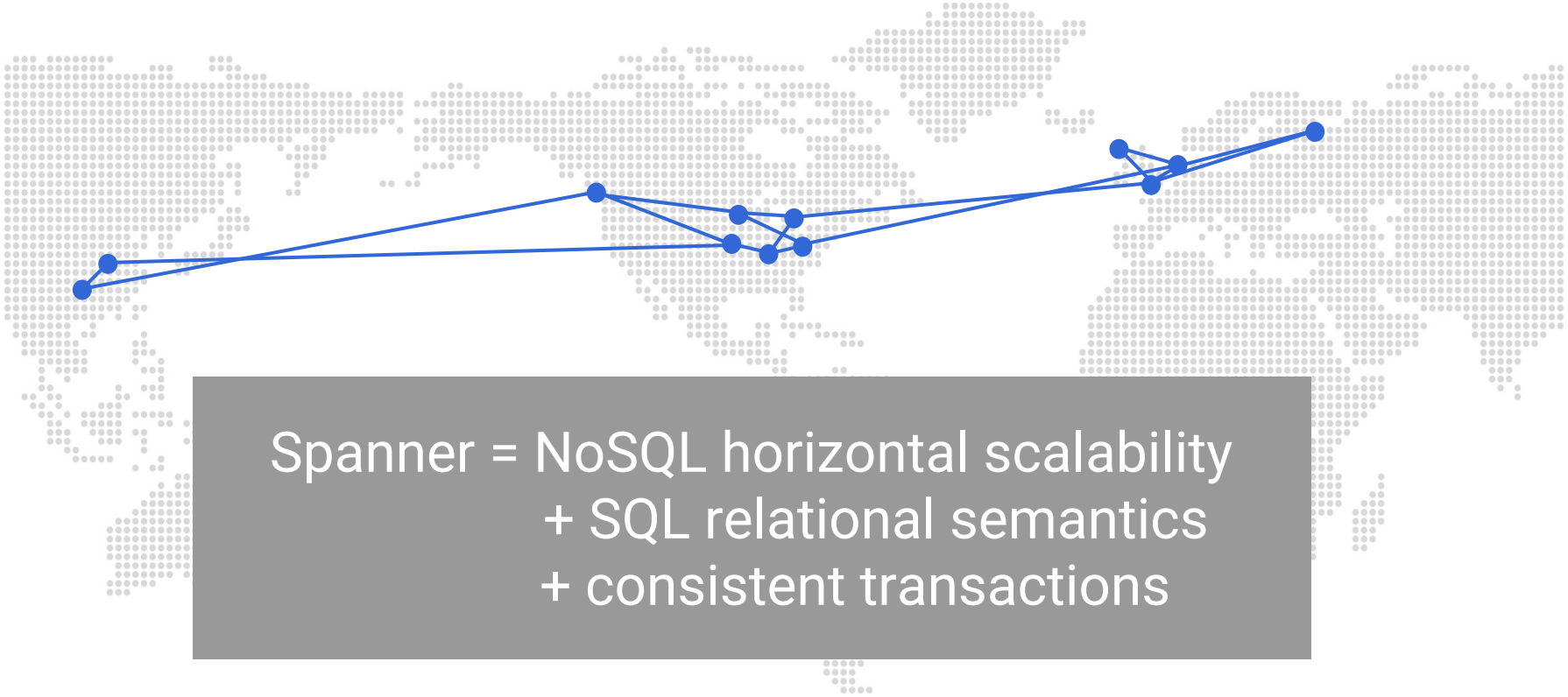
How does it compare?

	CLOUD SPANNER	TRADITIONAL RELATIONAL	TRADITIONAL NON-RELATIONAL
Schema	✓ **Yes**	✓ Yes	✗ No
SQL	✓ **Yes**	✓ Yes	✗ No
Consistency	✓ **Strong**	✓ Strong	✗ Eventual
Availability	✓ **High**	✗ Failover	✓ High
Scalability	✓ **Horizontal**	✗ Vertical	✓ Horizontal
Replication	✓ **Automatic**	↻ Configurable	↻ Configurable

What alternatives are there?



Spanner tries to be both, SQL and distributed, how?



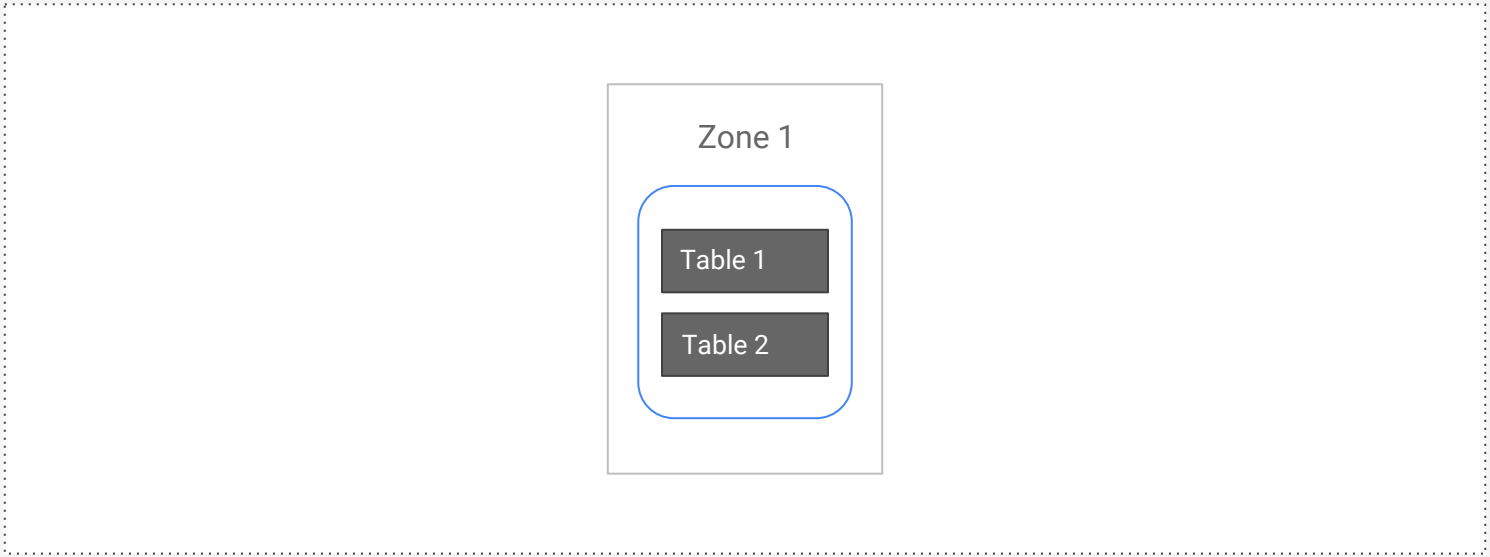
Spanner = NoSQL horizontal scalability
+ SQL relational semantics
+ consistent transactions

Open standards

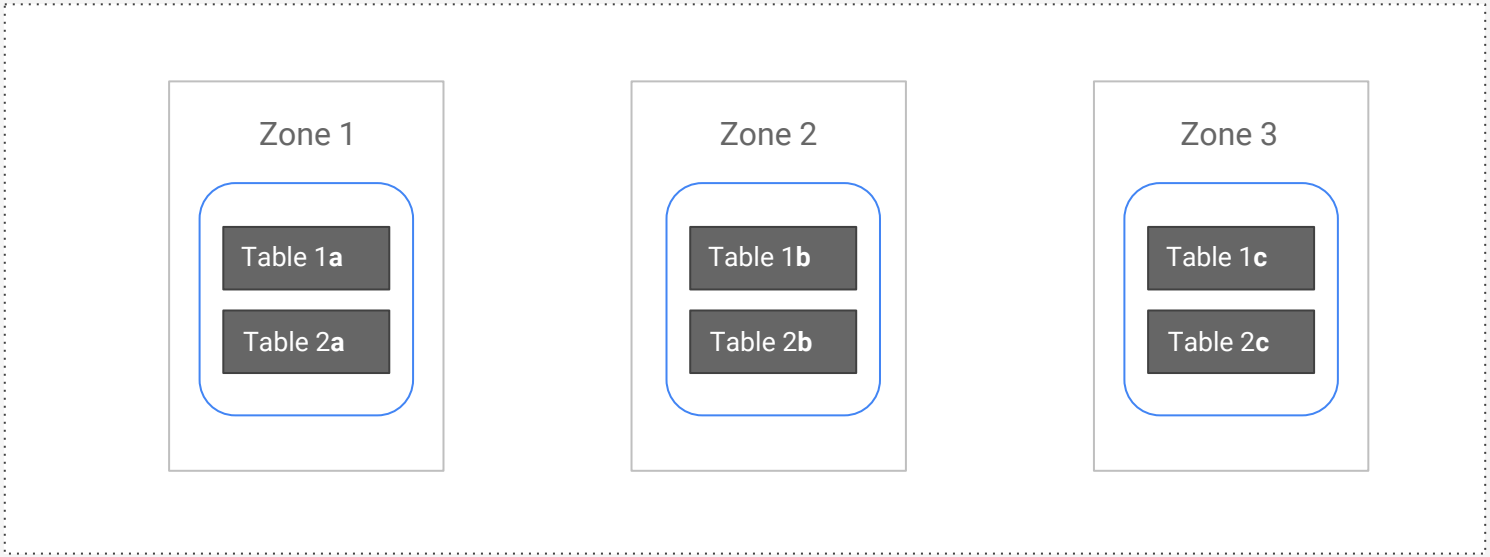
- Standard SQL (ANSI 2011)
- Encryption, Audit logging, Identity and Access Management
- Client libraries in popular languages (Java, C#, Python, Go, Node.js, etc.)
- JDBC driver

How Spanner works

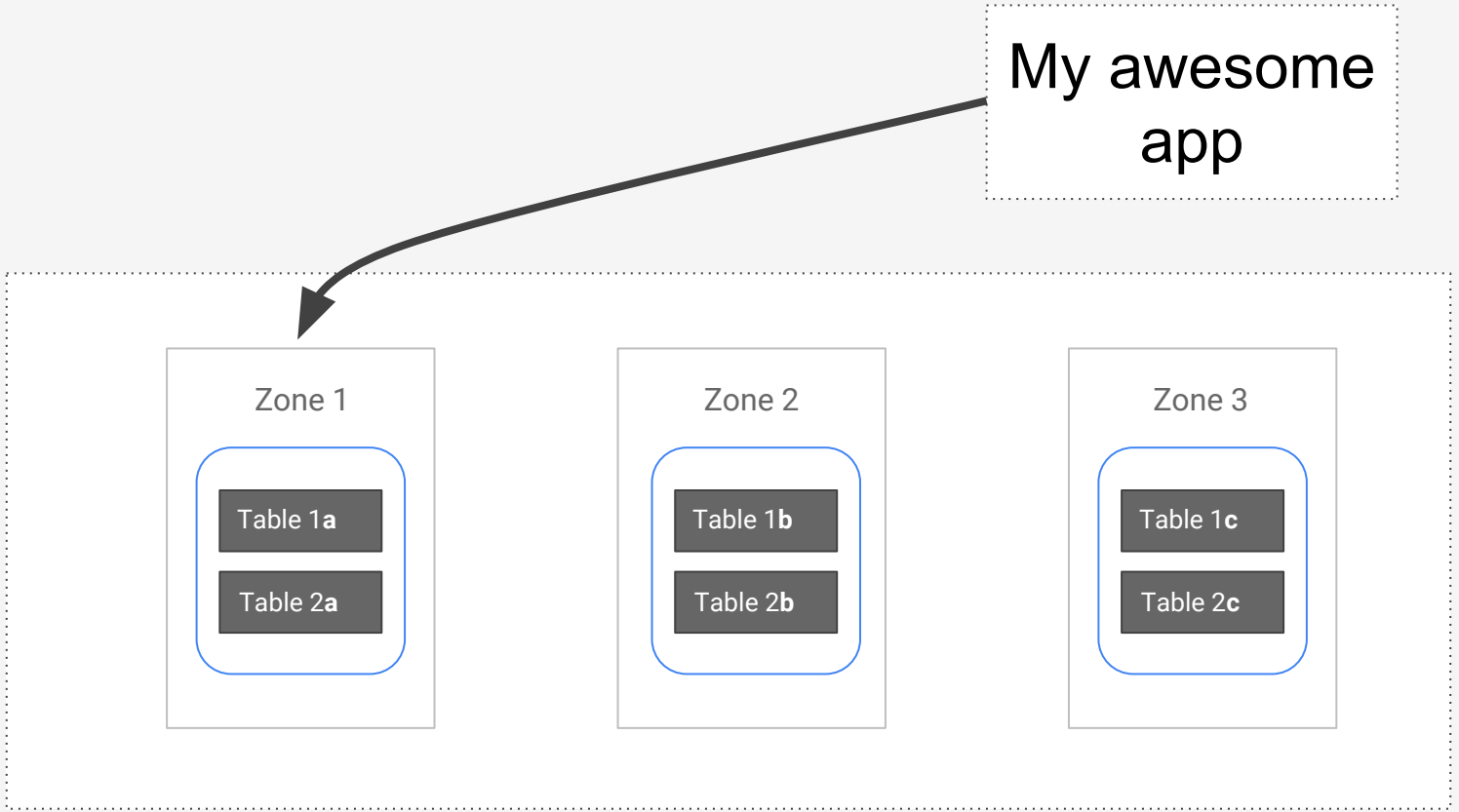
Traditional Database layout



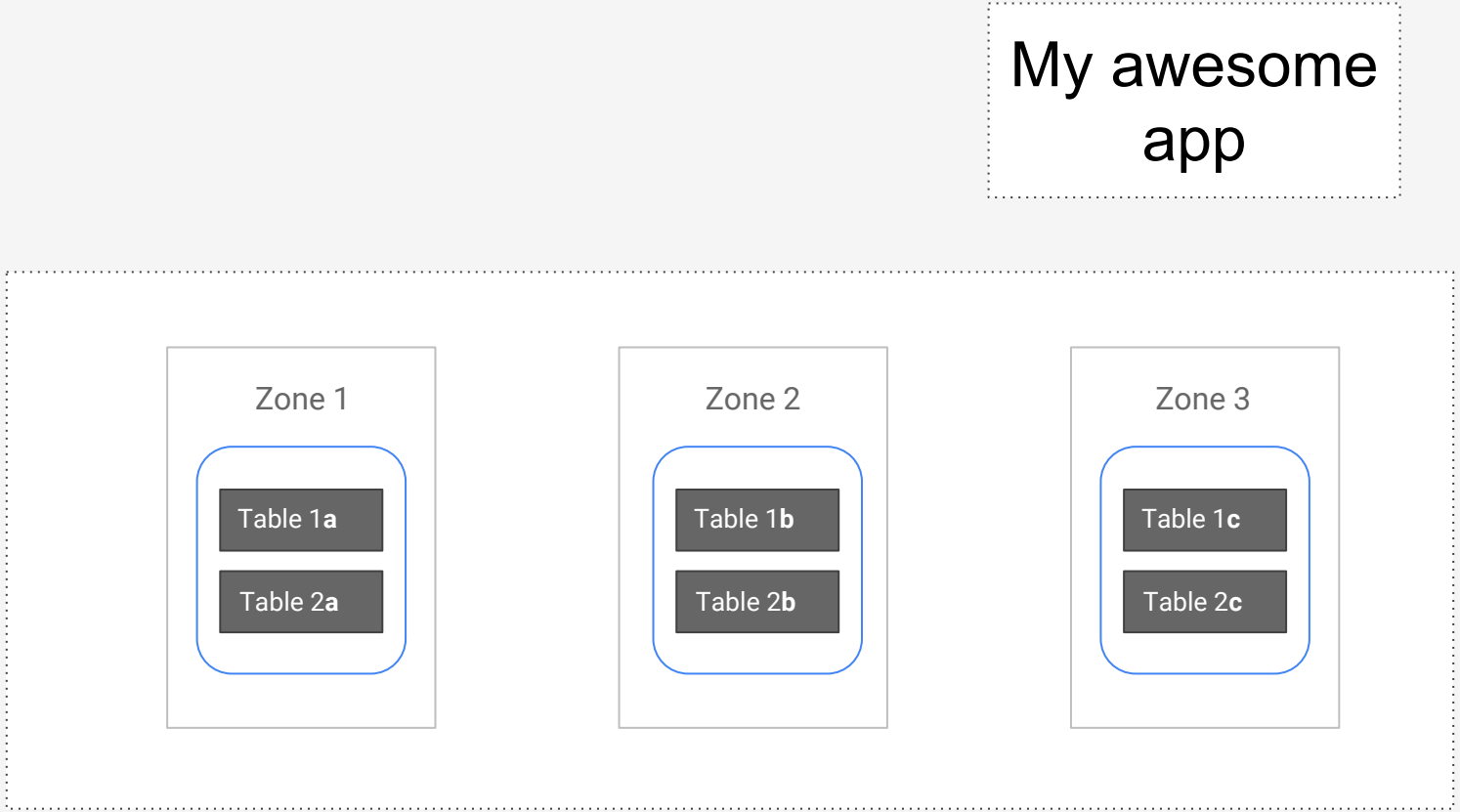
Traditional Database layout -- sharded



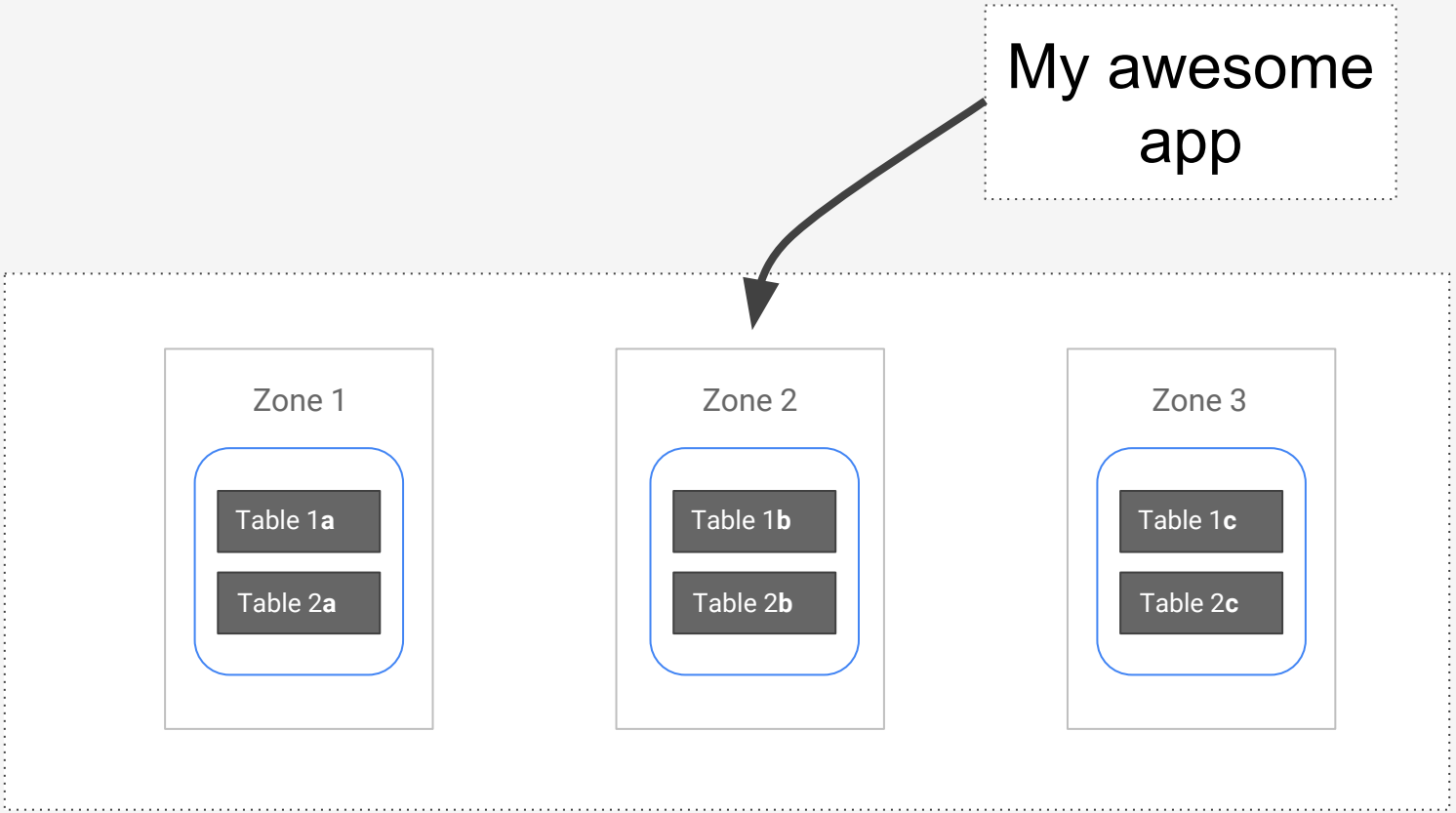
Traditional Database layout



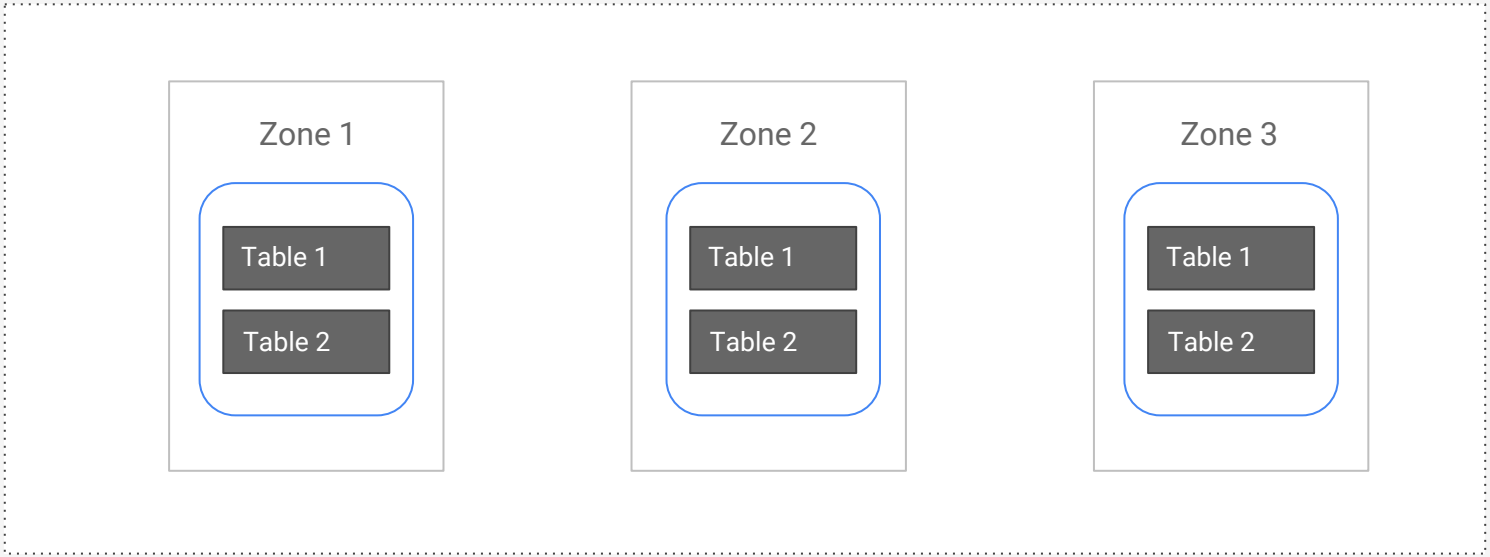
Traditional Database layout



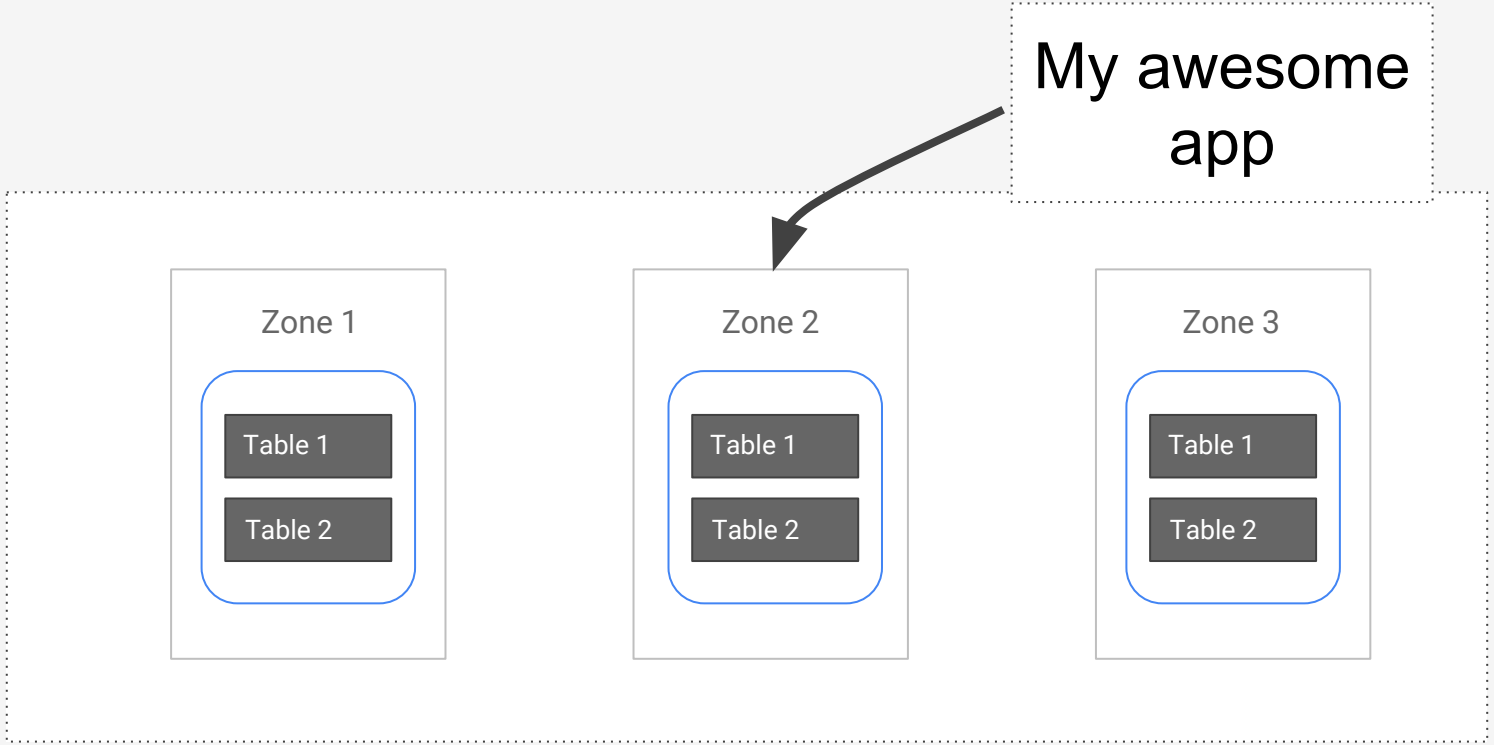
Traditional Database layout



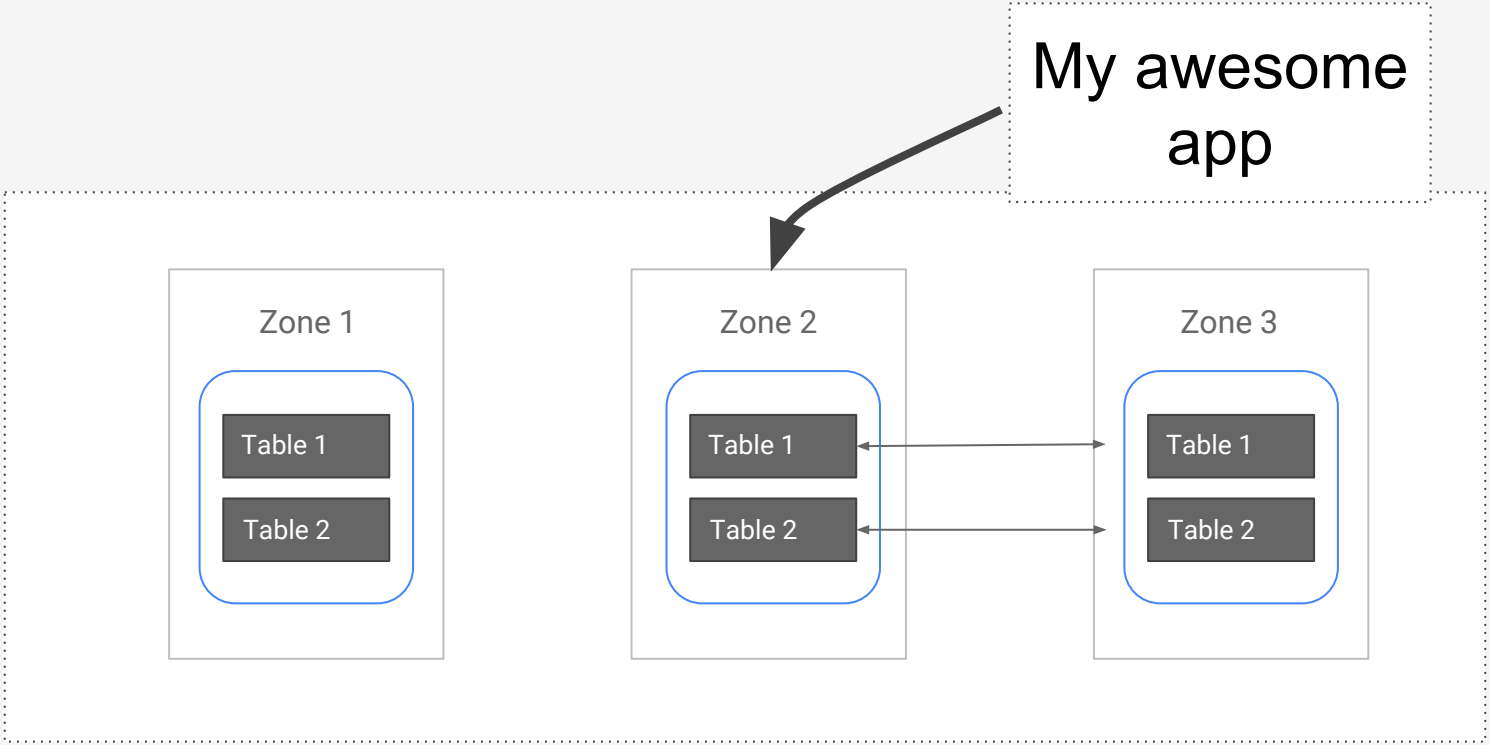
Spanner Database layout



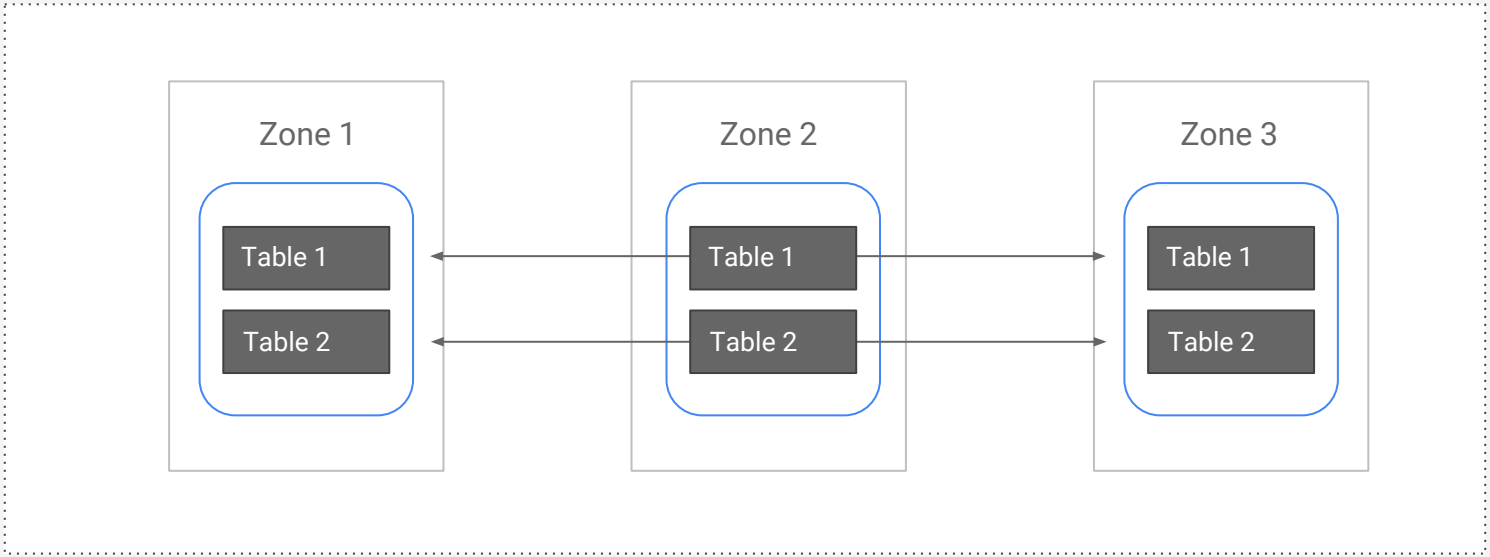
Transactions



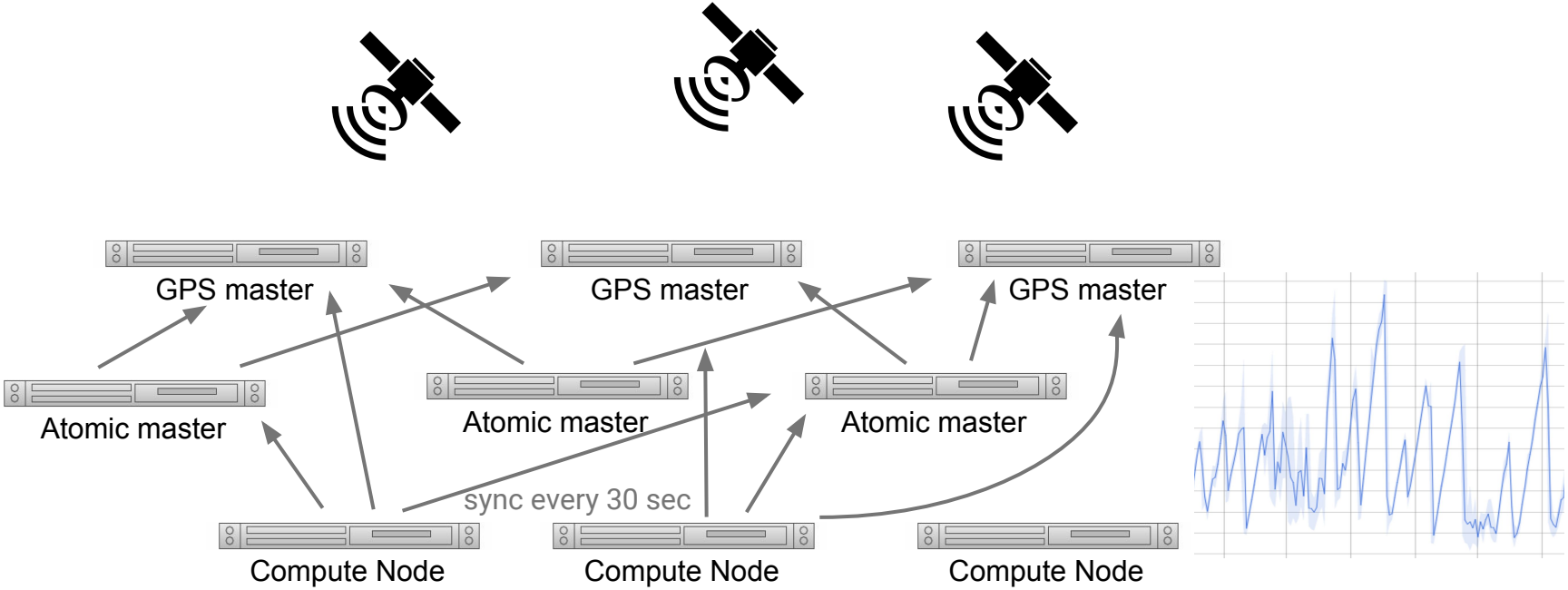
Transactions



Transactions

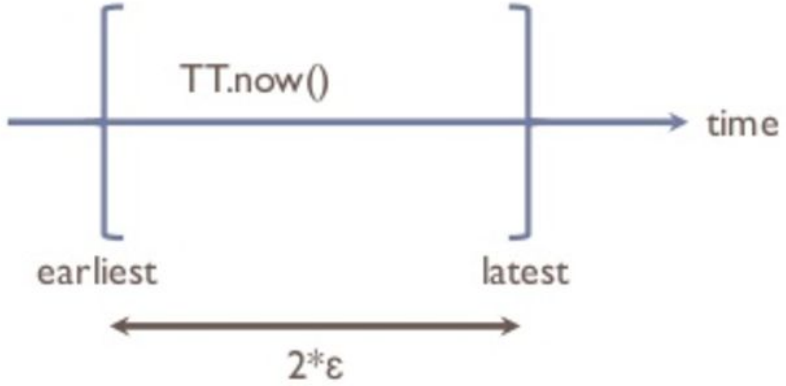


TrueTime Infrastructure

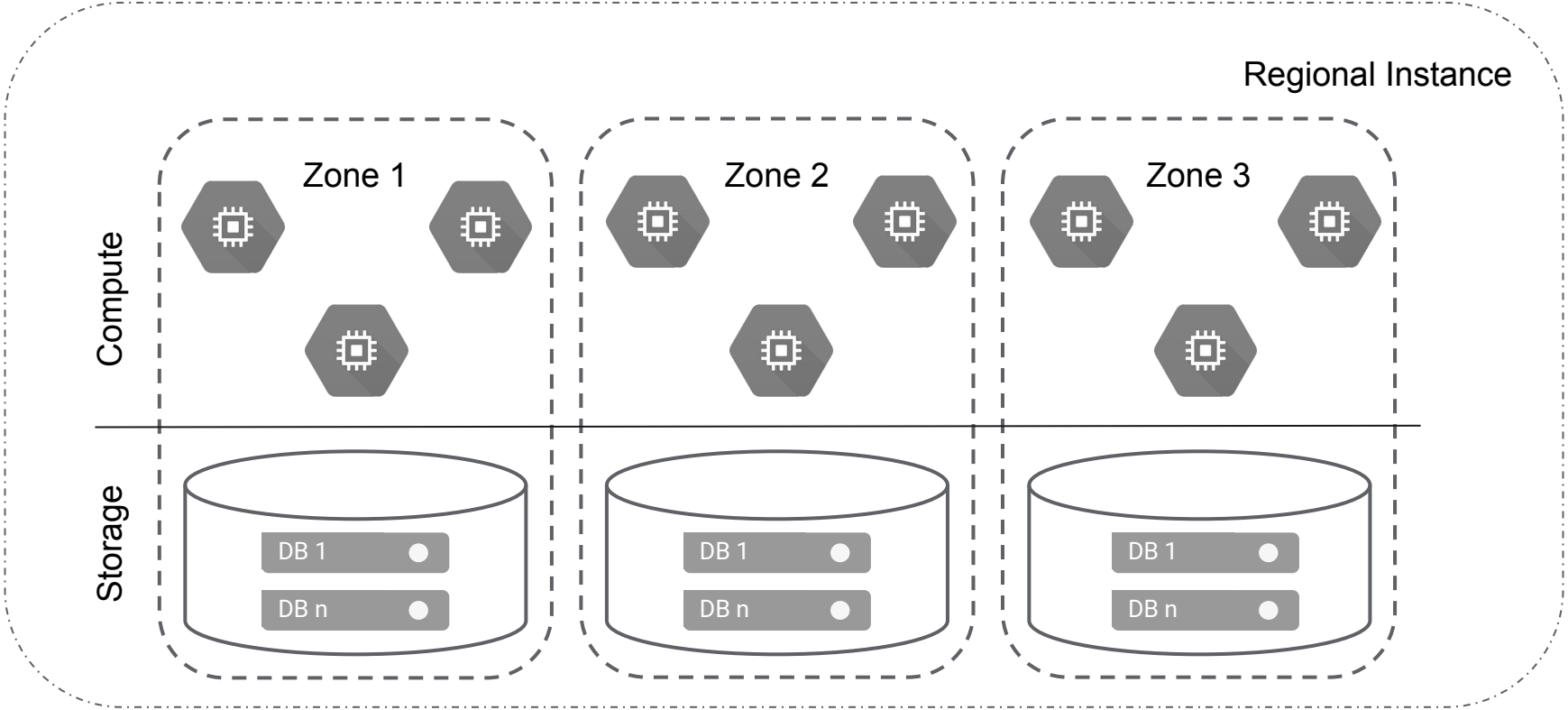


TrueTime - virtual global clock

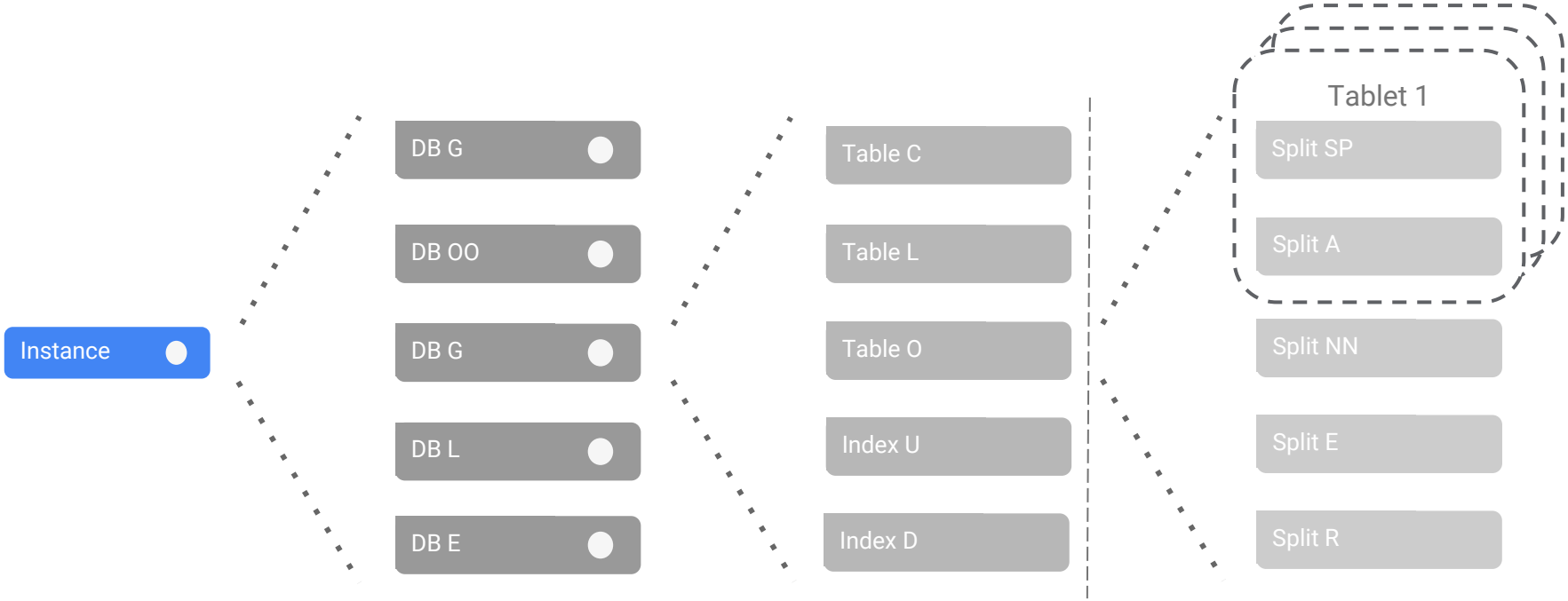
- Quantifies the “worst” possible error / drift between clocks in all datacenters around the world (global clock)
- TrueTime.Now() gives you an interval [*t*1,*t*2]; *t*2 = *t*1 + 2ε



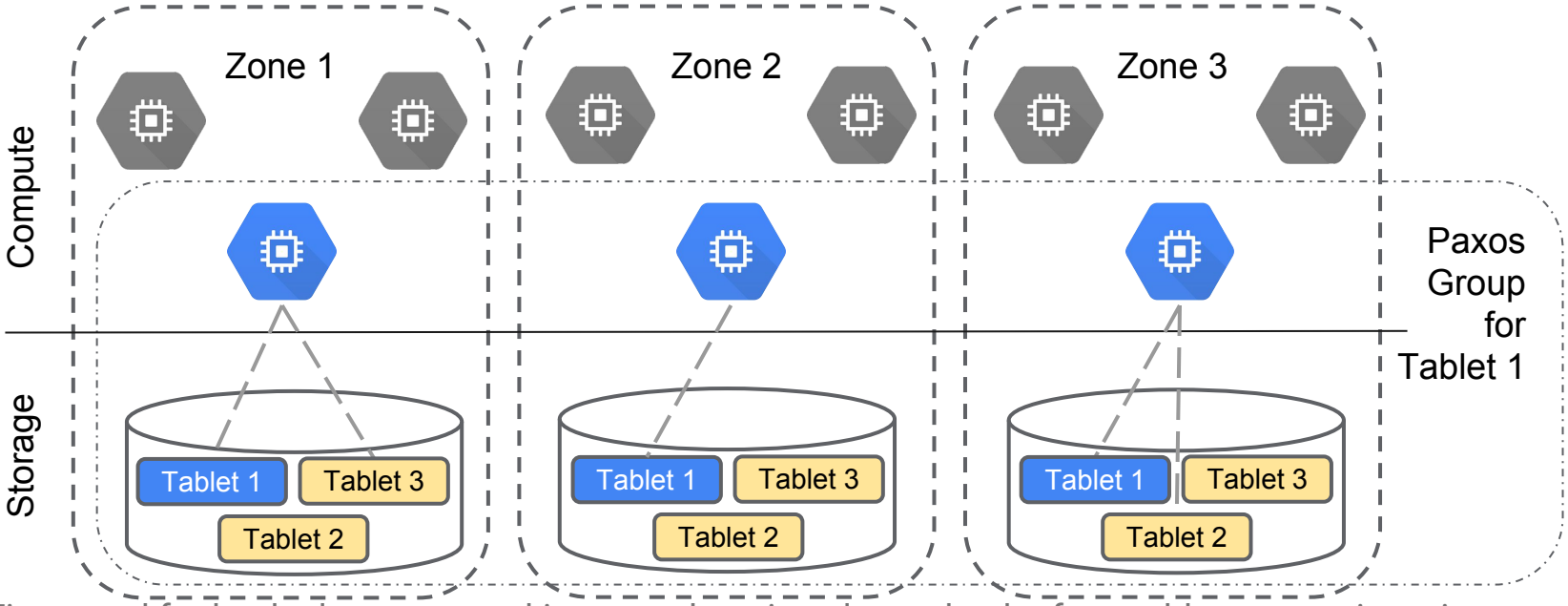
Spanner Internals



Spanner Internals



Spanner Internals

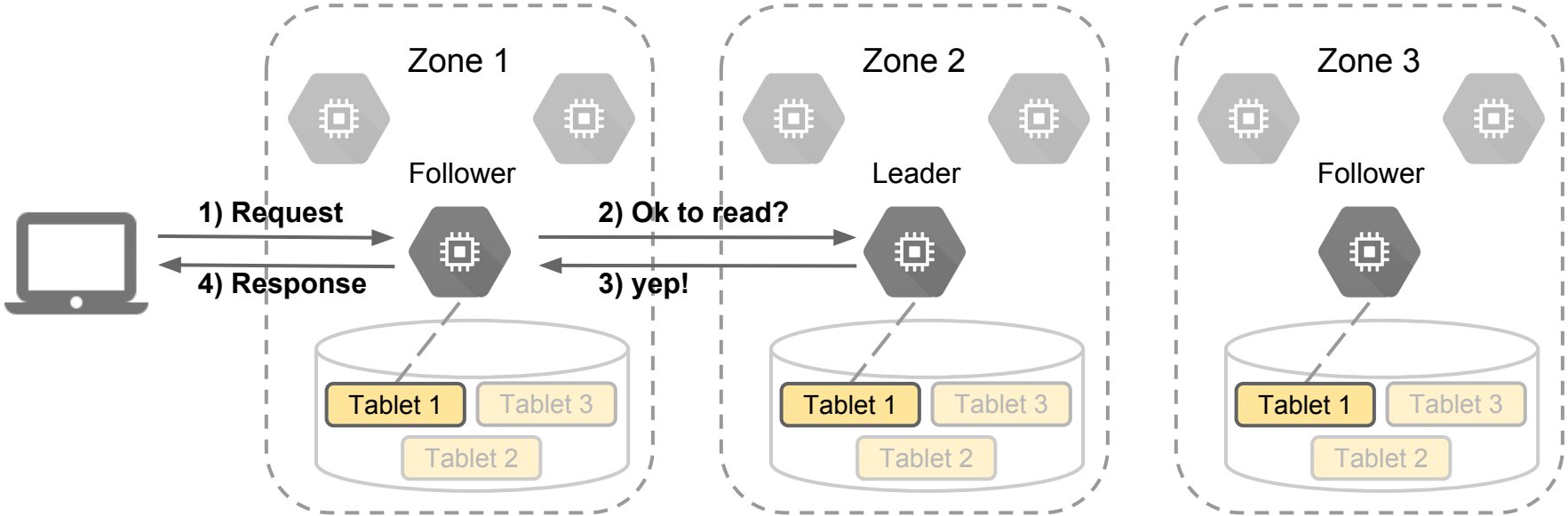


* TrueTime used for leader leases → making sure there is only one leader for a tablet at any given time

Life of a Query

Consistent Read

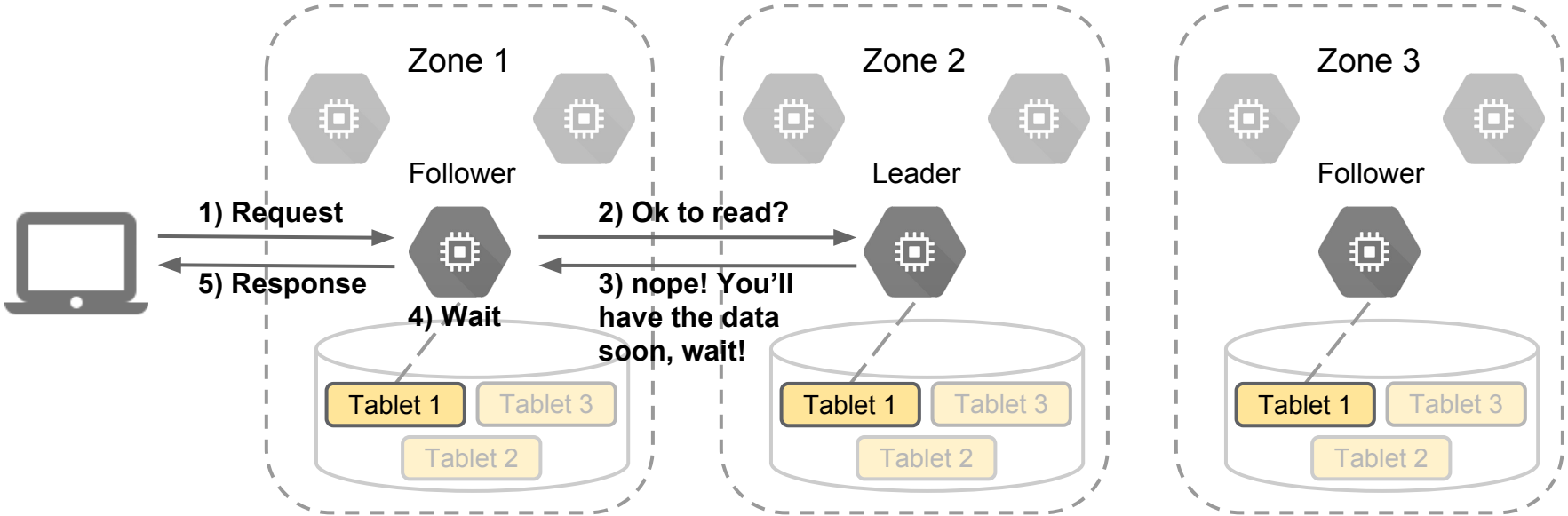
```
SELECT * FROM Company WHERE Name = 'Google';
```



Life of a Query

Consistent Read

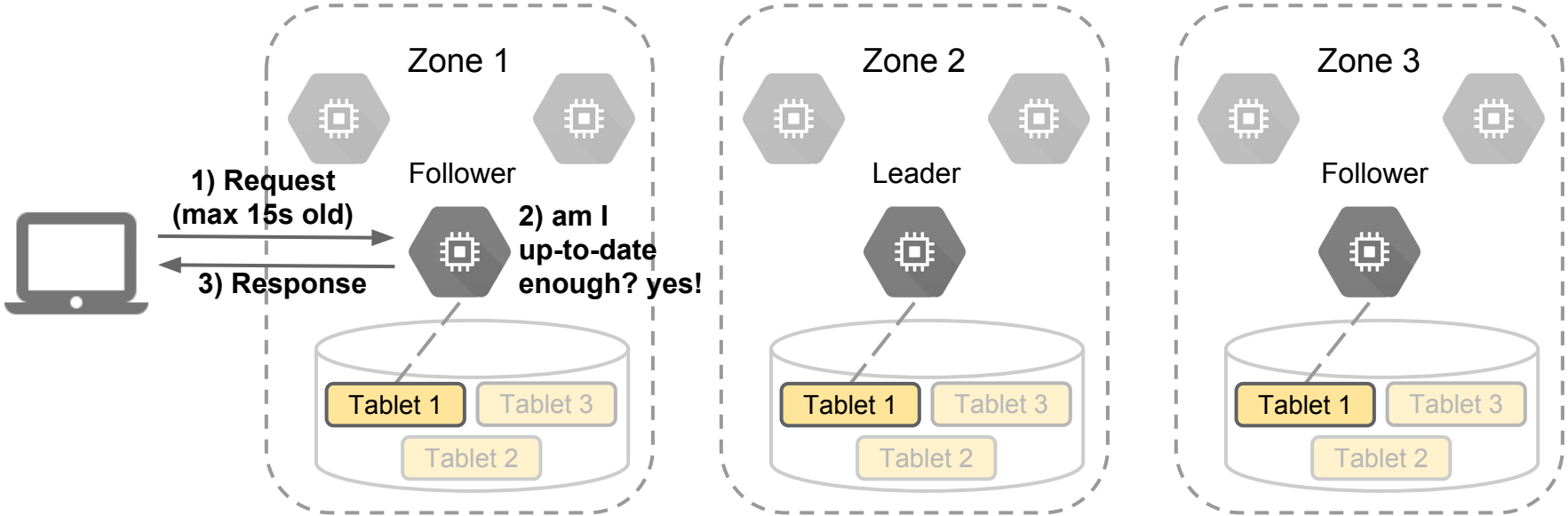
```
SELECT * FROM Company WHERE Name = 'Google';
```



Life of a Query

Time-bounded (stale) reads

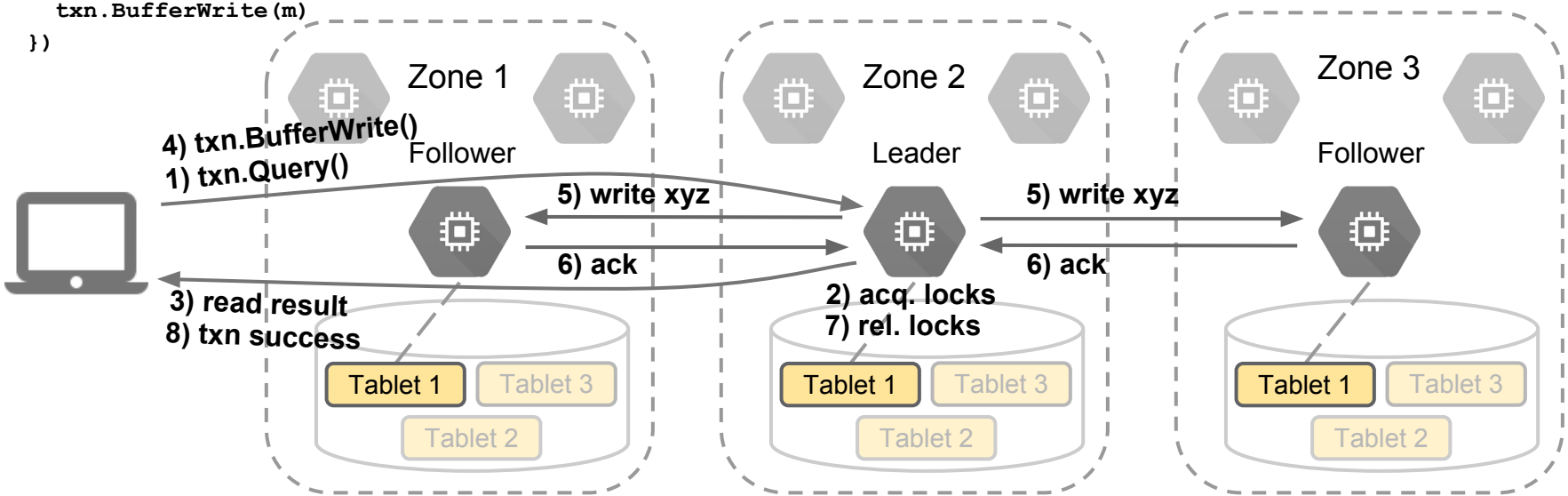
```
SELECT * FROM Company WHERE Name = 'Google';
```



Life of a Query

Read-Write Transaction

```
dbClient.ReadWriteTransaction(ctx, func(txn *spanner.ReadWriteTransaction) error {
  txn.Query(ctx, stmt).Do(func(r *spanner.Row) error {...})
  m = append(m, spanner.InsertOrUpdate("TableName", []string{"ColA", "ColB"},
                       []interface{}{"93bc3d", "Lorem Ipsum"}))
  txn.BufferWrite(m)
})
```



Relational Data Layout

SingerId	SingerName
1	Beatles
2	U2
3	Pink Floyd

SingerId	AlbumId	AlbumName
1	1	Help!
1	2	Abbey Road
3	1	The Wall

Interleave Data Layout

1	Beatles	
1	1	Help!
1	2	Abbey Road
2	U2	
3	Pink Floyd	
3	1	The Wall

Relational 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,
  AlbumName STRING(MAX),
) PRIMARY KEY(SingerId, AlbumId)
INTERLEAVE IN Singers;
```

No Downtime
Schema Migrations

```
ALTER TABLE Singers
ADD COLUMN Age INT64;
```

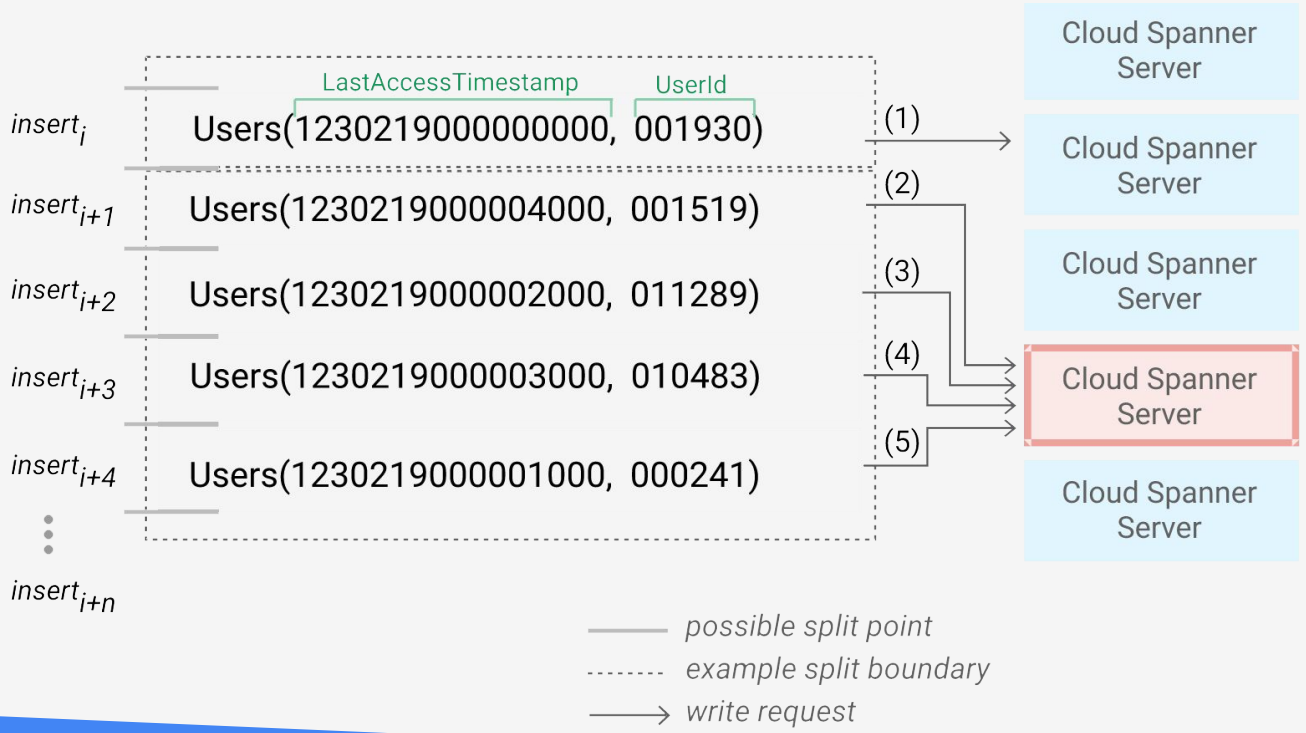
Schema Migration

```
CREATE TABLE `terms` (
  `id` bigint(19) unsigned NOT NULL AUTO_INCREMENT,
  `set_id` bigint(19) unsigned NOT NULL,
  `term` text NOT NULL,
  `definition` text NOT NULL,
  `photo` varchar(255) NOT NULL,
  `term_crc` int(11) unsigned NOT NULL,
  `rank` int(11) unsigned NOT NULL,
  `last_modified` int(11) unsigned NOT NULL DEFAULT '0',
  `is_deleted` tinyint(3) unsigned NOT NULL DEFAULT '0',
  `term_custom_audio_id` int(11) unsigned DEFAULT NULL,
  `definition_custom_audio_id` int(11) unsigned DEFAULT NULL,
  PRIMARY KEY (`id`,`set_id`),
  KEY `set_id_is_deleted` (`set_id`,`is_deleted`)
) ENGINE=InnoDB AUTO_INCREMENT=0 DEFAULT CHARSET=utf8mb4 ROW_FORMAT=COMPRESSED
/*!50100 PARTITION BY HASH (set_id) PARTITIONS 25 */
```

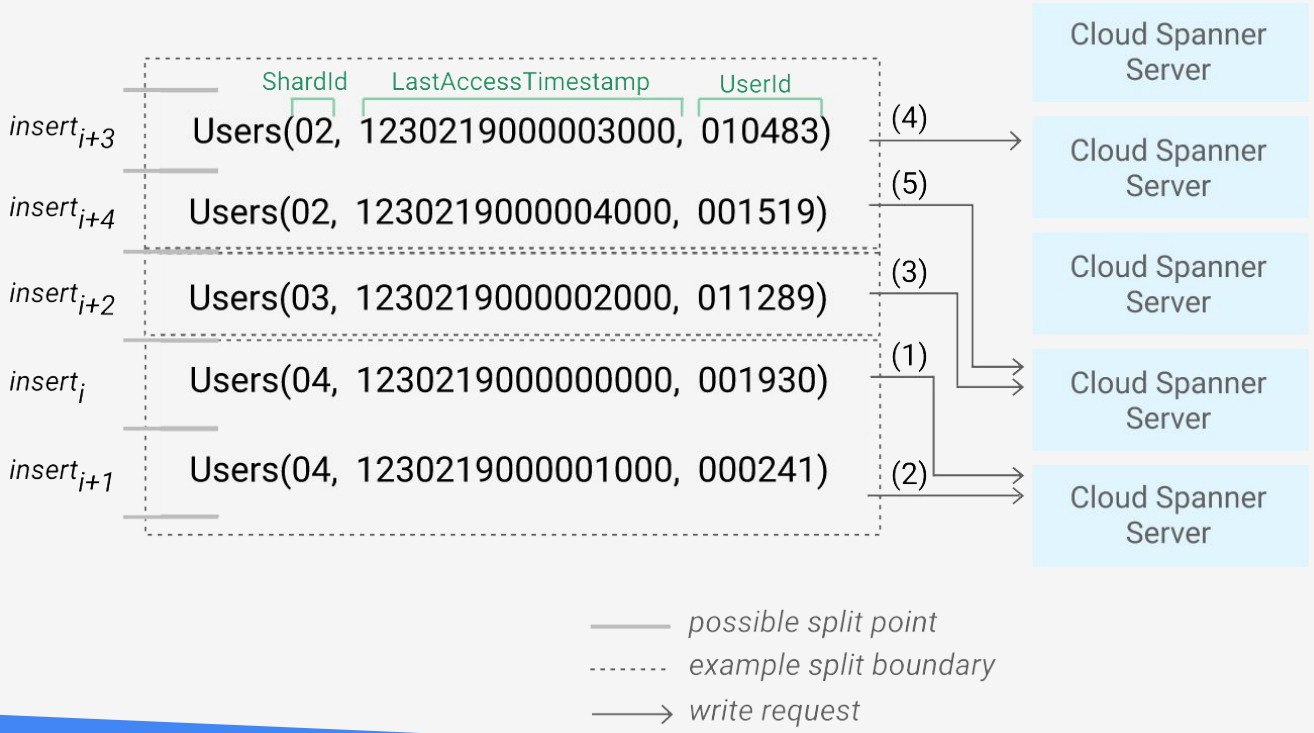
```
CREATE TABLE terms (
    id INT64 NOT NULL,
    set_id INT64 NOT NULL,
    spanner_id STRING(36) NOT NULL,      -- v4 uuid
    spanner_set_id STRING(36) NOT NULL, -- v4 uuid
    term STRING(MAX) NOT NULL,
    definition STRING(MAX) NOT NULL,
    image STRING(1024) NOT NULL,
    term_crc INT64 NOT NULL,
    rank INT64 NOT NULL,
    last_modified TIMESTAMP NOT NULL,  -- defaults to 0
    is_deleted INT64 NOT NULL,         -- defaults to 0
    term_custom_audio_id INT64,
    definition_custom_audio_id INT64
) PRIMARY KEY(spanner_set_id, spanner_id);

CREATE INDEX terms_id on terms(id);
```

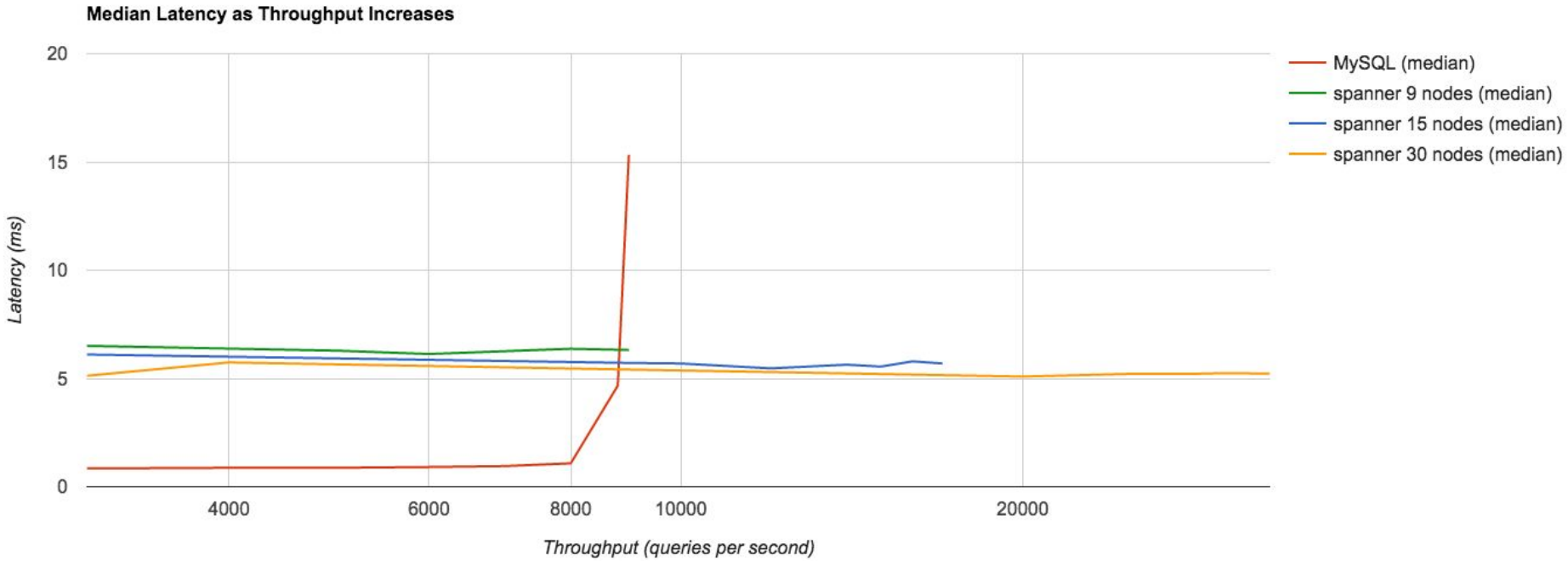
Anti-Pattern: Timestamp Ordering



Anti-Pattern: Sequences



Median Latency



Demo

Thank You

@joe_int