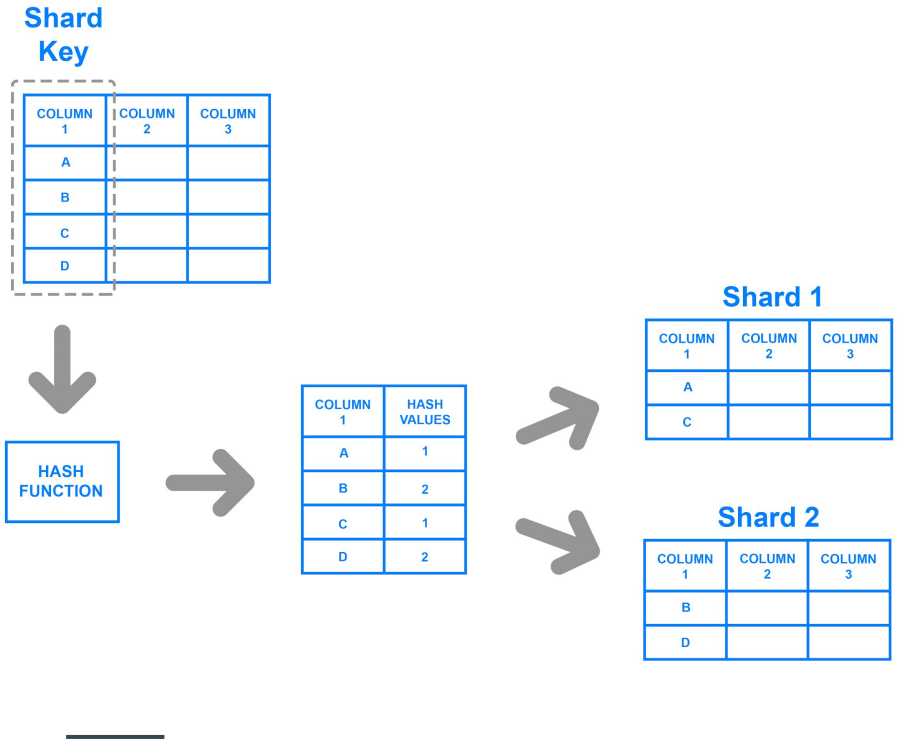

MONTANA
STATE UNIVERSITY

Sharding

Scaling Systems Horizontally

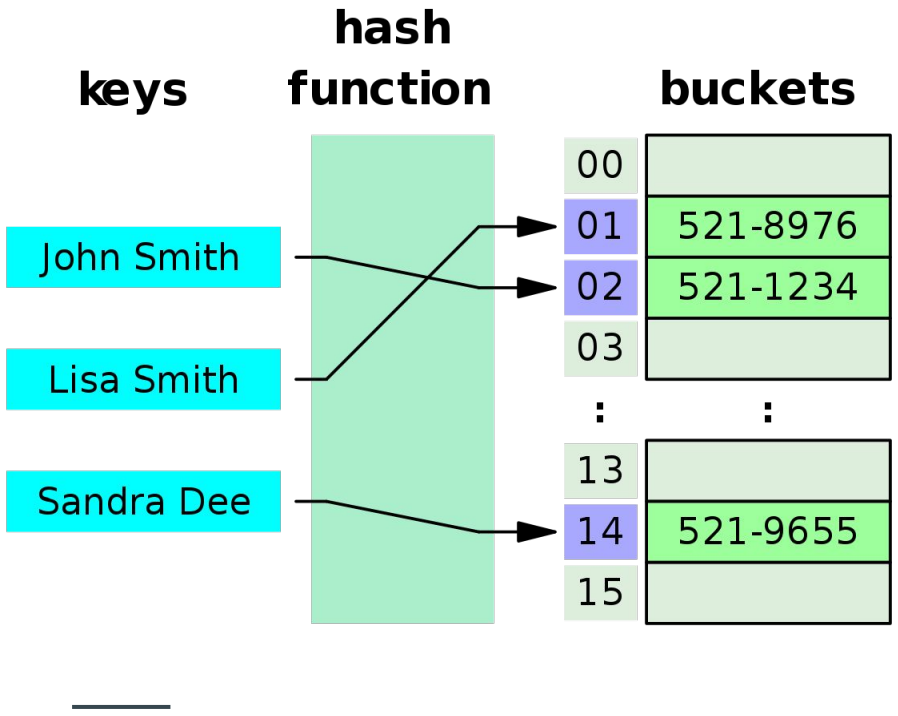
Sharding

- Sharding is a technology related to, but not identical with, clustering
- Closely related to the concept of hashing/hash tables



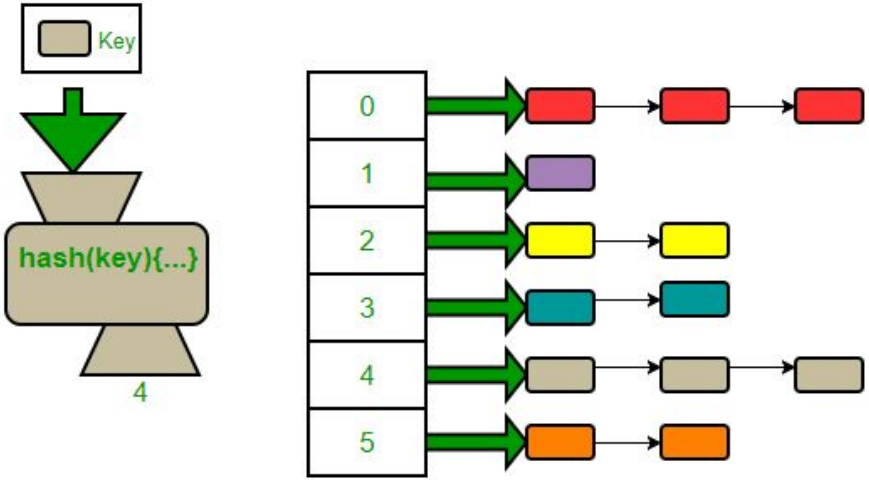
Hash Table Review

- Recall how a hashtable works:
  - A *key* is chosen for the table
  - A *hash function* is used to compute a number
    - The hash function should be as random as possible given the input
  - The number returned from the hash function is modded by the number of buckets
  - The *value* is stored in that bucket



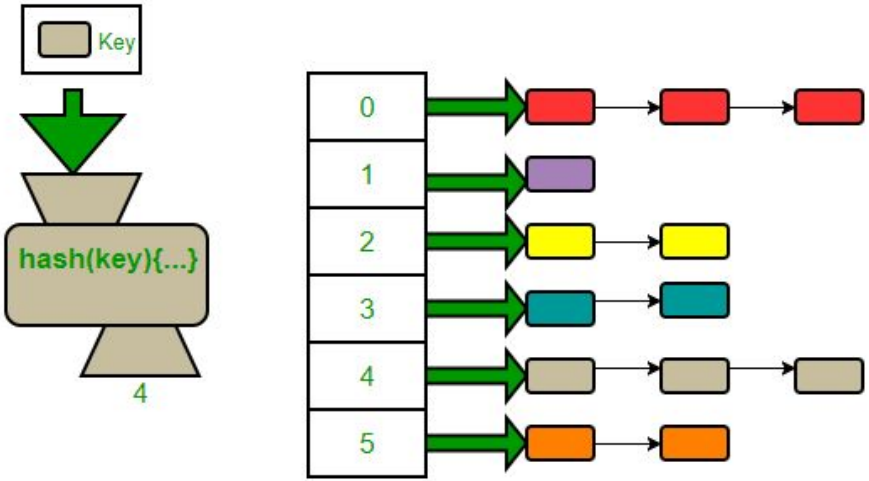
Hash Table Review

- Note that more than one element can live in a particular bucket
- There is *not* a 1-1 mapping between hash function results and items



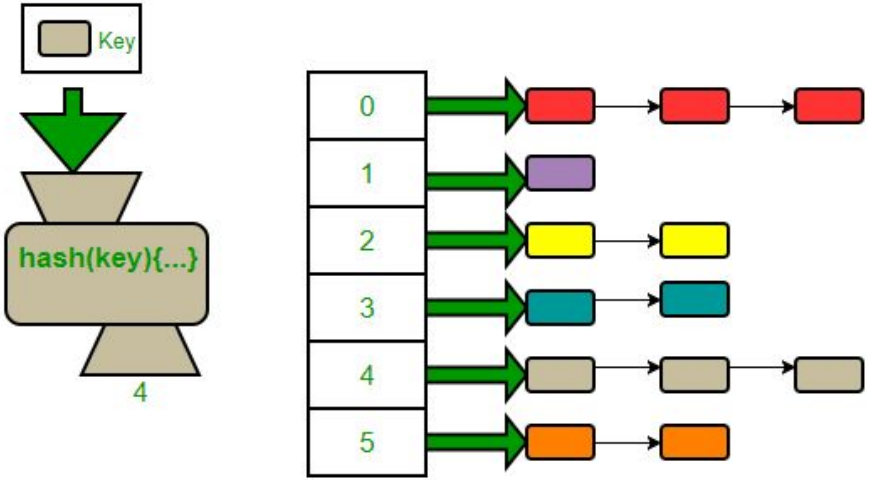
Hash Table Review

- This means that when you look something up in a hash table it is not just a simple bucket lookup
- Instead, you look up the appropriate bucket and do a list scan for the key



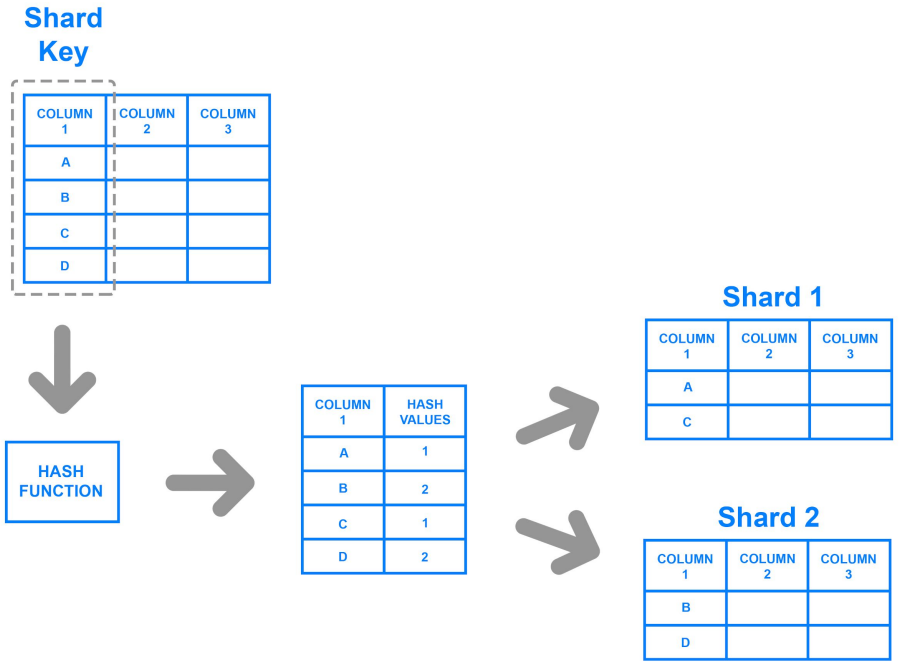
Hash Table Review

- What if a bucket has too many entries in it?
- This may be due to
  - A bad hash function
  - Bad luck
  - Too many entries in the hash table
- If the number of entries for all buckets is too high, it might be time to expand the number of buckets and *rehash* items



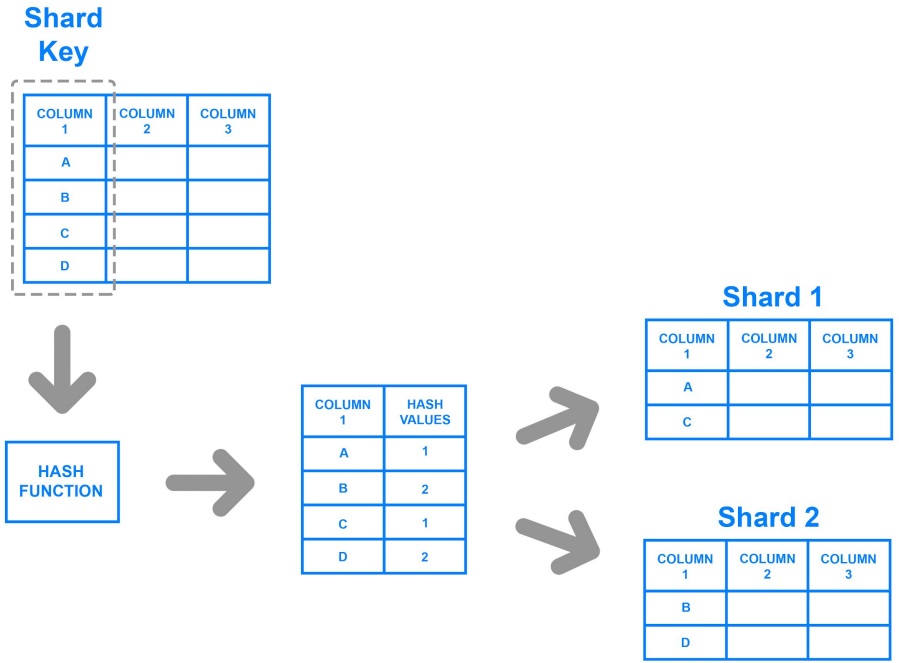
Sharding

- To return to sharding, here we have a simple table with a column, *Column 1* that we are going to *shard* on
- We compute the hash value of Column 1 and then select the *shard* based on that value modded by the number of shards



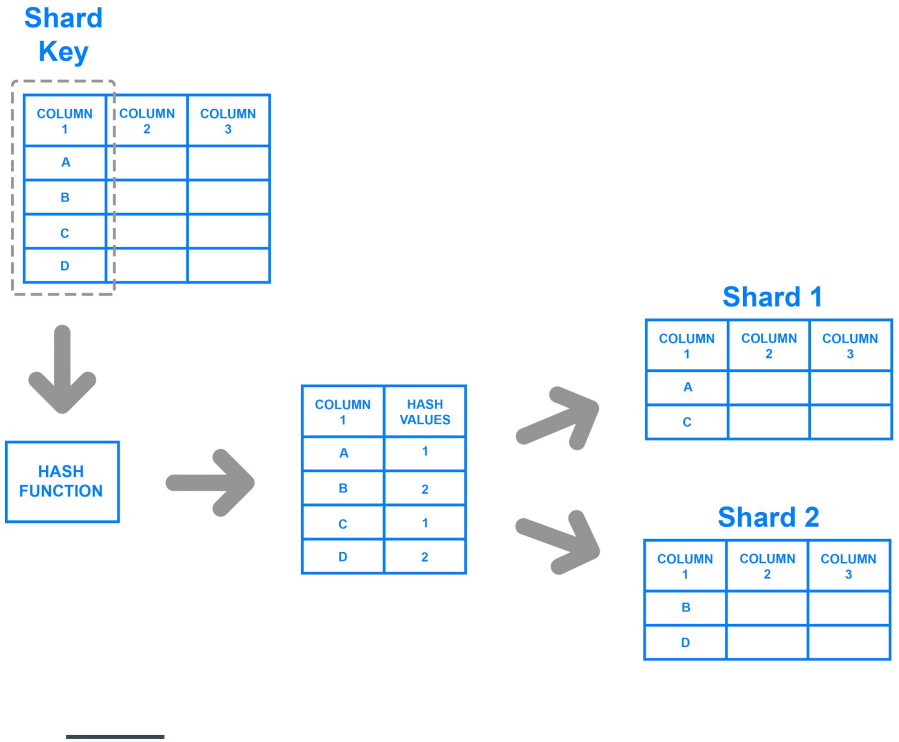
Sharding

- Logic is very close to that of hash tables
  - A “shard” = a “bucket”
- Advantages
  - Load is distributed across two shards evenly
    - Assuming a good hash function
  - Table sizes reduced
  - Index sizes reduced
  - Parallelism increased



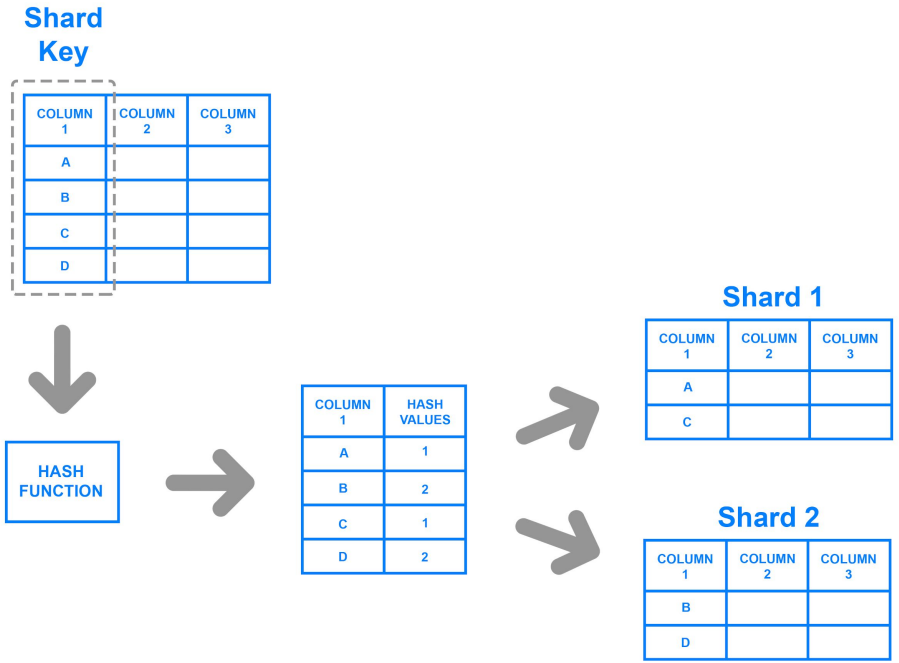
Sharding

- This is a form of *horizontal scaling*
  - If you want to handle more load just add more shards
    - “Just”
- Compare with *vertical scaling*
  - Increase CPU, Memory, etc. on a single machine
- Note that sharding is a *share nothing* architecture
  - Shards know nothing about other shards



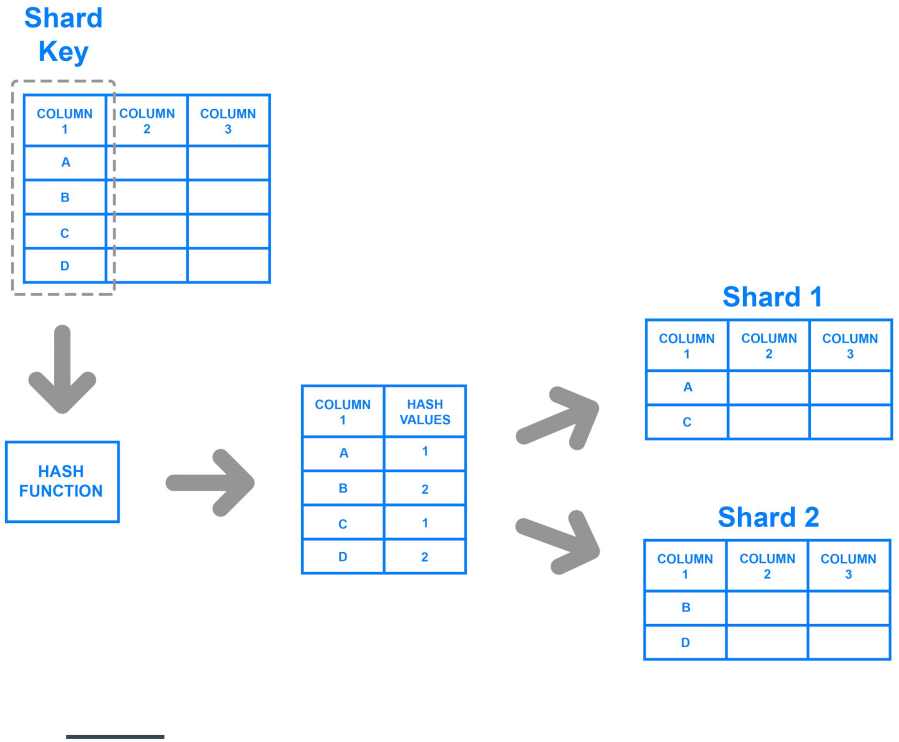
Sharding

- Disadvantages
  - Increased complexity of SQL
    - If you want to run queries over all rows in a table, you must consult multiple databases
  - Sharding complexity
    - You must now have sharding logic in your application layer



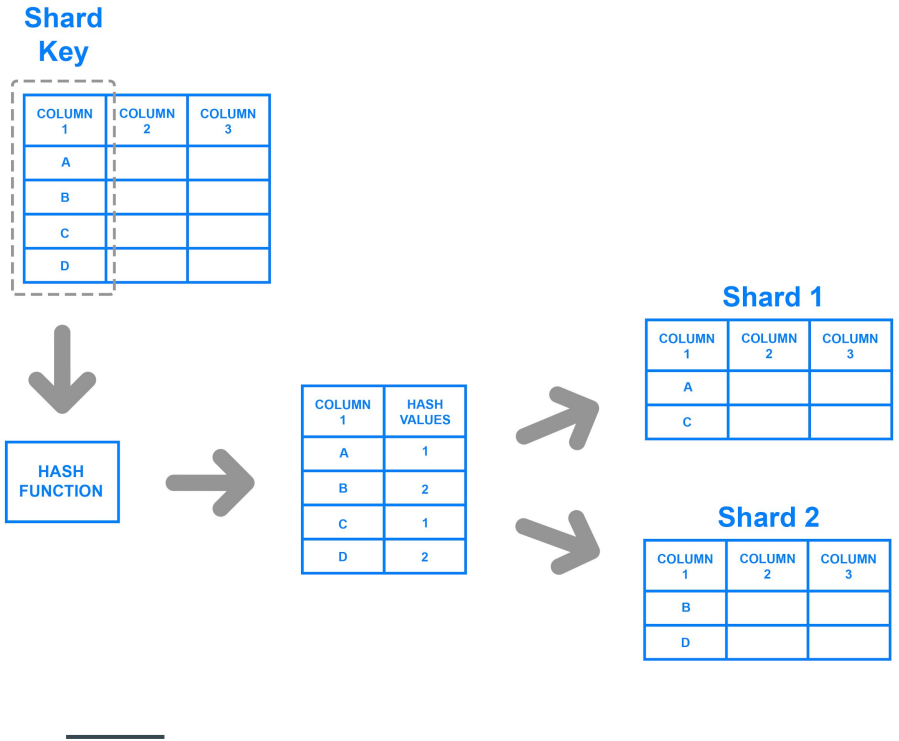
Sharding

- Disadvantages
  - No failover mechanism
    - Sharding does not directly handle failover
  - Failover is more complex
    - If you implement failover on top of sharing, you have more complexity
  - Operational issues
    - Adding or removing columns involve multiple servers



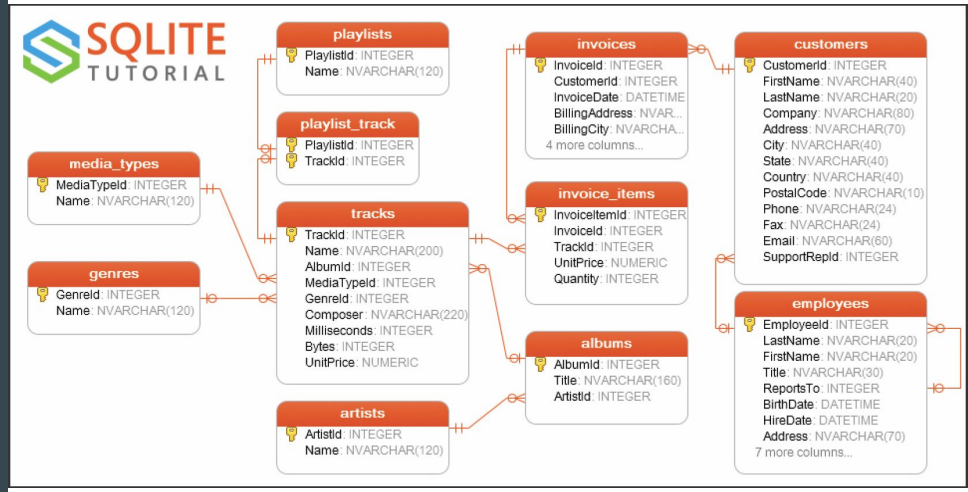
Sharding

- Despite these issues, sharding is widely used in industry
- It simply provides too much of an advantage over vertical scaling, so the complexity is worth it
  - NB: in mature systems!



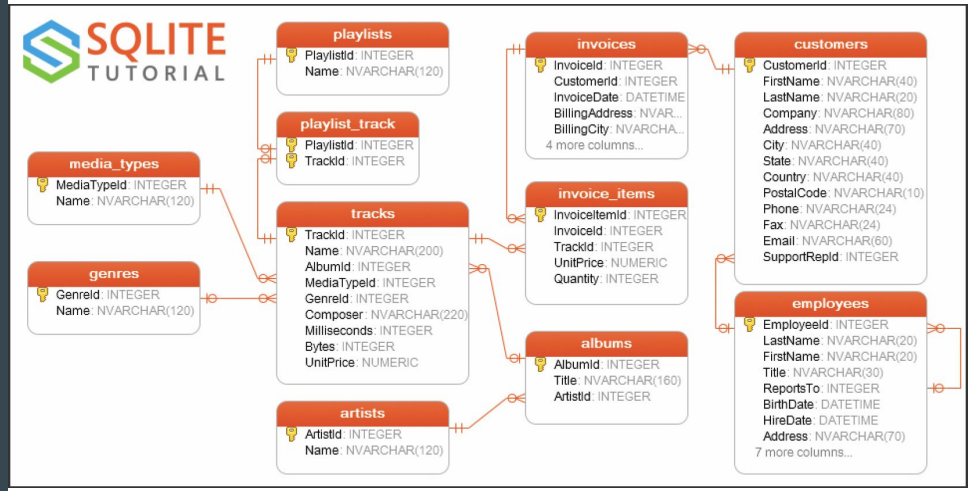
Sharding & Schemas

- Recall our database schema
- Which table is the best candidate for sharding in this database schema?
- What are some problems with sharding on that table?



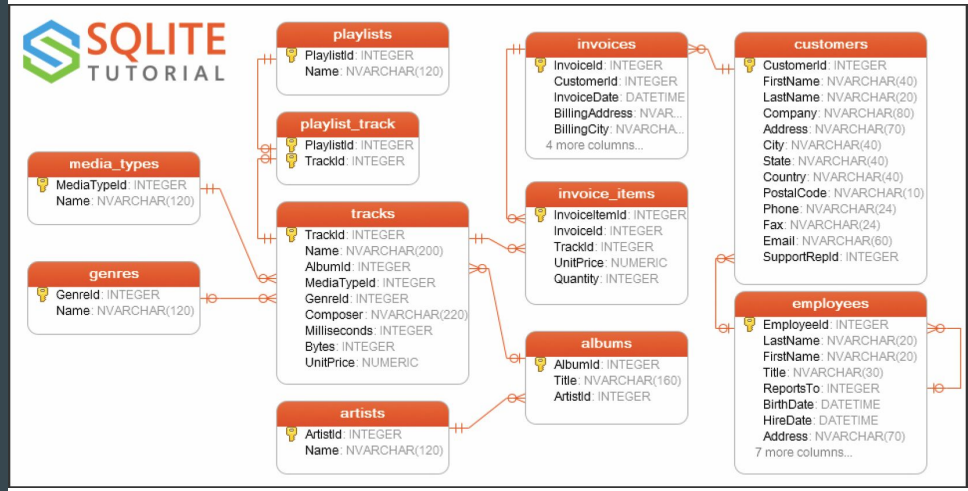
Sharding & Schemas

- Artists are the obvious table to shard on
  - Albums and tracks are both associated with Artist, so they will all end up in the same shard
- But there are problems with this!
  - Other tables refer to tracks that are *not* tied to artists



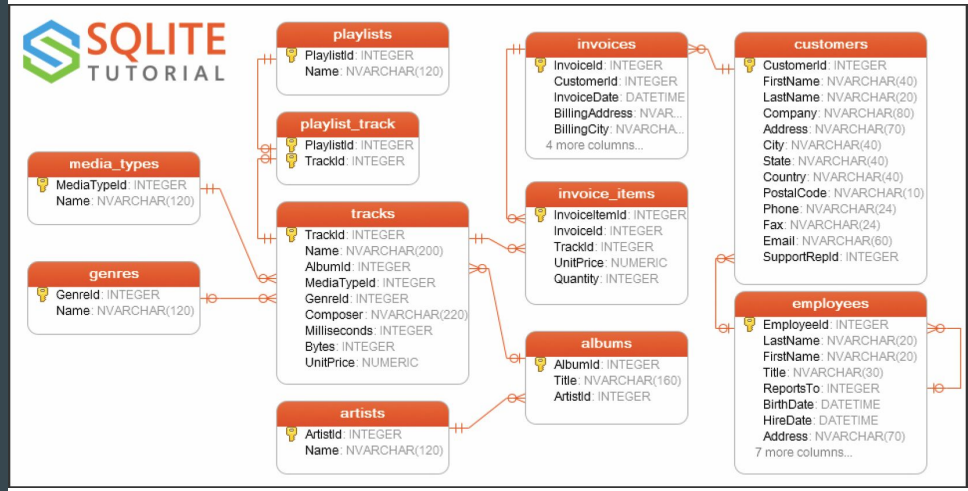
Sharding & Schemas

- What about
  - Media types and genres?
    - Maybe duplicate across all shards?
  - Play lists?
    - Hmmm, no good option
  - Invoice Items
    - Yikes



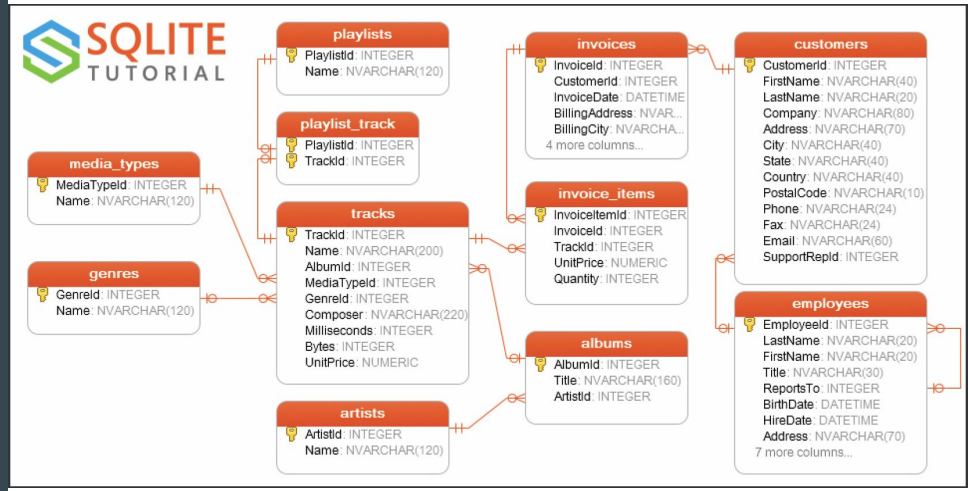
Sharding & Schemas

- Our schema is *not* a good candidate for a sharding architecture



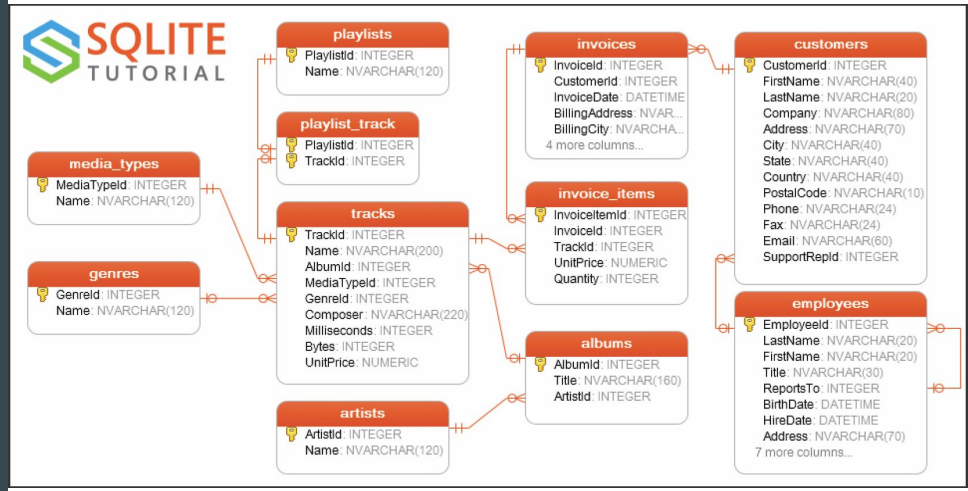
Sharding & Schemas

- However, what if we offered ChinookSAAS where this functionality was provided to multiple users?
- Now we can partition on user, and all data for a given user lives on the same shard
  - Very shard friendly!
  - Very scalable!



Sharding Implementation

- Sharding can be implementing many different ways, with two primary options
  - Application level sharding: You explicitly write sharding logic in your application code
  - Data-store level sharding: The datastore transparently shards data for you



Rails Sharding

- Rails 6.1 introduced database sharding as a framework feature
- Works with any backing data store
- Can be integrated into the request setup to automatically shard
    - Almost always on user

```ruby
ActiveRecord::Base.connected_to(role: :reading) do
  User.first # reads from default replica
  Dog.first # reads from default replica

  AnimalsRecord.connected_to(role: :writing, shard: :one) do
    User.first # reads from default replica
    Dog.first # reads from shard one primary
  end

  User.first # reads from default replica
  Dog.first # reads from default replica

  ApplicationRecord.connected_to(role: :writing, shard: :two) do
    User.first # reads from shard two primary
    Dog.first # reads from default replica
  end
end
```

Rails Sharding

- A lot of advantages to this approach
  - Data-store agnostic
  - Pretty simple to understand
  - Works well for simple sharding schemes
- Disadvantages
  - Almost zero administrative help for your shards

```ruby
ActiveRecord::Base.connected_to(role: :reading) do
  User.first # reads from default replica
  Dog.first # reads from default replica

  AnimalsRecord.connected_to(role: :writing, shard: :one) do
    User.first # reads from default replica
    Dog.first # reads from shard one primary
  end

  User.first # reads from default replica
  Dog.first # reads from default replica

  ApplicationRecord.connected_to(role: :writing, shard: :two) do
    User.first # reads from shard two primary
    Dog.first # reads from default replica
  end
end
```

Rails Sharding

- A lot of advantages to this approach
  - Data-store agnostic
  - Pretty simple to understand
  - Works well for simple sharding schemes
- Disadvantages
  - Almost zero administrative help for your shards

```ruby
ActiveRecord::Base.connected_to(role: :reading) do
  User.first # reads from default replica
  Dog.first # reads from default replica

  AnimalsRecord.connected_to(role: :writing, shard: :one) do
    User.first # reads from default replica
    Dog.first # reads from shard one primary
  end

  User.first # reads from default replica
  Dog.first # reads from default replica

  ApplicationRecord.connected_to(role: :writing, shard: :two) do
    User.first # reads from shard two primary
    Dog.first # reads from default replica
  end
end
```

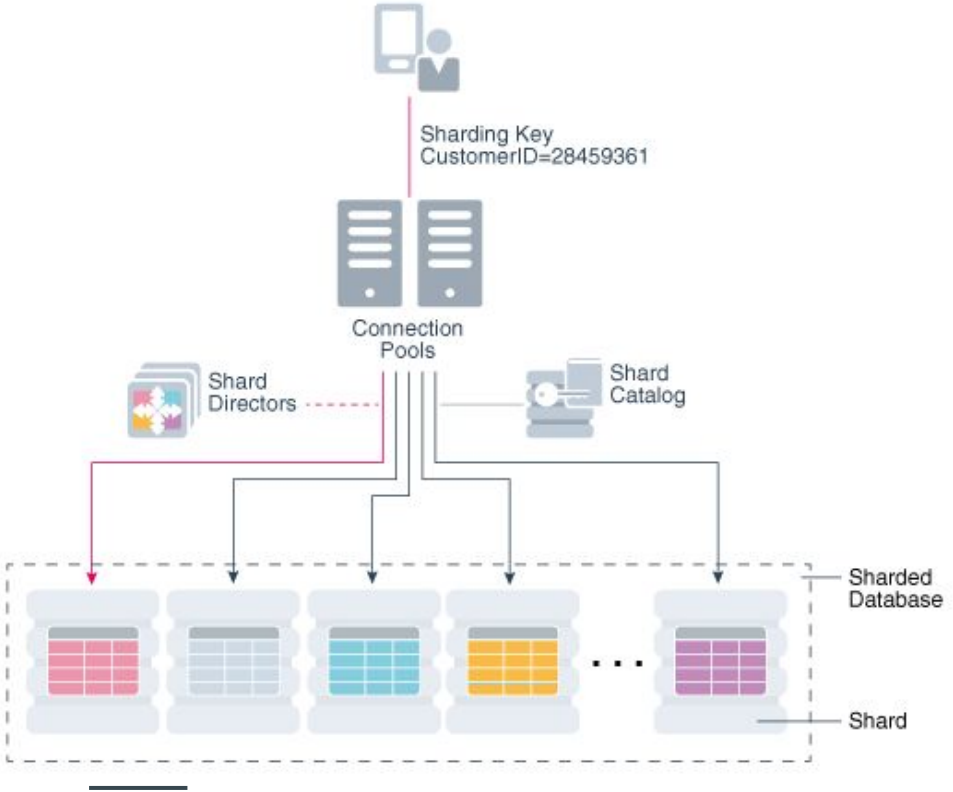
Rails Sharding

- This style of application level sharding can be implemented in almost any programming environment
- Increasingly seeing libraries for doing application level sharding

```ruby
ActiveRecord::Base.connected_to(role: :reading) do
  User.first # reads from default replica
  Dog.first # reads from default replica

  AnimalsRecord.connected_to(role: :writing, shard: :one) do
    User.first # reads from default replica
    Dog.first # reads from shard one primary
  end

  User.first # reads from default replica
  Dog.first # reads from default replica

  ApplicationRecord.connected_to(role: :writing, shard: :two) do
    User.first # reads from shard two primary
    Dog.first # reads from default replica
  end
end
```

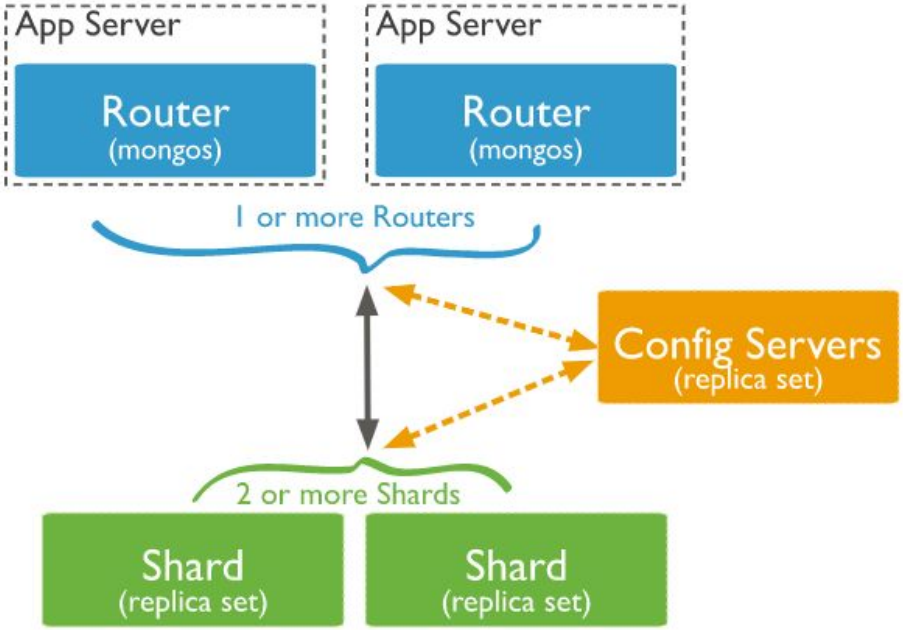
DB Sharding

- MySQL and PostgreSQL have sharding solutions but it is not baked into the core of either
- Oracle includes a sophisticated sharding architecture
  - Shard directors make sharding transparent to database connections



Mongo Sharding

- MongoDB has integrated sharding
  - Has had sharding for a long time
  - One of the early advantages of Mogo
- Mongo is more friendly for sharding because it does not emphasize inter-document relationships



Mongo Sharding

- Don’t have to worry about data consistency if you don’t worry about data consistency

Mongo Sharding

- Sharding can be accomplished with a one-liner once you have the shard server configuration set up properly
- Even I have to admit that's pretty sweet

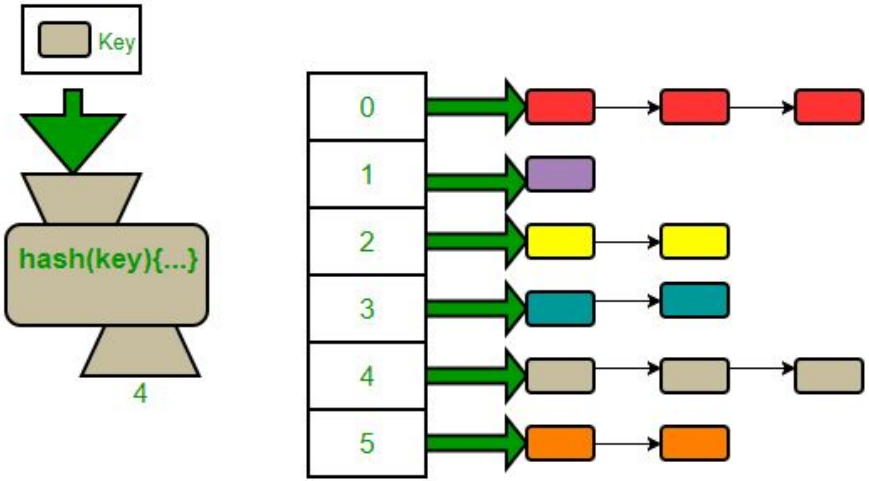
```
sh.shardCollection(<namespace>, <key>)
```

Redis Sharding

- It turns out that redis automatically shards in a clustered configuration
- This is possible because redis acts like a big hash table
  - No inter-relations between items in a Redis store

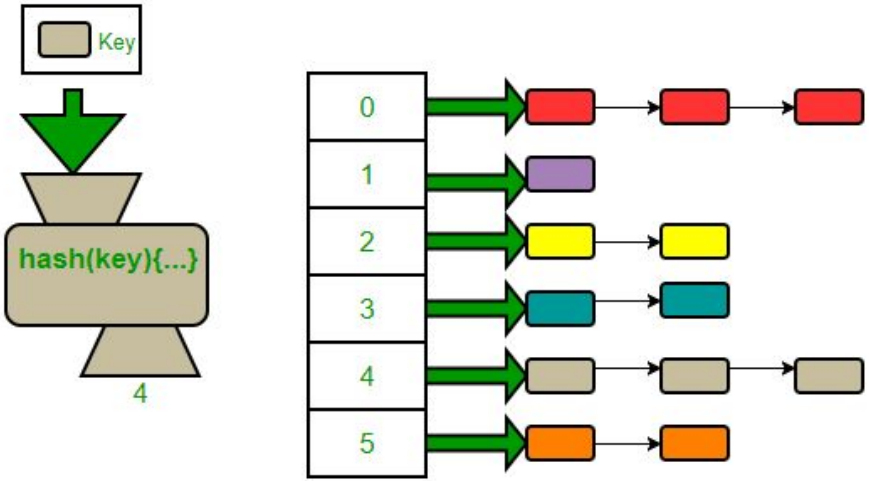
Rehashing Shards

- As with hash table buckets, shards may become too large
- When this happens you need to increase the number of shards and *rehash* your data to the new, appropriate shard
  - This is a “Stop The World” event



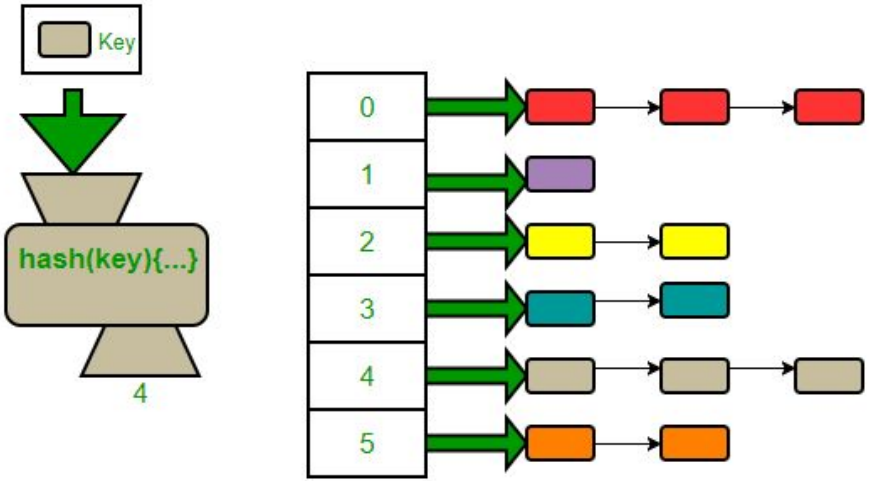
Rehashing Shards

- Google uses sharding heavily internally
- The greatest fear (15 years ago) was “Will I need to rehash my shard?”
  - Probably much better infrastructure now



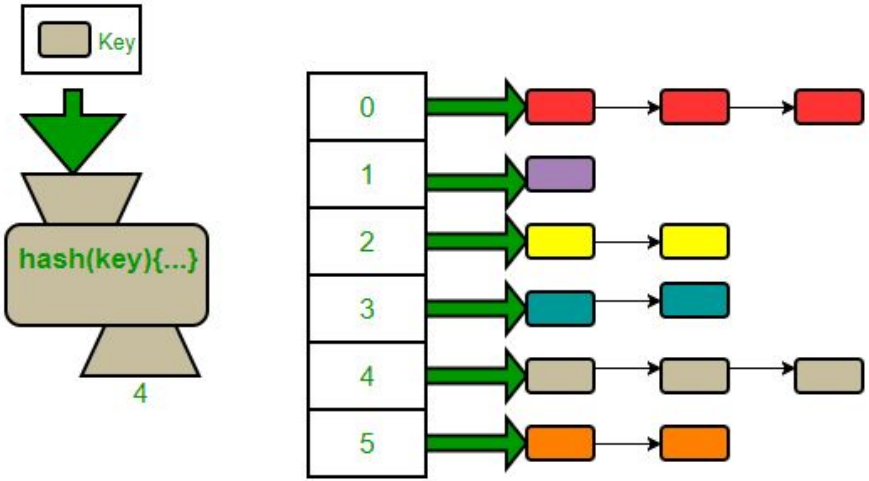
Sharding Alternative

- Sharding aims to split your data *statistically*, using a good hash key to distribute your data across shards
- What if, rather than that, we assigned a user a shard number on creation?
  - Rather than computing a hash key, just use the users shard#



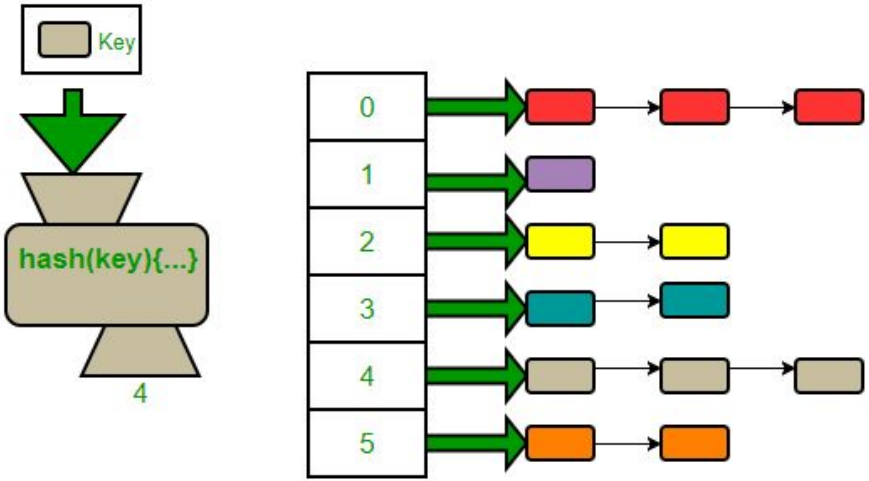
A Sharding Alternative

- Advantages
  - Simple
  - Easier to scale horizontally
    - Just add a new server and start sending traffic to it
  - User info can be moved between shards one at a time, rather than a “stop the world” event



A Sharding Alternative

- Disadvantages
  - No statistical guarantees that you are spreading your data out effectively
  - More complex key generation logic
  - Given only the non-shard user data, not possible to determine which shard they are on



Sharding

- Sharding is a useful technique for *horizontal scaling* of data stores
- It is closely related to *hash tables*
- Can be difficult to implement, depending on if your schema is *shard friendly* or not
- Many systems provide sharding infrastructure
  - Application Level
  - Datastore level
- Resharding can be *very* expensive
- Despite the complexity around sharding, it is widely used due to its scaling advantages


MONTANA
STATE UNIVERSITY