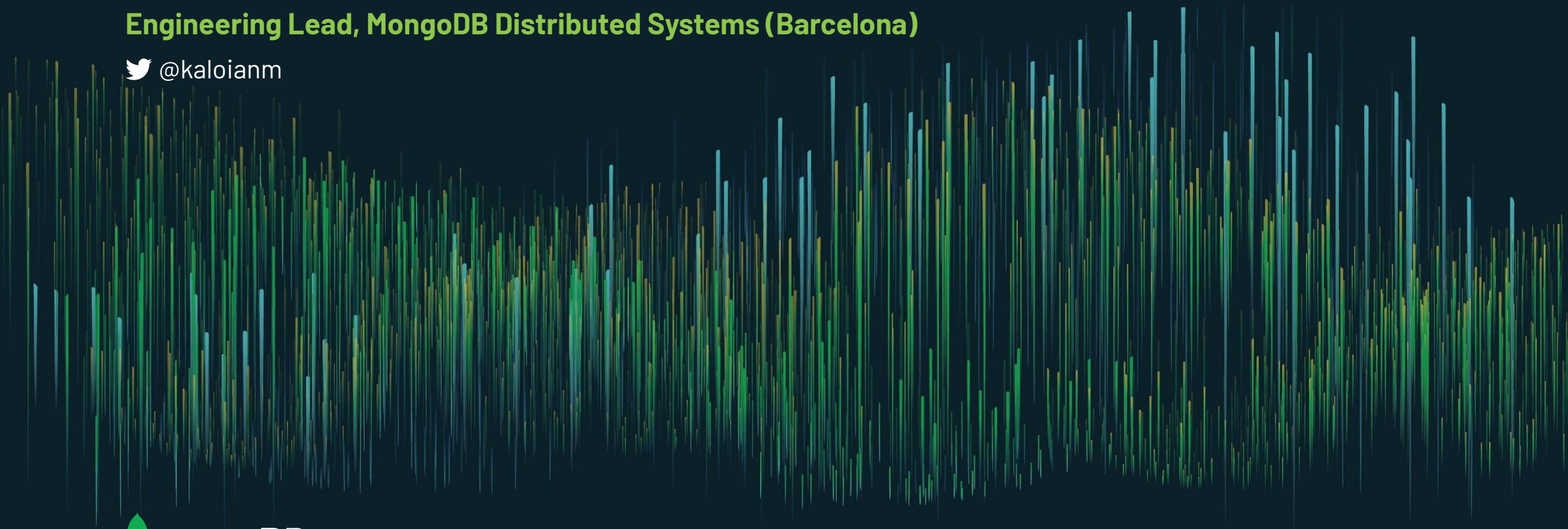
MongoDB Server Engineering

Kaloian Manassiev
Engineering Lead, MongoDB Distributed Systems (Barcelona)
@kaloianm



mongoDB.

Talk outline

- What is MongoDB and who uses it?
- Engineering team and code base facts
- C++ and the basic building blocks
- MongoDB sharding

What is MongoDB?

- Universal database for scalable modern applications
- Uses the ***document*** data model
- Supports flexible schema
- Supports transactions

Document model

Relational model

People Table

ID	Name	Job	House_ID	Car_ID
P001	Leo	Product Marketing	H001	C001
P002	Doug	Sales Enablement	H002	C002

Houses Table

House_ID	House_Bedrooms	House_Bathrooms	House_Basement
H001	0	1	no
H002	4	3	yes

Cars Table

CID	Car_Make	Car_Model	Car_Year
C001	Toyota	Corolla	1989
C002	Tesla	Model S	2016

Object-relational mapping (ORM)

- **class Person**
  - std::vector<House> getHouses()
  - void setHouses(const std::vector<House>&)
  - std::vector<Car> getCars()
  - void setCars(const std::vector<Car>&)

Relational model

People Table

ID	Name	Job	House_ID	Car_ID
P001	Leo	Product Marketing	H001	C001
P002	Doug	Sales Enablement	H002	C002

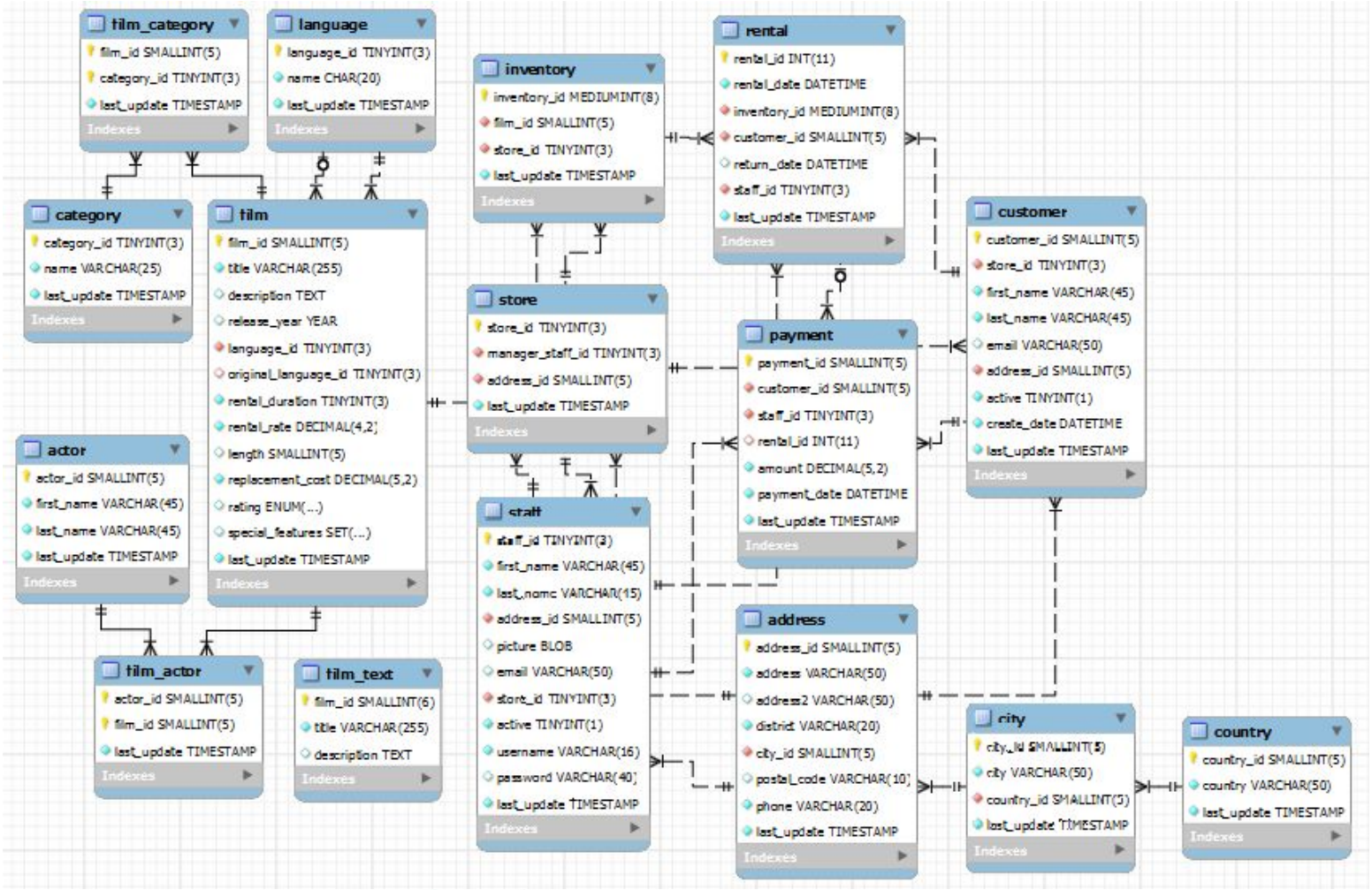
Houses Table

House_ID	House_Bedrooms	House_Bathrooms	House_Basement
H001	0	1	no
H002	4	3	yes

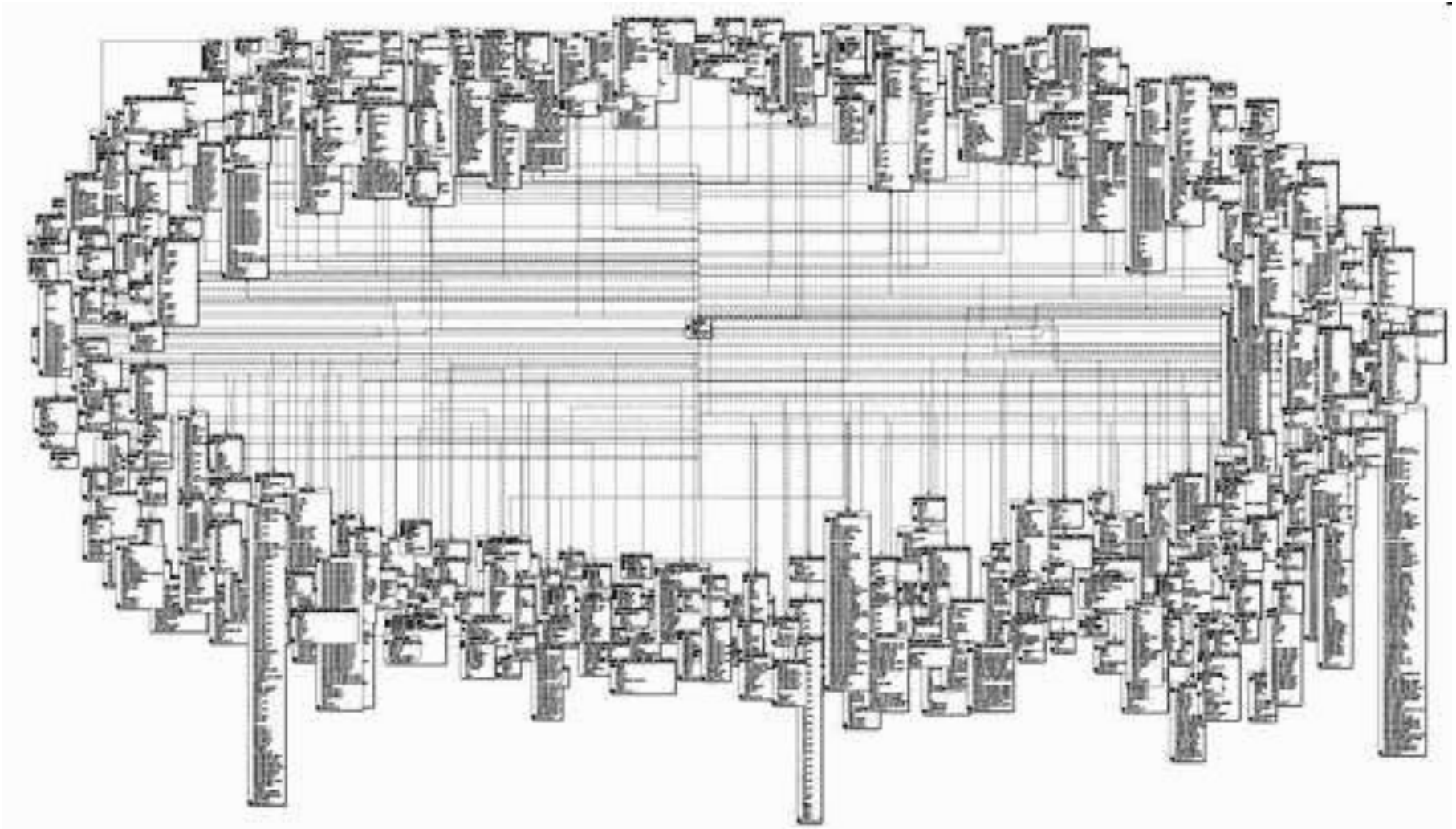
Cars Table

CID	Car_Make	Car_Model	Car_Year
C001	Toyota	Corolla	1989
C002	Tesla	Model S	2016

Relational model



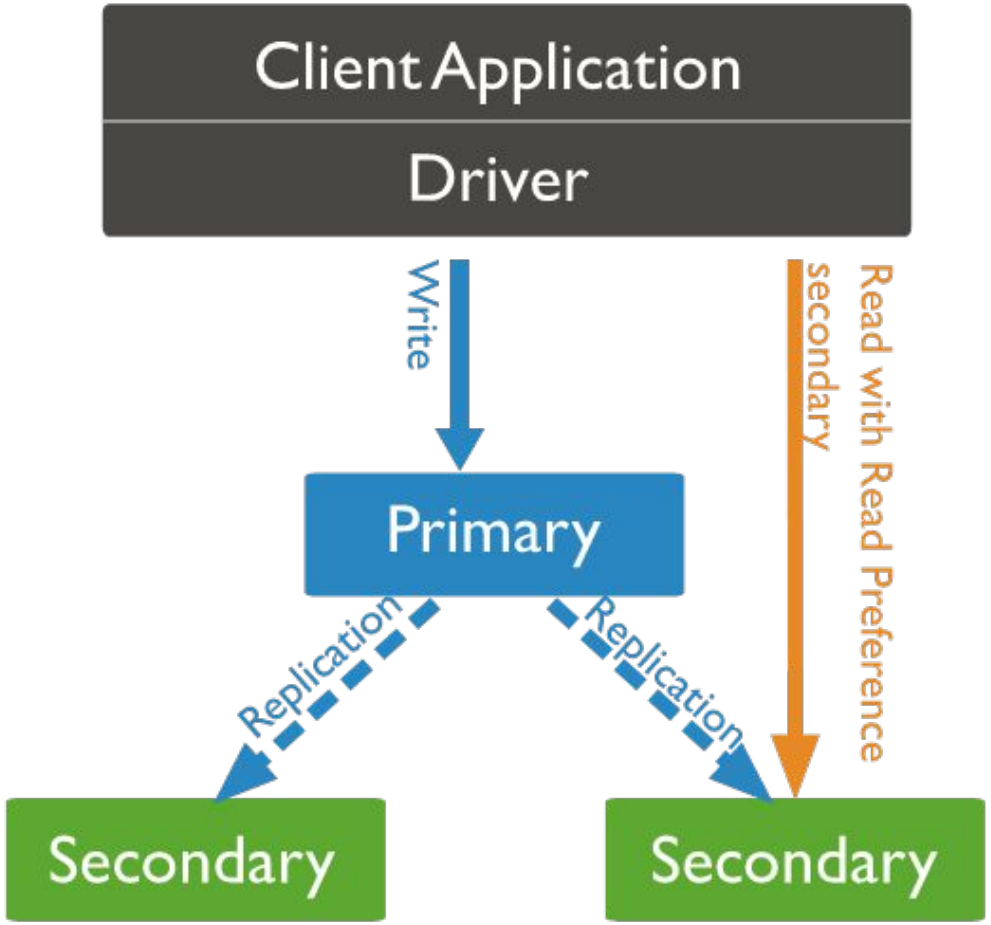
Relational model



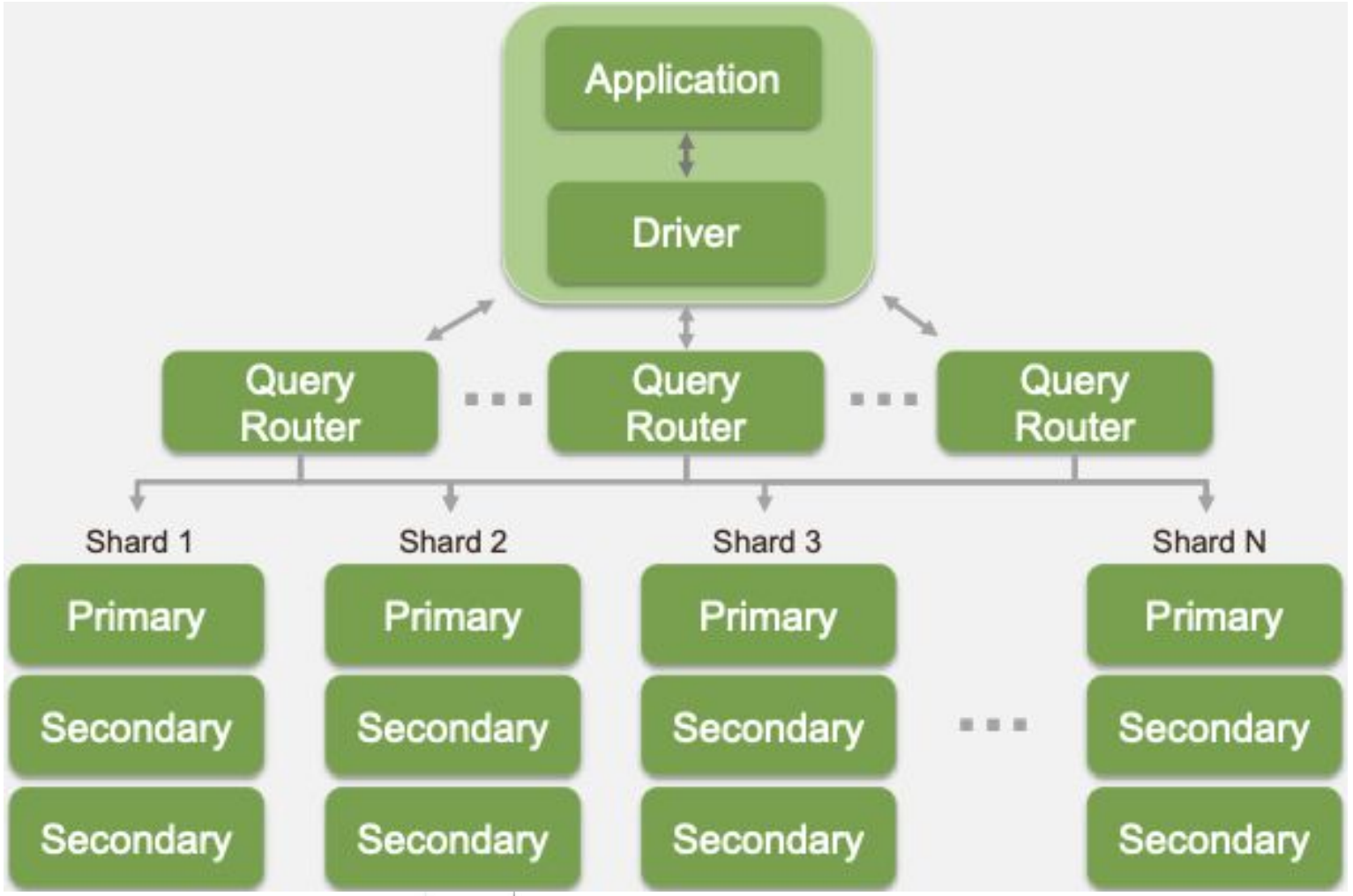
Document model

```
{
    _ID: “P001”,
    Name: “Leo”,
    Job: “Product Marketing”,
    House_Bedrooms: 0,
    House_Bathrooms: 1,
    House_Basement: “no”,
    Car_Make: “Toyota”,
    Car_Model: “Corolla”,
    Car_Year: 1989
}
```

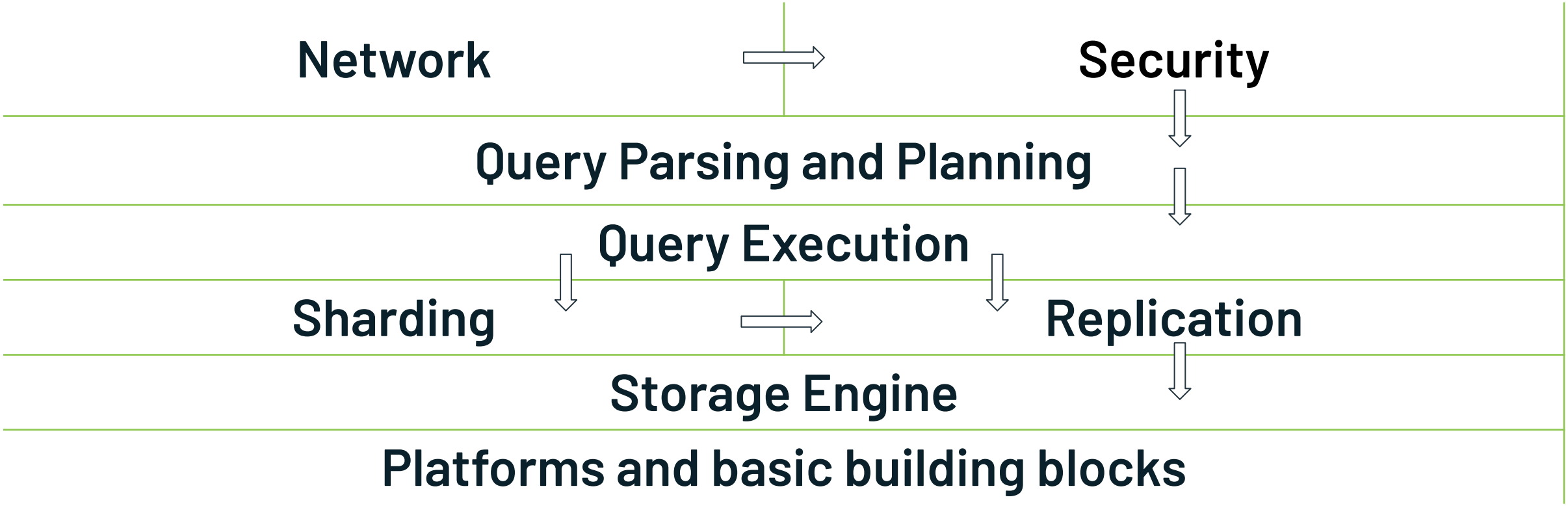
MongoDB replication



MongoDB sharding



MongoDB architecture



The MongoDB company

- Founded in 2009 and currently at 1,600+ employees
- ~300 engineers
  - Server and drivers
    - ~200 engineers in 5 different countries (USA, Ireland, Australia, Germany and Spain)
    - 4 members of the C++ Standards Committee
  - Cloud
    - ~90 engineers

The MongoDB server code base

- Open source and hosted on Github
- 860,000 lines of C++17 code
  - 1,33M lines if all the C++ unit-tests are counted
- 370,000 lines of JavaScript code for administrative utilities and tests
- Runs under continuous integration after (almost) every commit

The MongoDB server code base

- Uses SCons as a build system
- Multi-platform compilation for RHEL7.2, Ubuntu, Windows, OSX, ARM, PPC64LE, IBM s360x and others
- Uses external libraries, such as Boost 1.70.0, ASIO, MozJS and Abseil

The good, the bad and the ugly of using C++

- (Almost) direct control over memory allocations
  - TCMalloc instead of the default CRT allocator
- Many compiler optimisations
- Support for exceptions

- Debugging of (optimised) core dumps is tough

Most used standard C++ features

- **Containers:** std::vector, std::set, std::map, std::unordered_map (Abseil)
- **Smart pointers:** boost::optional/std::unique_ptr/shared_ptr/weak_ptr
- **Synchronization:** std::mutex, std::condition_variable, Futures (custom implementation)
- **Threading:** std::thread

Getting to C++17

- 2014 - C++98
- 2015 - C++98 for production and C++11 for new features
- 2016 - C++11 for production and C++14 for new features
- 2017 - C++14 for production and C++17 for new features
- 2018 - C++17 for production

Most used C++17 features

- if constexpr
- if (Type init; cond)
- CTAD (Class Template Argument Deduction)
- Guaranteed copy elision
- Structured bindings - for (auto&& [key, value] : someMap)
- inline variables
- [[nodiscard]]

Avoiding global variables

- Everything running under an instance of server is rooted under a single ServiceContext
- Use Decorations in order to dynamically extend the ServiceContext with child services

Class decorations

- Extending a class at runtime
- **service_context.h:** `class ServiceContext final : public Decorable<ServiceContext>`
- **lscache.cpp:** `const auto getLSCache = ServiceContext::declareDecoration<std::unique_ptr<LogicalSessionCache>>();`

Startup and shut down

- **Startup**
  - Static initialization (enforced by ASAN/UBSAN)
  - Per-component initialization functions
- **Shutdown**
  - Per-component shutdown functions
  - Static destruction (enforced by ASAN/UBSAN)

Errors and exceptions

- All errors described through a single `Status` class
  - Stores `OK` or `code`, `message` and `extra_info`
- All exceptions derive from a single class
  - `class DBException : public std::exception`
- 1:1 conversion between Status and DBException

Why custom Futures?

- Interruptibility and deadline enforcing
- Control over which threads execute the continuations
- Runtime diagnosability (who is waiting on what)
- Richer integration with our error types and exceptions

- `std::future::get()` becomes
  - `void mongo::Future::get(Interruptible*)`
  - `Status mongo::Future::getNoThrow(Interruptible*)`

Why custom Mutexes?

- Runtime diagnosability and deadlock detection
  - Which thread owns which mutex
  - Which thread is blocked on which mutex
- Runtime discovery of hot mutexes (without attaching gdb or perf)
- *(Future plan)* Support for user-level cooperative multithreading

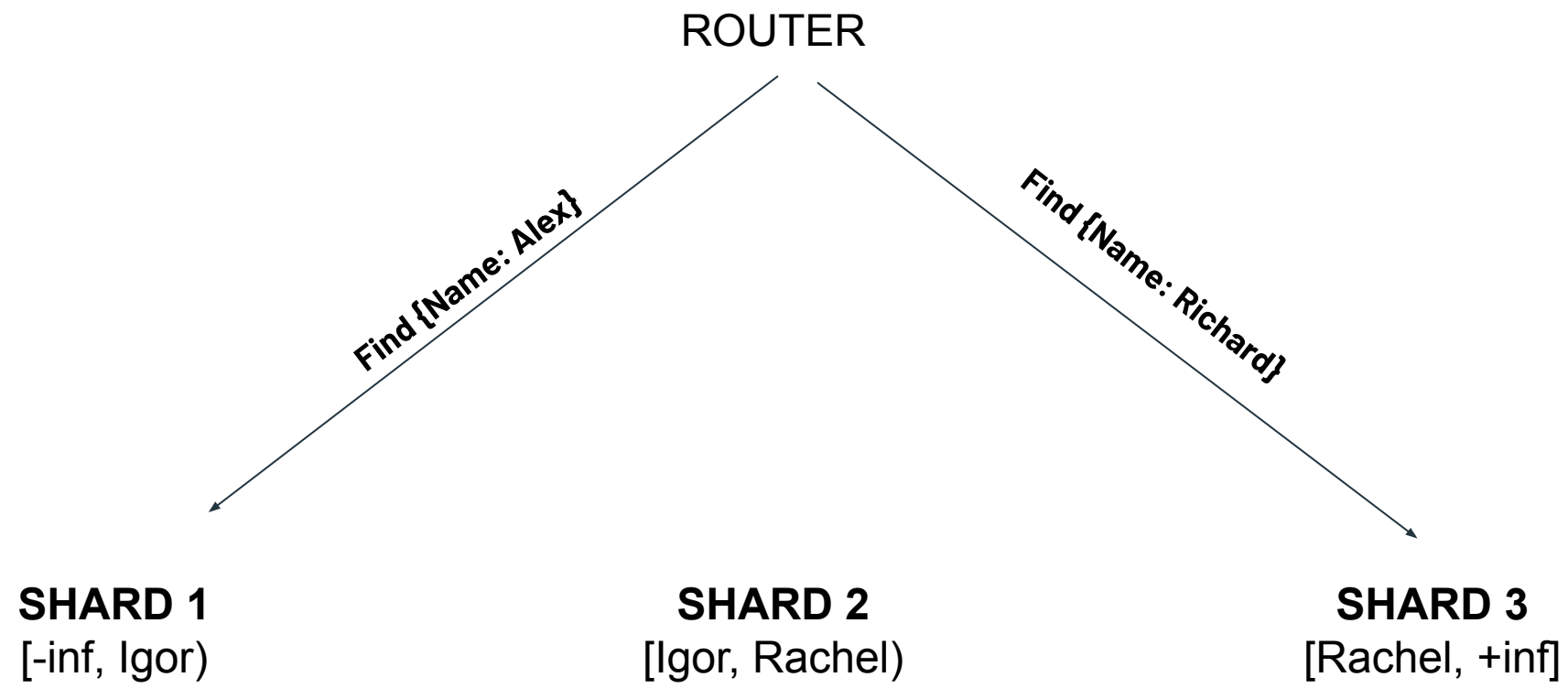
- `std::mutex _mutex` becomes
  - `mongo::Mutex _mutex = MONGO_MAKE_LATCH("HOT_MUTEX");`

Next:
MongoDB Sharding

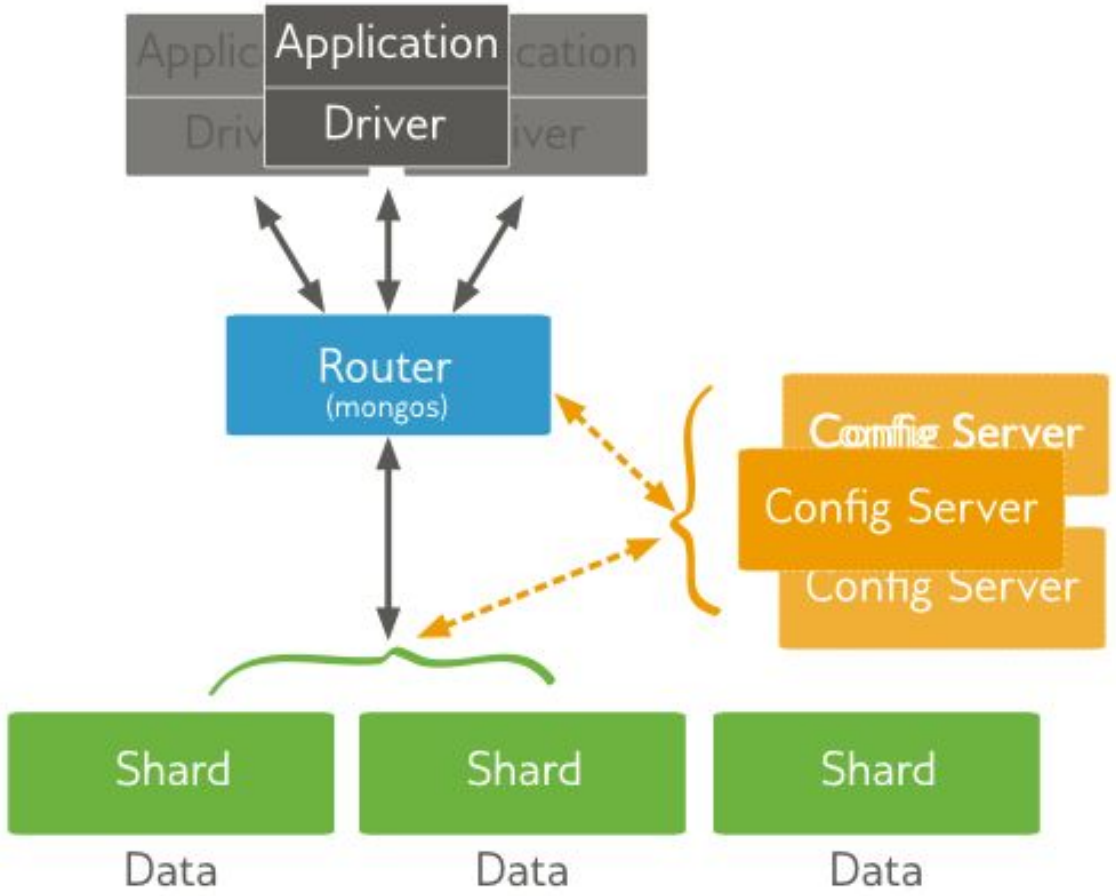
MongoDB sharding purpose

- Horizontally scale-out data
- Present unified view of the database server
- Periodically rebalance data without downtime
- Distributed transactions

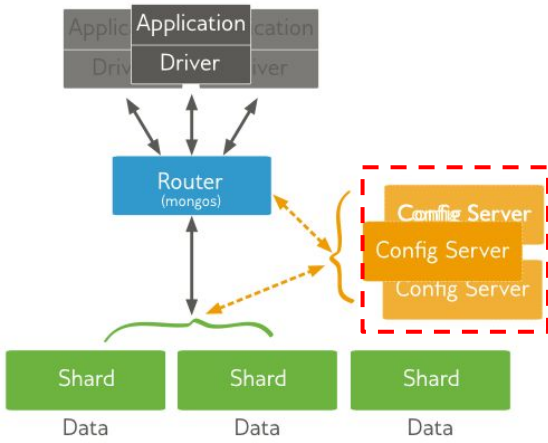
Data partitioning and routing



Sharding architecture

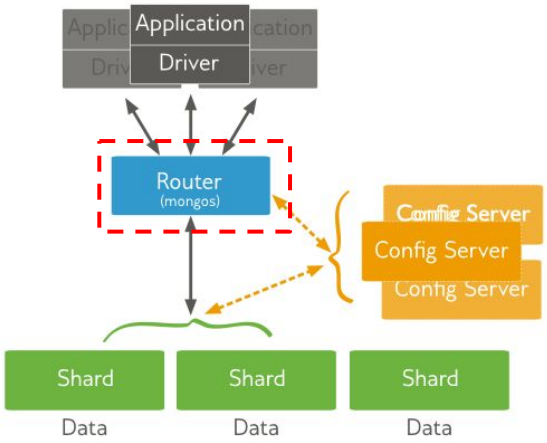


Config server



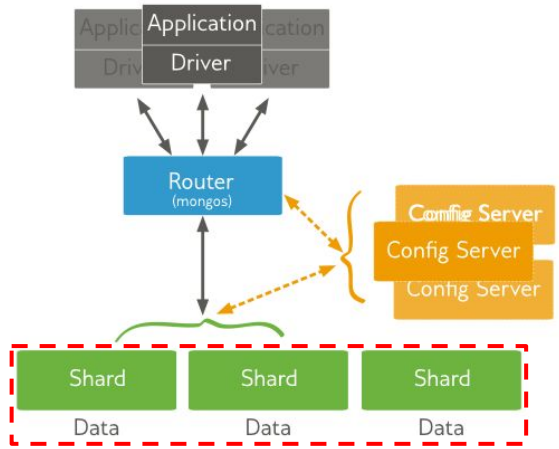
- Durably persists where data is located
- Uses replication for durability and high availability
- Does not contain any user data, only metadata

Router(s)



- Caches where the data is located
  - Reads it from the config server
- Has no persistent storage
- Speaks the database server protocol

Shard(s)



- Caches what data **it** owns
  - Reads it from t he config server
- Durably persists the actual data of the collection

Routing metadata

SHARD 1
[-inf, Igor)

SHARD 2
[Igor, Rachel)

SHARD 3
[Rachel, +inf]

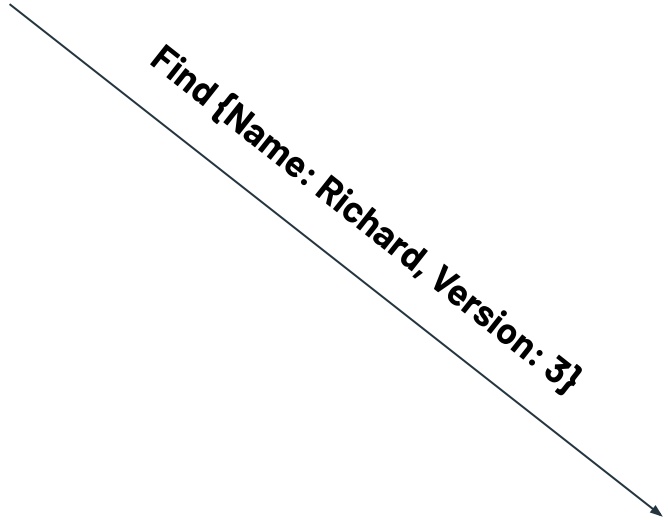
```
{min: -inf, max: Igor, shard: Shard1, version: 1}
{min: Igor, max: Rachel, shard: Shard2, version: 2}
{min: -inf, max: Igor, shard: Shard3, version: 3}
```

Routing metadata

CONFIG

```
{min: -inf, max: Igor,
   shard: Shard1, version: 1}
{min: Igor, max: Rachel,
   shard: Shard2, version: 2}
{min: -inf, max: Igor,
   shard: Shard3, version: 3}
```

ROUTER
(cache of config at **Version: 3**)



SHARD 1
[-inf, Igor)
Version: 1

SHARD 2
[Igor, Rachel)
Version: 2

SHARD 3
[Rachel, +inf]
Version: 3

Routing metadata

CONFIG

```
{min: -inf, max: Igor,
   shard: Shard1, version: 1}
{min: Igor, max: Rachel,
   shard: Shard2, version: 2}
{min: -inf, max: Igor,
   shard: Shard3, version: 3}
```

ROUTER
(cache of config at **Version: 3**)

Find {Name: Igor, Version: 2}

SHARD 1
[-inf, Igor)
Version: 1

SHARD 2
[Igor, Rachel)
Version: 2

SHARD 3
[Rachel, +inf]
Version: 3

Routing metadata

CONFIG

```
{min: -inf, max: Igor,
   shard: Shard1, version: 1}
{min: Igor, max: John,
   shard: Shard2, version: 2}
{min: John, max: James,
   shard: Shard2, version: 2}
{min: James, max: Rachel,
   shard: Shard2, version: 2}
{min: -inf, max: Igor,
   shard: Shard1, version: 3}
```

ROUTER
(cache of config at **Version: 3**)

Find {Name: Igor, Version: 2}

SHARD 1
[-inf, Igor)
Version: 1

SHARD 2
[Igor, John), [John, James)
[James, Rachel)
Version: 2

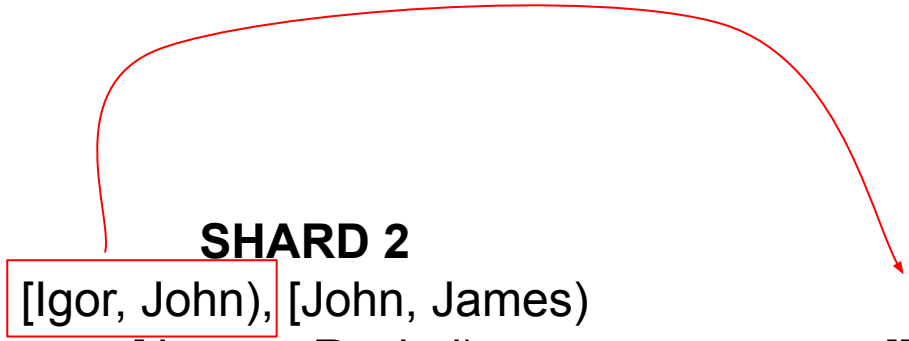
SHARD 3
[Rachel, +inf]
Version: 3

Routing metadata

CONFIG

```
{min: -inf, max: Igor,
   shard: Shard1, version: 1}
{min: Igor, max: John,
   shard: Shard2, version: 2}
{min: John, max: James,
   shard: Shard2, version: 2}
{min: James, max: Rachel,
   shard: Shard2, version: 2}
{min: -inf, max: Igor,
   shard: Shard1, version: 3}
```

ROUTER
(cache of config at **Version: 3**)



SHARD 1
[-inf, Igor)
Version: 1

SHARD 2
[Igor, John), [John, James)
[James, Rachel)
Version: 2

SHARD 3
[Rachel, +inf]
Version: 3

Routing metadata

CONFIG

```
{min: -inf, max: Igor,
   shard: Shard1, version: 1}
{min: Igor, max: John,
   shard: Shard3, version: 4}
{min: John, max: James,
   shard: Shard2, version: 5}
{min: James, max: Rachel,
   shard: Shard2, version: 2}
{min: -inf, max: Igor,
   shard: Shard1, version: 3}
```

ROUTER
(cache of config at **Version: 3**)

SHARD 1
[-inf, Igor)
Version: 1

SHARD 2
[John, James) [James, Rachel)
Version: 5

SHARD 3
[Igor, John), [Rachel, +inf]
Version: 4

Routing metadata

CONFIG

```
{min: -inf, max: Igor,
   shard: Shard1, version: 1}
{min: Igor, max: John,
   shard: Shard3, version: 4}
{min: John, max: James,
   shard: Shard2, version: 5}
{min: James, max: Rachel,
   shard: Shard2, version: 2}
{min: -inf, max: Igor,
   shard: Shard1, version: 3}
```

ROUTER
(cache of config at **Version: 3**)

Find {Name: Igor, Version: 2}

Version: 2 < 5

SHARD 1
[-inf, Igor)
Version: 1

SHARD 2
[John, James) [James, Rachel)
Version: 5

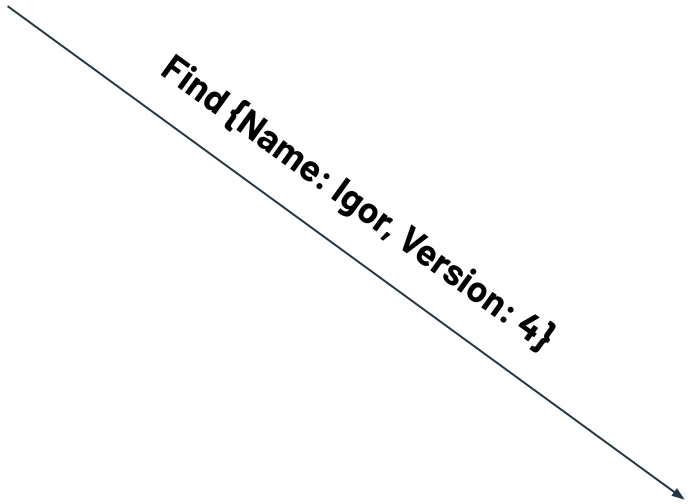
SHARD 3
[Igor, John), [Rachel, +inf]
Version: 4

Routing metadata

CONFIG

```
{min: -inf, max: Igor,
   shard: Shard1, version: 1}
{min: Igor, max: John,
   shard: Shard3, version: 4}
{min: John, max: James,
   shard: Shard2, version: 5}
{min: James, max: Rachel,
   shard: Shard2, version: 2}
{min: -inf, max: Igor,
   shard: Shard1, version: 3}
```

ROUTER
(cache of config at **Version: 5**)



SHARD 1
[-inf, Igor)
Version: 1

SHARD 2
[John, James) [James, Rachel)
Version: 5

SHARD 3
[Igor, John), [Rachel, +inf]
Version: 4


Questions?