

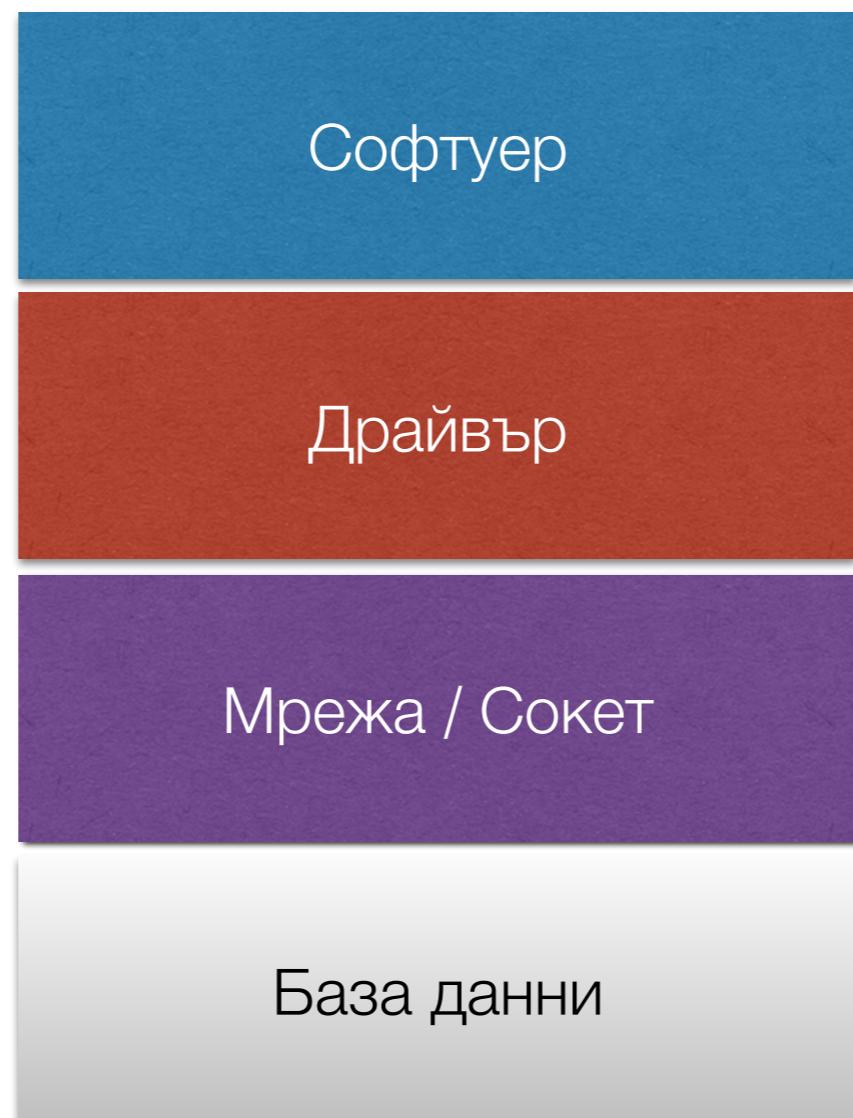
Драйвъри за MongoDB

Георги Христозов

Драйвъри в ДБМС

- връзката между софтуер и база данни
- обикновено са сравнително тънък слой в структурата на софтуера

Драйвъри



Какво правят

- грижат се за цялата комуникация на ниско ниво
- осигуряват API, което служи за абстракция

Какво НЕ правят

- не осигуряват object mapping (с изключения)
- не абстрагират изцяло базата (с изключения)

MongoDB драйвър

- аналогичен интерфейс на shell-а
- не дава гъвкавостта на JDBC/ODBC - „вързани“ сте за MongoDB
- по тази причина директната му употреба трябва да се ограничава

MongoDB драйвъри

- официални: C, C++, C#/.NET, Erlang, Go, Java, Node, Perl, PHP, Python, Ruby, Scala
- неофициални: ActionScript 3, Clojure, D, Dart, Delphi, F#, Groovy, Haskell, Lua, MatLab, Objective-C, OCaml, PowerShell, Prolog, R, Scheme, Smalltalk...

MongoDB драйвър

- осъществява комуникация между приложението и MongoDB
- TCP връзка на порт 27017
- custom протокол - съвместим само с MongoDB

Формат на header

```
struct MsgHeader {
    int32 messageLength; // total message size, including this
    int32 requestID;    // identifier for this message
    int32 responseTo;   // requestID from the original request
    int32 opCode;        // request type
}
```

Формат на операция

```
struct OP_UPDATE {
    MsgHeader header;           // standard message header
    int32      ZERO;            // 0 - reserved for future use
    cstring    fullCollectionName; // "dbname.collectionname"
    int32      flags;            // bit vector
    document   selector;        // the query to select
    document   update;          // the update to perform
}
```

Други грижи на драйвъра

- удобна BSON сериализация/десериализация за дадения език
- управление на връзките и ресурсите
- единно API между различни клиенти и сървъри

Блокиране

За клиента класическата write операция изглежда така:

1. Изпращане на заявка
2. Изчакване сървърът да запише данните
3. Обработка на резултата

Блокиране

- nevinaги се интересуваме от резултатите от операциите по записване
- nevinaги се интересуваме дали операциите по записване изобщо са приключили
- „плащаме“ overhead при всяка заявка

БЛОКИРАНЕ

```
function updatePost(data, success, failure) {  
    this.stats.insert({page: "post"});  
    var is0k = this.posts.update(data);  
    if (is0k) {  
        success();  
    } else {  
        failure();  
    }  
}
```

Винаги ли е нужно?

- двете операции са последователни и синхронни
- може (както в дадения код) да не ни интересува дали записването на събитие е успешно
- можем да изпратим първата заявка и да продължим с втората, без да очакваме отговор

Не-блокирующий вариант

```
function updatePost(data, success, failure) {  
    this.stats.insertUnsafe({page: "post"});  
    var isOk = this.posts.update(data);  
    if (isOk) {  
        success();  
    } else {  
        failure();  
    }  
}
```

Контрол

- при всяка заявка може да са задава WriteConcern
- той указва на драйвъра дали и какво събитие да чака преди да продължи
- избира се според конкретната ситуация - няма универсални решения

Нива на WriteConcern

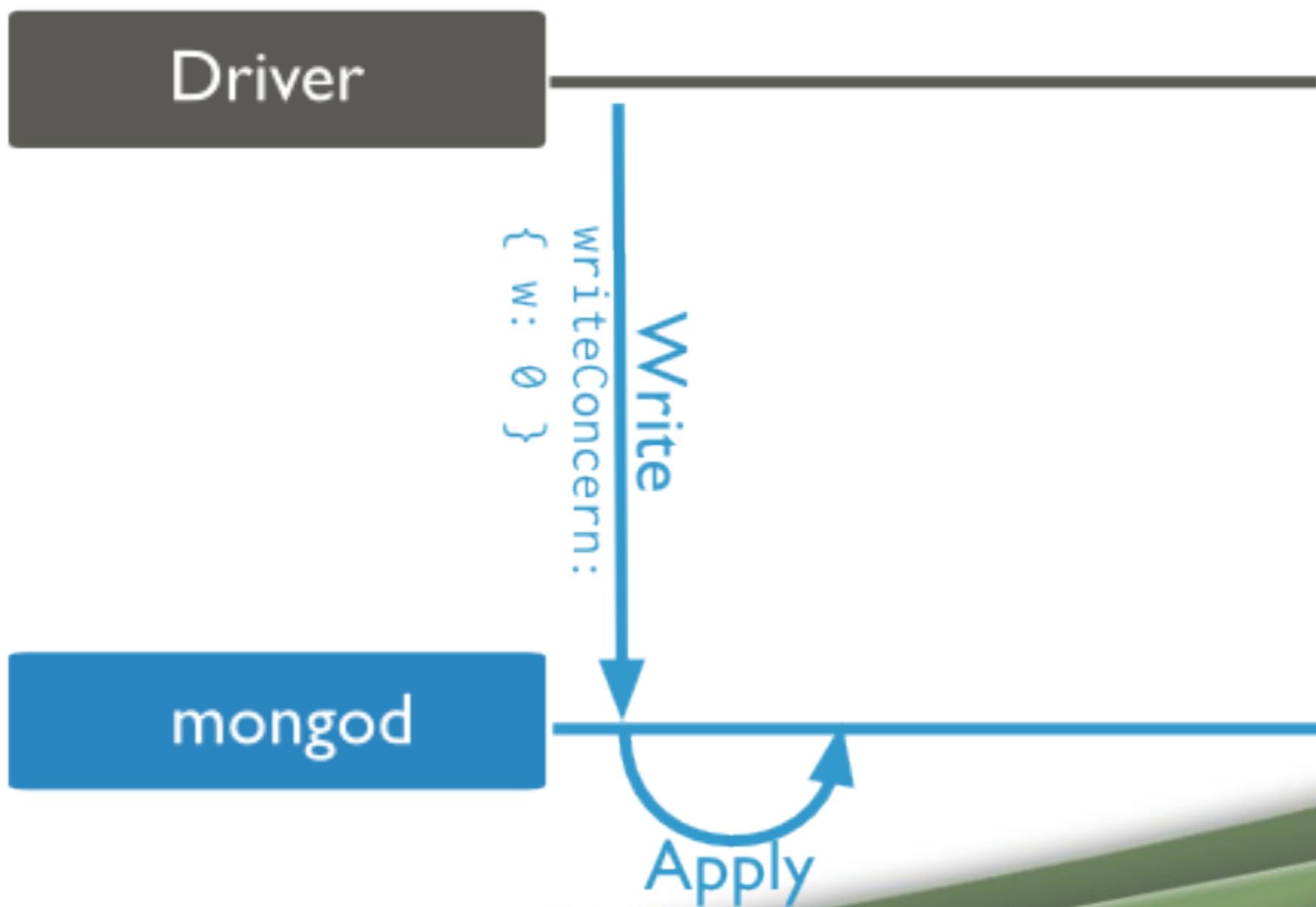
(в намаляваща надеждност)

- replica acknowledged
- journaled
- acknowledged (по подразбиране)
- unacknowledged
- без проверка

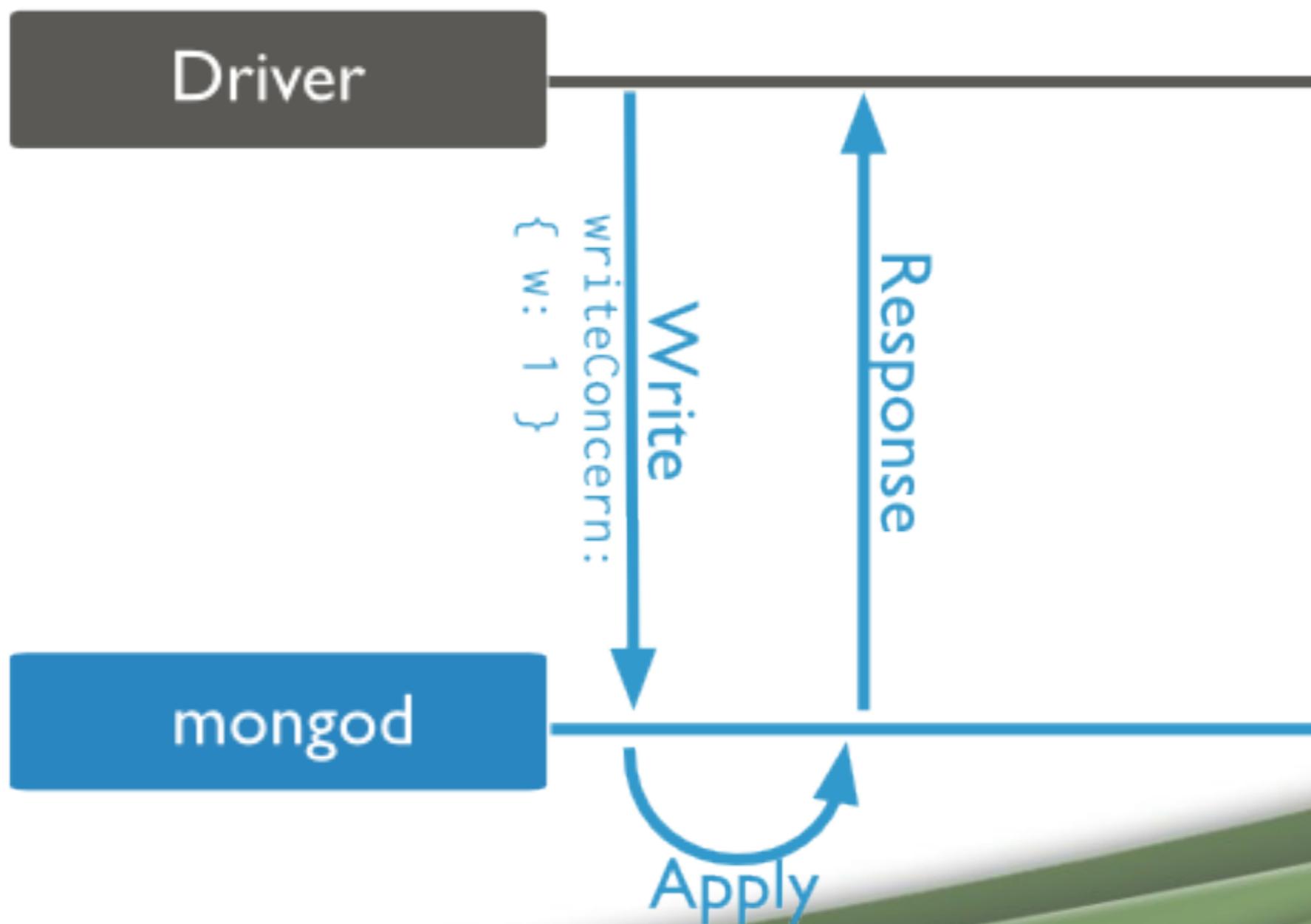
Без проверка



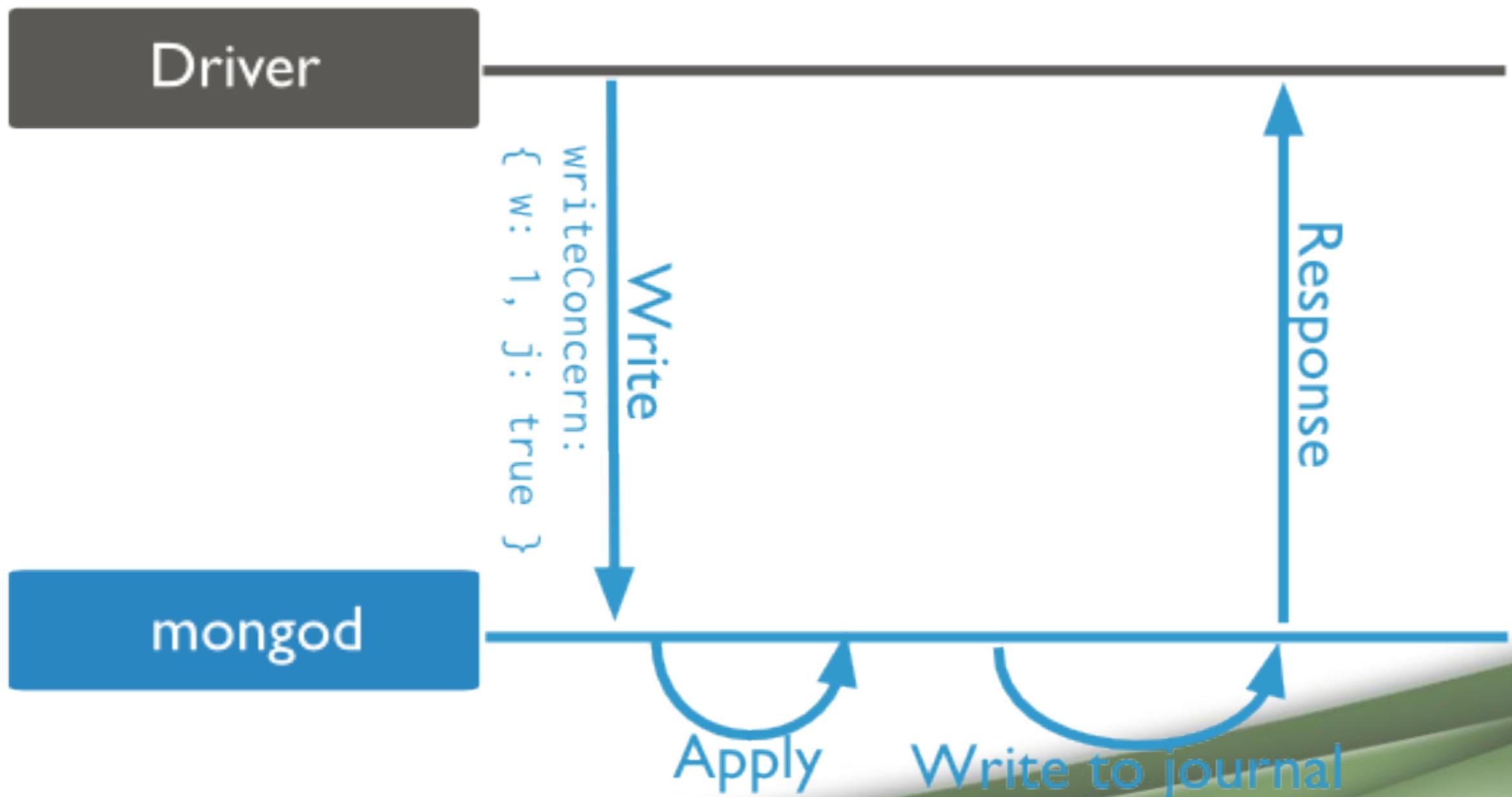
Unacknowledged



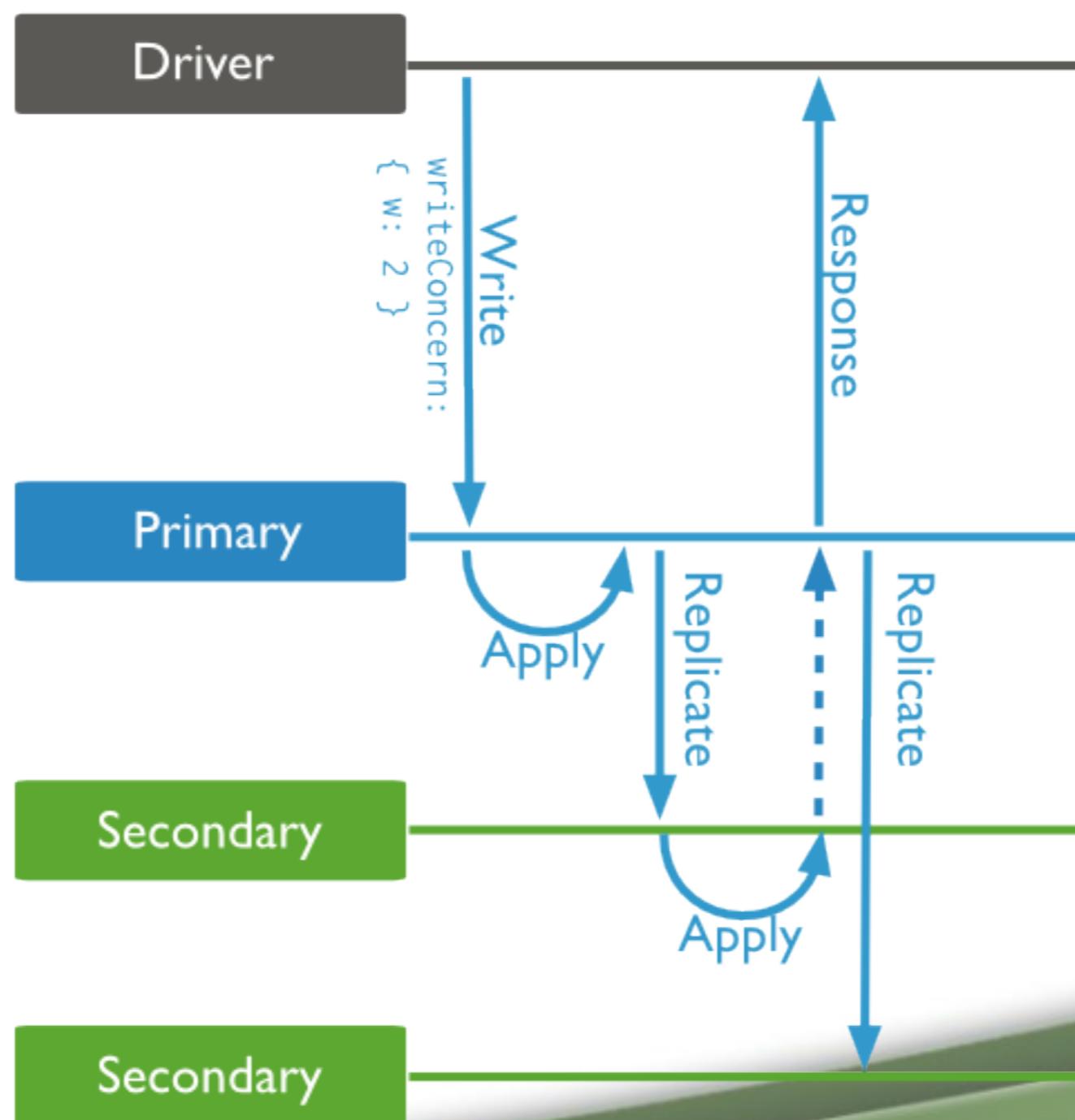
Acknowledged



Jounaled



Replica acknowledged



Благодаря!