СБЕР

# Spring Data R2DBC. I’m telling you the last time

Mikhail Polivakha

# Поливаха Михаил

- **Active OpenSource member (including Spring projects)**
- **Technical Writer**
- **Spring Айо Community board member**

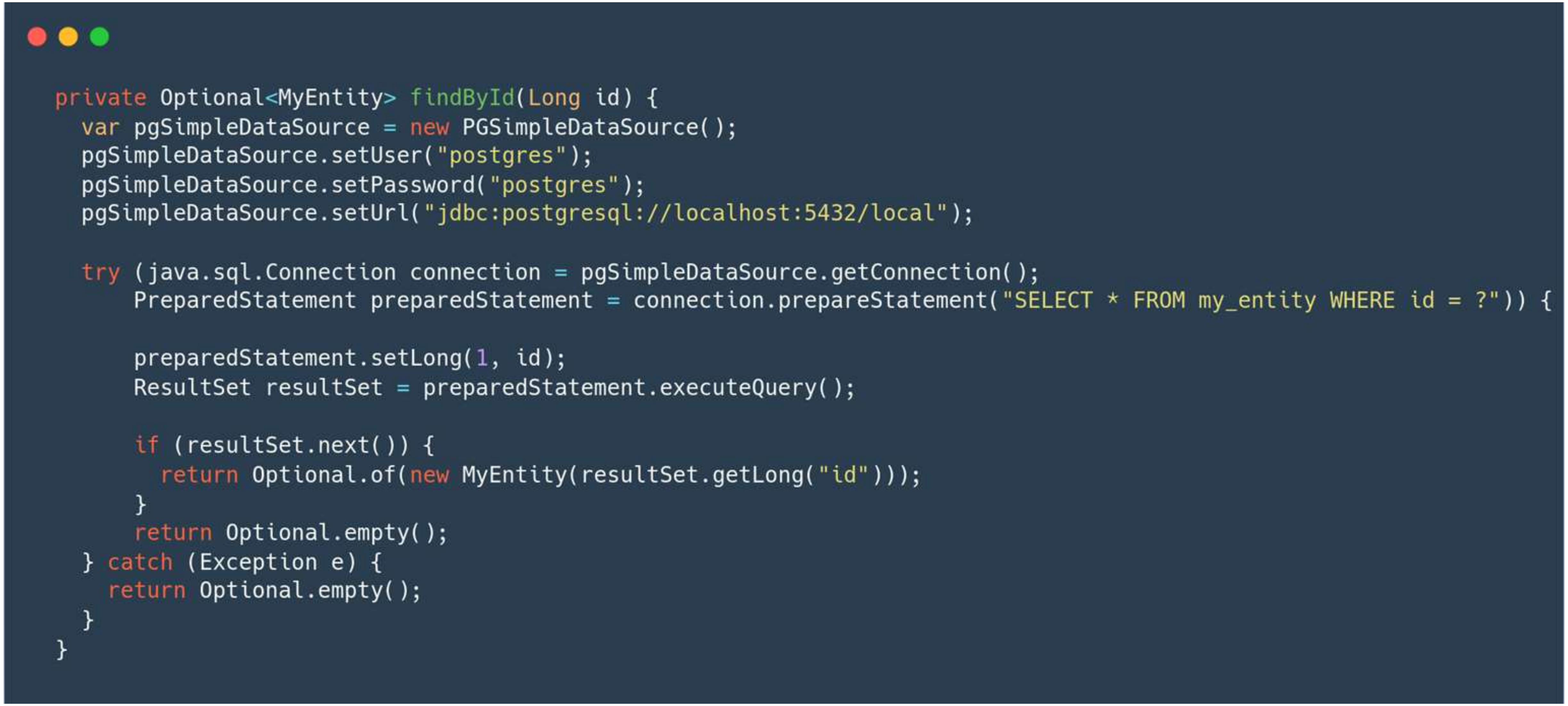
**Contacts:**

- **Telegram: @mipo256**
- **GitHub: mipo256**
- **Blog: mpolivaha.com**
- **Email: mikhailpolivakha@gmail.com**

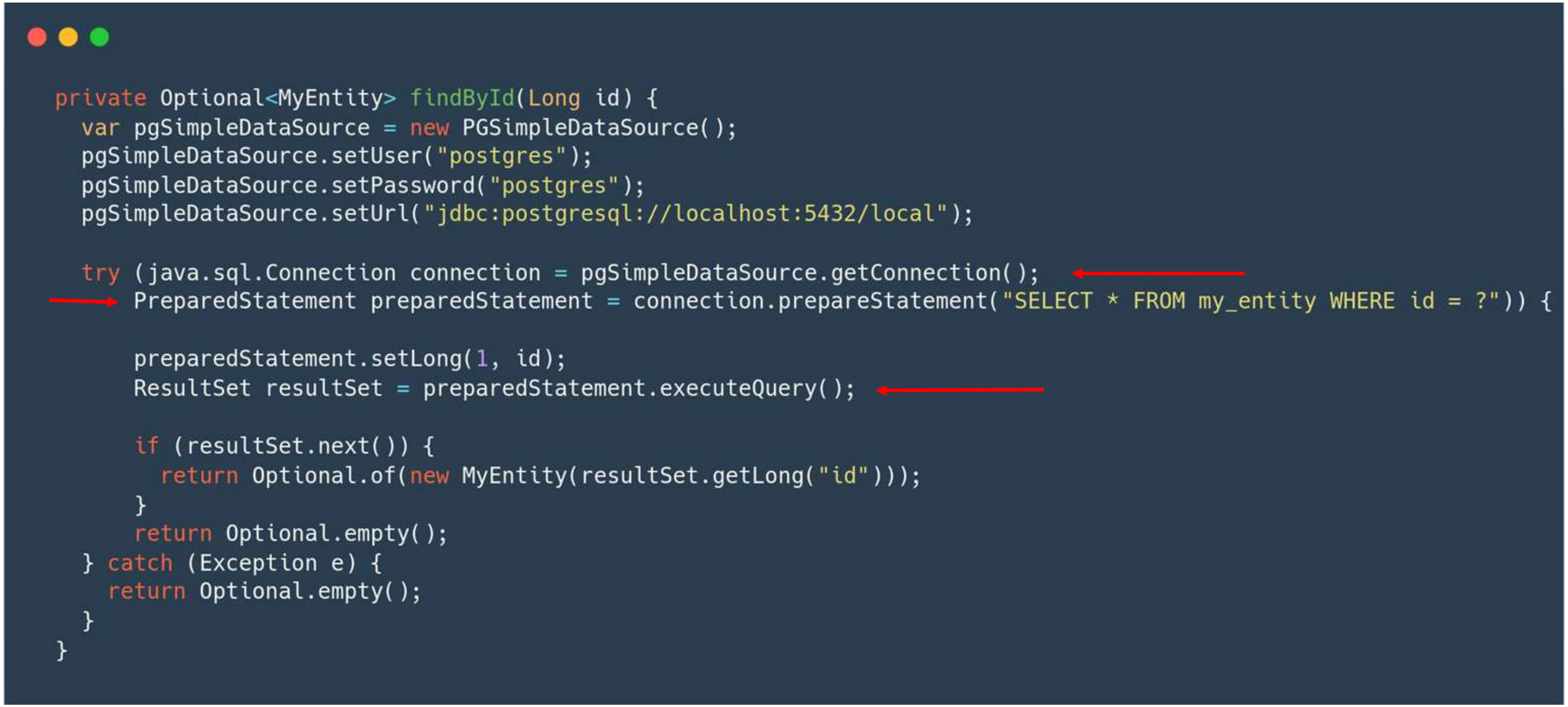
# Agenda

- Review some other attempts to work with RDBMS reactively
- How typical R2DBC drivers work
- Reactive != Asynchronous
- We’re also going to talk about spring-data-r2dbc and challenges it faces
- What should you do?

## Evolution of Reactive RDBMS drivers

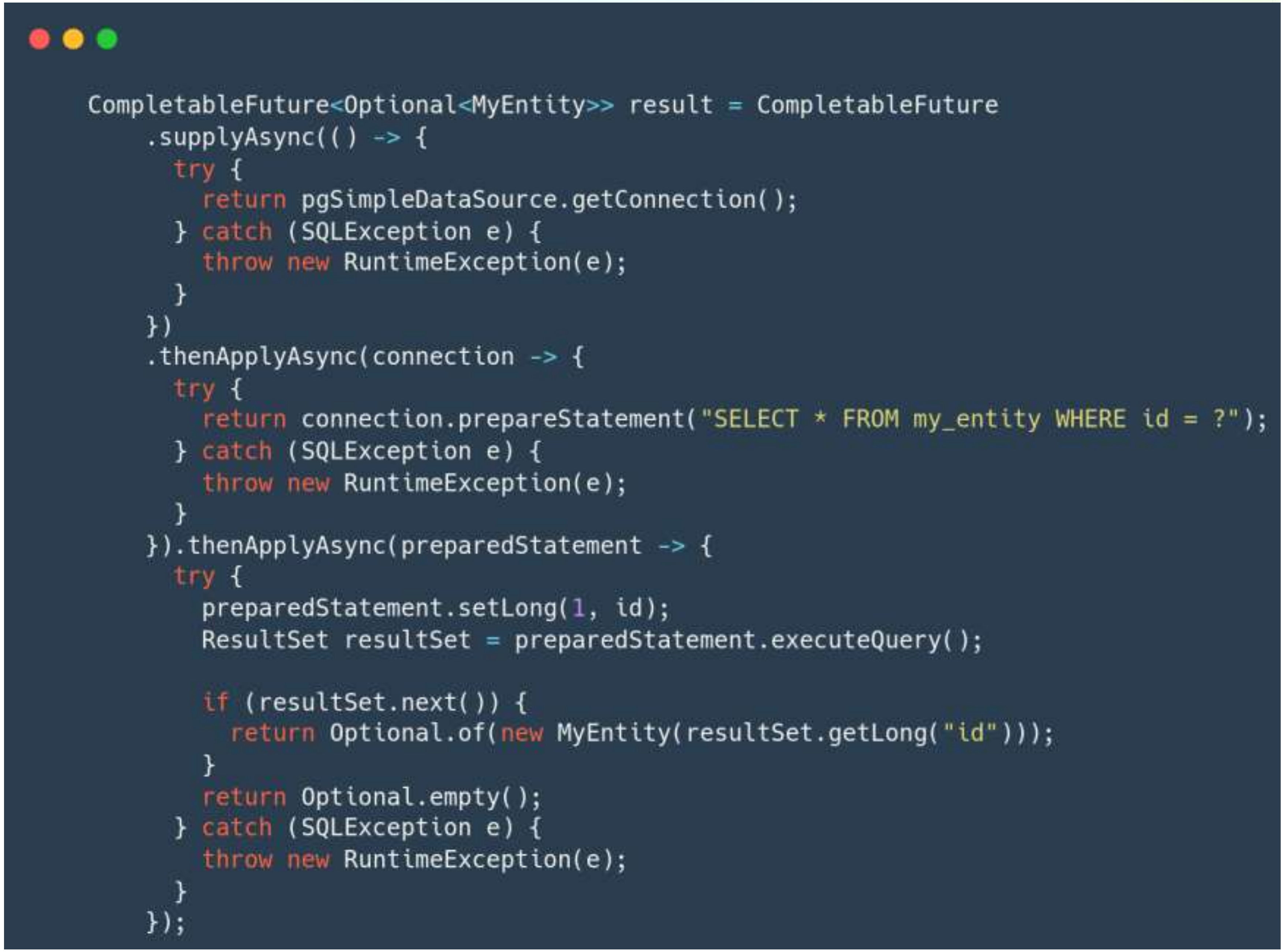


```java
private Optional<MyEntity> findById(Long id) {
  var pgSimpleDataSource = new PGSimpleDataSource();
  pgSimpleDataSource.setUser("postgres");
  pgSimpleDataSource.setPassword("postgres");
  pgSimpleDataSource.setUrl("jdbc:postgresql://localhost:5432/local");

  try (java.sql.Connection connection = pgSimpleDataSource.getConnection();
      PreparedStatement preparedStatement = connection.prepareStatement("SELECT * FROM my_entity WHERE id = ?")) {

      preparedStatement.setLong(1, id);
      ResultSet resultSet = preparedStatement.executeQuery();

      if (resultSet.next()) {
        return Optional.of(new MyEntity(resultSet.getLong("id")));
      }
      return Optional.empty();
  } catch (Exception e) {
    return Optional.empty();
  }
}
```



```java
private Optional<MyEntity> findById(Long id) {
  var pgSimpleDataSource = new PGSimpleDataSource();
  pgSimpleDataSource.setUser("postgres");
  pgSimpleDataSource.setPassword("postgres");
  pgSimpleDataSource.setUrl("jdbc:postgresql://localhost:5432/local");

  try (java.sql.Connection connection = pgSimpleDataSource.getConnection();
      PreparedStatement preparedStatement = connection.prepareStatement("SELECT * FROM my_entity WHERE id = ?")) {

      preparedStatement.setLong(1, id);
      ResultSet resultSet = preparedStatement.executeQuery();

      if (resultSet.next()) {
        return Optional.of(new MyEntity(resultSet.getLong("id")));
      }
      return Optional.empty();
  } catch (Exception e) {
    return Optional.empty();
  }
}
```

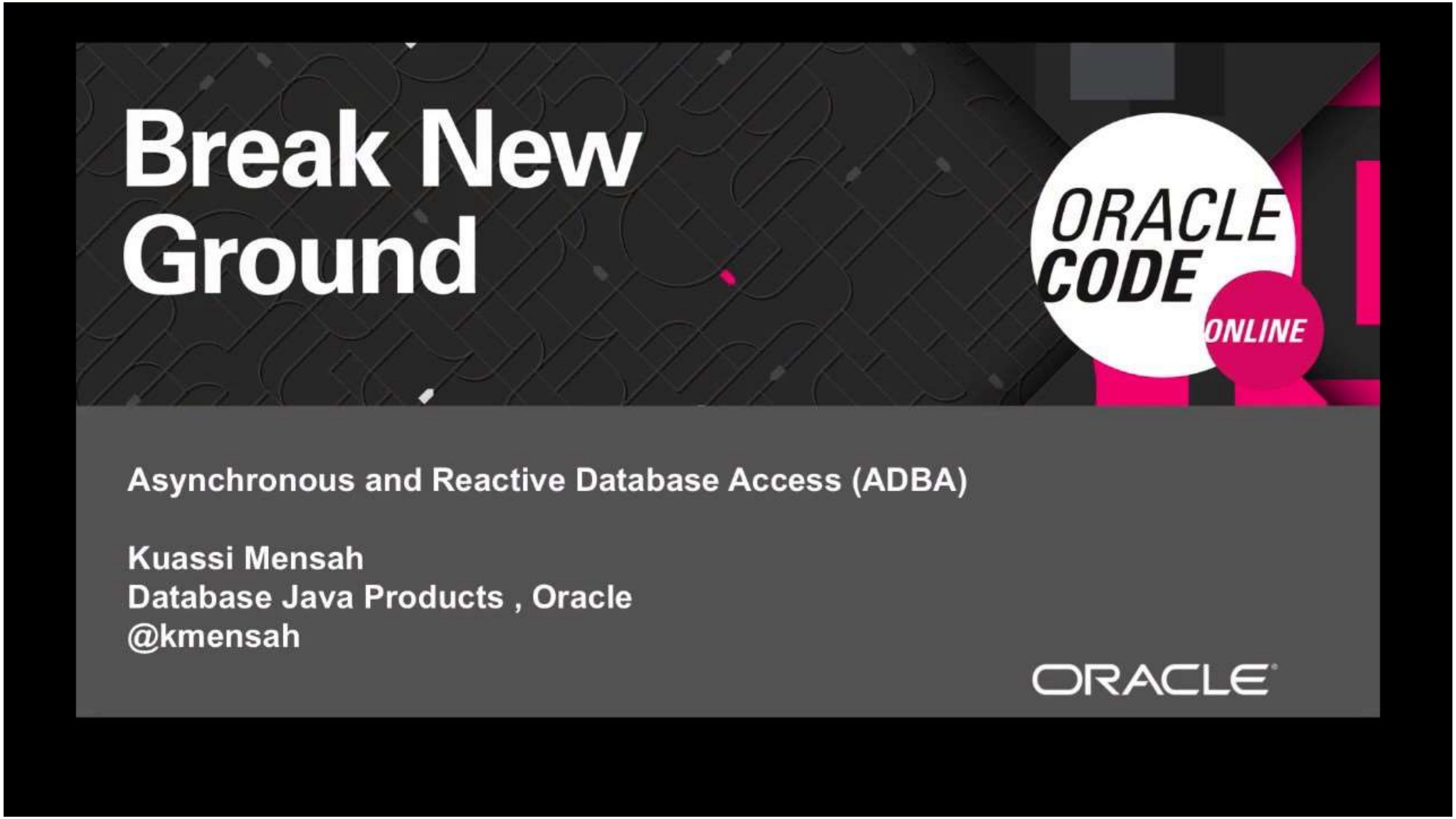
## An easy fix!



```
CompletableFuture<Optional<MyEntity>> result = CompletableFuture
    .supplyAsync(() -> {
      try {
        return pgSimpleDataSource.getConnection();
      } catch (SQLException e) {
        throw new RuntimeException(e);
      }
    })
    .thenApplyAsync(connection -> {
      try {
        return connection.prepareStatement("SELECT * FROM my_entity WHERE id = ?");
      } catch (SQLException e) {
        throw new RuntimeException(e);
      }
    }).thenApplyAsync(preparedStatement -> {
      try {
        preparedStatement.setLong(1, id);
        ResultSet resultSet = preparedStatement.executeQuery();

        if (resultSet.next()) {
          return Optional.of(new MyEntity(resultSet.getLong("id")));
        }
        return Optional.empty();
      } catch (SQLException e) {
        throw new RuntimeException(e);
      }
    });
```

Just use the Java's Future, why always overcomplicate?

## ADBA - Asynchronous Database Access



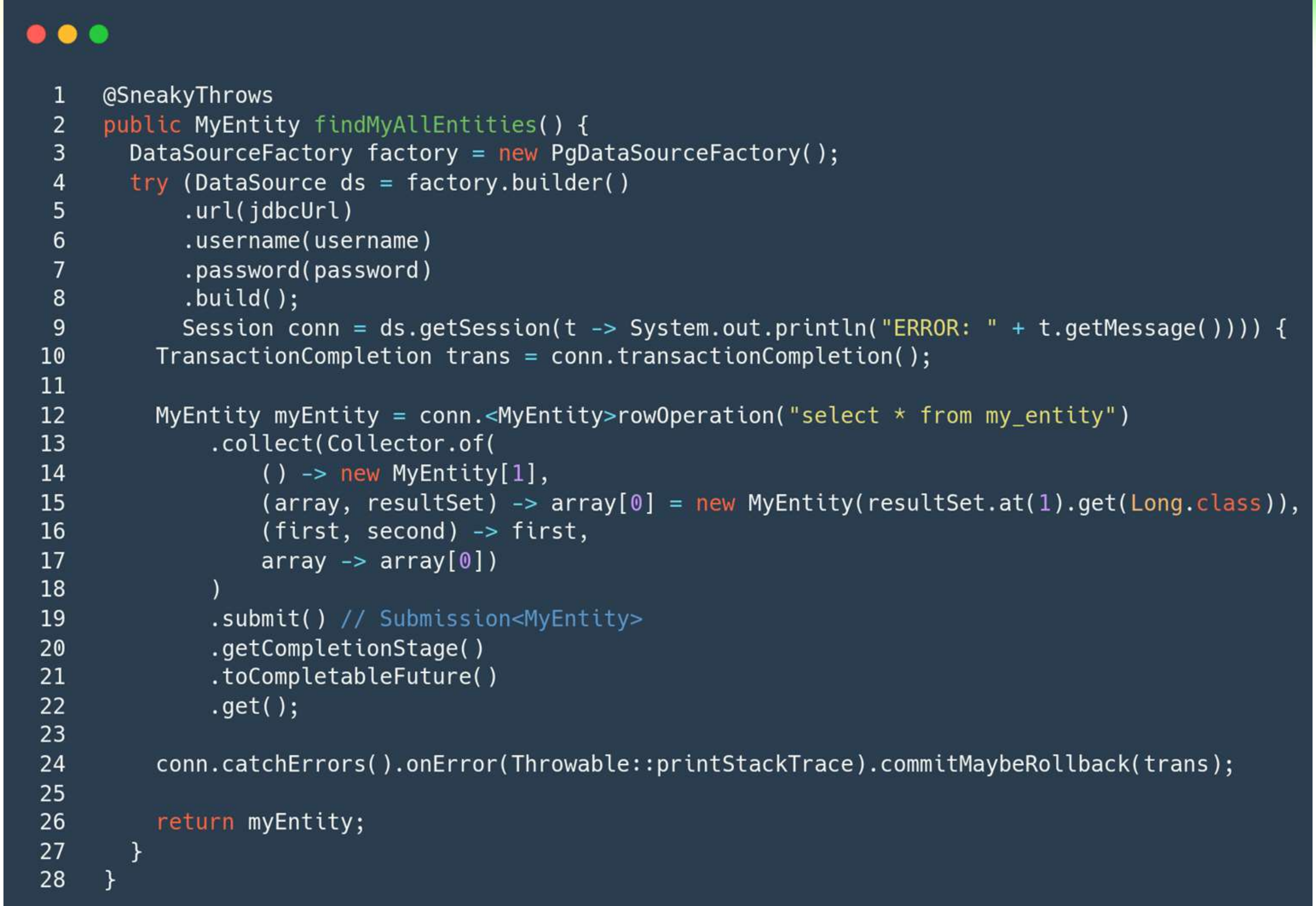
Break New Ground

ORACLE CODE ONLINE

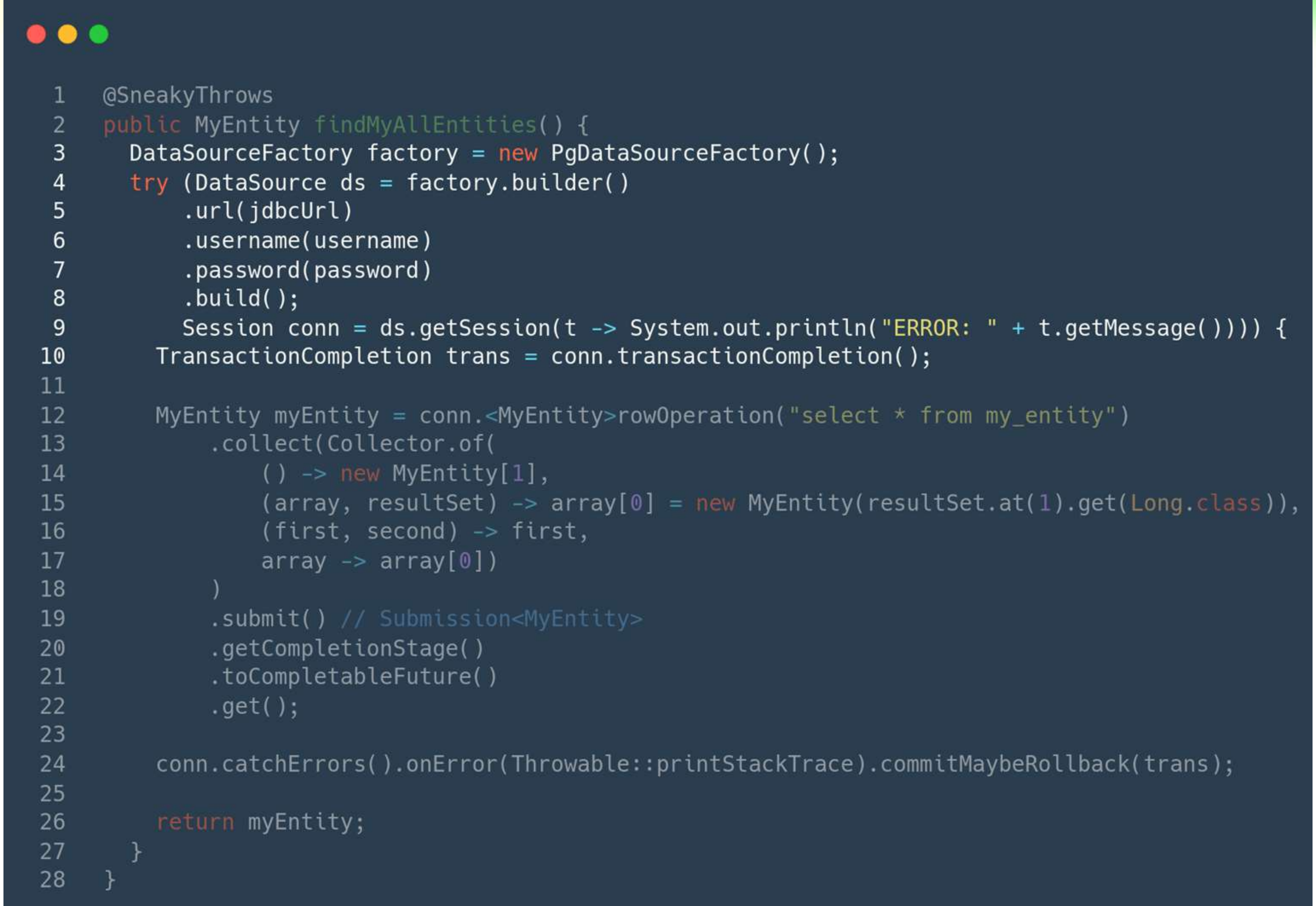
Asynchronous and Reactive Database Access (ADBA)

Kuassi Mensah
Database Java Products , Oracle
@kmensah

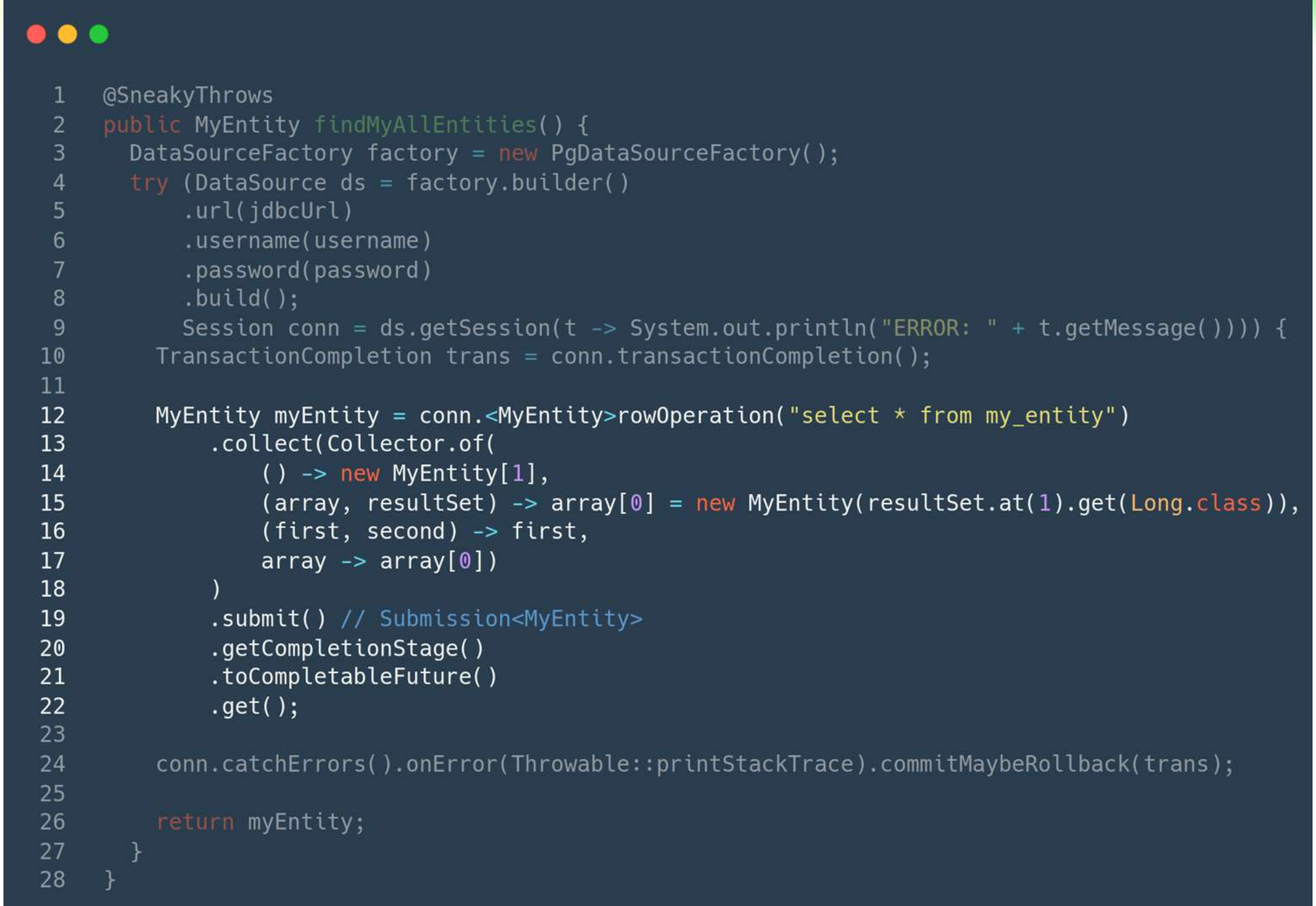
ORACLE



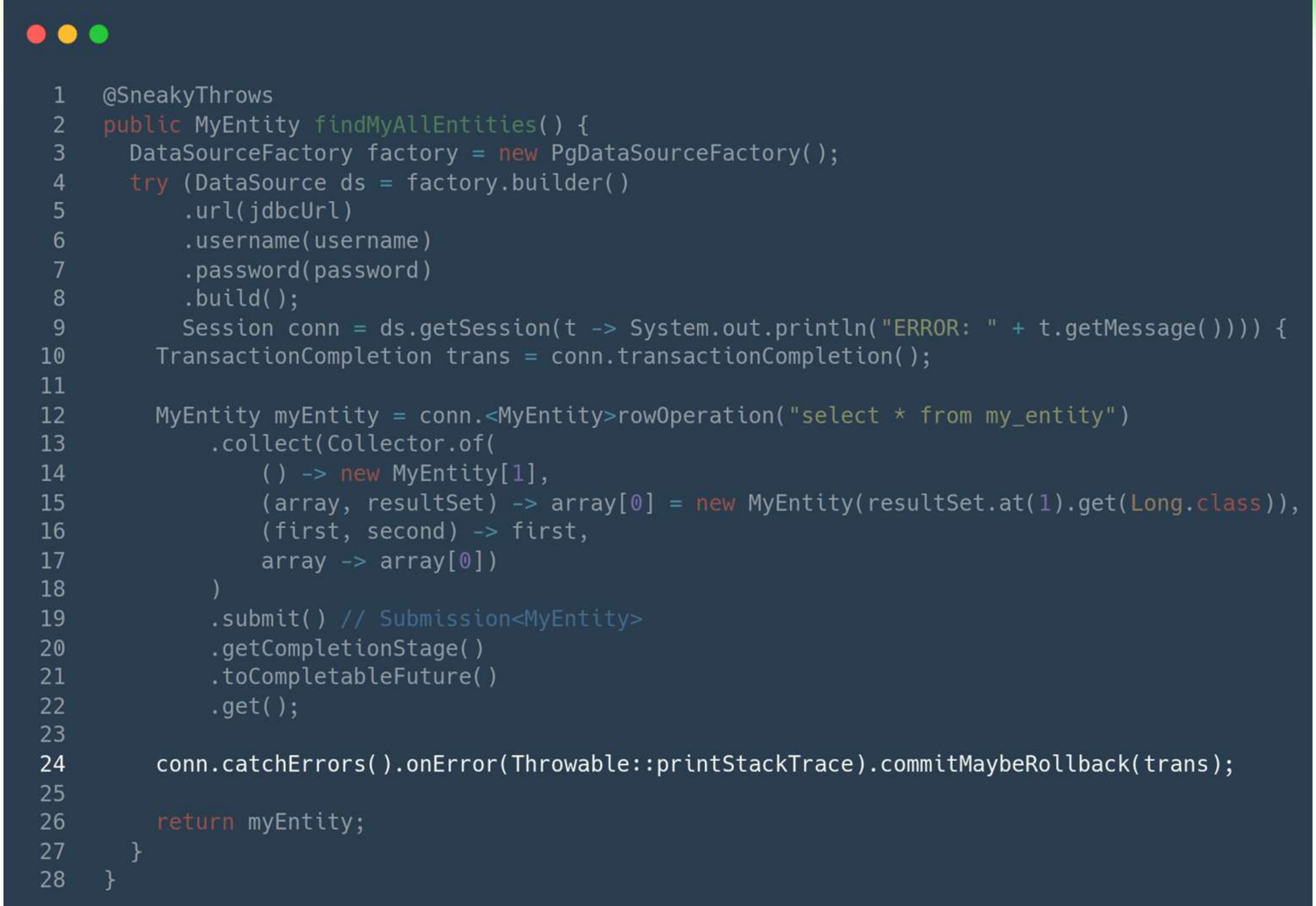
```java
@SneakyThrows
public MyEntity findMyAllEntities() {
  DataSourceFactory factory = new PgDataSourceFactory();
  try (DataSource ds = factory.builder()
        .url(jdbcUrl)
        .username(username)
        .password(password)
        .build();
        Session conn = ds.getSession(t -> System.out.println("ERROR: " + t.getMessage()))) {
    TransactionCompletion trans = conn.transactionCompletion();

    MyEntity myEntity = conn.<MyEntity>rowOperation("select * from my_entity")
        .collect(Collector.of(
            () -> new MyEntity[1],
            (array, resultSet) -> array[0] = new MyEntity(resultSet.at(1).get(Long.class)),
            (first, second) -> first,
            array -> array[0])
        )
        .submit() // Submission<MyEntity>
        .getCompletionStage()
        .toCompletableFuture()
        .get();

    conn.catchErrors().onError(Throwable::printStackTrace).commitMaybeRollback(trans);

    return myEntity;
  }
}
```



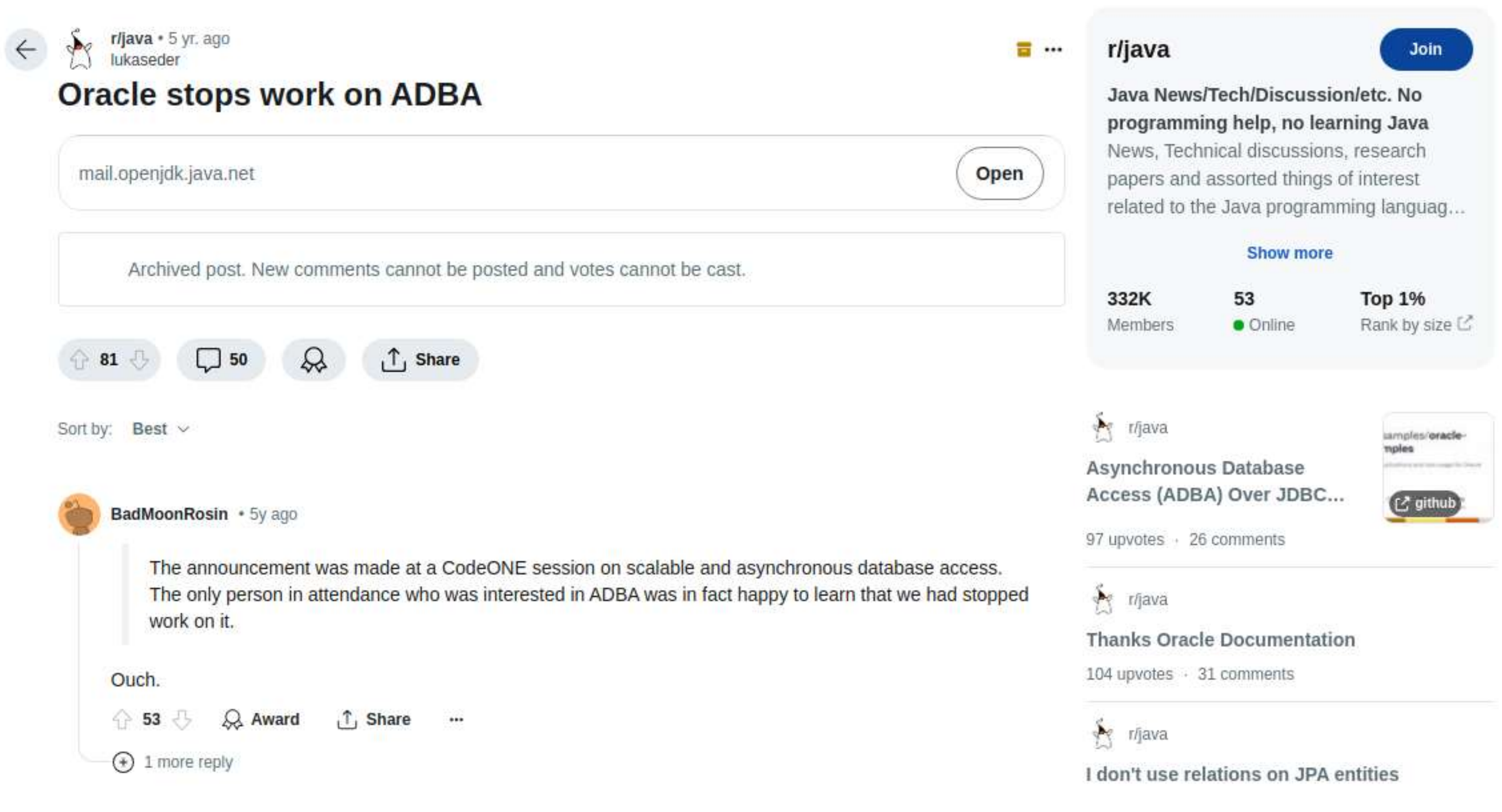
```java
@SneakyThrows
public MyEntity findMyAllEntities() {
  DataSourceFactory factory = new PgDataSourceFactory();
  try (DataSource ds = factory.builder()
      .url(jdbcUrl)
      .username(username)
      .password(password)
      .build();
      Session conn = ds.getSession(t -> System.out.println("ERROR: " + t.getMessage()))) {
    TransactionCompletion trans = conn.transactionCompletion();

    MyEntity myEntity = conn.<MyEntity>rowOperation("select * from my_entity")
        .collect(Collector.of(
            () -> new MyEntity[1],
            (array, resultSet) -> array[0] = new MyEntity(resultSet.at(1).get(Long.class)),
            (first, second) -> first,
            array -> array[0])
        )
        .submit() // Submission<MyEntity>
        .getCompletionStage()
        .toCompletableFuture()
        .get();

    conn.catchErrors().onError(Throwable::printStackTrace).commitMaybeRollback(trans);

    return myEntity;
  }
}
```



```java
@SneakyThrows
public MyEntity findMyAllEntities() {
  DataSourceFactory factory = new PgDataSourceFactory();
  try (DataSource ds = factory.builder()
      .url(jdbcUrl)
      .username(username)
      .password(password)
      .build();
      Session conn = ds.getSession(t -> System.out.println("ERROR: " + t.getMessage()))) {
    TransactionCompletion trans = conn.transactionCompletion();

    MyEntity myEntity = conn.<MyEntity>rowOperation("select * from my_entity")
        .collect(Collector.of(
            () -> new MyEntity[1],
            (array, resultSet) -> array[0] = new MyEntity(resultSet.at(1).get(Long.class)),
            (first, second) -> first,
            array -> array[0])
        )
        .submit() // Submission<MyEntity>
        .getCompletionStage()
        .toCompletableFuture()
        .get();

    conn.catchErrors().onError(Throwable::printStackTrace).commitMaybeRollback(trans);

    return myEntity;
  }
}
```



```java
@SneakyThrows
public MyEntity findMyAllEntities() {
  DataSourceFactory factory = new PgDataSourceFactory();
  try (DataSource ds = factory.builder()
      .url(jdbcUrl)
      .username(username)
      .password(password)
      .build();
      Session conn = ds.getSession(t -> System.out.println("ERROR: " + t.getMessage()))) {
    TransactionCompletion trans = conn.transactionCompletion();

    MyEntity myEntity = conn.<MyEntity>rowOperation("select * from my_entity")
        .collect(Collector.of(
            () -> new MyEntity[1],
            (array, resultSet) -> array[0] = new MyEntity(resultSet.at(1).get(Long.class)),
            (first, second) -> first,
            array -> array[0])
        )
        .submit() // Submission<MyEntity>
        .getCompletionStage()
        .toCompletableFuture()
        .get();

    conn.catchErrors().onError(Throwable::printStackTrace).commitMaybeRollback(trans);

    return myEntity;
  }
}
```



r/java • 5 yr. ago
lukaseder

Oracle stops work on ADBA

mail.openjdk.java.net Open

Archived post. New comments cannot be posted and votes cannot be cast.

81 · 50 · Share

Sort by: Best

BadMoonRosin • 5y ago

> The announcement was made at a CodeONE session on scalable and asynchronous database access. The only person in attendance who was interested in ADBA was in fact happy to learn that we had stopped work on it.

Ouch.

53 · Award · Share · ···

1 more reply

r/java Join

**Java News/Tech/Discussion/etc. No programming help, no learning Java**
News, Technical discussions, research papers and assorted things of interest related to the Java programming languag...

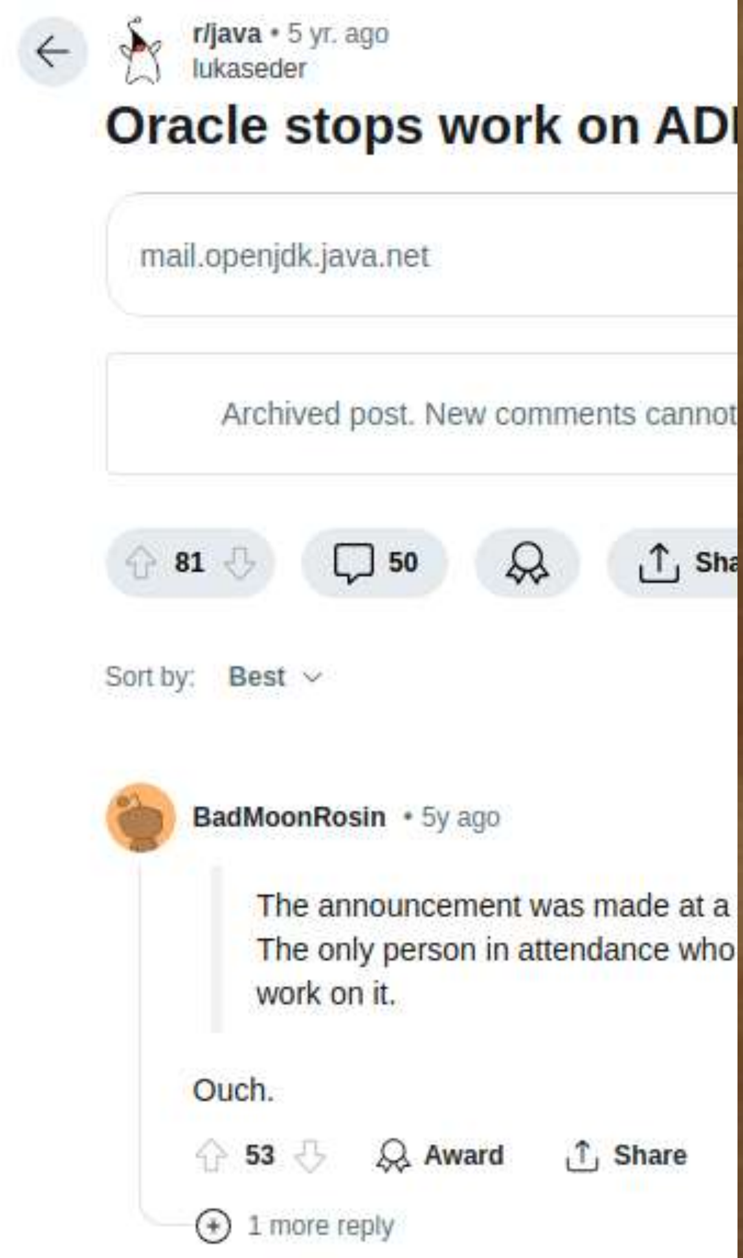
Show more

| 332K | 53 | Top 1% |
|---|---|---|
| Members | ● Online | Rank by size |

r/java
Asynchronous Database Access (ADBA) Over JDBC...
97 upvotes · 26 comments

r/java
Thanks Oracle Documentation
104 upvotes · 31 comments

r/java
I don't use relations on JPA entities



r/java • 5 yr. ago
lukaseder

Oracle stops work on ADI

mail.openjdk.java.net

Archived post. New comments cannot

81 50 Sha

Sort by: Best

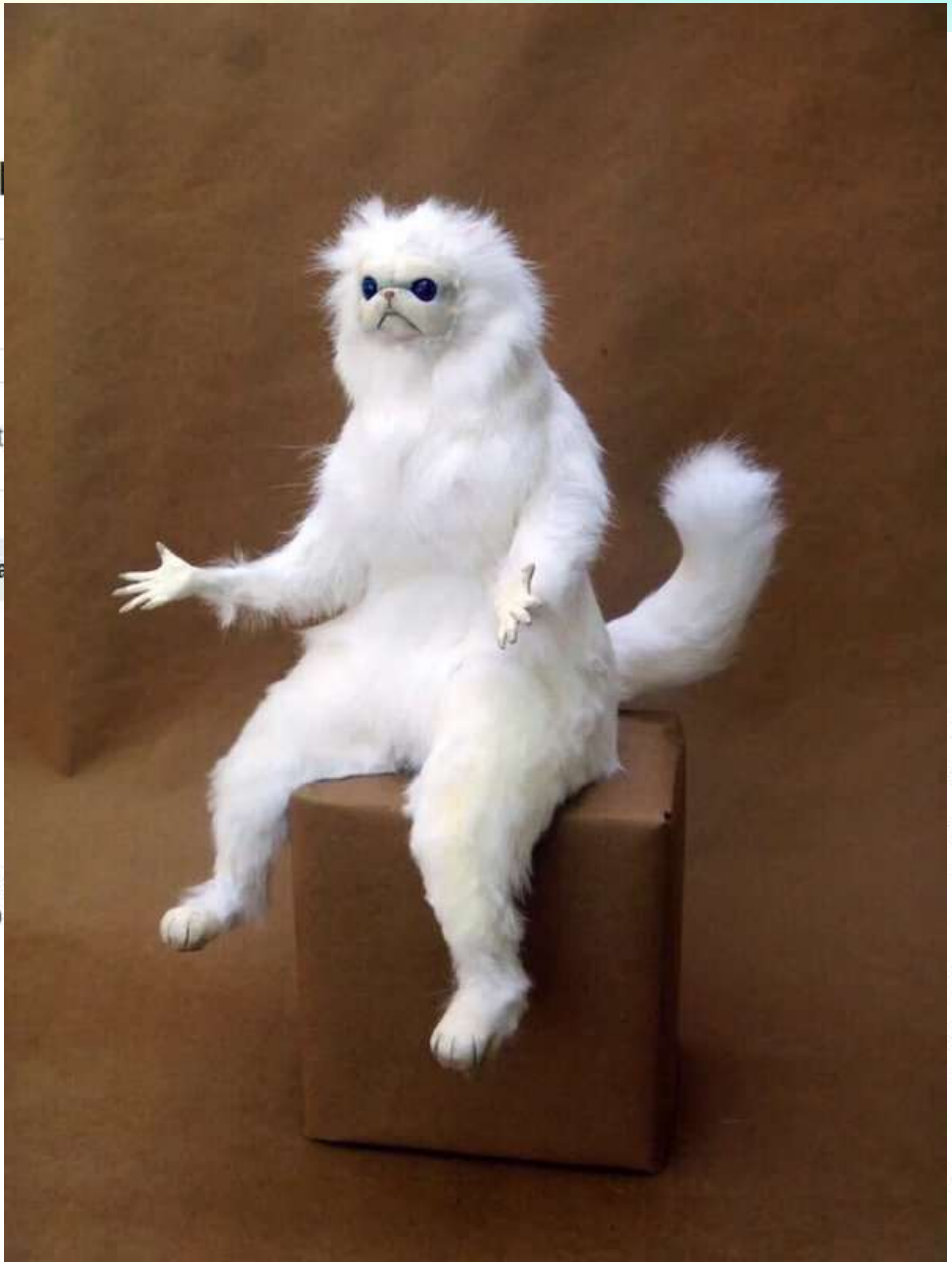
BadMoonRosin • 5y ago

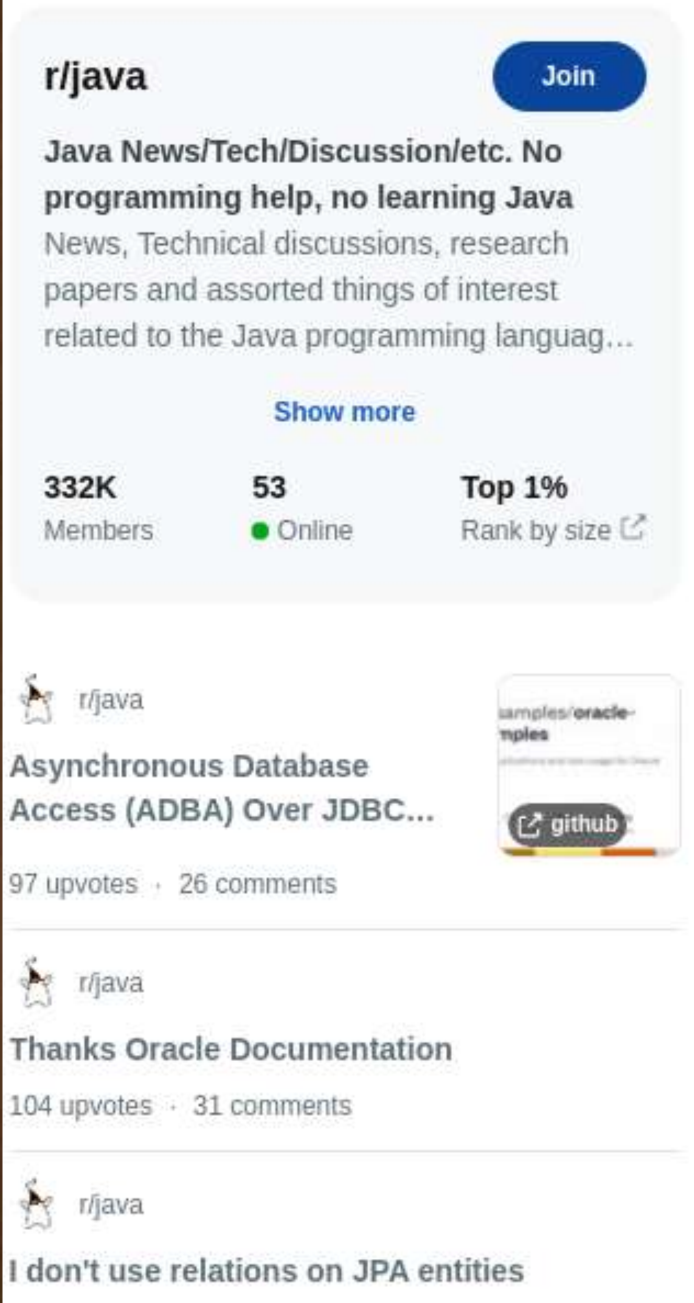
> The announcement was made at a
> The only person in attendance who
> work on it.

Ouch.

53 Award Share

1 more reply





r/java

Join

**Java News/Tech/Discussion/etc. No programming help, no learning Java**

News, Technical discussions, research papers and assorted things of interest related to the Java programming languag...

Show more

332K Members

53 Online

Top 1% Rank by size

r/java

Asynchronous Database Access (ADBA) Over JDBC...

github

97 upvotes · 26 comments

r/java

Thanks Oracle Documentation

104 upvotes · 31 comments

r/java

I don't use relations on JPA entities

## What happened to ADBA?[1]

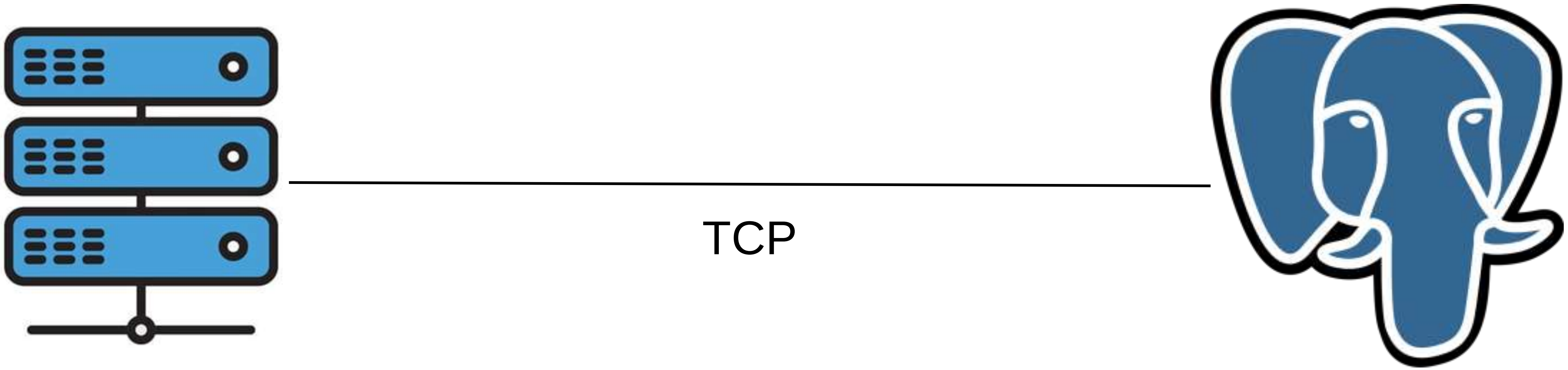
1. Project Loom Firbes solve a lot of problems that reactivity is targeting, *but not all.*

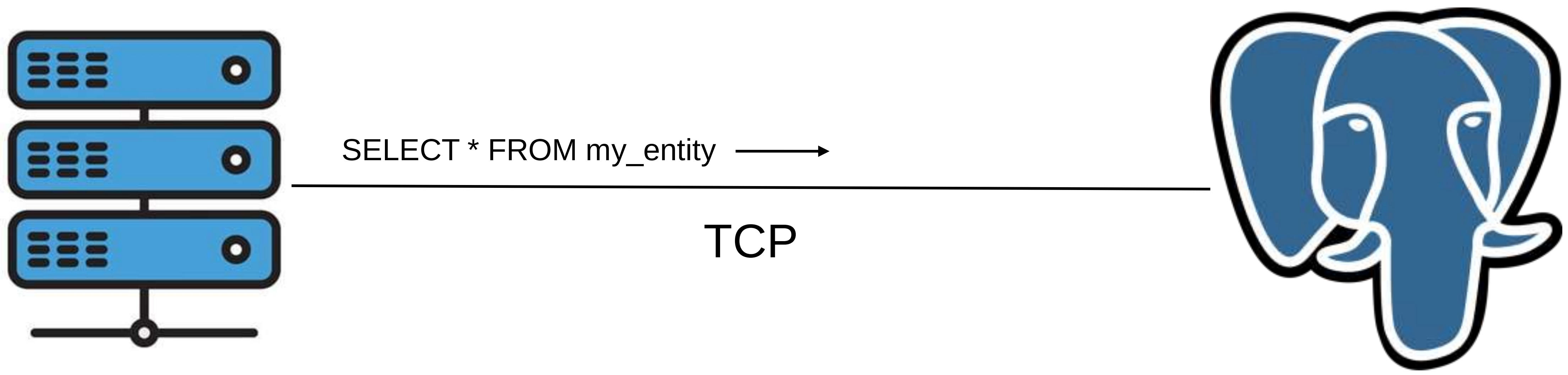
2. The cost for writing reactive applications is harder to “test, debug, maintain and understand”.

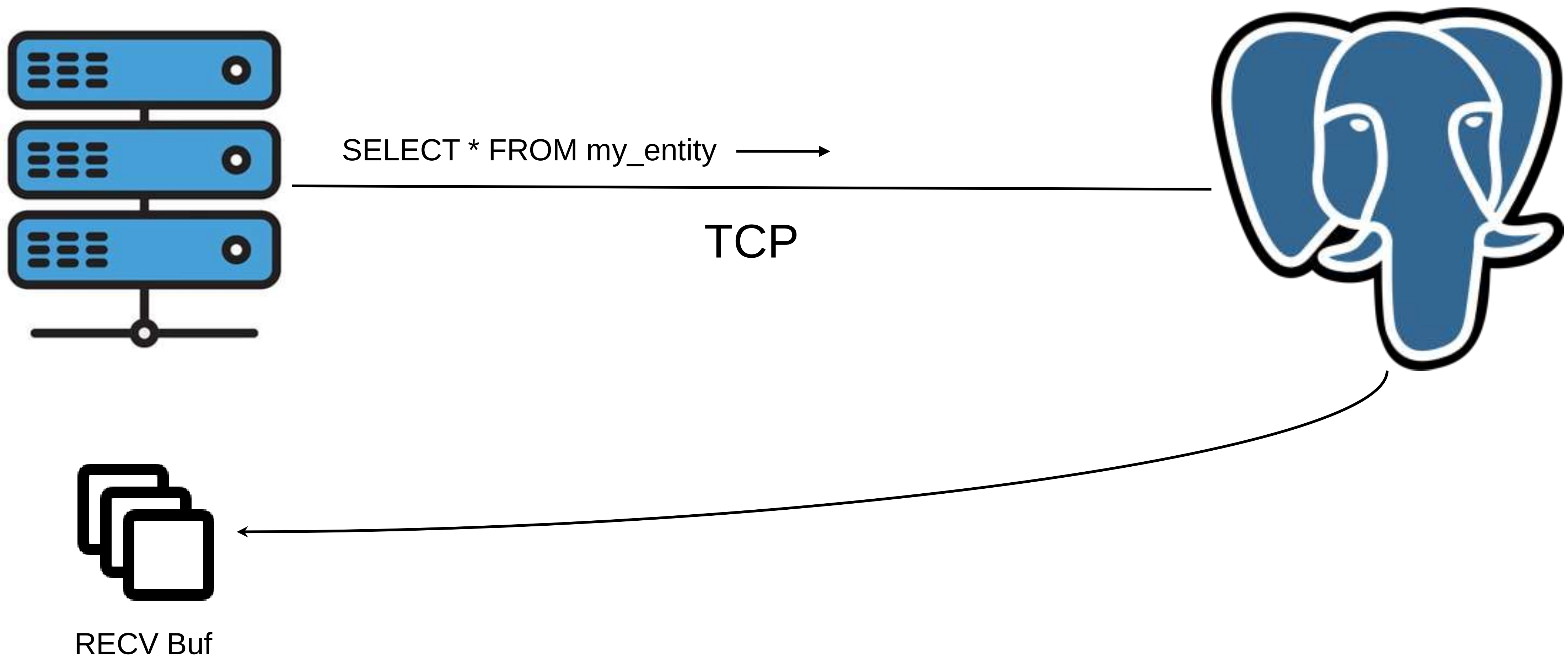
3. Considering the trade-offs, that just not worth it.

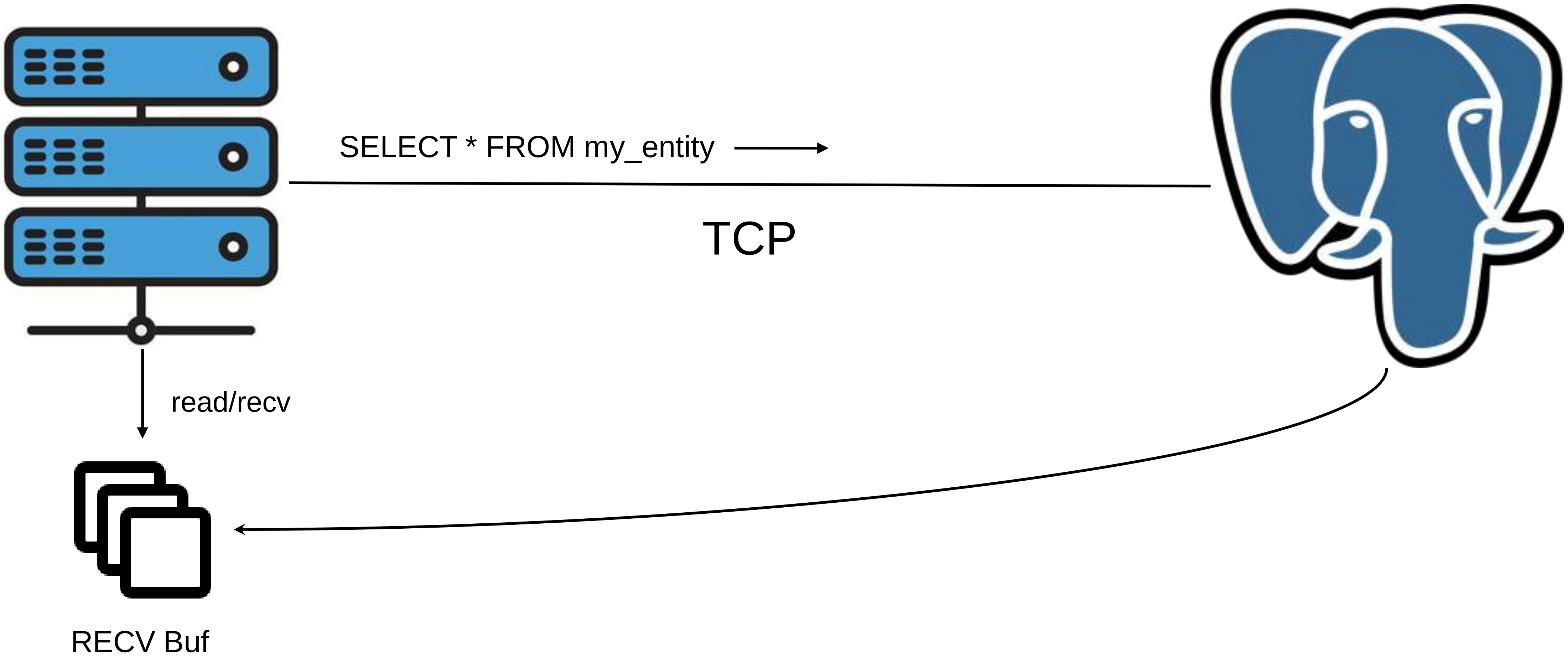
[1]https://mail.openjdk.org/pipermail/jdbc-spec-discuss/2019-September/000529.html

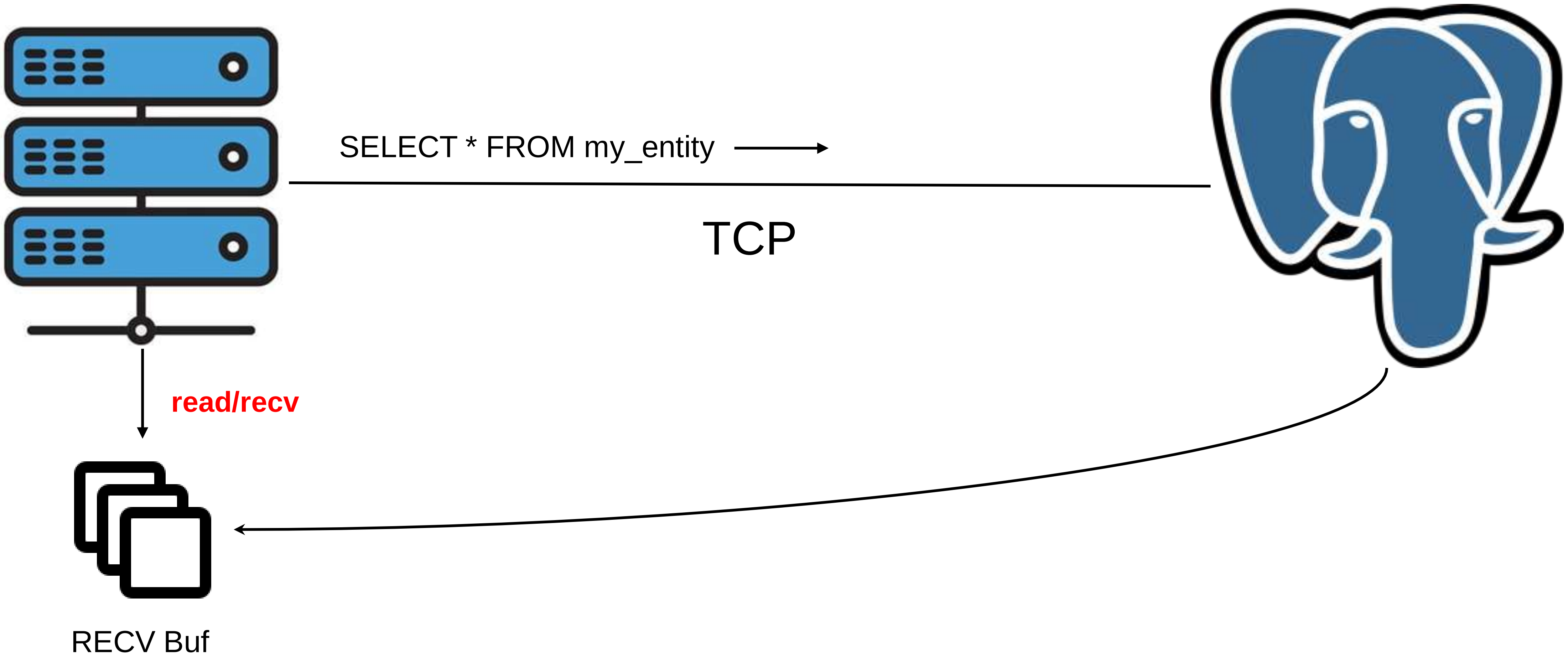
## Reactive APIs for Databases

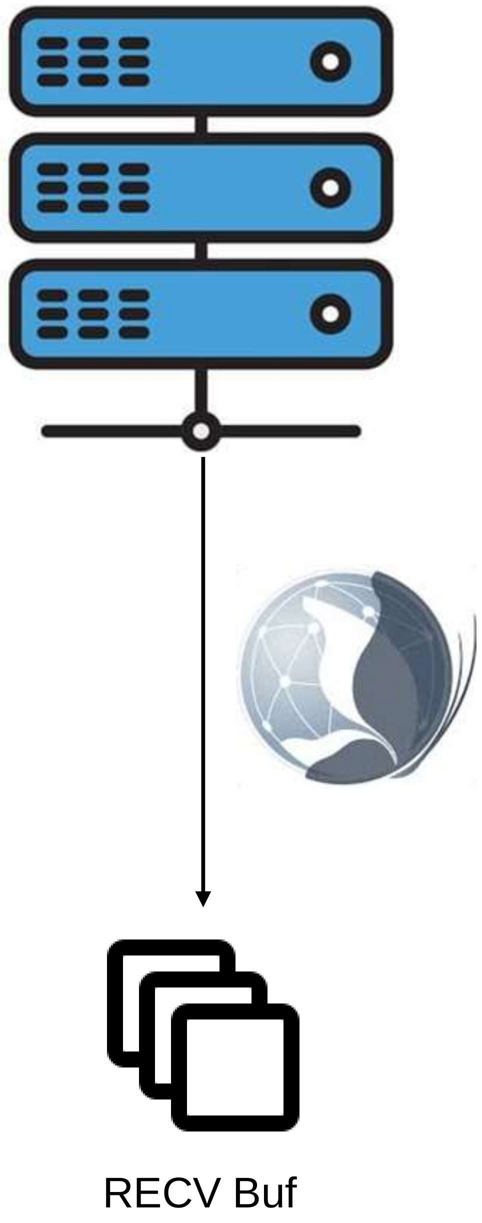

TCP


SELECT * FROM my_entity
TCP

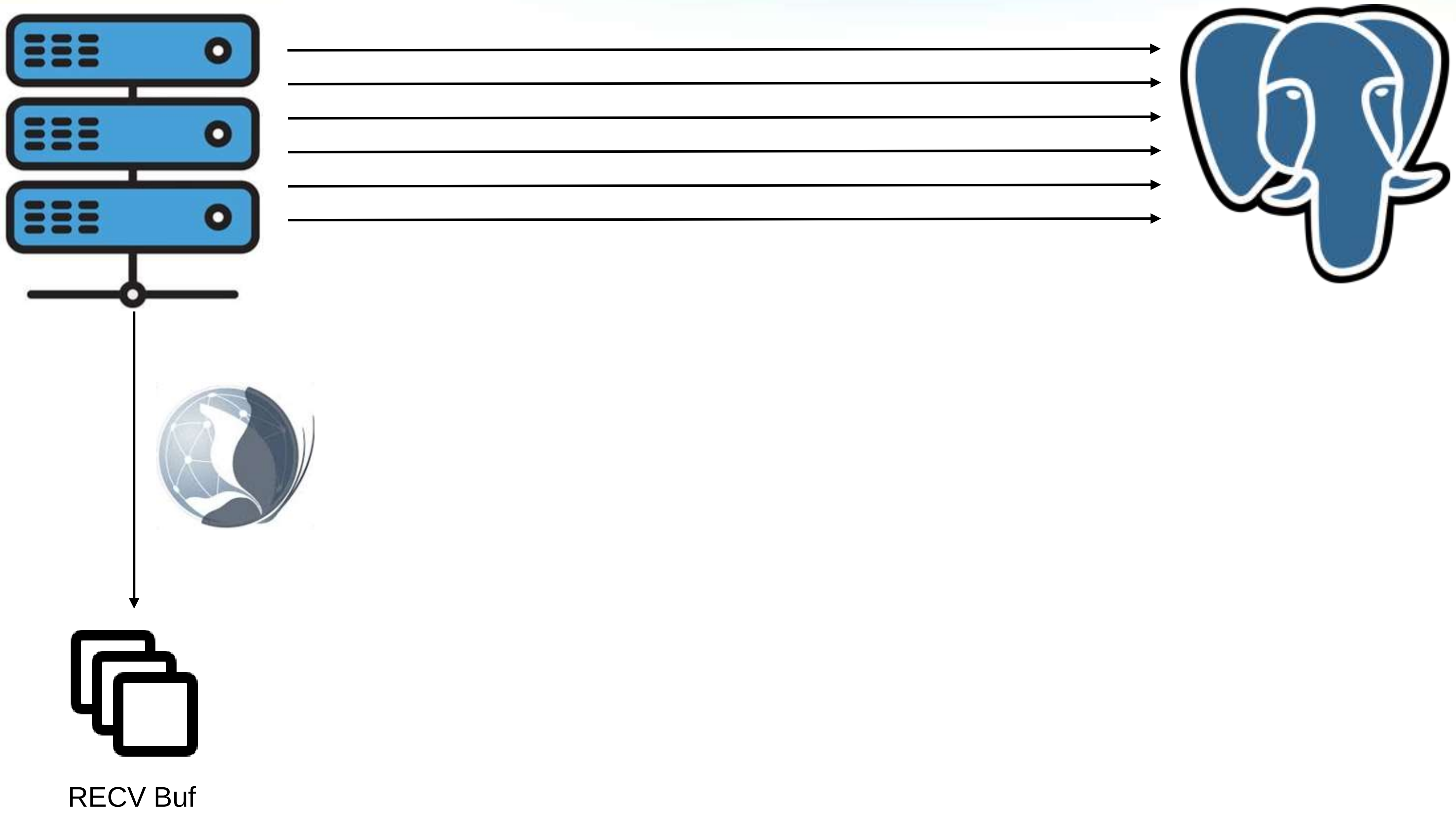

SELECT * FROM my_entity
TCP
RECV Buf

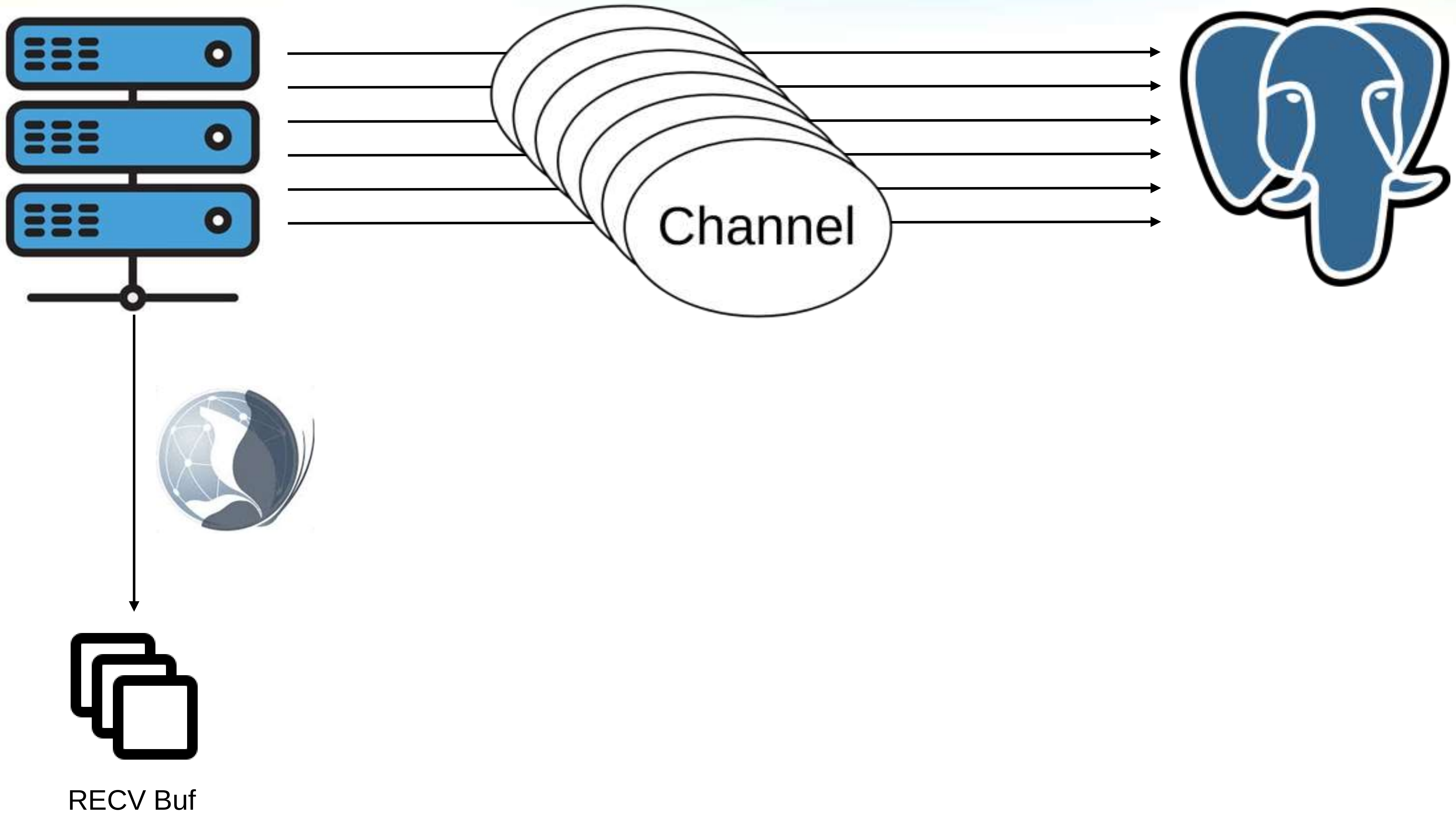

SELECT * FROM my_entity
TCP
read/recv
RECV Buf

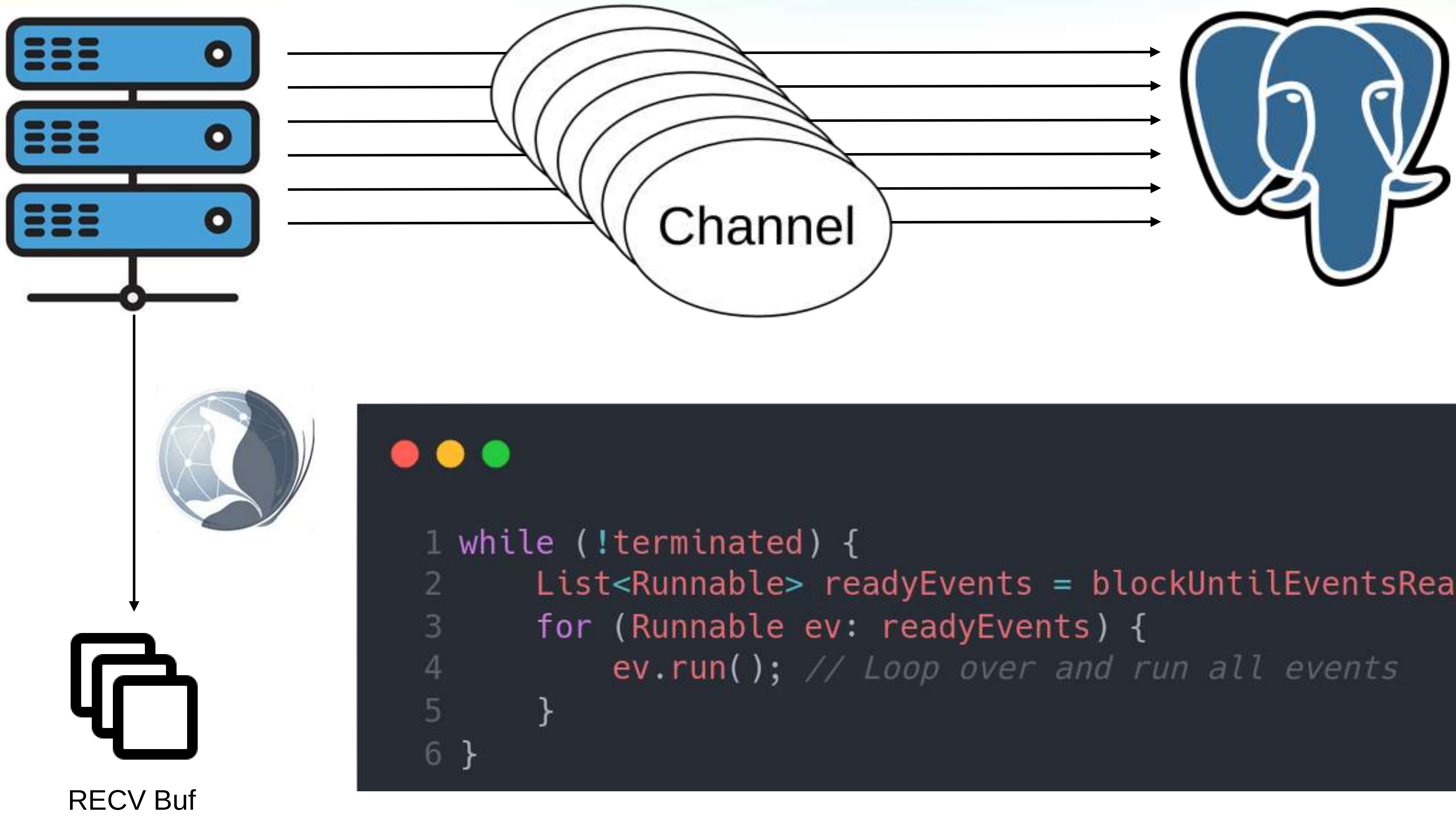

SELECT * FROM my_entity
TCP
read/recv
RECV Buf

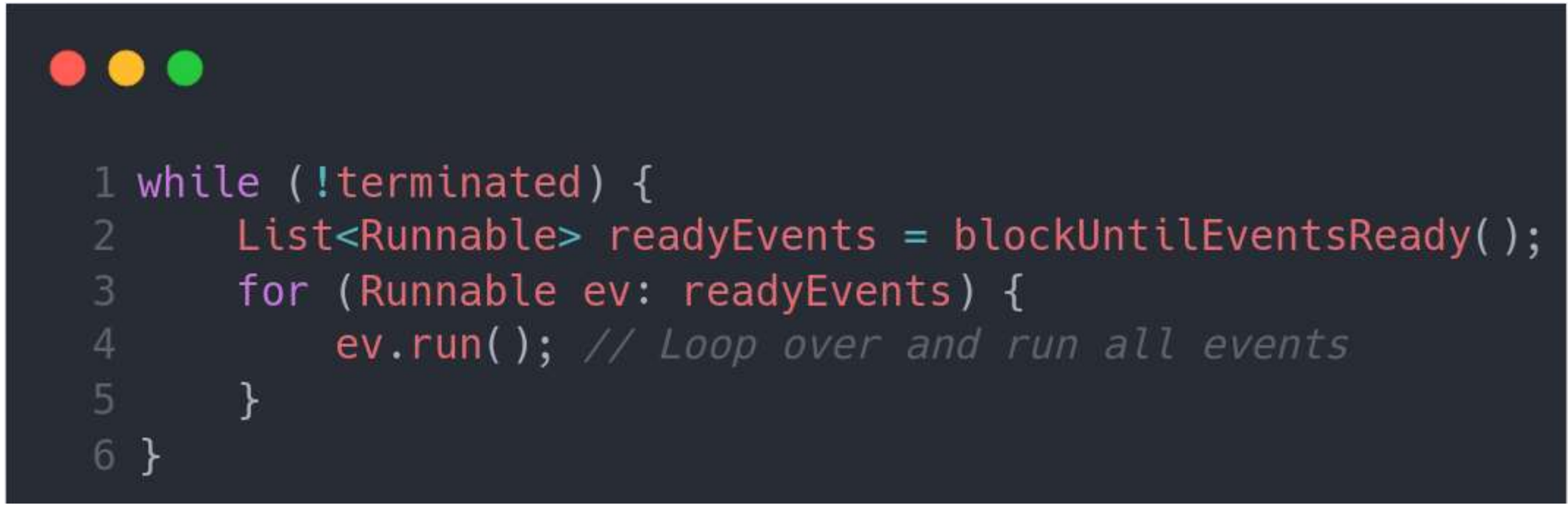

RECV Buf

RECV Buf

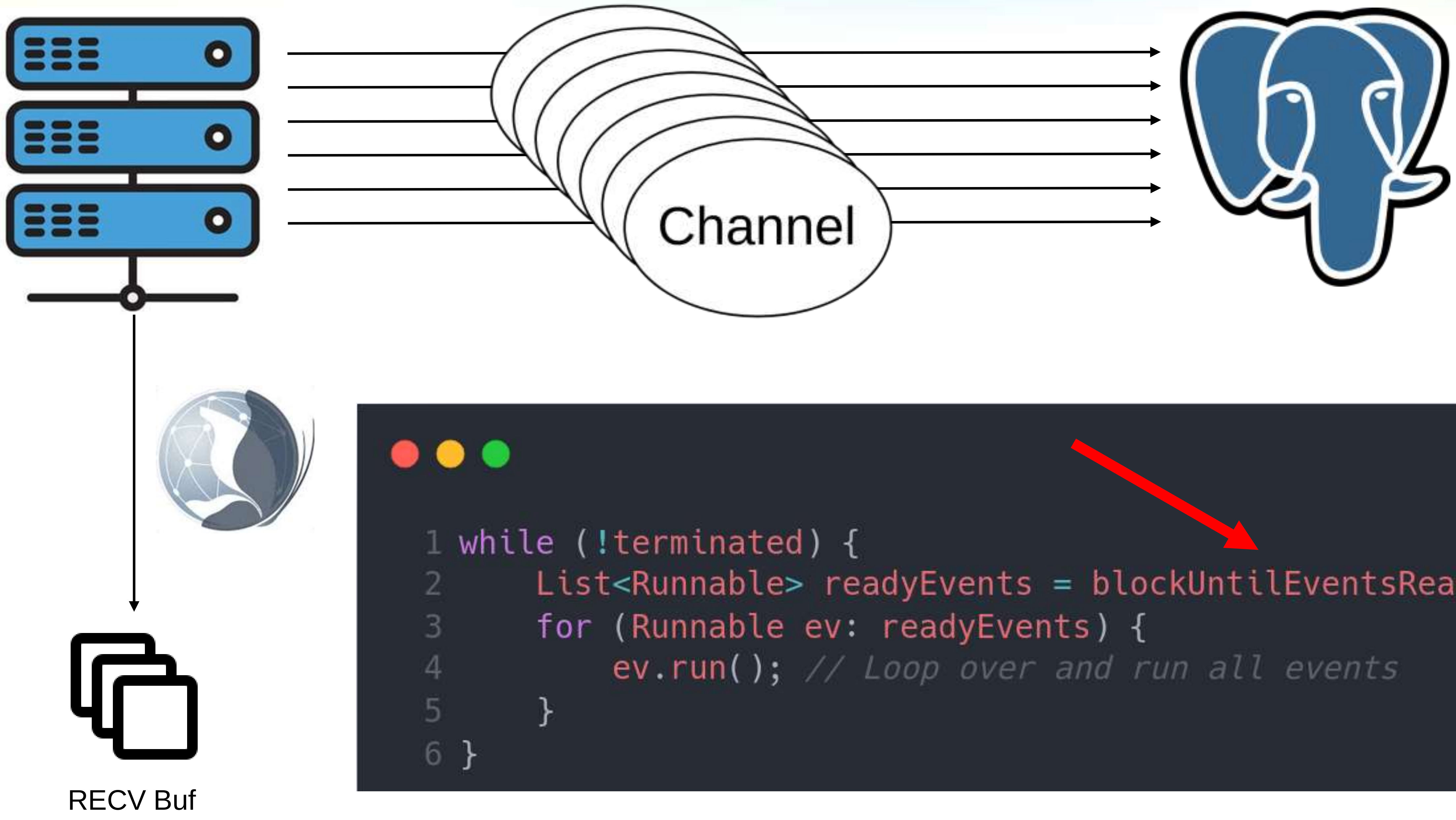

RECV Buf


Channel
RECV Buf





```
while (!terminated) {
    List<Runnable> readyEvents = blockUntilEventsReady();
    for (Runnable ev: readyEvents) {
        ev.run(); // Loop over and run all events
    }
}
```



```
while (!terminated) {
    List<Runnable> readyEvents = blockUntilEventsReady();
    for (Runnable ev: readyEvents) {
        ev.run(); // Loop over and run all events
    }
}
```

## Blocking EventLoop Thread is not fatal

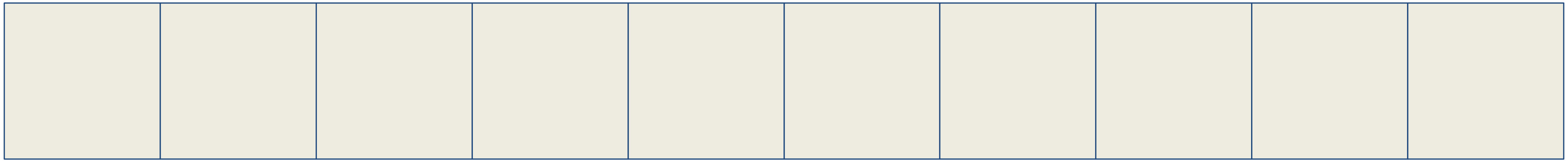
1. Because multiple Channels are handled by a single Netty EventLoop Thread, the block time is significantly reduced

# Blocking EventLoop Thread is not fatal

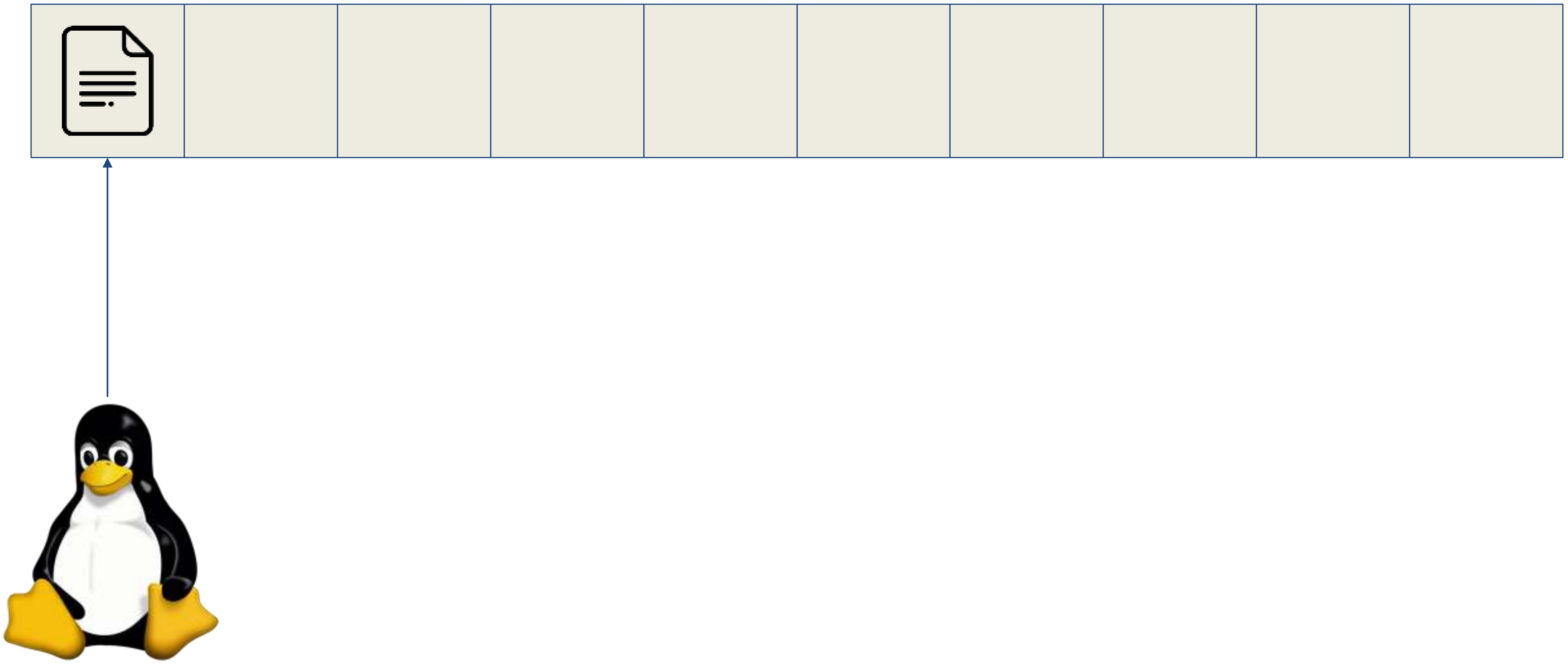
1. Because multiple Channels are handled by a single Netty EventLoop Thread, the block time is significantly reduced

2. The EventLoop Thread is typically not the thread from the Schedulers pool used for reactive assembly pipeline

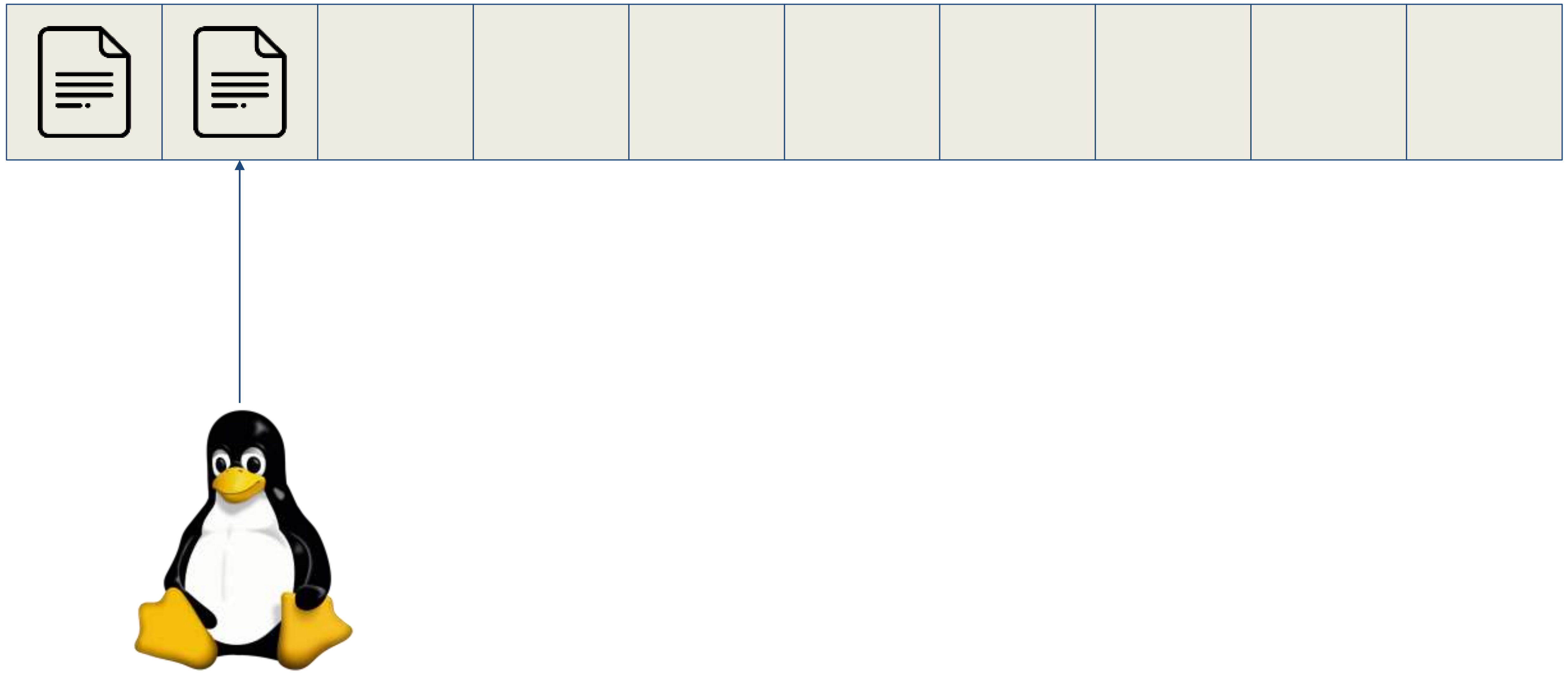
## recv buffer



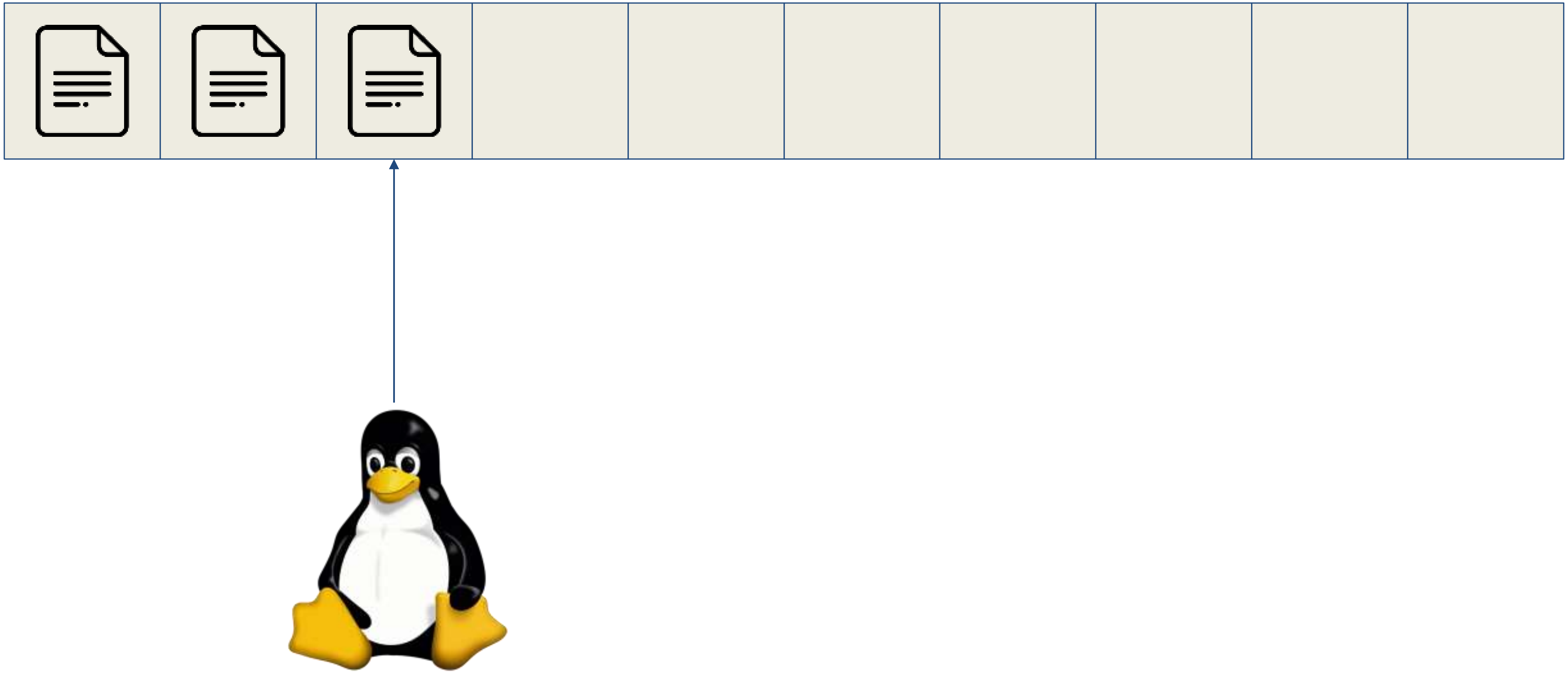
## recv buffer



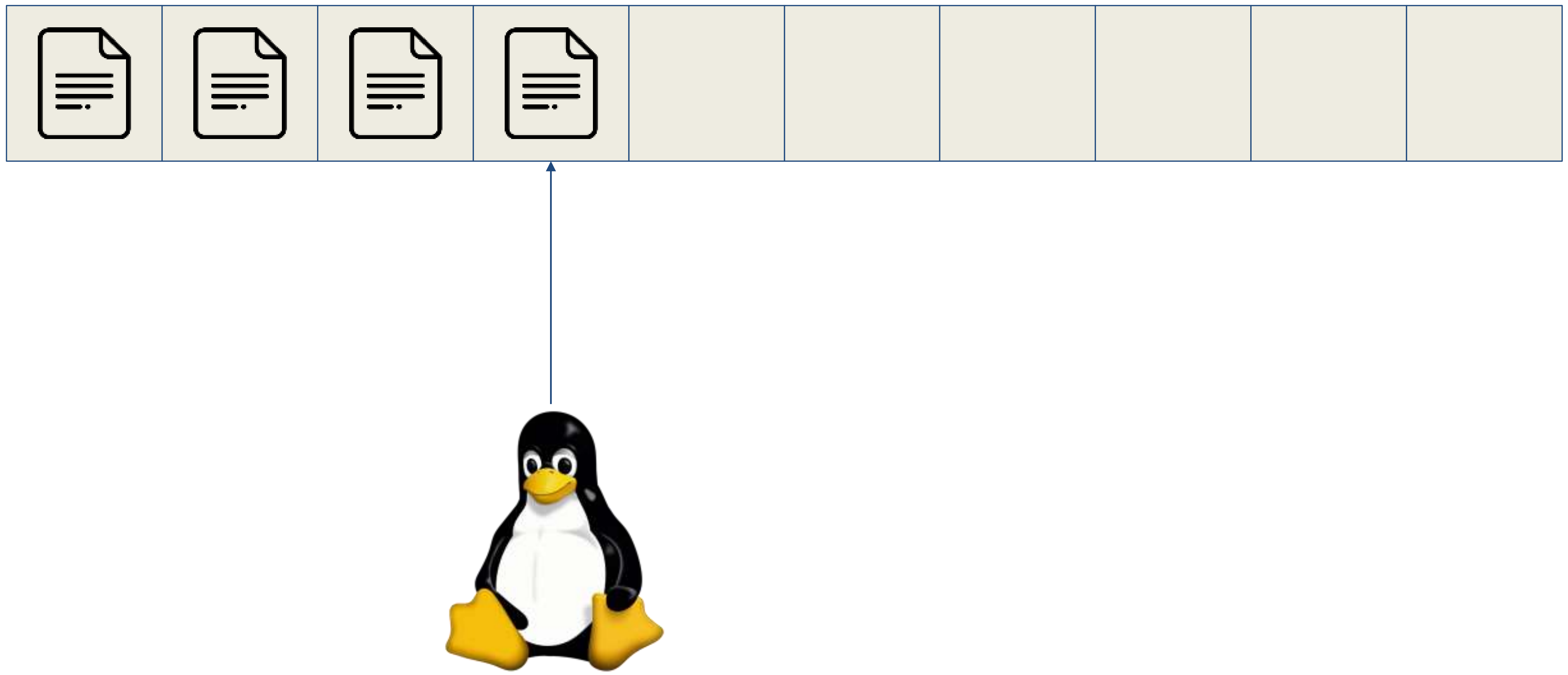
## recv buffer



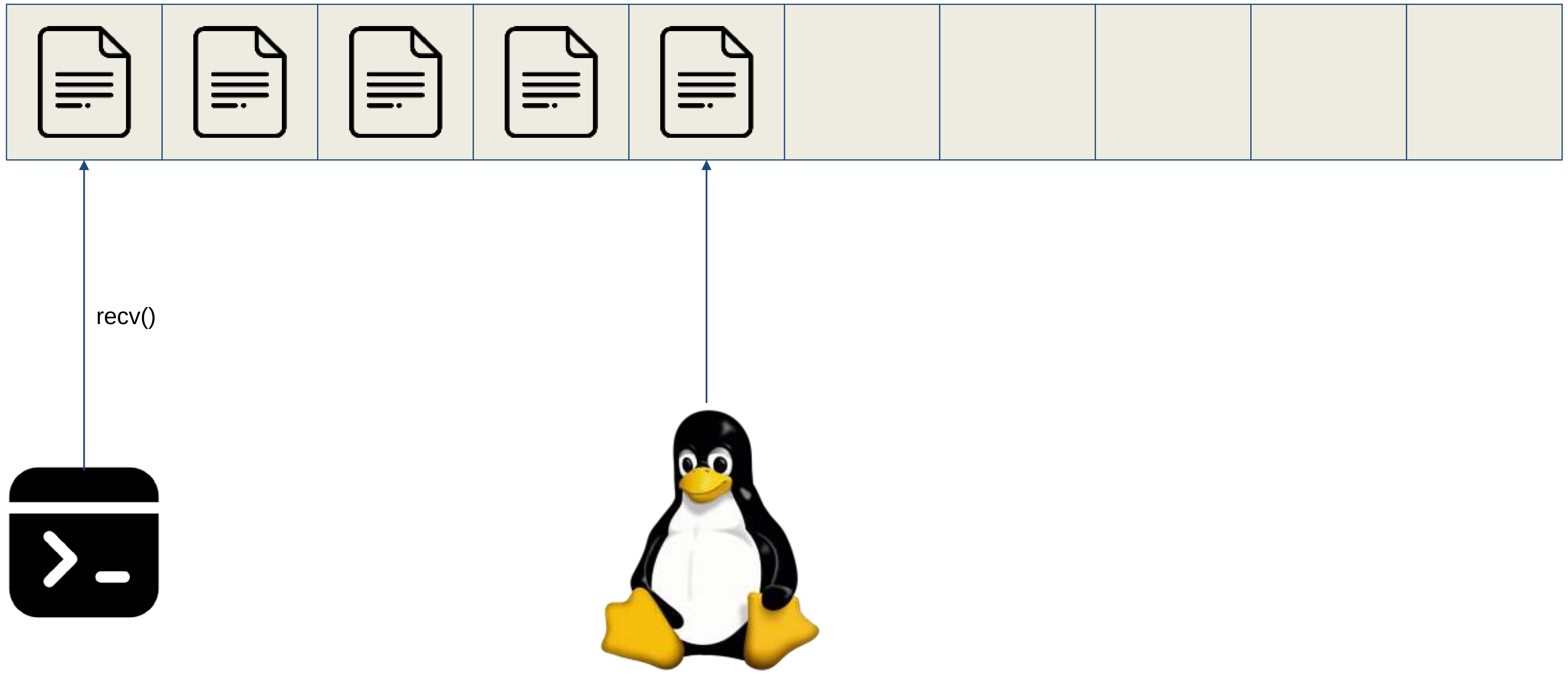
## recv buffer



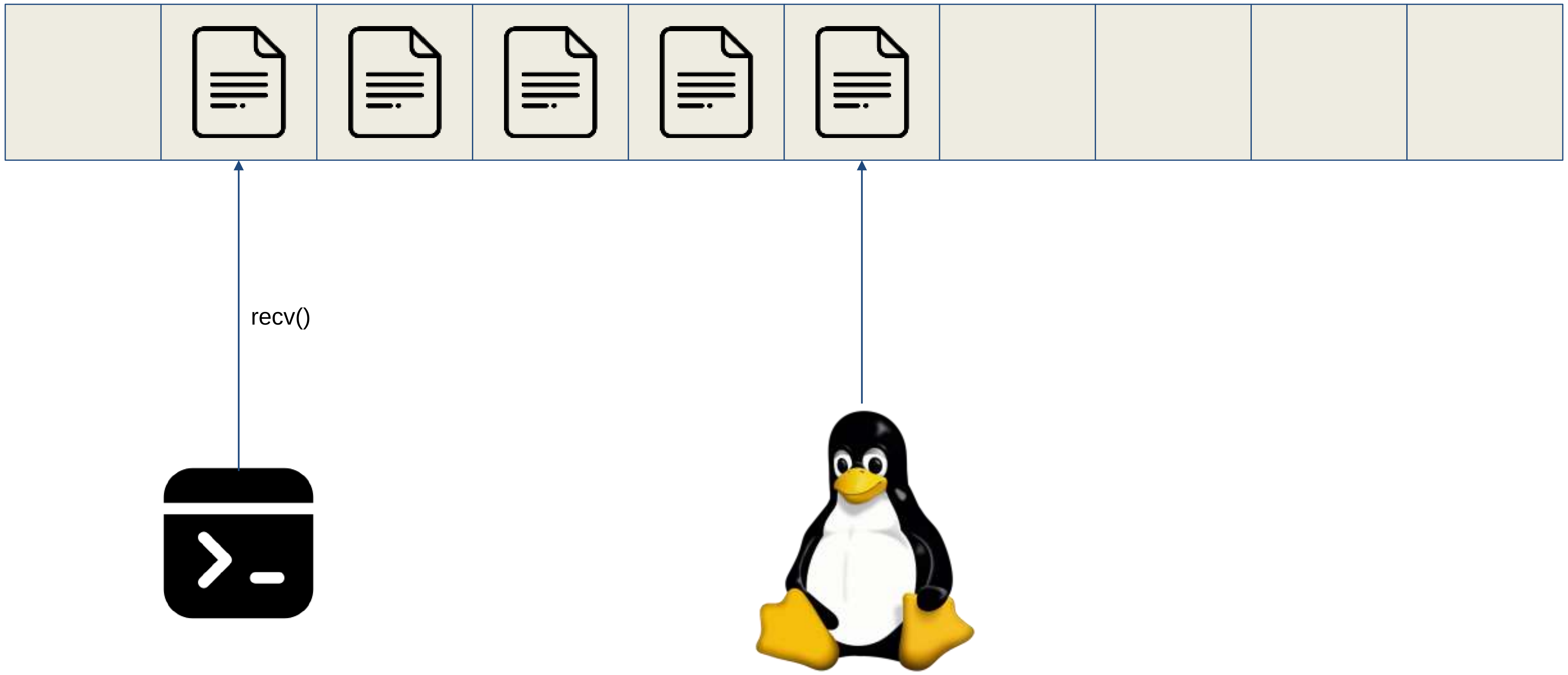
## recv buffer



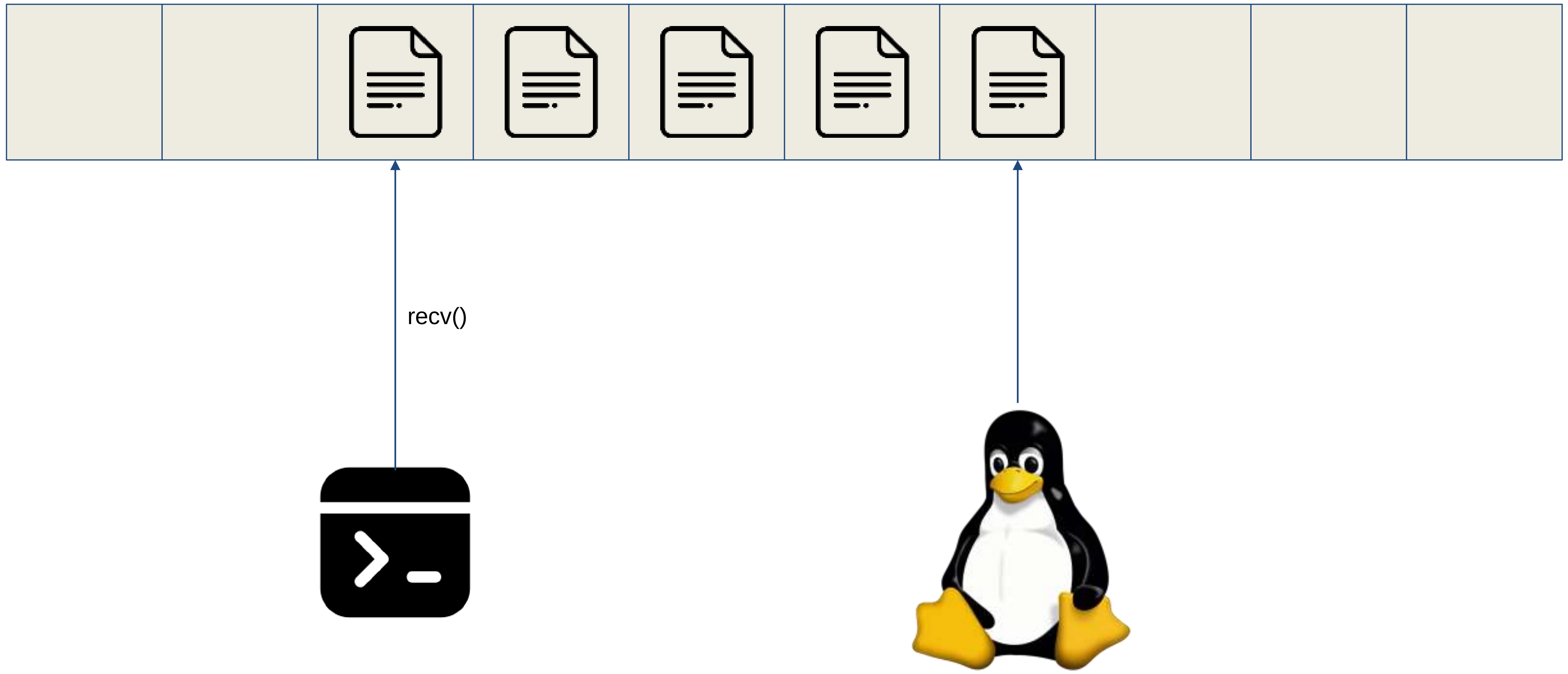
# recv buffer



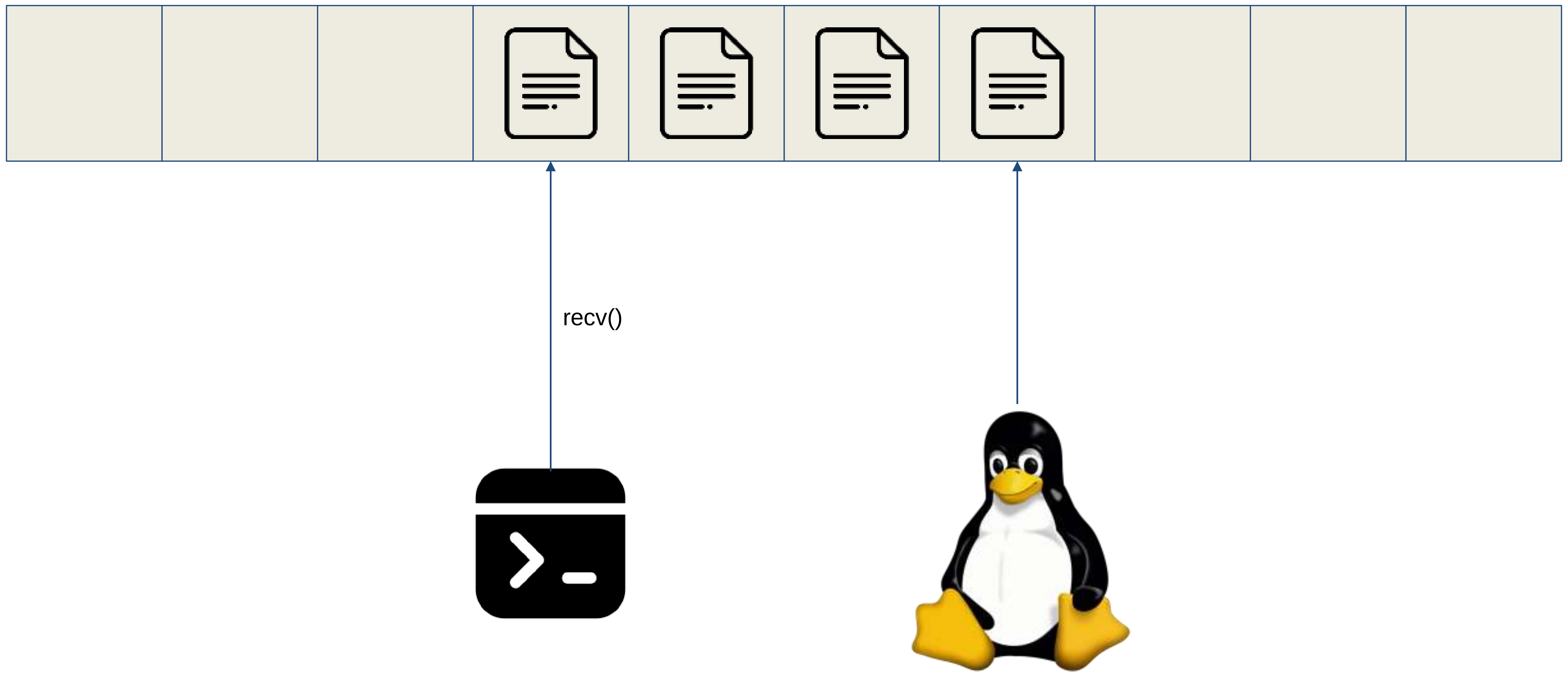
## recv buffer



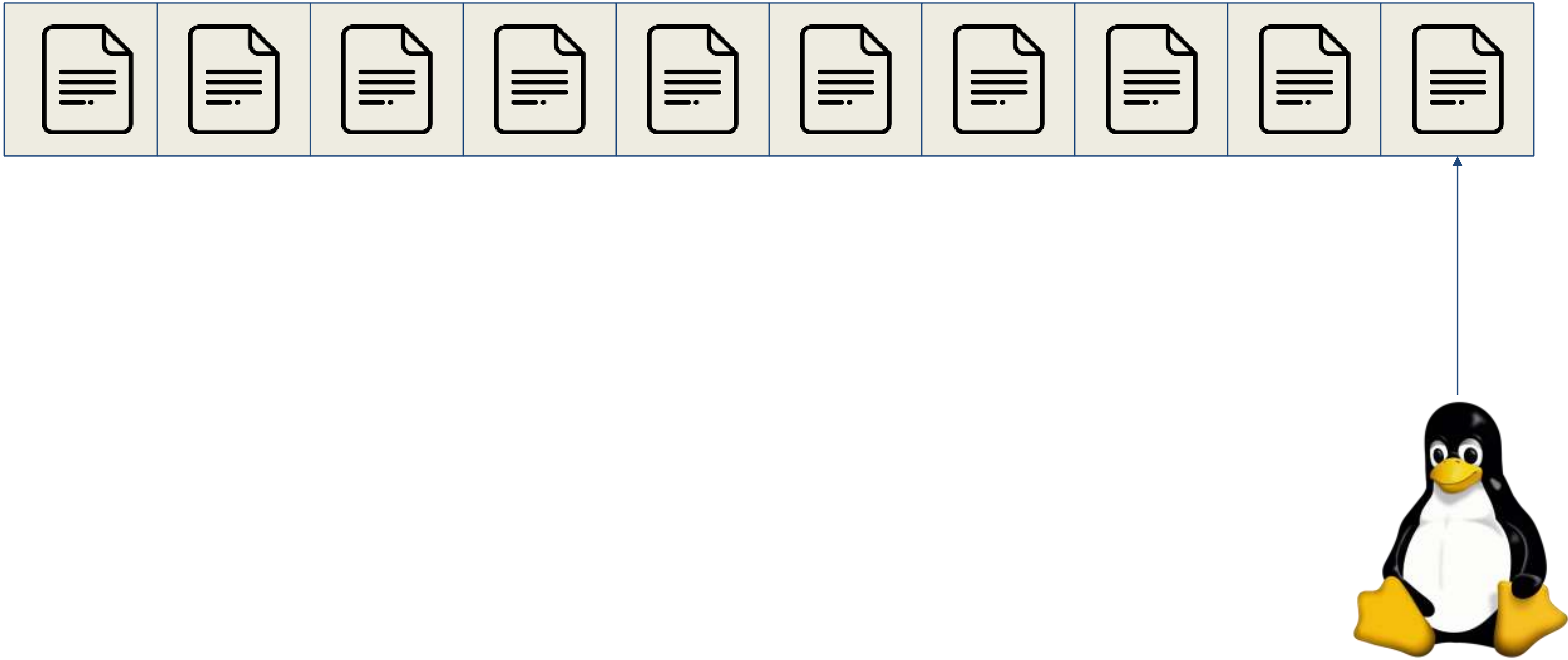
## recv buffer

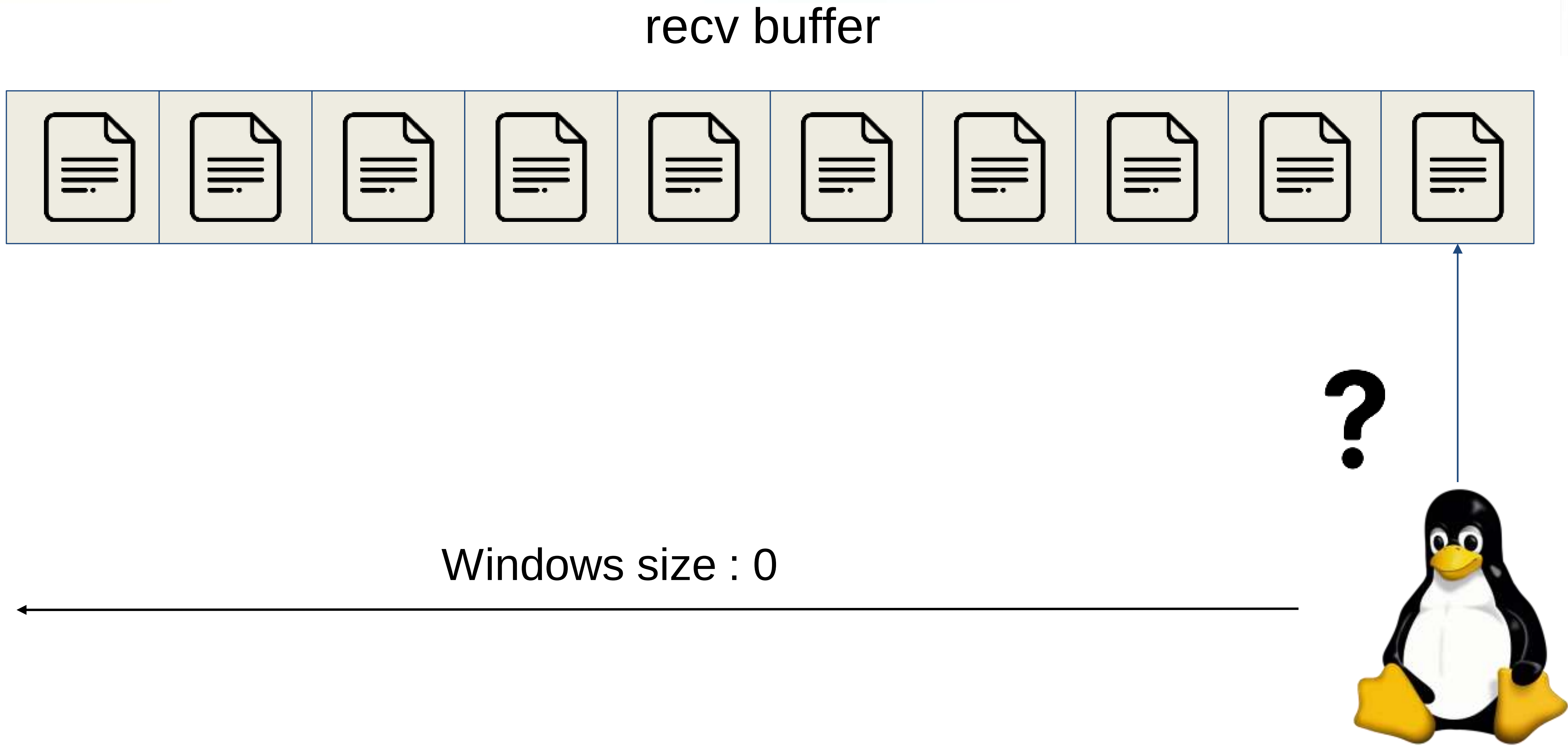


## recv buffer

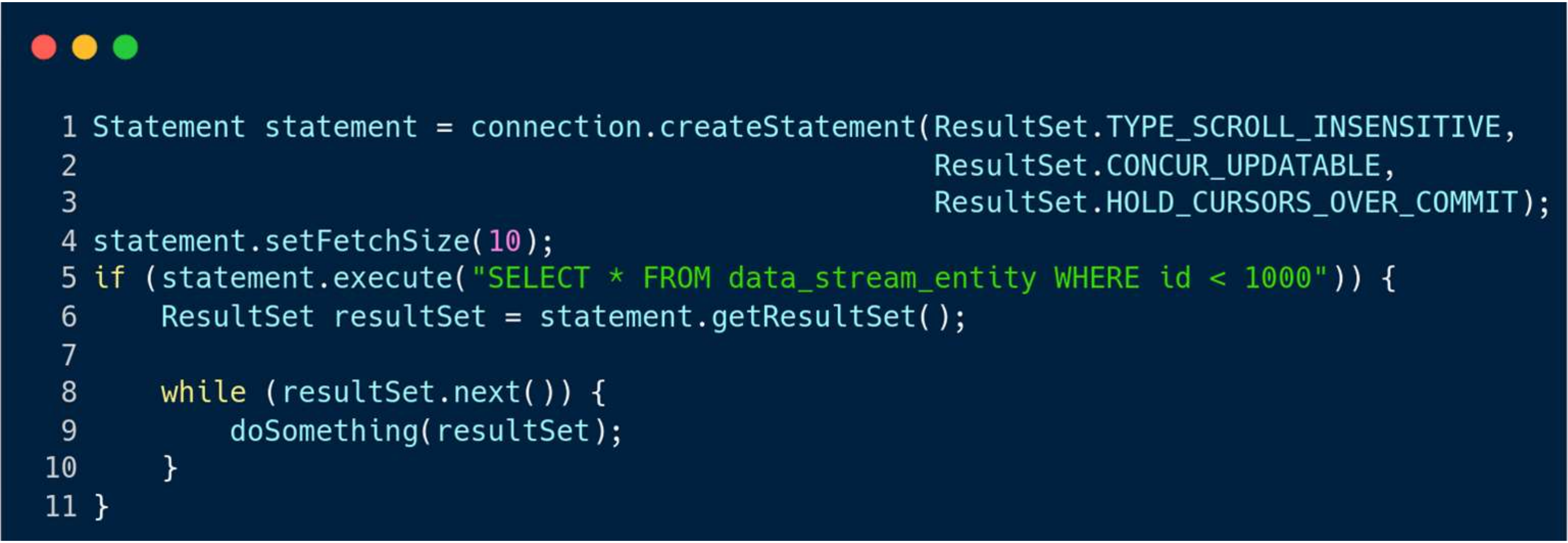


## recv buffer



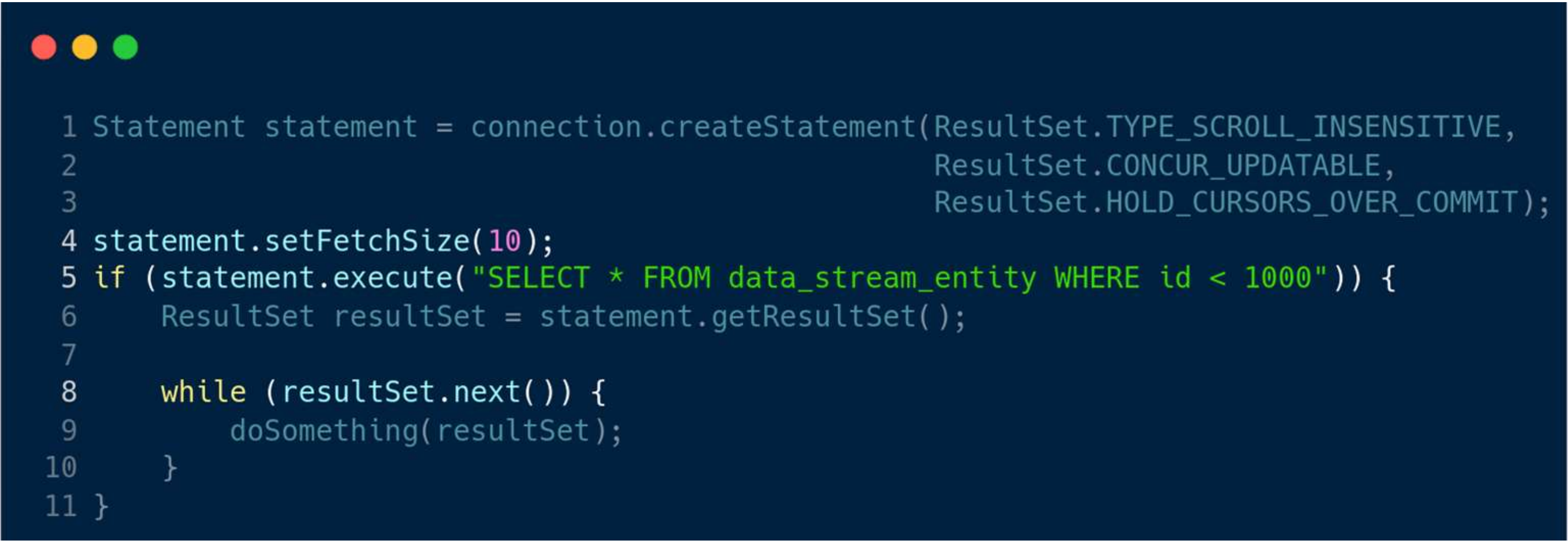

recv buffer
?
Windows size : 0

## We already have fetch size!



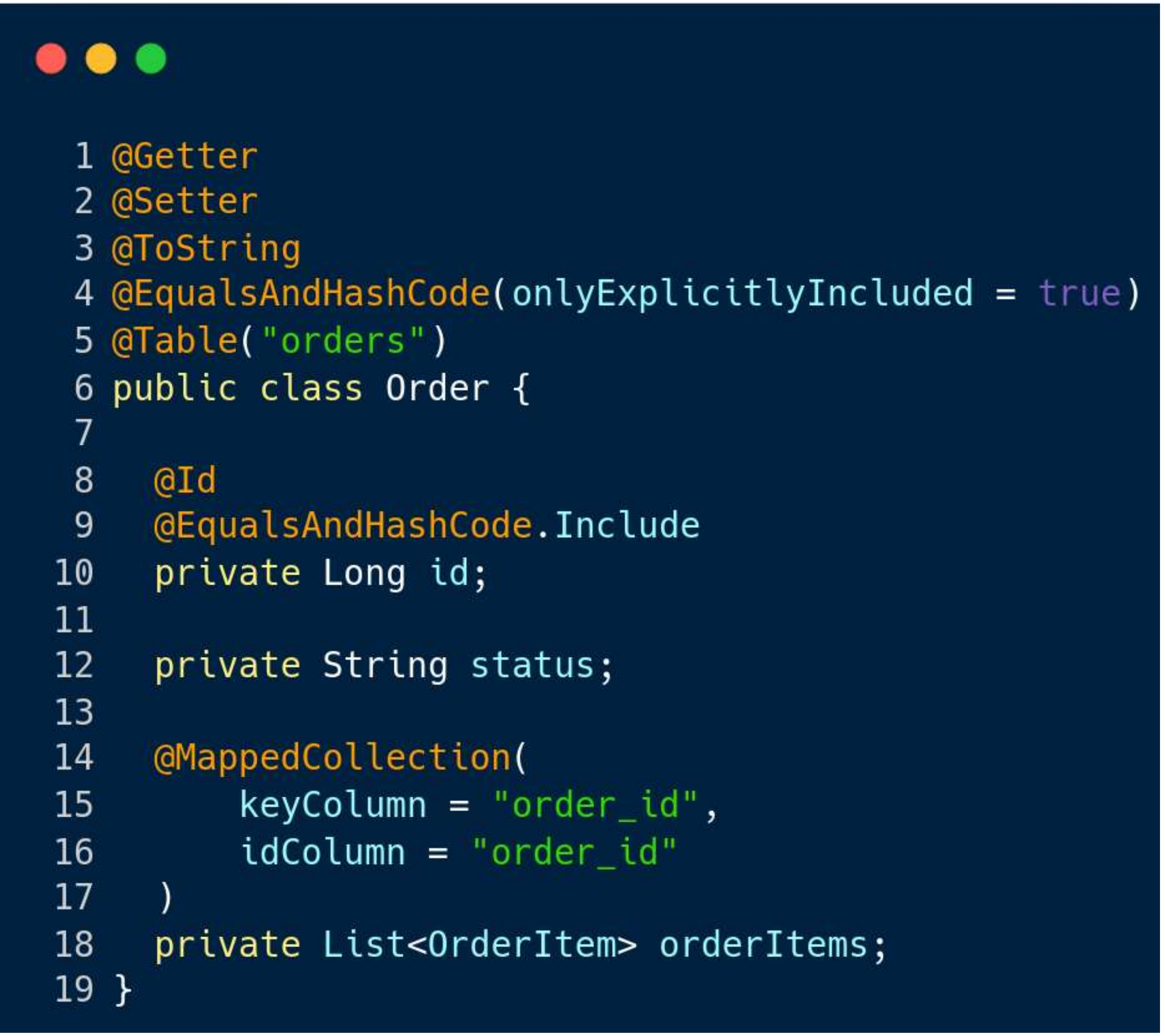
```java
Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                                 ResultSet.CONCUR_UPDATABLE,
                                                 ResultSet.HOLD_CURSORS_OVER_COMMIT);
statement.setFetchSize(10);
if (statement.execute("SELECT * FROM data_stream_entity WHERE id < 1000")) {
    ResultSet resultSet = statement.getResultSet();

    while (resultSet.next()) {
        doSomething(resultSet);
    }
}
```

# We already have fetch size!

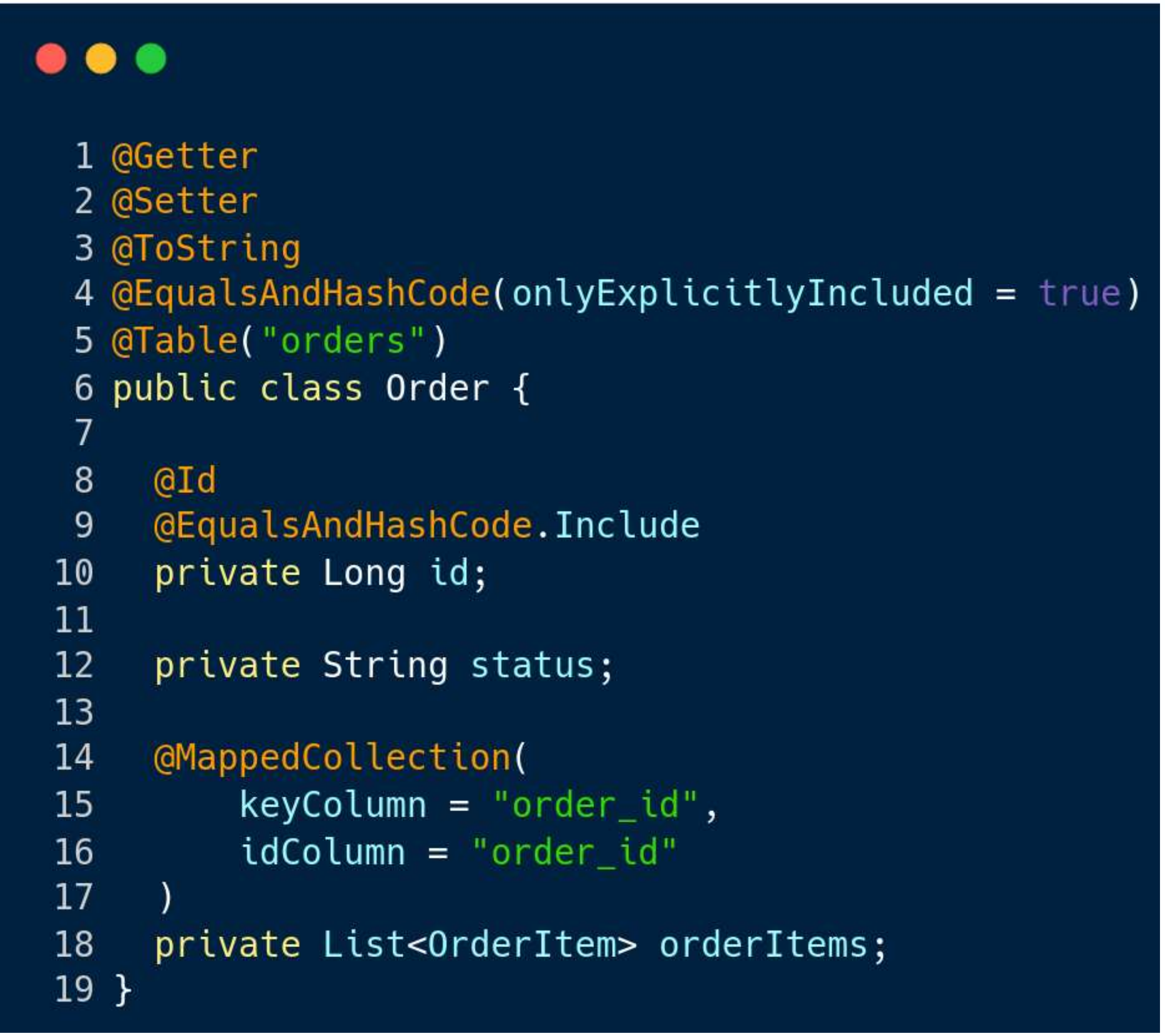


```java
Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                                 ResultSet.CONCUR_UPDATABLE,
                                                 ResultSet.HOLD_CURSORS_OVER_COMMIT);
statement.setFetchSize(10);
if (statement.execute("SELECT * FROM data_stream_entity WHERE id < 1000")) {
    ResultSet resultSet = statement.getResultSet();

    while (resultSet.next()) {
        doSomething(resultSet);
    }
}
```

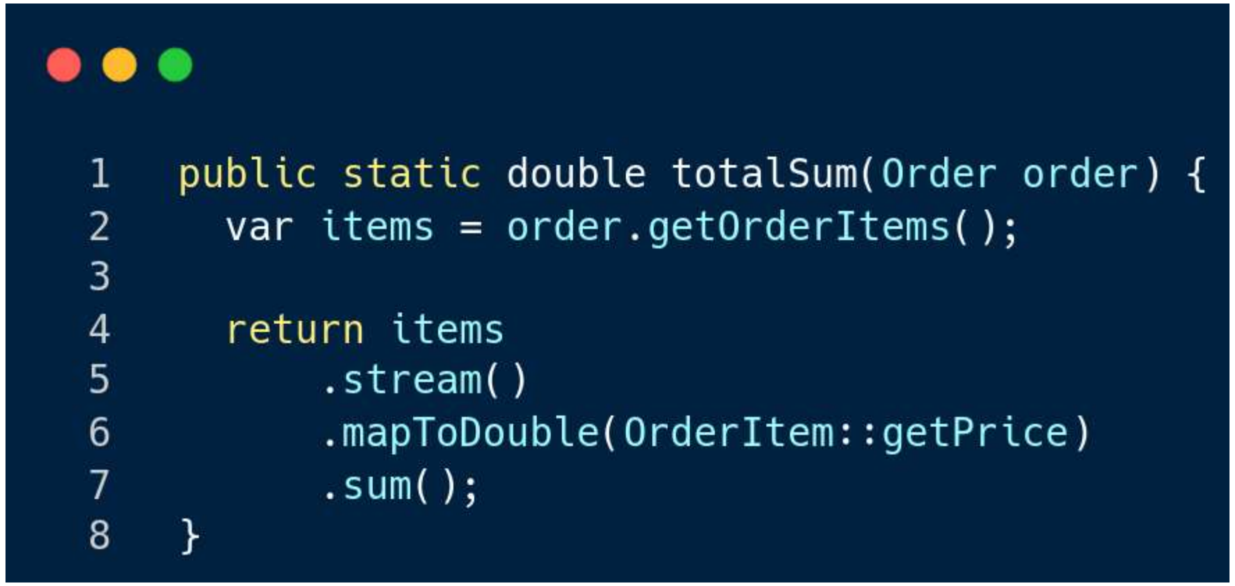
## Spring-Data-R2DBC restrictions



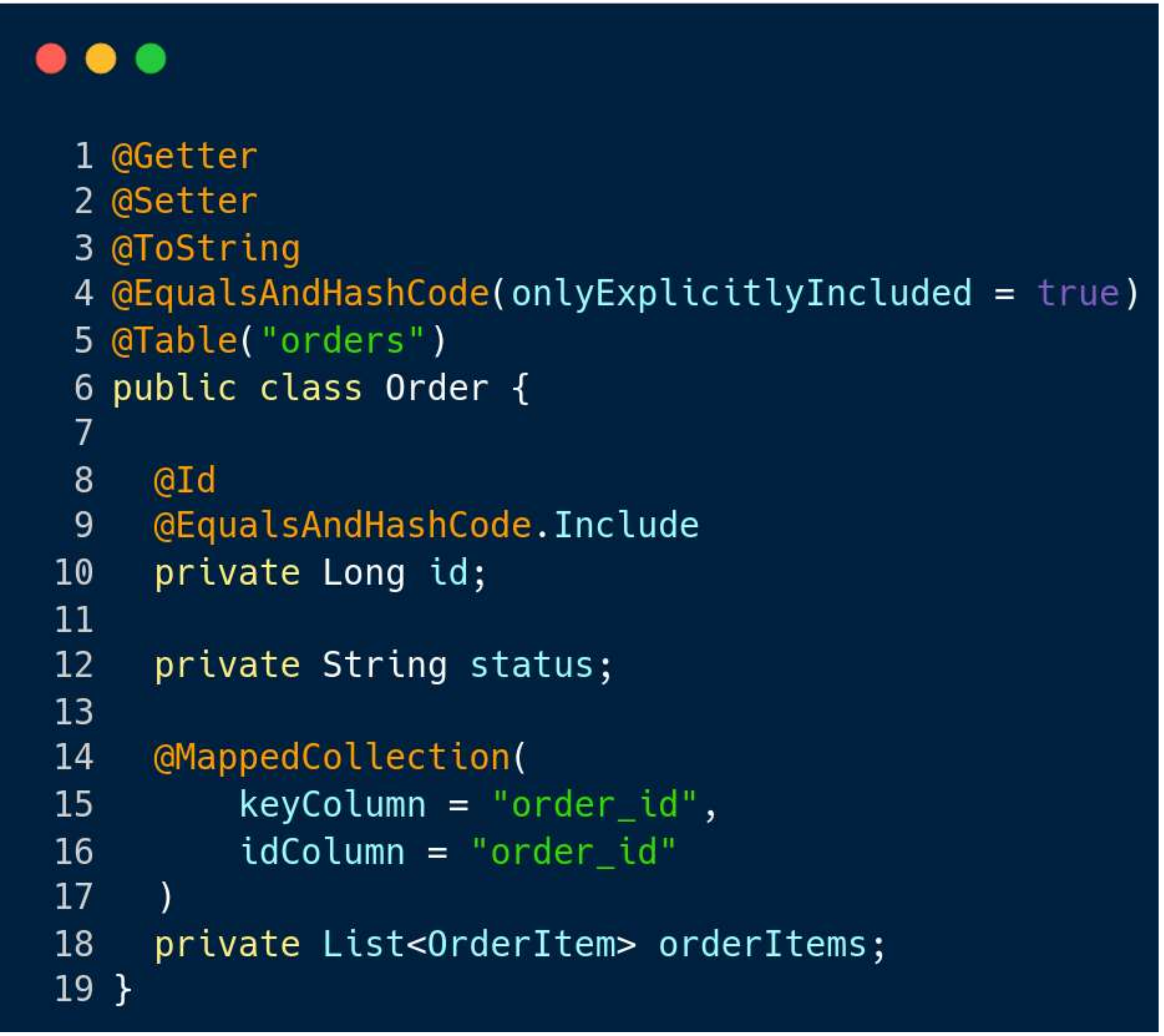
```java
@Getter
@Setter
@ToString
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
@Table("orders")
public class Order {

  @Id
  @EqualsAndHashCode.Include
  private Long id;

  private String status;

  @MappedCollection(
      keyColumn = "order_id",
      idColumn = "order_id"
  )
  private List<OrderItem> orderItems;
}
```



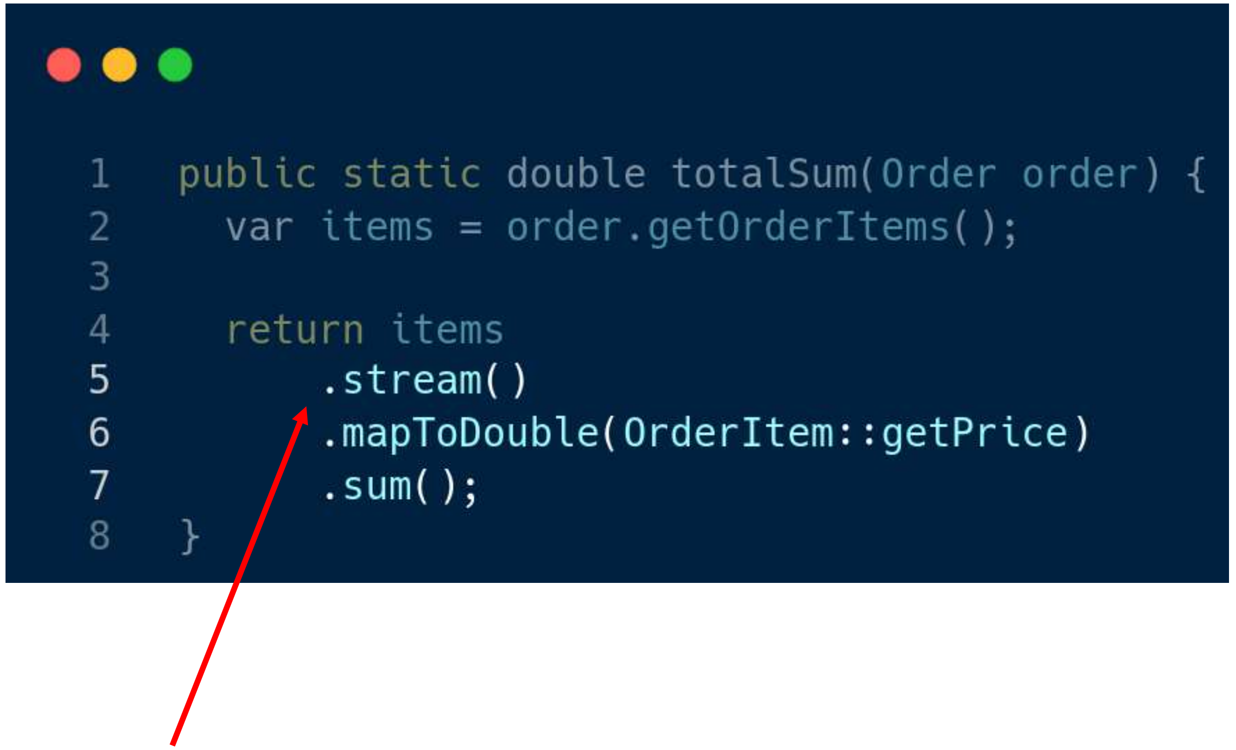
```java
@Getter
@Setter
@ToString
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
@Table("orders")
public class Order {

  @Id
  @EqualsAndHashCode.Include
  private Long id;

  private String status;

  @MappedCollection(
      keyColumn = "order_id",
      idColumn = "order_id"
  )
  private List<OrderItem> orderItems;
}
```



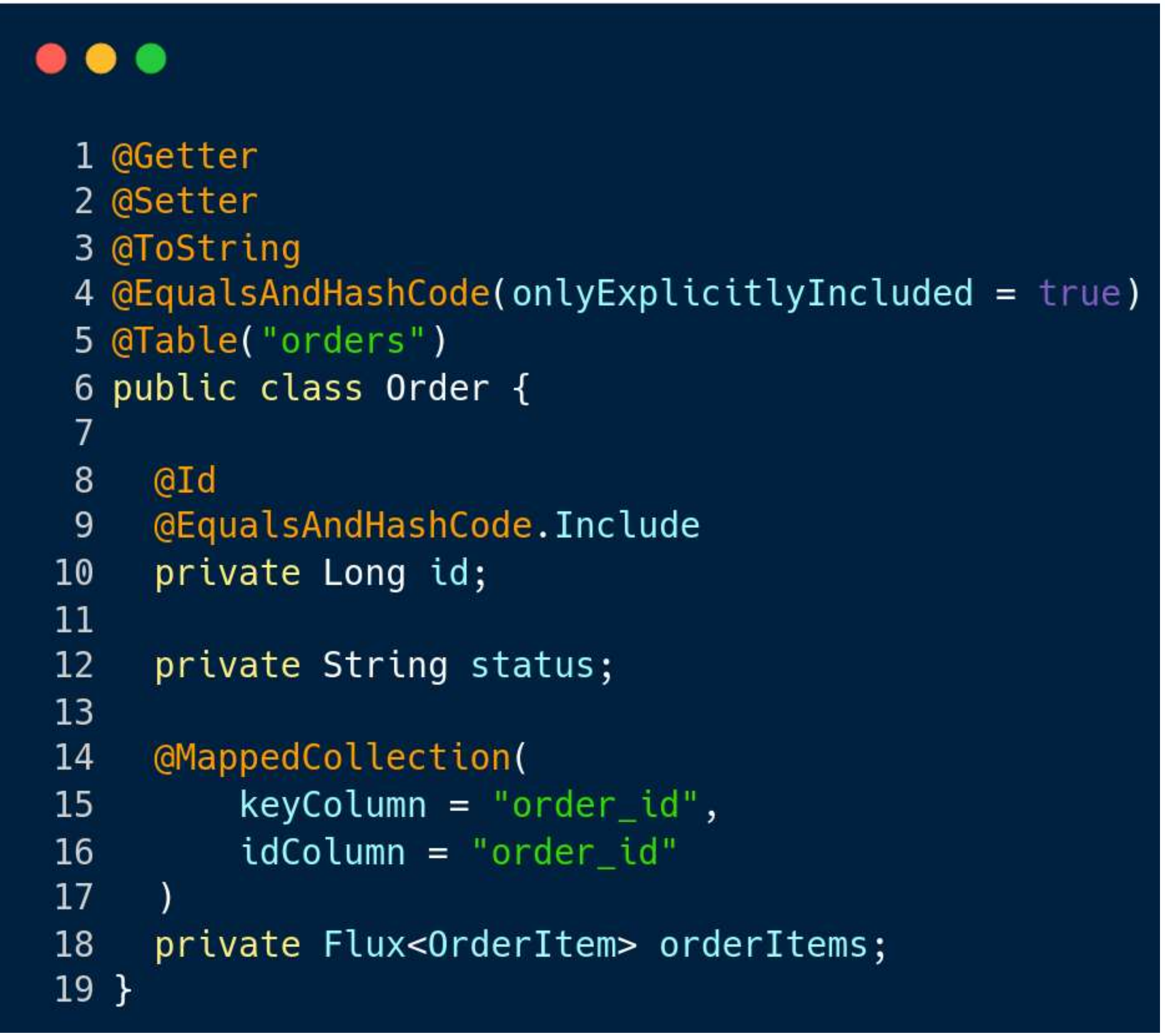
```java
public static double totalSum(Order order) {
  var items = order.getOrderItems();

  return items
      .stream()
      .mapToDouble(OrderItem::getPrice)
      .sum();
}
```



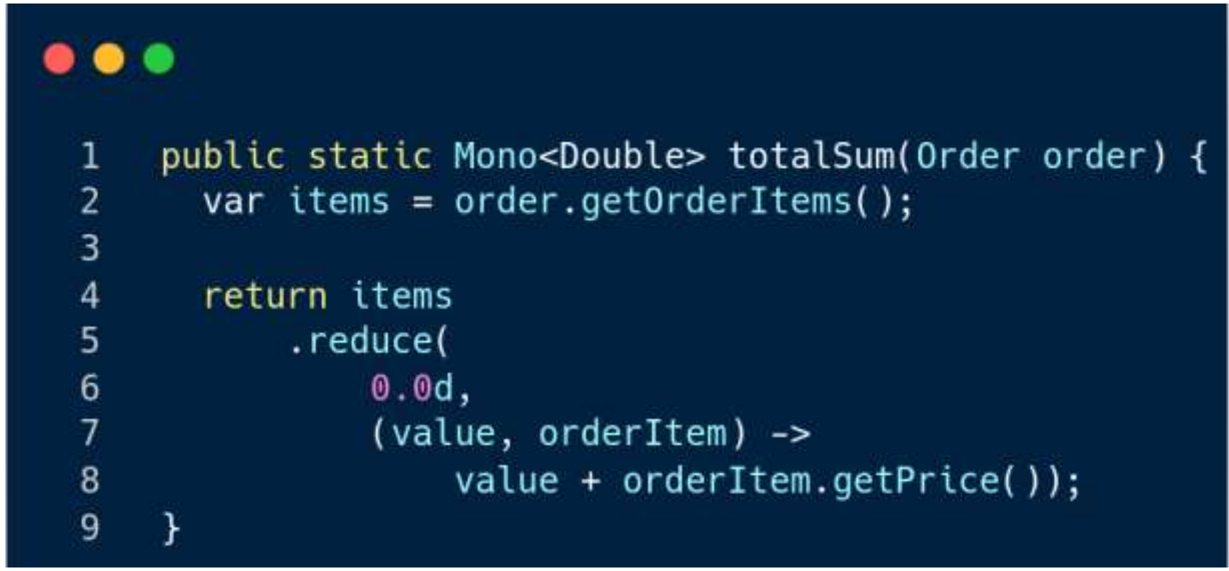
```java
@Getter
@Setter
@ToString
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
@Table("orders")
public class Order {

  @Id
  @EqualsAndHashCode.Include
  private Long id;

  private String status;

  @MappedCollection(
      keyColumn = "order_id",
      idColumn = "order_id"
  )
  private List<OrderItem> orderItems;
}
```



```java
public static double totalSum(Order order) {
  var items = order.getOrderItems();

  return items
        .stream()
        .mapToDouble(OrderItem::getPrice)
        .sum();
}
```



```java
@Getter
@Setter
@ToString
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
@Table("orders")
public class Order {

  @Id
  @EqualsAndHashCode.Include
  private Long id;

  private String status;

  @MappedCollection(
      keyColumn = "order_id",
      idColumn = "order_id"
  )
  private Flux<OrderItem> orderItems;
}
```



```java
public static Mono<Double> totalSum(Order order) {
  var items = order.getOrderItems();

  return items
      .reduce(
          0.0d,
          (value, orderItem) ->
              value + orderItem.getPrice());
}
```

## Why ReactiveRelation<T> is controversial

1. We’re polluting our domain model with reactive stack, which is not a very good thing

## Why ReactiveRelation<T> is controversial

1. We’re polluting our domain model with reactive stack, which is not a very good thing

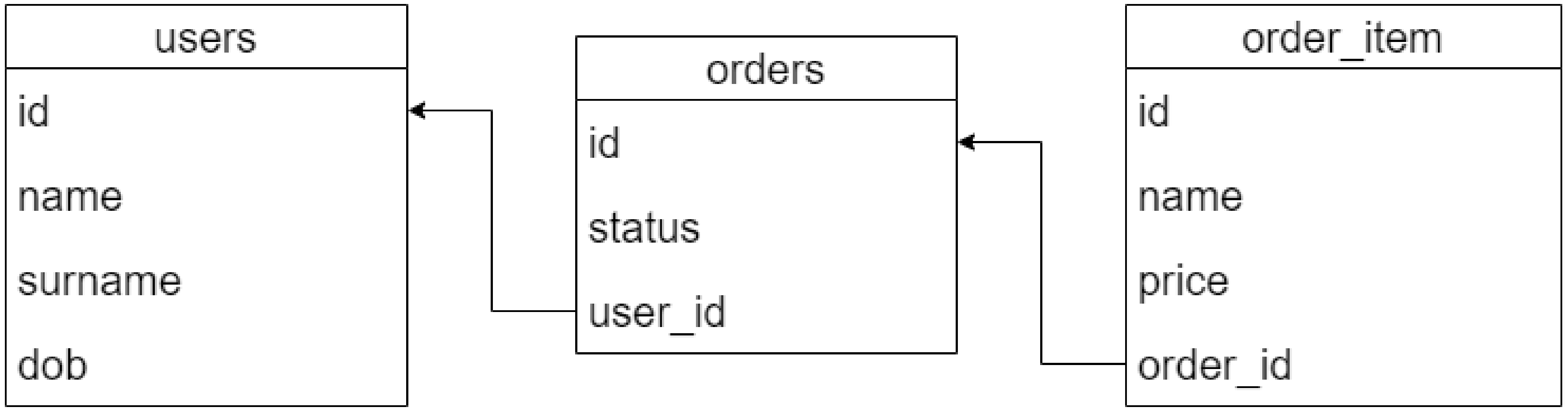
2. There is no clear mutation abilities. E.g. there is no way to add an element to a Flux.

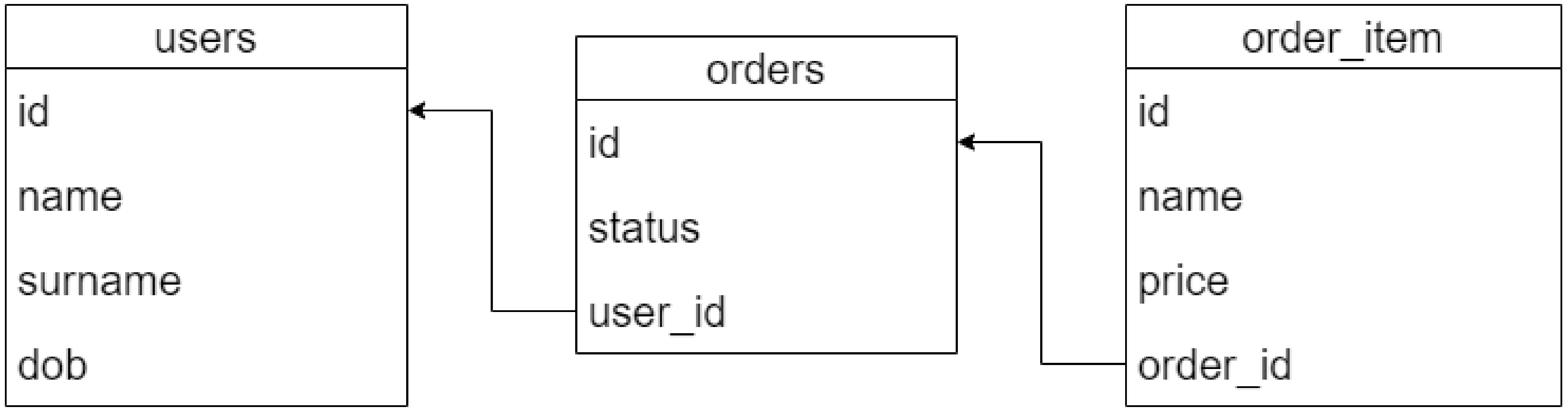
## Why ReactiveRelation<T> is controversial

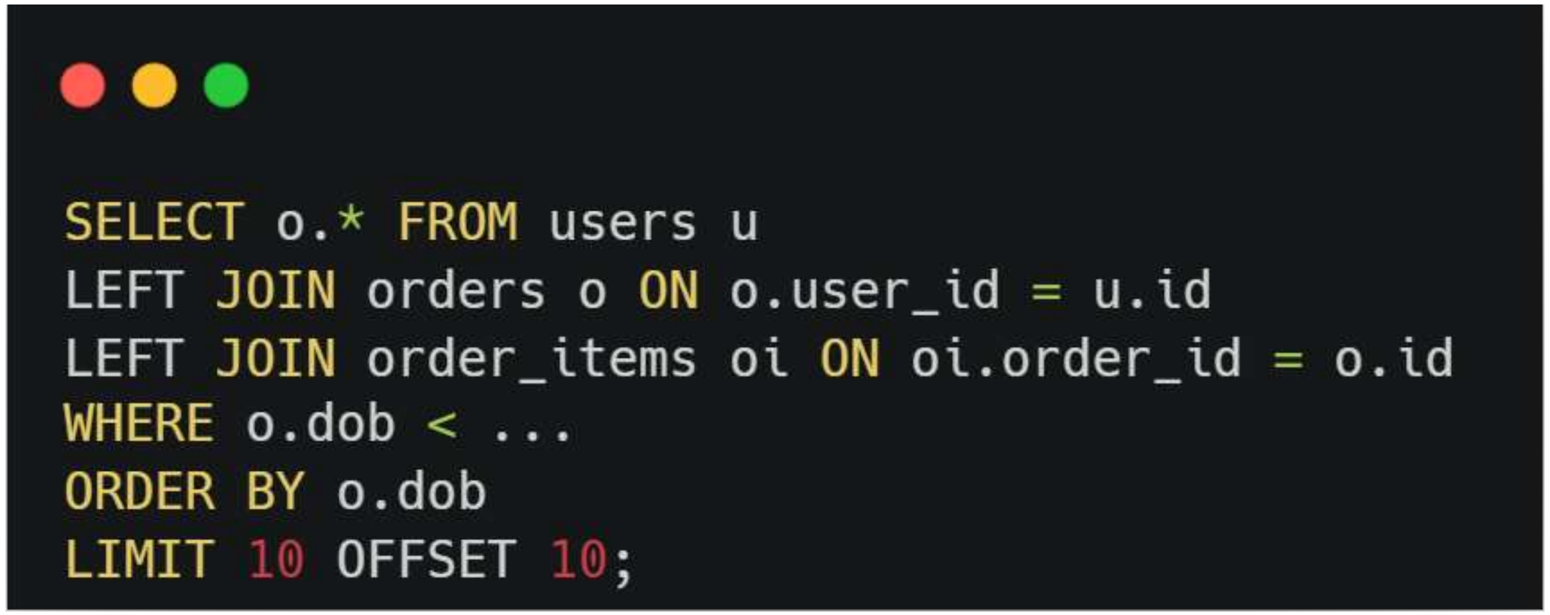
1. We’re polluting our domain model with reactive stack, which is not a very good thing

2. There is no clear mutation abilities. E.g. there is no way to add an element to a Flux.

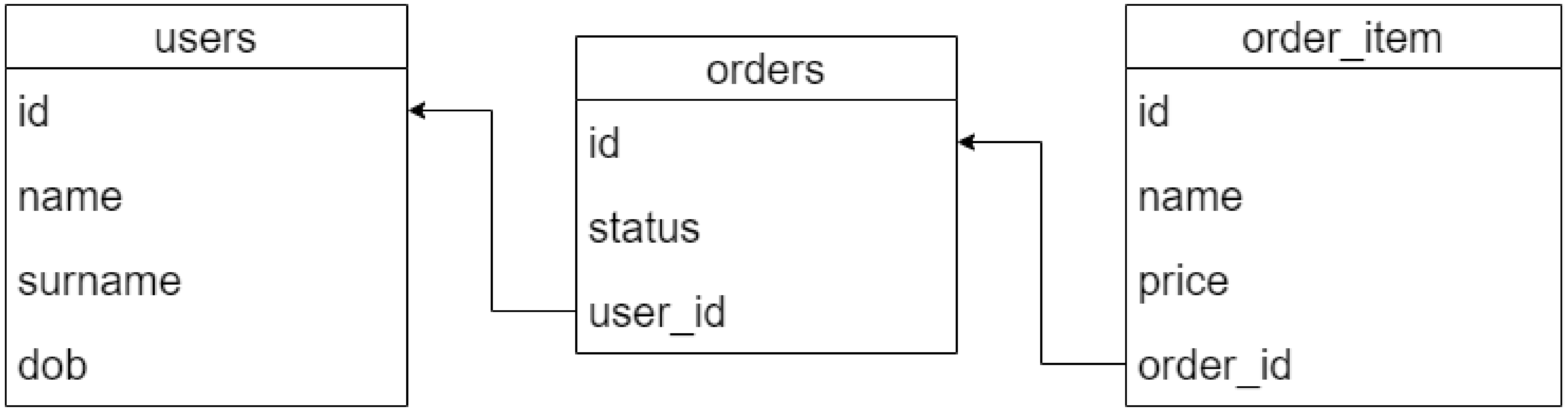
3. Transaction boundaries for reactive relations have to be honored. There is no out of the box solution for that.

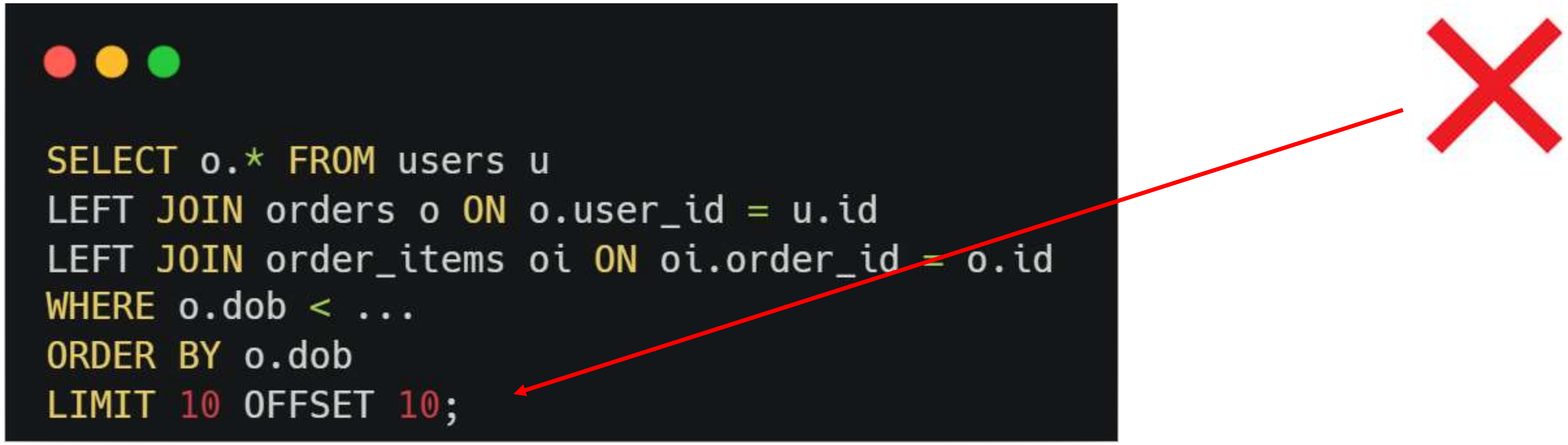

users
id
name
surname
dob
orders
id
status
user_id
order_item
id
name
price
order_id



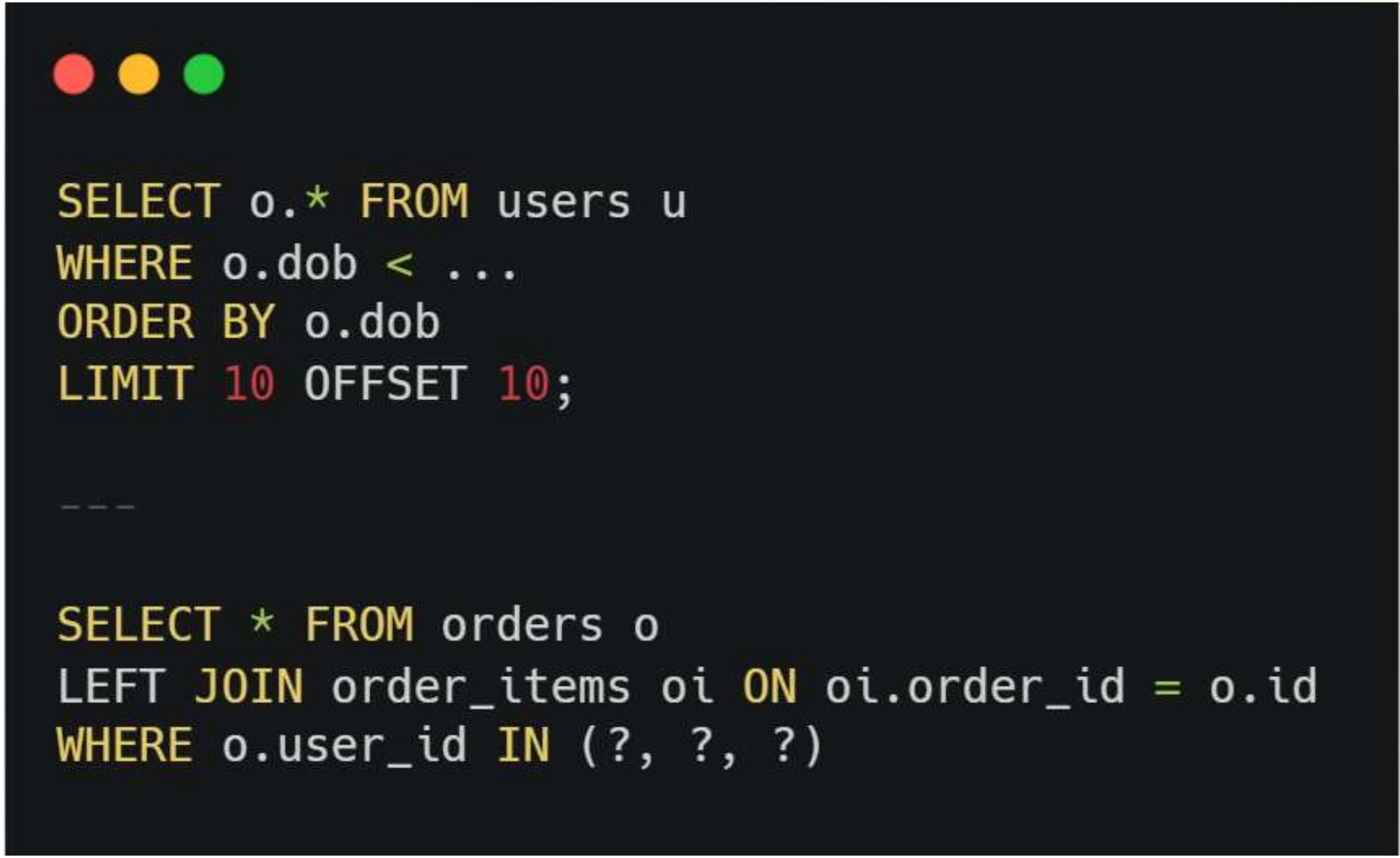


```sql
SELECT o.* FROM users u
LEFT JOIN orders o ON o.user_id = u.id
LEFT JOIN order_items oi ON oi.order_id = o.id
WHERE o.dob < ...
ORDER BY o.dob
LIMIT 10 OFFSET 10;
```



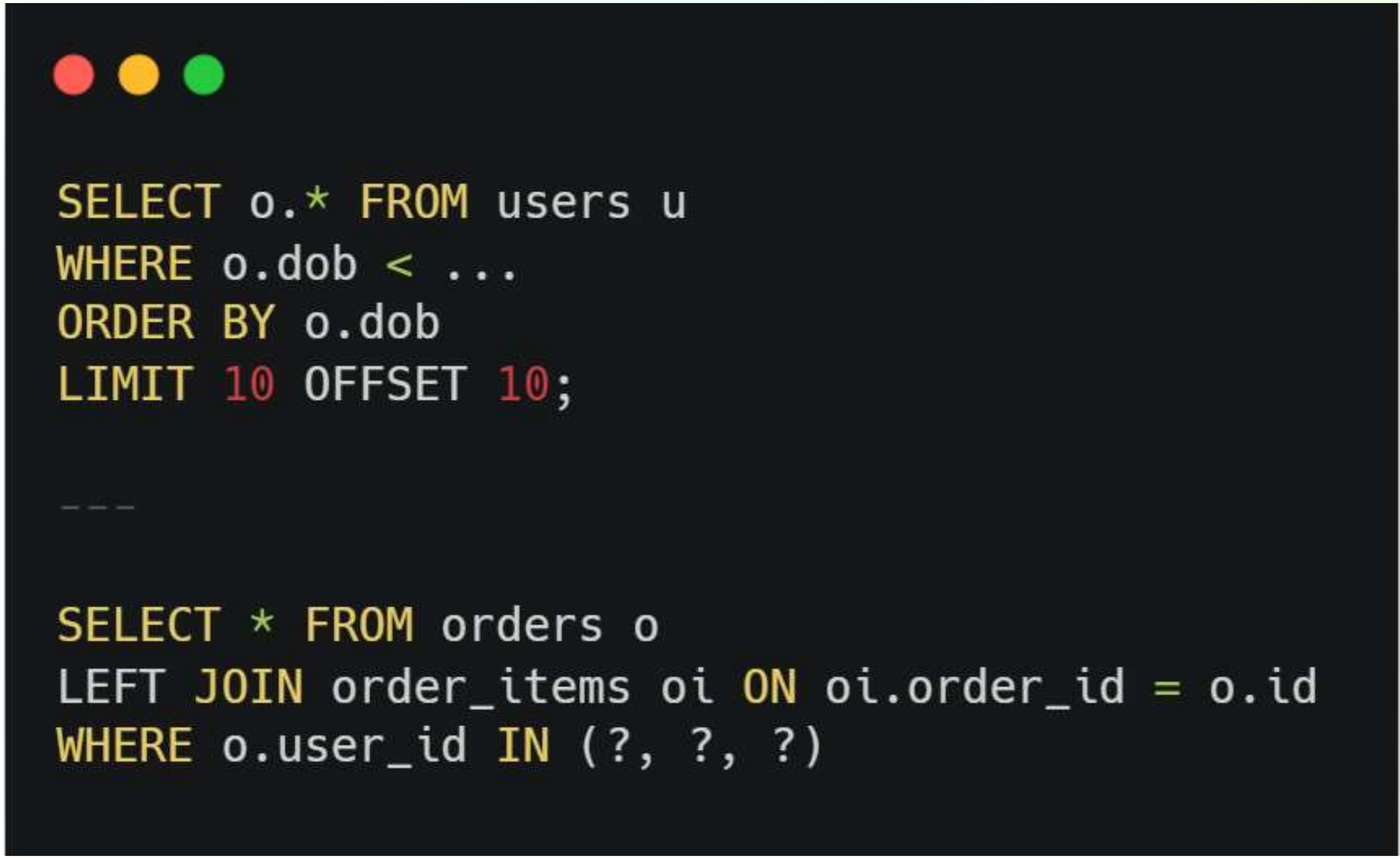


```
SELECT o.* FROM users u
LEFT JOIN orders o ON o.user_id = u.id
LEFT JOIN order_items oi ON oi.order_id = o.id
WHERE o.dob < ...
ORDER BY o.dob
LIMIT 10 OFFSET 10;
```



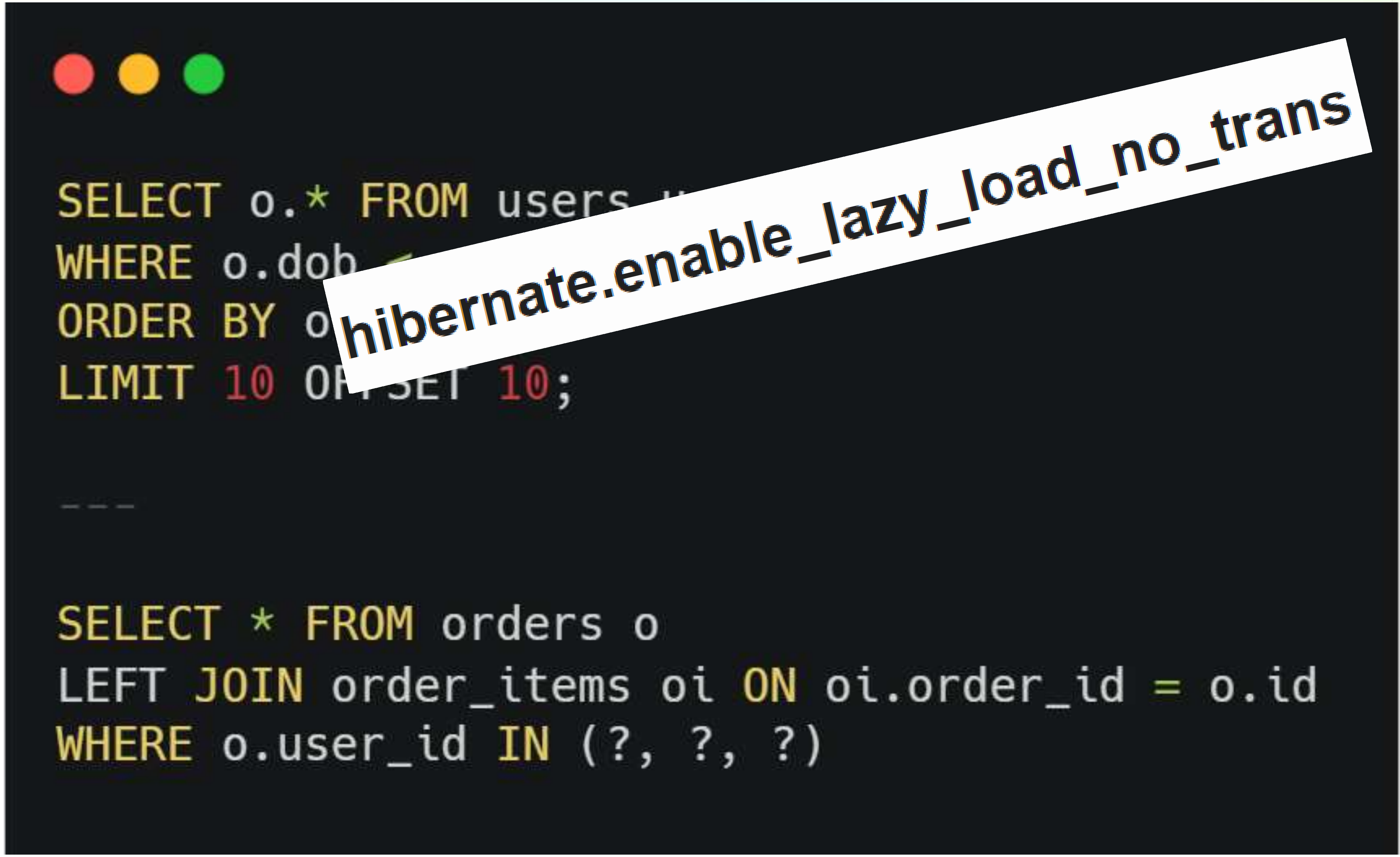
```sql
SELECT o.* FROM users u
WHERE o.dob < ...
ORDER BY o.dob
LIMIT 10 OFFSET 10;

---

SELECT * FROM orders o
LEFT JOIN order_items oi ON oi.order_id = o.id
WHERE o.user_id IN (?, ?, ?)
```

Transaction



```sql
SELECT o.* FROM users u
WHERE o.dob < ...
ORDER BY o.dob
LIMIT 10 OFFSET 10;

---

SELECT * FROM orders o
LEFT JOIN order_items oi ON oi.order_id = o.id
WHERE o.user_id IN (?, ?, ?)
```

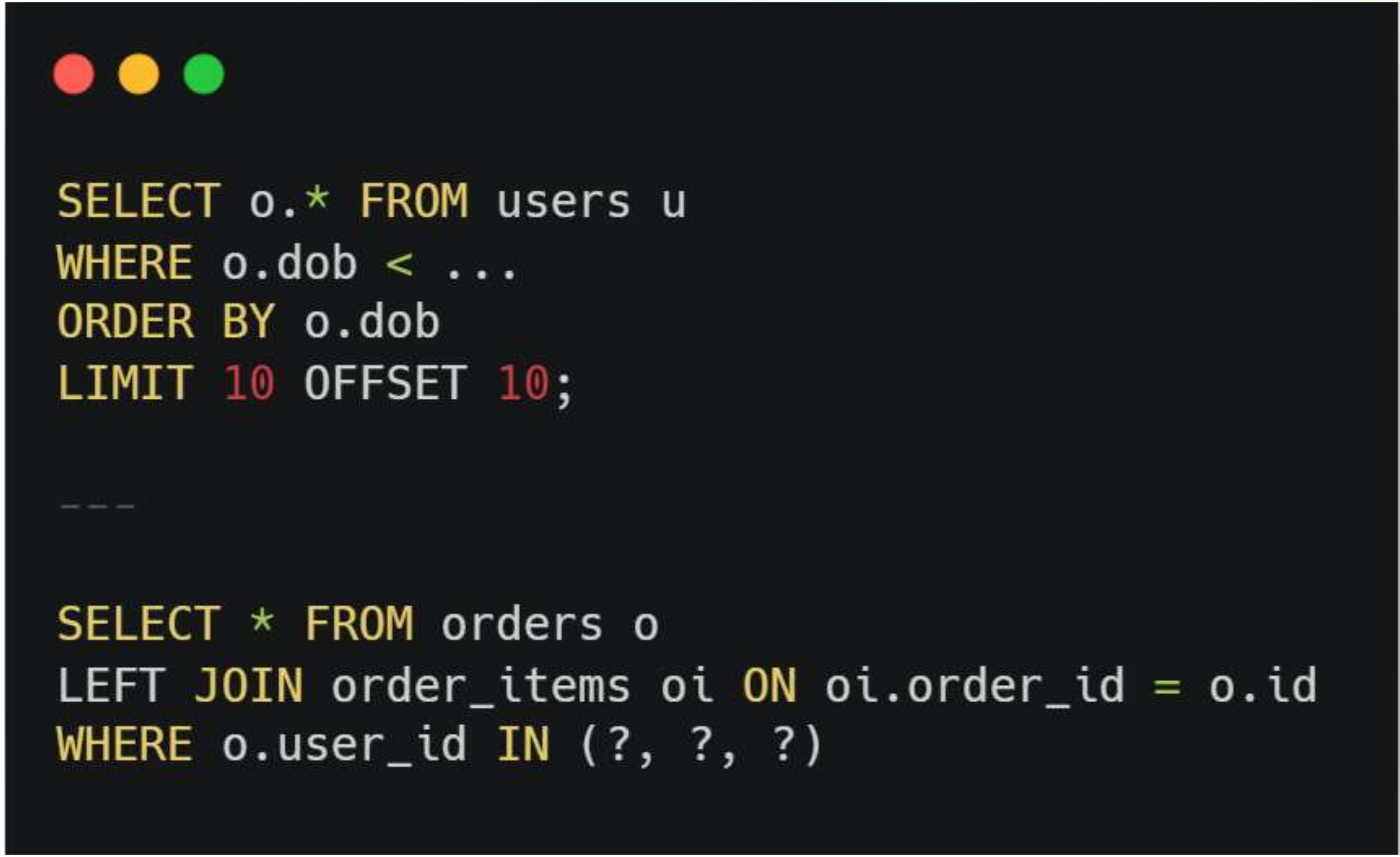
Transaction



hibernate.enable_lazy_load_no_trans

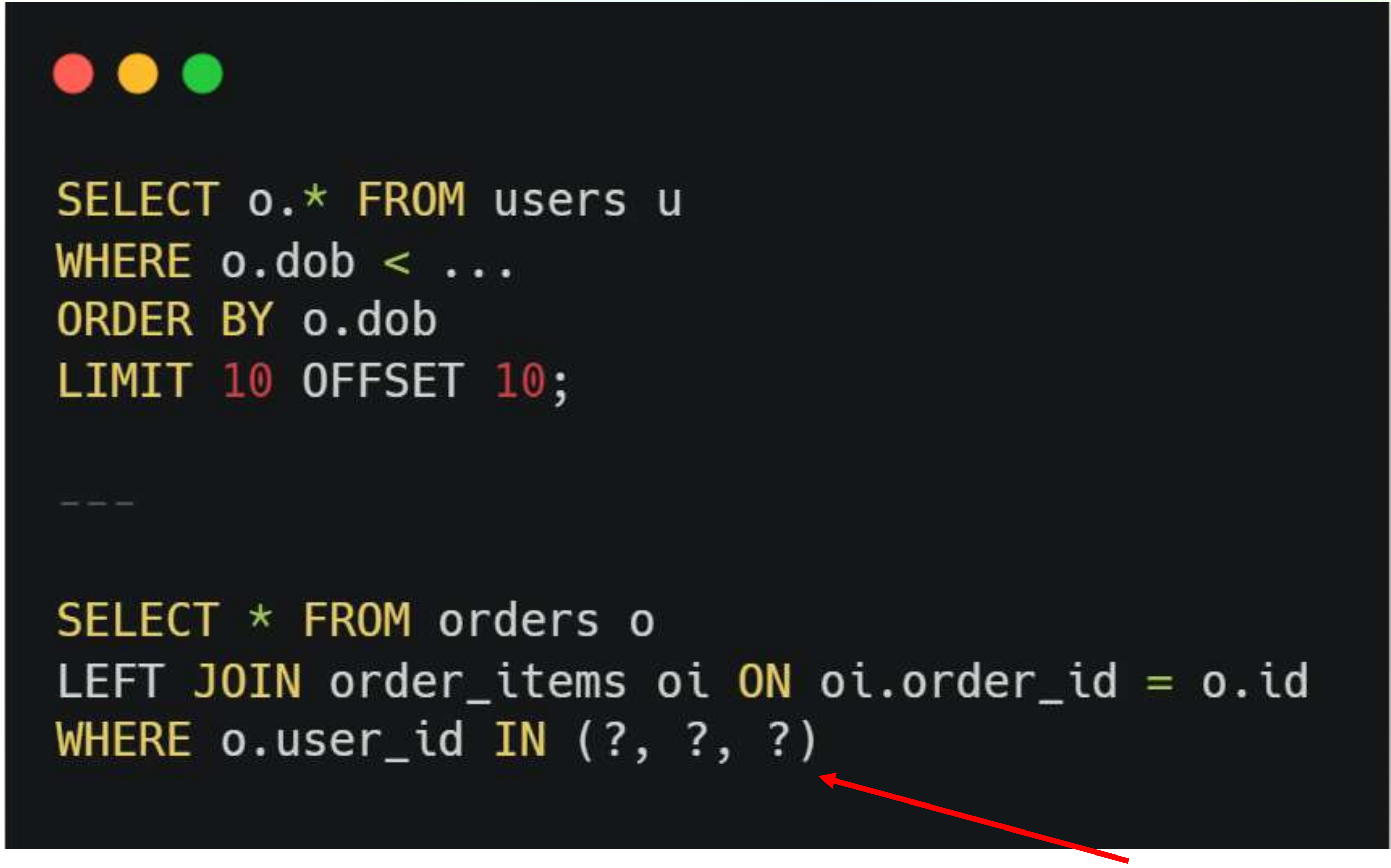
```
SELECT o.* FROM users
WHERE o.dob <
ORDER BY o
LIMIT 10 OFFSET 10;

---

SELECT * FROM orders o
LEFT JOIN order_items oi ON oi.order_id = o.id
WHERE o.user_id IN (?, ?, ?)
```

Transaction



```sql
SELECT o.* FROM users u
WHERE o.dob < ...
ORDER BY o.dob
LIMIT 10 OFFSET 10;

---

SELECT * FROM orders o
LEFT JOIN order_items oi ON oi.order_id = o.id
WHERE o.user_id IN (?, ?, ?)
```

Transaction

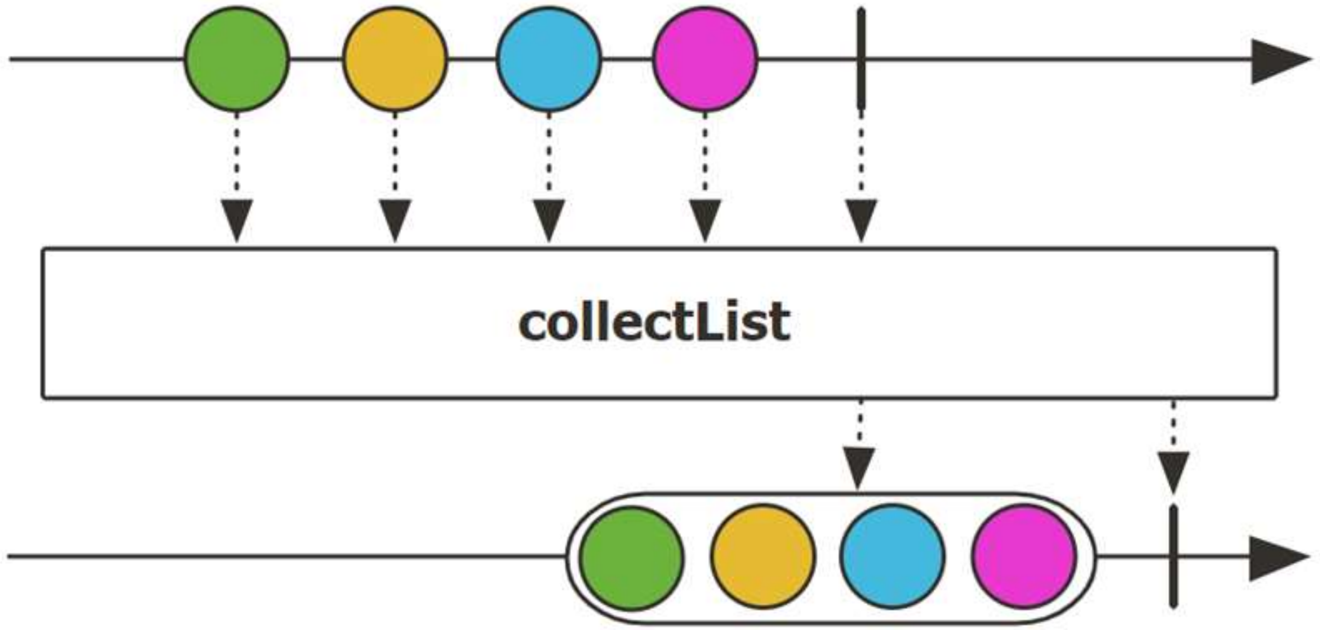


```sql
SELECT o.* FROM users u
WHERE o.dob < ...
ORDER BY o.dob
LIMIT 10 OFFSET 10;

---

SELECT * FROM orders o
LEFT JOIN order_items oi ON oi.order_id = o.id
WHERE o.user_id IN (?, ?, ?)
```

## collectList

```
public final Mono<List<T>> collectList()
```

Collect all elements emitted by this `Flux` into a `List` that is emitted by the resulting `Mono` when this sequence completes, emitting the empty `List` if the sequence was empty.



**Discard Support:** This operator discards the elements in the `List` upon cancellation or error triggered by a data signal.

**Returns:**

a `Mono` of a `List` of all values from this `Flux`

# Relations in Reactive World. Conclusion

1. In general, there are a couple of ways to do so, and all of them have some flaws

# Relations in Reactive World. Conclusion

1. In general, there are a couple of ways to do so, and all of them have some flaws

2. However, in general, it is possible to implement, but again, considering the trade-offs.

# Relations in Reactive World. Conclusion

1. In general, there are a couple of ways to do so, and all of them have some flaws

2. However, in general, it is possible to implement, but again, considering the trade-offs.

3. However, first level of nesting can be solved relatively easy

# Relations in Reactive World. Conclusion

1. In general, there are a couple of ways to do so, and all of them have some flaws

2. However, in general, it is possible to implement, but again, considering the trade-offs.

3. However, first level of nesting can be solved relatively easy

4. Virtual threads and Cursors can be a good option.

Thank you for your attention!

# Поливаха Михаил

- Active OpenSource member (including Spring projects)
- Technical Writer
- Spring Айо Community board member

Contacts:
- Telegram: @mipo256
- GitHub: mipo256
- Blog: mpolivaha.com
- Email: mikhailpolivakha@gmail.com