RED HAT
DEVELOPERS

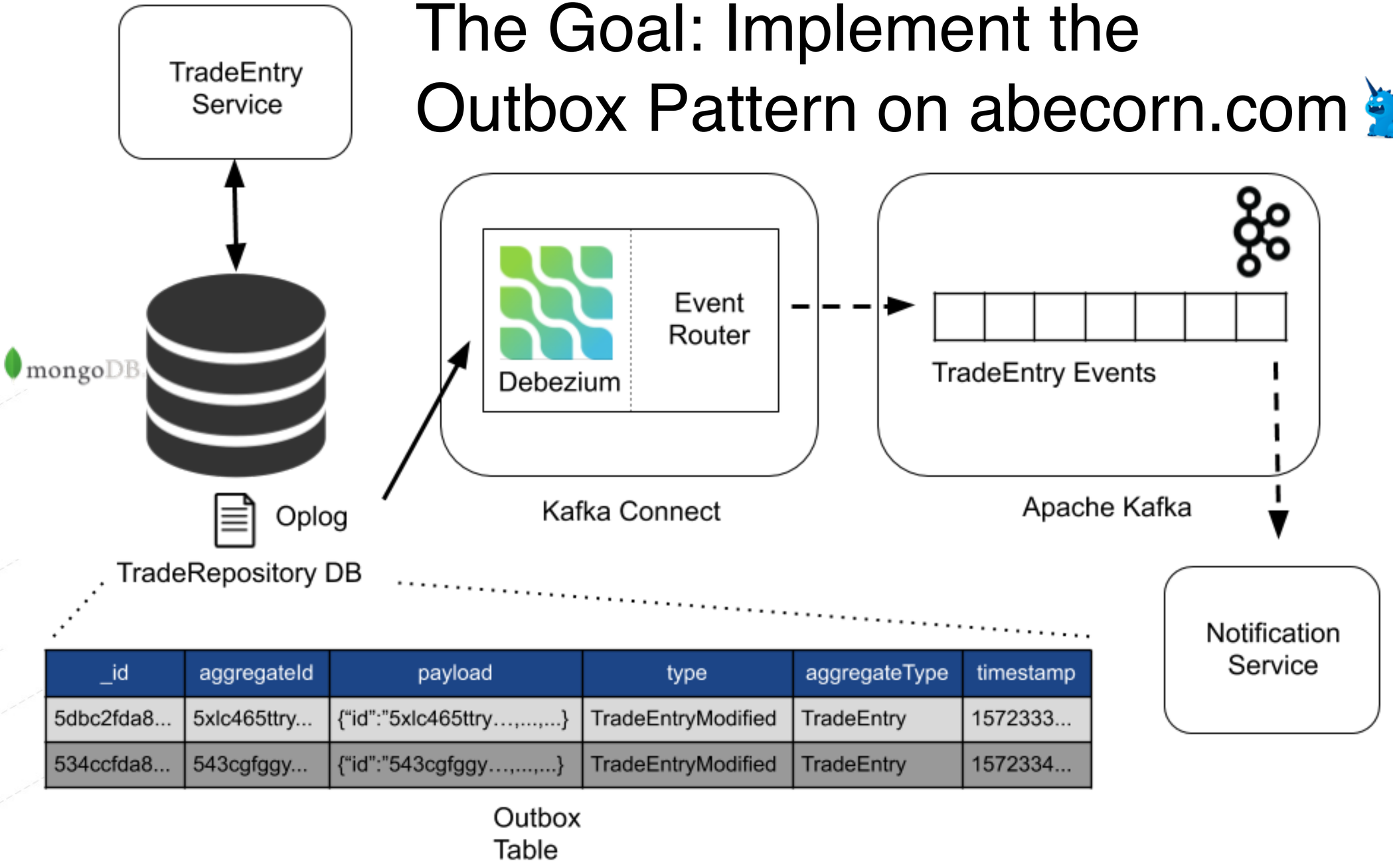
CQRS and Event Sourcing with Openshift 4

Dean Peterson
Specialist Solution Architect
dpeterso@redhat.com

Agenda

- Description of Demo
- What is Event Sourcing
- What is CQRS
- Why Event Sourcing and CQRS?
- Implementation and Demo

The Goal: Implement the Outbox Pattern on abecorn.com

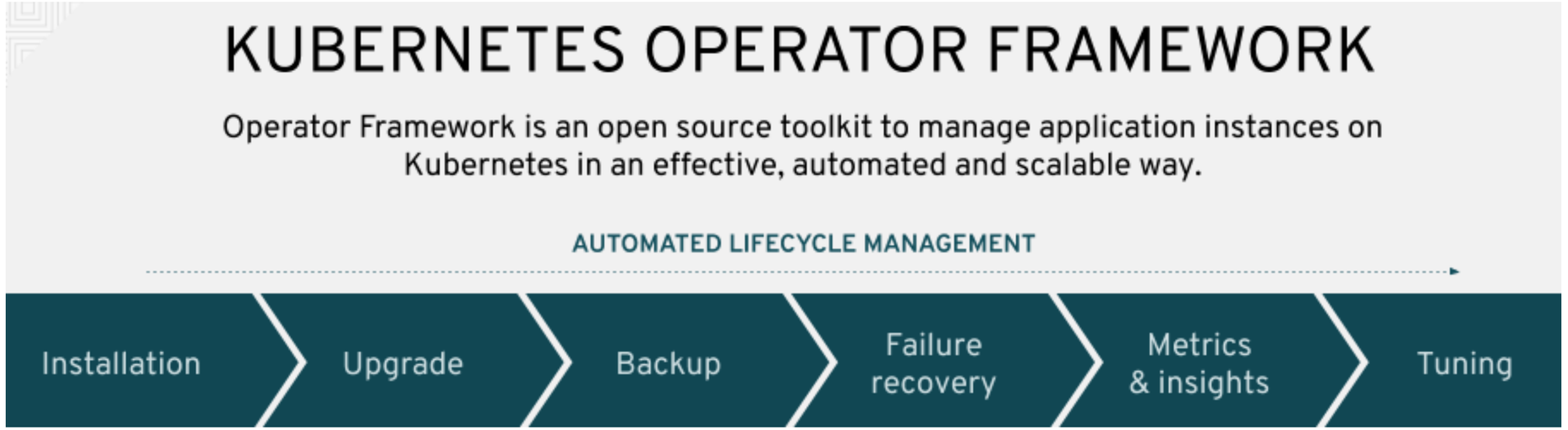


_id	aggregateId	payload	type	aggregateType	timestamp
5dbc2fda8...	5xlc465ttry...	{“id”:”5xlc465ttry…,...,...}	TradeEntryModified	TradeEntry	1572333...
534ccfda8...	543cgfggy...	{“id”:”543cgfggy…,...,...}	TradeEntryModified	TradeEntry	1572334...

Outbox Table

Operator Framework

- Operator SDK
  - Developers build, package and test an operator
  - No knowledge of Kubernetes API complexities required
- Operator Lifecycle Manager
  - Helps install, update, manage the lifecycle of all of the operators in cluster
- Operator Metering
  - Usage reporting for operators and resources within Kubernetes



How was data managed 10 years ago?

Terrified about Entity Beans

Hibernate to the rescue!

Replacing XML with @Annotations

CQS
(Command-Query Separation)

“Asking a question should not change the answer”
(Bertrand Meyer)

POJOs as an (Anemic) Domain Model

Event Sourcing

Account

ID	CUSTOMER_ID	BALANCE
1001	990	1000
1002	991	0
1003	991	-500
1004	992	300

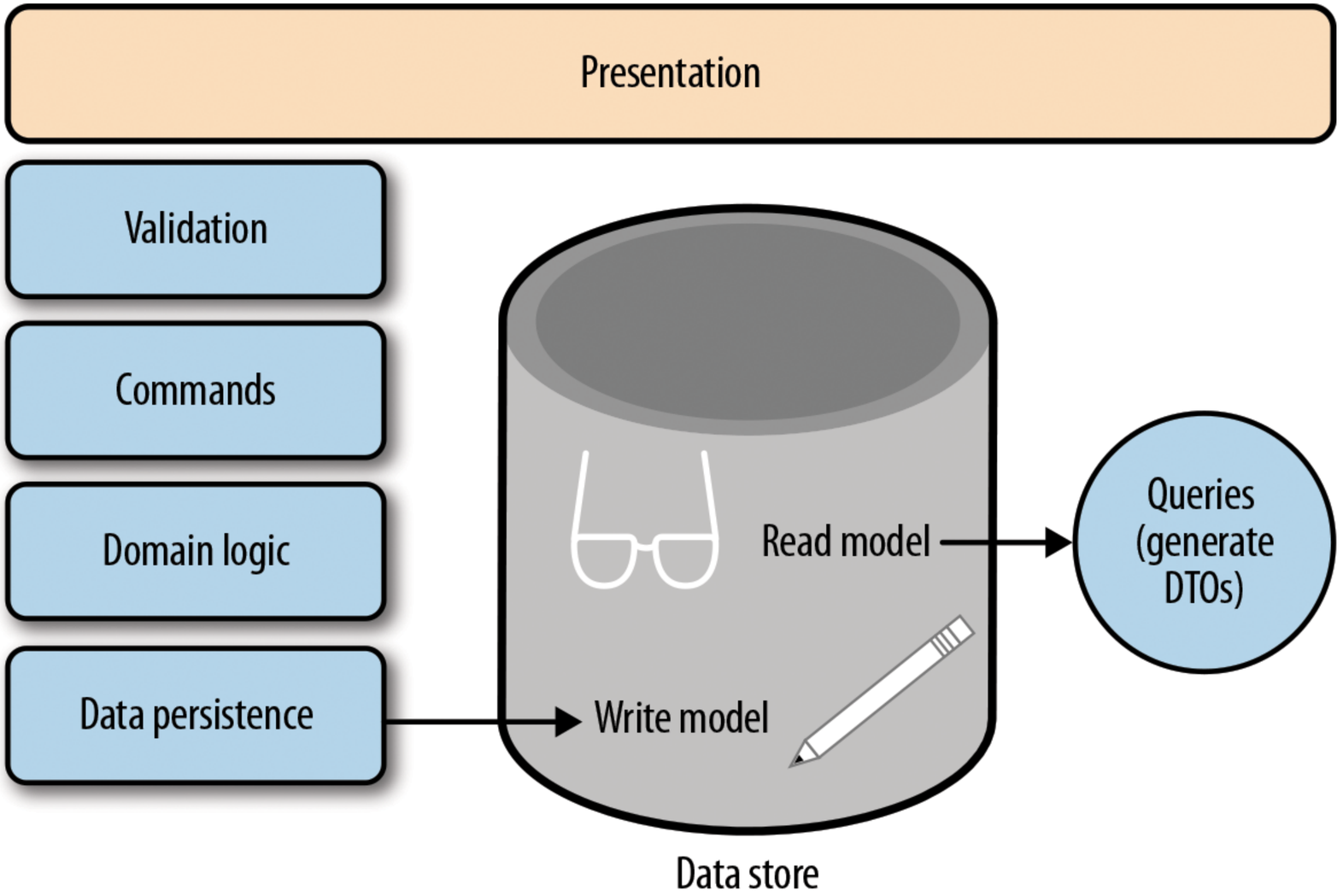
Transactions

ID	ACCOUNT_ID	TIMESTAMP	OP	AMOUNT
1	1001	1234567890	C	1000
2	1002	1234567891	C	200
3	1001	1234567900	D	300
4	1001	1234567995	D	150

Enables you to think in the **Events** that happened in the system

CQRS
(Command Query Responsibility Segregation)

CQRS (Command Query Responsibility Segregation)



ID	NAME	PHONE	ADDRESS	BIRTH
1	Burr	222-222-2323	901 South St	12/12/1968
2	Edson	222-333-3434	112 North Dr	03/03/1978
3	John	111-456-4545	666 Iron St	06/06/1966
4	Doe	333-789-7890	777 Boeing Dr	07/07/1977

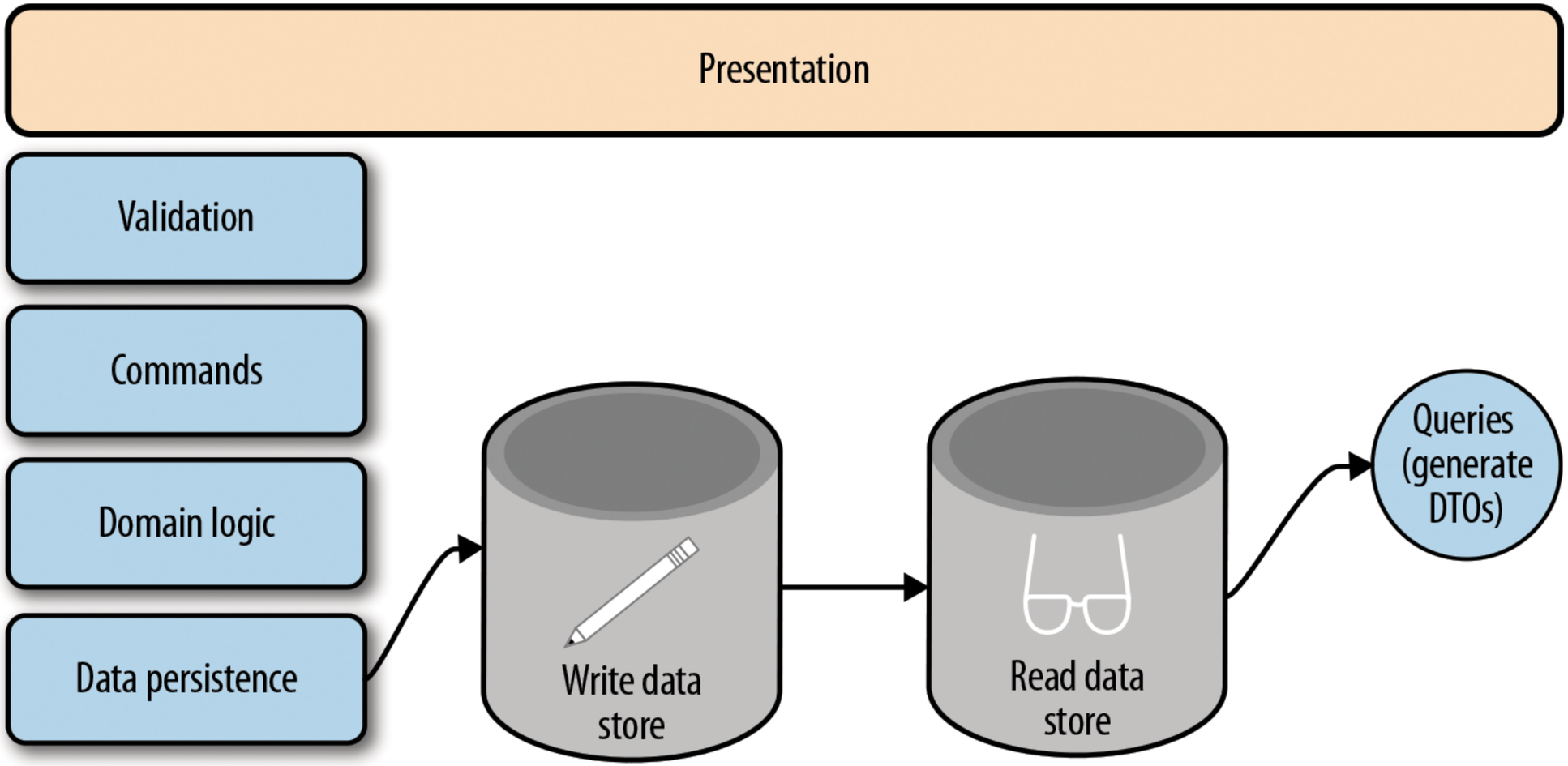
```
INSERT INTO CUSTOMER(ID,NAME,PHONE,ADDRESS,BIRTH);

SELECT * FROM CUSTOMER;
```

```
SELECT ID, NAME, PHONE FROM CUSTOMER;
```

```
SELECT ID, NAME, ADDRESS FROM CUSTOMER;
```

CQRS with separate data stores



CQRS & Event Sourcing

ID	CUSTOMER_ID	BALANCE
1001	990	1000
1002	991	0
1003	991	-500
1004	992	300

ID	ACCOUNT_ID	TIMESTAMP	OP	AMOUNT
1	1001	1234567890	C	1000
2	1002	1234567891	C	200
3	1001	1234567900	D	300
4	1001	1234567995	D	150

Account

Transactions

ID	CUSTOMER_ID	BALANCE
1001	990	1000
1002	991	0
1003	991	-500
1004	992	300

ID	ACCOUNT_ID	TIMESTAMP	OP	AMOUNT
1	1001	1234567890	C	1000
2	1002	1234567891	C	200
3	1001	1234567900	D	300
4	1001	1234567995	D	150

READ MODEL

Account

Transactions

WRITE MODEL

Why CQRS?

Performance

Distribution
Availability
Integration
Analytics

```
SELECT ID, NAME, AGE, AVG_BILL
  FROM CUSTOMER_REPORT_VIEW;
```

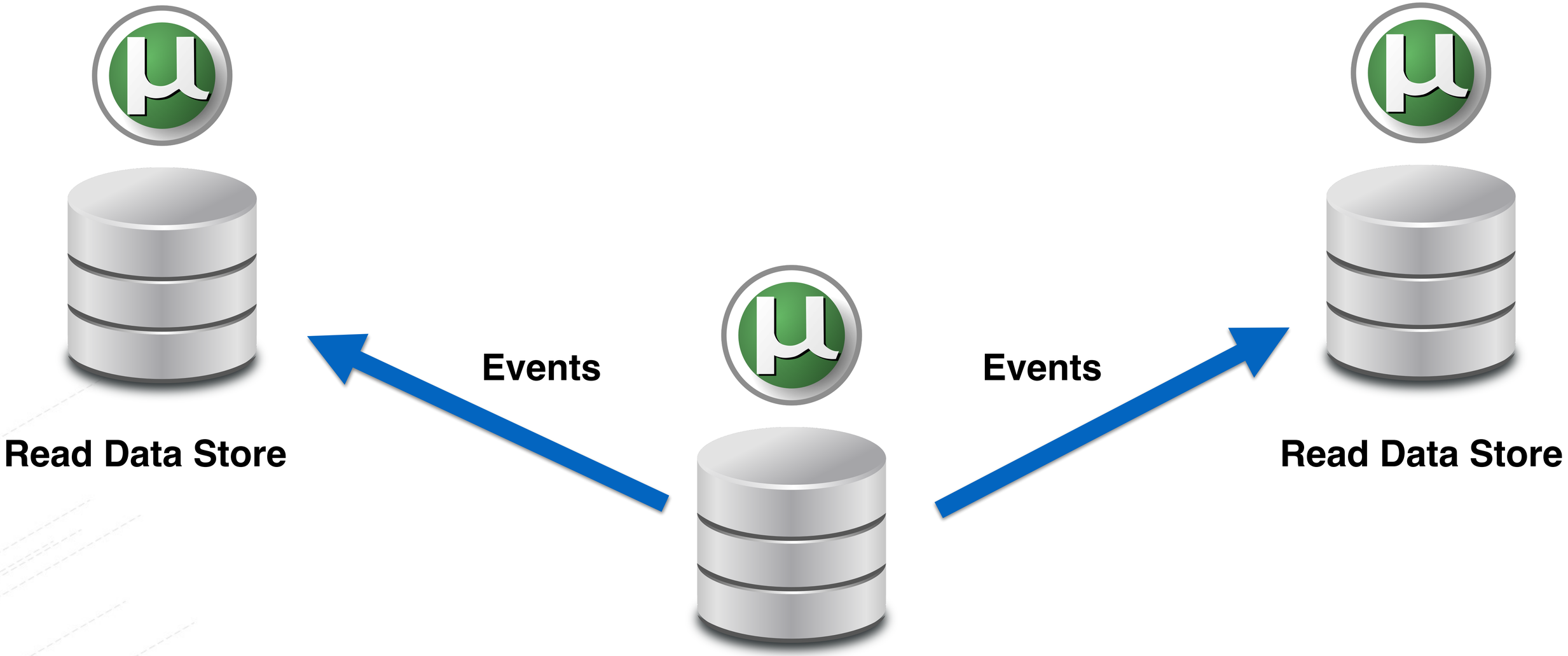
```
SELECT ID, PHONE, LAST_PAYMENT_AMOUNT
  FROM CUSTOMER_BILLING_VIEW;
```

Canonical Source of Information
(Write Data Store)

Read Data Store

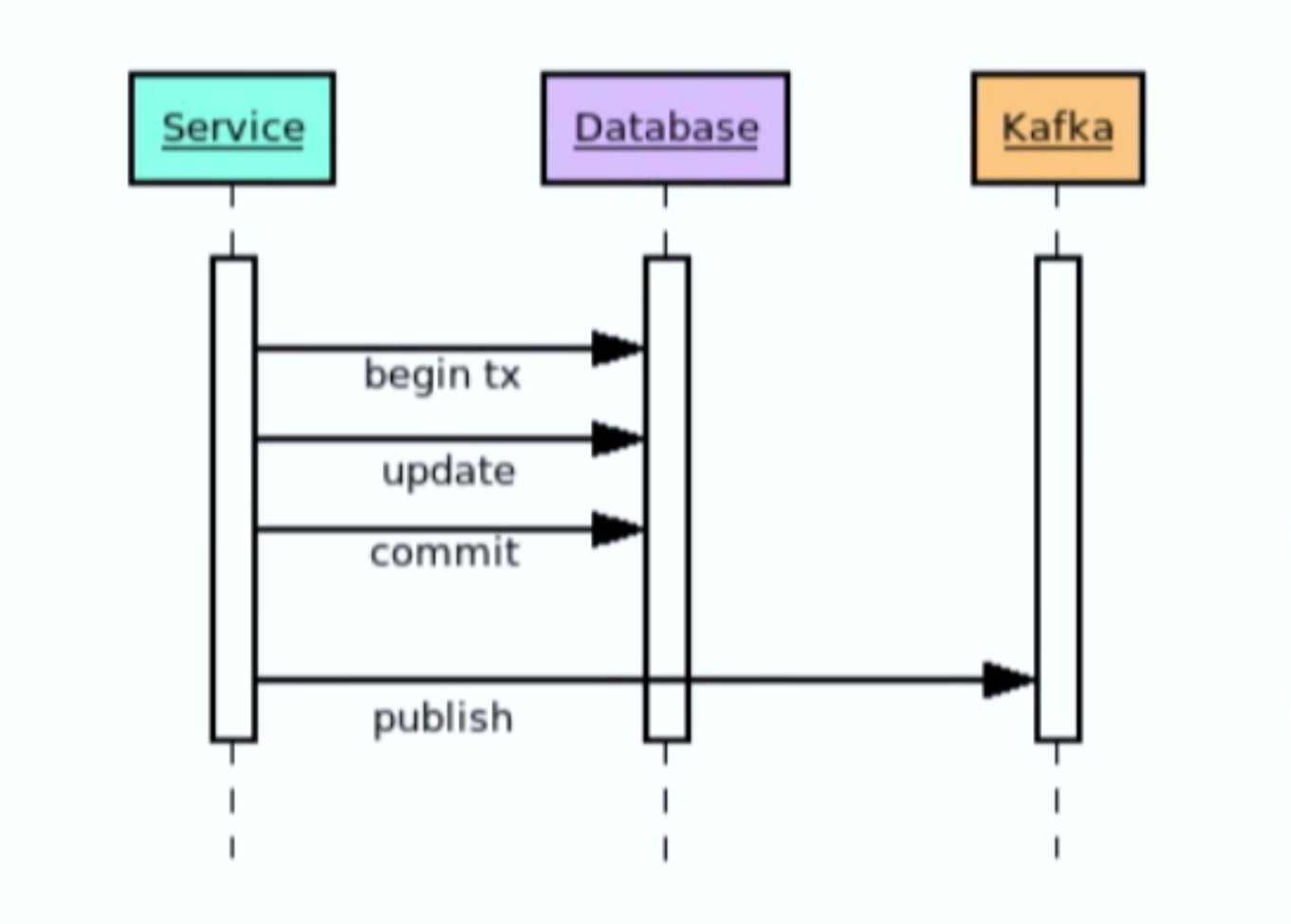
Canonical Source of Information
(Write Data Store)

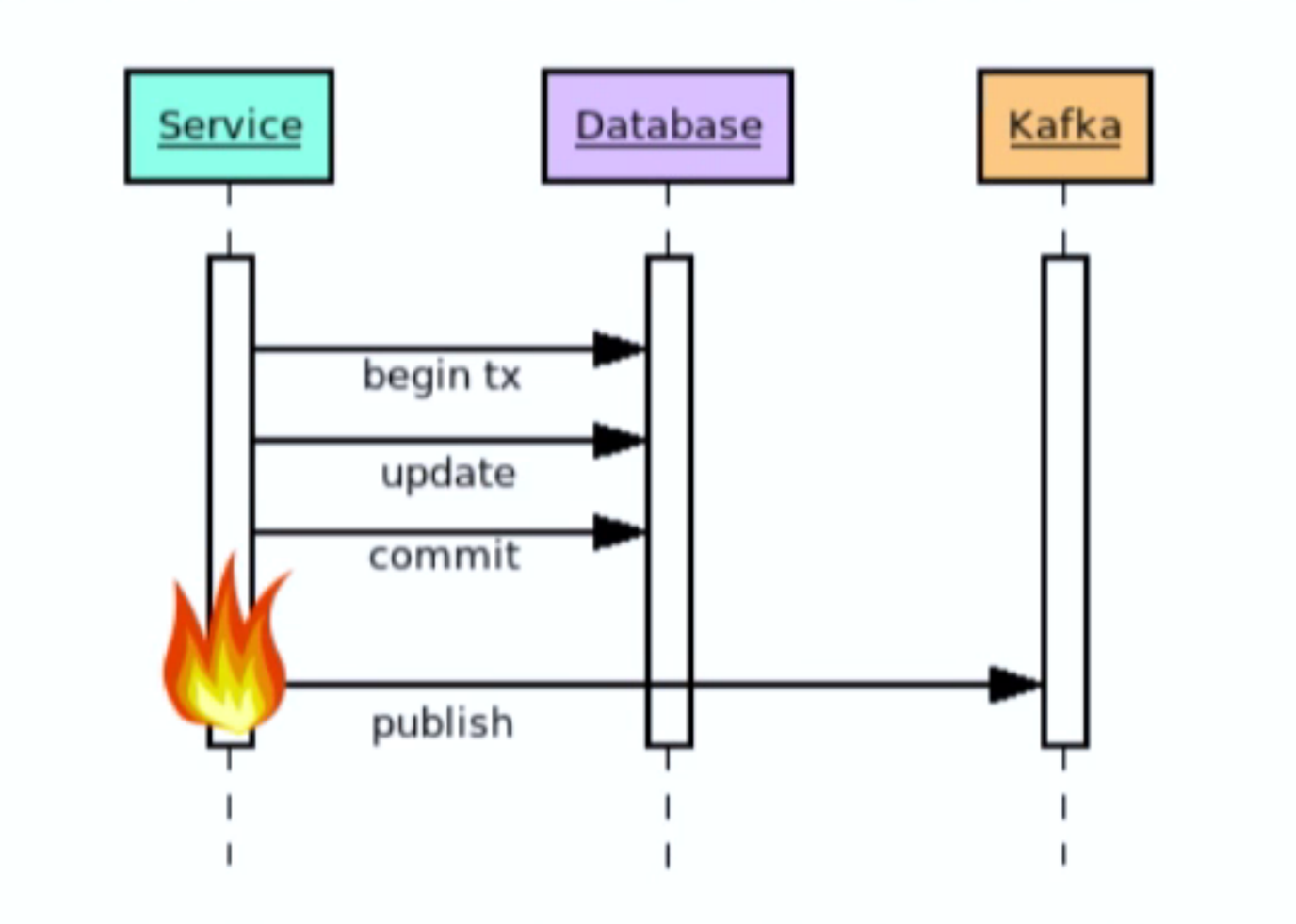
Read Data Store

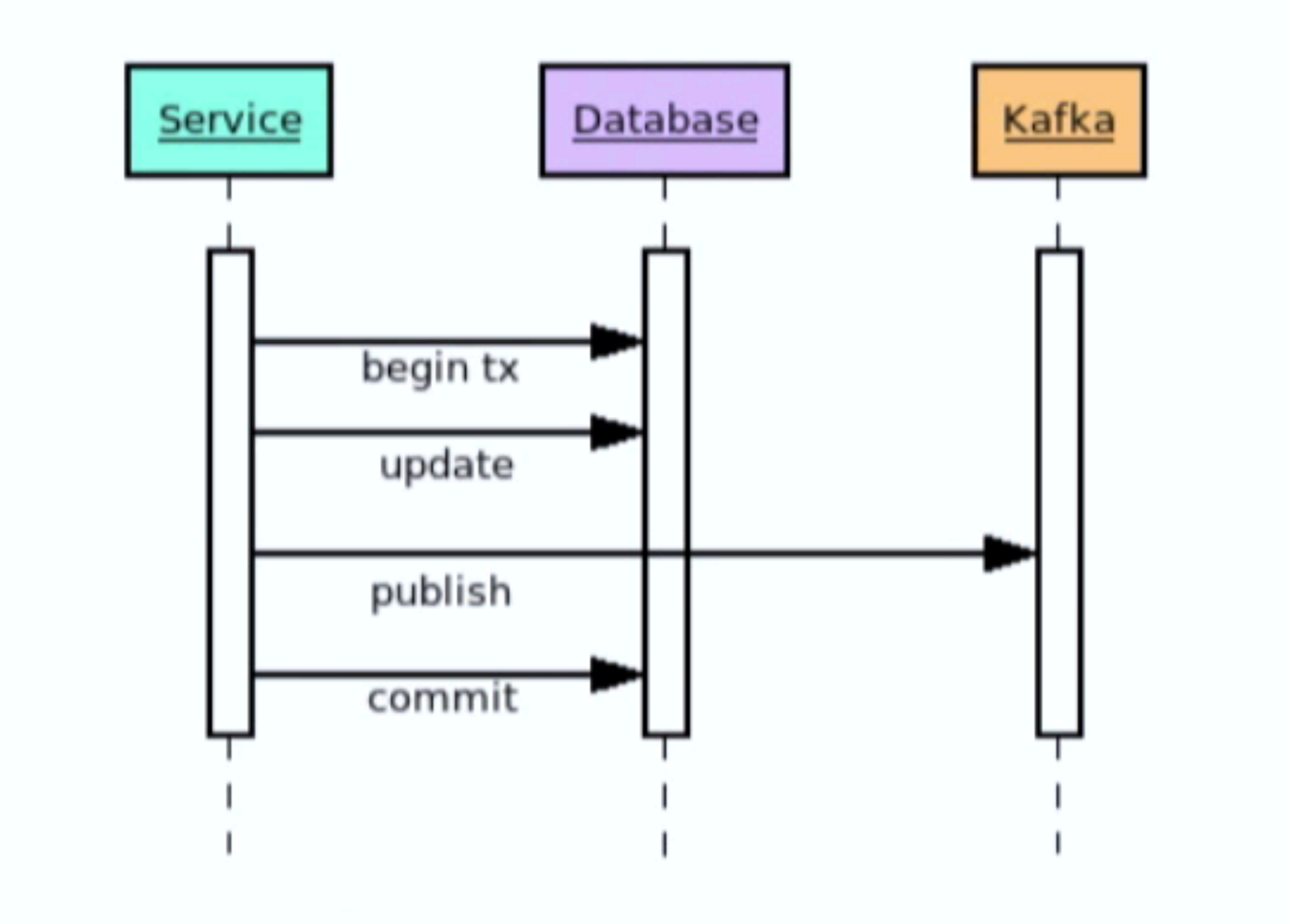

Events
Events
Read Data Store
Read Data Store
Canonical Source of Information
(Write Data Store)

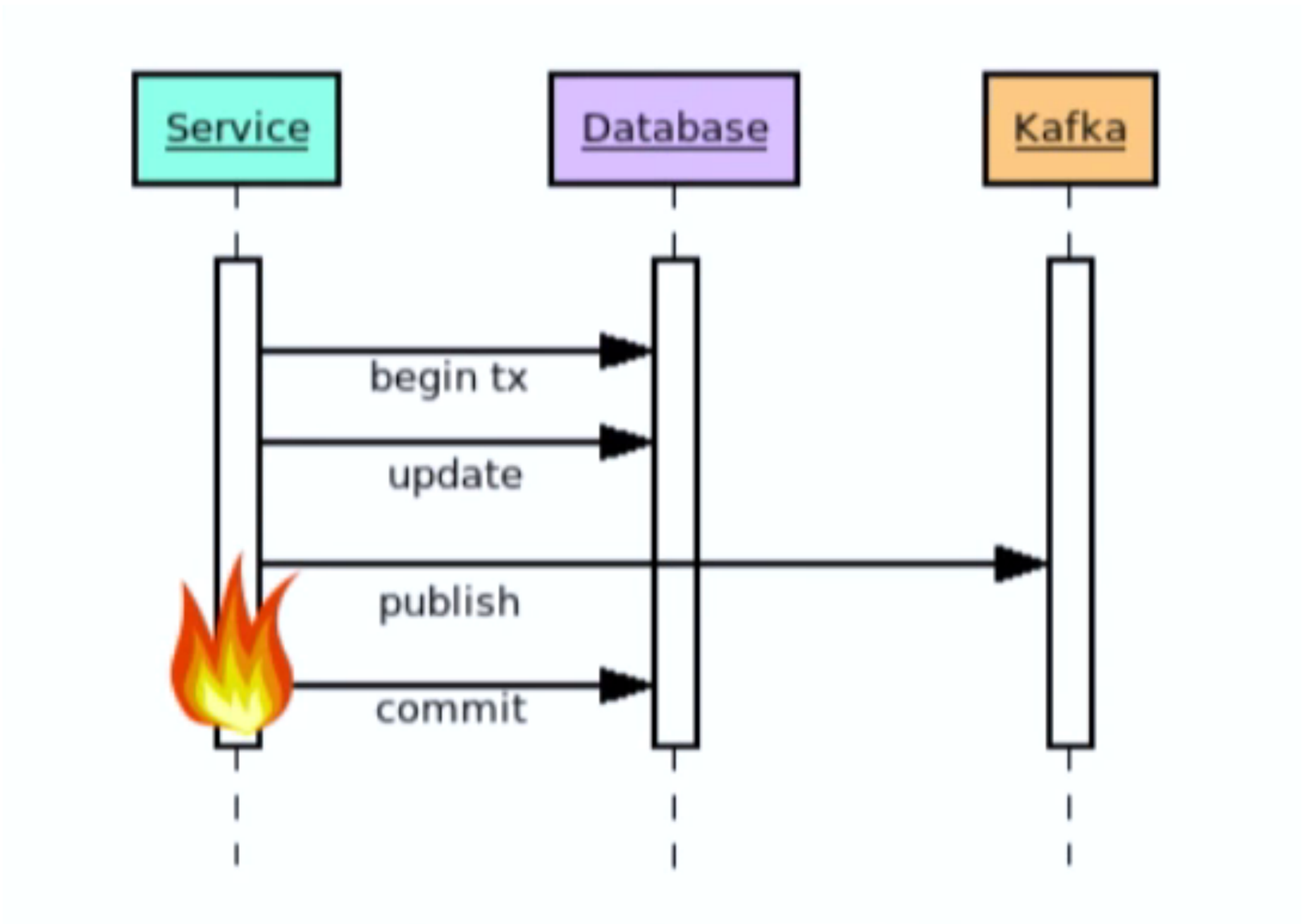
Different choices of technology depending on the requirements

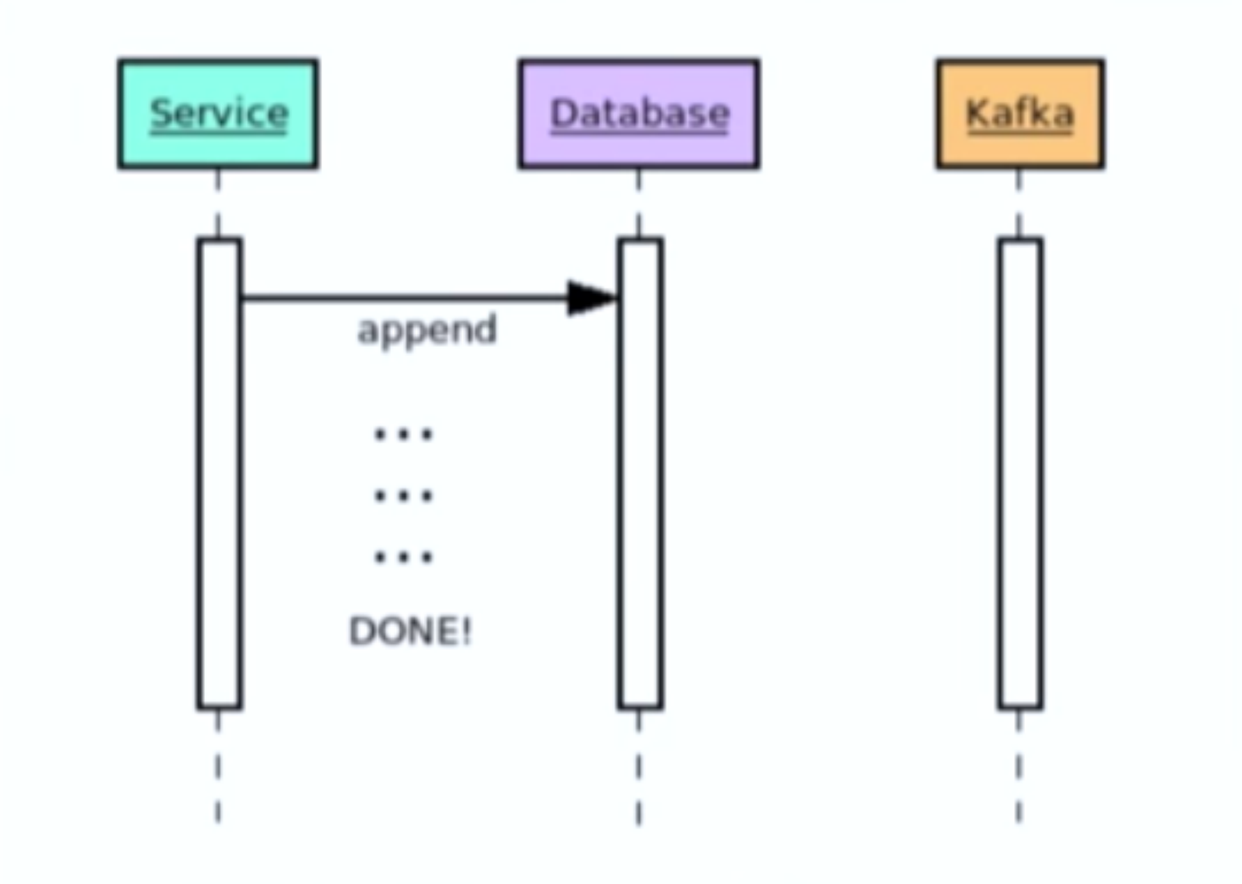
Latency
Size
Staleness
Ownership
Security
Type?

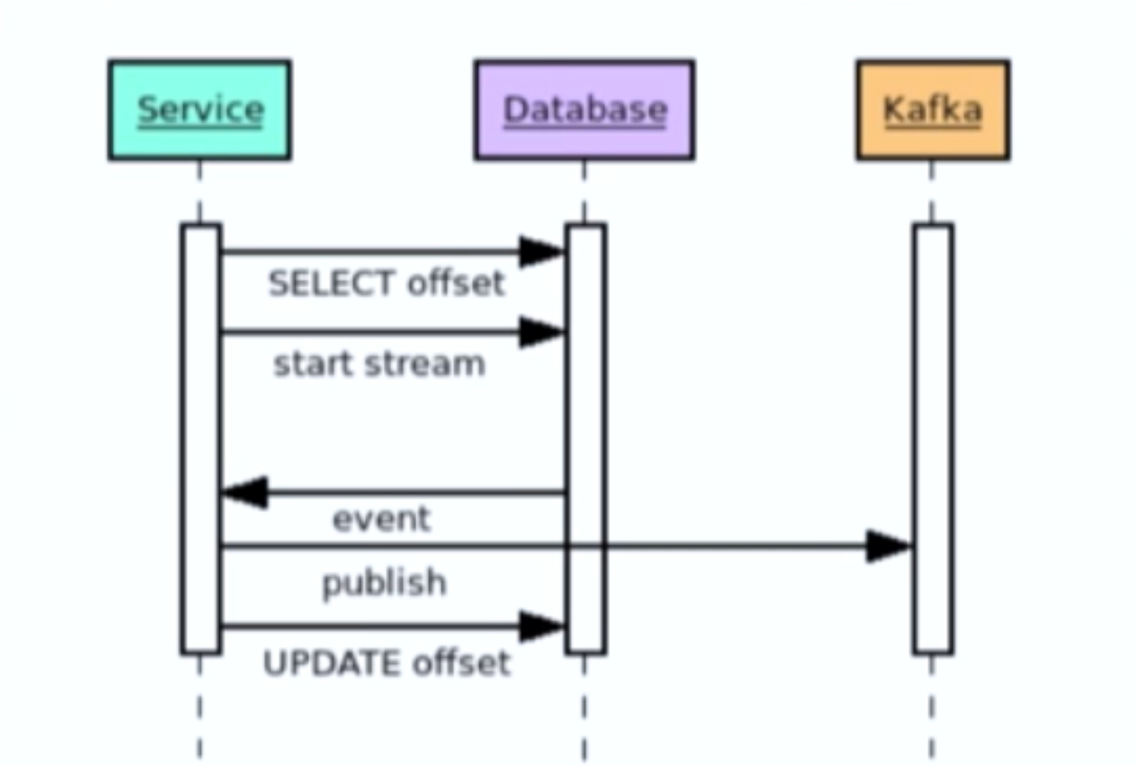












Back to the Demo

Message Brokers

http://activemq.apache.org/

https://kafka.apache.org/

Change Data Capture

debezium

http://debezium.io/

```
118     @Inject
119     private ShippingRuleRepository shippingRuleRepository;
120
121     @Inject
122     private Event<ExportedEvent> exportEvent;
123
124     public TradeEntry persist(TradeEntry tradeEntry) {
125         if (tradeEntry.getCondition() != null) {
126             // Clean the html for condition
127
128             Whitelist whiteList = Whitelist.relaxed().addProtocols("img", "src", "http", "https");
129             whiteList.addAttributes("span", "style");
130
131             String safe = Jsoup.clean(tradeEntry.getCondition(), whiteList);
132             tradeEntry.setCondition(safe);
133         }
134
135         if (tradeEntry.getCountry() == null || tradeEntry.getCountry().isEmpty()) {
136             tradeEntry.setCountry("US");
137         }
138
139         if (tradeEntry.getId() != null) {
140             em.merge(tradeEntry);
141         } else {
142             em.persist(tradeEntry);
143         }
144
145         itemEventSrc.fire(tradeEntry);
146         if(tradeEntry.isModified())
147         {
148             exportEvent.fire(TradeEntryModifiedEvent.of(tradeEntry));
149         }
150         return tradeEntry;
151     }
```

```java
public class TradeEntryModifiedEvent implements ExportedEvent {

    private static ObjectMapper mapper = new ObjectMapper();

    private final String id;
    private final String tradeEntry;
    private final Long timestamp;

    private TradeEntryModifiedEvent(String id, String tradeEntry) {
        this.id = id;
        this.tradeEntry = tradeEntry;
        this.timestamp = (new Date()).getTime();
    }

    public static TradeEntryModifiedEvent of(TradeEntry tradeEntry) {
        ObjectNode asJson = mapper.createObjectNode()
                .put("id", tradeEntry.getId());


        return new TradeEntryModifiedEvent(tradeEntry.getId(), asJson.toString());
    }

    @Override
    public String getAggregateId() {
        return id;
    }

    @Override
    public String getAggregateType() {
        return "TradeEntry";
    }

    @Override
    public String getType() {
        return "TradeEntryModified";
    }

    @Override
    public Long getTimestamp() {
        return timestamp;
    }

    @Override
    public String getPayload() {
        return tradeEntry;
    }
}
```

```java
@ApplicationScoped
public class EventSender {

    @Inject
     private EntityManager entityManager;

     public void onExportedEvent(@Observes ExportedEvent event) {
         OutboxEvent outboxEvent = new OutboxEvent(
                 event.getAggregateType(),
                 event.getAggregateId(),
                 event.getType(),
                 event.getPayload(),
                 event.getTimestamp()
         );

         // This will produce an INSERT followed by a DELETE;
         // So the events table will always be empty, but still both events will be captured from
         // the log by Debezium (and the latter will be ignored)
         entityManager.persist(outboxEvent);
         entityManager.remove(outboxEvent);
     }
}
```

```java
@JsonIgnoreProperties(ignoreUnknown = true)
@Entity
public class OutboxEvent {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Type(type = "objectid")
    private String id;

    @NotNull
    private String aggregateType;

    @NotNull
    private String aggregateId;

    @NotNull
    private String type;

    @NotNull
    private Long timestamp;

    private String payload;

    OutboxEvent() {
    }

    public OutboxEvent(String aggregateType, String aggregateId, String type, String payload, Long timestamp) {
        this.aggregateType = aggregateType;
        this.aggregateId = aggregateId;
        this.type = type;
        this.payload = payload;
        this.timestamp = timestamp;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getAggregateId() {
        return aggregateId;
    }

    public void setAggregateId(String aggregateId) {
        this.aggregateId = aggregateId;
    }

    public String getAggregateType() {
```

RED HAT
DEVELOPERS

Thank you!


plus.google.com/+RedHat


facebook.com/redhatinc


linkedin.com/company/red-hat


twitter.com/RedHatNews


youtube.com/user/RedHatVideos