EVENT DRIVEN ARCHITECTURES
WITH APACHE KAFKA

BJORN.BESKOW@CALLISTAENTERPRISE.SE

CADEC 2019.01.24 & 2019.01.30 | CALLISTAENTERPRISE.SE

CALLISTA

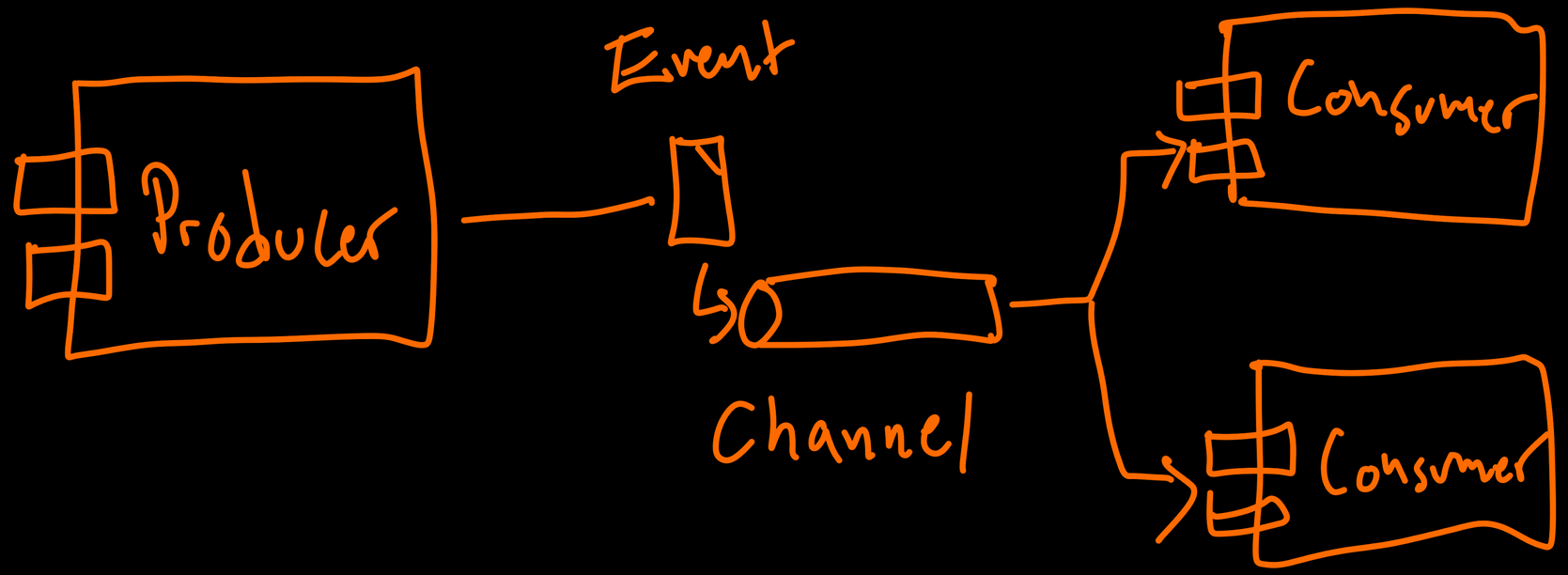
— ENTERPRISE —

AGENDA

- Short background
  - Events
  - Apache Kafka
- *Event Notification*
  - What? Why? How?
- *Event-Carried State Transfer*
  - What? Why? How?
- *Event Sourcing*
  - What? Why? How?
- Sum Up

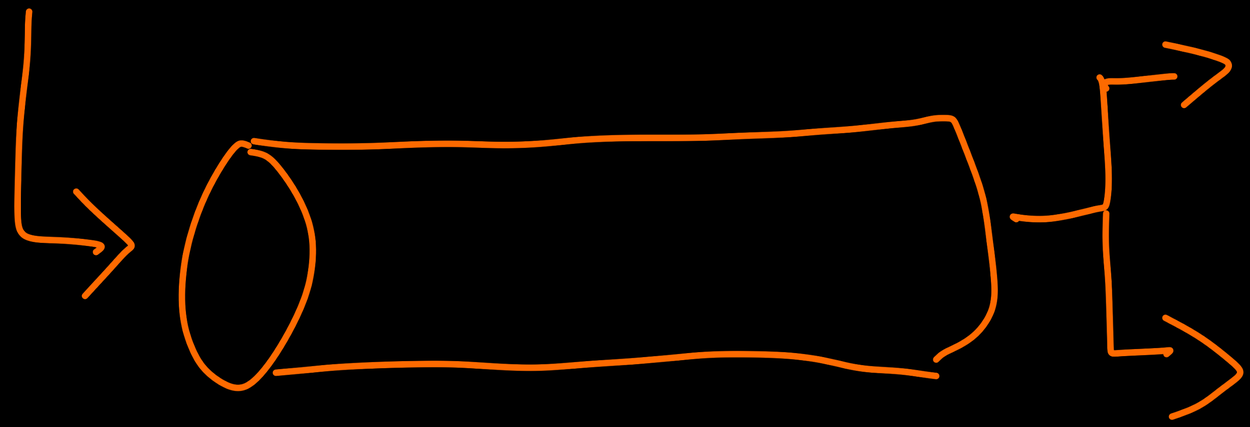
https://martinfowler.com/articles/201701-event-driven.html

EVENTS

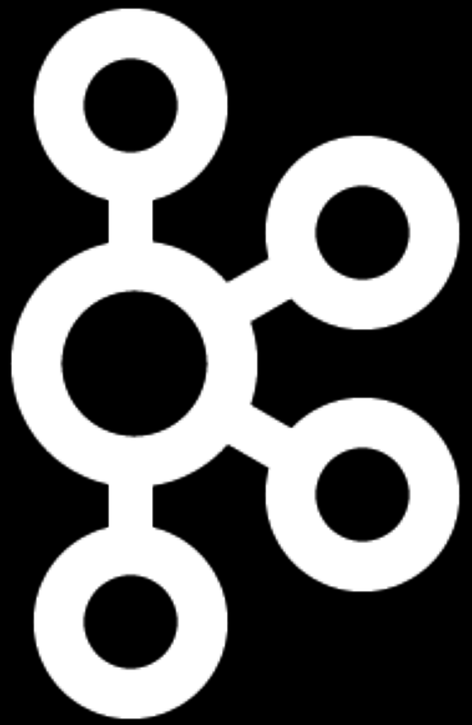


REQUIREMENTS ON AN EVENT MESSAGING BACKBONE

- Robust
- Resilient
- Scalable
- Performant



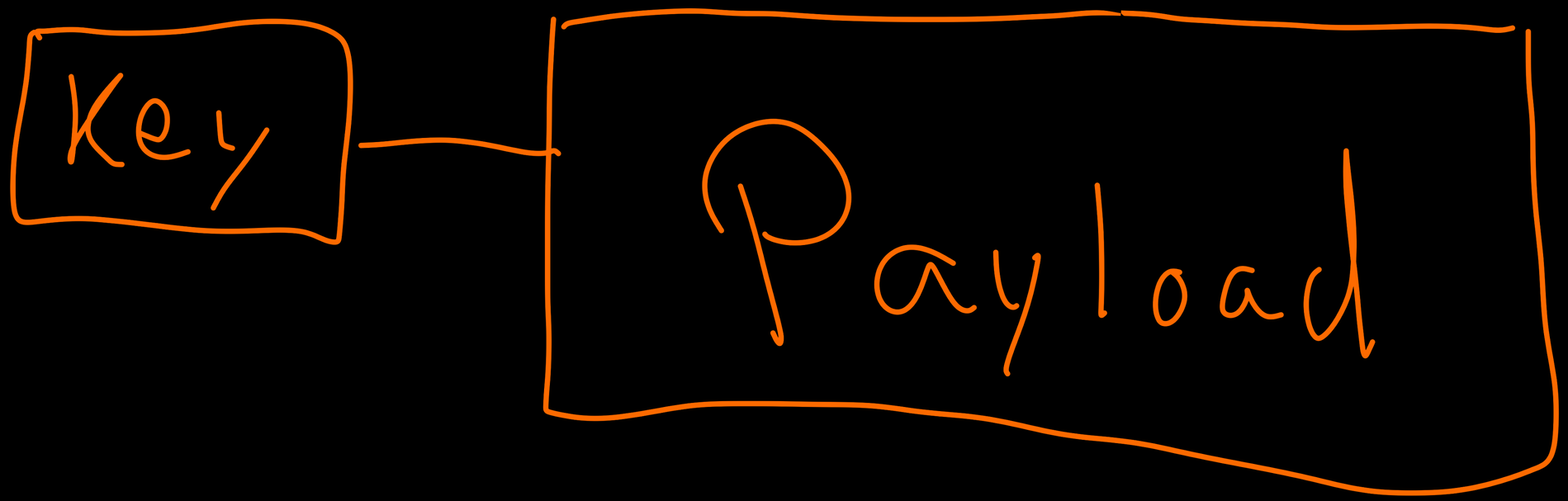
INTRODUCING APACHE KAFKA



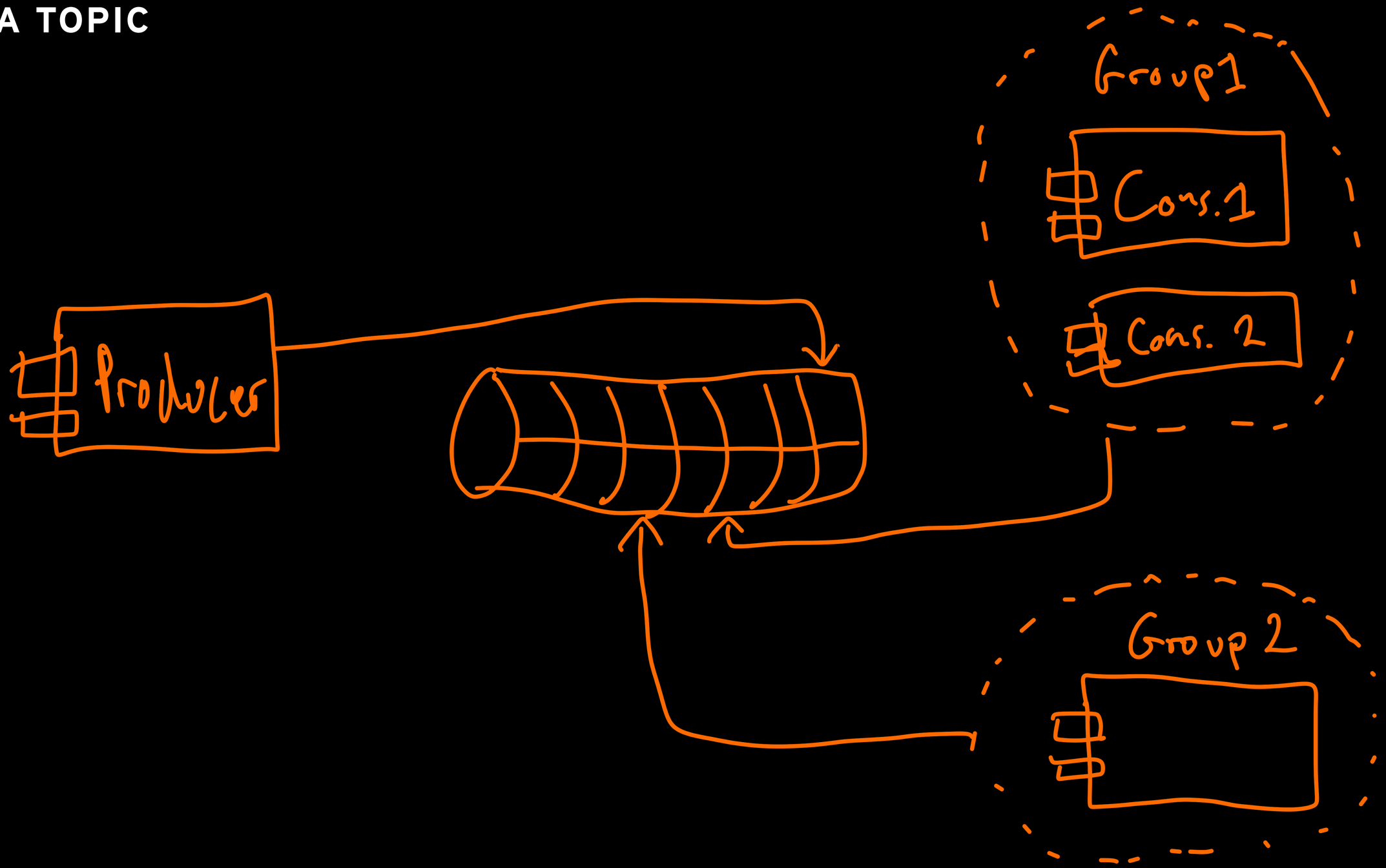
APACHE kafka®

A distributed streaming platform

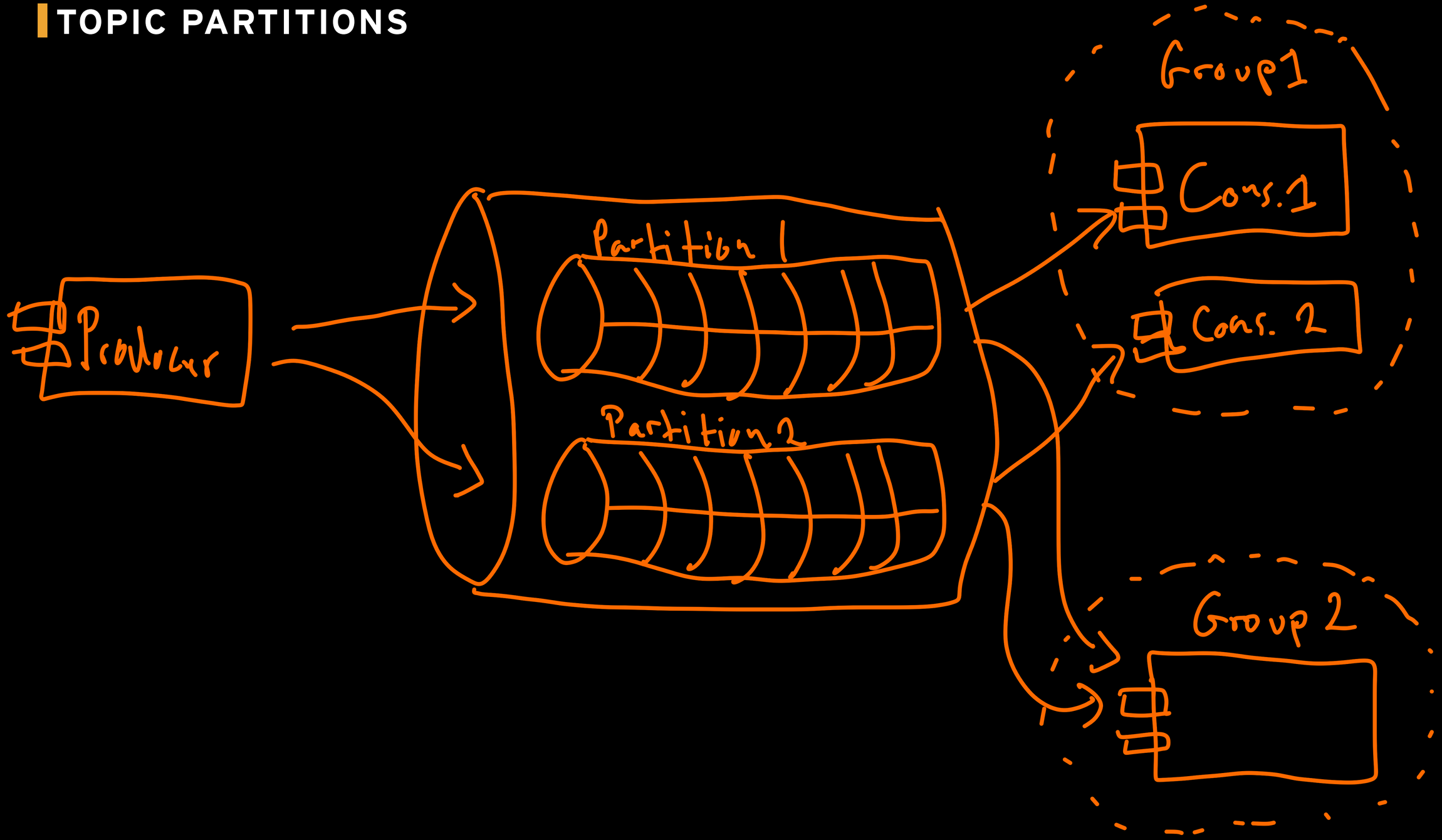
KAFKA RECORD



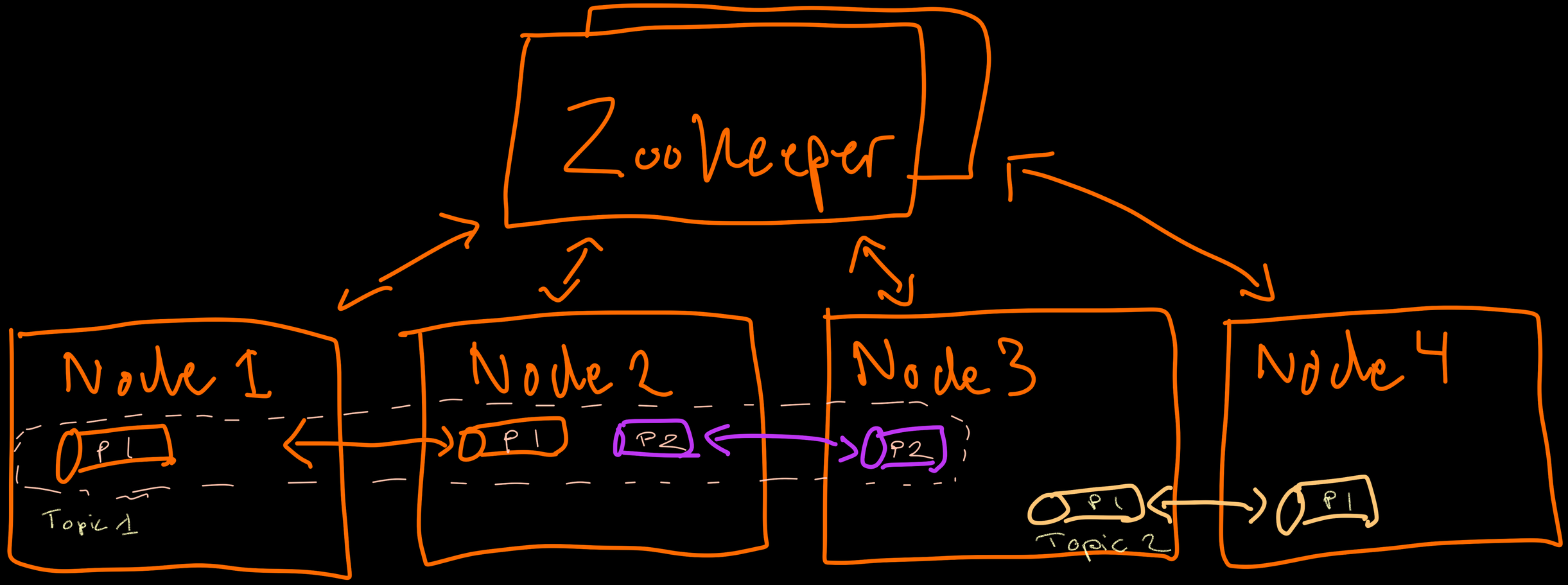
KAFKA TOPIC



TOPIC PARTITIONS



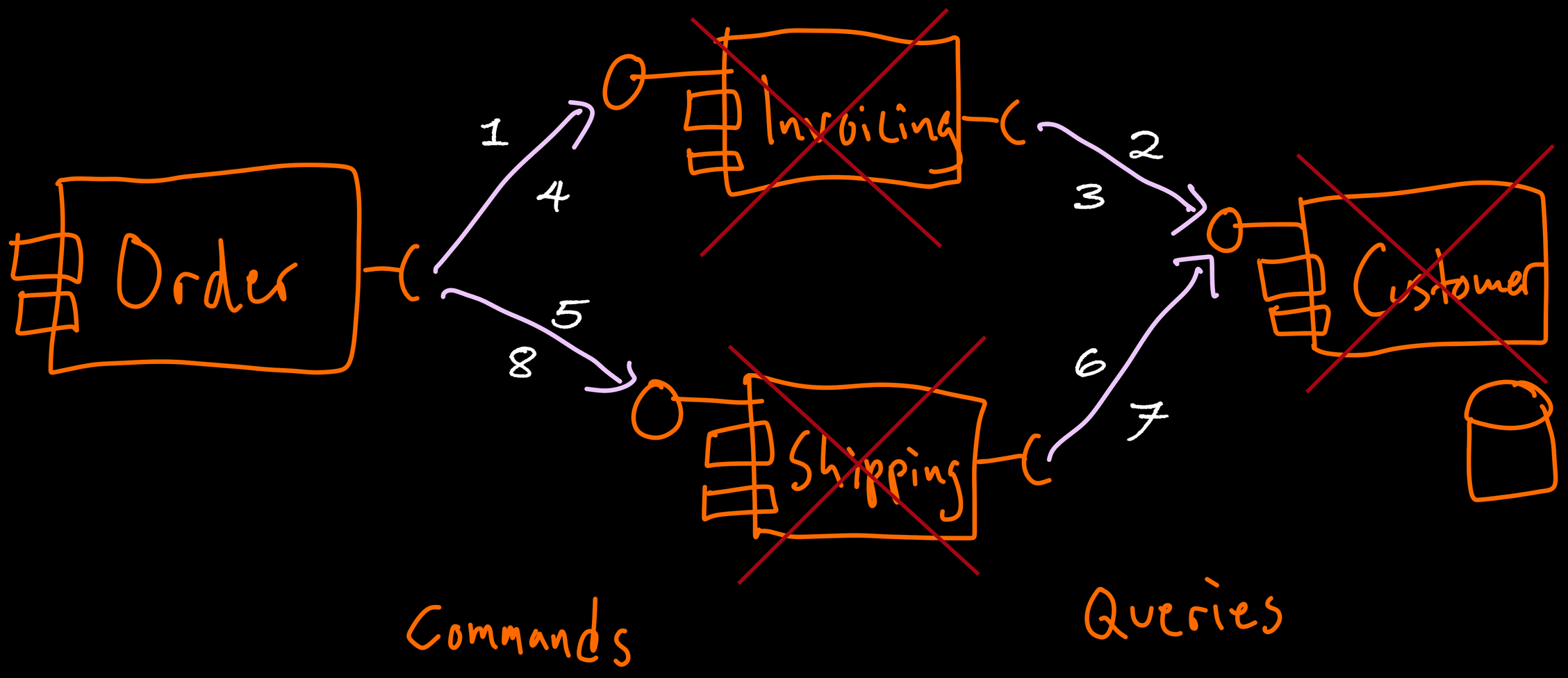
APACHE KAFKA CLUSTERING



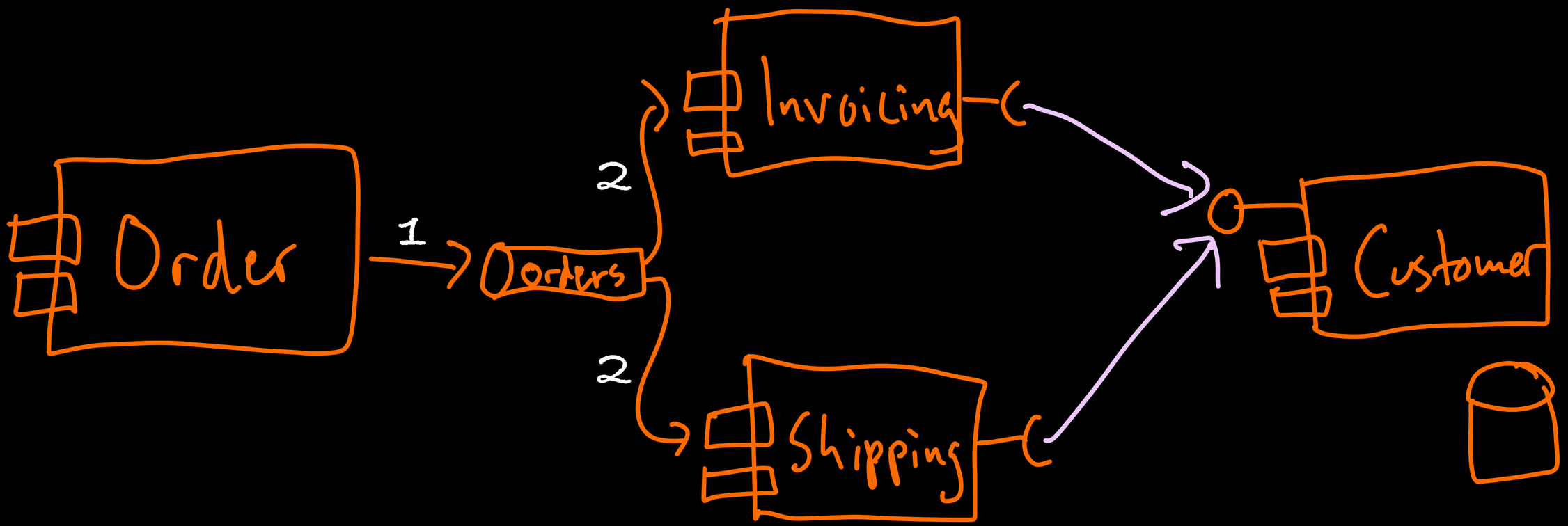
AGENDA

- Short background
  - Events
  - Apache Kafka
- *Event Notification*
  - What? Why? How?
- Event-Carried State Transfer
  - What? Why? How?
- Event Sourcing
  - What? Why? How?
- Sum Up

DEPENDENCIES AND COUPLING

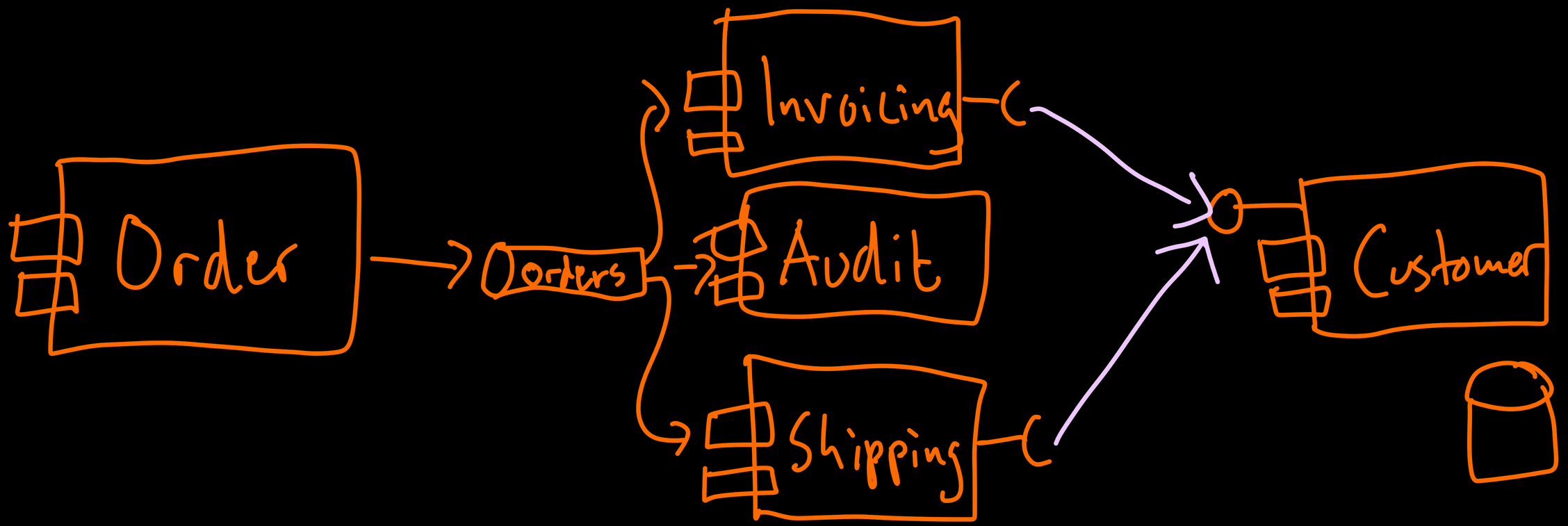


EVENT NOTIFICATION

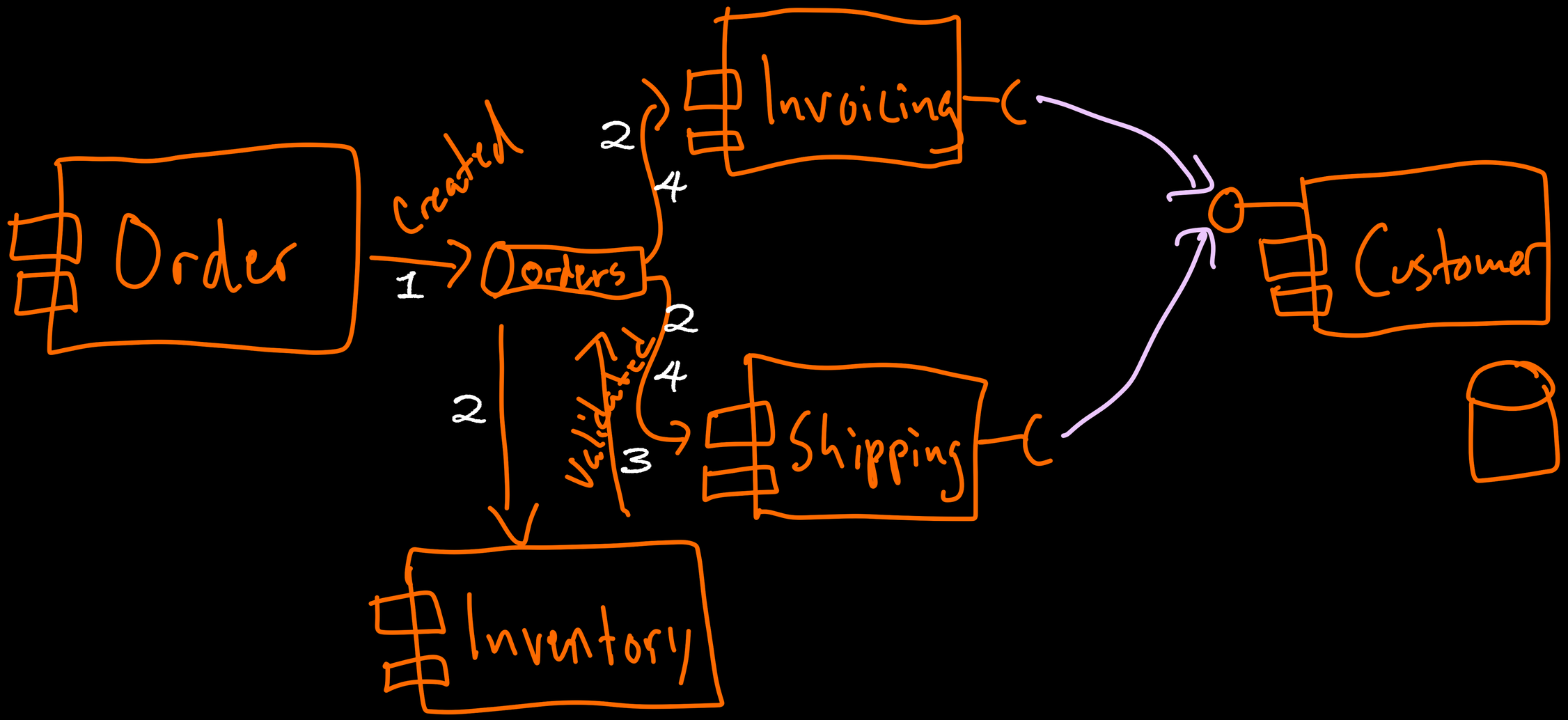


EVENT NOTIFICATION

- Inversion of Control gives flexibility



BUSINESS FLEXIBILITY: CHOREOGRAPHY



EVENT NOTIFICATION: DRIVERS

- Reduced Coupling
- Flexibility & Agility
- Resilience
- Performance & Parallelism

EVENT NOTIFICATION EXAMPLE: REPLACING SERVICE DEPENDENCIES

```
@Autowired
private InvoicingClient invoicing;

@Autowired
private ShippingClient shipping;

public void orderPlaced(Order order) {
  ...
  invoicing.createInvoice(order);
  shipping.createShipping(order);
}
```

EVENT NOTIFICATION EXAMPLE: EVENT PRODUCER

```
@Autowired
private KafkaTemplate<String, Order> kafkaTemplate;

@Value("${kafka.topic.order}")
private String orderTopic;

public void orderPlaced(Order order) {
  ...
  order.setState(CREATED);
  kafkaTemplate.send(orderTopic, order.getOrderNo(), order);
}
```

EVENT NOTIFICATION EXAMPLE: ORDER VALIDATOR

```
@KafkaListener(topics = "${kafka.topic.orders}")
public void receive(Order order) {
  if (order.getState().equals(CREATED)) {
    // Validate order ...
    order.setState(VALIDATED);
    kafkaTemplate.send(orderTopic, order.getOrderNo(), order);
  }
}
```

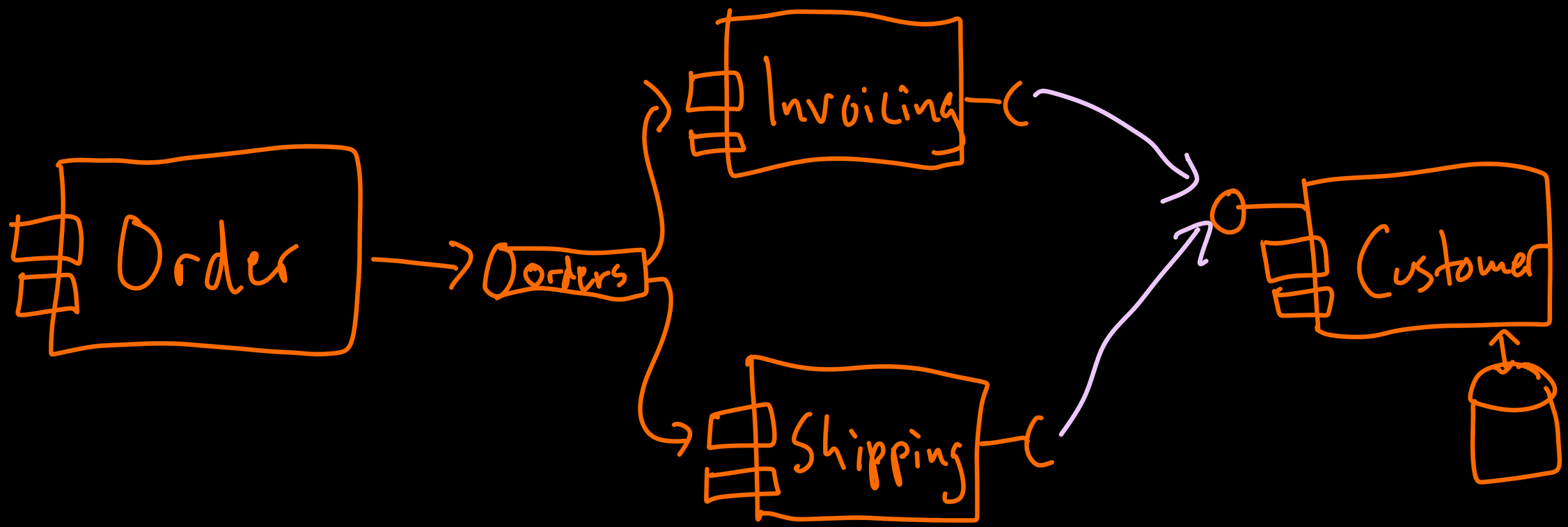
EVENT NOTIFICATION EXAMPLE: EVENT CONSUMER

```
@KafkaListener(topics = "${kafka.topic.orders}")
public void receive(Order order) {
  if (order.getState().equals(VALIDATED)) {
    // Create invoice or shipping …
 }
}
```

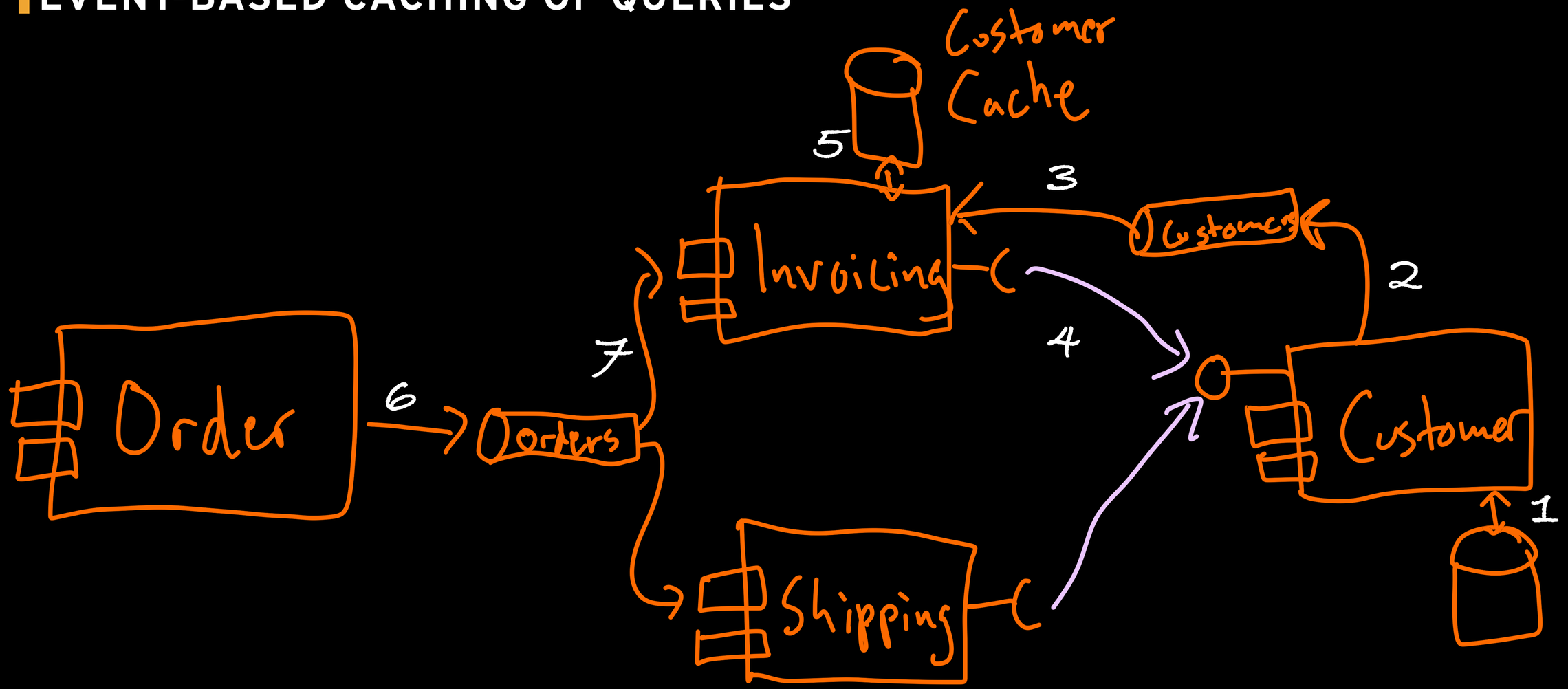
EVENT NOTIFICATION EXAMPLE: TOPIC DEFINITION

```
@Bean
 public NewTopic orderTopic() {
   int partitions = 2;
   short replicationFactor = 3;
   Map<String, String> configs = new HashMap<>();
   configs.put("retention.ms", "60000"); // 1 minute
   configs.put("retention.bytes", "524288"); // 512kb
   return new NewTopic(orderTopic, partitions,
                       replicationFactor).configs(configs);
 }
```

QUERYING FOR DATA



EVENT-BASED CACHING OF QUERIES



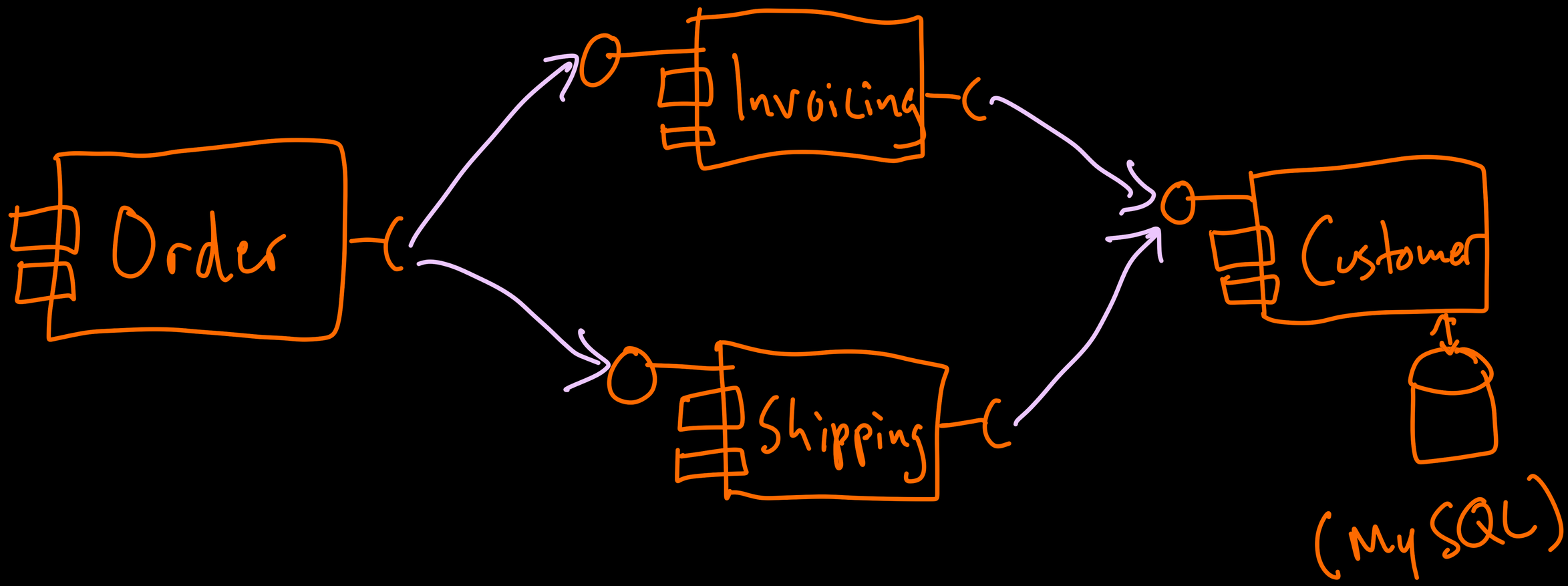
EVENT-BASED CACHING OF QUERIES EXAMPLE: EVENT PRODUCER

```
@PostMapping(value = "/customer")
public Customer create(@RequestBody Customer customer) {
  ...
  customerRepository.save(customer);
  customerEventSender.send(customer.getId(), CREATED);
  ...
}

@DeleteMapping(value = "/customer/{id}")
public void deleteById(@PathVariable String id) {
  ...
  customerRepository.deleteById(id);
  customerEventSender.send(id, DELETED);
  ...
}
```

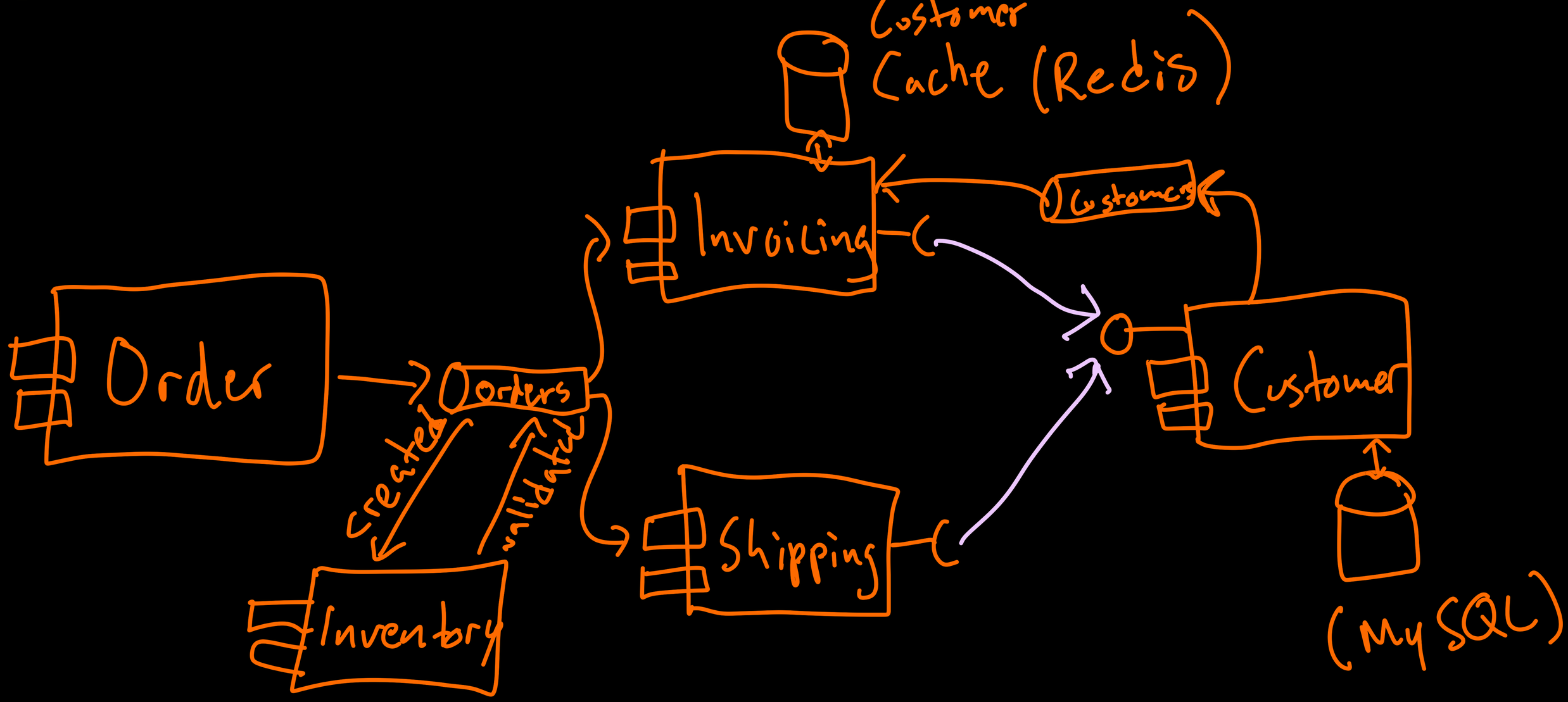
EVENT-BASED CACHING OF QUERIES EXAMPLE: EVENT CONSUMER

```
@KafkaListener(topics = "${kafka.topic.customers}")
public void receive(ConsumerRecord<String, String> record) {
  String id = record.key();
  EventType eventType = EventType.valueOf(record.value());
  switch(eventType) {
    case CREATED: case UPDATED:
      Customer customer = customerService.getCustomerById(id);
      customerRepository.save(customer);
     break;
    case DELETED:
      customerRepository.deleteById(id);
      break;
  }
}
```

DEMO: GOING FROM SYNCHRONOUS DEPENDENCIES …



... TO EVENT NOTIFICATION



https://github.com/callistaenterprise/cadec2019-eda.git

EVENT NOTIFICATION: PROS AND CONS

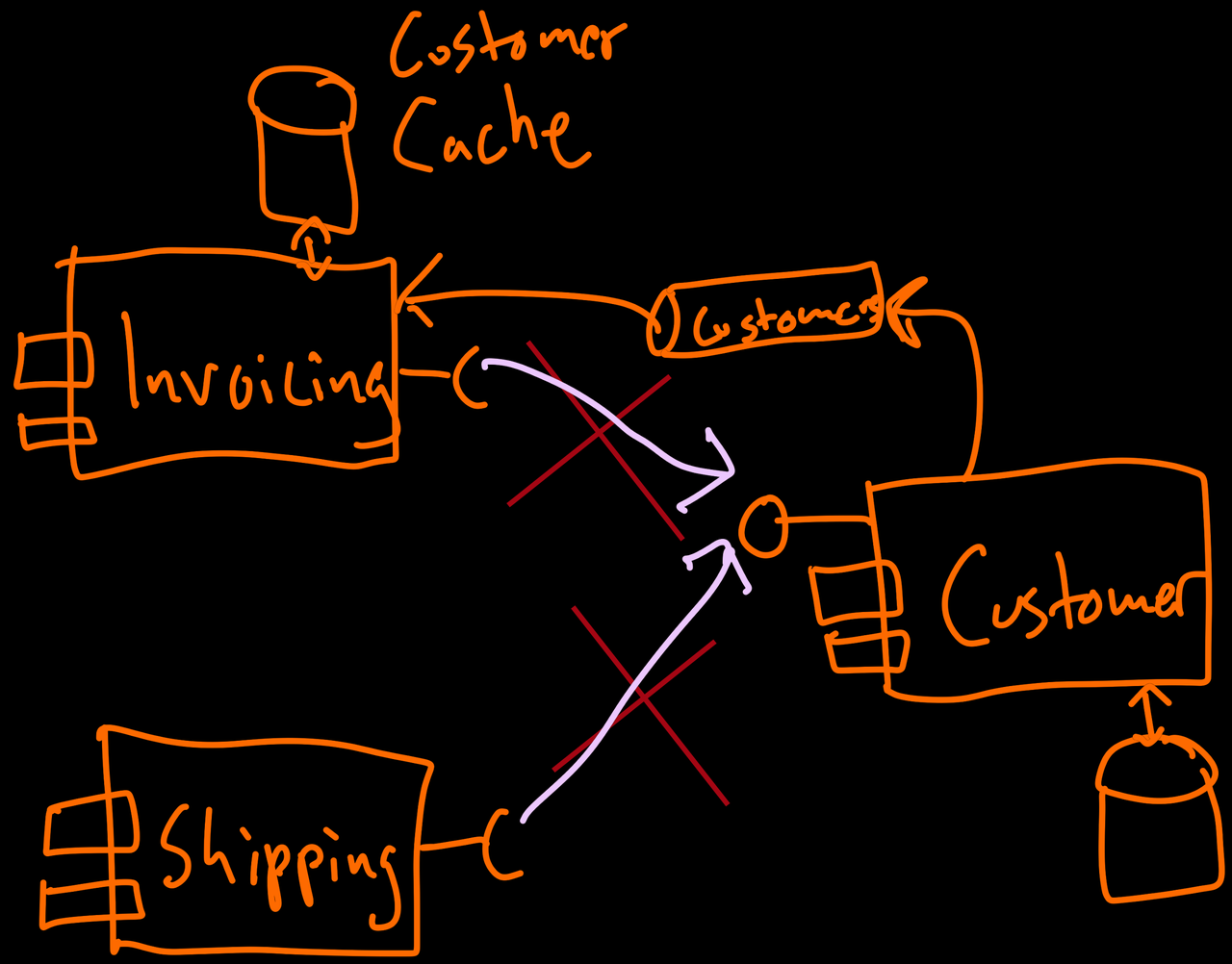
- Agility
- Resilience
- Performance
- Scalability

- Added complexity & Cost
  - Asynchronous Events & Coordination
  - Monitoring
  - Error handling
  - Testing

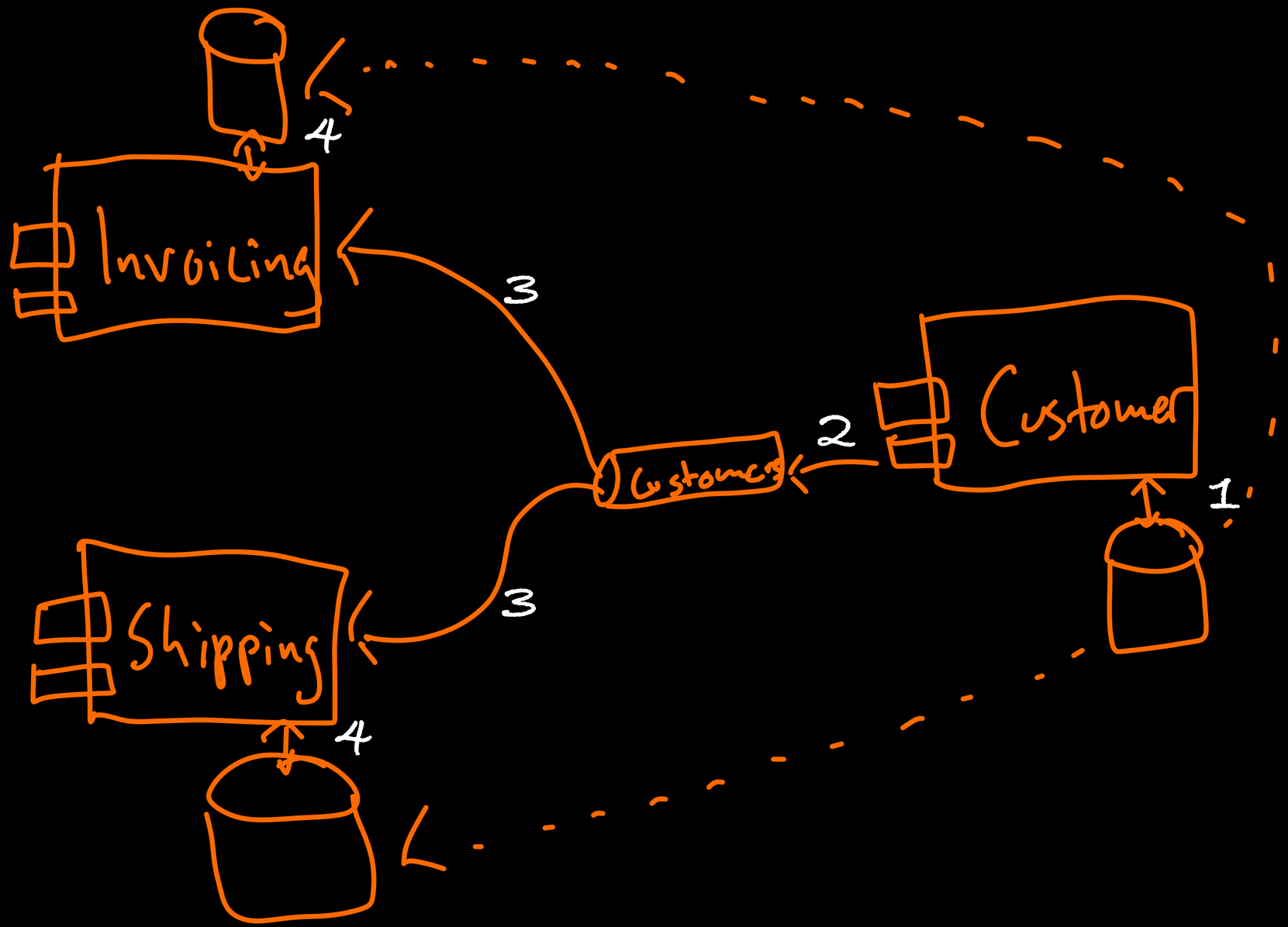
AGENDA

- Short background
  - Events
  - Apache Kafka
- Event Notification
  - What? Why? How?
- *Event-Carried State Transfer*
  - What? Why? How?
- Event Sourcing
  - What? Why? How?
- Sum Up

EVENT-CARRIED STATE TRANSFER



EVENT-CARRIED STATE TRANSFER



EVENT-CARRIED STATE TRANSFER: DRIVERS

- Autonomicity
- Performance/latency
- Different view on data
- Need to aggregate/correlate data from multiple sources

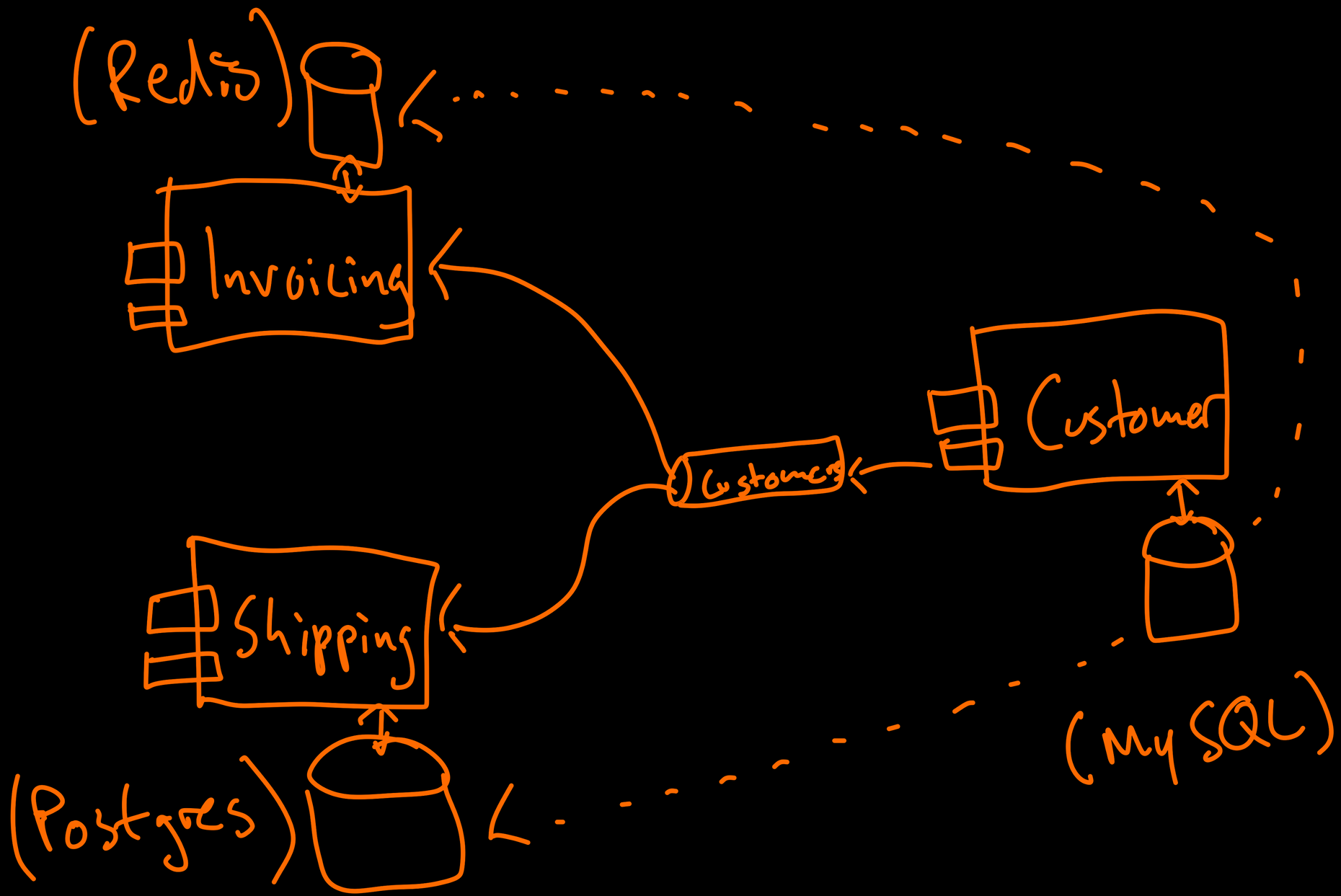
EVENT-CARRIED STATE TRANSFER EXAMPLE: PRODUCER

```
@PostMapping(value = "/customer")
public Customer create(@RequestBody Customer customer) {
  ...
  customerRepository.save(customer);
  customerEventSender.send(customer.getId(), customer);
  ...
}

@DeleteMapping(value = "/customer/{id}")
public void deleteById(@PathVariable String id) {
  ...
  customerRepository.deleteById(id);
  customerEventSender.send(id, null);
  ...
}
```

EVENT-CARRIED STATE TRANSFER EXAMPLE: CONSUMER

```
@KafkaListener(topics = "${kafka.topic.customers}")
public void receive(ConsumerRecord<String, Customer> record) {
  String id = record.key();
  Customer customer = record.value();
  if (customer != null) {
   customerRepository.save(customer);
  } else {
   customerRepository.deleteById(id);
  }
}
```

DEMO: EVENT-CARRIED STATE TRANSFER



EVENT-DRIVEN STATE TRANSFER: PROS AND CONS

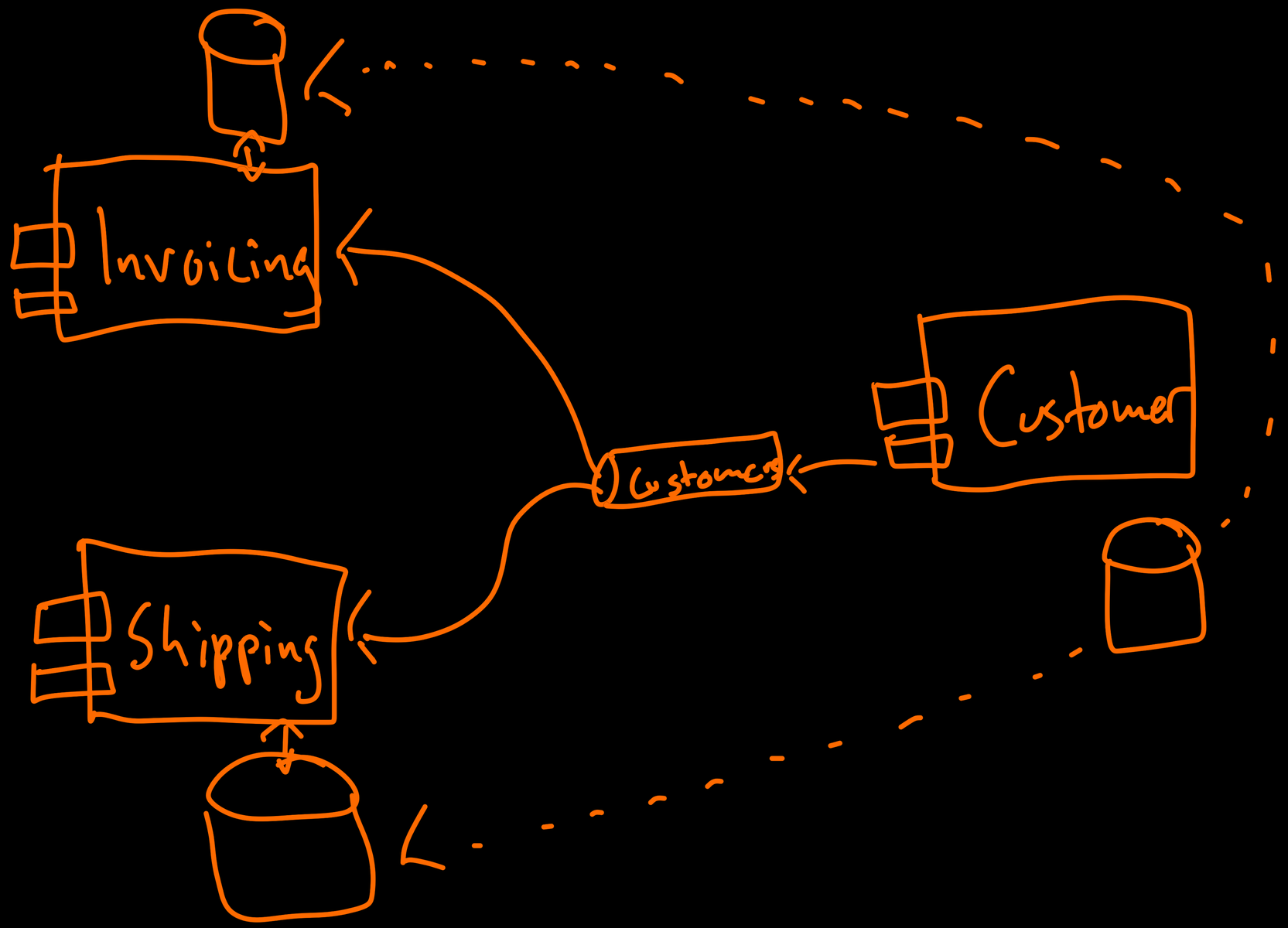
- Full Autonomy
- Performance/Latency
- Scalability

- Even more complexity
  - Data duplication
  - Eventual Consistency
  - Bootstrapping new consumers

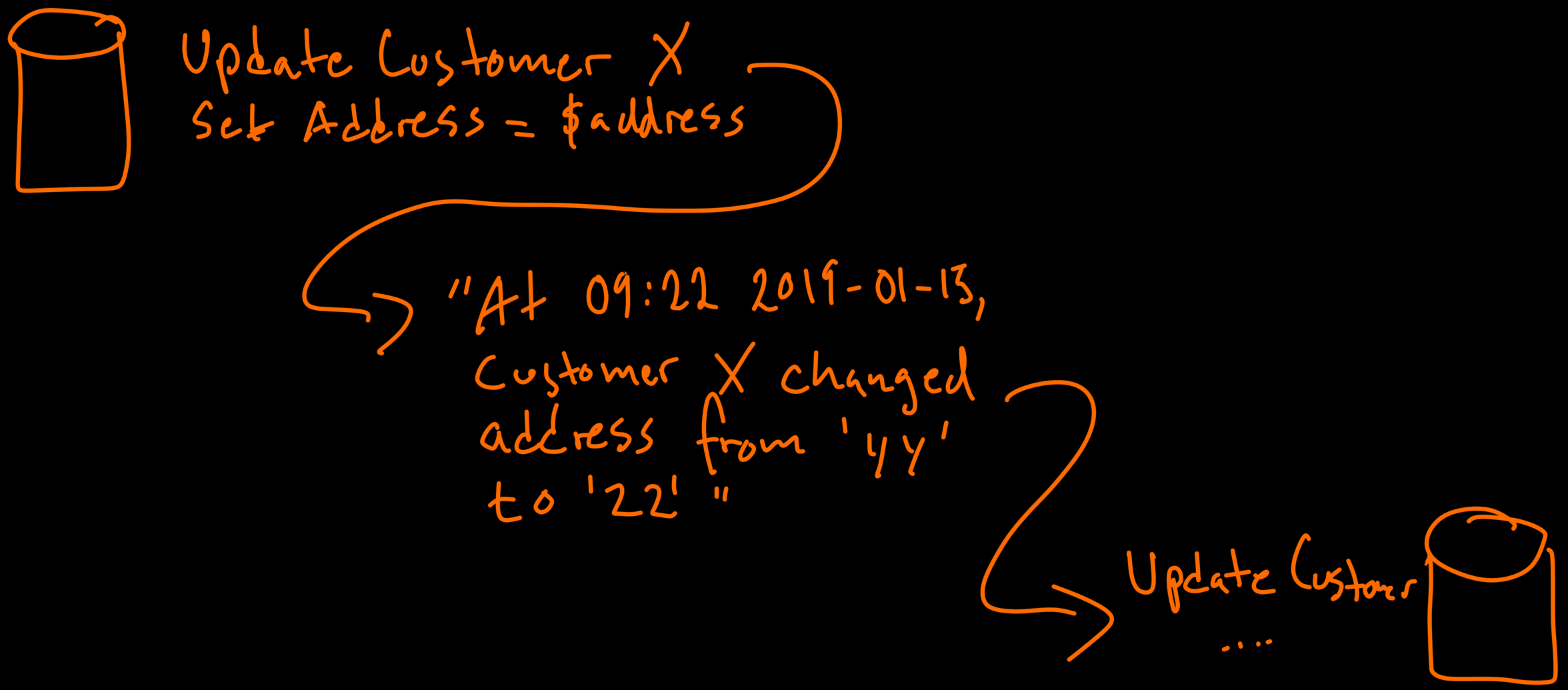
AGENDA

- Short background
  - Events
  - Apache Kafka
- Event Notification
  - What? Why? How?
- Event-Carried State Transfer
  - What? Why? How?
- *Event Sourcing*
  - What? Why? How?
- Sum Up

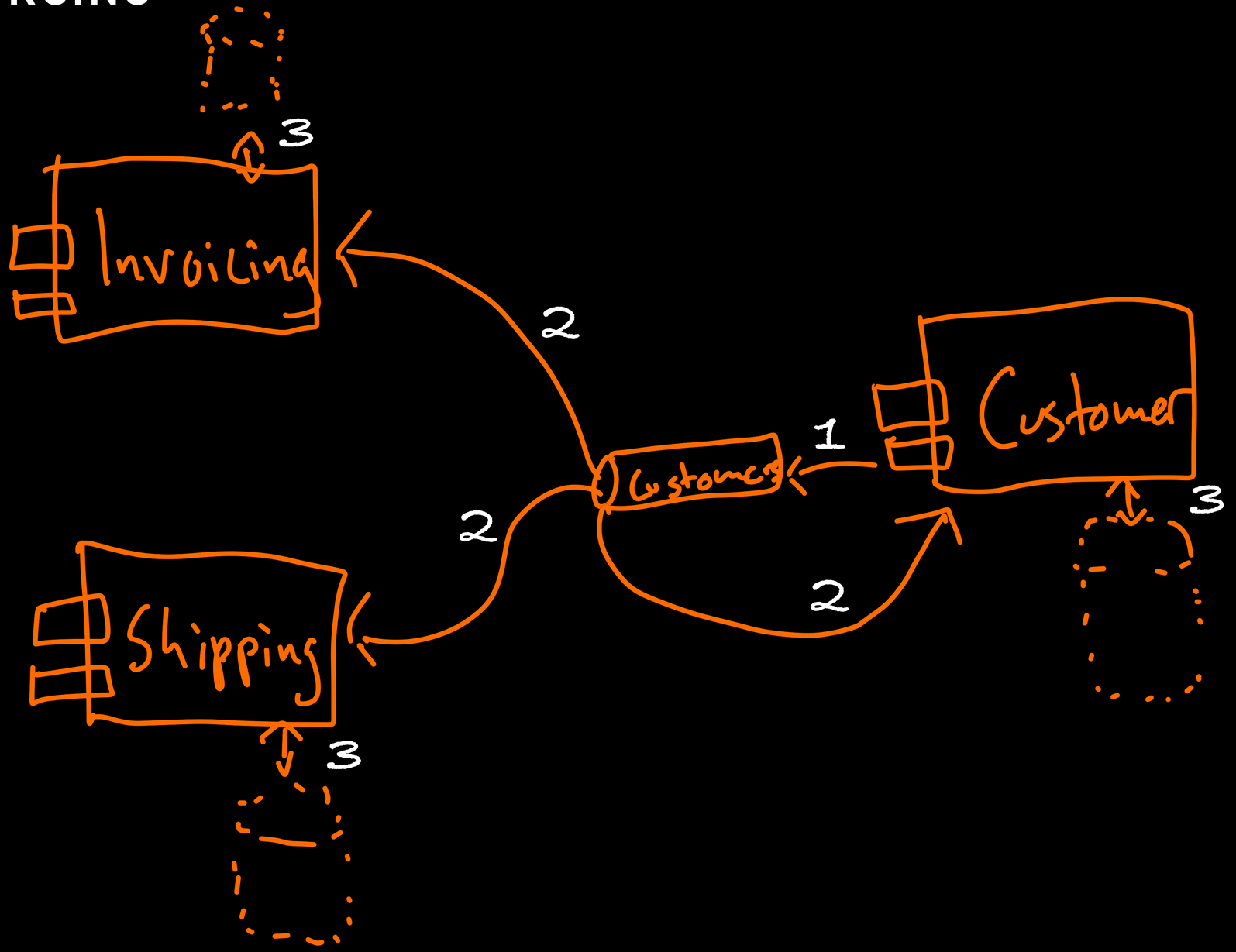
DATA DUPLICATION



TURNING THE DATABASE INSIDE-OUT



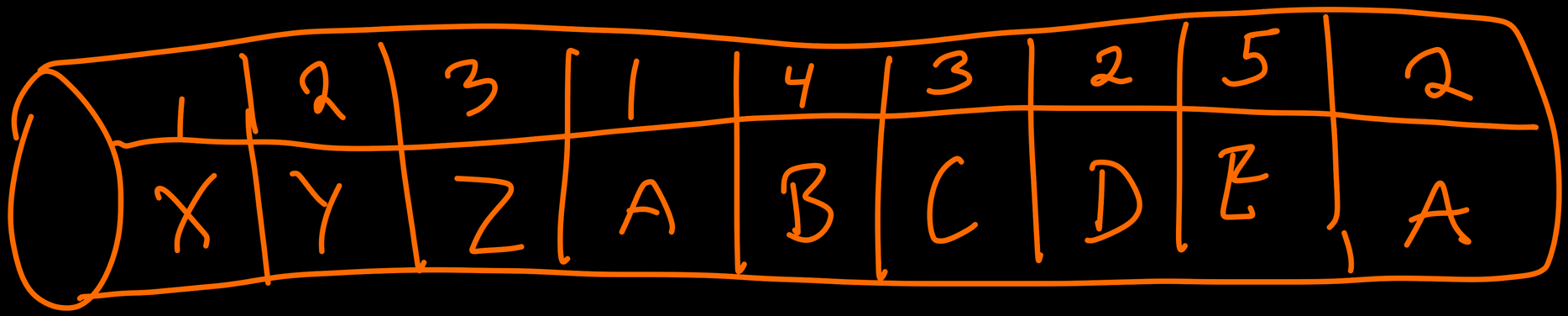
EVENT SOURCING



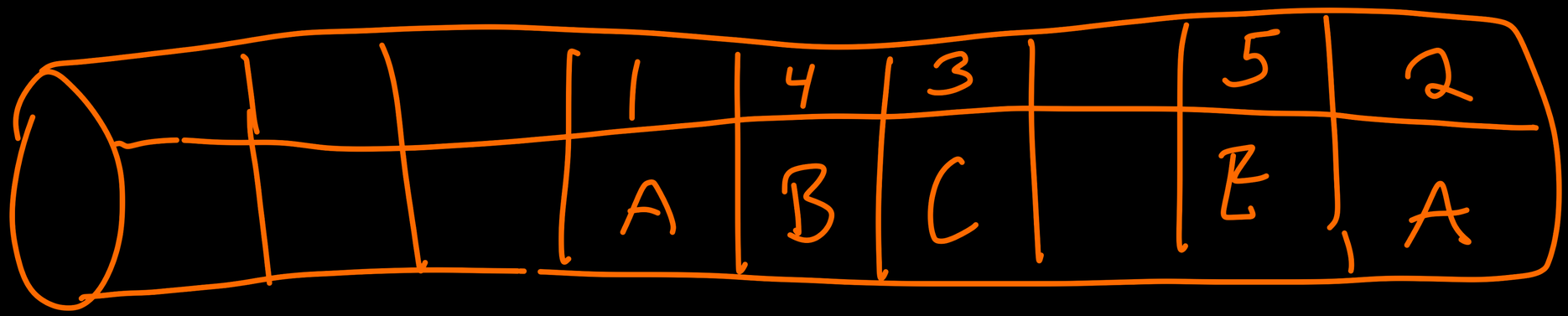
EVENT SOURCING DRIVERS

- Single Source of Truth
- Easier to bootstrap new subscribers
- Enhanced flexibility
  - May correlate/join data from multiple streams
  - Introduce timing windows
  - Replay events to recreate historic states
  - …

KAFKA LOG COMPACTION



KAFKA LOG COMPACTION



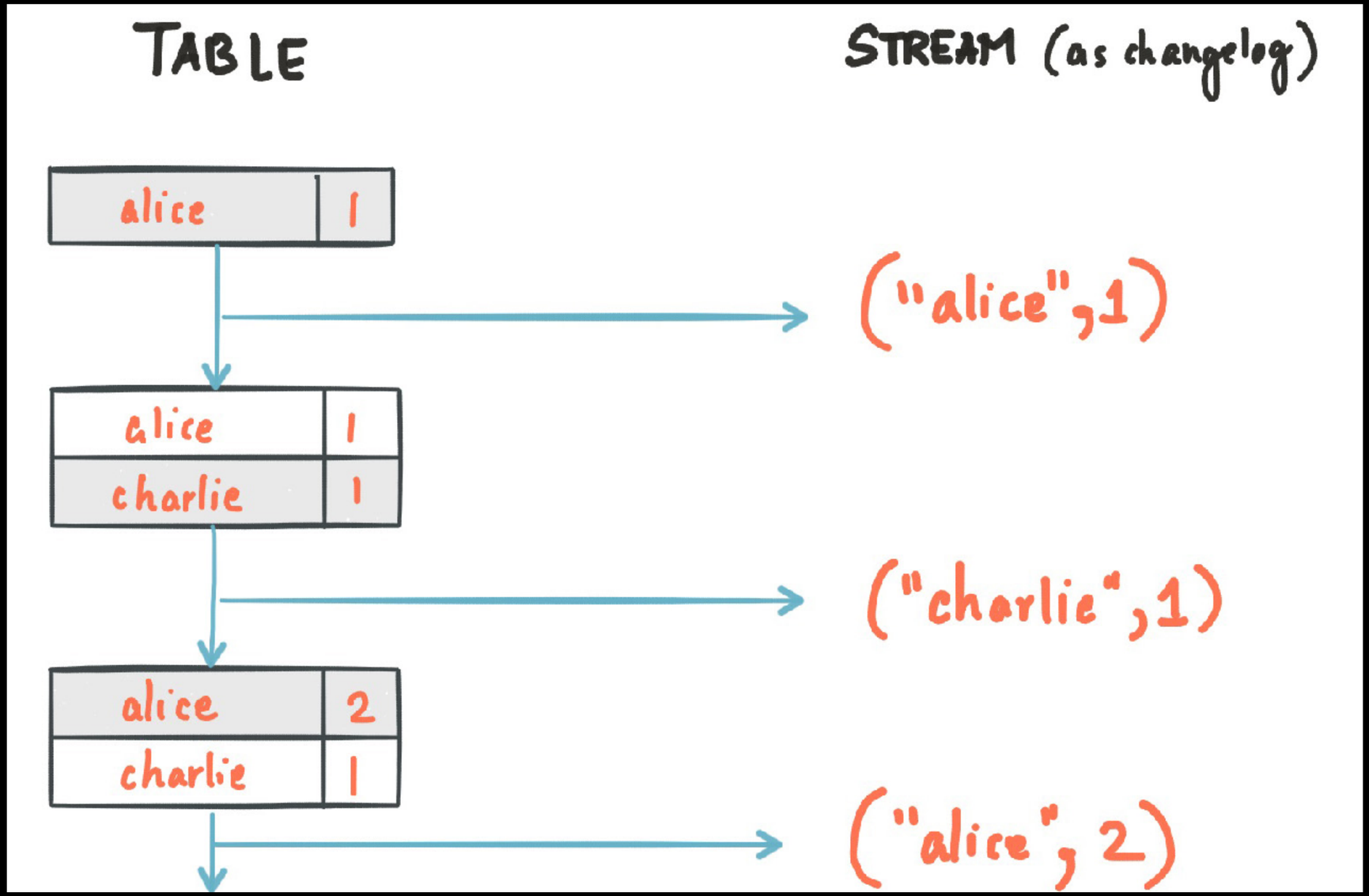
KAFKA LOG COMPACTION EXAMPLE

```
@Bean
public NewTopic customersTopic() {
   int partitions = 2;
   short replicationFactor = 2;
   Map<String, String> configs = new HashMap<>();
   configs.put("cleanup.policy", "compact");
   configs.put("min.compaction.lag.ms", "600000"); // 10 minutes
   configs.put("retention.ms", "-1");
   configs.put("retention.bytes", "-1");
   return new NewTopic(customersTopic, partitions,
                       replicationFactor).configs(configs);
}
```

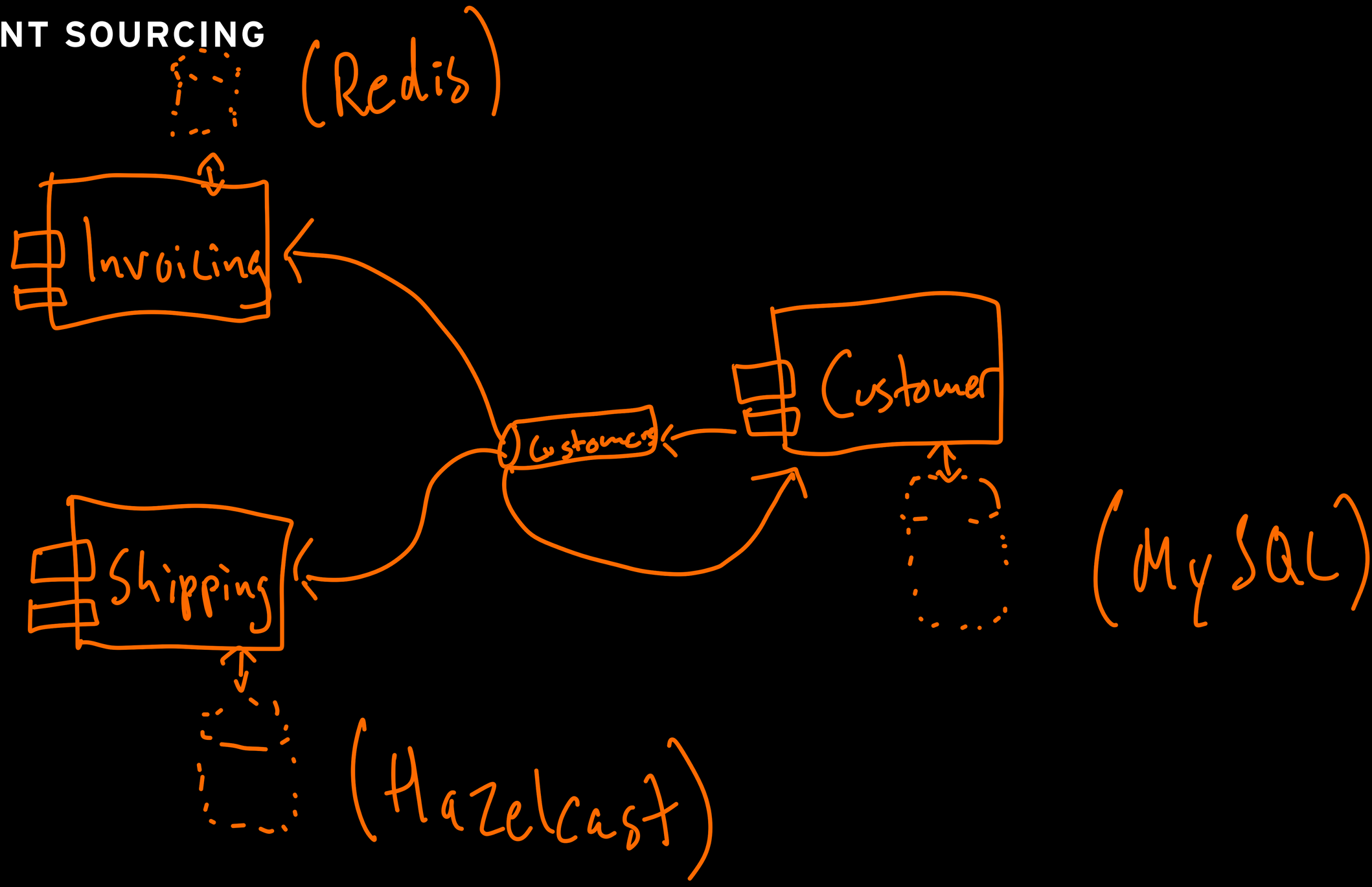
EVENT SOURCING EXAMPLE: CONSUMER

```
@Component
public class CustomerEventReceiver implements ConsumerSeekAware {
  ...
  @KafkaListener(topics = "${kafka.topic.customers}")
  public void receive(ConsumerRecord<String, Customer> record) {
    ...
    customerCache.save(customer);
    ...
  }

  @Override
  public void onPartitionsAssigned(Map<TopicPartition, Long> assignments,
      ConsumerSeekCallback callback) {
    for (TopicPartition topicPartition : assignments.keySet()) {
      callback.seekToBeginning(topicPartition.topic(),
                               topicPartition.partition());
    }
  }

}
```

KAFKA STREAMS KTABLE



DEMO: EVENT SOURCING



EVENT SOURCING: PROS AND CONS

- Single source of Data
- Data Flexibility
- Audit friendly
- Replay capable

- Even more complexity
  - Log compaction
  - Schema evolution
  - Eventual Consistency

CONCLUSIONS

- Event Driven Architectures comes in different ‘flavours’:
  - Event Notification
  - Event-Driven State Transfer
  - Event Sourcing

CONCLUSIONS

- Event Driven Architectures may enable you to
  - Further decouple your services
  - Enhance flexibility and changeability
  - Enhance autonomy, by sharing even less
  - Exploit parallelism further
  - Reduce latency

CONCLUSIONS

- Event Driven Architectures however comes with a potentially substantial cost, due to increased complexity
  - Consistency
  - Duplication
  - Schema Evolution
  - Error handling
  - Testability

Thank you!

BJORN.BESKOW@CALLISTAENTERPRISE.SE