

Exploring Lightweight Event Sourcing

Erik Rozendaal <erozendaal@silverline.com>
@erikrozendaal



silverline
ONLY RESULTS MATTER

GOTO Amsterdam 2011

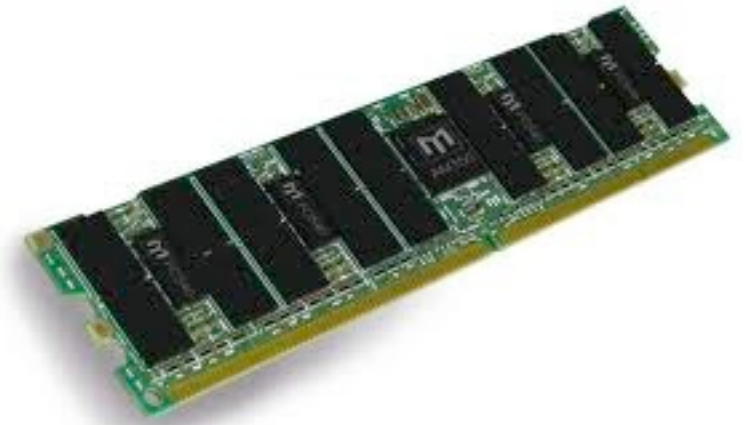
Goals

- The problem of data persistence
- Understand event sourcing
- Show why Scala is well-suited as implementation language

Problem

Data must be *durable*





But current
applications are *lossy*

Lossy?

Lossy?

- UPDATE invoice WHERE id = 1234
SET total_amount = 230

Lossy?

- UPDATE invoice WHERE id = 1234
SET total_amount = 230
- What happened to the previous order amount?

Lossy?

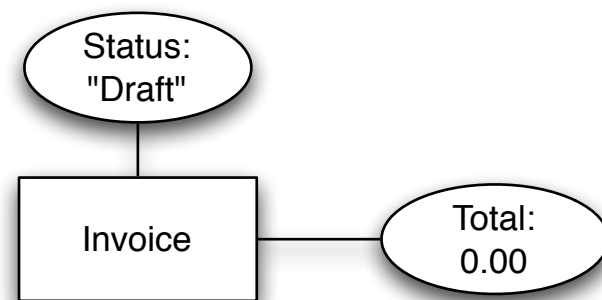
- UPDATE invoice WHERE id = 1234
SET total_amount = 230
- What happened to the previous order amount?
- Why was the order amount changed?

Lossy?

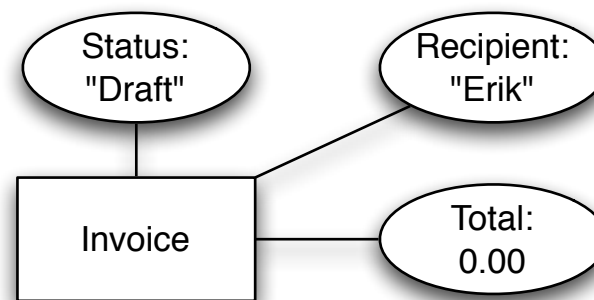
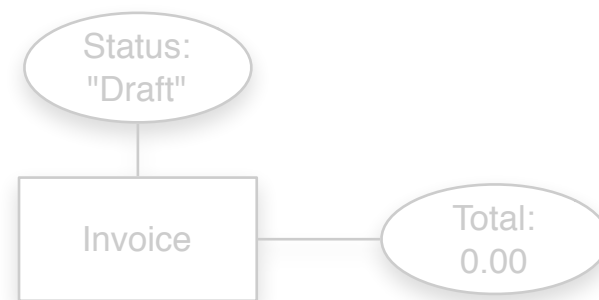
- UPDATE invoice WHERE id = 1234
SET total_amount = 230
- What happened to the previous order amount?
- Why was the order amount changed?
- Application behavior is not captured!

Behavior?

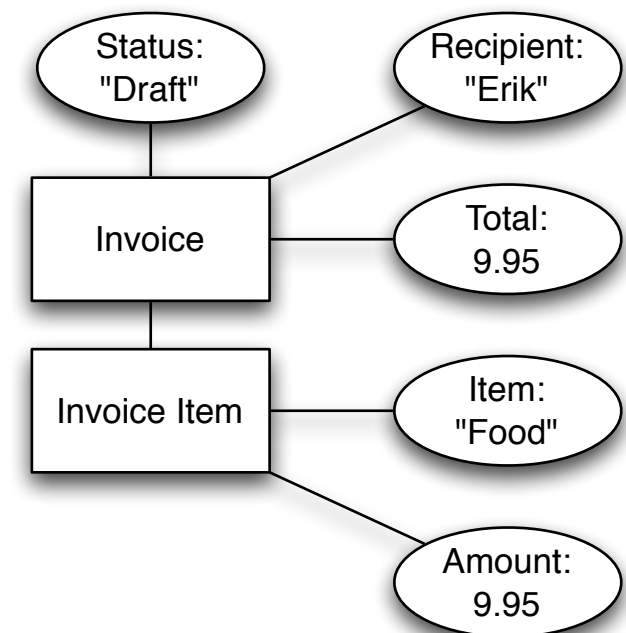
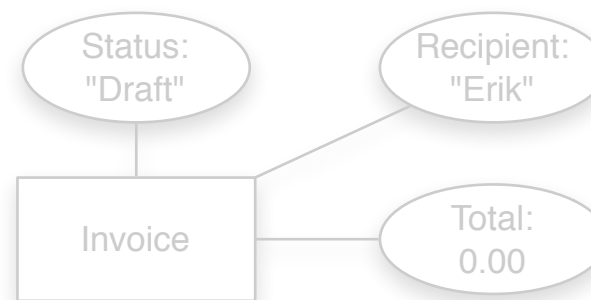
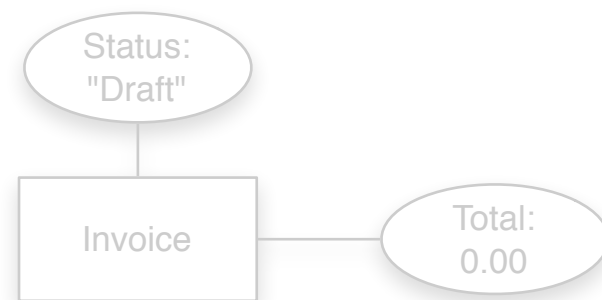
Behavior?



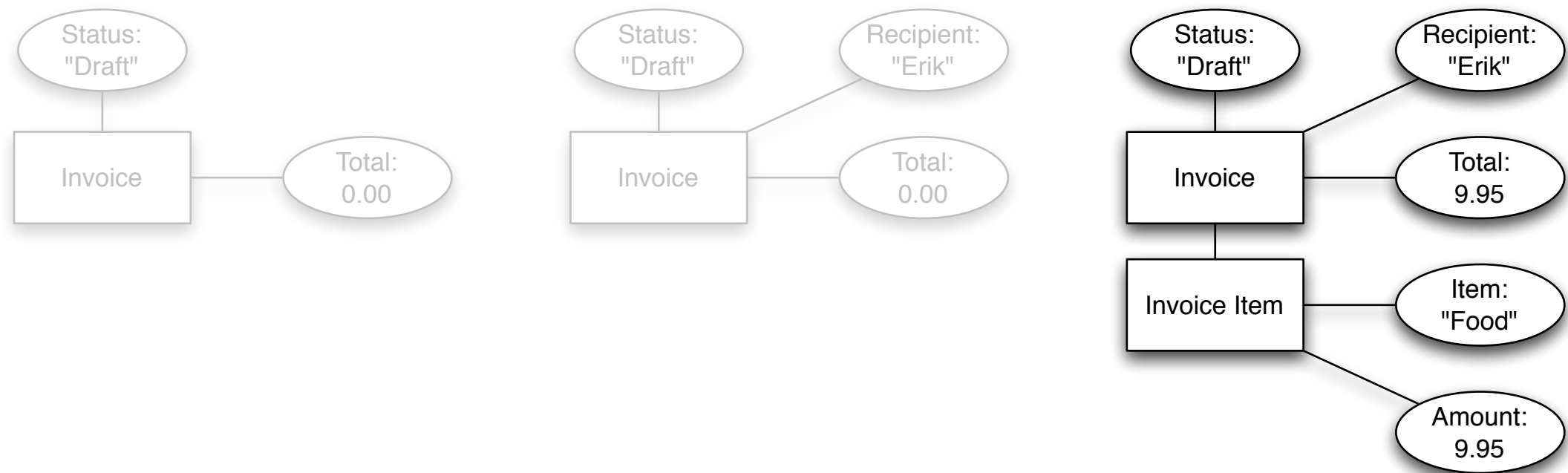
Behavior?



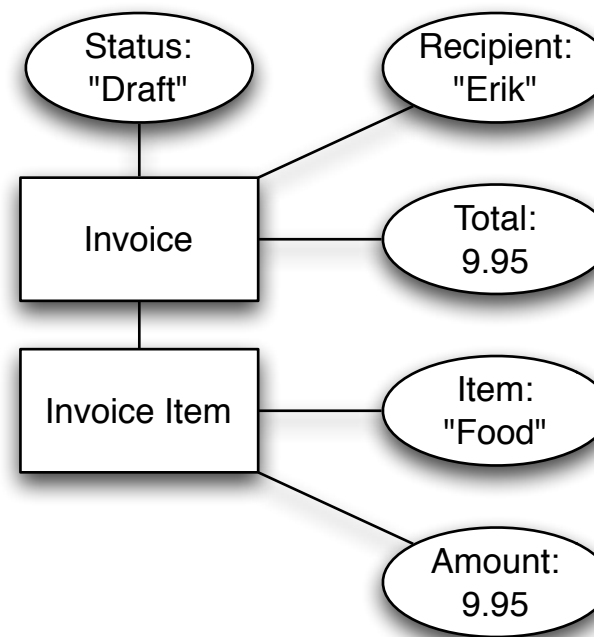
Behavior?



It's just data mutation



It's just data mutation



N-Tier

N-Tier

- Presentation, Service, and Data Layers

N-Tier

- Presentation, Service, and Data Layers
- Shared data model (“domain”)

N-Tier

- Presentation, Service, and Data Layers
- Shared data model (“domain”)
- Heavy use of a single, global, mutable variable (“the database”)

Object-Relational Mapper

“Inexperienced programmers love magic because it saves their time. Experienced programmers hate magic because it wastes their time.”

– @natpryce

Transactions

- Start transaction
- SELECT copy of data from database
- ORM hydrates objects to give program private copy of data
- ORM compares mutated program copy with initial copy to generate UPDATES
- Commit transaction



Performance Tuning

Performance Tuning

- Do you know the queries generated by your ORM?

Performance Tuning

- Do you know the queries generated by your ORM?
- What's the query execution plan?

Performance Tuning

- Do you know the queries generated by your ORM?
- What's the query execution plan?
- Optimize reads versus writes

“Domain” Model

“Domain” Model

- Presentation layer needs wide access to many parts

“Domain” Model

- Presentation layer needs wide access to many parts
- Service layer is only interested in subset related to application behavior

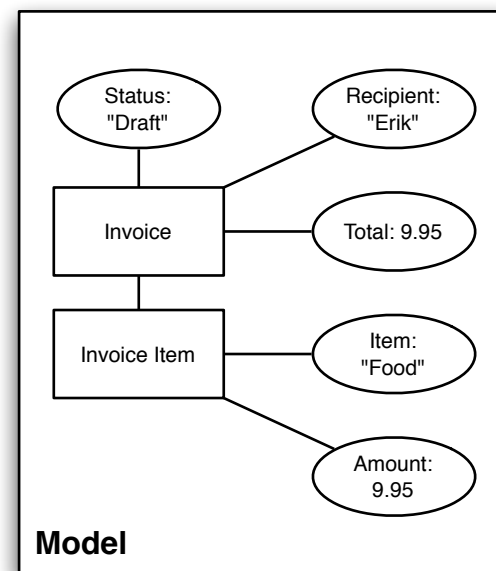
“Domain” Model

- Presentation layer needs wide access to many parts
- Service layer is only interested in subset related to application behavior
- ORM tightly couples domain to relational model

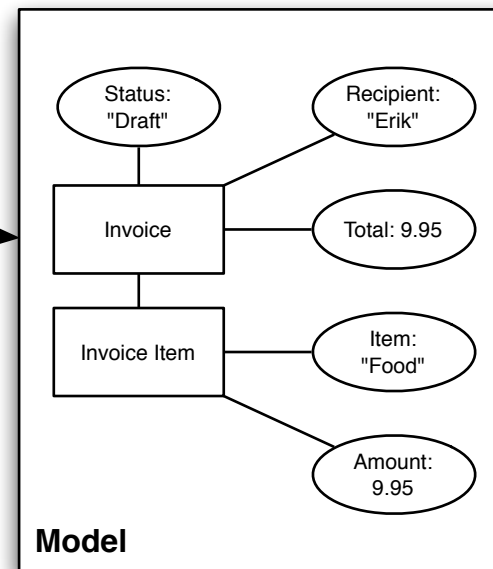
“Domain” Model

- Presentation layer needs wide access to many parts
- Service layer is only interested in subset related to application behavior
- ORM tightly couples domain to relational model
- High coupling, low cohesion

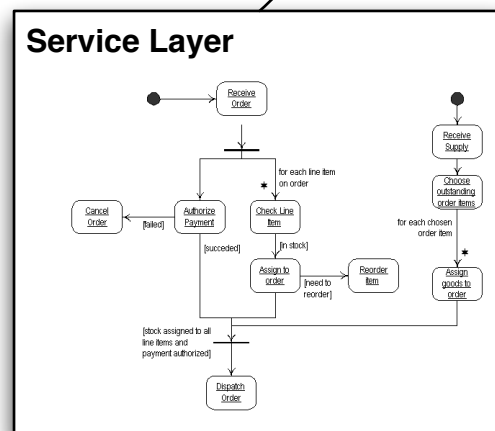
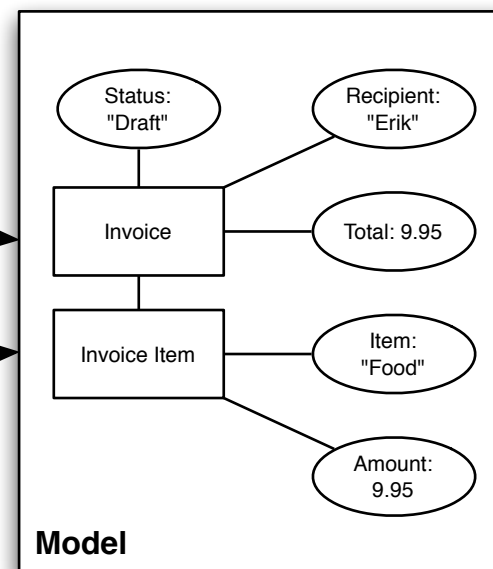
Domain Model



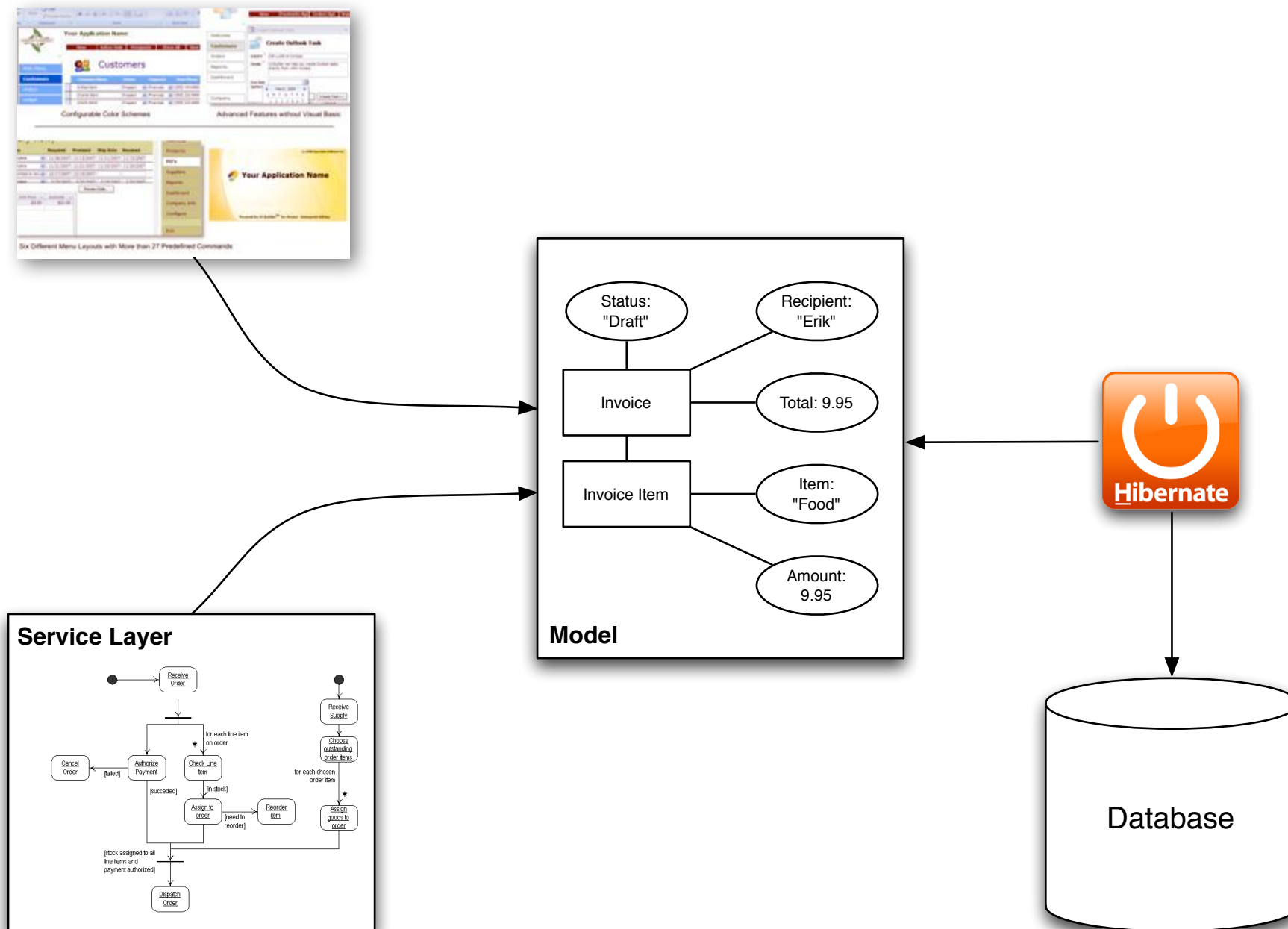
Domain Model



Domain Model



Domain Model



**Is it any surprise that
we're struggling to build
modular, maintainable
applications?**

Domain Driven Design

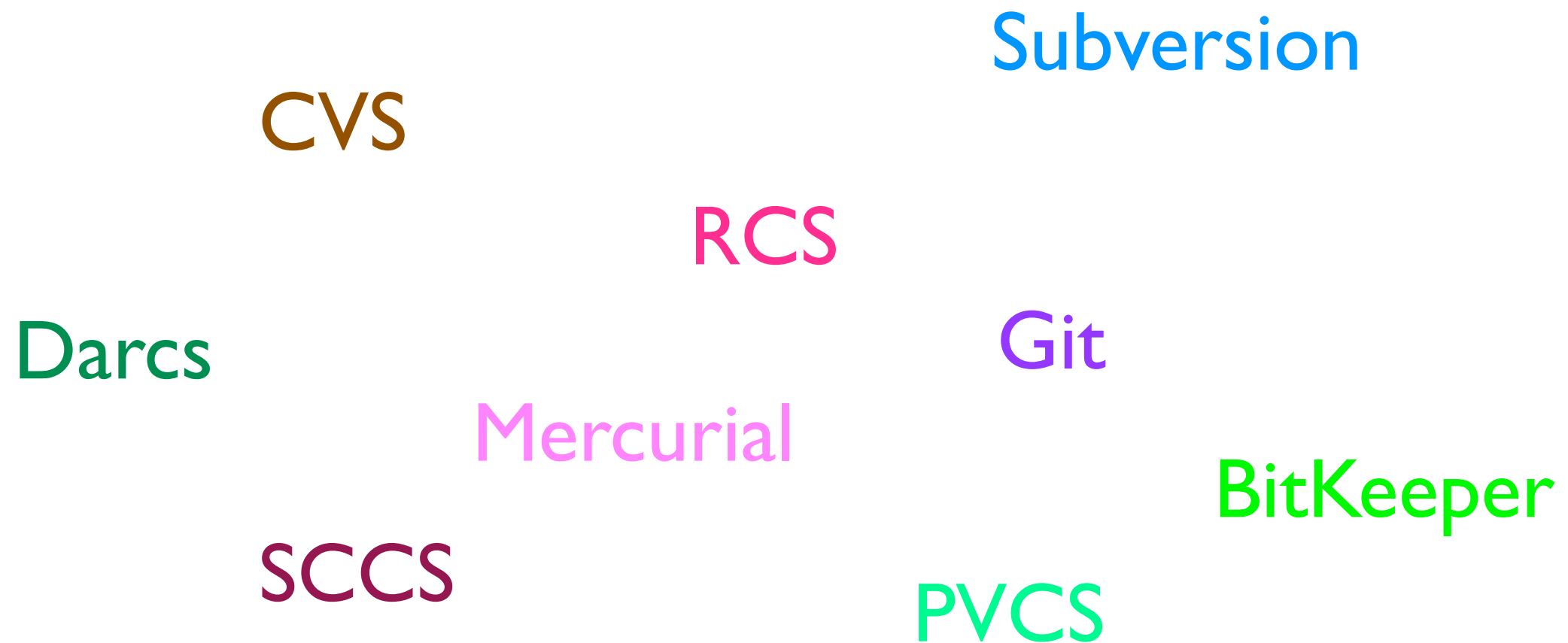
- **Domain-Driven Design** An approach to software development that suggests that (1) For most software projects, the primary focus should be on the domain and domain logic; and (2) Complex domain designs should be based on a model.
- **Domain Expert** A member of a software project whose field is the domain of the application, rather than software development. Not just any user of the software, the domain expert has deep knowledge of the subject.
- **Ubiquitous Language** A language structured around the domain model and used by all team members to connect all the activities of the team with the software.

Event Sourcing

- All state changes are explicitly captured using *domain events*
- Capture the *intent* of the user and the related *data*
- Events represent the *outcome* of application behavior

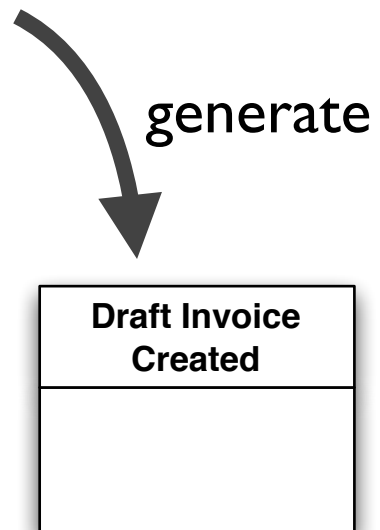
Source Control

Source Control

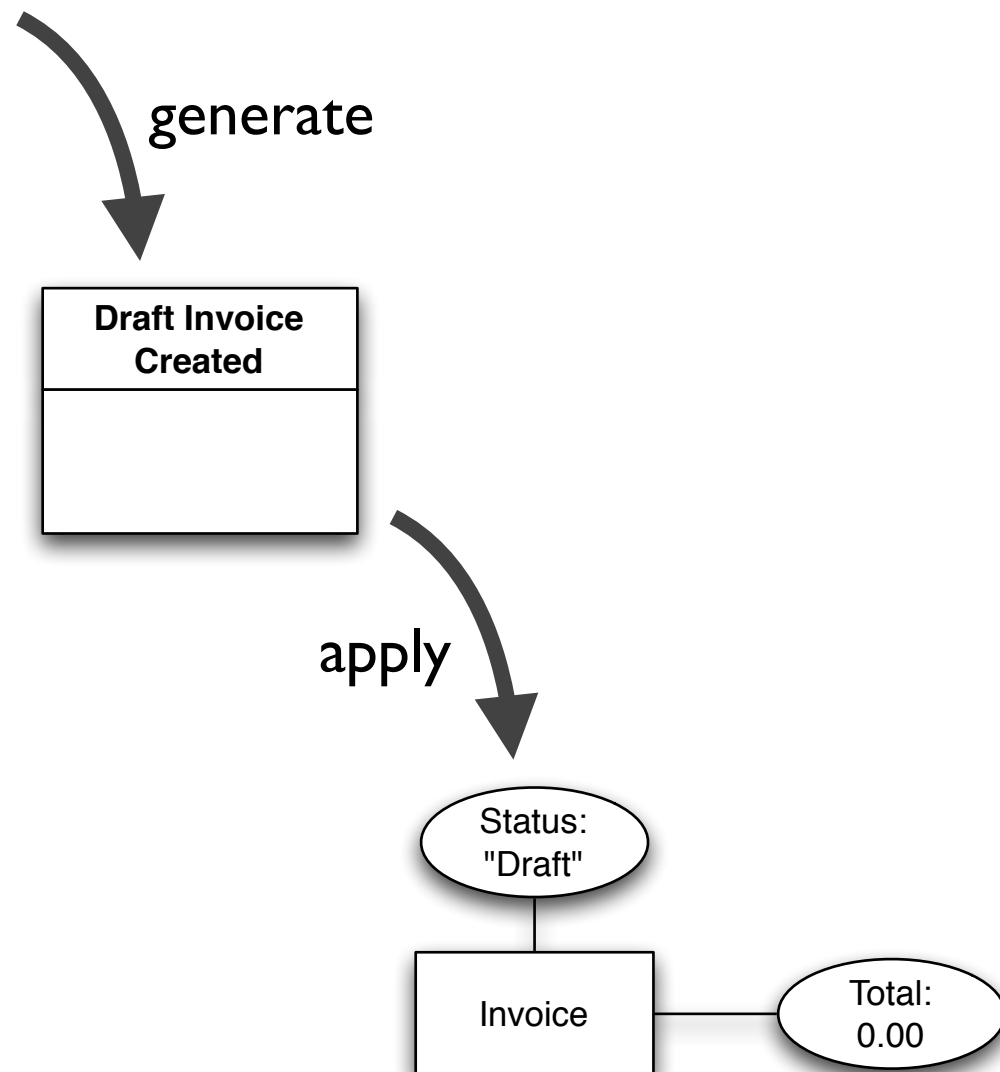


Domain Behavior

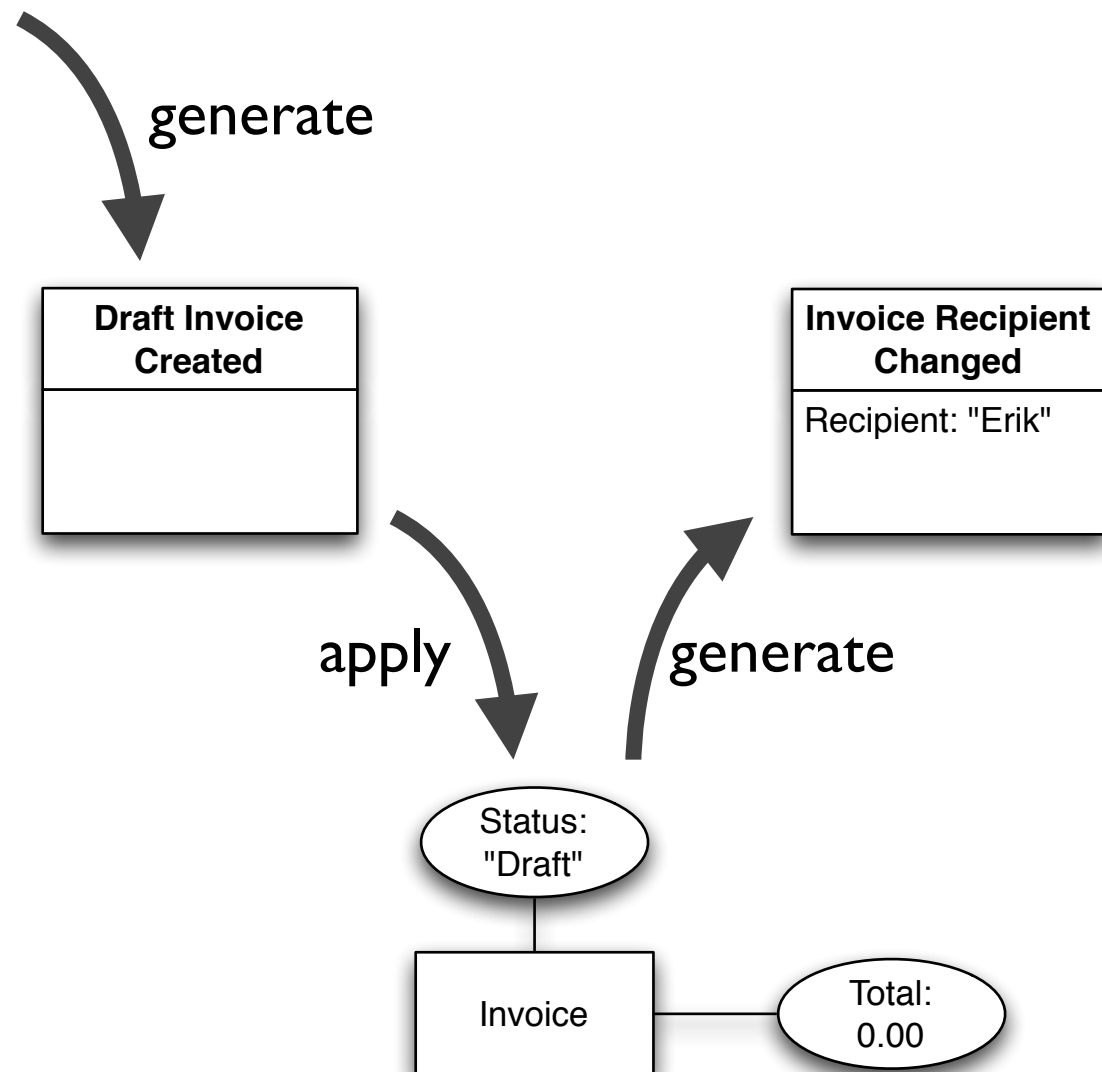
Domain Behavior



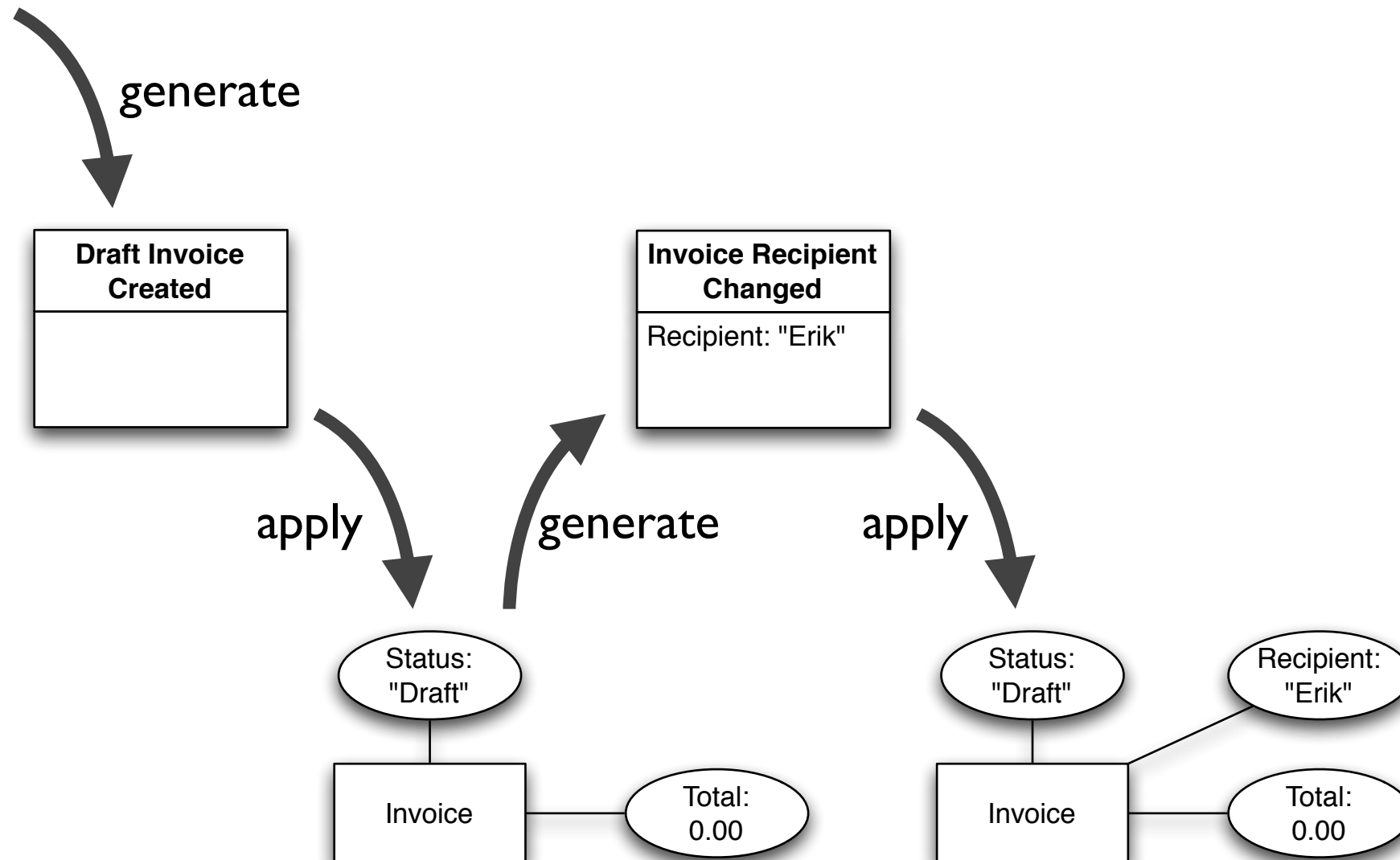
Domain Behavior



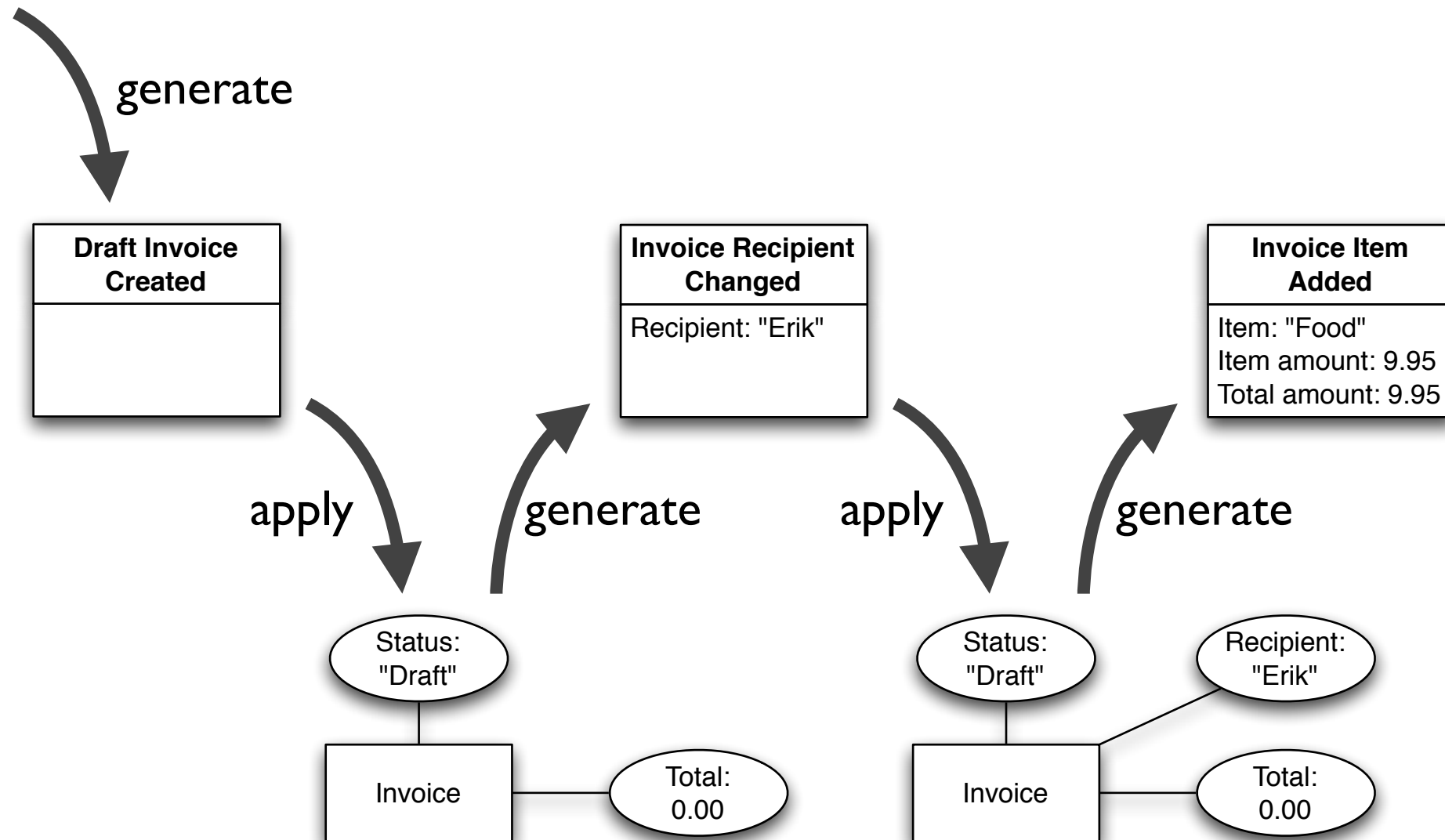
Domain Behavior



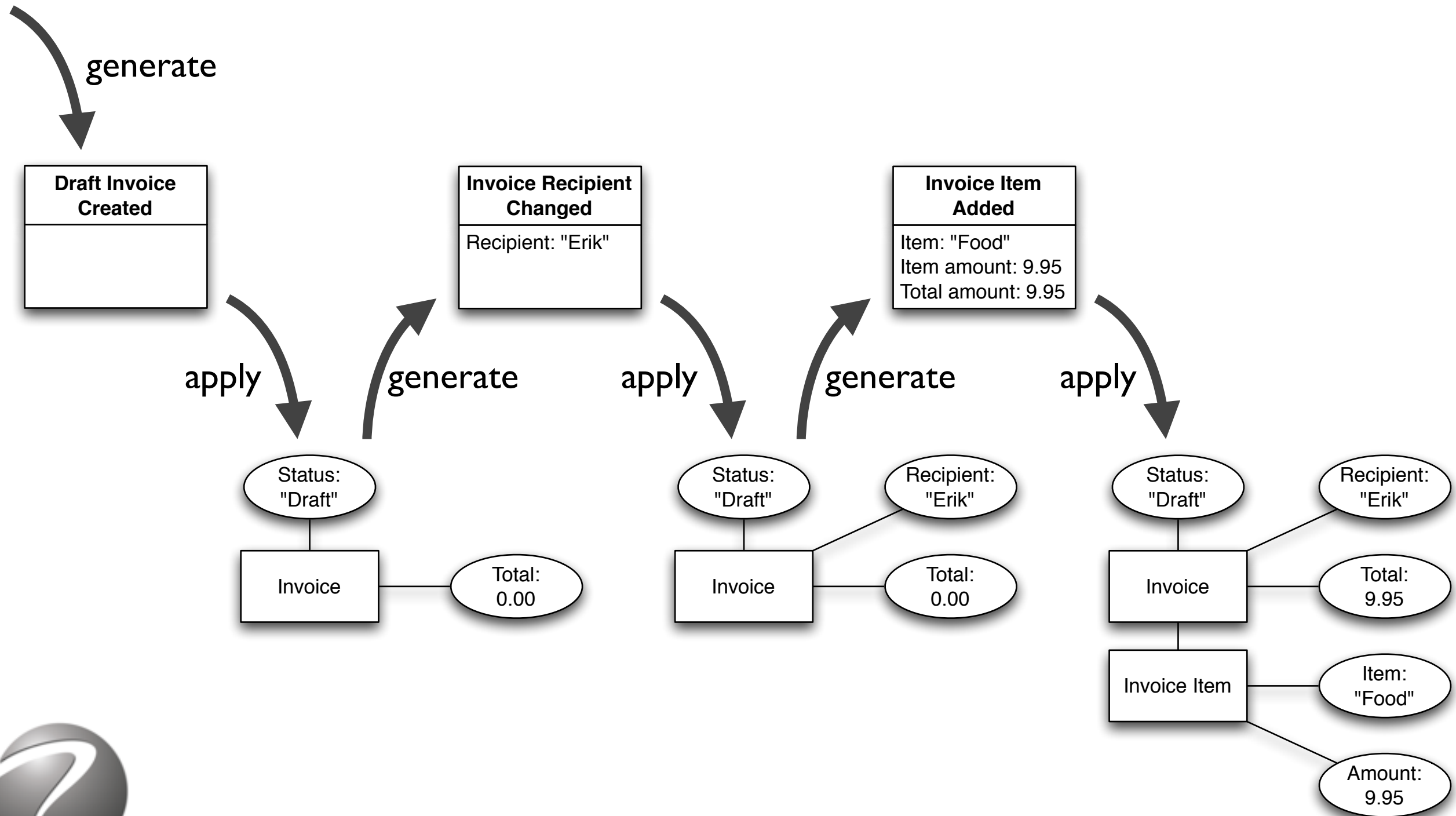
Domain Behavior



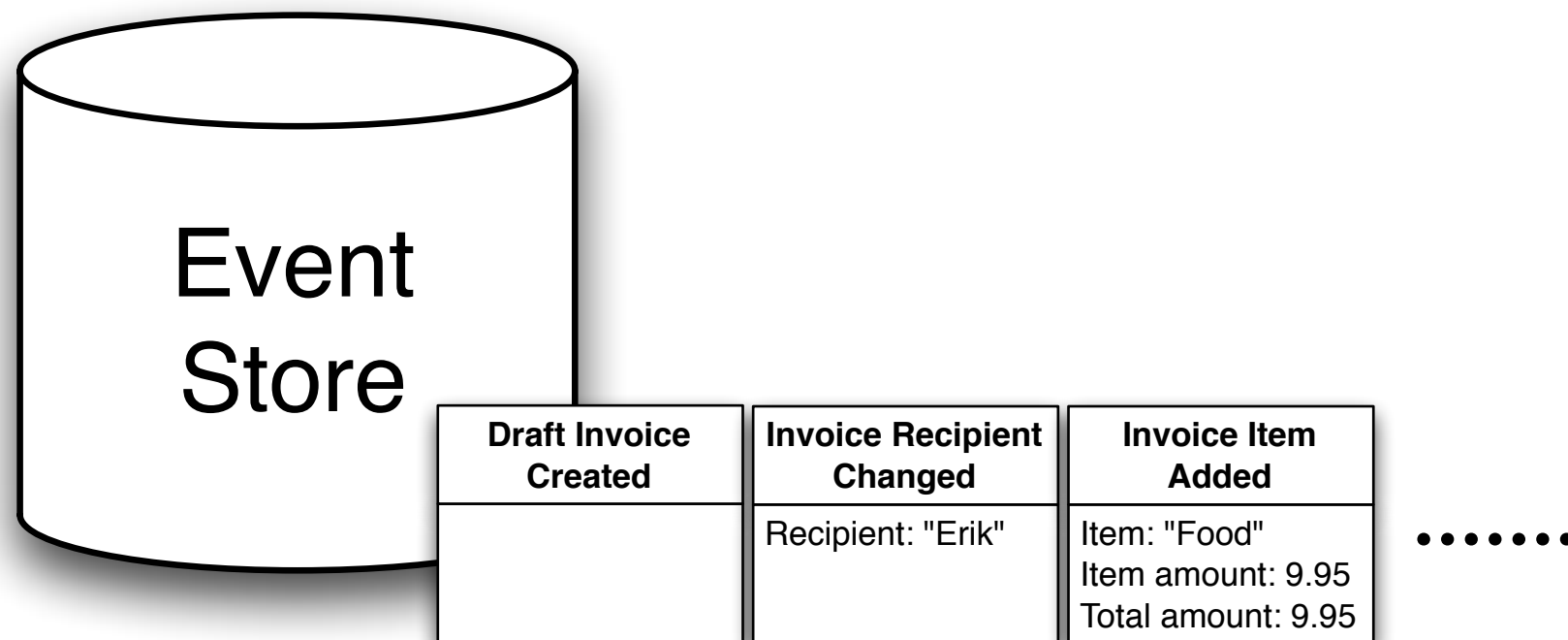
Domain Behavior



Domain Behavior

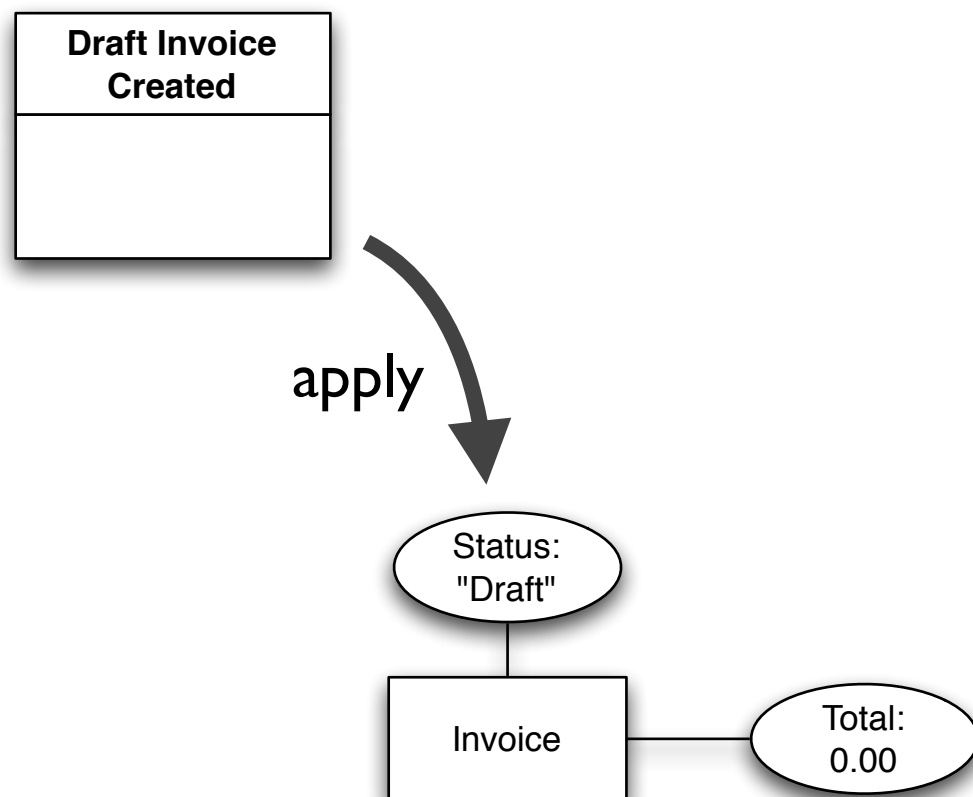


Only the events need to be stored on disk

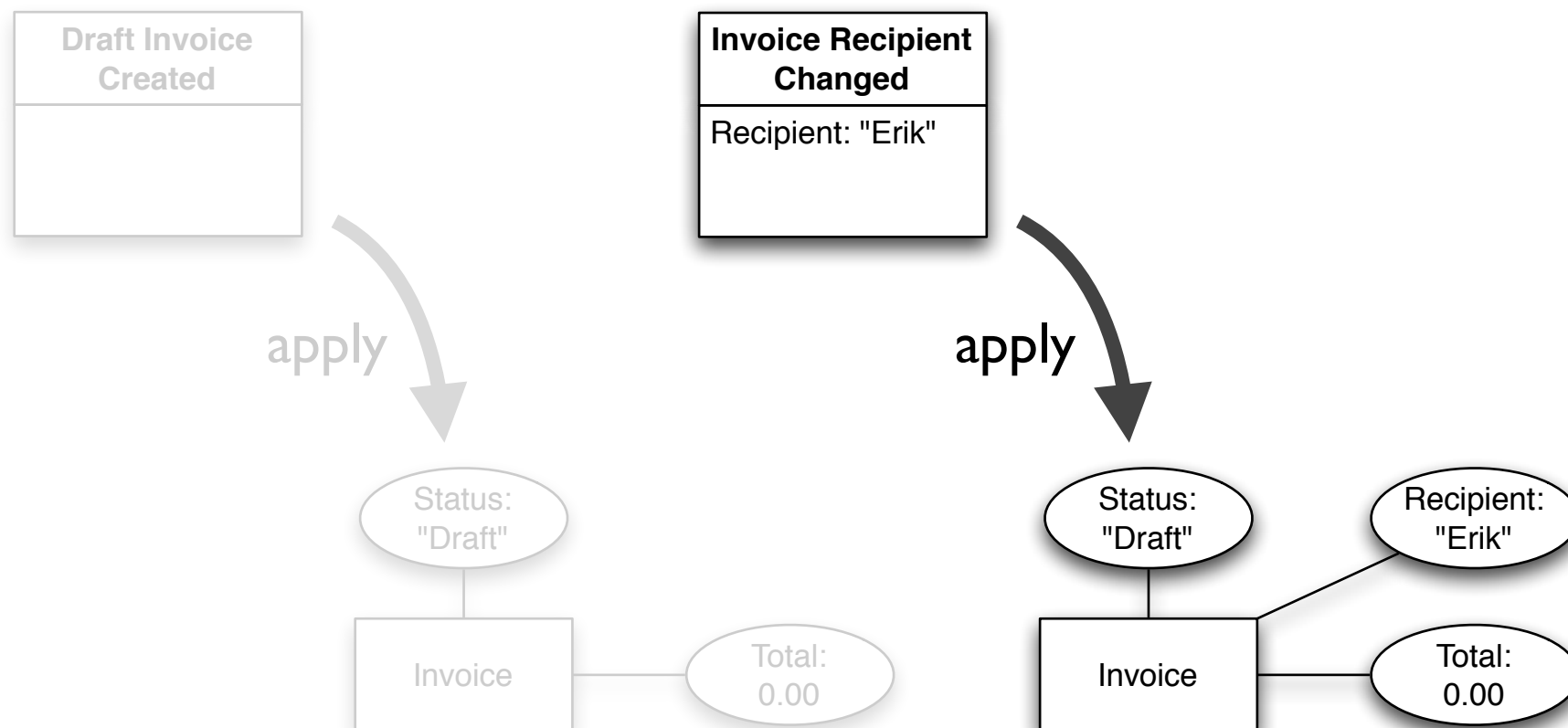


Reloading from history

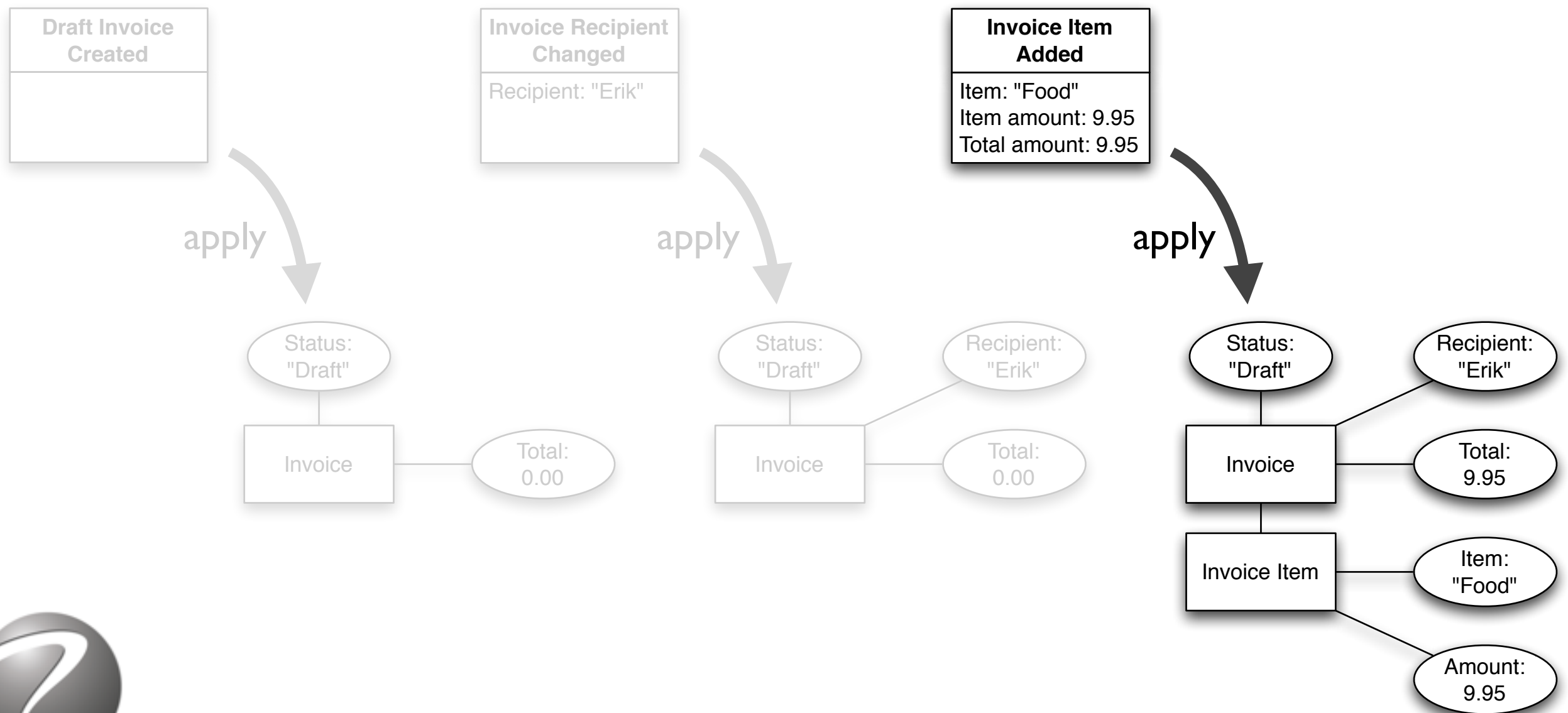
Reloading from history



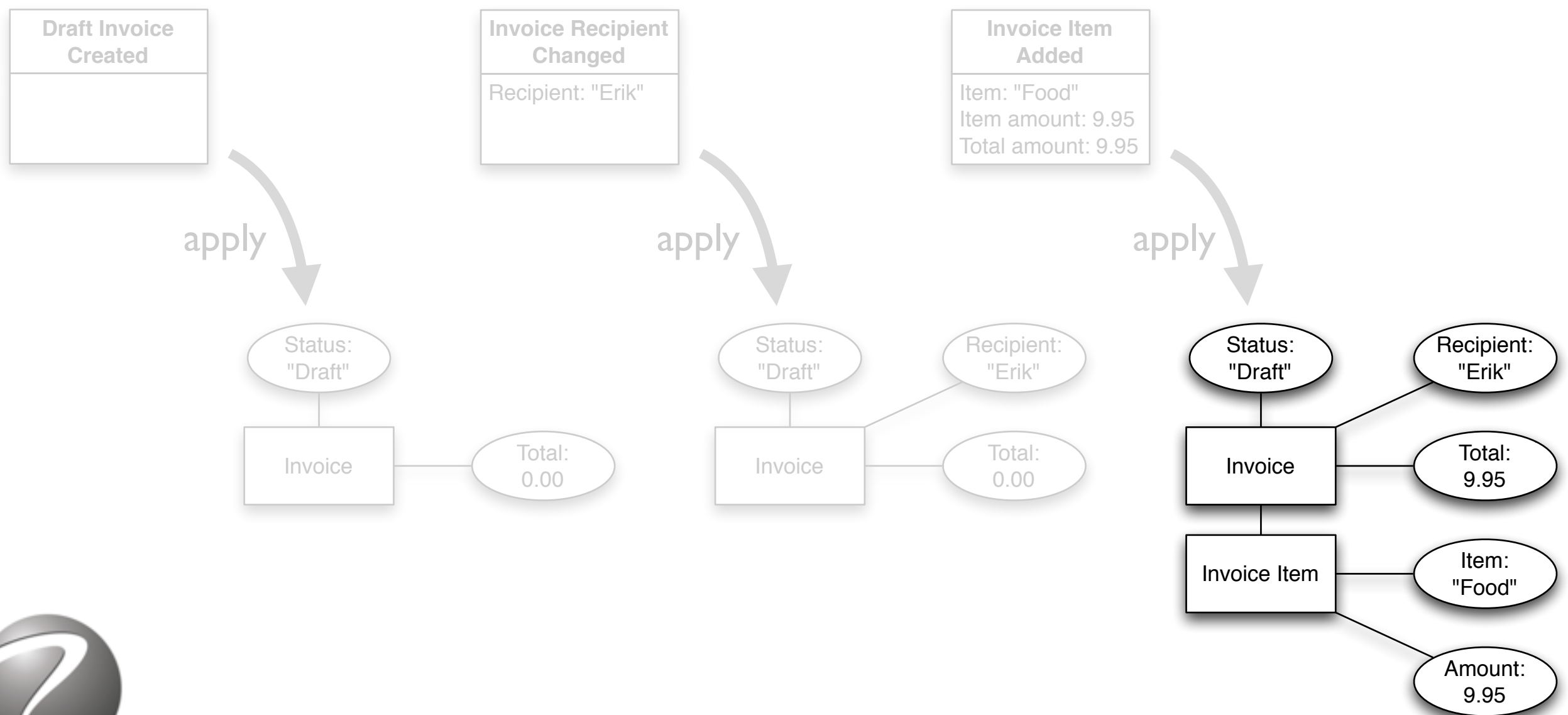
Reloading from history



Reloading from history



Reloading from history



YAGNI?

YAGNI?

- On the checkout page we'd like to promote products that customers previously removed from their shopping cart.

YAGNI?

- On the checkout page we'd like to promote products that customers previously removed from their shopping cart.
- Can you tell us how many people removed a product but bought it later anyway?

YAGNI?

- On the checkout page we'd like to promote products that customers previously removed from their shopping cart.
- Can you tell us how many people removed a product but bought it later anyway?
- ... over the past 5 years?
... and how much time passed in-between?

YAGNI?

- We have reports of a strange bug, but have not been able to isolate it. Could you look at the production data to find out what's going on?

Is it worth it?

Is it worth it?

- Make the event sourcing implementation as simple as possible

Is it worth it?

- Make the event sourcing implementation as simple as possible
- ... while avoiding the complexities of databases, ORMs, etc.

Is it worth it?

- Make the event sourcing implementation as simple as possible
- ... while avoiding the complexities of databases, ORMs, etc.
- ... unless you want or need them :)

Implementation



Implementation

- Events for durability



Implementation

- Events for durability
- Keep current state in RAM (Memory Image)



Implementation

- Events for durability
- Keep current state in RAM (Memory Image)
- Scala case classes to define events and immutable data structures



Implementation

- Events for durability
- Keep current state in RAM (Memory Image)
- Scala case classes to define events and immutable data structures
- Independent components composed into a single application



Simple functionality example

- “CRUD”: no domain logic, so no aggregate
- So we’ll persist events directly from the UI
- Useful to get started
- ... and even complex applications still contain simple functionality

“CRUD”

“CRUD”

Create/Update/Delete

Date range to
All times are shown as Pacific Standard Time (GMT -08:00).

Textarea
Block of help text to describe the field above if need be.

List of options ☒ Option one is this and that—be sure to include why it's great
☐ Option two can is something else and selecting it will deselect options 1

“CRUD”

Create/Update/Delete

Date range to
All times are shown as Pacific Standard Time (GMT -08:00).

Textarea
Block of help text to describe the field above if need be.

List of options ☒ Option one is this and that—be sure to include why it's great
☐ Option two can is something else and selecting it will deselect options 1

HTTP POST

Validate input and
generate event

“CRUD”

Create/Update/Delete

Date range to
All times are shown as Pacific Standard Time (GMT -08:00).

Textarea

Block of help text to describe the field above if need be.

List of options
☒ Option one is this and that—be sure to include why it's great
☐ Option two can is something else and selecting it will deselect options 1

HTTP POST

Validate input and
generate event

Save Event

Event
Store

“CRUD”

Create/Update/Delete

Date range: May 1, 2011 12:00am to May 8, 2011 11:59pm
All times are shown as Pacific Standard Time (GMT -08:00).

Textarea
Block of help text to describe the field above if need be.

List of options:
☒ Option one is this and that—be sure to include why it's great
☐ Option two can is something else and selecting it will deselect options 1

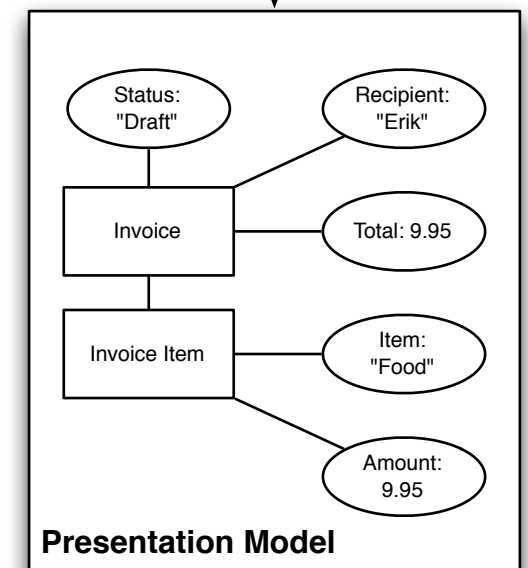
HTTP POST

Validate input and
generate event

Save Event

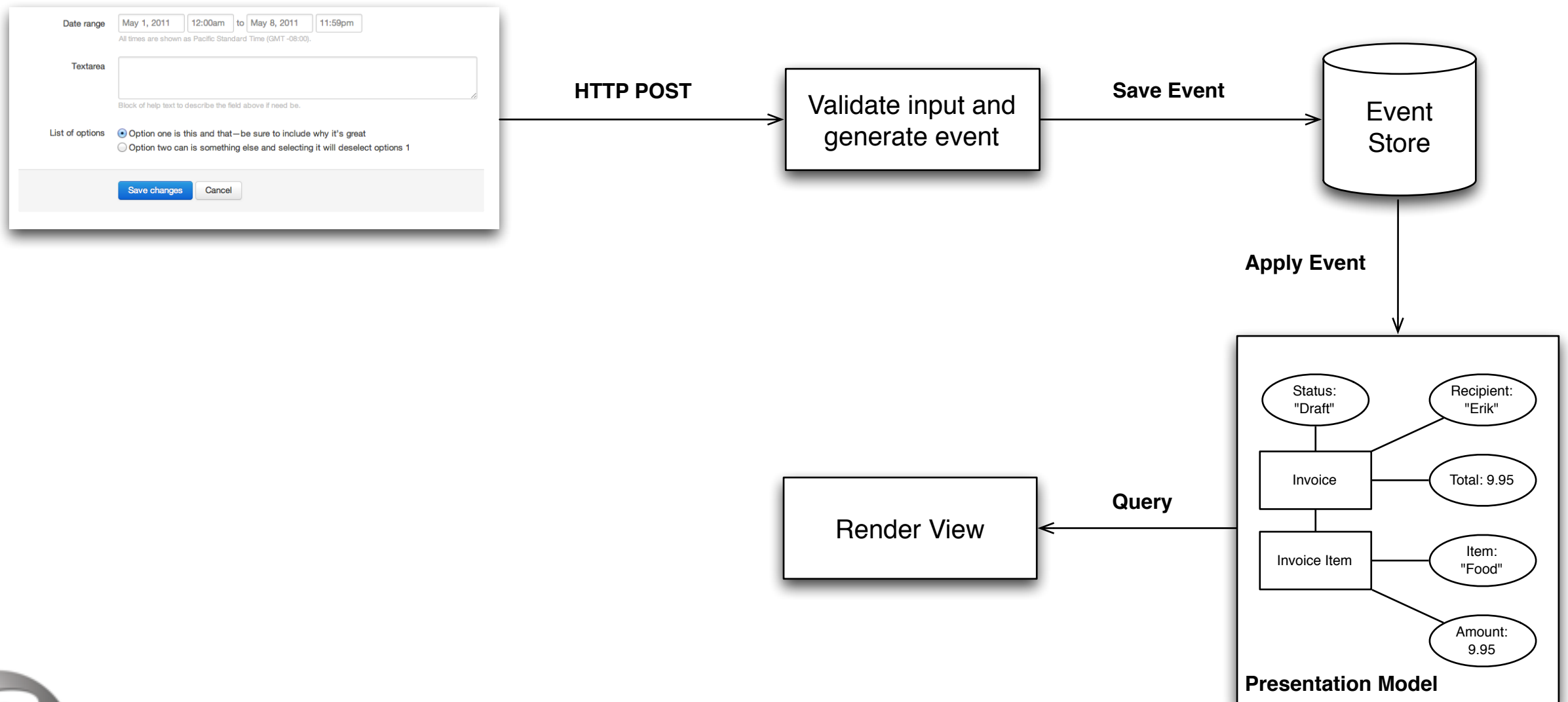
Event
Store

Apply Event



“CRUD”

Create/Update/Delete



“CRUD”

Create/Update/Delete

Date range: May 1, 2011 12:00am to May 8, 2011 11:59pm
All times are shown as Pacific Standard Time (GMT -08:00).

Textarea
Block of help text to describe the field above if need be.

List of options:
☒ Option one is this and that—be sure to include why it's great
☐ Option two can is something else and selecting it will deselect options 1

HTTP POST

Validate input and
generate event

Save Event

Event
Store

Apply Event

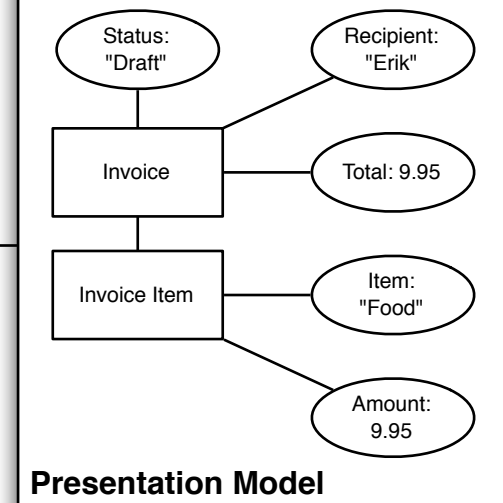
Read

#	First Name	Last Name	Language
1	Some	One	English
2	Joe	Sixpack	English
3	Stu	Dent	Code

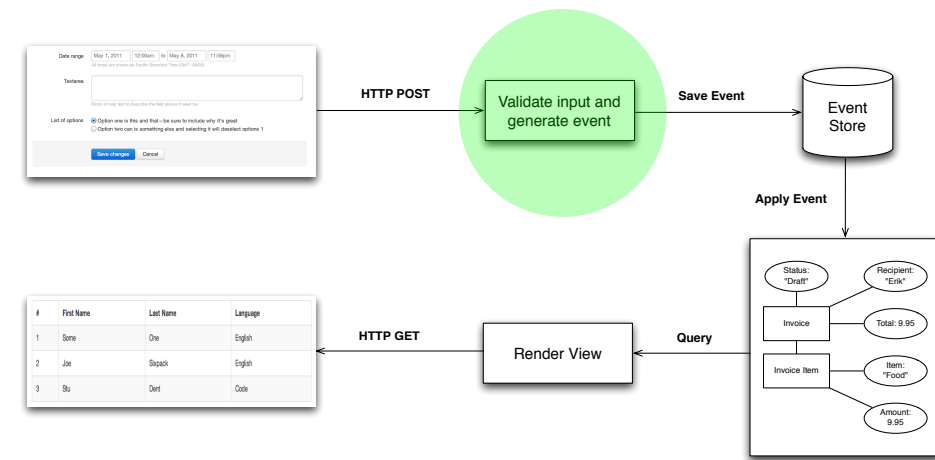
HTTP GET

Render View

Query



Controller

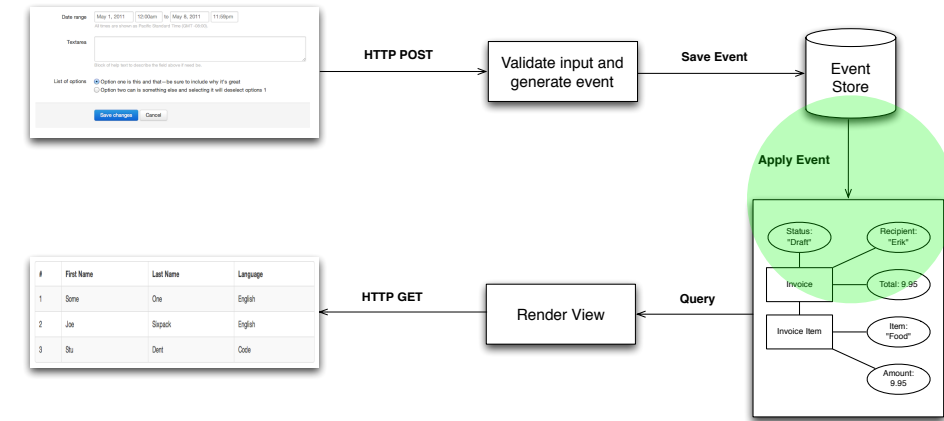


```

post("/todo") {
  field("text", required) match {
    case Success(text) =>
      commit(ToDoItemAdded(ToDoItem(UUID.randomUUID(), text)))
      redirect(url("/todo"))
    case Failure(error) =>
      new ToDoView(toDoItems, Some(error)),
  }
}

```

Memory Image



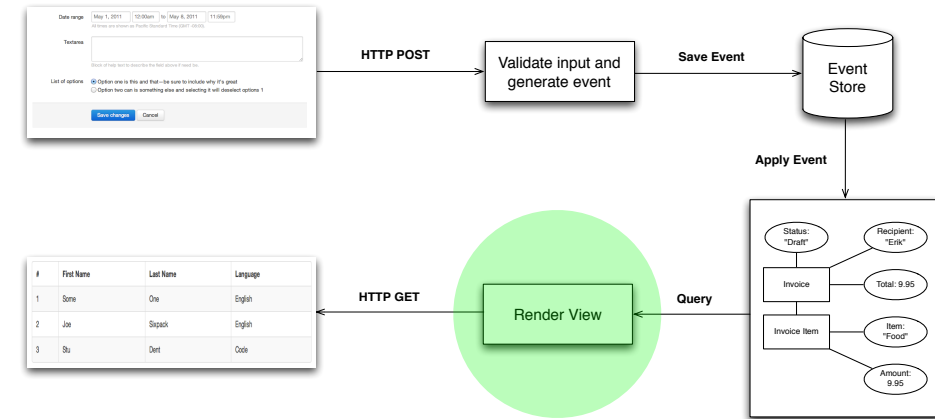
```

case class TodoItems (
  all          : Map[UUID, TodoItem] = Map.empty,
  recentlyAdded: Vector[UUID]       = Vector.empty) {

  def apply(event: TodoItemEvent) = event match {
    case TodoItemAdded(item) =>
      copy(all + (item.id -> item), recentlyAdded :+ item.id)
    // [... handle other event types ...]
  }

  def mostRecent(count: Int) = recentlyAdded.takeRight(count).map(all).reverse
}
  
```

View

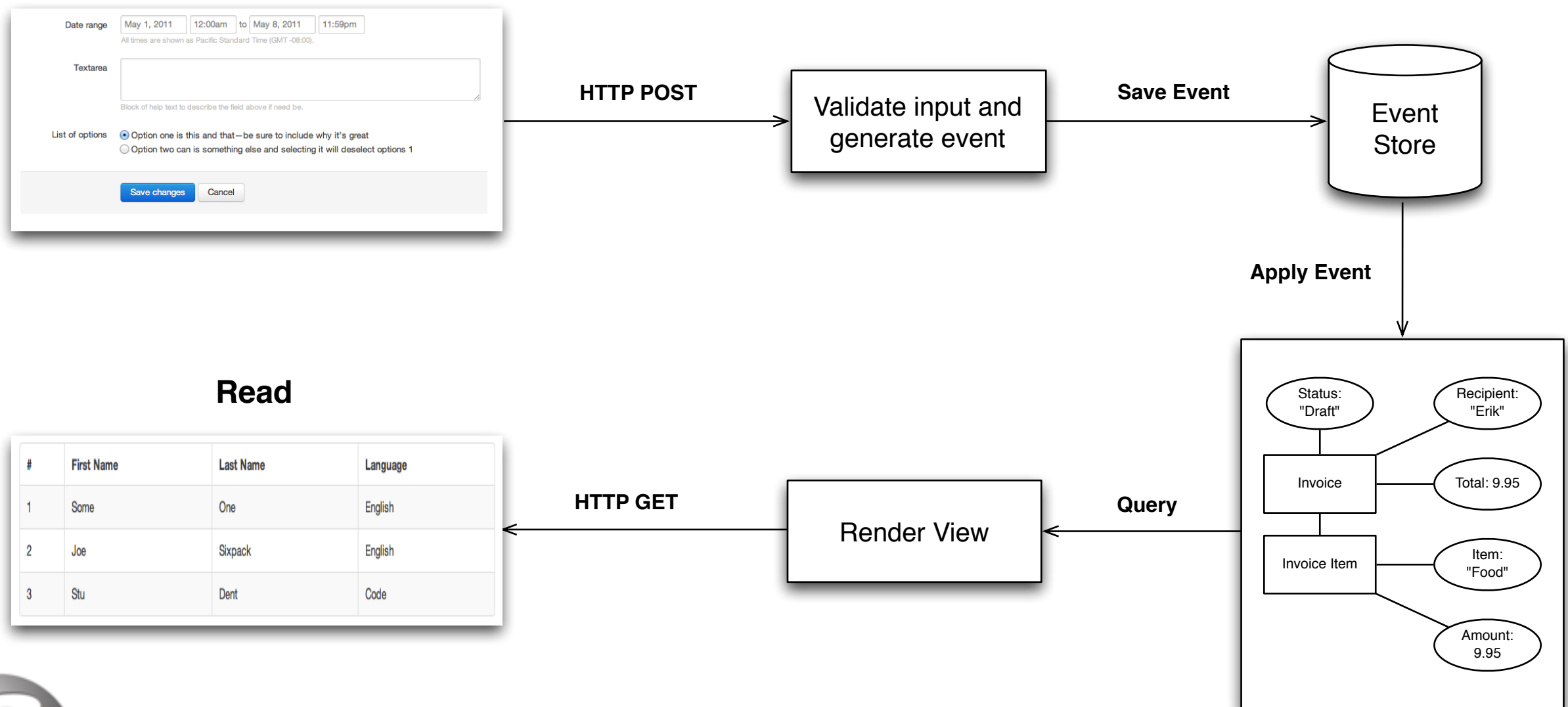


```

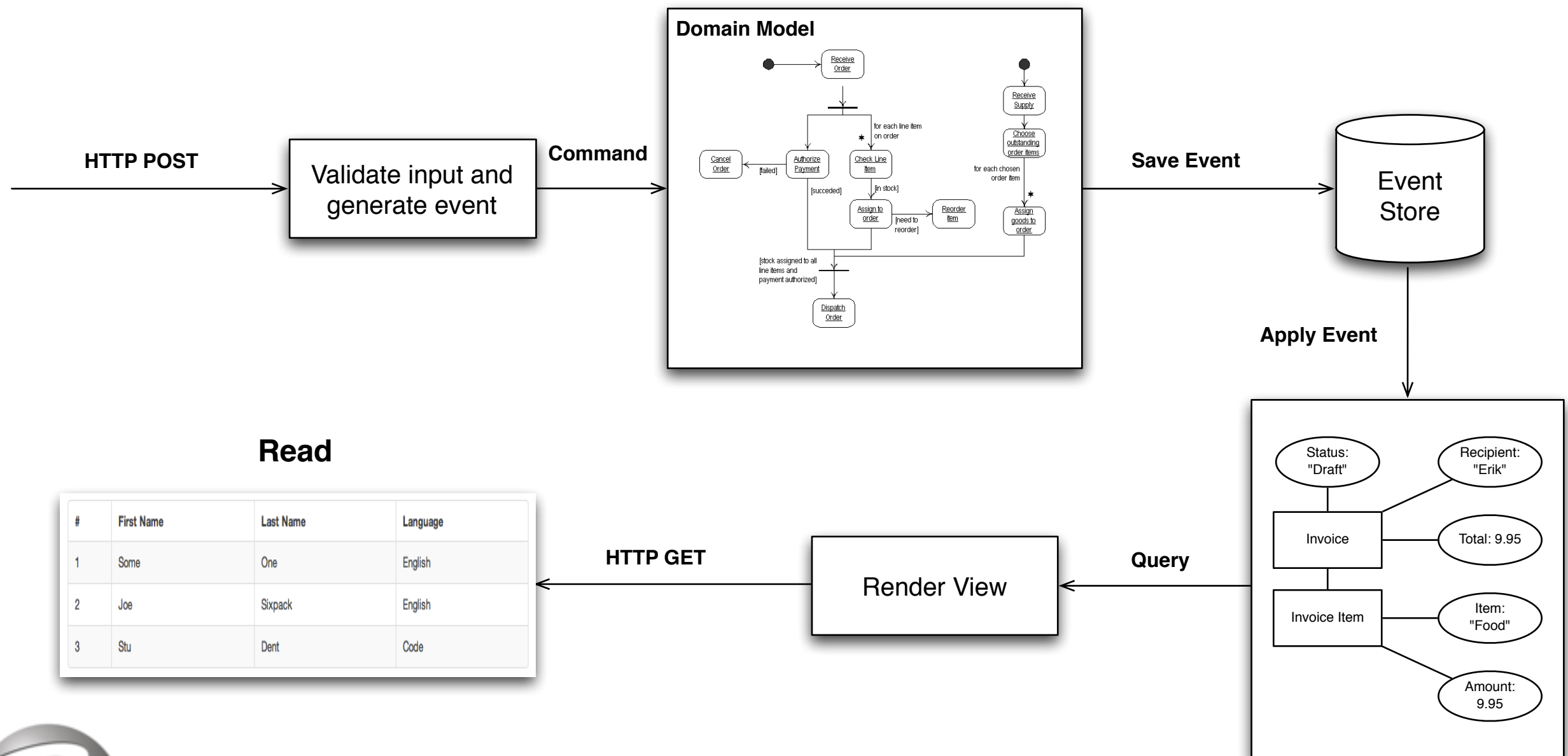
<table class="zebra-striped">
  <thead><tr><th>To-Do</th></tr></thead>
  <tbody>{
    for (item <- todoItems.mostRecent(20)) yield {
      <tr><td>{ item.text }</td></tr>
    }
  }</tbody>
</table>

```

Domain Driven Design



Domain Driven Design



Domain Code

```
sealed trait InvoiceEvent
case class InvoiceCreated() extends InvoiceEvent
case class InvoiceRecipientChanged(recipient: String) extends InvoiceEvent
case class InvoiceItemAdded(item: InvoiceItem, totalAmount: BigDecimal) extends InvoiceEvent
case class InvoiceSent(sentOn: LocalDate, paymentDueOn: LocalDate) extends InvoiceEvent
```

Domain Code

```
case class DraftInvoice(  
  recipient: Option[String] = None,  
  nextItemId: Int = 1,  
  items: Map[Int, InvoiceItem] = Map.empty) extends Invoice {  
  
  def addItem(description: String, amount: BigDecimal): Behavior[DraftInvoice] = ...  
  
  // [... other domain logic ...]  
  
  private def itemAdded = when[InvoiceItemAdded] { event =>  
    // ...  
  }  
  
}
```

Domain Code

```
case class DraftInvoice(  
  recipient: Option[String] = None,  
  nextItemId: Int = 1,  
  items: Map[Int, InvoiceItem] = Map.empty) extends Invoice {
```

The “Brains”

```
  def addItem(description: String, amount: BigDecimal): Behavior[DraftInvoice] = ...  
  
  // [... other domain logic ...]
```

```
  private def itemAdded = when[InvoiceItemAdded] { event =>  
    // ...  
  }  
  
}
```

Domain Code

```
case class DraftInvoice(  
  recipient: Option[String] = None,  
  nextItemId: Int = 1,  
  items: Map[Int, InvoiceItem] = Map.empty) extends Invoice {
```

The “Brains”

```
  def addItem(description: String, amount: BigDecimal): Behavior[DraftInvoice] = ...  
  
  // [... other domain logic ...]
```

The “Muscle”

```
  private def itemAdded = when[InvoiceItemAdded] { event =>  
    // ...  
  }
```

```
}
```

Domain Code

```
case class DraftInvoice(  
  recipient: Option[String] = None,  
  nextItemId: Int = 1,  
  items: Map[Int, InvoiceItem] = Map.empty) extends Invoice {  
  
  // ...  
  
  def addItem(description: String, amount: BigDecimal): Behavior[DraftInvoice] =  
    itemAdded(InvoiceItemAdded(InvoiceItem(nextItemId, description, amount), totalAmount + amount))  
  
  private def totalAmount = items.values.map(_.amount).sum  
  
  private def readyToSend_? = recipient.isDefined && items.nonEmpty  
  
  // ...  
  
  private def itemAdded = when[InvoiceItemAdded] { event =>  
    copy(nextItemId = nextItemId + 1, items = items + (event.item.id -> event.item))  
  }  
}
```

Domain Code

```
case class DraftInvoice(  
  recipient: Option[String] = None,  
  nextItemId: Int = 1,  
  items: Map[Int, InvoiceItem] = Map.empty) extends Invoice {  
  
  ...  
  
  /** Reload from history. */  
  protected[this] def applyEvent = recipientChanged orElse itemAdded orElse sent  
  
  ...  
}
```

Cooperating Users

Create/Update/Delete

Date range: May 1, 2011 12:00am to May 8, 2011 11:59pm
All times are shown as Pacific Standard Time (GMT -08:00).

Textarea
Block of help text to describe the field above if need be.

List of options:
☒ Option one is this and that—be sure to include why it's great
☐ Option two can is something else and selecting it will deselect options 1

HTTP POST

Validate input and
generate event

Save Event

Event
Store

Apply Event

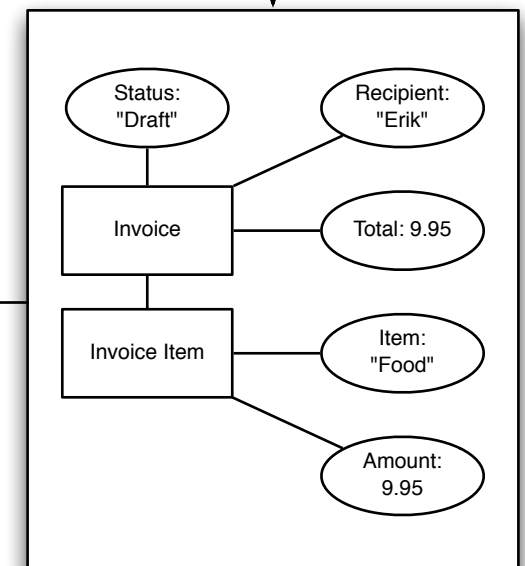
Read

#	First Name	Last Name	Language
1	Some	One	English
2	Joe	Sixpack	English
3	Stu	Dent	Code

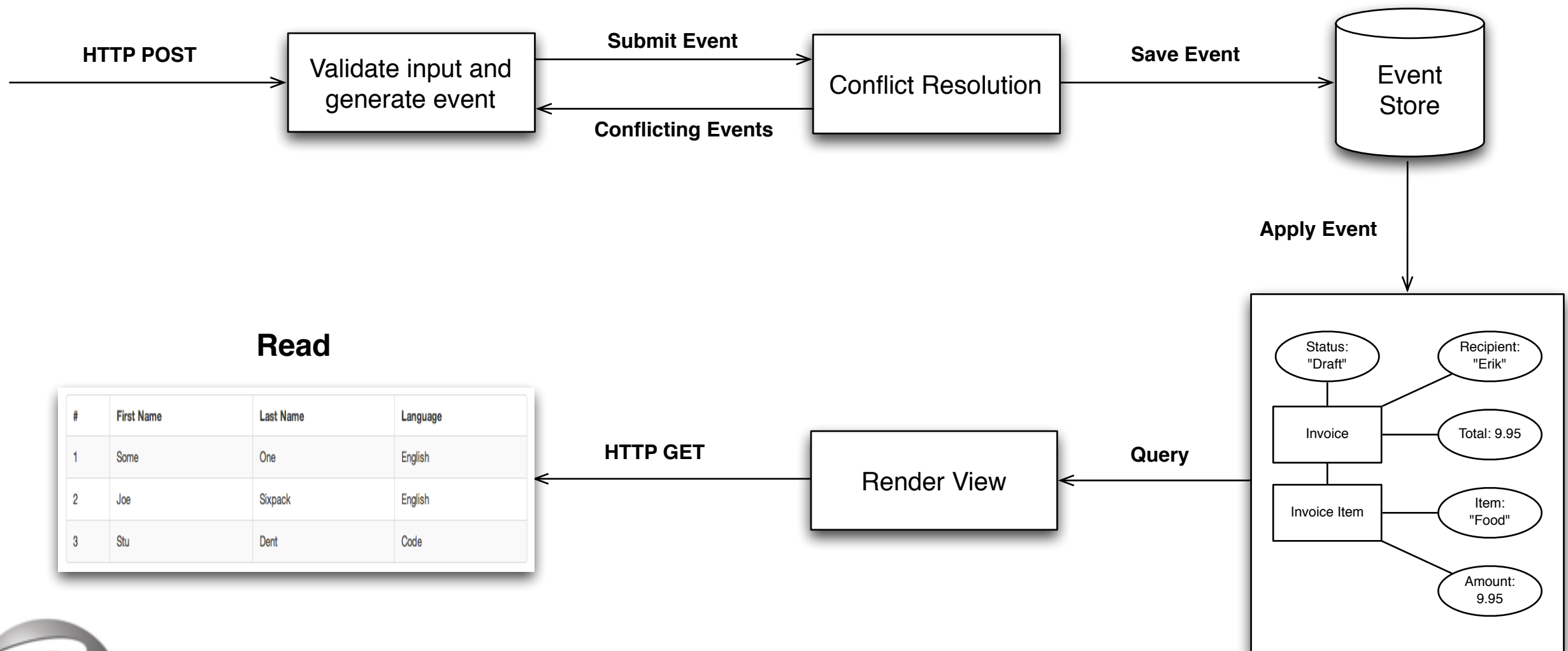
HTTP GET

Render View

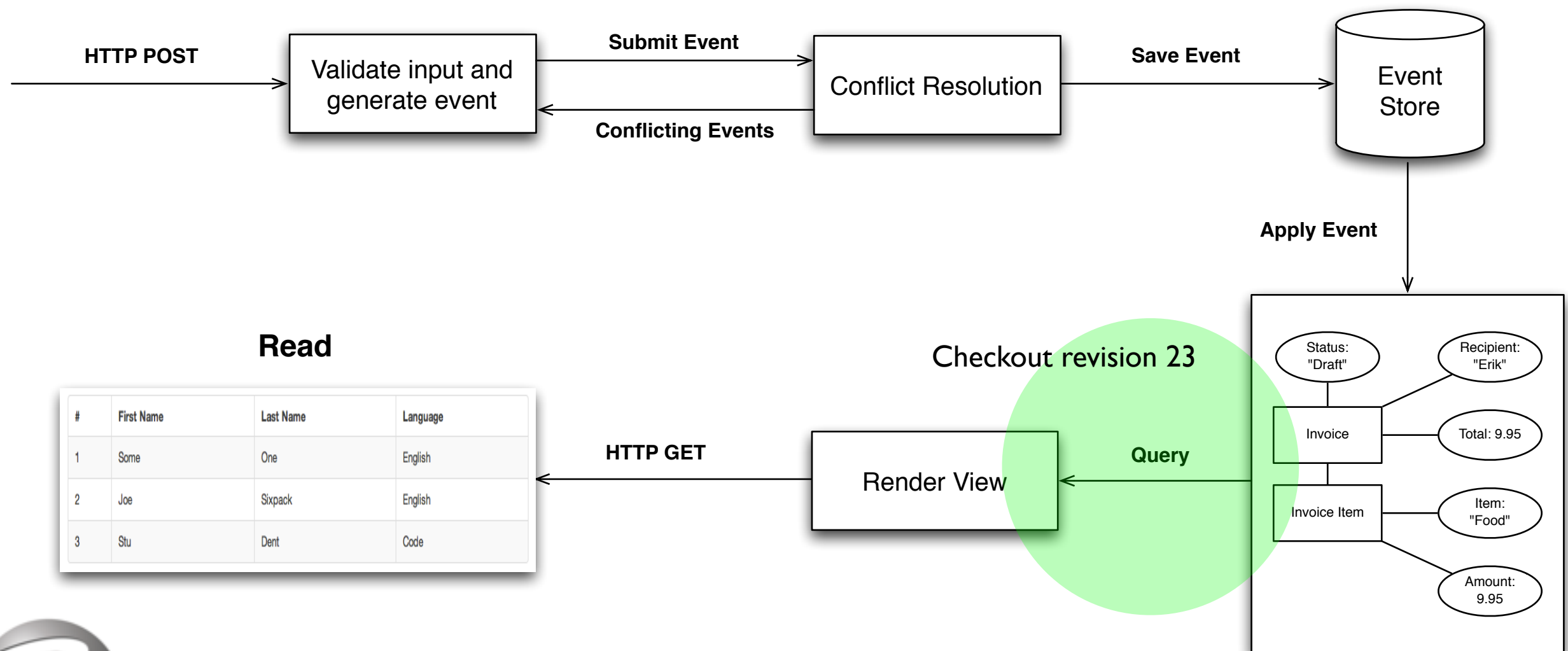
Query



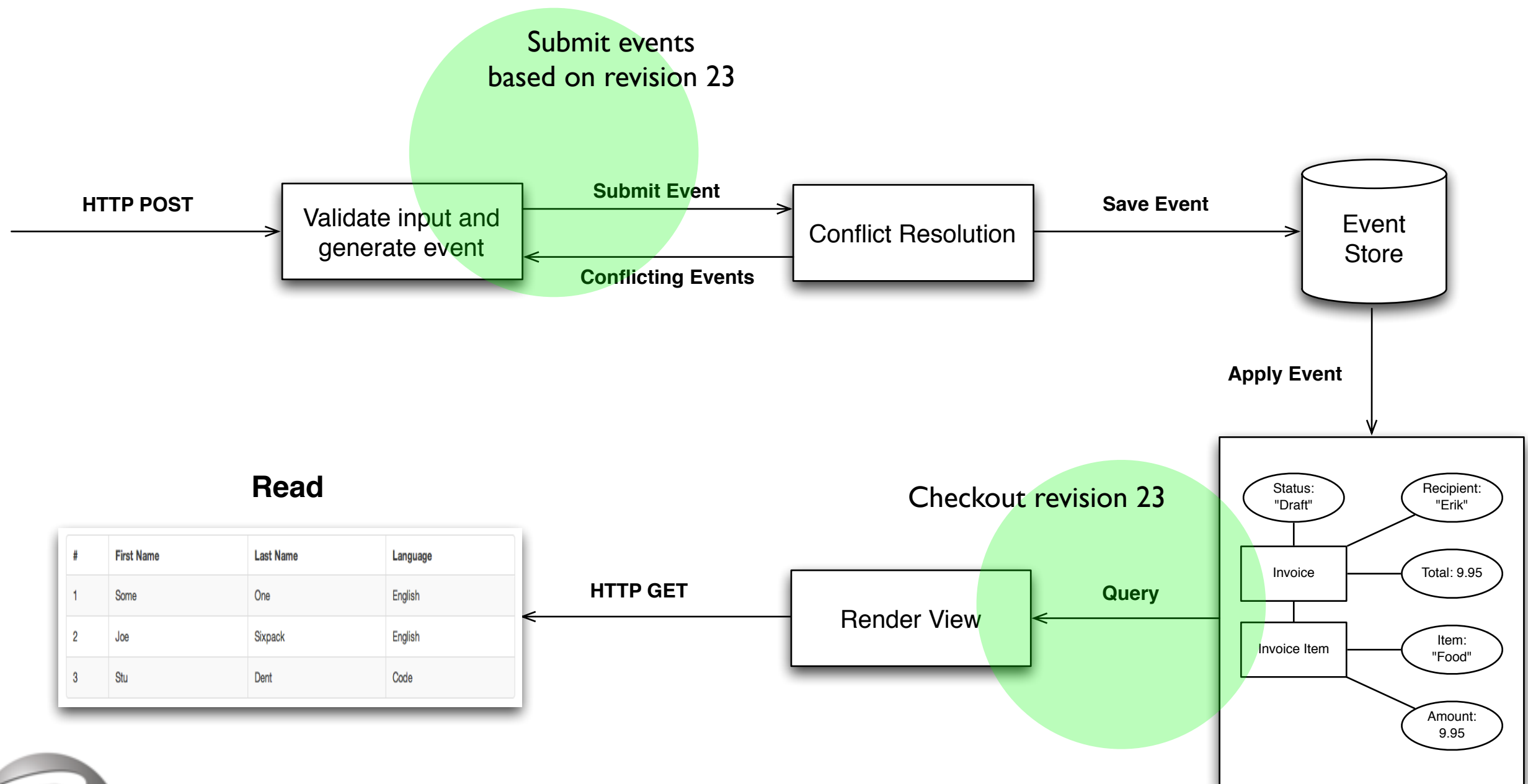
Cooperating Users



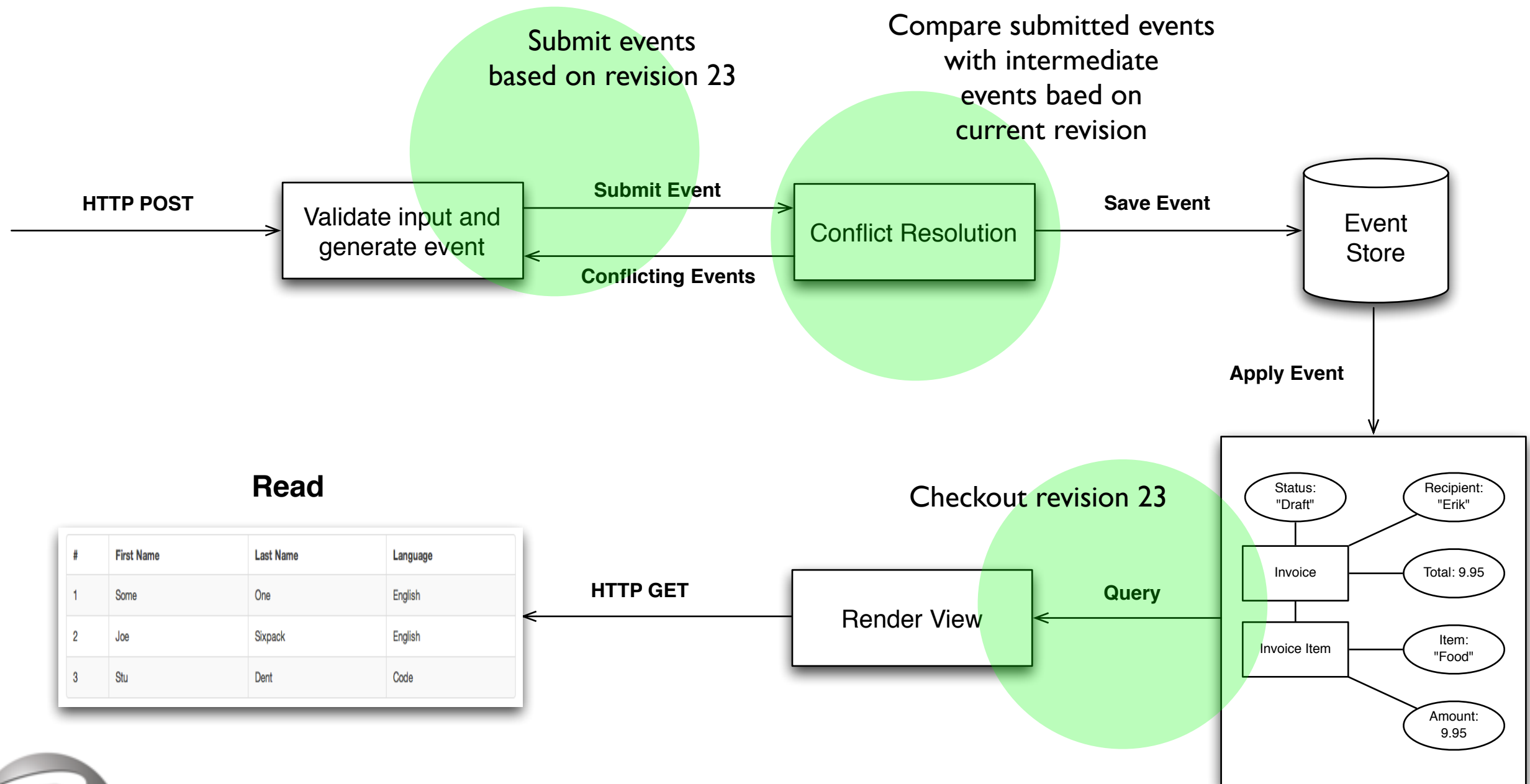
Cooperating Users



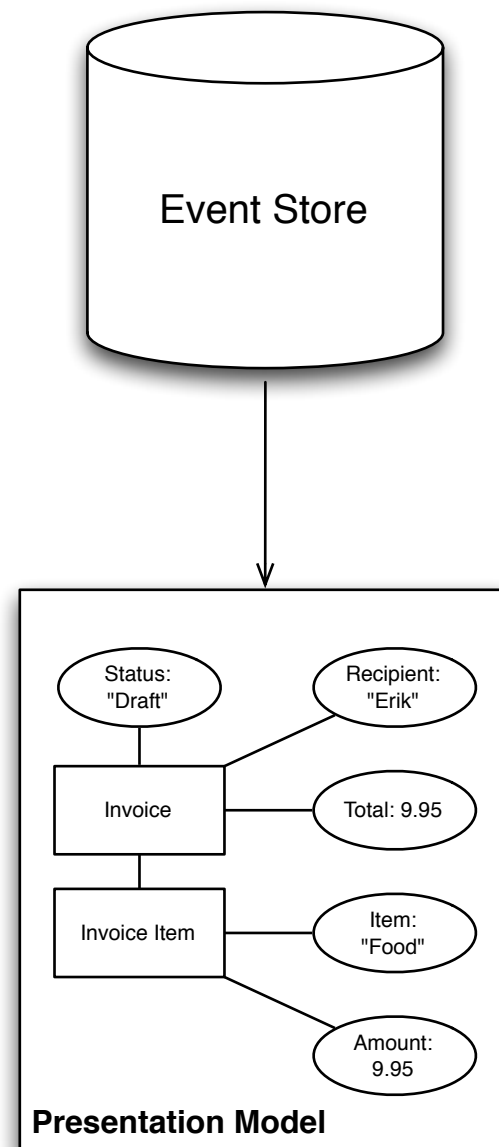
Cooperating Users



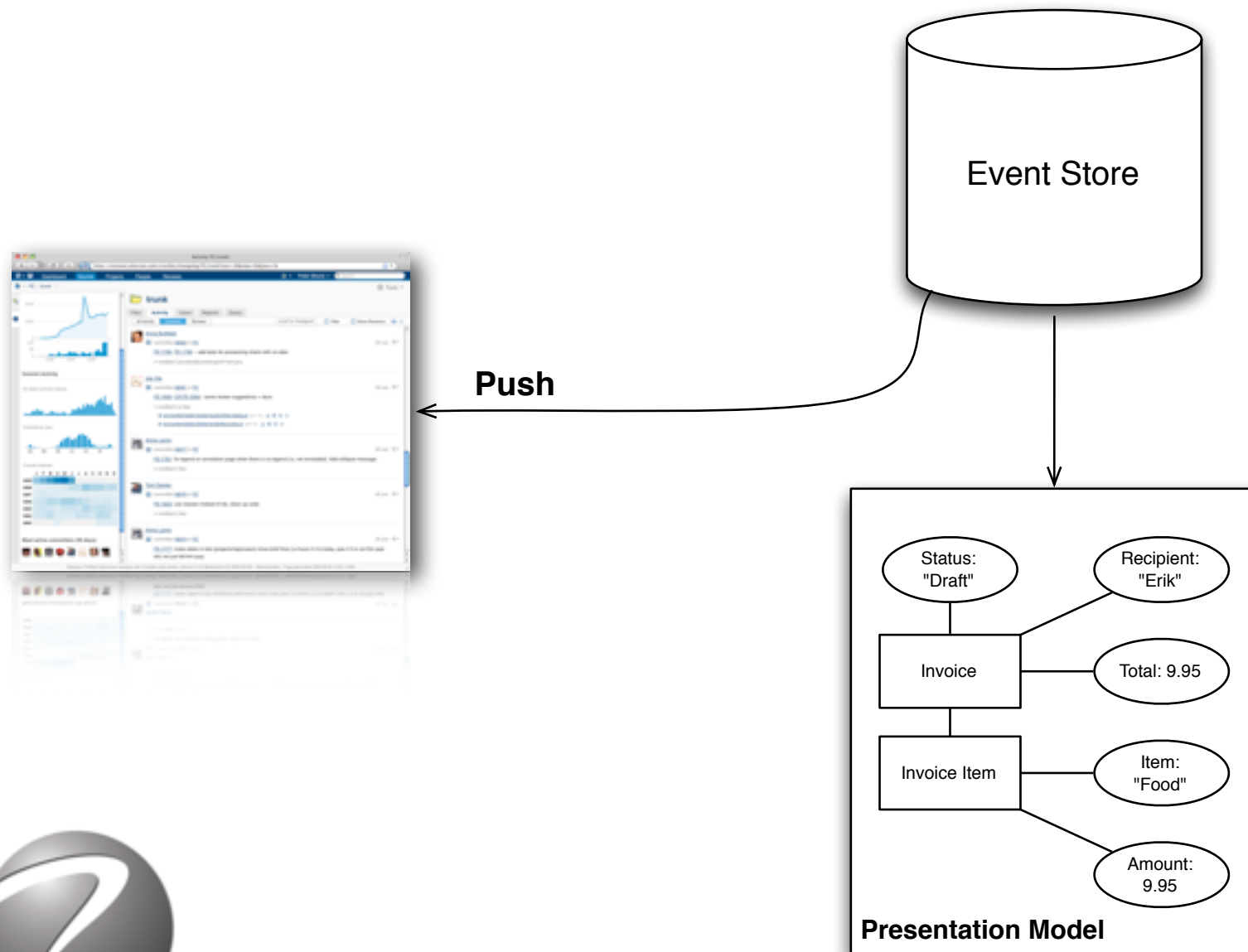
Cooperating Users



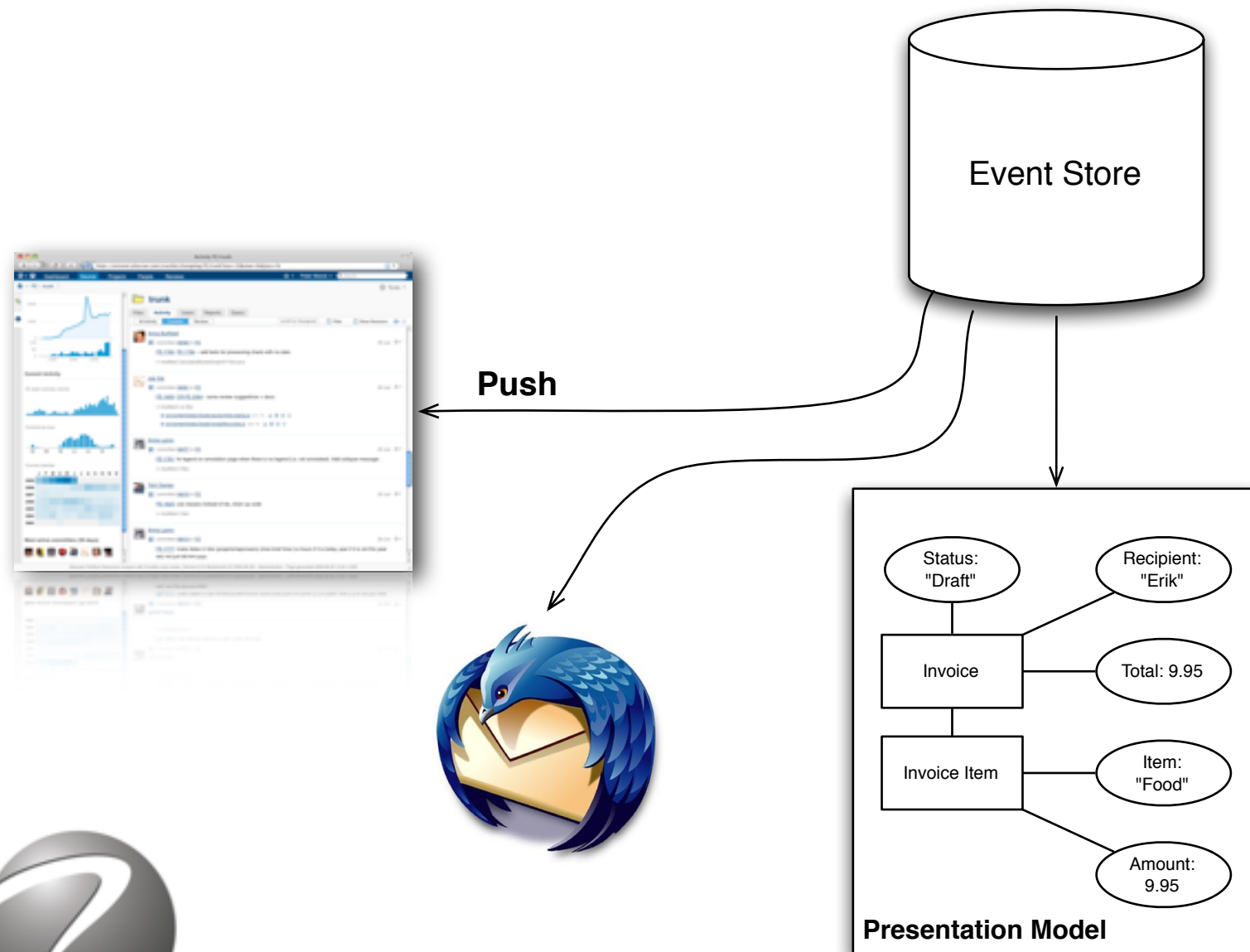
Events



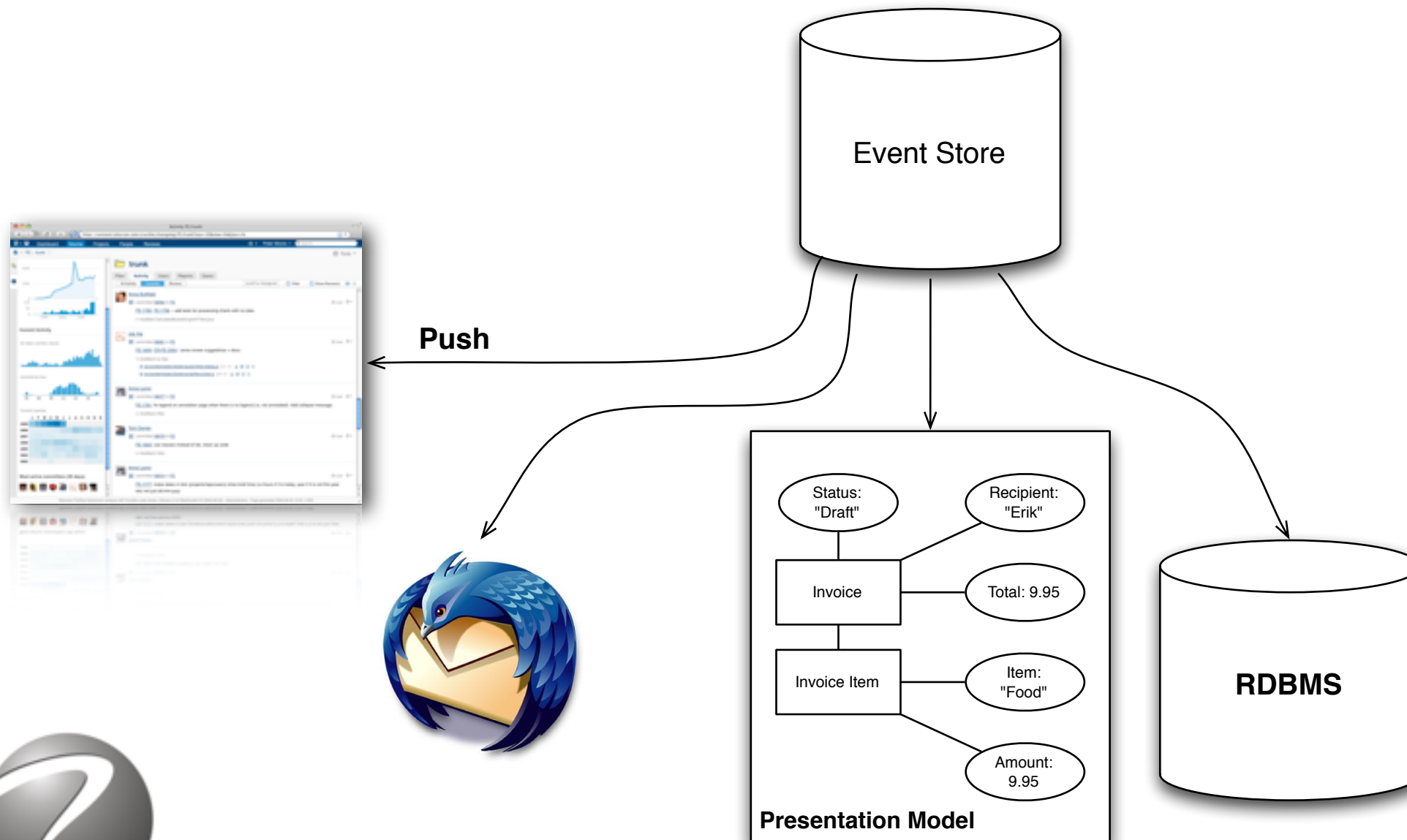
Events



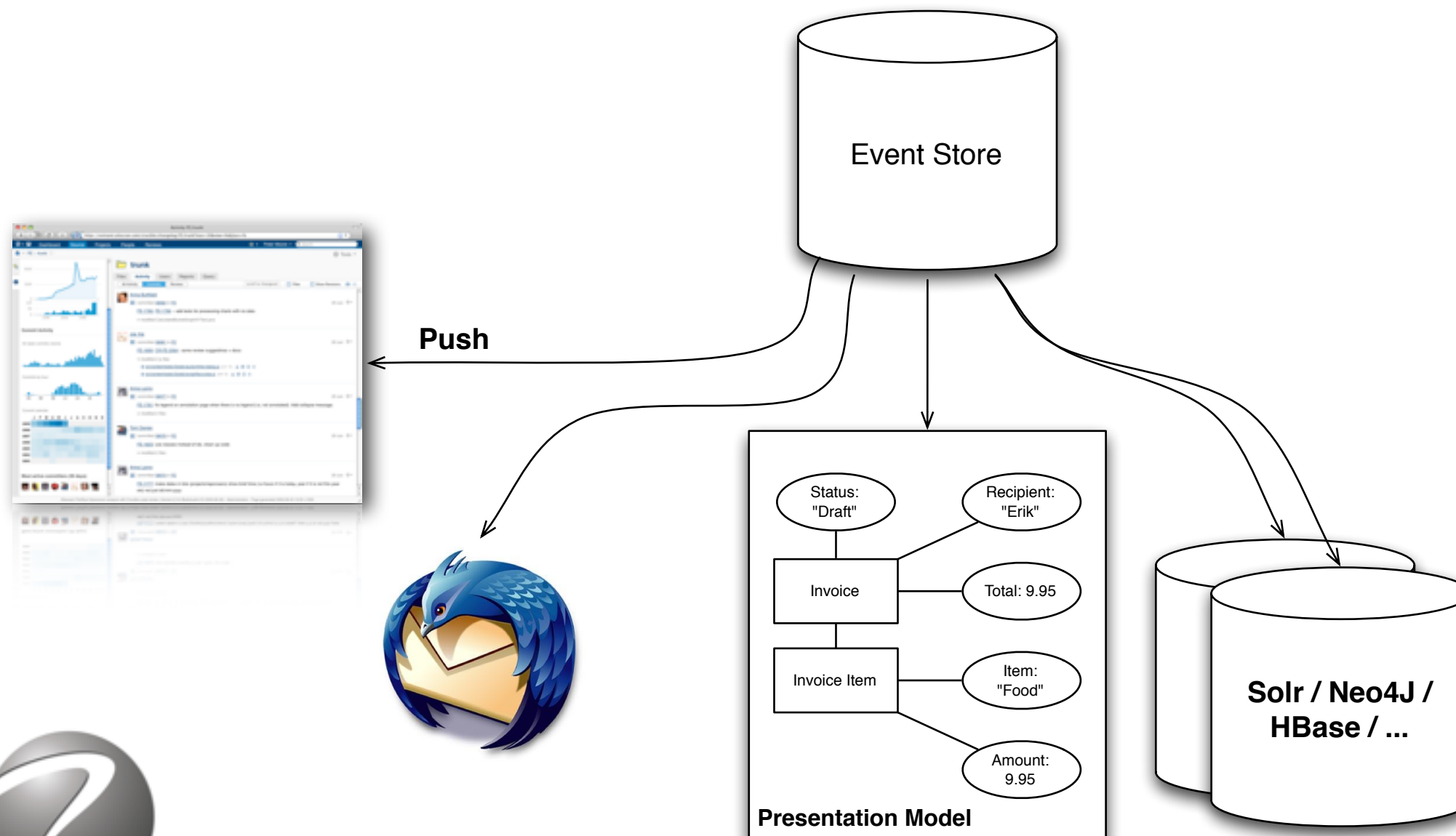
Events



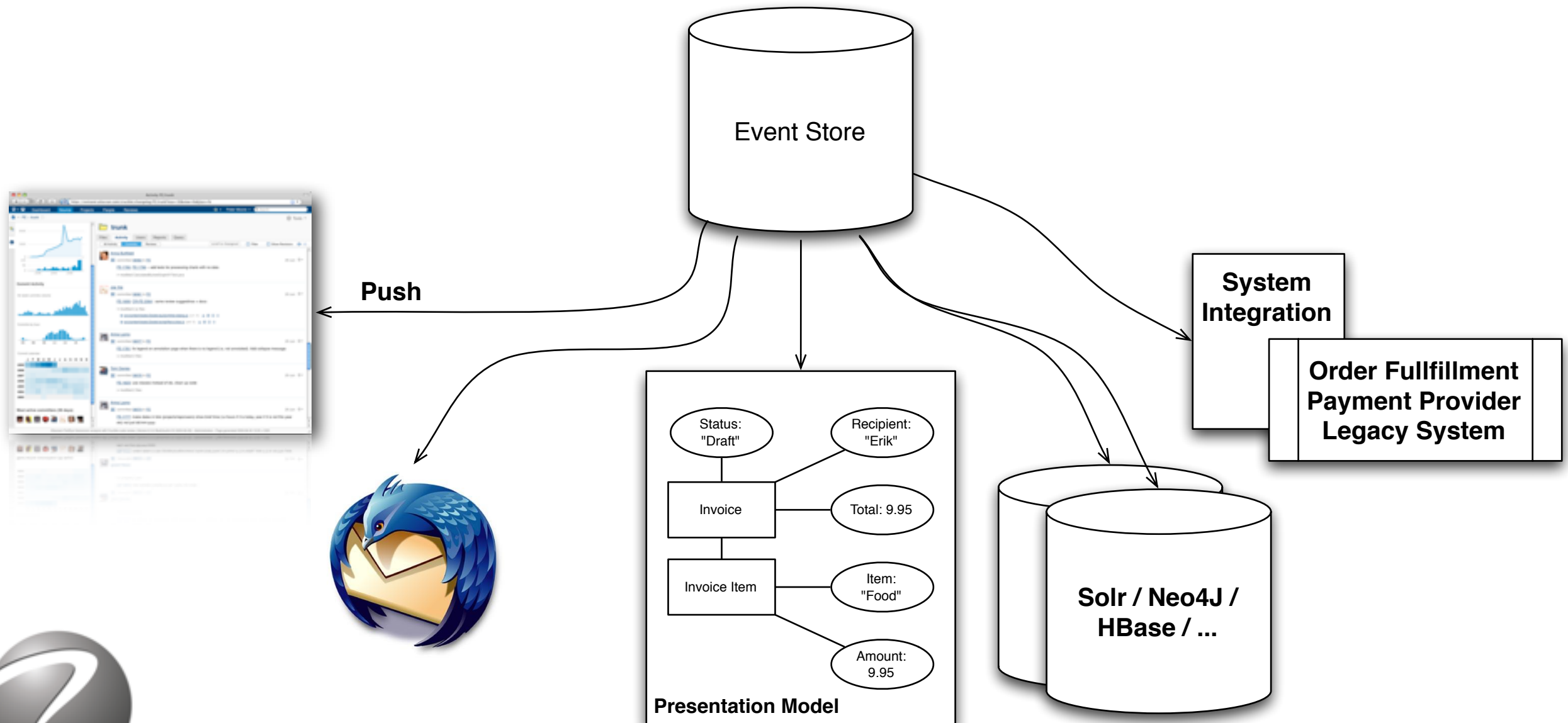
Events



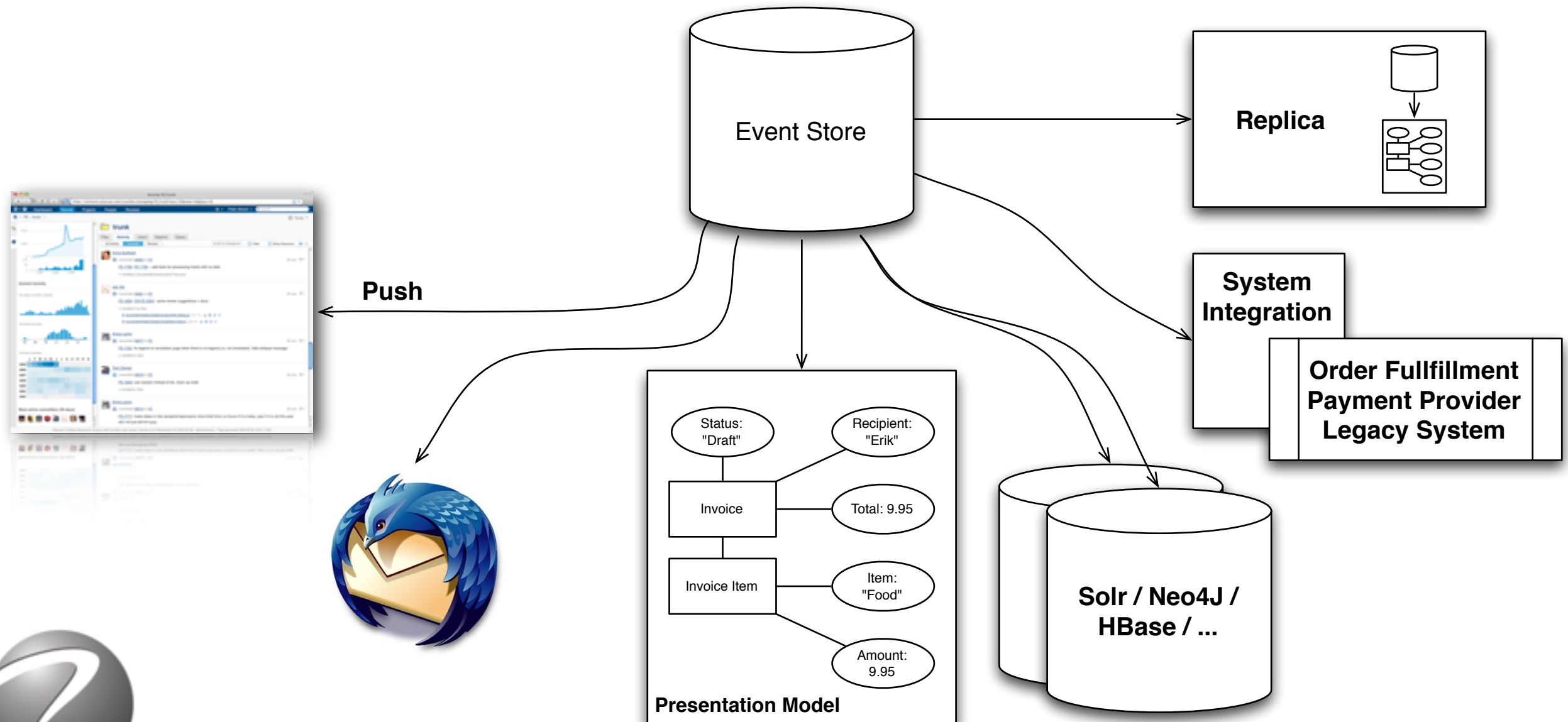
Events



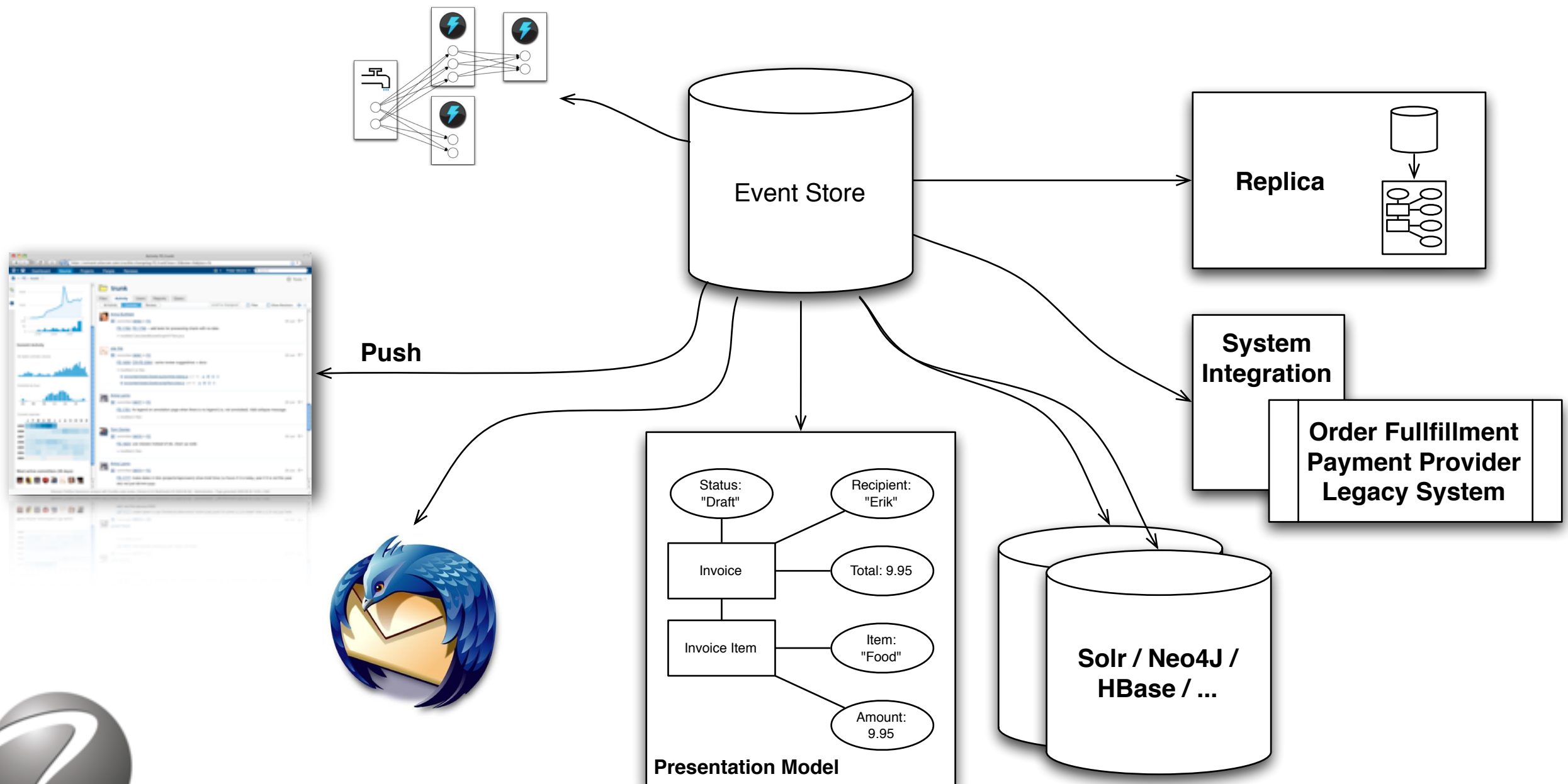
Events



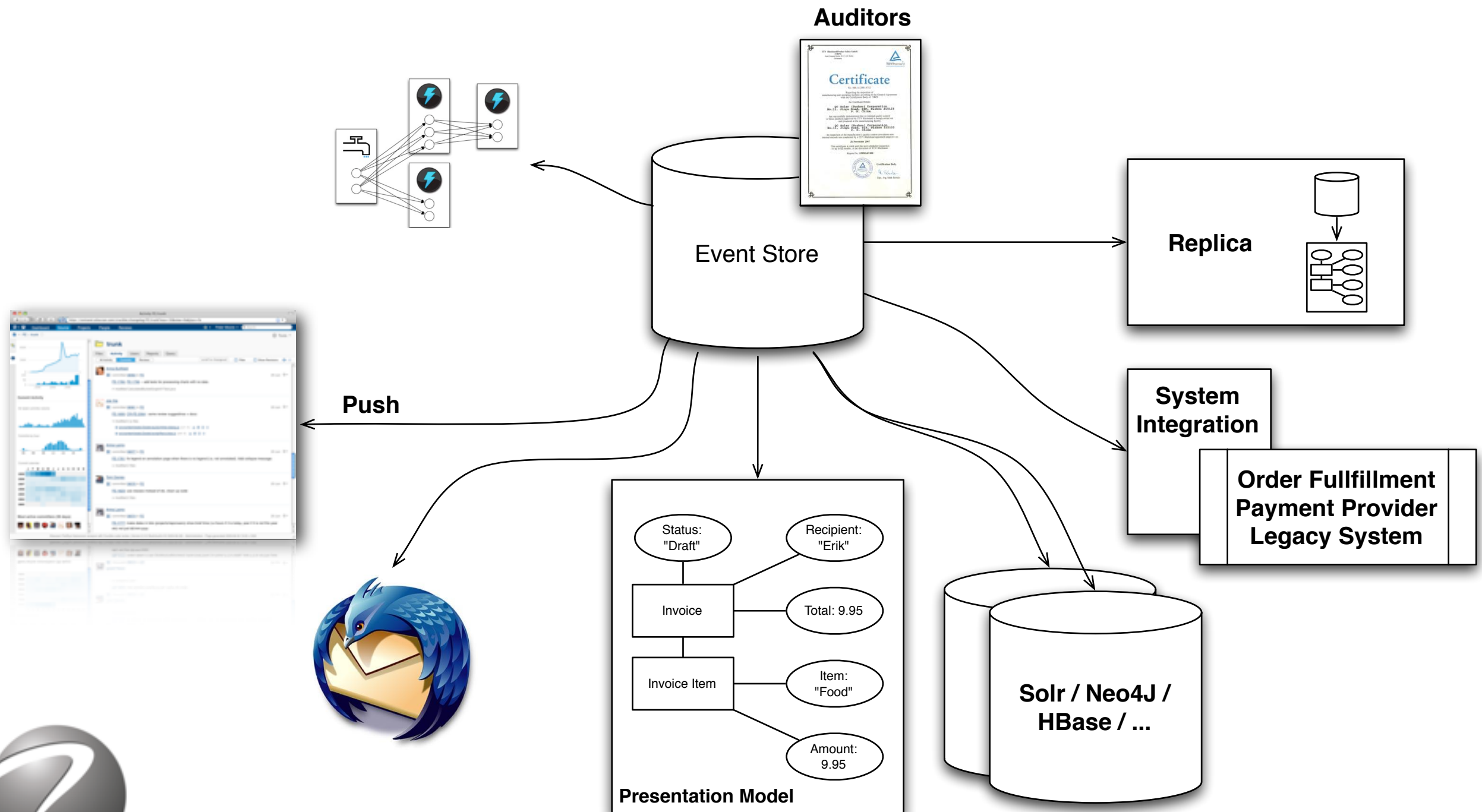
Events



Events



Events



Event Store

- Stores sequence of events
- Dispatches events when successfully committed
- Replays events on startup
- It's your application's transaction log

Writing Events to Disk

```
private class JournalFileWriter(file: File, var sequence: Long) {  
    private val checksum = new CRC32()  
    private val fileOutputStream = new FileOutputStream(file)  
    private val dataOutputStream = new DataOutputStream(  
        new CheckedOutputStream(  
            new BufferedOutputStream(fileOutputStream), checksum))  
  
    def write(commit: Array[Byte]) {  
        sequence += 1  
        checksum.reset()  
        dataOutputStream.writeLong(sequence)  
        dataOutputStream.writeInt(commit.size)  
        dataOutputStream.write(commit)  
        dataOutputStream.writeInt(checksum.getValue().toInt)  
    }  
  
    def sync(metadata: Boolean = false) {  
        dataOutputStream.flush()  
        fileOutputStream.getChannel().force(metadata)  
    }  
}
```

Example Application

- <https://github.com/erikrozendaal/scala-event-sourcing-example>
- (soon) <https://github.com/zilverline/lessdb-example>
- Blog series coming soon at <http://blog.zilverline.com>

Implementation Limitations

- Single-threaded event store using disruptor pattern
 - ~ 5.000 commits per second
- Number of events to replay on startup
 - ~ 70.000 JSON serialized events/second
 - ~ 200.000 protobuf serialized events/second
- Total number of objects to store in main memory
 - JVM can run with large heaps (tens of gigabytes)



But ready to scale

- Advanced event store implementations support SQL, MongoDB, Amazon SimpleDB, etc.
- Use persisted view model for high volume objects (fixes startup time and memory usage)
- Easy partitioning of aggregates (consistency boundary with unique identifier)
- Load aggregates on-demand, use snapshotting

Conclusion

- Fully capture historical information
- Independent components where each part of the application *can* use its own data model
- Easier to fully understand compared to traditional ORM based approach
- Need to learn “Event-Driven” thinking

Thanks!



silverline
ONLY RESULTS MATTER

References

- Example code <https://github.com/erikrozendaal/scala-event-sourcing-example>
- CQRS <http://cqrsinfo.com/>
- Greg Young, “Unshackle Your Domain”
<http://www.infoq.com/presentations/greg-young-unshackle-qcon08>
- Pat Helland, “Life Beyond Distributed Transactions: An Apostate's Opinion”
<http://www.ics.uci.edu/~cs223/papers/cidr07p15.pdf>
- Erik Rozendaal, “Towards an immutable domain model” <http://blog.zilverline.com/2011/02/10/towards-an-immutable-domain-model-monads-part-5/>
- Martin Fowler, “Memory Image”, <http://martinfowler.com/bliki/MemoryImage.html>
- Martin Fowler, “The LMAX Architecture”, <http://martinfowler.com/articles/lmax.html>

