Credit Card Fraud Detection
A Modern Architecture

Tuesday, August 22, 2017

Colin MacNaughton and Igor Mihaljevic

</>kode41 Neeve Research

Introductions

Colin MacNaughton

Head of Engineering at Neeve Research, the creators of the X Platform: a platform for building In Memory enterprise applications that are high performance, easy to author, and easy to maintain.

Igor Mihaljevic

Lead Engineer a Kode41, a services firm specializing in high performance system design and development. Igor's professional focus is software architecture of Ad Bidding, Online Games, and Social Network analysis with BigData.

Who is Neeve Research?

- Headquartered in Silicon Valley
- Creators of the X Platform™- Memory Oriented Application Platform.
- Passionate about high performance computing.
- Running in production at Fortune 100-300

What does Fraud Detection In our Connected World look like?

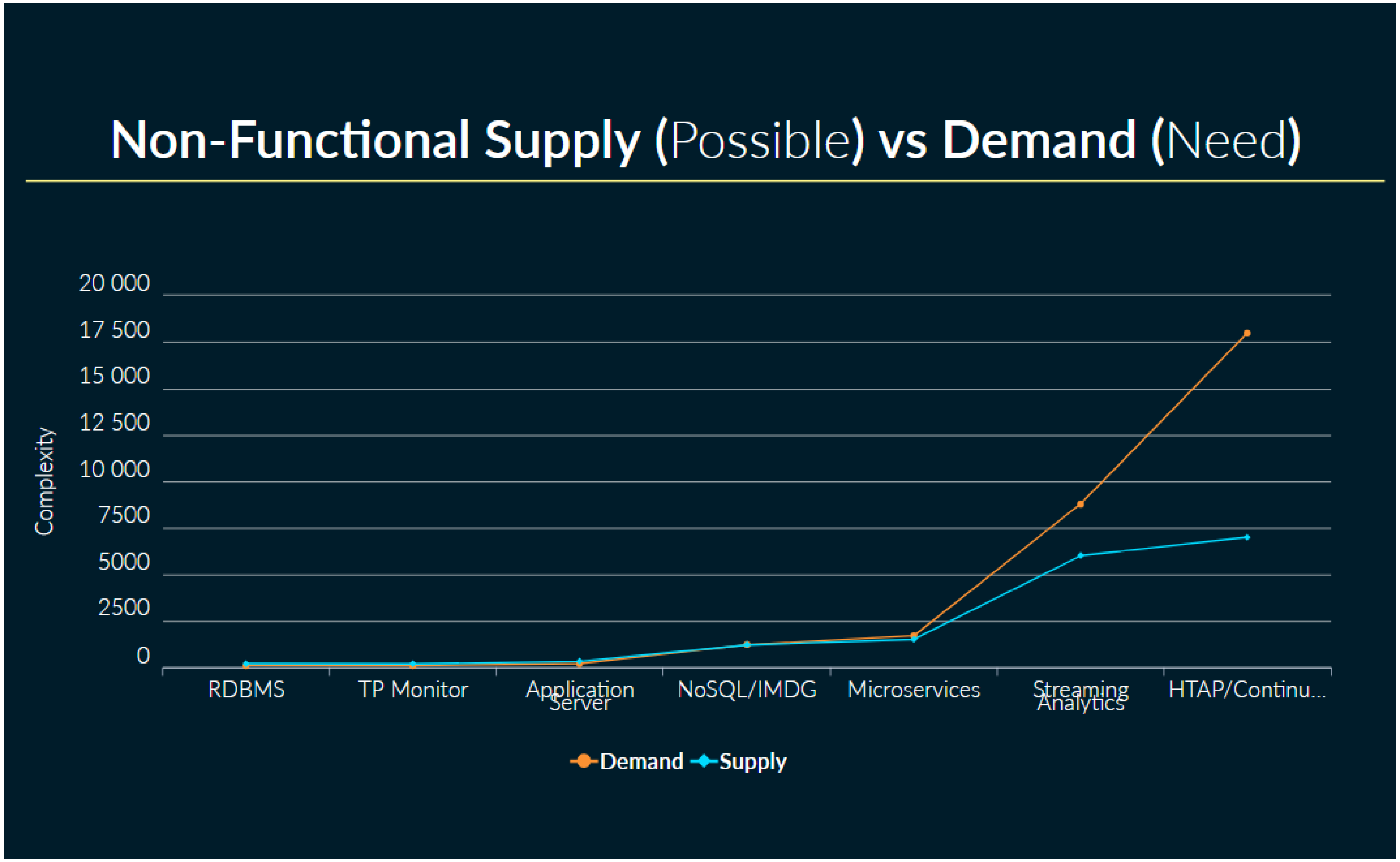
Rapid Growth in E-commerce = Rapid Growth in Fraud

Rapid Growth = Rapid Response

Considerations:

1. Leverage Detection strategies
2. Reduce Impact on user/customer experience
3. Control Cost

Increasing Complexity and Demand



Enabling Technologies

Microservices (Multi Agent Architectures)

- Break down applications into business functions with <u>private</u> data that communicate via messaging -> Agility, Innovation, <u>*Scalability*</u>

Stream Processing and Analytics

- Continuous analytics on data in motion replaces batch processing -> provides real time insight from diverse sources.

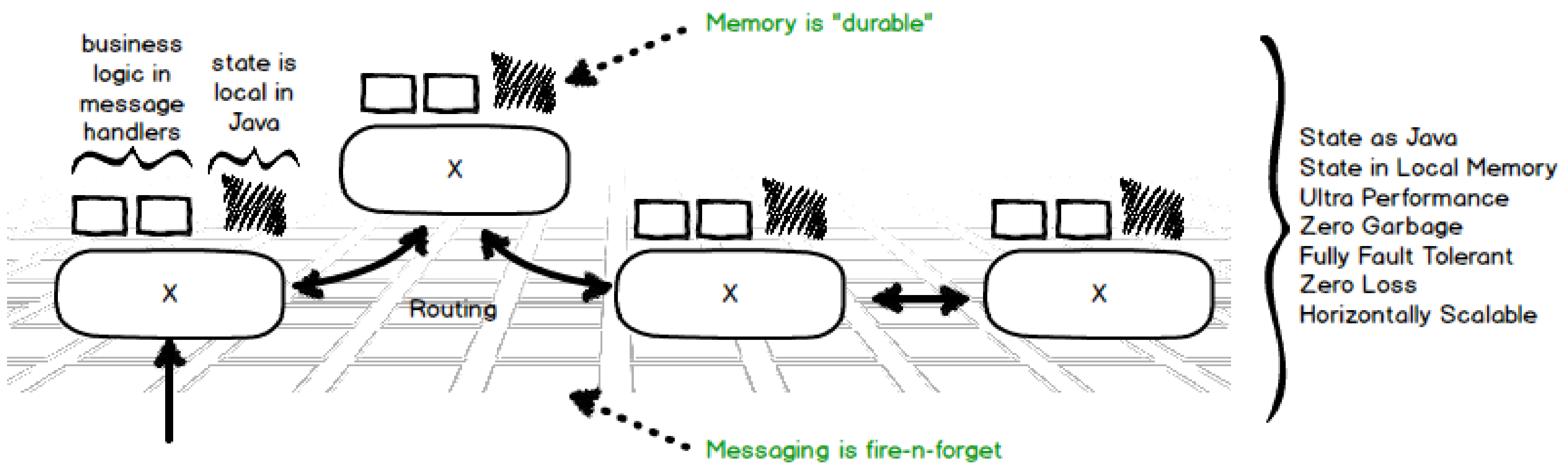
HTAP

- Leverages In Memory Computing Techniques -> allows near real time analytical processing on operational data without impacting operational updates.

What Is X?

The X Platform is a *memory-oriented* platform for building *multi-agent, transactional* Java-based enterprise applications.

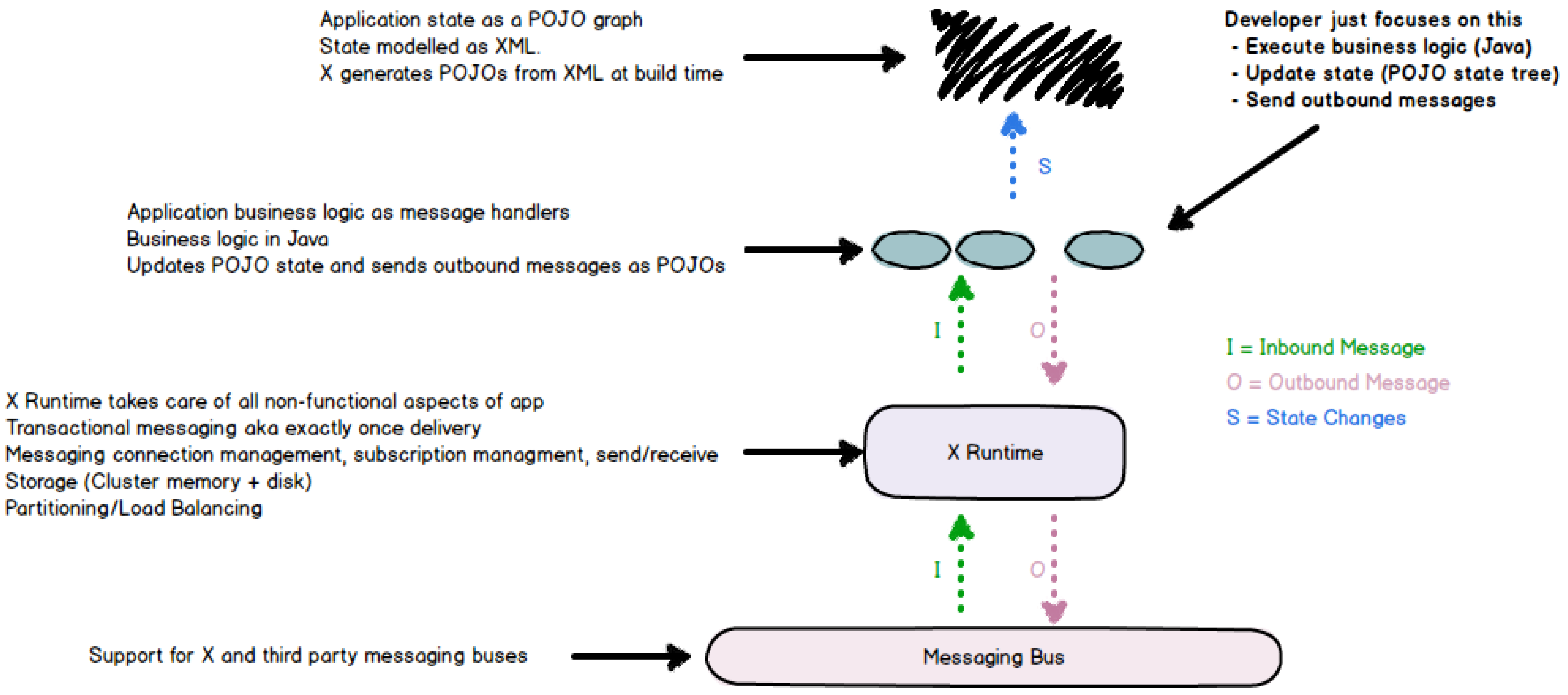
The Big Picture



- ✓ Message Driven
- ✓ Stateful
- ✓ Multi-Agent
- ✓ Totally Available
- ✓ Horizontally Scalable
- ✓ Ultra Performant

</>kode41 NEEVE RESEARCH

An X Application



Application state as a POJO graph
State modelled as XML.
X generates POJOs from XML at build time

Developer just focuses on this
- Execute business logic (Java)
- Update state (POJO state tree)
- Send outbound messages

Application business logic as message handlers
Business logic in Java
Updates POJO state and sends outbound messages as POJOs

X Runtime takes care of all non-functional aspects of app
Transactional messaging aka exactly once delivery
Messaging connection management, subscription managment, send/receive
Storage (Cluster memory + disk)
Partitioning/Load Balancing

Support for X and third party messaging buses

An X Application Decomposed

Messaging
Annotation based handler discovery

Messaging
Generated from XML

```
@EventHandler
final public void onAuthRequest(AuthRequestMessage message
                                Repository state) {
    // instantiate a new cc transaction
    final Transaction txn = Transaction.create();

    // extract from message into a transaction
    AuthRequestMessageExtractor.extract(message, txn);

    // update transaction state
    txn.setState(TransactionState.PendingAuth);

    Customer customer = state.getCustomers().get(txn.getCustomerId()
    customer.getTransactions().add(txn)

    // create a fraud detection request
    final FraudDetectionRequest req = FraudDetectionRequest.create();

    // populate the request
    FraudDetectionRequestPopulator.populate(req, txn);

    // send the event
    sendMessage(req);
}
```

State Management
POJO passed in with message handler

State Management
POJO/XML based application state

Object pooling and pre-allocation facilities for zero garbage

Messaging
Extractor and populators to efficiently "copy" data between domain and messages

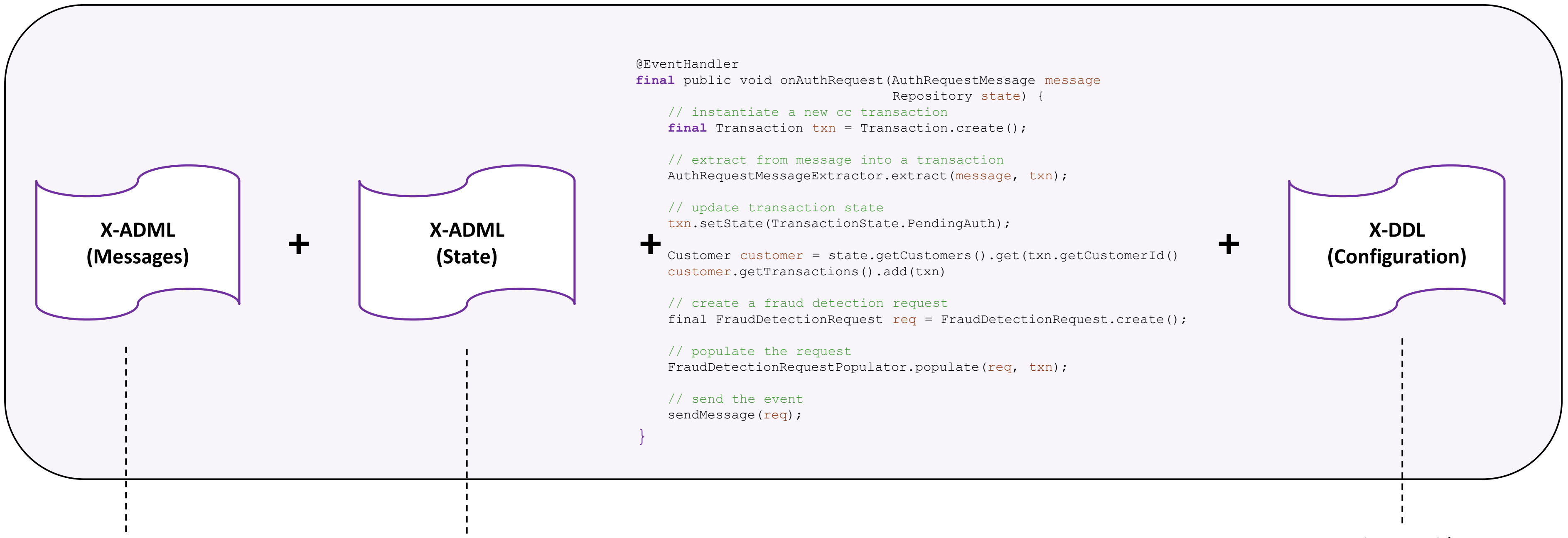
State Management
Clustered Java map/collections

Messaging
Create, populate, send…

X Development in a Nutshell

X Application

=



```
@EventHandler
final public void onAuthRequest(AuthRequestMessage message
                                Repository state) {
    // instantiate a new cc transaction
    final Transaction txn = Transaction.create();

    // extract from message into a transaction
    AuthRequestMessageExtractor.extract(message, txn);

    // update transaction state
    txn.setState(TransactionState.PendingAuth);

    Customer customer = state.getCustomers().get(txn.getCustomerId()
    customer.getTransactions().add(txn)

    // create a fraud detection request
    final FraudDetectionRequest req = FraudDetectionRequest.create();

    // populate the request
    FraudDetectionRequestPopulator.populate(req, txn);

    // send the event
    sendMessage(req);
}
```

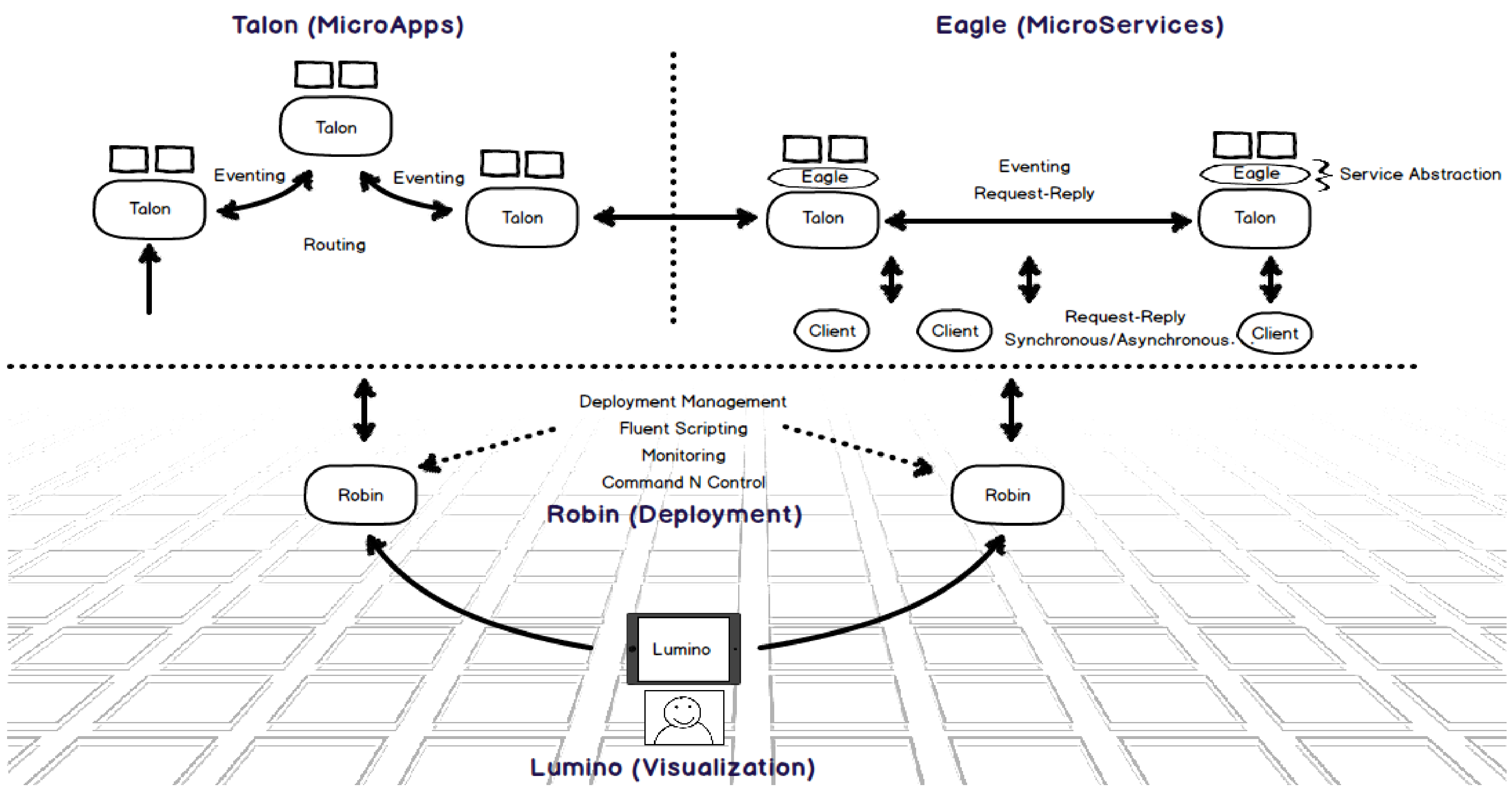
Not required for vertical specific models such as FIX

Not required for Event Sourcing

Messaging Providers +
Wiring Together Apps +
HA Options +
Tuning Knobs

</>kode41 NEEVE RESEARCH

The X Platform



A Credit Card Processing Pipeline built using X Platform

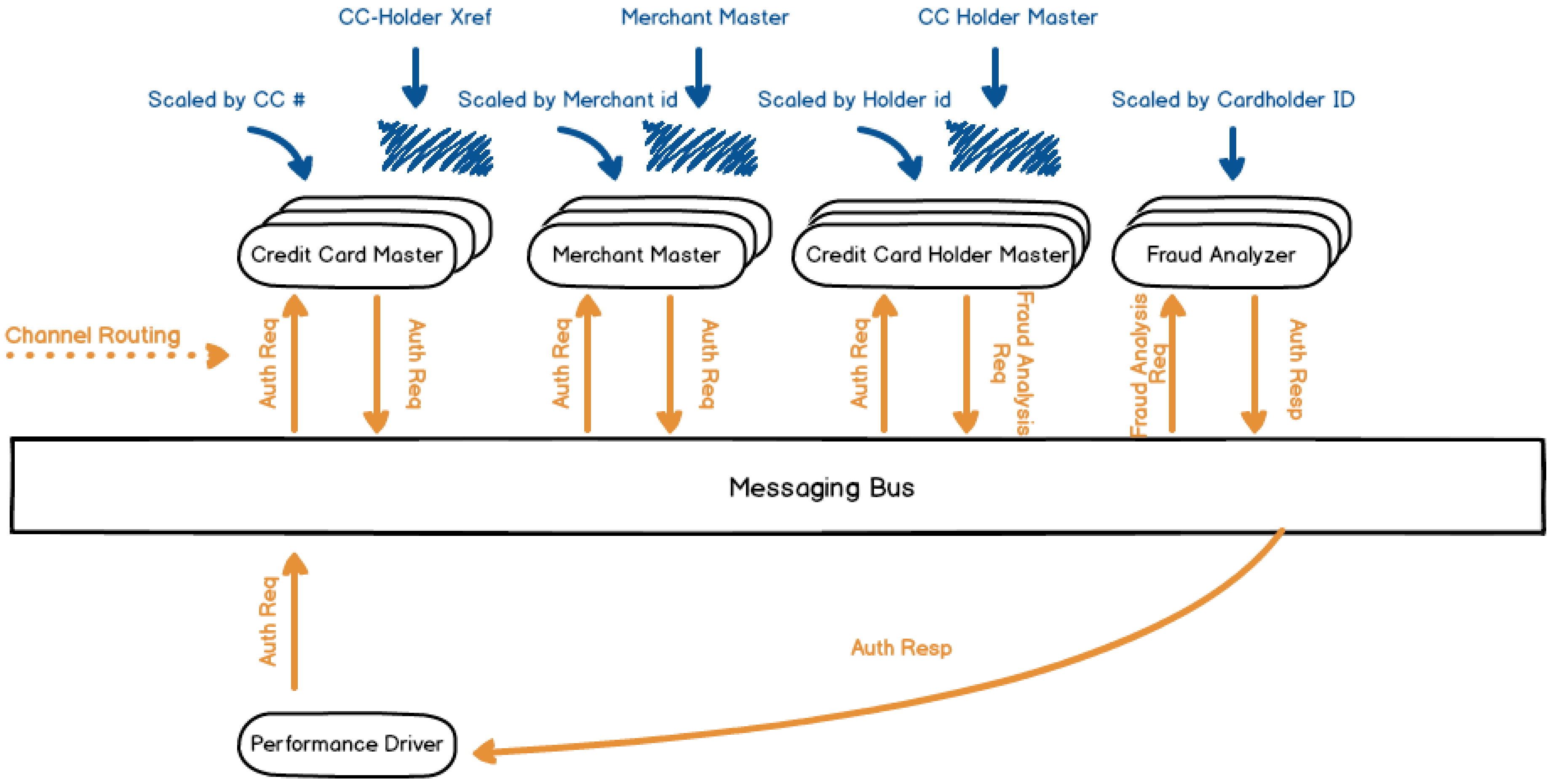
Demo Objectives

- Show
  - How to organize complex payment processing/Fraud Detection system with microservices
  - Microservices design with rich data model
  - How to scale for storage capacity and computation power
  - Handle failures without loss of service

Functionality

- Receive CC Authorization Request
  - Identify Card Holder
  - Identify Merchant
  - Perform Fraud Checks using
    - CC Holder Specific Information
    - Transaction History
- Send CC Authorization Response

Flow



</>kode41 NEEVE RESEARCH

DEMO

Let’s see it in action.

Performance

200k Merchants

40k Card Holders

80k Cards

2 partitions per agent

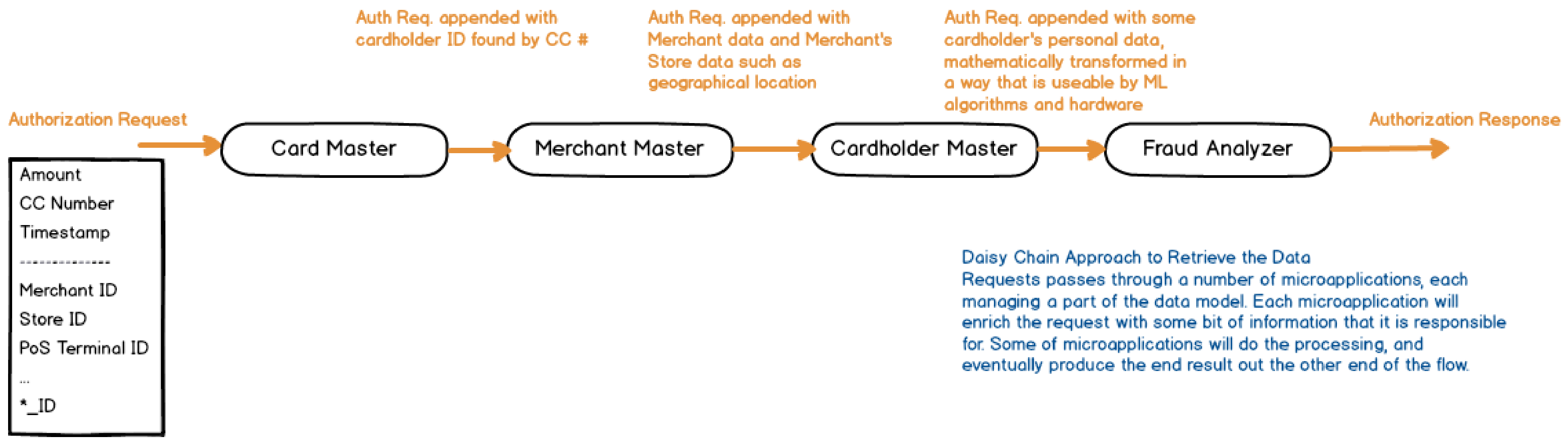
All agents running on just 2 servers

7,500 auth/sec, Full HA + X-Once

Auth Response Time = <5ms

</> kode41 NEEVE RESEARCH

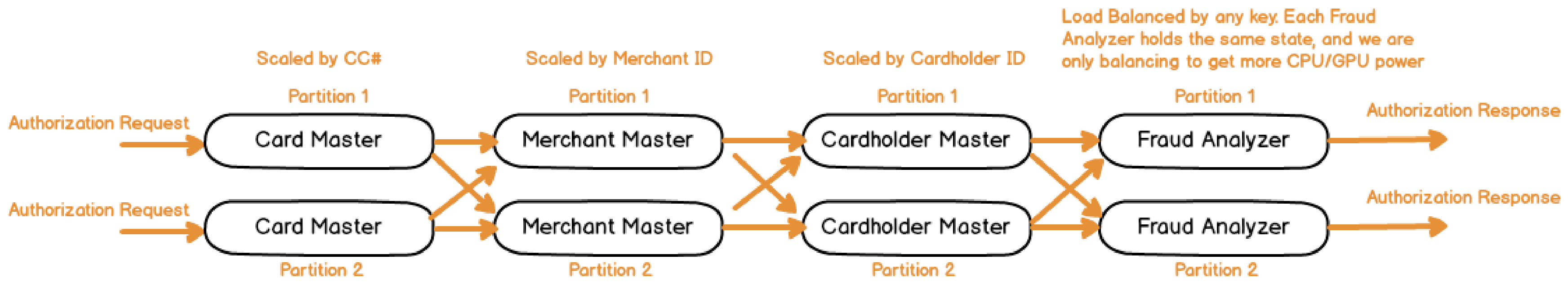
Daisy Chain Message Flow


Authorization Request
Amount
CC Number
Timestamp

Merchant ID
Store ID
PoS Terminal ID
...
*_ID
Card Master
Auth Req. appended with cardholder ID found by CC #
Merchant Master
Auth Req. appended with Merchant data and Merchant's Store data such as geographical location
Cardholder Master
Auth Req. appended with some cardholder's personal data, mathematically transformed in a way that is useable by ML algorithms and hardware
Fraud Analyzer
Authorization Response

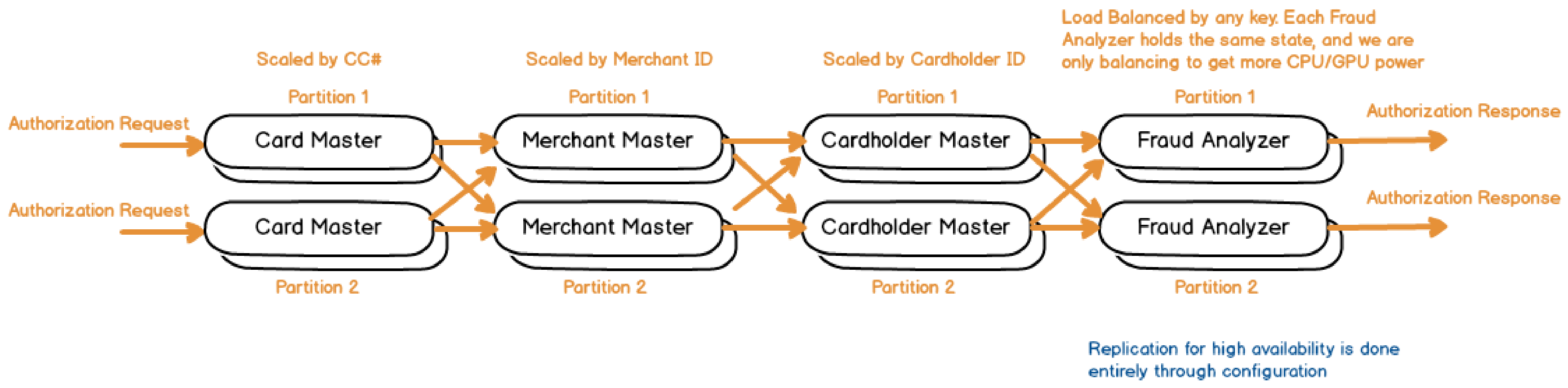
Daisy Chain Approach to Retrieve the Data
Requests passes through a number of microapplications, each managing a part of the data model. Each microapplication will enrich the request with some bit of information that it is responsible for. Some of microapplications will do the processing, and eventually produce the end result out the other end of the flow.

Scaling the system



Scaling is done by configuring routing on message channels, and using channel keys to subscribe apps to only a portion of the traffic. In our example each microapplication has two partitions.

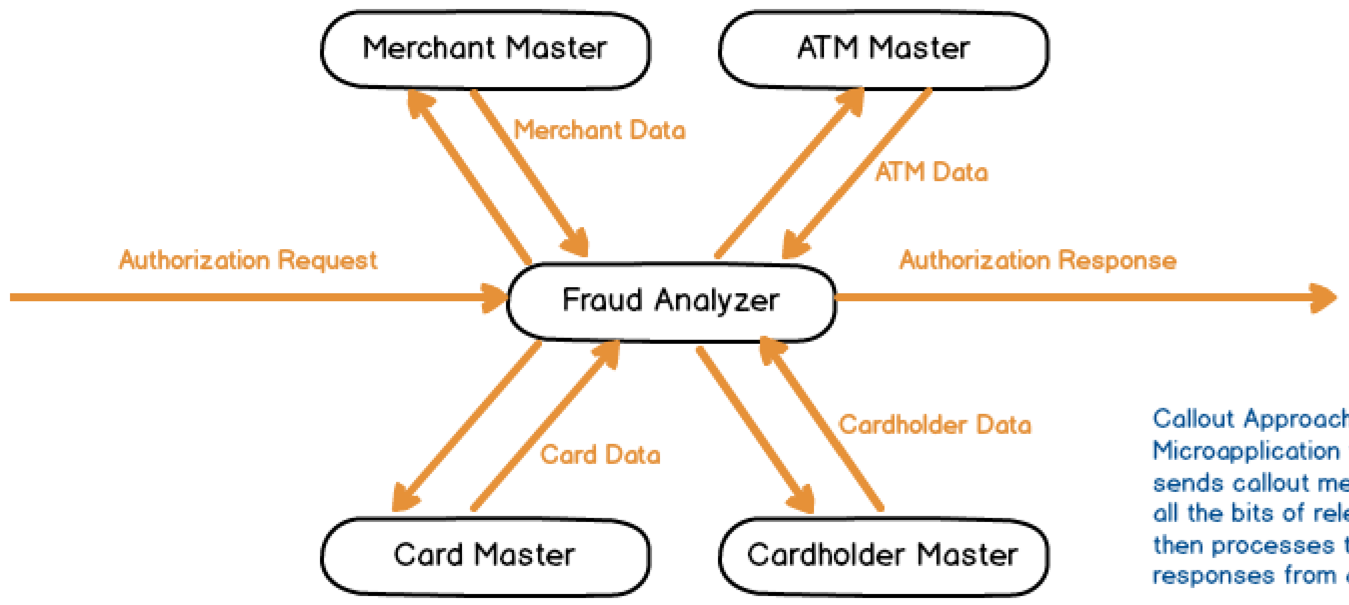
High Availability



CODE

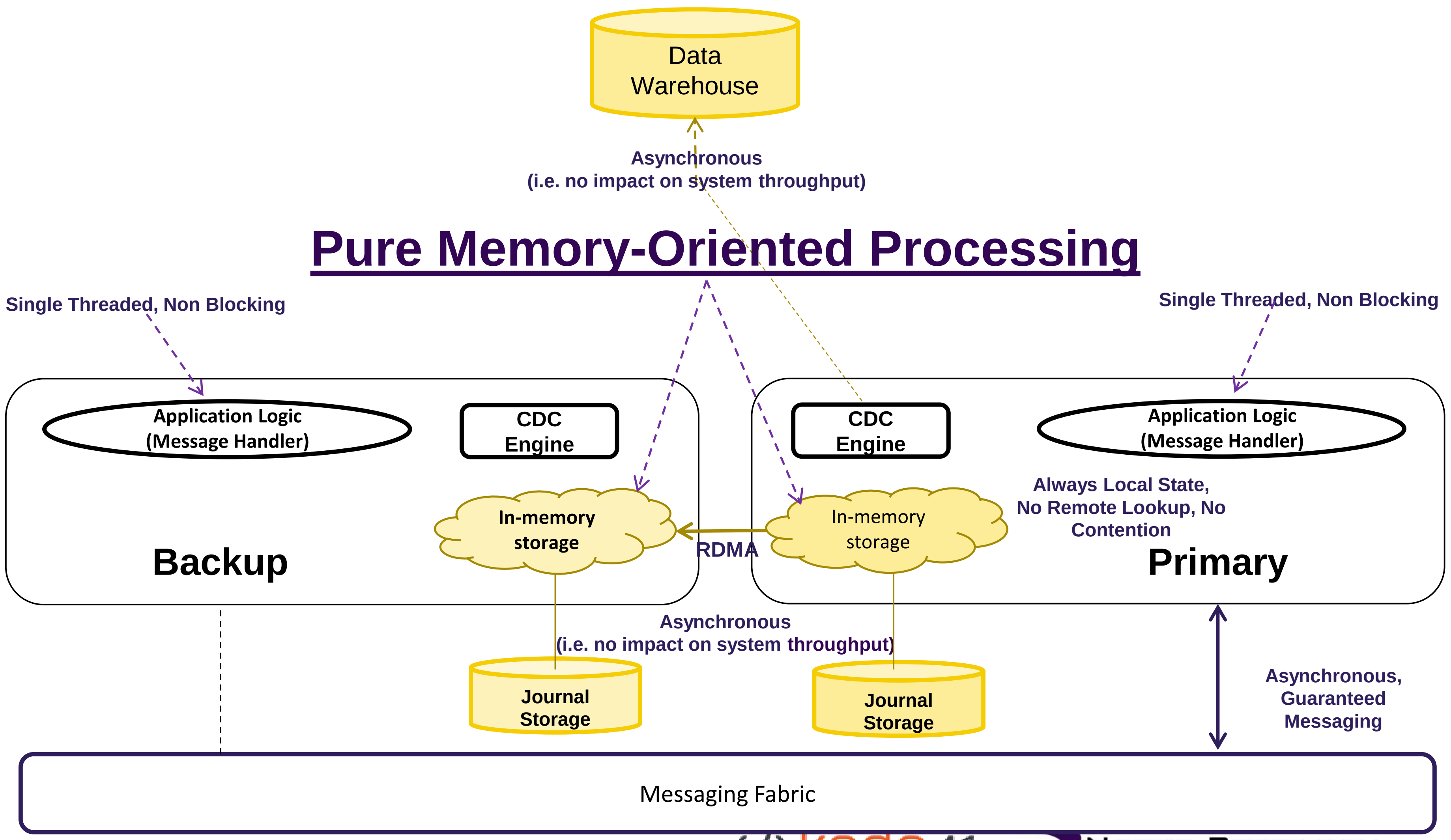
REVIEW.

Future Improvements: Adapting to Data Model Complexity



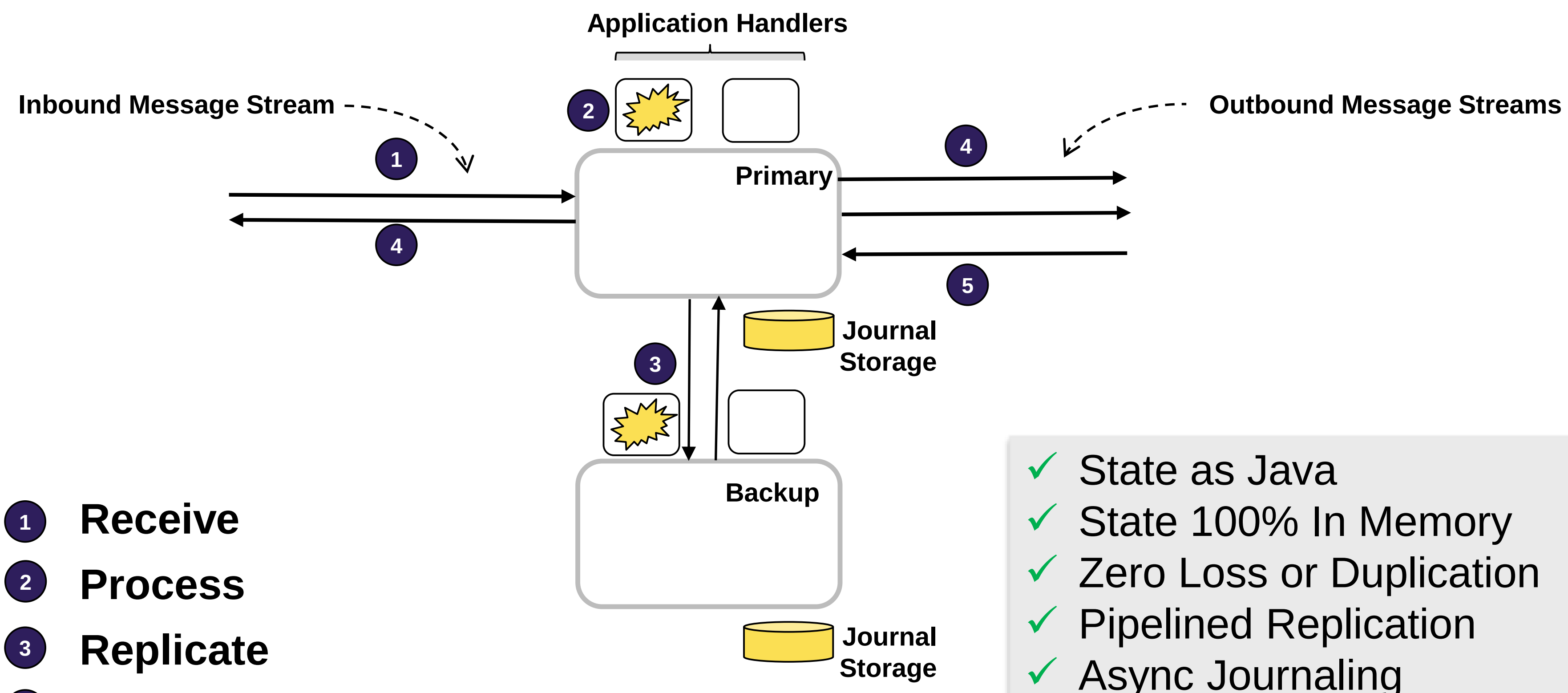
Callout Approach to Retrieve the Data
Microapplication that does the processing sends callout message to other appsto get all the bits of relevant information. Caller then processes the request upon receiving responses from all the microapps.

Reliability


Data Warehouse
Asynchronous
(i.e. no impact on system throughput)
Pure Memory-Oriented Processing
Single Threaded, Non Blocking
Single Threaded, Non Blocking
Application Logic (Message Handler)
CDC Engine
CDC Engine
Application Logic (Message Handler)
Always Local State, No Remote Lookup, No Contention
In-memory storage
In-memory storage
RDMA
Backup
Primary
Asynchronous
(i.e. no impact on system throughput)
Journal Storage
Journal Storage
Asynchronous, Guaranteed Messaging
Messaging Fabric

</> kode41 NEEVE RESEARCH

X Platform High Availabilty

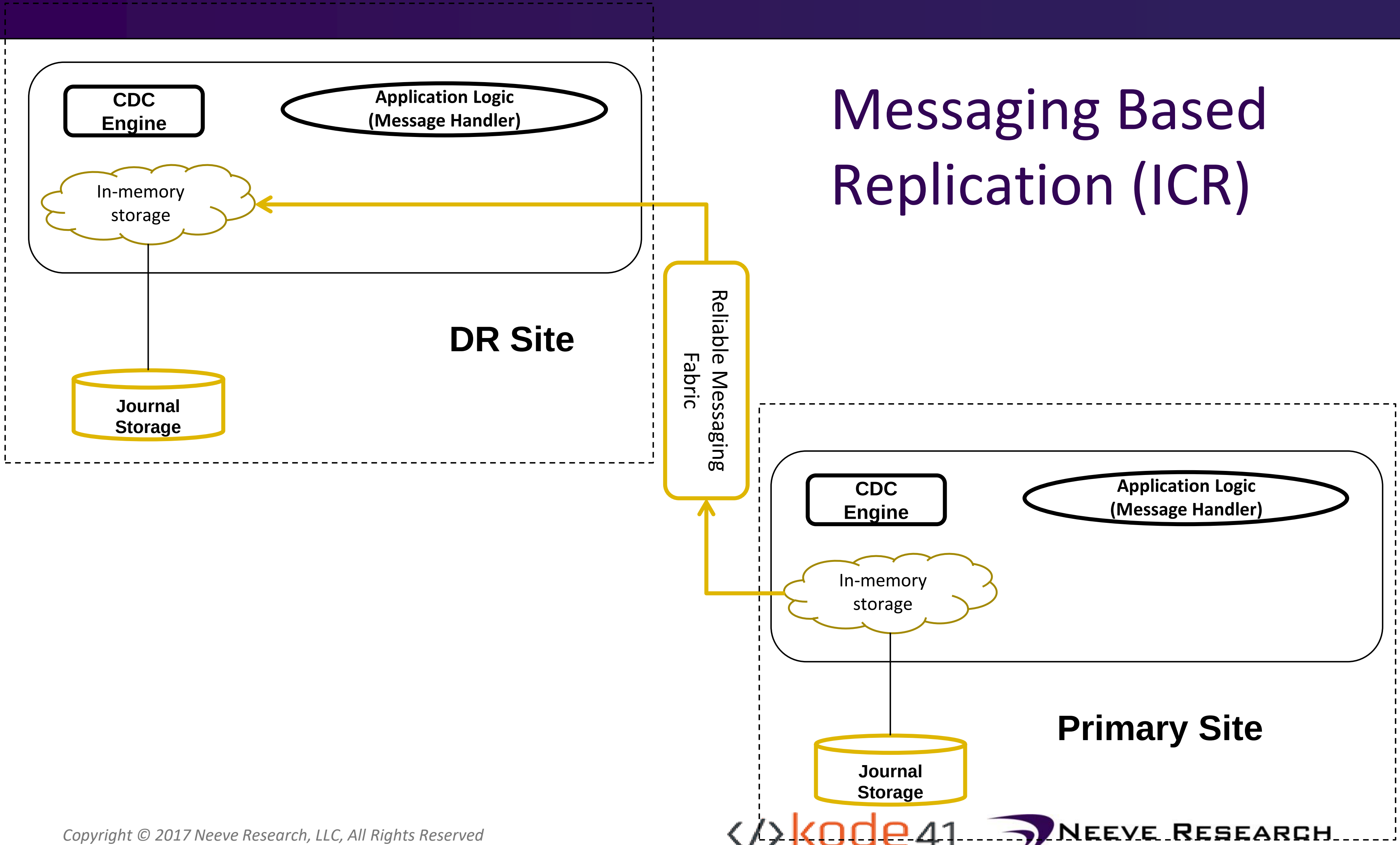


1 **Receive**
2 **Process**
3 **Replicate**
4 **Send Out / Ack**
5 **Inbound Acks**

- ✓ State as Java
- ✓ State 100% In Memory
- ✓ Zero Loss or Duplication
- ✓ Pipelined Replication
- ✓ Async Journaling
- ✓ Messages as Java
- ✓ Pipelined Messaging
- ✓ Pooling for Zero Garbage

Disaster Recovery

Messaging Based Replication (ICR)



Why X for HTAP?

- **Easy to Build**
  - Focus on domain
    - Pure Java
- **Easy to Maintain**
  - Pristine domain
    - No infrastructure bleed
- **Easy to Support**
  - Stock hardware
  - Small Footprint
  - Simple abstractions
  - Easy tools
- **Very, very fast**

✓ **No Compromise**

Agility, Availability, Scalability, Performance

Low Barrier to Entry

- Easy to get started
  - Easy to spin a new app from archetypes or sample apps.
  - Annotate methods on types of interests
  - Wire together applications via configuration.
- Easy to integrate
  - Easily offload transactionally consistent, asynchronous state to data warehousing in the back office via CDC.
  - Built in support for a variety of messaging fabrics.
- Rich and easy to use monitoring tools

Getting Started with X Platform™

Getting Started Guide

https://docs.neeveresearch.com

Get the Demo Source

https://github.com/neeveresearch/nvx-apps

(will be posted to GitHub soon!)

Questions