Amazon Order Tracking

Agenda

- Who is this guy?
- Problem statement
- How we solved it
- Lessons

Who am I?

- Raymond Smith (rasm@amazon.com)
- Software Developer 20+ years; 3 years a manager
- Work on Amazon’s Grocery Business
- We are looking to hire 2022 graduates for 2023 start

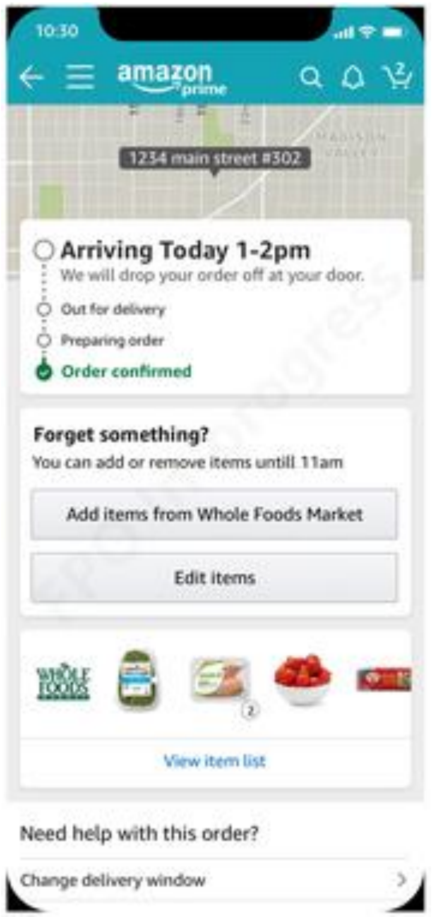
Context: Amazon Grocery

- Amazon supports sales by its own Grocery Brands, and by third-parties using Amazon.com
- Available in: US, UK, France, Germany, Italy, Spain, Singapore, India, and Japan. More being added all the time.
- Grocery has a large development organization (more than 500 people). Even more developers in greater Amazon (tens of thousands).
- There is no way to coordinate that many developers – so we distribute the work both at a people level (Two Pizza Teams) and a Technical Level (loosely coupled, message based systems).

Customer Problem: Where are my Groceries?

- At Amazon we always work-backwards from Customer.
- Grocery Customers have some unique needs:
  - They are relying on their order to feed themselves and their families.
  - If an order cannot be delivered, or items are missing, they likely cannot wait and will have to arrange an alternative.
- Amazon's existing notifications helped but were not great:
  - They didn't provide enough detail on where the order was at
  - They were not interactive
- Our goal was to provide a better customer experience for Amazon Customers

Problem: Display Status of Customer’s Order


10:30

amazon prime

1234 main street #302

Arriving Today 1-2pm

We will drop your order off at your door.

Out for delivery

Preparing order

Order confirmed

Forget something?

You can add or remove items untill 11am

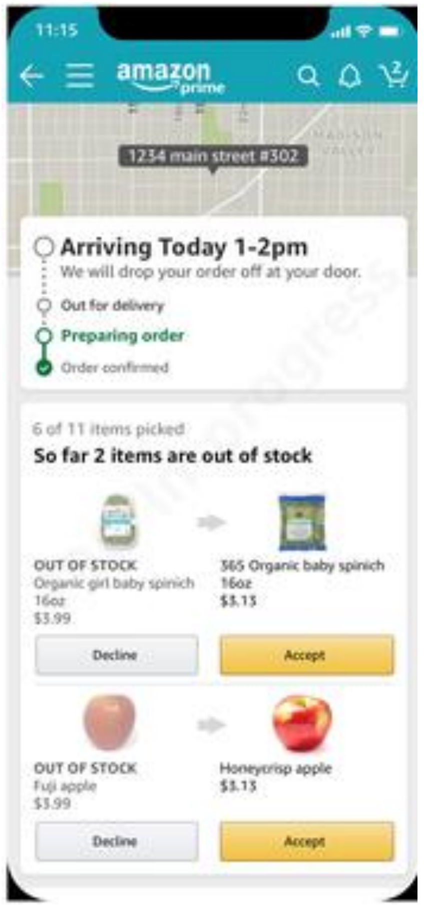
Add items from Whole Foods Market

Edit items

View item list

Need help with this order?

Change delivery window


11:15

amazon prime

1234 main street #302

Arriving Today 1-2pm

We will drop your order off at your door.

Out for delivery

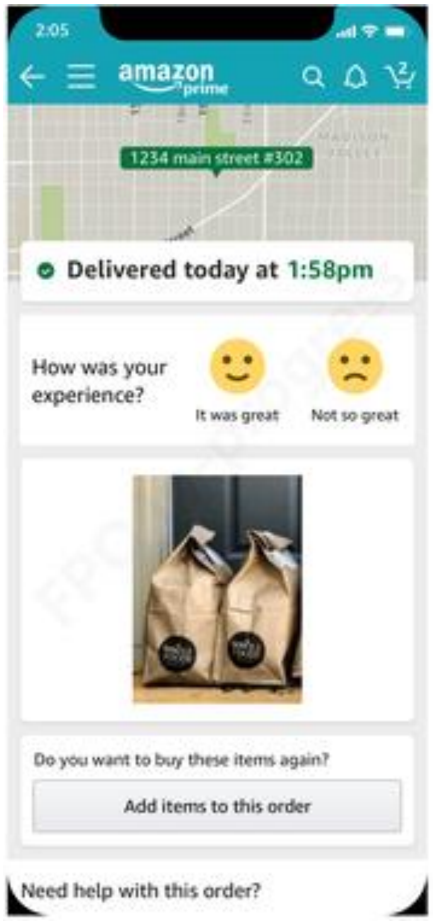
Preparing order

Order confirmed

6 of 11 items picked

So far 2 items are out of stock

OUT OF STOCK Organic girl baby spinich 16oz $3.99	365 Organic baby spinich 16oz $3.13
Decline	Accept
OUT OF STOCK Fuji apple $3.99	Honeycrisp apple $3.13
Decline	Accept


2:05

amazon prime

1234 main street #302

Delivered today at 1:58pm

How was your experience?

It was great

Not so great

Do you want to buy these items again?

Add items to this order

Need help with this order?

Complications

Complications:

- 20+ services involved in fulfilling an order
- No one service owns the live state of the order
- Variations between lines of business (when events are sent)
- We don’t know exactly all of the things we want to track
- Existing Order Page is on older technology

Things in our favour:

- Loose coupling and two pizza teams let us move fast
- Rich events available with reasonably well defined formats
- Some strong common concepts: Customer ID, Order IDs, etc

Core Technical Problem

Core problem: form a unified view by integrating events from multiple systems

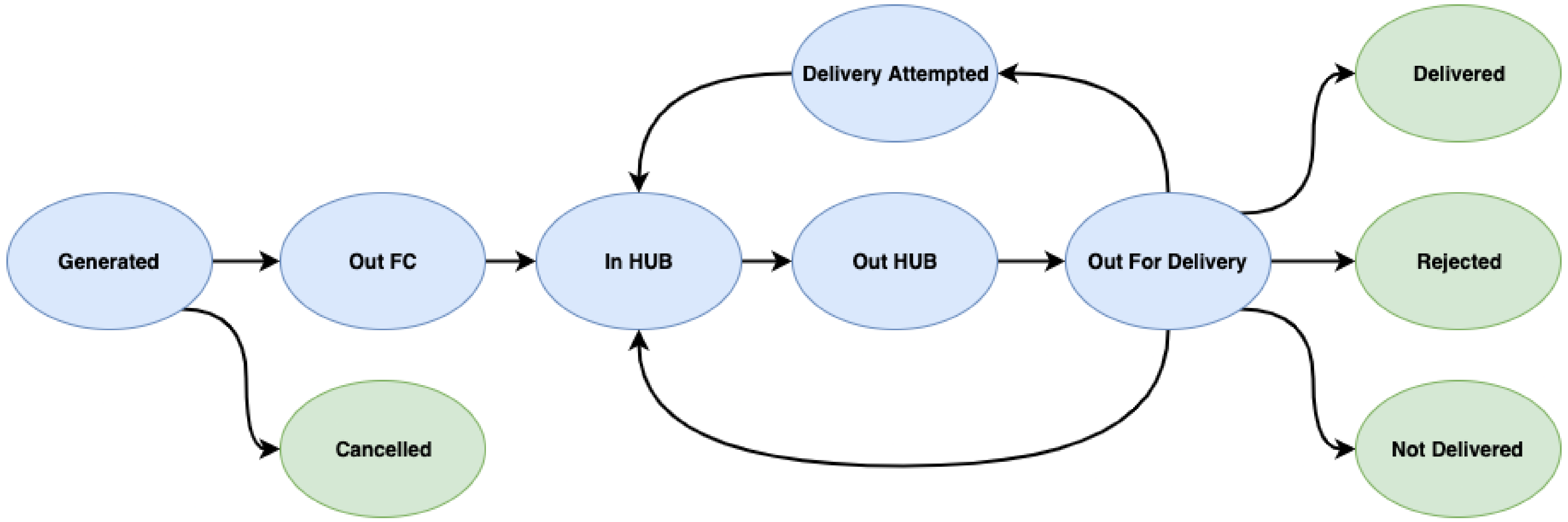
Subproblems:

- Out of order events
- Handling missing data
- Source systems constantly evolving: new events, milestones, etc
- Keeping track of state

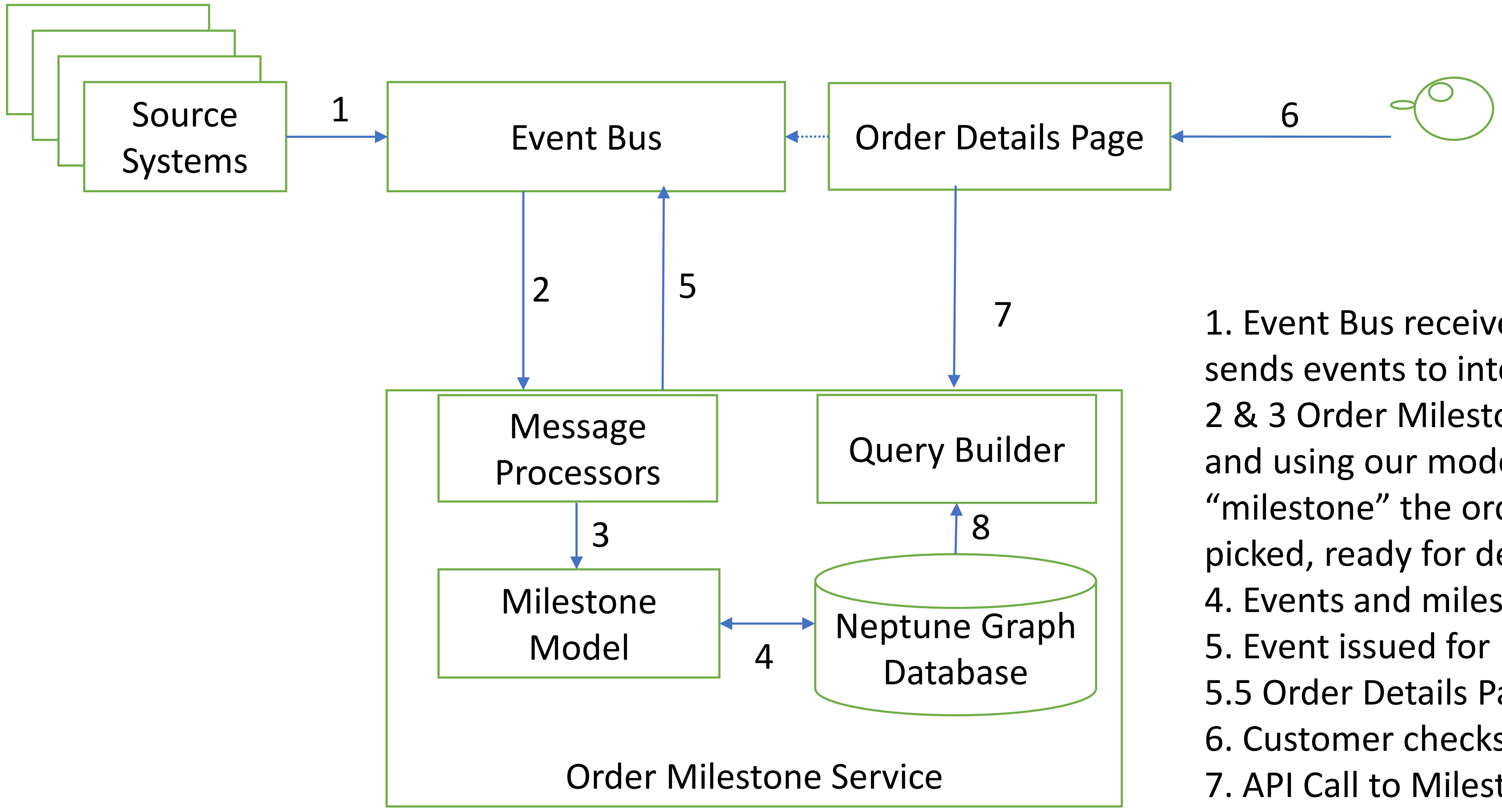
We already had an event bus platform for the unified view, but we did not have anything which kept track of state.

High Level Approach

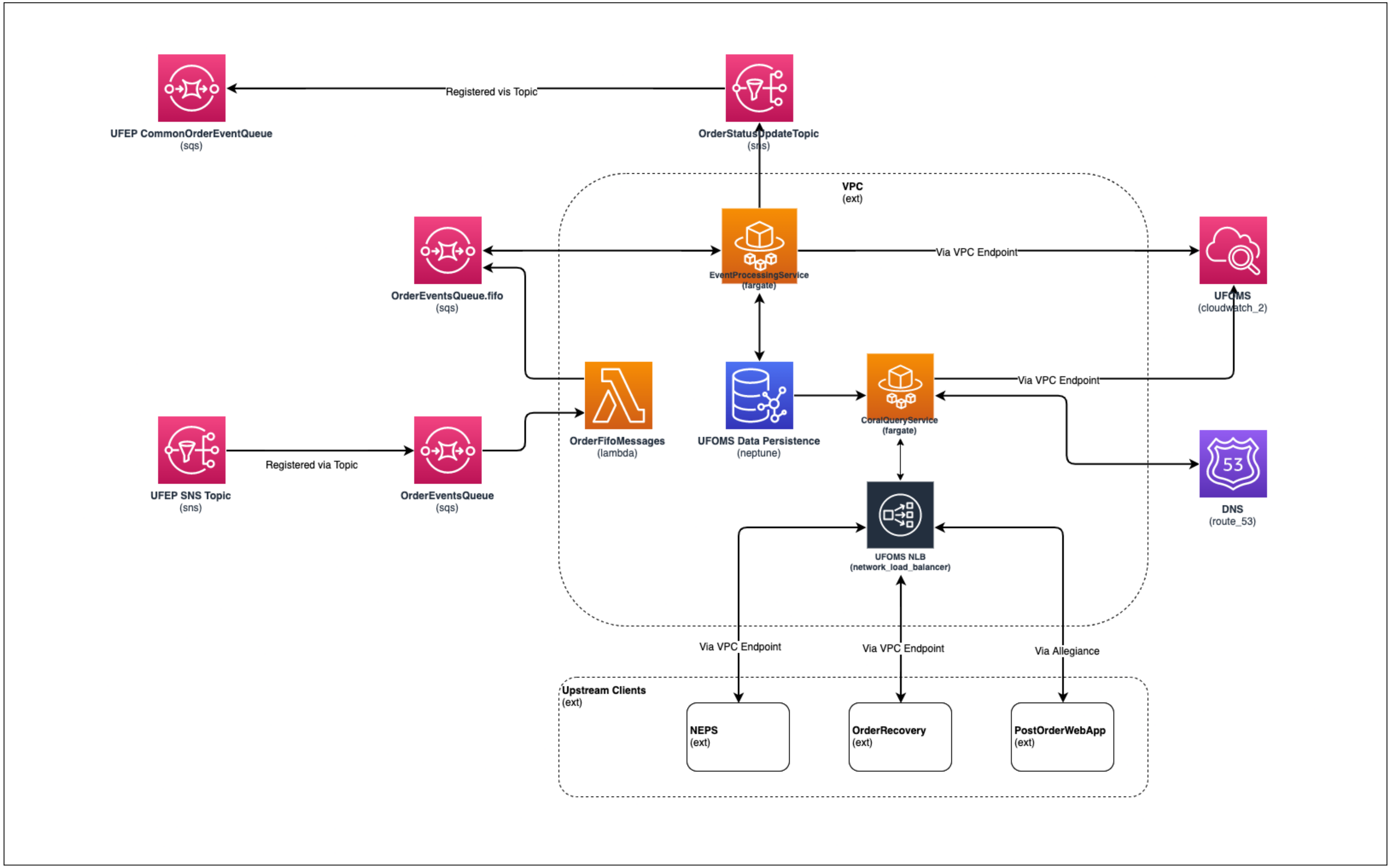
- Define a common model for the key states of an order
- Gather all the events into a common form
- Rules to trigger when different states detected
- UI to listen for state triggers for display


Delivery Attempted
Delivered
Generated
Out FC
In HUB
Out HUB
Out For Delivery
Rejected
Cancelled
Not Delivered

(Simplified) Architecture Diagram



1. Event Bus receives events, normalizes, and sends events to interested systems.
2 & 3 Order Milestone Service receives events, and using our model determines which "milestone" the order has reached (created, picked, ready for delivery, etc)
4. Events and milestones stored in database
5. Event issued for new milestone
5.5 Order Details Page listens for milestone events
6. Customer checks on order status
7. API Call to Milestone Service
8. Queries for state in Neptune, data returned


UFEP CommonOrderEventQueue
(sqs)
Registered vis Topic
OrderStatusUpdateTopic
(sns)
VPC
(ext)
OrderEventsQueue.fifo
(sqs)
EventProcessingService
(fargate)
Via VPC Endpoint
UFOMS
(cloudwatch_2)
OrderFifoMessages
(lambda)
UFOMS Data Persistence
(neptune)
CoralQueryService
(fargate)
Via VPC Endpoint
UFEP SNS Topic
(sns)
Registered via Topic
OrderEventsQueue
(sqs)
DNS
(route_53)
UFOMS NLB
(network_load_balancer)
Via VPC Endpoint
Via VPC Endpoint
Via Allegiance
Upstream Clients
(ext)
NEPS
(ext)
OrderRecovery
(ext)
PostOrderWebApp
(ext)

For Reference: Technologies we used

- AWS:
  - Compute: Lambda (serverless compute), Fargate (container compute)
  - Persistence: Neptune
  - Networking: ALB/ELB, Route 53, and VPC
  - Messaging: SNS and SQS
- Apache Tinker Pop & Gremlin
- Java

Processing Events

- Distributed Systems 101: Networks are not reliable, event ordering cannot be guaranteed, the same event may be received multiple times. (This is the first Fallacies of Distributed Computing.)
- Many ways to handle this
- Our approach:
  - Idempotent processing: the same event should result in the same change to the database. Achieved by using business keys (ie order id, etc).
  - Infer missing events: if we receive a “cancel order event” we can assume there was a “create order event” and fill it with placeholder information until we receive the “create event”.

Simplified (Faked) Example

- When Events are in order:

```
// order created event
{ Id: ”Order A”, Ordered: [ { ASIN: “B06WW2YGSJ”, QTY: 2 },
{ASIN: “B00AXYMRYA”, QTY: 1} ], Status: ”Created” }
// warehouse starts work event
{Id: ”Order A”, Ordered: [{ ASIN: “B06WW2YGSJ”, QTY: 2 },
{ASIN: “B00AXYMRYA”, QTY: 1}], Status: ”Processing” } ],
Status: ”Processing” }
// no oranges event
{”Order A”, Ordered: [{ ASIN: “B06WW2YGSJ”, QTY: 2 }, {ASIN:
“B00AXYMRYA”, QTY: 1}], Shorted: [{ASIN: “B00AXYMRYA”, QTY:
1}], Status: ”Processing” }
// order is out for delivery
{ Id: “Order A”, {”Order A”, Ordered: [{ ASIN: “B06WW2YGSJ”,
QTY: 2 }, {ASIN: “B00AXYMRYA”, QTY: 1}], Shorted: [{ASIN:
“B00AXYMRYA”, QTY: 1}], Status: ”Out For Delivery” }
```

Simplified (Faked) Example

- When Events are in *out* order (out for delivery comes first):

```
// order is out for delivery
{ Id: “Order A”, Ordered: PLACEHOLDER, Status: ”Out For
Delivery” }
// order created event
{ Id: ”Order A”, Ordered: [ { ASIN: “B06WW2YGSJ”, QTY: 2 },
{ASIN: “B00AXYMRYA”, QTY: 1} ], Status: ”Out For Delivery” }
// warehouse starts work event
{Id: ”Order A”, Ordered: [{ ASIN: “B06WW2YGSJ”, QTY: 2 },
{ASIN: “B00AXYMRYA”, QTY: 1}], Status: ”Processing” } ],
Status: ”Out For Delivery” }
// no oranges event
{Id: ”Order A”, Ordered: [{ ASIN: “B06WW2YGSJ”, QTY: 2 },
{ASIN: “B00AXYMRYA”, QTY: 1}], Shorted: [{ASIN:
“B00AXYMRYA”, QTY: 1}], Status: ”Out For Delivery” }
```

Processing Events: a changing world

- Problem: the source systems and business processes are highly changeable.
- We did not want to have to rework our designs whenever there was a change. So we went looking for a flexible data-store.

Neptune Graph Database

- Graph Database was a natural fit for a “graph of events”
- We could use Gremlin queries to determine milestones by matching sequences of events.
- Very flexible schema allowed us to add events at will.

Neptune Controversy

- Introducing Neptune proved very controversial.
- Most Brisbane developers preferred using DynamoDB, and had gotten very good at using it for diverse purposes.
- Neptune has some serious limitations compared to DynamoDB: expensive, requires downtime for maintenance (up to 8 hours / year), requires developers to tune type of CPU and memory used. DynamoDB is cheap, requires no downtime, and only requires no complex tuning.

So, why stick with Neptune?

- A graph database was a natural fit for the data we were processing.
- Innovating requires us to move outside our comfort zones.
- How has Neptune turned out?
    - AWS has been making it better and better
    - The more flexible design has paid off as it allowed us to incorporate new very easily (around 1 – 2 day, verus 2 -- 3 weeks for similar changes elsewhere).

Aside: Radical Innovation

- Software Development is a creative activity. It is still more a craft than an engineering discipline. People become stuck in “local maximas” of technology they know well.
- Whenever you are making a radical break you need to:
  - double check that you are innovating for your customer – ensure there is genuine benefit
  - show empathy to your colleagues, actively listen to their concerns
  - have a viable plan B if your new technology does not work out
  - define success criteria, and be humble
    - if your plan does not work out, switch to plan b
  - be brave enough to be misunderstood

The Progress Tracker Team

amparry

Adrian
Parry

shawnzx

Shawn
Chang

vsilchan

Uladzimir
Silchanka

jackwc

Jack
Willis-Craig

rasm

Raymond
Smith

ces

Courtney
Sowter

homanl

Homan
Lee

yuechang

Gary
Yin

aipan

Aishwarya
Panchbhai

kamatv

Vignesh
Kamath

anisdh

View Custom Photo
Anis
Dhapa

lbw

View Badge Photo
Craig
Hurst

Key Takeaways

- Always start with your customer, and work backwards
- Don’t be afraid to try new things – but be humble and have a plan B
- Message Based Systems are powerful way to decouple systems and teams
- Distributed Systems 101: the network is not reliable – plan for that
    - Idempotence
    - Inferring state

Discussion

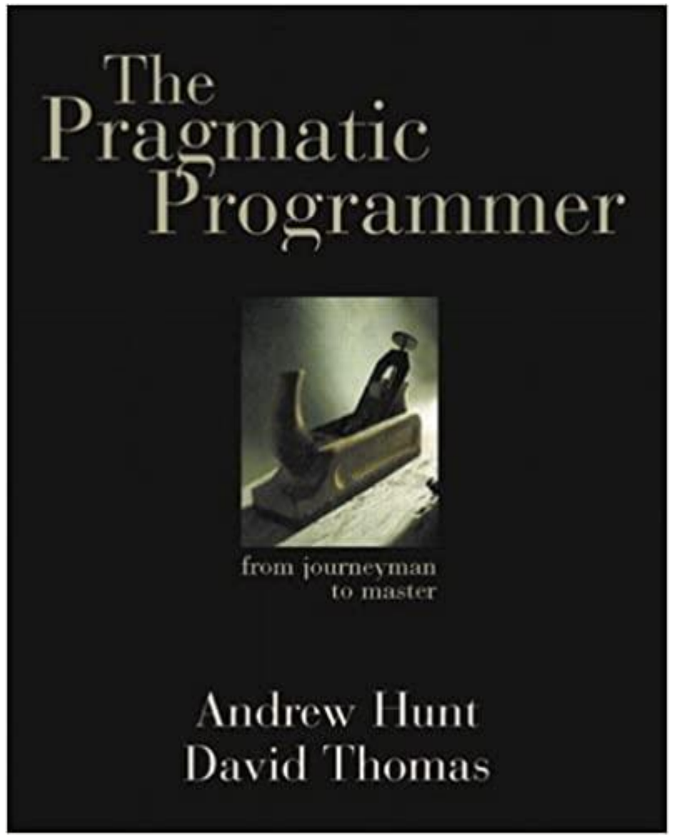
Questions?

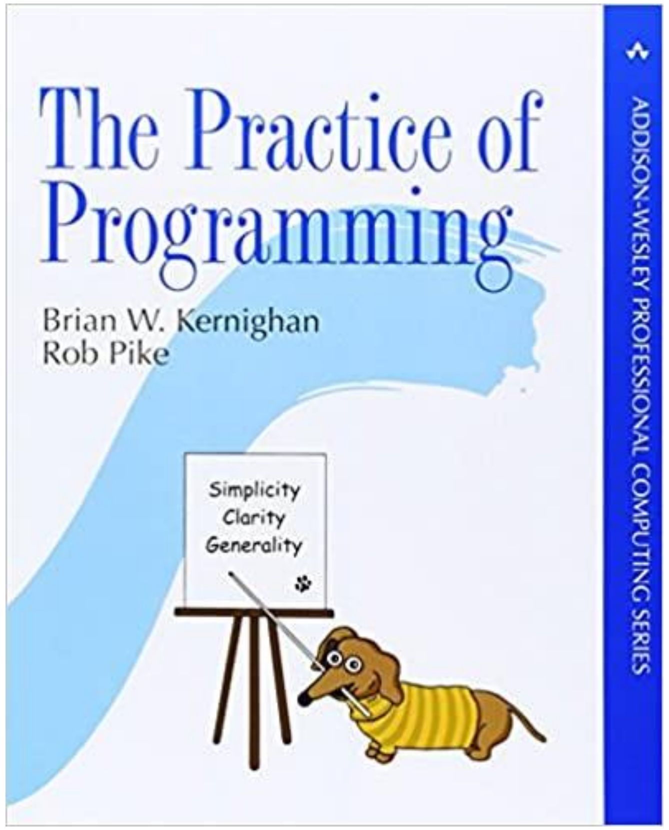
Comments?

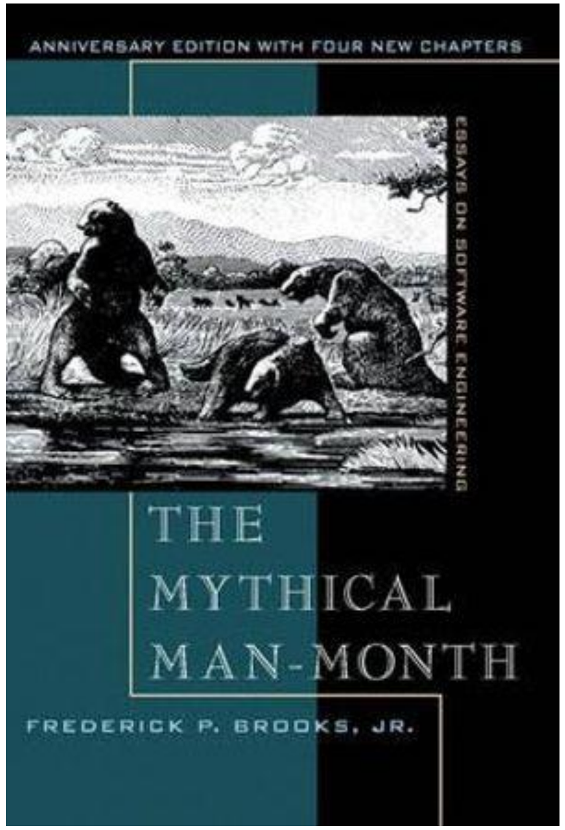
Thoughts?

Bonus Material: Books to Read

These three books teach timeless lessons for programmers.




The Practice of Programming
Brian W. Kernighan
Rob Pike
Simplicity
Clarity
Generality
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.