ADVENTURES IN SCALING FROM **ZERO** TO **5 BILLION** DATA POINTS PER DAY

Dave Torok

Distinguished Architect

Comcast Corporation

2 April, 2019

Flink Forward – San Francisco 2019

COMCAST

COMCAST CUSTOMER RELATIONSHIPS

30.3 MILLION OVERALL CUSTOMER RELATIONSHIPS AT 2018 YEAR END

25.1 MILLION RESIDENTIAL HIGH-SPEED INTERNET CUSTOMERS AT 2018 YEAR END

1.2 MILLION RESIDENTIAL HIGH-SPEED INTERNET CUSTOMER NET ADDITIONS IN 2018

COMCAST

DELIVER THE ULTIMATE CUSTOMER EXPERIENCE

IS THE CUSTOMER HAVING A GOOD EXPERIENCE FOR HIGH SPEED DATA (HSD) SERVICE?

IF THE CUSTOMER ENGAGES US DIGITALLY, CAN WE OFFER A SELF-SERVICE SOLUTION?

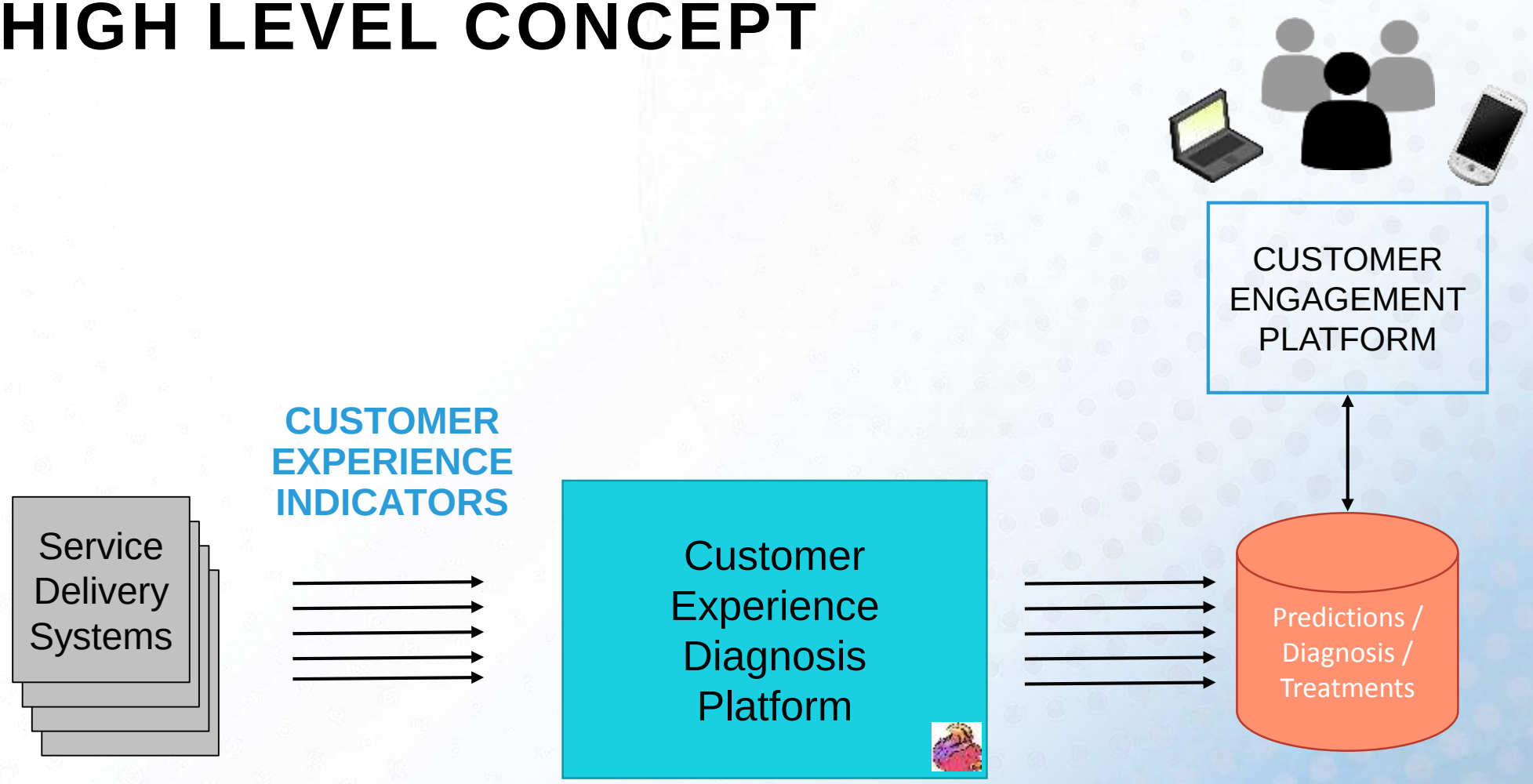
IF THERE IS AN ISSUE CAN WE OFFER OUR AGENTS AND TECHNICIANS A DIAGNOSIS TO HELP SOLVE THE PROBLEM QUICKER?

REDUCE THE TIME TO RESOLVE ISSUES

REDUCE COST TO THE BUSINESS AND THE CUSTOMER

COMCAST

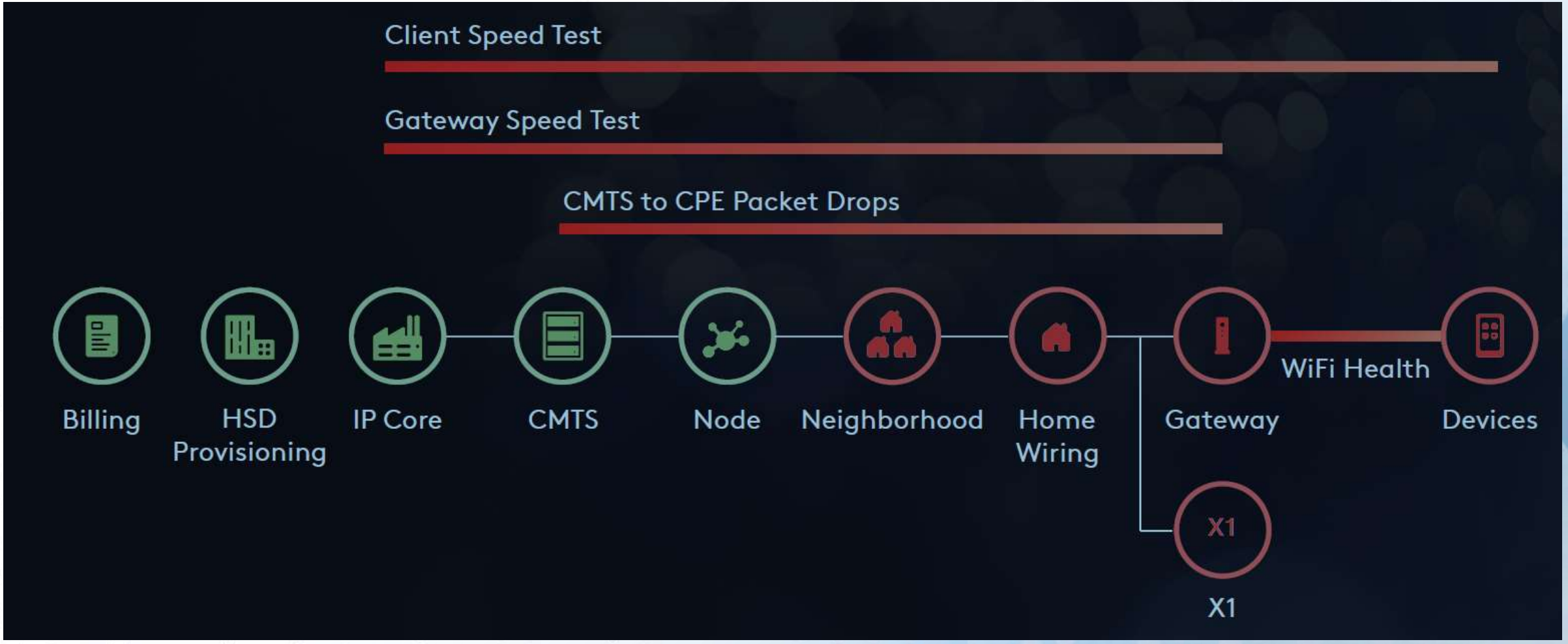
HIGH LEVEL CONCEPT



Comcast collects, stores, and uses all data in accordance with our privacy disclosures to users and applicable laws.

COMCAST

CUSTOMER EXPERIENCE INDICATORS



Comcast collects, stores, and uses all data in accordance with our privacy disclosures to users and applicable laws.

COMCAST

NINE ADVENTURES IN SCALING

THE **TRIGGER AND DIAGNOSIS** PROBLEM

THE **REST** PROBLEM

THE **INEFFICIENT OBJECT HANDLING** PROBLEM

THE **FEATURE STORE** PROBLEM

THE **VOLUME** PROBLEM

THE **CUSTOMER STATE** PROBLEM

THE **CHECKPOINT** PROBLEM

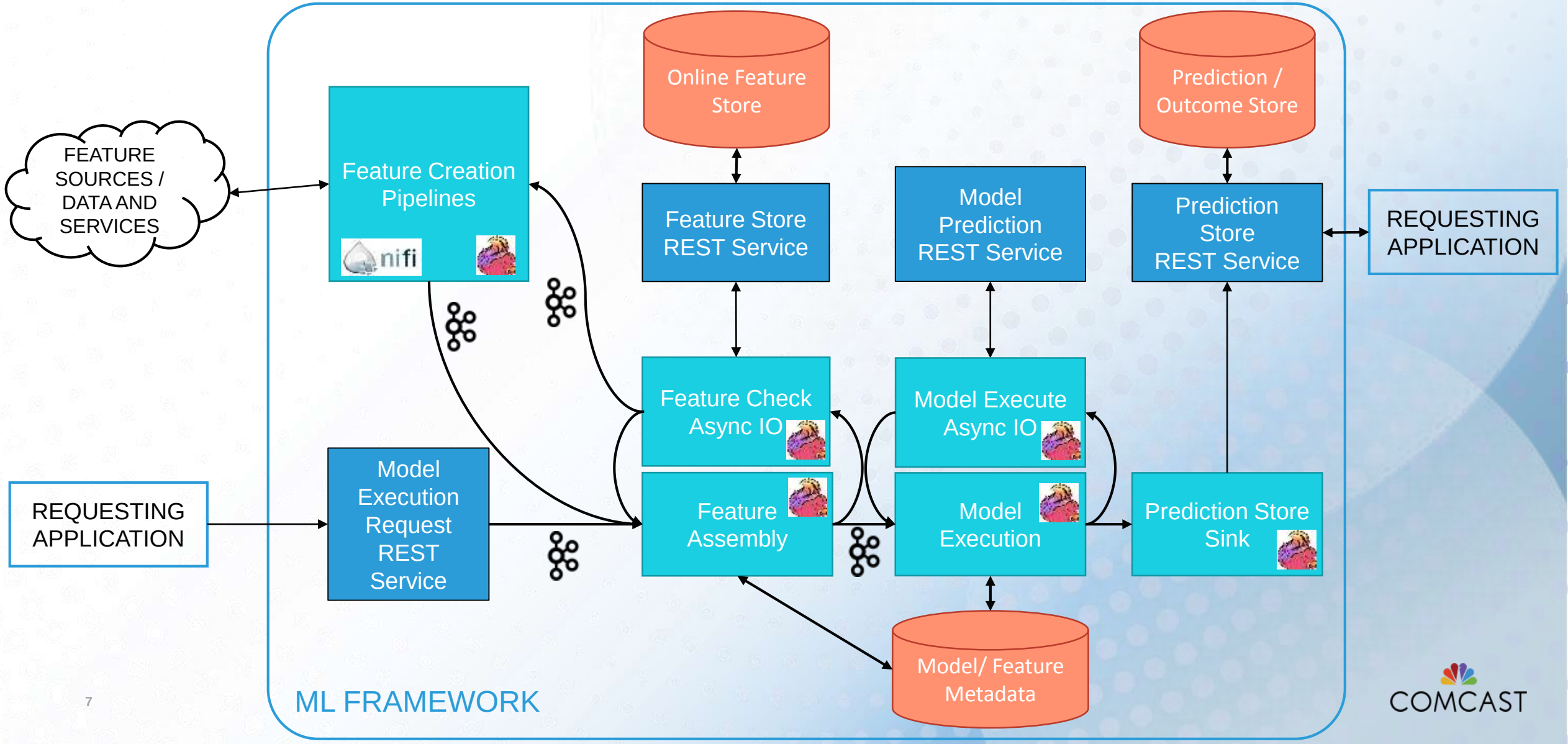
THE **TRIGGER AND DIAGNOSIS** PROBLEM

THE **REALLY HIGH VOLUME AND FEATURE STORE** PROBLEM #2

COMCAST

ML FRAMEWORK ARCHITECTURE – 2018

"EMBEDDING FLINK THROUGHOUT AN OPERATIONALIZED STREAMING ML LIFECYCLE" – SF FLINK FORWARD 2018



COMCAST

THE **TRIGGER AND DIAGNOSIS** PROBLEM

MAY 2018 – INITIAL VOLUMES

INDICATOR #1

9 MILLION

INDICATOR #2

166 MILLION

INDICATOR #3

1.2 MILLION

INDICATOR #4

2500 (SMALL TRIAL)

TOTAL

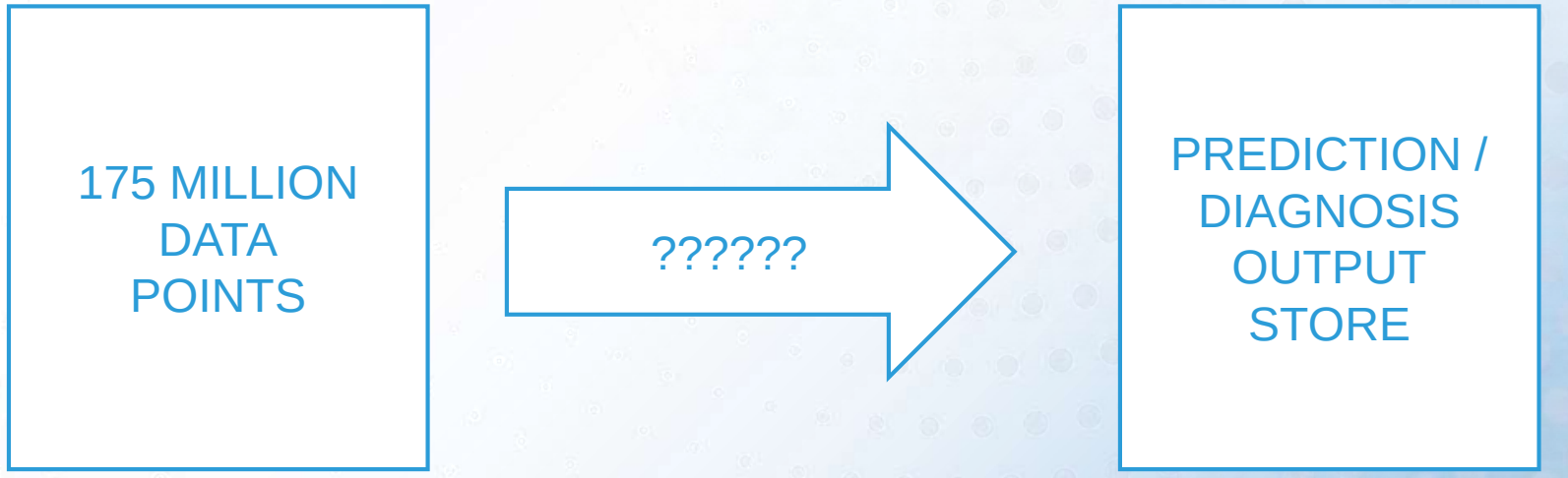
175

MILLION / DAY

Comcast collects, stores, and uses all data in accordance with our privacy disclosures to users and applicable laws.

COMCAST

175 MILLION PREDICTIONS?



COMCAST

INTRODUCE A ‘TRIGGER’ CONDITION

GREEN

GREEN

GREEN

RED

GREEN

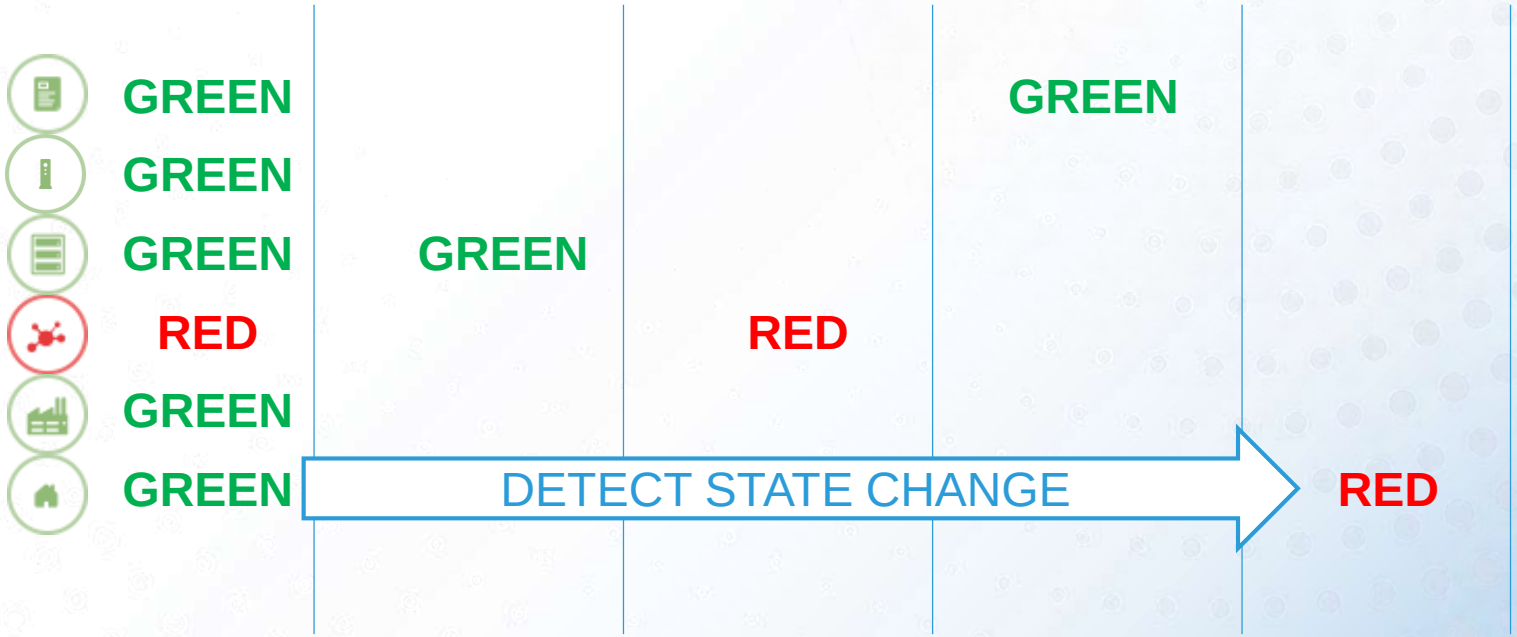
GREEN

COMCAST

INTRODUCE A ‘TRIGGER’ CONDITION

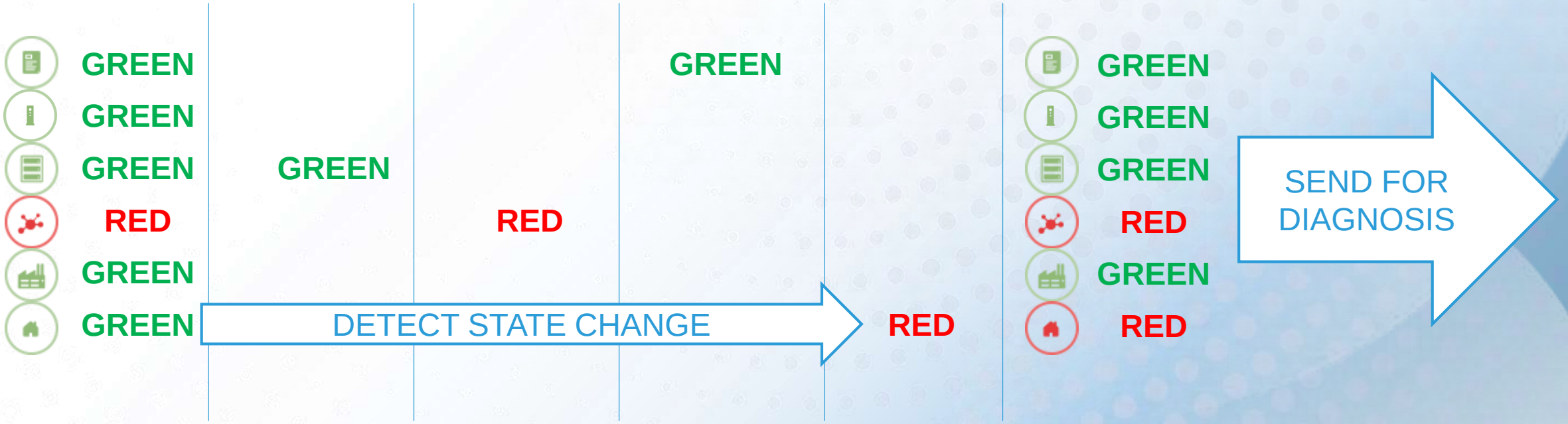
COMCAST

INTRODUCE A ‘TRIGGER’ CONDITION



COMCAST

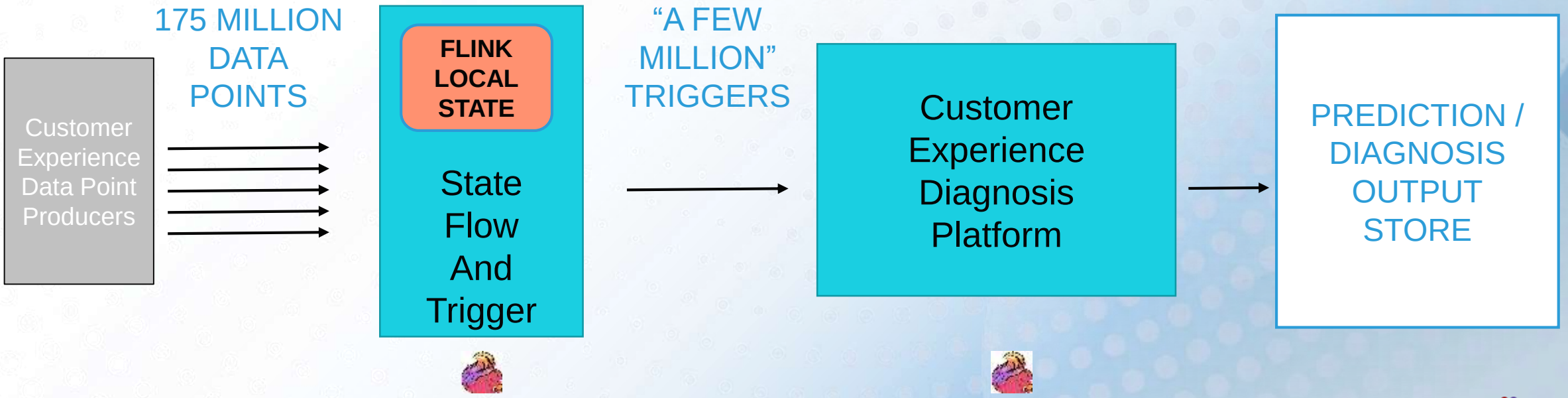
INTRODUCE A ‘TRIGGER’ CONDITION



COMCAST

STATE MANAGER AND TRIGGER

INTRODUCE FLINK LOCAL STATE

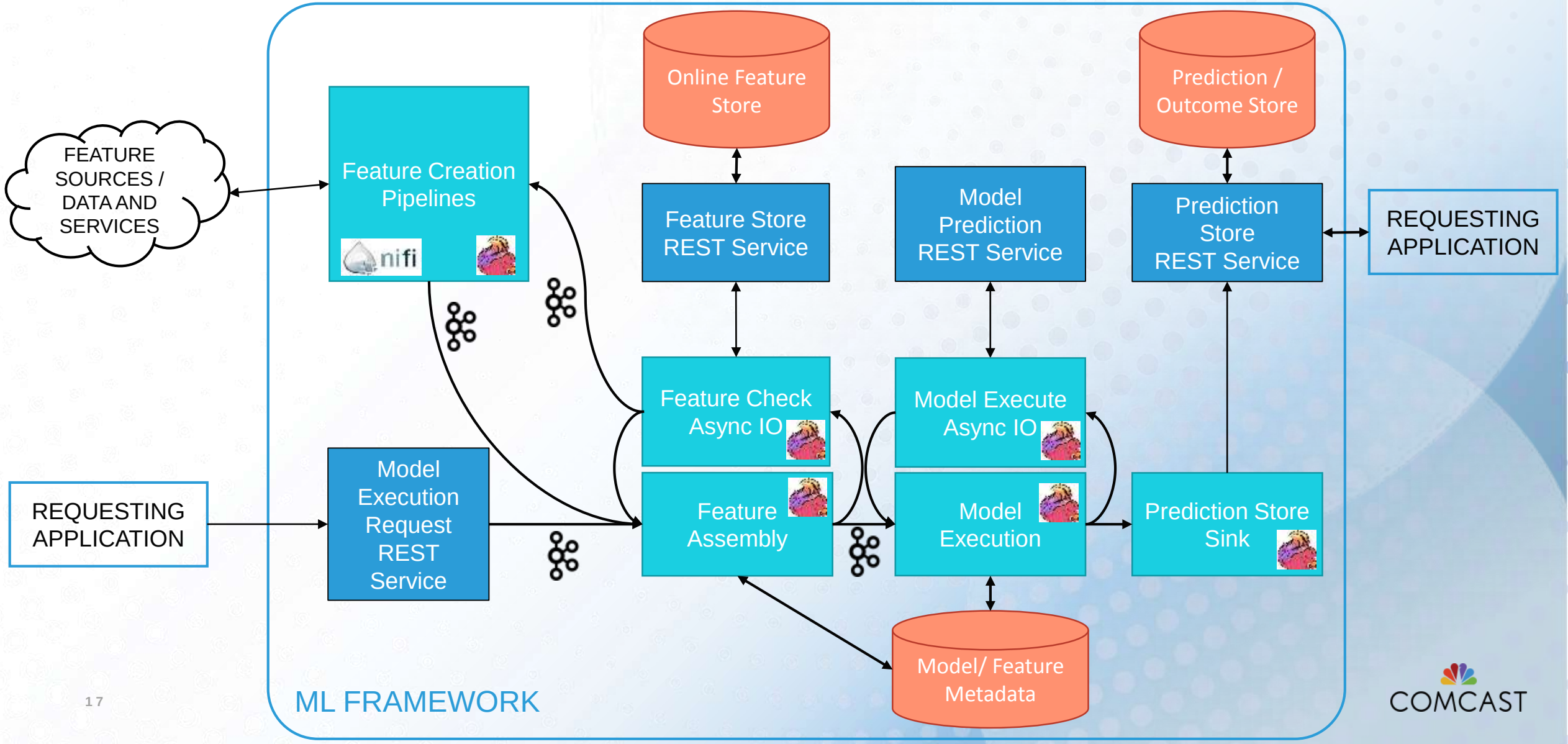


COMCAST

THE **REST** PROBLEM

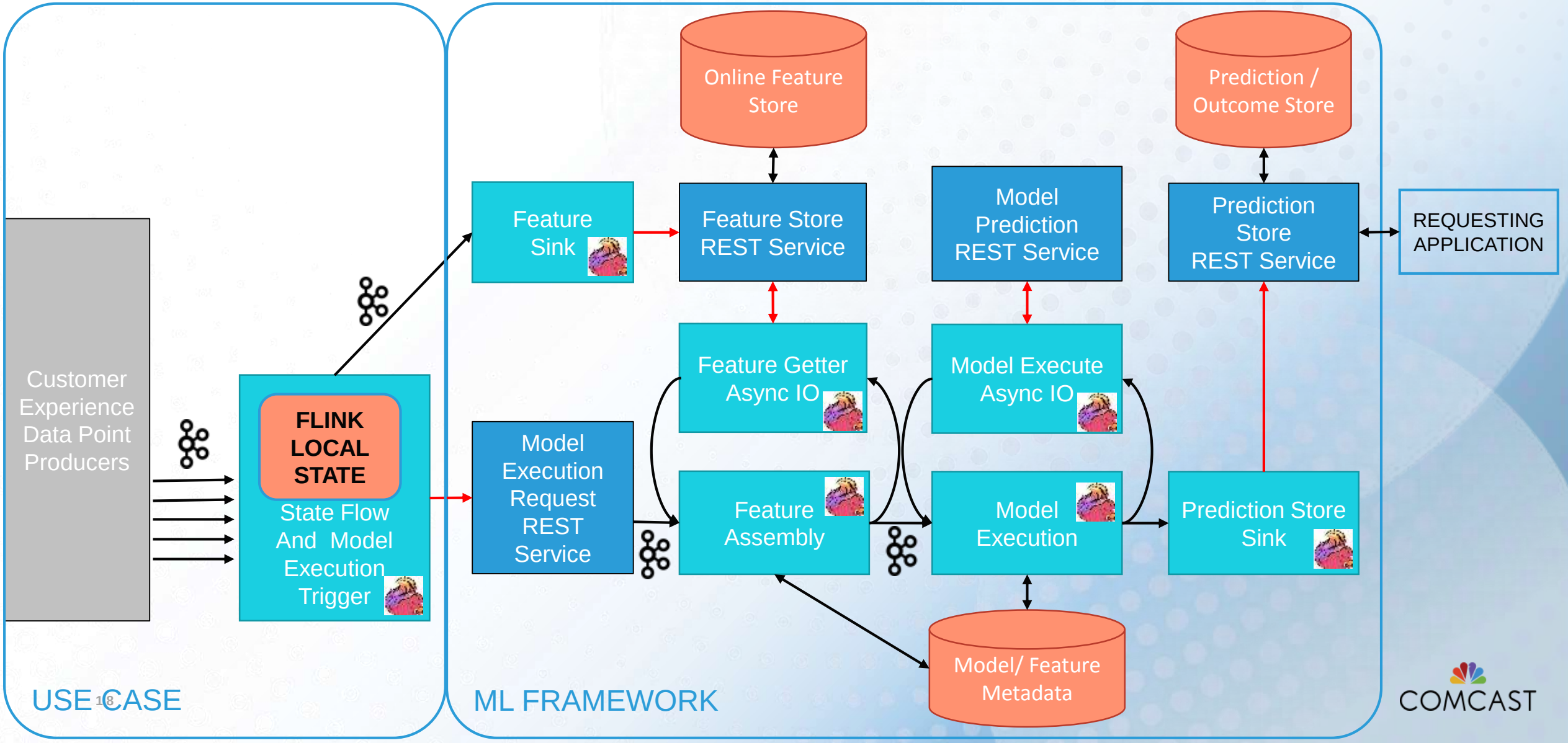
ML FRAMEWORK ARCHITECTURE – 2018

”EMBEDDING FLINK THROUGHOUT AN OPERATIONALIZED STREAMING ML LIFECYCLE” – SF FLINK FORWARD 2018



COMCAST

INITIAL PLATFORM ARCHITECTURE



COMCAST

ORIGINAL ML FRAMEWORK ARCHITECTURE

EVERY COMMUNICATION WAS A REST SERVICE

ML FRAMEWORK ISOLATED FROM DATA POINT USE CASE BY DESIGN

FLINK ASYNC + REST IS DIFFICULT TO SCALE ELASTICALLY

COMCAST

TUNING ASYNC I/O – LATENCY AND VOLUME

MAX THROUGHPUT PER SECOND = (SLOTS) * (THREADS) * (1000 / TASK DURATION MSEC)

(12 SLOTS) * (20 THREADS) * (1000 / 300) = 800 / SECOND

APACHE HTTP CLIENT / EXECUTOR THREADS

setMaxTotal

setDefaultMaxPerRoute (THREADS)

FLINK ASYNC MAX OUTSTANDING

RichAsyncFunction “Capacity”

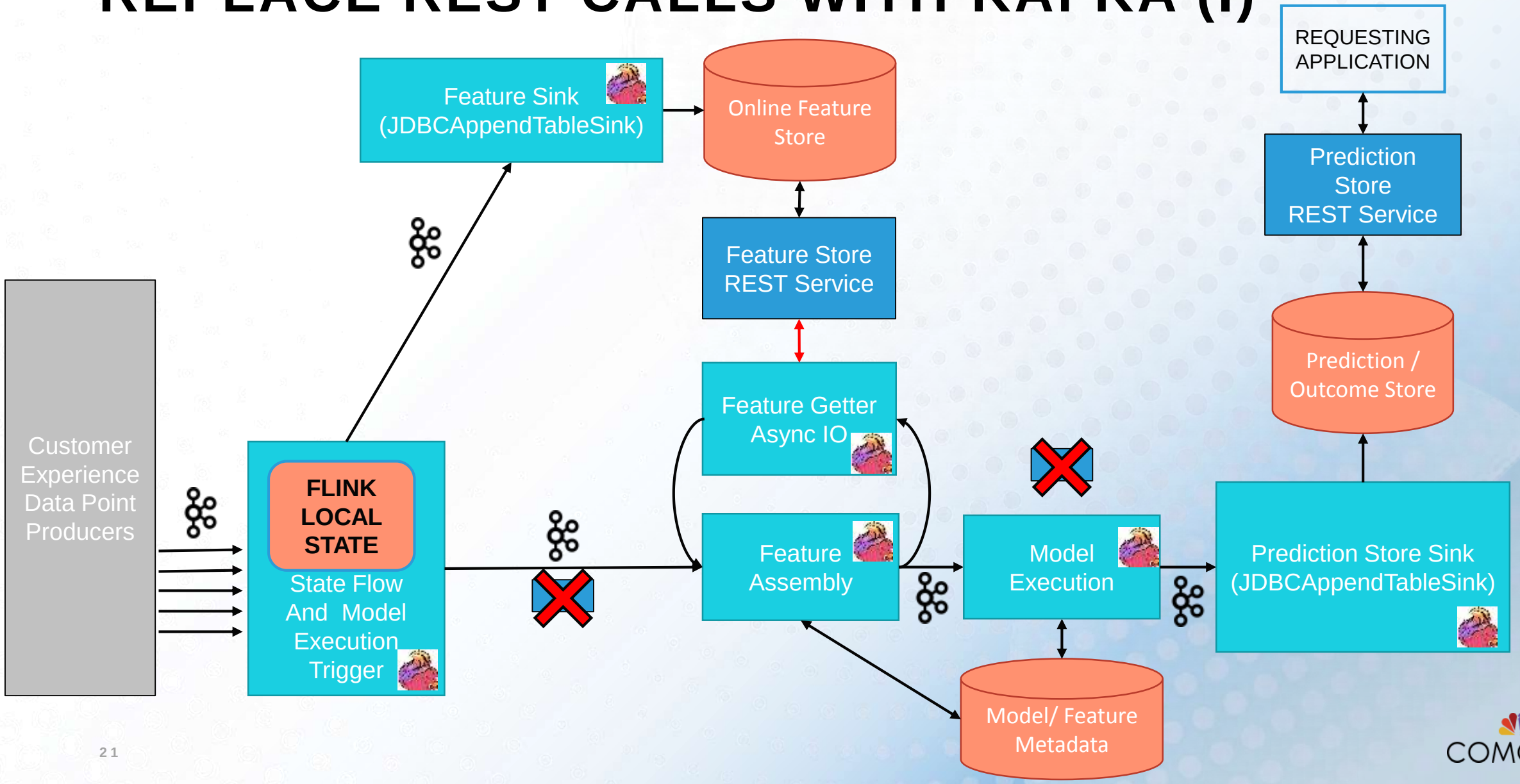
AsyncDataStream.unorderedWait() with max capacity parameter

RATE THROTTLING

```
if (numberOfRequestsPerPeriod > 0) {
int numberOfProcesses = getRuntimeContext()
.getNumberOfParallelSubtasks();
        numberOfRequestsPerPeriodPerProcess =
Math.max(1, numberOfRequestsPerPeriod /
numberOfProcesses);
}
```

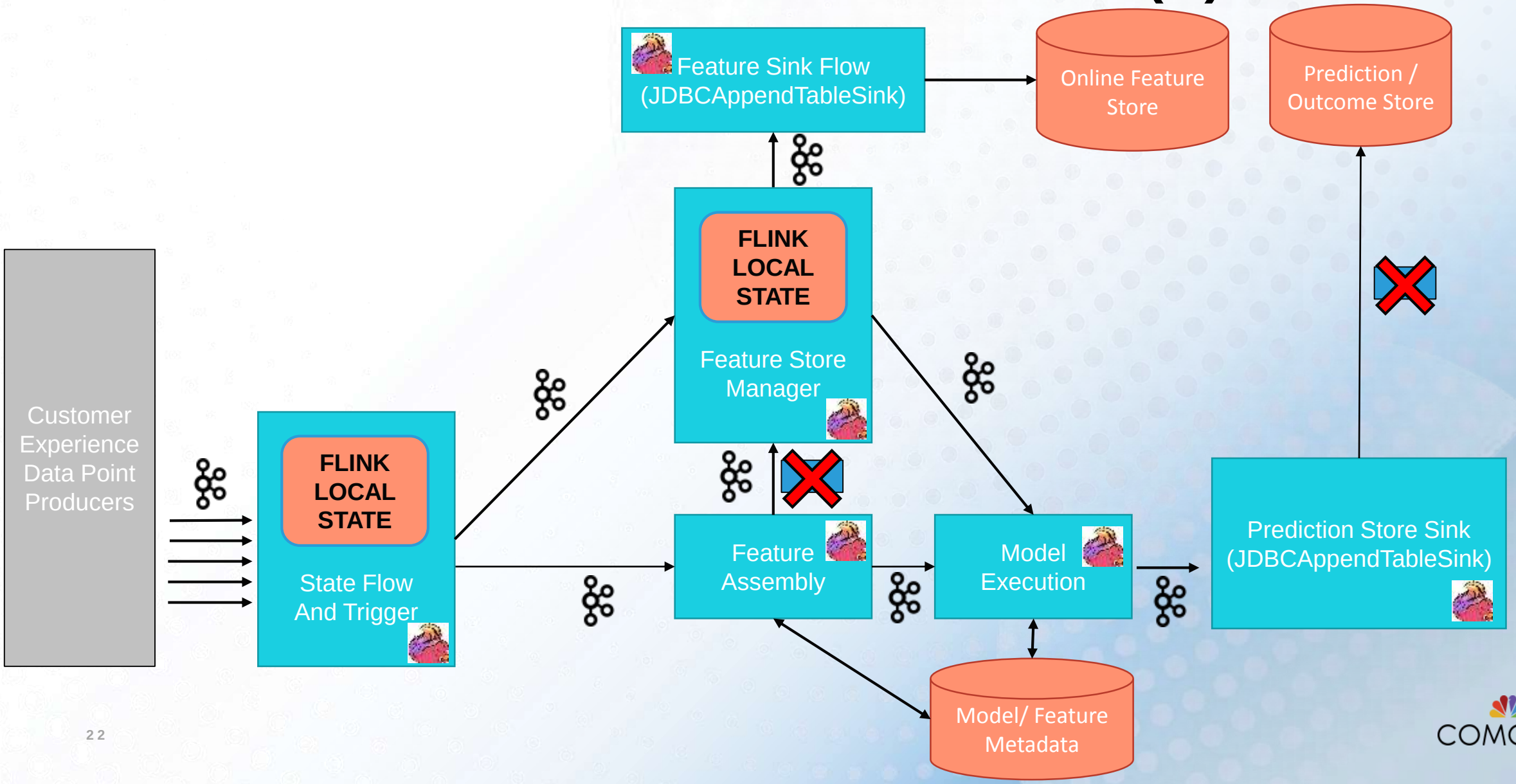

COMCAST

REPLACE REST CALLS WITH KAFKA (I)



COMCAST

REPLACE REST CALL WITH KAFKA (II)



COMCAST

THE **INEFFICIENT OBJECT HANDLING** PROBLEM

HAS THIS EVER HAPPENED TO YOU?

```
String transformedJson =
new JoltJsonTransformer()
        .transformJson(myJsonValue,
        pTool.getRequired(SPEC_PATH));
JSONObject inputPayload = new
JSONObject(transformedJson);
try {
aType = inputPayload.getJSONObject("info")
        .getJSONObject(“aType").
        getString("string");
} catch (Exception e) {
// ignore, invalid object
aType = STATUS_INVALID;
}
```

New “JOLT” JSON Transformer

(but don’t cache it)

→ New ObjectMapper()

(but don’t cache it)

→ Parse JSON string into Map

→ Load and parse “Transform” JSON from the resource path

(but don’t cache it)

→ Serialize transformed Map *back* into string result

COMCAST

HAS THIS EVER HAPPENED TO YOU?

```
String transformedJson =
new JoltJsonTransformer()
        .transformJson(myJsonValue,
        pTool.getRequired(SPEC_PATH));
JSONObject inputPayload = new
JSONObject(transformedJson);
try {
aType = inputPayload.getJSONObject("info")
        .getJSONObject("aType").
        getString("string");
} catch (Exception e) {
// ignore, invalid object
aType = STATUS_INVALID;
}
```

New "JOLT" JSON Transformer

(but don't cache it)

→ New ObjectMapper()

(but don't cache it)

→ Parse JSON string into Map

→ Load and parse "Transform" JSON from the resource path

(but don't cache it)

→ Serialize transformed Map *back* into string result

Then let's make a NEW JSONObject from that string

COMCAST

HAS THIS EVER HAPPENED TO YOU?

```
String transformedJson =
new JoltJsonTransformer()
        .transformJson(myJsonValue,
        pTool.getRequired(SPEC_PATH));
JSONObject inputPayload = new
JSONObject(transformedJson);
try {
aType = inputPayload.getJSONObject("info")
        .getJSONObject("aType").
        getString("string");
} catch (Exception e) {
// ignore, invalid object
aType = STATUS_INVALID;
}
```

New "JOLT" JSON Transformer

(but don't cache it)

→ New ObjectMapper()

(but don't cache it)

→ Parse JSON string into Map

→ Load and parse "Transform" JSON from the resource path

(but don't cache it)

→ Serialize transformed Map *back* into string result

Then let's make a NEW JSONObject from that string

Go deep into the path to get value

COMCAST

HAS THIS EVER HAPPENED TO YOU?

```
String transformedJson =
new JoltJsonTransformer()
        .transformJson(myJsonValue,
        pTool.getRequired(SPEC_PATH));
JSONObject inputPayload = new
JSONObject(transformedJson);
try {
aType = inputPayload.getJSONObject("info")
        .getJSONObject("aType").
        getString("string");
} catch (Exception e) {
// ignore, invalid object
aType = STATUS_INVALID;
}
```

New "JOLT" JSON Transformer

(but don't cache it)

→ New ObjectMapper()

(but don't cache it)

→ Parse JSON string into Map

→ Load and parse "Transform" JSON from the resource path

(but don't cache it)

→ Serialize transformed Map *back* into string result

Then let's make a NEW JSONObject from that string

Go deep into the path to get value

Cast exception? STATUS_INVALID

SLIGHTLY MORE EFFICIENT SOLUTION

```
if (mapper == null)
  mapper = new ObjectMapper();
```

Cache our new ObjectMapper()

```
Map<?, ?> resultMap =
JsonUtils.readValue(mapper, (String)
result, Map.class);
```

Parse JSON string into Map *ONCE*

```
MapTreeWalker mapWalker = new
MapTreeWalker(resultMap);
```

Lightweight Map Traversal utility

```
final String aType =
       mapWalker.step("info")
      .step(“aSchemaInfo")
      .step(“aType")
      .step("string")
      .<String>get().
      orElse(STATUS_INVALID);
```

Java Stream syntax

Optional.ofNullable

Optional.orElse

FLINK OBJECT EFFICIENCIES

```
StreamExecutionEnvironment env;
env.getConfig().enableObjectReuse();
```

REDUCE OBJECT CREATION AND GARBAGE COLLECTION

SEMANTIC ANNOTATIONS

```
@NonForwardedFields
@ForwardedFields
```

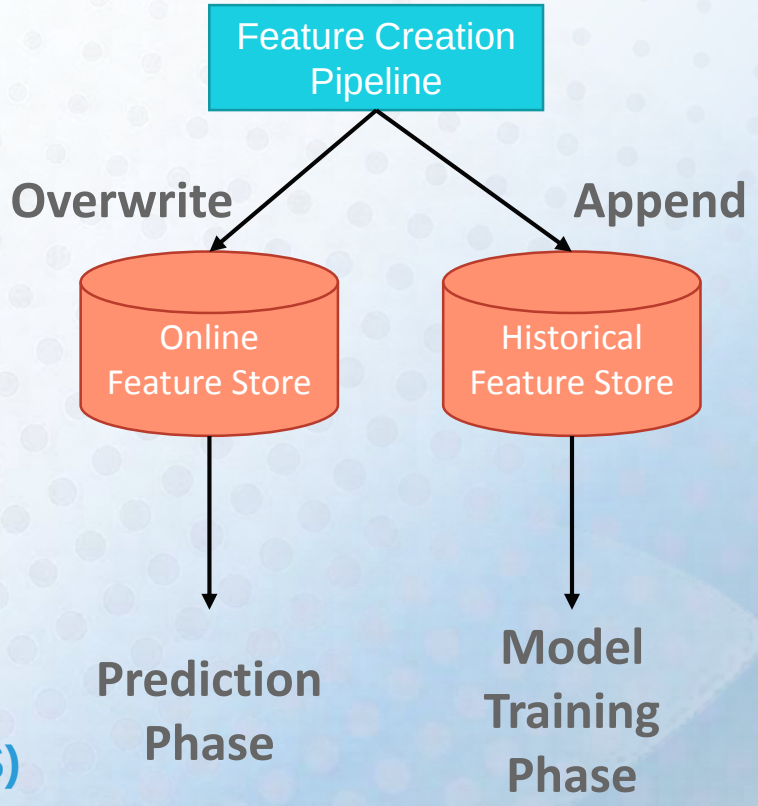
BE CAREFUL OF SIDE EFFECTS WITHIN OPERATORS

COMCAST

THE **FEATURE STORE** PROBLEM

WHAT IS A FEATURE STORE?

“FEATURE” IS A DATA POINT INPUT TO ML MODEL



WE NEED TO:

- **STORE ALL THE INPUT DATA POINTS**
- **SNAPSHOT AT ANY MOMENT IN TIME**
- **ASSOCIATE WITH A DIAGNOSIS (MODEL OUTPUT)**
- **HAVE ACCESS TO THE DATA POINTS (API FOR OTHER APPS)**
- **STORE ALL DATA POINTS FOR ML MODEL TRAINING**

COMCAST

FIRST TRY: AWS AURORA RDBMS

"HEY IT SHOULD GET US UP TO 10,000 / SECOND"

…THE UPSERT PROBLEM

DIDN'T WORK MUCH PAST 3,000 / SECOND

…SOON OUR INPUT RATE WAS 20,000 / SECOND

COMCAST

MITIGATION: STORE ONLY AT DIAGNOSIS TIME

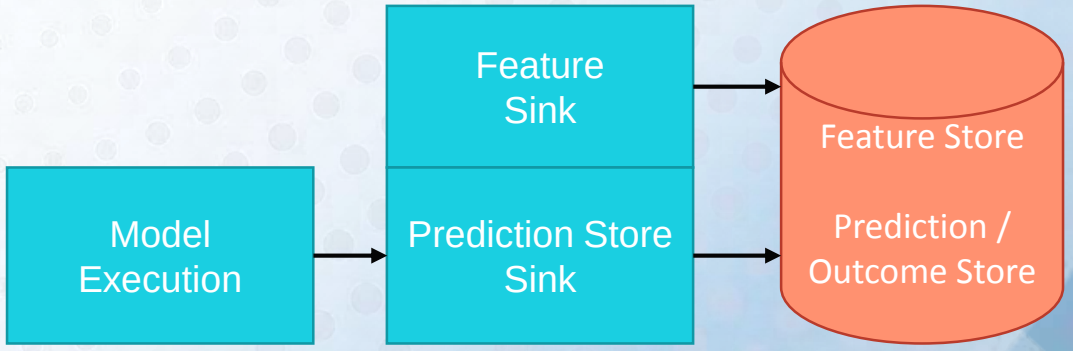
STOPPED STORING ALL RAW DATA POINTS

ONLY STORE DATA POINTS ALONGSIDE DIAGNOSIS

CON: ONLY A SMALL % OF DATA POINTS

CON: DATA NOT CURRENT, ONLY AS OF RECENT DIAGNOSIS

CON: LIMITED USEFULNESS



COMCAST

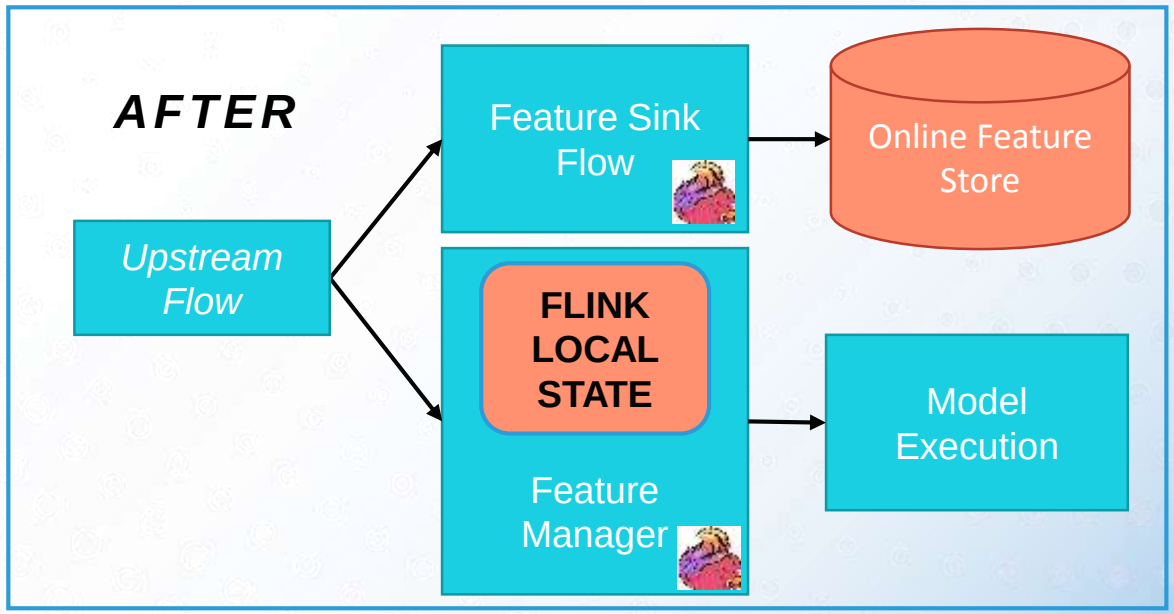
OPTIMIZING THE WRITE PATH

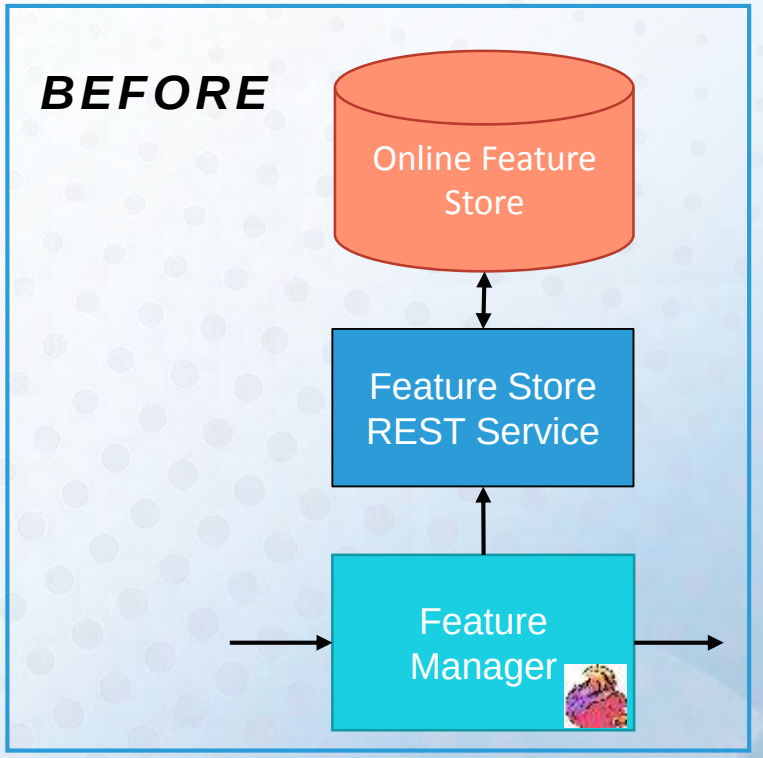
WHICH FLINK FLOW WRITES TO FEATURE STORE?

FEATURE STORE NOT NEEDED FOR TRIGGER

FEATURE STORE NOT NEEDED FOR MODEL EXECUTION

SEPARATE ‘TRANSFORM AND SINK’ FLOW





COMCAST

ALTERNATIVE DATA STORES

REDIS

STORE ONLY LATEST DATA POINTS?

CASSANDRA

COST AND COMPLEXITY?

FLINK QUERYABLE STATE

PRODUCTION HARDENING?

READ VOLUME?

TIME SERIES DB

DRUID

INFLUX

COMCAST

REDIS

WE ONLY NEED < 7 DAYS OF DATA, ONLY LATEST VALUE, SO USE REDIS AS THE KV STORE

ACCESSIBLE BY OTHER CONSUMERS OR VIA A SERVICE

COULDN'T PUT MORE THAN 8,000 / SECOND FOR A 'REASONABLE' CLUSTER SIZE

FLINK SINK CONNECTOR IS NOT CLUSTER-ENABLED

ONE OBJECT PUT PER CALL

JEDIS VS LETTUCE FRAMEWORK AND PIPELINING

SOME OPTIMIZATION IF ABLE TO BATCH REDIS SHARDS BY PRE-COMPUTING

COMCAST

FLINK QUERYABLE STATE

REUSE OUR FEATURE REQUEST/RESPONSE FRAMEWORK

SCALABLE, FAST

OUR READ LOAD IS LOW (10 / SECOND)

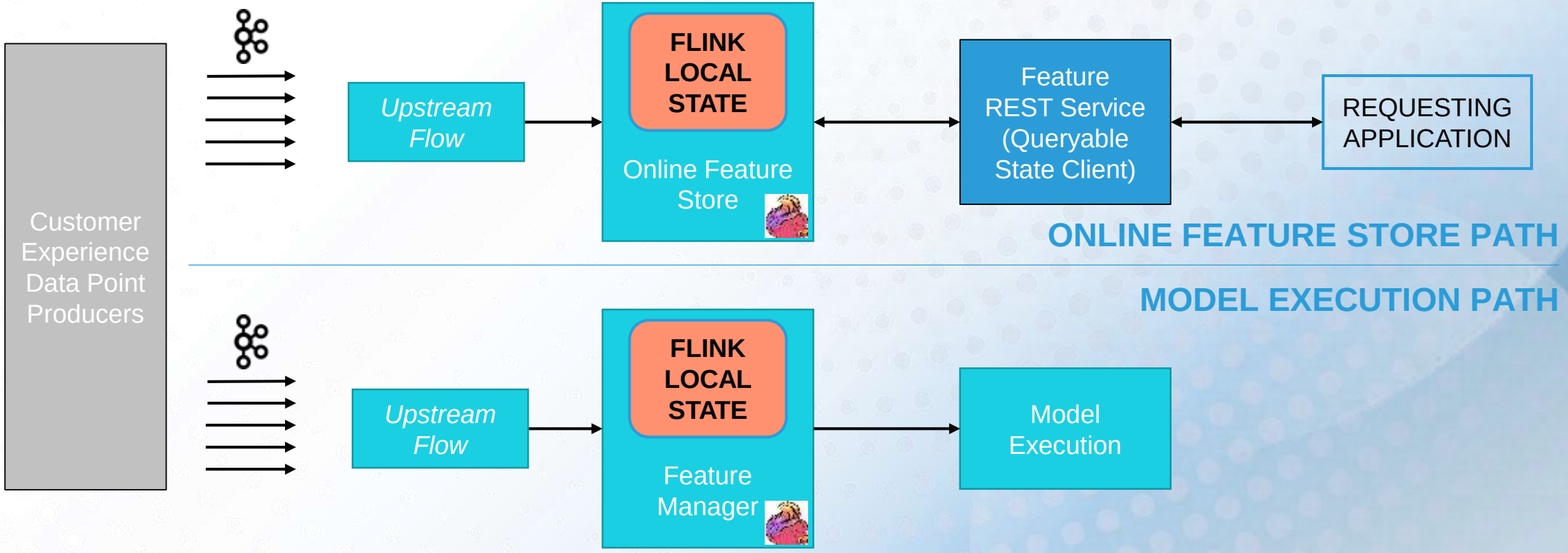
EXPENSIVE (RELATIVELY)

NOT A “PRODUCTION” CAPABILITY

NOT A “DATABASE” TECHNOLOGY

COMCAST

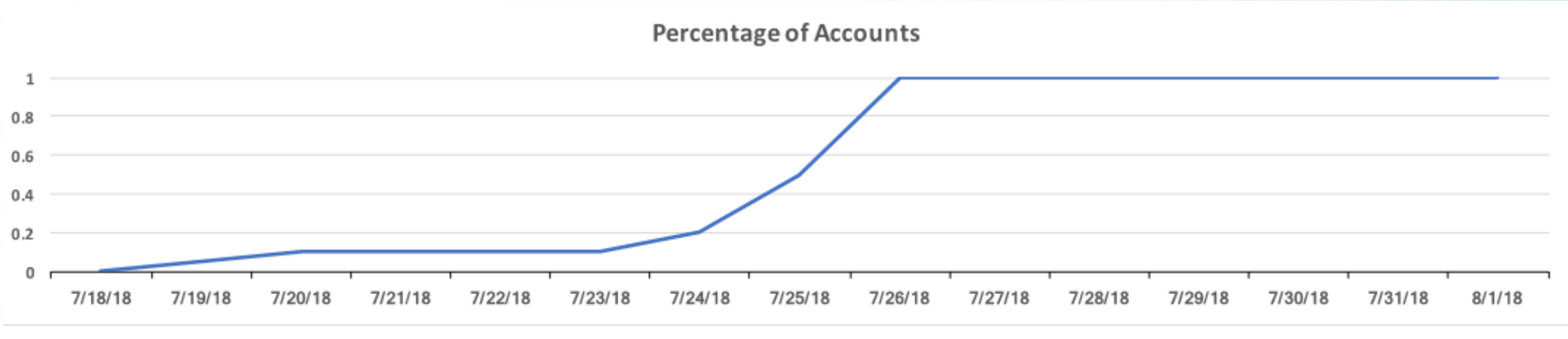
QUERYABLE STATE - ONLINE FEATURE STORE



COMCAST

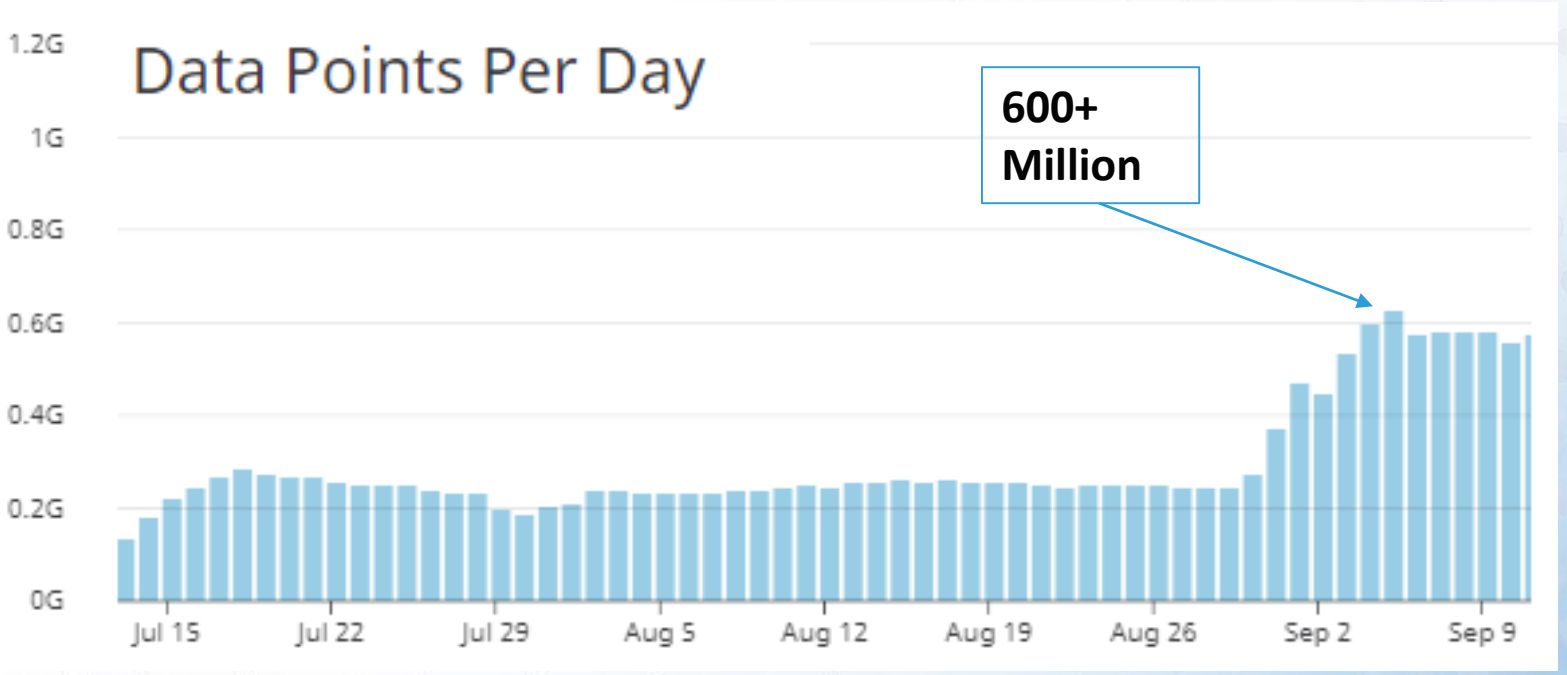
THE **VOLUME** PROBLEM

RAMPED UP PRODUCTION – JULY 2018



COMCAST

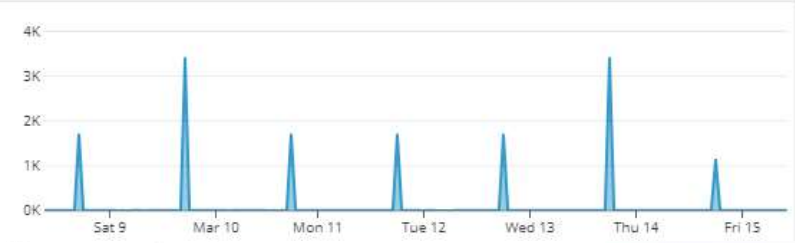
ADDED NEW DATA POINT TYPES



COMCAST

INCREASED DATA POINT FREQUENCY

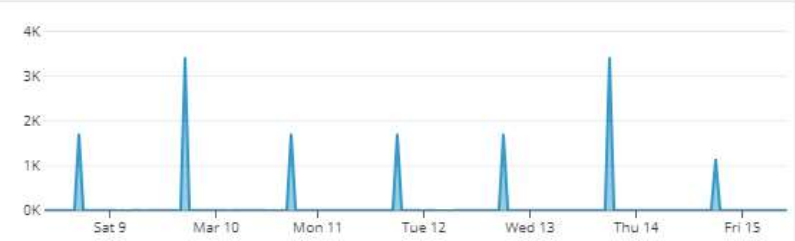
ONCE EVERY 24 HOURS



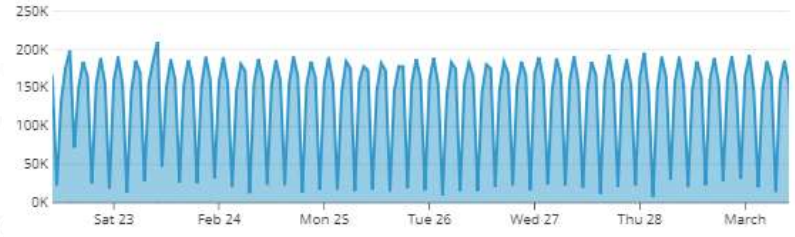
COMCAST

INCREASED DATA POINT FREQUENCY

ONCE EVERY 24 HOURS



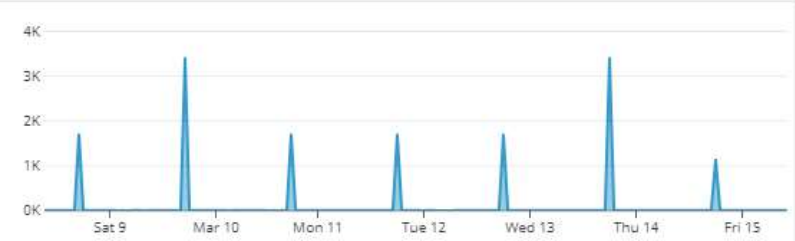
ONCE EVERY 4 HOURS



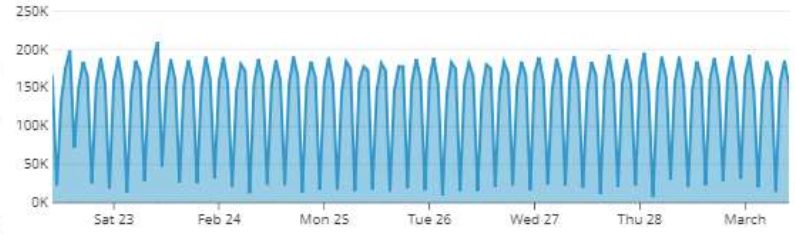
COMCAST

INCREASED DATA POINT FREQUENCY

ONCE EVERY 24 HOURS



ONCE EVERY 4 HOURS



ONCE EVERY 5 MINUTES ???

COMCAST

SYMPTOM: KAFKA LATENCY INCREASING

FLINK CLUSTER COULDN'T KEEP UP WITH KAFKA VOLUME

COMCAST

KAFKA OPTIMIZATION

INCREASE PARTITIONS

1 KAFKA PARTITION

READ BY

1 FLINK TASK THREAD

MATCH PARALLELISM TO PARTITIONS (OR EVEN MULTIPLE)

100 KAFKA PARTITIONS

50 SLOT PARALLELISM

50 KAFKA PARTITIONS

50 SLOT PARALLELISM

ADJUST KAFKA CONFIGURATION

 # KAFKA CONSUMER SETTINGS

MAX.POLL.RECORDS = ~~500~~ 10000

FETCH.MIN.BYTES = ~~1~~ 102400

FETCH.MAX.WAIT.MS = ~~500~~ 1000

 # KAFKA PRODUCER SETTINGS

BUFFER.MEMORY = 268435456

BATCH.SIZE = ~~131072~~ 524288

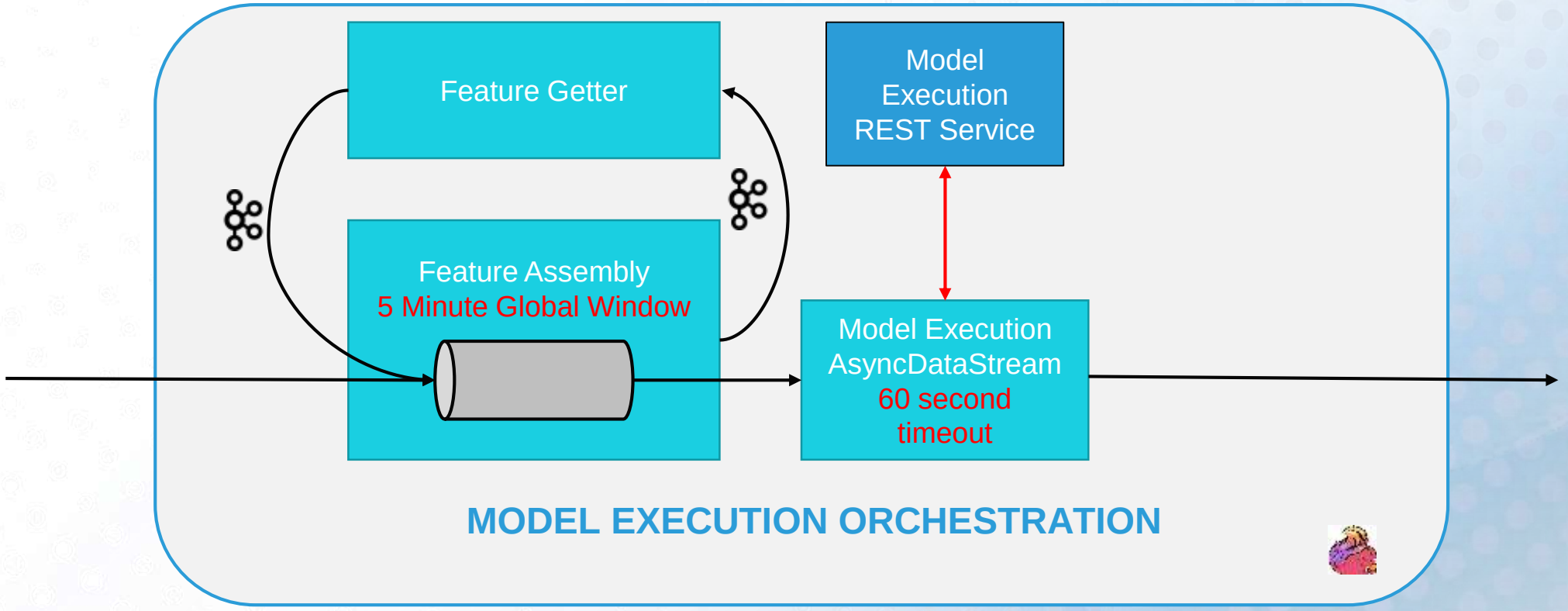
LINGER.MS = ~~0~~ 50

REQUEST.TIMEOUT.MS = ~~90000~~ 600000

COMCAST

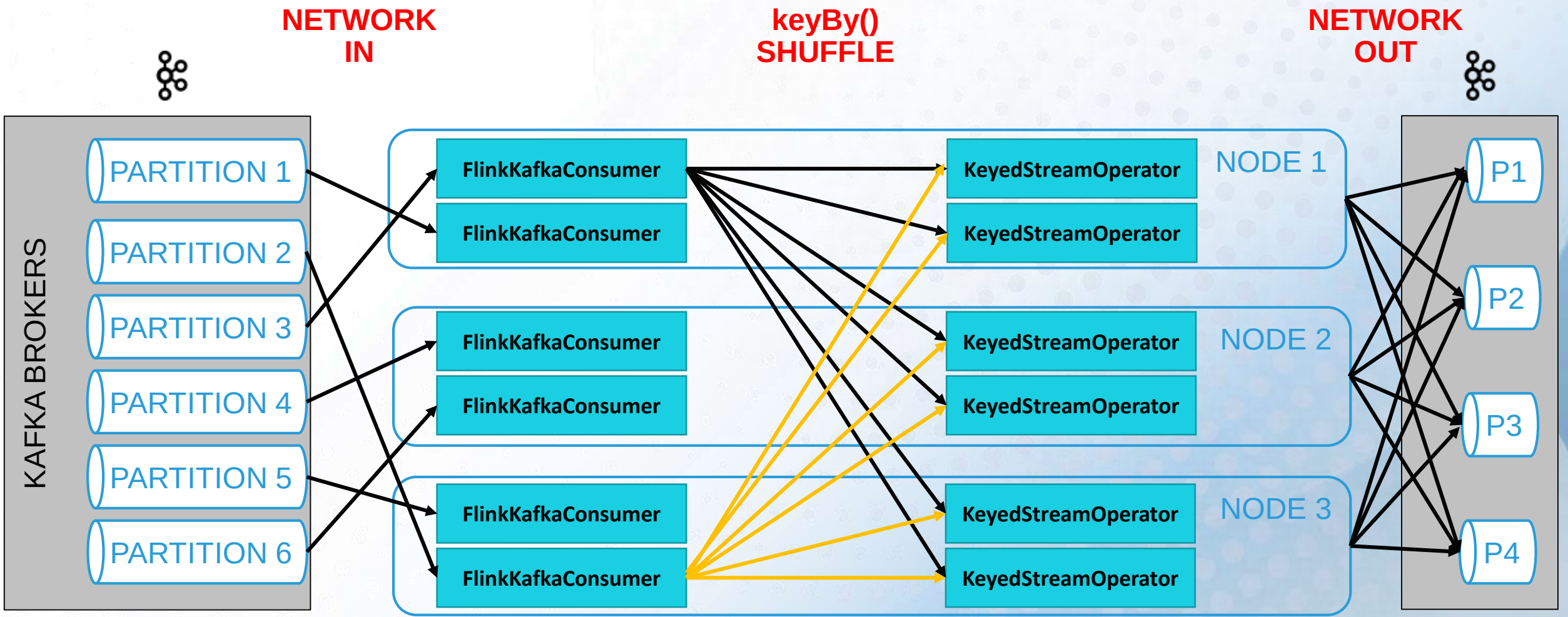
MODEL EXECUTION BACKUP PROBLEM

GENERAL SLOWNESS BACKS UP MODEL EXECUTION PIPELINE



COMCAST

CLUSTER NETWORK I/O TRAFFIC



COMCAST

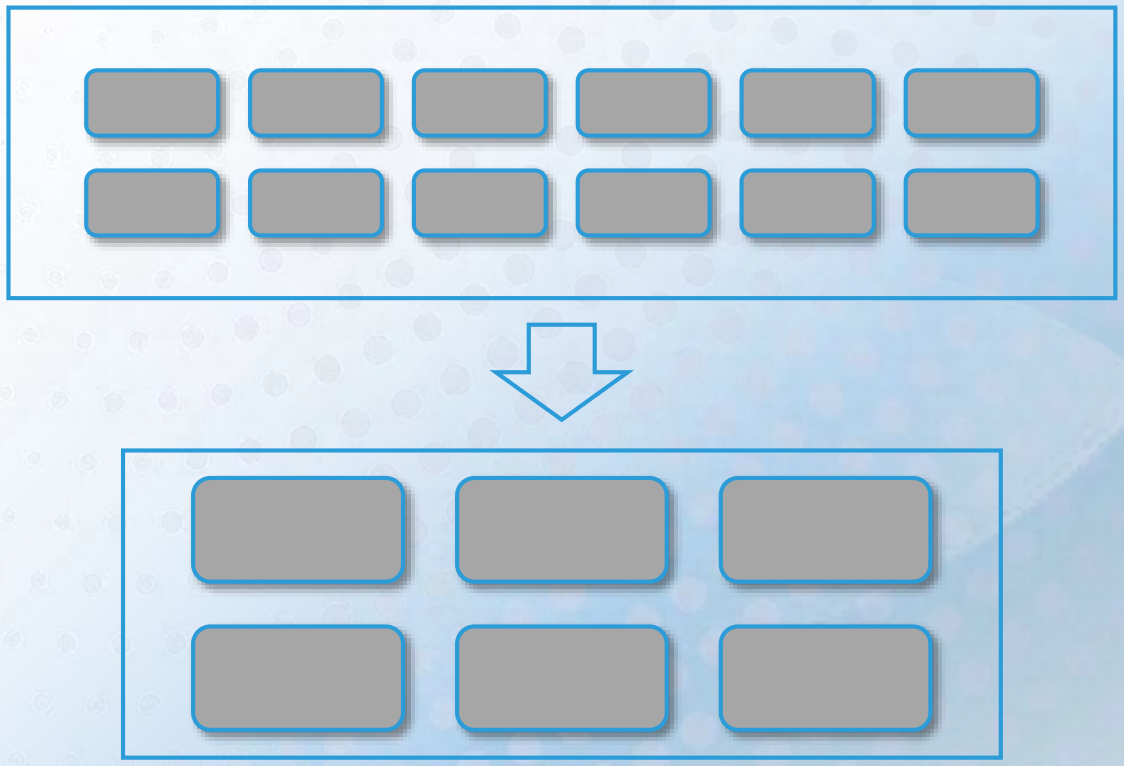
CLUSTER NETWORK I/O VOLUME

(10,000 MESSAGES / SECOND) * (1KB / MESSAGE) =
10 MB / SECOND

100 MBITS/SEC NETWORK

SOLUTION – LARGER NODE TYPES WITH MORE VCPU AND HIGHER PARALLELISM… FEWER NODES IN CLUSTER

SWEET SPOT SEEMED TO BE ~20 NODES



GOOD READ: HTTPS://WWW.VERVERICA.COM/BLOG/HOW-TO-SIZE-YOUR-APACHE-FLINK-CLUSTER-GENERAL-GUIDELINES

COMCAST

THE **CUSTOMER STATE** PROBLEM

KEEPING TRACK OF DATA POINT STATE

25+ MILLION HIGH SPEED INTERNET CUSTOMERS

10+ DATA POINT TYPES

TWO FLINK STATES TO MANAGE:

RED / GREEN "TRIGGER"

MAPSTATE () / KEYBY (ENTITY ID)

[ENTITY ID] (DATA POINT TYPE, STATE)

25 MILLION CUSTOMERS

10+ DATA POINT TYPES

AVG 33 BYTES PER ENTITY ID

~1GB TOTAL

QUERYABLE STATE FEATURE STORE

UP TO 15KB (JSON) PER DATA POINT

10KB * 25 MILLION = 250 GB

1KB * 25 MILLION = 25 GB

APPROXIMATE TOTAL = 1 TB

COMCAST

THE **CHECKPOINT** PROBLEM

CHECKPOINT TUNING

FEATURE MANAGER – UP TO 1TB OF STATE

2.5 MINUTES WRITING A CHECKPOINT EVERY 10 MINUTES

TURNED OFF FOR AWHILE!

(RATIONALE: WOULD ONLY TAKE A FEW HOURS TO RECOVER ALL STATE)

INCREMENTAL CHECKPOINTING HELPED

LONG ALIGNMENT TIMES ON HIGH PARALLELISM

REDUCED THE SIZE OF STATE FOR 4 AND 24 HOUR WINDOWS USING FEATURE LOOKUP TABLE

COMCAST

THE **HA** PROBLEM

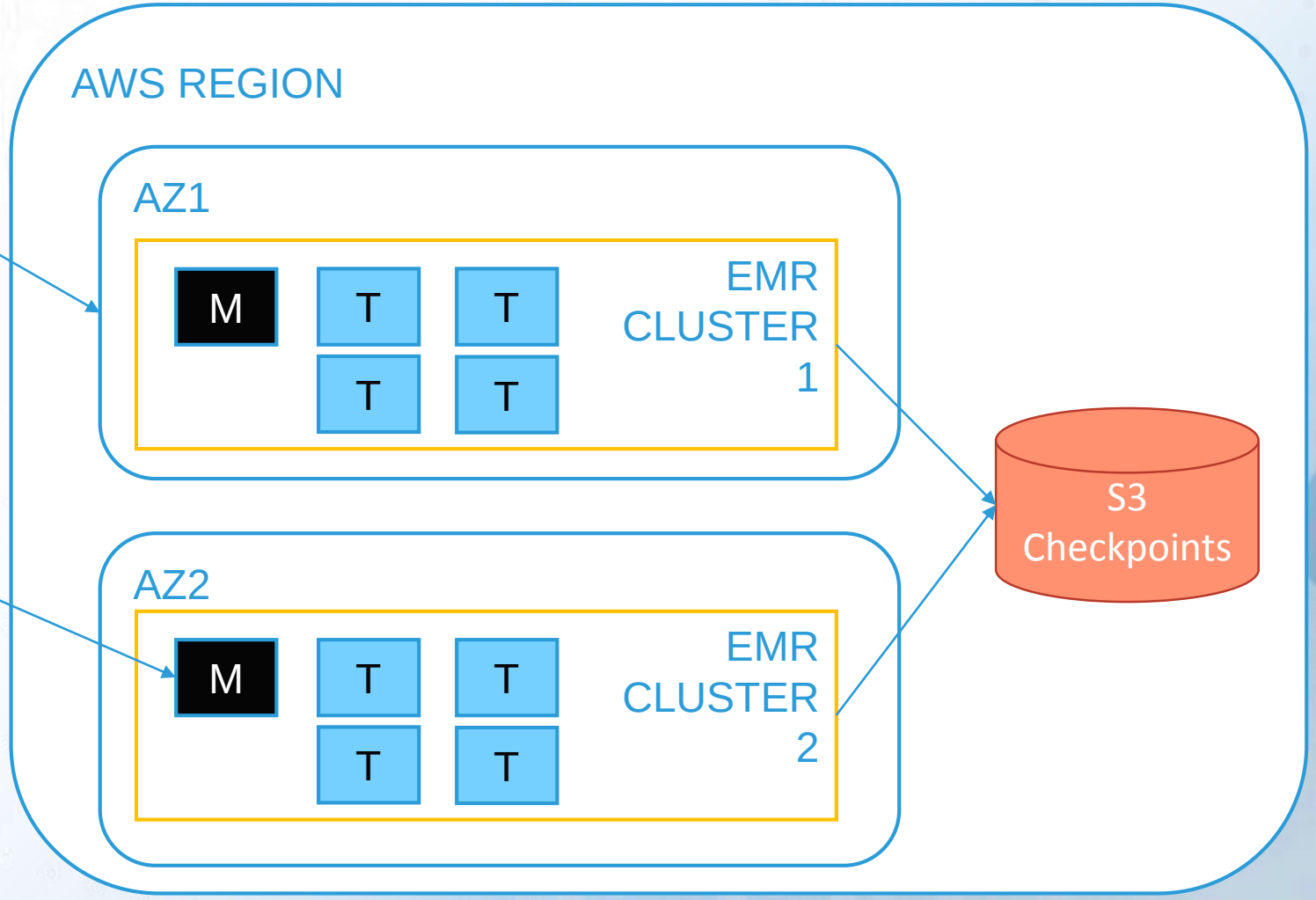
AMAZON EMR AND HIGH AVAILABILITY

EMR FLINK CLUSTER

SINGLE AVAILABILITY ZONE (AZ)

SINGLE MASTER NODE

NOT HA



COMCAST

AMAZON AWS EMR

SESSIONS KEPT DYING

TMP DIR IS PERIODICALLY CLEARED

BREAKS ‘BLOB STORE’ ON LONG RUNNING JOBS

SOLUTION - CONFIGURE TO NOT BE IN /TMP

jobmanager.web.tmpdir

blob.storage.directory

EMR FLINK VERSION LAGS

APACHE RELEASE BY 1-2 MONTHS

COMCAST

TECHNOLOGY OPTIONS

FLINK ON KUBERNETES

AMAZON EKS (ELASTIC KUBERNETES SERVICE)

SELF-MANAGED KUBERNETES ON EC2

~~DATA ARTISANS~~ VERVERICA PLATFORM

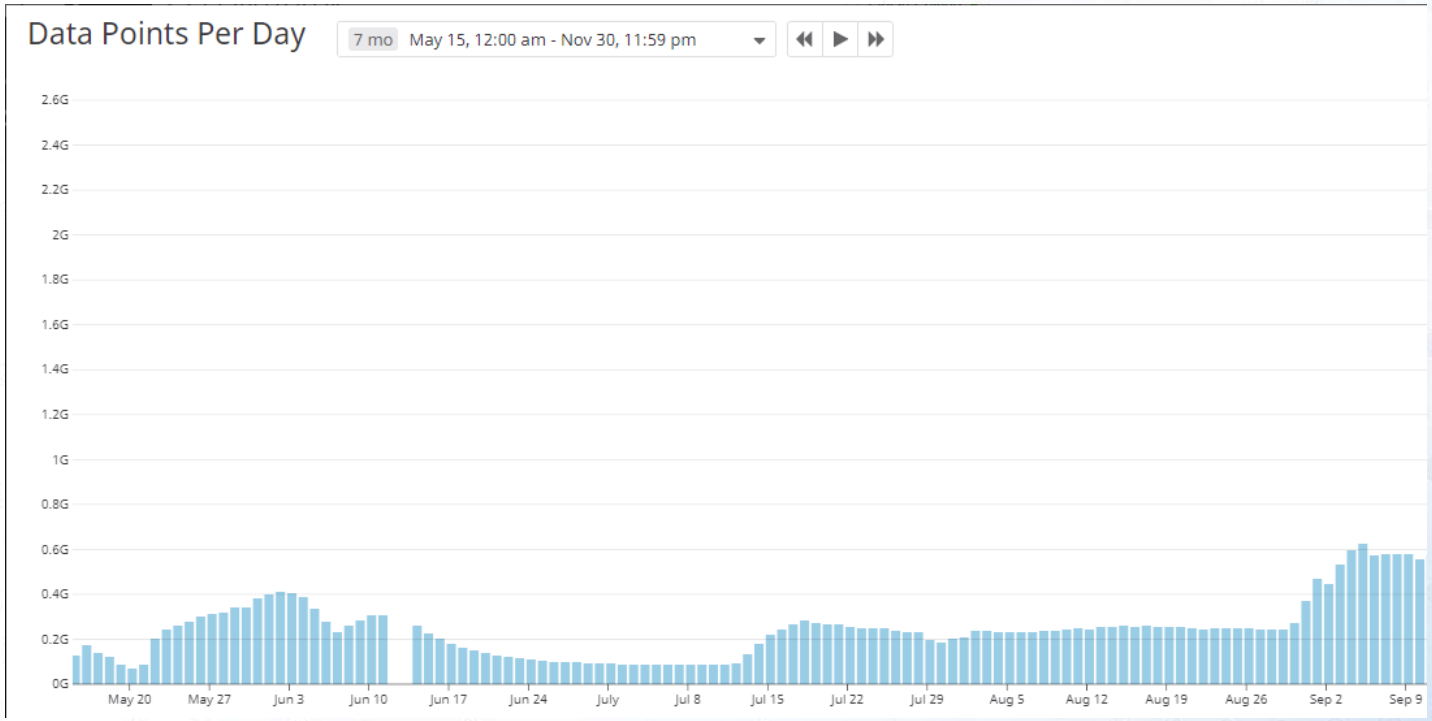
Bonus:
Reduced Cost / Overhead by having fewer Master Nodes

Less overhead for smaller (low parallelism) flows

COMCAST

THE **REALLY HIGH VOLUME AND FEATURE STORE** PROBLEM #2

DATA POINT VOLUME MAY–NOVEMBER 2018

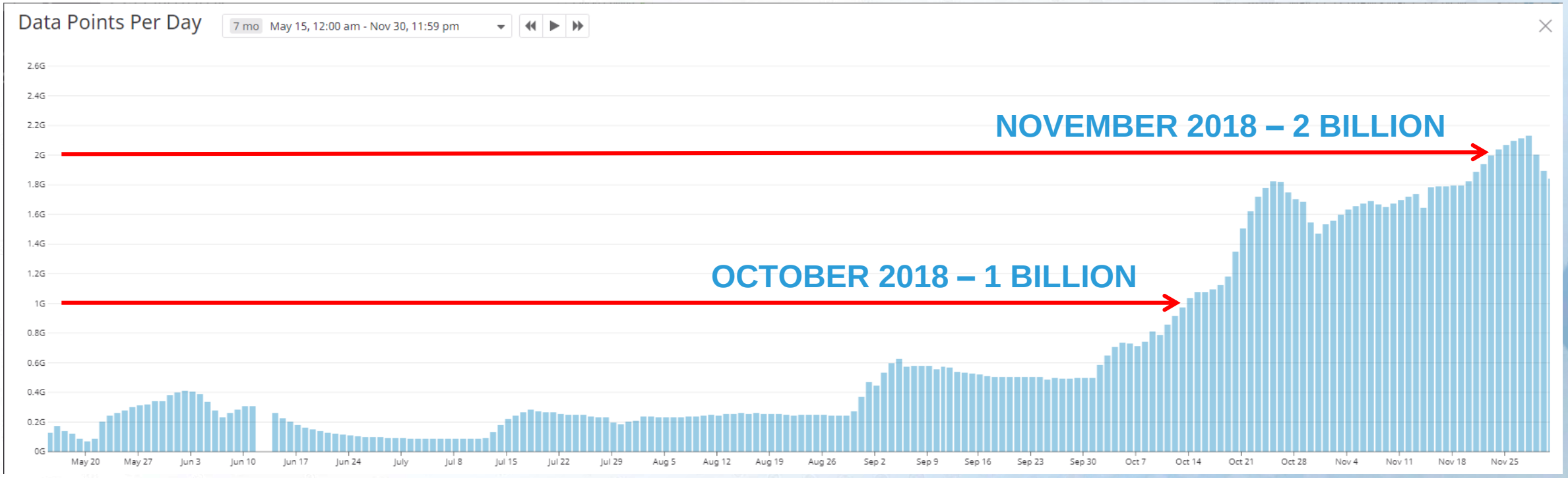


COMCAST

DATA POINT VOLUME MAY–NOVEMBER 2018

COMCAST

DATA POINT VOLUME MAY–NOVEMBER 2018

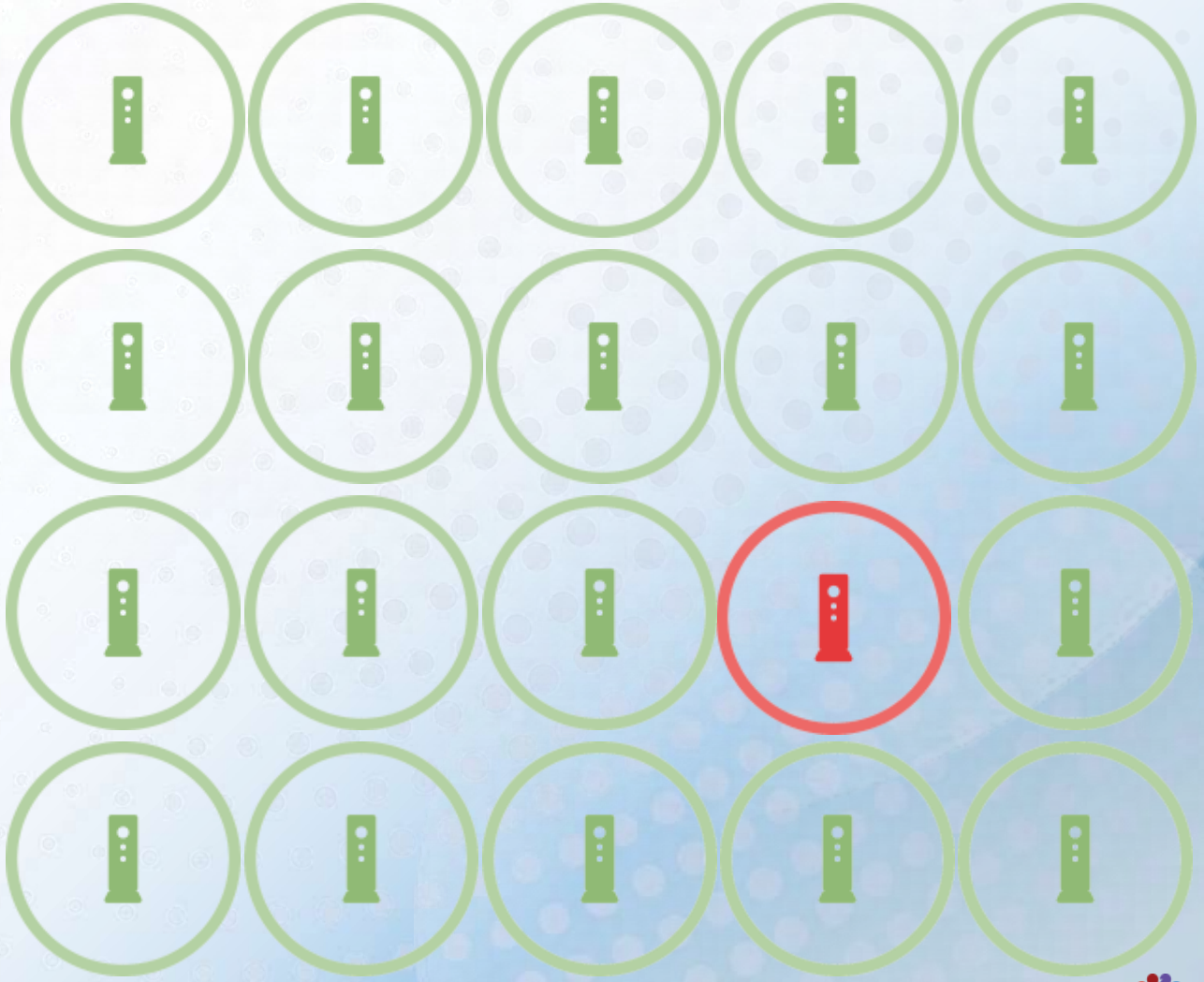


COMCAST

LET'S ADD ANOTHER 3 BILLION!

15 MILLION DEVICES

3 BILLION DATA POINTS



COMCAST

SOLUTION APPROACH

1 ISOLATE TRIGGER FOR HIGH-VOLUME DATA POINT TYPES

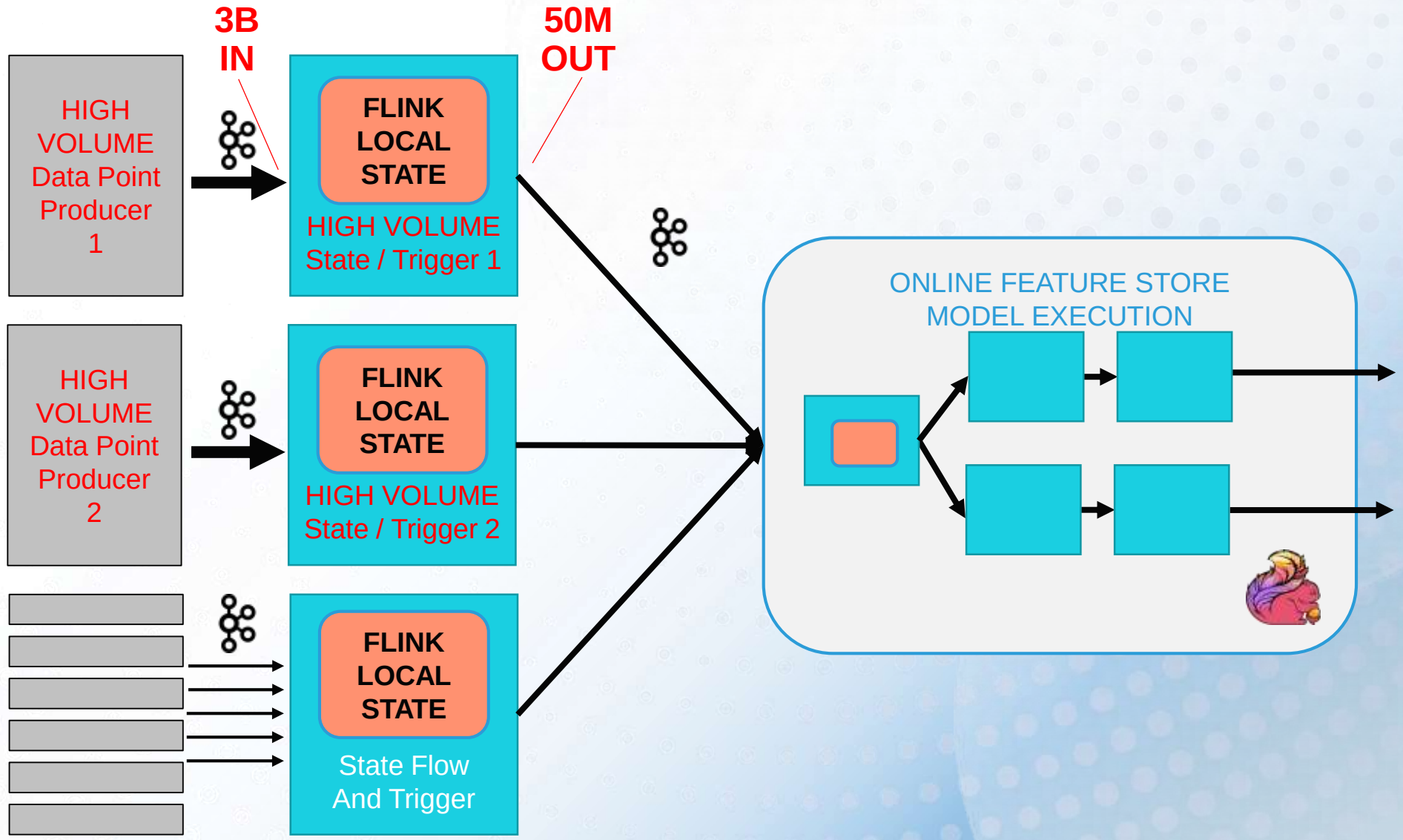
2 ISOLATE ONLINE FEATURE STORE FOR HIGH- VOLUME DATA POINT TYPES

3 ISOLATE DEDICATED FEATURE STORAGE FLOW TO HIGH-SPEED FEATURE STORE

4 SEPARATE HISTORICAL FEATURE STORAGE (S3 WRITER) FROM REAL TIME MODEL EXECUTION FLOWS

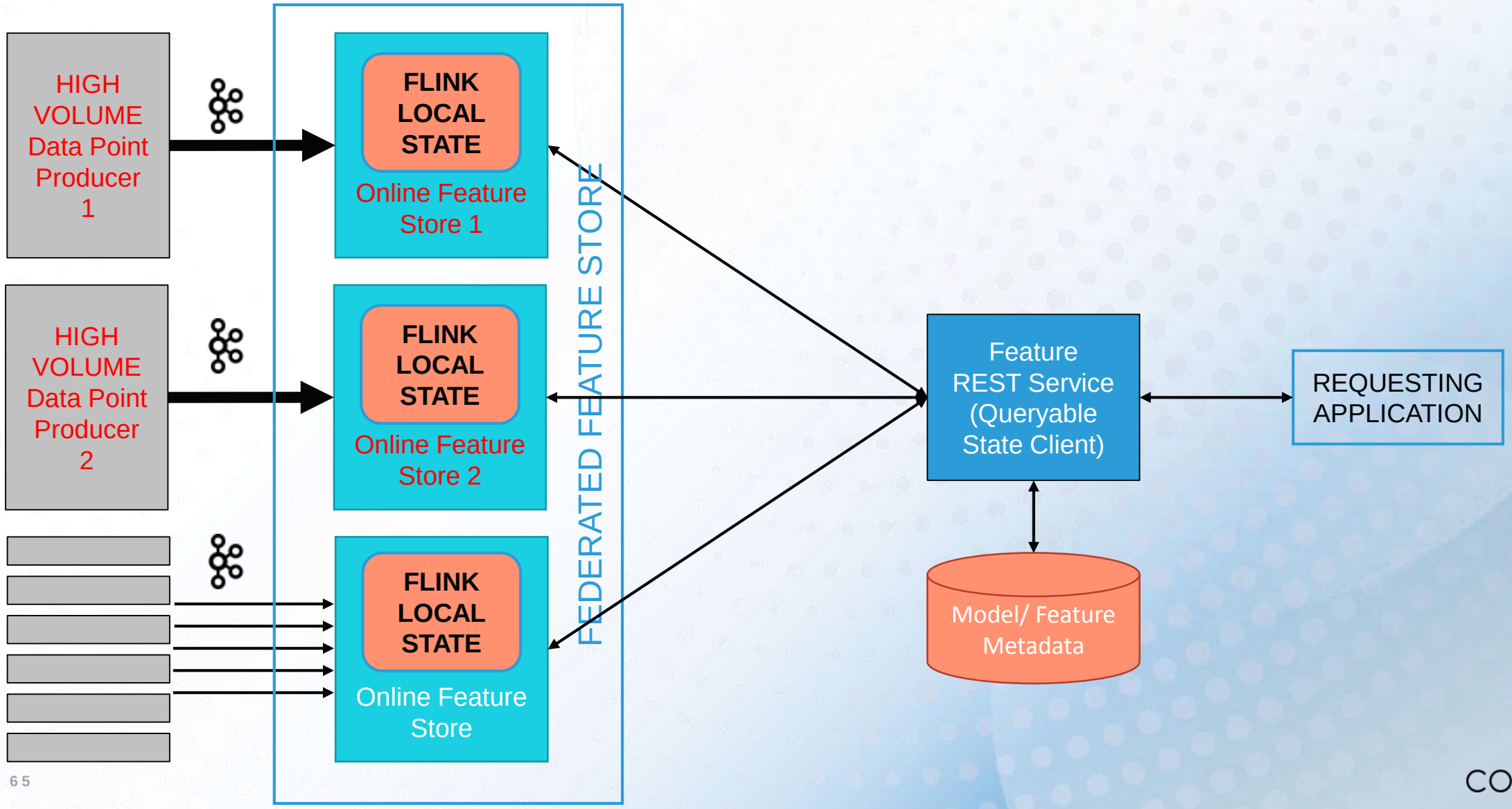
COMCAST

1: SEPARATE HIGH VOLUME TRIGGER FLOWS



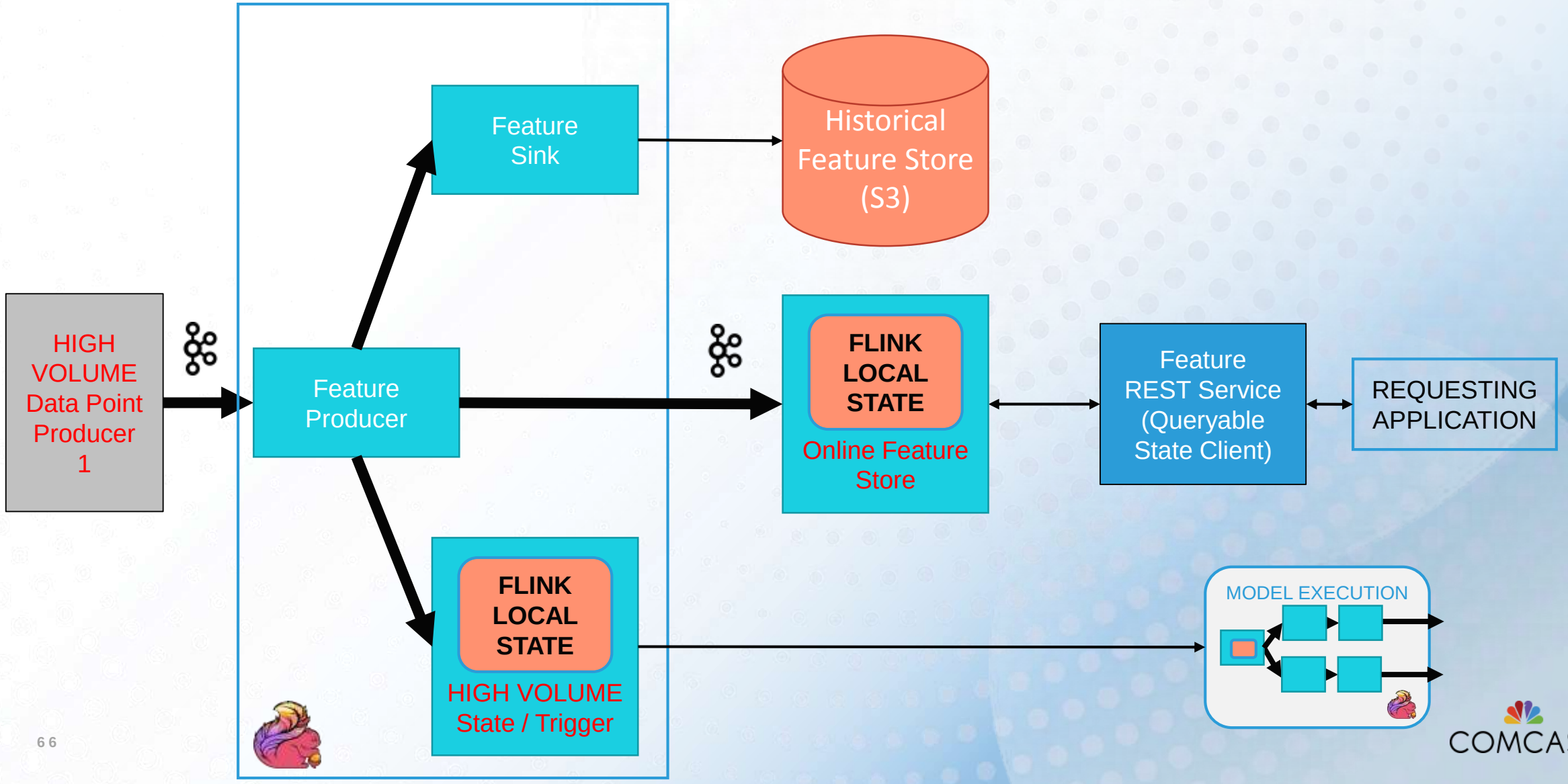
COMCAST

2: FEDERATED ONLINE FEATURE STORE



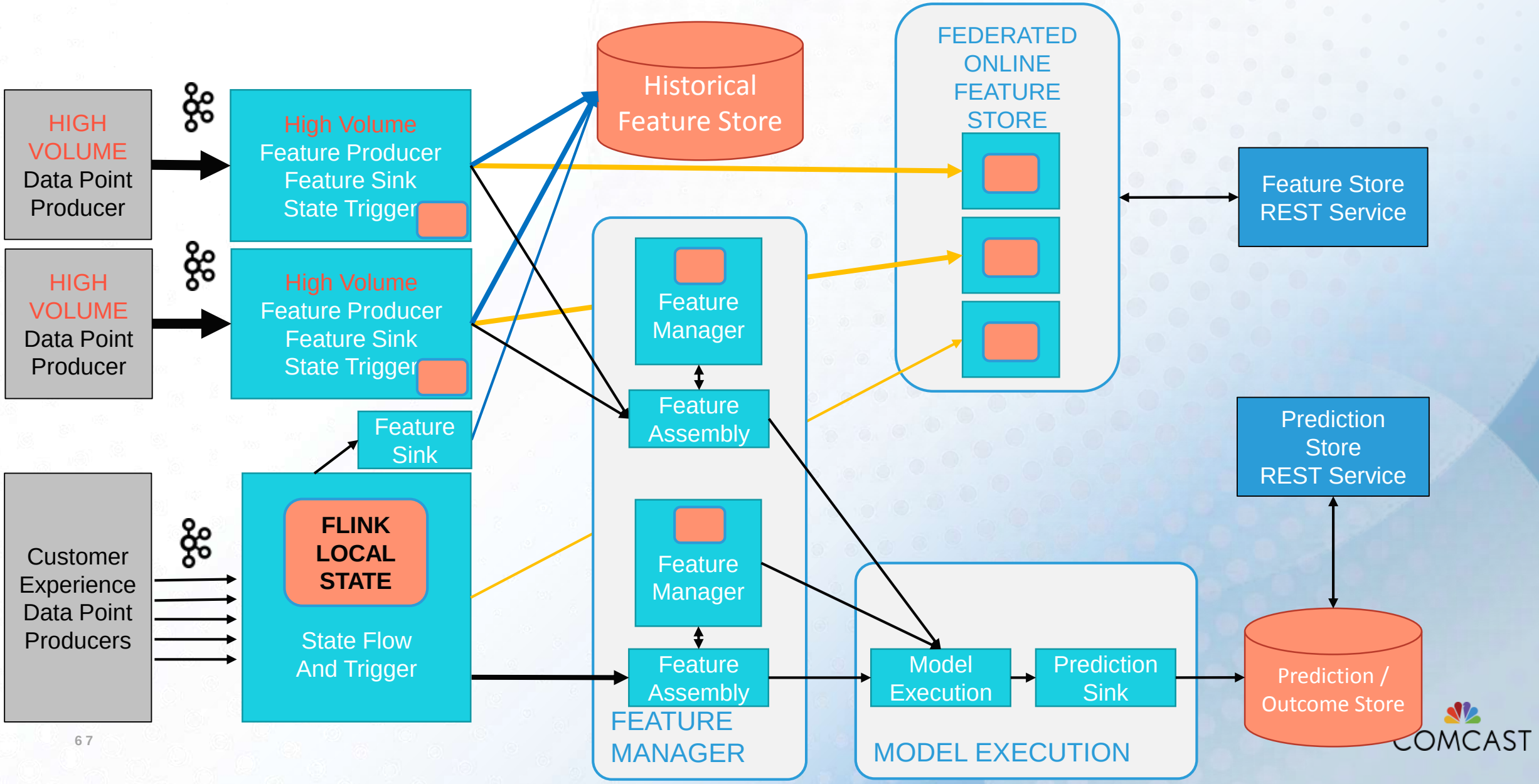
COMCAST

3+4: HISTORICAL FEATURE STORE



COMCAST

CURRENT ARCHITECTURE



COMCAST

WHERE ARE WE TODAY

INDICATORS

725+ MILLION DATA POINTS PER DAY

HIGH-VOLUME INDICATORS

6 BILLION PER DAY

TOTAL

~7 BILLION DATA POINTS PER DAY

COMCAST

WHERE ARE WE TODAY – FLINK CLUSTERS

CLUSTERS

14

INSTANCES

150

VCPU

1100

RAM

5.8 TB

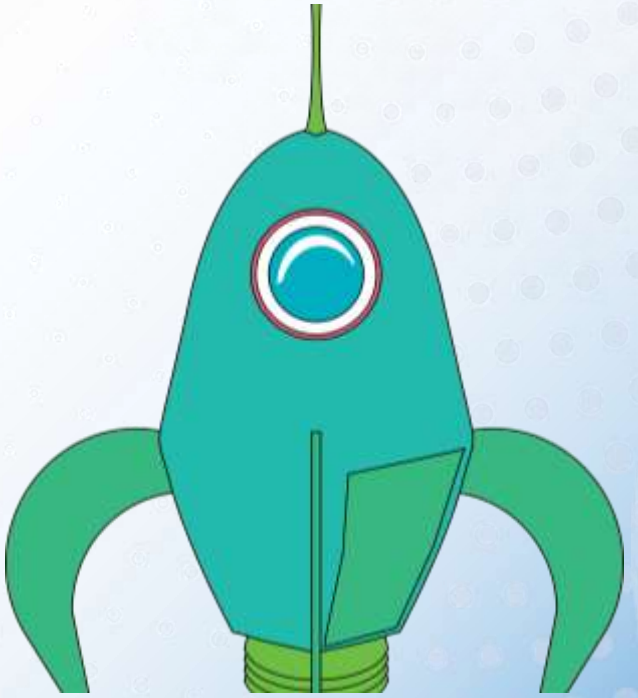
COMCAST

FUTURE WORK

EXPAND CUSTOMER EXPERIENCE INDICATORS TO OTHER COMCAST PRODUCTS

IMPROVED ML MODELS BASED ON RESULTS AND CONTINUOUS RETRAINING

MODULAR MODEL EXECUTION VIA KUBEFLOW AND GRPC CALLS



COMCAST

WE'RE HIRING!

PHILADELPHIA
WASHINGTON, D.C.
MOUNTAIN VIEW
DENVER

COMCAST

THANK YOU!


COMCAST