NEM Management

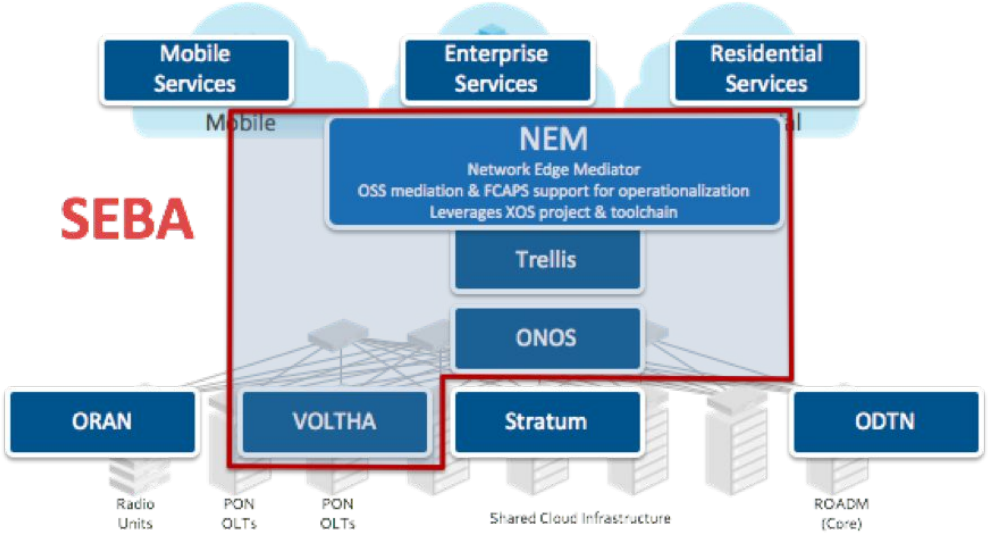
Service Lifecycle, Upgrades, API, and Tools

Scott Baker
Zack Williams

What is the NEM ?

"Network Edge Mediator"

- Interfaces operator OSS to SEBA services
- FCAPS
- Lifecycle management
  - Pods are long-lived
  - Pods are dynamic
- Dynamic -> Extensibility
  - New services
  - New workflows



The NEM Manages the pod

- NEM has configuration and authoritative state for many services
  - ONOS Apps
  - Subscribers
  - OLT and ONU admin state

- Manage the POD --> Manage the NEM

Managing the NEM …

- Service Management
  - Service A needs to work with Service B
  - Abstractions may span A and B
  - Northbound consumers want a unified abstraction.
- Container Management
  - Deploy on hardware
  - Scheduling
  - Redeploy/migrate if hardware fails
- Upgrade, Backup/Restore, …
  - Span Container & Service Management

XOS

Docker, Kubernetes, Helm

Presenting a unified NEM interface

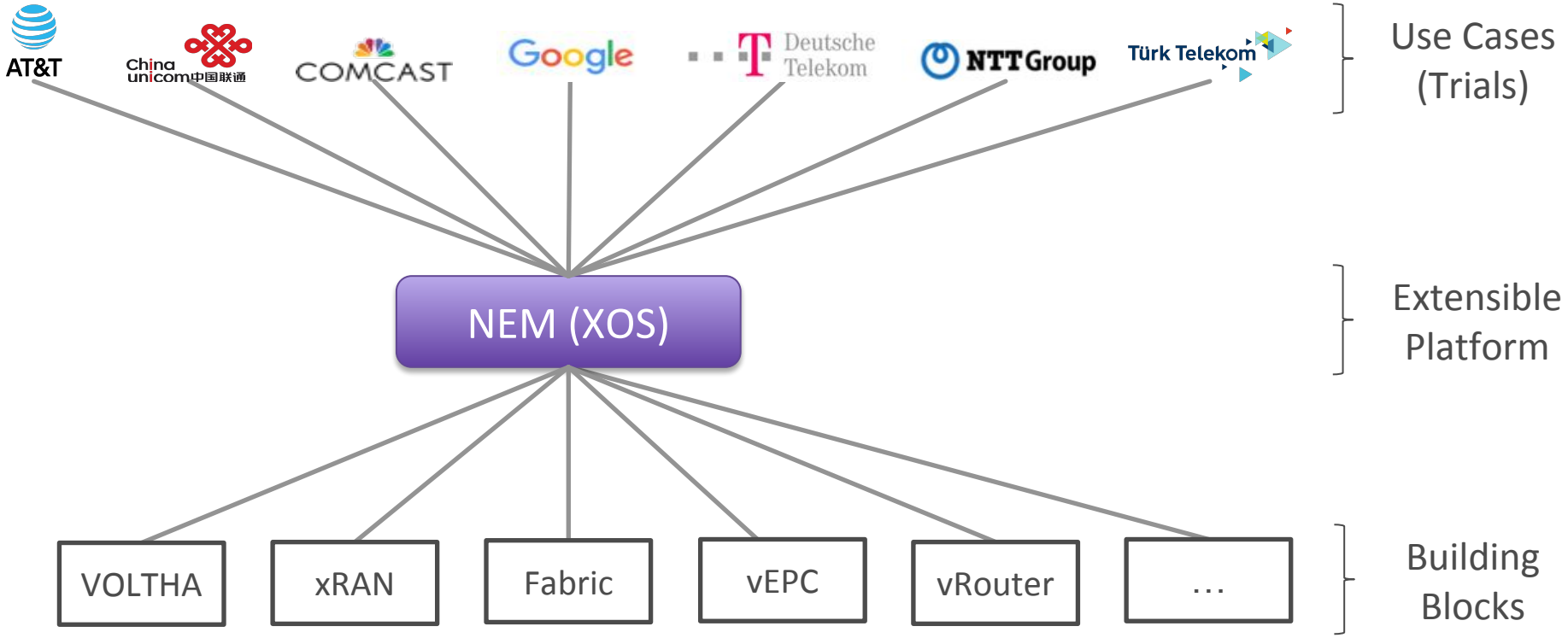
Why do we unify?

- Provide coherent interface to collection of disaggregated components
- Tools to avoid NxM scenario (N northside masters and M components)

Where do we unify?

- We created a component called “XOS”
- XOS presents a single gRPC interface northbound for operations
- XOS allows services to be plugged in southbound

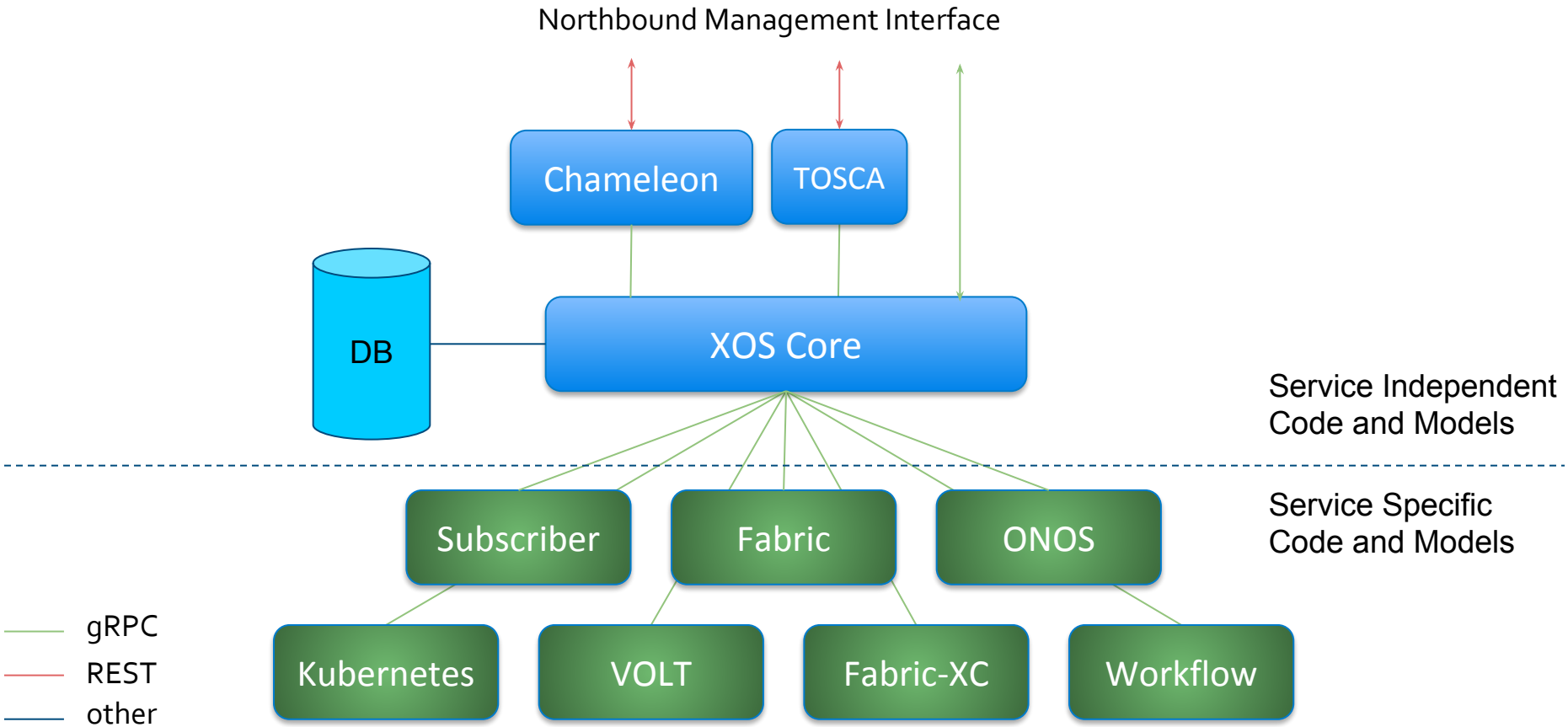
Unified Data Model – Integrate Across Components



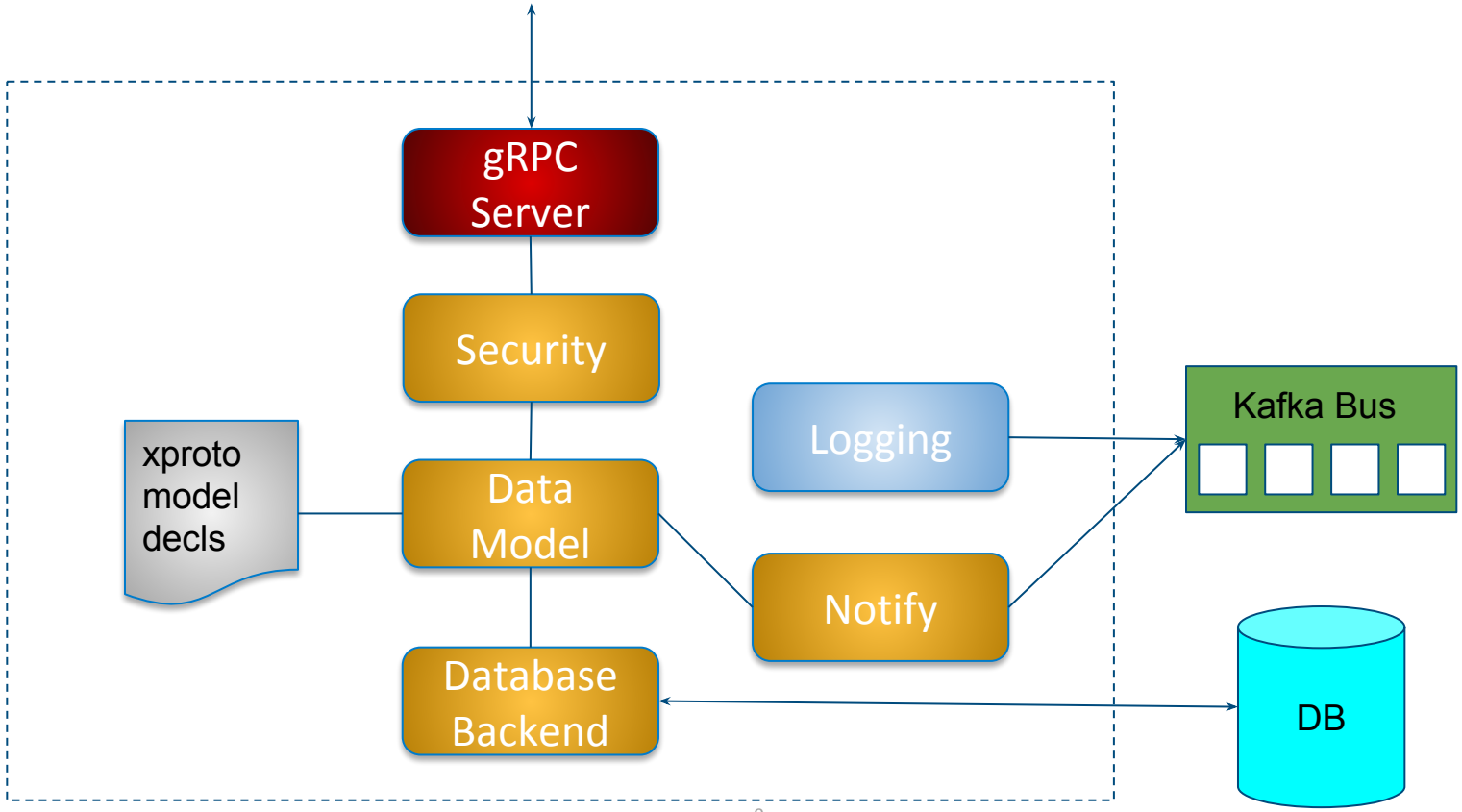
Drilling down into XOS

- Macro architecture / Containerization
- XOS Core architecture
- XOS Data model architecture

XOS Macro Architecture



XOS Core Architecture



The XOS Data Model

Relational Database

- Based on Django

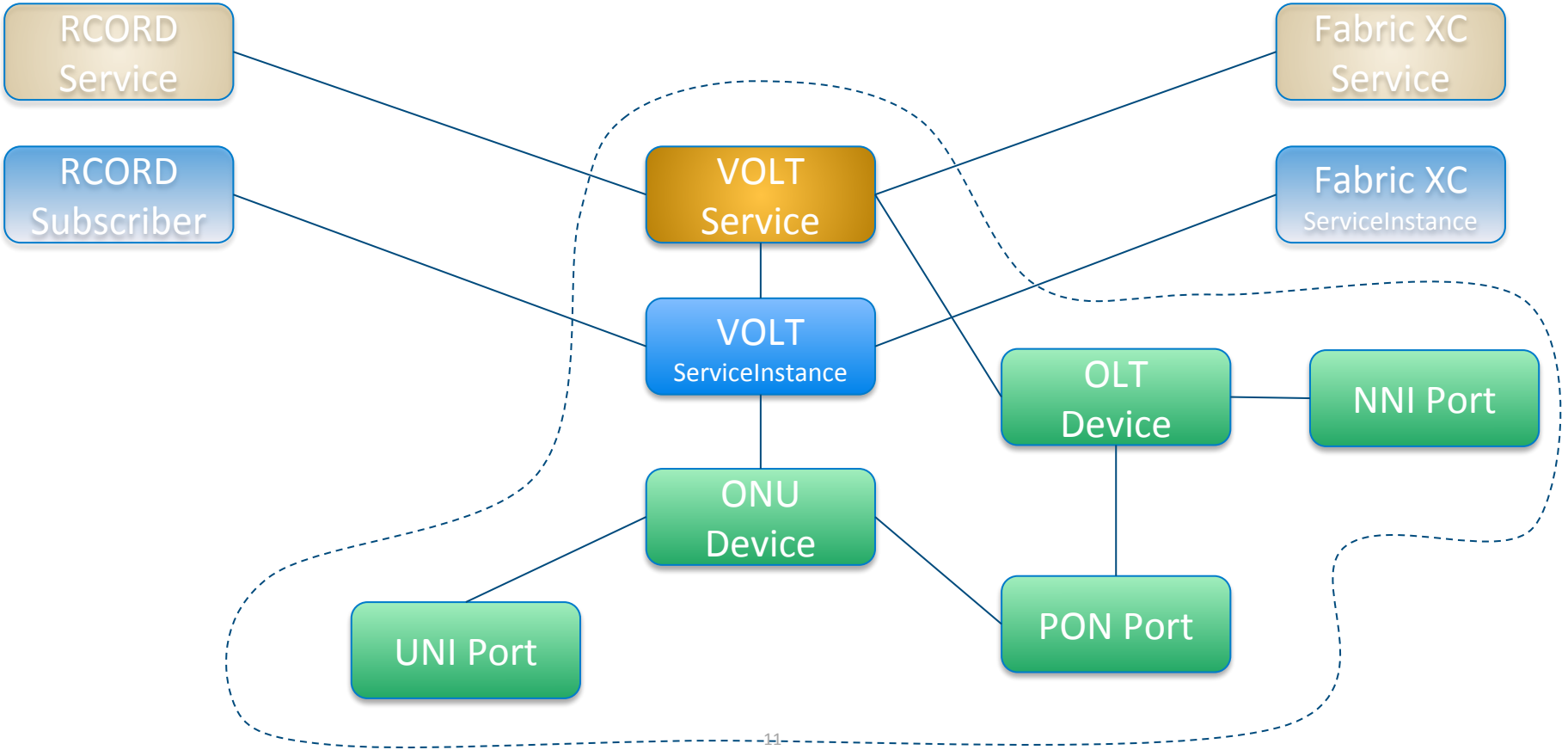
Base Models (stuff you get for free)

- Users and Permissions
- Compute and Network Resources
- Services, Tenancy, and Dependencies
- Chains

Extensibility (value you can add)

- Any service can add new models
- Service models can inherit from base models

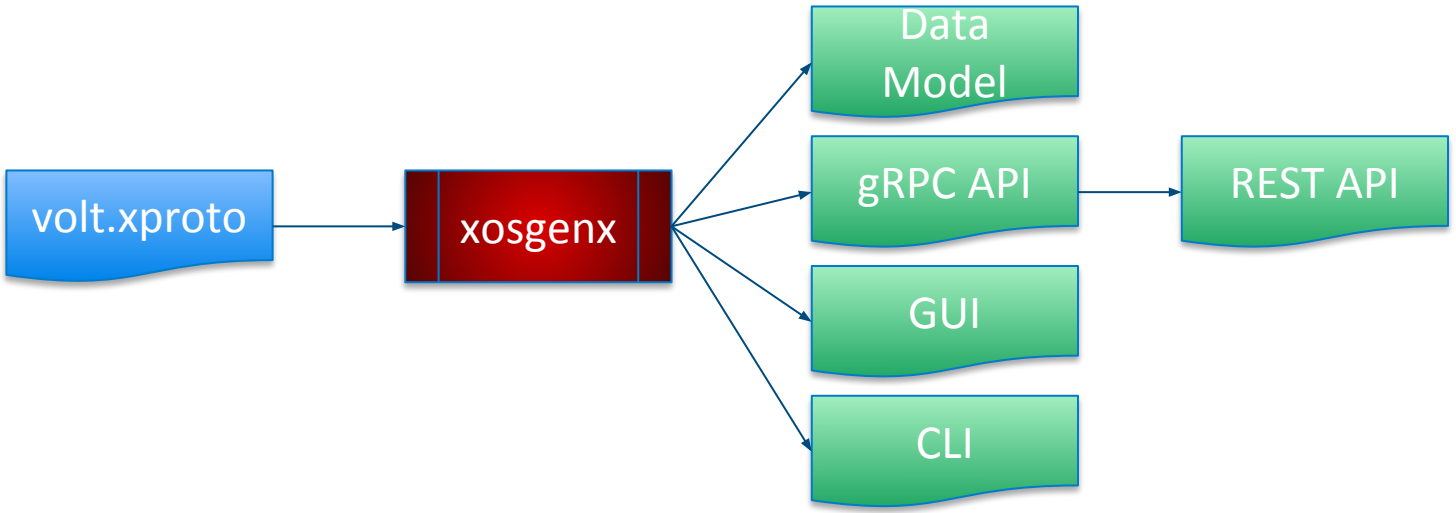
Data Model Example (VOLT Service)



Data Model Description Language (xproto)

"xproto", the XOS data modeling language

- Based on Google Protobuf, extended with relational features
- Used to autogenerate various targets (REST, GUI, etc)
- Make a change in one place, not six different places



xproto example

```
message ONUDevice (XOSBase){
   option verbose_name = "ONU Device";
   option description = "Represents a physical ONU device";

   required manytoone pon_port->PONPort:onu_devices = 1:1001 [];
   required string serial_number = 2 [max_length = 254, tosca_key=True, unique = True];
   required string vendor = 3 [max_length = 254];
   required string device_type = 4 [help_text = "Device Type", default = "asfvolt16_olt",
max_length = 254];

   optional string device_id = 5 [max_length = 254, feedback_state = True];
   optional string admin_state = 6 [choices = "(('DISABLED', 'DISABLED'), ('ENABLED', 'ENABLED'))",
default="ENABLED", help_text = "admin_state"];
   optional string oper_status = 7 [help_text = "oper_status", feedback_state = True];
   optional string connect_status = 8 [help_text = "connect_status", feedback_state = True];
}
```

The Data Model evolves over time

Service stack changes over time

- New services may be added
- Old services may be removed
- Existing services may be upgraded
  - New models
  - New fields to existing models

The data model will change over time

- New/updated xproto from the developers
- Service upgrades commanded by the operators
- How does the core handle this?

Data Model Migration

Migrate the *schema* - structural changes

- Add models or fields
- Delete models or fields
- Rename models or fields
- Change the type of a field

Migrate the *live data*

- Semantic transformation
- Example:
  - ModelV1 has fields (first_name, last_name)
  - ModelV2 has fields (name)
  - Someone has to implement: `v2.name = v1.first_name + " " + v1.last_name`

How XOS implements migration

We leverage Django's built-in migration support

- Uses python-based migration scripts
- Supports both schema and data migration

Developers write migration scripts by hand

- We wrote a tool called `xos-migrate` to do most of the work
- Developers only need to manually write the complex parts as necessary

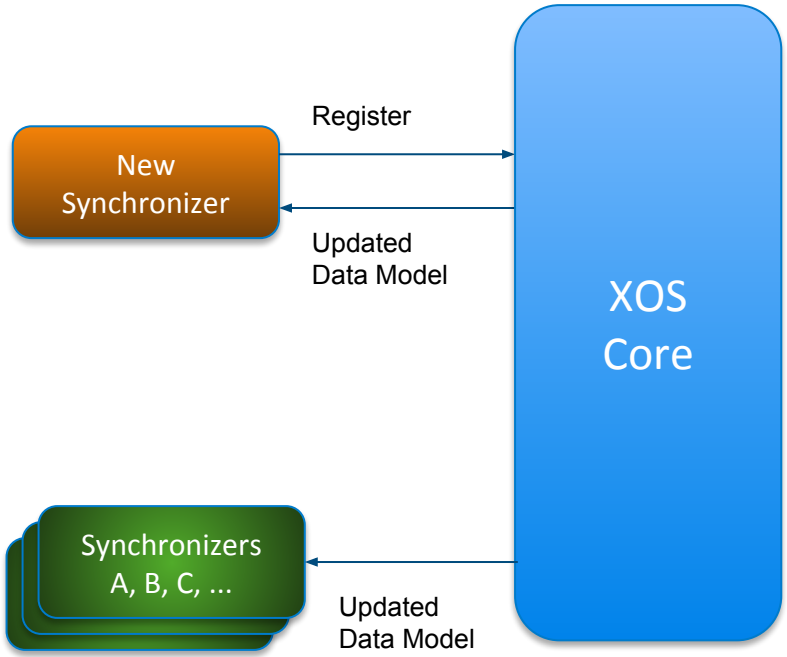
Upgrade is driven by Synchronizers

- Synchronizer supplies models and migration scripts
- XOS-core stops, runs migration scripts automatically, and restarts
- All synchronizers receive the new models

Service Lifecycle: Adding a Service

How do we bring up a new service?

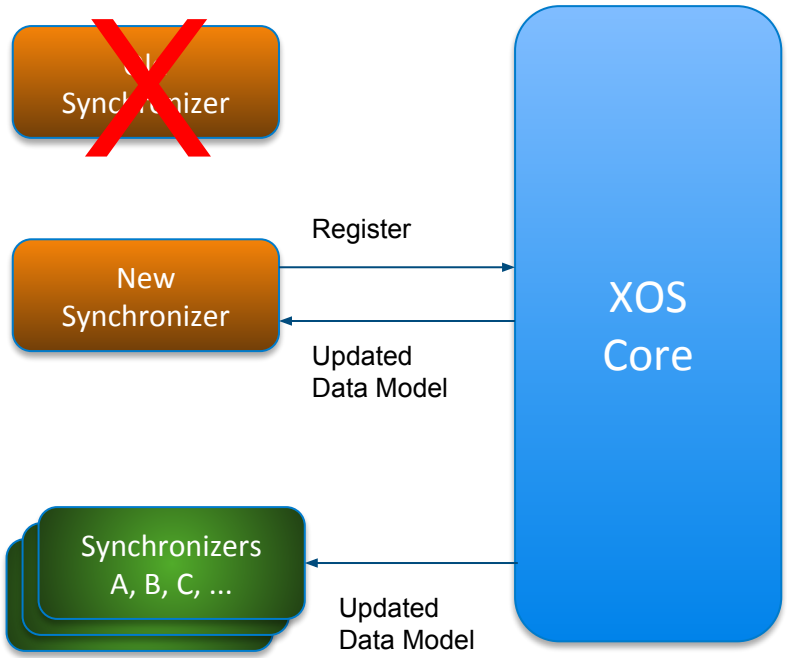
- Service deploys a new synchronizer
- Synchronizer registers service models and migrations with XOS core
- XOS core stops
- XOS core migrates the data model
- XOS core restarts
- XOS core pushes new data model to all synchronizers
- Synchronizers begin serving requests



Service Lifecycle: Upgrading a Service

How do we upgrade a service?

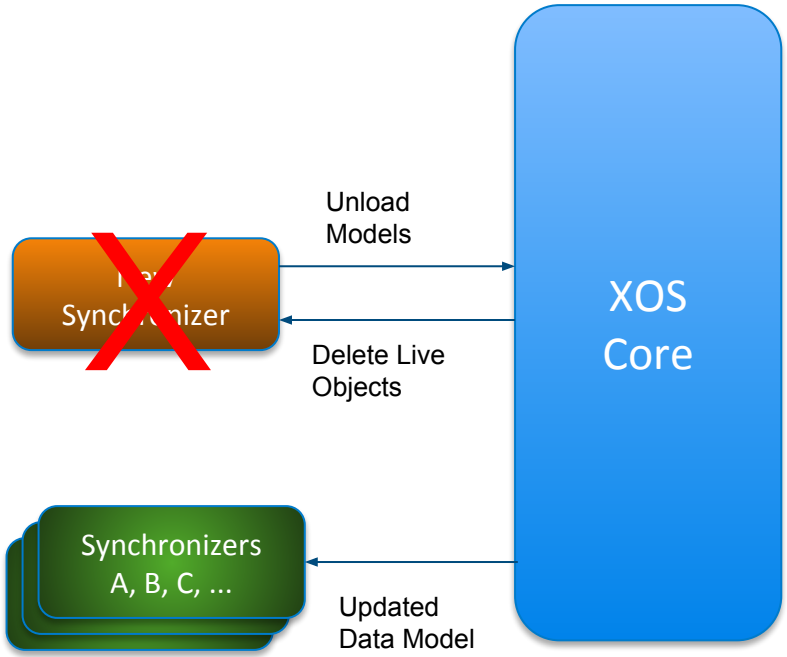
- Service destroys old synchronizer
- Service deploys a new synchronizer
- Synchronizer registers updated service models and migrations with XOS core
- XOS core stops
- XOS core migrates the data model
- XOS core restarts
- XOS core pushes new data model to all synchronizers
- Synchronizers begin serving requests



Service Lifecycle: Deleting a Service

How do we delete a service?

- Service calls “UnloadModels” API
- XOS deletes any live objects from data model
- Service destroys synchronizer
- XOS core stops
- XOS core migrates the data model
- XOS core restarts
- XOS core pushes new data model to all synchronizers
- Synchronizers begin serving requests



Leveraging Helm and Kubernetes

SEBA launches all its services in containers hosted on Kubernetes

Services are packaged into Helm Charts

Helm natively supports upgrades of charts:

```
$ helm upgrade att-workflow workflows/att-workflow \
       --set att-workflow-driver.image.tag=<new_version>
Release "att-workflow" has been upgraded.
```

Helm Upgrade process

The Helm side of the upgrade process is straightforward - terminate the old service pod, start the new one

```
==> v1/Pod(related)
NAME                                                READY   STATUS        RESTARTS   AGE
att-workflow-att-workflow-driver-6cdb76cbc9-qxsbp   1/1     Running       0          80s
att-workflow-att-workflow-driver-db478b467-fpxh6    0/1     Terminating   0          23m
```

XOS handles the internal details of the service upgrade

"Boring and uneventful" is **ideal** in this case.

Live Demo - Cordctl

We’re going to conclude with a demo of `cordctl`

- CLI tool for managing the NEM via XOS
- List the service inventory
- Inspect/create/update/delete models
- Perform backup/restore

Demo script 1 - Help and Service Inventory

```
# show the config file
cat ~/.cord/config
# retrieve server version / ensure connectivity
cordctl version
# show brief help
cordctl
# show detailed help
cordctl -h
# view the service inventory
cordctl service list
```

Demo script 2 - Inspecting Models

```
# List types of models
cordctl modeltype list
# List user models
cordctl model list User
# List users in a more concise format
cordctl model list User --format="table{{.id}}\t{{.email}}"
# Create a user
cordctl model create User --set-field firstname=John,lastname=Doe,email=john@doe.com,site_id=1
# Update a user
cordctl model update User 2 --set-field phone=111-222-3333
# Show our update was successful
cordctl model list User --format="table{{.id}}\t{{.email}}\t{{.phone}}"
# Delete user
cordctl model delete User 2

# Tour of the data model -- If time permits, show some other interesting models
cordctl model list RCORDSubscriber --format="table{{.id}}\t{{.name}}\t{{.c_tag}}\t{{.s_tag}}\t{{.onu_device}}"
cordctl model list ONUDevice
cordctl model list OLTDevice --format="table{{.id}}\t{{.dp_id}}\t{{.serial_number}}\t{{.host}}\t{{.admin_state}}"
cordctl model list AttWorkflowDriverServiceInstance
```

Demo script 3 - Backup and Restore

```
# Create a backup
cordctl backup create mybackup.raw
# Restore a backup
cordctl backup restore mybackup.raw
```