On-boarding Services: Developing Synchronizers

Sapan Bhatia, sapan@opennetworking.org
Scott Baker, scottb@opennetworking.org

CORD Build Nov. 7-9, 2017

An Operator Led Consortium


Goals of this Talk

- “Having modeled my service, how do I make it functional?”
- “What is a Synchronizer? How do I develop one?”
- “How do I follow best practices to produce a robust CORD service?”
- “What are some interesting and important problems in this space?”

Goals of this Talk

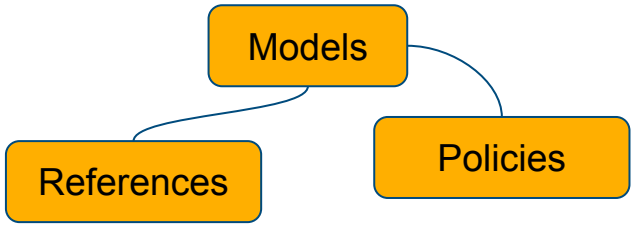
- “Having modeled my service, how do I make it functional?”
- “What is a Synchronizer? How do I develop one?”
- “How do I follow best practices to produce a robust CORD service?”
- “What are some interesting and important problems in this space?”

Want to take on a challenging problem?

- Apply to participate in a brigade
  - Leading is a great way to apply your expertise to a real problem
  - Enlisting is a great way of building expertise in an area
- Find me for a chat: sapan-cord-build.youcanbook.me
- Or anyone on the platform team: Andy Bavier, Scott Baker, Matteo Scandolo, Larry Peterson, Luca Prete, Gopi Taget, Zack Williams

The Big Picture

XOS Data Modeling Abstractions



XOS APIs

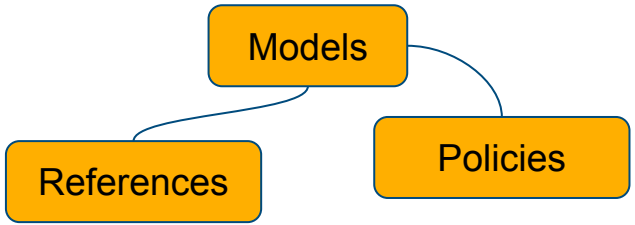
GRPC
TOSCA
DJANGO

Docker Containers
Docker Containers
OpenStack VM

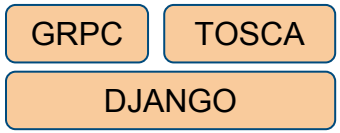
VNF mechanisms

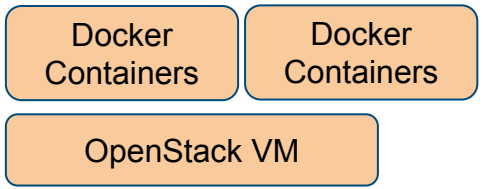
The Big Picture

XOS Data Modeling Abstractions
(Technology-agnostic)



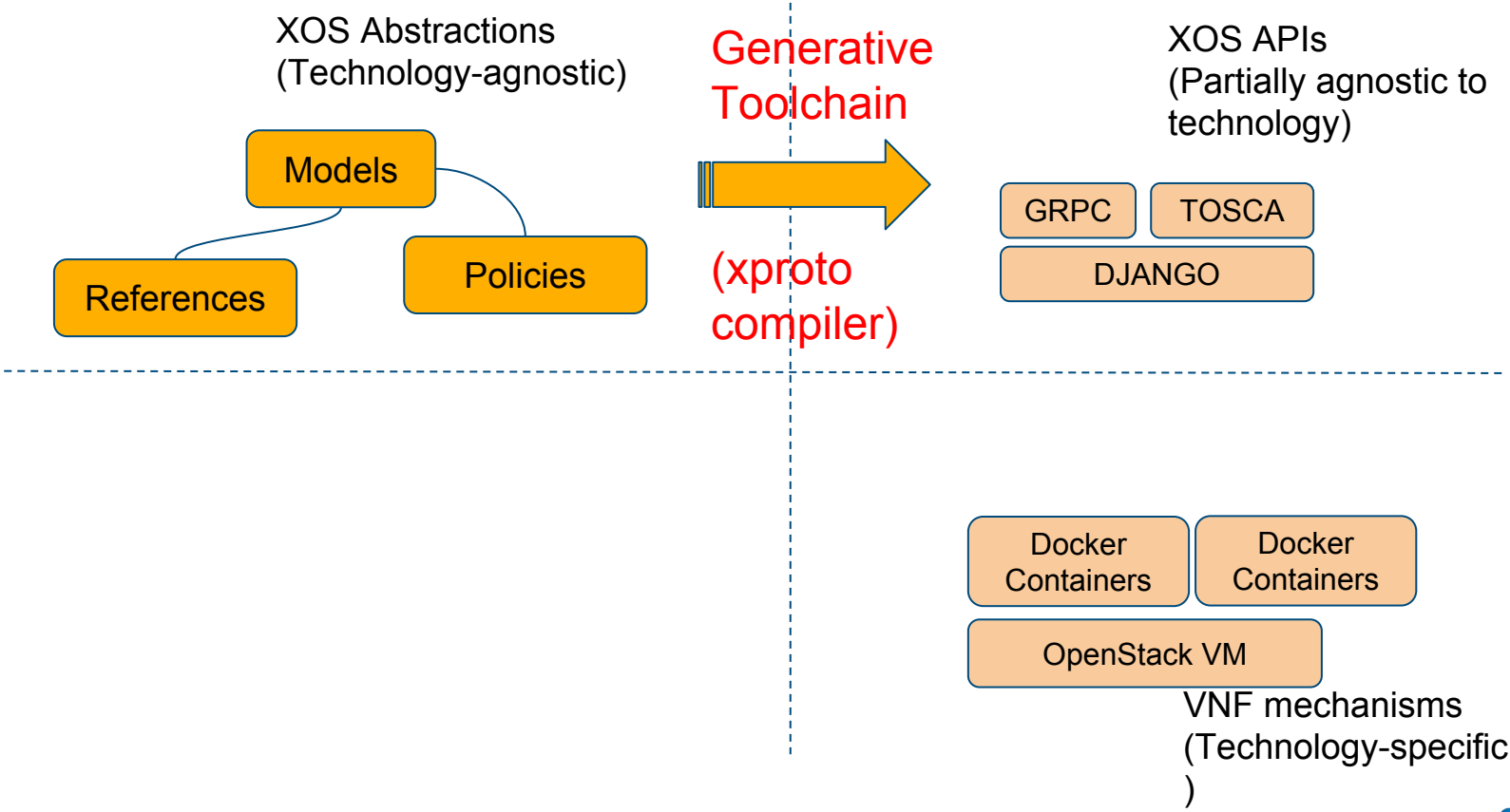
XOS APIs
(Partially agnostic to technology)



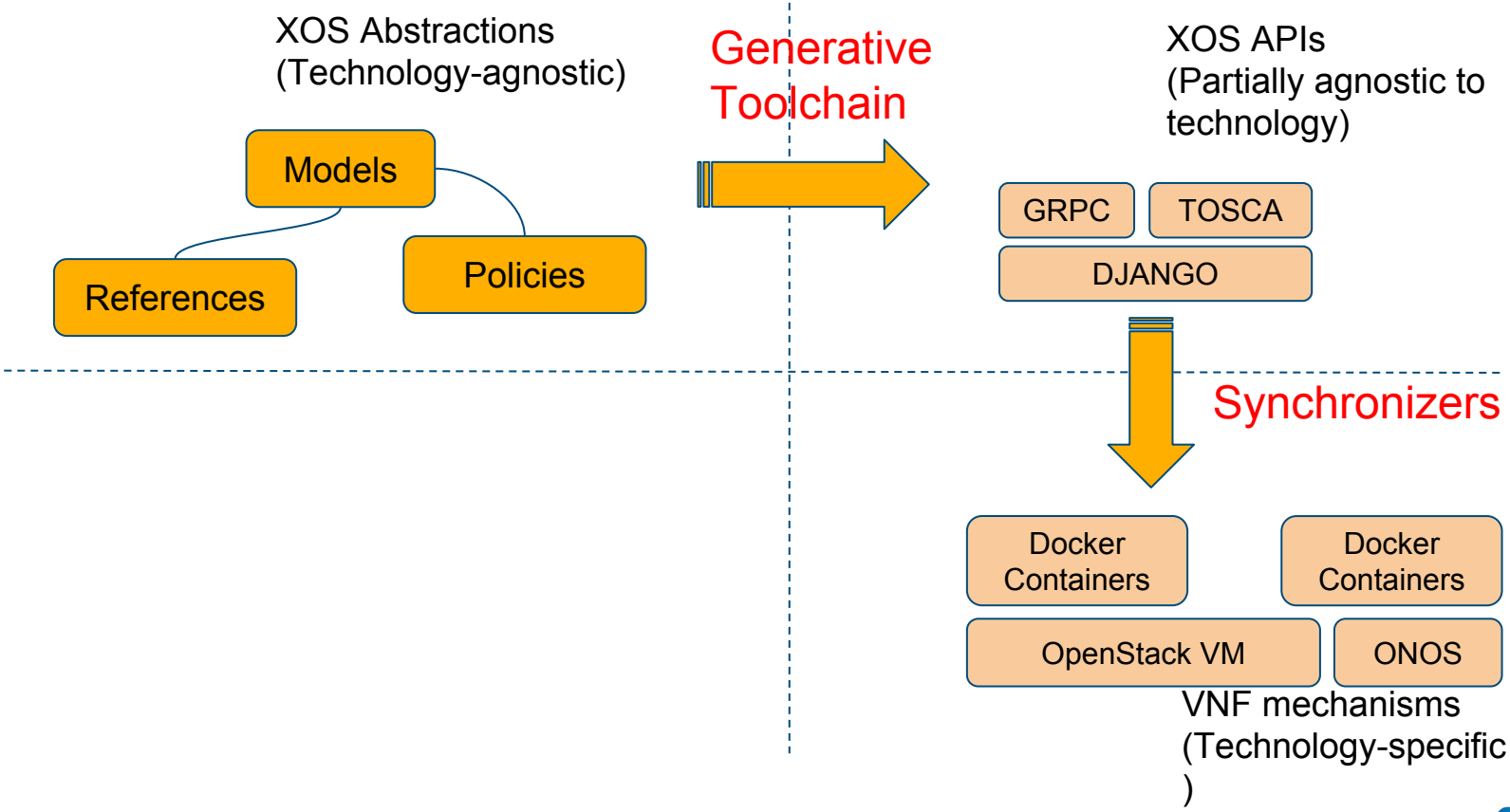


VNF mechanisms
(Technology-specific)

The Big Picture



The Big Picture

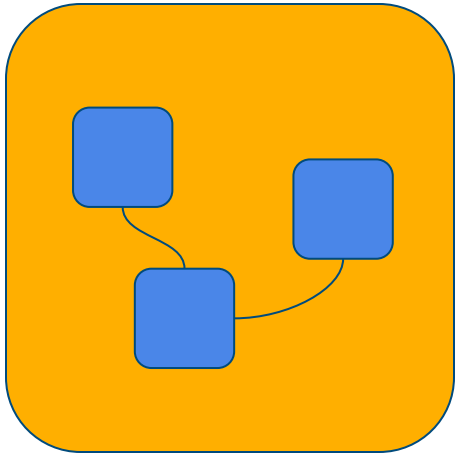


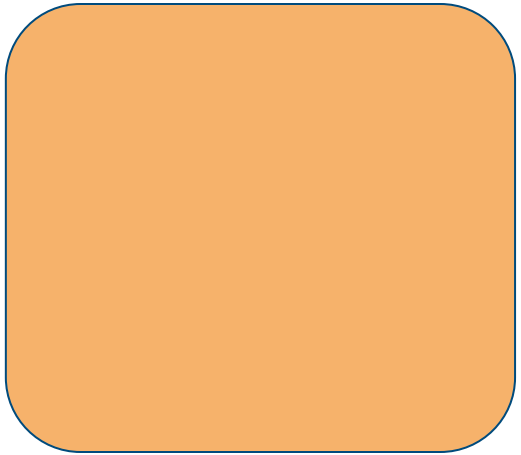
Key features of Synchronizers

- Goal-driven rather than message-driven
- Synchronizers are robust to errors
- Dependencies mirror data model
- Designed to help maximize scale up

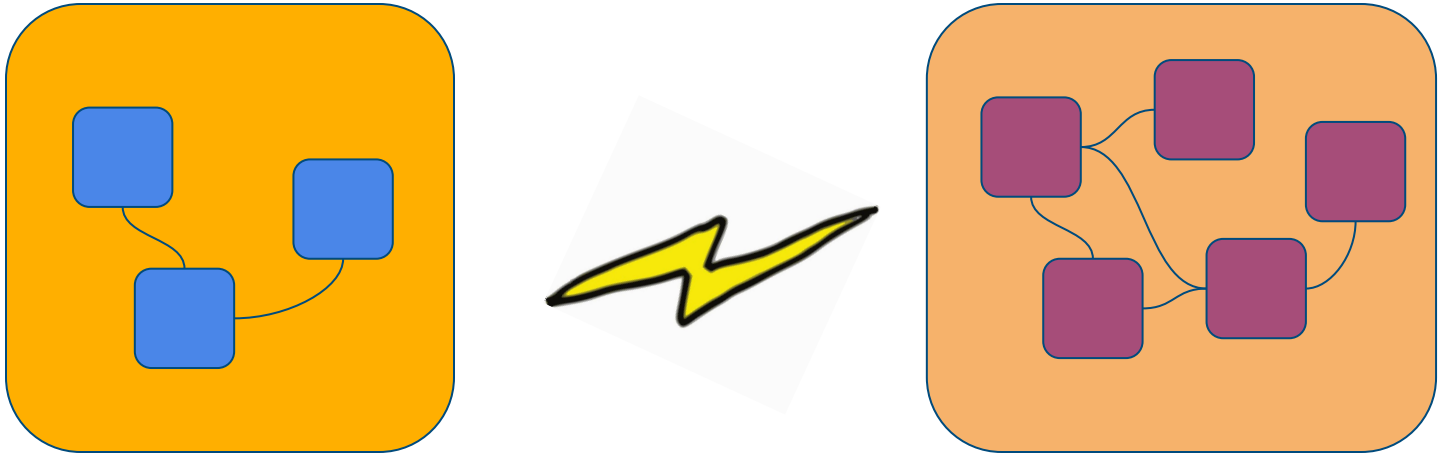
Goal-driven Synchronization

Goal-driven Synchronization

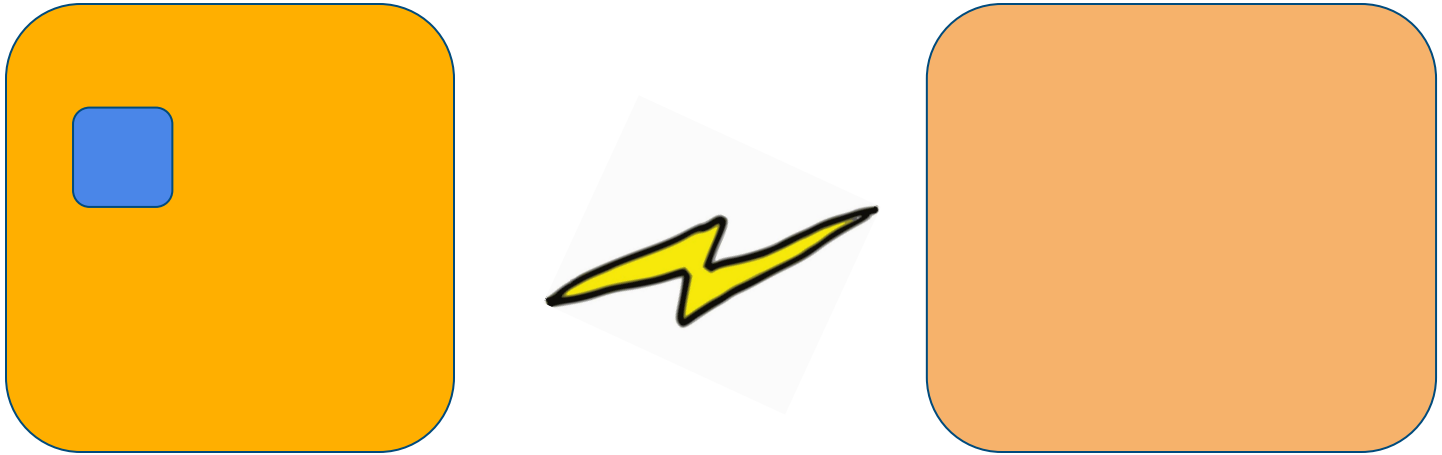




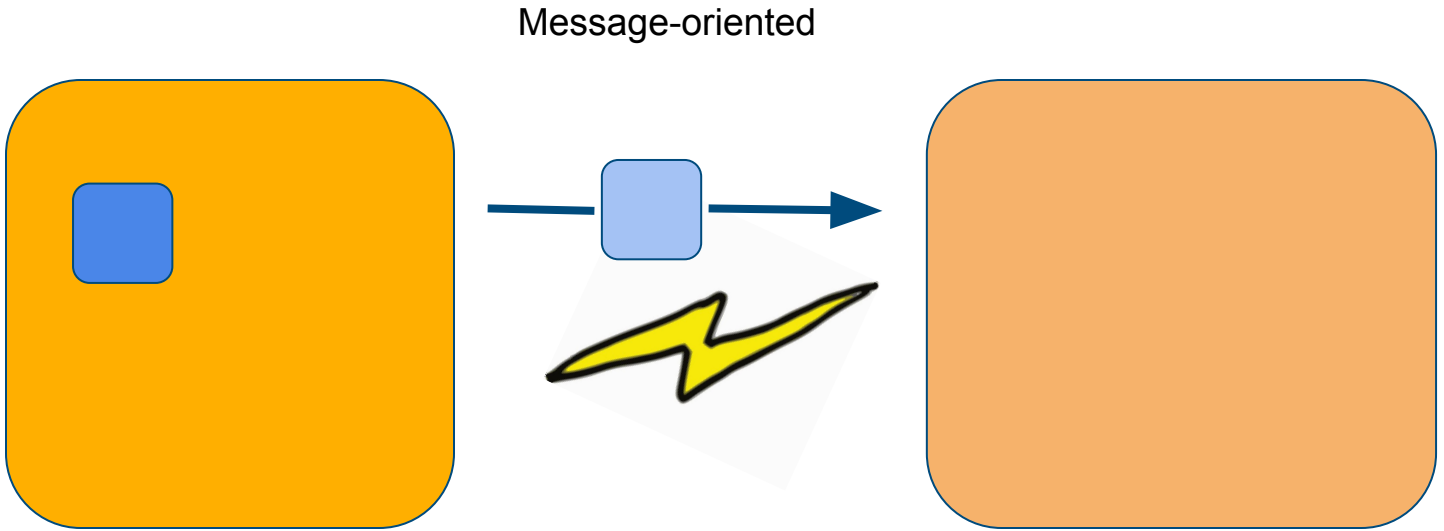
Goal-driven Synchronization



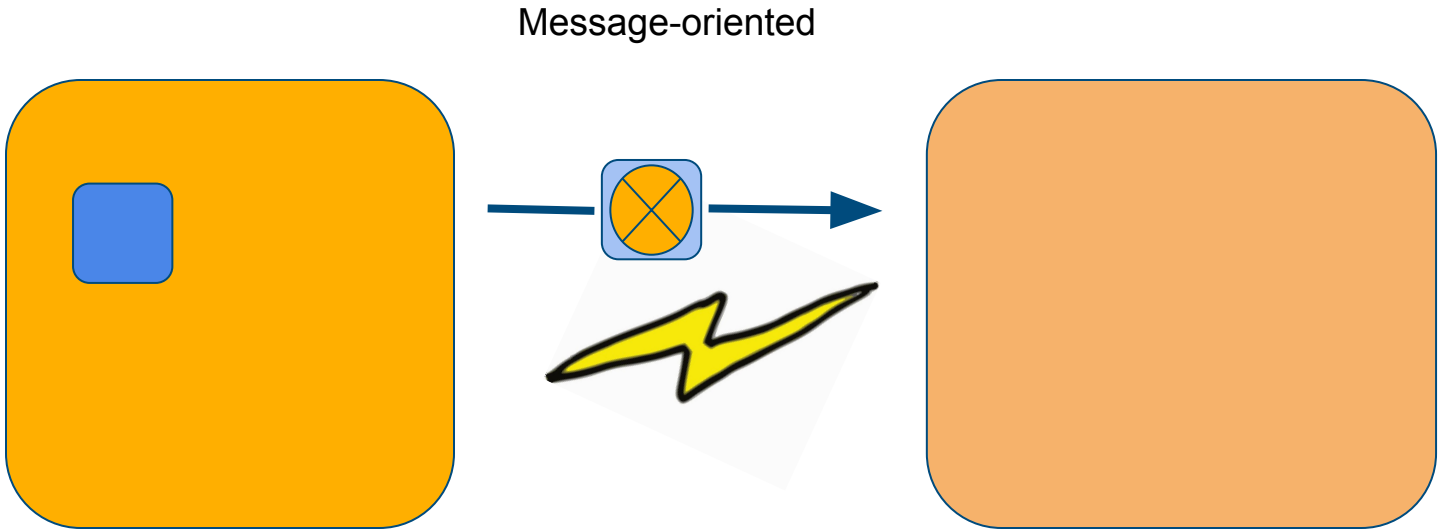
How would message-driven synchronization work?



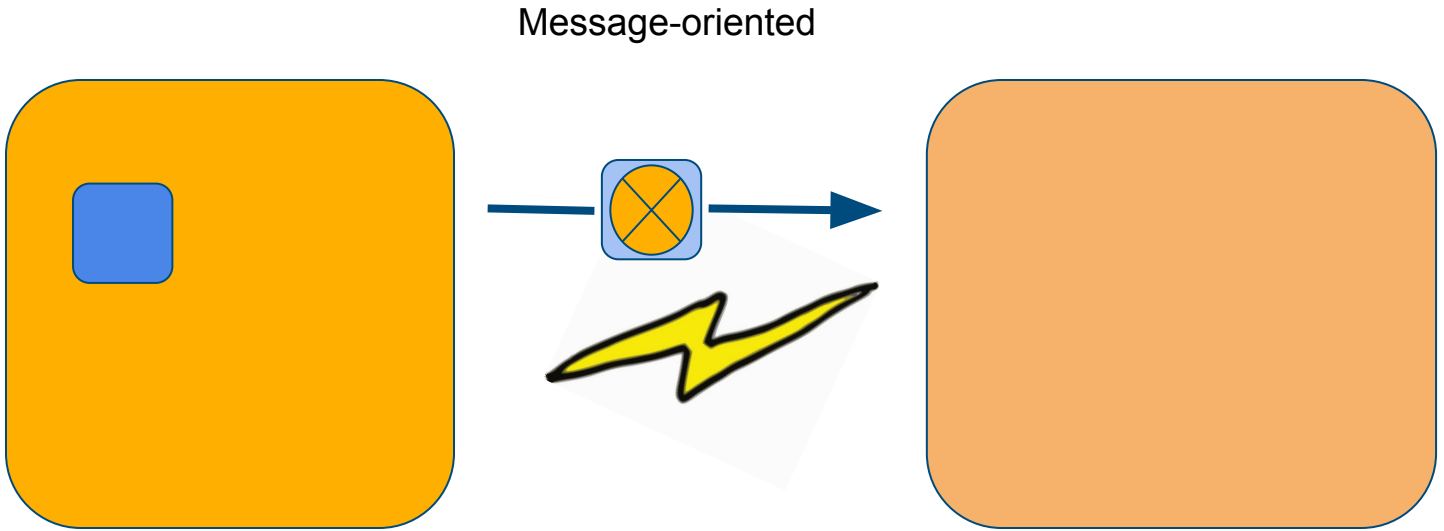
How would message-driven synchronization work?



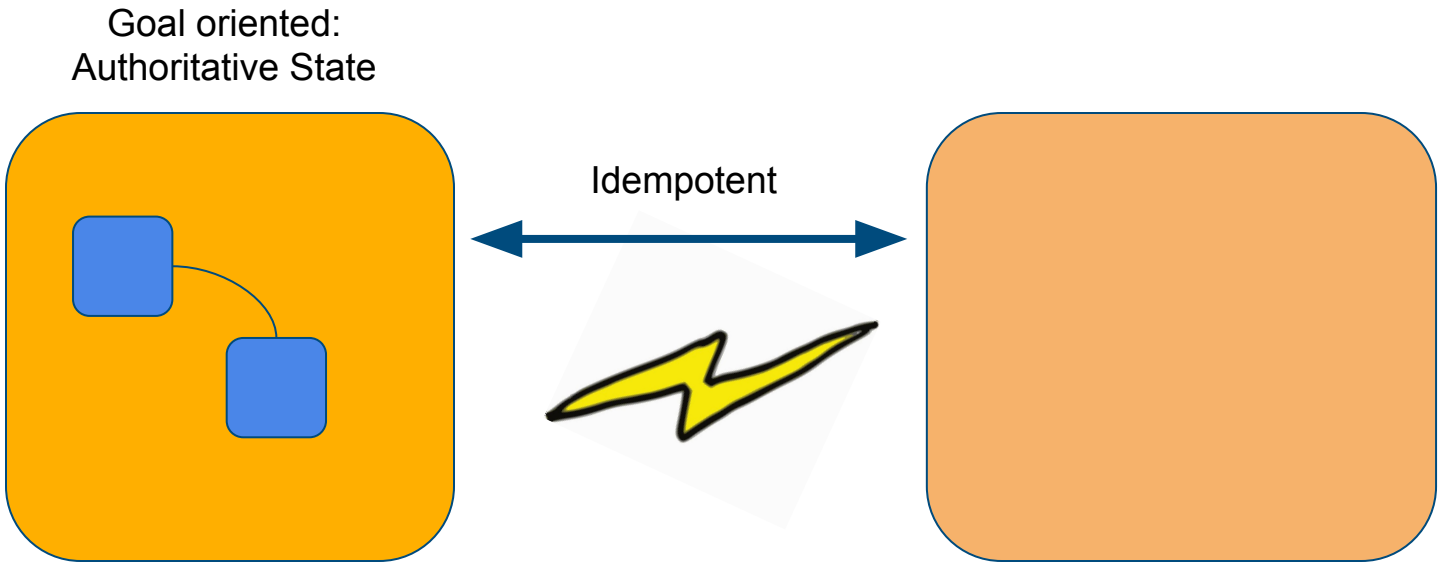
How would message-driven synchronization work?



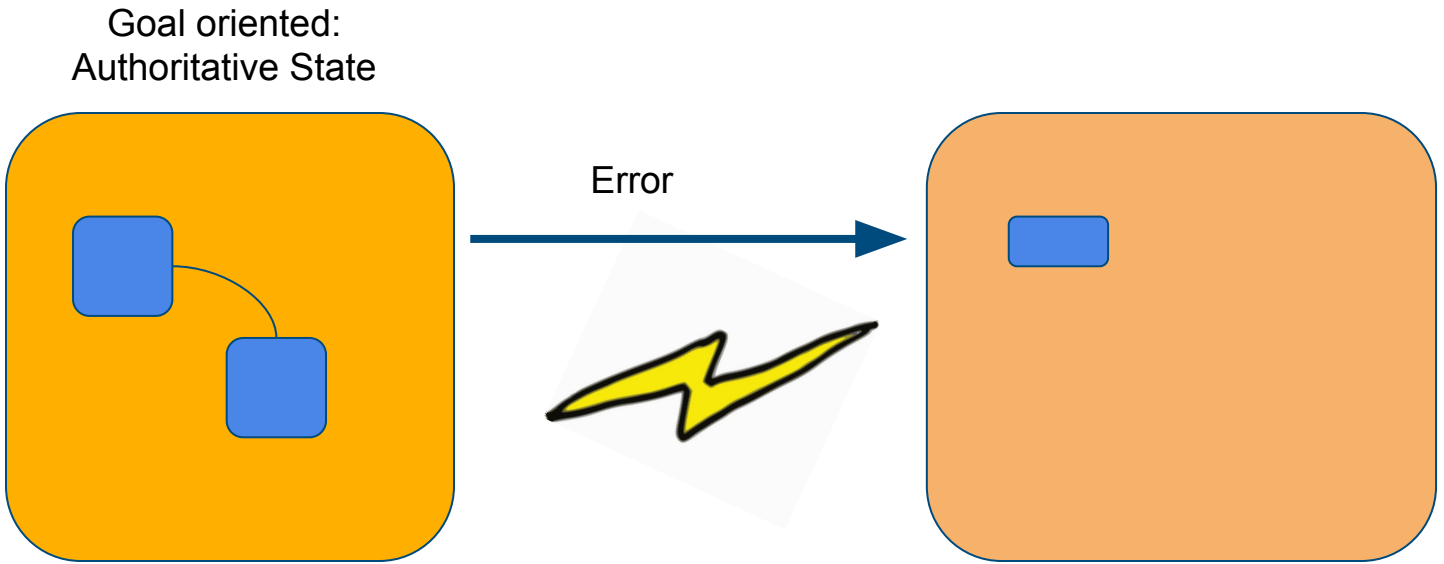
How would message-driven synchronization work?



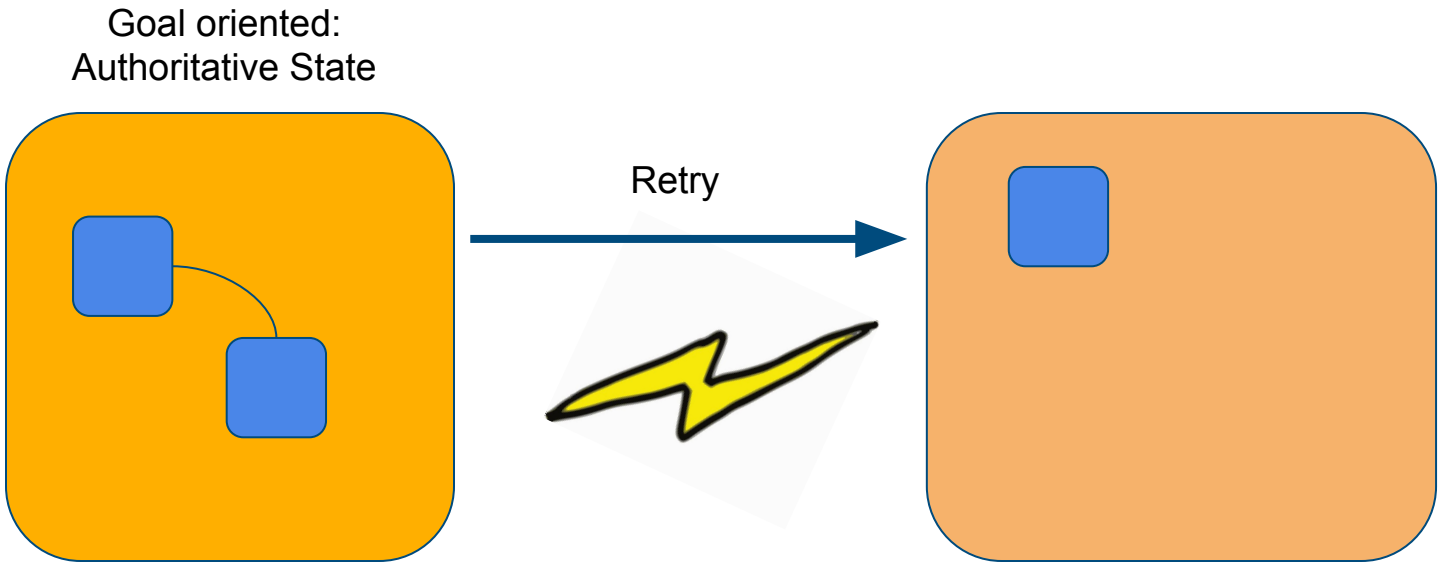
Goal-driven Synchronization



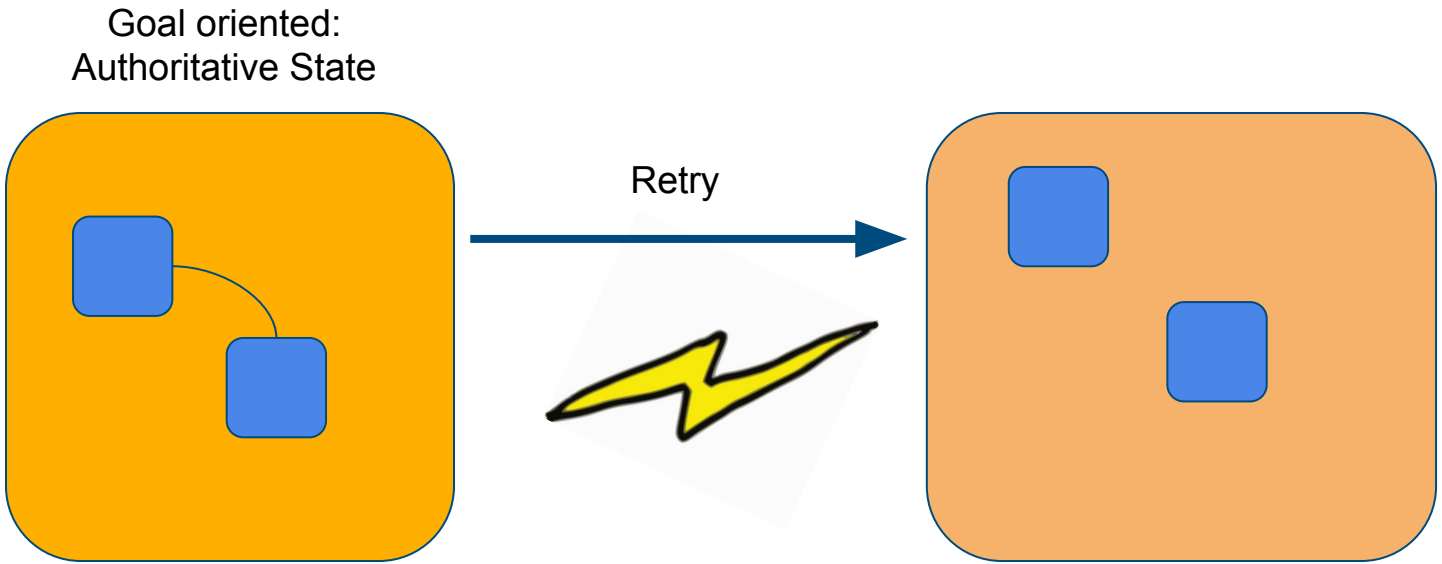
Goal-driven Synchronization



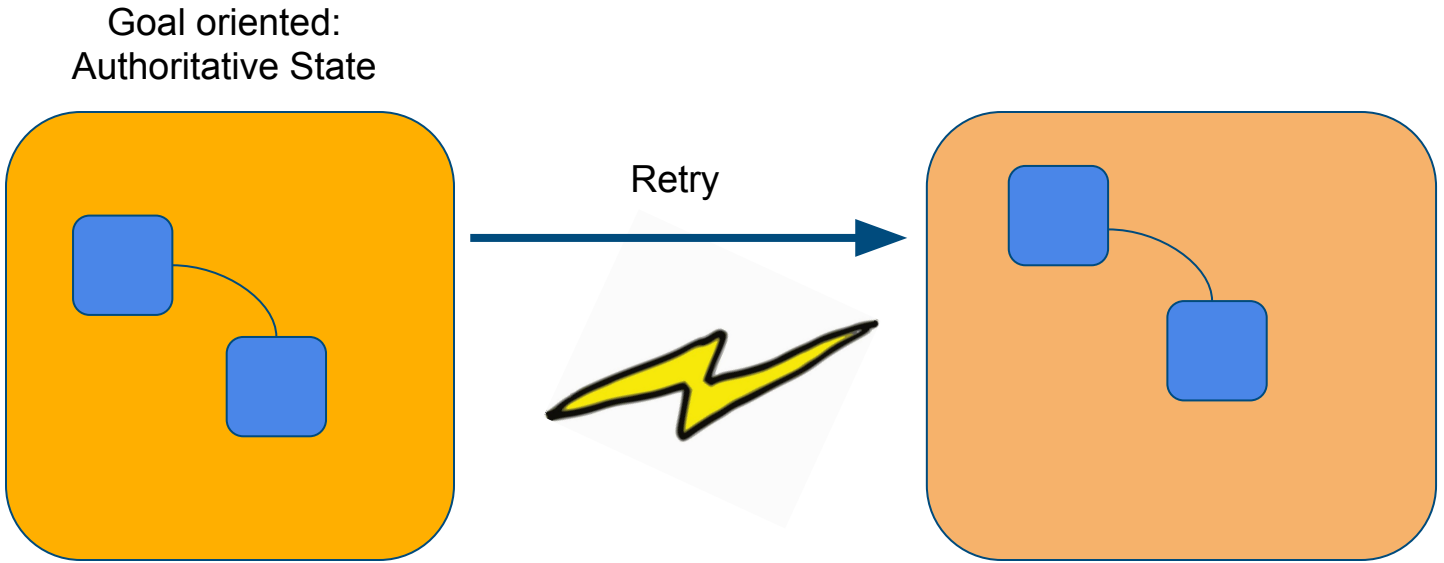
Goal-driven Synchronization



Goal-driven Synchronization



Goal-driven Synchronization



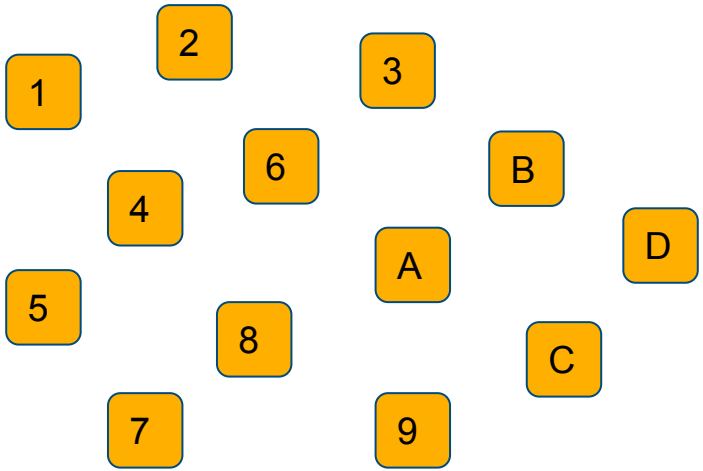
Synchronizer dependencies mirror data model

- Objects are guaranteed to be synchronized in dependency order
- Synchronizers are agnostic to the type of dependencies
  - Static dependencies between models
  - Dynamic dependencies between service instances
- Dependencies are fine-grained
  - Between objects, not models
- Dependencies are conservative
  - If you cannot evaluate a dependency, one is assumed

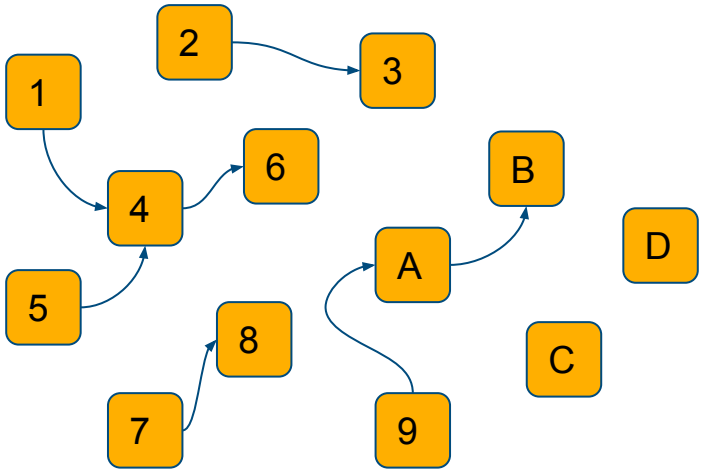
Designed for scale up

- Two parts to scaling up
  - Divide work into independently schedulable units
  - Dispatch units in contexts that run concurrently

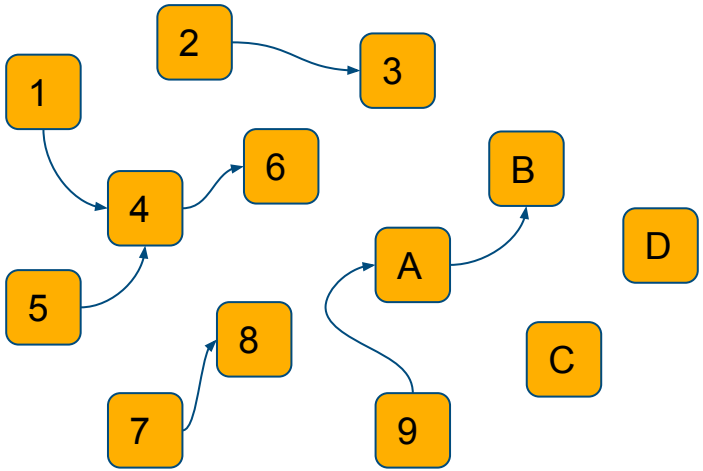
Designed for scale-up



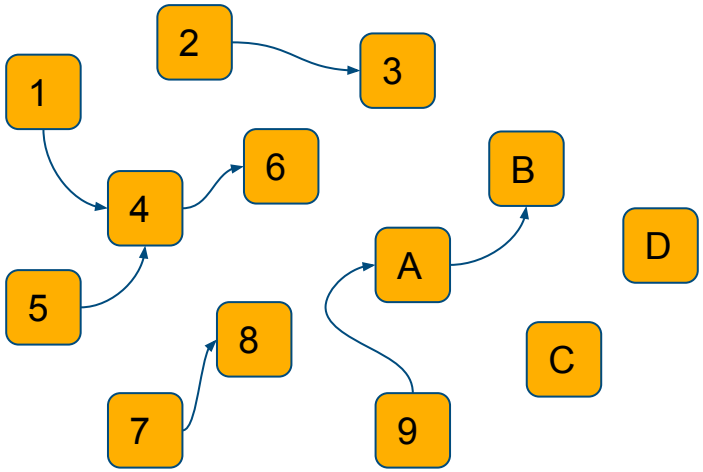
1. Extract dependencies

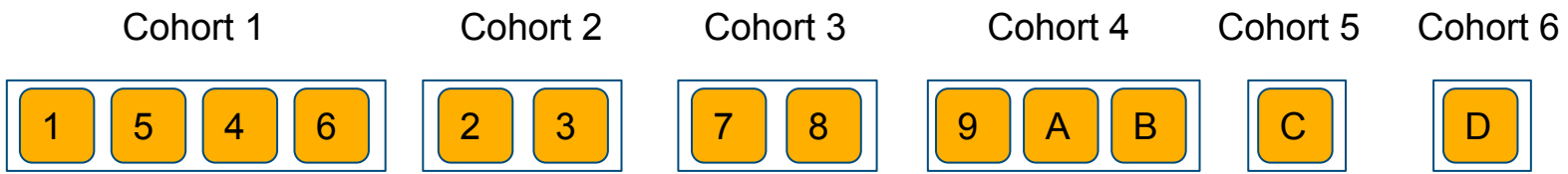


2. Connected components + Topological Sort

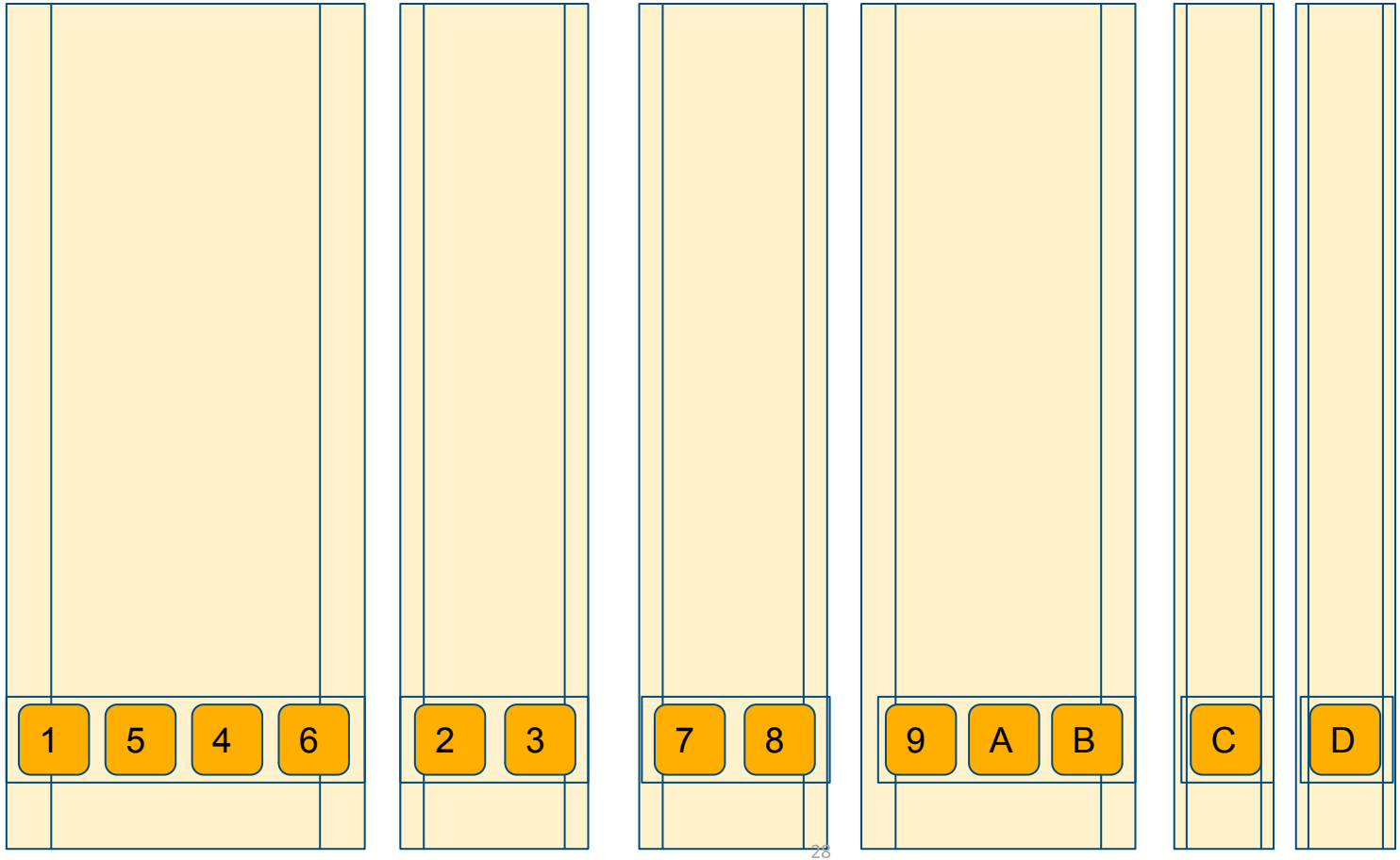


3. Cohorts

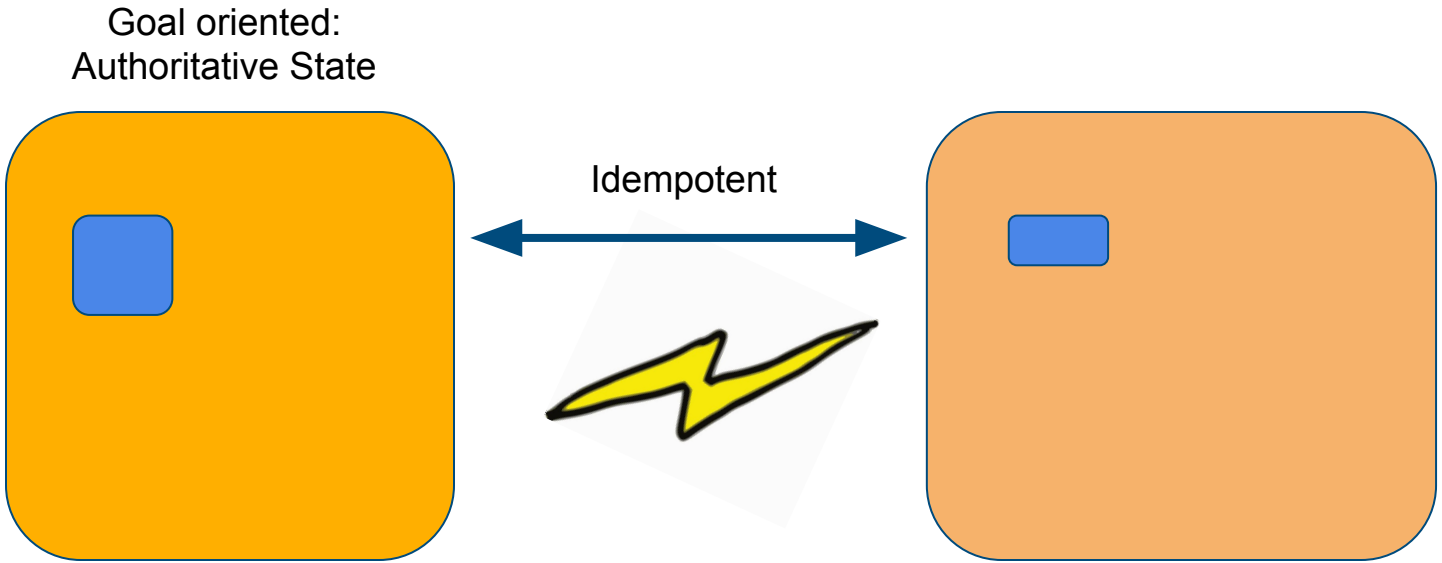

1
2
3
4
5
6
7
8
9
A
B
C
D


Cohort 1
1
5
4
6
Cohort 2
2
3
Cohort 3
7
8
Cohort 4
9
A
B
Cohort 5
C
Cohort 6
D

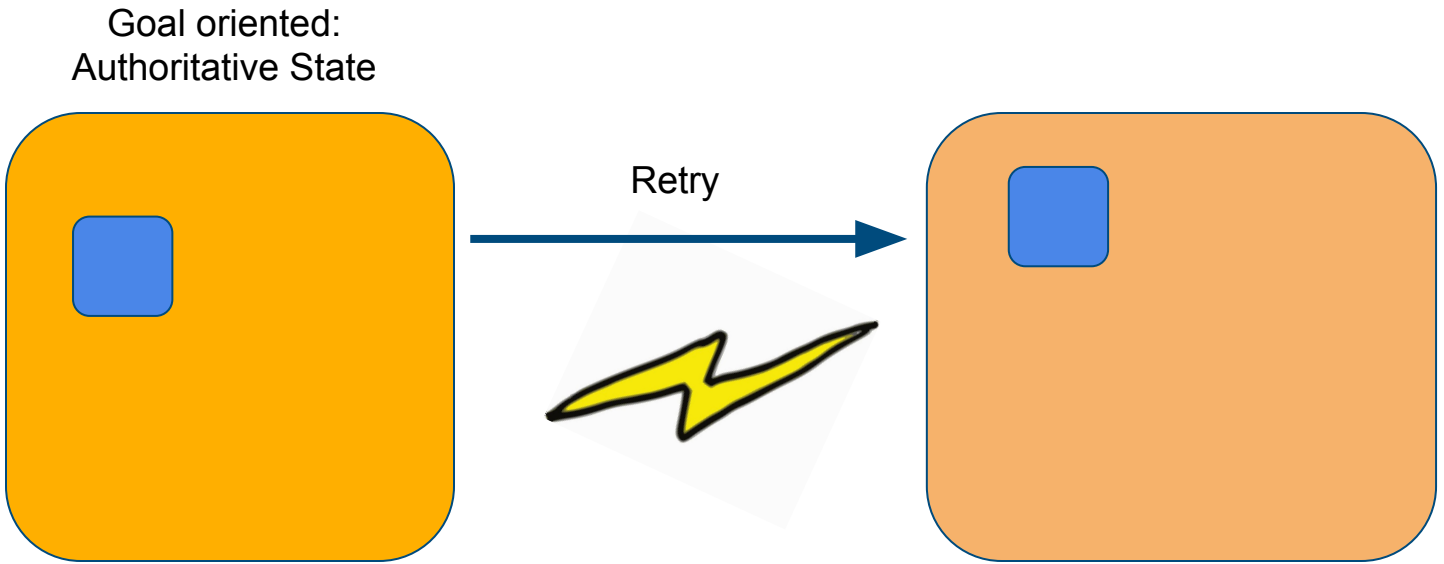
3. Schedule. Currently: Threads



Robustness to errors



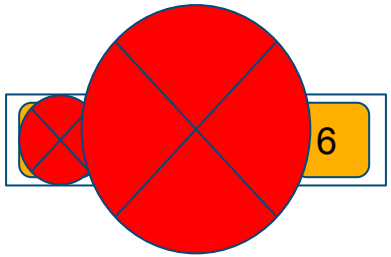
Robustness to errors



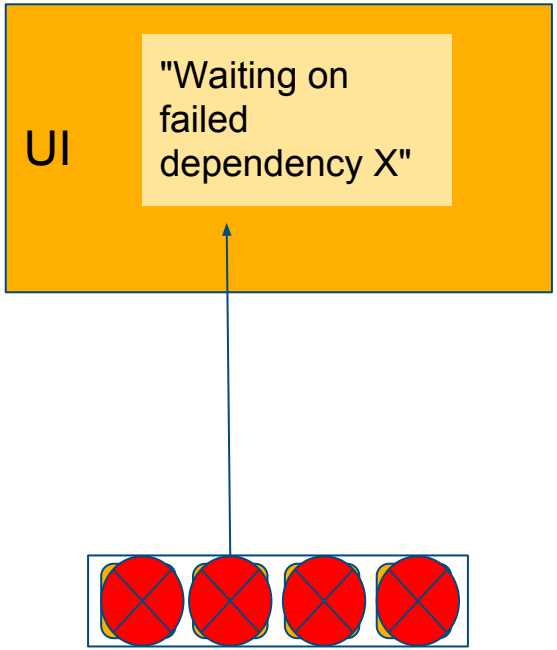
Robustness to errors

Robustness to errors: Error propagation

Robustness to errors: Error propagation



Robustness to errors: Error propagation

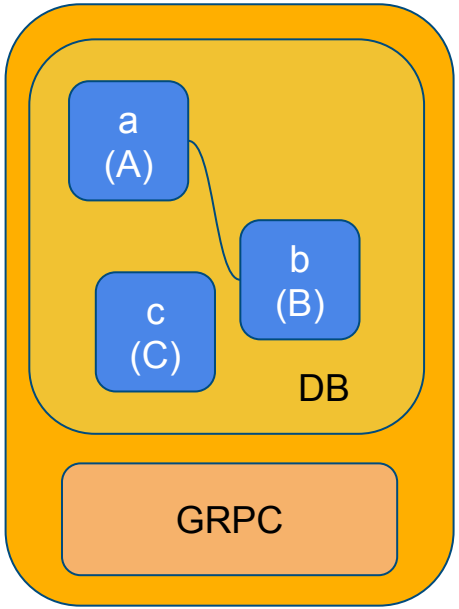


Errors reported in ELK Stack via structured logging

- error: Instance did not get IP address
- model: Instance
- Id: 7
- synchronizer_name: OpenStack
- sync_step: SyncPorts
- ansible_playbook: ...

But you only get these benefits if you follow best practices.

Synchronizer flow: a bird's-eye view



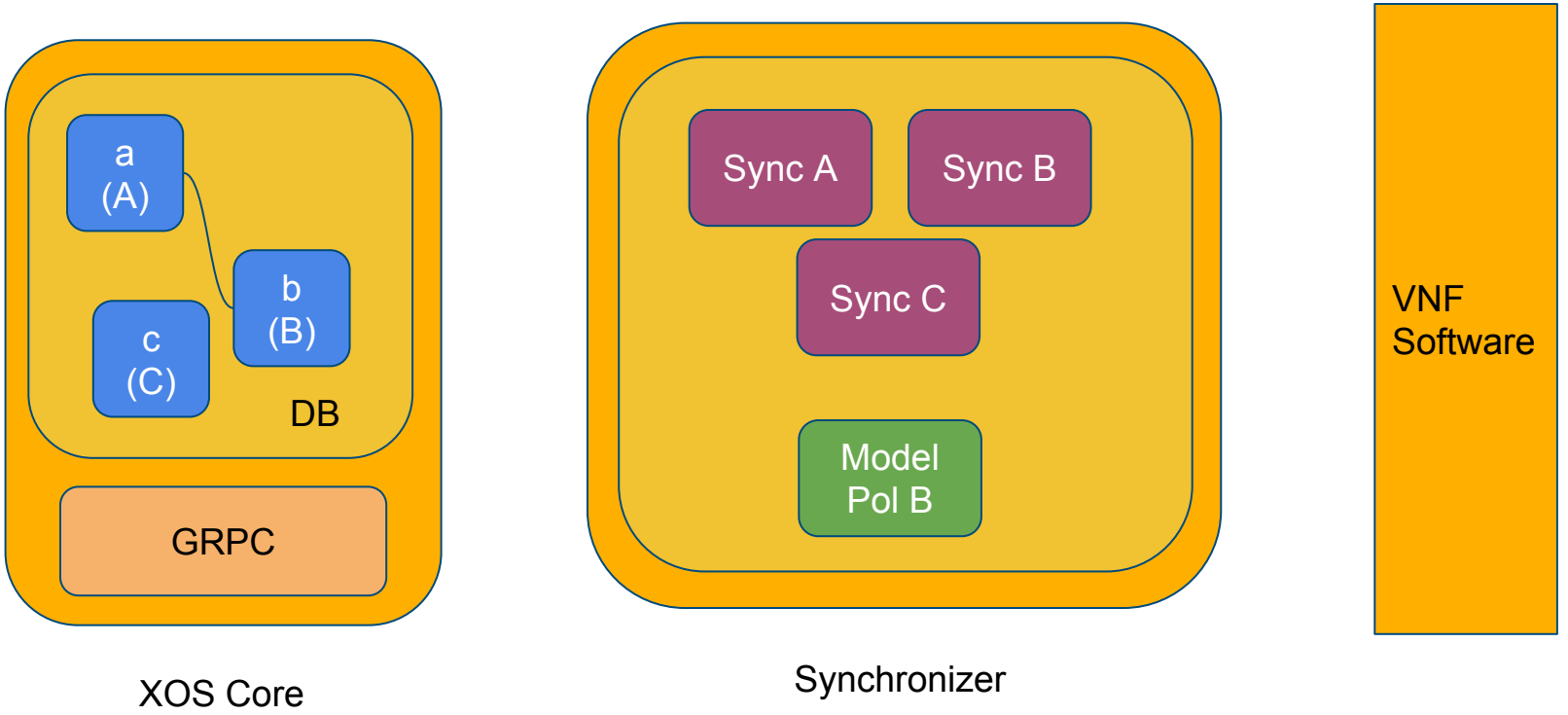
XOS Core

Three objects a, b and c have been created, of types models A, B and C

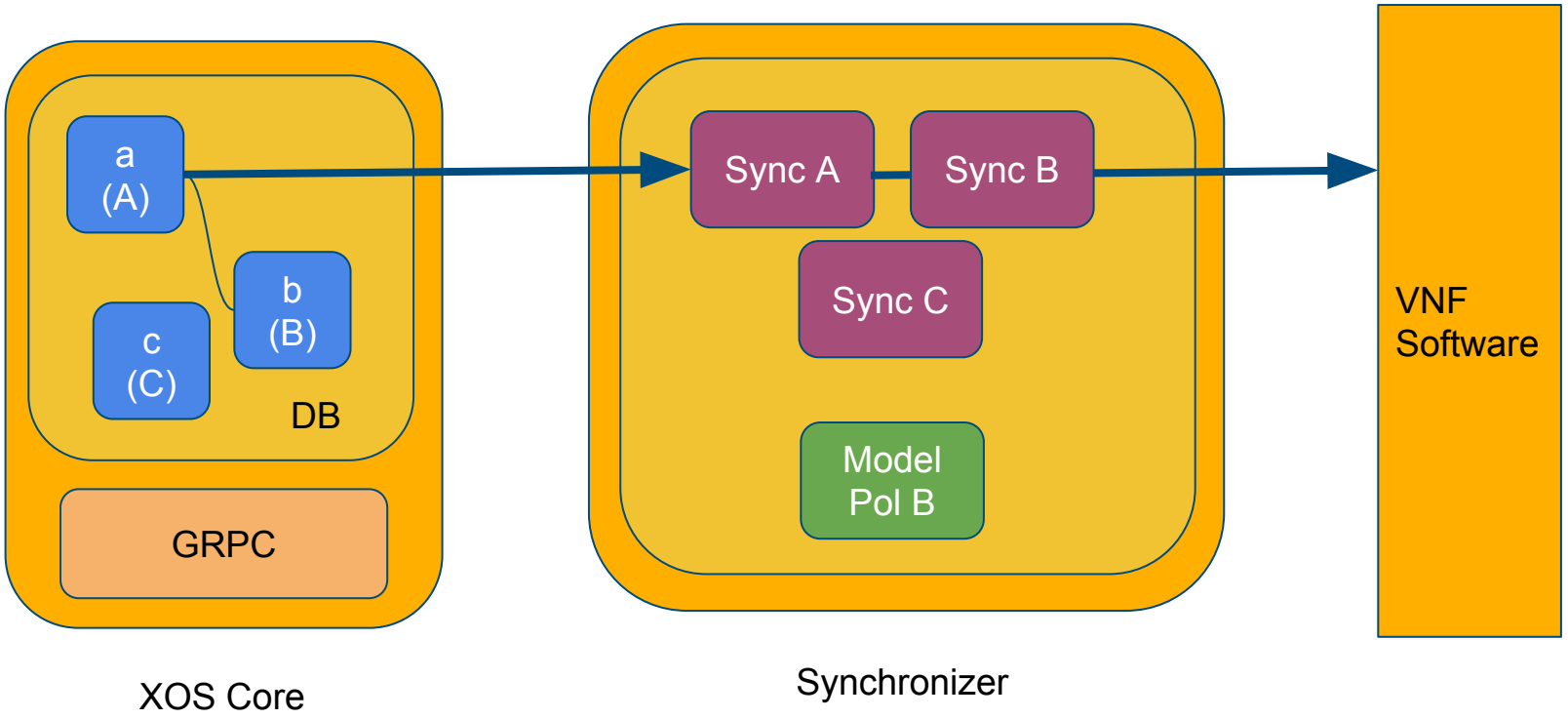
Objects consist of two parts:

Declarative state
Feedback state

Synchronizer flow: a bird's-eye view

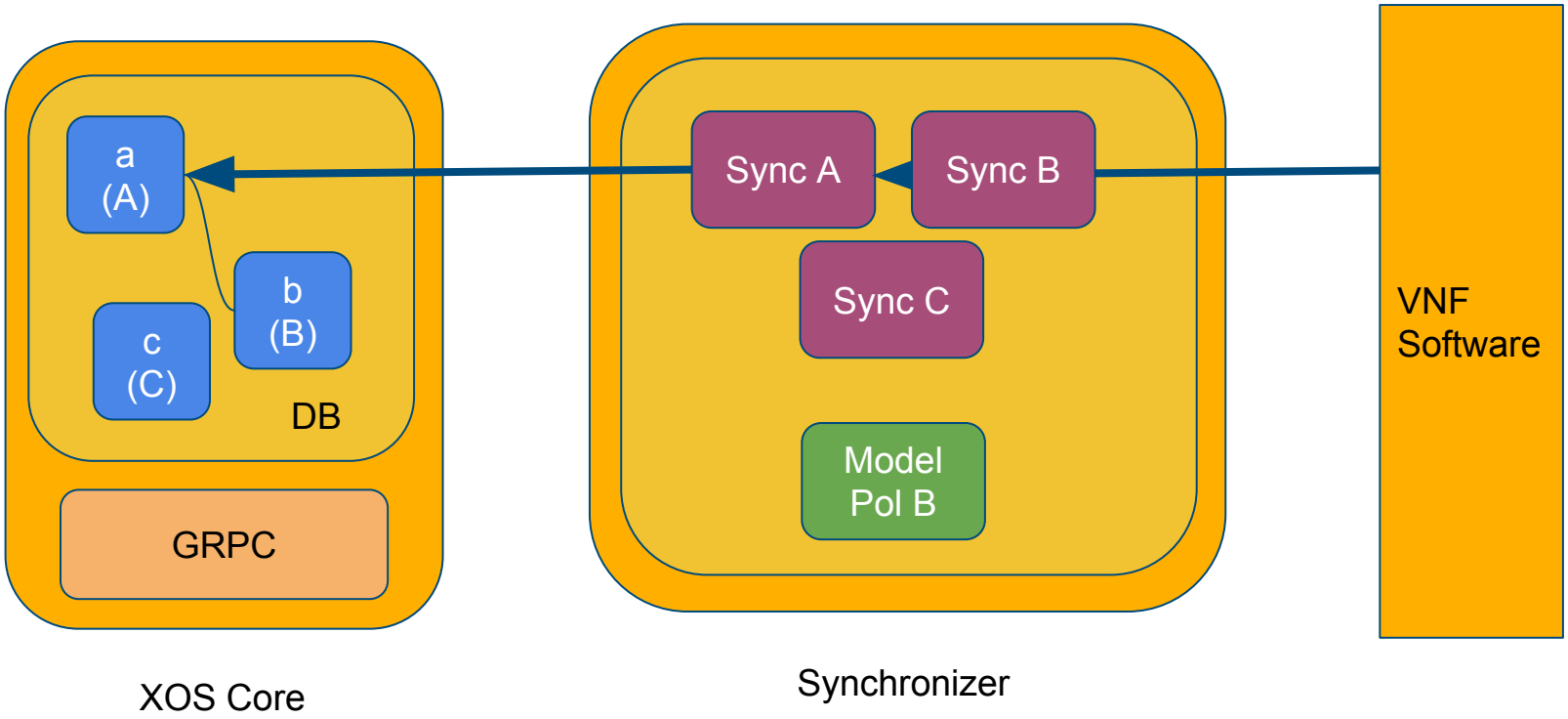


Synchronizer flow: a bird's-eye view



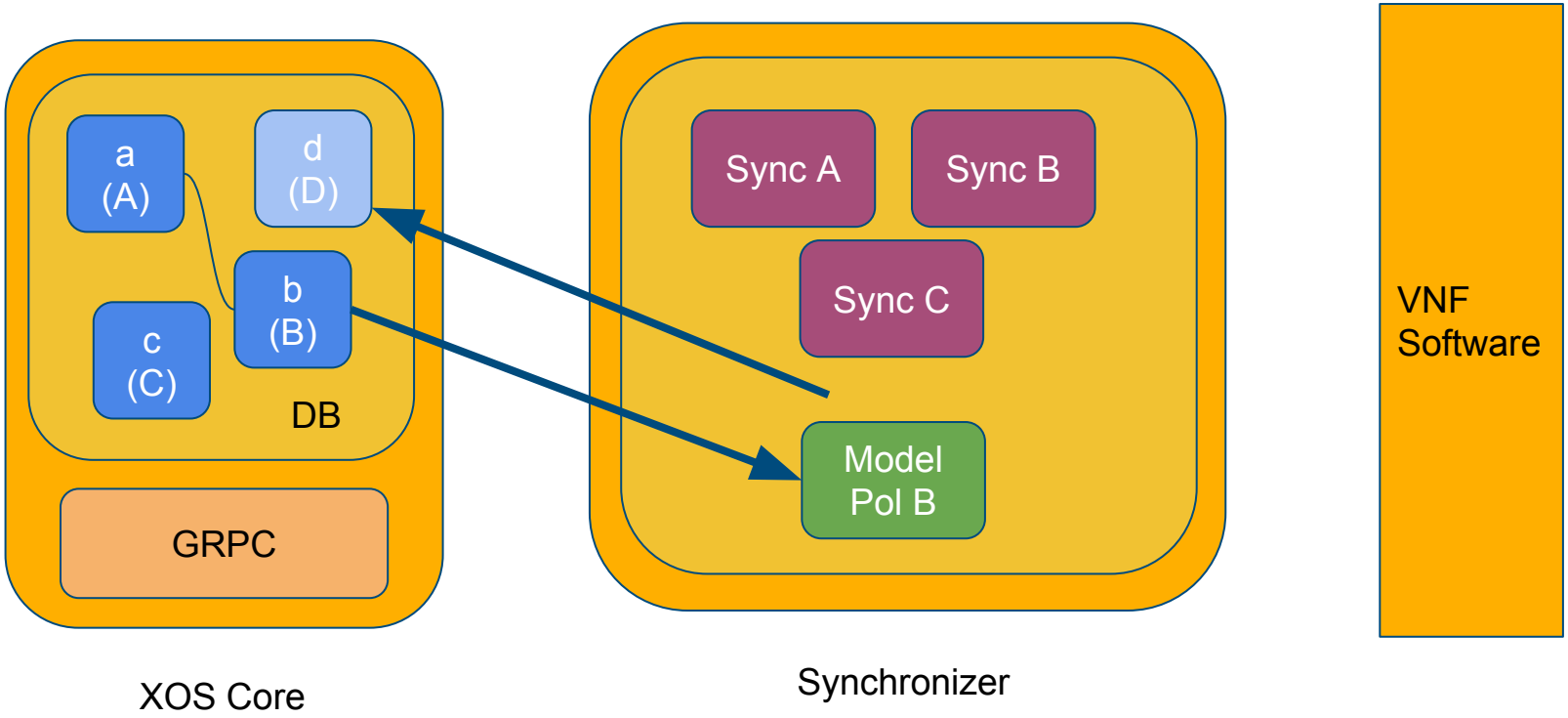
Translate declarative state

Synchronizer flow: a bird's-eye view



Transfer feedback state

Synchronizer flow: a bird's-eye view

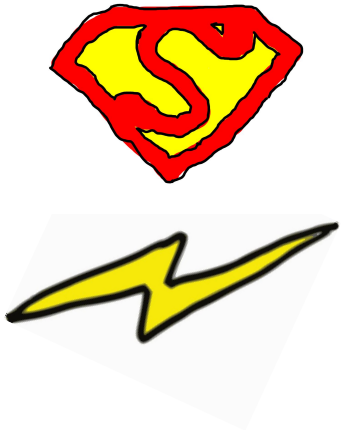


What is a Synchronizer made up of?

- Model policies
  - Configure the data model (add, delete, edit objects)
  - Can read/write declarative state
- Sync steps
  - Translate XOS state into VNF configuration
  - Can read declarative state and write feedback state
- Ansible playbook
  - Standard interface over which VNF configuration is propagated to VNF software
  - Ansible is not a requirement
- Boilerplate - Launch script, config file

Let's write a Synchronizer

XOS Core with Data Model



Back-end VNF API or Data Model

ExampleService ServiceInstance Model

```
message ExampleServiceInstance (TenantWithContainer){
     option verbose_name = "Example Service Instance";
     required string tenant_message = 1 [help_text = "Tenant Message to
Display", max_length = 254, null = False, db_index = False, blank = False];
     optional manytoone
foreground_color->Color:serviceinstance_foreground_colors = 2 [db_index =
True, null = True, blank = True];
     optional manytoone
background_color->Color:serviceinstance_background_colors = 3 [db_index =
True, null = True, blank = True];
}
```

Generate a synchronizer stub

```
xosgenx
    --target synchronizer.xtarget
    --output ./
    --write-to-file target
    exampleservice.xproto
```

Outcome of generation

Sync steps:

- sync_exampleservice.py
- sync_exampleserviceinstance.py
- sync_color.py
- sync_embedded_image.py

Model dependencies:

- model-deps.yaml

Outcome of generation

Model policies:

- model_policy_exampleservice.py
- model_policy_exampleserviceinstance.py
- model_policy_color.py
- model_policy_embedded_image.py

Note: There's no Ansible playbook here

model-deps.yaml

```
{
    "ExampleService": [
    ],
    "Color": [
    ],
    "ExampleServiceInstance": [
        ["Color", "foreground_color",
"serviceinstance_foreground_colors"],
        ["Color", "background_color",
"serviceinstance_background_colors"]
    ],
    "EmbeddedImage": [
        ["ExampleServiceInstance", "serviceinstance",
"embedded_images"]
    ]
}
```

model-deps.yaml

```
{
    "ExampleService": [
    ],
    "Color": [
    ],
    "ExampleServiceInstance":
        ["Color", "foreground_color",
"serviceinstance_foreground_colors"]
        ["Color", "background_color",
"serviceinstance_background_colors"]
    ],
    "EmbeddedImage": [
        ["ExampleServiceInstance", "serviceinstance",
"embedded_images"]
    ]
}
```

Used to compute the dynamic dependency graph

model-deps.yaml

(You can edit this file)

```
{
    "ExampleService" : [
    ],
    "Color": [
    ],
    "ExampleServiceInstance" : [
        ["Color", "foreground_color",
"serviceinstance_foreground_colors"],
        ["Color", "background_color",
"serviceinstance_background_colors"]
    ],
    "EmbeddedImage": [
        ["ExampleServiceInstance", "serviceinstance",
"embedded_images"]
    ]
}
```

sync_example_serviceinstance.py (stub)

```
class SyncExampleServiceInstance(SyncInstanceUsingAnsible):
    observes=ExampleServiceInstance
    service_key_name =
"/opt/xos/synchronizers/exampleservice/exampleservice_private_key"

    template_name = "sync_exampleserviceinstance.yaml"
```

sync_example_serviceinstance.py (stub)

```
def get_extra_attributes(self, o):
        fields = {
             "tenant_message": o.tenant_message,
             "foreground_color": o.foreground_color,
             "background_color": o.background_color
        }
        # TODO: Change the above map to map data model fields into
parameters in the Ansible playbook
        # Once you have done that, drop the line below

        raise Exception("Not implemented")

        return fields
```

sync_example_serviceinstance.py (stub)

```
def get_extra_attributes(self, o):
        fields = {}
        fields['tenant_message'] = o.tenant_message
        exampleservice =  self.get_exampleservice(o)
        fields['service_message'] = exampleservice.service_message
        if o.foreground_color:
            fields["foreground_color"] = o.foreground_color.html_code
        if o.background_color:
            fields["background_color"] = o.background_color.html_code
        images=[]
        for image in o.embedded_images.all():
            images.append({"name": image.name,
                           "url": image.url})
        fields["images"] = images
        return fields
```

Ansible Playbook

```
- hosts: "{{ instance_name }}"
  connection: ssh
  user: ubuntu
  sudo: yes
  gather_facts: no
  vars:
    - tenant_message: "{{ tenant_message }}"
    - service_message: "{{ service_message }}"
    - foreground_color: "{{ foreground_color }}"
    - background_color: "{{ background_color | default("#FFFFFF") }}"
    - images:
        {% for image in images %}
        - name: {{ image.name }}
          url: {{ image.url }}
        {% endfor %}

  roles:
    - install_apache
    - create_index
```

Ansible Playbook

```
- hosts: "{{ instance_name }}"
  connection: ssh
  user: ubuntu
  sudo: yes
  gather_facts: no
  vars:
    - tenant_message: "{{ tenant_message }}"
    - service_message: "{{ service_message }}"
    - foreground_color: "{{ foreground_color | default("#000000") }}"
    - background_color: "{{ background_color | default("#FFFFFF") }}"
    - images:
        {% for image in images %}
        - name: {{ image.name }}
          url: {{ image.url }}
        {% endfor %}

  roles:
    - install_apache
    - create_index
```

Error handling

- Upon encountering an error, simply raise an exception
- The error message propagates to the UI
- The Synchronizer retries, and continues to do so until it succeeds
- Exponential backoff can be configured in production environments
- Exceptions automatically block dependent objects

Logging

- XOS uses a logger called multistructlog
- Thin wrapper around Structlog, with Structlog interface
- Logs simultaneously to several backends: console, file, ELKStack
- Log context bound to the logger: data model object, Sync Step, ...
- Example of log statement:

```
except NoIPException, e:
    log.exception("Interface does not have IP", ip = ip_address, e = e)
    raise e
```

Notes about best practices

- Synchronizer steps must be idempotent
- Back-end resources must be identified via feedback state
  - Essential for cleanups
- Break up services into models at logical boundaries
  - Easier to maintain and observe in UI
  - Better parallelism

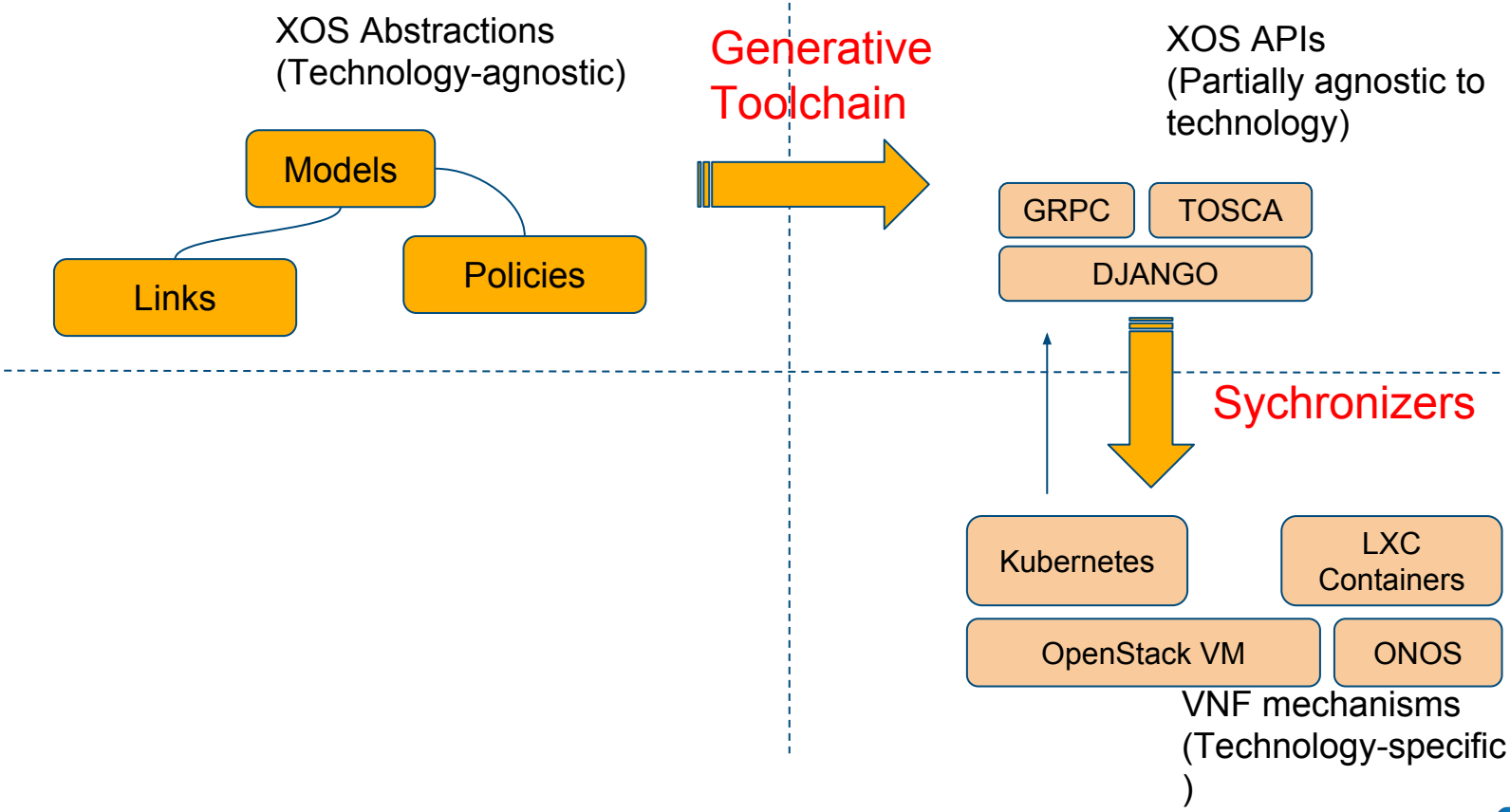
Opportunity: Synchronizer Performance Brigade

- Synchronizer's work divided into independent cohorts
  - Opportunity to scale up
- **Synchronizer is not reentrant**
  - **High latency**
  - **Objects should get processed even while cohorts are being executed**
- Opportunity not fully utilized
  - Context for parallelization is threads (only vertical scale up)
  - Implement distributed run queue

Opportunity: Extend generative (xproto) toolchain

- Code generation simplifies development and leads to reliable code
- Tasks:
  - Identify common patterns in real services
  - Express those patterns in xproto representations
  - Autogenerate stub services to match those patterns

Opportunity: Static Synchronizers



Resources

- CORD Guide:
  - http://guide.opencord.org/