# From Zero to Orchestrated

Domain Models

# Agenda

- 8:00 – 8:30 Introduction
- 8:30 – 9:00 Bootstrapping of the development environment
- **9:00 – 9:30 Details of the Orchestrator domain models**
- 9:30 – 10:00 Break
- 10:00 – 11:30 Development of your first Orchestrator workflow
- 11:30 – 13:00 Lunch
- 13:00 – 14:30 Integration of OSS and BSS to your workflow
- 14:30 – 16:00 Tailoring the Orchestrator to your needs (Discussion)

# Topics

1. Domain model basics
2. Example domain models
3. Tips for building domain models
4. Updating existing domain models

# Domain Model Basics

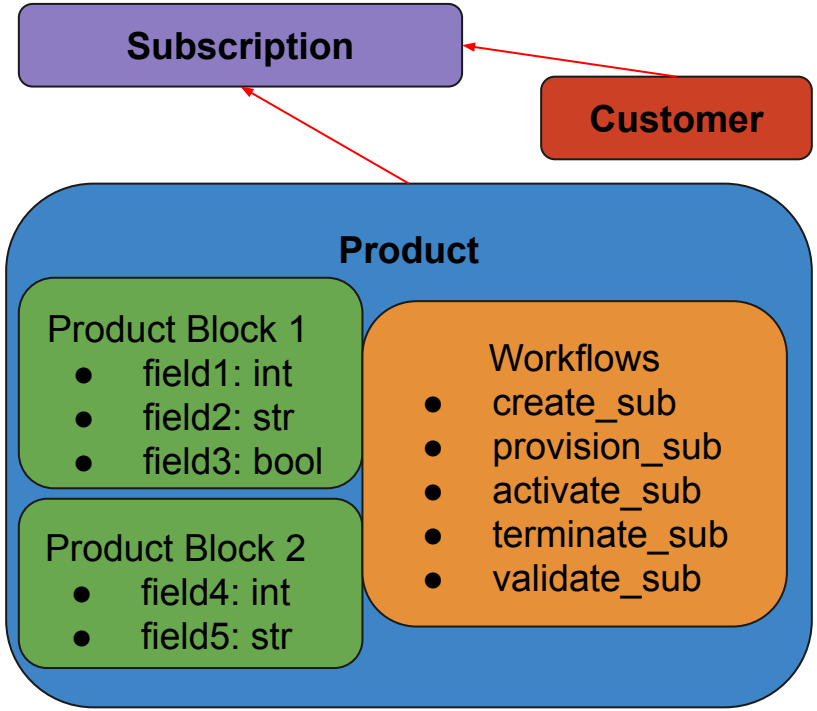
- What is a domain model
- Components of a domain model
  - Product
  - Product Blocks
  - Resources
  - Workflows

# What is a domain model?

- A representation of a product offering, the reusable components of that product, and the actions an engineer can take to manage a customer's subscription to the product
  - Product offering: **Product**
  - Components of product: **Product Block** and its **Resources**
  - Management actions: **Workflows**

# Domain Model Basics

- **Subscription**: associates a customer with a product
- **Product**: a collection of product blocks and workflows
- **Product block**: a reusable collection of fields
- **Workflow**: an action that can
  - modify the content of product blocks
  - update the state of the subscription
  - effect change in an external system



# Domain Model Basics

**Product**

- Defines an offering provided to users
- Composed of one or more product blocks and a set of workflows (typically at least 3)

**Product Block**

- Reusable collection of resources

**Resource**

- A pydantic field on a product block

**Workflow**

- A script-like process to manage a subscription

# Domain Model Basics

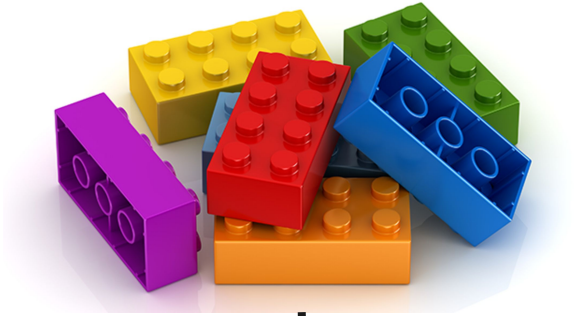
- Kinds of data that are good to place product blocks
  - Resource IDs
  - Resource names
  - Configurable values relevant to the customer's subscription
    - i.e., speed, bandwidth, object state
  - References to other domain models
- Kinds of data that are not great for product blocks
  - Duplicate data from an external system that frequently changes
    - Anything not used to identify a resource or configure the network
    - Will require constant updating, else the subscription will become out-of-sync

# Domain Model Basics

## Workflows

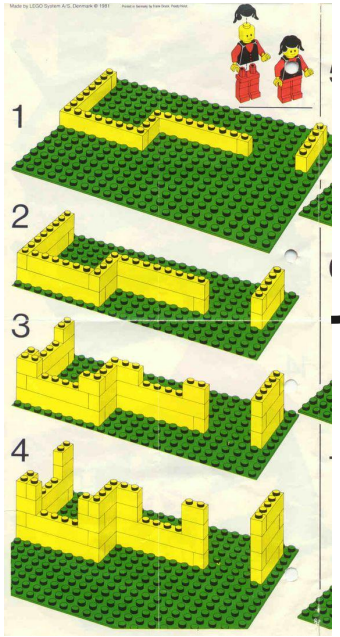
- What are workflows
- Role of workflows with other constructs
- Examples

# What are Workflows?



Finite resources exist in other systems

Workflows gather resources from those systems and associates them together to create a subscription




Orchestrator contains the final product

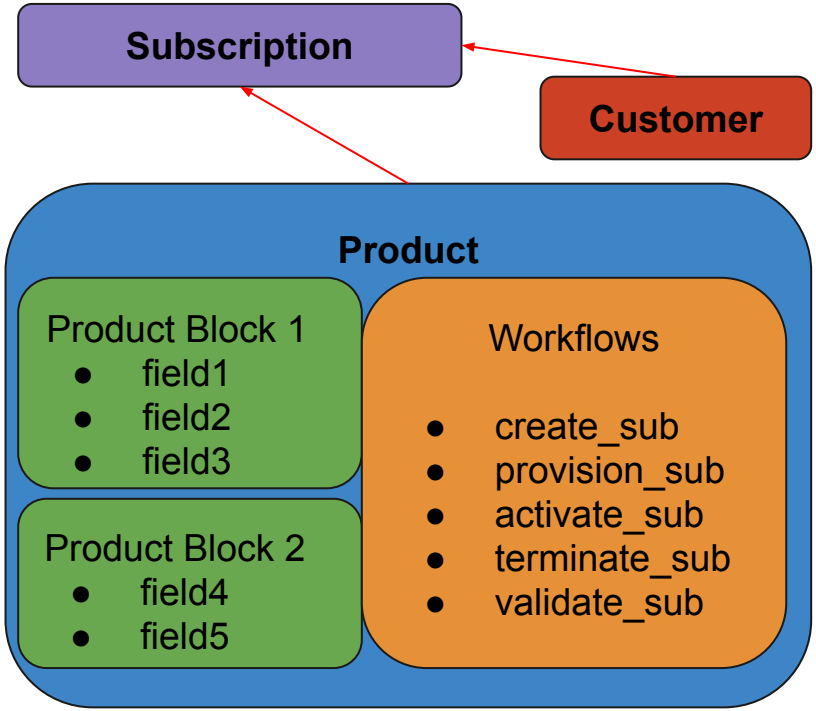
# What are Workflows?

- Workflows are actions that can be taken on a product subscription
- Workflows instantiate, modify, and terminate subscriptions
- Typical kinds of workflows:
  - create_subscription (instantiates a subscription to the product)
  - modify_subscription (toggle change in an external system)
  - terminate_subscription (ends a subscription to the product)
  - validate_subscription (checks the data in external systems is in sync with the domain model)

# Role of workflows with other constructs

- Workflows belong to a product
- Workflows instantiate and modify a customer's subscription to a product
- Workflows make changes to the product blocks associated with a product
- Workflows manage the lifecycle of a subscription:

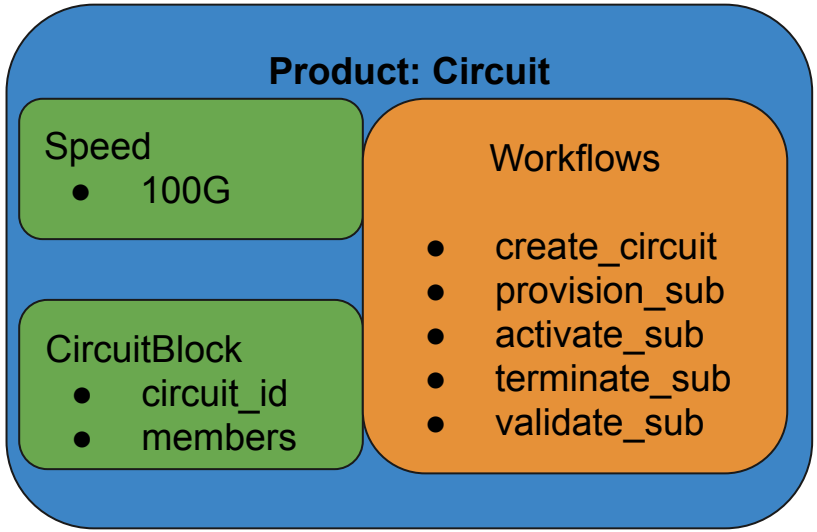
  inactive -> provisioning -> active ->terminated



# Using Domain Models

- Define a set of Product Blocks, each with a set of resources
- Write workflows to
  - Instantiate and modify a subscription to the product
  - Make changes to external systems
- Create a Product that is composed of product blocks and workflows
- Workflows carry a subscription through a lifecycle:

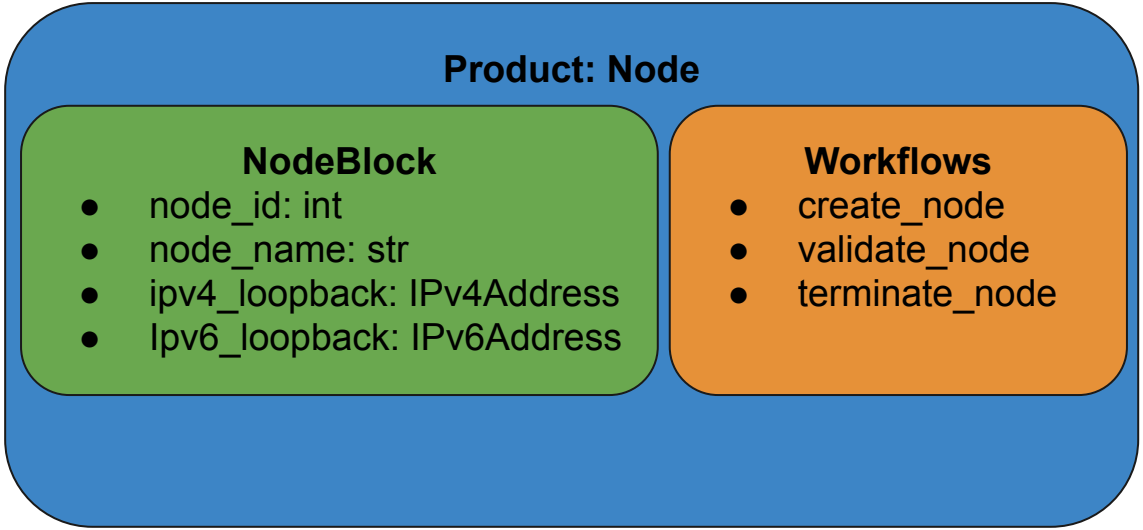
  inactive -> provisioning -> active -> terminated



# Example domain models: diagrams

**Node: represents a router, switch, or transponder**

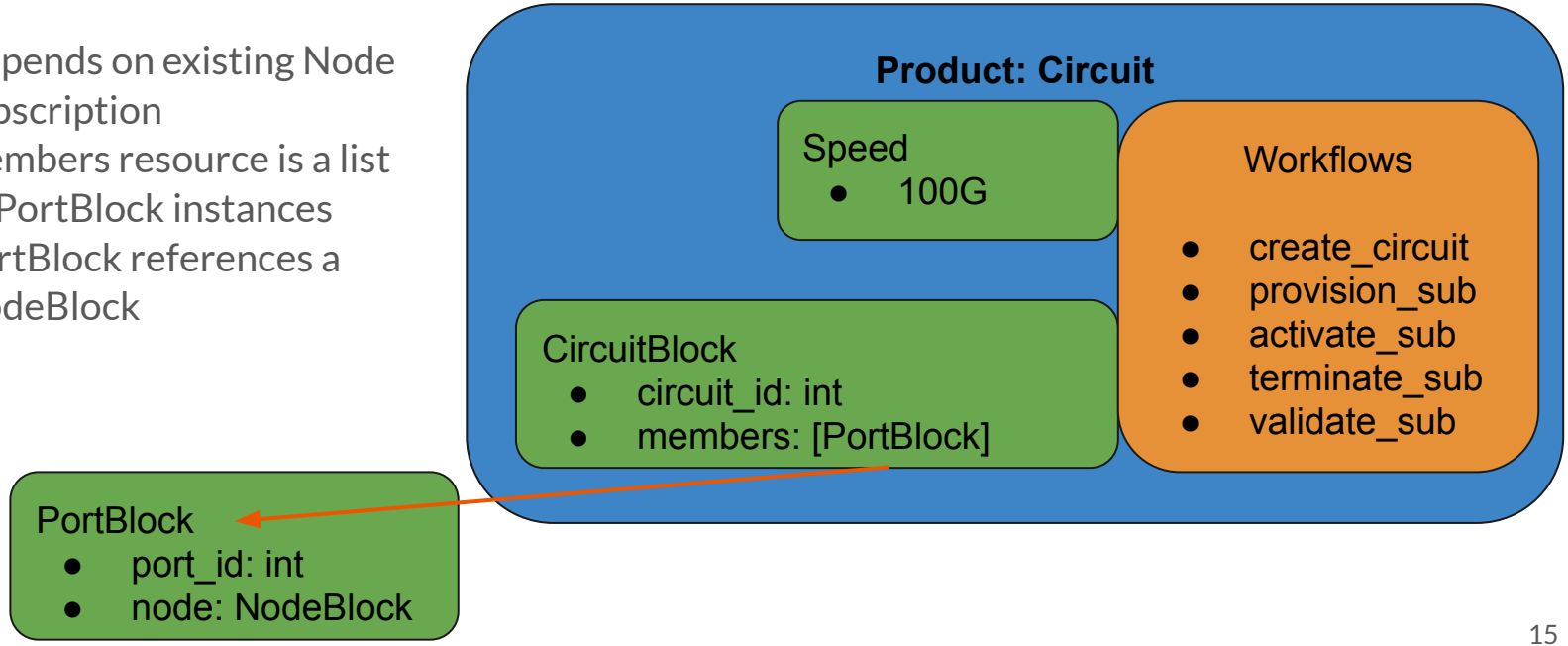
- Customer subscribes to node product
- Engineer uses workflows to instantiate, manage, and terminate subscription



# Example domain models: diagrams

**Circuit: represents a layer 3 connection to a customer**

- Depends on existing Node subscription
- members resource is a list of PortBlock instances
- PortBlock references a NodeBlock



# Example domain models: the code

Product

Product Block

```
class Node(NodeProvisioning, lifecycle=[SubscriptionLifecycle.ACTIVE]):
    node: NodeBlock
```

Product Block

Fields

```
class NodeBlock(NodeBlockProvisioning, lifecycle=[SubscriptionLifecycle.ACTIVE]):
    """Node  with optional fields to use in the active lifecycle state."""

    node_id: int
    node_name: str
    ipv4_loopback: IPv4Address
    ipv6_loopback: IPv6Address
```

# Circuit Example

Product

Product Blocks

Product Block

Fields

```python
class Speed(strEnum):
    HUNDREDG = "100G"


class CircuitInactive(SubscriptionModel, is_base=True):
    # Equipment state is planned
    speed: Speed
    circuit: CircuitBlockInactive


class CircuitProvisioning(
    CircuitInactive, lifecycle=[SubscriptionLifecycle.PROVISIONING]
):
    speed: Speed
    circuit: CircuitBlockProvisioning
```

```python
class CircuitBlockInactive(ProductBlockModel, product_block_name="Circuit"):
    """Object model for a Circuit as used by
    Backbone Link Service"""

    members: PortPair[Layer3InterfaceInactive]
    circuit_id: int | None = None
    under_maintenance: bool | None = None


class CircuitBlockProvisioning(
    CircuitBlockInactive, lifecycle=[SubscriptionLifecycle.PROVISIONING]
):
    """Circuit with fields for use in the provisioning lifecycle"""

    members: PortPair[Layer3InterfaceProvisioning]
    circuit_id: int
    under_maintenance: bool
```

```python
class PortPair(SubscriptionInstanceList[T]):
    min_items = 2
    max_items = 2
```

```python
class Layer3InterfaceInactive(
    ProductBlockModel, product_block_name="Layer 3 Interface"
):
    port: PortInactive
    v6_ip_address: IPv6Interface | None = None
```

```python
class PortInactive(ProductBlockModel, product_block_name="Port"):
    port_id: int | None = None
    port_description: str | None = None
    port_name: str | None = None
    node: NodeBlock | None = None
```

# Tips for building your own domain models

- Try to think in real-world, reusable blocks
  - Node, Server, Circuit are basic, reusable concepts
  - NokiaServer, ECMPCircuit, ManagementPort are less reusable constructs that can be constructed from simpler blocks
- Enumerate types of a basic component
  - Most offerings can be encapsulated by some combination of Circuit, Node, and Port
  - Adding enumerations for things like Speed, CircuitType allows for making several products with the same blocks
- If something changes, you can always modify the domain model via a migration

# Updating existing domain models

- Domain models are stored in a postgres database
- They are created, updated, and removed by postgres migrations
- To change a domain model:
  - make the change in its representation in its product_type and product_block files
  - use migration tool to generate a migration to enact the change:

```
> python main.py db migrate-domain-model “migration_name”
```