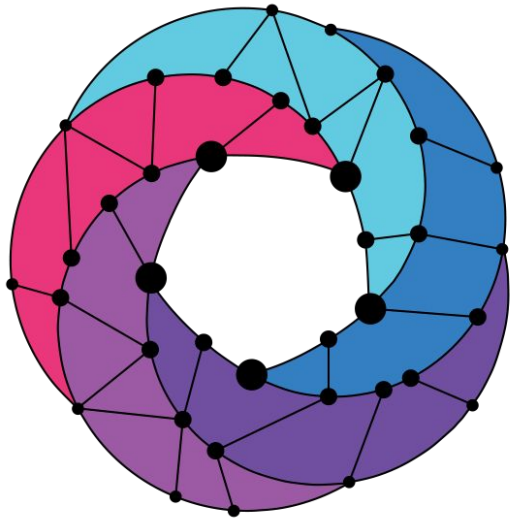


Open Source
MANO

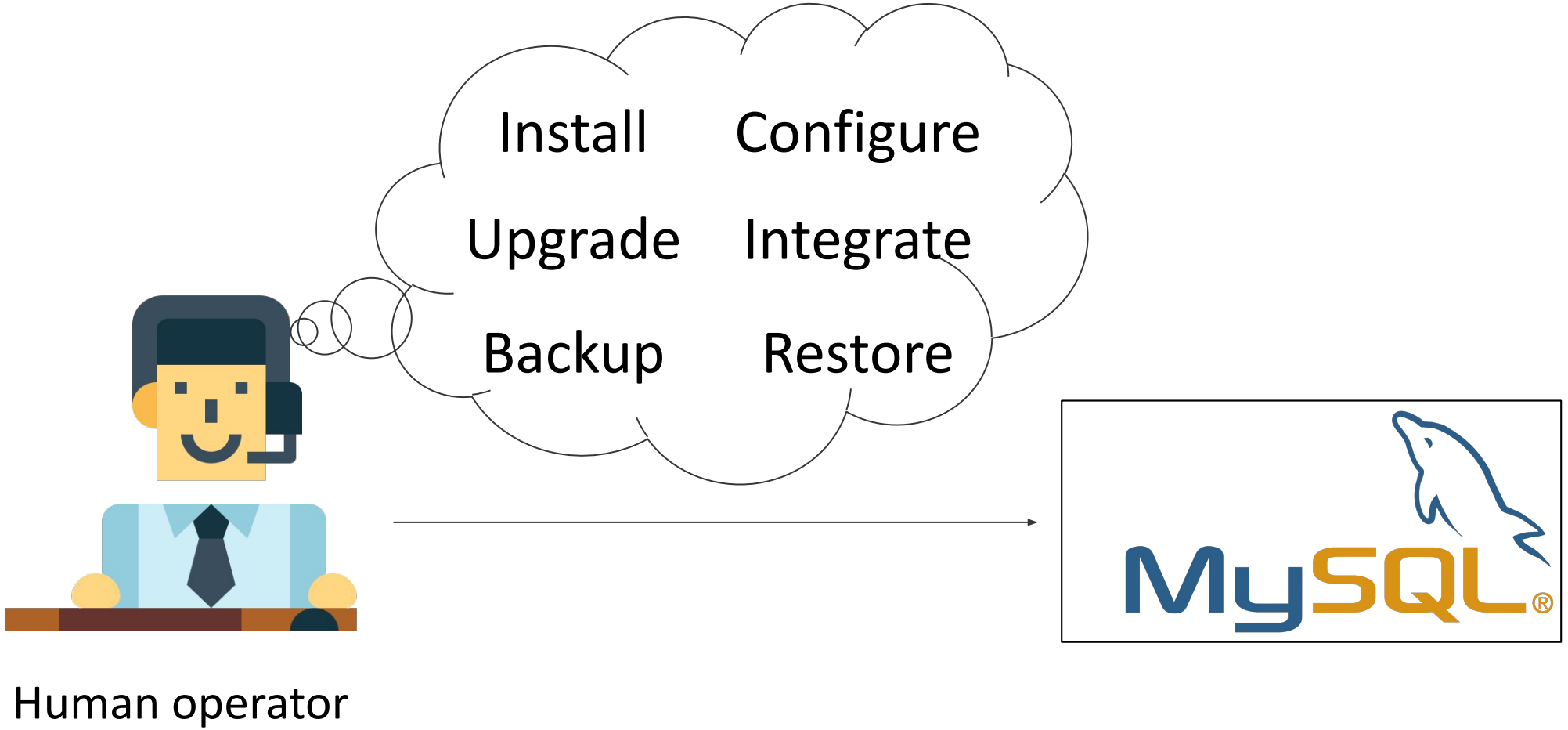
Introduction to OSM Primitives

David Garcia (Canonical)

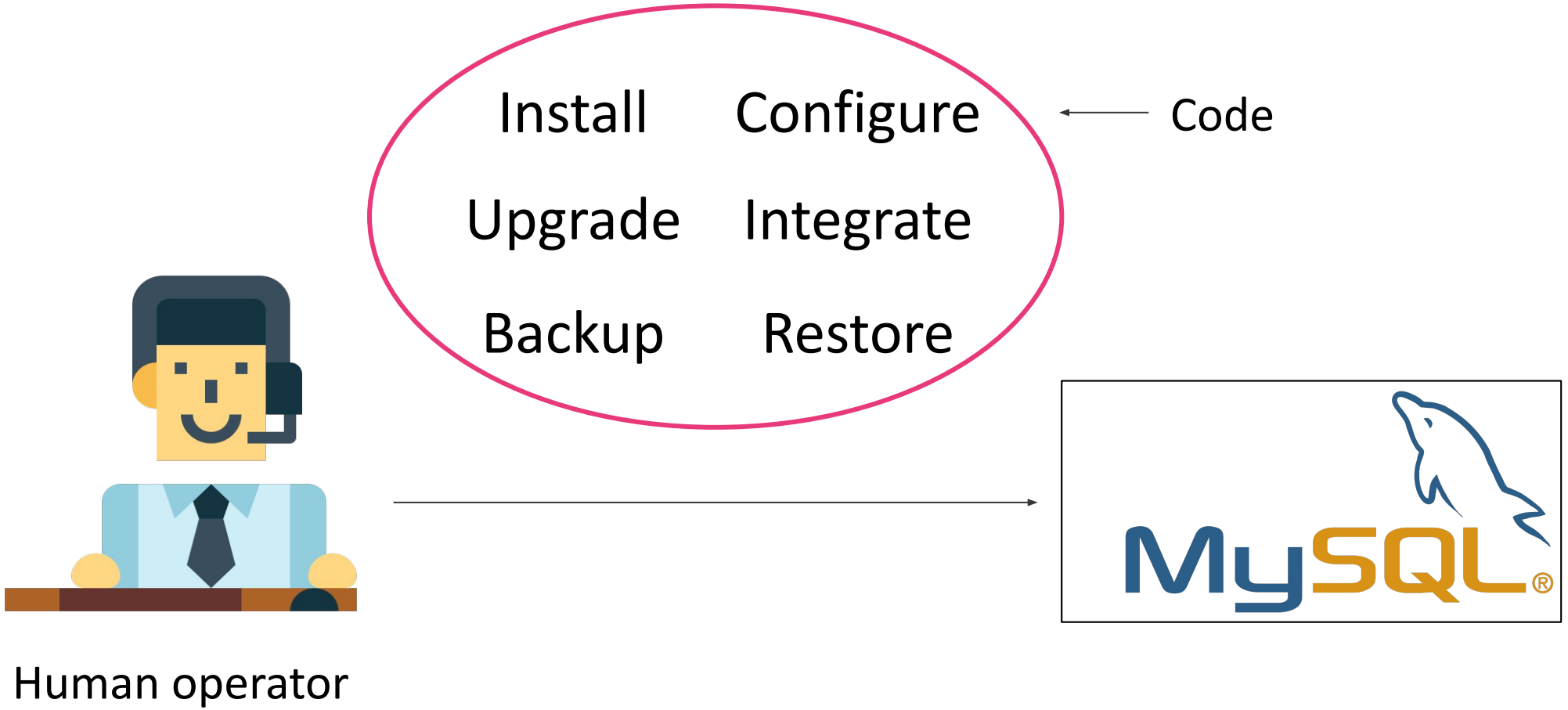
ETSI

Primitives are actions exposed by the operator

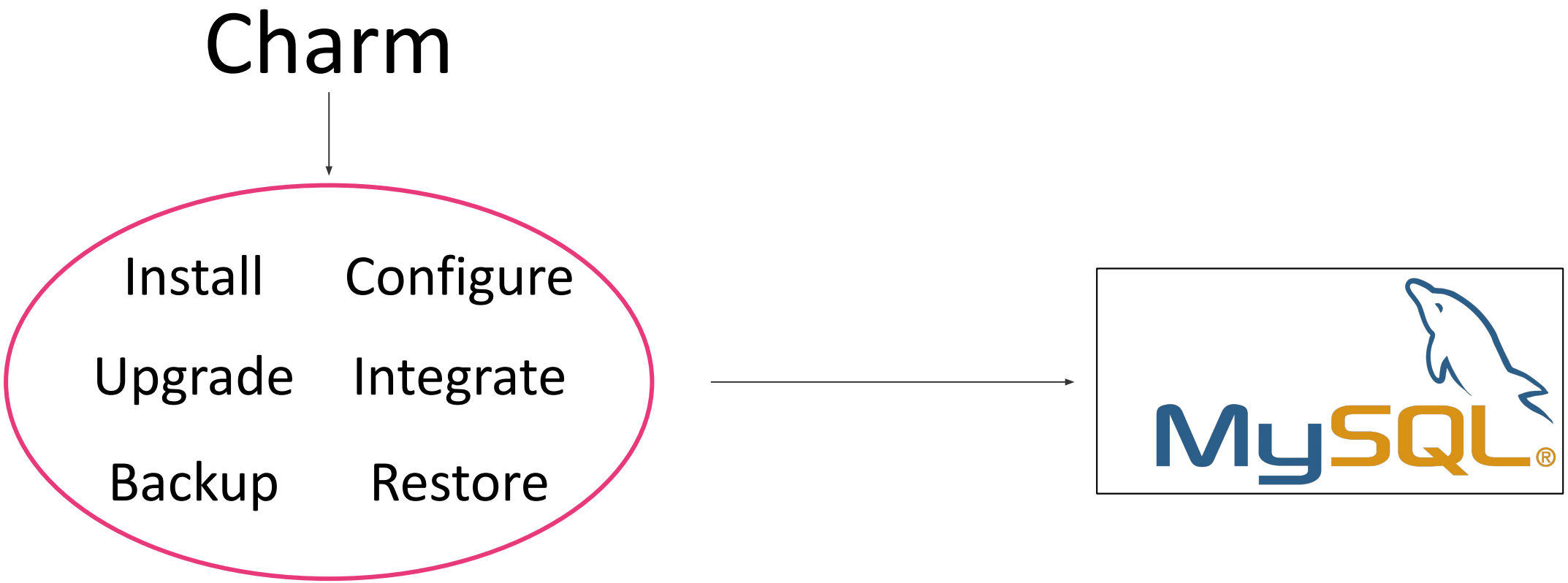
Operator Pattern



Operator Pattern



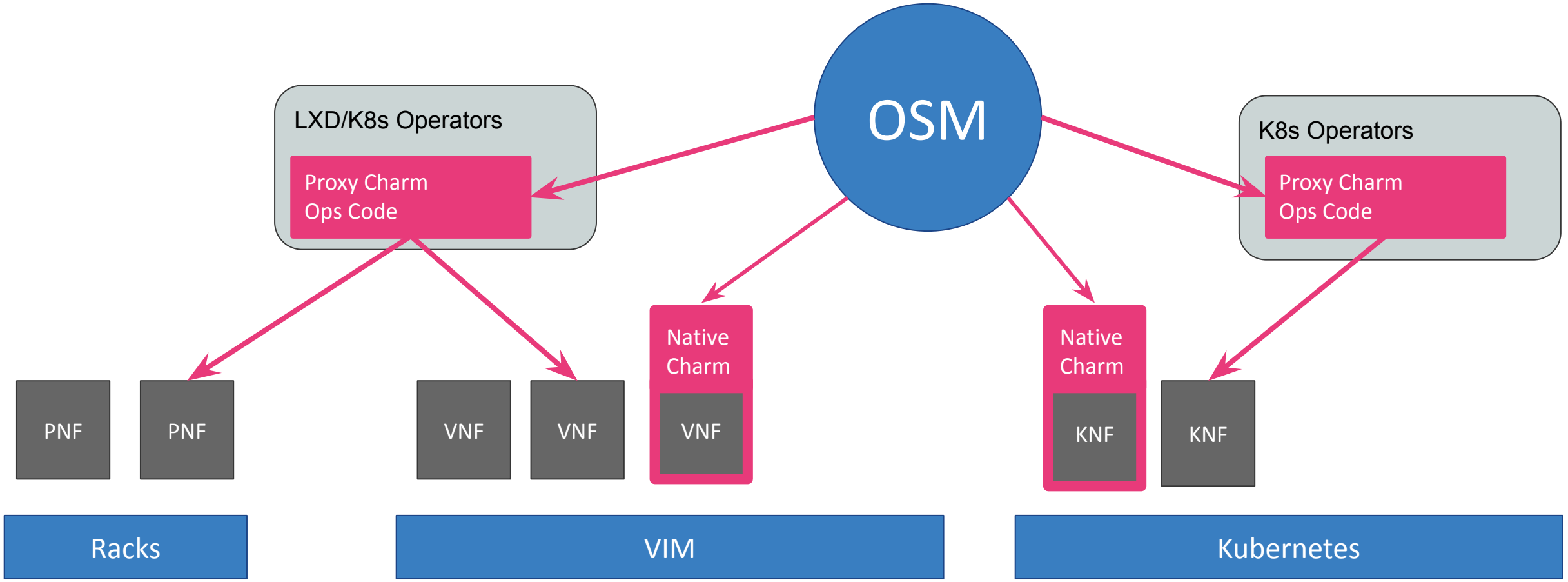
Operator Pattern



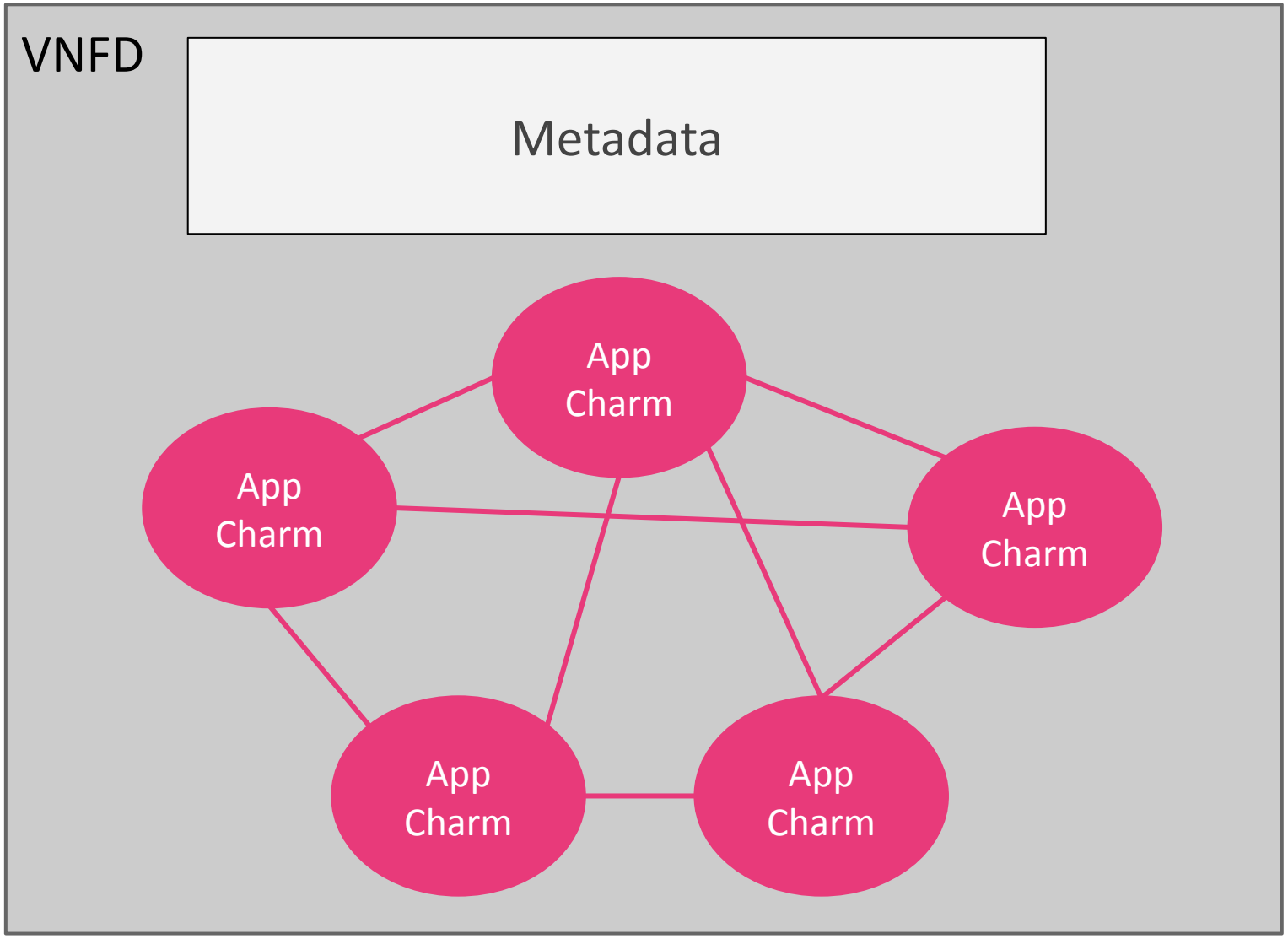
Open Source MANO

Charms are universal operators

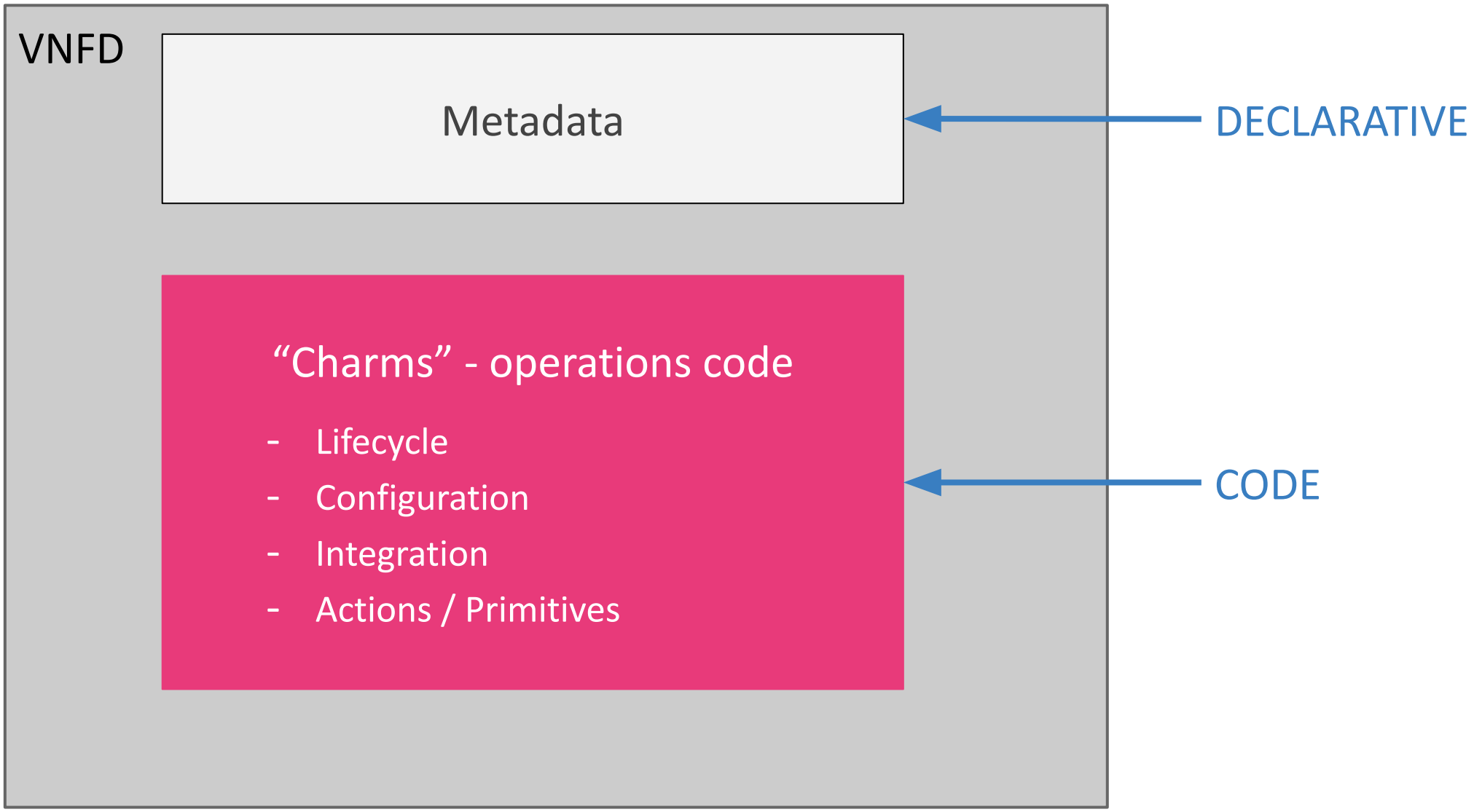
Reality is messy and mixed



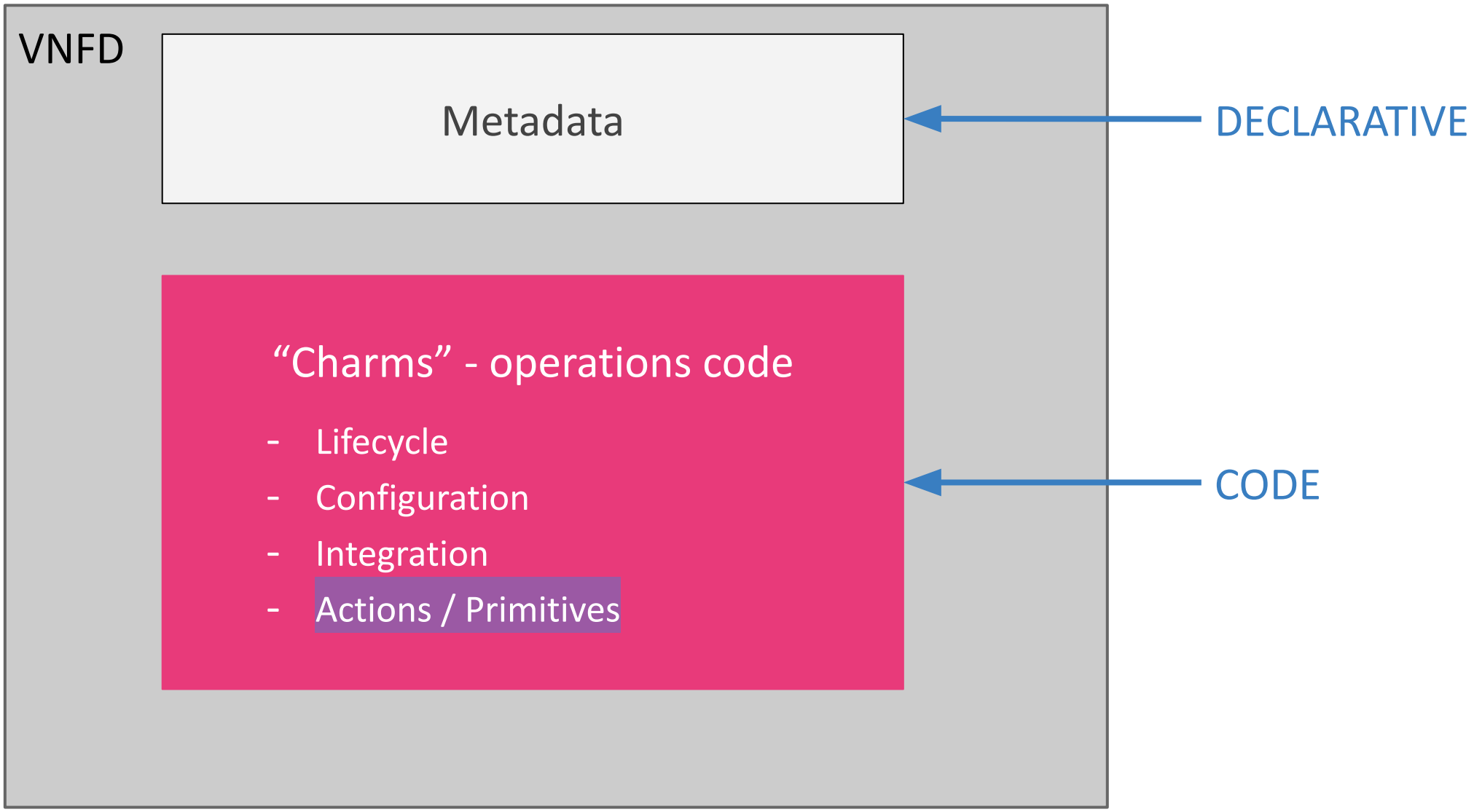
One VNF is many apps and integration



VNFD



VNFD

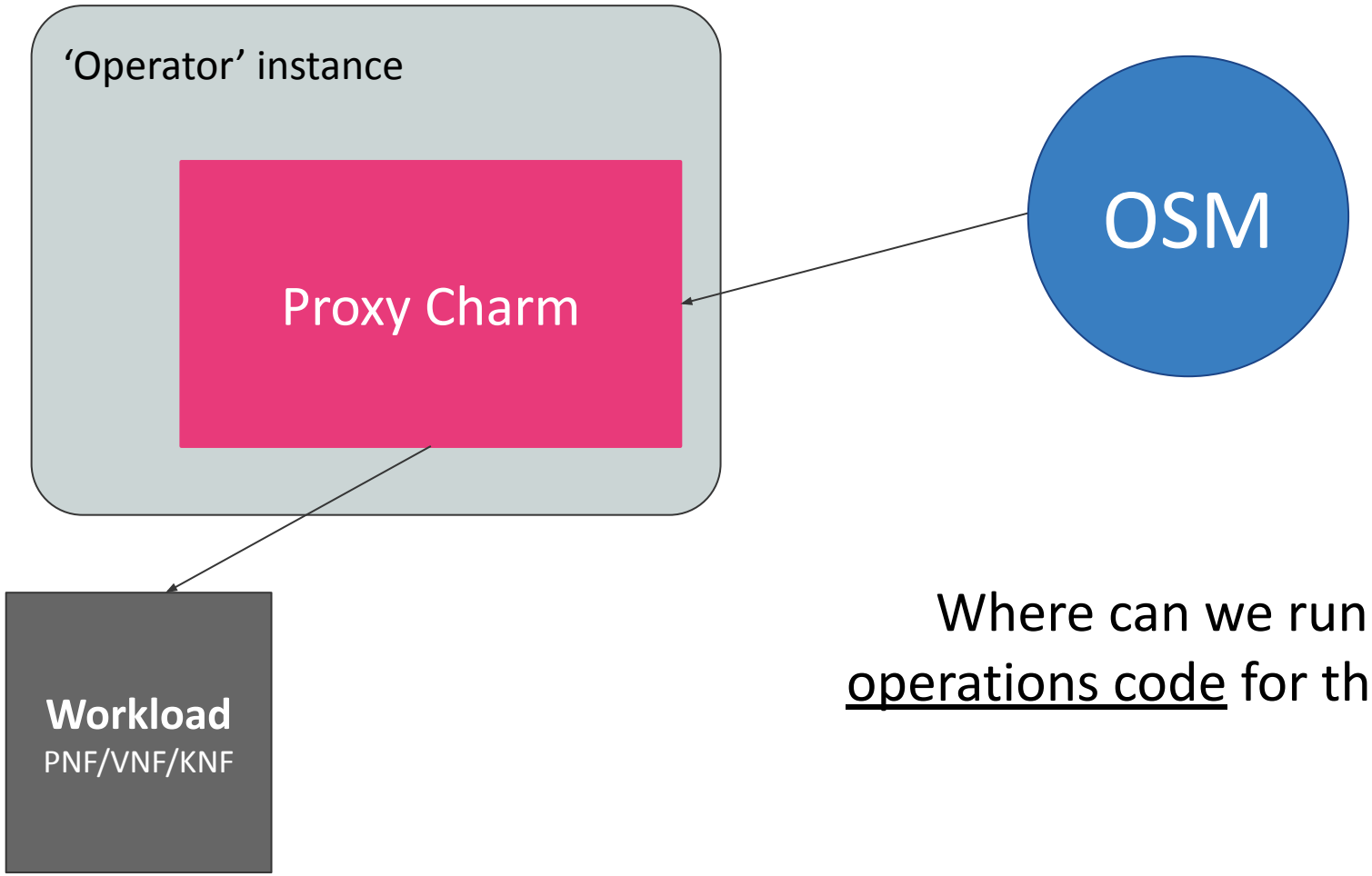


Primitives are actions exposed by the operator

Open Source MANO

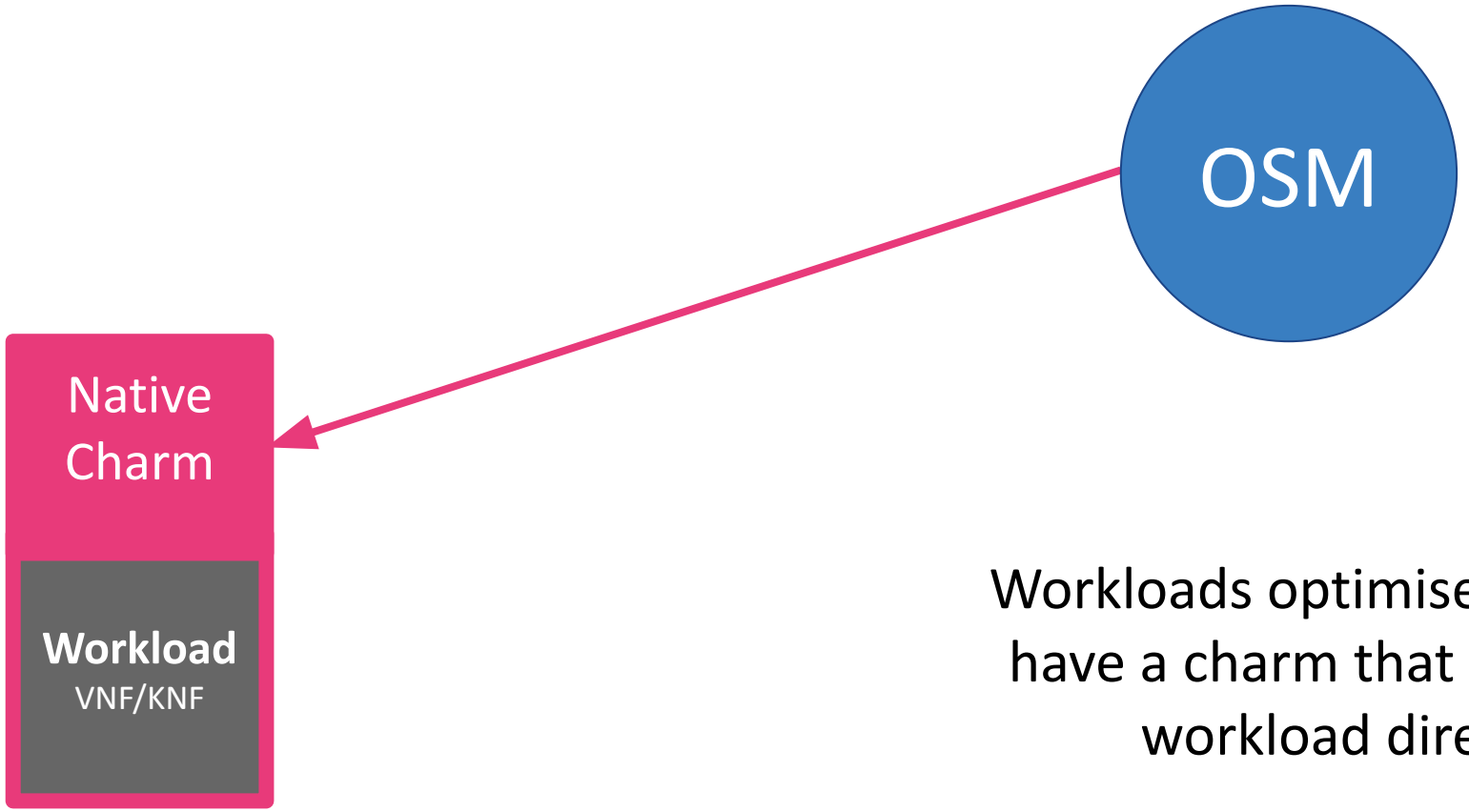
OSM Primitives are actions exposed by the Charm

Operating “proxy” workloads



Where can we run our own operations code for this workload?

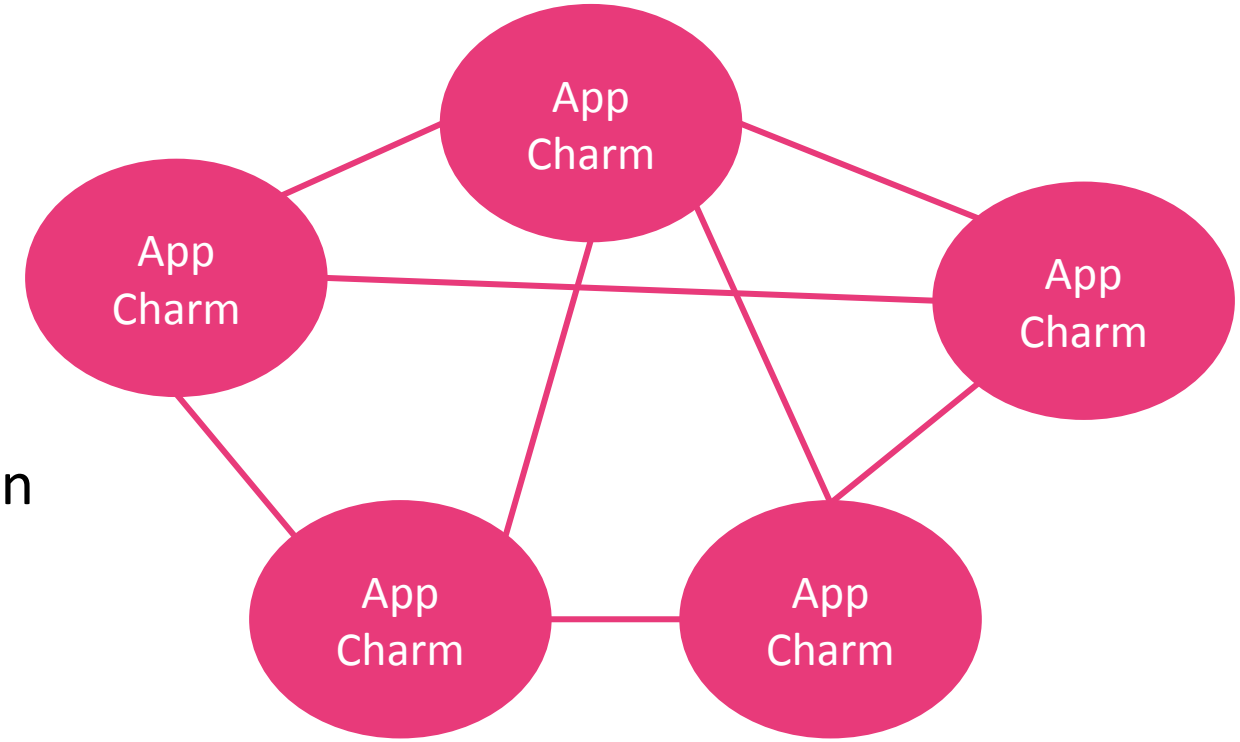
Operating “native” workloads



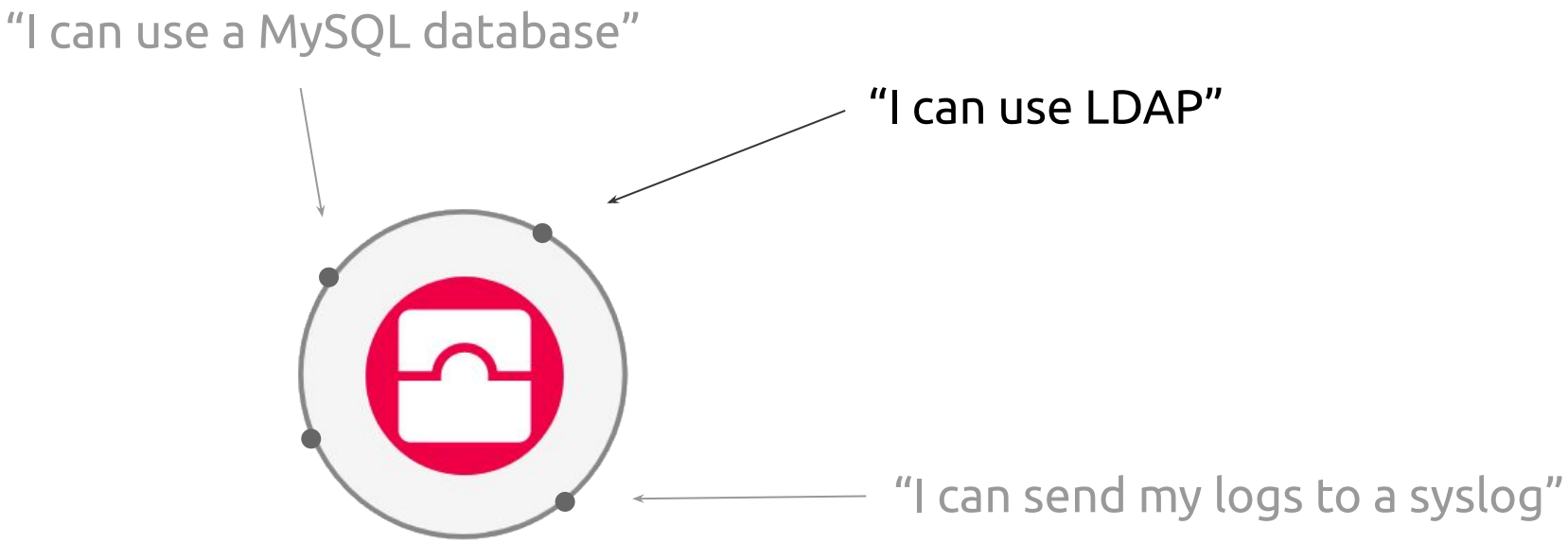
Workloads optimised for OSM have a charm that drives the workload directly.

Integration is first-class in the VNFd

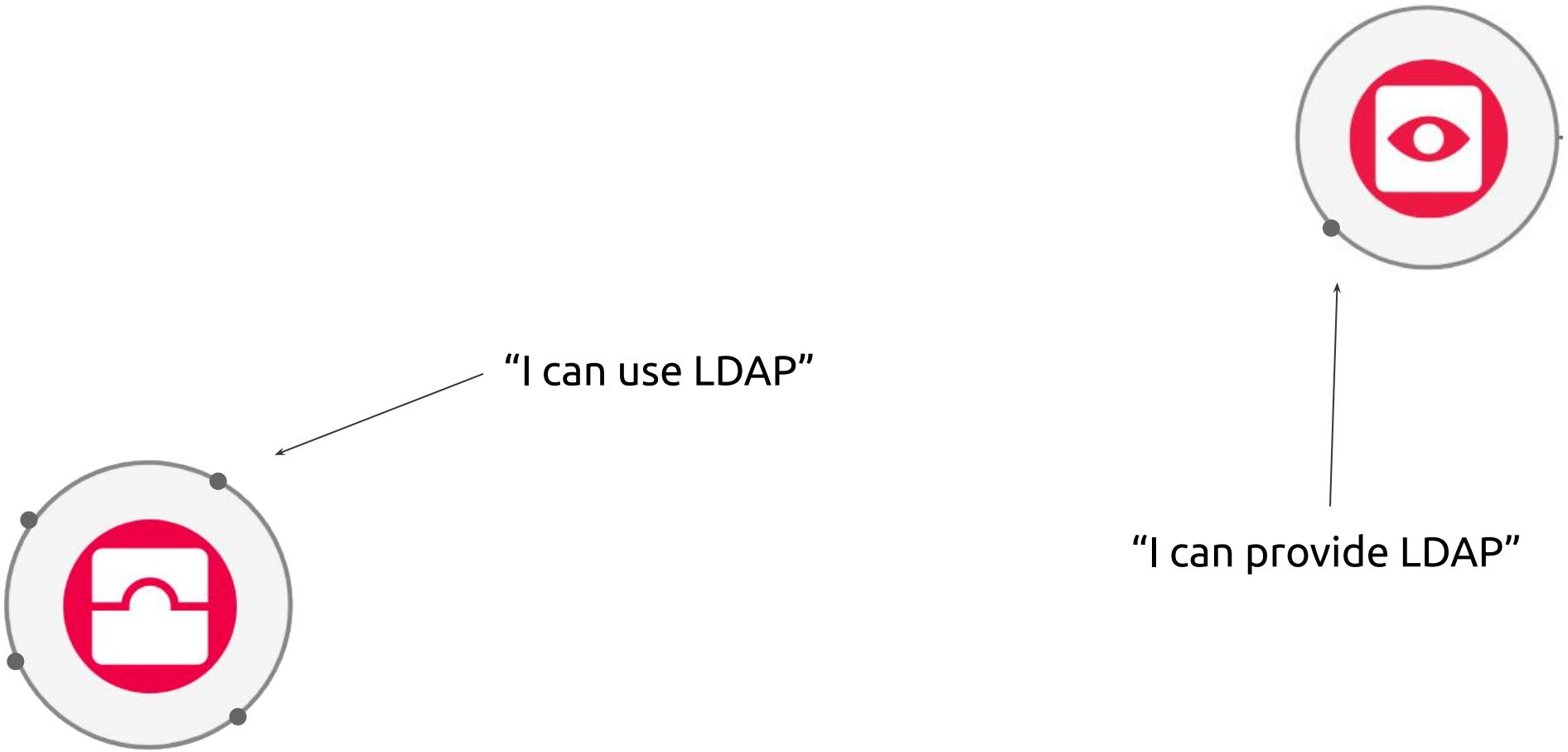
Lines of integration in the VNFd



Charms declare typed integration points

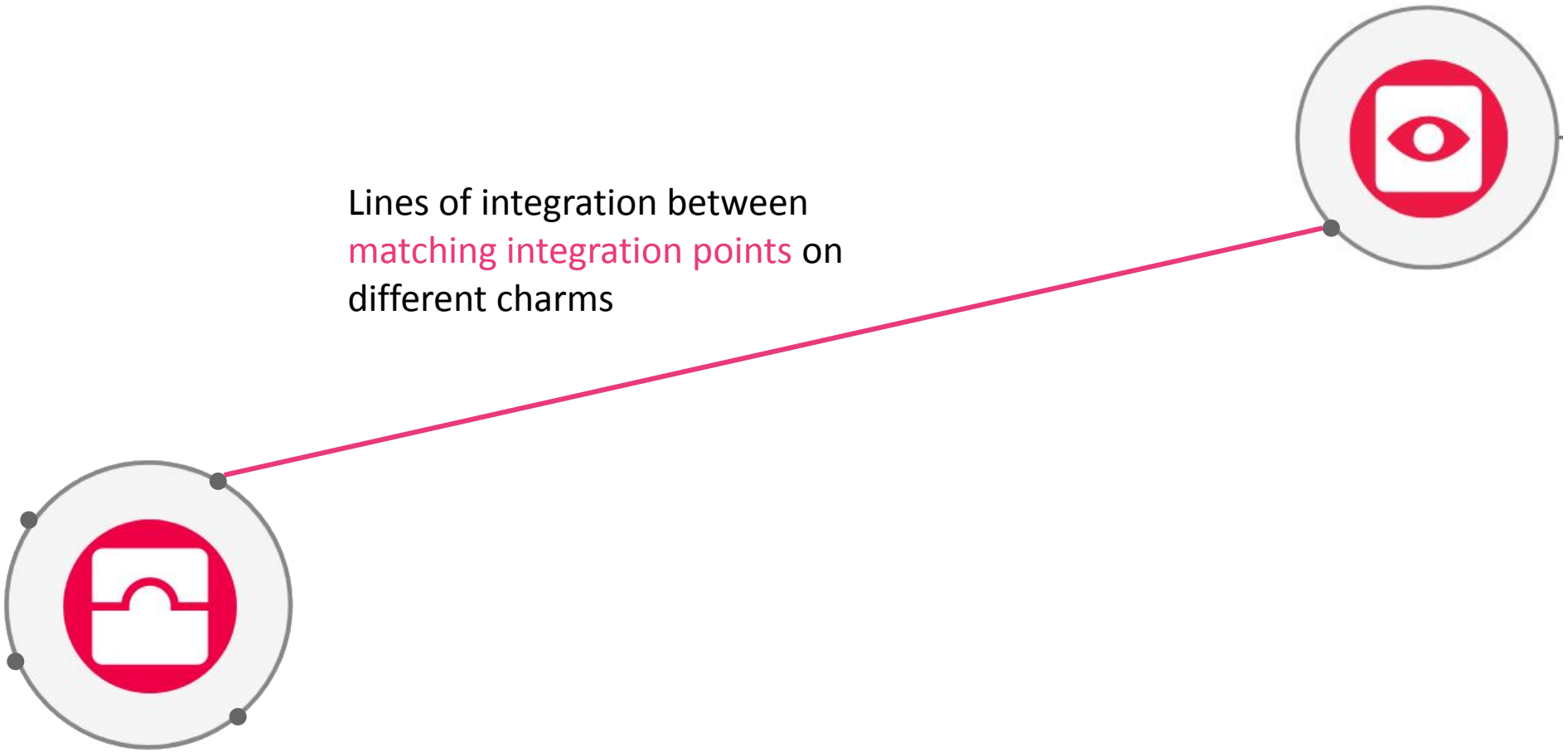


Charms declare typed integration points

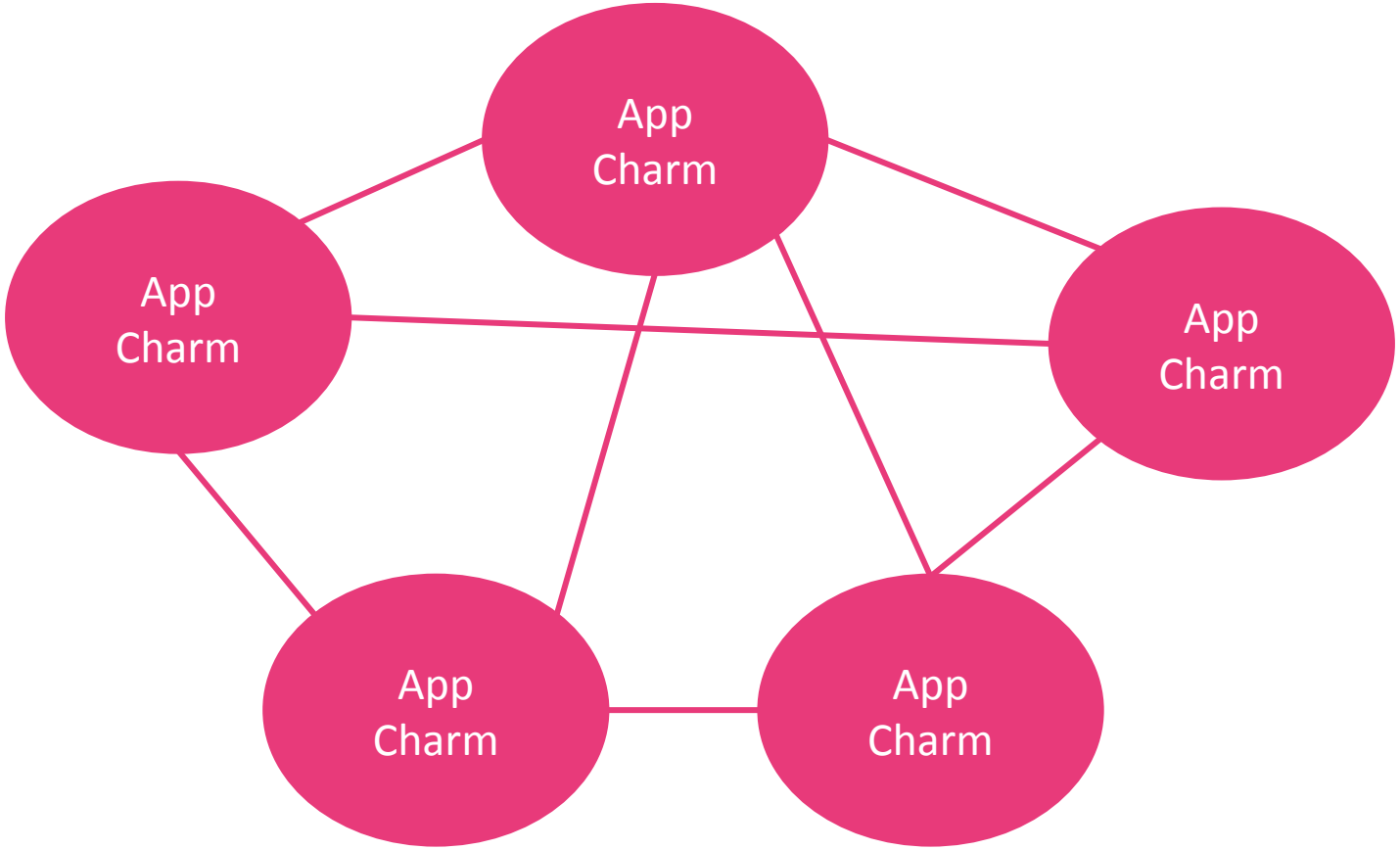


Matching integration points can be related

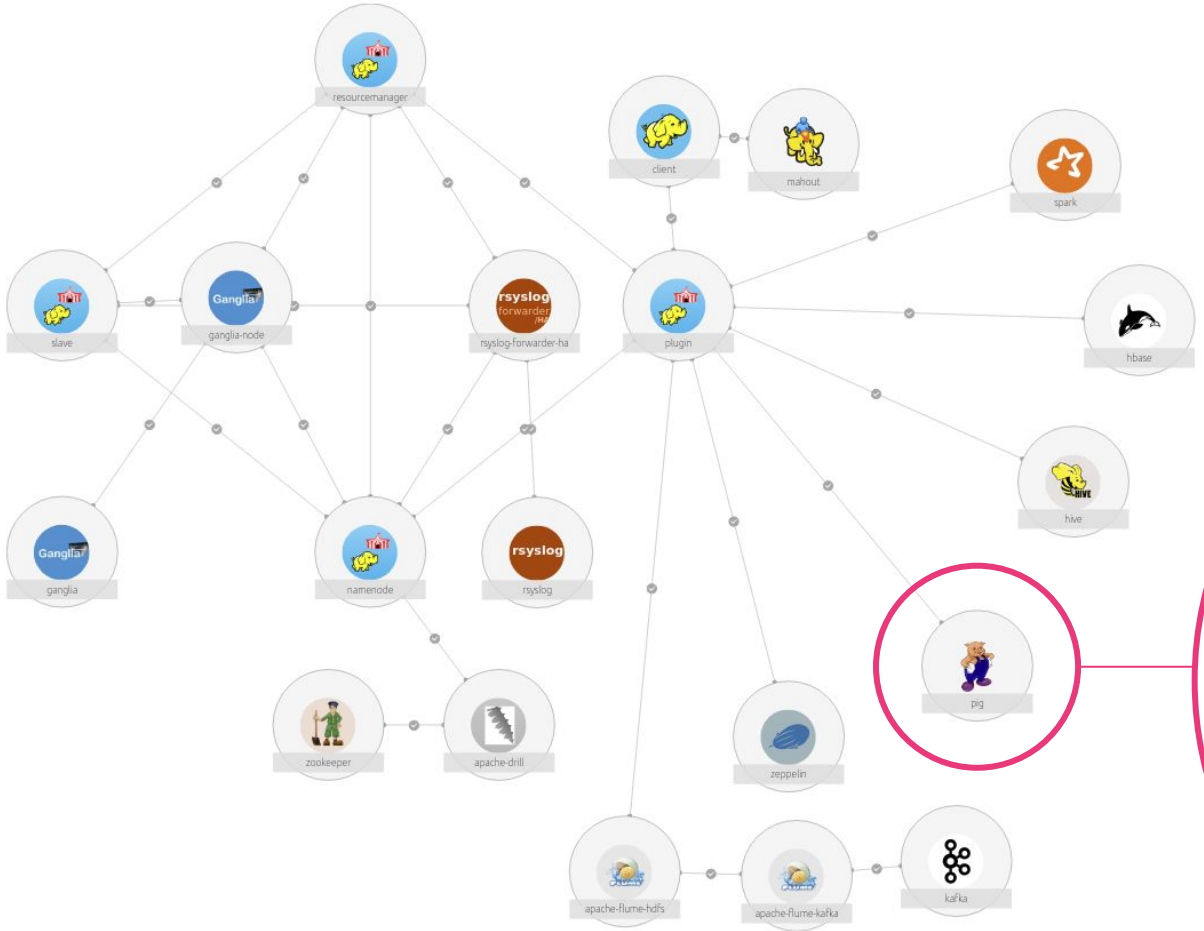
Lines of integration between matching integration points on different charms



Composition gives complex integrations



VNFds can describe complex integrations



Lifecycle scripts
Config scripts
Integration scripts
Action scripts

OSM primitives are Charm Action scripts

- Backup
- Monitor
- Debug
- Add users, policies, rules, etc.
- Manage certificates, keys, etc.
- Rotate logs

Each ‘ primitive’ is a **charm action** script that takes parameters and produces output.

Charms are packages of scripts to drive apps

Open Source MANO

Lifecycle scripts

- install
- config
- update
- remove
- scale

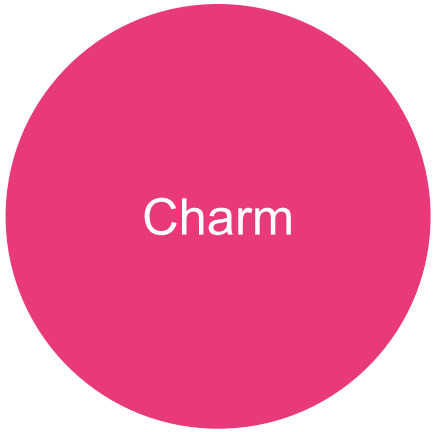
Integration scripts

- relate-mysql
- relate-ldap
- relate-proxy
- relate-...

“Action” scripts are OSM Primitives

“action: backup”
“action: restore”
“action: scan-viruses”
“action: health-check”
“action: add-repo”
“action: ...”
“action: ...”
“action: ...”

These are your operations primitives.



Charm describes Action parameters

Charm metadata describes the action parameters.

Each Action is a script, usually in Python or Bash.

```
addurl:
    description: "Add squid config"
    params:
        url:
            description: "URL that will be allowed"
            type: string
            default: ""
deleteurl:
    description: "Delete allowed URL squid config"
    params:
        url:
            description: "URL that will stop to be allowed"
            type: string
            default: ""
```

Charm Action script in bash

Open Source MANO

Actions can be written in bash for very simple cases.

```
#!/bin/bash

URL=`action-get url`

if ! grep -Fxq "http_access allow allowedurls"
/etc/squid/squid.conf
then
    sed  -i '/^# And finally deny all .*/i http_access allow
allowedurls\n' /etc/squid/squid.conf
fi

sed -i "/^http_access allow allowedurls.*/i acl allowedurls
dstdomain \.$URL" /etc/squid/squid.conf

kill -HUP `cat /var/run/squid.pid`
```

Charm Action script in python

It is common to write actions in Python using the standard Operator Framework.

```
def on_deleteurl_action(self, event):
     """Handle the deleteurl action."""
     url = event.params["url"]

     line_to_delete = "acl allowedurls dstdomain .{}".format(url)
     line_deleted = False

     with open("/etc/squid/squid.conf", "r") as f:
         lines = f.readlines()
     with open("/etc/squid/squid.conf", "w") as f:
         for line in lines:
             if line_to_delete not in line:
                 f.write(line)
             else:
                 line_deleted = True

     if line_deleted:
         event.set_results({"output": "URL deleted succesfully"})
         subprocess.check_output(
           "kill -HUP `cat /var/run/squid.pid`", shell=True)
     else:
         event.fail("No URL was deleted")
```

Open Source
MANO

Juju Controller

Operator Lifecycle Manager

OSM Architecture

Open Source MANO

NBI
"Northbound Interface"

UI
"User Interface"

LCM
"Lifecycle Manager"

RO
"Resource Orchestrator"

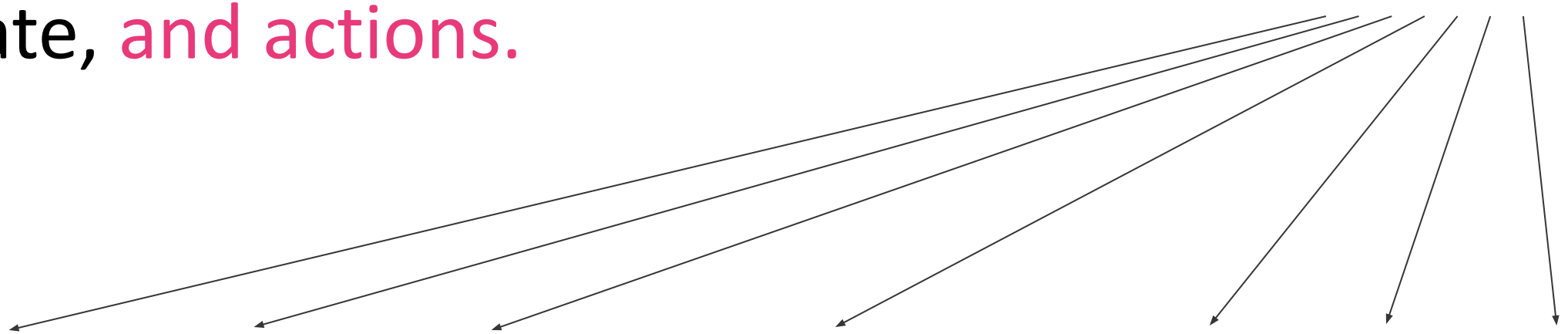
VCA
"VNF Configuration Abstraction"

Juju drives application operations on machine and kubernetes substrates

Open Source MANO

Juju

Install, update, configure, scale, integrate, and actions.



aws

Microsoft Azure

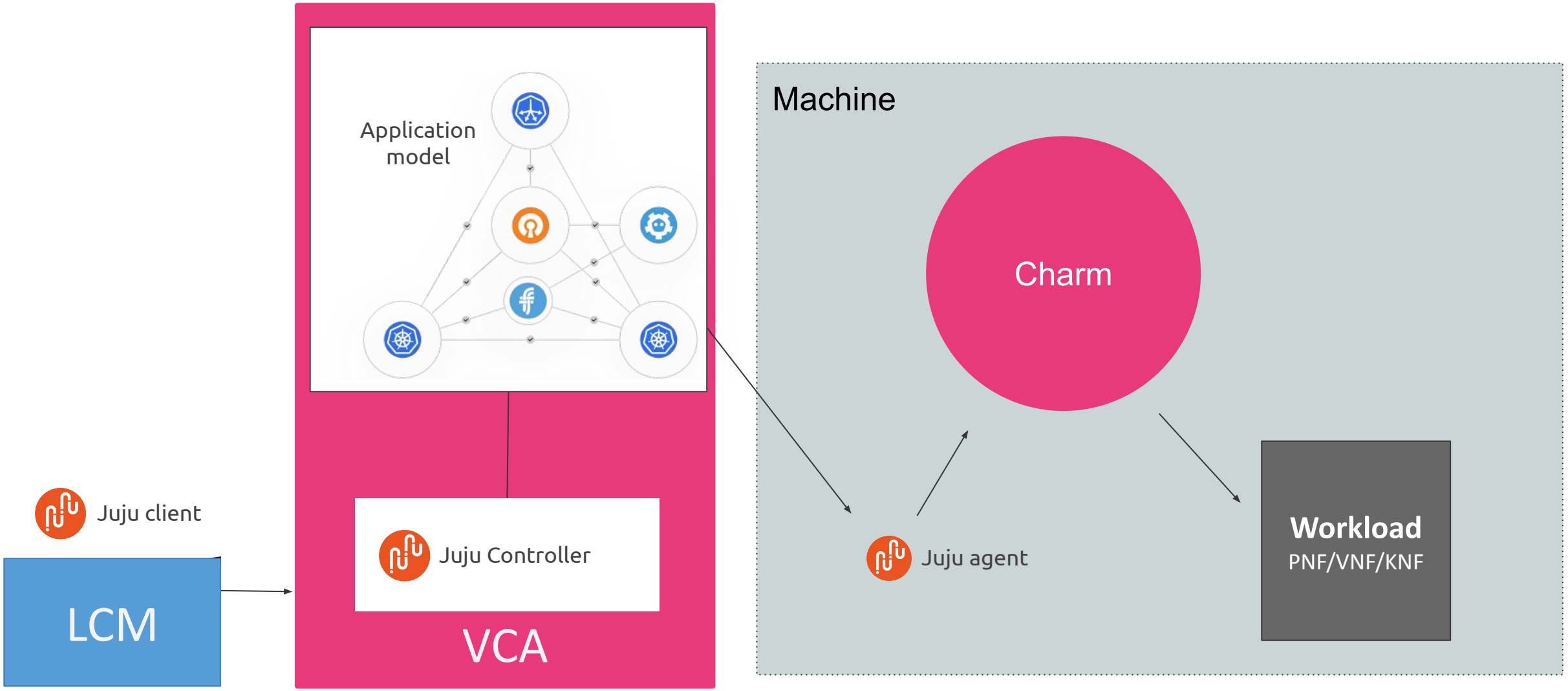
Google Cloud

MAAS

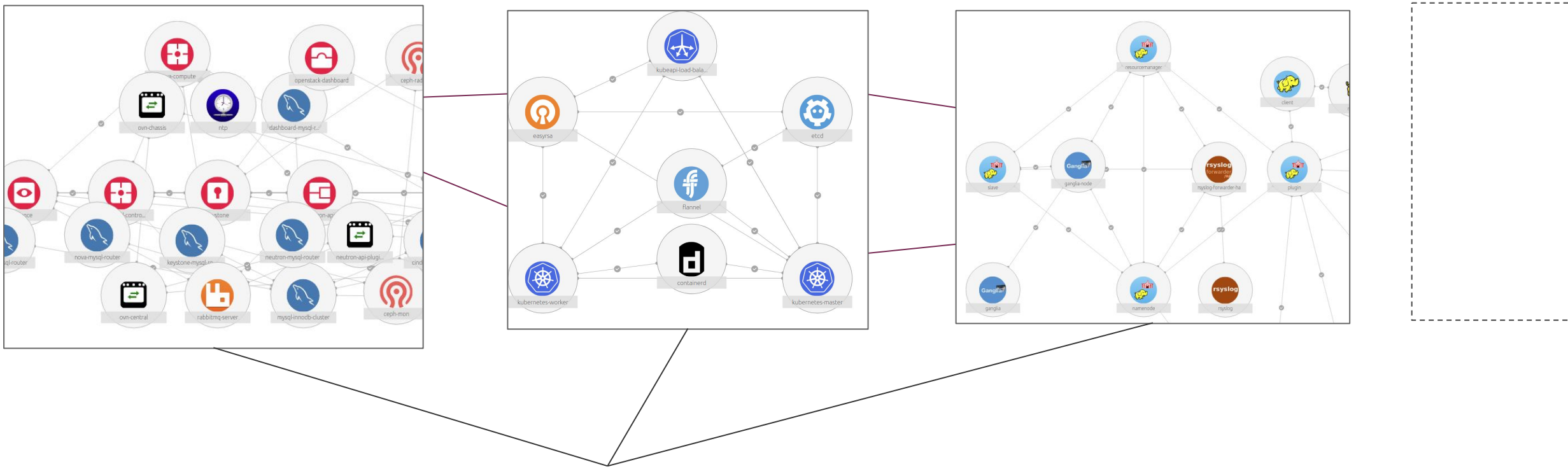
vmware

openstack.

Juju architecture

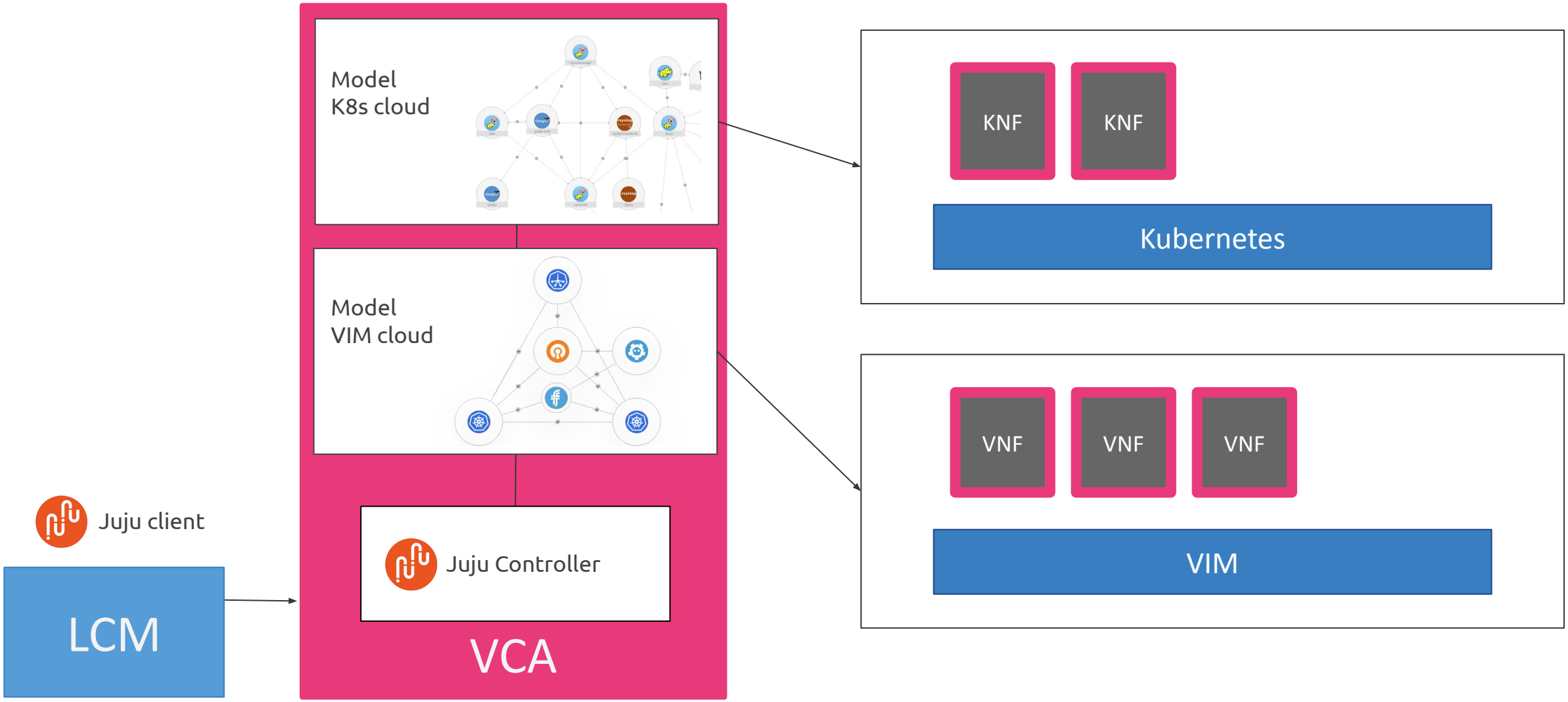


Juju controller manages multiple models

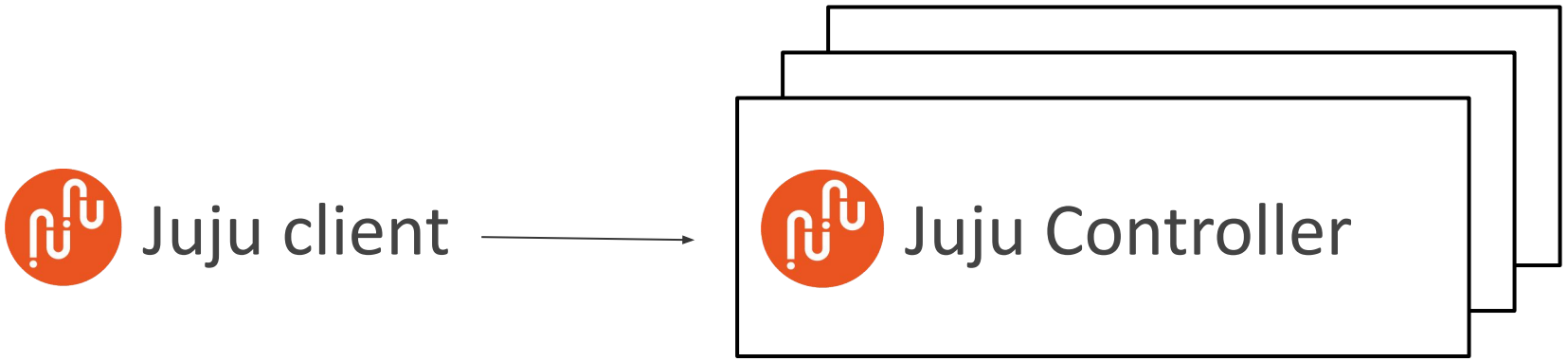


Juju client → Juju Controller

VCA uses multiple models for scenario

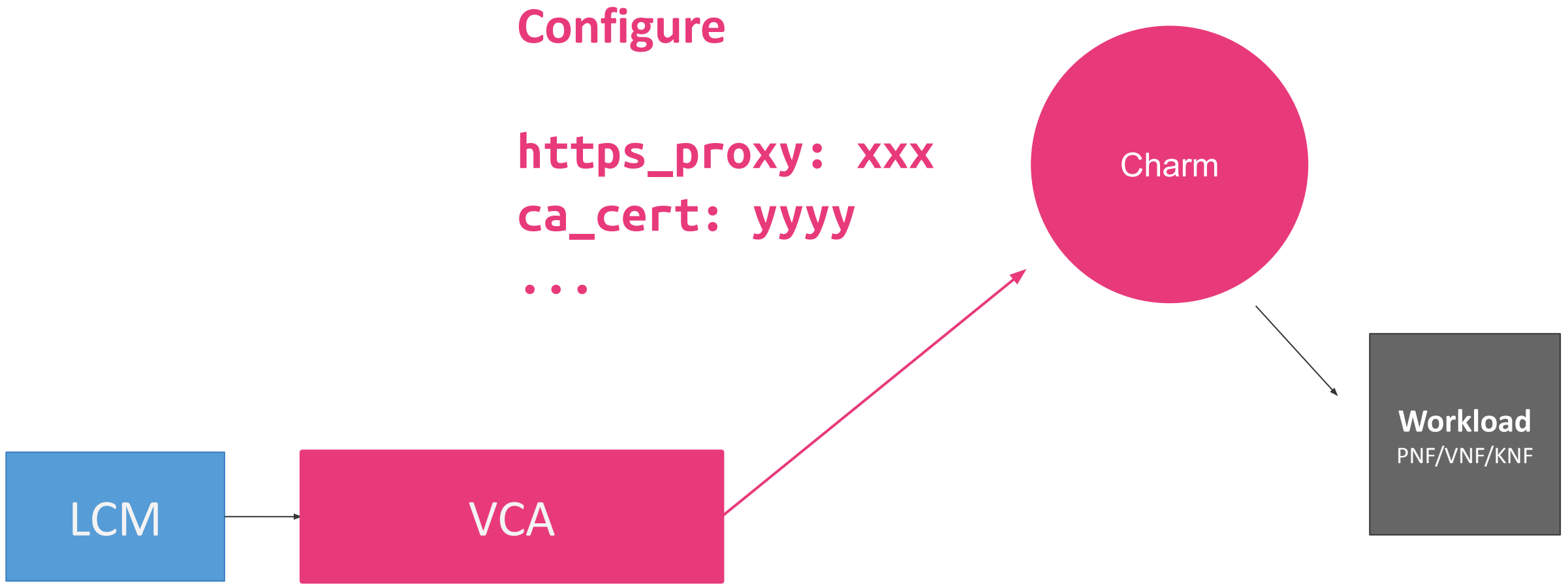


VCA can be highly available

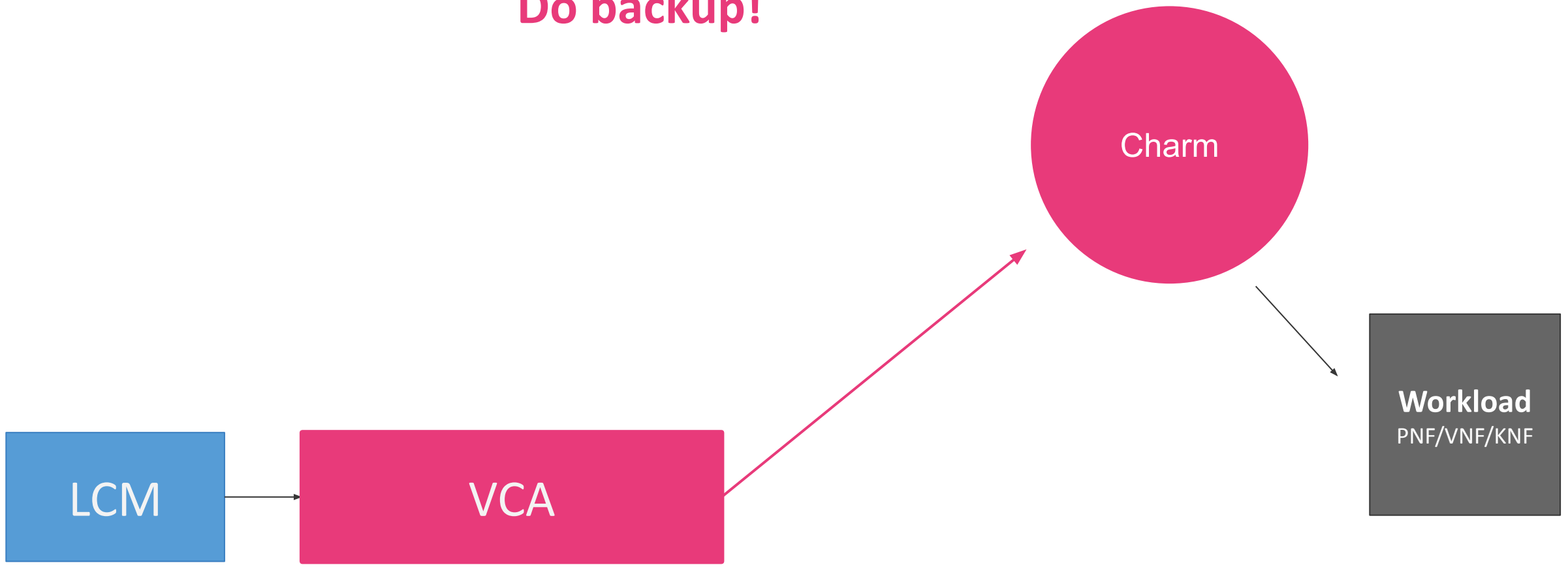


VCA coordinates all OSM Primitives

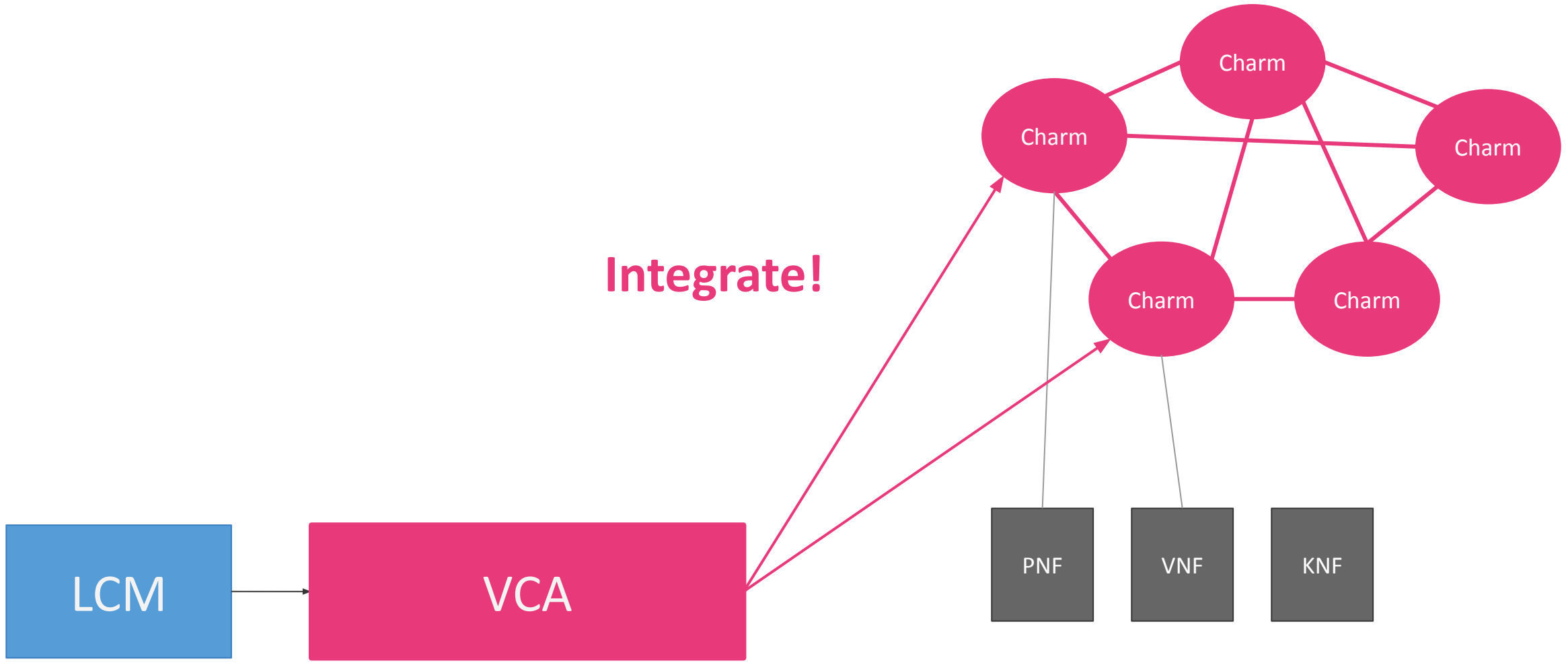
Configuration



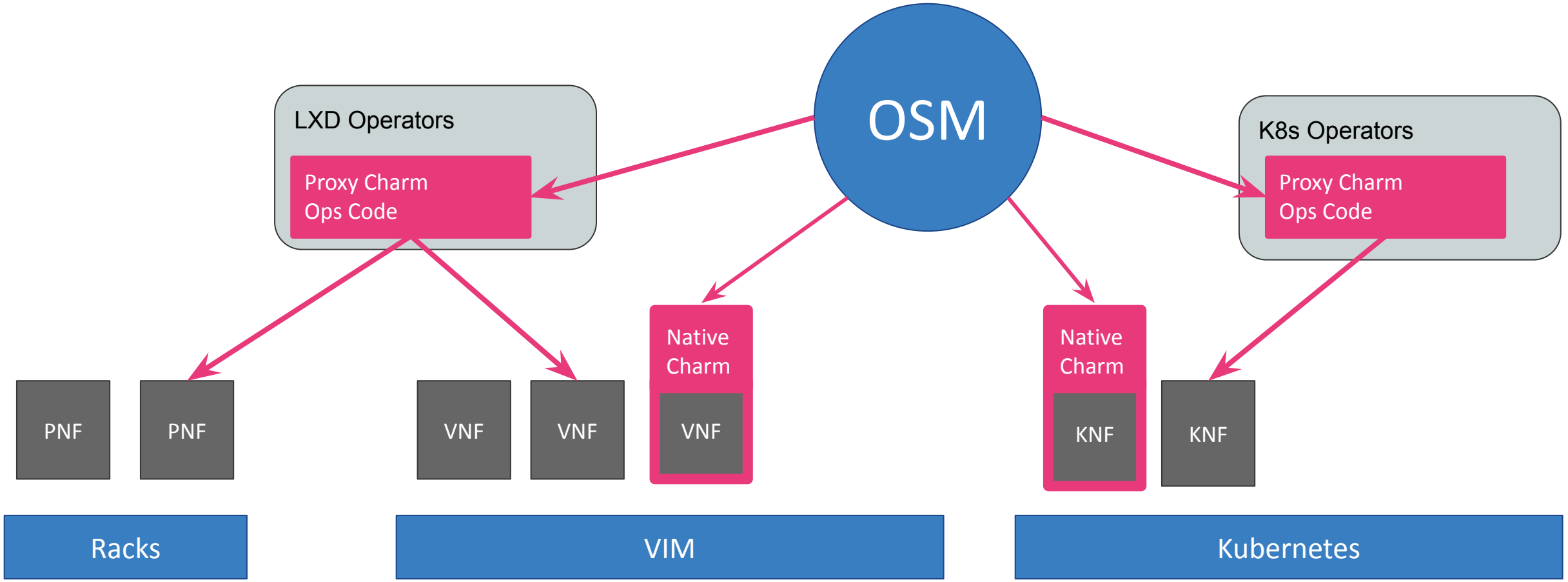
Actions



Integration



Reality is messy and mixed



Open Source
MANO

How operators actually look like?

Create your first charm


```
$ sudo snap install charmcraft --edge
$ mkdir test-operator
$ cd test-operator/
$ charmcraft init --name test
```

Folder structure


```
$ tree
.
├── actions.yaml
├── config.yaml
├── LICENSE
├── metadata.yaml
├── README.md
├── requirements-dev.txt
├── requirements.txt
├── run_tests
├── src
│   └── charm.py
└── tests
    ├── __init__.py
    └── test_charm.py
```

metadata.yaml


```
name: test
description: |
  This is a test charm
summary: |
  This charm does nothing :/
series: [focal]
```

config.yaml


```
options:
  log_level:
    default: INFO
    description: |
      Possible values are:
       - DEBUG
       - INFO
       - CRITICAL
       - ERROR
       - WARNING
    type: string
```

actions.yaml


```
create-directory:
  description: Creates a directory
  params:
    path:
      description: "Path to the directory"
      type: string
      default: ""
  required:
   - path
```

src/charm.py


```
class TestCharm(CharmBase):
    """Charm the service."""

    def __init__(self, *args):
        super().__init__(*args)
        self.framework.observe(self.on.install, self._on_install)
        self.framework.observe(self.on.start, self._on_start)
        self.framework.observe(self.on.stop, self._on_stop)
        self.framework.observe(
            self.on.create_directory_action, self._on_create_directory_action
        )
[...]
```

src/charm.py


```
class TestCharm(CharmBase):
    """Charm the service."""

    def __init__(self, *args):
        super().__init__(*args)
        self.framework.observe(self.on.install, self._on_install)
        self.framework.observe(self.on.start, self._on_start)
        self.framework.observe(self.on.stop, self._on_stop)
        self.framework.observe(
            self.on.create_directory_action, self._on_create_directory_action
        )

    def _on_install(self, event):
        subprocess.run(["snap", "install", "hello"])
[...]
```

src/charm.py


```
class TestCharm(CharmBase):
    """Charm the service."""

    def __init__(self, *args):
        super().__init__(*args)
        self.framework.observe(self.on.install, self._on_install)
        self.framework.observe(self.on.start, self._on_start)
        self.framework.observe(self.on.stop, self._on_stop)
        self.framework.observe(
            self.on.create_directory_action, self._on_create_directory_action
        )
[...]

    def _on_start(self, event):
        subprocess(["snap", "start", "hello"], check=False)
[..]
```

src/charm.py


```
class TestCharm(CharmBase):
    """Charm the service."""

    def __init__(self, *args):
        super().__init__(*args)
        self.framework.observe(self.on.install, self._on_install)
        self.framework.observe(self.on.start, self._on_start)
        self.framework.observe(self.on.stop, self._on_stop)
        self.framework.observe(
            self.on.create_directory_action, self._on_create_directory_action
        )
[...]

    def _on_stop(self, event):
        subprocess(["snap", "stop", "hello"], check=False)
[...]
```

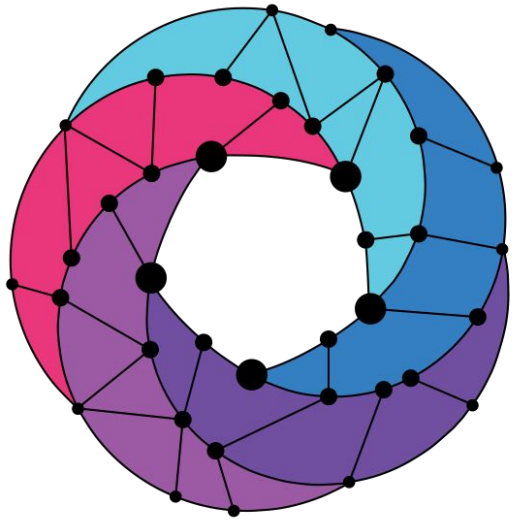
src/charm.py


```
class TestCharm(CharmBase):
    """Charm the service."""

    def __init__(self, *args):
        super().__init__(*args)
        self.framework.observe(self.on.install, self._on_install)
        self.framework.observe(self.on.start, self._on_start)
        self.framework.observe(self.on.stop, self._on_stop)
        self.framework.observe(
            self.on.create_directory_action, self._on_create_directory_action
        )
[...]

    def _on_create_directory_action(self, event):
        path = event.params["path"]
        subprocess.run(["mkdir", "-p", path])
```

Build and deploy the charm

```
$ charmcraft build
$ juju deploy ./test.charm
```

NOTE: For the deploy, we need to bootstrap a juju controller first.



Open Source MANO

Find us at:

osm.etsi.org

osm.etsi.org/wikipub