Selinon

Distributed computing with Python

Fridolín Pokorný
<fridolin@redhat.com>
Twitter: @fridex

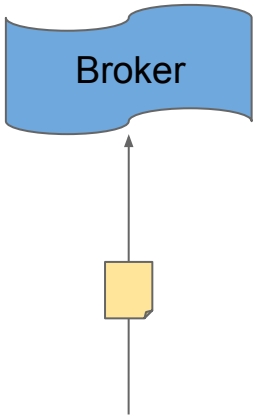
$ whoami

- Fridolín Pokorný
- Brno, Czech republic
- reverse engineering
- now Red Hat AICoE
  - machine learning, big data processing
  - AF_KTLS, Selinon, Thoth

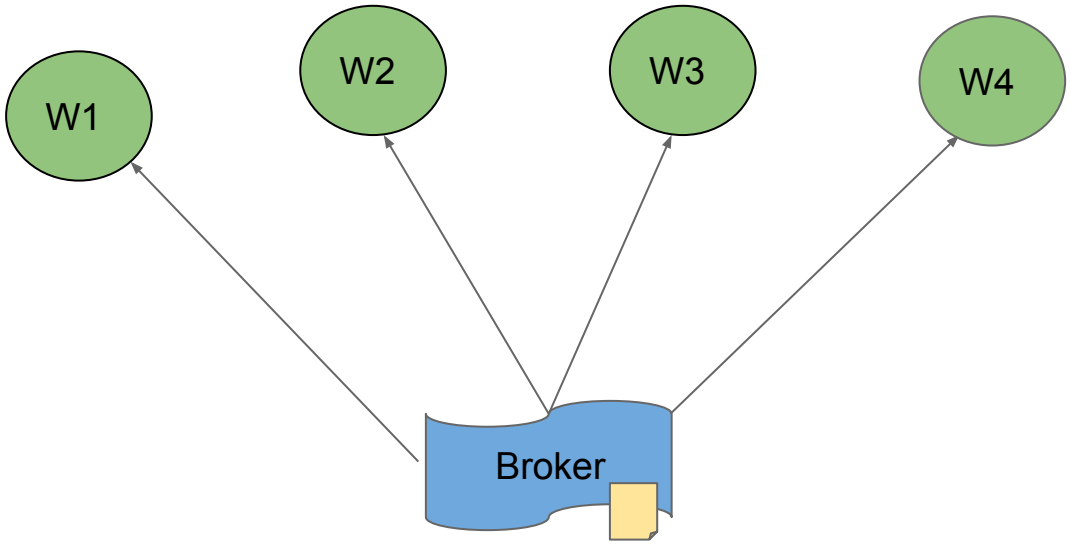
Distributed Computing!

Celery Overview

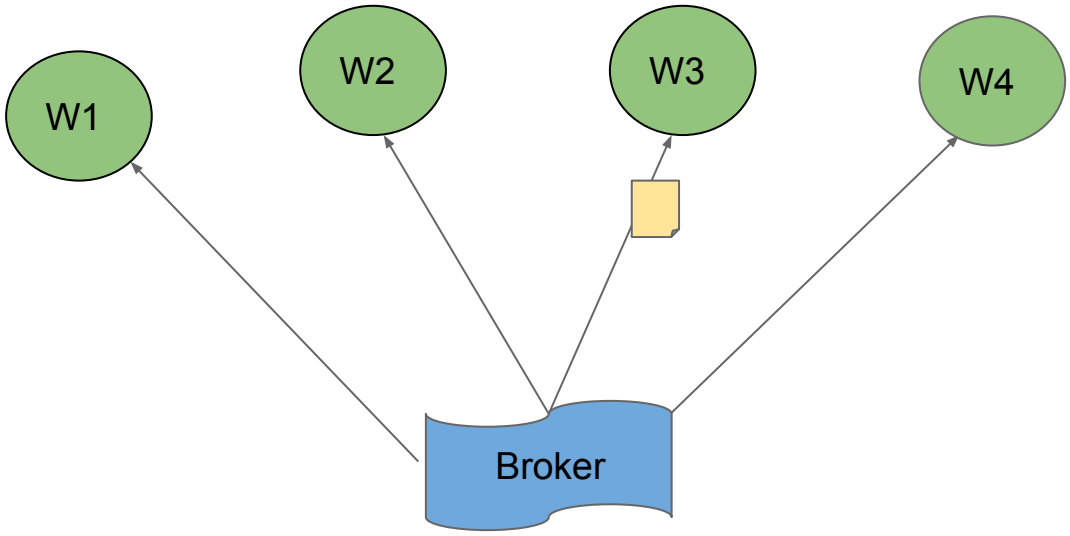
Celery Overview



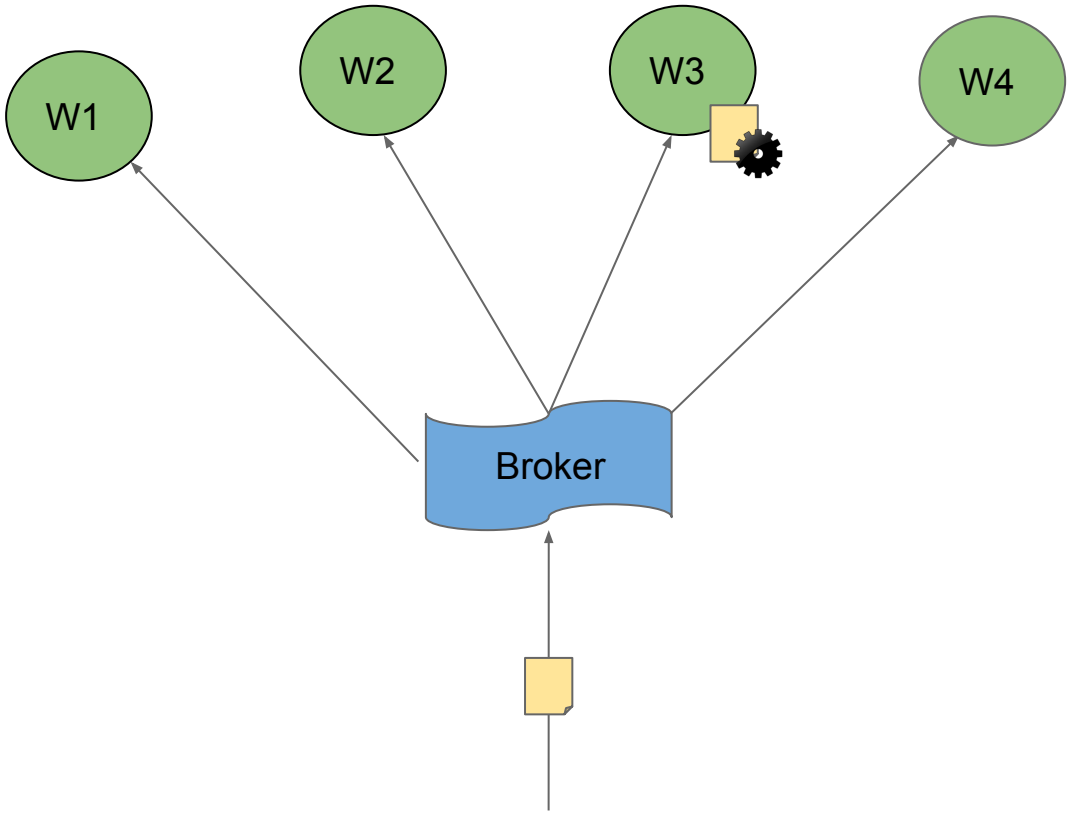
Celery Overview



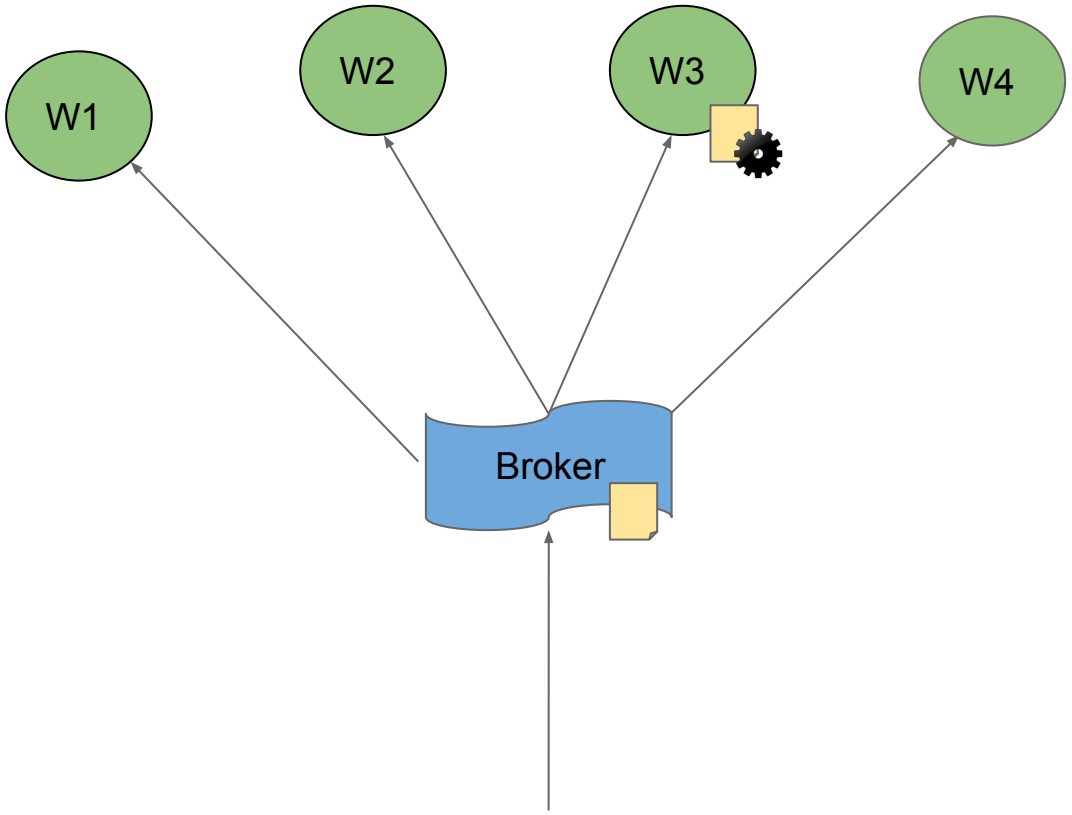
Celery Overview



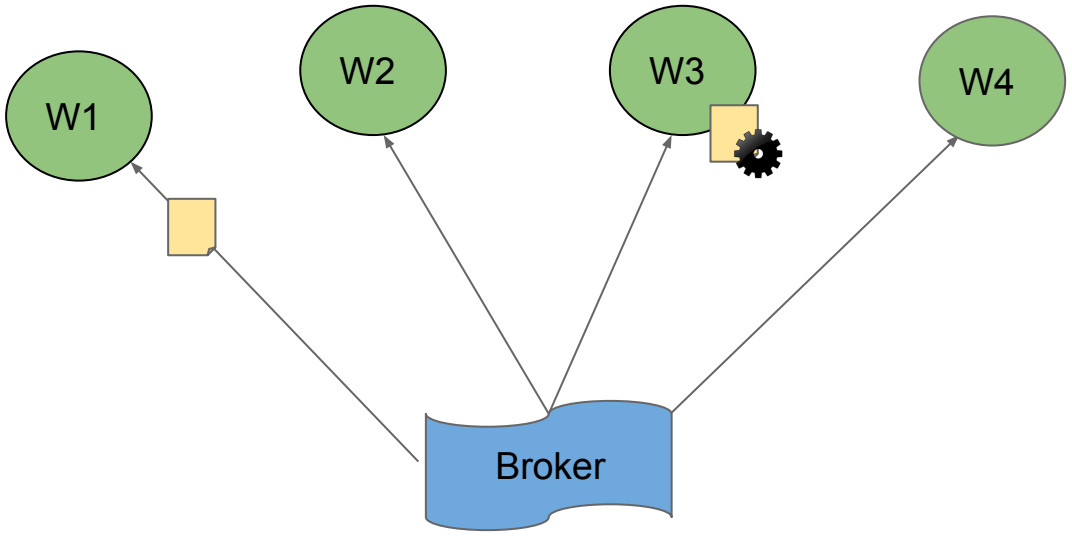
Celery Overview



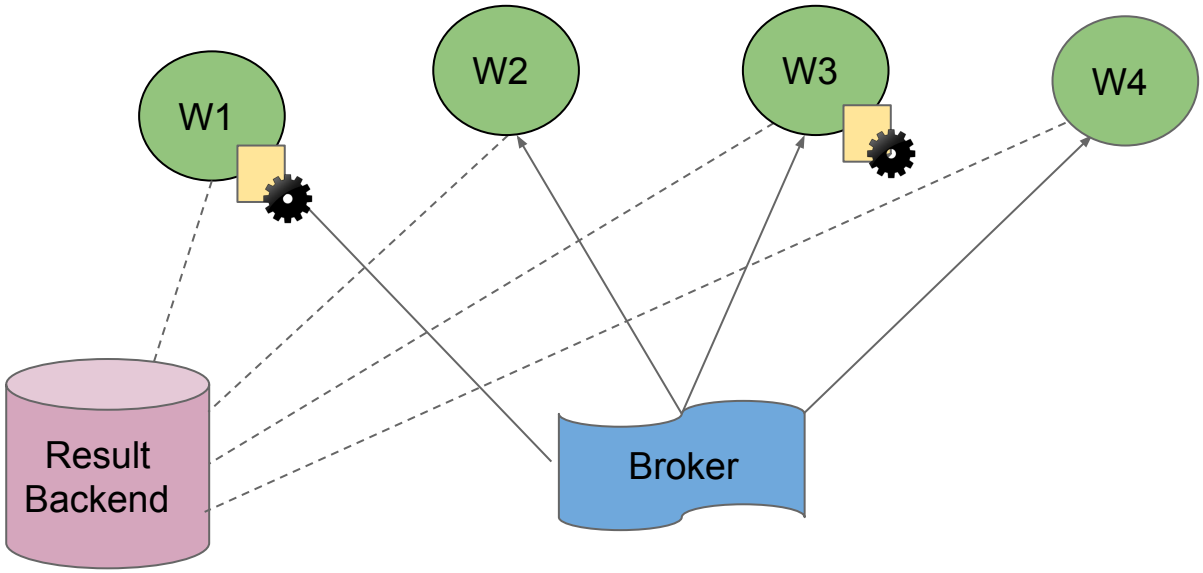
Celery Overview



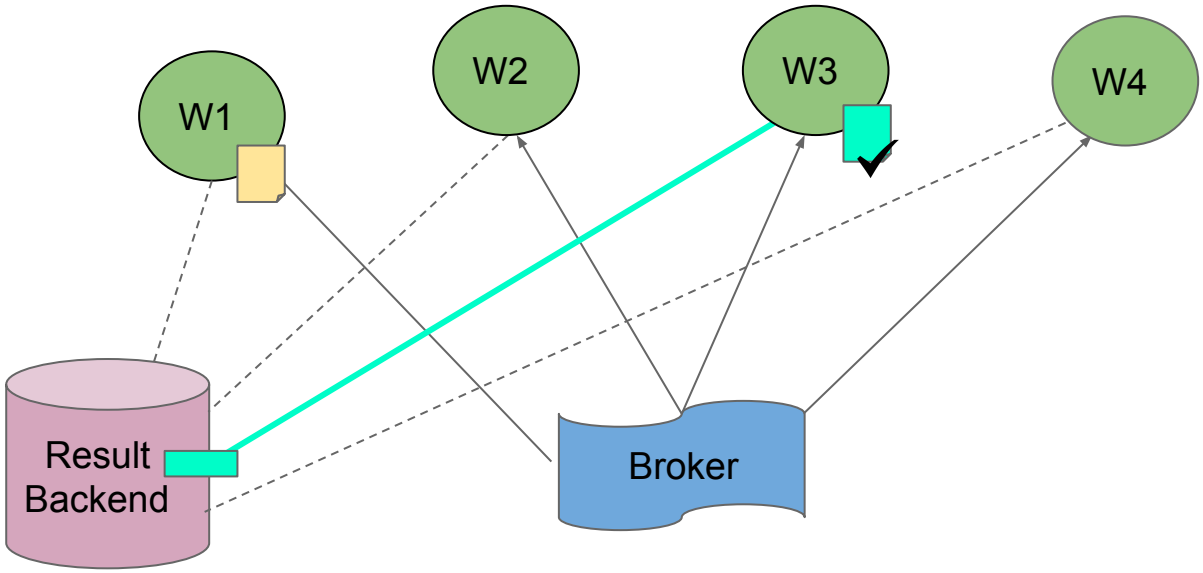
Celery Overview



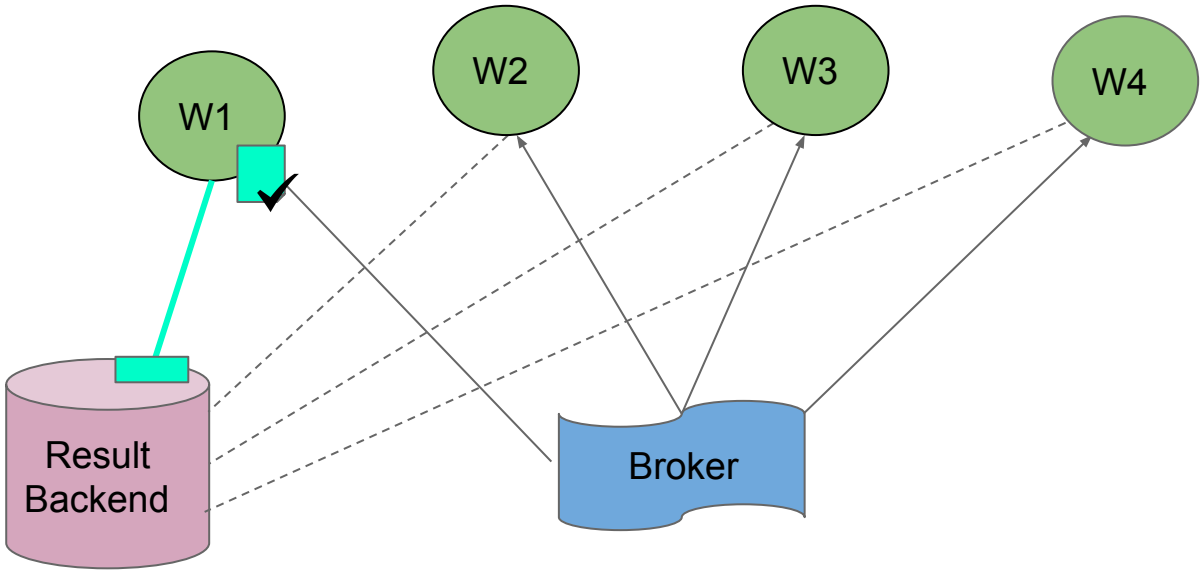
Celery Overview



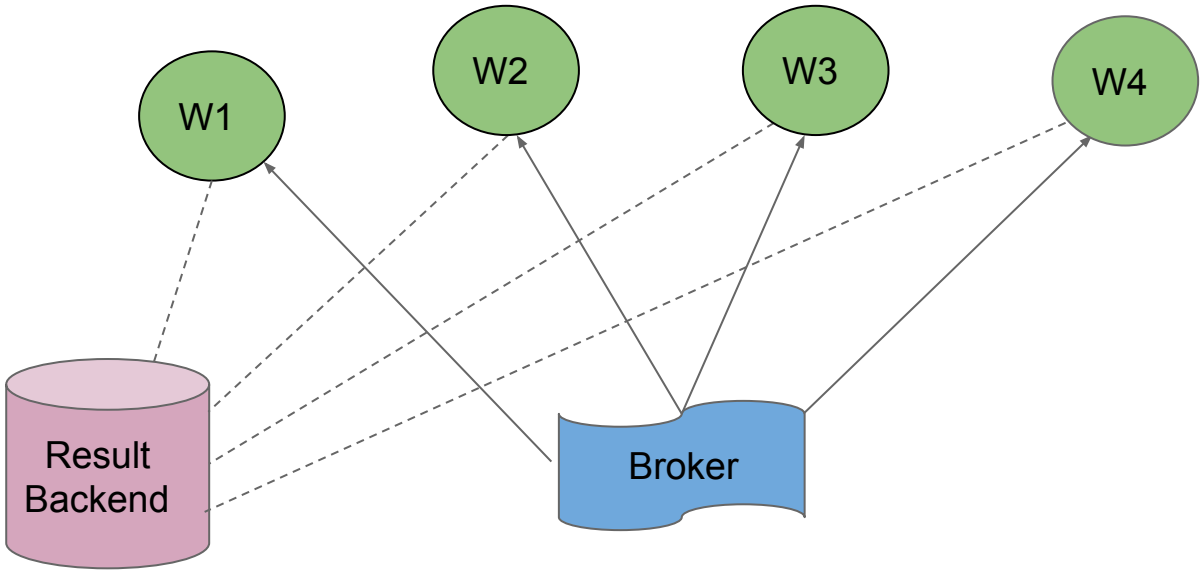
Celery Overview



Celery Overview



Celery Overview

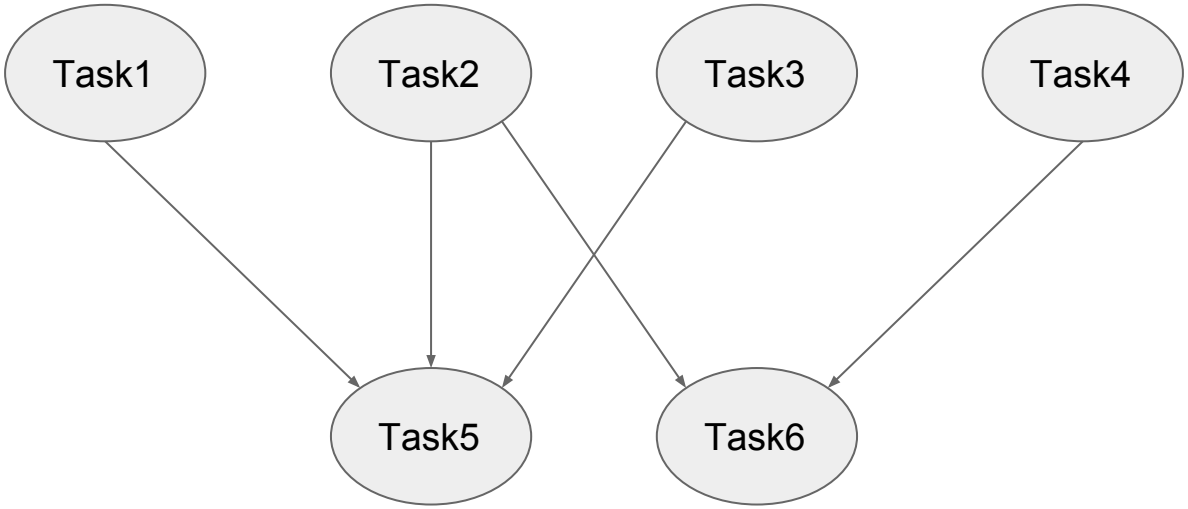


Celery Project

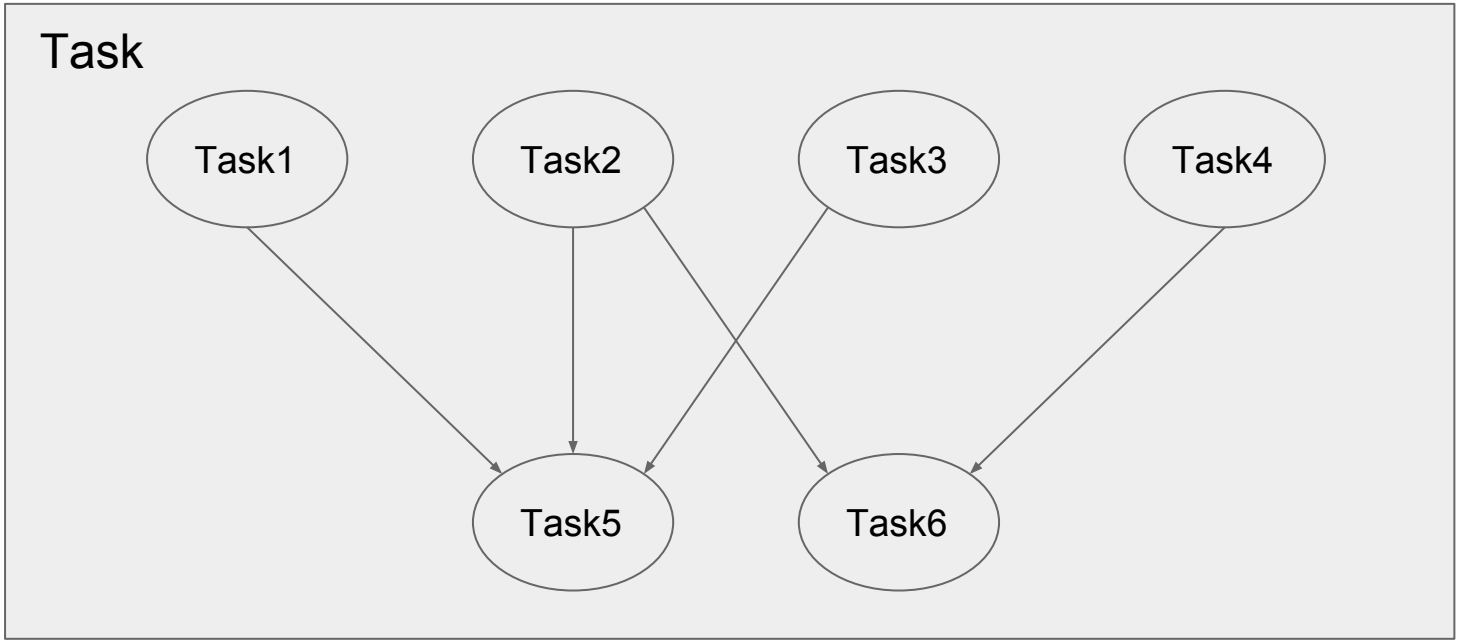
- Celery project
  - http://celeryproject.org/
- Distributed task queue
- Django Celery

Task Flow!

Flow Design



Flow Design

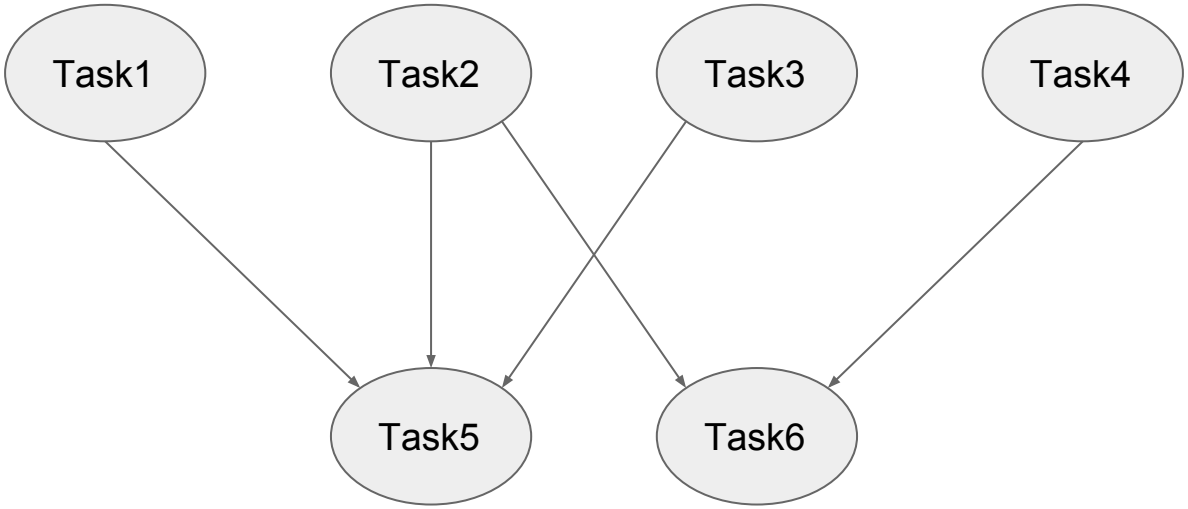


Dependencies Between Tasks - Flows

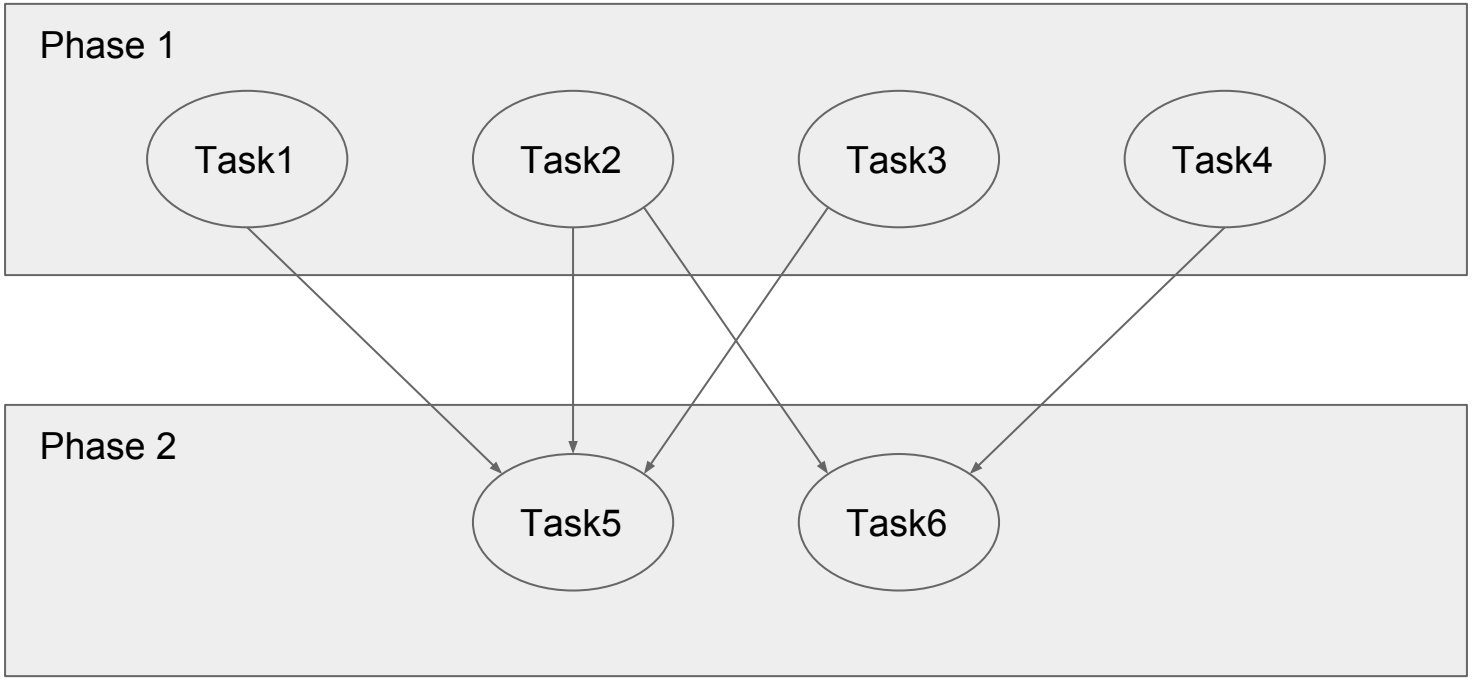
- “Celery primitives”

- Group
- Chain
- Chord
- Map
- Starmap
- Chunks

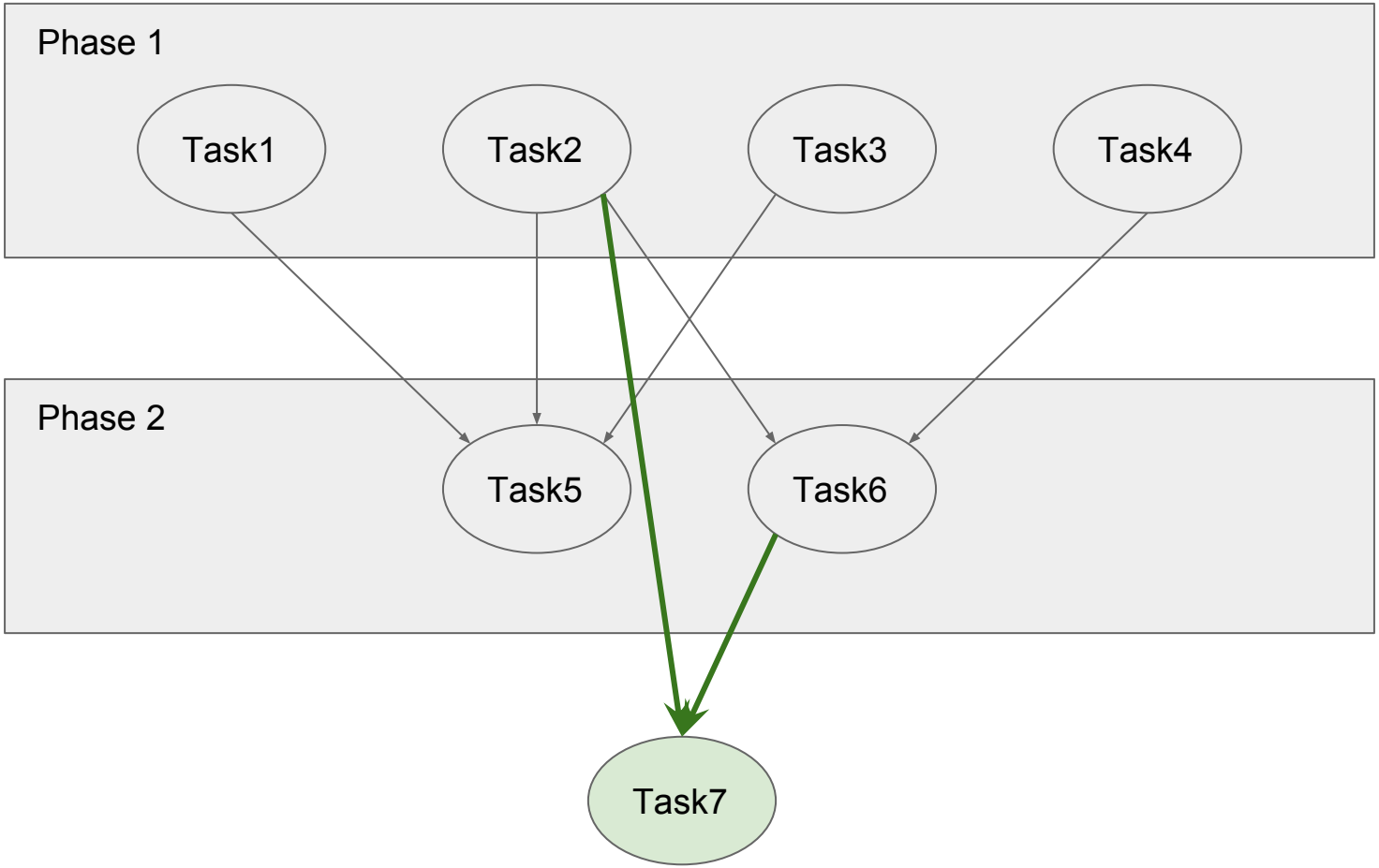
Flow Design



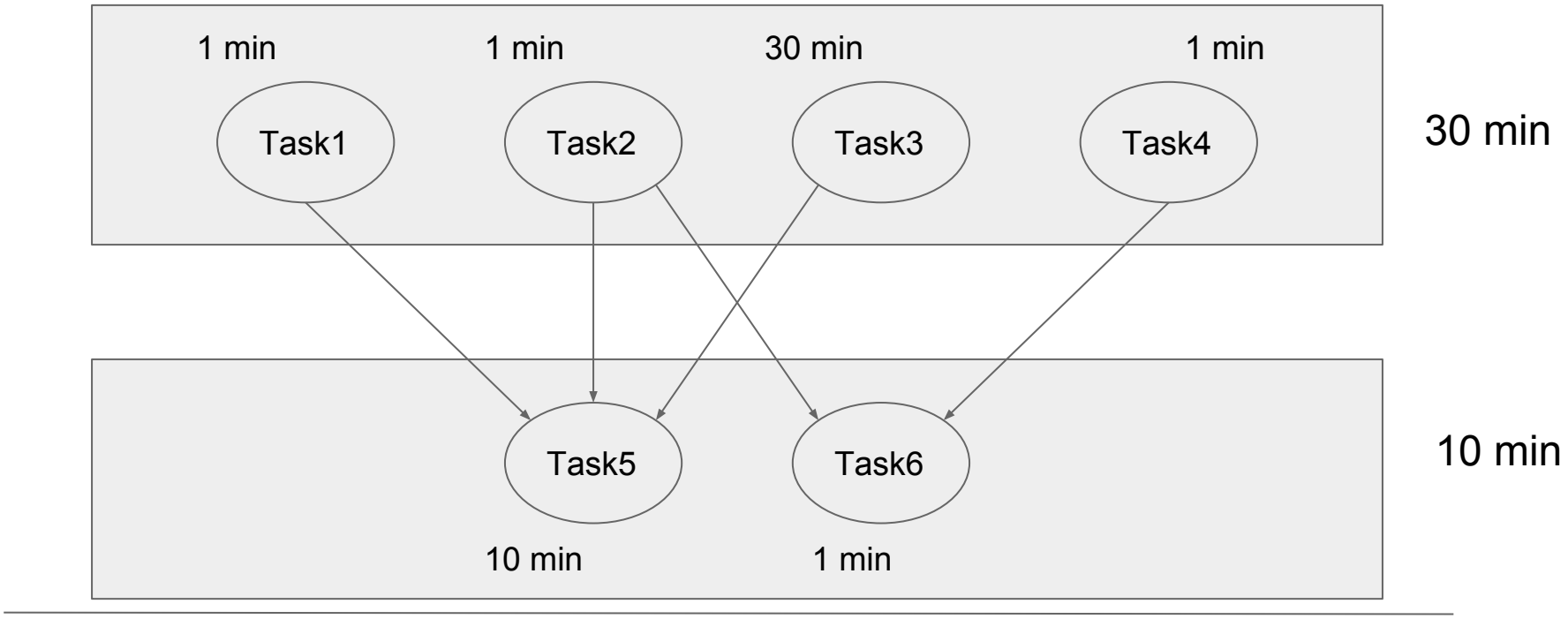
Flow Design



Flow Design

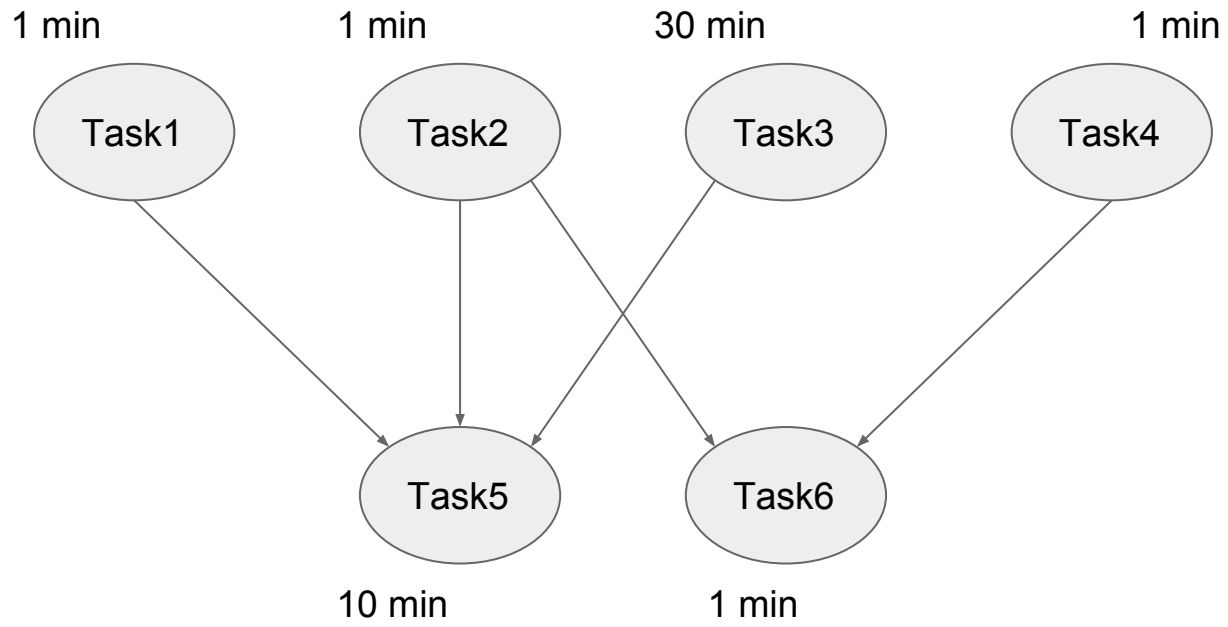


Flow Design

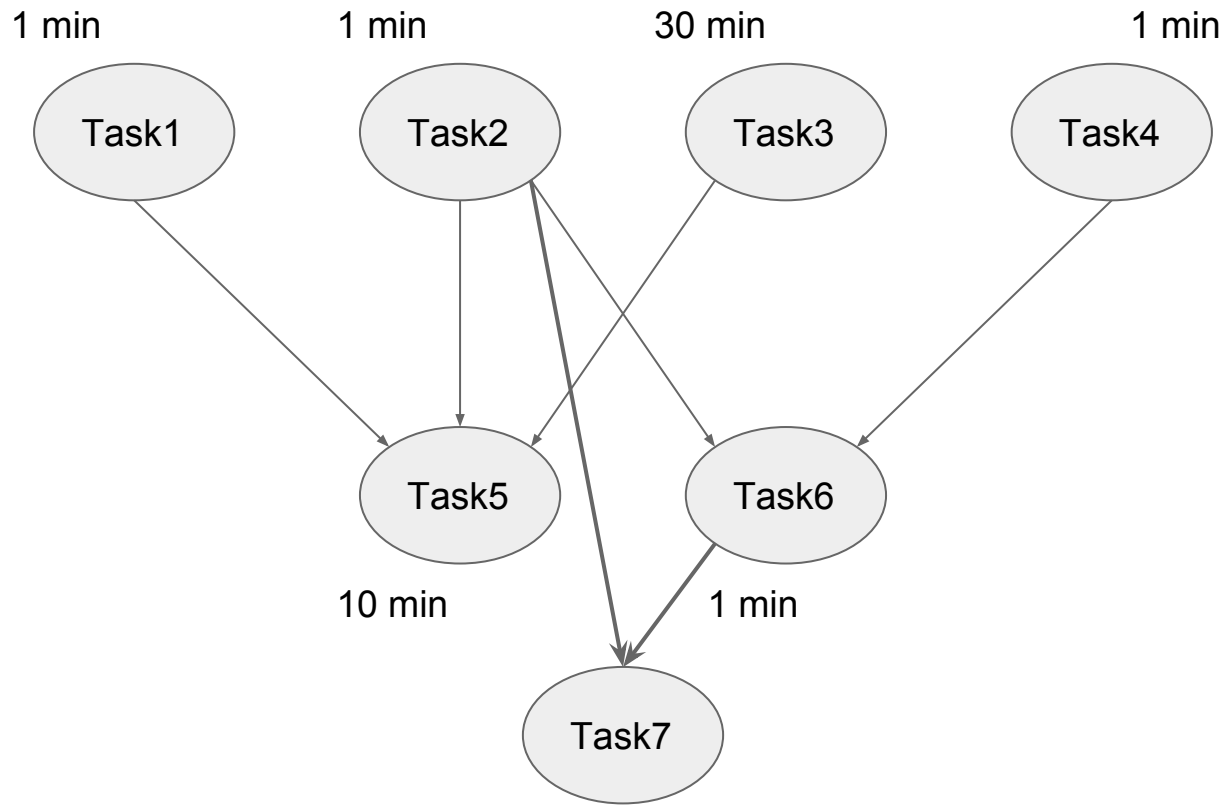


Total: 40 min

Flow Design


1 min
1 min
30 min
1 min
Task1
Task2
Task3
Task4
Task5
Task6
10 min
1 min
40 min
2 min

Flow Design


1 min
1 min
30 min
1 min
Task1
Task2
Task3
Task4
Task5
Task6
10 min
1 min
Task7

Pitfalls

- Adding new tasks breaks the design
- Complex, not straightforward
- Hard-coded logic
- What about task failures?
- Reusability of task implementation?
- Different storages/databases?
- ...

Introducing Selinon

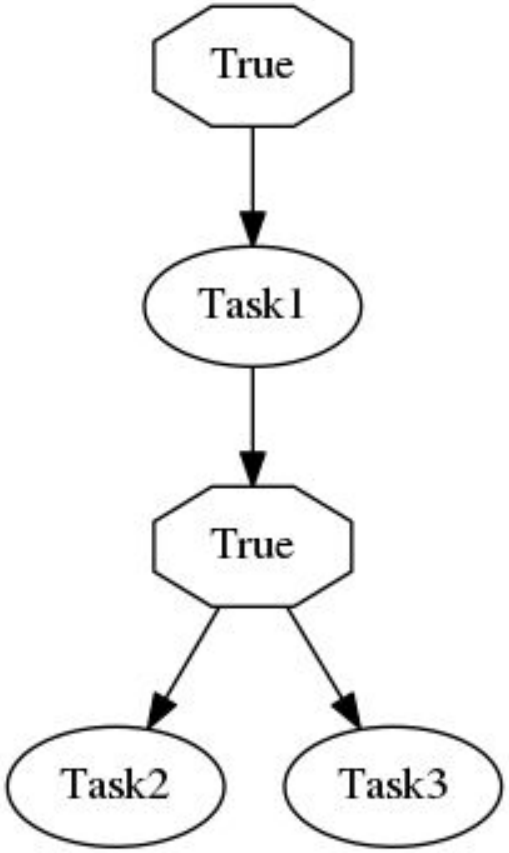
- *Selinon* means celery in Greek
- Separate task flow logic into YAML files
- Grouping tasks into flows
- Create graph of dependencies between:
  - Tasks
  - Flows
  - Task & Storages
  - Fallback tasks

Selinon Task

```
from selinon import SelinonTask


class Task1(SelinonTask):

    def run(self, node_args):
        res = node_args[“A“] * node_args[“B“]
        return {“foo“: res}
```

YAML Configuration

```
tasks:
  - name: Task1
    import: myproject.tasks

  -  ...

flow-definitions:
  - name: flow1
    edges:
       - from:
         to: Task1

       - from: Task1
         to:
           - Task2
           - Task3
```

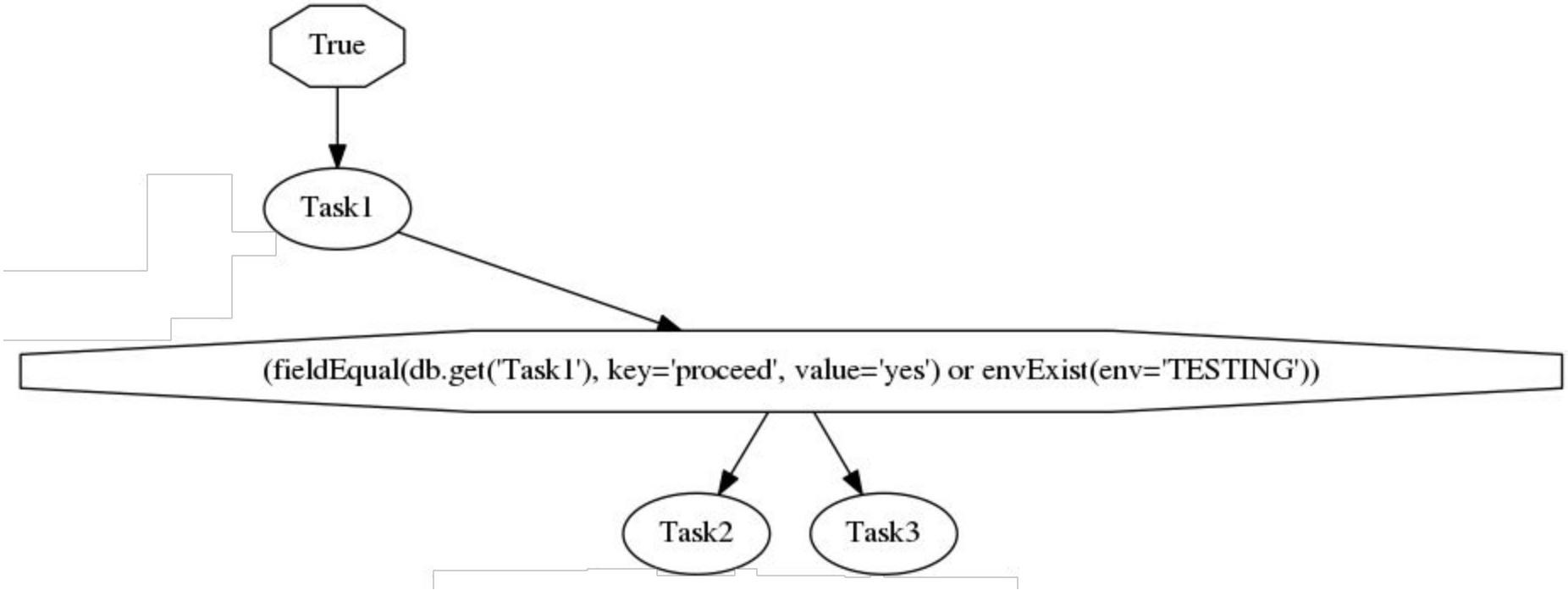


Conditions

Conditions

```
flow-definitions:
   …
      edges:
         - from: Task1
           to:
             - Task2
             - Task3
           condition:
            or:
              - name: fieldEqual
                node: Task1
                args:
                  key: proceed
                  value: yes

              - name: envExist
                args:
                  env: TESTING
```

Conditions


True
Task1
(fieldEqual(db.get('Task1'), key='proceed', value='yes') or envExist(env='TESTING'))
Task2
Task3

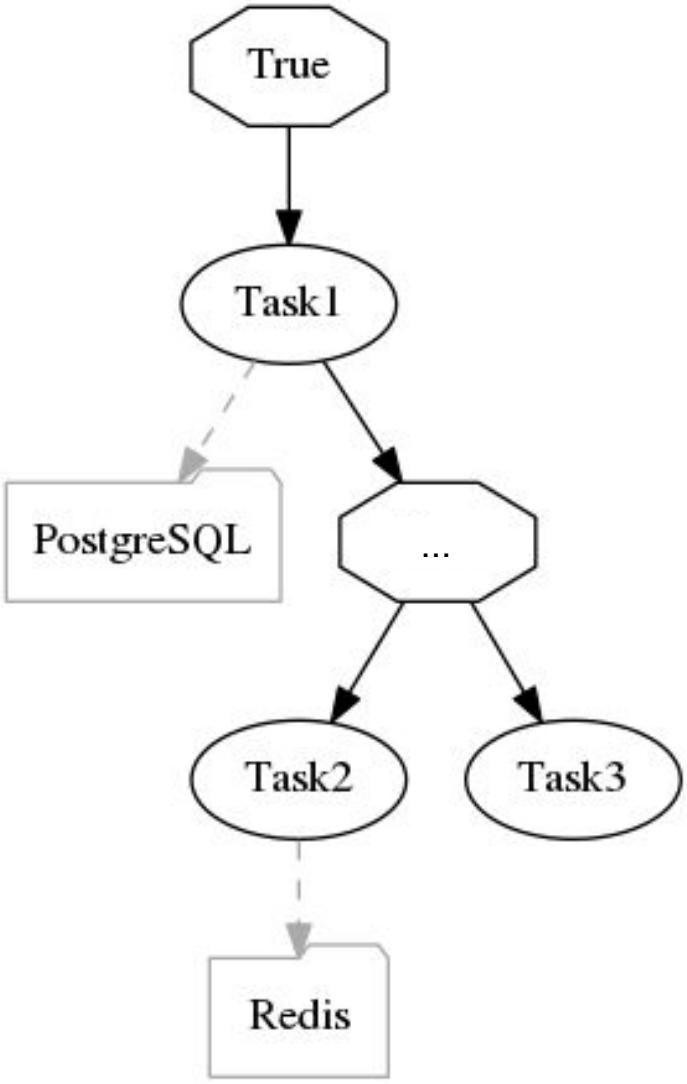
Storages & databases

Selinon Data Storage

```
from selinon import DataStorage


class Redis(DataStorage):

    def connect(self, ...):
        ...

    def retrieve(self, ...):
        ...

    def store(self, ...):
        ...
```

Storages & databases

```
tasks:
  - name: Task1
    import: myproject.tasks
    storage: PostgreSQL

  - name: Task2
    import: myproject.tasks
    storage: Redis
  - ...

storages:
  - name: PostgreSQL
    import: myproject.db
    configuration: ...

  - name: Redis
    import: myproject.db
    configuration: ...
```

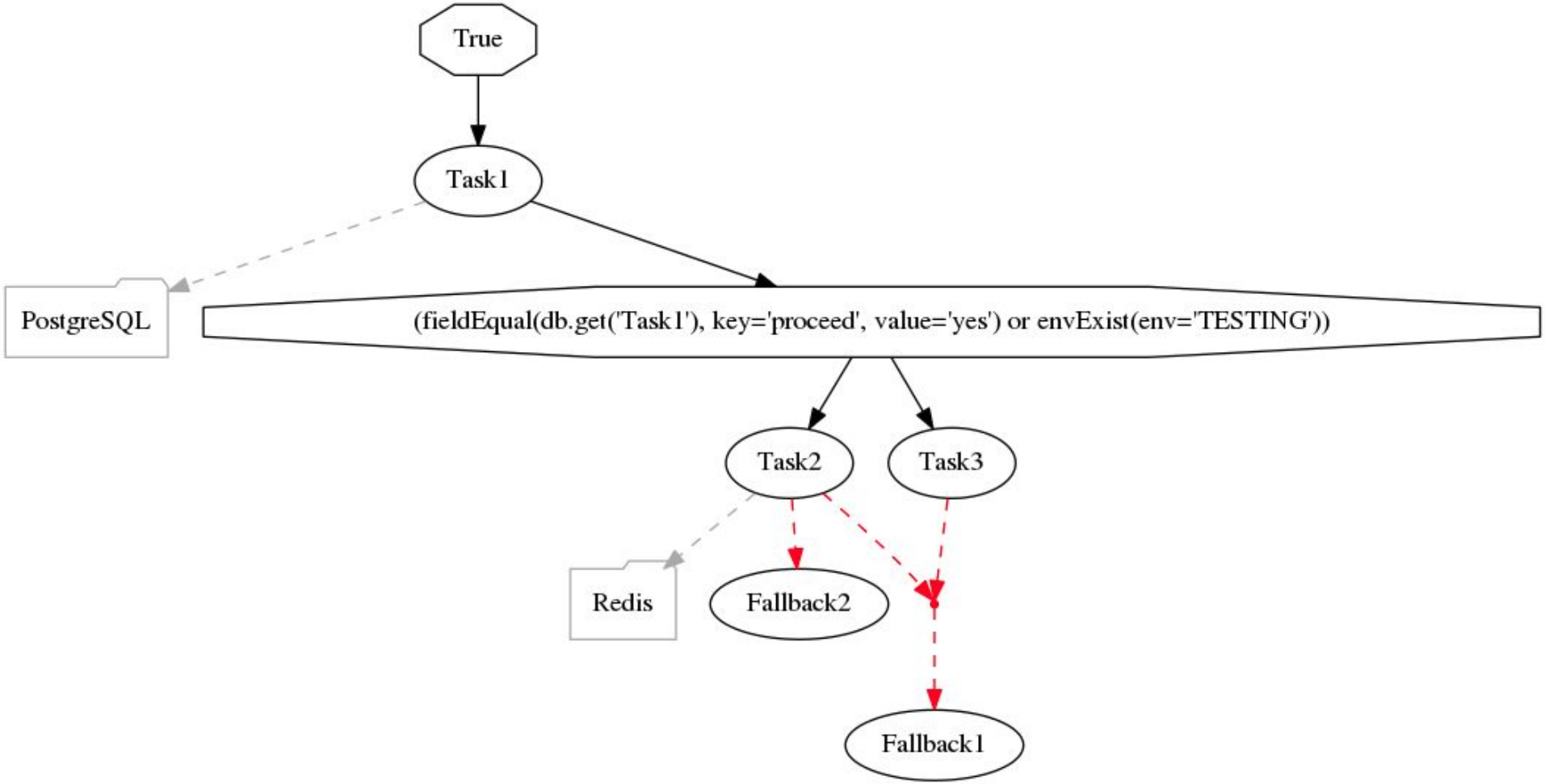


Fallback Tasks & Fallback flows

Fallback tasks & Fallback flows

```
flow-definitions:
    …
      edges:
        ...
       failures:
         - nodes:
             - Task2
             - Task3
           fallback:
             - Fallback1

         - nodes:
             - Task2
           fallback:
             - Fallback2
```

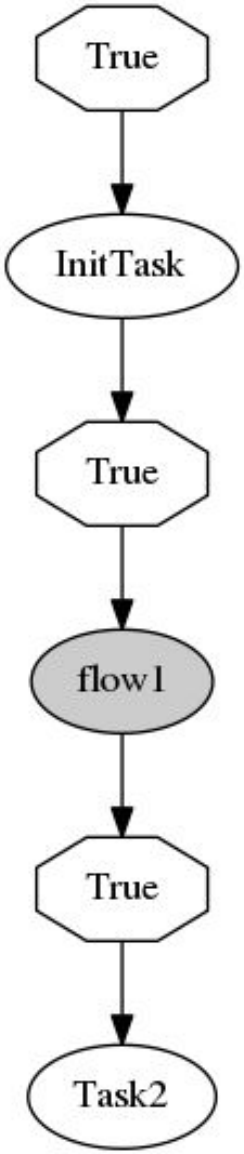
Fallback tasks and flows



Subflows

YAML Configuration

```yaml
flow-definitions:
  - name: flow2
    edges:

      - from:
        to: InitTask

      - from: InitTask
        to: flow1

      - from: flow1
        to: Task2
```



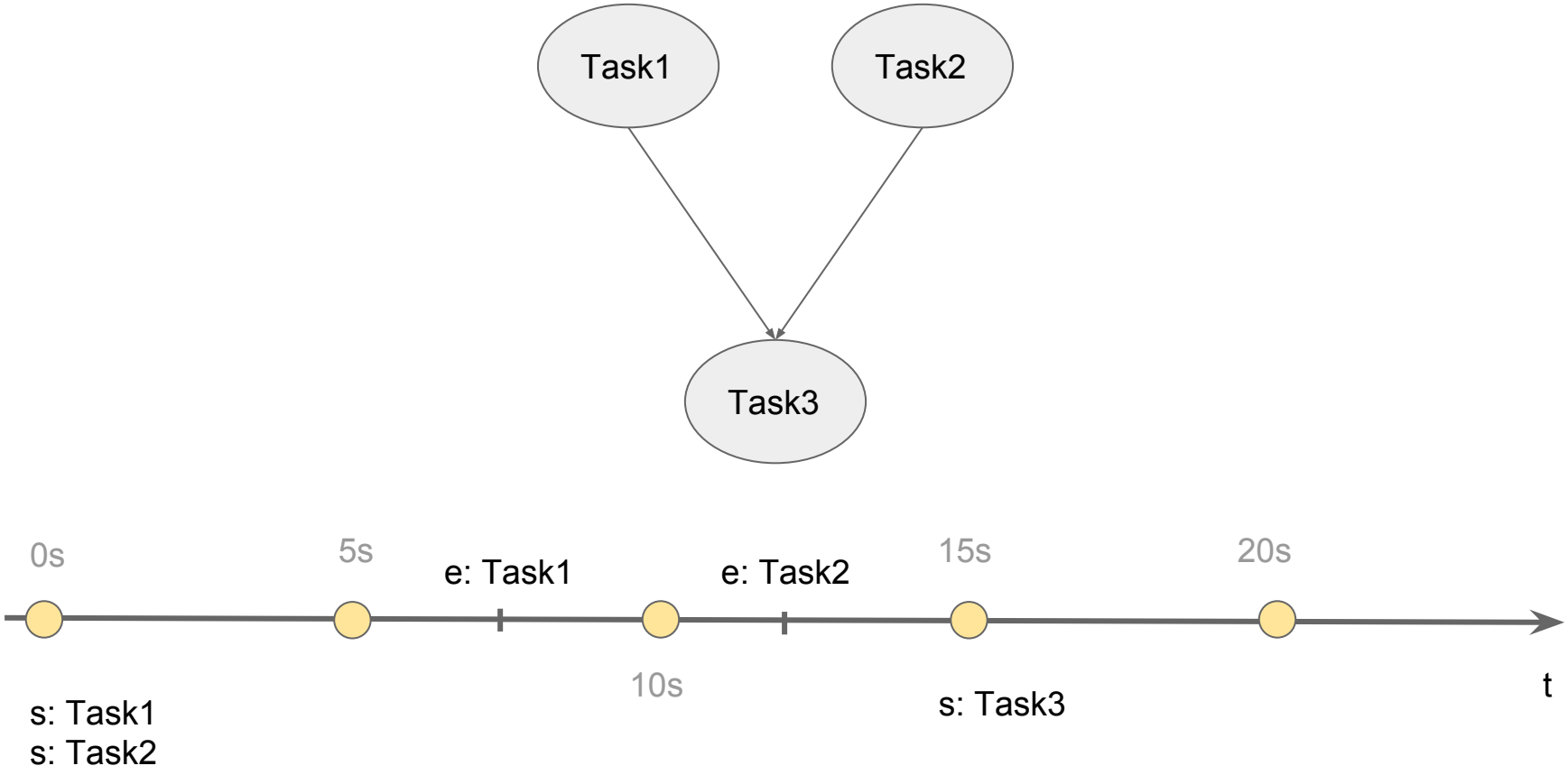
How does SELINON work?

Selinon

- Key idea: *Dispatcher task*
  - Periodically scheduled based on configuration
  - Check the current state of the flow
  - Schedule new tasks if needed
  - Flow of tasks, flow of arguments
- YAML configuration files
  - Reusability of flows (nodes)
  - Additional system checks
  - Flow visualization
  - ...

Selinon


selinon


Task1
Task2
Task3
0s
5s
e: Task1
10s
e: Task2
15s
20s
t
s: Task1
s: Task2
s: Task3

Other features

- Caches
- Task and flow throttling
- Task and flow prioritization
- Optimization of Dispatcher scheduling
- Tracepoints
- Migrations
- Selective task flows - flow replay
- ...

MIGRATIONS

- Changes in the flow structure (runtime)
- Tainted flows:
  - Retry
  - Ignore
  - Fail

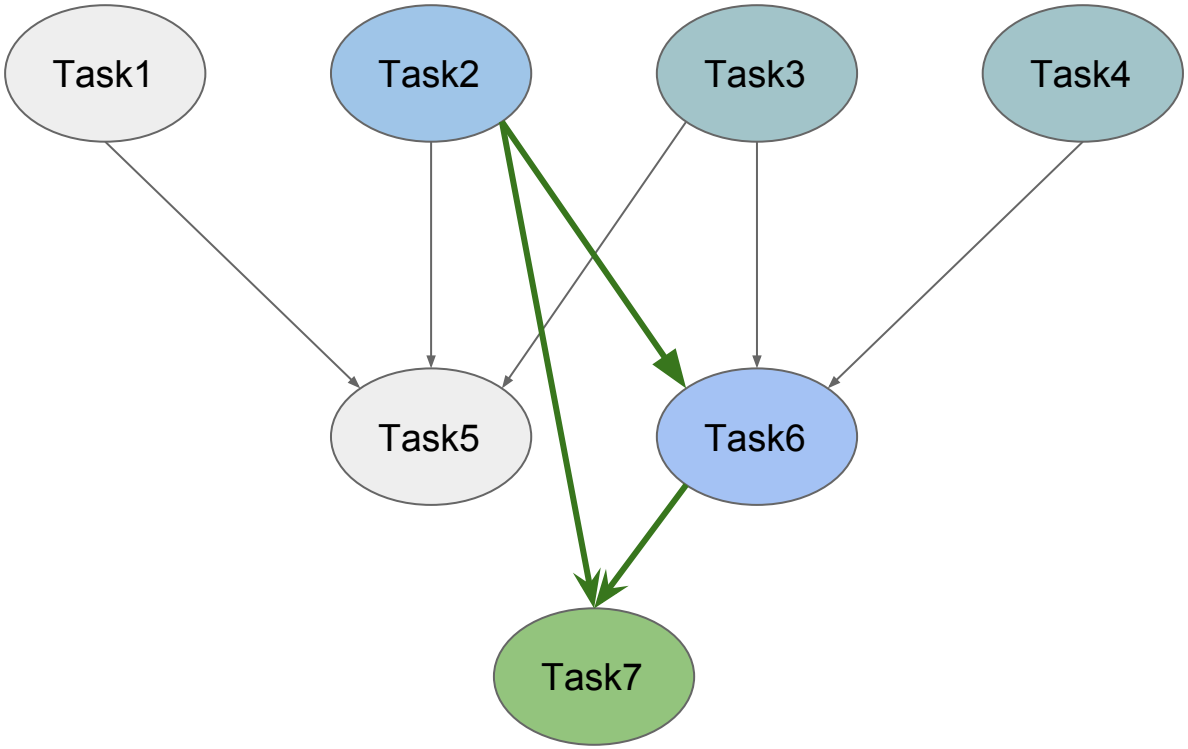
```
flow-definitions:
  - name: flow2
    edges:

      - from:
        to: InitTask

      - from: InitTask
        to: ~~flow1~~ Task3

      - from: ~~flow1~~ Task3
        to: Task2
```

Selective task run

- Flow replay
- No need to recompute results

Selective task run

- Flow replay
- No need to recompute results



Selinon

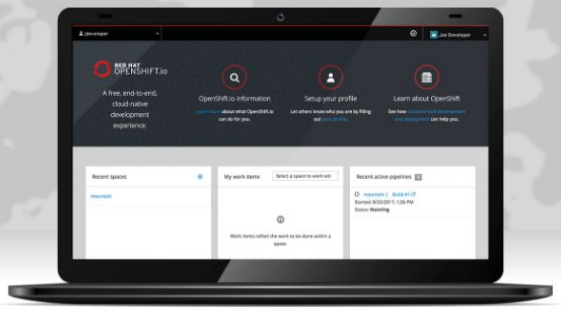
- Built on top of Celery
- Simple YAML configuration
- Separation of task logic and result storing
- Conditional task execution
- Group tasks into flows
- Advanced task flow handling with fallbacks
- System diagnostics based on tracepoints

RED HAT OPENSHIFT.io FEATURES GET INVOLVED DOCUMENTATION LOG IN SIGN UP redhat

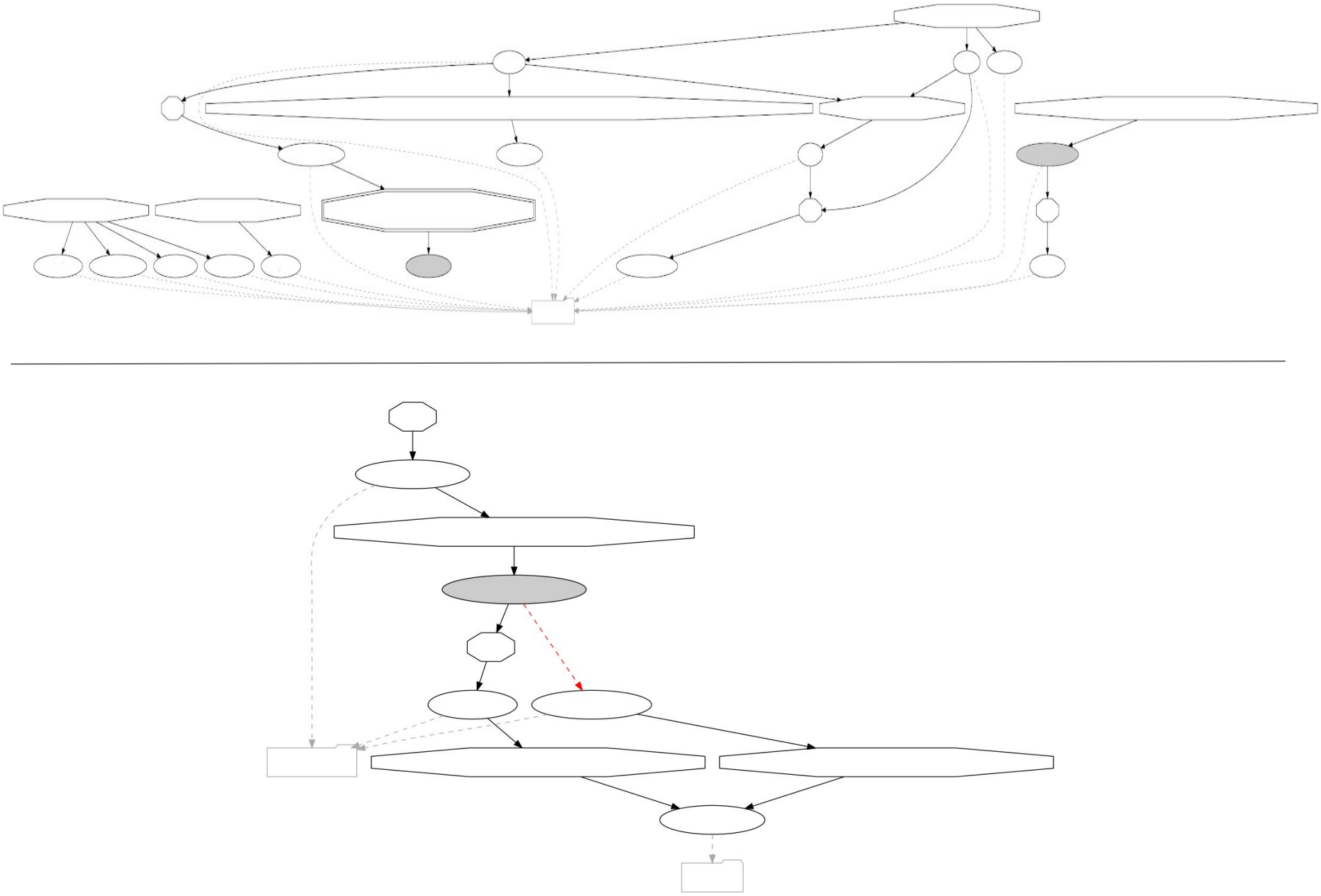
WELCOME TO OPENSHIFT.IO

OpenShift.io is an open online development environment for planning, creating, and deploying hybrid cloud services.

SIGN UP



Learn more about the features



DEMO

https://github.com/selinon/demo-deployment

Questions?

https://github.com/selinon