Selinon

Distributed computing with Python

Fridolín Pokorný
<fridolin@redhat.com>

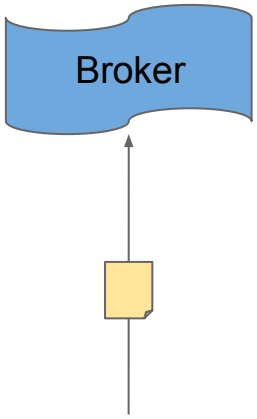
$ whoami

- fridolin@redhat.com
- fresh graduate VUT FIT
- AVG reverse engineering
- now Red Hat
  - AF_KTLS
  - Selinon

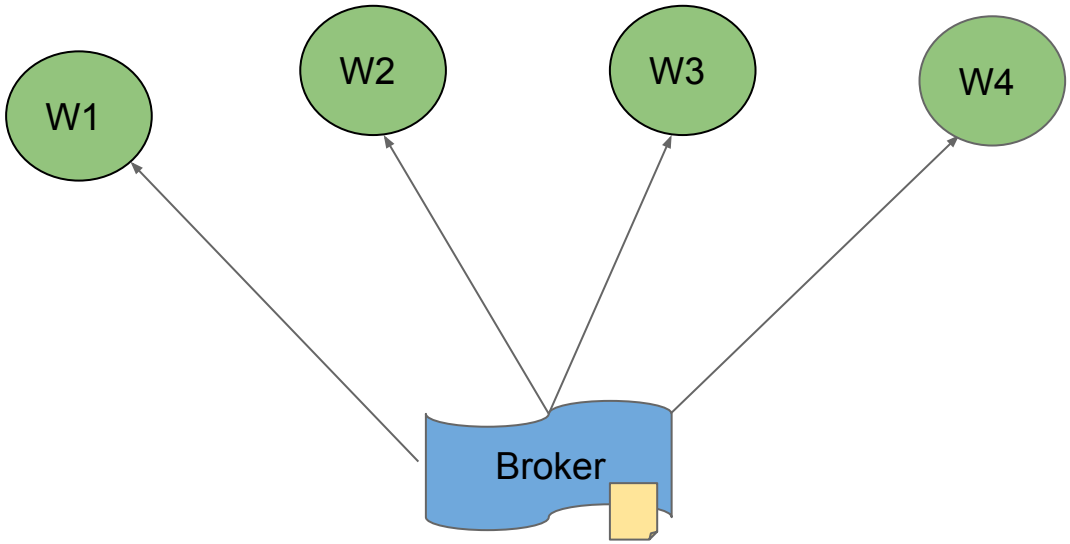
Distributed Computing!

Celery Overview

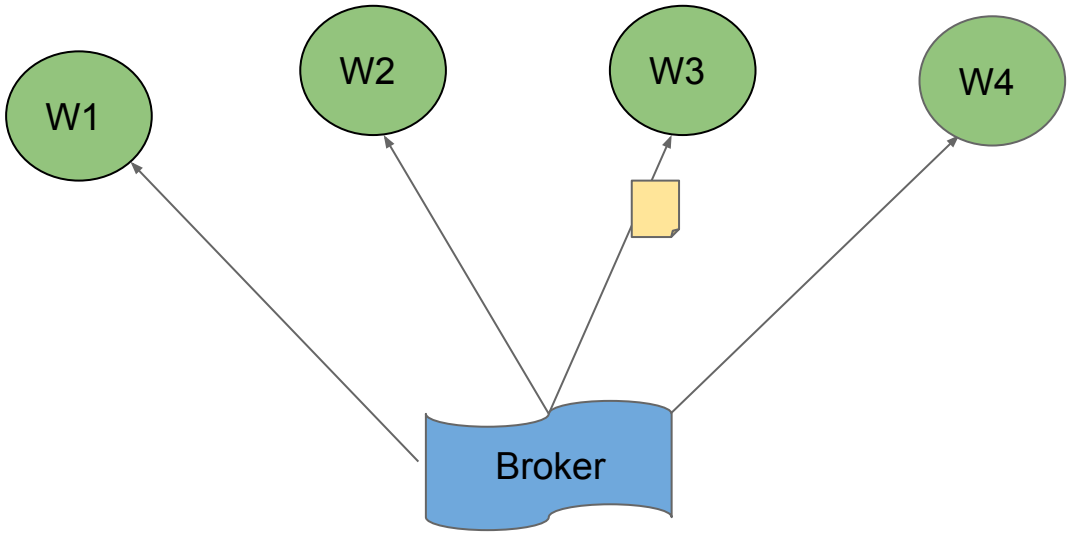
Celery Overview



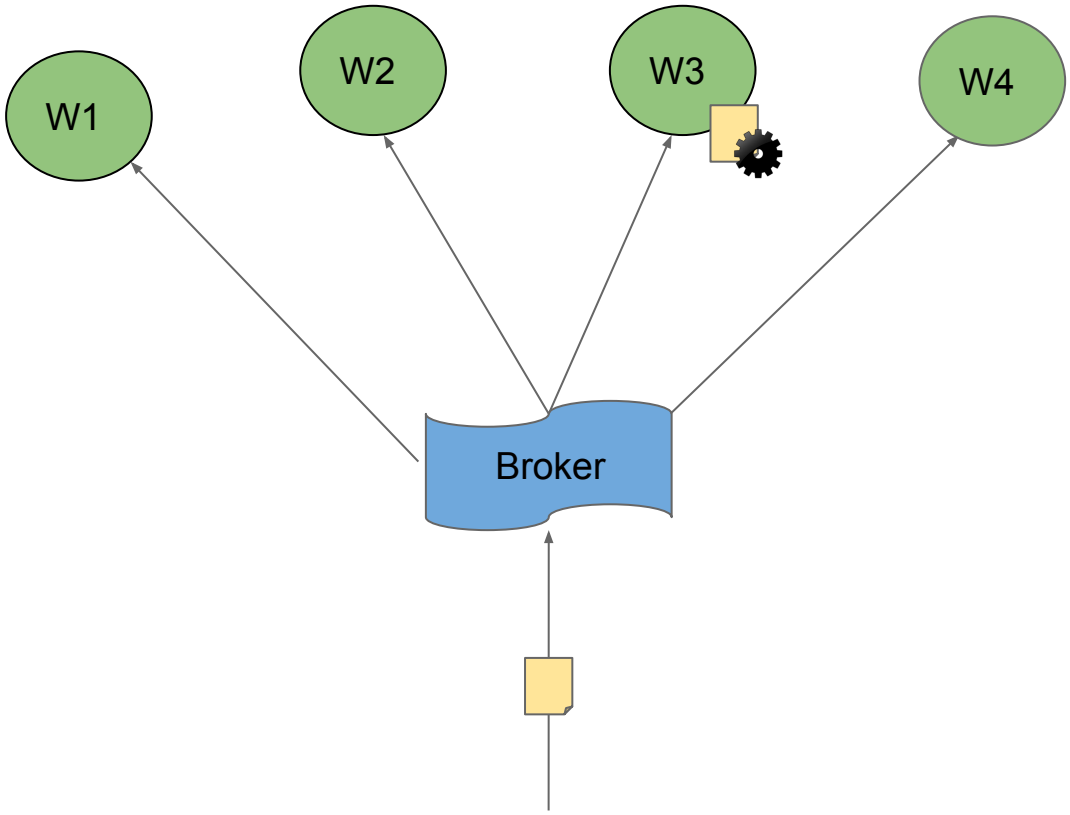
Celery Overview



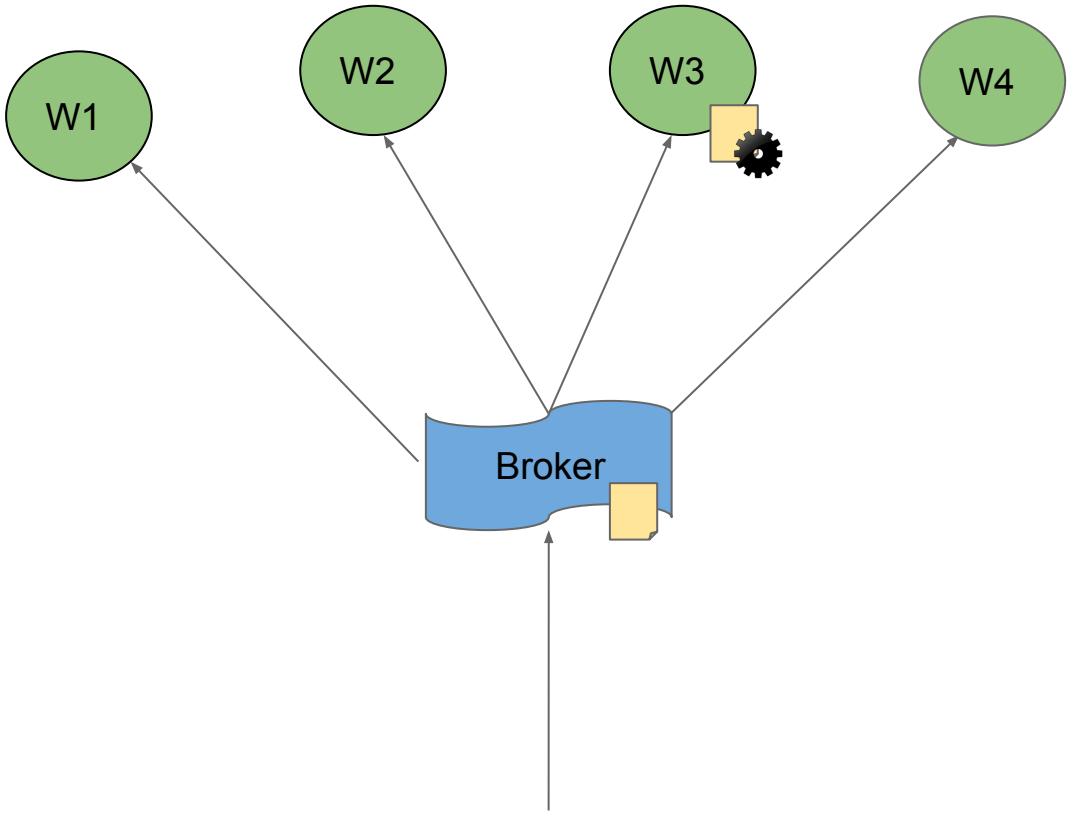
Celery Overview



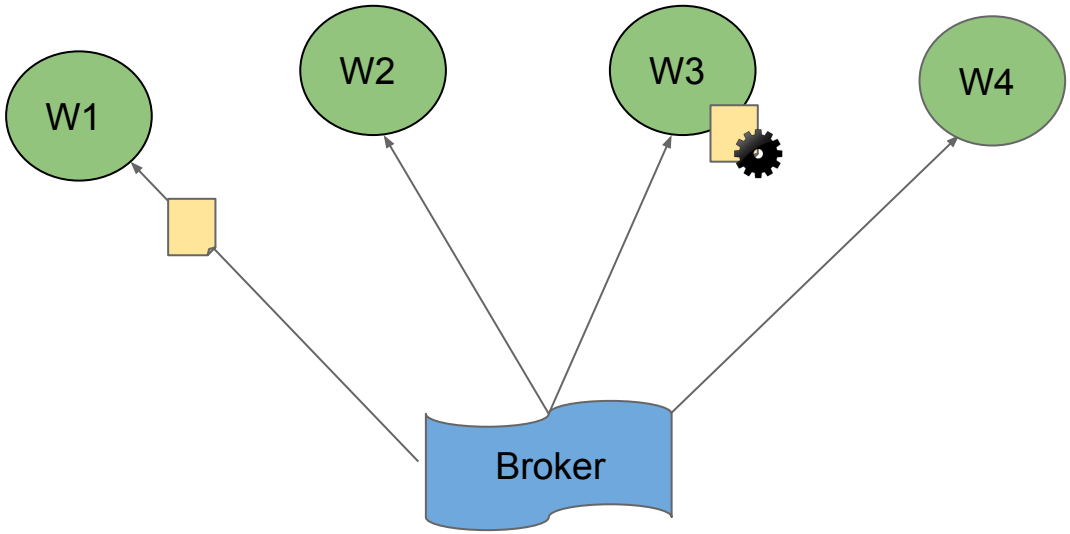
Celery Overview



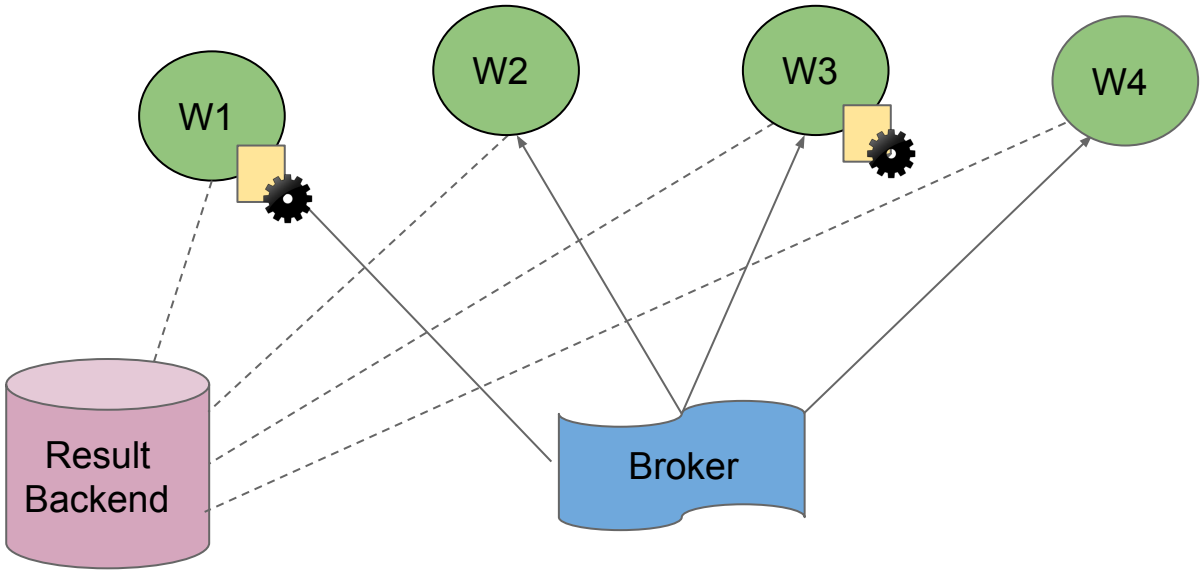
Celery Overview



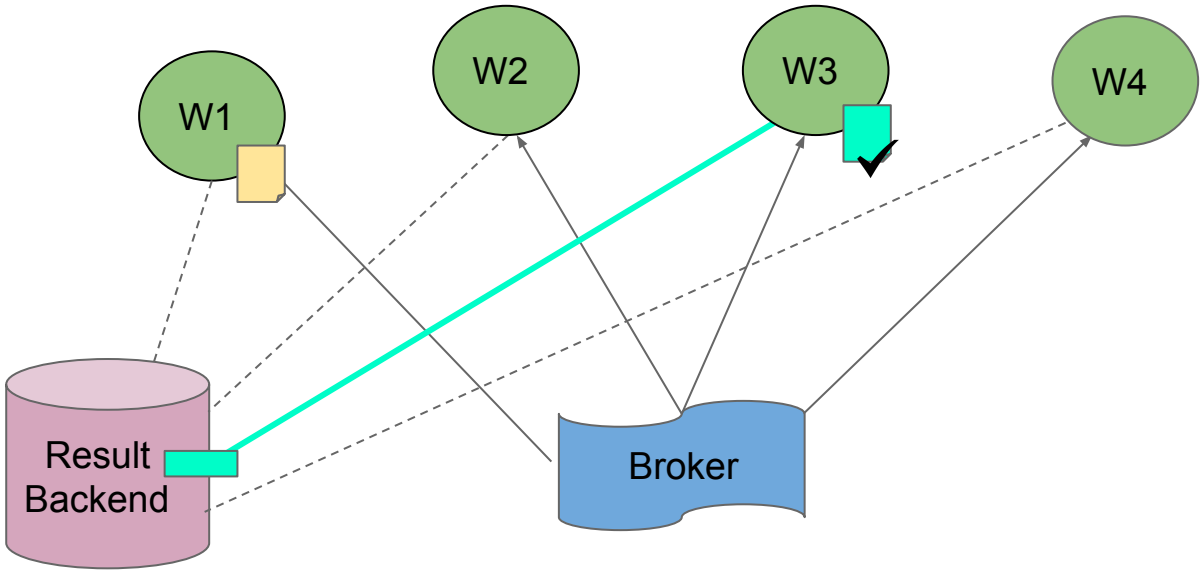
Celery Overview



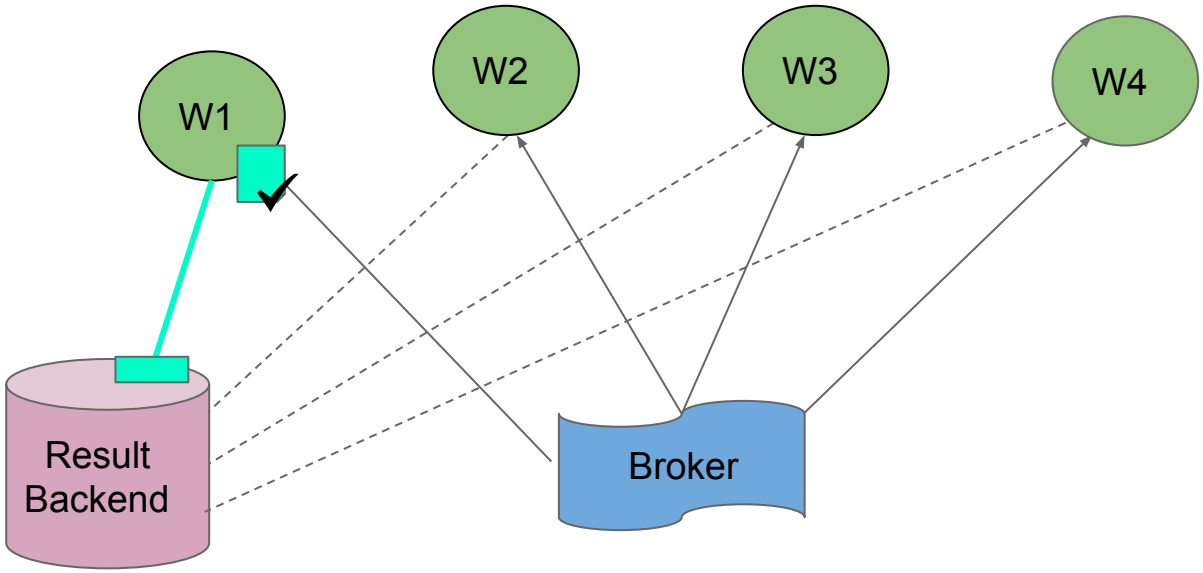
Celery Overview



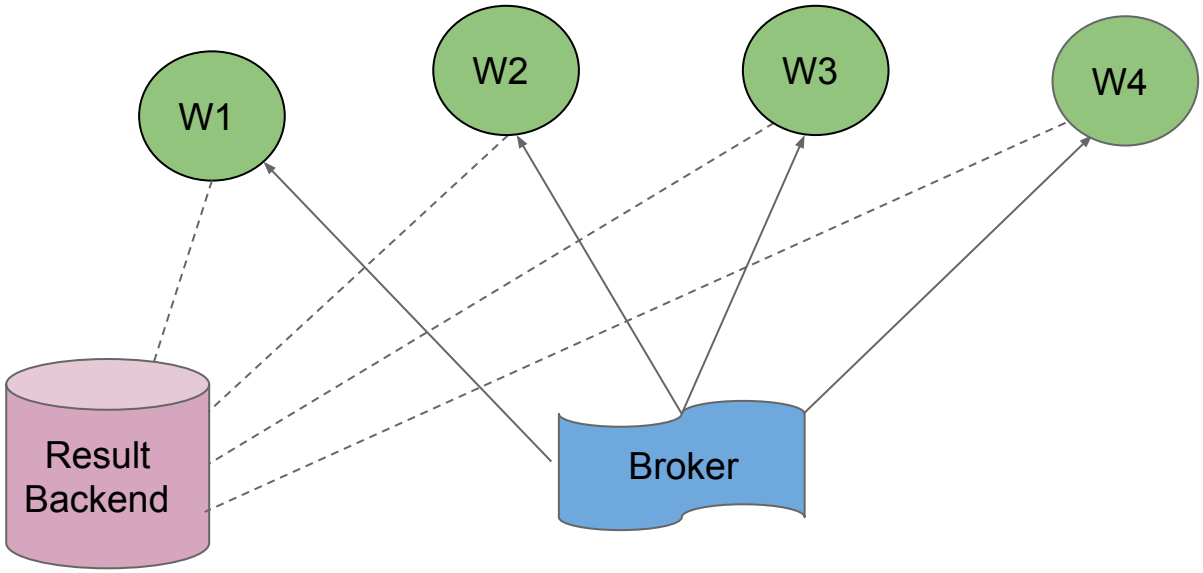
Celery Overview



Celery Overview



Celery Overview

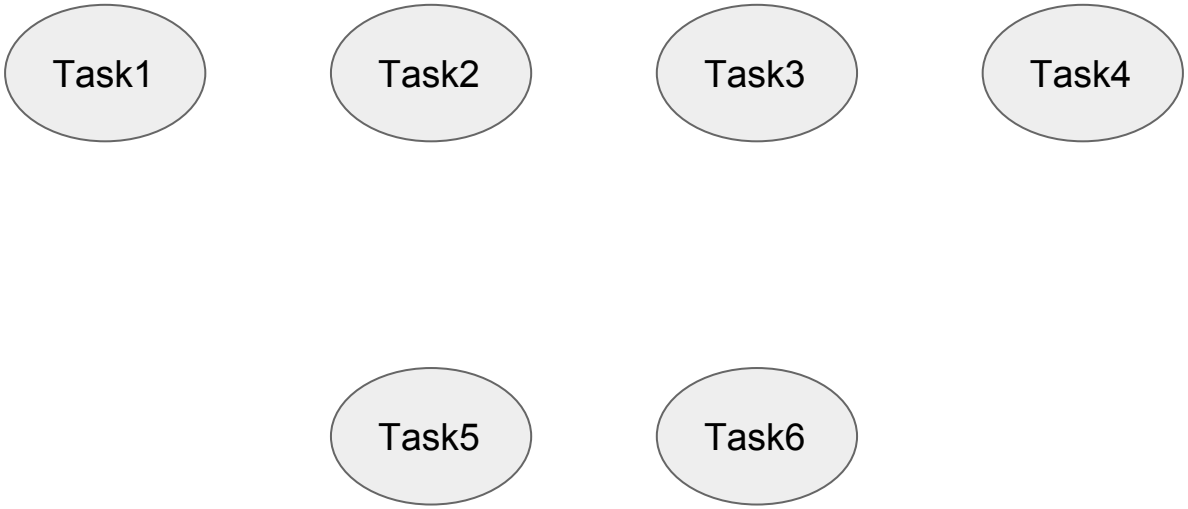


Celery Project

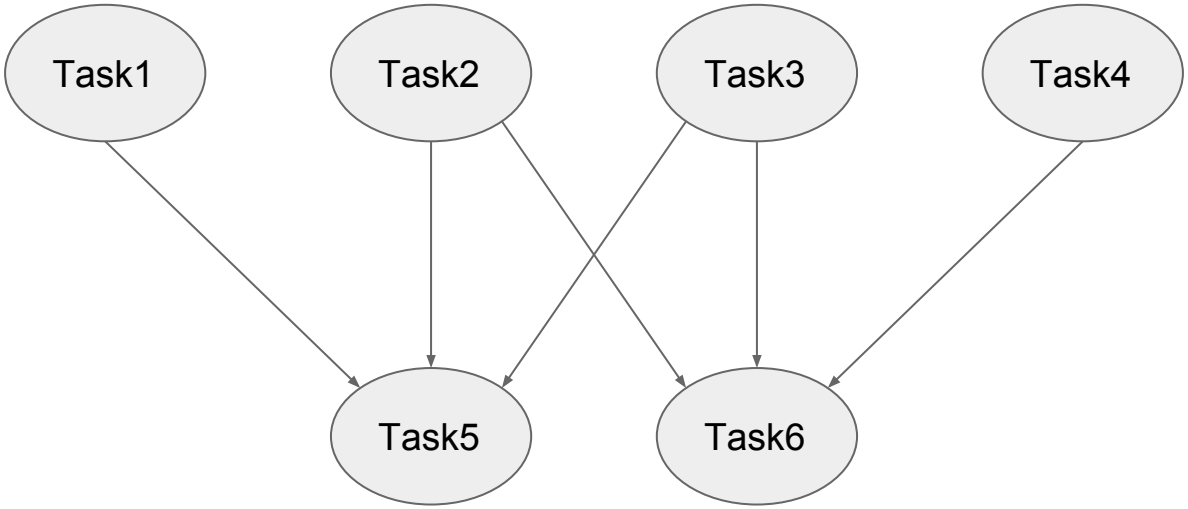
- Celery project
  - http://celeryproject.org/
- Distributed task queue
- Django Celery

Task Flow!

Flow Design



Flow Design

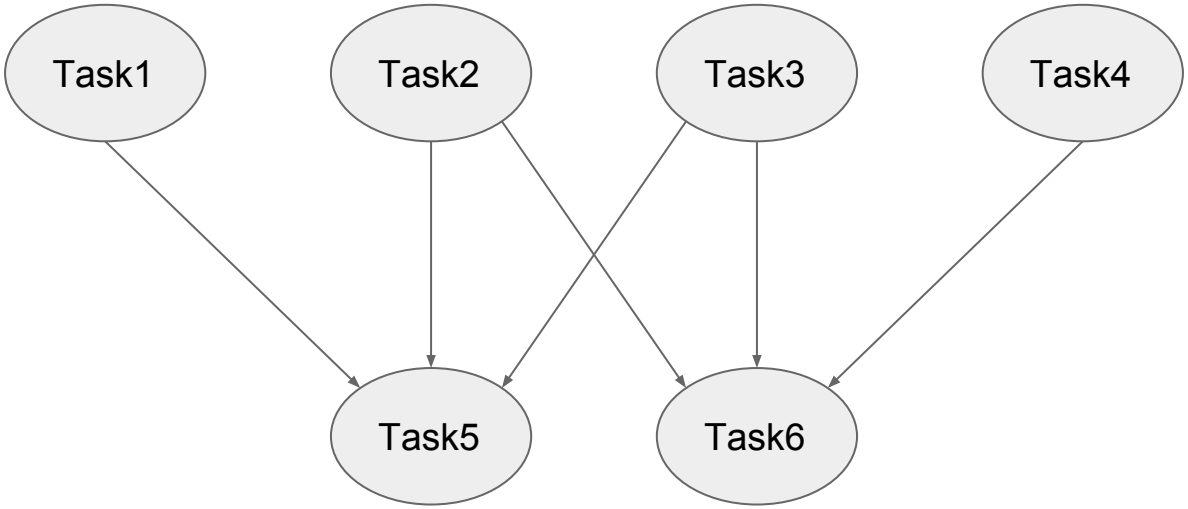


Dependencies Between Tasks - Flows

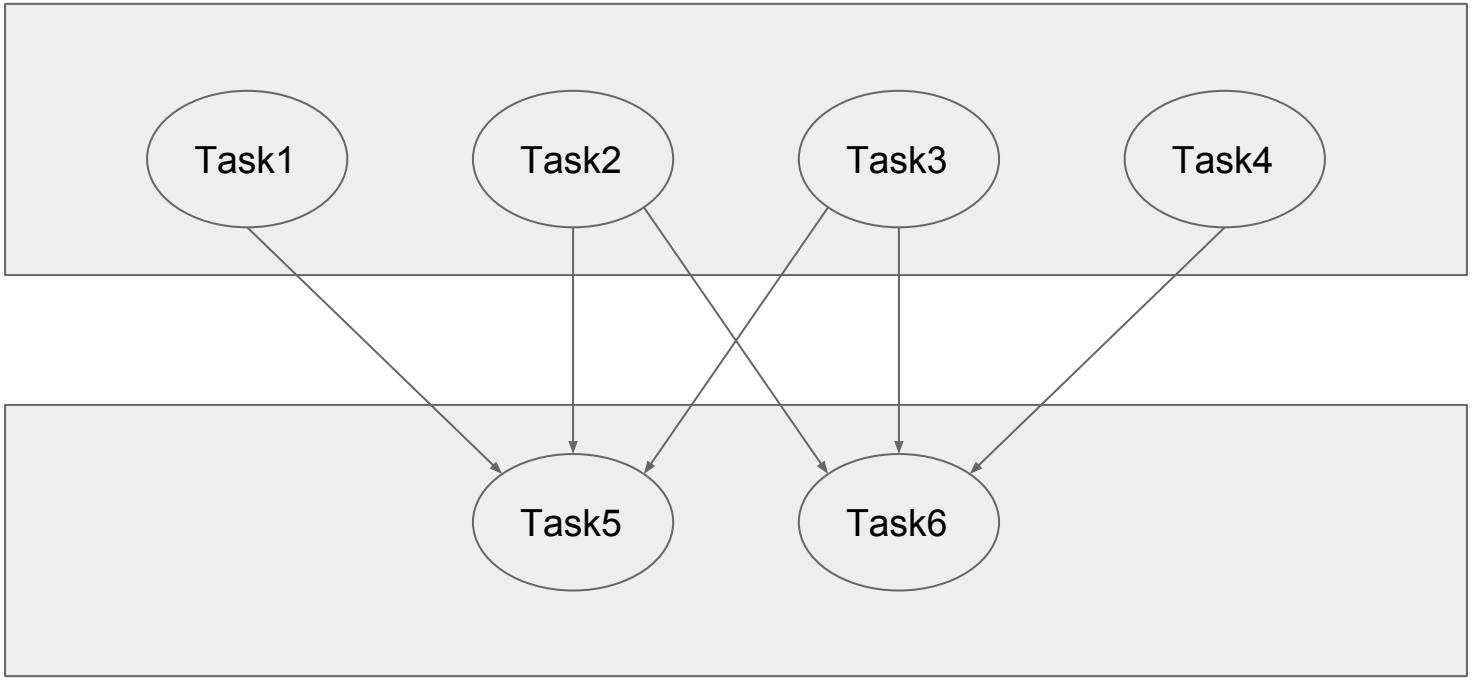
- “Celery primitives”

- Group
- Chain
- Chord
- ...

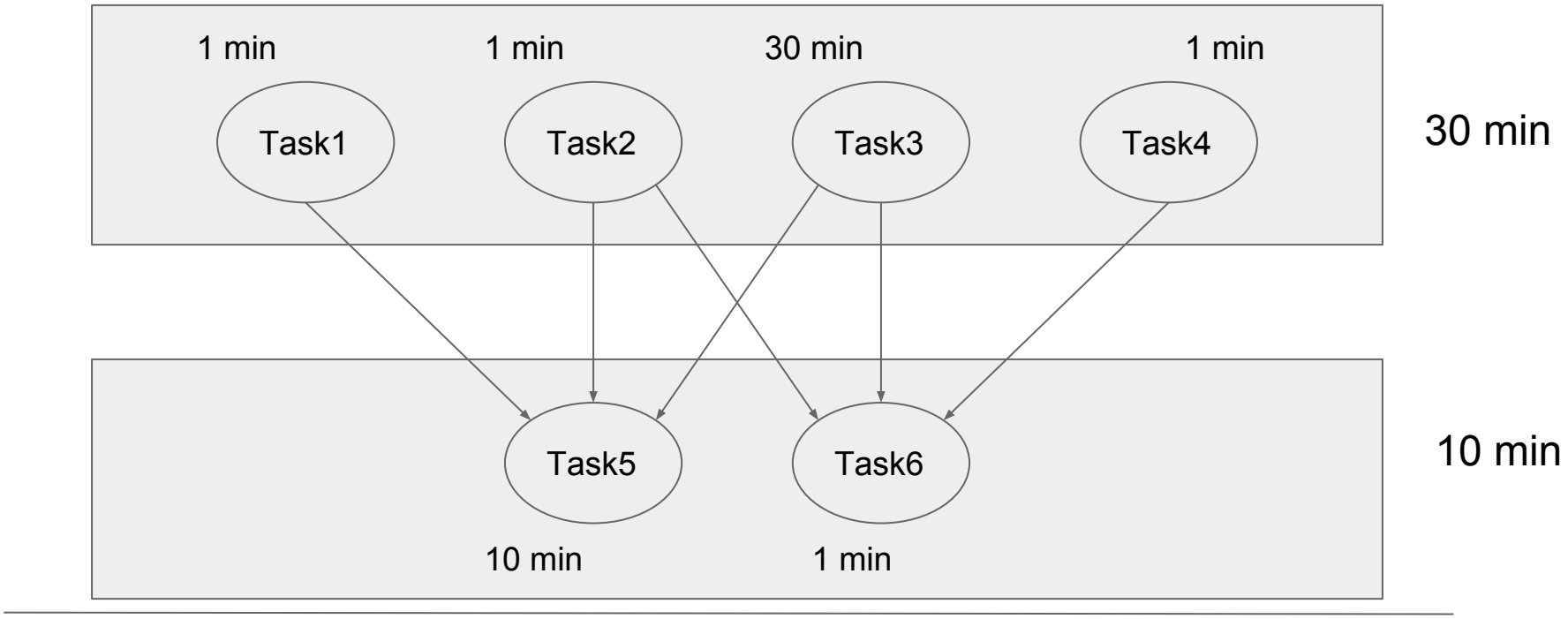
Flow Design



Flow Design

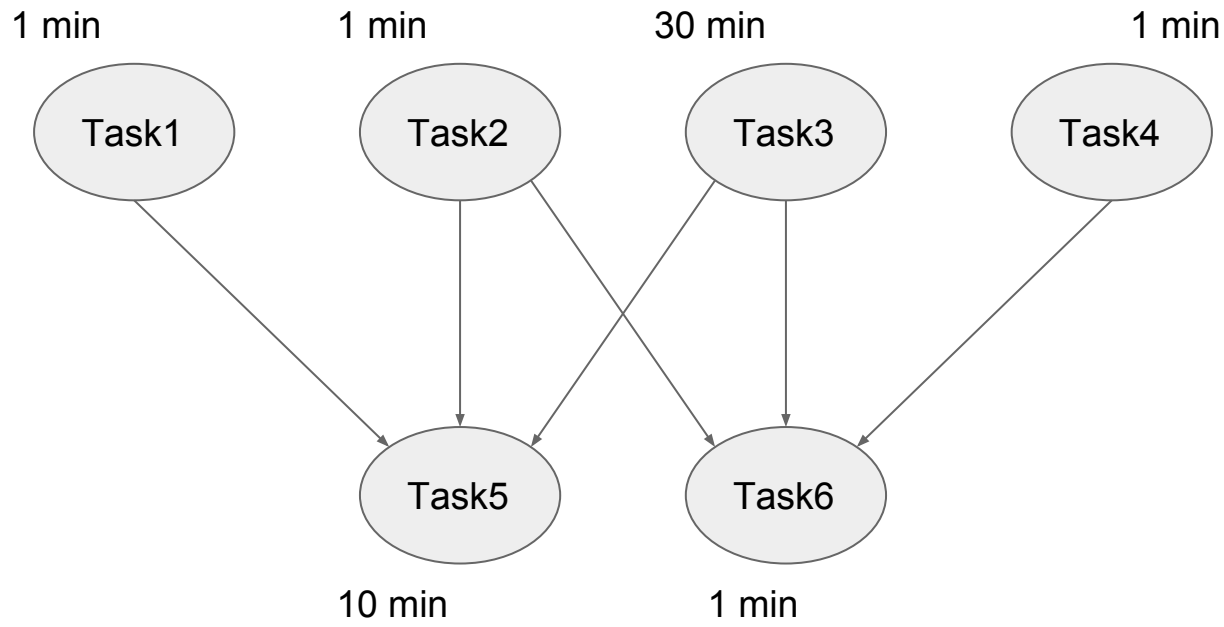


Flow Design



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
31 min

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 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}
```

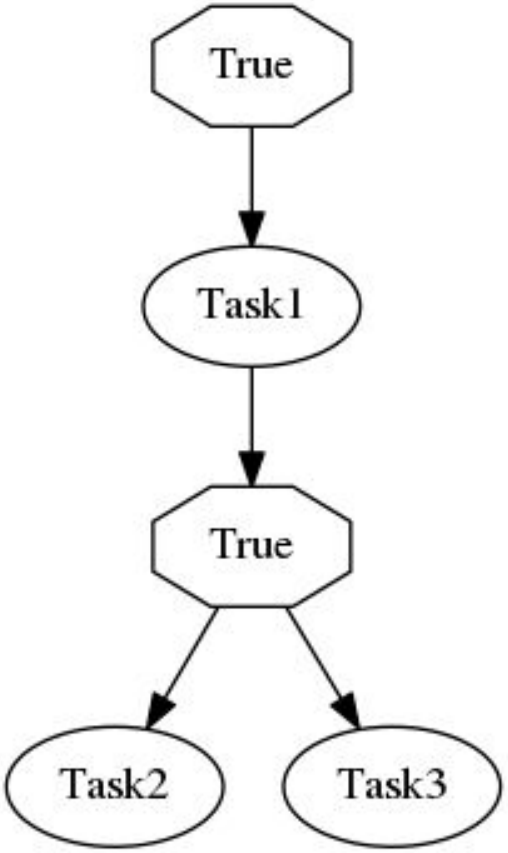
YAML Configuration

```yaml
tasks:
  - name: Task1
    import: myproject.tasks
    queue: Task1_v1

  -  ...

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

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



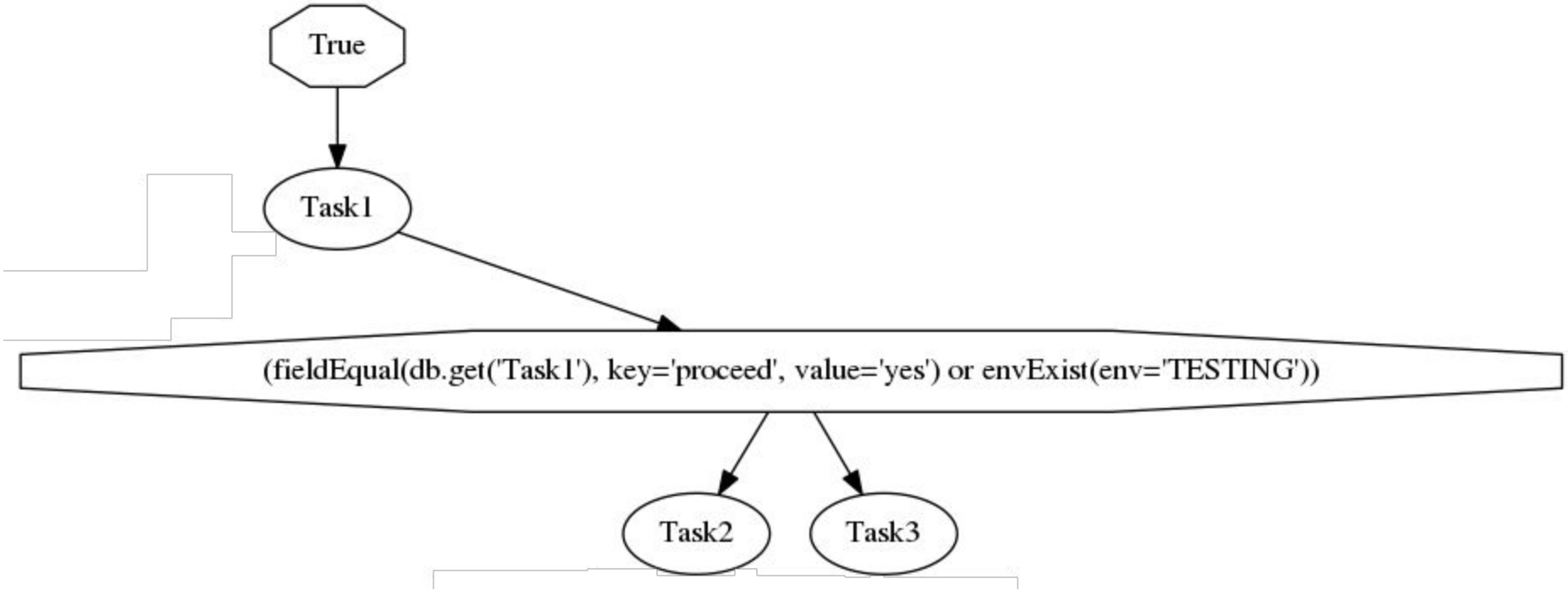
Conditions

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, ...):
        ...
```

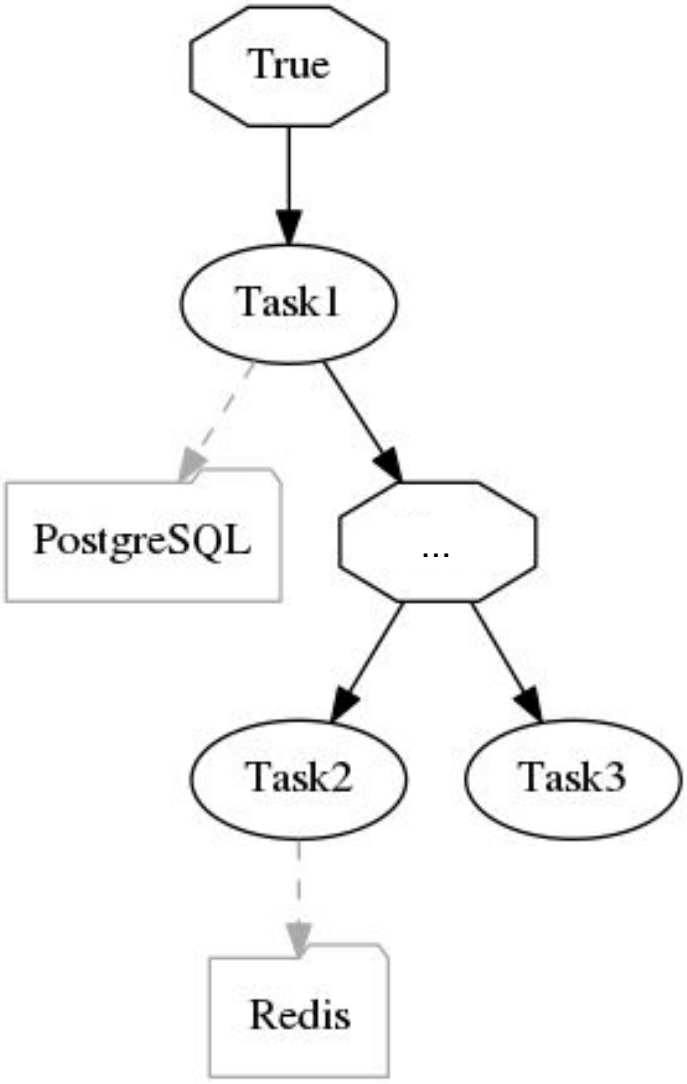
Storages & databases

```
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: ...
```



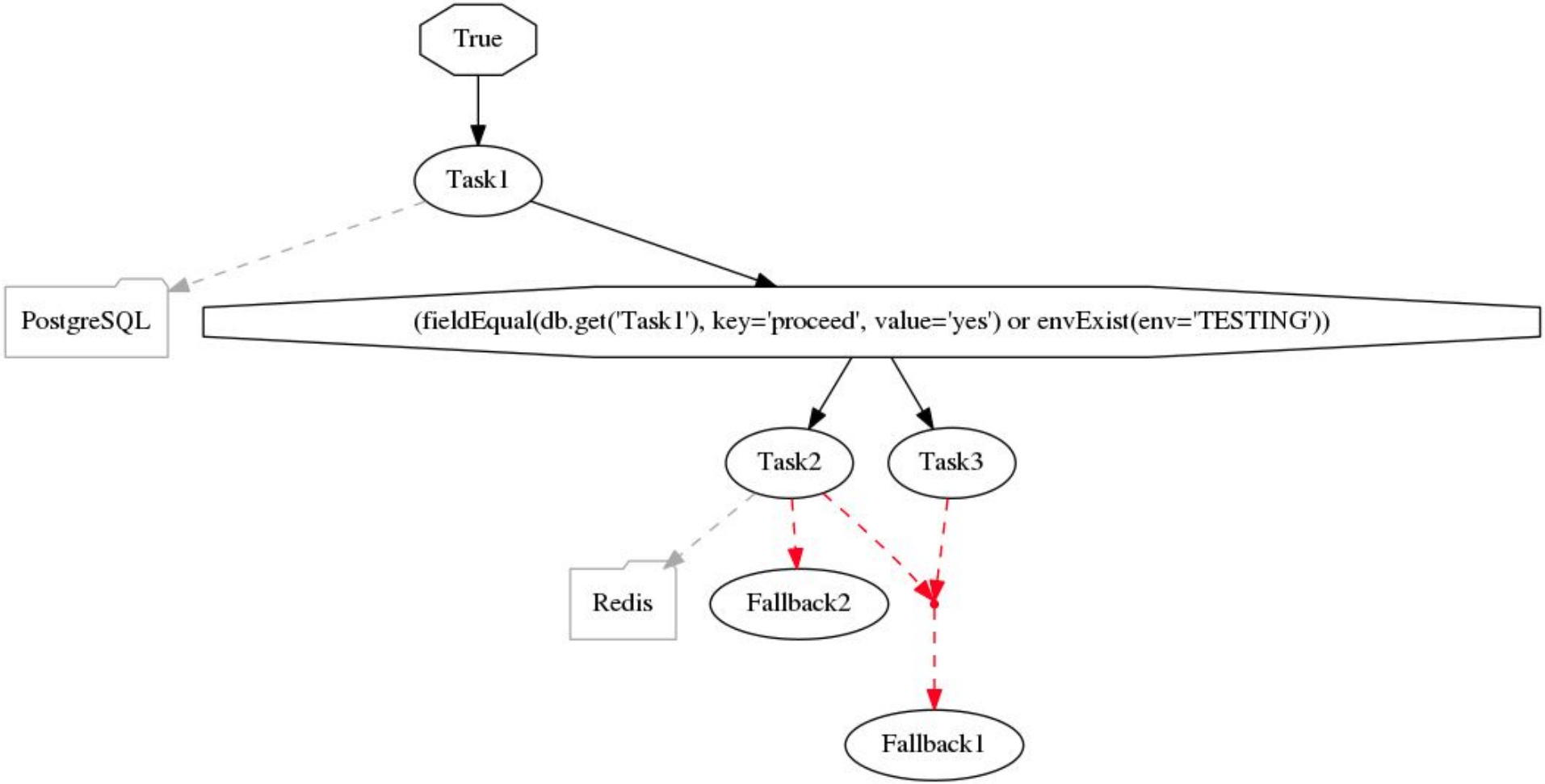
Fallback Tasks & Fallback flows

Fallback tasks & Fallback flows

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

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

Fallback tasks and flows



Subflows

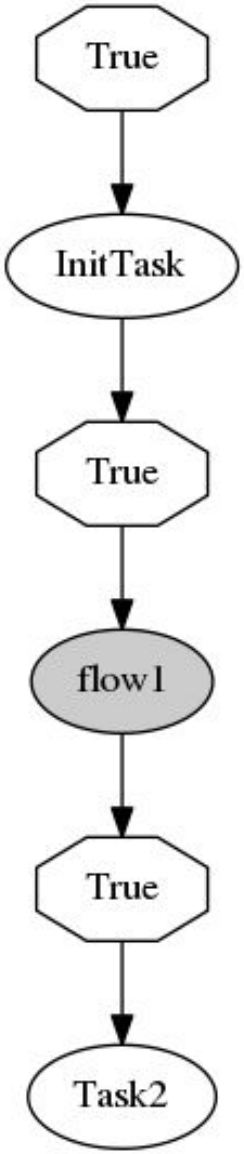
YAML Configuration

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

      - from:
        to: InitTask

      - from: InitTask
        to: flow1

      - from: flow1
        to: Task2
```



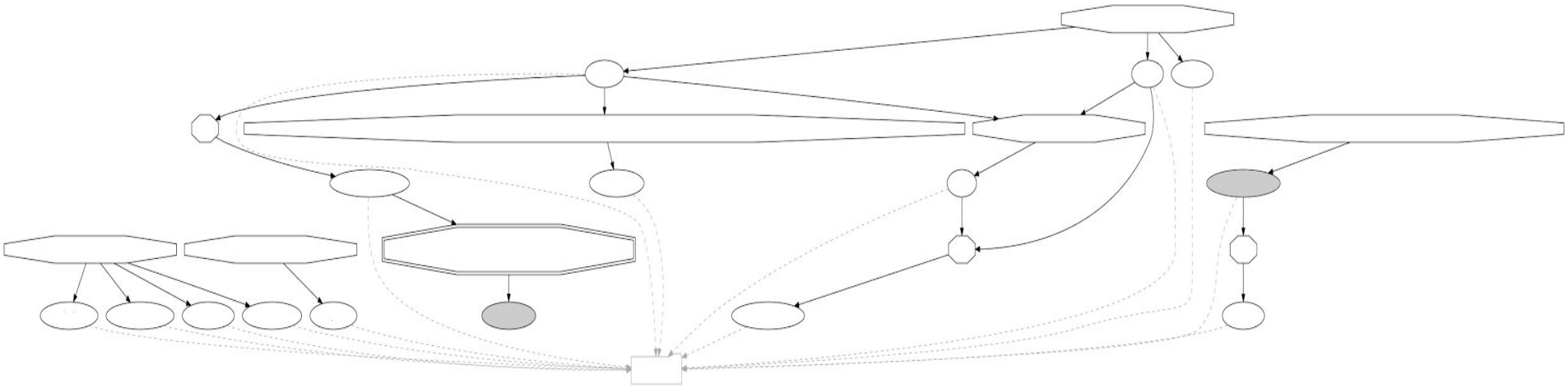
How does SELINON work?

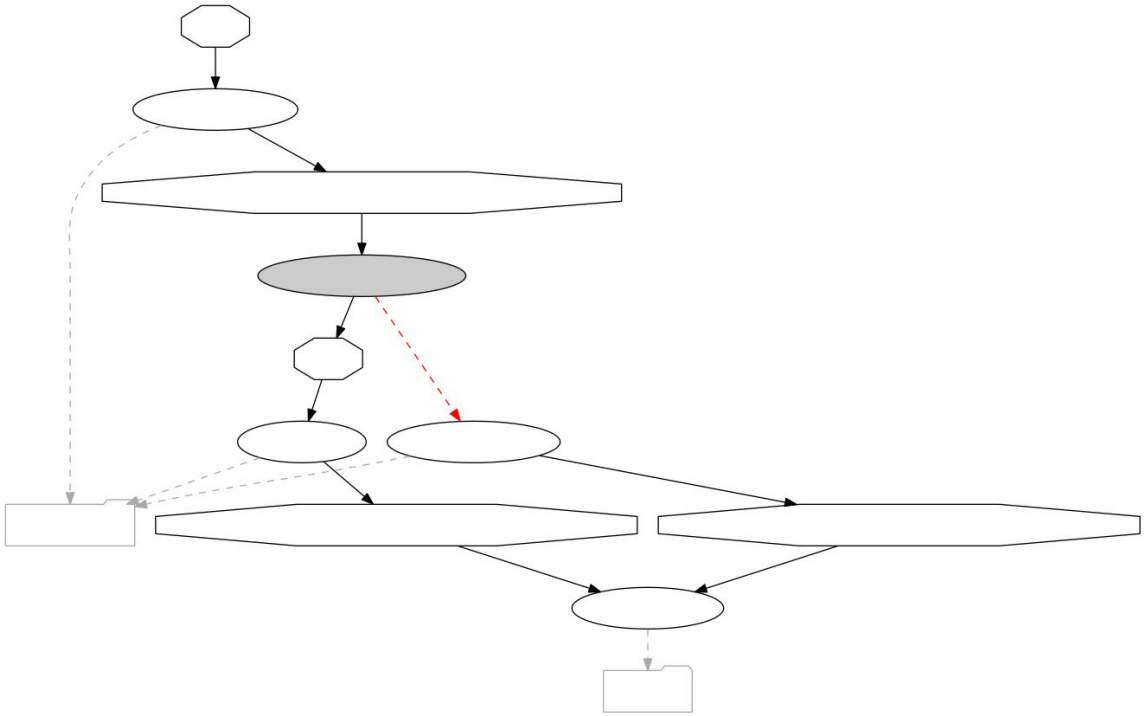
Selinon

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

OTHER FEATURES

- Caches
- Task and flow throttling
- Task and flow prioritization
- Optimization of Dispatcher scheduling
- Tracepoints
- Graphs are not DAG
- Cluster
- ...





Questions?

https://github.com/selinon