



SELINON

DISTRIBUTED COMPUTING WITH PYTHON

Fridolín Pokorný
<fridolin@redhat.com>

\$ whoami

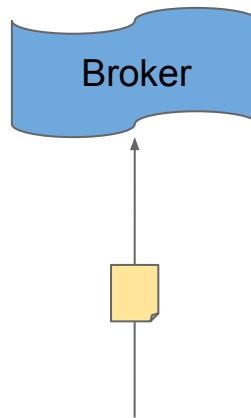
- Fridolín Pokorný
- Brno, Czech republic
- reverse engineering
- now Red Hat AICoE
 - Selinon
 - Thoth

DISTRIBUTED COMPUTING!

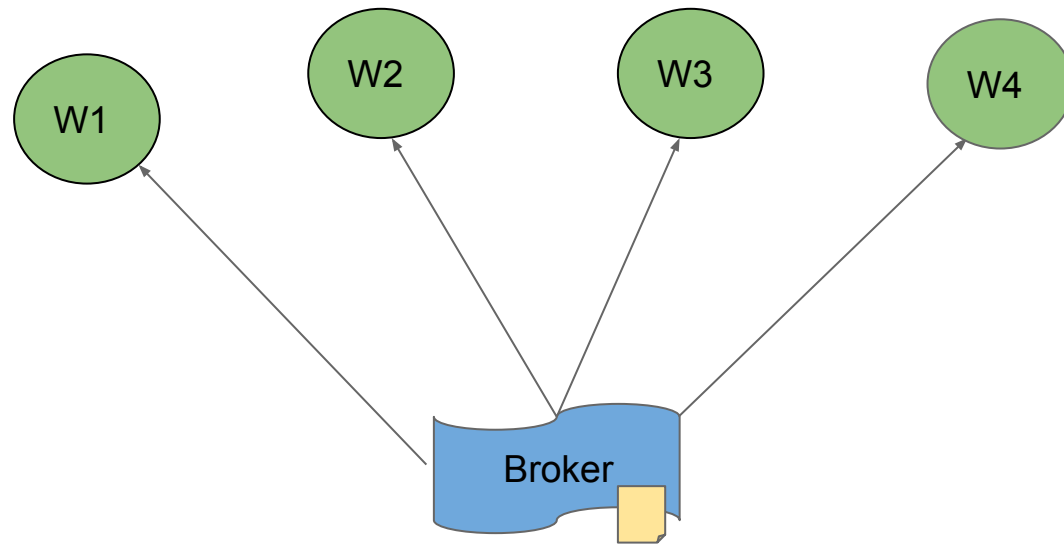
CELERY OVERVIEW

 message

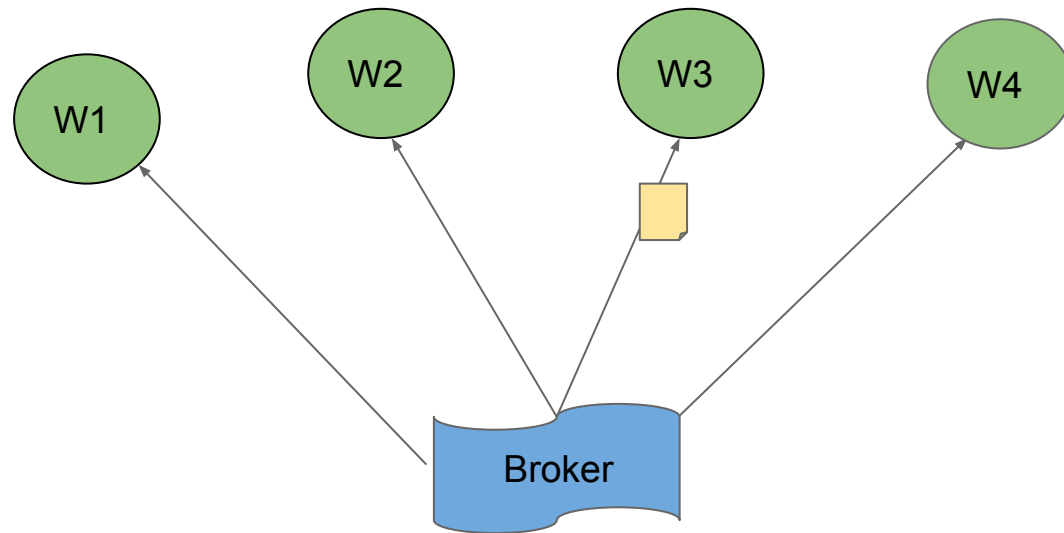
CELERY OVERVIEW



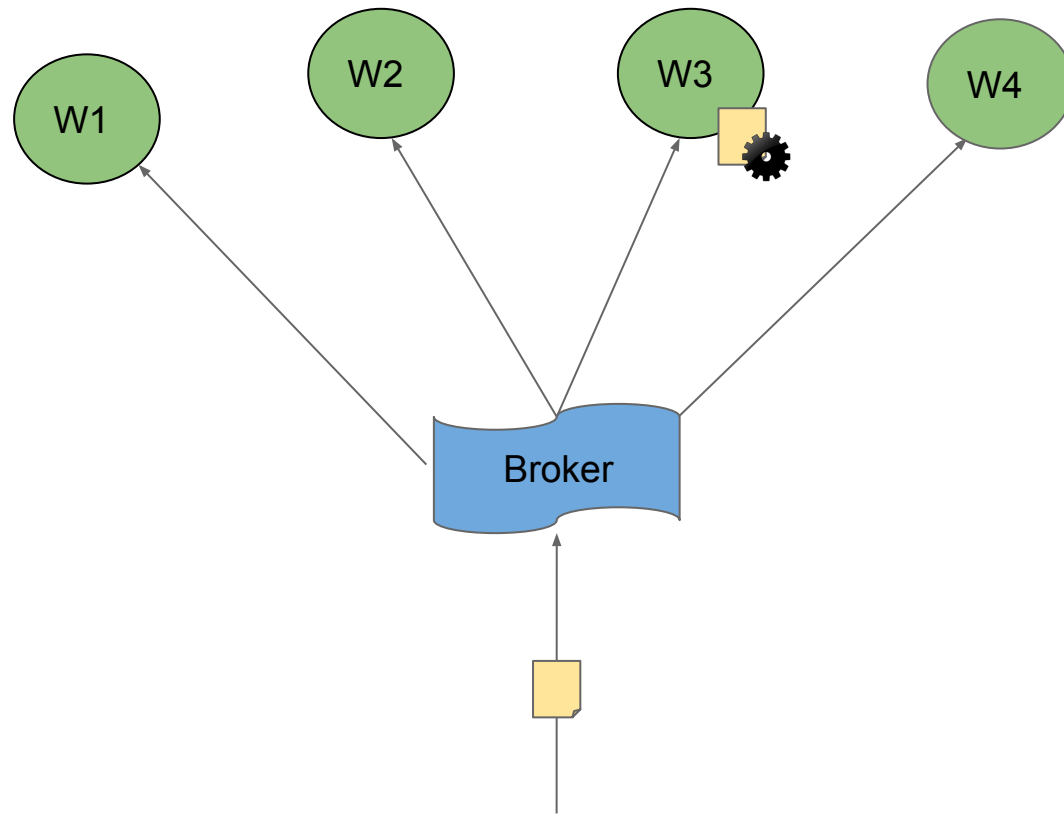
CELERY OVERVIEW



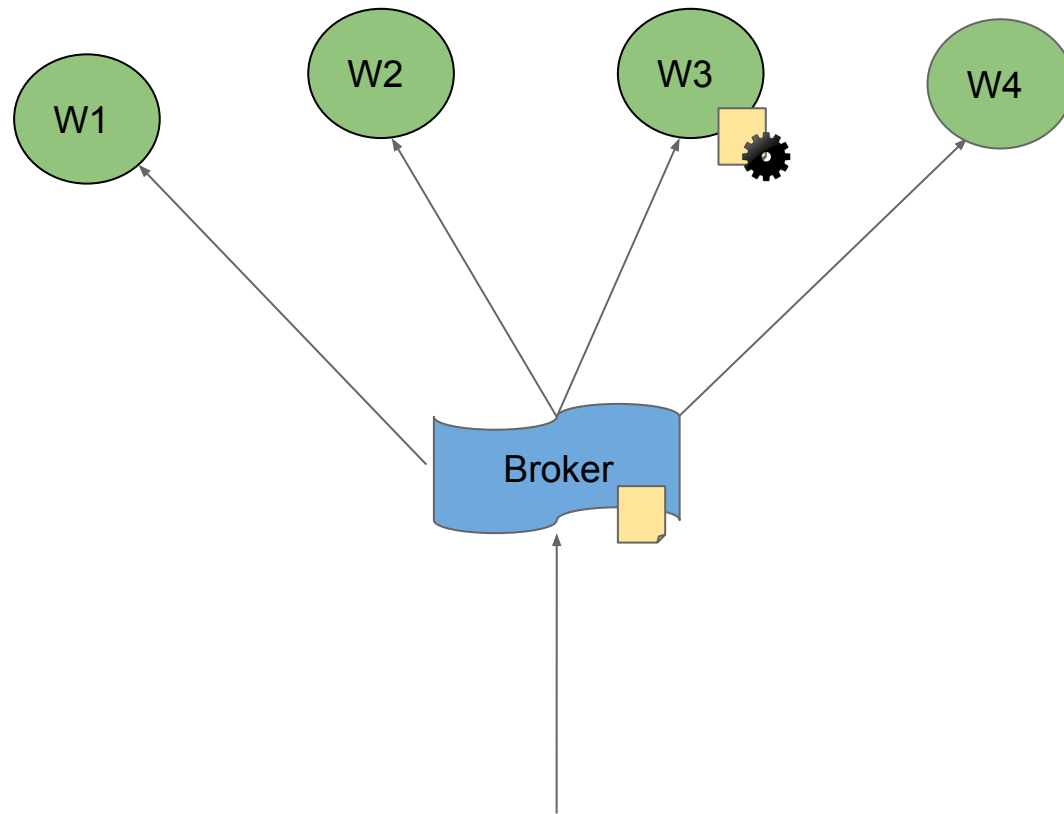
CELERY OVERVIEW



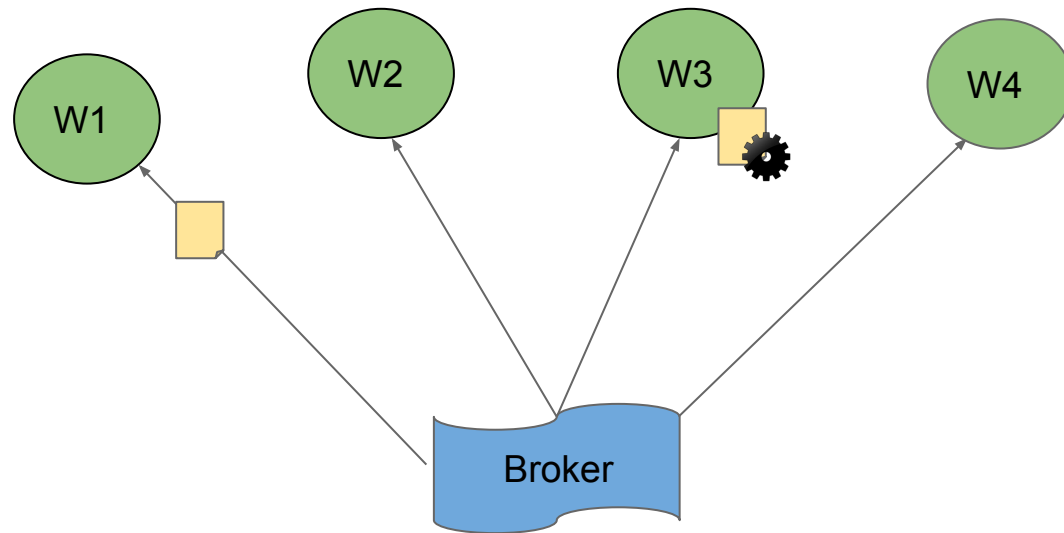
CELERY OVERVIEW



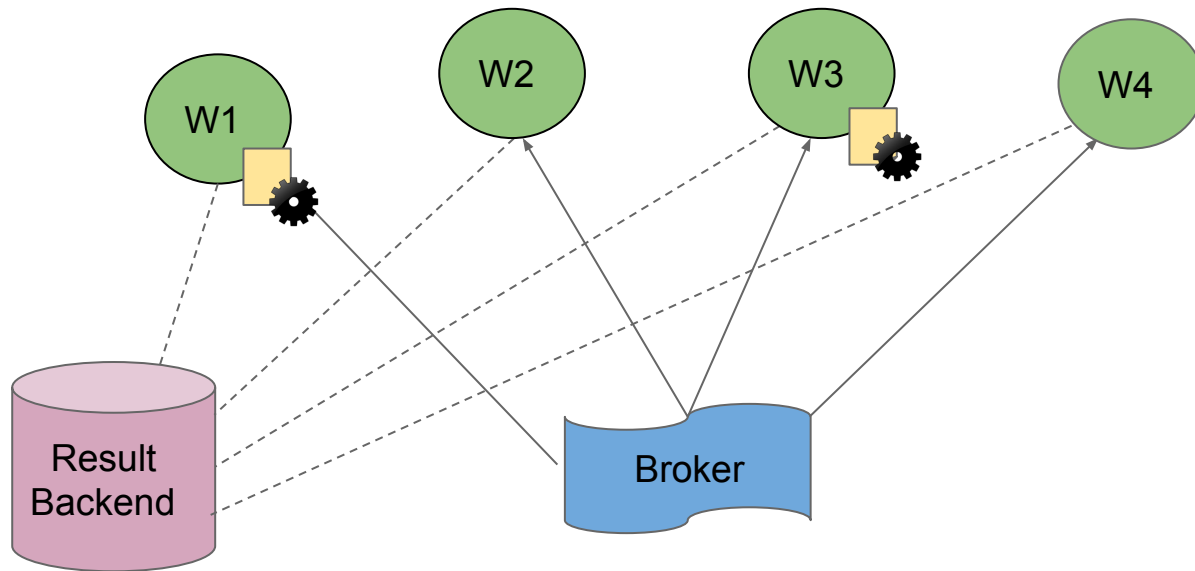
CELERY OVERVIEW



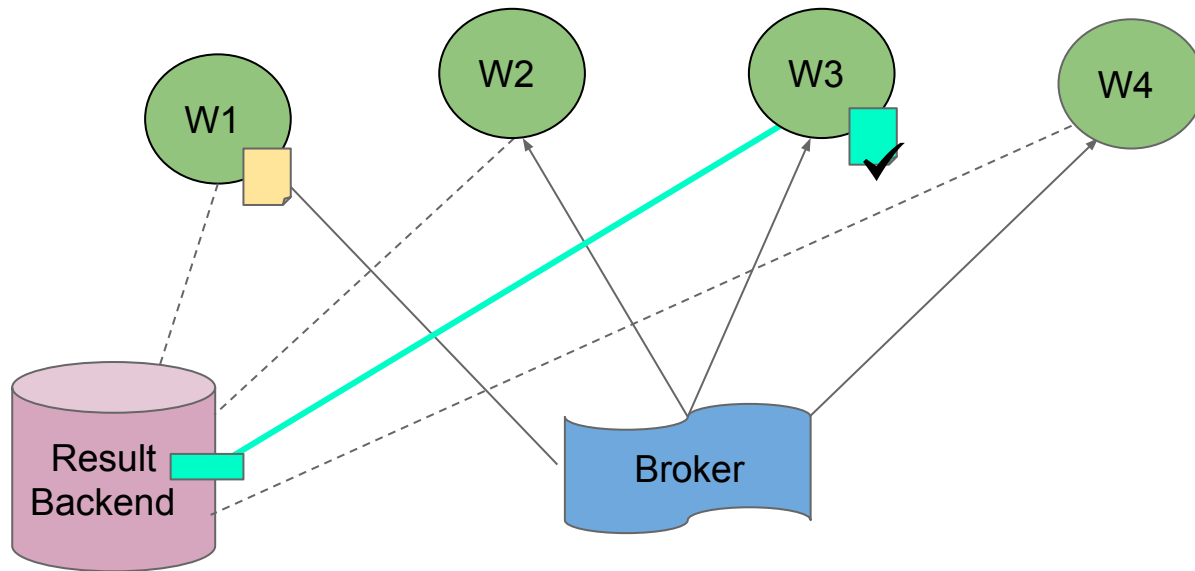
CELERY OVERVIEW



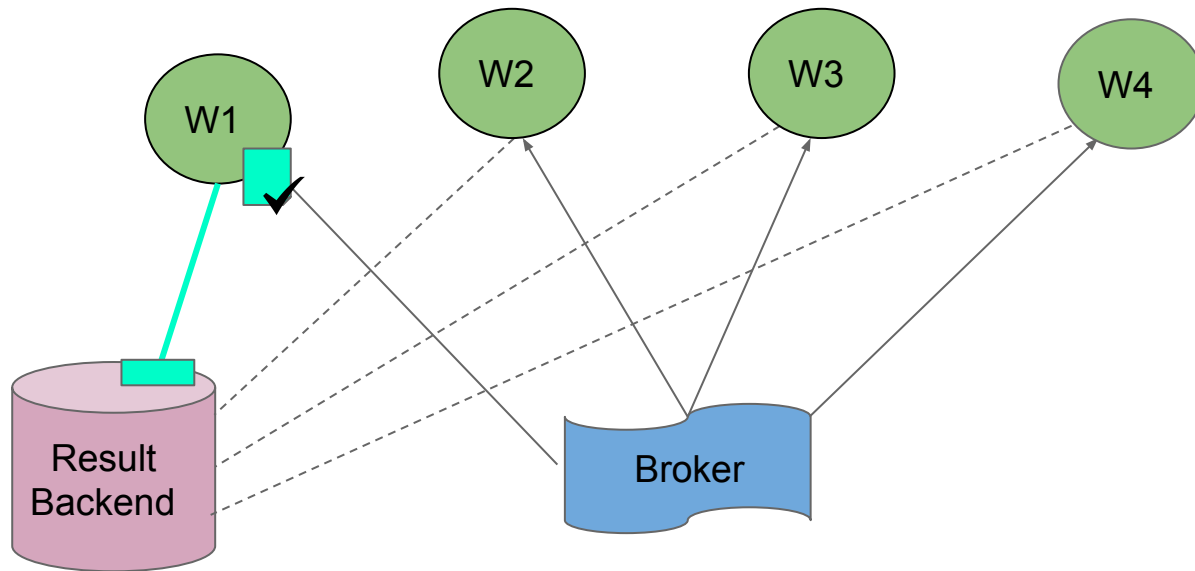
CELERY OVERVIEW



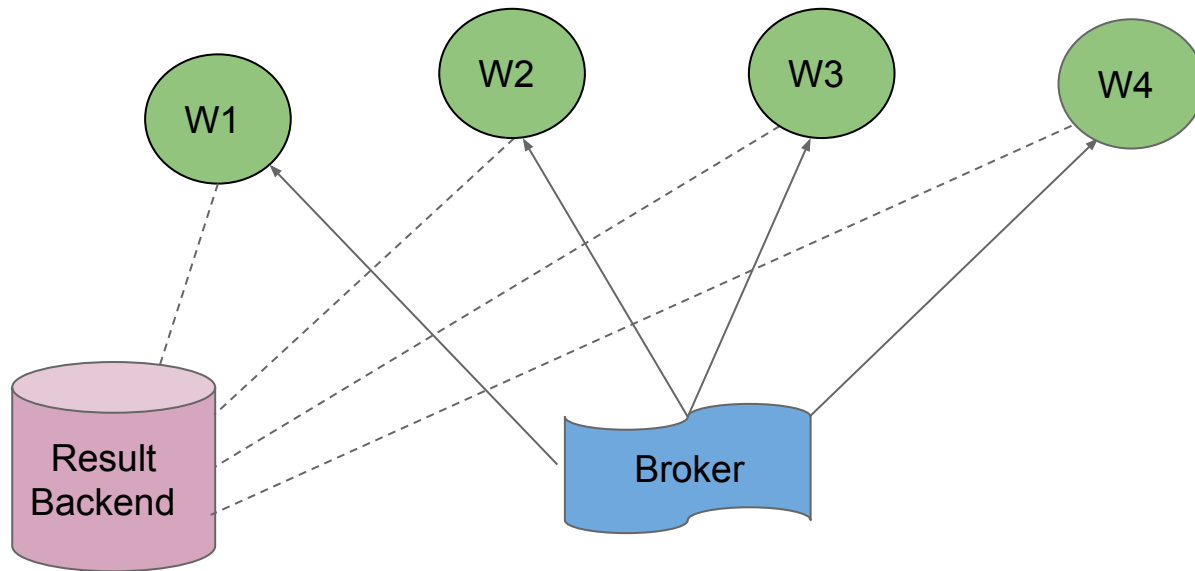
CELERY OVERVIEW



CELERY OVERVIEW



CELERY OVERVIEW



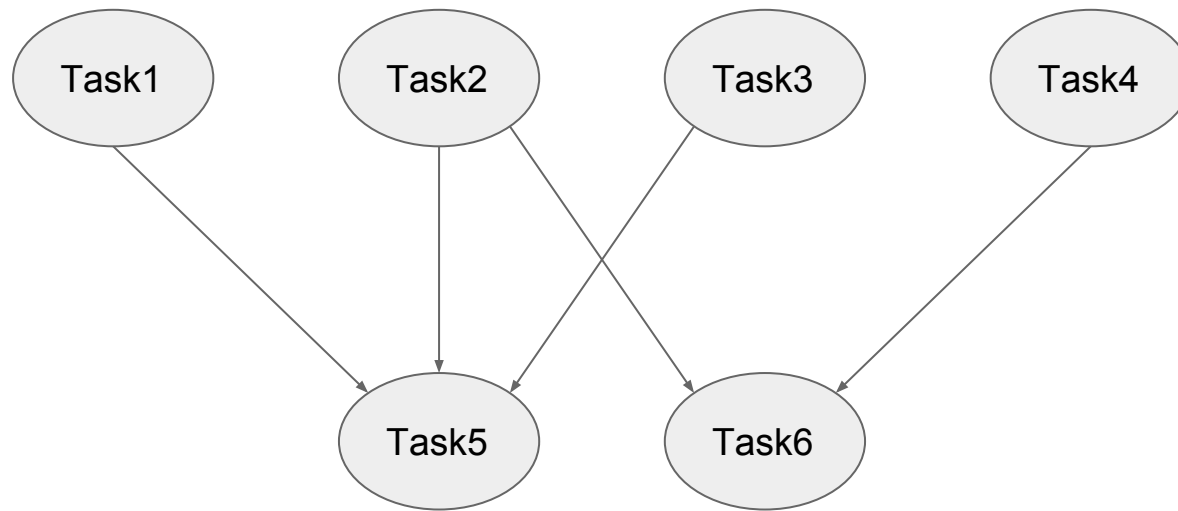
CELERY PROJECT



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

TASK FLOW!

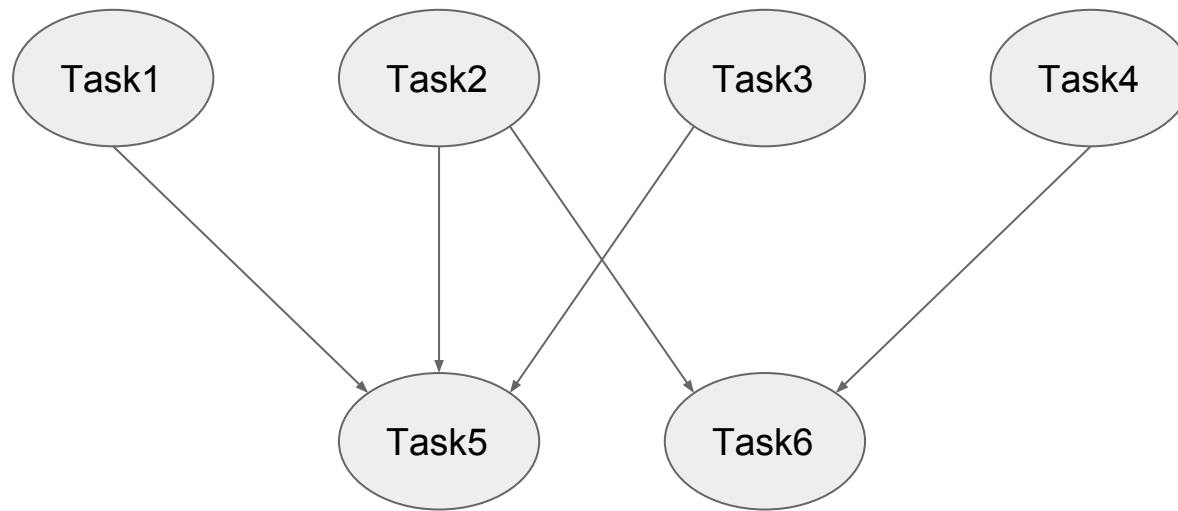
FLOW DESIGN



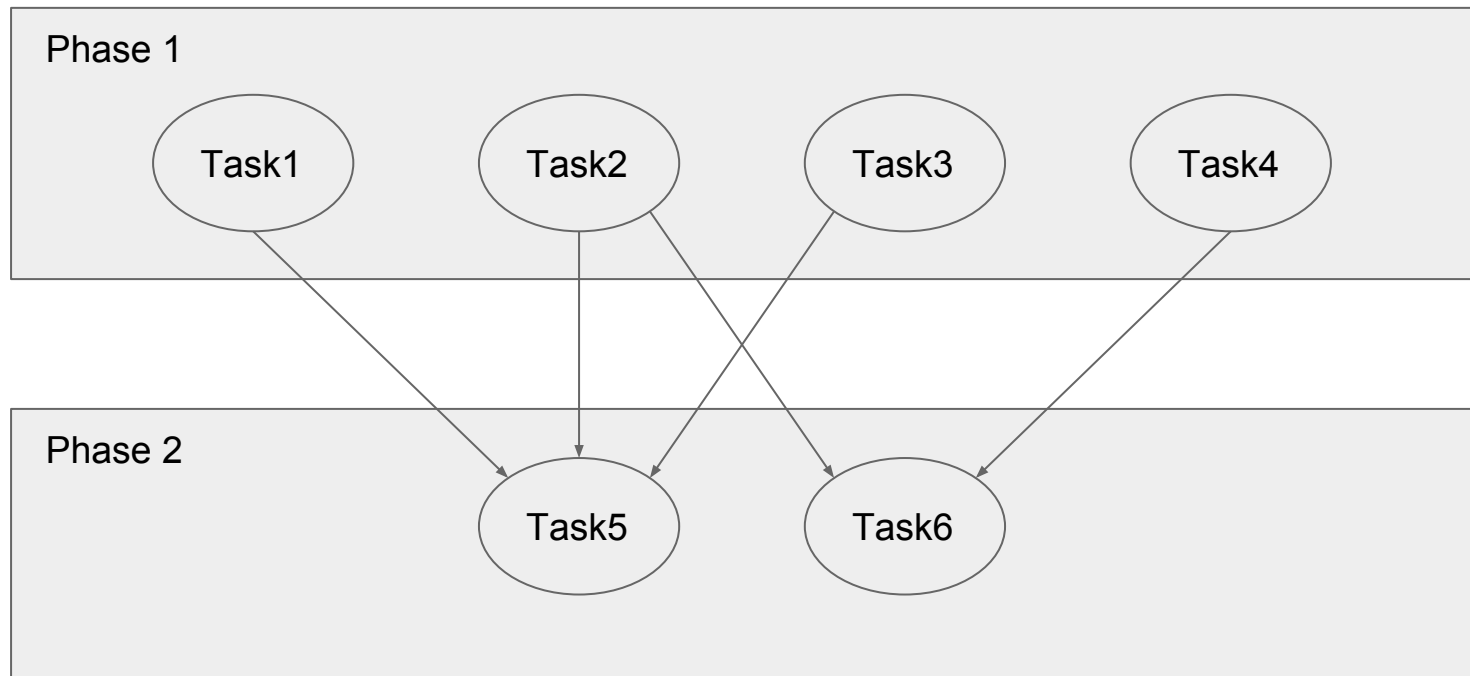
DEPENDENCIES BETWEEN TASKS - FLOWS

- “Celery primitives”
 - Group
 - Chain
 - Chord
 - Map
 - Starmap
 - Chunks

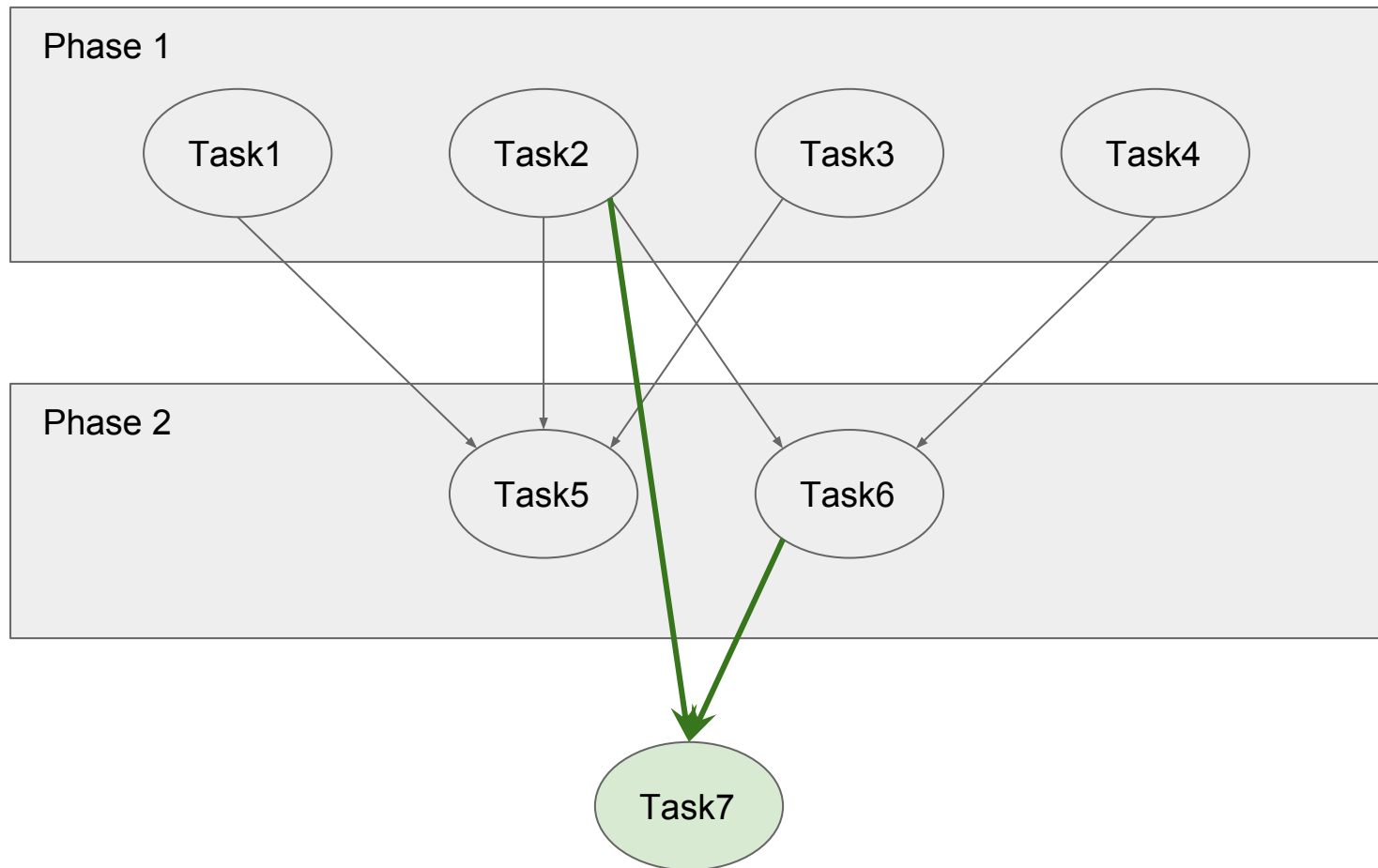
FLOW DESIGN



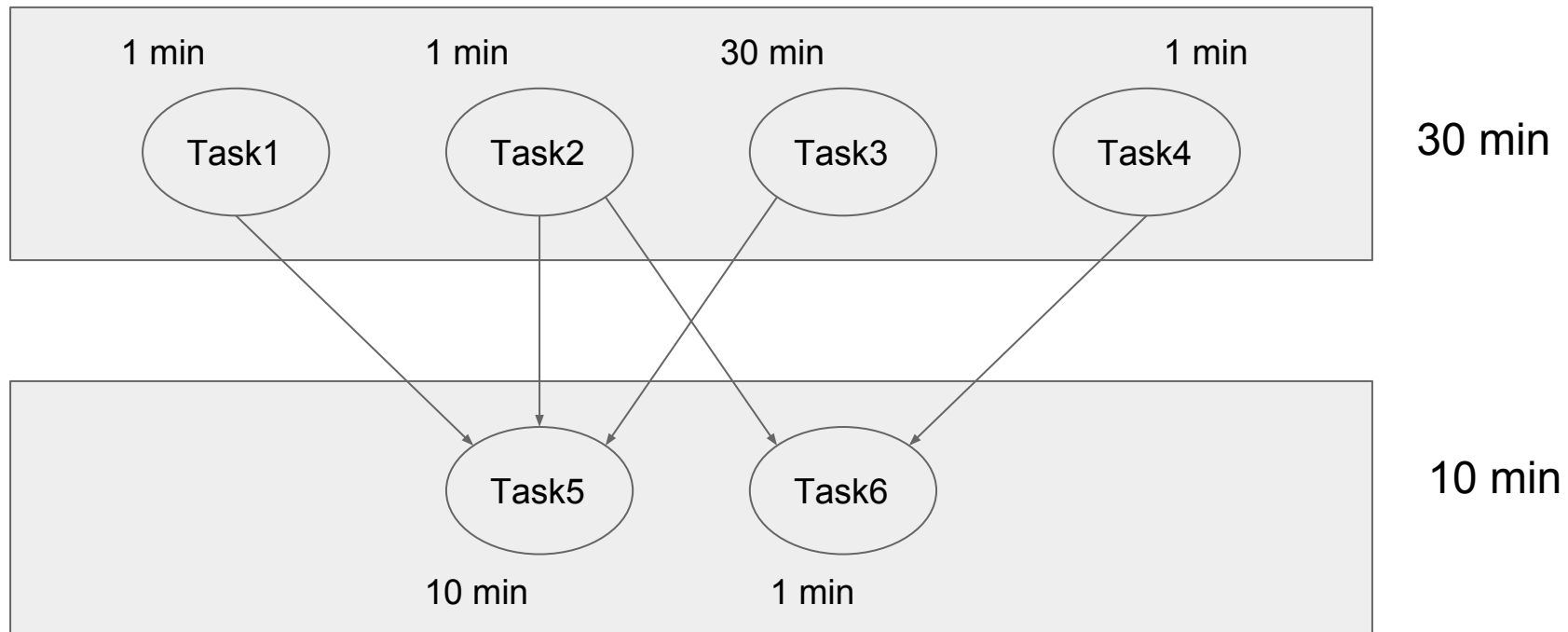
FLOW DESIGN



FLOW DESIGN

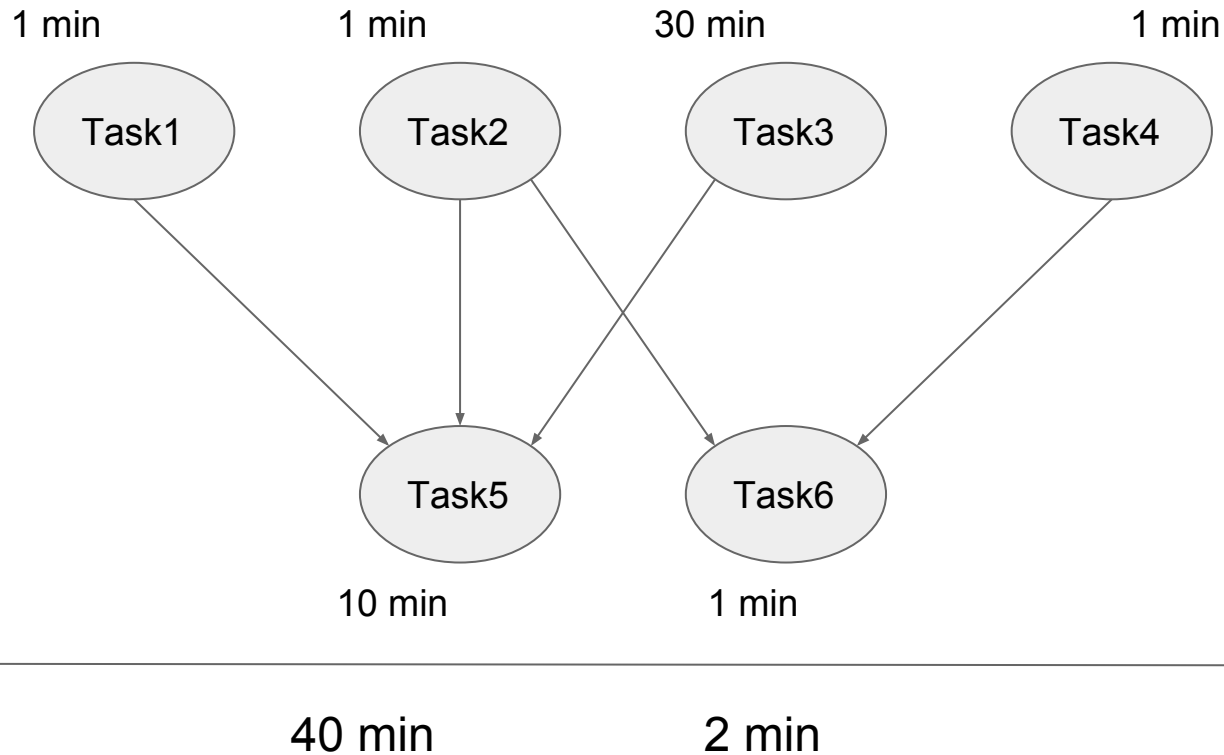


FLOW DESIGN

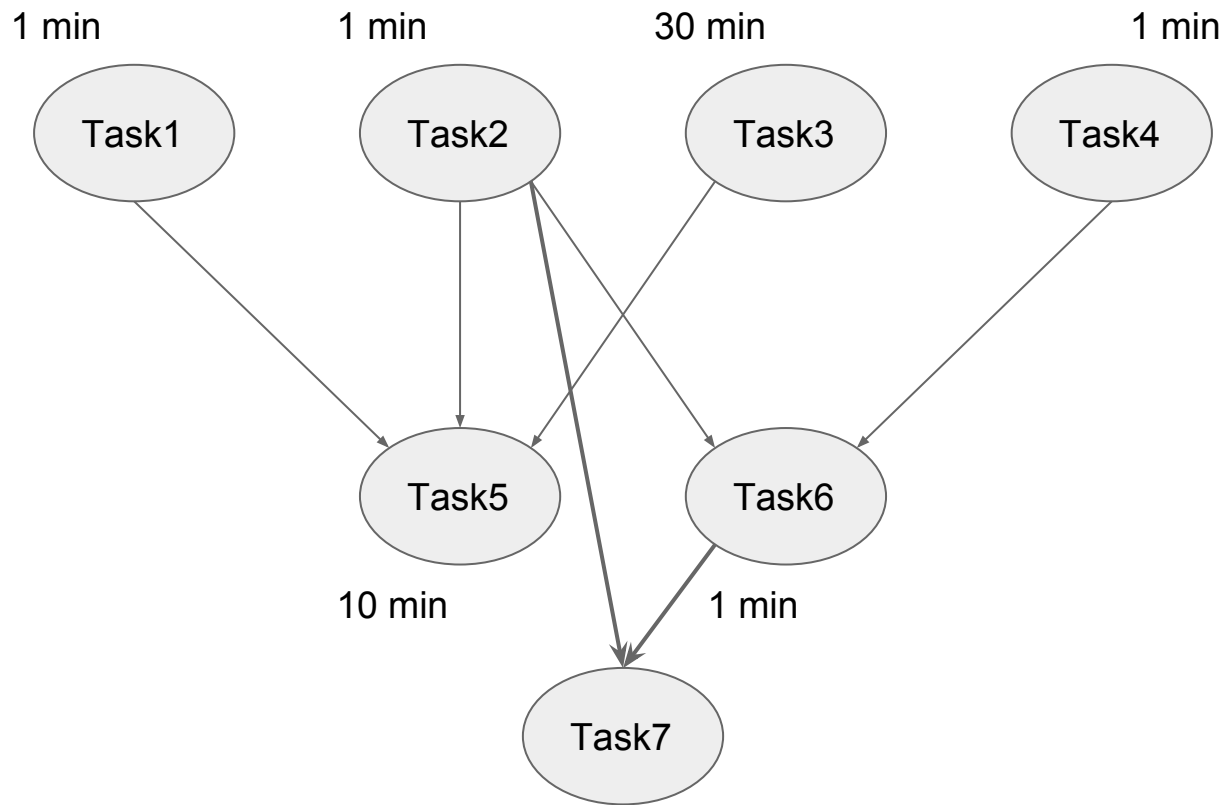


Total: 40 min

FLOW DESIGN



FLOW DESIGN



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

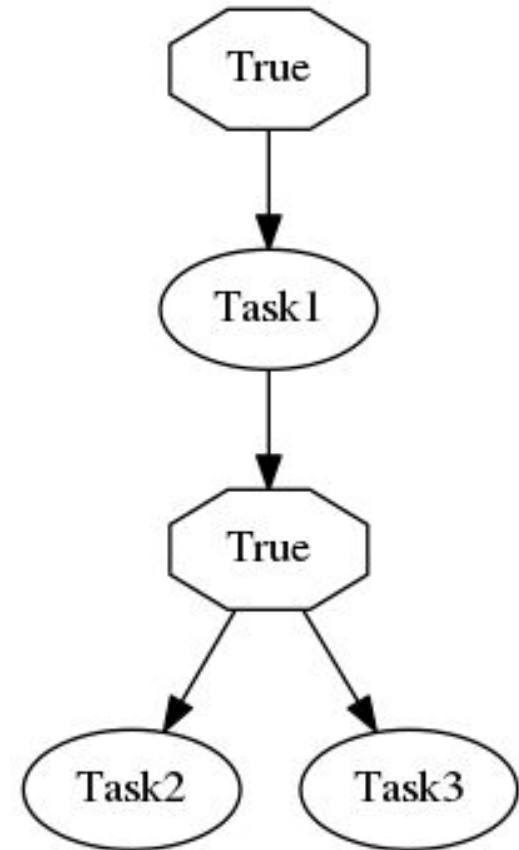
class Task1(SelinonTask):

    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
```



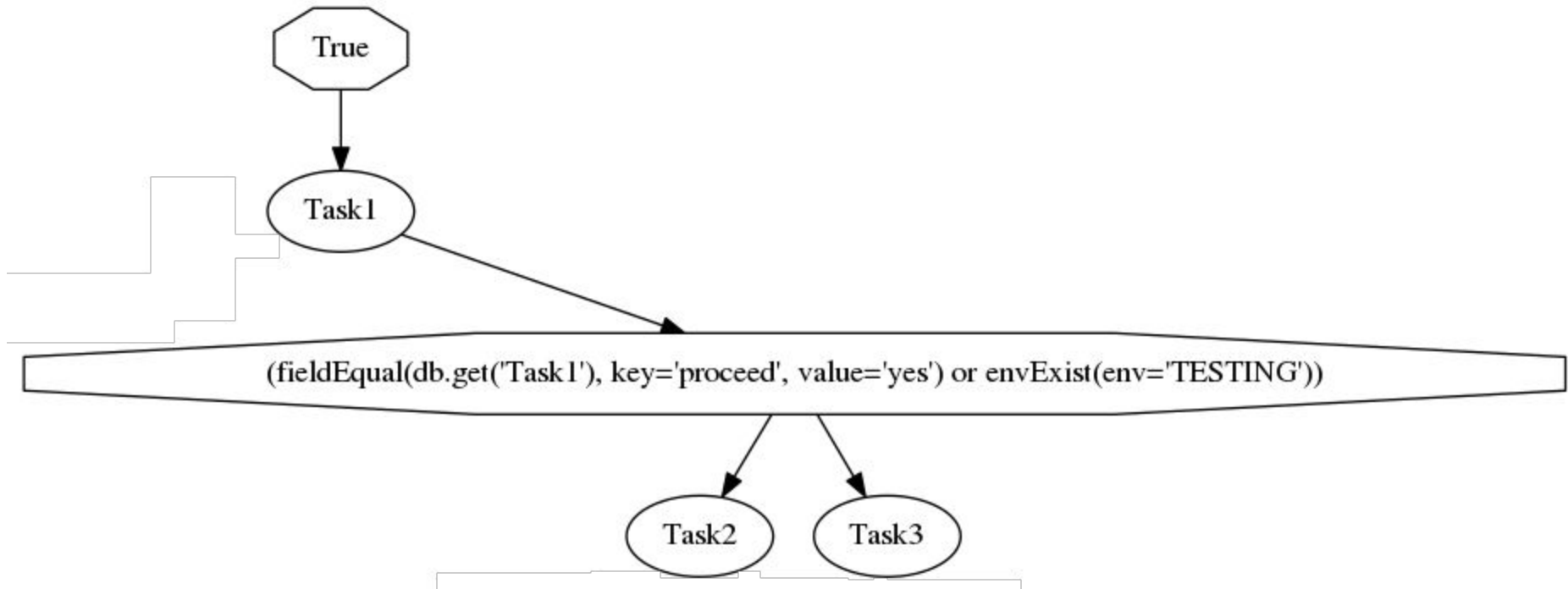
CONDITIONS

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



STORAGES & DATABASES

SELINON DATA STORAGE

```
from selinon import DataStorage

class Redis(DataStorage):

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

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

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

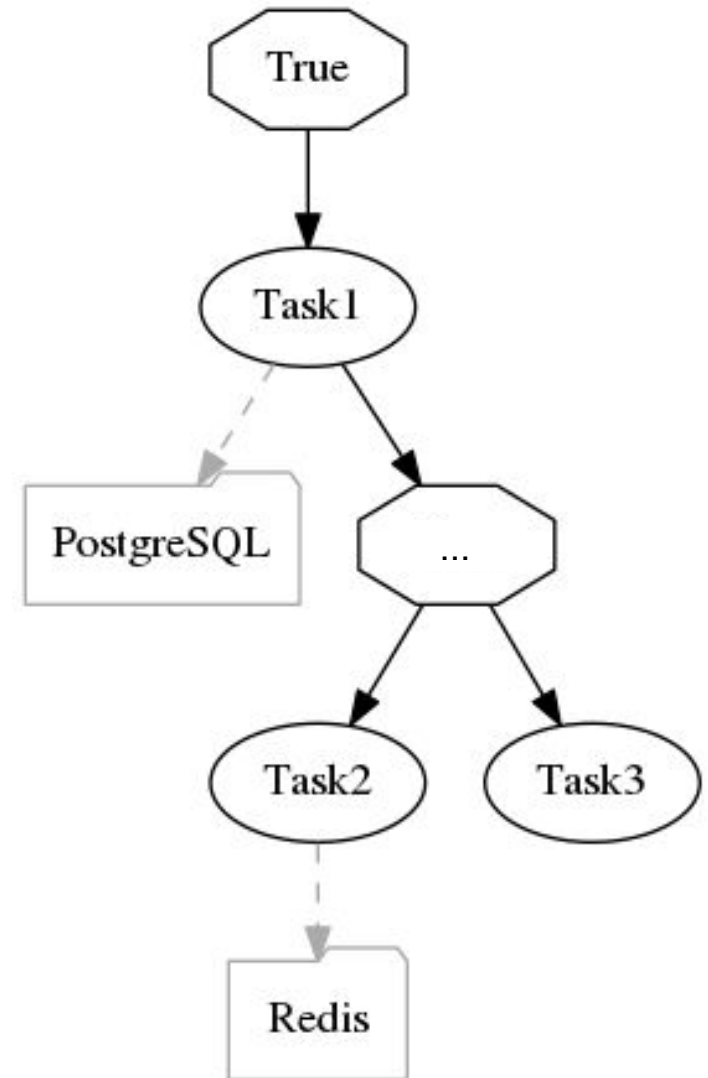
STORAGES & DATABASES

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: ...



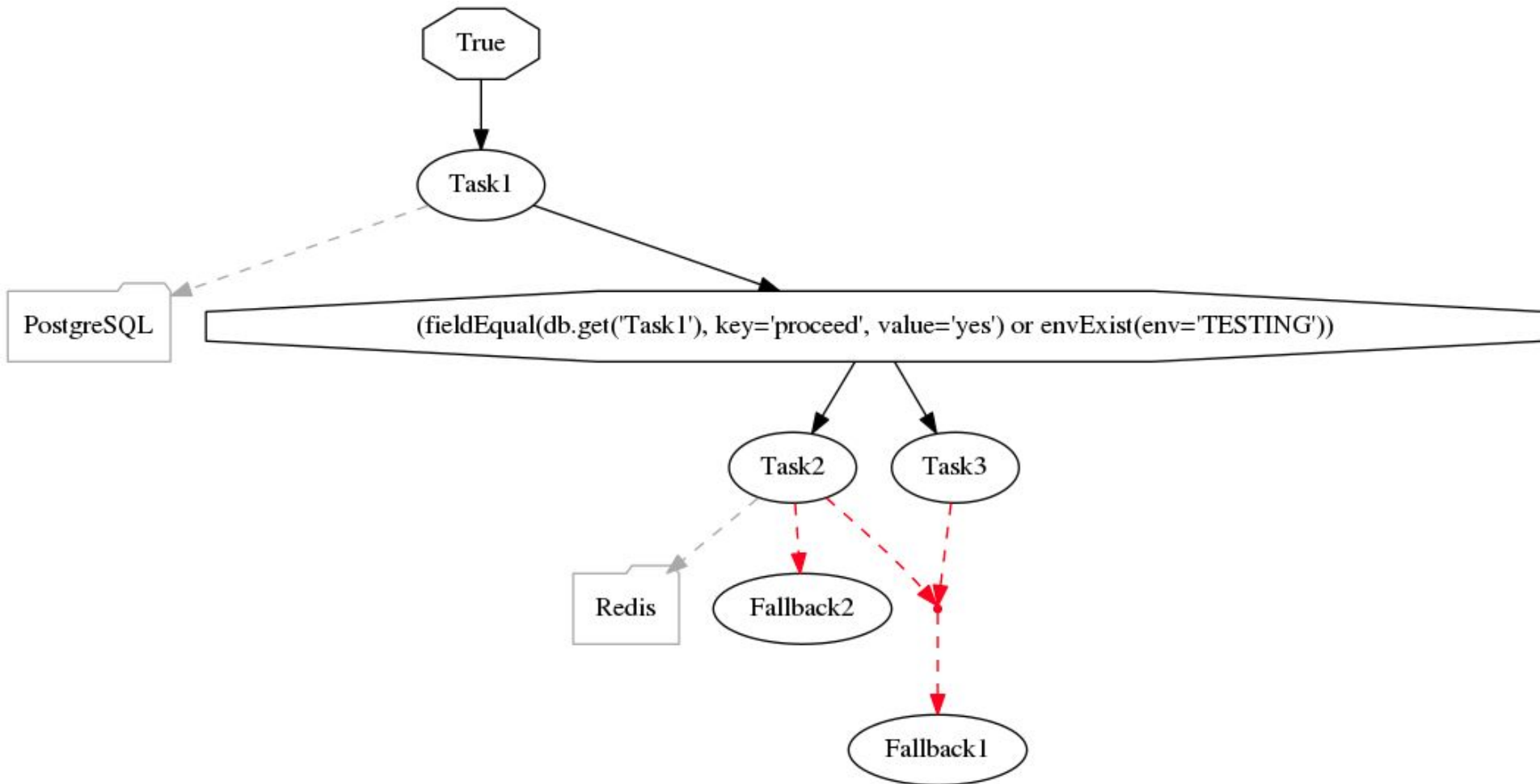
FALLBACK TASKS & FALLBACK FLOWS

FALLBACK TASKS & FALLBACK FLOWS

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

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

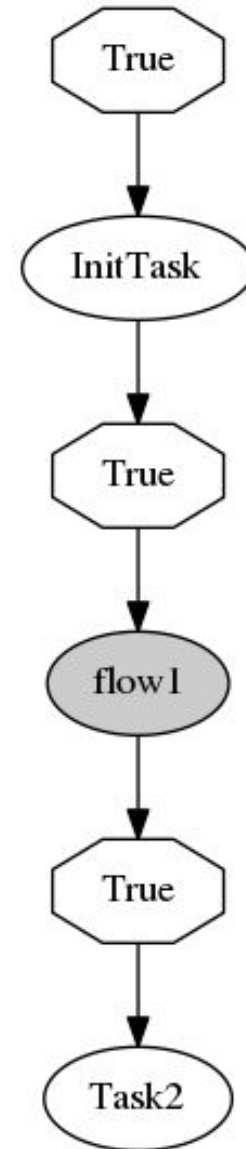
FALLBACK TASKS AND FLOWS



SUBFLOWS

YAML CONFIGURATION

```
flow-definitions:  
  - name: flow2  
    edges:  
      - from:  
        to: InitTask  
      - from: InitTask  
        to: flow1  
      - from: flow1  
        to: Task2
```



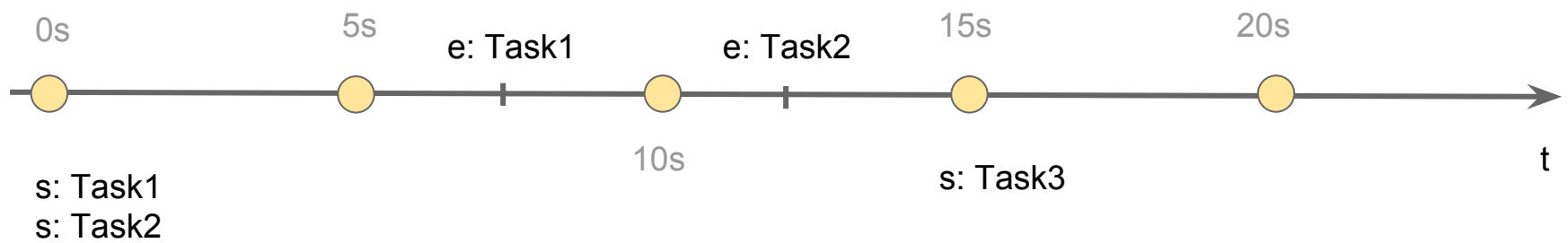
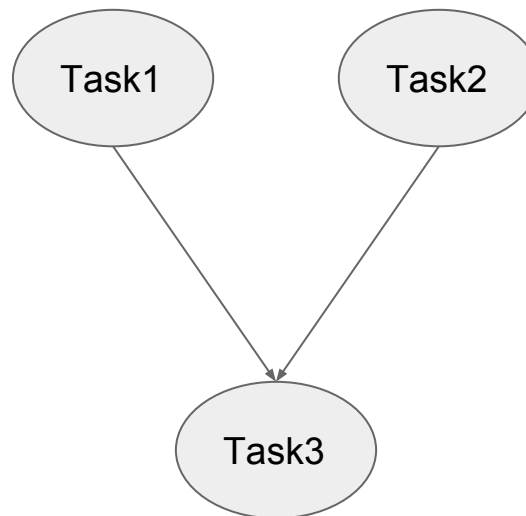
HOW DOES SELINON WORK?

SELINON



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

SELINON



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

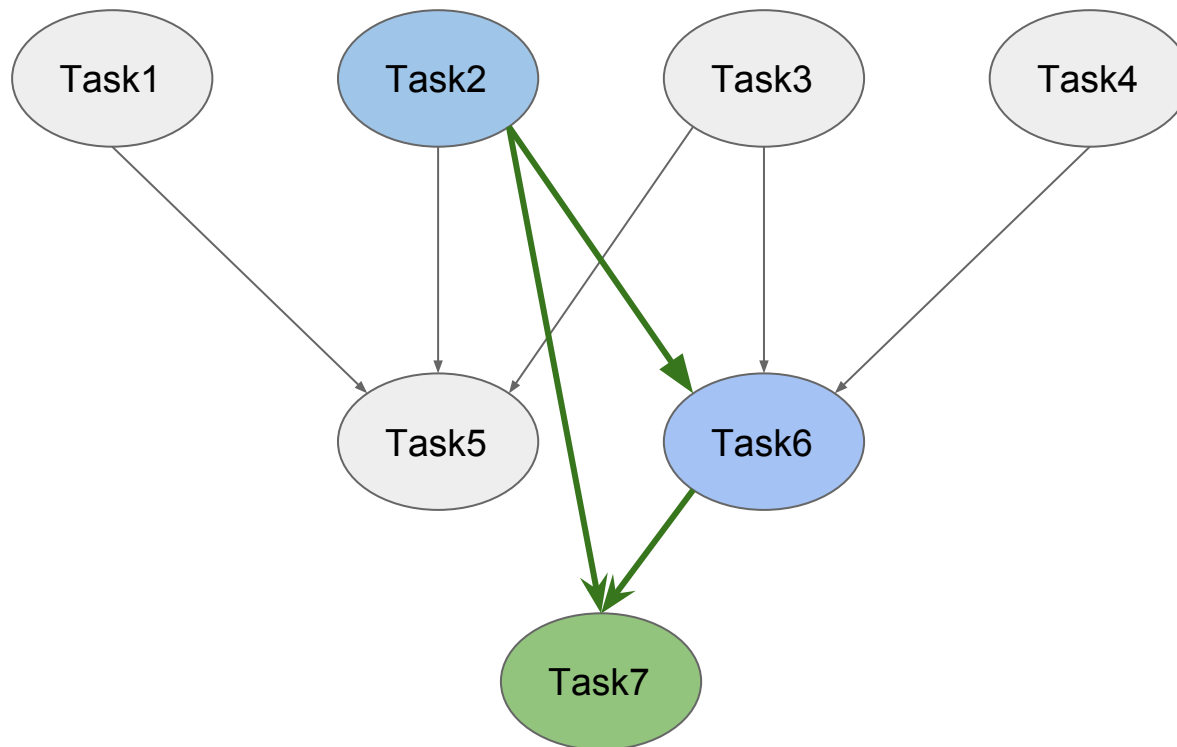
- Flow replay
- No need to recompute results



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



QUESTIONS?



<https://github.com/selinon>