Celery

A Distributed Task Queue

Idan Gazit

PyWeb-IL 8 / 29th September 2009

What is Celery?

Celery is a...

Distributed
Asynchronous
Task Queue
For Django

Celery is a...

Distributed
Asynchronous
Task Queue
For Django

Celery is a...

Distributed
Asynchronous
Task Queue
For Django

Celery is a...

Distributed
Asynchronous
Task Queue
For Django

Celery is a...

Distributed
Asynchronous
Task Queue
~~For Django~~

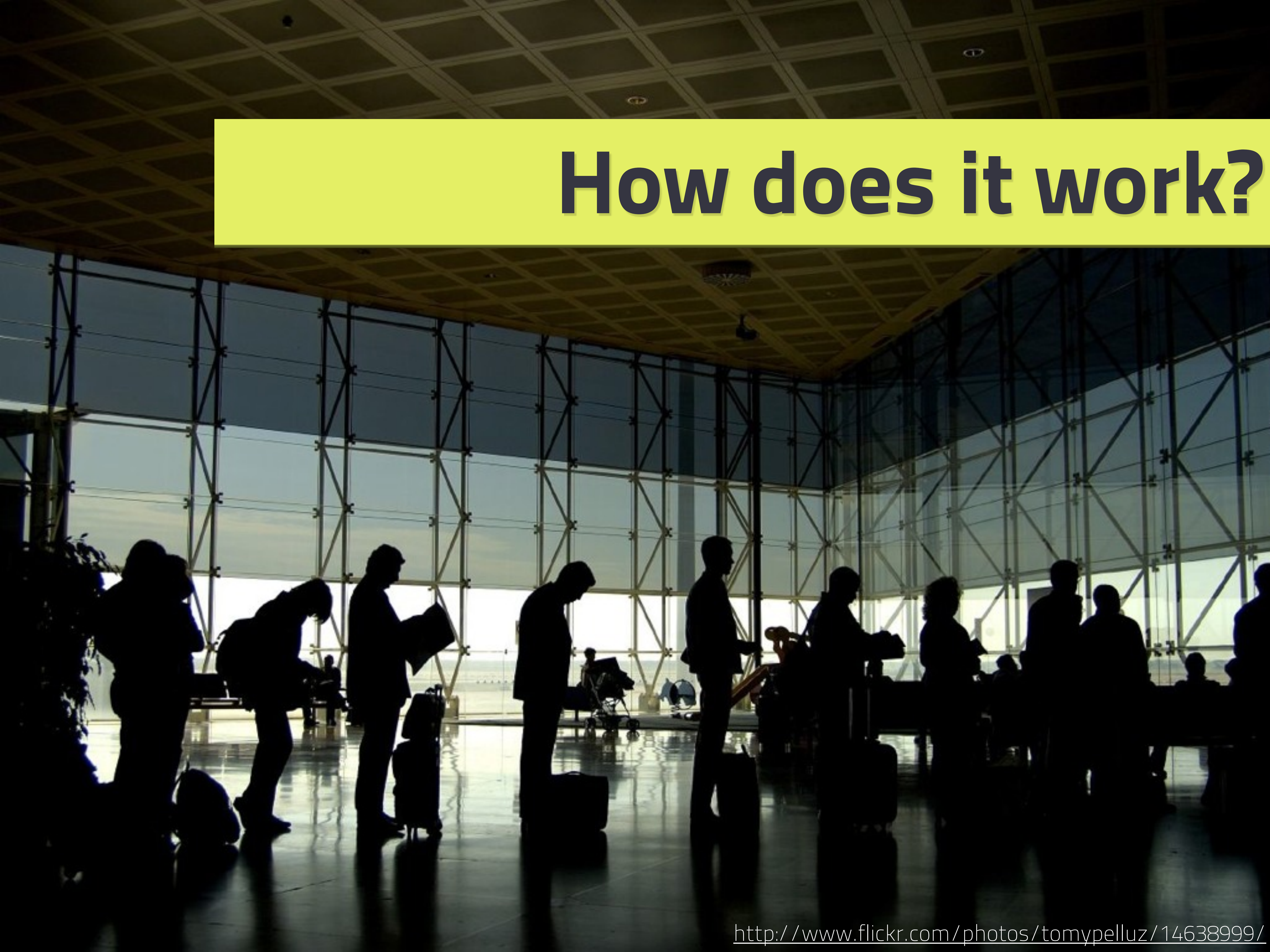
since 0.8

What can I use it for?

http://www.flickr.com/photos/jabzg/2145312172/

Potential Uses

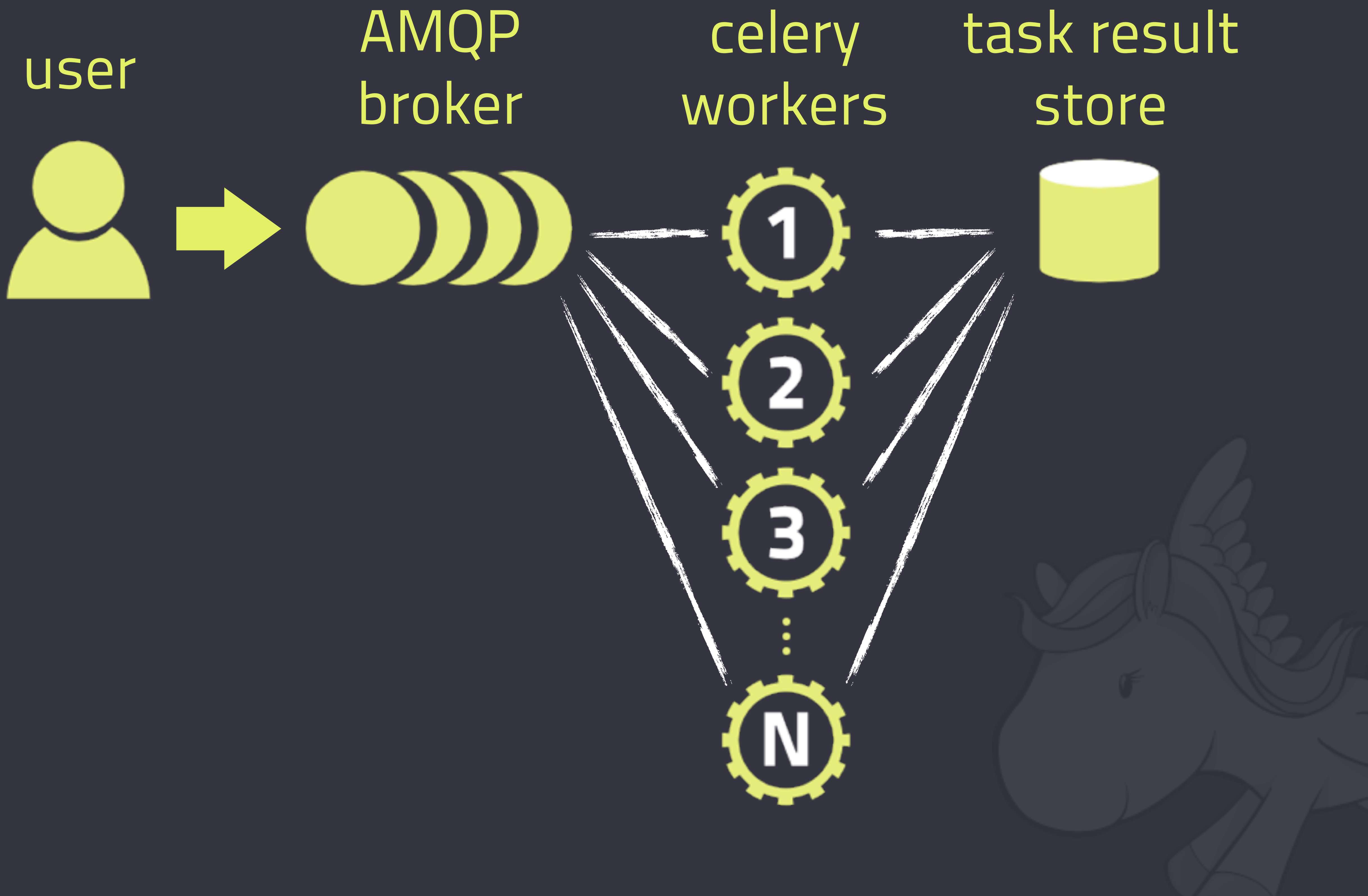
- **Anything that needs to run asynchronously, e.g. outside of the request-response cycle.**
- Background computation of 'expensive queries' (ex. denormalized counts)
- Interactions with external API's (ex. Twitter)
- Periodic tasks (instead of cron & scripts)
- Long-running actions with results displayed via AJAX.



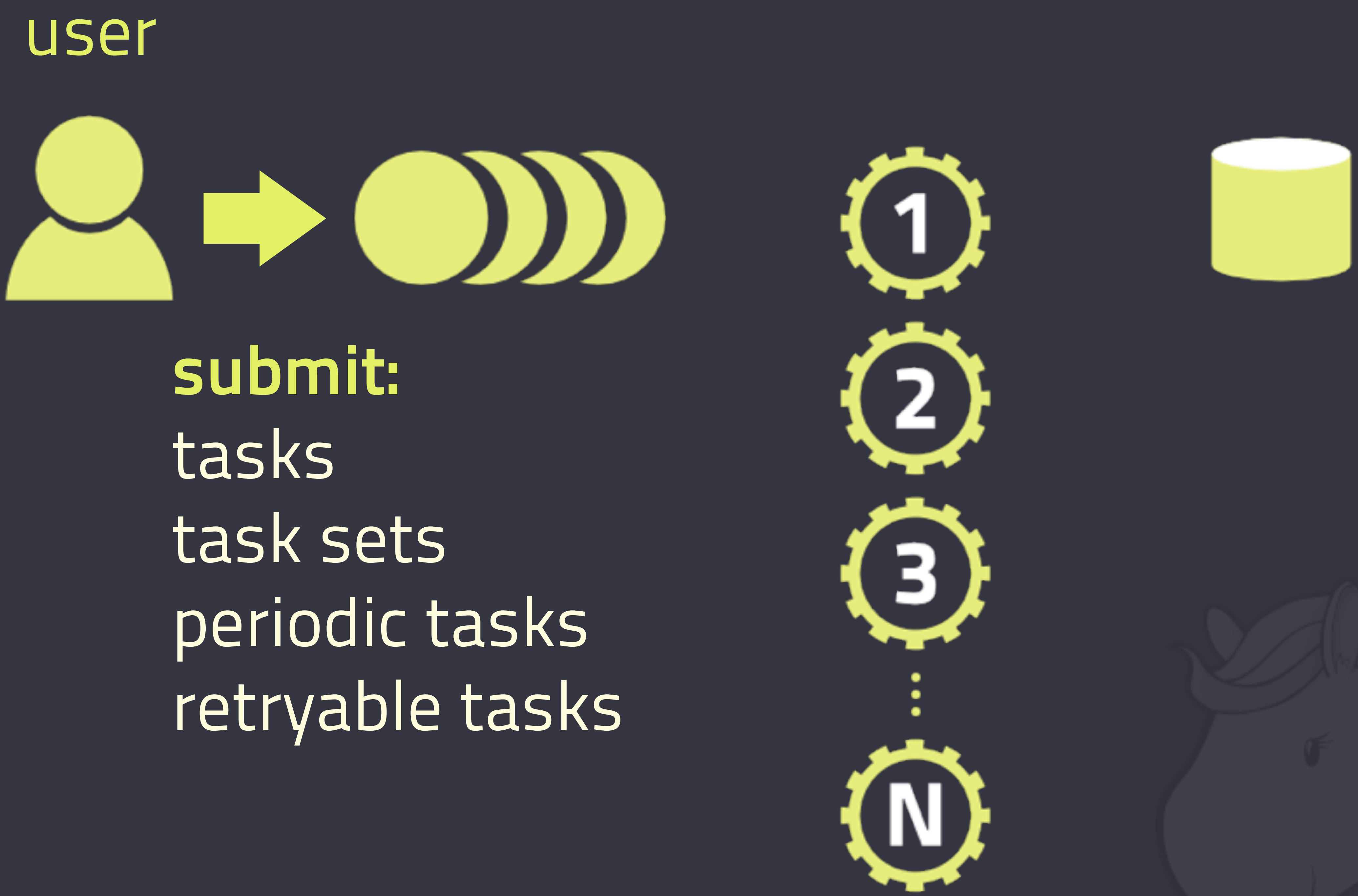
How does it work?

http://www.flickr.com/photos/tomypelluz/14638999/

Celery Architecture



Celery Architecture



submit:

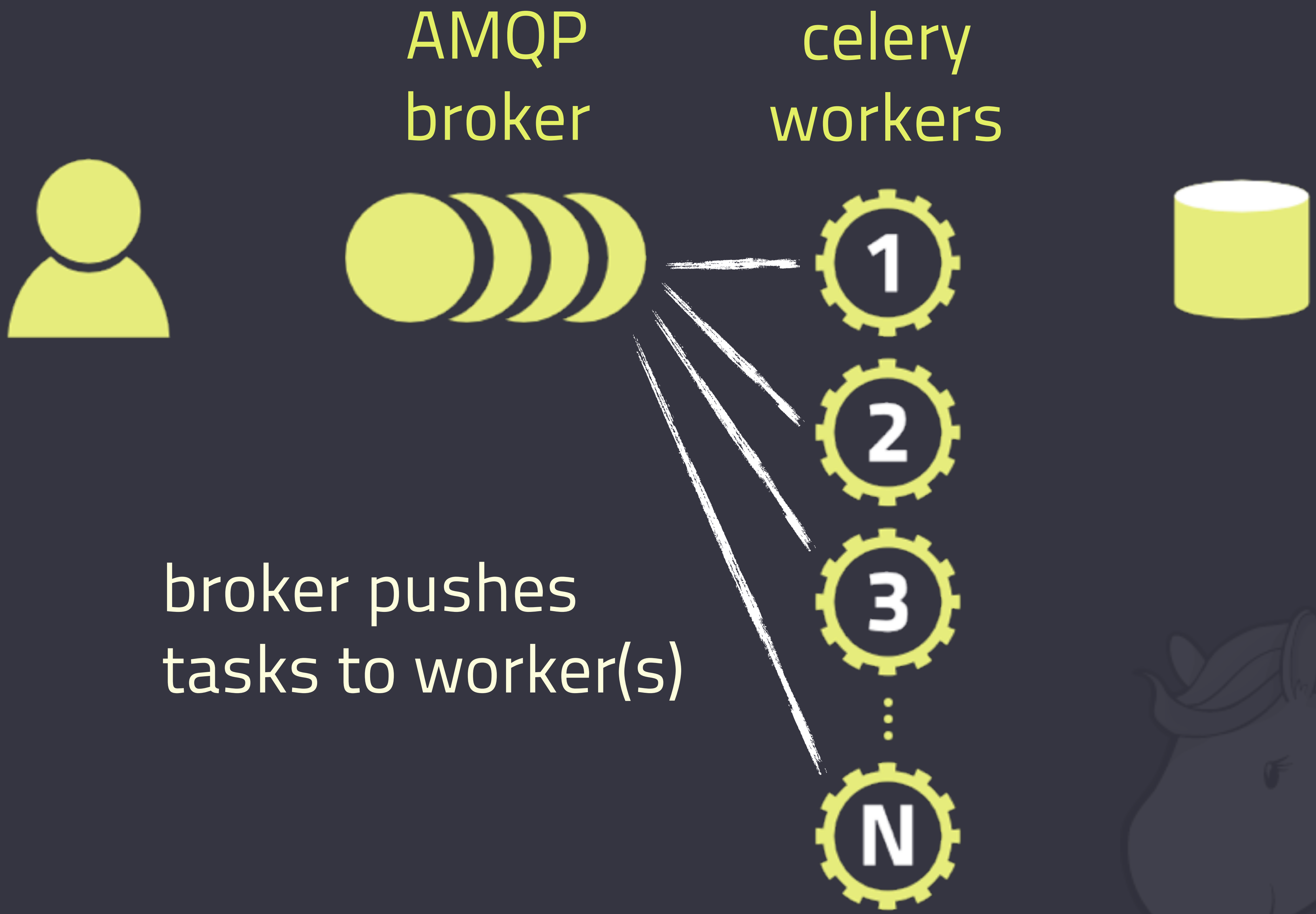
tasks

task sets

periodic tasks

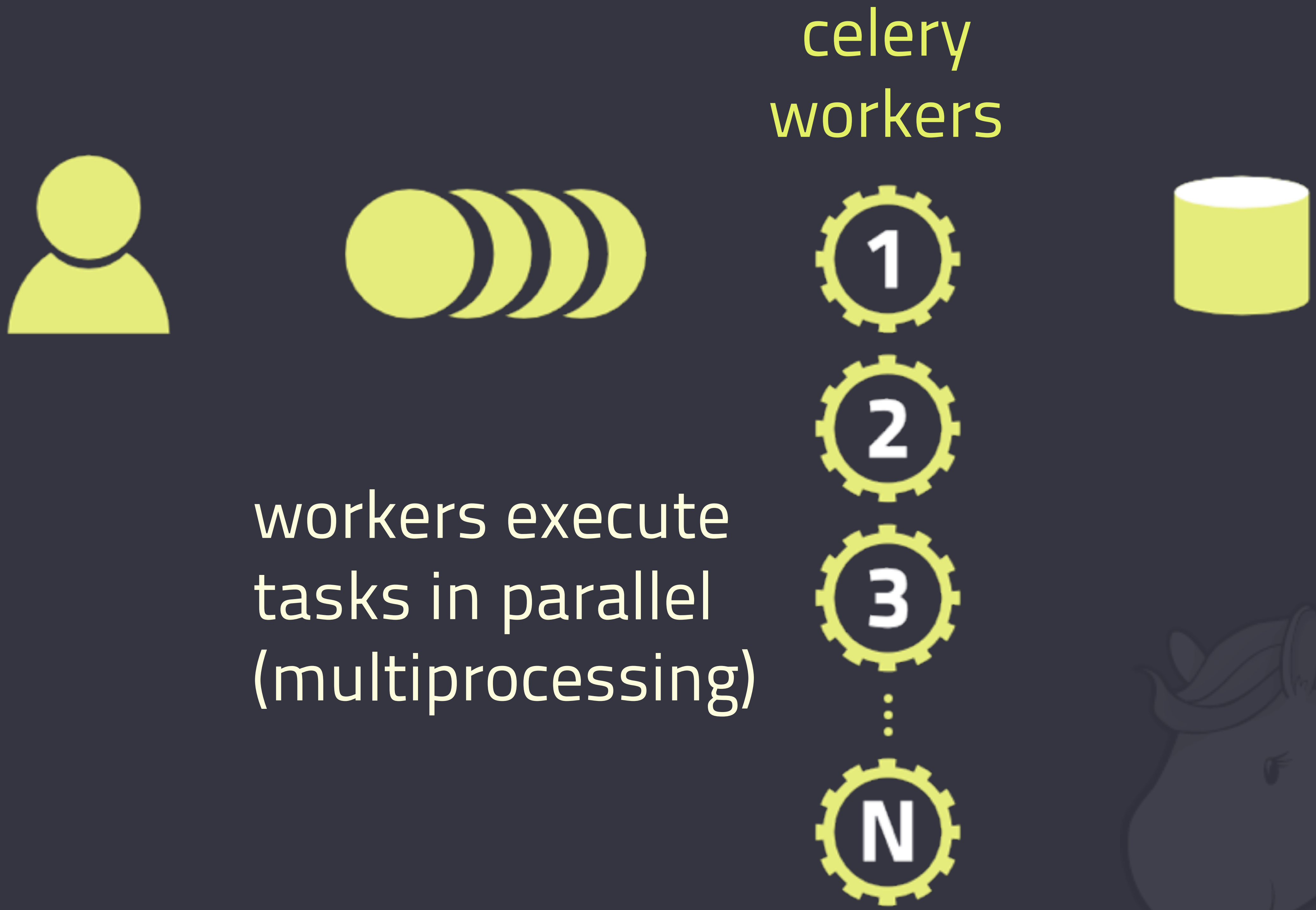
retryable tasks

Celery Architecture



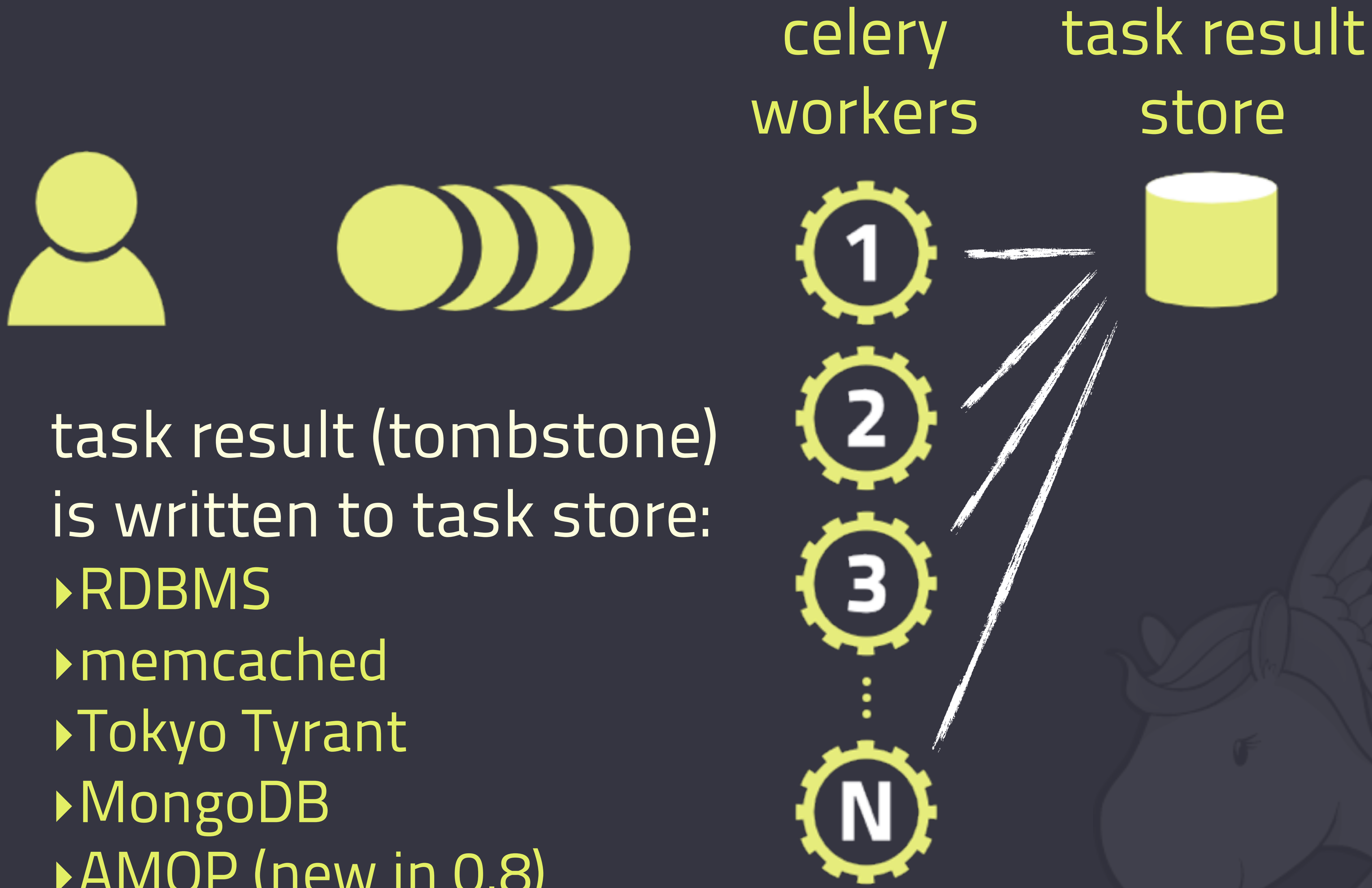
broker pushes
tasks to worker(s)

Celery Architecture



workers execute
tasks in parallel
(multiprocessing)

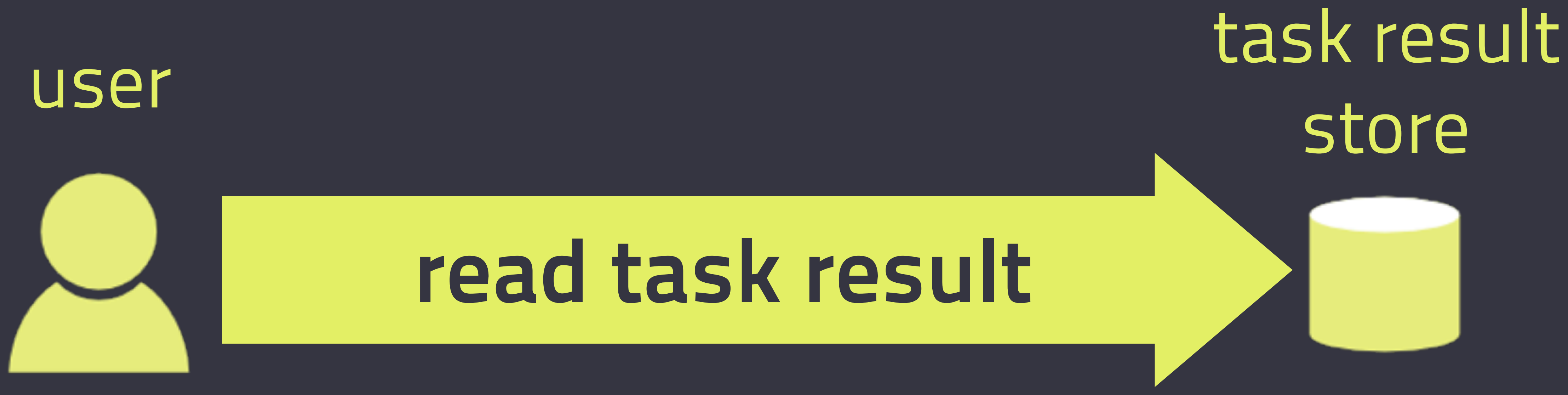
Celery Architecture



task result (tombstone) is written to task store:

- RDBMS
- memcached
- Tokyo Tyrant
- MongoDB
- AMQP (new in 0.8)

Celery Architecture



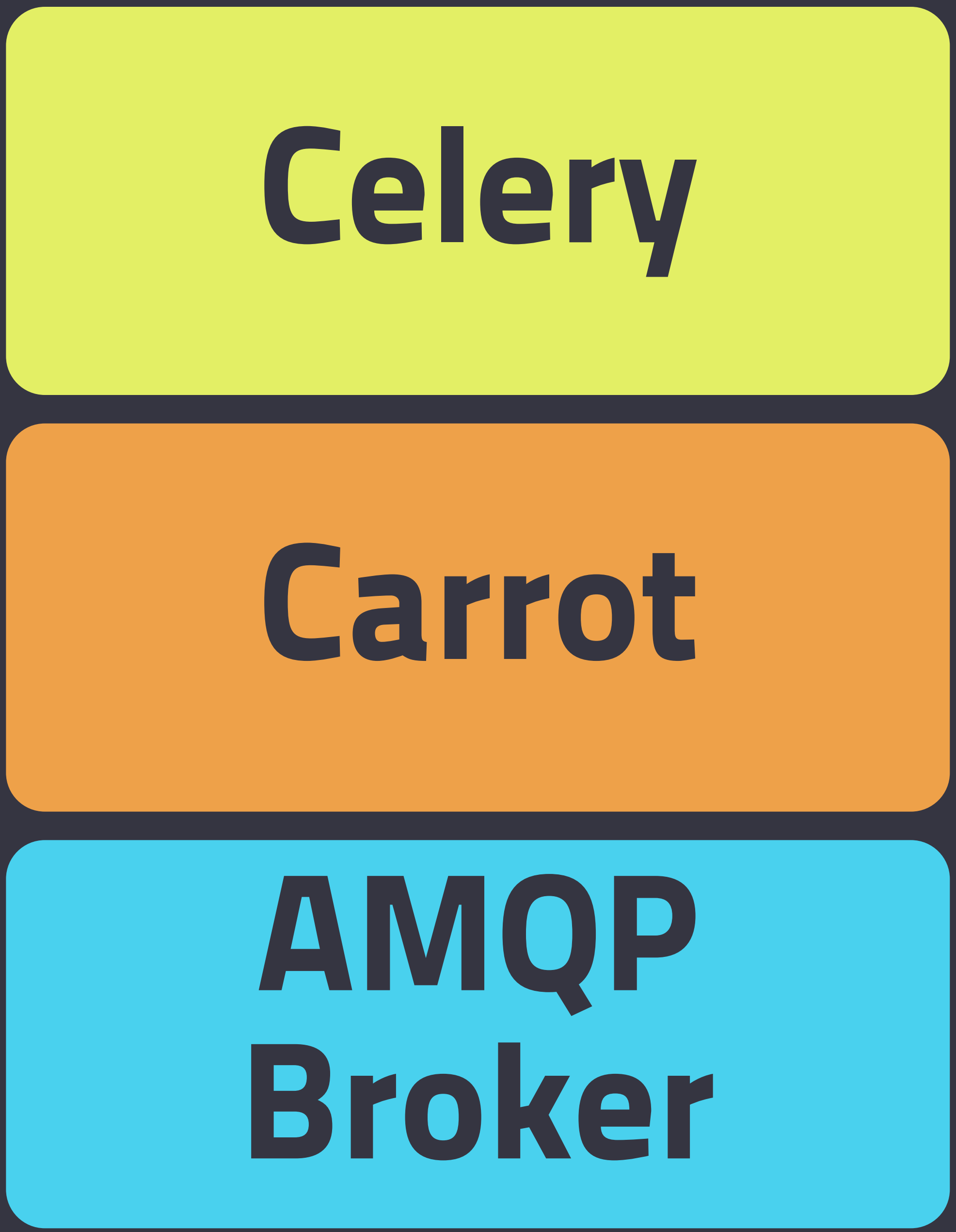
Celery Architecture

Celery uses...

Carrot to talk to...

AMQP Broker

Celery Architecture



pip install celery

(dependency of celery)

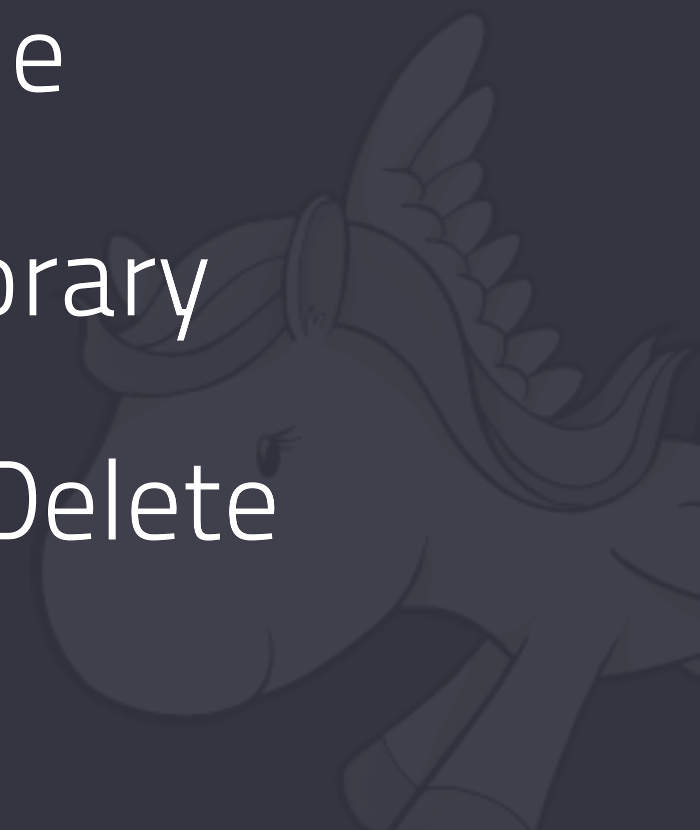
Celery Architecture

Celery	pip install celery
Carrot	(dependency of celery)
RabbitMQ	http://www.rabbitmq.com

AMQP is... Complex.

AMQP is Complex

- VHost
- Exchanges
  - Direct
  - Fanout
  - Topic
- Routing Keys
- Bindings
- Queues
  - Durable
  - Temporary
  - Auto-Delete



bit.ly/amqp_intro

I Can Haz Celery?

Adding Celery

1. get & install an AMQP broker
   (pay attention to vhosts, permissions)

2. add Celery to INSTALLED_APPS

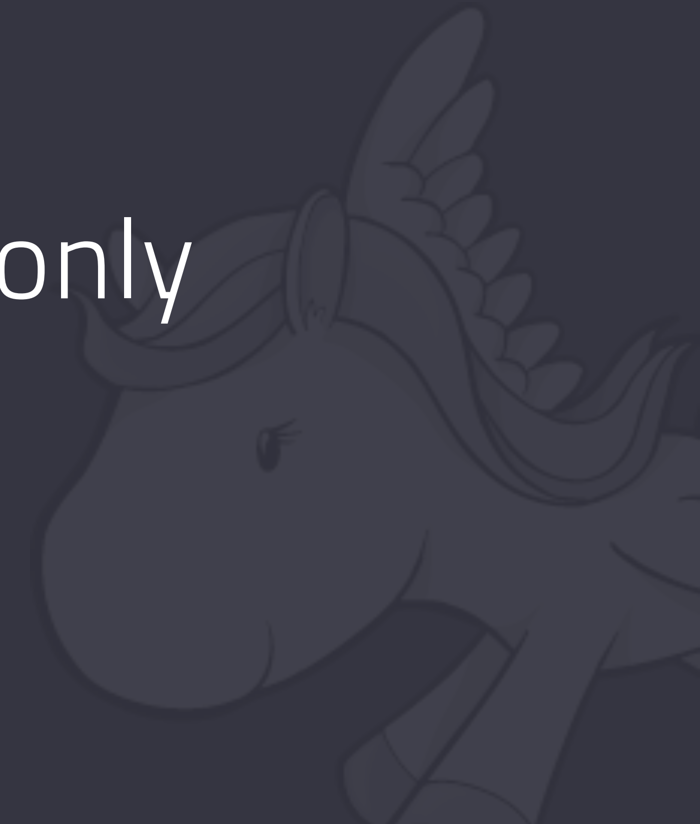
3. add a few settings:

```
AMQP_SERVER = "localhost"
AMQP_PORT = 5672
AMQP_USER = "myuser"
AMQP_PASSWORD = "mypassword"
AMQP_VHOST = "myvhost"
```

4. manage.py syncdb

Celery Workers

- Run at least 1 celery worker server
- manage.py celeryd
  (--detatch for production)
- Can be on different machines
- Celery guarantees that tasks are only executed once



Tasks

Tasks

- Define tasks in your app
- app_name/tasks.py
- register & autodiscovery (like admin.py)

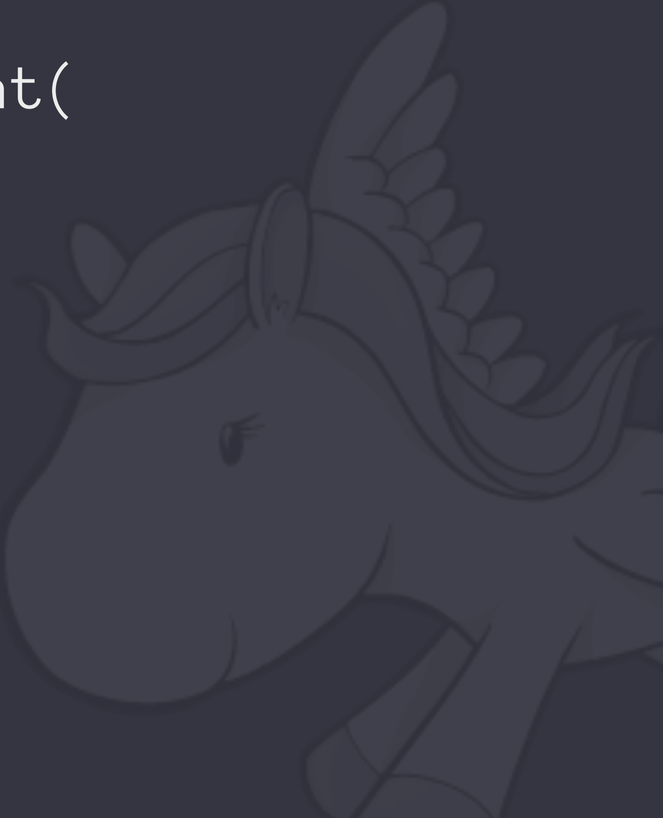
Task

```python
from celery.task import Task
from celery.registry import tasks

class FetchUserInfoTask(Task):
    def run(self, screen_name, **kwargs):
        logger = self.get_logger(**kwargs)
        try:
            user = twitter.users.show(id=screen_name)
            logger.debug("Successfully fetched {0}".format(screen_name))
        except TwitterError:
            logger.error("Unable to fetch {0}: {1}".format(
                screen_name, TwitterError))
            raise

        return user

tasks.register(FetchUserInfoTask)
```



Run It!

```
>>> from myapp.tasks import FetchUserInfoTask
>>> result = FetchUserInfoTask.delay('idangazit')
```

Task Result

- **result.ready()**
  true if task has finished
- **result.result**
  the return value of the task or exception instance if the task failed
- **result.get()**
  blocks until the task is complete then returns result or exception
- **result.successful()**
  returns True/False of task success

Why even check results?

Chained Tasks

```python
from celery.task import Task
from celery.registry import tasks

class FetchUserInfoTask(Task):
    def run(self, screen_name, **kwargs):
        logger = self.get_logger(**kwargs)
        try:
            user = twitter.users.show(id=screen_name)
            logger.debug("Successfully fetched {0}".format(screen_name))
        except TwitterError:
            logger.error("Unable to fetch {0}: {1}".format(
                screen_name, TwitterError))
            raise
        else:
            ProcessUserTask.delay(user)

        return user

tasks.register(FetchUserInfoTask)
```

Task Retries

Task Retries

```python
from celery.task import Task
from celery.registry import tasks

class FetchUserInfoTask(Task):
    default_retry_delay = 5 * 60 # retry in 5 minutes
    max_retries = 5

    def run(self, screen_name, **kwargs):
        logger = self.get_logger(**kwargs)
        try:
            user = twitter.users.show(id=screen_name)
            logger.debug("Successfully fetched {0}".format(screen_name))
        except TwitterError, exc:
            self.retry(args=[screen_name,], kwargs=**kwargs, exc)
        else:
            ProcessUserTask.delay(user)

        return user

tasks.register(FetchUserInfoTask)
```

Periodic Tasks

Periodic Tasks

```python
from celery.task import PeriodicTask
from celery.registry import tasks
from datetime import timedelta

class FetchMentionsTask(Task):
    run_every = timedelta(seconds=60)

    def run(self, **kwargs):
        logger = self.get_logger(**kwargs)
        mentions = twitter.statuses.mentions()
        for m in mentions:
            ProcessMentionTask.delay(m)

        return len(mentions)

tasks.register(FetchMentionsTask)
```

Task Sets

Task Sets

```
>>> from myapp.tasks import FetchUserInfoTask
>>> from celery.task import TaskSet
>>> ts = TaskSet(FetchUserInfoTask, args=(
            ['ahikman'], {},
            ['idangazit'], {},
            ['daonb'], {},
            ['dibau_naum_h'], {}))
>>> ts_result = ts.run()
>>> list_of_return_values = ts.join()
```

MOAR SELRY!

Celery.Views

Celery.Views

- Celery ships with some django views for launching / getting the status of tasks.
- JSON views perfect for use in your AJAX (err, AJAJ) calls.
- celery.views.apply(request, task_name, *args)
- celery.views.is_task_done(request, task_id)
- celery.views.task_status(request, task_id)

Routable Tasks

Routable Tasks

- "I want tasks of type X to only execute on this specific server"
- Some extra settings in settings.py:

```
CELERY_AMQP_EXCHANGE = "tasks"
CELERY_AMQP_PUBLISHER_ROUTING_KEY = "task.regular"
CELERY_AMQP_EXCHANGE_TYPE = "topic"
CELERY_AMQP_CONSUMER_QUEUE = "foo_tasks"
CELERY_AMQP_CONSUMER_ROUTING_KEY = "foo.#"
```

- set the task's routing key:

```
class MyRoutableTask(Task):
    routing_key = 'foo.bars'
```

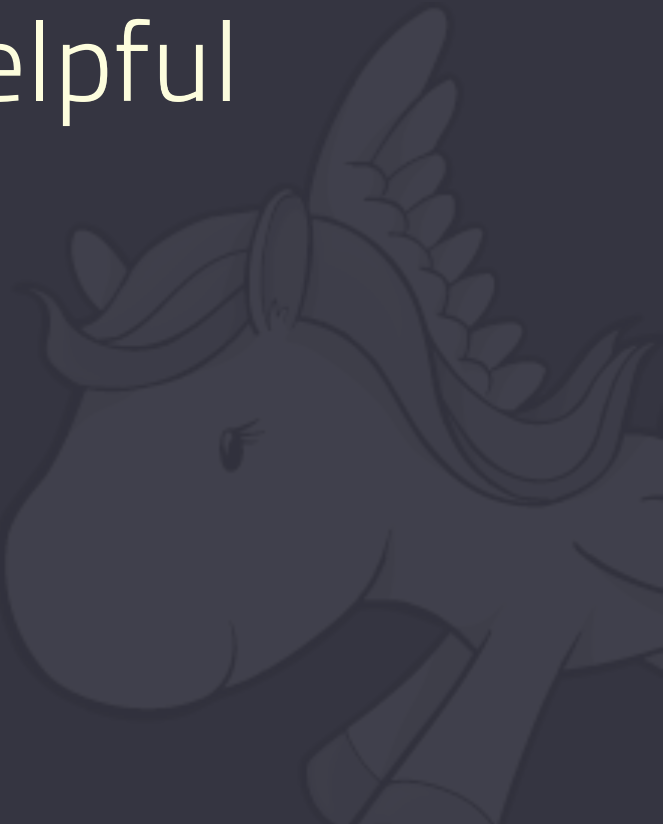
like django, it's just python.

Support

#celery on freenode

http://groups.google.com/group/celery-users/

AskSol (the author) is friendly & helpful



Fin.

@idangazit
idan@pixane.com