Python Celery

The Distributed Task Queue

Mahendra M

@mahendra

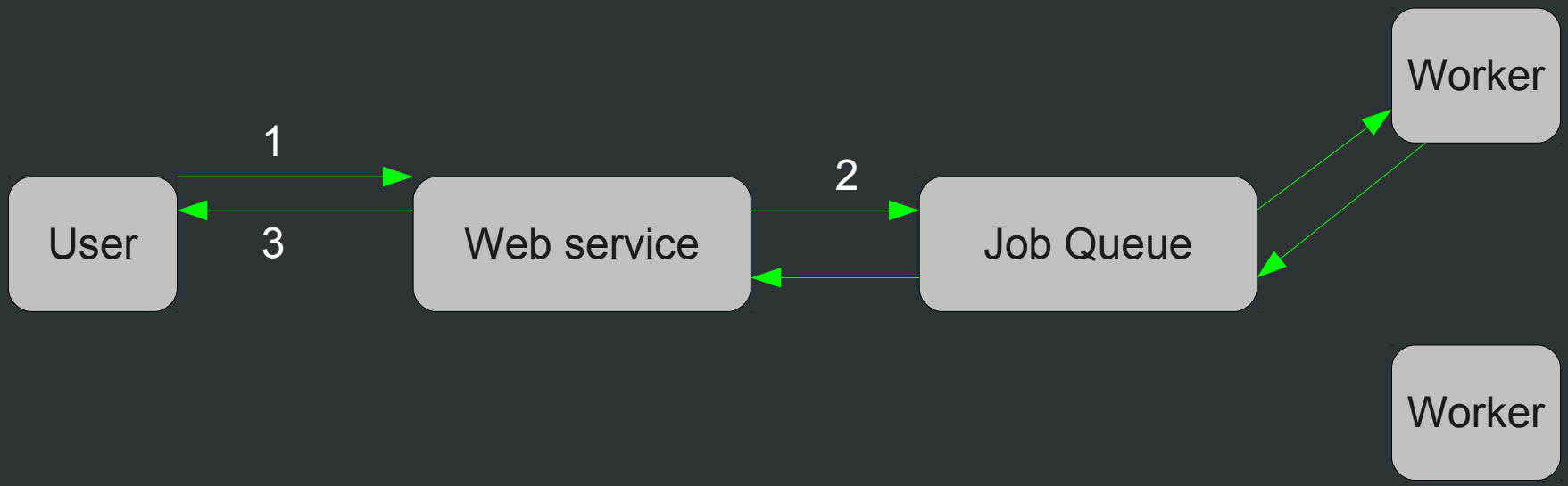
About me

- Solutions Architect at Infosys, Product Incubation Group
- Worked on FOSS for 10 years
- BLUG, FOSS.in (ex) member
- Linux, NetBSD embedded developer
- Mostly Python programmer
    - Mostly sticks to server side stuff
- Loves CouchDB and Twisted

Job Queues – the need

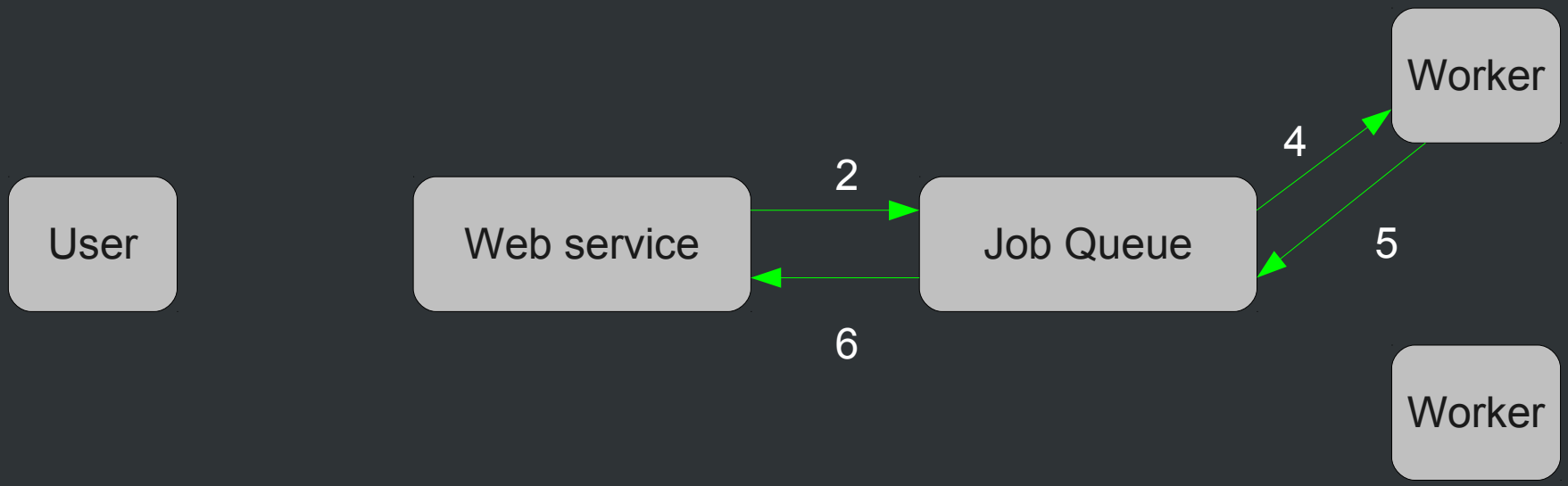
- More complex systems on the web
- Asynchronous processing is required
  - Flickr – Image resizing
  - User profiles – synchronization
- Asynchronous database updates
  - Click counters
  - Likes, favourites, recommend

How it works



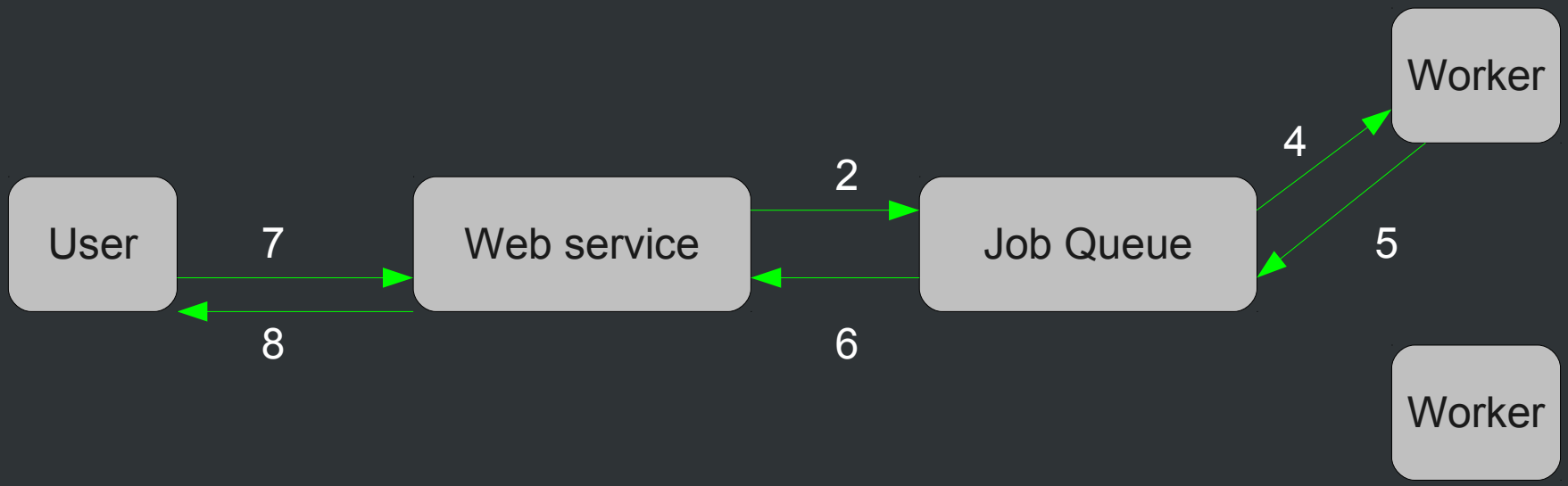
Placing a request

How it works



The job is executed

How it works

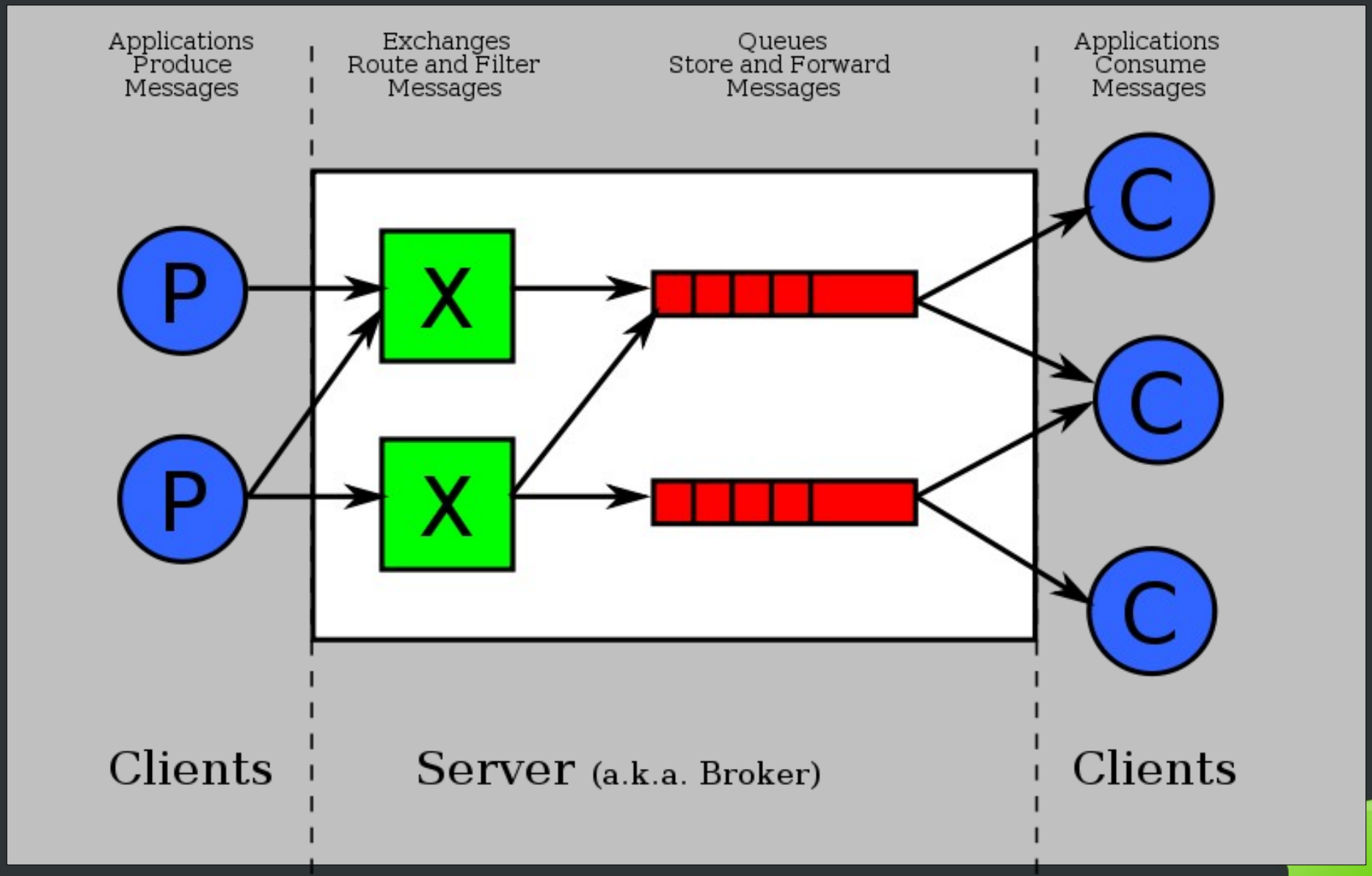


Client fetches the result

AMQP

- Advanced Message Queuing Protocol
  - Self explanatory :-)
- Open, language agnostic, implementation agnostic
- Transmits messages from producers to consumers
- Immensely popular
- Open source implementations available.

AMQP


Applications
Produce
Messages
Exchanges
Route and Filter
Messages
Queues
Store and Forward
Messages
Applications
Consume
Messages
P
P
X
X
C
C
C
Clients
Server (a.k.a. Broker)
Clients

AMQP

Queues are bound to exchanges to determine message delivery

- Direct - From Exchange to Queue
- Topic - Queue is selected based on a topic
- Fanout – All queues are selected
- Headers – based on message headers

AMQP as a job queue

- AMQP structure is similar to our job queue design
- Jobs are sent as messages
- Job results are sent back as messages
- Celery Framework simplifies this for us

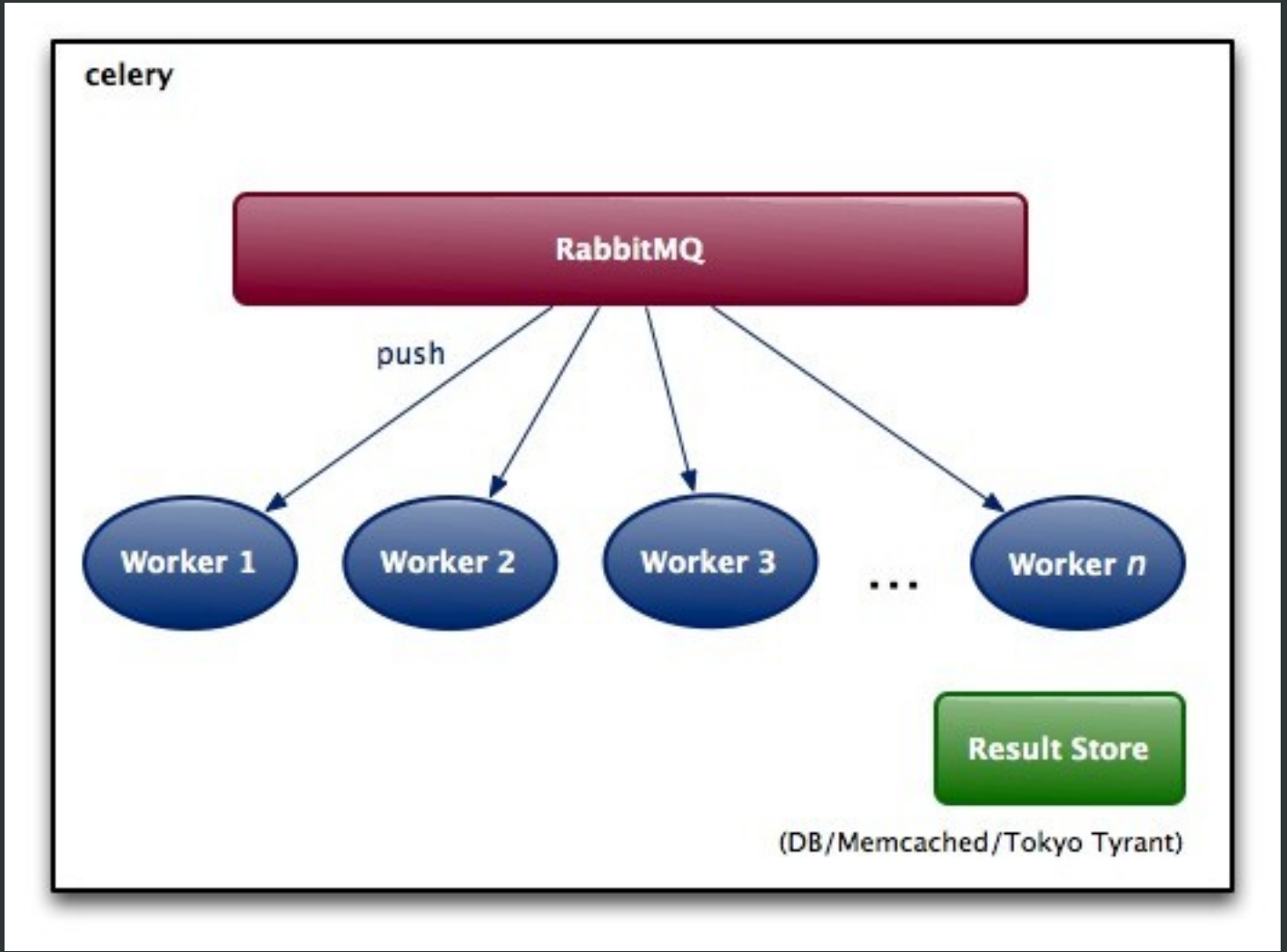
Python Celery

- Python based distributed task queue built on top of message queues
- Very robust, good error handling, guaranteed ...
- Distributed (across machines)
- Concurrent within a box
- Supports job scheduling (eta, cron, date, …)
- Synchronous and asynchronous operations
- Retries and task grouping

Python Celery ...

- Web hooks
- Job Routing
    - based on AMQP message routing
- Remote control of workers
    - Rate limit, delete, revoke tasks
- Monitoring
- Tracebacks of errors, Email notifications
- **Django Integration**

Celery Architecture



Defining a task

```
from celery.decorators import task

@task
def add(x, y):
    return x + y


>>> result = add.delay(4, 4)
>>> result.wait() # wait for result
8
>>>
```

Running a task

- Synchronous run
  - task.apply(…)
- Asynchrnous
  - task.apply_async(…)
- Tasksets – schedule task with different arguments
  - Think of it like map reduce
- Scheduled execution
- Auto retries and max_retry support
- Ensure worker availability

Django Features

- 'djcelery' in INSTALLED_APPS
- Uses Django features
  - ORM for storing task details
  - settings.py for configuration
  - Celery commands are part of django commands
  - Run celery workers using manage.py
  - Task registeration and auto discovery - tasks.py
- Schedule jobs directly from view handlers
- View handlers for task status monitoring via Ajax

Demo

Advantages of AMQP

- Scaling is an ”admin” job
  - Workers can be added and removed any time
  - Scale on need basis
  - Deploy on cloud setups
- Jobs can routed to workers based on admin setups
- Jobs can be prioritized based on AMQP protocol
  - Not supported in rabbitmq (not sure of 2.x)
- Can be deployed on a single node also.

Where should I use

- Background computations
- Anything outside the request-response cycle
- Run System commands or applications
    - Imagemagick (convert) for resize
- Integration with external systems (APIs)
- Use webhooks for Integrating independent systems
- Result aggregations (db updates, like, ratings ..)

Where to avoid ?

- Ensure that you absolutely need a task queue
- Sometimes it might be easier to avoid it
- Simple database updates / inserts (log like)
- Sending emails/sms (it is already a message queue ...)

Links

- http://celeryproject.org
- http://celery.org/docs/getting-started/
- http://amqp.org/
- http://en.wikipedia.org/AMQP
- http://rabbitmq.org/
- http://slideshare.net/search/slideshow?q=celery