Going Beyond Data Parallelism

The adventures ahead!*

- Meeting the "narrator"**
- What is Data Parallelism & When is it not enough
- A detour to Appendix A from my Ray book
- My side quest: to promote my books
- My employers (probable) goal: make you interested in Netflix data engineering

Who am I?

- Pronouns are she/her
- Apache Spark PMC (think committer with tenure)
- previously Apple, IBM, Alpine, Databricks, Google, Foursquare & Amazon
- co-author of High Performance Spark, Learning Spark, Kubeflow for Machine Learning + in progress Scaling Python with Dask and Scaling Python with Ray
- Twitter: @holdenkarau
- Livestreams: https://youtube.com/user/holdenkarau
- Github https://github.com/holdenk
- Currently at **Netflix (my org is hiring)** - but any mistakes are my own

Probable (relevant) Biases

- I'm used to working with large scale datasets
- I've mostly worked at the platform level for the past decade
- I'm a Spark committer and I've written some of it
- I have contributed code to Ray and Dask but much less
- I've written books on Spark and am writing books on Ray and Dask
- I think functional programming is *cool*

Quick Refresher on Data Parallelism

- Split up the date into partitions
- Most of the time: apply the same logic to each partition
- Sometimes: re-combine the partitions in some way

- Or see: Distributed Computing 4 Kids

When is this not enough?

- Tracking state & weights during ML models (current top of mind)
- Smaller tasks
- Non-uniform tasks
- etc.

What are some options?

Local:

- Joblib, multiprocessing
- Locks + Shared memory

Distributed:

- Tasks
- Actors
- Locks + shared memory
- DB

Why this is hard:

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable'" – Leslie lamport

What do (distributed) tasks look like?

tl;dr – Functions with decorators

Dask Distributed Tasks

```
@dask.delayed
def remote_hi():
    import os
    import socket
    return f"Running on {socket.gethostname()} in pid {os.getpid()}"
```

Ray Distributed Tasks

```
@ray.remote
def remote_hi():
    import os
    import socket
    return f"Running on {socket.gethostname()} in pid
{os.getpid()}"
```

How are they different?

Dask Delayed

- Default* to lazy
- Centralized* scheduler

Ray Remote

- Default to eager (futures)
- Distributed* scheduler

How are they same?

- Distributed & Local Scheduler options
- Chainable
- Recursive*
- Low (but non-zero) overhead
- Futures available
- etc.

Task Fault tolerance

- Restart on failure
- Yes this can have some "unintended" side effects

And we can (sort of) make the Distributed look local

- https://docs.ray.io/en/latest/ray-more-libs/joblib.html
- https://ml.dask.org/joblib.html

And same for multiprocessing etc.

Does Spark have tasks?

Yes… but not exposed

And Actors?

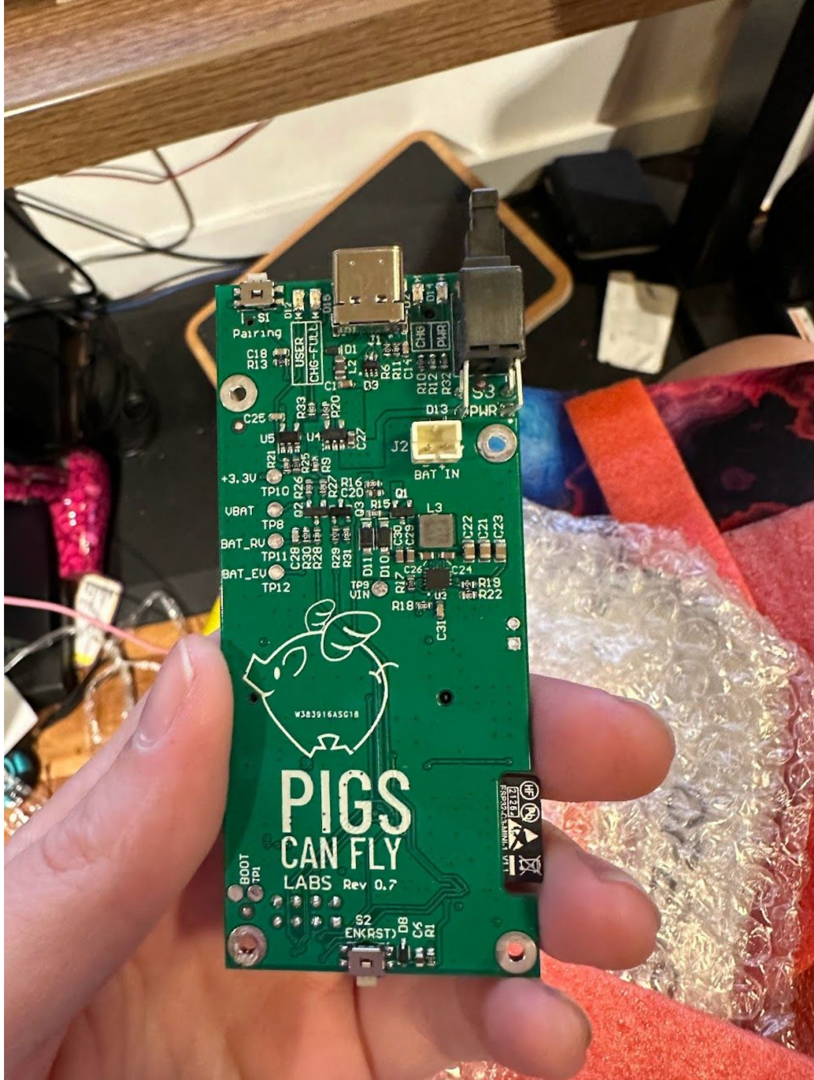
Think like tasks + restrictions to make handling state "easier."

Communicate with message passing

Encapsulate state

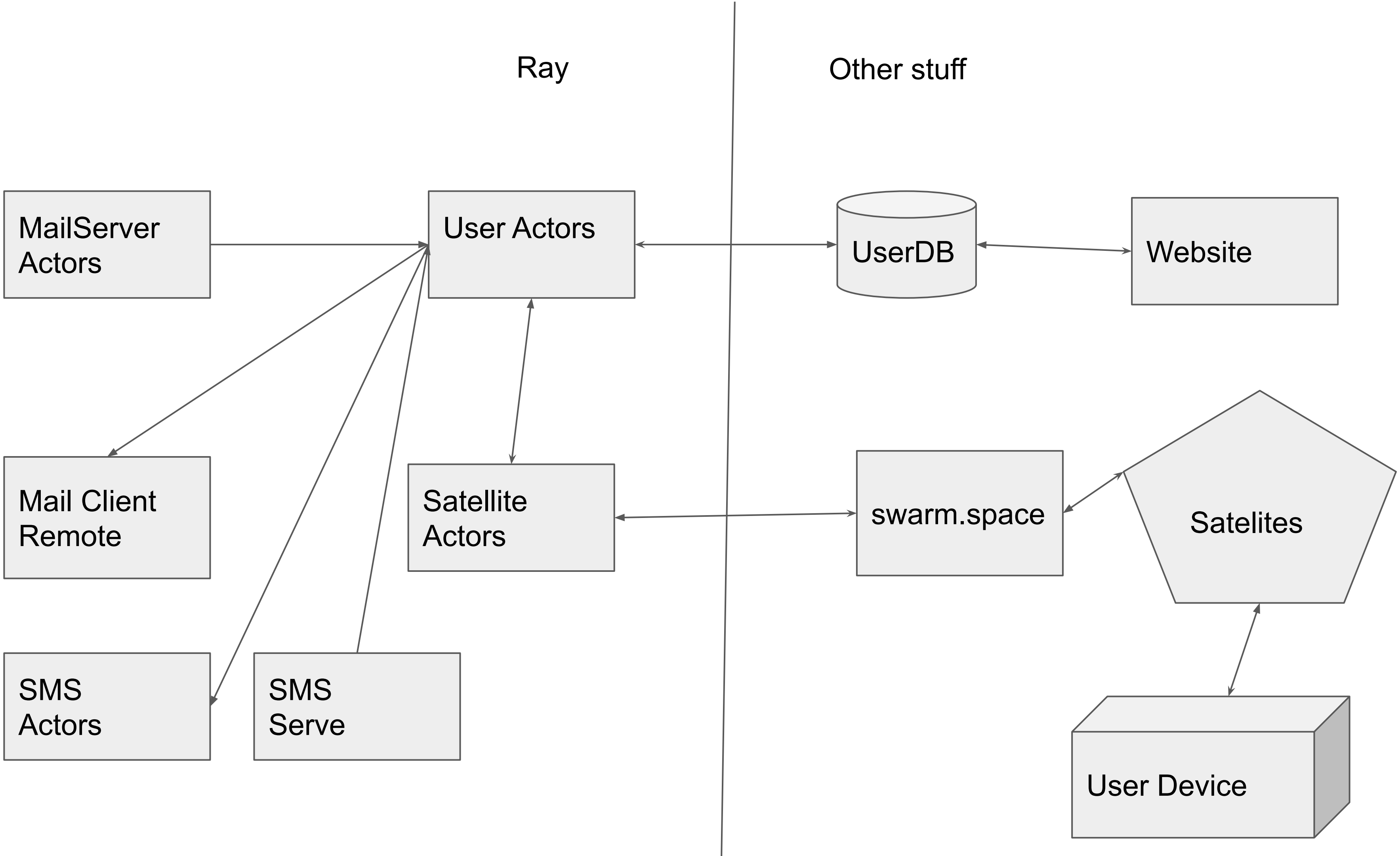
Neat! What does it look like?

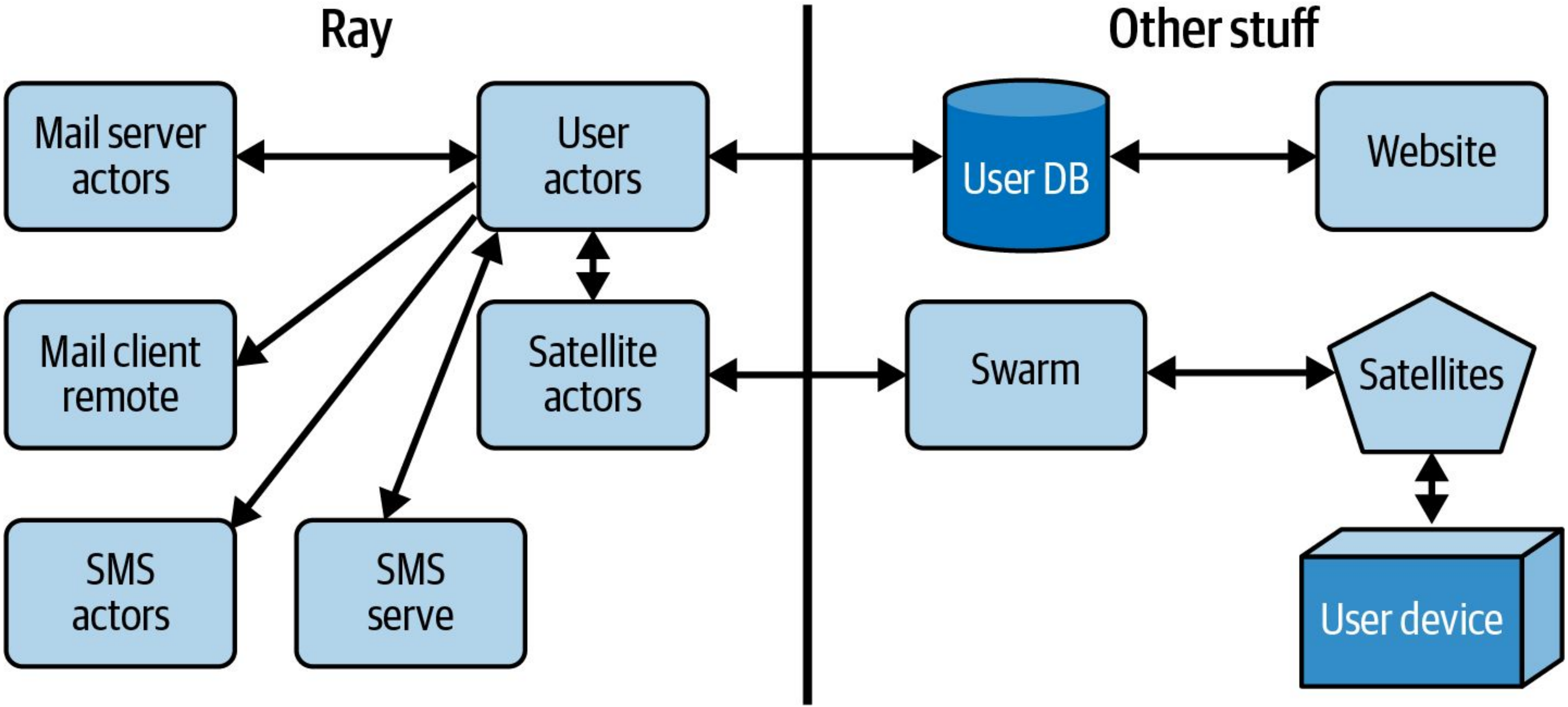
```python
class SatelliteClientBase():
    """
    Base client class for talking to the swarm.space APIs.
    """

    def __init__(self, settings: Settings, idx: int, poolsize: int):
        # Annoying setup work goes here
```



```python
async def run(self):
    # Is it there yet?
    print("Prepairing to run.")
    self.running = True
    while self.running:
        try:
            self._login()
            while True:
                await asyncio.sleep(self.delay)
                await self.check_msgs()
        except Exception as e:
            print(f"Error {e} while checking messages.")
            logging.error(f"Error {e}, retrying")
    async def send_message(self, protocol: int, msg_from: str, msg_to: int, data: str):
        messagedata = MessageDataPB()  # noqa
        messagedata.from_device = False
        message = messagedata.message.add()
        message.text = data
        message.protocol = protocol
        message.to = msg_from
        encoded = base64.b64encode(messagedata.SerializeToString())
        request_dict = {
            "deviceType": 0,
            "deviceId": msg_to,
            "userApplicationId": 1000,
            "data": encoded
        }
        request_encoded = json.dumps(request_dict)
        return self.session.post(
            self._sendMessageURL,
            data=request_encoded,
            headers=self._sendMessageHeaders
```

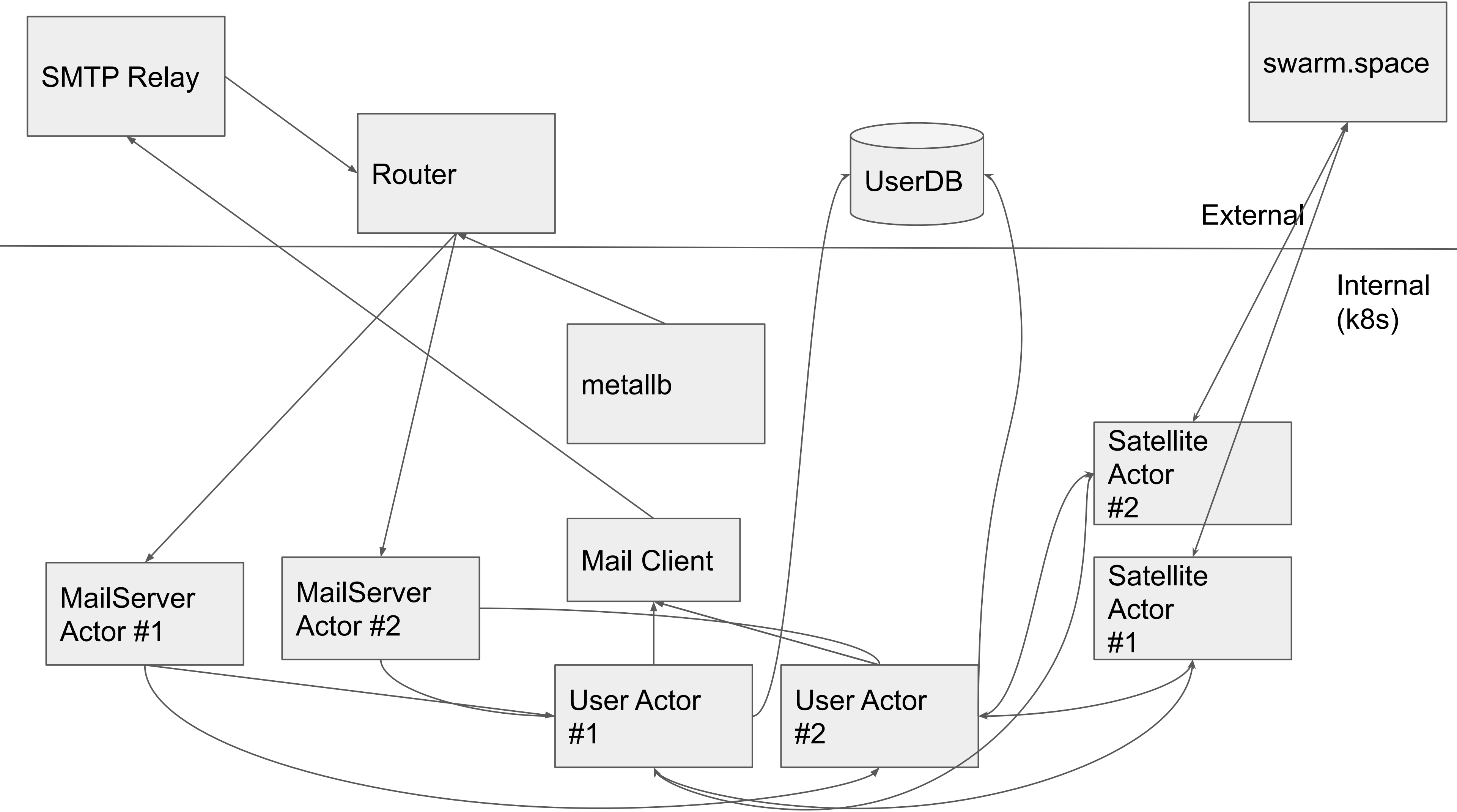
```python
# This is the magic that makes it an actor :D (We separate this out so we can test a non-actor version too).

@ray.remote(max_restarts=-1)

class SatelliteClient(SatelliteClientBase):

    """

    Connects to swarm.space API.

     """
```


Ray
Other stuff
MailServer Actors
User Actors
UserDB
Website
Mail Client Remote
Satellite Actors
swarm.space
Satelites
SMS Actors
SMS Serve
User Device


Ray
Other stuff
Mail server actors
User actors
User DB
Website
Mail client remote
Satellite actors
Swarm
Satellites
SMS actors
SMS serve
User device

Oook what happens if it gets "busy"?

- Well… unlikely given our project
- Actor pools give us what we want, but initialization order
- Routing the messages becomes complicated


SMTP Relay
Router
UserDB
swarm.space
External
Internal
(k8s)
metallb
Satellite
Actor
#2
Mail Client
MailServer
Actor #1
MailServer
Actor #2
Satellite
Actor
#1
User Actor
#1
User Actor
#2

Ray Actor Fault tolerance

Mark them as restartable, but you need to write the recovery code.

There is no magic here (except maybe a database).

So… code?

https://github.com/PigsCanFlyLabs/message-backend-ray

+

https://github.com/scalingpythonml/scalingpythonml

Ok ok fine. What's up with Ray + Netflix?

- We train models with Ray :D
- No we don't make Satellite Communication backends :p
- See https://netflixtechblog.com/scaling-media-machine-learning-at-netflix-f19b400243

Dask Actor Fault tolerance

Hopes and dreams

Does Spark have Actors?

No

A word from my employer:

We are actively hiring for the Data Platform organization (remote and in person)

I don't think it's my specific team, but it's on our sister teams :)

https://jobs.netflix.com/teams/data-platform

As well as DSE etc (w/ remote US roles)

But most importantly….

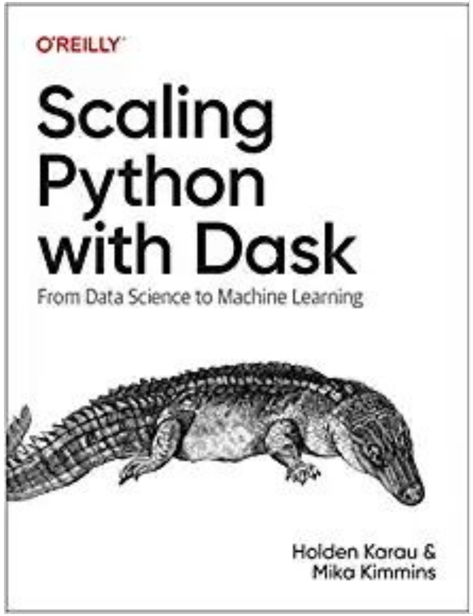
Buy several copies of my books :p (or read them on safari, I think I get money from that?)

O'REILLY

High Performance Spark

BEST PRACTICES FOR SCALING & OPTIMIZING APACHE SPARK

Holden Karau & Rachel Warren

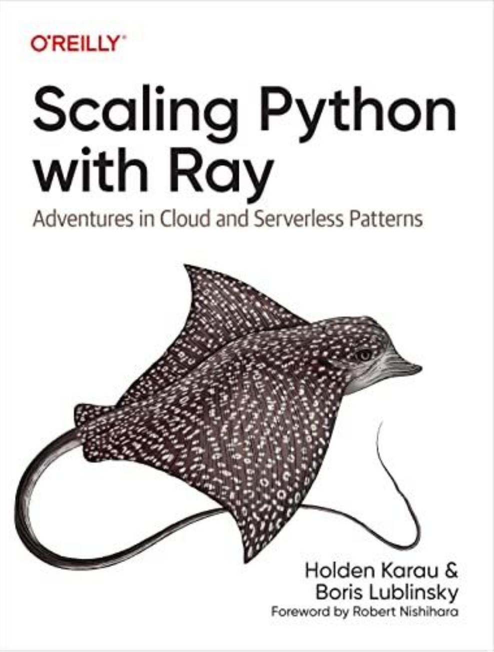


O'REILLY

Scaling Python with Dask

From Data Science to Machine Learning

Holden Karau & Mika Kimmins



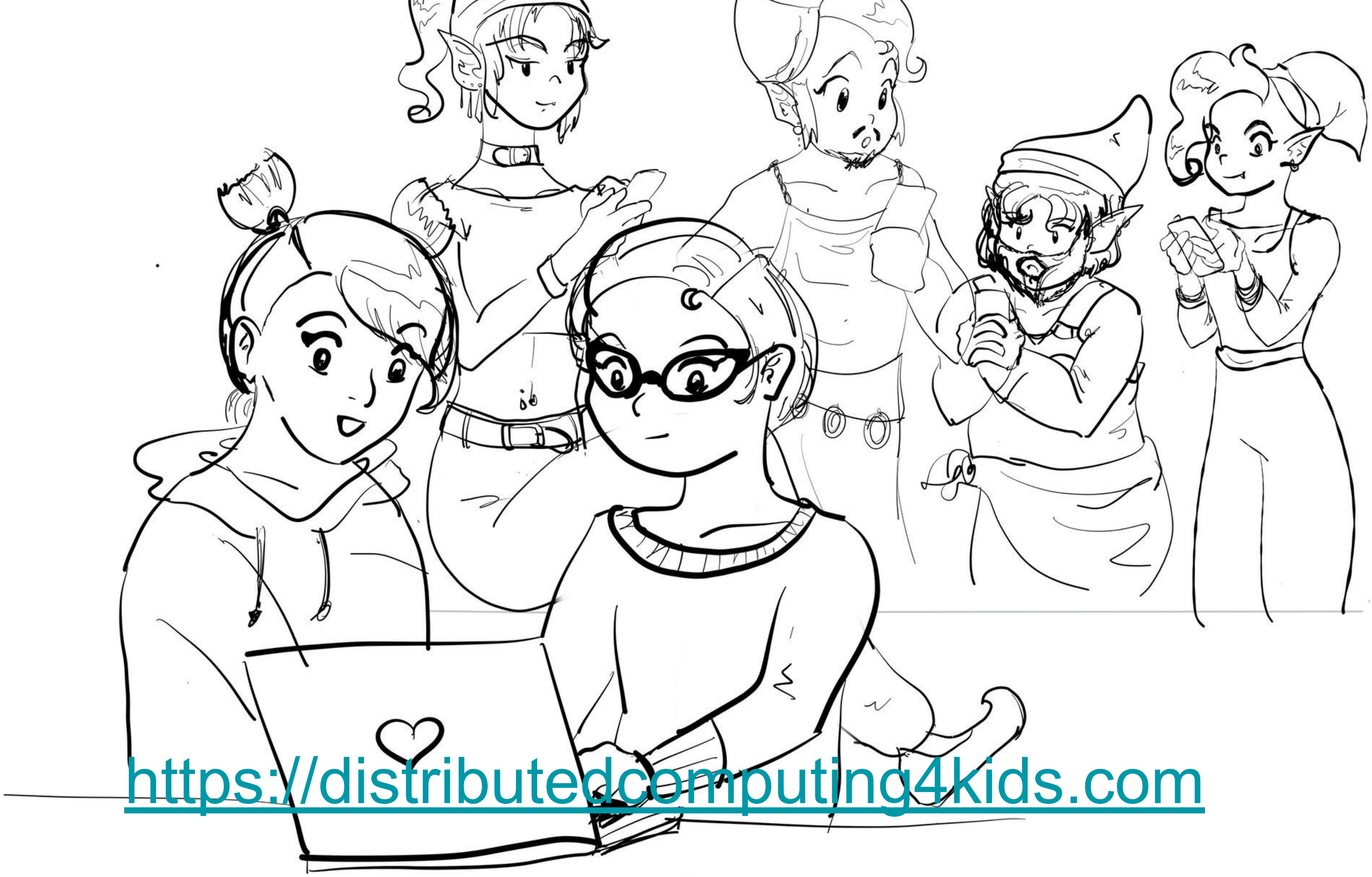
O'REILLY

Scaling Python with Ray

Adventures in Cloud and Serverless Patterns

Holden Karau & Boris Lublinsky

Foreword by Robert Nishihara



https://distributedcomputing4kids.com