Plénière Communauté DS #3

Python évolue, ne ratons pas le train !

Plan

- Types annotations
- Coroutines / Async
- Aller plus loin

Types annotations

Annotation optionnelle des types de variables (PEP 484 - May 2015)

```
def sum(a: int, b: int) -> int:
    return a + b

sum("hello", " world")

# $ python ex1.py
# $ mypy ex1.py
# ex1.py:5: error: Argument 1 to "sum" has incompatible type "str"; expected "int"
# ex1.py:5: error: Argument 2 to "sum" has incompatible type "str"; expected "int"
```

Types annotations - Intérêts

- Pratique : Utilisé par les IDE pour vérifier et aider
- Largement adopté : Numpy, Pandas, Scikit, etc.
- Déviation de l'utilisation des annotations
  - Documentation du code / API (`Swagger`)
  - Vérification des types de données en entrée (`Typer`, `FastAPI`)
  - Validation des structures de données (`Pydantic`)
- Long terme : ouvre la porte à de nouveaux Runtimes plus optimisés

Types annotations - Outils

- `typing` module
  - Définit une hiérarchie de types pour aller du delà du `int` et `str`
  - Exemple : `Optionnal[Callable[[int], str]]`
  - Fournit les bases pour construire ses propres types
- `mypy` : runtime Python qui vérifie les types
  - Pas performant (ne pas utiliser en runtime par défaut)
  - Utilisé surtout pour les checks (black < flake8 < mypy)

Types annotations - Pydantic

```
class User(BaseModel):
    id: int
    name = "John Doe"
    signup_ts: Optional[datetime] = None
    friends: List[int] = []

User(**{"id": "123", "signup_ts": "2019-06-01 12:22", "friends": [1, 2, "3"]})
# id=123 signup_ts=datetime.datetime(2019, 6, 1, 12, 22) friends=[1, 2, 3] name='John Doe'

User(**{"id": "123", "signup_ts": "ds"})
# pydantic.error_wrappers.ValidationError: 1 validation error for User signup_ts
#   invalid datetime format (type=value_error.datetime)i
```

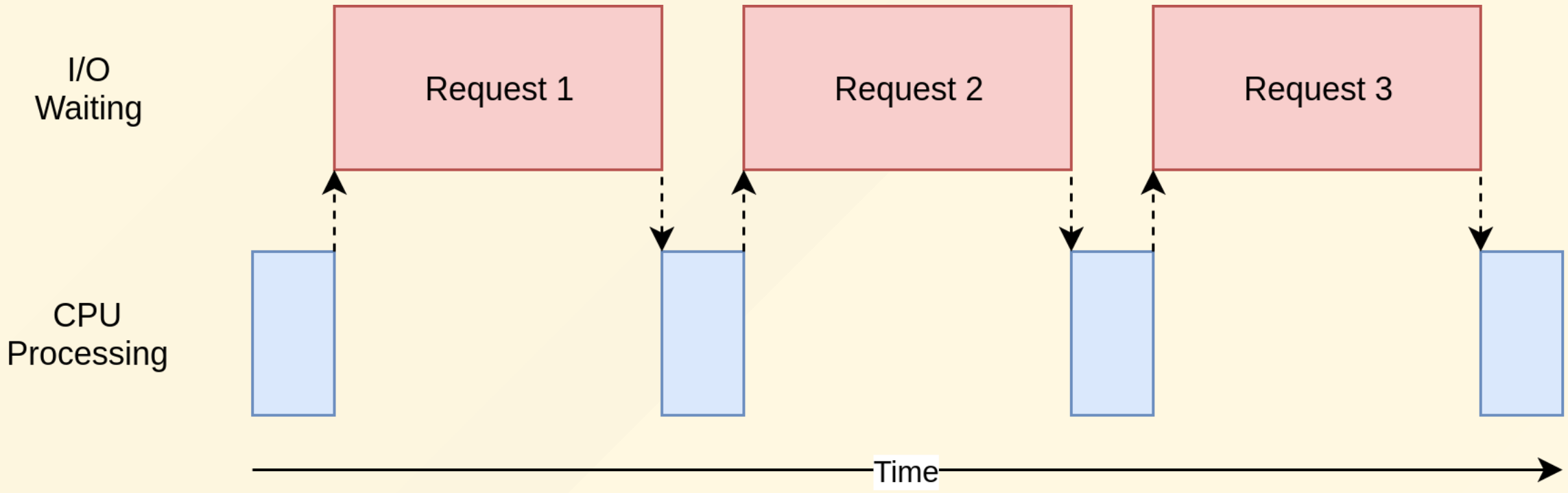
Types annotions - Take aways

- Prendre comme habitude d'annoter (**partiellement** et sans se contraindre)
- Arrêter les **commentaires** qui documentent les types
- Inclure **mypy** dans notre chaine CI/CD (Tests)

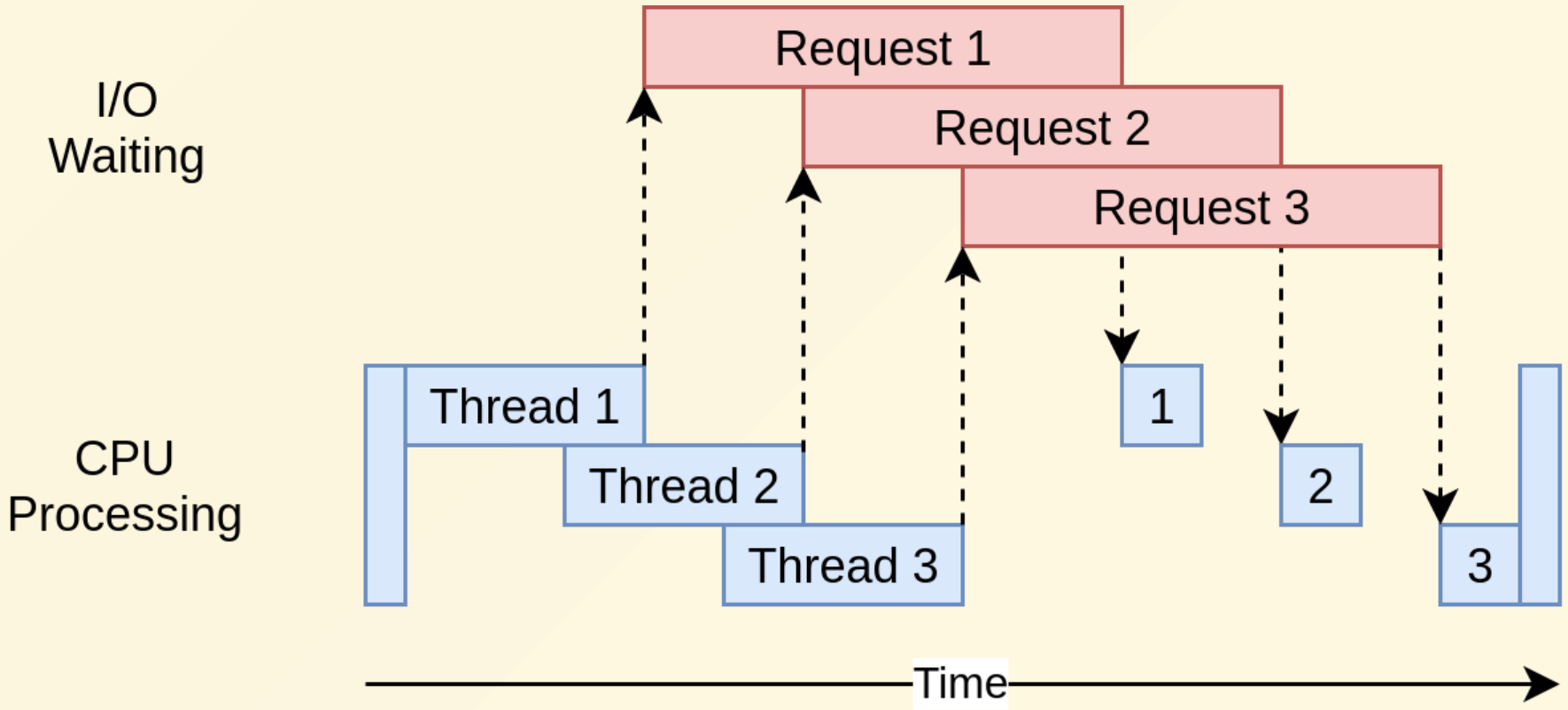
Asynchrone

- Paradigme démocratisé par `Nodejs` pour optimiser l'implémentation de serveurs Web
- Avant : 1 connexion = 1 thread dédié qui gère la réponse
- Problème : Les threads passent leur temps à attendre (des interruptions)
- Après : Queue d'exécution / Pool de workers / On rend la main sur interruption
- Utilisation efficace des ressources

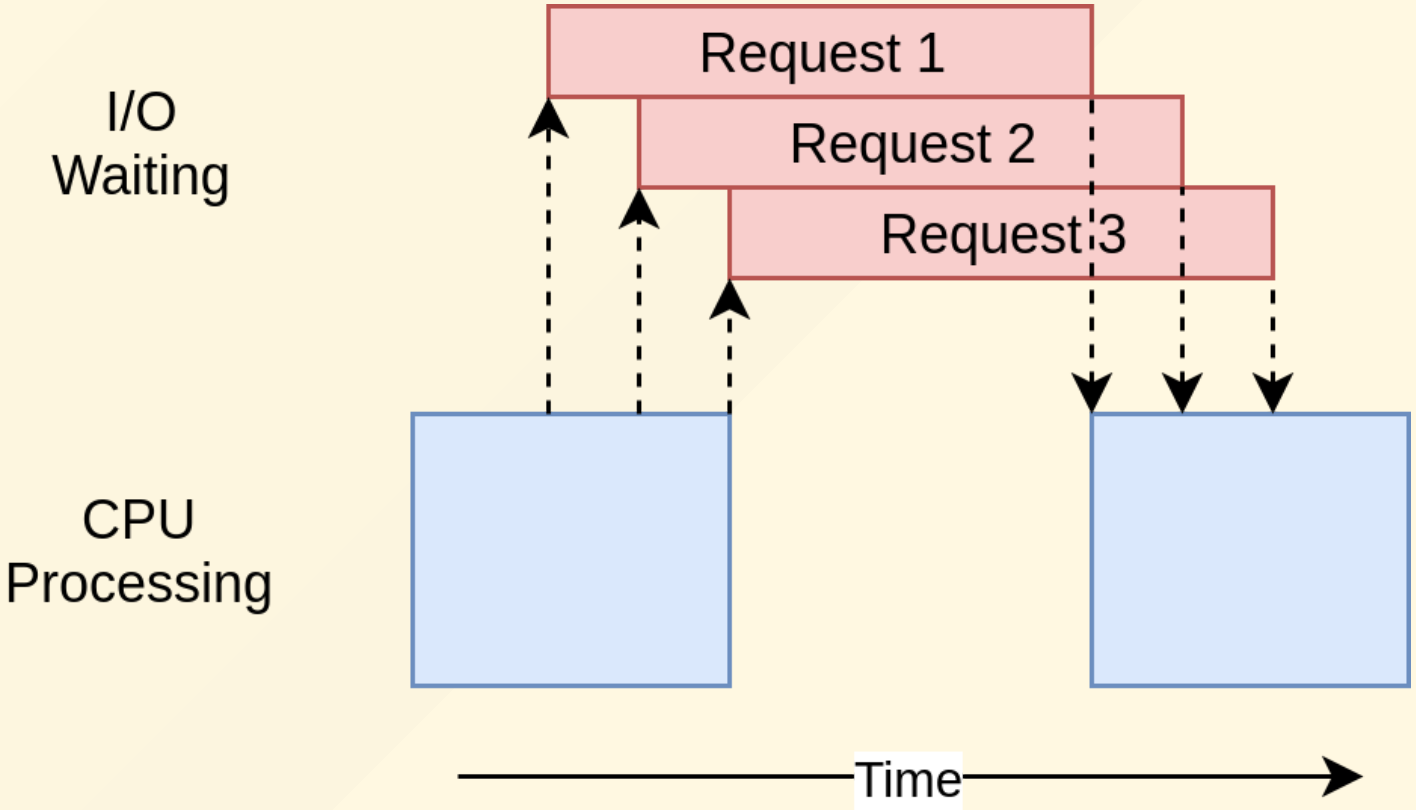
Asynchrone - Illustration - Long time ago



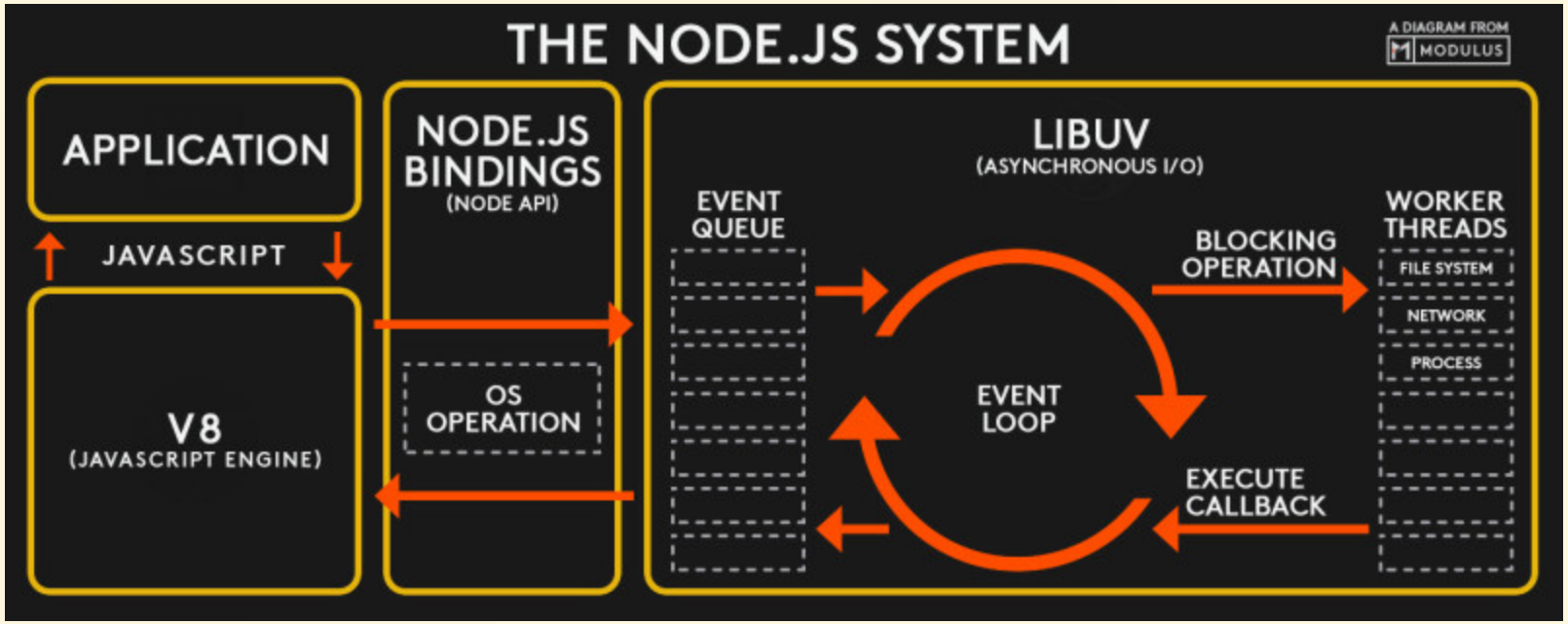
Asynchrone - Illustration - Avant



Asynchrone - Illustration - Async



Asynchrone - Nodejs



Asynchrone - And so what?

- Python supporte nativement le paradigme (PEP 492 - Mai 2015)
- Module `asyncio` natif (> Python 3.5)
- Écosystème favorisant le paradigme asynchrone sur les I/O

```
async with aiofiles.open('filename', mode='r') as f:
    contents = await f.read()

async with aiohttp.ClientSession() as session:
    async with session.get('http://python.org') as response:
        html = await response.text()
```

Asynchrone - Et moi, je gagne quoi?

```
def get(url):
    try:
        start = time.time()
        resp = requests.get(url)
        print(f"{url} ok - {len(resp.text)} bytes - {time.time() - start:.2f} sec")
    except Exception as e:
        print(f"{url} ko - {e}")

def main(urls):
    return [get(url) for url in urls]

start = time.time()
main(websites.split("\n"))  # 100 urls
print(f"Took {time.time() - start} sec")
```

Asynchrone - Et moi, je gagne quoi?

```
$ python get-urls.py
https://www.youtube.com ok - 10555 bytes - 0.20 sec
https://www.facebook.com ok - 194165 bytes - 0.15 sec
https://www.baidu.com ok - 2443 bytes - 1.25 sec
https://www.yahoo.com ok - 510461 bytes - 0.92 sec
https://www.amazon.com ok - 2671 bytes - 0.40 sec
...
http://www.onet.pl ok - 812159 bytes - 0.75 sec
http://www.googleadservices.com ok - 1557 bytes - 0.03 sec
http://www.accuweather.com ok - 268 bytes - 0.06 sec
http://www.googleweblight.com ok - 1615 bytes - 0.11 sec
http://www.answers.yahoo.com ok - 68570 bytes - 2.66 sec
Took 94.78780889511108 sec
```

Asynchrone - Et moi, je gagne quoi?

```
async def get(url):
    try:
        start = time.time()
        async with aiohttp.ClientSession() as session:
            async with session.get(url=url) as response:
                resp = await response.read()
                print(f"{url} ok - {len(resp)} bytes - {time.time() - start:.2f} sec")
    except Exception as e:
        print(f"{url} ko - {e}")

async def main(urls):
    await asyncio.gather(*[get(url) for url in urls])

start = time.time()
asyncio.run(main(websites.split("\n")))  # 100 urls
print(f"Took {time.time() - start} sec")
```

Asynchrone - Et moi, je gagne quoi?

```
$ python async-get-urls.py
https://www.wikipedia.org ok - 73321 bytes - 0.21 sec
https://www.facebook.com ok - 194167 bytes - 0.23 sec
https://www.google.co.in ok - 13669 bytes - 0.26 sec
https://www.google.de ok - 13617 bytes - 0.28 sec
https://www.bing.com ok - 77723 bytes - 0.31 sec
...
http://www.alipay.com ok - 24057 bytes - 3.18 sec
http://www.weibo.com ok - 93937 bytes - 3.63 sec
http://www.coccoc.com ok - 64687 bytes - 3.76 sec
http://www.hao123.com ok - 370371 bytes - 4.56 sec
http://www.youku.com ok - 370831 bytes - 4.73 sec
Took 4.755102157592773 sec
```

Asynchrone - What is next

- Web Dev : privilégier les frameworks / middlewares async (`FastAPI`)
- Data Eng : frameworks de calcul distribué (`distributed`, `dask`)

Aller plus loin (FastAPI)

FastAPI

- Un bon candidat pour détrôner flask
- Levier sur l'asynchrone pour gagner en performances
- Levier sur le typage pour facilier les checks (query params, forms)

Aller plus loin (FastAPI)

```python
from typing import Optional
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}")
def read_item(item_id: int, q: Optional[str] = None):
    return {"item_id": item_id, "q": q}
```

Aller plus loin (FastAPI)

20-queries (bar) | Data table | Latency | Framework overhead

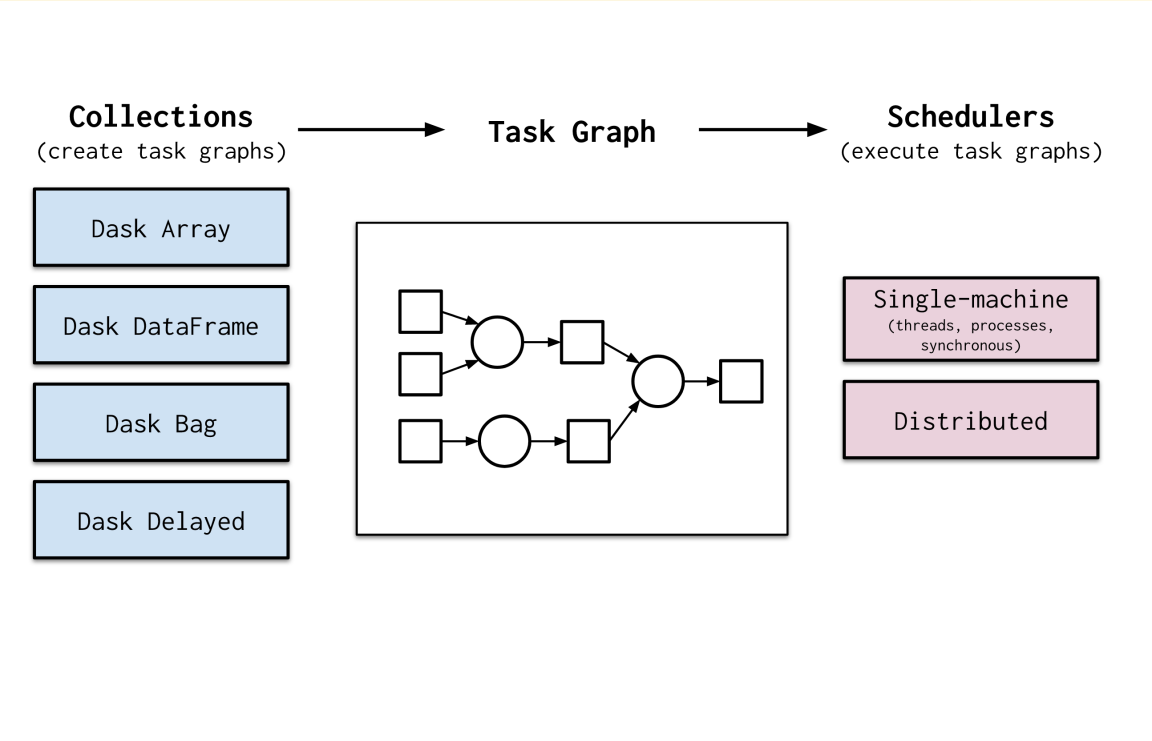
Responses per second at 20 queries per request, Test environment (39 tests)

Rnk	Framework	Performance (higher is better)		Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	fastapi	14,442	100.0% (31.5%)	0	Mcr	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
2	starlette	14,363	99.5% (31.3%)	0	Plt	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
3	uvicorn	14,284	98.9% (31.1%)	0	Plt	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
4	blacksheep	14,159	98.0% (30.9%)	0	Plt	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
5	aiohttp-pg-raw	12,019	83.2% (26.2%)	0	Mcr	Py	asy	Gun	Lin	Pg	Lin	Raw	Rea
6	tornado-py3-uvloop	11,778	81.6% (25.7%)	0	Plt	Py	Non	Tor	Lin	Pg	Lin	Raw	Rea
7	bottle-raw	8,247	57.1% (18.0%)	0	Mcr	Py	Mei	Non	Lin	My	Lin	Raw	Rea
8	api_hour	7,018	48.6% (15.3%)	0	Mcr	Py	asy	Gun	Lin	Pg	Lin	Raw	Rea
9	flask-raw	6,969	48.3% (15.2%)	0	Mcr	Py	Mei	Non	Lin	My	Lin	Raw	Rea
10	flask-pypy2-raw	6,821	47.2% (14.9%)	0	Mcr	Py	Tor	Non	Lin	My	Lin	Raw	Rea
11	morepath	6,208	43.0% (13.5%)	0	Mcr	Py	Mei	Gun	Lin	Pg	Lin	Ful	Rea
12	web2py-optimized	5,825	40.3% (12.7%)	0	Ful	Py	Mei	Non	Lin	My	Lin	Ful	Rea
13	weppy-pypy2	5,403	37.4% (11.8%)	0	Ful	Py	Tor	Non	Lin	Pg	Lin	Ful	Rea
14	api_hour-mysql	4,978	34.5% (10.9%)	0	Mcr	Py	asy	Gun	Lin	My	Lin	Raw	Rea
15	weppy	3,140	21.7% (6.8%)	0	Ful	Py	Mei	Non	Lin	Pg	Lin	Ful	Rea
16	weppy-nginx-uwsgi	3,109	21.5% (6.8%)	0	Ful	Py	uWS	ngx	Lin	Pg	Lin	Ful	Rea
17	weppy-py3	3,107	21.5% (6.8%)	0	Ful	Py	Mei	Non	Lin	Pg	Lin	Ful	Rea
18	web2py	2,338	16.2% (5.1%)	0	Ful	Py	Mei	Non	Lin	My	Lin	Ful	Rea
19	aiohttp	2,293	15.9% (5.0%)	0	Mcr	Py	asy	Gun	Lin	Pg	Lin	Ful	Rea
20	tornado-pypy2	2,192	15.2% (4.8%)	0	Plt	Py	Non	Tor	Lin	Mo	Lin	Raw	Rea
21	flask-nginx-uwsgi	1,618	11.2% (3.5%)	0	Mcr	Py	Non	ngx	Lin	My	Lin	Ful	Rea
22	django-postgresql	1,607	11.1% (3.5%)	0	Ful	Py	Non	Mei	Lin	Pg	Lin	Ful	Rea
23	bottle-pypy2	1,581	10.9% (3.4%)	0	Mcr	Py	Tor	Non	Lin	My	Lin	Ful	Rea
24	flask	1,576	10.9% (3.4%)	0	Mcr	Py	Mei	Non	Lin	My	Lin	Ful	Rea
25	django-py3	1,570	10.9% (3.4%)	0	Ful	Py	Non	Mei	Lin	My	Lin	Ful	Rea
26	django	1,459	10.1% (3.2%)	0	Ful	Py	Non	Mei	Lin	My	Lin	Ful	Rea

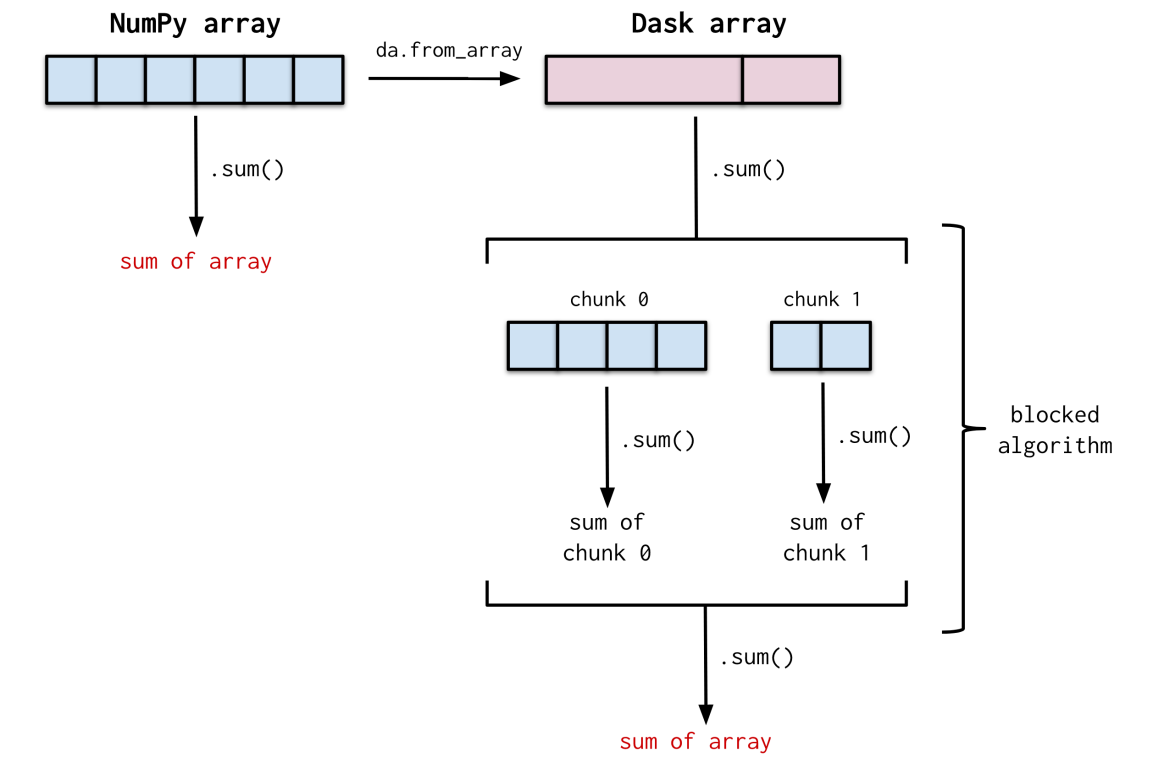
Aller plus loin (Dask)

- Distribue nos bons vieux Pandas DF (et plus)
- Très bien architecturé et code accessible (pure Python - No JVM)
- Surcouche et reuse de l'existant (Numpy / Pandas / Scikit)
- C'est le nouveau standard pour scaler les librairies Data
- Async : exécution de jobs data sur `distributed`

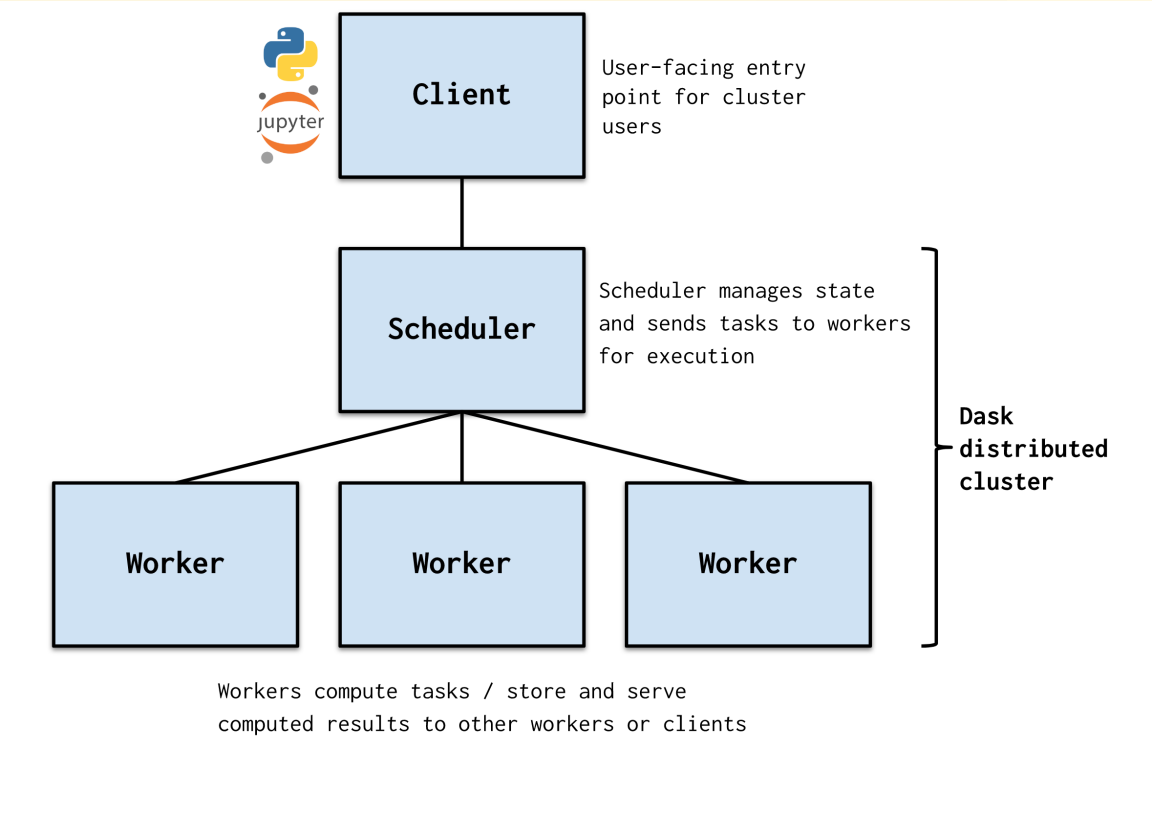
Aller plus loin (Dask)



Aller plus loin (Dask)



Aller plus loin (Dask)



Questions?