Mars: a tensor-based unified framework for large-scale data computation

Xuye (Chris) Qin (Alibaba)

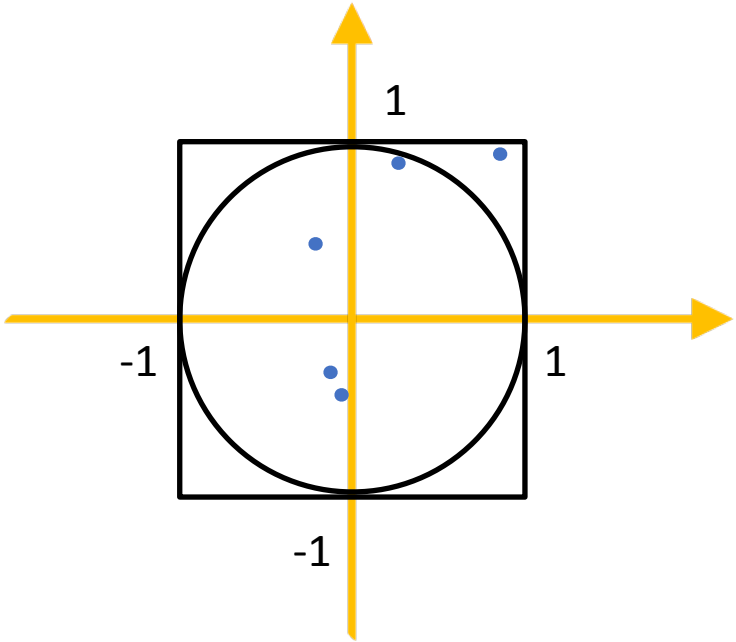
Профессиональная конференция для Python-разработчиков

About me

- Senior software engineer at Alibaba
- Working on combining big data and Python
- Recently focusing on Mars as core developer and architect
- Developed a framework named PyODPS supports pandas-like DataFrame which can run on big data platform MaxCompute and Databases like Postgresql and MySQL

What’s Mars? Let’s start from an example

Estimating Pi with Monte Carlo methods



x	y
-0.186633	0.445155
0.286809	-0.923266
0.891767	0.979931
-0.183977	-0.277083
-0.098763	-0.316315
...	...

$$p = \frac{inside}{N} = \frac{\pi r^2}{4r^2}$$

$$\pi = \frac{inside}{N} * 4$$

Implementing with pure python

```python
import random
from math import sqrt

N = 10 ** 7

inside = 0
for _ in range(N):
    x = random.uniform(-1, 1)
    y = random.uniform(-1, 1)
    if sqrt(x ** 2 + y ** 2) < 1:
        inside += 1

pi = 4 * inside / N
print('pi: %.5f' % pi)
```


Moscow Python Conf++ 2019

Implementing with pure python

```python
import random
from math import sqrt

N = 10 ** 7

inside = 0
for _ in range(N):
    x = random.uniform(-1, 1)
    y = random.uniform(-1, 1)
    if sqrt(x ** 2 + y ** 2) < 1:
        inside += 1

pi = 4 * inside / N
print('pi: %.5f' % pi)
```

```
pi: 3.14266
CPU times: user 10.7 s, sys: 15.4 ms, total: 10.7 s
Wall time: 10.7 s
```


Moscow Python Conf++ 2019

Accelerate with cython

```
1  import random
2  from libc.math cimport sqrt
3
4  cdef size_t N = 10 ** 7
5  cdef size_t inside = 0
6  cdef float x, y, pi
7
8  for _ in range(N):
9      x = random.uniform(-1, 1)
10     y = random.uniform(-1, 1)
11     if sqrt(x ** 2 + y ** 2) < 1:
12         inside += 1
13
14 pi = 4 * inside / N
15 print('pi: %.5f' % pi)
```


Moscow Python Conf++ 2019

Accelerate with cython

```
1  import random
2  from libc.math cimport sqrt
3
4  cdef size_t N = 10 ** 7
5  cdef size_t inside = 0
6  cdef float x, y, pi
7
8  for _ in range(N):
9      x = random.uniform(-1, 1)
10     y = random.uniform(-1, 1)
11     if sqrt(x ** 2 + y ** 2) < 1:
12         inside += 1
13
14 pi = 4 * inside / N
15 print('pi: %.5f' % pi)
```

Type declaration

Moscow Python Conf++ 2019

Accelerate with cython

```
1  import random
2  from libc.math cimport sqrt
3
4  cdef size_t N = 10 ** 7
5  cdef size_t inside = 0
6  cdef float x, y, pi
7
8  for _ in range(N):
9      x = random.uniform(-1, 1)
10     y = random.uniform(-1, 1)
11     if sqrt(x ** 2 + y ** 2) < 1:
12         inside += 1
13
14 pi = 4 * inside / N
15 print('pi: %.5f' % pi)
```

Type declaration

```
pi: 3.14214
CPU times: user 5.76 s, sys: 15 ms, total: 5.78 s
Wall time: 5.8 s
```


Moscow Python Conf++ 2019

Accelerate with cython

```
1  import random
2  from libc.math cimport sqrt
3
4  cdef size_t N = 10 ** 7
5  cdef size_t inside = 0
6  cdef float x, y, pi
7
8  for _ in range(N):
9      x = random.uniform(-1, 1)
10     y = random.uniform(-1, 1)
11     if sqrt(x ** 2 + y ** 2) < 1:
12         inside += 1
13
14 pi = 4 * inside / N
15 print('pi: %.5f' % pi)
```

Type declaration

```
pi: 3.14214
CPU times: user 5.76 s, sys: 15 ms, total: 5.78 s
Wall time: 5.8 s
```

~45% faster

Moscow Python Conf++ 2019

Implement with numpy

```
1 import numpy as np
2
3 N = 10 ** 7
4
5 data = np.random.uniform(-1, 1, size=(N, 2))
6 inside = (np.sqrt((data ** 2).sum(axis=1)) < 1).sum()
7 pi = 4 * inside / N
8 print('pi: %.5f' % pi)
```


Moscow Python Conf++ 2019

Implement with numpy

```
1 import numpy as np
2
3 N = 10 ** 7
4
5 data = np.random.uniform(-1, 1, size=(N, 2))
6 inside = (np.sqrt((data ** 2).sum(axis=1)) < 1).sum()
7 pi = 4 * inside / N
8 print('pi: %.5f' % pi)
```

```
pi: 3.14113
CPU times: user 582 ms, sys: 166 ms, total: 748 ms
Wall time: 757 ms
```


Moscow Python Conf++ 2019

Implement with numpy

```
1 import numpy as np
2
3 N = 10 ** 7
4
5 data = np.random.uniform(-1, 1, size=(N, 2))
6 inside = (np.sqrt((data ** 2).sum(axis=1)) < 1).sum()
7 pi = 4 * inside / N
8 print('pi: %.5f' % pi)
```

```
pi: 3.14113
CPU times: user 582 ms, sys: 166 ms, total: 748 ms
Wall time: 757 ms
```

Over 10x faster

Moscow Python Conf++ 2019

Accelerate numpy with numexpr

```
1  import numpy as np
2  import numexpr as ne
3
4  N = 10 ** 7
5
6  x = np.random.uniform(-1, 1, N)
7  y = np.random.uniform(-1, 1, N)
8  inside = ne.evaluate(
9      'sum(where(sqrt(x ** 2 + y ** 2) < 1, 1, 0))')
10 pi = 4 * inside / N
11 print('pi: %.5f' % pi)
```


Moscow Python Conf++ 2019

Accelerate numpy with numexpr

```
1  import numpy as np
2  import numexpr as ne
3
4  N = 10 ** 7
5
6  x = np.random.uniform(-1, 1, N)
7  y = np.random.uniform(-1, 1, N)
8  inside = ne.evaluate(
9      'sum(where(sqrt(x ** 2 + y ** 2) < 1, 1, 0))')
10 pi = 4 * inside / N
11 print('pi: %.5f' % pi)
```

Fuse operations

Moscow Python Conf++ 2019

Accelerate numpy with numexpr

```
1  import numpy as np
2  import numexpr as ne
3
4  N = 10 ** 7
5
6  x = np.random.uniform(-1, 1, N)
7  y = np.random.uniform(-1, 1, N)
8  inside = ne.evaluate(
9      'sum(where(sqrt(x ** 2 + y ** 2) < 1, 1, 0))')
10 pi = 4 * inside / N
11 print('pi: %.5f' % pi)
```

Fuse operations

```
pi: 3.14124
CPU times: user 337 ms, sys: 51.6 ms, total: 388 ms
Wall time: 396 ms
```


Moscow Python Conf++ 2019

Accelerate numpy with numexpr

```
1  import numpy as np
2  import numexpr as ne
3
4  N = 10 ** 7
5
6  x = np.random.uniform(-1, 1, N)
7  y = np.random.uniform(-1, 1, N)
8  inside = ne.evaluate(
9      'sum(where(sqrt(x ** 2 + y ** 2) < 1, 1, 0))')
10 pi = 4 * inside / N
11 print('pi: %.5f' % pi)
```

Fuse operations

```
pi: 3.14124
CPU times: user 337 ms, sys: 51.6 ms, total: 388 ms
Wall time: 396 ms
```

Over 25x faster

Moscow Python Conf++ 2019

Use cupy to leverage GPU

```
1 import cupy as cp
2
3 N = 10 ** 7
4
5 data = cp.random.uniform(-1, 1, size=(N, 2))
6 inside = (cp.sqrt((data ** 2).sum(axis=1)) < 1).sum()
7 pi = 4 * inside / N
8 print('pi: %.5f' % pi)
```


Moscow Python Conf++ 2019

Use cupy to leverage GPU

```
1 import cupy as cp
2
3 N = 10 ** 7
4
5 data = cp.random.uniform(-1, 1, size=(N, 2))
6 inside = (cp.sqrt((data ** 2).sum(axis=1)) < 1).sum()
7 pi = 4 * inside / N
8 print('pi: %.5f' % pi)
```

Import replacement

Moscow Python Conf++ 2019

Use cupy to leverage GPU

```
1 import cupy as cp
2
3 N = 10 ** 7
4
5 data = cp.random.uniform(-1, 1, size=(N, 2))
6 inside = (cp.sqrt((data ** 2).sum(axis=1)) < 1).sum()
7 pi = 4 * inside / N
8 print('pi: %.5f' % pi)
```

Import replacement

```
pi: 3.14207
CPU times: user 20 ms, sys: 10.1 ms, total: 30.1 ms
Wall time: 36 ms
```


Moscow Python Conf++ 2019

Use cupy to leverage GPU

```
1 import cupy as cp
2
3 N = 10 ** 7
4
5 data = cp.random.uniform(-1, 1, size=(N, 2))
6 inside = (cp.sqrt((data ** 2).sum(axis=1)) < 1).sum()
7 pi = 4 * inside / N
8 print('pi: %.5f' % pi)
```

Import replacement

```
pi: 3.14207
CPU times: user 20 ms, sys: 10.1 ms, total: 30.1 ms
Wall time: 36 ms
```

Over 270x faster

Moscow Python Conf++ 2019

Let’s scale up the data to 1000x larger

```python
import numpy as np

N = 10 ** 10

data = np.random.uniform(-1, 1, size=(N, 2))
inside = (np.sqrt((data ** 2).sum(axis=1)) < 1).sum()
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

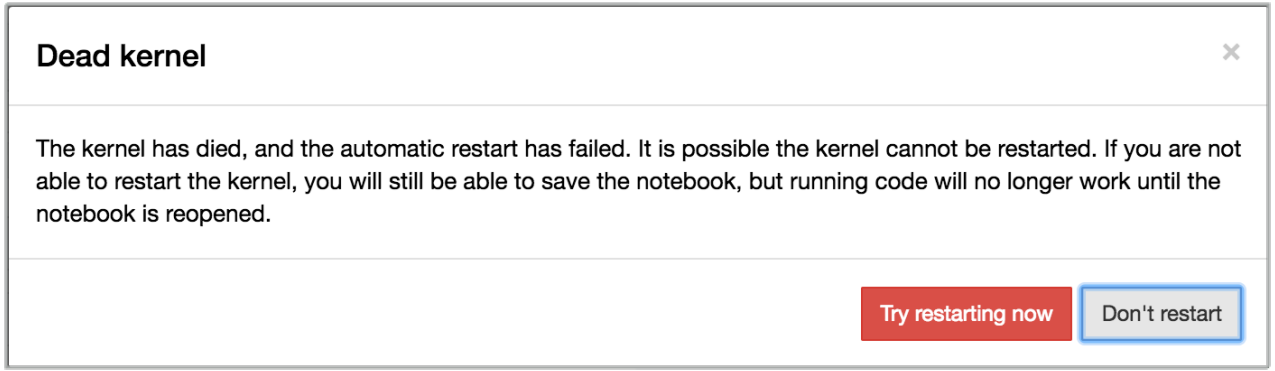

Moscow Python Conf++ 2019

```python
import numpy as np

N = 10 ** 10

data = np.random.uniform(-1, 1, size=(N, 2))
inside = (np.sqrt((data ** 2).sum(axis=1)) < 1).sum()
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-1-28227489136f> in <module>()
      3 N = 10 ** 10
      4
----> 5 data = np.random.uniform(-1, 1, size=(N, 2))
      6 inside = (np.sqrt((data ** 2).sum(axis=1)) < 1).sum()
      7 pi = 4 * inside / N

mtrand.pyx in mtrand.RandomState.uniform (numpy/random/mtrand/mtrand.c:19349)()

mtrand.pyx in mtrand.cont2_array_sc (numpy/random/mtrand/mtrand.c:7415)()

MemoryError:
```


Moscow Python Conf++ 2019

What’s more…



Moscow Python Conf++ 2019

What’s more…

Dead kernel

The kernel has died, and the automatic restart has failed. It is possible the kernel cannot be restarted. If you are not able to restart the kernel, you will still be able to save the notebook, but running code will no longer work until the notebook is reopened.

Try restarting now | Don't restart

Moscow Python Conf++ 2019

Split data into chunks

```
1  import numpy as np
2
3  N = 10 ** 10
4  CHUNK_SIZE = 10 ** 7
5
6  inside = 0
7  for _ in range(N // CHUNK_SIZE):
8      data = np.random.uniform(-1, 1, size=(CHUNK_SIZE, 2))
9      inside += (np.linalg.norm(data, axis=1) < 1).sum()
10 pi = 4 * inside / N
11 print('pi: %.5f' % pi)
```


Moscow Python Conf++ 2019

Split data into chunks

Iterate 1000 times

```
1  import numpy as np
2
3  N = 10 ** 10
4  CHUNK_SIZE = 10 ** 7
5
6  inside = 0
7  for   in range(N // CHUNK_SIZE):
8      data = np.random.uniform(-1, 1, size=(CHUNK_SIZE, 2))
9      inside += (np.linalg.norm(data, axis=1) < 1).sum()
10 pi = 4 * inside / N
11 print('pi: %.5f' % pi)
```


Moscow Python Conf++ 2019

Split data into chunks

Iterate 1000 times

```
1  import numpy as np
2
3  N = 10 ** 10
4  CHUNK_SIZE = 10 ** 7
5
6  inside = 0
7  for   in range(N // CHUNK_SIZE):
8      data = np.random.uniform(-1, 1, size=(CHUNK_SIZE, 2))
9      inside += (np.linalg.norm(data, axis=1) < 1).sum()
10 pi = 4 * inside / N
11 print('pi: %.5f' % pi)
```

```
pi: 3.14160
CPU times: user 9min 28s, sys: 2min 44s, total: 12min 13s
Wall time: 12min 15s
```


Moscow Python Conf++ 2019

Split data into chunks

Iterate 1000 times

```
1  import numpy as np
2
3  N = 10 ** 10
4  CHUNK_SIZE = 10 ** 7
5
6  inside = 0
7  for   in range(N // CHUNK_SIZE):
8      data = np.random.uniform(-1, 1, size=(CHUNK_SIZE, 2))
9      inside += (np.linalg.norm(data, axis=1) < 1).sum()
10 pi = 4 * inside / N
11 print('pi: %.5f' % pi)
```

```
pi: 3.14160
CPU times: user 9min 28s, sys: 2min 44s, total: 12min 13s
Wall time: 12min 15s
```

Leverage only 1 CPU core

Moscow Python Conf++ 2019

The problem

Moscow Python Conf++ 2019

The problem

Moscow Python Conf++ 2019

The problem

- Only small subset of Numpy operations can natively be parallel, e.g. tensordot
- Consider multithreading/ multiprocessing
- How about distributed?

Moscow Python Conf++ 2019

Multithreading vs. Multiprocessing

```
1  import concurrent.futures
2  import numpy as np
3
4  N = 10 ** 10
5  CHUNK_SIZE = 10 ** 7
6
7
8  def chunk_inside():
9      data = np.random.uniform(-1, 1,
10                               size=(CHUNK_SIZE, 2))
11     return (np.linalg.norm(data, axis=1) < 1).sum()
12
13
14 pool = concurrent.futures.ThreadPoolExecutor(4)
15 futures = [pool.submit(chunk_inside) for _ in
16            range(N // CHUNK_SIZE)]
17 inside = sum(future.result() for future in futures)
18 pi = 4 * inside / N
19 print('pi: %.5f' % pi)
```


Moscow Python Conf++ 2019

Multithreading vs. Multiprocessing

```python
import concurrent.futures
import numpy as np

N = 10 ** 10
CHUNK_SIZE = 10 ** 7


def chunk_inside():
    data = np.random.uniform(-1, 1,
                             size=(CHUNK_SIZE, 2))
    return (np.linalg.norm(data, axis=1) < 1).sum()


pool = concurrent.futures.ThreadPoolExecutor(4)
futures = [pool.submit(chunk_inside) for _ in
           range(N // CHUNK_SIZE)]
inside = sum(future.result() for future in futures)
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

```
pi: 3.14159
CPU times: user 9min 37s, sys: 2min 49s, total: 12min 26s
Wall time: 6min 26s
```


Moscow Python Conf++ 2019

Multithreading vs. Multiprocessing

```python
import concurrent.futures
import numpy as np

N = 10 ** 10
CHUNK_SIZE = 10 ** 7


def chunk_inside():
    data = np.random.uniform(-1, 1,
                             size=(CHUNK_SIZE, 2))
    return (np.linalg.norm(data, axis=1) < 1).sum()


pool = concurrent.futures.ThreadPoolExecutor(4)
futures = [pool.submit(chunk_inside) for _ in
           range(N // CHUNK_SIZE)]
inside = sum(future.result() for future in futures)
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

```
pi: 3.14159
CPU times: user 9min 37s, sys: 2min 49s, total: 12min 26s
Wall time: 6min 26s
```

Up to ~2x

Moscow Python Conf++ 2019

Multithreading vs. Multiprocessing

```python
import concurrent.futures
import numpy as np

N = 10 ** 10
CHUNK_SIZE = 10 ** 7


def chunk_inside():
    data = np.random.uniform(-1, 1,
                             size=(CHUNK_SIZE, 2))
    return (np.linalg.norm(data, axis=1) < 1).sum()


pool = concurrent.futures.ThreadPoolExecutor(4)
futures = [pool.submit(chunk_inside) for _ in
           range(N // CHUNK_SIZE)]
inside = sum(future.result() for future in futures)
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

```python
import concurrent.futures
import numpy as np

N = 10 ** 10
CHUNK_SIZE = 10 ** 7


def chunk_inside():
    data = np.random.uniform(-1, 1,
                             size=(CHUNK_SIZE, 2))
    return (np.linalg.norm(data, axis=1) < 1).sum()


pool = concurrent.futures.ProcessPoolExecutor(4)
futures = [pool.submit(chunk_inside) for _ in
           range(N // CHUNK_SIZE)]
inside = sum(future.result() for future in futures)
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

```
pi: 3.14159
CPU times: user 9min 37s, sys: 2min 49s, total: 12min 26s
Wall time: 6min 26s
```

Up to ~2x

Moscow Python Conf++ 2019

Multithreading vs. Multiprocessing

```python
import concurrent.futures
import numpy as np

N = 10 ** 10
CHUNK_SIZE = 10 ** 7


def chunk_inside():
    data = np.random.uniform(-1, 1,
                             size=(CHUNK_SIZE, 2))
    return (np.linalg.norm(data, axis=1) < 1).sum()


pool = concurrent.futures.ThreadPoolExecutor(4)
futures = [pool.submit(chunk_inside) for _ in
           range(N // CHUNK_SIZE)]
inside = sum(future.result() for future in futures)
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

```python
import concurrent.futures
import numpy as np

N = 10 ** 10
CHUNK_SIZE = 10 ** 7


def chunk_inside():
    data = np.random.uniform(-1, 1,
                             size=(CHUNK_SIZE, 2))
    return (np.linalg.norm(data, axis=1) < 1).sum()


pool = concurrent.futures.ProcessPoolExecutor(4)
futures = [pool.submit(chunk_inside) for _ in
           range(N // CHUNK_SIZE)]
inside = sum(future.result() for future in futures)
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

```
pi: 3.14159
CPU times: user 9min 37s, sys: 2min 49s, total: 12min 26s
Wall time: 6min 26s
```

Up to ~2x

Moscow Python Conf++ 2019

Multithreading vs. Multiprocessing

```python
import concurrent.futures
import numpy as np

N = 10 ** 10
CHUNK_SIZE = 10 ** 7


def chunk_inside():
    data = np.random.uniform(-1, 1,
                             size=(CHUNK_SIZE, 2))
    return (np.linalg.norm(data, axis=1) < 1).sum()


pool = concurrent.futures.ThreadPoolExecutor(4)
futures = [pool.submit(chunk_inside) for _ in
           range(N // CHUNK_SIZE)]
inside = sum(future.result() for future in futures)
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

```
pi: 3.14159
CPU times: user 9min 37s, sys: 2min 49s, total: 12min 26s
Wall time: 6min 26s
```

Up to ~2x

```python
import concurrent.futures
import numpy as np

N = 10 ** 10
CHUNK_SIZE = 10 ** 7


def chunk_inside():
    data = np.random.uniform(-1, 1,
                             size=(CHUNK_SIZE, 2))
    return (np.linalg.norm(data, axis=1) < 1).sum()


pool = concurrent.futures.ProcessPoolExecutor(4)
futures = [pool.submit(chunk_inside) for _ in
           range(N // CHUNK_SIZE)]
inside = sum(future.result() for future in futures)
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

```
pi: 3.14158
CPU times: user 929 ms, sys: 378 ms, total: 1.31 s
Wall time: 3min 38s
```


Moscow Python Conf++ 2019

Multithreading vs. Multiprocessing

```python
import concurrent.futures
import numpy as np

N = 10 ** 10
CHUNK_SIZE = 10 ** 7


def chunk_inside():
    data = np.random.uniform(-1, 1,
                             size=(CHUNK_SIZE, 2))
    return (np.linalg.norm(data, axis=1) < 1).sum()


pool = concurrent.futures.ThreadPoolExecutor(4)
futures = [pool.submit(chunk_inside) for _ in
           range(N // CHUNK_SIZE)]
inside = sum(future.result() for future in futures)
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

```
pi: 3.14159
CPU times: user 9min 37s, sys: 2min 49s, total: 12min 26s
Wall time: 6min 26s
```

Up to ~2x

```python
import concurrent.futures
import numpy as np

N = 10 ** 10
CHUNK_SIZE = 10 ** 7


def chunk_inside():
    data = np.random.uniform(-1, 1,
                             size=(CHUNK_SIZE, 2))
    return (np.linalg.norm(data, axis=1) < 1).sum()


pool = concurrent.futures.ProcessPoolExecutor(4)
futures = [pool.submit(chunk_inside) for _ in
           range(N // CHUNK_SIZE)]
inside = sum(future.result() for future in futures)
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

```
pi: 3.14158
CPU times: user 929 ms, sys: 378 ms, total: 1.31 s
Wall time: 3min 38s
```

Up to ~3x

Moscow Python Conf++ 2019

Let’s consider the more complex computation

```
1 import numpy as np
2
3 a = np.random.rand(100000, 100000)
4 (a.dot(a.T) - a).std()
```


Moscow Python Conf++ 2019

Let’s consider the more complex computation

```
1 import numpy as np
2
3 a = np.random.rand(100000, 100000)
4 (a.dot(a.T) - a).std()
```

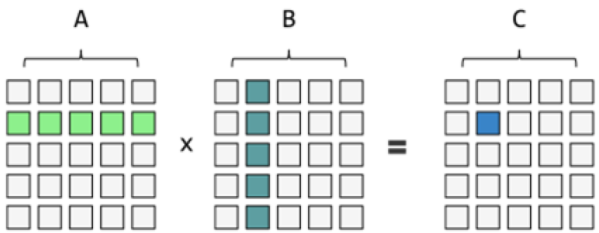
~75G

Moscow Python Conf++ 2019

Let’s consider the more complex computation

```
1 import numpy as np
2
3 a = np.random.rand(100000, 100000)
4 (a.dot(a.T) - a).std()
```

~75G



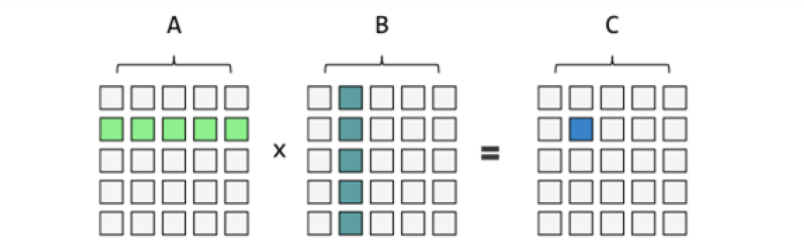
Moscow Python Conf++ 2019

Let’s consider the more complex computation

```
import numpy as np

a = np.random.rand(100000, 100000)
(a.dot(a.T) - a).std()
```

~75G

Hard to implement parallel version manually



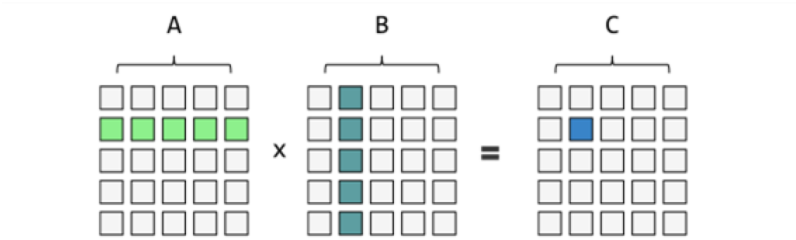
Moscow Python Conf++ 2019

Let’s consider the more complex computation

```
import numpy as np

a = np.random.rand(100000, 100000)
(a.dot(a.T) - a).std()
```

~75G

Hard to implement parallel version manually



- Estimating Pi with Monte Carlo is easy to reach parallel
- Many tasks are hard to implement parallel version manually
- Even harder to make the execution distributed

What framework do we need?

- Familiar interface, import replace
- Scalability
  - Scale in: leverage multi-cores
  - Scale out: execution on thousands of workers
- GPU acceleration
- Fuse operands as many as possible
- In-memory computing, support for spilling to external storage like disk

Moscow Python Conf++ 2019

Mars tensor

- Numpy compatible API to process large tensor/ndarray
- Cover 70% of commonly used interfaces of Numpy
- Schedulers
  - Threaded
  - Local cluster
  - Distributed

Moscow Python Conf++ 2019

Mars tensor interfaces

- Tensor creation
  - ones
  - empty
  - zeros
  - ones_like
  - …
- Random sampling
  - rand
  - randint
  - beta
  - binomial
  - …
- Basic manipulations
  - astype
  - transpose
  - broadcast_to
  - …
- Aggregation
  - sum
  - nansum
  - max
  - all
  - mean
  - …
- Indexing
  - Slice
  - Boolean indexing
  - Fancy indexing (partial support)
  - newaxis
  - Ellipsis
- Discrete Fourier transform
- Linear Algebra
  - QR
  - SVD
  - Cholesky
  - inv
  - norm
  - …

Moscow Python Conf++ 2019

Smooth transition from Numpy to Mars tensor

```
import numpy as np

N = 10 ** 10

data = np.random.uniform(-1, 1, size=(N, 2))
inside = (np.linalg.norm(data, axis=1) < 1).sum()
pi = 4 * inside / N
print('pi: %.5f' % pi)
```


Moscow Python Conf++ 2019

Smooth transition from Numpy to Mars tensor

```
import numpy as np

N = 10 ** 10

data = np.random.uniform(-1, 1, size=(N, 2))
inside = (np.linalg.norm(data, axis=1) < 1).sum()
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

~150G, may cause out of memory

Moscow Python Conf++ 2019

Smooth transition from Numpy to Mars tensor

```python
import numpy as np

N = 10 ** 10

data = np.random.uniform(-1, 1, size=(N, 2))
inside = (np.linalg.norm(data, axis=1) < 1).sum()
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

~150G, may cause out of memory

```python
import mars.tensor as mt

N = 10 ** 10

data = mt.random.uniform(-1, 1, size=(N, 2))
inside = (mt.linalg.norm(data, axis=1) < 1).sum()
pi = (4 * inside / N).execute()
print('pi: %.5f' % pi)
```


Moscow Python Conf++ 2019

Smooth transition from Numpy to Mars tensor

```python
import numpy as np

N = 10 ** 10

data = np.random.uniform(-1, 1, size=(N, 2))
inside = (np.linalg.norm(data, axis=1) < 1).sum()
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

~150G, may cause out of memory

```python
import mars.tensor as mt

N = 10 ** 10

data = mt.random.uniform(-1, 1, size=(N, 2))
inside = (mt.linalg.norm(data, axis=1) < 1).sum()
pi = (4 * inside / N).execute()
print('pi: %.5f' % pi)
```

Import replacement

Moscow Python Conf++ 2019

Smooth transition from Numpy to Mars tensor

```python
import numpy as np

N = 10 ** 10

data = np.random.uniform(-1, 1, size=(N, 2))
inside = (np.linalg.norm(data, axis=1) < 1).sum()
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

~150G, may cause out of memory

```python
import mars.tensor as mt

N = 10 ** 10

data = mt.random.uniform(-1, 1, size=(N, 2))
inside = (mt.linalg.norm(data, axis=1) < 1).sum()
pi = (4 * inside / N).execute()
print('pi: %.5f' % pi)
```

Import replacement

Trigger execution

Moscow Python Conf++ 2019

Smooth transition from Numpy to Mars tensor

```
1 import numpy as np
2
3 N = 10 ** 10
4
5 data = np.random.uniform(-1, 1, size=(N, 2))
6 inside = (np.linalg.norm(data, axis=1) < 1).sum()
7 pi = 4 * inside / N
8 print('pi: %.5f' % pi)
```

~150G, may cause out of memory

```
1 import mars.tensor as mt
2
3 N = 10 ** 10
4
5 data = mt.random.uniform(-1, 1, size=(N, 2))
6 inside = (mt.linalg.norm(data, axis=1) < 1).sum()
7 pi = (4 * inside / N).execute()
8 print('pi: %.5f' % pi)
```

Import replacement

Trigger execution

```
pi: 3.14157
CPU times: user 10min 39s, sys: 3min 12s, total: 13min 52s
Wall time: 3min 44s
used -61.2852 MiB RAM in 224.23s, peaked 1218.23 MiB above
current, total RAM usage 523.22 MiB
```


Moscow Python Conf++ 2019

Smooth transition from Numpy to Mars tensor

```python
import numpy as np

N = 10 ** 10

data = np.random.uniform(-1, 1, size=(N, 2))
inside = (np.linalg.norm(data, axis=1) < 1).sum()
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

~150G, may cause out of memory

```python
import mars.tensor as mt

N = 10 ** 10

data = mt.random.uniform(-1, 1, size=(N, 2))
inside = (mt.linalg.norm(data, axis=1) < 1).sum()
pi = (4 * inside / N).execute()
print('pi: %.5f' % pi)
```

Import replacement

Trigger execution

```
pi: 3.14157
CPU times: user 10min 39s, sys: 3min 12s, total: 13min 52s
Wall time: 3min 44s
used -61.2852 MiB RAM in 224.23s, peaked 1218.23 MiB above
current, total RAM usage 523.22 MiB
```

~1G memory usage

Moscow Python Conf++ 2019

Smooth transition from Numpy to Mars tensor

```python
import numpy as np

N = 10 ** 10

data = np.random.uniform(-1, 1, size=(N, 2))
inside = (np.linalg.norm(data, axis=1) < 1).sum()
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

~150G, may cause out of memory

```python
import mars.tensor as mt

N = 10 ** 10

data = mt.random.uniform(-1, 1, size=(N, 2))
inside = (mt.linalg.norm(data, axis=1) < 1).sum()
pi = (4 * inside / N).execute()
print('pi: %.5f' % pi)
```

Import replacement

Trigger execution

Close performance to handwritten multiprocessing version

```
pi: 3.14157
CPU times: user 10min 39s, sys: 3min 12s, total: 13min 52s
Wall time: 3min 44s
used -61.2852 MiB RAM in 224.23s, peaked 1218.23 MiB above
current, total RAM usage 523.22 MiB
```

~1G memory usage

Moscow Python Conf++ 2019

Graph mode vs. Eager mode

- Graph mode is default mode, born for performance
- Eager mode is more friendly for developing and debugging

```
>>> from mars.config import options
>>> options.eager_mode = True
>>> import mars.tensor as mt
>>>
>>> a = mt.arange(10)
>>> a
Tensor <op=TensorArange, shape=(10,), key=00f46848d449879e7786fadf909a7490, data=
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])>
>>> a.sum()
Tensor <op=TensorSum, shape=(), key=309966f894a74b273447a81dd7730330, data=
45>
```


Moscow Python Conf++ 2019

GPU and sparse support

```
import mars.tensor as mt

a = mt.random.rand(1000, 2000, gpu=True)
(a + 1).sum(axis=1)
```

GPU

```
import mars.tensor as mt

a = mt.eye(1000, sparse=True, gpu=True)
(a + 1).sum(axis=1)
```

Spars

Moscow Python Conf++ 2019

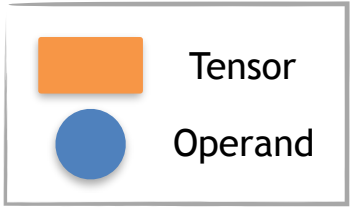
How Mars make things parallel

- Computation graph (DAG)
- Tile: turn a coarse-grained(tensor-level) graph into a fine-grained(chunk-level) graph
- Scheduling

Moscow Python Conf++ 2019

Tensor expressions and coarse-grained graph

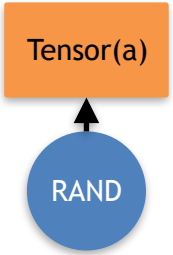
```
import mars.tensor as mt

a = mt.random.rand(1000, 2000)
b = (a + 1).sum(axis=1)
```

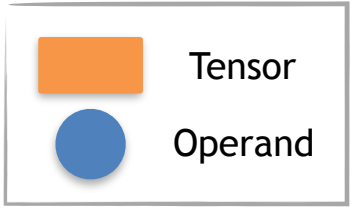


Moscow Python Conf++ 2019

Tensor expressions and coarse-grained graph

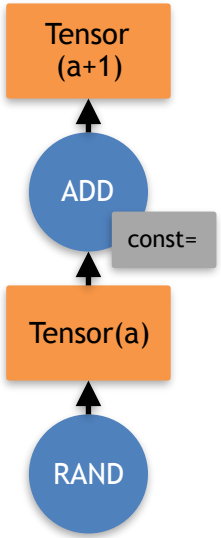
```
import mars.tensor as mt

a = mt.random.rand(1000, 2000)
b = (a + 1).sum(axis=1)
```

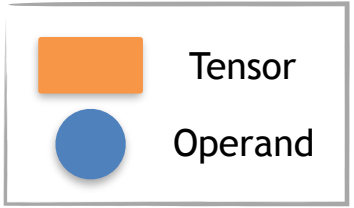




Tensor expressions and coarse-grained graph

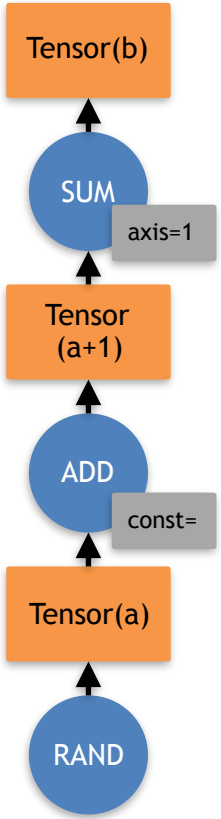
```
import mars.tensor as mt

a = mt.random.rand(1000, 2000)
b = (a + 1).sum(axis=1)
```

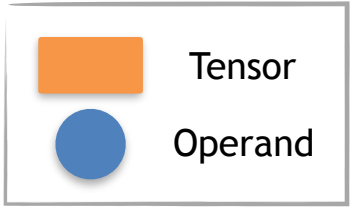




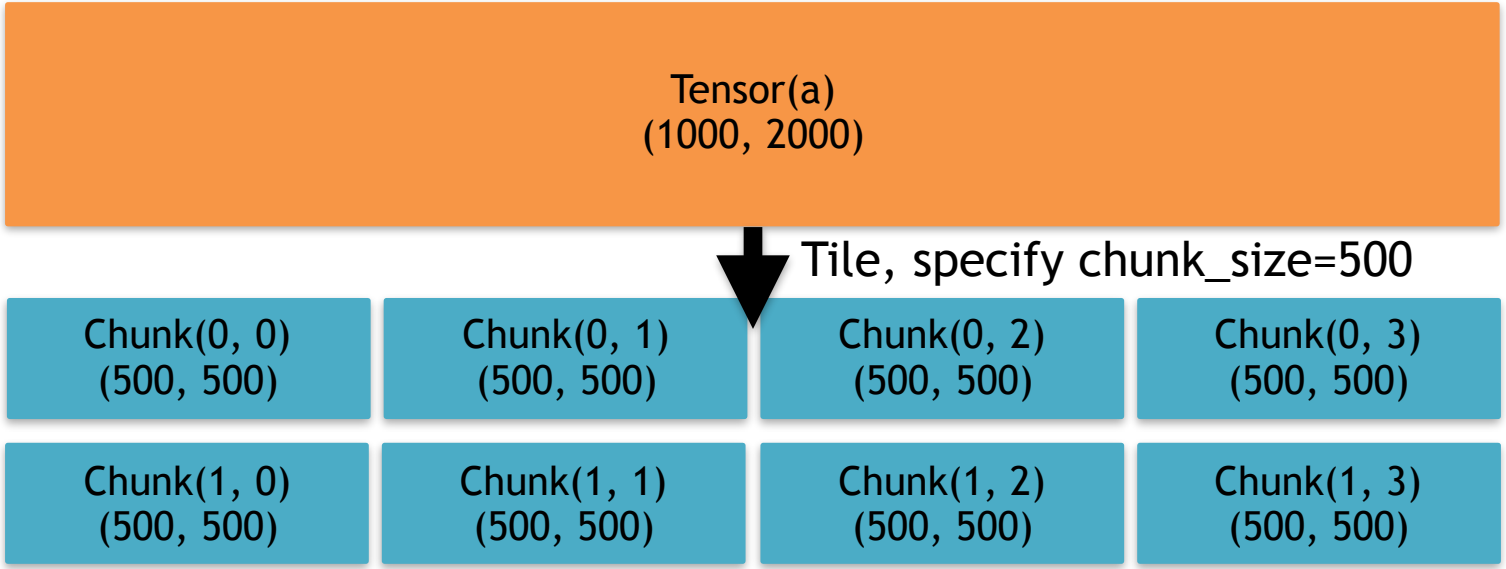
Tensor expressions and coarse-grained graph

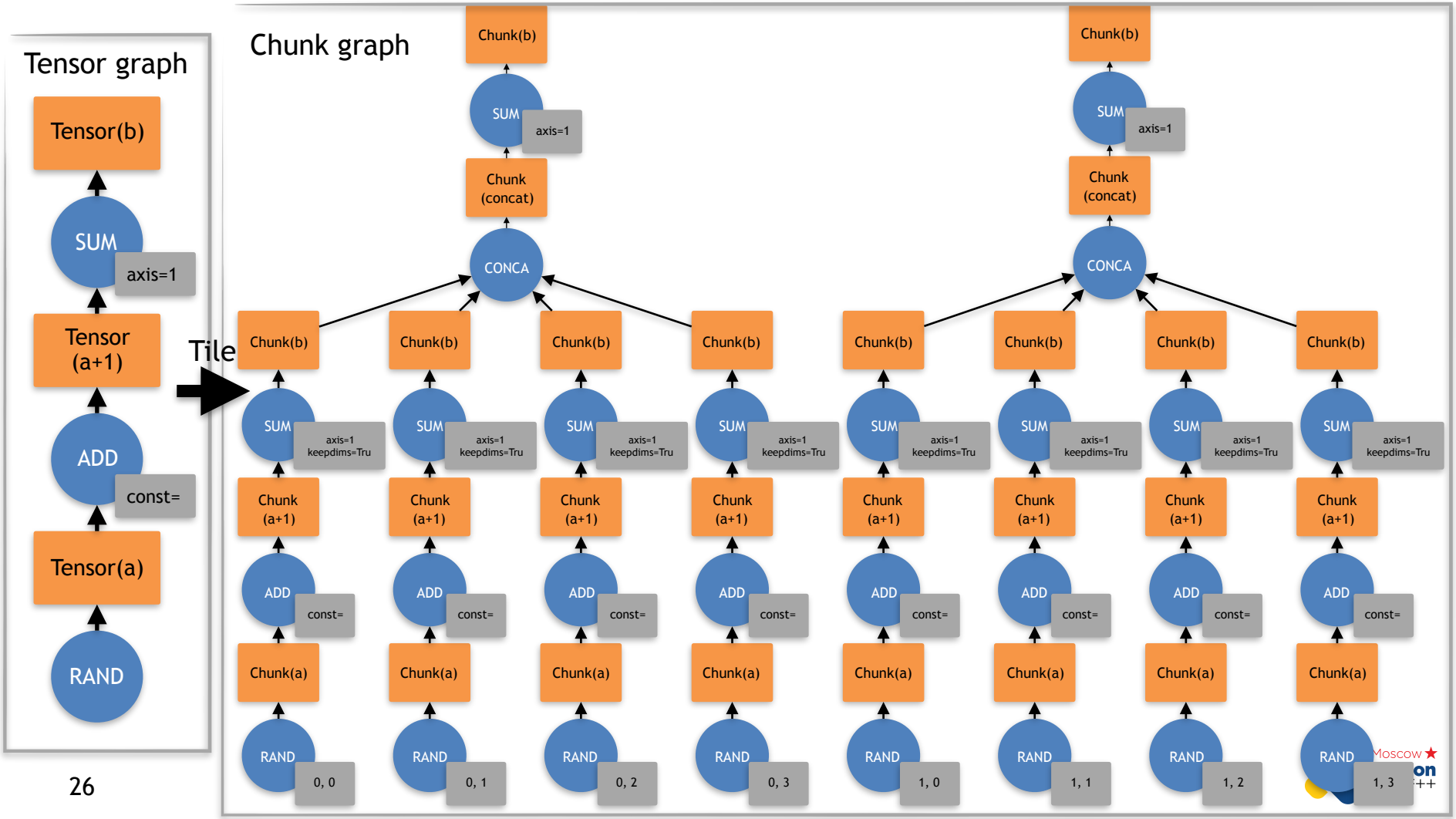
```
import mars.tensor as mt

a = mt.random.rand(1000, 2000)
b = (a + 1).sum(axis=1)
```



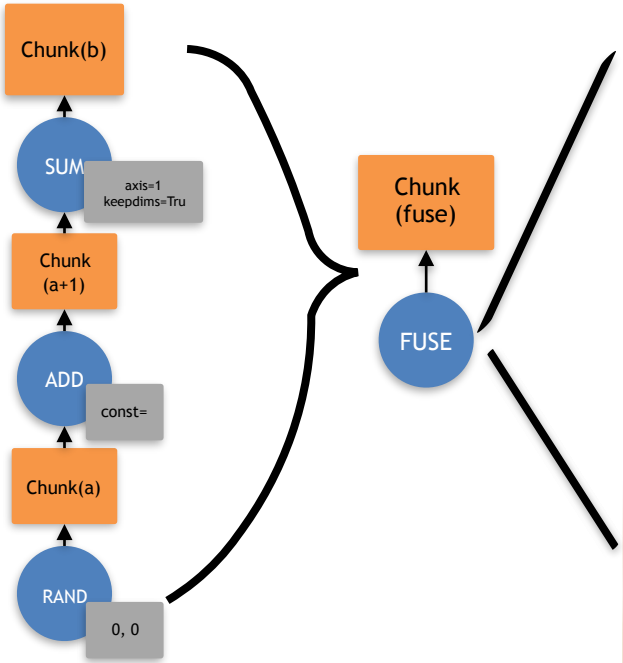


Tile: from tensor graph to chunk graph




Tensor graph
Tensor(b)
SUM
axis=1
Tensor (a+1)
ADD
const=
Tensor(a)
RAND
Tile
Chunk graph
Chunk(b)
SUM
axis=1
Chunk (concat)
CONCA
Chunk(b)
SUM
axis=1 keepdims=Tru
Chunk (a+1)
ADD
const=
Chunk(a)
RAND
0, 0
0, 1
0, 2
0, 3
1, 0
1, 1
1, 2
1, 3

Fuse optimization




```python
import numexpr as ne

ne.evaluate('sum(a+1, axis=1)')
```

```python
import cupy as cp

fusion_kernel = cp.ReductionKernel(
    'float64 x',   # input param
    'float64 z',   # output param
    'x + 1',       # map
    'a + b',       # reduce
    'z = a',       # post reduction
    '0',           # identity value
    'fusion'       # kernel name
)
fusion_kernel(a, axis=1)
```


Moscow Python Conf++ 2019

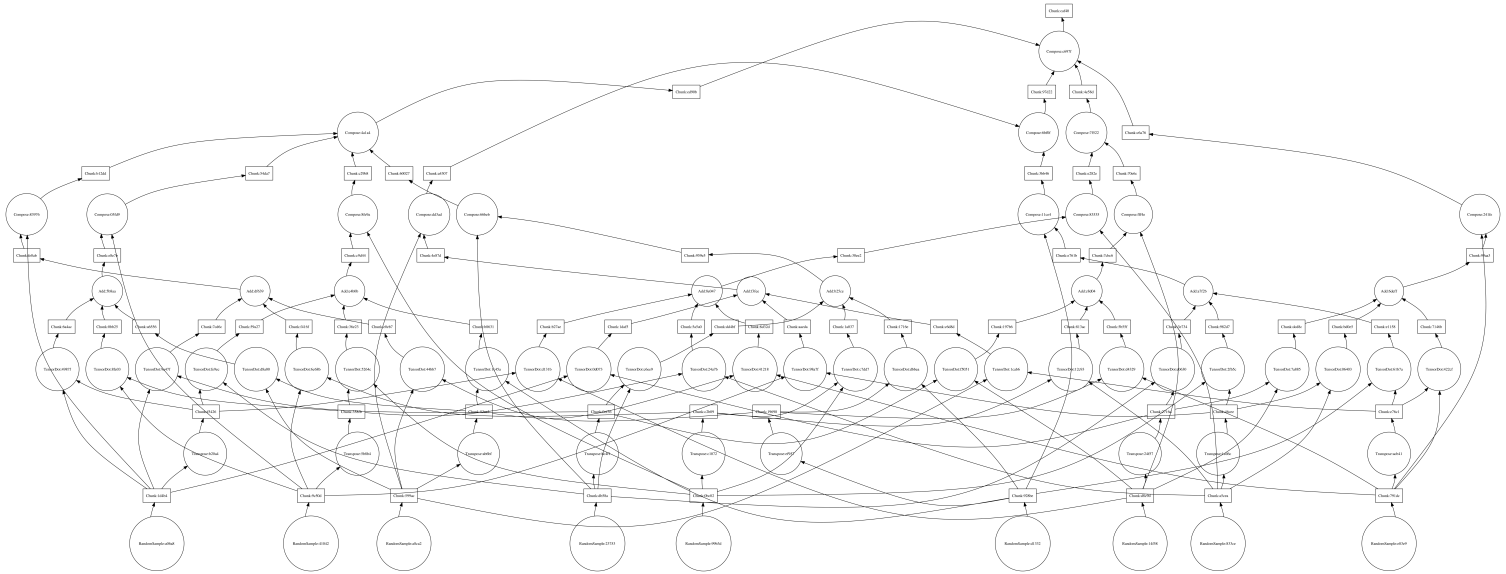
Complex chunk-level graph

```
1 import numpy as np
2
3 a = np.random.rand(100000, 100000)
4 (a.dot(a.T) - a).std()
```


Moscow Python Conf++ 2019

Complex chunk-level graph

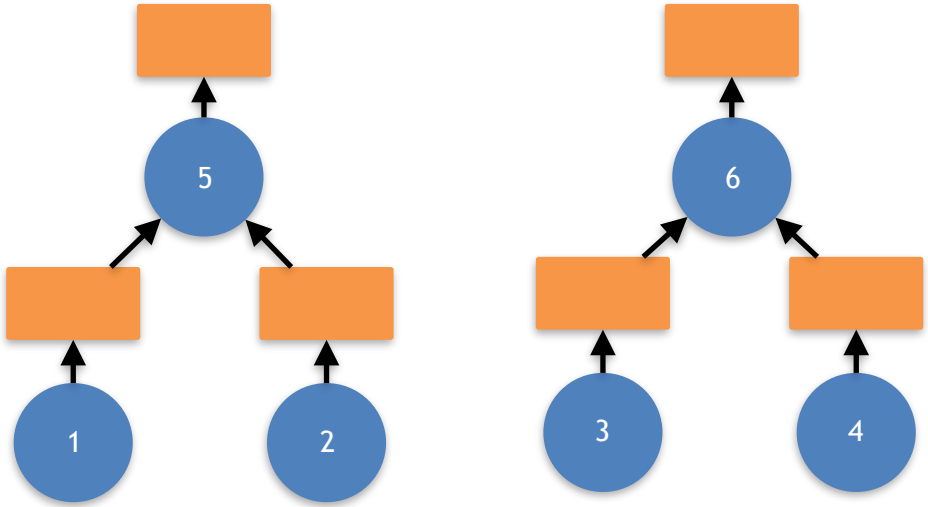
```
import numpy as np

a = np.random.rand(100000, 100000)
(a.dot(a.T) - a).std()
```



Moscow Python Conf++ 2019

Scheduling key points

- How operands with no inputs allocated to workers impact huge on performance
- Operands with bigger depth should be executed first



Moscow Python Conf++ 2019

Difficulties we were facing

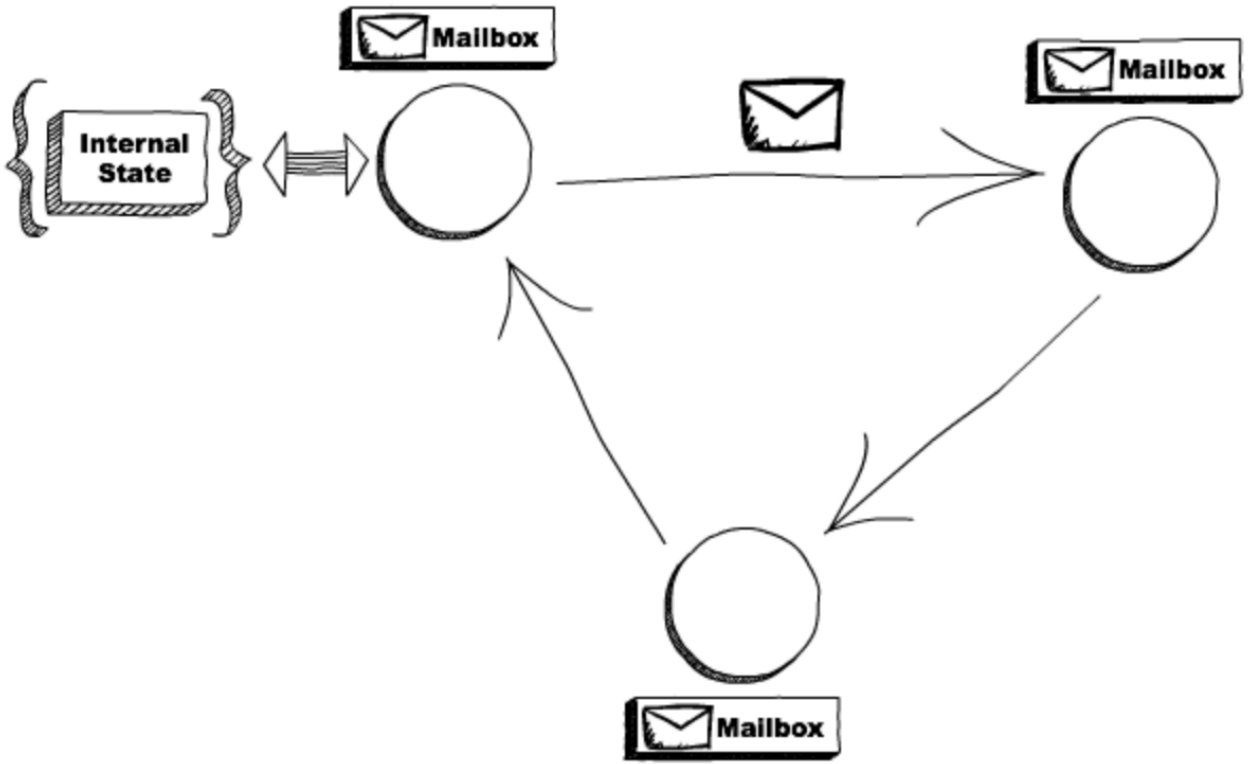
- Classic master-workers architecture, master could be a single point
- How can workers release GIL
- The logic of Master could be extremely complicated, how can we decouple the components to make the code neater and more understandable

Moscow Python Conf++ 2019

Actor model

Moscow Python Conf++ 2019

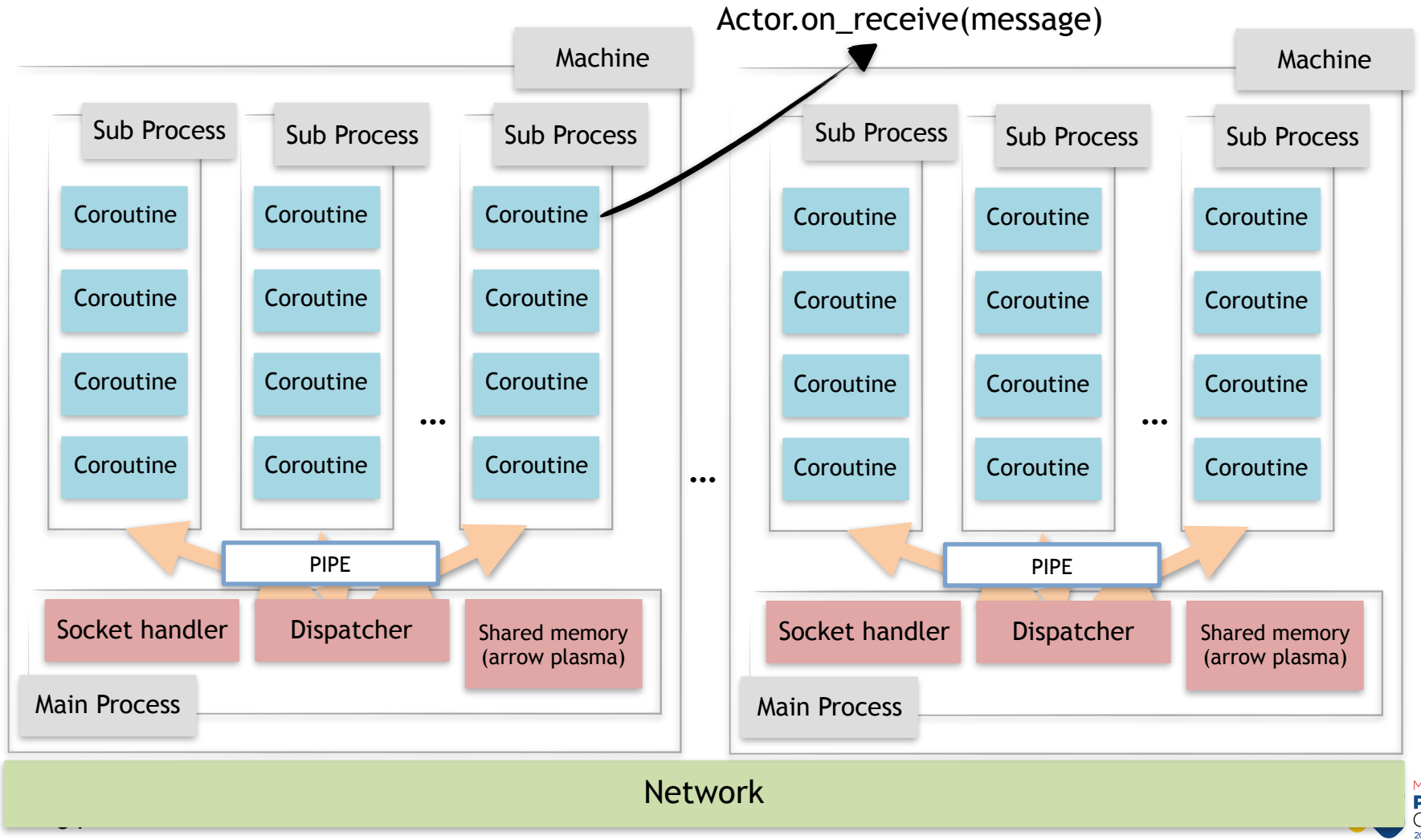
Actor model



Mars Actors

- No de facto actor model library for Python
- Mars actors
  - A lightweight actor model framework implemented from scratch
  - Mars distributed including schedulers and workers are based on Mars actors
  - Implemented with gevent, will support asycnio backend in the future

Moscow Python Conf++ 2019


Actor.on_receive(message)
Machine
Machine
Sub Process
Sub Process
Sub Process
Sub Process
Sub Process
Sub Process
Coroutine
PIPE
Socket handler
Dispatcher
Shared memory
(arrow plasma)
Main Process
Network

Implement estimating Pi with handwritten Actor model

```python
import numpy as np
from mars.actors import Actor, create_actor_pool


class ChunkInside(Actor):
    def on_receive(self, chunk_size):
        data = np.random.uniform(-1, 1, size=(chunk_size, 2))
        return (np.linalg.norm(data, axis=1) < 1).sum()


class PiCalculator(Actor):
    def on_receive(self, n):
        CHUNK_SIZE = 10 ** 7
        chunk_inside_refs = [self.ctx.create_actor(ChunkInside)
                             for _ in range(n // CHUNK_SIZE)]
        futures = [chunk_inside_ref.send(CHUNK_SIZE, wait=False)
                   for chunk_inside_ref in chunk_inside_refs]
        inside = sum(future.result() for future in futures)
        pi = 4 * inside / n
        return pi


N = 10 ** 10

with create_actor_pool(n_process=4) as pool:
    caculator_ref = pool.create_actor(PiCalculator)
    pi = caculator_ref.send(N)
    print('pi: %.5f' % pi)
```


Moscow Python Conf++ 2019

Implement estimating Pi with handwritten Actor model

```python
import numpy as np
from mars.actors import Actor, create_actor_pool


class ChunkInside(Actor):
    def on_receive(self, chunk_size):
        data = np.random.uniform(-1, 1, size=(chunk_size, 2))
        return (np.linalg.norm(data, axis=1) < 1).sum()


class PiCalculator(Actor):
    def on_receive(self, n):
        CHUNK_SIZE = 10 ** 7
        chunk_inside_refs = [self.ctx.create_actor(ChunkInside)
                             for _ in range(n // CHUNK_SIZE)]
        futures = [chunk_inside_ref.send(CHUNK_SIZE, wait=False)
                   for chunk_inside_ref in chunk_inside_refs]
        inside = sum(future.result() for future in futures)
        pi = 4 * inside / n
        return pi


N = 10 ** 10

with create_actor_pool(n_process=4) as pool:
    caculator_ref = pool.create_actor(PiCalculator)
    pi = caculator_ref.send(N)
    print('pi: %.5f' % pi)
```

Actor class

Moscow Python Conf++ 2019

Implement estimating Pi with handwritten Actor model

```python
import numpy as np
from mars.actors import Actor, create_actor_pool


class ChunkInside(Actor):
    def on_receive(self, chunk_size):
        data = np.random.uniform(-1, 1, size=(chunk_size, 2))
        return (np.linalg.norm(data, axis=1) < 1).sum()


class PiCalculator(Actor):
    def on_receive(self, n):
        CHUNK_SIZE = 10 ** 7
        chunk_inside_refs = [self.ctx.create_actor(ChunkInside)
                             for _ in range(n // CHUNK_SIZE)]
        futures = [chunk_inside_ref.send(CHUNK_SIZE, wait=False)
                   for chunk_inside_ref in chunk_inside_refs]
        inside = sum(future.result() for future in futures)
        pi = 4 * inside / n
        return pi


N = 10 ** 10

with create_actor_pool(n_process=4) as pool:
    caculator_ref = pool.create_actor(PiCalculator)
    pi = caculator_ref.send(N)
    print('pi: %.5f' % pi)
```

Actor class

create_actor can create actors on other machines by specifying **address**

Moscow Python Conf++ 2019

Implement estimating Pi with handwritten Actor model

```python
import numpy as np
from mars.actors import Actor, create_actor_pool


class ChunkInside(Actor):
    def on_receive(self, chunk_size):
        data = np.random.uniform(-1, 1, size=(chunk_size, 2))
        return (np.linalg.norm(data, axis=1) < 1).sum()


class PiCalculator(Actor):
    def on_receive(self, n):
        CHUNK_SIZE = 10 ** 7
        chunk_inside_refs = [self.ctx.create_actor(ChunkInside)
                             for _ in range(n // CHUNK_SIZE)]
        futures = [chunk_inside_ref.send(CHUNK_SIZE, wait=False)
                   for chunk_inside_ref in chunk_inside_refs]
        inside = sum(future.result() for future in futures)
        pi = 4 * inside / n
        return pi


N = 10 ** 10

with create_actor_pool(n_process=4) as pool:
    caculator_ref = pool.create_actor(PiCalculator)
    pi = caculator_ref.send(N)
    print('pi: %.5f' % pi)
```

Actor class

create_actor can create actors on other machines by specifying **address**

```
pi: 3.14162
CPU times: user 584 ms, sys: 280 ms, total: 863 ms
Wall time: 3min 31s
```


Moscow Python Conf++ 2019

Implement estimating Pi with handwritten Actor model

```python
import numpy as np
from mars.actors import Actor, create_actor_pool


class ChunkInside(Actor):
    def on_receive(self, chunk_size):
        data = np.random.uniform(-1, 1, size=(chunk_size, 2))
        return (np.linalg.norm(data, axis=1) < 1).sum()


class PiCalculator(Actor):
    def on_receive(self, n):
        CHUNK_SIZE = 10 ** 7
        chunk_inside_refs = [self.ctx.create_actor(ChunkInside)
                             for _ in range(n // CHUNK_SIZE)]
        futures = [chunk_inside_ref.send(CHUNK_SIZE, wait=False)
                   for chunk_inside_ref in chunk_inside_refs]
        inside = sum(future.result() for future in futures)
        pi = 4 * inside / n
        return pi


N = 10 ** 10

with create_actor_pool(n_process=4) as pool:
    caculator_ref = pool.create_actor(PiCalculator)
    pi = caculator_ref.send(N)
    print('pi: %.5f' % pi)
```

Actor class

create_actor can create actors on other machines by specifying **address**

```
pi: 3.14162
CPU times: user 584 ms, sys: 280 ms, total: 863 ms
Wall time: 3min 31s
```

Faster than multiprocessing

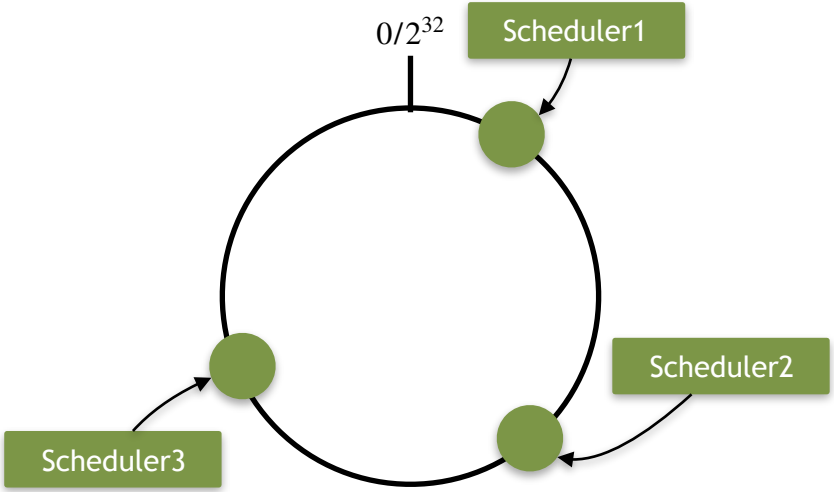
Moscow Python Conf++ 2019

Mars distributed

- Based on Mars Actors
  - Leverage multi-cores, release GIL
  - Easy to write distributed program
  - Great IO performance thanks to coroutine libraries
  - Easy to decouple logic into a bunch of actors
- Mars schedulers
  - Multiple schedulers, actors distributed to different schedulers according to consistent hashing
- Mars workers
  - Maintain a queue, schedule operand to slots according to the priority calculated before execution on schedulers
  - Fault tolerance if some workers are down

Moscow Python Conf++ 2019

Client



Worker1

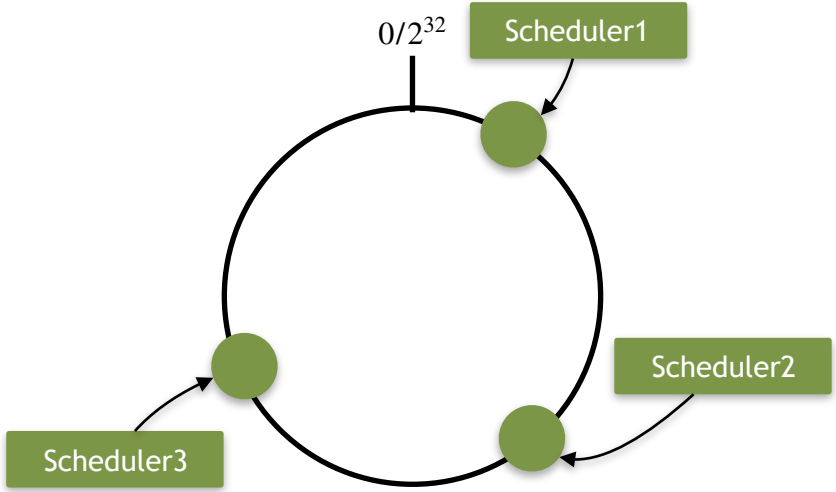
Worker2

Worker3

Worker4

Worker5

Moscow Python Conf++ 2019



Worker1

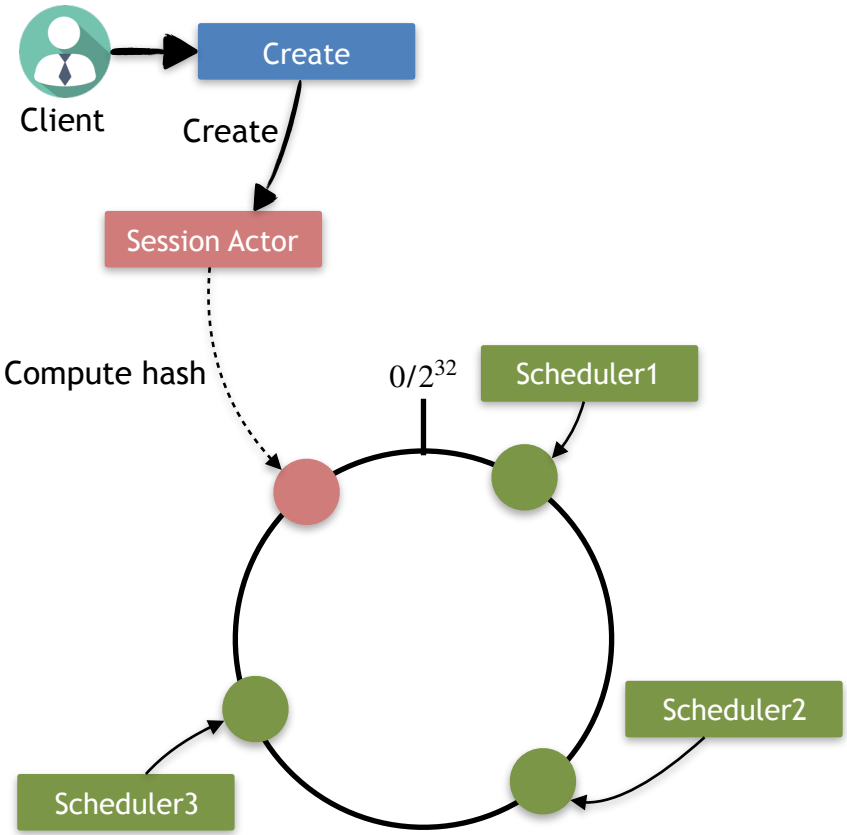
Worker2

Worker3

Worker4

Worker5

Moscow Python Conf++ 2019



Worker1

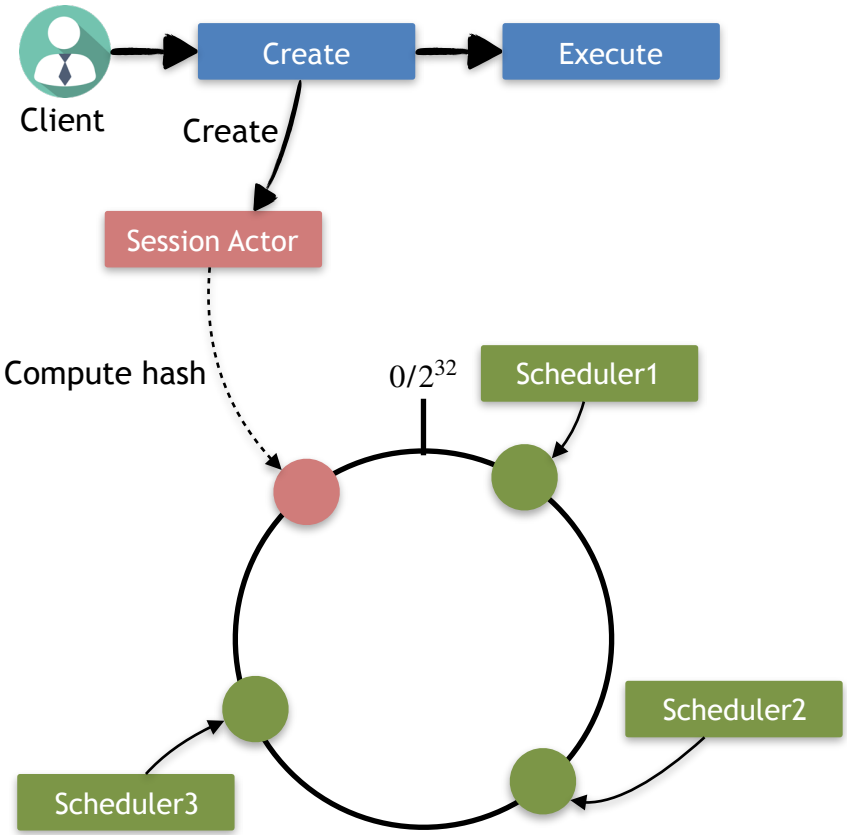
Worker2

Worker3

Worker4

Worker5

Moscow Python Conf++ 2019



Worker1

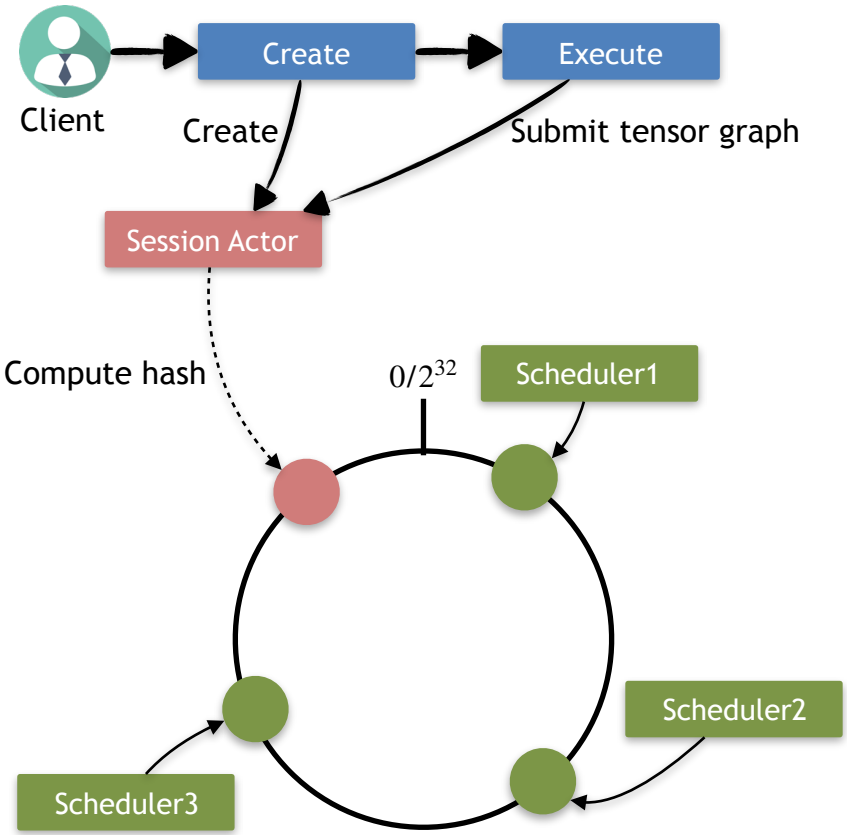
Worker2

Worker3

Worker4

Worker5

Moscow Python Conf++ 2019



Worker1

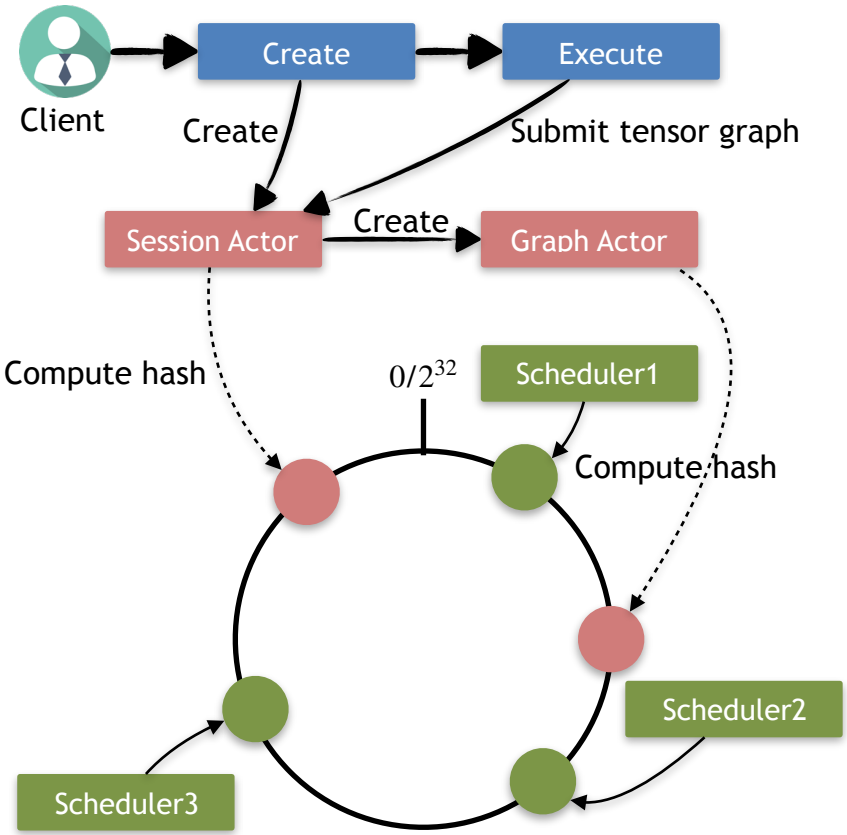
Worker2

Worker3

Worker4

Worker5

Moscow Python Conf++ 2019



Worker1

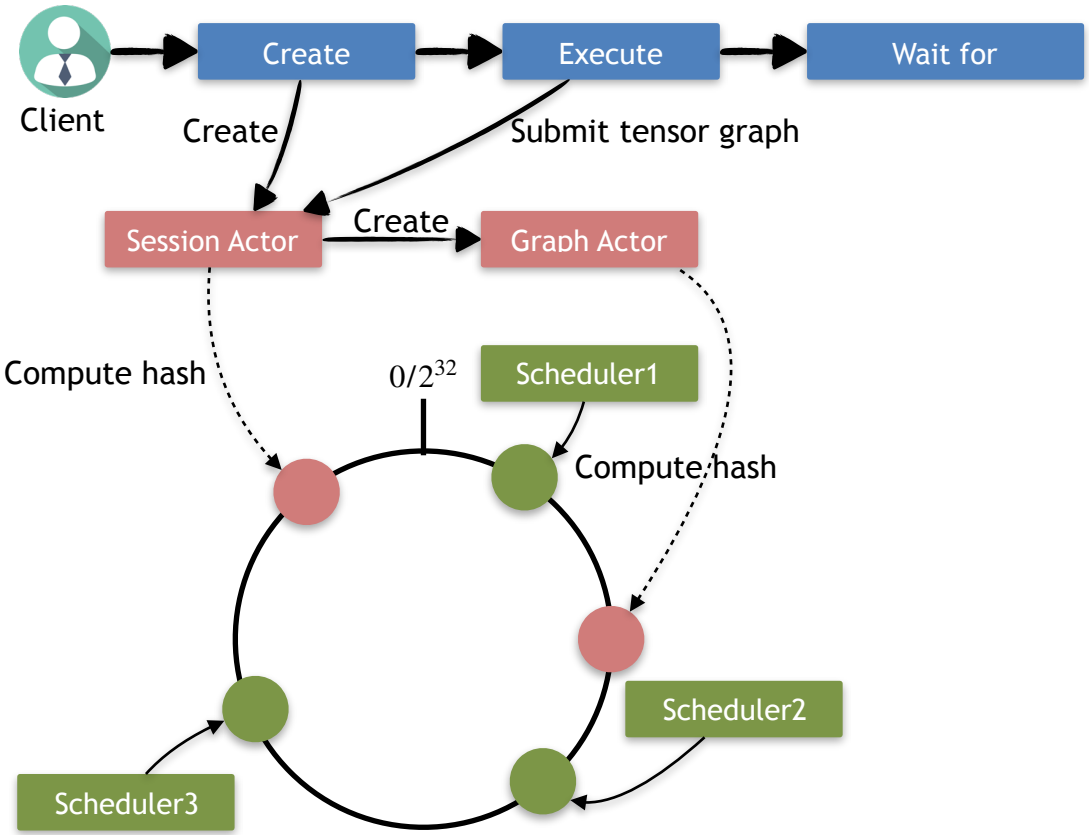
Worker2

Worker3

Worker4

Worker5

Moscow Python Conf++ 2019



Worker1

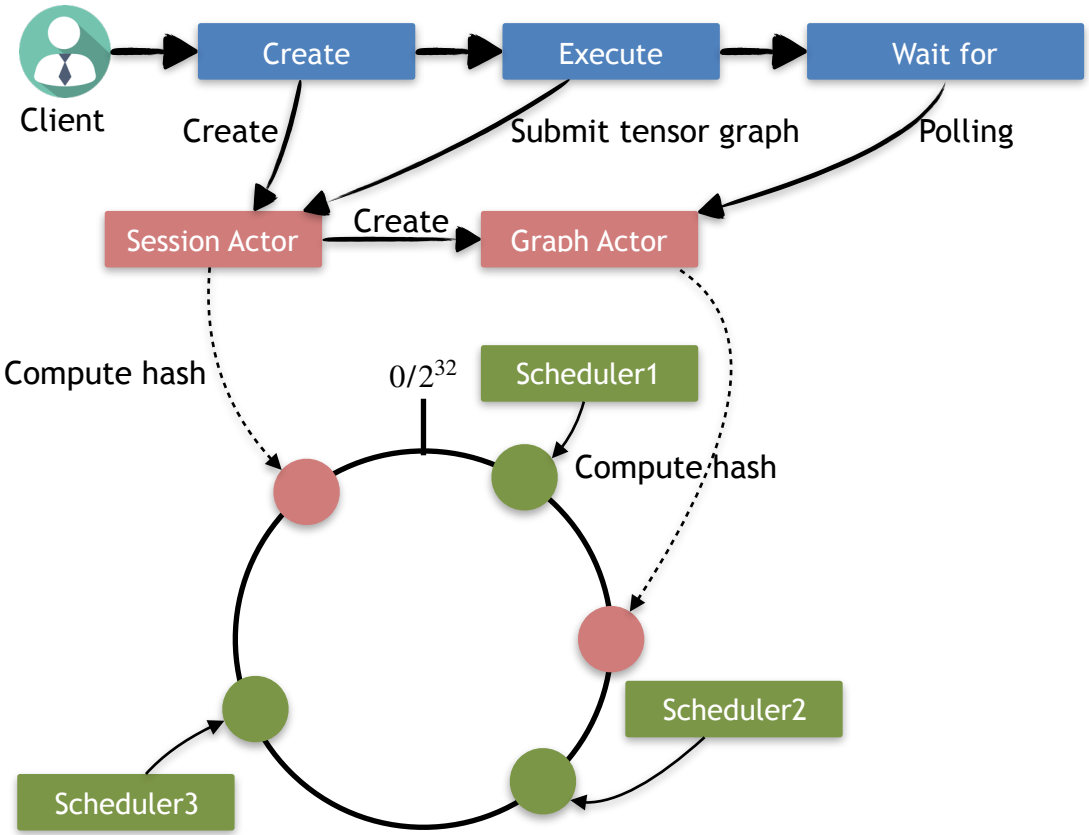
Worker2

Worker3

Worker4

Worker5

Moscow Python Conf++ 2019



Worker1

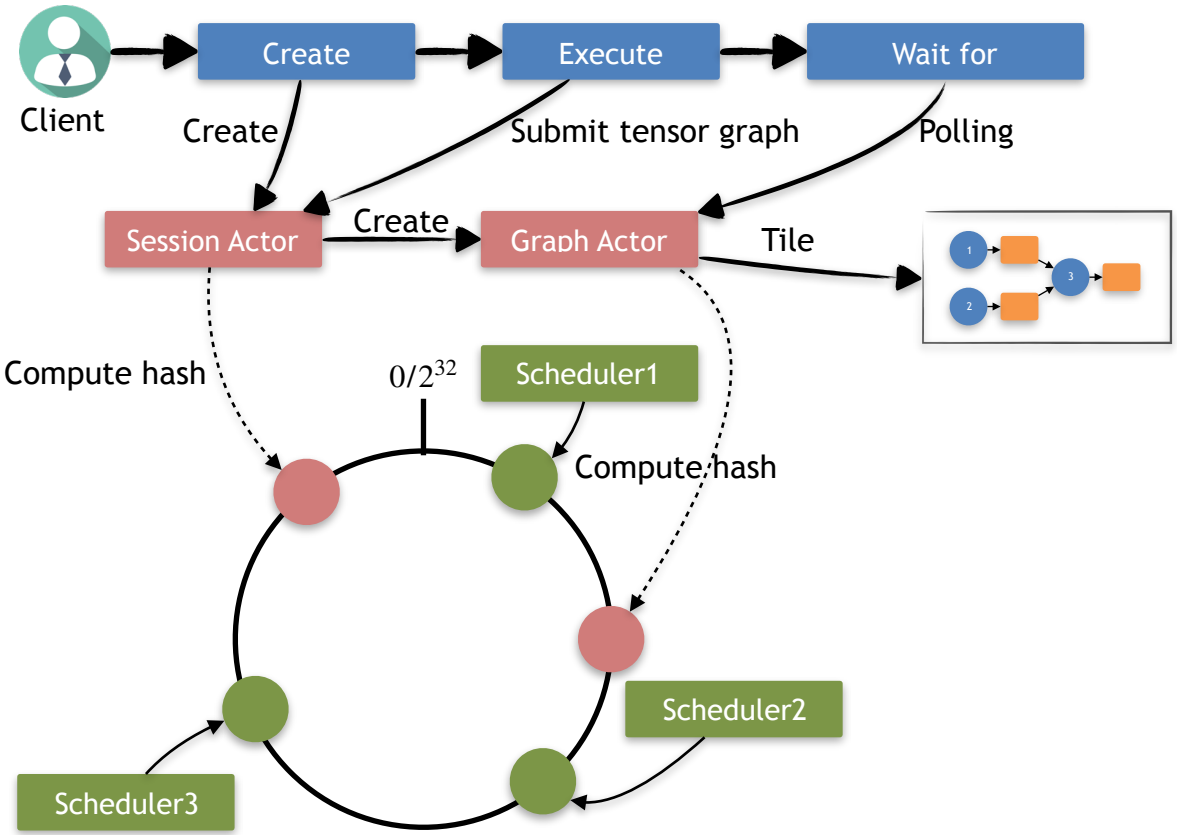
Worker2

Worker3

Worker4

Worker5

Moscow Python Conf++ 2019



Worker1

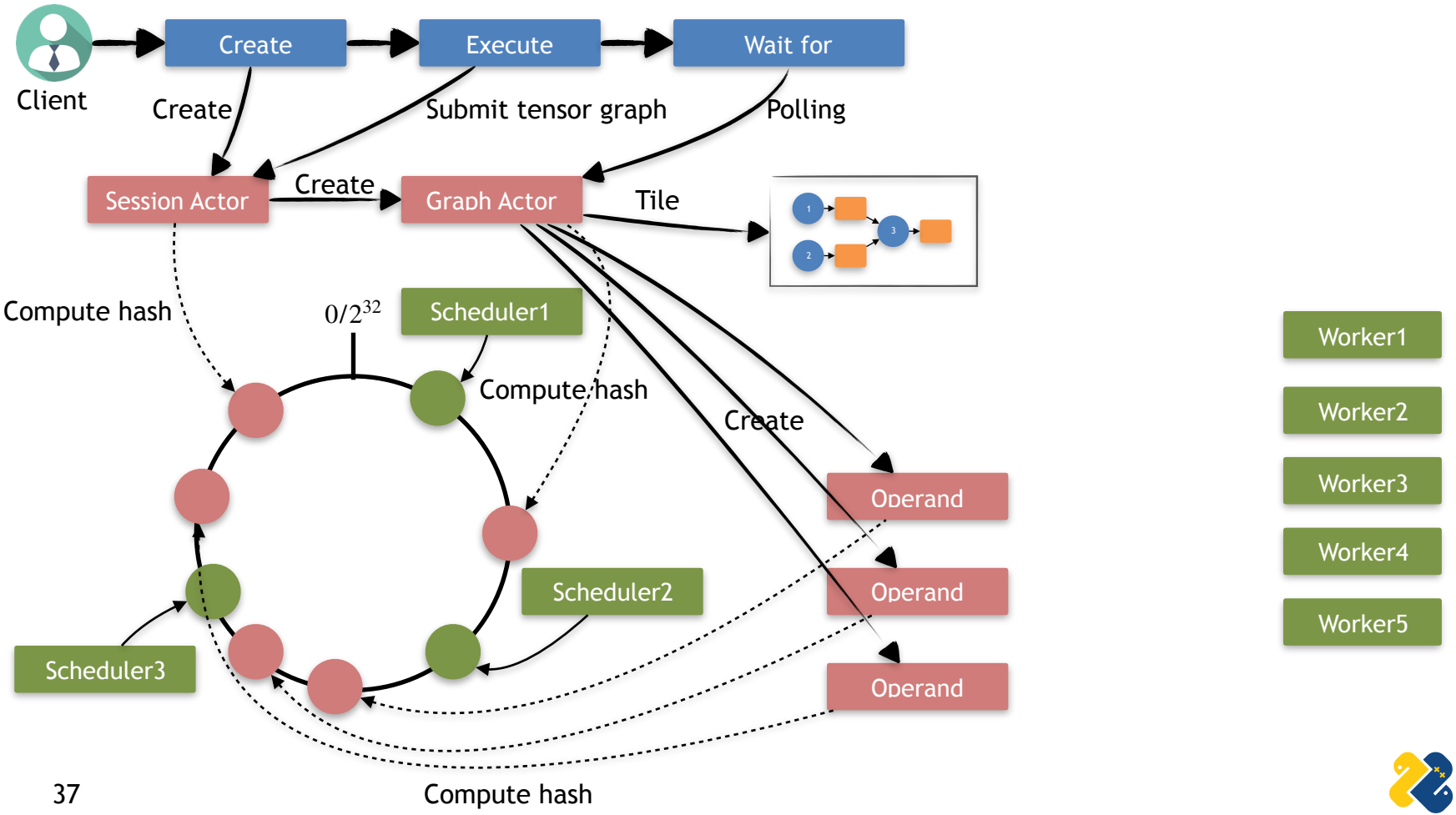
Worker2

Worker3

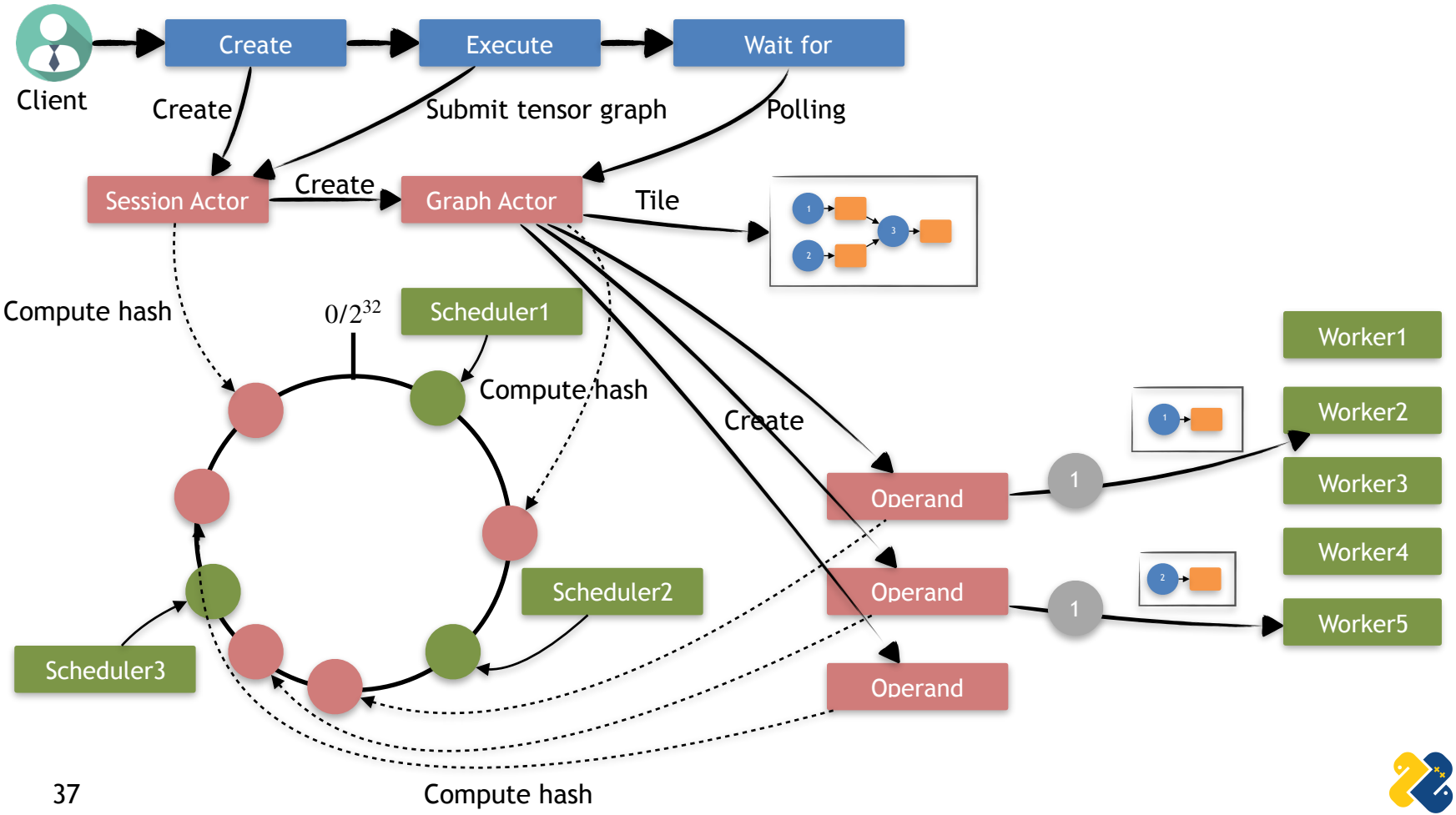
Worker4

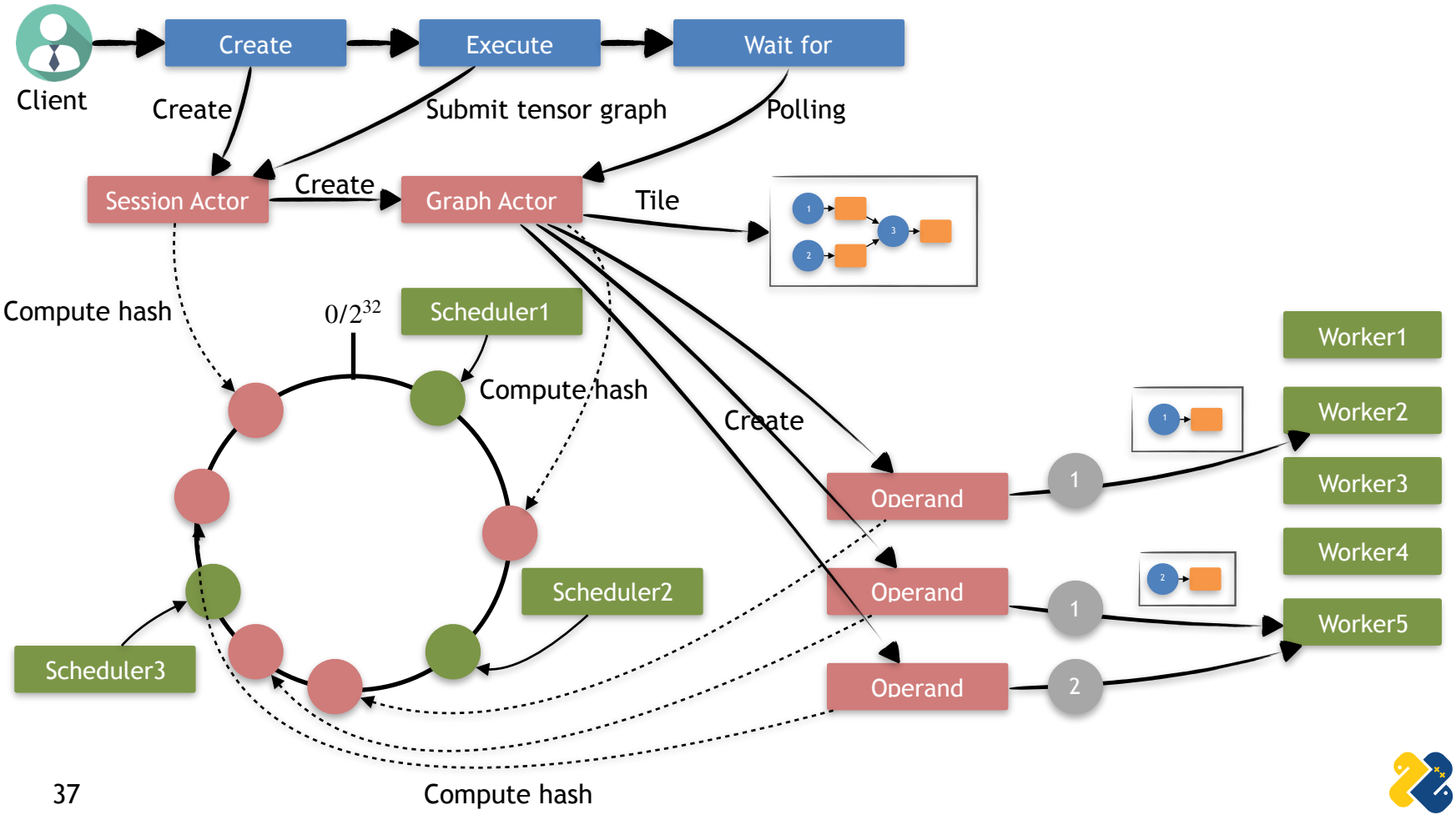
Worker5

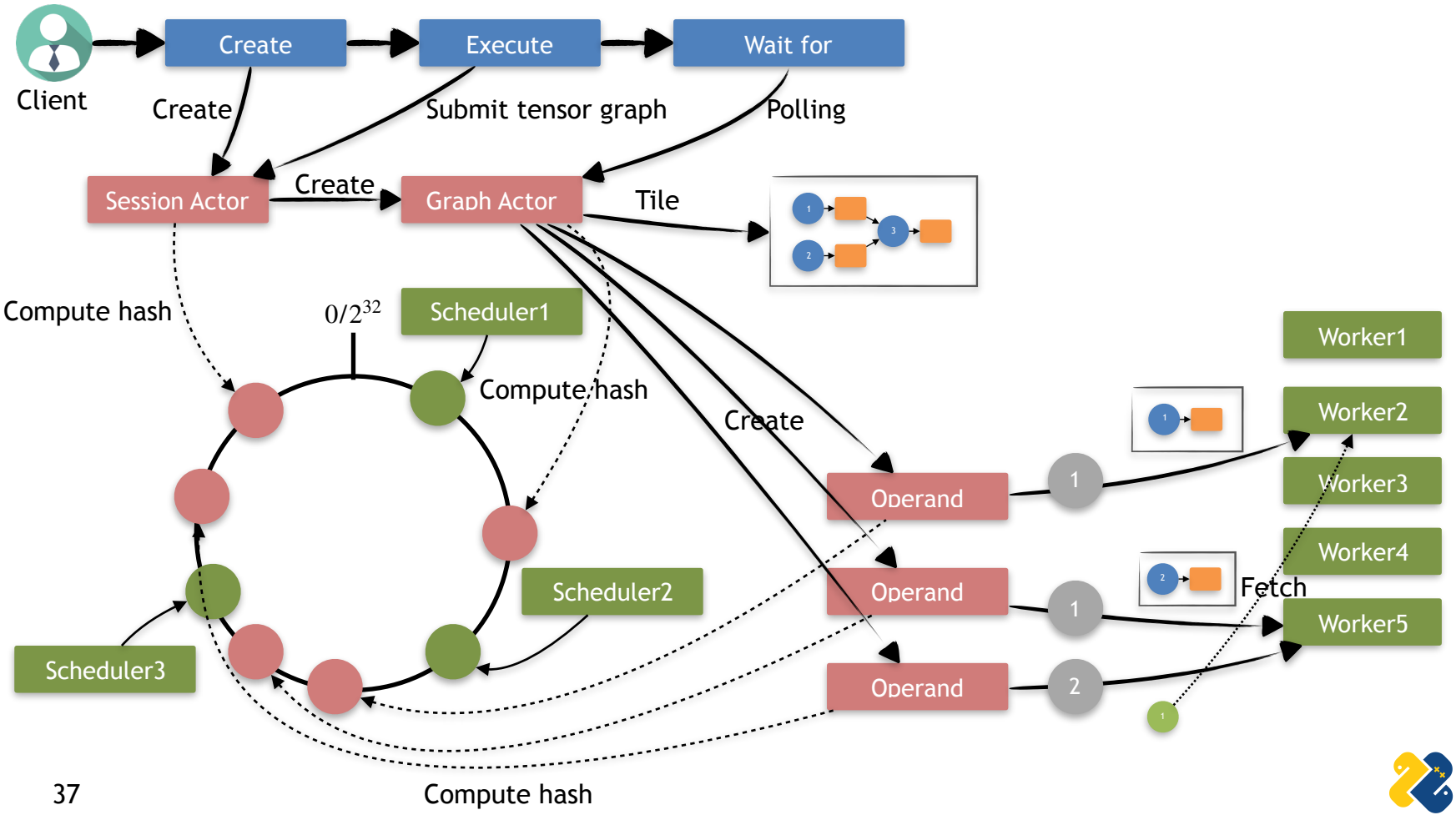
Moscow Python Conf++ 2019

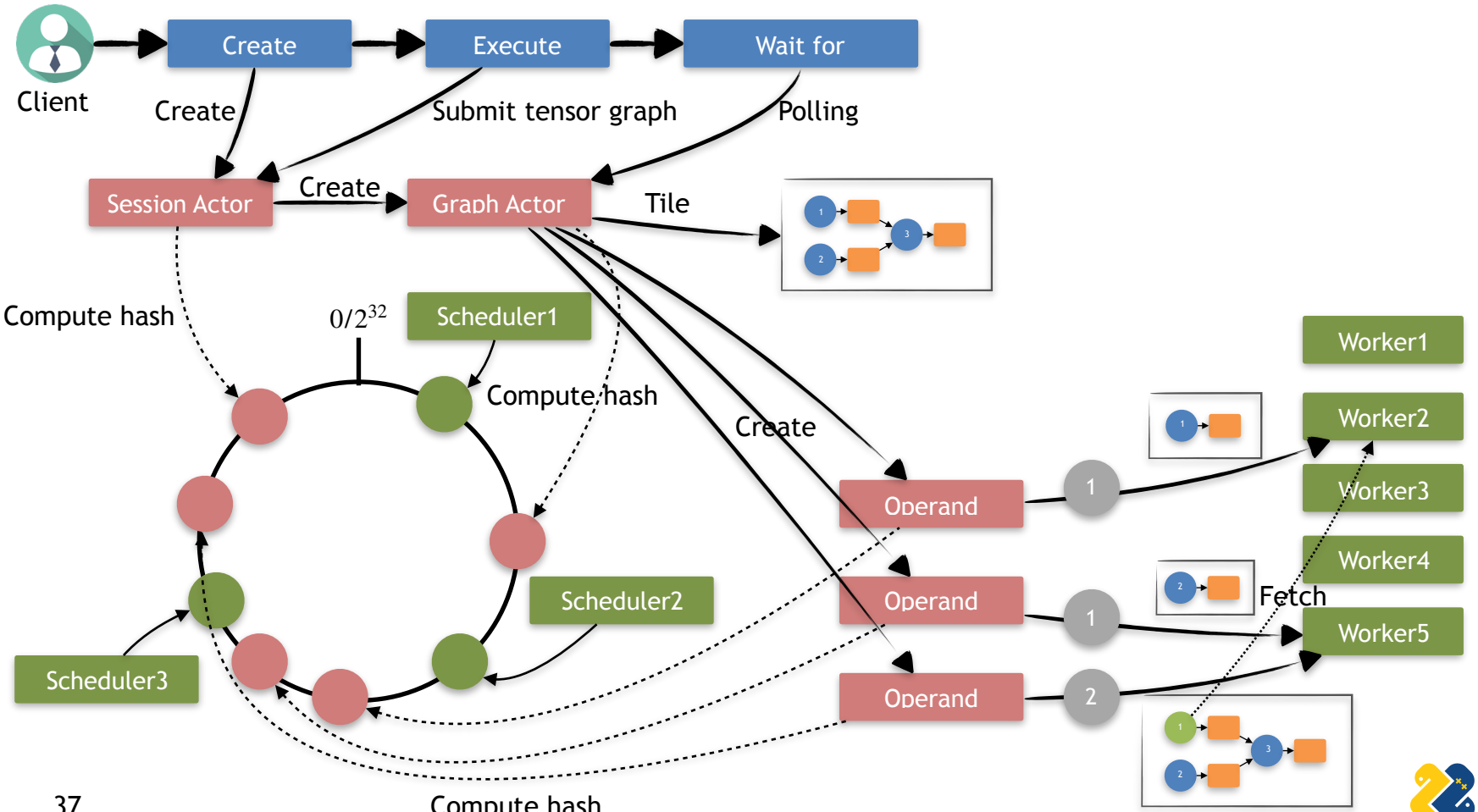


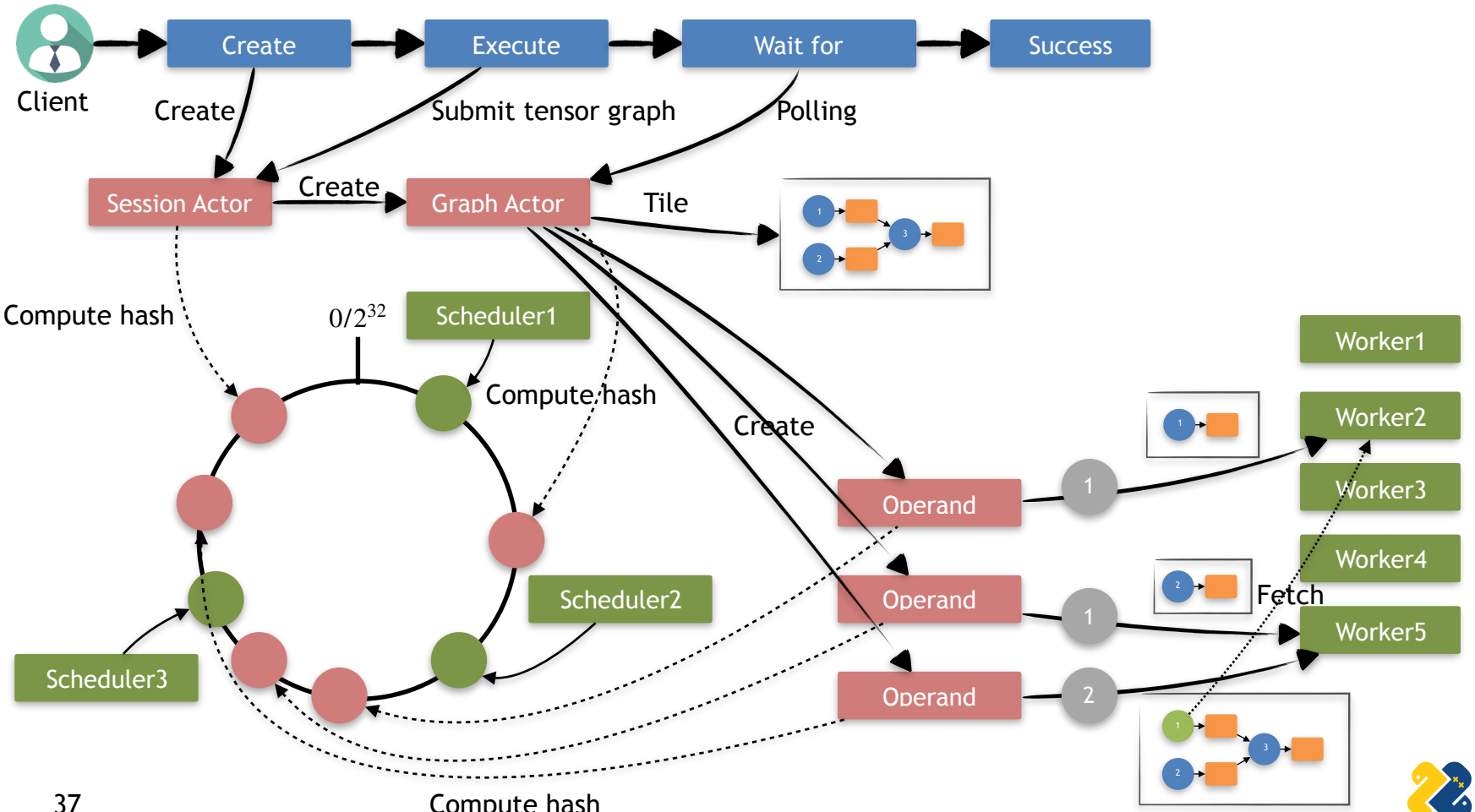
Moscow Python Conf++ 2019

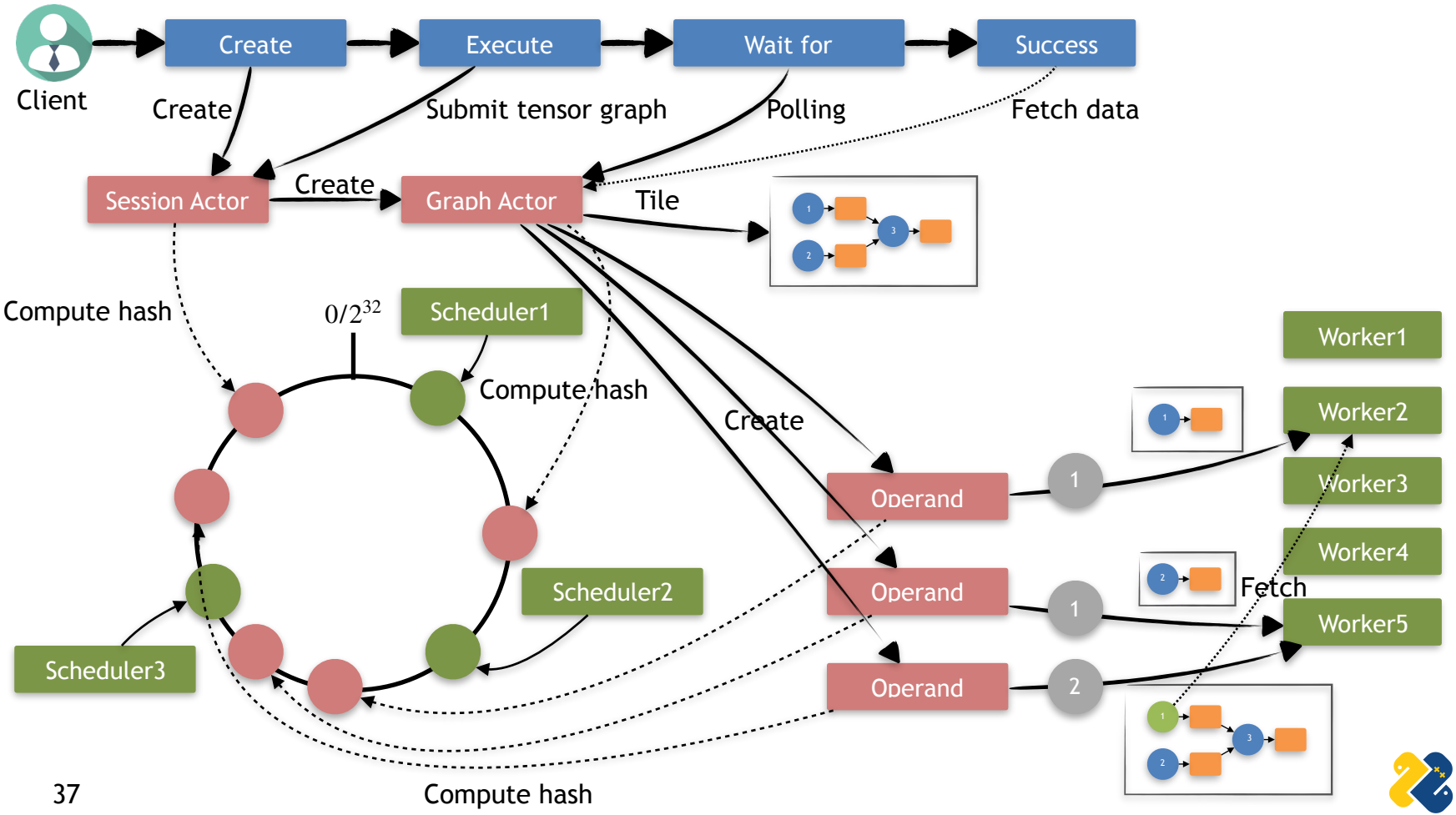

Client
Create
Execute
Wait for
Create
Submit tensor graph
Polling
Session Actor
Create
Graph Actor
Tile
Compute hash
0/2^32
Scheduler1
Compute hash
Create
Scheduler2
Scheduler3
Operand
Operand
Operand
1
1
Worker1
Worker2
Worker3
Worker4
Worker5
Compute hash

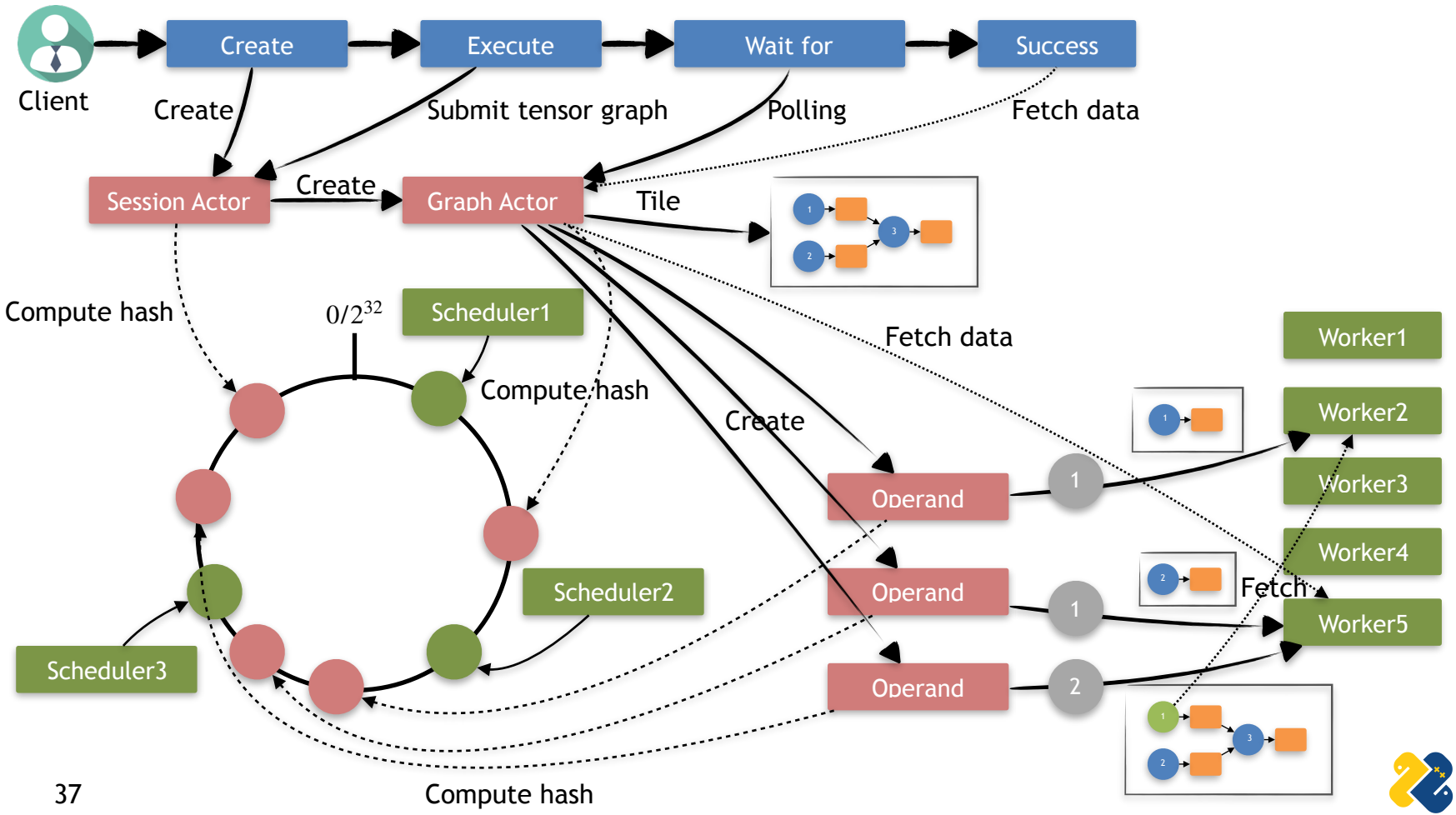

Client
Create
Execute
Wait for
Create
Submit tensor graph
Polling
Session Actor
Create
Graph Actor
Tile
Compute hash
0/2^32
Scheduler1
Compute hash
Create
Scheduler2
Scheduler3
Operand
Operand
Operand
Worker1
Worker2
Worker3
Worker4
Worker5
Compute hash

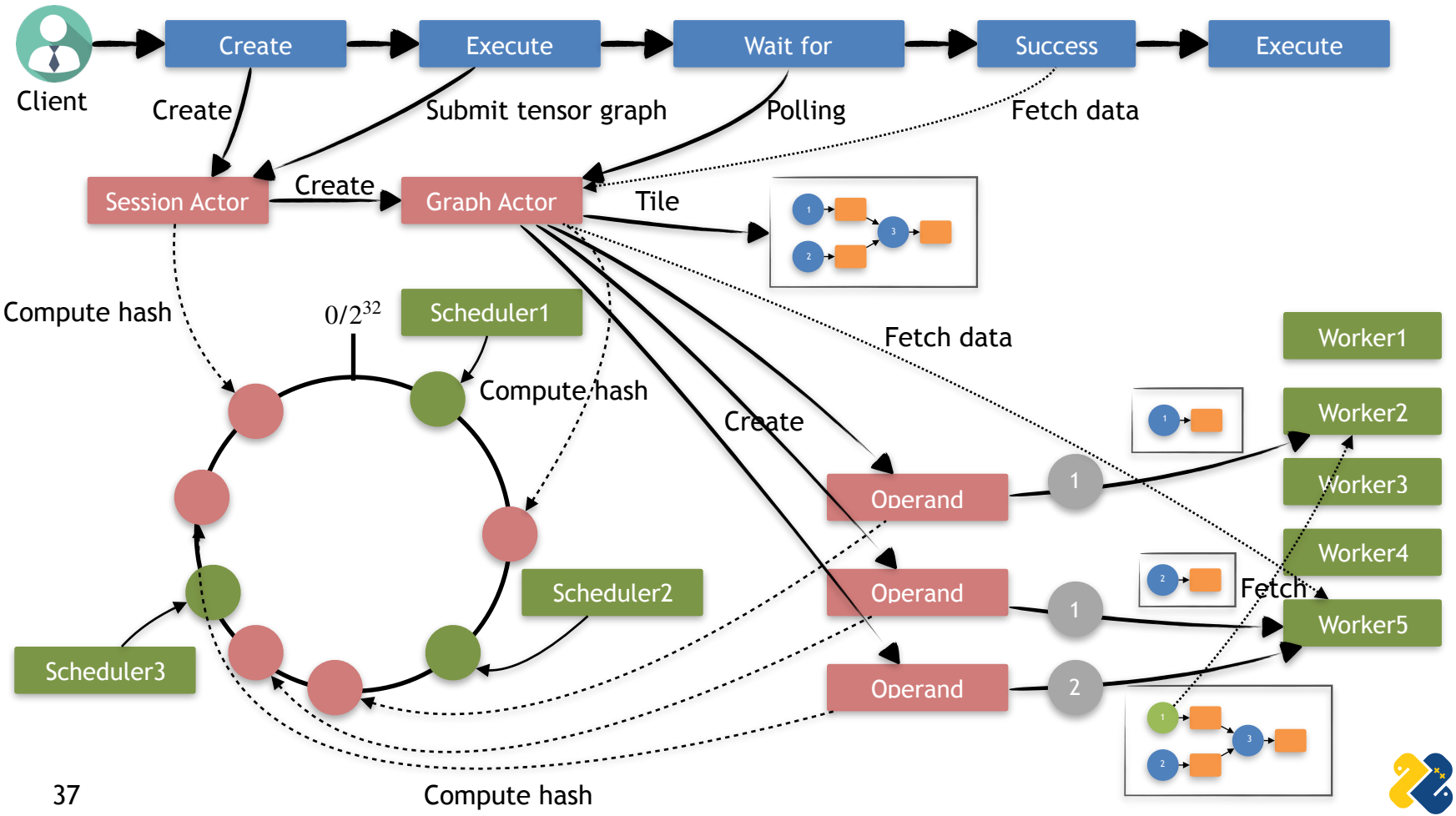

Client
Create
Execute
Wait for
Create
Submit tensor graph
Polling
Session Actor
Create
Graph Actor
Tile
Compute hash
0/2^32
Scheduler1
Compute hash
Create
Scheduler2
Scheduler3
Operand
Operand
Operand
Worker1
Worker2
Worker3
Worker4
Worker5
Fetch
Compute hash


Client
Create
Execute
Wait for
Create
Submit tensor graph
Polling
Session Actor
Create
Graph Actor
Tile
Compute hash
0/2^32
Scheduler1
Compute hash
Create
Scheduler2
Scheduler3
Operand
Operand
Operand
1
1
2
Worker1
Worker2
Worker3
Worker4
Worker5
Fetch
Compute hash


Client
Create
Execute
Wait for
Success
Create
Submit tensor graph
Polling
Session Actor
Create
Graph Actor
Tile
Compute hash
0/2^32
Scheduler1
Compute hash
Create
Scheduler2
Scheduler3
Operand
Operand
Operand
1
1
2
Worker1
Worker2
Worker3
Worker4
Worker5
Fetch
Compute hash

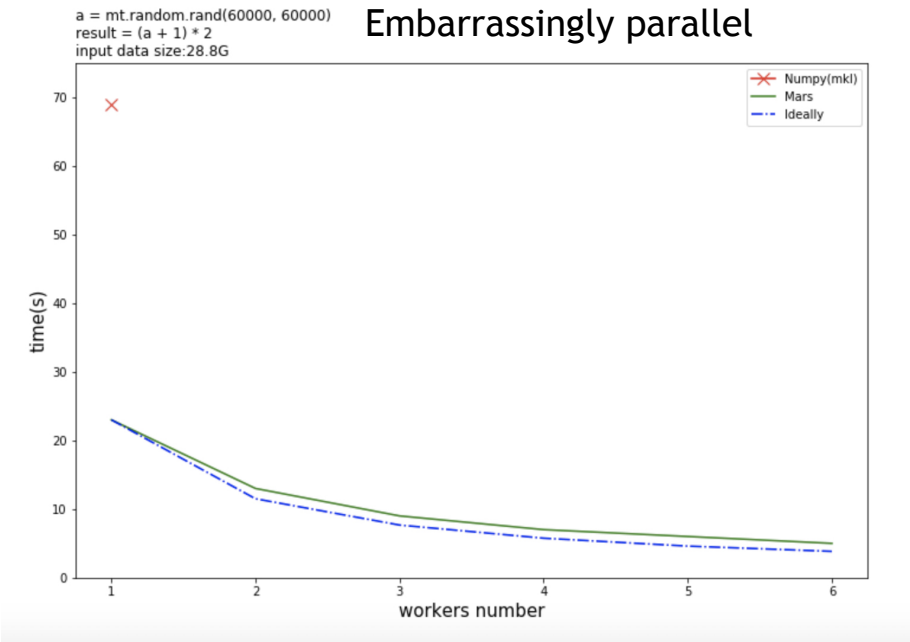

Client
Create
Execute
Wait for
Success
Create
Submit tensor graph
Polling
Fetch data
Session Actor
Create
Graph Actor
Tile
Compute hash
0/2^32
Scheduler1
Compute hash
Create
Scheduler2
Scheduler3
Operand
Operand
Operand
1
1
2
Worker1
Worker2
Worker3
Worker4
Worker5
Fetch
Compute hash

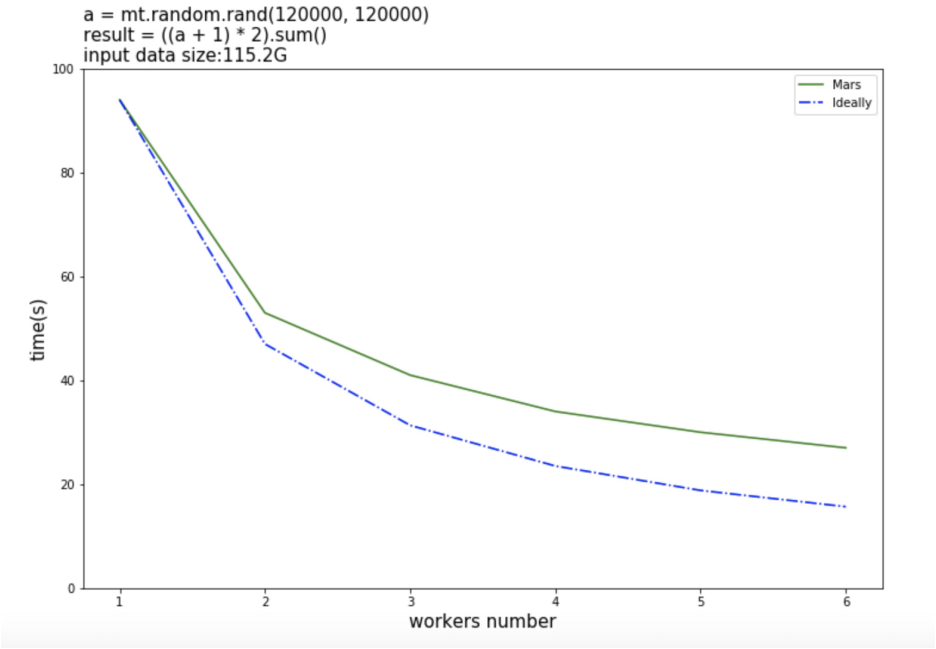

Client
Create
Execute
Wait for
Success
Create
Submit tensor graph
Polling
Fetch data
Session Actor
Create
Graph Actor
Tile
Compute hash
0/2^32
Scheduler1
Compute hash
Fetch data
Create
Worker1
Worker2
Worker3
Worker4
Worker5
Operand
Operand
Operand
Scheduler2
Scheduler3
Fetch
Compute hash


Client
Create
Execute
Wait for
Success
Execute
Create
Submit tensor graph
Polling
Fetch data
Session Actor
Create
Graph Actor
Tile
Compute hash
0/2^32
Scheduler1
Compute hash
Fetch data
Create
Worker1
Worker2
Worker3
Worker4
Worker5
Operand
Operand
Operand
Scheduler2
Scheduler3
Fetch
Compute hash

Benchmark

Embarrassingly parallel





Mars

- https://github.com/mars-project/mars
- https://docs.mars-project.io
- Welcome to use, open issue if find any bug, join our community, contribute code

Moscow Python Conf++ 2019

Roadmap

- Community means a lot for us
- Development
  - Mars tensor: Add more unimplemented Numpy and Scipy API
  - Mars DataFrame: mimic pandas API (working in progress)
  - Mars learn: distributed scikit-learn, XGBoost and so forth
  - Mars remote: remote functions and classes
  - Multiple languages support beyond Python

Moscow Python Conf++ 2019

Thank you for coming

Q&A