PYCON CHINA 2018

Mars—基于矩阵的统一计算框架

2018.10.14 中国 北京

秦续业

CONTENTS | 目录

- 背景
- 一个例子
- Mars 核心
- Mars 原理
- 未来计划

PYCON
CHINA 2018

python™

NumPy

jupyter

IP[y]: IPython
Interactive Computing

python™

Numba

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

matplotlib

Scipy

NumPy

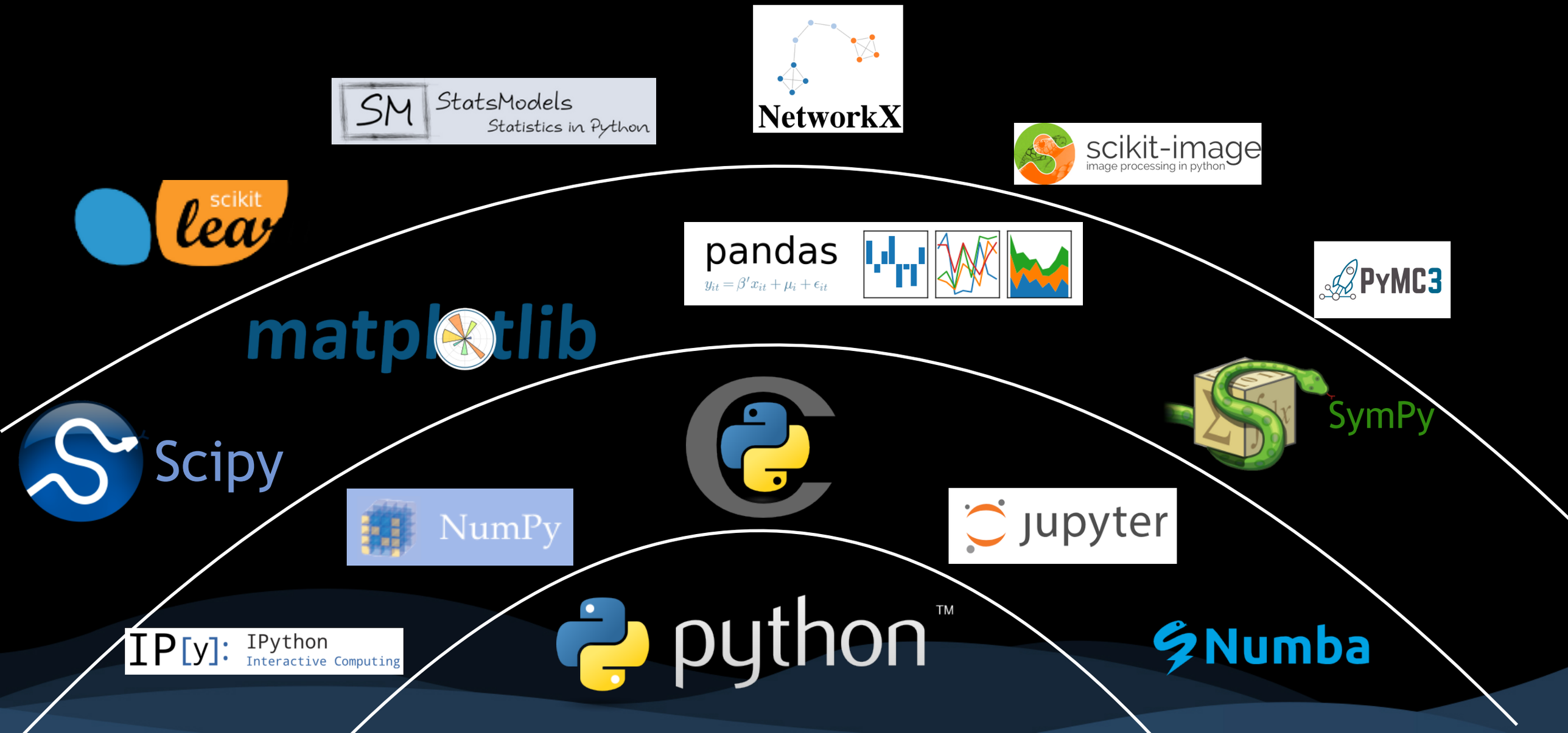
SymPy

jupyter

IP[y]: IPython Interactive Computing

python™

Numba


SM StatsModels
Statistics in Python
NetworkX
scikit-image
image processing in python
scikit learn
pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$
PyMC3
matplotlib
Scipy
SymPy
NumPy
jupyter
IP[y]: IPython
Interactive Computing
python™
Numba

biopython

DIPY

NIPY

PYCON CHINA 2018

The Astropy Project

sunpy
A Community Python Library for Solar Physics

SM StatsModels
Statistics in Python

NetworkX

scikit-image
image processing in python

还有很多
很多...

scikit learn

pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

PyMC3

matplotlib

Scipy

SymPy

NumPy

jupyter

IP[y]: IPython
Interactive Computing

python™

Numba

biopython

DIPY

NIPY

PYCON CHINA 2018

The Astropy Project

sunpy
A Community Python Library for Solar Physics

SM StatsModels
Statistics in Python

NetworkX

scikit-image
image processing in python

还有很多
很多...

scikit learn

pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

PyMC3

matplotlib

Scipy

SymPy

NumPy

jupyter

IP[y]: IPython
Interactive Computing

python™

Numba

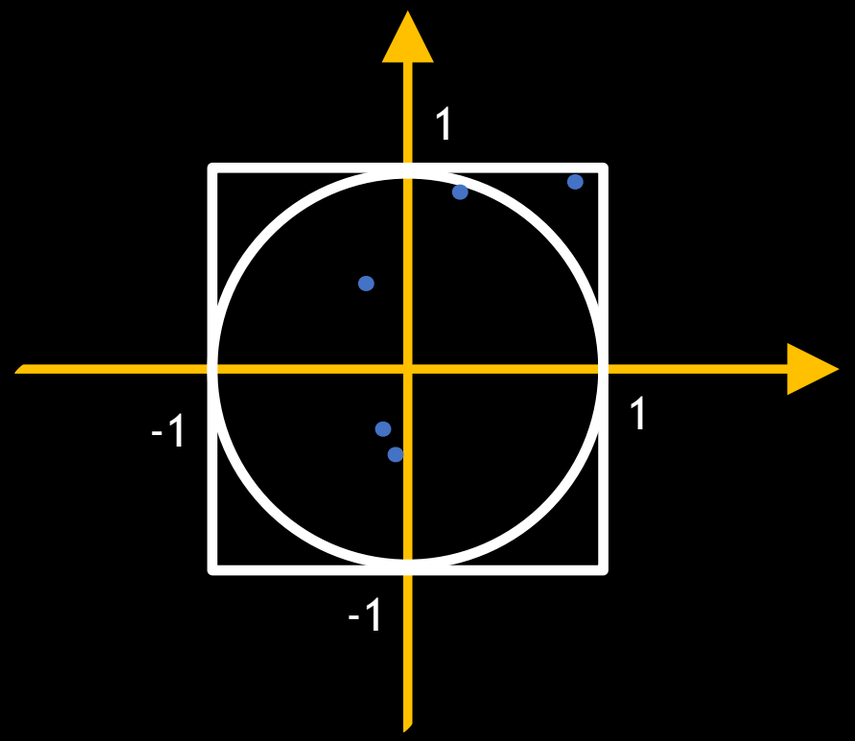
要理解数据的本质，往往需要

- 多维数组
- 矩阵乘法
- 傅里叶变换
- 线性代数
- 机器学习
- ……

CONTENTS | 目录

- 背景
- 一个例子
- Mars 核心
- Mars 原理
- 未来计划

蒙特卡洛方法求解 pi



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

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

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

纯 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
```

PYCON CHINA 2018

Cython 优化

PYCON CHINA 2018

```
import random
from math import sqrt

def cy_pi():
    cdef size_t N = 10 ** 7
    cdef size_t inside = 0
    cdef float x, y, pi

    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: 3.14116
CPU times: user 6.57 s, sys: 8.97 ms, total: 6.58 s
Wall time: 6.59 s
```

Cython 优化

PYCON CHINA 2018

```
import random
from math import sqrt

def cy_pi():
    cdef size_t N = 10 ** 7
    cdef size_t inside = 0
    cdef float x, y, pi

    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)
```

声明类型

```
pi: 3.14116
CPU times: user 6.57 s, sys: 8.97 ms, total: 6.58 s
Wall time: 6.59 s
```

Cython 优化

PYCON CHINA 2018

```
import random
from math import sqrt

def cy_pi():
    cdef size_t N = 10 ** 7
    cdef size_t inside = 0
    cdef float x, y, pi

    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)
```

声明类型

```
pi: 3.14116
CPU times: user 6.57 s, sys: 8.97 ms, total: 6.58 s
Wall time: 6.59 s
```

提升 ~40%

使用 Numpy


```
import numpy as np

N = 10 ** 7

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)
```

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

使用 Numpy


```
import numpy as np

N = 10 ** 7

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)
```

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

提升 10x+

使用 numexpr 进一步优化


```python
import numpy as np
import numexpr as ne

N = 10 ** 7

x = np.random.uniform(-1, 1, N)
y = np.random.uniform(-1, 1, N)
inside = ne.evaluate(
    'sum(where(sqrt(x ** 2 + y ** 2) < 1, 1, 0))')
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

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

使用 numexpr 进一步优化


```
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)
```

多个操作合并

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

使用 numexpr 进一步优化


```
import numpy as np
import numexpr as ne

N = 10 ** 7

x = np.random.uniform(-1, 1, N)
y = np.random.uniform(-1, 1, N)
inside = ne.evaluate(
    'sum(where(sqrt(x ** 2 + y ** 2) < 1, 1, 0))')
pi = 4 * inside / N
print('pi: %.5f' % pi)
```

多个操作合并

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

提升 25x+

使用 cupy 用 GPU 加速


```
import cupy as cp

N = 10 ** 7

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

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

使用 cupy 用 GPU 加速

PYCON CHINA 2018

```
import cupy as cp

N = 10 ** 7

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

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

使用 cupy 用 GPU 加速


```
import cupy as cp

N = 10 ** 7

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

Import 替换

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

使用 cupy 用 GPU 加速


```
import cupy as cp

N = 10 ** 7

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

Import 替换

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

提升 270x+

现在我们把问题规模扩大 1000 倍

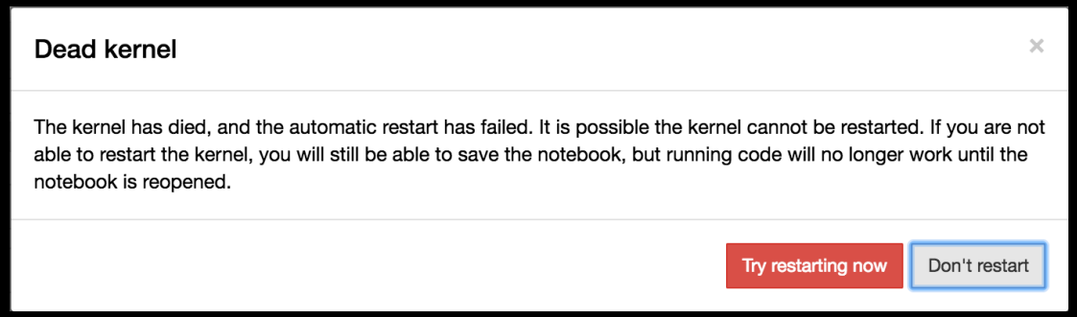
```python
import numpy as np

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)
```

```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)
```

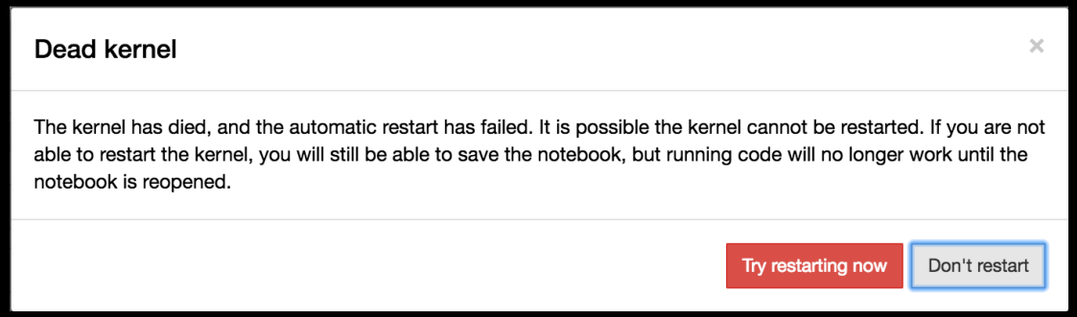
```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-1-28227489136f> in <module>()
      3 N = 10 ** 10
      4
----> 5 data = np.random.uniform(-1, 1, size=(N, 2))
      6 inside = (np.sqrt((data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

更惨的是

更惨的是


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

更惨的是


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

38

分块处理


```python
import numpy as np

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

inside = 0
for _ in range(N // CHUNK_SIZE):
    data = np.random.uniform(-1, 1, size=(
            CHUNK_SIZE, 2))
    inside += (np.linalg.norm(data, axis=1) < 1)
                .sum()

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

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

分块处理

```
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=(
9              CHUNK_SIZE, 2))
10     inside += (np.linalg.norm(data, axis=1) < 1)
11                .sum()
12
13 pi = 4 * inside / N
14 print('pi: %.5f' % pi)
```

处理 1000 个分块

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

分块处理


```python
import numpy as np

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

inside = 0
for _ in range(N // CHUNK_SIZE):
    data = np.random.uniform(-1, 1, size=(
            CHUNK_SIZE, 2))
    inside += (np.linalg.norm(data, axis=1) < 1)
               .sum()

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

处理 1000 个分块

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

只有单核运行

iPad.

iPad.

Numpy 并行化?

Numpy 并行化?

- 使用支持并行的原语，tensordot 矩阵乘法支持并行
- 多线程/多进程
- 分布式

多线程和多进程

多线程和多进程

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

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

PYCON CHINA 2018

多线程和多进程

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

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

提升至 ~2x

PYCON CHINA 2018

多线程和多进程

PYCON CHINA 2018

```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
```

提升至 ~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
```

多线程和多进程

PYCON CHINA 2018

```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
```

提升至 ~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
```

多线程和多进程

PYCON CHINA 2018

```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
```

提升至 ~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)
```

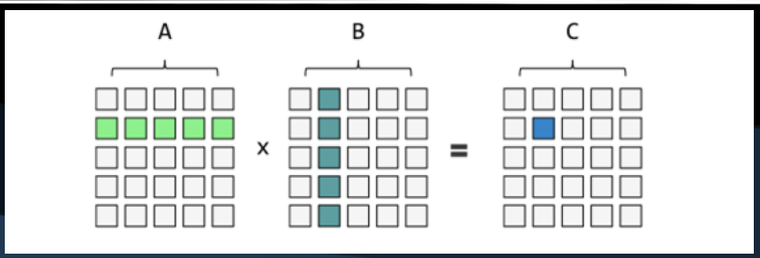
```
pi: 3.14158
CPU times: user 929 ms, sys: 378 ms, total: 1.31
s
Wall time: 3min 38s
```

提升至 3x+

- 蒙特卡洛方法计算 pi 能充分并行
- 考虑更复杂的情况
- 分布式?

- 蒙特卡洛方法计算 pi 能充分并行
- 考虑更复杂的情况
- 分布式?

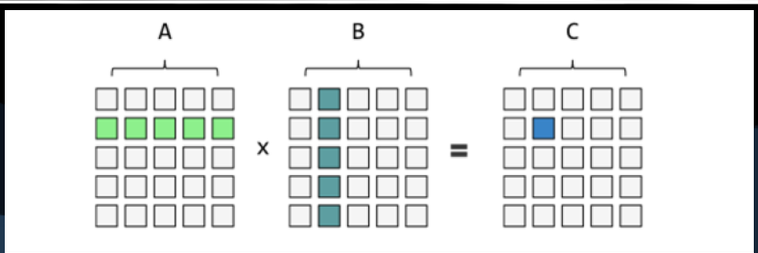
```
import numpy as np

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



- 蒙特卡洛方法计算 pi 能充分并行
- 考虑更复杂的情况
- 分布式?

```
import numpy as np

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

数据量~75G

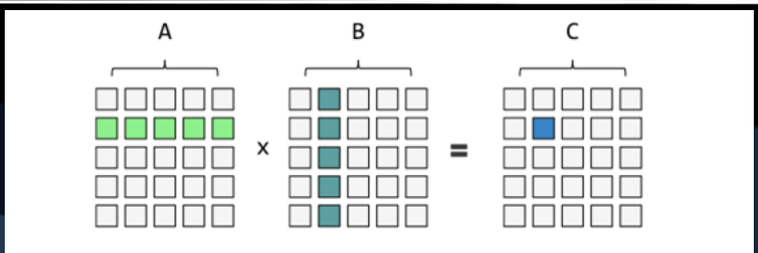


- 蒙特卡洛方法计算 pi 能充分并行
- 考虑更复杂的情况
- 分布式?

```
import numpy as np

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

数据量~75G

手写任务并行极其困难



我们需要什么样的框架?

- 熟悉的接口，import 替换
- 可扩展性
  - Scale in：单机并行执行
  - Scale out：分布式，千台规模
- 支持硬件加速：GPU 执行
- 优化：操作合并（fusion）
- 内存计算，但支持溢出控制

CONTENTS | 目录

- 背景
- 一个例子
- Mars 核心
- Mars 原理
- 未来计划

Mars tensor：分布式多维矩阵计算框架

```
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)
```

Mars tensor：分布式多维矩阵计算框架

```
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，OOM

Mars tensor：分布式多维矩阵计算框架

```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，OOM

```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)
```

Mars tensor：分布式多维矩阵计算框架

```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，OOM

```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)
```

Mars tensor：分布式多维矩阵计算框架

import 替换

```
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，OOM

```
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)
```

Mars tensor：分布式多维矩阵计算框架

```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，OOM

import 替换

```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)
```

手动触发执行

Mars tensor：分布式多维矩阵计算框架

```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，OOM

import 替换

```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)
```

手动触发执行

```
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
```

Mars tensor：分布式多维矩阵计算框架

import 替换

```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，OOM

```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)
```

手动触发执行

```
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
```

单机执行和手写并行相当

Mars tensor：分布式多维矩阵计算框架

```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，OOM

import 替换

```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)
```

手动触发执行

```
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
```

占用少量内存

单机执行和手写并行相当

Tensor API

- numpy 接口 100% 兼容
- 实现了 70% 的常见 numpy 接口
  - 创建 tensor：ones、empty、zeros、ones_like 等
  - 随机采样：rand、randint、beta、binomial 等
  - 基本操作：astype、transpose、broadcast_to 等
  - 聚合操作：sum、nansum、max、all、mean 等
  - 索引：切片、布尔索引、花式索引（部分支持）、增加维度、省略号
  - 离散傅里叶变换
  - 线性代数：qr、svd、cholesky 分解；范数
  - rechunk

GPU 和稀疏矩阵

PYCON CHINA 2018

```python
import mars.tensor as mt

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

GPU

```python
import mars.tensor as mt

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

Sparse

CONTENTS | 目录

- 背景
- 一个例子
- Mars 核心
- **Mars 原理**
- 未来计划

Mars 怎么做并行?

- 计算图
- Tile：粗粒度图到细粒度图
- 调度

表达式和粗粒度图

```
import mars.tensor as mt

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

表达式和粗粒度图

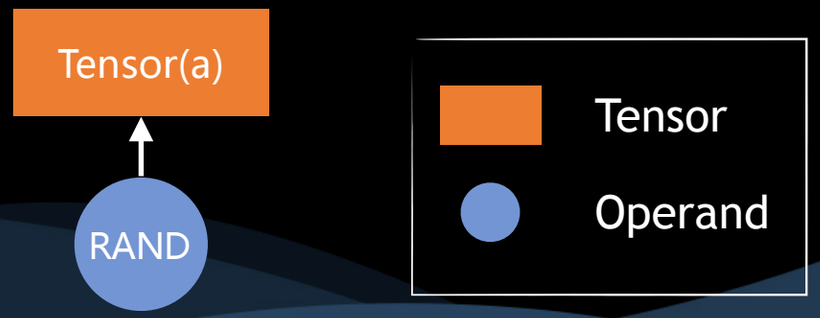
```
import mars.tensor as mt

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

Tensor

Operand

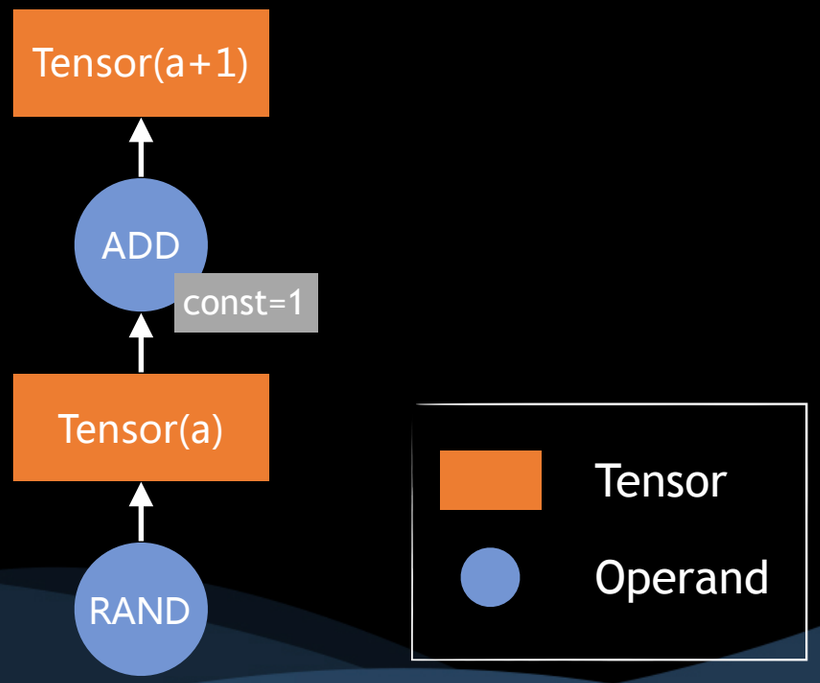
表达式和粗粒度图

```
import mars.tensor as mt

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



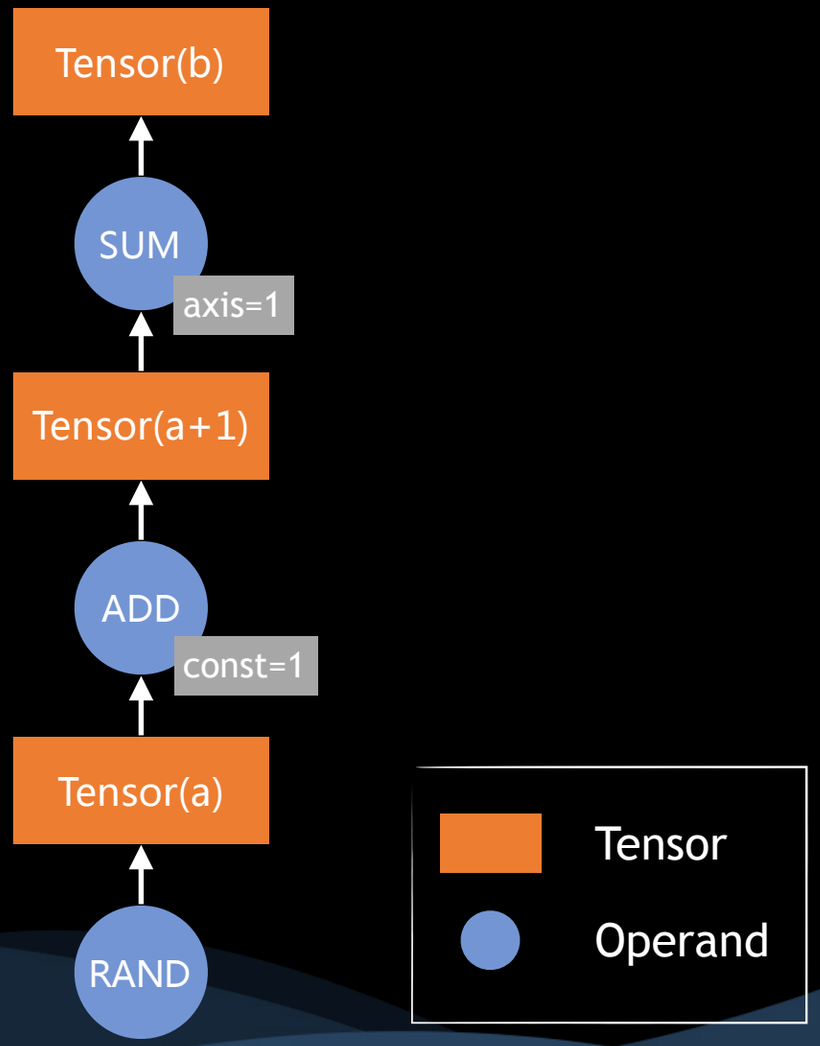
表达式和粗粒度图

```
import mars.tensor as mt

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



表达式和粗粒度图

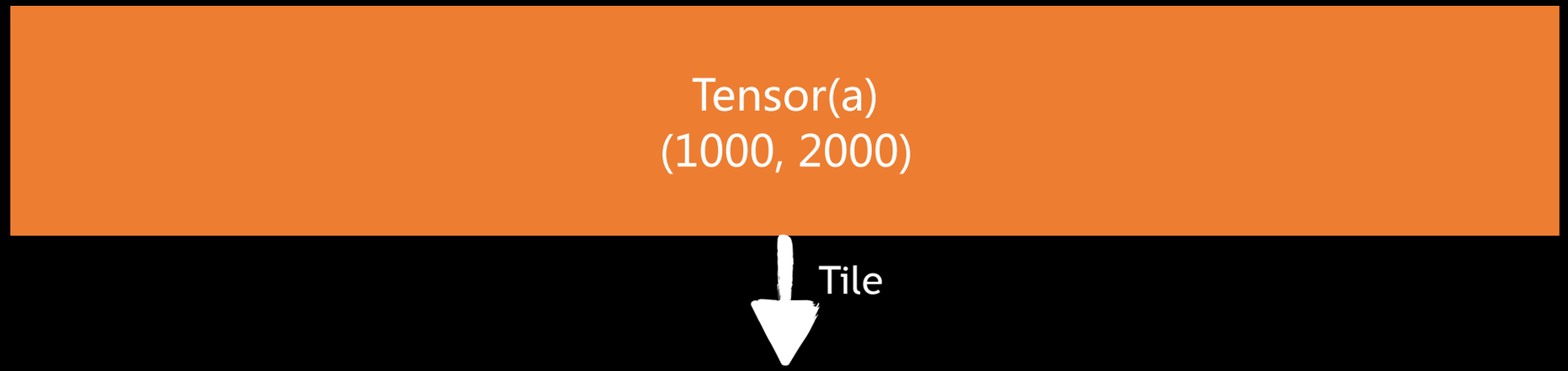
```
import mars.tensor as mt

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



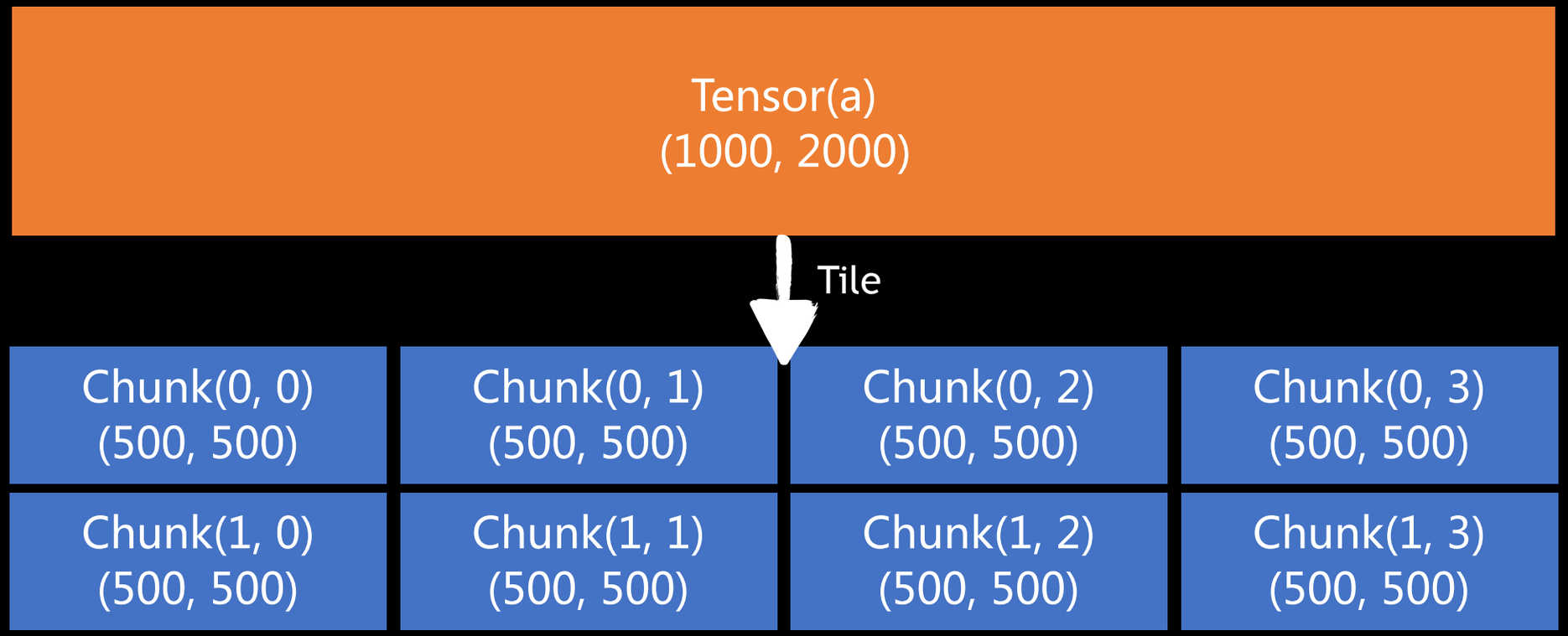
Tile：从粗粒度图到细粒度图

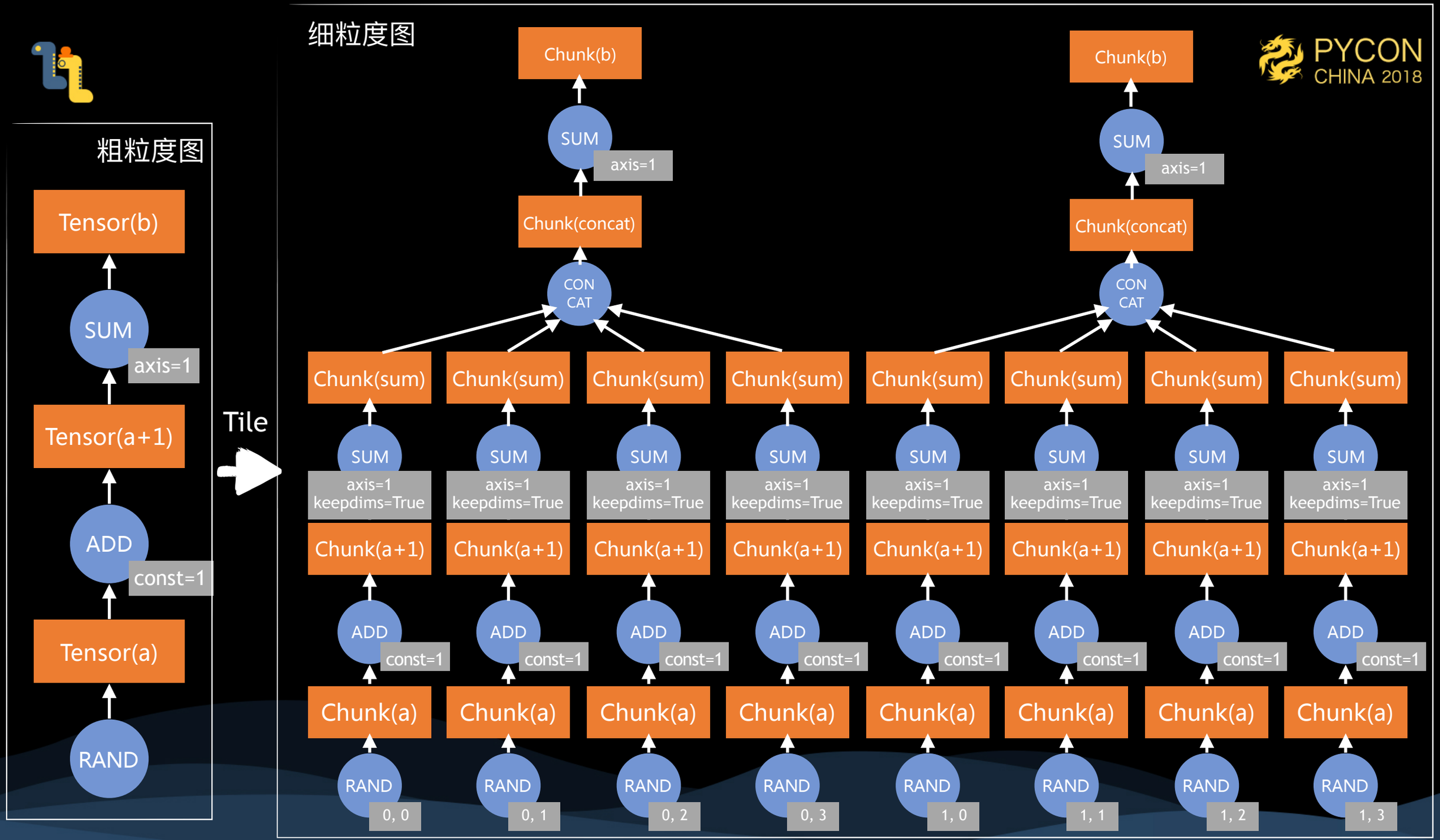
Tensor(a)
(1000, 2000)

Tile：从粗粒度图到细粒度图

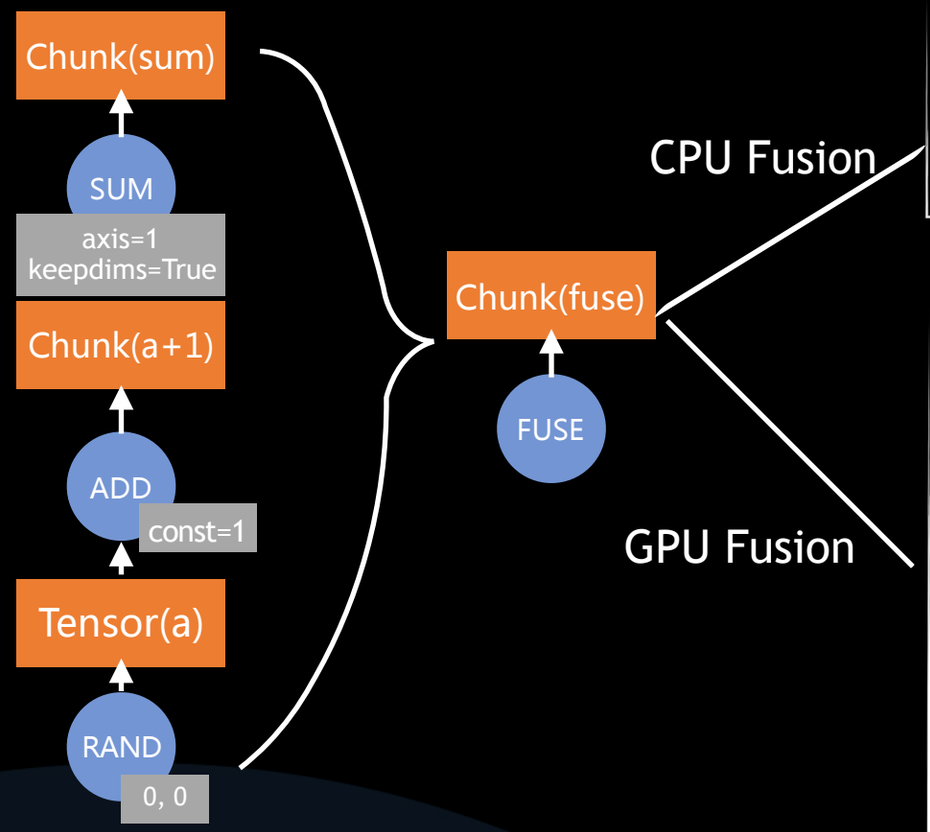


Tile：从粗粒度图到细粒度图




粗粒度图
Tensor(b)
SUM
axis=1
Tensor(a+1)
ADD
const=1
Tensor(a)
RAND
Tile
细粒度图
Chunk(b)
SUM
axis=1
Chunk(concat)
CON
CAT
Chunk(sum)
SUM
axis=1
keepdims=True
Chunk(a+1)
ADD
const=1
Chunk(a)
RAND
0, 0
0, 1
0, 2
0, 3
1, 0
1, 1
1, 2
1, 3
PYCON
CHINA 2018

Fusion 优化：合并操作



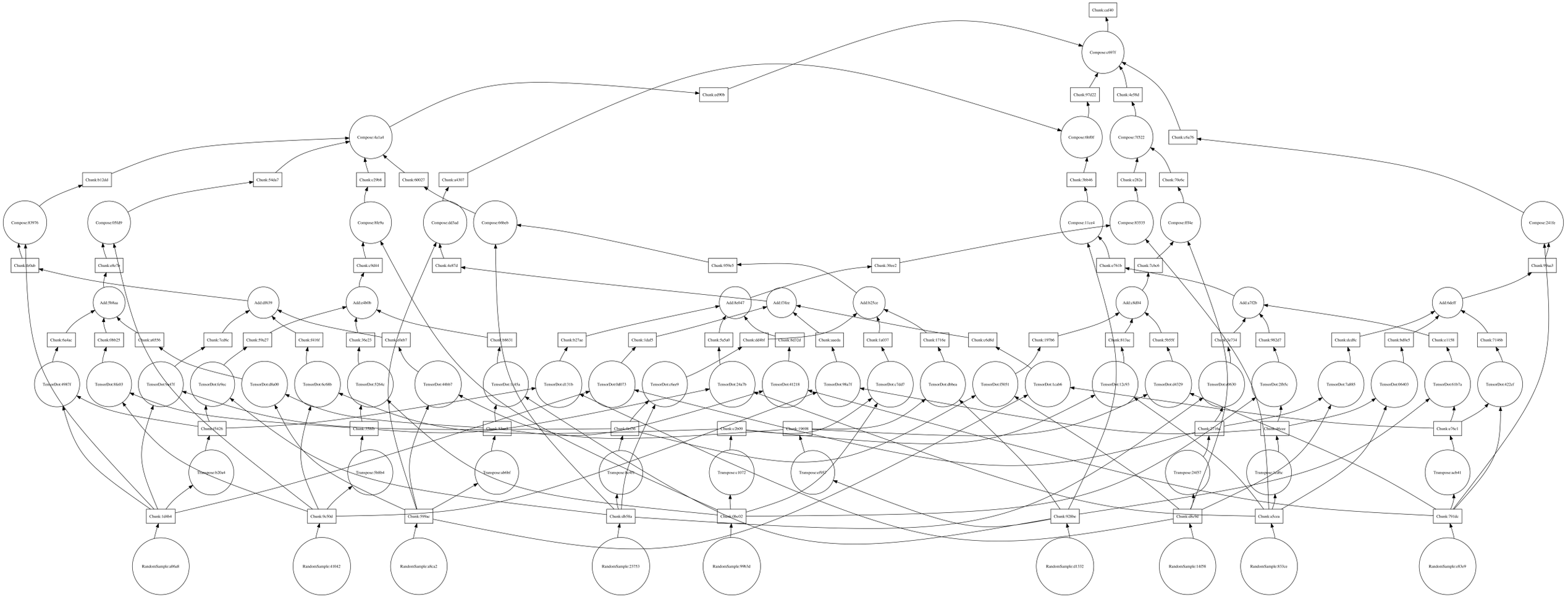
CPU Fusion

```python
import numexpr as ne

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

GPU Fusion

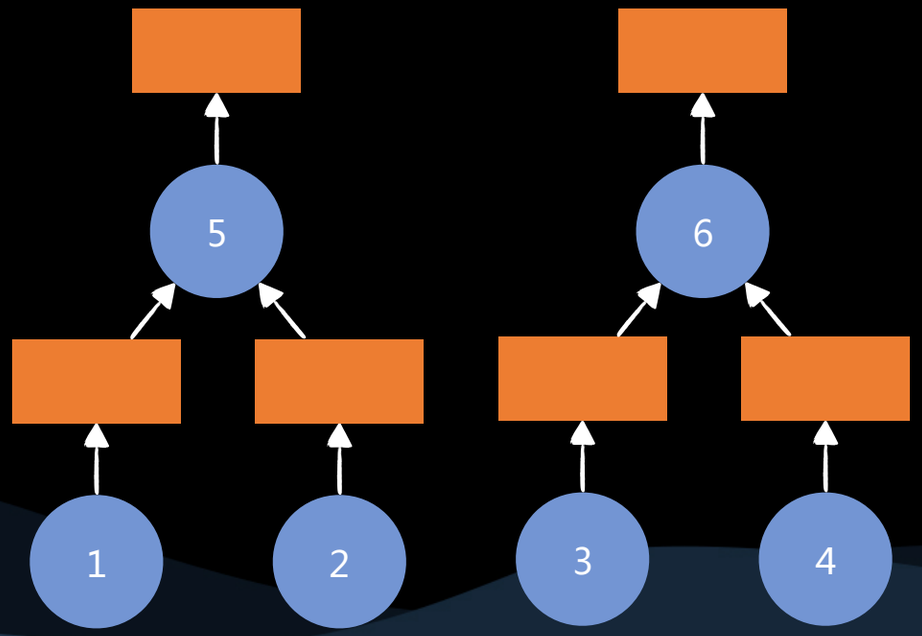
```python
import cupy as cp

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)
```

```python
import mars.tensor as mt

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



细粒度调度的要点

- 初始节点分配很重要
- 执行顺序深度优先



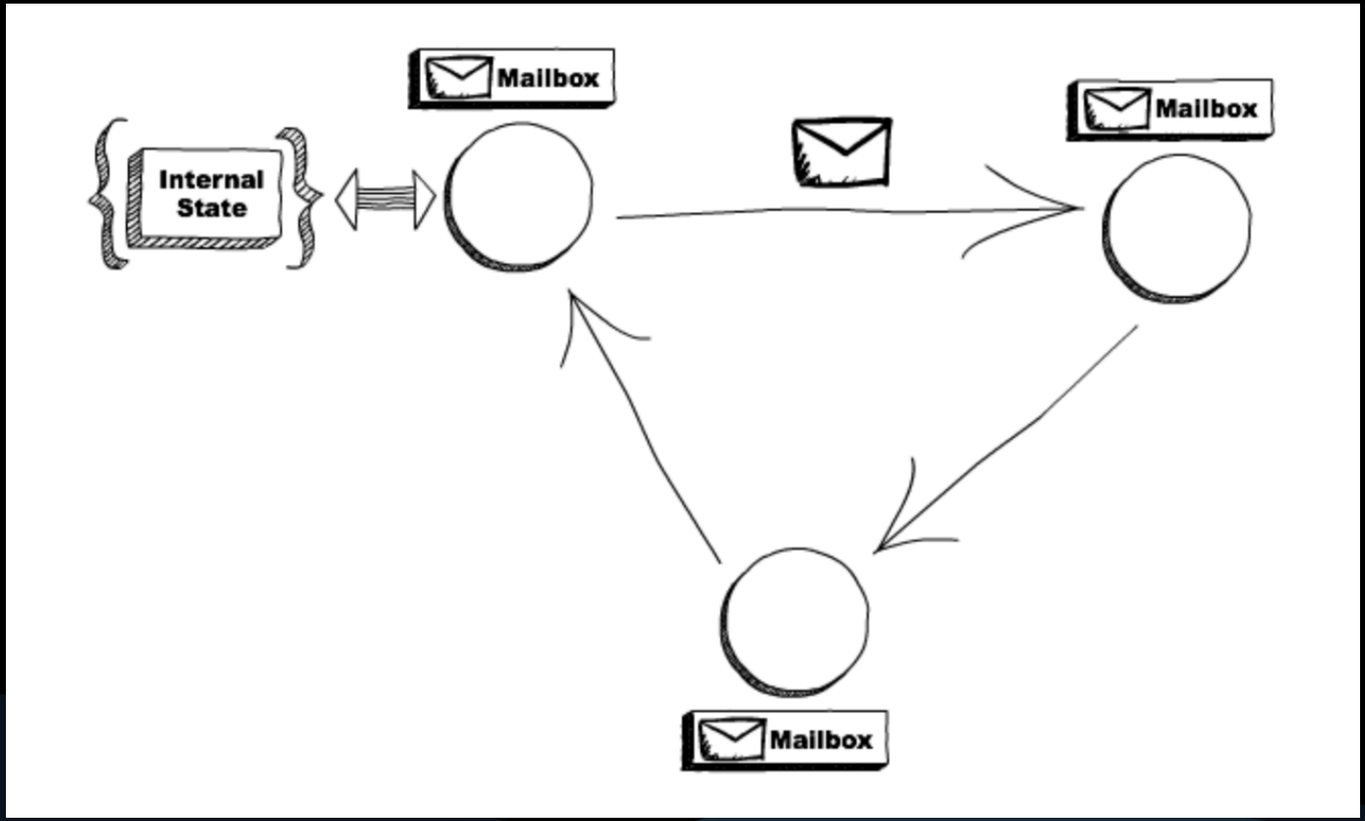
Mars 本质上是细粒度、异构图的调度系统

Mars 本质上是细粒度、异构图的调度系统

- 难点：
  - master-slave 架构，master 如何避免单点
  - worker 如何 release GIL
  - 代码如何写能防止过耦合，尤其是 master 相关代码

Actor model !!!

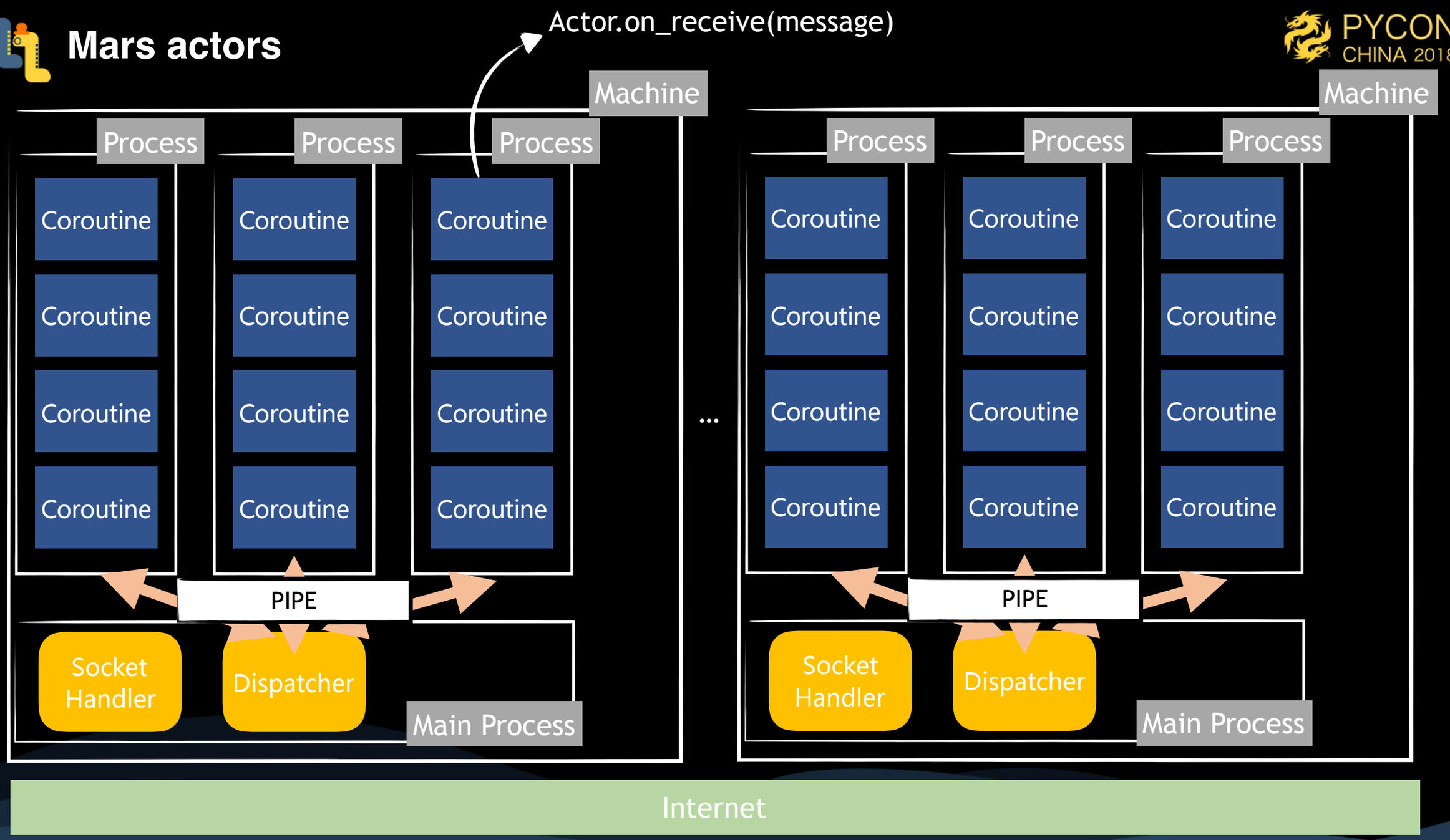
Actor model !!!



Mars actors

- 轻量 Actor 框架
- 整个 Mars 分布式的基础，Mars 分布式 Scheduler、Worker 都在 Actor 层之上

Mars actors


Actor.on_receive(message)
Machine
Process
Process
Process
Coroutine
PIPE
Socket Handler
Dispatcher
Main Process
...
Machine
Process
Process
Process
Coroutine
PIPE
Socket Handler
Dispatcher
Main Process
Internet

用 Actor 来实现蒙特卡洛计算 pi

```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)
```

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

用 Actor 来实现蒙特卡洛计算 pi

PYCON CHINA 2018

```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 类

Actor 类

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

用 Actor 来实现蒙特卡洛计算 pi

PYCON CHINA 2018

```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 类

Actor 类

create_actor 传 address 参数，可以指定创建的机器

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

用 Actor 来实现蒙特卡洛计算 pi

PYCON CHINA 2018

```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 类

Actor 类

create_actor 传 address 参数，可以指定创建的机器

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

快过多进程版本

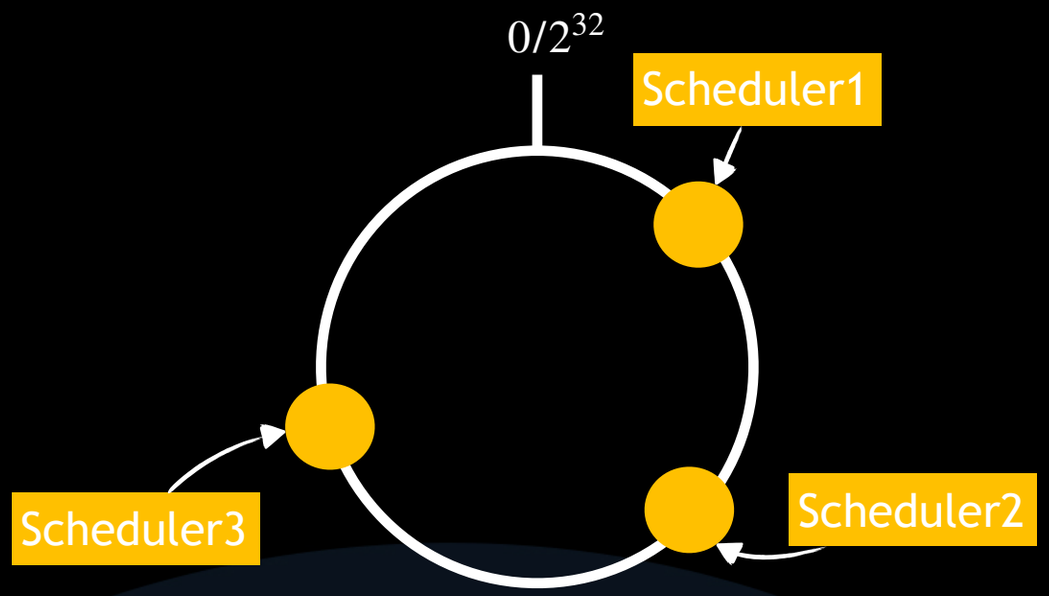
Actor 优势

- 在多核并行，release GIL
- 分布式运行变得十分容易
- 实现上由于使用 coroutine 库，如 gevent，有更好的 IO 性能
- 强迫写出逻辑清晰，职责单一的类

Mars 如何去除 Scheduler 单点?

- 一致性哈希，将 Scheduler 上的 Actor 按一致性哈希分布
- etcd：Actor 信息异步同步至 etcd

Client



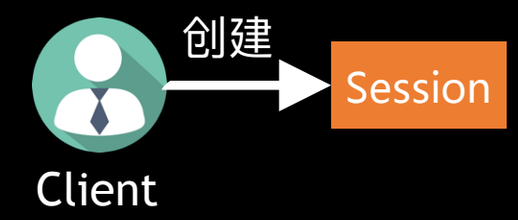
Worker1

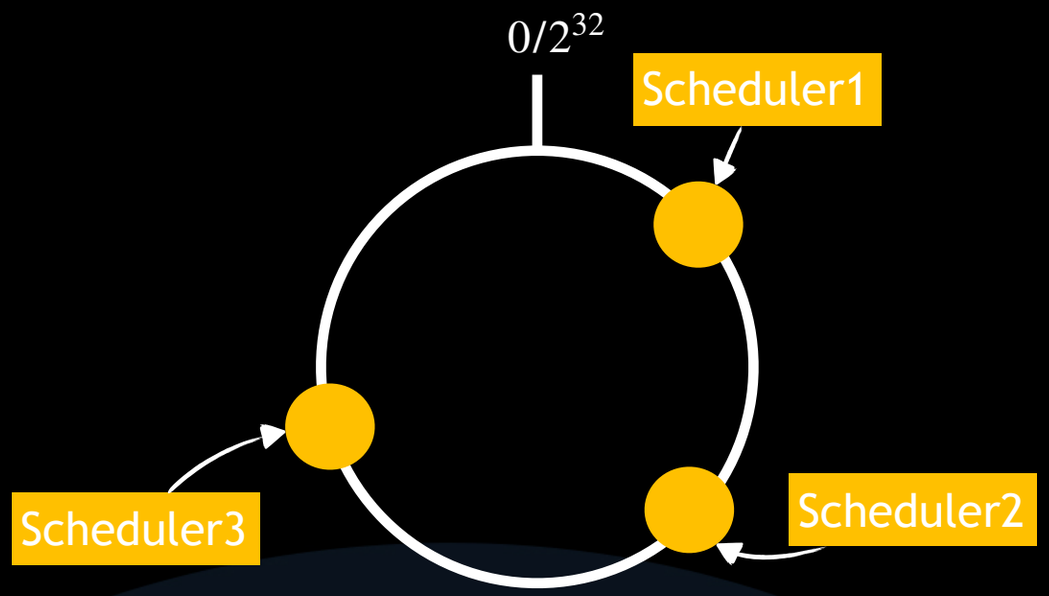
Worker2

Worker3

Worker4

Worker5





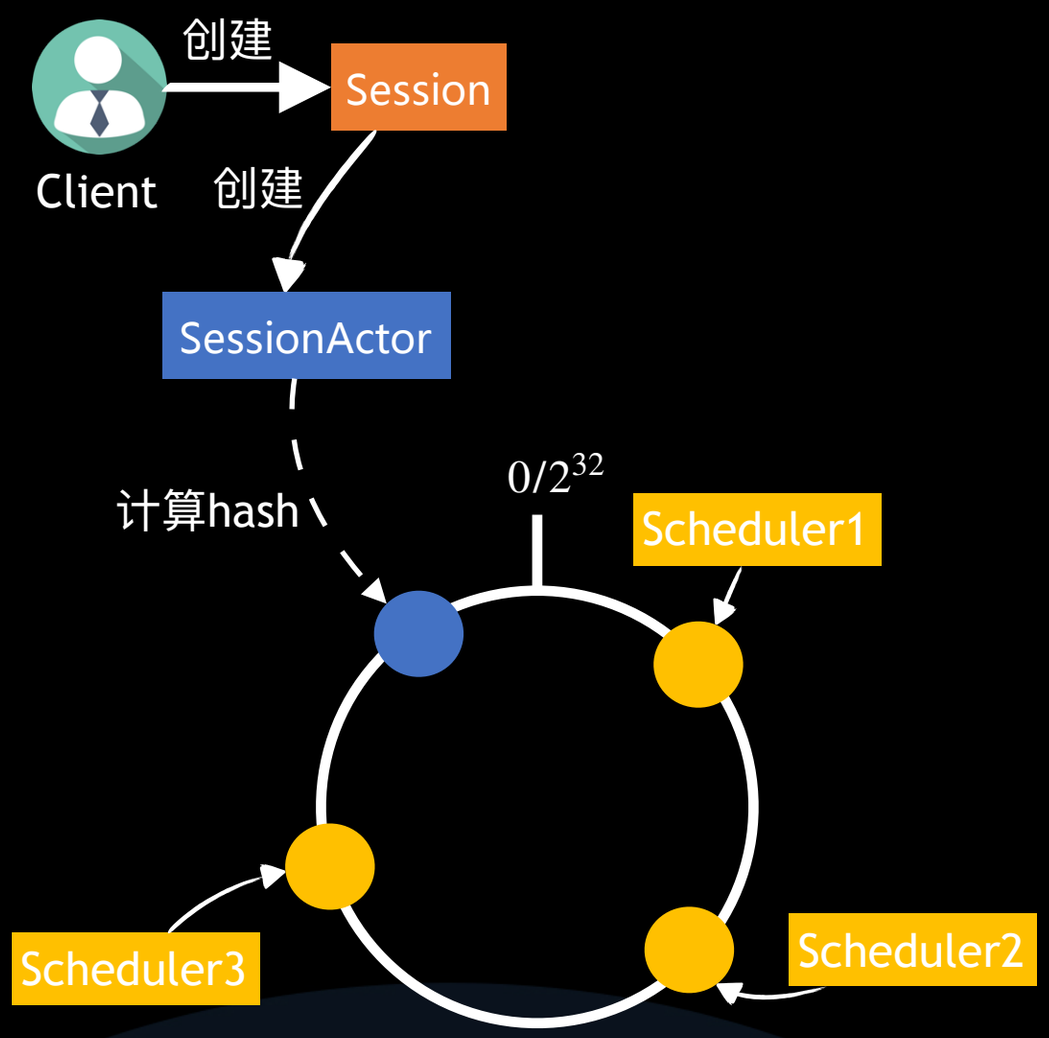
Worker1

Worker2

Worker3

Worker4

Worker5



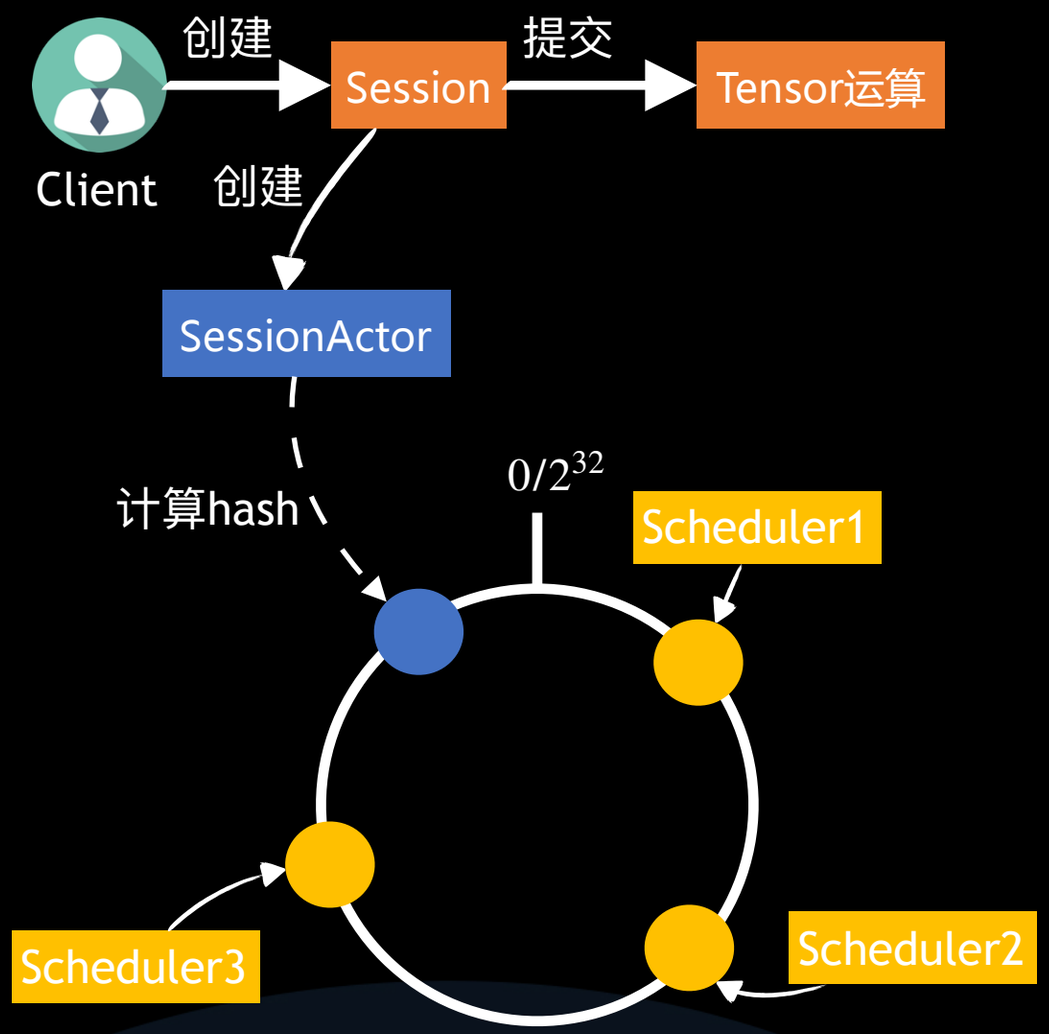
Worker1

Worker2

Worker3

Worker4

Worker5



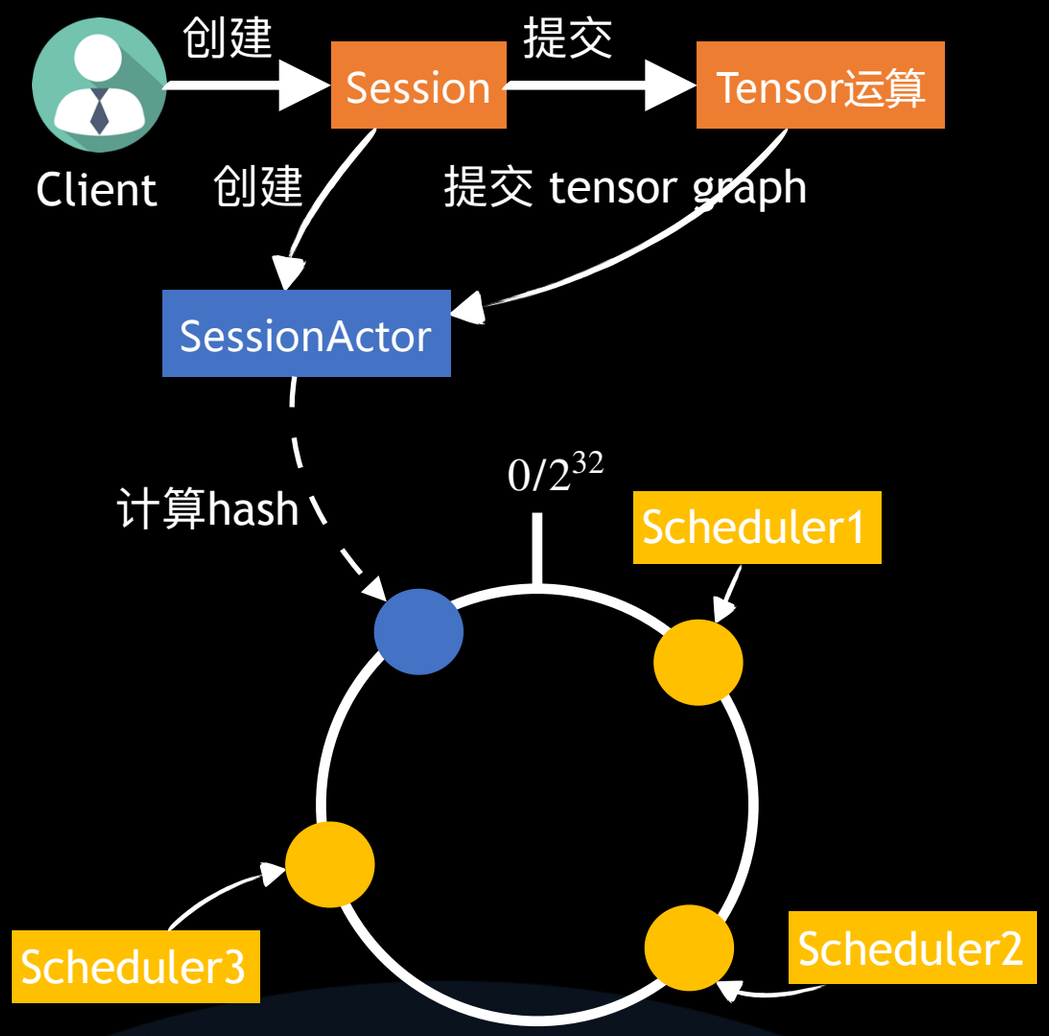
Worker1

Worker2

Worker3

Worker4

Worker5



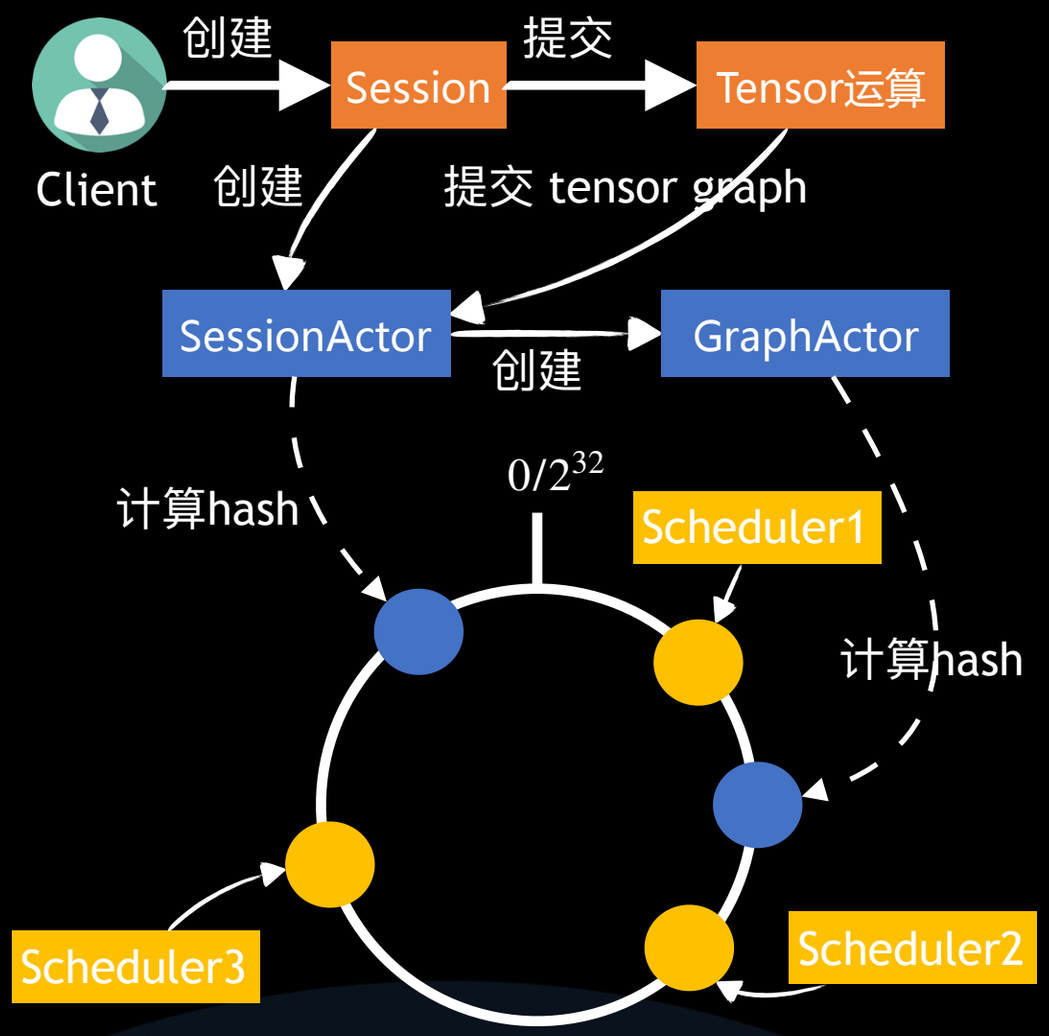
Worker1

Worker2

Worker3

Worker4

Worker5



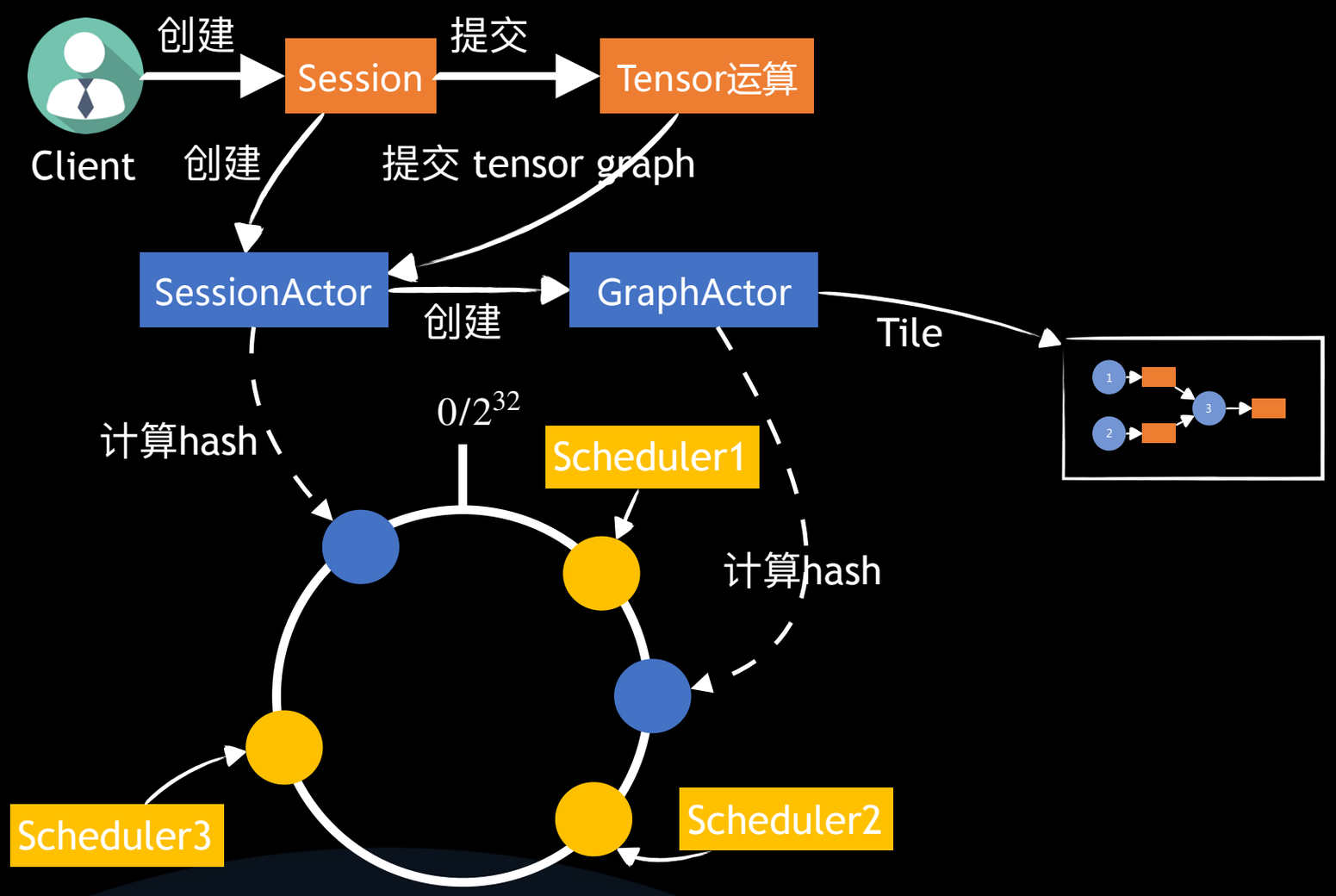
Worker1

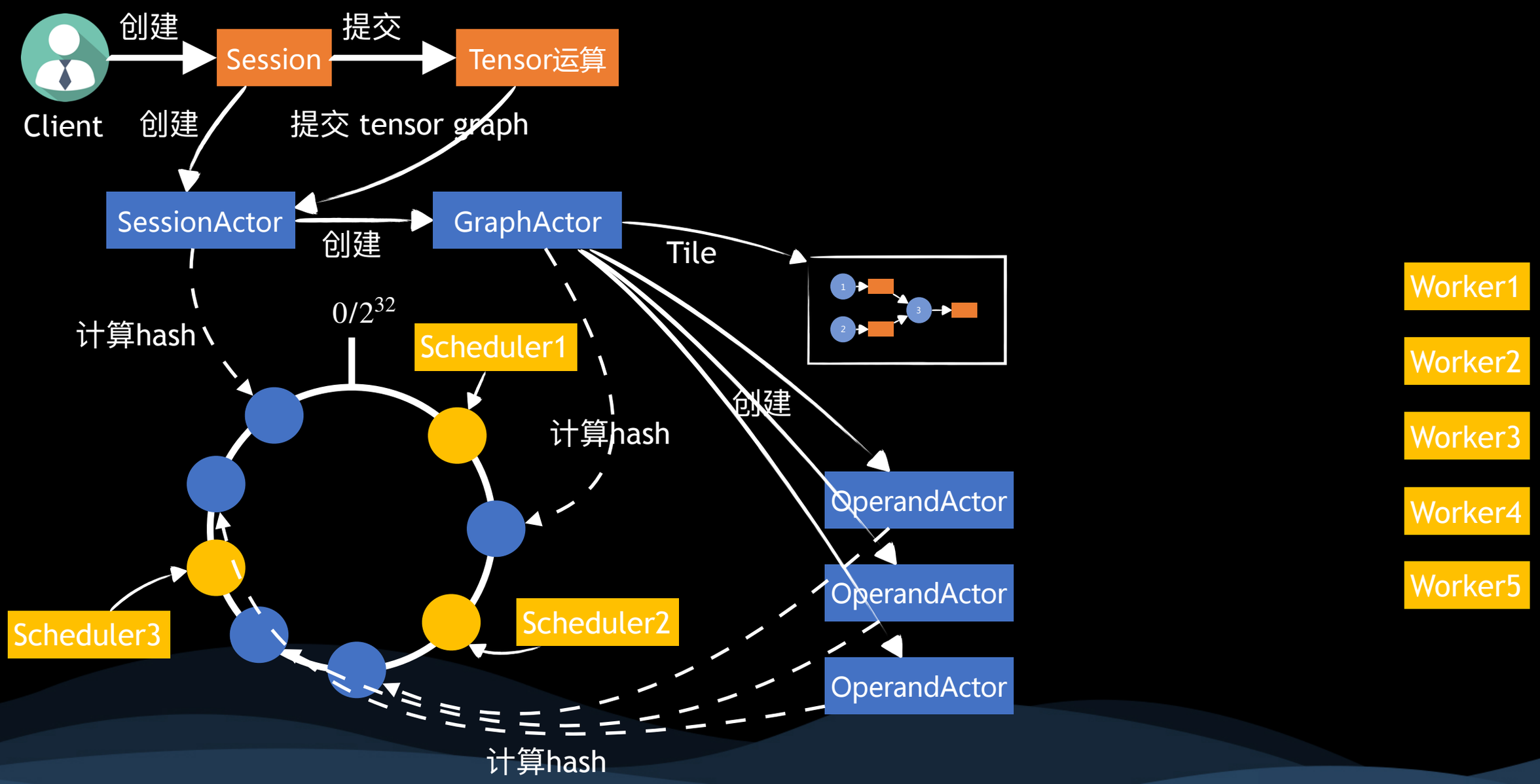
Worker2

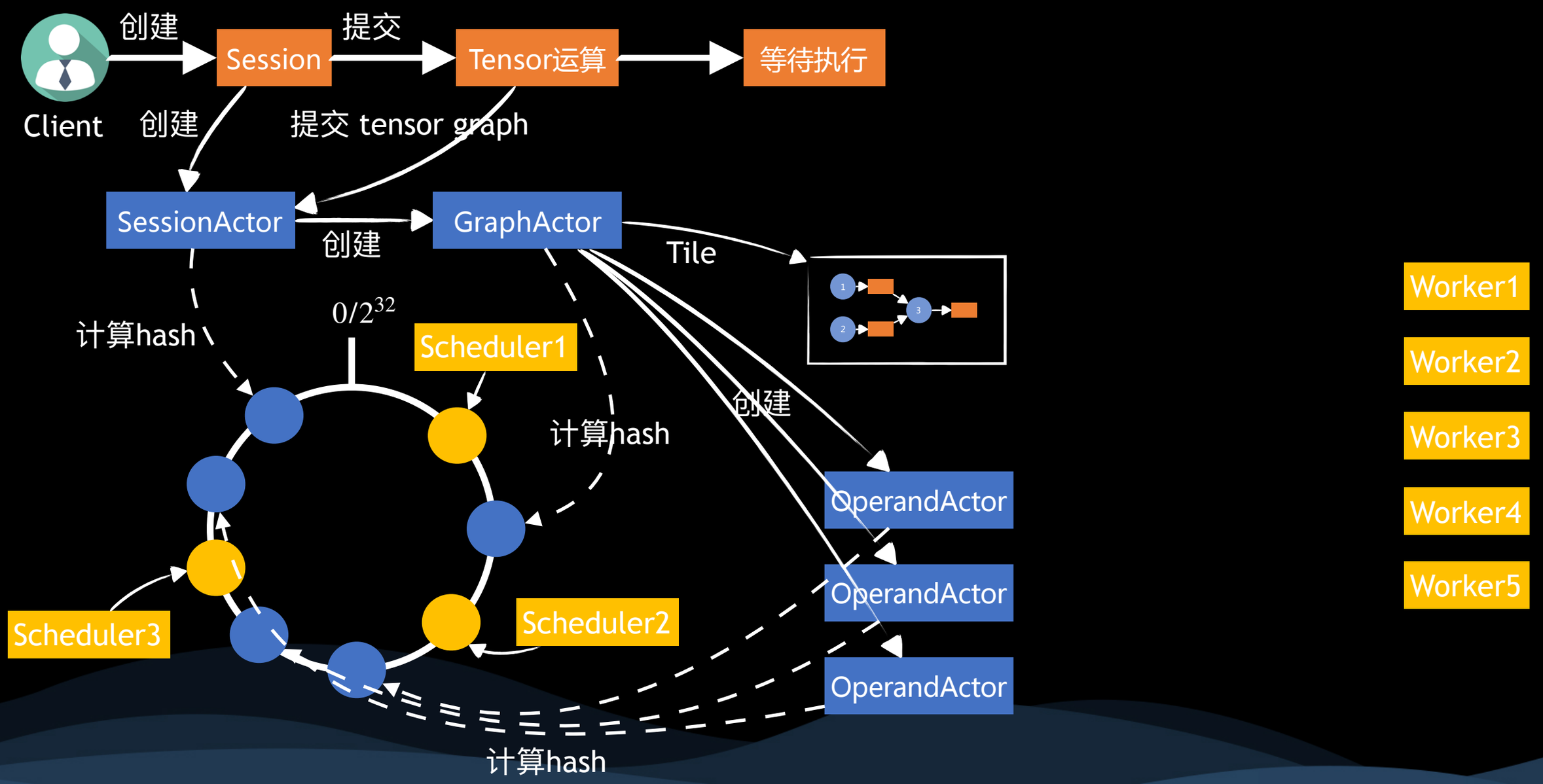
Worker3

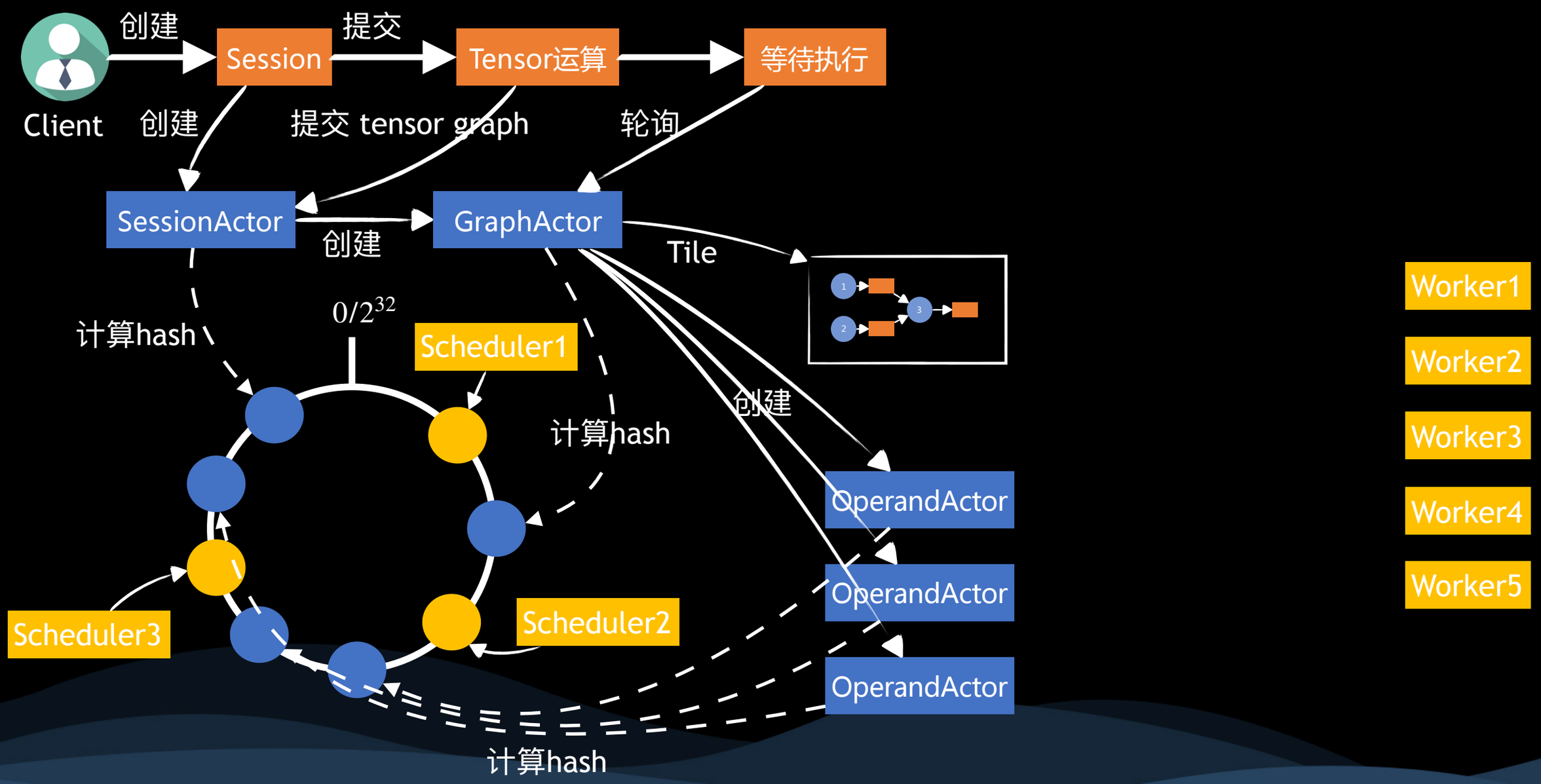
Worker4

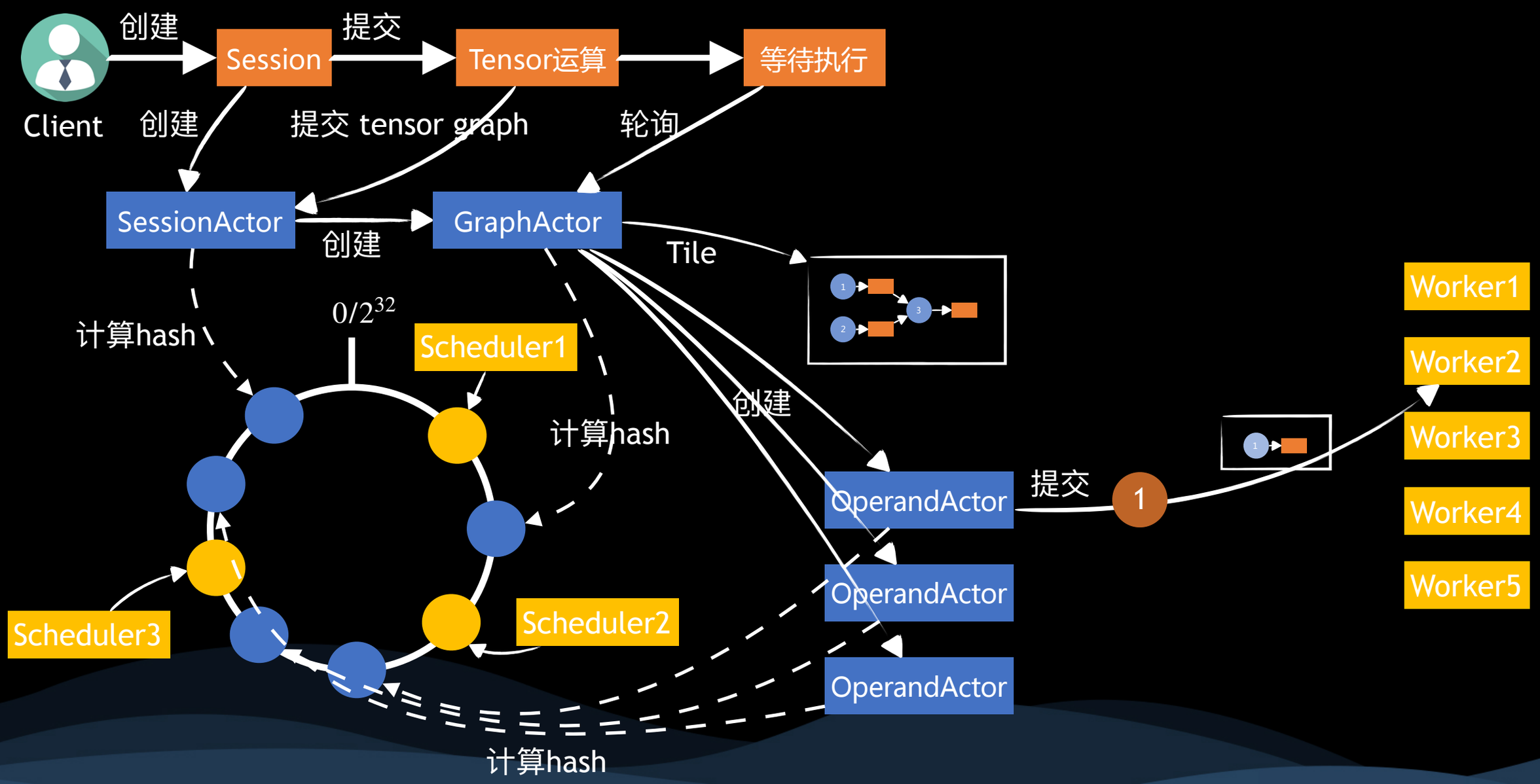
Worker5

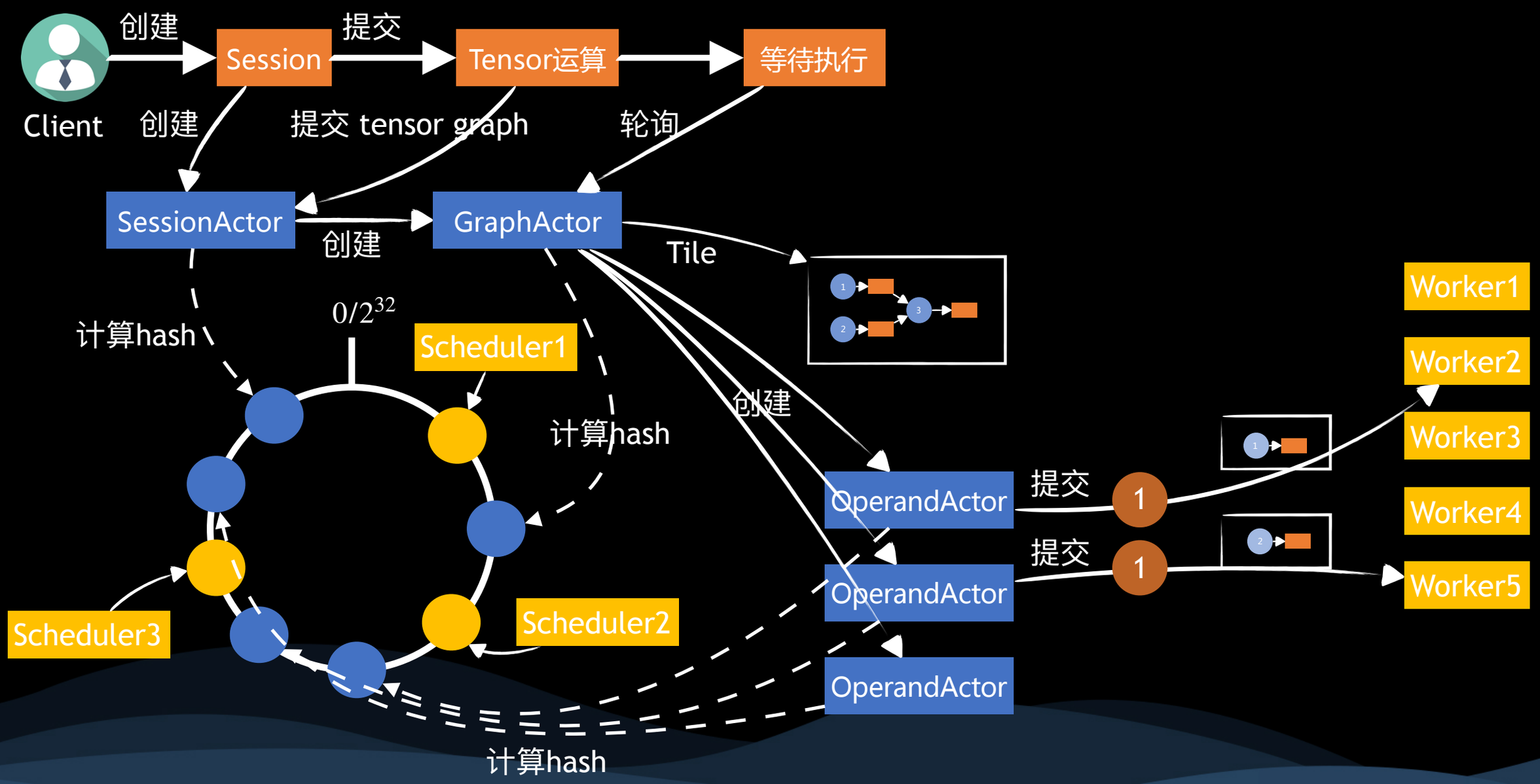

创建
提交
Session
Tensor运算
Client
创建
提交 tensor graph
SessionActor
GraphActor
创建
Tile
计算hash
0/2^32
Scheduler1
计算hash
Scheduler3
Scheduler2
Worker1
Worker2
Worker3
Worker4
Worker5

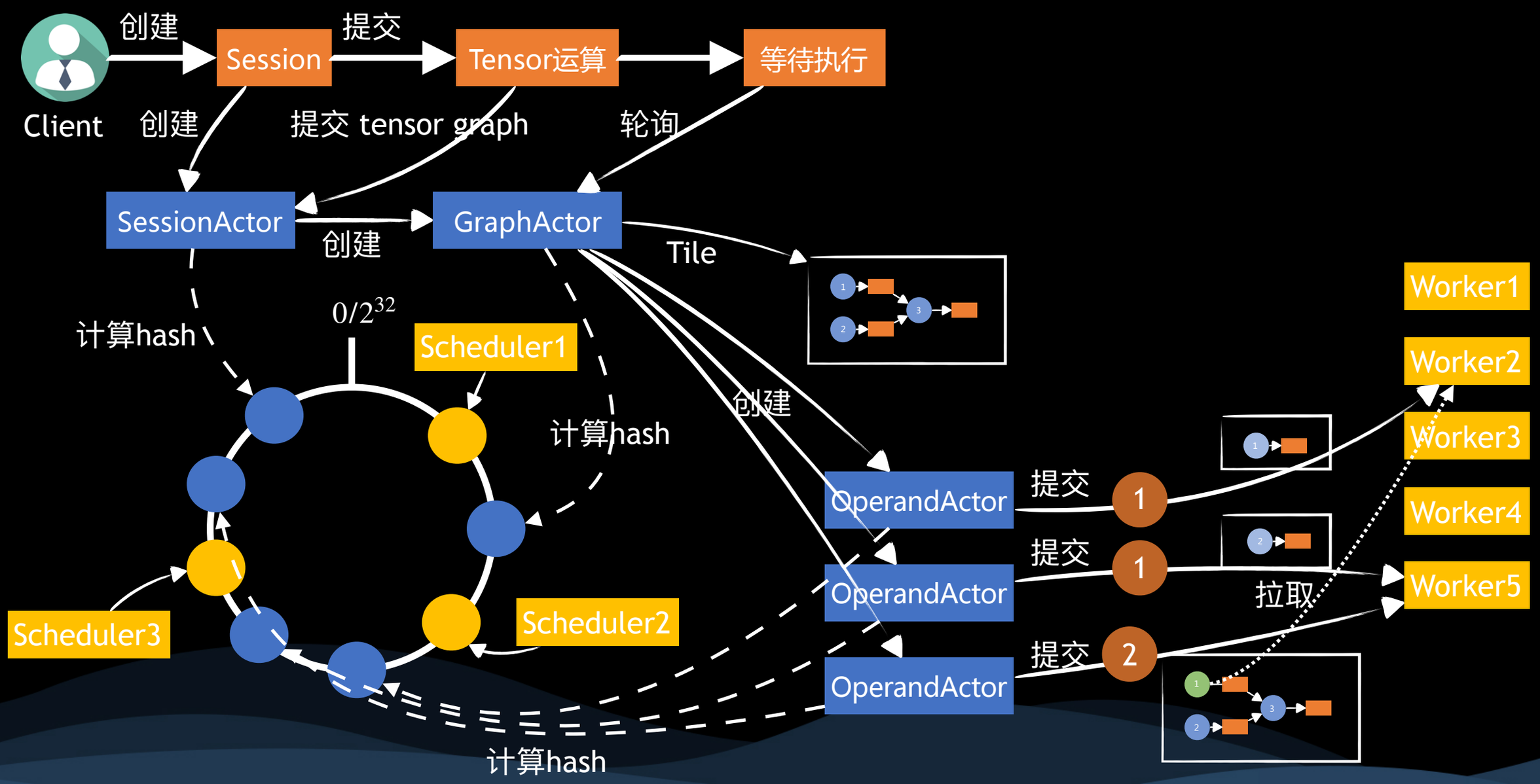

创建
提交
Client
Session
Tensor运算
创建
提交 tensor graph
SessionActor
创建
GraphActor
Tile
计算hash
0/2^32
Scheduler1
计算hash
创建
OperandActor
OperandActor
OperandActor
Scheduler3
Scheduler2
计算hash
Worker1
Worker2
Worker3
Worker4
Worker5

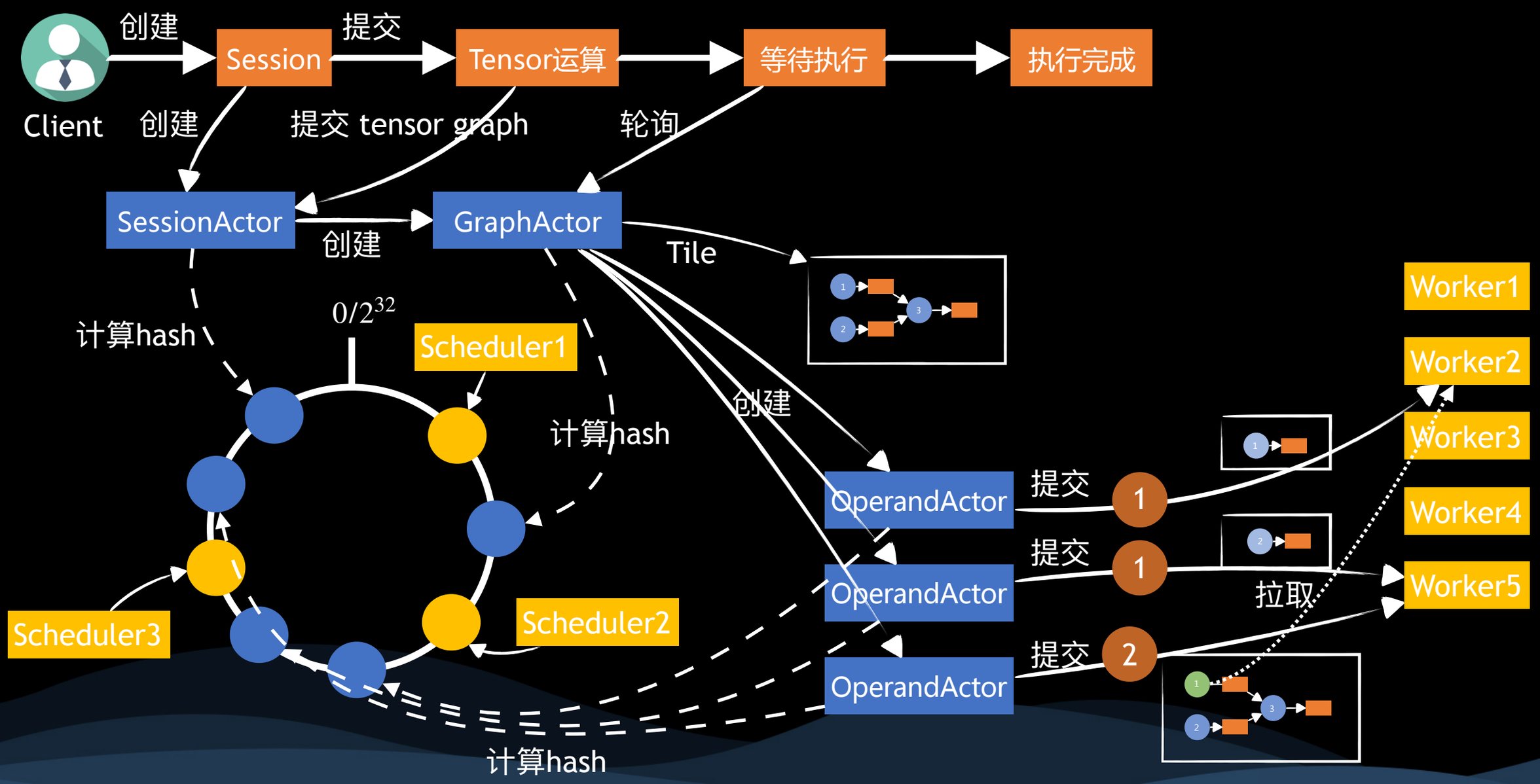

创建
提交
Session
Tensor运算
等待执行
Client
创建
提交 tensor graph
SessionActor
创建
GraphActor
Tile
计算hash
0/2^32
Scheduler1
计算hash
创建
OperandActor
OperandActor
OperandActor
Scheduler2
Scheduler3
计算hash
Worker1
Worker2
Worker3
Worker4
Worker5

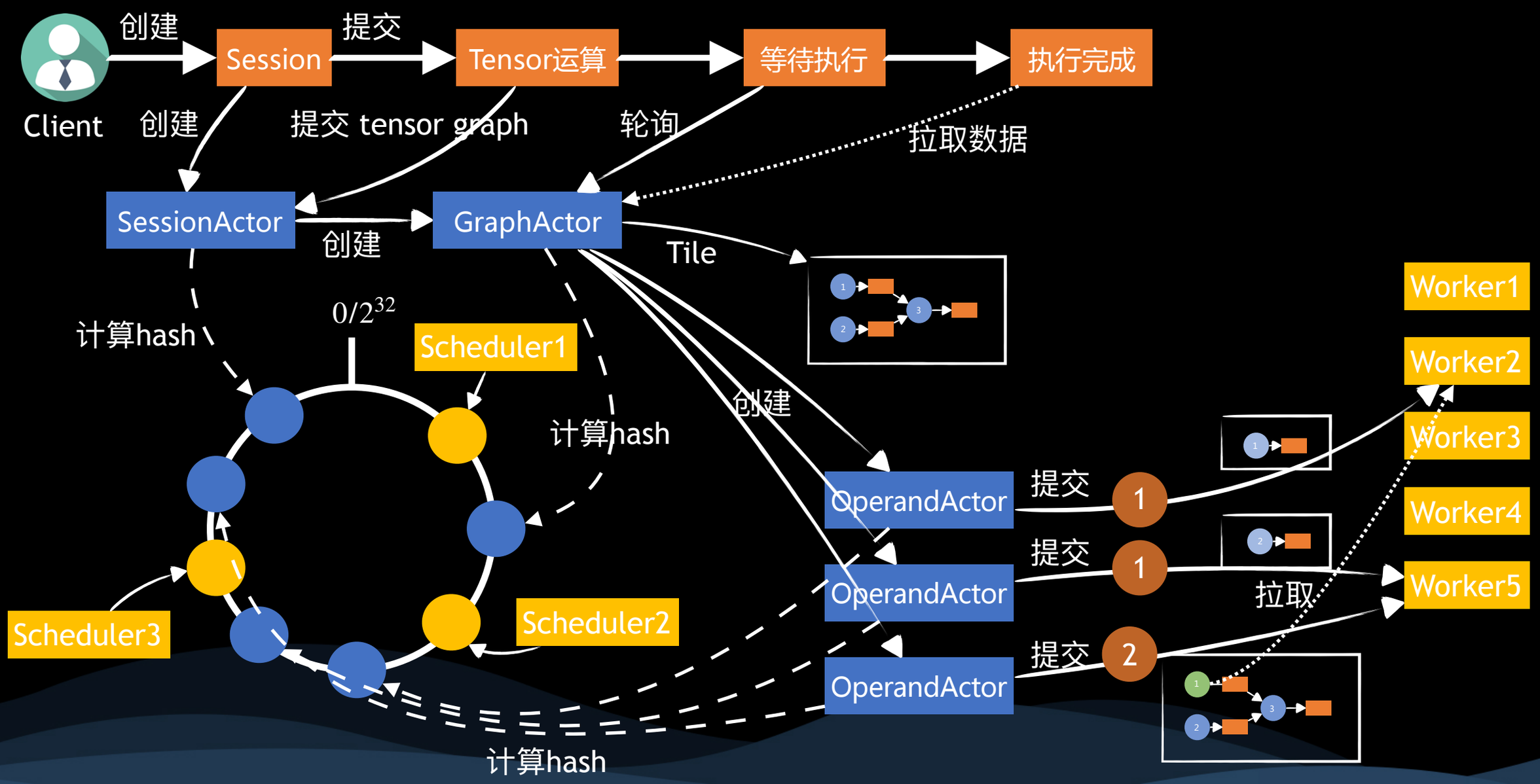

创建
提交
Session
Tensor运算
等待执行
Client
创建
提交 tensor graph
轮询
SessionActor
GraphActor
创建
Tile
计算hash
0/2^32
Scheduler1
计算hash
创建
OperandActor
OperandActor
OperandActor
Scheduler2
Scheduler3
计算hash
Worker1
Worker2
Worker3
Worker4
Worker5

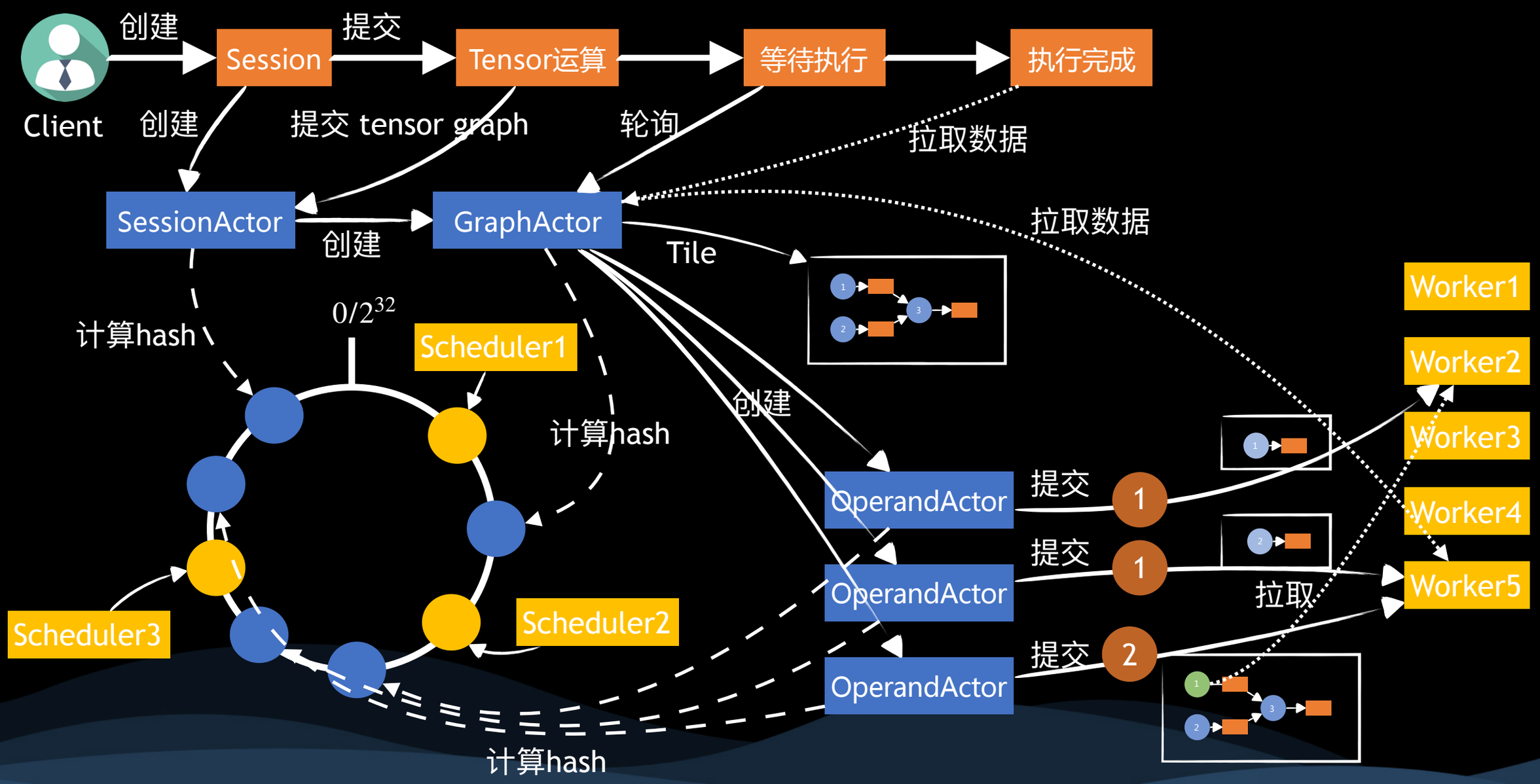

创建
提交
Client
Session
Tensor运算
等待执行
创建
提交 tensor graph
轮询
SessionActor
创建
GraphActor
Tile
计算hash
0/2^32
Scheduler1
计算hash
创建
OperandActor
提交
1
OperandActor
OperandActor
Scheduler3
Scheduler2
计算hash
Worker1
Worker2
Worker3
Worker4
Worker5

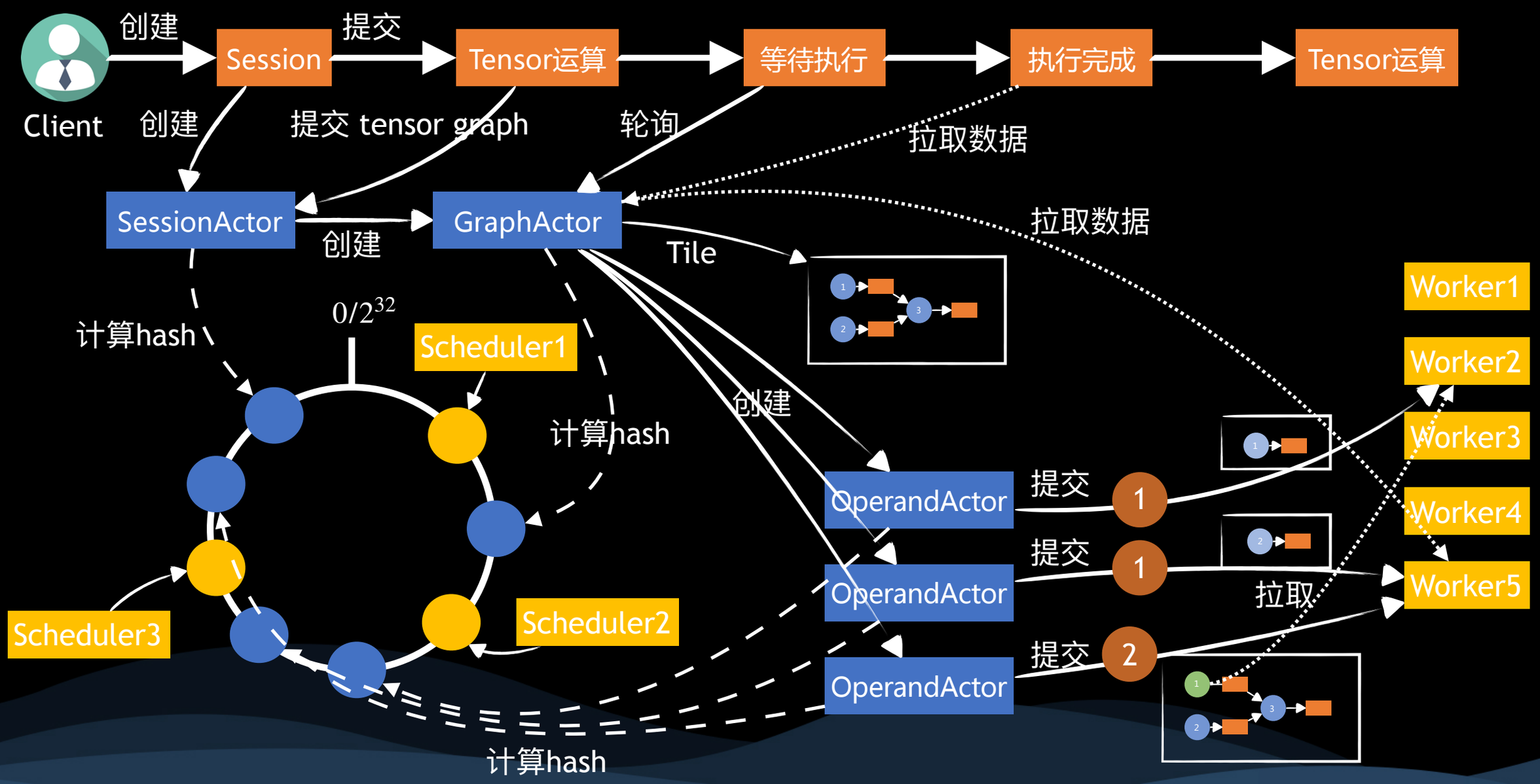

创建
提交
Client
Session
Tensor运算
等待执行
创建
提交 tensor graph
轮询
SessionActor
创建
GraphActor
Tile
计算hash
0/2^32
Scheduler1
计算hash
创建
Scheduler3
Scheduler2
OperandActor
OperandActor
OperandActor
提交
提交
1
1
Worker1
Worker2
Worker3
Worker4
Worker5
计算hash


创建
提交
Client
Session
Tensor运算
等待执行
创建
提交 tensor graph
轮询
SessionActor
创建
GraphActor
Tile
计算hash
0/2^32
Scheduler1
计算hash
创建
Scheduler3
Scheduler2
计算hash
OperandActor
OperandActor
OperandActor
提交
提交
提交
1
1
2
拉取
Worker1
Worker2
Worker3
Worker4
Worker5

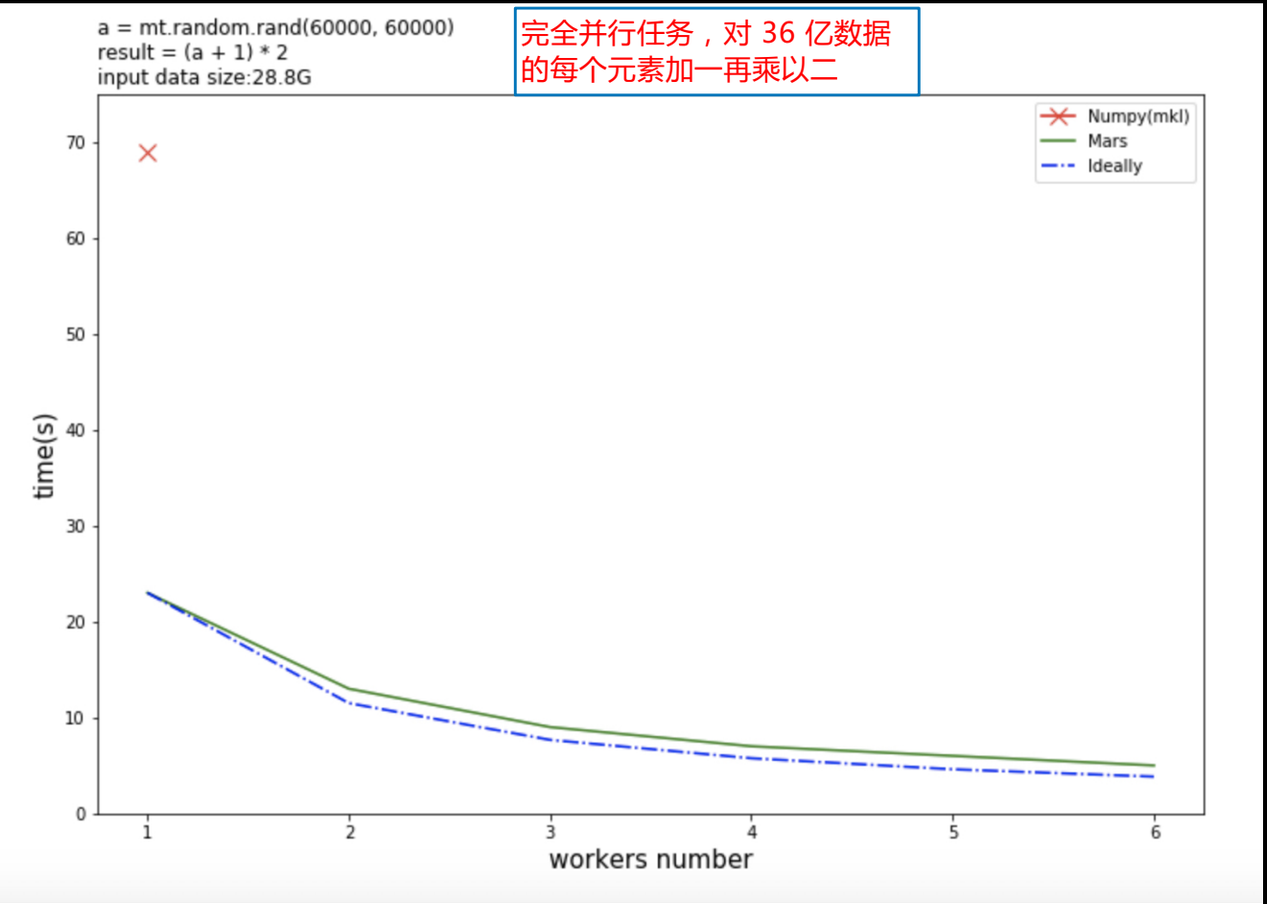

创建
提交
Client
Session
Tensor运算
等待执行
执行完成
创建
提交 tensor graph
轮询
SessionActor
创建
GraphActor
Tile
计算hash
0/2^32
Scheduler1
计算hash
创建
Scheduler3
Scheduler2
计算hash
OperandActor
OperandActor
OperandActor
提交
提交
提交
1
1
2
拉取
Worker1
Worker2
Worker3
Worker4
Worker5

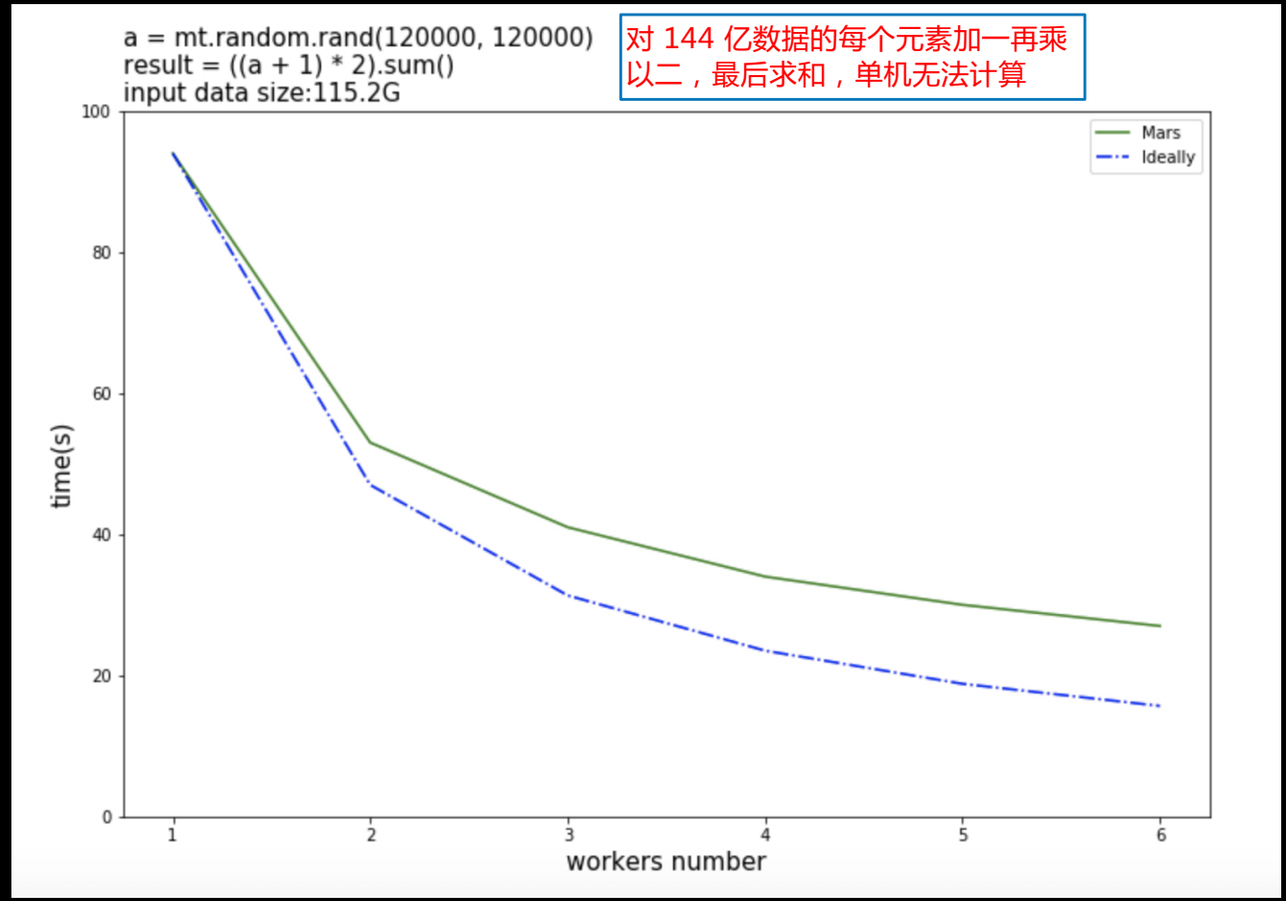

Client
创建
Session
提交
Tensor运算
等待执行
执行完成
创建
提交 tensor graph
轮询
拉取数据
SessionActor
创建
GraphActor
Tile
计算hash
0/2^32
Scheduler1
计算hash
创建
Scheduler2
Scheduler3
计算hash
OperandActor
OperandActor
OperandActor
提交
提交
提交
1
1
2
拉取
Worker1
Worker2
Worker3
Worker4
Worker5


创建
提交
Client
Session
Tensor运算
等待执行
执行完成
创建
提交 tensor graph
轮询
拉取数据
SessionActor
GraphActor
创建
拉取数据
Tile
计算hash
0/2^32
Scheduler1
计算hash
创建
Scheduler3
Scheduler2
计算hash
OperandActor
OperandActor
OperandActor
提交
提交
提交
1
1
2
拉取
Worker1
Worker2
Worker3
Worker4
Worker5


创建
Session
提交
Tensor运算
等待执行
执行完成
Tensor运算
Client
创建
提交 tensor graph
轮询
拉取数据
SessionActor
创建
GraphActor
拉取数据
Tile
计算hash
0/2^32
Scheduler1
计算hash
创建
Scheduler3
Scheduler2
计算hash
OperandActor
OperandActor
OperandActor
提交
提交
提交
1
1
2
拉取
Worker1
Worker2
Worker3
Worker4
Worker5

性能


a = mt.random.rand(60000, 60000)
result = (a + 1) * 2
input data size:28.8G
完全并行任务，对 36 亿数据的每个元素加一再乘以二
Numpy(mkl)
Mars
Ideally
time(s)
workers number


a = mt.random.rand(120000, 120000)
result = ((a + 1) * 2).sum()
input data size:115.2G
对 144 亿数据的每个元素加一再乘以二，最后求和，单机无法计算
Mars
Ideally
time(s)
workers number

Mars

- 开放源代码
- 我们的征途是星辰大海
- 欢迎大家贡献代码，使用，提 issue，共建 Mars

CONTENTS | 目录

- 背景
- 一个例子
- Mars 核心
- Mars 原理
- 未来计划

更好的 Mars

- 循环的支持，eager mode
- Tensor：支持更多 Numpy 和 Scipy 接口
- DataFrame：100% 兼容 pandas API，基于 tensor，支持 GPU 计算
- 机器学习：提供 scikit-learn 接口
- 提供更灵活的用户自定义代码：函数和类的调度
- 多语言支持

PYCON CHINA 2018

THANK YOU

- Github: https://github.com/chineking
- Blog: http://qinxuye.me