2019 Python开发者日
2019年4月13-14日 · 北京
让开发者紧跟技术潮流

CSDN

用Mars来并行和分布式执行Numpy

秦续业

2019.4.13

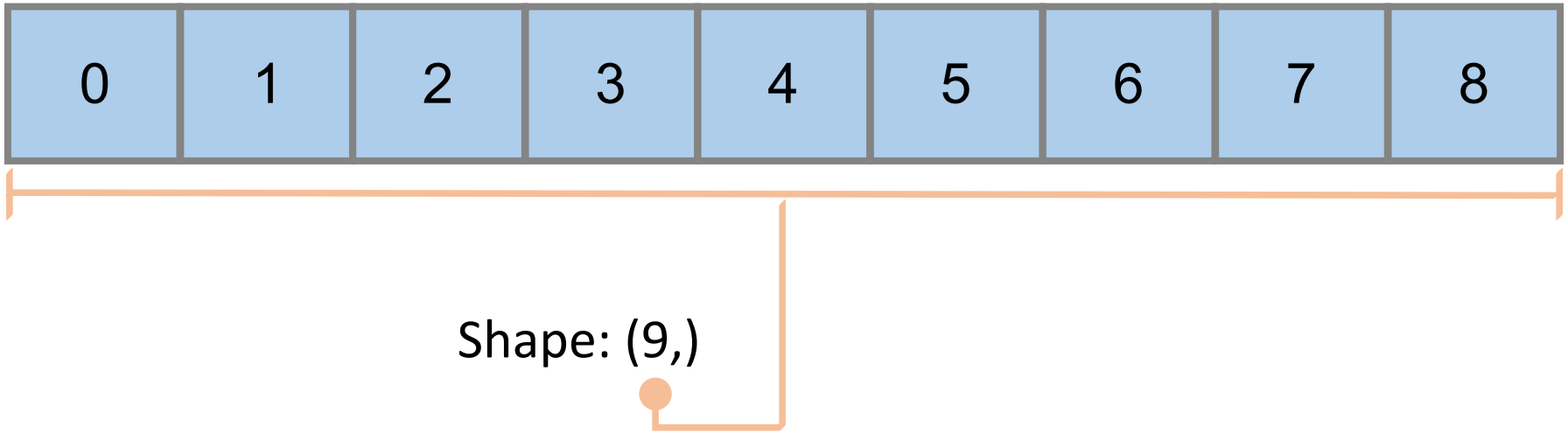
关于我

- 阿里巴巴技术专家
- 主要工作是结合大数据与 Python 语言
- 最近作为 Mars 项目的核心开发者和架构师 Focus 在 Mars 项目
- 曾经开发过 PyODPS（pandas-like API，支持在大数据平台、数据库以及本地执行）和 cola

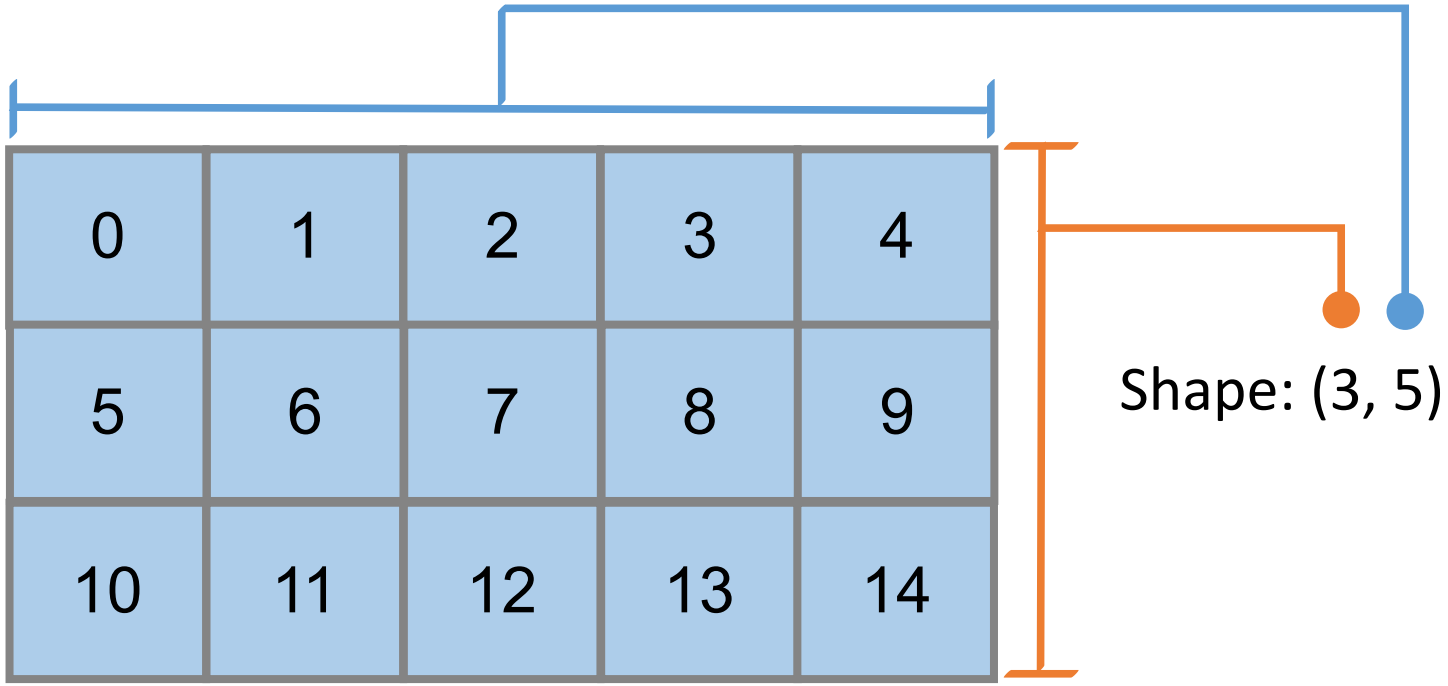
- ndarray
- 对整组数据快速运算的标准数学函数（无需编写循环）
- 读写磁盘数据的工具和操作内存映射文件的工具
- 线性代数、随机数生成和傅里叶变换

NumPy

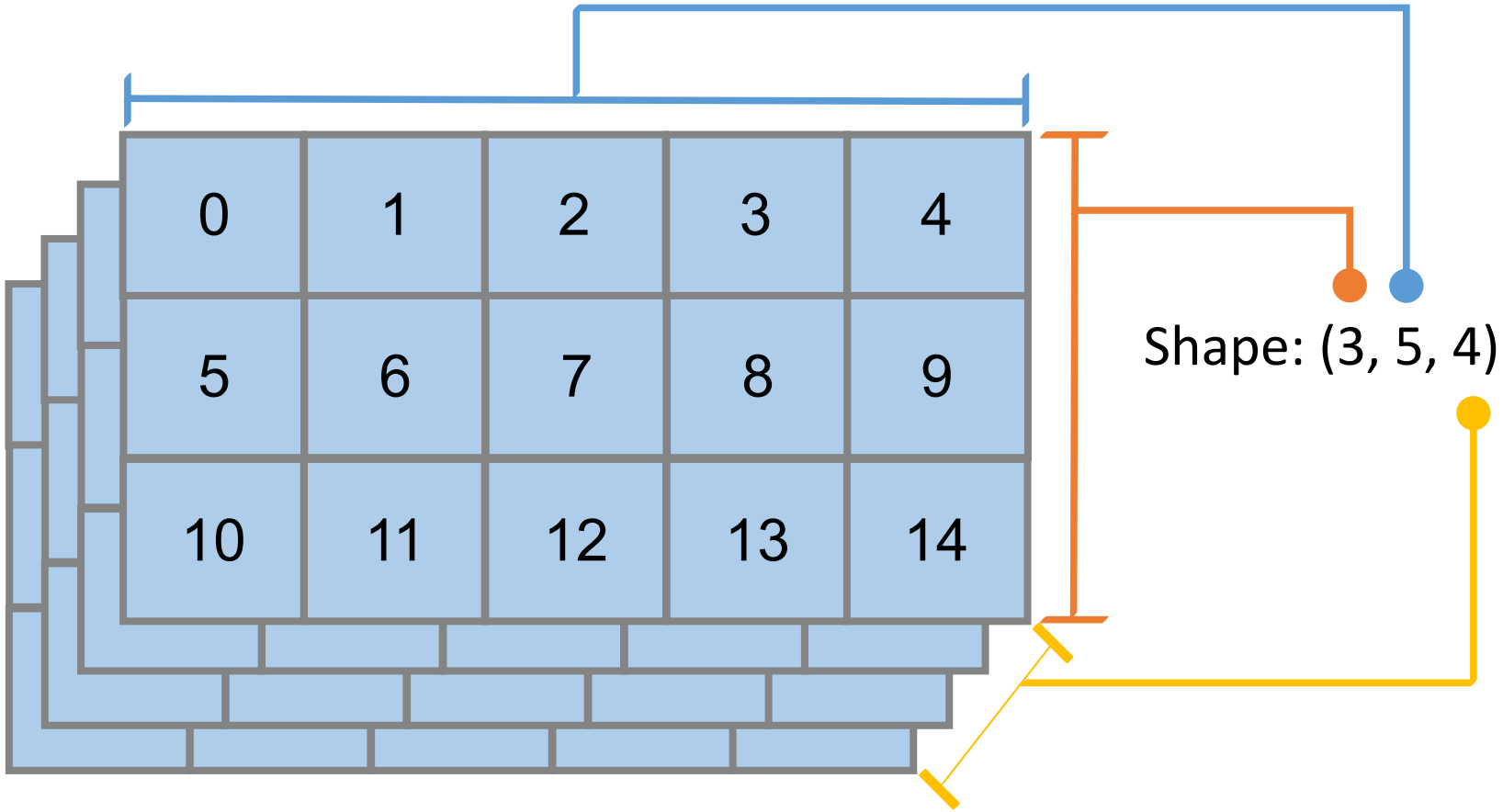
1维数组



2维数组



3维数组



创建ndarray

- 从Python数组创建

- 范围

创建ndarray

- 从Python数组创建

```
In [2]: np.array([1, 2, 3, 4])
Out[2]: array([1, 2, 3, 4])
```

- 范围

创建ndarray

- 从Python数组创建

```
In [2]: np.array([1, 2, 3, 4])
Out[2]: array([1, 2, 3, 4])
```

```
In [3]: np.array([1, 2, 3, 4], dtype=float)
Out[3]: array([1., 2., 3., 4.])
```

- 范围

创建ndarray

- 从Python数组创建

```
In [2]: np.array([1, 2, 3, 4])
Out[2]: array([1, 2, 3, 4])
```

```
In [3]: np.array([1, 2, 3, 4], dtype=float)
Out[3]: array([1., 2., 3., 4.])
```

- 范围

```
In [4]: np.arange(10)
Out[4]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [5]: np.arange('2019-04-01', '2019-04-13', dtype='datetime64')
Out[5]:
array(['2019-04-01', '2019-04-02', '2019-04-03', '2019-04-04',
       '2019-04-05', '2019-04-06', '2019-04-07', '2019-04-08',
       '2019-04-09', '2019-04-10', '2019-04-11', '2019-04-12'],
      dtype='datetime64[D]')

In [6]: np.linspace(0, 1, 5)
Out[6]: array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

创建ndarray

- 全0、全1、空数组
- 矩阵

创建ndarray

- 全0、全1、空数组

```
In [8]: np.zeros((2, 1, 3))
Out[8]:
array([[[0., 0., 0.]],

       [[0., 0., 0.]]])

In [10]: np.ones((1, 4))
Out[10]: array([[1., 1., 1., 1.]])

In [11]: np.empty((3, 1))
Out[11]:
array([[-1.49166815e-154],
       [ 1.73059759e-077],
       [ 2.96439388e-323]])
```

- 矩阵

创建ndarray

- 全0、全1、空数组

```
In [8]: np.zeros((2, 1, 3))
Out[8]:
array([[[0., 0., 0.]],

       [[0., 0., 0.]]])

In [10]: np.ones((1, 4))
Out[10]: array([[1., 1., 1., 1.]])

In [11]: np.empty((3, 1))
Out[11]:
array([[-1.49166815e-154],
       [ 1.73059759e-077],
       [ 2.96439388e-323]])
```

- 矩阵

```
In [12]: np.eye(3)
Out[12]:
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])

In [13]: np.diag([1, 2, 3, 4])
Out[13]:
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])

In [14]: np.diag(np.ones((3, 4)))
Out[14]: array([1., 1., 1.])

In [15]: np.triu(np.ones((3, 4)))
Out[15]:
array([[1., 1., 1., 1.],
       [0., 1., 1., 1.],
       [0., 0., 1., 1.]])
```

索引与切片

- 切片

索引与切片

- 切片

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

索引与切片

- 切片

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

arr[0:2, :]

索引与切片

- 切片

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

arr[0:2, :]

arr[2, 1:]

索引与切片

- 切片

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

arr[0:2, :]

arr[2, 1:]

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

索引与切片

- 切片

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

arr[0:2, :]

arr[2, 1:]

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

arr[::2, ::3]

索引与切片

- 布尔索引
- 花式索引
- 省略号和增加维度

索引与切片

- 布尔索引

```
In [16]: a = np.arange(15).reshape(3, 5)

In [17]: a
Out[17]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])

In [18]: b = a % 3 == 0

In [19]: b
Out[19]:
array([[ True, False, False,  True, False],
       [False,  True, False, False,  True],
       [False, False,  True, False, False]])

In [20]: a[b]
Out[20]: array([ 0,  3,  6,  9, 12])
```

- 花式索引

- 省略号和增加维度

索引与切片

- 布尔索引

```
In [16]: a = np.arange(15).reshape(3, 5)

In [17]: a
Out[17]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])

In [18]: b = a % 3 == 0

In [19]: b
Out[19]:
array([[ True, False, False,  True, False],
       [False,  True, False, False,  True],
       [False, False,  True, False, False]])

In [20]: a[b]
Out[20]: array([ 0,  3,  6,  9, 12])
```

- 花式索引

```
In [23]: i = np.array([[0, 1], [1, 2]])

In [24]: j = [[2, 1], [4, 4]]

In [25]: a[i, j]
Out[25]:
array([[ 2,  6],
       [ 9, 14]])
```

- 省略号和增加维度

索引与切片

- 布尔索引

```
In [16]: a = np.arange(15).reshape(3, 5)

In [17]: a
Out[17]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])

In [18]: b = a % 3 == 0

In [19]: b
Out[19]:
array([[ True, False, False,  True, False],
       [False,  True, False, False,  True],
       [False, False,  True, False, False]])

In [20]: a[b]
Out[20]: array([ 0,  3,  6,  9, 12])
```

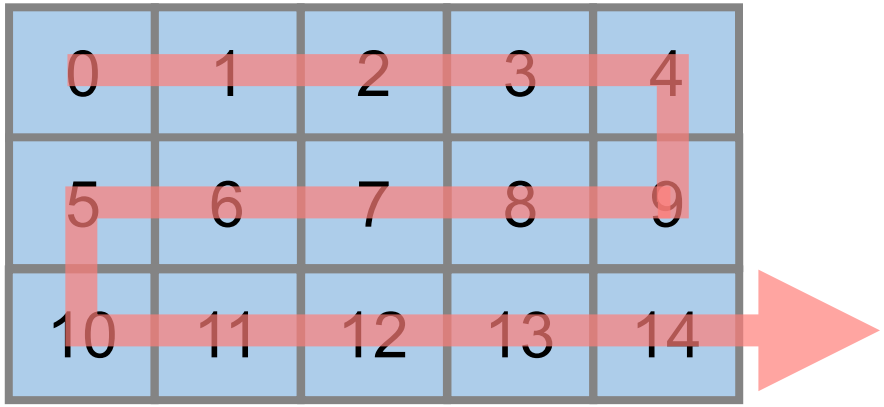
- 花式索引

```
In [23]: i = np.array([[0, 1], [1, 2]])

In [24]: j = [[2, 1], [4, 4]]

In [25]: a[i, j]
Out[25]:
array([[ 2,  6],
       [ 9, 14]])
```

- 省略号和增加维度

```
In [26]: a = np.arange(4)

In [28]: a[np.newaxis, ..., None]
Out[28]:
array([[[0],
        [1],
        [2],
        [3]]])

In [29]: _.shape
Out[29]: (1, 4, 1)
```

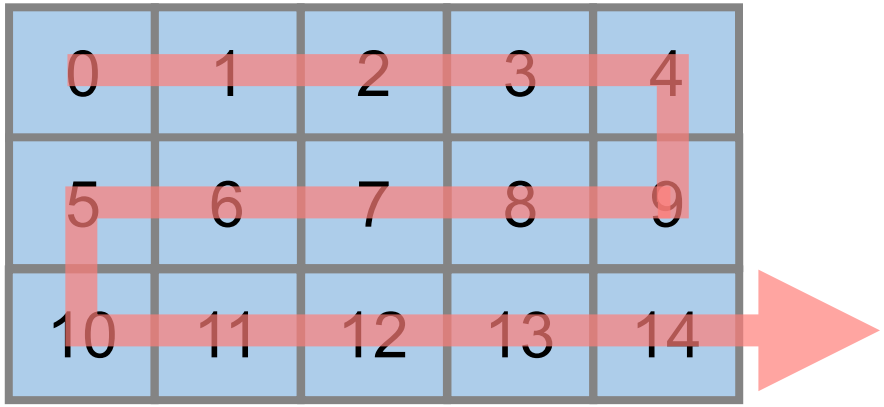
聚合




```
In [51]: a.sum()
Out[51]: 105
```

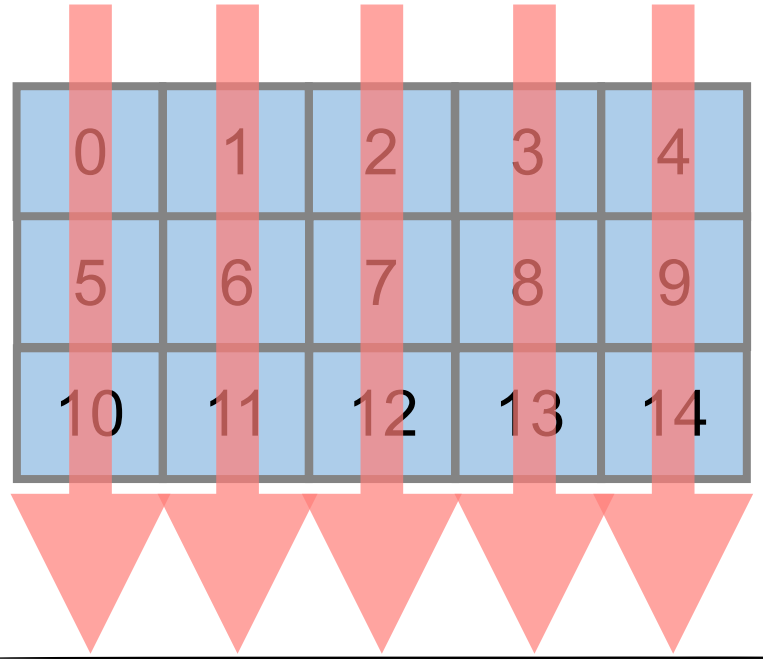
axis: None

聚合



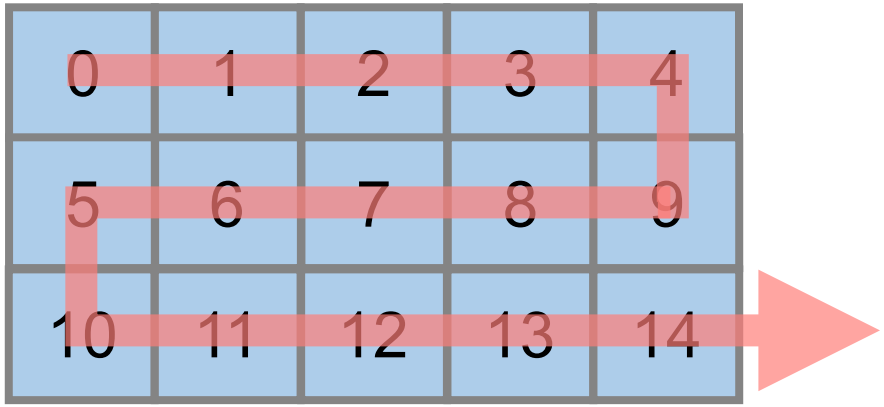

```
In [51]: a.sum()
Out[51]: 105
```

axis: None



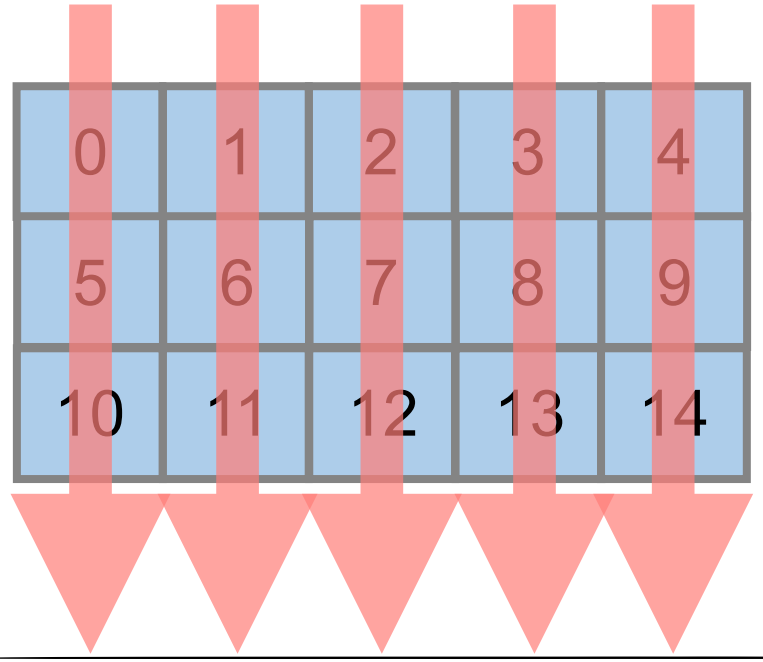

```
In [52]: a.sum(axis=0)
Out[52]: array([15, 18, 21, 24, 27])
```

聚合

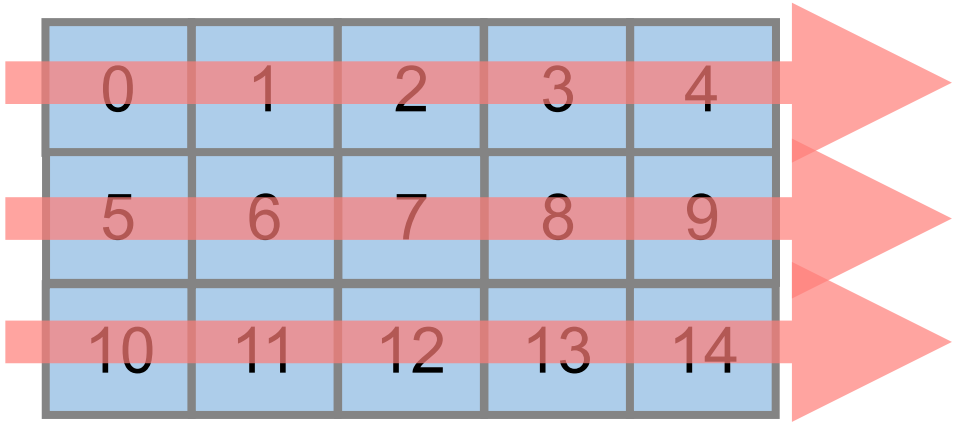



```
In [51]: a.sum()
Out[51]: 105
```

axis: None



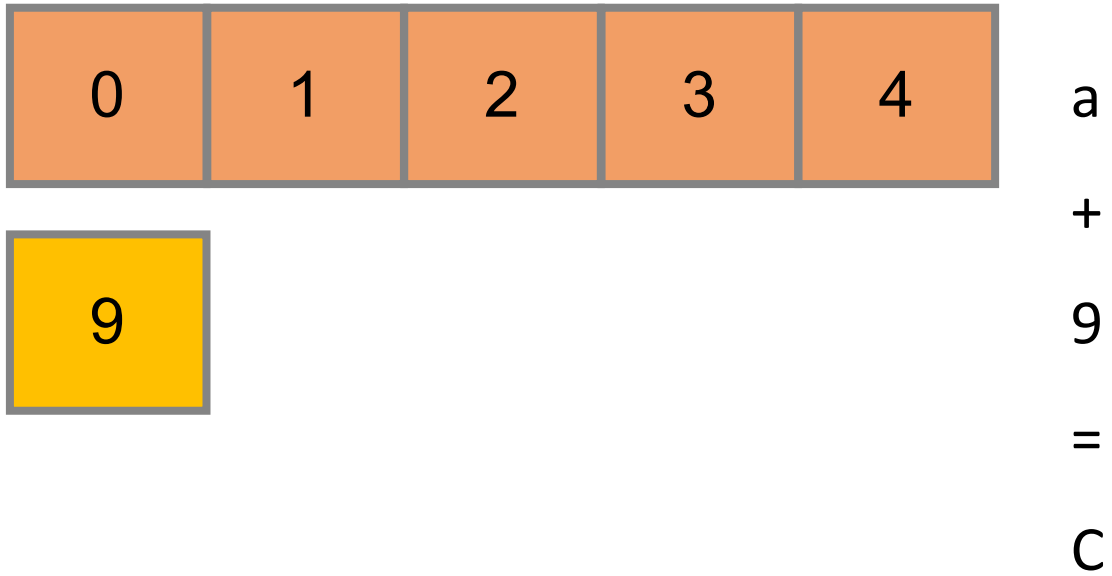

```
In [52]: a.sum(axis=0)
Out[52]: array([15, 18, 21, 24, 27])
```




```
In [53]: a.sum(axis=1)
Out[53]: array([10, 35, 60])
```

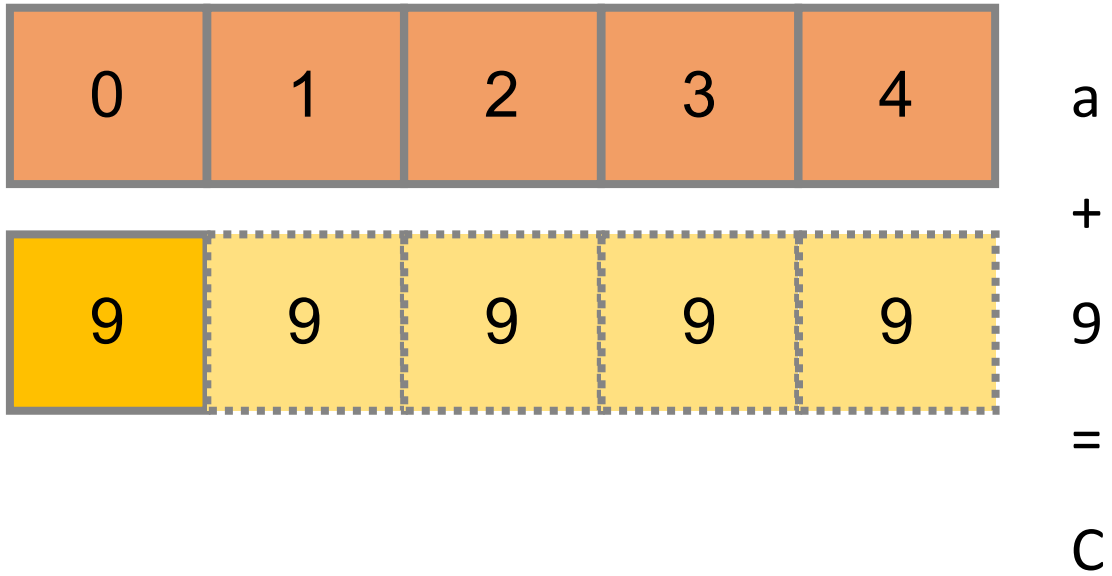
广播

c = a + 9



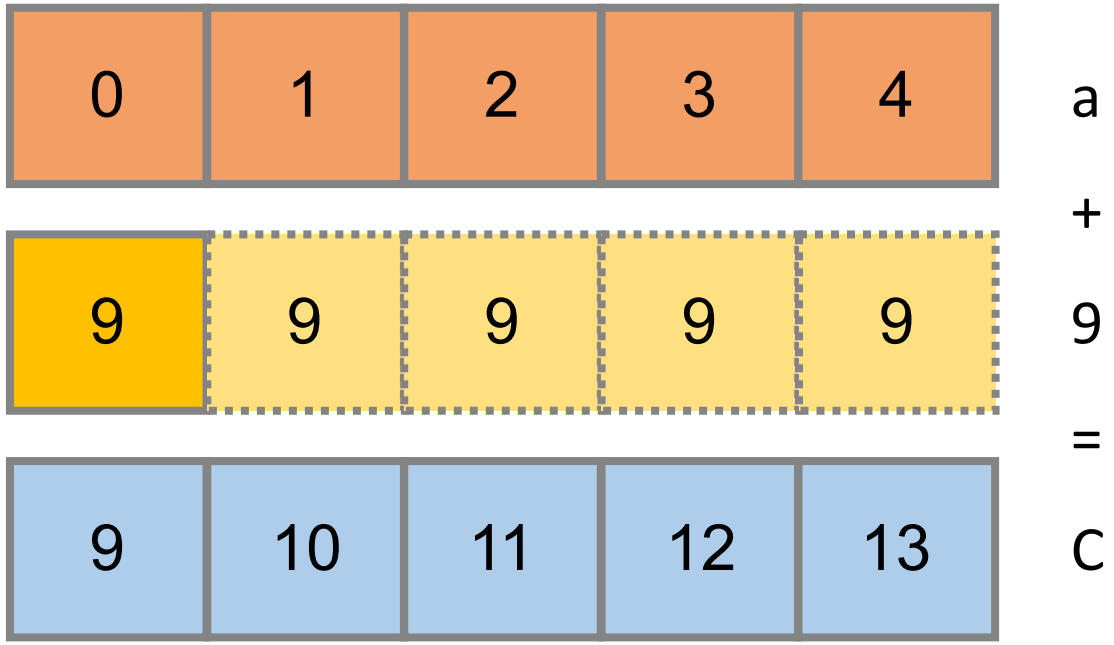
广播

c = a + 9



广播

c = a + 9



广播规则

两个数组，从后向前开始的维度，如果相同或者其中一个为1，则可以广播。
广播会在缺失或者长度为1的维度上进行。

```
A      (1d array):             3
B      (2d array):         2 x 3
Result (2d array):         2 x 3

A      (2d array):         6 x 1
B      (3d array):     1 x 6 x 4
Result (3d array):     1 x 6 x 4

A      (4d array): 3 x 1 x 6 x 4
B      (3d array):     2 x 1 x 4
Result (4d array): 3 x 2 x 6 x 4
```

广播实例：求向量每两个元素之间的差


```
In [69]: a = np.arange(1000)

In [70]: %time a[:, np.newaxis] - a
CPU times: user 3.32 ms, sys: 4.85 ms, total: 8.17
ms
Wall time: 6.33 ms
Out[70]:
array([[   0,   -1,   -2, ..., -997, -998, -999],
       [   1,    0,   -1, ..., -996, -997, -998],
       [   2,    1,    0, ..., -995, -996, -997],
       ...,
       [ 997,  996,  995, ...,    0,   -1,   -2],
       [ 998,  997,  996, ...,    1,    0,   -1],
       [ 999,  998,  997, ...,    2,    1,    0]])
```

广播实例：求向量每两个元素之间的差


```
In [69]: a = np.arange(1000)

In [70]: %time a[:, np.newaxis] - a
CPU times: user 3.32 ms, sys: 4.85 ms, total: 8.17
ms
Wall time: 6.33 ms
Out[70]:
array([[   0,   -1,   -2, ..., -997, -998, -999],
       [   1,    0,   -1, ..., -996, -997, -998],
       [   2,    1,    0, ..., -995, -996, -997],
       ...,
       [ 997,  996,  995, ...,    0,   -1,   -2],
       [ 998,  997,  996, ...,    1,    0,   -1],
       [ 999,  998,  997, ...,    2,    1,    0]])
```

1000 x 1

广播实例：求向量每两个元素之间的差


```
In [69]: a = np.arange(1000)

In [70]: %time a[:, np.newaxis] - a
CPU times: user 3.32 ms, sys: 4.85 ms, total: 8.17
ms
Wall time: 6.33 ms
Out[70]:
array([[   0,   -1,   -2, ..., -997, -998, -999],
       [   1,    0,   -1, ..., -996, -997, -998],
       [   2,    1,    0, ..., -995, -996, -997],
       ...,
       [ 997,  996,  995, ...,    0,   -1,   -2],
       [ 998,  997,  996, ...,    1,    0,   -1],
       [ 999,  998,  997, ...,    2,    1,    0]])
```

1000 x 1

```
In [73]: %%time
    ...: b = np.empty((1000, 1000), dtype=int)
    ...: for i in range(1000):
    ...:     for j in range(1000):
    ...:         b[i, j] = a[i] - a[j]
    ...:
    ...:
CPU times: user 512 ms, sys: 5.87 ms, total: 518 ms
Wall time: 517 ms

In [74]: b
Out[74]:
array([[   0,   -1,   -2, ..., -997, -998, -999],
       [   1,    0,   -1, ..., -996, -997, -998],
       [   2,    1,    0, ..., -995, -996, -997],
       ...,
       [ 997,  996,  995, ...,    0,   -1,   -2],
       [ 998,  997,  996, ...,    1,    0,   -1],
       [ 999,  998,  997, ...,    2,    1,    0]])
```

广播实例：求向量每两个元素之间的差


```
In [69]: a = np.arange(1000)

In [70]: %time a[:, np.newaxis] - a
CPU times: user 3.32 ms, sys: 4.85 ms, total: 8.17
ms
Wall time: 6.33 ms
Out[70]:
array([[   0,   -1,   -2, ..., -997, -998, -999],
       [   1,    0,   -1, ..., -996, -997, -998],
       [   2,    1,    0, ..., -995, -996, -997],
       ...,
       [ 997,  996,  995, ...,    0,   -1,   -2],
       [ 998,  997,  996, ...,    1,    0,   -1],
       [ 999,  998,  997, ...,    2,    1,    0]])
```

1000 x 1

```
In [73]: %%time
    ...: b = np.empty((1000, 1000), dtype=int)
    ...: for i in range(1000):
    ...:     for j in range(1000):
    ...:         b[i, j] = a[i] - a[j]
    ...:
    ...:
CPU times: user 512 ms, sys: 5.87 ms, total: 518 ms
Wall time: 517 ms

In [74]: b
Out[74]:
array([[   0,   -1,   -2, ..., -997, -998, -999],
       [   1,    0,   -1, ..., -996, -997, -998],
       [   2,    1,    0, ..., -995, -996, -997],
       ...,
       [ 997,  996,  995, ...,    0,   -1,   -2],
       [ 998,  997,  996, ...,    1,    0,   -1],
       [ 999,  998,  997, ...,    2,    1,    0]])
```

广播实例：求向量每两个元素之间的差


```
In [69]: a = np.arange(1000)

In [70]: %time a[:, np.newaxis] - a
CPU times: user 3.32 ms, sys: 4.85 ms, total: 8.17
ms
Wall time: 6.33 ms
Out[70]:
array([[   0,   -1,   -2, ..., -997, -998, -999],
       [   1,    0,   -1, ..., -996, -997, -998],
       [   2,    1,    0, ..., -995, -996, -997],
       ...,
       [ 997,  996,  995, ...,    0,   -1,   -2],
       [ 998,  997,  996, ...,    1,    0,   -1],
       [ 999,  998,  997, ...,    2,    1,    0]])
```

1000 x 1

快80x

```
In [73]: %%time
    ...: b = np.empty((1000, 1000), dtype=int)
    ...: for i in range(1000):
    ...:     for j in range(1000):
    ...:         b[i, j] = a[i] - a[j]
    ...:
    ...:
CPU times: user 512 ms, sys: 5.87 ms, total: 518 ms
Wall time: 517 ms

In [74]: b
Out[74]:
array([[   0,   -1,   -2, ..., -997, -998, -999],
       [   1,    0,   -1, ..., -996, -997, -998],
       [   2,    1,    0, ..., -995, -996, -997],
       ...,
       [ 997,  996,  995, ...,    0,   -1,   -2],
       [ 998,  997,  996, ...,    1,    0,   -1],
       [ 999,  998,  997, ...,    2,    1,    0]])
```

广播实例：求向量每两个元素之间的差


```
In [69]: a = np.arange(1000)

In [70]: %time a[:, np.newaxis] - a
CPU times: user 3.32 ms, sys: 4.85 ms, total: 8.17
ms
Wall time: 6.33 ms
Out[70]:
array([[    0,    -1,    -2, ..., -997, -998, -999],
       [    1,     0,    -1, ..., -996, -997, -998],
       [    2,     1,     0, ..., -995, -996, -997],
       ...,
       [  997,   996,   995, ...,    0,    -1,    -2],
       [  998,   997,   996, ...,    1,     0,    -1],
       [  999,   998,   997, ...,    2,     1,     0]])
```

1000 x 1

```
In [73]: %%time
    ...: b = np.empty((1000, 1000), dtype=int)
    ...: for i in range(1000):
    ...:     for j in range(1000):
    ...:         b[i, j] = a[i] - a[j]
    ...:
    ...:
CPU times: user 512 ms, sys: 5.87 ms, total: 518 ms
Wall time: 517 ms

In [74]: b
Out[74]:
array([[    0,    -1,    -2, ..., -997, -998, -999],
       [    1,     0,    -1, ..., -996, -997, -998],
       [    2,     1,     0, ..., -995, -996, -997],
       ...,
       [  997,   996,   995, ...,    0,    -1,    -2],
       [  998,   997,   996, ...,    1,     0,    -1],
       [  999,   998,   997, ...,    2,     1,     0]])
```

快80x

面向数组的编程和思维方式

广播实例进阶：计算两两节点间的欧式距离


```
In [76]: a = np.random.rand(1000, 10)

In [77]: %%time
    ...: b = a[:, np.newaxis, :] - a
    ...: print(np.sqrt((b ** 2).sum(-1)))
    ...:
    ...:
[[0.         1.34092722 1.03167424 ... 1.28462032 1.10915735 1.30688143]
 [1.34092722 0.         1.54184418 ... 1.36835935 1.46841795 1.50831086]
 [1.03167424 1.54184418 0.         ... 1.18083129 0.86086453 1.16513362]
 ...
 [1.28462032 1.36835935 1.18083129 ... 0.         0.93105133 1.23865215]
 [1.10915735 1.46841795 0.86086453 ... 0.93105133 0.         1.19356203]
 [1.30688143 1.50831086 1.16513362 ... 1.23865215 1.19356203 0.        ]]
CPU times: user 103 ms, sys: 75.9 ms, total: 179 ms
Wall time: 193 ms
```

广播实例进阶：计算两两节点间的欧式距离


```
In [76]: a = np.random.rand(1000, 10)

In [77]: %%time
    ...: b = a[:, np.newaxis, :] - a
    ...: print(np.sqrt((b ** 2).sum(-1)))
    ...:
    ...:
[[0.         1.34092722 1.03167424 ... 1.28462032 1.10915735 1.30688143]
 [1.34092722 0.         1.54184418 ... 1.36835935 1.46841795 1.50831086]
 [1.03167424 1.54184418 0.         ... 1.18083129 0.86086453 1.16513362]
 ...
 [1.28462032 1.36835935 1.18083129 ... 0.         0.93105133 1.23865215]
 [1.10915735 1.46841795 0.86086453 ... 0.93105133 0.         1.19356203]
 [1.30688143 1.50831086 1.16513362 ... 1.23865215 1.19356203 0.        ]]
CPU times: user 103 ms, sys: 75.9 ms, total: 179 ms
Wall time: 193 ms
```

1000 x 1 x 10

广播实例进阶：计算两两节点间的欧式距离


```
In [76]: a = np.random.rand(1000, 10)

In [77]: %%time
    ...: b = a[:, np.newaxis, :] - a
    ...: print(np.sqrt((b ** 2)
    ...:
    ...:
[[0.         1.34092722 1.03167
 [1.34092722 0.         1.54184
 [1.03167424 1.54184418 0.
 ...
 [1.28462032 1.36835935 1.18083
 [1.10915735 1.46841795 0.86086
 [1.30688143 1.50831086 1.16513
CPU times: user 103 ms, sys: 75
Wall time: 193 ms
```

1000 x 1 x 10

```
In [78]: %%time
    ...: from math import sqrt
    ...: b = np.empty((1000, 1000))
    ...: for i in range(1000):
    ...:     for j in range(1000):
    ...:         acc = 0.0
    ...:         for k in range(10):
    ...:             acc += (a[i, k] - a[j, k]) ** 2
    ...:         b[i, j] = sqrt(acc)
    ...: print(b)
    ...:
    ...:
[[0.         1.34092722 1.03167424 ... 1.28462032 1.10915735 1.30688143]
 [1.34092722 0.         1.54184418 ... 1.36835935 1.46841795 1.50831086]
 [1.03167424 1.54184418 0.         ... 1.18083129 0.86086453 1.16513362]
 ...
 [1.28462032 1.36835935 1.18083129 ... 0.         0.93105133 1.23865215]
 [1.10915735 1.46841795 0.86086453 ... 0.93105133 0.         1.19356203]
 [1.30688143 1.50831086 1.16513362 ... 1.23865215 1.19356203 0.        ]]
CPU times: user 10 s, sys: 29.5 ms, total: 10 s
Wall time: 10.1 s
```

广播实例进阶：计算两两节点间的欧式距离


```
In [76]: a = np.random.rand(1000, 10)

In [77]: %%time
    ...: b = a[:, np.newaxis, :] - a
    ...: print(np.sqrt((b ** 2)
    ...:
    ...:
[[0.         1.34092722 1.03167
 [1.34092722 0.         1.54184
 [1.03167424 1.54184418 0.
 ...
 [1.28462032 1.36835935 1.18083
 [1.10915735 1.46841795 0.86086
 [1.30688143 1.50831086 1.16513
CPU times: user 103 ms, sys: 75
Wall time: 193 ms
```

1000 x 1 x 10

```
In [78]: %%time
    ...: from math import sqrt
    ...: b = np.empty((1000, 1000))
    ...: for i in range(1000):
    ...:     for j in range(1000):
    ...:         acc = 0.0
    ...:         for k in range(10):
    ...:             acc += (a[i, k] - a[j, k]) ** 2
    ...:         b[i, j] = sqrt(acc)
    ...: print(b)
    ...:
    ...:
[[0.         1.34092722 1.03167424 ... 1.28462032 1.10915735 1.30688143]
 [1.34092722 0.         1.54184418 ... 1.36835935 1.46841795 1.50831086]
 [1.03167424 1.54184418 0.         ... 1.18083129 0.86086453 1.16513362]
 ...
 [1.28462032 1.36835935 1.18083129 ... 0.         0.93105133 1.23865215]
 [1.10915735 1.46841795 0.86086453 ... 0.93105133 0.         1.19356203]
 [1.30688143 1.50831086 1.16513362 ... 1.23865215 1.19356203 0.        ]]
CPU times: user 10 s, sys: 29.5 ms, total: 10 s
Wall time: 10.1 s
```

广播实例进阶：计算两两节点间的欧式距离


```
In [76]: a = np.random.rand(1000, 10)

In [77]: %%time
    ...: b = a[:, np.newaxis, :] - a
    ...: print(np.sqrt((b ** 2)
    ...:
    ...:
[[0.         1.34092722 1.03167
 [1.34092722 0.         1.54184
 [1.03167424 1.54184418 0.
 ...
 [1.28462032 1.36835935 1.18083
 [1.10915735 1.46841795 0.86086
 [1.30688143 1.50831086 1.16513
CPU times: user 103 ms, sys: 75
Wall time: 193 ms
```

1000 x 1 x 10

快>50x

```
In [78]: %%time
    ...: from math import sqrt
    ...: b = np.empty((1000, 1000))
    ...: for i in range(1000):
    ...:     for j in range(1000):
    ...:         acc = 0.0
    ...:         for k in range(10):
    ...:             acc += (a[i, k] - a[j, k]) ** 2
    ...:         b[i, j] = sqrt(acc)
    ...: print(b)
    ...:
    ...:
[[0.         1.34092722 1.03167424 ... 1.28462032 1.10915735 1.30688143]
 [1.34092722 0.         1.54184418 ... 1.36835935 1.46841795 1.50831086]
 [1.03167424 1.54184418 0.         ... 1.18083129 0.86086453 1.16513362]
 ...
 [1.28462032 1.36835935 1.18083129 ... 0.         0.93105133 1.23865215]
 [1.10915735 1.46841795 0.86086453 ... 0.93105133 0.         1.19356203]
 [1.30688143 1.50831086 1.16513362 ... 1.23865215 1.19356203 0.        ]]
CPU times: user 10 s, sys: 29.5 ms, total: 10 s
Wall time: 10.1 s
```

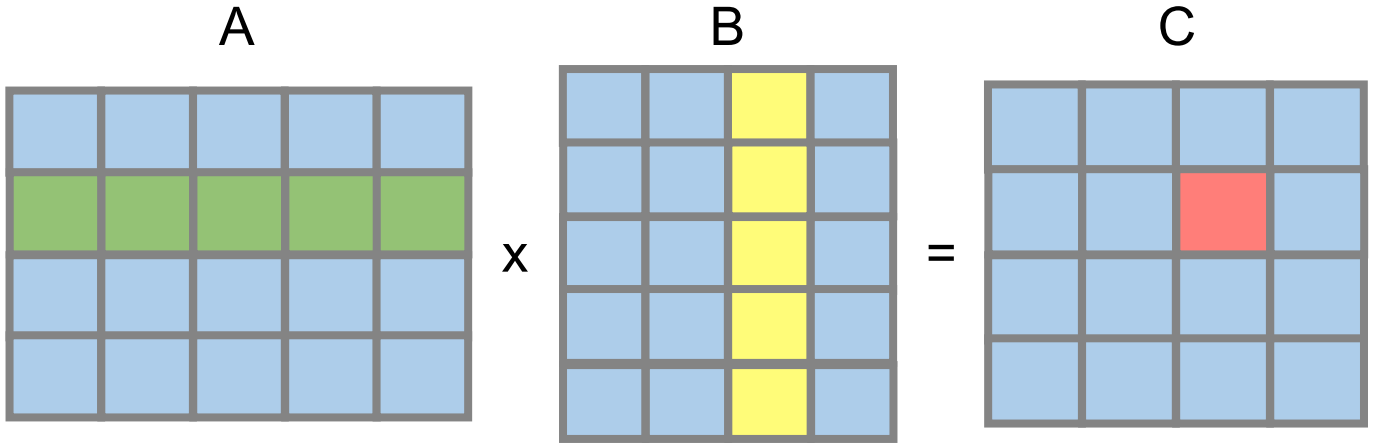
线性代数

- 矩阵乘法

- 矩阵分解
  - Cholesky
  - QR
  - SVD

线性代数

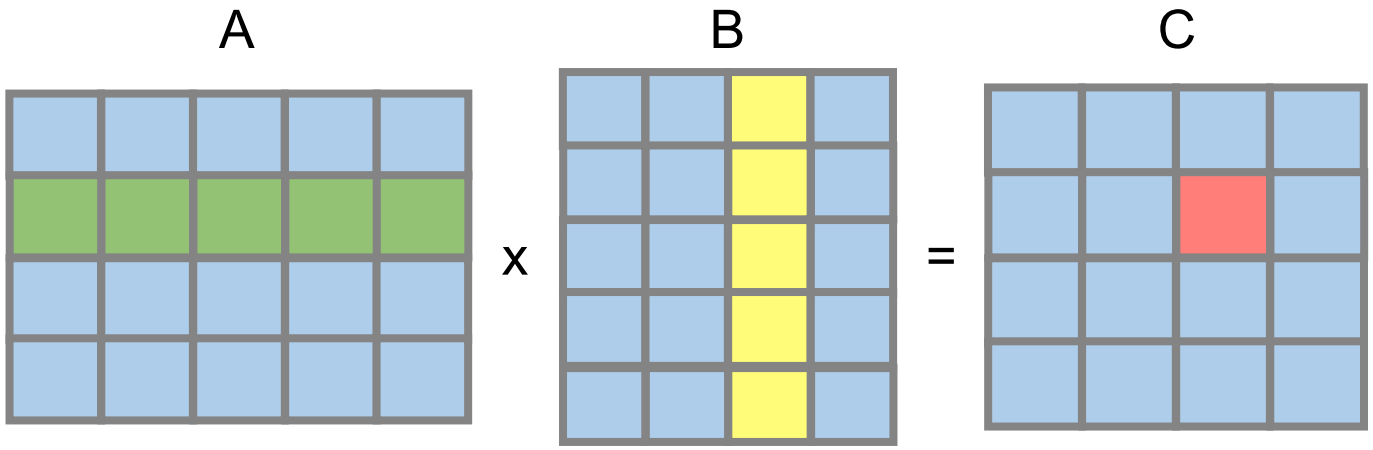
- 矩阵乘法



- 矩阵分解
  - Cholesky
  - QR
  - SVD

线性代数

- 矩阵乘法

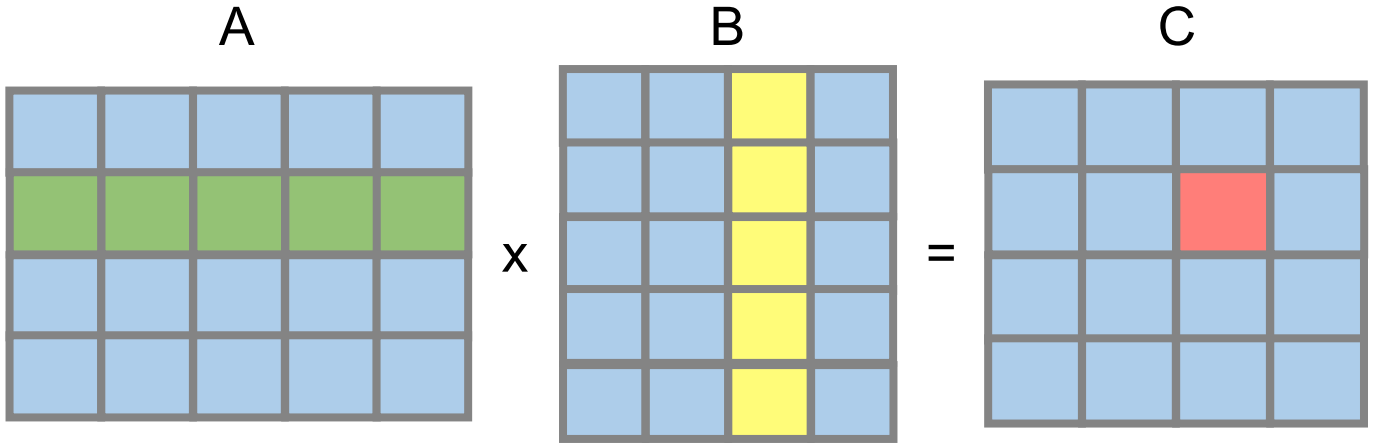


- 矩阵分解
  - Cholesky
  - QR
  - SVD

```
In [124]: a = np.arange(20).reshape(4, 5)

In [125]: b = np.arange(20).reshape(5, 4)

In [126]: a.dot(b)
Out[126]:
array([[120, 130, 140, 150],
       [320, 355, 390, 425],
       [520, 580, 640, 700],
       [720, 805, 890, 975]])

In [127]: a @ b
Out[127]:
array([[120, 130, 140, 150],
       [320, 355, 390, 425],
       [520, 580, 640, 700],
       [720, 805, 890, 975]])
```

线性代数

- 矩阵乘法



- 矩阵分解
  - Cholesky
  - QR
  - SVD

```
In [124]: a = np.arange(20).reshape(4, 5)

In [125]: b = np.arange(20).reshape(5, 4)

In [126]: a.dot(b)
Out[126]:
array([[120, 130, 140, 150],
       [320, 355, 390, 425],
       [520, 580, 640, 700],
       [720, 805, 890, 975]])

In [127]: a @ b
Out[127]:
array([[120, 130, 140, 150],
       [320, 355, 390, 425],
       [520, 580, 640, 700],
       [720, 805, 890, 975]])
```

```
In [128]: a = np.random.rand(20000, 100)

In [129]: u, s, v = np.linalg.svd(a)

In [130]: np.allclose(u[:, :100].dot(np.diag(s)).dot(v), a)
Out[130]: True
```

Numpy是标准（numpy-*ish* API）


```
In [1]: import numpy as np

In [3]: %%time
   ...: a = np.random.rand(5000, 10)
   ...: _ = ((a[:, np.newaxis, :] - a) ** 2).sum(axis=-1)
   ...:
   ...:
CPU times: user 2.97 s, sys: 923 ms, total: 3.9 s
Wall time: 2.77 s
```

```
In [2]: import cupy as cp

In [4]: %%time
   ...: a = cp.random.rand(5000, 10)
   ...: _ = ((a[:, cp.newaxis, :] - a) ** 2).sum(axis=-1)
   ...:
   ...:
CPU times: user 373 ms, sys: 163 ms, total: 536 ms
Wall time: 538 ms
```

Numpy是协议

- NEP-18： __array_function__ 协议，所有实现了这个协议的库都可以通过 numpy 函数调用。
- 使用需要设置环境变量 NUMPY_EXPERIMENTAL_ARRAY_FUNCTION=1
- 1.16 开始支持，1.17 可能默认打开

```
$ NUMPY_EXPERIMENTAL_ARRAY_FUNCTION=1 ipython
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import numpy as np

In [2]: import cupy as cp

In [3]: %%time
   ...: a = cp.random.rand(50, 5)
   ...: np.linalg.svd(a)
   ...:
   ...:
CPU times: user 416 ms, sys: 249 ms, total: 665 ms
Wall time: 670 ms
```

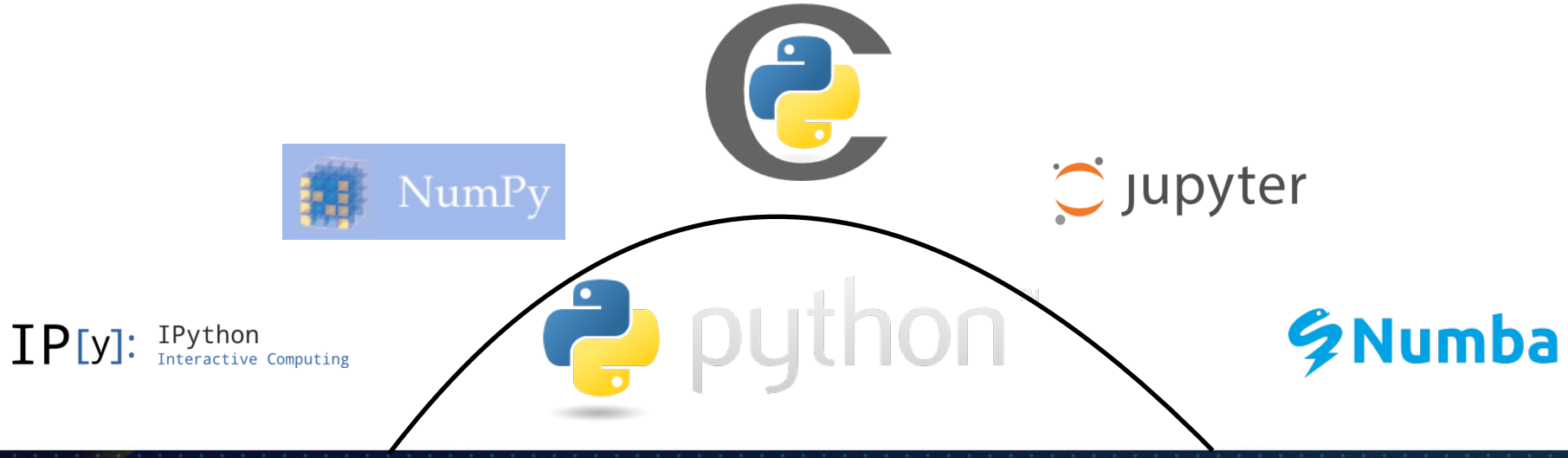
Numpy是协议

- NEP-18： __array_function__ 协议，所有实现了这个协议的库都可以通过 numpy 函数调用。
- 使用需要设置环境变量 NUMPY_EXPERIMENTAL_ARRAY_FUNCTION=1
- 1.16 开始支持，1.17 可能默认打开

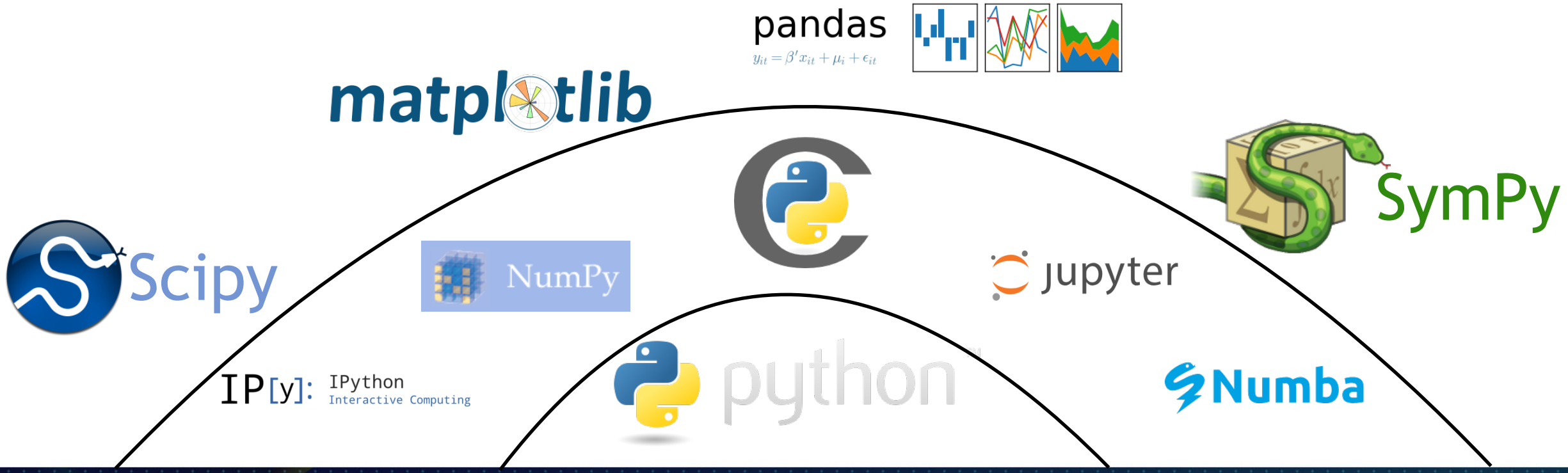
```
$ NUMPY_EXPERIMENTAL_ARRAY_FUNCTION=1 ipython
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import numpy as np

In [2]: import cupy as cp

In [3]: %%time
   ...: a = cp.random.rand(50, 5)
   ...: np.linalg.svd(a)
   ...:
   ...:
CPU times: user 416 ms, sys: 249 ms, total: 665 ms
Wall time: 670 ms
```

Numpy 函数能作用在 cupy ndarray

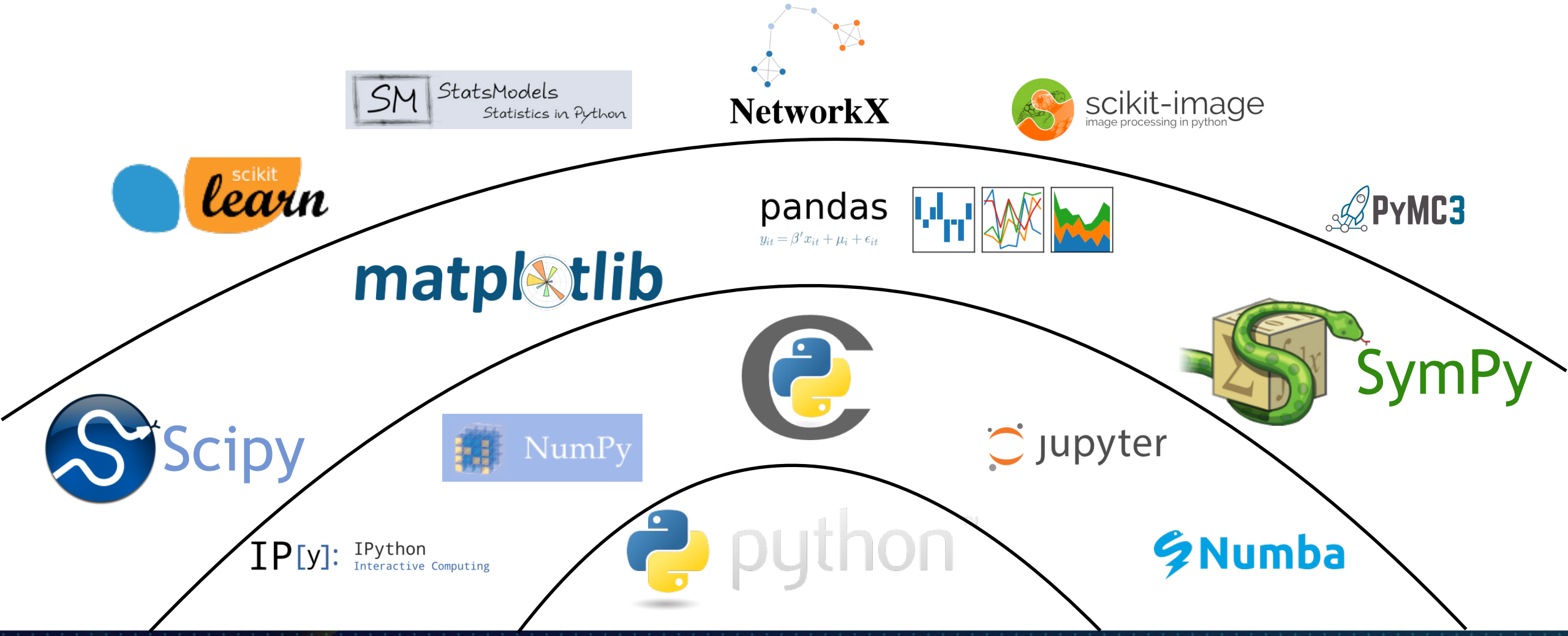
Numpy是基础


NumPy
jupyter
IP[y]: IPython
Interactive Computing
python
Numba

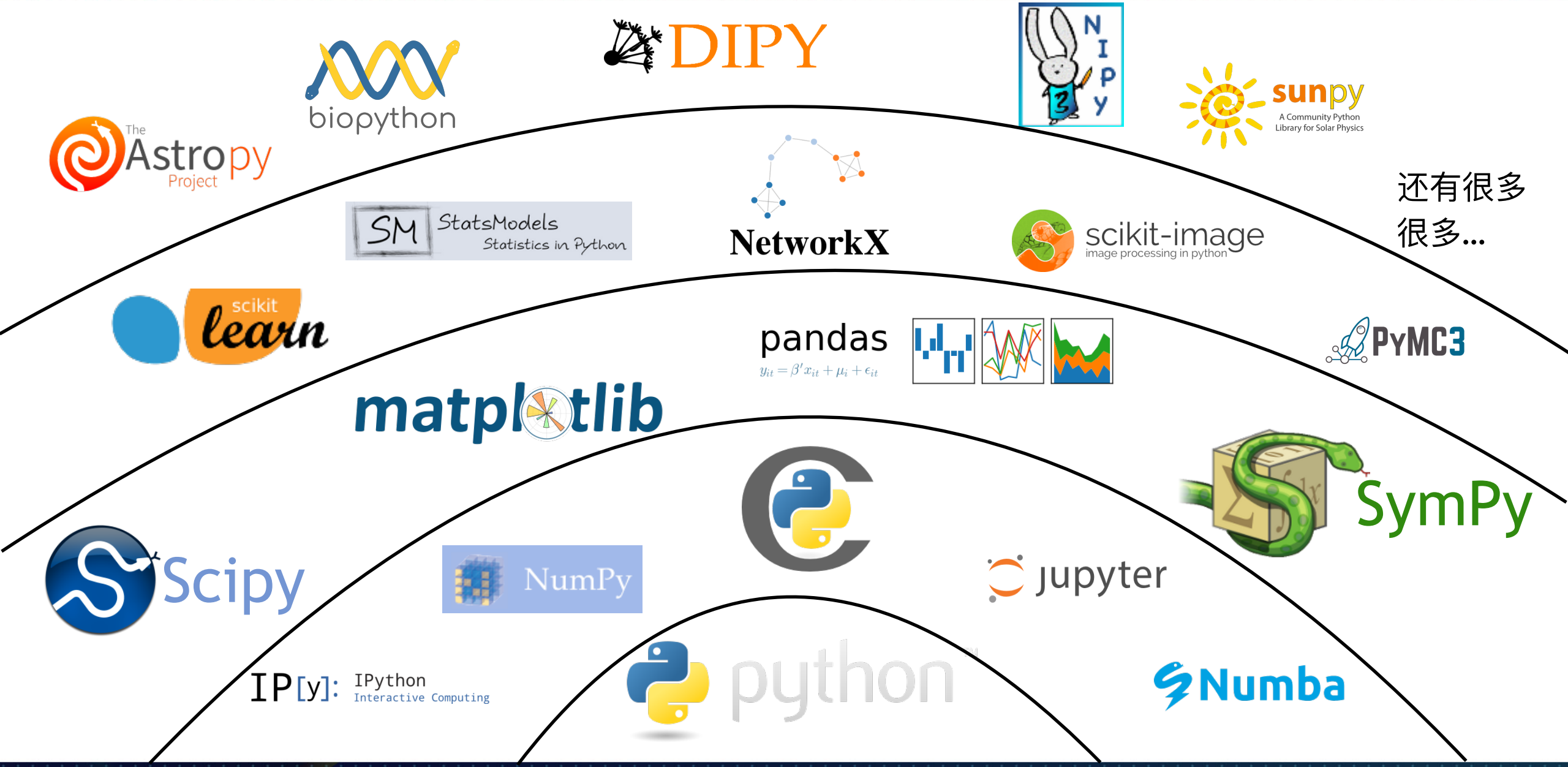
Numpy是基础


pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$
matplotlib
SymPy
Scipy
NumPy
jupyter
IP[y]: IPython
Interactive Computing
python
Numba

Numpy是基础


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
SymPy
Scipy
NumPy
jupyter
IP[y]: IPython
Interactive Computing
python
Numba

Numpy是基础


biopython
DIPY
NIPY
sunpy
A Community Python Library for Solar Physics
The Astropy Project
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
SymPy
Scipy
NumPy
jupyter
IP[y]: IPython
Interactive Computing
python
Numba

问题是……

问题是……


```
In [3]: %time _ = (a + 1) * 2
CPU times: user 1.55 ms, sys: 1.99 ms, total: 3.53 ms
Wall time: 2.34 ms
```

```
In [4]: %time _ = np.std(a, axis=1)
CPU times: user 1.71 ms, sys: 1.84 ms, total: 3.54 ms
Wall time: 2.8 ms
```

```
In [7]: %time _ = np.fft.fft(a)
CPU times: user 18.7 ms, sys: 9.55 ms, total: 28.3 ms
Wall time: 19.6 ms
```

问题是……


```
In [3]: %time _ = (a + 1) * 2
CPU times: user 1.55 ms, sys: 1.99 ms, total: 3.53 ms
Wall time: 2.34 ms
```

```
In [4]: %time _ = np.std(a, axis=1)
CPU times: user 1.71 ms, sys: 1.84 ms, total: 3.54 ms
Wall time: 2.8 ms
```

```
In [7]: %time _ = np.fft.fft(a)
CPU times: user 18.7 ms, sys: 9.55 ms, total: 28.3 ms
Wall time: 19.6 ms
```

绝大部分操作都
不能利用多核

问题是……


```
In [3]: %time _ = (a + 1) * 2
CPU times: user 1.55 ms, sys: 1.99 ms, total: 3.53 ms
Wall time: 2.34 ms
```

```
In [4]: %time _ = np.std(a, axis=1)
CPU times: user 1.71 ms, sys: 1.84 ms, total: 3.54 ms
Wall time: 2.8 ms
```

```
In [7]: %time _ = np.fft.fft(a)
CPU times: user 18.7 ms, sys: 9.55 ms, total: 28.3 ms
Wall time: 19.6 ms
```

```
In [6]: %time _ = np.linalg.svd(a)
CPU times: user 1.31 s, sys: 63.8 ms, total: 1.38 s
Wall time: 354 ms
```

```
In [9]: %time _ = a @ a.T
CPU times: user 75.8 ms, sys: 5.66 ms, total: 81.5 ms
Wall time: 24.2 ms
```

绝大部分操作都
不能利用多核

问题是……


```
In [3]: %time _ = (a + 1) * 2
CPU times: user 1.55 ms, sys: 1.99 ms, total: 3.53 ms
Wall time: 2.34 ms
```

```
In [4]: %time _ = np.std(a, axis=1)
CPU times: user 1.71 ms, sys: 1.84 ms, total: 3.54 ms
Wall time: 2.8 ms
```

```
In [7]: %time _ = np.fft.fft(a)
CPU times: user 18.7 ms, sys: 9.55 ms, total: 28.3 ms
Wall time: 19.6 ms
```

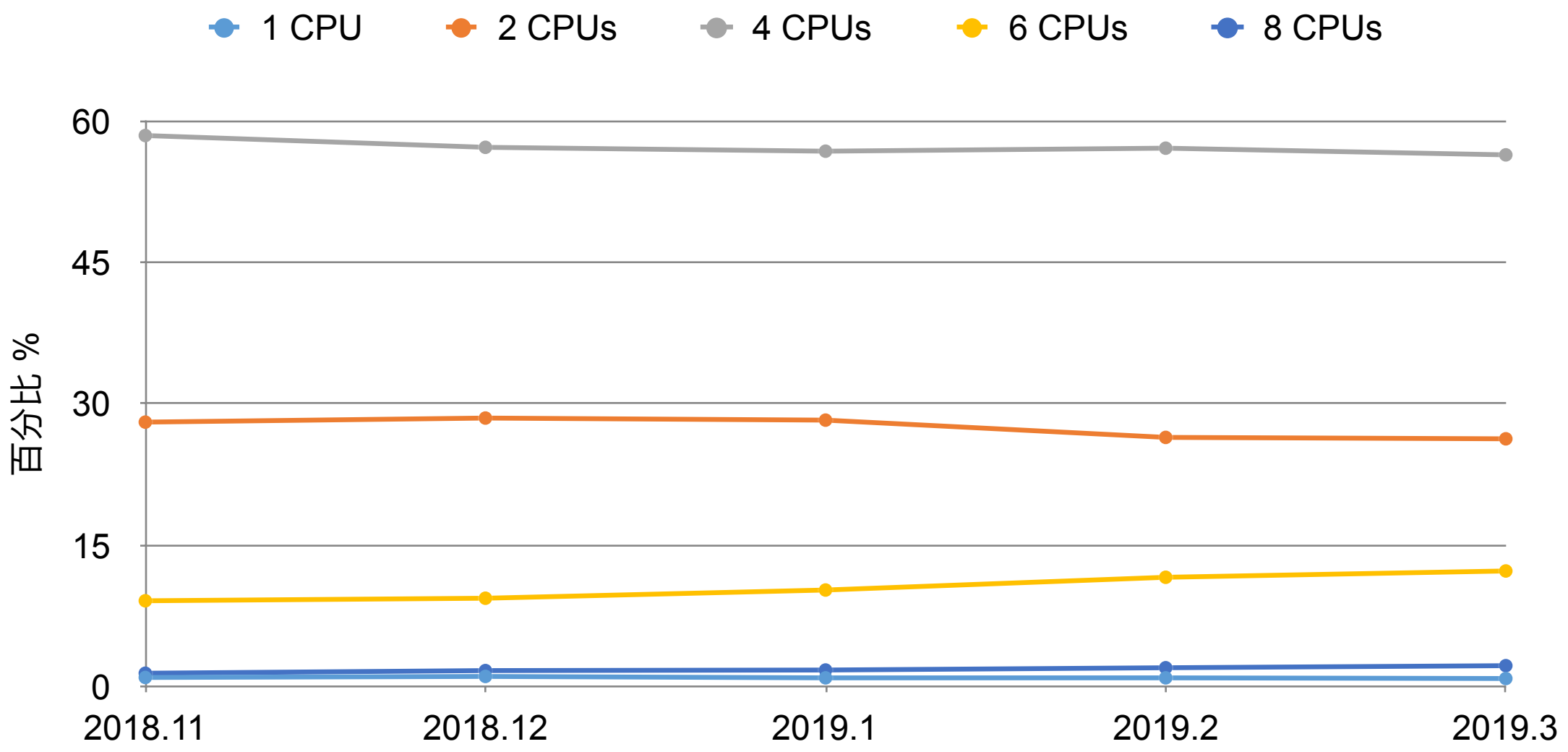
绝大部分操作都
不能利用多核

```
In [6]: %time _ = np.linalg.svd(a)
CPU times: user 1.31 s, sys: 63.8 ms, total: 1.38 s
Wall time: 354 ms
```

```
In [9]: %time _ = a @ a.T
CPU times: user 75.8 ms, sys: 5.66 ms, total: 81.5 ms
Wall time: 24.2 ms
```

少量线性代数模块
能利用多核

现在是多核时代



- 数据采纳自steam：https://store.steampowered.com/hwsurvey/cpus/

一台机器可以 Scale up，提升 CPU、GPU、内存，但资源总是有限的。
我们就需要分布式。

Mars tensor 是并行和分布式化的 Numpy


```
In [5]: import numpy as np

In [6]: a = np.random.rand(5000, 10)

In [7]: %%time
   ...: b = a[:, np.newaxis, :] - a
   ...: print(np.sqrt(b ** 2).sum(-1))
   ...:
   ...:
[[0.         2.82510037 4.46837739 ... 3.27907163 3.4719372  3.37709946]
 [2.82510037 0.         3.32650516 ... 2.53584057 3.22452394 2.45815415]
 [4.46837739 3.32650516 0.         ... 3.08332206 2.3909553  3.10370547]
 ...
 [3.27907163 2.53584057 3.08332206 ... 0.         2.21482239 3.39591248]
 [3.4719372  3.22452394 2.3909553  ... 2.21482239 0.         2.76933364]
 [3.37709946 2.45815415 3.10370547 ... 3.39591248 2.76933364
0.        ]]
CPU times: user 3.8 s, sys: 2.72 s, total: 6.52 s
Wall time: 3.44 s
```

Mars tensor 是并行和分布式化的 Numpy


```
In [5]: import numpy as np

In [6]: a = np.random.rand(5000, 10)

In [7]: %%time
   ...: b = a[:, np.newaxis, :] - a
   ...: print(np.sqrt(b ** 2).sum(-1))
   ...:
   ...:
[[0.         2.82510037 4.468
 [2.82510037 0.         3.326
 [4.46837739 3.32650516 0.
 ...
 [3.27907163 2.53584057 3.083
 [3.4719372  3.22452394 2.390
 [3.37709946 2.45815415 3.103
0.        ]]
CPU times: user 3.8 s, sys: 2
Wall time: 3.44 s
```

```
In [8]: import mars.tensor as mt

In [12]: ma = mt.array(a, chunk_size=2000)

In [13]: %%time
    ...: b = ma[:, mt.newaxis, :] - ma
    ...: print(mt.sqrt(b ** 2).sum(-1).execute())
    ...:
    ...:
[[0.         2.82510037 4.46837739 ... 3.27907163 3.4719372  3.37709946]
 [2.82510037 0.         3.32650516 ... 2.53584057 3.22452394 2.45815415]
 [4.46837739 3.32650516 0.         ... 3.08332206 2.3909553  3.10370547]
 ...
 [3.27907163 2.53584057 3.08332206 ... 0.         2.21482239 3.39591248]
 [3.4719372  3.22452394 2.3909553  ... 2.21482239 0.         2.76933364]
 [3.37709946 2.45815415 3.10370547 ... 3.39591248 2.76933364 0.        ]]
CPU times: user 6.92 s, sys: 1.42 s, total: 8.34 s
Wall time: 1.95 s
```

Mars tensor 是并行和分布式化的 Numpy


```
In [5]: import numpy as np

In [6]: a = np.random.rand(5000, 10)

In [7]: %%time
   ...: b = a[:, np.newaxis, :] - a
   ...: print(np.sqrt(b ** 2).sum(-1))
   ...:
   ...:
[[0.          2.82510037 4.468
 [2.82510037 0.         3.326
 [4.46837739 3.32650516 0.
 ...
 [3.27907163 2.53584057 3.083
 [3.4719372  3.22452394 2.390
 [3.37709946 2.45815415 3.103
0.         ]]
CPU times: user 3.8 s, sys: 2
Wall time: 3.44 s
```

import 替换

```
In [8]: import mars.tensor as mt

In [12]: ma = mt.array(a, chunk_size=2000)

In [13]: %%time
    ...: b = ma[:, mt.newaxis, :] - ma
    ...: print(mt.sqrt(b ** 2).sum(-1).execute())
    ...:
    ...:
[[0.          2.82510037 4.46837739 ... 3.27907163 3.4719372  3.37709946]
 [2.82510037 0.         3.32650516 ... 2.53584057 3.22452394 2.45815415]
 [4.46837739 3.32650516 0.         ... 3.08332206 2.3909553  3.10370547]
 ...
 [3.27907163 2.53584057 3.08332206 ... 0.         2.21482239 3.39591248]
 [3.4719372  3.22452394 2.3909553  ... 2.21482239 0.         2.76933364]
 [3.37709946 2.45815415 3.10370547 ... 3.39591248 2.76933364 0.        ]]
CPU times: user 6.92 s, sys: 1.42 s, total: 8.34 s
Wall time: 1.95 s
```

Mars tensor 是并行和分布式化的 Numpy


```
In [5]: import numpy as np

In [6]: a = np.random.rand(5000, 10)

In [7]: %%time
   ...: b = a[:, np.newaxis, :] - a
   ...: print(np.sqrt(b ** 2).sum(-1))
   ...:
   ...:
[[0.          2.82510037 4.468
 [2.82510037 0.         3.326
 [4.46837739 3.32650516 0.
 ...
 [3.27907163 2.53584057 3.083
 [3.4719372  3.22452394 2.390
 [3.37709946 2.45815415 3.103
0.         ]]
CPU times: user 3.8 s, sys: 2
Wall time: 3.44 s
```

import 替换

```
In [8]: import mars.tensor as mt

In [12]: ma = mt.array(a, chunk_size=2000)

In [13]: %%time
    ...: b = ma[:, mt.newaxis, :] - ma
    ...: print(mt.sqrt(b ** 2).sum(-1).execute())
    ...:
    ...:
[[0.          2.82510037 4.46837739 ... 3.27907163 3.4719372  3.37709946]
 [2.82510037 0.         3.32650516 ... 2.53584057 3.22452394 2.45815415]
 [4.46837739 3.32650516 0.         ... 3.08332206 2.3909553  3.10370547]
 ...
 [3.27907163 2.53584057 3.08332206 ... 0.         2.21482239 3.39591248]
 [3.4719372  3.22452394 2.3909553  ... 2.21482239 0.         2.76933364]
 [3.37709946 2.45815415 3.10370547 ... 3.39591248 2.76933364 0.        ]]
CPU times: user 6.92 s, sys: 1.42 s, total: 8.34 s
Wall time: 1.95 s
```

chunk_size 可以不填，会自动根据配置的chunk大小计算，其对性能影响巨大

Mars tensor 是并行和分布式化的 Numpy


```
In [5]: import numpy as np

In [6]: a = np.random.rand(5000, 10)

In [7]: %%time
   ...: b = a[:, np.newaxis, :] - a
   ...: print(np.sqrt(b ** 2).sum(-1))
   ...:
   ...:
[[0.          2.82510037 4.468
 [2.82510037 0.         3.326
 [4.46837739 3.32650516 0.
 ...
 [3.27907163 2.53584057 3.083
 [3.4719372  3.22452394 2.390
 [3.37709946 2.45815415 3.103
0.         ]]
CPU times: user 3.8 s, sys: 2
Wall time: 3.44 s
```

import 替换

```
In [8]: import mars.tensor as mt

In [12]: ma = mt.array(a, chunk_size=2000)

In [13]: %%time
    ...: b = ma[:, mt.newaxis, :] - ma
    ...: print(mt.sqrt(b ** 2).sum(-1).execute())
    ...:
    ...:
[[0.          2.82510037 4.46837739 ... 3.27907163 3.4719372  3.37709946]
 [2.82510037 0.         3.32650516 ... 2.53584057 3.22452394 2.45815415]
 [4.46837739 3.32650516 0.         ... 3.08332206 2.3909553  3.10370547]
 ...
 [3.27907163 2.53584057 3.08332206 ... 0.         2.21482239 3.39591248]
 [3.4719372  3.22452394 2.3909553  ... 2.21482239 0.         2.76933364]
 [3.37709946 2.45815415 3.10370547 ... 3.39591248 2.76933364 0.        ]]
CPU times: user 6.92 s, sys: 1.42 s, total: 8.34 s
Wall time: 1.95 s
```

chunk_size 可以不填，会自动根据配置的chunk大小计算，其对性能影响巨大

延迟执行，可以做更多优化

Mars tensor 是并行和分布式化的 Numpy


```
In [5]: import numpy as np

In [6]: a = np.random.rand(5000, 10)

In [7]: %%time
   ...: b = a[:, np.newaxis, :] - a
   ...: print(np.sqrt(b ** 2).sum(-1))
   ...:
   ...:
[[0.          2.82510037 4.468
 [2.82510037 0.         3.326
 [4.46837739 3.32650516 0.
 ...
 [3.27907163 2.53584057 3.083
 [3.4719372  3.22452394 2.390
 [3.37709946 2.45815415 3.103
0.         ]]
CPU times: user 3.8 s, sys: 2
Wall time: 3.44 s
```

import 替换

```
In [8]: import mars.tensor as mt

In [12]: ma = mt.array(a, chunk_size=2000)

In [13]: %%time
    ...: b = ma[:, mt.newaxis, :] - ma
    ...: print(mt.sqrt(b ** 2).sum(-1).execute())
    ...:
    ...:
[[0.          2.82510037 4.46837739 ... 3.27907163 3.4719372  3.37709946]
 [2.82510037 0.         3.32650516 ... 2.53584057 3.22452394 2.45815415]
 [4.46837739 3.32650516 0.         ... 3.08332206 2.3909553  3.10370547]
 ...
 [3.27907163 2.53584057 3.08332206 ... 0.         2.21482239 3.39591248]
 [3.4719372  3.22452394 2.3909553  ... 2.21482239 0.         2.76933364]
 [3.37709946 2.45815415 3.10370547 ... 3.39591248 2.76933364 0.        ]]
CPU times: user 6.92 s, sys: 1.42 s, total: 8.34 s
Wall time: 1.95 s
```

chunk_size 可以不填，会自动根据配置的chunk大小计算，其对性能影响巨大

延迟执行，可以做更多优化

Mars tensor 是并行和分布式化的 Numpy


```
In [5]: import numpy as np

In [6]: a = np.random.rand(5000, 10)

In [7]: %%time
   ...: b = a[:, np.newaxis, :] - a
   ...: print(np.sqrt(b ** 2).sum(-1))
   ...:
   ...:
[[0.          2.82510037 4.468
 [2.82510037 0.         3.326
 [4.46837739 3.32650516 0.
 ...
 [3.27907163 2.53584057 3.083
 [3.4719372  3.22452394 2.390
 [3.37709946 2.45815415 3.103
0.         ]]
CPU times: user 3.8 s, sys: 2
Wall time: 3.44 s
```

import 替换

```
In [8]: import mars.tensor as mt

In [12]: ma = mt.array(a, chunk_size=2000)

In [13]: %%time
    ...: b = ma[:, mt.newaxis, :] - ma
    ...: print(mt.sqrt(b ** 2).sum(-1).execute())
    ...:
    ...:
[[0.          2.82510037 4.46837739 ... 3.27907163 3.4719372  3.37709946]
 [2.82510037 0.         3.32650516 ... 2.53584057 3.22452394 2.45815415]
 [4.46837739 3.32650516 0.         ... 3.08332206 2.3909553  3.10370547]
 ...
 [3.27907163 2.53584057 3.08332206 ... 0.         2.21482239 3.39591248]
 [3.4719372  3.22452394 2.3909553  ... 2.21482239 0.         2.76933364]
 [3.37709946 2.45815415 3.10370547 ... 3.39591248 2.76933364 0.        ]]
CPU times: user 6.92 s, sys: 1.42 s, total: 8.34 s
Wall time: 1.95 s
```

chunk_size 可以不填，会自动根据配置的chunk大小计算，其对性能影响巨大

延迟执行，可以做更多优化

自动利用多核，提升 ~40%

如果两两计算距离的节点数增加到 50 万

```
import mars.tensor as mt

a = mt.random.rand(500000, 10, chunk_size=10000)
b = a[:, mt.newaxis, :] - a
mt.triu(mt.sqrt((b ** 2).sum(axis=-1)))
```

如果两两计算距离的节点数增加到 50 万

```
import mars.tensor as mt

a = mt.random.rand(500000, 10, chunk_size=10000)
b = a[:, mt.newaxis, :] - a
mt.triu(mt.sqrt((b ** 2).sum(axis=-1)))
```

- 最终结果数据都需要 1.82T 的存储

如果两两计算距离的节点数增加到 50 万

```
import mars.tensor as mt

a = mt.random.rand(500000, 10, chunk_size=10000)
b = a[:, mt.newaxis, :] - a
mt.triu(mt.sqrt((b ** 2).sum(axis=-1)))
```

- 最终结果数据都需要 1.82T 的存储
- 5000 节点计算量的 1 万倍，单机预计需要 5.42 小时（前提是有 18.2T 内存的机器）

如果两两计算距离的节点数增加到 50 万

```
import mars.tensor as mt

a = mt.random.rand(500000, 10, chunk_size=10000)
b = a[:, mt.newaxis, :] - a
mt.triu(mt.sqrt((b ** 2).sum(axis=-1)))
```

- 最终结果数据都需要 1.82T 的存储
- 5000 节点计算量的 1 万倍，单机预计需要 5.42 小时（前提是有 18.2T 内存的机器）
- Mars 在分布式环境 37 workers（8 核 32G）7 分钟计算完成

Eager mode

- Graph mode 是默认计算模式，即通过 execute 触发计算，为性能而生
- Eager mode 对开发和 debug 更友好

Eager mode

- Graph mode 是默认计算模式，即通过 execute 触发计算，为性能而生
- Eager mode 对开发和 debug 更友好

```
In [27]: from mars.config import options

In [28]: options.eager_mode = True

In [30]: a = mt.arange(10)

In [31]: a
Out[31]:
Tensor <op=TensorArange, shape=(10,), key=00f46848d449879e7786fadf909a7490, data=
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])>

In [32]: mt.sum(a)
Out[32]:
Tensor <op=TensorSum, shape=(), key=309966f894a74b273447a81dd7730330, data=
45>

In [33]: mt.dot(a, a)
Out[33]:
Tensor <op=TensorTensorDot, shape=(), key=bca45e00ce32cf77a6cbe9335ca750fd, data=
285>
```

Mars 和 NEP-18

- Mars 实现了 __array_function__ 接口
- 配合 eager mode，甚至都不需要 import 替换

Mars 和 NEP-18

- Mars 实现了 __array_function__ 接口
- 配合 eager mode，甚至都不需要 import 替换

```
In [6]: np.sum(a)
Out[6]:
Tensor <op=TensorSum, shape=(), key=309966f894a74b273447a81dd7730330, data=
45>

In [7]: np.inner(a, a)
Out[7]:
Tensor <op=TensorTensorDot, shape=(), key=bca45e00ce32cf77a6cbe9335ca750fd,
data=
285>

In [9]: np.linalg.qr(a[:4, None])
Out[9]:
(Tensor <op=TensorQR, shape=(4, 1), key=acb171fa4c532f281e5030c88a380735, data=
 array([[ 0.        ],
        [-0.26726124],
        [-0.53452248],
        [-0.80178373]])>,
 Tensor <op=TensorQR, shape=(1, 1), key=bdaf2384e07ab9316600e61cf57c61f7, data=
 array([[-3.74165739]])>)
```

Mars tensor API：实现了 70% 常见 Numpy 接口

- 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
  - …

GPU 和 sparse

- GPU 支持，通过指定 gpu=True

```
import mars.tensor as mt

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

- 稀疏支持，指定 sparse=True
  - 可以和 GPU 联用
  - 目前只支持1维和2维张量

```
import mars.tensor as mt

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

外部存储

- Numpy 的 npy 和 npz 文件格式无法适用于并行和分布式场景
- 支持分块数组存储的系统和格式：
  - TileDB（支持 HDFS 等存储，Mars 已经支持）
  - Zarr
  - HDF5
  - netCDF

外部存储

- Numpy 的 npy 和 npz 文件格式无法适用于并行和分布式场景
- 支持分块数组存储的系统和格式：
  - TileDB（支持 HDFS 等存储，Mars 已经支持）
  - Zarr
  - HDF5
  - netCDF

```
In [9]: a = np.arange(12).reshape(3, 4)

In [10]: b = mt.array(a, chunk_size=2)

In [11]: b.totiledb('tensor_b').execute()
Out[11]: array([], shape=(0, 0), dtype=int64)

In [12]: mt.fromtiledb('tensor_b')[:3, :3].execute()
Out[12]:
array([[ 0,  1,  2],
       [ 4,  5,  6],
       [ 8,  9, 10]])
```

更少的内存占用和充分并行


```
In [4]: %%time
   ...: a = np.random.rand(15000, 15000)
   ...: ((a + 1) * 2).mean()
   ...:
   ...:
CPU times: user 4.96 s, sys: 1.49 s, total: 6.45 s
Wall time: 4.25 s
In [4] used 1718.0039 MiB RAM in 4.35s, peaked 1716.61 MiB above current, total RAM
usage 1775.18 MiB
```

```
In [4]: %%time
   ...: a = mt.random.rand(15000, 15000)
   ...: ((a + 1) * 2).mean().execute()
   ...:
   ...:
CPU times: user 7.35 s, sys: 3.97 s, total: 11.3 s
Wall time: 1.98 s
In [4] used 456.5508 MiB RAM in 2.08s, peaked 1019.80 MiB above current, total RAM
usage 576.78 MiB
```

更少的内存占用和充分并行


```
In [4]: %%time
   ...: a = np.random.rand(15000, 15000)
   ...: ((a + 1) * 2).mean()
   ...:
   ...:
CPU times: user 4.96 s, sys: 1.49 s, total: 6.45 s
Wall time: 4.25 s
In [4] used 1718.0039 MiB RAM in 4.35s, peaked 1716.61 MiB above current, total RAM
usage 1775.18 MiB
```

并行

```
In [4]: %%time
   ...: a = mt.random.rand(15000, 15000)
   ...: ((a + 1) * 2).mean().execute()
   ...:
   ...:
CPU times: user 7.35 s, sys: 3.97 s, total: 11.3 s
Wall time: 1.98 s
In [4] used 456.5508 MiB RAM in 2.08s, peaked 1019.80 MiB above current, total RAM
usage 576.78 MiB
```

更少的内存占用和充分并行


```
In [4]: %%time
   ...: a = np.random.rand(15000, 15000)
   ...: ((a + 1) * 2).mean()
   ...:
   ...:
CPU times: user 4.96 s, sys: 1.49 s, total: 6.45 s
Wall time: 4.25 s
In [4] used 1718.0039 MiB RAM in 4.35s, peaked 1716.61 MiB above current, total RAM
usage 1775.18 MiB
```

并行

更少的内存

```
In [4]: %%time
   ...: a = mt.random.rand(15000, 15000)
   ...: ((a + 1) * 2).mean().execute()
   ...:
   ...:
CPU times: user 7.35 s, sys: 3.97 s, total: 11.3 s
Wall time: 1.98 s
In [4] used 456.5508 MiB RAM in 2.08s, peaked 1019.80 MiB above current, total RAM
usage 576.78 MiB
```

Graph：粗粒度（Tensor）和细粒度（Chunk）图

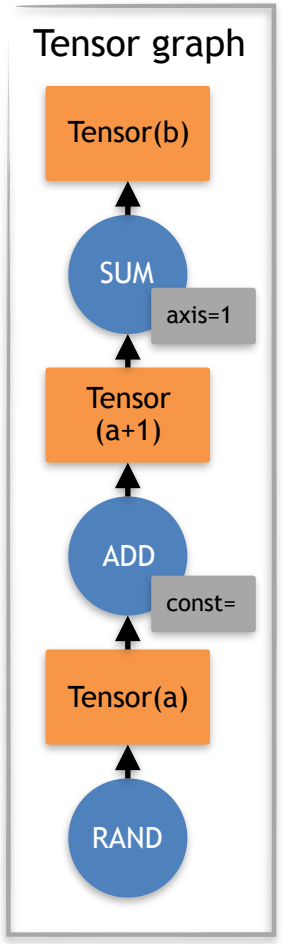

```
import mars.tensor as mt

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

服务端

客户端

Graph：粗粒度（Tensor）和细粒度（Chunk）图


```
import mars.tensor as mt

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

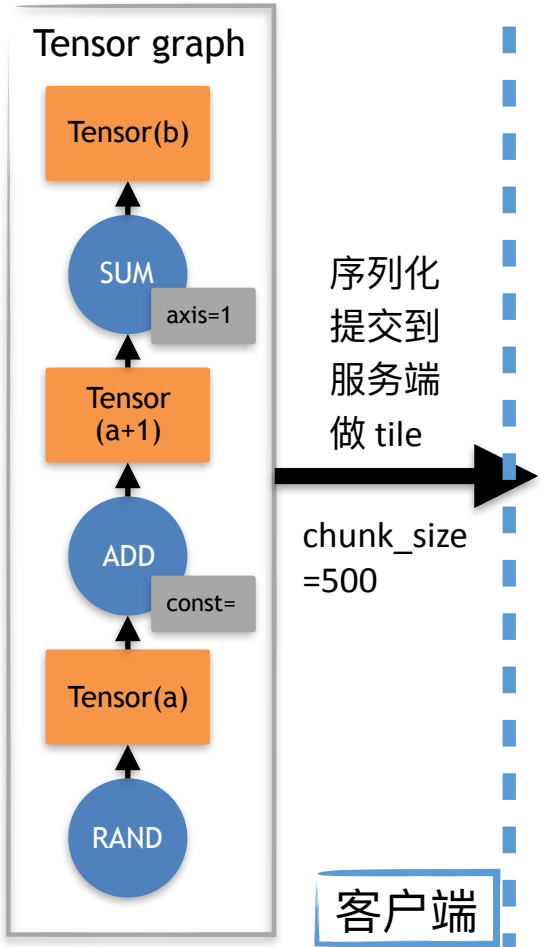


服务端

客户端

Graph：粗粒度（Tensor）和细粒度（Chunk）图


```
import mars.tensor as mt

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



序列化
提交到
服务端
做 tile

chunk_size
=500

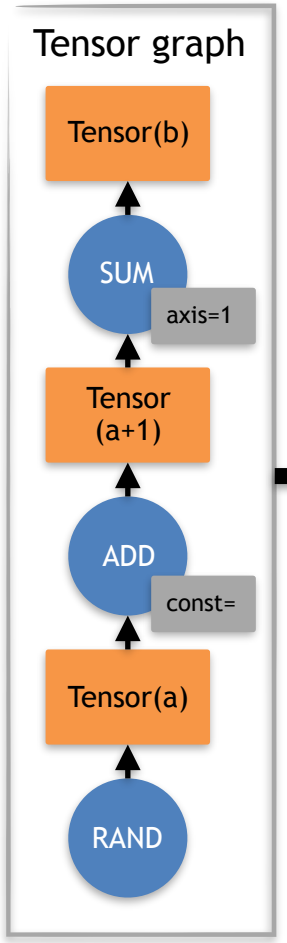
服务端

客户端

Graph：粗粒度（Tensor）和细粒度（Chunk）图

2019 让开发者紧跟技术潮流
Python开发者日

```
import mars.tensor as mt

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

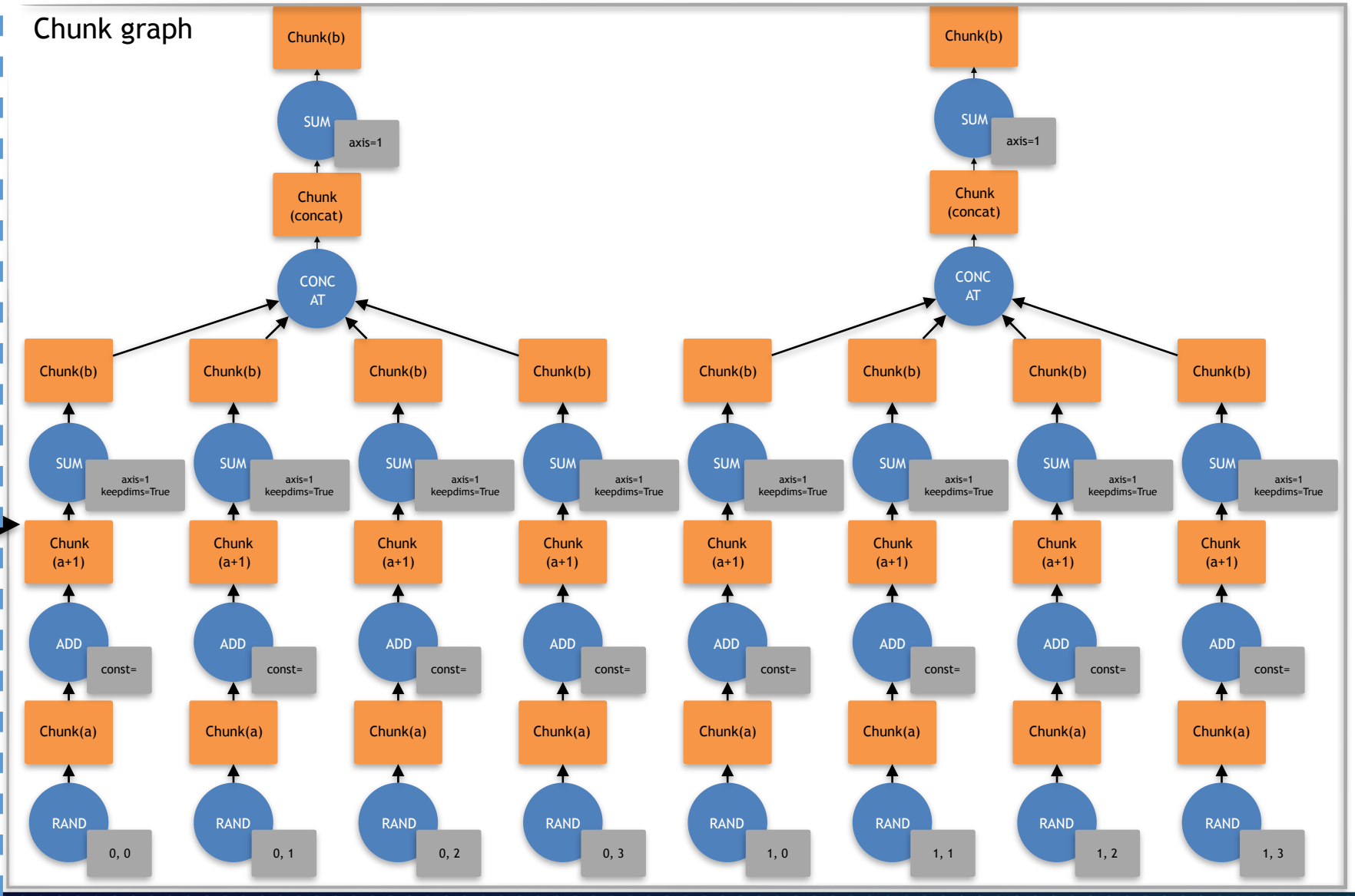


序列化
提交到
服务端
做 tile

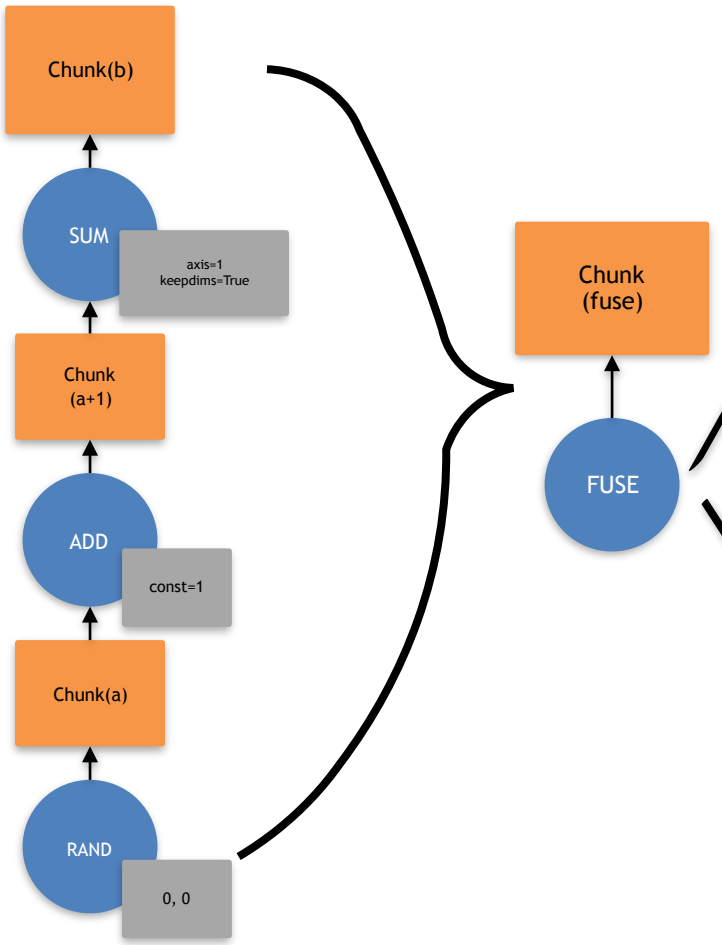
chunk_size
=500

客户端

服务端



优化：操作合并




```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)
```

优化：剪枝

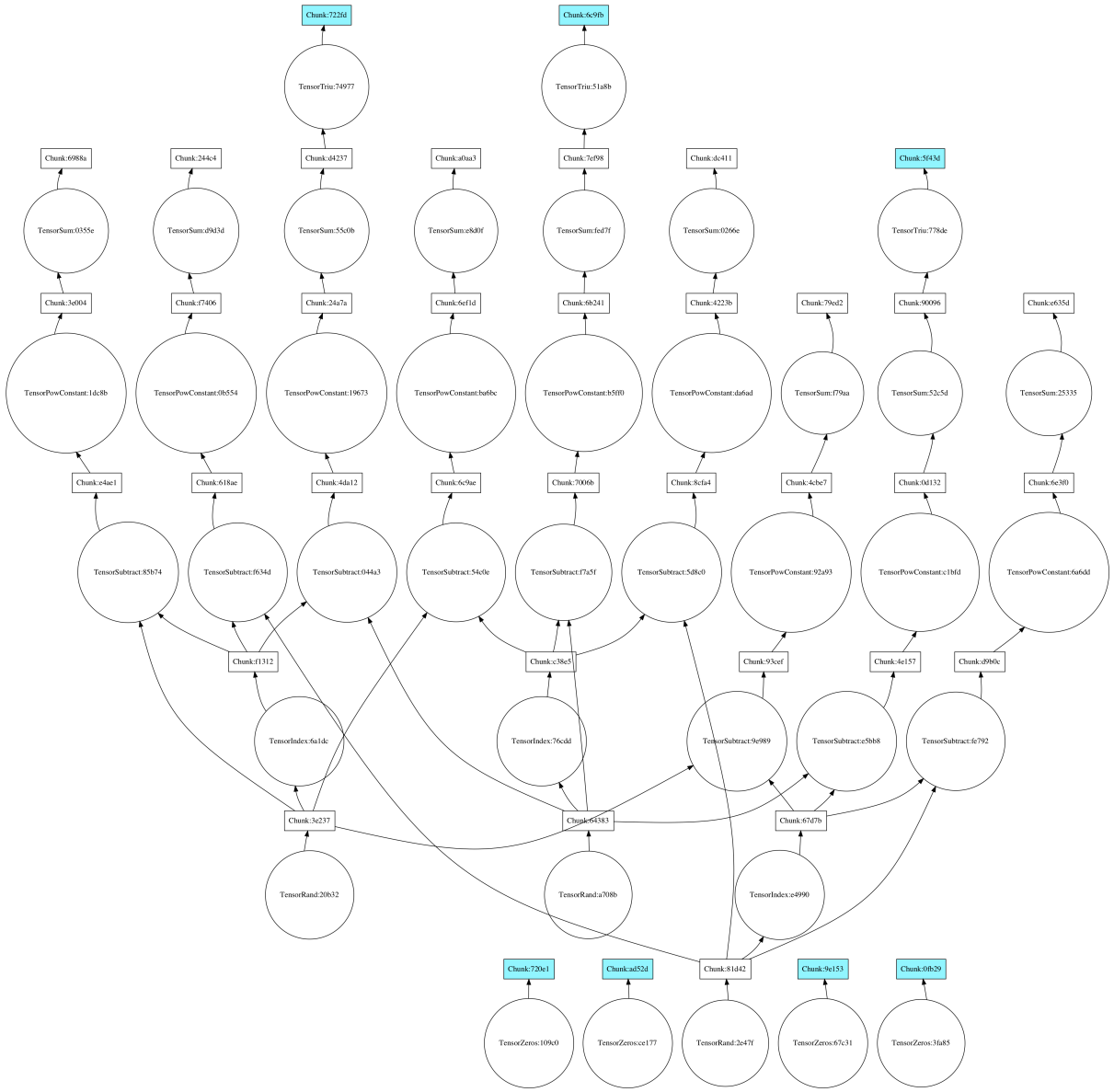

```
a = mt.random.rand(5000, 10, chunk_size=2000)
b = ((a[:, mt.newaxis, :] - a) ** 2).sum(axis=-1)
c = mt.triu(b)
```

- 取上三角矩阵对 Numpy 来说对性能提升没有帮助
- 对 Mars 意义重大，对于处于下三角的 Operand 直接替换成 ones

优化：剪枝


```
a = mt.random.rand(5000, 10, chunk_size=2000)
b = ((a[:, mt.newaxis, :] - a) ** 2).sum(axis=-1)
c = mt.triu(b)
```

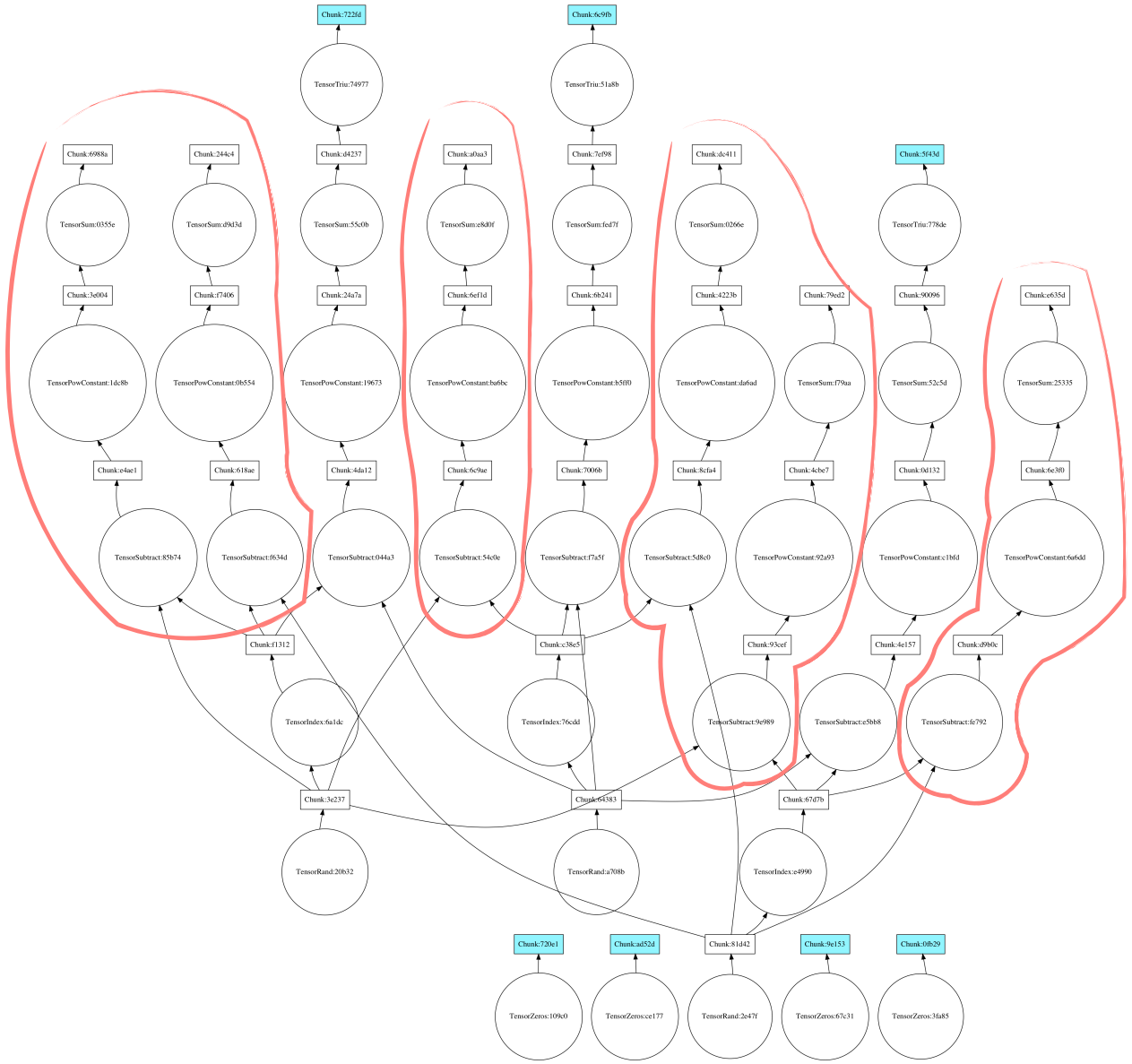
- 取上三角矩阵对 Numpy 来说对性能提升没有帮助
- 对 Mars 意义重大，对于处于下三角的 Operand 直接替换成 ones



优化：剪枝


```
a = mt.random.rand(5000, 10, chunk_size=2000)
b = ((a[:, mt.newaxis, :] - a) ** 2).sum(axis=-1)
c = mt.triu(b)
```

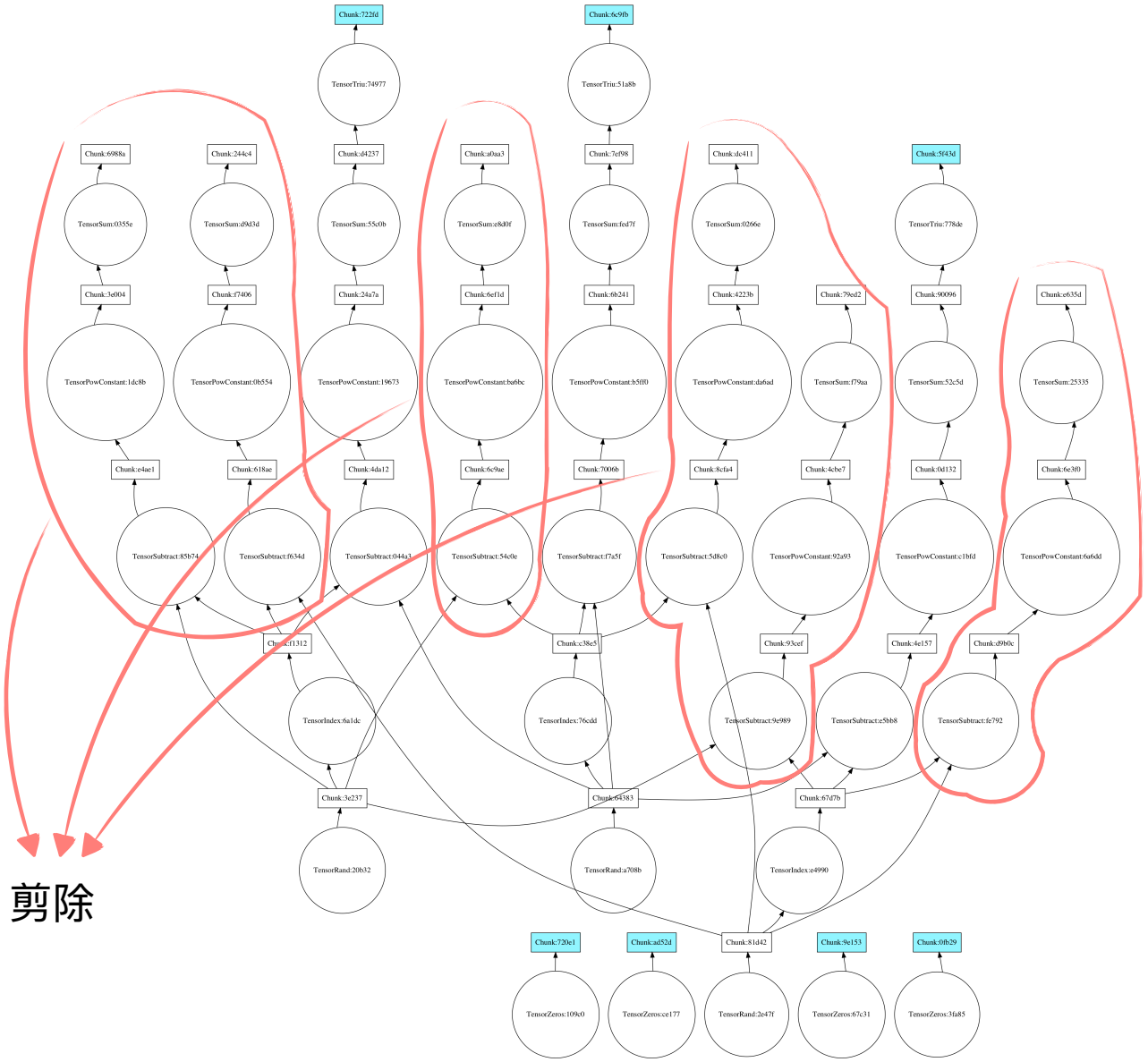
- 取上三角矩阵对 Numpy 来说对性能提升没有帮助
- 对 Mars 意义重大，对于处于下三角的 Operand 直接替换成 ones



优化：剪枝

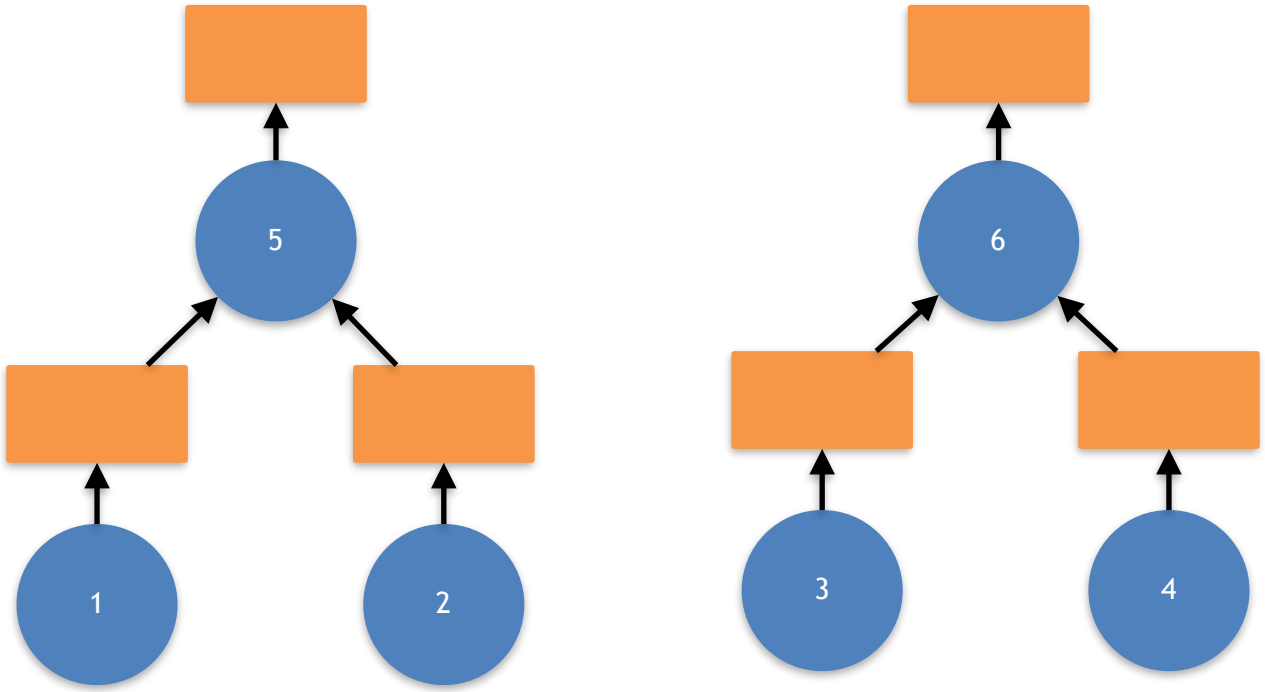

```
a = mt.random.rand(5000, 10, chunk_size=2000)
b = ((a[:, mt.newaxis, :] - a) ** 2).sum(axis=-1)
c = mt.triu(b)
```

- 取上三角矩阵对 Numpy 来说对性能提升没有帮助
- 对 Mars 意义重大，对于处于下三角的 Operand 直接替换成 ones



调度

- 初始节点分配很重要
- 执行顺序深度优先



Mars tensor 的挑战

Mars tensor 的挑战

- chunk_size 对性能影响巨大，怎么切分张量往往对性能影响很大

Mars tensor 的挑战

- chunk_size 对性能影响巨大，怎么切分张量往往对性能影响很大
- 更好的 fuse 优化，如 (a[:, mt.newaxis, :] - a).sum(axis=-1) 会产生巨大的中间过程膨胀

Mars tensor 的挑战

- chunk_size 对性能影响巨大，怎么切分张量往往对性能影响很大
- 更好的 fuse 优化，如 (a[:, mt.newaxis, :] - a).sum(axis=-1) 会产生巨大的中间过程膨胀
- 分布式对 GPU 的支持还有所欠缺

性能

1

2

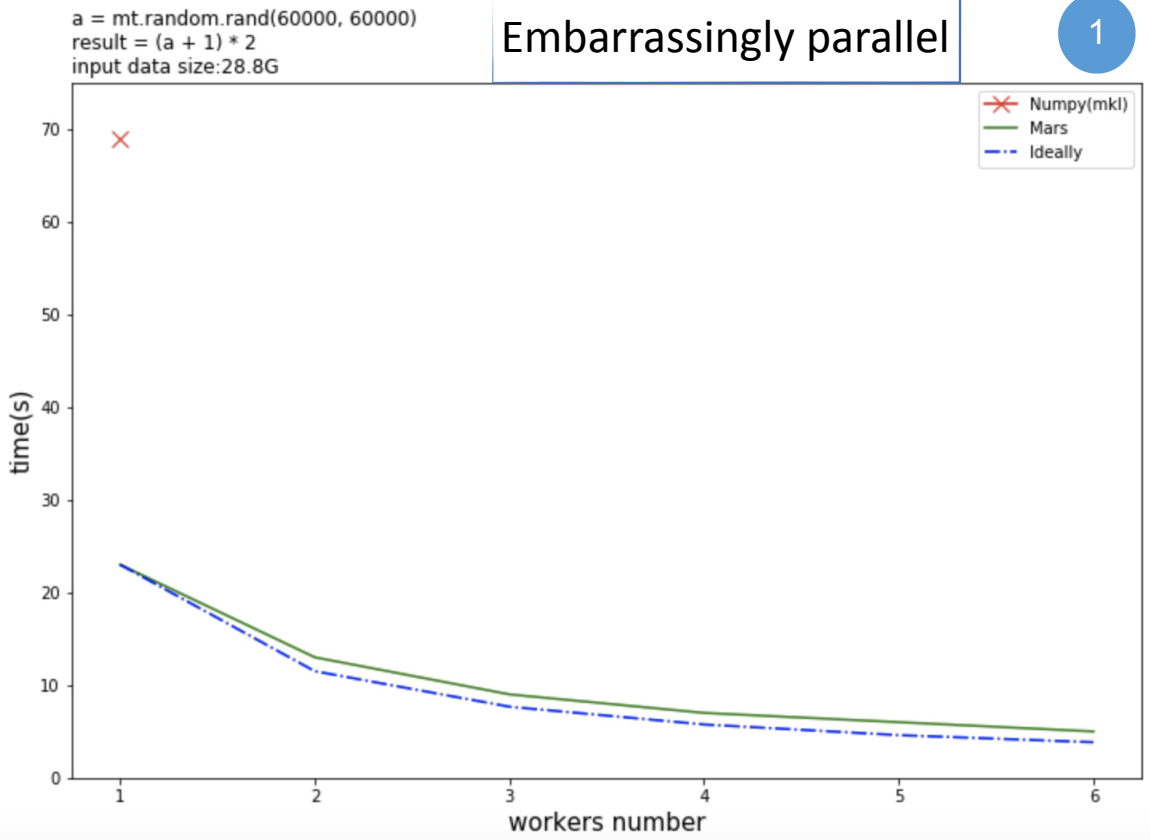
3

性能

Embarrassingly parallel

1

2

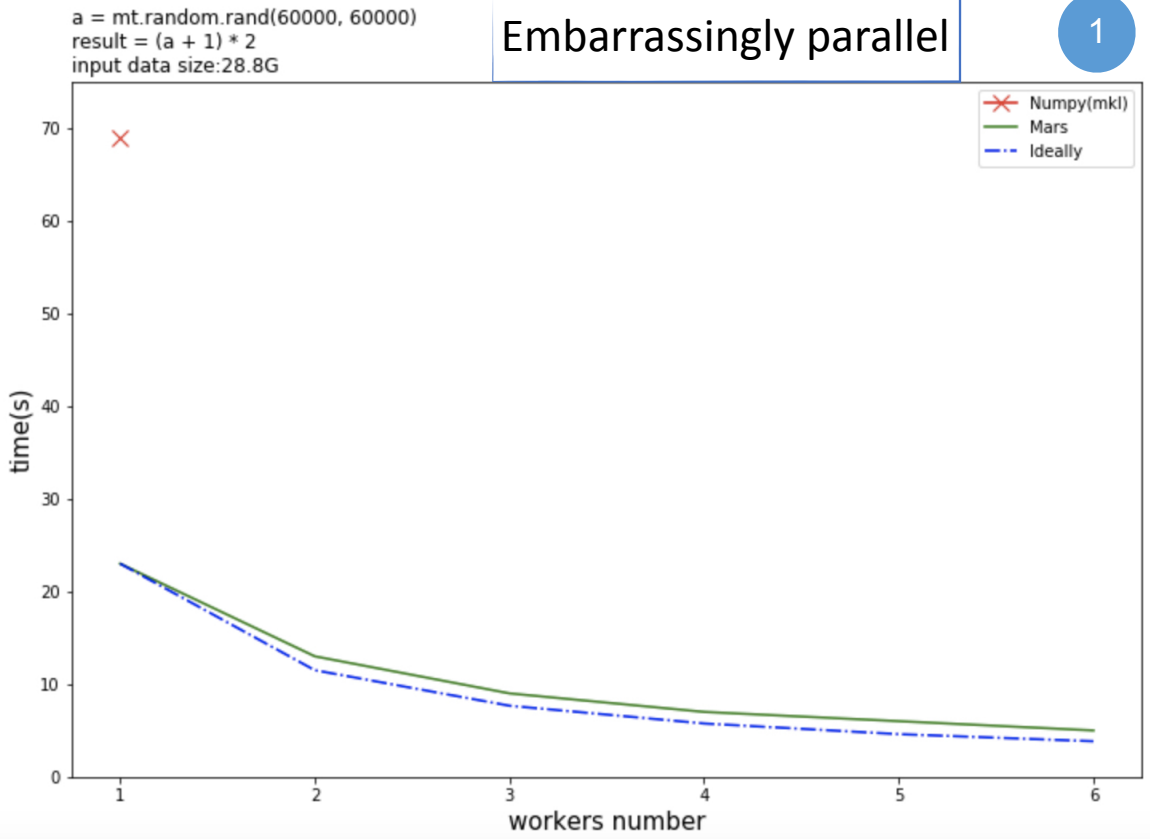


3

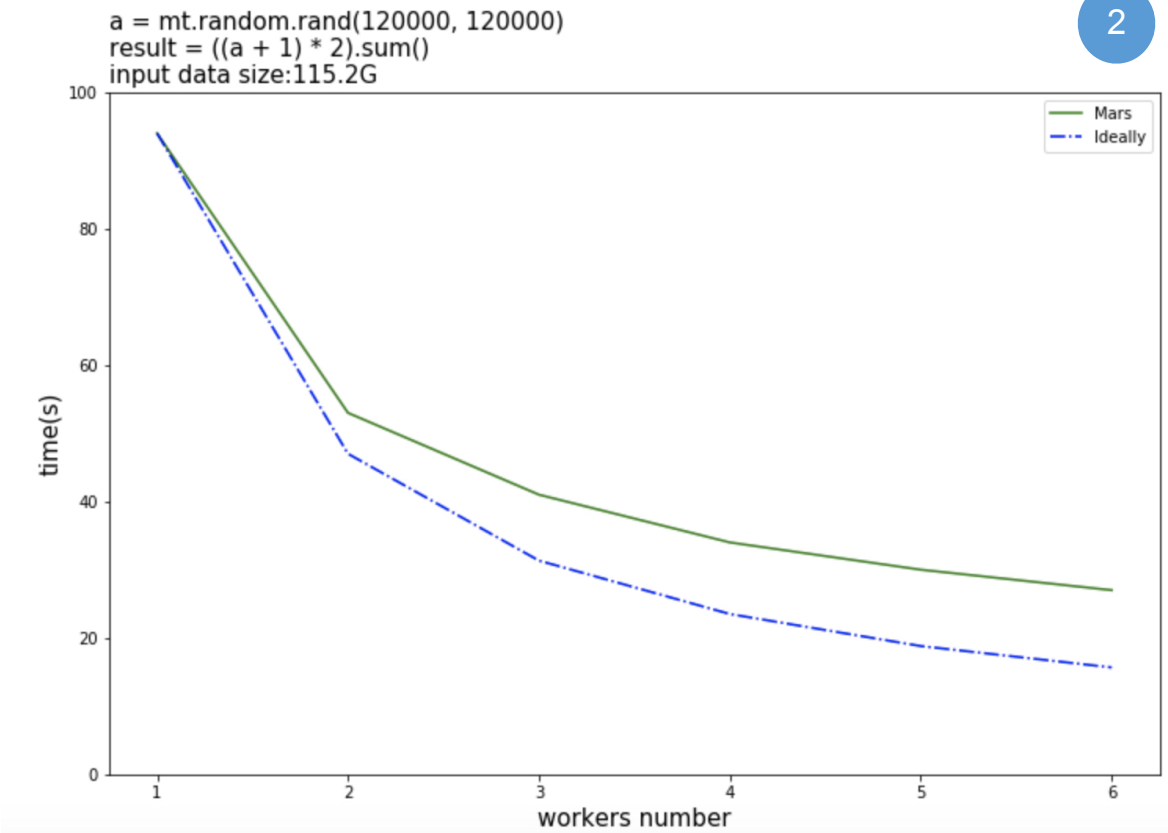
性能

Embarrassingly parallel

1



2

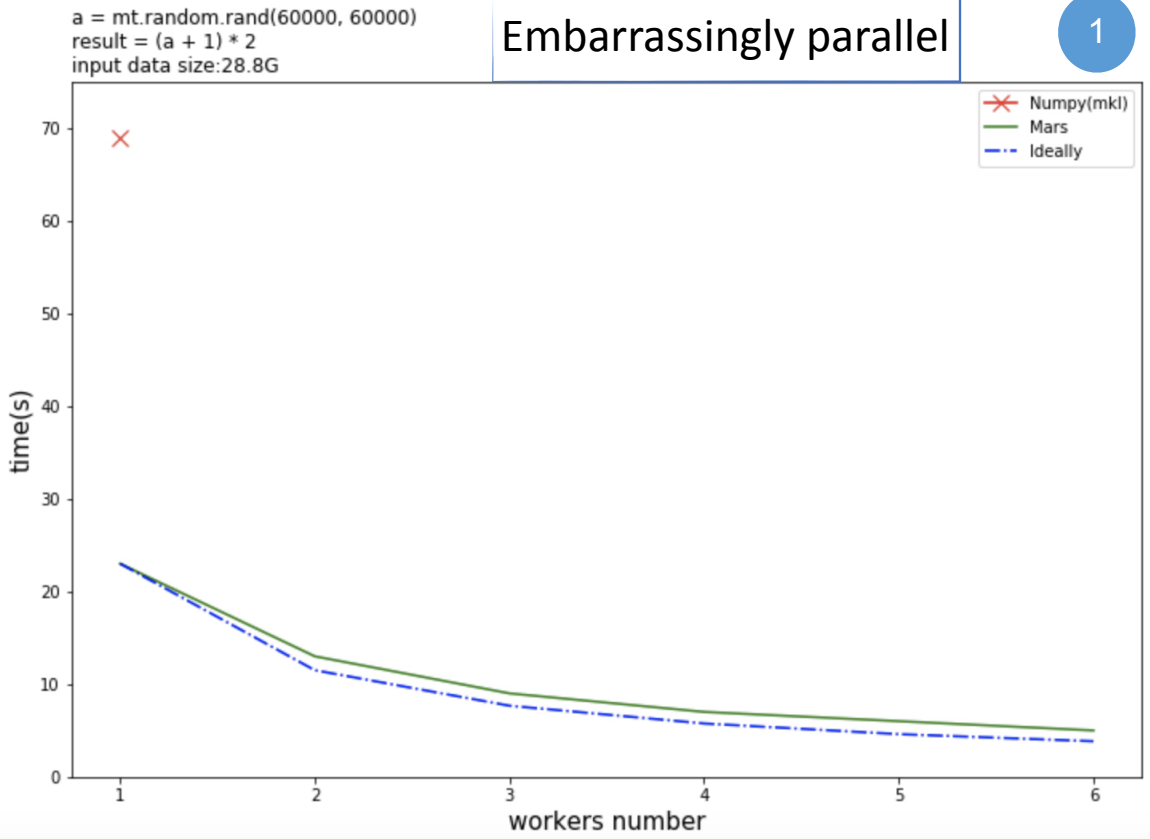


3

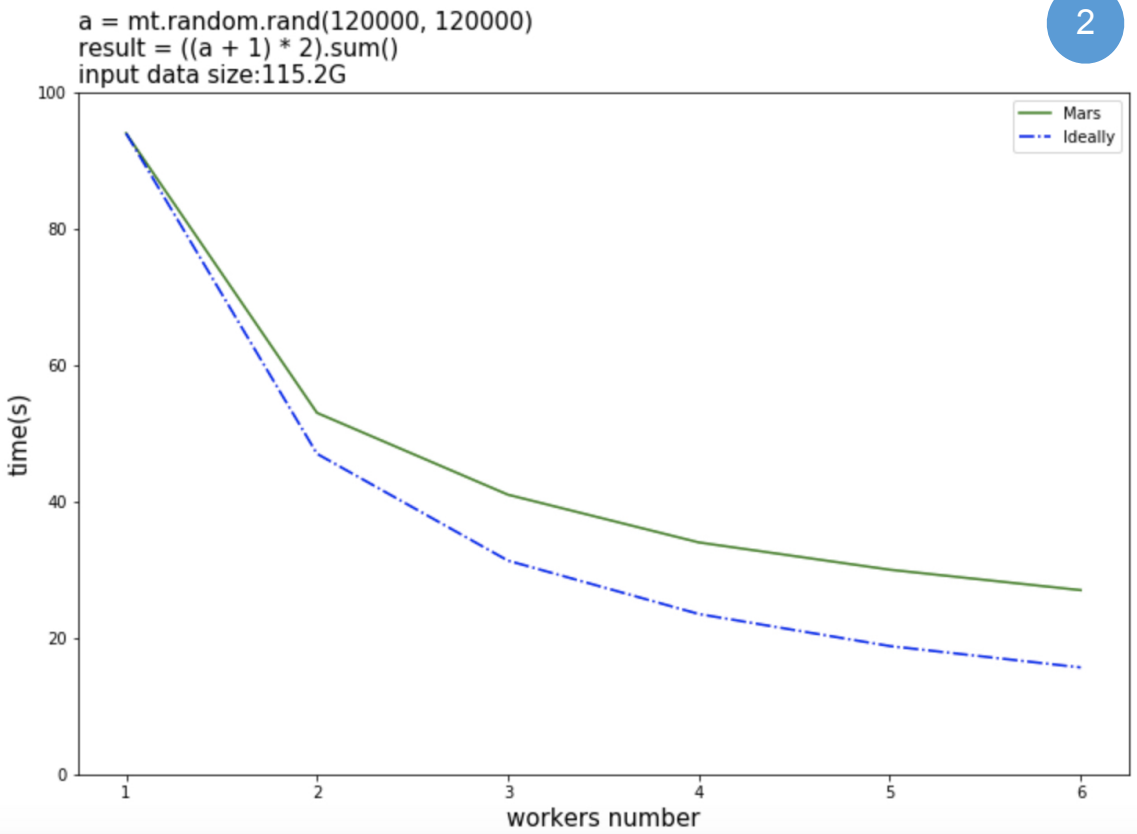
性能

Embarrassingly parallel

1



2



- 核心：千亿数据，2.25T大小矩阵乘以千亿数据、2.25T大小矩阵
- MapReduce：千行代码，9000CU，10个小时
- Mars：5行代码，1600CU，2.5小时

3

- https://github.com/mars-project/mars

- https://github.com/mars-project/mars
- https://docs.mars-project.io/zh_CN/latest/

- https://github.com/mars-project/mars
- https://docs.mars-project.io/zh_CN/latest/
- 开源4个月，1300+ star

2019 让开发者紧跟技术潮流
Python开发者日

- https://github.com/mars-project/mars
- https://docs.mars-project.io/zh_CN/latest/
- 开源4个月，1300+ star
- 点 star 不迷路，欢迎使用，开 issue 来 report bug，能贡献代码就更好了

Mars 下一步计划

Mars 下一步计划

- 社区对我们很重要

Mars 下一步计划

- 社区对我们很重要
- 开发计划：

Mars 下一步计划

- 社区对我们很重要
- 开发计划：
  - Mars tensor：支持更多 Numpy 和 Scipy 接口

Mars 下一步计划

- 社区对我们很重要
- 开发计划：
  - Mars tensor：支持更多 Numpy 和 Scipy 接口
  - Mars DataFrame：分布式 pandas，保持和 pandas 接口兼容，且支持 GPU（开发中）

Mars 下一步计划

- 社区对我们很重要
- 开发计划：
  - Mars tensor：支持更多 Numpy 和 Scipy 接口
  - Mars DataFrame：分布式 pandas，保持和 pandas 接口兼容，且支持 GPU（开发中）
  - Mars learn：分布式 scikit-learn、XGBoost 等

Mars 下一步计划

- 社区对我们很重要
- 开发计划：
  - Mars tensor：支持更多 Numpy 和 Scipy 接口
  - Mars DataFrame：分布式 pandas，保持和 pandas 接口兼容，且支持 GPU（开发中）
  - Mars learn：分布式 scikit-learn、XGBoost 等
  - Mars remote：远程执行 Python 函数和类

Mars 下一步计划

- 社区对我们很重要
- 开发计划：
  - Mars tensor：支持更多 Numpy 和 Scipy 接口
  - Mars DataFrame：分布式 pandas，保持和 pandas 接口兼容，且支持 GPU（开发中）
  - Mars learn：分布式 scikit-learn、XGBoost 等
  - Mars remote：远程执行 Python 函数和类
  - 多语言支持

我的联系方式

- Github：@chineking
- 微博：@秦续业
- 微信：右侧二维码

秦续业

浙江 杭州

扫一扫上面的二维码图案，加我微信

CSDN

2019
Python开发者日

让开发者紧跟技术潮流

Thank you