巨大なテキストデータを保存・復元する

～最強のPickle編～

機械学習の結果など

```
[
    {'id1': ['data1': 0.65442,
             'data2': 1.32543,
             ...
            ]
    },
    {'id2': ['data1': -3.15776,
             'data2': 5.32543,
             ...
            ]
    },
    {'id3': ['data1': 6.23147,
             'data2': 0.44531,
             ...
            ]
    }, ...
]
```

よくPickleで保存する

```
import pickle
with open("large_data.pkl", "wb") as f:
    pickle.dump(data, f)
```

巨大なPickleファイルを読み込むと...

```
import pickle
with open("large_data.pkl", "rb") as f:
    data = pickle.load(f)
```

遅すぎる！！

```
178.57 sec
```

他にも、

散々待たされて、メモリエラー

```
*** set a breakpoint in malloc_error_break to debug
python(3716,0xa08ed1d4) malloc: *** mach_vm_map(size=1048576) f
*** error: can't allocate region securely
*** set a breakpoint in malloc_error_break to debug
Traceback (most recent call last):
  File "/System/Library/Frameworks/Python.framework/Versions/2.
    dispatch[key](self)
  File "/System/Library/Frameworks/Python.framework/Versions/2.
    self.stack.append({})
MemoryError
```

そもそもPickleってなに？

Pickleとは

- オブジェクトをバイト列に変換する方法（Serialization）
  - 例：dict型の変数 -> バイナリファイル
  - 例：定義したクラスインスタンス -> バイナリファイル
- 直接ファイルを開いても読めない
- Pythonにしかない
- 機械学習モデルの保存によく使われる

Pickleを早くする方法を調べました

Pickleを早くする方法４つ

- `cPickle` モジュールを使う
- `protocol=4` を使う
- `fast=True` を使う
- `pickletools.optimize()` を使う

1つ目：`cPickle`

- 標準のpickleをC言語で書き直したやつ
  - `pickle`：Pythonで書いてある
  - `cPickle`：C言語で書いてある
- 使い方

```
import pickle
```

↓

```
import cPickle # Python2系
import _pickle # Python3系
```

2つ目：`protocol=4`

- pickleモジュールの最新プロトコルを使う
  - `protocol=0`: python2系のデフォルト
  - `protocol=3`: python3系のデフォルト、2系では使えない
  - `protocol=4`: 巨大オブジェクトをサポート
- 使い方

```
pickle.dump(x, f)
```

↓

```
pickle.Pickler(f, protocol=4).dump(x)
```

3つ目：`fast=True`

- 余分な PUT 命令コードを生成しなくなる
- 公式によると廃止予定
- 使い方

```
pickle.dump(data)
```

↓

```
p = pickle.Pickler(f)
p.fast = True # ここ
p.dump(data)
```

4つ目：`pickletools.optimize()`

- 余分な PUT 命令コードを生成しなくなる（`fast=True` と同じ）
- 使い方

```
pickle.dump(data)
```

↓

```
import pickletools

pickled = pickle.dumps(data)
opt = pickletools.optimize(pickled)
pickle.dump(opt, f)
```

比較してみた

比較するもの

- 時間
  - `dump` にかかる時間
  - `load` にかかる時間
- 最大メモリ使用量
  - `dump` に使うメモリ
  - `load` に使うメモリ
- `dump` されたファイルサイズ

使うデータ

大体１GBのデータ

```
[
    {
     'id': 1,
     'data': ['data11', 'data12', ..., 'data1100']
    },
    ...
    {
     'id': 500000,
     'data': ['data5000001', ..., 'data500000100']
    }
]
```

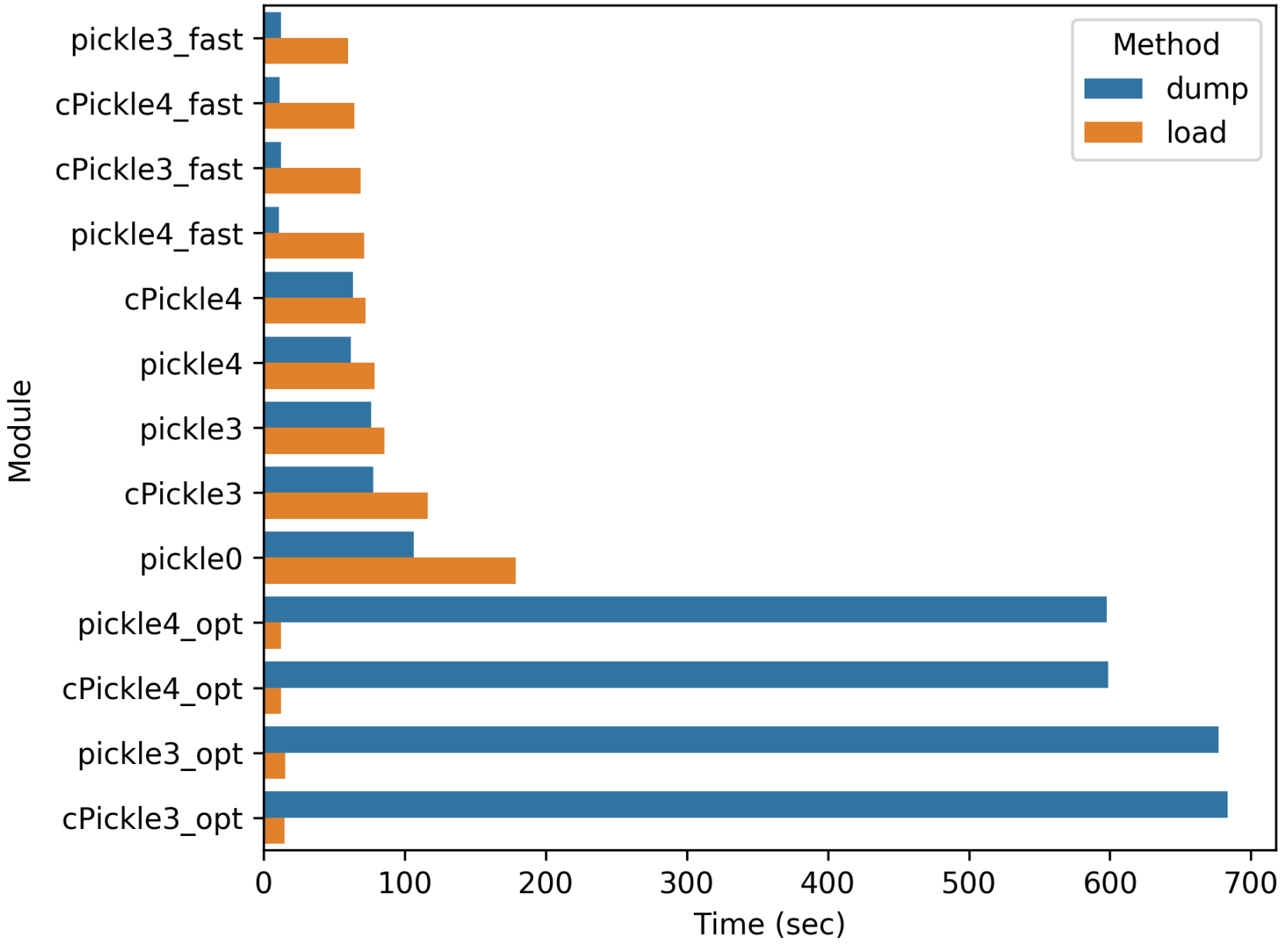
環境

- Python 3.7.1
- MacBook Air (Mid 2013)

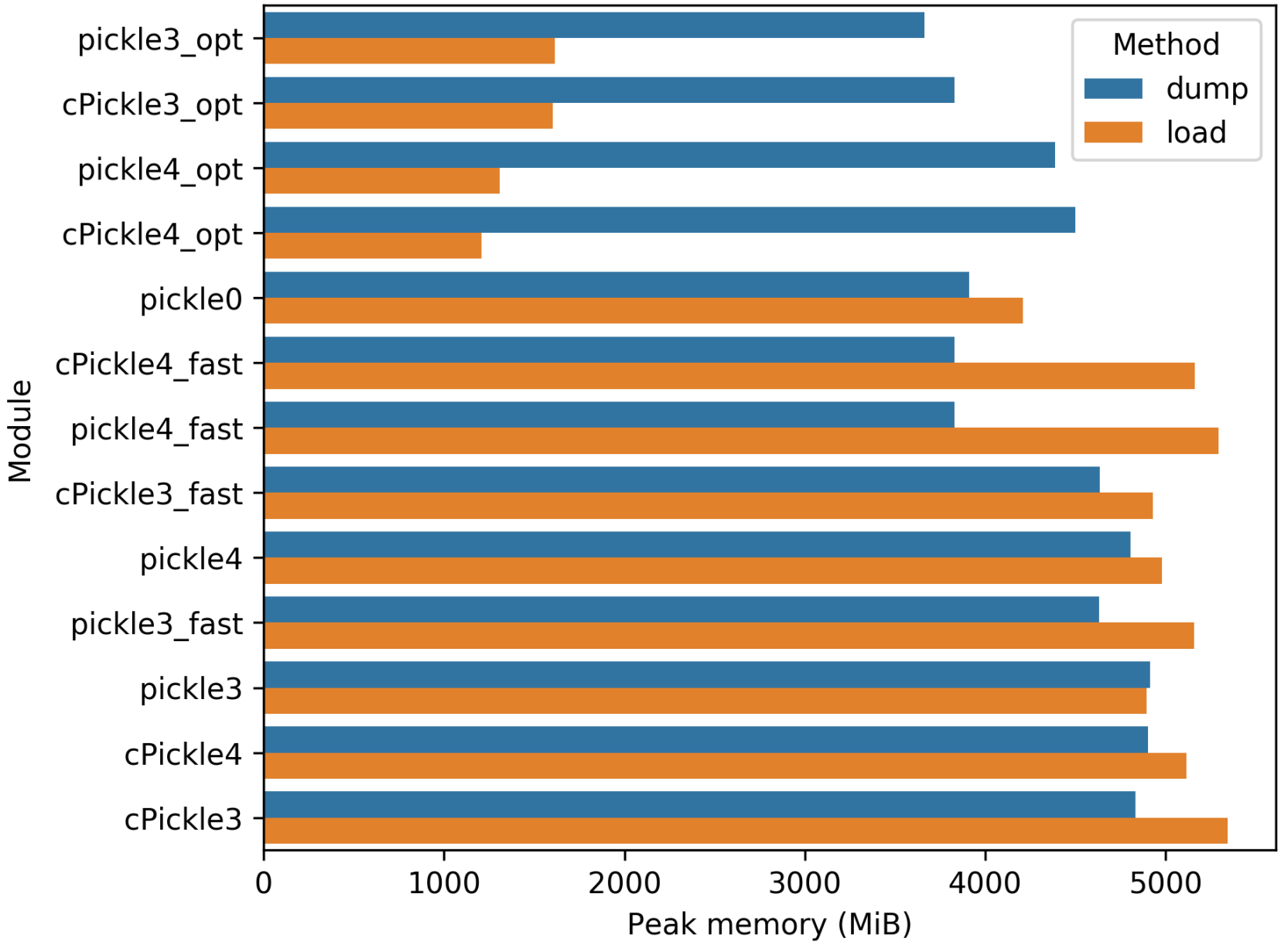
```
Processor: Intel Core i5 1.3 GHz
Memory: 8 GB
```

結果

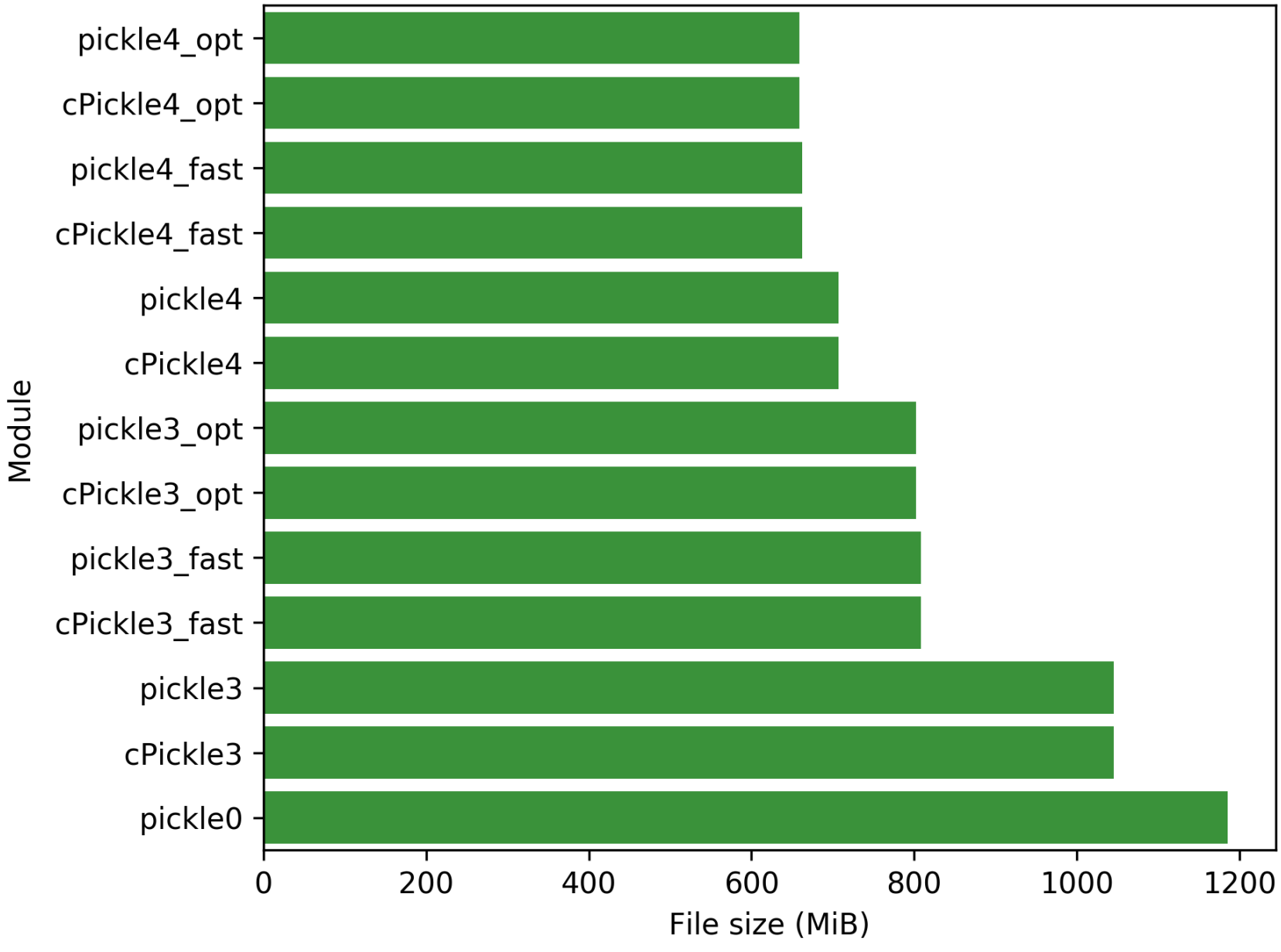
時間（合計の小さい順）


Method
dump
load
pickle3_fast
cPickle4_fast
cPickle3_fast
pickle4_fast
cPickle4
pickle4
pickle3
cPickle3
pickle0
pickle4_opt
cPickle4_opt
pickle3_opt
cPickle3_opt
Module
0
100
200
300
400
500
600
700
Time (sec)

最大メモリ使用量（合計の小さい順）

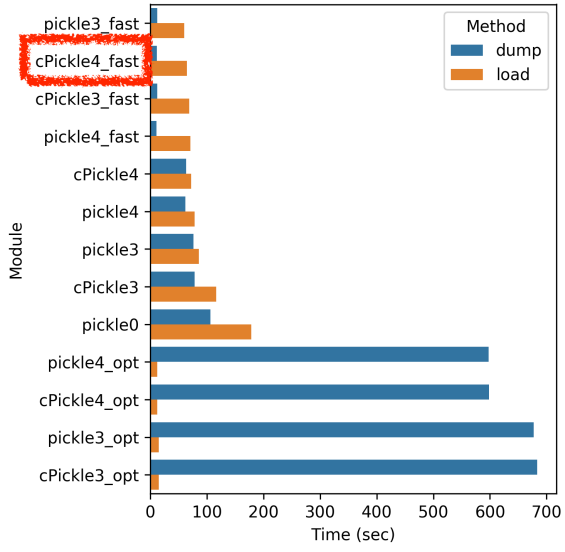


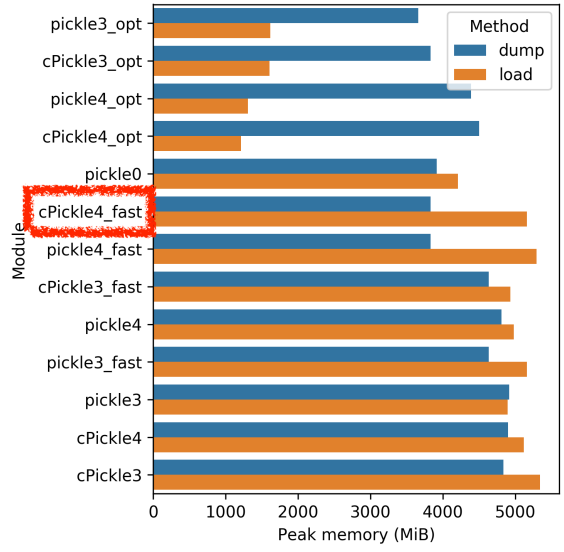
ファイルサイズ（小さい順）

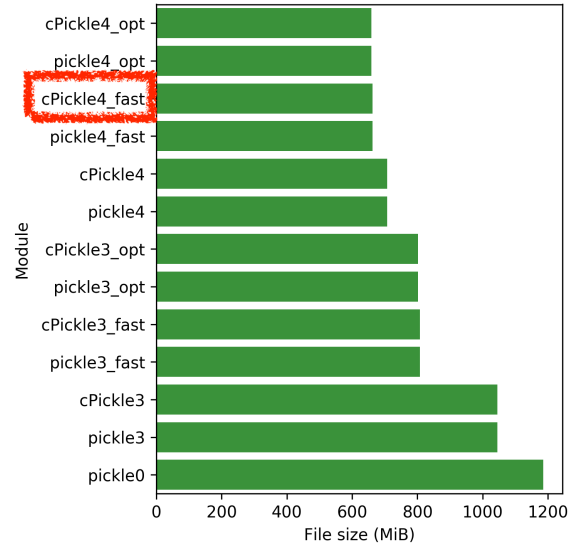


総合優勝

cPickle4_fast

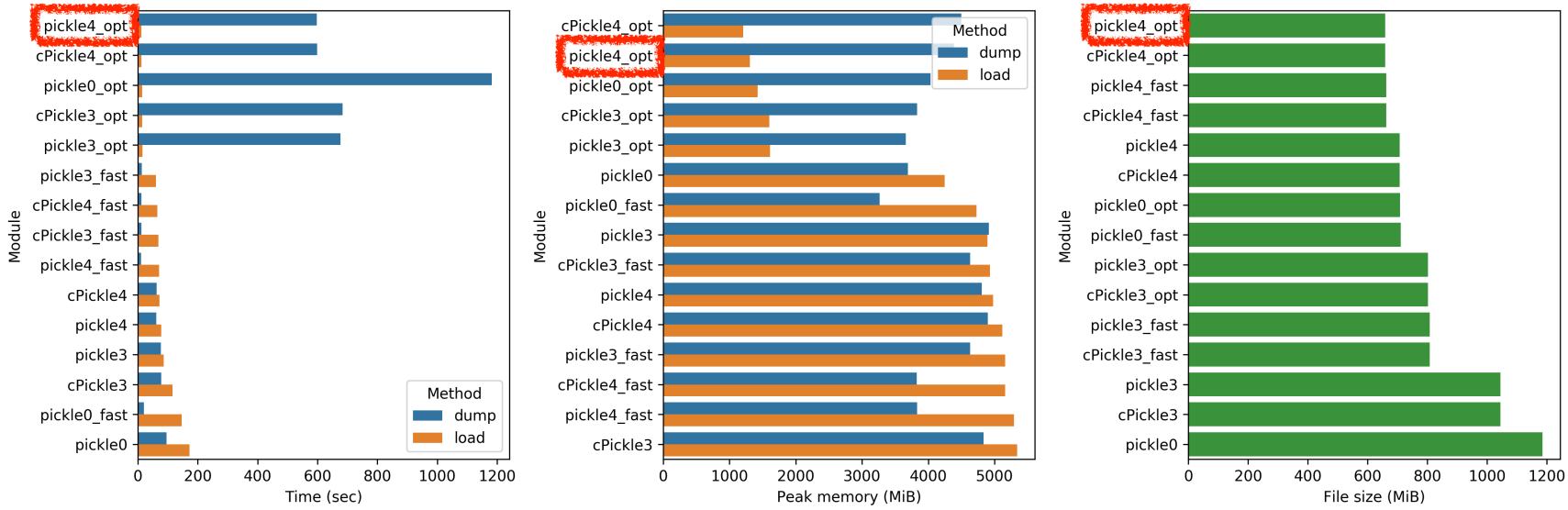






`load` 最速賞（`dump` 時間は無視）

pickle4_opt



まとめ

- `load` と `dump` かなり早い、メモリふつう、ファイルほぼ最小
  ただし廃止予定

```
import _pickle as cPickle

p = cPickle.Pickler(f, protocol=4)
p.fast = True
p.dump(data)
```

- `dump` 激遅、 `load` 時に最速・メモリ最小、ファイル最小

```
import pickle
import pickletools

p = pickle.Pickler(f, protocol=4)
pickled = p.dumps(data)
opt = pickletools.optimize(pickled)
pickle.dump(opt, f)
```

次回

巨大なテキストデータを保存・復元する

～最強のSerializationモジュール編～

```
pickle
cPickle
json
csv
msgpack
hdfstore
marshal
dill
cloudpickle
hickle
...
```