

# [544] Cache-Friendly Code

Tyler Caraza-Harter

# Learning Objectives

- write cache-friendly code with Numpy and PyArrow
- use memory mappings via PyArrow to access data that is larger than physical memory
- enable swapping to alleviate memory pressure
- configure Docker memory limits on physical memory used

# Outline

CPU: L1-L3

Demos: Numpy+PyArrow...

Background: Virtual Address Spaces

OS (Operating System): Page Cache

Demos: PyArrow+Docker

# Granularity

If a process reads 1 byte and misses, *how much data should the CPU bring into the cache?*

- **too little:** we'll have many more misses if we read nearby bytes soon
- **too much:** wasteful to load data to cache that might never be accessed

L1-L3 cache data in units called **cache lines**

- modern CPUs typically 64 bytes (for example, 8 int64 numbers)
- M1/M2 uses 128

# Cache Lines and Misses



how many  
misses?

# Cache Lines and Misses



how many  
misses?



how many  
misses?

# Cache Lines and Misses



how many  
misses?



how many  
misses?

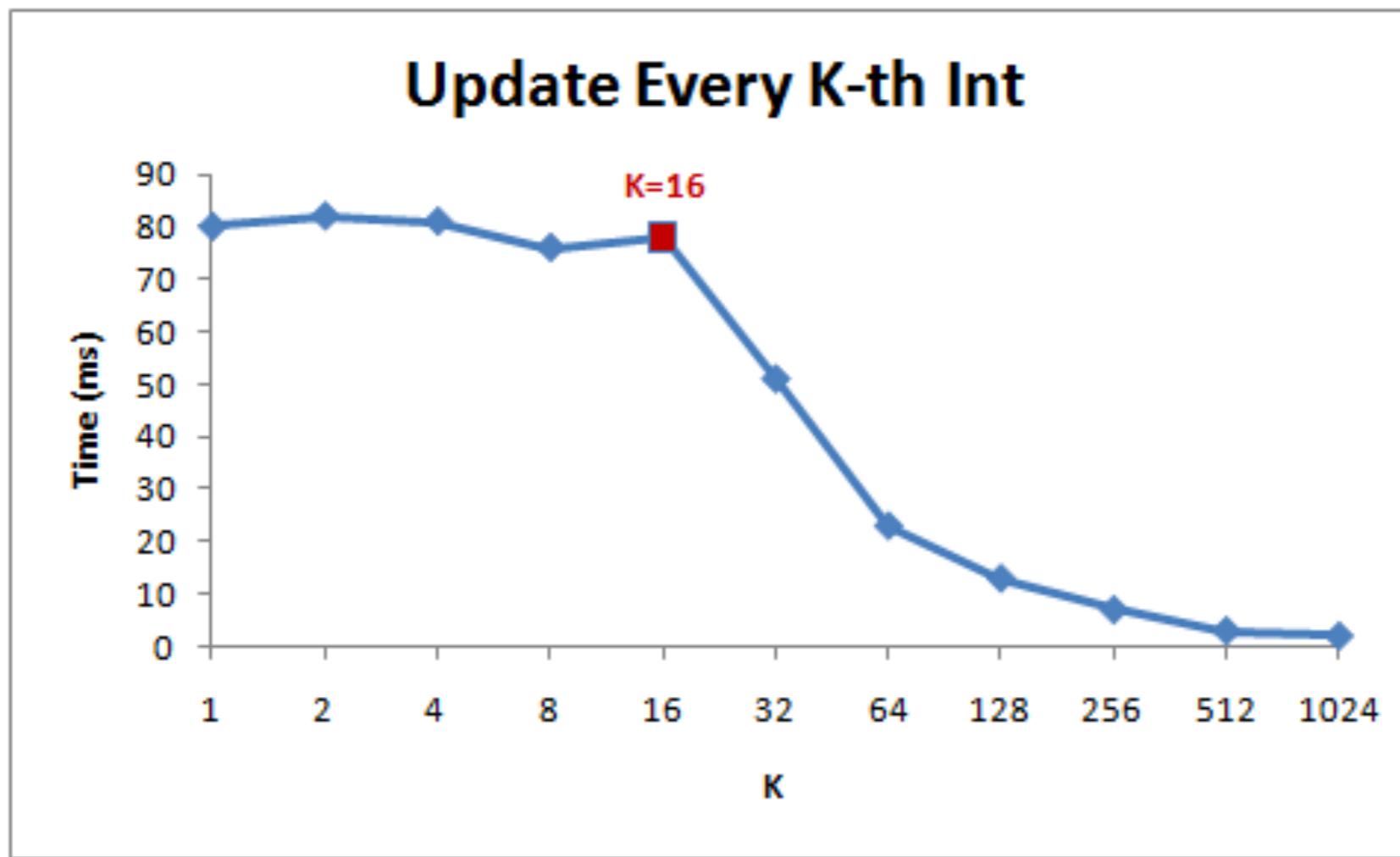


how many  
misses?

# Example 1: Step and Multiply

as K gets bigger, we do fewer multiplications. But does it matter?

```
for (int i = 0; i < arr.Length; i += K) arr[i] *= 3;
```

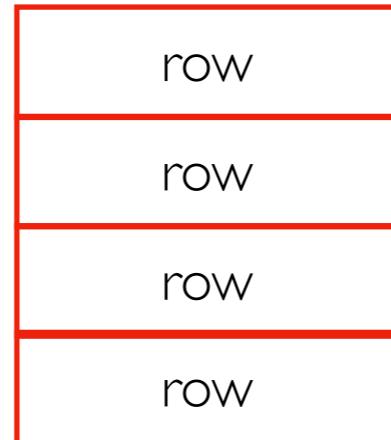


[Gallery of Processor Cache Effects](#)

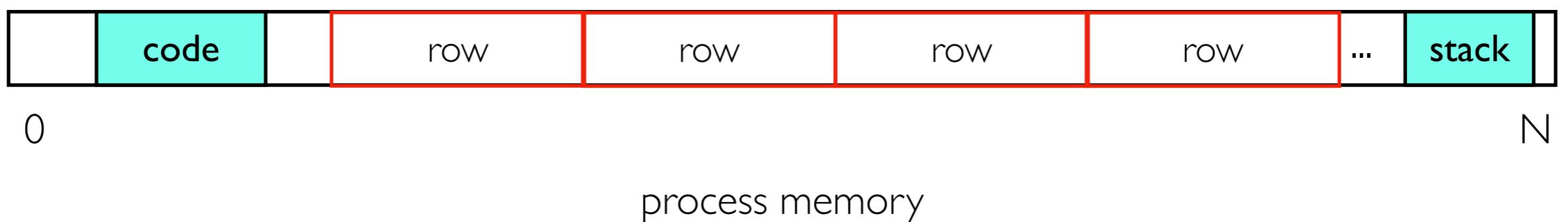
<http://igoro.com/archive/gallery-of-processor-cache-effects/>

# Example 2: Matrices

matrix of numbers  
**logically**, 2-dimensional

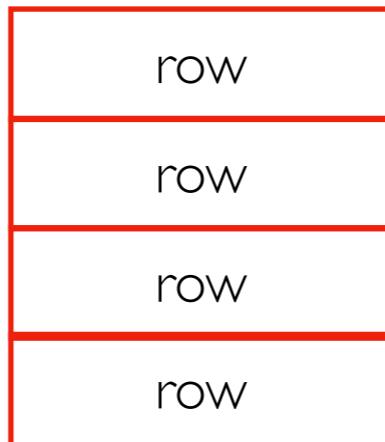


**physically**, those rows are arranged along  
1-dimension in the virtual address space

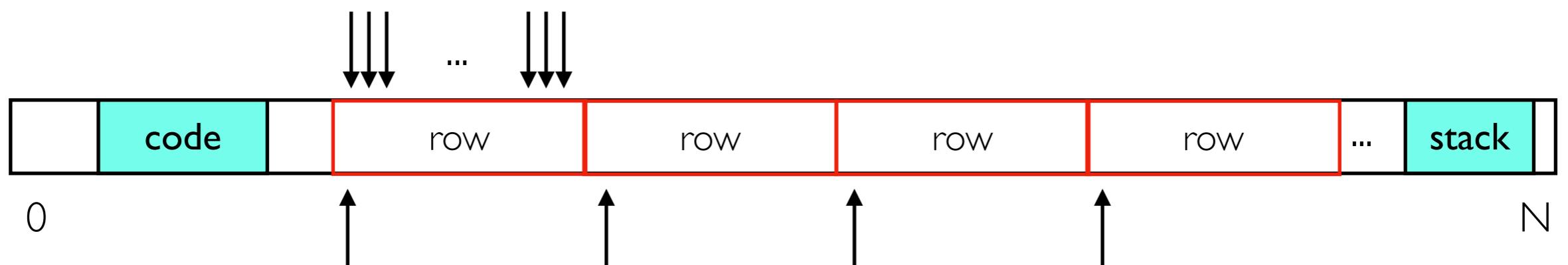


# Example 2: Matrices

matrix of numbers  
**logically**, 2-dimensional



summing over row:  
data consolidated over few cache lines



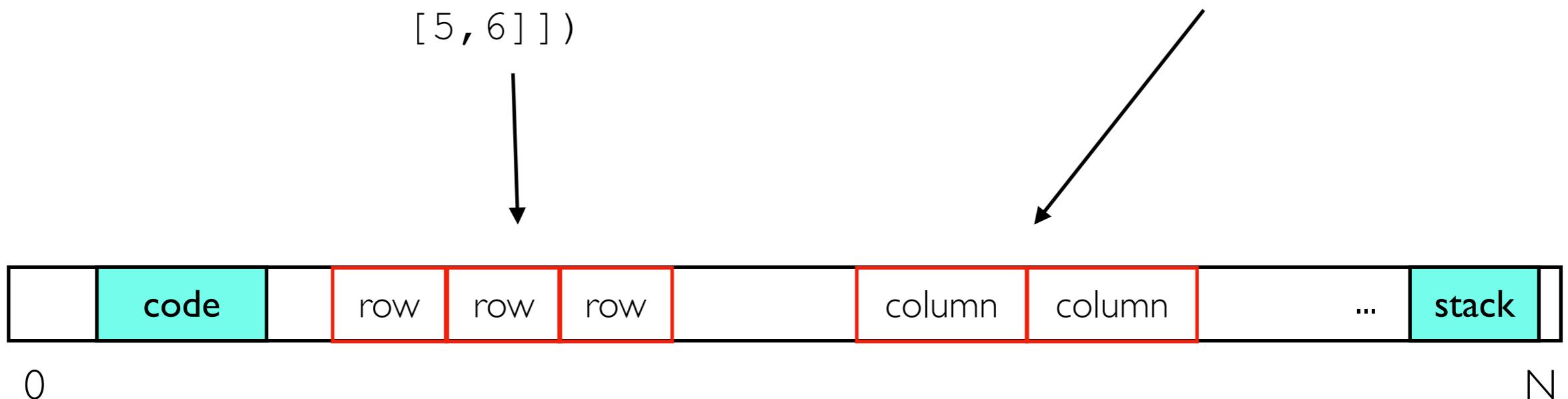
summing over column: each number is in its own cache line and triggers a cache miss

# Numpy: Controlling Layout with Transpose

for efficiency, transpose doesn't actually move/copy data, meaning we can get fast column sum by (a) putting column data in rows and (b) transposing

```
np.array([[1, 2],  
         [3, 4],  
         [5, 6]])
```

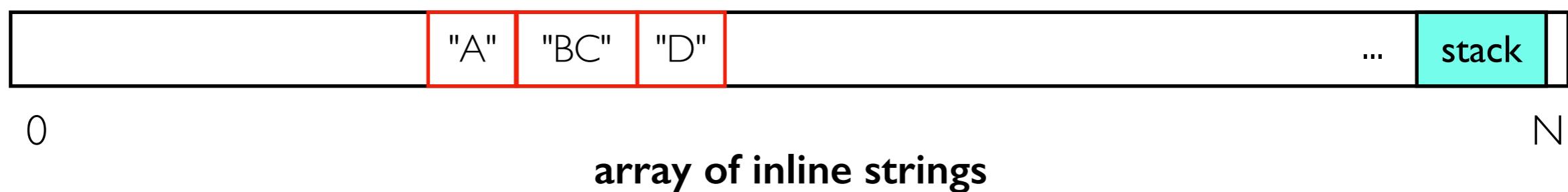
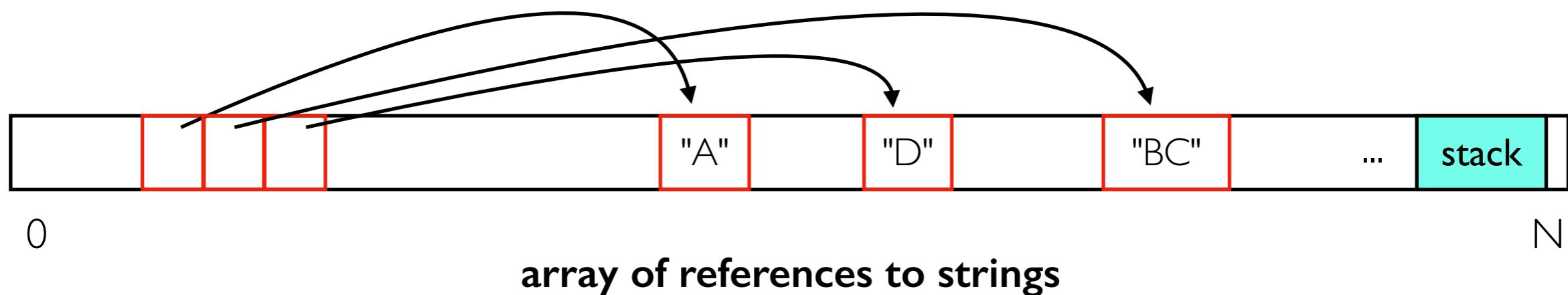
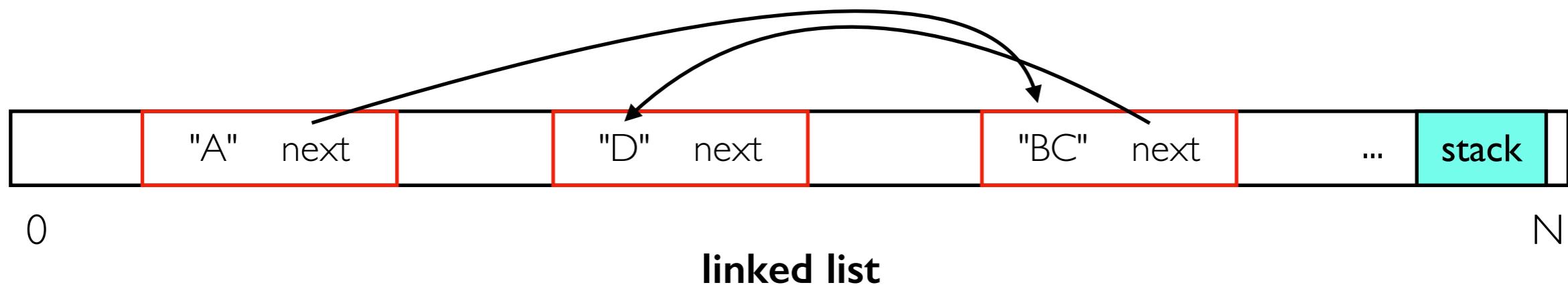
```
np.array( [ [1, 3, 5],  
           [2, 4, 6] ] ) .T
```



any calculations on the two tensors will produce the same results, but they'll each be faster for different access patterns!

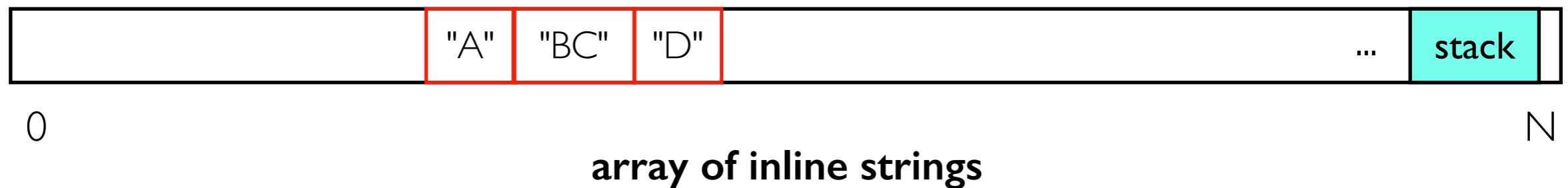
# Example 3: Ordered Collections of Strings

which layout is most cache friendly?

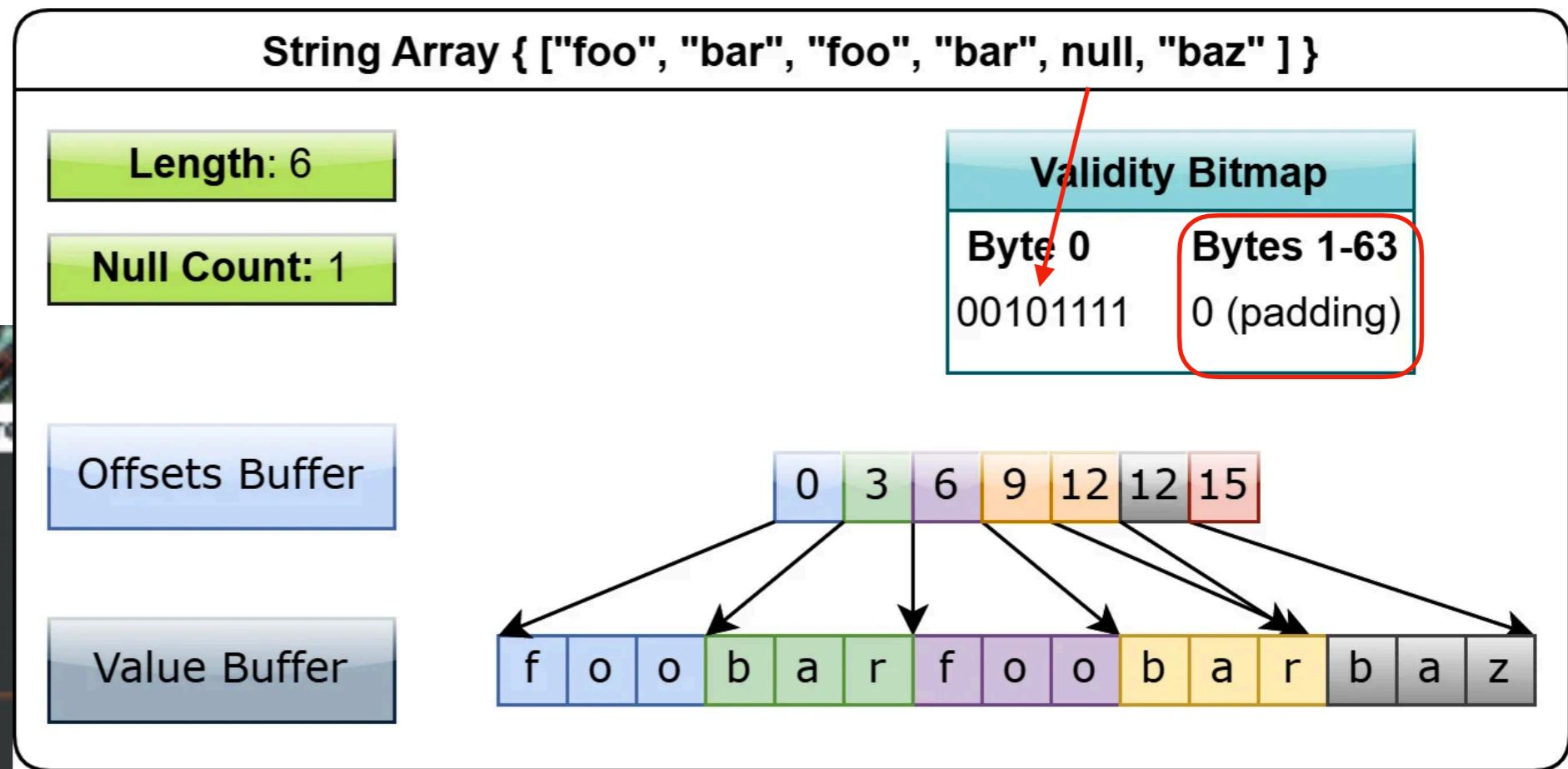


# Example 3: Ordered Collections of Strings

how to tell the end of one string from the start of the next?  
how to jump immediately to string at index i?  
how support null/None?



# PyArrow String Array Data Structure



## In-Memory Analytics with Apache Arrow

Perform fast and efficient data analytics on both flat and hierarchical structured data

- collection of named arrays is a Table
- arrays for different types, each cache friendly
- null support for types like int (not forced into floats)

# Outline

CPU: L1-L3

Demos: Numpy+PyArrow...

Background: Virtual Address Spaces

OS (Operating System): Page Cache

Demos: PyArrow+Docker

# Outline

CPU: L1-L3

Demos: Numpy+PyArrow...

Background: Virtual Address Spaces

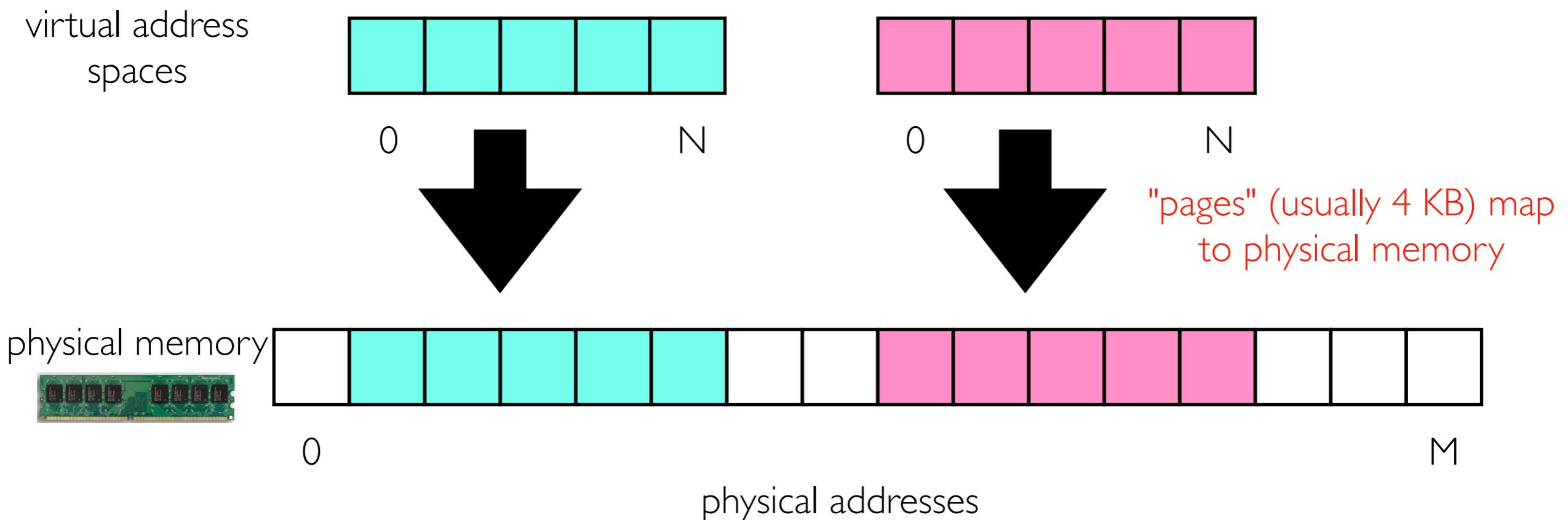
OS (Operating System): Page Cache

Demos: PyArrow+Docker

# Processes and Address Spaces

## Address spaces

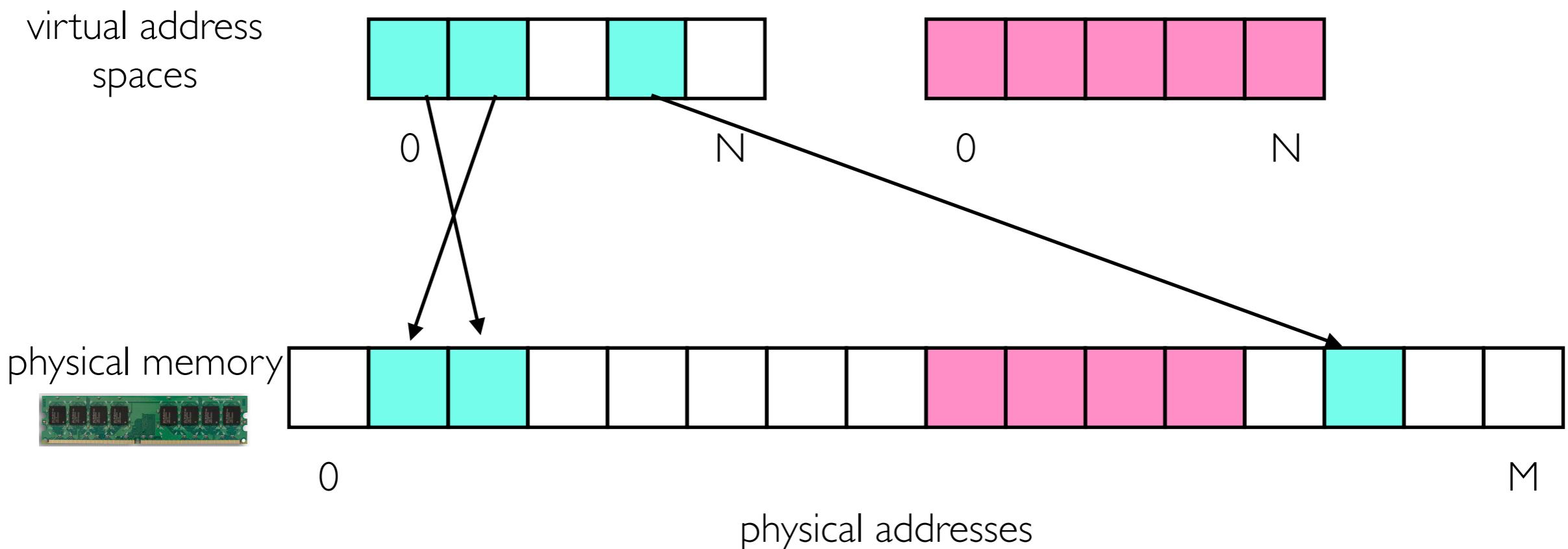
- A **process** is a running **program**
- Each process has it's own **virtual address space**
- The same virtual address generally refers to different memory in different processes
- Regular processes cannot directly access **physical memory** or other addr spaces



# Processes and Address Spaces

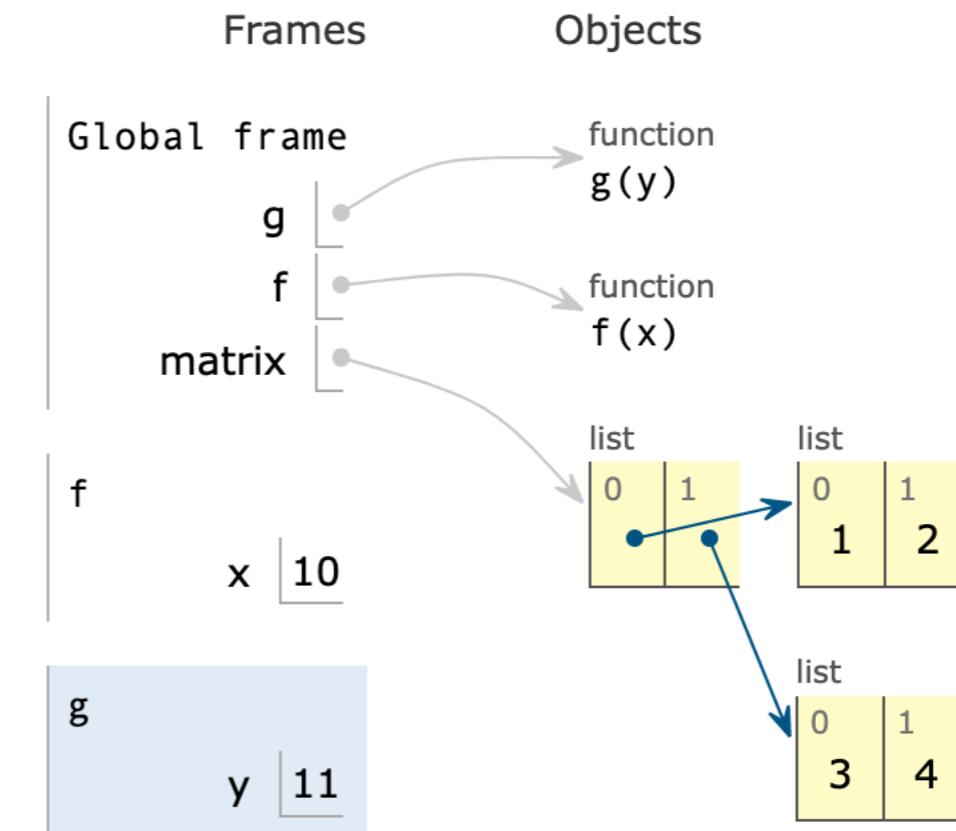
## Address spaces

- A **process** is a running **program**
- Each process has it's own **virtual address space**
- The same virtual address generally refers to different memory in different processes
- Regular processes cannot directly access **physical memory** or other addr spaces
- Address spaces can have holes ( $N$  is usually MUCH bigger than  $M$ )
- Physical memory for a process need not be contiguous



# What goes in an address space?

```
1 def g(y):
2     return y * 2
3
4 def f(x):
5     return g(x+1)
6
7 matrix = [[1,2], [3,4]]
8 f(10)
```



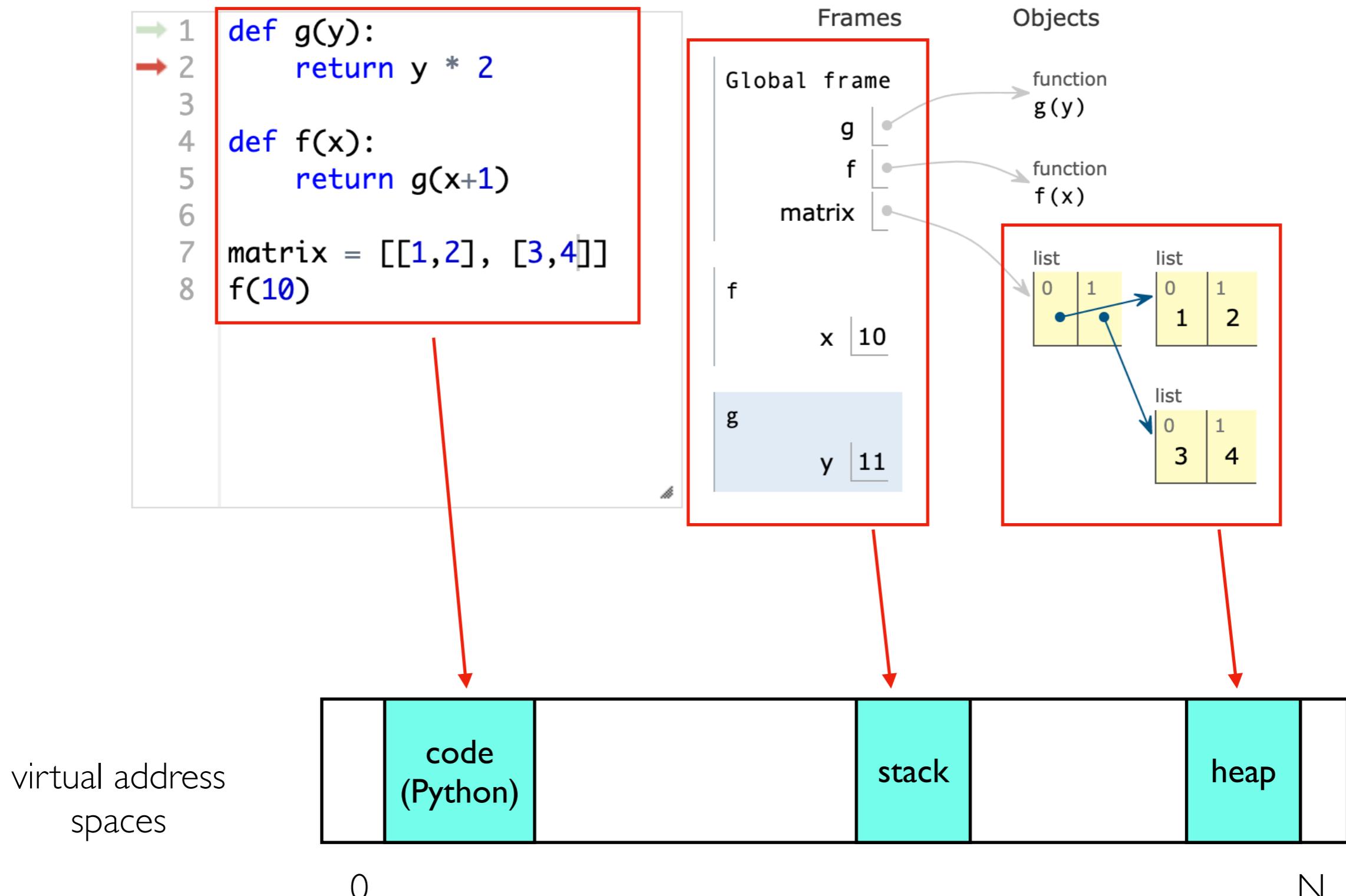
<https://pythontutor.com/>

# virtual address spaces

# what goes here?

0

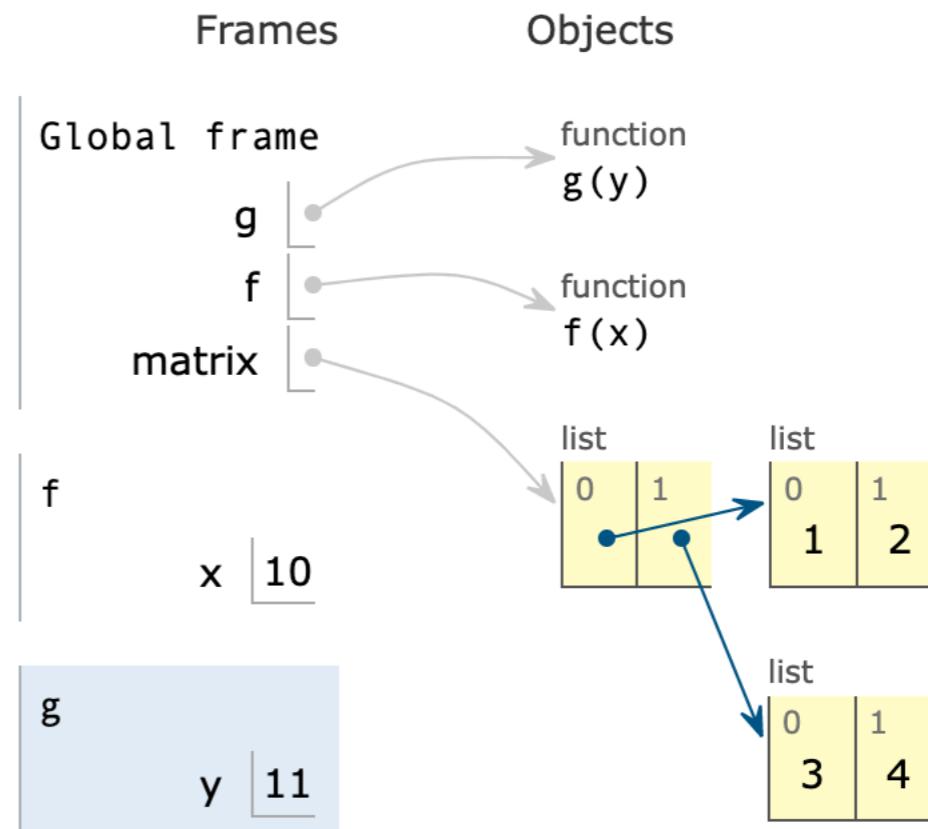
# What goes in an address space?



Note: code and heap generally not contiguous

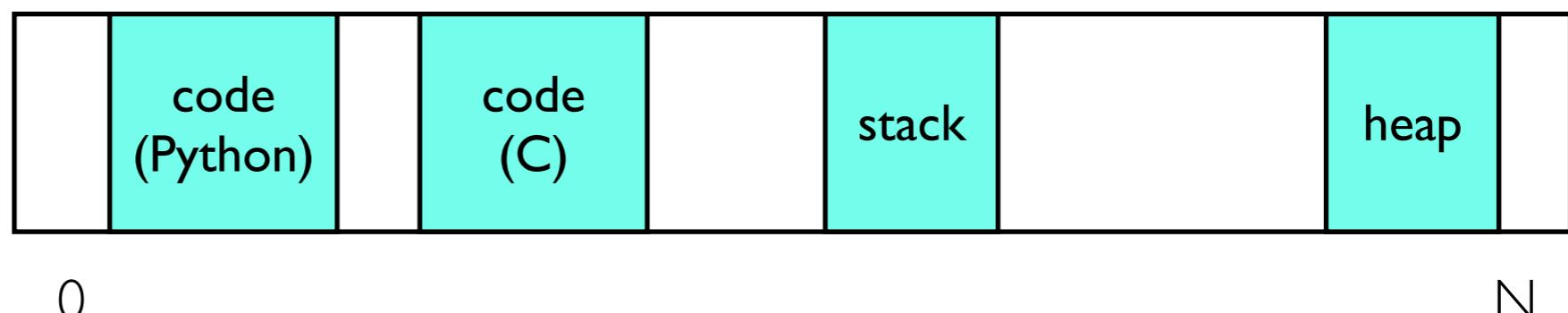
# What goes in an address space?

```
1 def g(y):
2     return y * 2
3
4 def f(x):
5     return g(x+1)
6
7 matrix = [[1,2], [3,4]]
8 f(10)
```



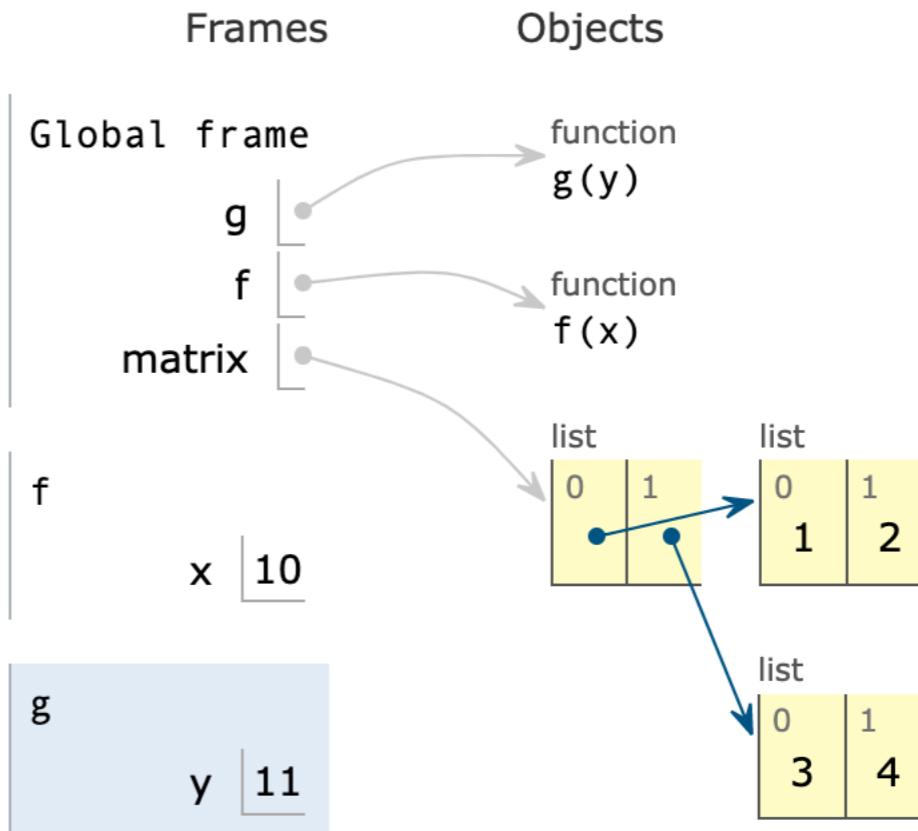
# virtual address spaces

some packages  
(like numpy)

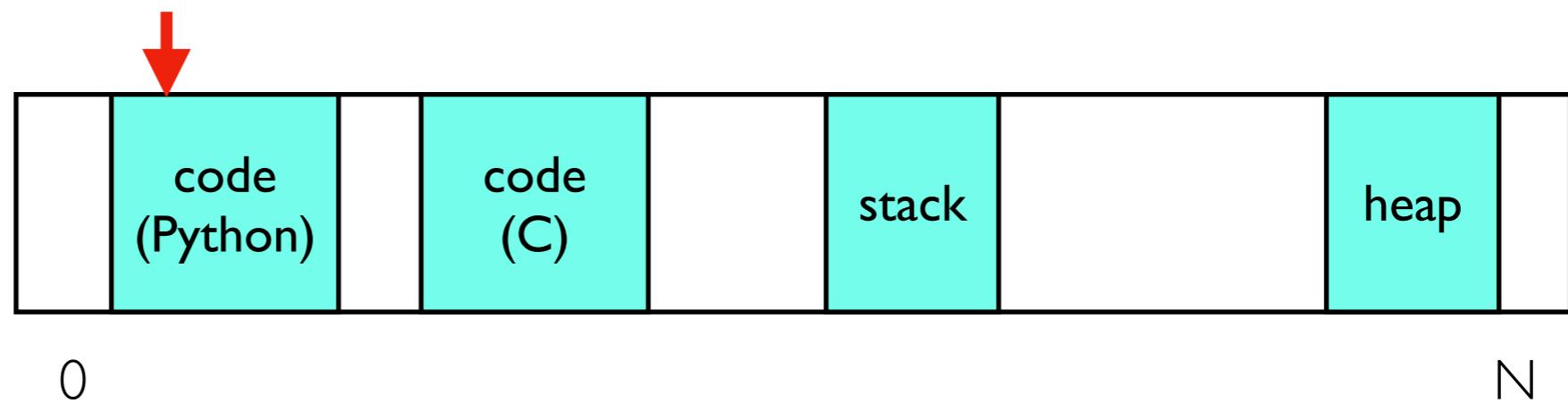


# How does code execute?

```
1 def g(y):  
2     return y * 2  
3  
4 def f(x):  
5     return g(x+1)  
6  
7 matrix = [[1,2], [3,4]]  
8 f(10)
```



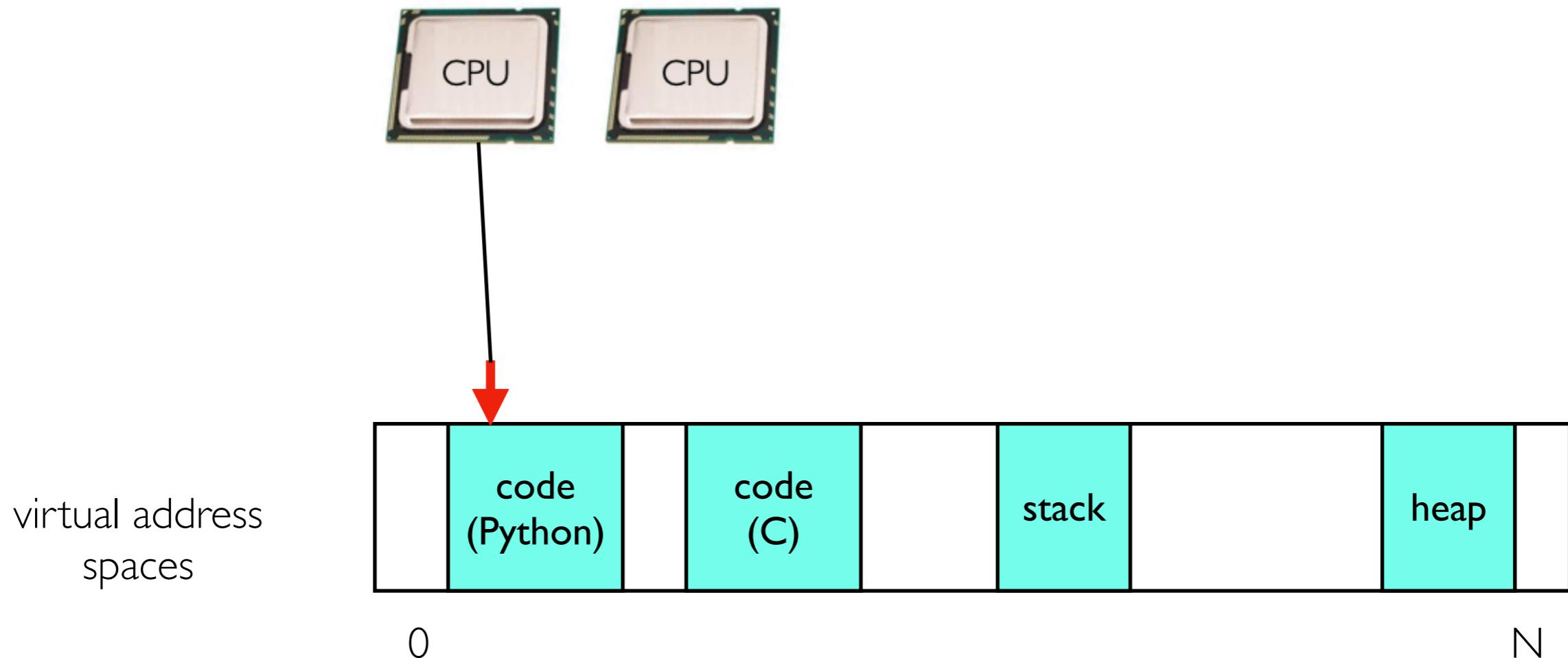
virtual address  
spaces



# How does code execute?

CPUs

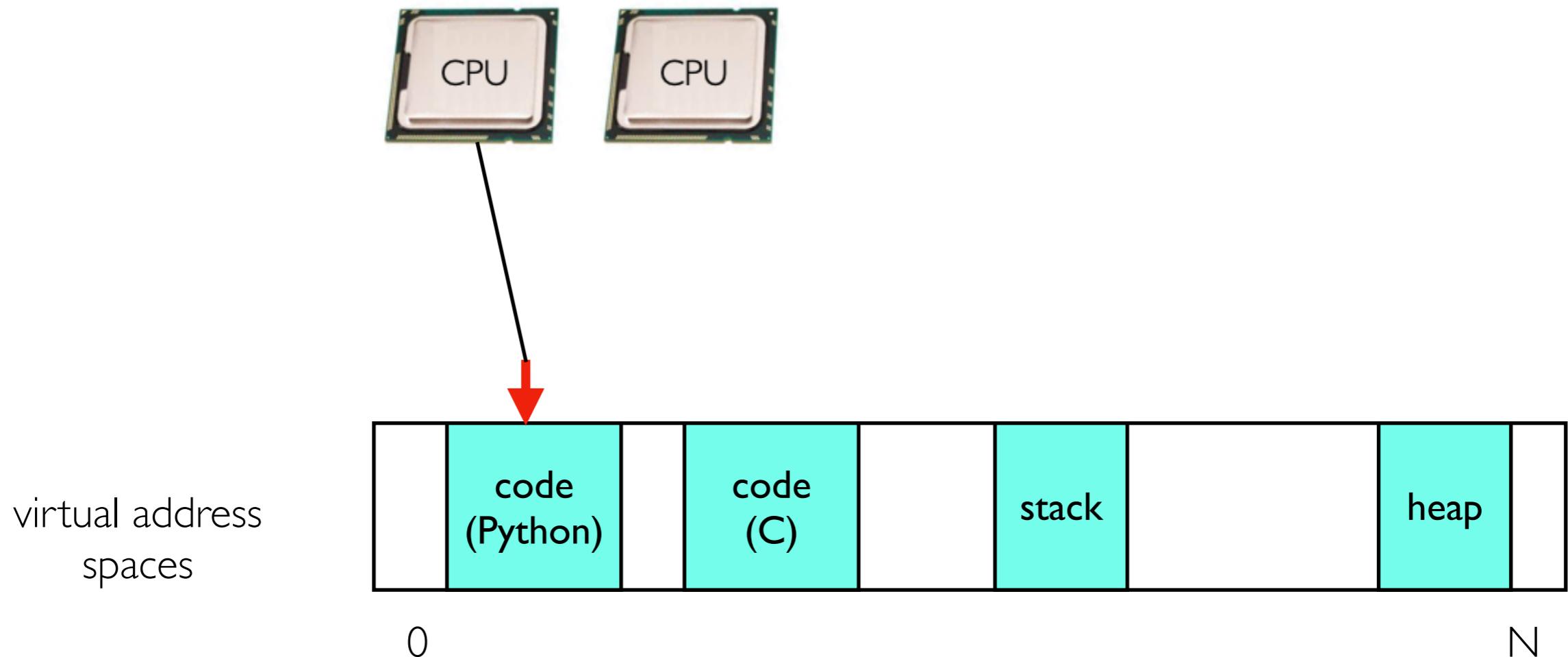
- CPUs are attached to at most one **instruction pointer** at any given time
- they run code by executing instructions and advancing the instruction pointer
- **Note:** interpreter left out for simplicity (CPU points to interpreter code, which points to Python bytecode)



# How does code execute?

CPUs

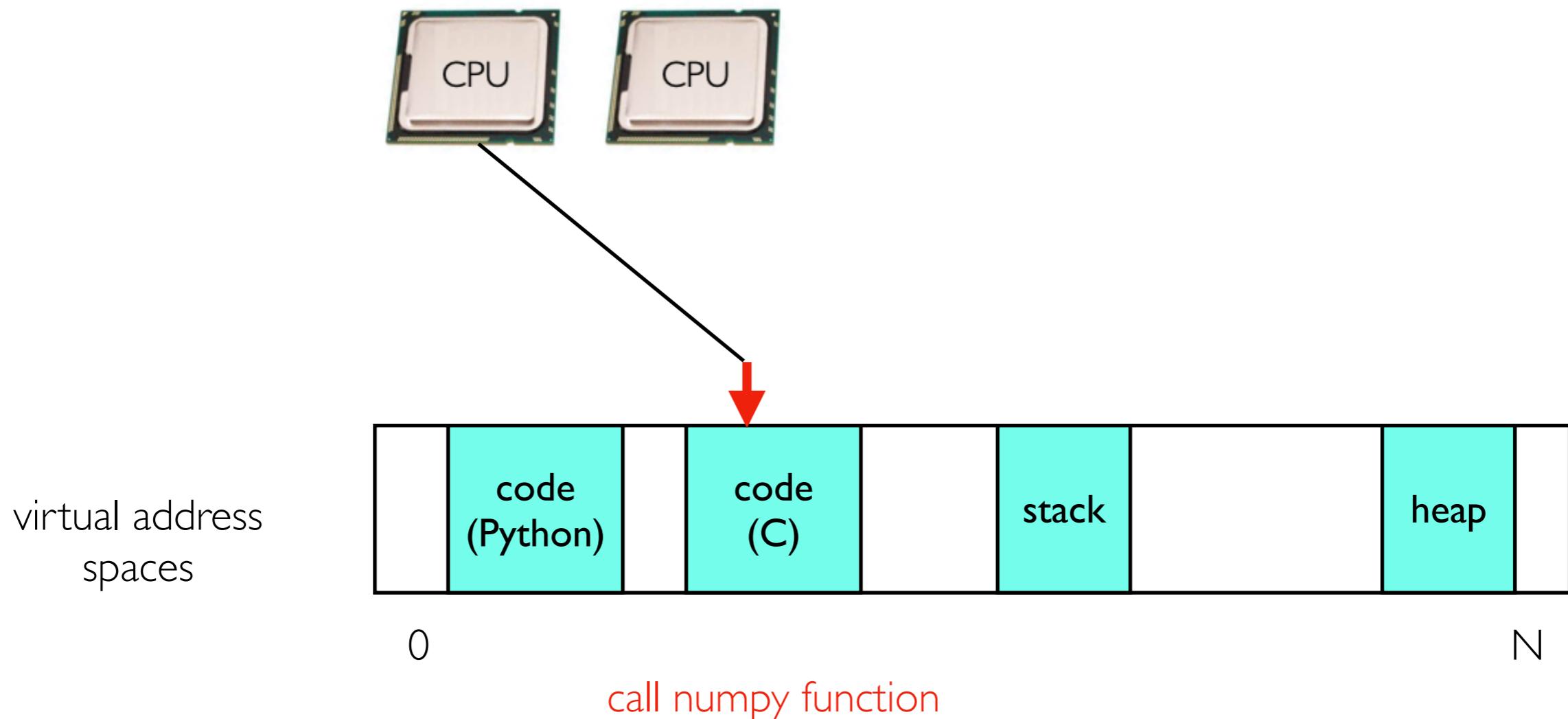
- CPUs are attached to at most one **instruction pointer** at any given time
- they run code by executing instructions and advancing the instruction pointer
- **Note:** interpreter left out for simplicity (CPU points to interpreter code, which points to Python bytecode)



# How does code execute?

CPUs

- CPUs are attached to at most one **instruction pointer** at any given time
- they run code by executing instructions and advancing the instruction pointer
- **Note:** interpreter left out for simplicity (CPU points to interpreter code, which points to Python bytecode)



# Outline

CPU: L1-L3

Demos: Numpy+PyArrow...

Background: Virtual Address Spaces

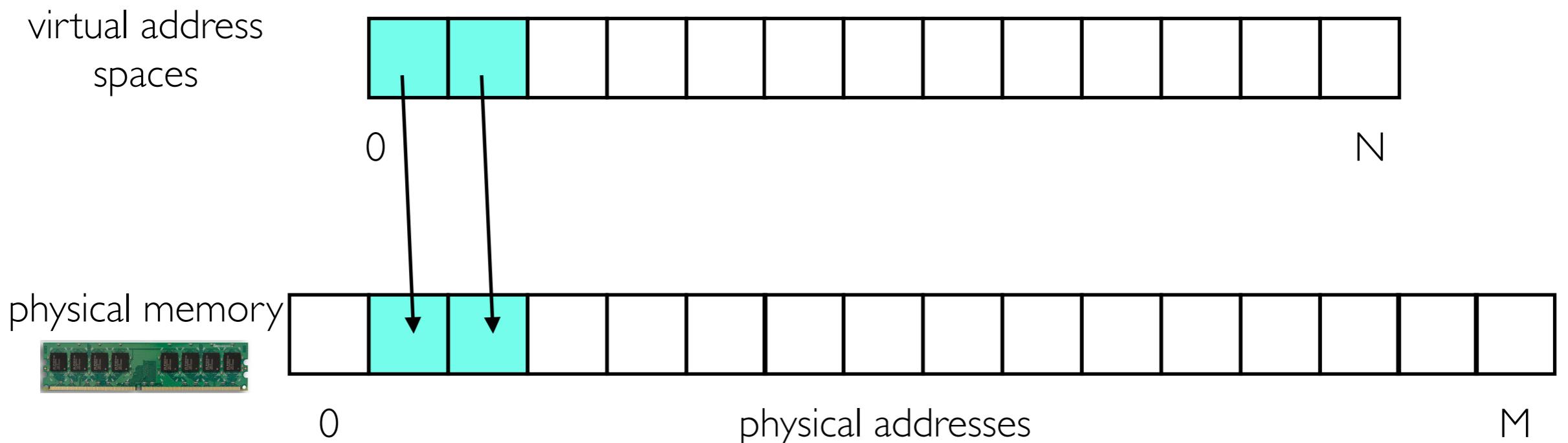
OS (Operating System): Page Cache

Demos: PyArrow+Docker

# mmap (Memory Map)

An mmap call can add new regions to a virtual address space. Two varieties:

- anonymous
- backed by a file



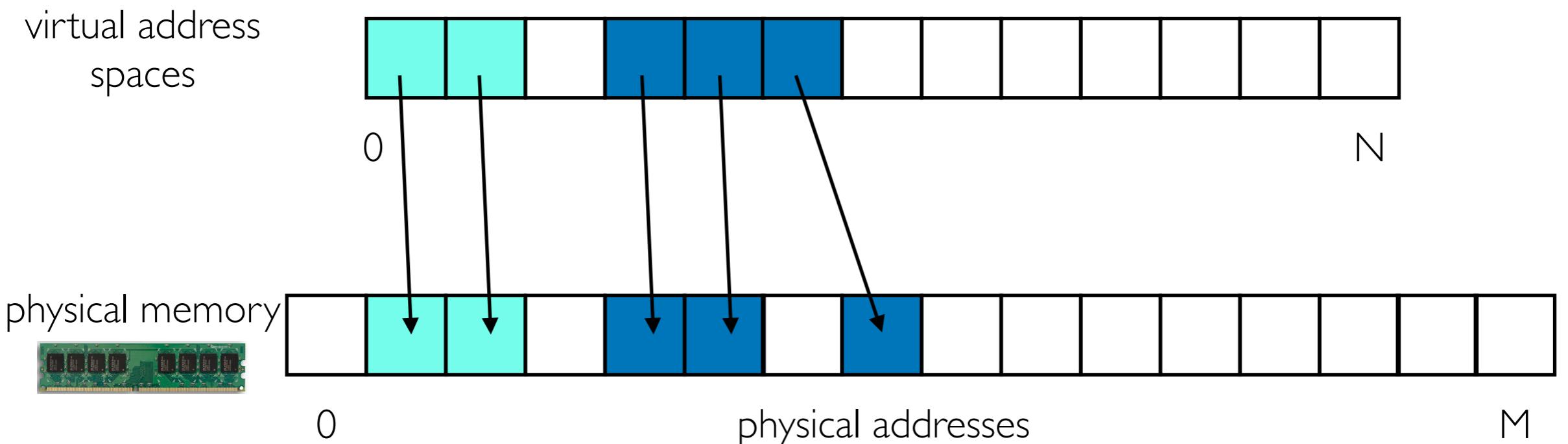
# Anonymous mmap

An mmap call can add new regions to a virtual address space. Two varieties:

- anonymous
- backed by a file

```
import mmap  
mm = mmap.mmap(-1, 4096*3)
```

anonymous 3 pages



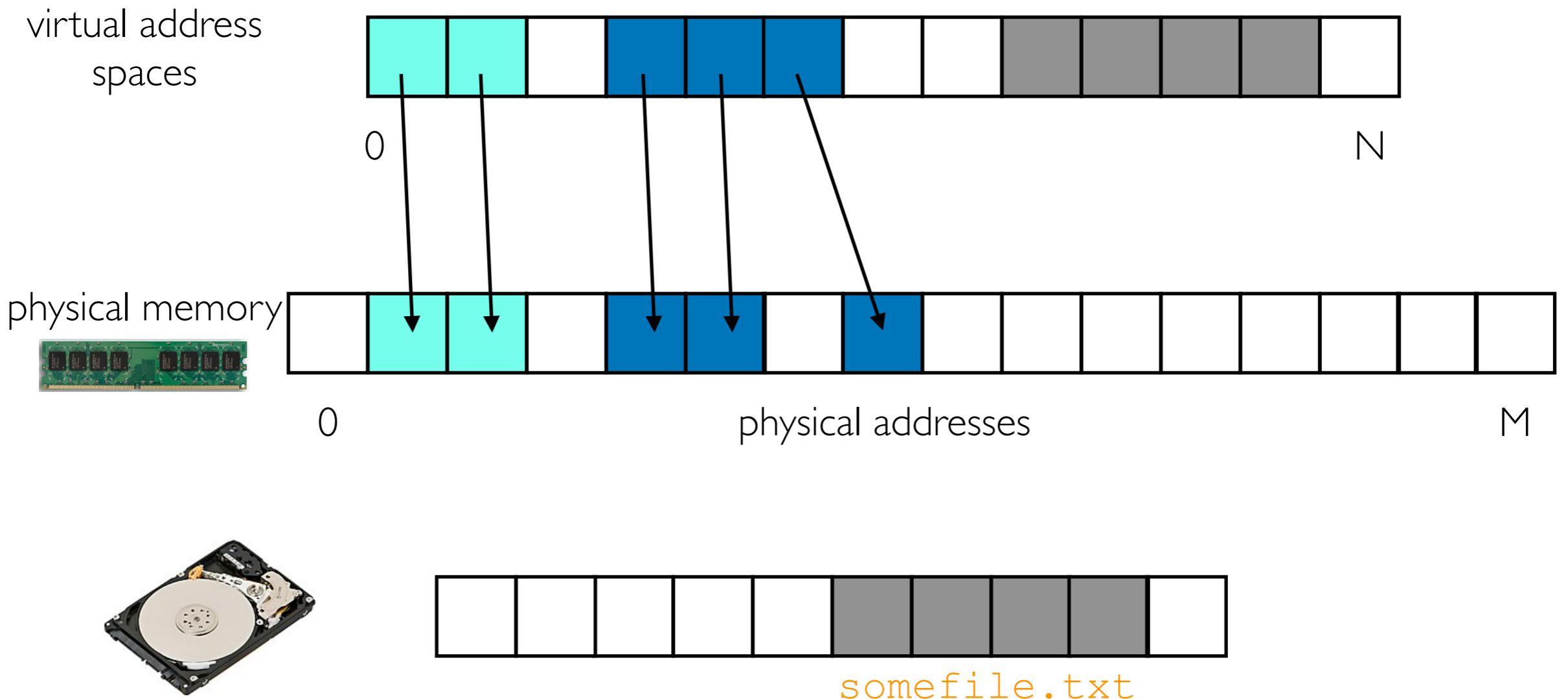
- Python (and other language runtimes) will mmap some anonymous memory when they need more heap space
- this will be used for Python objects (ints, lists, dicts, DataFrames, etc.)

# File-Backed mmap

An mmap call can add new regions to a virtual address space. Two varieties:

- anonymous
- backed by a file

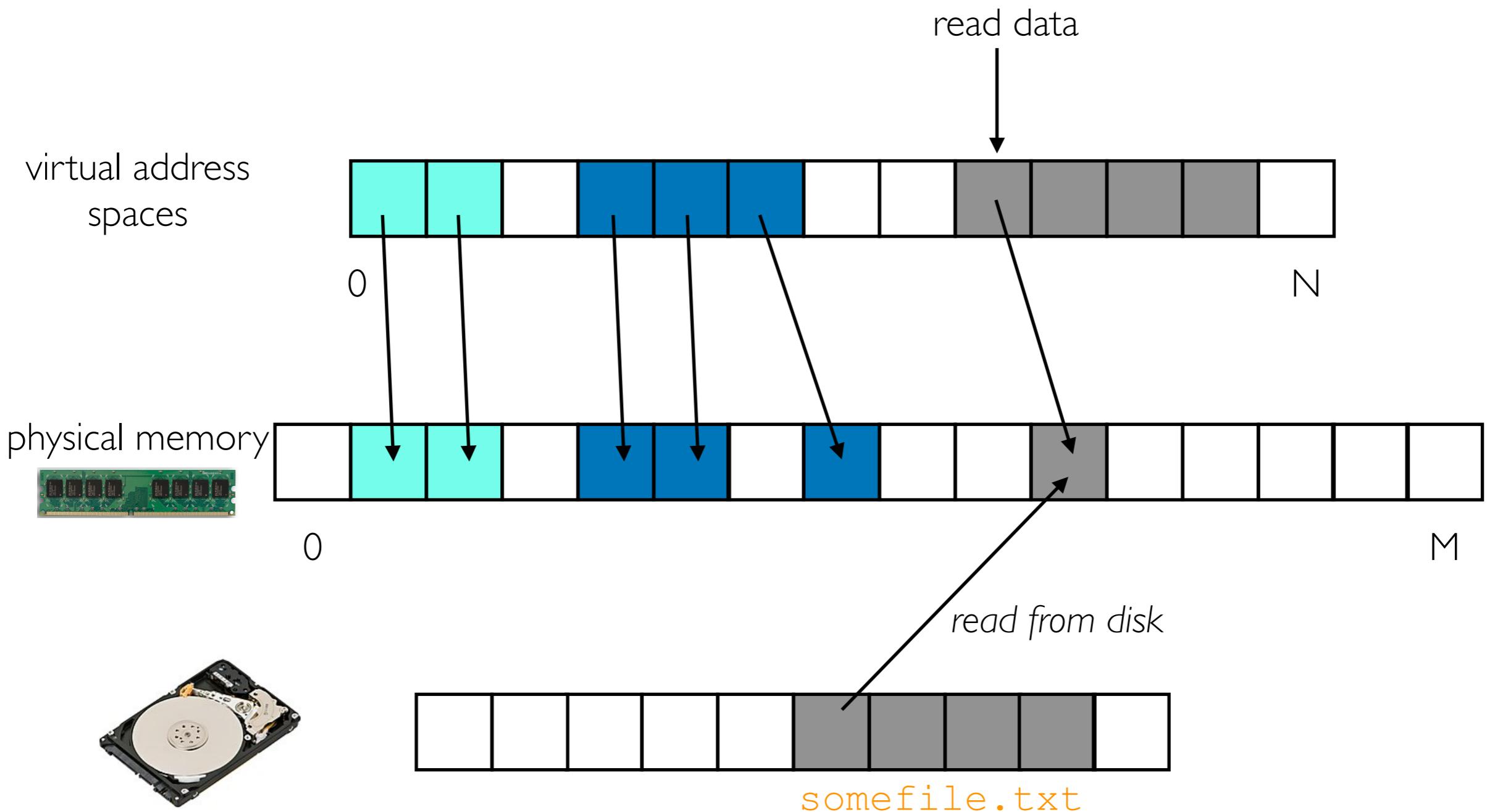
```
import mmap
f = open("somefile.txt", mode="rb")
mm = mmap.mmap(f.fileno(), 0, # 0 means all
                access=mmap.ACCESS_READ)
```



# File-Backed mmap

An mmap call can add new regions to a virtual address space. Two varieties:

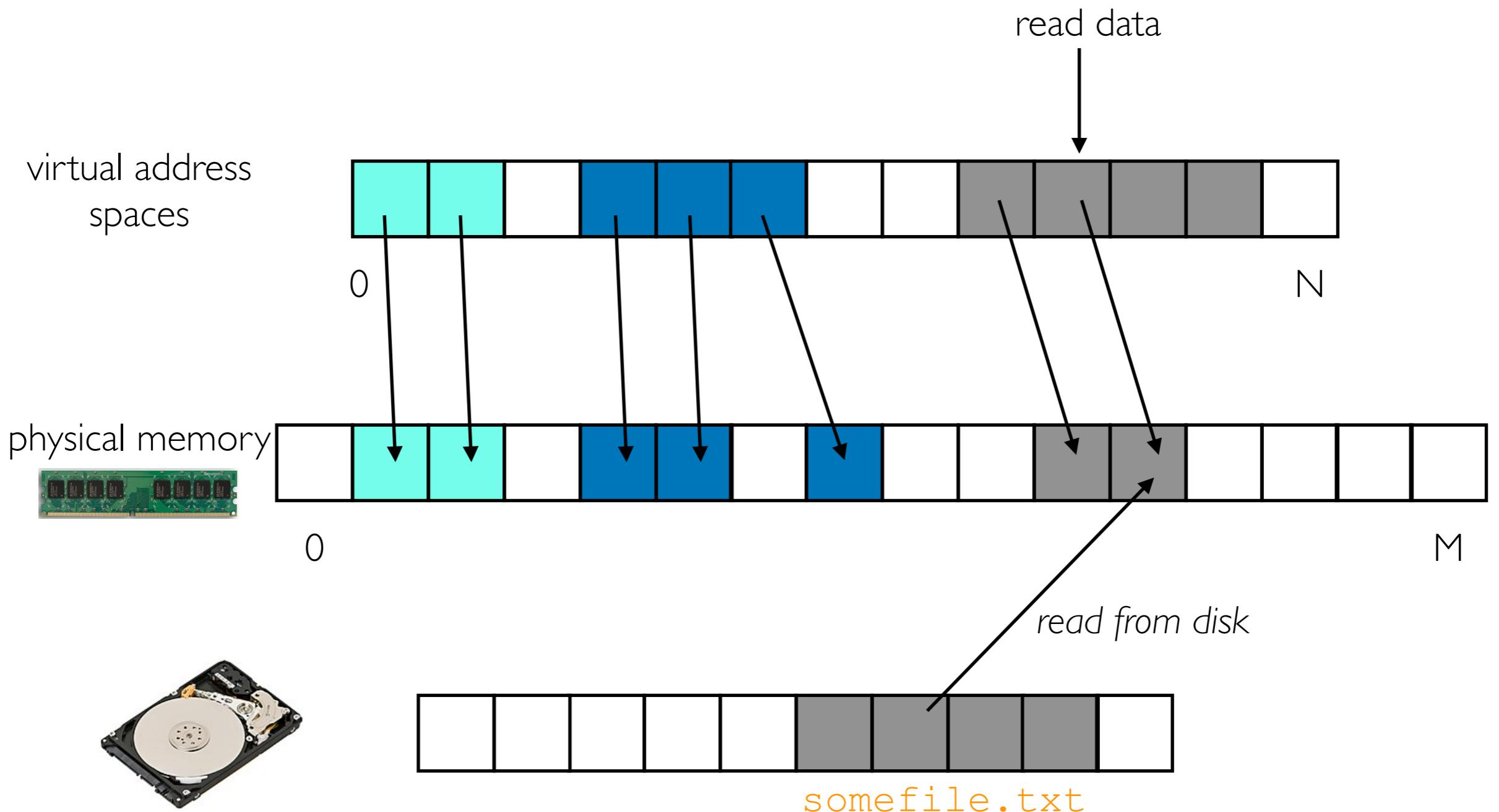
- anonymous
  - backed by a file



# File-Backed mmap

An mmap call can add new regions to a virtual address space. Two varieties:

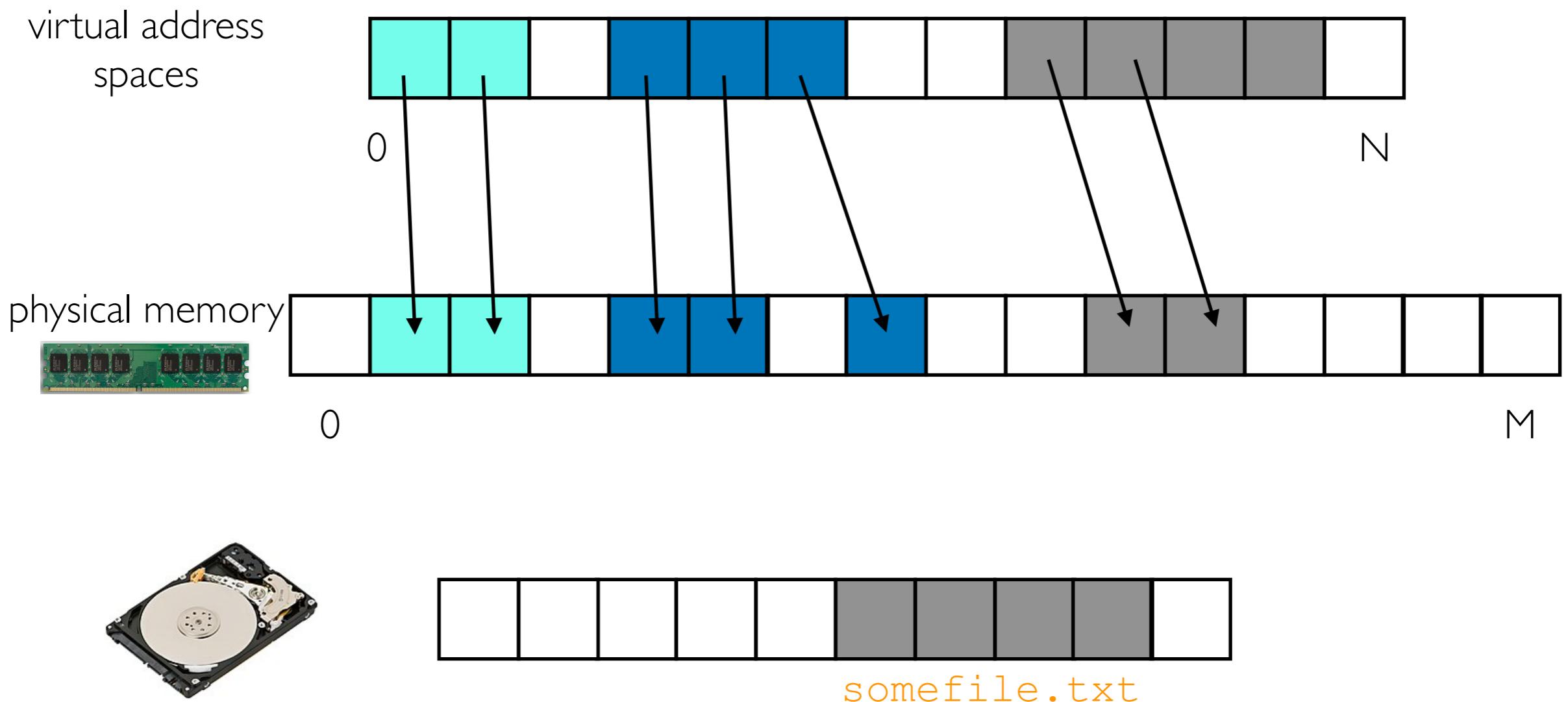
- anonymous
- backed by a file



# File-Backed mmap

An mmap call can add new regions to a virtual address space. Two varieties:

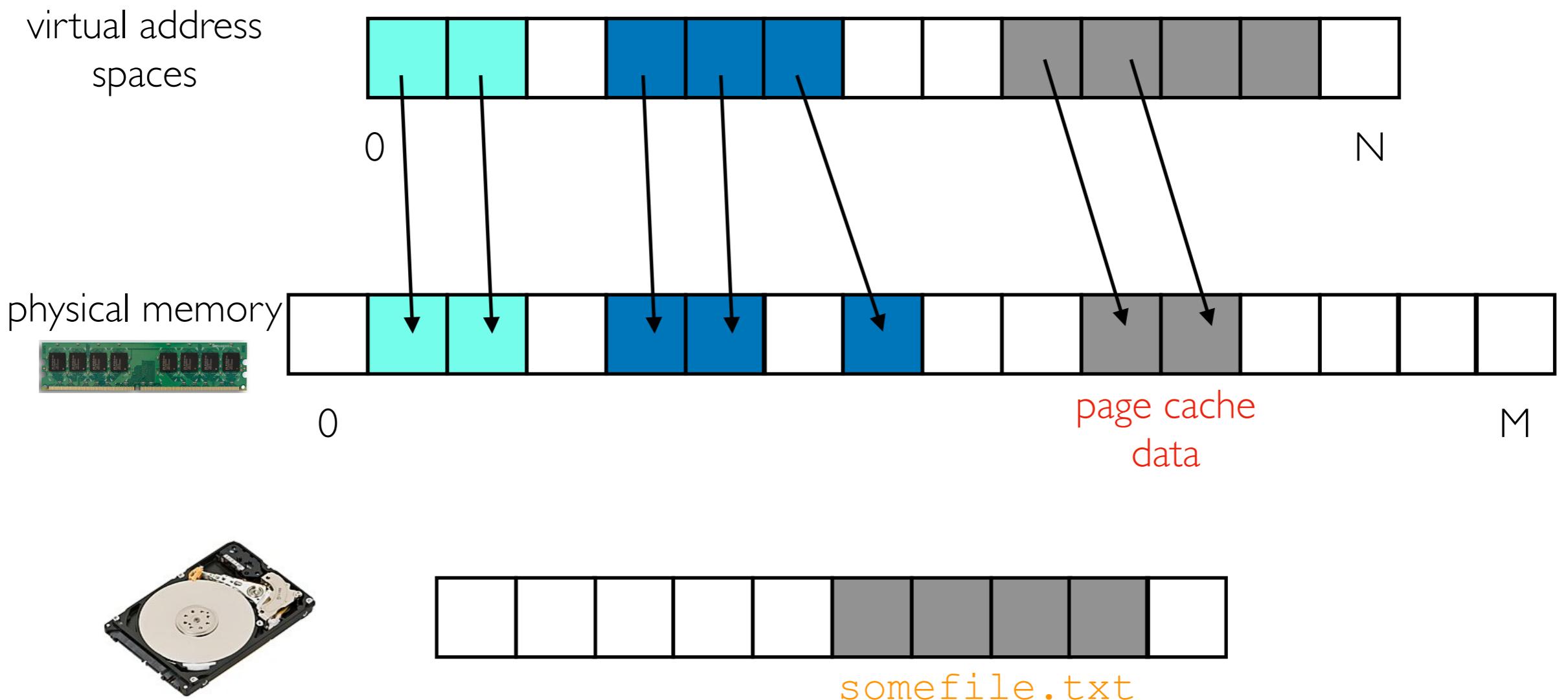
- anonymous
  - backed by a file
- **virtual** memory used:  $9 \times \text{pagesize} = 36 \text{ KB}$
  - **physical** memory used:  $7 \times \text{pagesize} = 28 \text{ KB}$



# File-Backed mmap

An mmap call can add new regions to a virtual address space. Two varieties:

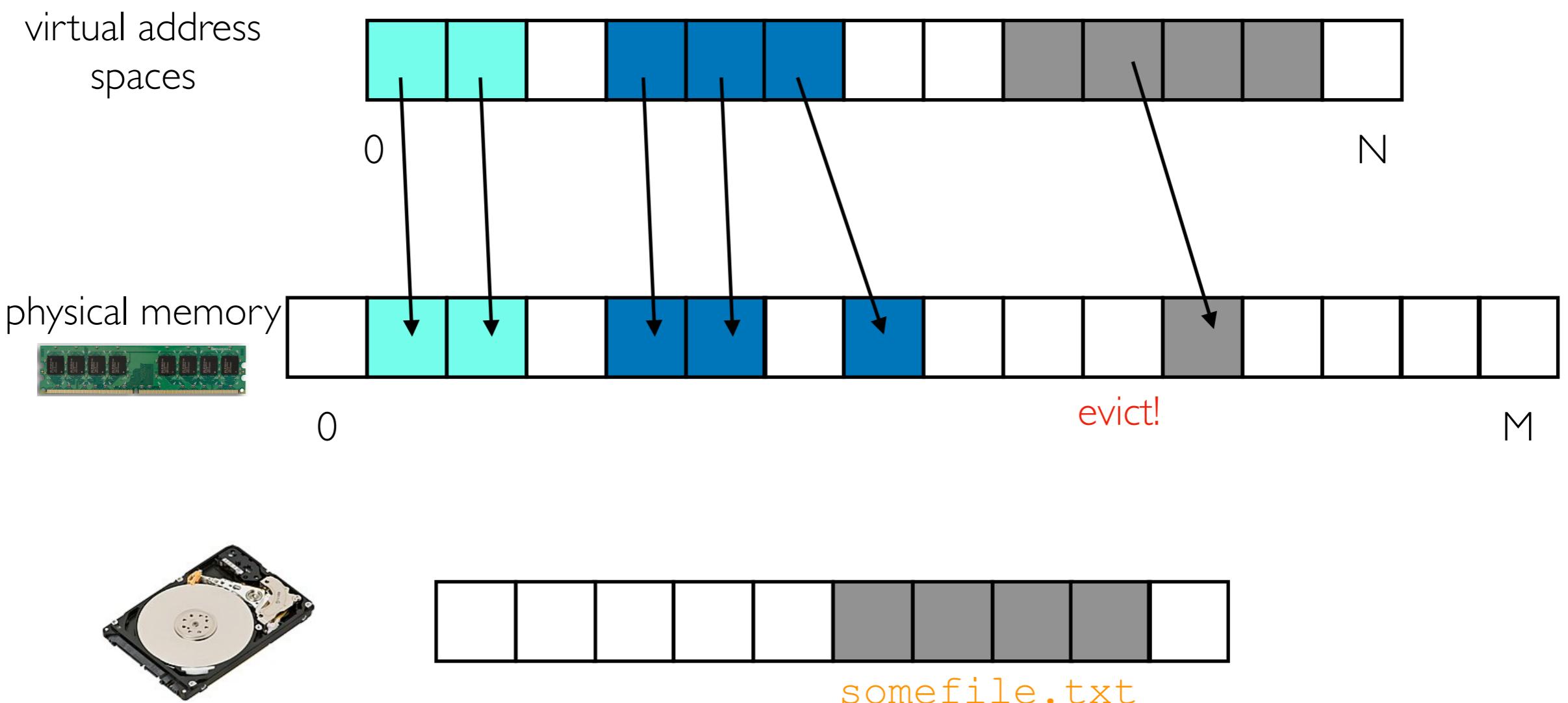
- anonymous
  - backed by a file
- data loaded for accesses to file-backed mmap regions are part of the "page cache"



# File-Backed mmap

An mmap call can add new regions to a virtual address space. Two varieties:

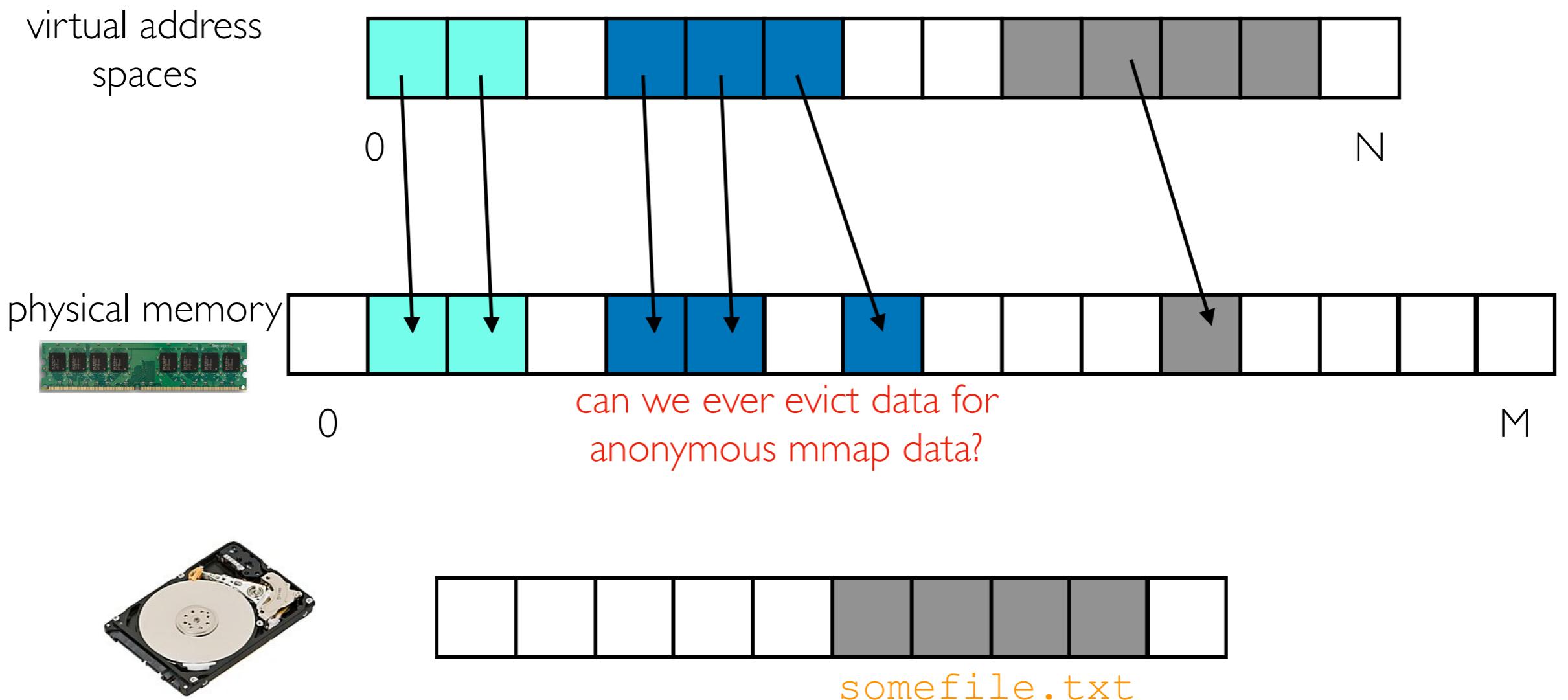
- anonymous
  - backed by a file
- data loaded for accesses to file-backed mmap regions are part of the "page cache"
  - it works like a cache because there is another copy on disk, so we can evict under memory pressure



# Swap Space

An mmap call can add new regions to a virtual address space. Two varieties:

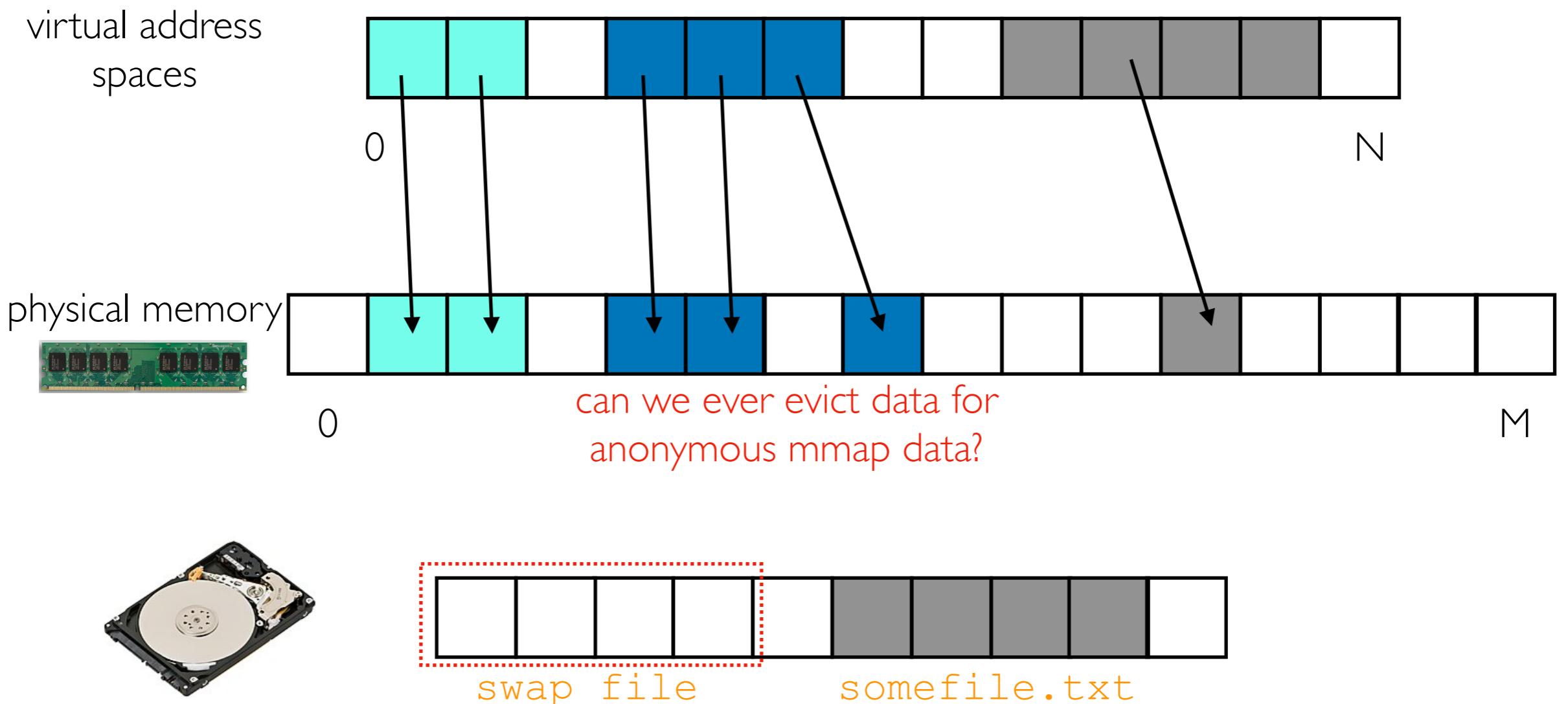
- anonymous
- backed by a file



# Swap Space

An mmap call can add new regions to a virtual address space. Two varieties:

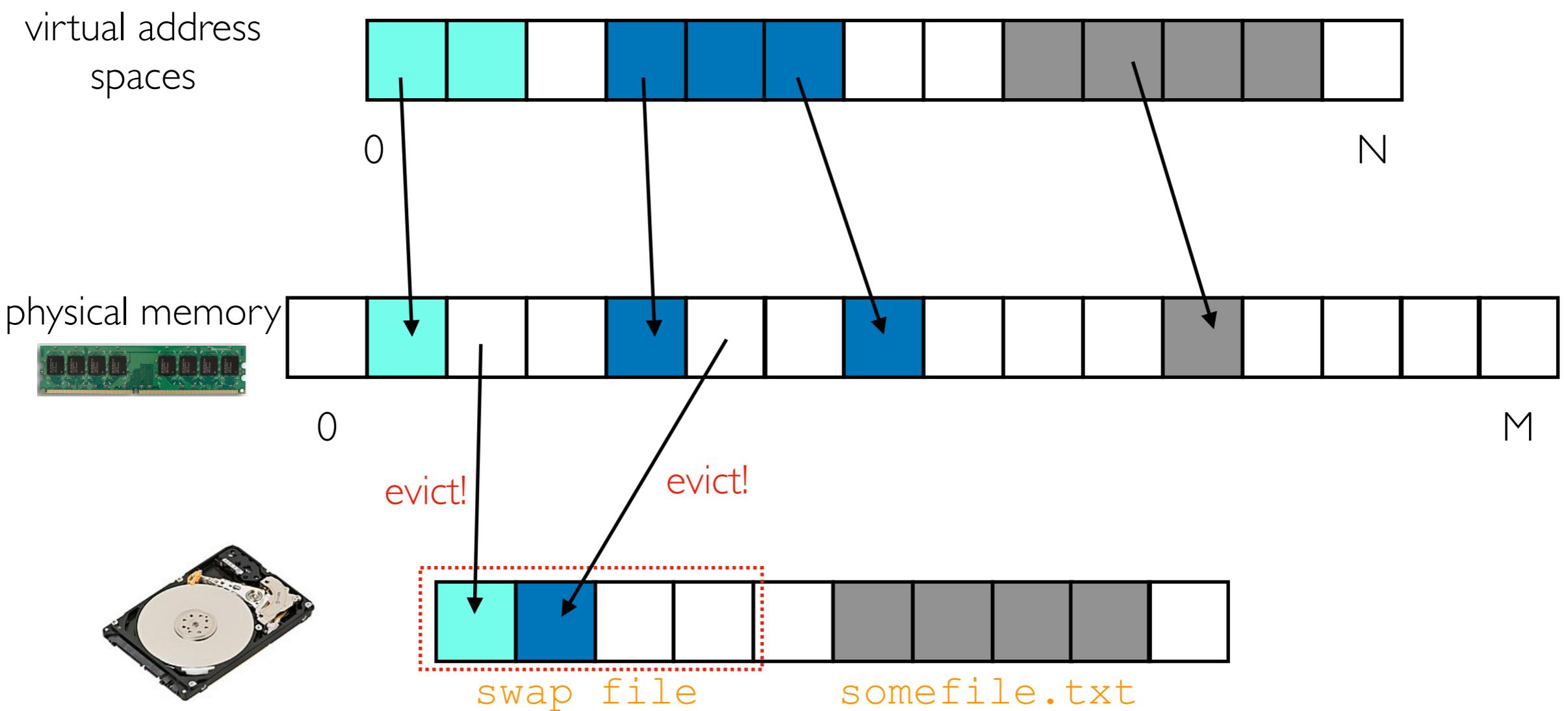
- anonymous
  - backed by a file
- we can create same space (a swap file) to which the OS can evict data from anonymous mappings



# Swap Space

An mmap call can add new regions to a virtual address space. Two varieties:

- anonymous
  - backed by a file
- we can create same space (a swap file) to which the OS can evict data from anonymous mappings
  - of course, if we access these virtual addresses again, it will be slow to bring the data back



# Outline

CPU: L1-L3

Demos: Numpy+PyArrow...

Background: Virtual Address Spaces

OS (Operating System): Page Cache

Demos: PyArrow+Docker