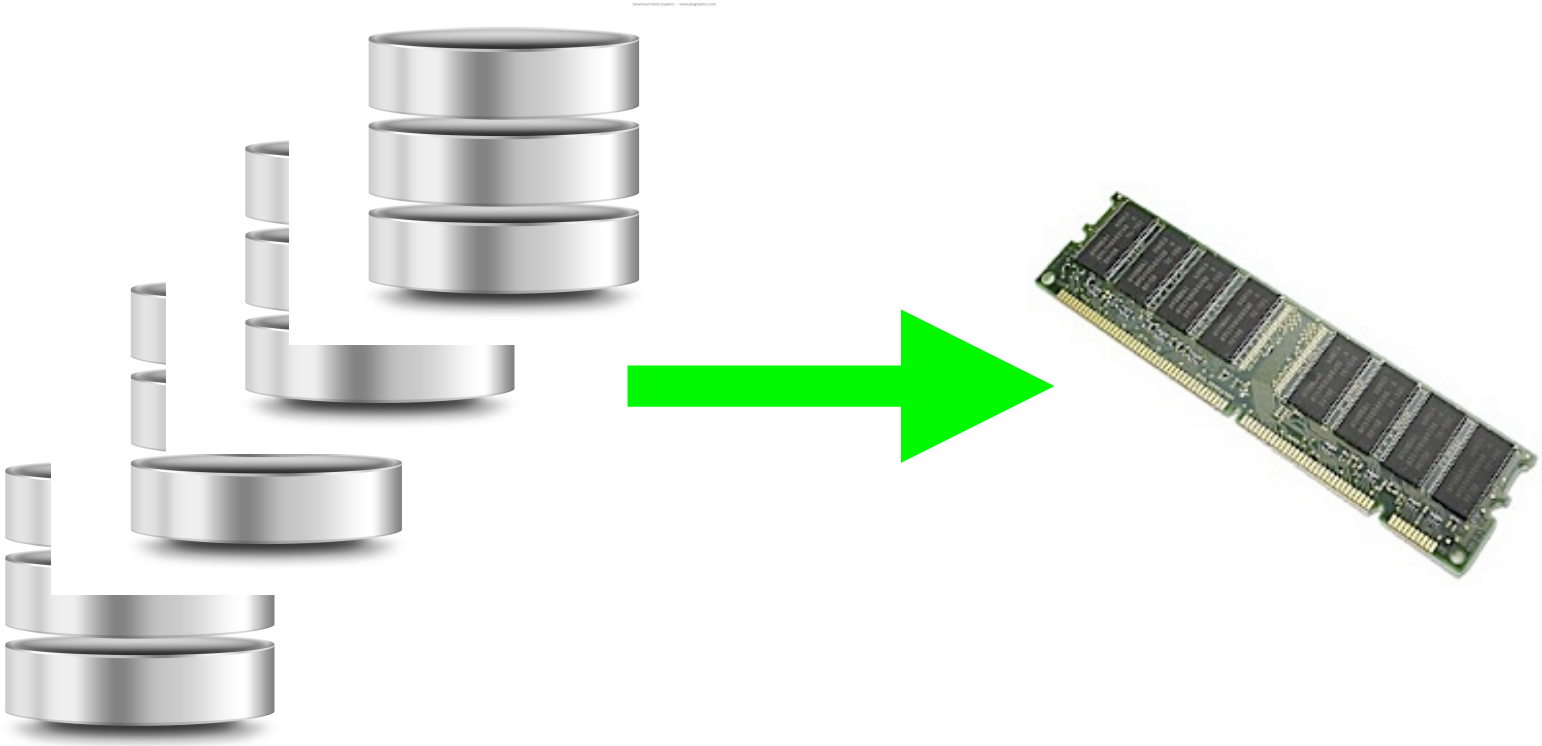
# Discussion 3

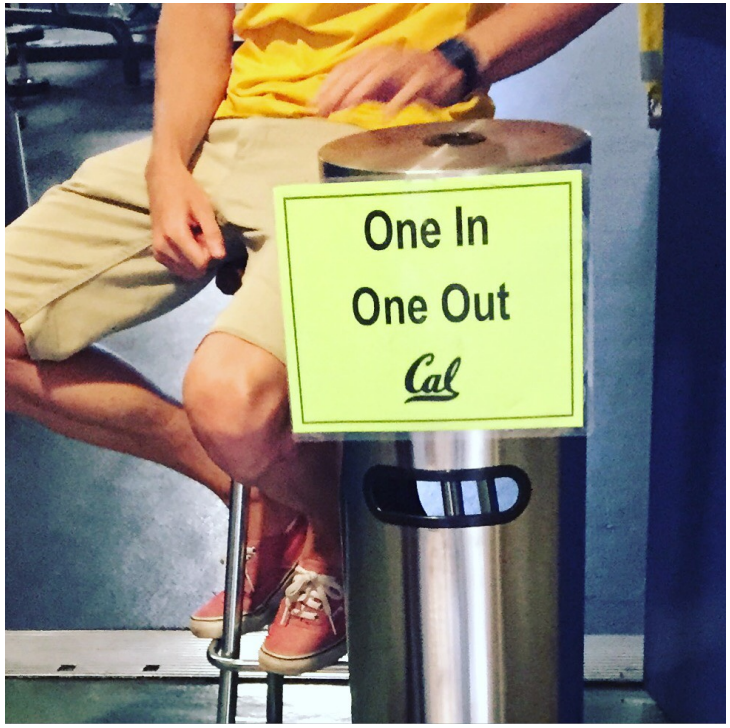
Buffer Management and PySpark Introduction

# Buffers



http://www.psdgraphics.com/psd-icons/silver-computer-database-icon-psd/
http://www.igcseict.info/theory/1/comp/

# Cannot Fit All



# Terminology

- Buffering
- Buffer pool
- Buffer Replacement Policy
- Caching
- Pages
- Frames

# Worksheet

# Picking a Page to Remove

- LRU
  - Clock (why?)
- MRU

# Least/Most Recently Used

- Policy?
- Rational?
- Failure cases?

"*When a file is being repeatedly scanned in a [Looping Sequential] reference pattern, MRU is the best replacement algorithm.*". Chou and DeWitt

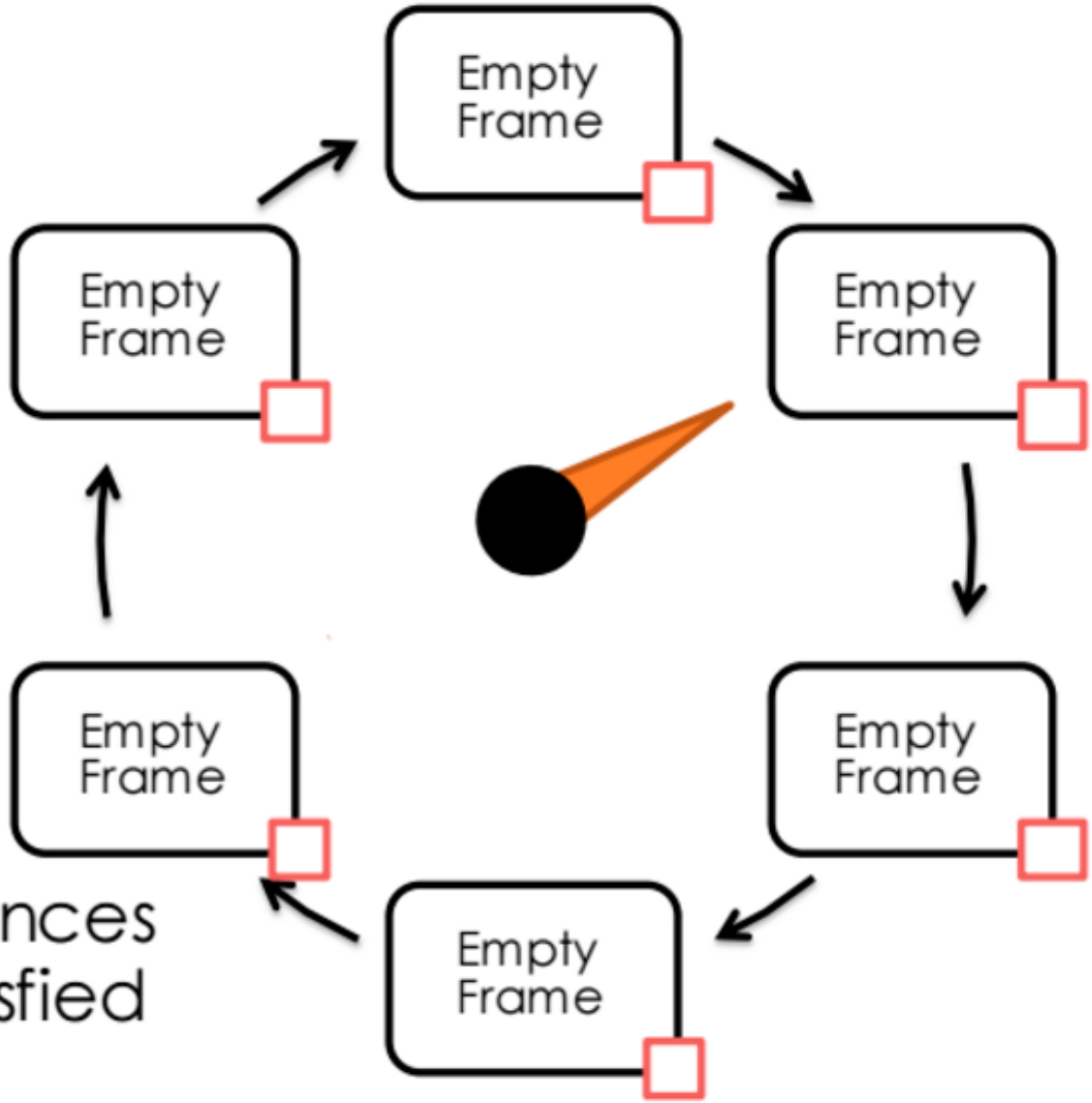

# Clock Replacement Policy

Approx. to LRU

Arrange frames in logical cycle

Introduce "Ref. Bit"

Clock arm advances until request satisfied




amplab


PySpark

# Concepts

- iterators
- generators
- lambdas
- Resilient Distributed Dataset

# Iterators

```
>>> x = iter([1, 2, 3])
>>> x
<listiterator object at 0x1004ca850>
>>> x.next()
1
>>> x.next()
2
>>> x.next()
3
>>> x.next()
```

```
>>> for line in open("a.txt"):
...     print line,
...
first line
second line
```

# Generators

```
1 # Build and return a list
2 def firstn(n):
3     num, nums = 0, []
4     while num < n:
5         nums.append(num)
6         num += 1
7     return nums
8
9 sum_of_first_n = sum(firstn(1000000))
```

```
1 # a generator that yields items instead of returning a list
2 def firstn(n):
3     num = 0
4     while num < n:
5         yield num
6         num += 1
7
8 sum_of_first_n = sum(firstn(1000000))
```

# List Comprehension

```
# List comprehensions:
x = range(100)

y = [n**2 for n in x if n < 5]

print y

y2 = [n**2 if n % 2 else 0 for n in y]
print y2

print [a * b for a in y for b in y2]
```

```
[0, 1, 4, 9, 16]
[0, 1, 0, 81, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 81, 0, 0, 4, 0, 324, 0, 0, 9, 0, 729, 0, 0, 16, 0, 1296, 0]
```

# Lambda

```
class MyClass(object):
    def func(self, s):
        return s
    def doStuff(self, rdd):
        return rdd.map(self.func)
```

```
class MyClass(object):
    def __init__(self):
        self.field = "Hello"
    def doStuff(self, rdd):
        return rdd.map(lambda s: self.field + s)
```

```
# Lambda Expressions

def convert_me(n):
    return 1./ n ** 2

convert_you = lambda x: 1./x ** 2

convert_me(10) == convert_you(10)
```

```
True
```

# Resilient Distributed Dataset

- Abstraction for immutable, partitioned collection of elements that can be operated on in parallel.
- Creating them:
  - **parallelizing** an existing collection in your driver program
  - referencing a **dataset** in an external storage system, such as a shared filesystem, HDFS, HBase

# Parallelized Collections

Created: calling SparkContext's parallelize method on an existing iterable or collection

```
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)
```

# External Datasets

```
distFile = sc.textFile("data.txt")
```

# Functions

- **`collect()`**
  - bring the RDD to the driver node
- **`first()`**
  - Return the first element of the dataset (similar to take(1)).