MapReduce

DSC 232R

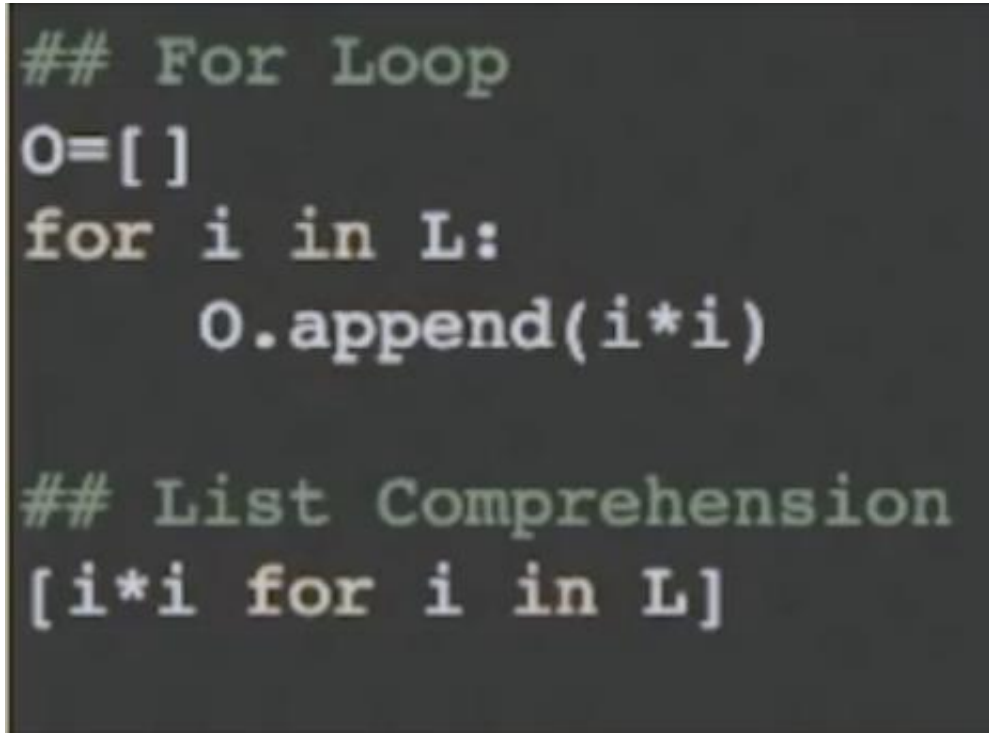
Achieving locality by being oblivious to order

- To minimize cache misses we want to process data sequentially.
- To compute in parallel on several CPUs, we want processing in each CPU to be independent of the others.
- As a programmer, we want to achieve sequentiality and parallelism, **without knowing the details of the hardware**.
- **Approach:** write code that expresses the desired end result, without specifying how to get there.
- **MapReduce:** perform operations on arrays without specifying the order of the computation.

Map: square each item

- list L =[0,1,2,3]
- Compute the square of each item
- output: [0,1,4,9]

Traditional

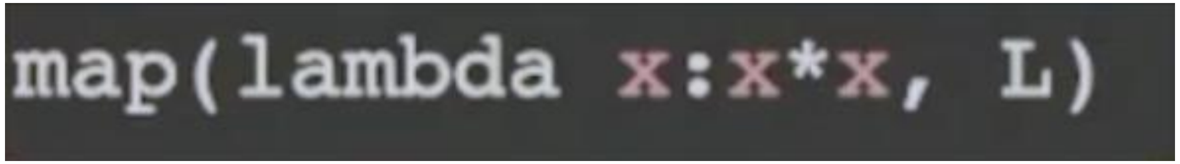


```
## For Loop
O=[]
for i in L:
    O.append(i*i)

## List Comprehension
[i*i for i in L]
```

compute from first to last in order

MapReduce



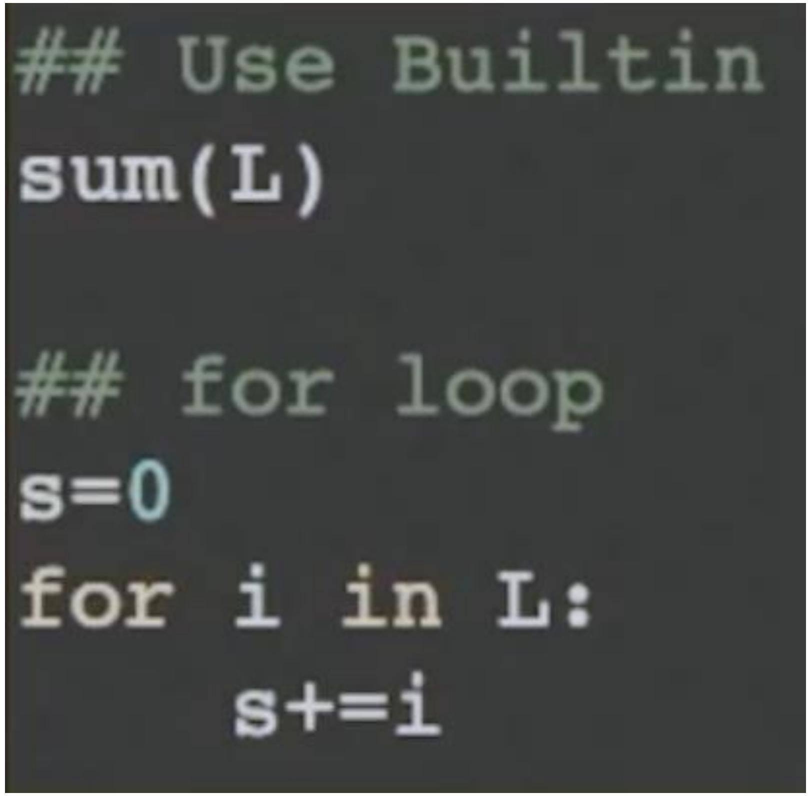
```
map(lambda x:x*x, L)
```

computation order is not specified

Reduce: compute the sum

- A list L=[3,1,5,7]
- Find the sum (16)

Traditional

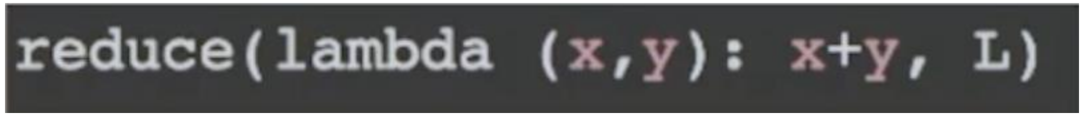


```
## Use Builtin
sum(L)

## for loop
s=0
for i in L:
    s+=i
```

compute from first to last in order

MapReduce



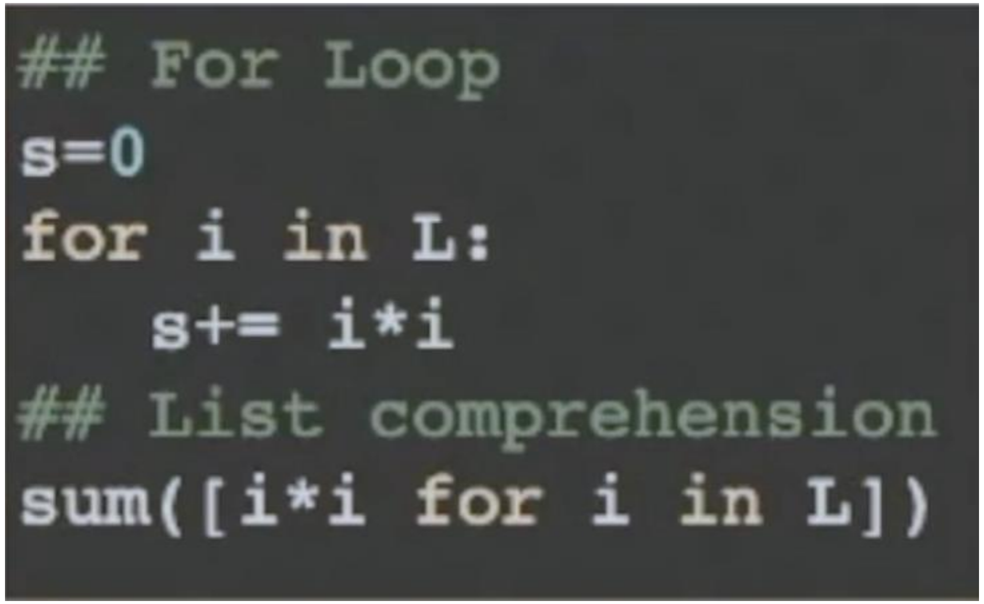
```
reduce(lambda (x,y): x+y, L)
```

computation order is not specified

Map + Reduce

- list L=[0,1,2,3]
- Compute the sum of the squares
- Note the differences

Traditional

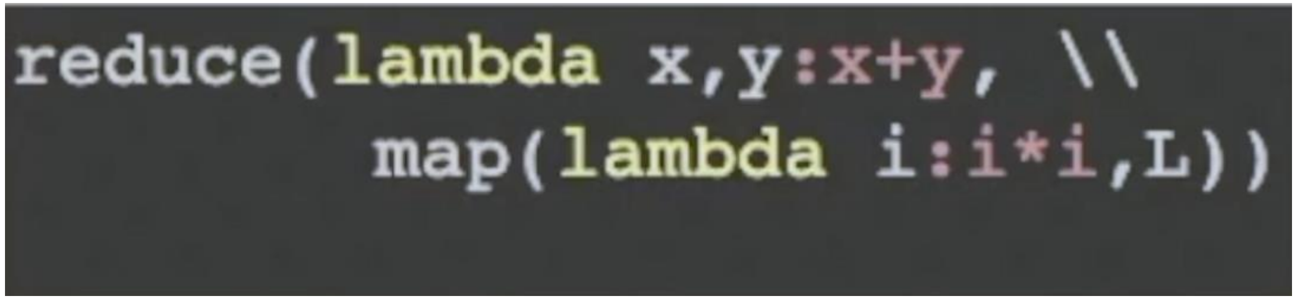


```
## For Loop
s=0
for i in L:
   s+= i*i
## List comprehension
sum([i*i for i in L])
```

compute from first to last in order

Immediate execution

MapReduce



```
reduce(lambda x,y:x+y, \\
        map(lambda i:i*i,L))
```

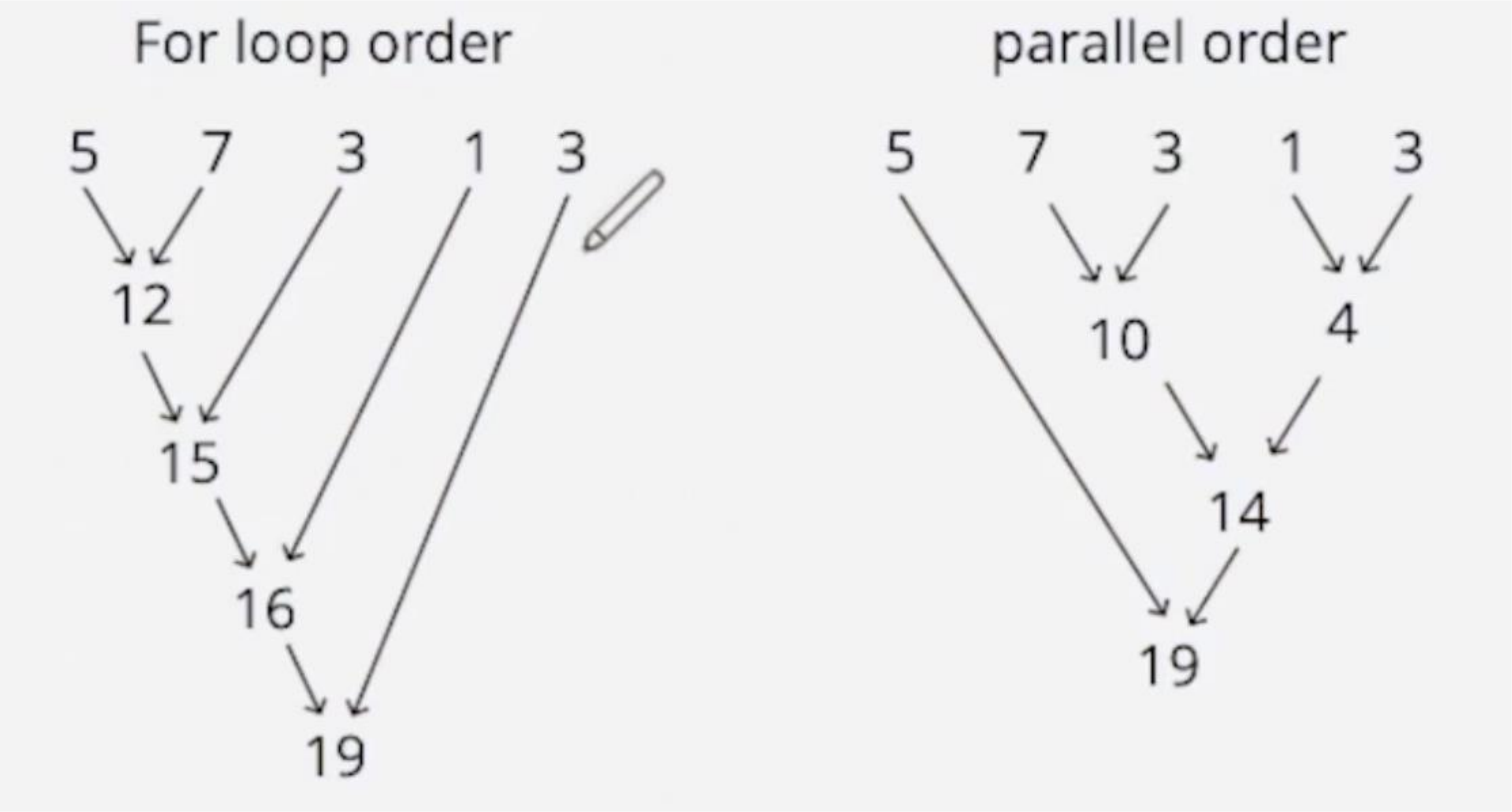
computation order is not specified

Execution **plan**

Order independence

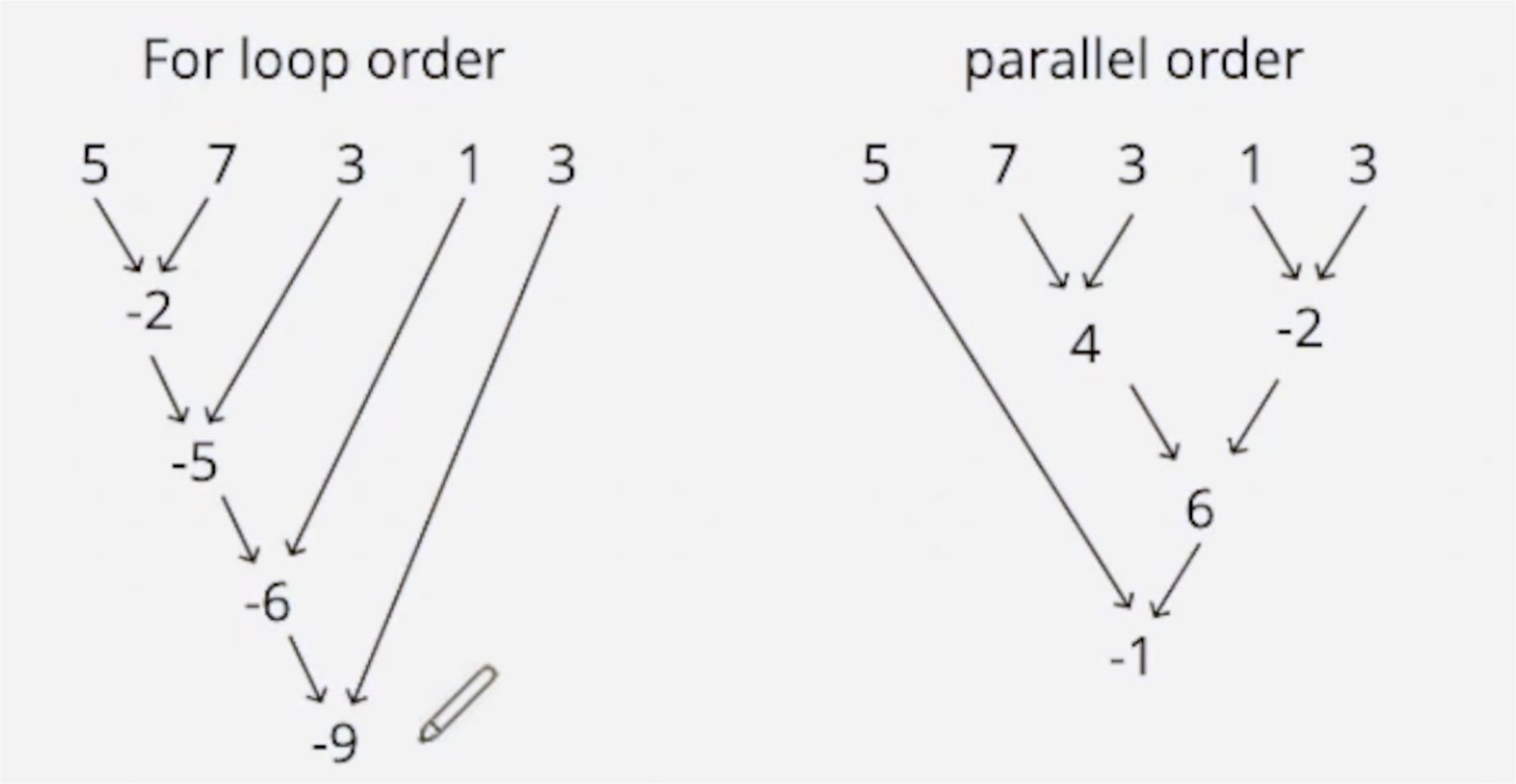
- The result of map or reduce must not depend on the order

sum does not depend on computation order



Result does not depend on order

difference depends on computation order



Result depends on order

Computing the average incorrectly

Average = data.reduce(lambda a,b: (a+b)/2)

data=[1,2,3], average is 2

Computed Average = ((1+2/2+3)/2 = 2.25

Computing the average correctly

```
sum,count = data.map(lambda x: (x,1))
        .reduce(lambda P1,P2:
                (P1[0]+P2[0], P1[1]+P2[1]))
```

Average = sum/count

```
        [1,2,3].map(lambda x: (x,1)) = [(1,1),(2,1),(3,1)]
        sum, count = [(1,1),(2,1),(3,1)].reduced() = 6,3
        average = 6/3 = 2
        data=[1,2,3], average is 2
```

Why Order Independence?

- Computation order can be chosen by compiler/optimizer.
- Allows for **parallel computation** of sums of subsets.
    - Modern hardware calls for parallel computation but parallel computation is very hard to program.
- Using MapReduce programmer **exposes** to the compiler opportunities for parallel computation.

Spark and MapReduce

- MapReduce is the basis for many systems.
- For big data: Hadoop and Spark.