Map Reduce
Map: square each item

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

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

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

Map-Reduce

```python
map(lambda x:x*x, L)
```
Reduce: compute the sum

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

```python
## Use Builtin
sum(L)

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

Map-Reduce

```python
reduce(lambda (x,y): x+y, L)
```
Map + Reduce

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

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

Map-Reduce

```python
reduce(lambda x,y:x+y, \
       map(lambda i:i*i,L))
```
The Wrong way

reduce(\lambda x,y: x+y*y)

- Map, Reduce operations should not depend on:
  - Order of items in the list (commutativity)
  - Order of operations (Associativity)
- It is this independence that allows parallel computation.
Order independence

- The result of map or reduce does not depend on the order
computation order of a sum

For loop order:
5
\downarrow
12
\downarrow
15
\downarrow
16
\downarrow
19

parallel order:
5
\downarrow
10
\downarrow
14
\downarrow
19

3
\downarrow
1
\downarrow
3

19

Result should not depend on order
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 map-reduce programmer exposes to the compiler opportunities for parallel computation.