map, reduce, pipelines, spark and all that

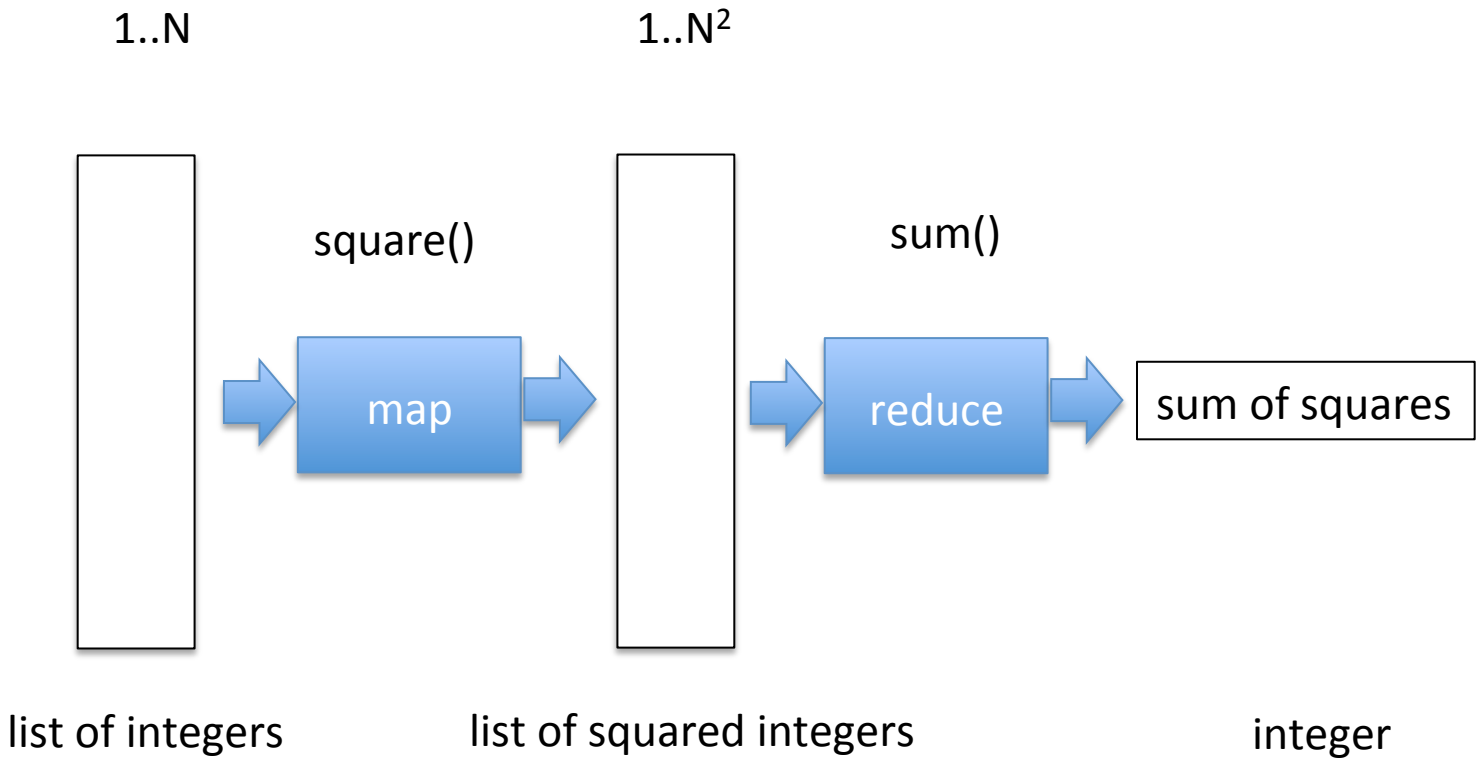
Computer Science for Bioinformatics

140.636

Map/Reduce

- **Map**: consumes a list and applies a function to each element of the list and returns a transformed list
- **Reduce**: consumes a list and applies a rolling calculation to sequential pairs of values in the list and returns a single value, e.g. summing a vector of values

Map/Reduce


1..N
1..N2
square()
sum()
map
reduce
sum of squares
list of integers
list of squared integers
integer

map/reduce in Perl

```
use List::Util ("reduce");

my @nums= 1..10;

sub square { return( $_ * $_) }
sub sum    { return $a+$b }

my $sum = reduce(\&sum, 0, map(square, @nums));

print ("the sum = $sum\n");
```

map() is built-in function and square() is our *callback* function for map()
reduce() is in List::Util and sum() is our *callback* function for reduce()

map/reduce in Python

```
import functools

nums = nums = list(range(1,11))
def square(x):
        return(x*x)

def sum(sumval, nextval):
        return(sumval+nextval)

result = functools.reduce(sum, list(map(square,nums)))
```

map() is built-in function and square() is our *callback* function for map()
reduce() is in functools and sum() is our *callback* function for reduce()

Why?

- Simpler to read than a for-loop (at least after you get used to it)
- A “programming pattern” that extends naturally to the “functional programming” paradigm.
- map() and reduce() functions can be implemented in an optimized fashion by the language developers so they are generally faster than for-loops, especially in interpreted languages, e.g. R, which brings us to...
- **The real reason is that it allows us to take advantage of the principle of pipelining at multiple levels.**

Conceptual introduction to pipelining

From CS232 Introduction to Computer Architecture, University of Illinois

https://www.youtube.com/watch?v=doJpguZFTe0

Example problem

- An illustrative problem: Suppose we want to count the number of occurrences of each word in Melville's Moby Dick

"Call me Ishmael. Some years ago—never mind how long precisely—having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever"

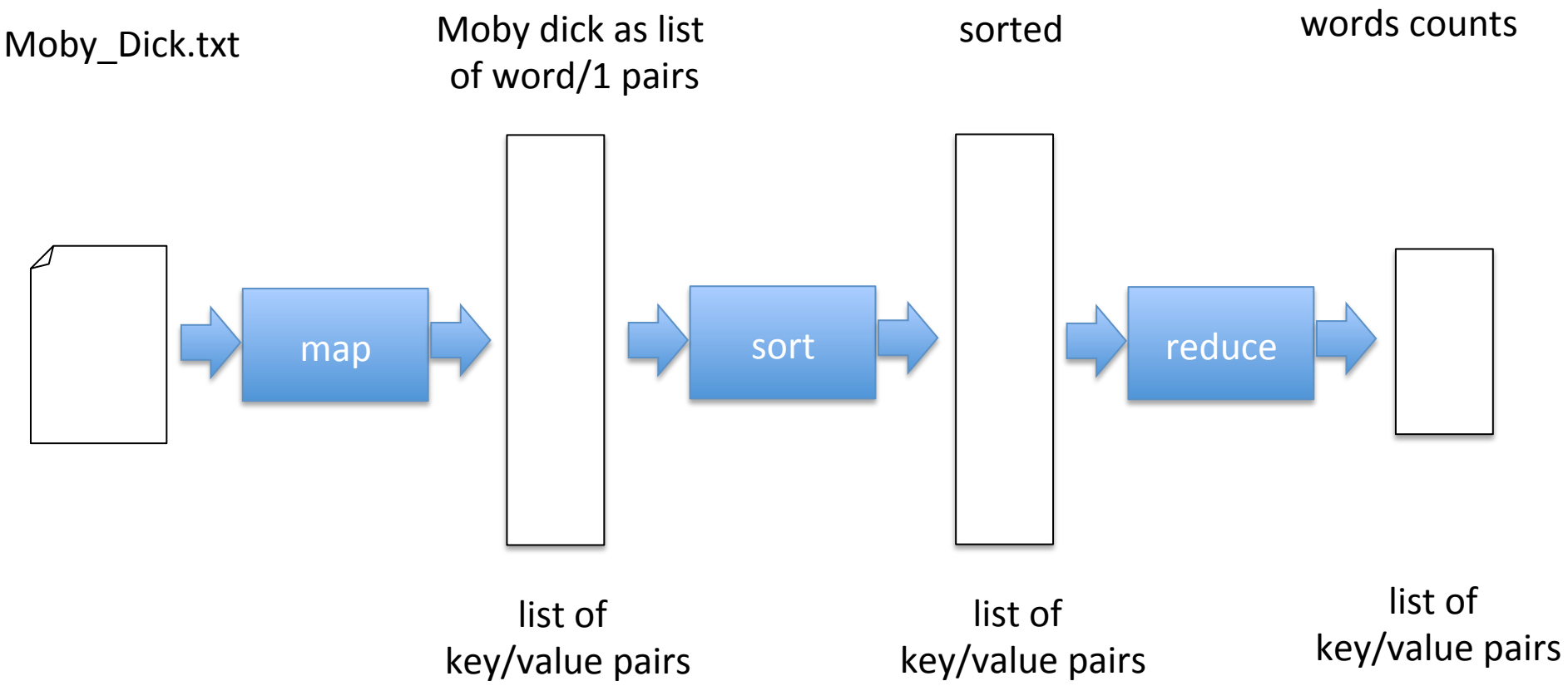
The obvious approach: use a dictionary to accumulate word occurrences

```
import sys
import re

counts={} # a key = word, value = counts

for line in sys.stdin:
   line = re.sub(r'^\W+|\W+$','',line)
   words = re.split(r'\W+',line)
   for word in words:
      word = word.lower()
      counts[word] = counts.get(a_word,0) +1

print(counts)
```

pipelined approaches to vector, list and dictionary processing

- Scatter/Gather
  - old-as the original supercompters (Cray-1)
- Map/Reduce
  - process lists of objects (not just numeric vectors)
  - recall Perl and Python examples
- Map/{shuffle, sort, filter}/Reduce
  - Process lists (key/object pairs)
  - disk-based: Apache Hadoop
  - memory-based: Apache Spark

Map/Sort/Reduce



Map/Sort/Reduce

- **Map:** transform a list of N objects (words) into a list of key/value pairs (e.g keys =words and value=1)
- **Sort:** order the list by keys
  (e.g. sort by key so identical words are together)
- **Reduce**: Reduce the sorted list into a shorter list of unique-keys/results

map.py

```
import sys
import re

counts={}

for line in sys.stdin:
   line = re.sub(r'^\W+|\W+$','',line)
   words = re.split(r'\W+',line)

   for word in words:
      print(word.lower() + "\t1")
```

key	value
Call	1
me	1
Ishmael	1
Some	1
years	1
ago	1
never	1
mind	1
how	1
long	1
precisely	1
having	1
little	1
or	1
=	

sort

```
./map.py < moby_dick.txt | sort
```

key	value
zodiac	1
zogranda	1
zone	1
zone	1
zone	1
zone	1
zone	1
zoned	1
zoned	1
zones	1
zones	1
zones	1
zoology	1
zoology	1
zoroaster	1

reduce.py

```
# initialize – read & process first line
line = sys.stdin.readline()
prev_key,value = line.split('\t')
running_sum = int(value)

# process: read, process, print results
for line in sys.stdin:
    key, value = line.split('\t')
    if key != prev_key:
        print( str(running_sum)+'\t'+ prev_key)
        prev_key = key
        running_sum = 0
    running_sum = running_sum + int(value)

# clean-up print last result
print( str(running_sum) + '\t' + prev_key)
```

Demo & discussion

- Demo: on a compute node:

```
cd users/sph140636/shared/code_examples/mapreduce
./wordcount.sh
```

- Discussion
  - used disk instead of memory
  - unbuffered i/o in map.py and reduce.py
  - A trivial key/value output in map stage
  - Only uses a single core at a time
  - A small data set so not really faster than the ‘dumb’ approach

Spark

- Spark
  - Exellent tutorials at Berkeley
  - Spark installs and runs well on MacOS
  - Do not install spark on cluster − it does not play well with the grid engine
  - http://ampcamp.berkeley.edu/6/#agenda
  - http://ampcamp.berkeley.edu/6/exercises/