Performance, Parallelism, and Distributed Data Analytics with Spark

David E. Culler

CS8 – Computational Structures in Data Science

http://inst.eecs.berkeley.edu/~cs88

Lecture 13

April 25, 2016

Computational Concepts Toolbox

- Data type: values, literals, operations,
- Expressions, Call expression
- Variables
- Assignment Statement
- Sequences: tuple, list
- Dictionaries
- Data structures
- Tuple assignment
- Function Definition Statement
- Conditional Statement
- Iteration: list comp, for, while
- Lambda function expr.

- Higher Order Functions
  - as Values, Args, Results
- Higher order function patterns
  - Map, Filter, Reduce
  - Function factories
- Recursion
  - Linear, Tail, Tree
- Abstract Data Types
- Mutation
- Object Oriented Programming
- Classes
- Iterators and Generators
- Exceptions
  - assert, try, except, raise

Today: Performance and Parallelism

- **Understanding ways of looking at performance**
  - Complexity – asymptotic scaling
  - Amdahl's Law – impact of enhancements (including parallelism)
- **Data analytics in the cloud – SPARK**
  - Map / reduce paradigm
  - RDDs
  - Arrays, Key-Value, Data frames / Tables
- **HKN survey before lab**
- **Lab – Hands on with DatabBricks / SPARK**

- **Administrative**
  - Next week review, No new homework
  - Final: FRIDAY, MAY 13, 2016 8-11A, Location: 306 SODA
  - Review session to be scheduled

Complexity – asymptotic analysis

- **Example: Matrix Multiply**
  - **How many Multiplies? Adds? Ops? How much time ?**
  - **As a function of n ?**

```
for i in 0 to n-1:
    for j in 0 to n-1:
        C[i][j] = 0
        for k in 0 to n-1:
            C[i][j] = C[i][j] + A[i][k]*B[k][j]
```

We say it is $O(n^3)$ "big-O of n^3" as an *asymptotic upper bound*

time(n) < $c \cdot n^3$, for some suitably large constant *c* for any instance of the inputs of size n.

A subtle example

- **What is the "complexity" of finding the average number of factors of numbers up to n?**

```
def factors(n):
  return [x for x in range(2, max(n, ceil(sqrt(n))))
                                       if n % x == 0]

def ave_factor(n):
    all_factors = map(factors, range(n))
    all_lens = map(len, all_factors)
    return sum(all_lens)/n
```

$n^{1/2}$

n

```
from timeit import default_timer as timer

def timeit(fun):
    """ Rtn timer for fun(i) in secs. """
    def timer_fun(i):
        start = timer()
        fun(i)
        end = timer()
        return (end-start)
    return timer_fun
```

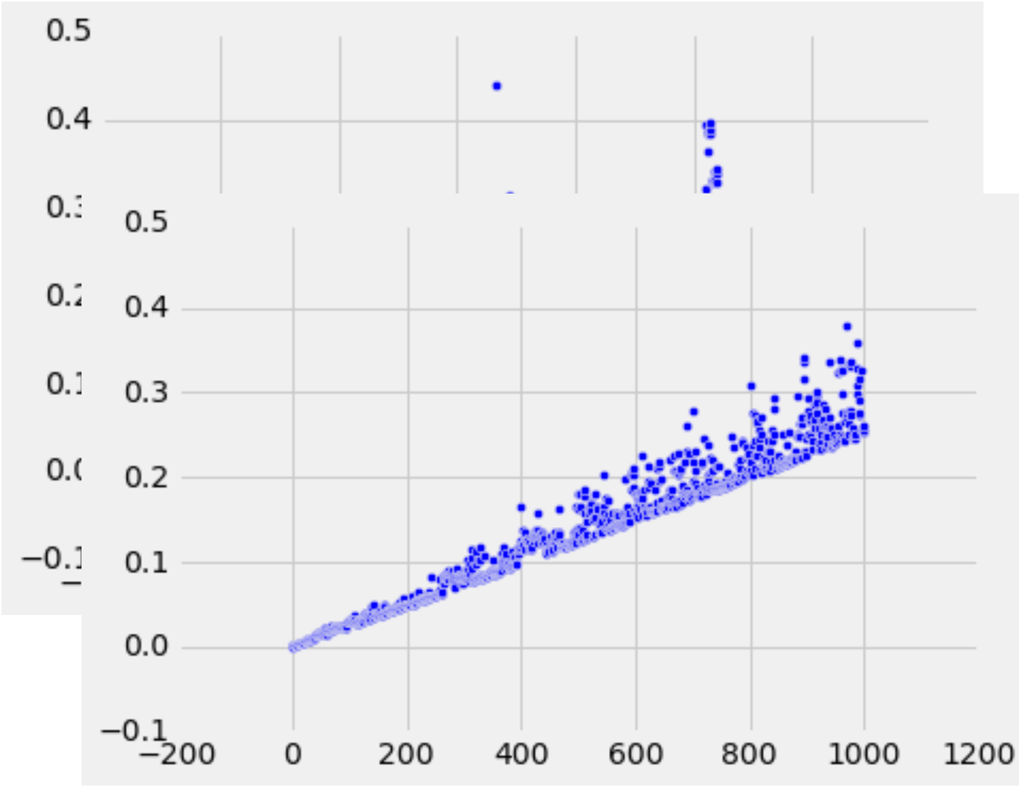
How long does factors take?

In [9]:
```
tbl = Table().with_column('n', np.arange(0,1000, 1))
tbl['factors'] = tbl.apply(factors, 'n')
tbl['n_factors'] = tbl.apply(len, 'factors')
tbl['secs'] = tbl.apply(timeit(factors), 'n')
tbl
```

Out[9]:

n	factors	n_factors	secs
0	[]	0	9.76503e-06
1	[]	0	2.40898e-06
2	[]	0	1.34797e-06
3	[]	0	3.49898e-06
4	[2]	1	2.74903e-06
5	[]	0	2.43704e-06
6	[2, 3]	2	3.019e-06
7	[]	0	2.78e-06
8	[2, 4]	2	3.28396e-06
9	[3]	1	3.74601e-06

... (990 rows omitted)



A subtle example

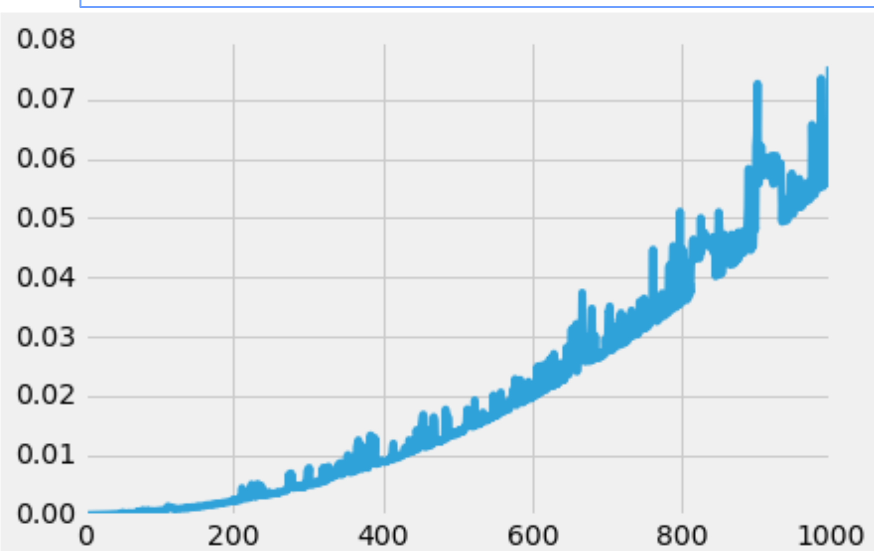
- **What is the complexity of finding the average number of factors of numbers up to n?**

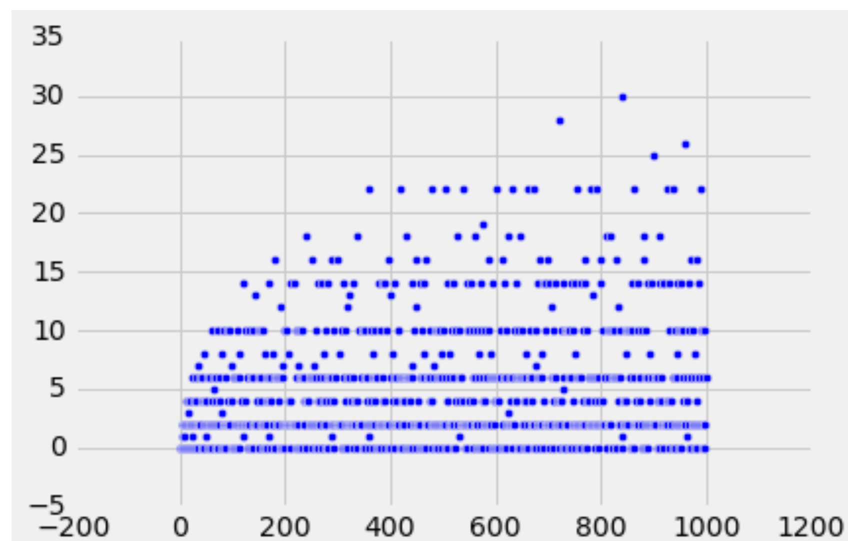
```
def factors(n):
  return [x for x in range(2, max(n, ceil(sqrt(n))))
                                          if n % x == 0]

def ave_factor(n):
    all_factors = map(factors, range(n))
    all_lens = map(len, all_factors)
    return sum(all_lens)/n
```

$n^{1/2}$

n





Amdahl's Law

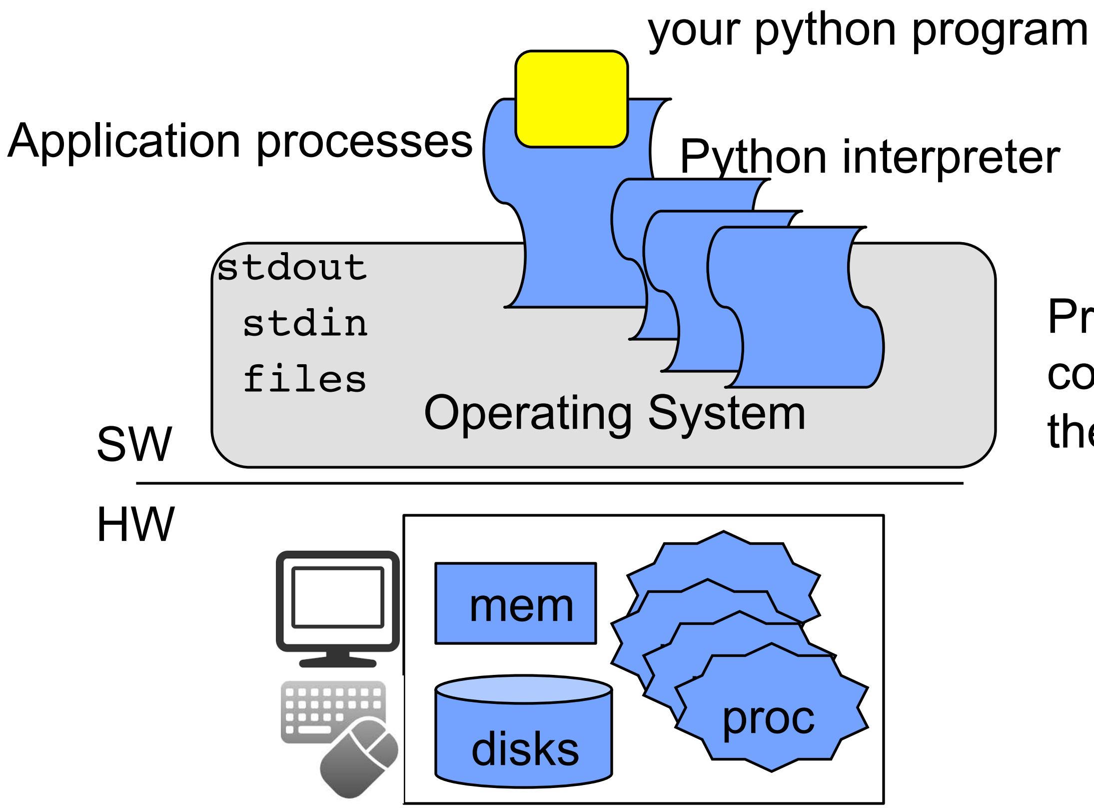
- Let $T_1(n)$ be the time to execute the program serially, and
- $T_p(n)$ be the time with parallelism p, and
- s_n be the fraction of the program that remains serial when parallelized

- SpeedUp(n) = $T_1(n) / T_p(n)$

$$<= \frac{T_1(n)}{s_n T_1(n) + (1-s_n)\, T_1(n)/p} < 1/s_n$$

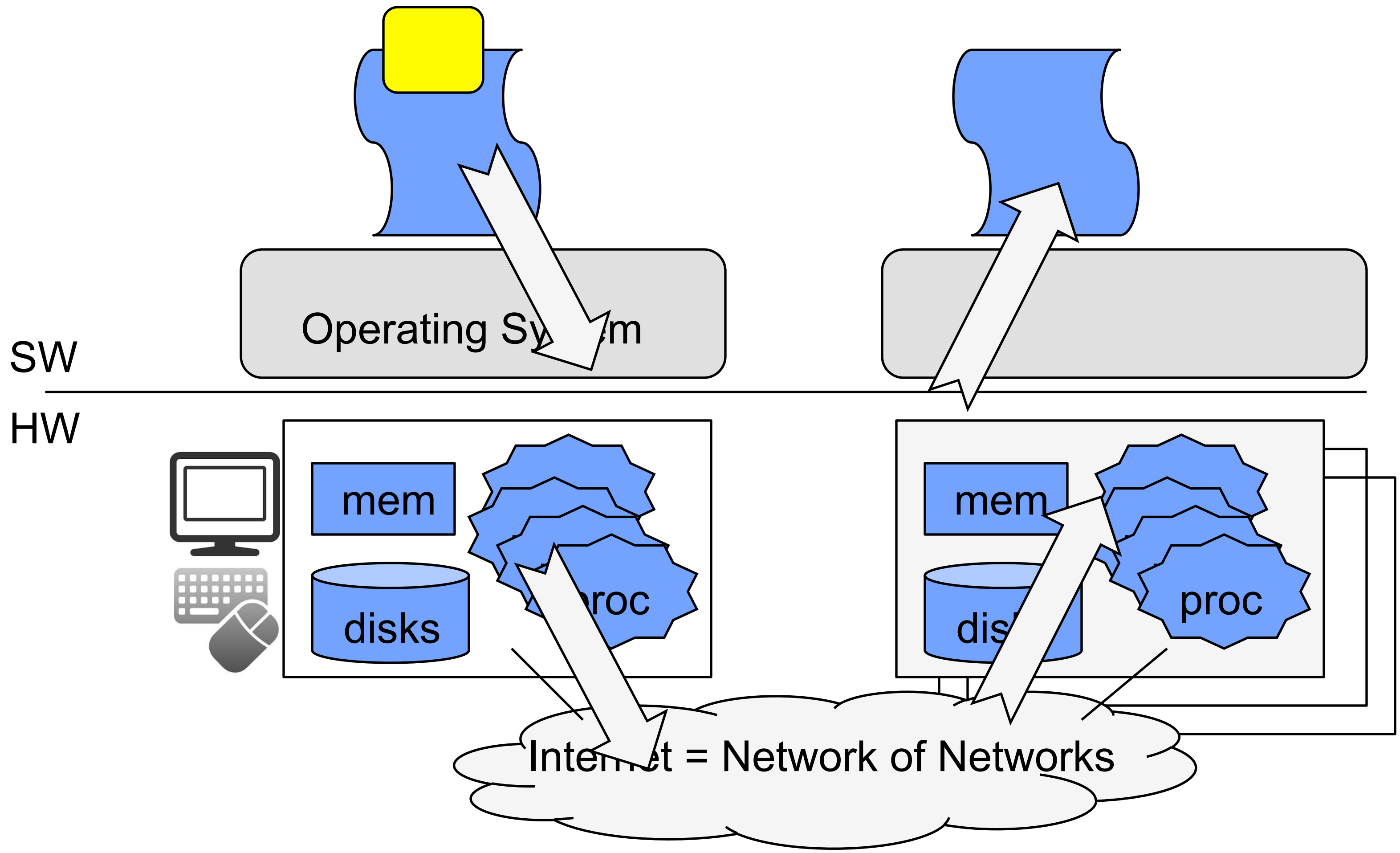
- Often, as the data gets large, the work that can be parallelized grows faster than the size of the data

Layers of Computer Systems

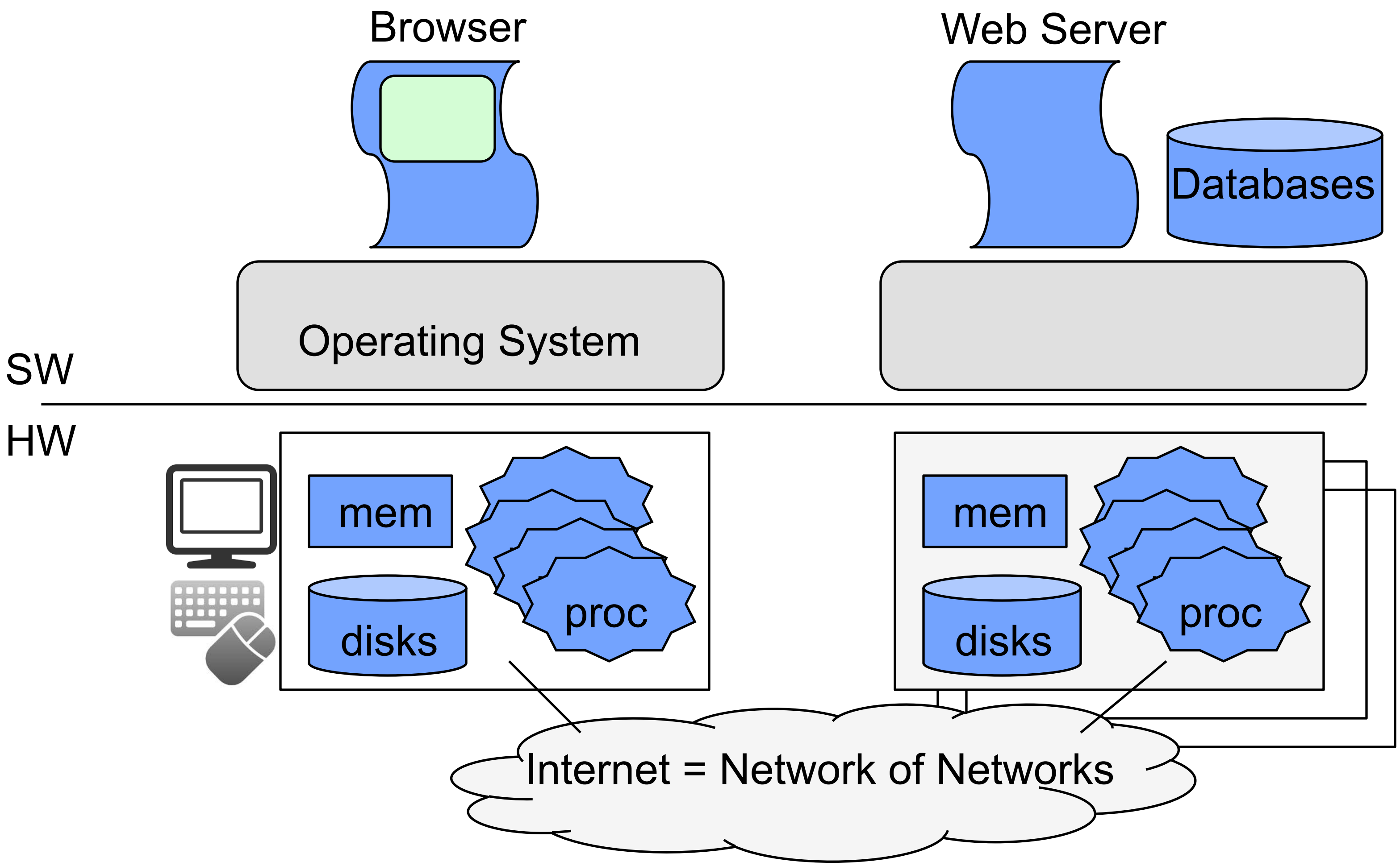


Provide a stable, consistent, convenient "virtualization" of the underlying hardware

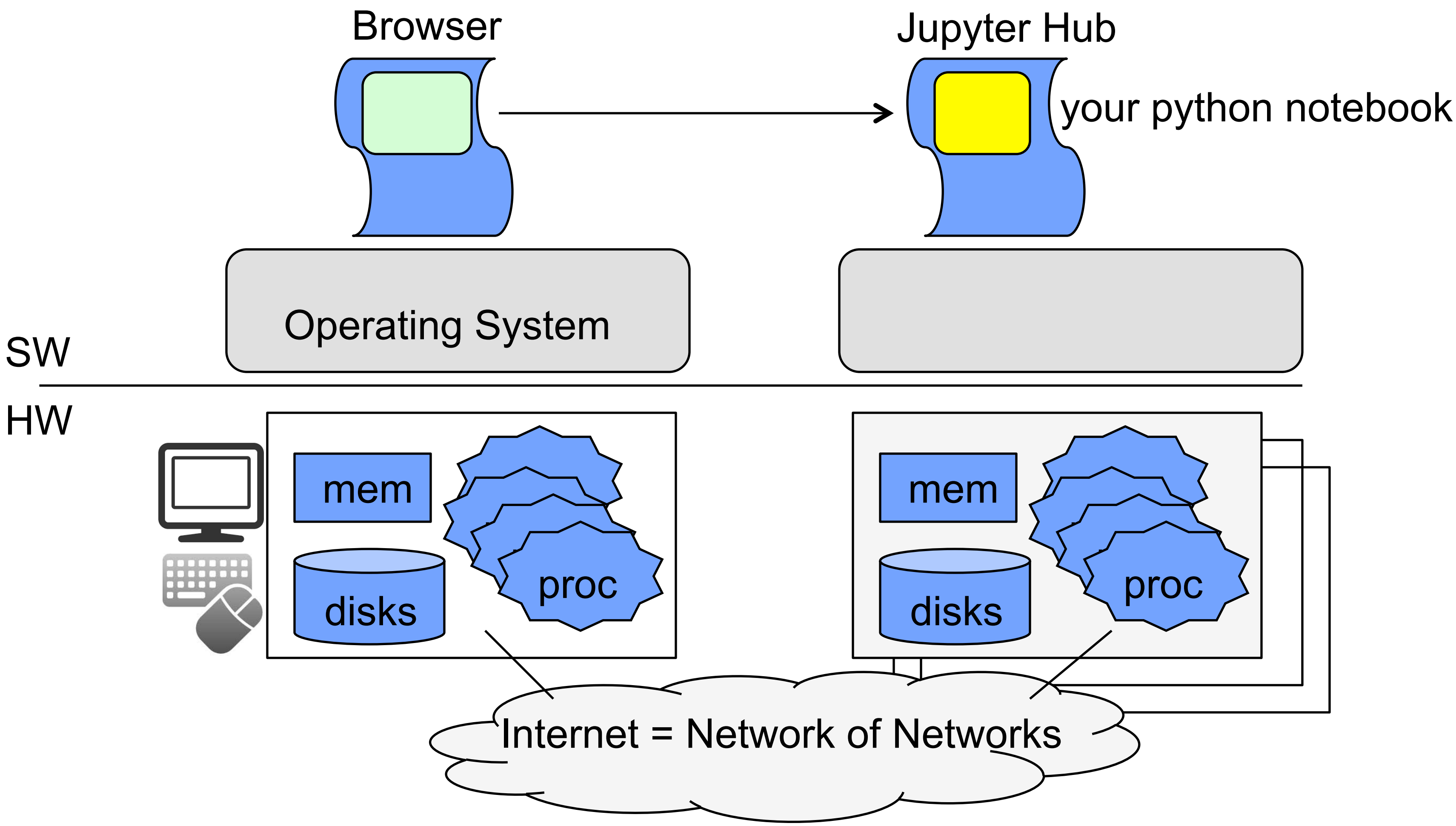
Distributed Computer Systems



Distributed Computer Systems

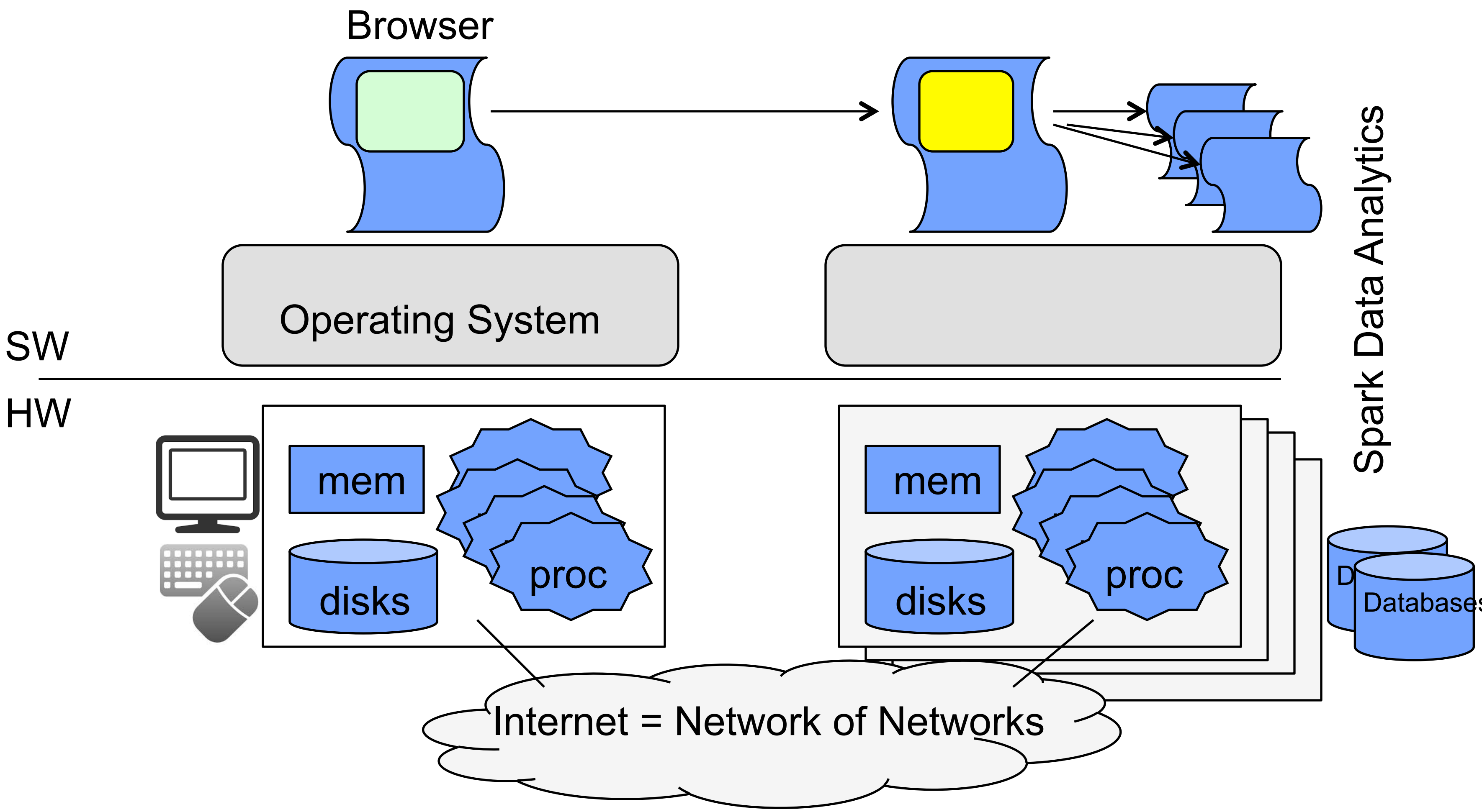


In Data8



Browser
Jupyter Hub
your python notebook
Operating System
SW
HW
mem
disks
proc
mem
disks
proc
Internet = Network of Networks

… on BIG DATA


Browser
Spark Data Analytics
Operating System
SW
HW
mem
disks
proc
mem
disks
proc
Databases
Internet = Network of Networks

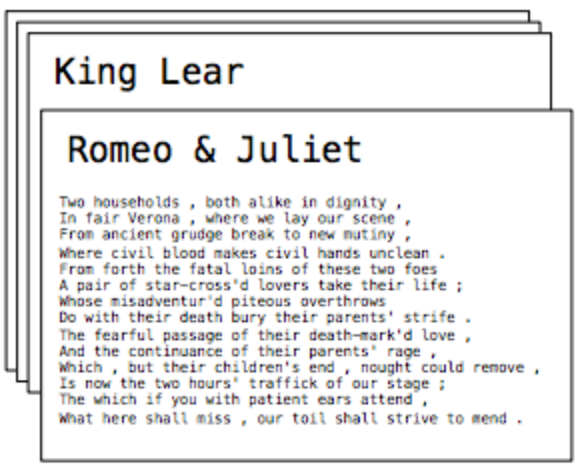
Big Data Examples

- Facebook daily logs: 60 Terabytes (60,000 GB)
- 1,000 Genomes: 200 TB
- Google web index: 10+ Petabytes (10,000 TB)
- Time to read 1 TB @ 100 MB/s ? – 3 hours

- Clusters – thousands of complete computer systems, networked closely
  - (mostly) independent failures
  - Engineered at massive scale

Apache Spark (from Berkeley)

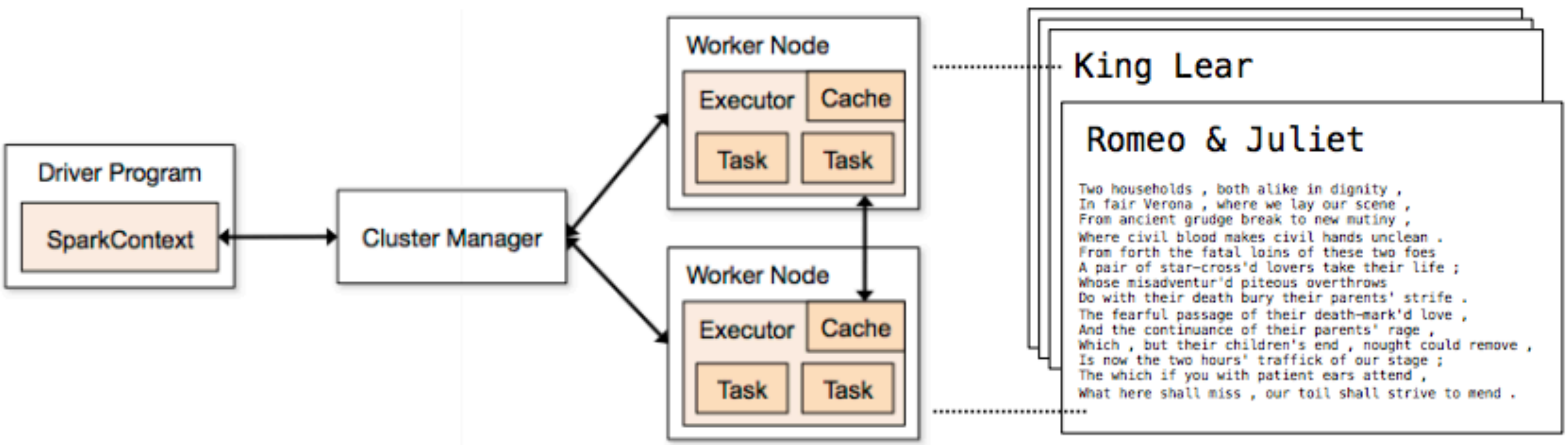
- Data processing system that provides a simple interface to analytics on large data
- A Resilient Distributed Dataset (RDD) is a collection of values or key-value pairs
- Support the operations you are familiar with
  - Data-Parallel: map, filter, reduce
  - Database: join, union, intersect
  - OS: sort, distinct, count
- All of can be performed on RDDs that are partitioned across machines



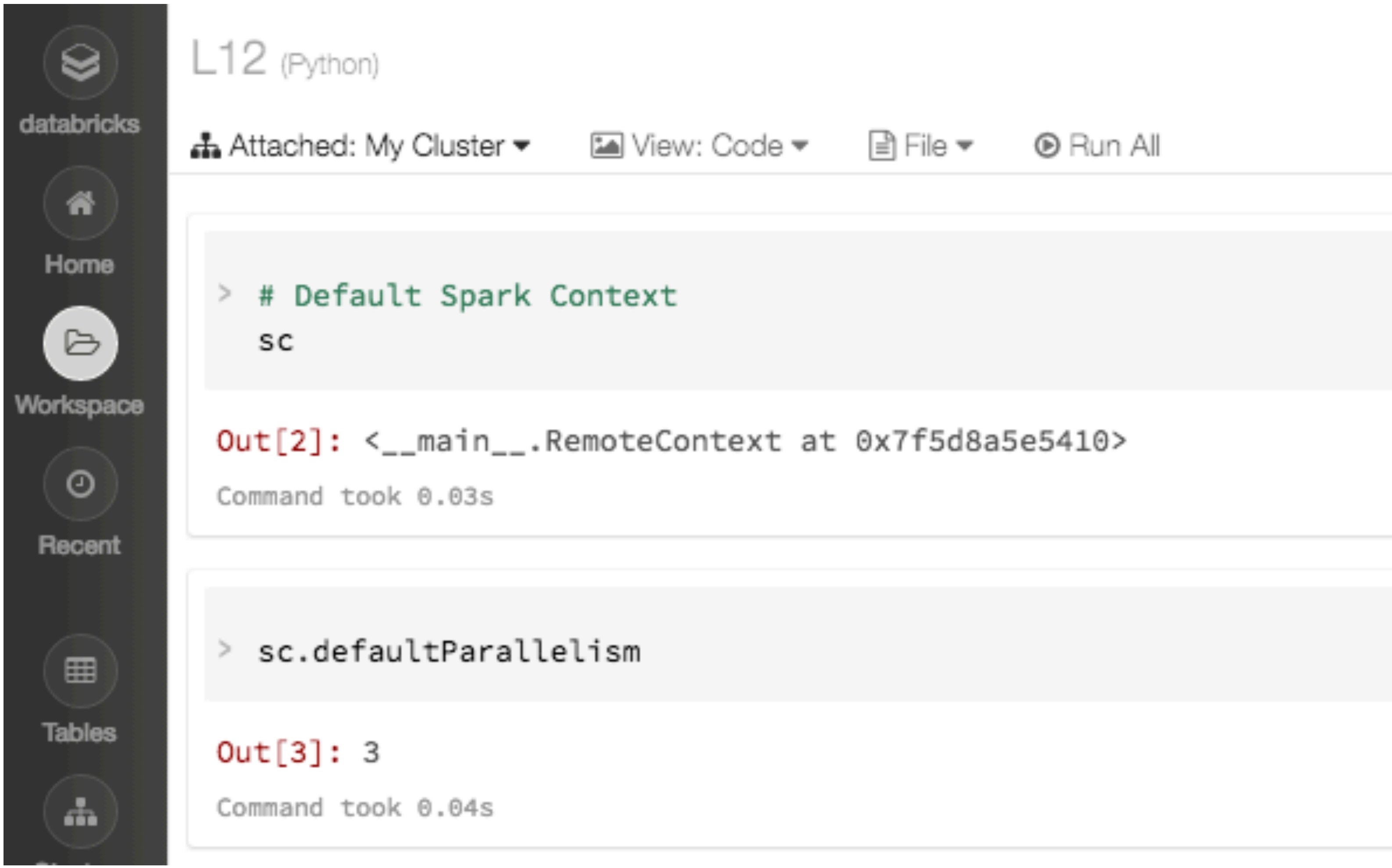
Spark Execution Model

Processing is defined centrally and executed remotely

- A RDD is distributed over workers
- A driver program defines transformations and actions on RDDs
- A cluster manager assigns task to workers
- Workers perform computation, store data, & communicate with each other
- Final results communicate back to driver



Spark Context



RDD of values

```
n = 1000
data = range(n)
rdd_data = sc.parallelize(data)
rdd_data
```

```
Out[5]: ParallelCollectionRDD[31] at parallelize at PythonRDD.scala:423
Command took 0.07s
```

Looking at results

```
> n = 1000
  data = range(n)
  rdd_data = sc.parallelize(data)
  rdd_data
```

Out[5]: ParallelCollectionRDD[31] at parallelize at

Command took 0.07s

```
> rdd_data.take(10)
```

▾ (1) Spark Jobs

▸ Job 0 View (Stages: 1/1)

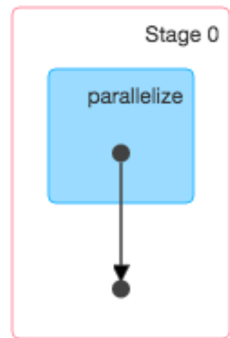
Out[6]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Command took 1.72s

```
>
```

Sun 24 April

▾DAG Visualization



Completed Stages (1)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input
0	2873135457951023881	rdd_data.take(10) runJob at PythonRDD.scala:393 +details	2016/04/24 23:26:17	0.6 s	1/1	

Map / Collect

```
> n = 12
  data = xrange(2,n+2)
  rdd_data = sc.parallelize(data)
  rdd_data.map(factors).collect()
```

▾ (1) Spark Jobs

▸ Job 4 View (Stages: 1/1)

```
Out[12]: [[], [], [2], [], [2, 3], [], [2, 4], [3], [2, 5], [], [2, 3, 4, 6], []]
Command took 0.12s
```

Data Distribution

```
> # Let's see how the data is distributed - glom
  rdd_data.glom().collect()
```

▸ (1) Spark Jobs

```
Out[14]: [[2, 3, 4, 5], [6, 7, 8, 9], [10, 11, 12, 13]]
Command took 0.07s
```

```
> sc.parallelize(range(100), 10).glom().collect()
```

▸ (1) Spark Jobs

```
Out[15]:
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
 [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
 [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
 [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
 [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
 [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
 [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
 [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
 [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
 [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]]
Command took 0.27s
```

Distribute/Map/Reduce

Attached: My Cluster View: Code File Run All Publish Comments Revision history

```
> sc.parallelize(range(100), 10).glom().collect()
```

(1) Spark Jobs

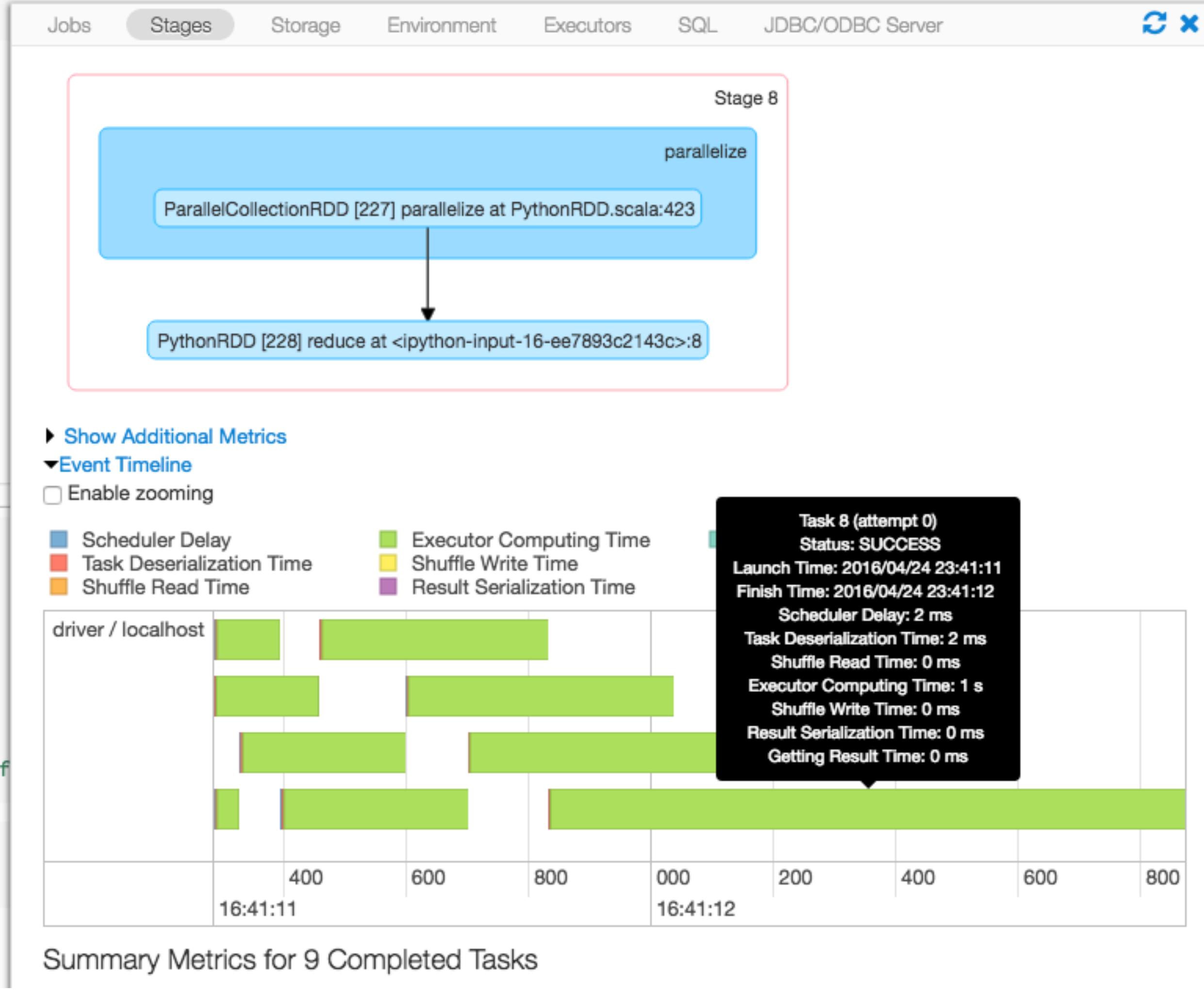
```
Out[15]:
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
 [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
 [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
 [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
 [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
 [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
 [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
 [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
 [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
 [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]]
Command took 0.27s
```

```
> # Spread, transform, reduce on a lot more data
  from operator import add
  n = 10000
  data = range(n)
  rdd_data = sc.parallelize(data,9)
  rdd_factors = rdd_data.map(factors)
  rdd_lens = rdd_factors.map(len)
  rdd_lens.reduce(add)/float(n) # python 2 be caref
```

(1) Spark Jobs

Job 8 View (Stages: 1/1)

```
Out[16]: 7.3646
Command took 1.71s
```



Encapsulating the Parallelism

```
> def p_ave_factors(n):
      data = range(n)
      rdd_data = sc.parallelize(data,sc.defaultParallelism*3)
      rdd_factors = rdd_data.map(factors)
      rdd_lens = rdd_factors.map(len)
      return rdd_lens.reduce(add)/float(n)
```

Command took 0.08s

```
> p_ave_factors(10000)
```

▸ (1) Spark Jobs

Out[20]: 7.3646

Command took 1.71s

Summary: RDD operations (so far)

- **Transformation**
  https://spark.apache.org/docs/latest/programming-guide.html#transformations
  - map(fun), filter(fun)
  - flatMap(fun) – each item may be mapped to zero or more outputs
  - sample, union, intersection, distinct
  - join
- **Action**
  - reduce(fun), collect(), count(), first(), take(n)

Key-value RDDs

```
> # Key value stores, start as list of tuples, not dictionary
  from operator import add
  d = [('one', 1), ('two', 2), ('one', 3), ('free', 5), ('free', 42)]
  d_rdd = sc.parallelize(d)
  d_rdd.groupByKey().collect()
```

▸ (1) Spark Jobs

```
Out[1]:
[('one', <pyspark.resultiterable.ResultIterable at 0x7f7e0e424c50>),
 ('free', <pyspark.resultiterable.ResultIterable at 0x7f7e0cb30450>),
 ('two', <pyspark.resultiterable.ResultIterable at 0x7f7e0cb30150>)]
Command took 1.34s
```

Map the values in a group
- not add

```
> d_rdd.groupByKey().mapValues(sum).collect()
```

▸ (1) Spark Jobs

```
Out[6]: [('one', 4), ('free', 47), ('two', 2)]
Command took 0.38s
```

Key-Value RDD operations

```
> d_rdd.reduceByKey(add).collect()
```

▸ (1) Spark Jobs

```
Out[7]: [('one', 4), ('free', 47), ('two', 2)]
Command took 0.28s
```

```
> d_rdd.countByKey()
```

▸ (1) Spark Jobs

```
Out[8]: defaultdict(<type 'int'>, {'two': 1, 'free': 2, 'one': 2})
Command took 0.17s
```

```
> d_rdd.reduceByKeyLocally(add)
```

▸ (1) Spark Jobs

```
Out[10]: {'free': 47, 'one': 4, 'two': 2}
Command took 0.12s
```

Summary: RDD operations (cont)

- **Transformation** https://spark.apache.org/docs/latest/programming-guide.html#transformations
  - map(fun), filter(fun)
  - flatMap(fun) – each item may be mapped to zero or more outputs
  - sample, union, intersection, distinct, join
  - groupByKey(), reduceByKey(fun), aggregateByKey, sortByKey
- **Action**
  - reduce(fun), collect(), count(), first(), take(n)
  - takeSample
  - countByKey

Building an RDD from a text file

```
> import os.path
  baseDir = os.path.join('databricks-datasets')
  inputPath = os.path.join('cs100', 'lab1', 'data-001', 'shakespeare.txt')
  fileName = os.path.join(baseDir, inputPath)

  shakespeareRDD = sc.textFile(fileName)
  shakespeareRDD.take(14)
```

▸ (1) Spark Jobs

```
Out[12]:
[u'1609',
 u'',
 u'THE SONNETS',
 u'',
 u'by William Shakespeare',
 u'',
 u'',
 u'',
 u'                     1',
 u'  From fairest creatures we desire increase,',
 u"  That thereby beauty's rose might never die,",
 u'  But as the riper should by time decease,',
 u'  His tender heir might bear his memory:',
 u'  But thou contracted to thine own bright eyes,']
Command took 0.28s
```

Count, Filter and stats

```
> shakespeareRDD.count()
```

▸ (1) Spark Jobs

```
Out[13]: 122395
Command took 0.63s
```

```
> import string
  string.split
  sp_words = shakespeareRDD.map(string.split).filter(lambda x: len(x) > 0)
  sp_words.map(len).stats()
```

▸ (1) Spark Jobs

```
Out[14]: (count: 112902, mean: 7.82377637243, stdev: 2.70356395722, max: 21.0, min: 1.0)
Command took 1.31s
```

flatMap

```
> def clean_word(s):
    res = ""
    for c in s:
      if c in string.ascii_letters:
        res += c
    return res
```

Command took 0.07s

```
> sp_flat = sp_words.flatMap(lambda x:x).map(string.lower).map(clean_word).filter(lambda x: len(x)>0)
  sp_flat.take(10)
```

(1) Spark Jobs

```
Out[16]:
[u'the',
 u'sonnets',
 u'by',
 u'william',
 u'shakespeare',
 u'from',
 u'fairest',
 u'creatures',
 u'we',
 u'desire']
```

Command took 0.22s

Values => Key-Value

```
from operator import add
sp_kv = sp_flat.map(lambda x: (x,1)).reduceByKey(add).sortBy(lambda x: x[1], ascending=False)
sp_kv.take(20)
```

(3) Spark Jobs

- Job 13 View (Stages: 2/2)
- Job 14 View (Stages: 1/1, 1 skippe
- Job 15 View (Stages: 2/2, 1 skippe

```
Out[17]:
[(u'the', 27361),
 (u'and', 26028),
 (u'i', 20681),
 (u'to', 19150),
 (u'of', 17463),
 (u'a', 14593),
 (u'you', 13615),
 (u'my', 12481),
 (u'in', 10956),
 (u'that', 10890),
 (u'is', 9134),
 (u'not', 8497),
 (u'with', 7771),
 (u'me', 7769),
 (u'it', 7678),
 (u'for', 7558),
 (u'be', 6857),
 (u'his', 6857),
 (u'your', 6655),
 (u'this', 6602)]
```

Details for Job 15

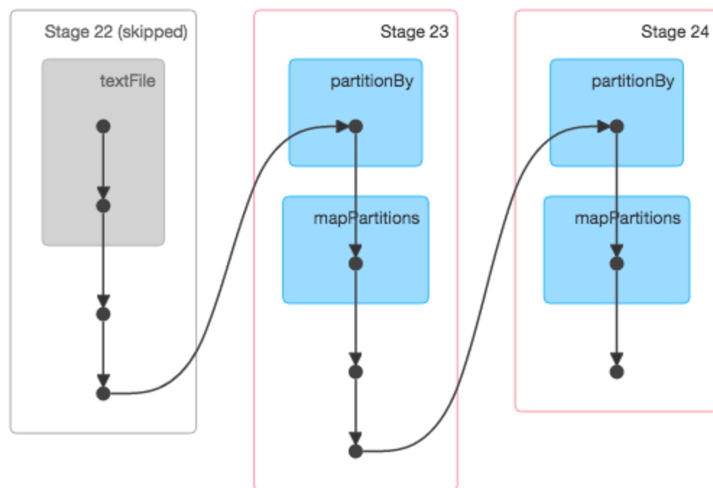
Status: SUCCEEDED

Job Group: 628891043112620844_9007556272146961988_0dc5beb06c1d44c3904f04d795e41b21

Completed Stages: 2

Skipped Stages: 1

- Event Timeline
- DAG Visualization



Completed Stages (2)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Inpu
24	628891043112620844	from operator import add sp_kv = sp_flat.map(la... runJob at PythonRDD.scala:393 +details	2016/04/25 05:07:19	21 ms	1/1	
23	628891043112620844	from operator import add sp_kv = sp_flat.map(la... sortBy at <ipython-input-17-a79f625fad0b>:2 +details	2016/04/25 05:07:19	0.1 s	2/2	

Data Frames / SQL

```
> if "mnt/" not in [x.name for x in dbutils.fs.ls("/")] or 'cs61a/' not in [x.name for x in dbutils.fs.ls("/mnt")]:
    dbutils.fs.mount('s3n://AKIAJUIYIBOAUUTJ3G5A:SU%2FsifB7wuzeWexDJCMTBVxG7MLIbZkk4ZcB+qzd@berkeley-cs61a','/mnt/cs61a/')
  reviews_dataset = '/mnt/cs61a/yelp_reviews_dataset_small.json'
  reviews = sqlContext.read.json(path=reviews_dataset)
  reviews.show()
```

▸ (2) Spark Jobs

```
+--------------------+----------+--------------------+-----+--------------------+------+--------------------+-------+
|         business_id|      date|           review_id|stars|                text|  type|             user_id|  votes|
+--------------------+----------+--------------------+-----+--------------------+------+--------------------+-------+
|vcNAWiLM4dR7D2nww...|2007-05-17|15SdjuK7DmYqUAj6r...|    5|dr. goldberg offe...|review|Xqd0DzHaiyRqVH3WR...|[1,0,2]|
|vcNAWiLM4dR7D2nww...|2014-01-02|kMu0knsSUFW2DZXqK...|    5|Top notch doctor ...|review|jE5xVugujSaskAoh2...|[0,0,0]|
|vcNAWiLM4dR7D2nww...|2014-01-08|onDPFgNZpMk-bT1zl...|    5|Dr. Eric Goldberg...|review|QnhQ8G51XbUpVEyWY...|[0,0,0]|
|vcNAWiLM4dR7D2nww...|2014-08-01|bOJD0Kc3wGioat3oS...|    1|I'm writing this ...|review|tAB7GJpUuaKF4W-3P...|[0,0,1]|
|vcNAWiLM4dR7D2nww...|2014-12-12|QzjRXUNSGk3PySEcg...|    5|I love Dr. Goldbe...|review|GP-h9colXgkT79BW7...|[0,0,0]|
|UsFtqoBl7naz8AVUB...|2014-10-29|7N9j5YbBHBW6qguE5...|    2|Wing sauce is lik...|review|PP_xoMSYlGr2pb67B...|[0,0,0]|
|cE27W9VPgO88Qxe4o...|2014-07-11|S-G0D8Cy7PnqShoBZ...|    4|I drove by yester...|review|ljwgUJowB69klaR8A...|[0,0,0]|
|HZdLhv6COCleJMo7n...|2013-06-10|fBQ69-NU9ZyTjjS7T...|    5|THANK YOU ROB! i ...|review|JbAeIYc89Sk8SWmrB...|[7,3,7]|
|HZdLhv6COCleJMo7n...|2014-09-04|UzMViMQZuSxOr5wrr...|    4|I visited this st...|review|zo_soThZw8eVglPbC...|[0,0,0]|
|mVHrayjG3uZ_RLHkL...|2013-03-15|jVVv_DA5mCDB6medi...|    5|Can't miss stop f...|review|m1FpV3EAeggaAdfPx...|[0,0,0]|
|mVHrayjG3uZ_RLHkL...|2014-09-29|5uyYmniYyIB_wtKty...|    4|Wonderful reuben....|review|u9ULAsnYTdYH65Haj...|[0,0,0]|
|KayYbHCt-RkbGcPdG...|2010-10-11|v_uEDbK5fP1UJpkXN...|    4|This would be my ...|review|ay9H1RpjbBkaiXGxf...|[2,2,2]|
|KayYbHCt-RkbGcPdG...|2011-12-22|UrukGX1emhSRe2fGd...|    3|Good for cheap dr...|review|bcwr1bFov3PSa1FiG...|[0,0,0]|
```

Data frames: select, filter

```
> reviews.select('stars', 'text').show()
```

▸ (1) Spark Jobs

```
+-----+--------------------+
|stars|                text|
+-----+--------------------+
|    5|dr. goldberg offe...|
|    5|Top notch doctor ...|
```

```
> reviews.filter(reviews['stars'] == 5).show()
```

▸ (1) Spark Jobs

```
+--------------------+----------+--------------------+-----+--------------------+------+--------------------+-------+
|         business_id|      date|           review_id|stars|                text|  type|             user_id|  votes|
+--------------------+----------+--------------------+-----+--------------------+------+--------------------+-------+
|vcNAWiLM4dR7D2nww...|2007-05-17|15SdjuK7DmYqUAj6r...|    5|dr. goldberg offe...|review|Xqd0DzHaiyRqVH3WR...|[1,0,2]|
|vcNAWiLM4dR7D2nww...|2014-01-02|kMu0knsSUFW2DZXqK...|    5|Top notch doctor ...|review|jE5xVugujSaskAoh2...|[0,0,0]|
|vcNAWiLM4dR7D2nww...|2014-01-08|onDPFgNZpMk-bT1zl...|    5|Dr. Eric Goldberg...|review|QnhQ8G51XbUpVEyWY...|[0,0,0]|
|vcNAWiLM4dR7D2nww...|2014-12-12|QzjRXUNSGk3PySEcg...|    5|I love Dr. Goldbe...|review|GP-h9colXgkT79BW7...|[0,0,0]|
|HZdLhv6COCleJMo7n...|2013-06-10|fBQ69-NU9ZyTjjS7T...|    5|THANK YOU ROB! i ...|review|JbAeIYc89Sk8SWmrB...|[7,3,7]|
|mVHrayjG3uZ_RLHkL...|2013-03-15|jVVv_DA5mCDB6medi...|    5|Can't miss stop f...|review|m1FpV3EAeggaAdfPx...|[0,0,0]|
|KayYbHCt-RkbGcPdG...|2014-02-16|0klMyorClST8NYGJq...|    5|Grew up near here...|review|h-A_xNeB_xSbc0psq...|[0,0,0]|
|fNGIbpazjTRdXgwRY...|2014-03-21|f5WKGxGq-XTHJXPXh...|    5|If you are search...|review|aOHQ9MlorpvL71Y6q...|[0,0,0]|
```

```
|    4|I've been informe...|
|    5|If you are search...|
|    4|Rocky's has been ...|
+-----+--------------------+
only showing top 20 rows
```

Data Frames: groupBy

```
> reviews.groupBy('business_id').count().show()
```

▸ (1) Spark Jobs

```
+--------------------+-----+
|         business_id|count|
+--------------------+-----+
|FsY-8nYOCXyj9FoVx...|    3|
|SzHTdZR3yY1WBYUxk...|    2|
|3zgswf_NfBJpeAoWe...|   13|
|SXvbOMPd7jNgTkY6p...|    1|
|Lvk4P_Npmueqs-n1h...|    3|
|oQJ4try-o-181bhsX...|    1|
|vA8ed8BFvQxz4HFt8...|    5|
|FqgotmZY0WcNjyDJh...|    2|
|gZJmtLYGNLoAFU82X...|    1|
|dLwTMpf63CxWGFRd8...|    2|
```

Data Frames => key-value

```
> def star_mapper(review):
    return [(review.stars, review.text)]
  reviews.flatMap(star_mapper).take(10)
```

▸ (1) Spark Jobs

```
Out[25]:
[(5,
  u"dr. goldberg offers everything i look for in a general practitioner.  he's nice anc
s patients; he's affiliated with a top-notch hospital (nyu) which my parents have expla
and you can get referrals to see specialists without having to see him first.  really,
have about him, but i'm really drawing a blank."),
 (5,
  u"Top notch doctor in a top notch practice. Can't say I am surprised when I was refer
one of the best medical schools in the country. \nIt is really easy to get an appointme
 (5,
  u'Dr. Eric Goldberg is a fantastic doctor who has correctly diagnosed every issue tha
ry accessible and we have been able to schedule appointments with him and his staff ver
being his patients for many years to come.'),
 (1,
```

DF => map => group => reduce

```
> def biz_mapper(review):
      return [(review.business_id, 1)]
  counts = reviews.flatMap(biz_mapper).groupByKey().mapValues(sum)
  counts.take(10)
```

▸ (1) Spark Jobs

```
Out[27]:
[(u'OlpyplEJ_c_hFxyand_Wxw', 14),
 (u'sOcZcXcNm8LmdoOYqEDqpg', 3),
 (u'FFCkoA_L3cqYXtHtLyvxwA', 24),
 (u'A-y20kJLEs-FE7g_idjbPw', 5),
 (u'FFdlPSZCGgTdg1CAfrlvlw', 6),
 (u'pKp50rYh0iWZYKiWmWmLow', 2),
 (u'MJtnKhA3l-2ZFzhneuSccw', 4),
 (u'SNpVV5viJ2aPylP6bkAx8Q', 10),
 (u'fx4coO0OyW7Qe8vdLnlLiA', 2),
 (u'j13Aby6-9ZyklpG_W7qsew', 2)]
Command took 5.54s
```

```
> counts.top(10, key=lambda x: x[1])
```

▸ (1) Spark Jobs

```
Out[28]:
[(u'4bEjOyTaDG24SY5TxsaUNQ', 823),
 (u'zt1TpTuJ6y9n551sw9TaEg', 685),
 (u'2e2e7WgqU1BnpxmQL5jbfw', 648),
 (u'sIyHTizqAiGu12XMLX3N3g', 524),
 (u'Xhg93cMdemu5pAMkDoEdtQ', 521),
```

Data Frame Operations

- sqlContext.read.json(path=…)
- count(), distinct(), first()
- select(*cols), drop(col),
- flatMap(fun), map(fun)
- filter(condition), where(condition)
- groupBy(*cols)
- intersect(other), join(other)
- orderBy(*cols), sort(*cols)
- sample()
- stat, take, show
- https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame

Summary

- Performance is both about algorithmic "complexity" and implementation constants
  - Make the common case fast
- Parallelism
  - Often the parallel work scales with the data
- Master – Worker Model of Parallel data processing on clusters (in the cloud)
- RDDs of values, key-value
- Data Frame / SQL (like Tables)
- New concepts: flatMap, groupBy

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3044375856741396/398677364991930/1602914200610255/latest.html