Lecture 12:

Cluster Computing

(MapReduce and Spark)

Parallel Computing
Stanford CS149, Winter 2019

Which program performs better?

Program 1

```
void add(int n, float* A, float* B, float* C) {
    for (int i=0; i<n; i++)
       C[i] = A[i] + B[i];
}

void mul(int n, float* A, float* B, float* C) {
    for (int i=0; i<n; i++)
       C[i] = A[i] * B[i];
}


float* A, *B, *C, *D, *E, *tmp1, *tmp2;

// assume arrays are allocated here

// compute E = D + ((A + B) * C)
add(n, A, B, tmp1);
mul(n, tmp1, C, tmp2);
add(n, tmp2, D, E);
```

Two loads, one store per math op (arithmetic intensity = 1/3)

Two loads, one store per math op (arithmetic intensity = 1/3)

Overall arithmetic intensity = 1/3

Program 2

```
void fused(int n, float* A, float* B, float* C, float* D, float* E) {
    for (int i=0; i<n; i++)
       E[i] = D[i] + (A[i] + B[i]) * C[i];
}

// compute E = D + (A + B) * C
fused(n, A, B, C, D, E);
```

Four loads, one store per 3 math ops (arithmetic intensity = 3/5)

The transformation of the code in program 1 to the code in program 2 is called "loop fusion"

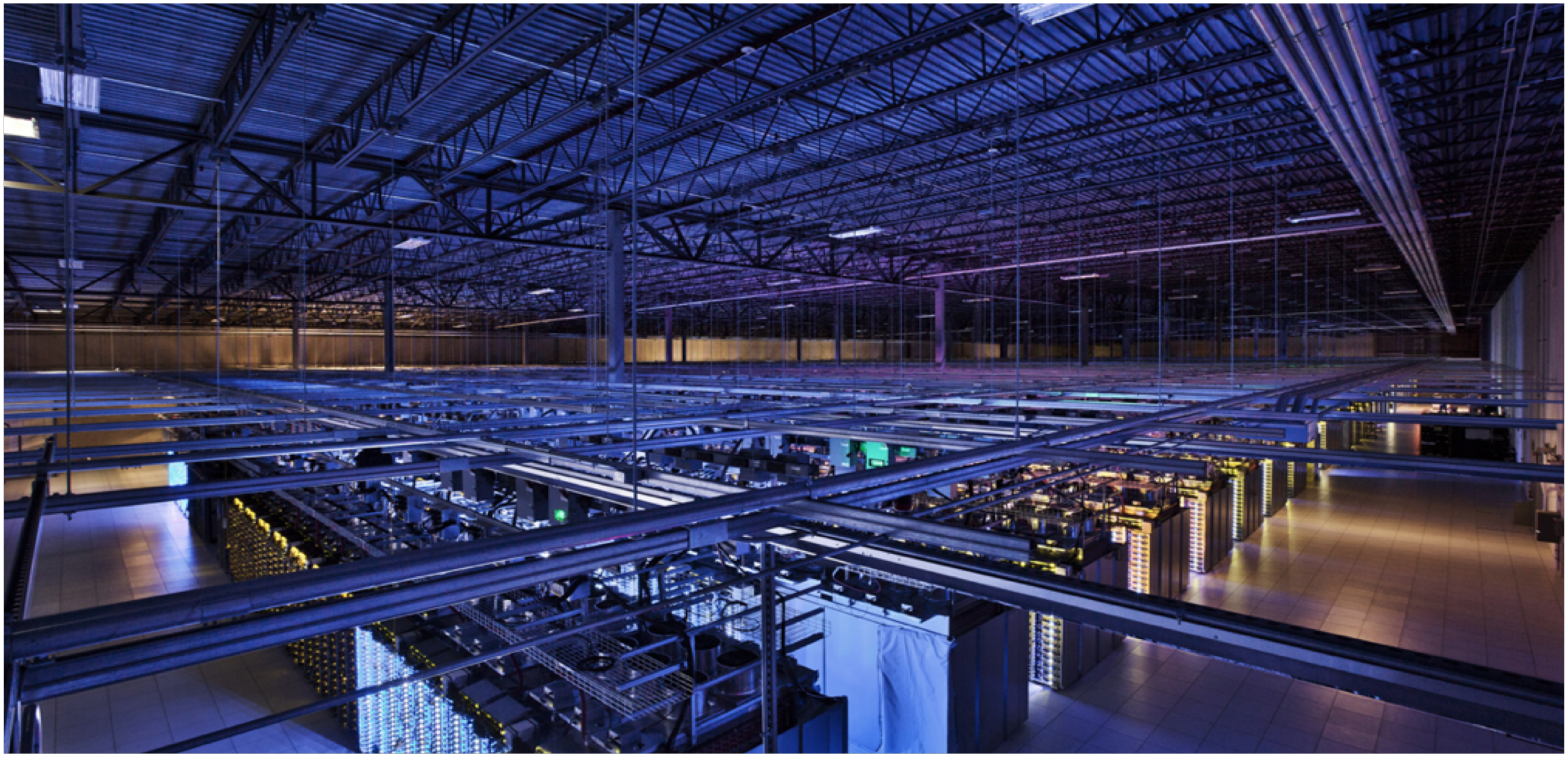
Example involves globally restructuring the order of computation to improve producer-consumer locality

(improve arithmetic intensity of program)

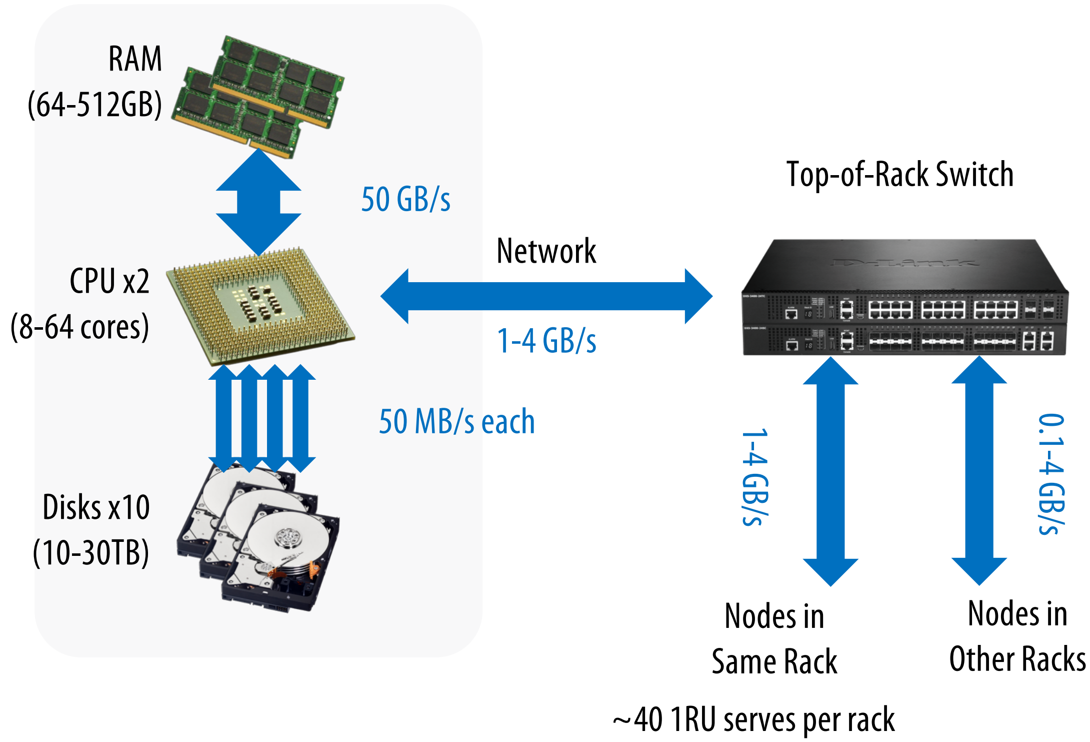
Commodity Clusters

- Standard architecture:
  - Cluster of commodity Linux servers (multicore x86)
  - Private memory ⇒ separate address spaces
  - Ethernet network ⇒ (relatively) low bandwidth
- Inexpensive
  - Built from commodity processors, networks & storage
  - 1000s of nodes for $<$ \$10M
- How to organize computations on this architecture?
  - Mask issues such as load balancing and failures

Warehouse Size Cluster



Typical Commodity Server



Using A Cluster

- Want to process 100TB (1 day Facebook data, $4000 to store!)
- On 1 node: scanning @ 50MB/s = 23 days
- On 1000 nodes: scanning @ 50MB/s = 33 min
- But, very hard to utilize 1000 nodes!
  - Hard to program 8,000 cores
  - Something breaks every hour
  - Need efficient, reliable and usable framework

Cluster Storage Systems

- First order problem: if nodes can fail, how can we store data persistently?

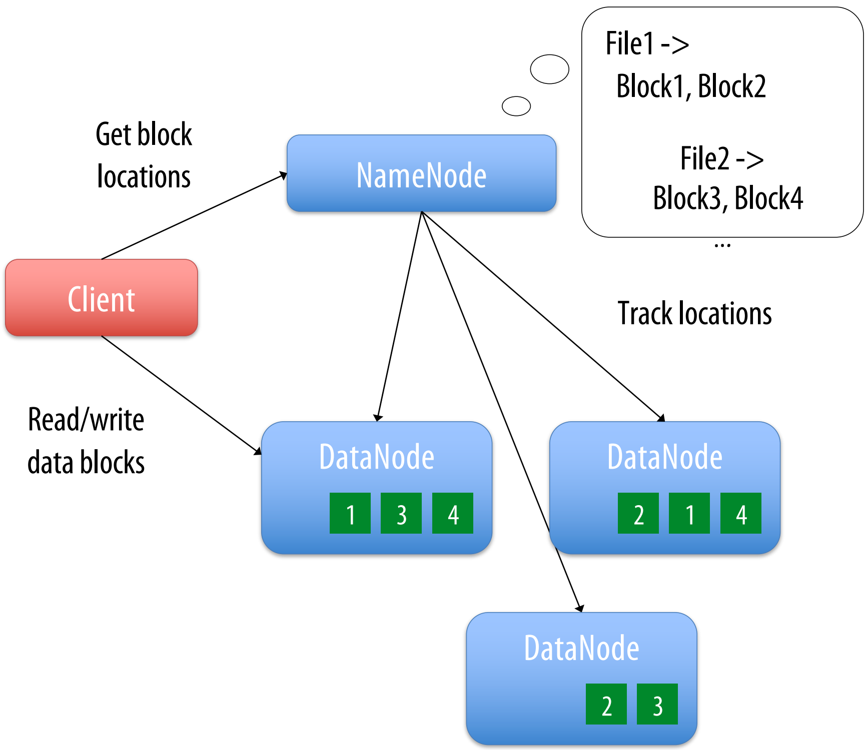
- Answer: distributed storage systems
  - Provide global namespace
  - Examples: Google GFS, Hadoop HDFS, Amazon S3

- Typical usage patterns
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common (e.g. log files)

Distributed File Systems

- **Chunk servers**
  - a.k.a. DataNodes in HDFS
  - File is split into contiguous chunks (usually 64-256 MB)
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks
- **Master node**
  - a.k.a. NameNode in HDFS
  - Stores metadata; might be replicated
- **Client library for file access**
  - Talks to master to find chunk (data) servers
  - Connects directly to chunk servers to access data

Example: Hadoop Distributed FS (HDFS)



- Global namespace
- Files split into ~200MB blocks
- Each block replicated on multiple DataNodes
- Intelligent client
  - Finds locations of blocks from NameNode; requests data from DataNode

Cluster Computing Frameworks

- Want frameworks to automatically handle:
  - Work placement (e.g. data locality)
  - Load balancing
  - Communication
  - Failures & stragglers

- Examples: MapReduce, Dryad, Spark, etc

First Major Example: MapReduce

- **Data type: key-value records**

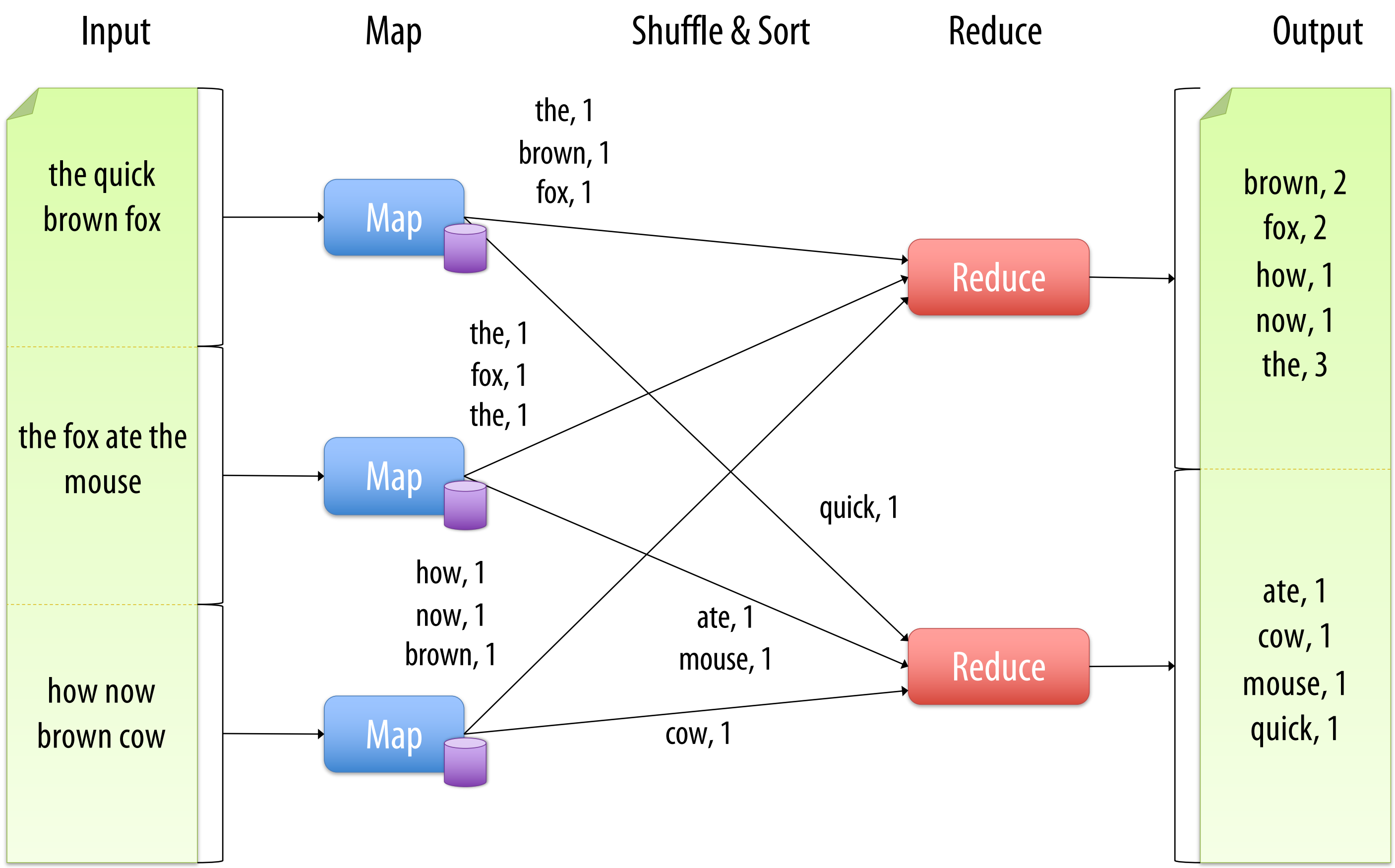
- **Map function:**

$$(K_{in}, V_{in}) \Rightarrow list(K_{inter}, V_{inter})$$

- **Reduce function:**

$$(K_{inter}, list(V_{inter})) \Rightarrow list(K_{out}, V_{out})$$

MapReduce Execution



Benefits

- By providing a functional API, MapReduce greatly simplified cluster programming:
  - Automatic division of job into tasks
  - Locality-aware scheduling
  - Load balancing
  - Recovery from failures & stragglers

A log of page views on the course web site

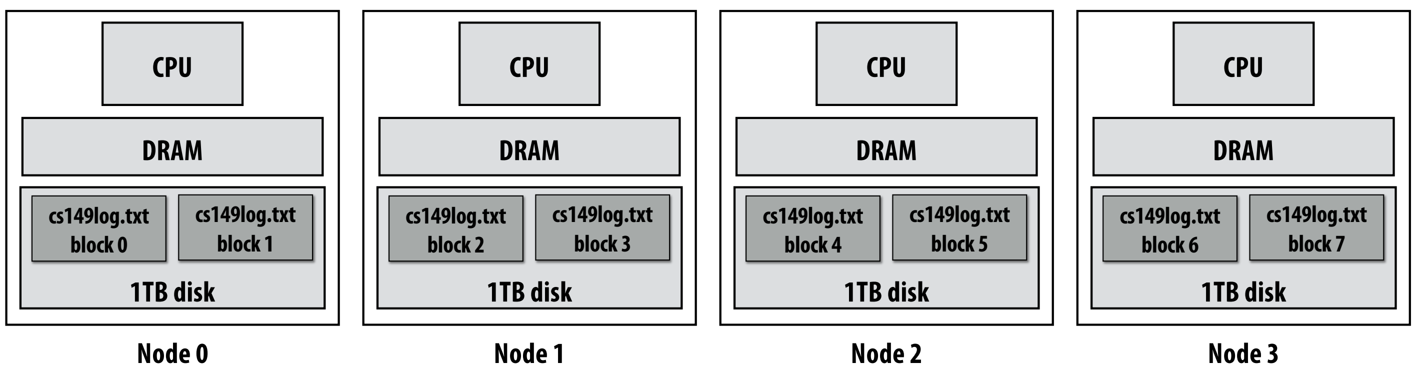
```
171.67.216.21 - - [12/Feb/2019:22:45:03 -0800] "GET /cs248/winter19/keep_alive HTTP/1.0" 200 815 "http://cs248.stanford.edu/winter19/lectures" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:65.0) Gecko/20100101 Firefox/65.0"
171.67.216.21 - - [12/Feb/2019:22:45:03 -0800] "GET /cs248/winter19/keep_alive HTTP/1.0" 200 815 "http://cs248.stanford.edu/winter19/lecture/pipeline" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:65.0) Gecko/20100101 Firefox/65.0"
171.67.216.21 - - [12/Feb/2019:22:45:06 -0800] "GET /cs149/winter19/lecture/whyparallelism/slide_039 HTTP/1.0" 200 3072 "-" "Mozilla/5.0 (compatible; YandexBot/3.0; +http://yandex.com/bots)"
171.67.216.21 - - [12/Feb/2019:22:45:06 -0800] "GET /cs149/winter19/keep_alive HTTP/1.0" 404 2039 "http://35.227.169.186/cs149/winter19/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81 Safari/537.36"
171.67.216.21 - - [12/Feb/2019:22:46:25 -0800] "GET /cs149/winter19/lecture/synchronization/slide_009 HTTP/1.0" 200 2751 "-" "Mozilla/5.0 (compatible; YandexBot/3.0; +http://yandex.com/bots)"
171.67.216.21 - - [12/Feb/2019:22:48:03 -0800] "GET /cs248/winter19/keep_alive HTTP/1.0" 200 295 "http://cs248.stanford.edu/winter19/lecture/texture/slide_017" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:65.0) Gecko/20100101 Firefox/65.0"
171.67.216.21 - - [12/Feb/2019:22:48:05 -0800] "GET /cs248/winter19/lecture/acceleration/slide_005 HTTP/1.0" 200 2440 "-" "Mozilla/5.0 (compatible; YandexBot/3.0; +http://yandex.com/bots)"
171.67.216.21 - - [12/Feb/2019:22:48:08 -0800] "GET /cs248/winter19/keep_alive HTTP/1.0" 200 295 "http://cs248.stanford.edu/winter19/lecture/texture/slide_013" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:65.0) Gecko/20100101 Firefox/65.0"
171.67.216.22 - - [12/Feb/2019:22:50:22 -0800] "GET /cs248/winter19/lecture/acceleration/slide_053 HTTP/1.0" 200 3193 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_052" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.22 - - [12/Feb/2019:22:50:22 -0800] "GET /cs248/winter19content/lectures/09_acceleration/images/slide_053.jpg HTTP/1.0" 200 197443 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_053" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari
537.36"
171.67.216.22 - - [12/Feb/2019:22:50:24 -0800] "GET /cs248/winter19/lecture/acceleration/slide_054 HTTP/1.0" 200 2586 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_053" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.22 - - [12/Feb/2019:22:50:24 -0800] "GET /cs248/winter19content/lectures/09_acceleration/images/slide_054.jpg HTTP/1.0" 200 206565 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_054" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari
537.36"
171.67.216.22 - - [12/Feb/2019:22:50:25 -0800] "GET /cs248/winter19/lecture/acceleration/slide_055 HTTP/1.0" 200 2586 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_054" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.22 - - [12/Feb/2019:22:50:25 -0800] "GET /cs248/winter19content/lectures/09_acceleration/images/slide_055.jpg HTTP/1.0" 200 322211 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_055" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari
537.36"
171.67.216.22 - - [12/Feb/2019:22:50:25 -0800] "GET /cs248/winter19/lecture/acceleration/slide_056 HTTP/1.0" 200 3086 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_055" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.22 - - [12/Feb/2019:22:50:26 -0800] "GET /cs248/winter19content/lectures/09_acceleration/images/slide_056.jpg HTTP/1.0" 200 213676 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_056" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari
537.36"
171.67.216.22 - - [12/Feb/2019:22:50:26 -0800] "GET /cs248/winter19content/profile_pictures/anon.jpg HTTP/1.0" 304 144 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_056" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.22 - - [12/Feb/2019:22:50:26 -0800] "GET /cs248/winter19content/profile_pictures/maq.jpg HTTP/1.0" 404 504 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_056" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.22 - - [12/Feb/2019:22:50:28 -0800] "GET /cs248/winter19/lecture/acceleration/slide_057 HTTP/1.0" 200 2586 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_056" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.22 - - [12/Feb/2019:22:50:29 -0800] "GET /cs248/winter19content/lectures/09_acceleration/images/slide_057.jpg HTTP/1.0" 200 32497 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_057" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/5
37.36"
171.67.216.22 - - [12/Feb/2019:22:50:30 -0800] "GET /cs248/winter19/lecture/acceleration/slide_056 HTTP/1.0" 200 3086 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_057" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.22 - - [12/Feb/2019:22:50:30 -0800] "GET /cs248/winter19content/profile_pictures/maq.jpg HTTP/1.0" 404 504 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_056" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.21 - - [12/Feb/2019:22:50:34 -0800] "GET /cs149/winter19/keep_alive HTTP/1.0" 200 848 "http://cs149.stanford.edu/winter19/home" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:50:54 -0800] "GET /cs149/winter19content/lectures/10_synchronization/images/slide_017.jpg HTTP/1.0" 200 221758 "-" "Googlebot-Image/1.0"
171.67.216.23 - - [12/Feb/2019:22:50:59 -0800] "GET /cs149/winter19content/lectures/09_consistency/images/slide_015.jpg HTTP/1.0" 200 59473 "-" "Googlebot-Image/1.0"
171.67.216.23 - - [12/Feb/2019:22:51:01 -0800] "GET /cs348k/fall18/keep_alive HTTP/1.0" 200 845 "http://cs348k.stanford.edu/fall18/" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.22 - - [12/Feb/2019:22:51:42 -0800] "GET /cs348k/robots.txt HTTP/1.0" 404 475 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"
171.67.216.22 - - [12/Feb/2019:22:51:43 -0800] "GET /cs348k/.well-known/assetlinks.json HTTP/1.0" 404 492 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"
171.67.216.22 - - [12/Feb/2019:22:51:49 -0800] "GET /cs248/winter19/keep_alive HTTP/1.0" 200 903 "http://cs248.stanford.edu/winter19/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36"
171.67.216.21 - - [12/Feb/2019:22:52:44 -0800] "GET /cs248/winter19/lecture/acceleration/slide_057 HTTP/1.0" 200 2586 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_056" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.21 - - [12/Feb/2019:22:52:47 -0800] "GET /cs248/winter19/lecture/acceleration/slide_058 HTTP/1.0" 200 2586 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_057" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.21 - - [12/Feb/2019:22:52:48 -0800] "GET /cs248/winter19content/lectures/09_acceleration/images/slide_058.jpg HTTP/1.0" 200 127709 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_058" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari
537.36"
171.67.216.21 - - [12/Feb/2019:22:53:20 -0800] "GET /cs348k/fall18/keep_alive HTTP/1.0" 200 853 "http://cs348k.stanford.edu/fall18/lecture/dnntrain/slide_002" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81 Safari/537.36"
171.67.216.22 - - [12/Feb/2019:22:55:15 -0800] "GET /cs348k/fall18/keep_alive HTTP/1.0" 200 853 "http://cs348k.stanford.edu/fall18/lectures" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36"
171.67.216.22 - - [12/Feb/2019:22:55:17 -0800] "GET /cs348k/fall18/lecture/camerapipeline2/slide_021 HTTP/1.0" 200 2434 "-" "Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)"
171.67.216.22 - - [12/Feb/2019:22:55:17 -0800] "GET /cs149/winter19/keep_alive HTTP/1.0" 200 817 "http://cs149.stanford.edu/winter19/lecture/perfopt2" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:64.0) Gecko/20100101 Firefox/64.0"
171.67.216.22 - - [12/Feb/2019:22:55:17 -0800] "GET /cs149/winter19/keep_alive HTTP/1.0" 200 817 "http://cs149.stanford.edu/winter19/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:64.0) Gecko/20100101 Firefox/64.0"
171.67.216.23 - - [12/Feb/2019:22:55:48 -0800] "GET /cs149/winter19/ HTTP/1.0" 200 4465 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:55:48 -0800] "GET /cs149/winter19/assets/third_party/jquery/cookie/jquery.cookie.js HTTP/1.0" 200 1150 "http://cs149.stanford.edu/winter19/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:55:48 -0800] "GET /cs149/winter19/assets/third_party/codemirror-3.0/lib/codemirror.css HTTP/1.0" 200 2280 "http://cs149.stanford.edu/winter19/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:55:48 -0800] "GET /cs149/winter19/assets/third_party/jquery/timeago/jquery.timeago.js HTTP/1.0" 200 1987 "http://cs149.stanford.edu/winter19/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:55:48 -0800] "GET /cs149/winter19/assets/third_party/google-code-prettify/prettify.css HTTP/1.0" 200 621 "http://cs149.stanford.edu/winter19/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:55:48 -0800] "GET /cs149/winter19/assets/third_party/jquery/1.8.3/jquery.min.js HTTP/1.0" 200 33750 "http://cs149.stanford.edu/winter19/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:55:48 -0800] "GET /cs149/winter19/assets/third_party/codemirror-3.0/lib/codemirror.js HTTP/1.0" 200 47816 "http://cs149.stanford.edu/winter19/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:55:48 -0800] "GET /cs149/winter19/assets/css/main.css HTTP/1.0" 200 4045 "http://cs149.stanford.edu/winter19/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:55:48 -0800] "GET /cs149/winter19/assets/third_party/codemirror-3.0/mode/markdown/markdown.js HTTP/1.0" 200 3979 "http://cs149.stanford.edu/winter19/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:55:48 -0800] "GET /cs149/winter19/assets/js/comments.js HTTP/1.0" 200 3878 "http://cs149.stanford.edu/winter19/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:55:48 -0800] "GET /cs149/winter19/assets/third_party/google-code-prettify/prettify.js HTTP/1.0" 200 6340 "http://cs149.stanford.edu/winter19/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:55:48 -0800] "GET /cs149/winter19/assets/js/main.js HTTP/1.0" 200 768 "http://cs149.stanford.edu/winter19/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:55:49 -0800] "GET /cs149/winter19/assets/images/favicon/stanford.png HTTP/1.0" 404 507 "http://cs149.stanford.edu/winter19/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.22 - - [12/Feb/2019:22:55:50 -0800] "GET /cs248/winter19/lecture/acceleration/slide_059 HTTP/1.0" 200 3197 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_058" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36"
171.67.216.22 - - [12/Feb/2019:22:55:50 -0800] "GET /cs248/winter19content/lectures/09_acceleration/images/slide_059.jpg HTTP/1.0" 200 120026 "http://cs248.stanford.edu/winter19/lecture/acceleration/slide_059" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari
537.36"
171.67.216.23 - - [12/Feb/2019:22:56:16 -0800] "GET /cs149/ HTTP/1.0" 302 524 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:56:16 -0800] "GET /cs149/winter19/ HTTP/1.0" 200 4608 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36"
171.67.216.23 - - [12/Feb/2019:22:56:16 -0800] "GET /cs149/winter19/assets/third_party/jquery/1.8.3/jquery.min.js HTTP/1.0" 200 33750 "http://cs149.stanford.edu/winter19/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36"
```

Let’s say CS149 gets very popular…

The log of page views gets quite large…

Assume cs149log.txt is a large file, stored in a distributed file system, like HDFS

Below: cluster of 4 nodes, each node with a 1 TB disk
Contents of cs149log.txt are distributed evenly in blocks across the cluster



Imagine your professors want to know a bit more about the glut of students visiting the CS149 web site…

For example:
"What type of mobile phone are all these students using?"

Consider a simple programming model

```
// called once per line in input file by runtime
// input: string (line of input file)
// output: adds (user_agent, 1) entry to list
void mapper(string line, multimap<string,string>& results) {
   string user_agent = parse_requester_user_agent(line);
   if (is_mobile_client(user_agent))
     results.add(user_agent, 1);
}

// called once per unique key (user_agent) in results
// values is a list of values associated with the given key
void reducer(string key, list<string> values, int& result) {
    int sum = 0;
    for (v in values)
       sum += v;
    result = sum;
}


// iterator over lines of text file
LineByLineReader input(“hdfs://cs149log.txt”);

// stores output
Writer output(“hdfs://…”);

// do stuff
runMapReduceJob(mapper, reducer, input, output);
```

(The code above computes the count of page views by each type of mobile phone.)

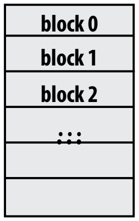
Let’s design an implementation of runMapReduceJob

Step 1: running the mapper function

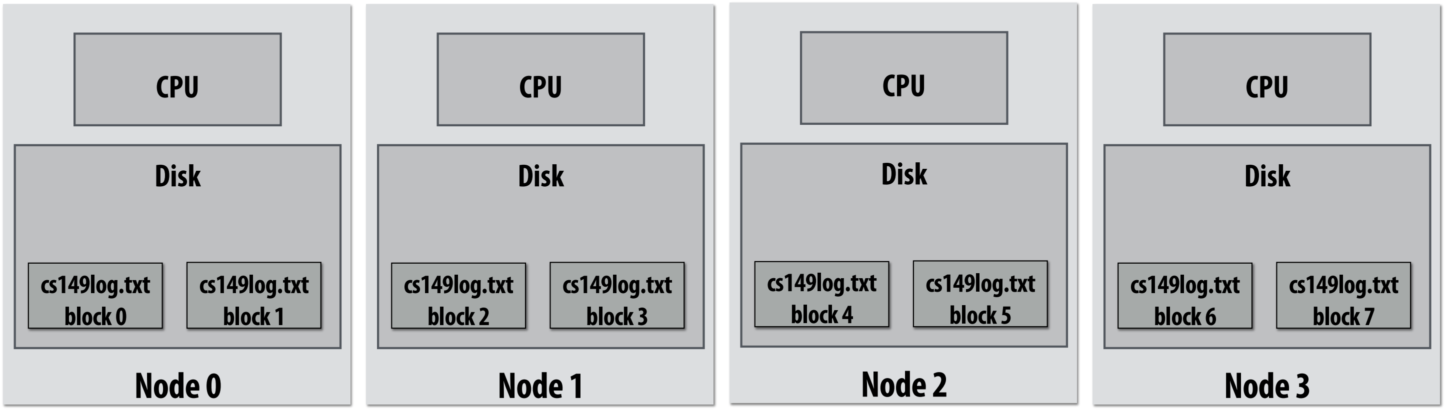
```
// called once per line in file
void mapper(string line, multimap<string,string>& results) {
   string user_agent = parse_requester_user_agent(line);
   if (is_mobile_client(user_agent))
     results.add(user_agent, 1);
}

// called once per unique key in results
void reducer(string key, list<string> values, int& result) {
    int sum = 0;
    for (v in values)
       sum += v;
    result = sum;
}

LineByLineReader input("hdfs://cs149log.txt");
Writer output("hdfs://…");
runMapReduceJob(mapper, reducer, input, output);
```

Step 1: run mapper function on all lines of file
Question: How to assign work to nodes?

Idea 1: use work queue for list of input blocks to process
Dynamic assignment: free node takes next available block



Idea 2: data distribution based assignment: Each node processes lines in blocks of input file that are stored locally.



Steps 2 and 3: gathering data, running the reducer

```
// called once per line in file
void mapper(string line, map<string,string> results) {
   string user_agent = parse_requester_user_agent(line);
   if (is_mobile_client(user_agent))
     results.add(user_agent, 1);
}

// called once per unique key in results
void reducer(string key, list<string> values, int& result) {
    int sum = 0;
    for (v in values)
       sum += v;
    result = sum;
}

LineByLineReader input("hdfs://cs149log.txt");
Writer output("hdfs://…");
runMapReduceJob(mapper, reducer, input, output);
```

Step 2: Prepare intermediate data for reducer

Step 3: Run reducer function on all keys

Question: how to assign reducer tasks?

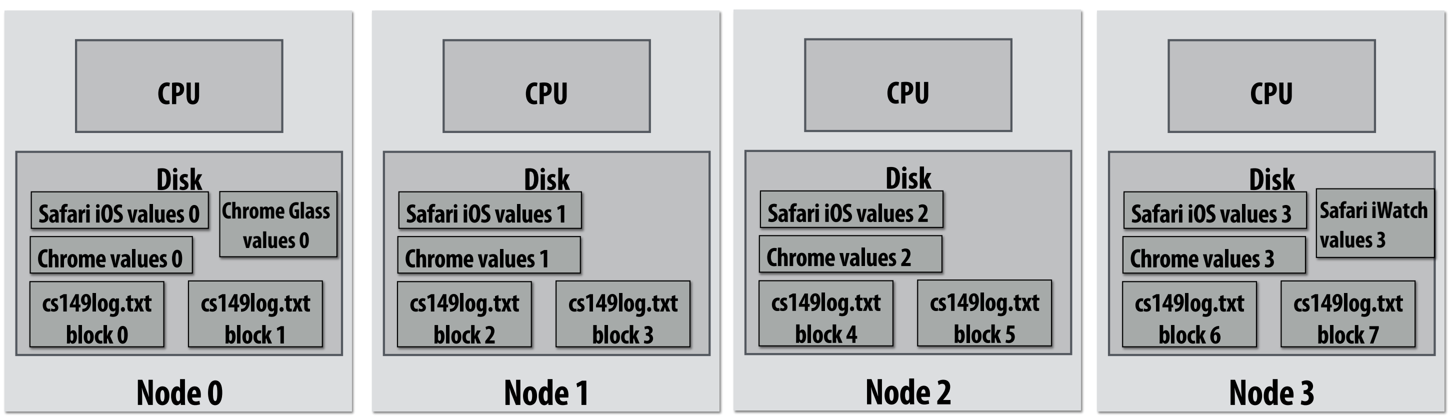
Question: how to get all data for key onto the correct worker node?

R reducer tasks

task = hash(key) mod R

Keys to reduce:
(generated by mapper):

Safari iOS
Chrome
Safari iWatch
Chrome Glass
…



Steps 2 and 3: gathering data, running the reducer

```
// gather all input data for key, then execute reducer
// to produce final result
void runReducer(string key, reducer, result) {
   list<string> inputs;
   for (n in nodes) {
        filename = get_filename(key, n);
        read lines of filename, append into inputs;
   }
   reducer(key, inputs, result);
}
```

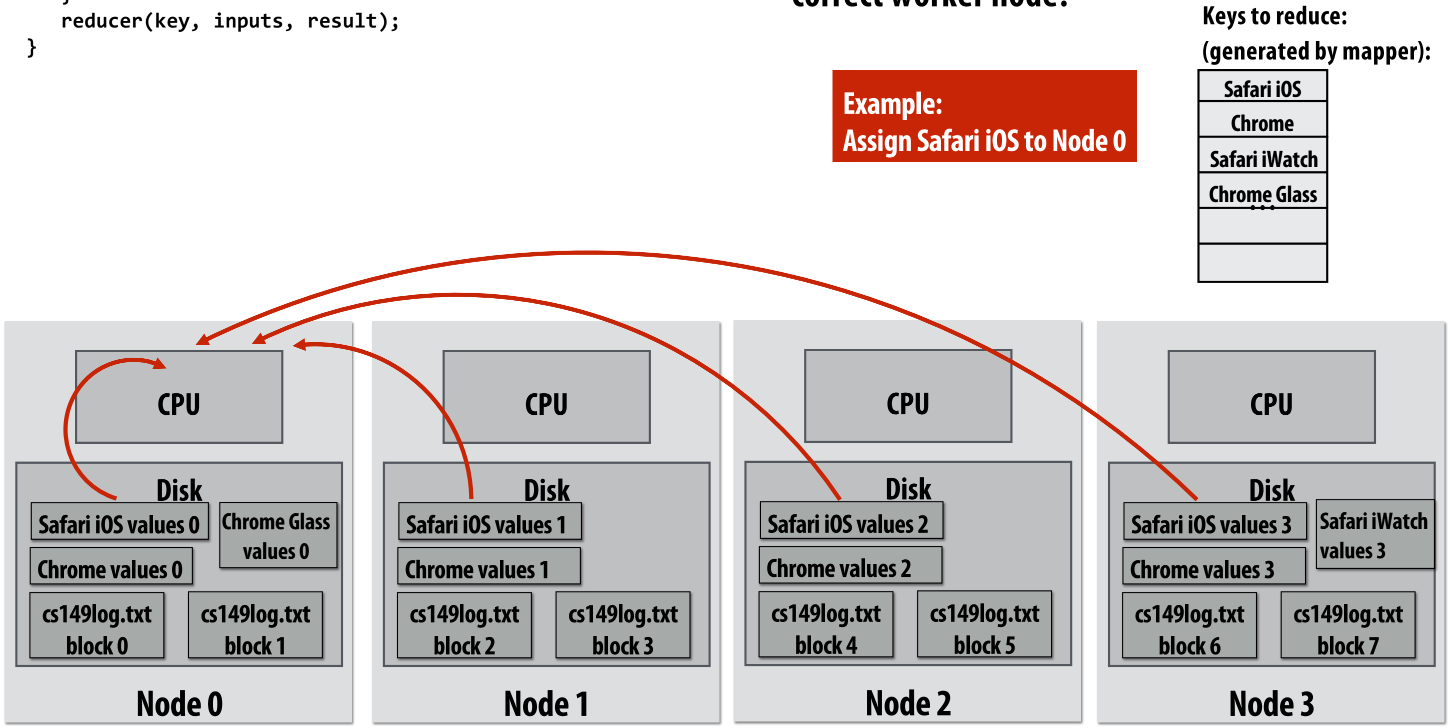
Step 2: Prepare intermediate data for reducer.

Step 3: Run reducer function on all keys.

Question: how to assign reducer tasks?

Question: how to get all data for key onto the correct worker node?

Example:
Assign Safari iOS to Node 0

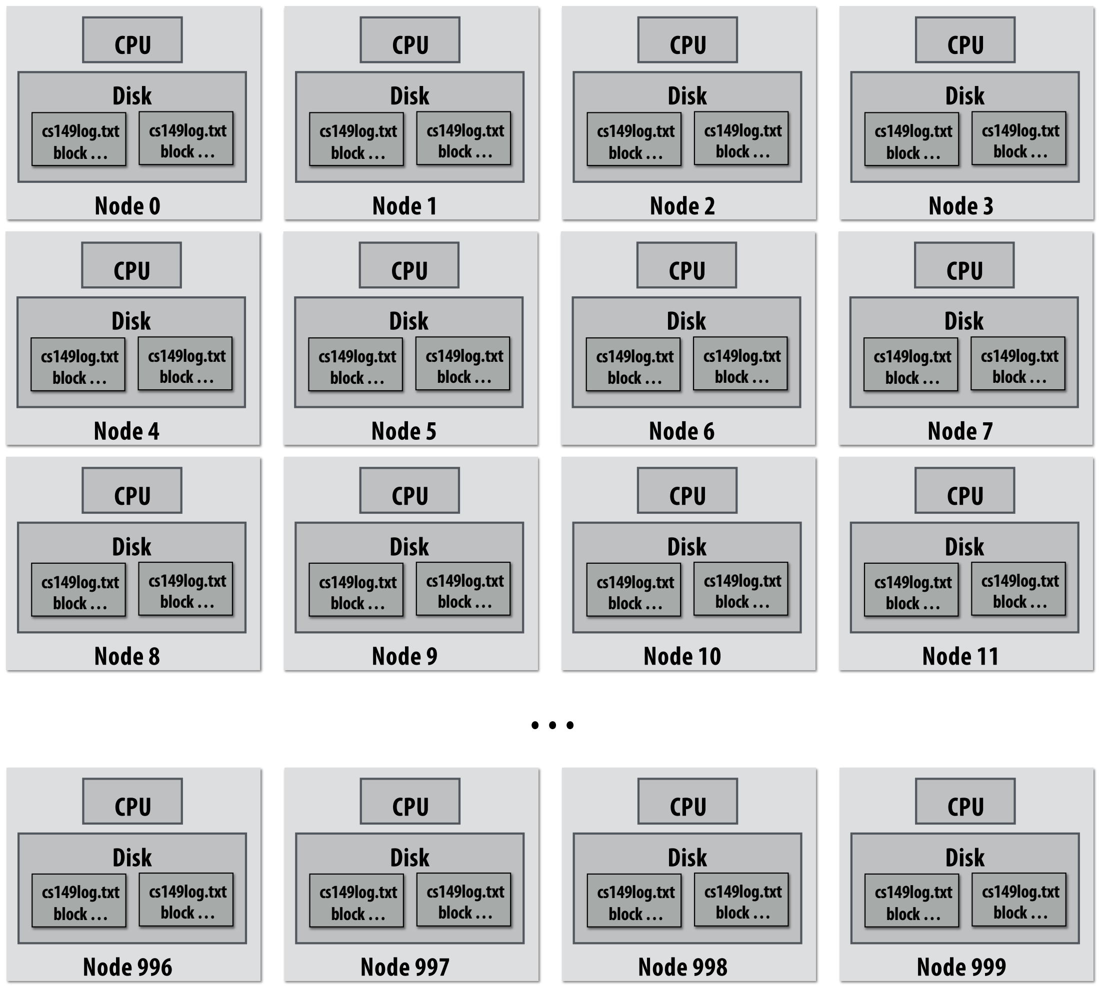


MapReduce Execution Summary

- **map() reduce()**

1. **Partition input key/value pairs into chunks, run map() tasks in parallel**
2. **After all map()s are complete, consolidate all emitted values for each unique emitted key**
3. **Now partition space of output map keys, and run reduce() in parallel**

Additional implementation challenges at scale



Nodes may fail during program execution

Some nodes may run slower than others
(due to different amounts of work, heterogeneity in the cluster, etc..)

Job scheduler responsibilities

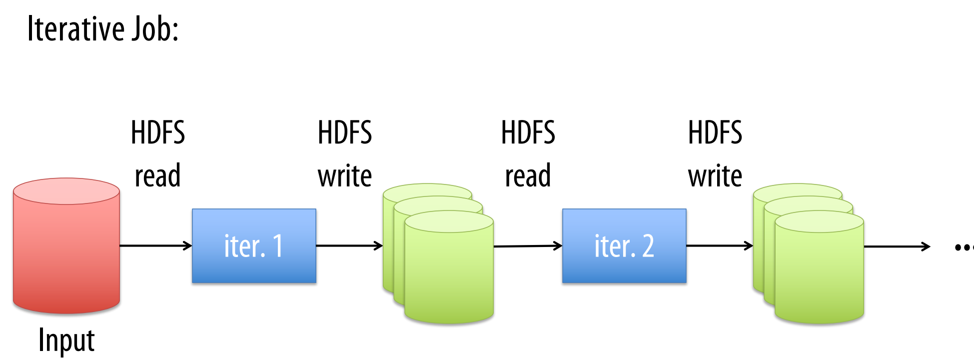
- **Exploit data locality: "move computation to the data"**
  - Run mapper jobs on nodes that contain input files
  - Run reducer jobs on nodes that already have most of data for a certain key
- **Handling node failures**
  - Scheduler detects job failures and reruns job on new machines
    - This is possible since inputs reside in persistent storage (distributed file system)
  - Scheduler duplicates jobs on multiple machines (reduce overall processing latency incurred by node failures)
- **Handling slow machines**
  - Scheduler duplicates jobs on multiple machines

runMapReduceJob problems?

- Permits only a very simple program structure
  - Programs must be structured as: map, followed by reduce by key
  - See DryadLINQ for generalization to DAGs
- Iterative algorithms must load from disk each iteration
  - Example of graph processing (page rank):

```
void pagerank_mapper(graphnode n, map<string,string> results) {
   float val = compute update value for n
   for (dst in outgoing links from n)
     results.add(dst.node, val);
}

void pagerank_reducer(graphnode n, list<float> values, float& result) {
    float sum = 0.0;
    for (v in values)
       sum += v;
    result = sum;
}

for (i = 0 to NUM_ITERATIONS) {
   input = load graph from last iteration
   output = file for this iteration output
   runMapReduceJob(pagerank_mapper, pagerank_reducer, result[i-1], result[i]);
}
```

Problems with MapReduce



1. **Performance**
   - No primitives for data sharing between jobs!
   - Need to go to distributed file system each read/write
2. **Programmability**
   - 10s of map/reduce functions create spaghetti code
   - API does not provide type safety

in-memory, fault-tolerant distributed computing

http://spark.apache.org/

[Zaharia et al. NSDI 2012]

Goals

- Programming model for cluster-scale computations where there is significant reuse of intermediate datasets
  - Iterative machine learning and graph algorithms
  - Interactive data mining: load large dataset into aggregate memory of cluster and then perform multiple ad-hoc queries
- Don't want incur inefficiency of writing intermediates to persistent distributed file system (want to keep it in memory)
  - Challenge: efficiently implementing fault tolerance for large-scale distributed in-memory computations.

Fault tolerance for in-memory calculations

- Replicate all computations
  - Expensive solution: decreases peak throughput
- Checkpoint and rollback
  - Periodically save state of program to persistent storage
  - Restart from last checkpoint on node failure
- Maintain log of updates (commands and data)
  - High overhead for maintaining logs

Recall map-reduce solutions:

- Checkpoints after each map/reduce step by writing results to file system
- Scheduler's list of outstanding (but not yet complete) jobs is a log
- Functional structure of programs allows for restart at granularity of a single mapper or reducer invocation (don't have to restart entire program)

Spark Programming Model

Resilient distributed datasets (RDDs)

- Immutable, partitioned collections of objects
- Created through parallel *transformations* (map, filter, groupBy, join, …) on data in stable storage
- Can be *cached* for efficient reuse

Actions on RDDs

- Count, reduce, collect, save, …
- Generate result on master

Creating RDDs

```
// Turn a Scala collection into an RDD
saprk.parallelize(List(1, 2, 3))

// Load text file from local FS, HDFS, or S3
spark.textFile(“file.txt”)
spark.textFile(“directory/*.txt”)
spark.textFile(“hdfs://namenode:9000/path/file”)
```

Basic Transformations

```
// Pass each element through a function
val squares = nums.map(x => x*x) // {1, 4, 9}

// Keep elements passing a predicate
val even = squares.filter(x => x % 2 == 0) // {4}

// Map each element to zero or more others
nums.flatMap(x => 1 to x) // => {1, 1, 2, 1, 2, 3}
```

Basic RDD Actions

```
val nums = spark.parallelize(List(1, 2, 3))

// Retrieve RDD contents as a local collection
nums.collect() // => Array(1, 2, 3) could be too big!

// Return first K elements
nums.take(2) // => Array(1, 2)

// Count number of elements
nums.count() // => 3

// Merge elements with an associative function
nums.reduce((a, b) => a + b) // => 6

// Write elements to a text file
nums.saveAsTextFile("hdfs://file.txt")
```

Working with Key-Value Pairs

- **Spark's "distributed reduce" transformations operate on RDDs of key-value pairs**

- **Scala pair syntax:**
```
val pair = (a, b)  // sugar for new Tuple2(a, b)
```

- **Accessing pair elements:**
```
pair._1  // => a
pair._2  // => b
```

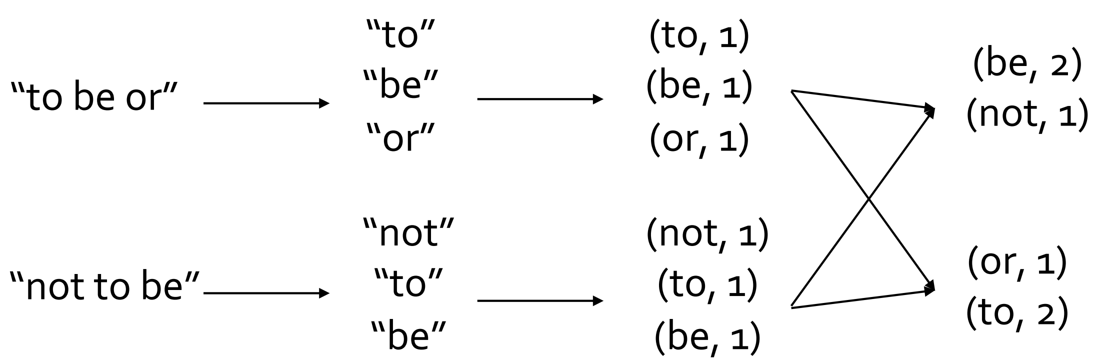
Some Key-Value Operations

```
val pets = sc.parallelize(List((“cat”, 1), (“dog”, 1), (“cat”, 2)))
pets.reduceByKey(_ + _) // => {(cat, 3), (dog, 1)}
pets.groupByKey() // => {(cat, Seq(1, 2)), (dog, Seq(1)}
pets.sortByKey()  // => {(cat, 1), (cat, 2), (dog, 1)}
```

- `reduceByKey` **also locally combines on map side**

Example: Word Count

```
val lines = spark.textFile("hamlet.txt")

val counts = lines.flatMap(line => line.split(" "))
                  .map(word => (word, 1))
                  .reduceByKey(_ + _)
```



Resilient distributed dataset (RDD)

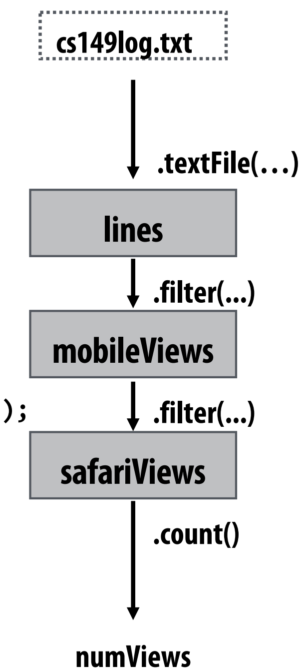
Spark's key programming abstraction:

- Read-only ordered collection of records (immutable)
- RDDs can only be created by deterministic ***transformations*** on data in persistent storage or on existing RDDs
- ***Actions*** on RDDs return data to application

RDDs

```
// create RDD from file system data
var lines = spark.textFile("hdfs://cs149log.txt");

// create RDD using filter() transformation on lines
var mobileViews = lines.filter((x: String) => isMobileClient(x));

// another filter() transformation
var safariViews = mobileViews.filter((x: String) => x.contains("Safari"));

// then count number of elements in RDD via count() action
var numViews = safariViews.count();
```

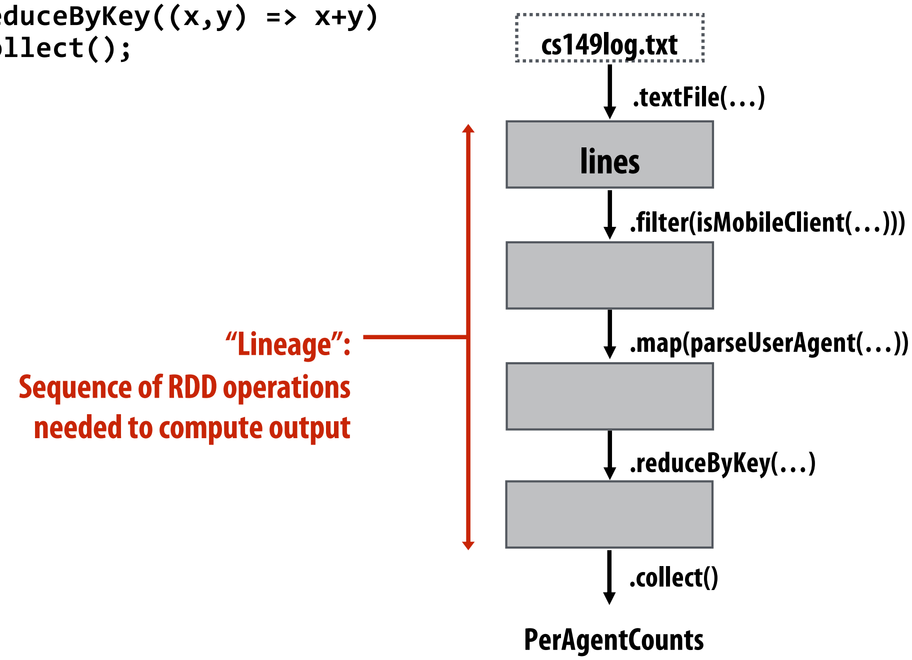
int



Repeating the map-reduce example

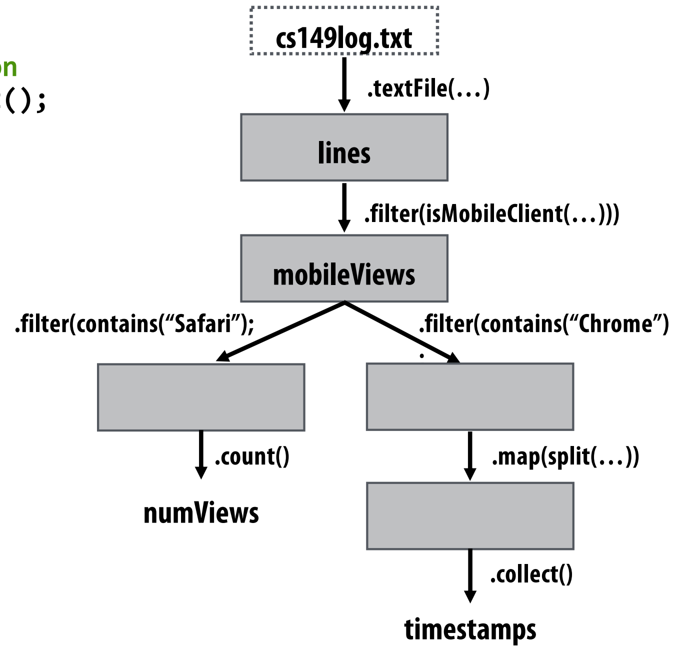
```
// 1. create RDD from file system data
// 2. create RDD with only lines from mobile clients
// 3. create RDD with elements of type (String,Int) from line string
// 4. group elements by key
// 5. call provided reduction function on all keys to count views
var perAgentCounts =  spark.textFile("hdfs://cs149log.txt")
                           .filter(x => isMobileClient(x))
                           .map(x => (parseUserAgent(x),1));
                           .reduceByKey((x,y) => x+y)
                           .collect();
```

Array[String,int]



Another Spark program

```
// create RDD from file system data
var lines = spark.textFile(“hdfs://cs149log.txt”);

// create RDD using filter() transformation on lines
var mobileViews = lines.filter((x: String) => isMobileClient(x));

// instruct Spark runtime to try to keep mobileViews in memory
mobileViews.persist();

// create a new RDD by filtering mobileViews
// then count number of elements in new RDD via count() action
var numViews = mobileViews.filter(_.contains(“Safari”)).count();

// 1. create new RDD by filtering only Chrome views
// 2. for each element, split string and take timestamp of
//    page view
// 3. convert RDD to a scalar sequence (collect() action)
var timestamps = mobileViews.filter(_.contains(“Chrome”))
                            .map(_.split(“ ”)(0))
                            .collect();
```



RDD transformations and actions

Transformations: (data parallel operators taking an input RDD to a new RDD)

$map(f : T \Rightarrow U)$	:	$RDD[T] \Rightarrow RDD[U]$
$filter(f : T \Rightarrow Bool)$	:	$RDD[T] \Rightarrow RDD[T]$
$flatMap(f : T \Rightarrow Seq[U])$	:	$RDD[T] \Rightarrow RDD[U]$
$sample(fraction : Float)$	:	$RDD[T] \Rightarrow RDD[T]$ (Deterministic sampling)
$groupByKey()$	:	$RDD[(K, V)] \Rightarrow RDD[(K, Seq[V])]$
$reduceByKey(f : (V, V) \Rightarrow V)$	:	$RDD[(K, V)] \Rightarrow RDD[(K, V)]$
$union()$	:	$(RDD[T], RDD[T]) \Rightarrow RDD[T]$
$join()$	:	$(RDD[(K, V)], RDD[(K, W)]) \Rightarrow RDD[(K, (V, W))]$
$cogroup()$	:	$(RDD[(K, V)], RDD[(K, W)]) \Rightarrow RDD[(K, (Seq[V], Seq[W]))]$
$crossProduct()$	:	$(RDD[T], RDD[U]) \Rightarrow RDD[(T, U)]$
$mapValues(f : V \Rightarrow W)$	:	$RDD[(K, V)] \Rightarrow RDD[(K, W)]$ (Preserves partitioning)
$sort(c : Comparator[K])$	:	$RDD[(K, V)] \Rightarrow RDD[(K, V)]$
$partitionBy(p : Partitioner[K])$	:	$RDD[(K, V)] \Rightarrow RDD[(K, V)]$

Actions: (provide data back to the "host" application)

$count()$	:	$RDD[T] \Rightarrow Long$
$collect()$	:	$RDD[T] \Rightarrow Seq[T]$
$reduce(f : (T, T) \Rightarrow T)$	:	$RDD[T] \Rightarrow T$
$lookup(k : K)$	:	$RDD[(K, V)] \Rightarrow Seq[V]$ (On hash/range partitioned RDDs)
$save(path : String)$	:	Outputs RDD to a storage system, *e.g.*, HDFS

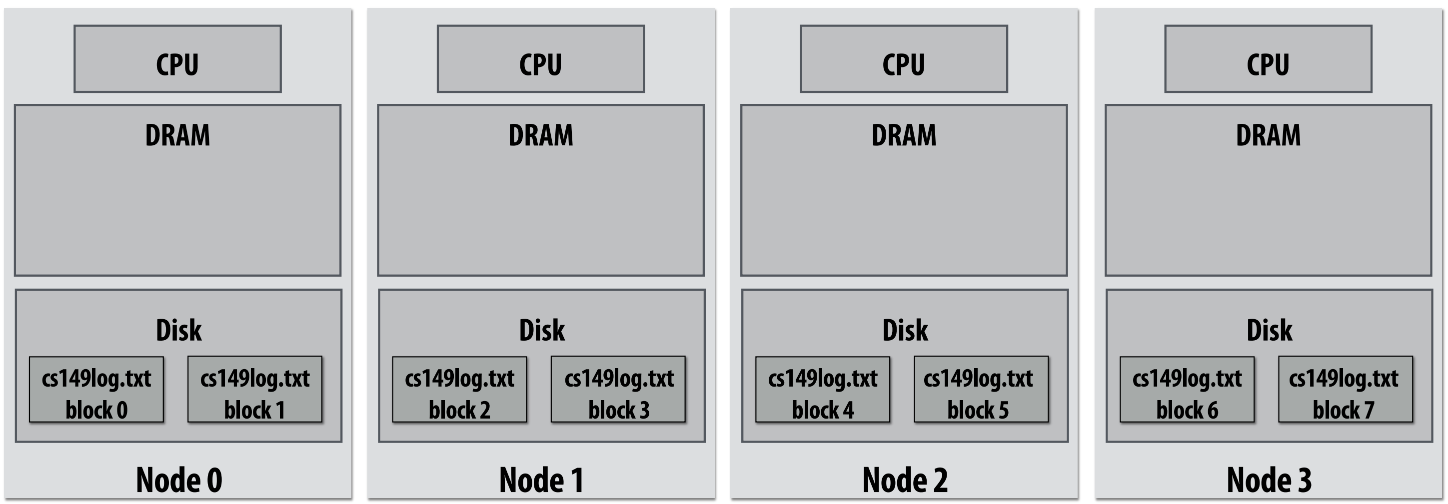
Very similar to Scala Collections API

How do we implement RDDs?

In particular, how should they be stored?

```
var lines = spark.textFile("hdfs://cs149log.txt");
var lower = lines.map(_.toLower());
var mobileViews = lower.filter(x => isMobileClient(x));
var howMany = mobileViews.count();
```

Question: should we think of RDD's like arrays?

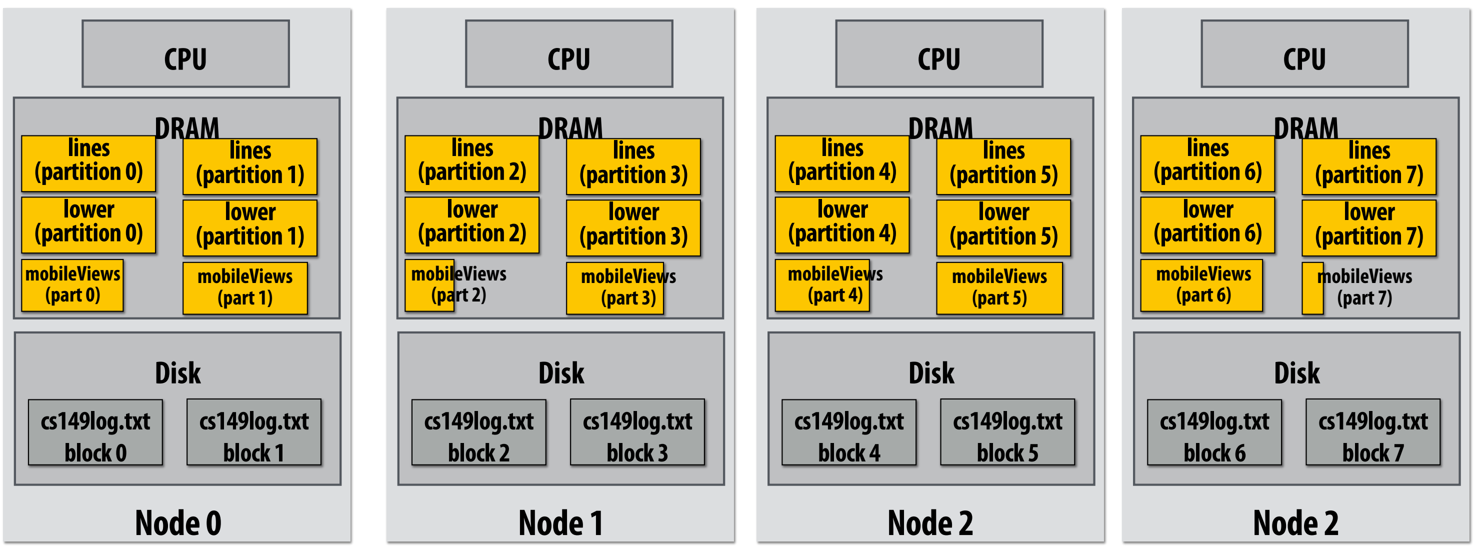


How do we implement RDDs?

In particular, how should they be stored?

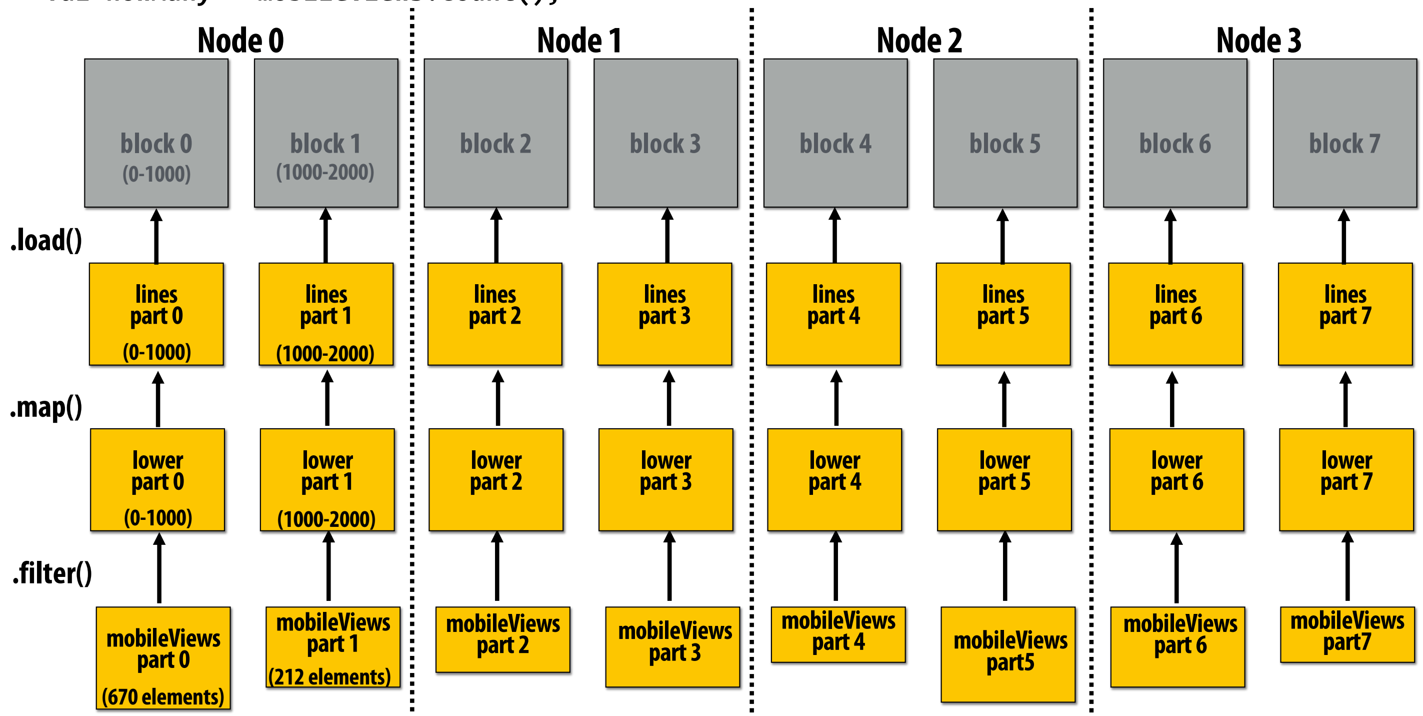
```
val lines = spark.textFile(“hdfs://cs149log.txt”);
val lower = lines.map(_.toLower());
val mobileViews = lower.filter(x => isMobileClient(x));
val howMany = mobileViews.count();
```

In-memory representation would be huge! (larger than original file on disk)



RDD partitioning and dependencies

```
val lines = spark.textFile("hdfs://cs149log.txt");
val lower = lines.map(_.toLower());
val mobileViews = lower.filter(x => isMobileClient(x));
val howMany = mobileViews.count();
```



Black lines show dependencies between RDD partitions.

Implementing sequence of RDD ops efficiently

```
val lines = spark.textFile("hdfs://cs149log.txt");
val lower = lines.map(_.toLower());
val mobileViews = lower.filter(x => isMobileClient(x));
val howMany = mobileViews.count();
```

Recall "loop fusion" examples from opening slides of lecture

The following code stores only a line of the log file in memory, and only reads input data from disk once ("streaming" solution)

```
int count = 0;
while (inputFile.eof()) {
   string line = inputFile.readLine();
   string lower = line.toLower;
   if (isMobileClient(lower))
     count++;
}
```

A simple interface for RDDs

```
val lines = spark.textFile("hdfs://cs149log.txt");
val lower = lines.map(_.toLower());
val mobileViews = lower.filter(x => isMobileClient(x));
val howMany = mobileViews.count();
```

```
// create RDD by mapping map_func onto
input (parent) RDD
RDD::map(RDD parent, map_func) {
   return new RDDFromMap(parent, map_func);
}

// create RDD by filtering input (parent) RDD
RDD::filter(RDD parent, filter_func) {
   return new RDDFromFilter(parent, filter_func);
}

// create RDD from text file on disk
RDD::textFile(string filename) {
   return new RDDFromTextFile(open(filename));
}

// count action (forces evaluation of RDD)
RDD::count() {
   int count = 0;
   while (hasMoreElements()) {
      var el = next();
      count++;
   }
}
```

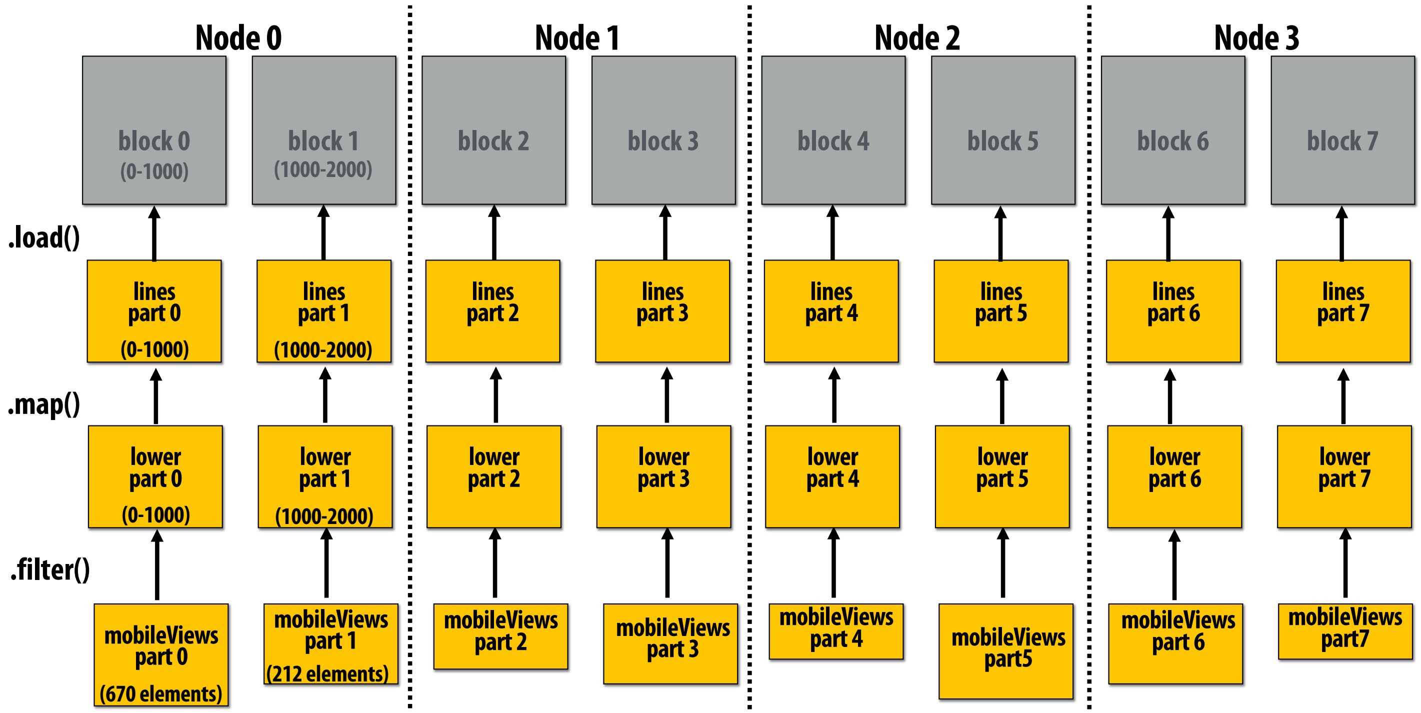
```
RDD::hasMoreElements() {
   parent.hasMoreElements();
}

// overloaded since no parent exists
RDDFromTextFile::hasMoreElements() {
   return !inputFile.eof();
}

RDDFromTextFile::next() {
   return inputFile.readLine();
}

RDDFromMap::next() {
   var el = parent.next();
   return map_func(el);
}

RDDFromFilter::next() {
  while (parent.hasMoreElements()) {
    var el = parent.next();
    if (filter_func(el))
      return el;
}
```

Narrow dependencies

```
val lines = spark.textFile("hdfs://cs149log.txt");
val lower = lines.map(_.toLower());
val mobileViews = lower.filter(x => isMobileClient(x));
val howMany = mobileViews.count();
```

"Narrow dependencies" = each partition of parent RDD referenced by at most one child RDD partition

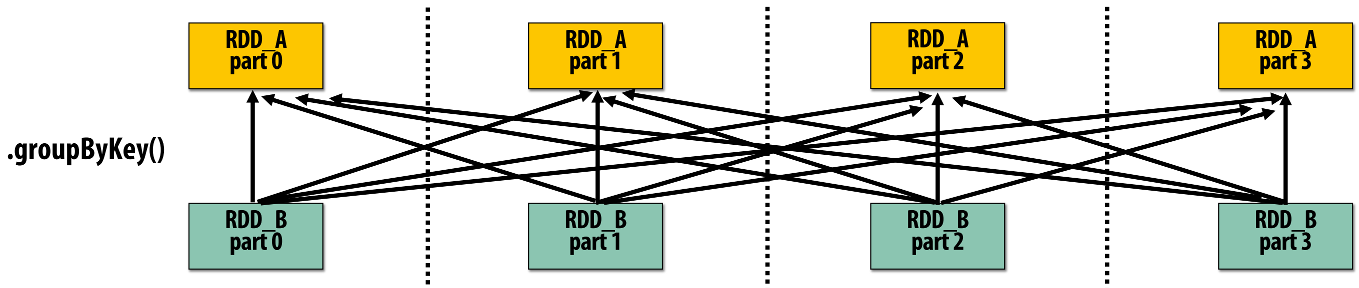
- Allows for fusing of operations (here: can apply map and then filter all at once on input element)
- In this example: no communication between nodes of cluster (communication of one int at end to perform count() reduction)



Wide dependencies

groupByKey: RDD[(K,V)] → RDD[(K,Seq[V])]

"Make a new RDD where each element is a sequence containing all values from the parent RDD with the same key."



Wide dependencies = each partition of parent RDD referenced by multiple child RDD partitions

Challenges:

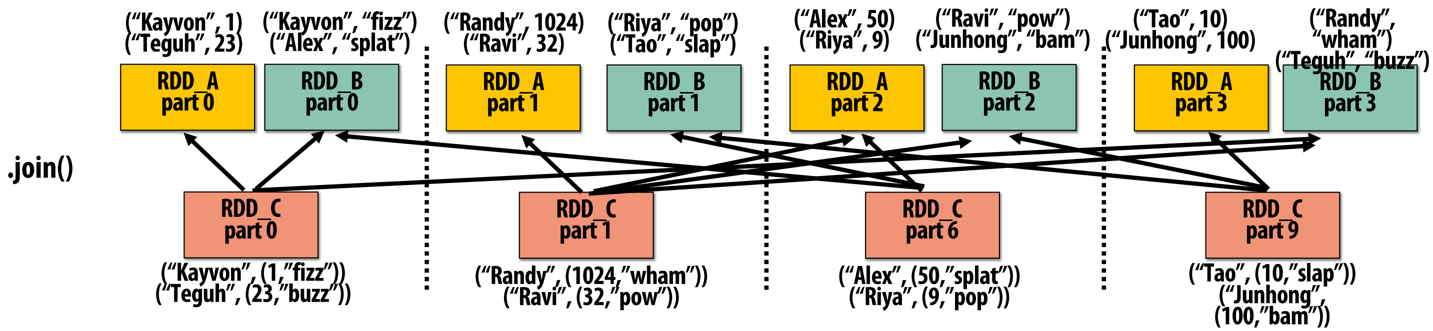
- Must compute all of RDD_A before computing RDD_B
  - Example: groupByKey() may induce all-to-all communication as shown above
- May trigger significant recomputation of ancestor lineage upon node failure
  (I will address resilience in a few slides)

Cost of operations depends on partitioning

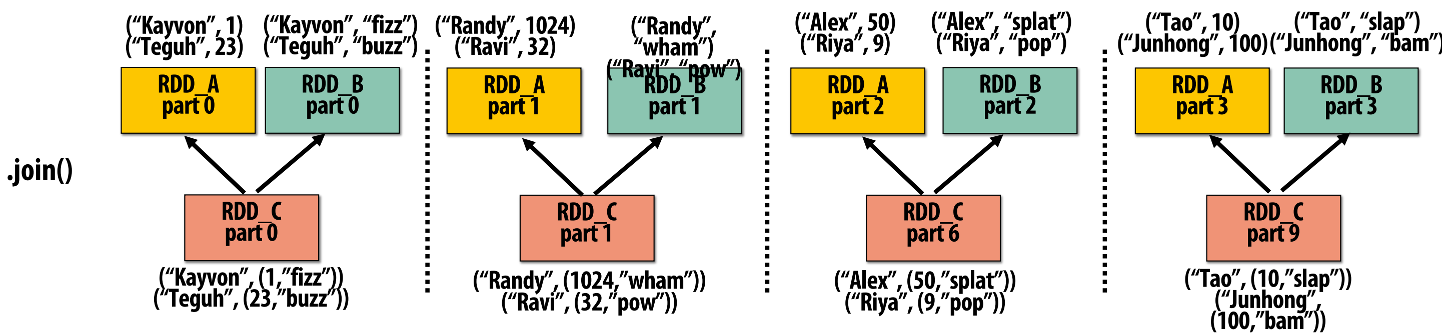
join: RDD[(K,V)], RDD[(K,W)] → RDD[(K,(V,W))]

Assume data in RDD_A and RDD_B are partitioned by key: hash username to partition id

RDD_A and RDD_B have different hash partitions: join creates wide dependencies



RDD_A and RDD_B have same hash partition: join only creates narrow dependencies



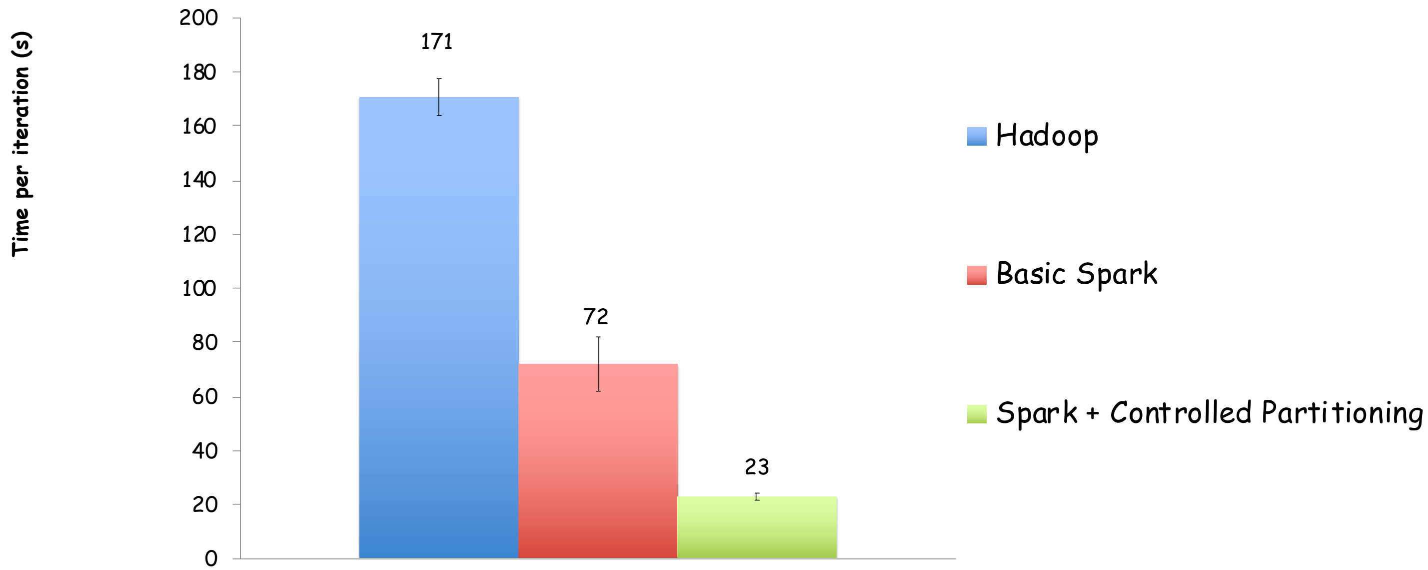
PartitionBy() transformation

- **Inform Spark on how to partition an RDD**
  - e.g., HashPartitioner, RangePartitioner

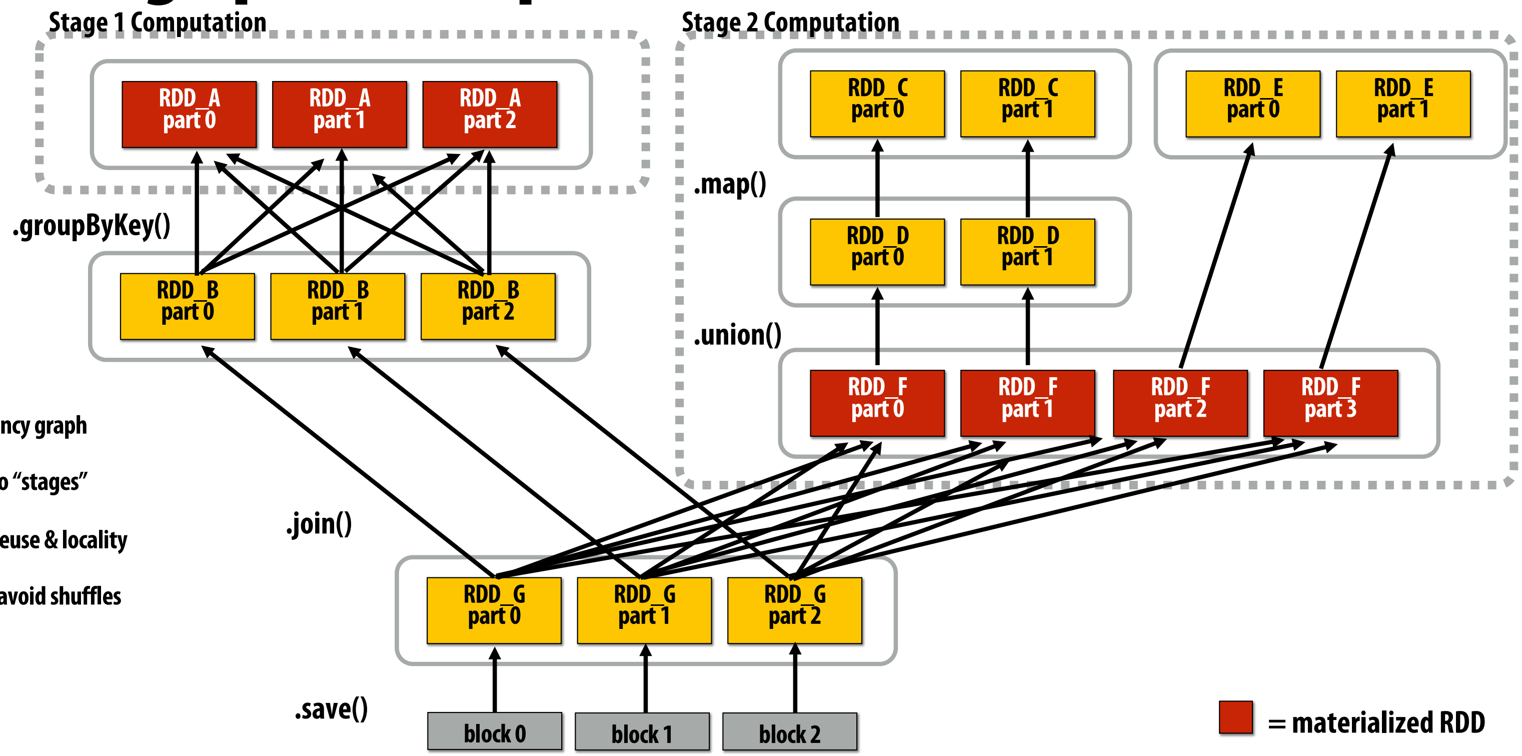
```
// create RDD from file system data
val lines = spark.textFile("hdfs://cs149log.txt");
val clientInfo = spark.textFile("hdfs://clientssupported.txt"); // (useragent, "yes"/"no")

// create RDD using filter() transformation on lines
val mobileViews = lines.filter(x => isMobileClient(x)).map(x => parseUserAgent(x));

// HashPartitioner maps keys to integers
val partitioner = spark.HashPartitioner(100);

// inform Spark of partition
// .persist() also instructs Spark to try to keep dataset in memory
val mobileViewPartitioned = mobileViews.partitionBy(partitioner).persist();
val clientInfoPartitioned = clientInfo.partitionBy(partitioner).persist();

// join useragents with whether they are supported or not supported
// Note: this join only creates narrow dependencies due to the explicit partitioning above
void joined = mobileViewPartitioned.join(clientInfoPartitioned);
```

- **.persist():**
  - Inform Spark this RDD's contents should be retained in memory
  - .persist(RELIABLE) = store contents in durable storage (like a checkpoint)

PageRank Performance



Scheduling Spark computations



Captures RDD dependency graph

Pipelines functions into "stages"

Cache-aware for data reuse & locality

Partitioning-aware to avoid shuffles

Actions (e.g., save()) trigger evaluation of Spark lineage graph.

Stage 1 Computation: do nothing since input already materialized in memory

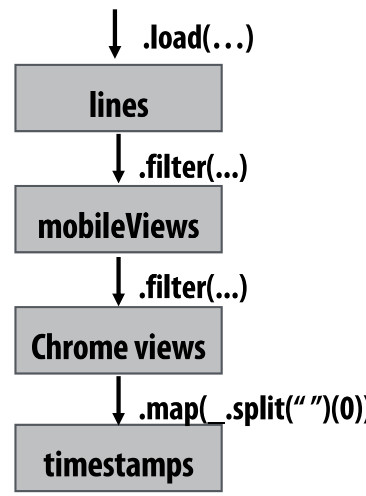
Stage 2 Computation: evaluate map in fused manner, only actually materialize RDD F

Stage 3 Computation: execute join (could stream the operation to disk, do not need to materialize)

Implementing resilience via lineage

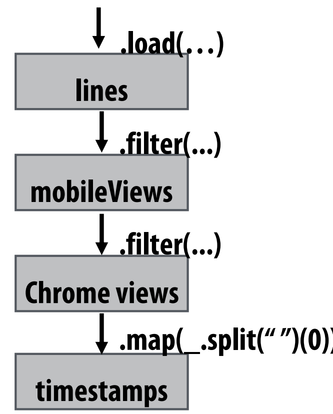
- RDD transformations are bulk, deterministic, and functional
  - Implication: runtime can always reconstruct contents of RDD from its lineage (the sequence of transformations used to create it)
  - Lineage is a log of transformations
  - Efficient: since the log records bulk data-parallel operations, overhead of logging is low (compared to logging fine-grained operations, like in a database)

```
// create RDD from file system data
val lines = spark.textFile("hdfs://cs149log.txt");

// create RDD using filter() transformation on lines
val mobileViews = lines.filter((x: String) => isMobileClient(x));

// 1. create new RDD by filtering only Chrome views
// 2. for each element, split string and take timestamp of
//    page view (first element)
// 3. convert RDD To a scalar sequence (collect() action)
val timestamps = mobileView.filter(_.contains("Chrome"))
                           .map(_.split(" ")(0));
```

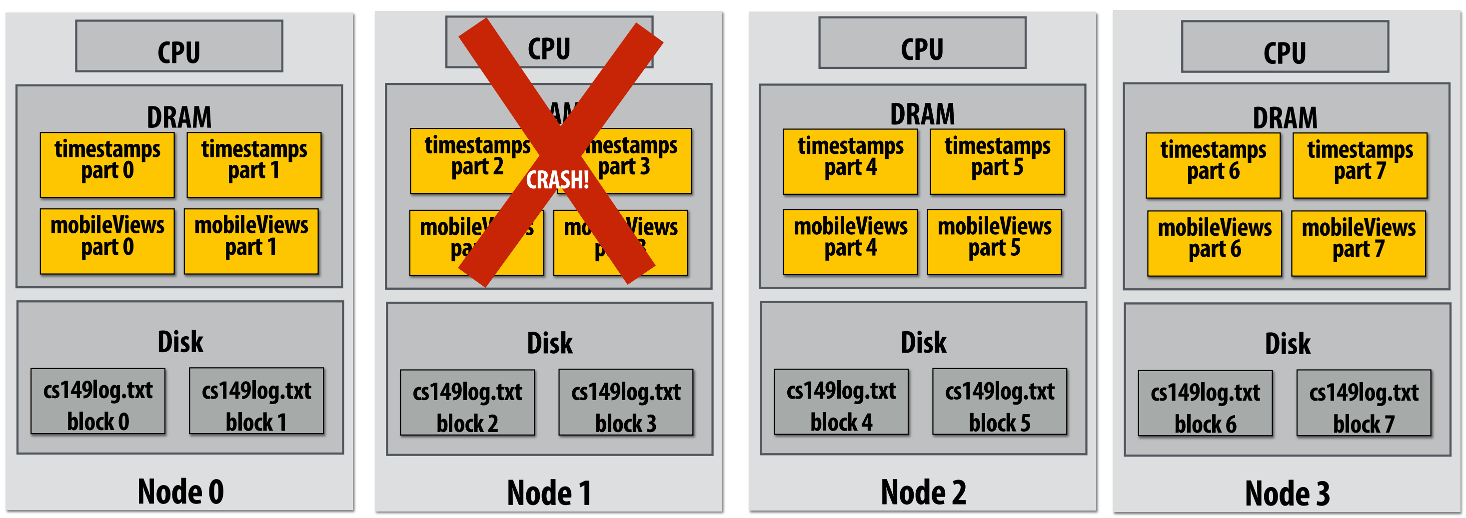


Upon node failure: recompute lost RDD partitions from lineage

```
val lines       = spark.textFile("hdfs://cs149log.txt");
val mobileViews = lines.filter((x: String) => isMobileClient(x));
val timestamps  = mobileView.filter(_.contains("Chrome"))
                            .map(_.split(" ")(0));
```



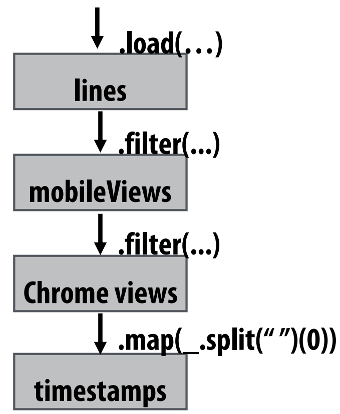
Must reload required subset of data from disk and recompute entire sequence of operations given by lineage to regenerate partitions 2 and 3 of RDD timestamps.



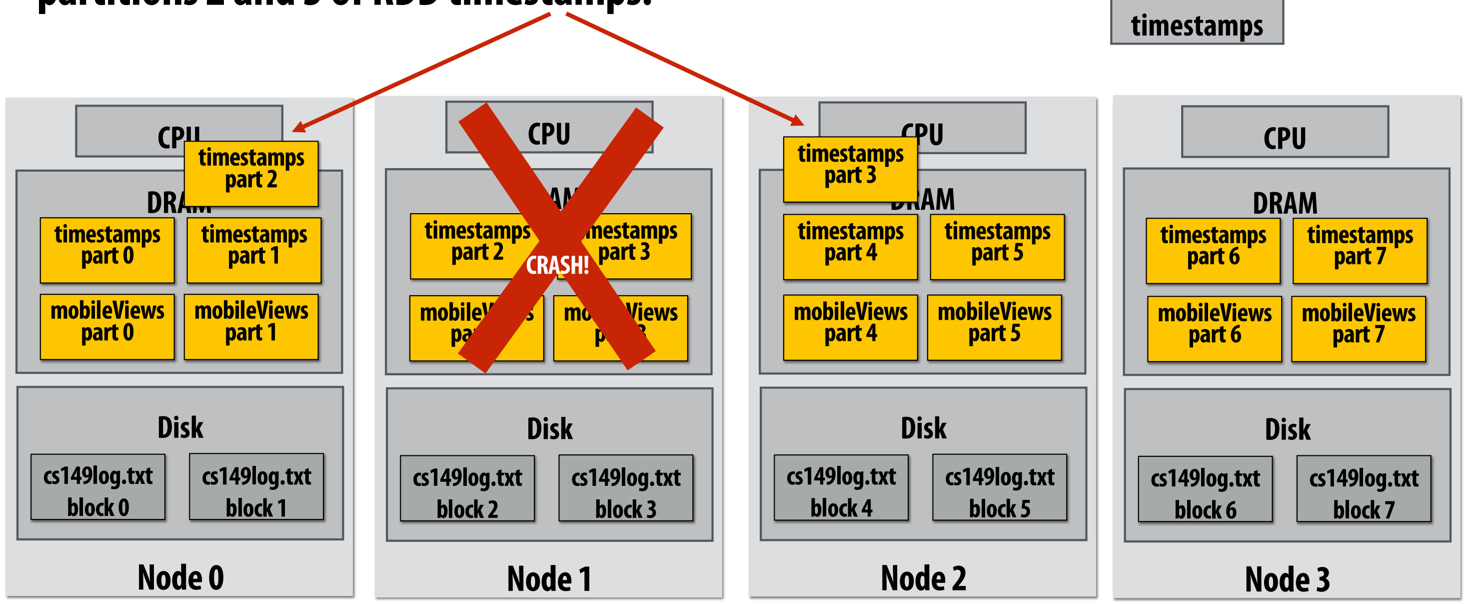
Note: (not shown): file system data is replicated so assume blocks 2 and 3 remain accessible to all nodes

Upon node failure: recompute lost RDD partitions from lineage

```
var lines       = spark.textFile("hdfs://cs149log.txt");
var mobileViews = lines.filter((x: String) => isMobileClient(x));
var timestamps  = mobileView.filter(_.contains("Chrome"))
                             .map(_.split(" ")(0));
```

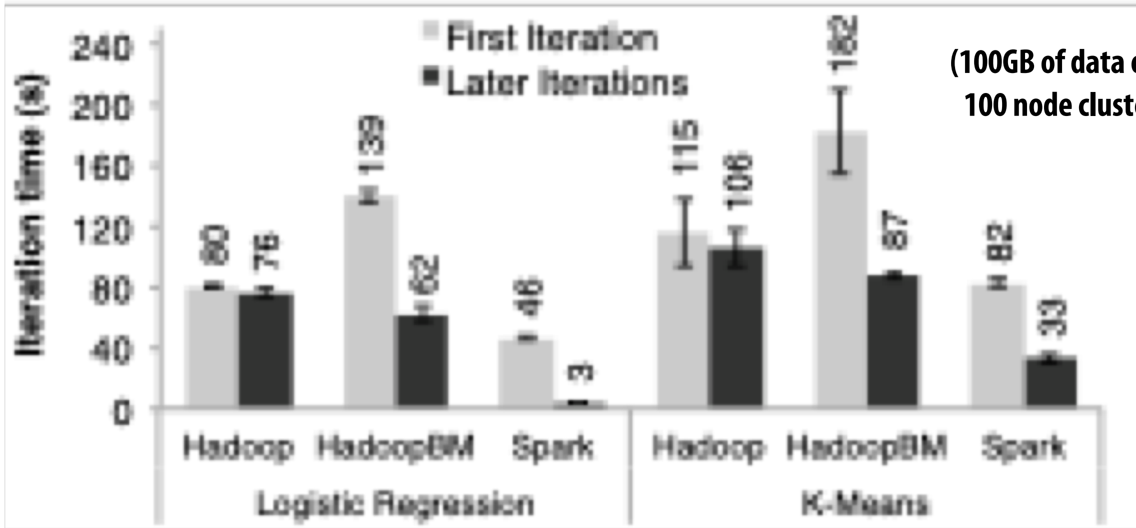


Must reload required subset of data from disk and recompute entire sequence of operations given by lineage to regenerate partitions 2 and 3 of RDD timestamps.



Note: (not shown): file system data is replicated so assume blocks 2 and 3 remain accessible to all nodes

Spark performance



(100GB of data on a 100 node cluster)

HadoopBM = Hadoop Binary In-Memory (convert text input to binary, store in in-memory version of HDFS)

Q. Wait, the baseline parses text input in each iteration of an iterative algorithm? A. Yes.

Anything else puzzling here?

HadoopBM's first iteration is slow because it runs an extra Hadoop job to copy binary form of input data to in memory HDFS

Accessing data from HDFS, even if in memory, has high overhead:

- Multiple mem copies in file system + a checksum
- Conversion from serialized form to Java object

Caution: "scale out" is not the entire story

- Distributed systems designed for cloud execution address many difficult challenges, and have been instrumental in the explosion of "big-data" computing and large-scale analytics
  - Scale-out parallelism to many machines
  - Resiliency in the face of failures
  - Complexity of managing clusters of machines
- But scale out is not the whole story:

20 Iterations of Page Rank

scalable system	cores	twitter	uk-2007-05
GraphChi [10]	2	3160s	6972s
Stratosphere [6]	16	2250s	-
X-Stream [17]	16	1488s	-
Spark [8]	128	857s	1759s
Giraph [8]	128	596s	1235s
GraphLab [8]	128	249s	833s
GraphX [8]	128	419s	462s
Single thread (SSD)	1	300s	651s
Single thread (RAM)	1	275s	-

name	twitter_rv [11]	uk-2007-05 [4]
nodes	41,652,230	105,896,555
edges	1,468,365,182	3,738,733,648
size	5.76GB	14.72GB

Vertex order (SSD)	1	300s	651s
Vertex order (RAM)	1	275s	-
Hilbert order (SSD)	1	242s	256s
Hilbert order (RAM)	1	110s	-

Further optimization of the baseline brought time down to 110s

["Scalability! At what COST?" McSherry et al. HotOS 2015]

Caution: "scale out" is not the entire story

Label Propagation
[McSherry et al. HotOS 2015]

scalable system	cores	twitter	uk-2007-05
Stratosphere [6]	16	950s	-
X-Stream [17]	16	1159s	-
Spark [8]	128	1784s	$\geq$ 8000s
Giraph [8]	128	200s	$\geq$ 8000s
GraphLab [8]	128	242s	714s
GraphX [8]	128	251s	800s
Single thread (SSD)	1	153s	417s

from McSherry 2015:

"The published work on big data systems has fetishized scalability as the most important feature of a distributed data processing platform. While nearly all such publications detail their system's impressive scalability, few directly evaluate their absolute performance against reasonable benchmarks. To what degree are these systems truly improving performance, as opposed to parallelizing overheads that they themselves introduce?"

COST = "Configuration that Outperforms a Single Thread"

Perhaps surprisingly, many published systems have unbounded COST—i.e., no configuration outperforms the best single-threaded implementation—for all of the problems to which they have been applied.

BID Data Suite (1 GPU accelerated node)
[Canny and Zhao, KDD 13]

Page Rank

System	Graph VxE	Time(s)	Gflops	Procs
Hadoop	?x1.1B	198	0.015	50x8
Spark	40Mx1.5B	97.4	0.03	50x2
Twister	50Mx1.4B	36	0.09	60x4
PowerGraph	40Mx1.4B	3.6	0.8	64x8
BIDMat	60Mx1.4B	6	0.5	1x8
BIDMat+disk	60Mx1.4B	24	0.16	1x8

Latency Dirichlet Allocation (LDA)

System	Docs/hr	Gflops	Procs
Smola[15]	1.6M	0.5	100x8
PowerGraph	1.1M	0.3	64x16
BIDMach	3.6M	30	1x8x1

Performance improvements to Spark

- With increasing DRAM sizes and faster persistent storage (SSD), there is interest in improving the CPU utilization of Spark applications
  - Goal: reduce "COST"

- Efforts looking at adding efficient code generation to Spark ecosystem (e.g., generate SIMD kernels, target accelerators like GPUs, etc.) to close the gap on single node performance
  - RDD storage layouts must change to enable high-performance SIMD processing (e.g., struct of arrays instead of array of structs)
  - See Spark's Project Tungsten, Weld [Palkar Cidr '17], IBM's SparkGPU

- High-performance computing ideas are influencing design of future performance-oriented distributed systems
  - Conversely: the scientific computing community has a lot to learn from the distributed computing community about elasticity and utility computing

Spark summary

- Introduces opaque sequence abstraction (RDD) to encapsulate intermediates of cluster computations (previously… frameworks like Hadoop/MapReduce stored intermediates in the file system)
  - Observation: "files are a poor abstraction for intermediate variables in large-scale data-parallel programs"
  - RDDs are read-only, and created by deterministic data-parallel operators
  - Lineage tracked and used for locality-aware scheduling and fault-tolerance (allows recomputation of partitions of RDD on failure, rather than restore from checkpoint *)
    - Bulk operations allow overhead of lineage tracking (logging) to be low.
- Simple, versatile abstraction upon which many domain-specific distributed computing frameworks are being implemented.
  - See Apache Spark project: spark.apache.org

* Note that .persist(RELIABLE) allows programmer to request checkpointing in long lineage situations.

Modern Spark ecosystem

Compelling feature: enables integration/composition of multiple domain-specific frameworks (since all collections implemented under the hood with RDDs and scheduled using Spark scheduler)

Spark SQL

```
sqlCtx = new HiveContext(sc)
results = sqlCtx.sql(
  "SELECT * FROM people")
names = results.map(lambda p: p.name)
```

Interleave computation and database query
Can apply transformations to RDDs produced by SQL queries

Spark MLlib

Machine learning library build on top of Spark abstractions.

```
points = spark.textFile("hdfs://...")
              .map(parsePoint)

model = KMeans.train(points, k=10)
```

Spark GraphX

GraphLab-like library built on top of Spark abstractions.

```
graph = Graph(vertices, edges)
messages = spark.textFile("hdfs://...")
graph2 = graph.joinVertices(messages) {
  (id, vertex, msg) => ...
}
```