Lecture 9:

Spark

(Distributed Computing on a Cluster)

Parallel Computing

Stanford CS149, Fall 2022

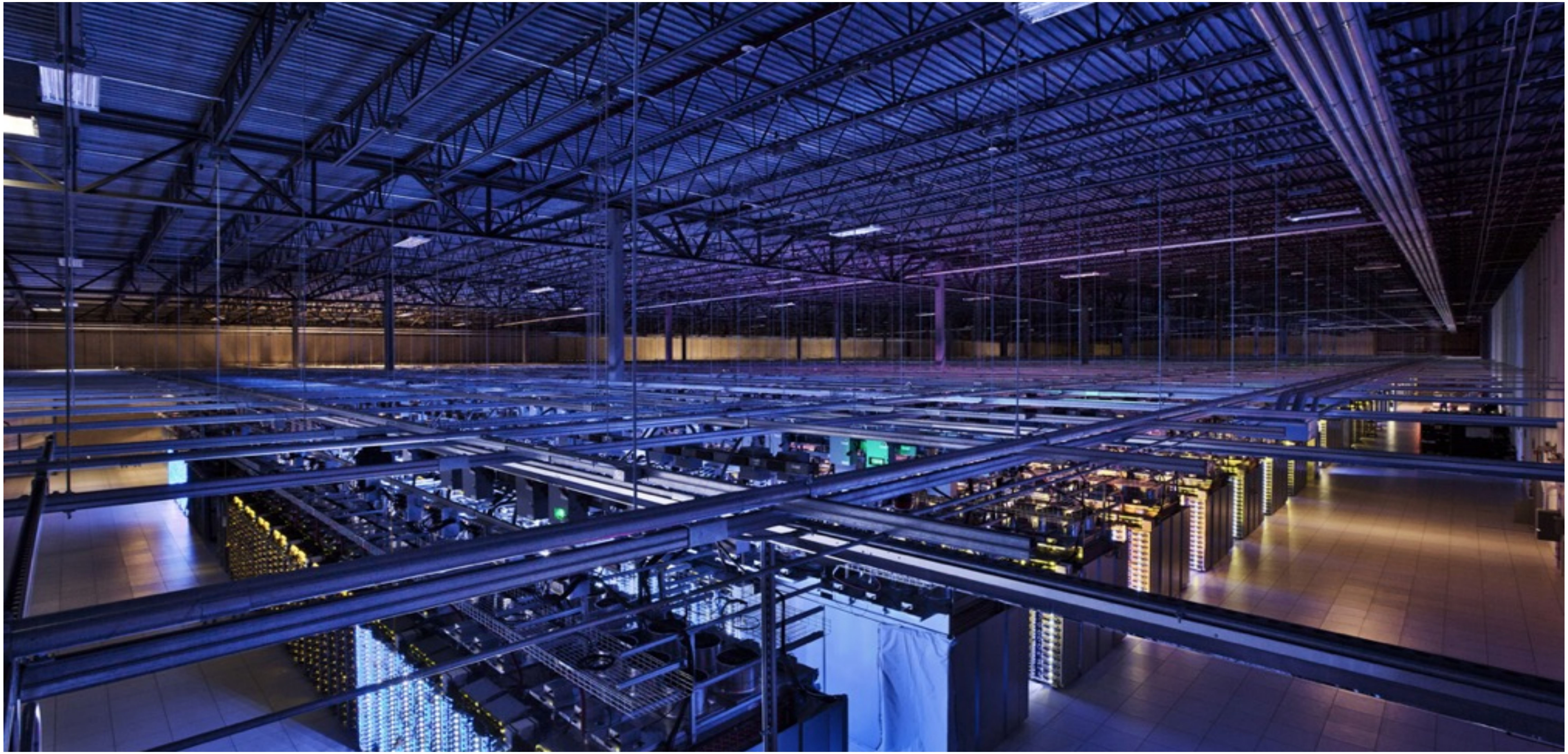
Today's Theme

- How do you program with 10,000–100,000 cores?
- How do you ensure you don't loose data if some component of the system fails?
- Programming model: data parallel operations (e.g. Map and Reduce)
- Make data parallel operations:
  - Scalable (100, 000 cores)
  - Fault-tolerant (don't loose data when something fails)
  - Efficient (optimize system performance with efficient use of memory)

Why Use A Cluster?

- Want to process 100TB of log data (1 day @Facebook)
- On 1 node: scanning @ 50MB/s = 23 days
- On 1000 nodes: scanning @ 50MB/s = 33 min
- But, very hard to utilize 1000 or 100,000 nodes!
  - Hard to program 16,000 cores
  - Something breaks every hour
  - Need efficient, reliable and usable framework

Warehouse Size Cluster



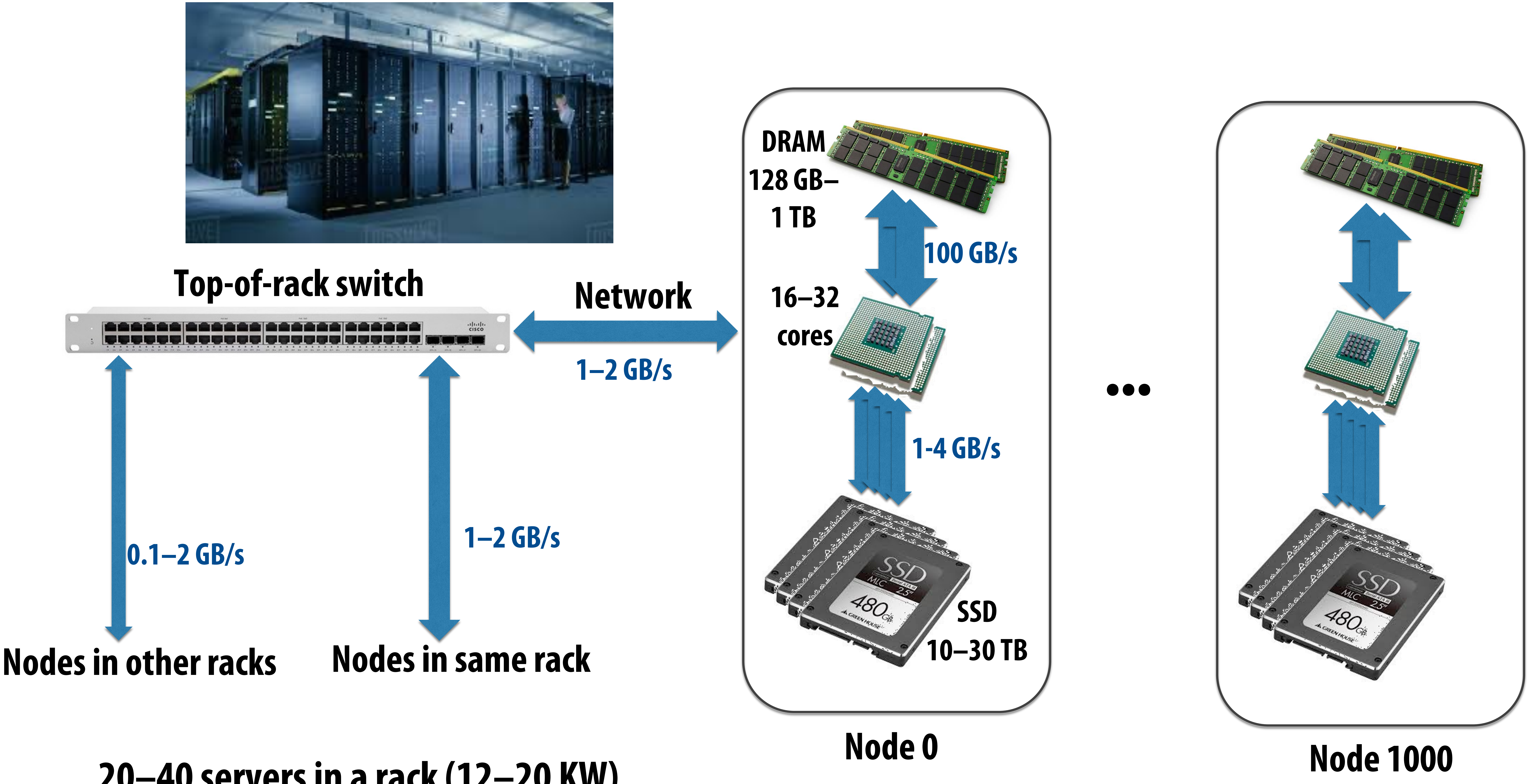
Warehouse-Scale Computers (WSC)

- **Standard architecture:**
  - Cluster of commodity Linux nodes (multicore x86)
  - Private memory ⇒ separate address spaces & separate OS
  - Ethernet network ⇒ >10Gb today

- **Cheap**
  - Built from commodity processors, networks & storage
  - 1000s of nodes for < $10M
  - WSC network is customized and expensive
    - Use a supercomputer networking ideas to provide high bandwidth across the datacenter

- **How to organize computations on this architecture?**
  - Mask issues such as load balancing and failures

Warehouse-Scale Cluster Node (Server)

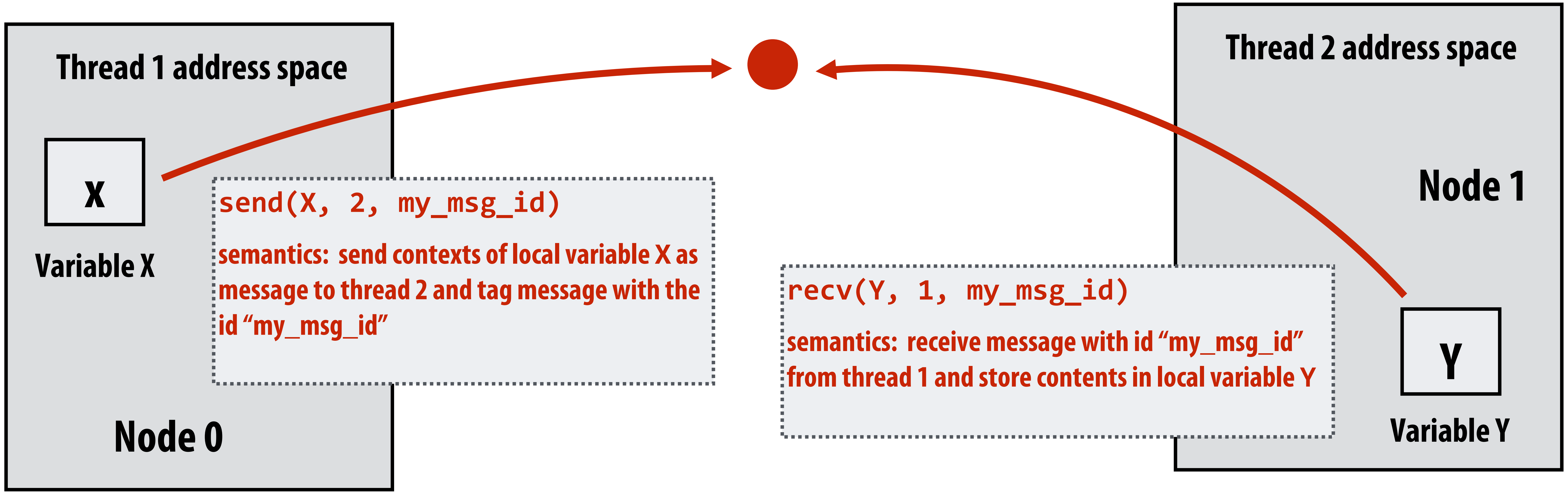


20–40 servers in a rack (12–20 KW)

Consider bandwidths, what conclusions can you make?

Message passing model (abstraction)

- Threads operate within their own private address spaces
- Threads communicate by sending/receiving messages
  - send: specifies recipient, buffer to be transmitted, and optional message identifier ("tag")
  - receive: sender, specifies buffer to store data, and optional message identifier
  - Sending messages is the only way to exchange data between threads 1 and 2
    - Why?



(Communication operations shown in red)

Illustration adopted from Culler, Singh, Gupta

Storage Systems

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

- Answer: Distributed File System
  - Provides global file namespace
  - Google GFS, Hadoop HDFS

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

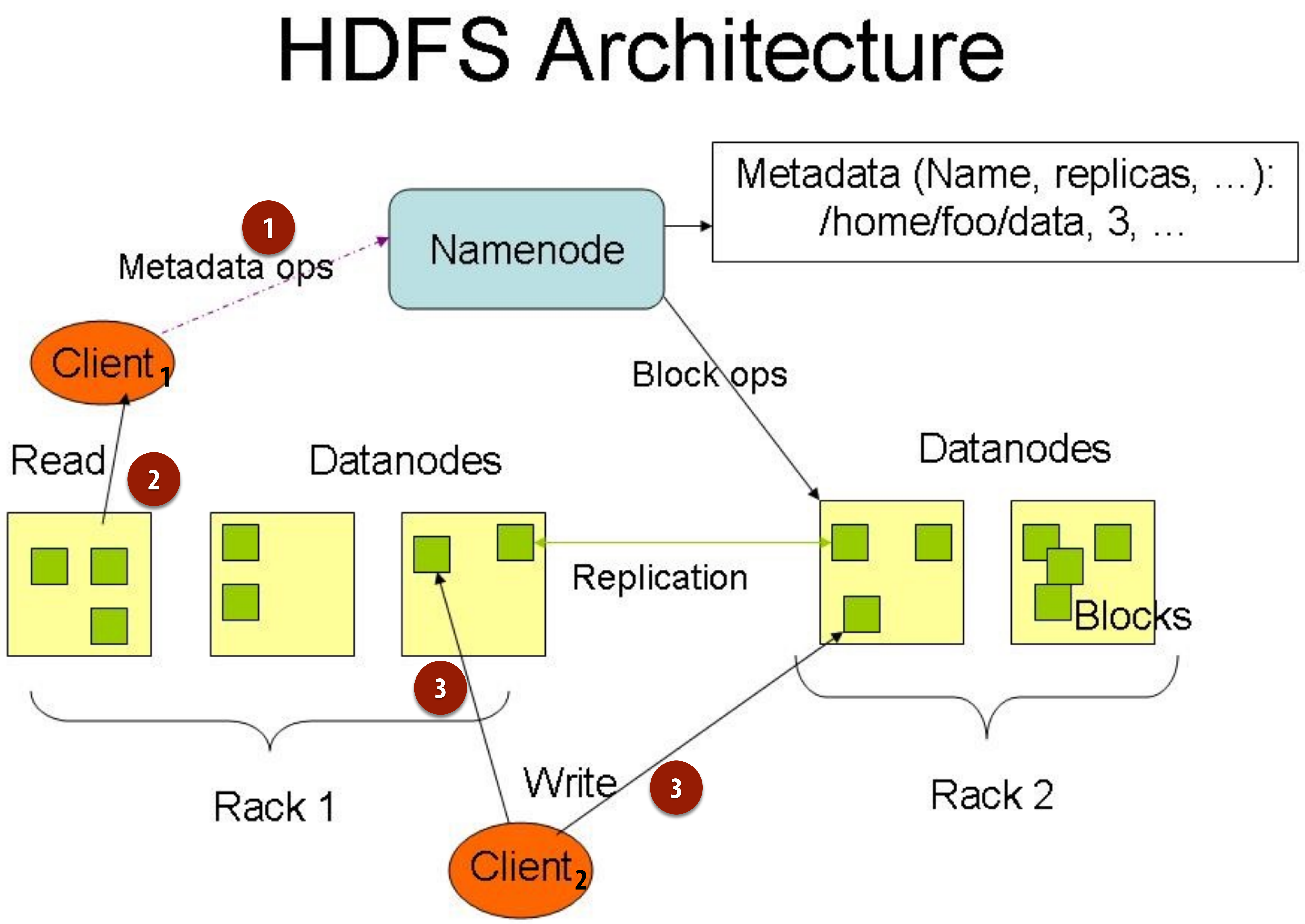
Distributed File System (GFS)

- **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; usually replicated

- **Client library for file access**
  - Talks to master to find chunk (data) servers
  - Connects directly to chunk servers to access data

Hadoop Distributed File System (HDFS)



- **Global namespace**
- **Files broken into blocks**
  - **– Typically 256 MB each**
  - **– Each block replicated on multiple DataNodes**
- **Intelligent Client**
  - **– Client finds locations of blocks from NameNode**
  - **– Client accesses data directly from DataNode**

Let's say CS149 gets very popular…

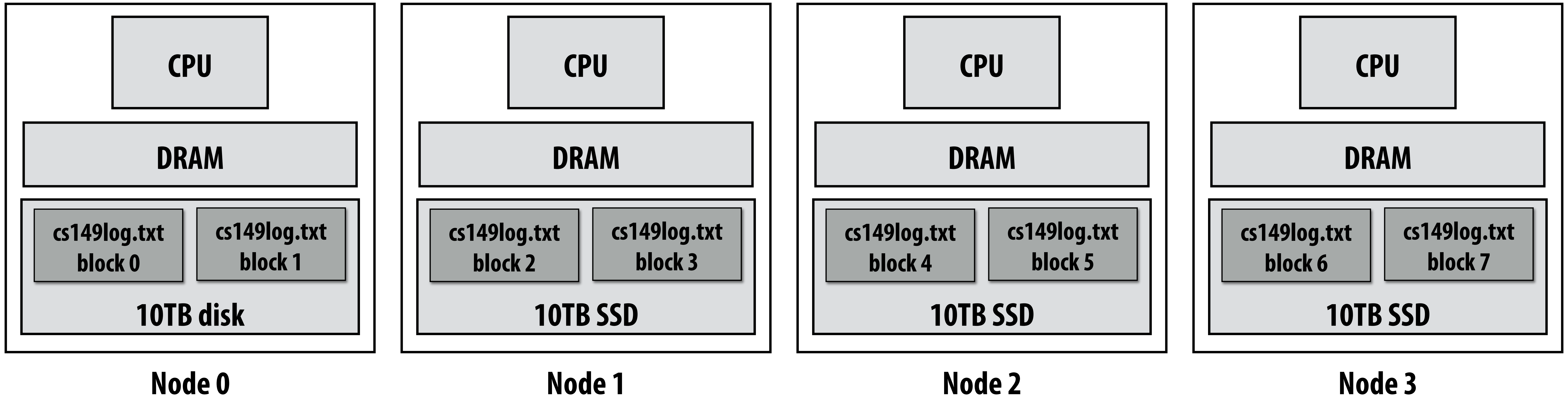
A log of page views on the course web site

```
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"
```

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 10 TB SSD
Contents of cs149log.txt are distributed evenly in blocks across the cluster



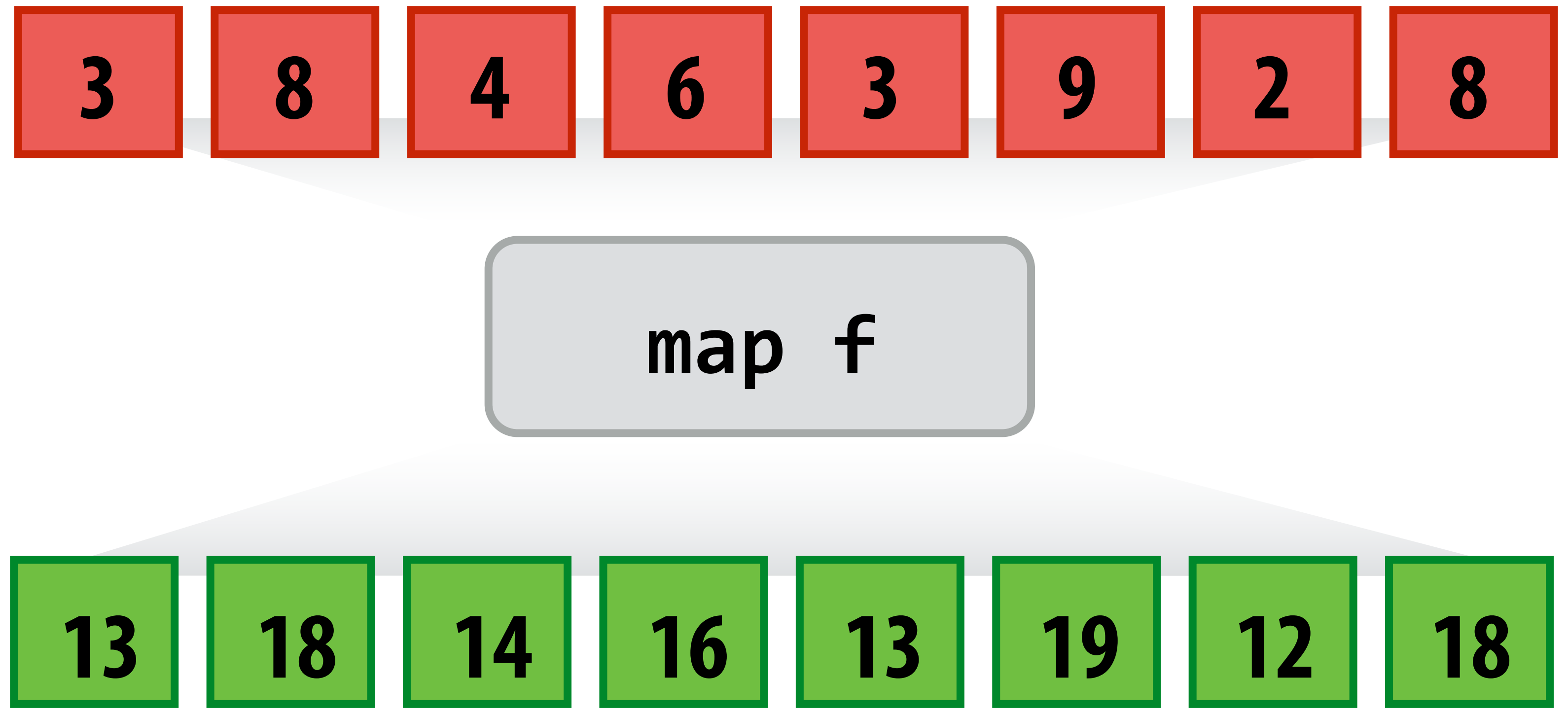
Imagine your professors want to know a bit more about the glut of students visiting the CS149 web site…

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

Map

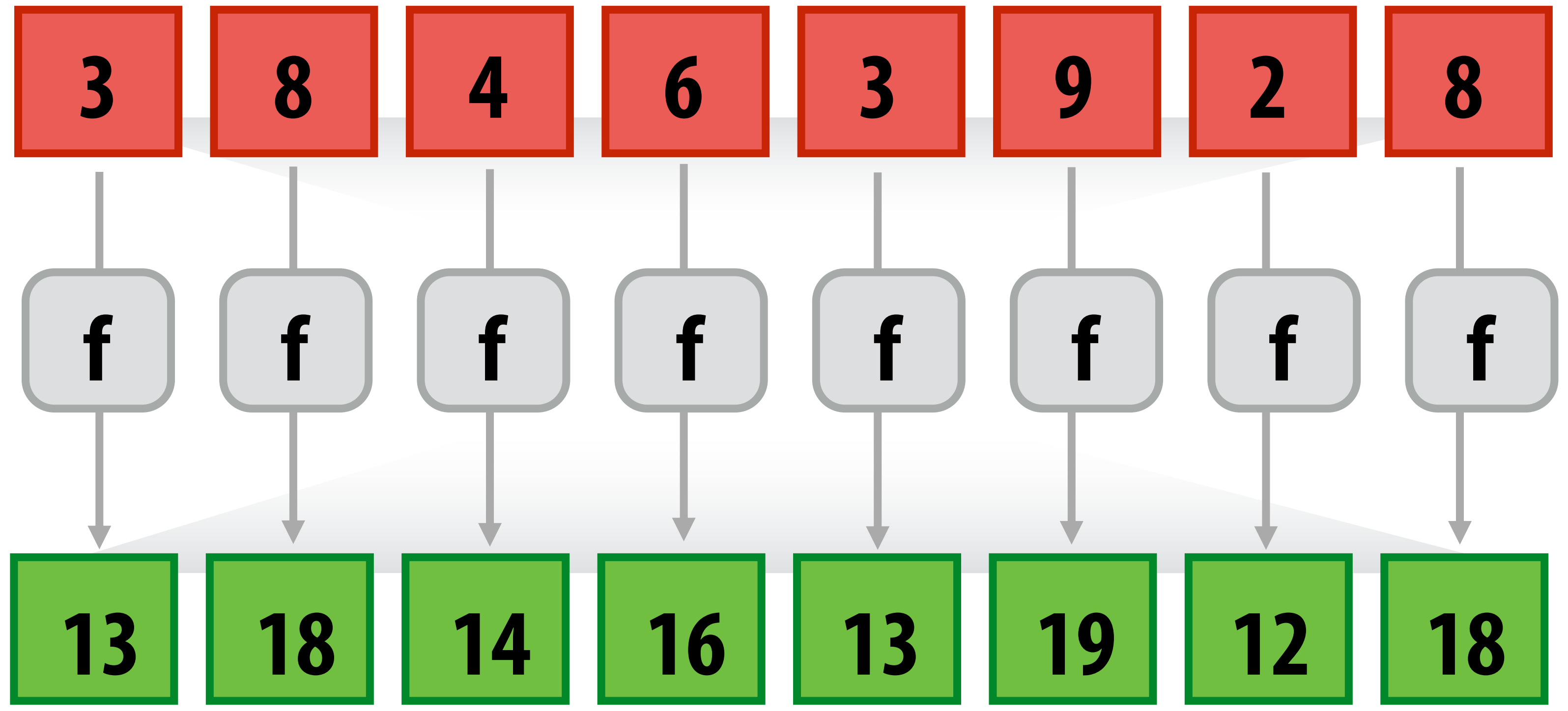
- Higher order function (function that takes a function as an argument)
- Applies side-effect free unary function `f :: a -> b` to all elements of input sequence, to produce output sequence of the same length
- In a functional language (e.g., Haskell)
  - `map :: (a -> b) -> seq a -> seq b`
- In C++:

```
template<class InputIt, class OutputIt, class UnaryOperation>
OutputIt transform(InputIt first1, InputIt last1, OutputIt d_first,
                   UnaryOperation unary_op);
```



C++

```
int f(int x) { return x + 10; }

int a[] = {3, 8, 4, 6, 3, 9, 2, 8};
int b[8];
std::transform(a, a+8, b, f);
```



Haskell

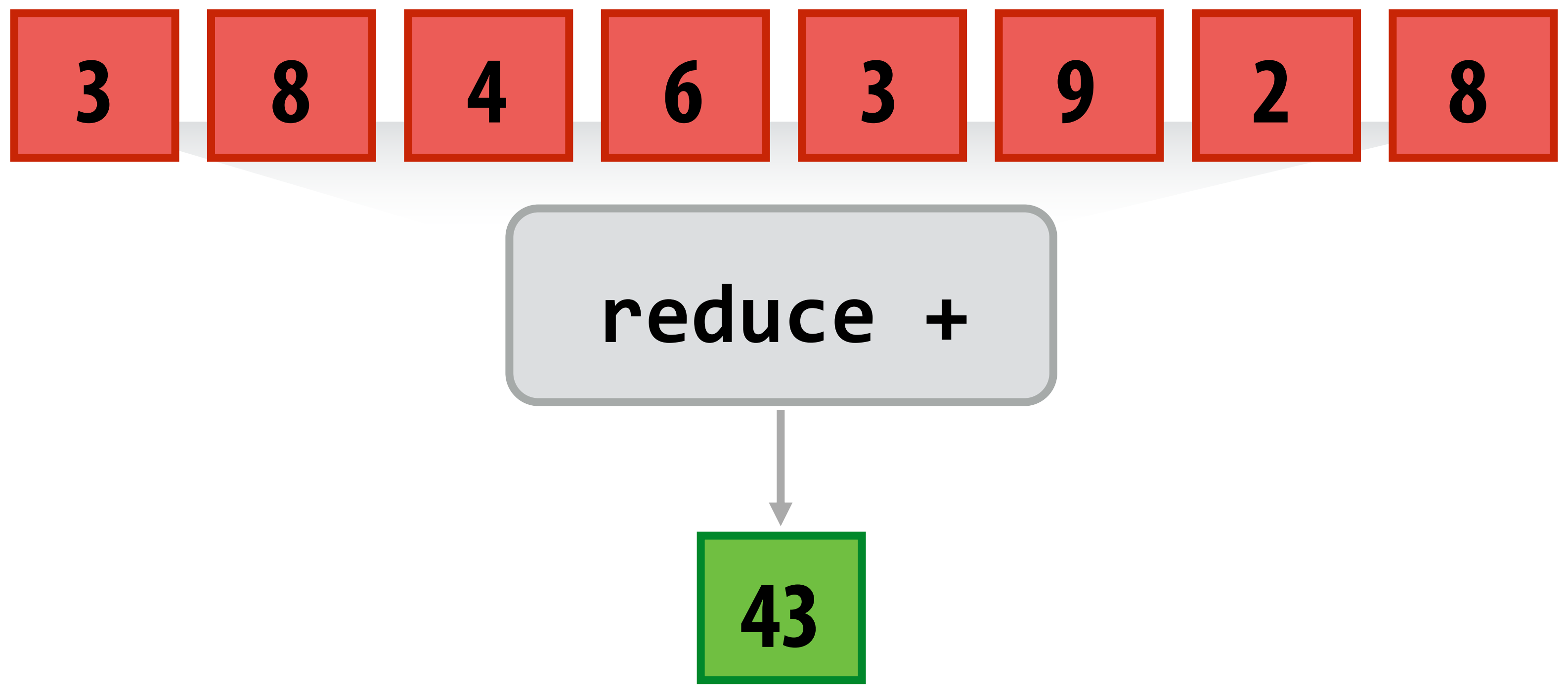
```
a = [3, 8, 4, 6, 3, 9, 2, 8]
f x = x + 10
b = map f a
```

Reduce

- Apply binary operation f to each element and an accumulated value
  - f :: (b,a) -> b
  - reduce :: ((b,a) -> b) -> seq a -> b

E.g., in Scala:

```
def reduce[A](f: (B, A) => B, l: List[A]): B
```



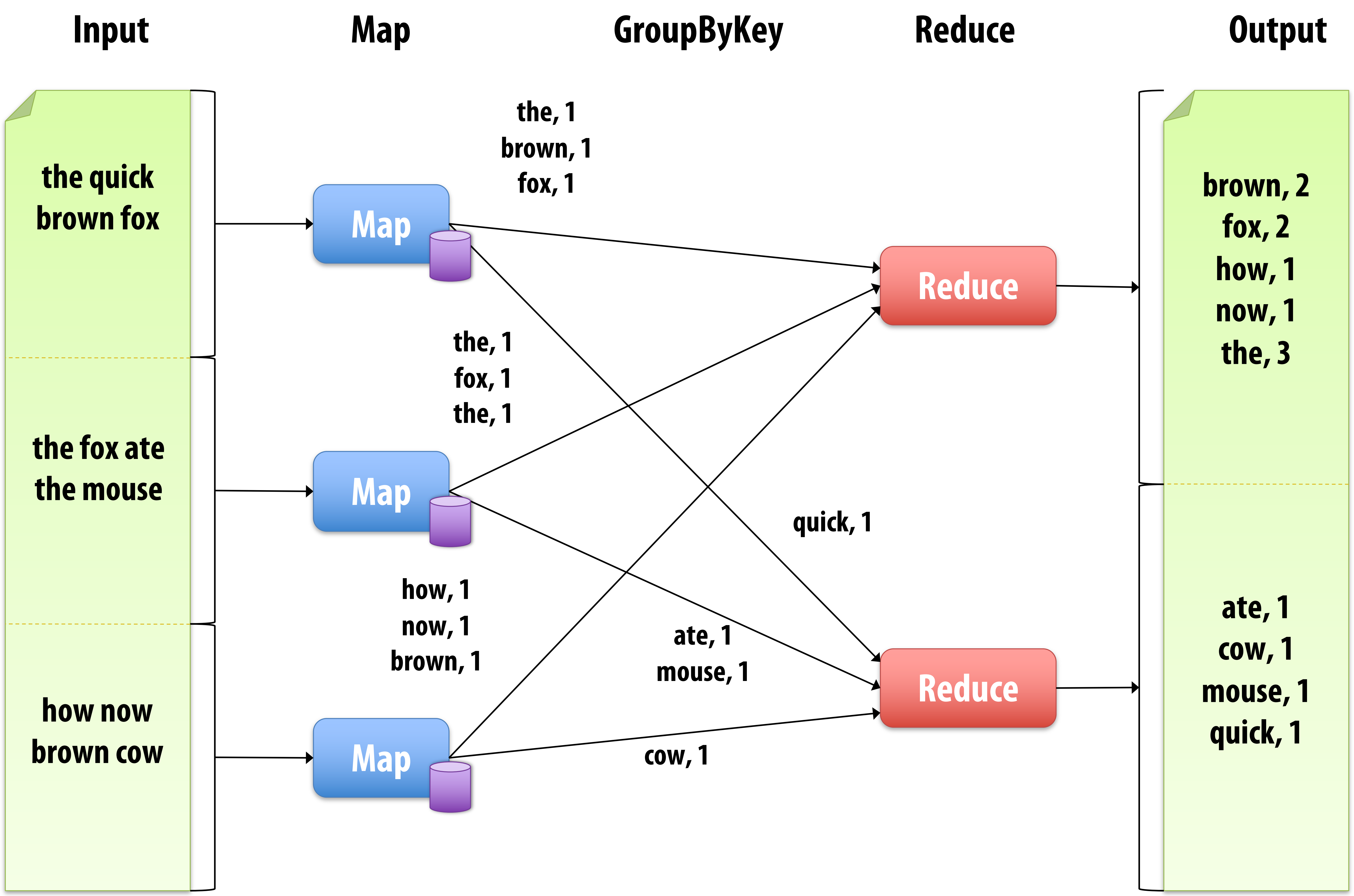
MapReduce Programming Model

```
// 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

MapReduce Dataflow for Word Count



Should be called MapGroupByKeyReduce

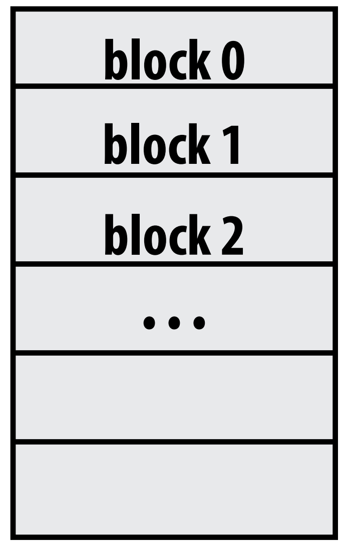
Step 1: Running the mapper function

```
// 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);
```

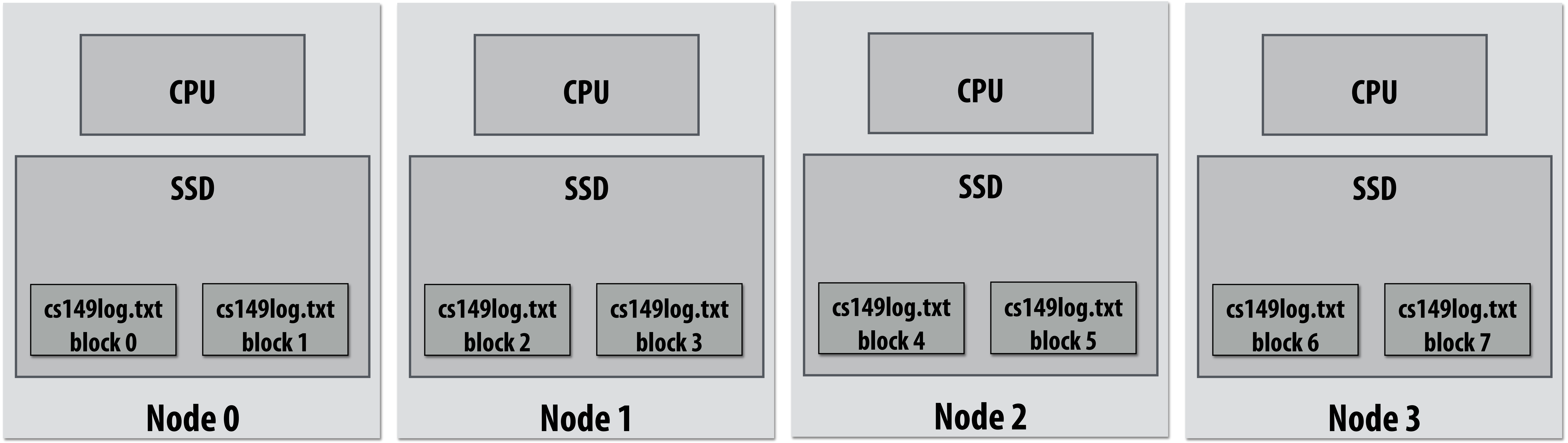
Step 1: run mapper function on all lines of file

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

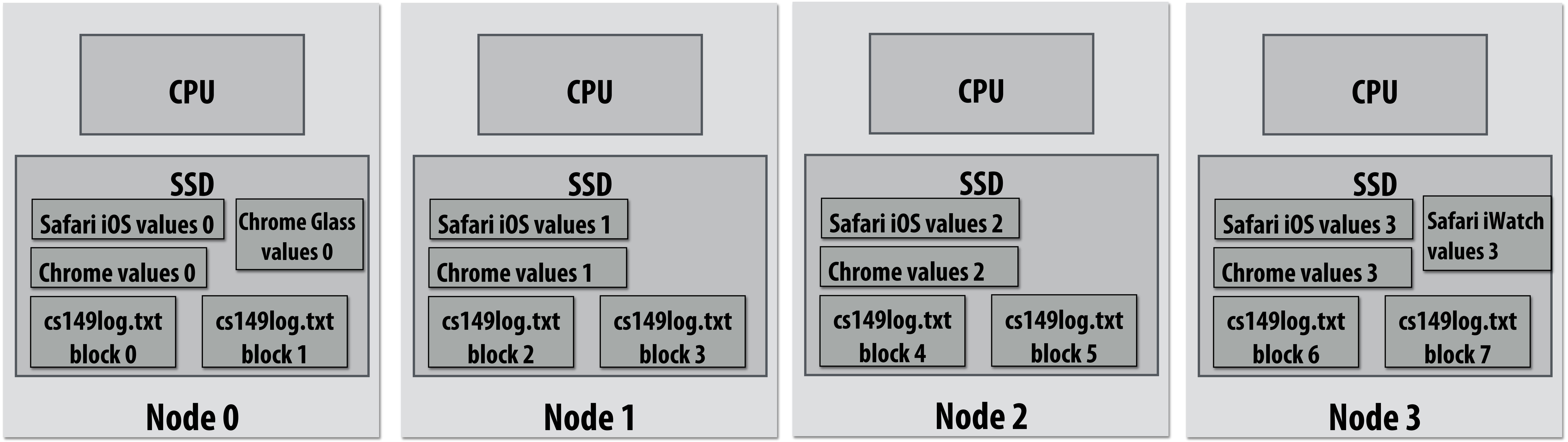
Step 3: Run reducer function on all keys

Question 1: how to assign reducer tasks?

Question 2: how to get all data for key onto the correct reduce worker node?

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);
}
```

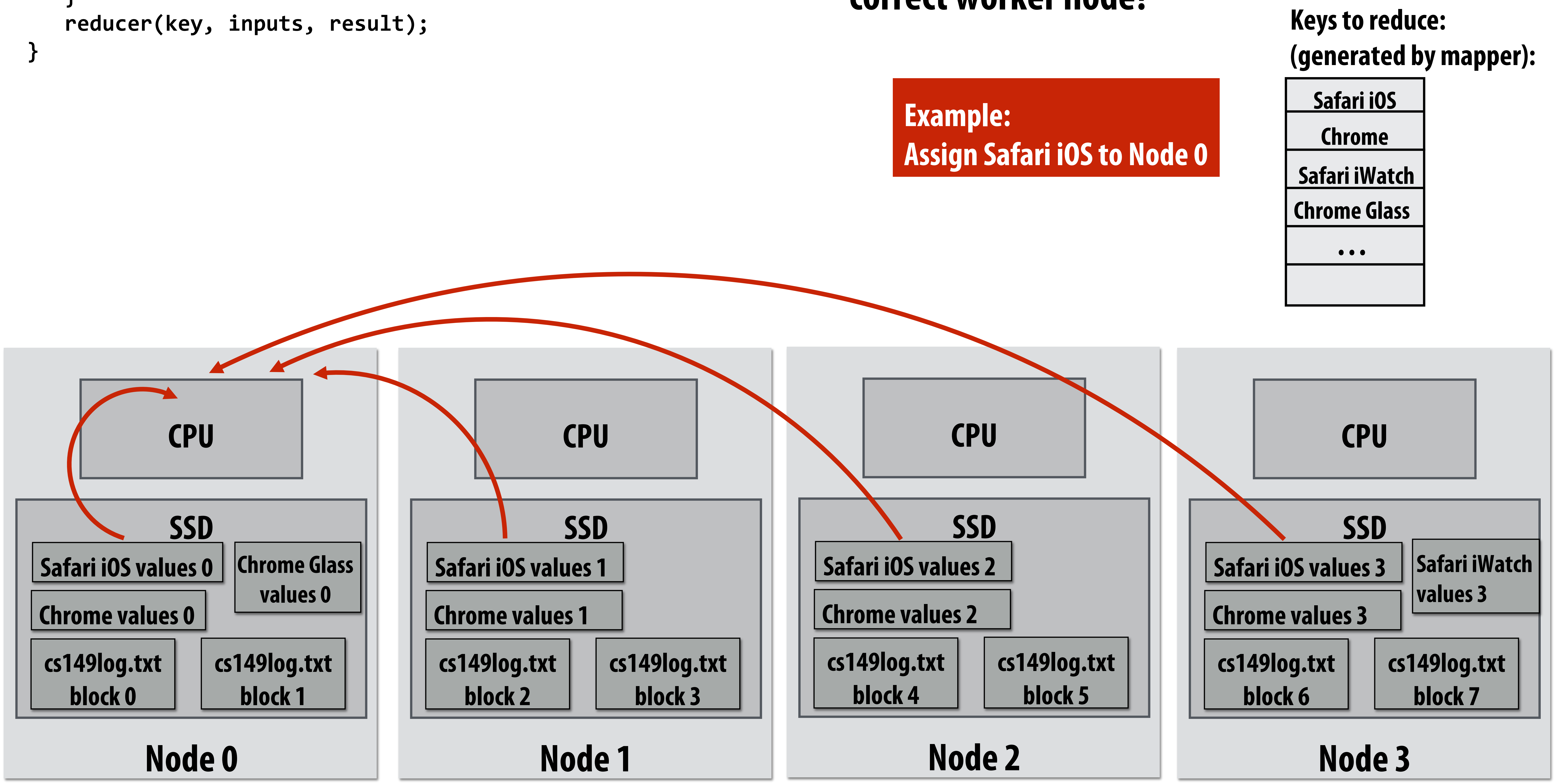
Step 2: Prepare intermediate data for reducer.

Step 3: Run reducer function on all keys.

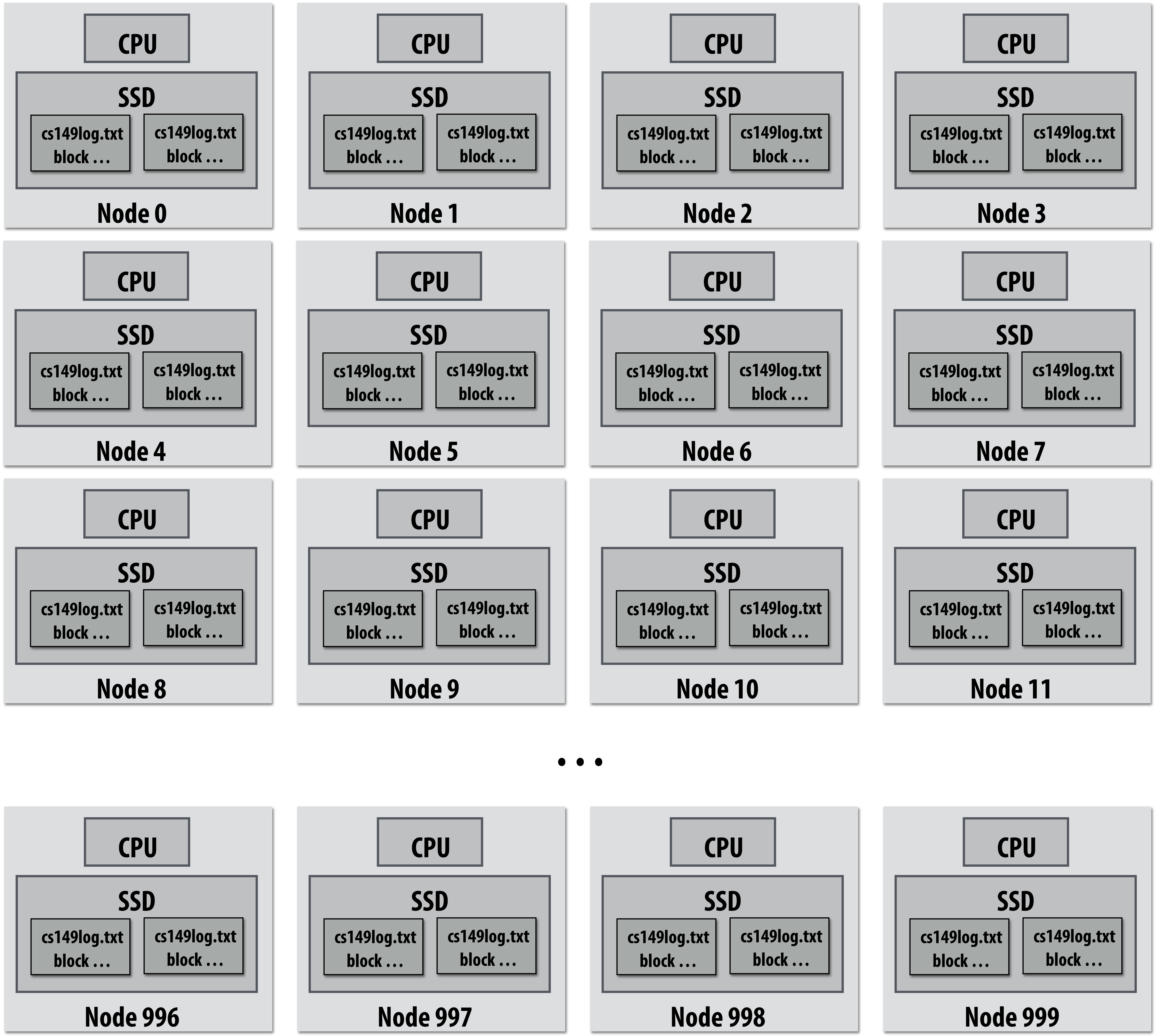
Question: how to assign reducer tasks?

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

Example: Assign Safari iOS to Node 0



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

MapReduce Benefits

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

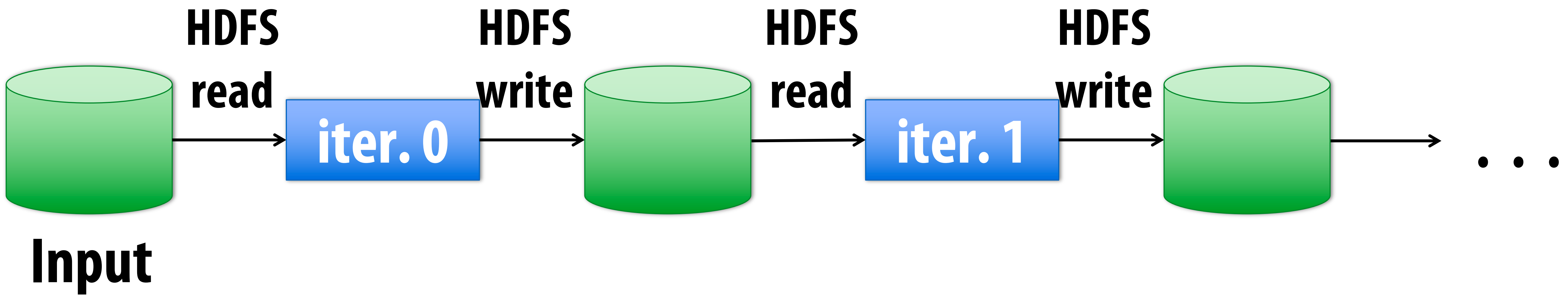
- But… the story doesn't end here!

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

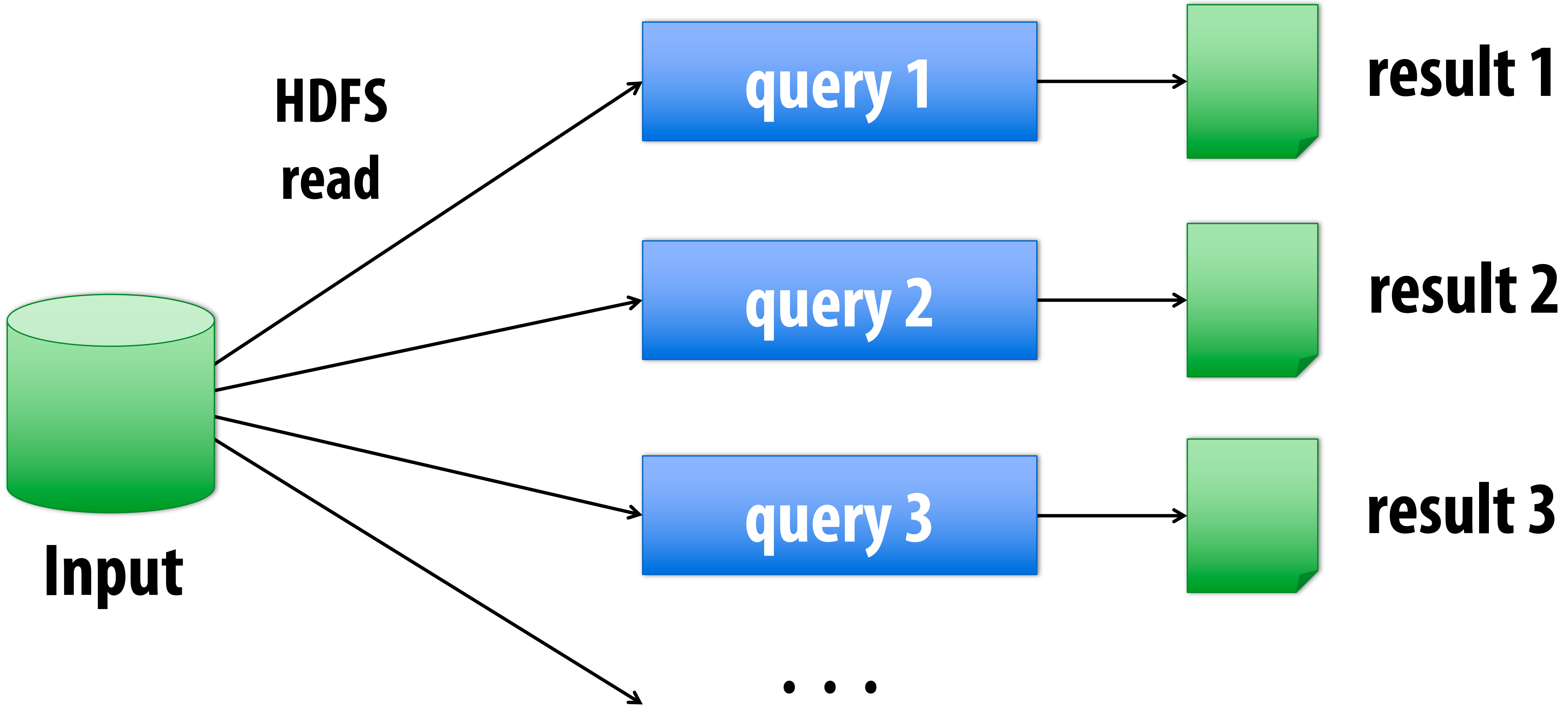
- Iterative algorithms must load from disk each iteration
  - Example graph processing:

```
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]);
}
```

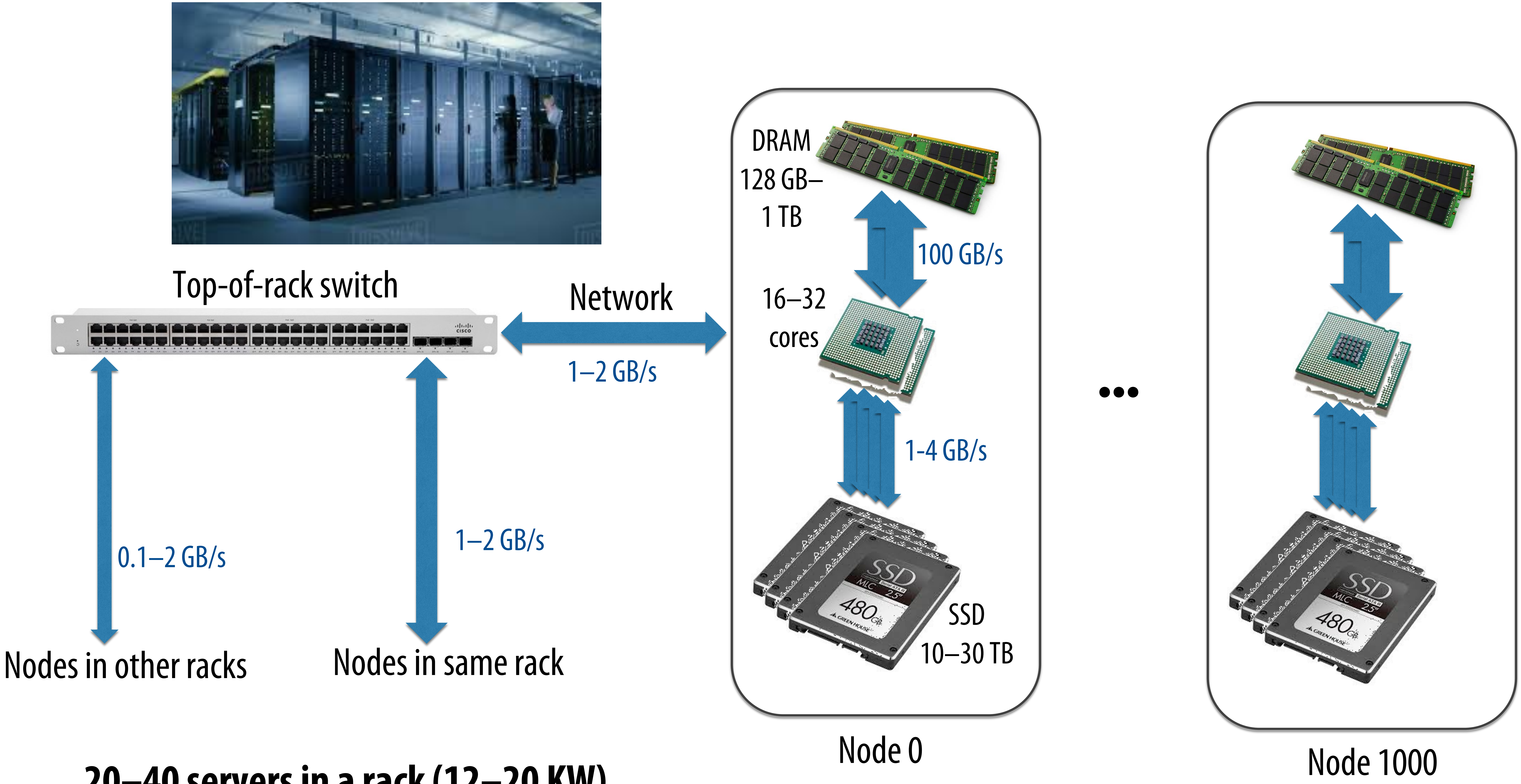


MapReduce Limitations

- MapReduce greatly simplified “big data” analysis
- But users quickly needed more:
  - More complex, multi-stage applications (e.g. iterative machine learning & graph processing)
  - More interactive ad-hoc queries



Warehouse-Scale Cluster Node (Server)



20–40 servers in a rack (12–20 KW)

Consider bandwidths, what conclusions can you make?

2009: Application Trends

- **Despite huge amounts of data, many working sets in big data clusters fit in memory**

Memory (GB)	Facebook (% jobs)	Microsoft (% jobs)	Yahoo! (% jobs)
8	69	38	66
16	74	51	81
32	**96**	**82**	**97.5**
64	97	98	99.5
128	98.8	99.4	99.8
192	99.5	100	100
256	99.6	100	100

*G Ananthanarayanan, A. Ghodsi, S. Shenker, I. Stoica, "Disk-Locality in Datacenter Computing Considered Irrelevant", HotOS 2011

Apache Spark

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)

Resilient Distributed Dataset (RDD)

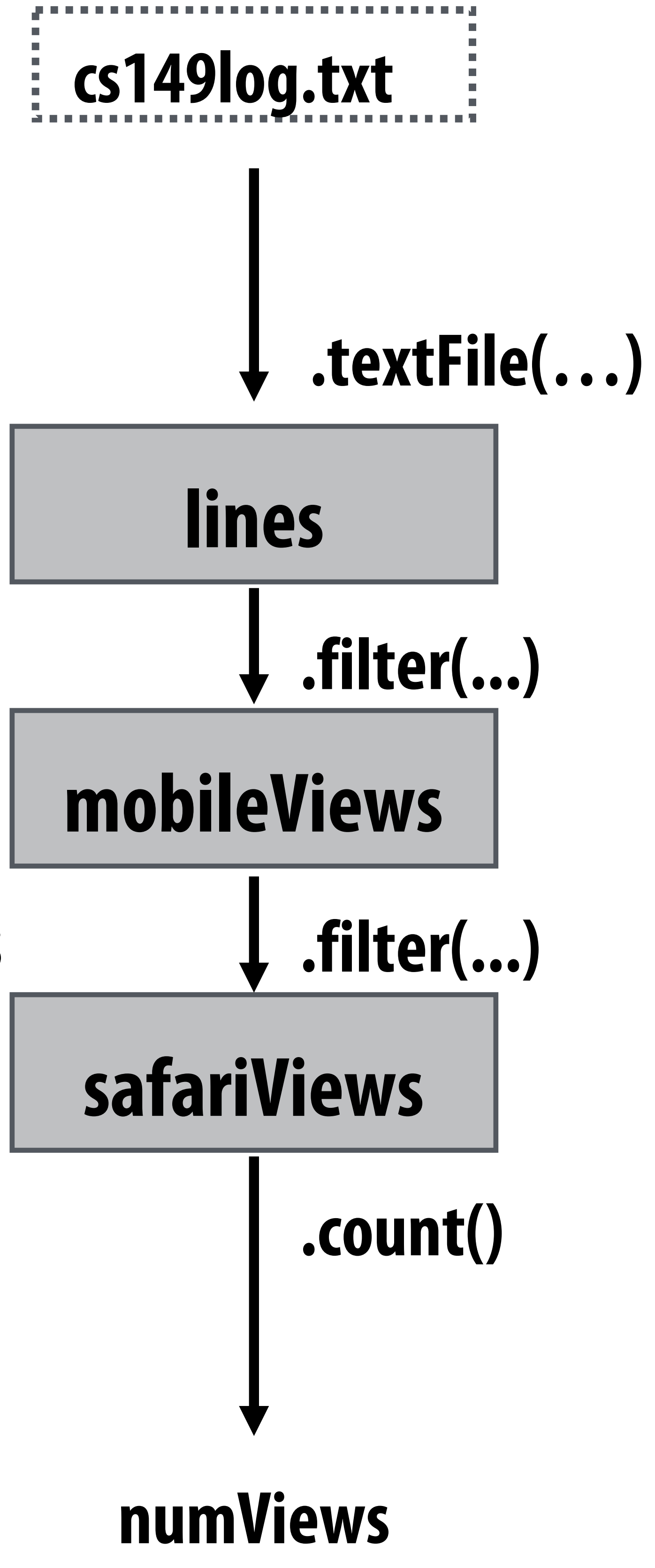
Spark's key programming **abstraction**:

- 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
val lines = spark.textFile(“hdfs://cs149log.txt”);

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

// another filter() transformation
val safariViews = mobileViews.filter((x: String) => x.contains(“Safari”));

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

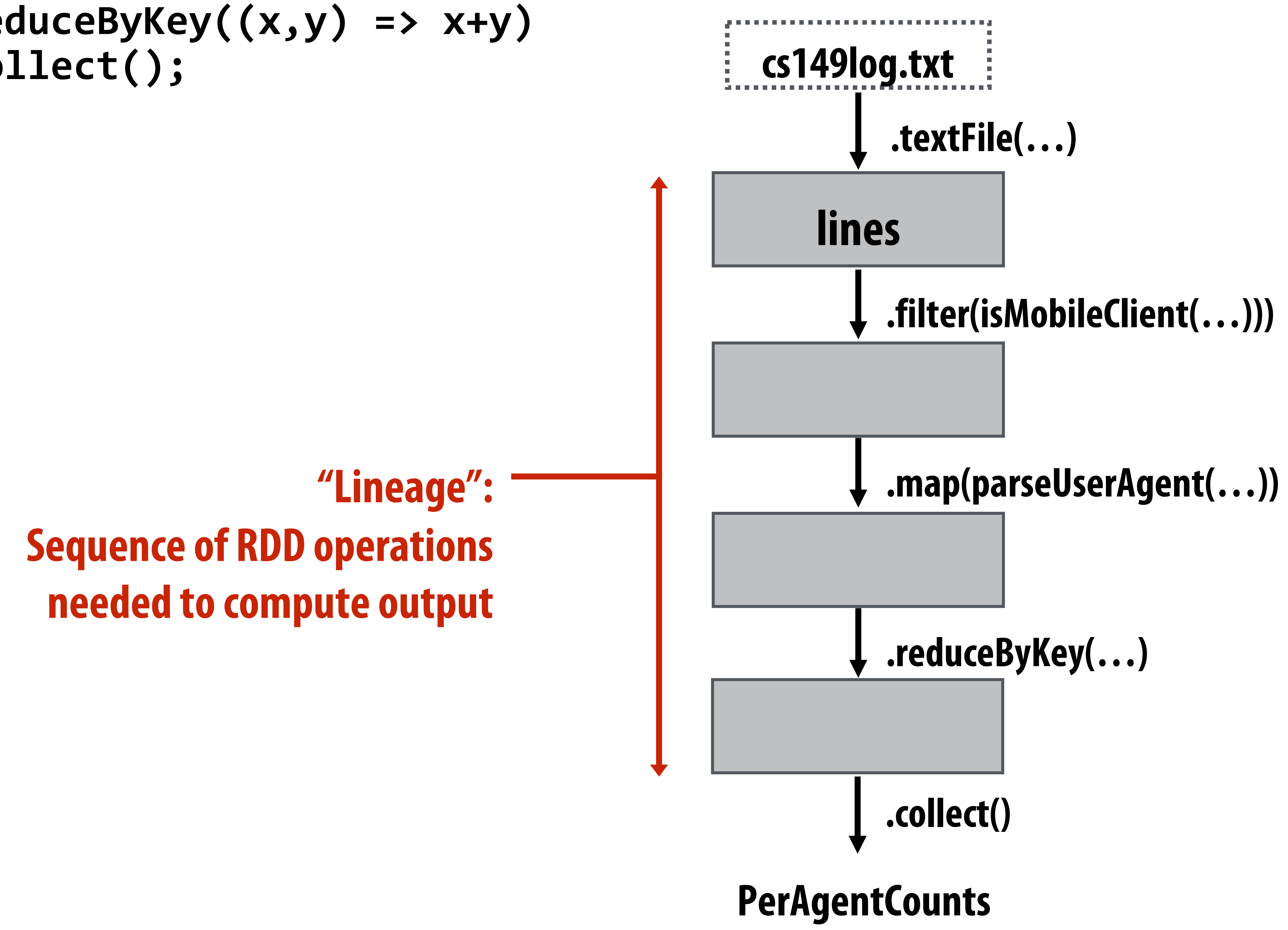
int



Repeating the MapReduce Example

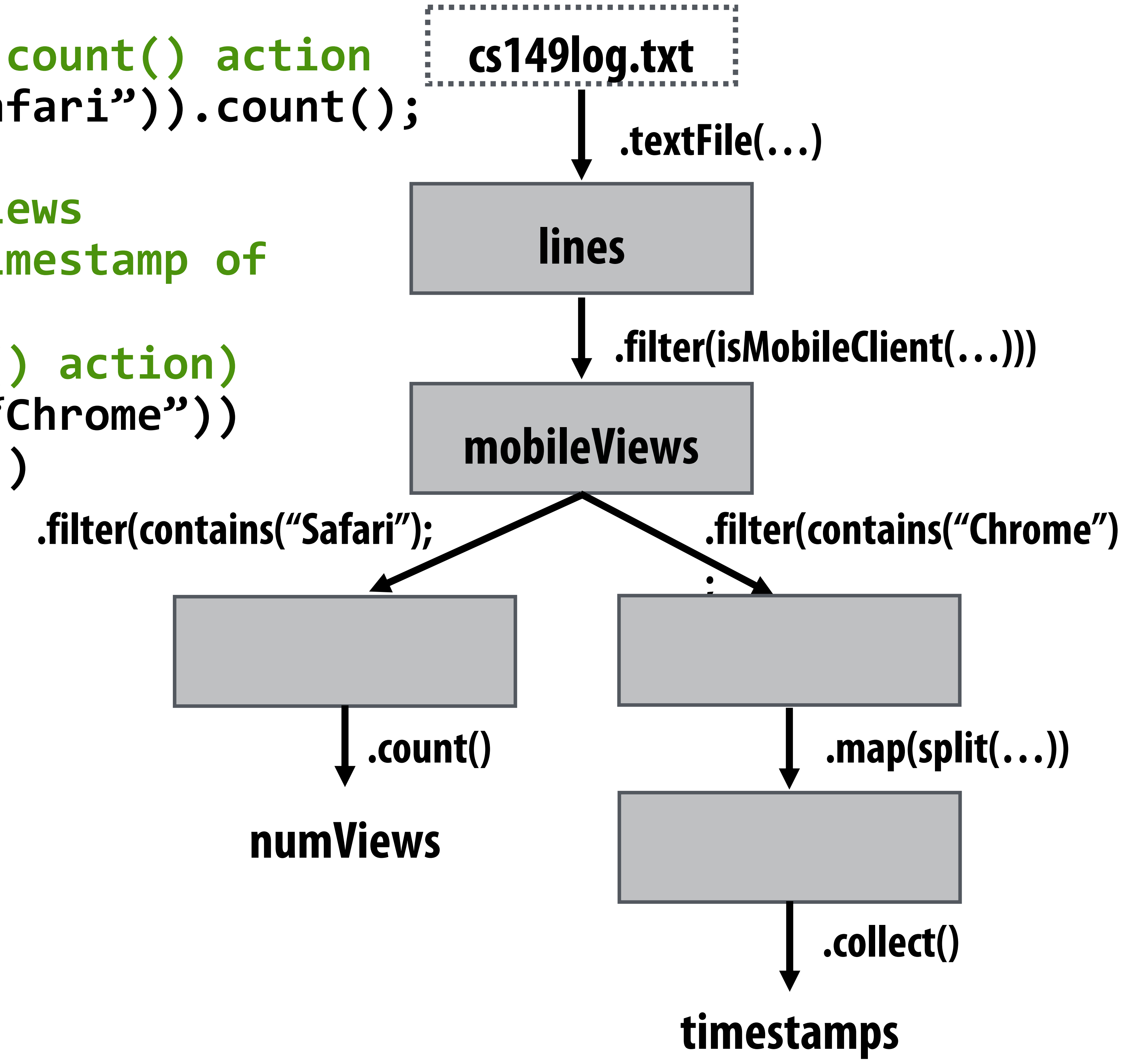
```
// 1. create RDD from file system data
// 2. create RDD with only lines from mobile clients
// 3. create RDD with elements of type (String,Int) from line string
// 4. group elements by key
// 5. call provided reduction function on all keys to count views
val perAgentCounts =  spark.textFile("hdfs://cs149log.txt")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

// create RDD using filter() transformation on lines
val 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
val 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)
val 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)

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

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

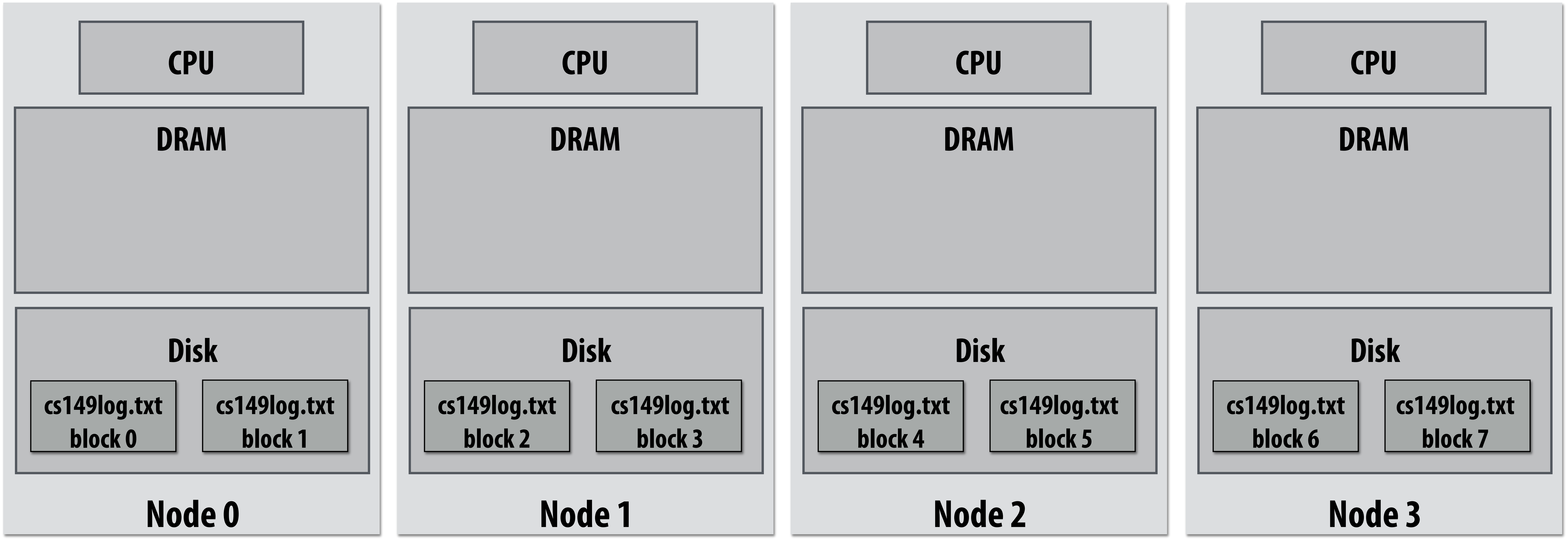
$$
\begin{aligned}
count() &\quad : \quad \mathrm{RDD[T]} \Rightarrow \mathrm{Long} \\
collect() &\quad : \quad \mathrm{RDD[T]} \Rightarrow \mathrm{Seq[T]} \\
reduce(f : (\mathrm{T}, \mathrm{T}) \Rightarrow \mathrm{T}) &\quad : \quad \mathrm{RDD[T]} \Rightarrow \mathrm{T} \\
lookup(k : \mathrm{K}) &\quad : \quad \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{Seq[V]} \ \text{(On hash/range partitioned RDDs)} \\
save(path : \mathrm{String}) &\quad : \quad \text{Outputs RDD to a storage system, } \textit{e.g.,} \text{ HDFS}
\end{aligned}
$$

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();
```

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



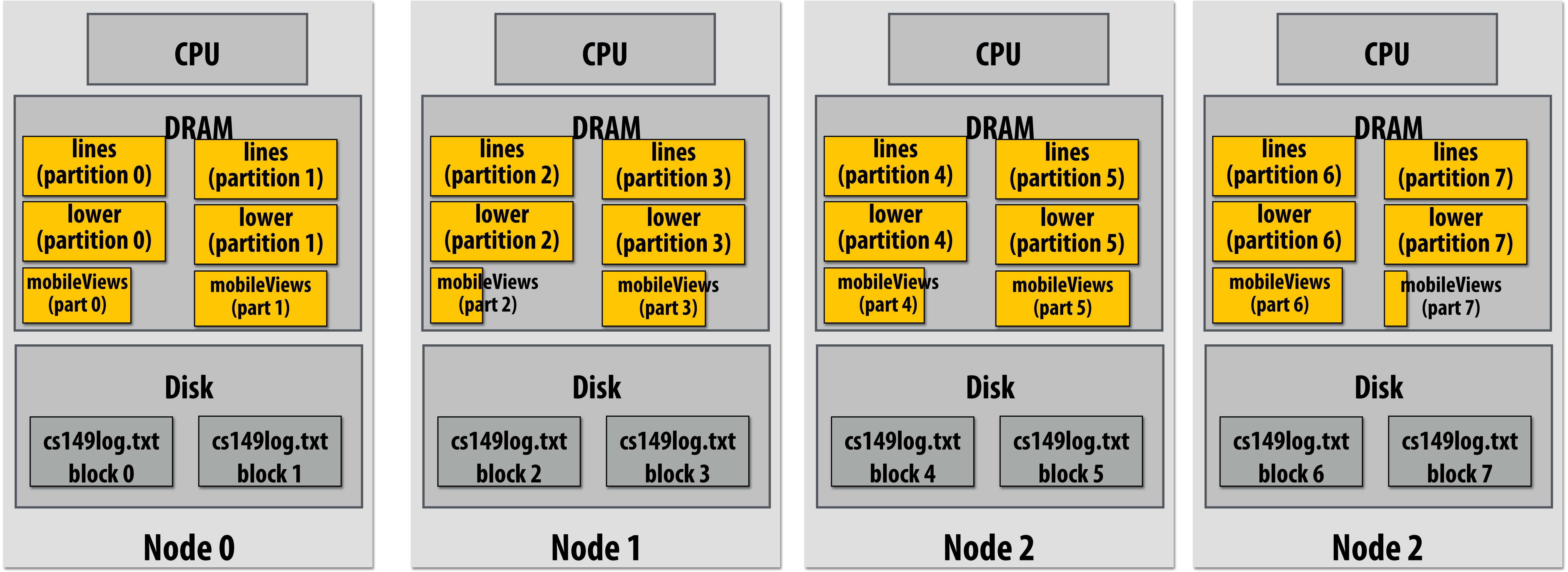
How do we implement RDDs?

In particular, how should they be stored?

Parallel Performance = Parallelism + Locality

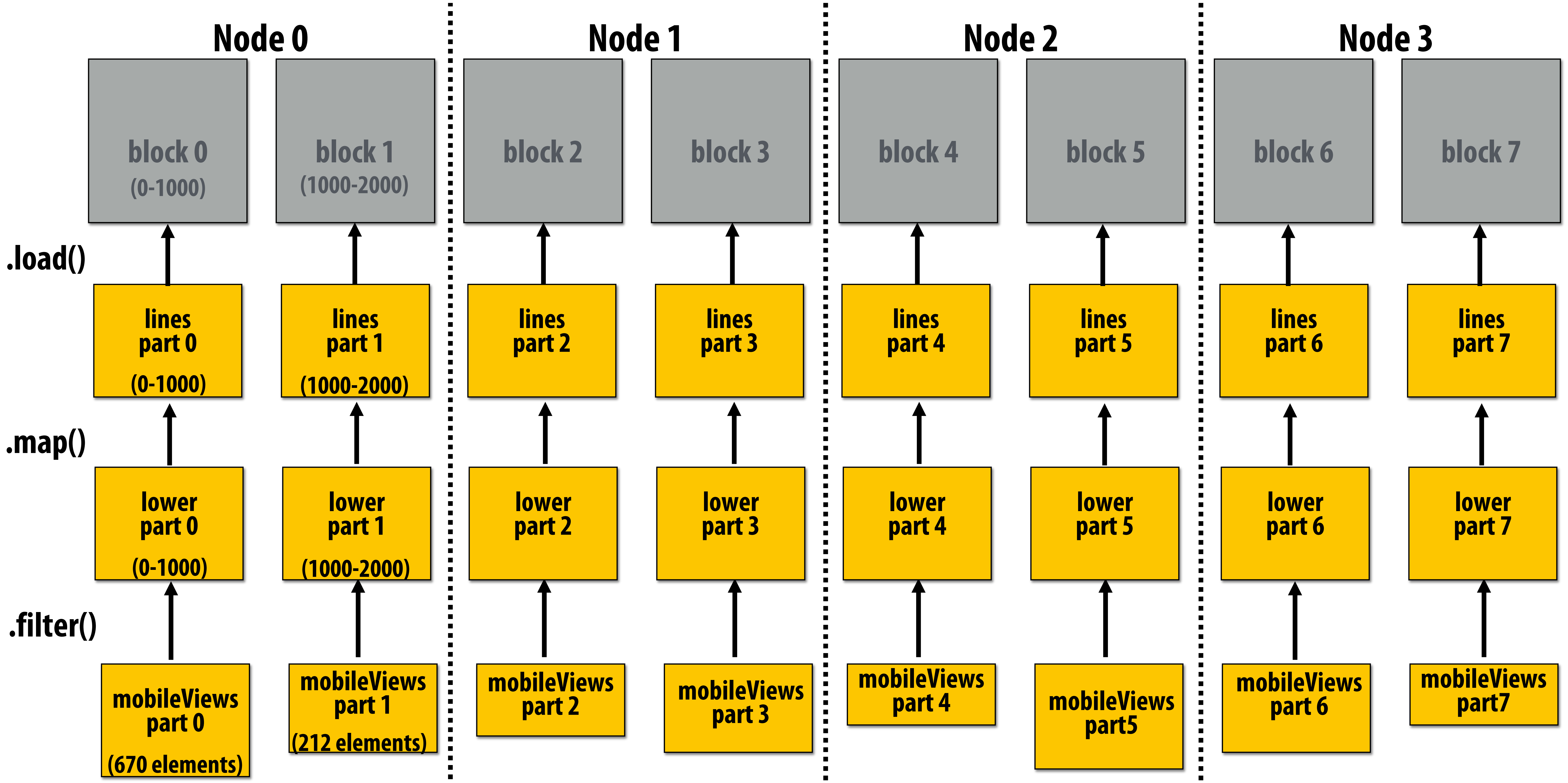
```
val lines = spark.textFile("hdfs://cs149log.txt");
val lower = lines.map(_.toLower());
val mobileViews = lower.filter(x => isMobileClient(x));
val howMany = mobileViews.count();
```

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

Review: 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);
```

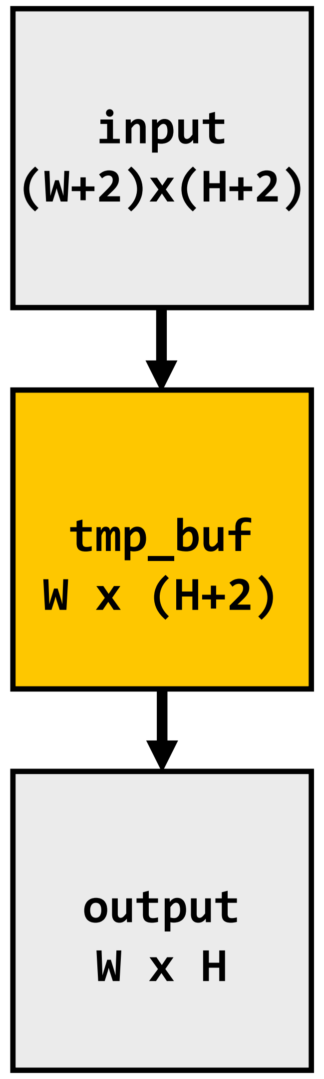
Four loads, one store per 3 math ops (arithmetic intensity = 3/5)

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

Review: why did we perform this transform?

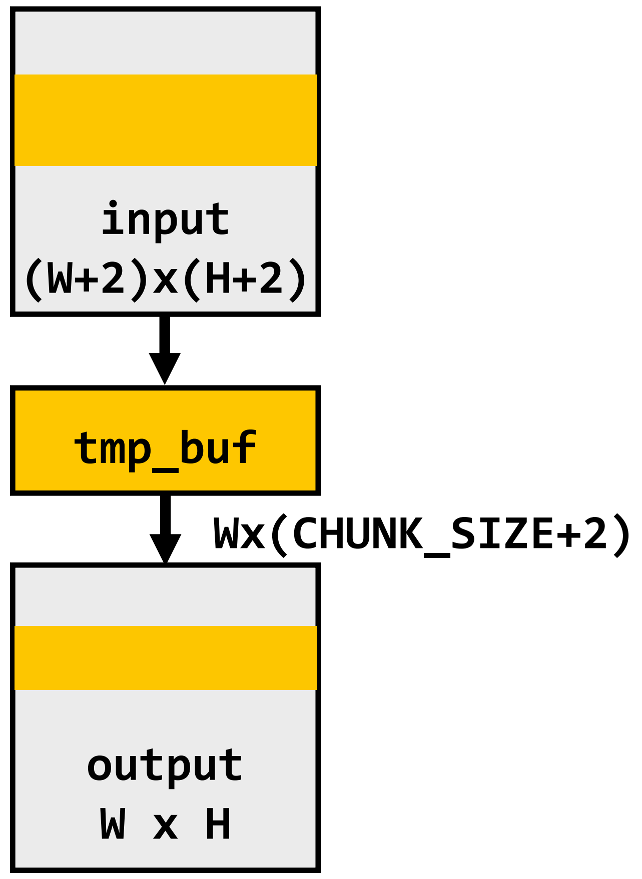
Program 1

```
int WIDTH = 1024;
int HEIGHT = 1024;
float input[(WIDTH+2) * (HEIGHT+2)];
float tmp_buf[WIDTH * (HEIGHT+2)];
float output[WIDTH * HEIGHT];

float weights[] = {1.0/3, 1.0/3, 1.0/3};

// blur image horizontally
for (int j=0; j<(HEIGHT+2); j++)
  for (int i=0; i<WIDTH; i++) {
    float tmp = 0.f;
    for (int ii=0; ii<3; ii++)
      tmp += input[j*(WIDTH+2) + i+ii] * weights[ii];
    tmp_buf[j*WIDTH + i] = tmp;
  }

// blur tmp_buf vertically
for (int j=0; j<HEIGHT; j++) {
  for (int i=0; i<WIDTH; i++) {
    float tmp = 0.f;
    for (int jj=0; jj<3; jj++)
      tmp += tmp_buf[(j+jj)*WIDTH + i] * weights[jj];
    output[j*WIDTH + i] = tmp;
  }
}
```



Program 2

```
int WIDTH = 1024;
int HEIGHT = 1024;
float input[(WIDTH+2) * (HEIGHT+2)];
float tmp_buf[WIDTH * (CHUNK_SIZE+2)];
float output[WIDTH * HEIGHT];

float weights[] = {1.0/3, 1.0/3, 1.0/3};

for (int j=0; j<HEIGHT; j+CHUNK_SIZE) {

  // blur region of image horizontally
  for (int j2=0; j2<CHUNK_SIZE+2; j2++)
    for (int i=0; i<WIDTH; i++) {
      float tmp = 0.f;
      for (int ii=0; ii<3; ii++)
        tmp += input[(j+j2)*(WIDTH+2) + i+ii] *
weights[ii];
      tmp_buf[j2*WIDTH + i] = tmp;

  // blur tmp_buf vertically
  for (int j2=0; j2<CHUNK_SIZE; j2++)
    for (int i=0; i<WIDTH; i++) {
      float tmp = 0.f;
      for (int jj=0; jj<3; jj++)
        tmp += tmp_buf[(j2+jj)*WIDTH + i] * weights[jj];
      output[(j+j2)*WIDTH + i] = tmp;
    }
}
```



Both of the previous examples involved globally restructuring the order of computation to improve producer-consumer locality

(improve arithmetic intensity of program)

Fusion with RDDs

- **Why is it possible to fuse RDD transformations such as map and filter but not possible with transformations such as groupByKey and Sort?**

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

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

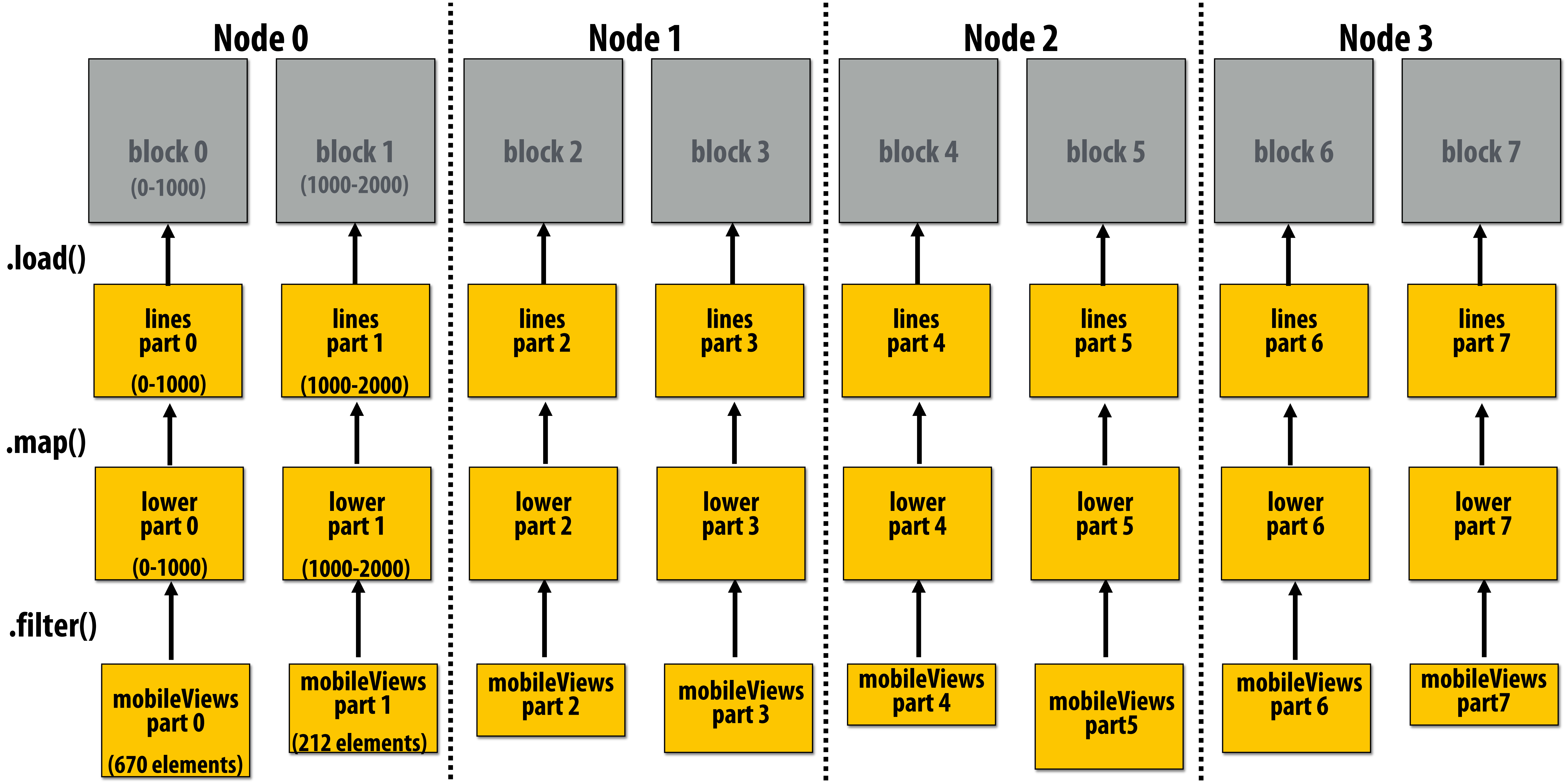
```
int count = 0;
while (inputFile.eof()) {
   string line = inputFile.readLine();
   string lower = line.toLower;
   if (isMobileClient(lower))
     count++;
}
```

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

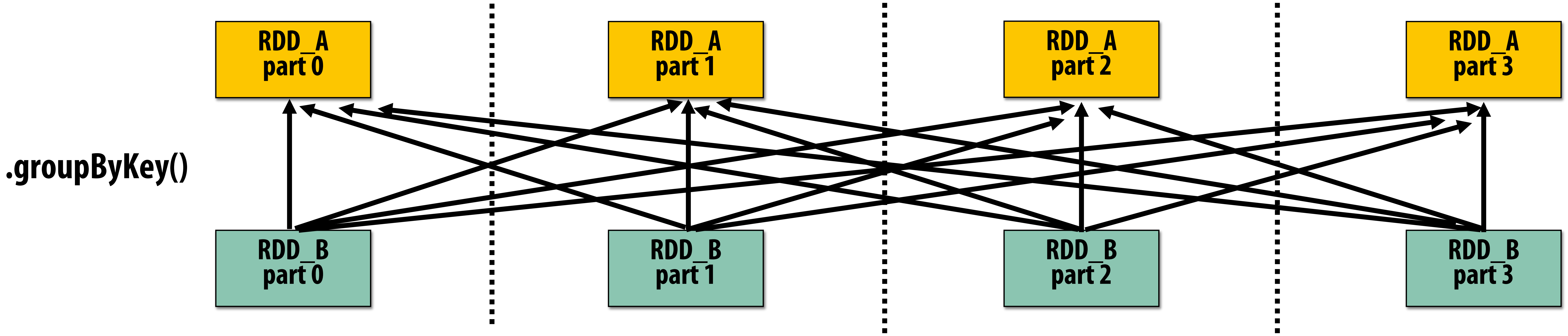
- Allows for fusing of operations (here: can apply map and then filter all at once on input element)
- In this example: no communication between nodes of cluster (communication of one int at end to perform count() reduction)



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

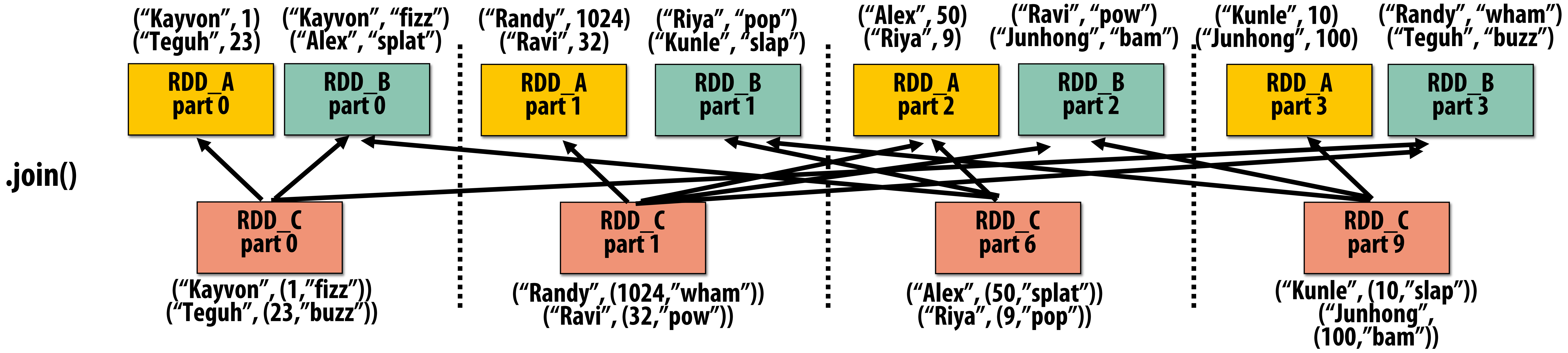
  (I will address resilience in a few slides)

Cost of operations depends on partitioning

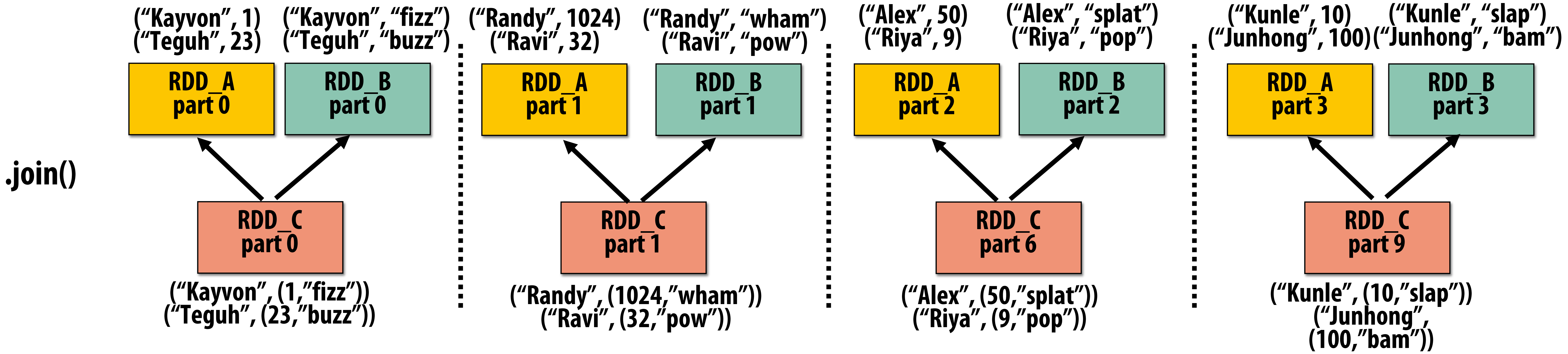
join: RDD[(K,V)], RDD[(K,W)] → RDD[(K,(V,W))]

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

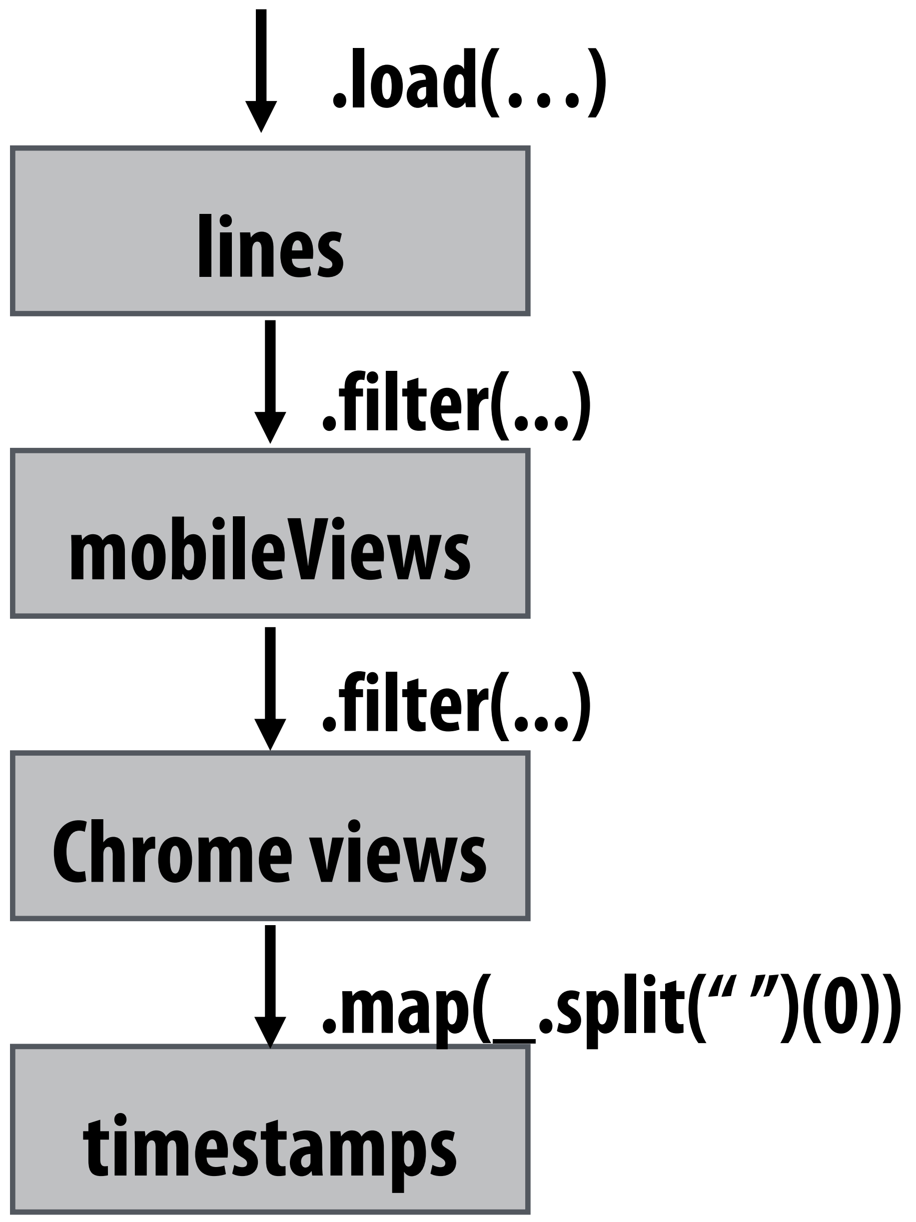
```
// create RDD from file system data
val lines = spark.textFile("hdfs://cs149log.txt");
val clientInfo = spark.textFile("hdfs://clientssupported.txt"); // (useragent, "yes"/"no")

// 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)

Implementing Resilience via Lineage

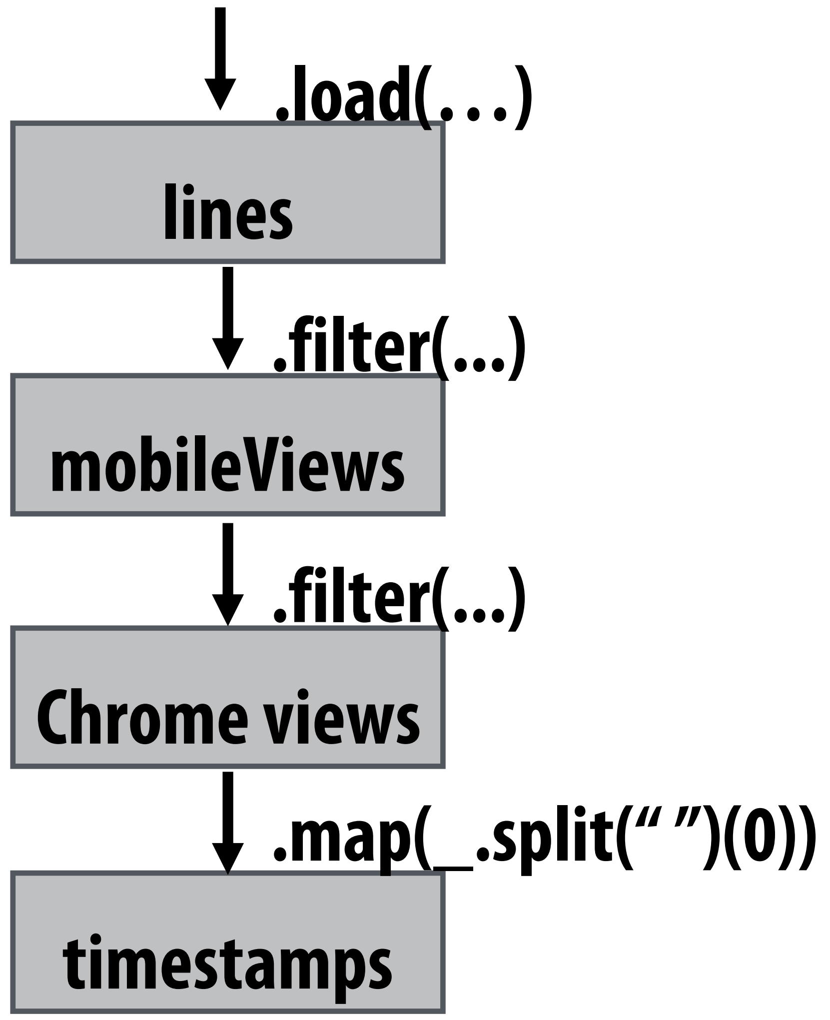
- **RDD transformations are bulk, deterministic, and functional**
  - Implication: runtime can always reconstruct contents of RDD from its lineage (the sequence of transformations used to create it)
  - Lineage is a log of transformations
  - Efficient: since the log records bulk data-parallel operations, 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));
```

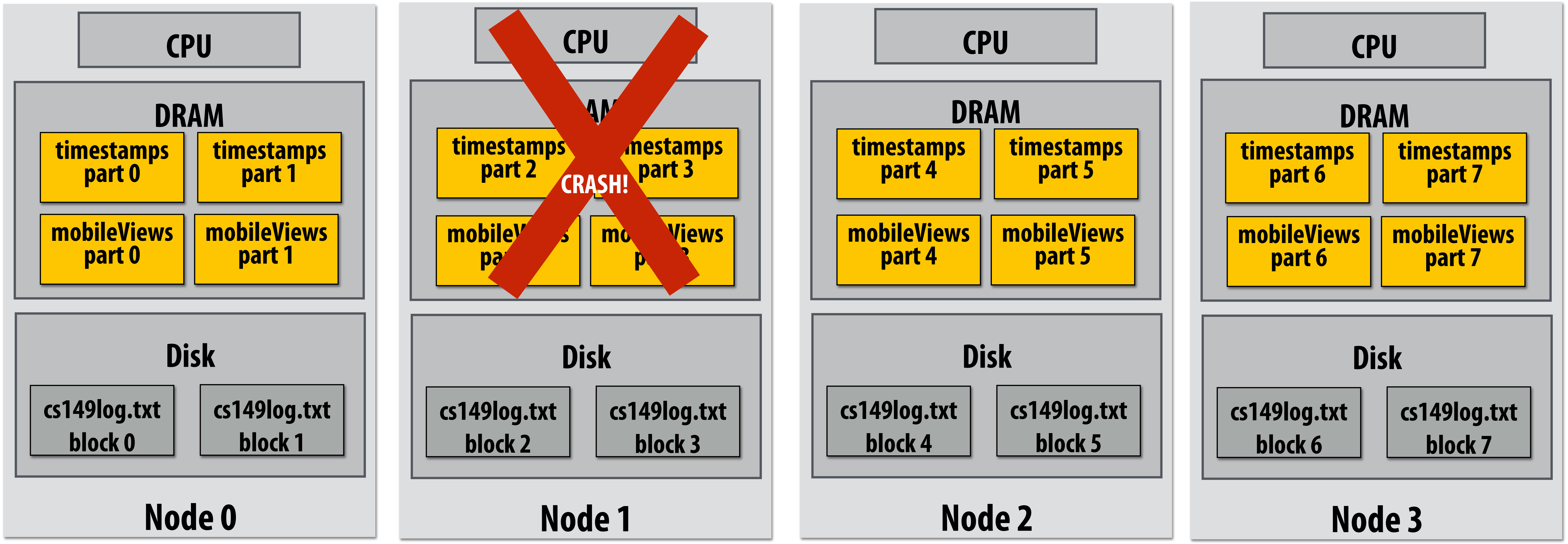


Upon Node Failure: Recompute Lost RDD Partitions from Lineage

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



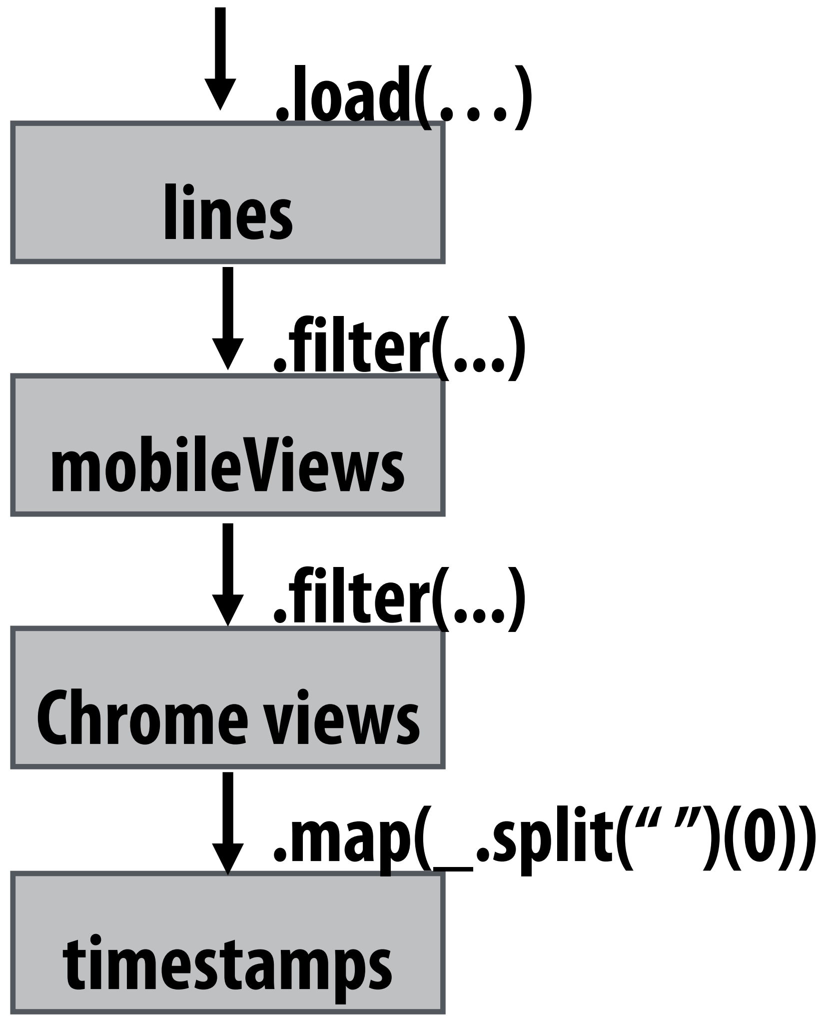
Must reload required subset of data from disk and recompute entire sequence of operations given by lineage to regenerate partitions 2 and 3 of RDD timestamps.



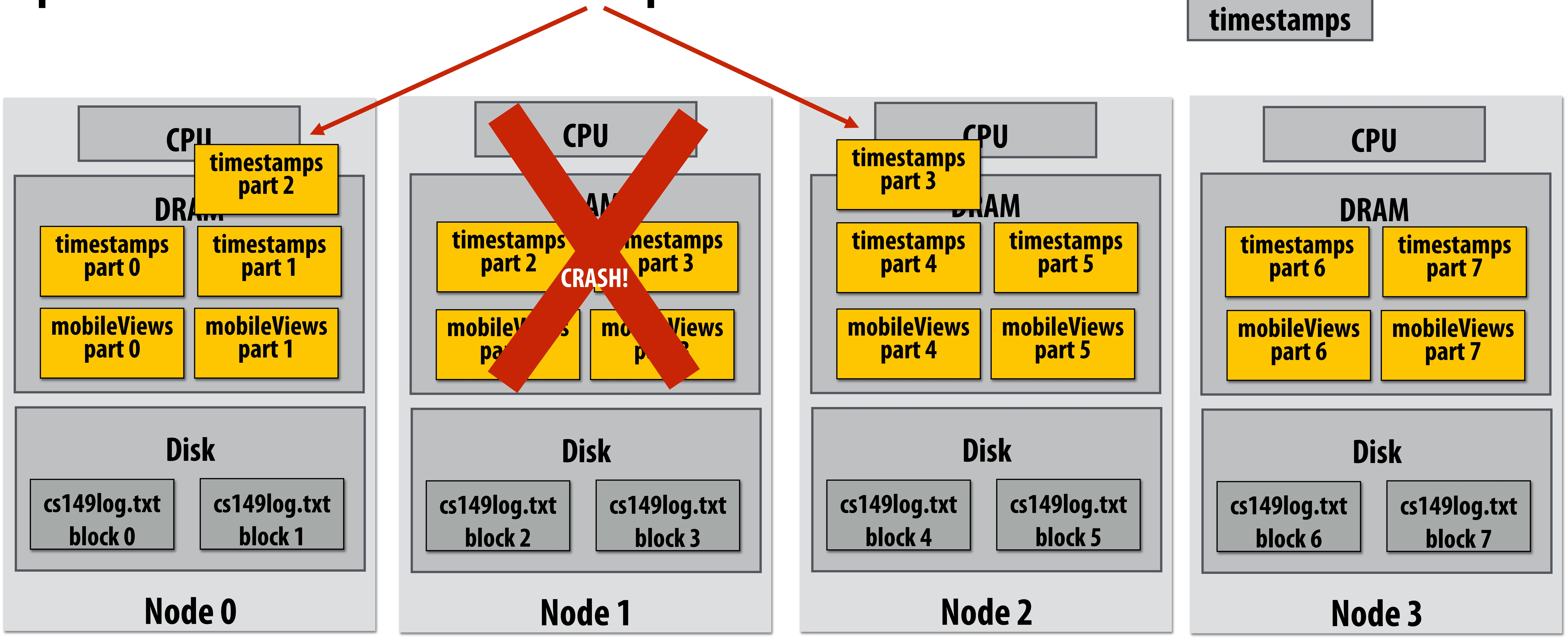
Note: (not shown): file system data is replicated so assume blocks 2 and 3 remain accessible to all nodes

Upon Node Failure: Recompute Lost RDD Partitions from Lineage

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

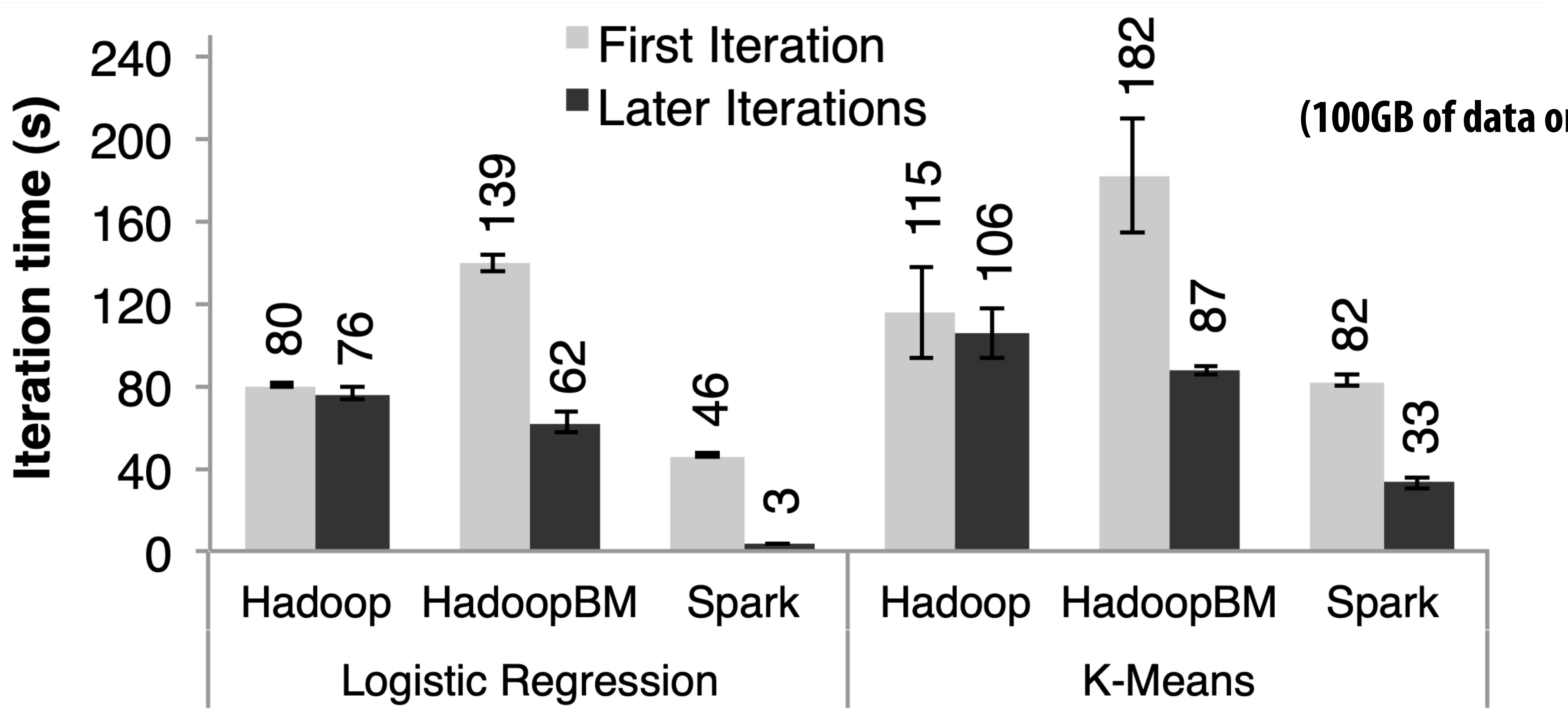


Must reload required subset of data from disk and recompute entire sequence of operations given by lineage to regenerate partitions 2 and 3 of RDD timestamps



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

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)

Anything else puzzling here?

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

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

Further optimization of the baseline brought time down to 110s

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

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

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

Machine learning library build on top of Spark abstractions.

Spark GraphX

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

GraphLab-like library built on top of Spark abstractions.