Google Cluster Computing Faculty Training Workshop

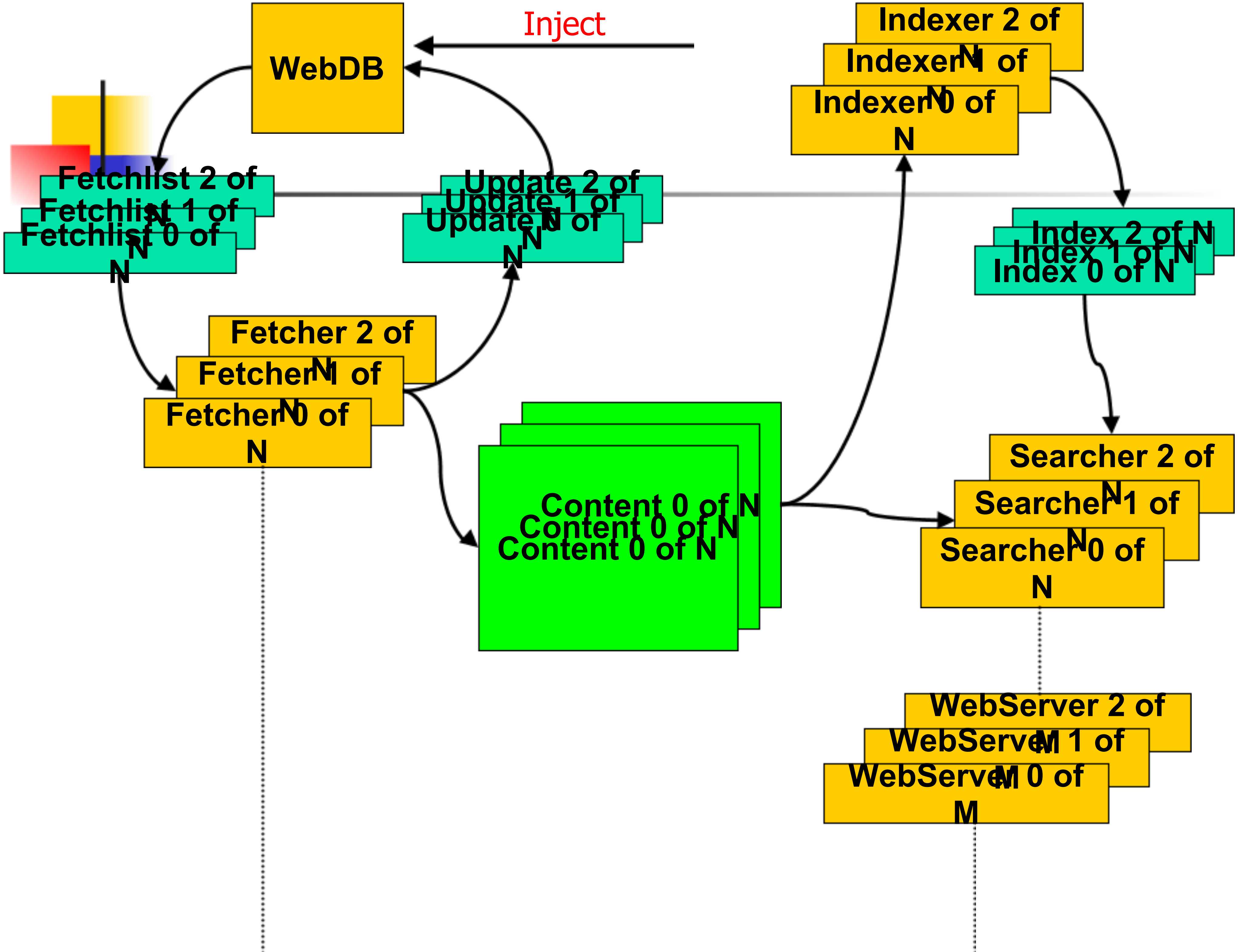
Module III: Nutch

Meta-details

- Built to encourage public search work
  - Open-source, w/pluggable modules
  - Cheap to run, both machines & admins
- Goal: Search more pages, with better quality, than any other engine
  - Pretty good ranking
  - Has done ~ 200M pages, more possible
- Hadoop is a spinoff

Outline

- Nutch design
  - Link database, fetcher, indexer, etc…
- Hadoop support
  - Distributed filesystem, job control


Inject
WebDB
Fetchlist 2 of N
Fetchlist 1 of N
Fetchlist 0 of N
Update 2 of N
Update 1 of N
Update 0 of N
Fetcher 2 of N
Fetcher 1 of N
Fetcher 0 of N
Content 0 of N
Content 0 of N
Content 0 of N
Indexer 2 of N
Indexer 1 of N
Indexer 0 of N
Index 2 of N
Index 1 of N
Index 0 of N
Searcher 2 of N
Searcher 1 of N
Searcher 0 of N
WebServer 2 of M
WebServer 1 of M
WebServer 0 of M

Moving Parts

- Acquisition cycle
  - WebDB
  - Fetcher
- Index generation
  - Indexing
  - Link analysis (maybe)
- Serving results

WebDB

- Contains info on all pages, links
  - URL, last download, # failures, link score, content hash, ref counting
  - Source hash, target URL
- Must always be consistent
- Designed to minimize disk seeks
  - 19ms seek time x 200m new pages/mo = ~44 days of disk seeks!
- Single-disk WebDB was huge headache

Fetcher

- Fetcher is very stupid. Not a "crawler"
- Pre-MapRed: divide "to-fetch list" into k pieces, one for each fetcher machine
- URLs for one domain go to same list, otherwise random
  - "Politeness" w/o inter-fetcher protocols
  - Can observe robots.txt similarly
  - Better DNS, robots caching
  - Easy parallelism
- Two outputs: pages, WebDB edits

WebDB/Fetcher Updates

URL: http://www.about.com/index.html
LastUpdated: 3/22/05
ContentHash: MD5_sdflkjweroiwelksd
URL: http://www.cnn.com/index.html
LastUpdated: Today!
ContentHash: MD5_balboglerropewolefbag
URL: http://www.yahoo/index.html
LastUpdated: 4/07/05
ContentHash: MD5_toewkekqmekkalekaa
URL: http://www.yahoo.com/index.html
LastUpdated: Today!
ContentHash: MD5_toewkekqmekkalekaa

Edit: DOWNLOAD_CONTENT
URL: http://www.yahoo/index.html
ContentHash: MD5_toewkekqmekkalekaa
Edit: DOWNLOAD_CONTENT
URL: http://www.cnn.com/index.html
ContentHash: MD5_balboglerropewolefbag
Edit: NEW_LINK
URL: http://www.flickr.com/index.html
ContentHash: None

Fetcher edits

[illegible] new database

Indexing

- Iterate through all k page sets in parallel, constructing inverted index
- Creates a “searchable document” of:
  - URL text
  - Content text
  - Incoming anchor text
- Other content types might have a different document fields
  - Eg, email has sender/receiver
  - Any searchable field end-user will want
- Uses Lucene text indexer

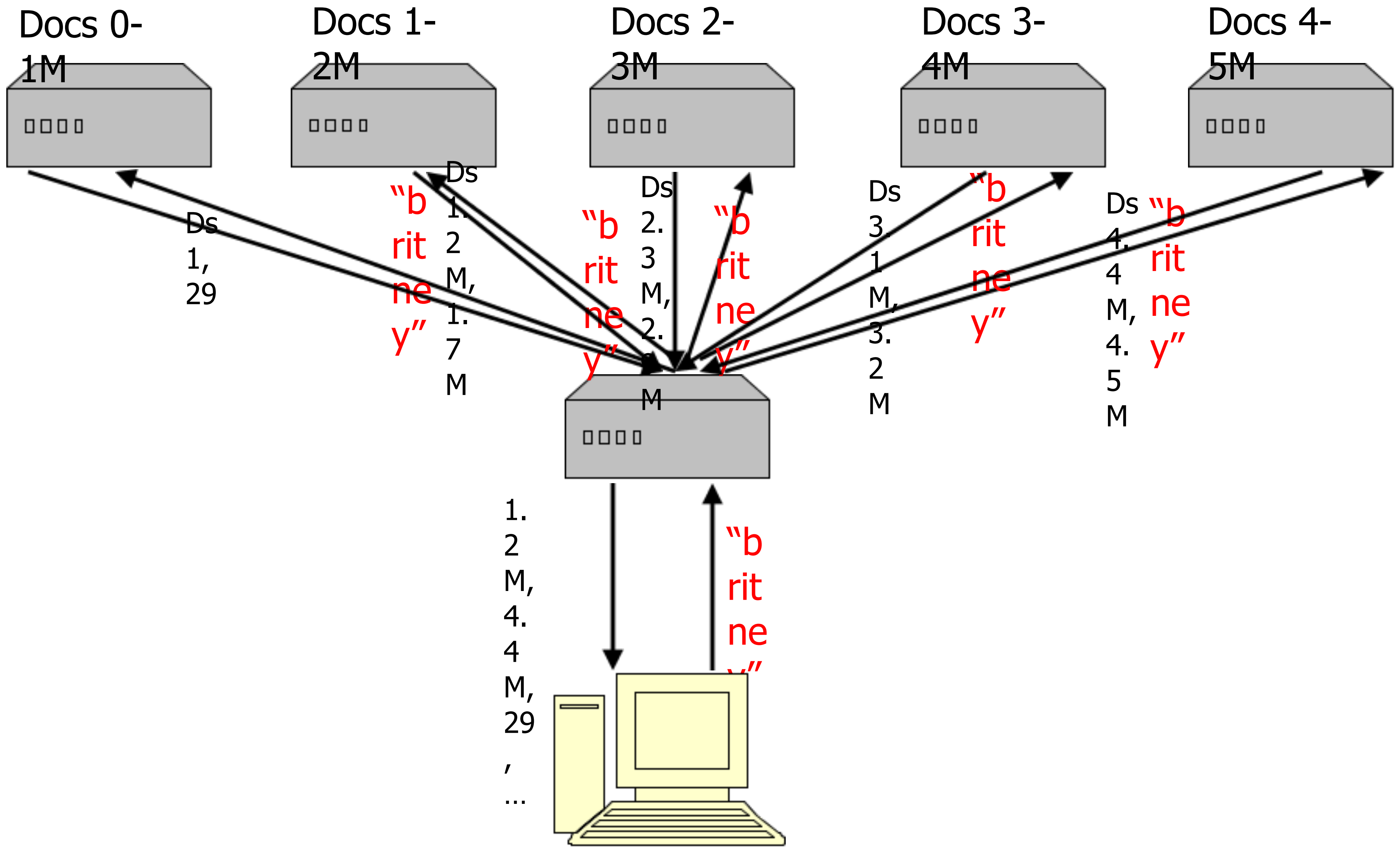
Link analysis

- A page's relevance depends on both intrinsic and extrinsic factors
  - Intrinsic: page title, URL, text
  - Extrinsic: anchor text, **link graph**
- PageRank is most famous of many
- Others include:
  - HITS
  - OPIC
  - Simple incoming link count
- Link analysis is sexy, but importance generally overstated

Link analysis (2)

- Nutch performs analysis in WebDB
  - Emit a score for each known page
  - At index time, incorporate score into inverted index
- Extremely time-consuming
  - In our case, disk-consuming, too (because we want to use low-memory machines)
- Fast and easy:
  - 0.5 * log(# incoming links)

Query Processing



Administering Nutch

- Admin costs are critical
  - It’s a hassle when you have 25 machines
  - Google has >100k, probably more
- Files
  - WebDB content, working files
  - Fetchlists, fetched pages
  - Link analysis outputs, working files
  - Inverted indices
- Jobs
  - Emit fetchlists, fetch, update WebDB
  - Run link analysis
  - Build inverted indices

Administering Nutch (2)

- Admin sounds boring, but it's not!
  - Really
  - I swear
- Large-file maintenance
  - Google File System (Ghemawat, Gobioff, Leung)
  - Nutch Distributed File System
- Job Control
  - Map/Reduce (Dean and Ghemawat)
  - Pig (Yahoo Research)
- Data Storage (BigTable)

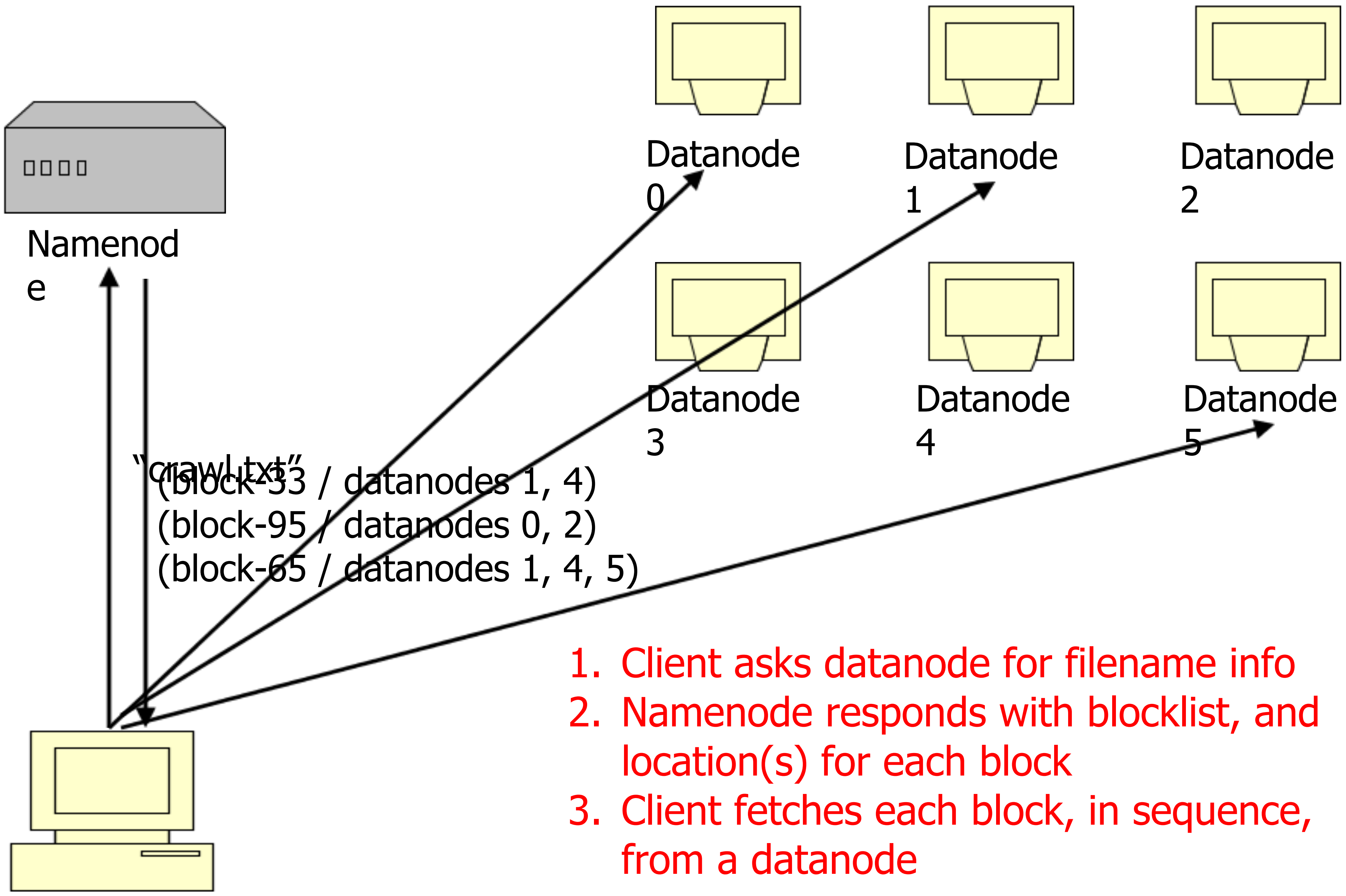
Nutch Distributed File System

- Similar, but not identical, to GFS
- Requirements are fairly strange
  - Extremely large files
  - Most files read once, from start to end
  - Low admin costs per GB
- Equally strange design
  - Write-once, with delete
  - Single file can exist across many machines
  - Wholly automatic failure recovery

NDFS (2)

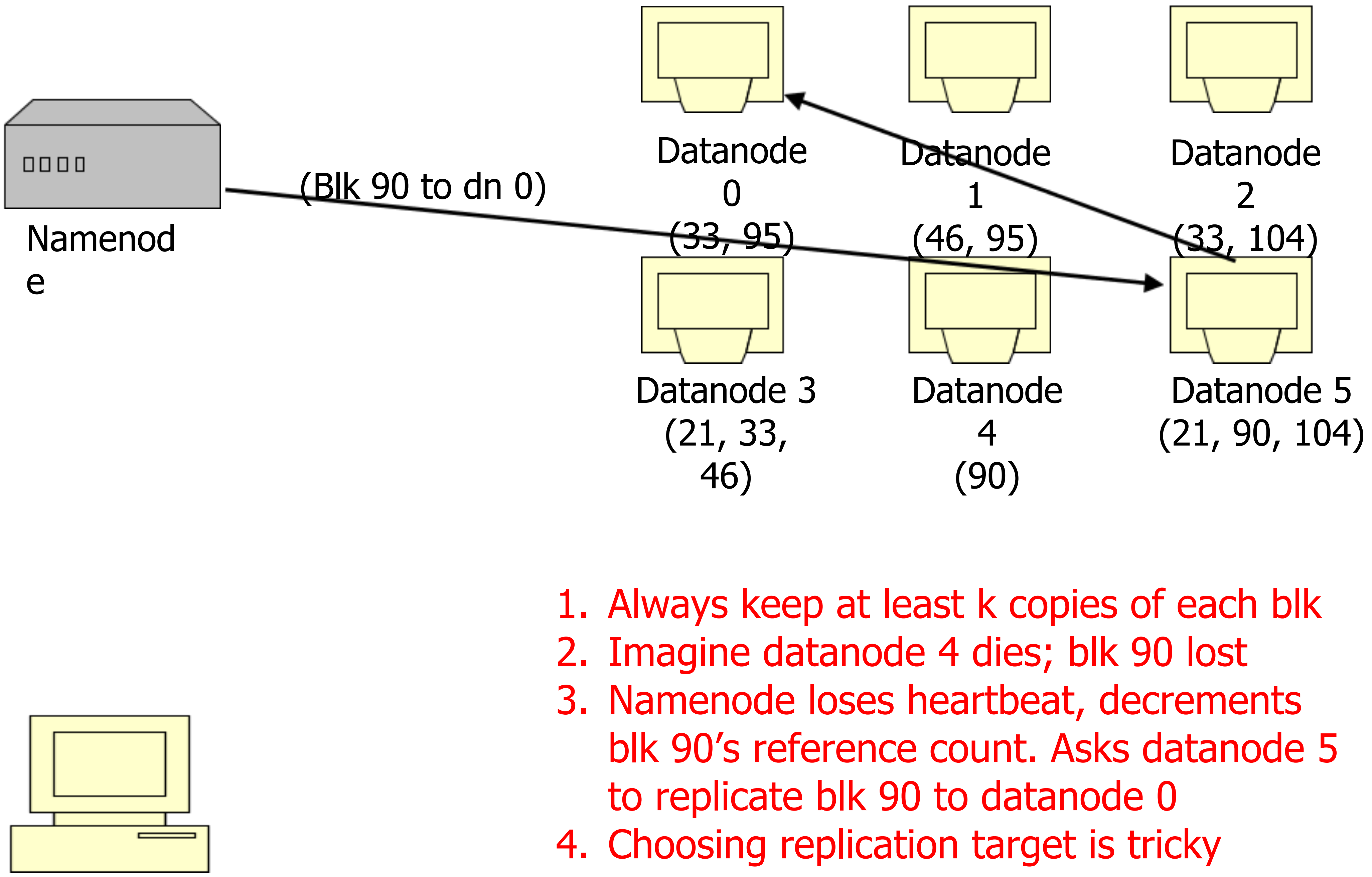
- Data divided into blocks
- Blocks can be copied, replicated
- Datanodes hold and serve blocks
- Namenode holds metainfo
  - Filename • block list
  - Block • datanode-location
- Datanodes report in to namenode every few seconds

NDFS File Read


Datanode 0
Datanode 1
Datanode 2
Datanode 3
Datanode 4
Datanode 5
Namenode
"crawl.txt"
(block-33 / datanodes 1, 4)
(block-95 / datanodes 0, 2)
(block-65 / datanodes 1, 4, 5)

1. Client asks datanode for filename info
2. Namenode responds with blocklist, and location(s) for each block
3. Client fetches each block, in sequence, from a datanode

NDFS Replication



1. Always keep at least k copies of each blk
2. Imagine datanode 4 dies; blk 90 lost
3. Namenode loses heartbeat, decrements blk 90’s reference count. Asks datanode 5 to replicate blk 90 to datanode 0
4. Choosing replication target is tricky

Map/Reduce

- Map/Reduce is programming model from Lisp (and other places)
  - Easy to distribute across nodes
  - Nice retry/failure semantics
- map(key, val) is run on each item in set
  - emits key/val pairs
- reduce(key, vals) is run for each unique key emitted by map()
  - emits final output

Map/Reduce (2)

- Task: count words in docs
  - Input consists of (url, contents) pairs
  - map(key=url, val=contents):
    - For each word w in contents, emit (w, “1”)
  - reduce(key=word, values=uniq_counts):
    - Sum all “1”s in values list
    - Emit result “(word, sum)”

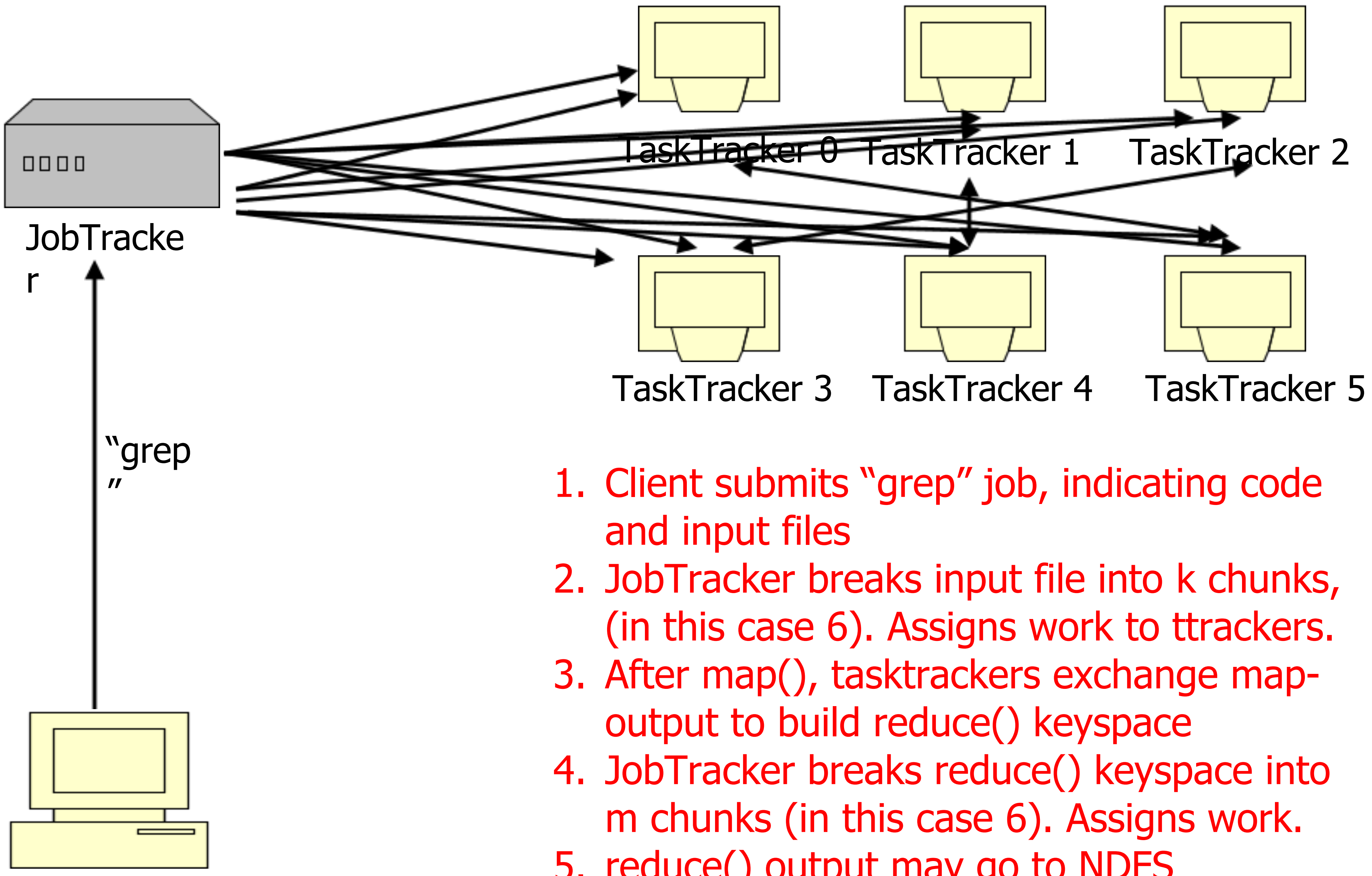
Map/Reduce (3)

- Task: grep
  - Input consists of (url+offset, single line)
  - map(key=url+offset, val=line):
    - If contents matches regexp, emit (line, "1")
  - reduce(key=line, values=uniq_counts):
    - Don't do anything; just emit line
- We can also do graph inversion, link analysis, WebDB updates, etc

Map/Reduce (4)

- How is this distributed?
  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
- If map() or reduce() fails, reexecute!

Map/Reduce Job Processing



1. Client submits "grep" job, indicating code and input files
2. JobTracker breaks input file into k chunks, (in this case 6). Assigns work to ttrackers.
3. After map(), tasktrackers exchange map-output to build reduce() keyspace
4. JobTracker breaks reduce() keyspace into m chunks (in this case 6). Assigns work.
5. reduce() output may go to NDFS

Nutch & Hadoop

- NDFS stores the crawl and indexes
- MapReduce for indexing, parsing, WebDB construction, even fetching
  - Broke previous 200M/mo limit
  - Index-serving?
- Required massive rewrite of almost every Nutch component

Conclusion

- http://www.nutch.org/
  - Partial documentation
  - Source code
  - Developer discussion board
- "Lucene in Action" by Hatcher, Gospodnetic (or you can borrow mine)
- Read the Google papers on GFS, MapReduce, and BigTable; we relied heavily on these papers.
- Questions?