Chapter 15:
Distributed Systems in Use

Distributed Systems in Use

- Objectives
  - To understand the difference between “Distributed Computing” and “Distributed Multimedia Systems”
  - To selectively discuss two example systems in use, one each
- Topics
  - Commodity cluster and its requirements as well as example
  - Distribution and parallelization of tasks and data
  - Programming abstraction for seamless distribution
    - MapReduce and an example system: Hadoop and HDFS
  - Distributed multimedia systems
  - Multimedia system stream characteristics
    - Quality-of-Service and its management
  - Data stream adaptations and real-time approaches

Motivation

- Distributed computing helps to scale up applications to handle more data at low costs
  - Low hardware costs, low engineering costs
- Need for many "cheap" machines
  - Fixes problem of "speed"
  - Reliability problems remain
    - Within large clusters, computers fail every day
  - Coordination of (heterogeneous) machines

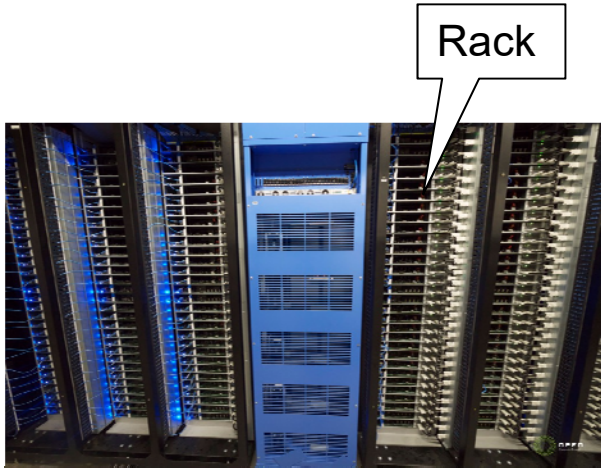
- Need common (standardized) infrastructure/framework
  - Must be efficient, easy-to-use/program, reliable
  - Must include control and management functionality

A Practical Problem at Hand

- The Web
  - 20+ billion Web pages
  - With an on average size of 20 kByte = 400+ TByte
- The Computer
  - 1 machine reads 100 MByte/s from Hard Disk (HD)
  - ⇨ ~1 month to “read” the Web
- Computers and HDs fail
  - Failure assumption of a machine app. every 1,000 days
    - Google’s data centers with app. 1,000,000 machines (2011)
      - ⇨ 1000 machines are down every day

A Solution Option

- Commodity clusters
  - Distribute data and tasks automatically
- Requirements
  - “Really” large data storage
  - A “really” large database
  - A programming model and execution environment
  - Distributed coordination services
  - Plug-in scalability
    - Just put in another “main board”/hard drive
  - Efficient operation (daily costs)
  - Simple maintenance
    - Without interruption of services (hot standby)



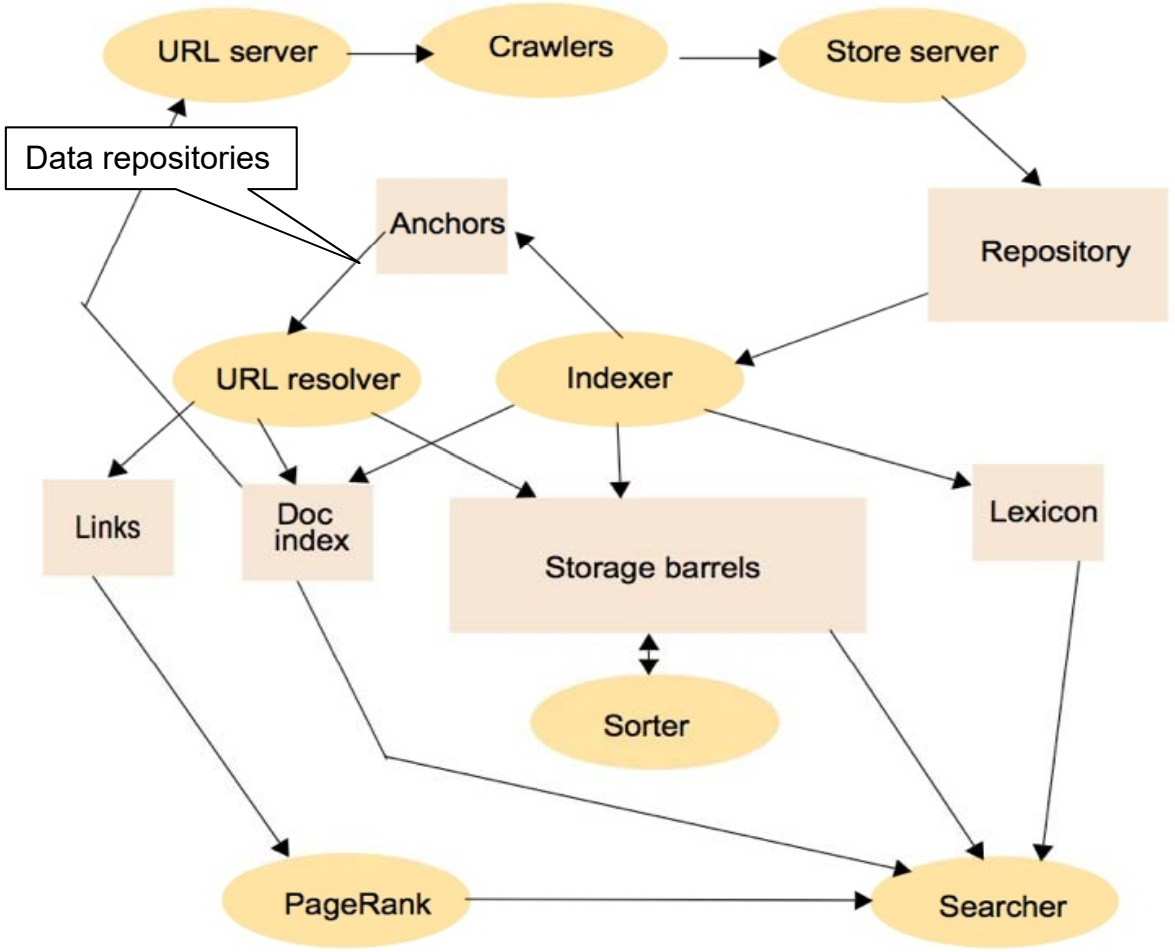
© 2021 UZH, CSG@IfI

A Solution Example

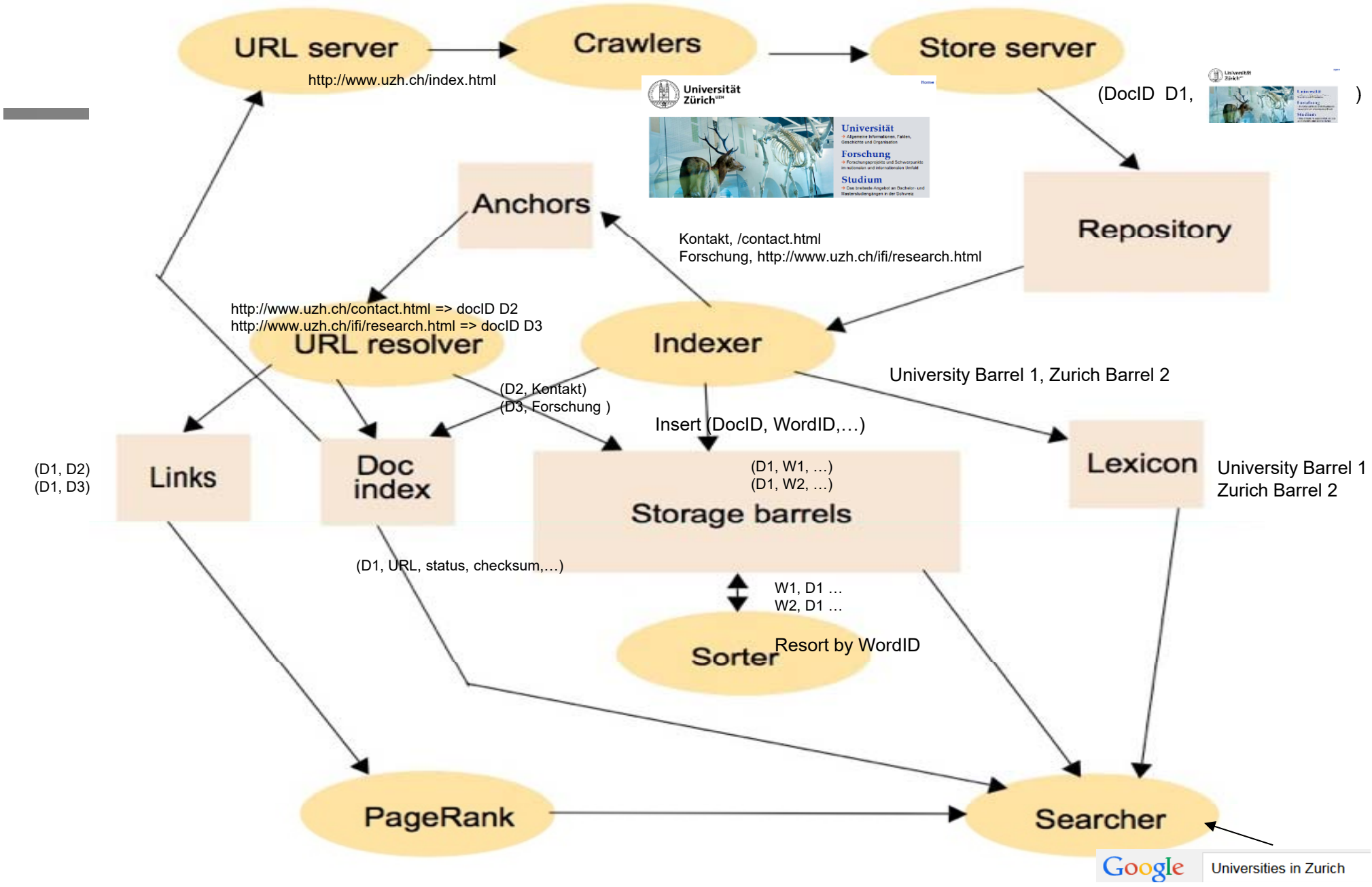
- Google's original architecture of its search engine

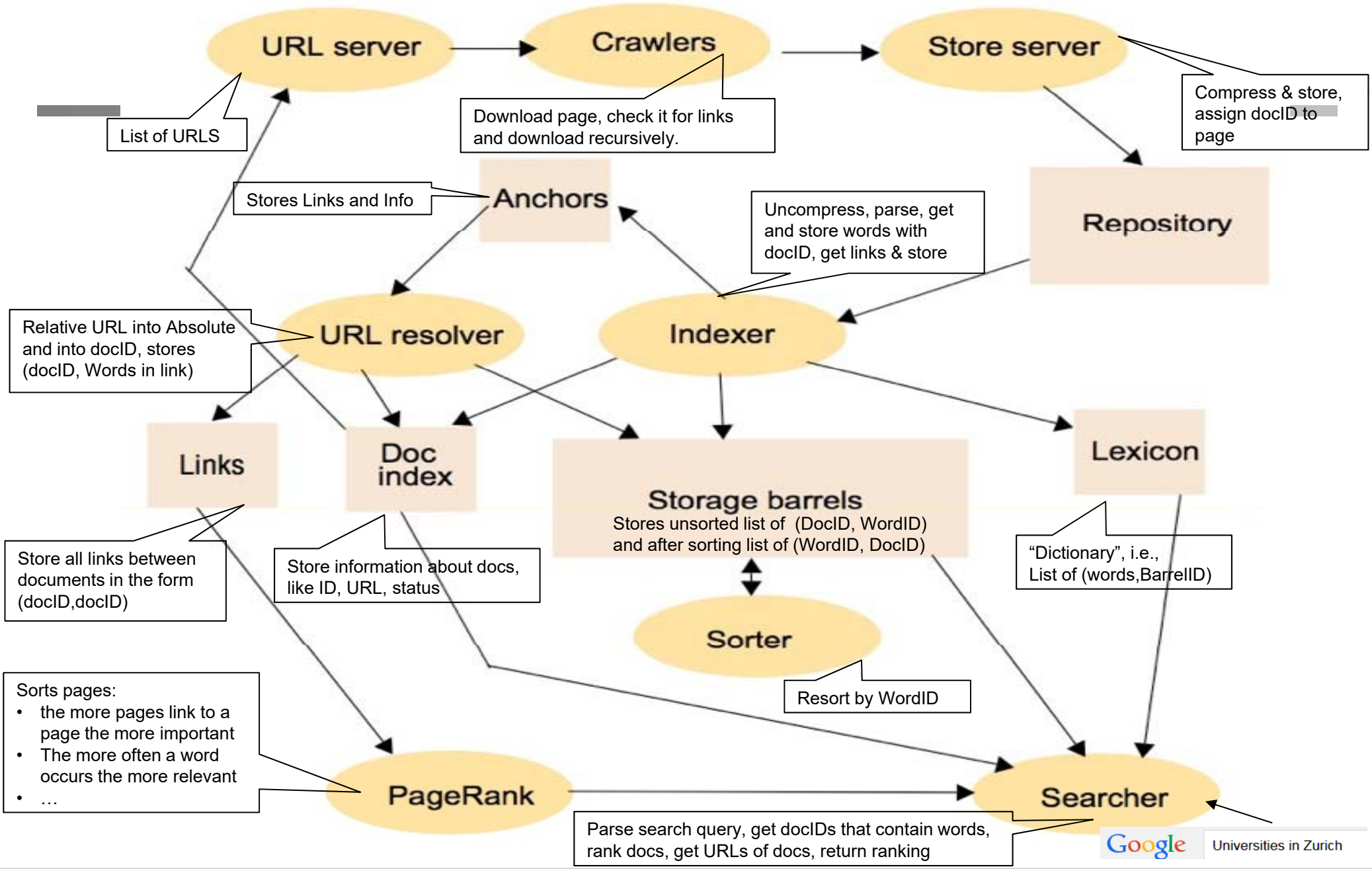
Updating "Google's data" with data from the Web

Handling requests from clients



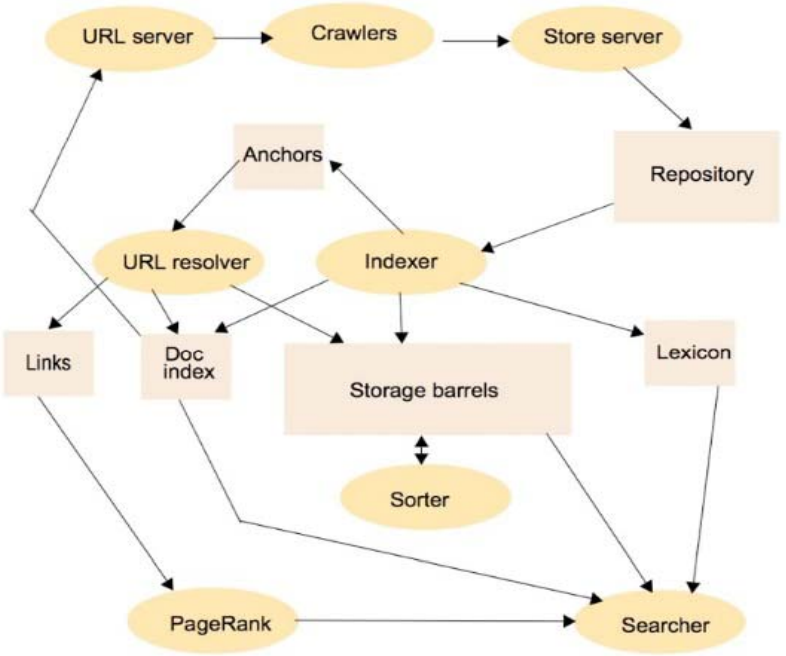
http://infolab.stanford.edu/~backrub/google.html


URL server
Crawlers
Store server
http://www.uzh.ch/index.html
(DocID D1,
)
Universität Zürich
Universität
Forschung
Studium
Anchors
Repository
Kontakt, /contact.html
Forschung, http://www.uzh.ch/ifi/research.html
http://www.uzh.ch/contact.html => docID D2
http://www.uzh.ch/ifi/research.html => docID D3
URL resolver
Indexer
University Barrel 1, Zurich Barrel 2
(D2, Kontakt)
(D3, Forschung)
Insert (DocID, WordID,…)
(D1, D2)
(D1, D3)
Links
Doc index
(D1, W1, …)
(D1, W2, …)
Storage barrels
Lexicon
University Barrel 1
Zurich Barrel 2
(D1, URL, status, checksum,…)
W1, D1 …
W2, D1 …
Sorter
Resort by WordID
PageRank
Searcher
Google
Universities in Zurich


URL server
Crawlers
Store server
List of URLS
Download page, check it for links and download recursively.
Compress & store, assign docID to page
Stores Links and Info
Anchors
Uncompress, parse, get and store words with docID, get links & store
Repository
Relative URL into Absolute and into docID, stores (docID, Words in link)
URL resolver
Indexer
Links
Doc index
Storage barrels
Stores unsorted list of (DocID, WordID) and after sorting list of (WordID, DocID)
Lexicon
Store all links between documents in the form (docID,docID)
Store information about docs, like ID, URL, status
"Dictionary", i.e., List of (words,BarrelID)
Sorter
Resort by WordID
Sorts pages:
• the more pages link to a page the more important
• The more often a word occurs the more relevant
• …
PageRank
Searcher
Parse search query, get docIDs that contain words, rank docs, get URLs of docs, return ranking
Google
Universities in Zurich

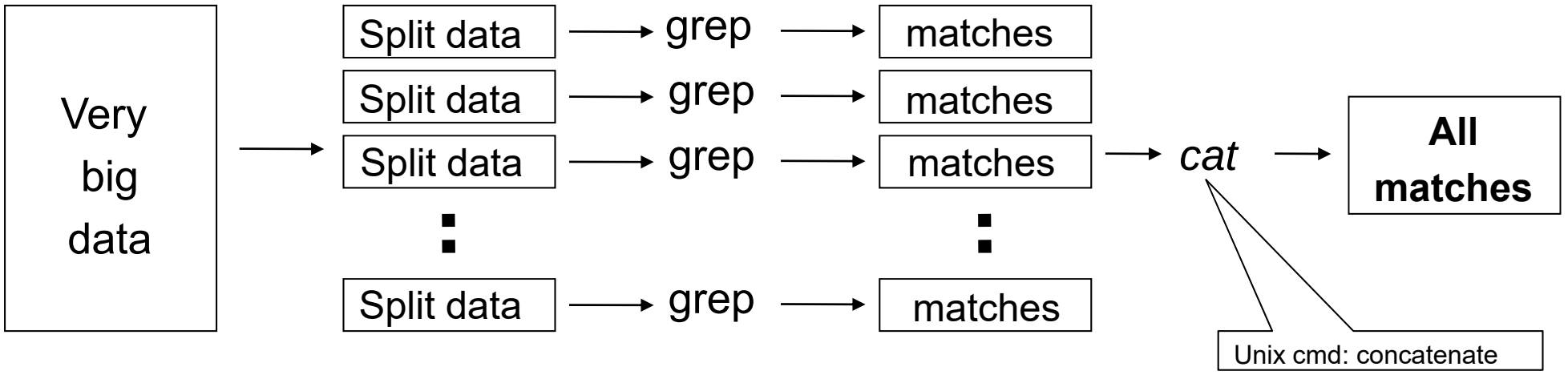
Distribution and Parallelization of Tasks

- Task parallelism
  - Each “oval” is a server
    - *E.g.*, URL server, Crawler, Indexer
- Data parallelism
  - Distribution of tasks and data
    - *E.g.*, Indexer
      - (Full) documents distributed among several servers for parsing
    - *E.g.,* each server only parses parts of a document

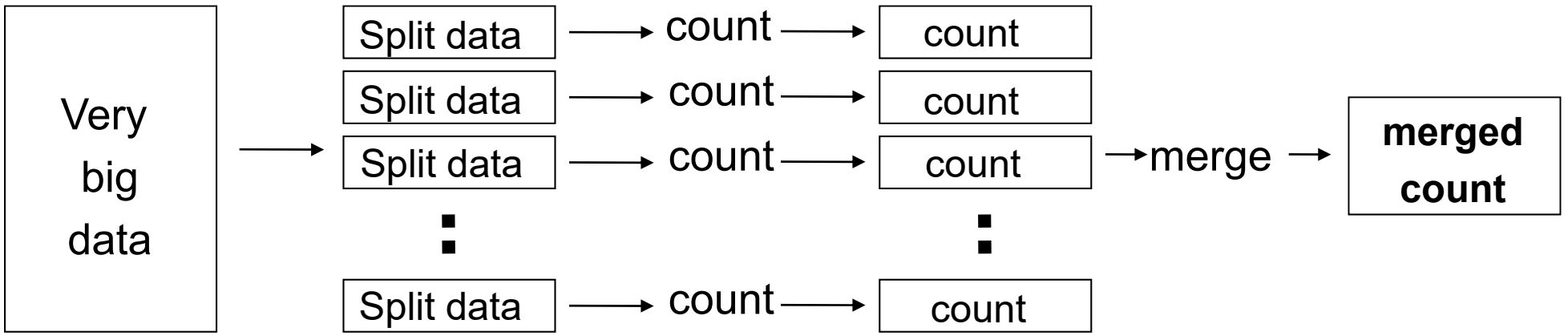


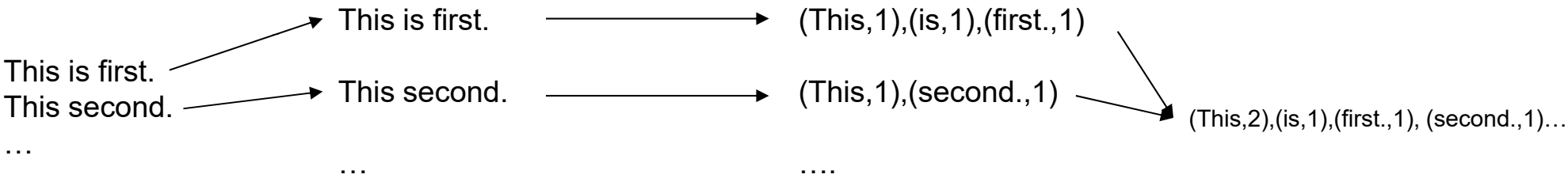
- General abstraction needs to
  - … be simple
  - … allow for automatic distribution of (data and) tasks (in a cluster)
  - … be platform-agnostic

Distributed Grep (Search)



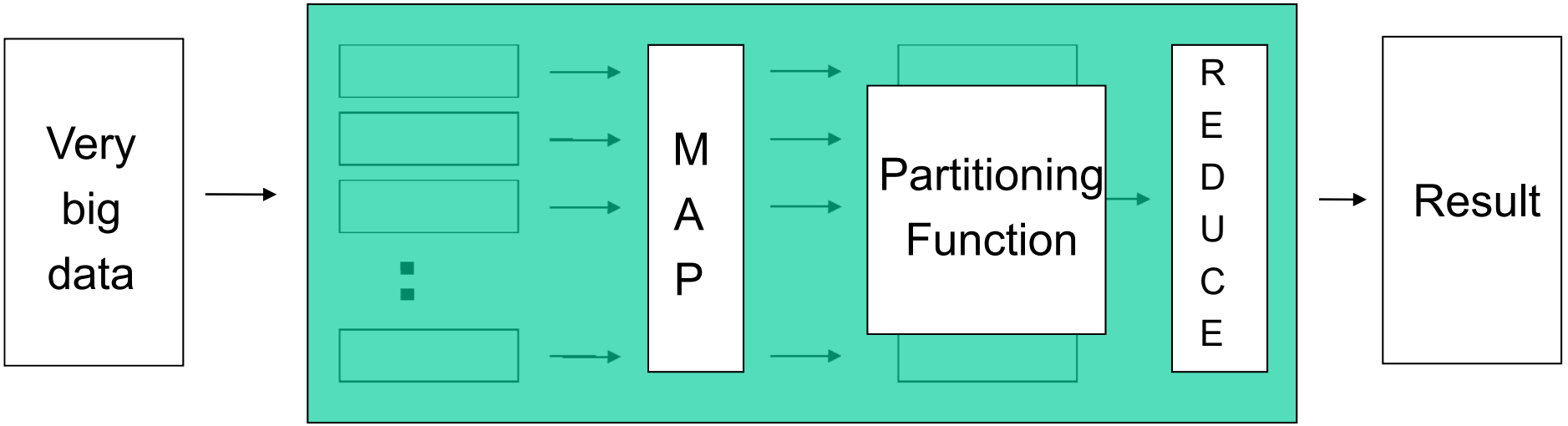
Distributed Word Count





Bulk Synchronous Parallel

- A programming abstraction for synchronous cluster computing: MapReduce

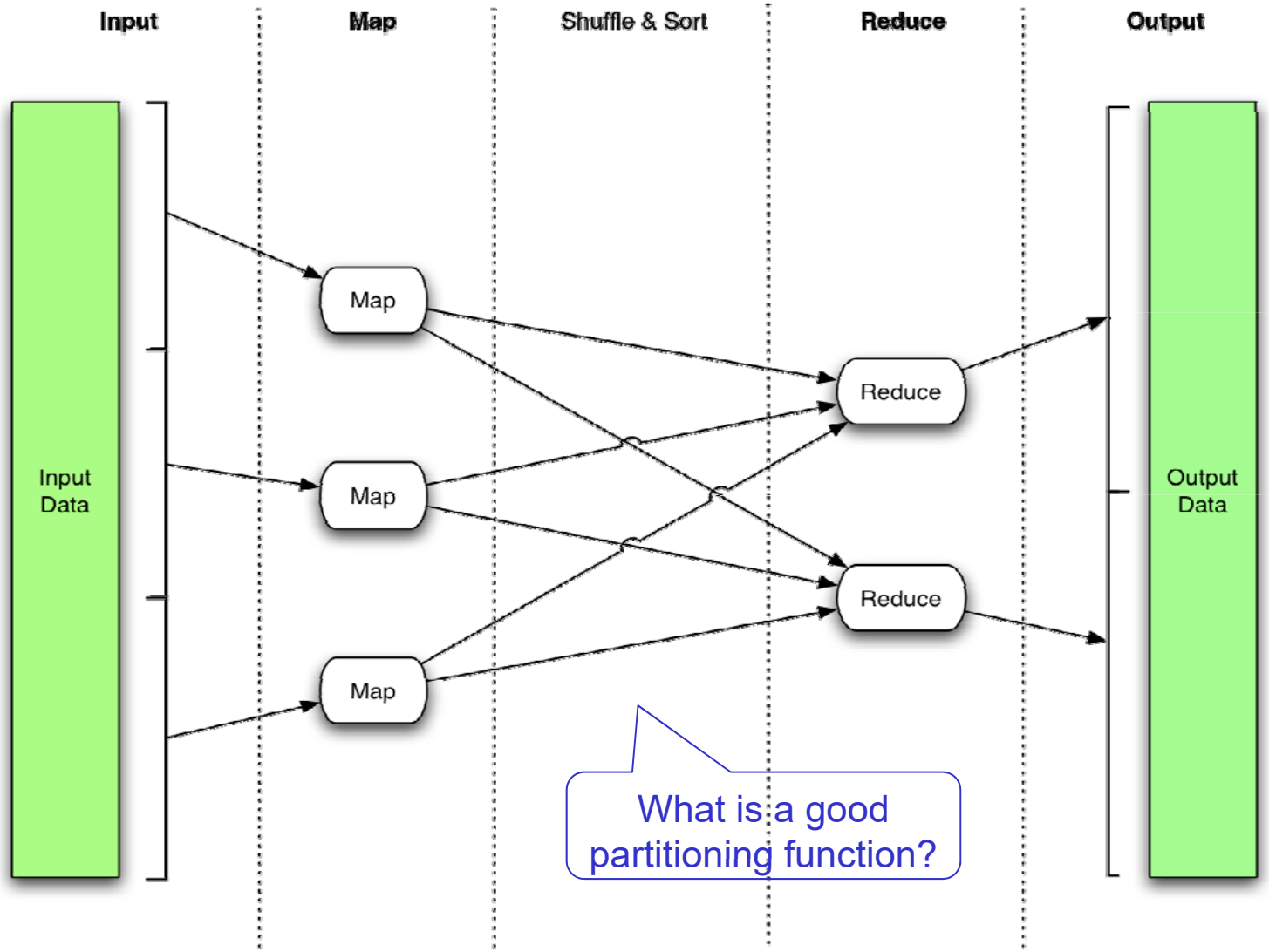


- Map: (a) accept input key/value pair and (b) emit intermediate key/value pair
- Reduce: (a) accept intermediate key/value* pair and (b) emit output key/value pair

MapReduce Characteristics

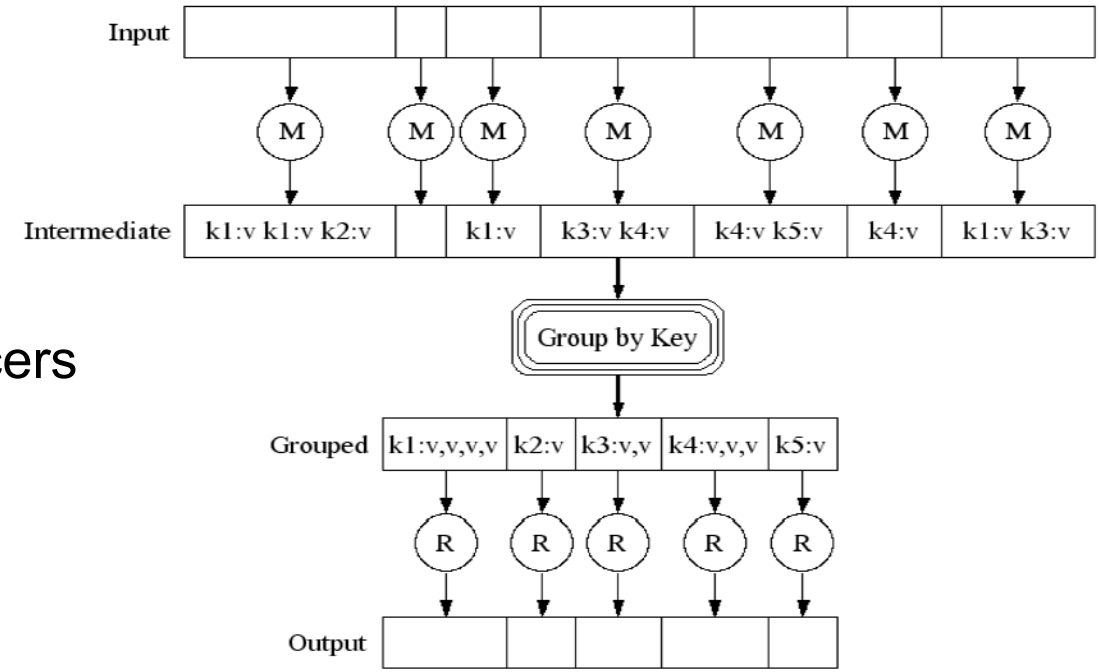
- MapReduce is a programming model for efficient distributed computing
- Works like a Unix pipeline
  - cat input | grep | sort | uniq -c | cat > output
  - Input | Map | Shuffle & Sort | Reduce | Output
- Efficiency from
  - Streaming through data and reducing seeks
  - Pipelining
- A good fit for many applications, such as
  - Log data processing
  - Web index building

MapReduce Data Flow



Partitioning Function (1)

- Each Map function's output allocated to a particular reducer
  - Sharding sets up "communities", groups
  - Inputs
    - Key and number of reducers
  - Output
    - Index of desired reducer



- Default approach
  - Hashing the key: `hash(key) mod R`
    - Using hash value modulo number of reducers
      - Gives R partitions and relatively well-balanced partitions

MapReduce Example Operations (1)

- Distributed grep for counting number of pattern occurrences
  - Map

    ```
    if match(value,pattern) emit(value,1)
    ```
  - Reduce

    ```
    emit(key,sum(value*))
    ```

- Distributed word count
  - Map

    ```
    for all w in value do emit(w,1)
    ```
  - Reduce

    ```
    emit(key,sum(value*))
    ```

MapReduce Example Operations (2)

- Sorting
  - For each partition given to a reducer
    - Ordering guarantee within a partition
  - Can be slower than map tasks

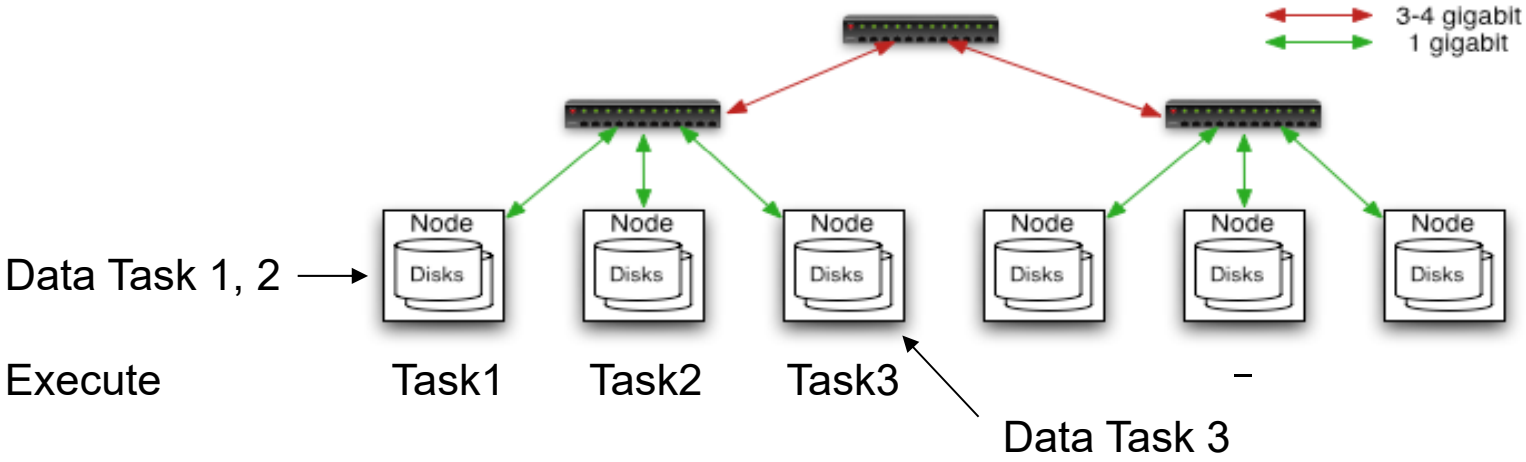
- Distributed Sort
  - Map

    ```
    emit(key,value)
    ```
  - Reduce (with R=1)

    ```
    emit(key,value)
    ```

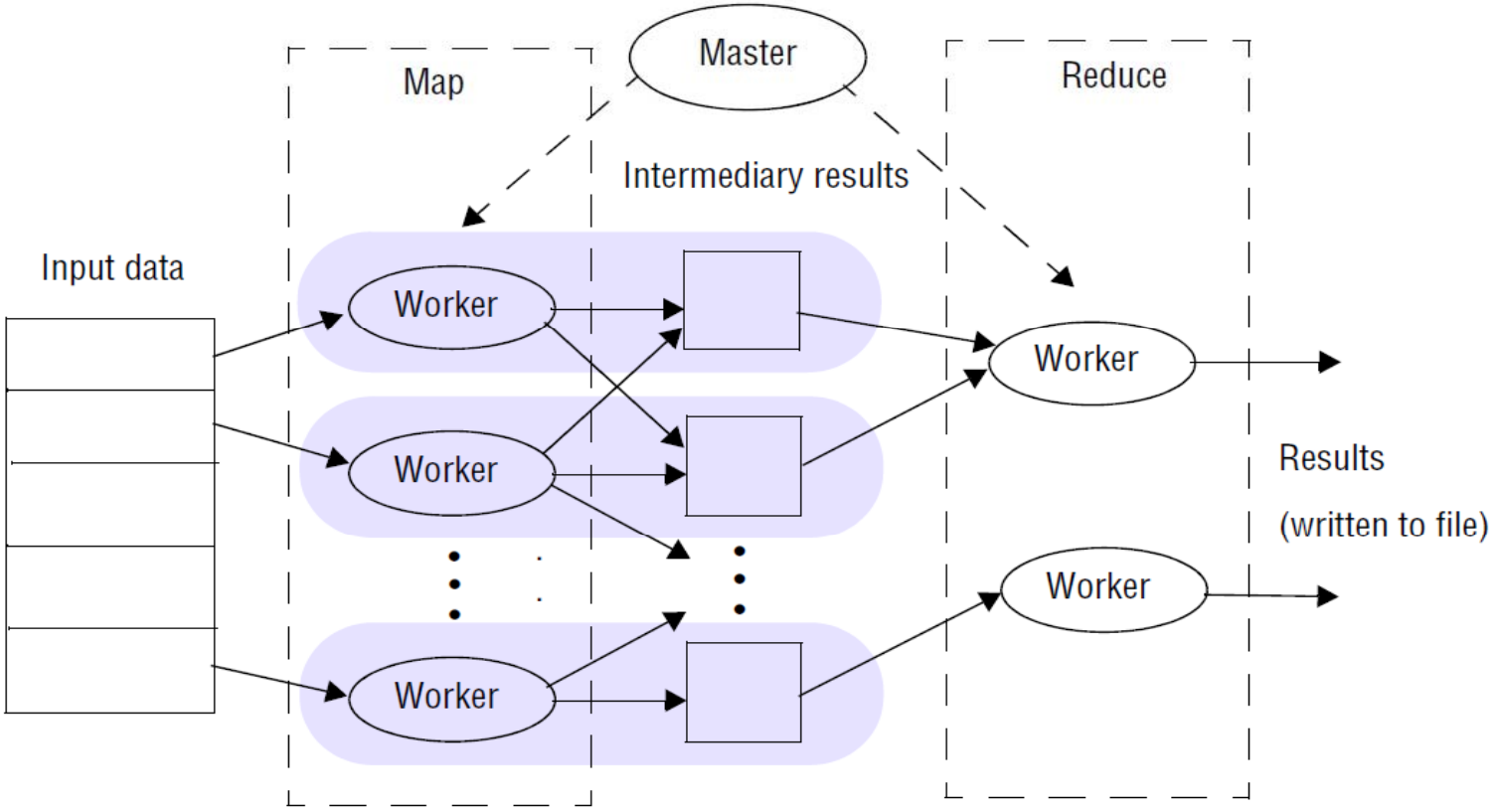
MapReduce Features

- Java and C++ APIs
- Each task can process data larger than RAM size
- Automatic re-execution on failure
- Locality optimizations
  - MapReduce queries file system for locations of input data
  - Map tasks are scheduled close to the inputs, when possible



Automatic Re-execution on Failure

- Master assigns tasks and monitors progress
  - Master “pings” workers and re-schedules tasks failing



Optimizations

- Bottlenecks (mostly) with bandwidth
  - Compress data for transmission
    - Standard compression, such as zip
    - Combiners as “Mini-reduce” on local mapper output
    - Example word count
      - Naïve; mapper node: `emits (WordA,1), (WordA,1)`
      - Smarter; count words on output: `emit (WordA, 2)`
  - Some workers slower than others
    - *E.g.*, due to “bad” disks doing error correction
  - Some tasks are slow, called “stragglers”
    - If program close to finish then
      - Schedule (remaining) tasks multiple times
    - Take fastest response
      - If tasks takes longer than expected, schedule again

MapReduce Overall Evaluation

- Suitable for
  - … a cluster or willingness to use cloud service
  - … working with large datasets
  - … working with independent data (or assumed to be)
  - … splitting work into independent tasks
- Always a cast into map and reduce possible

- Unsuitable for
  - … dependent data sets
  - … control algorithms with many iterations
  - … non-homogenous tasks
  - … *e.g.*, automation systems or control networks

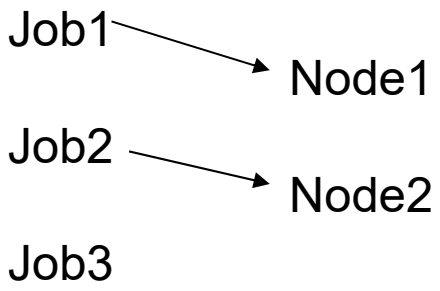
MapReduce System Example

- (Apache) Hadoop
  - Open source framework in Java
    - Distributed File System – “distributes” data
      - HDFS (Hadoop Distributed File System)
    - Map/Reduce – “distributes” application
  - Runs on
    - Linux, Mac OS/X, Windows, and Solaris
    - Commodity hardware
  - Hadoop MapReduce architecture is master/slave

	Master	Slave
MapReduce	jobtracker	tasktracker
HDFS	namenode	datanode

Selected Hadoop Operational Details

- Reduce or maps per jobs (per node)
  - Part of job configuration
    - Can use defaults
  - Recommendation
    - Reduce: 0.95 to 1.75 jobs per node
      - 0.95: All reducer jobs start in parallel after map
      - 1.75: Fast nodes finish and launch second
        - Recommended for load balancing, but more overhead
    - Map: 10 – 100 jobs per node



- Commodity Hardware Cluster
  - Typically in 2 level architecture, nodes are commodity PCs
  - 40 nodes/rack
    - Uplink from rack is at about 10 Gbit/s, rack-internal speed app. 1 Gbit/s

Hadoop Distributed File System (HDFS)

- Single namespace for entire cluster
  - Managed by a single “namenode”
  - Files are single-writer and append-only
  - Optimized for streaming reads of large files

Very similar to GFS

- Files are broken into large blocks
  - Typically 128 MByte
  - Replicated to several “datanodes” for reliability purposes
- Client talks to both “namenode” and “datanodes”
  - Data is never sent through the “namenode”
  - Throughput of file system scales nearly linearly with #nodes
- Access from Java, C, or command line

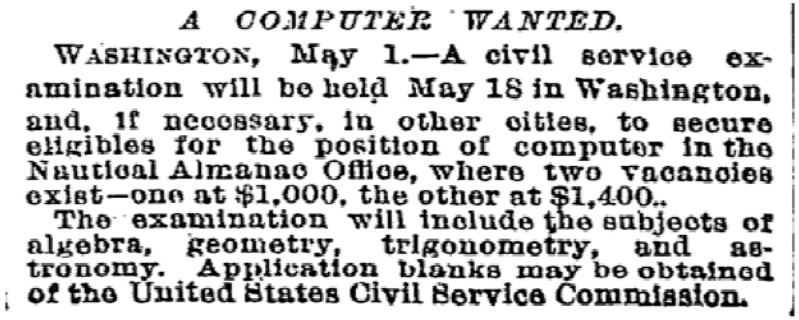
HDFS Reliability

- Data is checked with CRC-32
- File Creation
  - Client computes checksum per 512 Byte
  - “datanode” stores the checksum
- File access
  - Client retrieves data and checksum from “datanode”
- If validation fails
  - Client tries other replicas
  - Periodic Validation

CRC: Cyclic Redundancy Code

Hadoop Example Applications

- Offline conversion of data
  - Public domain articles from 1851-1922 of the New York Times
  - Hadoop to convert scanned images to PDF
    - Ran 100 Amazon EC2 instances for approximately 24 hours
    - 4 TByte of input
    - 1.5 TByte of output



A COMPUTER WANTED.

WASHINGTON, May 1.—A civil service examination will be held May 18 in Washington, and, if necessary, in other cities, to secure eligibles for the position of computer in the Nautical Almanac Office, where two vacancies exist—one at $1,000, the other at $1,400.

The examination will include the subjects of algebra, geometry, trigonometry, and astronomy. Application blanks may be obtained of the United States Civil Service Commission.

Published in 1892. Copyright by New York Times

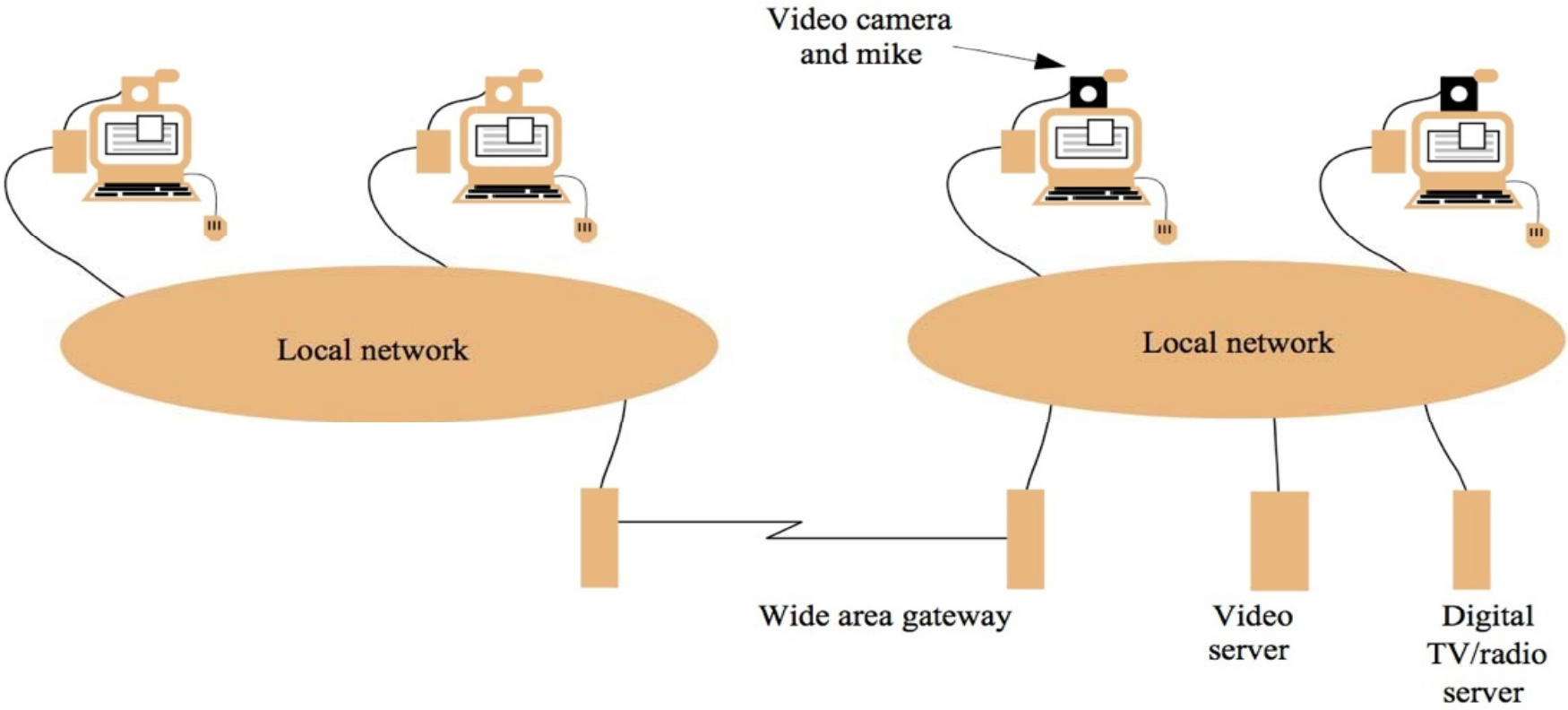
- Google's (Search Engine) Indexer
  - Rewritten in 2003 using MapReduce
  - Code lines from 3800 to 7000
- MapReduce is used by many industries
  - Facebook, ABB, Amazon, …
    - http://wiki.apache.org/hadoop/PoweredBy

Distributed Multimedia Systems (1)

- Multimedia systems
  - Generate and consume continuous stream of data in real-time
    - Large quantities of audio, video, subtitles stored decentrally
    - Timely processing and delivery essential
  - Flow specifications express “acceptable” values for
    - Data rates, delivery delay, loss or erroneous data
      - High resolution, interactive streams, reliable data streams
  - Allocation and scheduling of resources for data streams
    - Quality and/or service management
    - System tasks of processing capacity (CPU), memory and storage (RAM, HD, archive), and network bandwidth
    - Performed in response to application and user requirements
      - Quality-of-Service (QoS) with weak or strong guarantees

Distributed Multimedia Systems (2)

- Sources and destinations typically distributed across the world
  - Each being part of a Local Area Network (LAN)



Characteristics of Multimedia Streams

- Resource requirements of data streams dynamic
- Balancing resource costs against data stream quality

	Data rate (approximate)	*Sample or frame* *frequency*	*size*
Telephone speech	64 kbps	8 bits	8000/sec
CD-quality sound	1.4 Mbps	16 bits	44,000/sec
Standard TV video (uncompressed)	120 Mbps	up to 640 x 480 pixels x 16 bits	24/sec
Standard TV video (MPEG-1 compressed)	1.5 Mbps	variable	24/sec
HDTV video (uncompressed)	1000–3000 Mbps	up to 1920 x 1080 pixels x 24 bits	24–60/sec
HDTV video MPEG-2 compressed)	10–30 Mbps	variable	24–60/sec

Distributed Multimedia System Categories

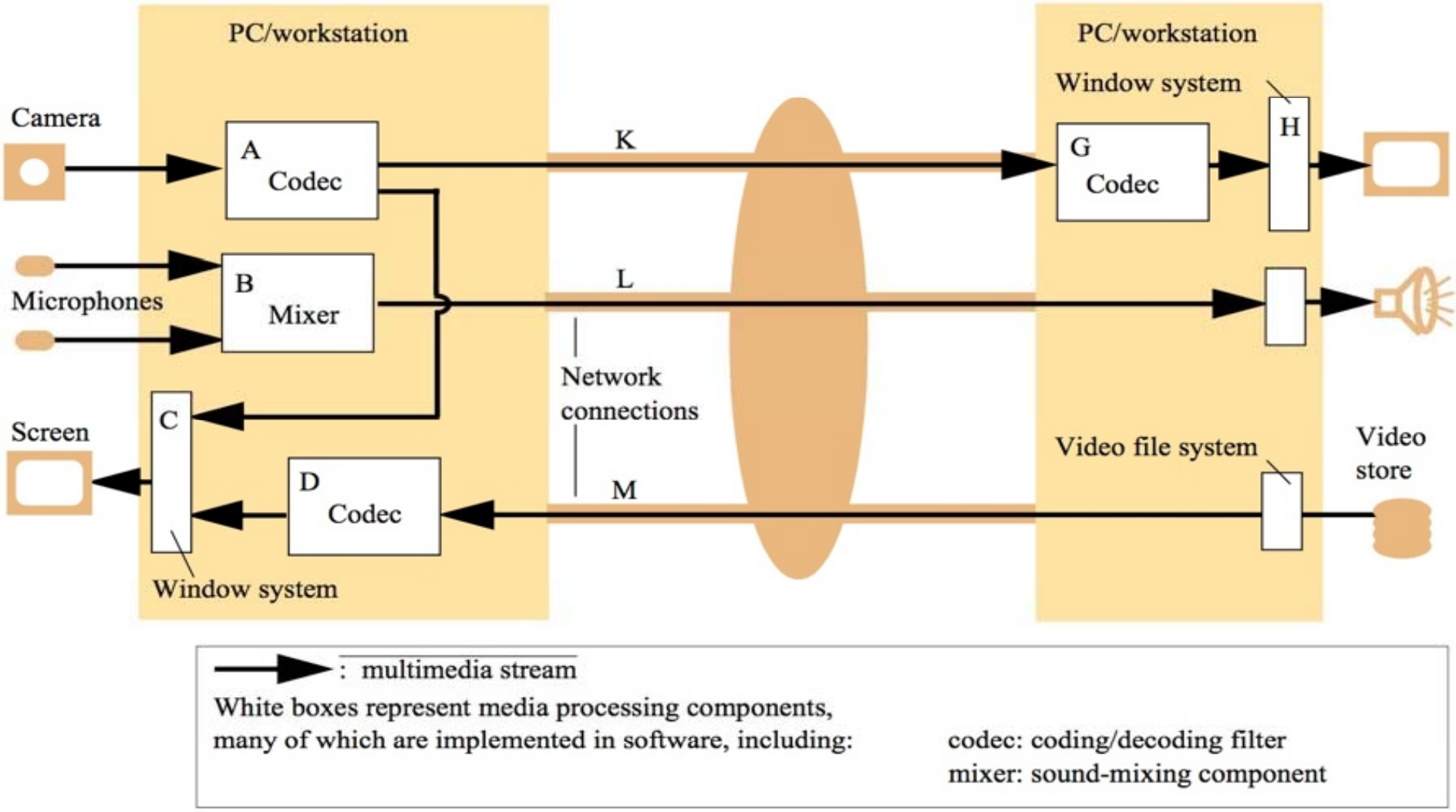
- Main service categories
  - Video-on-Demand (VoD)
  - Video-on-Reservation (VoR)
- Web-based multimedia
  - *E.g.*, YouTube, Spotify, Netflix
- Interactive applications
  - IP Telephony
  - Video conferencing

QoS Specifications

- Selected major resource requirements for main software components and network connections
  - Examples

	Component	*Bandwidth*		*Latency*	*Loss rate*	*Resources required*
	Camera	Out:	10 frames/sec, raw video 640x480x16 bits		Zero	
A	Codec	In:	10 frames/sec, raw video	Interactive	Low	10 ms CPU each 100 ms;
		Out:	MPEG-1 stream			10 Mbytes RAM
B	Mixer	In:	2 44 kbps audio	Interactive	Very low	1 ms CPU each 100 ms;
		Out:	1 44 kbps audio			1 Mbytes RAM
H	Window system	In:	various	Interactive	Low	5 ms CPU each 100 ms;
		Out:	50 frame/sec framebuffer			5 Mbytes RAM
K	Network connection	In/Out:	MPEG-1 stream, approx 1.5 Mbps	Interactive	Low	1.5 Mbps, low-loss stream protocol
L	Network connection	In/Out:	Audio 44 kbps	Interactive	Very low	44 kbps, very low-loss stream protocol

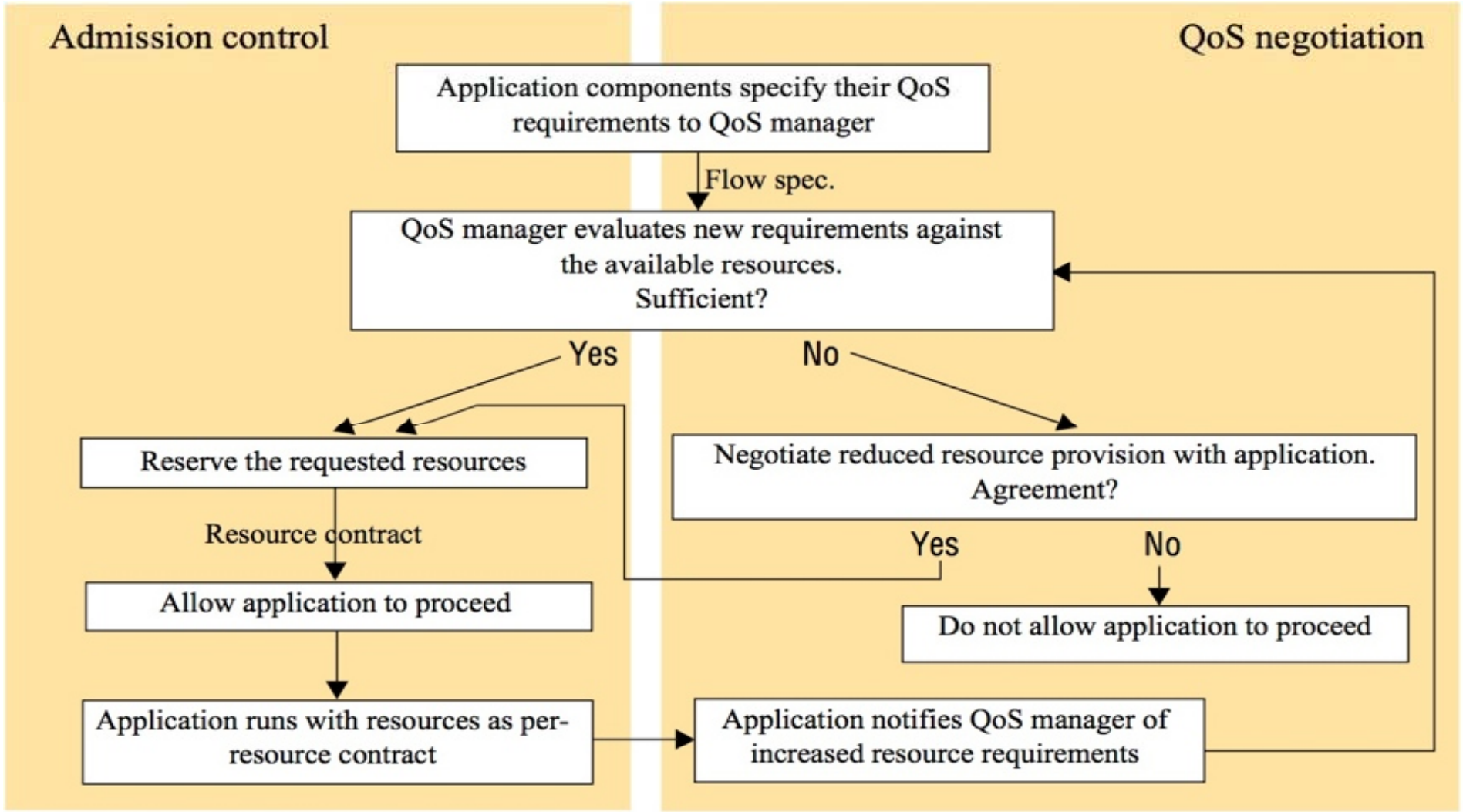
Infrastructure Components



QoS Management

- Competition between multimedia and conventional traffic
  - Media streams may compete, too
- QoS management enables the maintenance of
  - Concurrent use of physical resources
    - Multitasking in an Operating System
    - Multiplexing of data streams on one network access
  - Key task
    - Resource allocation scheme to meet all data stream requirements on a certain machine, with a dedicated network access, and the selected services provider's specifications

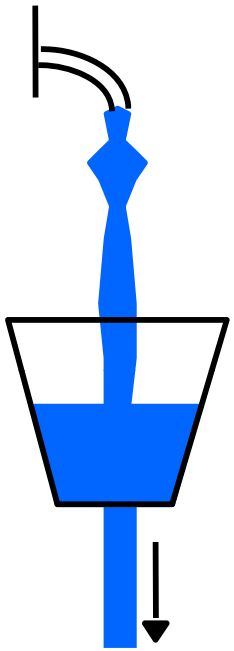
QoS Manager


Admission control
QoS negotiation
Application components specify their QoS requirements to QoS manager
Flow spec.
QoS manager evaluates new requirements against the available resources.
Sufficient?
Yes
No
Reserve the requested resources
Resource contract
Allow application to proceed
Application runs with resources as per-resource contract
Negotiate reduced resource provision with application.
Agreement?
Yes
No
Do not allow application to proceed
Application notifies QoS manager of increased resource requirements

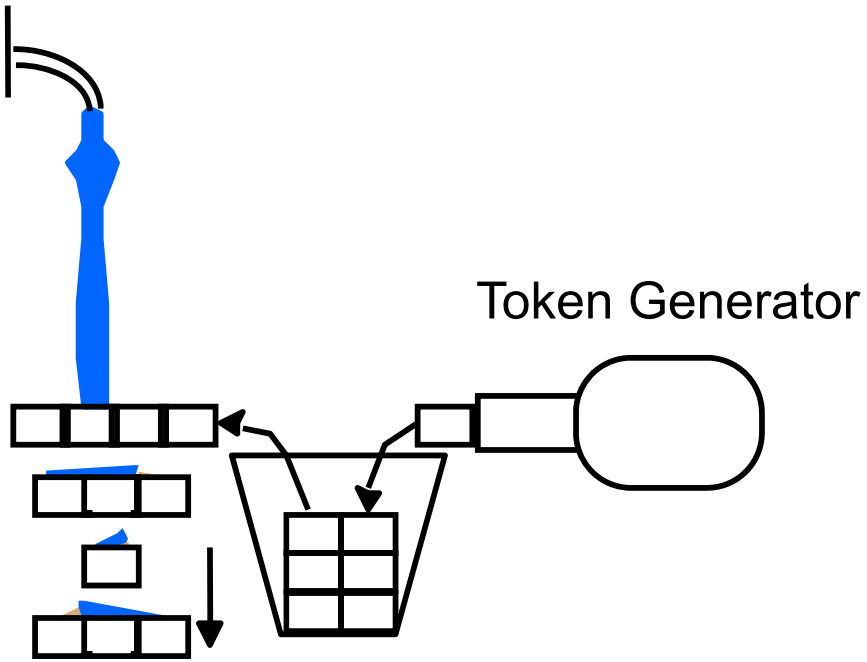
Example Functionality: Traffic Shaper

- Use of output buffering to smooth the flow of data
  - Bandwidth parameter ideally an approximation of traffic pattern
  - Two example shapers

Leaky Bucket



Token Bucket



RFC 1363 Flow Spec

- For compatibility reasons specifications of flows are standardized in eleven 16 bit name-value pairs
  - Internet-based view
  - Alternatives exist
    - Stream burstiness
    - Worst case delay bound
    - QoS classes
  - No full agreement!

	Protocol version
Bandwidth:	Maximum transmission unit
	Token bucket rate
	Token bucket size
	Maximum transmission rate
Delay:	Minimum delay noticed
	Maximum delay variation
Loss:	Loss sensitivity
	Burst loss sensitivity
	Loss interval
	Quality of guarantee

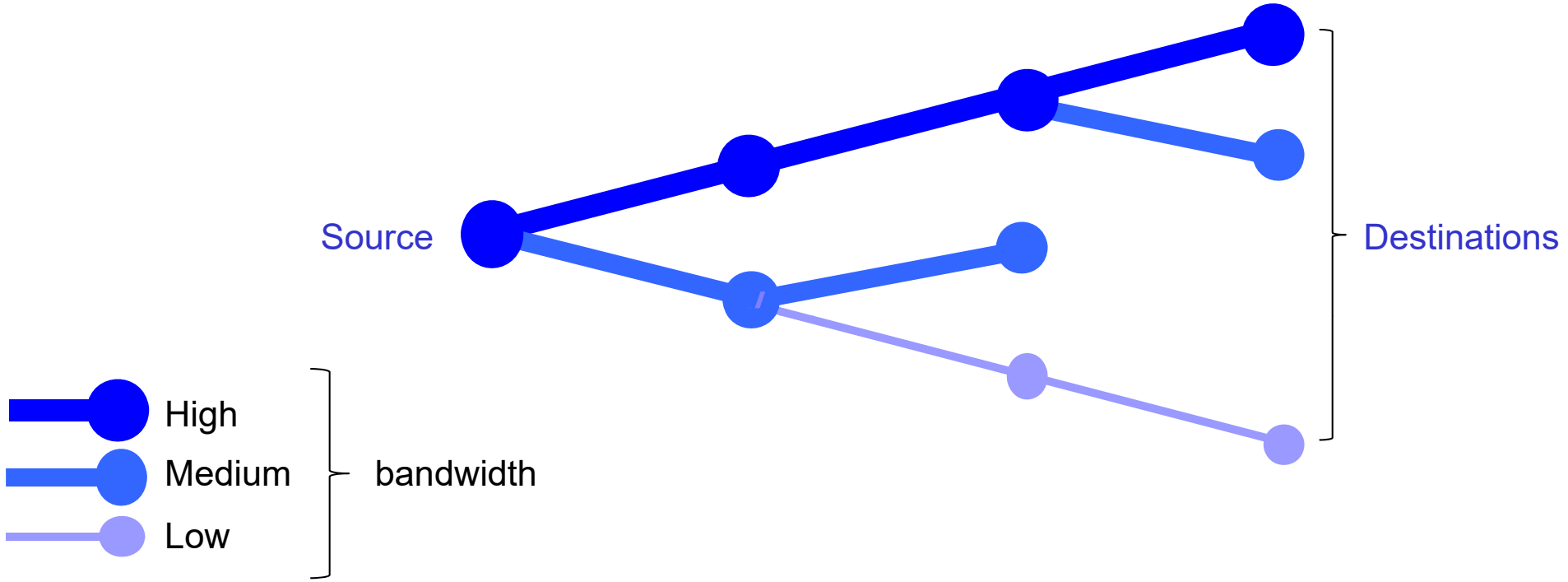
RFC: Request for Comments

Data Stream Adaptation (1)

- In case QoS cannot be met, adaptations may “help”
  - Certain probabilities of errors are adjusted
  - Different levels of quality match to different presentation qualities
- Scaling (at data source)
  - Adaptation of data stream’s bandwidth to network bandwidth
  - Live stream sampling easy
  - Stored stream sampling depends on encoding scheme used
  - Temporal scaling: reduction of audio/video resolution
  - Spatial scaling: reduction of pixels per image
  - Frequency scaling: modification of compression per image
  - Amplitudinal scaling: reduction of color depth per image

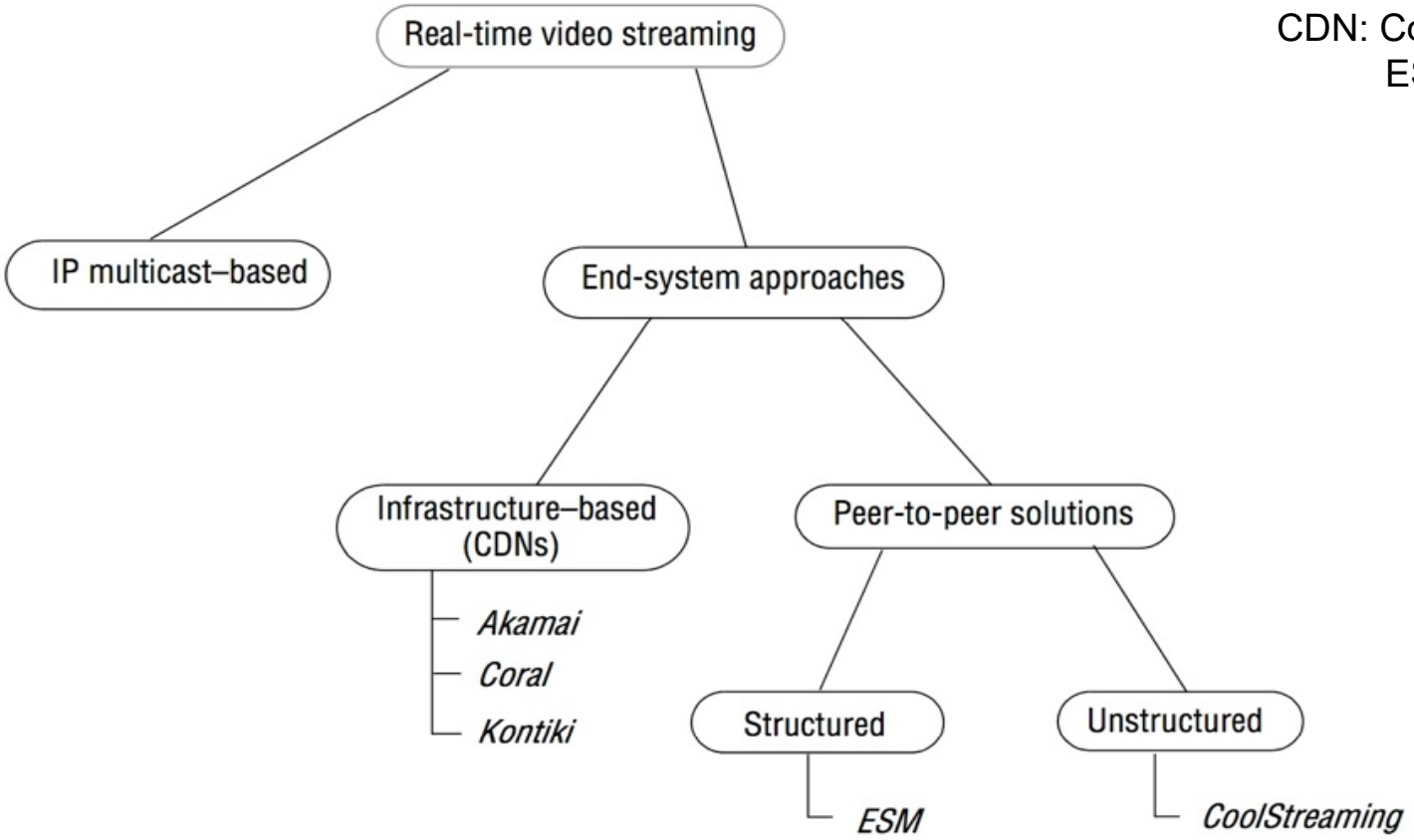
Data Stream Adaptation (2)

- Filtering (on data in transit)
  - Multiple receivers receiving the same stream does not allow for scaling
  - QoS adaptation as close to the receiver as possible



Approaches to Real-time Video Streaming

- Real-time streaming focuses on the unchanged "timing relation" of data stream elements in transit



CDN: Content Distribution System
ESM: End-system Multicast