Distributed Data Management

Serge Abiteboul

INRIA Saclay, Collège de France, ENS Cachan

Distributed computing

A **distributed system** is an application that coordinates the actions of several computers to achieve a specific task.

Distributed computing is a lot about **data**

- System state
- Session state including security information
- Protocol state
- Communication: exchanging data
- User profile
- ... and of course, the actual “application data”

Distributed computing is about querying, updating, communicating data ⤳ **distributed data management**

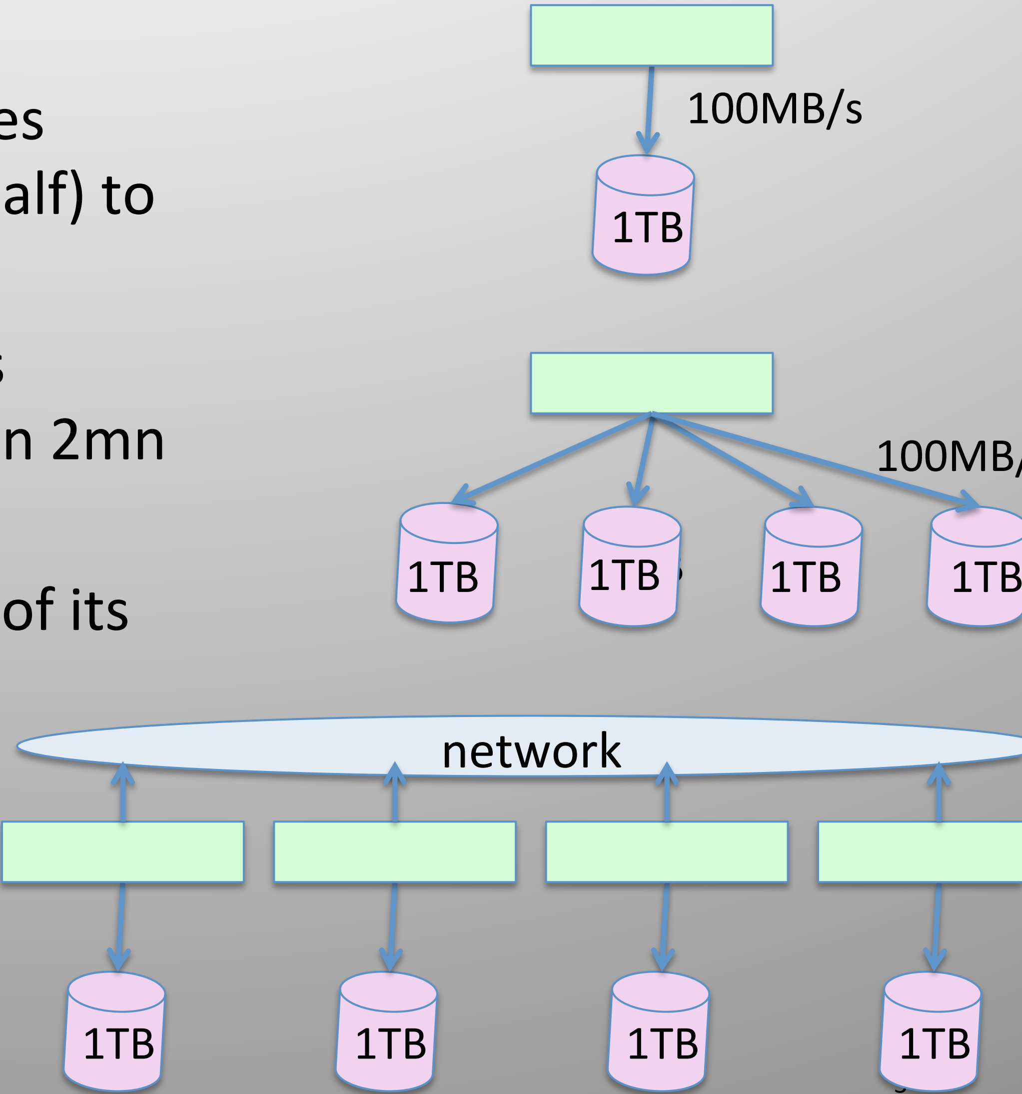
Parallelism and distribution

Sequential access: 166 minutes (more than 2 hours and a half) to read a 1 TB disk

Parallel access: With 100 disks working in parallel, less than 2mn

Distributed access: With 100 computers, each disposing of its own local disk: each CPU processes its own dataset

This is scalable



Organization

1. Data management architecture
2. Parallel architecture

    Zoom on two technologies

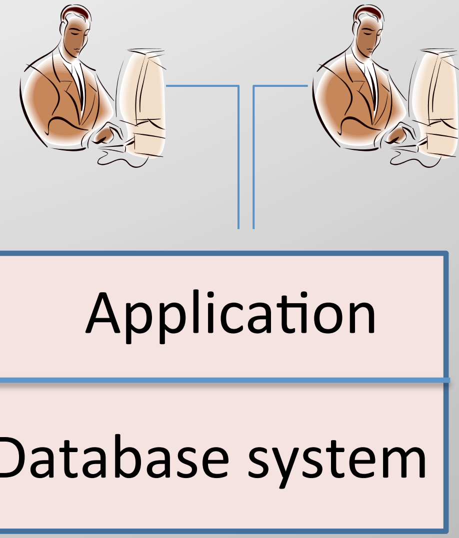
3. Cluster (*grappe*): MapReduce
4. P2P: storage and indexing
5. Limitations of distribution
6. Conclusion

Data management architecture

Deployment architecture

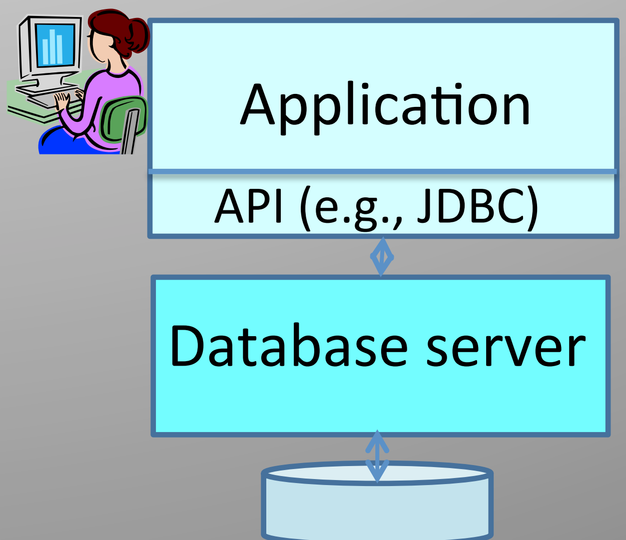
Centralized

- Multi-user mainframe & terminals



Client-Server

- Multi-user server & terminals workstation
- Client: application & Graphical interface
- Server: database system



Deployment architecture – 3 tier

Client is a browser that

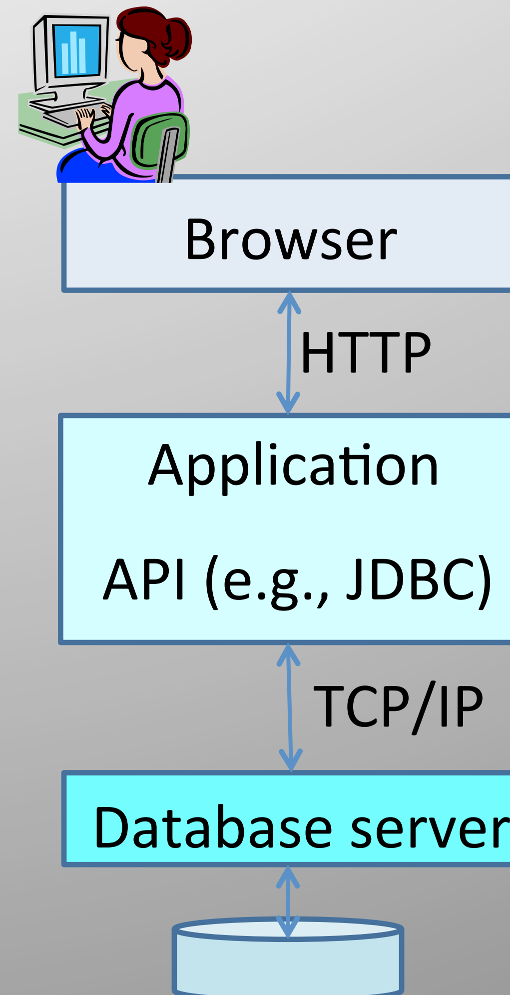
- Displays content, e.g. in HTML
- Communicates via HTTP

Central tier

- Generates the content for the client
- Runs the application logic
- Communicates with the database

Data server tier

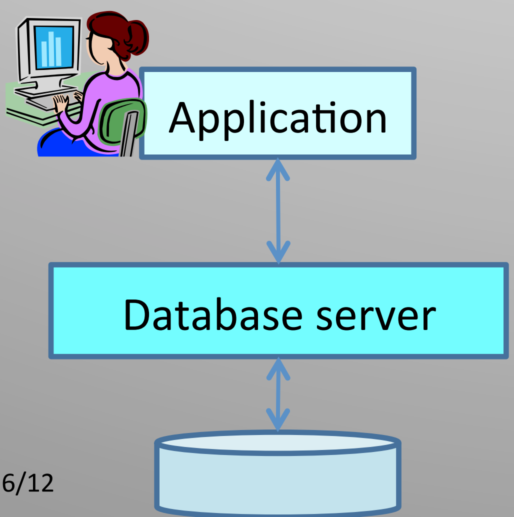
- Serves data just like in the client/server case



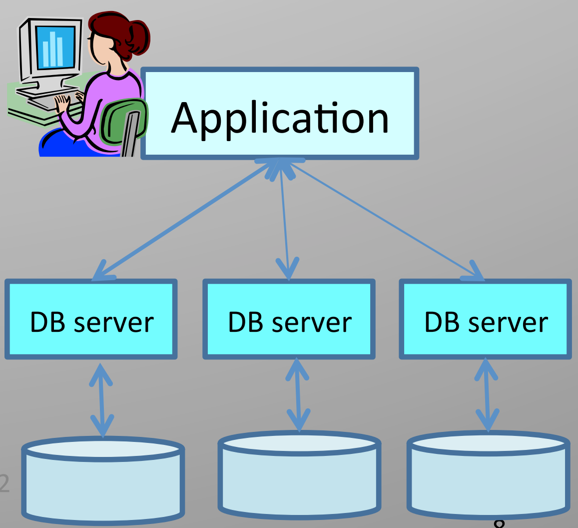
Another dimension: Server architecture

Deployment: client/server

- Example 1
  - Server: single machine

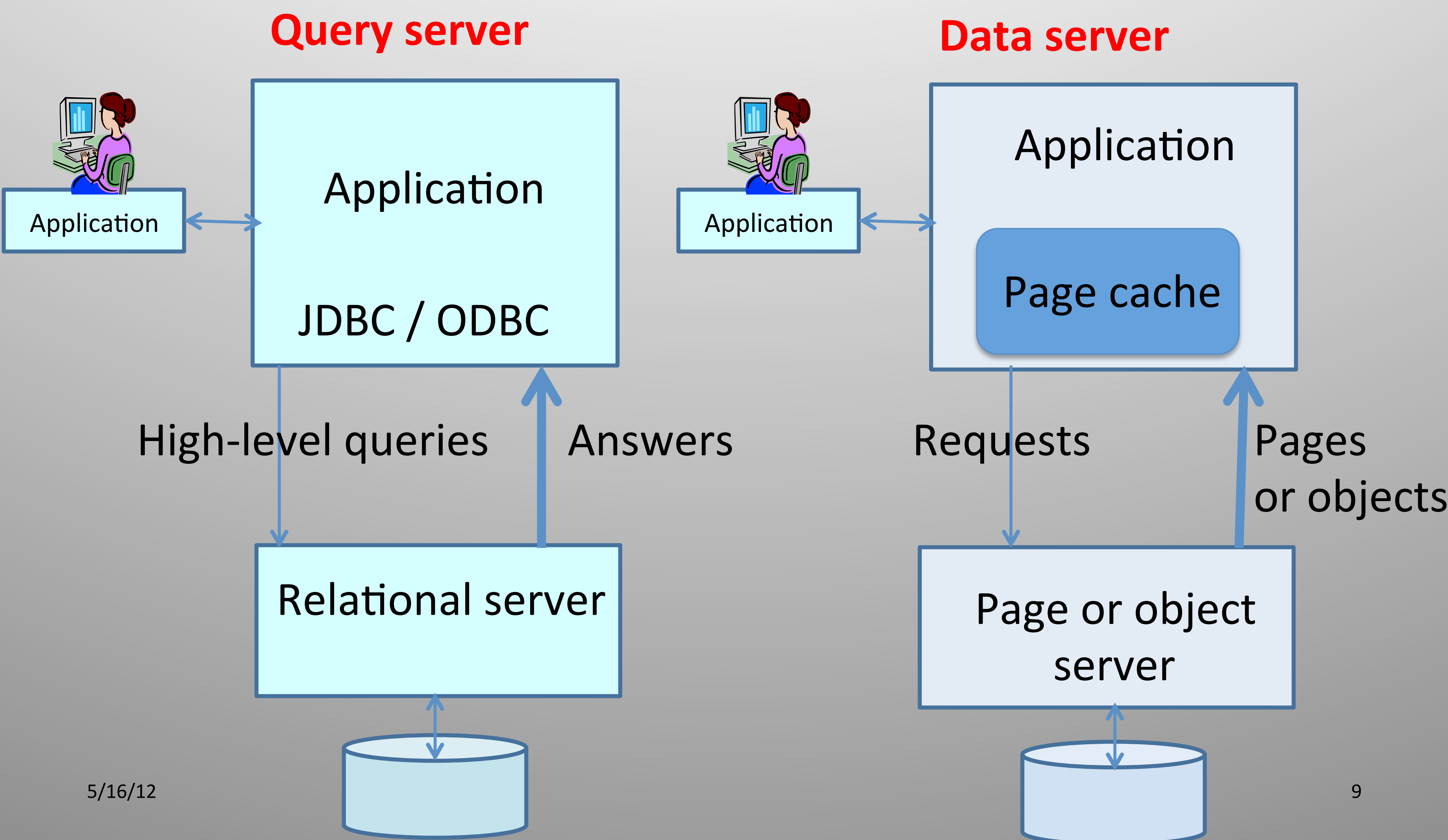


- Example 2
  - Server: parallel machine



5/16/12

Server architecture: Query vs. page/object



Parallel architecture

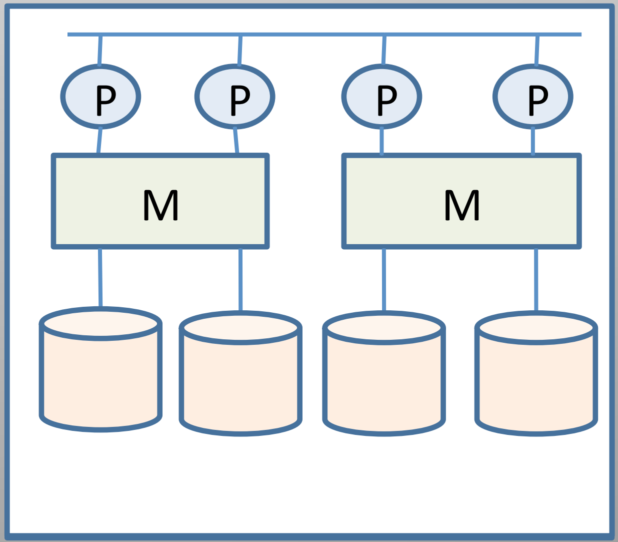
Parallel architecture

The architecture of a server is typically

- multi-CPU, multi-memory, multi-disk
- Based on very fast network

Architectures

- **Shared memory**
- **Shared disk**
- **Shared nothing**
- **Hybrid**



Comparison

Shared memory

- The bus becomes the bottleneck beyond **32-64 processors**
- Used in practice in machine of 4 to 8 processors

Shared disk

- Inter-CPU communication slower
- Good for fault-tolerance
- Bottleneck pushed to **hundreds of processors**

No sharing – only for very parallelizable applications

- Higher communication cost
- Scaling to **thousands of processors**
- Adapted for analysis of large data sets

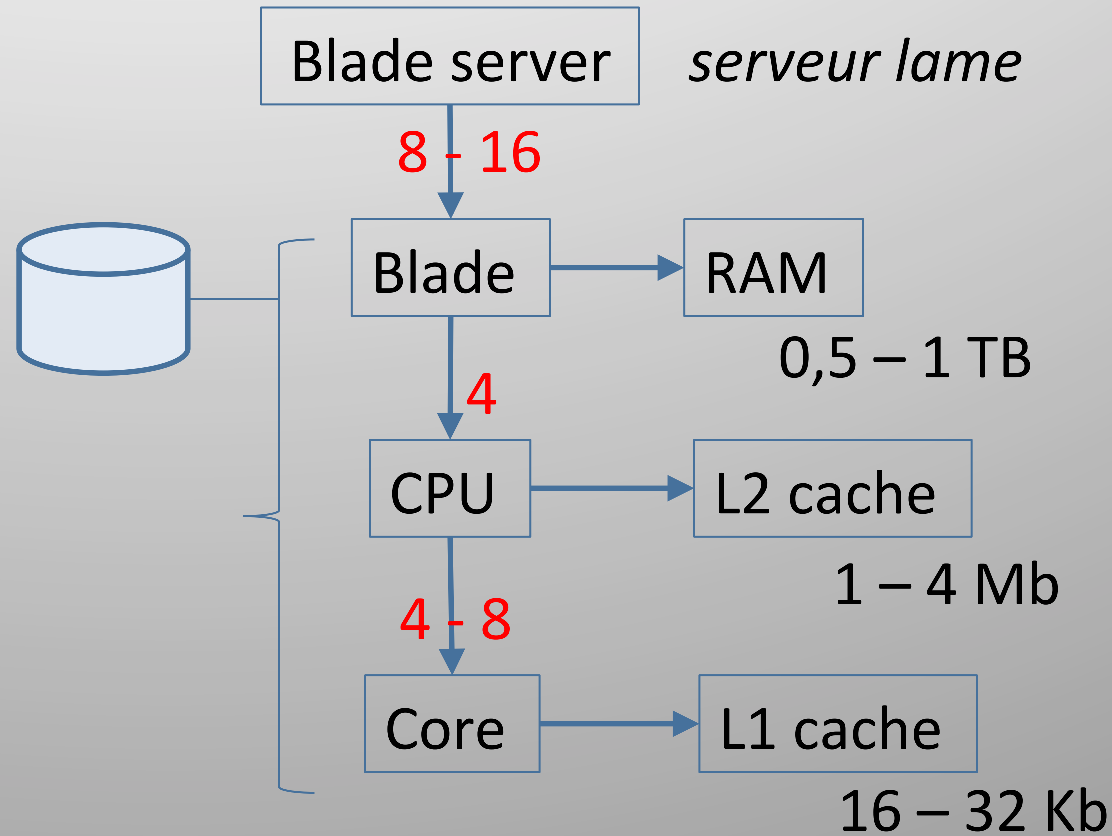
Main memory database

Beyond 100 *cores*

Beyond 10 Tb memory

Complex programming to leverage parallelism

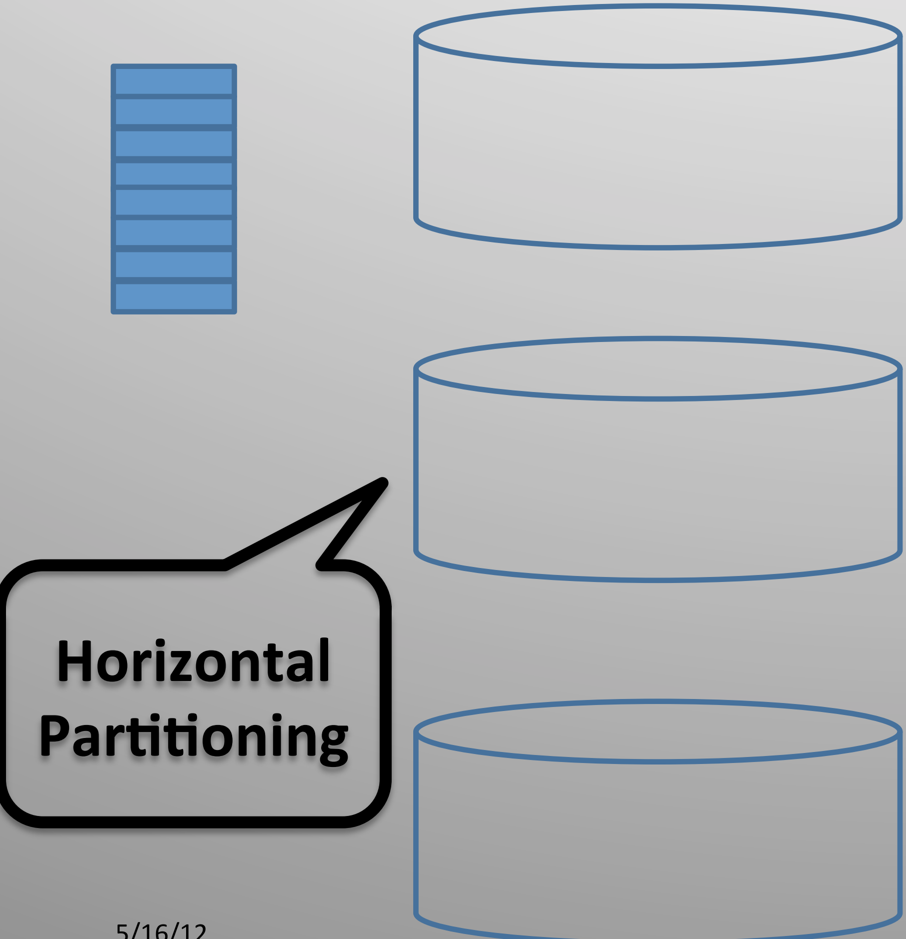
Issue: computing power and memory throughput augment – latency augments much less



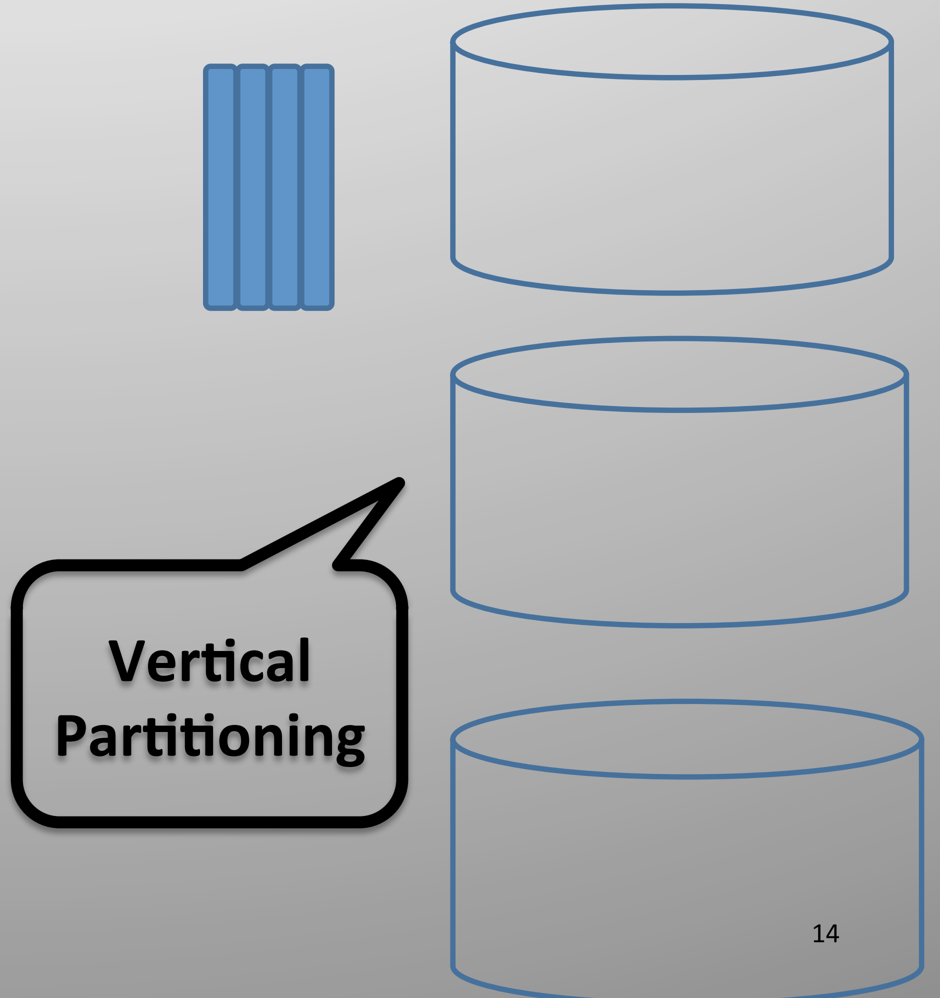
blades share nothing

Massive parallelism and partitioning

- **Line store**



- **Column store**



Line vs. column store

Lines

- Read/write a full tuple: fast
- Read/write an attribute for the entire relation: slow
- Limited compression
- Slow aggregation

- Adapted to **transactional** applications

Columns

- Read/write a full tuple: slow
- Read/write an attribute for the entire relation: fast
- Excellent compression
- Fast aggregation

- Adapted to **decisional** applications

Massive parallelism & column store

Parallelism

SGBD-R : Teradata
Neteeza(IBM)
DATAllegro (Microsoft)
Open source : Hadoop
(in a few minutes…)

Column store

Sybase IQ
Kickfire (Teradata)
:Open source MonetDB

Parallelism & column store

Exasol
Vertica
Greenplum (EMC)
Opensource: Hadoop HBase

Cluster: MapReduce

To process (e.g. to analyze) large quantities of data

- Use parallelism
- Push data to machines

MapReduce

MapReduce : a computing model based on heavy distribution that scales to huge volumes of data

- 2004 : Google publication
- 2006: open source implementation, Hadoop

Principles

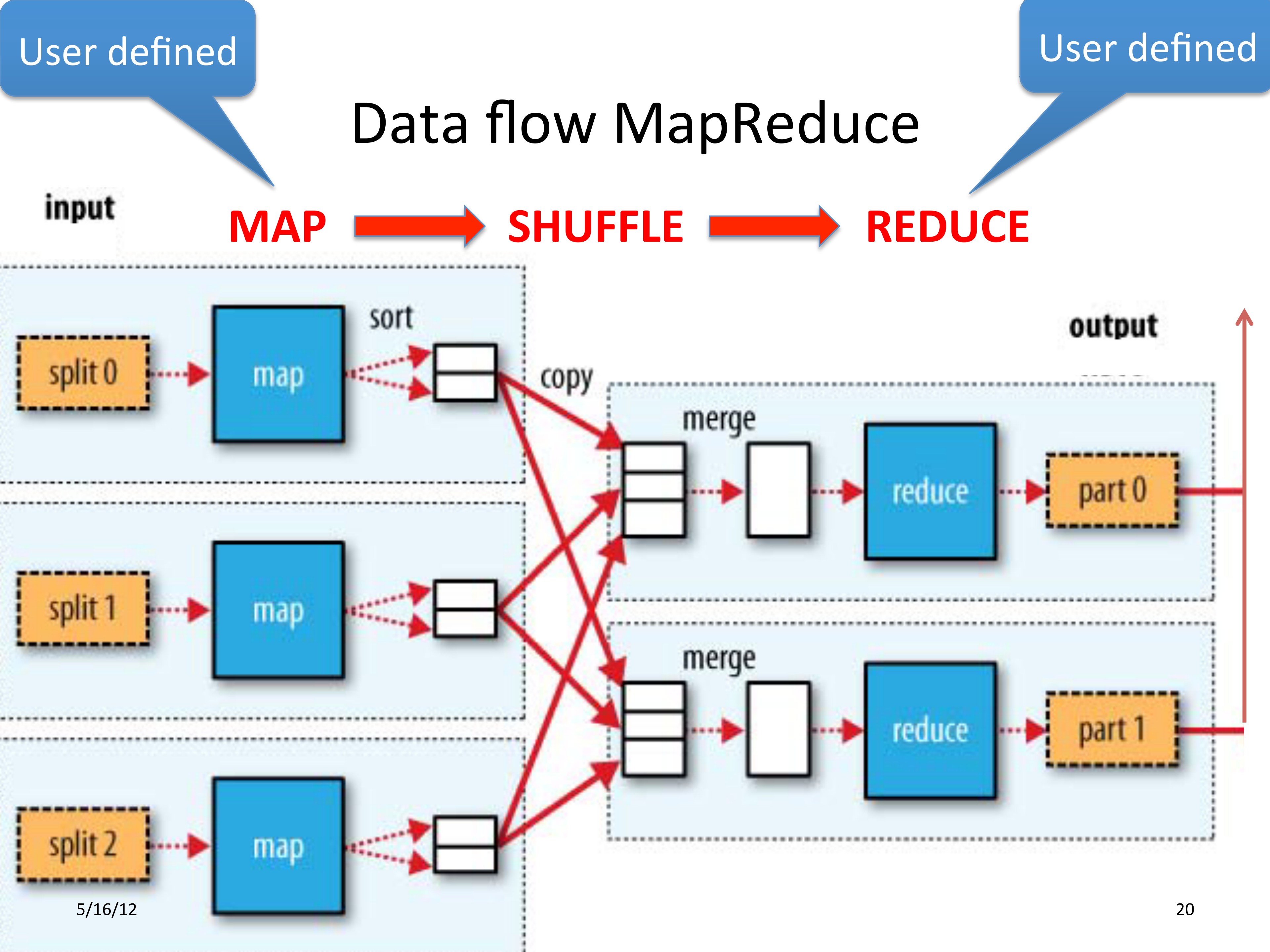
- Data distributed on a large number of **shared nothing machines**
- Parallel execution; processing pushed to the data

MapReduce

Three operations on key-value pairs

Map	**user-defined**	(*transforme*)
Shuffle	fixed behavior	(*mélange*)
Reduce	**user-defined**	(*réduire*)

Data flow MapReduce



MapReduce example

- Count the number of occurrences of each word in a large collection of documents

Map

Jaguar 1
Atari 1
Felidae 1
Jaguar 1…

u1 jaguar world mammal felidae family.

u2 jaguar atari keen use 68K family device.

u3 mac os jaguar available price us 199 apple new family pack

u4 such ruling family incorporate jaguar their name

Jaguar 1
Available 1
Apple 1
Jaguar 2…

Shuffle

Jaguar 1
Atari 1
Felidae 1
Jaguar 1…
…
Jaguar 1
Available 1
Apple 1
Jaguar 2
…

Jaguar 1,1,1,2
Mammal 1
Family 1,1,1
Available 1
…

Reduce

Jaguar 1,1,1,2

Mammal 1

Family 1,1,1

Available 1

…

Jaguar 5

Mammal 1

Family 3

Available 1

…

MapReduce functionalities

Map: (K, V) → list (K’, V’) ; typically:
- Filter, select a (new) key, project, transform
- Split results in M files for M reducers

Shuffle: list (K’, V’) → list (K’, list (V’))
- Regroup the pairs with the same keys

Reduce: (K’, list (V’)) → list (K’’, V’’) ; typically:
- Aggregation(COUNT, SUM, MAX)
- Combination, filtering (example join)

Optional optimization : combine: list(V’) → V’
- Run on a mapper to combine pairs with the same key into a single pair

Hadoop

- Open source, Apache implementation in Java
  - Main contribution from Yahoo
- Main components
  - Hadoop file system (HDFS)
  - MapReduce (MR)
  - Hive: simple data warehouse based on HDFS and MR
  - Hbase: key-value column store on HDFS
  - Zookeeper: coordination service for distributed applications
  - Pig: dataflow language on HDFS and MR ☞
- Java and C++ API
  - Streaming API for other language
- Very active community

Pig Latin

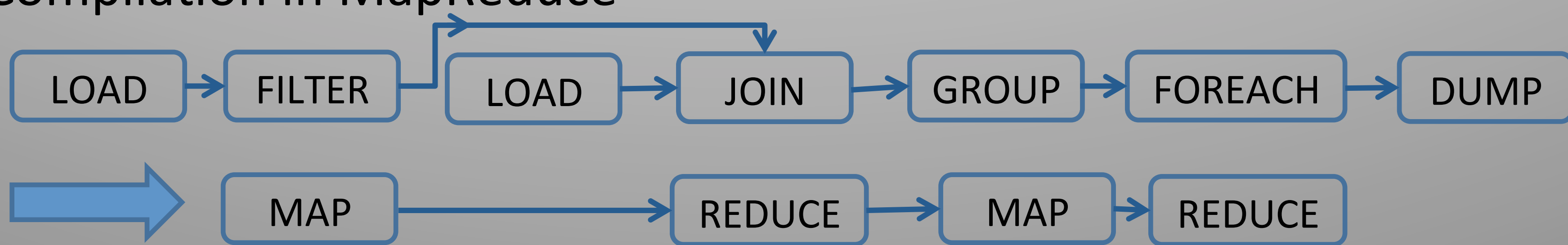
For some author, count how many editors this author has

N1NF model

Example

```
Books = LOAD 'book.txt' AS (title: chararray, author: chararray,…);
Abiteboul = FILTER Books BY author == 'Serge Abiteboul';
Edits = LOAD 'editors.txt' AS (title: chararray, editor: chararray);
Joins = JOIN Abiteboul BY title, Edits BY title;
Groups = GROUP Joins BY Abiteboul::author;
Number = FOREACH groups GENERATE group, COUNT(Joins.editor);
DUMP Number
```

Compilation in MapReduce



What's going on with Hadoop

- Limitations
  - Simplistic data model & no ACID transaction
  - Limited to batch operation
  - Limited to extremely parallelisable applications
- Good recovery to failure
- Scales to huge quantities of data
  - For smaller data, it is simpler to use large flash memory or main memory database
- Main usage today (sources: TDWI, Gartner)
  - Marketing and customer management
  - Business insight discovery

Where does this technology fit

Data warehouse

ETL	Storage	Business Intelligence
X		
	X	
X	X	X

P2P: storage and indexing

To index large quantities of data

- Use existing resources
- Use parallelism
- Use replication

Peer-to-peer architecture

P2P: Each machine is both a server and a client

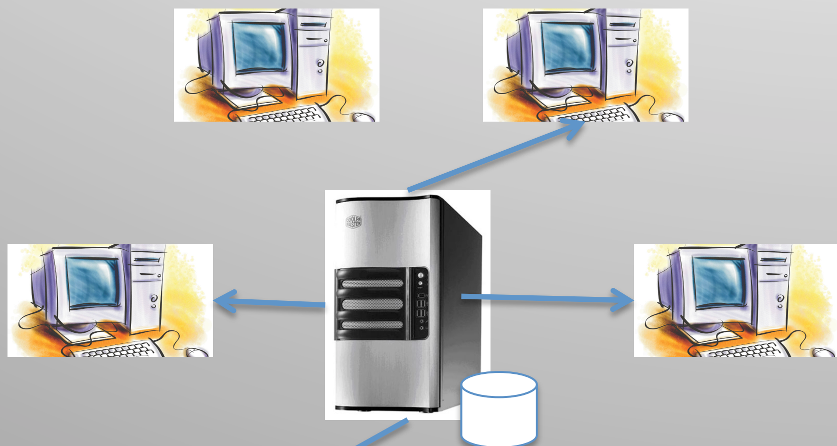
Use the resources of the network

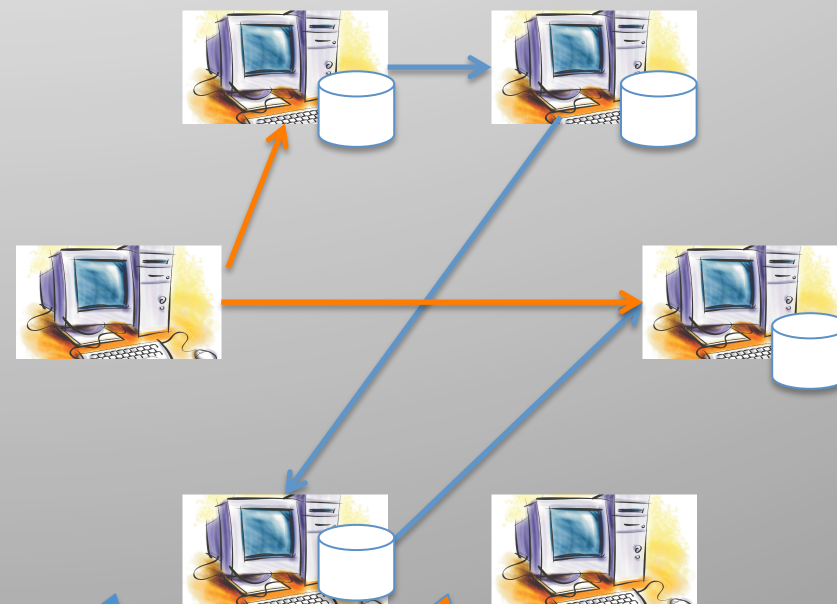
- Machines with free cycles, available memory/disk)

- Communication: Skype
- Processing: seti@home, foldit
- Storage: emule

Power of parallelism

Performance, availability, etc.



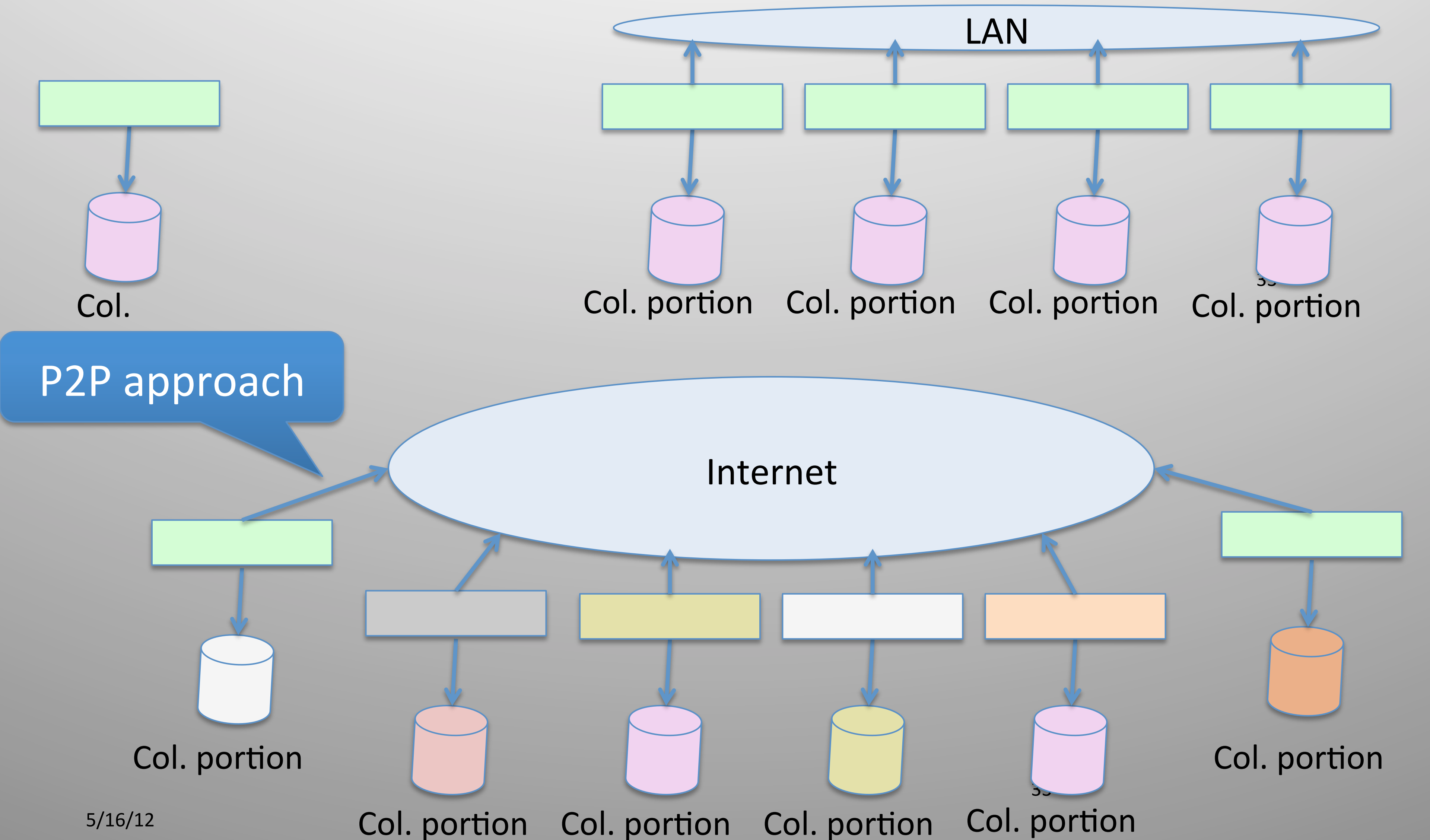


Server saturates

P2P scales

Parallel loads

Managing a large collection



Difficulties

- Peers are autonomous, less reliable
- Network connection is much slower (WAN vs. LAN)
- Peers are heterogeneous
  - Different processor & network speeds, available memories
- **Peers come and go**
  - Possibly high churn out (*taux de désabonnement)*

- Possibly much larger number
- Possible to have peers “nearby on the network”

And the index?

Centralized index: a central server keeps a general index

- Napster

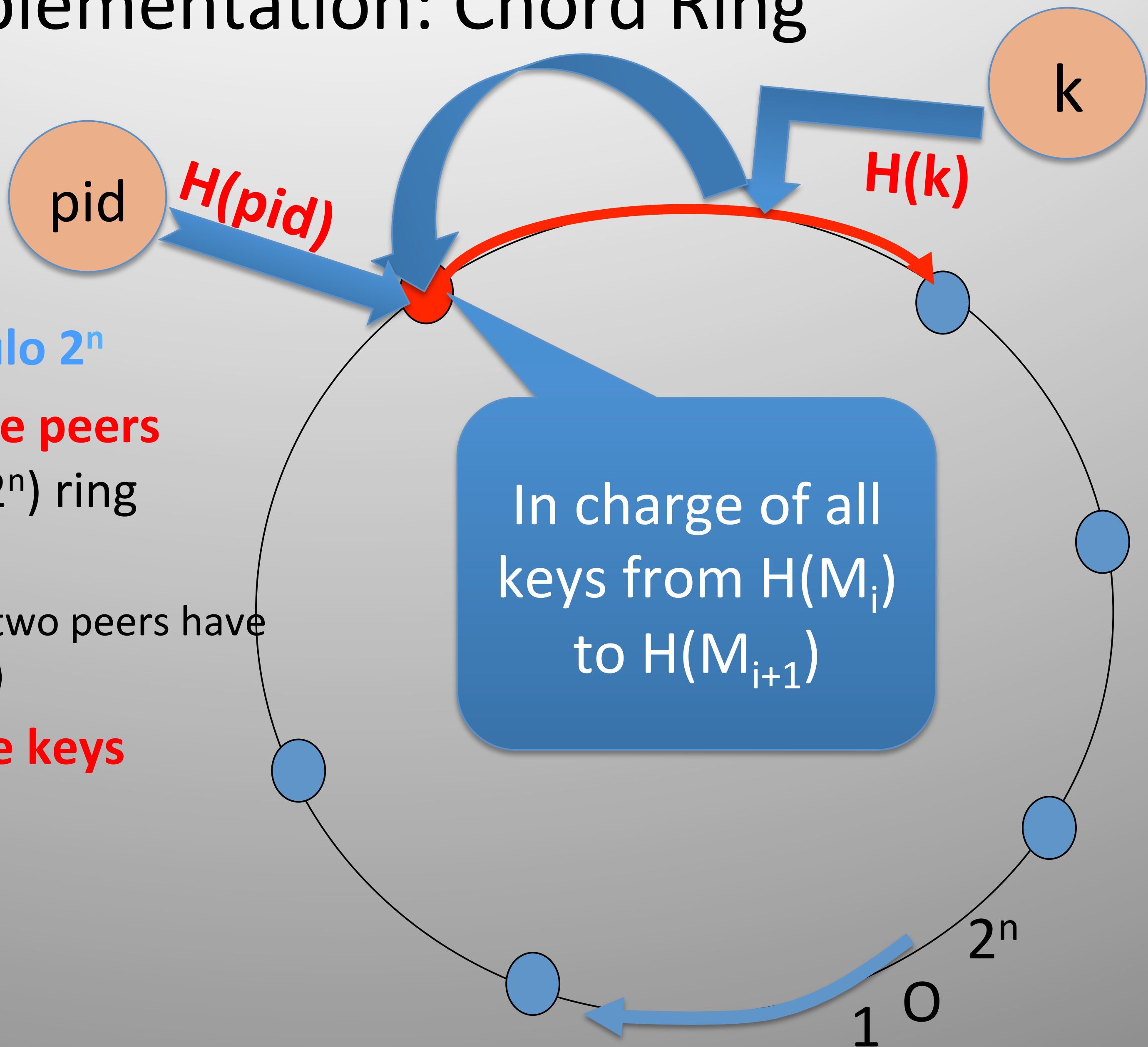
Pure P2P: communications are by flooding

- Each request is sent to all neighbors (modulo time-to-life)
- Gnutella 0.4, Freenet

Structured P2P: no central authority and indexing using an "overlay“ network (*réseau surimposé*)

- Chord, Pastry, Kademlia
- **Distributed HASH table**: index search in O (log(n))

Implementation: Chord Ring



Hashing is **modulo 2^n**

H **distributes the peers**

around the (0.. 2^n) ring

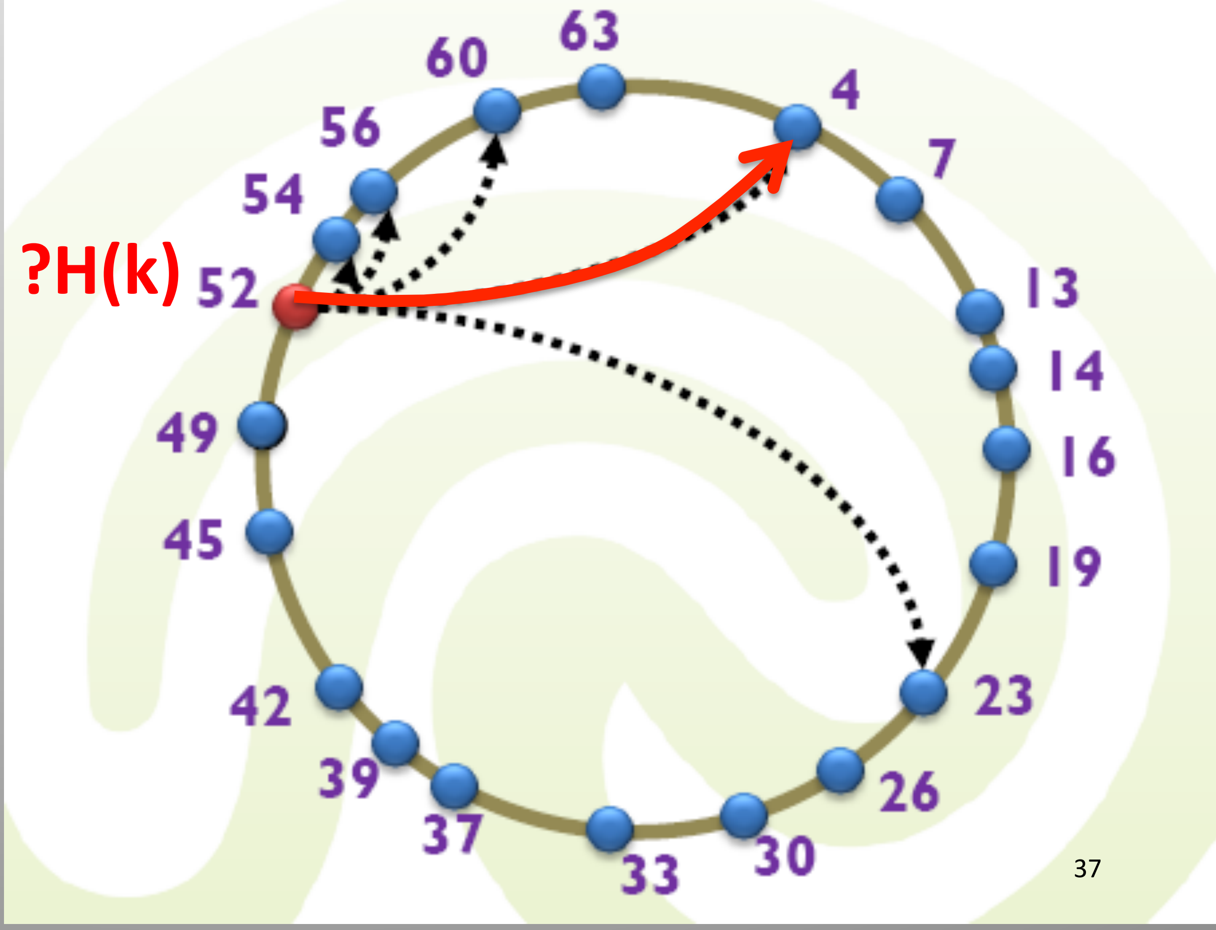
- $O < H(pid) < 2^n$
- We assume no two peers have the same H(pId)

H distributes the keys

around the ring

Perform a search in log(n)

- Each node has a routing table with : « fingers »
- Key k with
  H(k) = 13
- 4<13<23
- Forward the
  query to the
  peer in charge
  of 4…

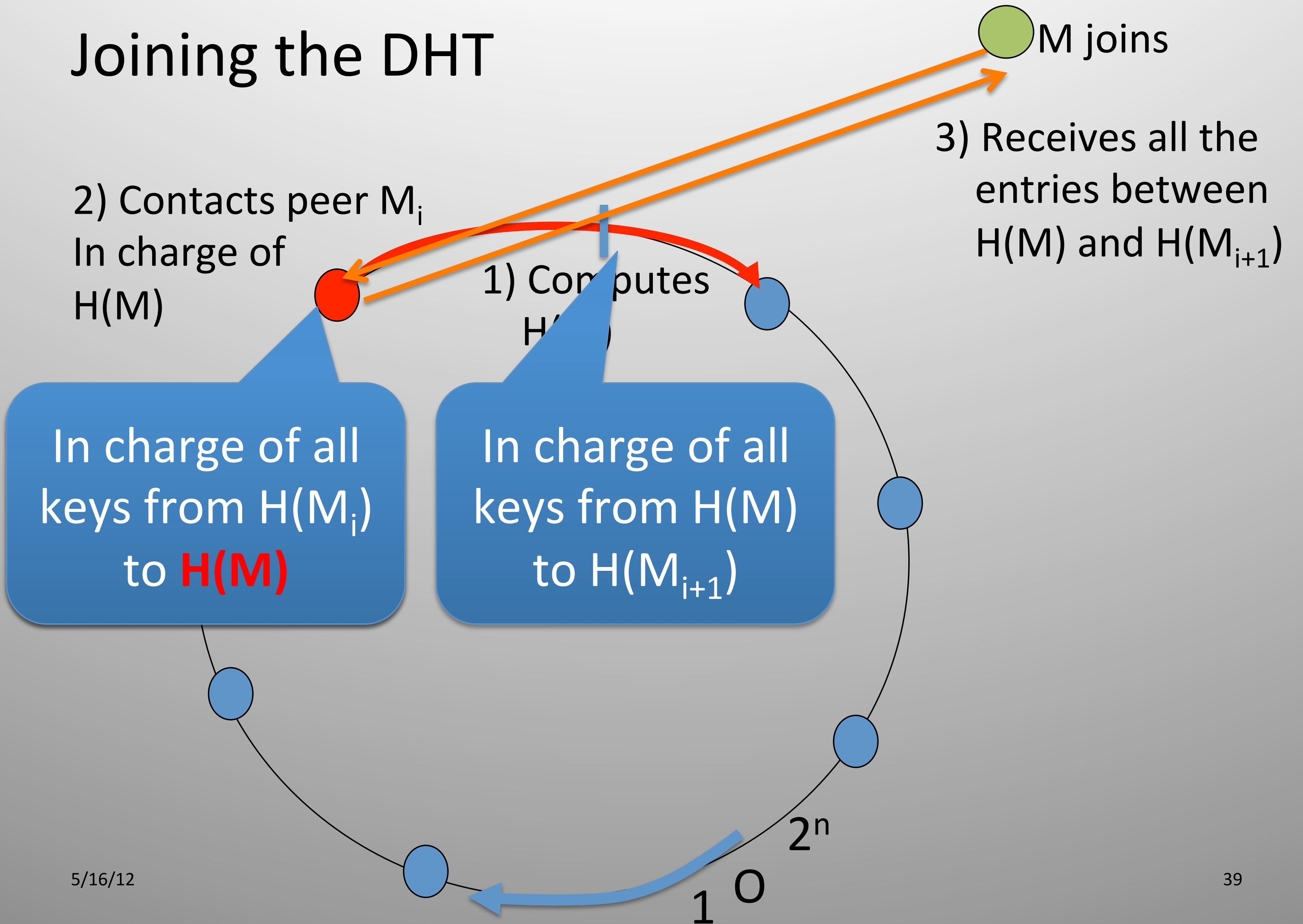


Search in log(n)

- Ask any peer for key k
- This peers knows log(n) peers and the smallest key of each
- Ask the peer with key immediately less than H(k)
- In the worst case, divide by 2 the search space
- After log(n) in the worst case, find the peer in charge of k

- Same process to add an entry for k
- Or to find the values for key k

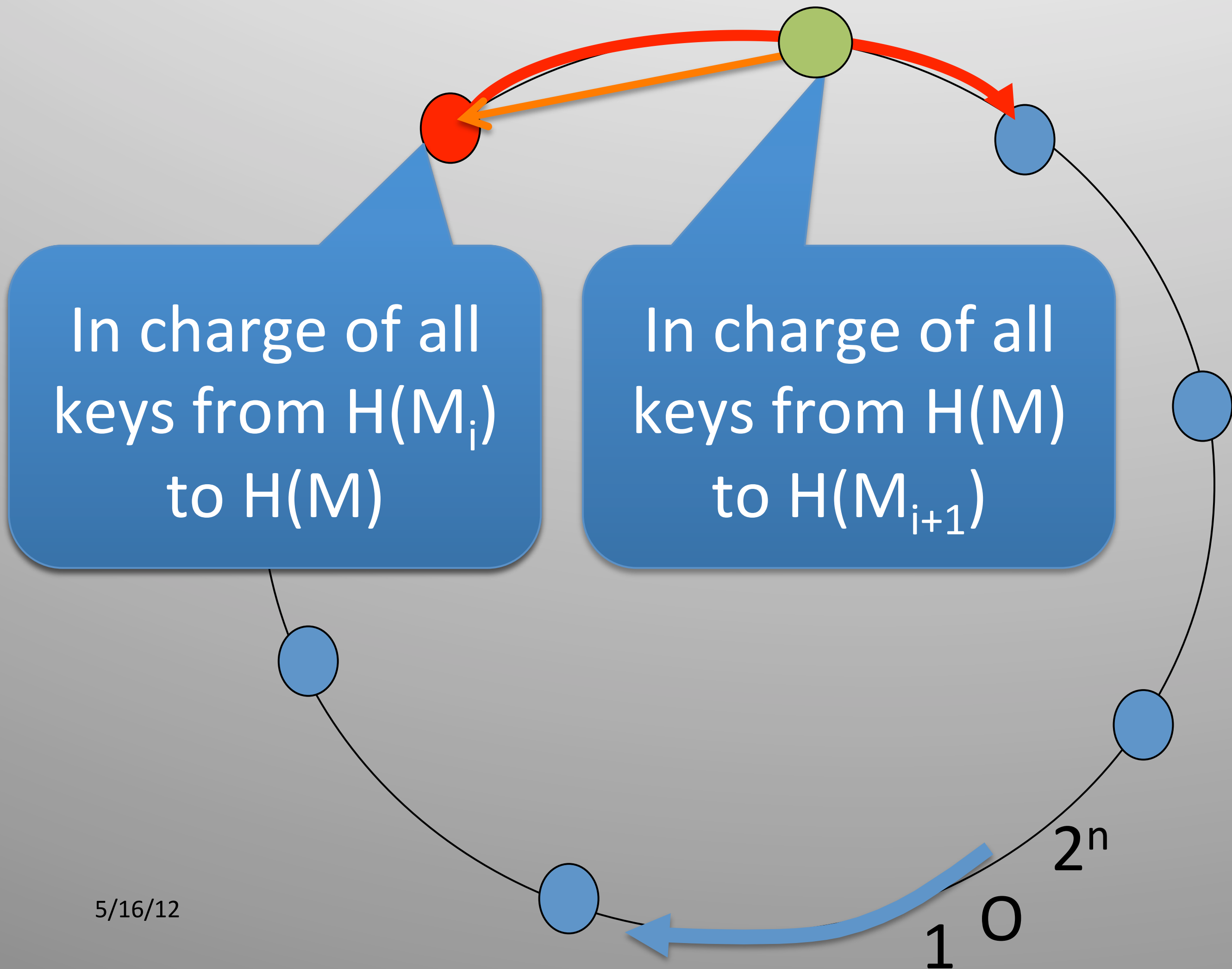
Joining the DHT



Leaving the DHT

M leaves

1) Sends to previous peer on the ring all its entries (between H(M) and $H(M_{i+1})$)



Issues

- When peers come and go, maintenance of finger tables is tricky
- Peer may leave without notice: only solution is **replication**
  - Use several hash function H1, H2, H3 and maintain each piece of information on 3 machines

Advantages & disadvantages

- Advantages
  - Scaling
  - Cost effective: take advantage of existing resources
  - Performance, availability, reliability (potentially because of redundancy but rarely the case in practice)
- Disadvantages
  - Servers may be selfish, unreliable $\leadsto$ hard to guarantee service quality
  - Communication overhead
  - Servers come and go $\leadsto$ need replication
    replication overhead
  - Slower response
  - Updates are expensive

Limitations of distribution: CAP theorem

Main idea

- Use heavy distribution
- Use heavy replication (at least for popular data)

- Is this the magical solution to any management of huge data?
- Yes for **very parallelizable problems** and **static** data collections
- **If there are many updates:**

Overhead: for each update, we have to realize as many updates as there are replicas

Problem: the replicas start diverging

Properties of distributed data management systems

Scalability refers to the ability of a system to continuously evolve in order to support a growing amount of tasks

Efficiency

- response time (or latency): the delay to obtain the first item, and
- throughput (or bandwidth): the number of items delivered in a given period unit (e.g., a second)

CAP properties

Consistency = all replicas of a fragment are always equal

- Not to be confused with ACID consistency
- Similar to ACID atomicity: an update atomically updates all replicas
- At a given time, all nodes see the same data

Availability

- The data service is always available and fully operational
- Even in presence of node failures
- Involves several aspects:

    Failure recovery

    Redundancy: Data replication on several nodes

CAP properties

Partition Tolerance

- The system must respond correctly even in presence of node failures
- Only accepted exception: total network crash
- However, often multiple partitions may form; the system must
  - prevent this case of ever happening
  - Or tolerate forming and merging of partitions without producing failures

Distribution and replication: limitations

CAP theorem: Any highly-scalable distributed storage system using replication can only achieve a maximum of two properties out of consistency, availability and partition tolerance

- Intuitive; main issue is to formalize and prove the theorem
  - Conjecture by Eric Brewer
  - Proved by Seth Gilbert, Nancy Lynch
- In most cases, **consistency is sacrificed**
  - Many application can live with minor inconsistencies
  - Leads to using weaker forms of consistency than ACID

Conclusion

Trends

The cloud

Massive parallelism

Main memory DBMS

Open source software

Trends (continued)

Big data (OLAP)

- Publication of larger and larger volumes of interconnected data
- Data analysis to increase its value
  - Cleansing, duplicate elimination, data mining, etc.
- For massively parallel data, a simple structure is preferable for performance
  - Key / value > relational or OLAP
  - But a rich structure is essential for complex queries

Massive transactional systems (OLTP)

- Parallelism is expensive
- Approaches such as MapReduce are not suitable

3 principles?

New massively parallel systems ignore the 3 principles

- Abstraction, universality & independence

Challenge: Build the next generation of data management systems that would meet the requirements of extreme applications without sacrificing any of the three main database principles

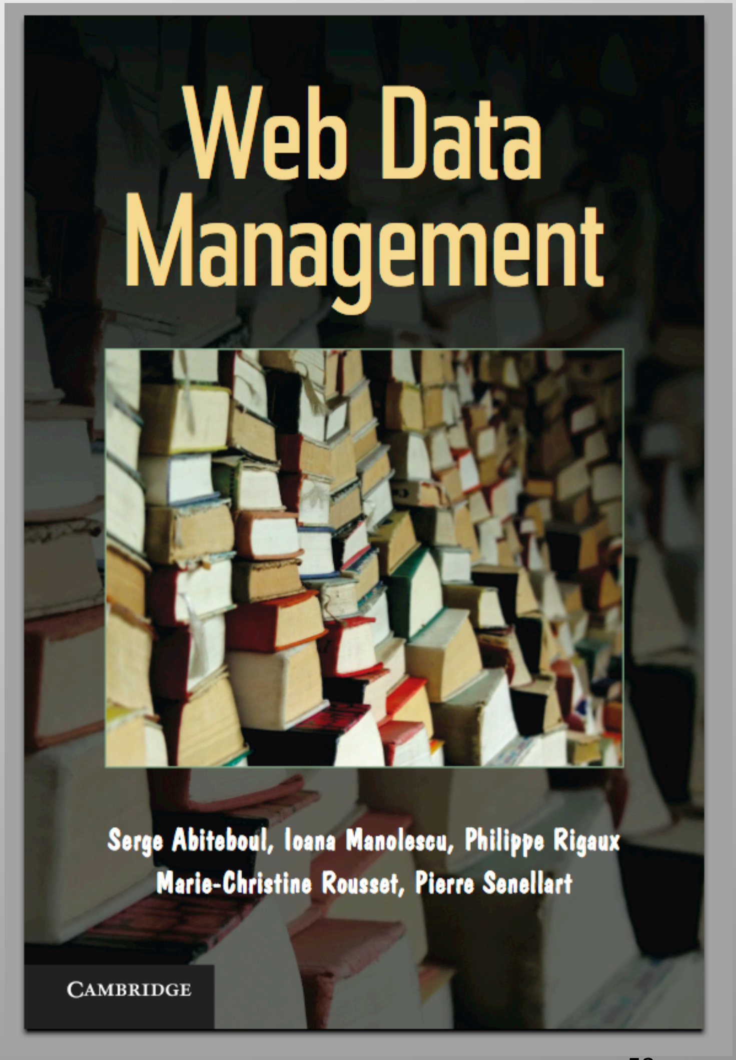
Reference

Again the Webdam book:
webdam.inria.fr/Jorge

Partly based on some joint presentation with Fernando Velez at Data Tuesday, in Microsoft Paris

Also:

Principles of distributed database systems, Tamer Özsu, Patrick Valduriez, Prentice Hall




Merci !
FRAPAR.

Gerhard Weikum

- **Max-Planck-Institut für Informatik**
- Fellow: ACM, German Academy of Science and Engineering
- Previous positions: Prof. Saarland University, ETH Zurich, MCC in Austin, Microsoft Research in Redmond
- PC chair of conferences like ACM SIGMOD, Data Engineering, and CIDR
- President of the VLDB Endowment
- **ACM SIGMOD Contributions Award in 2011**