Architectures for massive data management

Ioana Manolescu
INRIA Saclay & Ecole Polytechnique
ioana.manolescu@inria.fr
http://pages.saclay.inria.fr/ioana.manolescu/

M2 Data and Knowledge
Université de Paris Saclay

Who am I

- Senior **researcher** (DR) at INRIA Saclay
- I work on **algorithms and systems** for efficient management of complex data
  - Complex data: complex structure (trees, graphs…) and/or complex semantics (meaning)
  - Data management: queries, updates
  - Efficiency: fast evaluation; low resource usage
- Always interested in developing some novel massive DM system ☺ and looking for good interns
- PhD: INRIA Rocquencourt, France, 2001
- Master: Ecole Normale Supérieure and U. Paris 6
- Undergraduate: Polytechnic U. Bucharest, Romania

What is your background in data management?

Please fill in the **questionnaire**

Course goal

1. Discuss the main characteristics (dimensions) of massive data management platforms
   – « Big Data »
2. Present the main classes of such systems, according to the above dimensions
3. Analyze advantage/disadvantage trade-offs
4. Introduce some open research issues

Architectures to be covered

- Distributed databases
- Parallel infrastructures
  - MPI
  - MapReduce
- P2P systems
  - Structured
  - Unstructured
- Key-value stores
  - Redis, DynamoDB
- Large distributed tables (Spanner, F1)
- Graph databases: Neo4J, Virtuoso
- Pregel
- Flink
  - Standard
  - Streaming
  - Spark
- A few selected research prototypes

Course organization

- **Instructor**: Ioana Manolescu (INRIA)

- **Location**: U. Paris Sud
  - Lecture: **D201**
  - Lab: **D104** (follow the schedule!)
- **Evaluation**: exam + lab work

Practical matters

- You will have access to the course material in PDF format
  - At least, we will send them on the list: dk-students-2016@googlegroups.com
- Feel free to contact me
  - Ask us questions during course days
  - Write (usable ☺) e-mail ioana.manolescu@inria.fr

Today's course plan

1. **Motivation**: Big Data
2. Database fundamentals (recall or crash-course)
   1. ACID properties
   2. Query evaluation in database management systems (DBMSs)
   3. Distributed databases
3. Main classes of massive data management systems
4. Massively distributed computing models: MPI, P2P, DFS

MOTIVATION: BIG DATA

Defining Big Data: the V's

- Volume
  - Scale
- Velocity
  - Speed of producing and consuming the data
- Variety
  - Very different sources and data types
- Veracity
  - Is the data correct / certain / true?

Where does the data volume come from? (1)

- Human-produced data
  - Web content: Web pages, blogs, social networks, tweets…
  - Twitter: 7 Terabytes (1 tera = 10^{18}) per day
  - Facebook: 10 Terabytes per day

Where does the data volume come from? (1)

- Human-produced data
  - Web content: Web pages, blogs, social networks, tweets…
  - Twitter: 7 Terabytes (1 tera = 10^{18}) per day
  - Facebook: 10 Terabytes per day
- Machine-produced data
  - Log data from all kind of servers
  - Real world devices: payments, telecom, energy, weather, transportation, shipment…

    Sensors, including on (US) highways and trains

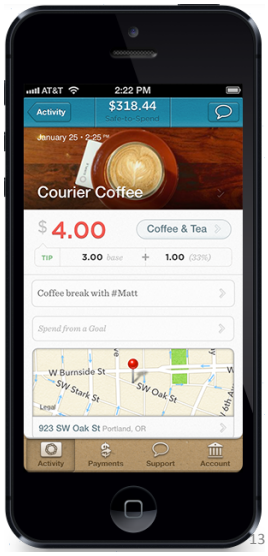
Gazpar (GDF)

Linky (EDF)

More data than we thought: bank transactions

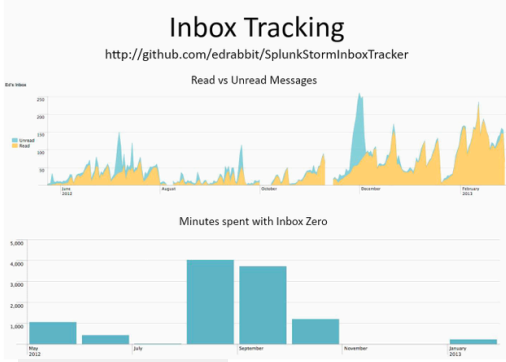
- The first historical database benchmark application
- (TPC benchmark: bank, branches, clients, accounts, tellers, transactions etc.)

- Simple.com: bare-bones financial services + access to transaction data
  - 80 attributes per transaction
  - Banks typically use only 4 ("Date/Libellé/ Euros")
  - Applications to exploit the complete transaction data



Human-produced data / Quantified Self

- "How big is your unread inbox?"
- "How did your heart rate change while exercising?"
- Social media content (including professional use), company intranets, photos etc.



Capturing and storing human user activity

- Gordon Bell (Microsoft), 2009
  http://edition.cnn.com/2009/TECH/09/25/total.recall.microsoft.bell/index.html?eref=rss_latest
- "video equipment, cameras and audio recorders to capture his conversations, commutes, trips and experiences. […] SenseCam that would hang around a person's neck and automatically capture every detail of life in photo form.

[…] saves everything -- from restaurant receipts (he takes pictures of them) to correspondence, bills and medical records. He makes PDF files out of every Web page he views.

[…] more than 350 gigabytes worth, not including the streaming audio and video -- is a replica of Bell's biological memory.
It's actually better, he says, because, if you back up your data in enough places, this digitized "e-memory" never forgets.

Gordon Bell, Microsoft, 2009

"It's like having a multimedia transcript of your life.

By about 2020 […] our entire life histories will be online and searchable.

Location-aware smartphones and inexpensive digital memory storage in the "cloud" of the Internet make the transition possible and inevitable.
No one will have to fret about storing the details of their lives in their heads anymore. We'll have computers for that.

And this revolution will "*change what it means to be human*"

Where does the data volume come from? (2)

- Machine-produced data
  - Log data from all kind of servers
  - Real world devices: payments, telecom, energy, weather, transportation, shipment…

  Sensors, including on (US) highways and trains

Gazpar (GDF) 

 Linky (EDF)

Where does the data volume come from? (2)

- E.g. french railway system: surveillance trains for the normal and high-speed lines
- TGV specially equipped for measuring while circulating at 320 km/h:
  - rail geometry
  - train/rail interaction
  - rail signalization and communication devices
  - electric power availability etc.
  - 150 sensors, 20 cameras
  - complete tour of France's high speed network in 6 days

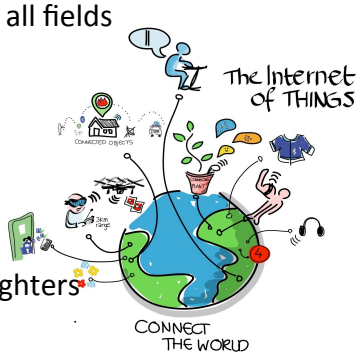
Machine-produced data: « Internet of Things » (IoT)

"The **Internet of Things** (**IoT**) is the interconnection of uniquely identifiable devices over the Internet

Expected to usher in **automation** in nearly all fields

Sample connected "things":

- Fridge → supermarket ☺
- heart monitoring implants
- biochip transponders on farm animals
  automobiles with built-in sensors
- field operation devices that assist fire-fighters



Already there:

- smart thermostat systems
- washer/dryers using wifi for remote monitoring.

Machine-produced data: « Internet of Things » (IoT)

"The **Internet of Things** (**IoT**) is the interconnection of uniquely identifiable devices over the Internet

Expected to usher in **automation** in nearly all fields

Sample connected "things":

- Fridge
- heart
- biochi
  autor
- field o

Already there:

- smart thermostat systems
- washer/dryers using wifi for remote monitoring.

"All the information gathered by all the sensors in the world isn't worth very much if there isn't **an infrastructure in place to analyze it in real time**.
Cloud-based applications are the key to using leveraged data. The Internet of Things doesn't function without cloud-based applications to interpret and transmit the data coming from all these sensors."
https://www.wired.com/2014/11/the-internet-of-things-bigger/

Defining Big Data: the V's

- Volume
  - Scale
- Velocity
  - Speed of producing and consuming the data
- Variety
  - Very different sources and data types
- Veracity
  - Is the data correct / certain / true?

Big Data velocity

- How much data is produced e.g. per second
- Data enters a **pipeline** consisting of storage and/or processing
  - Store-then-process: for off-line data analysis. Storage by itself is a challenge sometimes, e.g. data links to/from clouds are rather slow
  - Process-then-store: for data whose interest is maximized upon arrival (real-time processing)
  - Process-then-discard: sensor/monitoring (if nothing happens)

Sample high-throughput data streams

- French IT company runs a data center of 2000 servers
- 5000 energy efficiency indicators (temperature, electricity consumption etc.) are measured every 20 seconds x 50 Kb per measure result = 170 Terabytes / year

- Currently unable to store the data all data → sample (measure more rarely)
- May miss important things when they happen

The importance of current/recent data

- Real-time applications work only / mostly with **the latest data**
  - Embedded control mechanisms based on sensor data, e.g., "*this railway wagon component is breaking*" (now!)
  - Intrusion or malfunctioning detection…
- Keeping **humans engaged**
  - Customer relationship management *while the client is on the phone* with the customer service (see: “ordering a pizza in the future” video)
  - Recommending places where your friends are hanging out *now*

Defining Big Data: the V's

- Volume
  - Scale
- Velocity
  - Speed of producing and consuming the data
- Variety
  - Different sources, data formats, data types
- Veracity
  - Is the data correct / certain / true?

Big data heterogeneity (variety)

- Each new data type has added up on the old ones
- Enterprise data typically has high per-byte value ($/byte)
  - Hard to explain that "we will not need this database in the future"
- In many areas, legal obligation to keep old data (e.g. railway sensors, telecom, commercial…)

→ Data model & data management system soup
  - hierarchical, relational, object-oriented, XML, RDF, JSON, key-value pairs…

Sample relational database

Clients

NumClient	Nom	Adresse	Ville	Age
1	Julie	1 rue Dugommier	Paris	22
…	…	…	…	…

Comptes

NumCompte	Type	Découvert	NumClient
12345	Courant	1000	1
…	…	…	…

Transaction

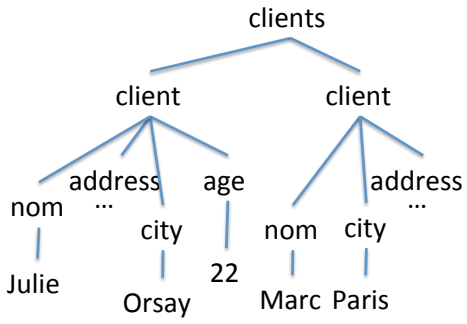
NumCompte	Montant	Date	Info
12345	-40,00	5/10/11	Retrait
12345	+23,45	6/10/11	Remb. MAIF
12345	-300,00	7/10/11	Chaussures

XML: extensible markup language

W3C, 2008

clients.xml:

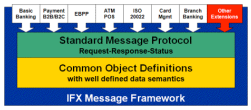
```
<clients>
 <client><nom>Julie</nom>
  <address>1, rue Dugommier</address>
  <city>Paris</city><age>22</age>
 </client>
 <client><nom>Marc</nom>…
 </client>
</clients>
```



Flexible
Platform-independent
Separate content from presentation
Schema possible (not compulsory)

XML applications

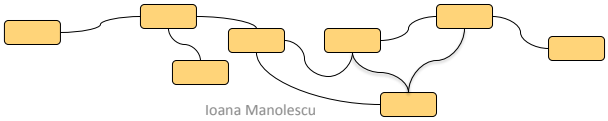
- Main language for the Web: **XHTML, XML Schema, SVG, RSS,** …
- Web Services: **SOAP, WSDL, BP4WS**
- **MathML** (mathematical markup language)
- **CML** (chemical markup language)
- **SMILE** (synchronized multimedia integration language)
- Financial Exchange (**IFX**)
- The Text Encoding Initiative (**TEI**)



Critique of XML

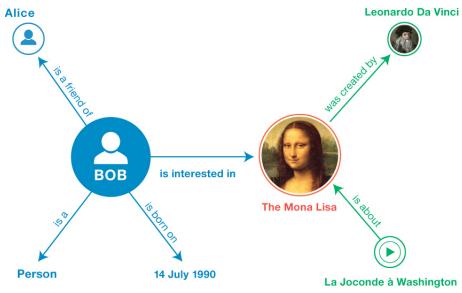
- Each information ends up in only one place
- OK for "classification" applications, structured text
- Fundamentally restrictive for **data = real world**!

Tim Berners-Lee, WWW proposal, CERN, 1998:
"*Many systems are organised hierarchically. A tree has the practical advantage of giving every node a unique name. However,* ***it does not allow the system to model the real world***."
(On newsgroups)*: "Typically, a discussion under one newsgroup will develop into a different topic, at which point* ***it ought to be in a different part of the tree***."



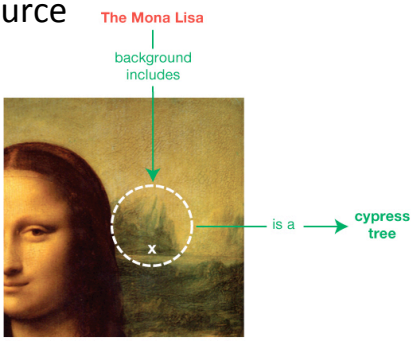
Graph data format for the Web: RDF (see also pre-requisite RDF course!)

- Resource Description Format, W3C, 2003
- Resources have properties with values.
- URIs (Universal Resource Identifiers) identify resources
- Resources, properties, or values may be specified by an URI.
- Properties and values may be constants



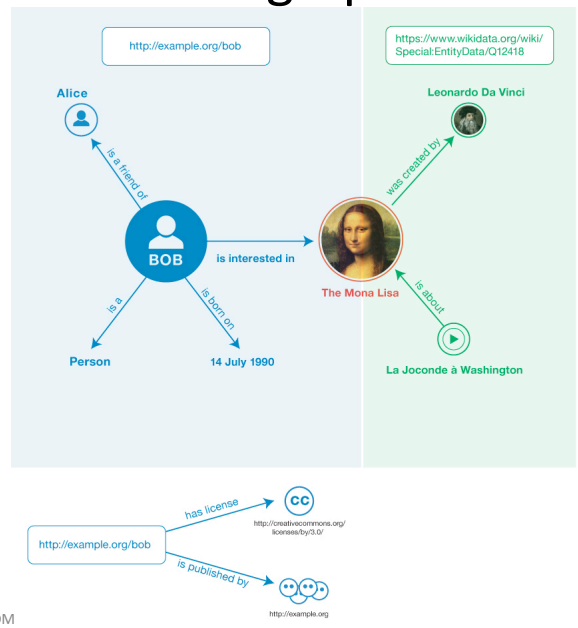
RDF feature: blank nodes

- Unnamed resource



- « Labeled null »

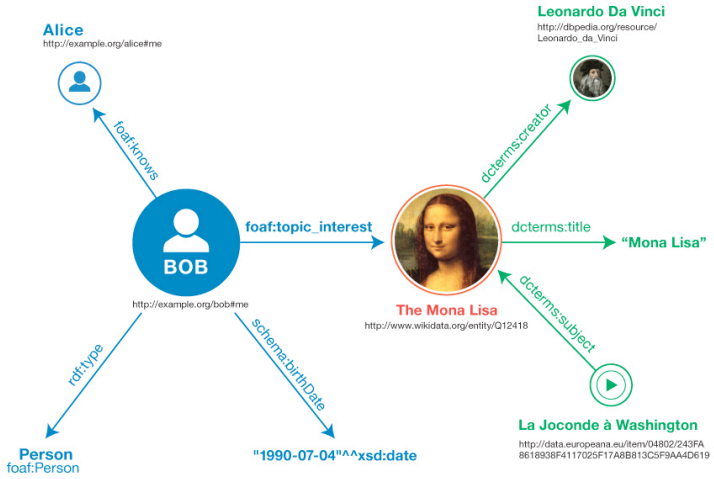
RDF graphs



RDF Schema constructs

Construct	Syntactic form	Description
Class (a class)	**C** rdf:type rdfs:Class	**C** (a resource) is an RDF class
Property (a class)	**P** rdf:type rdf:Property	**P** (a resource) is an RDF property
type (a property)	**I** rdf:type **C**	**I** (a resource) is an instance of **C** (a class)
subClassOf (a property)	**C1** rdfs:subClassOf **C2**	**C1** (a class) is a subclass of **C2** (a class)
subPropertyOf (a property)	**P1** rdfs:subPropertyOf **P2**	**P1** (a property) is a sub-property of **P2** (a property)
domain (a property)	**P** rdfs:domain **C**	domain of **P** (a property) is **C** (a class)
range (a property)	**P** rdfs:range **C**	range of **P** (a property) is **C** (a class)

Typed RDF graph

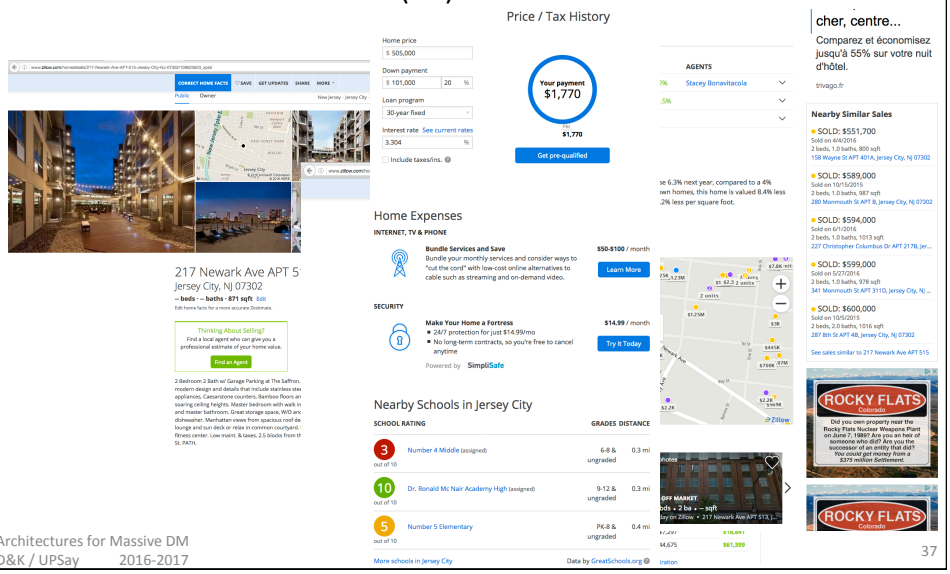


RDF reasoning

- RDF allow expressing **data *and* knowledge**
- Example:
  - if X *teaches a class*
  - then X *is a person*, *is an instructor*, *belongs to the school giving the class*, and *works for the university which includes the school*
- Reasoning exploits knowledge to infer *implicit data*

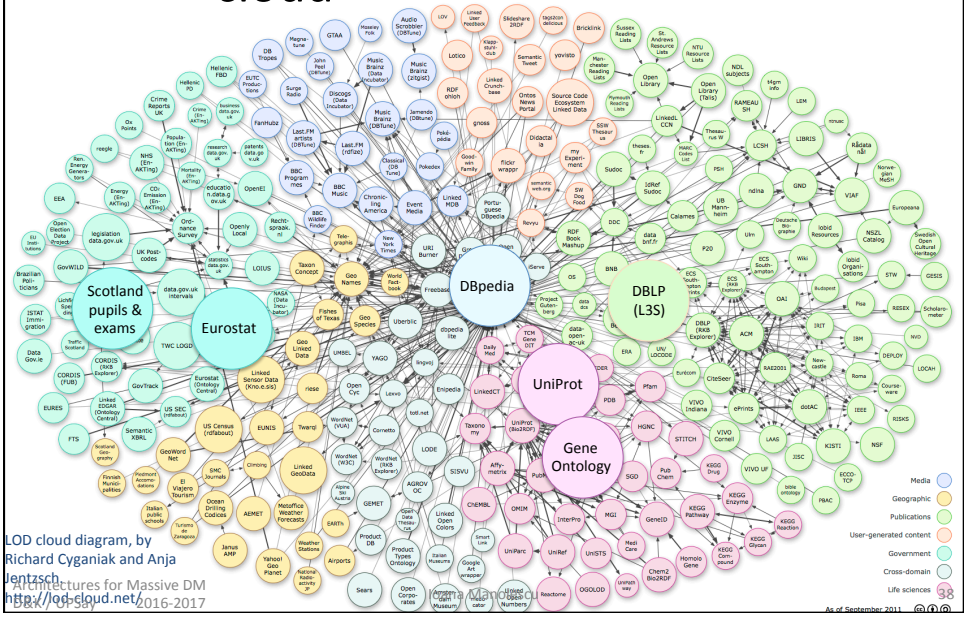
Varied Big Data has huge value potential

- Real estate ad from Zillow (US):



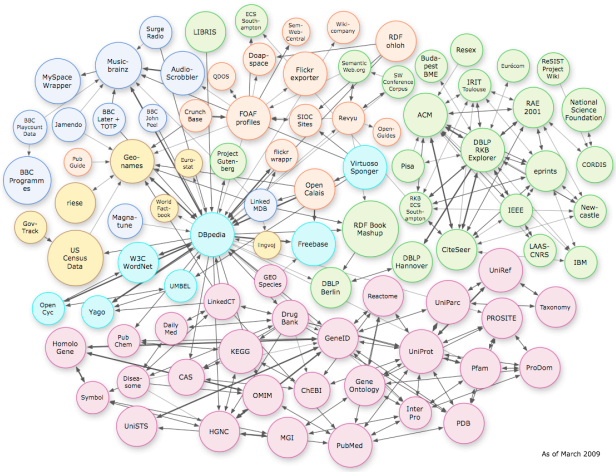
Linked Open Data (LOD) cloud

Linked Open Government Data project (logd.rw.rpi.edu): 10^{10} triples.

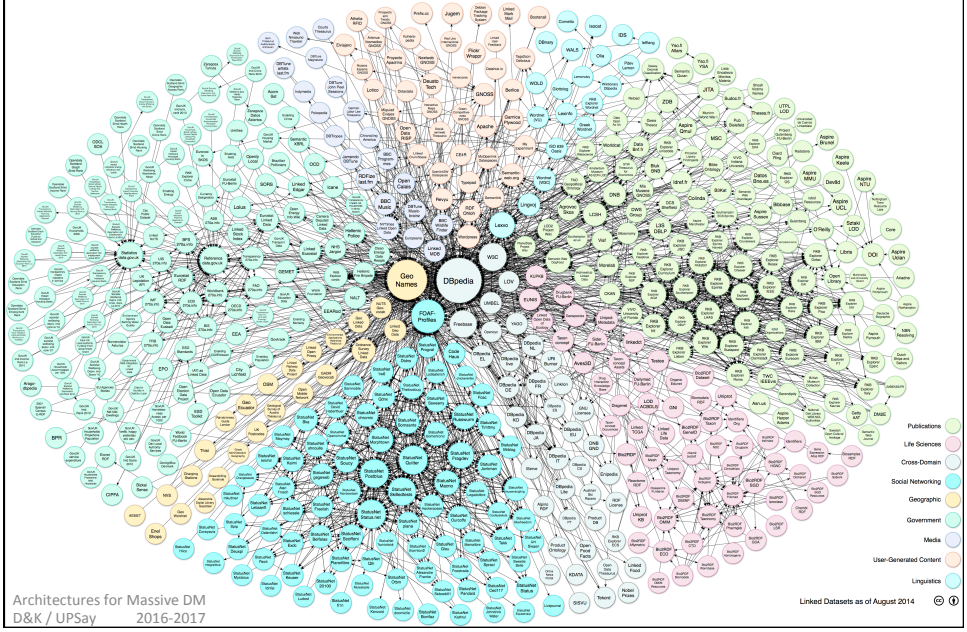


LOD cloud diagram, by Richard Cyganiak and Anja Jentzsch
http://lod-cloud.net/

LOD cloud 2009



LOD cloud 2014

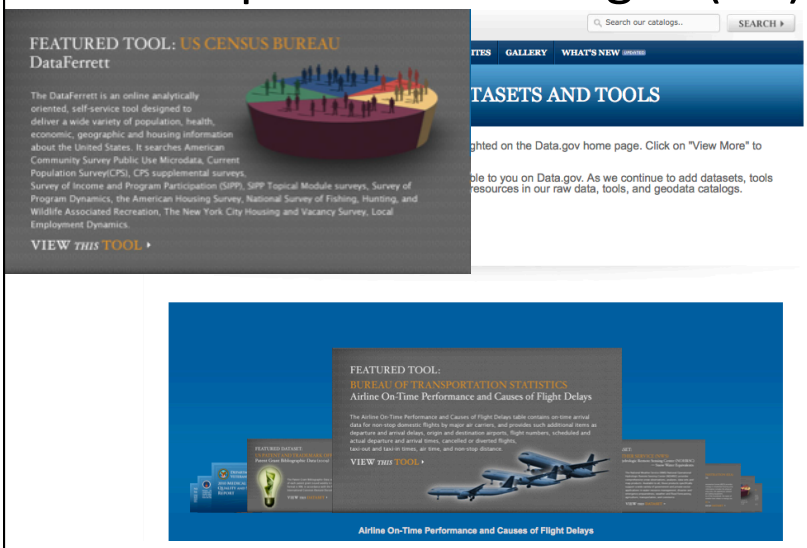


Open vs. linked data

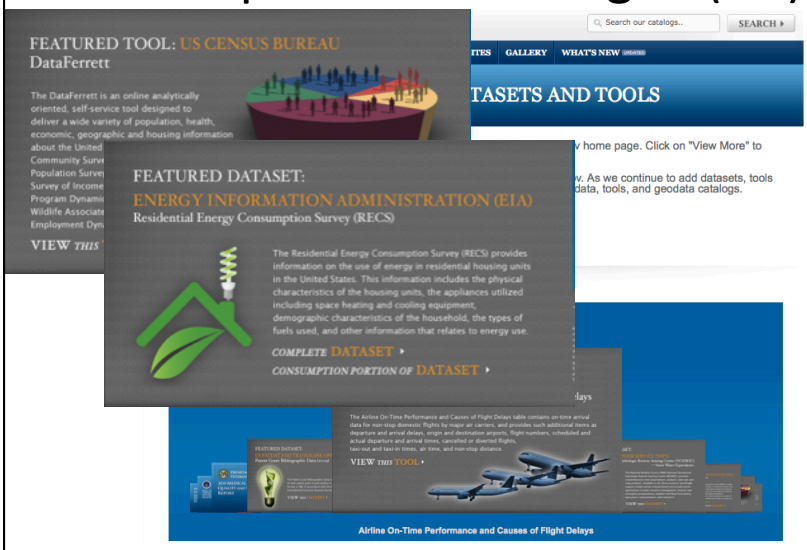
1. **Linked Data**:
   "recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using **URIs** and **RDF**"
   - (Tim Berners-Lee) vision for the Web
2. **Open Data**:
   "idea that certain data should be freely available to everyone to use and republish as they wish, without restrictions from copyright, patents or other mechanisms of control"
   - In principle, orthogonal to the Linked aspect
   - In practice, Linked is a technical mean toward Open

Open Data: data.gov (US)

Open Data: data.gov (US)



Open Data: data.gov (US)

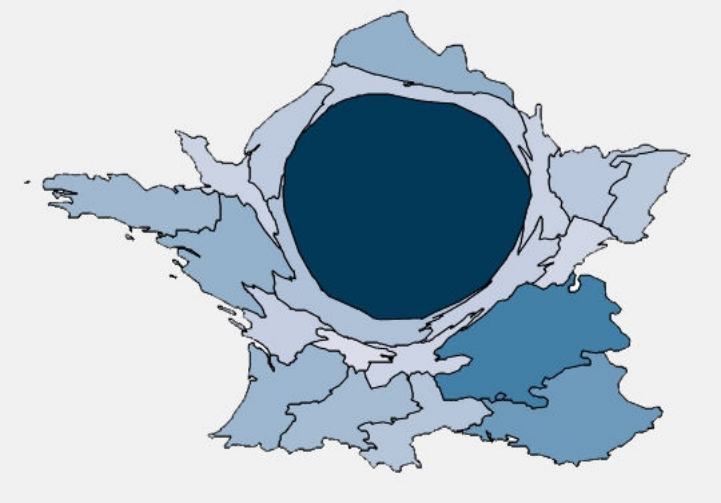


Open Data: data.gov (US)

Open Data: data.gov (US)

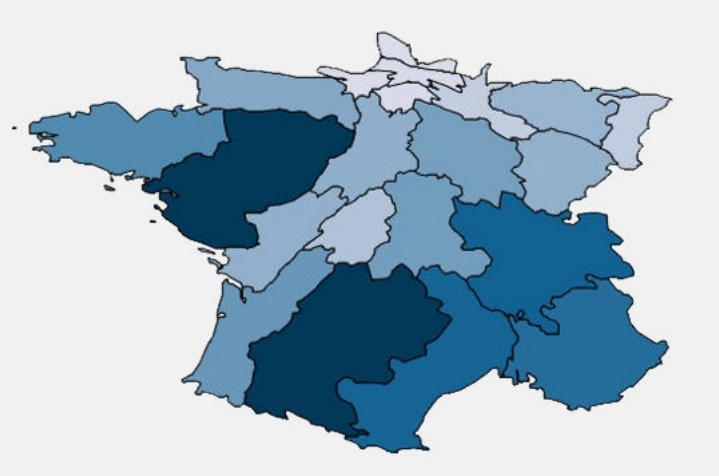
Open Data from Etalab (FR)

GDP per French region (Le Journal Du Net)



Open Data from Etalab (FR)

Organic agriculture per French region



Big data heterogeneity (variety)

Hierarchical, relational, object-oriented, XML, RDF, JSON, key-value pairs…

Data model & data management system soup

– hierarchical, relational, object-oriented, XML, RDF, JSON, key-value pairs…

Traditionally this has been solved (time and $ permitting) with **data migration** / **ETL** (extract-transform-load)

– Heterogeneous data and high throughput may make ETL impractical for Big Data

JavaScript Object Notation

Human-readable XML ☺

1. Object = set of (attribute, value) pairs
2. Array = list of values.
3. Value = string | number | true | false | null | object | Array

```
{"menu": {
  "header": "SVG Viewer",
  "items": [
    {"id": "Open"},
    {"id": "OpenNew", "label": "Open New"},
    {"id": "ZoomIn", "label": "Zoom In"},
    {"id": "ZoomOut", "label": "Zoom Out"},
    {"id": "OriginalView", "label": "Original View"},
    null,
    {"id": "Quality"},
    {"id": "Pause"},
    {"id": "Mute"},
    {"id": "Help"},
    {"id": "About", "label": "About CVG Viewer..."}
  ]
}}
```

Defining Big Data: the V's

- Volume
  - Scale
- Velocity
  - Speed of producing and consuming the data
- Variety
  - Very different sources and data types
- Veracity
  - Is the data correct / certain / true?

Big Data veracity

- Is this true? (What is the probability?)
- Contradictory sources (1 vs. 2 clocks)
- Errors in the data
  - Humans introduce many errors
  - Sensors may have failures or erroneous readings
    - Light or heating sensors in a building
    - Wear and tear
- Tackled by **data curation / cleaning / quality** tools for regular (or at least homogeneous) data

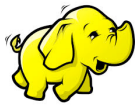
Big Data veracity

- Data reconciliation / entity extraction
- Large-scale ontologies such as YAGO, DBPedia, Google Knowledge Base (> Freebase)
  - Reference database of core facts

    - Places (city/country etc.), people (public figures, scientists, artists etc.), events (born, died, emigrated, was created…), time
  - Ontologies automatically extracted from Web and other specific sources → need for reconciliation

Big picture on big data

1. **Volume, velocity, variety, veracity**
2. Probably not all the data has the same **value** $/B
   - This is why existing databases will stay!
3. Very large, unstructured, uncertain-value data may fit in scalable, relatively slow systems (e.g. MapReduce/Hadoop)
   - Massive parallelism for dummies (to be seen)
   - Mining, batch, iterative processing



4. Large SW companies develop "database-style" layers on Map/Reduce, sometimes in Open Source
   - A bit too early to tell the benefits
5. Many flavors and brands of **parallelism**, **distribution** (and data models)

DATABASE FUNDAMENTALS (RECALL OR CRASH-COURSE INTRODUCTION)
Part 1: database properties

Which of these is/acts like a database?

From the user's perspective

MySQL	**Excel**	**Oracle**	**Hadoop**	**Google**	**GMail**
Facebook	**Twitter**	**Emacs**	**Skype**	**Firefox**	**Python**

Fundamental database properties: ACID

- **Atomicity**: either all operations involved in a transactions are done, or none of them is
  - E.g. bank payment
- **Consistency**: application-dependent constraint
  E.g. every client has a single birthdate
- **Isolation**: concurrent operations on the database are executed as if each ran alone on the system
  - E.g. if a debit and a credit operation run concurrently, the final result is still correct
- **Durability**: data will not be lost nor corrupted even in the presence of system failure during operation execution

Jim Gray, ACM Turing Award 1998 for « fundamental contributions to databases and transaction management »

Which of these is/acts like a database?

From the user's perspective

MySQL ✓	**Excel** ✗	**Oracle** ✓	**Hadoop** ✗	**Google**	**GMail**
Facebook ✓	**Twitter** ~	**Emacs** ✗	**Skype** ✗	**Firefox** ~	**Python** ~

Twitter, Skype and Firefox include / are built on database servers

Twitter: no delete; small data items

Skype: local database+index of all conversations, mirrorring the one from Microsoft. Gets corrupted ☹

Firefox: includes a tiny SQL server for the bookmarks

ACID properties

- **Atomicity**: per transaction (cf. boundaries)
- **Consistency**: difference in the expressive power of the constraints
  - E.g. SQL create table constraint syntax (MySQL):

CREATE TABLE tbl_name (create_definition,...) [table_options] [partition_options]

create_definition: *col_name column_definition* |
[**CONSTRAINT** [*symbol*]] **PRIMARY KEY** [*index_type*] (*index_col_name*,...) [*index_option*] ... |
{INDEX|KEY} [*index_name*] [*index_type*] (*index_col_name*,...) [*index_option*] ... |
[**CONSTRAINT** [*symbol*]] **UNIQUE** [**INDEX**|**KEY**] [*index_name*] [*index_type*]
(*index_col_name*,...) [*index_option*] (...) |
CHECK (*expr*)

column_definition: *data_type* [**NOT NULL** | **NULL**] [DEFAULT *default_value*]
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY] (...)

ACID properties

Consistency (continued)

- SQL constraint syntax (within create table):

[**CONSTRAINT** [*symbol*]] **FOREIGN KEY** [*index_name*]
(*index_col_name*, ...)
REFERENCES *tbl_name* (*index_col_name*,...)
[**ON DELETE** *reference_option*]
[**ON UPDATE** *reference_option*]
reference_option: **RESTRICT | CASCADE | SET NULL | NO ACTION**

- Key-value store: REDIS
  - a data item can have only one value for a given property
- Key-value store: DynamoDB
  - The value of a data item can be constrained to be unique, *or* allowed to be a set
- Hadoop File System (HDFS): no constraints

ACID properties

- **Isolation**: concurrent operations on the database are executed as if each ran alone on the system
  - Watch out for: read-write (RW) or write-write (WW) conflicts
  - Conflict granularity depends on the data model
- **An example of advanced isolation support: SQL**
  - E.g. SQL

Isolation Level	Dirty Read	Non Repeatable Read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Snapshot	No	No	No
Serializable	No	No	No

  - High isolation conflicts with high transaction throughput
  - E.g. HDFS: a file is never modified (written only once and integrally)

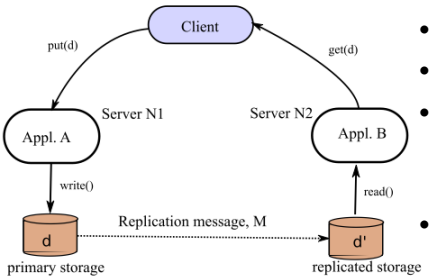
Limits of ACIDity in large distributed systems: the **CAP theorem**

- Eric Brewer, « Symposium on Principles of Distributed Computing », 2000 (conjecture)
- Proved in 2002

- No distributed system can simultaneously provide

1. **Consistency** (all nodes see the same data at the same time)
2. **Availability** (node failures do not prevent survivors from continuing to operate)
3. **Partition tolerance** (the system continues to operate despite arbitrary message loss)

CAP theorem by example



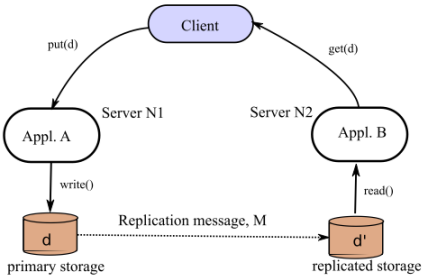
- Primary and replica store
- Applications A and B on servers
- Client writes a new d value through A, which propagates d to the replica (replacing the old d')
- Subsequently, client reads from B

What if a failure occurs in the system?

Communication missed between primary and replica

1. If we want **Partition tolerance** (let the system function) → the Client reads old data (**no Consistency**)
2. If we want **Consistency**, e.g. make the write+replica msg an atomic transaction (to avoid missed communications) → **no Availability** (we may wait for the msg forever if failure)

CAP theorem: what can we do?

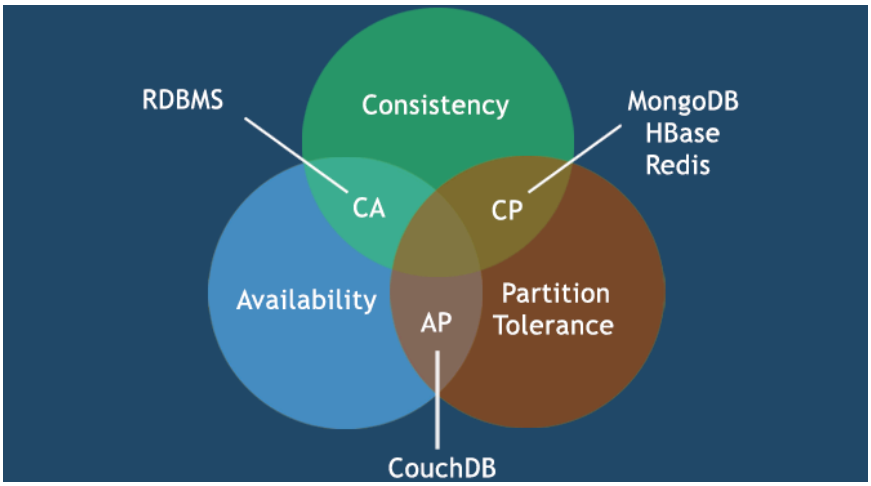


- **Partition tolerance**: we must have it (cannot block if one machine fails)
- Then one must *trade some consistency for availability*

Eventual consistency model:

- The replication message is asynchronous (non-blocking)
- N1 keeps sending the message until acknowledge by N2 (*eventually* the replica and primary store are consistent)
- In the mean time, the client works on inconsistent data (« I had already removed this from the basket once! »)

NoSQL systems vs. CAP theorem

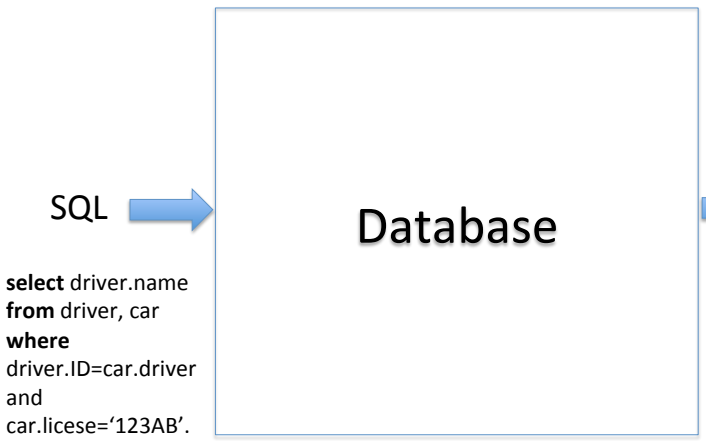


Modern systems (e.g. NoSQL) arose exactly because of the partition tolerance is a must in large-scale distributed systems

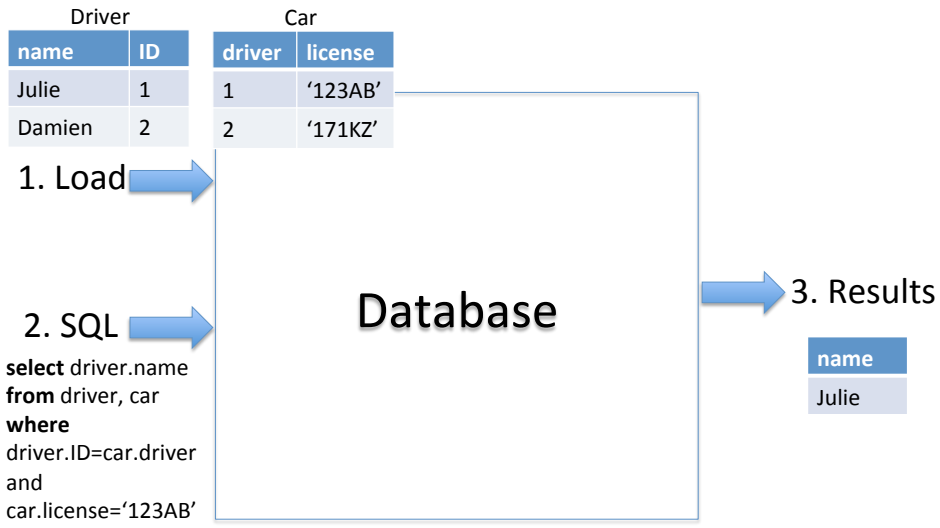
65

DATABASE FUNDAMENTALS RECALL (OR CRASH-COURSE INTRODUCTION)
Part 2: database internals

What's in a database?



What's in a database?



What's in a database?

Driver

name	ID
Julie	1
Damien	2

Car

driver	license
1	'123AB'
2	'171KZ'

1. Load

2. SQL

select driver.name
from driver, car
where
driver.ID=car.driver
and
car.license='123AB'

Database

Storage system (**disk**, memory, SSD…)

3. Results

name
Julie

What's in a database?

1. Load

2. SQL

select driver.name
from driver, car
where
driver.ID=car.driver
and
car.license='123AB'

Database

Driver

name	ID
Julie	1
Damien	2

Car

driver	license
1	'123AB'
2	'171KZ'

3. Results

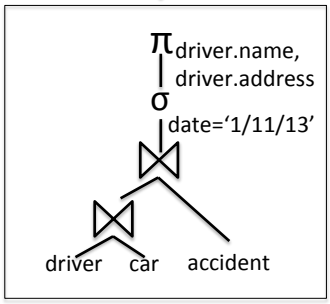
name
Julie

What's in a database?

SQL

select driver.name, driver.address
from driver, car, accident
where driver.ID=car.driver and car.license=accident.carLicense and accident.date='1/11/13'

select… from driver, car, accident where…



Query language

Logical plan

Driver

name	ID
Julie	1
Damien	2

Accident

driver	license
1	'123AB'
2	'171KZ'

Car

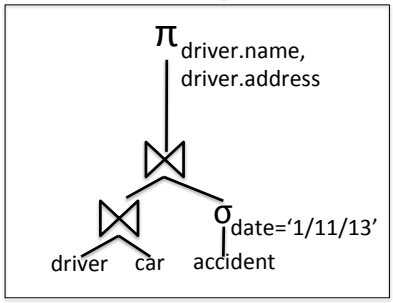
Results

What's in a database?

SQL

select driver.name, driver.address
from driver, car, accident
where driver.ID=car.driver and car.license=accident.carLicense and accident.date='1/11/13'

select… from driver, car, accident where…



Query language

Logical plan 1

Logical plan 2

Driver

name	ID
Julie	1
Damien	2

Accident

driver	license
1	'123AB'
2	'171KZ'

Car

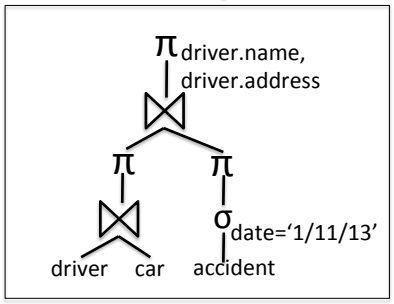
Results

What's in a database?

SQL

select driver.name, driver.address
from driver, car, accident
where driver.ID=car.driver and car.license=accident.carLicense and accident.date='1/11/13'

select… from driver, car, accident where…



Driver

name	ID
Julie	1
Damien	2

Accident

driver	license
1	'123AB'
2	'171KZ'

Car

Query language

Logical plan 1

Logical plan 2

Logical plan 3

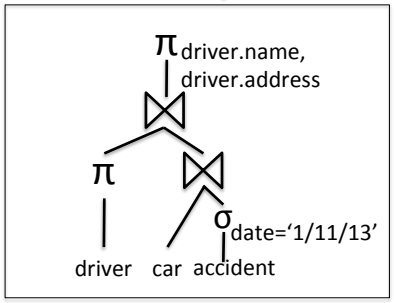
Results

What's in a database?

SQL

select driver.name, driver.address
from driver, car, accident
where driver.ID=car.driver and car.license=accident.carLicense and accident.date='1/11/13'

select… from driver, car, accident where…



Driver

name	ID
Julie	1
Damien	2

Accident

driver	license
1	'123AB'
2	'171KZ'

Car

Query language

Logical plan 1

Logical plan 2

Logical plan 3

Logical plan 4

Results

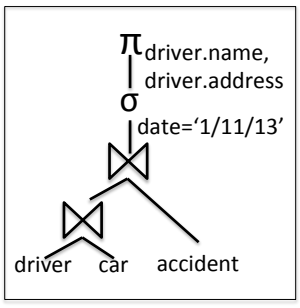
Logical query optimization

- Enumerates logical plans
- All logical plans compute the query result
  - They are **equivalent**
- Some are (much) more **efficient** than others
- **Logical optimization**: moving from a plan to a more efficient one
  - Pushing selections
  - Pushing projections
  - Join reordering: most important source of optimizations

Logical query optimization example

1.000.000 cars, 1.000.000 drivers, 1.000 accidents, 2 cars per accident, 10 accidents on 1/11/13

« Name and address of drivers in accidents on 1/1/2013? »



$10^{12} + 10^9 + 2000 + 20$ operations ~ $\mathbf{10^{12}} + 10^9$

Cost of an operator: depends on the number of tuples (or tuple pairs) which it must process
e.g. c_disk x number of tuples read from disk
e.g. c_cpu x number of tuples compared

Cardinality of an operator's output: how many tuples result from this operator produce

The cardinality of one operator's output determines the cost of its parent operator!

Plan cost: the sum of the costs of all operators in a plan

Total scan costs: $10^6+10^6+10^3$

Driver-car join cost estimation: $10^6 \times 10^6 = 10^{12}$

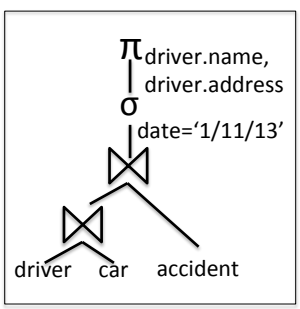
Driver-car join size estimation: 10^9

Driver-car-accident cost estimation: $10^9 \times 10^3 = 10^{12}$

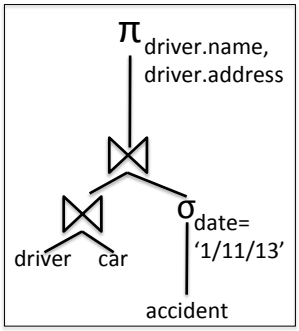
Logical query optimization example

1.000.000 cars, 1.000.000 drivers, 1.000 accidents, 2 cars per accident, 10 accidents on 1/11/13

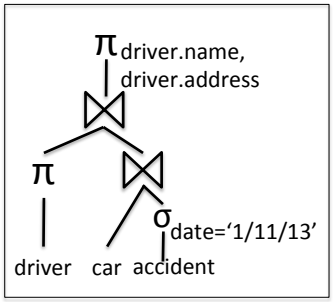
« Name and address of drivers in accidents on 1/1/2013? »



$10^{12} + 10^9 + 2000 + 20$ operations ~ $\mathbf{10^{12}} + 10^9$



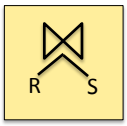
$10^{12} + 1000 + 10^7 + 20$ operations ~ $\mathbf{10^{12}} + 10^3$



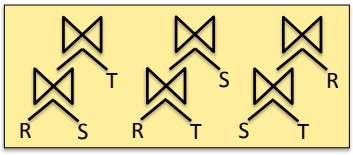
$10^6 + 10^6 + 10^7 + 2*10^7 + 20$ operations ~ $\mathbf{3*10^7}$

Join ordering is the main problem in logical query optimization

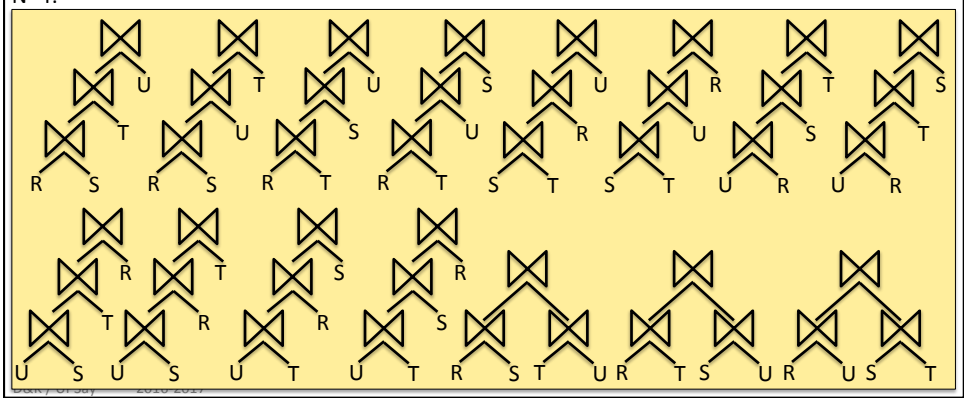
N=2:



N=3:



N=4:



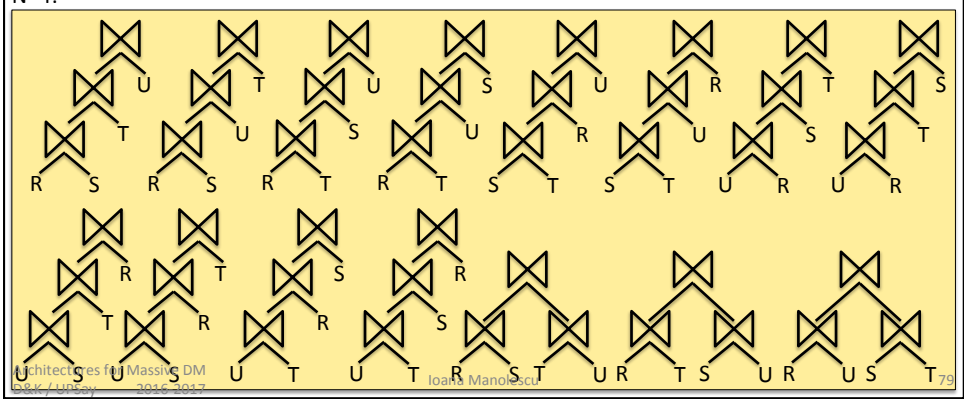
Join ordering is the main problem in logical query optimization

$$Plans(n+1) = (n+1) * Plans(n) + \tfrac{1}{2} * \Sigma_{i=1}^{(n/2)} Plans(i) * Plans(n+1-i)$$

High (exponential) complexity → many heuristics

- Exploring only left-linear plans etc.

N=4:



Logical query optimization needs statistics

Exact statistics:

- 1.000.000 cars, 1.000.000 drivers, 1.000 accidents

Approximate / estimated statistics:

- 2 cars per accident, 10 accidents on 1/11/13

Statistics are gathered

- When **loading** the data: take advantage of the scan
- **Periodically** or upon **request** (e.g. analyze in the Postgres RDBMS)
- At **runtime**: modern systems may do this to change the data layout (e.g., dynamic indexing – to be seen next)

Statistics on the base data vs. on results of operations not evaluated (yet):

- « On average 2 cars per accident »
- For each column R.a, store:
  |R|, |R.a| (number of distinct values), min{R.a}, max{R.a}
- Assume **uniform distribution** in R.a
- Assume **independent distribution**
  - of values in R.a vs values in R.b; of values in R.a vs values in S.c
- + simple probability computations

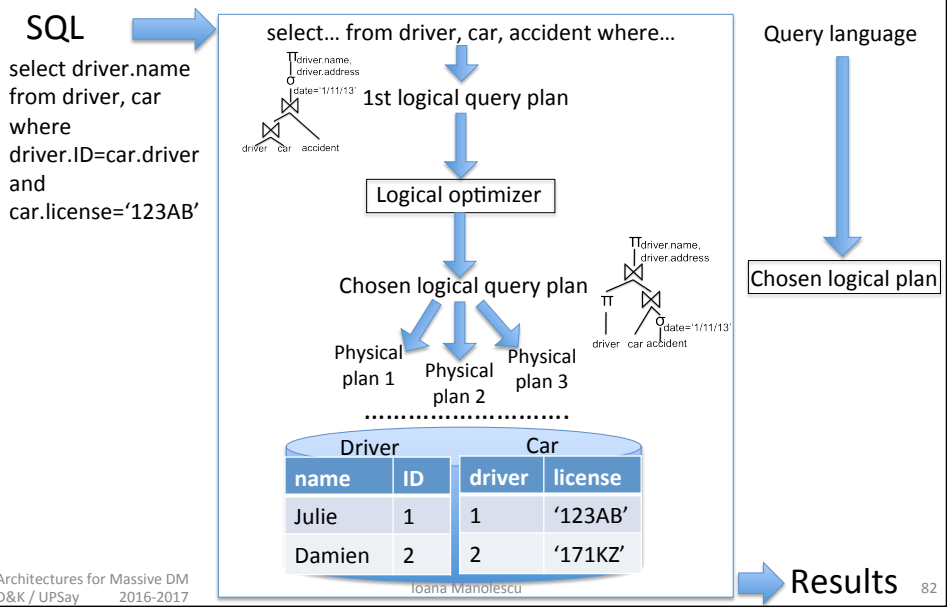
More on statistics

- For each column R.a, store:
  |R|, |R.a| (number of distinct values), min{R.a}, max{R.a}
- Assume **uniform distribution** in R.a
- Assume **independent distribution**
  - of values in R.a vs values in R.b; of values in R.a vs values in S.c

- The **uniform distribution** assumption is frequently wrong
  - Real-world distribution are skewed (popular/frequent values)
- The **independent distribution** assumption is sometimes wrong
  - « Total » counter-example: *functional dependency*
  - Partial but strong enough to ruin optimizer decisions: *correlation*

- Actual optimizers use more sophisticated statistic informations
  - **Histograms**: equi-width, equi-depth
  - Trade-offs: size vs. maintenance cost vs. control over estimation error

Database internal: query optimizer



Physical query plans

Made up of **physical operators** =

algorithms for implementing logical operators

Example: equi-join (R.a=S.b)

Nested loops join:
```
foreach t1 in R{
  foreach t2 in S {
    if t1.a = t2.b then output (t1 || t2)
  }
}
```

Merge join: // requires sorted inputs
```
repeat{
 while (!aligned) { advance R or S };
 while (aligned) { copy R into topR, S into topS };
 output topR x topS;
} until (endOf(R) or endOf(S));
```

Hash join: // builds a hash table in memory
```
While (!endOf(R)) { t ← R.next; put(hash(t.a), t); }
While (!endOf(S)) { t ← S.next;
                    matchingR = get(hash(S.b));
                    output(matchingR x t);
                  }
```


Physical query plans

Made up of **physical operators** =

algorithms for implementing logical operators

Example: equi-join (R.a=S.b)

Nested loops join: O(|R|x|S|)
```
foreach t1 in R{
  foreach t2 in S {
    if t1.a = t2.b then output (t1 || t2)
  }
}
```

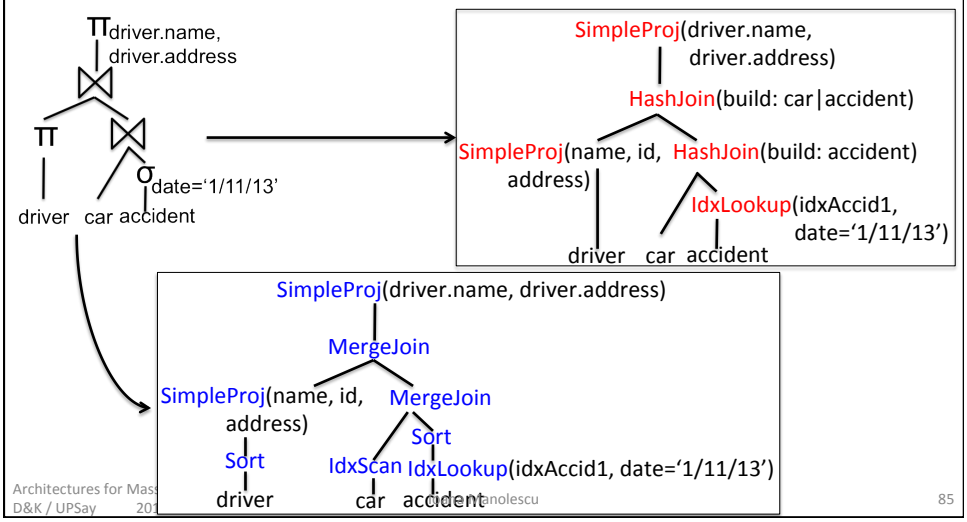
Merge join: // requires sorted inputs O(|R|+|S|)
```
repeat{
 while (!aligned) { advance R or S };
 while (aligned) { copy R into topR, S into topS };
 output topR x topS;
} until (endOf(R) or endOf(S));
```

Hash join: // builds a hash table in memory
```
While (!endOf(R)) { t ← R.next; put(hash(t.a), t); }
While (!endOf(S)) { t ← S.next;
                    matchingR = get(hash(S.b));
                    output(matchingR x t);
                  }
```
O(|R|+|S|)

Also:
Block nested loops join
Index nested loops join
Hybrid hash join
Hash groups / teams
…

Physical optimization

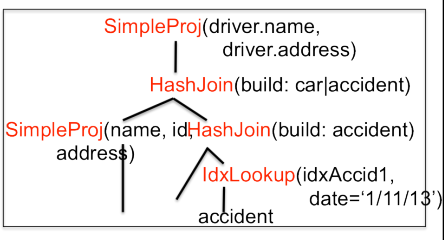
Possible physical plans produced by physical optimization for our sample logical plan:



Physical plan performance

Metrics characterizing a physical plan

- **Response time**: between the time the query starts running to the we know it's end of results
- **Work** (resource consumption)
  - How many **I/O** calls (blocks read)
    - Scan, IdxScan, IdxAccess; Sort; HybridHash (or spilling HashJoin)
  - How much **CPU**
    - All operators
  - (Distributed plans: **network** traffic)
- **Total work**: work made by all operators



Query optimizers in action

Most database management systems have an « explain » functionality → physical plans. Below sample Postgres output:

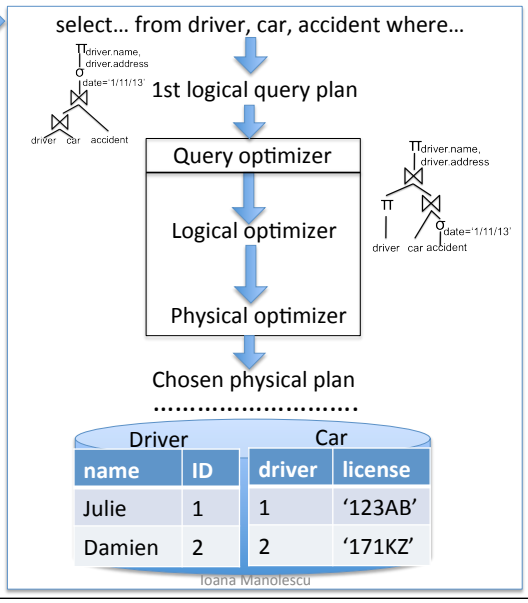
```
EXPLAIN SELECT * FROM tenk1;
                        QUERY PLAN
-------------------------------------------------------------
Seq Scan on tenk1 (cost=0.00..458.00 rows=10000 width=244)
```

```
EXPLAIN SELECT * FROM tenk1 t1, tenk2 t2
WHERE t1.unique1 < 100 AND t1.unique2 = t2.unique2;
                                QUERY PLAN
-------------------------------------------------------------------------------------
Hash Join (cost=232.61..741.67 rows=106 width=488)
  Hash Cond: ("outer".unique2 = "inner".unique2)
  -> Seq Scan on tenk2 t2 (cost=0.00..458.00 rows=10000 width=244)
  -> Hash (cost=232.35..232.35 rows=106 width=244)
       -> Bitmap Heap Scan on tenk1 t1 (cost=2.37..232.35 rows=106 width=244)
       Recheck Cond: (unique1 < 100)
       -> Bitmap Index Scan on tenk1_unique1 (cost=0.00..2.37 rows=106 width=0)
            Index Cond: (unique1 < 100)
```


Database internal: physical plan

SQL

select driver.name
from driver, car
where
driver.ID=car.driver
and
car.license='123AB'



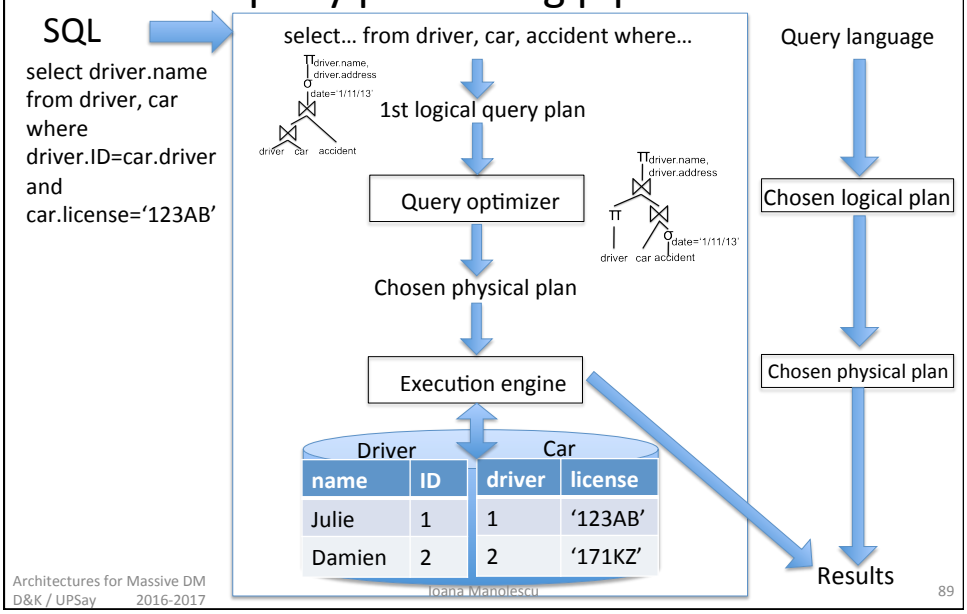
Query language

Chosen logical plan

Chosen physical plan

Results

Database internals: query processing pipeline



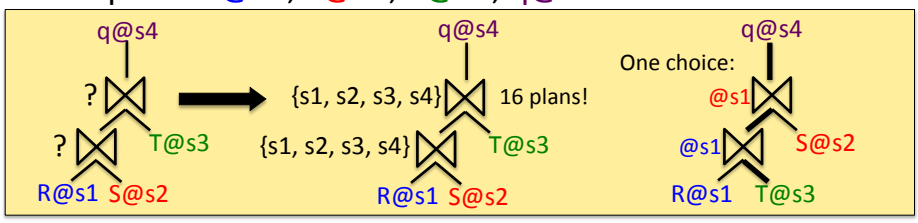
DISTRIBUTED DATA MANAGEMENT SYSTEMS

Oldest scenario: distributed databases

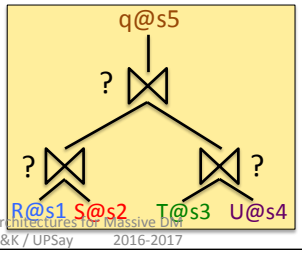
- Many *nodes* or *sites* (or peers…)
- Data is distributed
  - **Data catalog**: information on which data is stored where
    - Explicit : « All Paris sales are stored in Paris ».
      Ex: Relational table fragmentation (horizontal, vertical etc.)
      Catalog stored at a master/central server.
    - Implicit: « Data is distributed by the value of the city »
      (« somewhere »)
      Catalog split across all sites (P2P) or at a master (HadoopFS)
- Queries are distributed (may come from any site)
- Query processing is distributed
  - Operators may run on different sites → network transfer
  - Another layer of complexity to the optimization process

Distributed query optimization

Example 1: R@s1, S@s2, T@s3, q@s4



Example 2: R@s1, S@s2, T@s3, U@s4, q@s5

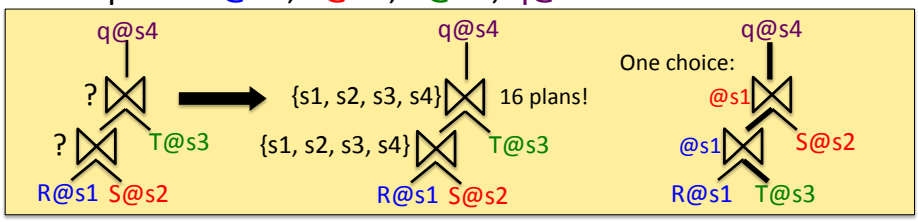


Plan pruning criteria if all the sites and network connections have equal performance

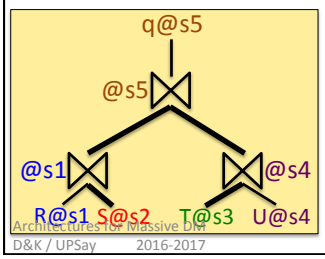
- *Ship the smaller collection*

Distributed query optimization

Example 1: R@s1, S@s2, T@s3, q@s4



Example 2: R@s1, S@s2, T@s3, U@s4, q@s5

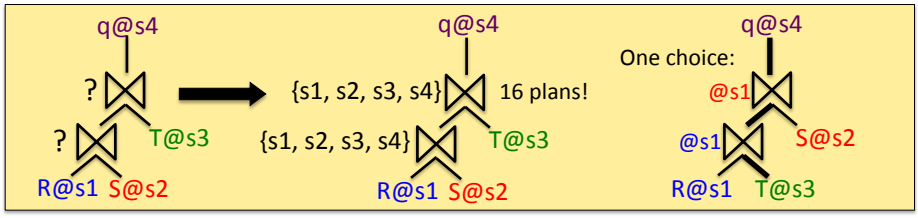


Plan pruning criteria if all the sites and network connections have equal performance:

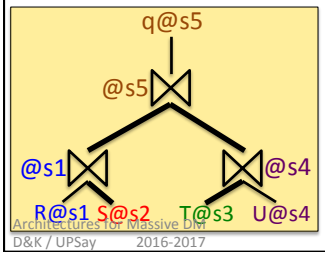
- *Ship the smaller collection.*
- *Transfer to join partner or the query site*

Distributed query optimization

Example 1: R@s1, S@s2, T@s3, q@s4



Example 2: R@s1, S@s2, T@s3, U@s4, q@s5



Plan pruning criteria if all the sites and network connections have equal performance:

- *Ship the smaller collection.*
- *Transfer to join partner or the query site*

This plan illustrates total effort != response time

Dimensions of distributed systems

- **Data model**:
  - Relations, trees (XML, JSON), graphs (RDF, others…), nested relations
  - Query language
- **Heterogeneity** (DM, QL): none, some, a lot
- **Scale**: small (~10-20 sites) or large (~10.000 sites)
- **ACID** properties
- **Control**:
  - Single master w/complete control over N slaves (Hadoop/HDFS)
  - Sites publish independently and process queries as directed by single master/*mediator*
  - Many-mediator systems, or peer-to-peer (P2P) with *super-peers*
  - Sites completely independent (P2P)

Distributed relational databases

- **DM**: relations; **language**: SQL; **ACID**: cf. SQL standard
- **Heterogeneity**: none
- **Control**:

Servers DB1@site1: R1(a,b), S1(a,c)

Server DB2@site2: R2(a,b), S2(a,c),

Server DB3@site3: R3(a,b),
S3(a,c) defined as:

```
select * from DB1.S1 union all
select * from DB2.S2 union all
select R1.a as a, R2.b as c from DB1.R1 r1, DB2.R2 r2
where r1.a=r2.a
```

Site3 decides what to import from site1, site2 (« hard links »)

Site1, site2 are independent servers

Also: replication policies, distribution etc. (usually with one or a few masters)

- **Size**: small