1A Databases
Lecture 8

Timothy G. Griffin

Computer Laboratory
University of Cambridge, UK

Michaelmas 2016

Lecture 8

- Sarah Mei's blog
- **O**n**L**ine **A**nalytical **P**rocessing (**OLAP**)
- **O**n**L**ine **T**ransaction **P**rocessing (**OLTP**)
- Cloud computing and distributed databases
- Column-oriented databases
- **C**onsistency, **A**vailability, **P**artition tolerance (**CAP**)

MongoDB is a popular document-oriented DBMS

Sarah Mei

« What Your Conference Proposal Is Missing

How To Prevent Inappropriat Presentations

Why You Should Never Use MongoDB

Disclaimer: I do not build database engines. I build web applications. I run 4-6 different projects every year, so I build **a lot** of web applications. I see apps with different requirements and different data storage needs. I've deployed most of the data stores you've heard about, and a few that you probably haven't.

Blog posted 11 November, 2013.

`http://www.sarahmei.com/blog/2013/11/11/why-you-should-never-use-mongodb`

Quotes from Sarah Mei's blog

- For quite a few years now, the received wisdom has been that **social data is not relational**, and that if you store it in a relational database, you're doing it wrong.
- Diaspora chose MongoDB for their social data in this zeitgeist. It was not an unreasonable choice at the time, given the information they had.
- You can see why this is attractive: all the data you need is already located where you need it.

Note

The blog describes implementation decisions made in 2010 for the development of a social networking platform called Diaspora. This was before mature graph-oriented databases were available.

Quotes from Sarah Mei's blog (2)

- You can also see why this is dangerous. Updating a user's data means walking through all the activity streams that they appear in to change the data in all those different places. This is very error-prone, and often leads to inconsistent data and mysterious errors, particularly when dealing with deletions.
- If your data looks like that, you've got documents. Congratulations! It's a good use case for Mongo. But if there's value in the links between documents, then you don't actually have documents. MongoDB is not the right solution for you. It's certainly not the right solution for social data, where links between documents are actually the most critical data in the system.

The project eventually migrated to a relational database (using about 50 tables). See `https://github.com/diaspora/diaspora`.

Yet another class of read-oriented databases

OLAP vs. OLTP

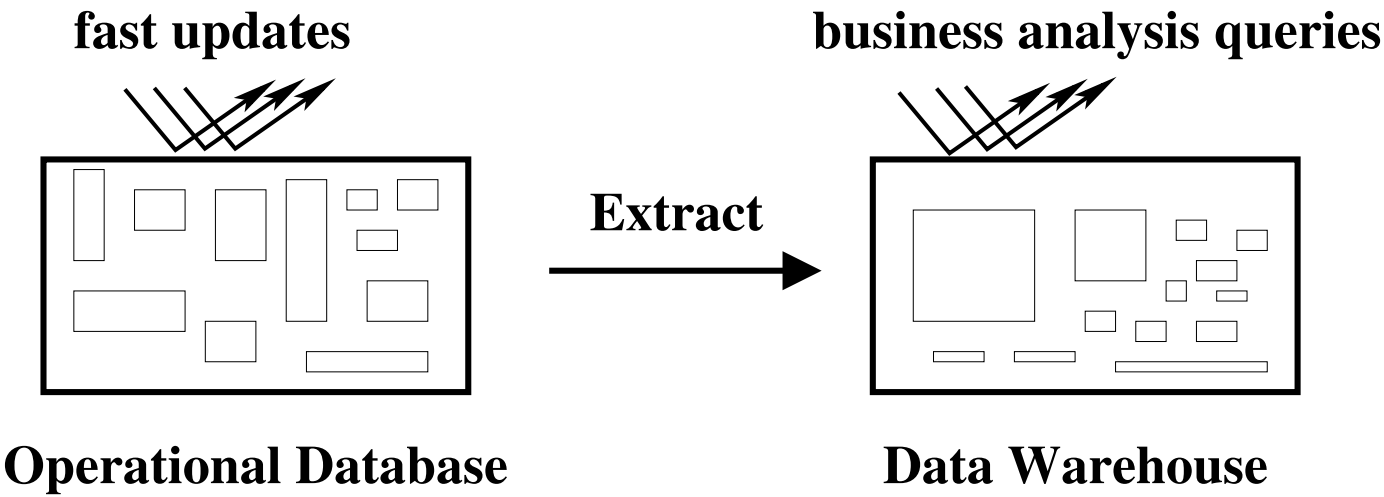
OLTP Online Transaction Processing

OLAP Online Analytical Processing

- Commonly associated with terms like Decision Support, Data Warehousing, etc.

	OLAP	**OLTP**
Supports	analysis	day-to-day operations
Data is	historical	current
Transactions mostly	reads	updates
optimized for	reads	updates
data redundancy	high	low
database size	humongous	large

Example : Data Warehouse (Decision support)



Limits of SQL aggregation

sale	prodId	storeId	amt
	p1	c1	12
	p2	c1	11
	p1	c3	50
	p2	c2	8

	c1	c2	c3
p1	12		50
p2	11	8	

- Flat tables are great for processing, but hard for people to read and understand.
- Pivot tables and cross tabulations (spreadsheet terminology) are very useful for presenting data in ways that people can understand.
- Note that some table **values** become column or row **names**!
- Standard SQL does not handle pivot tables and cross tabulations well.

A very influential paper [G+1997]

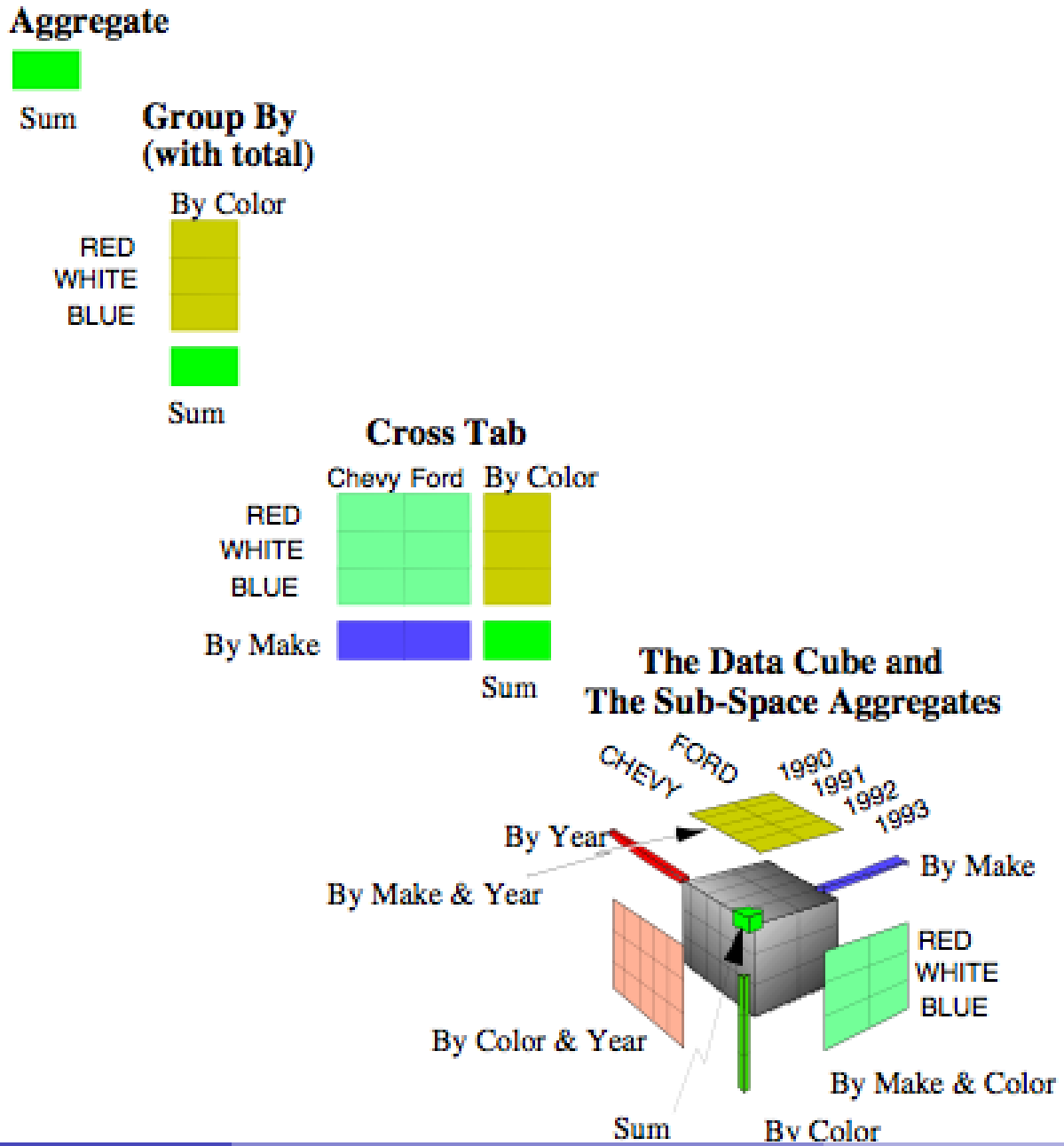
Data Mining and Knowledge Discovery 1, 29–53 (1997)

Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals*

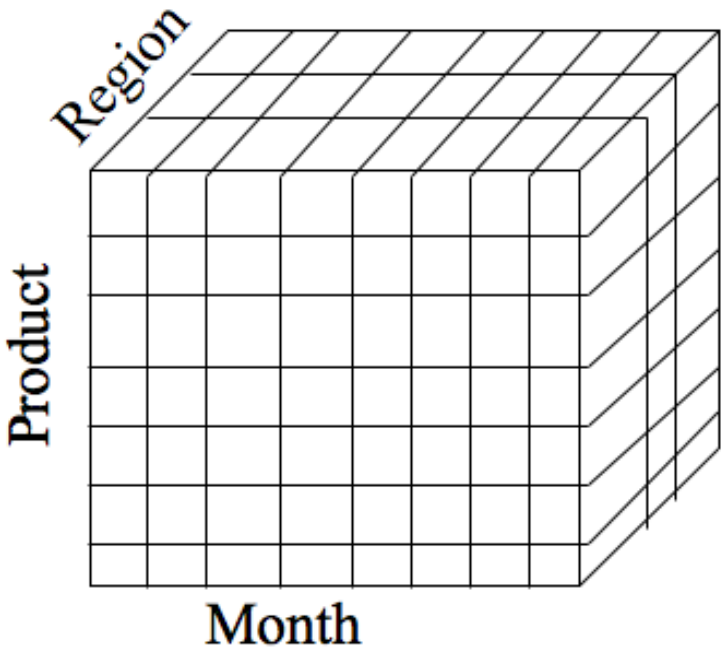
JIM GRAY Gray@Microsoft.com
SURAJIT CHAUDHURI SurajitC@Microsoft.com
ADAM BOSWORTH AdamB@Microsoft.com
ANDREW LAYMAN AndrewL@Microsoft.com
DON REICHART DonRei@Microsoft.com
MURALI VENKATRAO MuraliV@Microsoft.com
Microsoft Research, Advanced Technology Division, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052

FRANK PELLOW Pellow@vnet.IBM.com
HAMID PIRAHESH Pirahesh@Almaden.IBM.com
IBM Research, 500 Harry Road, San Jose, CA 95120

From aggregates to data cubes



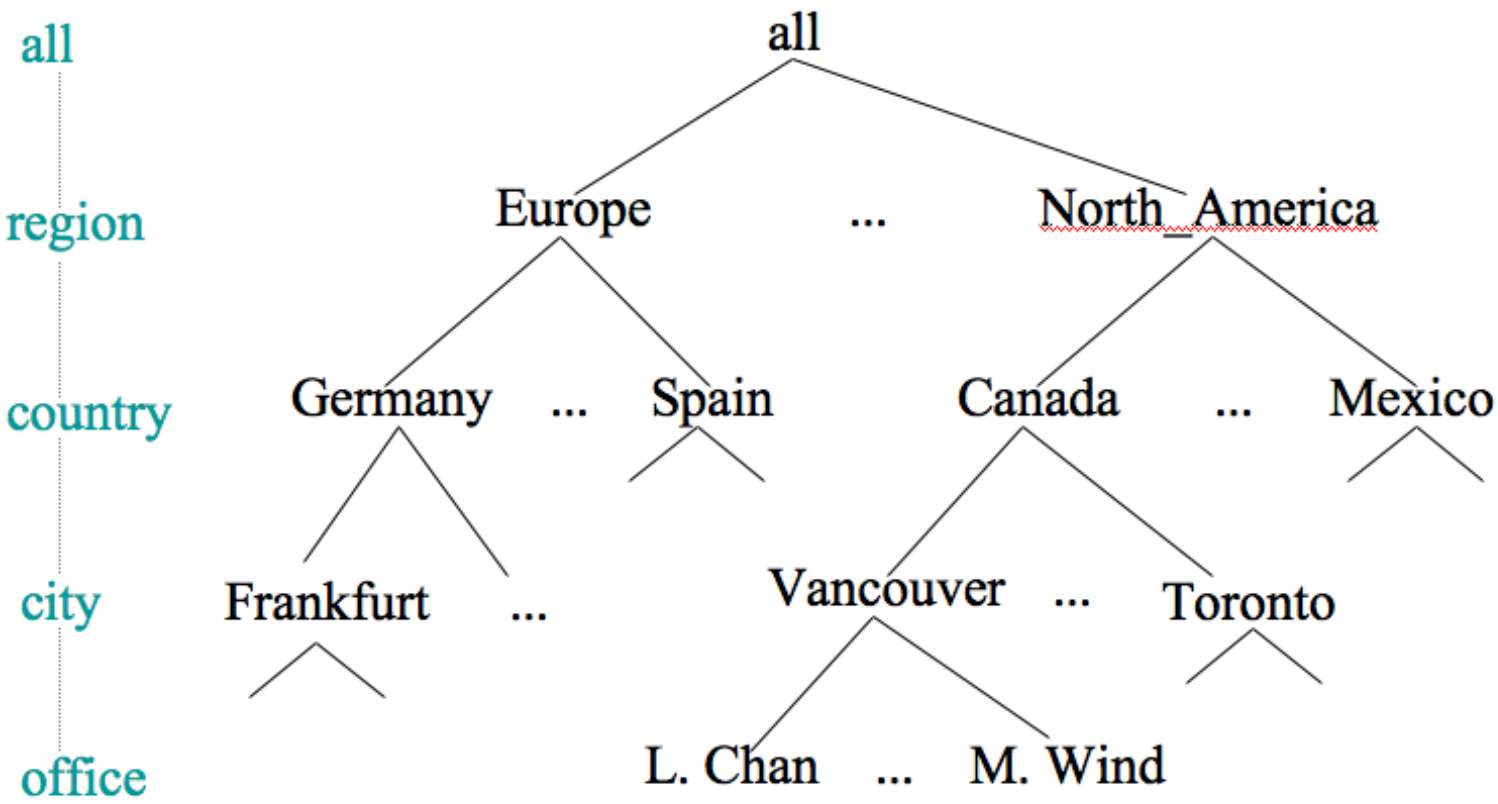
The Data Cube



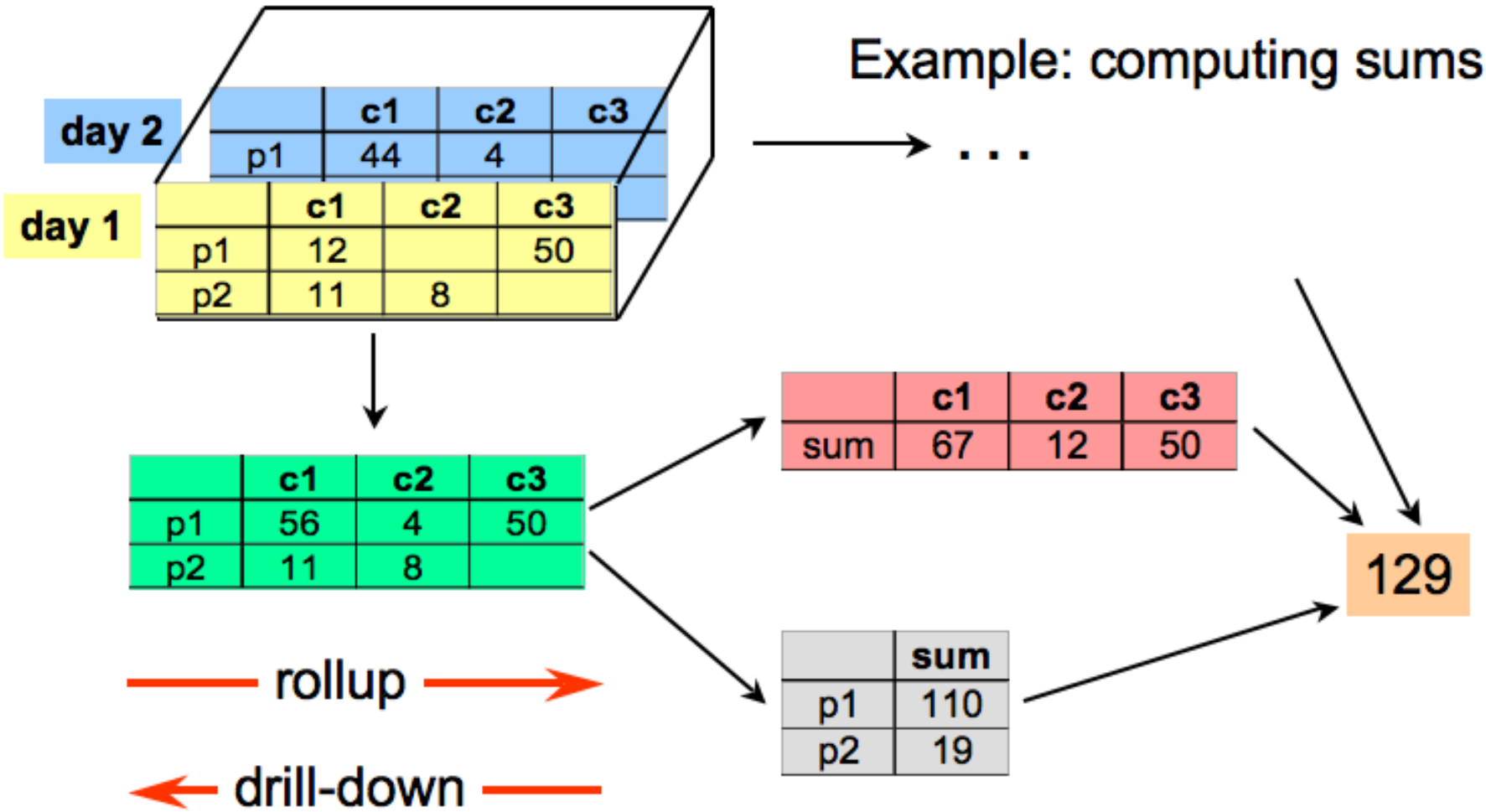
Dimensions:
Product,
Location,
Time

- Data modeled as an *n*-dimensional (hyper-) cube
- Each dimension is associated with a hierarchy
- Each “point” records facts
- Aggregation and cross-tabulation possible along all dimensions

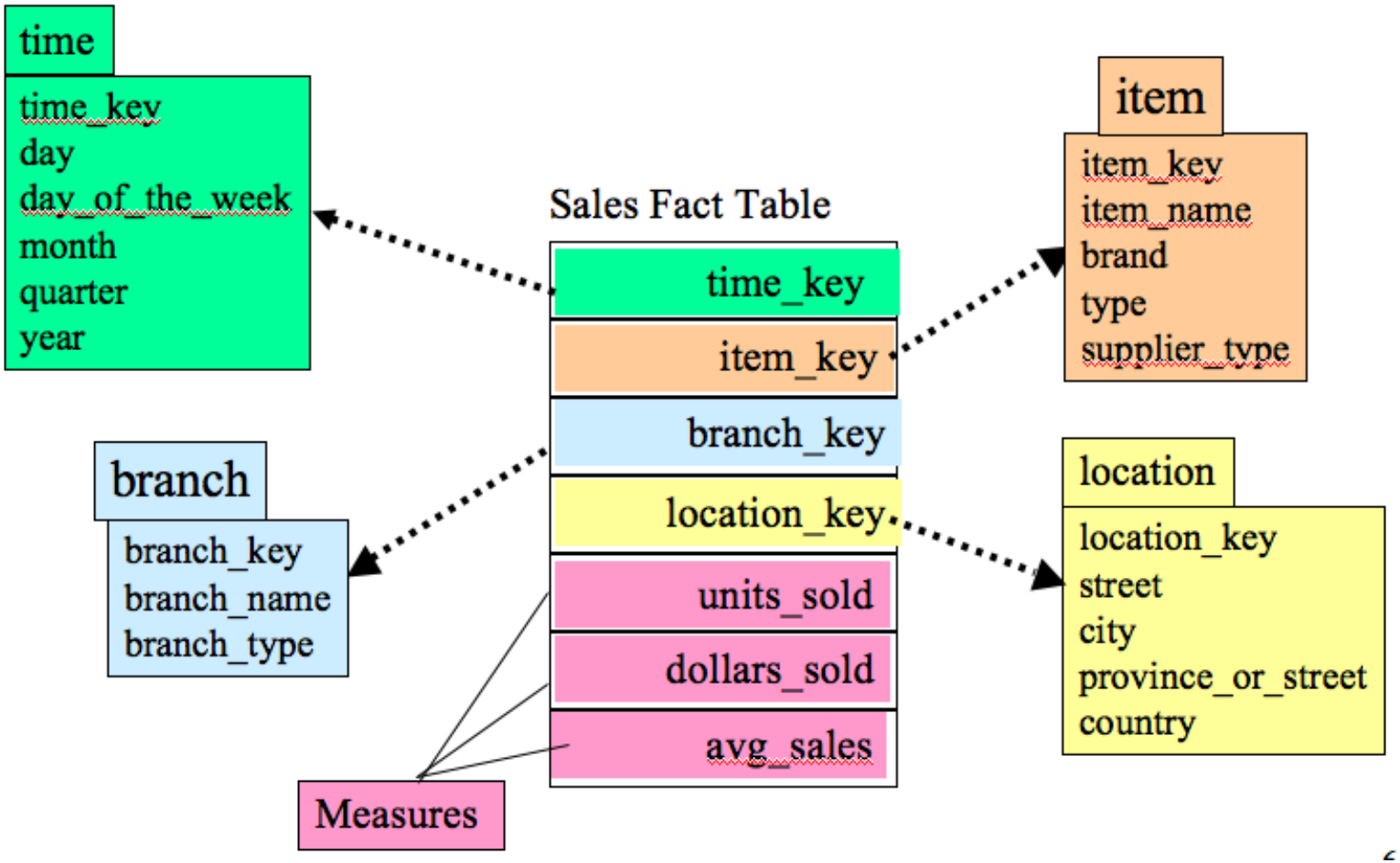
Hierarchy for **Location** Dimension



Cube Operations

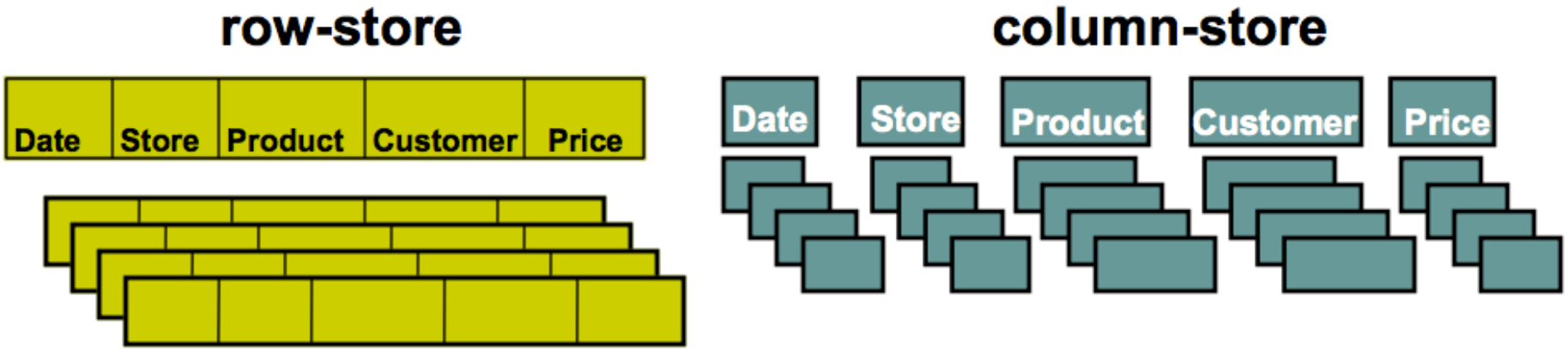


The Star Schema as a design tool



- In practice fact tables can be very large with hundreds of columns.
- Row-oriented table stores can be very inefficient since a typical query is concerned with only a few columns.

Column-oriented implementations



+ easy to add/modify a record

- might read in unnecessary data

+ only need to read in relevant data

- tuple writes require multiple accesses

=> *suitable for read-mostly, read-intensive, large data repositories*

From VLDB 2009 Tutorial: Column-Oriented Database Systems, by Stavros Harizopoulos, Daniel Abadi, Peter Boncz.

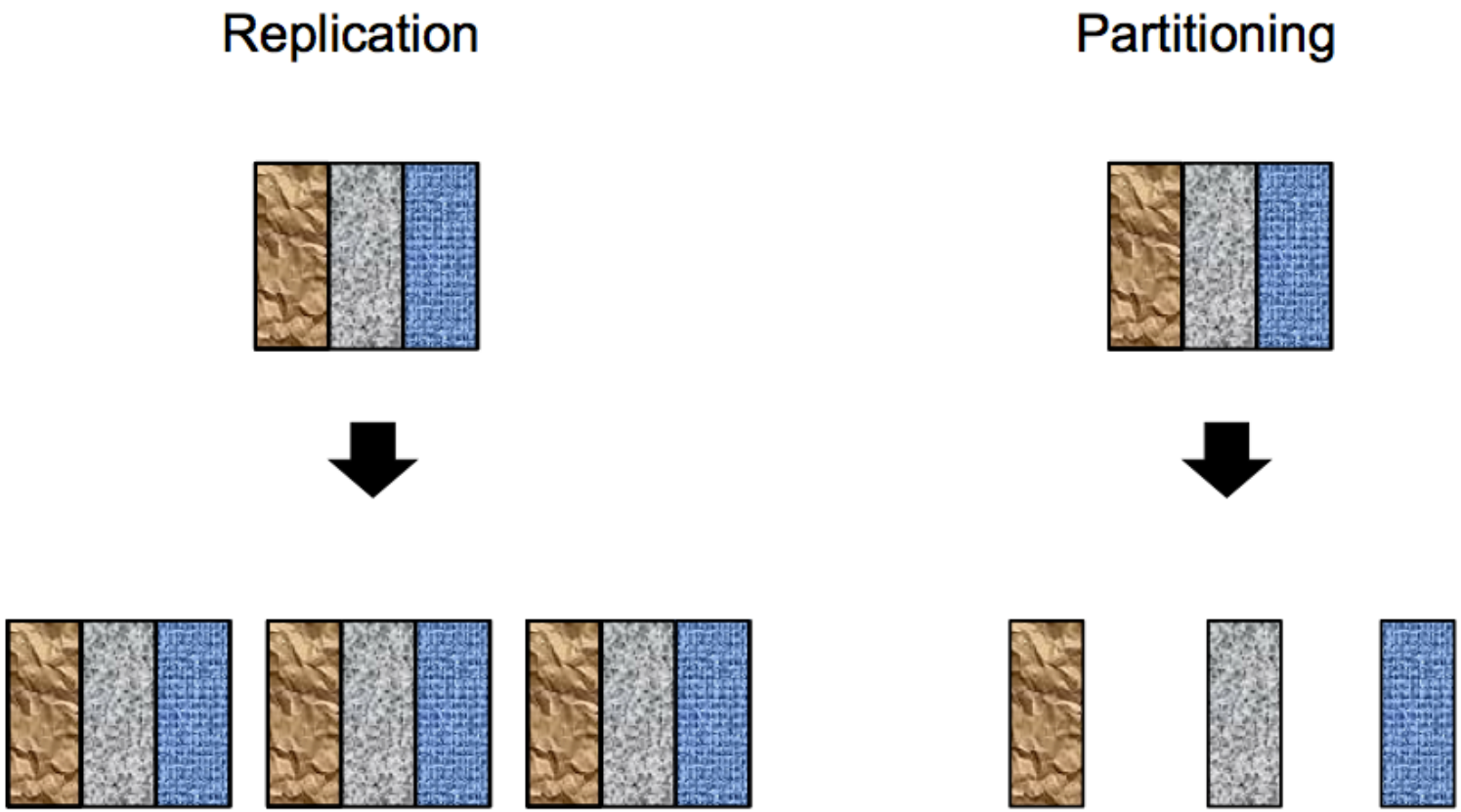
Distributed databases

Why distribute data?

- **Scalability**. The data set or the workload can be too large for a single machine.
- **Fault tolerance**. The service can survive the failure of some machines.
- **Lower Latency**. Data can be located closer to widely distributed users.

Distributed databases are an important technology supporting cloud computing.

How do we distribute the data?



Note: partitions themselves are often replicated.

Distributed databases pose difficult challenges

CAP concepts

- **Consistency**. All reads return data that is up-to-date.
- **Availability**. All clients can find some replica of the data.
- **Partition tolerance**. The system continues to operate despite arbitrary message loss or failure of part of the system.

It is very hard to achieve all three in a highly distributed database.

CAP principle

In a highly distributed system:

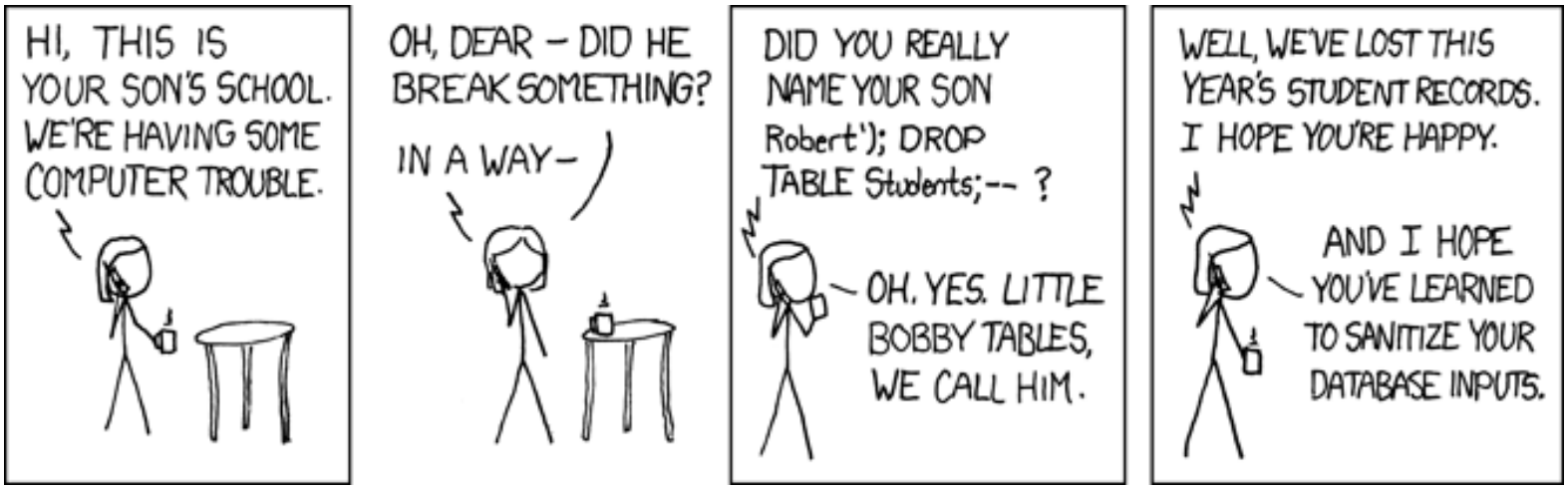
- Assume that network partitions and other connectivity problems will occur.
- Implementing transactional semantics is very difficult and slow.
- You are left engineering a **trade-off between availability and consistency**.

This gives rise to the notion of **eventual consistency**: if update activity ceases, then the system will eventually reach a consistent state.

What have we learned?

- Having a conceptual model of data is very useful, no matter which implementation technology is employed.
- There is a trade-off between fast reads and fast writes.
- There is no databases system that satisfies all possible requirements!
- It is best to understand pros and cons of each approach and develop integrated solutions where each component database is dedicated to doing what it does best.
- The future will see enormous churn and creative activity in the database field!

The End



(http://xkcd.com/327)