Data-warehousing issues by examples

- updates from legacy data sources
- qualifying the quality of a data warehouse,
- scalability, and performance.

Receiving data from legacy applications

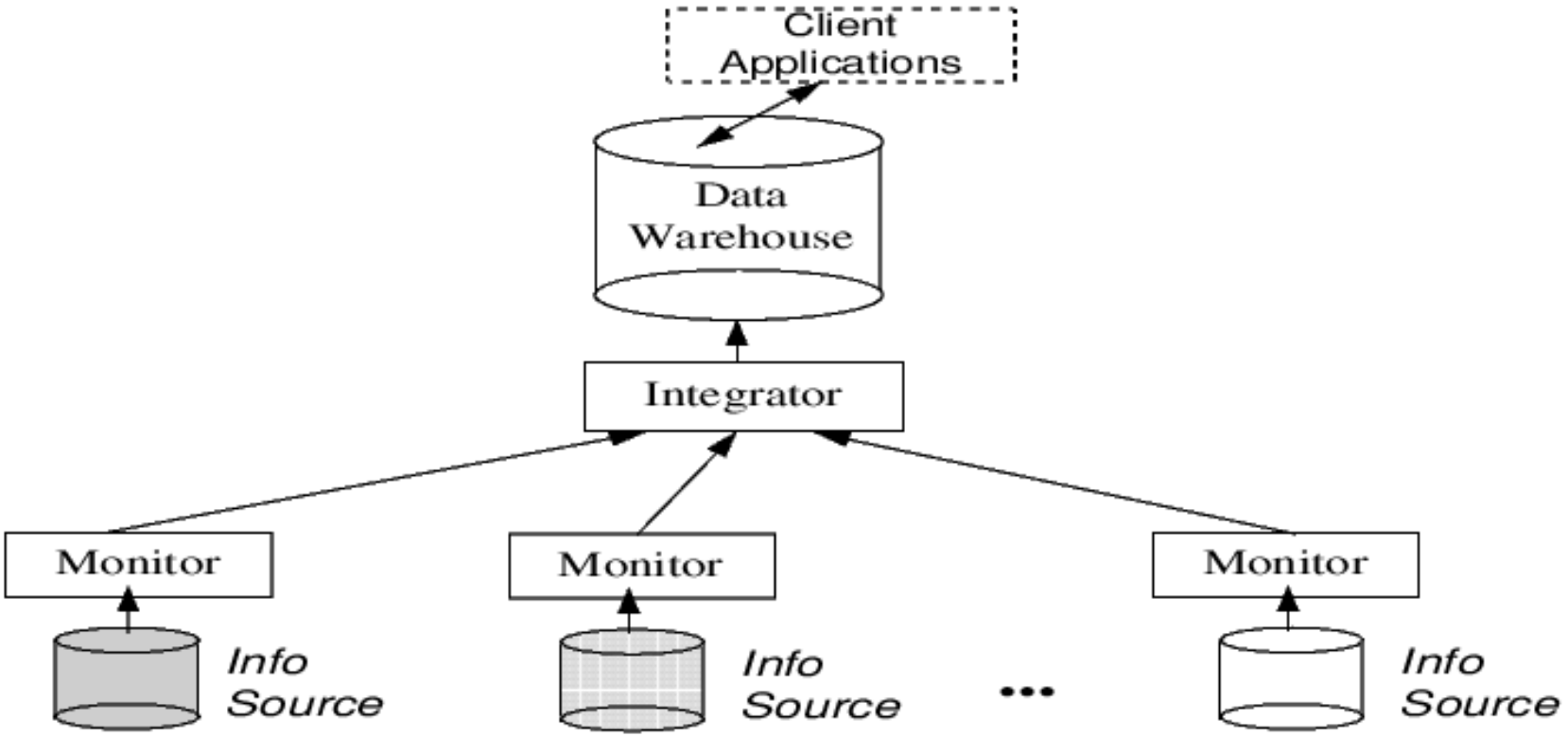
Your own app? ⇒ add a trigger to update the DW.

Unlikely to be possible for legacy applications!

What do you do if you have 16 COBOL applications running around?

Think... Think... What to do?

WHIPS – WareHouse Information Project at Stanford



Monitors

Monitors try to detect changes from (legacy) applications.

Assumption: most legacy applications allow for a data dump of their content, you can “diff” dumps!

Just fire a dump regularly (eg, daily).

3 types of changes: *insertion*, *deletion*, and *update*.

Detecting changes

If data dump can be assumed to be (key,value), problem becomes similar to symmetric outer join.

(That is you want “(key, old, new)”, “(key, old, null)”, “(key, null, new)”.)

However, a bit easier:

- don’t have to really match all keys: you can use a deletion and an insertion in place of an update (stream through the data)
- it is a routine task: you can build buffers to help you.

Streaming through the data!

Key idea: avoid having to build an index over the new data dump (which is $\Omega(n \log n)$)

Assume: you can iterate through rows in some roughly predefined order (the legacy app. dumps are somewhat consistent from day to day).

Streaming through the data! (2)

Suppose you have the old data (relational form):

```
k1  data1
k2  data2
k3  data3
```

and the new data

```
k2  data2
k3  data3
k1  data1
```

Compare rows by rows, and say “k1 is missing, delete k1”, then later ”k1 is new, insert k1, data1”.

Use auxiliary structures

Because the data warehouse updates are a routine task, you can build “permanent” auxiliary structures to help you.

For example, in the “stream through the data” strategy, you might keep a table sorted in the exact same order as what the legacy application provides. This way, you maximize your chances that the lazy. ⇒

Use auxiliary structures (2)

Hence, you might keep around this exact data dump:

```
k2  data2
k3  data3
k1  data1
```

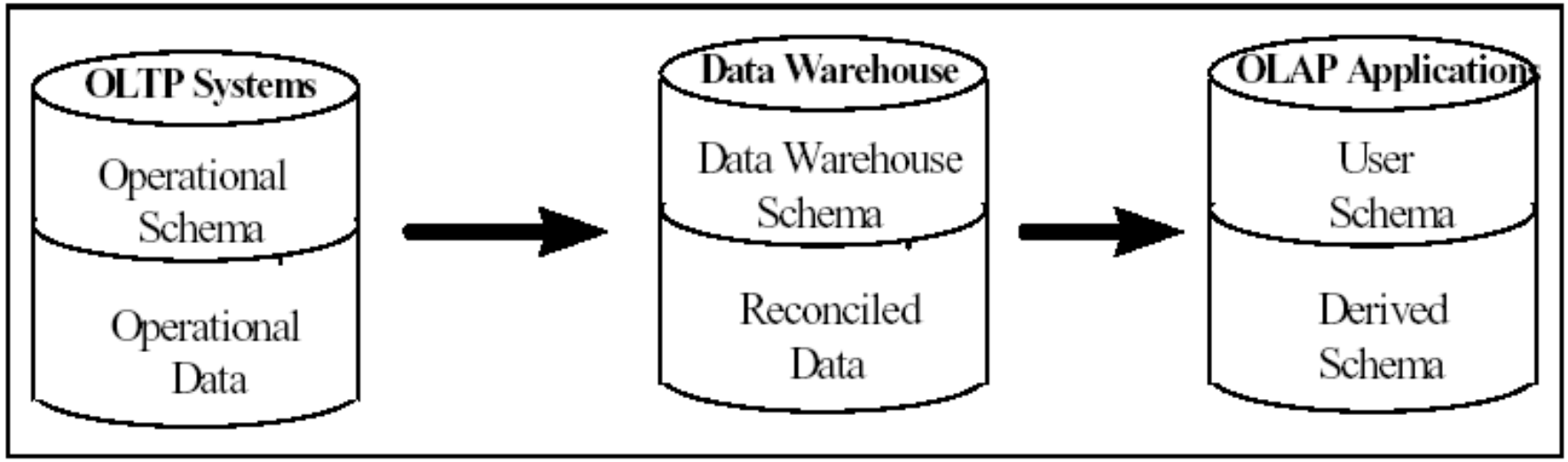
that you can diff against the new data. Presumably, your legacy app. is somewhat consistent in its dump.

Other auxiliary data structures are possible (all sorts of B-Trees and indexes).

Quality in Data Warehousing

A Data Warehouse can be seen as a buffer.

One can ask about the *quality* of the buffer. ⇒


OLTP Systems
Operational Schema
Operational Data
Data Warehouse
Data Warehouse Schema
Reconciled Data
OLAP Application
User Schema
Derived Schema

⇒

Quality Attributes

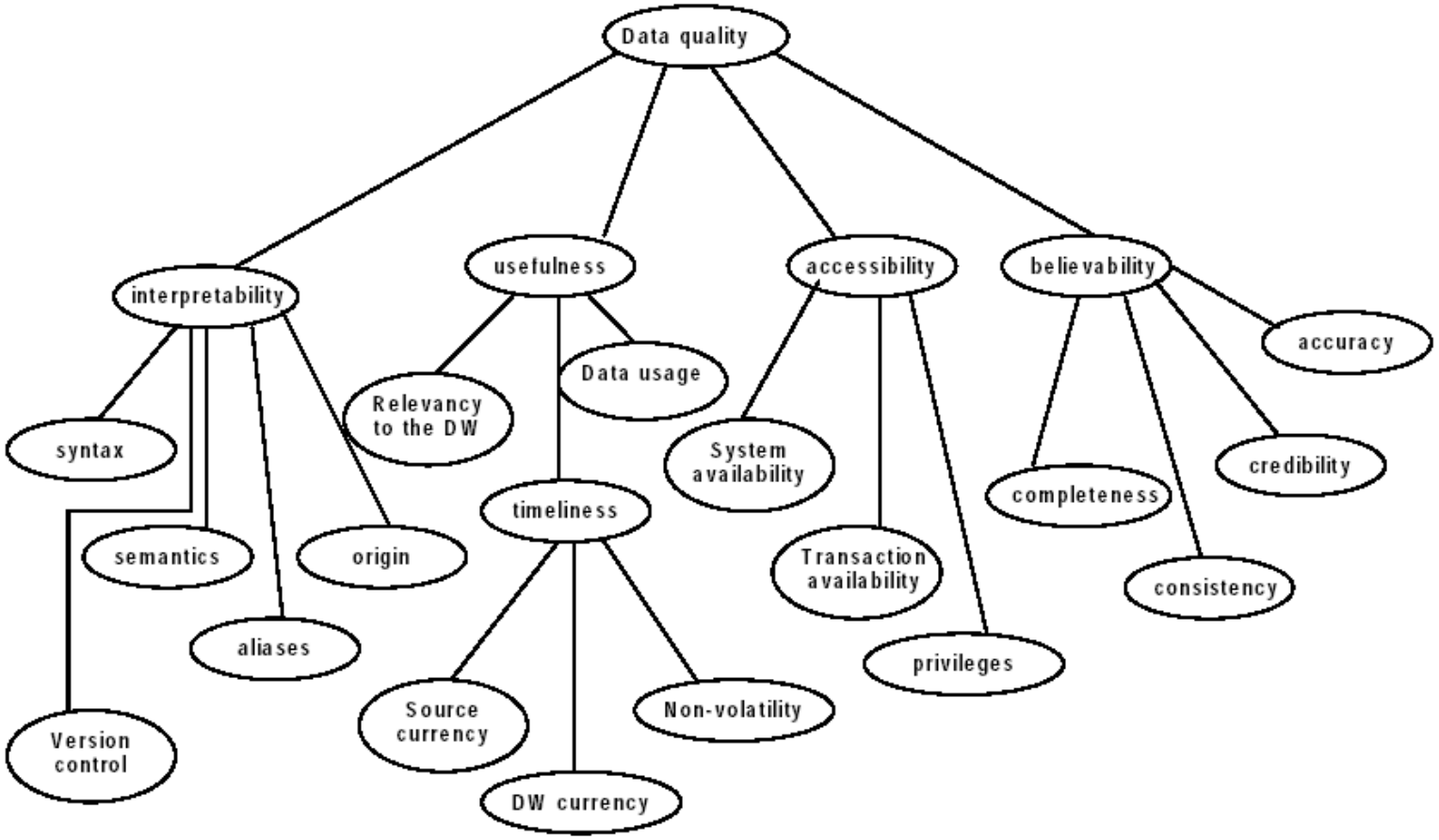
Interpretability Do I know . . . what the fields mean? . . . when the data was last updated?

Usefulness Is the data . . . relevant for my needs? . . . current?

Accessibility Do those needing the data have proper access? Is the system crashing or too slow?

Believability Am I missing too much data? Are there strong biases? Is the data quality consistent?

⇒


Data quality
interpretability
usefulness
accessibility
believability
accuracy
syntax
Relevancy to the DW
Data usage
System availability
credibility
completeness
timeliness
semantics
origin
Transaction availability
consistency
aliases
privileges
Source currency
Non-volatility
Version control
DW currency

The Data Warehouse of Newsgroups (DAWN)

Newsgroups are somewhat out-of-fashion, but the DAWN idea is widely applicable

Think of large numbers of posts (author, subject, date, body)

Must classify in many groups (Mac Programming, Wine, Cats...)

One post can be in many groups

From Traditional Newsgroups to Newsgroups as Views

Author of the post assigns it to a Newsgroup when it applies to many, author may use “cross-posting”: bad form

DAWN’s idea: a newsgroup is a view

Modern-day equivalent: Google Mail (gmail) offers a similar concept for sorting out mail, filters can be applied to “label” mail

Example: Definition of Newsgroup soc.culture.indian

- posting in current year
- body similar to 100+ articles already classified in soc.culture.indian

Example: Definition of Newsgroup att.forsale

- at most 30 days old
- poster works at AT&T
- has “for sale” in the subject

Problems

Newsgroup-selection problem which views should be eager (materialized) and which should be lazy (computed on the fly)?

newsgroup-maintenance problem must efficiently insert new article into a possibly large number of newsgroup

Some important ideas

Key idea: Have few attributes (date, length, title, author). Exploit this by indexing them.

Key idea: We know how to index text (think of suffix trees or suffix arrays or inverted indexes). Don’t be shy: index!

Newsgroup Maintenance

Do **not** check against all newsgroups (too expensive). They suggest using an “Independent Search Trees Algorithm”.

each newsgroup = rectangular region in space

article = point in space

Compute which newsgroups an article belong to by indexes

Handle “contains” constraints using string index data structure.

For Body, use word frequency and an inverted list for fast similarility searches.

Newsgroup Selection

- frequently accessed newsgroups have faster retrieval times
- rarely accessed newsgroup could tolerate slower access times.

Computing Newsgroups

Consider hierarchical sets of newsgroups as newsgroups (e.g. rec.*)

Given materialized view rec.cooking.*, you can compute rec.cooking.italian faster because fewer articles to search through $\longrightarrow$ computation by faster selection

Given materialized views canada.lang.official.fr and canada.lang.official.en, can quickly compute canada.lang.official.* $\longrightarrow$ computation by union

Newsgroup Selection As Graph

$G = (Q \cup V, E)$ where Q are the newsgroup queries (sets of newsgroups), V are the (potentially) materialized newsgroups and the hyperedges R are of the form $(q, \{v_1, \ldots, v_l\})$ where $q \in Q$ and $v_1, \ldots, v_l \in V$.

Each hyperedge is labeled with a query-cost: how expensive is it to build q from $\{v_1, \ldots, v_l\}$.

Newsgroup Selection As a Graph Problem

Find the smallest subset M of V such that all $q \in Q$ belong to at least one hyperedge with M such that the cost is no larger than a given threshold.

Newsgroup Selection is NP-Hard

(MINIMUM) SET COVER is NP-hard.

We can reduce SET COVER to a special case of the newsgroup-selection problem

$\Rightarrow$ the newsgroup-selection problem is NP-hard.

A Summary of DAWN

newsgroup-maintenance (insert of new data) is relatively easy

newsgroup-selection (materialized/non-materialized) is NP-hard.

Not all view-selection problems are NP-hard!

Data Warehouse Population Platform (DWPP)

DWPP is a recent (2003) project at Telecom Italia.

Lots of data including Call Detail Records (CDR)

- used for billing;
- used for Customer Relationship Management (CRM).

DWPP loads 20 million CDR per hour

hardware: 2TB disk, 12 CPUs and 16GB RAM.

Commercial ETL tools failed

One focus was on Extraction-Transformation-Loading (ETL) tools.

No commerical offering met their needs $\Rightarrow$ they had to roll their own

$\Rightarrow$

Review of ETL

1. extract relevant information from each source;
2. transform and integrate the data from multiple sources into a common format;
3. clean the resulting data according to business rules;
4. load and propagate the data to the data warehouse and/or data marts.

Common format

Usual idea: “the common data format (mentioned in step 2) lies inside a relational database”

Instead, DWPP converts all data into a flat-file format.

Loading and propagation of the data

They were particularly concerned with performance and maintenance (including data integrity) at the last step in their ETL tool.

Large Data Set: The main fact table has 3 billion traffic-detail records for customers. (And there are 26 million customers.)

Loading and propagation of the data: their solutions

- Partition the data
- Asynchronous Read/Transform/Write

Partition the data

All fact tables are partitioned by value and hash sub-partitioned due to:

- managing history needs - every day 1) add [and export for backup] new partition and 2) drop the oldest partition
- performance needs - Query servers work in parallel on different sub-partitions.

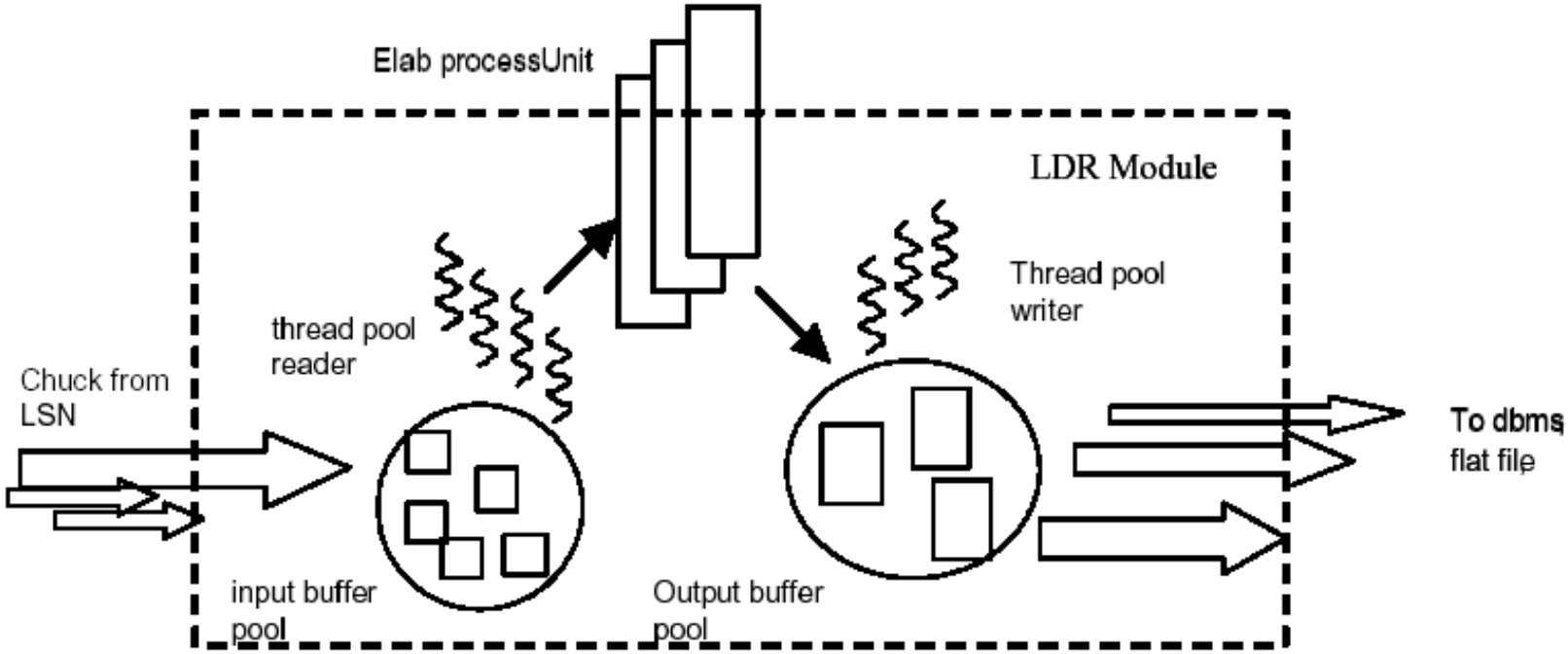
Asynchronous Read/Transform/Write

Input/Output Buffer Pool with memory segments used to read, write, and transform;

Reader Thread Pool with threads that open the input data files or chunks, once that the appropriate message has been detected from the Listener, and spread that data over the Input Buffer Pool;

Writer Thread Pool with threads that move transformed data from Output Buffer Pool to database tables (in direct or conventional path) or flat files.

Asynchronous Read/Transform/Write



Tools used in DWPP

Besides custom-built ETL, the project used some commercial tools.

Tools	Provider
ETL	-self-
OLAP	Oracle Discoverer
Data Mining	SAS Enterprise Miner

Some conclusions

- Getting data updates from legacy applications might be done using data dumps and diffs;
- You can measure the quality of a data warehouse: Interpretability, Usefulness, Accessibility, Believability;
- It might be very hard to determine automatically which views should be materialized;

Some conclusions (part 2)

- Try to stream through the legacy app. data instead of indexing it;
- Storing everything in a gigantic SQL Table is not always best (use flat files);
- You might have to build your own ETL tools;
- You should consider a partition of your data and Asynchronous Read/Transform/Write.