

Scalable heterogeneous stores for Digital City data management applications

Francesca Bugiotti[§], Damian Bursztyn[§],
Alin Deutsch[¶],
Ioana Ileana[§], Ioana Manolescu[§]

[§] OAK team, INRIA, France

[¶] DB group, UC. California San Diego



Data management case study: smart city data integration

- Datalyse French R&D project
 - **Relational** transport database
 - **RDF** Open Data produced by the city administration (cultural artefacts, events...)
 - **Graph** social network data harvested from various applications
 - May be used with or without **log** data from city Web site and various apps
- Mid-size IT companies clueless on what to use
- Easy to be wrong by orders of magnitude

The data management problem

- **Glut of varied** data management systems (DMS)
 - DM includes DBMS
- Different **data models**:
 - Relational, nested relational, tree, k-v, graphs, ...
- Different **data access capabilities** (from simple API to various query languages)
- Different **architectures**: disk- vs. memory-based, centralized vs. distributed etc.
- Different **performance**
- Different levels of **transaction support**



NoSQL DMSs



Cloud DMSs

The data management problem

- **Glut of varied** data management systems (DMS)
 - DM includes DBMS
- Different **data models**:
 - Relation
 - Different **performance**
 - API to var
 - Different **performance**
 - centralized
 - Different **performance**
 - Different levels of **transaction support**

How do we get **performance** for a variety of datasets on a variety of DMSs ?

NoSQL DMSs

graphs, ...
m simple

ory-based,

Cloud DMSs

The data management problem

- **Glut of varied** data management systems (DMS)

How do we get
performance

for a vari
on a va

Focus not on **beating the most specialized optimizations** of the most specialized engine for a **given model/application.**

- Different centrali
- Different **performance**
- Different levels of **transaction support**

NoSQL DMSs

DMSs

The data management problem

- **Glut of varied** data management systems (DMS)

How do we get
performance

for a vari
on a va

Focus not on **beating the most
specialized optimizations** of the

Focus on **robust performance for
varied data models** across a
**changing set of heterogeneous
DMSs**

NoSQL DMSs

- Different most
- Different centrali giv
- Different **perfo**
- Different levels

, ...
mple

The problem, qualified

With **correctness guarantees**

With **no hassle for the application layer**

Automatically

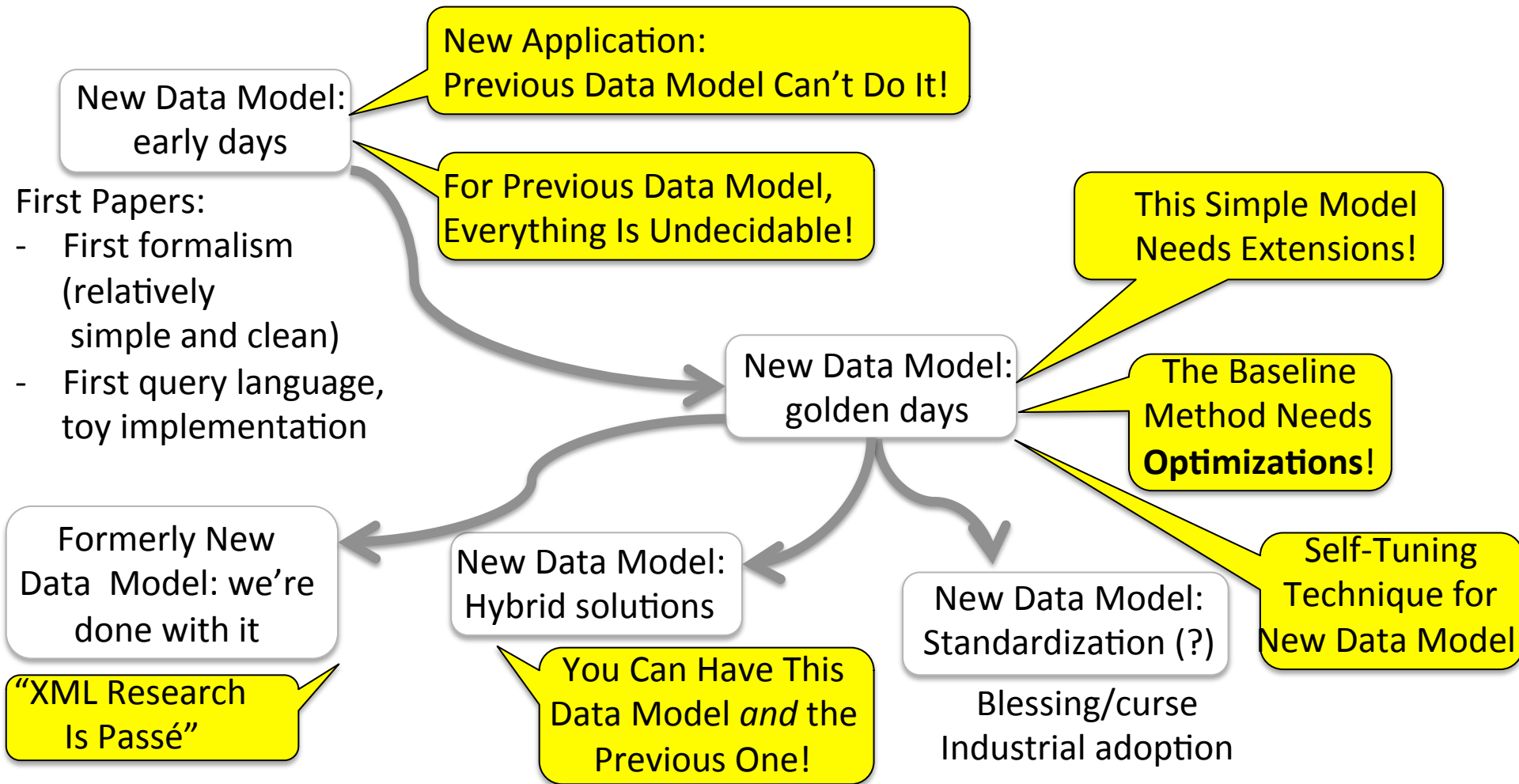
- Relation
- Different API to var

Resilient to changes

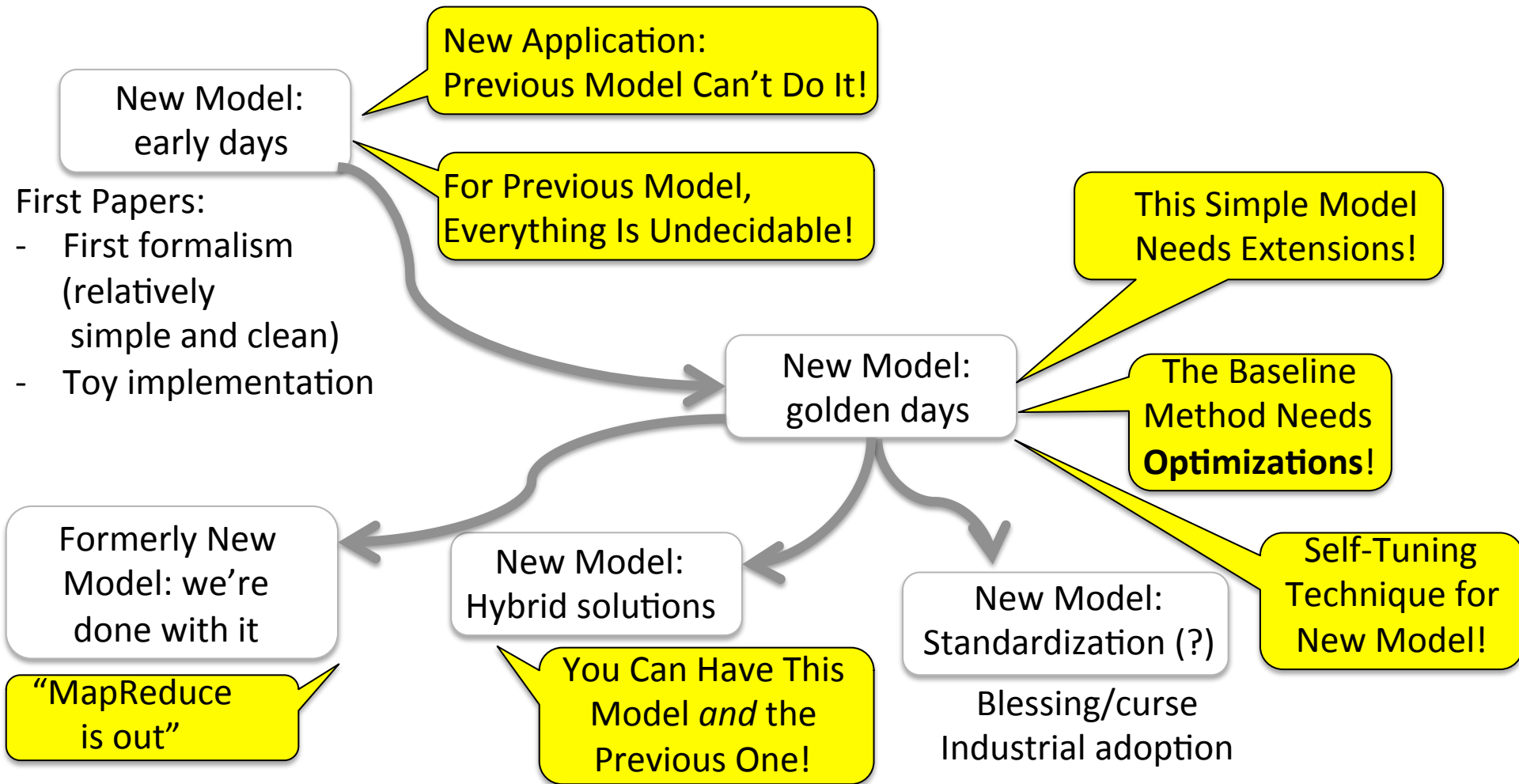
How do we get **performance** for a variety of datasets on a variety of DMSs ?

- Different **performance**
- Different levels of **transaction support**

A piece of self-derision: The Next Data Model That Will Save The World



A piece of self-derision: The Next ~~Data~~ Model That Will Save The World **Computing**



Estocada: invisible glue for heterogeneous stores

- Data models: **any/many** (side by side)
 - As the data is
- Systems: **any/many** (side by side)
 - Those available
- Store each data set as a set of **fragments**
 - Or splits / shards / partitions / indexes / materialized (potentially indexed) **views**
 - Each fragment resides in a DMS

Dataset fragmentations

R

A	B	C	D
1			
2			
3			
4			
5			
6			

=

A	B	C	D
1			
2			

U

A	B	C	D
3			
4			

=

A	B	C	D
5			
6			

U

A	B
1	
2	
3	
4	
5	
6	

U

A	C
1	
2	
3	
4	
5	
6	

U

U

A	D
1	
2	
3	
4	
5	
6	

A	B	C	D
1			
3			

U

A	B	C	D
5			
6			

Dataset fragmentations

Example: relational dataset R

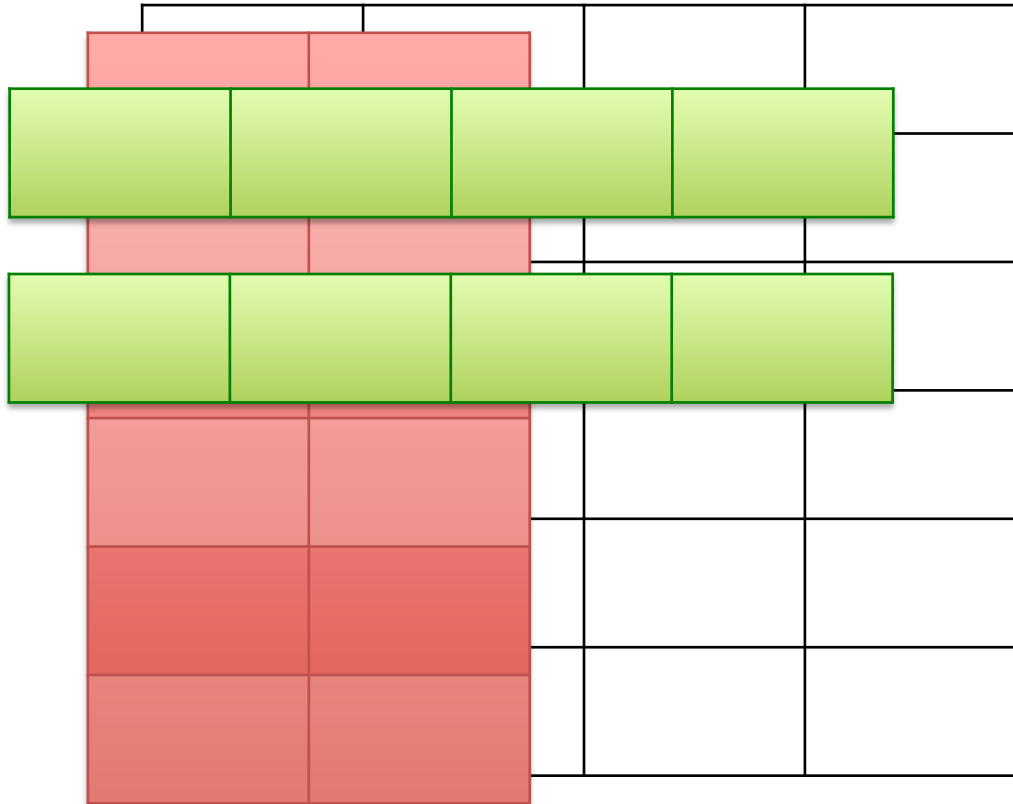
Dataset fragmentations

Example: relational dataset R

The diagram illustrates a relational dataset R with a grid of 8 rows and 4 columns. The first two columns are shaded in red, representing a fragmentation of the dataset. The remaining two columns are white. The grid is defined by black lines.

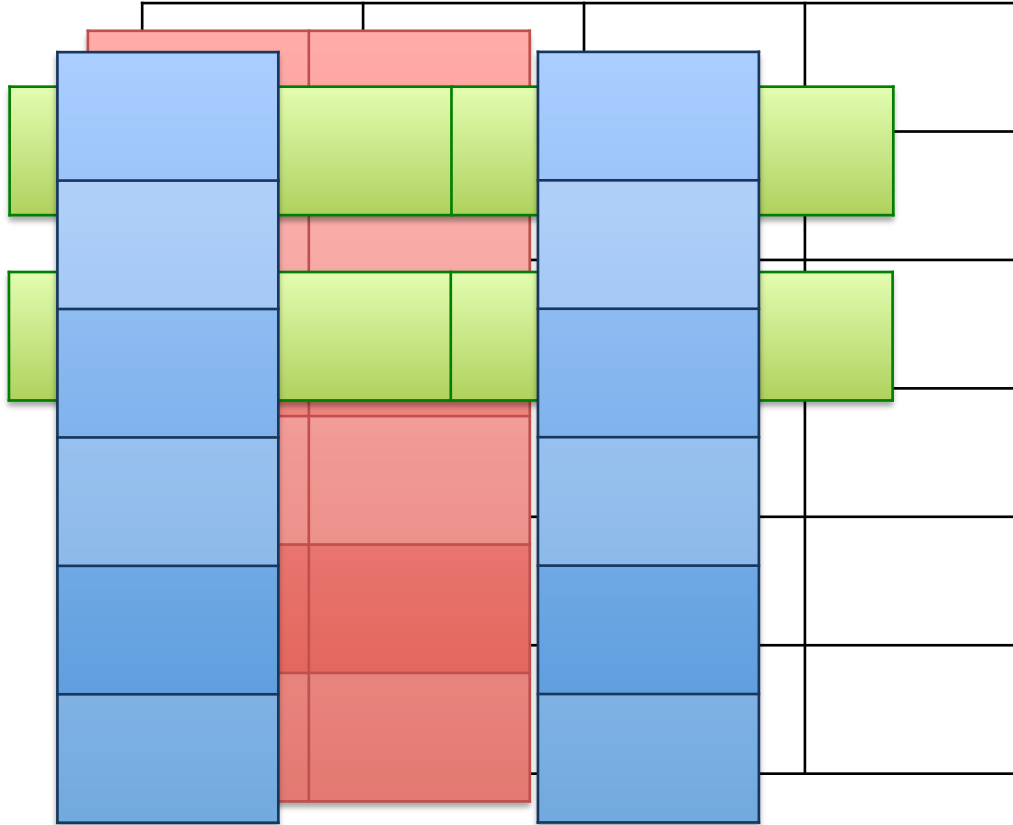
Dataset fragmentations

Example: relational dataset R



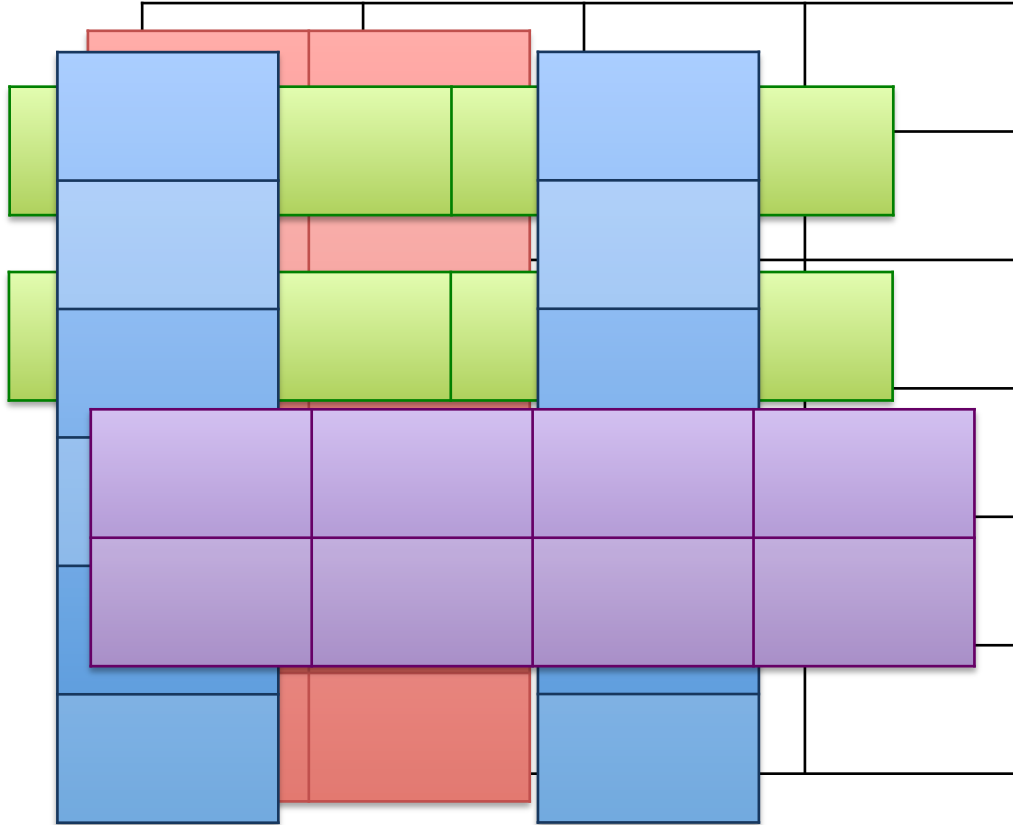
Dataset fragmentations

Example: relational dataset R



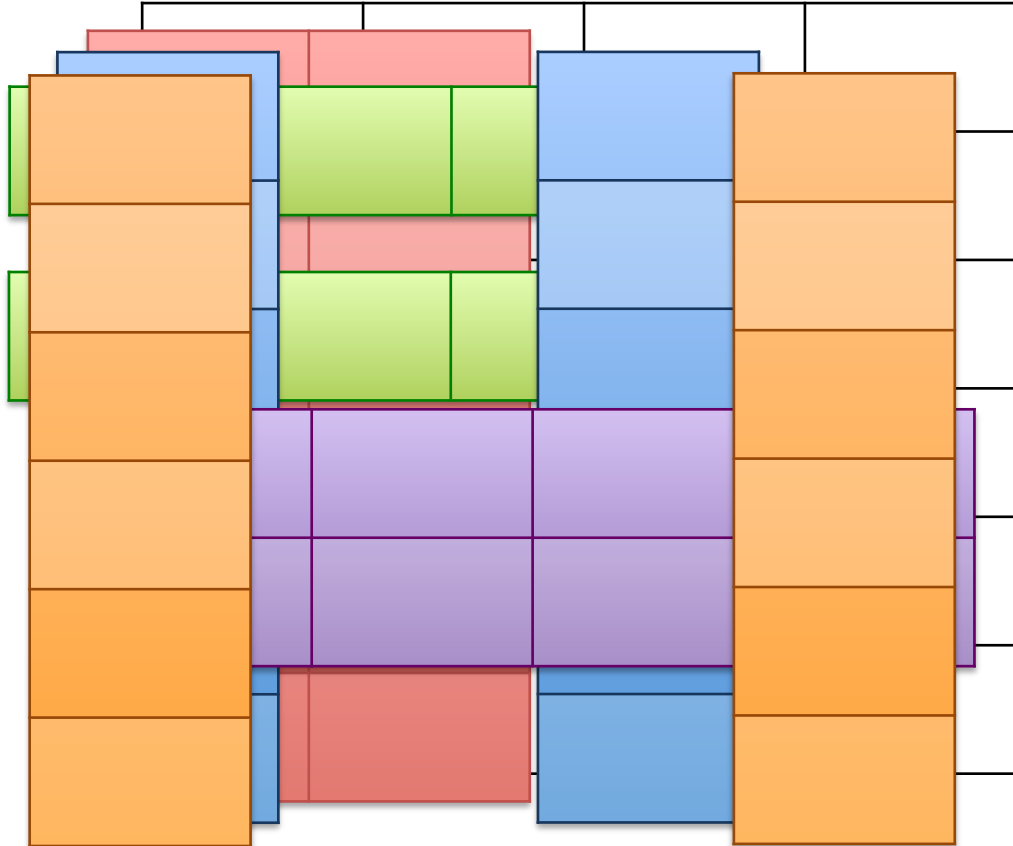
Dataset fragmentations

Example: relational dataset R



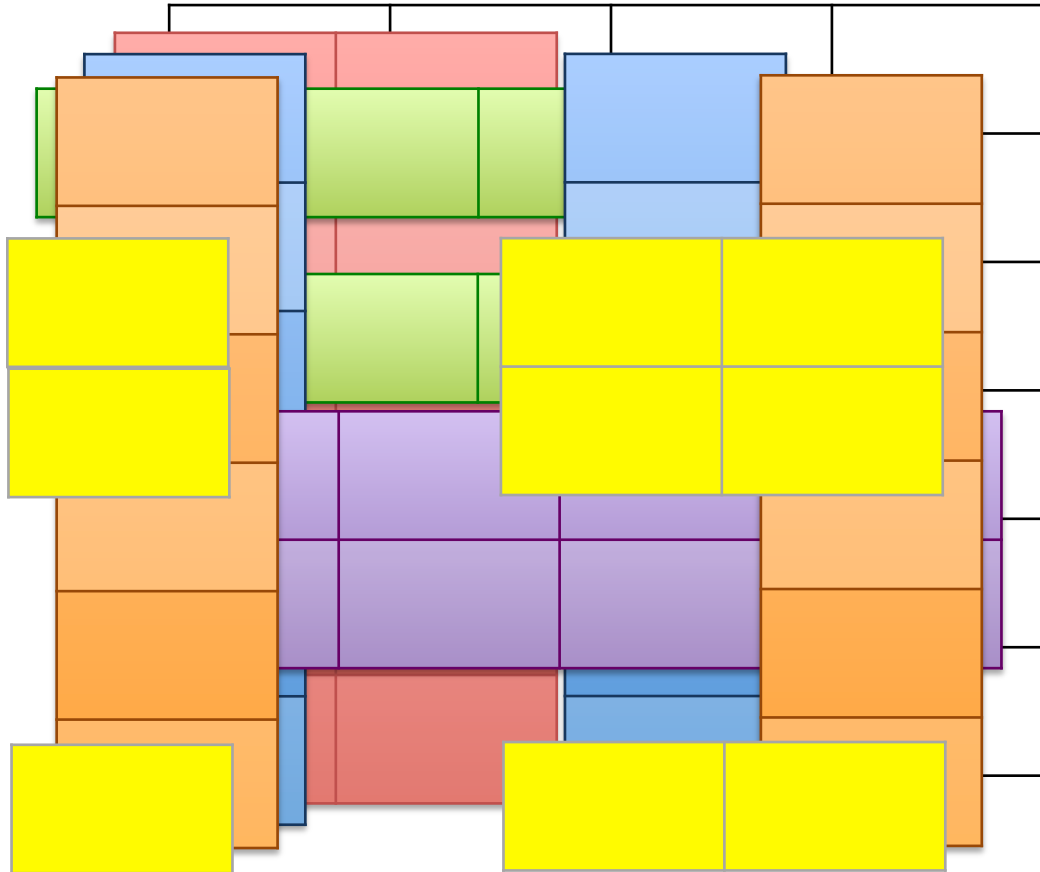
Dataset fragmentations

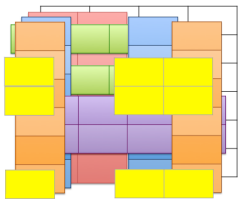
Example: relational dataset R



Dataset fragmentations

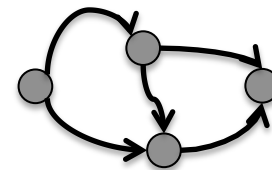
Example: relational dataset R





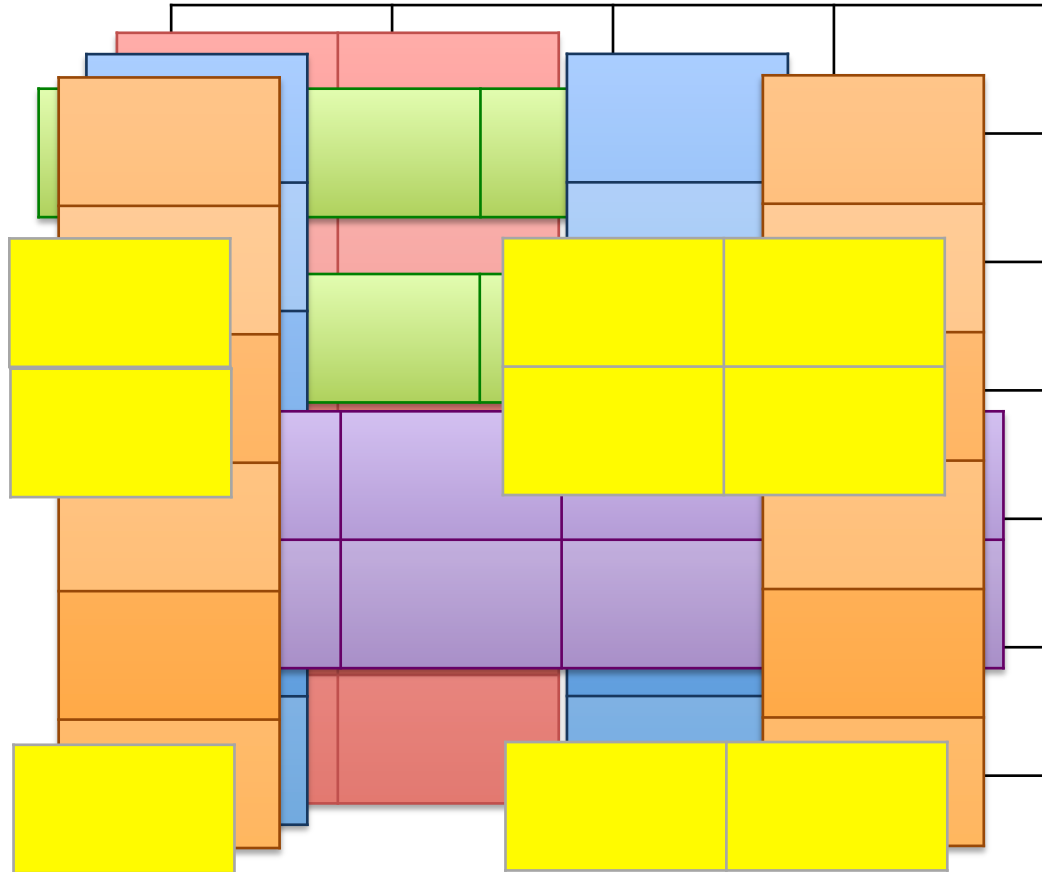
Fragmentations made of views

- The content of each fragment is described **declaratively**
- **Fragment = (materialized) view** [+ parameters]
 - « The name and addresses of all clients »
 - « The sales partitioned by zipcode »
- **Index = view with binding pattern**
 - « The name and addresses of all clients, by their age and zipcode »
 - Also: navigation in trees or graphs
key-value stores

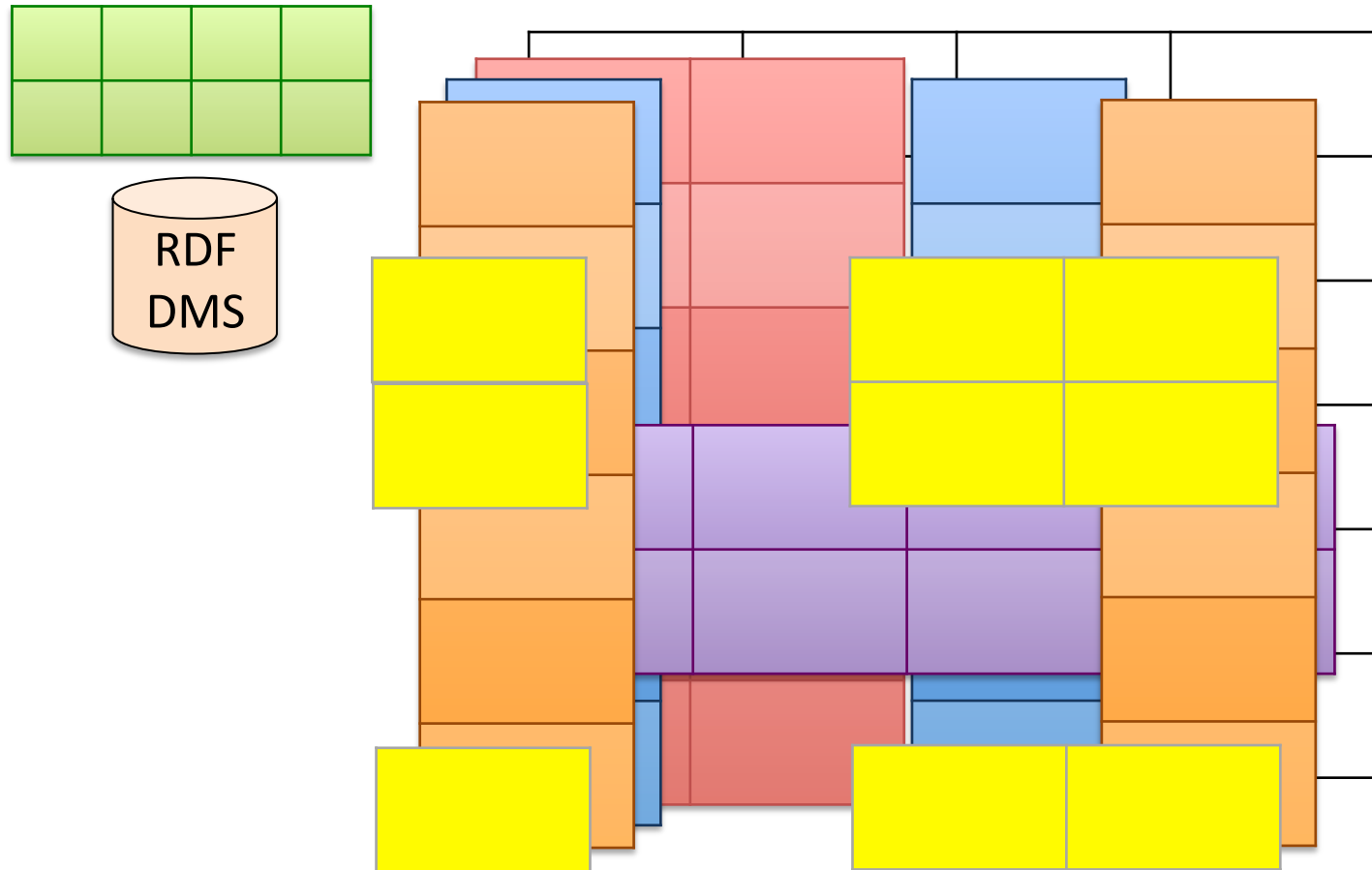


**Fragment = materialized view [+ parameters]
[+ binding pattern]**

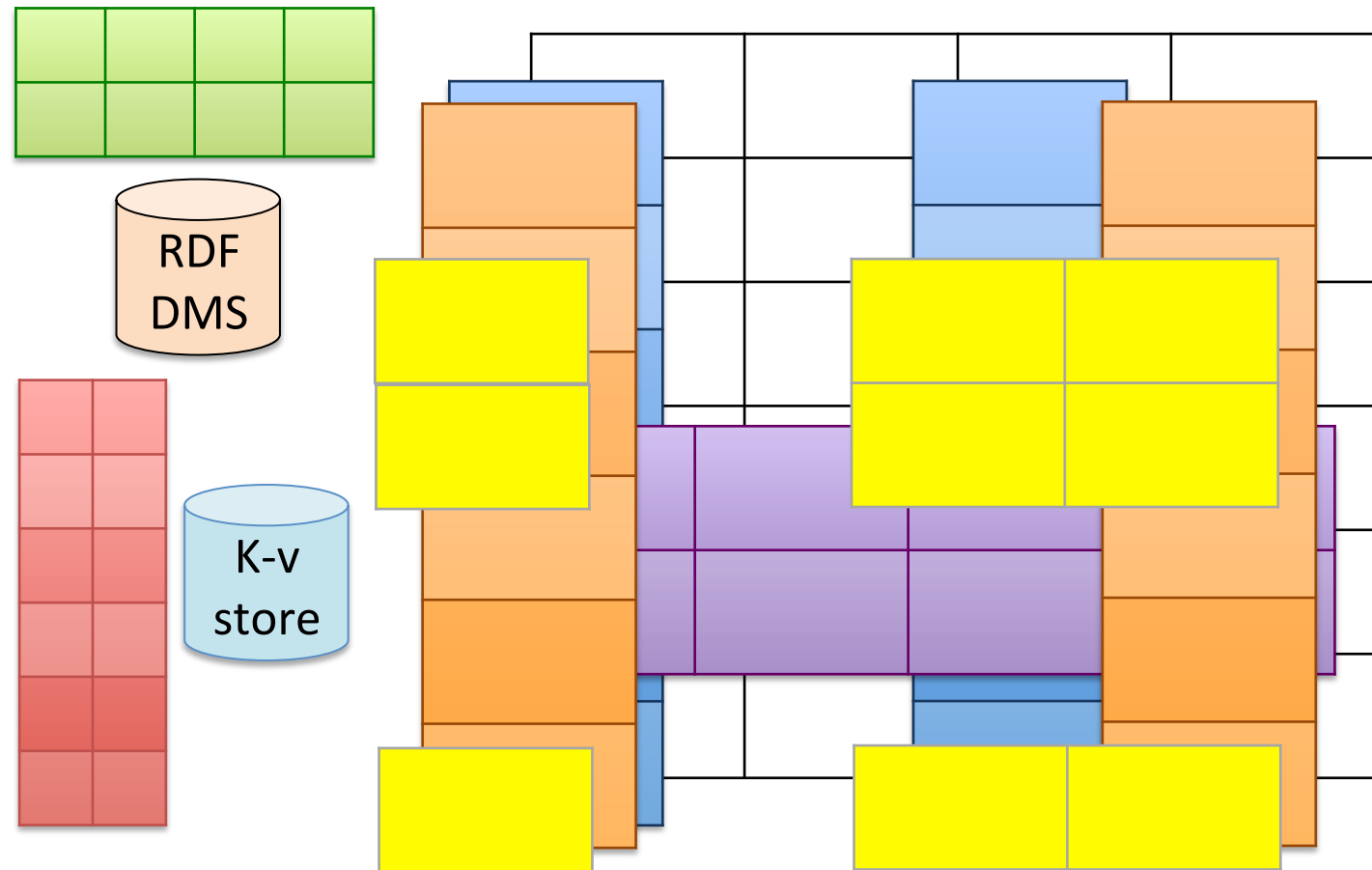
Fragments distribution across stores



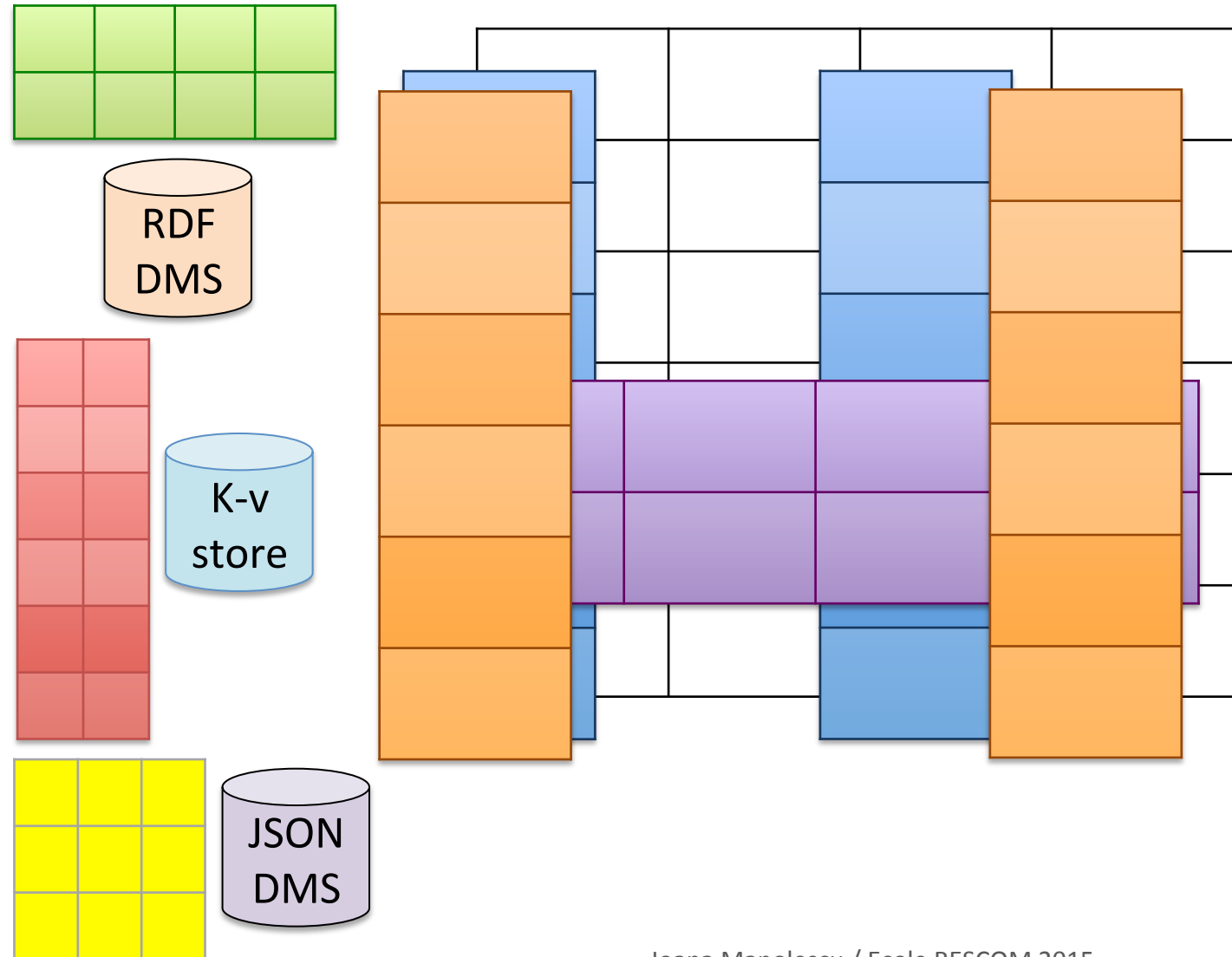
Fragments distribution across stores



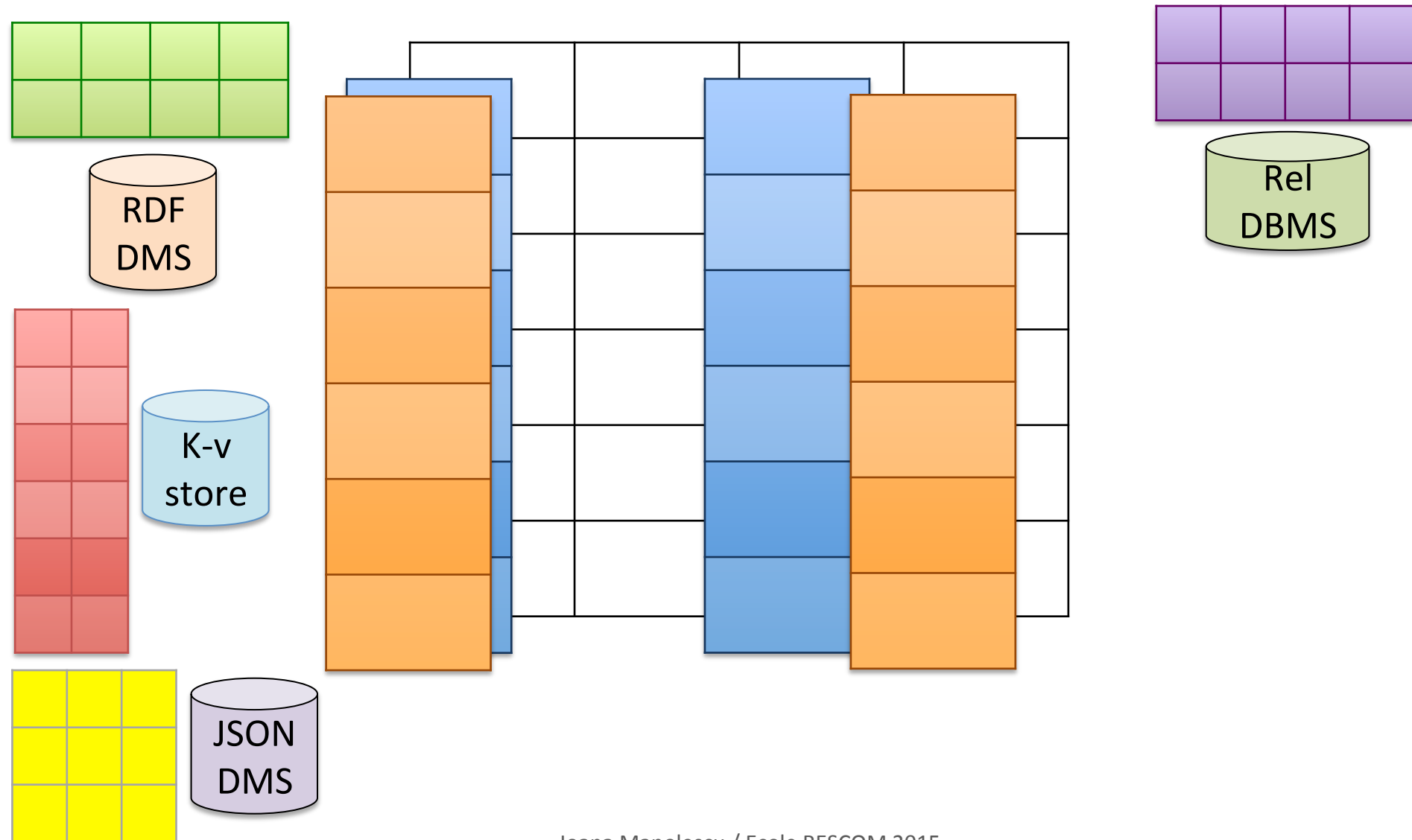
Fragments distribution across stores



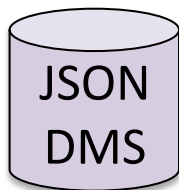
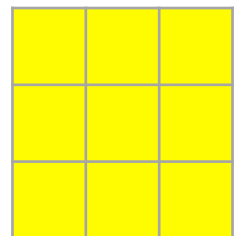
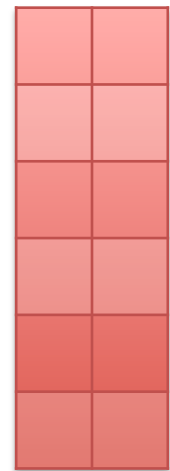
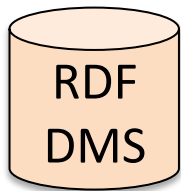
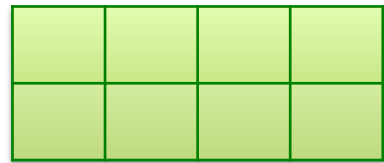
Fragments distribution across stores



Fragments distribution across stores

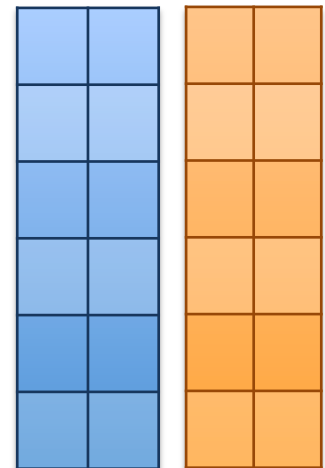
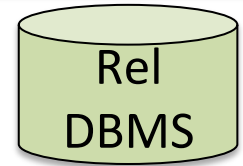


Fragments distribution across stores

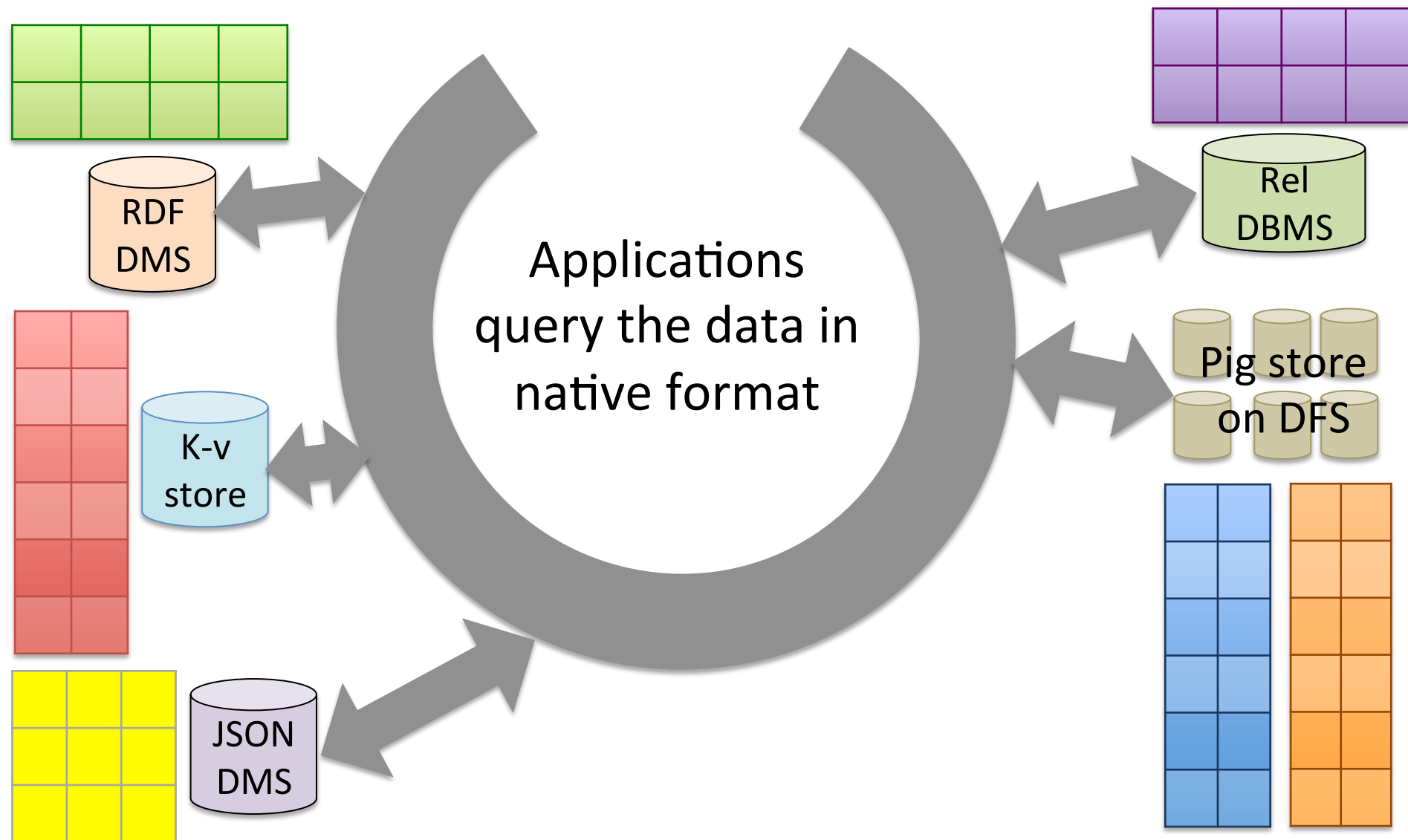


Data model translation
applied at loading

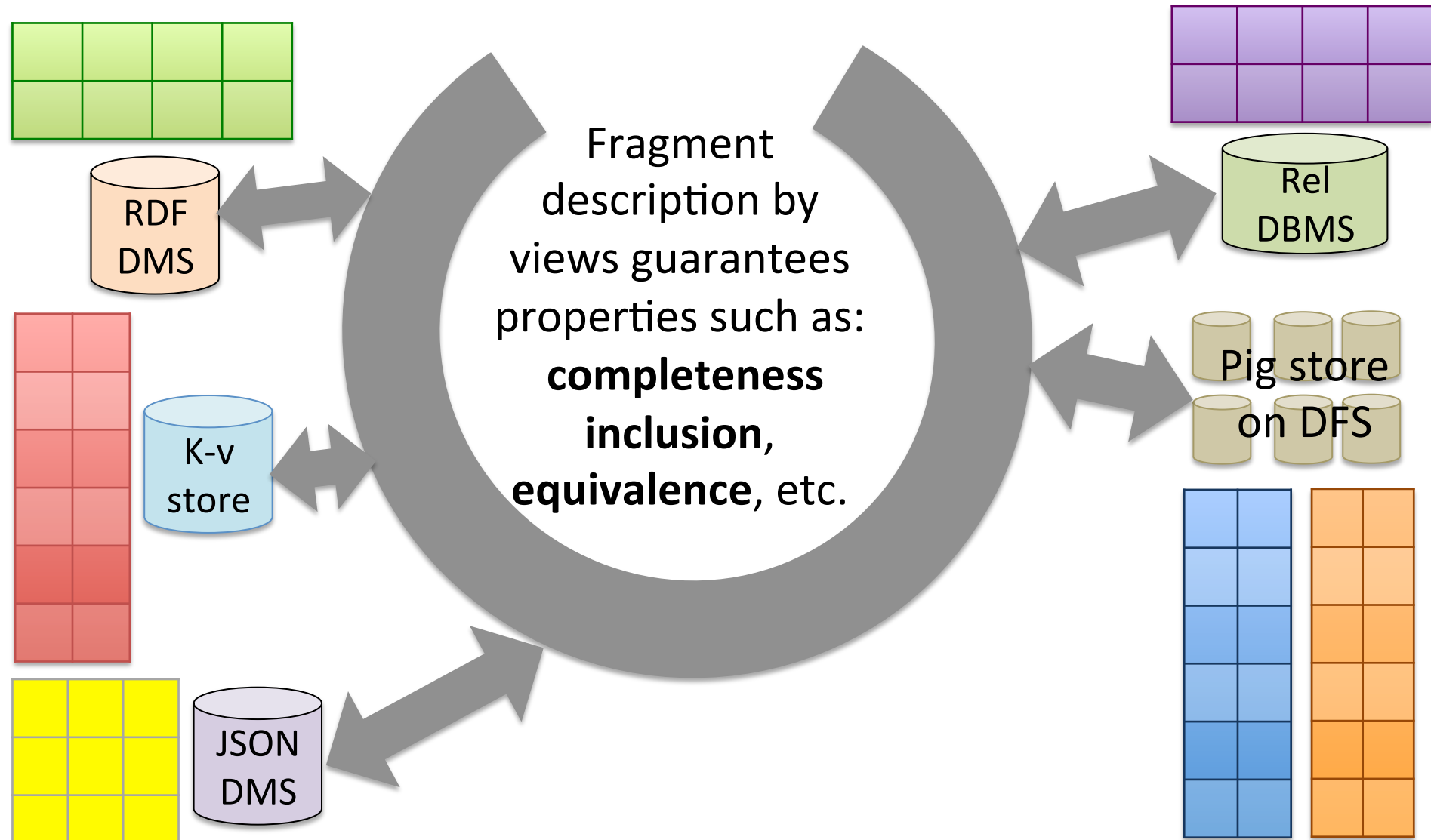
- « Dumb » translation
- The extraction logic is in the view

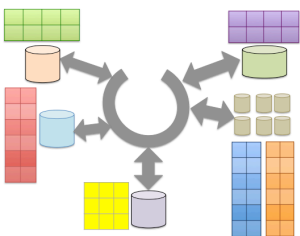


Fragments distribution across stores



Fragments distribution across stores

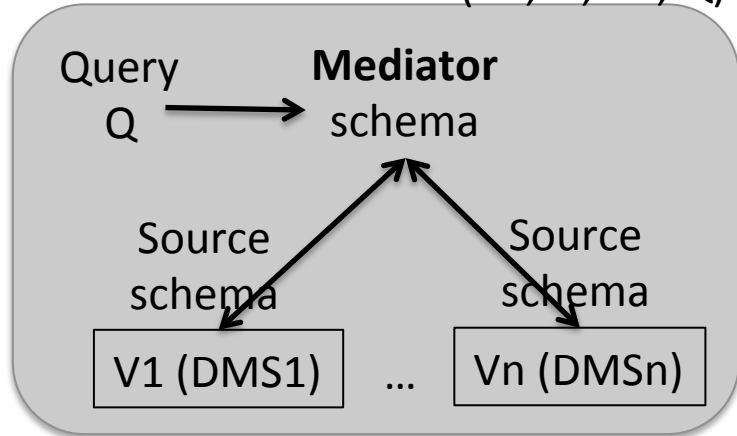




Query answering = **View-Based Rewriting**

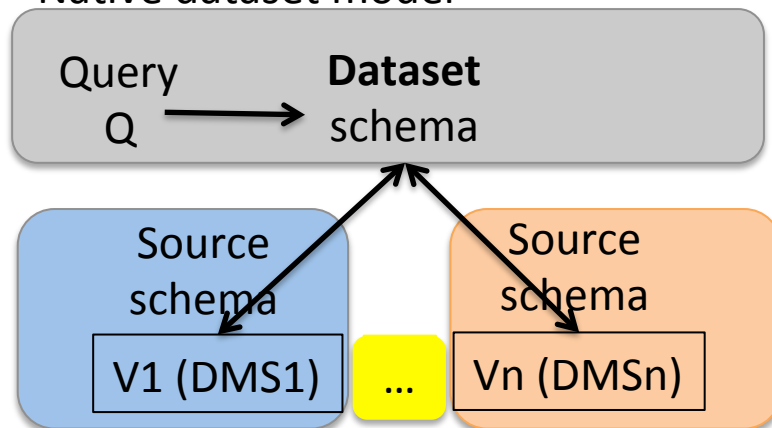
- VBR known for dramatic performance improvements
 - Basically no limit (e.g. view = query)!
 - P. Larson (2011 SIGMOD ToT): « the problem is explaining to a disappointed user that a modified view doesn't match any more! »
- Comparison with « Local As Views » mediation
 - ≠ data models

Common data model (V_1, \dots, V_n, Q)



VS.

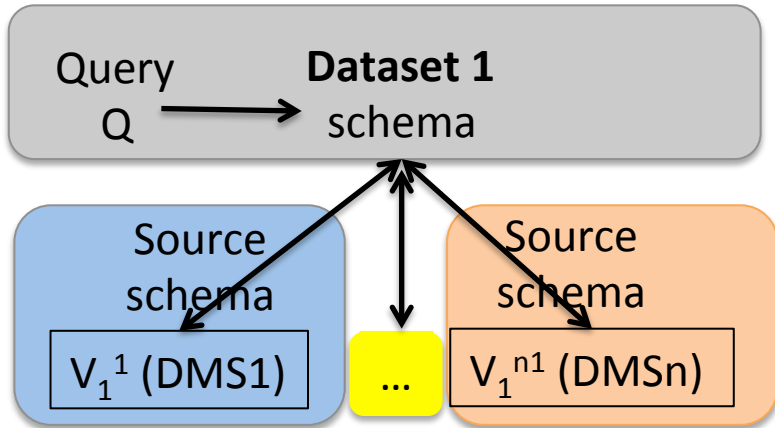
Native dataset model



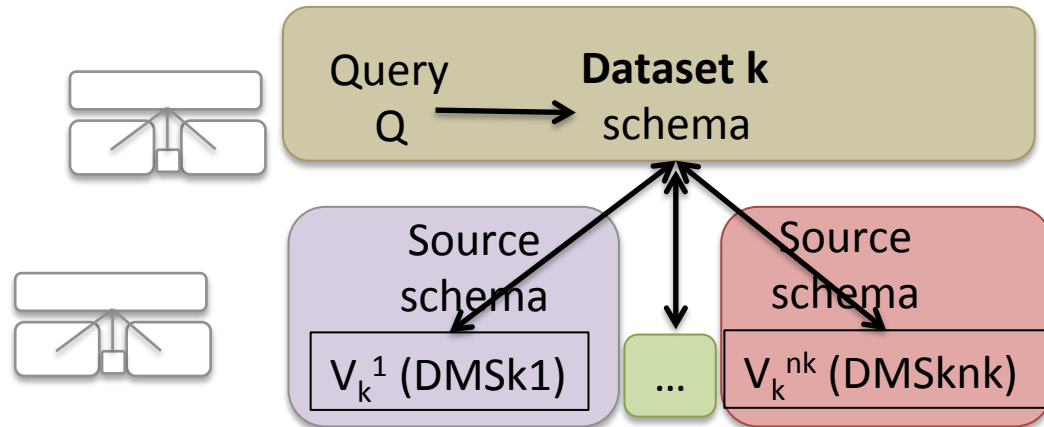
Query answering = view-based rewriting

- VBR known for dramatic performance improvements
- Comparison with « Local As Views » mediation:
 - ≠ data models
 - Side-by-side data models at the top

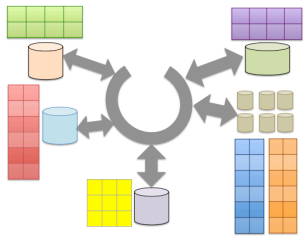
Native model of dataset 1



Native model of dataset k

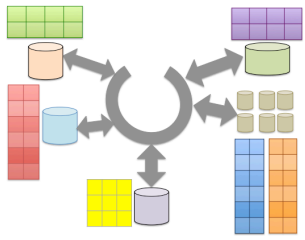


- Common benefit with LAV: Applications **unaware of the fragmentation!**
- Novel benefit: **fragments can migrate to ≠ systems and data models**



View-based rewriting with heterogeneous data models

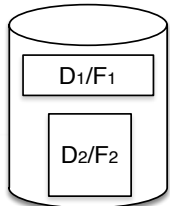
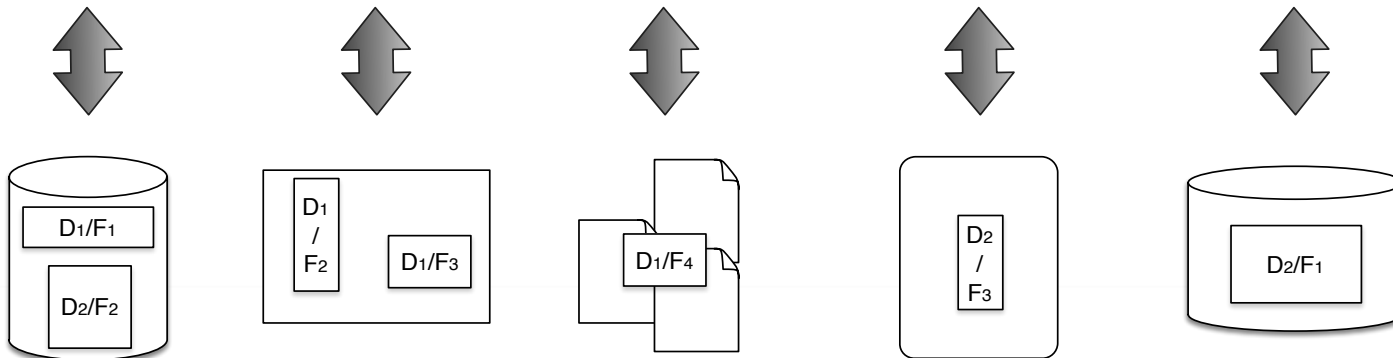
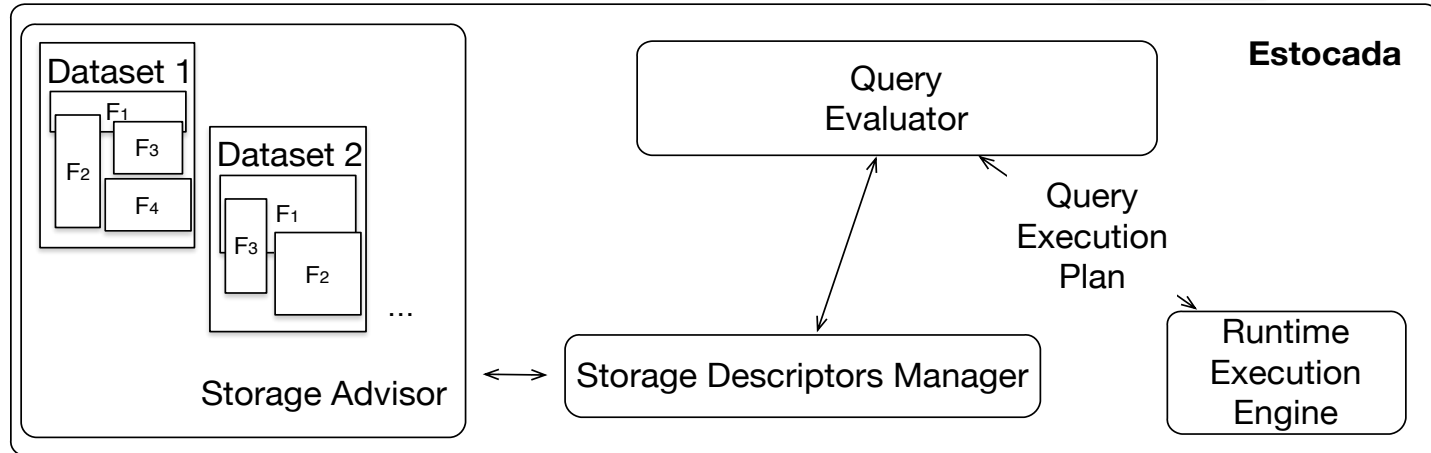
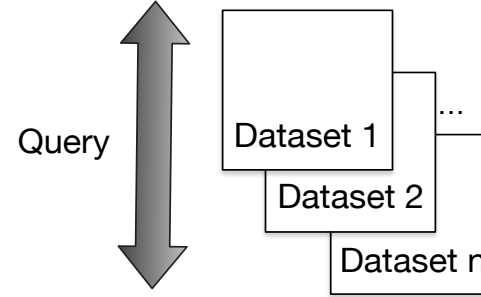
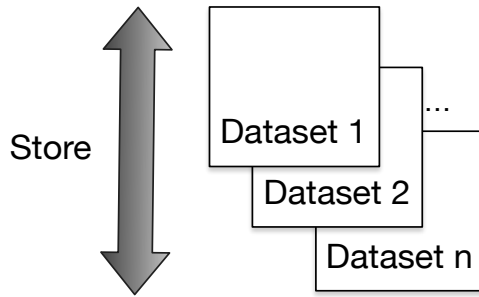
- **Model all problems as the most complex** and solve it there
 - No! (complexity; also: The Next Data Model that will Save the World... 😊)
- Have a set of **side-by-side rewriting algorithms** (dataset model M0, DMS models M1, M2, ...)
 - Impractical due to many data model combinations
- Our solution
 - Identify a **small set of « core » data models** which include the others
 - Have **side-by-side rewriting algorithms for this small core** (hint: many reuse opportunities)



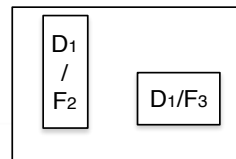
VBR with heterogeneous data models

- At the core:
 - The capacity to **describe dataset properties**
 - **Structure:** (nested) tuples, trees, graphs, bags
 - **Constraints:** keys, foreign keys, inclusion dependencies (e.g. also RDF semantics)
 - A **rewriting algorithm capable of leveraging all the information** to find all equivalent query rewritings
 - Constraints enable rewritings in cases when there would be none without them!
 - Starting point: efficient Chase-and-Backchase algorithm [Ileana, Cautis, Deutsch, Katsis, SIGMOD 2014]

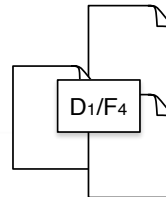
Estocada architecture



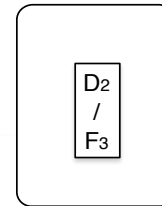
NoSQL System



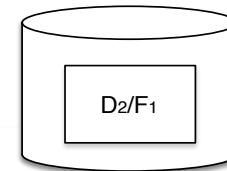
Key-value store



Document store



Nested relations store



Relational store

Estocada core modules

- VBR
 - Outputs: queries to DMSs (in their native language) + remaining integration operations
 - DMS capability descriptions exploited here.
- Runtime
 - To perform integration operations
 - For this, a single runtime (for the most expressive model, e.g. nested relations), should do
 - We may borrow one of the DMSs's runtime

What about performance?

- Select the rewriting likely to lead to the best query evaluation performance
 - **Cross-system cost model**
 - Not as crazy as it looks (cost model calibration '86; it works!)
 - Modest extension for binding patterns
- View recommendation
 - « Cross-model, cross-system data storage advisor »
 - Great progress in recent years on single-model storage (view, index etc.) recommendation
 - **Combinatorial problem** (select a subset of the possible views minimizing cost estimation)

Closest related work

- **Mixed-model VBR**
 - Agora [Manolescu, Florescu, Kossmann, VLDB 2001], Mars [Deutsch & Tannen, VLDB 2003]: XML
- Running a DMS on top of heterogeneous stores
 - **Federated databases** (Tomasac, Valduriez et al., 1996)
 - **Data integration** (wrappers / mediators) up to VIDA [Karpathiotakis, Alagiannis, Heinis, Branco, Ailamaki, CIDR 2015]
 - **Recent hybrid systems** [LeFevre, Sankaranarayanan, Hacigumus, Tatemura, Polyzotis, Carey, SIGMOD 2014]; [Jindal, Quiané-Ruiz, Dittrich, CIDR 2013]

Related issues

- Orthogonal concerns
 - Offering users a single integrated view
 - A la « global-as-views » data integration
 - Cleaning and extracting the data
- Not so orthogonal concerns
 - Providing common transactional properties across different DMS
 - « Failsafe mode »: use **Estocada** only for performance on top of a « vanilla » store
 - « Gradual »: failsafe → monitor application → recommend fragment deployment)

Wrap-up

- **Goal:**
 - Handle data efficiently on top of a given set of systems, even if it changes
 - Make the most out of each system
- **Method:**
 - View-based rewriting for multiple data models, under constraints
- **Status:** VBR ongoing, recommendation next

Thank you / Questions?

Fragments distribution across stores

