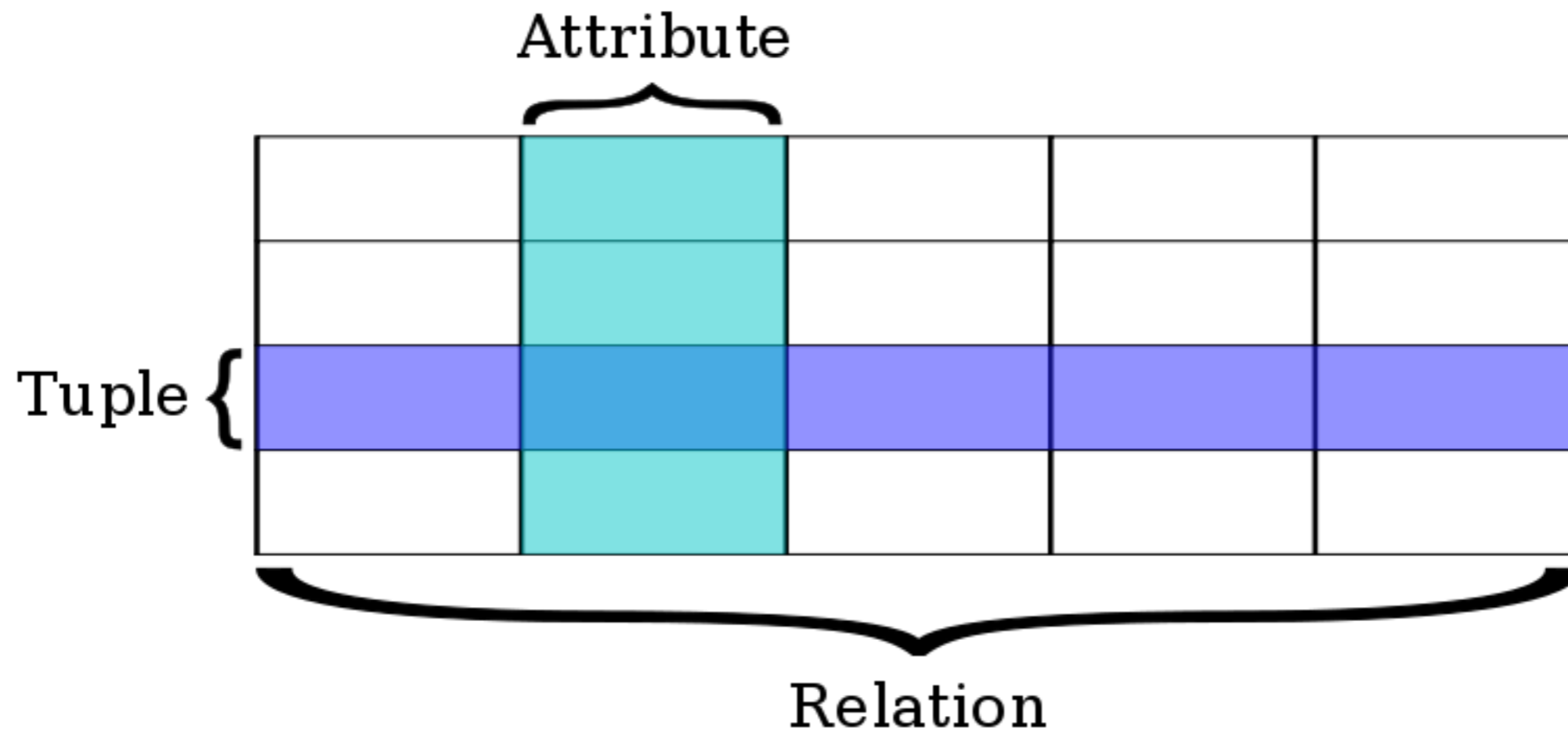


NoSQL -- What's that?

Software Languages Team
University of Koblenz-Landau
Ralf Lämmel and Andrei Varanovich

Types of databases

Relational Databases



[http://en.wikipedia.org/wiki/Relational_database] 13 Sep 2012

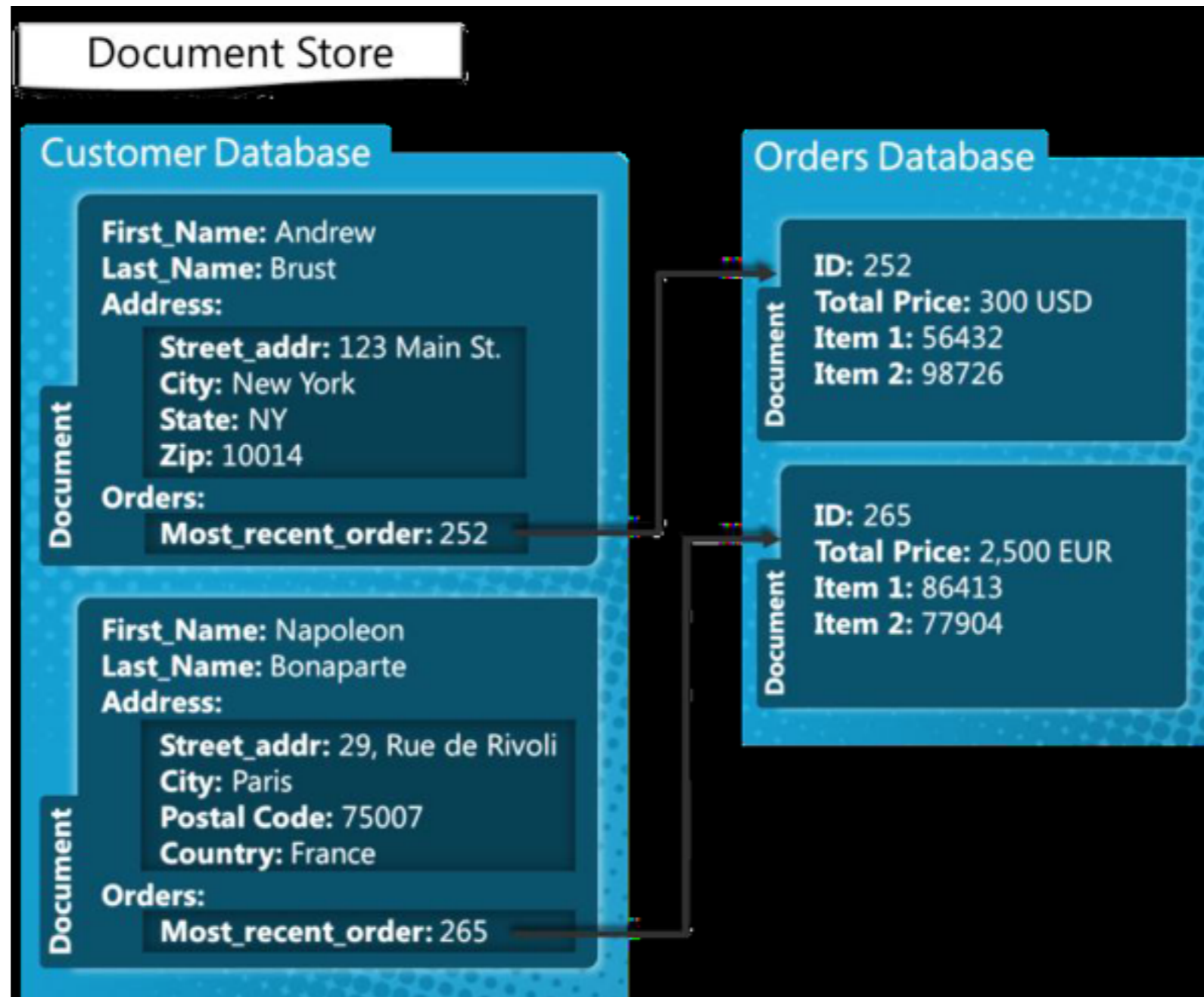
© 2012, I01 companies & Software Languages Team (University of Koblenz-Landau)

Relational Databases

- **Lineage:** E. F. Codd in A Relational Model of Data for Large Shared Data Banks
- **Data Model:** a set of relations
- **Example:** VoltDB, Clustrix, MySQL
- **Good at:** High performing, scalable OLTP. SQL access. Materialized views. Transactions matter. Programmer friendly transactions.

[<http://highscalability.com/blog/2011/6/20/35-use-cases-for-choosing-your-next-nosql-database.html>]

Document Databases



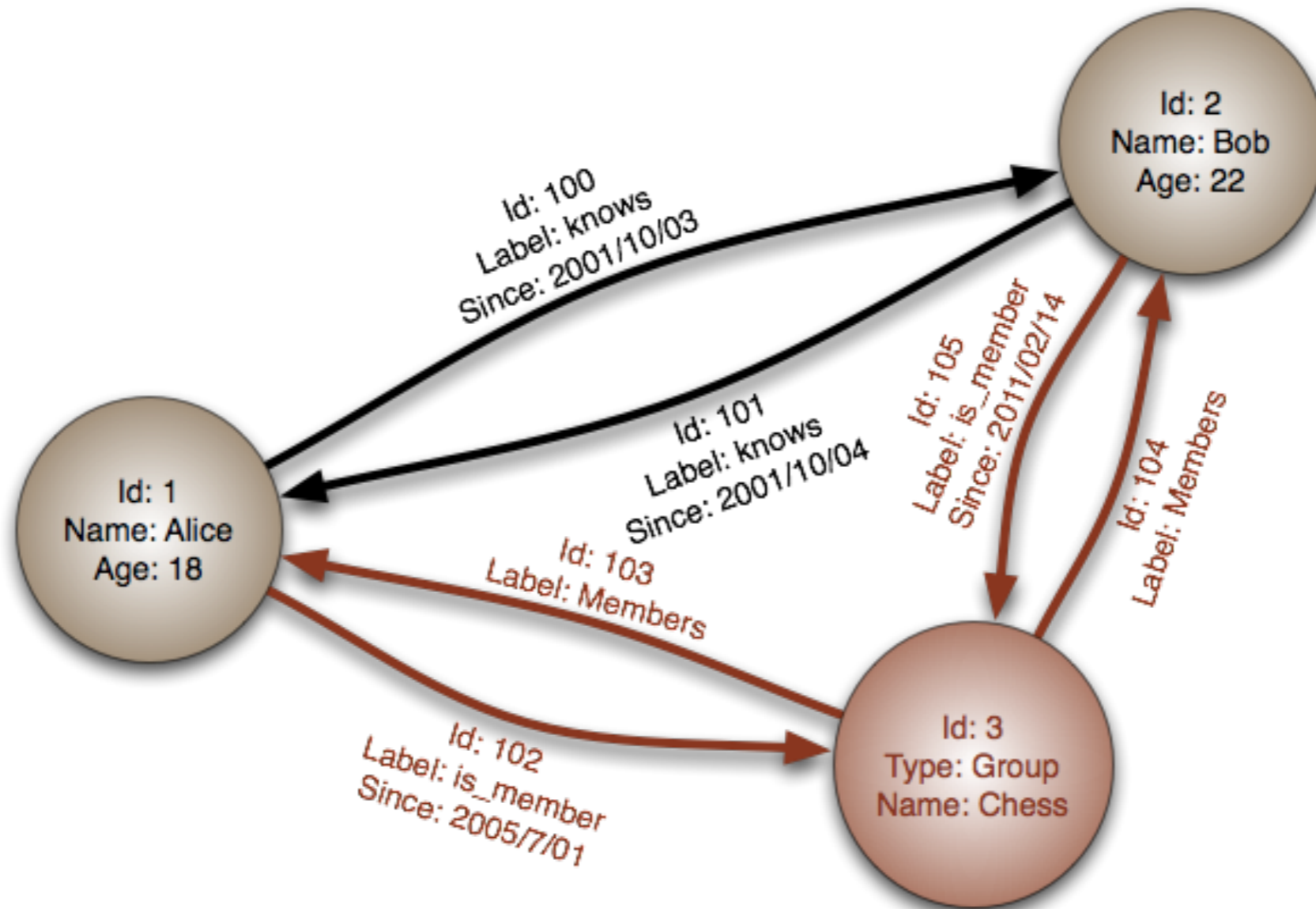
[<http://blogs.msdn.com/b/usisvde/archive/2012/04/05/getting-acquainted-with-nosql-on-windows-azure.aspx>]

Document Databases

- **Lineage:** Inspired by Lotus Notes.
- **Data model:** Collections of documents, which contain key-value collections.
- **Example:** CouchDB, MongoDB
- **Good at:** Natural data modeling. Programmer friendly. Rapid development. Web friendly, CRUD.

[<http://highscalability.com/blog/2011/6/20/35-use-cases-for-choosing-your-next-nosql-database.html>]

Graph Databases



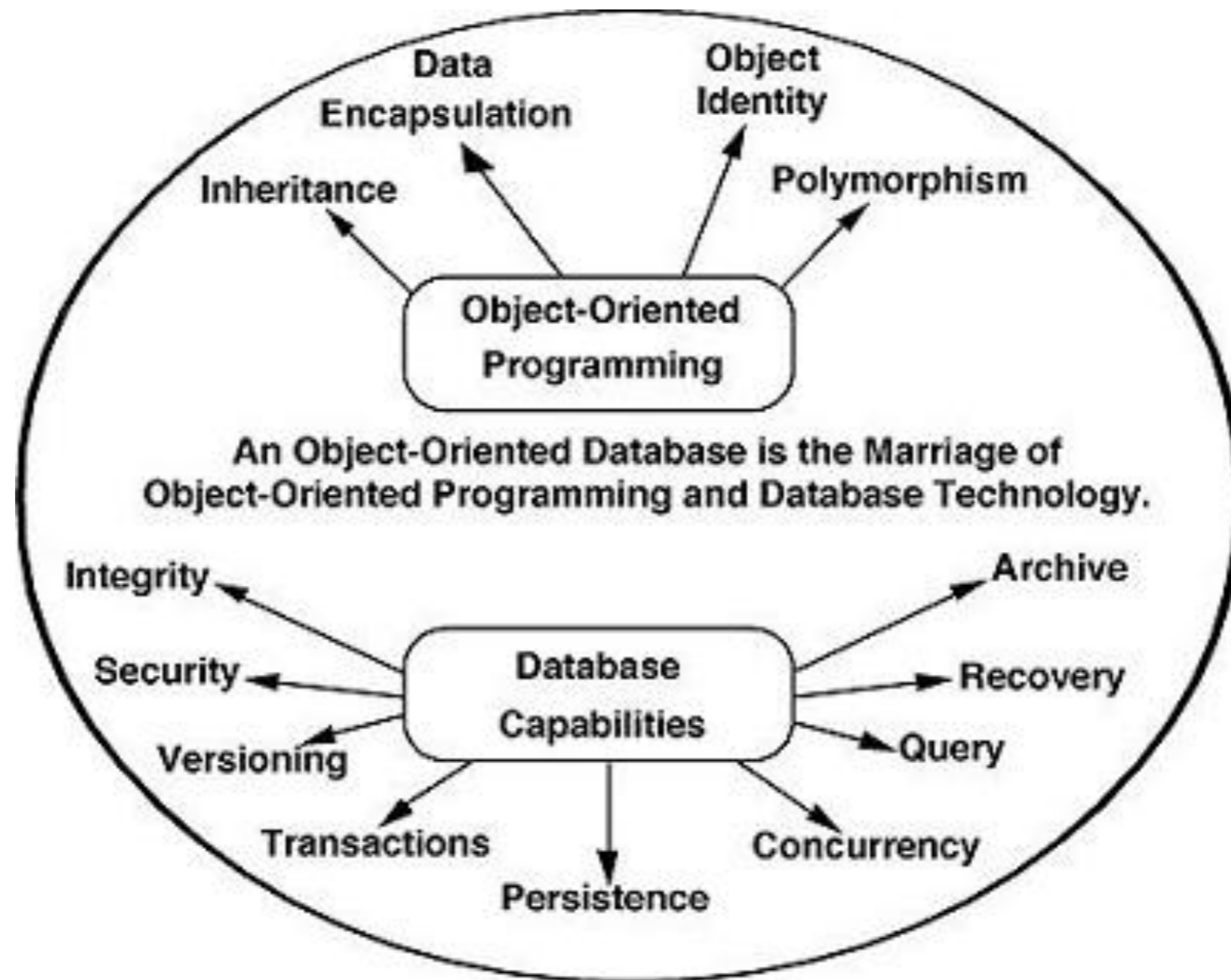
[http://en.wikipedia.org/wiki/Graph_database] 13 Sep 2012

Graph Databases

- **Lineage:** Euler and graph theory.
- **Data model:** Nodes & relationships, both which can hold key-value pairs
- **Example:** AllegroGraph, InfoGrid, Neo4j
- **Good at:** Rock complicated graph problems. Fast.

[<http://highscalability.com/blog/2011/6/20/35-use-cases-for-choosing-your-next-nosql-database.html>]

OO Databases



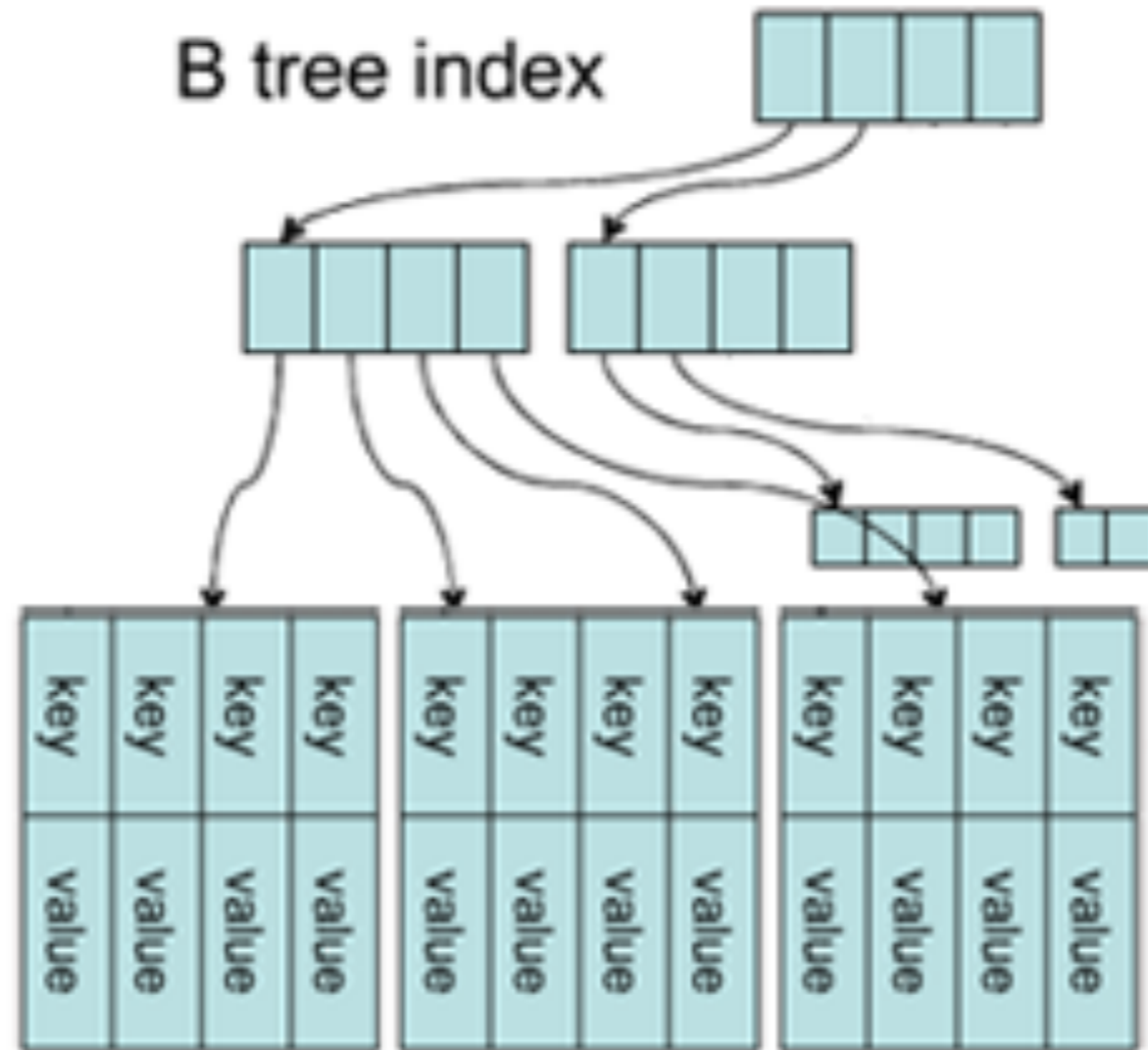
[<http://www.aspfree.com/c/a/Database/Introduction-to-RDBMS-OODBMS-and-ORDBMS/>]

OO Databases

- **Lineage:** Graph Database Research
- **Data Model:** Objects
- **Example:** Objectivity, Gemstone
- **Good at:** complex object models, fast key-value access, key-function access, and graph database functionality.

[<http://highscalability.com/blog/2011/6/20/35-use-cases-for-choosing-your-next-nosql-database.html>]

Key-Value Stores



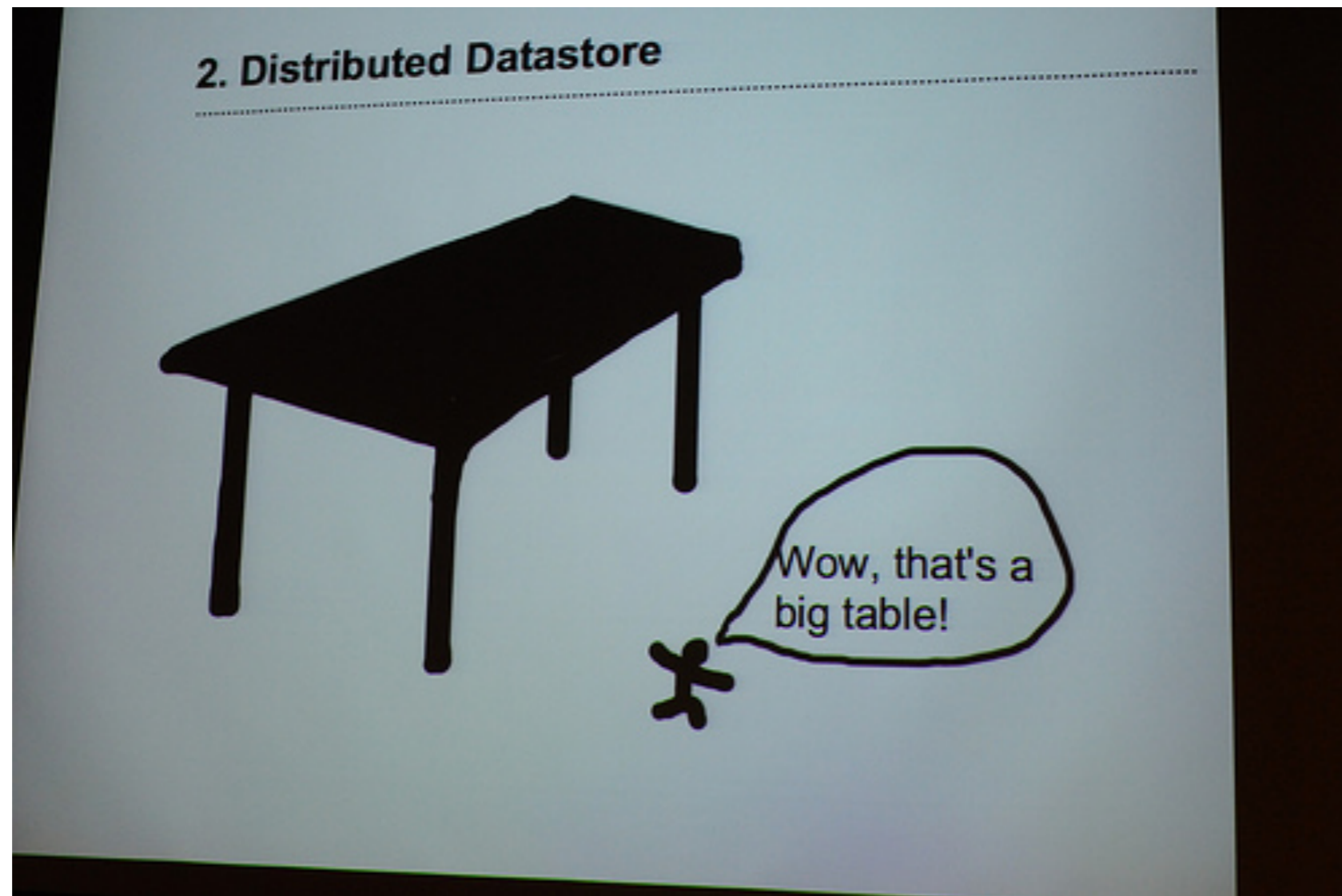
[<http://www.dotkam.com/2009/08/30/key-value-store-list/>]

Key-Value Stores

- **Lineage:** Amazon's Dynamo paper and Distributed Hash Tables.
- **Data model:** A global collection of KV pairs.
- **Example:** Membase, Riak
- **Good at:** Handles size well. Processing a constant stream of small reads and writes. Fast. Programmer friendly.

[<http://highscalability.com/blog/2011/6/20/35-use-cases-for-choosing-your-next-nosql-database.html>]

BigTable Clones



[<http://vicky2183.wordpress.com/2010/05/24/bigtable-is-the-database-used-by-google/>]

BigTable Clones

- **Lineage:** Google's BigTable paper.
- **Data model:** Column family, i.e., a tabular model where each row at least in theory can have an individual configuration of columns.
- **Example:** HBase, Hypertable, Cassandra
- **Good at:** Handles size well. Stream massive write loads. High availability. Multiple-data centers. MapReduce.

[<http://highscalability.com/blog/2011/6/20/35-use-cases-for-choosing-your-next-nosql-database.html>]

What's 'wrong' with relational databases?

Relational algebra

Indices

Stored procedures

SQL

A **relational database** is a collection of data items organized as a set of formally described tables from which data can be accessed easily. A relational database is created using the [relational model](#). The software used in a relational database is called a [relational database management system](#) (RDBMS). A relational database is the predominant choice in storing data, over other models like the [hierarchical database model](#) or the [network model](#).

[http://en.wikipedia.org/wiki/Relational_database] 12 Sep 2012

Primary keys

Foreign keys

Constraints

Strengths

- Serves needs of ‘information systems’ (OLTP)
- Supports ‘ad-hoc’ queries efficiently
- Declarative and general query language
- Well-understood formalization & implementation
- Data independence
- ACID transactions

ACID

- **Atomicity:** *All of the operations in the transaction will complete, or none will.*
- **Consistency:** *The database will be in a consistent state when the transaction begins and ends.*
- **Isolation:** *The transaction will behave as if it is the only operation being performed upon the database.*
- **Durability:** *Upon completion of the transaction, the operation will not be reversed.*

[<http://queue.acm.org/detail.cfm?id=1394128>]

Main issues

- Limited 'horizontal' scalability
- The infamous O/R impedance mismatch

Vertical scaling

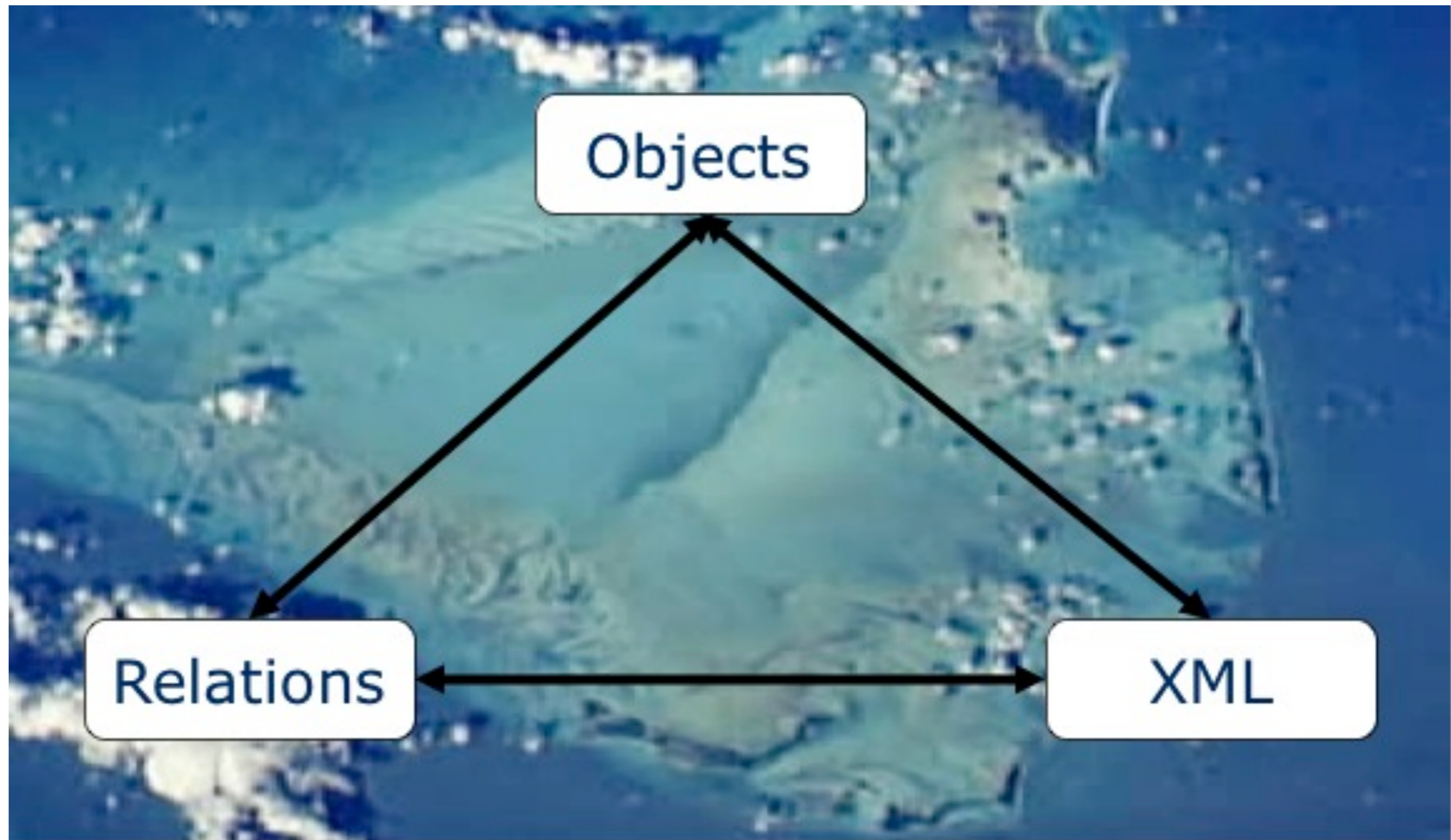
- General objective of scaling:
Deal with more data and more functions!
- Vertical scaling:
Use a larger (the largest) computer.
- Issues:
 - Limited
 - Expensive
 - Vendor lock

[<http://queue.acm.org/detail.cfm?id=1394128>]

The O/R impedance mismatch

- Object graphs versus normalized relations
- Identities versus keys
- Imperative versus declarative
- Inheritance versus lack thereof
- Data/behavior coupling versus data independence
- NULL versus null
- ...

The Bermuda Triangle of data processing



Objects versus relations

Objects

```
var _1579124585 = new Product
{
    Title = "The Right Stuff"
    Author = "Tom Wolfe",
    Year = 1979,
    Pages = 320,
    Keywords = new[] { "Book", "Hardcover", "American" },
    Ratings = new[] { "****", "4 stars" },
}
var Products = new[] { _1579124585 };
```

```
class Product
{
    string Title;
    string Author;
    int Year;
    int Pages;
    IEnumerable<string> Keywords;
    IEnumerable<string> Ratings;
}
```

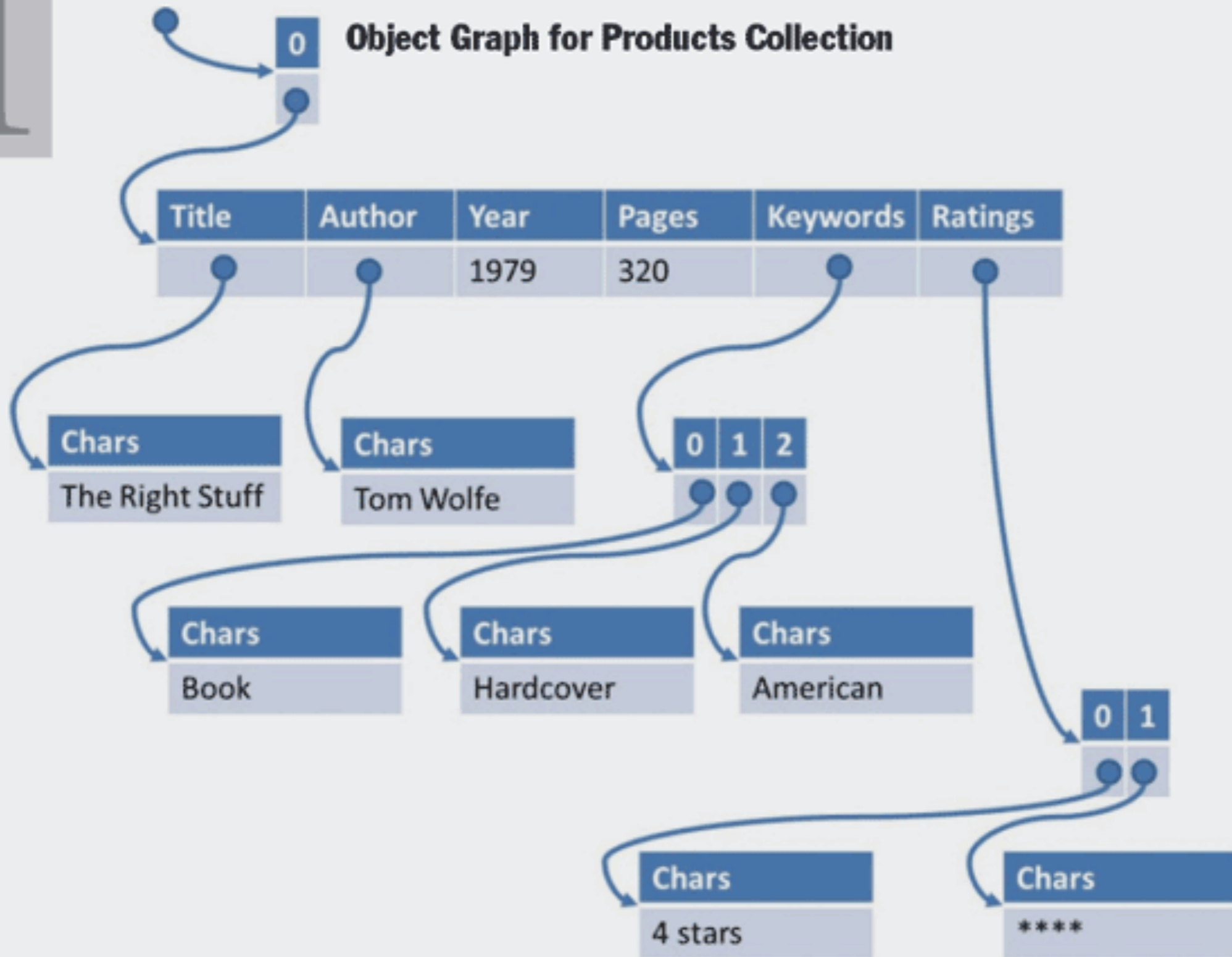
Object (graph)

The underlying class

[<http://queue.acm.org/detail.cfm?id=1961297>]

© 2012, IOI companies & Software Languages Team (University of Koblenz-Landau)

FIGURE 1



[<http://queue.acm.org/detail.cfm?id=1961297>]

© 2012, IOI companies & Software Languages Team (University of Koblenz-Landau)

Query for all products with '****' rating

LINQ syntax

```
var q = from product in Products
        where product.Ratings.Any(rating=>rating == "****")
        select new{ product.Title, product.Keywords };
```

```
var q = Products.Where(product=>
    product.Ratings.Any(rating=>rating == "****")).Select(product=>
    new{ product.Title, product.Keywords });
```

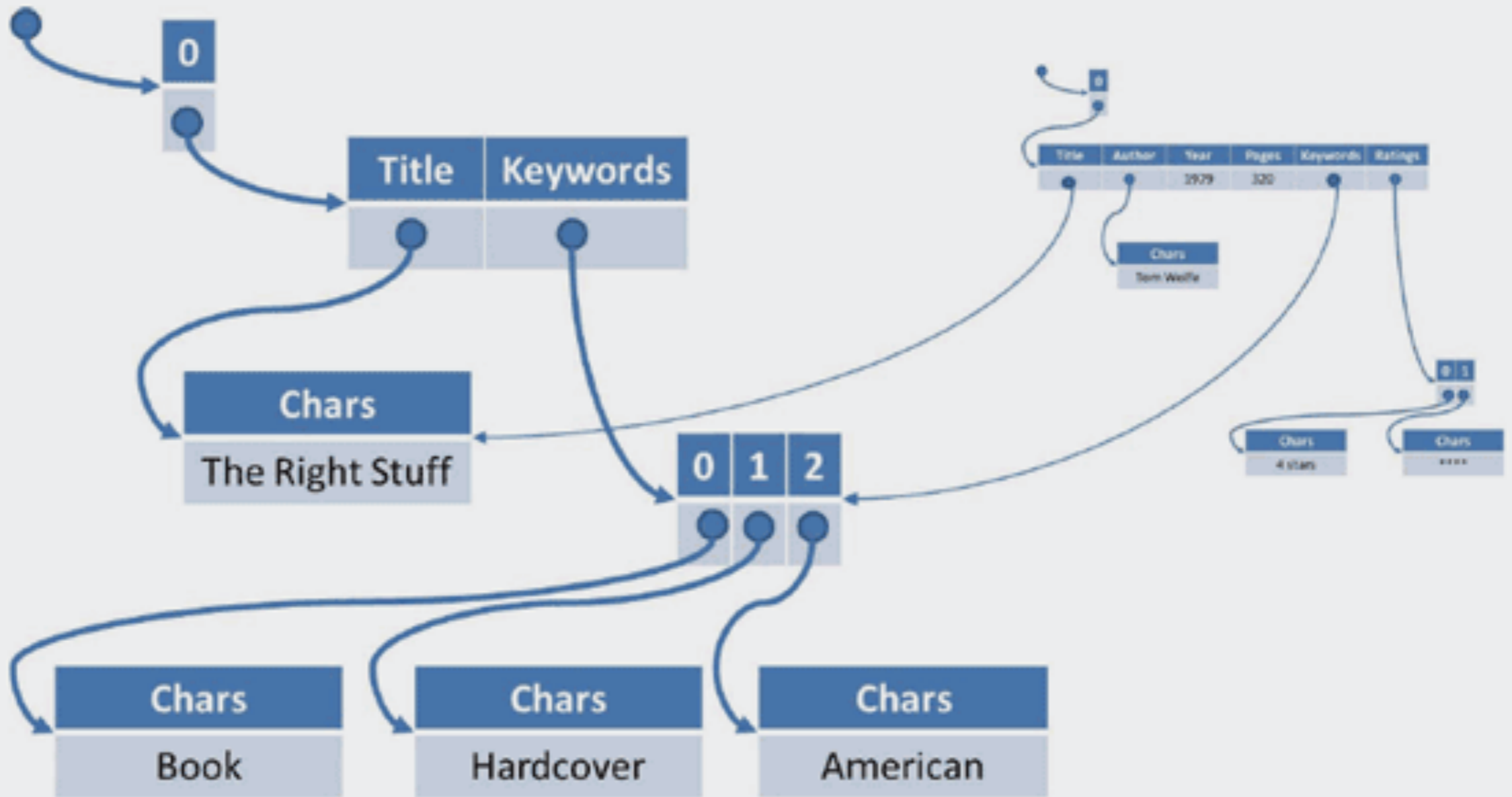
'unsugared'

[<http://queue.acm.org/detail.cfm?id=1961297>]

© 2012, IOI companies & Software Languages Team (University of Koblenz-Landau)

FIGURE 2

Object Graph for Books with Four-star Ratings



[<http://queue.acm.org/detail.cfm?id=1961297>]

© 2012, IOI companies & Software Languages Team (University of Koblenz-Landau)

Classes mimicking tables

```
class Products {  
  int ID;  
  string Title;  
  string Author;  
  int Year;  
  int Pages;  
}
```

```
class Keywords {  
  int ID;  
  string Keyword;  
  int ProductID;  
}
```

```
class Ratings {  
  int ID;  
  string Rating;  
  int ProductID;  
}
```

Needed for normalization

We assume a 1:n relationship.

[<http://queue.acm.org/detail.cfm?id=1961297>]

Populate 'tabelized' classes.

```
Products.Insert  
  ( 1579124585  
    , "The Right Stuff"  
    , "Tom Wolfe"  
    , 1979  
    , 320  
  );
```

```
Keywords.Insert  
  ( 4711, "Book"  
    , 1579124585  
  );  
Keywords.Insert  
  ( 1843, "Hardcover"  
    , 1579124585  
  );  
Keywords.Insert  
  ( 2012, "American"  
    , 1579124585  
  );  
Ratings.Insert  
  ( 787, "****"  
    , 1579124585  
  );  
Ratings.Insert  
  ( 747, "4 stars"  
    , 1579124585  
  );
```

[<http://queue.acm.org/detail.cfm?id=1961297>]

FIGURE 3**Relational Tables for Product Database**

Ratings

ID	Rating	ProductID
787	****	1579124585
747	4 stars	1579124585

Products

ID	Title	Author	Year	Pages
1579124585	The Right Stuff	Tom Wolfe	1979	304

Keywords

ID	Keyword	ProductID
4711	Book	1579124585
1843	Hardcover	1579124585
2012	American	1579124585

[<http://queue.acm.org/detail.cfm?id=1961297>]

© 2012, IOI companies & Software Languages Team (University of Koblenz-Landau)

Query for all products with '****' rating

```
var q = from product in Products
        from rating in Ratings
        from keyword in Keywords
        where product.ID == rating.ProductId
            && product.ID == keyword.ProductID
            && rating == "****"
        select new{ product.Title, keyword.Keyword };
```

LINQ works equally well as a query language on OO objects and tabelized objects.

[<http://queue.acm.org/detail.cfm?id=1961297>]

© 2012, IOI companies & Software Languages Team (University of Koblenz-Landau)

FIGURE 4

Tabular Result for Books with Four-star Ratings

Title	Keyword
The Right Stuff	Book
The Right Stuff	Hardcover
The Right Stuff	American

The query result is not normalized!

[<http://queue.acm.org/detail.cfm?id=1961297>]

Query for all products with '****' rating

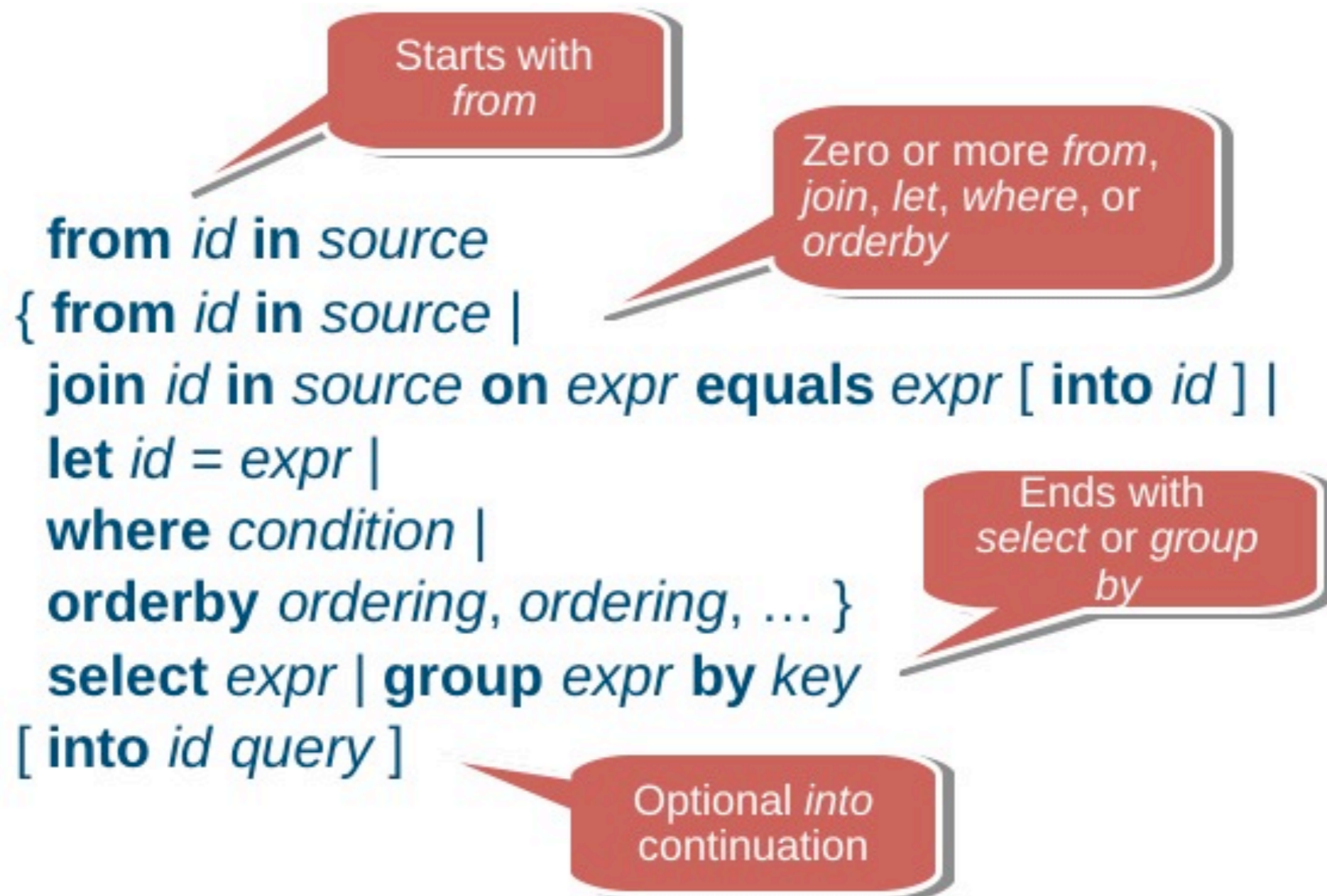
Use efficiency-aware join.

```
var q = from product in Products
        join rating in Ratings on product.ID equals rating.ProductId
        where rating == "****"
        select product into FourStarProducts
from fourstarproduct in FourStarProducts
join keyword in Keywords on product.ID equals keyword.ProductID
select new{ product.Title, keyword.Keyword };
```

[<http://queue.acm.org/detail.cfm?id=1961297>]

© 2012, IOI companies & Software Languages Team (University of Koblenz-Landau)

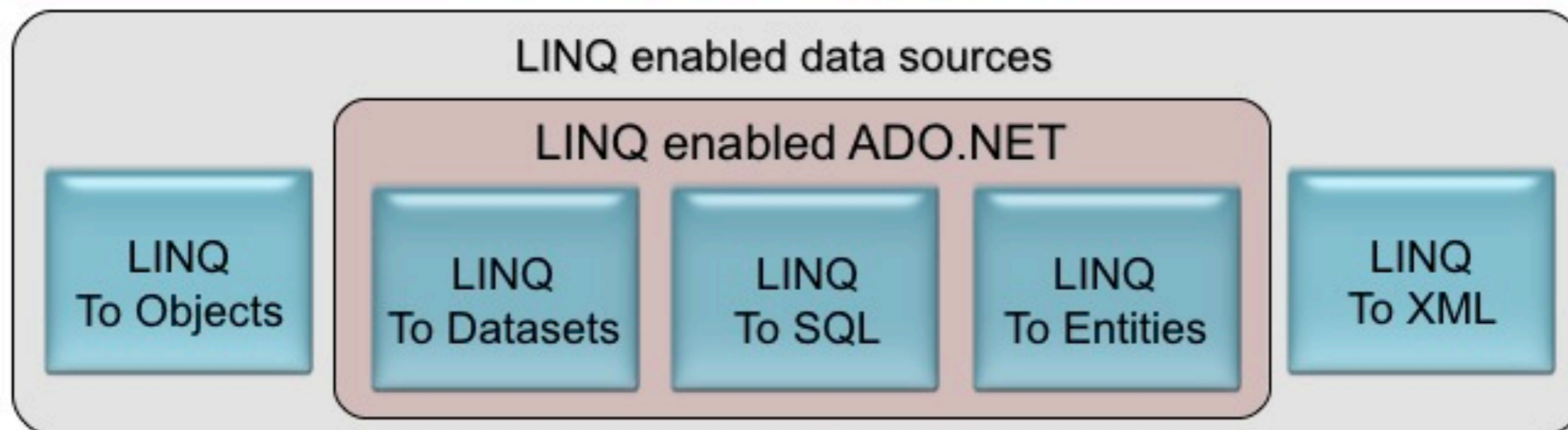
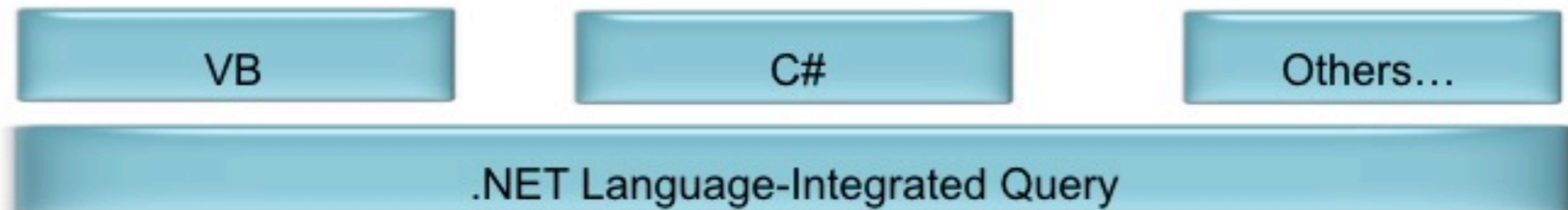
LINQ's query expressions



Standard Query Operators

Restriction	Where
Projection	Select, SelectMany
Ordering	OrderBy, ThenBy
Grouping	GroupBy
Quantifiers	Any, All
Partitioning	Take, Skip, TakeWhile, SkipWhile
Sets	Distinct, Union, Intersect, Except
Elements	First, FirstOrDefault, ElementAt
Aggregation	Count, Sum, Min, Max, Average
Conversion	ToArray, ToList, ToDictionary
Casting	OfType<T>

Language INtegrated Query (LINQ)



Objects



Relational



XML

LINQ provides one programming model for all types of data (objects, SQL, XML, DataSets)

Annotate OO-style classes to associate with tables

```
[Table(name="Products")]
class Product
{
    [Column(PrimaryKey=true)]int ID;
    [Column]string Title;
    [Column]string Author;
    [Column]int Year;
    [Column]int Pages;
    private EntitySet<Rating> _Ratings;
    [Association( Storage="_Ratings", ThisKey="ID",OtherKey="ProductID"
, DeleteRule="ONDELETECASCADE" )]
    ICollection<Rating> Ratings{ ... }

    private EntitySet<Keyword> _Keywords;
    [Association( Storage="_Keywords", ThisKey="ID"
, OtherKey="ProductID", DeleteRule="ONDELETECASCADE" )]
    ICollection<Keyword> Keywords{ ... }
}

[Table(name="Keywords")]
class Keyword { ... }

[Table(name="Ratings")]
class Rating { ... }
```

[<http://queue.acm.org/detail.cfm?id=1961297>]

© 2012, IOI companies & Software Languages Team (University of Koblenz-Landau)

Query for all products with '****' rating

Before normalization

```
var q = from product in Products
        where product.Ratings.Any(rating=>rating == "****")
        select new{ product.Title, product.Keywords };
```

```
var q = from product in Products
        where product.Ratings.Any(rating=>rating.Rating == "****")
        select new{ product.Title, product.Keywords };
```

After normalization

[<http://queue.acm.org/detail.cfm?id=1961297>]

Summary

versus

Objects	Relations
Parents point to children	Children point to parents
Intensional identity	Extensional identity
'Active' objects needed	Transactions available
Compositional	Non-compositional
Weak references	Referential transparency
Open world	Closed world

[<http://queue.acm.org/detail.cfm?id=1961297>]

© 2012, IOI companies & Software Languages Team (University of Koblenz-Landau)

Characteristics of NoSQL

Characteristics of a NoSQL database

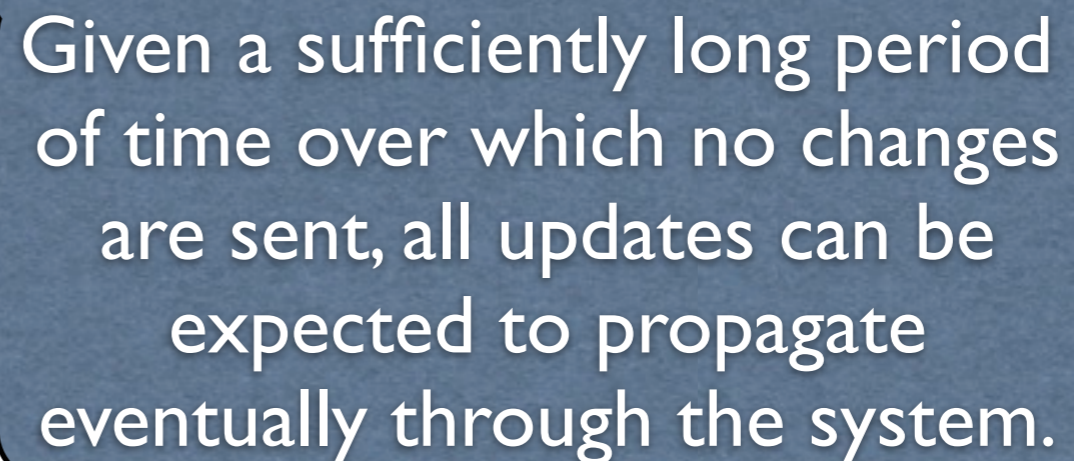
- **It does not use SQL as its query language.**
- It manages large volumes of data w/o fixed schema.
- Data is partitioned among different machines.
- JOIN operations are not quite usable.

[<http://en.wikipedia.org/wiki/NoSQL>] 12 Sep 2012

© 2012, I0I companies & Software Languages Team (University of Koblenz-Landau)

Characteristics of a NoSQL database

- **It may not give full ACID guarantees.**
- Eventual consistency is usually guaranteed.



Given a sufficiently long period of time over which no changes are sent, all updates can be expected to propagate eventually through the system.

[<http://en.wikipedia.org/wiki/NoSQL>] 12 Sep 2012

Characteristics of a NoSQL database

- **It has a distributed, fault-tolerant architecture.**
- Data is held in a redundant manner on multiple servers.
- Scaling out is achieved by adding more servers.
- Failure of a server can be tolerated.

[<http://en.wikipedia.org/wiki/NoSQL>] 12 Sep 2012

© 2012, I01 companies & Software Languages Team (University of Koblenz-Landau)

Characteristics of a NoSQL database

- **It is highly optimized and specialized.**
- Retrieve and append operations are optimized.
- Functionality focuses on record storage (key-value stores).
- It is much less flexible than full SQL systems.
- The key objective is scalability and performance.

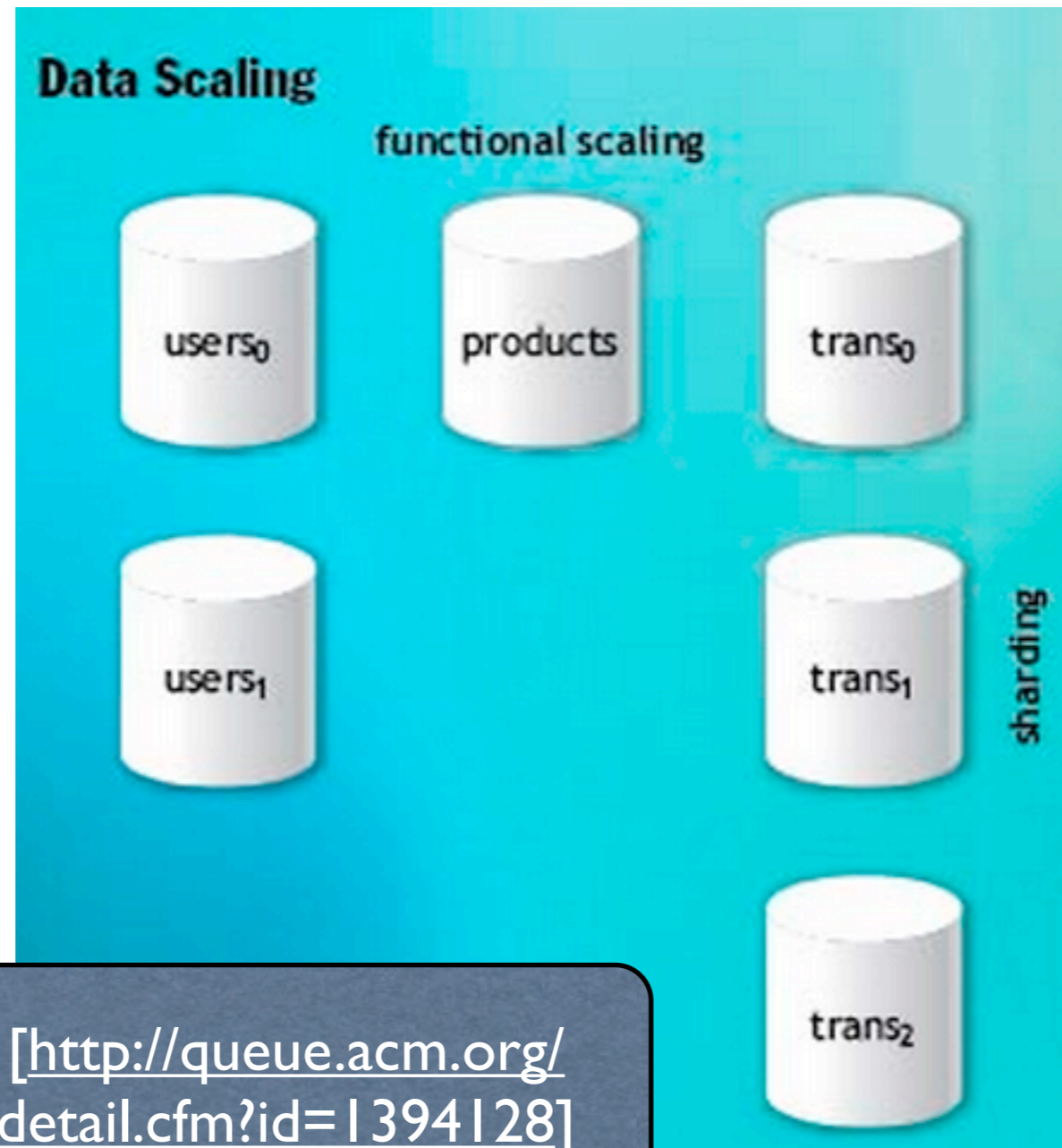
[<http://en.wikipedia.org/wiki/NoSQL>] 12 Sep 2012

Horizontal scaling

- Again: Deal with more data and more functions.
- **Vector 1** -- Functional scaling: group data by function and spread functional groups across databases.
- **Vector 2** -- Data scaling ('sharding'): Split data within functional areas across multiple databases.

[<http://queue.acm.org/detail.cfm?id=1394128>]

Horizontal scaling

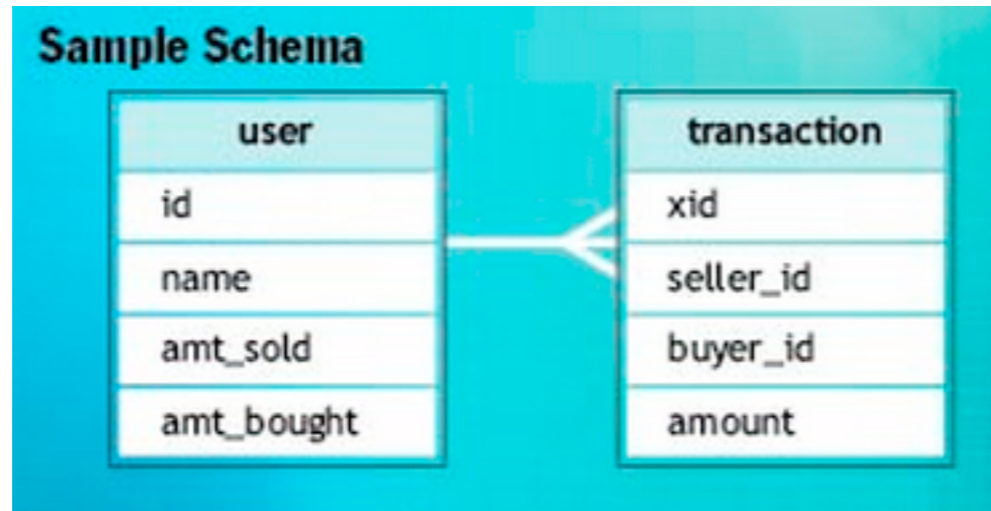


Problem with ACID

- Distribution requires 2-phase commit.
 - Each database precommits.
 - Vetoes may cause rollback.
 - Without vetoes, each database commits.

Availability is reduced
because it is product of
components' availabilities.

Assume functional scaling



Assume user and transaction to reside on different nodes.

ACID style

Begin transaction

Insert into transaction(xid, seller_id, buyer_id, amount);

Update user set amt_sold=amt_sold+\$amount where id=\$seller_id;

Update user set amt_bought=amount_bought+\$amount where id=\$buyer_id;

End transaction

[<http://queue.acm.org/detail.cfm?id=1394128>]

A Use Case for NoSQL

Tunable CAP tradeoffs. Are a few drops OK? Does your app need strong or weak consistency? Is availability more important or is consistency? Will being down be more costly than being wrong? It's nice to have products that give you a choice.

CAP =

- Consistency:** Sets of operations appear to occur at once.
- + **Availability:** Operations terminate in intended responses.
- + **Partition tolerance:** Operations complete despite unavailable components.

[<http://highscalability.com/blog/2010/12/6/what-the-heck-are-you-actually-using-nosql-for.html>]

BASE (basically available, soft state, eventually consistent)

Availability is achieved through *supporting partial failures without total system failure.*

Example: if users are partitioned across five database servers, BASE design encourages crafting operations in such a way that a user database failure impacts only the 20 percent of the users on that particular host.

[<http://queue.acm.org/detail.cfm?id=1394128>]

BASE (basically available, soft state, eventually consistent)

Consistency is achieved through *applying consistency patterns*.

Options:

- (i) Relax contract for reduced consistency.
- (ii) Use message queue to achieve eventual consistency.
- (iii) ...

[<http://queue.acm.org/detail.cfm?id=1394128>]

Reduced consistency

Begin transaction

```
Insert into transaction(id, seller_id, buyer_id, amount);
```

End transaction

Begin transaction

```
Update user set amt_sold=amt_sold+$amount where id=$seller_id;
```

```
Update user set amt_bought=amount_bought+$amount
```

```
where id=$buyer_id;
```

End transaction

If the contract stipulates that the running totals are estimates, this may be adequate.

[<http://queue.acm.org/detail.cfm?id=1394128>]

Eventual consistency

Begin transaction

```
Insert into transaction(id, seller_id, buyer_id, amount);
```

```
Queue message "update user("seller", seller_id, amount)";
```

```
Queue message "update user("buyer", buyer_id, amount)";
```

End transaction

For each message in queue

Begin transaction

```
Dequeue message
```

```
If message.balance == "seller"
```

```
Update user set amt_sold=amt_sold + message.amount  
where id=message.id;
```

```
Else
```

```
Update user set amt_bought=amt_bought + message.amount  
where id=message.id;
```

```
End if
```

End transaction

End for

The user is eventually updated and consistency is reached **unless failure occurs after dequeuing.**

A more advanced pattern could be used.

[<http://queue.acm.org/detail.cfm?id=1394128>]

Use cases of NoSQL

General Use Cases of NoSQL

Bigness

Write availability

Massive write performance

Schema migration

Fast key-value access

Distributed systems support

Programmer ease of use

Flexible schema and flexible datatypes

No single point of failure

Generally available parallel computing

More Specific Use Cases

- Run relational queries
- Complex transactions
- Secondary indexes
- Ever-growing data
- Social network ops
- Deep join depth
- Write always
- Multiple data centers
- Mobile platform
- Integrate with services
- Scale out
- Tolerate faults
- Fast in-memory access
- CRUD without joins
- Programmer-friendly
- Access patterns
- Dynamic properties
- Behavior close to data

Poor use cases

- OLTP
- Data integrity
- Data independence
- SQL
- Ad-hoc queries
- Complex relationships
- Maturity and stability

[<http://highscalability.com/blog/2010/12/6/what-the-heck-are-you-actually-using-nosql-for.html>]

Summary

You learned about ...

- limitations of relational databases,
- the O/R impedance mismatch,
- some bits of O/R mapping,
- characteristics of NoSQL databases,
- and use cases for NoSQL databases.

Resources

- **BASE: An Acid Alternative:**

<http://queue.acm.org/detail.cfm?id=1394128>

- **A co-Relational Model of Data for Large Shared Data Banks:**

<http://queue.acm.org/detail.cfm?id=1961297>

- **NOSQL Patterns:**

<http://horicky.blogspot.de/2009/11/nosql-patterns.html>

- **35+ Use Cases For Choosing Your Next NoSQL Database:**

<http://highscalability.com/blog/2011/6/20/35-use-cases-for-choosing-your-next-nosql-database.html>

- **NoSQL in the Enterprise**

<http://www.oddbms.org/download/WP-DataStax-NoSQL.pdf>

More resources

- Future of Data Architecture: NoSQL, Big Data, Linked Data and the Cloud:

<http://www.infoq.com/presentations/NoSQL-Panel-QCon-London-2012>

- The Impedance Imperative --
Tuples + Objects + Infosets = Too Much Stuff!:

http://www.jot.fm/issues/issue_2003_09/column1/

- Datalog and Emerging Applications:

<http://www.cs.ucdavis.edu/~green/papers/sigmod906t-huang.pdf>