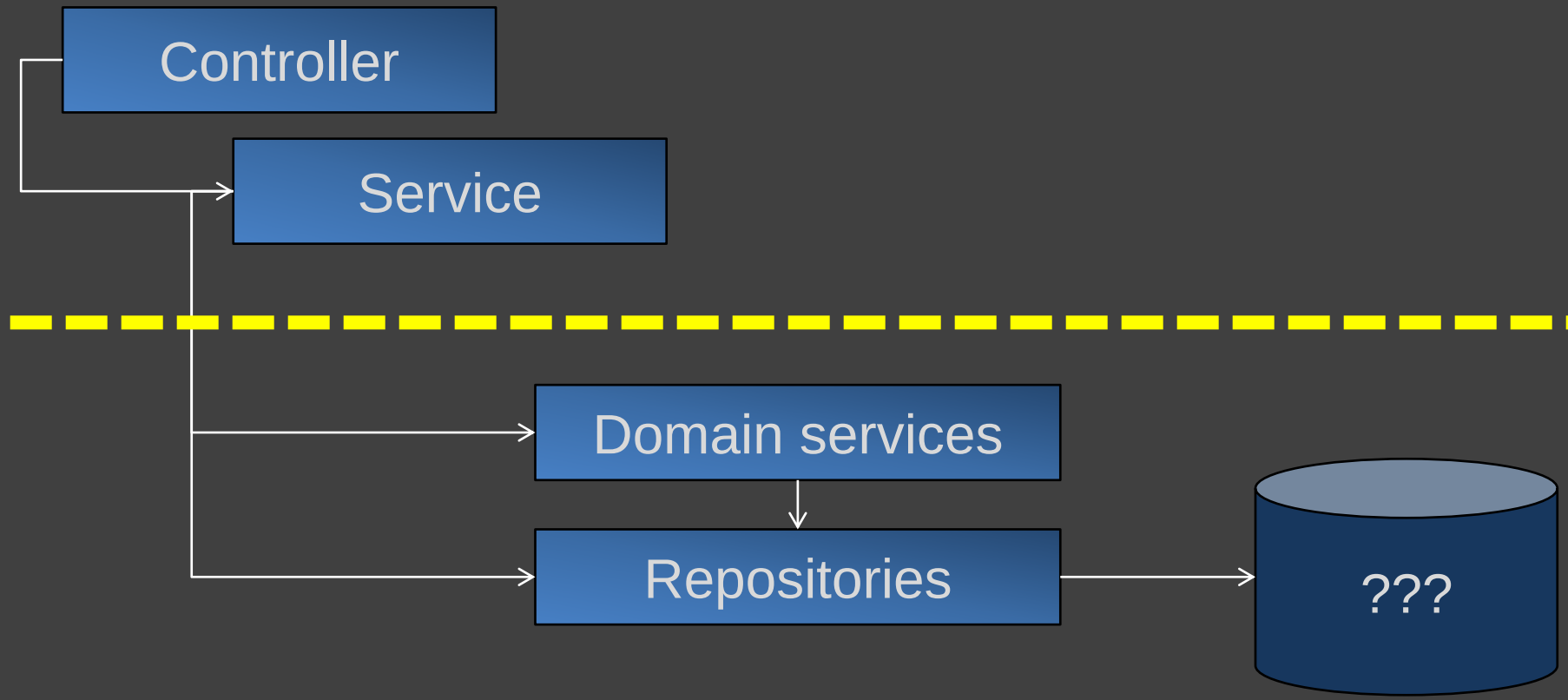
Polyglot Persistence

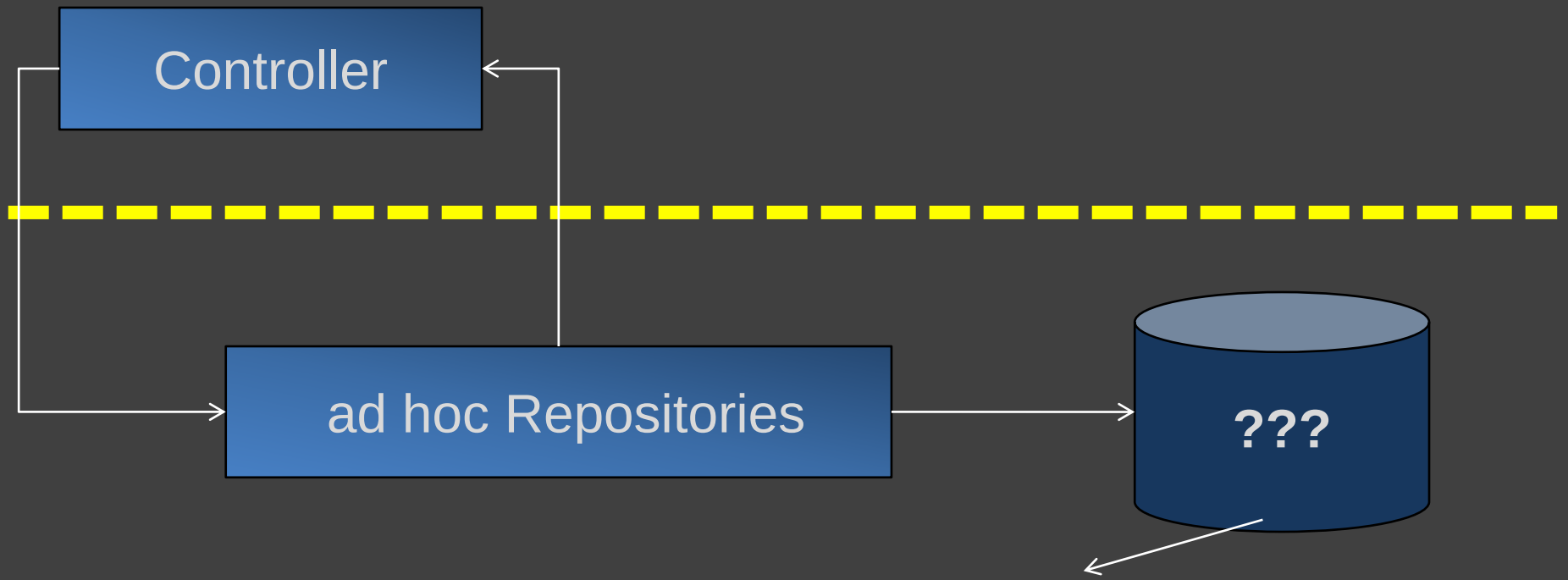
for the Common Application

JetBRAINS Dino Esposito

Persistence for the common apps – part 1



Persistence for the common apps – part 2

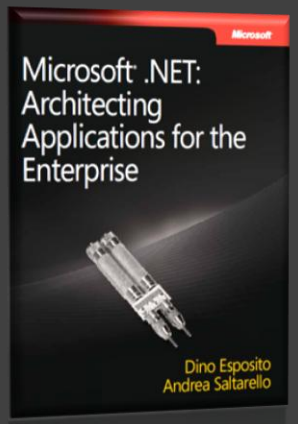


Same DB where you stored data
Ad hoc DB to read data from
Ad hoc API (HTTP, memory)

There's life beyond

relational.

> More and more, it gets hard to fit real data into the rigid schema of a relational model and more often than not a bit of redundancy helps saving queries thus making the application faster. No hype and no religion—it's just the evolution of the business and tools.



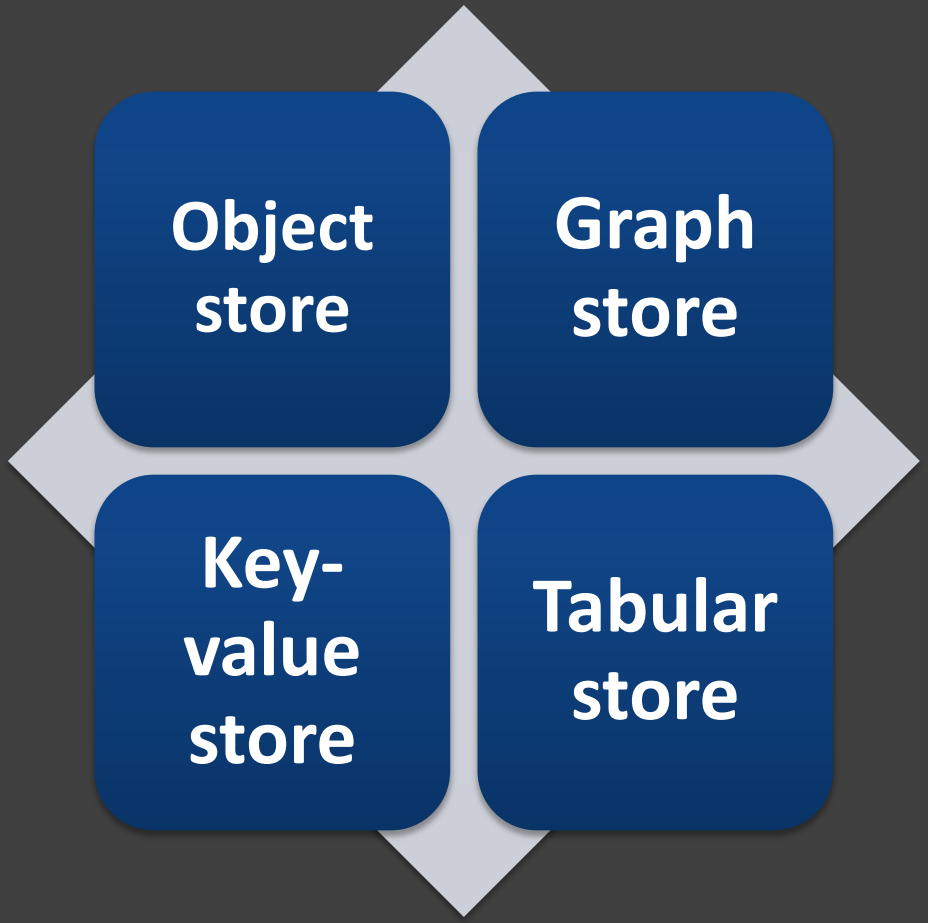
— NAA4E, Ch14

Relational vs Rest-of-world

- **Relational databases are not dead (and not even unhealthy)**
- **NoSQL stores are just another tool that may or may not be helpful**

If you're an architect who builds software for a client, your responsibility is understanding the mechanics of the domain and the characteristics of the data involved and then working out the best architecture possible.

Getting Familiar with NoSQL



Object stores

- **Used as plain relational stores**
  - Saved, indexed, queried
- **No records, just classes**
- **Each item has its own schema**
  - Retrieved by ID or through query
- **Abstracted to a collection of properties**

Graph stores

- Saves arbitrarily complex collections of data elements
- Support relationships between data elements

Key-value stores

- Work like a huge dictionary made of two fields—key and value
- Stored data is retrieved by key
- Values are serialized as JSON data

Tabular stores

- Differ from relational stores because of lack of normalization
  - Only 1NF fulfilled

John	Foo	1234
Jim	Bar	4567
Jim	Bar	0489

1	John	Foo
2	Jim	Bar

1	1234
2	4567
2	0489

No-SQL?

- **(Unpredictably large) volumes of data**
  - Millions of users
  - Thousands of queries per second
- **Semi-structured data coming in different forms, and needing same treatment**
  - Polymorphic data
- **Event stores**

Most compelling reason to look into **No-SQL** stores (document) is the ability to treat schemaless data

Schemaless

- Many different schemas
- Unpredictably changing
- Polymorphic data—same treatment
- Huge number of properties

Schemaless

- **Schemaless scenarios can be handled in a relational way**
  - SQL Server 2014 column-store
  - Ad hoc, specific data models
- **Same tools, nothing to learn**

Schemaless

- **Relational tables can grow fairly big**
  - Big enough for common applications
- **Relational tables can be sharded and scaled out**
  - Though costs of sharding are on you
- **Ecosystem of tools and products**
  - Large expertise within and outside IT departments

Summary of (relational) downsides

- Limited support for complex base types in both reading and writing via SQL (need of O/RM).
- Knowledge of the database structure is required to create ad hoc queries.
- Indexing over a large number of records (millions of rows) becomes slow.

Eventual consistency

core difference between relational and others

Consistency

- Eventual consistency is when reads and writes aren't aligned to the same data.
- Most NoSQL systems are eventually consistent in the sense that they guarantee that **if no updates are made to a given object for a sufficient period of time**, a query returns what the last command has written.

Problem of consistency

- When the system grows big you can't expect full consistency
- Expect full consistency in bounded context
- In some business scenarios eventual consistency is not acceptable (i.e., banking)
- You can force NoSQL to be consistent
  - But taking away the very part of it …

Polyglot persistence is when you know about all these problems and feel inspired to try to take the best of both relational and **No-SQL** worlds.

Generic e-commerce system

Data to handle

- Customers, orders, payments, products, and all that relates to a business transaction
- What you found out that your users prefer
- Documents (PDF of invoices/receipts), SVG maps of stores/venues
- Logs of user's activity (liked, clicked, viewed)
- Correlations: users living in the same area buying similar products, having "other" in common

- Customers, orders, payments, products
  - SQL Server or Azure SQL database via EF
- User preferences
  - Azure table storage
- Documents (PDF of invoices/receipts), SVG maps of stores/venues
  - RavenDB or MongoDB
- Logs of user's activity (liked, clicked, viewed)
  - Column store such as Cassandra
- Correlations
  - Graph DB such as Neo4j

Issues

- Single vendor
- Pet technology
- Resistance to change

If you're not in trouble with it

...

keep on using SQL

Thank you!