From ddd to DDD

An ongoing journey

data-driven development

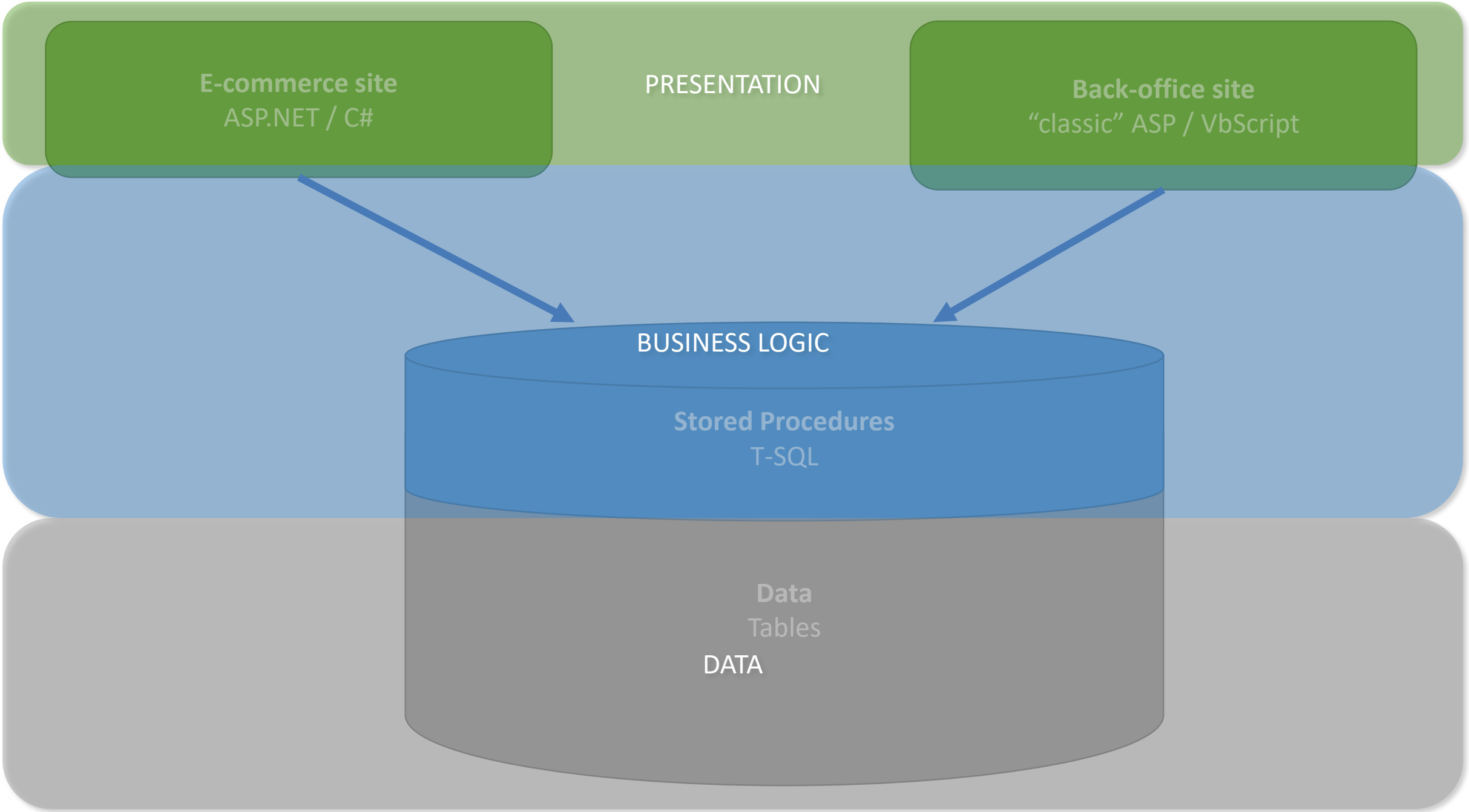
Domain-driven Design

Thibaud Desodt - @tsimbalar

What this is about

- Me, myself and I
- Code
- Architecture
- Learning process

Where it all started

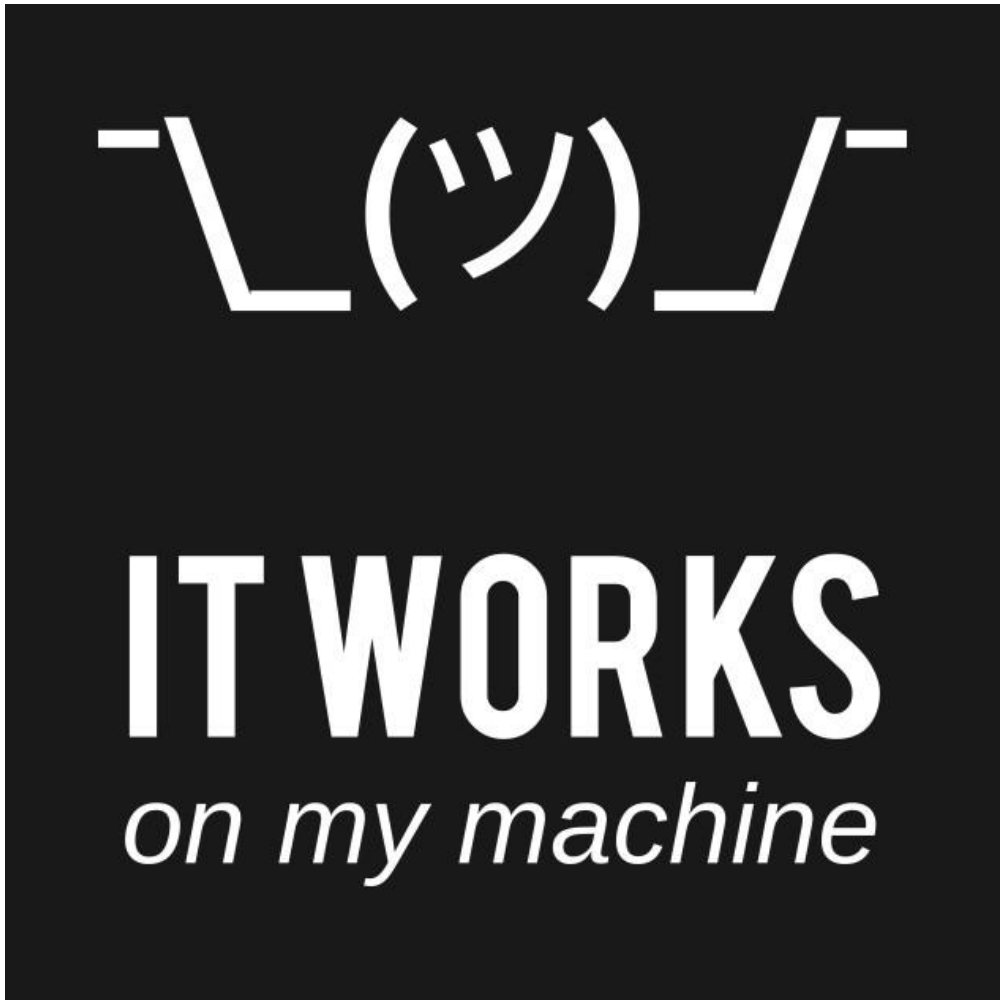

E-commerce site
ASP.NET / C#
PRESENTATION
Back-office site
“classic” ASP / VbScript
BUSINESS LOGIC
Stored Procedures
T-SQL
Data
Tables
DATA

And it was …

- Not super pleasant
  - VbScript
  - T-SQL
- Fragile
  - No automated tests
  - Tight coupling
    - Schema -> Stored Procs -> Website / admin site

… but it worked “well enough”!

- Customer value
  - Happy customer
  - Fast delivery of features
  - Reasonable perf
- Easy to work on
  - 1 feature ≈ 1 stored procedure



It worked well ... in **that** context

- Working alone on project
- Version 1 of the product
- Well-defined requirements
- Tight interactions with customer
- Simple domain

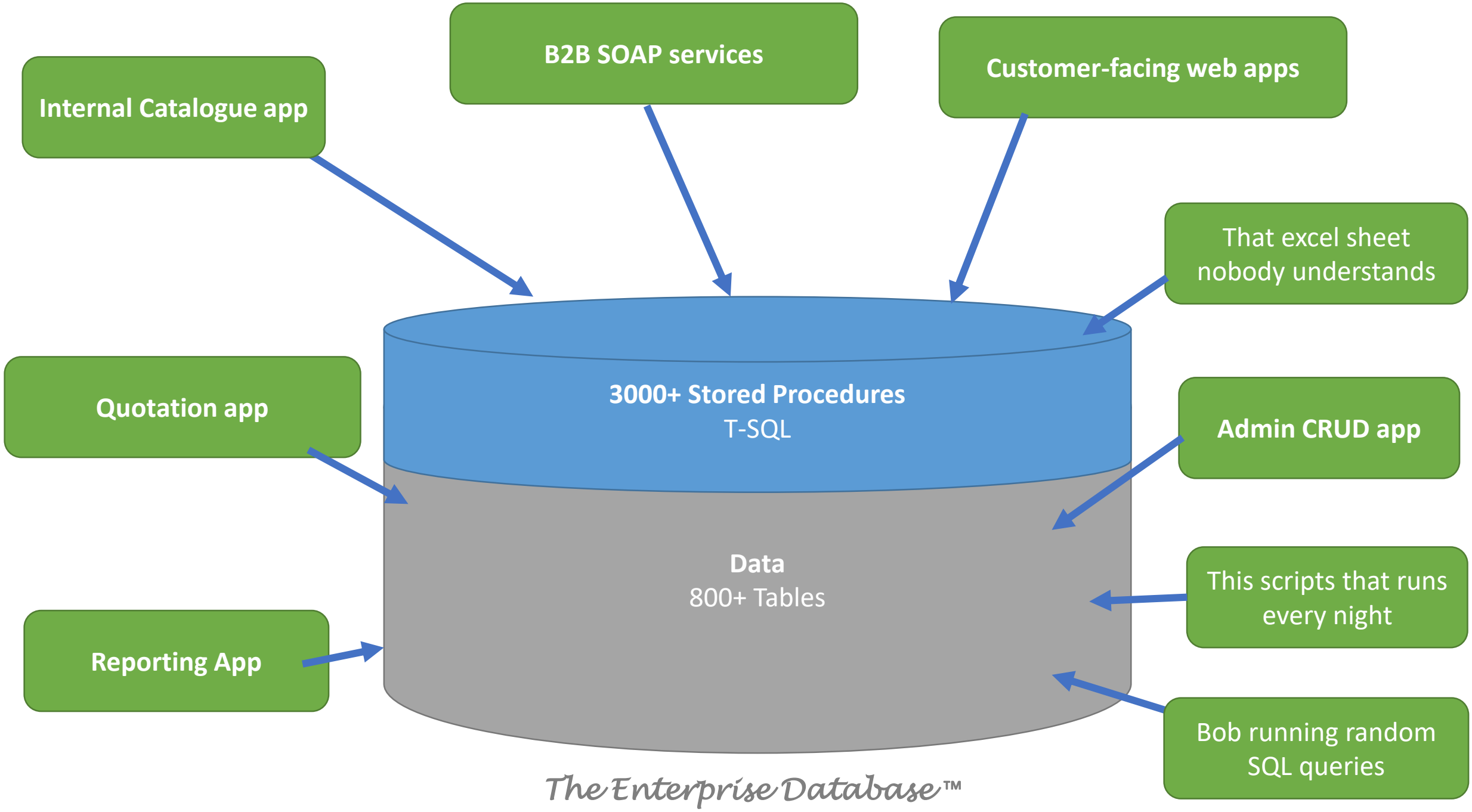
= ideal greenfield project

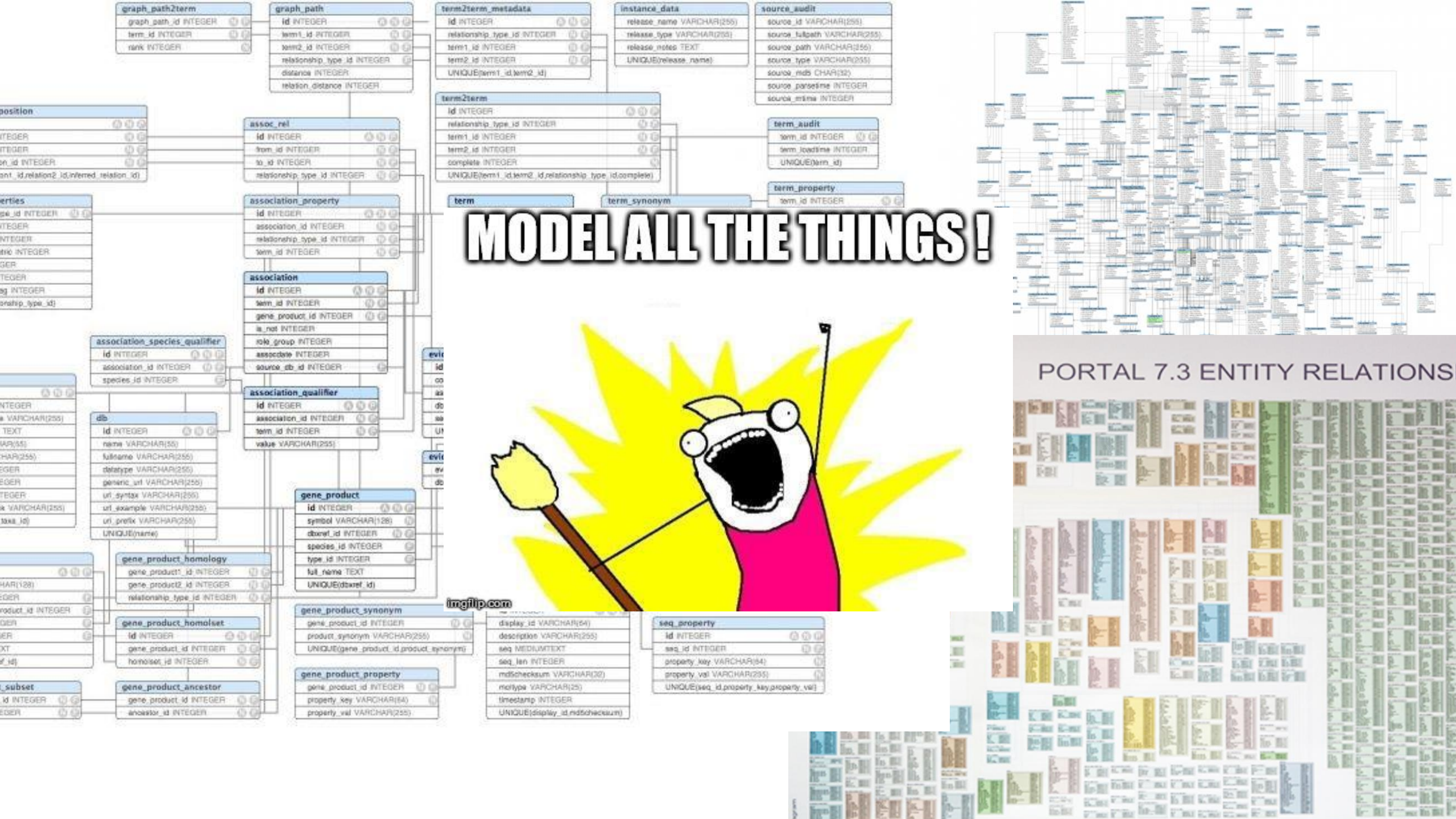
A few years later …

Similar approach, different context

A different context

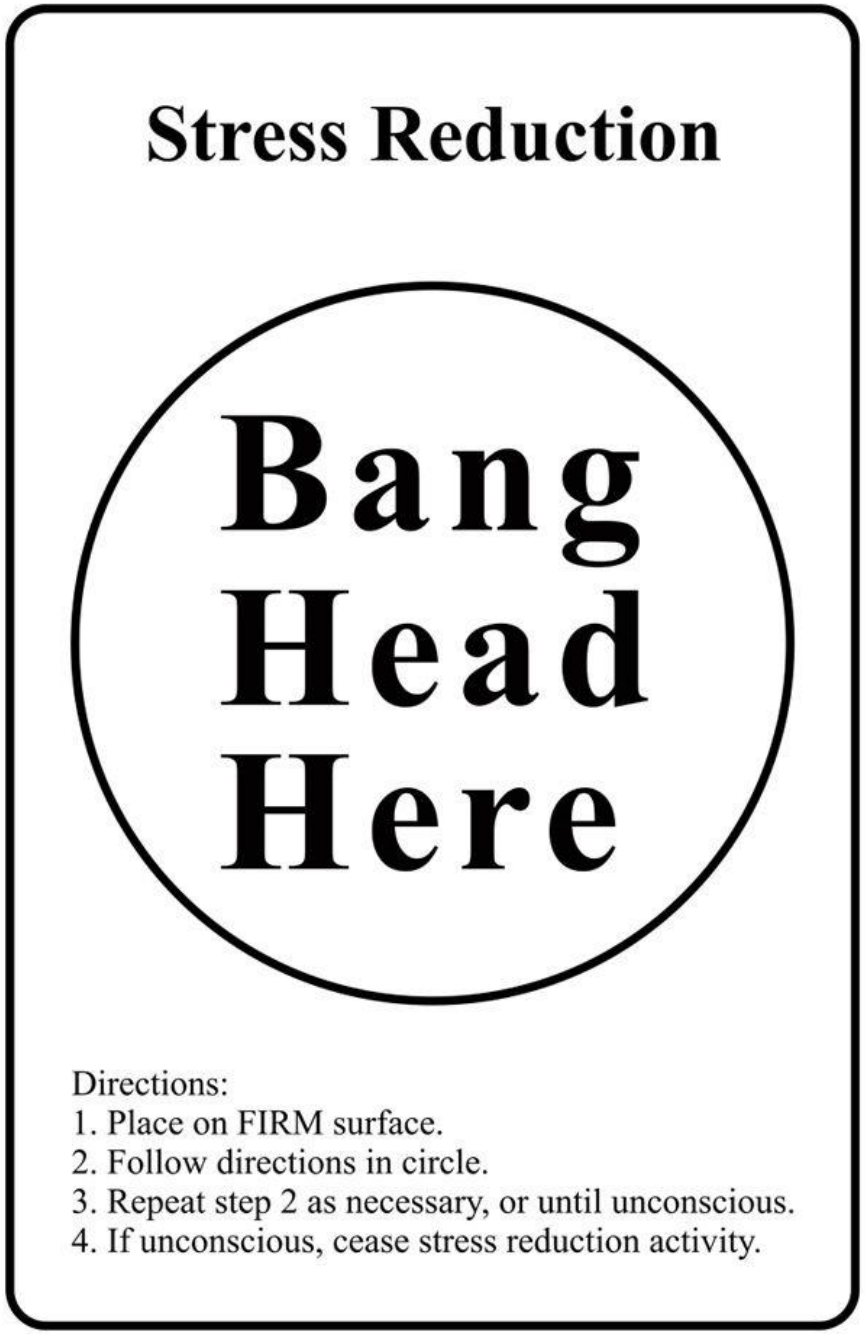
- 40,000+ employees company
- 100s of developers spread across the globe
- Many different systems accessing the database(s)


Internal Catalogue app
B2B SOAP services
Customer-facing web apps
That excel sheet nobody understands
Quotation app
3000+ Stored Procedures
T-SQL
Admin CRUD app
Data
800+ Tables
This scripts that runs every night
Reporting App
Bob running random SQL queries
The Enterprise Database™


MODEL ALL THE THINGS !
PORTAL 7.3 ENTITY RELATIONS
graph_path2term
graph_path_id INTEGER
term_id INTEGER
rank INTEGER
graph_path
id INTEGER
term1_id INTEGER
term2_id INTEGER
relationship_type_id INTEGER
distance INTEGER
relation_distance INTEGER
term2term_metadata
id INTEGER
relationship_type_id INTEGER
term1_id INTEGER
term2_id INTEGER
UNIQUE(term1_id,term2_id)
instance_data
release_name VARCHAR(255)
release_type VARCHAR(255)
release_notes TEXT
UNIQUE(release_name)
source_audit
source_id VARCHAR(255)
source_fullpath VARCHAR(255)
source_path VARCHAR(255)
source_type VARCHAR(255)
source_mdb CHAR(32)
source_parsetime INTEGER
source_mtime INTEGER
assoc_rel
id INTEGER
from_id INTEGER
to_id INTEGER
relationship_type_id INTEGER
term2term
id INTEGER
relationship_type_id INTEGER
term1_id INTEGER
term2_id INTEGER
complete INTEGER
UNIQUE(term1_id,term2_id,relationship_type_id,complete)
term_audit
term_id INTEGER
term_loadtime INTEGER
UNIQUE(term_id)
term_property
term_id INTEGER
term
term_synonym
association_property
id INTEGER
association_id INTEGER
relationship_type_id INTEGER
term_id INTEGER
association
id INTEGER
term_id INTEGER
gene_product_id INTEGER
is_not INTEGER
role_group INTEGER
assocdate INTEGER
source_db_id INTEGER
association_species_qualifier
id INTEGER
association_id INTEGER
species_id INTEGER
association_qualifier
id INTEGER
association_id INTEGER
term_id INTEGER
value VARCHAR(255)
db
id INTEGER
name VARCHAR(55)
fullname VARCHAR(255)
datatype VARCHAR(255)
generic_url VARCHAR(255)
url_syntax VARCHAR(255)
url_example VARCHAR(255)
url_prefix VARCHAR(255)
UNIQUE(name)
gene_product
id INTEGER
symbol VARCHAR(128)
dbxref_id INTEGER
species_id INTEGER
type_id INTEGER
full_name TEXT
UNIQUE(dbxref_id)
gene_product_homology
gene_product1_id INTEGER
gene_product2_id INTEGER
relationship_type_id INTEGER
gene_product_homolset
id INTEGER
gene_product_id INTEGER
homolset_id INTEGER
gene_product_synonym
gene_product_id INTEGER
product_synonym VARCHAR(255)
UNIQUE(gene_product_id,product_synonym)
gene_product_ancestor
gene_product_id INTEGER
ancestor_id INTEGER
gene_product_property
gene_product_id INTEGER
property_key VARCHAR(64)
property_val VARCHAR(255)
display_id VARCHAR(64)
description VARCHAR(255)
seq MEDIUMTEXT
seq_len INTEGER
md5checksum VARCHAR(32)
moltype VARCHAR(25)
timestamp INTEGER
UNIQUE(display_id,md5checksum)
seq_property
id INTEGER
seq_id INTEGER
property_key VARCHAR(64)
property_val VARCHAR(255)
UNIQUE(seq_id,property_key,property_val)
imgflip.com

Many issues

- Evolution is hard
- Testing is hard
- Versioning / collaboration is hard
- Performance is not great



the “solution” ….

- Team of 50 DBAs
- The “database” committee
- The “database” change process
- The “meta-database”
- Db replication
- Governance

Technical solutions
*… to solve **technical** issues*
*… introduced because of **technical** decisions*
*… with **no value** to the users*

*= **Accidental complexity***


How to Change Your Life
When You Feel STUCK
bigstockphoto.com/contributor=eelnosiva

Moving away from database-driven

- Persistence is an **implementation detail**

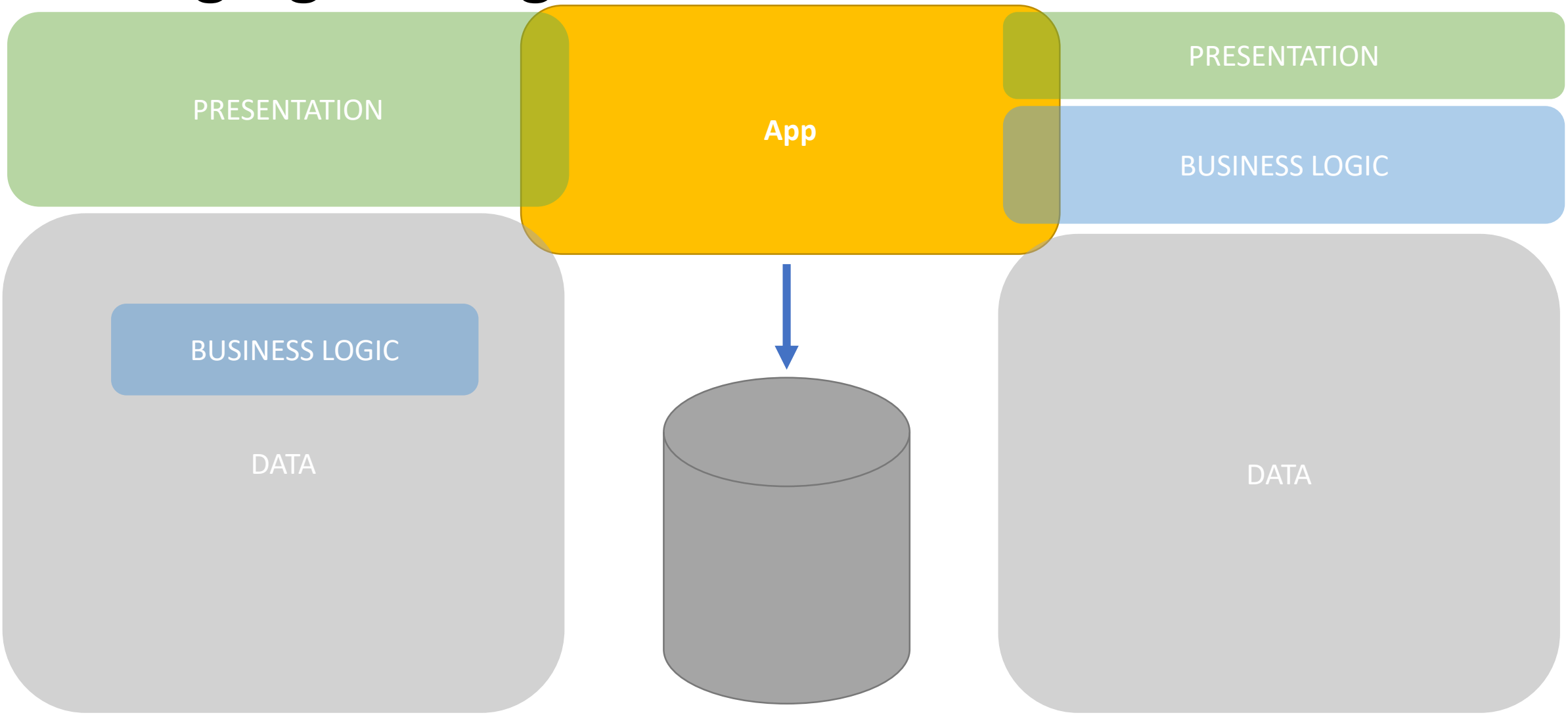
Relational DB, Document DB, Key-Value store, file …

Who cares ?

- Focus on **customer value**

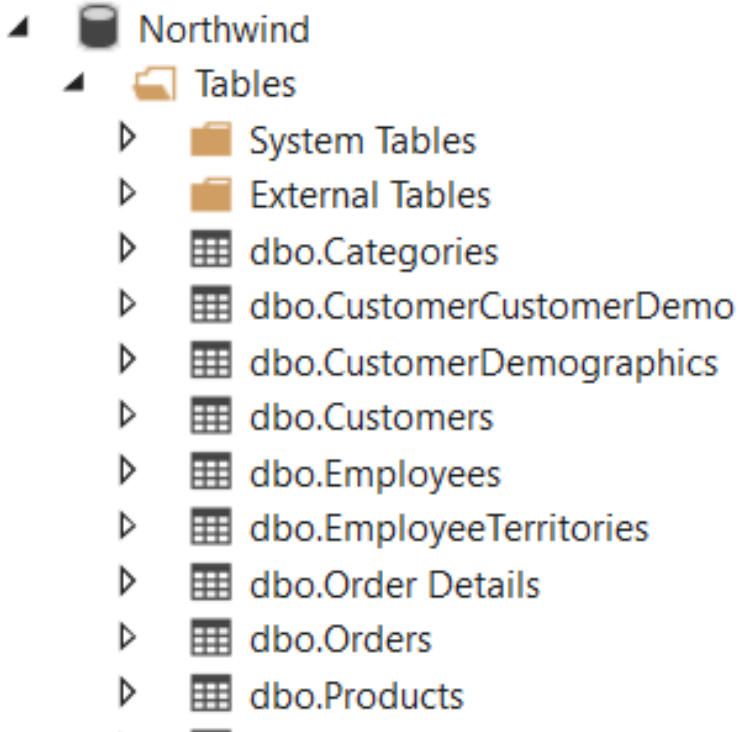
Business > *Tech*

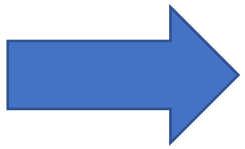
Bringing the logic back to the code

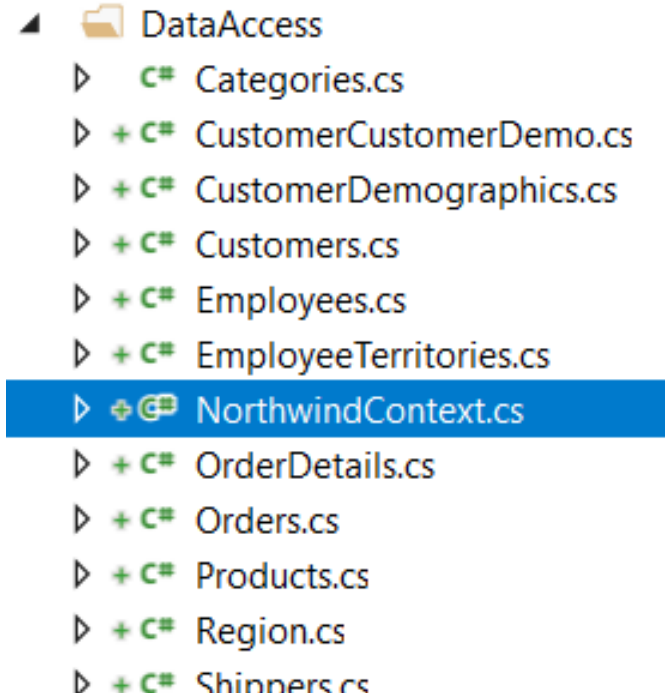


Bringing the logic back to the code

- **No more** Stored Procedures
- ORM (Entity Framework in that case)
- Data-access code + Entities* generated from database







* Not quite

```
using (var db = new NorthwindContext())
{
    var customer = await db.Customers
        .Where(e => e.CustomerId == customerId)
        .Include(e => e.Orders)
        .SingleOrDefaultAsync();

    if (customer.Orders.Count > 10)
    {
        customer.ContactTitle = "VIP";
    }

    await db.SaveChangesAsync();
```

```
using (var db = new NorthwindContext())
{
    var orders = await db.Orders
        .Where(o => o.OrderDetails.Count > 5)
        .Where(o => o.OrderDate.HasValue
                    && o.OrderDate.Value.Date == orderDate)
        .OrderBy(o => o.OrderId)
        .Include(o => o.Customer)
        .ToListAsync();

    return orders;
```

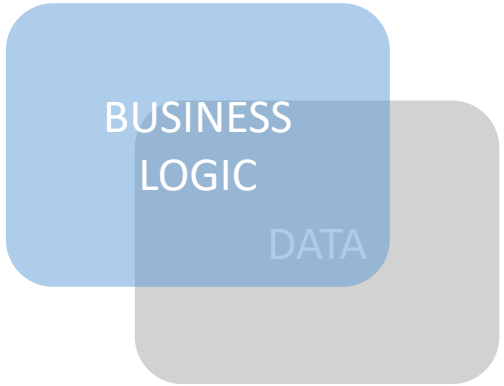
Status

That’s better !

- No more business logic in the DB
- Quite readable

Still a bit messy:

- Not testable
- Hard coupling



Better layering

Better layering

Services

BUSINESS LOGIC

Orchestrate data-flow for a given use-case

Repositories

+ Unit of Work

DATA

Abstract away details of how data is accessed

Repository

```
public interface IAnswerRepository
{
    Answer Get(int answerId);
    Answer GetByDossierId(int dossierId);
    void Add(Answer answer);
    void Update(Answer answer);
    void DeleteForDossier(int dossierId);
    Answer GetByAnswerPdfId(int answerPdfId);
    Answer GetByAccessToken(string accessToken);
    bool HasAnswerByDossierId(int dossierId);
}
```

A “Service”

```
public interface IAnswerService
{
    SaveAnswerResponse SaveAnswer(SaveAn
    GetAnswerResponse GetDemandeAnswer(Ge
    void AskSignature(AskAnswerSignaturel
}
```

```
public SaveAnswerResponse SaveAnswer(SaveAnswerRequest request)
{
    Validation the request

    // load entities from repo
    var answer = _answerRepository.Get(request.AnswerToSave.Id);

    // do stuff
    answer.AnswerAuthorizedSubType = request.AnswerToSave.AnswerAuthor
    // ... there would be lots of stuff here normally ...
    answer.AnswerLastModificationDate = DateTimeProvider.Instance.Now;

    // persist the changes
    _unitOfWork.Save();

    return new SaveAnswerResponse
    {
        AnswerId = answer.Id,
        AnswerStateValue = answer.AnswerState
    };
}
```

Status

Quite an improvement

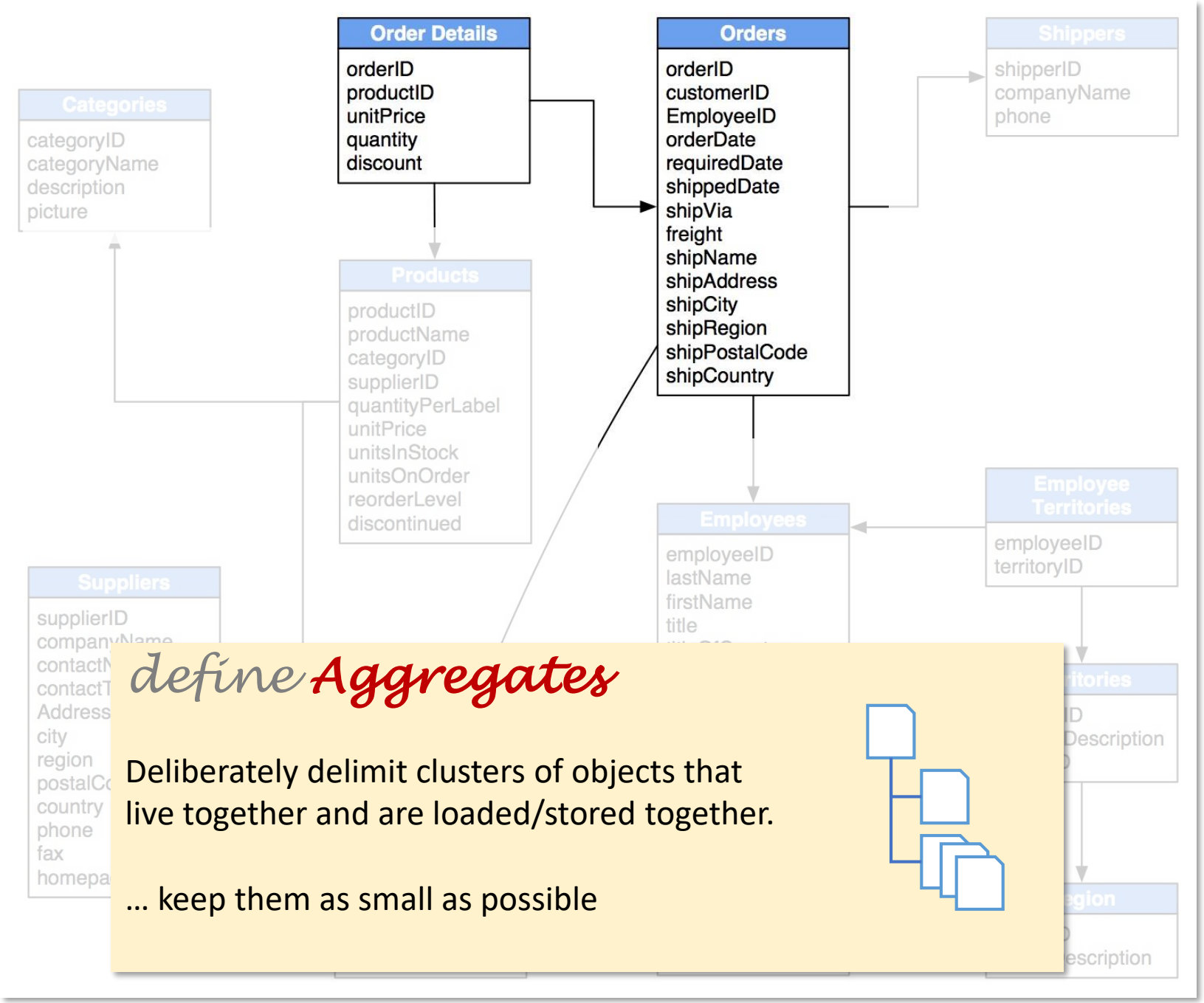
- Decoupled
- Testable
- Easy to know where functionality should live
- It worked fine initially

But … wait a minute …

Object graphs

*When loading an Order from DB … **what else should I load** ?*

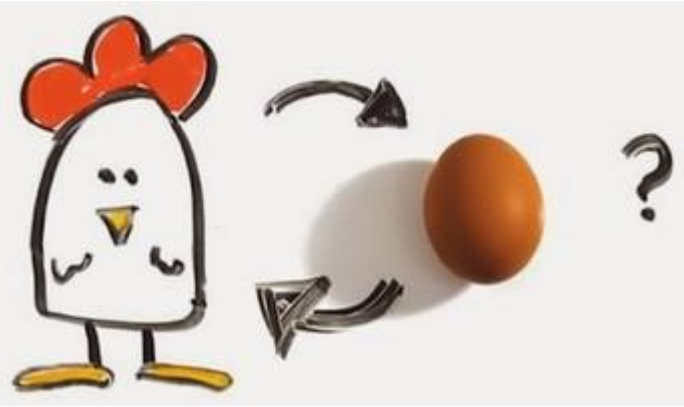
- All the relationships ?
- Some of them, and leave some unpopulated ?
- Some of them, and use lazy-loading ?



Object graphs

```
lic partial class Order

 public Order()...

 public int OrderId { get; set; }
 // ... snip ...

 public ICollection<OrderDetails> OrderDetails
```

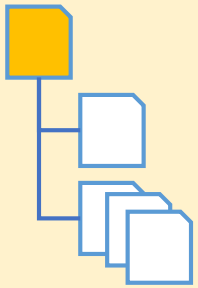
```
public partial class OrderDetails
{
    public int OrderId { get; set; }
    public int ProductId { get; set; }
    // ... snip ...

    public Order Order { get; set; }
}
```



identify the Aggregate Root

The unique entry point to the graph
Dependencies go only in one direction

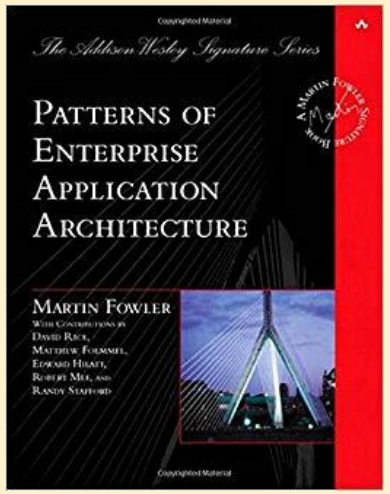
Repositories return Aggregates through their root



More smells …

```
public SaveAnswerResponse SaveAnswer(SaveAnswerRequest request)
{
    #region Validation

    var demande = BackOfficeRequestValidationHelper.ValidateRequestFo

    if (!IsAnswerEditable(demande))
    {
        throw new ForbiddenException(BusinessErrorMessages.DemandeNot
    }

    if (request.Answer
    #endreg  demande.Do
             demande.Do
    Answer   if (existi
    if (req  {
    {            var ex
                 demand
    ans      }
             demande.Is
             existingAn
             existingAn
```

```
public class DemandeAutorisation
{
    public DemandeAutorisation()...

    Populated at Creation

    /// <summary>
    /// Date et heure de la dernière modification.
    /// </summary>
    public DateTime LastModificationDate { get; se

    /// <summary>
    une de la transmission.

    e? TransmissionDate { get; set;

    Autorisation Dossier { get; set;

    s à la transaction (step 1)

    PartiesTransactionExportateur {
    PartiesTransactionImportateur {
```



Transaction Script

Anemic Domain Model

Encapsulation

- Do not expose setters
- Do expose only the Aggregate Root
- Do enforce invariants (guard clauses)
- Do mutations only through methods

*Make **invalid** states **impossible** to represent*

Application Service

```
public async Task<BusinessActionResult> AddComment(int messageId, string comment, int
{
    if (comment == null) throw new ArgumentNullException(nameof(comment));
    var message = await _messageRepository.GetById(messageId);
    if (message == null)
        throw new MessageNotFoundException($"Impossible de trouver un message avec l
    var utcNow = DateTimeProvider.Instance.UtcNow;
    message.AddComment(comment, utcNow, requesterId);
    await _adaUnitOfWork.SaveAsync();
    return BusinessActionResult.Success("Le commentaire a bien été ajouté. ");
}
```

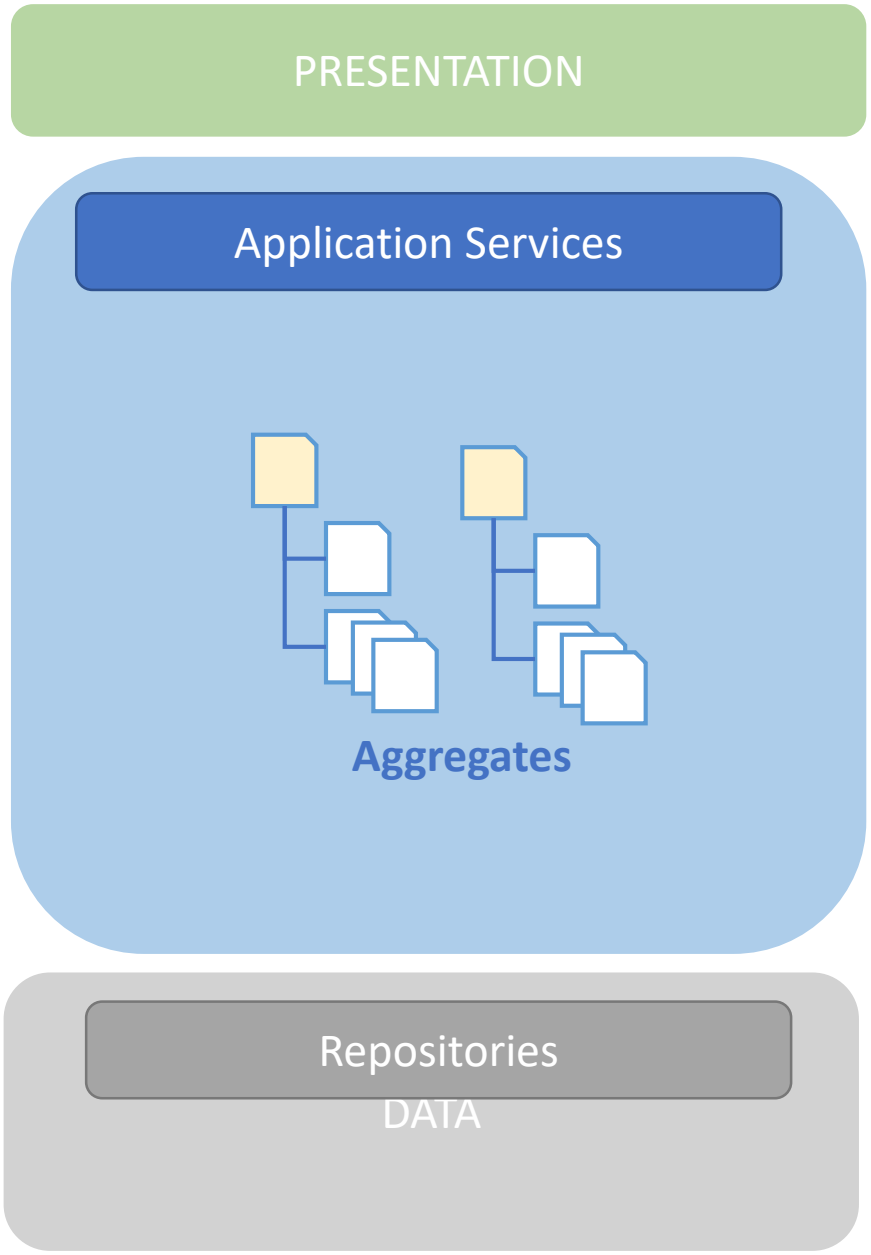
Entity mutation

```
public void RescopeTo(MessageScope scope, DateTime changeDate)
{
    if (scope == null) throw new ArgumentNullException(nameof(scope));
    if (this.Status != MessageStatus.InProgress)
    {
        throw new MessageRescopeException($"Change scope can only affect {MessageStatus
    }

    this.MarkAs(MessageStatus.New, changeDate, $"Ce message (scope précédent : {this.Sc
    State.Scope = scope.Value;
}
```

Status

Even better

- Decoupled
- Testable
- Easy to know where functionality should live
- Explicit models and methods
- Clean encapsulation



What about queries ?

We defined

- Small aggregates
- Repositories targeting only Aggregate Roots
- Transactional consistency
- Repositories hiding data-access

For views / reports we need

- JOINs across many tables
- Small queries on some tables
- No transactions
- Access to raw data / perf

Different needs require different tools

Separating Reads and Writes

- Commands vs Queries

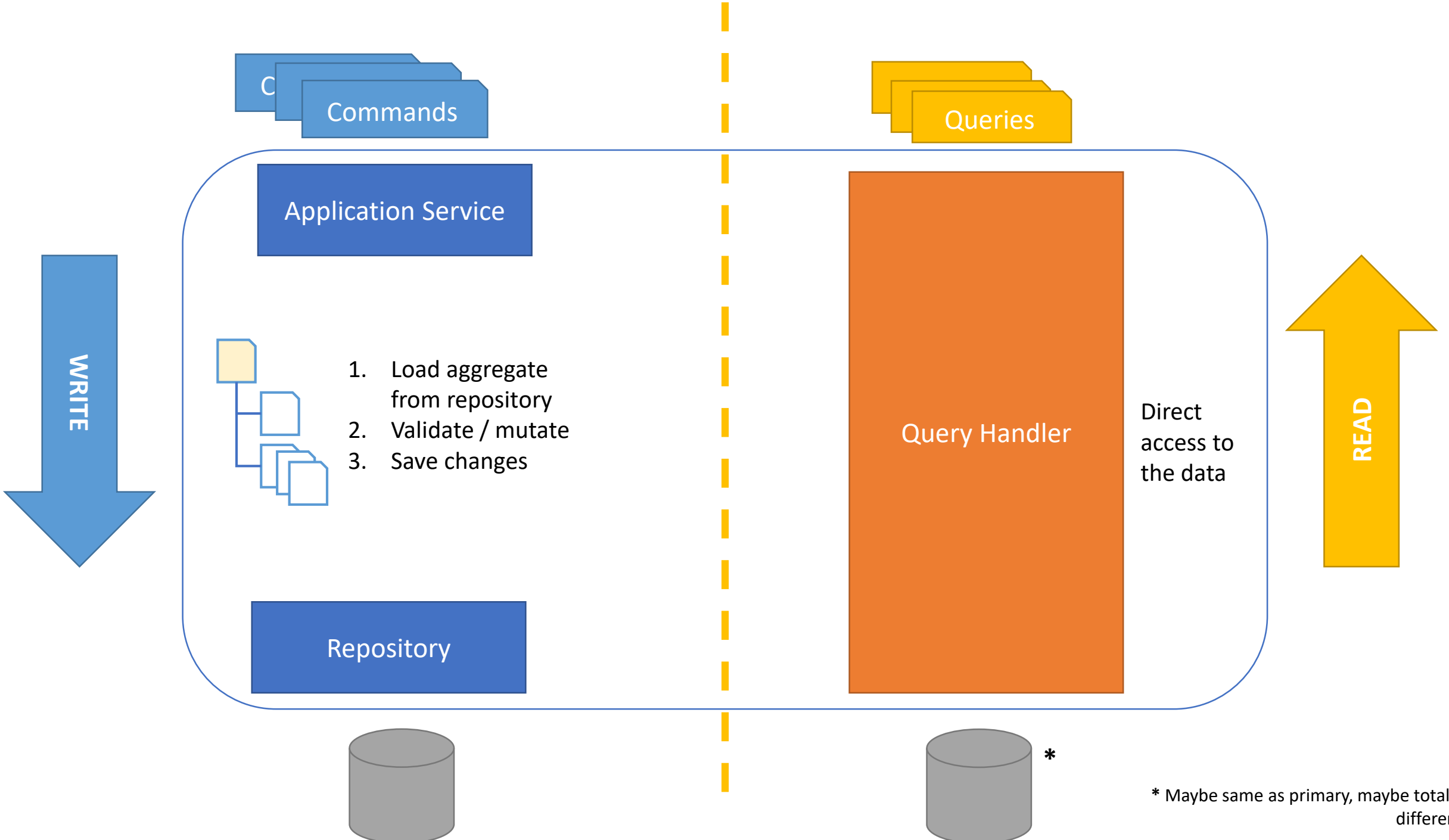
- Write Model vs Read Model

Aggregates

View-specific projections
As big or small as needed

CQS (**C**ommand **Q**uery **S**eparation)

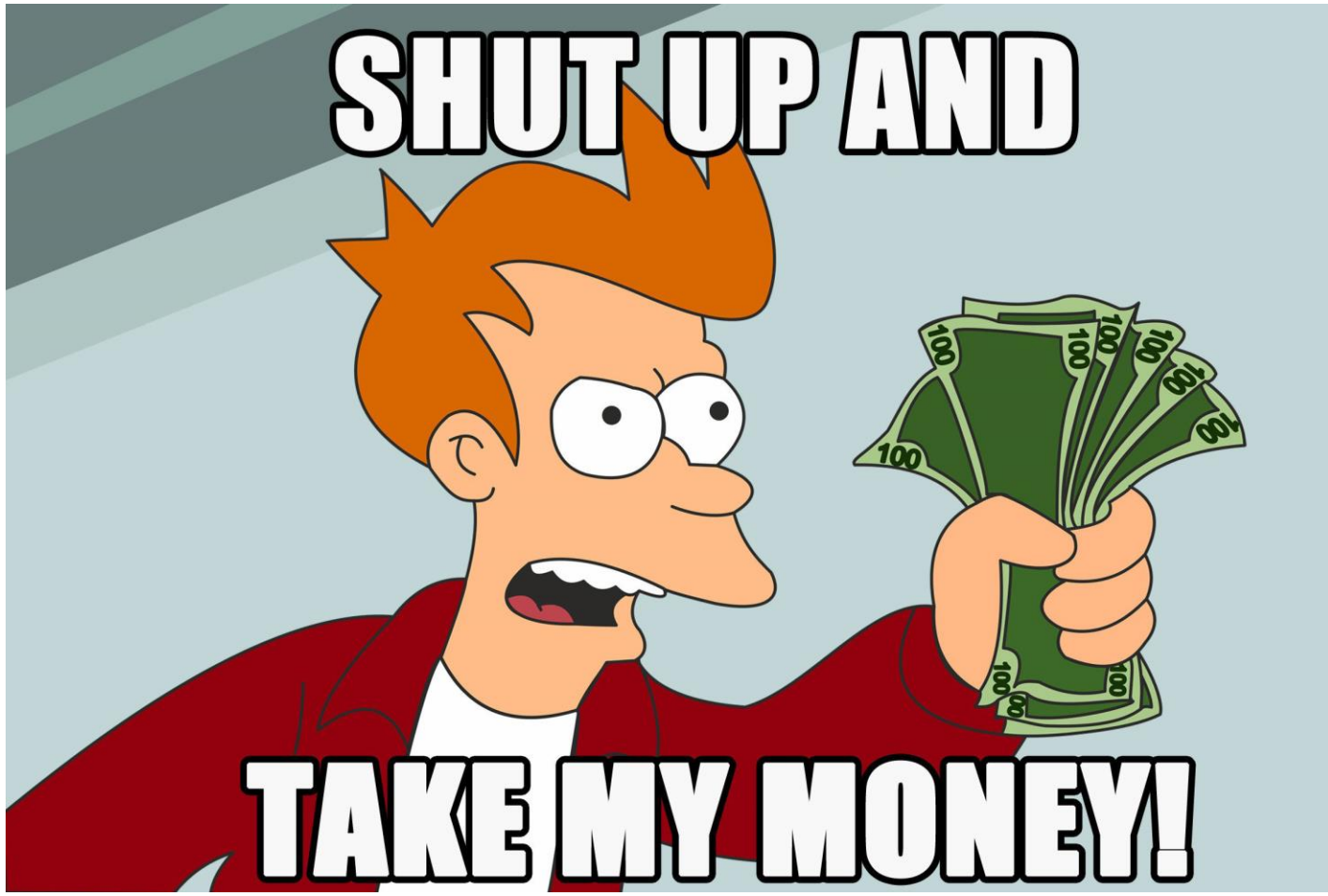
CQRS (**C**ommand **Q**uery **R**esponsibility **S**egregation)


Commands
Queries
Application Service
WRITE
1. Load aggregate from repository
2. Validate / mutate
3. Save changes
Repository
Query Handler
Direct access to the data
READ
*
* Maybe same as primary, maybe totally different

Status

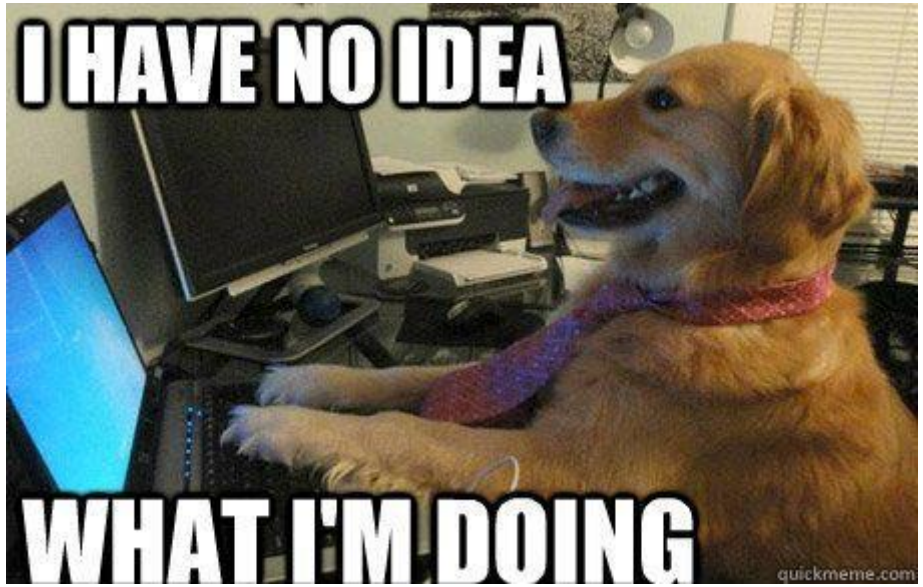
Good enough for now ☺

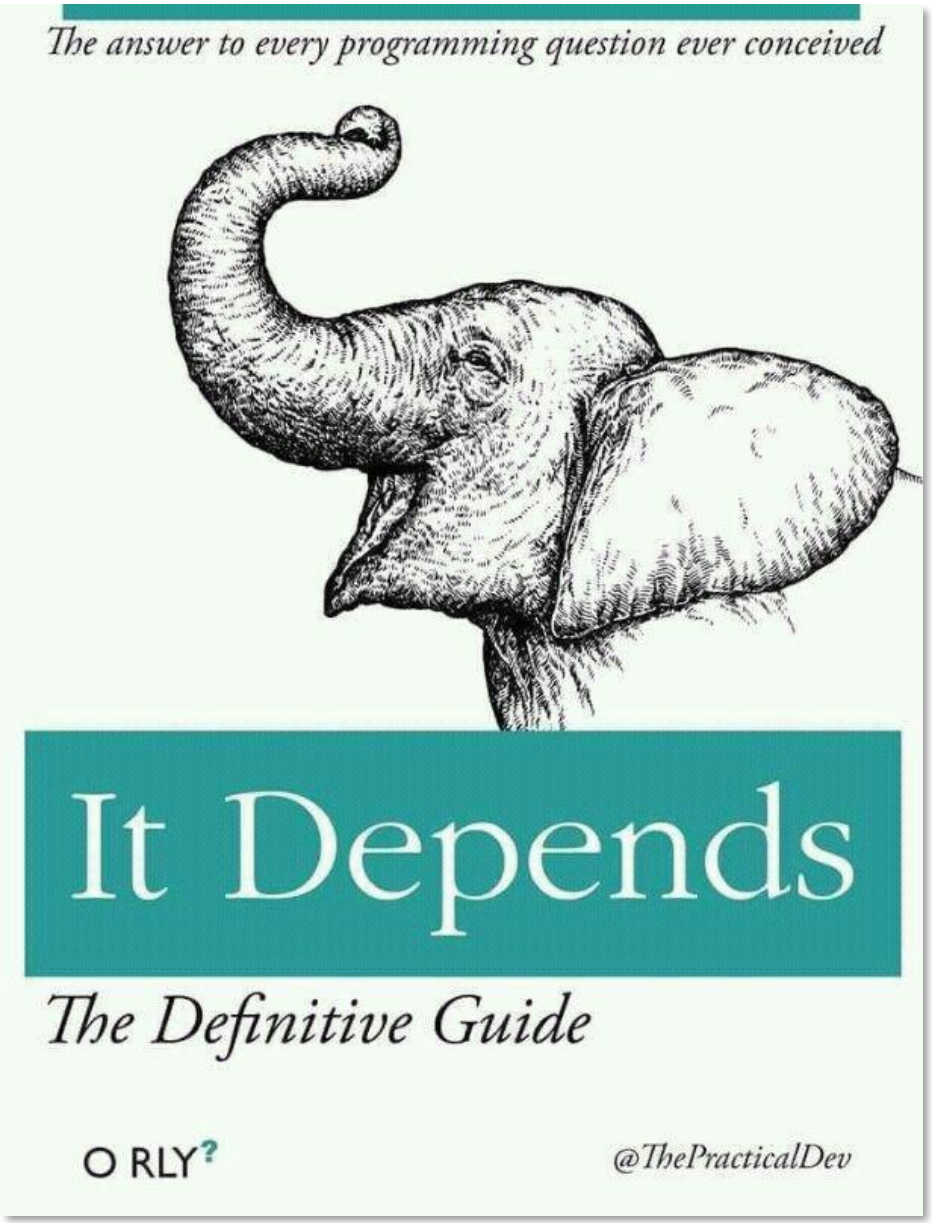
- Decoupled
- Testable
- Easy to know where functionality should live
- Explicit models and methods
- Clean encapsulation
- Correct writes
- Fast Reads



Conclusions

- Failing is learning
- The right tool for the right job
- Context is king

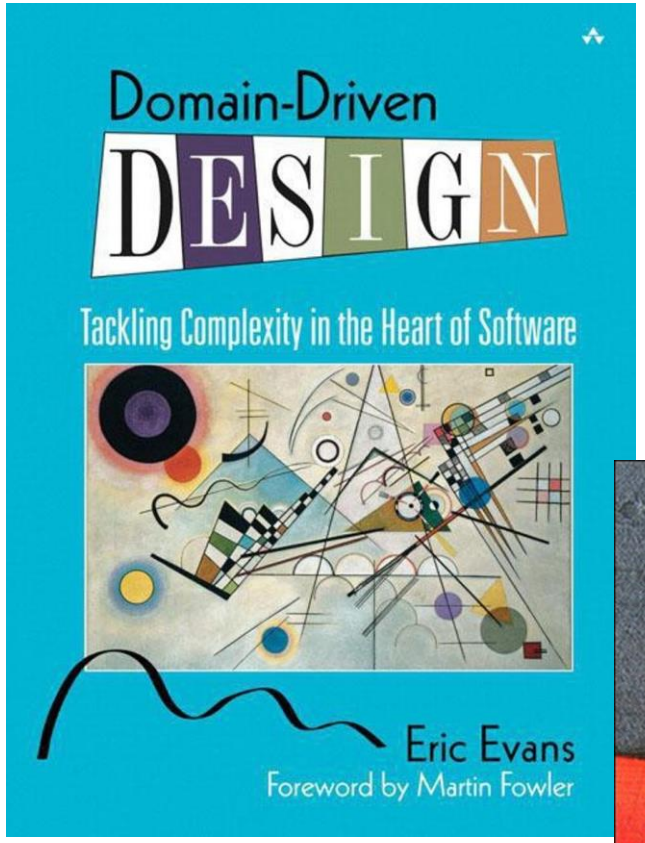


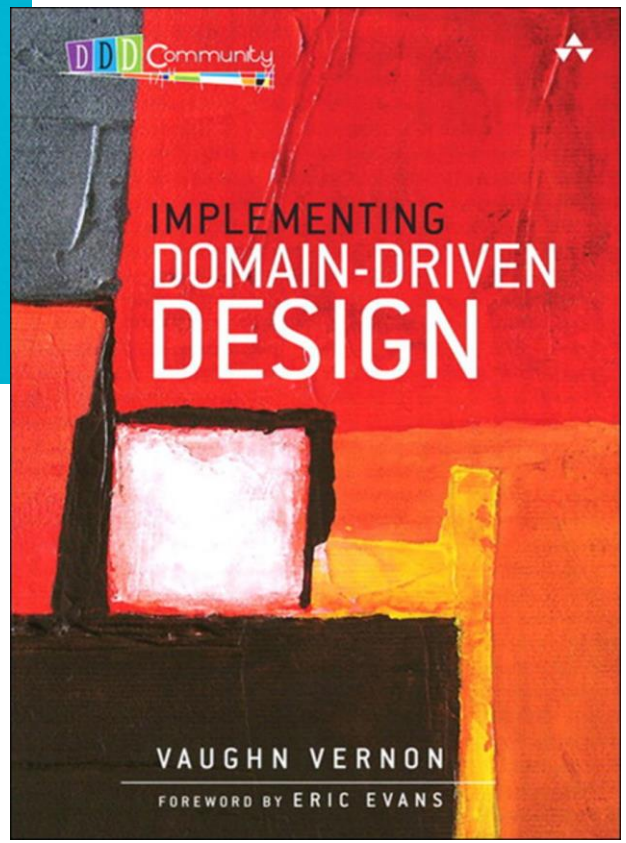


Going further

Some important concepts I left out :

- Value Objects
- Bounded Contexts
- Ubiquitous Language
- Domain Events
- Context Mapping
- …





Thanks for listening !

Questions ?