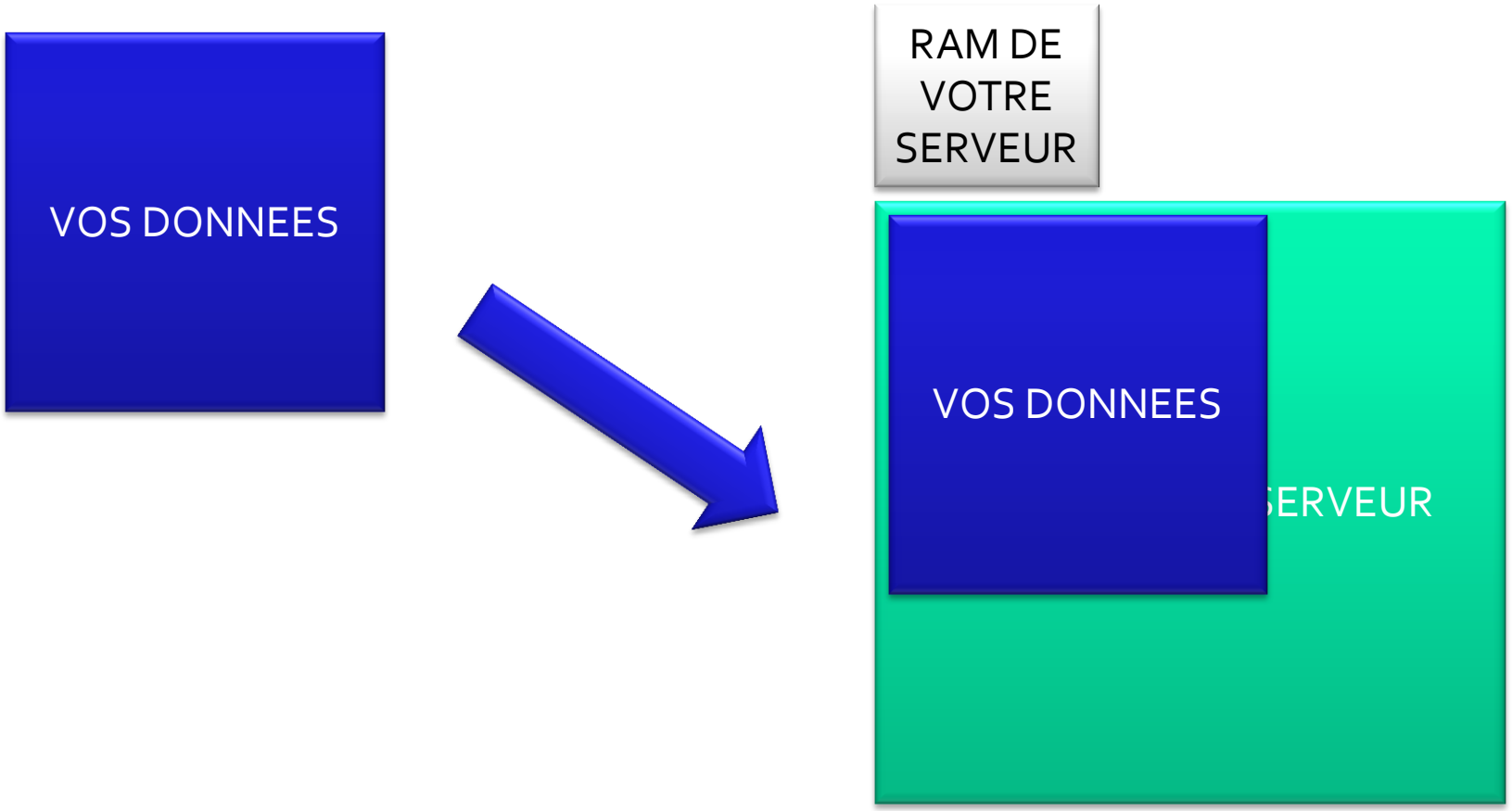
Prévalence : des solutions pour une architecture CQRS

JP Gouigoux / Guillaume Collic

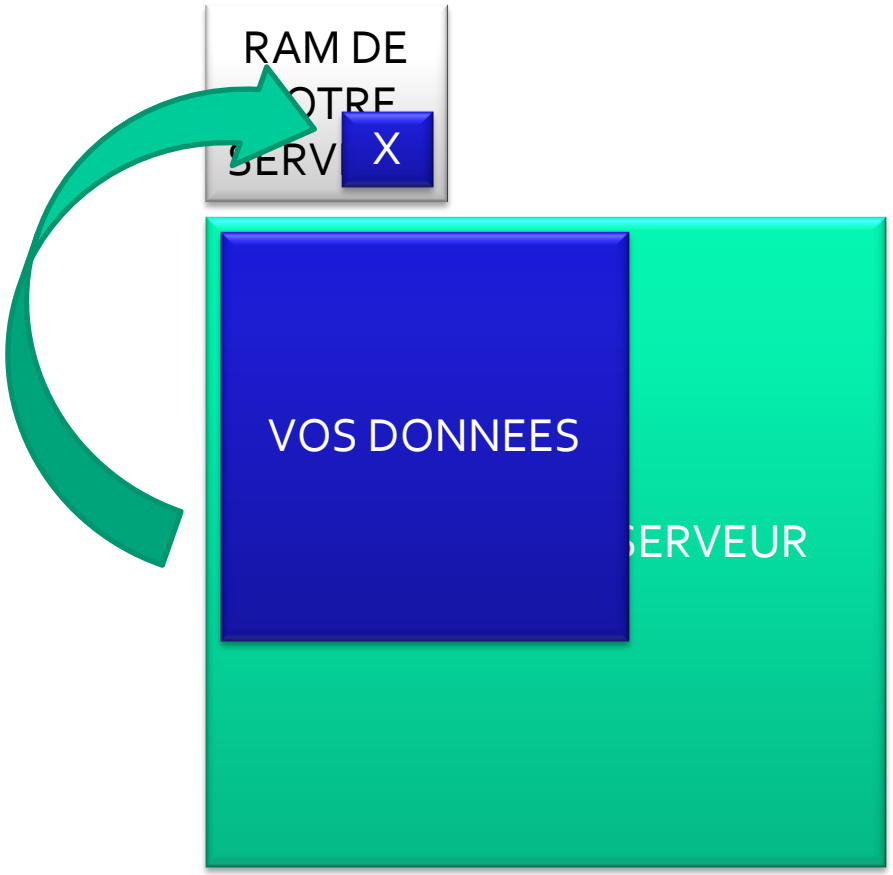
7 Octobre 2010

Il y a 20 ans…



Les conséquences

Besoin de monter efficacement de la donnée nécessaire au traitement depuis les disques durs vers la RAM

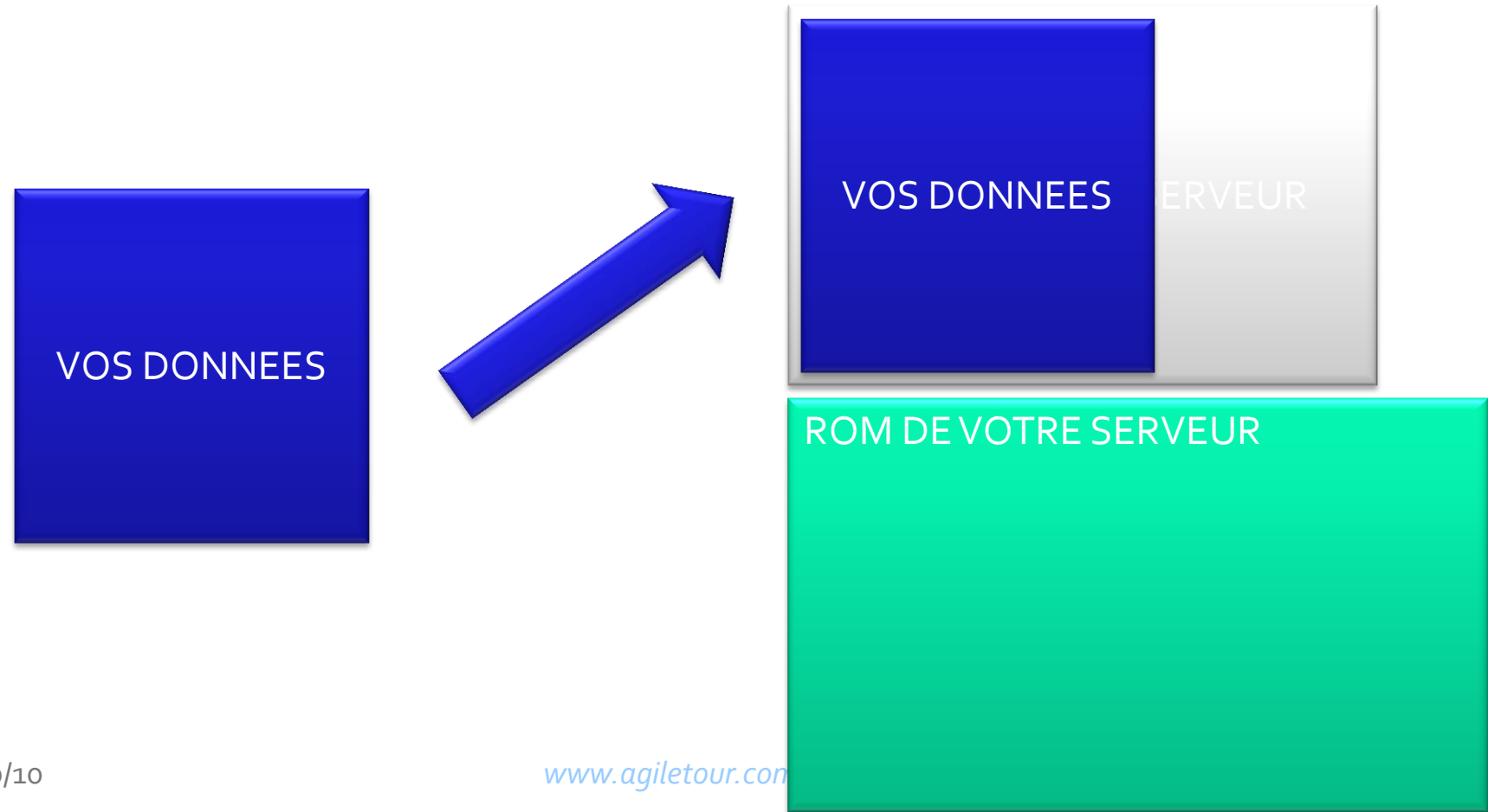


Pour réaliser ceci

- Langage de requête de données SQL
- Moteurs SGBD-R
- Gestion avancée de caches, d'index, de b-tree
- Des millions de lignes de code
- Des milliards de dollars de chiffre d'affaire

Tout ça pour monter des données de la ROM à la RAM…

Aujourd'hui



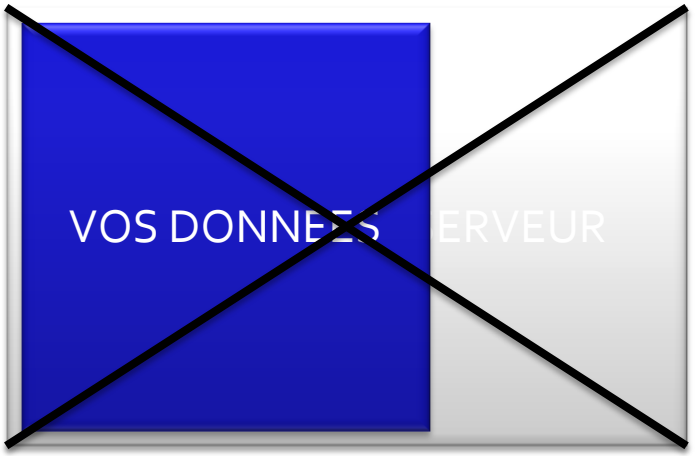
Les conséquences

- Plus besoin de base de données
- Plus d'optimisation SQL complexe
- Performances extraordinaires
- Plus de mapping O/R !!!
- En fait, plus de gestion de la persistence
- Le modèle, c'est le métier

Ca commence à sentir le DDD…

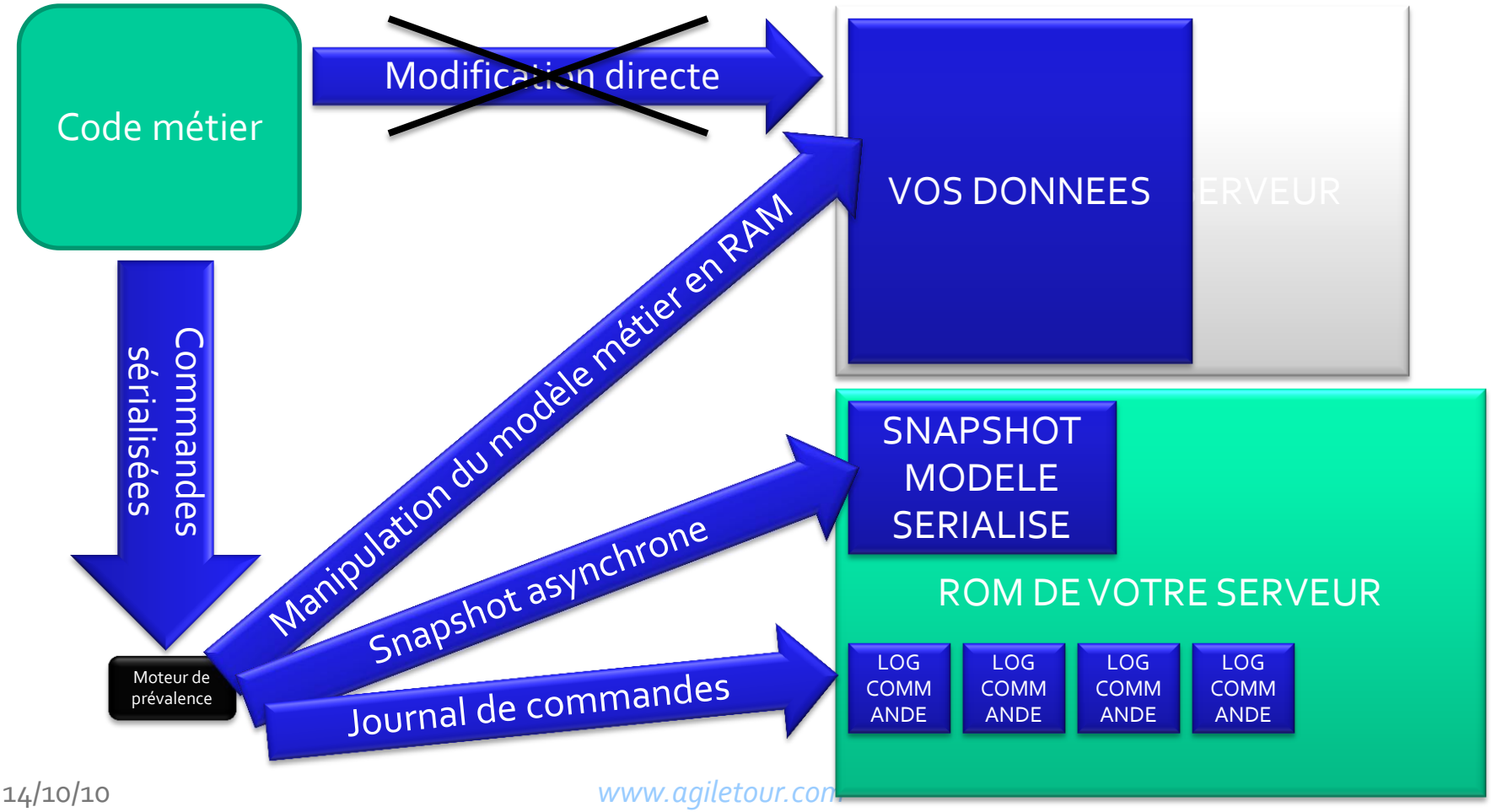
Oui, mais…

Panne d'électricité, arrêt du serveur pour maintenance



ROM DE VOTRE SERVEUR

Bienvenue à la prévalence



Ca ne vous rappelle rien ?

- Commandes sérialisées
- Modèle manipulé uniquement par des commandes
- Commandes naturellement transactionnelles
- Eviter la sur-qualité de consistance de donnée au profit de la rapidité (sans ACID)

Des notions de base de CQRS

Exemple de Linq

- SQL :

```
List<Personne> PersonnesDebutAnnuaire = new List<Personne>();
SqlCommand Commande = new SqlCommand(
    "SELECT nom, personne FROM PERSONNE WHERE nom LIKE 'A%' ORDER BY nom",
    ConnexionSQL);
using (SqlDataReader Lecteur = Commande.ExecuteReader(CommandBehavior.CloseConnection))
    while (Lecteur.Read())
        PersonnesDebutAnnuaire.Add(
            new Personne(
                Lecteur.GetString(0),
                Lecteur.GetString(1)));
```

- Linq en C# :

```
var PersonnesDebutAnnuaire = from personne in Donnees.ListePersonnes
                             where personne.Nom.StartsWith("A")
                             orderby personne.Prenom
                             select personne;
```

agile tour 2010

Plus loin avec Linq

```
decimal SommeMetier = Reservations
    .Where(Resa => Resa.UID.Split(new char[] { 'a' }, StringSplitOptions.RemoveEmptyEntries).Count() > 3)
    .Bareme(Resa => Resa.Description.Length, Resa => Resa.DateDebut)
    .Sum(Valeur => Valeur);
```

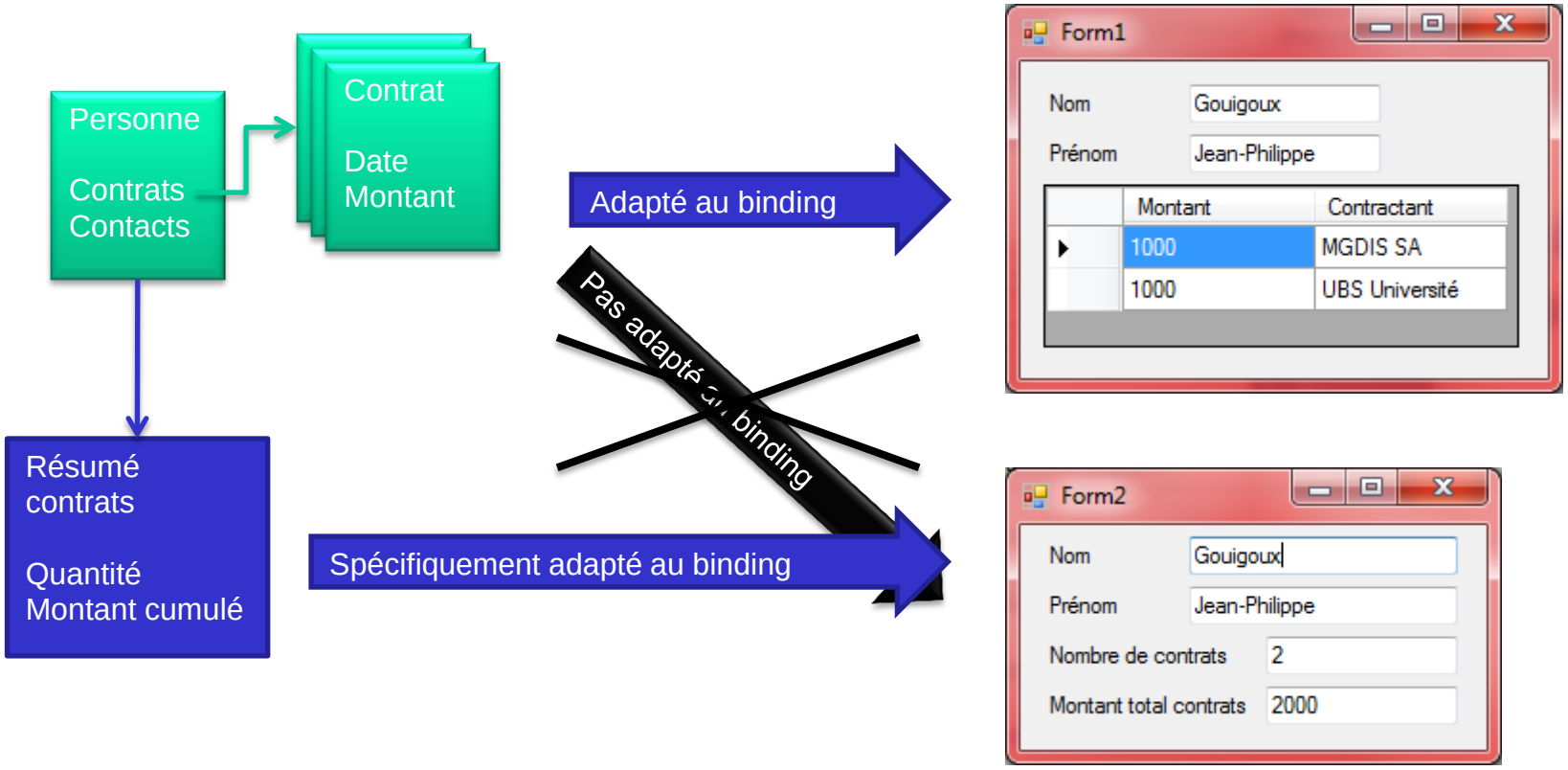
```
public static IEnumerable<decimal> Bareme<T>(this IEnumerable<T> Liste, Func<T, decimal> BaseCalcul, Func<T, DateTime> DateApplication)
{
    foreach (T Atome in Liste)
    {
        int[] Coefficients = new int[] { 0, 4, 9, 12 };
        DateTime DateCoefficients = DateApplication(Atome);
        if (DateCoefficients.DayOfWeek == DayOfWeek.Wednesday) Coefficients = new int[] { 0, 5, 10, 15 };
        else if (DateCoefficients.DayOfYear > 210) Coefficients = new int[] { 0, 3, 8, 20 };

        decimal Resultat = 0;
        decimal ValeurReference = BaseCalcul(Atome);
        if (ValeurReference < Coefficients[0]) Resultat = -1;
        else if (ValeurReference < Coefficients[1]) Resultat = 0;
        else if (ValeurReference < Coefficients[2]) Resultat = 1;
        else if (ValeurReference >= Coefficients[3]) Resultat = 2;
        else Resultat = 3;
        yield return Resultat;
    }
}
```

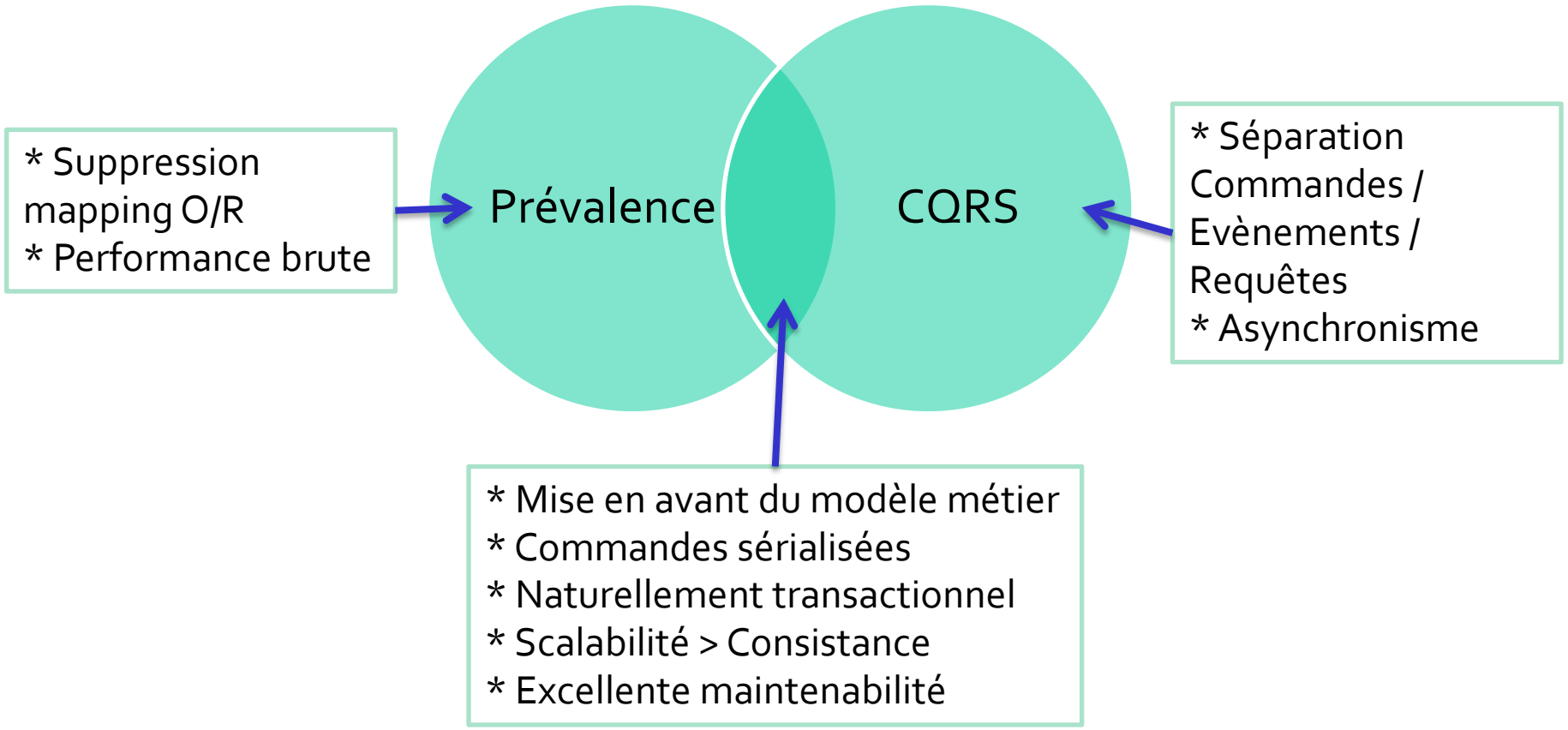
Performance ? Maintenabilité ?

- Prévalence + Linq :
  - 3 minutes pour coder la requête métier
  - 1,57 secondes pour calcul sur 250 000 objets
- SGBD + SQL :
  - Pas réussi à écrire une requête performante au bout de 30 minutes
  - Un spécialiste pourra le faire, mais pas en 3 minutes (je prends le pari)

Exemple de modèle dénormalisé



Prévalence ≠ CQRS



En pratique

- Framework Bamboo .NET
- 175 Ko seulement
- Plus de détails sur mon blog http://blogs.dotnet-france.com/jeanphilippeg
- Ou « prévalence + linq » sur Google
- Rien de mieux qu'un petit Dojo de démo (attention : application autonome)