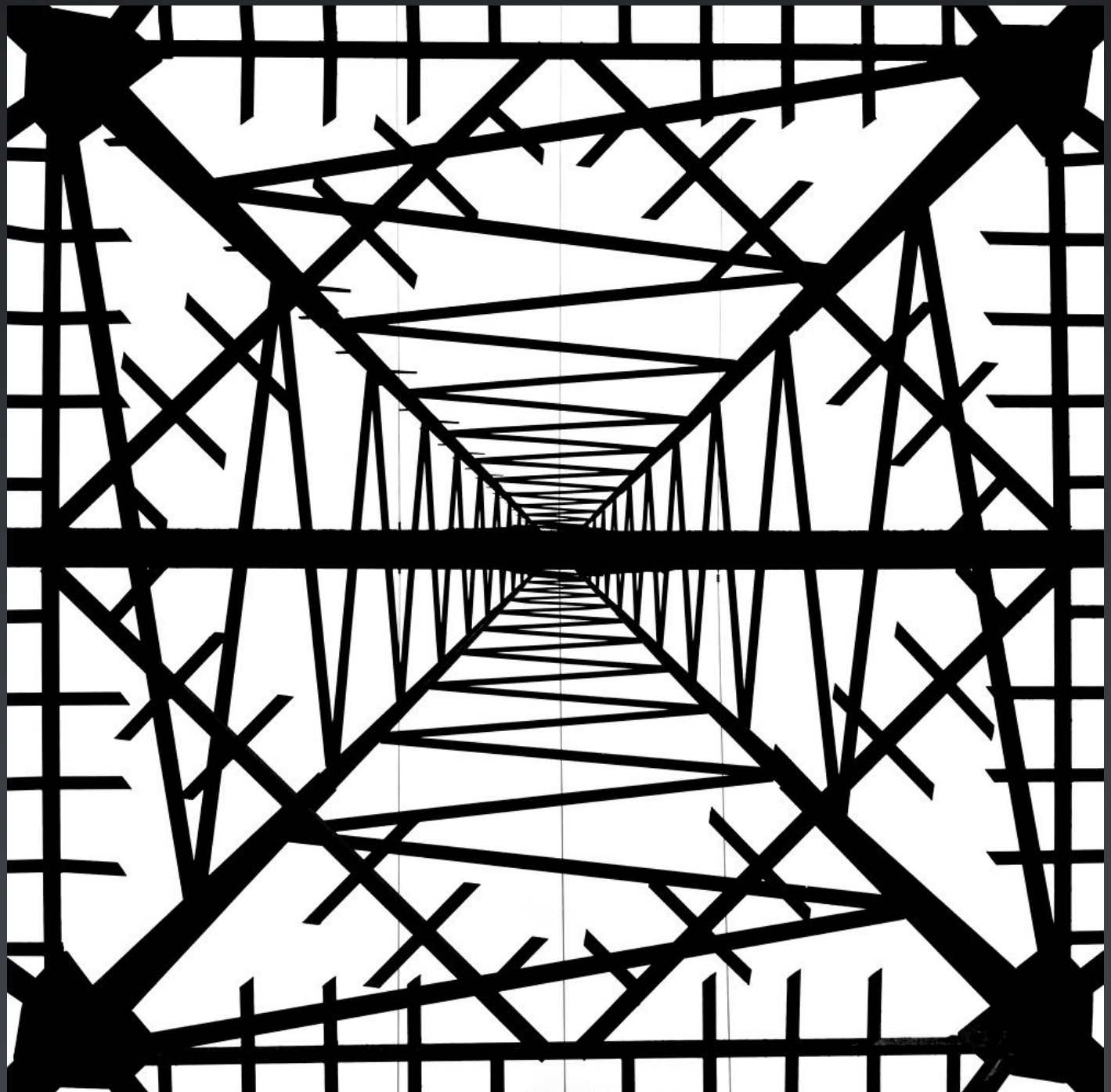
CQRS / EVENT SOURCING PAR L'EXEMPLE.

Julien Vinber - Meetup 22 novembre 2023 AFUP Montpellier



JULIEN VINBER

- Junior depuis plus de 20 ans
- Lead dev / Architect pour Maline Immobilier
- Un peu, mais plus trop organiser les Meetups
- Adorateur sataniste de Merise et CURL
- LinkedIn : @julienVinber

ATTENTION

Il ne faut pas prendre tout ce que je vais dire pour vérité absolue et immuable.

Ce que je vais présenter est un point de vue. À vous d'en retirer ce dont vous avez envie.

PETITS SONDAGES POUR COMMENCER

PETITS SONDAGES POUR COMMENCER

- Qui a déjà eu des problèmes de performance avec une Base de données ?

PETITS SONDAGES POUR COMMENCER

- Qui a déjà eu des problèmes de performance avec une Base de données ?
- Qui a déjà eu des problèmes de conception ?

PETITS SONDAGES POUR COMMENCER

- Qui a déjà eu des problèmes de performance avec une Base de données ?
- Qui a déjà eu des problèmes de conception ?
- Qui a déjà eu des problèmes de consistance ?

PETITS SONDAGES POUR COMMENCER

- Qui a déjà eu des problèmes de performance avec une Base de données ?
- Qui a déjà eu des problèmes de conception ?
- Qui a déjà eu des problèmes de consistance ?
- Qui a déjà été bloqué par la structure de la base ?

PETITS SONDAGES POUR COMMENCER

- Qui a déjà eu des problèmes de performance avec une Base de données ?
- Qui a déjà eu des problèmes de conception ?
- Qui a déjà eu des problèmes de consistance ?
- Qui a déjà été bloqué par la structure de la base ?
- Qui ne comprend pas pourquoi je pose ces questions ?

Je m'avance en prédisant qu'à part des étudiants, personne ne devrait avoir levé la main à la dernière question.

UN PEU DE CONTEXTE

Depuis toujours, on utilise fondamentalement 3 piliers pour la conception des bases de données :

UN PEU DE CONTEXTE

Depuis toujours, on utilise fondamentalement 3 piliers pour la conception des bases de données :

- Merise

UN PEU DE CONTEXTE

Depuis toujours, on utilise fondamentalement 3 piliers pour la conception des bases de données :

Merise SGBDR / SQL

UN PEU DE CONTEXTE

Depuis toujours, on utilise fondamentalement 3 piliers pour la conception des bases de données :

Merise SGBDR / SQL CRUD

UN PEU DE CONTEXTE

Depuis toujours, on utilise fondamentalement 3 piliers pour la conception des bases de données :

Merise SGBDR / SQL CRUD

POURQUOI ?

POURQUOI ?

👍 Cela marche.

POURQUOI ?

- 👍 Cela marche.
- 👍 C'est simple.

POURQUOI ?

- 👍 Cela marche.
- 👍 C'est simple.
- 👍 On peut faire à peu près tout avec.

POURQUOI ?

👍 Cela marche.
👍 C'est simple.
👍 On peut faire à peu près tout avec.
👍 C'est ce que l'on a appris et que tout le monde applique depuis 50 ans.

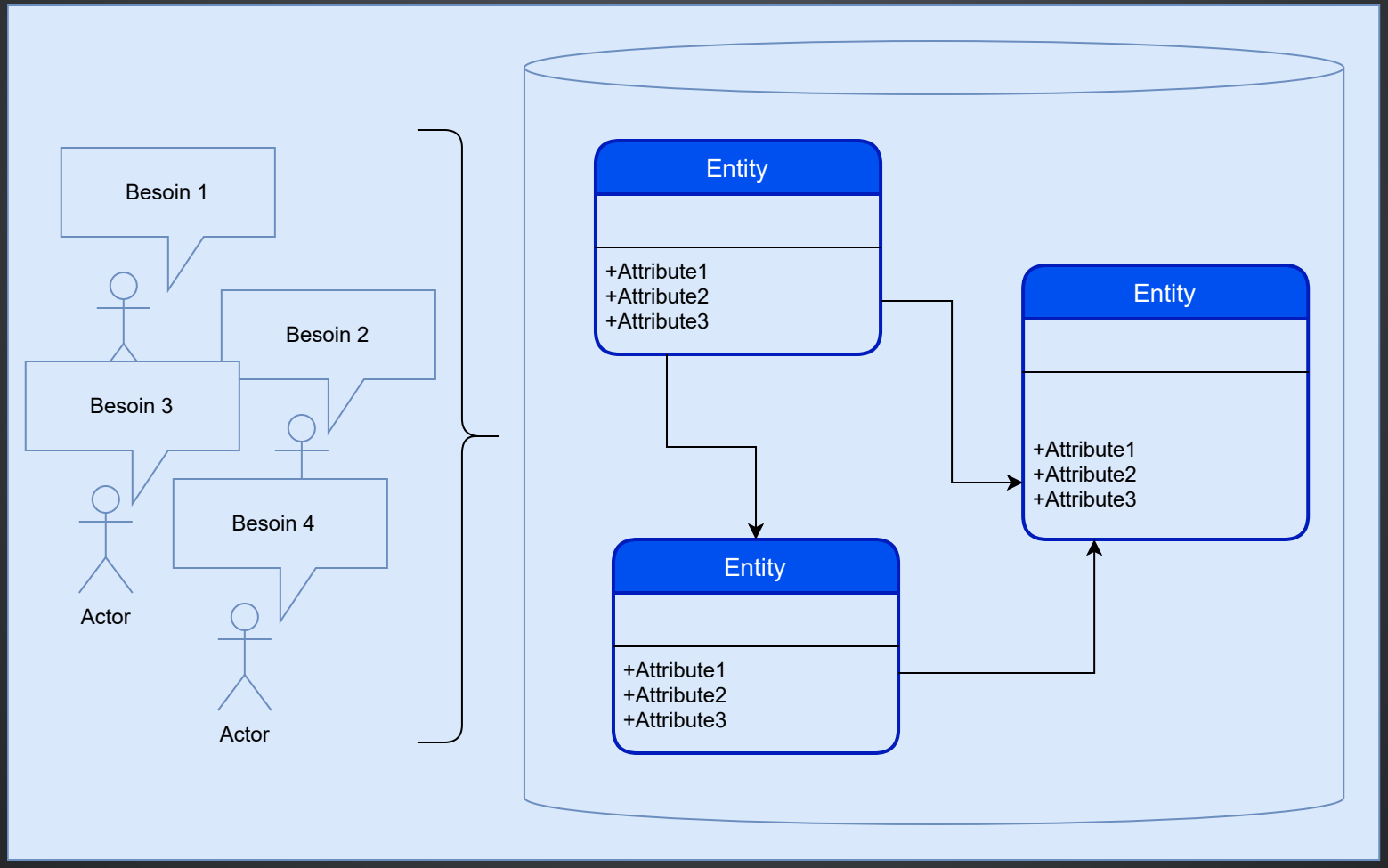
GRAND PRINCIPE

GRAND PRINCIPE

1) LA CONCEPTION

GRAND PRINCIPE

1) LA CONCEPTION



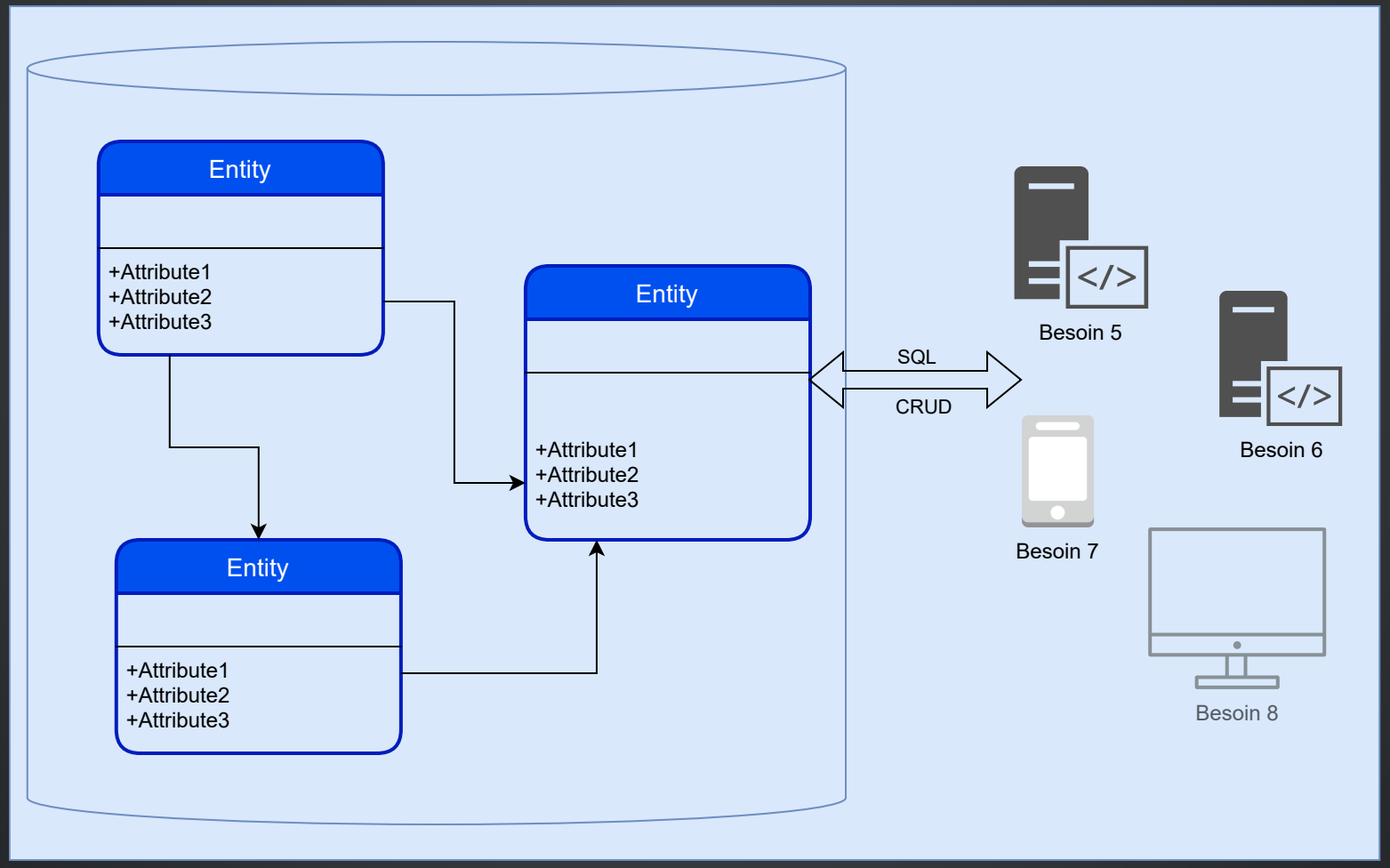
GRAND PRINCIPE

GRAND PRINCIPE

2)L'EXPLOITATION

GRAND PRINCIPE

2)L'EXPLOITATION



PROBLÈMES :

PROBLÈMES :

👎 Quand on peut tout faire, on est souvent mauvais partout.

PROBLÈMES :

👎 Quand on peut tout faire, on est souvent mauvais partout.
👎 La complexité de l'écriture des règles de gestion est exponentielle avec la complexité du modèle.

PROBLÈMES :

👎 Quand on peut tout faire, on est souvent mauvais partout.
👎 La complexité de l'écriture des règles de gestion est exponentielle avec la complexité du modèle.
→ Dans les faits, les règles sont incomplètes, les utilisateurs pouvant tout modifier vont créer de l'inconsistance.

PROBLÈMES :

👎 Quand on peut tout faire, on est souvent mauvais partout.
👎 La complexité de l'écriture des règles de gestion est exponentielle avec la complexité du modèle.
→ Dans les faits, les règles sont incomplètes, les utilisateurs pouvant tout modifier vont créer de l'inconsistance.
👎 Plus c'est gros, plus cela sera lent.

PROBLÈMES :

👎 Quand on peut tout faire, on est souvent mauvais partout.
👎 La complexité de l'écriture des règles de gestion est exponentielle avec la complexité du modèle.
→ Dans les faits, les règles sont incomplètes, les utilisateurs pouvant tout modifier vont créer de l'inconsistance.
👎 Plus c'est gros, plus cela sera lent.
👎 Modèle souvent rigide, et peu compatible avec la notion l'évolution permanente.

FACE À CE CONSTAT, JE VOUS PROPOSE UNE AUTRE FAÇON DE VOIR :

FACE À CE CONSTAT, JE VOUS PROPOSE UNE AUTRE FAÇON DE VOIR :

CQRS ET EVENT SOURCING

CQRS

Command and **Q**uery **R**esponsibility **S**egregation

https://martinfowler.com/bliki/CQRS.html

UNE AUTRE FAÇON DE VOIR CQRS

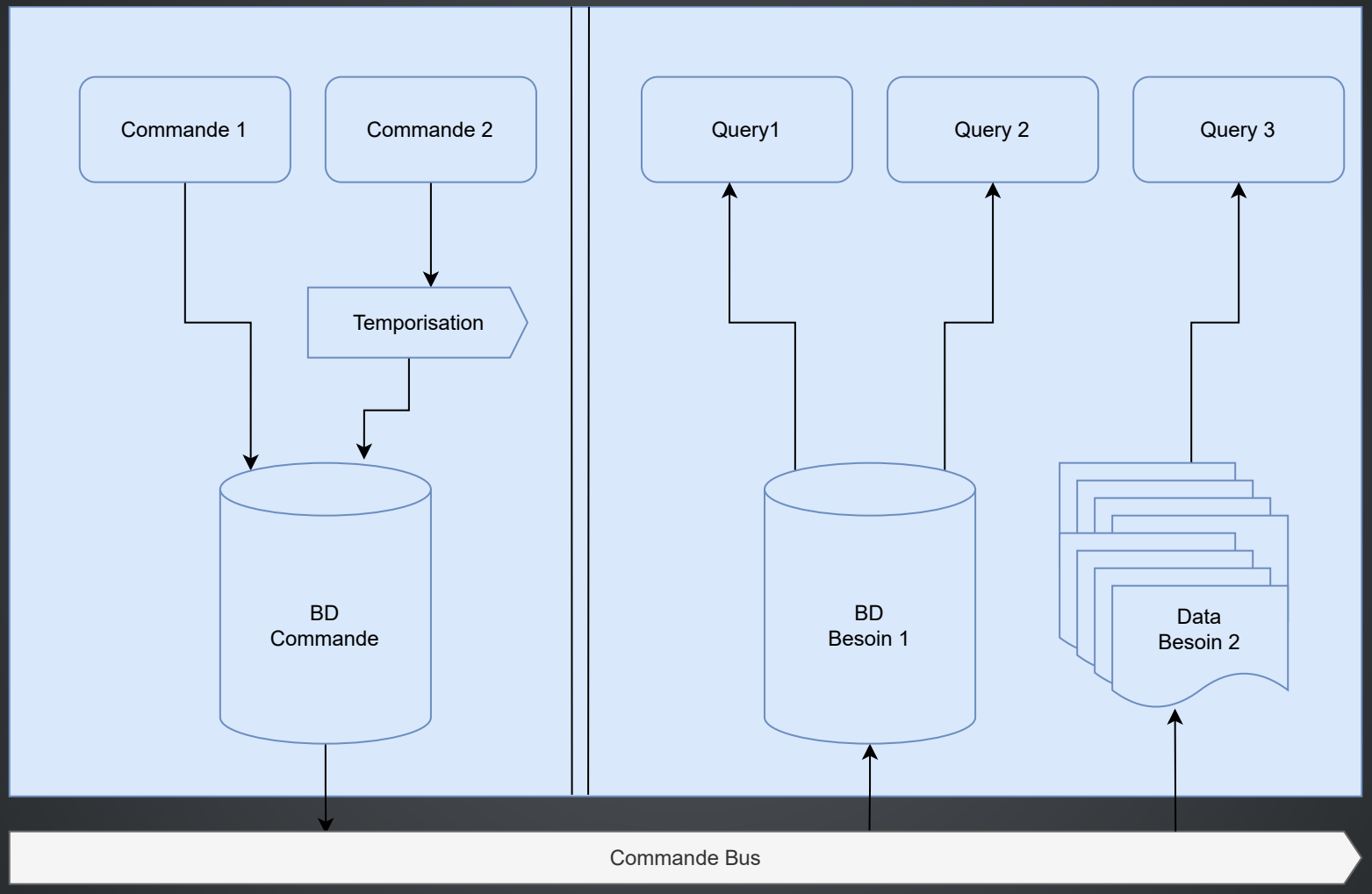
UNE AUTRE FAÇON DE VOIR CQRS

C'est le besoin qui pilote la technique.

UNE AUTRE FAÇON DE VOIR CQRS

C'est le besoin qui pilote la technique.

Il peut y avoir autant de réponses techniques que de besoins.


Commande 1
Commande 2
Temporisation
BD
Commande
Query1
Query 2
Query 3
BD
Besoin 1
Data
Besoin 2
Commande Bus

EVENT SOURCING

EVENT SOURCING

Capturez toutes les modifications apportées à l'état d'une application sous la forme d'une séquence d'événements.

https://martinfowler.com/eaaDev/EventSourcing.html

CONCRÈTEMENT :

LES "COMMAND"

CONCRÈTEMENT :

LES "COMMAND"

- 1 besoin = 1 commande

CONCRÈTEMENT :

LES "COMMAND"

- 1 besoin = 1 commande
- Toute commande DOIT être validée.

CONCRÈTEMENT :

LES "COMMAND"

- 1 besoin = 1 commande
- Toute commande DOIT être validée.
- 1 commande validée et complétée = 1 Event

CONCRÈTEMENT :

LES "COMMAND"

- 1 besoin = 1 commande
- Toute commande DOIT être validée.
- 1 commande validée et complétée = 1 Event
- Les évents sont sauvegardés ET envoyés pour traitement.

CONCRÈTEMENT :

PRÉPARATION DES DATA COTÉ "QUERY"

CONCRÈTEMENT :

PRÉPARATION DES DATA COTÉ "QUERY"

- Des "dispatcher" reçoivent l'ensemble des événements.

CONCRÈTEMENT :

PRÉPARATION DES DATA COTÉ "QUERY"

- Des "dispatcher" reçoivent l'ensemble des événements.
- En fonction du type d'évent, il appelle le bon "handler".

CONCRÈTEMENT :

PRÉPARATION DES DATA COTÉ "QUERY"

- Des "dispatcher" reçoivent l'ensemble des événements.
- En fonction du type d'évent, il appelle le bon "handler".
- Les "handler" mettent à jour leurs bases de données.

CONCRETEMENT :

LES "QUERY"

CONCRETEMENT :

LES "QUERY"

- Accès aux data normales.

CONCRETEMENT :

LES "QUERY"

- Accès aux data normales.
- Au détail presque les bases sont plus adaptées au besoin...

UN "EVENT"

Un event est la pour répondre à plusieurs questions :

UN "EVENT"

Un event est la pour répondre à plusieurs questions :

- Qui ?

UN "EVENT"

Un event est la pour répondre à plusieurs questions :

- Qui ?
- Quand ?

UN "EVENT"

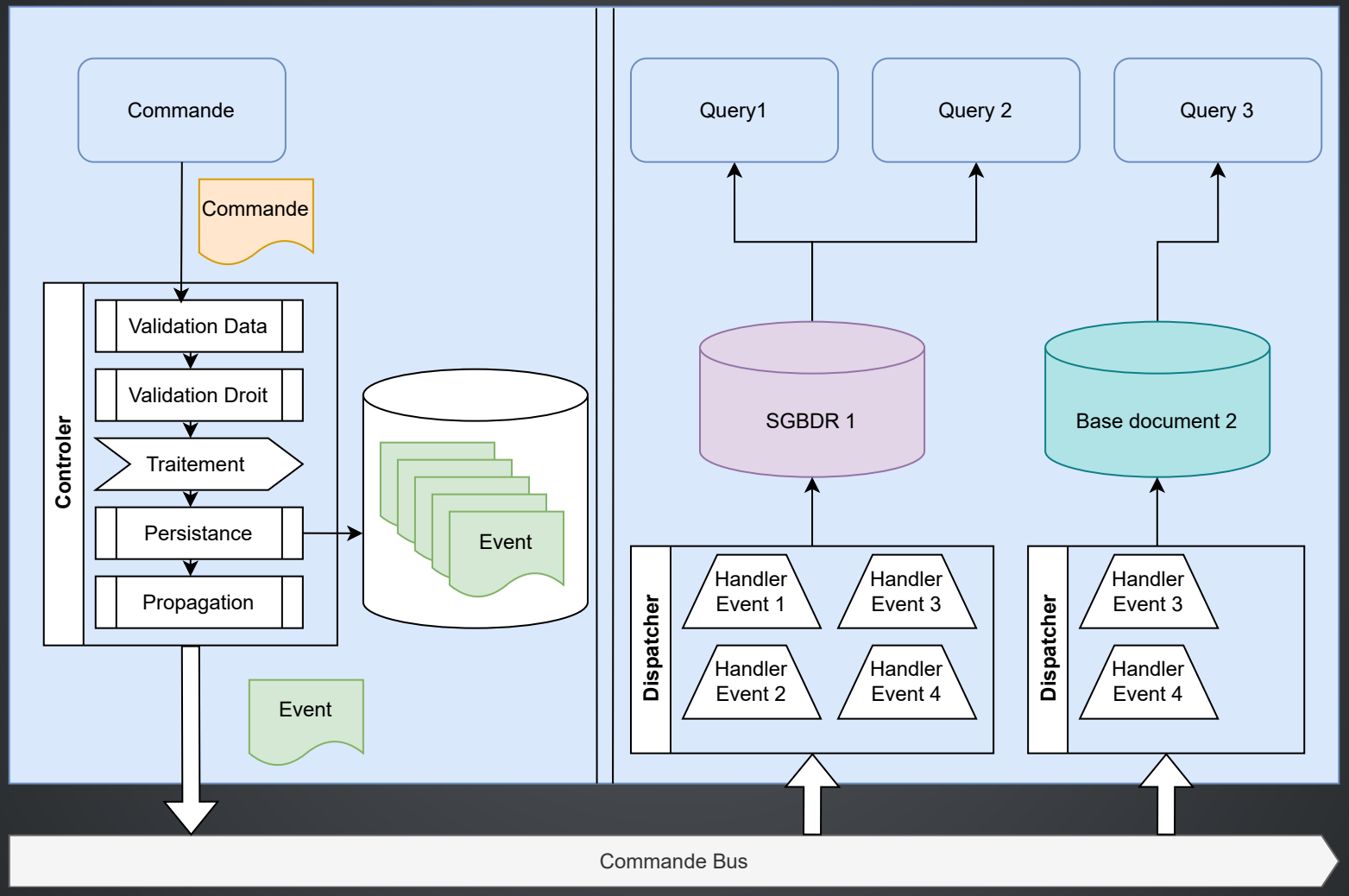
Un event est la pour répondre à plusieurs questions :

- Qui ?
- Quand ?
- Quoi ?

UN "EVENT"

Un event est la pour répondre à plusieurs questions :

- Qui ?
- Quand ?
- Quoi ?
- Avec quoi ?


Commande
Commande
Controler
Validation Data
Validation Droit
Traitement
Persistance
Propagation
Event
Event
Query1
Query 2
Query 3
SGBDR 1
Base document 2
Dispatcher
Handler Event 1
Handler Event 3
Handler Event 2
Handler Event 4
Dispatcher
Handler Event 3
Handler Event 4
Commande Bus

AVANTAGES

AVANTAGES

👍 Aucune perte d'information.

AVANTAGES

👍 Aucune perte d'information.
👍 Introduction de la notion d'intention.

AVANTAGES

- 👍 Aucune perte d'information.
- 👍 Introduction de la notion d'intention.
- 👍 On peut à tout moment TOUT reconstruire.

AVANTAGES

- 👍 Aucune perte d'information.
- 👍 Introduction de la notion d'intention.
- 👍 On peut à tout moment TOUT reconstruire.
- 👍 Grande tolérance à l'erreur.

AVANTAGES

- 👍 Aucune perte d'information.
- 👍 Introduction de la notion d'intention.
- 👍 On peut à tout moment TOUT reconstruire.
- 👍 Grande tolérance à l'erreur.
- 👍 L'inconsistance n'est plus qu'un simple bug.

AVANTAGES

- 👍 Aucune perte d'information.
- 👍 Introduction de la notion d'intention.
- 👍 On peut à tout moment TOUT reconstruire.
- 👍 Grande tolérance à l'erreur.
- 👍 L'inconsistance n'est plus qu'un simple bug.
- 👍 Règles métiers simples à spécifier.

AVANTAGES

👍 Aucune perte d'information.
👍 Introduction de la notion d'intention.
👍 On peut à tout moment TOUT reconstruire.
👍 Grande tolérance à l'erreur.
👍 L'inconsistance n'est plus qu'un simple bug.
👍 Règles métiers simples à spécifier.
👍 Pas besoin d'ajouter des exceptions dans le code.

AVANTAGES

👍 Aucune perte d'information.
👍 Introduction de la notion d'intention.
👍 On peut à tout moment TOUT reconstruire.
👍 Grande tolérance à l'erreur.
👍 L'inconsistance n'est plus qu'un simple bug.
👍 Règles métiers simples à spécifier.
👍 Pas besoin d'ajouter des exceptions dans le code.
👍 Facile à s'adapter pour les problèmes de performance.

AVANTAGES

👍 Aucune perte d'information.
👍 Introduction de la notion d'intention.
👍 On peut à tout moment TOUT reconstruire.
👍 Grande tolérance à l'erreur.
👍 L'inconsistance n'est plus qu'un simple bug.
👍 Règles métiers simples à spécifier.
👍 Pas besoin d'ajouter des exceptions dans le code.
👍 Facile à s'adapter pour les problèmes de performance.
👍 Par nature, le code est découpé et organisé.

AVANTAGES

👍 Aucune perte d'information.
👍 Introduction de la notion d'intention.
👍 On peut à tout moment TOUT reconstruire.
👍 Grande tolérance à l'erreur.
👍 L'inconsistance n'est plus qu'un simple bug.
👍 Règles métiers simples à spécifier.
👍 Pas besoin d'ajouter des exceptions dans le code.
👍 Facile à s'adapter pour les problèmes de performance.
👍 Par nature, le code est découpé et organisé.
👍 Possible de reproduire en local.

INCONVÉNIENTS

INCONVÉNIENTS

👎 Difficile de mettre cela en place après.

INCONVÉNIENTS

- 👎 Difficile de mettre cela en place après.
- 👎 C'est plus "naturel", mais moins intuitif.

INCONVÉNIENTS

👎 Difficile de mettre cela en place après.
👎 C'est plus "naturel", mais moins intuitif.
👎 Le code est simple, mais l'architecture complexe.

INCONVÉNIENTS

👎 Difficile de mettre cela en place après.
👎 C'est plus "naturel", mais moins intuitif.
👎 Le code est simple, mais l'architecture complexe.
👎 Cette approche pose les bases, mais tout reste à faire.

INCONVÉNIENTS

👎 Difficile de mettre cela en place après.
👎 C'est plus "naturel", mais moins intuitif.
👎 Le code est simple, mais l'architecture complexe.
👎 Cette approche pose les bases, mais tout reste à faire.
👎 Attendez-vous à devoir traiter des problèmes que vous n'auriez pas eus sans.

INCONVÉNIENTS

👎 Difficile de mettre cela en place après.
👎 C'est plus "naturel", mais moins intuitif.
👎 Le code est simple, mais l'architecture complexe.
👎 Cette approche pose les bases, mais tout reste à faire.
👎 Attendez-vous à devoir traiter des problèmes que vous n'auriez pas eus sans.
👎 Problème du temps de déploiement si reconstruction.

CONSEILS

CONSEILS

- Ne pas remettre à demain la qualité et le refactoring.

CONSEILS

- Ne pas remettre à demain la qualité et le refactoring.
- Prenez le temps de poser les bases.

CONSEILS

- Ne pas remettre à demain la qualité et le refactoring.
- Prenez le temps de poser les bases.
- Tolérance 0 sur les bugs "handler".

UN PEU DE PRATIQUE

https://gitlab.com/julienVinber/addressbook-cqrs-eventsourcing-symfony-demo