Piloté par les événements

Architecture

Sourcing d'événements et CQRS

Régis ATEMENGUÉ

@regis_ate www.regisatemengue.com

Objectifs du cours

- Comprendre le CQRS (ségrégation des responsabilités des requêtes de commande) et la source d'événements et l'appliquer à un système piloté par les événements

Sourcing d'événements et CQRS

E

S

- L'architecture basée sur les événements concerne principalement les services. Les événements peuvent
- également être utilisés comme éléments de base des données. Le sourcing d'événements et CQRS offrent
- un modèle pour stocker les données en tant qu'événements.

Problèmes avec les bases de données traditionnelles

- Les bases de données traditionnelles contiennent des données sur l'état actuel de l'entité. Cela
- est vrai pour les bases de données SQL et NoSQL.
    - Exemple : tableau des employés

id	firstname	lastname	gender	jobtitle	department	room	phoneextension
1	Martin	Tailor	male	Chief Executive Officer	CEO	A302	42
2	Ron	Williams	male	Marketing Manager	Marketing	A231	195
3	Mike	Copperfield	male	Chief Financial Officer	Administration	A304	219
4	Michael	Williams	male	Information Services Manager	Administration	A215	184
5	Chris	Miller	male	Vice President of Production	Production	A204	38
6	David	Norman	male	Vice President of Engineering	Production	A258	167
7	Kim	Marshal	female	Vice President of Sales	Sales	A329	74

- Ce tableau ne nous dit pas

a.Quel était le titre du poste de Martin Tailor ?

b.Quand Chris Miller a-t-il emménagé dans la salle A204 ?

c.L'un des employés a-t-il changé de nom ?

Problèmes avec les bases de données traditionnelles

- Les bases de données traditionnelles contiennent des données sur l'état actuel de l'entité. Il
- n'existe aucun moyen de voir les données historiques des entités.
- Les données sont un**instantané**d'un moment donné;

	A	B	C	D	E	F
1	**Date**	**Description**	**Withdrawals**	**Deposits**	**Balance**	**Category**
2		Previous balance			$78.00	
3	7/1/2022	Visa keb DKK 95,00 Mc Donalds V 503	$95.00		($17.00)	Takeout
4	7/2/2022	Wal-Mart 440	$300.00		($317.00)	Family fee
5	7/3/2022	Payroll Deposit - HOTEL		$2,100.00	$1,783.00	Income
6	7/4/2022	Interac Purchase - ELECTRONIC	$495.00		$1,288.00	Work fee
7	7/5/2022	Interac Purchase - SUPERMARKET	$240.00		$1,048.00	Family fee
8	7/6/2022	Cash (over the counter / at Egoli Bank ATM)		$1,280.00	$2,328.00	Other
9	7/7/2022	Monthly maintenance fee	$5.00		$2,323.00	Other
10	7/8/2022	Web Bill Payment - MASTERCARD	$55.00		$2,268.00	Work fee
11	7/9/2022	Telephone Bill Payment - VISA	$25.00		$2,243.00	Family fee
12	7/10/2022	Payroll Deposit - HOTEL		$2,100.00	$4,343.00	Income
13	7/11/2022	Dankort-nota Louis Nielsen 27650	$20.00		$4,323.00	Shopping
14	7/12/2022	GOOGLE *YouTubePremium	$12.99		$4,310.01	Subscription
15	7/13/2022	BRANDNEW CAR service	$20.00		$4,290.01	Car

Recherche d'événements

Recherche d'événements

Modèle de sourcing d'événements

- Un modèle de magasin de données dans lequel chaque modification des données est capturée et enregistrée. Utilisez le
- sourcing d’événements pour capturer les changements d’état sous forme de séquence d’événements. Les bases de
- données stockent une liste de modifications pour l'entité, pas pour l'entité elle-même.
- Aucune mise à jour ni suppression, juste des insertions.
- Contrairement aux actions CRUD sur les ressources, le sourcing d'événements utilise uniquement des actions CR (créer et lire). Chaque
- ligne documente un changement dans une ou plusieurs propriétés de l'entité.
- Dans ce modèle, la base de données est appelée**Boutique d'événements**

Prenons un exemple d'utilisation du sourcing d'événements pour stocker les commandes dans un système logiciel de commerce électronique. Le***service de commande***devrait pouvoir stocker les événements suivants :

Recherche d'événements

RésuméOrdreÉvénement

Événement de base abstrait pour d'autres événements concrets contenant des propriétés d'horodatage et d'identifiant de commande

- **CommandeCrééeÉvénement**

  Contient des informations de base sur la commande

- **CommandePaiementÉvénement**

  Contient des informations sur le paiement de la commande

- **ÉvénementModificationCommande**

  Contient des informations sur les modifications apportées par le client à la commande avant emballage

- **CommandePackagedEvent**

  Contient des informations sur la personne qui a collecté et emballé la commande

- **CommandeAnnuléeÉvénement**

  Décrit que le client a annulé la commande et que la commande ne doit pas être expédiée

- **CommandeExpédiéeÉvénement**

  Contient des informations sur le partenaire logistique et l'identifiant de suivi de l'expédition de la commande

- **CommandeLivréeÉvénement**

  Contient des informations sur le point de retrait de la commande livrée

Recherche d'événements

Ce n’est pas ainsi que fonctionnent la grande majorité des applications aujourd’hui.

La plupart des applications fonctionnent en stockant l'état actuel et en utilisant les états stockés pour traiter les transactions commerciales.

Recherche d'événements

Comment pouvons-nous visualiser l’état actuel d’une entité ?

En rejouant les événements qui lui sont liés, dès le premier événement

Event_id	timestamp	Event
1	2009-04-12	Employee Marc Copperfield joined
2	2009-04-12	Employees Marc Copperfield started work to sales Departement
3	2012-04-12	Jobs Title of Marc Copperfield updated to Vice President of Sales
4	2023-04-12	Jobs Title of Marc Copperfield updated to Chief Financial Officier
5	2023-08-05	Ron Williams Joined
6	2023-08-05	Jobs Title of Ron Willions updated to Marketing Manager

3	Mike	Copperfield	male	Chief Financial Officer	Administration	A304	219

Recherche d'événements

avantages

- Extrêmement facile à visualiser les données historiques
- Structure de base de données simple
- Opérations de base de données simples (pas de mises à jour, pas de concurrence)
- Insertions très rapides

Les inconvénients:

- L'affichage de l'état actuel de l'entité est fastidieux et lent
- Grande capacité de base de données (nombreux enregistrements par entité)

CQRS

Ségrégation des responsabilités des commandes et des requêtes

Modèle CQRS (ségrégation des responsabilités des requêtes de commande)

Moyens:

- Séparer les commandes (mises à jour/insertion/suppression) des requêtes Chacune d'entre
- elles se trouve dans une base de données distincte
- La base de données de commandes est implémentée en tant que magasin d'événements pour améliorer les performances et la simplicité.
- La base de données de requêtes stocke les entités

Utilisez le modèle CQRS si vous souhaitez utiliser un modèle différent pour les opérations de création/mise à jour (= commande) par rapport au modèle que vous souhaitez utiliser pour interroger des informations.

Considérons l'exemple de service de commande précédent qui utilisait le sourcing d'événements. Dans le service de commande, toutes les commandes sont des événements. Nous voulons que les utilisateurs puissent interroger efficacement les commandes. Nous devrions avoir une représentation supplémentaire d'une commande en plus des événements car il est inefficace de toujours générer l'état actuel d'une commande en rejouant tous les événements associés.

Modèle CQRS (ségrégation des responsabilités des requêtes de commande)

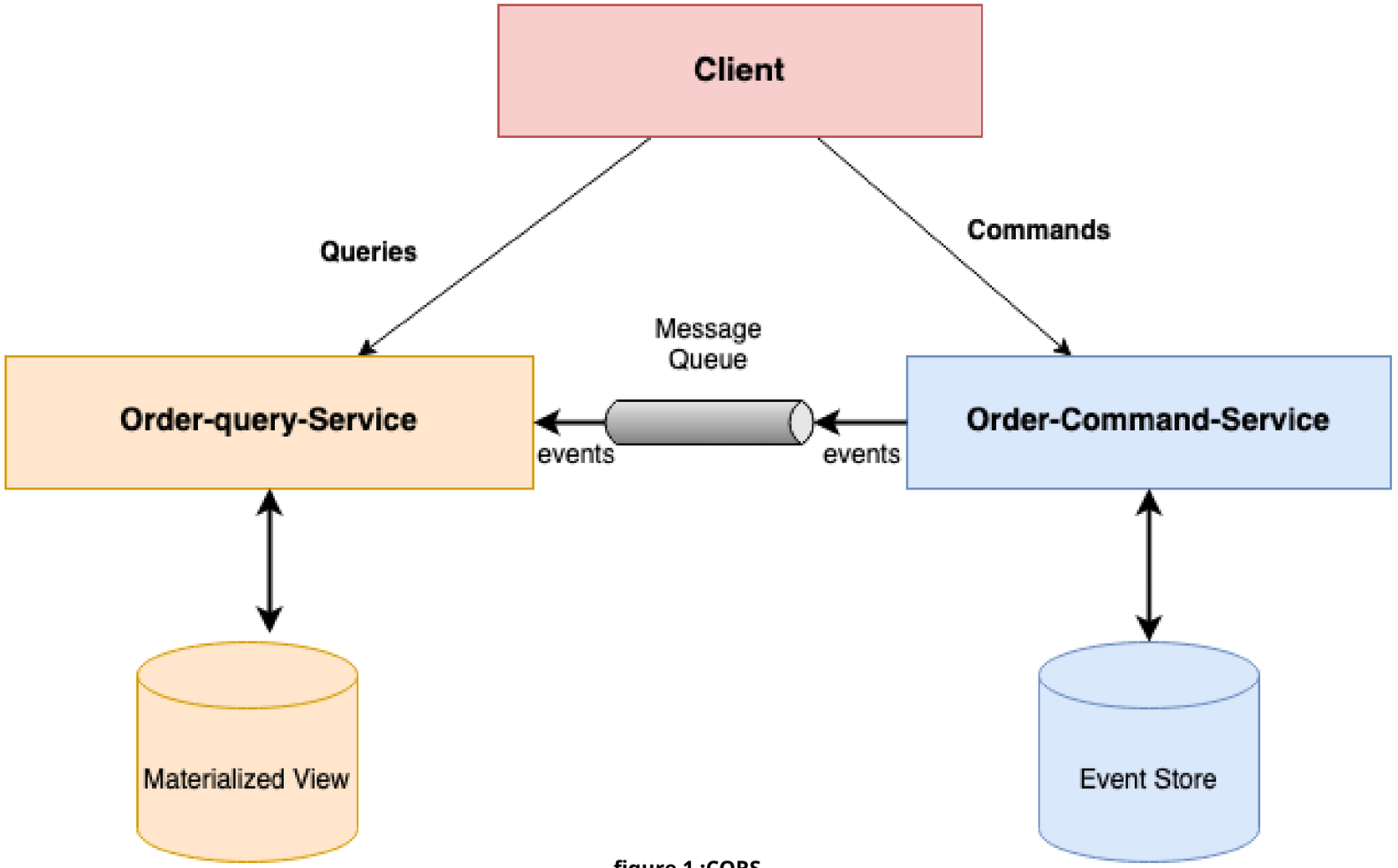


figure 1 :CQRS

Modèle CQRS (ségrégation des responsabilités des requêtes de commande)

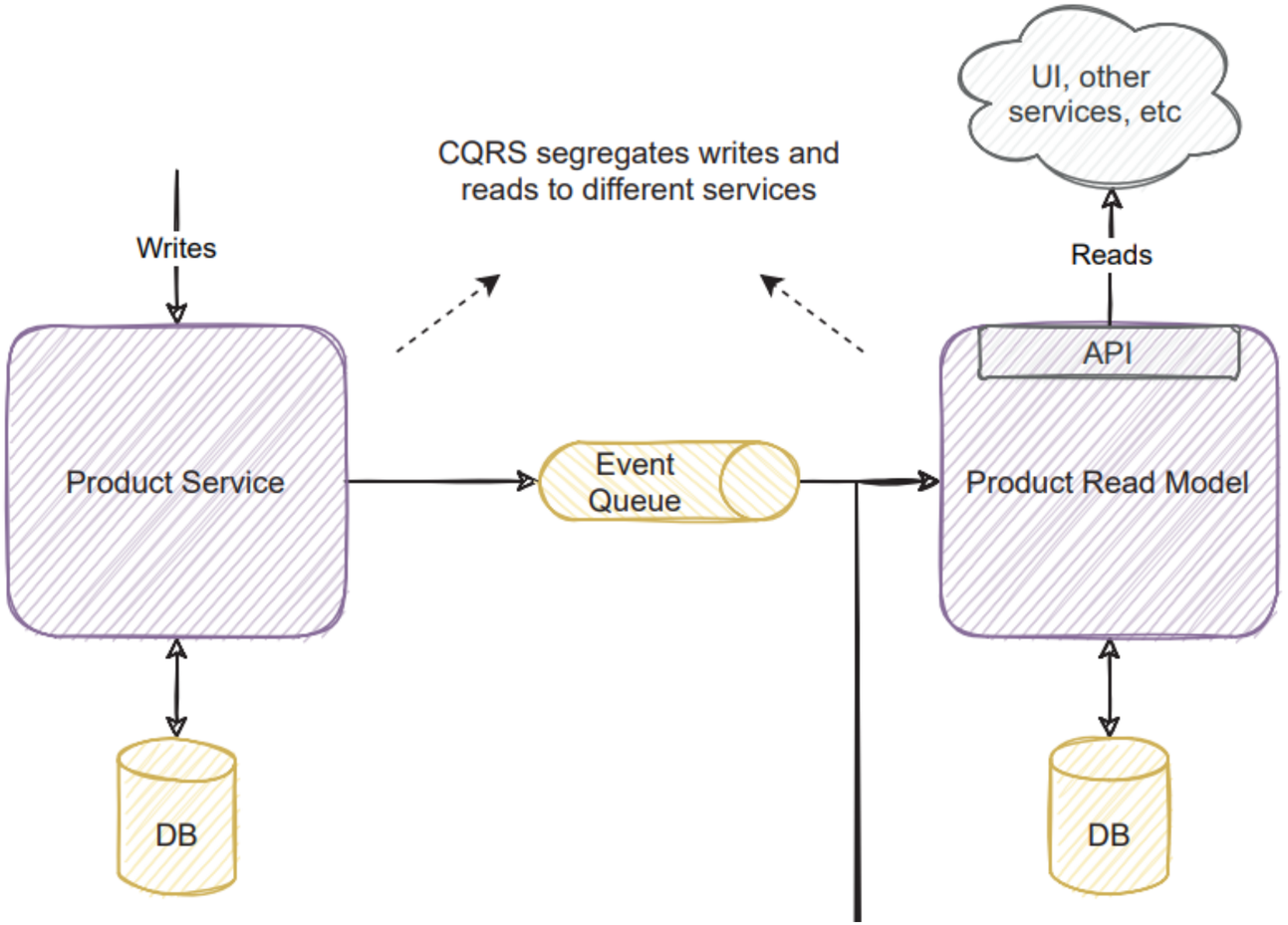


fig 2 CQRS

Modèle CQRS (ségrégation des responsabilités des requêtes de commande)

Saveurs de CQRS

Modèle CQRS (ségrégation des responsabilités des requêtes de commande)

Saveurs de CQRS Regular : applications CRUD

Régulier–au lieu de l'approche CRUD classique avec 3 couches, nous séparons la couche métier en couche de commande et couche de requête. Pour la pile de commandes, vous devez utiliser un modèle qui convient mieux : modèle de domaine, module de table, script de transaction,... Pour la pile de requêtes, utilisez uniquement le code qui fait le travail.

– Outils ORM que vous utiliseriez normalement, base de données avec laquelle vous êtes à l'aise. Pour le contexte de la base de données, vous n'avez besoin que d'une partie de lecture afin de pouvoir créer une façade pour le contexte afin d'exposer uniquement les fonctions de requête et les fonctions de lecture restent derrière la façade afin que personne ne puisse les utiliser. La version standard utilise une seule base de données pour les deux piles. Mais comment mettre à jour la partie de lecture dont la commande a modifié certaines données ? Eh bien, lorsque le travail de la base de données est terminé, nous effectuons simplement un appel Post-Redirect-Get sur la vue pour recharger les données.

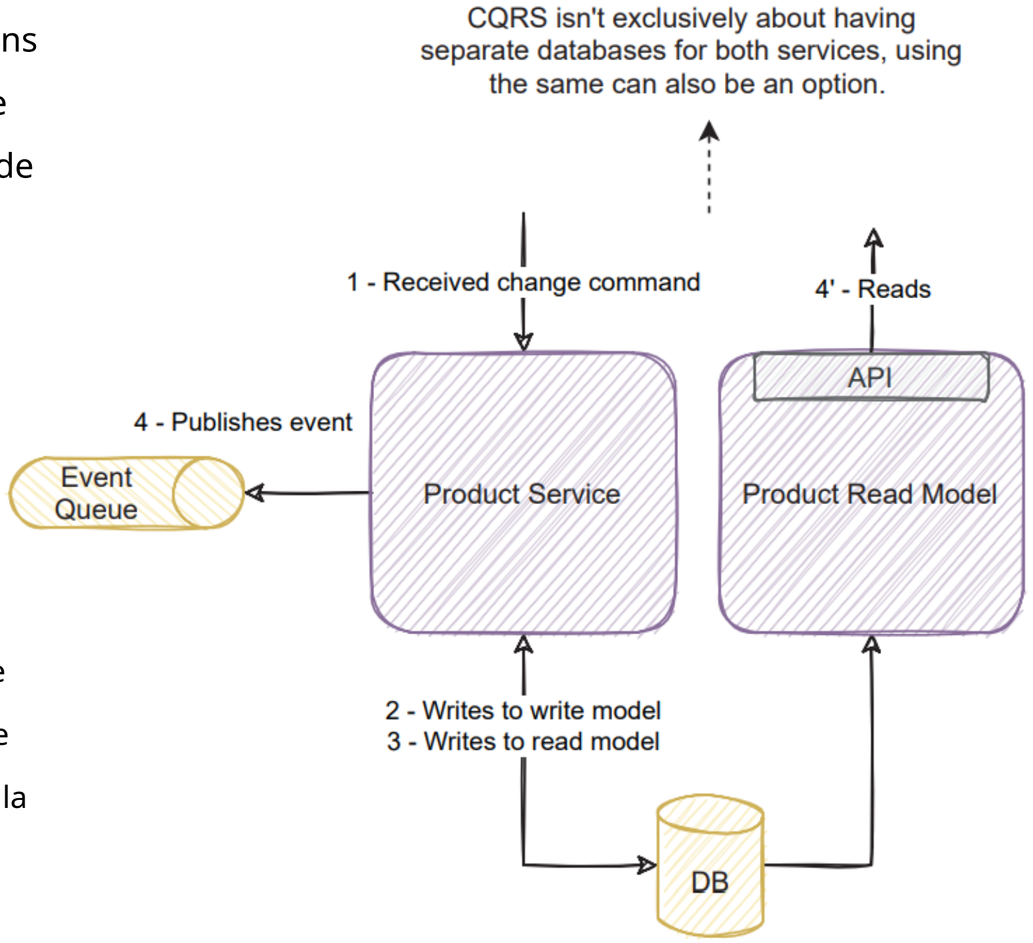


fig 3 CQRS régulier

Modèle CQRS (ségrégation des responsabilités des requêtes de commande)

Versions de CQRS Premiun : synchronisation du stockage des requêtes de commande

Prime–nous avons une pile de commandes et une pile de lecture séparées comme en mode normal, mais ici nous utilisons deux bases de données différentes – une pour chaque pile. L'important est que les bases de données restent synchronisées. Et encore une fois, pour la pile de commandes, vous utilisez uniquement le modèle qui convient le mieux : une architecture orientée tâches avec un modèle de domaine ou un script de transaction et éventuellement un stockage ad hoc (base de données relationnelle, NoSql ou stockage d'événements). Et pour la pile de requêtes encore une fois, utilisez uniquement le code qui fait le travail (outil ORM que vous préférez, stockage ad hoc). L'important est de savoir comment définiriez-vous les données obsolètes et quelle est l'importance des données obsolètes pour votre application ? Discutons des possibilités de synchronisation des données entre la base de données de commandes et la base de données de requêtes.

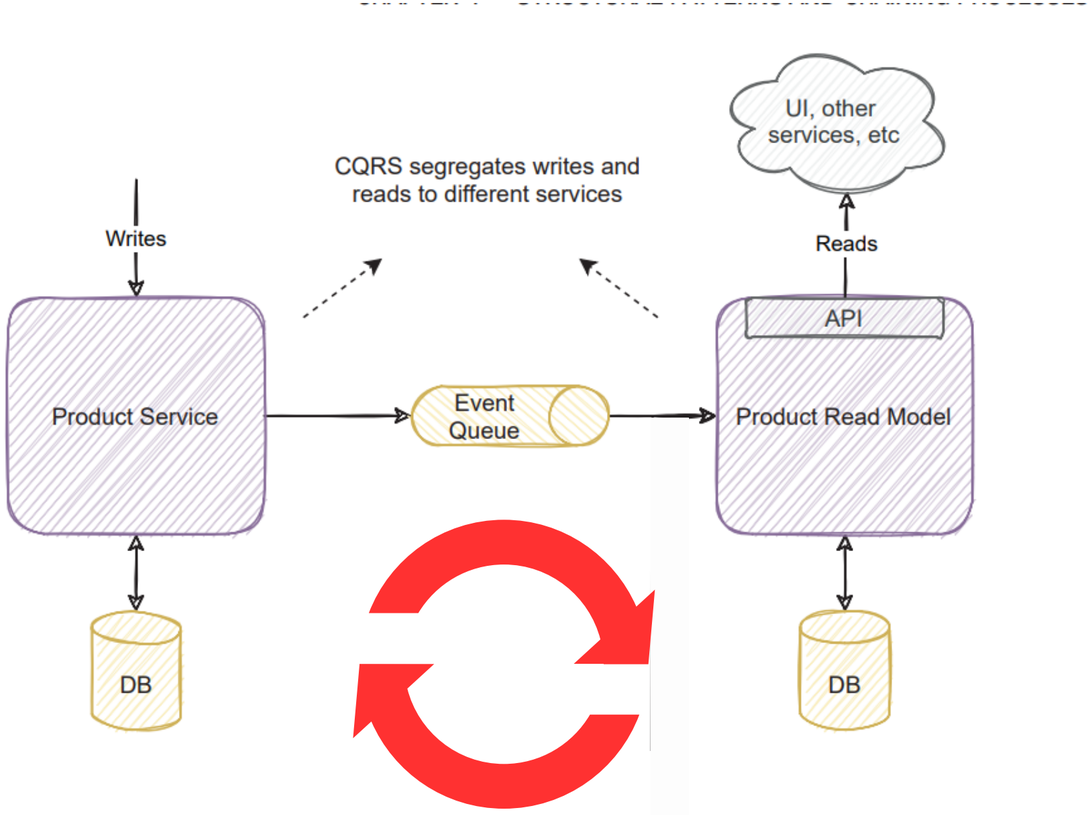
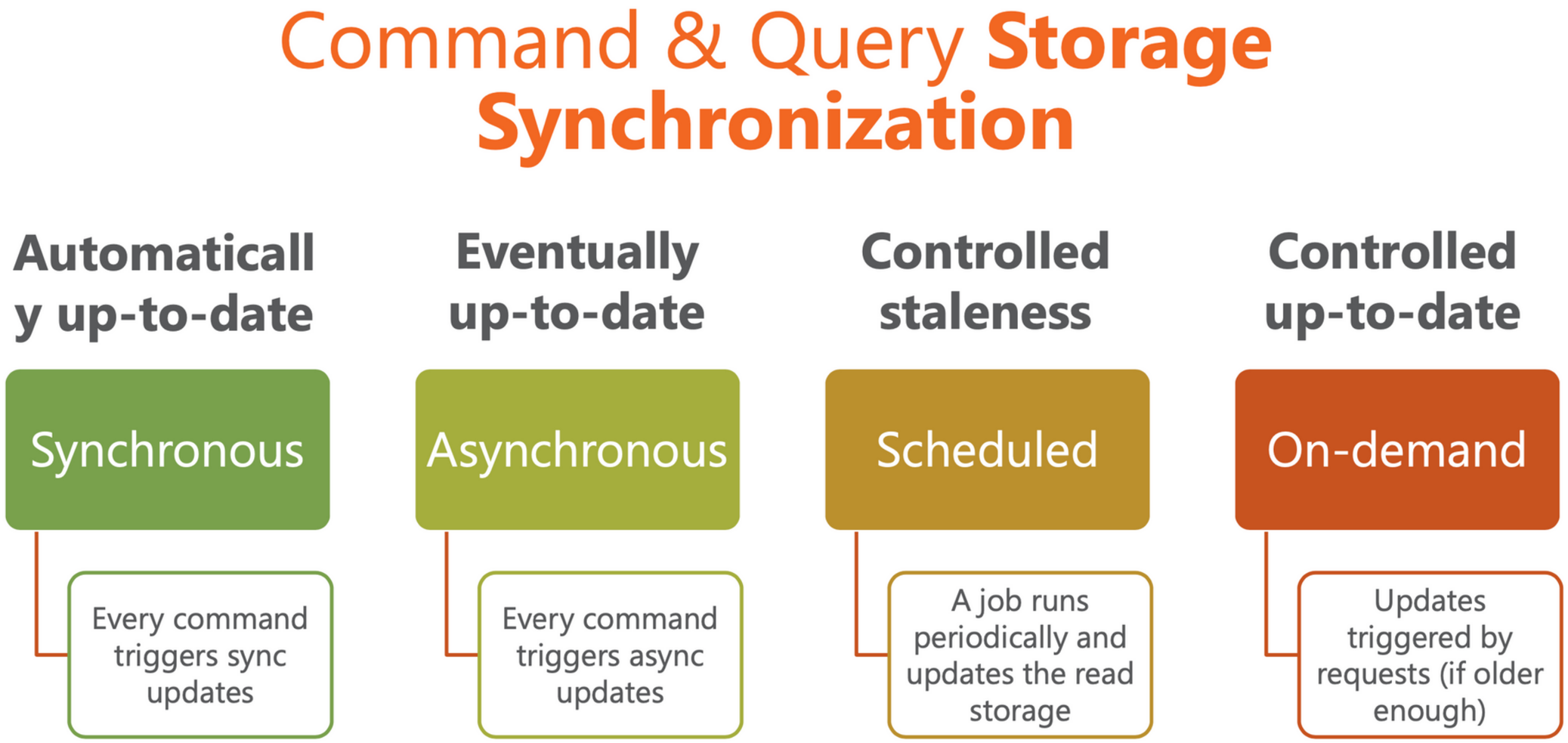


fig 4CQRS Premium

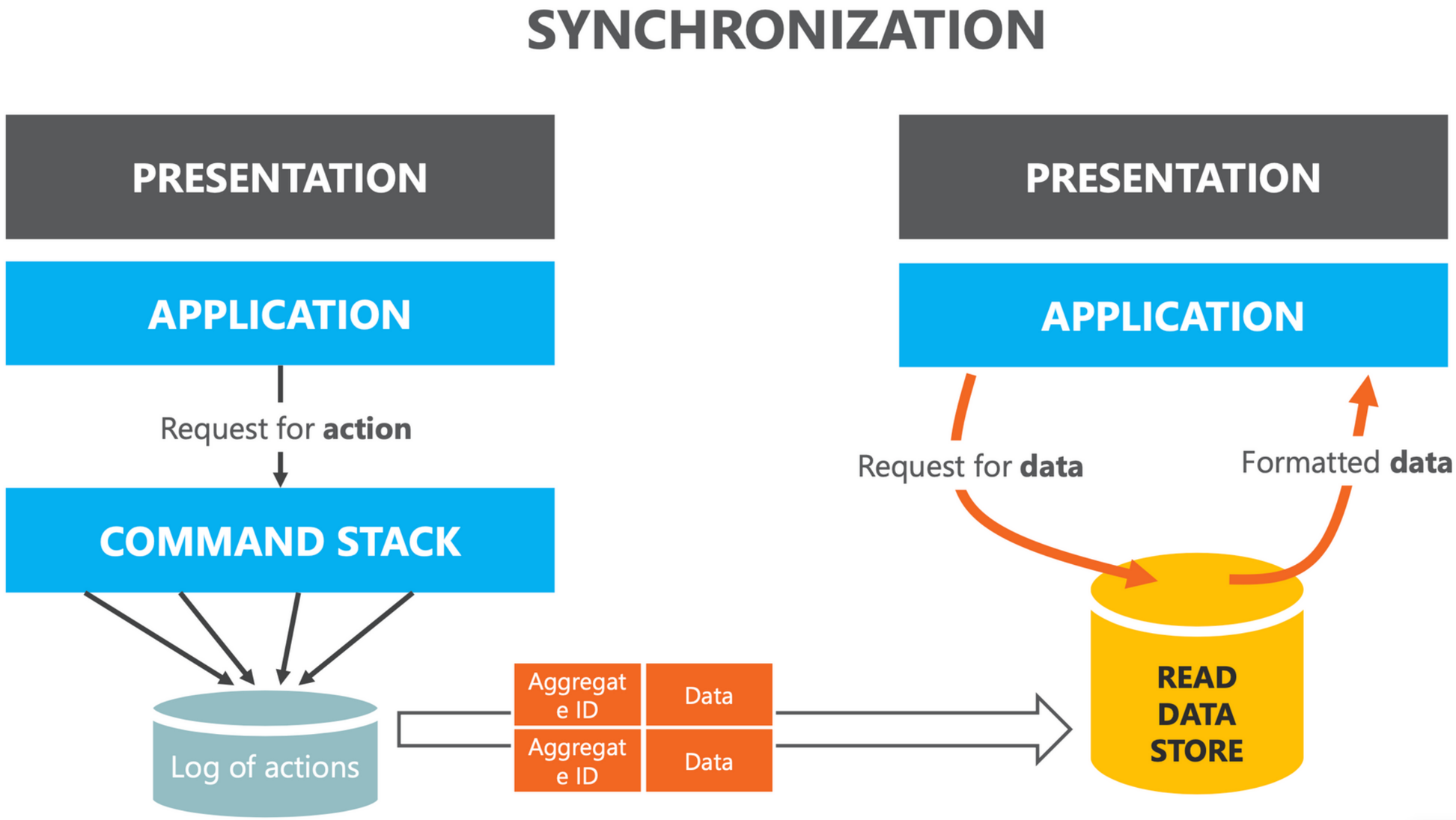
Modèle CQRS (ségrégation des responsabilités des requêtes de commande)

Versions de CQRS Premiun : synchronisation du stockage des requêtes de commande



Modèle CQRS (ségrégation des responsabilités des requêtes de commande)

Versions de CQRS Premiun : synchronisation du stockage des requêtes de commande



Modèle CQRS (ségrégation des responsabilités des requêtes de commande)

Saveurs de CQRS Deluxe

DELUXE, qui inclut DDD, séparation de pile de commandes/ requêtes et sourcing d'événements

De luxe–identique au premium mais avec**orchestration basée sur des messages**des tâches grâce à l'utilisation de messages et d'événements. Les messages et les événements apportent de nouveaux outils dans le jeu : le service bus. Permettez-moi de jeter un œil à un flux de travail typique dans une saveur de luxe. Mon scénario implique des sagas, NServiceBus, NEventStore :

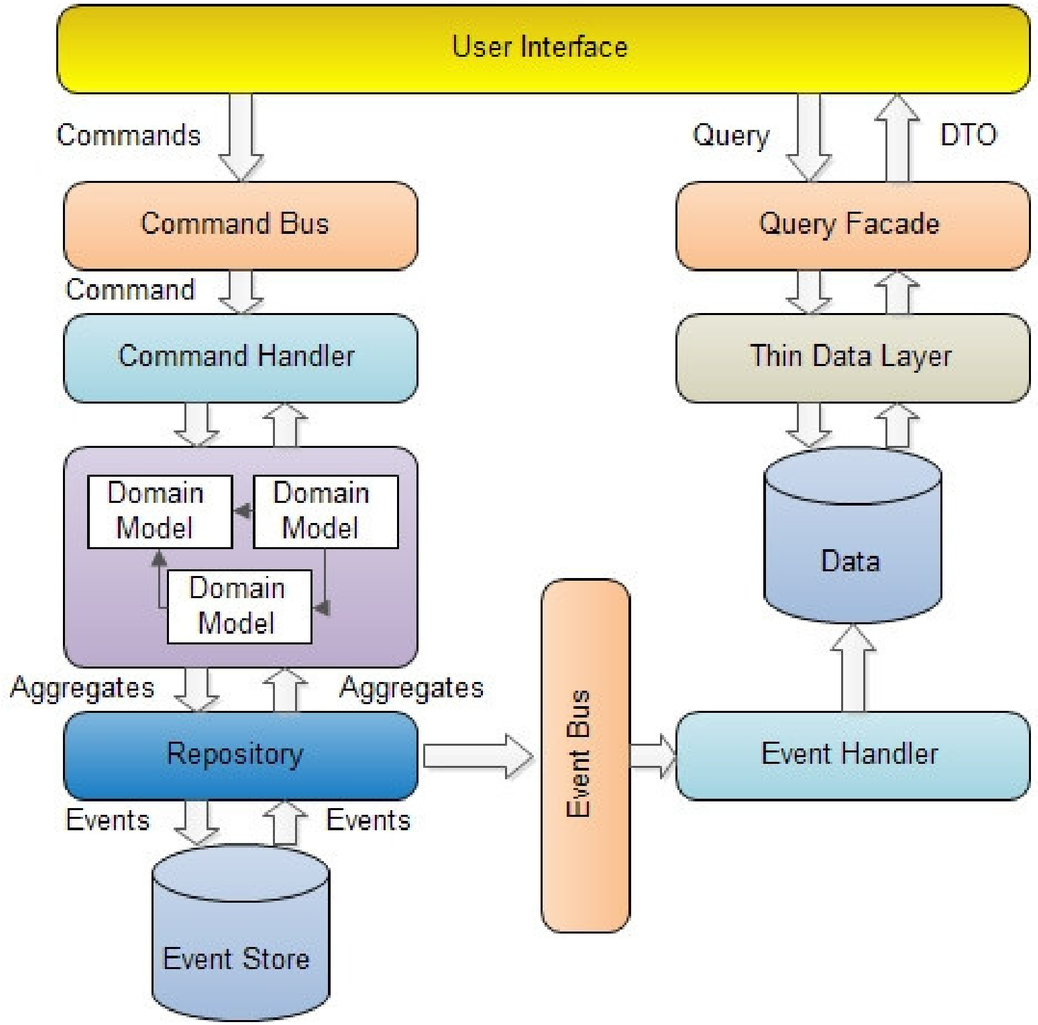


fig 5 CQRS Deluxe

Modèle CQRS (ségrégation des responsabilités des requêtes de commande)

Avantages

Combine les professionnels de l'Event Sourcing avec les requêtes d'entités traditionnelles. Aucune incidence sur les performances lors des requêtes.

Aucune amélioration des performances lors de l'interrogation des entités

Les inconvénients

Les données de l'entité ne sont pas mises à jour en temps réel. Difficile à configurer et à maintenir

Quand utiliser le sourcing d’événements et CQRS

CQRS n’est pas une solution miracle et ne doit pas non plus être appliqué à tous les cas d’utilisation. Cela implique une pièce mobile supplémentaire et augmente la complexité de la solution. Le fait que les écritures soient séparées des lectures avec une file d'attente d'événements entre les deux signifie que le côté lecture est également finalement cohérent si ce n'était déjà fait. Donc *quand devrions-nous utiliser CQRS ?*

- Quand l’accès aux données historiques est extrêmement important
    - Réglementation, finance, santé, etc.

- Lorsque les données sont volumineuses et que la relecture des événements n'est pas
- possible. Lorsque les performances sont critiques (insertions ou requêtes).**s)**
- si nous avons un service où les lectures dépassent largement les écritures et que nous avons besoin de mettre à l'échelle le différentiel. Avoir un modèle d'écriture conceptuel sensiblement différent d'un modèle de lecture complexe pourrait également être une bonne raison