EVOLUTION

Architecture NewBank

V7: Scale
Big loads
Multi regions

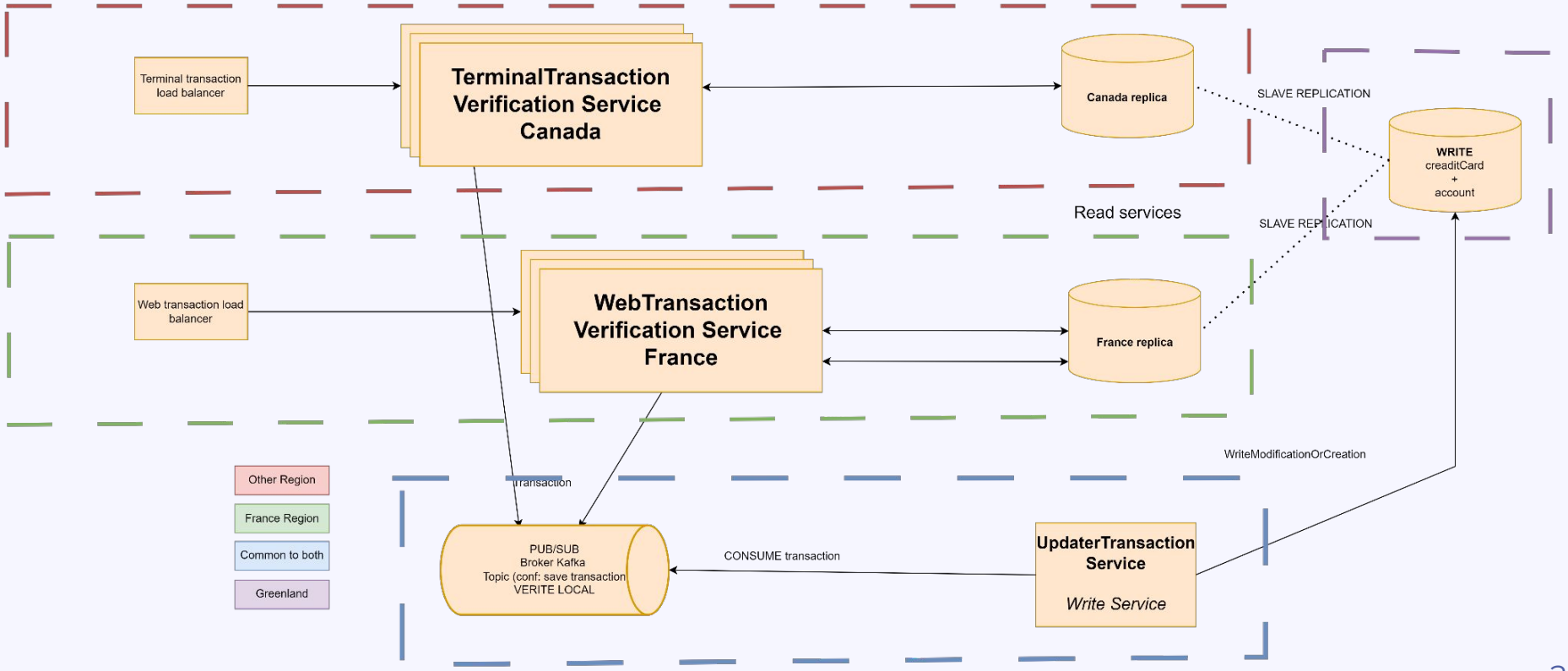
Igor Melnyk

Tobias Bonifay

Ayman Bassy

Mathieu Schalkwijk

Architecture Multiregionale AL Construction



Problématiques Confrontées Al-Construction

En cas de montée en charge :

- **Saturation mémoire** dans Spring (garbage collector)
- Limite de connexions simultanées, database limitante
- **Timeout** de connexion DB
- **Pire des cas :** Surconsommation ressource allouée
  ⇒ **Arrêt du conteneur**
  ⇒ Cas de la DB : provoque **arrêt multiple** (“depends on”)

⇒ **Echec de la transaction**

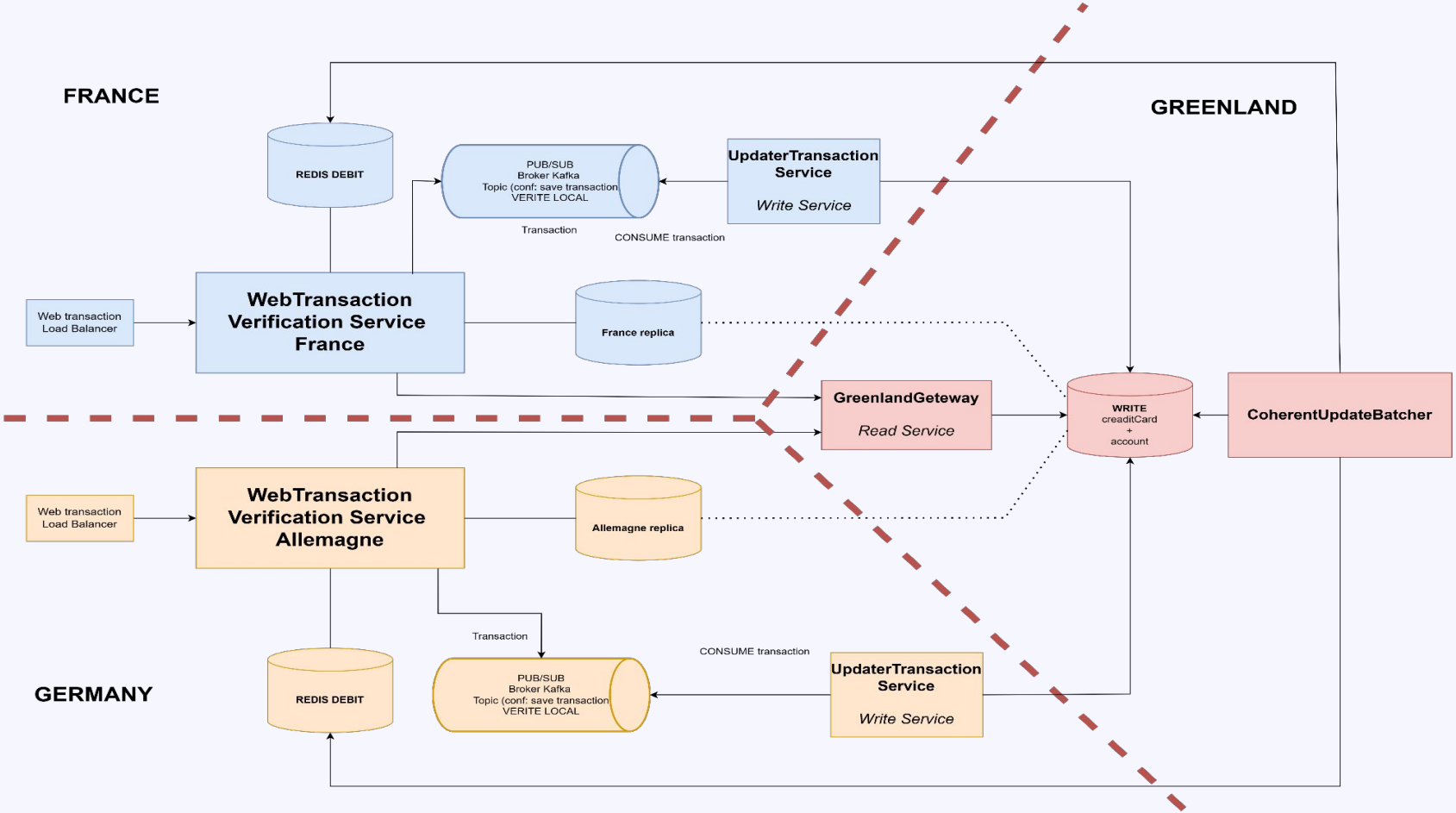
Evolutions

- **Algorithme réaliste de vérification des CBs**
  ⇒ Temps de traitement de la requête plus long

- **Lever l'hypothèse de temporisation entre transactions**
  ⇒ Multiples transactions en même temps

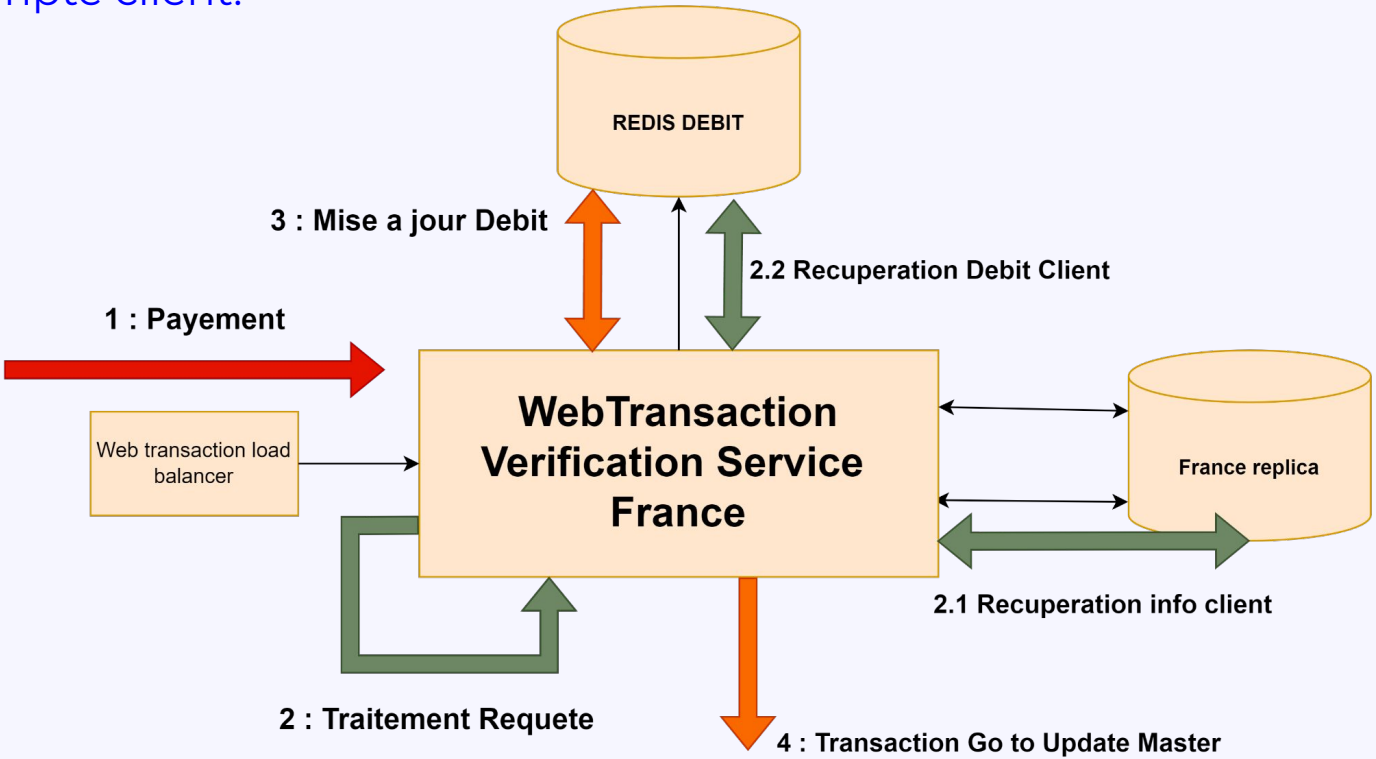
- **Failover sur une région connexe**
  ⇒ Accent mis sur la disponibilité du service de paiement

Nouvelle architecture



Traitement d'une transaction

Mise en place d'un notion de débit : représentation du paiement non encore prélevé du compte client.



Nouveau choix d'architecture

Mise à jour cohérente du système : Batch Process

Objectif :

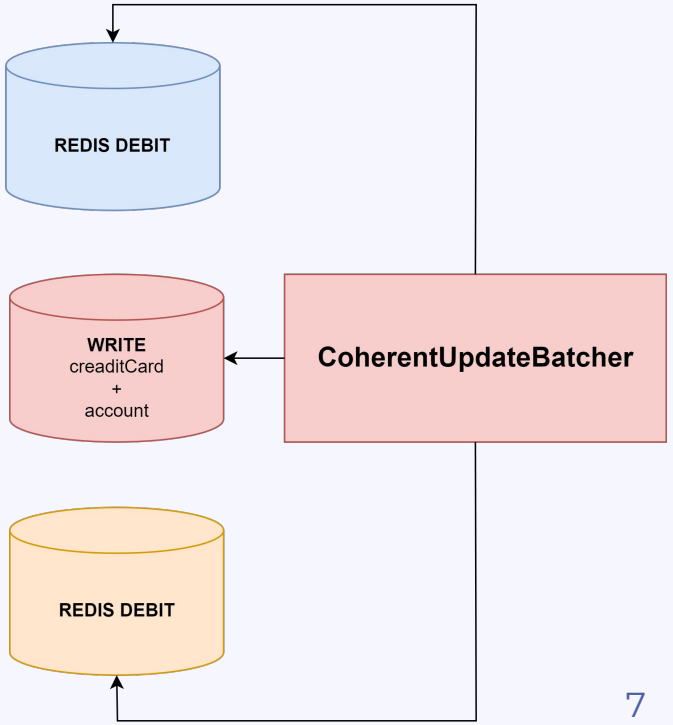
- **Remise à zéro** des débits clients contenus dans **Redis** et sur **Master.**
- Mise à jour des comptes bancaires **à leur vraie valeur.**

Processus (++ Disponibilité) :

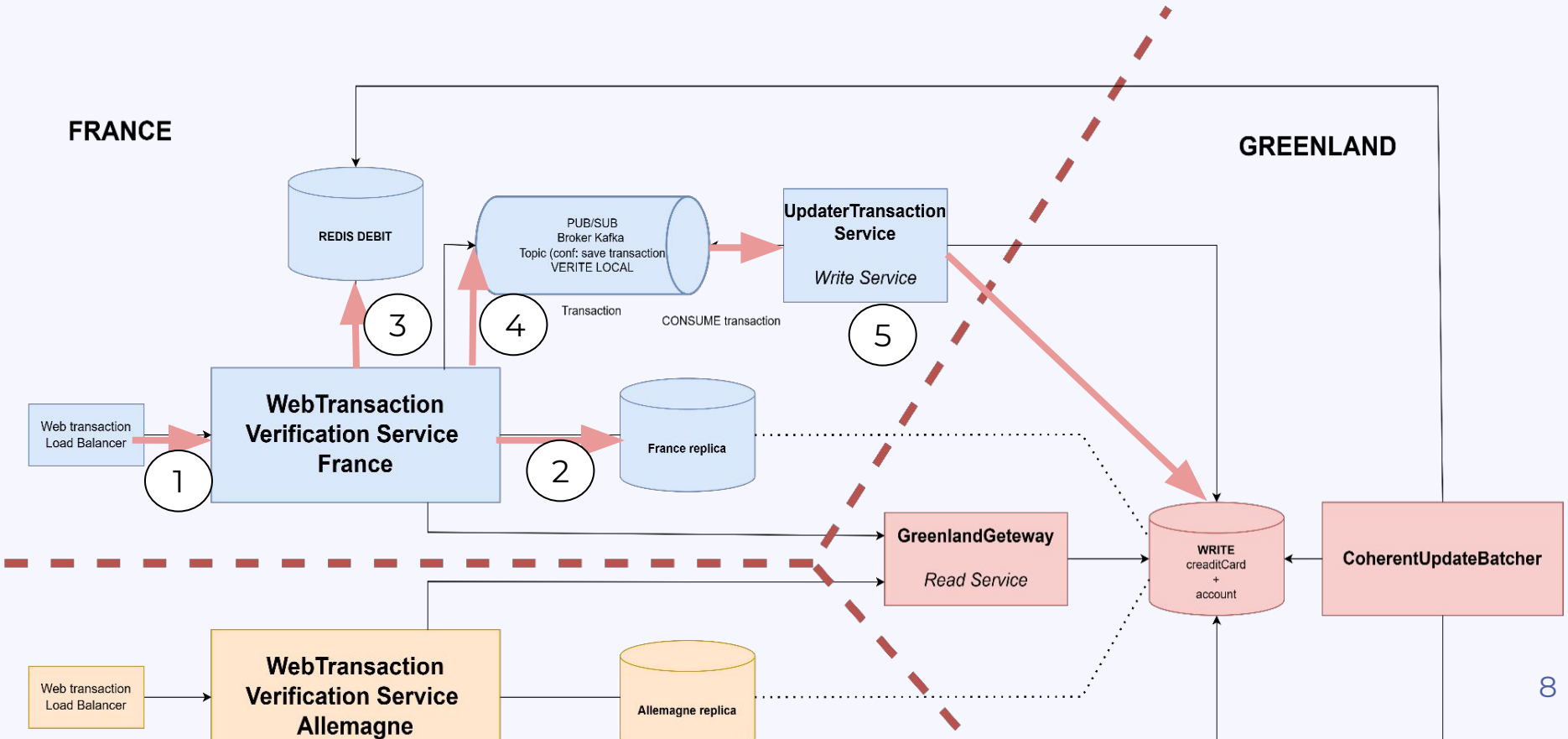
- Agit **client après client**
  ⇒ Paiement indisponible pour un client **seulement** pendant la période de mise à jour de son compte.
- Agit sur une **période de faible activité cliente.**

Exemple :

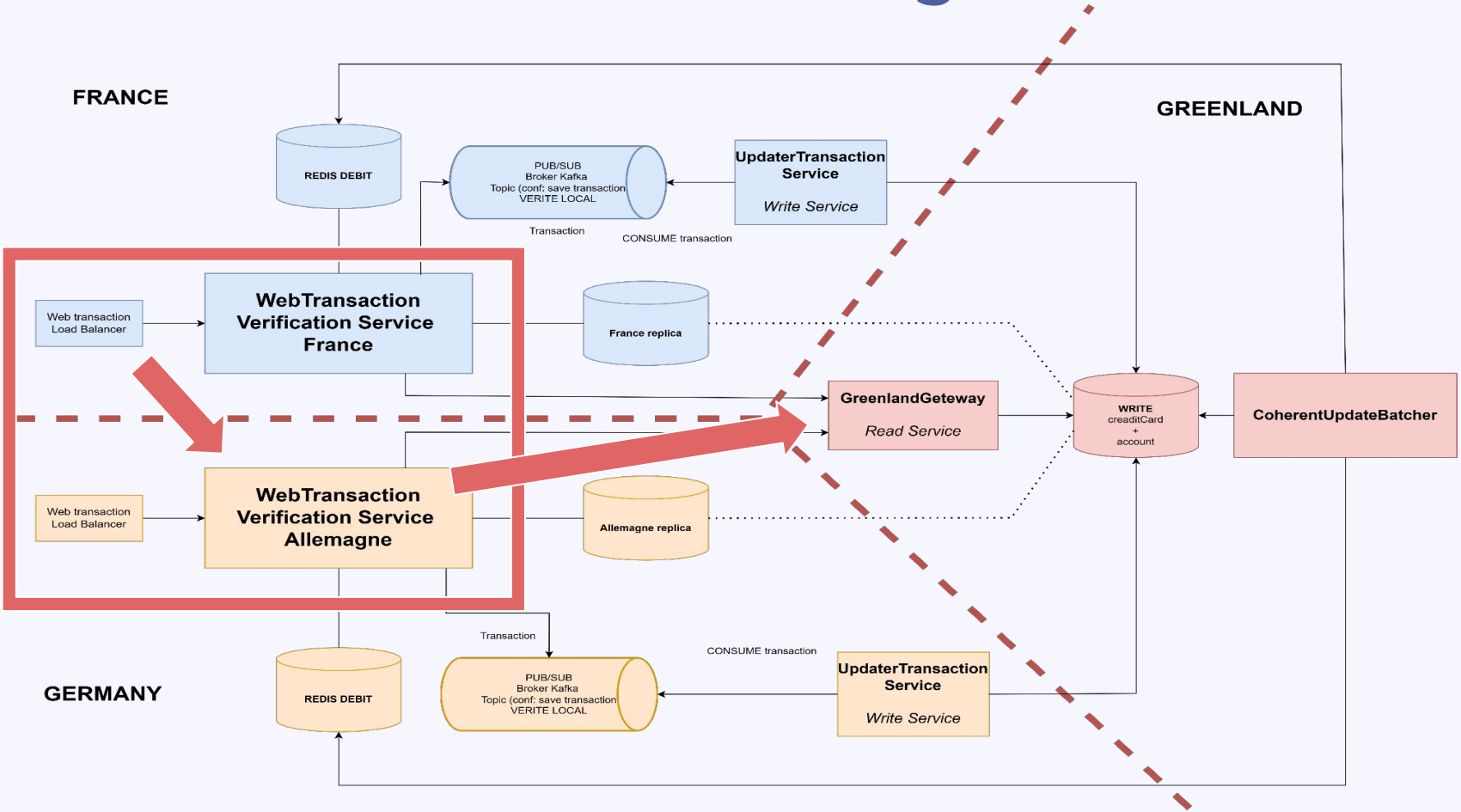
- **Compte client : 50$, Débit Redis: 0$**
- **Paiement de 30$**
- **Compte client : 50$, Débit Redis: 30$**
- **Mise à jour client …**
- **Compte client : 20$, Débit Redis: 0$**



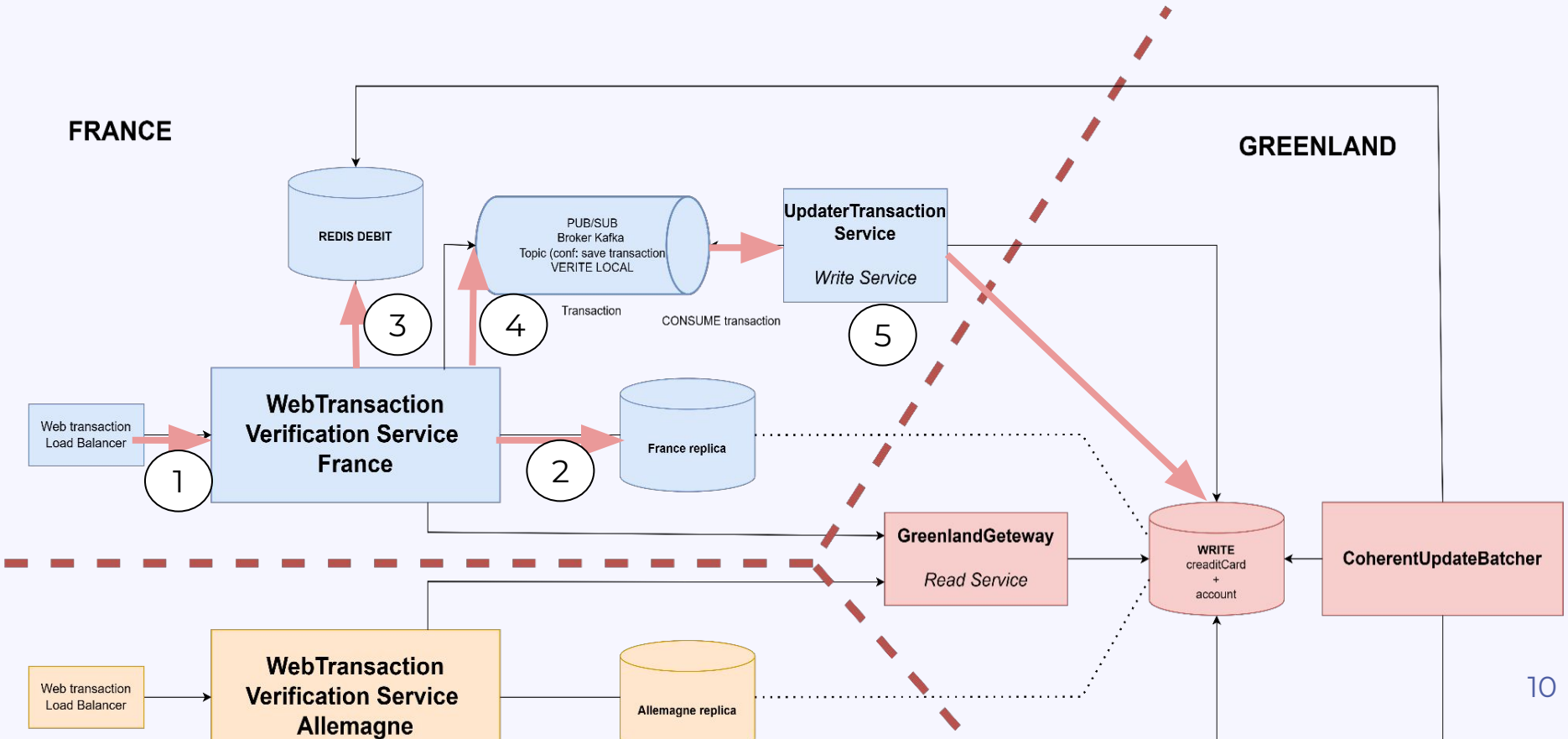
Scénario : transactions consécutives



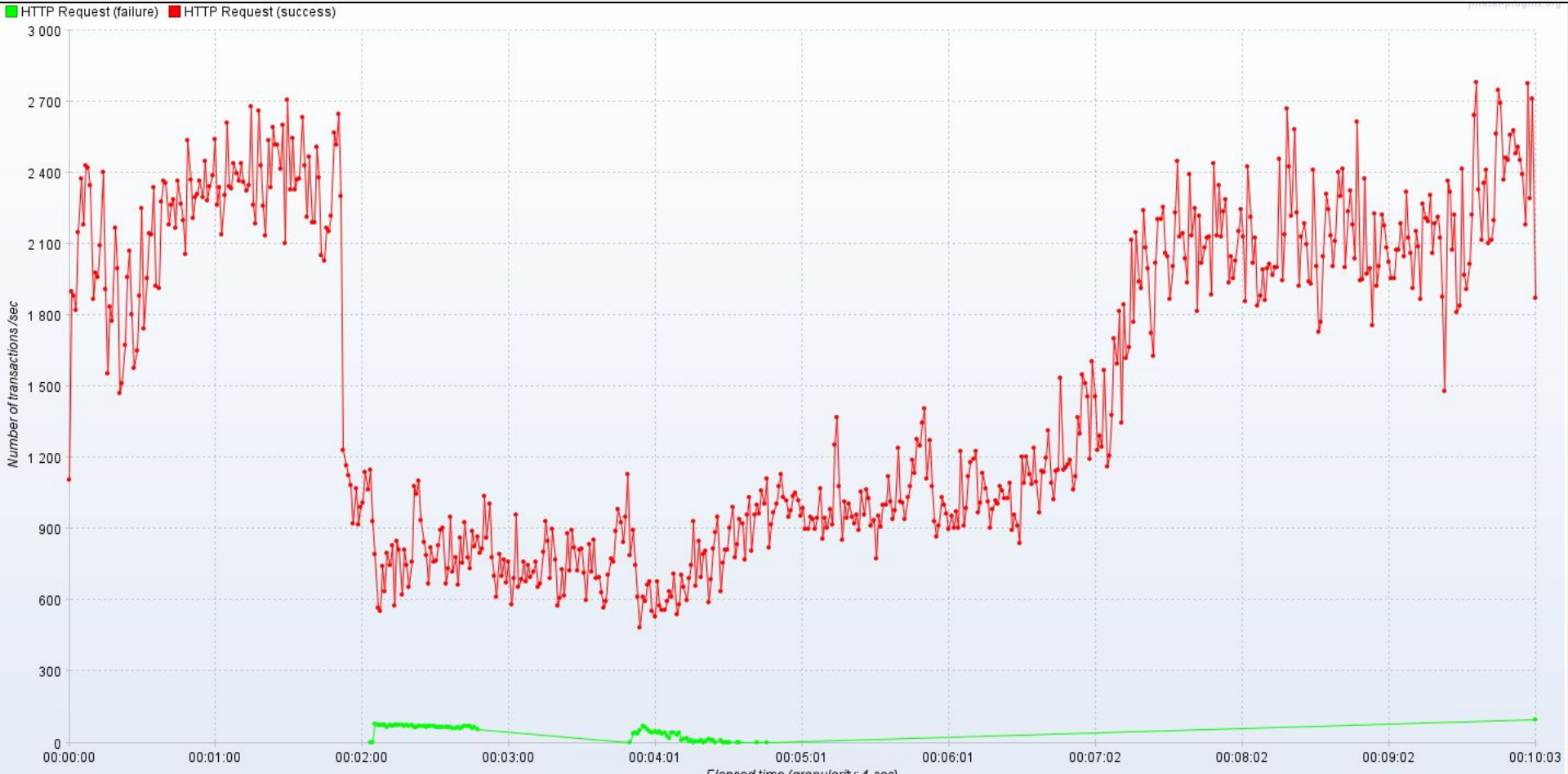
Failover inter région



Scénario : grosse charge de requêtes



Débit de transactions lors d'un scale down



Choix Technologique

Nouvelle database : REDIS

Avantage Database :

- **Ecriture RAM** : read-write rapide
- Structure de données **flexible** :

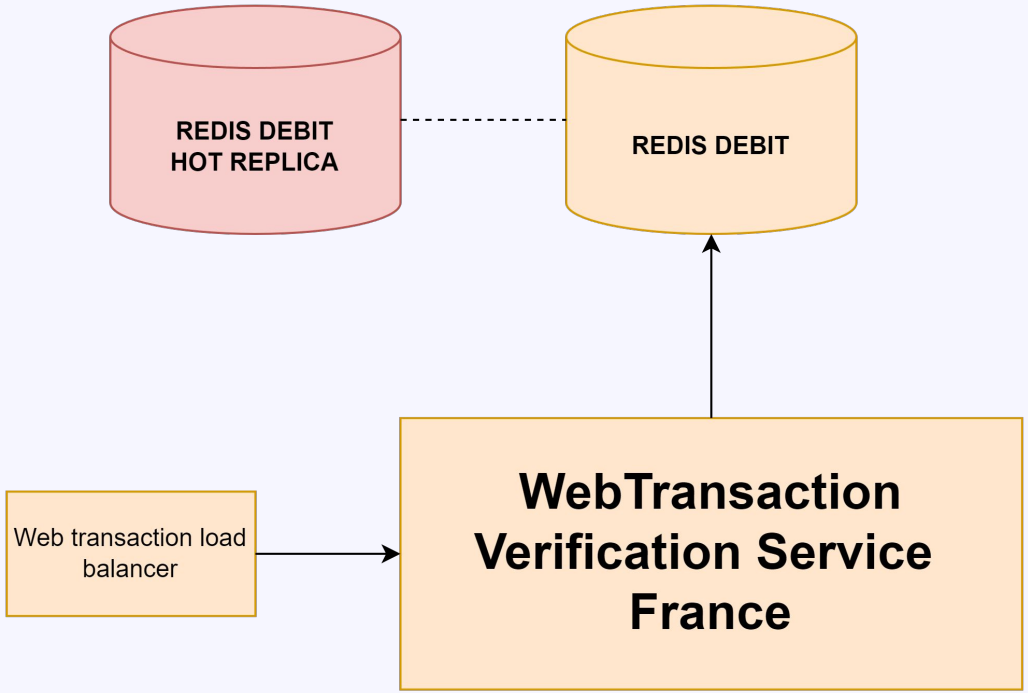
```
{
   “debitTotal” : integer,
   “debitFromTransaction : [ int , int , int ]
}
```

Avantage Architecture :

- Meilleur maintient de la **consistance** en cas d'enchaînement de transactions.

Possible Drawback :

- **Perte des informations** dans redis avant mise à jour cohérente.
  ⇒ **Hot Replica**



Faiblesses de l’architecture

[Coherence] : Un client ayant payé suite à un failOver, puis effectuant un nouveau paiement dans sa région par la suite, subira un déphasage de débit dans REDIS.

[Scaling] : Avec la mise en place du failOver, un pic de paiements pendant une période de failOver saturera assurément la base de données slave de la région relay. Les clients locaux et externes effectuent des paiements sur la région.

[Fonctionnel] : Les clients peuvent se retrouver dans le négatif suite à un paiement effectué juste après un paiement de failOver.

[Déploiement] : Il est difficile de faire de l'élasticité dynamique des instances avec docker-compose sans arrêter les conteneurs, ou sans utiliser manuellement un script avec “docker-compose --scale”.

Organisation de l’équipe

Igor

Gestion des transactions à l’étranger (G gateway)

Tobias

Vérification réaliste des CBs
CI-CD

Mathieu

Batch d’update de la balance
Load balancer failover

Ayman

Stockage des débits dans Redis et logique de débit en cours

Difficultés rencontrées

- Il est très difficile de faire **un équilibre** entre la **cohérence des données** ET la **disponibilité du système de paiement** pour les clients tout en restant **rapide.**

- **Des difficultés à réaliser la démonstration sur machine** (test de charge).

Perspectives futures

- Implémentation de la fonctionnalité restante **“diverse regulations”.**
- Mise en oeuvre d’une solution de **Cohérence pour le failover inter-régions :** Installation d’un **pub/sub au groenland,** récupérant les **événements de paiement des clients** ayant eu lieu dans le cadre d’un **failOver,** et mise en place d’un service lié à **Redis pour la consommation** de ce pub/sub**. Au redémarrage de la région,** ce service viendra mettre à jour Redis client par client.
- Réalisation d’une réplication master-slave de PostgreSQL des données clients en sharding sur la région des clients.
- Changement de supports de déploiement ⇒ Kubernetes / Docker Swarm.

Merci pour votre attention !