Communication par events asynchrones

Mode d’emploi chez Ovrsea

Opération Synchrone

Bloquant et impactant

```
const sendEmail = async () => {
  await new Promise(res => setTimeout(res, 2000));
  console.log("Email sent")
}

// Create shipment mutation resolver

console.log("Create shipment")
await sendEmail()
console.log("Done")
```

```
Create shipment
Email sent
Done
```

Opération Asynchrone

Non bloquant mais impactant

```
const sendEmail = async () => {
  await new Promise(res => setTimeout(res, 2000));
  console.log("Email sent")
}

// Create shipment mutation resolver

console.log("Create shipment")
sendEmail();
console.log("Done")
```

```
Create shipment
Done
Email sent
```

Evenement Asynchrone

Code non bloquant et impact délégué

```
// Create shipment mutation resolver

console.log("Create shipment")
emit("shipmentCreated")
console.log("Done")
```

Events, Event Driven Development

- Excellent moyen de découpler le métier et de produire une architecture scalable

```
// core/tracking/updateTracking.ts

emit("shipmentDeparted")
// Le tracking n'a que faire de mettre à jour les status, les tâches, les emails, etc
```

- Permettent de lancer des commandes

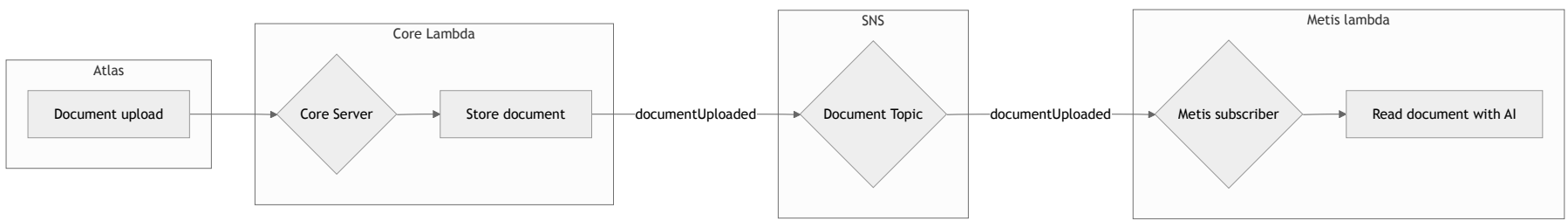
```
// Pour du découplage plus technique
on("EMAIL_SendBookingRequestAskedEmailCommand", (payload) => sendBookingRequestEmail(payload.shipmentId))

// Pour se simplifier la vie lors d'opérations manuelles
on("resyncTracking", (payload) => resyncTracking(payload.shipmentId))
```

- Nécessité de stocker les messages (Event Log) pour remonter le fil d'exécution

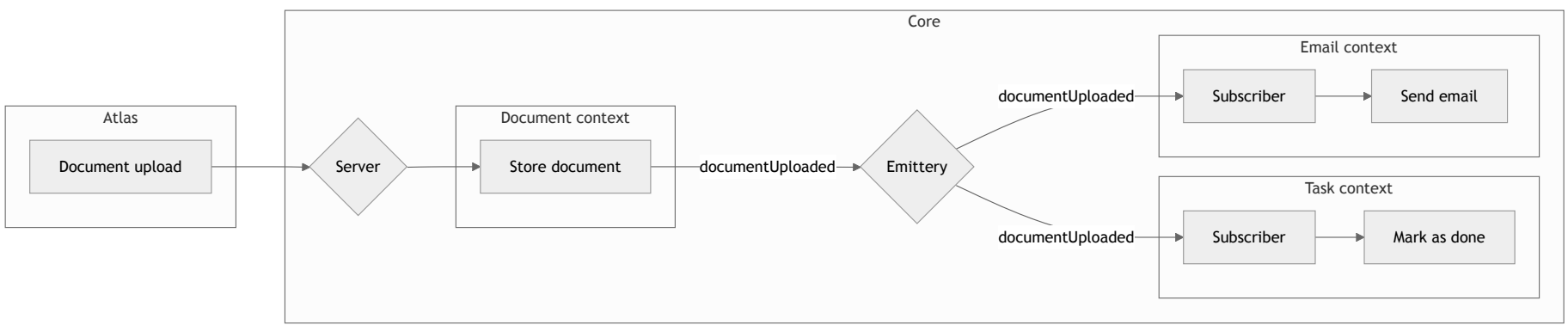
SNS

- Solution AWS
- Permet de trigger les composants d’une architecture distribuée
- Rien à implémenter
- Intégration naturelle avec l’écosystème AWS



Emittery

- Librairie JS Open-Source
- Monolithe modulaire: SNS n’a plus de sens pour tout ce qui ne regarde pas l’architecture distribuée
- Plus léger, plus rapide (pas de broker de messages, rien de distribué)
- Solution plus personnalisable et controlable



Gestion des erreurs

- Tous les messages sont sauvegardés dans une db
- En cas d’erreur, retry auto
- Rejouabilité à la main ou via Retool par exemple

Types de subscriber

- Possible de subscribe de 3 façons:
  - Async: le publisher resolve immédiatement
  - Sync: le publisher ne resolve que quand le subscriber resolve
  - Serial: Async mais les subscribers sont executés en série et non pas en parallèle

Code dives

- Core de notre implem emittery
- Exemples de publish/subscribes
- DB des messages
- Command runner et rejouabilité