

DOMAIN DRIVEN
DESIGN : TRAVAILLEZ
LES CONCEPTS EN
KATA

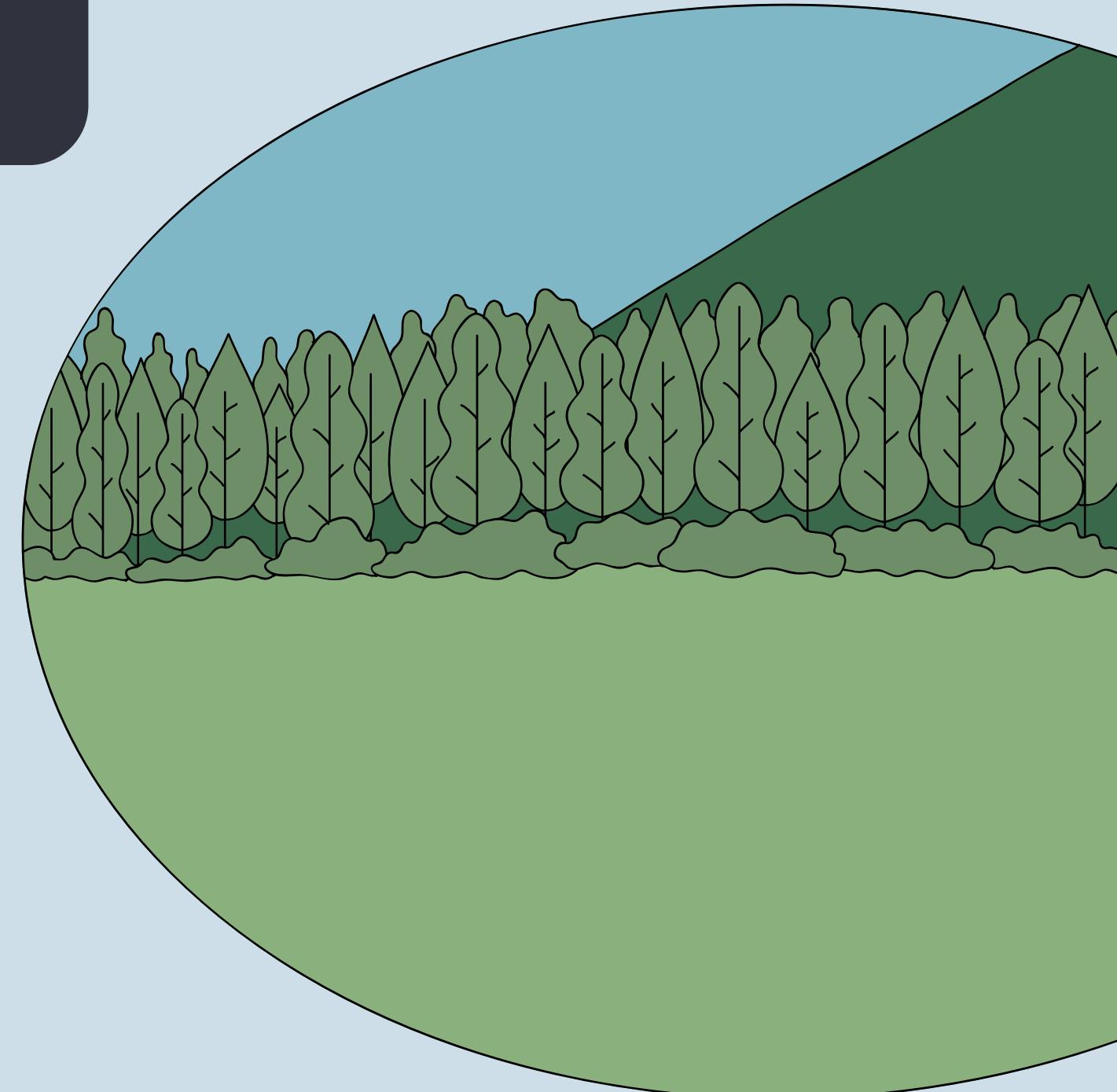
VOS animateurs



Alexandre Carbenay
TL et Formateur DDD @
Zenika



Alexandre Fruchaud
Formateur et Expert Kafka,
DDD et architecture



Aggrégat

Langage
Ubiquitaire

Context
Map

Contexte

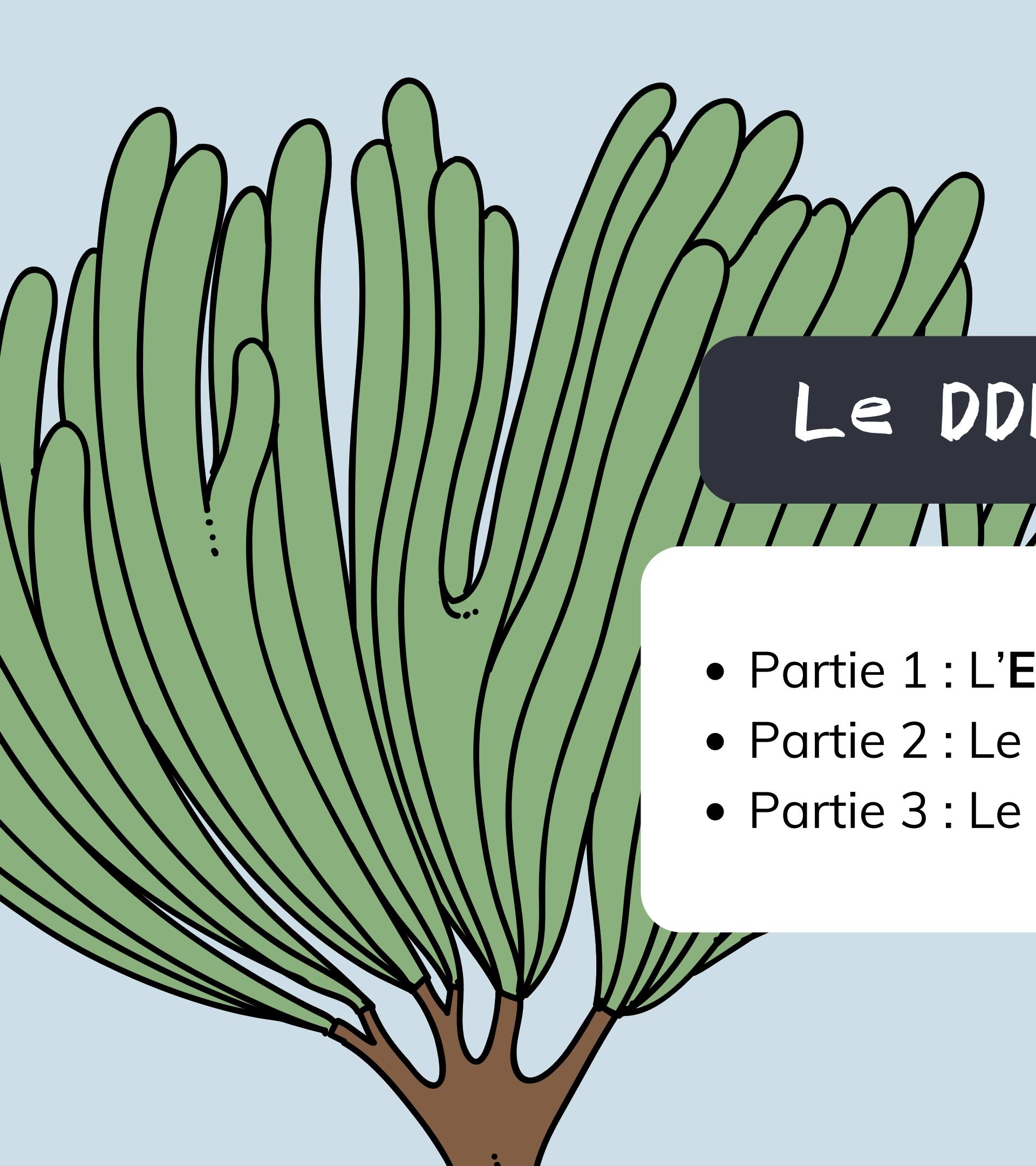
Entity

Sous-Domaine

Domain Model

Event

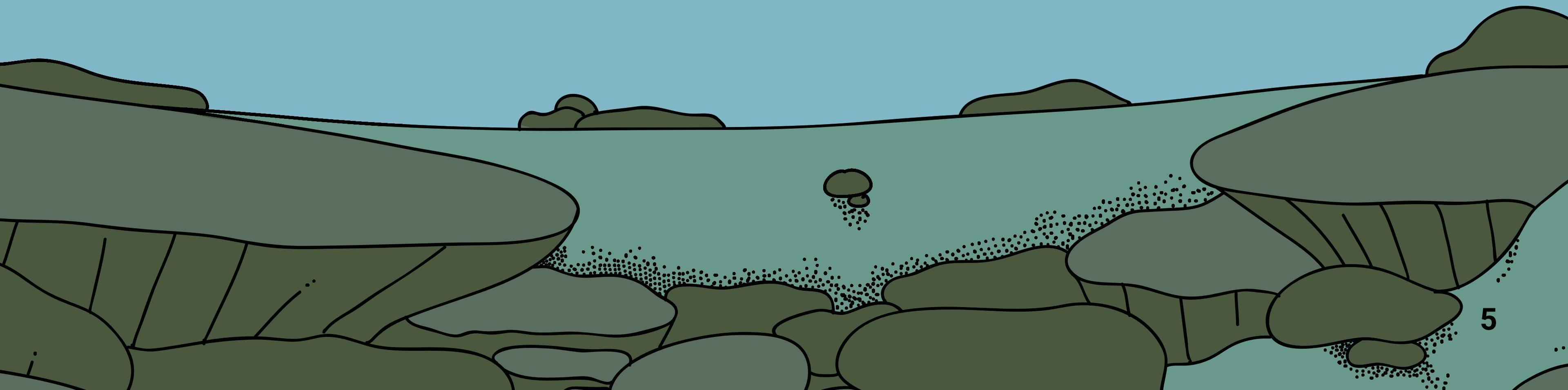
Contexte Borné



Le DDD par la pratique

- Partie 1 : L'Event Storming dans la savane
- Partie 2 : Le DDD Stratégique
- Partie 3 : Le DDD Tactique

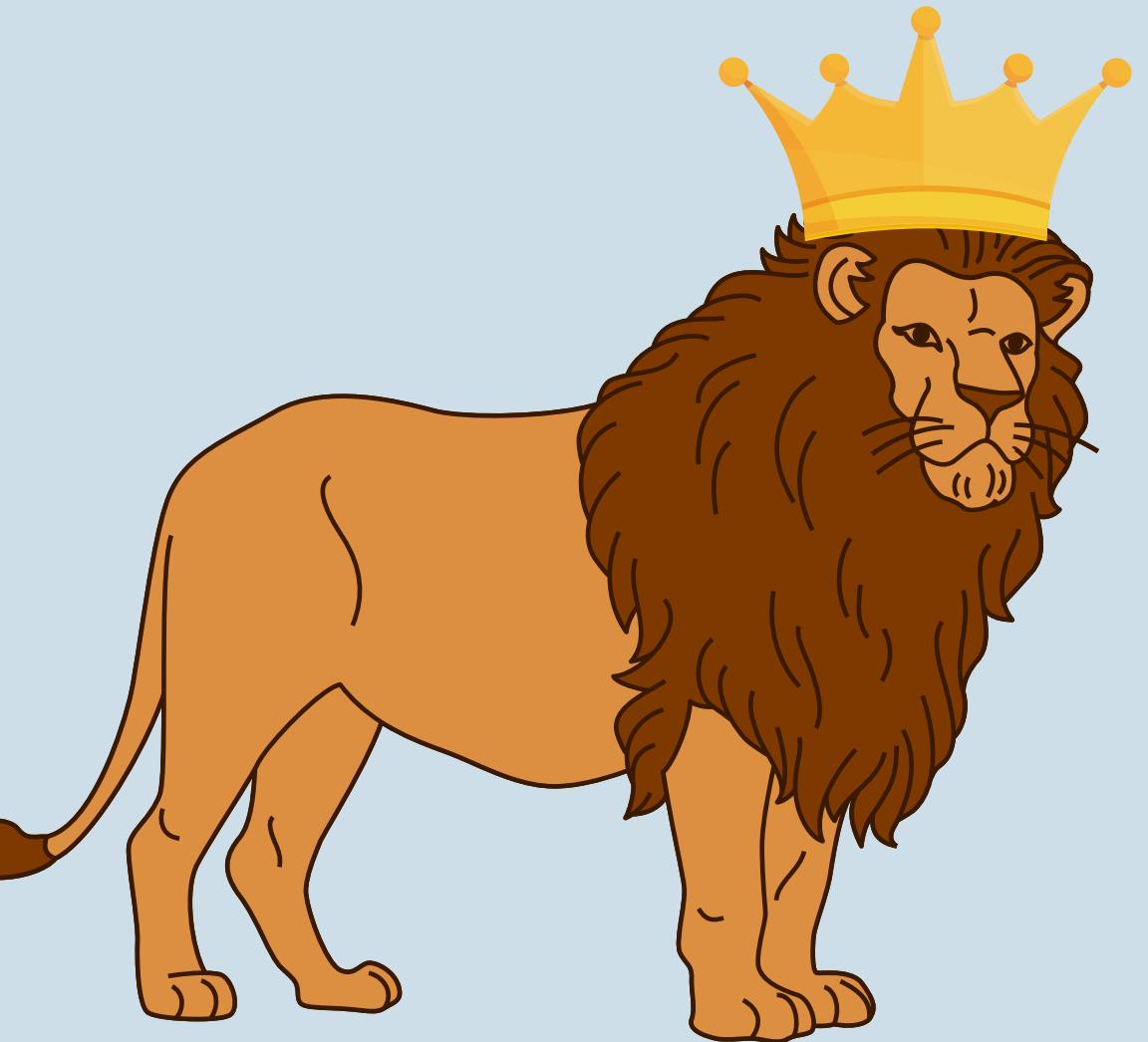
Un peu de contexte...





Un appel du roi de la savane...



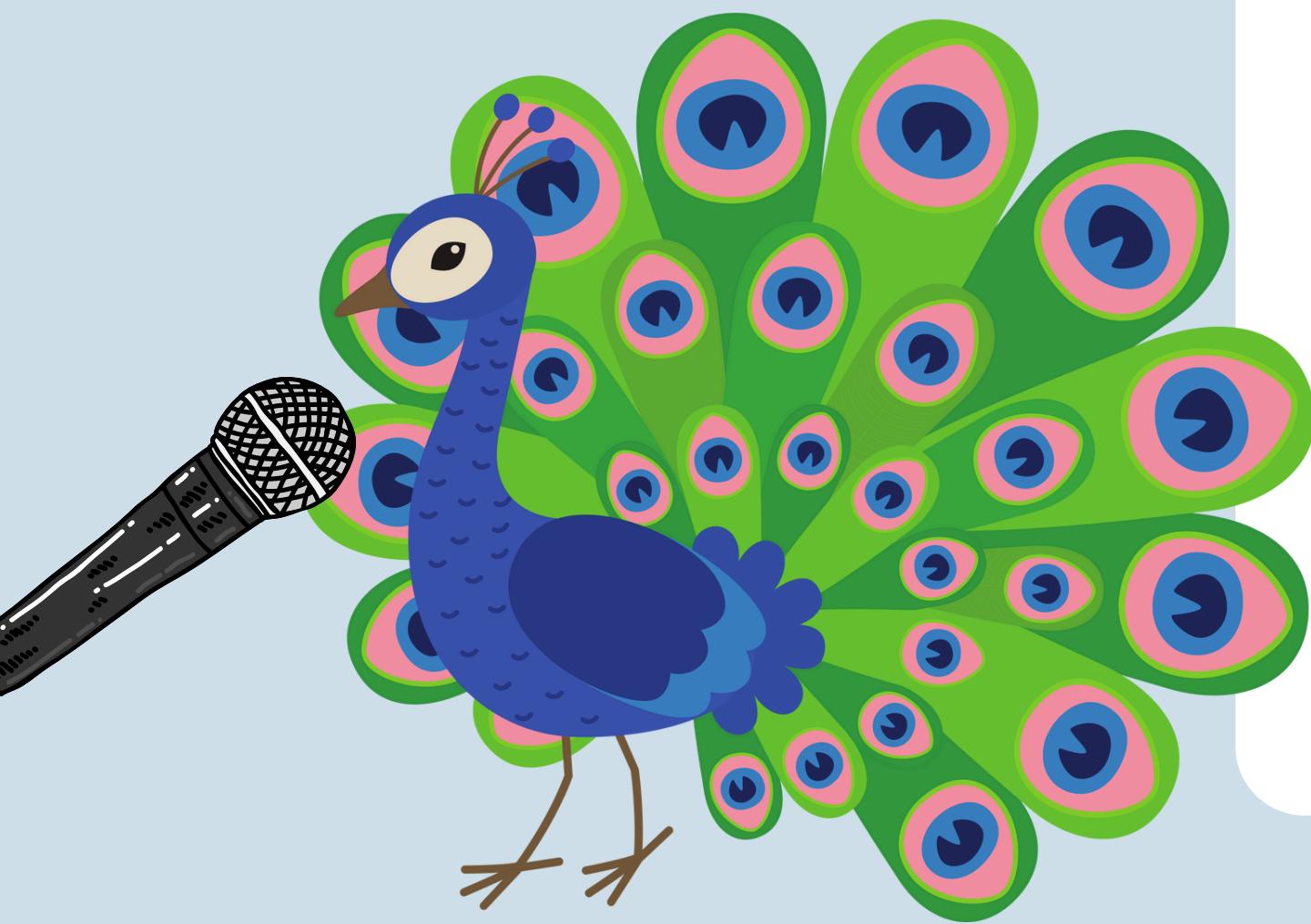


On a plein d'idées dans la savane...

Mais dès qu'on doit les implémenter, c'est la jungle !



Par exemple, on voulait construire un point d'eau...



*On a eu du mal à avoir des
retours des animaux
citoyens*



On a chargé un groupe de travail de s'organiser pour monter le projet...

Tout allait de travers ! C'était la cata...

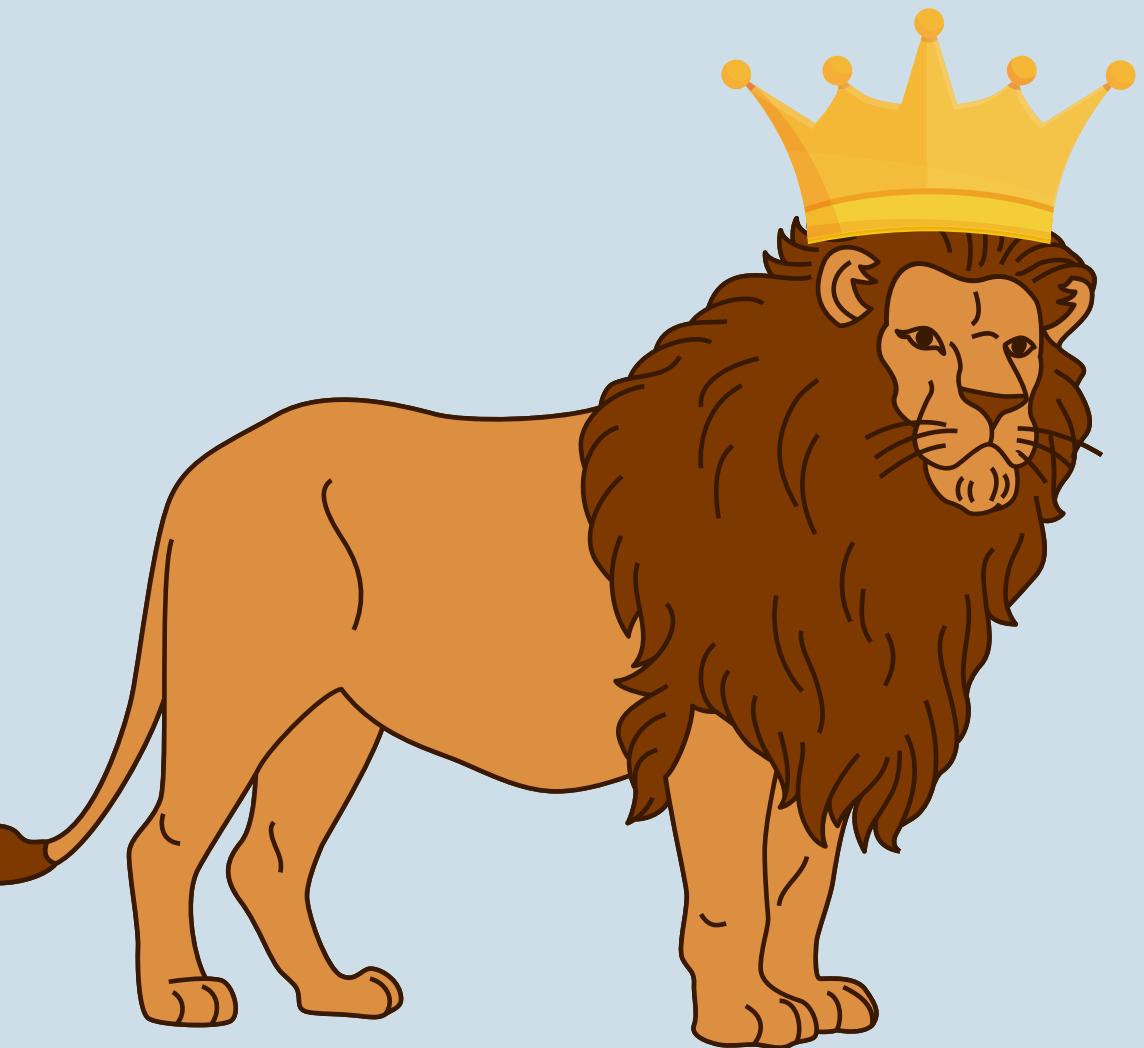




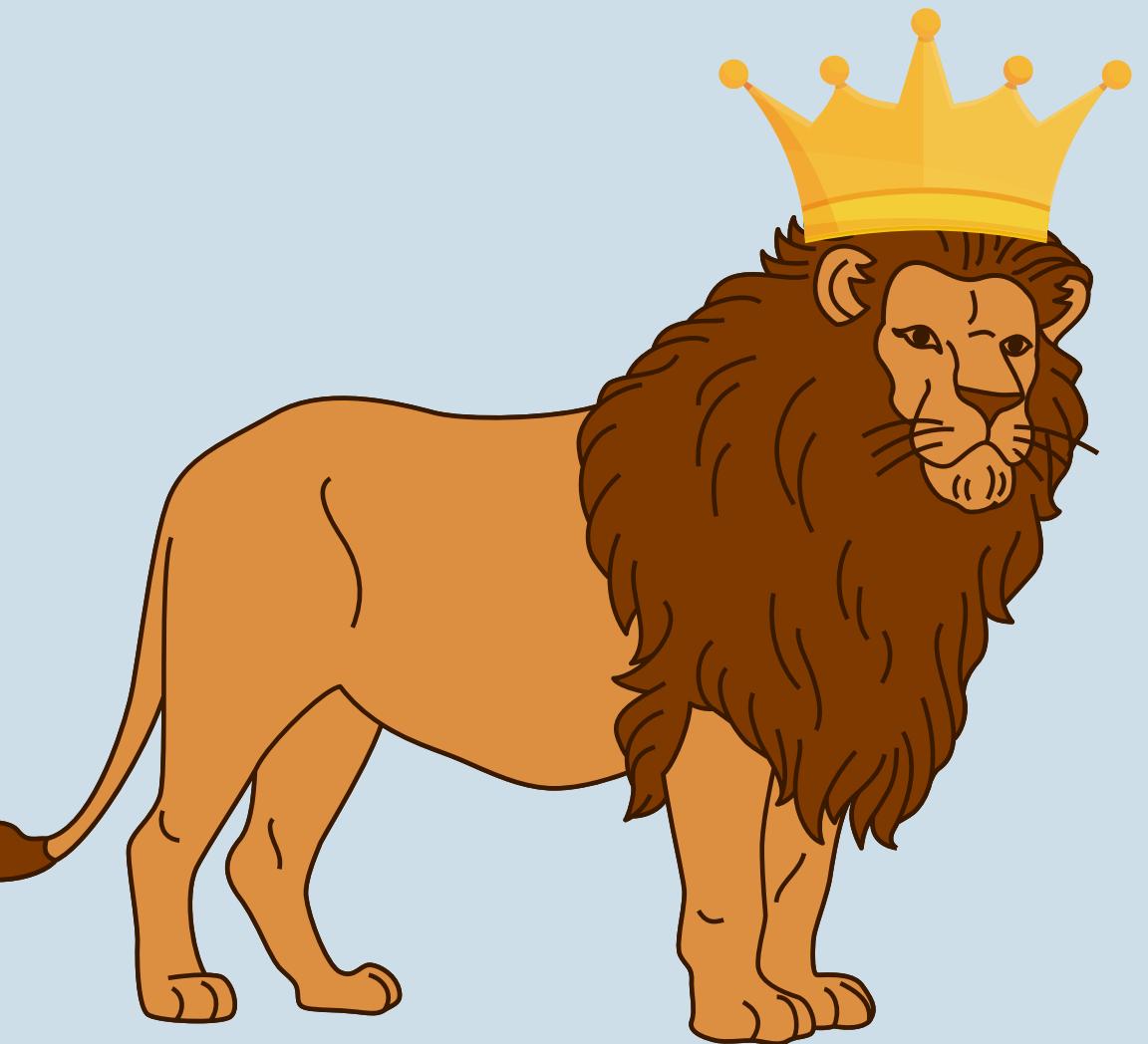
Et après, plus moyen de se souvenir qui a travaillé sur le projet. Ou quelles ont été les décisions concrètes.

Une vraie galère !





On voudrait un système qui nous permette de faire émerger des idées, les voter, et planifier les projets.



You pouvez nous construire
ça ?

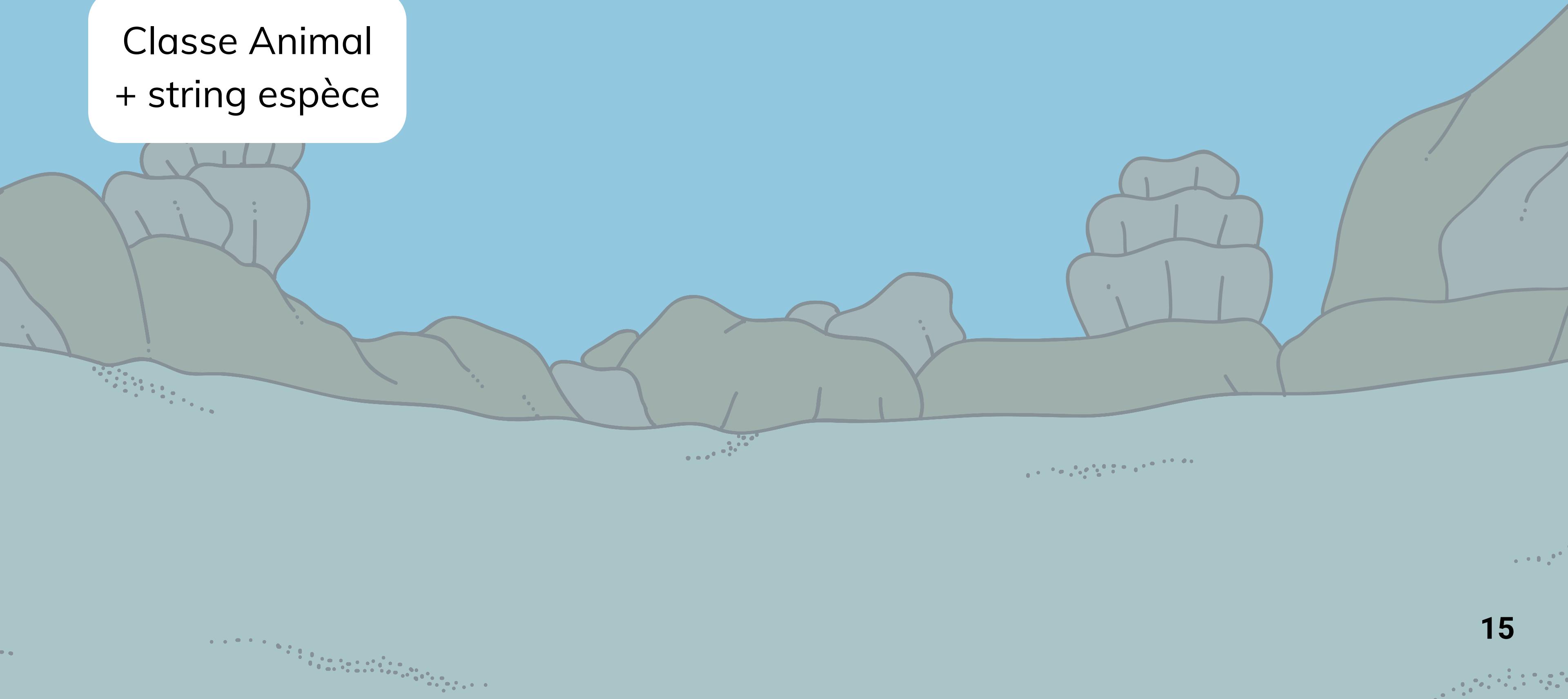


*prenons notre
casquette de codeur*



Premier essai

Classe Animal
+ string espèce



Premier essai

Classe Animal
+ string espèce

Classe Idée
+ string id
+ string type
+ string contenu
+ string[] amendements

Premier essai

Classe Animal
+ string espèce

Classe Idée
+ string id
+ string type
+ string contenu
+ string[] amendements

Classe Projet
+ string id
+ string description
+ date dateDebut
+ date dateFin

Premier essai

Classe Animal
+ string espèce

Classe Idée
+ string id
+ string type
+ string contenu
+ string[] amendements

Classe Projet
+ string id
+ string description
+ date dateDebut
+ date dateFin

Classe GroupeTravail
+ Animal[] membres
+ Projet projet

Premier essai

Classe Animal
+ string espèce

Classe Idée
+ string id
+ string type
+ string contenu
+ string[] amendements

Classe Projet
+ string id
+ string description
+ date dateDebut
+ date dateFin

Classe GroupeTravail
+ Animal[] membres
+ Projet projet

Classe SessionTravail
+ string id
+ Animal[] presents
+ date date

Premier essai

Classe Animal
+ string espèce

Classe Idée
+ string id
+ string type
+ string contenu
+ string[] amendements

Classe Projet
+ string id
+ string description
+ date dateDebut
+ date dateFin

Classe GroupeTravail
+ Animal[] membres
+ Projet projet

Classe SessionTravail
+ string id
+ Animal[] presents
+ date date

Classe Proposition
+ string id
+ string projectId
+ Details[] details

Premier essai

Classe Animal
+ string espèce

Classe Idée
+ string id
+ string type
+ string contenu
+ string[] amendements

Classe Projet
+ string id
+ string description
+ date dateDebut
+ date dateFin

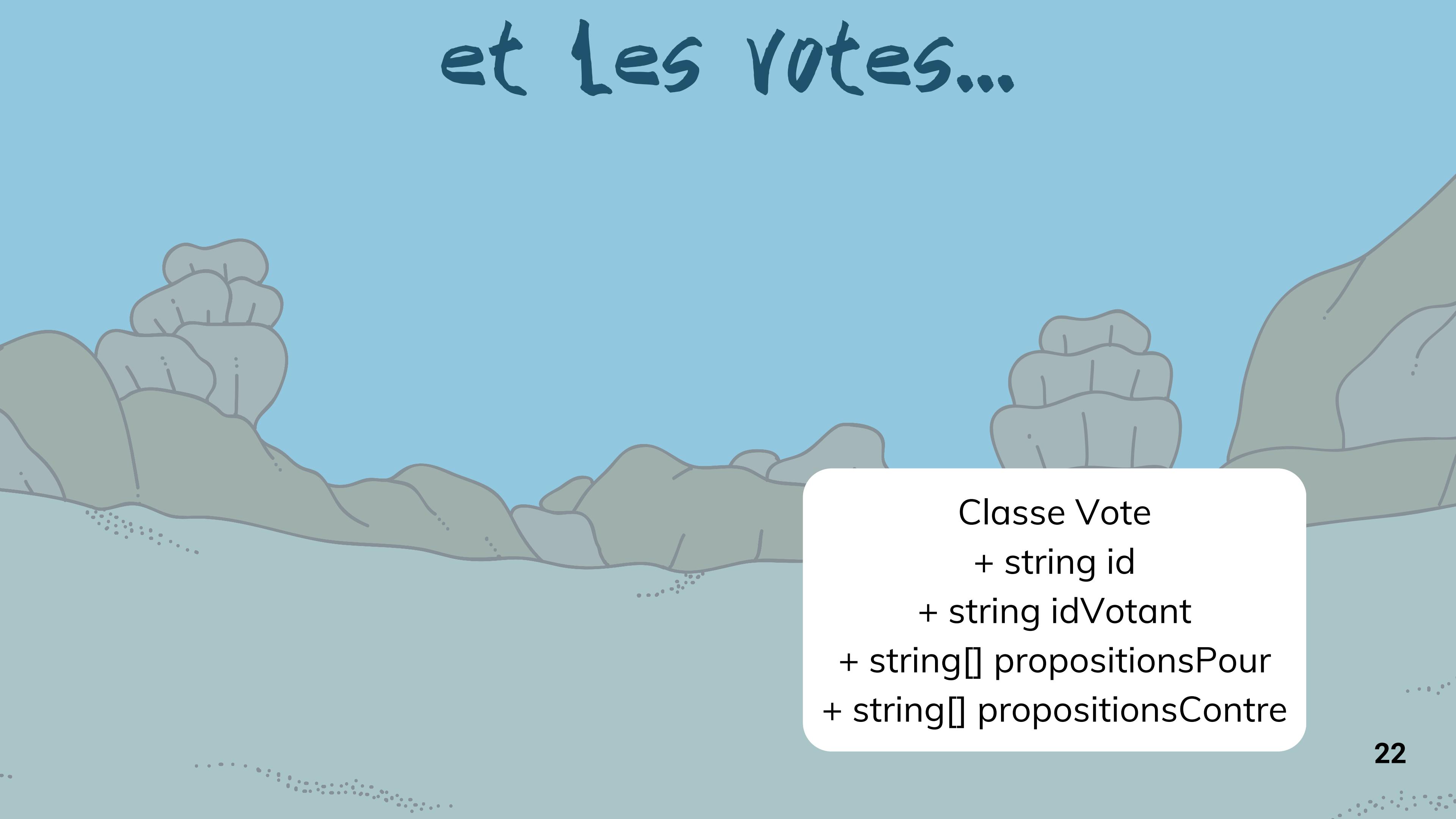
Classe GroupeTravail
+ Animal[] membres
+ Projet projet

Classe SessionTravail
+ string id
+ Animal[] presents
+ date date

Classe Proposition
+ string id
+ string projectId
+ Details[] details

Classe Details
+ string type
+ string contenu

et les votes...

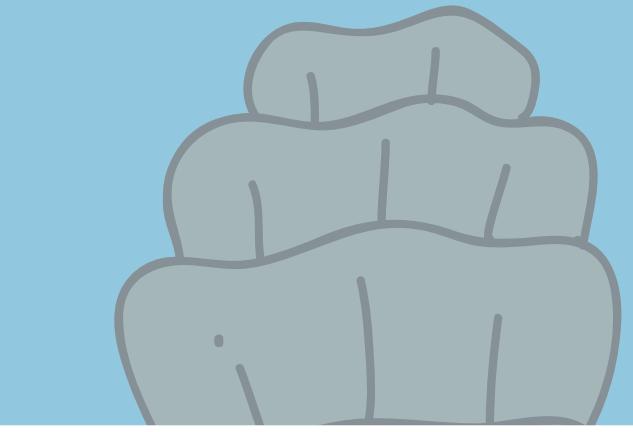


Classe Vote
+ string id
+ string idVotant
+ string[] propositionsPour
+ string[] propositionsContre

et les votes...

Classe SessionVote

- + string id
- + string projectId
- + date dateVote
- + heure debut
- + heure fin



Classe Vote

- + string id
- + string idVotant
- + string[] propositionsPour
- + string[] propositionsContre

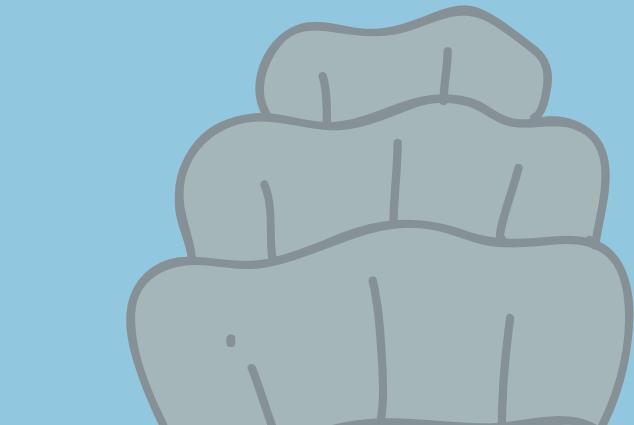
et les votes...

Classe SessionVote

- + string id
- + string projectId
- + date dateVote
- + heure debut
- + heure fin

Classe Votant

- + string id
- + string espece
- + bool aVote



Classe Vote

- + string id
- + string idVotant
- + string[] propositionsPour
- + string[] propositionsContre

et les votes...

Classe SessionVote

- + string id
- + string projectId
- + date dateVote
- + heure debut
- + heure fin

Classe Votant

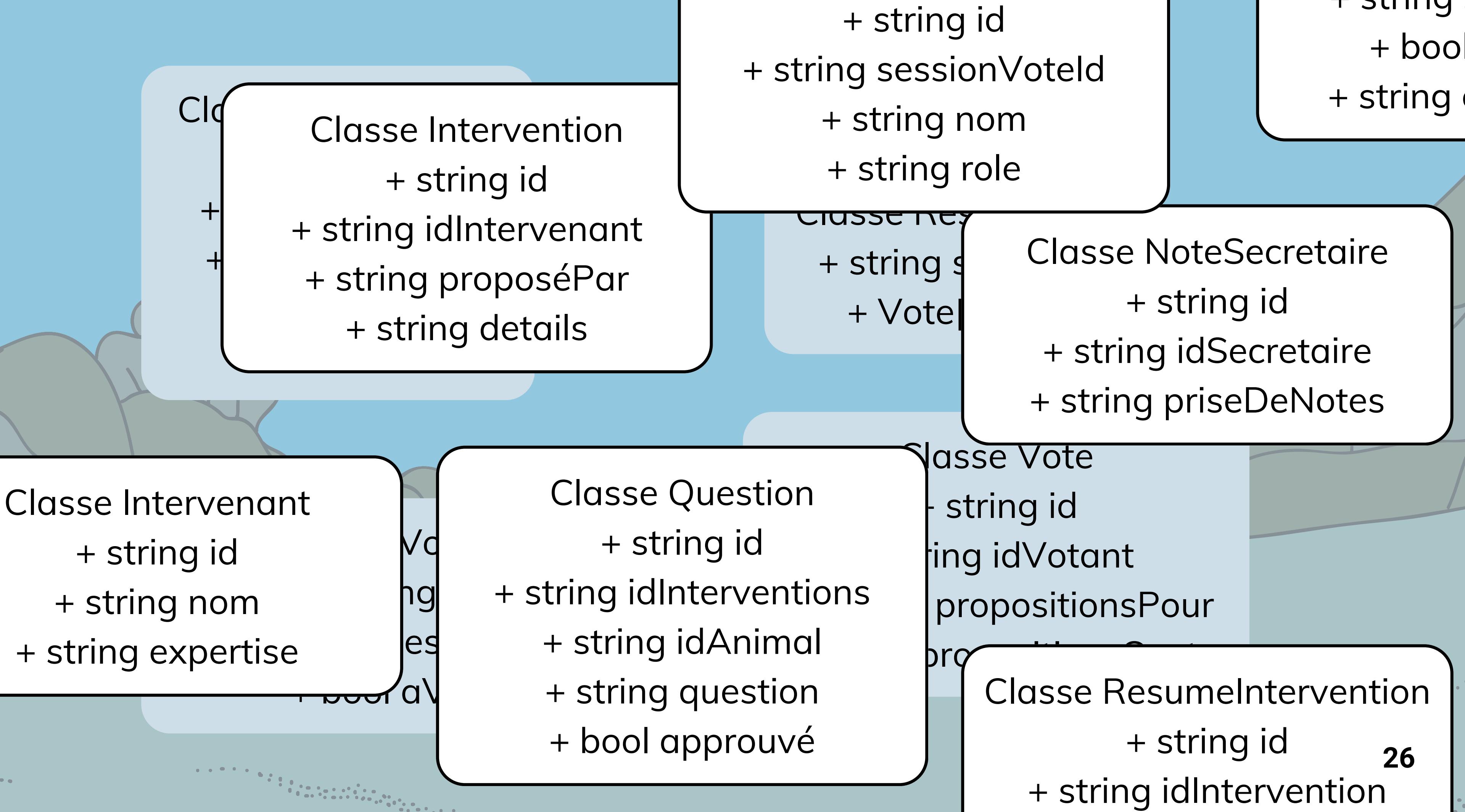
- + string id
- + string espece
- + bool aVote

Classe ResultatVote

- + string sessionId
- + Vote[] votes

Classe Vote

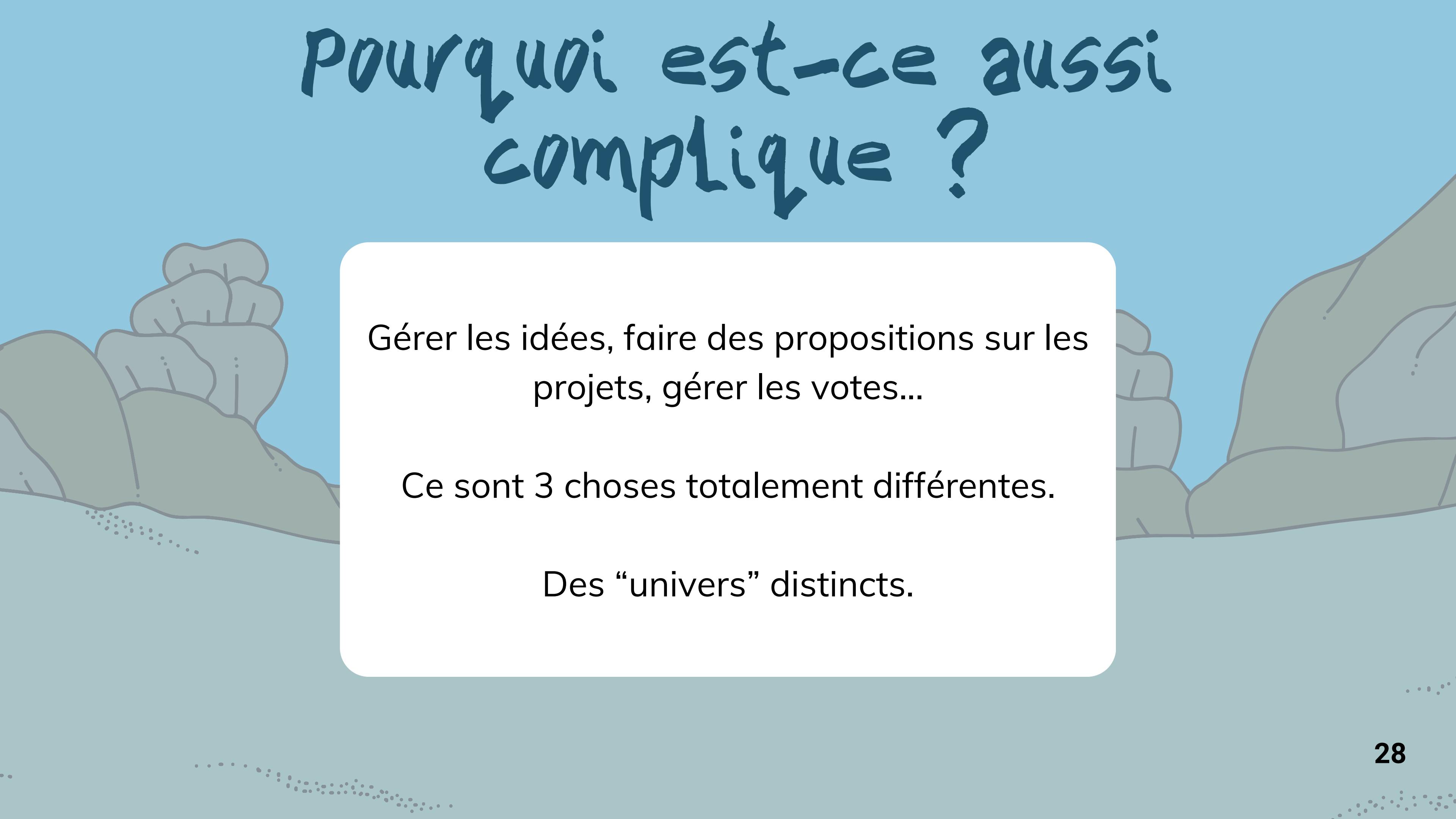
- + string id
- + string idVotant
- + string[] propositionsPour
- + string[] propositionsContre





Au Secours !!!

Pourquoi est-ce aussi complique ?

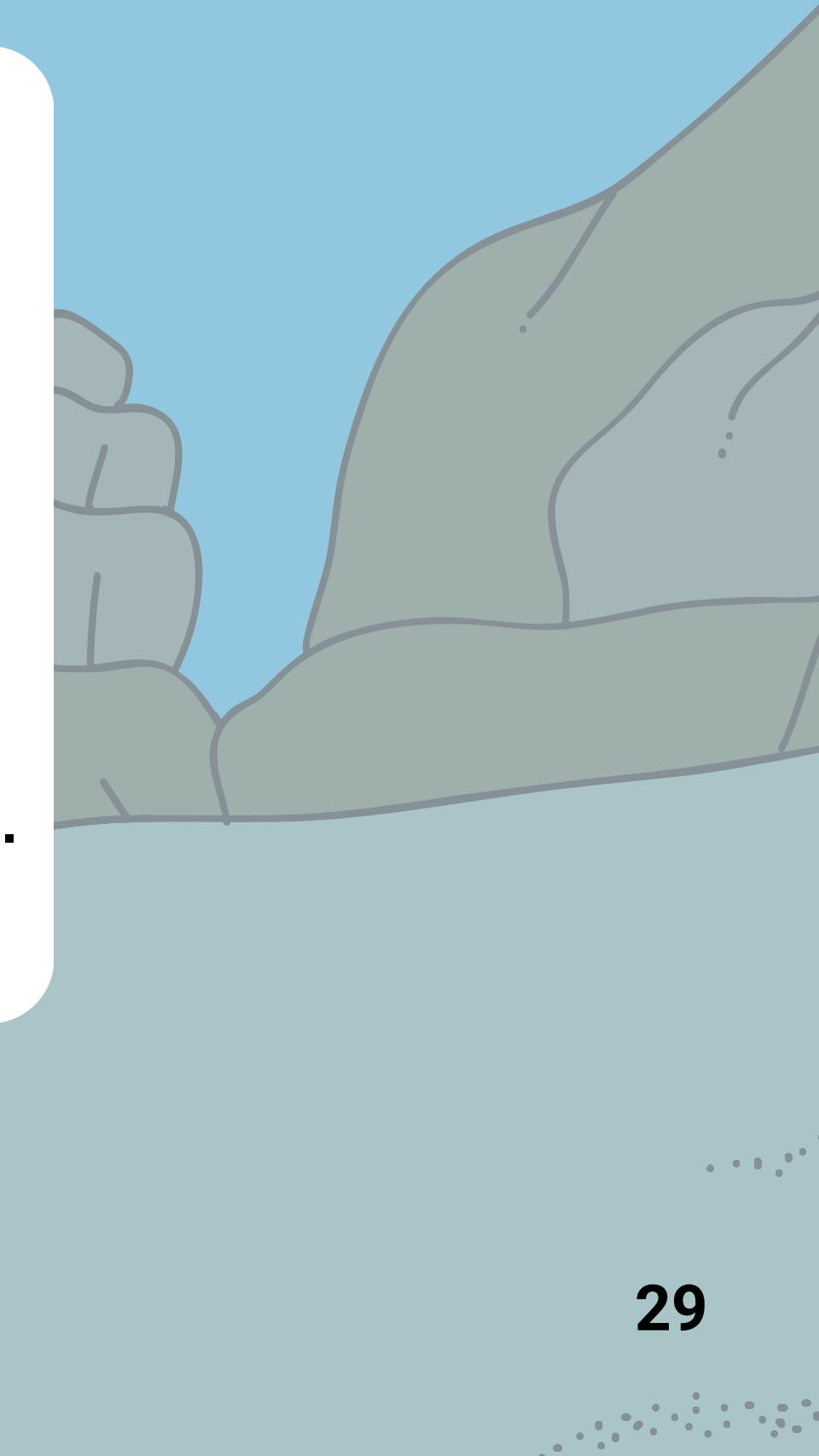


Gérer les idées, faire des propositions sur les projets, gérer les votes...

Ce sont 3 choses totalement différentes.

Des “univers” distincts.

Des univers distincts



Faire des propositions sur les projets, c'est gérer un groupe de travail qui réfléchit et prend des décisions.

Gérer une session de vote, c'est garantir une certaine transparence, un respect des règles...

Des univers distincts

Idéation

Groupe de travail

Vote

Des univers distincts

Idéation

Avoir des projets
partagés et
représentatifs

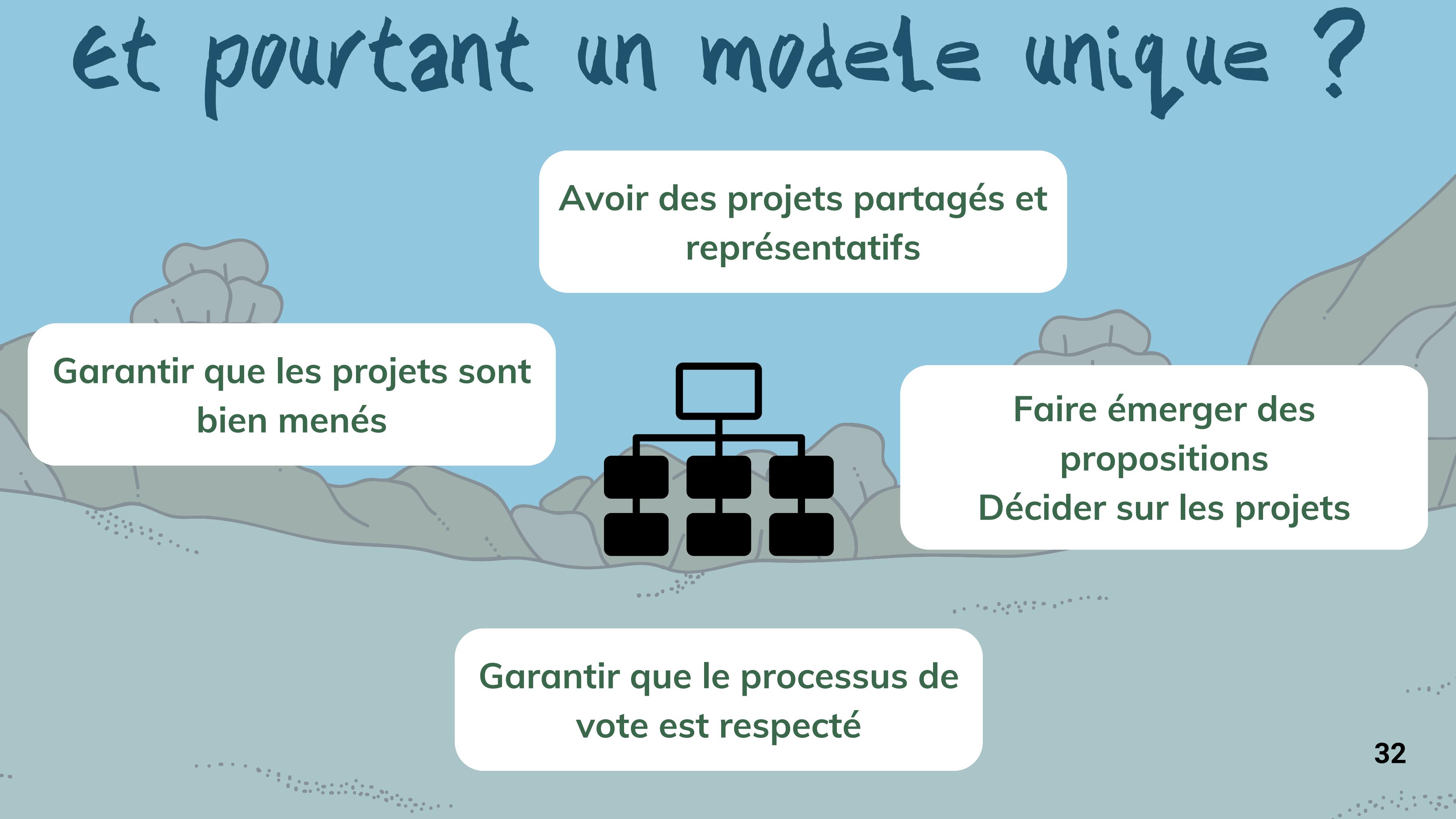
Groupe de travail

Faire émerger
des propositions
Décider sur les
projets

Vote

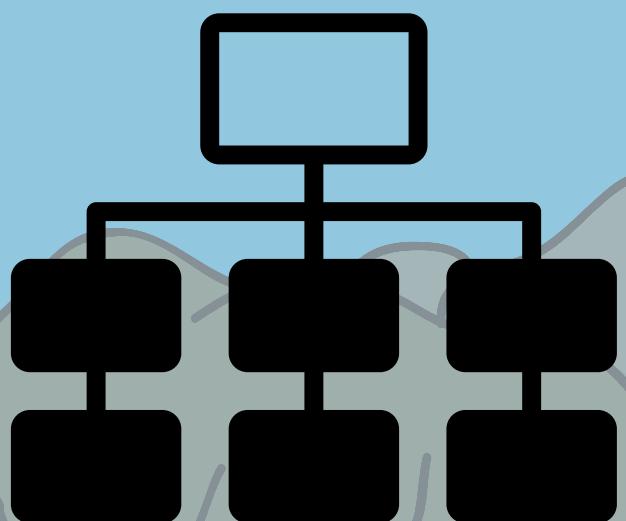
Garantir que le
processus de
vote est respecté

Et pourtant un modèle unique ?



Avoir des projets partagés et représentatifs

Garantir que les projets sont bien menés



Faire émerger des propositions
Décider sur les projets

Garantir que le processus de vote est respecté

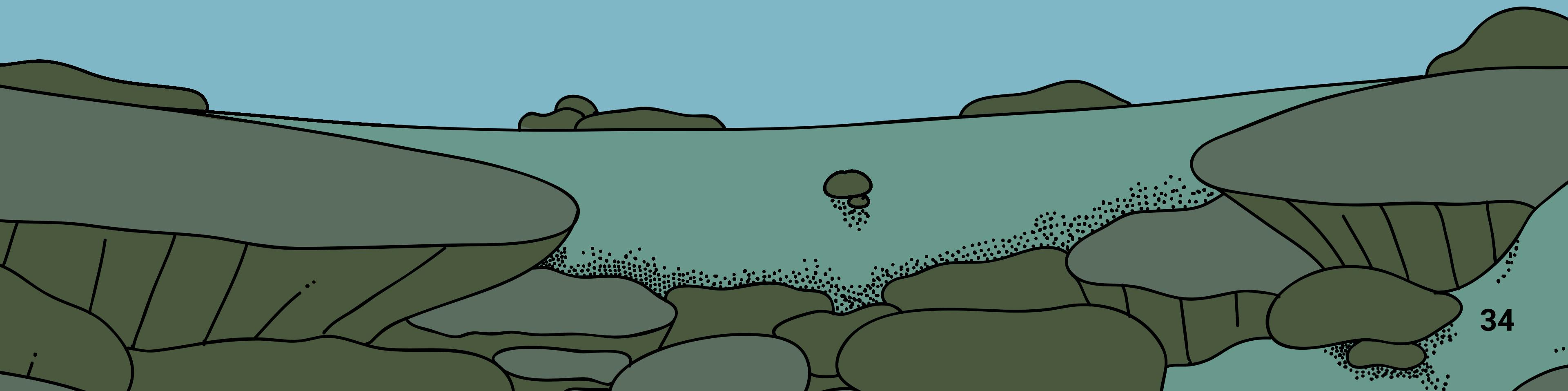
Allons-y

Pour gérer cette complexité, on va utiliser le DDD.
Notre principal outil : le découpage !



Nous allons donc séparer notre système en différents modules avec leurs propres responsabilités.

Event Storming



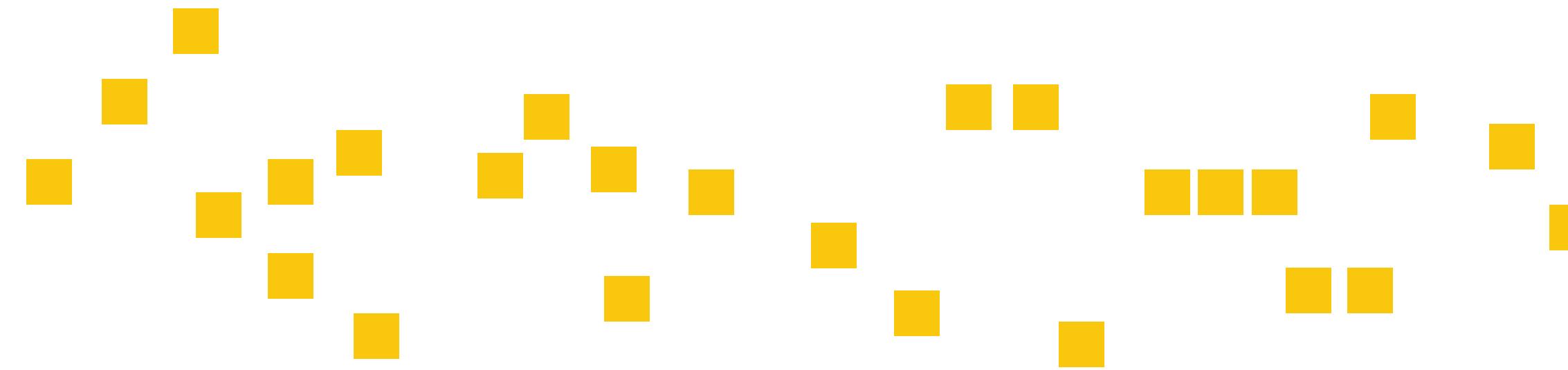




Demo

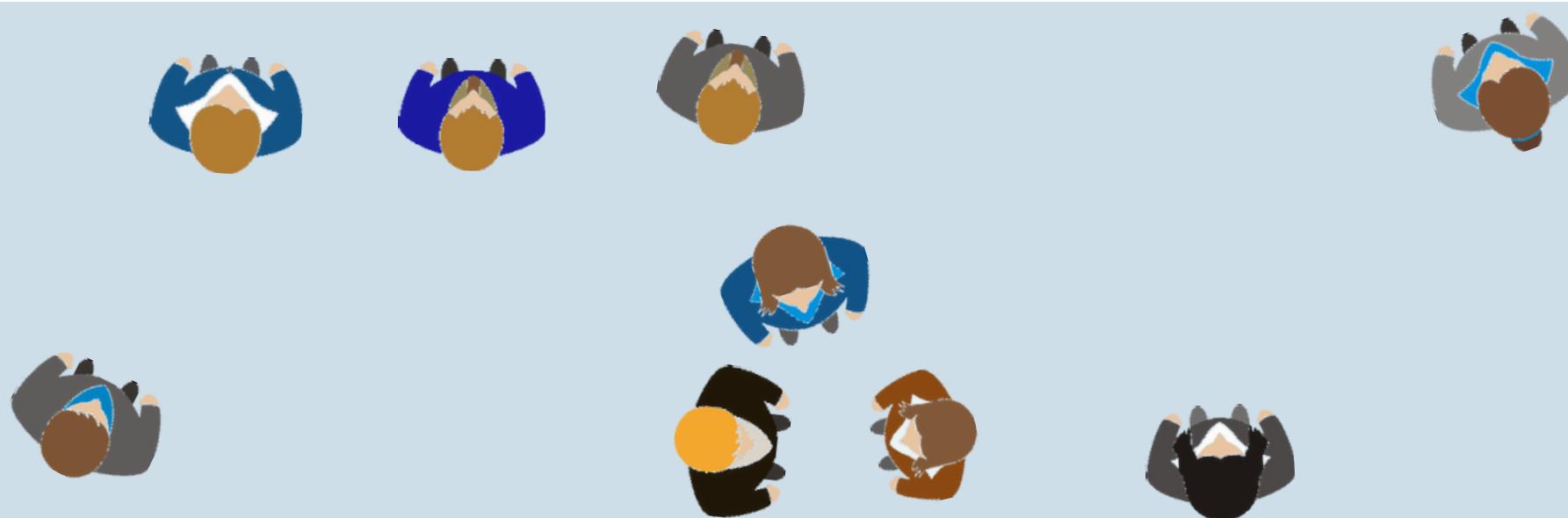
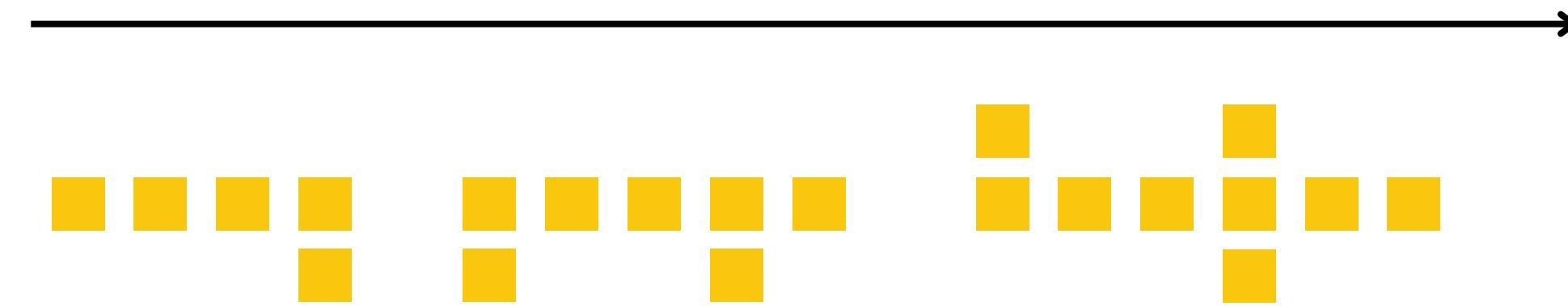
On liste les événements

Événement dans le passé

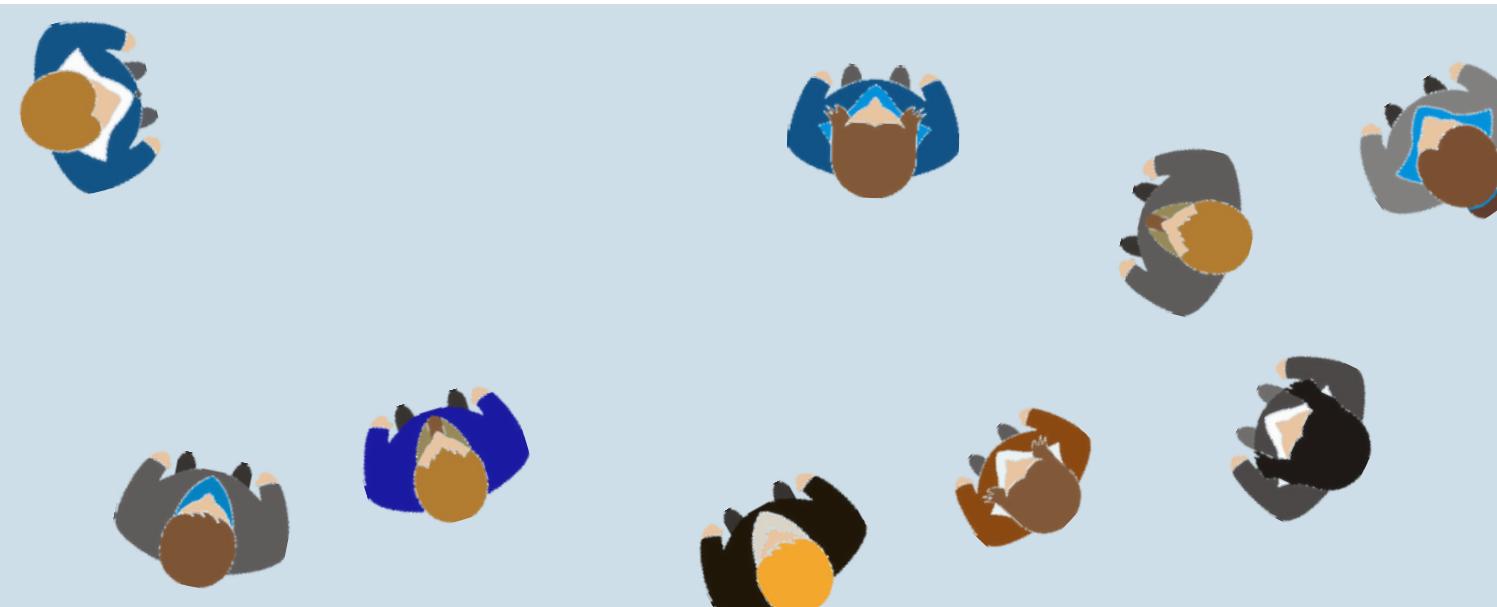
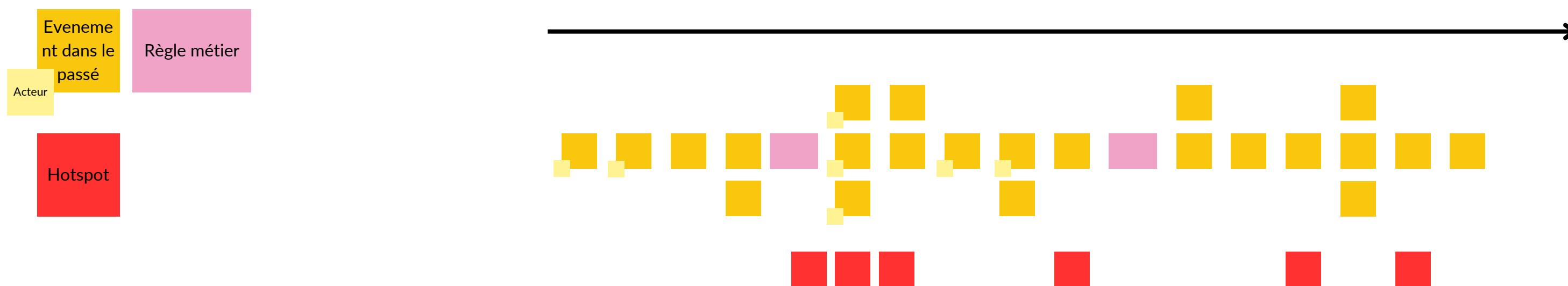


On organise les événements

Evenement dans le passé



On identifie les acteurs, règles métier et hotspots



A VOS post-it !

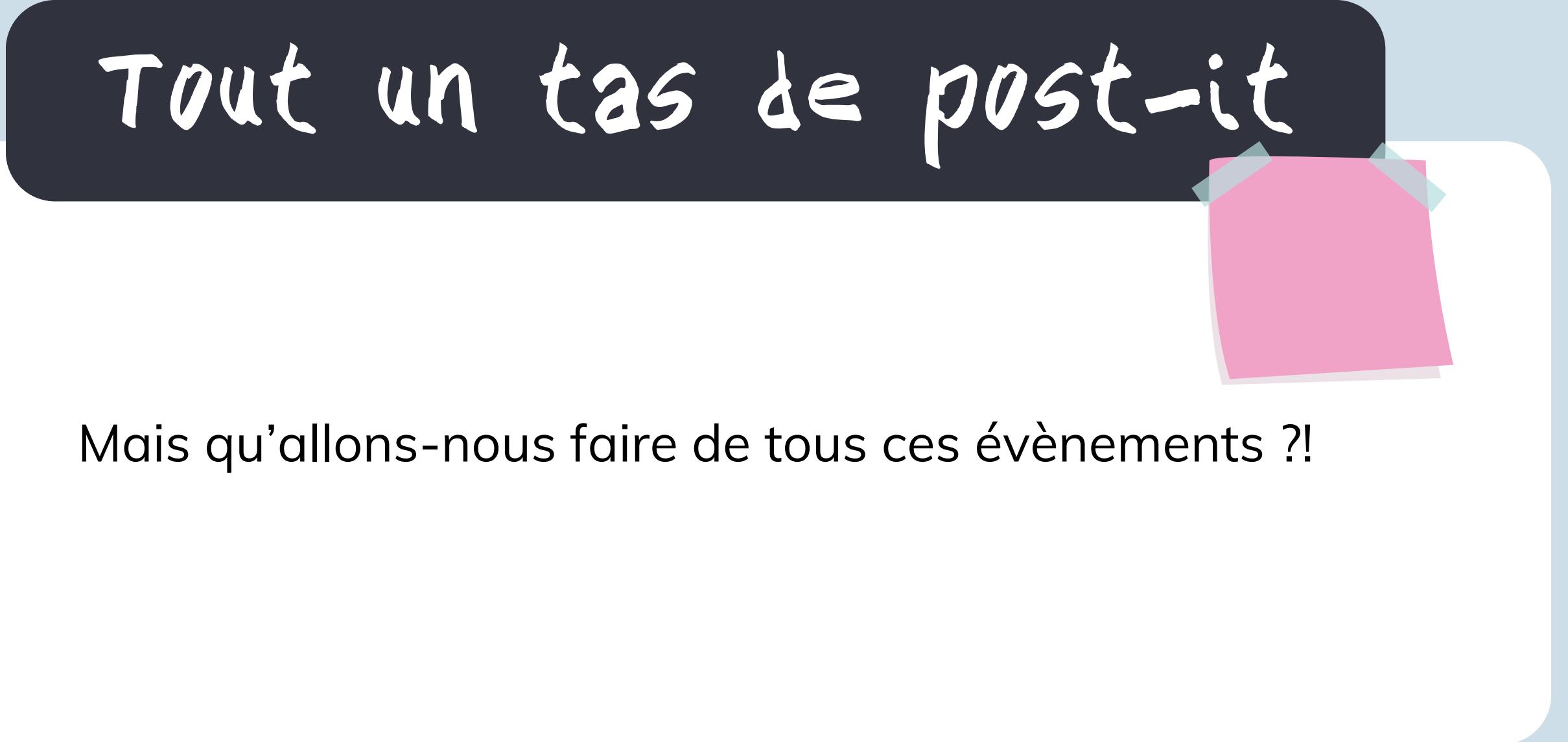
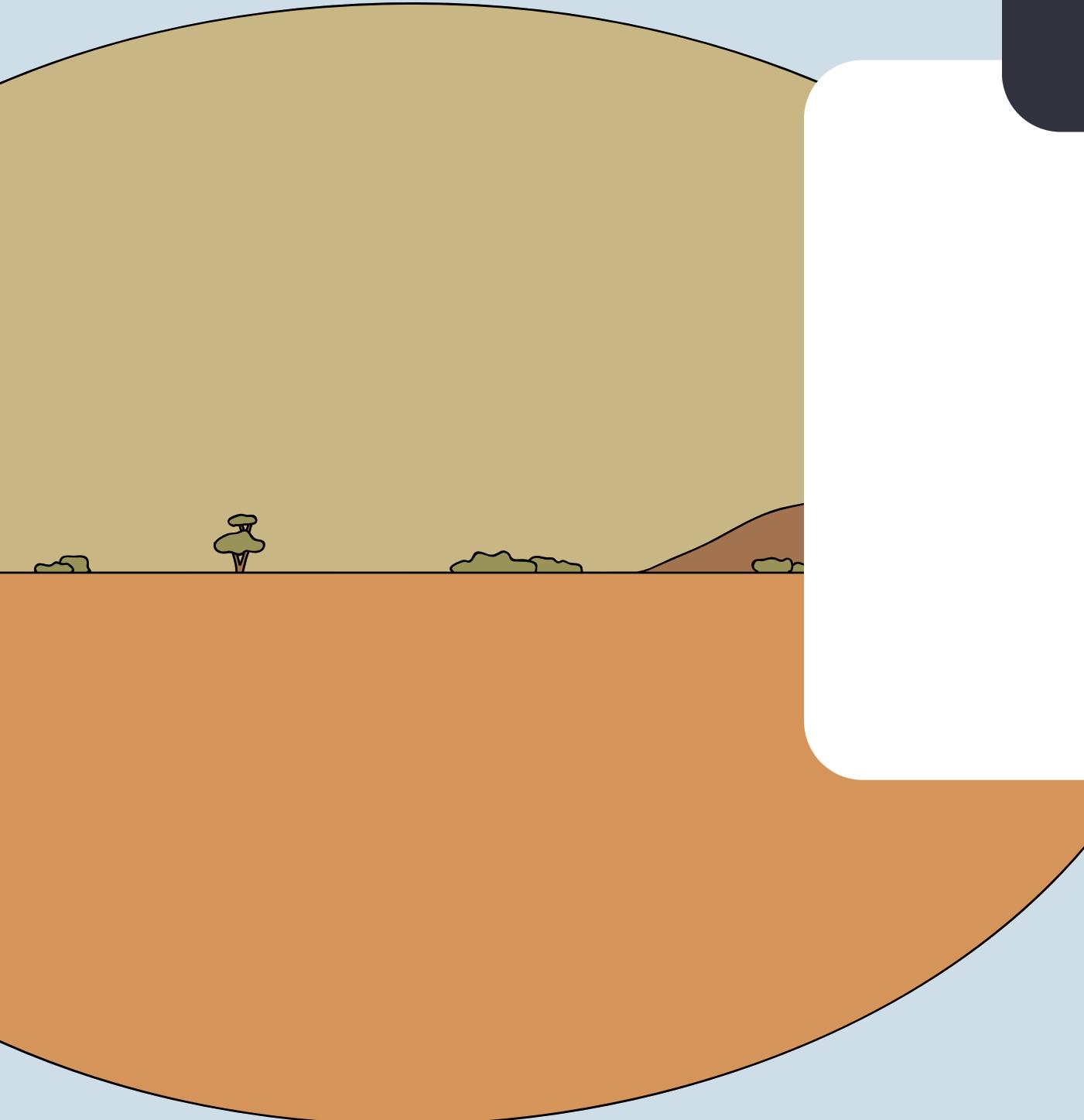
- 2 Groupes GAUCHE et DROITE
- 1 animateur par groupe
- 40 minutes pour faire un Event Storming
- Prenez un feuillet explicatif par personne

A vos post-it !

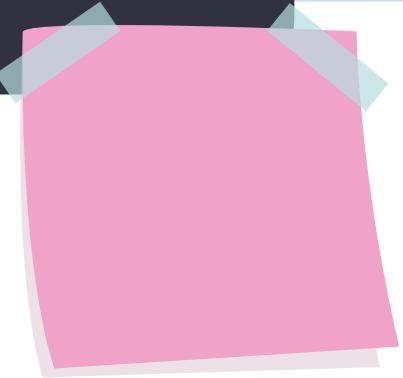




Debrief



Tout un tas de post-it



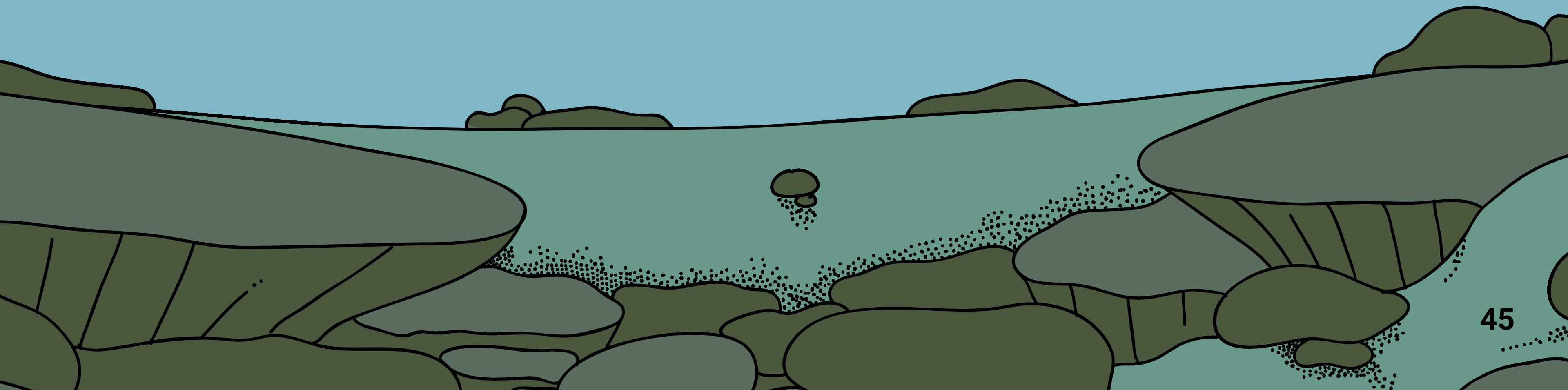
Mais qu'allons-nous faire de tous ces évènements ?!



C'est la pause !!!

5 minutes

DDD Strategique



Rappel

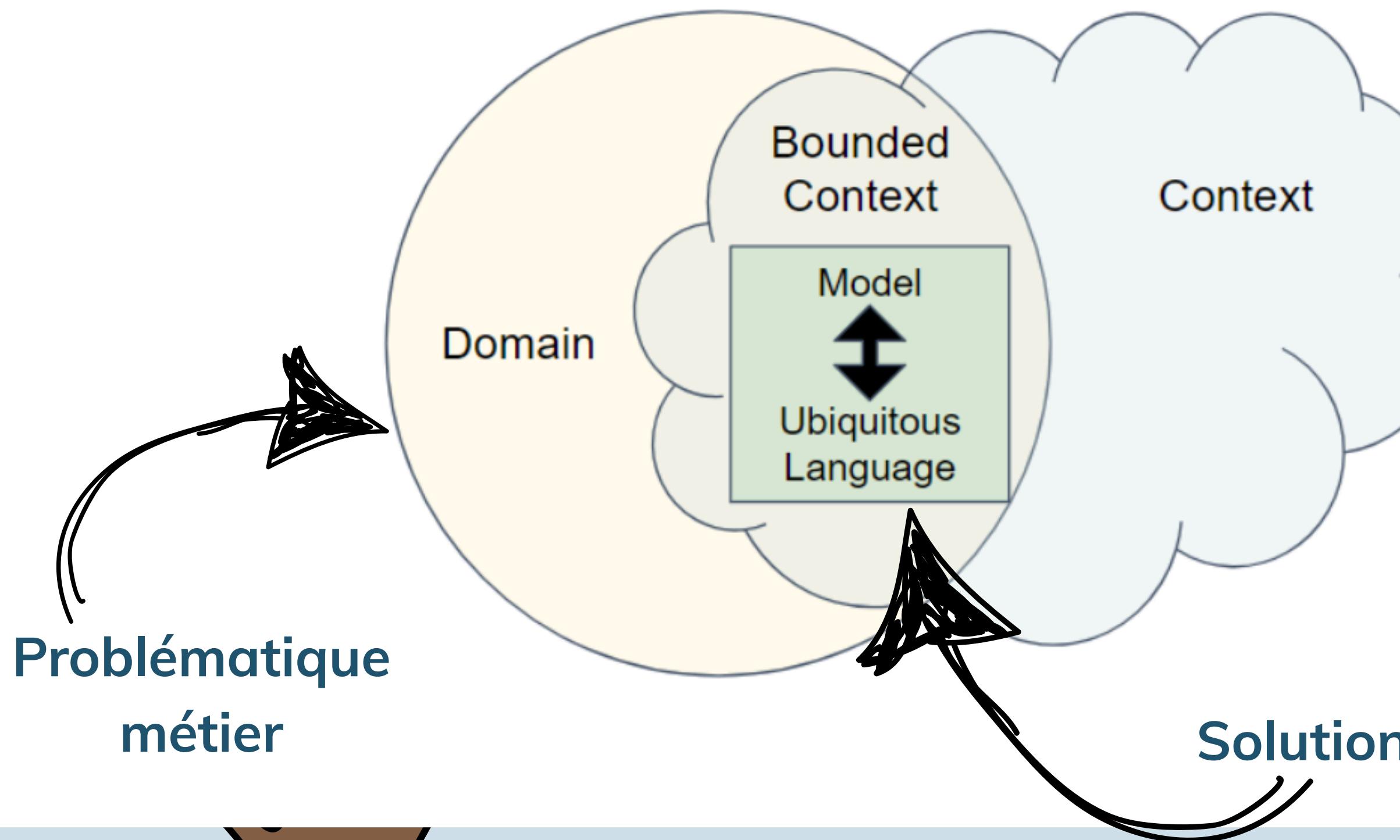
Le modèle unique n'est pas adapté - trop complexe.

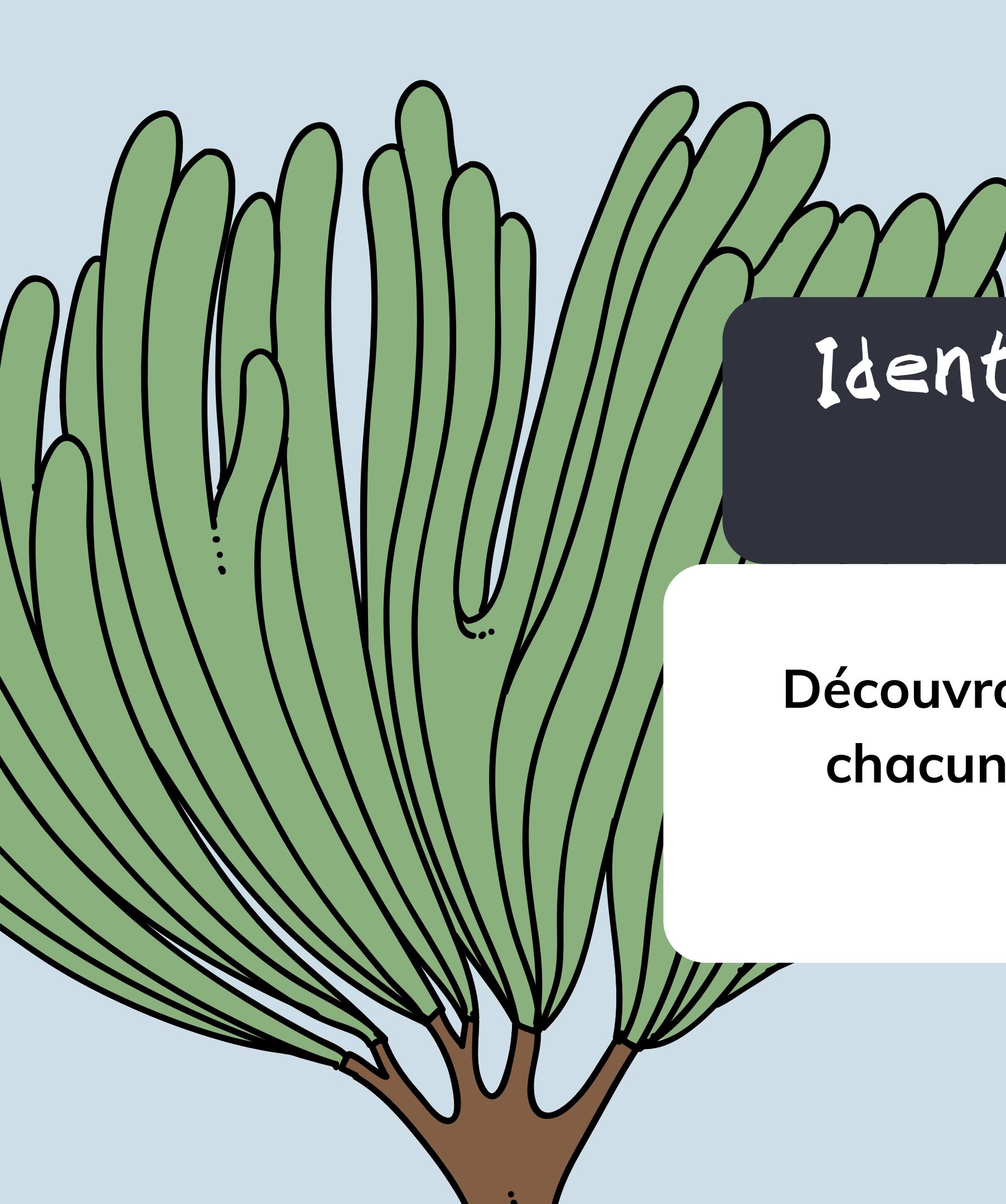
Chaque problématique a besoin d'un modèle adapté.

Identifions les différents sous-domaines qui correspondent aux problématiques.

1 problématique = 1 sous-domaine = 1 modèle qui y répond

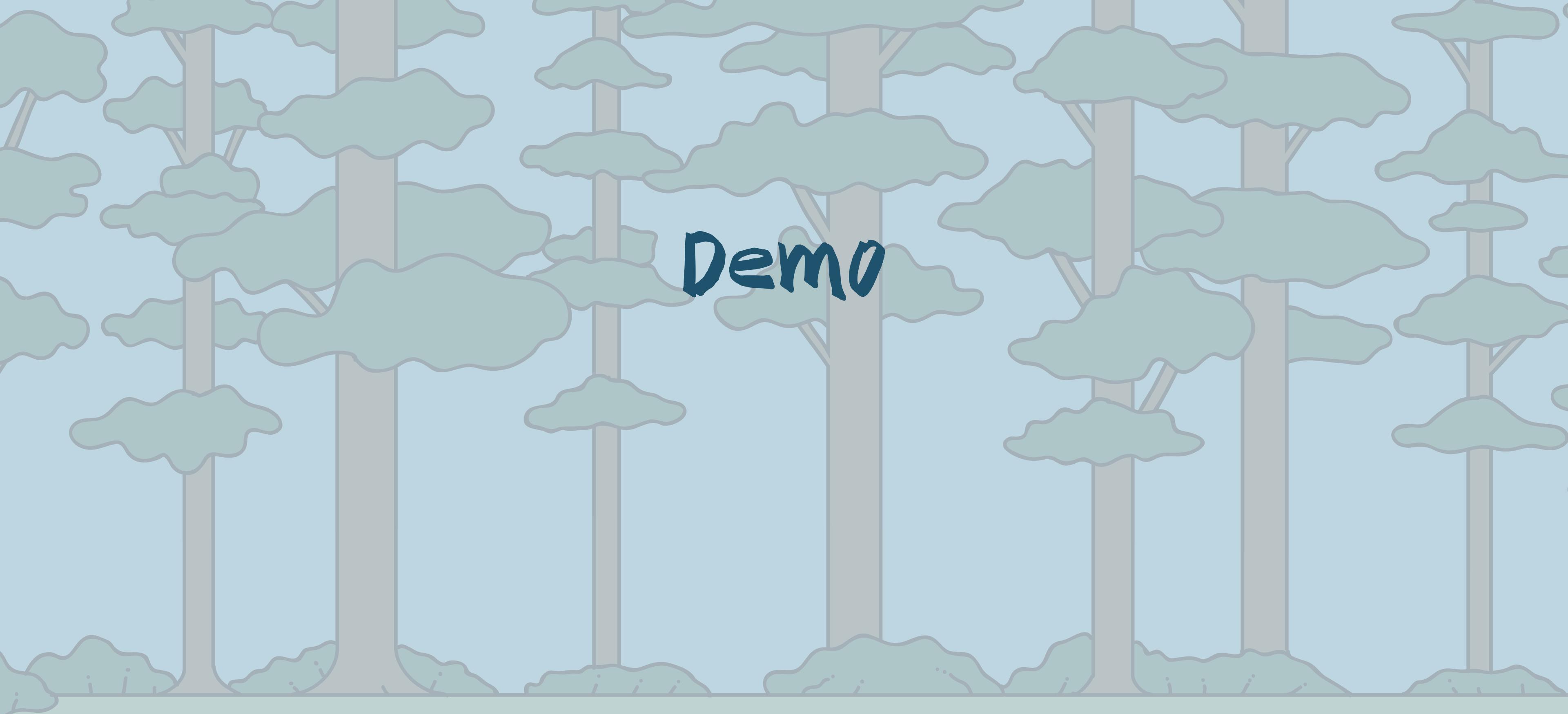
Concepts stratégiques





Identification des sous-domaines

Découvrons les sous-domaines qui auront chacun leur propre modèle à l'aide des évènements pivots.



Demo

NOS SOUS-DOMAINES

Idéation

Constitution du
groupe de travail

Groupe de travail

Vote

Journal de
décision

Mais se valent-ils tous ?



Est-ce que tous les sous-domaines sont aussi importants les uns que les autres ?

Mais se valent-ils tous ?

Est-ce que tous les sous-domaines sont aussi importants les uns que les autres ?

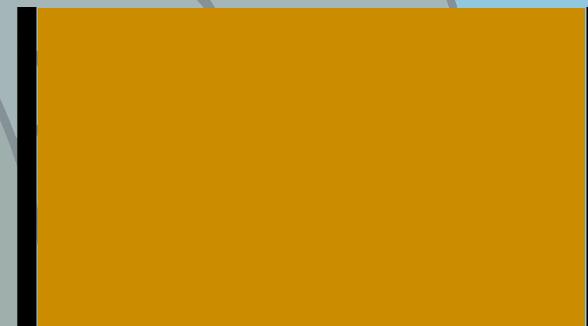
Si on veut développer le système, comment les prioriser ?

Lesquels apportent le plus de valeur métier à court terme ?



CORE

- Apporte le plus de valeur
- Différenciation Stratégique



SUPPORTING

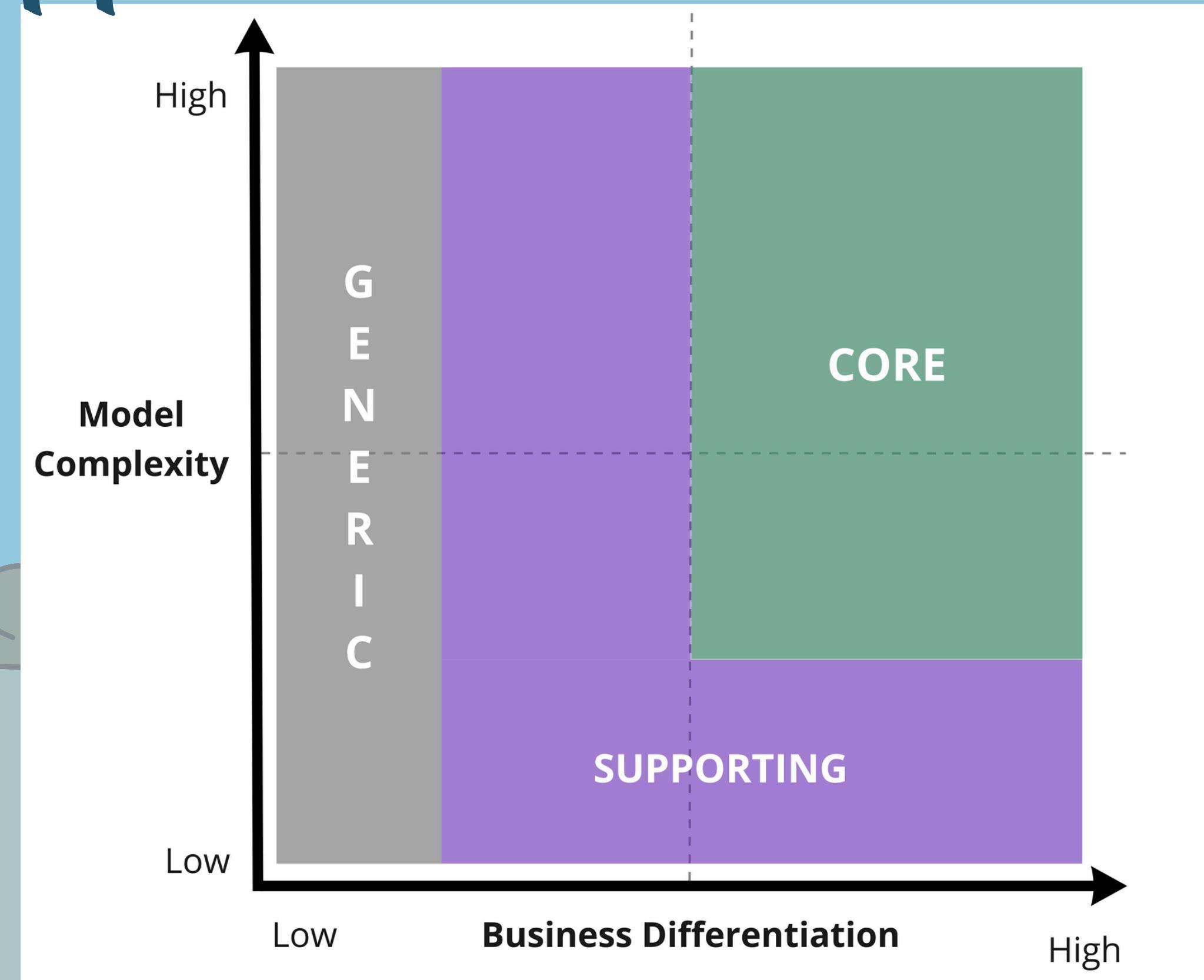
- Soutiennent le core
- Mais pas prioritaires

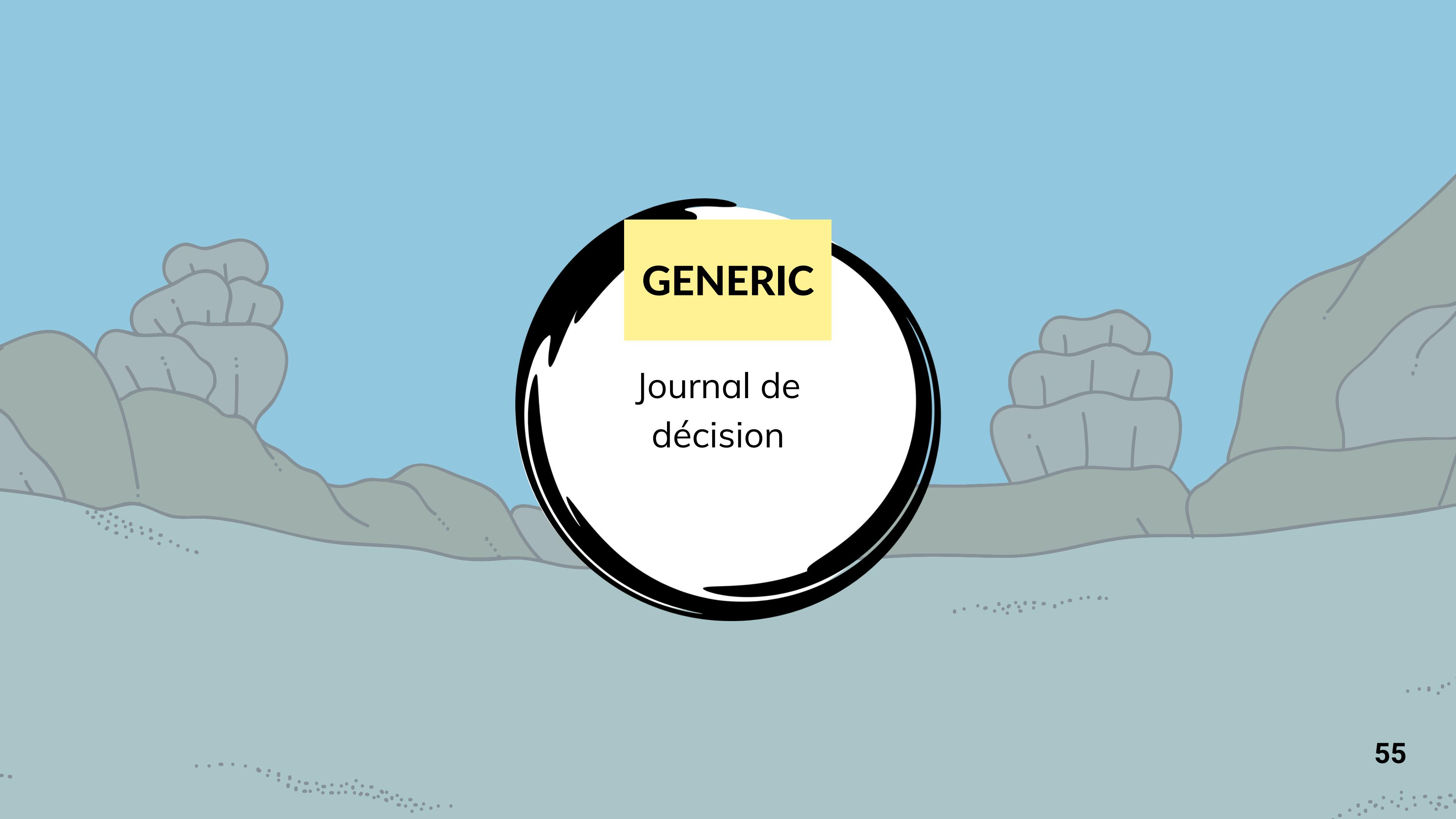


GENERIC

- Besoin standard
- Externalisable / sur étagère

Core, supporting, ou générique ?



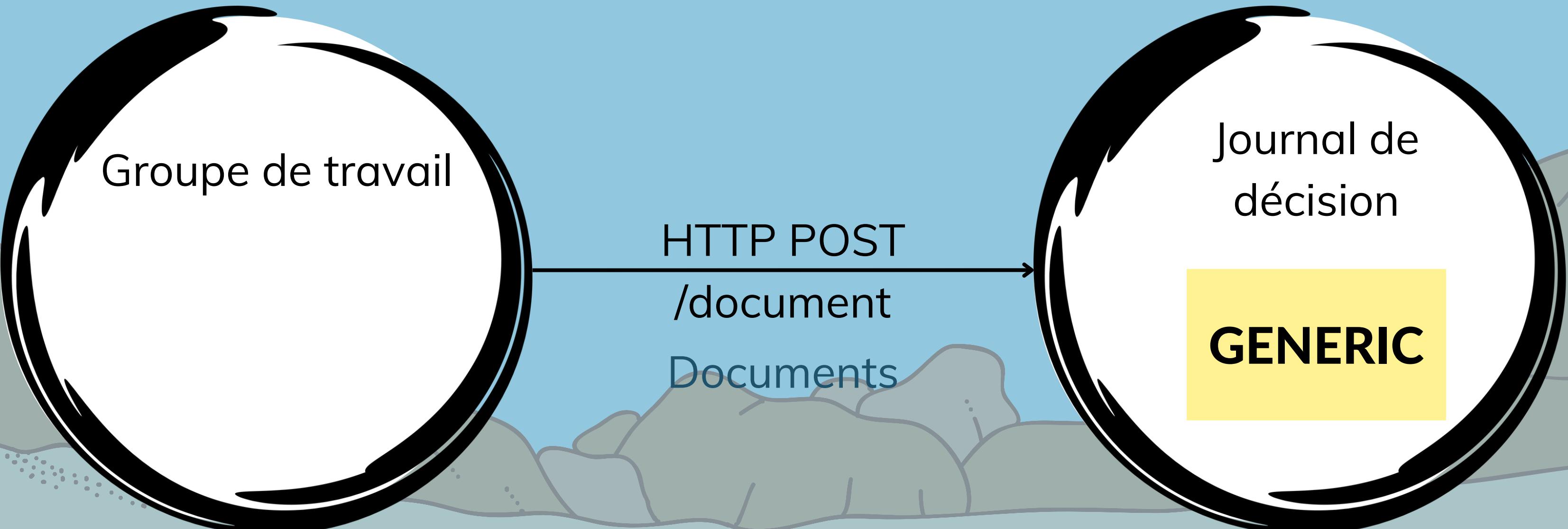


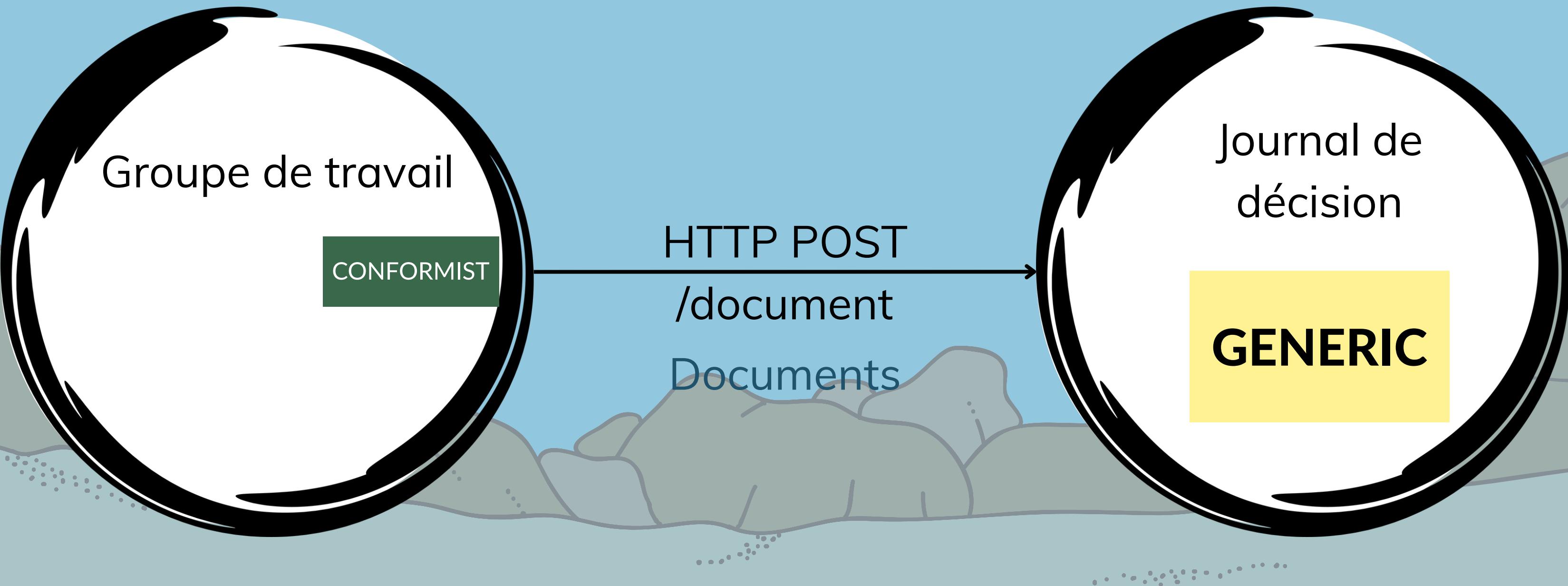
GENERIC

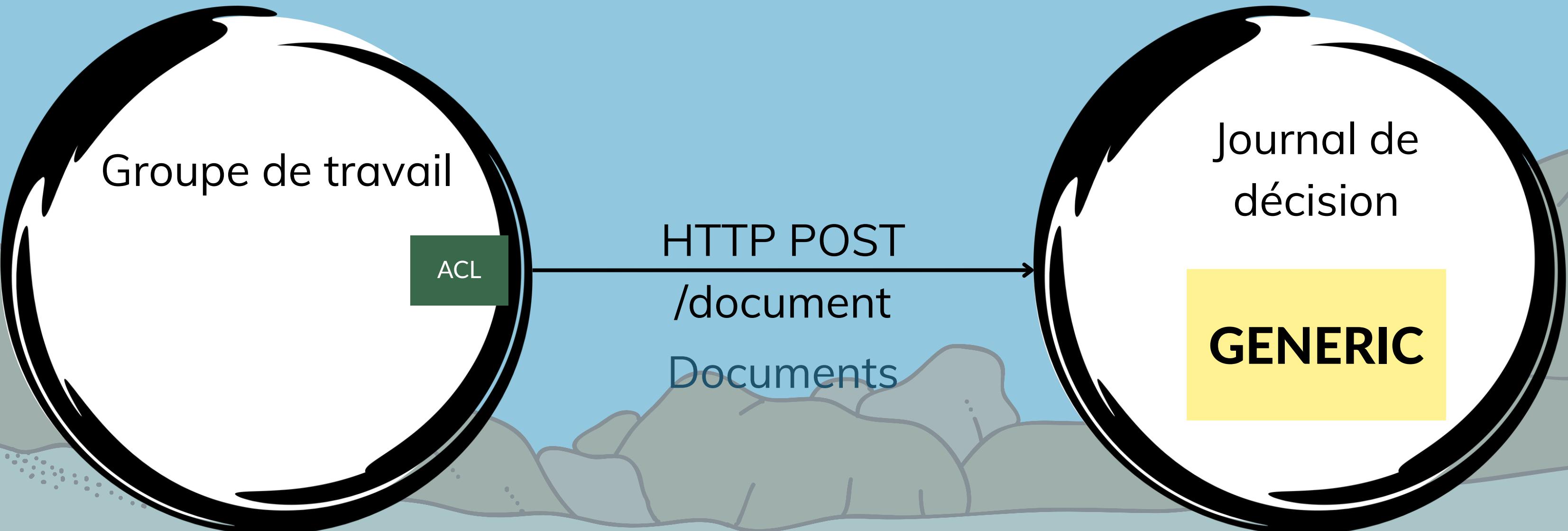
Journal de
décision

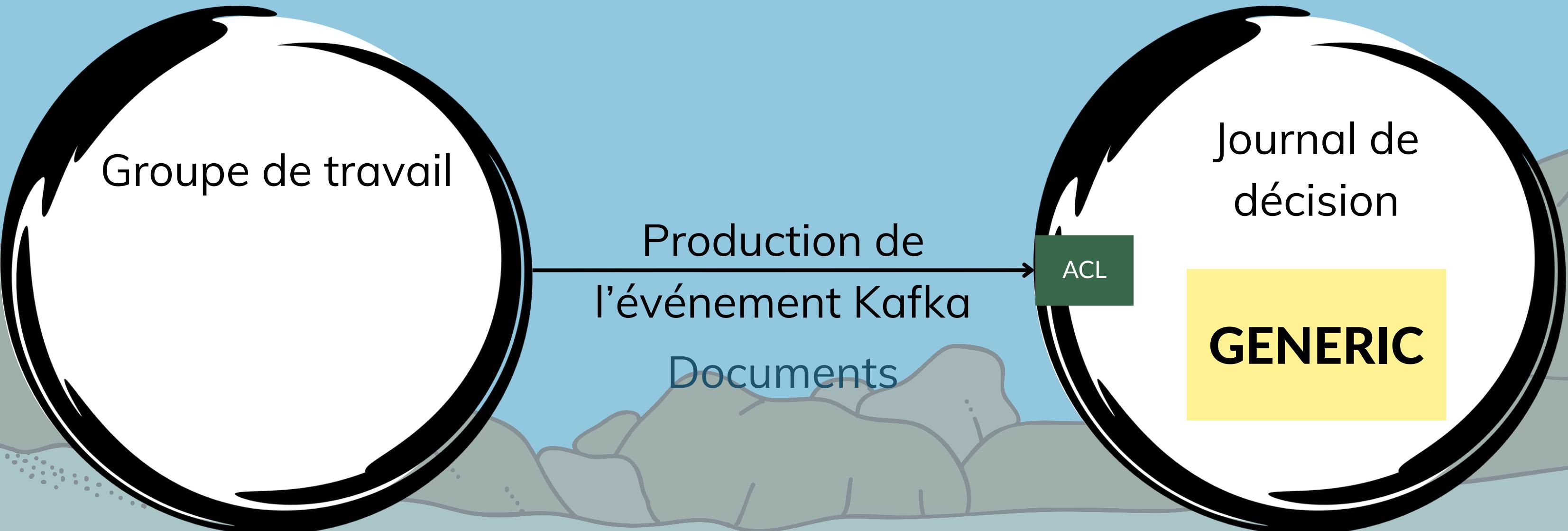
Independants, mais pas isolés

Mais les sous-domaines, ils parlent entre eux non ?









UPSTREAM / DOWNSTREAM

CUSTOMER / SUPPLIER

OPEN HOST SERVICE

SEPARATE WAYS

CONFORMIST

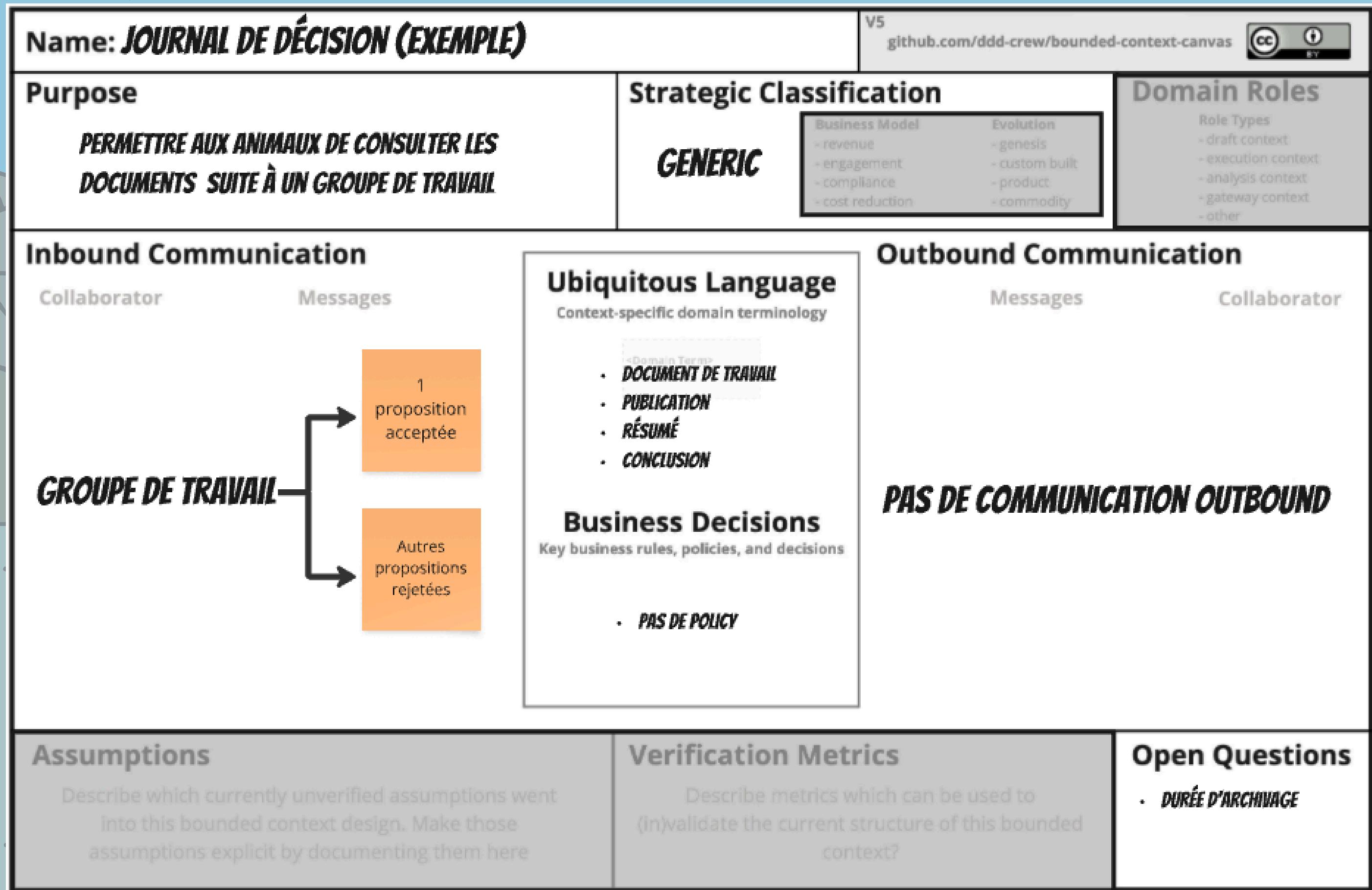
**ANTI-CORRUPTION LAYER
(ACL)**

PUBLISHED LANGUAGE

BIG BALL OF MUD

SHARED KERNEL

Bounded Context Canvas



A Votre tour

- 4 Groupes
- 30 minutes pour faire le plan stratégique
- Ouvrez l'enveloppe numéro 2

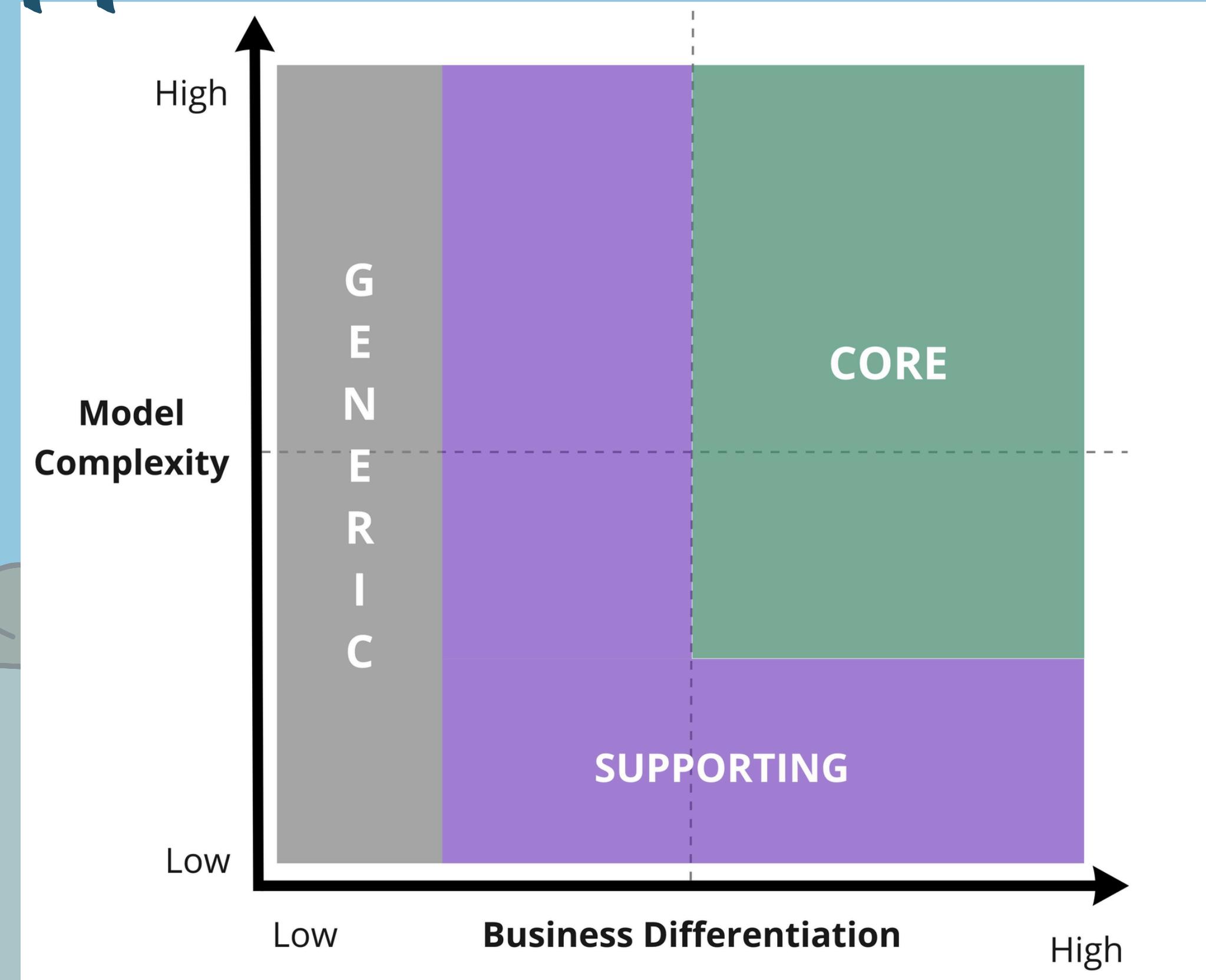
A Votre tour





Debrief

Core, supporting, ou générique ?



NOS SOUS-DOMAINES

SUPPORTING

Idéation

SUPPORTING

Constitution du
groupe de travail

SUPPORTING

Vote

GENERIC

Journal de
décision

CORE

Groupe de travail

NOS SOUS-DOMAINES

GENERIC

Idéation

SUPPORTING

Constitution du
groupe de travail

CORE

Groupe de travail

GENERIC

Vote

GENERIC

Journal de
décision

NOS SOUS-DOMAINES

SUPPORTING

Idéation

CORE

Constitution du
groupe de travail

SUPPORTING

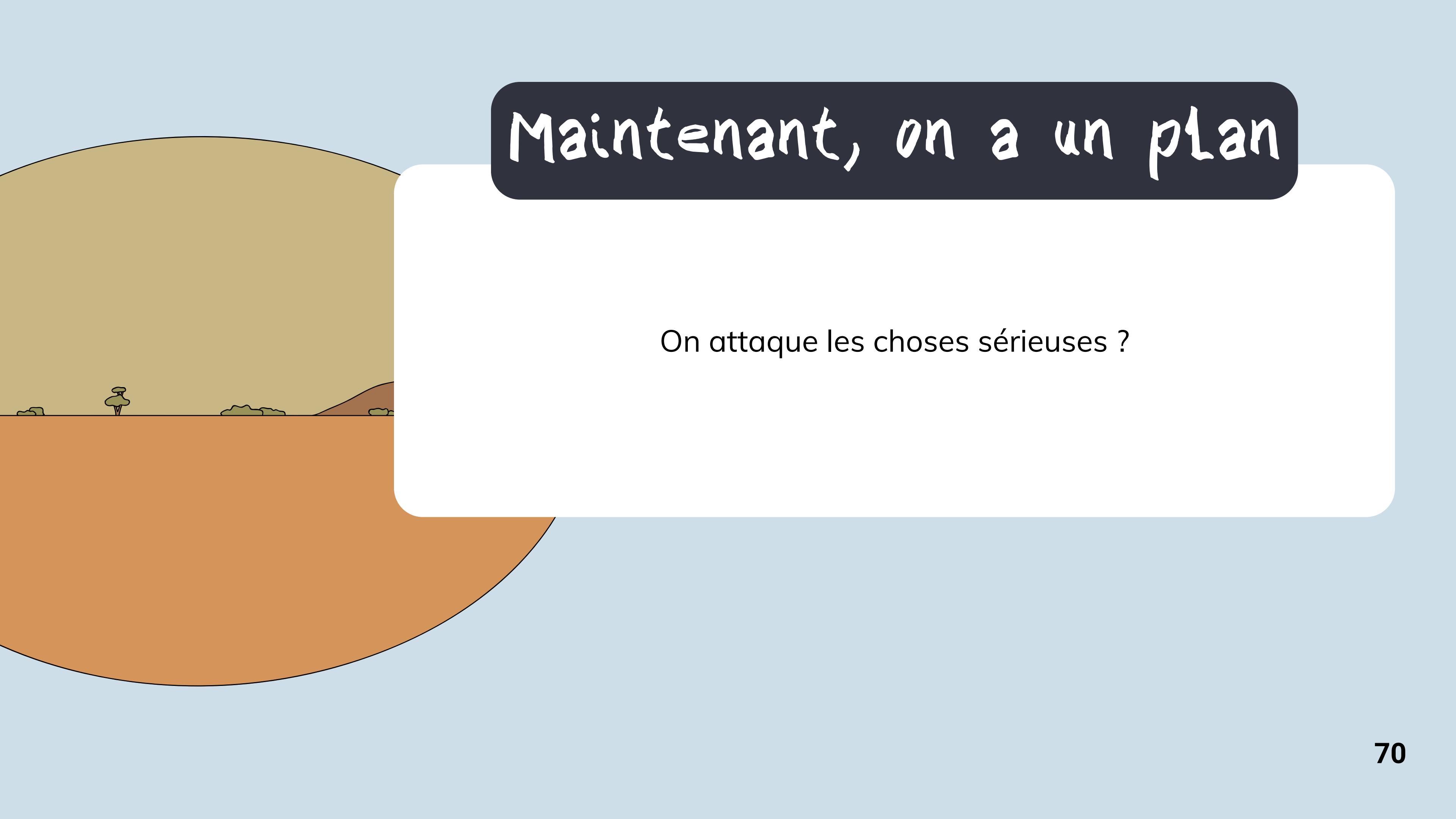
Groupe de travail

GENERIC

Vote

GENERIC

Journal de
décision



Maintenant, on a un plan

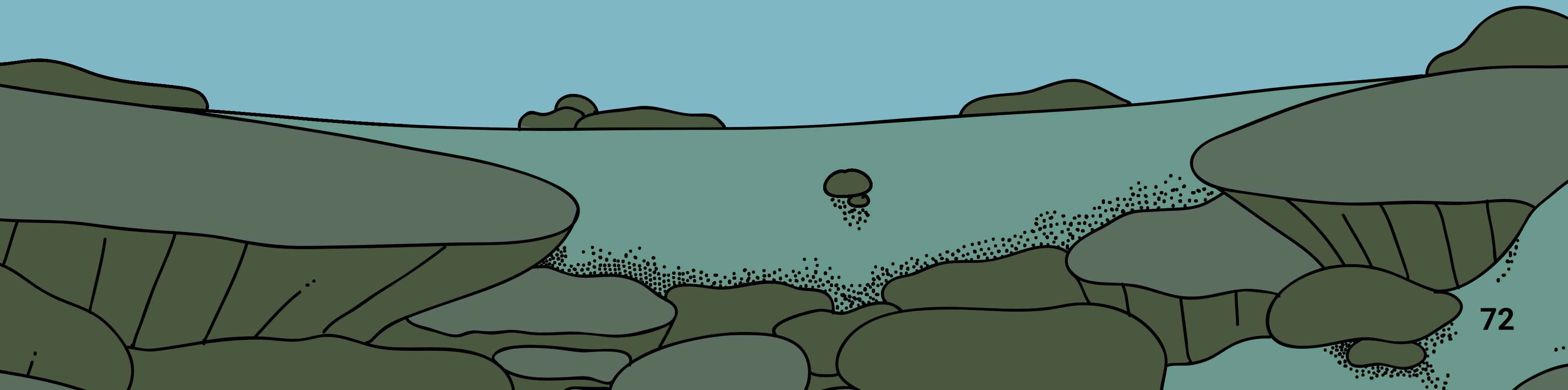
On attaque les choses sérieuses ?



C'est la pause !!!

5 minutes

DDD Tactique



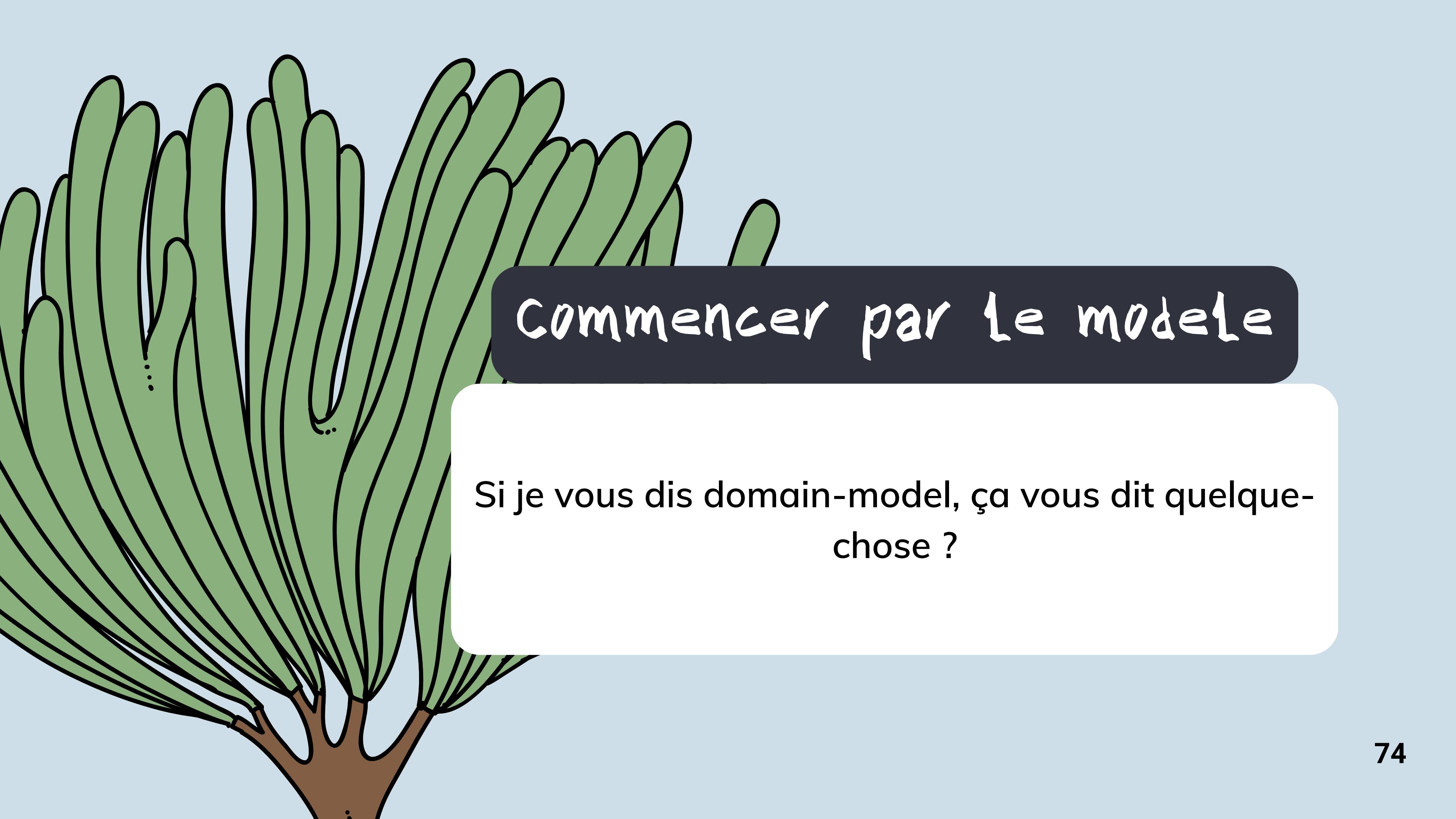
Rappel

Nous avons identifié nos sous-domaines et leurs dépendances.

Maintenant, il nous reste à créer les modèles de chacun de nos Bounded Context.

Nous allons nous concentrer sur la constitution du groupe de travail et les règles qui la régissent.

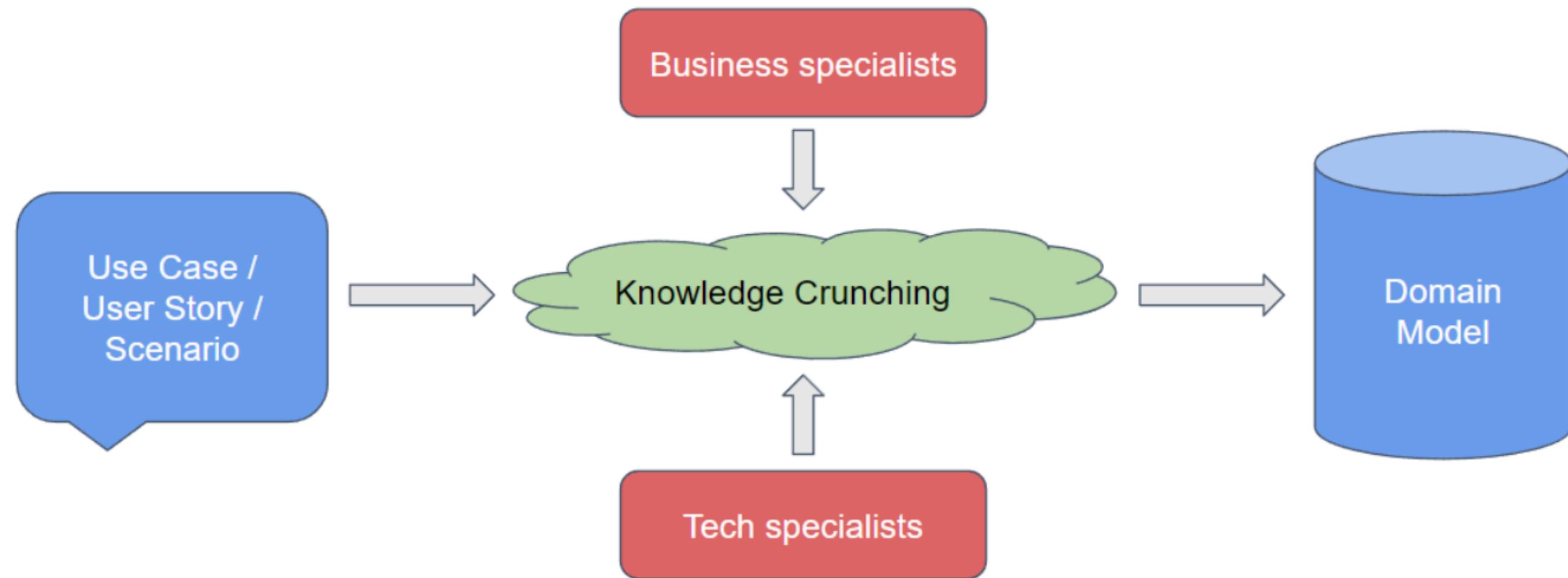
Pour cela, nous allons utiliser les outils tactiques du DDD.



Commencer par le modèle

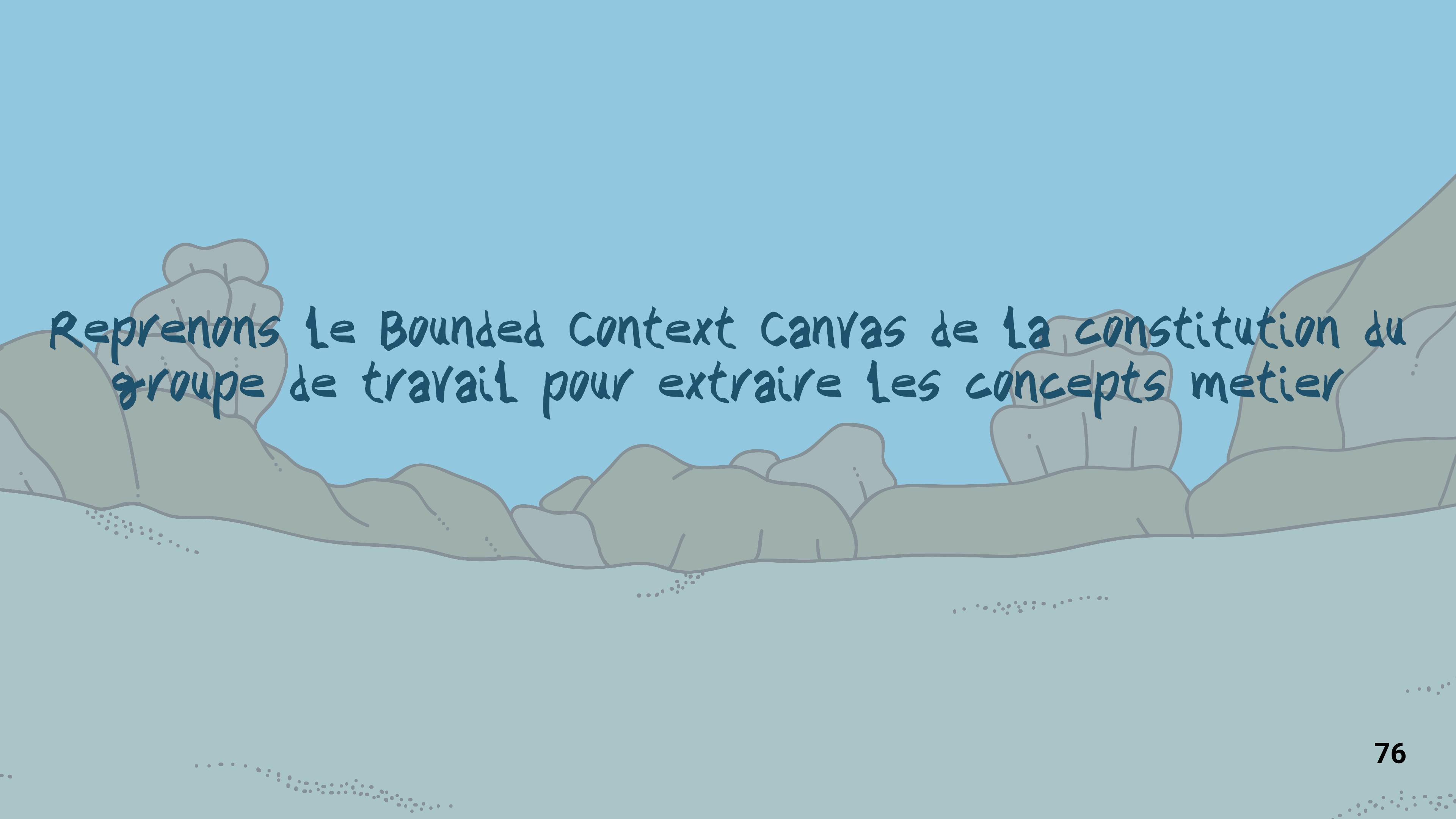
Si je vous dis domain-model, ça vous dit quelque-chose ?

Knowledge crunching



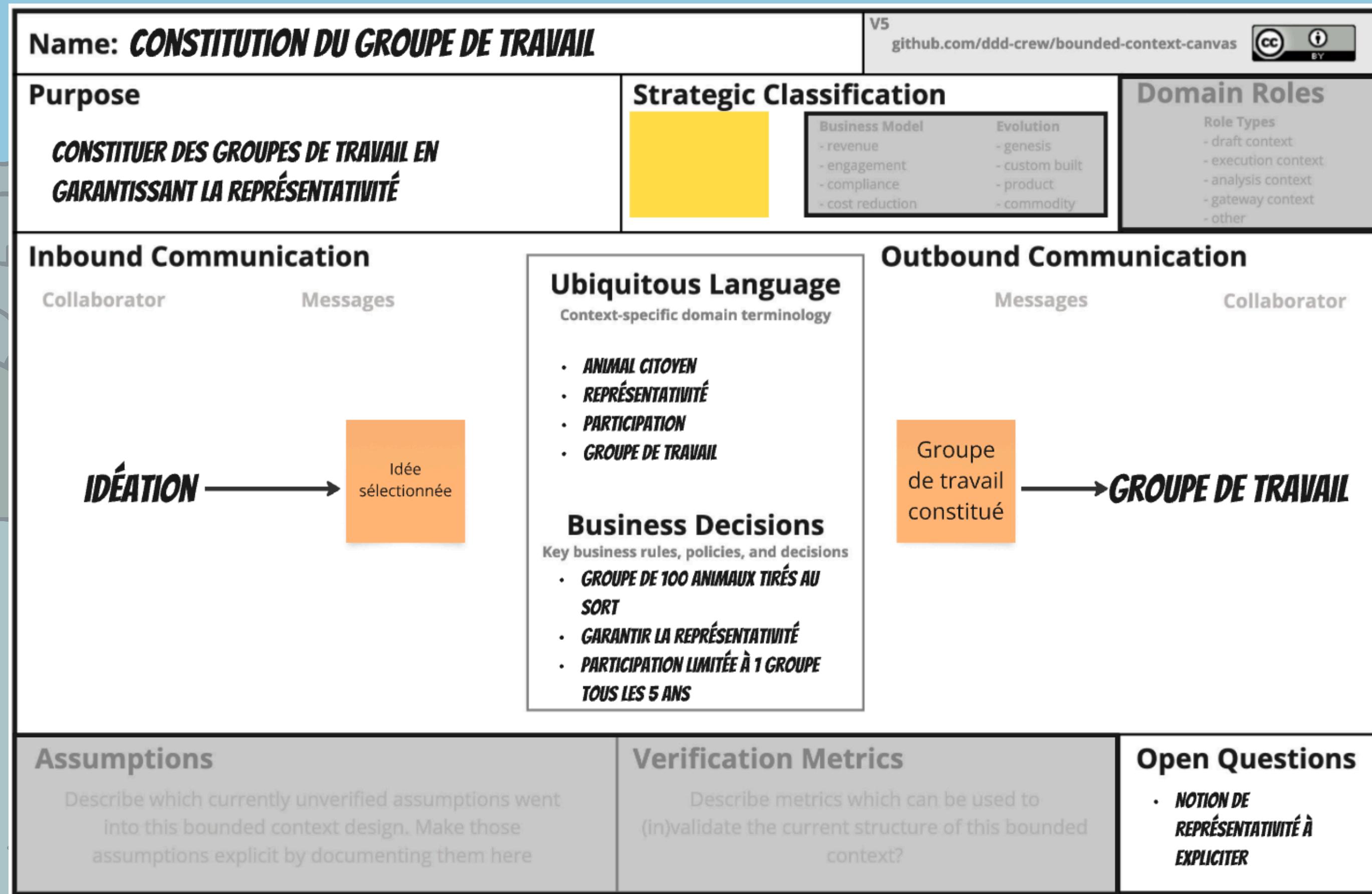
Attribution





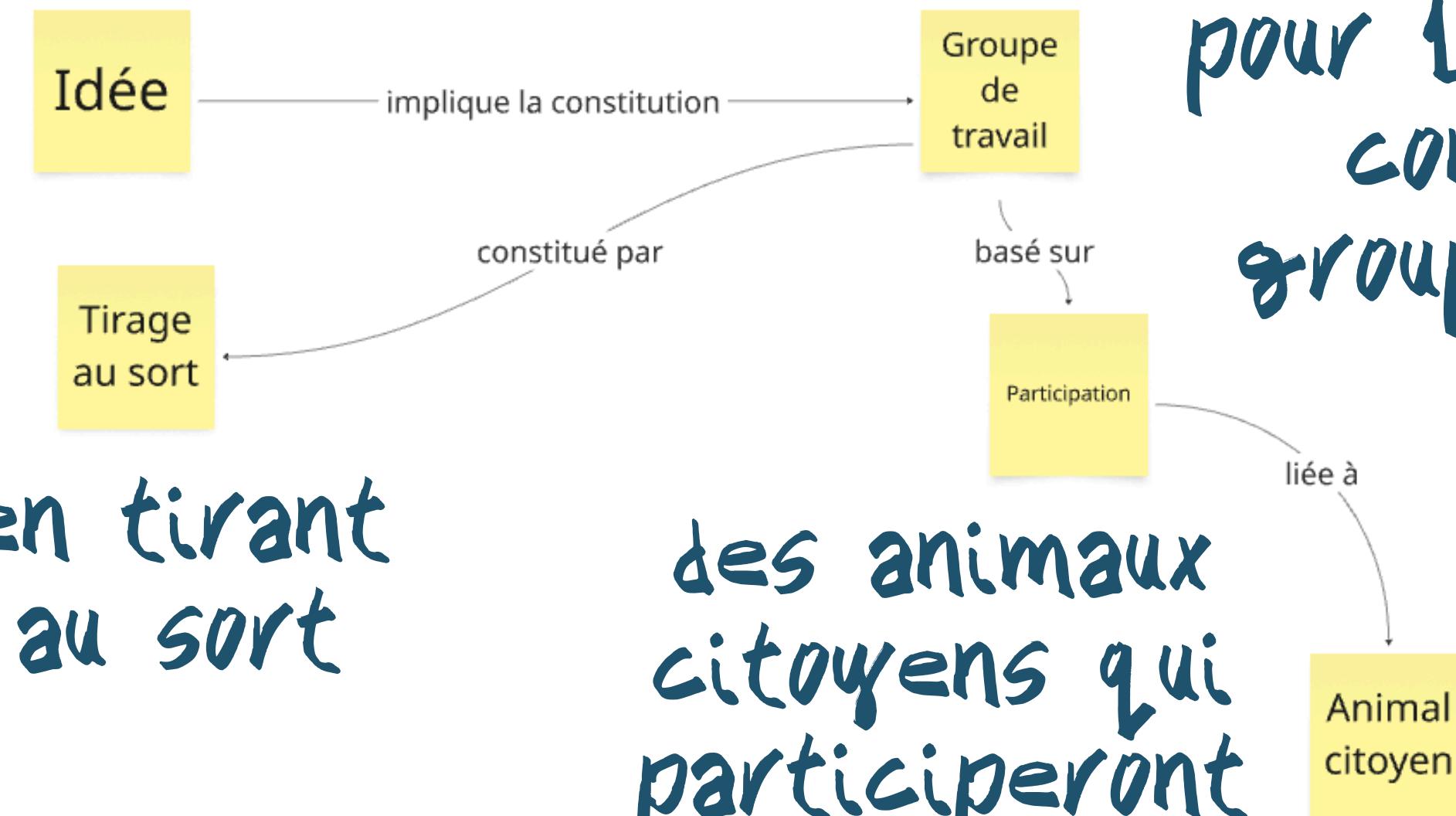
Reprendons le Bounded Context canvas de la constitution du groupe de travail pour extraire les concepts métier

Bounded Context Canvas



Determinons les concepts métier

De l'ideation, nous avons une idée



en tirant
au sort

des animaux
citoyens qui
participeront
au groupe

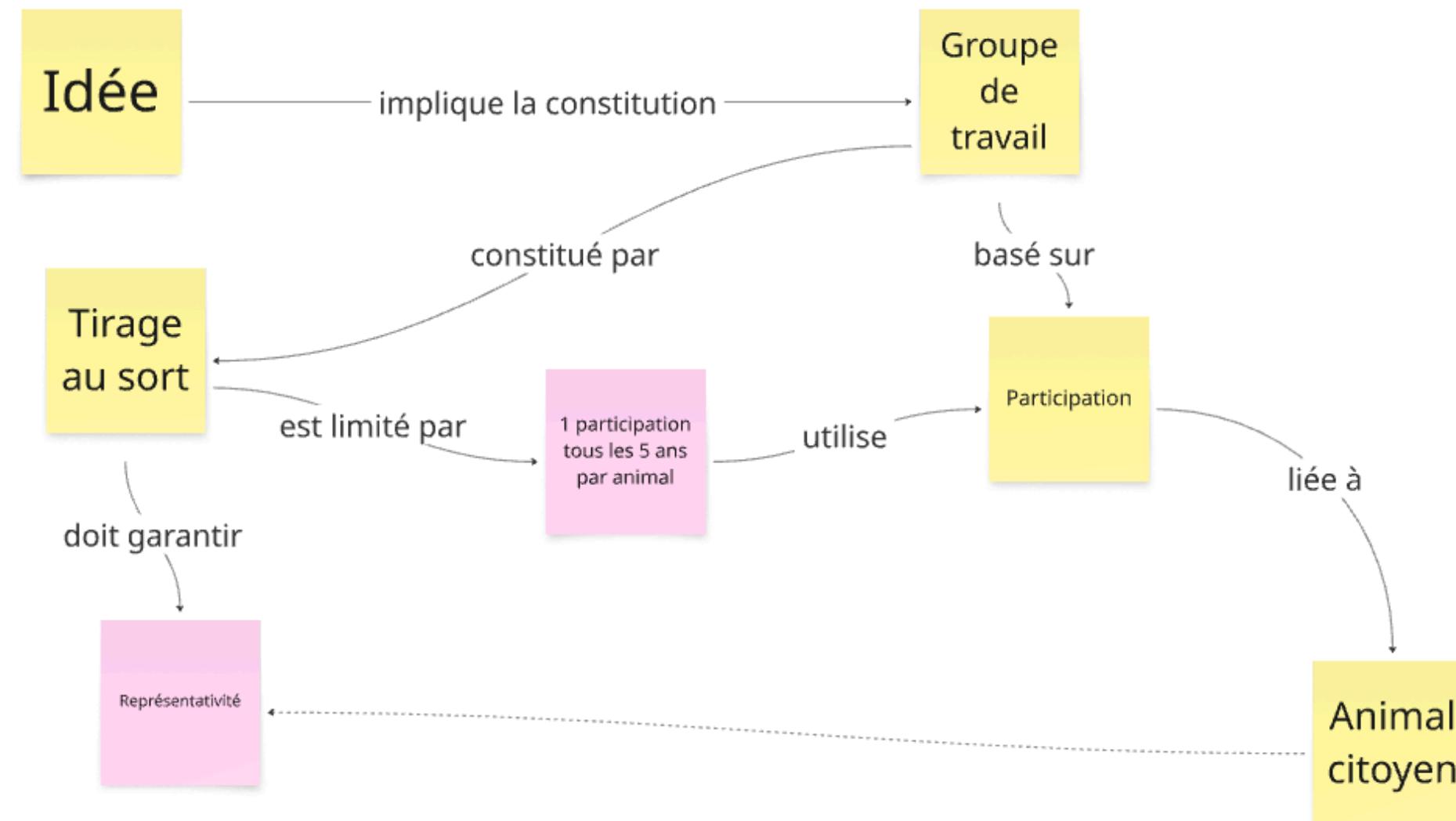
pour laquelle on va
constituer un
groupe de travail



On va maintenant expliciter les règles métier qui contraignent notre modèle

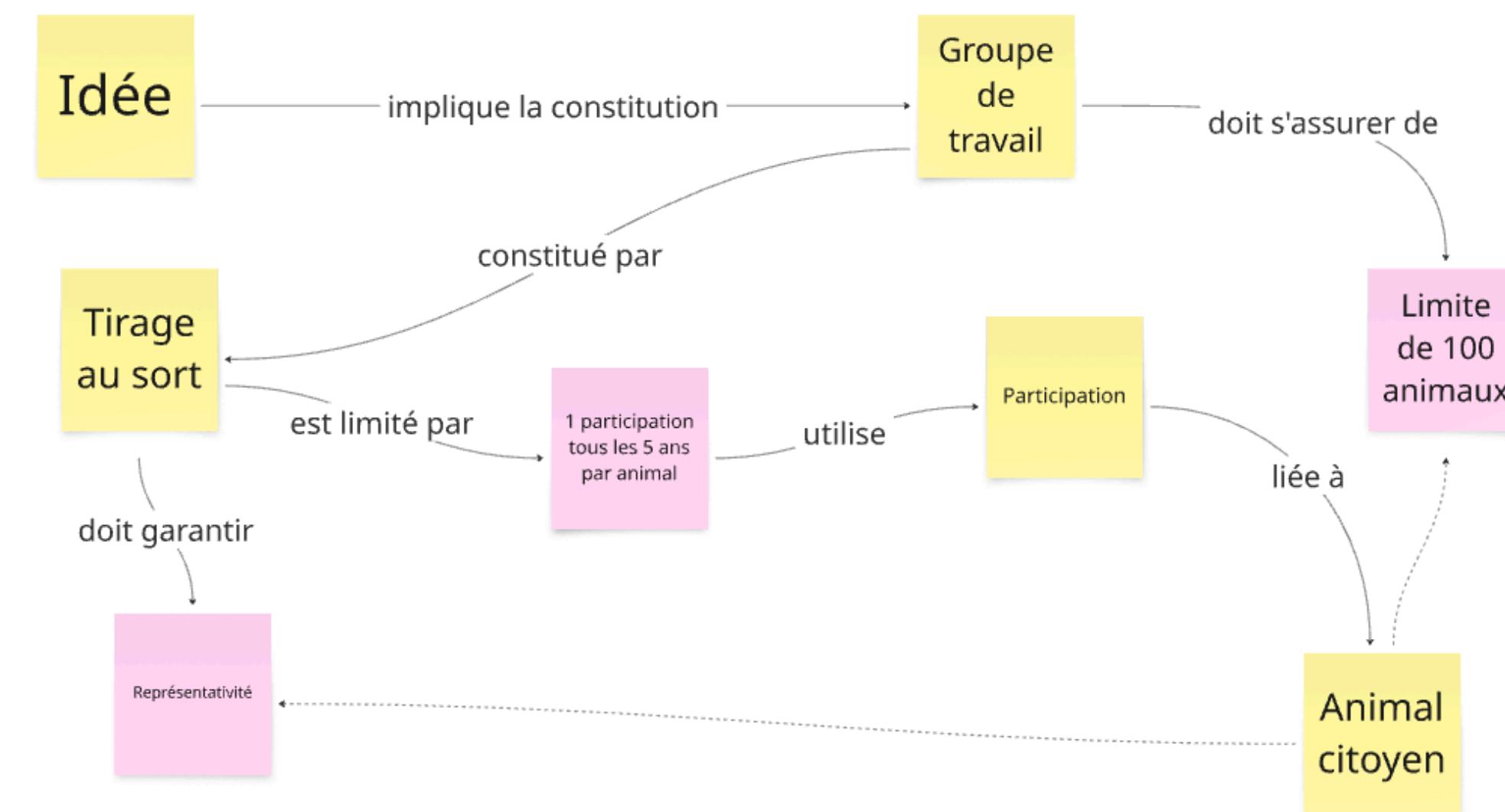
Règles du tirage au sort

Représentativité & Limite de participation



Règles du groupe de travail

Limite du nombre d'animaux dans un groupe

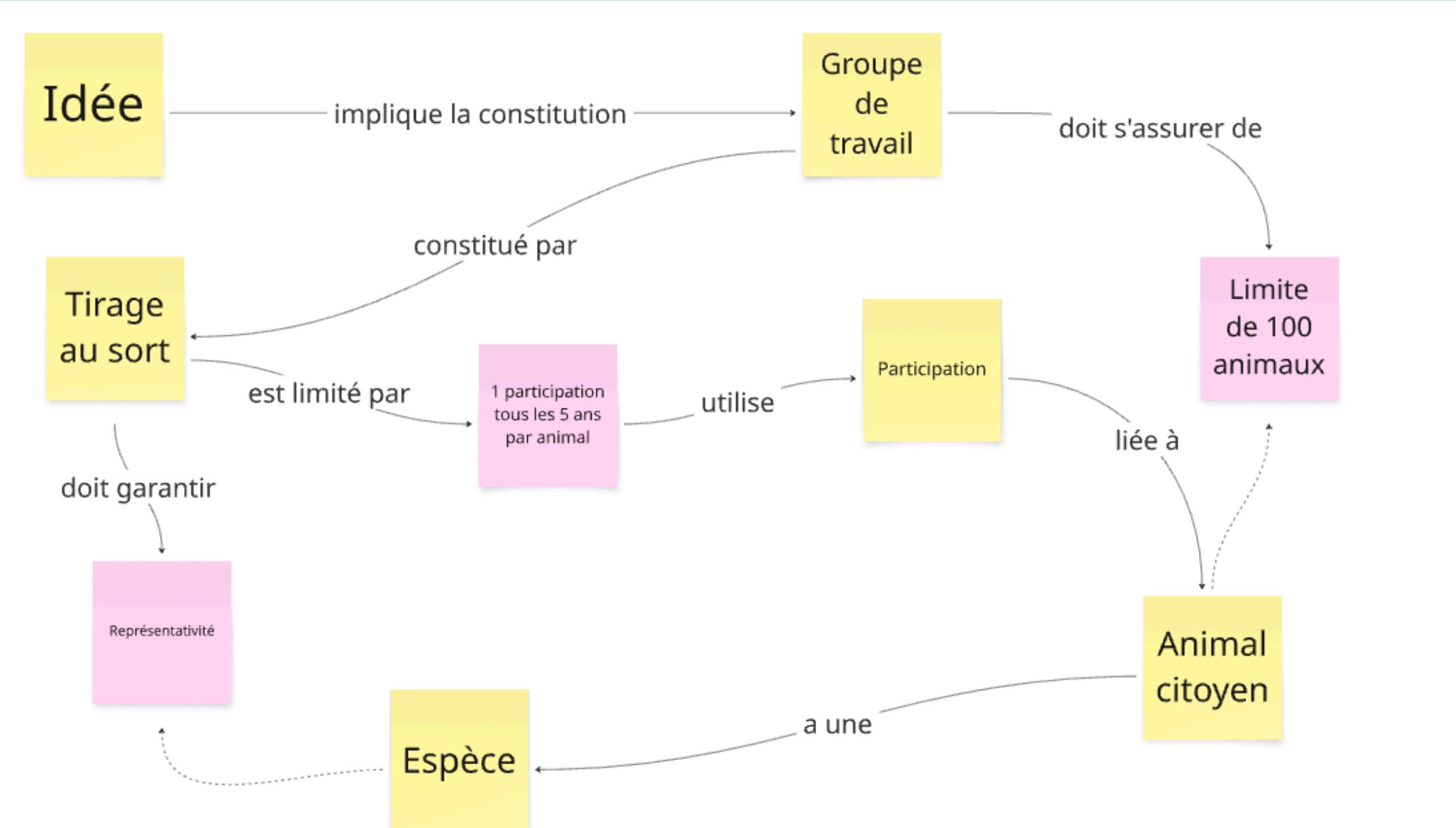




On peut maintenant faire émerger de nouveaux concepts qui semblent importants pour répondre à ces règles

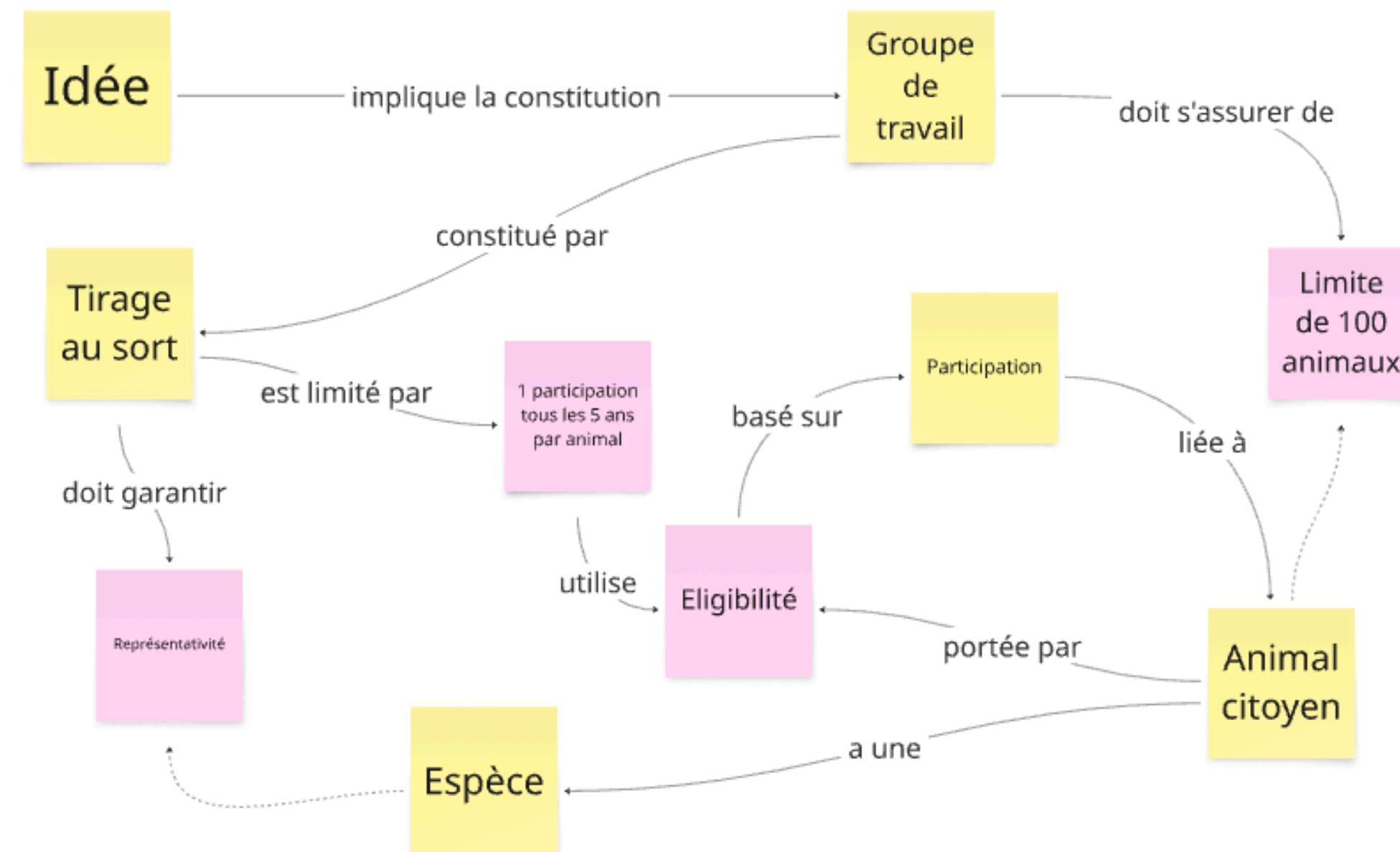
Raffinons notre modèle

La représentativité utilise la notion d'espèce



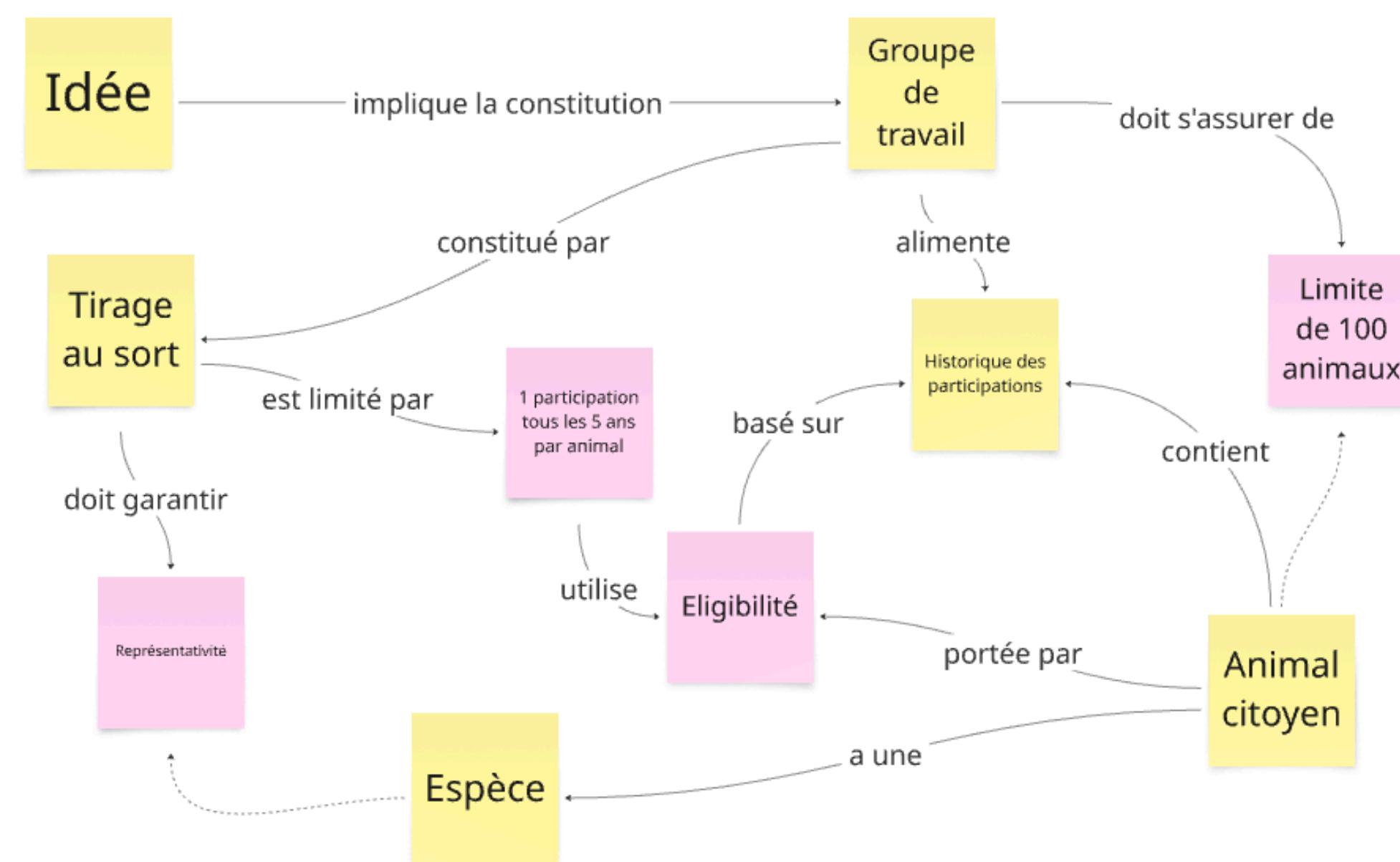
Raffinons notre modèle

C'est l'éligibilité qui permet de garantir la limite de participation



Raffinons notre modèle

Le groupe de travail alimente l'historique des participations

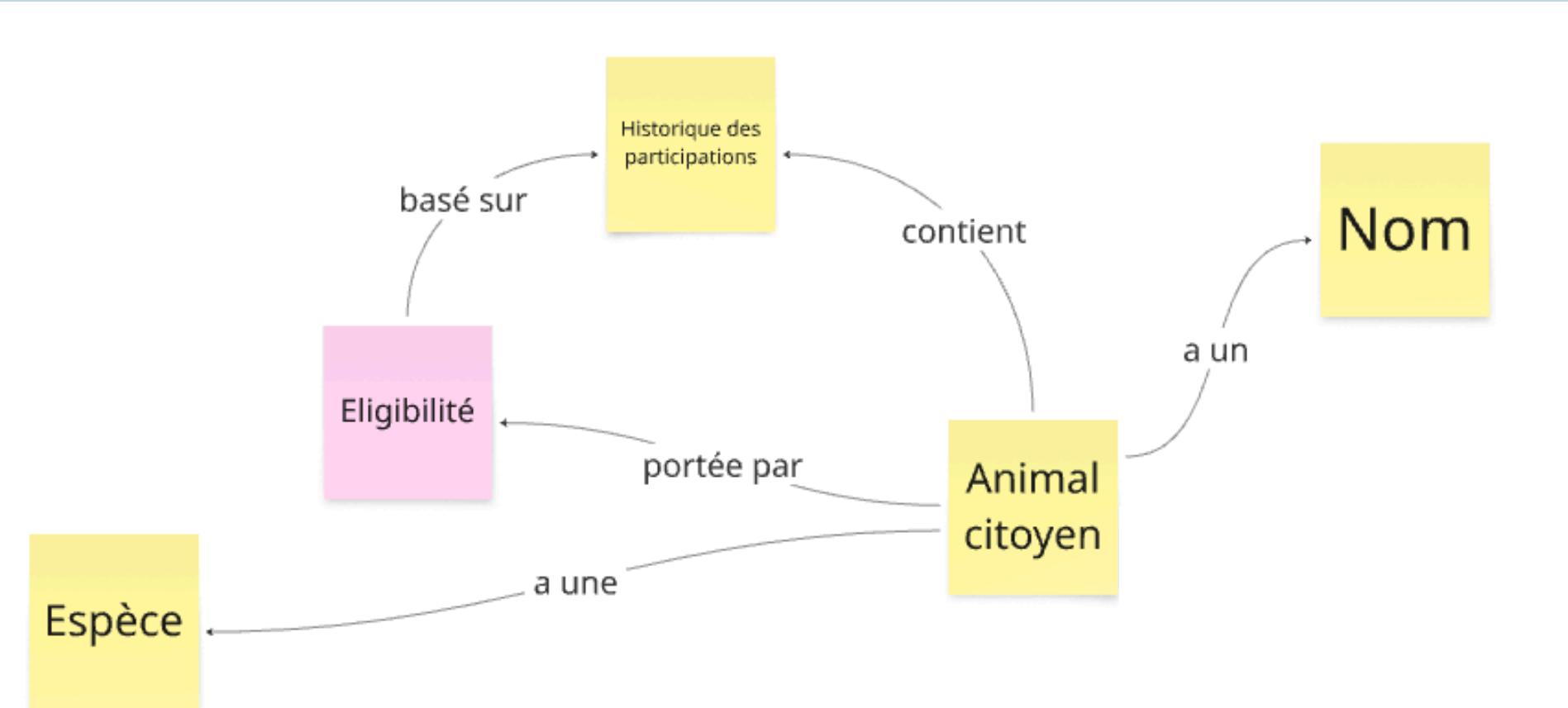




On peut rajouter des informations manquantes sur
les animaux

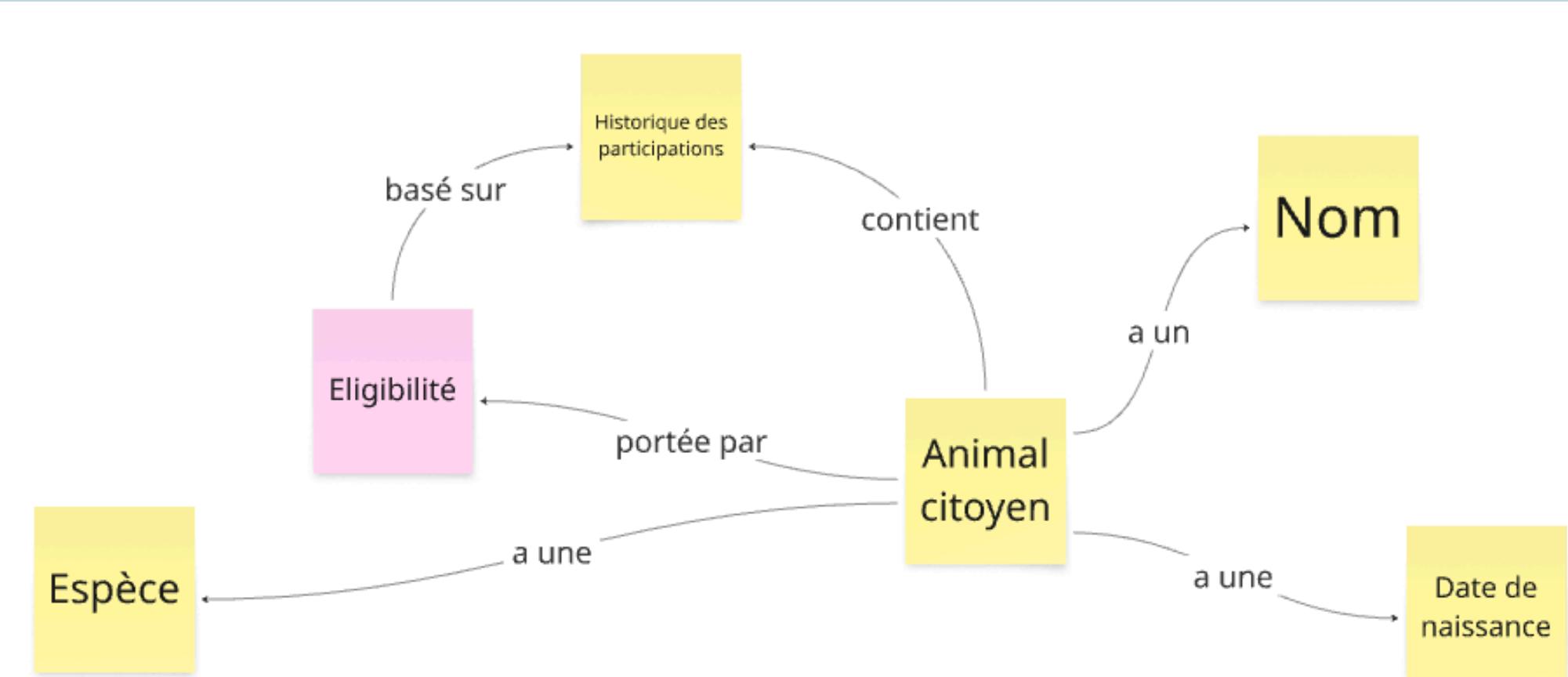
Est-ce qu'il manque des concepts ?

Le nom peut servir à identifier
l'animal



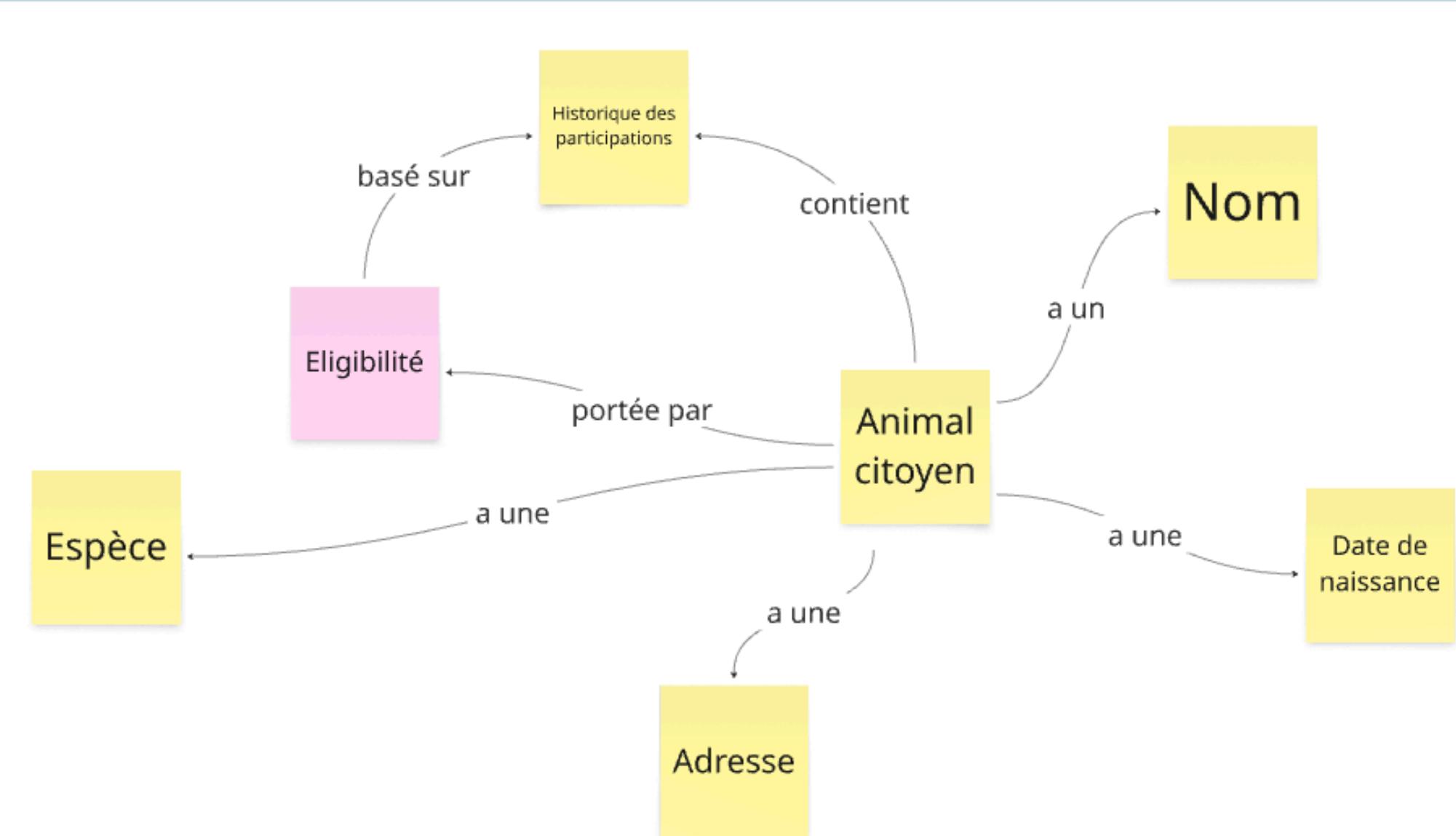
Est-ce qu'il manque des concepts ?

La date de naissance permet de filtrer pour la représentativité



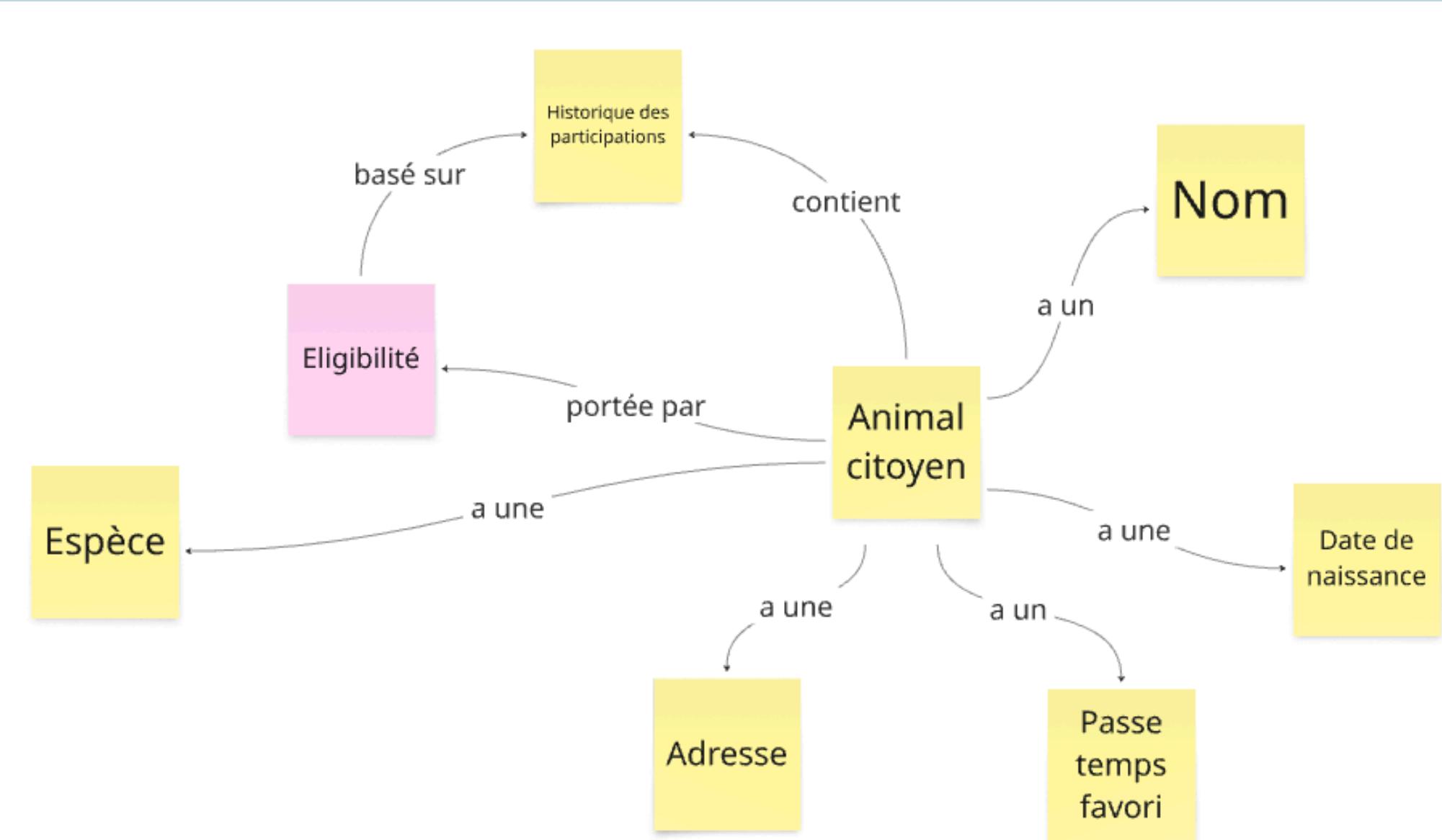
Est-ce qu'il manque des concepts ?

L'adresse peut aider à trouver les animaux qui seront impactés par une idée



Est-ce qu'il manque des concepts ?

Le passe temps favori peut ...
... est-ce qu'on en a vraiment besoin ?



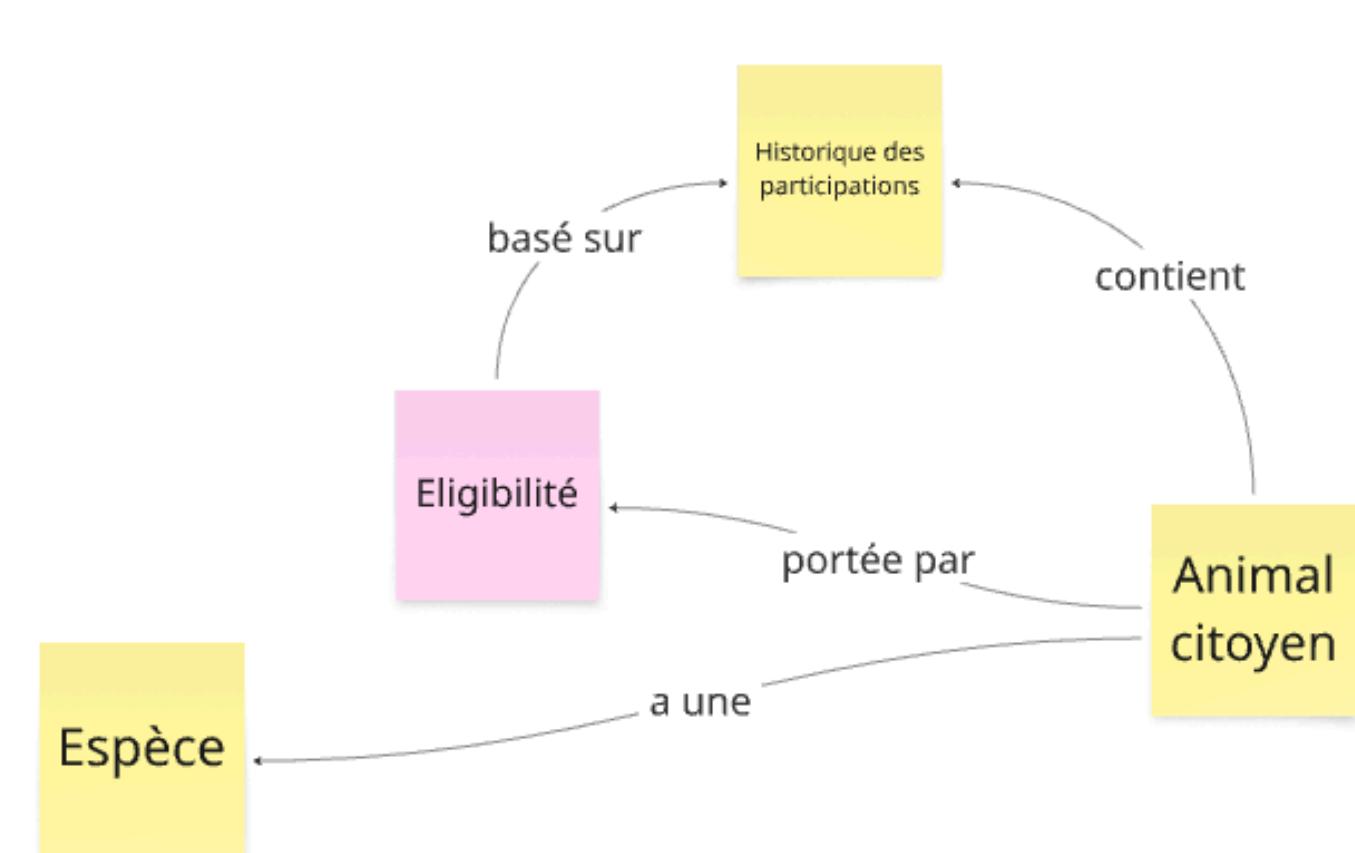


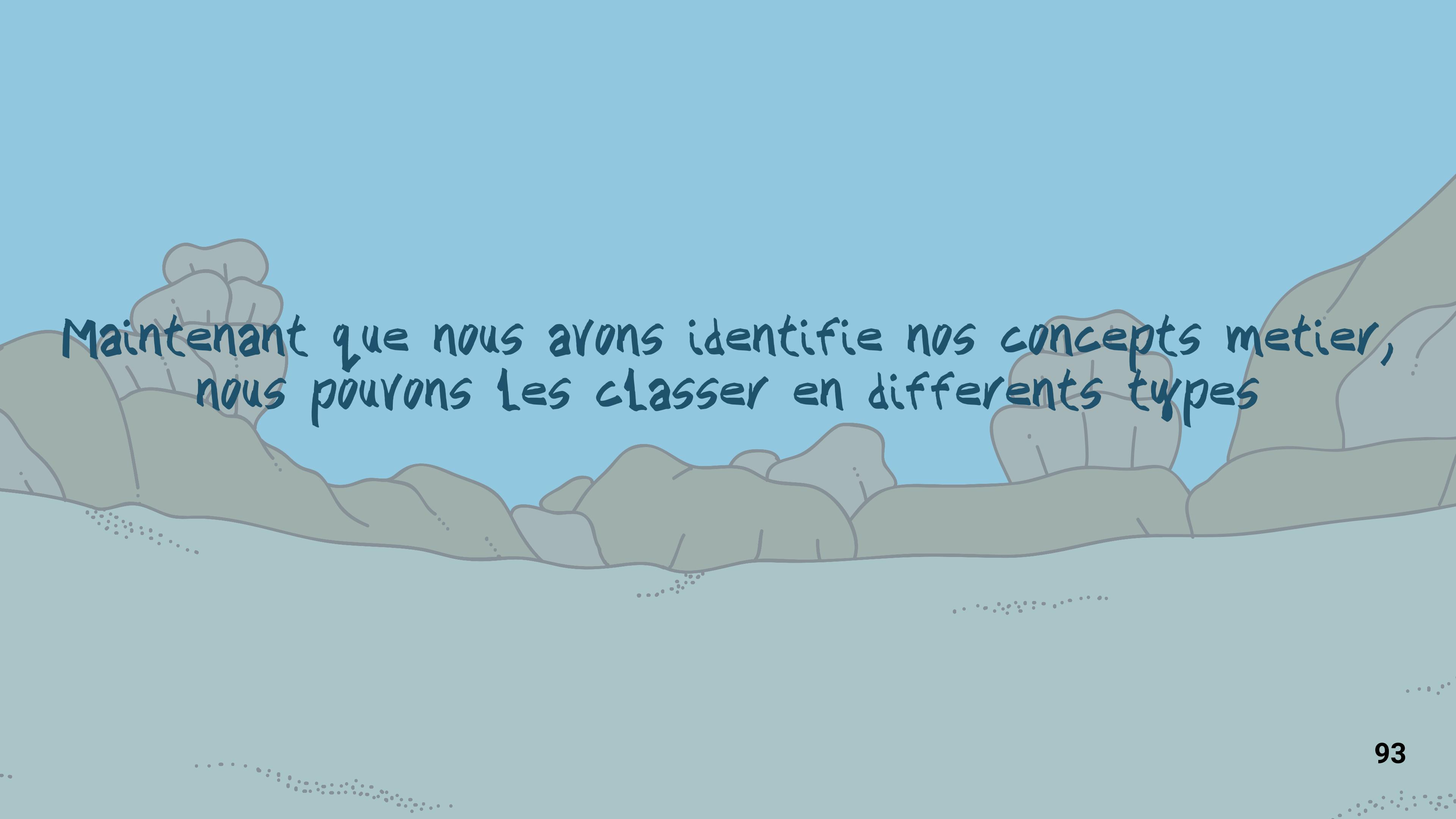
Le modèle doit rester concentré sur les
problématiques métier actuelles, pas sur
d'hypothétiques besoins futurs

Modèle Utile
Pas La Réalité

Est-ce qu'il manque des concepts ?

Le modèle doit se concentrer sur les règles métier du contexte



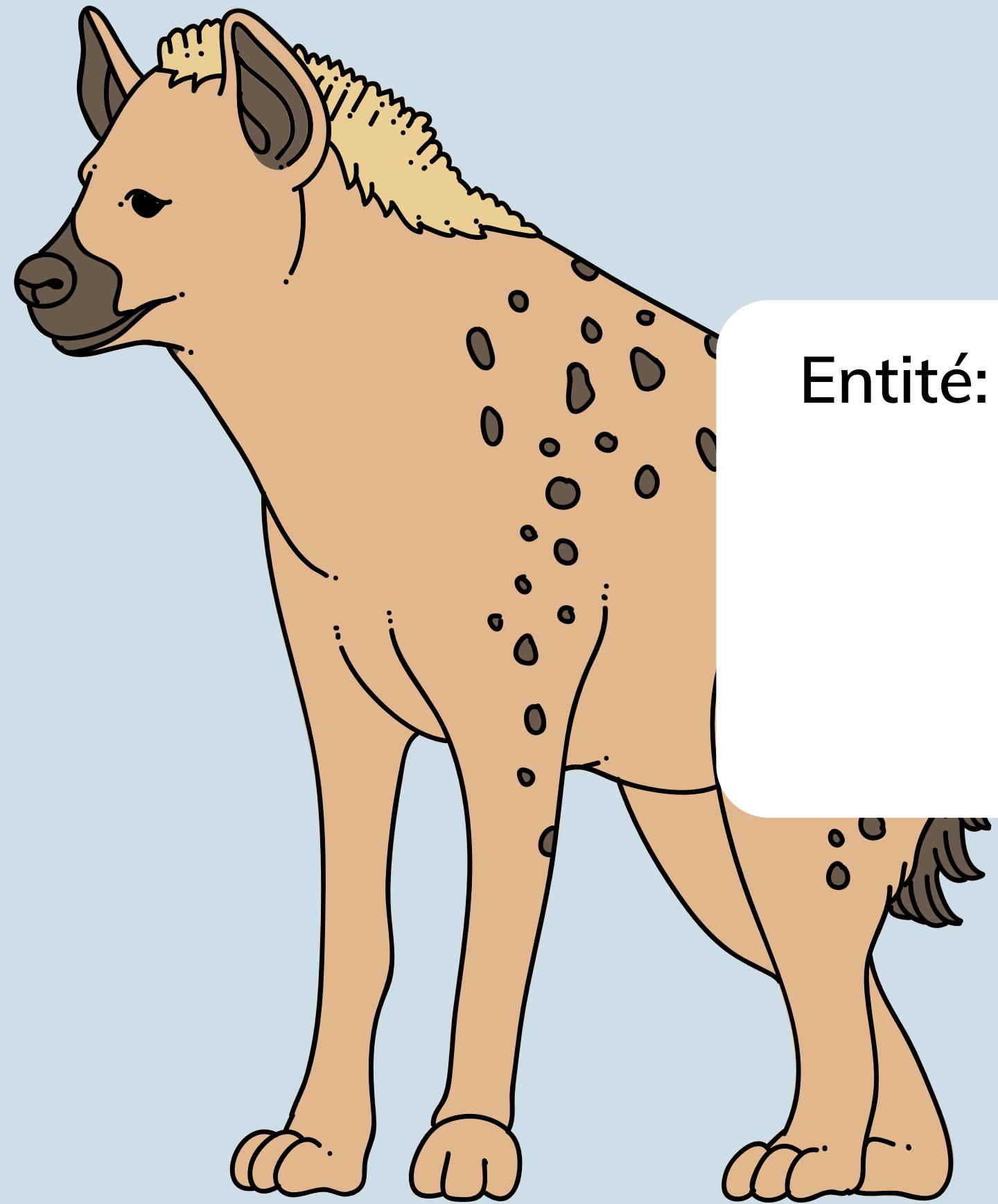


Maintenant que nous avons identifié nos concepts métier,
nous pouvons les classer en différents types



Value Object: seuls les attributs et la logique correspondante sont intéressants

No Identity
Immutability

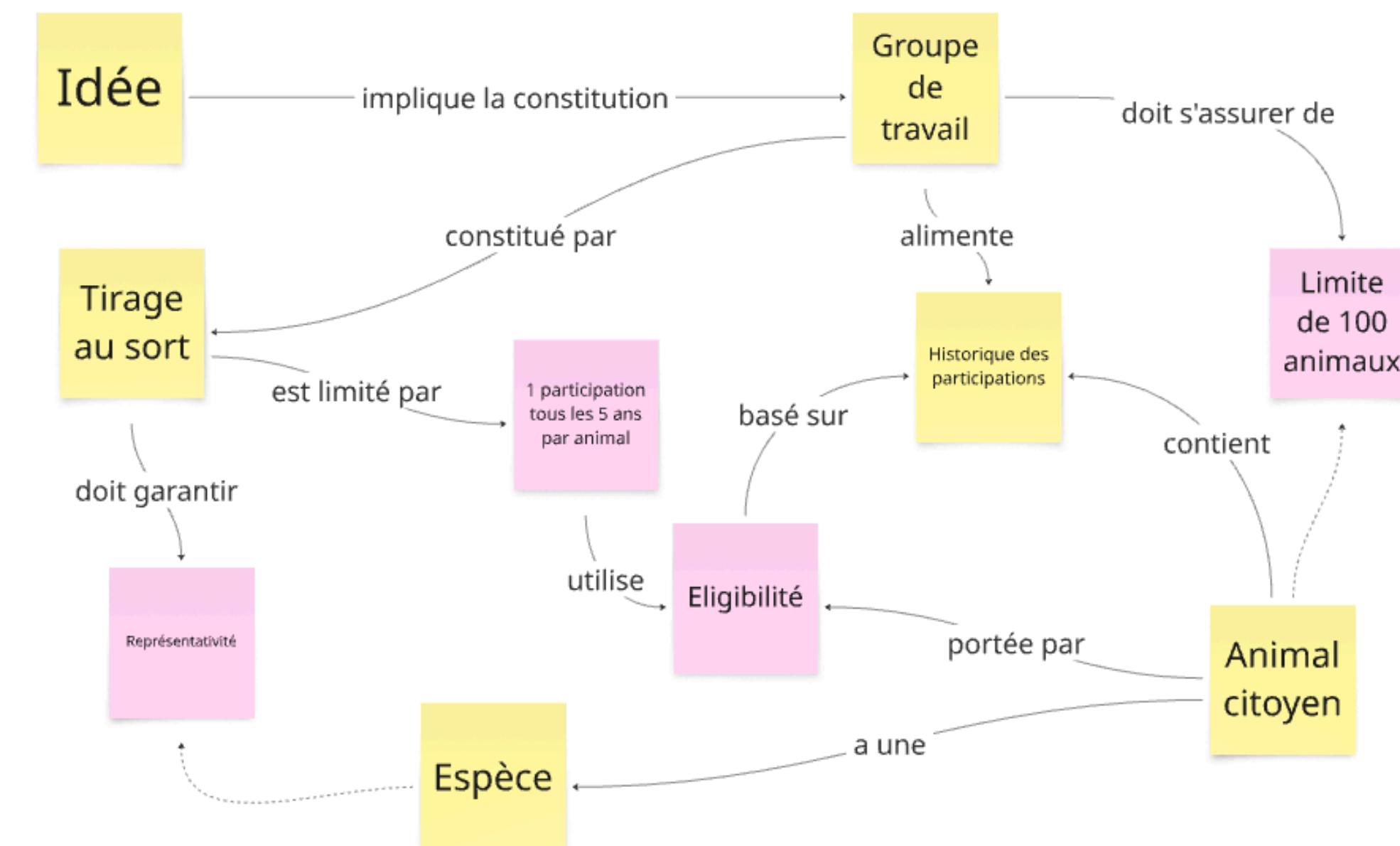


Entité: l'identité d'un objet le distingue des autres objets de la même classe.

Life Cycle
Mutability

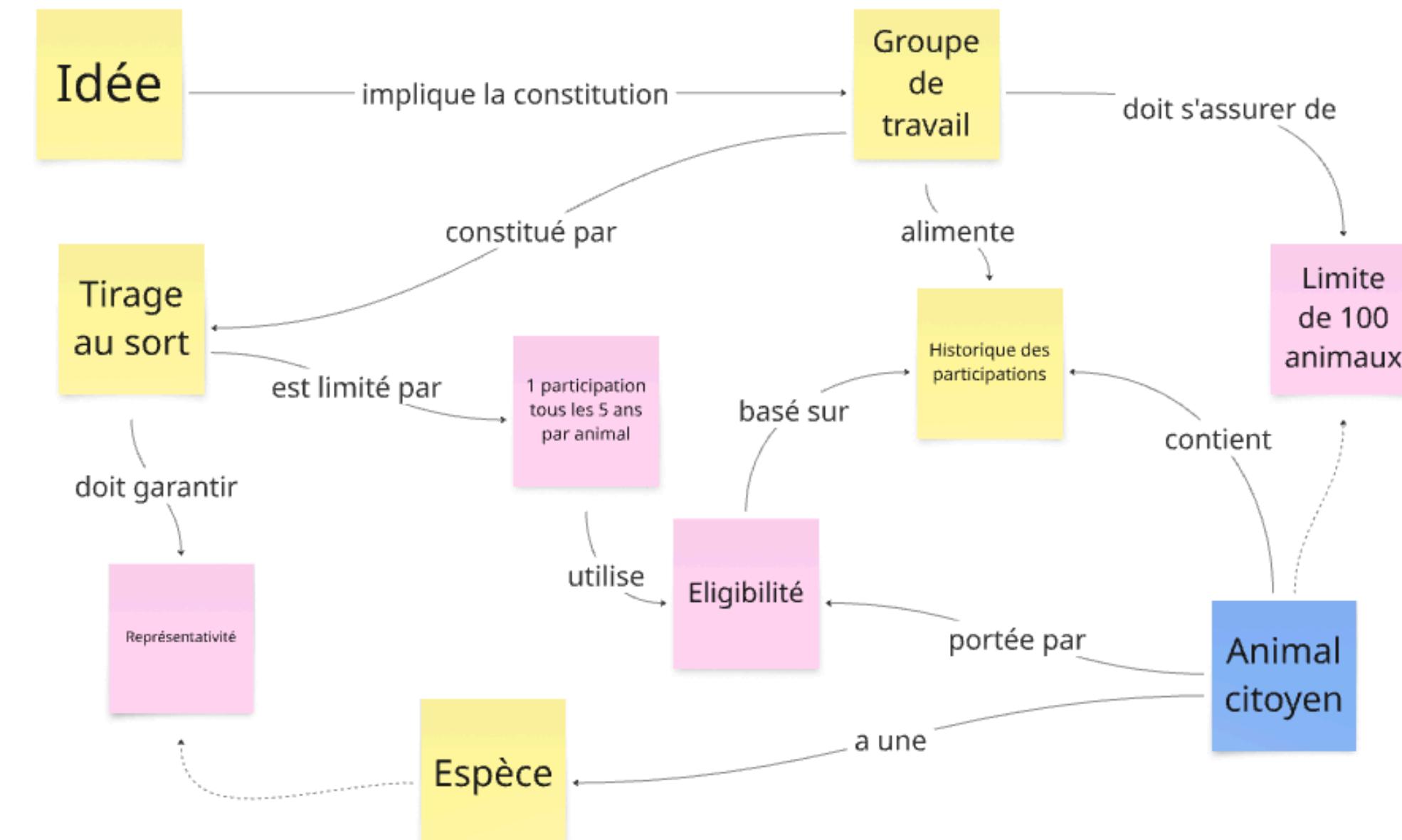
Value Object ou Entité ?

Un cas simple: Animal citoyen



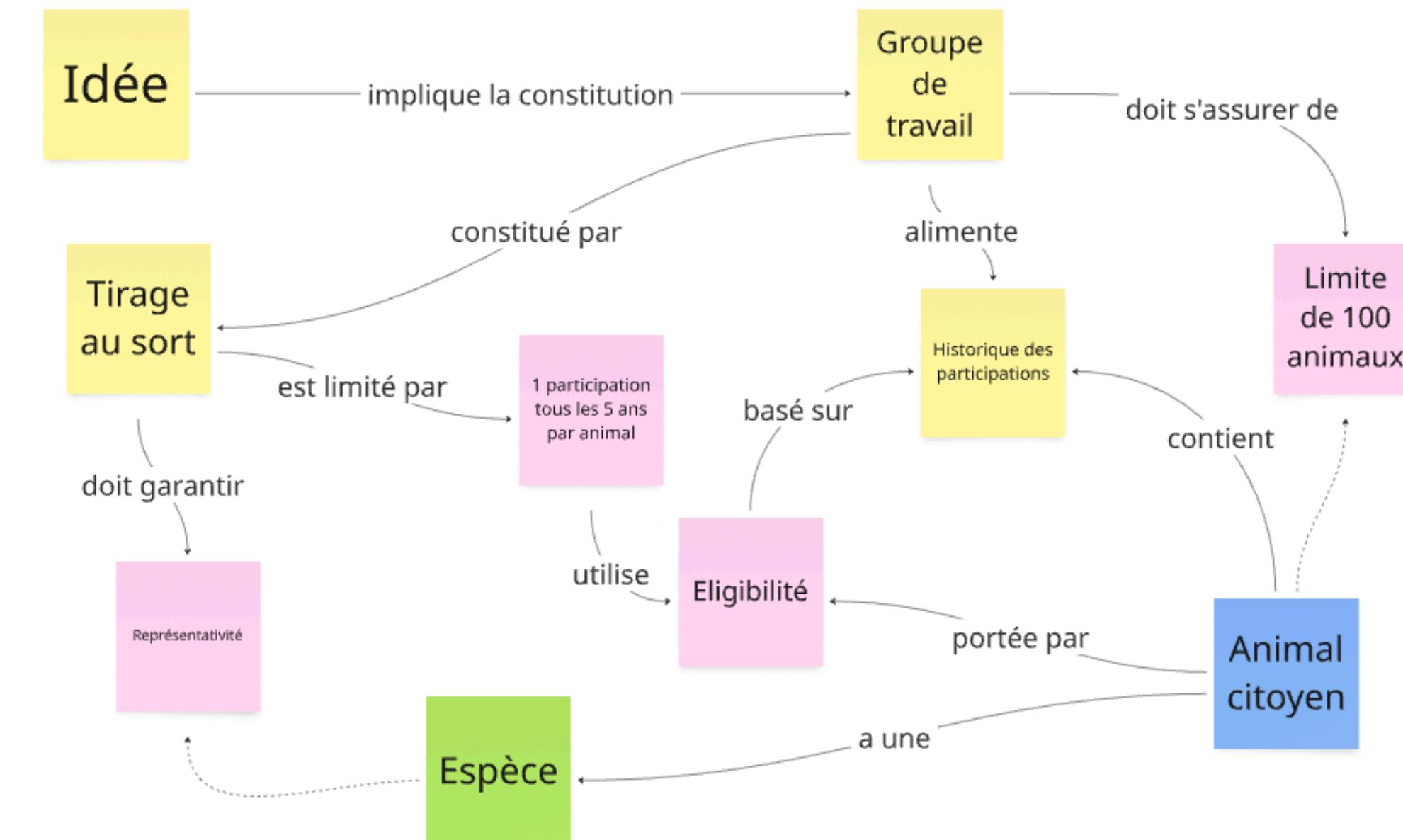
Value Object ou Entité ?

Un peu plus dur: Espèce



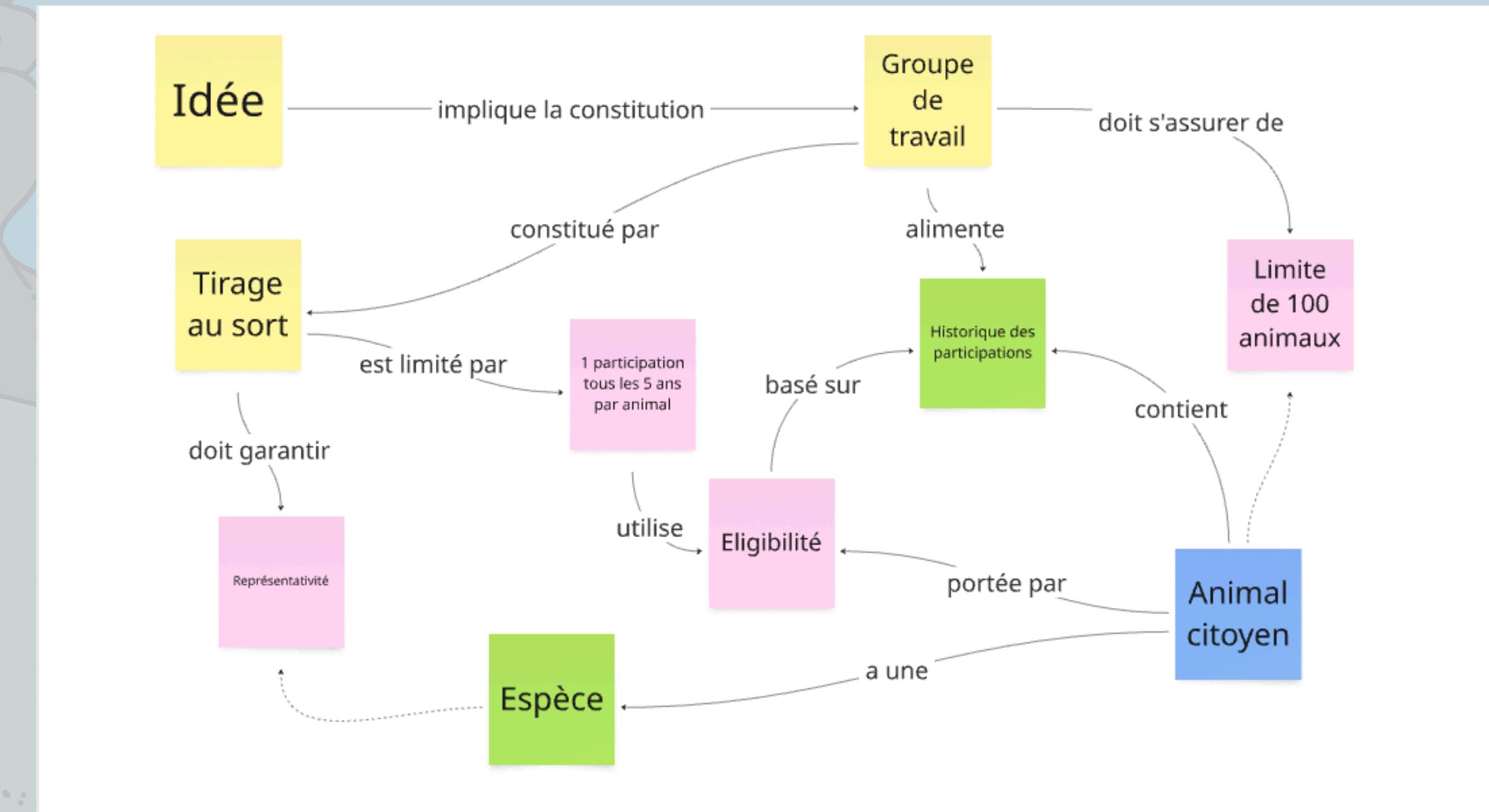
Value Object ou Entité ?

Et l'historique des participations ?



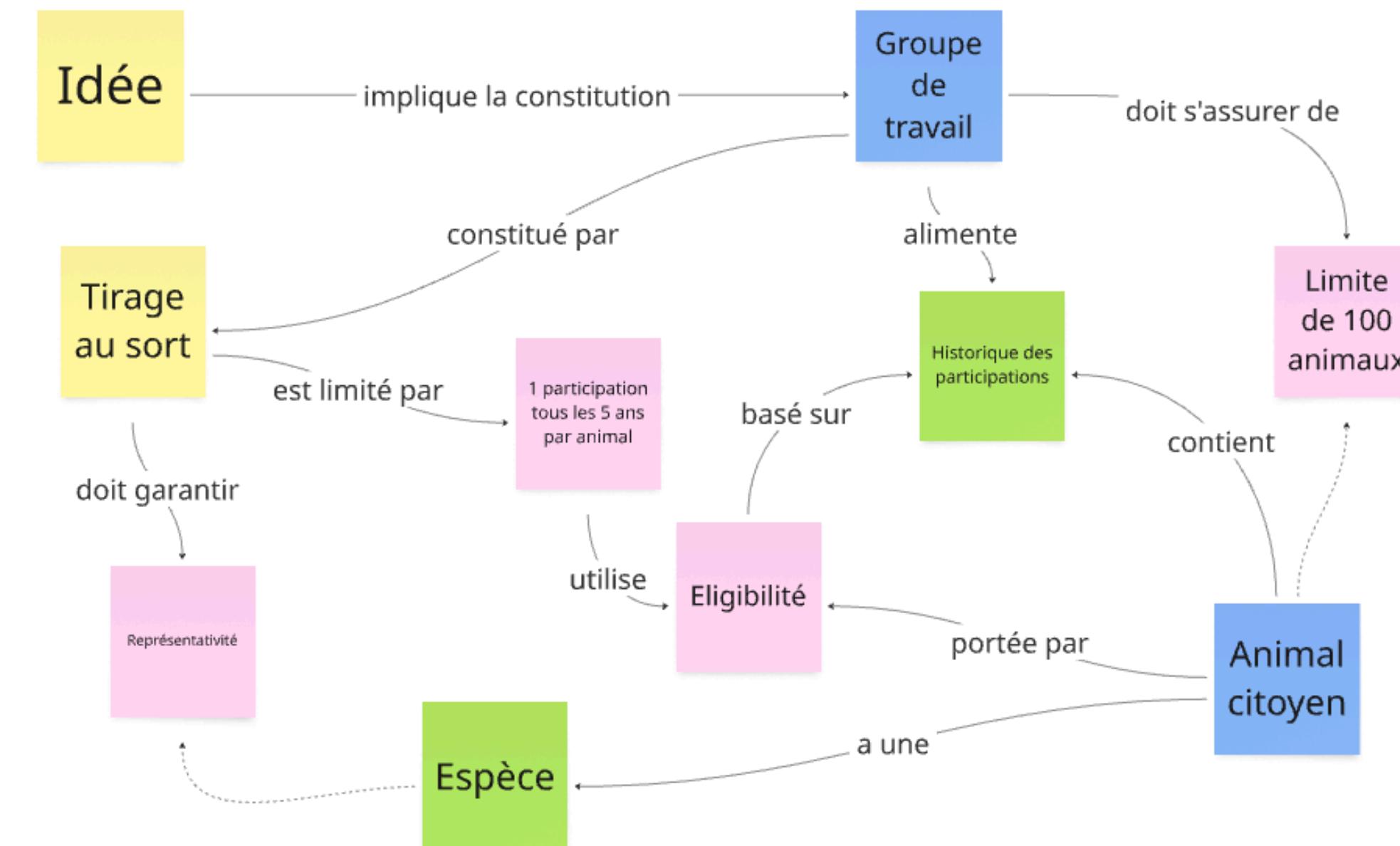
Value Object ou Entité ?

Quid du groupe de travail ?



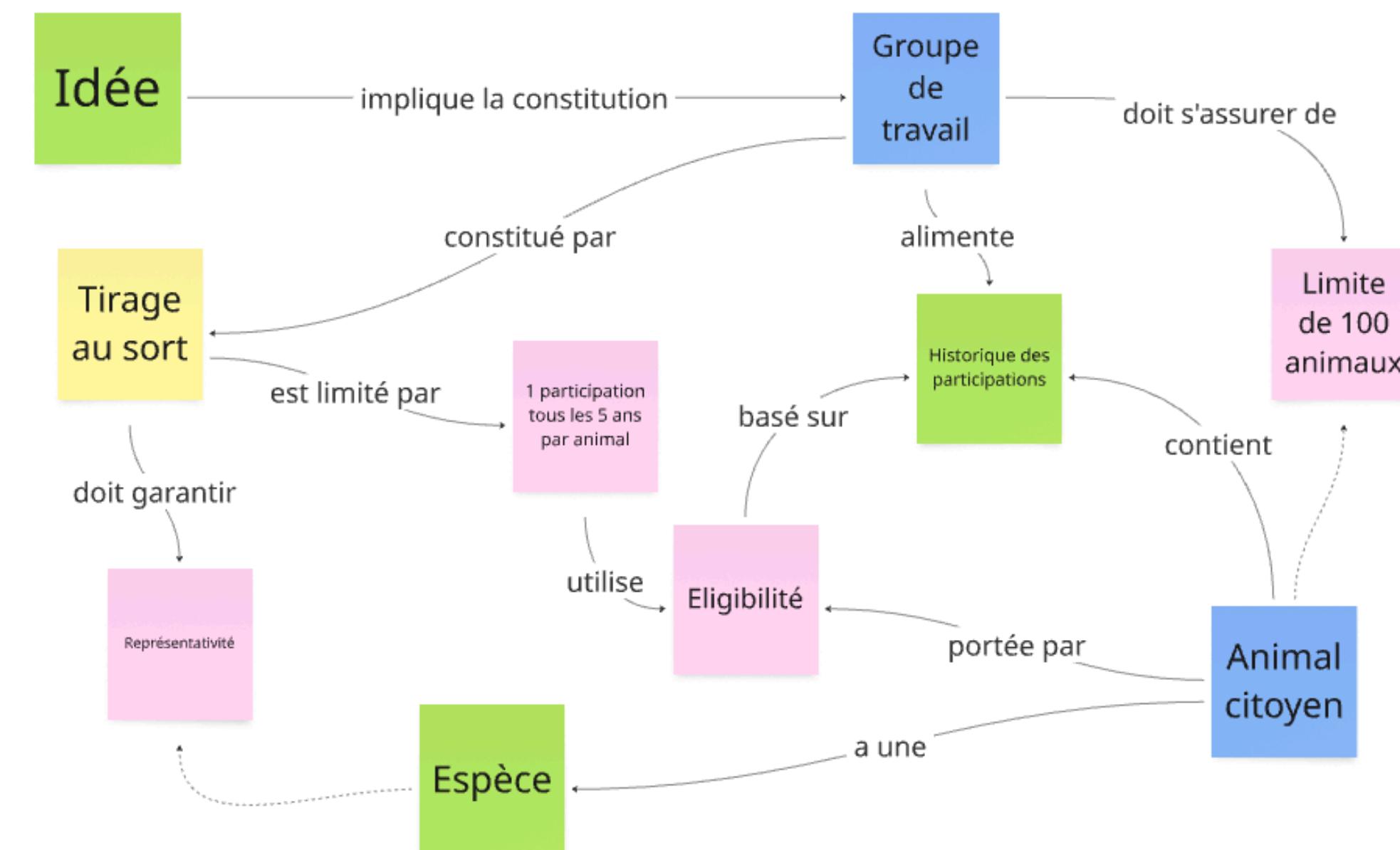
Value Object ou Entité ?

Un cas plus complexe: L'idée



Value Object ou Entité ?

Et le tirage au sort dans tout ça ?



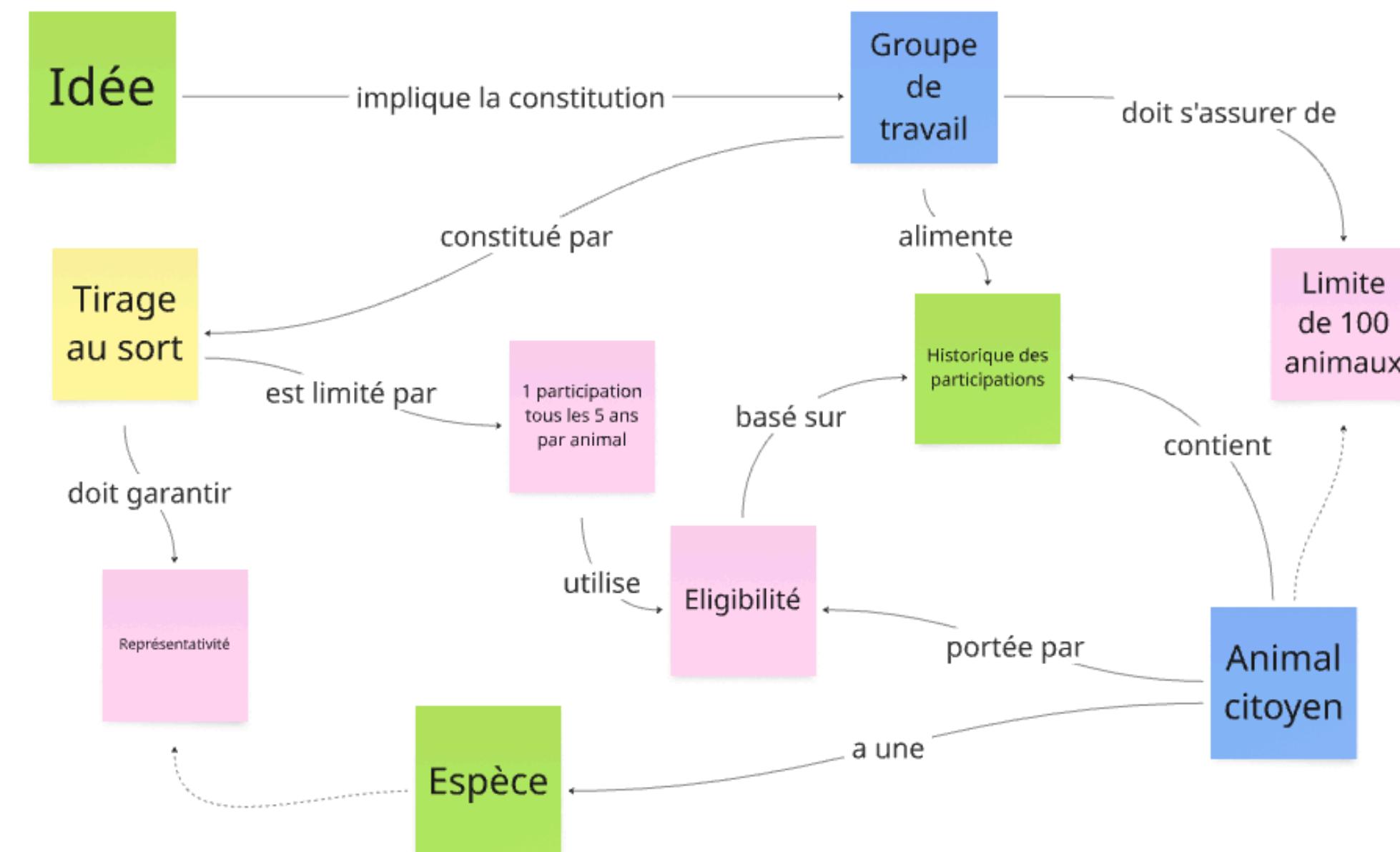


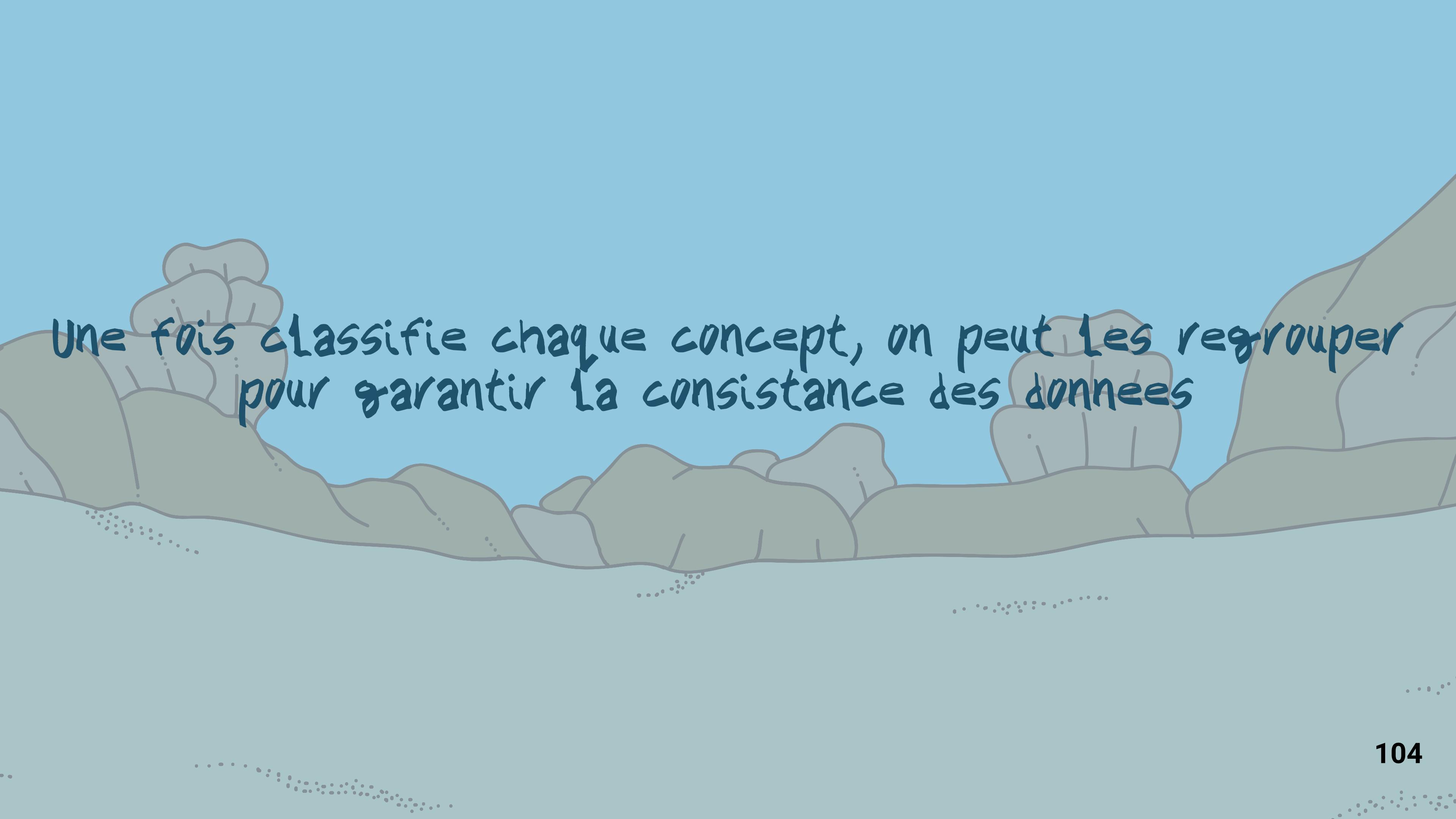
Un service du domaine porte un processus métier
qui ne peut pas naturellement être de la
responsabilité d'une entité ou d'un value object

SRP

Value Object ou Entité ?

Le tirage au sort est un service du domaine





Une fois classifiée chaque concept, on peut les regrouper pour garantir la consistance des données

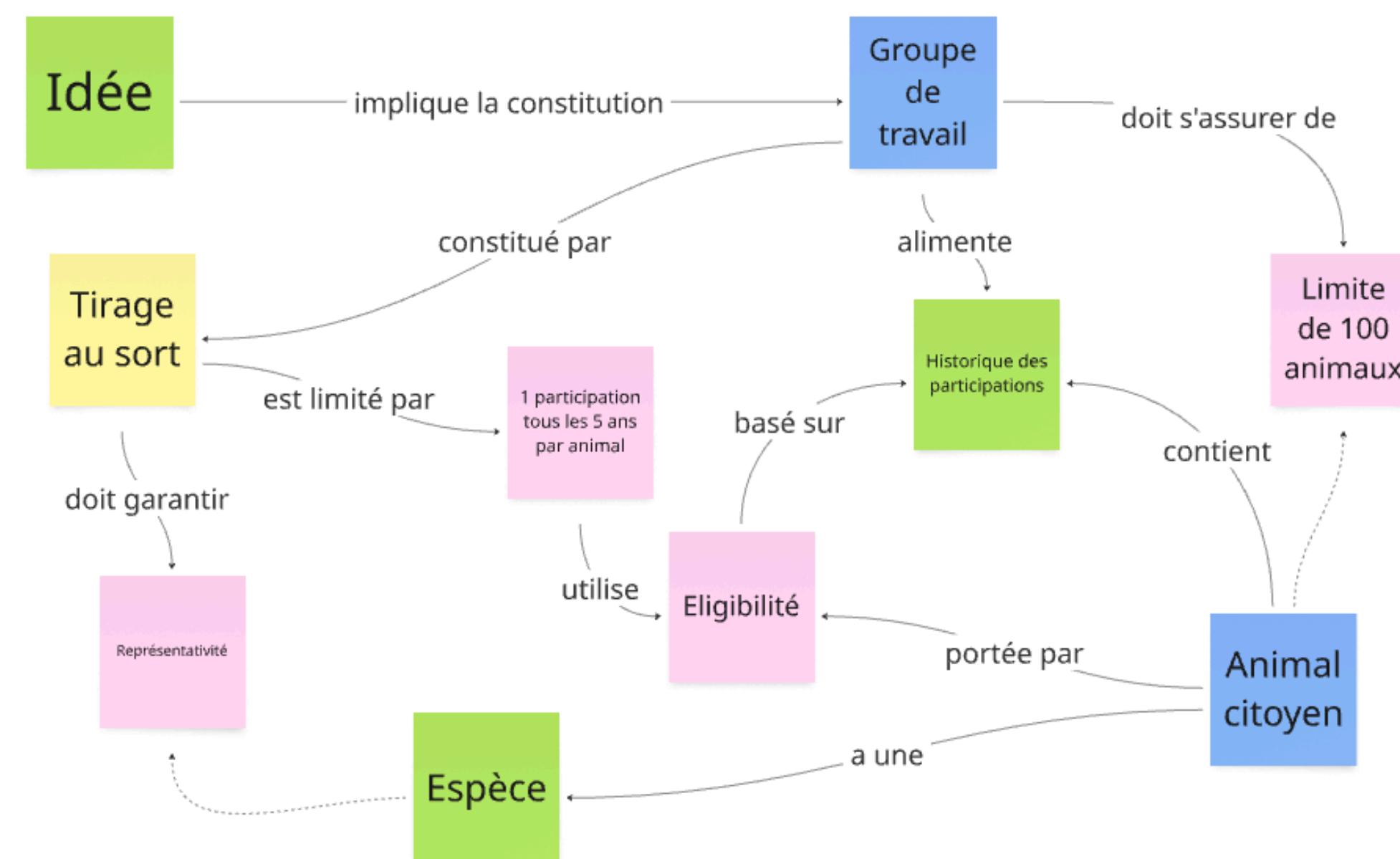


Les grappes d'objets, appelés **Agrégats**, gèrent les invariants métier pour garantir la cohérence entre tous les éléments de la grappe

Aggregate Root
Invariants Métier

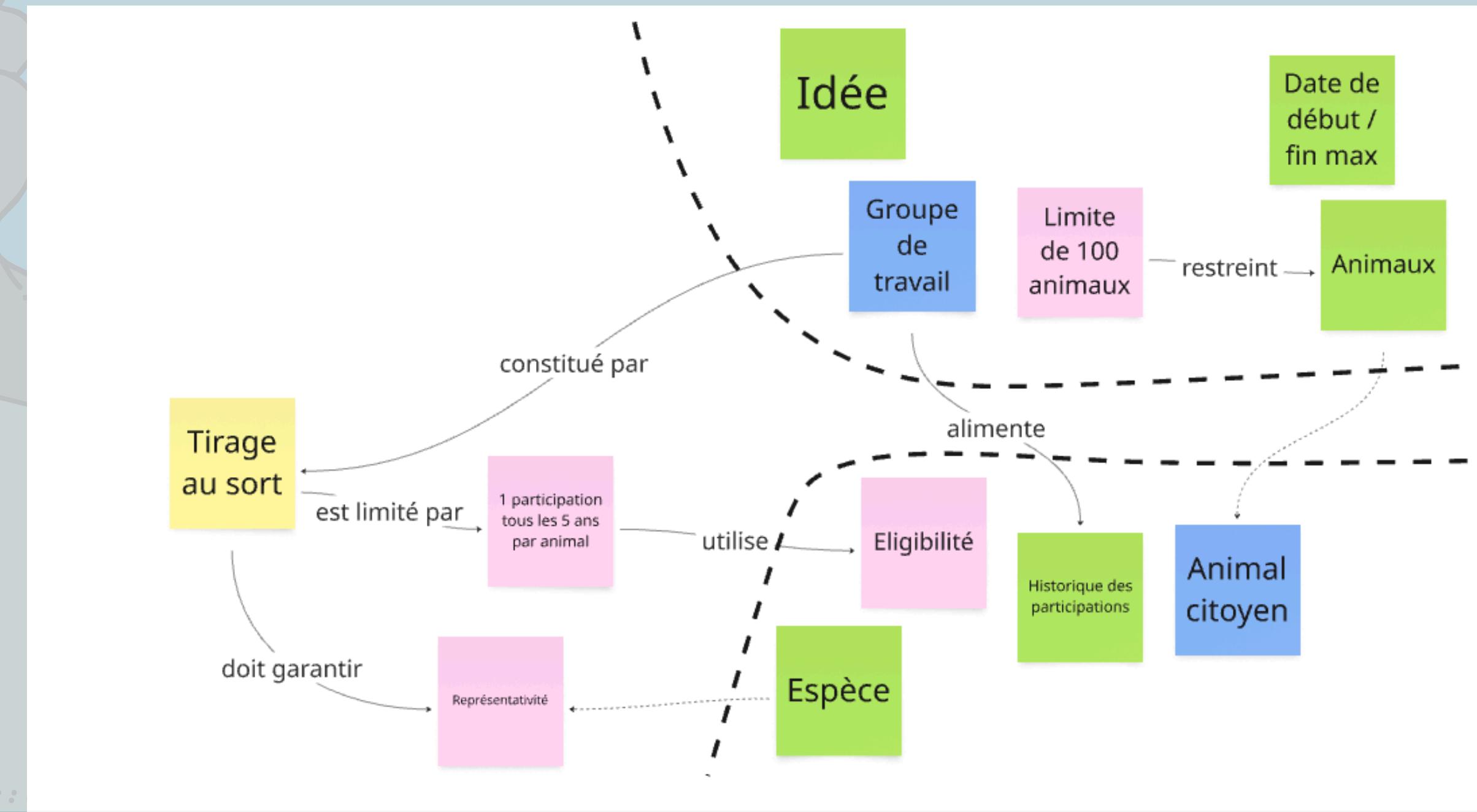
Quels agrégats ?

Comment regrouper les différents concepts pour garantir les invariants ?



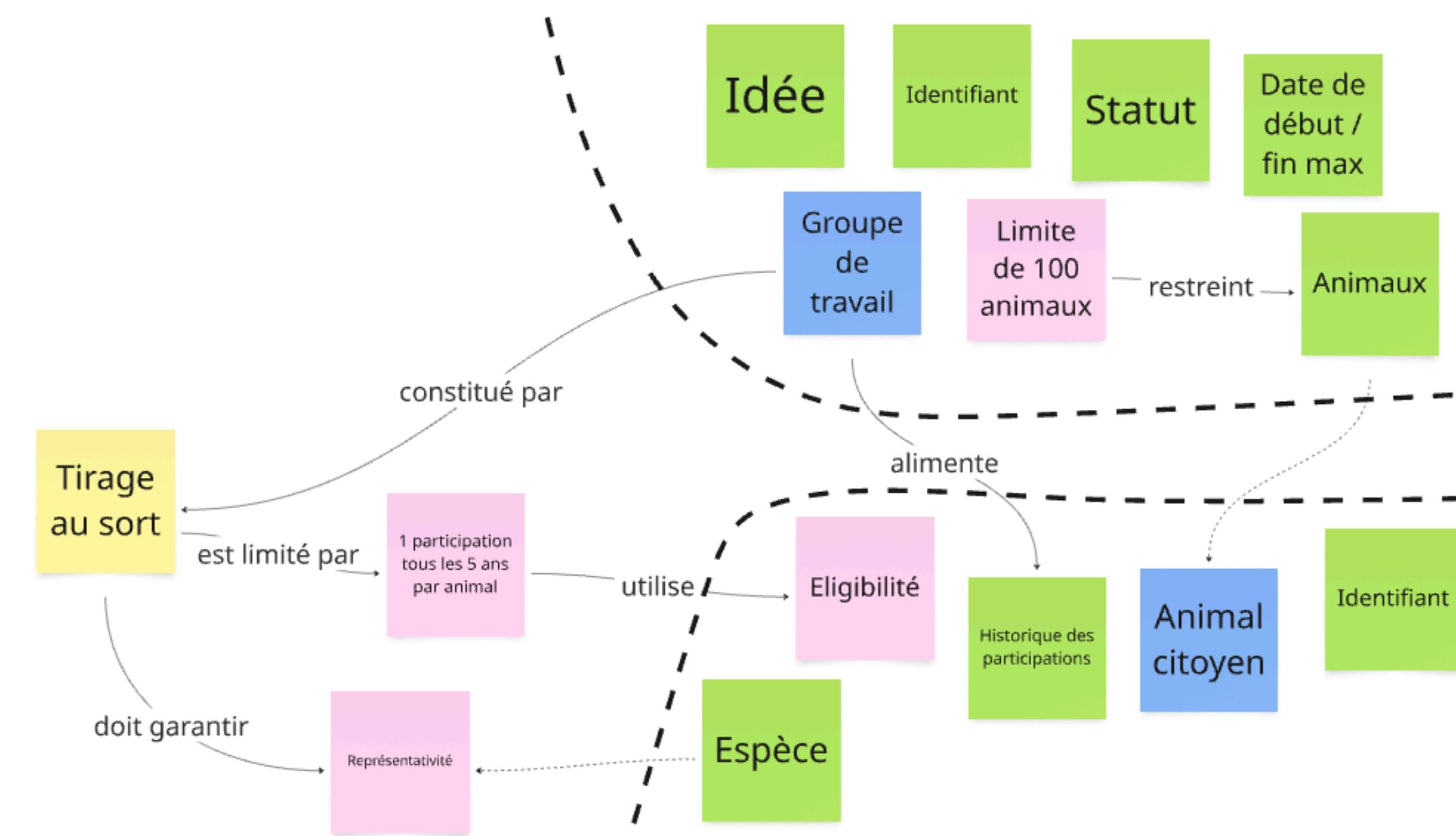
Quels agrégats ?

Groupe de travail / Animal citoyen



Quels agrégats ?

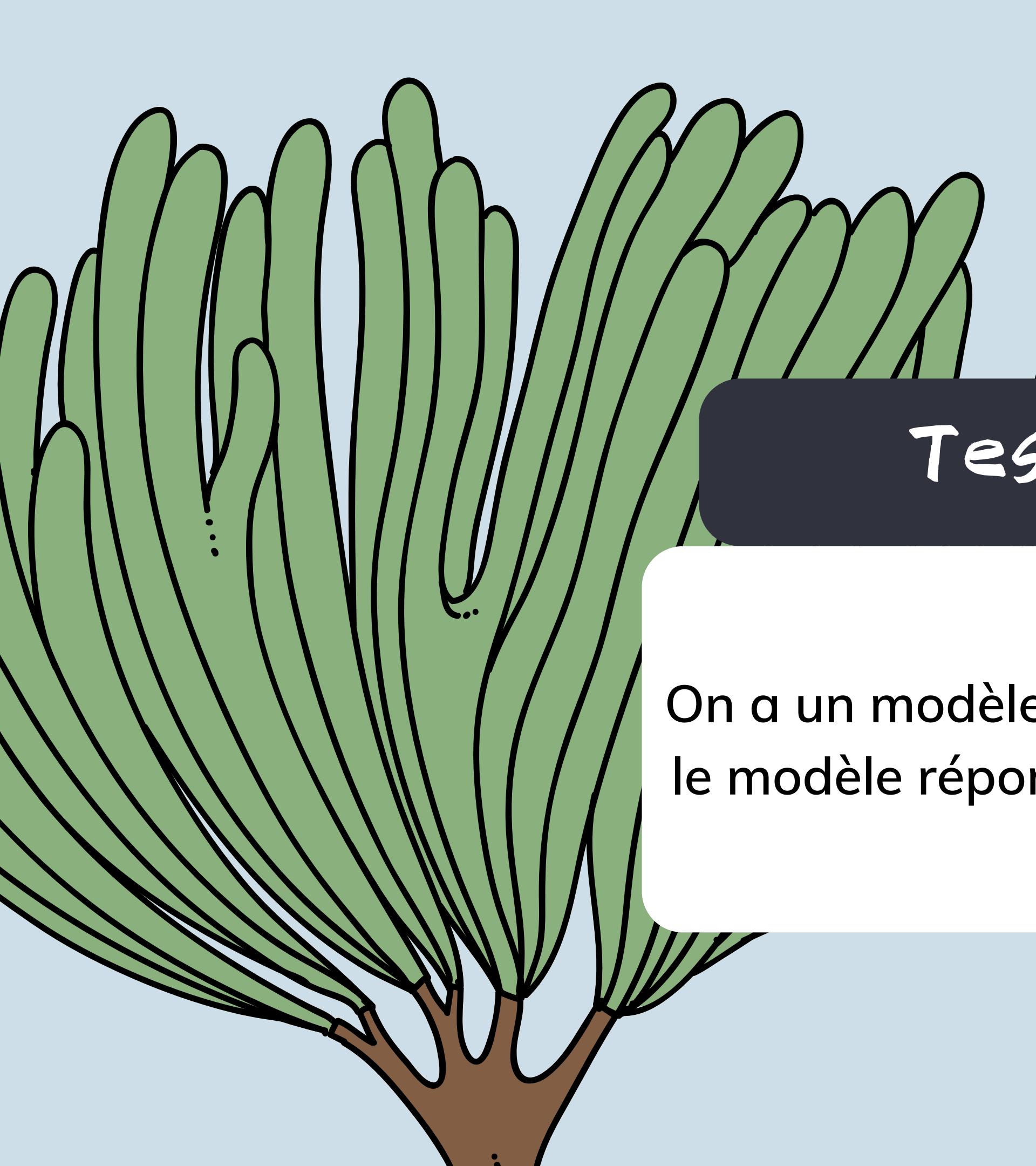
on ajoute les identifiants et la gestion du cycle de vie





Un agégrat peut contenir plusieurs entités, dont une seule peut être Aggregate Root.

Exemple: suivi de commande avec plusieurs lignes de commande qui peuvent être expédiées séparemment



Tester le modèle

On a un modèle, mais est-ce qu'on est garanti que le modèle répond aux différents use-cases métier ?

Use cases métier

- On sélectionne les animaux éligibles pour un nouveau tirage au sort.
- A la constitution du groupe de travail, on rajoute une participation à chaque animal.
- Les animaux tirés au sort sont rajoutés au groupe de travail.

Use Case 1

On sélectionne les animaux éligibles pour un nouveau tirage au sort.

on sélectionne les animaux éligibles pour un nouveau tirage au sort.

```
class AnimalCitoyen {  
  
    historiqueDesParticipations: Participation[]  
  
    estEligible(): boolean {  
        for(participation in historiqueDesParticipations) {  
            if(participation.bloqueEligibilite()) {  
                return false  
            }  
        }  
        return true  
    }  
}
```

On sélectionne les animaux éligibles pour un nouveau tirage au sort.

```
class AnimalCitoyen {  
  
    historiqueDesParticipations: Participation[]  
  
    estEligible(): boolean {  
        for(participation in historiqueDesParticipations) {  
            if(participation.bloqueEligibilite()) {  
                return false  
            }  
        }  
        return true  
    }  
}
```

```
class Participation {  
  
    idGroupeTravail: string  
    dateDebutParticipation: date  
  
    bloqueEligibilite(): boolean {  
        return new Date() - dateDebutParticipation < Duration("5 years")  
    }  
}
```

on sélectionne les animaux éligibles pour un nouveau tirage au sort.

```
class TirageAuSort {  
  
    getEligibleAnimauxCitoyens() : AnimalCitoyen[] {  
        ... ?  
    }  
}
```



On va avoir besoin de récupérer les données d'une base de stockage. Comment on fait ça en DDD ?

on sélectionne les animaux éligibles pour un nouveau tirage au sort.

```
class TirageAuSort {  
  
    repository: AnimauxCitoyenRepository  
  
    getEligibleAnimauxCitoyens() : AnimalCitoyen[] {  
        return repository.findAll()  
            .filter(animal => animal.estEligible())  
    }  
}
```

on sélectionne les animaux éligibles pour un nouveau tirage au sort.

```
class TirageAuSort {  
    repository: AnimauxCitoyenRepository  
  
    getEligibleAnimauxCitoyens(): AnimalCitoyen[] {  
        return repository.findAll()  
            .filter(animal => animal.estEligible())  
    }  
}  
  
class AnimauxCitoyenRepository {  
    findAll() {  
        SELECT * FROM ANIMAUX_CITOYEN  
        LEFT JOIN PARTICIPATION  
    }  
}
```



S'il y a de nombreux animaux en base, ça va être compliqué de tous les charger en mémoire...

on sélectionne les animaux éligibles pour un nouveau tirage au sort.

```
class TirageAuSort {  
  
    repository: AnimauxCitoyenRepository  
  
    getEligibleAnimauxCitoyens() : AnimalCitoyen[] {  
        return repository.findAllEligibles()  
    }  
}
```

```
class AnimauxCitoyenRepository {  
  
    findAllEligibles() {  
        SELECT * FROM ANIMAUX_CITOYEN  
        LEFT JOIN PARTICIPATION  
        WHERE DATE() - MAX(PARTICIPATION_DATE) > 5 YEARS  
    }  
}
```



Mais la règle d'éligibilité est présente 2 fois dans le code...

```
class Participation {  
  
    idGroupeTravail : string  
    dateDebutParticipation : date  
  
    bloqueEligibilite() : boolean {  
        return new Date() - dateDebutParticipation < Duration("5 years")  
    }  
  
}
```

```
class AnimauxCitoyenRepository {  
  
    findAllEligibles() {  
        SELECT * FROM ANIMAUX_CITOYEN  
        LEFT JOIN PARTICIPATION  
        WHERE DATE() - MAX(PARTICIPATION_DATE) > 5  
        YEARS  
    }  
  
}
```



Premiere solution envisagée

1) on ajoute le champ **eligible** à la table animaux_citoyens

id_animal	especie	+ eligible
1	lion	false
2	gazelle	true
3	tortue	false

2) un cron SQL qui passe tous les jours met à jour ce champ en fonction des participations

3) la méthode estEligible retourne ce champ

classe AnimalCitoyen

```
estEligible() : boolean {  
    return this.eligible  
}
```



Cela fonctionne, mais la règle d'éligibilité n'existe que dans le script SQL et plus dans le domain model

```
UPDATE ANIMAUX_CITOYEN
SET eligible = CASE
    WHEN DATE() - INTERVAL 5 YEAR < last_participation_date
    THEN FALSE
    ELSE TRUE
END
FROM (
    SELECT ANIMAUX_CITOYEN.id AS animal_id,
    MAX(PARTICIPATION.PARTICIPATION_DATE) AS
    last_participation_date
    FROM ANIMAUX_CITOYEN
    LEFT JOIN PARTICIPATION ON ANIMAUX_CITOYEN.id =
    PARTICIPATION.animal_id
    GROUP BY ANIMAUX_CITOYEN.id
) AS sub
WHERE ANIMAUX_CITOYEN.id = sub.animal_id
```

Deuxième solution envisagée

1) on ajoute le champ **eligible** à la table

animaux_citoyens

id_animal	especie	+ eligible
1	lion	false
2	gazelle	true
3	tortue	false

2) un cron typescript qui passe tous les jours met à jour ce champ en fonction des participations, en utilisant notre domain model

3) la méthode `estEligible` reste inchangée

classe AnimalCitoyen

```
estEligible() : boolean {  
    for(participation in historiqueDesParticipations) {  
        if(participation.bloqueEligibilite()) {  
            return false  
        }  
    }  
    return true  
}
```

Deuxième solution envisagée

Dans ce cas, le champ “eligible” n'est pas utilisé par notre domain model mais uniquement par notre repository.

```
class UpdateEligibleCron {  
  
    animauxCitoyenRepository: AnimauxCitoyenRepository  
  
    updateEligibiliteForAllAnimaux() {  
        var animaux = animauxCitoyenRepository.findAll()  
        for(animal in animaux) {  
            const eligible = animal.estEligible()  
            animauxCitoyenRepository.setEligibilite(animal.id, eligible)  
        }  
    }  
}
```

Deuxième solution envisagée

```
class AnimauxCitoyenRepository {  
  
    findAll() {  
        SELECT * FROM ANIMAUX_CITOYEN  
        LEFT JOIN PARTICIPATION  
    }  
  
    setEligible(animalId: AnimalCitoyenId, eligible: boolean) {  
        UPDATE ANIMAUX_CITOYEN  
        SET ELIGIBLE = $eligible  
    }  
}
```

on sélectionne les animaux éligibles pour un nouveau tirage au sort.

```
class TirageAuSort {  
  
    repository: AnimauxCitoyenRepository  
  
    getEligibleAnimauxCitoyens() : AnimalCitoyen[] {  
        return repository.findAllEligibles()  
    }  
}
```

```
class AnimauxCitoyenRepository {  
  
    findAllEligibles() {  
        SELECT * FROM ANIMAUX_CITOYEN  
        LEFT JOIN PARTICIPATION  
        WHERE ELIGIBLE = true  
    }  
}
```

Use Case 2

A la constitution du groupe de travail, on rajoute une participation à chaque animal.

A la constitution du groupe de travail, on rajoute une participation à chaque animal.

```
class TirageAuSort {  
  
    associerAnimauxAuGroupeDeTravail(groupeDeTravail: GroupeDeTravail, animaux: AnimalCitoyen[]) : void {  
        while(groupeDeTravail.statut !== Statut.CONSTITUE) {  
            var animal = this.selectionnerUnNouvelAnimal()  
            groupeDeTravail.ajouterParticipant(animal)  
        }  
    }  
  
    selectionnerUnNouvelAnimal(animaux: AnimalCitoyen[]) : AnimalCitoyen {}  
}
```



Hm... Ce n'est pas terrible d'exposer le statut du groupe de travail en direct au tirage au sort

A la constitution du groupe de travail, on rajoute une participation à chaque animal.

```
class TirageAuSort {  
  
    associerAnimauxAuGroupeDeTravail(groupeDeTravail: GroupeDeTravail, animaux: AnimalCitoyen[]) : void {  
        while(groupeDeTravail.nestPasConstitue()) {  
            var animal = this.selectionnerUnNouvelAnimal()  
            groupeDeTravail.ajouterParticipant(animal)  
        }  
    }  
  
    selectionnerUnNouvelAnimal() : AnimalCitoyen {}  
  
}
```

A la constitution du groupe de travail, on rajoute une participation à chaque animal.

```
class GroupeDeTravail {  
  
    statut: Statut  
  
    ajouterParticipant(animal: AnimalCitoyen) {  
        animal.ajouterParticipation(this)  
    }  
  
    nEstPasConstitue() : boolean {  
        return this.statut !== Statut.CONSTITUE  
    }  
}
```

A la constitution du groupe de travail, on rajoute une participation à chaque animal.

```
class AnimalCitoyen {  
    historiqueDesParticipations: Participation[]  
  
    estEligible(): boolean {}  
  
    ajouterParticipation(groupeDeTravail: GroupeDeTravail) {  
        if (!estEligible()) {  
            throw new Error("Impossible d'ajouter une participation: animal non éligible")  
        }  
        historiqueDesParticipations.push(new Participation(groupeDeTravail))  
    }  
}
```

Use Case 3

Les animaux tirés au sort sont rajoutés au groupe de travail.

Les animaux tirés au sort sont rajoutés au groupe de travail.

```
class GroupeDeTravail {  
  
    statut: Statut  
    animaux: AnimalCitoyen[]  
  
    ajouterParticipant(animal: AnimalCitoyen) {  
        animaux.push(animal)  
        animal.ajouterParticipation(this)  
    }  
}
```

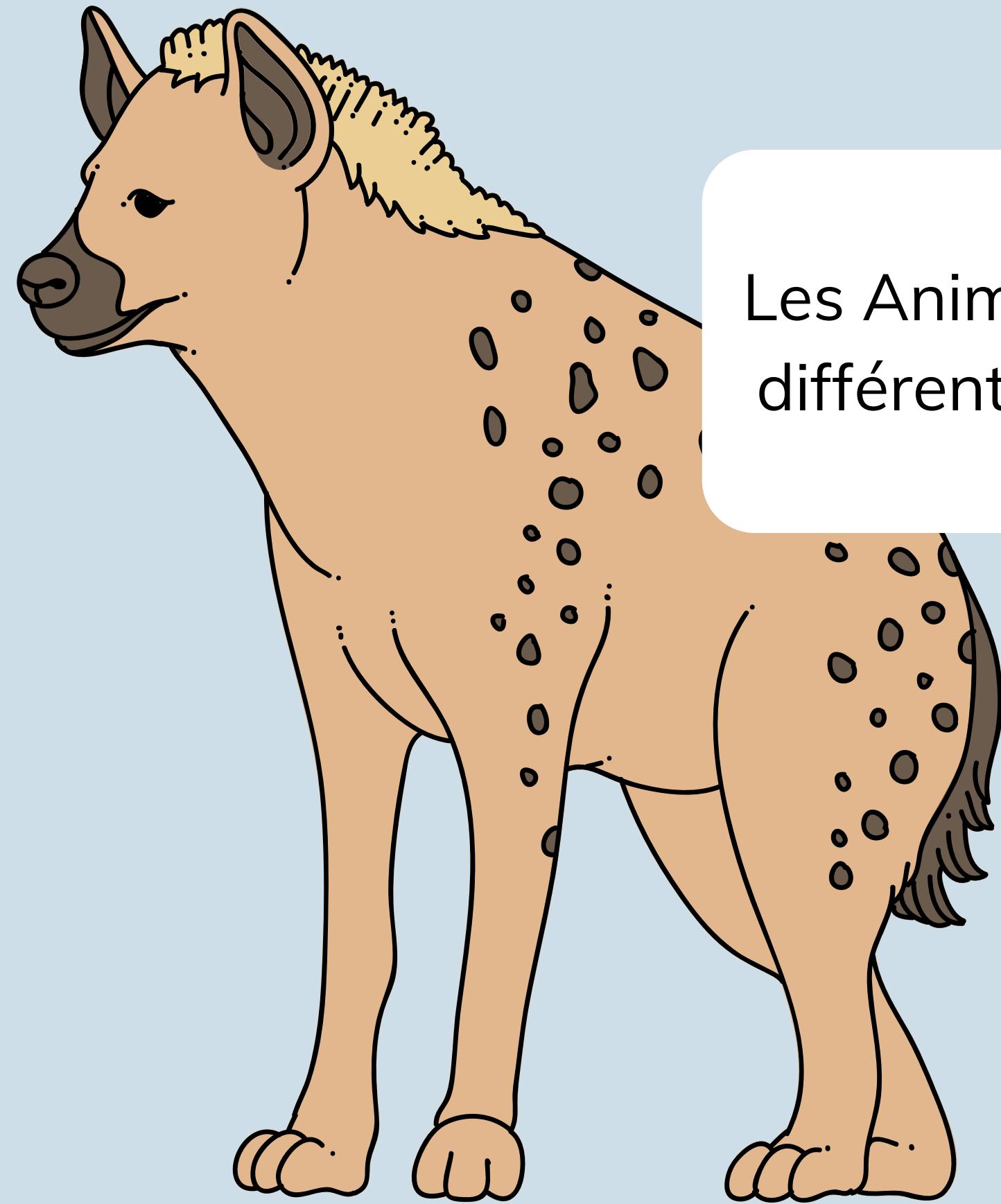


Dans le métier, on ne parle pas plutôt de participant ?
Peu importe que ça soit des animaux !

Ubiquitous Language
Context

Les animaux tirés au sort sont rajoutés au groupe de travail.

```
class GroupeDeTravail {  
  
    statut: Statut  
    participants: AnimalCitoyen[]  
  
    ajouterParticipant(participant: AnimalCitoyen) {  
        participants.push(participant)  
        participant.ajouterParticipation(this)  
    }  
}
```



Les Animaux et le Groupe de Travail sont 2 agrégats différents. On devrait référencer les animaux par ID.

Les animaux tirés au sort sont rajoutés au groupe de travail.

```
class AnimalCitoyenId {  
    id: string  
}
```

```
class GroupeDeTravail {  
  
    statut: Statut  
    participants: AnimalCitoyenId[]  
  
    ajouterParticipant(participant: AnimalCitoyen) {  
        participants.push(participant.id)  
        participant.ajouterParticipation(this)  
    }  
}
```



Mais un groupe de travail ne peut jamais avoir plus de 100 participants...

Les animaux tirés au sort sont rajoutés au groupe de travail.

```
class GroupeDeTravail {  
  
    statut: Statut  
    participants: AnimalCitoyenId[]  
  
    ajouterParticipant(participant: AnimalCitoyen) {  
        if(participants.length >= 100) {  
            throw new Error("Impossible d'ajouter l'animal", id, "le groupe est déjà plein")  
        }  
  
        participants.push(participant.id)  
        participant.ajouterParticipation(this)  
    }  
}
```

La seule manière d'ajouter un participant est de passer par l'agrégat *GroupeDeTravail*.



Le groupe de travail a un cycle de vie. Quand il atteint 100 animaux, il est constitué.

Les animaux tirés au sort sont rajoutés au groupe de travail.

```
class GroupeDeTravail {  
  
    participants: AnimalCitoyenId[]  
    statut: Statut  
  
    ajouterParticipant(participant: AnimalCitoyen) {  
        if(statut == Statut.CONSTITUE) {  
            throw new Error("Impossible d'ajouter l'animal", id, "le groupe est déjà constitué")  
        }  
  
        participants.push(participant.id)  
        participant.ajouterParticipation(this)  
  
        if(participants.length == 100) {  
            statut = Statut.CONSTITUE  
        }  
    }  
}
```

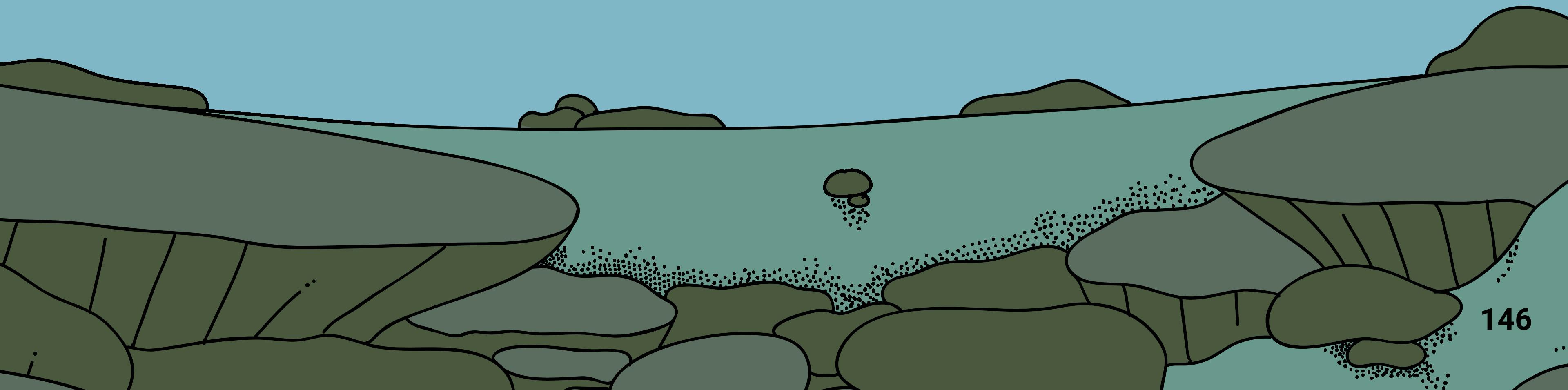


Le Bounded Context du Groupe de Travail démarre quand un groupe est constitué. Pour lui faire signifier, nous pouvons envoyer l'évènement pivot.

Les animaux tirés au sort sont rajoutés au groupe de travail.

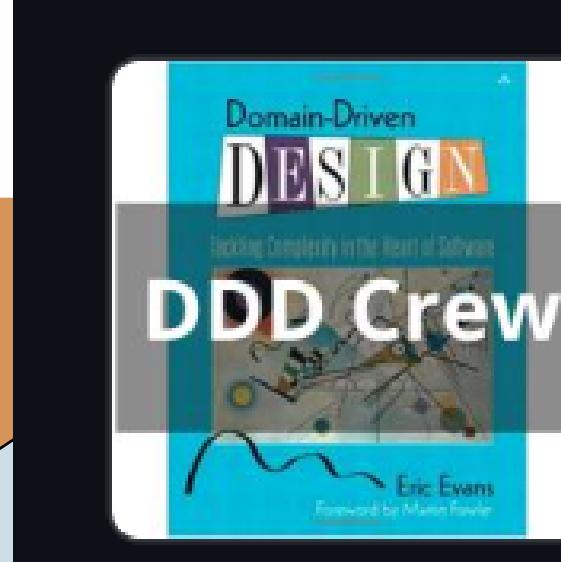
```
class GroupeDeTravail {  
  
    statut: Statut  
    participants: AnimalCitoyenId[]  
  
    ajouterParticipant(participant: AnimalCitoyen): GroupeDeTravailConstitueEvent | undefined {  
        if(statut == Statut.CONSTITUE) {  
            throw new Error("Impossible d'ajouter l'animal", id, "le groupe est déjà constitué")  
        }  
  
        participants.push(participant.id)  
        participant.ajouterParticipation(this)  
  
        if(participants.length == 100) {  
            statut = Statut.CONSTITUE  
            return new GroupeDeTravailConstitueEvent(this)  
        }  
  
        return  
    }  
}
```

Conclusion



DDD

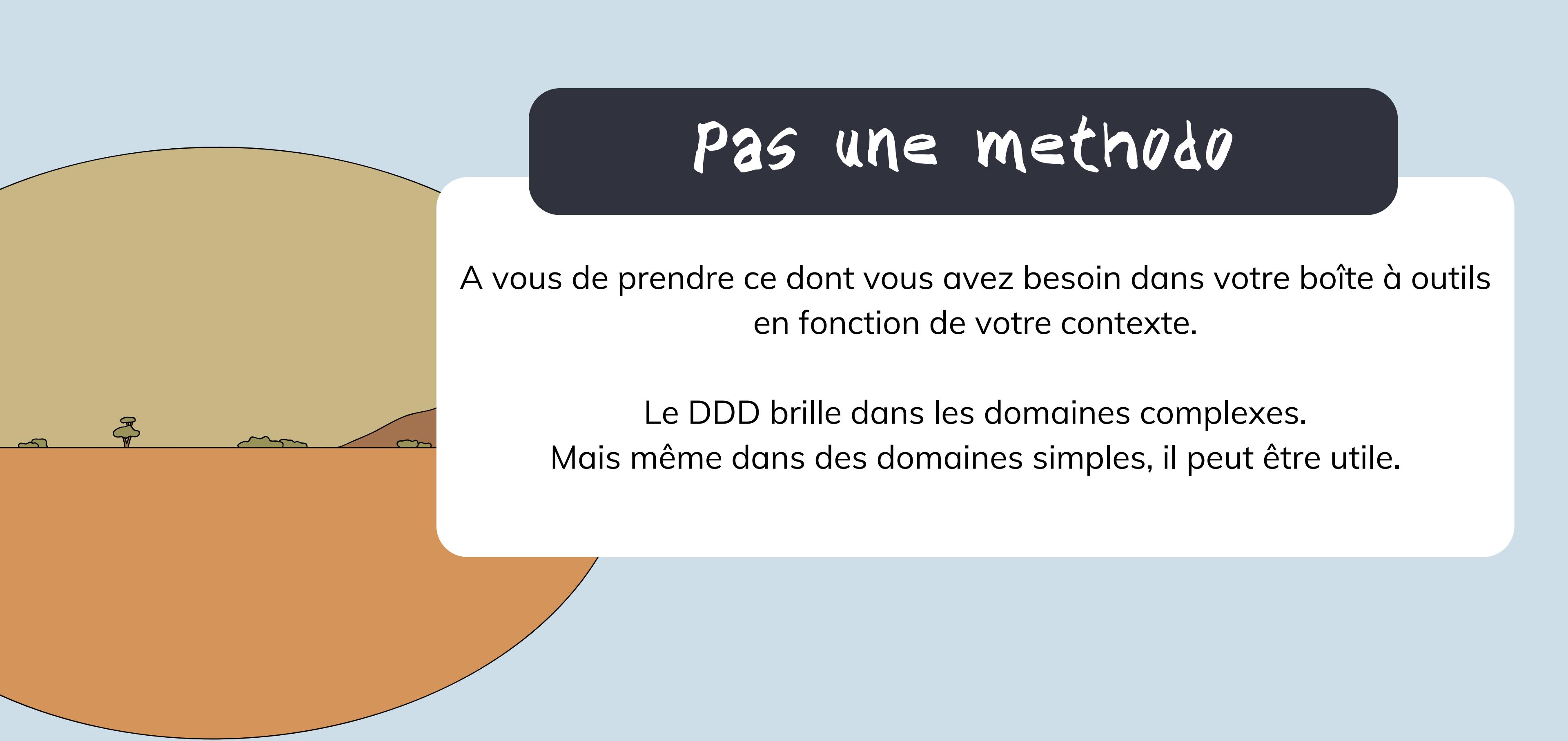
Le DDD apporte de nombreux outils, autant au niveau stratégique qu'au niveau tactique dans le code.



Domain-Driven Design Crew

4k followers

Worldwide



Pas une méthode

A vous de prendre ce dont vous avez besoin dans votre boîte à outils
en fonction de votre contexte.

Le DDD brille dans les domaines complexes.
Mais même dans des domaines simples, il peut être utile.



Une approche itérative

Vous ne pouvez pas avoir compris tout le domaine au début de votre travail de découverte, vous allez avoir besoin d'itérer avec les experts métier pour construire des modèles efficaces.

Le processus itératif nécessite également d'avoir un design souple, non imbriqué dans des considérations techniques.



Merci!

