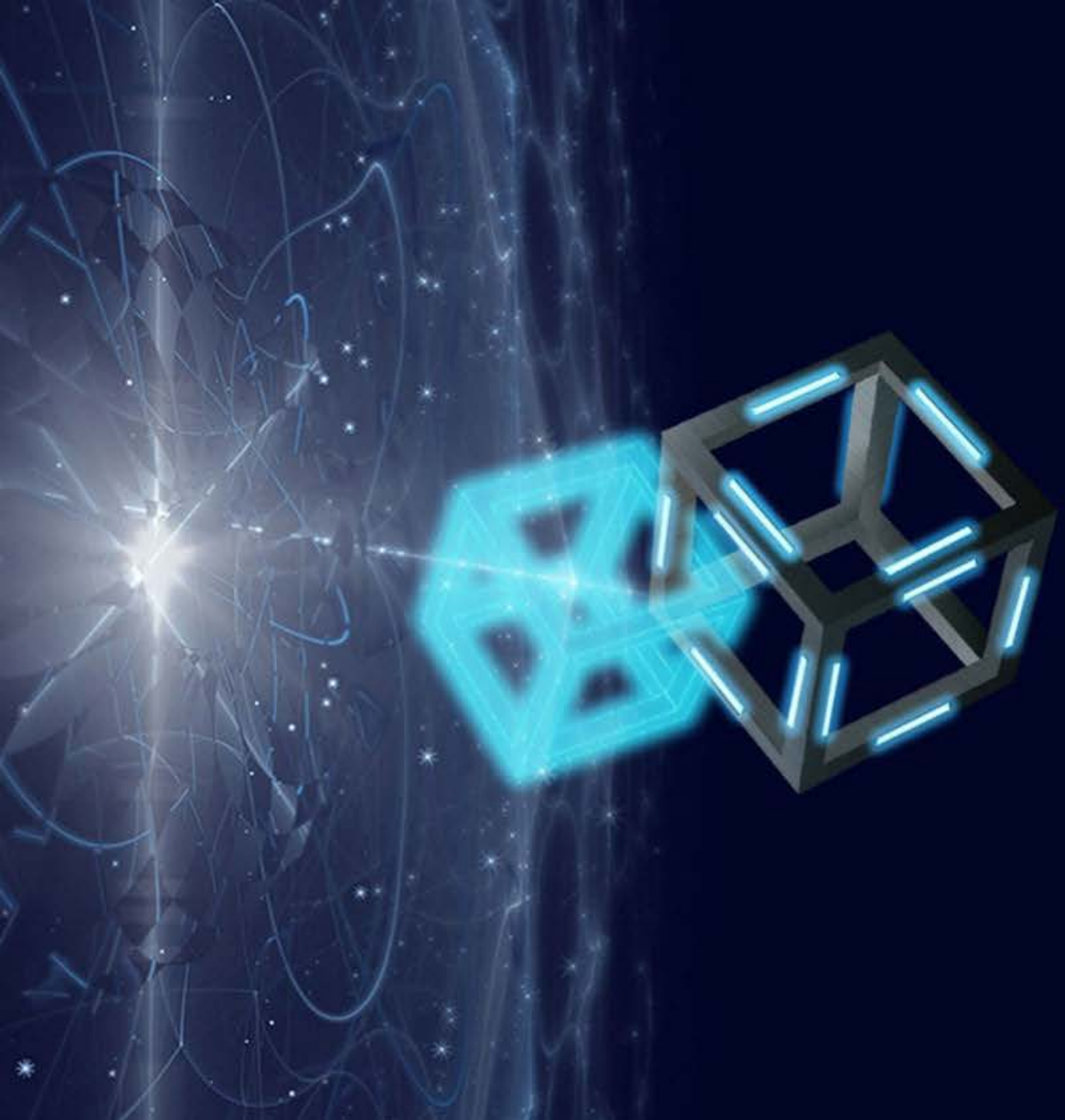
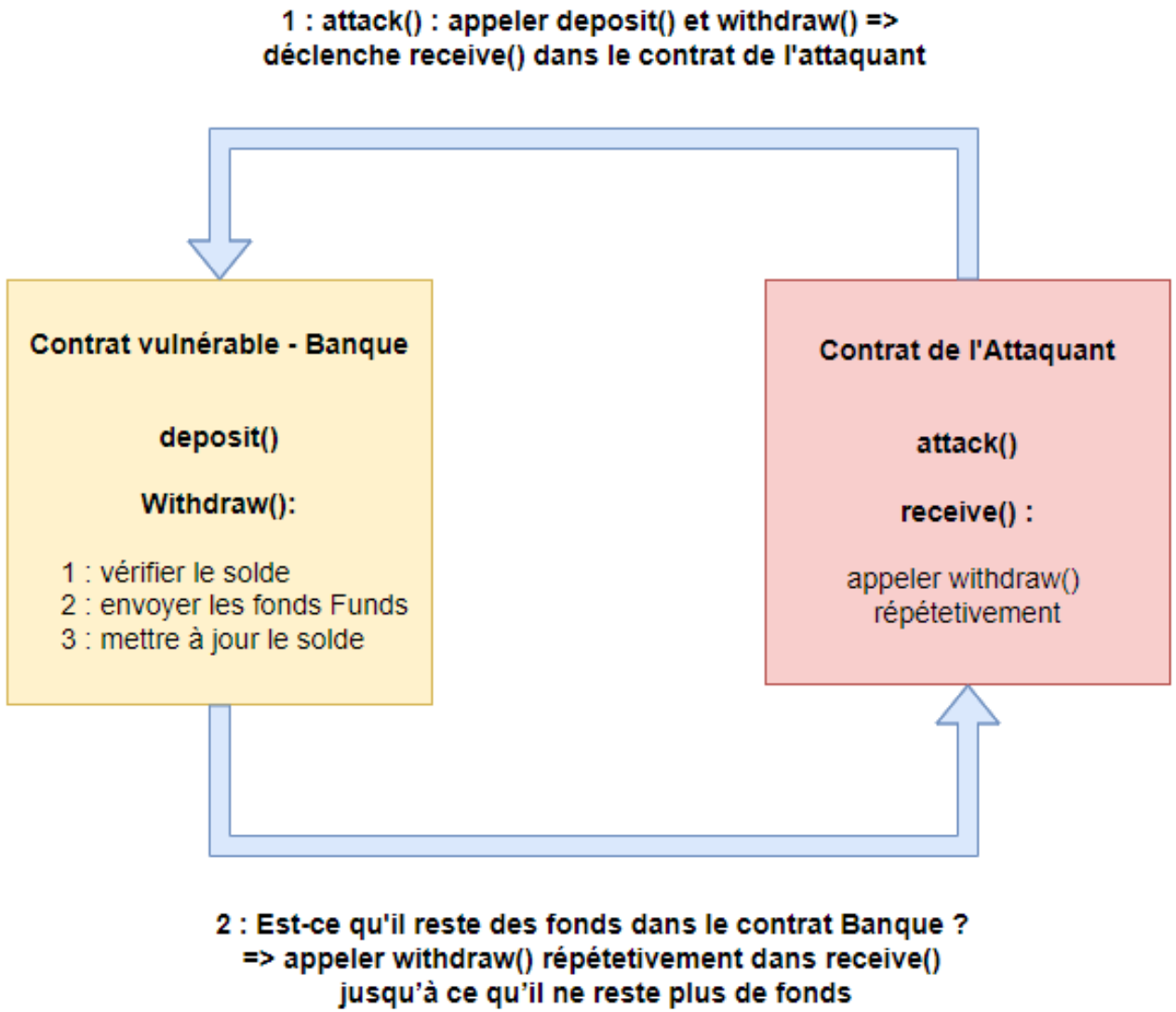

Développer et déployer des
contrats intelligents sur Ethereum

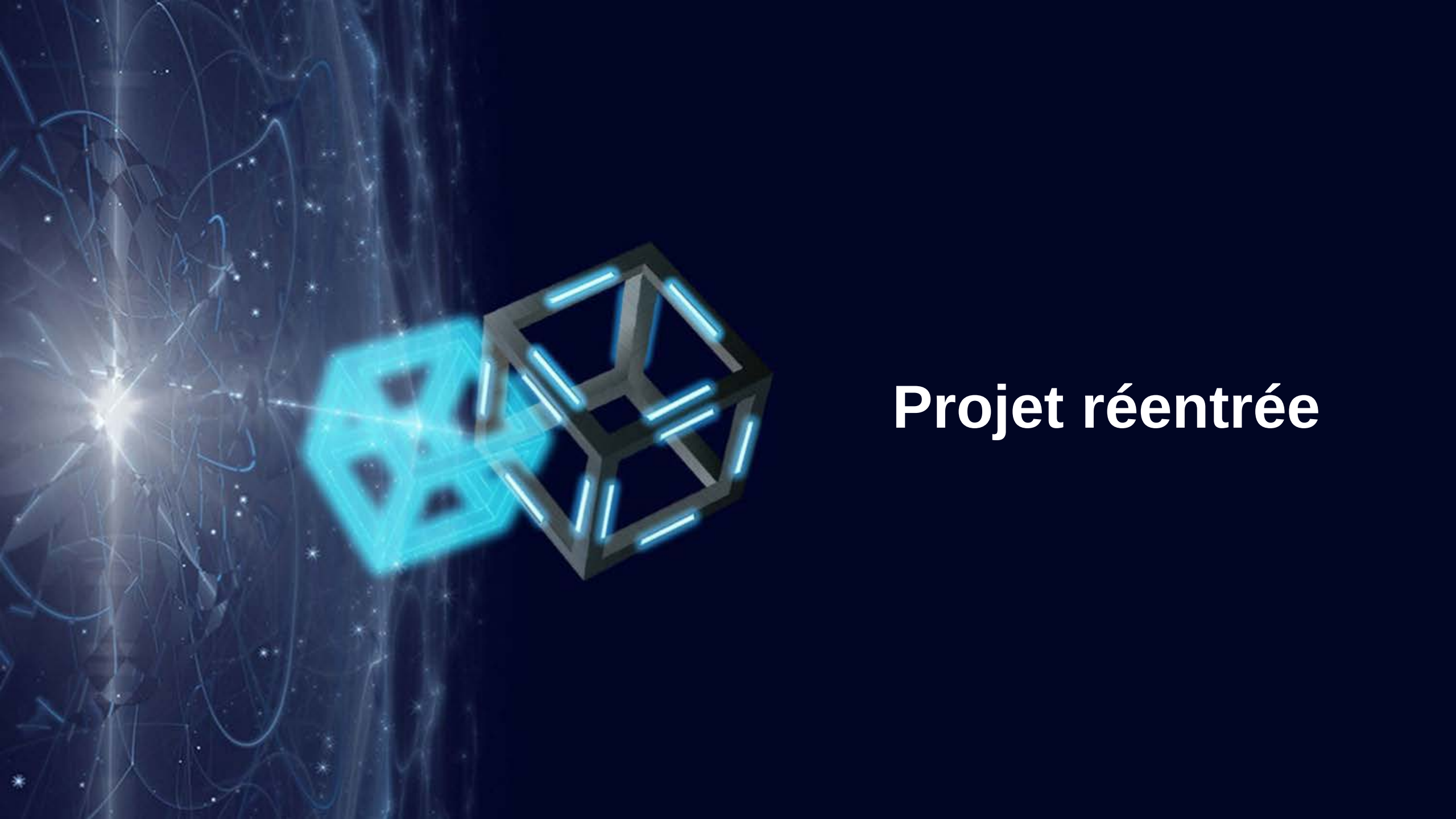


# Les attaques de réentrée

# Attaque de réentrée




- l'ATTAQUANT dépose des fonds à la BANQUE puis appelle retrait()
- L BANQUE vérifie si l'ATTAQUANT possède les fonds, puis il transfère les fonds à l'ATTAQUANT
- Lorsque l'ATTAQUANT reçoit les fonds, il exécute une fonction de réception (receive) qui appelle à nouveau retrait() sur BANQUE avant de pouvoir mettre à jour le solde
- Ce processus se poursuit jusqu'à tous les fonds aient été retirés de la BANQUE



# Projet réentrée

# Projet réentrée - besoins

**Contrat Banque:**

- Gérer les soldes des utilisateurs
- Déposer des fonds
- Retirer des fonds
  - Le montant déposé doit être >= fonds à retirer
  - Transfert de fonds à l'utilisateur
  - Mettre à jour le solde de l'utilisateur

**Contrat de l'Attaquant :**

- Créer le contrat de la Banque dans le constructeur
- Dans le contrat de l'Attaquant, appeler : ***deposit() and withdraw()***
- ***receive():*** vérifier s'il reste des fonds dans le contrat Banque => appeler ***withdraw()***

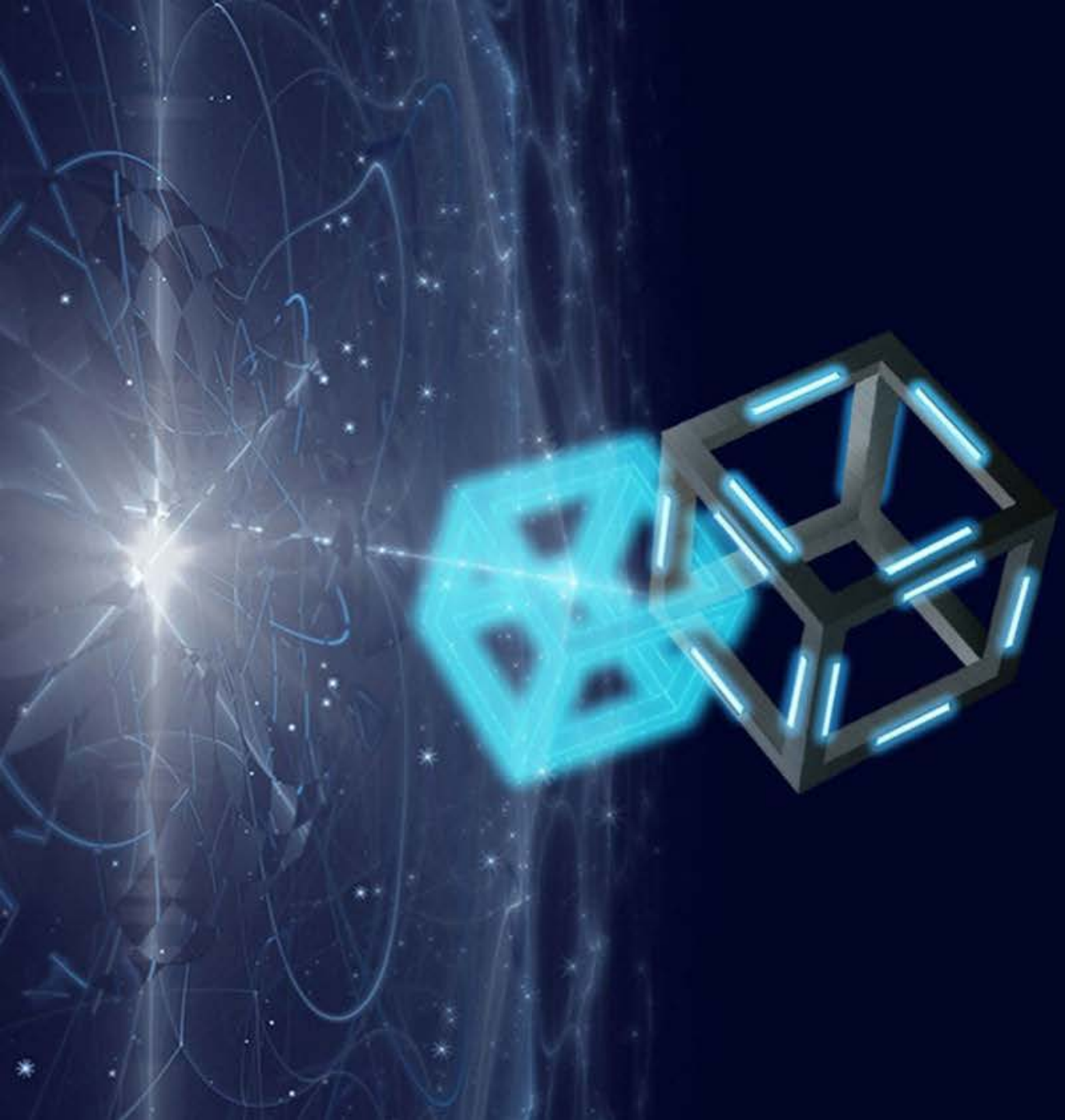
# Protection contre une attaque de réentrée

- Mauvaise conception du contrat vulnérable : vérifier le solde => envoyer les fonds => mettre à jour solde
- ***Checks-Effects-Interaction :*** modèle de conception recommandé pour le codage des contrats intelligents :
  - 1 : valider les données entrées (vérifier le solde)
  - 2 : apporter des changements aux variables d'état (mise à jour du solde)
  - 3 : interagir avec d'autres contrats (envoyer des fonds)
- Ethereum DAO hack 2016 : 3,6 M ETH (60 millions de dollars) ont été volés => Pour récupérer les fonds, une "hard-fork" a été initiée. La blockchain originale est maintenant ***Ethereum Classic***

# Projet réentrée - Exercice

**Protéger le contrat Banque :**

- Hériter le contrat Banque de ***ReentrancyGuard*** (OpenZeppelin)
- Ajouter le modificateur ***nonReentrant*** à la fonction withdraw()
- Utiliser le modèle de conception : "**Check-Effects-Interaction**"
- En utilisant ***transfer*** au lieu de ***call*** cette attaque n'aurait pas été possible. Cependant, la méthode ***transfer*** limite la consommation de gaz dans la fonction de réception à 2300 unités de gaz => cela pourrait casser certains contrats existants, car le coût du gaz peut changer pour différents opcodes, donc il n'est plus recommandé d'utiliser la méthode ***transfer***.



# DAO -

***organisation autonomes décentralisé***

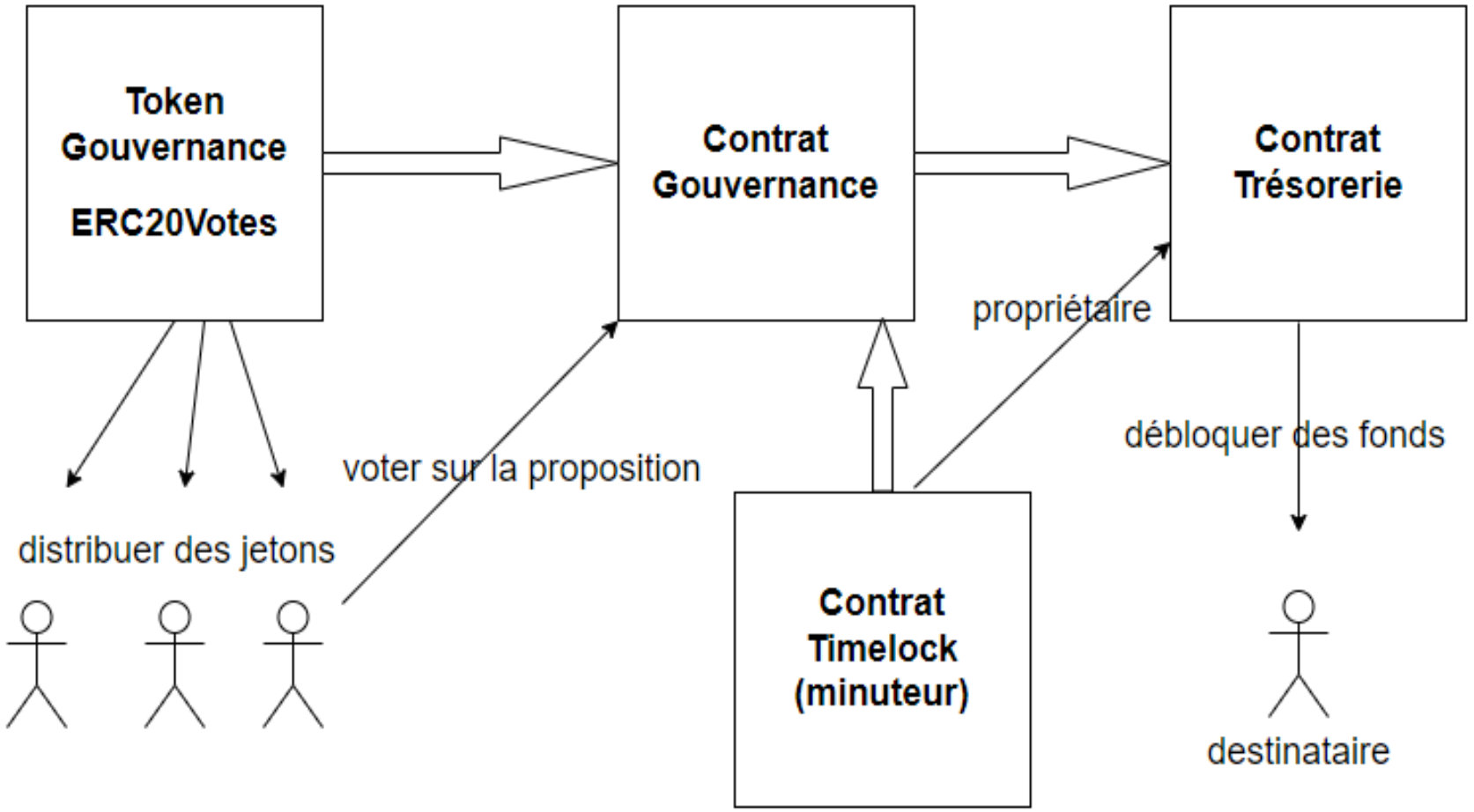
# C'est quoi, une DAO ?

- Organisation sans autorité centrale géré par la communauté
- Autonome et transparent : les contrats intelligents exécutent les décisions convenues et à tout moment, les propositions, les votes et le code des contrats peuvent être audités publiquement
- Gouverné entièrement par ses membres qui prennent collectivement des décisions critiques
- Les membres de la communauté créent des propositions sur diverses actions à prendre, puis ils votent sur chaque proposition
- Les propositions qui atteignent un certain niveau de consensus prédéfini sont ensuite acceptées et appliquées par des contrats intelligents

# Fonctionnement d'une DAO

- Une fois que les contrats (avec les règles défini par la communauté) sont déployé sur la blockchain, la DAO doit déterminer comment obtenir du financement
- Généralement réalisé par l'émission de jetons qui seront vendu (en partie) pour lever des fonds et remplir la trésorerie de la DAO
- Les détenteurs des jetons ont des droits de vote, qui sont proportionnels au nombre des jetons détenu par chaque individu
- Une fois le financement terminé, la DAO peut devenir opérationnel. Des propositions peuvent être créées, les membres peuvent voter et les propositions retenues peuvent être exécutées
- Dans certains protocoles, les détenteurs de jetons peuvent voter pour distribuer des fonds de trésorerie vers des corrections de bugs, des mises à jour du système... Cette approche permet aux développeurs de se joindre à une équipe ad hoc et de recevoir une rémunération

# Les contrats du projet




- Distribuer des jetons de gouvernance - par vente publique…
- Créer une proposition sur le contrat de gouvernance
- Les détenteurs de jetons de gouvernance peuvent voter sur cette proposition
- Si le vote est positif, la proposition est exécutée et les fonds sont envoyés au bénéficiaire
- Le "timelock" impose un délai après lequel les fonds sont libérés

# Token gouvernance

**ERC20Vote :**

- Extension de l’ERC20 pour gérer le vote et la délégation
- Conserve un historique (checkpoints) du pouvoir de vote de chaque compte

# Contrat Timelock - TimelockController

- Le contrat du Trésor détient les fonds pour la proposition.
- Lorsque le vote est adopté, le contrat Timelock (qui est le propriétaire du contrat Trésor) exécute la proposition (après un certain délai – timelock) et transfère les fonds au destinataire.

**Contrôle d’accès de TimelockController:**

- Le rôle de proposant (accordé au gouverneur) est en charge de mettre des opérations en file d’attente
- Le rôle d’exécuteur (peut également être accordé au gouverneur) est en charge de l’exécution des opérations déjà disponibles
- Le rôle d'administrateur(accordé automatiquement au Timelock) peut accorder et révoquer les deux rôles précédents

# Les opérations / propositions

Une opération est une transaction (ou un ensemble des transactions) qui peut être créé par un membre de la communité (la proposition). Après, elle doit être planifié (mis en file d'attente) par un proposant et exécuté (après un délai) par un exécuteur.

**Les opérations sont identifiées par un Id unique et suivent un cycle de vie spécifique :**

- ***Pending*** : après la création d'une proposition
- ***Active*** : vote en cours - la période de vote n'est pas encore passée
- ***Queued*** : l'opération était mise en file d'attente
- **Executed** : l'opération était exécutée

Le proposant appelle "***queue***" => déplace l'opération de ***Active*** à ***Queued*** => la minuterie est lancée => une fois la minuterie expirée, l'exécuteur peut exécuter l'opération ( envoyer transaction(s) ) => l'opération passe à l'état ***Executed***

# Création d’une proposition

Une proposition est une série d'action(s) que le contrat du gouvernance exécutera s’il est adopté. Chaque action consiste d'une adresse cible, le calldata (appel de fonction encodée), le montant d’Ether à inclure et une description.

**Exécution de la proposition :**

- Appeler la fonction ***propose*** du contrat gouvernance avec les paramètres requis :
  **await governor.propose([contractAddress], [0], [transferCalldata], “Give grant to team”)**
- Les délégués peuvent voter
- Une fois la période de vote terminée, si le quorum est atteint, la proposition est considérée comme réussie et peut être mise en attente d’exécution => appeler la fonction ***queue*** du gouvernance
- Après le délai (s’il y en a un), la proposition peut être exécutée => appeler la fonction ***exécute*** du contrat de gouvernance

# Contrat Gouvernance – les modules requis

- ***Governor*** : Contient la logique de base. On peut fournir un nom pour la DAO dans le constructeur
- ***GovernorVotes*** : Détermine le pouvoir de vote d'un compte en fonction de son solde. On doit fournir l'adresse du jeton de gouvernance dans le constructeur
- ***GovernorCountingSimple*** : Offre 3 options aux électeurs : Pour, Contre et Abstention. Seuls les votes Pour et Abstention sont comptés pour le quorum
- ***GovernorVotesQuorumFraction*** : Définit le quorum en pourcentage de l'offre totale de jetons. On doit fournir le quorum souhaité dans le constructeur. La plupart des gouverneurs exiger un quorum de 4-5%
- ***GovernorTimelockControl*** : Se connecte avec une instance d'un ***TimelockController*** => cette instance doit être fournie dans le constructeur

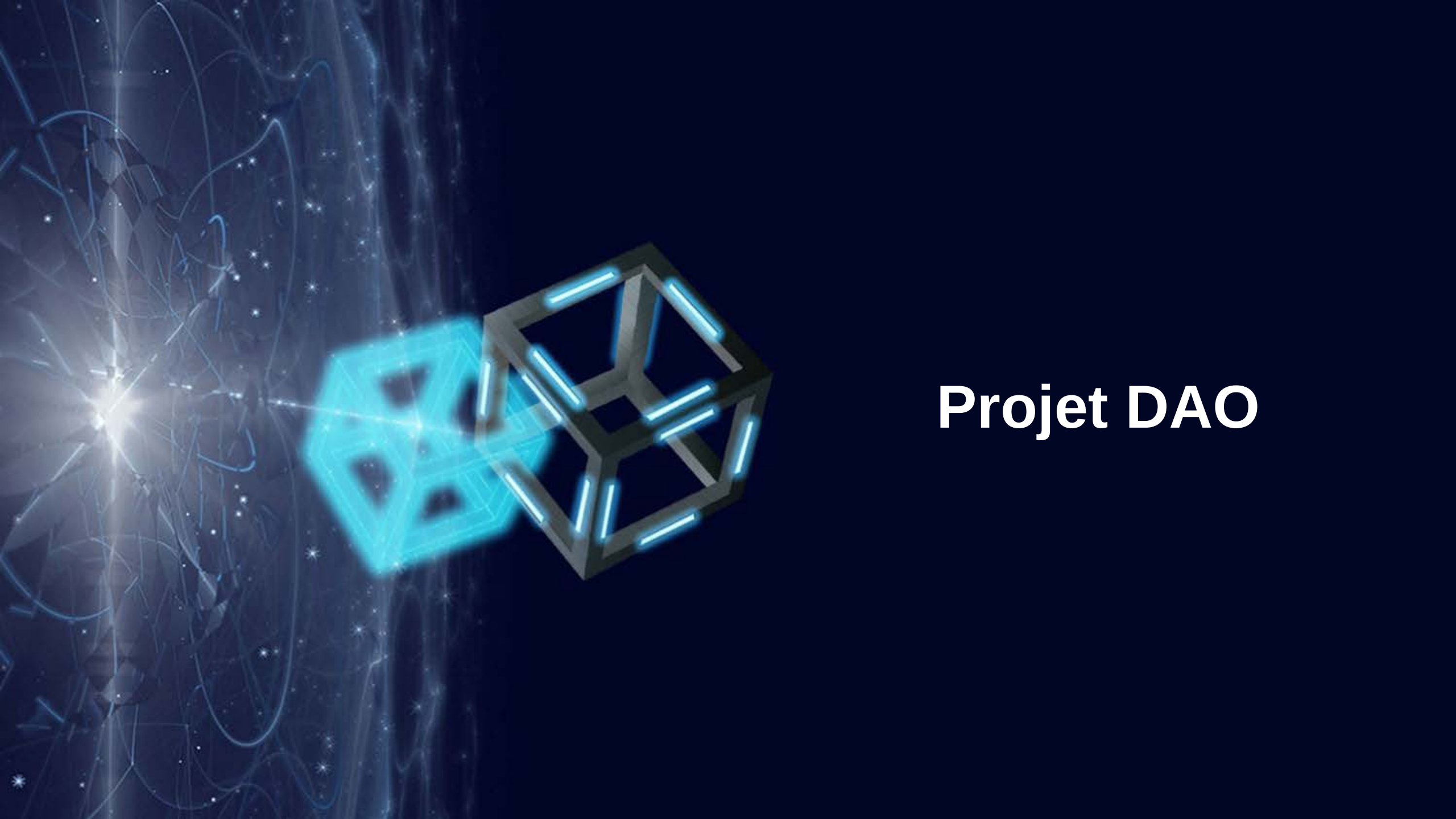
# Contrat Gouvernance – les paramètres requis

***Il faut définir les paramètres suivants :***

- ***votingDelay*** : Délai (en nombre de blocs) entre la création d'une proposition et le début du vote
- ***votingPeriod*** : Délai (en nombre de blocs) entre le début du vote et la fin du vote
- ***quorum*** : Quorum requis pour qu'une proposition soit acceptée

# Contrat Governance – l'interface

- ***propose***(address[] targets, uint256[] values, bytes[] calldatas, string description)
  → uint256 proposalId // créer une nouvelle proposition
- ***state***(uint256 proposalId) → enum IGovernor.ProposalState // retourner l'état de la proposition
- ***castVote***(uint256 proposalId, uint8 support) → uint256 balance // voter
- ***proposalVotes***(uint256 proposalId) → uint256 againstVotes, uint256 forVotes, uint256 abstainVotes // obtenir la répartition des voix
- ***queue***(address[] targets, uint256[] values, bytes[] calldatas, bytes32 descriptionHash)
  → uint256 // mettre en attente une proposition
- ***execute***(address[] targets, uint256[] values, bytes[] calldatas, bytes32 descriptionHash)
  → uint256 proposalId // exécuter une proposition réussie



# Projet DAO

# Projet DAO – les besoins

**Créer les contrats requis :**

- ***GovToken* : *ERC20Votes*** - fournir le nom, le symbole et l'approvisionnement => créer (mint) les jetons à l'adresse du propriétaire du contrat
- ***Treasury* :** contrat de type ***Ownable***, détient les fonds, transfert des fonds aux bénéficiaires
- ***Timelock* : *TimelockController*** – fournir délai minimum, proposant(s) et exécuteur(s)
- ***Governance* : *Governor, GovernorCountingSimple, GovernorVotes, GovernorVotesQuorumFraction, GovernorTimelockControl***
  => fournir délai de vote et période de vote

# Projet DAO – les besoins

**Créer le script de déploiement :**

- Signataires (Signer) pour l'exécuteur, le proposant, le bénéficiaire et 5 électeurs
- Déployer le ***jeton de gouvernance*** : approvisionnement initial : 1000
- Transférer 50 jetons de gouvernance à chaque électeur
- Déployer le contrat ***Timelock*** : ***minDelay = 0***
- Déployer le contrat de ***Gouvernance*** : ***quorum = 5, votingDelay = 0, votingPeriod = 5***
- Timelock : accorde des rôles de proposant et de l'exécuteur au contrat de gouvernance
- Déployer le contrat de ***Trésorerie*** : transférer les fonds (50 ETH), transférer la propriété au contrat **Timelock**

# Projet DAO – les besoins

**Créer le script de la proposition 1/2:**

- Auto-déléger le droit de vote aux 5 électeurs
- Afficher la trésorerie initiale et le solde du bénéficiaire
- Encoder la fonction ***releaseFunds*** du contrat de trésorerie => requis pour la proposition
- Créer une proposition : Il y a une action => cible : adresse de trésorerie, valeur : 0, calldata : fonction "***releaseFunds***" encodée
- Afficher l'ID de la proposition de l'événement ***ProposalCreated***
- Afficher l'état de la proposition

# Projet DAO – les besoins

**Créer le script de la proposition 2/2:**

- Afficher les blocs de la création & l'échéance de la proposition
- Voter
- Afficher le nombre de votes (pour, contre et abstention)
- Mettre la proposition en attente
- Exécuter la proposition
- Afficher l’état de la proposition
- Afficher la trésorerie finale et le solde du bénéficiaire