SMART CONTRACTS : UN DROIT DU CODE EN CODANT LE DROIT

— 1 —

Intro : CONTRACTUALISATION DES ROYALTIES

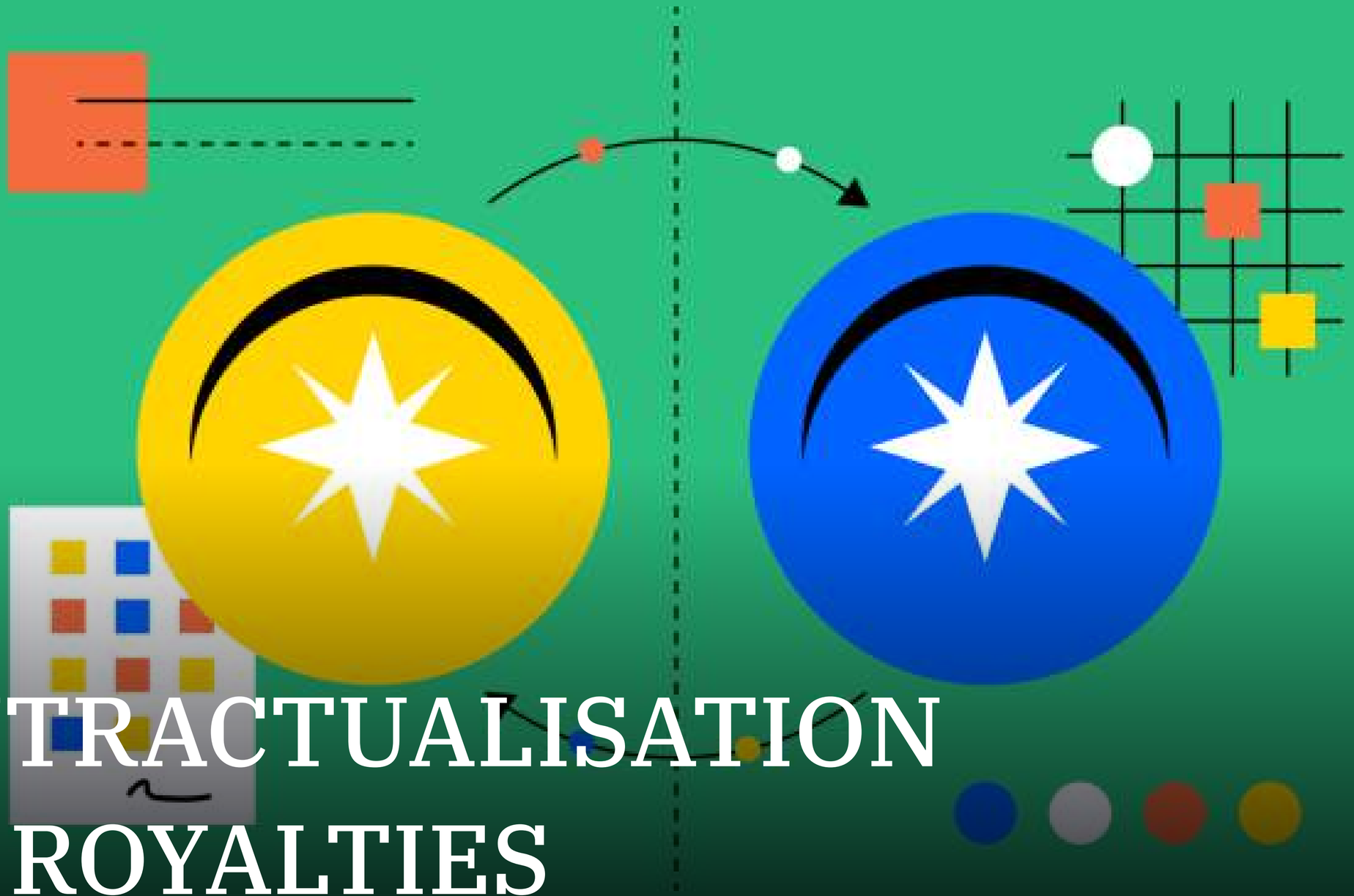
Démo : NFT CRYPTOPUNKS

— 2 —

Intro : DU PHYSIQUE AU VIRTUEL

Démo : STABLECOIN & USDC

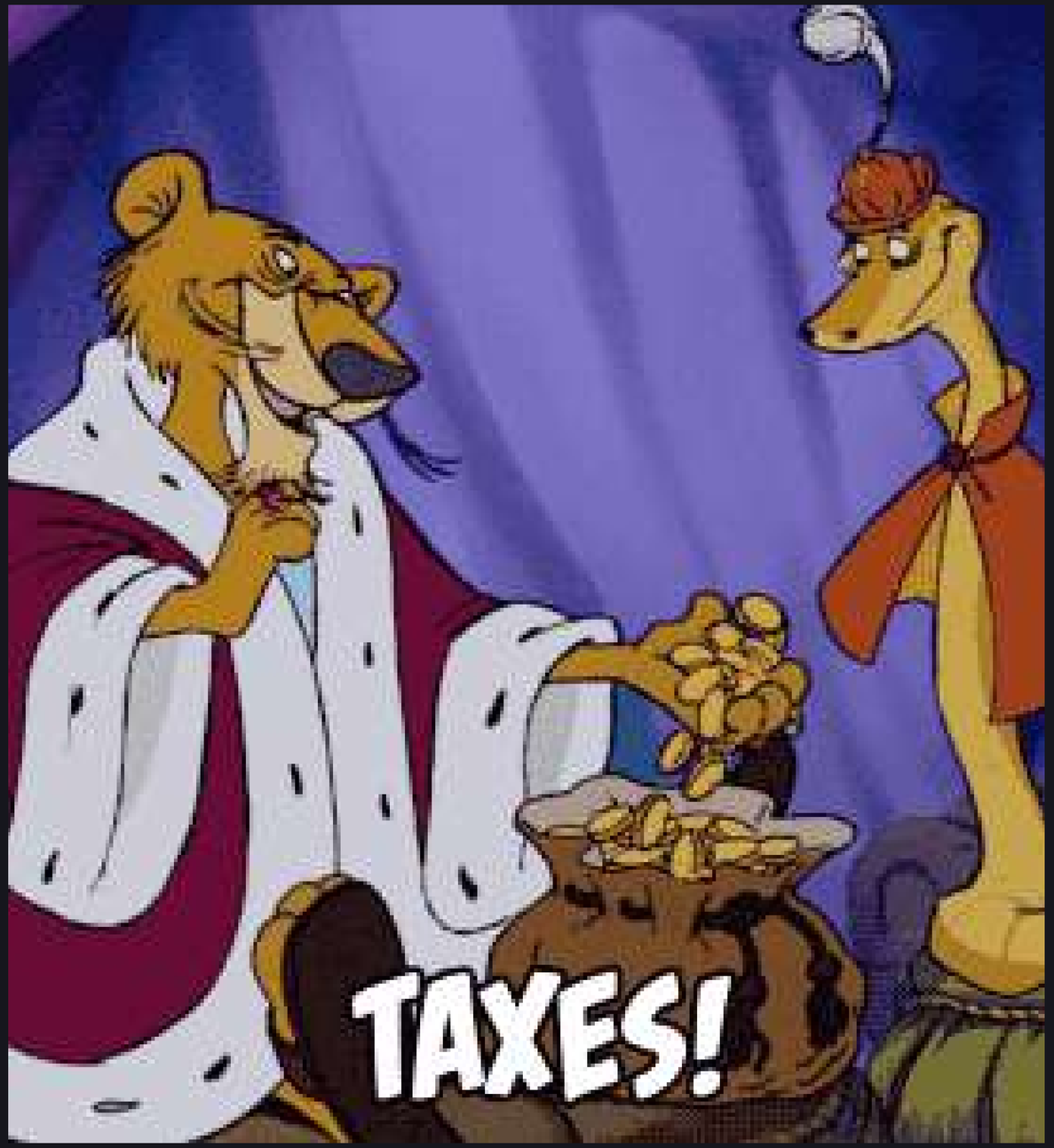
01



CONTRACTUALISATION DES ROYALTIES

LES ROYALTIES C’EST ROYAL !

Les *smart contracts* ouvrent la voie aux droits d’auteurs paramétrables. Mais de combien de contrat parle-t-on ?

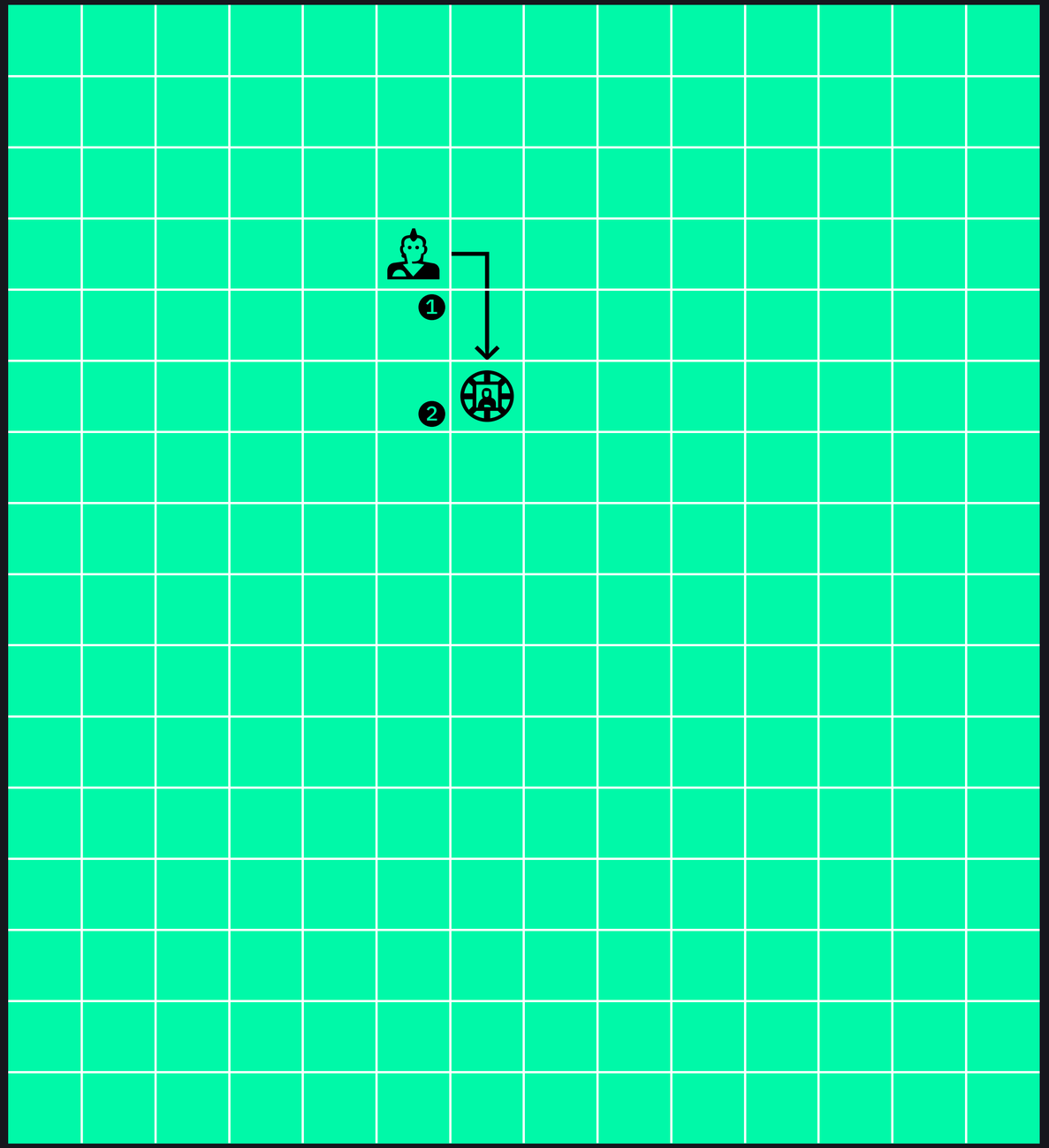


DROIT DE SUITE

Principe de base des royalties appliqué au NFT.

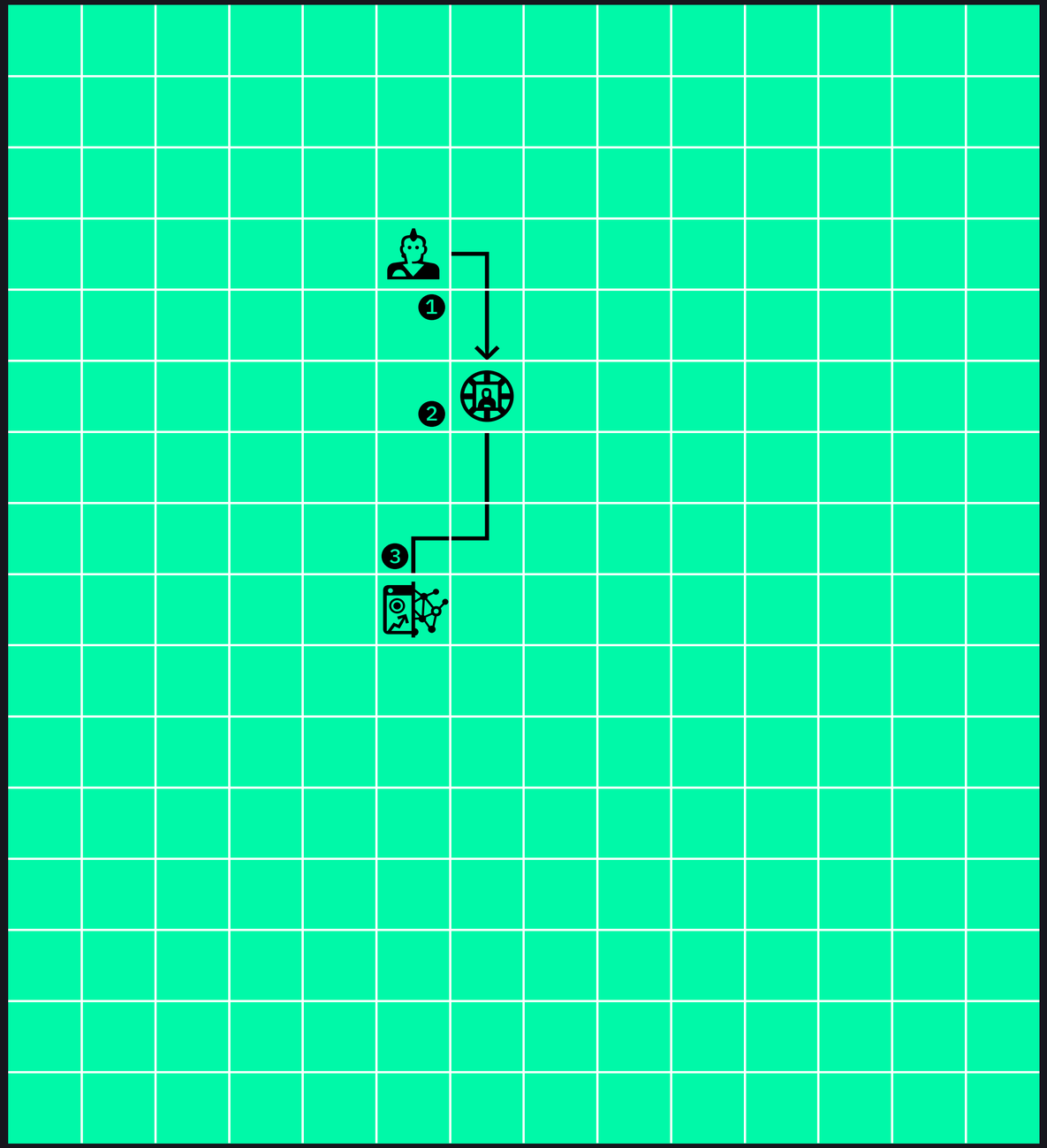
CONTRAT (SOCIAL) DES SMART CONTRACTS

La création d'une collection de NFT [2] nécessite la création d'un *smart contract*.



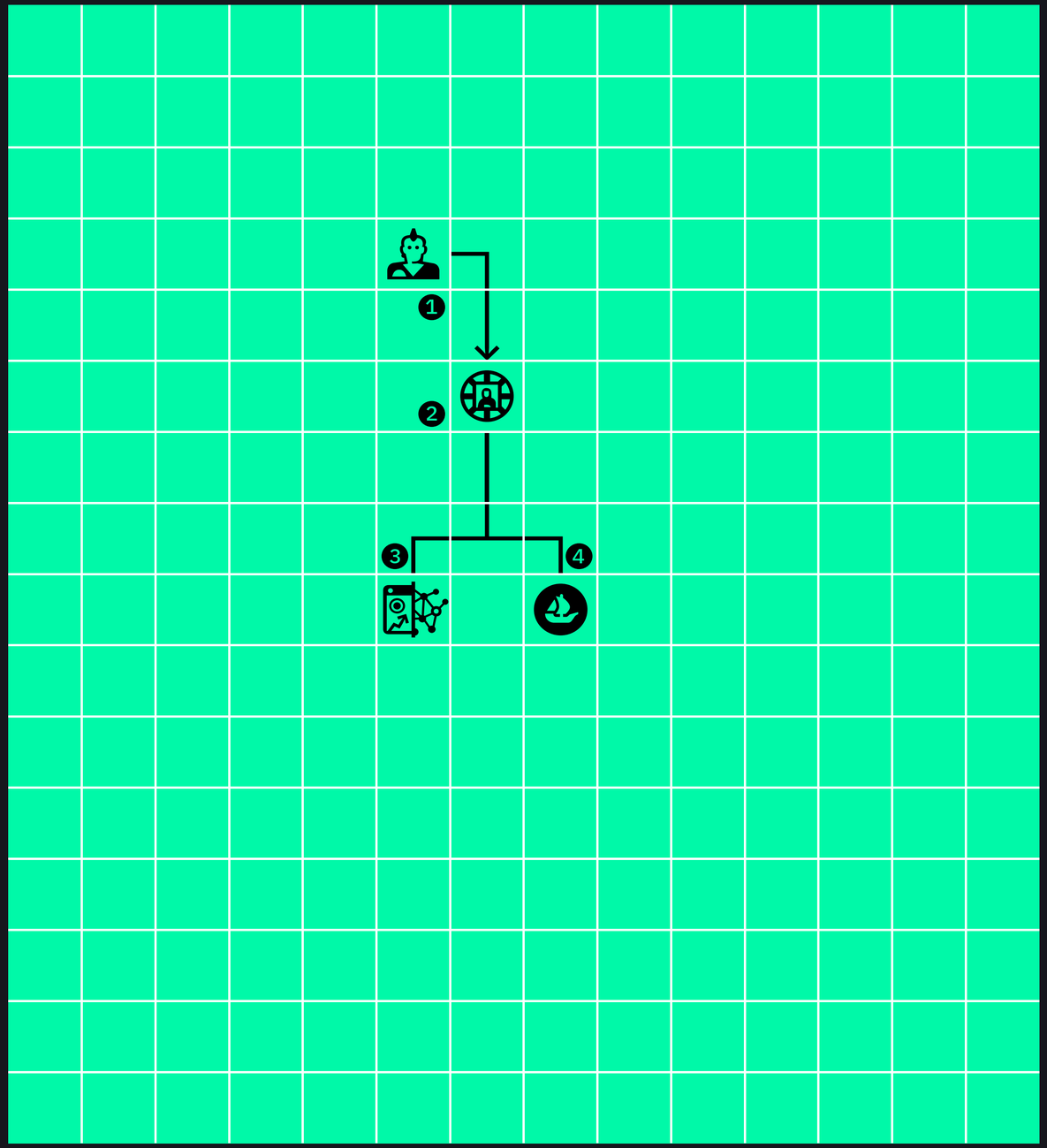
CONTRAT (SOCIAL) DES SMART CONTRACTS

Cette collection peut être mise en vente via une plateforme spécifiquement dédiée (Larva Lab) [3].



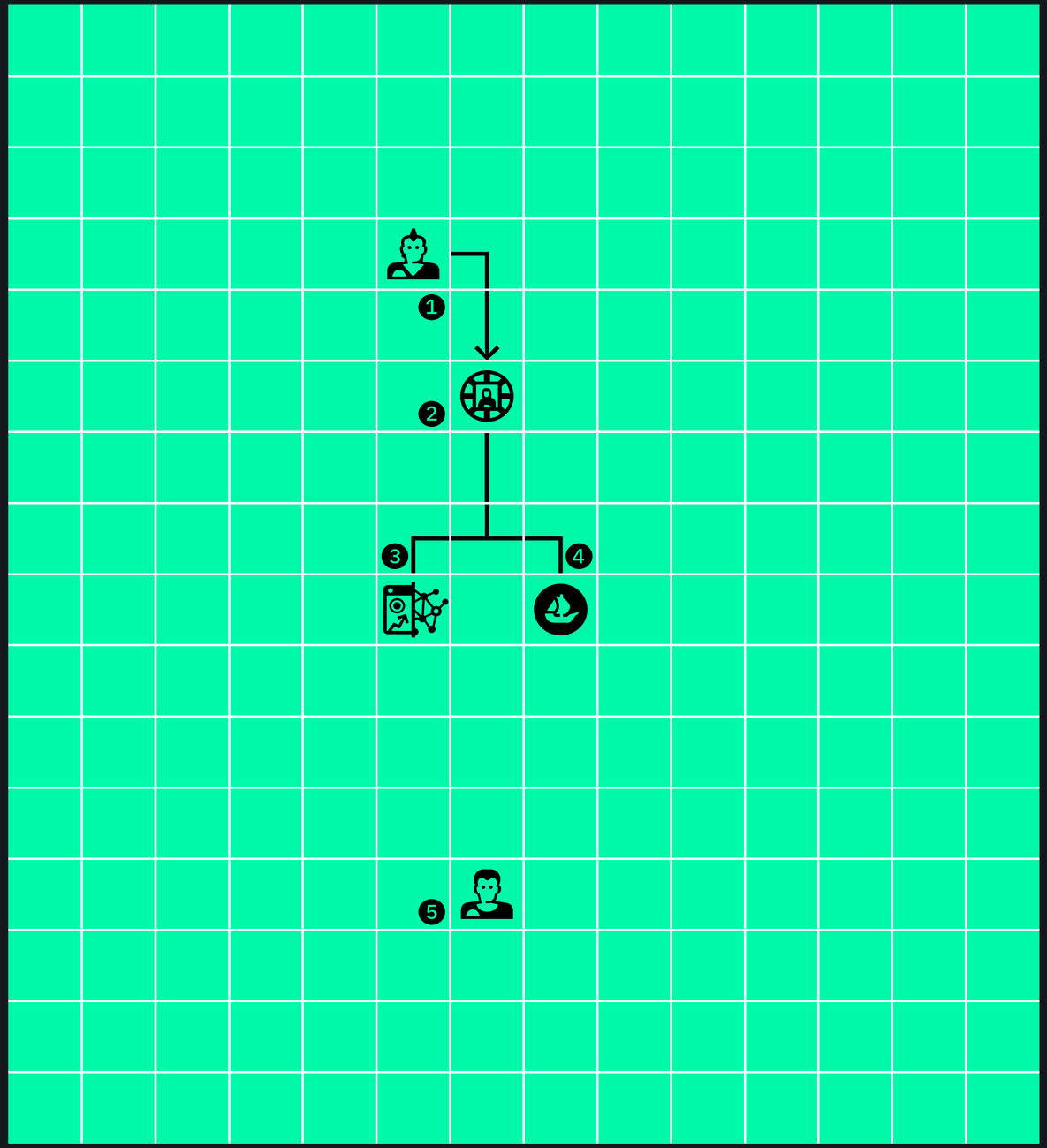
CONTRAT (SOCIAL) DES SMART CONTRACTS

Les NFT peuvent également être listés sur des *marketplaces* grand public comme OpenSea [4].



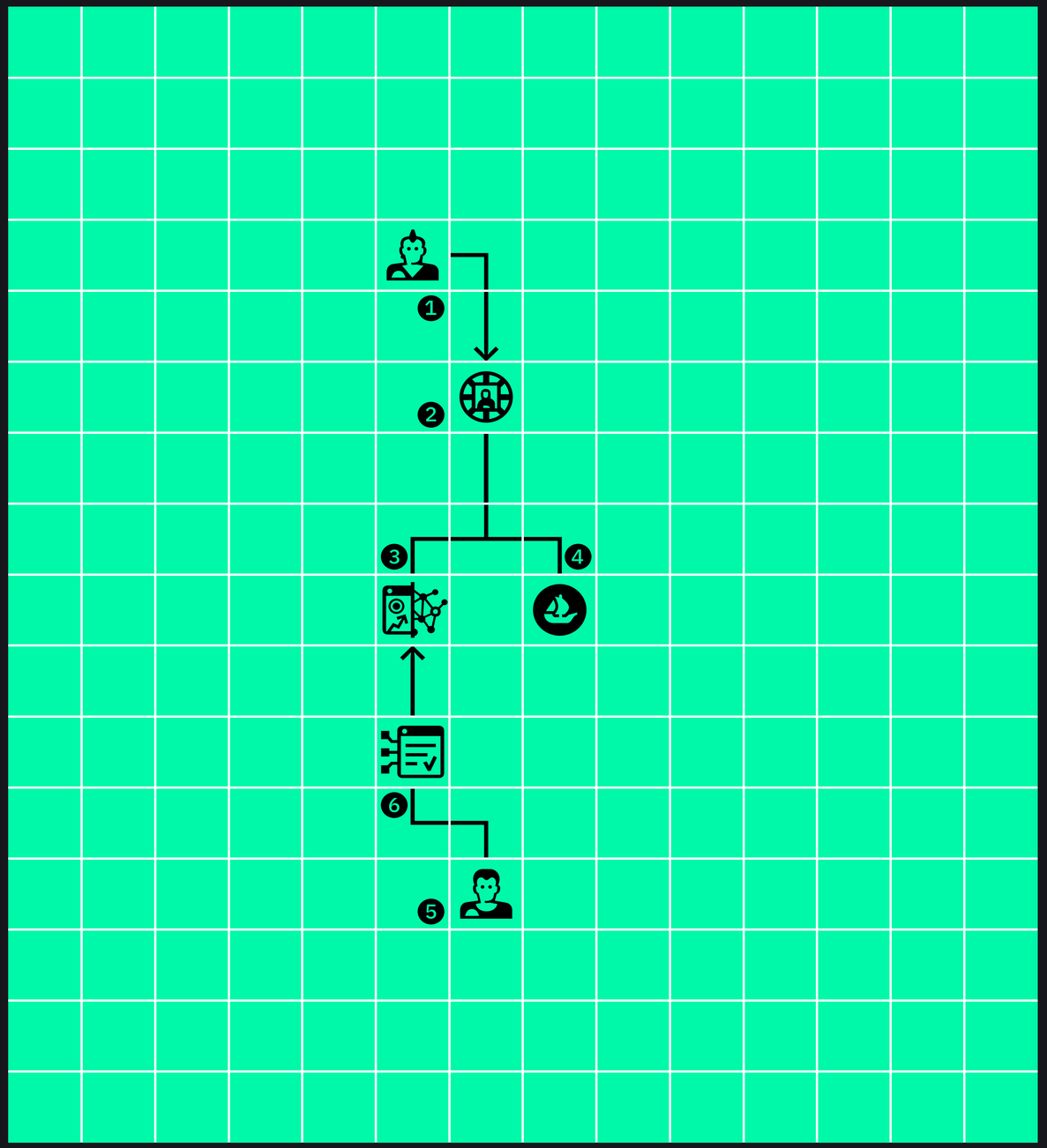
CONTRAT (SOCIAL) DES SMART CONTRACTS

Un double listing laisse donc le choix à un acheteur potentiel [5] de la marketplace à utiliser.



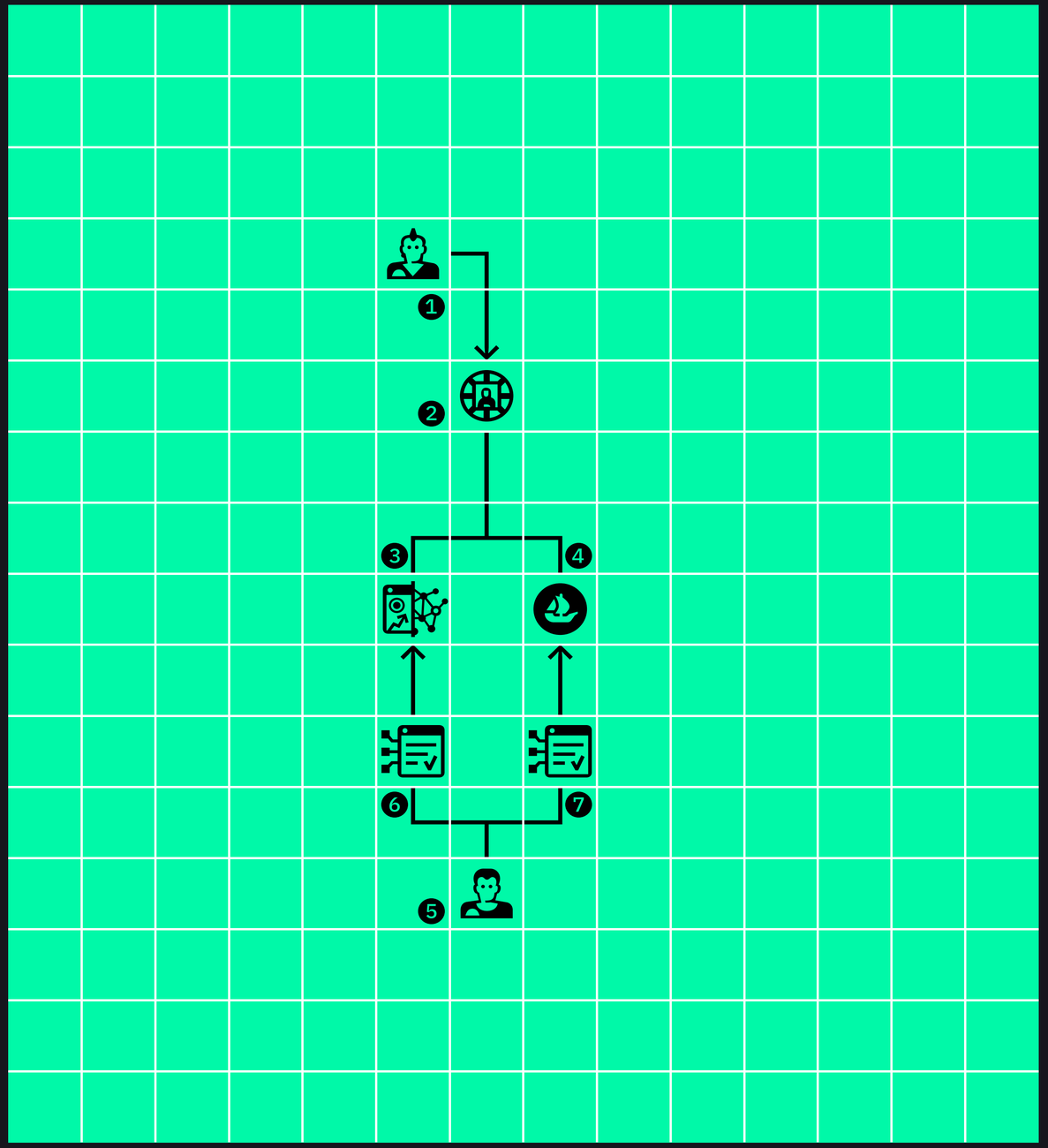
CONTRAT (SOCIAL) DES SMART CONTRACTS

Chacune de ces plateformes utilise son propre *smart contract* [6] de mise en vente.



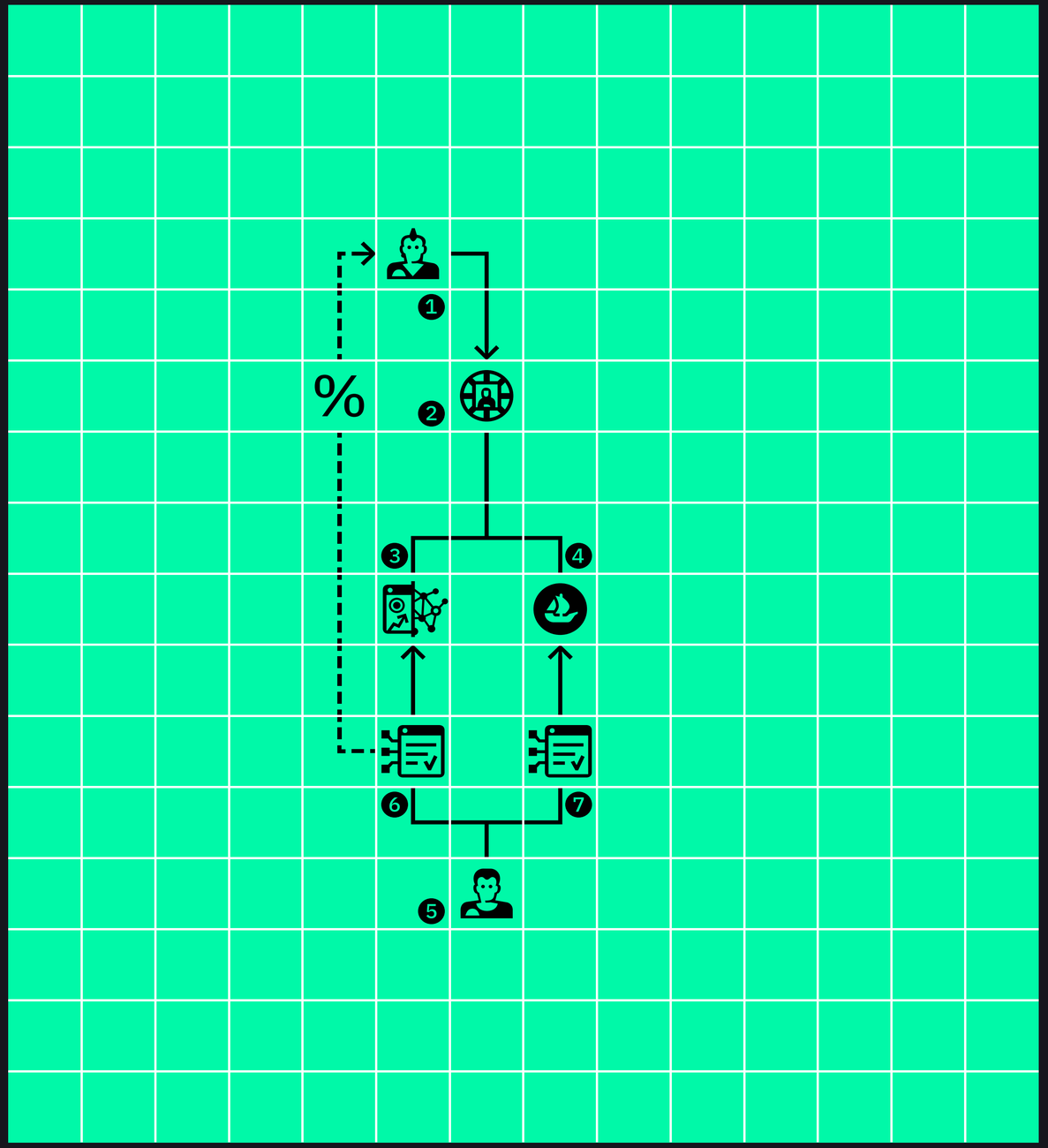
CONTRAT (SOCIAL) DES SMART CONTRACTS

Ces deux *smart-contracts* [7] peuvent donc définir des règles différentes.



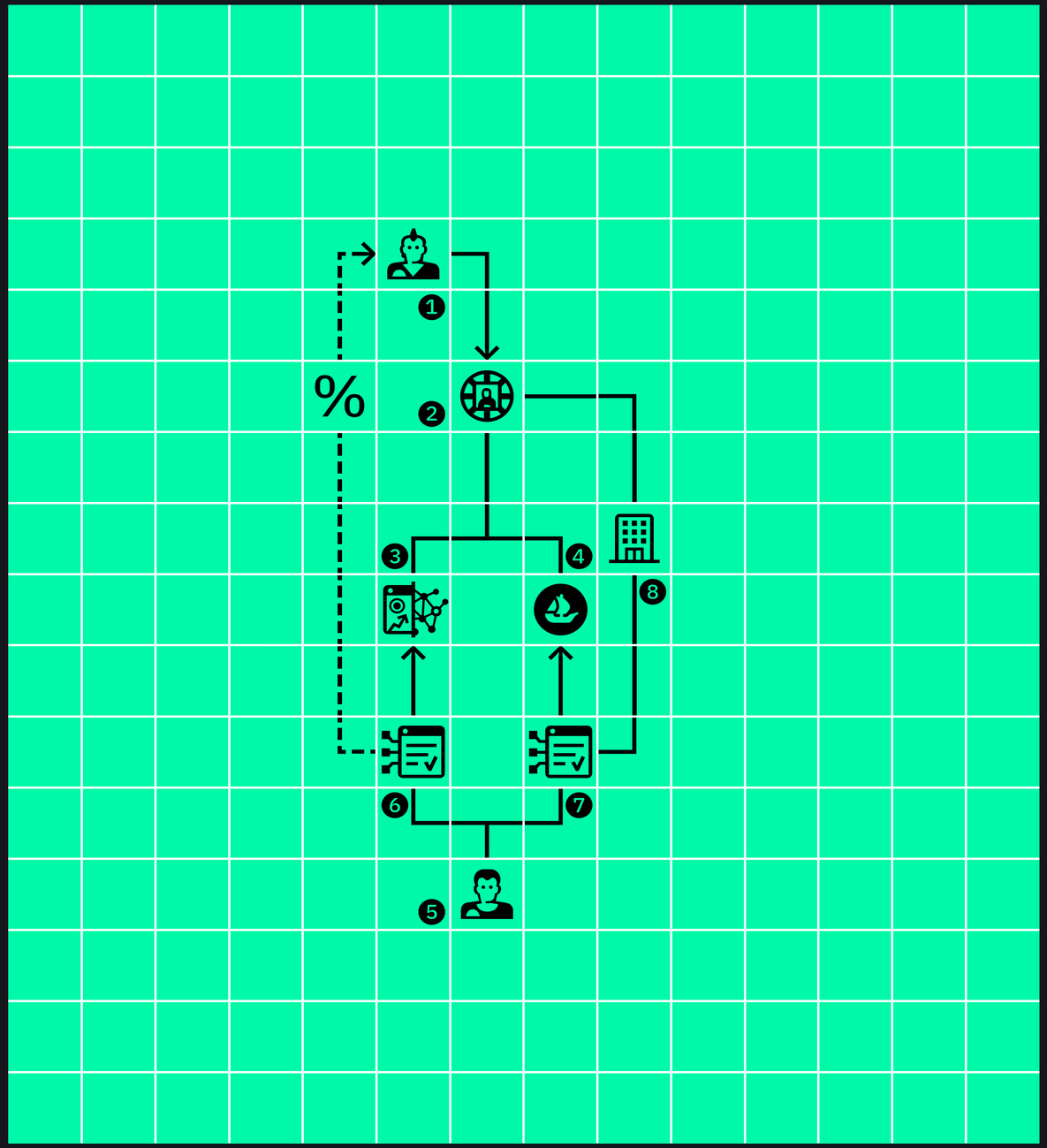
CONTRAT (SOCIAL) DES SMART CONTRACTS

Si elle est administrée par l'artiste lui-même, la *marketplace* permet une complète flexibilité des règles de vente et donc des royalties à attribuer.



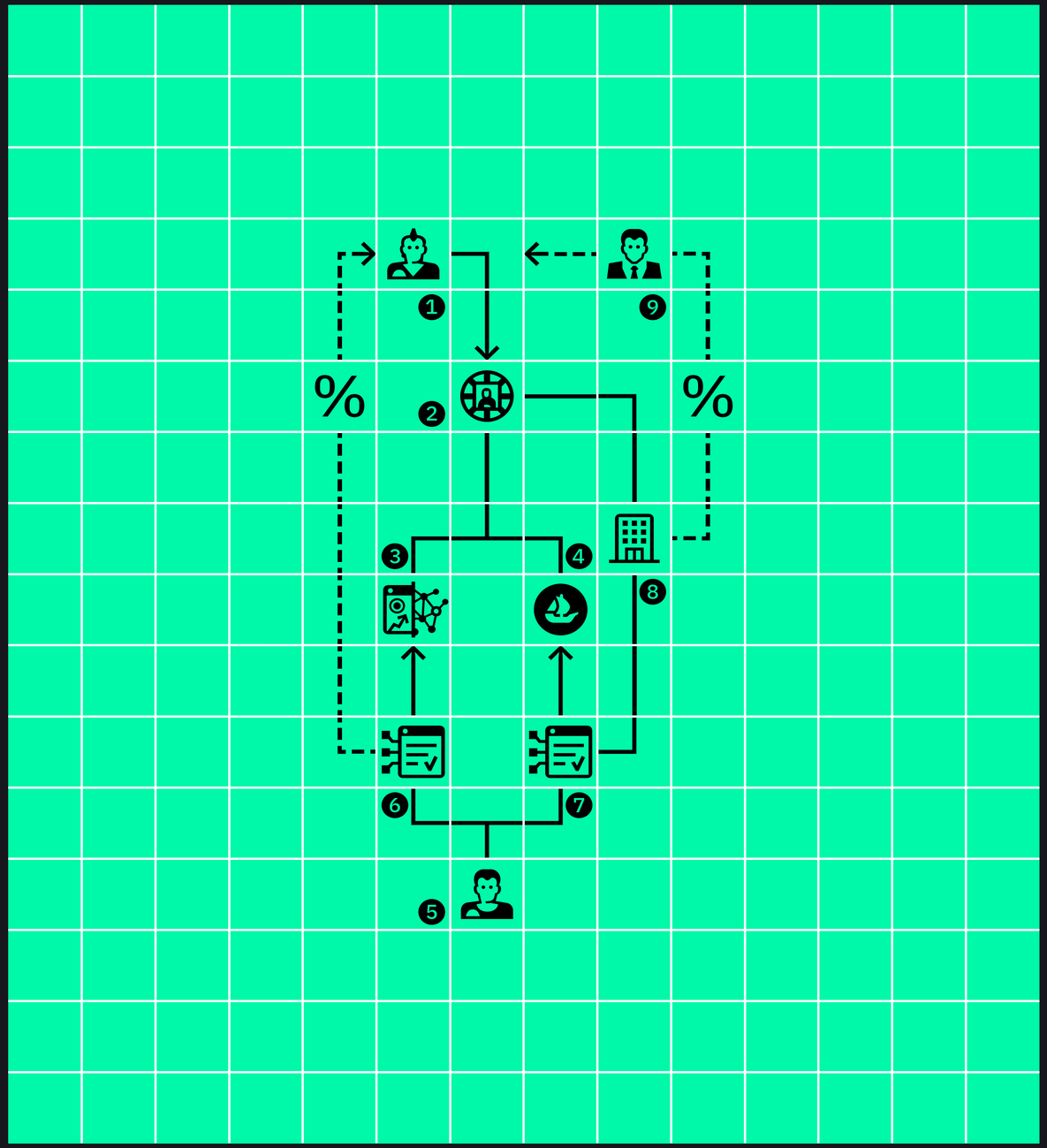
CONTRAT (SOCIAL) DES SMART CONTRACTS

La deuxième plateforme (OpenSea), étant propriétaire [8], elle ne laisse aucun pouvoir aux artistes en dehors de ses conditions.



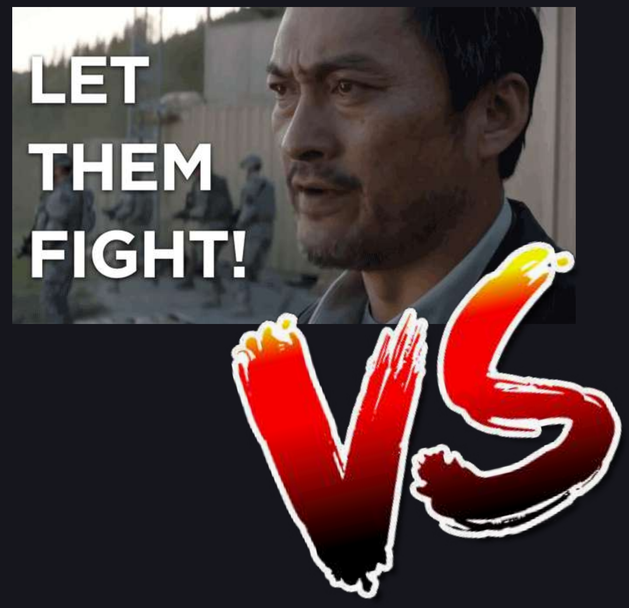
CONTRAT (SOCIAL) DES SMART CONTRACTS

Par exemple, OpenSea a limité le taux de royalties à 10% de manière arbitraire [9].

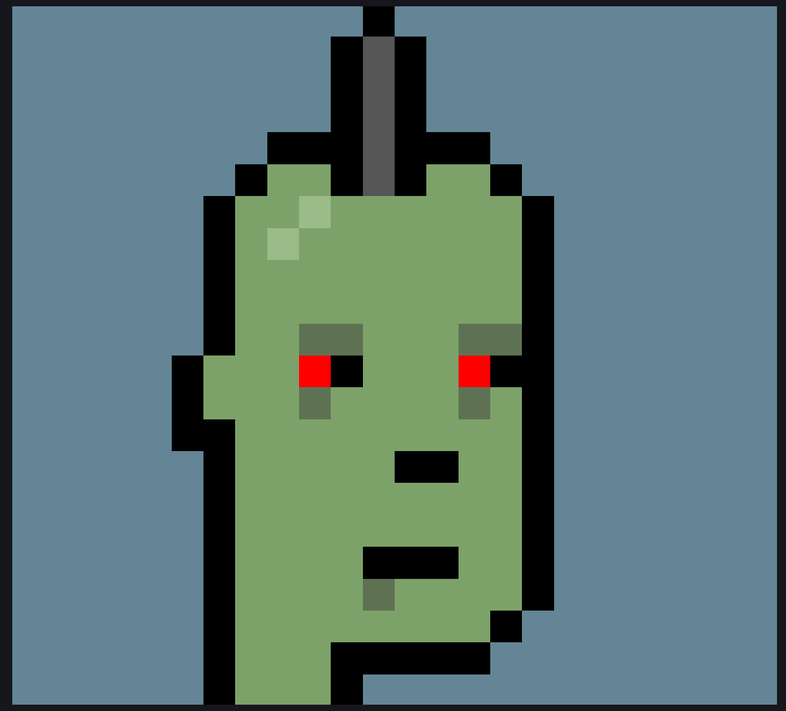


DÉMONSTRATION

WE ARE THE OUTKAST



CRYPTOPUNK



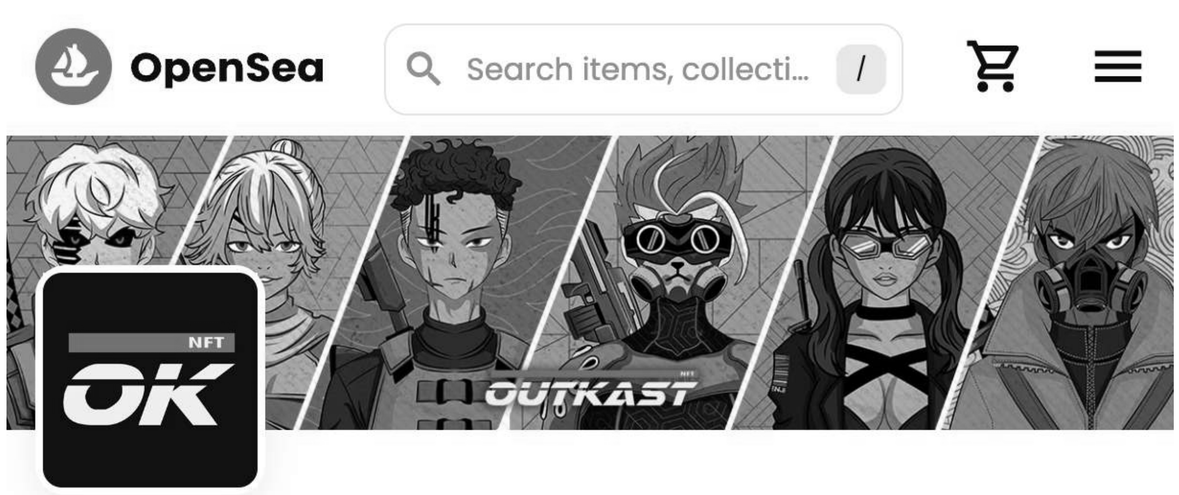
```
/*
// Presale is handled internally using the hash system, so no need for 2 seperate functions
*/
function mintOutkast(uint256 tokenQuantity, bytes calldata signature, uint256 nonce) external payable {
    require(isSaleActive, "Sale Inactive");
    require(totalSupply + tokenQuantity <= MAX_OUTKASTS, "Sold Out");
    require(tokenQuantity <= MAX_OUTKASTS_PER_TXNS, "Mint Overflow");
    require(OUTKAST_PRICE * tokenQuantity <= msg.value, "Insufficient Funds");

    bytes32 messageHash = hashMessage(msg.sender, tokenQuantity, nonce);
    require(messageHash.recover(signature) == _signatureVerifier, "Unrecognizable Hash");
    require(!usedHashes[messageHash], "Reused Hash");

    usedHashes[messageHash] = true;

    for (uint256 i = 0; i < tokenQuantity; i++) {
        _safeMint(msg.sender, ++totalSupply);
    }

    if(totalSupply == MAX_OUTKASTS) {
        saleHasConcluded = true;
    }
}
```

We are the Outkast
Minting

```solidity
/**
 * @dev See {IERC721-transferFrom}.
 */
function transferFrom(
    address from,
    address to,
    uint256 tokenId
) public virtual override {
    //solhint-disable-next-line max-line-length
    require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not owner nor approved");

    _transfer(from, to, tokenId);
}
```

We are the Outkast

Transfer


OpenSea
Search items, collecti...
/
NFT
OK
OUTKAST
WE ARE THE OUTKAST
By MNNT
Items 9772 · Created Sep 2021 · Creator earnings 5% · Chain Ethereum · Category Virtual Worlds
We Are the OutKast, the genesis collection of the Andrometa...
See more
947 ETH
total volume
0,0085 ETH
floor price
0,004 WETH
best offer
0,7%
listed
2580
owners
26%
unique owners

```
// Transfer ownership of a punk to another user without requiring payment
function transferPunk(address to, uint punkIndex) {
    if (!allPunksAssigned) throw;
    if (punkIndexToAddress[punkIndex] != msg.sender) throw;
    if (punkIndex >= 10000) throw;
    if (punksOfferedForSale[punkIndex].isForSale) {
        punkNoLongerForSale(punkIndex);
    }
    punkIndexToAddress[punkIndex] = to;
    balanceOf[msg.sender]--;
    balanceOf[to]++;
    Transfer(msg.sender, to, 1);
    PunkTransfer(msg.sender, to, punkIndex);
    // Check for the case where there is a bid from the new owner and refund it.
    // Any other bid can stay in place.
    Bid bid = punkBids[punkIndex];
    if (bid.bidder == to) {
        // Kill bid and refund value
        pendingWithdrawals[to] += bid.value;
        punkBids[punkIndex] = Bid(false, punkIndex, 0x0, 0);
    }
}
```

Cryptopunk
Transfer

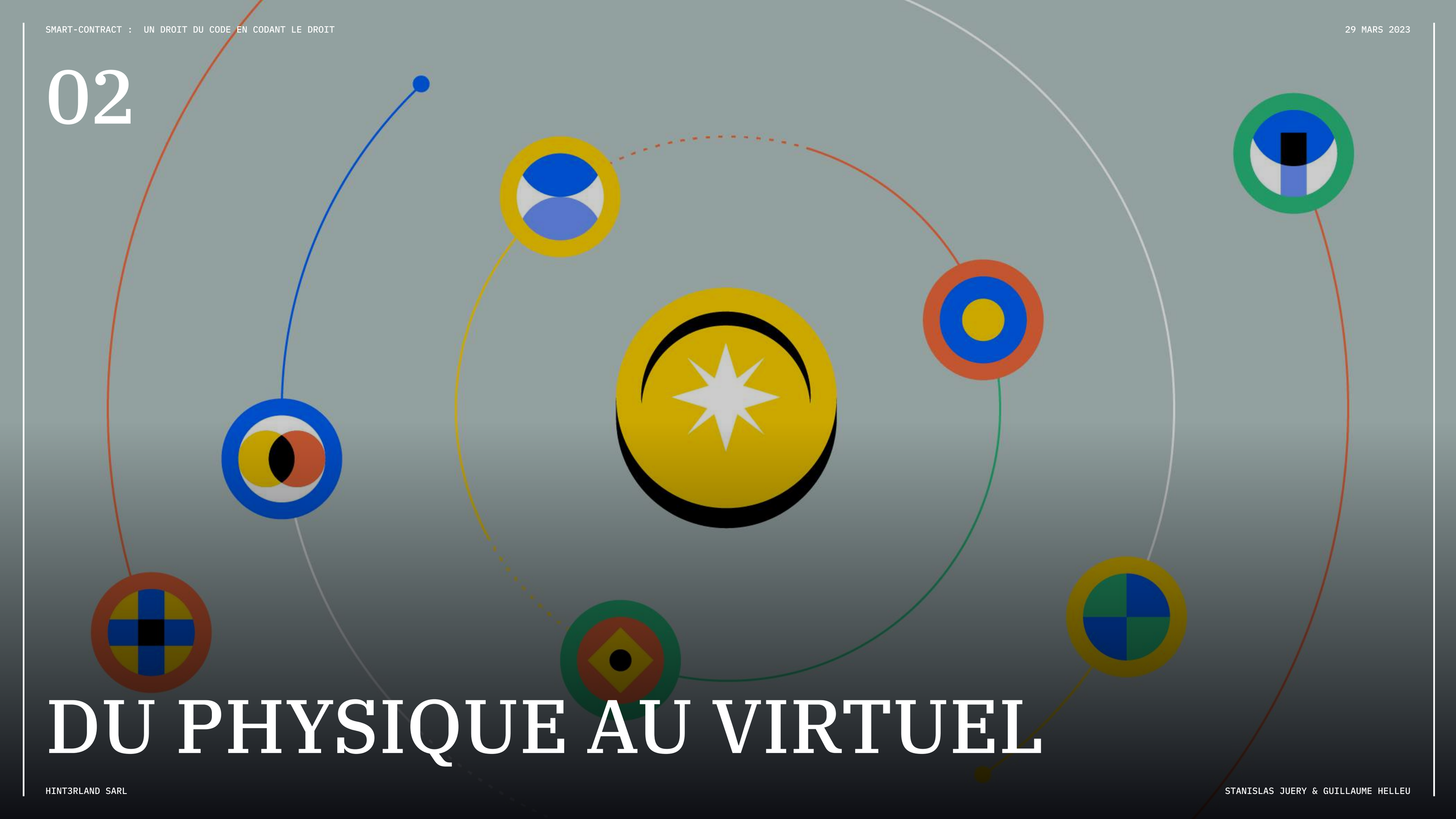
```
function offerPunkForSale(uint punkIndex, uint minSalePriceInWei) {
    if (!allPunksAssigned) throw;
    if (punkIndexToAddress[punkIndex] != msg.sender) throw;
    if (punkIndex >= 10000) throw;
    punksOfferedForSale[punkIndex] = Offer(true, punkIndex, msg.sender, minSalePriceInWei, 0x0);
    PunkOffered(punkIndex, minSalePriceInWei, 0x0);
}
```

Cryptopunk
Buy

```
function buyPunk(uint punkIndex) payable {
    if (!allPunksAssigned) throw;
    Offer offer = punksOfferedForSale[punkIndex];
    if (punkIndex >= 10000) throw;
    if (!offer.isForSale) throw;                // punk not actually for sale
    if (offer.onlySellTo != 0x0 && offer.onlySellTo != msg.sender) throw;  // punk not supposed to be sold to this user
    if (msg.value < offer.minValue) throw;      // Didn't send enough ETH
    if (offer.seller != punkIndexToAddress[punkIndex]) throw; // Seller no longer owner of punk

    address seller = offer.seller;

    punkIndexToAddress[punkIndex] = msg.sender;
    balanceOf[seller]--;
    balanceOf[msg.sender]++;
    Transfer(seller, msg.sender, 1);

    punkNoLongerForSale(punkIndex);
    pendingWithdrawals[seller] += msg.value;
    PunkBought(punkIndex, msg.value, seller, msg.sender);

    // Check for the case where there is a bid from the new owner and refund it.
    // Any other bid can stay in place.
    Bid bid = punkBids[punkIndex];
    if (bid.bidder == msg.sender) {
        // Kill bid and refund value
        pendingWithdrawals[msg.sender] += bid.value;
        punkBids[punkIndex] = Bid(false, punkIndex, 0x0, 0);
    }
}
```

Cryptopunk
Sale

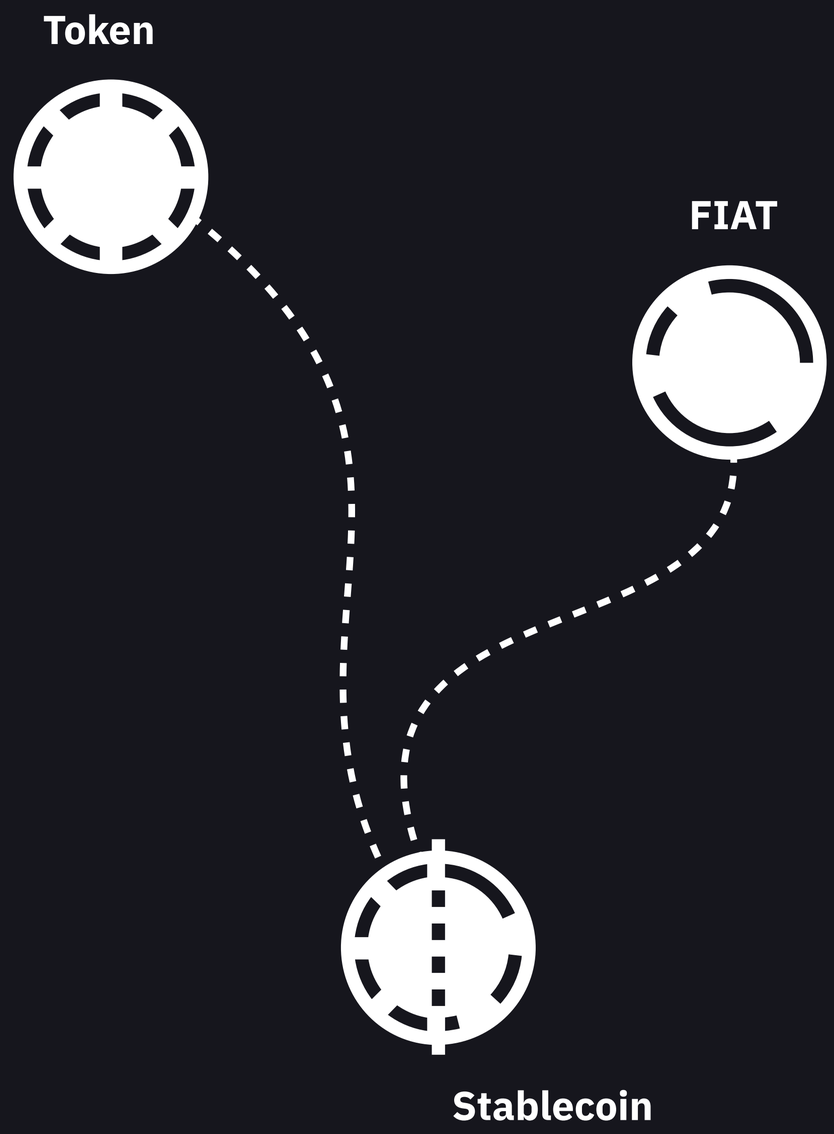
02



DU PHYSIQUE AU VIRTUEL

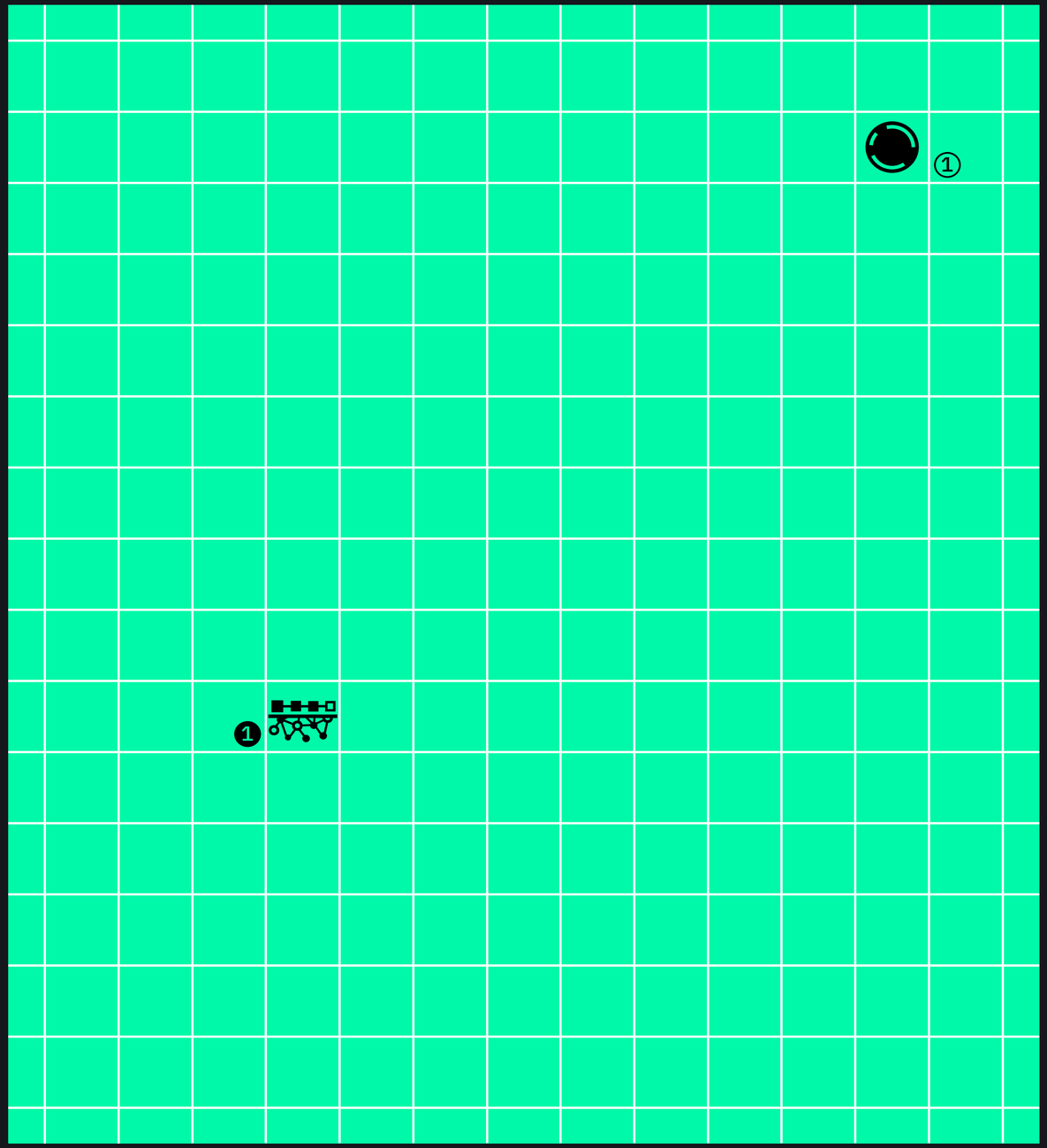
STABLECOINS

Les stablecoins ont pour mission de relier le système monétaire traditionnel aux systèmes décentralisés (DeFi).



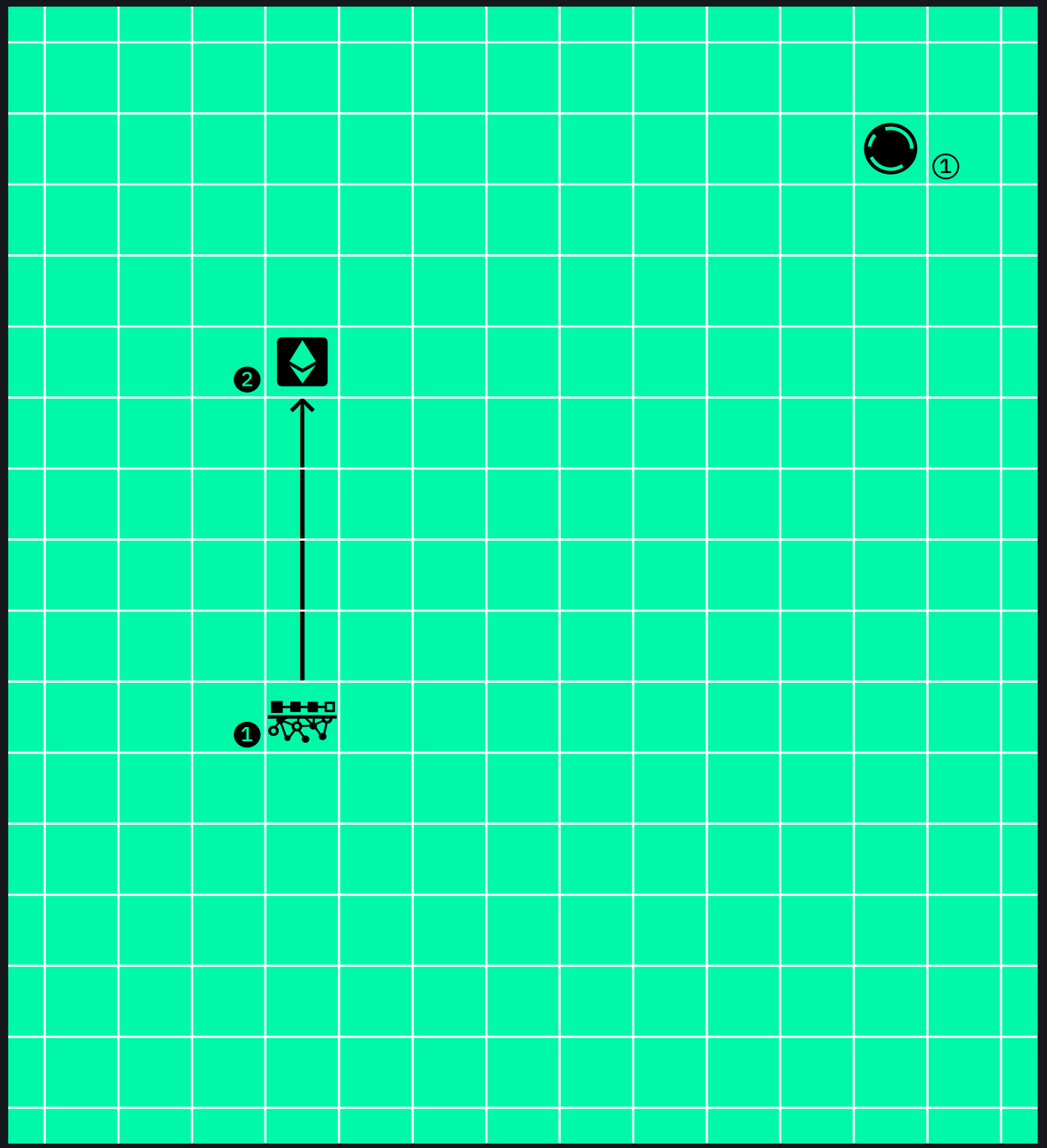
TAXONOMIE DES TOKENS

Relation et interconnexion entre les technologies blockchain [1] et les monnaies FIAT (1).



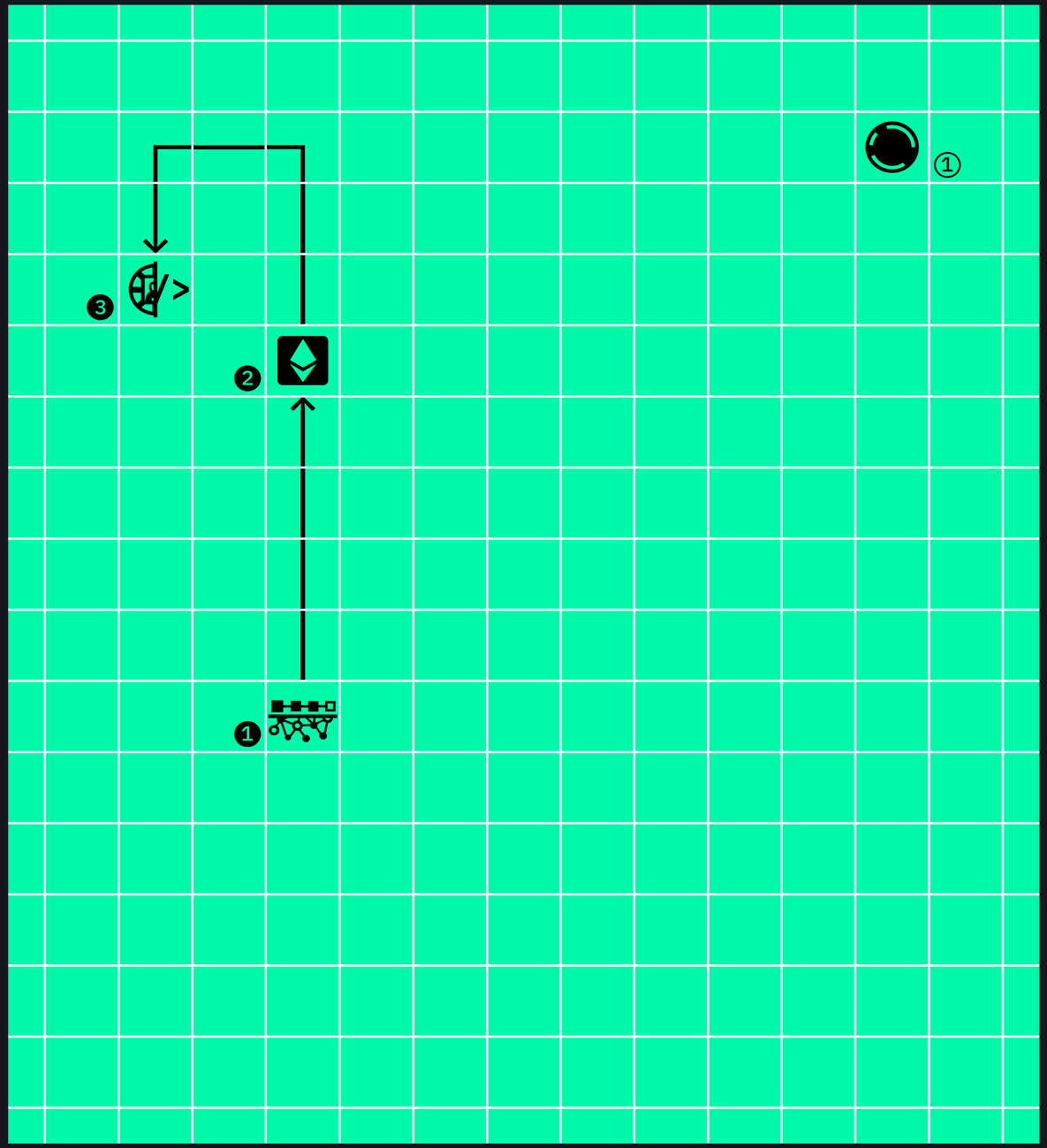
TAXONOMIE DES TOKENS

Par la mise en place de standards (ERC), Ethereum [2] ouvre la voie aux *tokens* paramétrables.



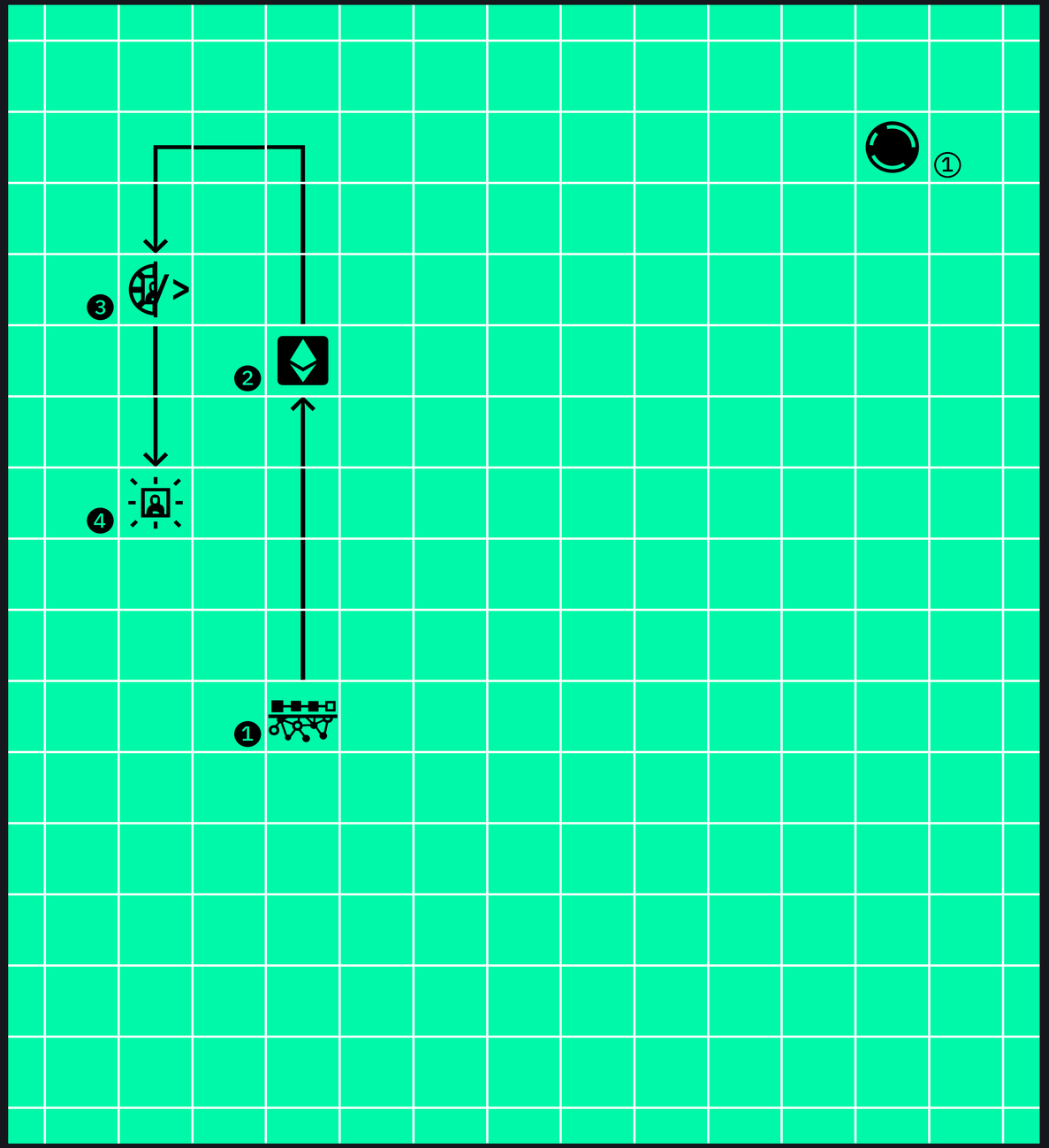
TAXONOMIE DES TOKENS

Proposé en janvier 2018, l’ERC–721 [3] établi la norme des NFT (*non-fungible token*).



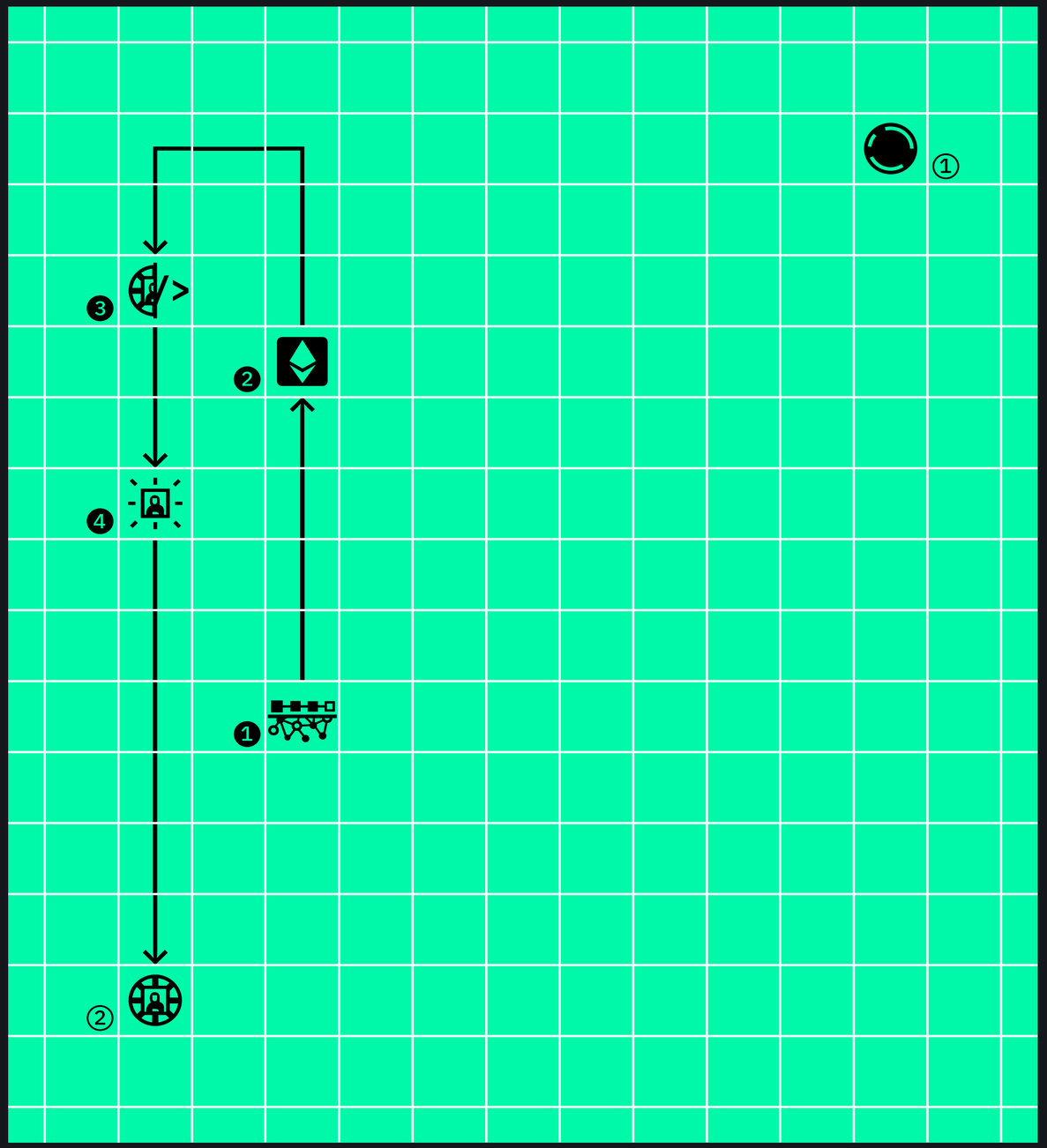
TAXONOMIE DES TOKENS

Il se caractérise par une non fongibilité [4] des *tokens* et permet la création d'actifs numériques uniques.



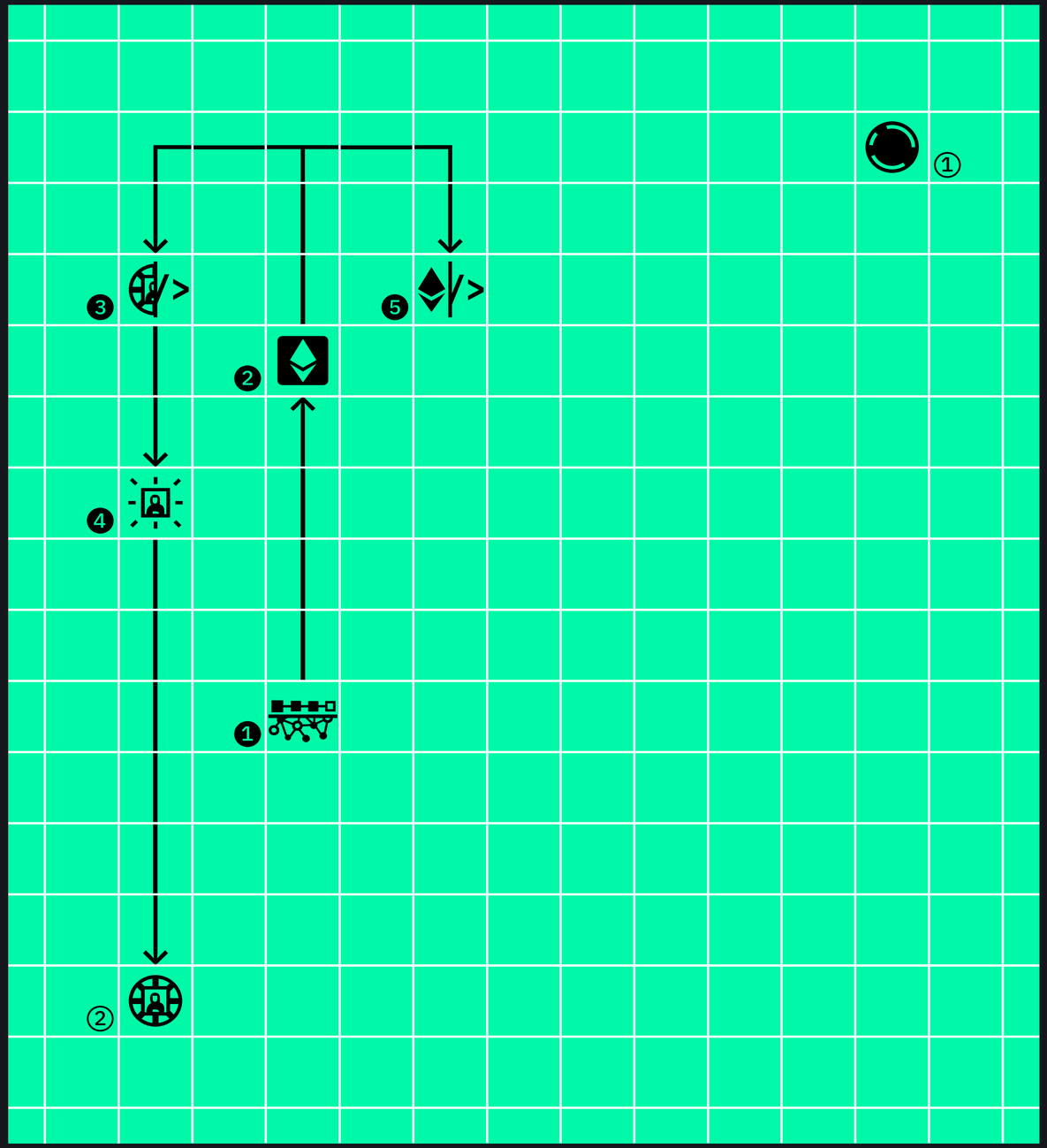
TAXONOMIE DES TOKENS

L'émergence du crypto-art entraîne une évolution du standard des NFT (2) avec l'apparition d'une nouvelle norme (ERC-1155).



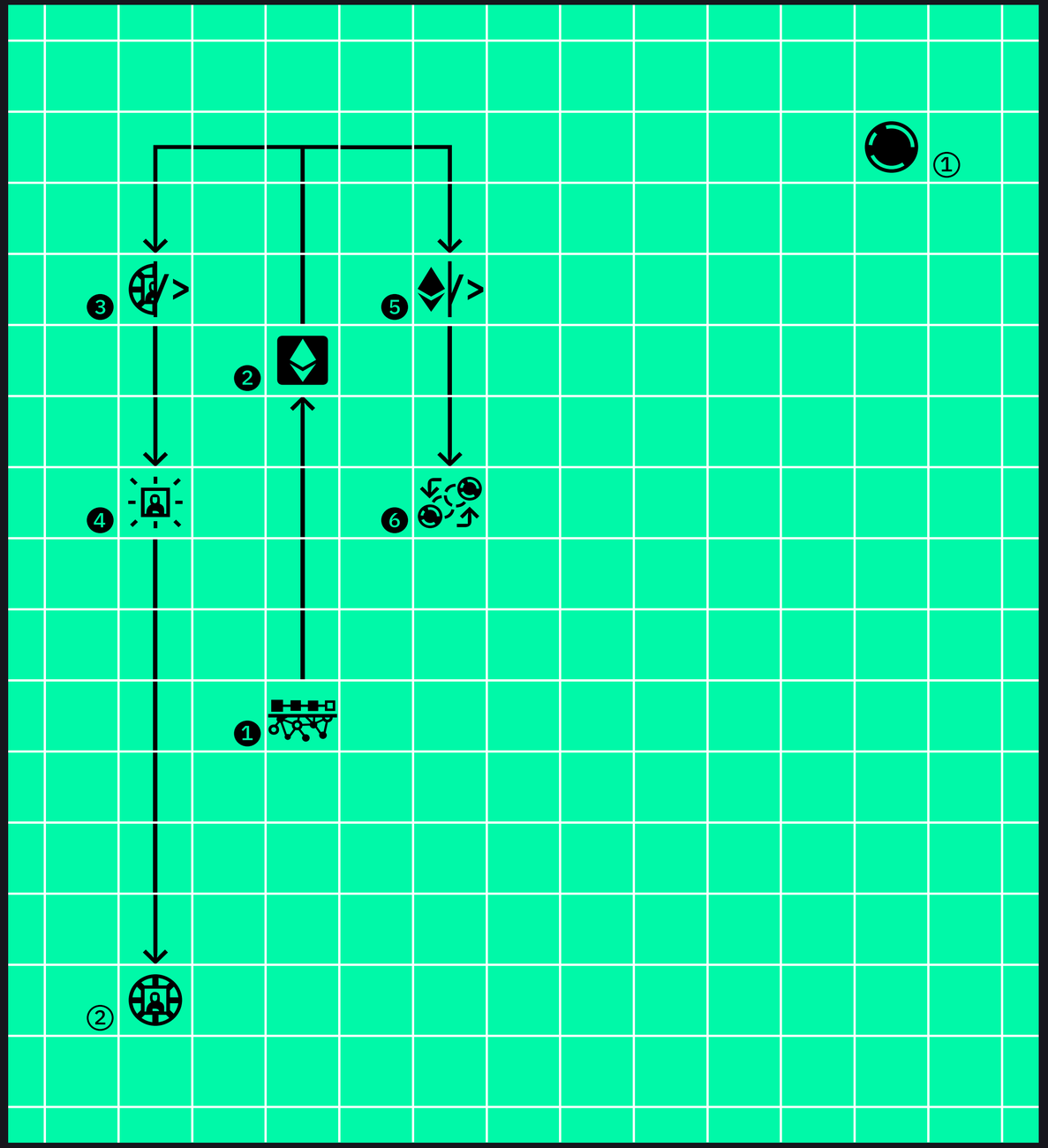
TAXONOMIE DES TOKENS

l’ERC–20 [5] reste cependant le standard de base pour la majorité des *tokens*.



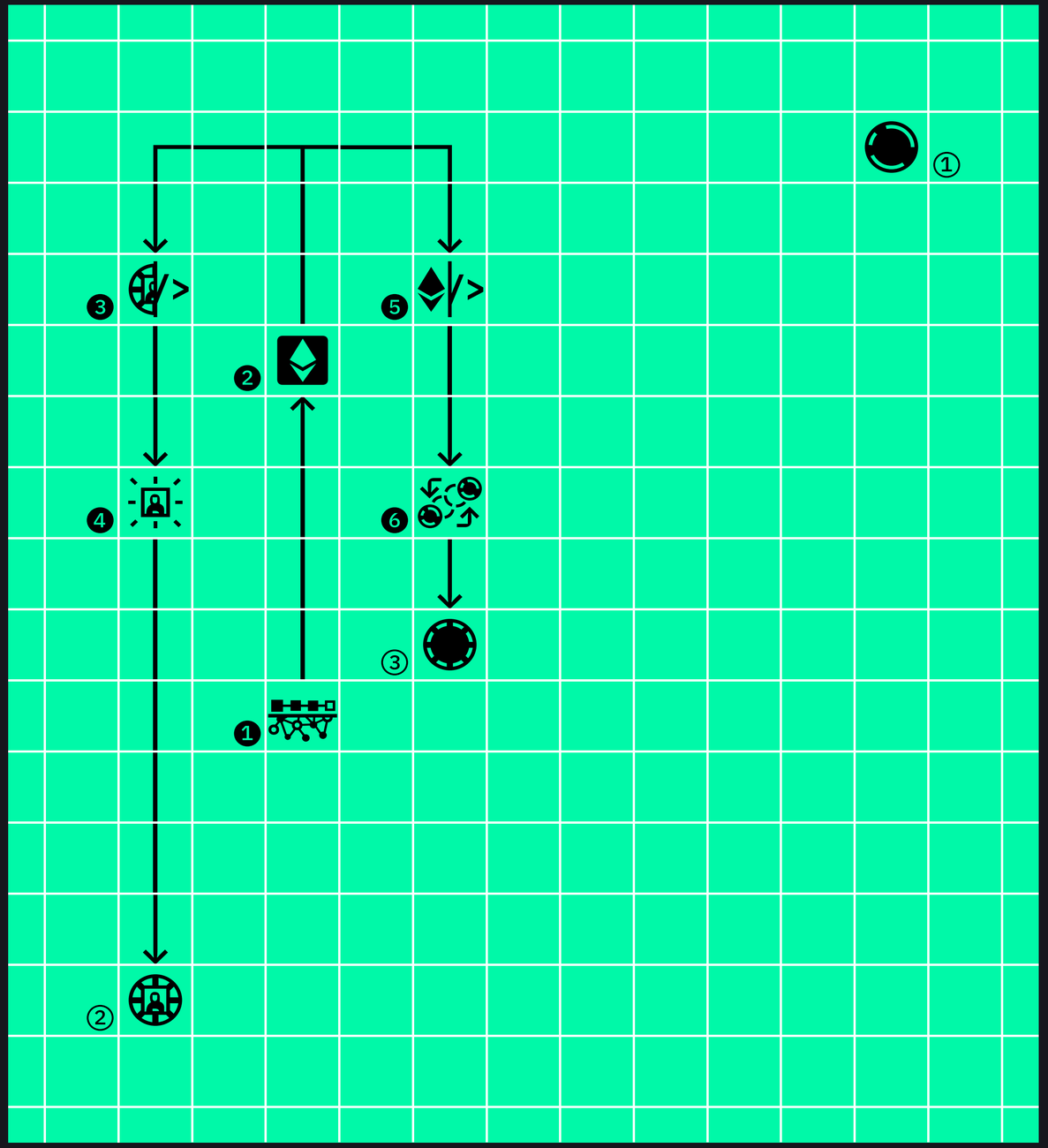
TAXONOMIE DES TOKENS

Il se caractérise par la fongibilité [6] des *tokens* entre eux.



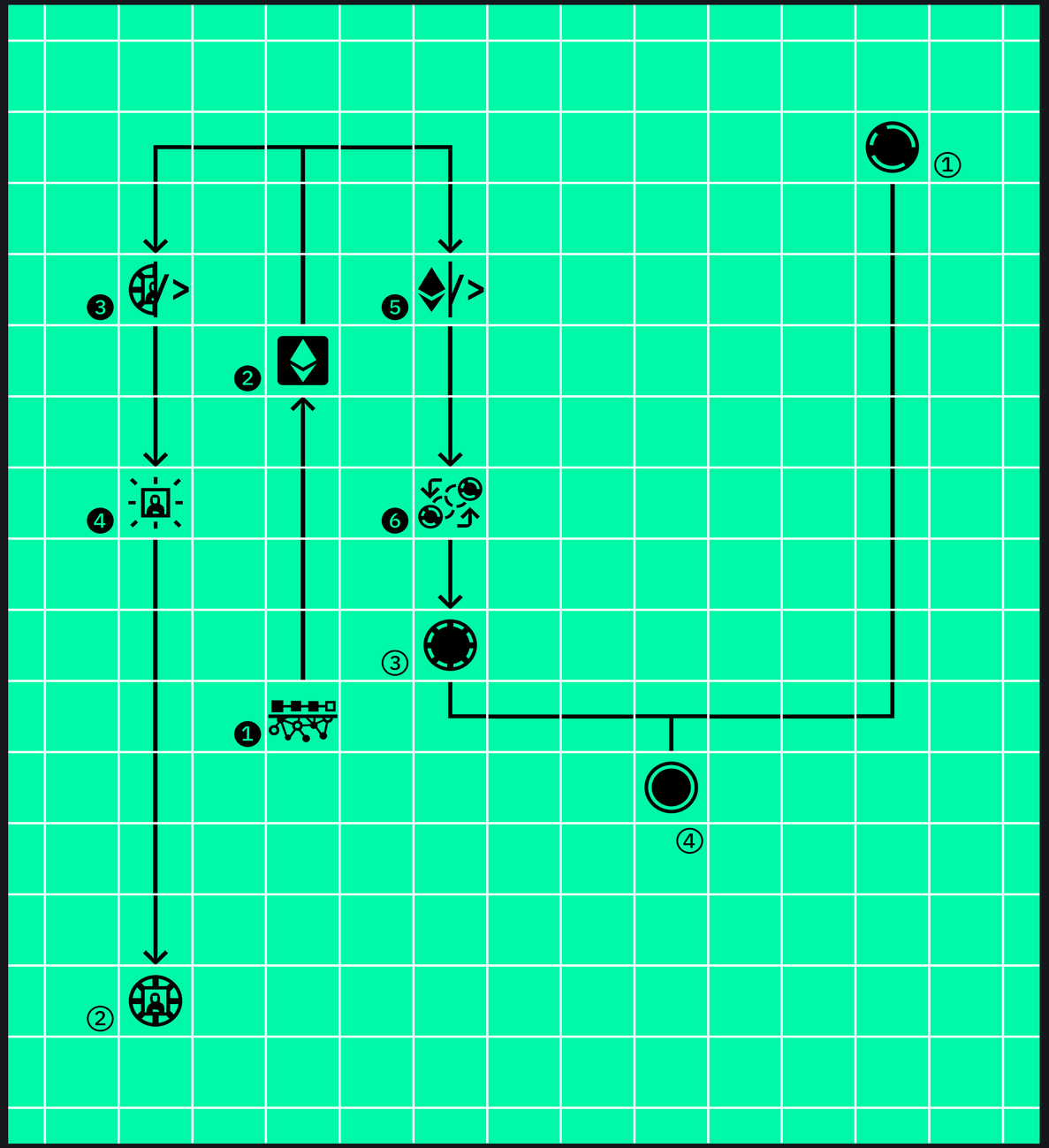
TAXONOMIE DES TOKENS

Cette flexibilité permet une grande paramétrabilité de ces *tokens* (3).



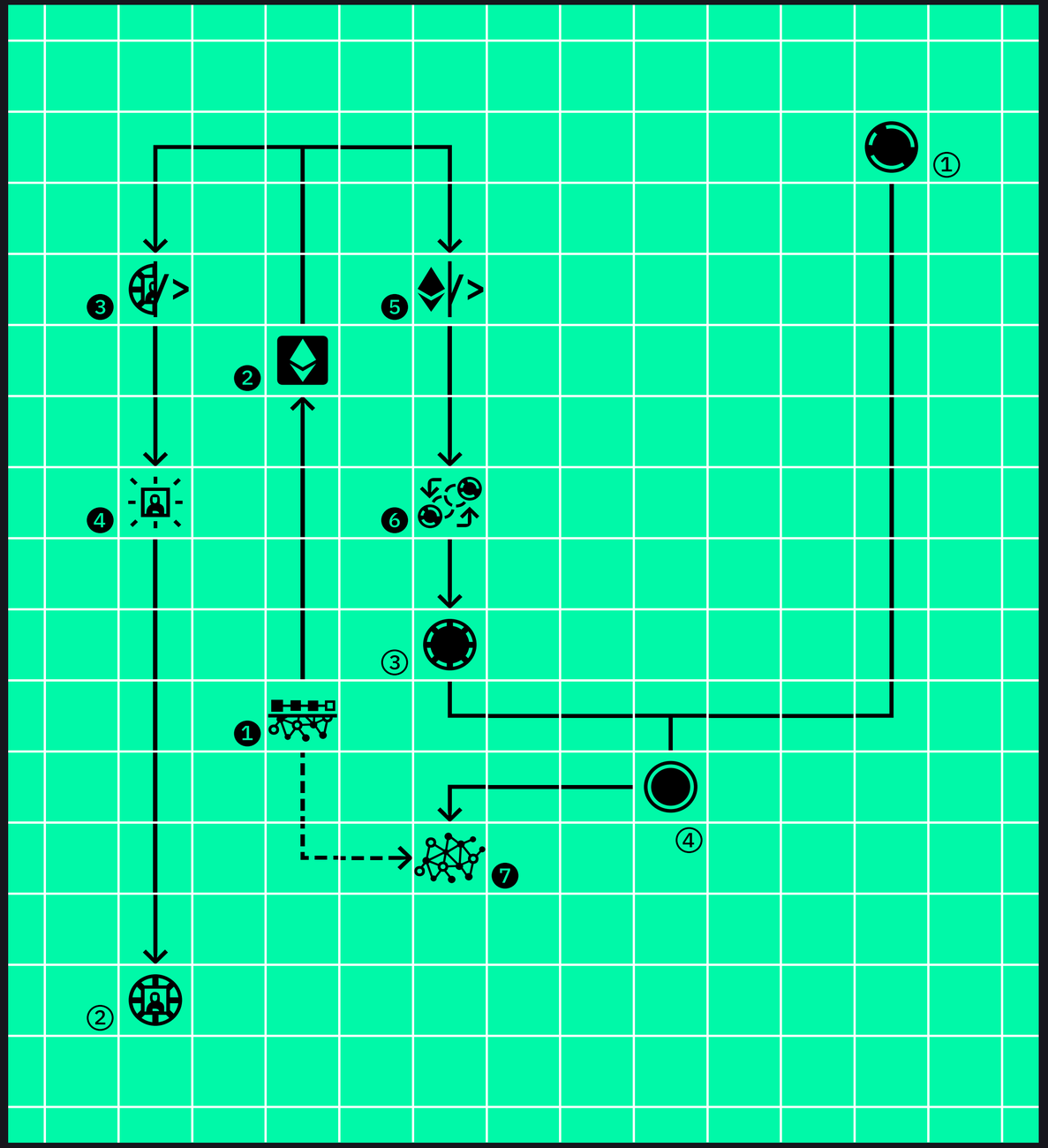
TAXONOMIE DES TOKENS

Si un *token* a une utilisation monétaire, il est désigné par le terme de “*coin*” (4).



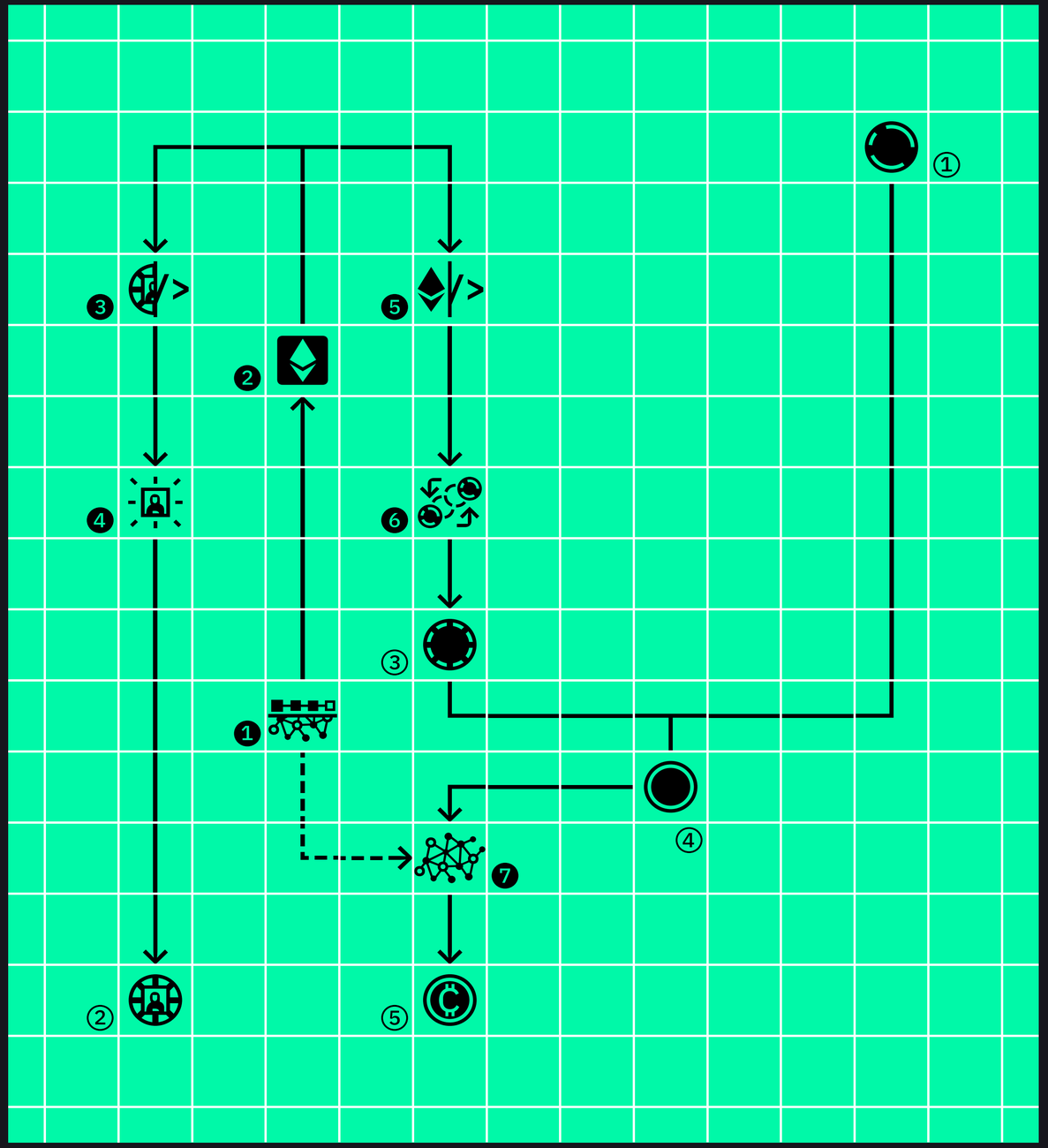
TAXONOMIE DES TOKENS

Ce “*coin*” peut être décentralisé [7] (blockchain publique).



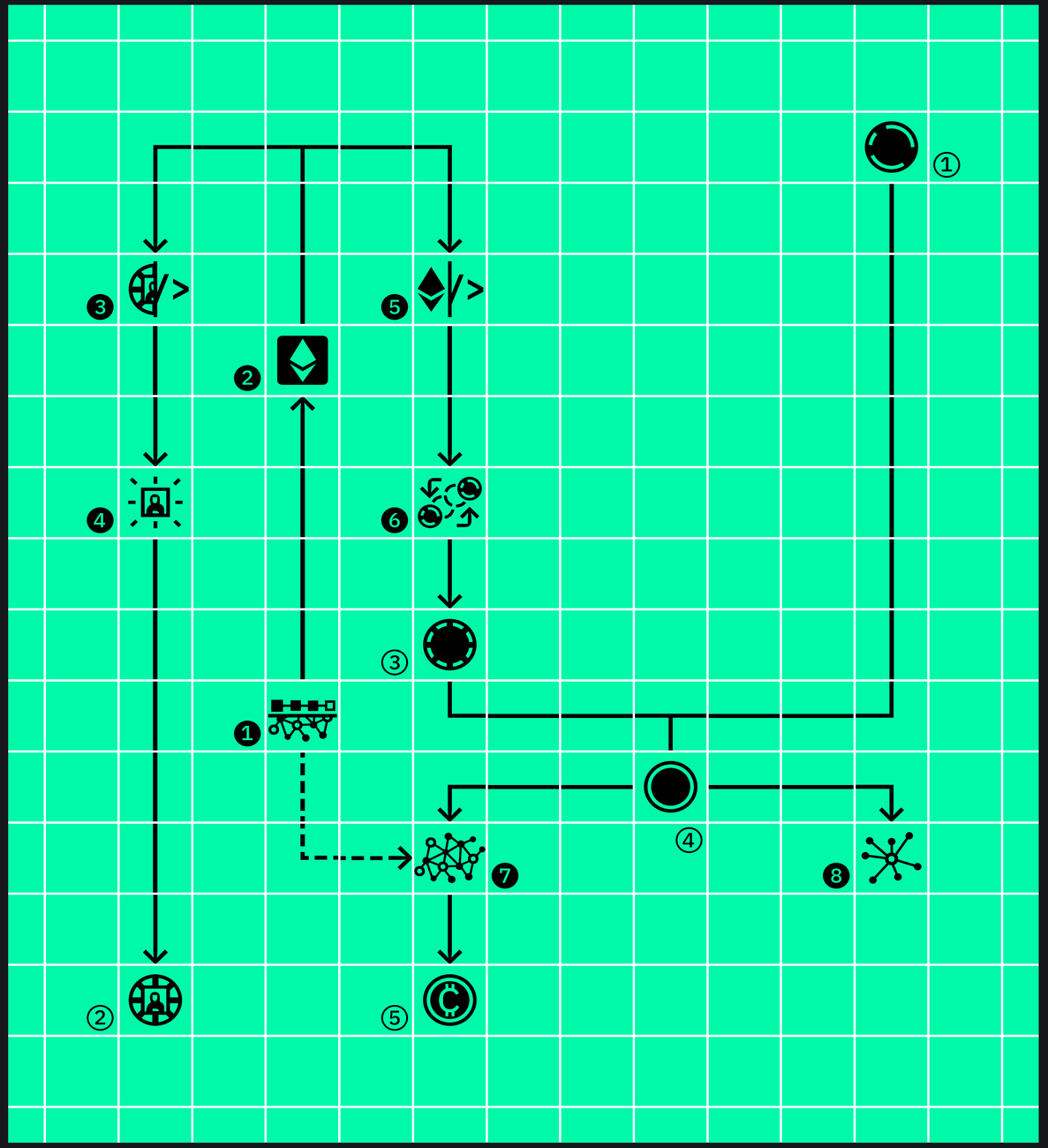
TAXONOMIE DES TOKENS

Cela le désigne par le terme de "cryptomonnaie" (5).



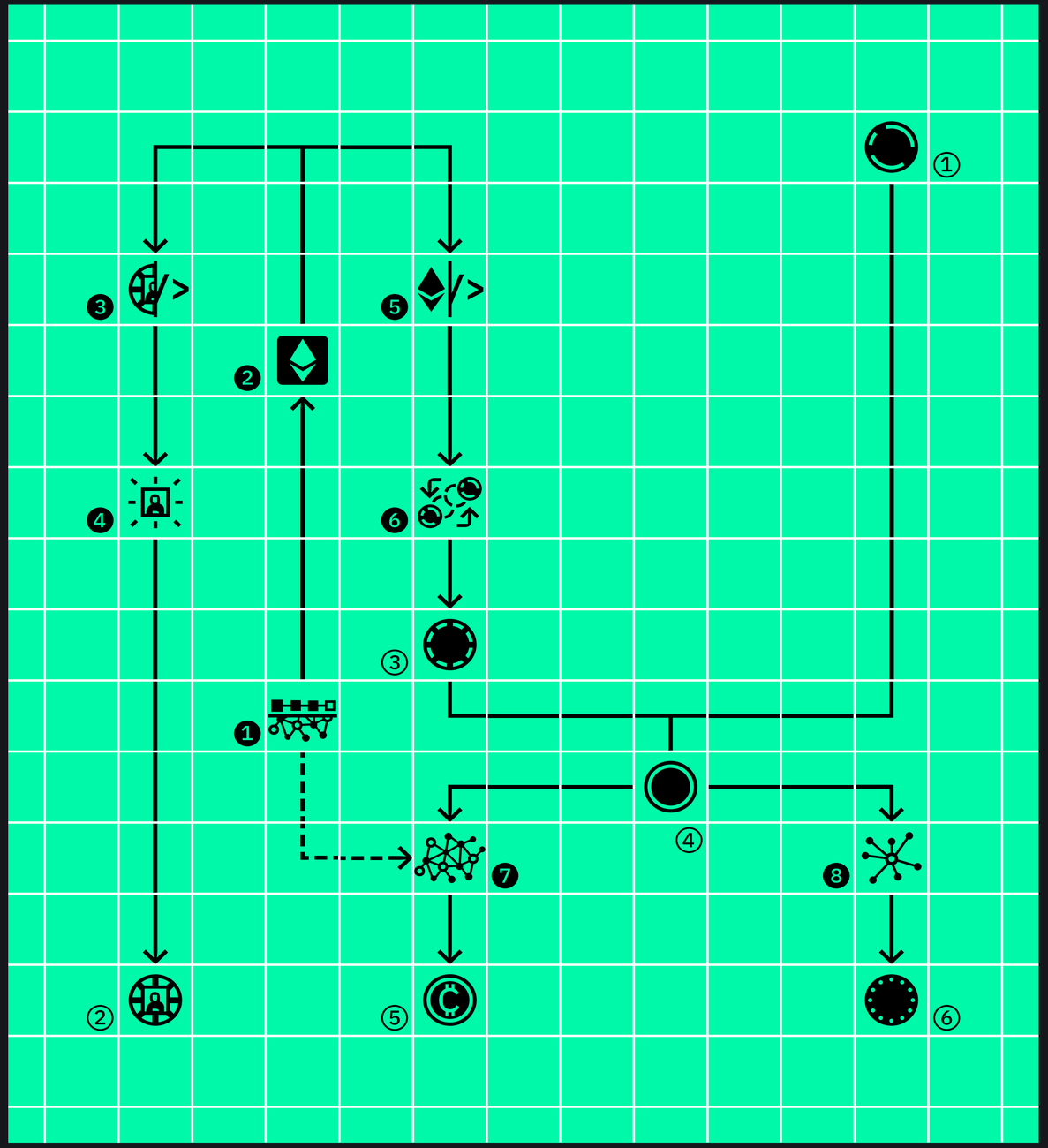
TAXONOMIE DES TOKENS

Si ce “*coin*” fonctionne sur une blockchain gouvernementale [8] (blockchain centralisée)…



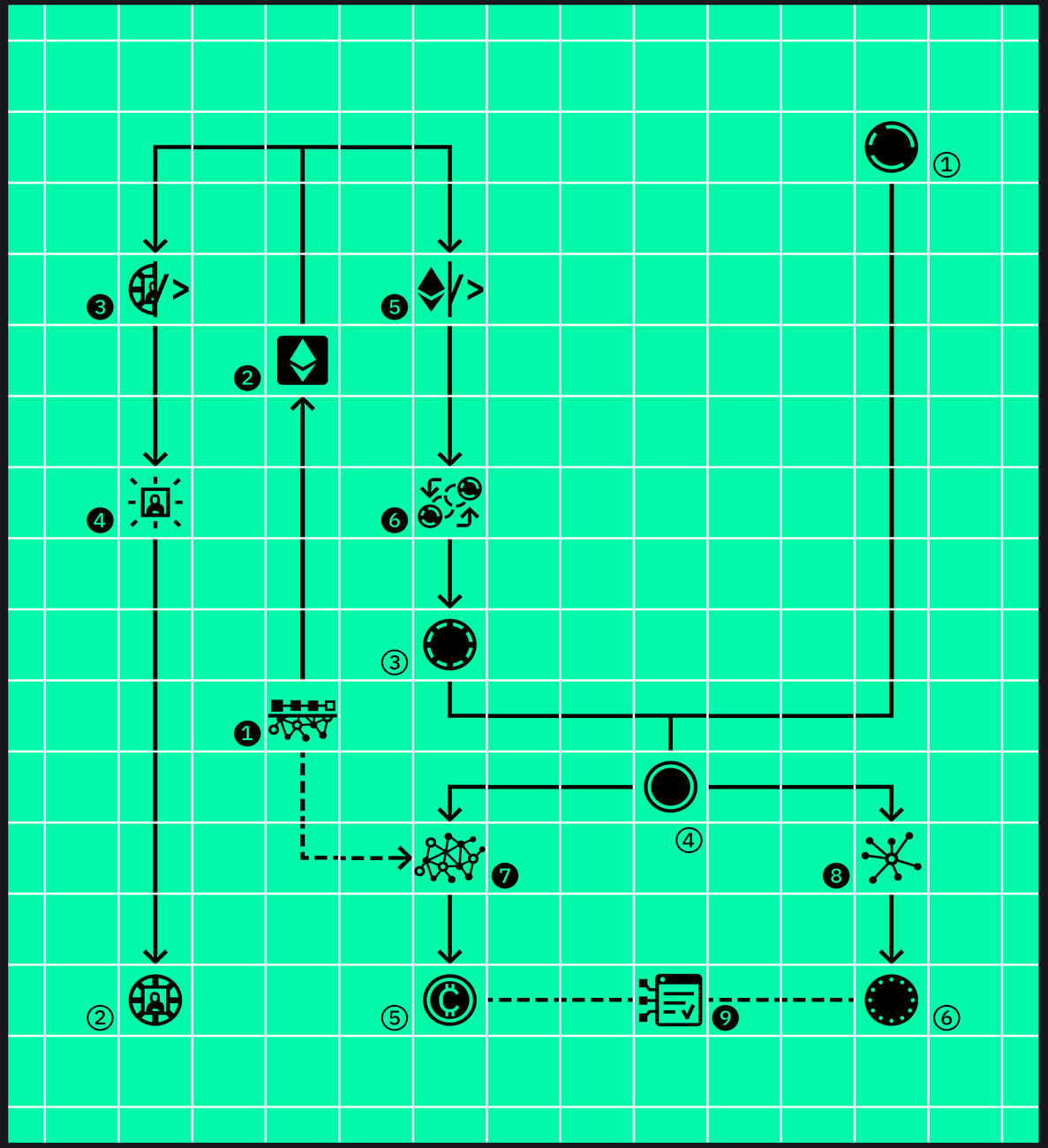
TAXONOMIE DES TOKENS

Alors il se désigne par le terme de *CBDC* (*Central Bank Digital Currency*) [6].



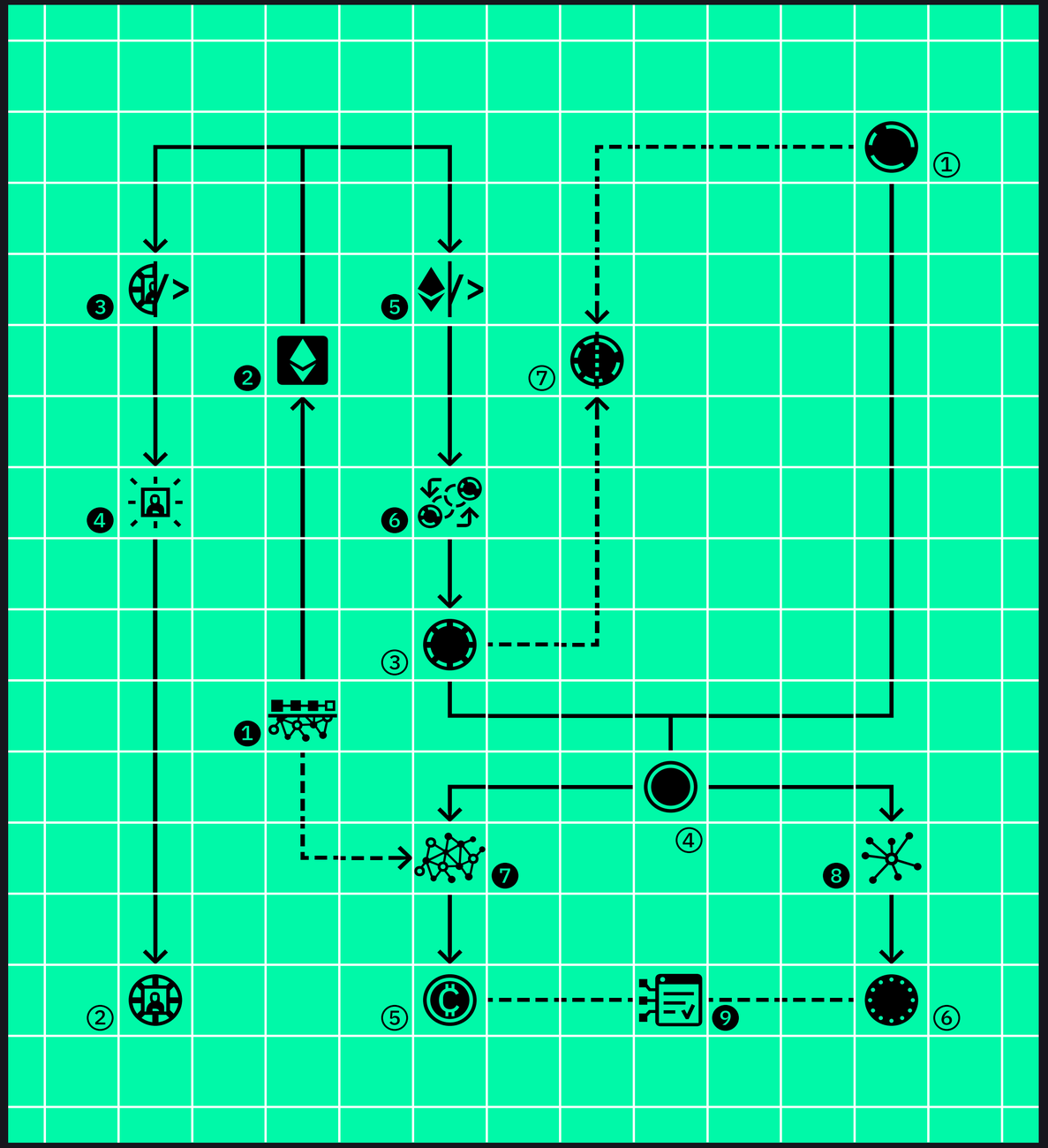
TAXONOMIE DES TOKENS

Cryptomonnaie et *CBDC* ont en commun d'être tout deux défini par un *smart contract* [9].



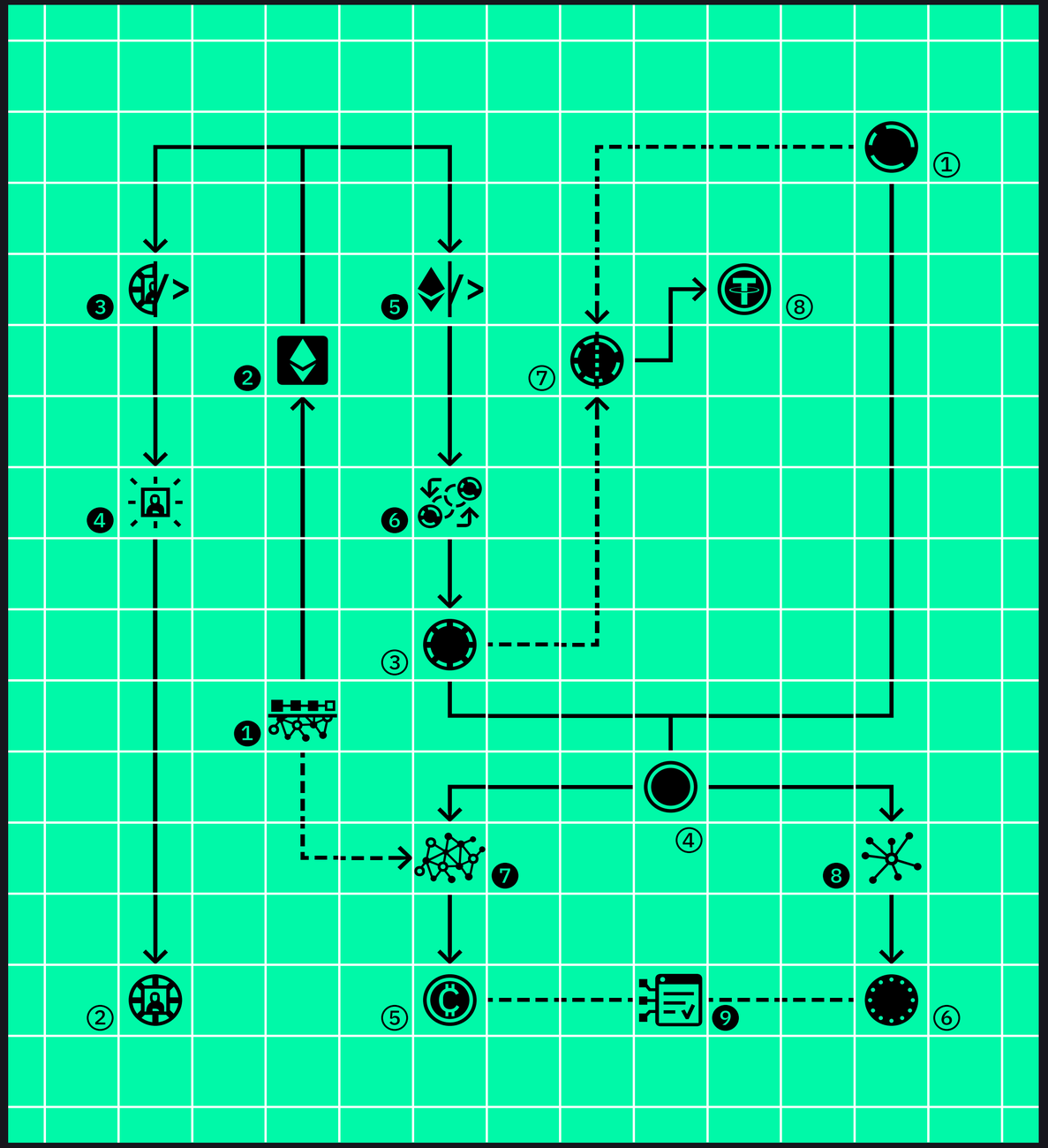
TAXONOMIE DES TOKENS

Les *smart contracts* peuvent également permettre de relier les monnaies FIAT (centralisé) aux *tokens* (décentralisé). C’est de ce lien qu’apparaissent les “*Stablecoin*” (7).



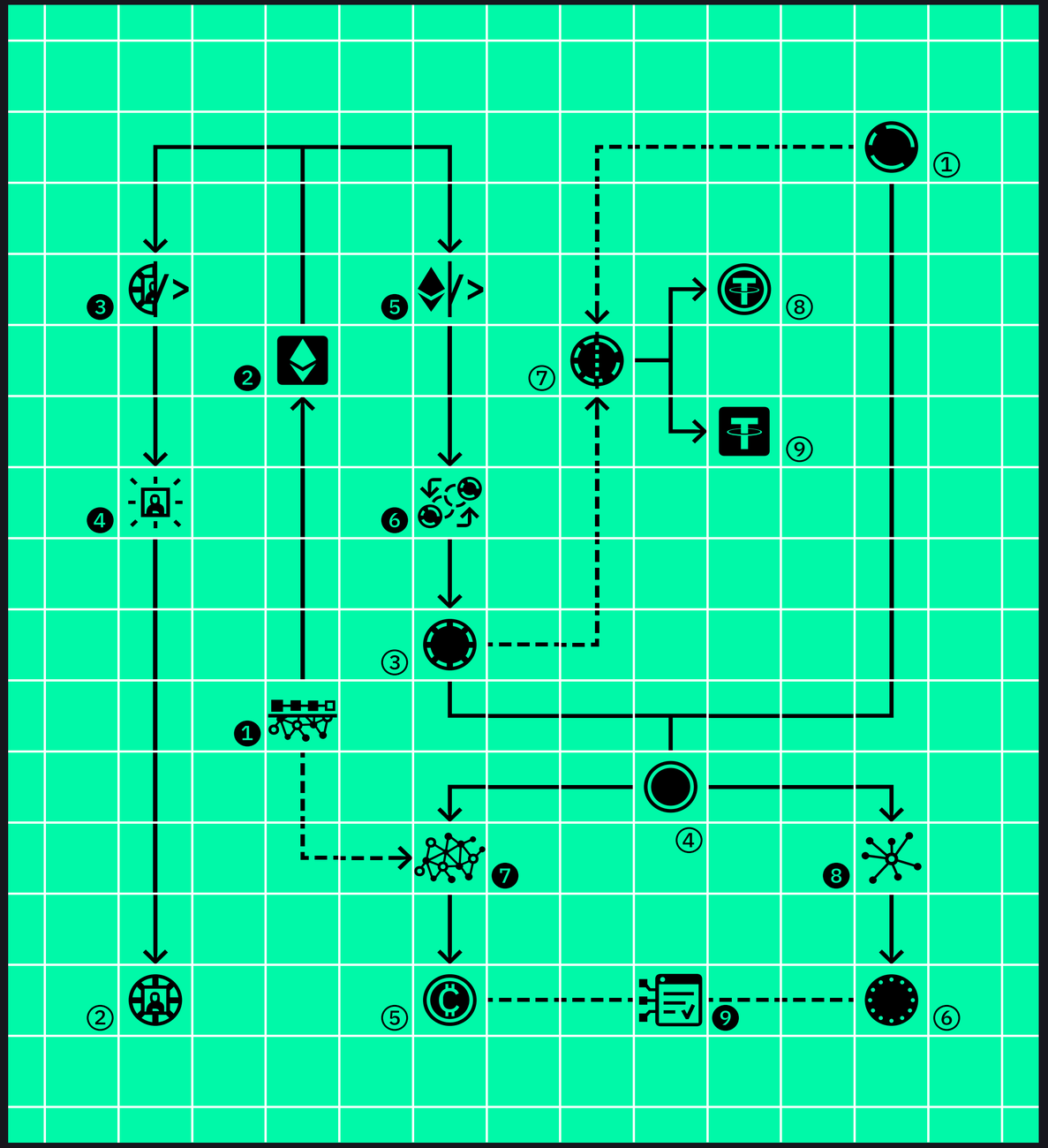
TAXONOMIE DES TOKENS

Lancés en 2014 les tethers [8] ont été l'une des premières forme de *stablecoin*.



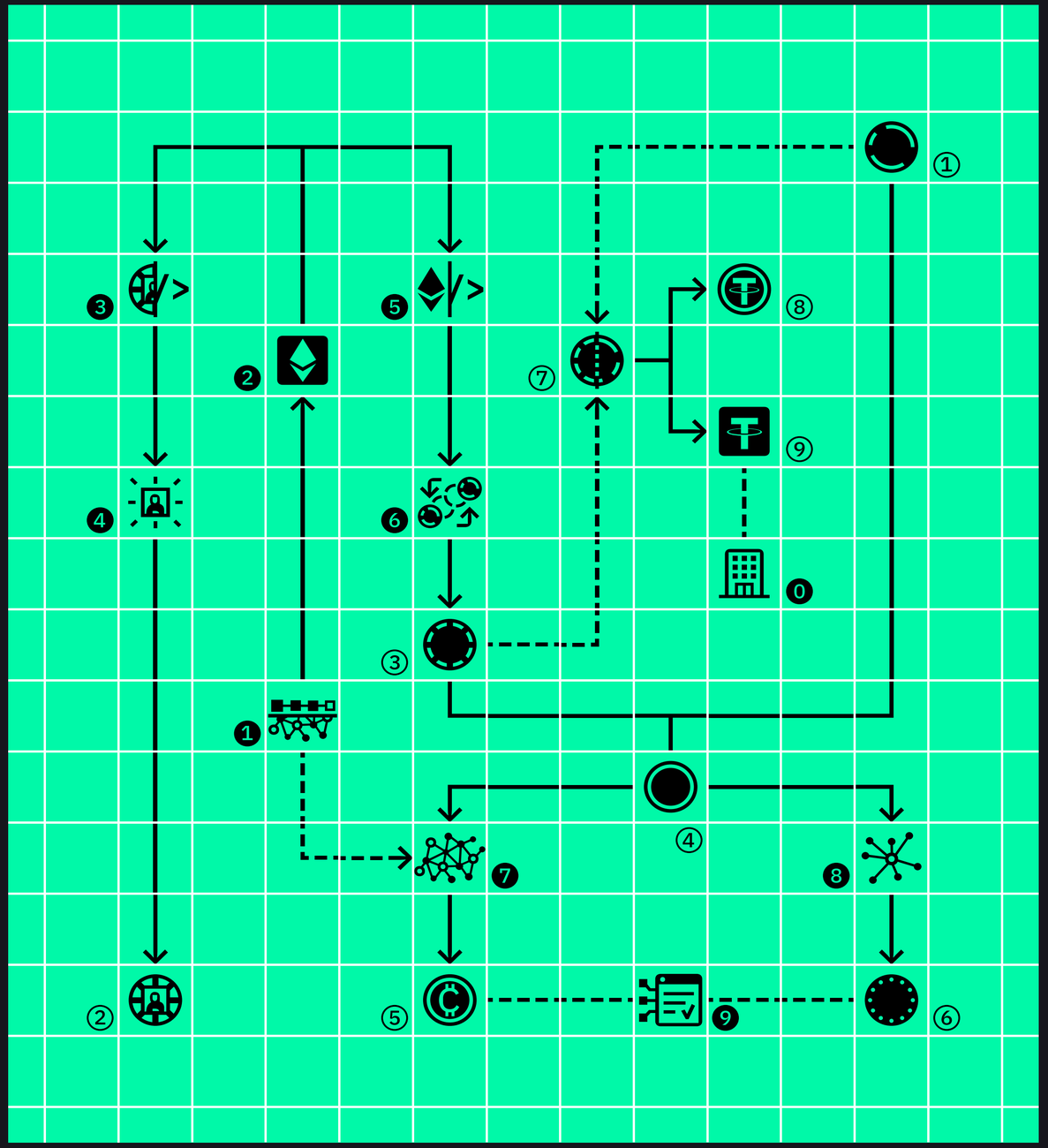
TAXONOMIE DES TOKENS

Ils sont émis par Tether Limited [9] qui est détenu par la société mère basée à Hong Kong : iFinex Inc.



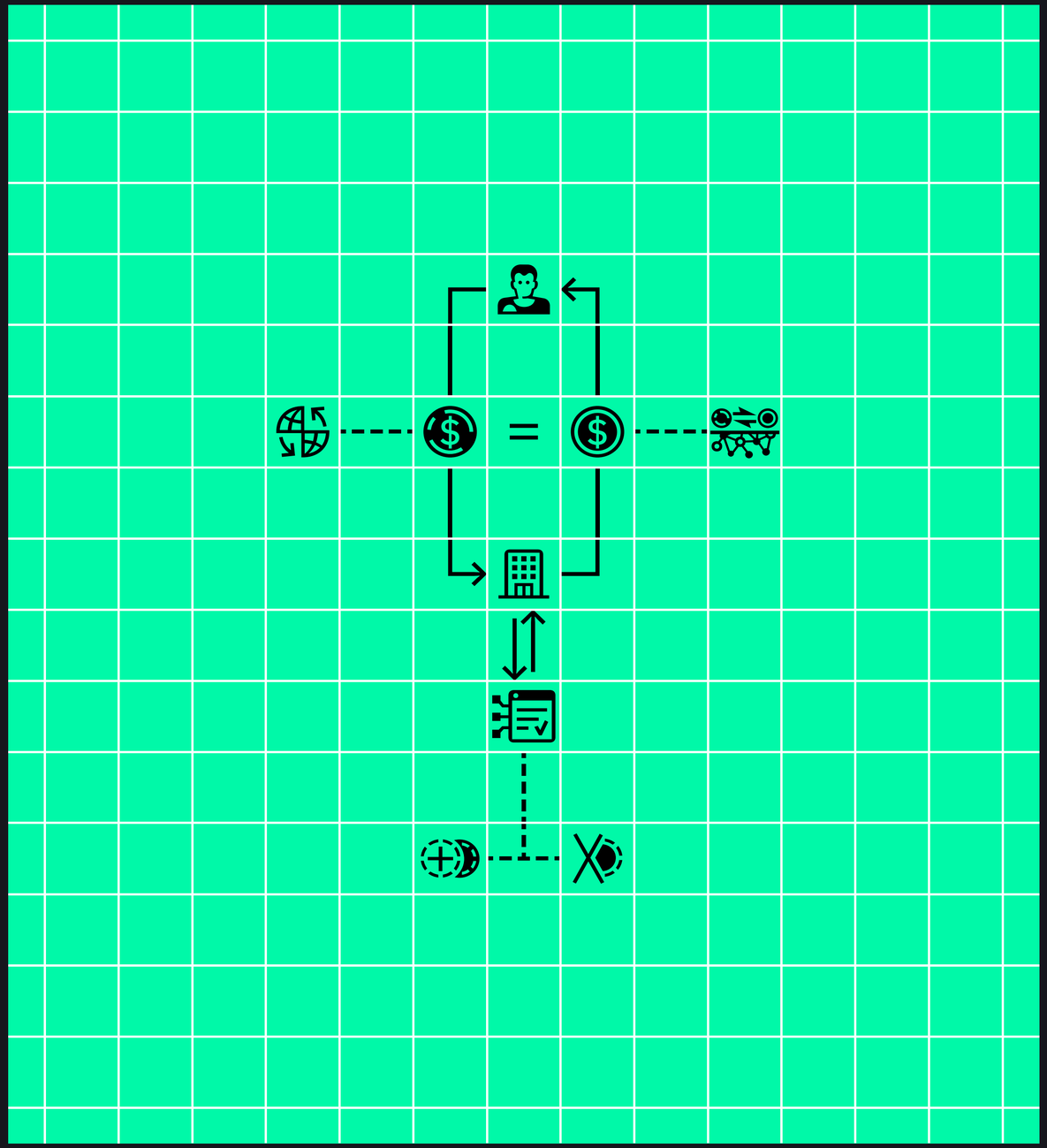
TAXONOMIE DES TOKENS

Cette dernière possède également une autre compagnie [10] de bourse de cryptomonnaies Bitfinex.



PRINCIPE DE BASE

Qui est l’intermédiare ?
Le smart-contract ou la compagnie émettrice ?



DÉMONSTRATION

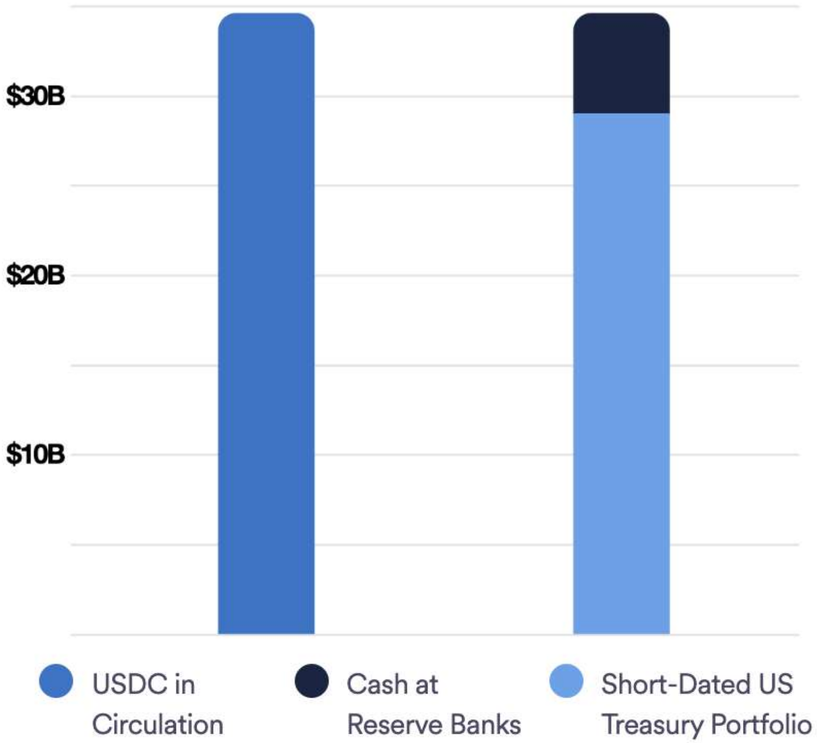
Contrat de “ménage à trois” ?

US Dollar x Circle x USDC

Balances

$34.6B
In Circulation

$34.6B
Reserves[1]



Issuance and Redemption

7 DAY CHANGE

$0.5B
Issued

$2.6B
Redeemed

-$2.0B Change in Circulation

30 DAY CHANGE

$9.2B
Issued

$16.7B
Redeemed

-$7.4B Change in Circulation

365 DAY CHANGE

$174.8B
Issued

$192.6B
Redeemed

-$17.7B Change in Circulation

```
/**
 * @notice Transfer tokens by spending allowance
 * @param from  Payer's address
 * @param to    Payee's address
 * @param value Transfer amount
 * @return True if successful
 */
function transferFrom(
    address from,
    address to,
    uint256 value
)
    external
    override
    whenNotPaused
    notBlacklisted(msg.sender)
    notBlacklisted(from)
    notBlacklisted(to)
    returns (bool)
{
    require(
        value <= allowed[from][msg.sender],
        "ERC20: transfer amount exceeds allowance"
    );
    _transfer(from, to, value);
    allowed[from][msg.sender] = allowed[from][msg.sender].sub(value);
    return true;
}

/**
 * @notice Transfer tokens from the caller
 * @param to    Payee's address
 * @param value Transfer amount
 * @return True if successful
 */
function transfer(address to, uint256 value)
    external
    override
    whenNotPaused
    notBlacklisted(msg.sender)
    notBlacklisted(to)
    returns (bool)
{
    _transfer(msg.sender, to, value);
    return true;
}
```

```
/**
 * @notice Increase the allowance by a given increment
 * @param spender   Spender's address
 * @param increment Amount of increase in allowance
 * @return True if successful
 */
function increaseAllowance(address spender, uint256 increment)
    external
    whenNotPaused
    notBlacklisted(msg.sender)
    notBlacklisted(spender)
    returns (bool)
{
    _increaseAllowance(msg.sender, spender, increment);
    return true;
}

/**
 * @notice Internal function to increase the allowance by a given increment
 * @param owner     Token owner's address
 * @param spender   Spender's address
 * @param increment Amount of increase
 */
function _increaseAllowance(
    address owner,
    address spender,
    uint256 increment
) internal override {
    _approve(owner, spender, allowed[owner][spender].add(increment));
}

/**
 * @dev Internal function to set allowance
 * @param owner     Token owner's address
 * @param spender   Spender's address
 * @param value     Allowance amount
 */
function _approve(
    address owner,
    address spender,
    uint256 value
) internal override {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");
    allowed[owner][spender] = value;
    emit Approval(owner, spender, value);
}
```

```solidity
/**
 * @dev Function to mint tokens
 * @param _to The address that will receive the minted tokens.
 * @param _amount The amount of tokens to mint. Must be less than or equal
 * to the minterAllowance of the caller.
 * @return A boolean that indicates if the operation was successful.
 */
function mint(address _to, uint256 _amount)
    external
    whenNotPaused
    onlyMinters
    notBlacklisted(msg.sender)
    notBlacklisted(_to)
    returns (bool)
{
    require(_to != address(0), "FiatToken: mint to the zero address");
    require(_amount > 0, "FiatToken: mint amount not greater than 0");

    uint256 mintingAllowedAmount = minterAllowed[msg.sender];
    require(
        _amount <= mintingAllowedAmount,
        "FiatToken: mint amount exceeds minterAllowance"
    );

    totalSupply_ = totalSupply_.add(_amount);
    balances[_to] = balances[_to].add(_amount);
    minterAllowed[msg.sender] = mintingAllowedAmount.sub(_amount);
    emit Mint(msg.sender, _to, _amount);
    emit Transfer(address(0), _to, _amount);
    return true;
}
```

USDC
Minting

Merci