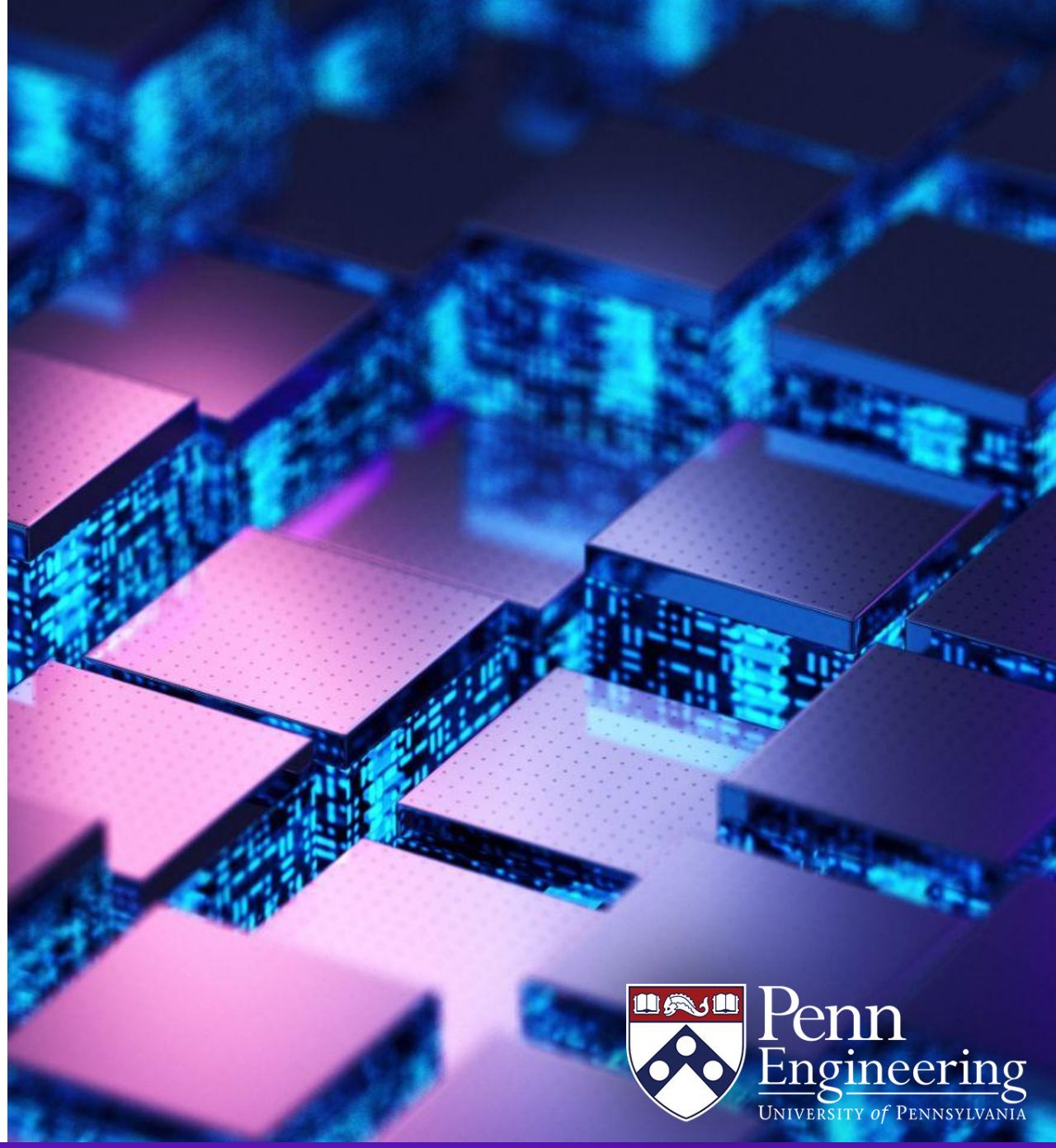
EAS 5830: BLOCKCHAINS

ERC-721s

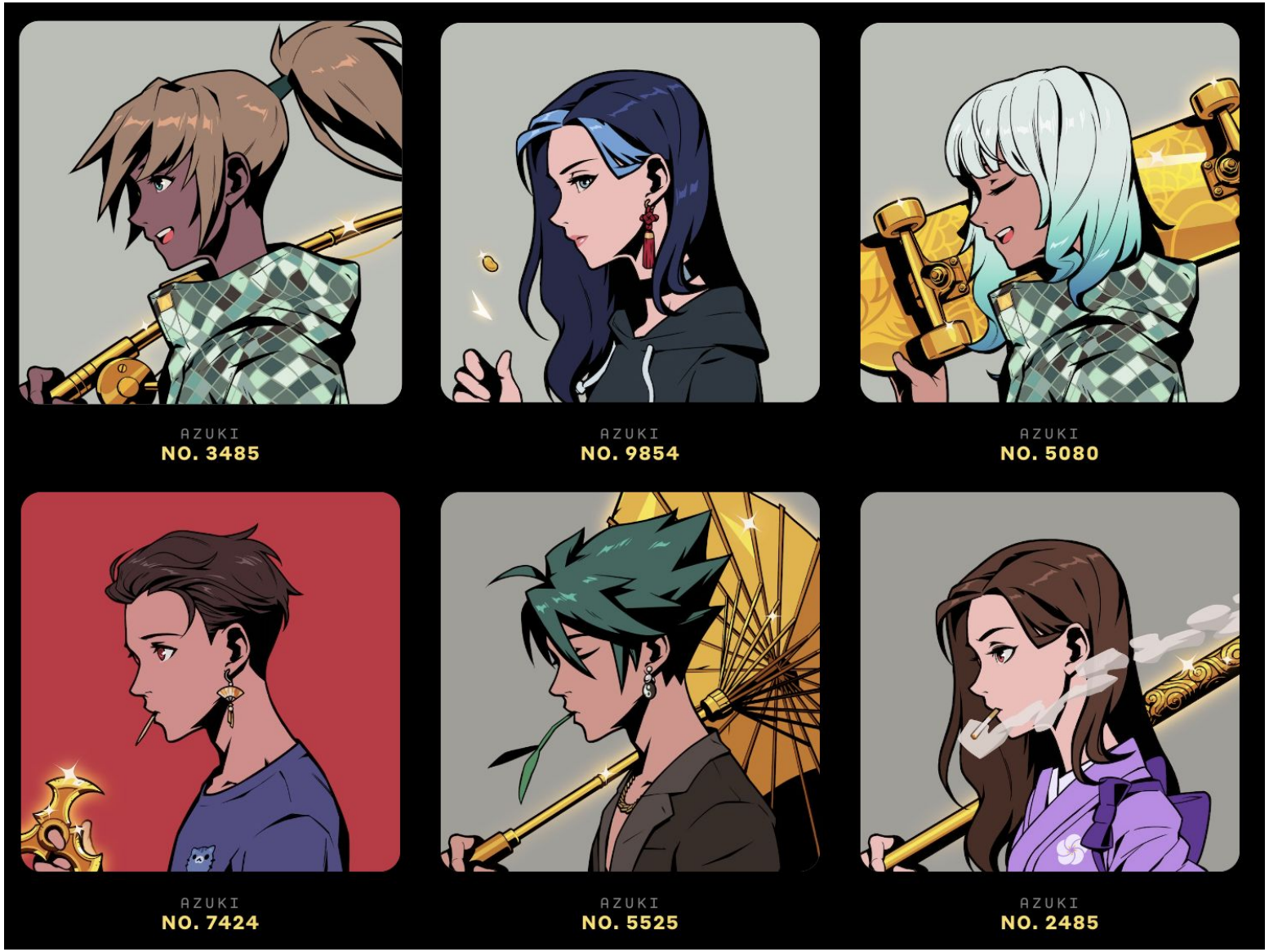
Professor Brett Hemenway Falk

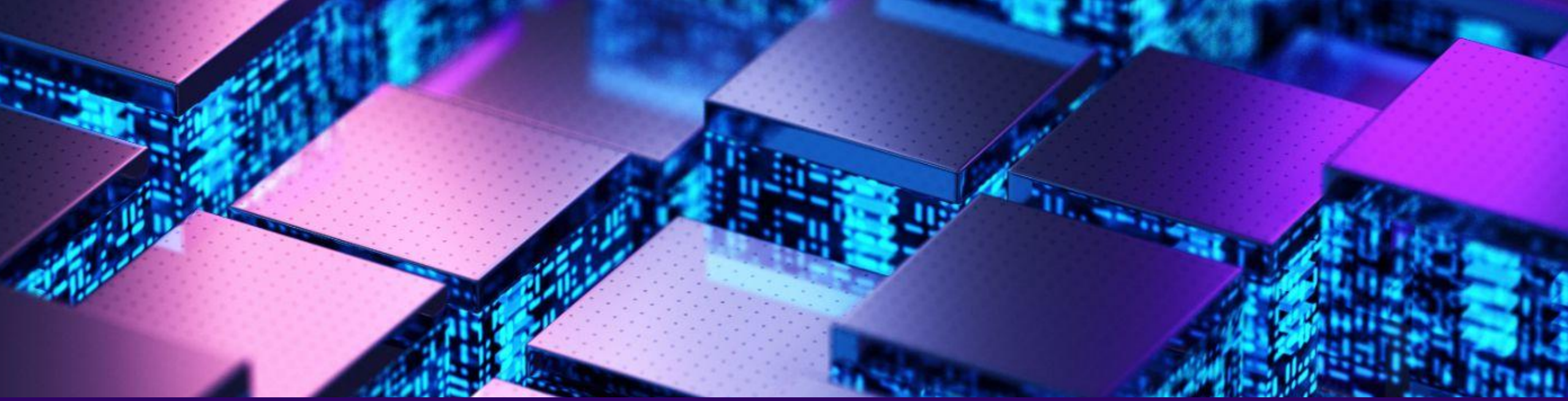


Penn Engineering
University of Pennsylvania

Fungible and Non-Fungible tokens

- ERC-20
  - Fungible
    - Every USDC is the same, you just need to keep track of how many each user has
- ERC-721
  - Non-fungible
    - Every Bored Ape is Different, so you need to keep track of *which* Bored Apes each user owns


AZUKI
NO. 3485
AZUKI
NO. 9854
AZUKI
NO. 5080
AZUKI
NO. 7424
AZUKI
NO. 5525
AZUKI
NO. 2485



The ERC-721 Standard

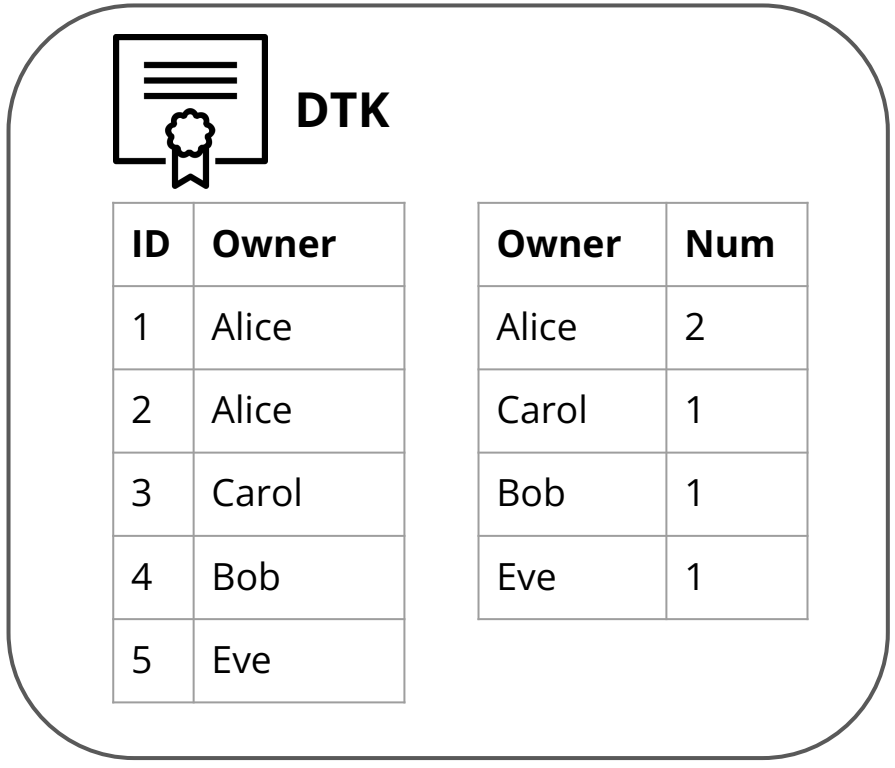
```
function balanceOf(address _owner) external view returns (uint256);
function ownerOf(uint256 _tokenId) external view returns (address);

function transferFrom(address _from, address _to, uint256 _tokenId) external payable;
function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) external payable;
function safeTransferFrom(address _from, address _to, uint256 _tokenId) external payable;

function approve(address _approved, uint256 _tokenId) external payable;
function setApprovalForAll(address _operator, bool _approved) external;
function getApproved(uint256 _tokenId) external view returns (address);
function isApprovedForAll(address _owner, address _operator) external view returns (bool);
```

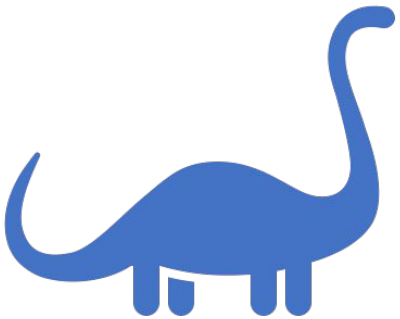
```
22          // Token name
23          string private _name;
24
25          // Token symbol
26          string private _symbol;
27
28          mapping(uint256 tokenId => address) private _owners;
29
30          mapping(address owner => uint256) private _balances;
31
32          mapping(uint256 tokenId => address) private _tokenApprovals;
```

Transfers



ID	Owner
1	Alice
2	Alice
3	Carol
4	Bob
5	Eve

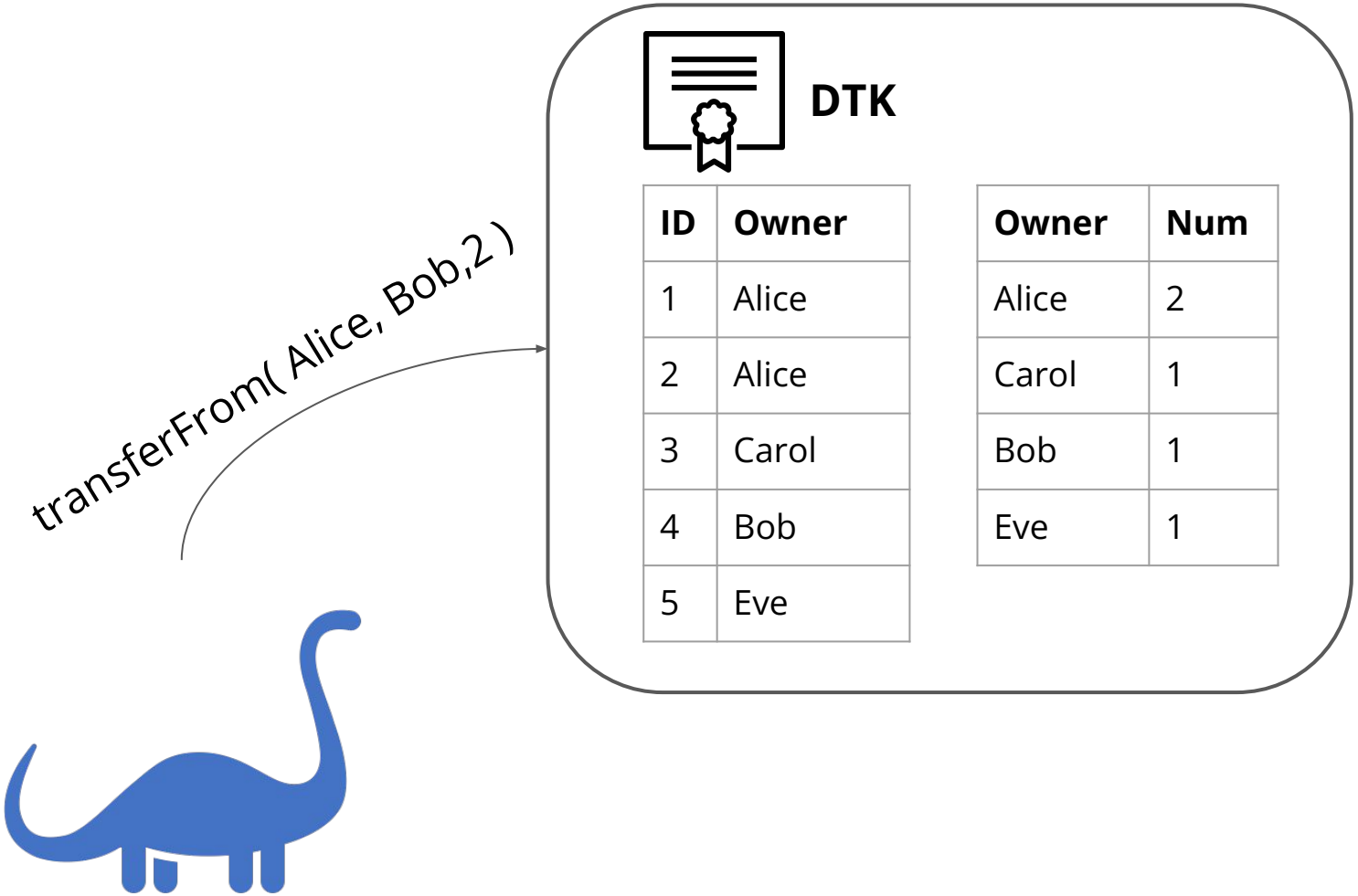
Owner	Num
Alice	2
Carol	1
Bob	1
Eve	1



Alice

Bob

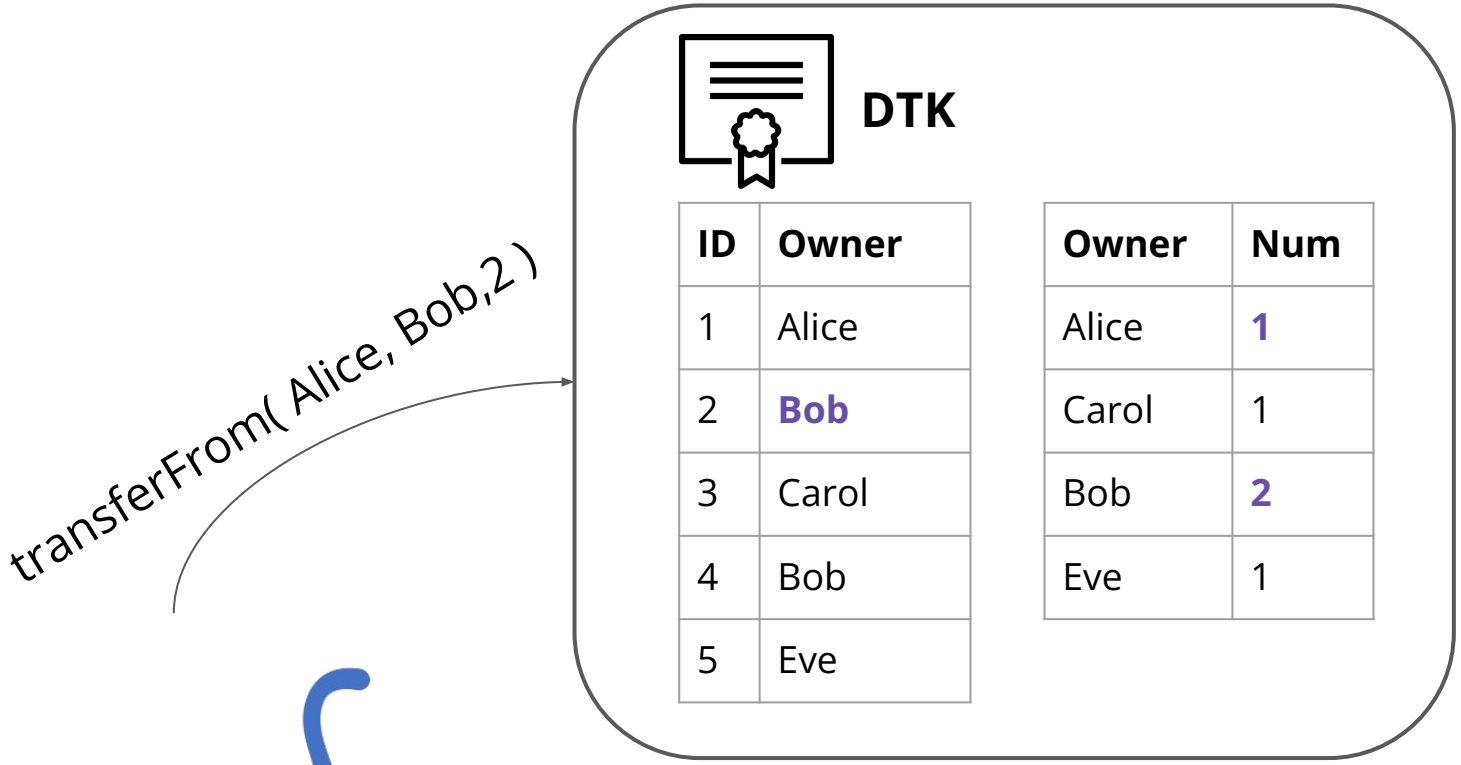
Transfers

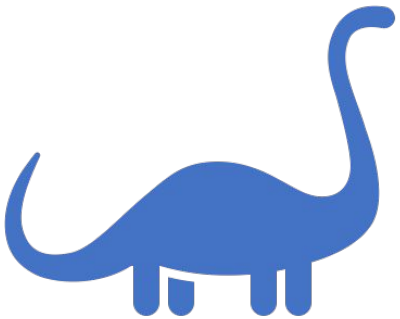


Alice

Bob

Transfers





Alice

Bob

Approvals

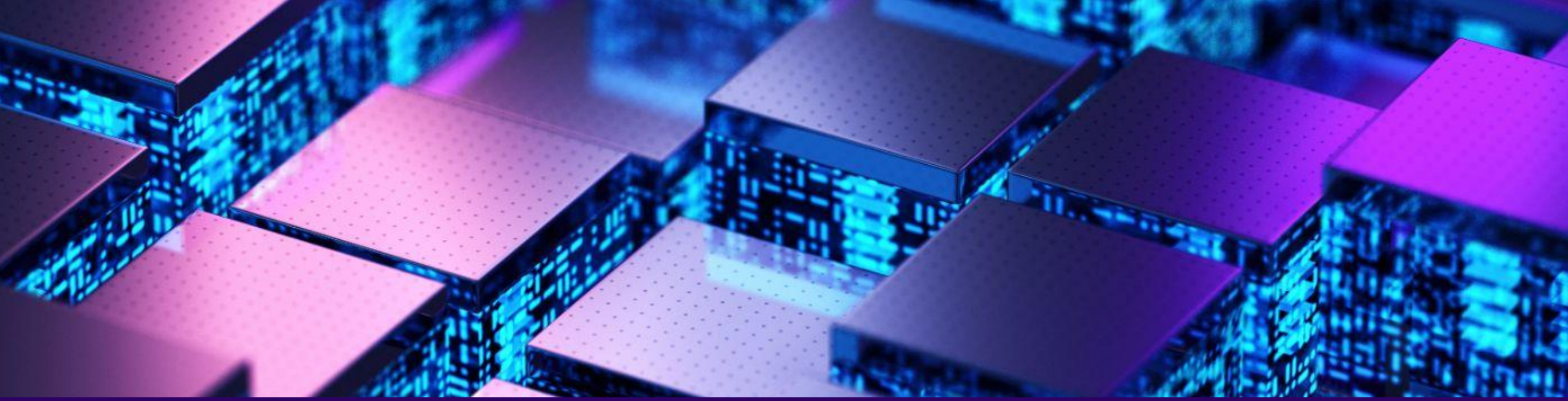
- transferFrom checks

Approvals

- transferFrom checks
  - msg.sender == from

Approvals

```
function _isAuthorized(address owner, address spender, uint256 tokenId) internal view virtual returns (bool) {
    return
        spender != address(0) &&
        (owner == spender || isApprovedForAll(owner, spender) || _getApproved(tokenId) == spender);
}
```



Metadata

ERC-721

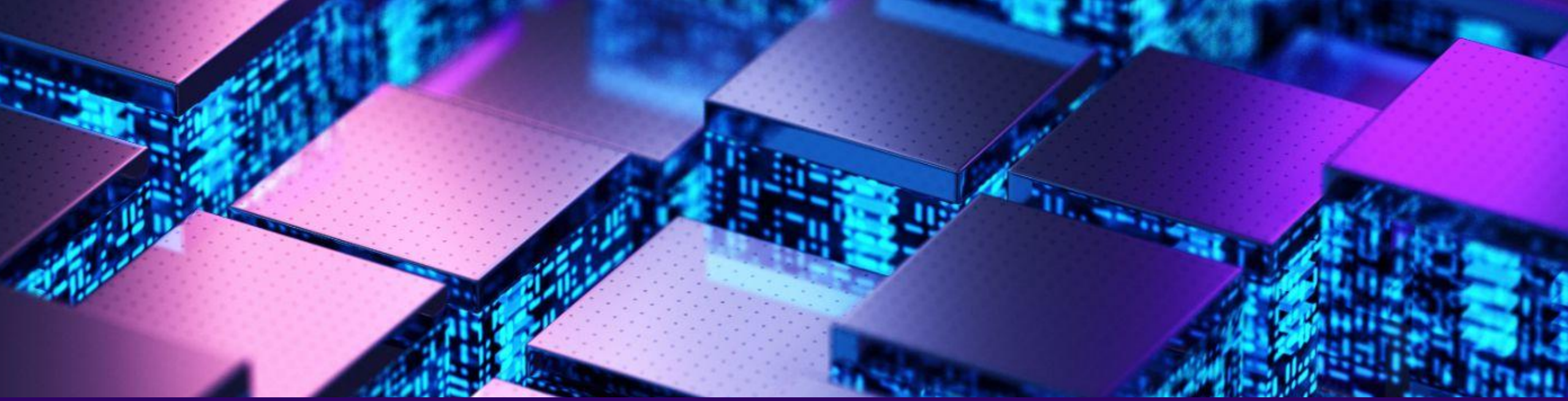
```
interface ERC721Metadata /* is ERC721 */ {
    /// @notice A descriptive name for a collection of NFTs in this contract
    function name() external view returns (string _name);

    /// @notice An abbreviated name for NFTs in this contract
    function symbol() external view returns (string _symbol);

    /// @notice A distinct Uniform Resource Identifier (URI) for a given asset.
    /// @dev Throws if `_tokenId` is not a valid NFT. URIs are defined in RFC
    ///  3986. The URI may point to a JSON file that conforms to the "ERC721
    ///  Metadata JSON Schema".
    function tokenURI(uint256 _tokenId) external view returns (string);
}
```

ERC-721


```
{
   "title": "Asset Metadata",
   "type": "object",
   "properties": {
      "name": {
         "type": "string",
         "description": "Identifies the asset to which this NFT represents"
      },
      "description": {
         "type": "string",
         "description": "Describes the asset to which this NFT represents"
      },
      "image": {
         "type": "string",
         "description": "A URI pointing to a resource with mime type image/* representing the asset to which this NFT represents. Consider making any images at a width between 320 and 1080 pixels and aspect ratio between 1.91:1 and 4:5 inclusive."
      }
   }
}
```



Minting

Minting

- You can hard-code a bunch of tokenIds into the contract when it’s launched

Minting

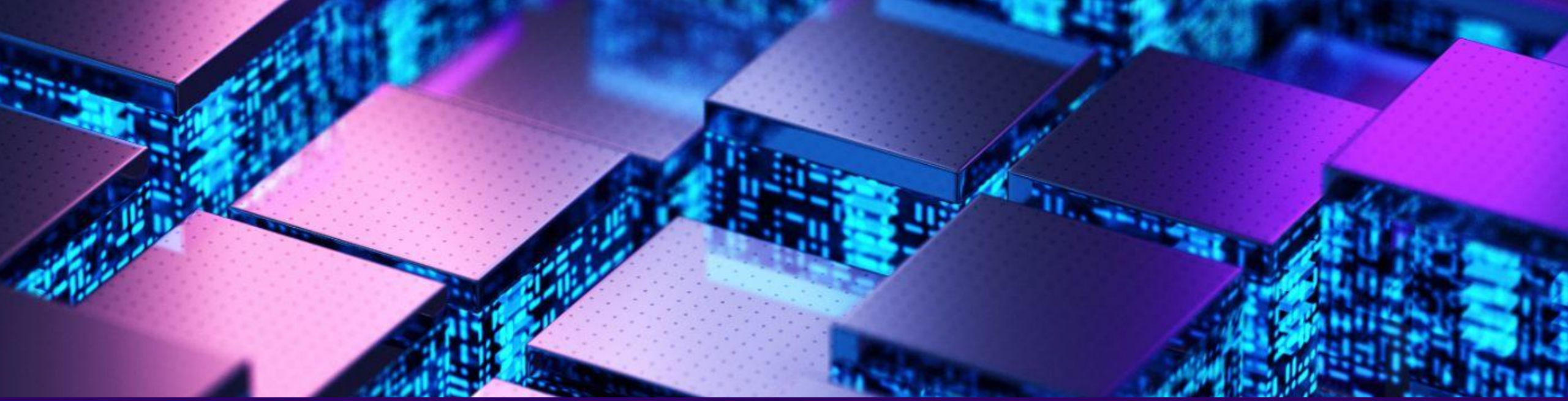
- You can hard-code a bunch of tokenIds into the contract when it’s launched
- You can specify tokenIds as a constructor argument

Minting

- You can hard-code a bunch of tokenIds into the contract when it’s launched
- You can specify tokenIds as a constructor argument
- You can “mint” tokens on demand

```
1974      function mintApe(uint numberOfTokens) public payable {
1975          require(saleIsActive, "Sale must be active to mint Ape");
1976          require(numberOfTokens <= maxApePurchase, "Can only mint 20 tokens at a time");
1977          require(totalSupply().add(numberOfTokens) <= MAX_APES, "Purchase would exceed max supply of Apes");
1978          require(apePrice.mul(numberOfTokens) <= msg.value, "Ether value sent is not correct");
1979
1980          for(uint i = 0; i < numberOfTokens; i++) {
1981              uint mintIndex = totalSupply();
1982              if (totalSupply() < MAX_APES) {
1983                  _safeMint(msg.sender, mintIndex);
1984              }
1985          }
1986
1987          // If we haven't set the starting index and this is either 1) the last saleable token or 2) the first token to be sold
1988          // the end of pre-sale, set the starting index block
1989          if (startingIndexBlock == 0 && (totalSupply() == MAX_APES || block.timestamp >= REVEAL_TIMESTAMP)) {
1990              startingIndexBlock = block.number;
1991          }
1992      }
```

```
47    function auctionMint(uint256 quantity) external payable callerIsUser {
48      uint256 _saleStartTime = uint256(saleConfig.auctionSaleStartTime);
49      require(
50        _saleStartTime != 0 && block.timestamp >= _saleStartTime,
51        "sale has not started yet"
52      );
53      require(
54        totalSupply() + quantity <= amountForAuctionAndDev,
55        "not enough remaining reserved for auction to support desired mint amount"
56      );
57      require(
58        numberMinted(msg.sender) + quantity <= maxPerAddressDuringMint,
59        "can not mint this many"
60      );
61      uint256 totalCost = getAuctionPrice(_saleStartTime) * quantity;
62      _safeMint(msg.sender, quantity);
63      refundIfOver(totalCost);
64    }
```



Administrators

```
1938        function reserveApes() public onlyOwner {
1939            uint supply = totalSupply();
1940            uint i;
1941            for (i = 0; i < 30; i++) {
1942                _safeMint(msg.sender, supply + i);
1943            }
1944        }
```

```
1960        function setBaseURI(string memory baseURI) public onlyOwner {
1961            _setBaseURI(baseURI);
1962        }
```

Admin accounts

- The contract can give “superusers” special powers

Admin accounts

- The contract can give “superusers” special powers
  - CryptoKitties contract can be paused, preventing all future transfers

Admin accounts

- The contract can give “superusers” special powers
  - CryptoKitties contract can be paused, preventing all future transfers
- An ERC-721 contract with an additional “seizure” function can pull tokens from another user