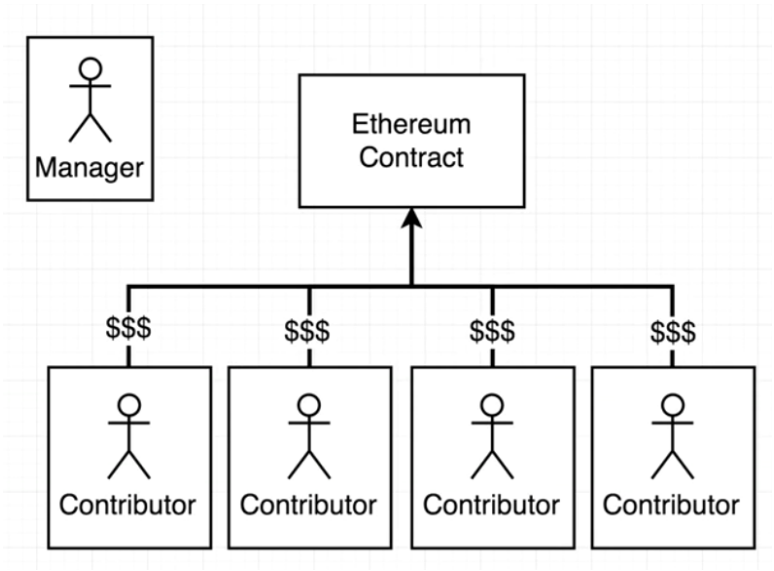
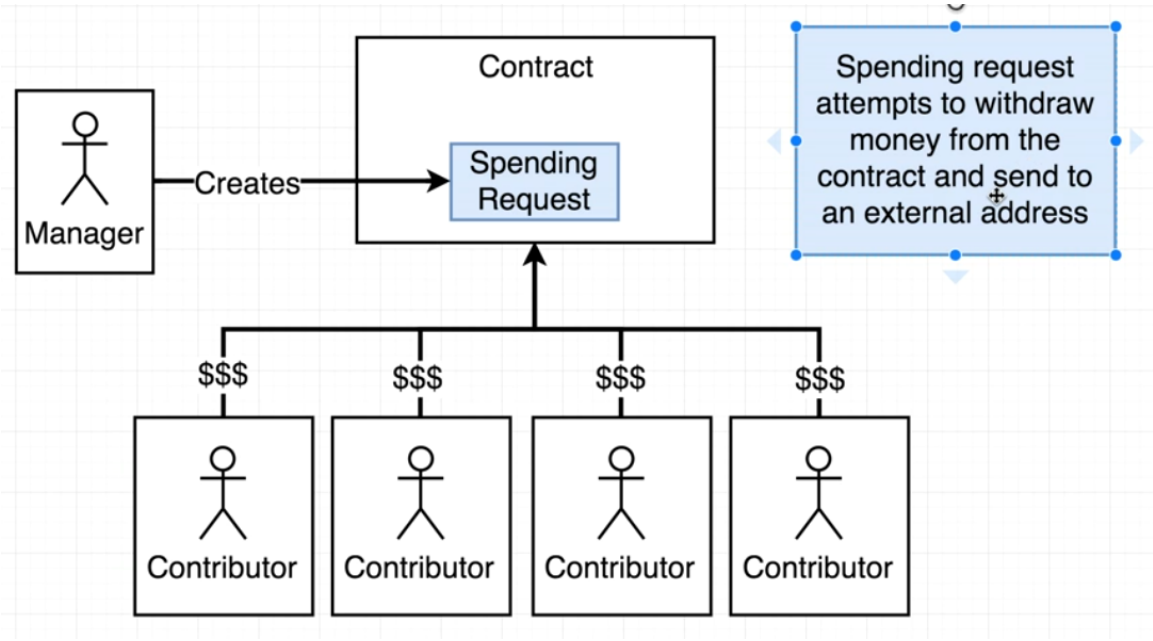
Real World Project With Ethereum

Kickstart Simulation Smart Contract

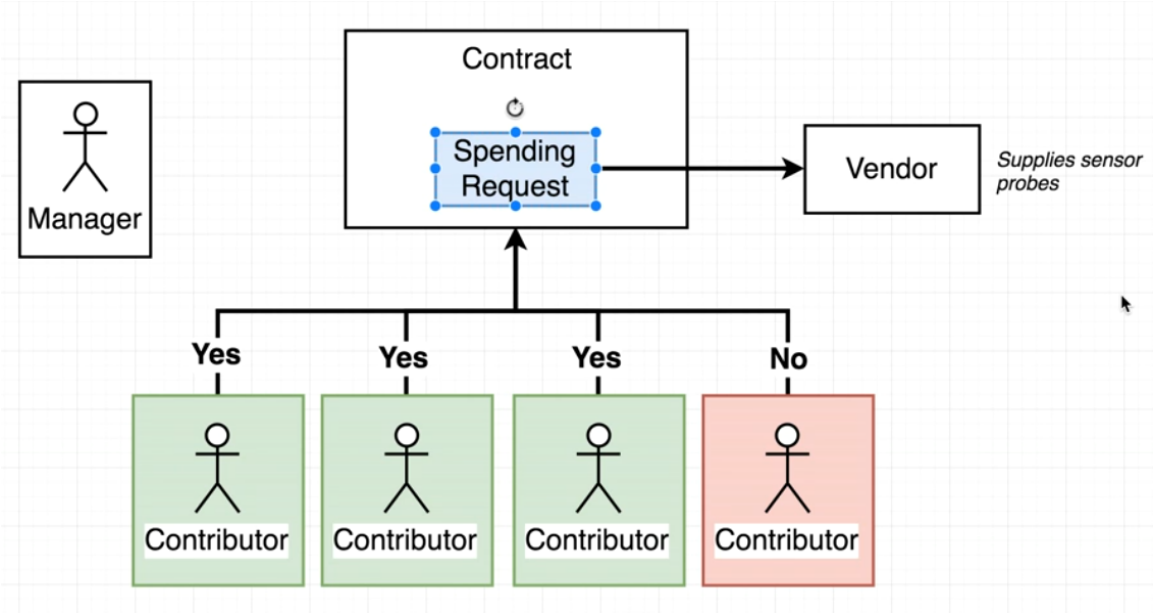
- The problem with kickstart is that a person with malicious intent can misuse the money.



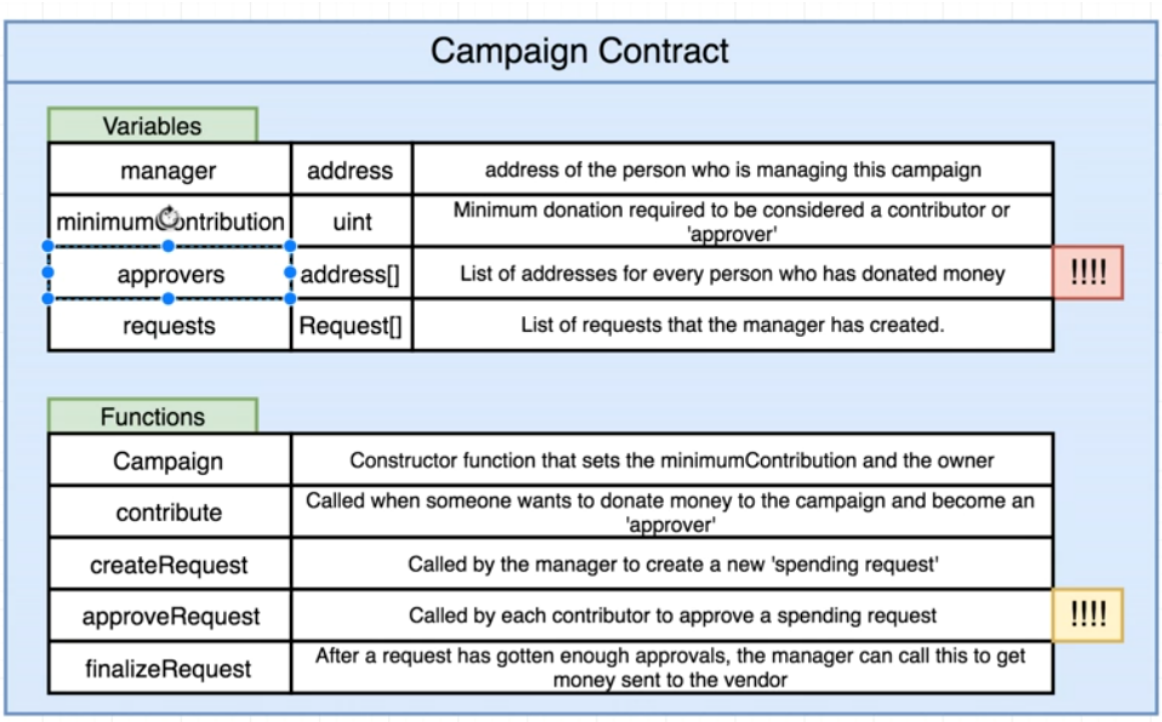
- We need to somehow control the outflow of money from idea manager to vendors.



- The contributors can vote (yes or no) corresponding to a spending request.



- Structure of the Campaign smart contract -



Campaign Contract

Variables

manager	address	address of the person who is managing this campaign	
minimumContribution	uint	Minimum donation required to be considered a contributor or 'approver'	
approvers	address[]	List of addresses for every person who has donated money	!!!!
requests	Request[]	List of requests that the manager has created.	

Functions

Campaign	Constructor function that sets the minimumContribution and the owner	
contribute	Called when someone wants to donate money to the campaign and become an 'approver'	
createRequest	Called by the manager to create a new 'spending request'	
approveRequest	Called by each contributor to approve a spending request	!!!!
finalizeRequest	After a request has gotten enough approvals, the manager can call this to get money sent to the vendor	

- `Request struct -`

Request Struct

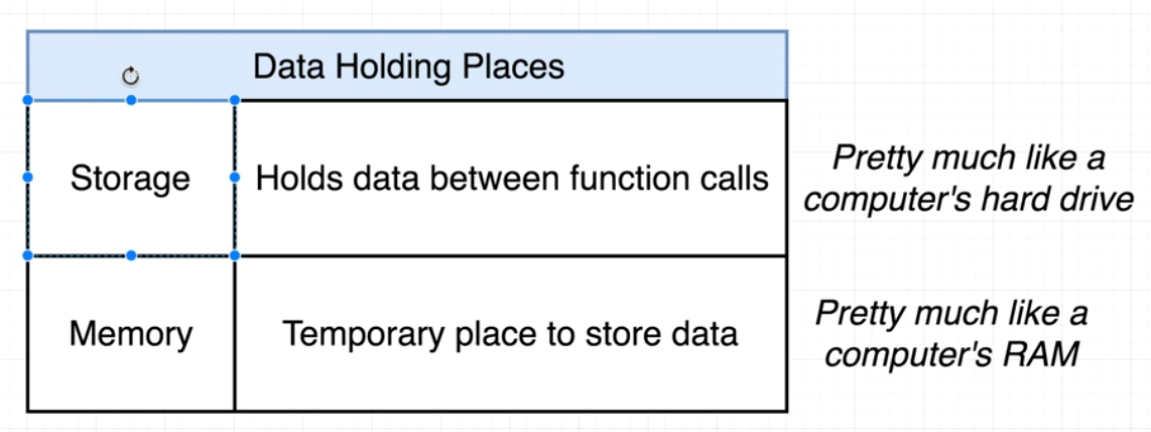
Name	Type	Purpose
description	string	Describes why the request is being created.
value	uint	Amount of money that the manager wants to send to the vendor
recipient	address	Address that the money will be sent to.
complete	bool	True if the request has already been processed (money sent)
???	???	Voting mechanism!

Types of vars in Solidity based on data storage -

- **Contract stores data of the vars acc. to its type -**

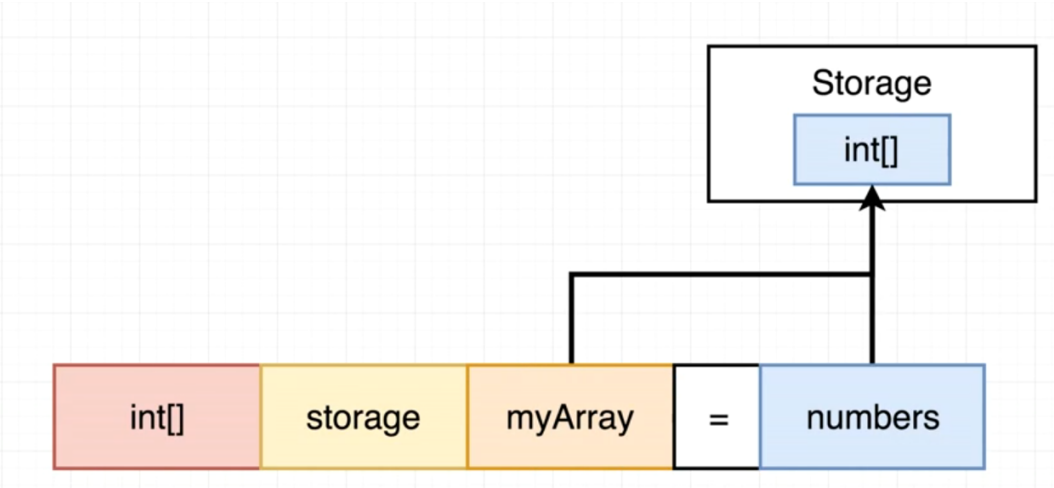
Variables are declared as either `storage`, `memory` or `calldata` to explicitly specify the location of the data.

- `storage` - variable is a state variable (store on blockchain)
- `memory` - variable is in memory and it exists while a function is being called
- `calldata` - special data location that contains function arguments

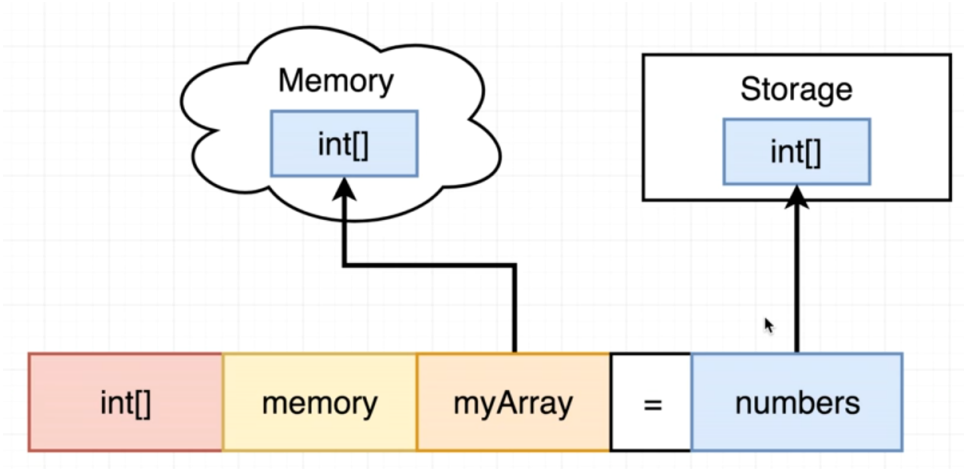


- **How Solidity vars reference values acc. to type of vars -**
  - Example 1
    - We have made myArray of storage type, so, it now points to the same memory location as numers array (storage var of the contract)
    - myArray[0] = 1 makes the 0th element of numbers array 1.

```
pragma solidity ^0.4.17;

contract Numbers {
    int[] public numbers;

    function Numbers() public {
        numbers.push(20);
        numbers.push(32);

        int[] storage myArray = numbers;
        myArray[0] = 1;
    }
}
```



- Example 2 -
    - Rather if we write int[] memory myArray = numbers, then a new copy of numbers array gets created and gets referenced to myArray var.



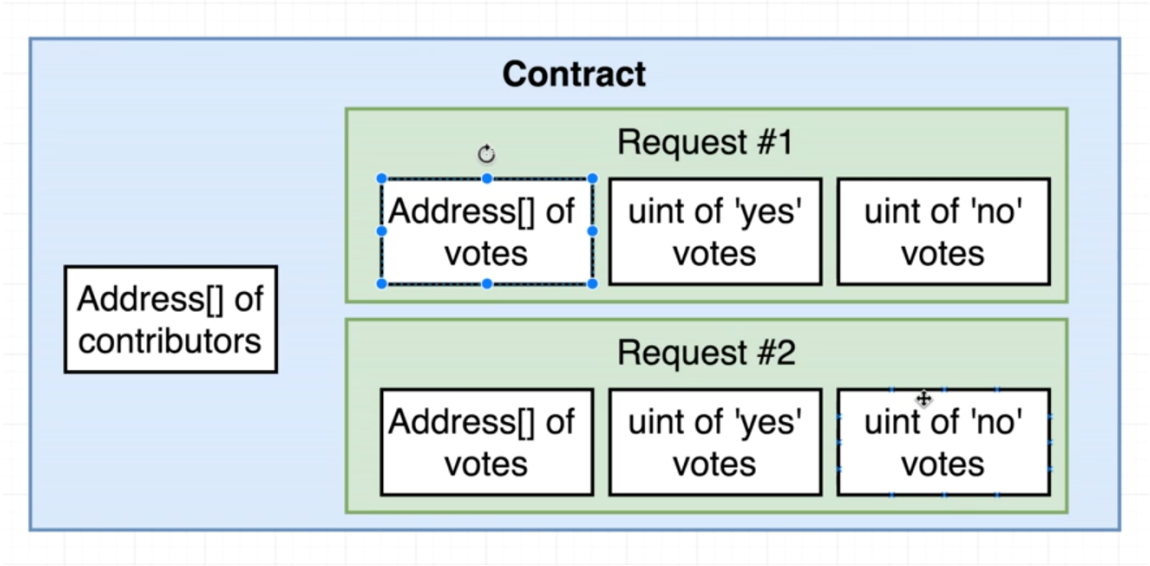
- Example 3 -
  - If we pass numbers array to a function call actually a COPY of it gets passed.
  - Hence, upon execution of the changeArray() function numbers array doesnt get changed.
  - IMP - **By default the function parameters are of memory type.**

```
pragma solidity ^0.4.17;

contract Numbers {
    int[] public numbers;

    function Numbers() public {
        numbers.push(20);
        numbers.push(32);

        changeArray(numbers);
    }

    function changeArray(int[] myArray) private {
        myArray[0] = 1;
    }
}
```

Voting System Requirements -

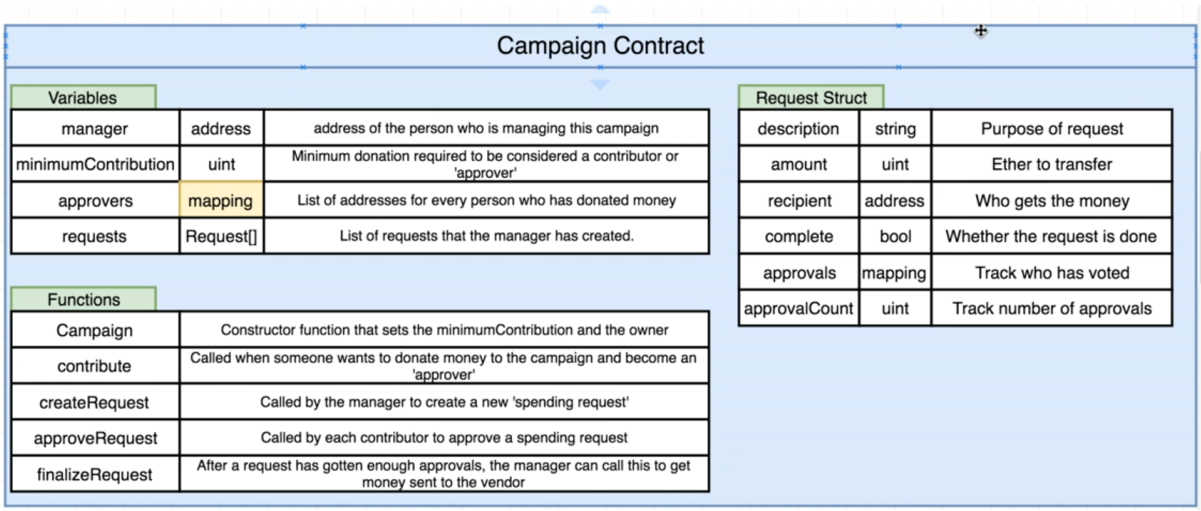
- One contributor should only be allowed a single vote on a given request.

- Should be able to handle large contributors for a given request.
- **Bad voting system -**



  - Extremely ineficient due to array DS.
  - **Solution → Maps.**
- **Good voting system -**
  - Instead of arrays use mapping.

Modified Contract -



- **A contract can deploy another contract.**

Campaign contract deploy approach -