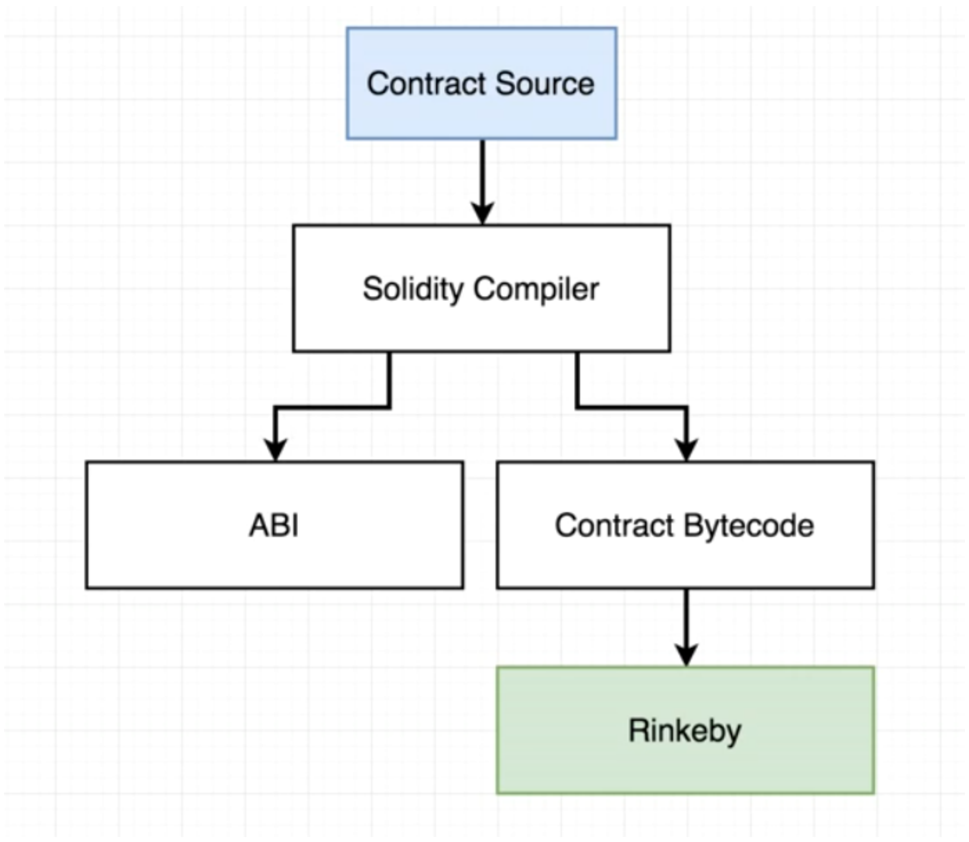
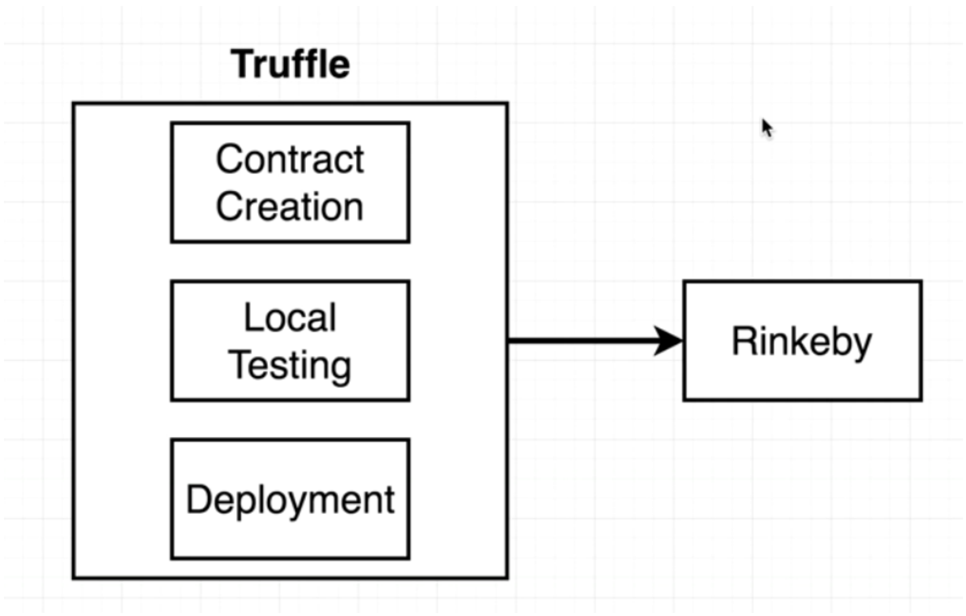
Smart Contracts with Solidity

Deploying our contract to a network -

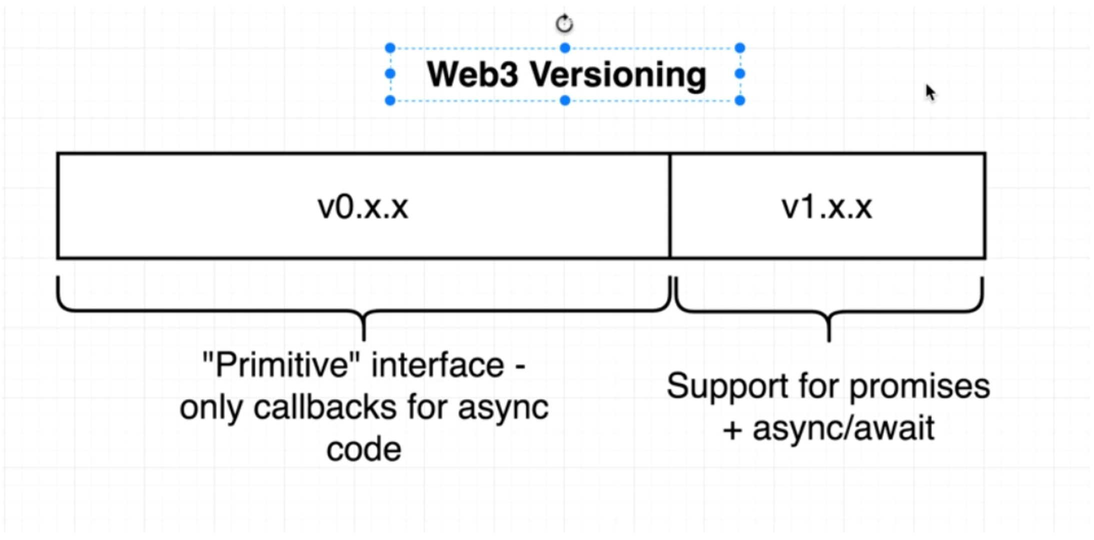


- Truffle is a one stop destination for developing Ethereum-based apps. Using it we can deploy our smart contract.

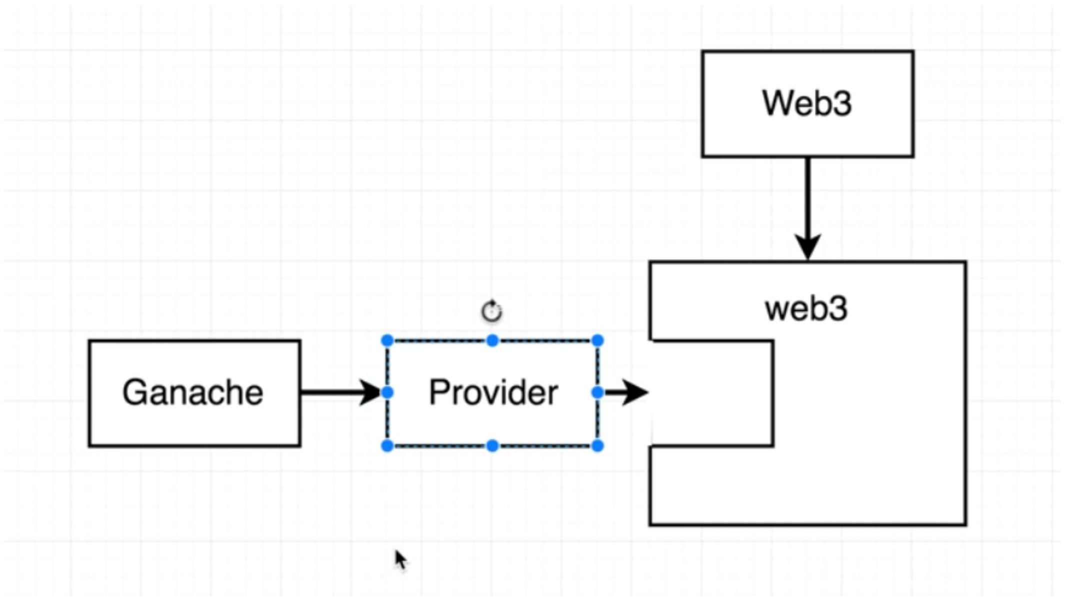


- Using solc npm module we can compile solidty code in js, doing this we will get the contract’s byte code and ABI.
    - ABI is an object that contains information about the methods existing in the contract, the input agrs (details), return type (details).

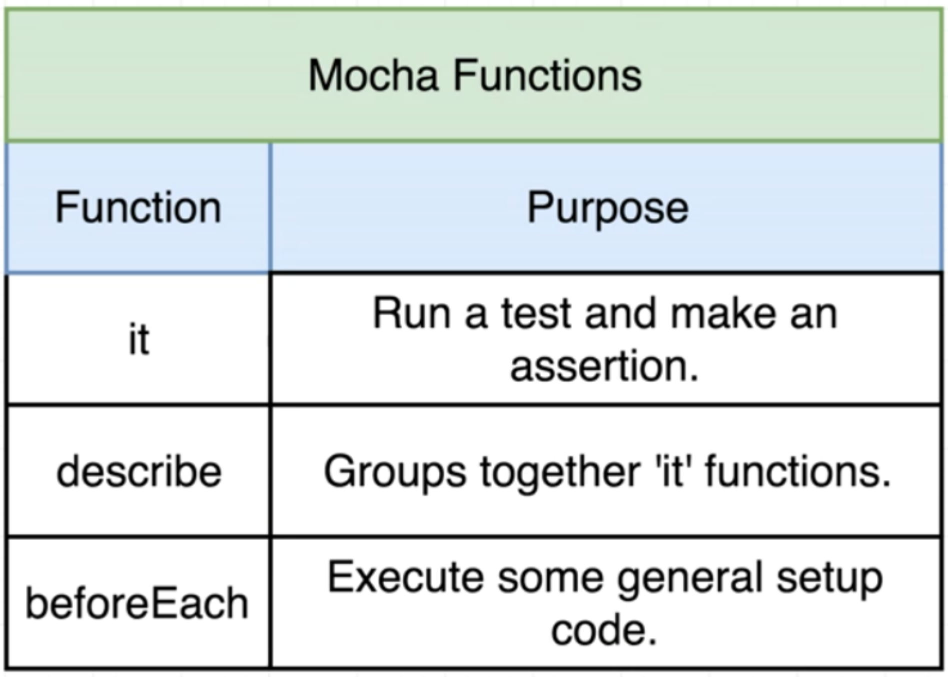
- Web3 npm package is the solution which people use to communicate between a JS app and the ethereum world.
    - This package serves as a portal for a JS dev into the ethereum world.
    - We can make multiple instances of the web3 lib using the imported constructor function, each instance will correspond to a connection of a network. (main net, kovan, etc)


Web3 Versioning
v0.x.x
v1.x.x
"Primitive" interface - only callbacks for async code
Support for promises + async/await

- Web3 instance once created requires some configuration, basically it needs a provider from a specific eth network.
  - Provider is a communication layer between the eth network and the web3 lib instance.

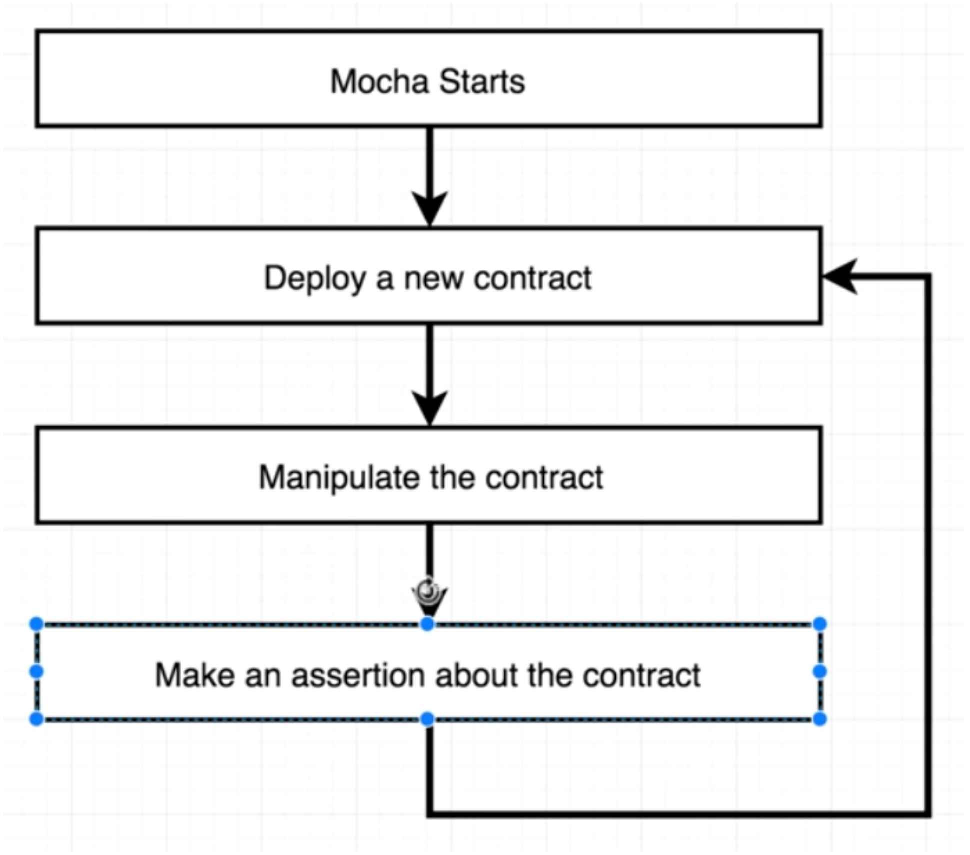


- **Gist of Mocha -**

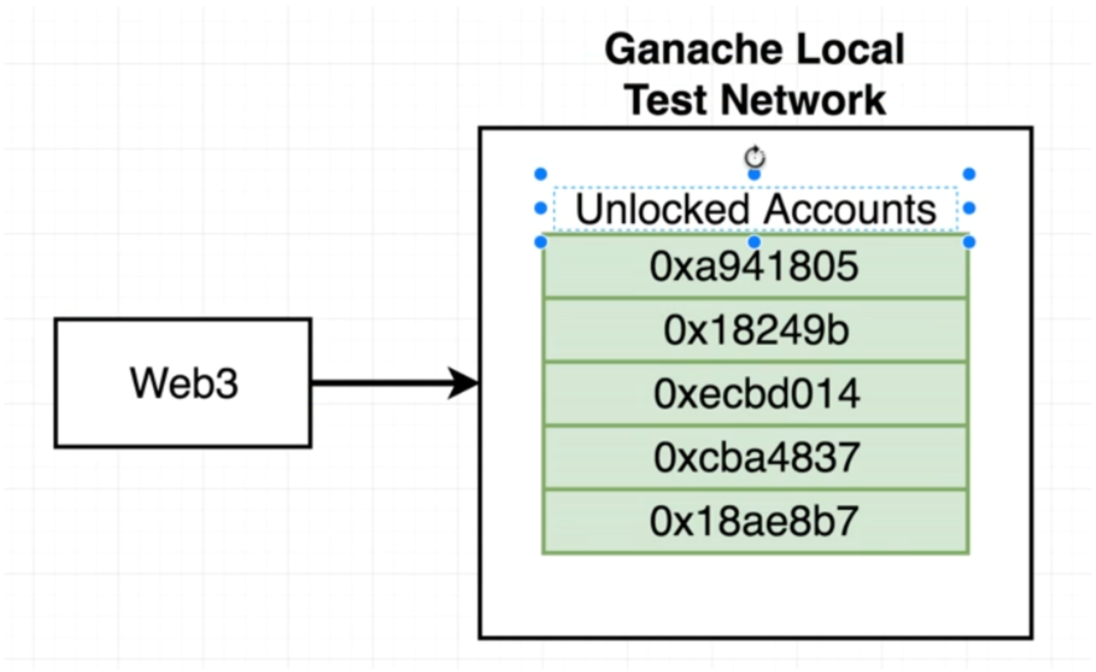


Mocha Functions	
Function	Purpose
it	Run a test and make an assertion.
describe	Groups together 'it' functions.
beforeEach	Execute some general setup code.

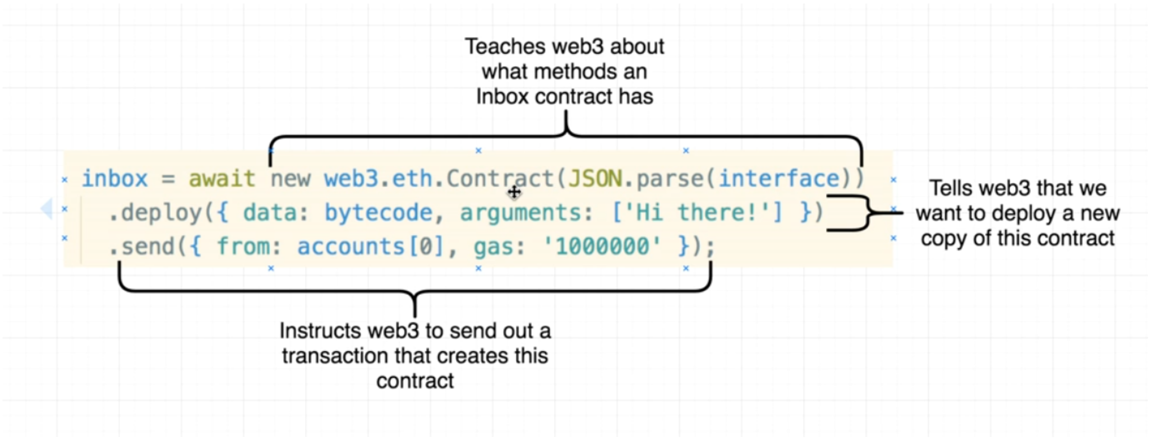
- **`Testing our smart contract with Mocha -`**



- We will use the web3 lib instance (which is connected to Ganache network) to retrieve the unlocked accounts. Using one of these accounts we will deploy our contract to the network, this will happen before every it() call.



- Deploying a contract -



```
inbox = await new web3.eth.Contract(JSON.parse(interface))
  .deploy({ data: bytecode, arguments: ['Hi there!'] })
  .send({ from: accounts[0], gas: '1000000' });
```

Teaches web3 about what methods an Inbox contract has
Tells web3 that we want to deploy a new copy of this contract
Instructs web3 to send out a transaction that creates this contract

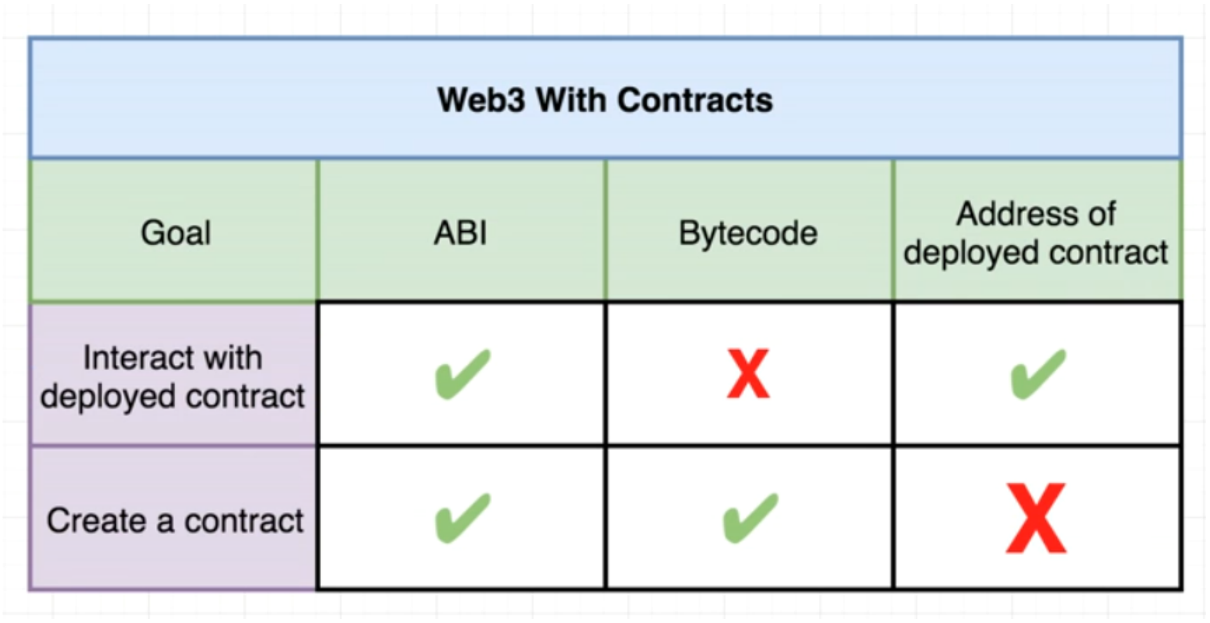
- The **Contract** constructor -
  - This contructor is used to either **access already deployed contracts on the blockchain**, or, **used to create & deploy new contracts.** The contructor returns

a contract object, which basically has the given interface.

    - 1st argument to Contract constructor is the ABI.

  - The **deploy** method -
    - Creates a transaction object with the data as the bytecode and the list of arguments that the contract constructor would expect.
  - The **send** method -
    - This method sends out the transaction to the blockchain network.
- Finally, **inbox.options.address** holds the address of the newly deployed contract (corresponding account ka address).

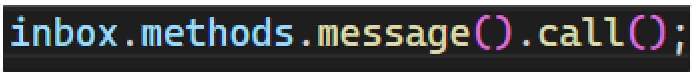
- **Uses of the Web3 lib -**
  - The web3 lib can be used to do 2 things interact with deployed contracts and create a contract.
  - For doing these 2 things we need the following inputs -



Web3 With Contracts			
Goal	ABI	Bytecode	Address of deployed contract
Interact with deployed contract	✓	X	✓
Create a contract	✓	✓	X

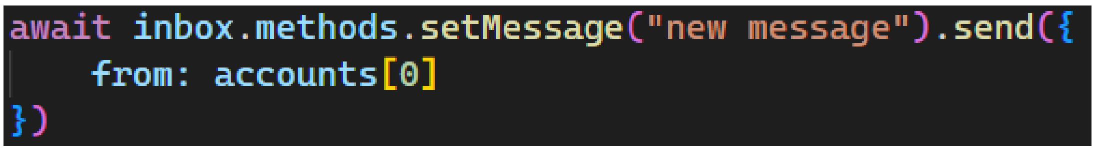
- **Calling methods of our contract from JS code -**

- **Important** - Nearly all the methods in web3 which we will call are asynchronous.
- Following is how we call a function which DOESNT involve a transaction -
  - The call is async.
  - message is the name of the function which we are calling. **Remember we have to call message().**
    - Also, 1st set of parenthesis mai wo arguments jate he jo us method call mai required hote he, meaning wo args which message() call needs, none here.
  - call() actually calls the method, sends it to the blockchain network. In this 2nd set of parenthesis we customize the transaction which we are sending to the network.
    - We may add stuff like who is issuing the transaction and how much gas to use.
    - Here, its a simple function call which doesnt require a transaction, so, no args to call() is required.



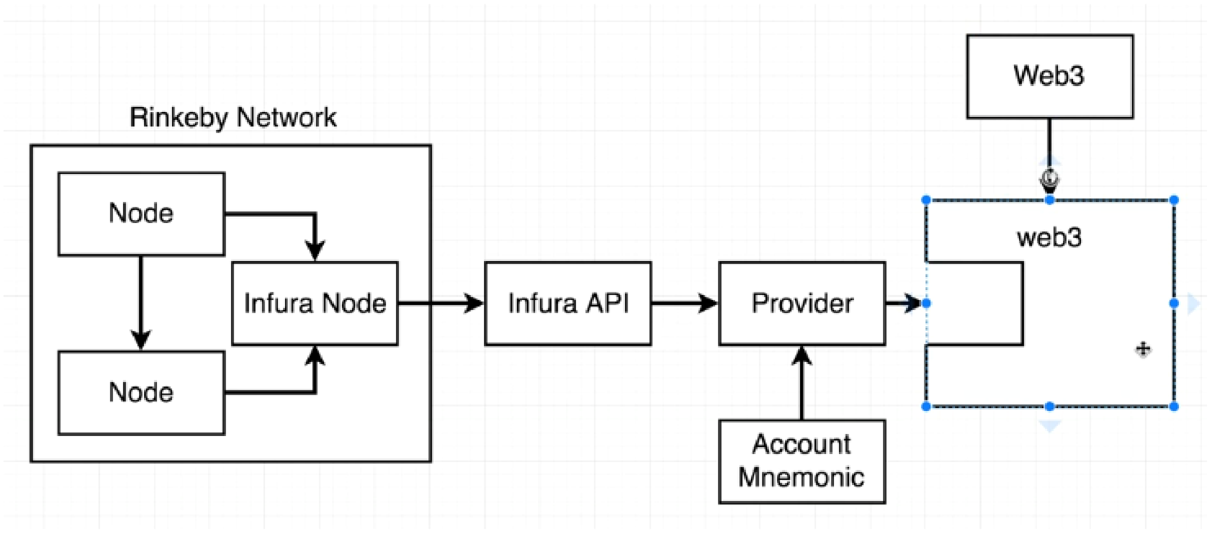
```
inbox.methods.message().call();
```

- Following is how we call a function which DOES involve a transaction -
  - We call the method with the required arguments, in this case thats setMessage(”___”).
  - Then we call **send()** which sends the transaction to the network.



```
await inbox.methods.setMessage("new message").send({
    from: accounts[0]
})
```

- **Deploying to a public test network -**
  - Yaha mujhe account specify karna padega, which will be used to deploy our contract, and contract deploy karne ke liye uss account mai ether bhi chahiye hoga.
  - To deploy we will need access to a node on the target network, we can do this in 2 ways -
    1. Create a local eth node and connect it to target network.
    2. Use Infura which gives us access to a node deployed on our target network (Rinkeby Network).
       a. Infura is a public API.
       b. We can think of it as our portal beyond web3 lib into the **Rinkeby Network**. Baki networks pe bhi node he.
  - Also, here, we will have to setup the provider manually, we will use truffle hd wallet provider for this.


Rinkeby Network
Node
Infura Node
Node
Infura API
Provider
Account Mnemonic
Web3
web3