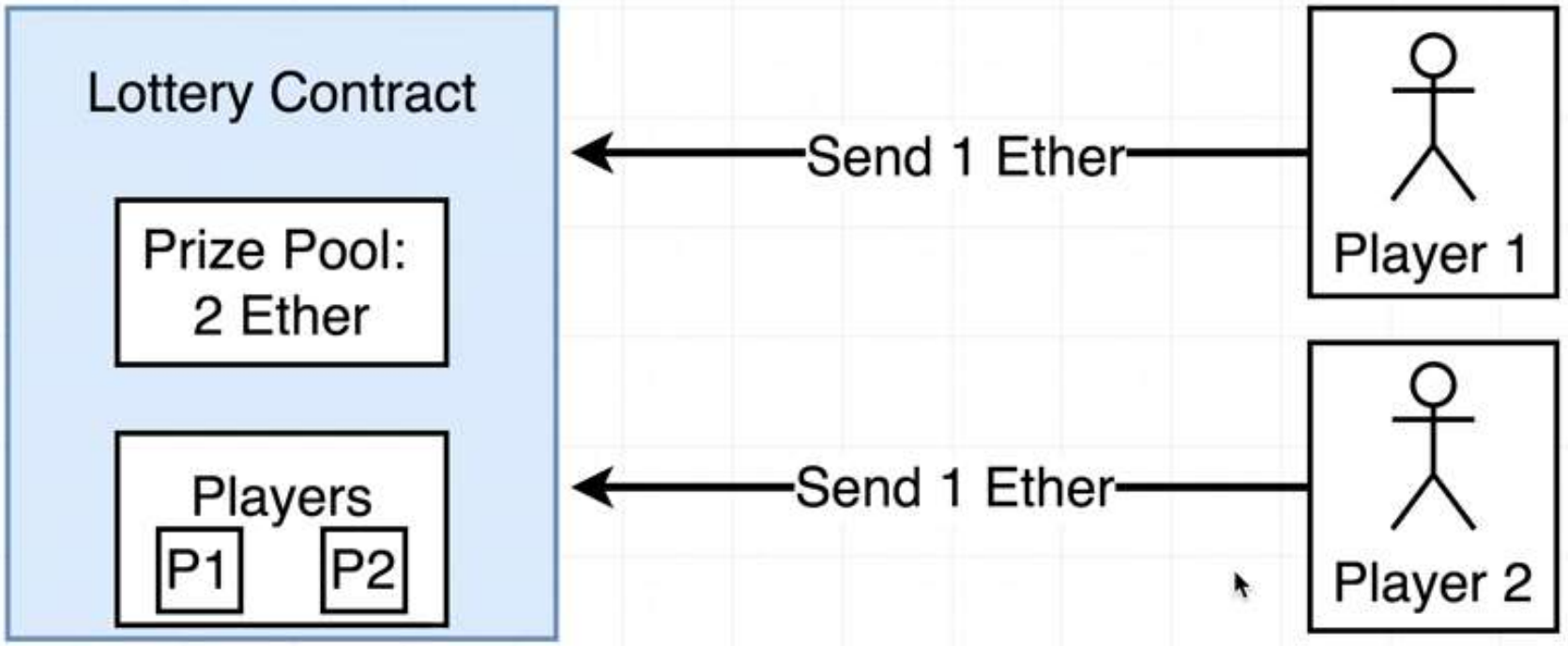
ethereum

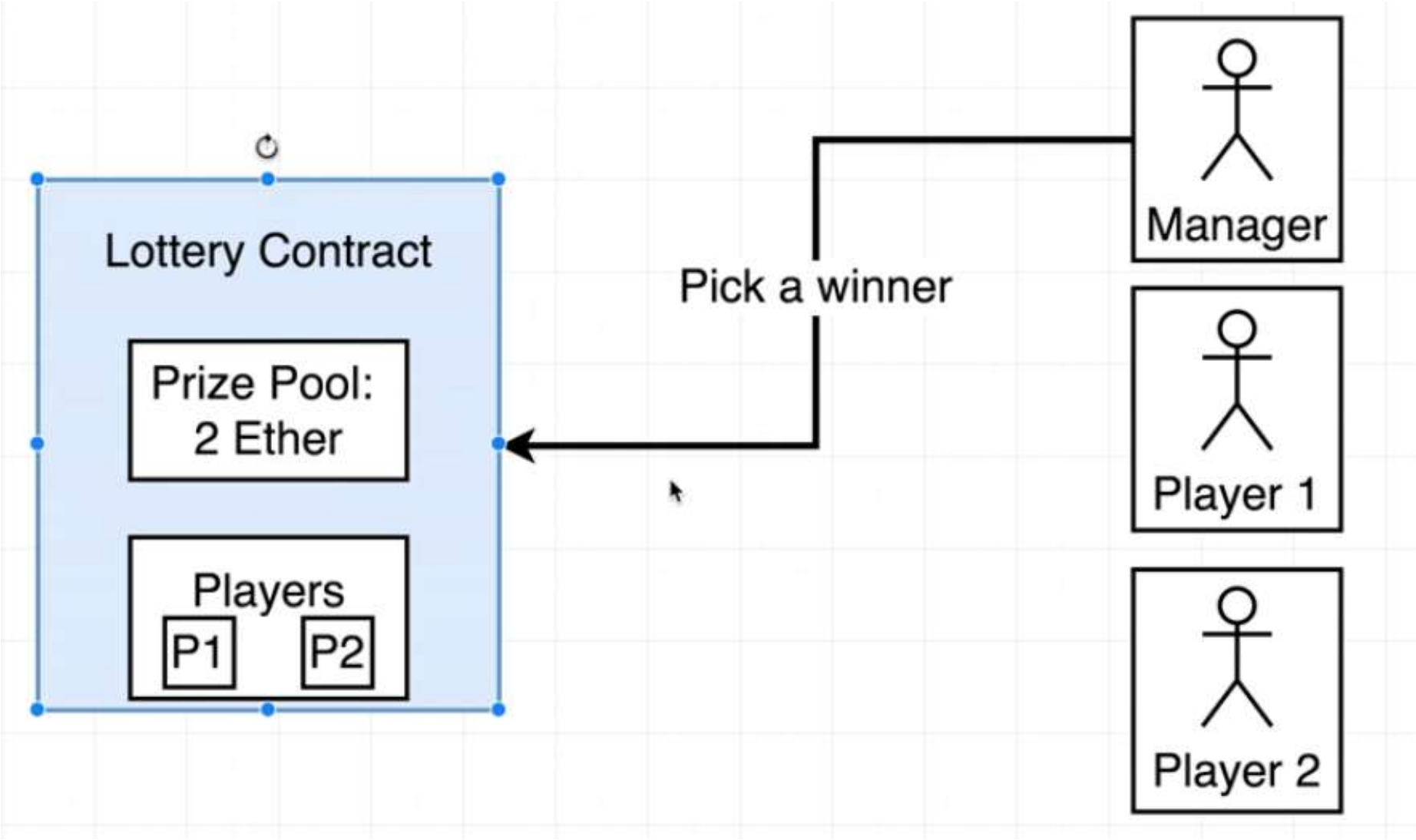
Advanced Contract

More advanced contract

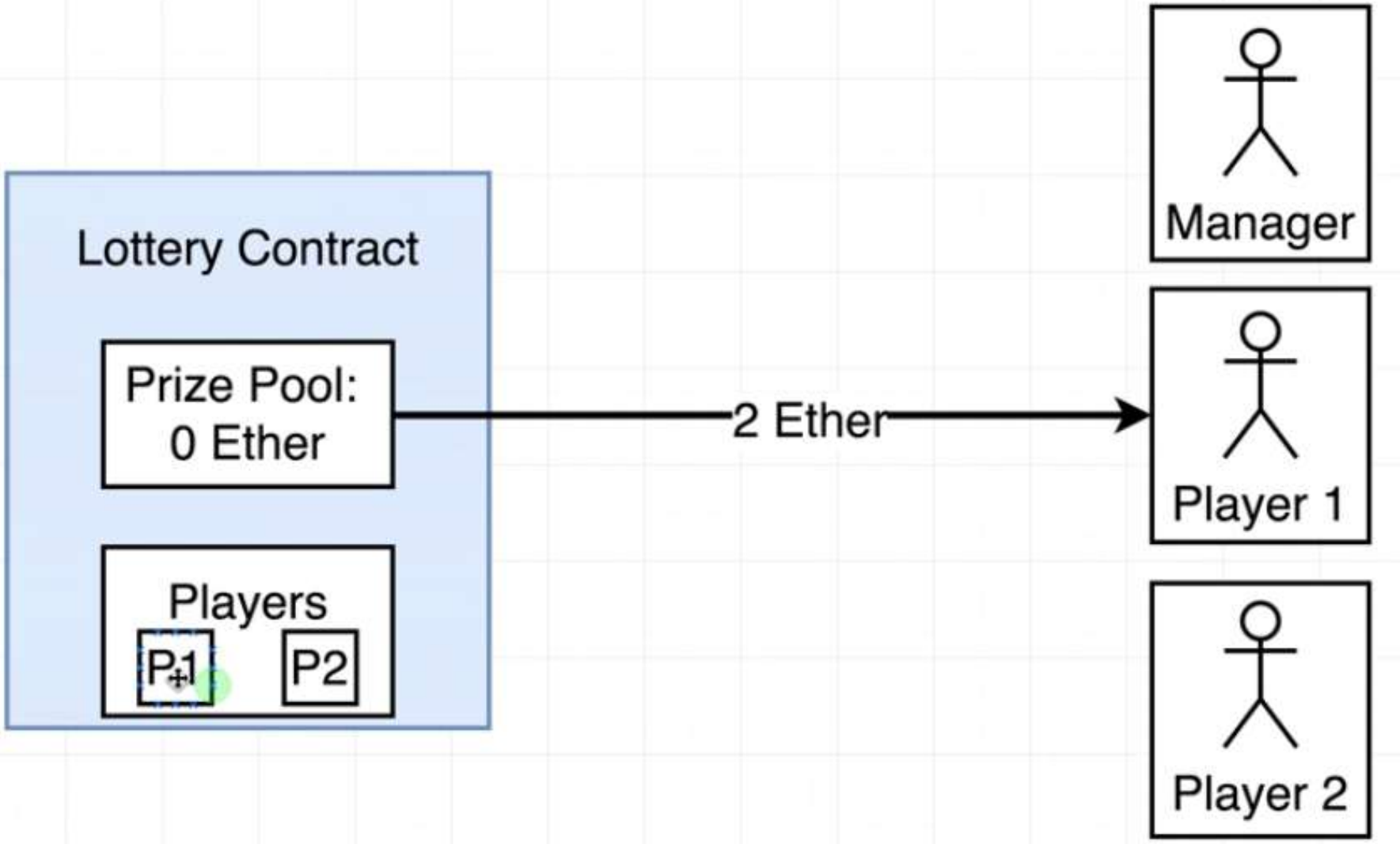


- Self repeating contract
- After sending Ethers, Player 1 & Player 2 becomes a part of game
- Ethers goes to prize pool

Contract Scenario



Contract Scenario

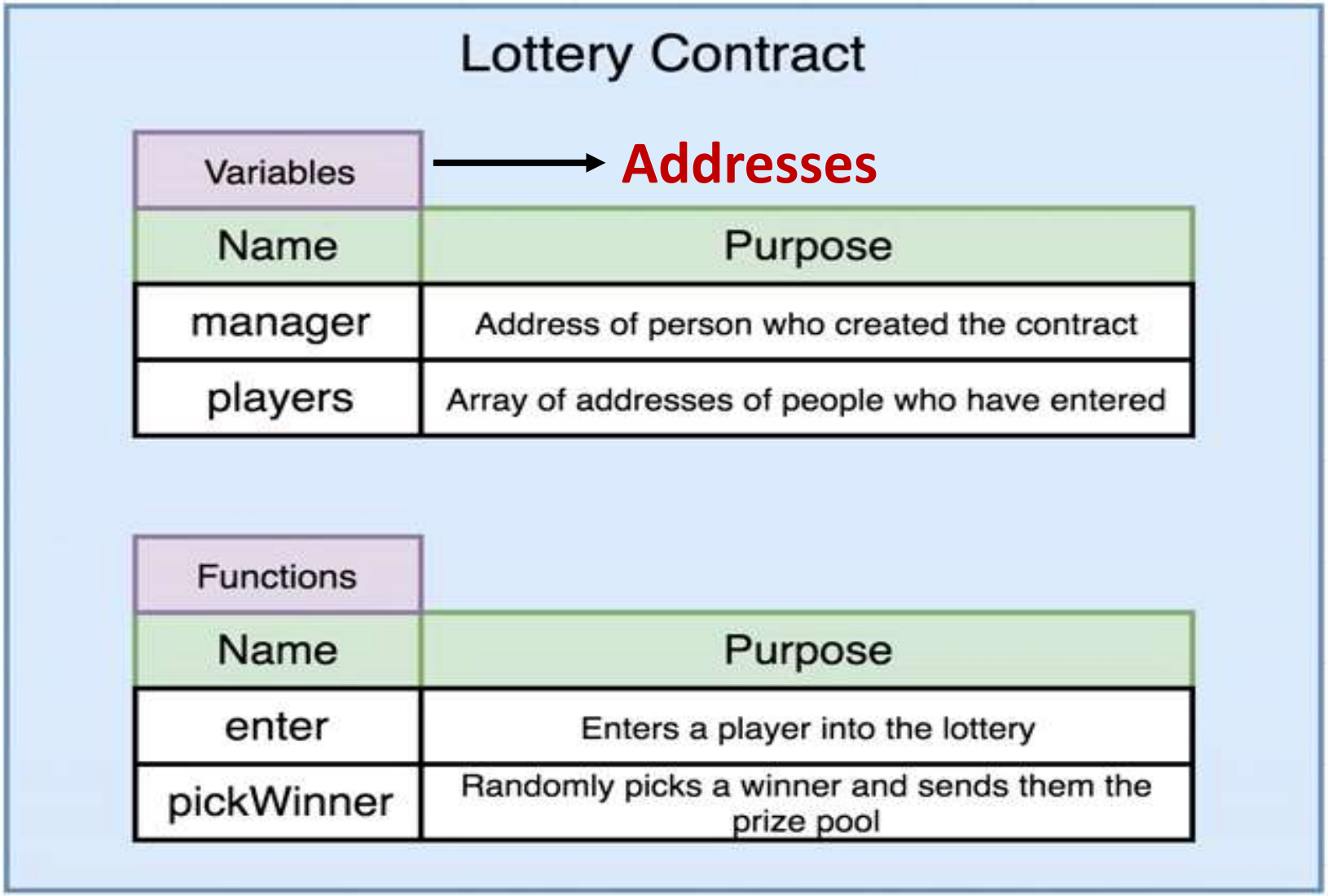


- **Declare winner**
- **Repeat same procedure for same/different players**

Expected Learning

- Basic data structures
- Enforce some level of security

Lottery Contract



```
pragma solidity ^0.4.17;

contract Lottery {
    address public manager;

    function Lottery() public {

    }
}
```

Int and uint

For uint the lower bound is always zero.

Integer Ranges		
Name	Lower Bound	Upper Bound
int8	-128	127
int16	-32,768	32,767
int32	-2,147,483,648	2,147,483,647
...	...	...
int256	Really, really negative	Really, really big

int == int256

- **Pay for storage of values**
- **Larger the data type → Greater amount to pay (Gas)**

Basic value types in solidity

Basic Types		
Name	Notes	Examples
string	Sequence of characters	"Hi there!" "Chocolate"
bool	Boolean value	true false
int	Integer, positive or negative. Has no decimal	0 -30000 59158
uint	'Unsigned' integer, positive number. Has no decimal	0 30000 999910
fixed/ufixed	'Fixed' point number. Number with a decimal after it	20.001 -42.4242 3.14
address	Has methods tied to it for sending money	0x18bae199c8dbae199c8d

Contract Requirements

- Keep contract as simple as possible
- Avoid complex mathematics from contract
- Deployer need to pay for each computation

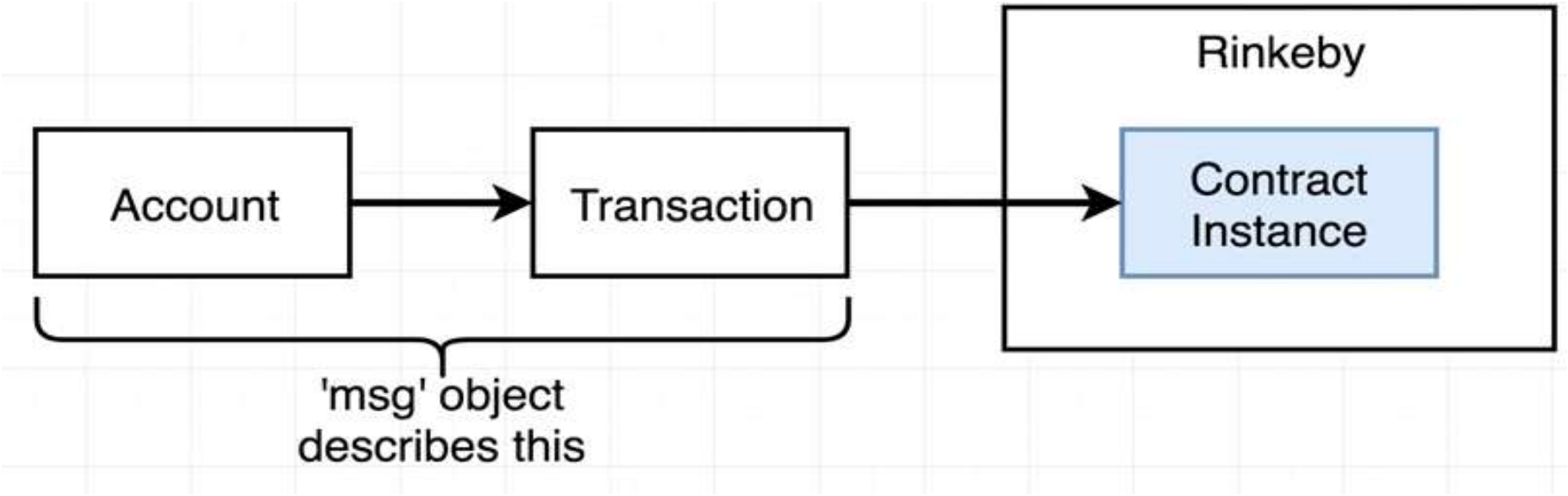
```
pragma solidity ^0.4.17;

contract Lottery {
    address public manager;

    function Lottery() public {

    }
}
```

Msg global variable



- **Msg is a magical global variable**
- **Present in all the functions of the contract**
- **Can be accessed anywhere without any declaration**
- **Available for both function calling and sending transaction**

The 'msg' Global Variable	
Property Name	Property Name
msg.data	'Data' field from the call or transaction that invoked the current function
msg.gas	Amount of gas the current function invocation has available
msg.sender	Address of the account that started the current function invocation
msg.value	Amount of ether (in wei) that was sent along with the function invocation

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract Lottery {
    address public manager;

    constructor() {
        manager = msg.sender;
    }
}
```

Basic reference types

Reference Types		
Name	**Notes**	**Examples**
fixed array	Array that contains a *single type* of element. Has an unchanging length	`int[3] --> [1, 2, 3]` `bool[2] --> [true, false]`
dynamic array	Array that contains a *single type* of element. Can change in size over time	`int[] --> [1,2,3]` `bool[] --> [true, false]`
mapping	Collection of key value pairs. Think of Javascript objects, Ruby hashes, or Python dictionary. All keys must be of the same type, and all values must be of the same type	`mapping(string => string)` `mapping(int => bool)`

struct	Collection of key value pairs that can have different types.	`struct Car {`
		`  string make;`
		`  string model;`
		`  uint value;`
		`}`

Basic reference types

- In lottery contract:
  - Case 1: Number of players are dynamic at any point of time
    - Solution: Fixed Array
  - Case 2: Number of players are fixed at any point of time
    - Solution: Dynamic Array

How arrays work in solidity

```
pragma solidity ^0.8.13;

contract Array_Test {
    uint[] public arr;

    constructor() {
        arr.push(33);
        arr.push(54);
        arr.push(98);
    }

    function getFirstElement() public view returns (uint) {
        return arr[0];
    }

    function getArrLength() public view returns (uint) {
        return arr.length;
    }
}
```

How arrays work in solidity

```
  function getArr() public view returns (uint[] memory) {

        return arr;
 }

function push(uint i) public {
        // Append to array
        // This will increase the array length by 1.

       arr.push(i);
 }
```

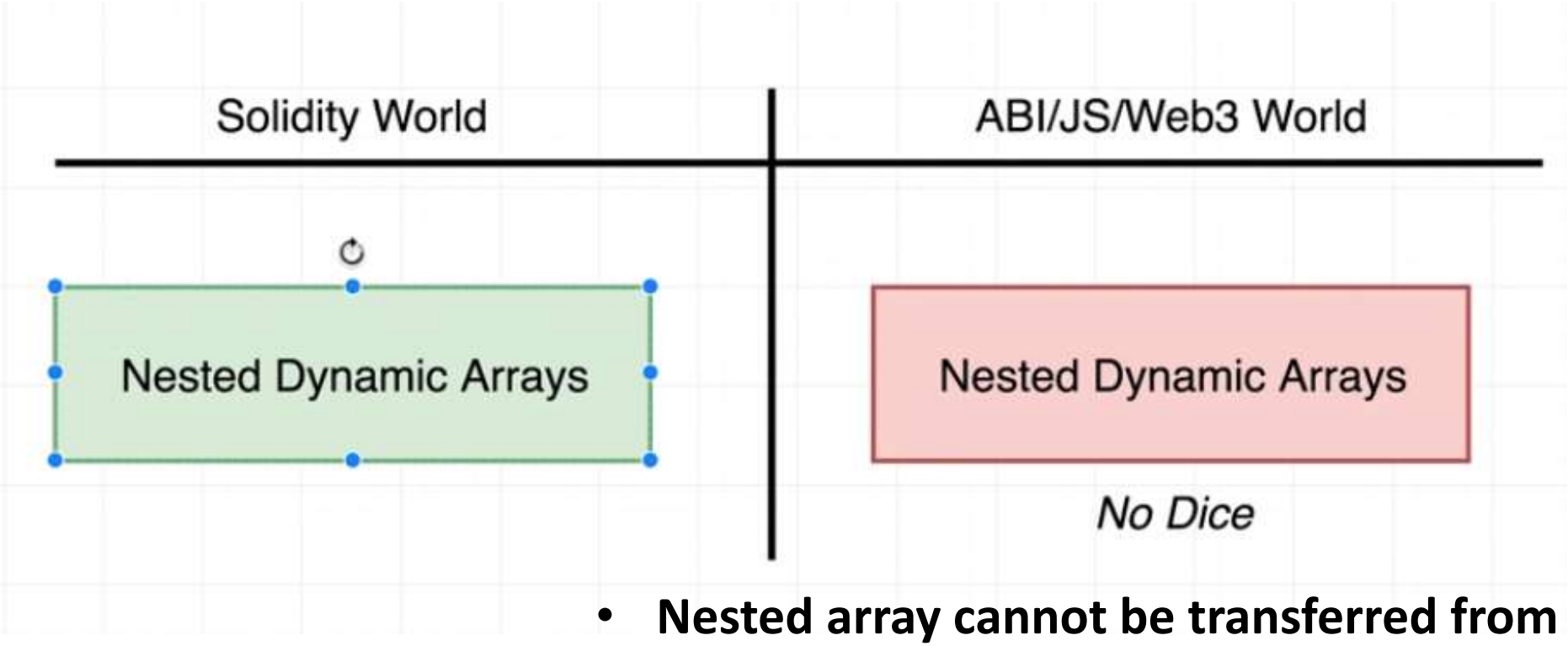
```
function pop() public {
 // Remove last element from array
 // This will decrease the array length by 1

        arr.pop();
 }

function remove(uint index) public {
 // Delete does not change the array length.
 // It resets the value at index to it's default
 // value, in this case 0

        delete arr[index];
}
```

Remark: Nested Array




```
const nestArr = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
]
```

- Nested array cannot be transferred from solidity to JS
- Cannot transfer array of strings to JS (e.g. weekdays, months etc.
- Limitation resolved in version 0.5.* series

Simple Contract body

```
pragma solidity ^0.8.13;

contract Lottery {

    constructor () {

    }
}
```

Include Players and Managers

```
pragma solidity ^0.8.13;

contract Lottery {
    address public manager;
    address [] public players;

    constructor () {
        manager = msg.sender;
    }
}
```

Create **enter** function

```
pragma solidity ^0.8.13;

contract Lottery {
    address public manager;
     address [] public players;

    constructor () {
        manager = msg.sender;
    }

function enter() public {
          players.push (msg.sender);
    }
}
```

No lottery Win because senders haven't sent any money

Send money & make sender payable

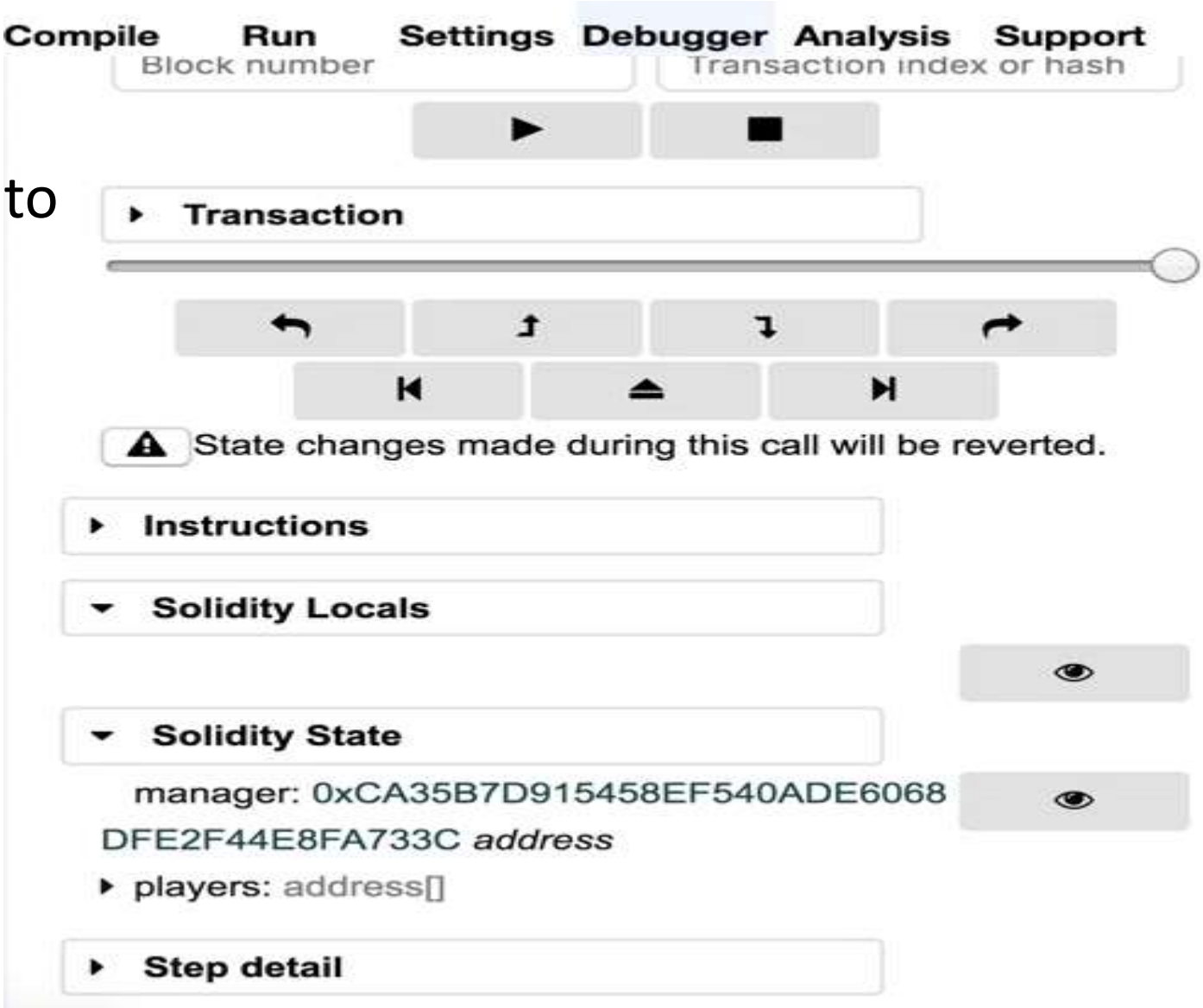
Sender should have certain amount to enter

```
function enter() public
payable
{
    require(msg.value > .01 ether);
    players.push(msg.sender);
}
```

- Require keyword does the work of **if..else**
- **If condition true → Proceed**
- **Else → stop execution**

Remix debugger

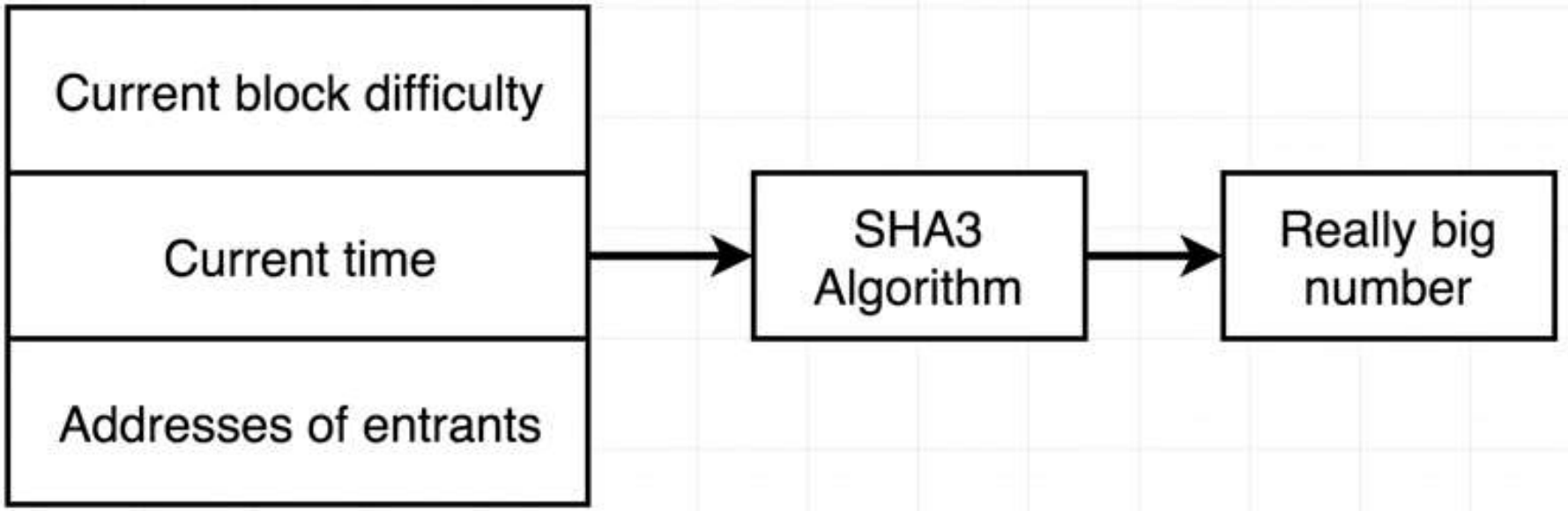
If sender tries to enter without sufficient ethers



Random Function

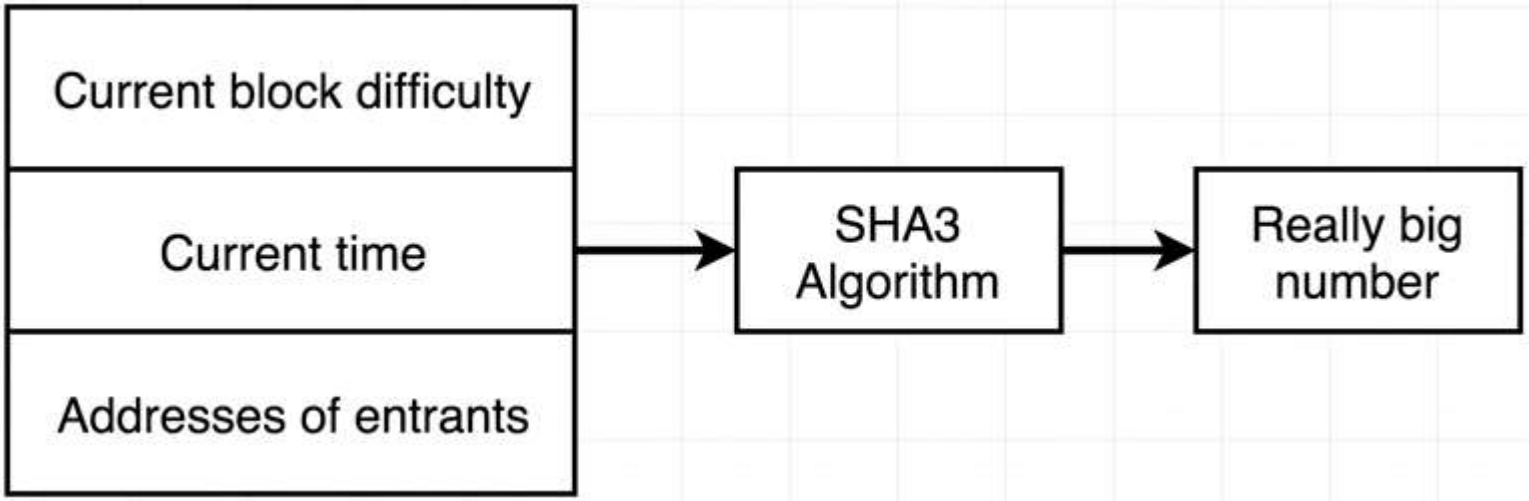
- No random generator in solidity
- Pseudo random number generator (mathematical algorithms)
- **Difficulty level:** ensures that blocks of transactions are added to the blockchain at regular intervals, even as more miners join the network.
- If the difficulty **remained the same**, it would **take less time between adding new blocks** to the blockchain as new miners join the network.

Picking a winner



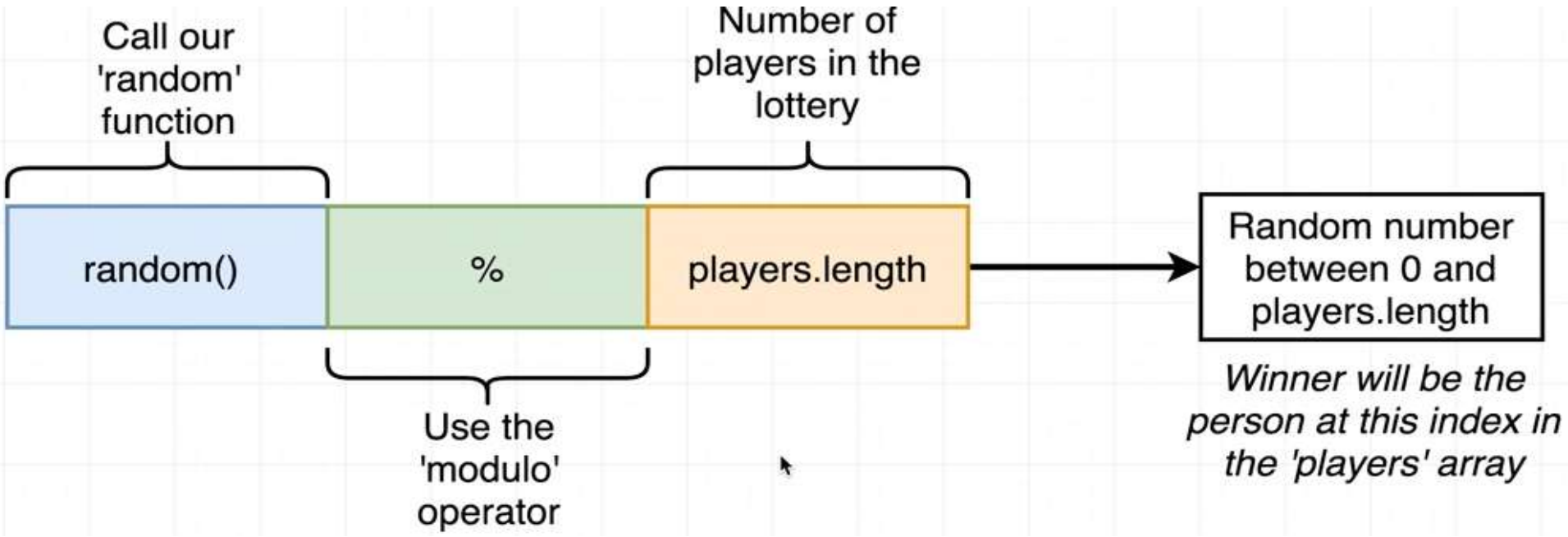
Difficulty level explanation

Picking a winner (Random Function)




```
function random() private view returns (uint) {
    return uint(keccak256(abi.encodePacked(
block.difficulty, block.timestamp, players.length)
));
}
```

Picking a winner




```
function pickWinner() public {
    uint8 index = uint8(random() % players.length);
}
```

Picking a winner

1

2

Current block difficulty

Current time

Addresses of entrants

```
function random() public view returns (uint) {
    return uint(keccak256(block.difficulty, now, players));
}
```

SHA3 Algorithm

Really big number

4

```
function pickWinner() public {
    uint index = random() % players.length;
}
```

Call our 'random' function

Number of players in the lottery

random()

%

players.length

Random number between 0 and players.length

Use the 'modulo' operator

3

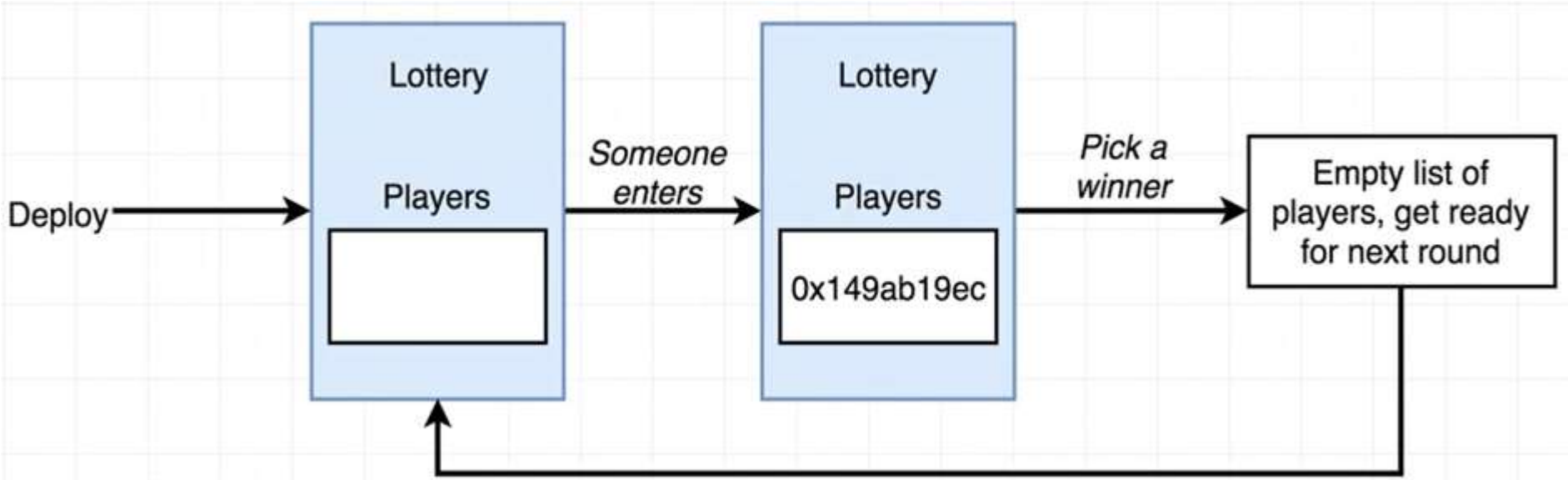
Winner will be the person at this index in the 'players' array

Sending money to winner and resetting the contract

```
function pickWinner() public {
    uint8 index = uint8(random() % players.length);

    players[index].transfer(address(this).balance);
    players = new address payable[](0);

//Resetting player list to 0 (Dynamic infinite loop)
}
```



Errors

8. Error

```
address payable[] public players;
```

9. Error

```
players.push(payable(msg.sender));
```

10. Error

```
players = new address payable [](0);
```

Only manager can call pick winner

```
function pickWinner() public {
    require(msg.sender == manager);

    uint8 index = uint8(random() % players.length);
    players[index].transfer(address(this).balance);
    players = new address payable[](0);
}
```

Select address of manager for picking the winner

Need of function Modifier

- **For code reusability**

```
function returnEntries() {
    require(msg.sender == manager);
}
```

- **Modifiers: uses to prevent Writing repeated lines of code.**
- **If want to write any other function that can be run only by the manager then →**
  - **simply use restricted modifier.**

Need of function Modifier AND Only manager can call pick winner

OR

```
function pickWinner() public
restricted
{
    uint8 index = uint8(random() % players.length);
    players[index].transfer(address(this).balance);
    players = new address payable[](0);
}


modifier restricted() {
        require(msg.sender == manager);
        _;
}
```

List of All Players

```
function getPlayers() public view returns (address[]) {
    return players;
}
```

```
function getPlayers() public view returns
                        (address payable [] memory)
{

      return players;

}
```

Next Class…..

Deployment & Test this contract on real test network

Simple mocha tests for lottery

```
it('requires a minimum amount of ether to enter', async () => {
  try {
    await lottery.methods.enter().send({
      from: accounts[0],
      value: 0
    });
    assert(false);
  } catch (err) {
    assert(err);
  }
});
```

```
it('only manager can call pickWinner', async () => {
  try {
    await lottery.methods.pickWinner().send({
      from: accounts[1]
    });
    assert(false);
  } catch (err) {
    assert(err);
  }
});
```