Smart Gambling

Ambreen Simon, Sebastien Vezina, Stephen Chen,
Wazarat Hussain, & Val De Franco

Background

Growing interest in adopting blockchain for online gambling:

- Privacy protection
- Data security
- Ease of virtual transactions
- Circumvent local regulations and restrictions

Aim

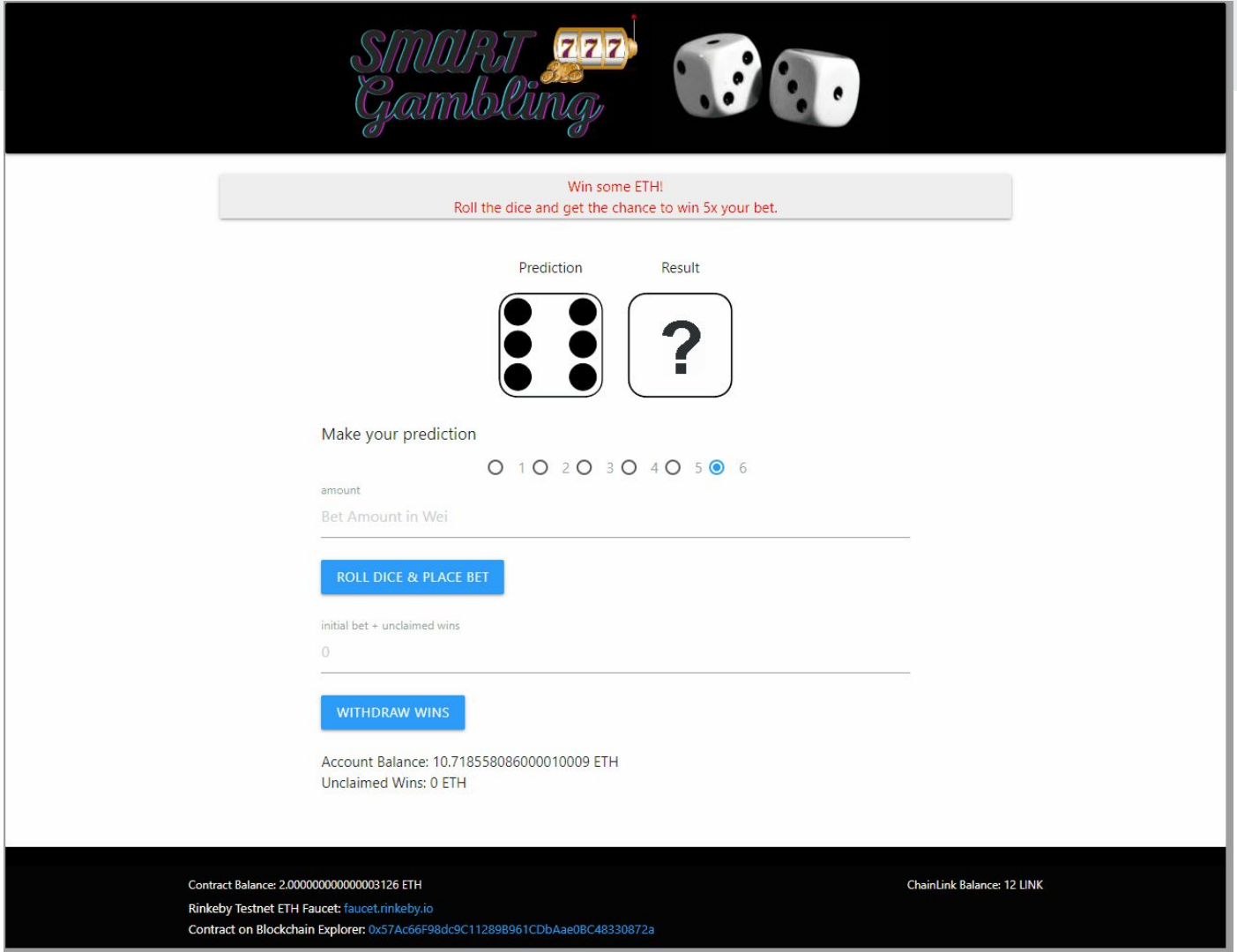
To create a simple online game utilizing:

- Smart contract
- Crypto-currency such as Ether for betting
- GUI (frontend)

Via GUI, a player will bet their wei and roll a virtual die.

- Winners receive 5x the bet + refund of the original bet.

DApp


SMART Gambling
777
Win some ETH!
Roll the dice and get the chance to win 5x your bet.
Prediction
Result
?
Make your prediction
1 2 3 4 5 6
amount
Bet Amount in Wei
ROLL DICE & PLACE BET
initial bet + unclaimed wins
0
WITHDRAW WINS
Account Balance: 10.718558086000010009 ETH
Unclaimed Wins: 0 ETH
Contract Balance: 2.000000000000003126 ETH
ChainLink Balance: 12 LINK
Rinkeby Testnet ETH Faucet: faucet.rinkeby.io
Contract on Blockchain Explorer: 0x57Ac66F98dc9C11289B961CDbAae0BC48330872a

Resources

- Solidity, Remix
- OpenZeppelin, SafeMath
- ChainLink
- Web3.js - Ethereum JavaScript API
- MetaMask, Rinkeby Ethereum Testnet
- HTML, JavaScript

Points of Fascination

Challenge:

Random Numbers

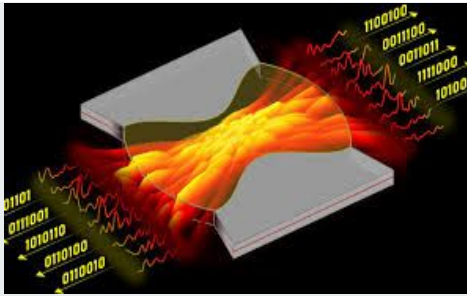
Solution:

Oracle = Chainlink + Verifiable Random Function

What is an Oracle?

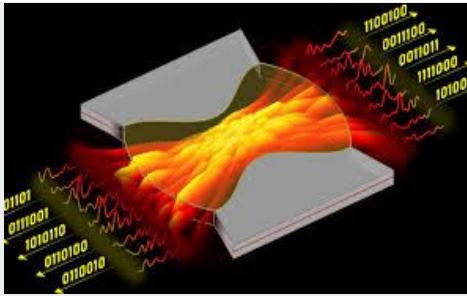
- Sends data from the outside world to the blockchain
- Smart contracts aren't designed to take care of everything
- Just like normal businesses, sometimes rely on suppliers or vendors, in this case, an Oracle such as Chainlink.

Random Numbers



- Random ≠ Random
  - Algorithms use a seed such as time or mouse movements.
  - Can be solved.
  - Usually random enough for most applications.

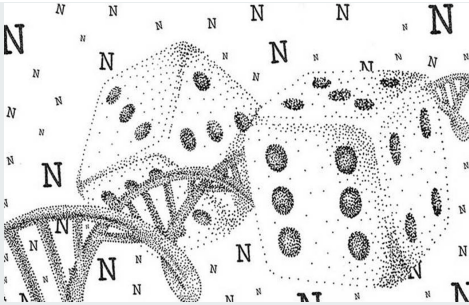
Random Numbers



Two other issues:

1. Potential conflict of interest with miners.
2. Decentralized vs. centralized - single point of failure.

Random Numbers



Options:

1. Use Blockhash or Block Timestamp.
2. Use Centralized API or Oracle.
3. Chainlink Decentralized Oracle - Verifiable Random Function (VRF).

Random Numbers

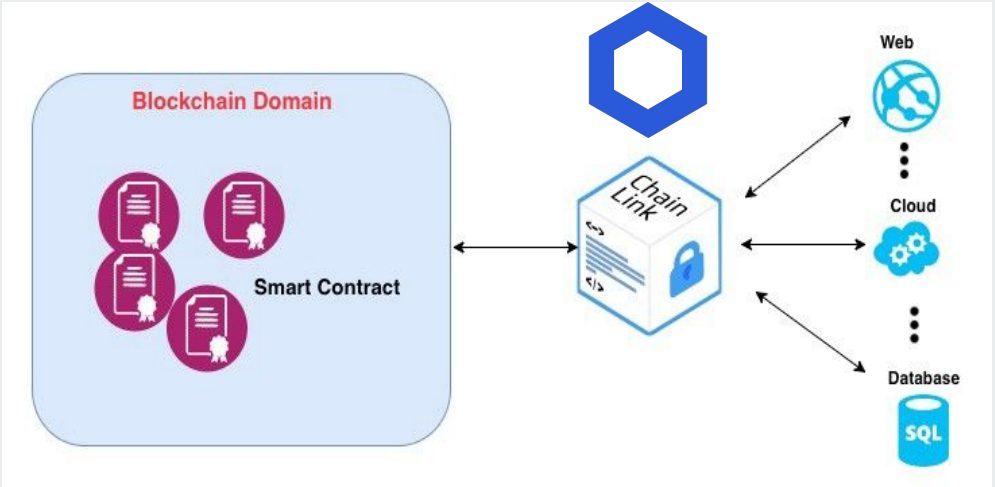
ALTERNATIVES	CRITERIA		
	Decentralized	Really Random	Miner Conflict of Interest
BlockHash	✓	✓	X
Centralized Oracle	X	?	✓
Chainlink / VRF	✓	✓	✓

Random Numbers

Solution: Chainlink/VRF.

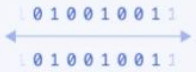
- As a network, multiple nodes operating.
- Removes single point of failure.
- VRF is usable for our project.
- Specifically, we use Chainlink.

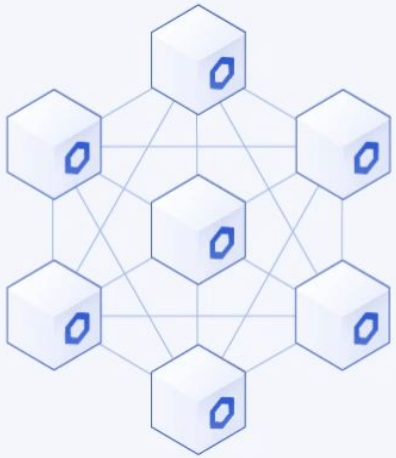
Chainlink (LINK)



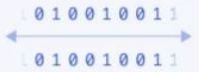
Chainlink (LINK)

VRF





Smart contracts



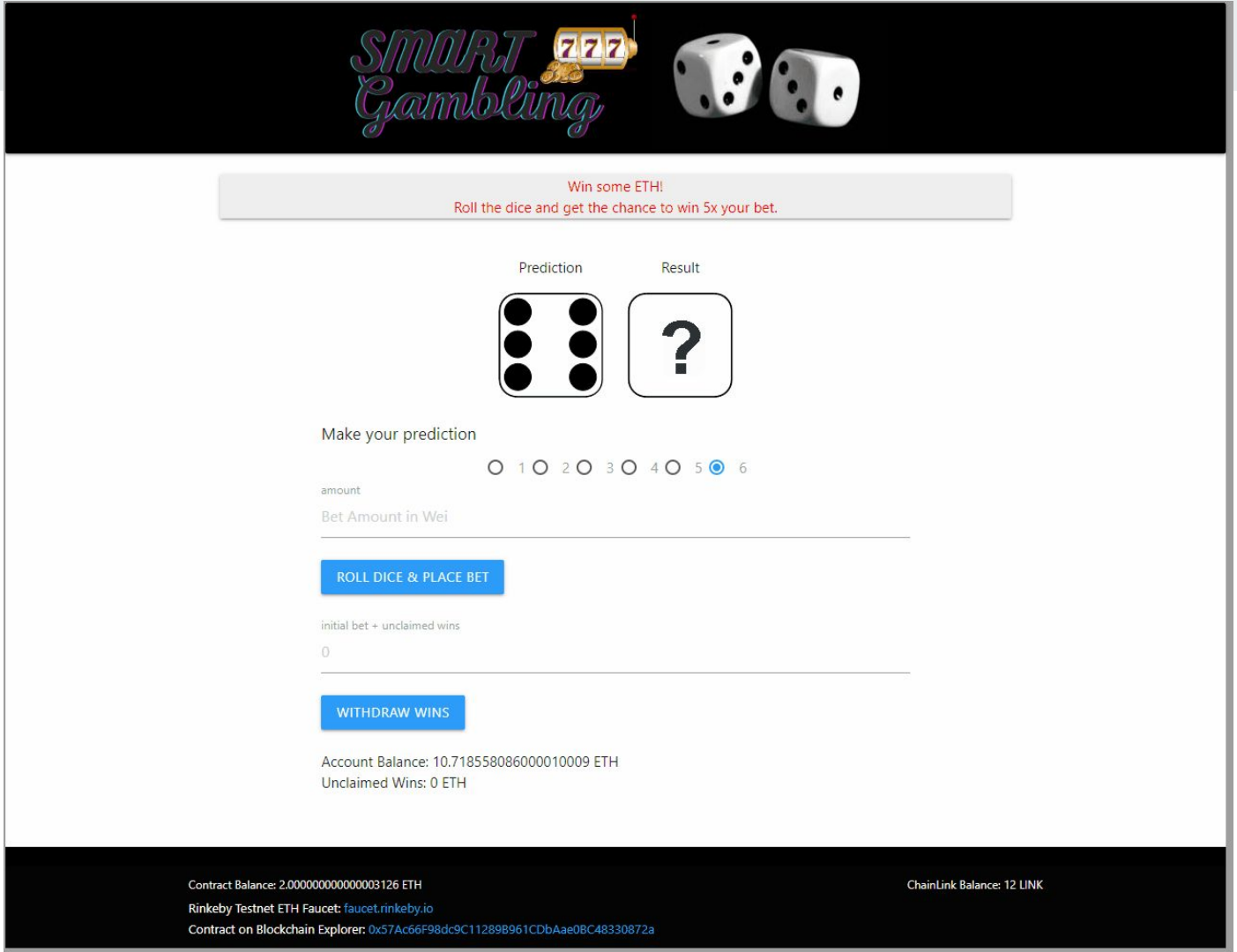
Events

- Bridge between contract & front-end.
- We are using 3 events: BetPlacedEvent, BetResultEvent, withdrawWinsEvent.
- Front-end monitors for these events to refresh GUI.

Players

- Supports multiple players: use player address and timestamp as unique identifier.
- Code must account for:
  - deposits
  - amount owing to players
  - unresolved bets in queue
  - withdrawals

Demo


SMART Gambling
777
Win some ETH!
Roll the dice and get the chance to win 5x your bet.
Prediction
Result
?
Make your prediction
1 2 3 4 5 6
amount
Bet Amount in Wei
ROLL DICE & PLACE BET
initial bet + unclaimed wins
0
WITHDRAW WINS
Account Balance: 10.718558086000010009 ETH
Unclaimed Wins: 0 ETH
Contract Balance: 2.000000000000003126 ETH
ChainLink Balance: 12 LINK
Rinkeby Testnet ETH Faucet: faucet.rinkeby.io
Contract on Blockchain Explorer: 0x57Ac66F98dc9C11289B961CDbAae0BC48330872a

Future Directions

Backend:

- Automatic refill of Link tokens on UniSwap or other platform.
- Determine real world value of Link in order to determine minimum bets.
- Create ERC20 tokens to incentivize and garner loyalty from players.

Frontend:

- More/different games.
- Implement interactive graphics.
- Multi-language interface.
- Provide log/history of player’s bets.

Questions?

Speaking Notes:

Slide 1:

Blockchain technologies are being used for many noble purposes. Not wanting to be different, our group has focused on the noble pursuit of gambling. Blockchain is well suited for gambling: it offers privacy protection and data security all with the ease of virtual transactions. In some cases, a player can circumvent local laws/regulations to gamble, as is their fundamental right!

Speaking Notes:

Slide 2:

Our project creates a smart contract using a GUI for an interface, and a smart contract as the backend. In our game, the player picks a number from 1 to 6, and rolls a six-sided die. If the die comes up with his/her # then they win.

Speaking Notes:

Slide 3:

You can see all the resources that our project uses here. Of particular note, we use Solidty to construct our smart contract in Remix. Our smart contract include OpenZeppelin and SafeMath. We also use an external blockchain database? called Chainlink and we connect it to our smart contract using Web3.js in conjunction with Ethereum JavaScript API. Our balance is checked with MetaMask and we have to use the Rinkeby Ethereum Testnet in order to join up with ChainLink, which we'll discuss shortly. Finally our frontend is done using HTML and Javascript.

Speaking Notes:

Slide 4:

Random number generation is a point of challenge in blockchain programming. Val will discuss this in more detail. For our contract, our solution is to use the ChainLink Oracle in concert with Verifiable Random Function (VRF).

Speaking Notes:

Slide 5:

In today's modern world, businesses tend towards specialization and accordingly do not do complete all tasks involved in product/service creation. They typically use supplier or vendors for inputs. Smart contracts are no different. They often need objective & credible information from outside the contract: an Oracle. Oracles typically provide financial information, but can also provide weather information, and in our case, a random number.

Speaking Notes:

Slide 6:

Random not equal to Random. The philosopher Aristotle would not be happy with this statement. Typically, programs use a “seed” to generate random numbers. Seeds could be mouse movements, or the time. If someone were to figure out the seed in advance, they could correctly predict the random outcome. In most programs, the random # generator is sufficient for its purposes. But blockchain/smart contracts involve the real transfer of asset value. In our case, players really want to be confident that the random outcome is random and not tampered with. If they aren’t confident, then the value of our enterprise collapses. It is therefore too risky to generate random#s within our contract.

Speaking Notes:

Slide 7

There are two other “fascinating” points with random#s and blockchain. First is the potential conflict of interest with miners. Even if we’re confident that we’ve generated a random#, how can we be sure to trust the miners to confirm? If a miner is actually a player, and the value at stake is large enough, they could in their role as a miner, choose to discard losing outcomes and only accept winning outcomes. The second issue involves centralization vs. decentralization. If we were to choose a centralized Oracle, this particular Oracle could go out of service or be corrupt. This increases enterprise risk, so using a decentralized Oracle is important by decreasing the chances of “points of failure”.

Speaking Notes:

Slide 8

On this slide you can see that we evaluated 3 options to the random number challenge.

Speaking Notes:

Slide 9

This slide lays out our choices and our evaluation criteria. As you can see the Chainlink/VRF satisfies all three of our criteria.

Speaking Notes:

Slide 10

Chainlink/VRF has multiple nodes running and therefore removes single point of failure when we seek information outside the contract. In terms of random #s, the Chainlink has checks and balances. When our contract calls Chainlink, we provide the seed. Thus Chainlink miners do not know the seed in advance. In fact they likely don't even know the purpose of generating the random #. So a Chainlink miner solves the computation, the other miners confirm it, and one response is sent back to our contract. This assures randomness. In terms of economic incentives, Chainlink uses carrot and stick. For the stick: Chainlink is a proof of stake blockchain, so miners can lose stake (value) if they provide false answers. For the carrot, our smart contract compensates the chainlink with Link tokens. API providers are compensated by Chainlink operators which we do not have to concern ourselves with: this cost is embedded in the Link price that we pay.

Speaking Notes:

Slide 10

I hope we've sated your fascination with random numbers. On this slide we have "events" in our contract. Events are the bridge between the contract and the front-end. The front-end listens for these events, in our case the three that you see here, refreshes the GUI so as to update the player.

Speaking Notes:

Slide 11

Our contract can handle multiple players. This is because we use the combination of timestamp, and player address as a unique identifier. Therefore we can accommodate a single player using multiple webpages to play our game.

In terms of auditing, to ensure that our contract has enough funds to accommodate all bets in play, our code must keep track of deposits, amounts owing, unresolved bets, and withdrawals.

Now I will hand it off to Sebastien to take us through the game.

Random Numbers

- Block Hash:
  - Potential conflict of interest with miners
  - If miners are betting, might not confirm losing bets and only confirm winning bets.
  - How can a non-mining player be sure?
  - Not Usable for our application.

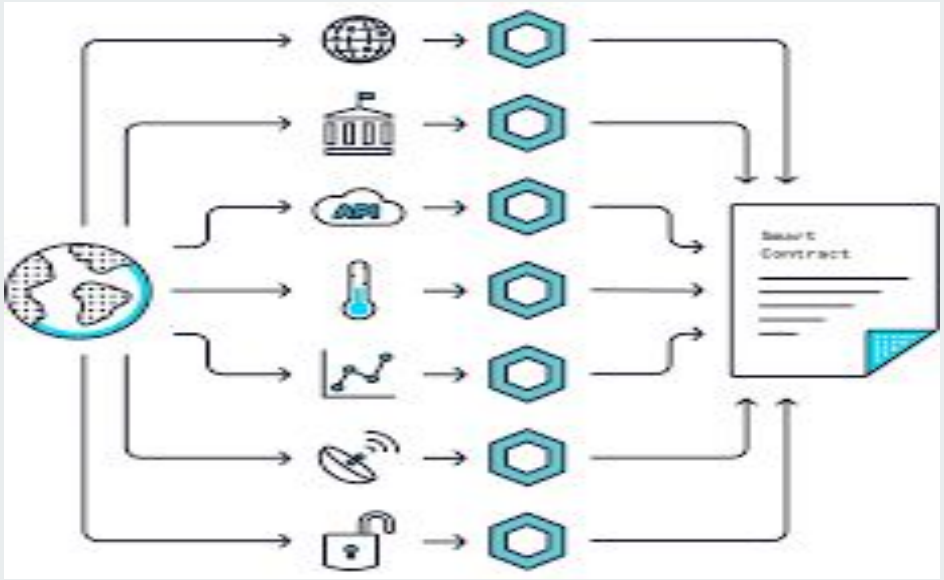
Random Numbers

- Centralized, External API or Oracle.
  - How can a player be sure that the outside source is truly generating a random #?
  - What if the sight goes down? Or corrupted?
  - Goes against principle of decentralization.
  - Not Usable for our application.

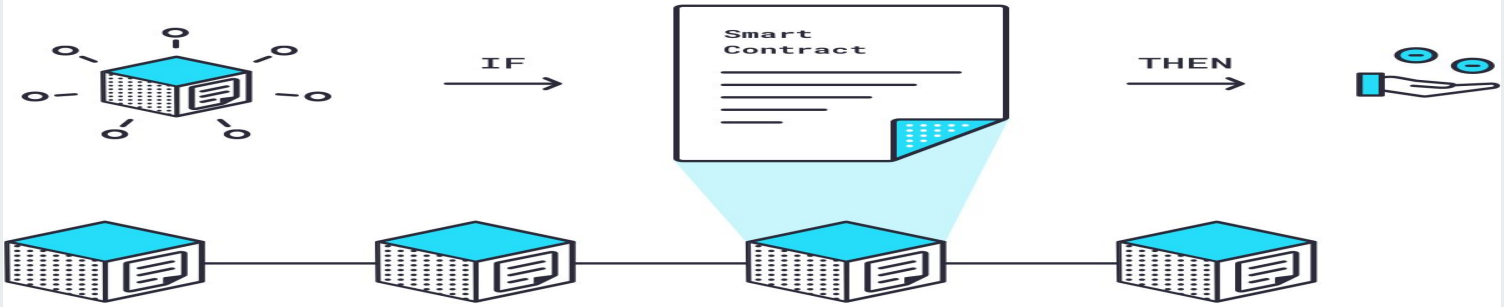
Random Numbers

- Solution: VRF.
  - Utilizes Chainlink = a blockchain.
  - Random # is verified/confirmed within the Chainlink blockchain.
  - So is decentralized, **not controlled**.

Chainlink Contracts

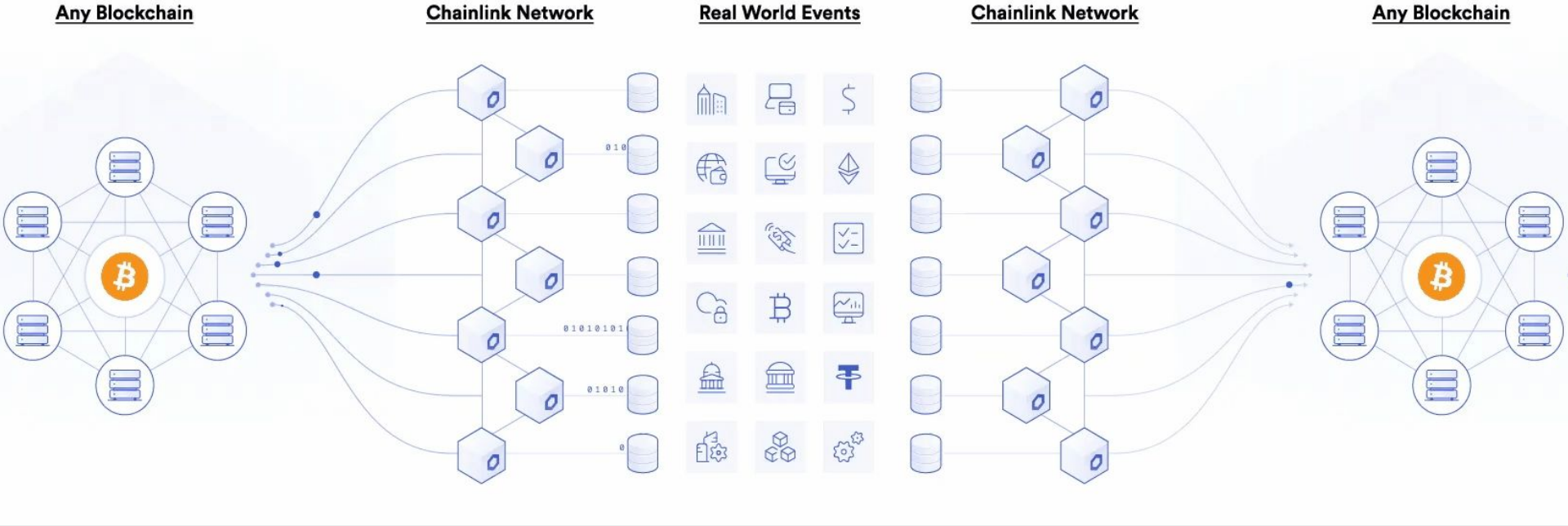


The benefits of Chain Link and how it functions



- Smart contracts are pre-specified agreements on the blockchain.
- It evaluates information and automatically execute when certain conditions are met.
- Crowdfunding is a good example: if a certain amount of Ether is deposited into a smart contract by a certain date, then payment will be released to the fundraiser


Real World Data
Chainlink Network
Any Blockchain
Chainlink Network
Real World Events


Any Blockchain
Chainlink Network
Real World Events
Chainlink Network
Any Blockchain
010
01010101
01010
0

What is an Oracle?

- Sends data from the outside world to the blockchain
- Smart contracts aren’t designed to take care of everything
- Just like normal businesses, sometimes rely on suppliers or vendors, in this case, an Oracle such as Chainlink.

Speaking Notes:

Slide 7:

Random not equal to Random. The philosopher Aristotle would not be happy with this statement. Typically, programs use a “seed” to generate random numbers. Seeds could be mouse movements, or the time. If someone were to figure out the seed in advance, they could correctly predict the random outcome. In most programs, the random # generator is sufficient for its purposes. But blockchain/smart contracts involve the real transfer of asset value. In our case, players really want to be confident that the random outcome is random and not tampered with. If they aren’t confident, then the value of our enterprise collapses.