PROBLEM SOLVING USING BLOCKCHAIN

B. RAMAMURTHY

BINA@BUFFALO.EDU

HTTP://WW.CSE.BUFFALO.EDU/FACULTY/BINA

RESEARCH ASSOCIATE PROFESSOR

COMPUTER SCIENCE AND ENGINEERING

DIRECTOR, BLOCKCHAIN THINKLAB

UNIVERSITY AT BUFFALO

APRIL 12, 2018

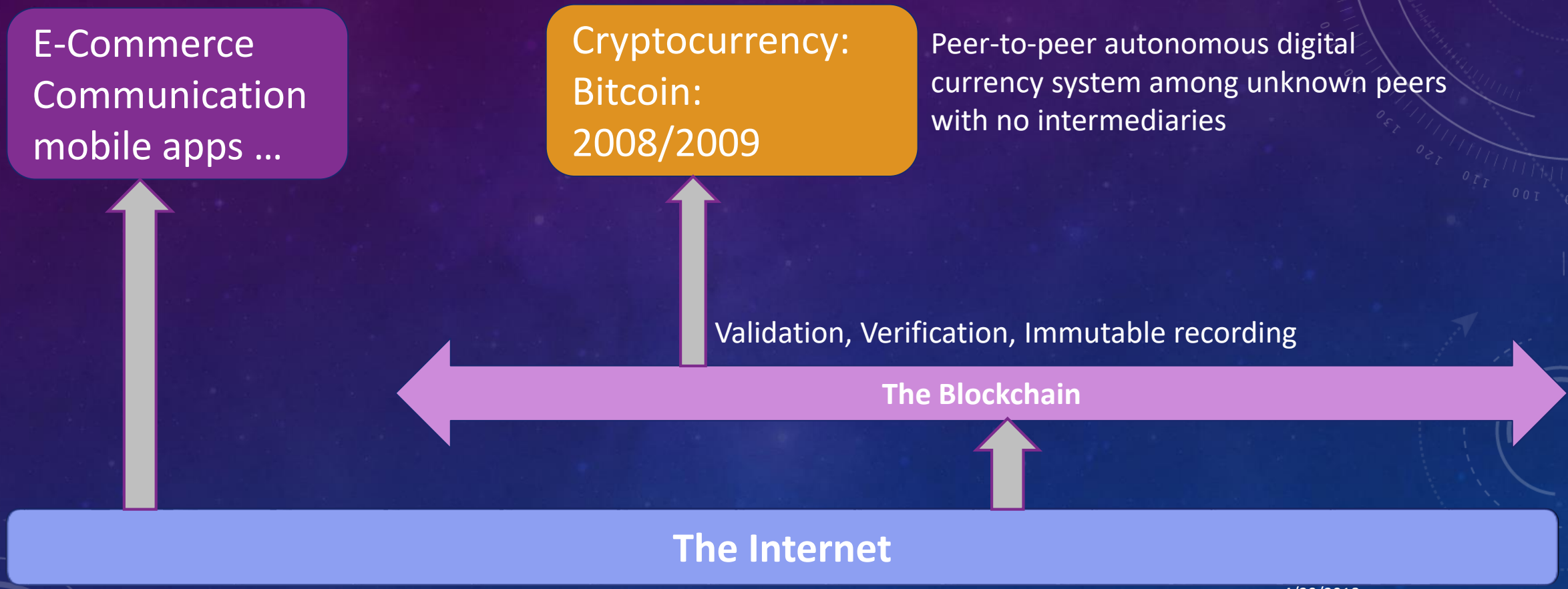
OVERVIEW AND GOALS

1. Blockchain Concepts (slides 3-17)
    - A brief history of blockchain
    - What is a blockchain?
2. Introduction to smart contracts (slides 18-43)
    - Designing and Developing SC
    - Solidity and Remix IDE
3. Problem Solving using Blockchain (slides 44-47)
    - Blockchain Buildathon: How to?
- Summary (slides 48-49)
- References (slide 50)
- Q & A

BLOCKCHAIN CONCEPTS

4/20/2018

A BRIEF HISTORY OF BITCOIN BLOCKCHAIN



A BRIEF HISTORY OF THE BLOCKCHAIN

Two major contributions of the famous cryptocurrency Bitcoin are

(i) A working digital currency system and

(ii) An autonomous decentralized technology called the blockchain.

WHAT IS A BLOCKCHAIN?

The Blockchain is

1. An enabler for **decentralization, without intermediaries.**
2. A model for **trust management** in a network where peers operate beyond boundaries of trust.
3. An **immutable ledger** of records.

Enabled automation, accountabilty, auditability, efficiency, accuracy, fidelity, fairness, and inclusivenes.

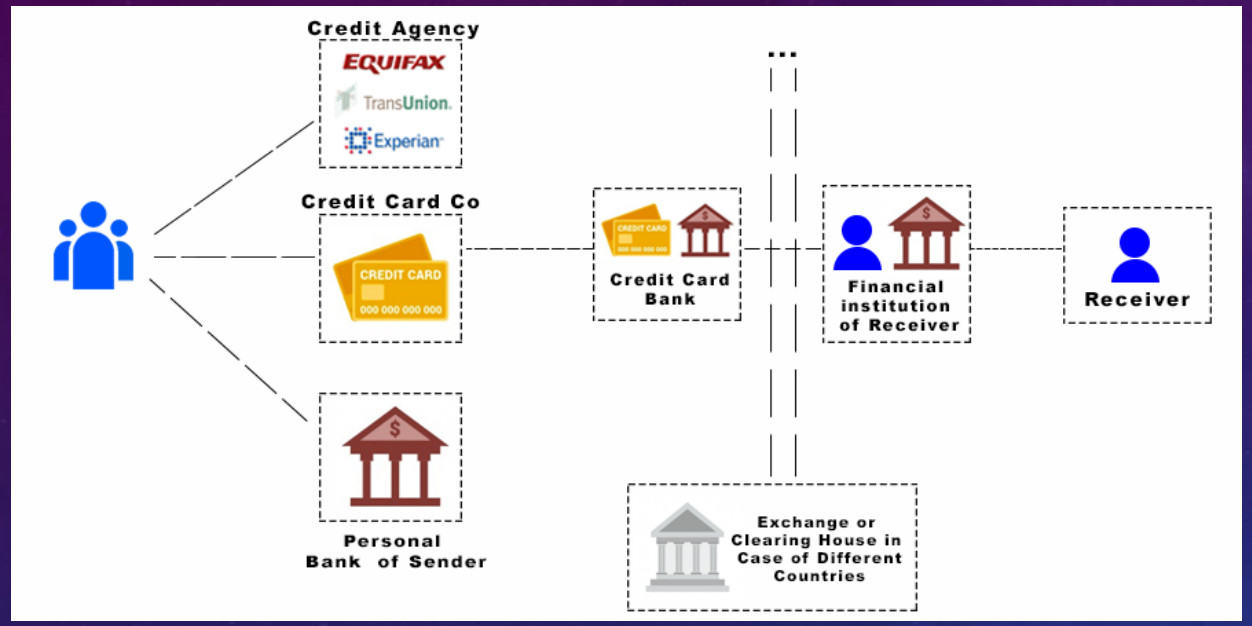
IT IS NOT ALL ABOUT CRYPTOCURRENCY!

- Peer to peer transactions with no intermediaries (no middleperson)
- Among unknown peers (not necessarily known to each other)
- Peers thus operate outside boundaries of trust
- Decentralized, peers hold their assets themselves (bearer assets)
- Transactions are not necessarily about currency (e.g. reports, complaints, warning signals, grades)

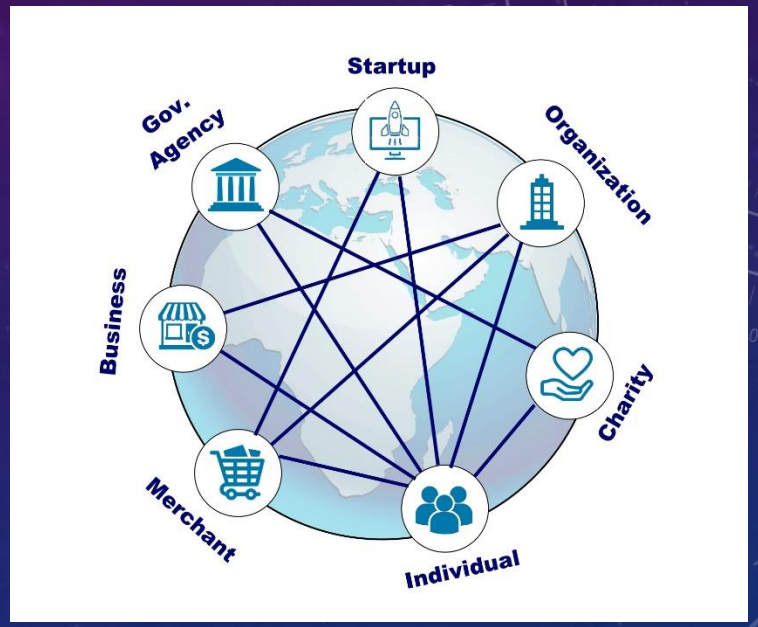
BLOCKCHAIN TO THE RESCUE

- Blockchain protocol (software, hardware) takes care of validation, immutable recording and verification of transactions.
- The protocol is robust; it has a strong foundation established on the outcomes from more than 30 years of scientific research in
  - Public key cryptography,
  - Secure hashing (SHA),
  - Peer-to-peer networks,
  - Consensus/agreement protocols

BLOCKCHAIN: THE DECENTRALIZATION INFRASTRUCTURE - 1



Traditional Centralized System



Decentralized System

Functions of the intermediaries are shifted to the peer participants and the blockchain nodes: Disintermediation.

BLOCKCHAIN: THE DISINTERMEDIATION - 2

The Blockchain infrastructure supports methods to:

-- **validate**, **verify** and confirm **transactions**

-- record the transactions in a **distributed ledger** of blocks

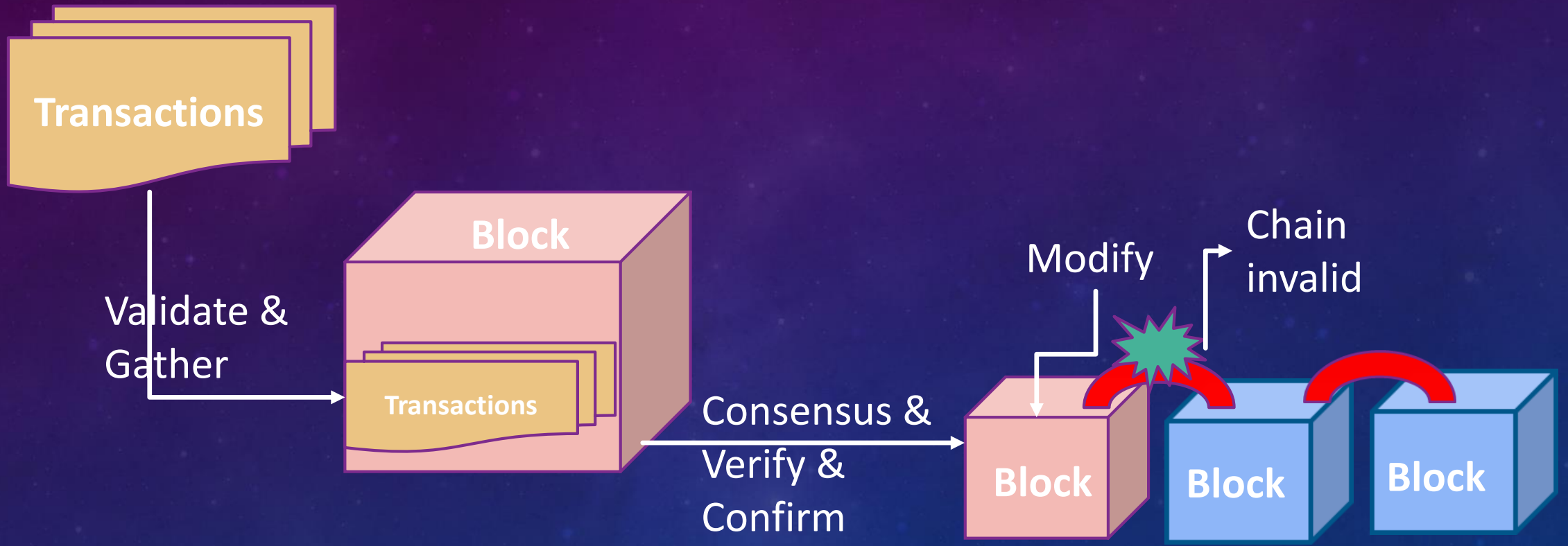
-- implement a **consensus protocol** for agreement on the validity of blocks

-- create a **tamper-proof record of blocks** (chain of blocks)

Validation, Verification, Consensus, Immutable Recording ➔ Trust, Security

All defined by protocols, implemented by software and realized autonomously.

BLOCKCHAIN: THE DISTRIBUTED IMMUTABLE LEDGER - 3

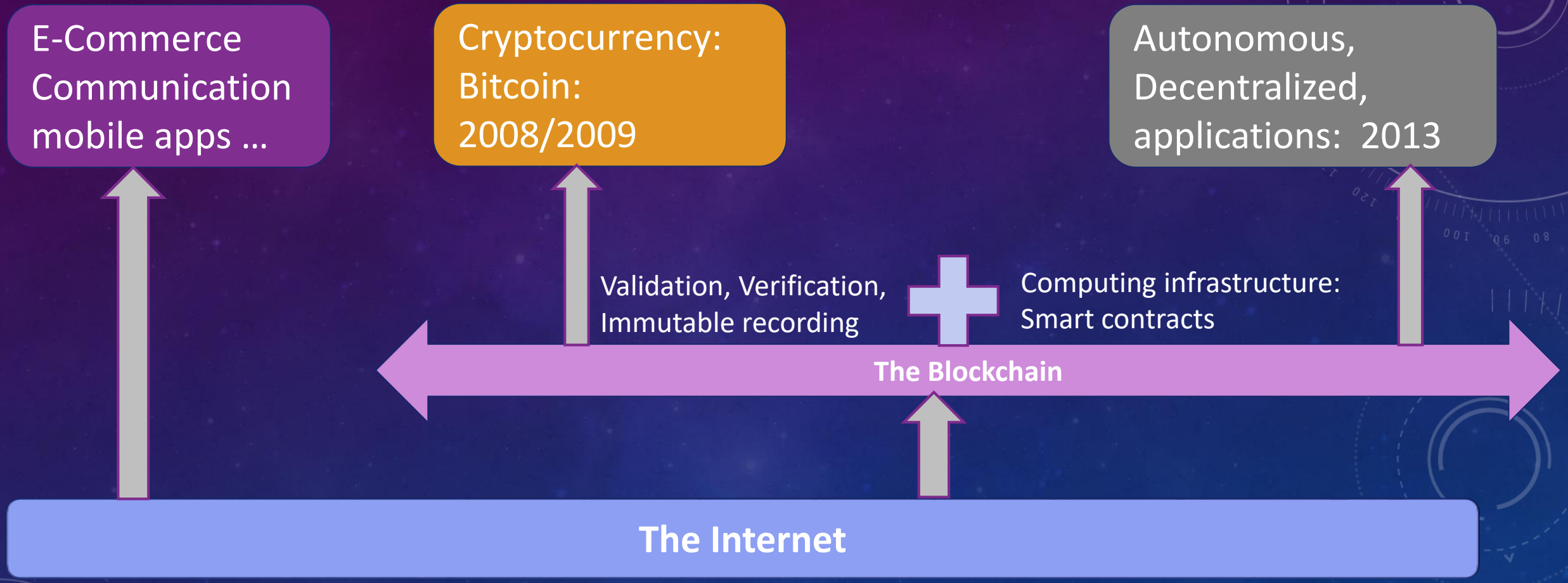


Chain link is the hash of elements of the previous block

SUMMARIZING, BC IS ABOUT

- Decentralization
- Disintermediation
- Distributed Immutable ledger

EVOLUTION OF THE BLOCKCHAIN INTO COMPUTING



SMART CONTRACTS

- Typically currency transfer is for buying a service, product and utility from a person or a business. There may be other conditions besides availability of funds for executing a transaction.
- A business transaction may involve rules, policies, laws, regulations and governing contexts.
- Smart contract allows for these real-world scenarios to be realized on a blockchain.
- Thus a smart contract enables a wide variety of decentralized applications of arbitrary complexity to be implemented on the Blockchain:
  - **from supply chains to disaster recovery**.
- Probably many of the applications for the Blockchain technology have not been conceived of yet.

SMART CONTRACTS USING SOLIDITY AND REMIX IDE

4/20/2018

LEARNING OBJECTIVES

- Define the concept of a smart contract.
- Discuss the syntax and the semantics of a smart contract programming language, Solidity.
- Solve a problem and design a smart contract solution.
- Learn Remix development environment for building and testing smart contracts.
- Deploy the smart contract using Remix and invoke it from the web interface provided by Remix IDE.

PEDAGODY

- It is imperative that you try the various concepts related to smart contracts on a test environment in order to understand and apply these concepts.

- We will use Remix Integrated Development Environment (IDE) that is a web interface for these hands-on exploration. At this time please check you are able to access this interface at remix.ethereum.org

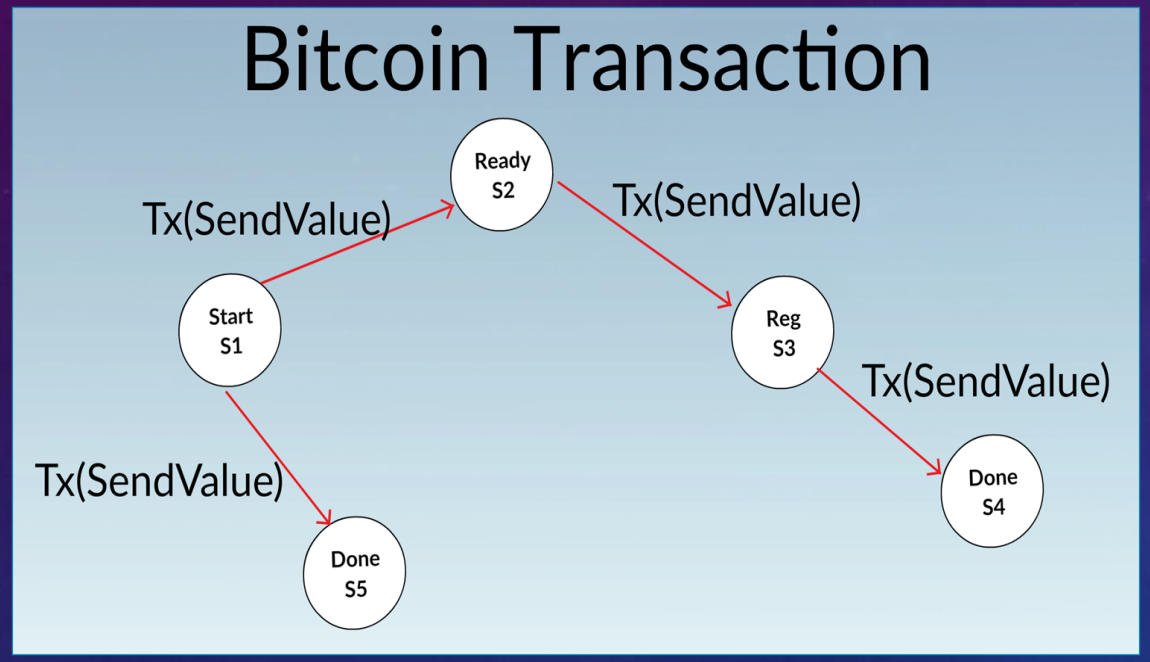
- Be warned that remix is only a development environment and it keeps changing as new features are being added almost every week.

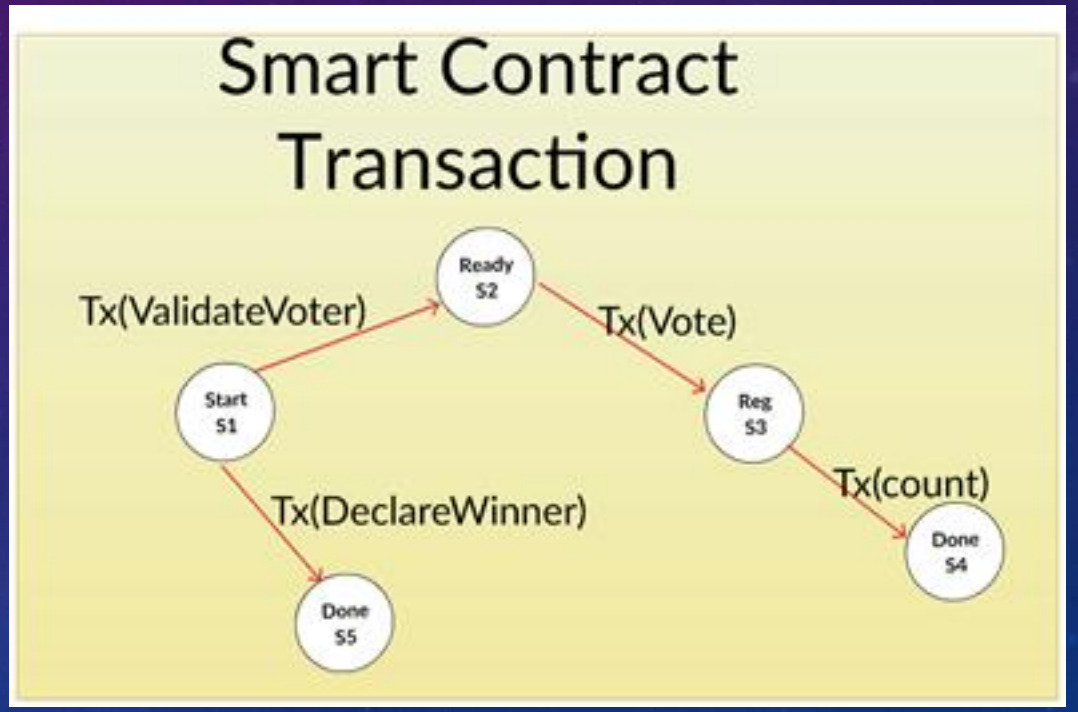
WHAT IS A SMART CONTRACT?

WHAT IS A SMART CONTRACT?

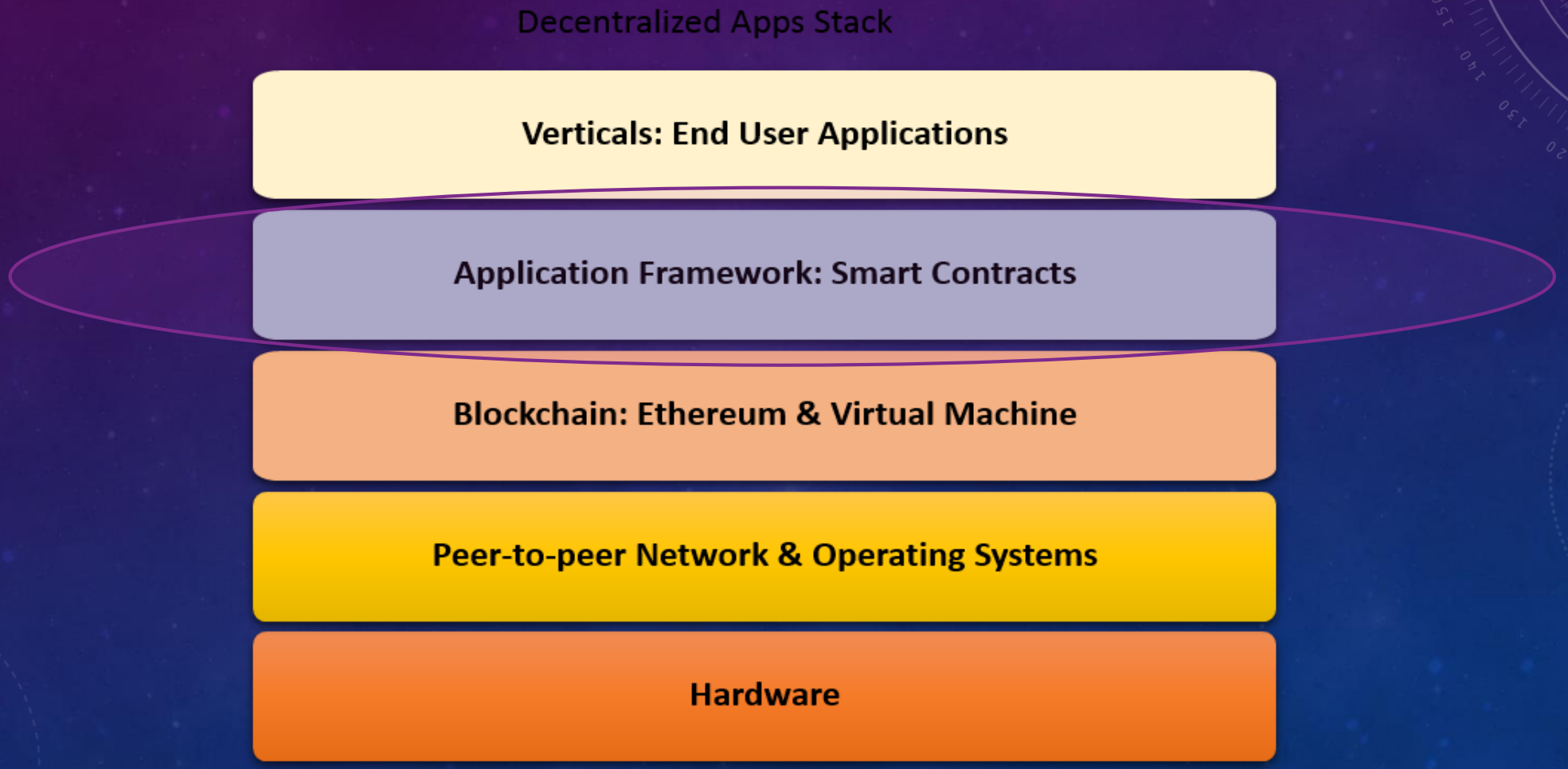
- A smart contract facilitates transactions for transfer of assets other than cryptocurrency.
- A smart contract allows specification of rules for an operation on the blockchain.
- It facilitates implementation of policies for transfer of assets in a decentralized network.
- The smart contract represents a business logic layer with the actual logic coded in a special high-level language.
- A smart contract embeds functions that can be invoked by “messages” that are like function calls. These messages and the input parameters for a message are specified in a transaction.

TRANSFER OF CURRENCY VS. SMART CONTRACT TRANSACTION (BY SEAN SANDERS)





DAPP STACK (DECENTRALIZED APPLICATION STACK)ON ETHEREUM BLOCKCHAIN



WHAT PROBLEM DOES A SMART CONTRACT SOLVE?

- Typically currency transfer is for buying a service, product and utility from a person or a business. There may be other conditions besides availability of funds for executing a transaction.
- A business transaction may involve rules, policies, laws, regulations and governing contexts.
- SC allows for these real-world scenarios to be realized on a blockchain.
- Smart Contract solves the problem of conditional transition from one state of the blockchain network to another.
- Thus a smart contract enables a wide variety of decentralized applications of arbitrary complexity to be implemented on the Blockchain: from supply chains to disaster recovery.
- Probably many of the applications for the Blockchain technology have not been conceived of yet. It is predicted that online shopping will takeover retails shopping for the first time this holiday season (2017-2018). Could you have imagined the dominance online shopping, mobile apps and uber 20 years ago?

BASIC STRUCTURE OF A SOLIDITY SMART CONTRACT

Contract in the Ethereum Blockchain has:

1. Pragma directive
2. Name of the contract
3. Data or the state variables that define the state of the contract
4. Collection of functions to carry out out the intent of an application
5. Other items we will discuss in the next lessons.

Identifiers representing these elements are restricted to ASCII character set. Make sure you select meaningful identifiers and follow Camel Case convention in naming them.

We will learn these concepts using simple contracts written in a high level language called Solidity.

Smart Contracts in Soldity by Example

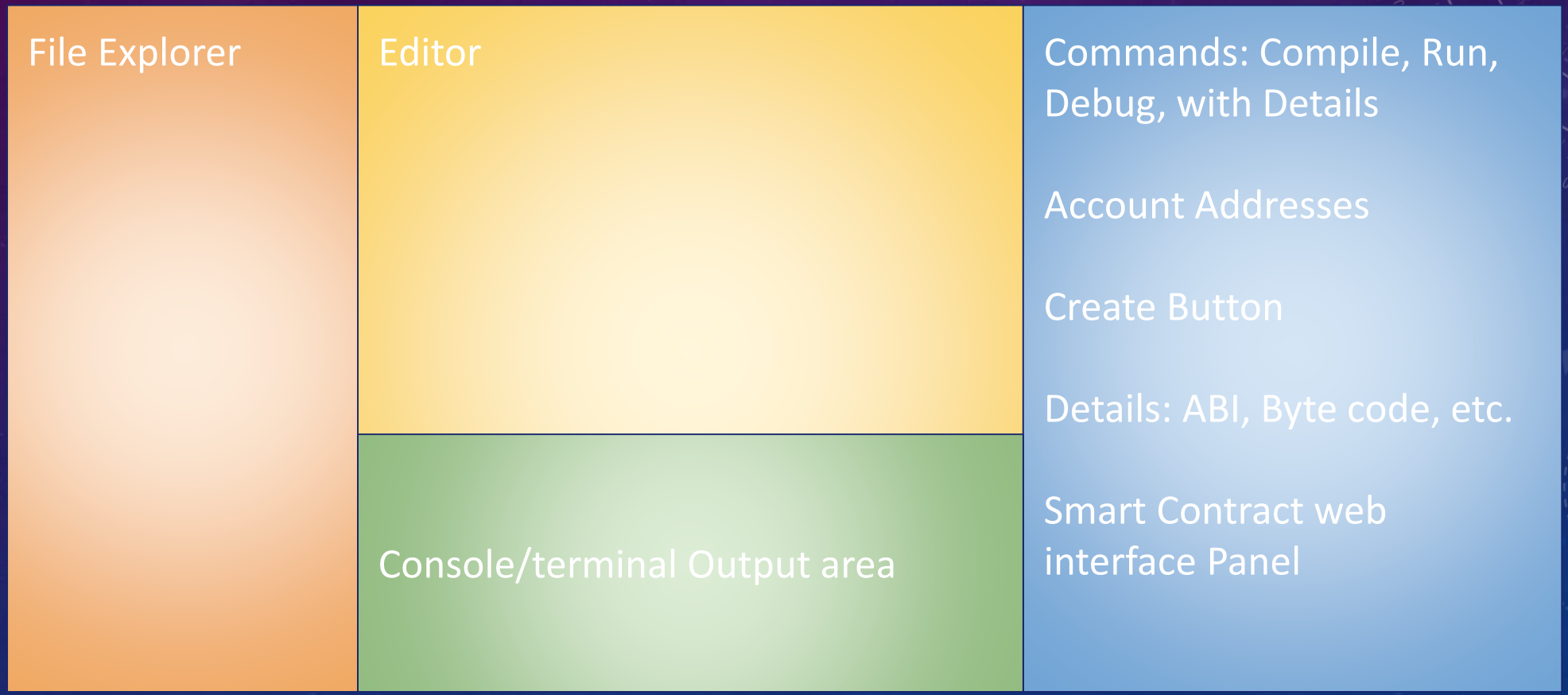
We will examine two a simple smart contracts:

1. Greeter
2. Coin

- These two examples are modified versions of the examples given in Solidity documentation.
- Our goal is to get an overview of the structure of a smart contract without getting into the details of the Solidity language.
- However we will explain every item in the smart contract we plan to explore.

```
pragma solidity ^0.4.0;

contract Greeter  {
    /* Define variable greeting of the type string */
    string public yourName;

    /* This runs when the contract is executed */
    function Greeter() public {
        yourName = "World";
    }

    function set(string name)public {
        yourName = name;
    }

    function hello() constant returns (string) {
        return yourName;
    }
}
```

REMIX ENVIRONMENT



Smart Contract Creation & Execution

- Let us now explore the creation and execution of smart contracts.
- We will begin with Remix environment. remix.ethreum.org
- Then we will compile and execute the two simple contracts” Greeter and Coin and understand the structure of a contract.
- We will also learn the features of the Remix IDE for supporting development and testing of contracts for your applications.

HERE ARE SOME BASIC ELEMENTS OF A SMART CONTRACT

1. Data or state variables
2. Functions : There are several types of functions allowed,
    1. Constructor (default or user specified; only one, meaning it cannot be overloaded)
    2. Fallback function (This is a powerful feature of an anonymous function that we will discuss in the best practices module later in this course.)
    3. Pure functions (no state change allowed, it computes and returns a value; example: math functions)
    4. Public functions (accessible from outside thru transactions, state changes recorded on the bc).
    5. Private functions (accessible only with the current contract)
    6. Internal functions (accessible inside current contract and inherited contract)
    7. External functions (can be accessed only from outside the smart contract, for viewing data, for example)
3. User defined types in struct and enums
4. Modifiers
5. Events

FUNCTION DEFINITION

Now we will step into the smart contract and look at the function definition: function definitons are similar to functions in any other high level language;

Function header followed by the code within curly brackets.

function header { function code }

Function code contains the local data and statements to process the data and return the results of the processing.

SOLIDITY DATA TYPES BY EXAMPLE

- Solidity supports many of the basic data types of a high-level language, we have given here just a few:

- uint : unsigned int of 256 bits
- int : integer, positive and negative value of 256 bits
- string: string of characters
- bool : that supports logic true and false value

- Default modifier for data is private, you explicitly state the “public” modifier if that is what is intended. For every data declared public, accessor (getter) function is automatically provided.

BASIC STATEMENTS

- The common statements available in any high level language are available in Solidity with very little variation: assignment statement, if..else, while, for, etc.
- We will learn them as we add code elements to the examples.
- We will develop Bidder smart contract in incremental steps starting with the basic design representation shown here. Always design before you code.
- Let us move to the Remix environment to work on this example.

ADDRESS DATA STRUCTURE

- Address is a special Solidity defined complex data type; It can hold a 20 byte Ethereum address, recall that it is the reference address to access a smart contract.
- Address data structure also contains the balance in Wei of the account represented by the address; it also supports a function “transfer” to transfer value to a specific address. This is shown in the figure here.

<address>.balance (uint256):

balance of the Address in Wei

<address>.transfer(uint256 amount):

transfer given amount of Wei to Address

MAPPING DATA STRUCTURE

- “Mapping” is a versatile data structure that is similar to a <key, value> store. It can also be thought of a hash table.
- The key is typically a secure hash of simple Solidity data such as the address, and the value in <key,value>pair can be any arbitrary type.

```
mapping (uint => string) phoneToName;


struct customer { unit idNum;
                  string name;
                  uint bidAmount;}
```

MSG IS SPECIAL DATA STRUCTURE

- msg is a complex data type specific to smart contract.
- It represents the call that is used to invoke a function of a smart contract.

- It supports many attributes, of which, we are interested in two of them now. You can always look up other msg details in the Solidity documentation.

- msg.sender that holds the address of the sender, msg.value that has the value in Wei sent by the sender:

```
address adr = msg.sender
uint amt = msg.value
```

- Usage: now you can write statements that verify and validate adr and amt to make sure the application specifics are met.

USE THESE DATA STRUCTURES: ADDRESS, MAPPING AND MSG

- Let us now use these data structures and mint some money.
- Let us look at a smart contract for “Coin” as specified in the Solidity documentation.

- Recall we always start with a design (say, a class diagram) whether we are analyzing a program or coding an application.

- Also make a note of the statements for if..else and assignment statement.

```
contract Coin {
    // The keyword "public" makes those variables readable from outside.
    address public minter;
    mapping (address => uint) public balances;
    // Events allow light clients to react on changes efficiently.
    event Sent(address from, address to, uint amount);
        function Coin() public{
        minter = msg.sender;
    }
    function mint(address receiver, uint amount) public  {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }
    function send(address receiver, uint amount) public {
        if (balances[msg.sender] < amount) return;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        Sent(msg.sender, receiver, amount);
    }
}
```

MODIFIERS

- Modifiers can change the behavior of a function: that is why this feature is referred to as a “modifier”
- It is also known as a function modifier since it is specified at the entry to a function and executed before the execution of a function begins.
- You can think of a modifier as a screen/filter protecting a function.
- A modifier is typically checks a condition using a “require” and if the condition failed, the transaction that called function can be reverted using revert function. This will completely reject the transaction and revert all its state changes. There will no recording on the blockchain.

MODIFIER AND REQUIRE

Let us understand modifier and require using the Coin smart contract functions:

We will add a modifier “onlyOwner” to the mint function. We will do this by these steps:

1. Define a modifier for the clause “onlyOwner”
2. Adding the special notation (_;) to the modifier definition that includes the function
3. Using the modifier clause in the function header

MODIFIER FOR COIN.SOL

```
modifier onlyOwner()
  {
     require(msg.sender == minter);
     _;
  }
  // This is the constructor whose code is
  // run only when the contract is created.
  function Coin() public{
     minter = msg.sender;
  }
  function mint(address receiver, uint amount) onlyOwner public  {
    // if (msg.sender != minter) return;
     balances[receiver] += amount;
  }
```

IMPORTANCE OF MODIFIERS

- Function modifiers along with the state-reverting functions of revert(), require() collectively support a robust error handling approach for a smart contract.

- These declarative features can be be used to perform formal verification and static analysis of a smart contract to make sure it implements the intent of a smart contract.

Summary

- We learned the purpose of smart contract and its critical role in transforming blockchain technology from enabling decentralized systems.
- We explored the structure and basic concepts of a smart contract through examples.
- We illustrated Remix (remix.ethereum.org) web IDE for deploying and interacting with a smart contract.
- We learned basics of Solidity language.

HOW TO SOLVE A PROBLEM WITH BLOCKCHAIN? THERE IS A DAPP FOR THAT.

- Blockchain is NOT solution for all your problems. It is NOT a data repository.
- You save only the “sliver” of info that is needed for business logic, provenance and governance on the blockchain.
- Write the use cases
- Choose your blockchain. Lets say we chose Ethereum. And Ethereum is not the only blockchain technology. There are many others.
- Design and Implement a smart contract solution in Solidity Language
- Test it using Remix IDE (Integrated Development Environment)
- Develop end-to-end Dapp using Truffle IDE with a graphical user interface
- Deploy it on a blockchain for the whole decentralized world to use.

PROBLEM SOLVING USING BLOCKCHAIN

4/20/2018

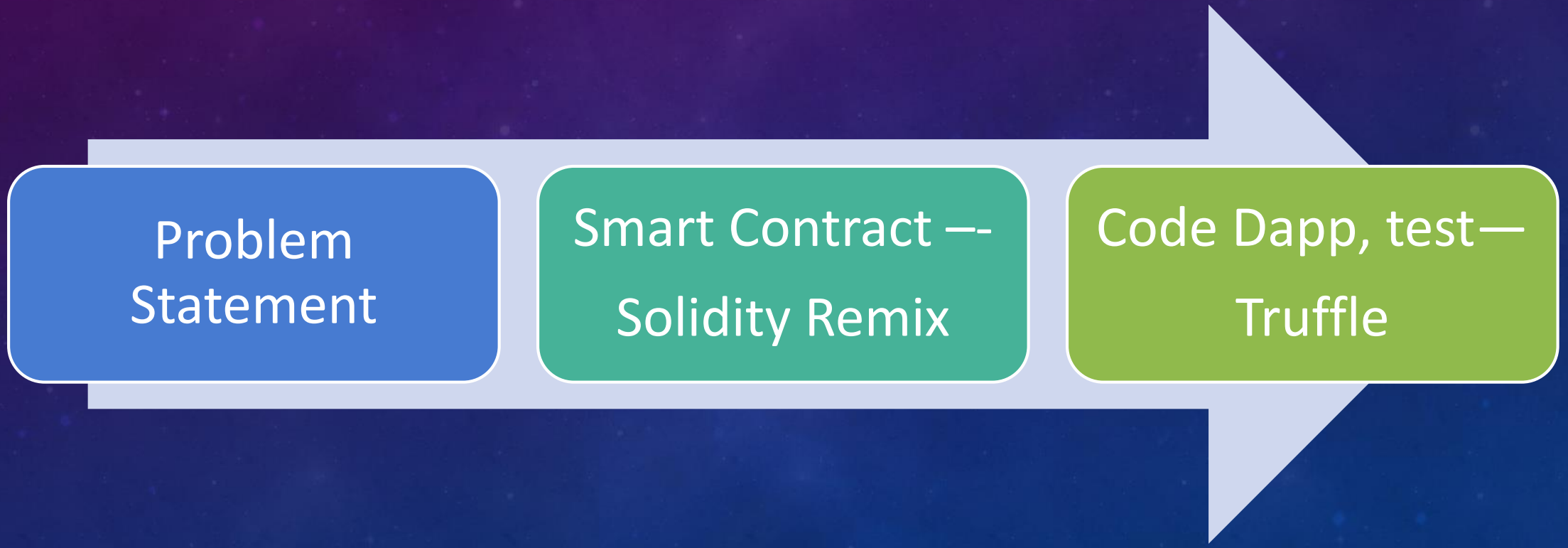
Decentralized Application Development

- Problem statement

- Analysis and design of smart contract solution
- Implement and test the smart contract using Remix IDE

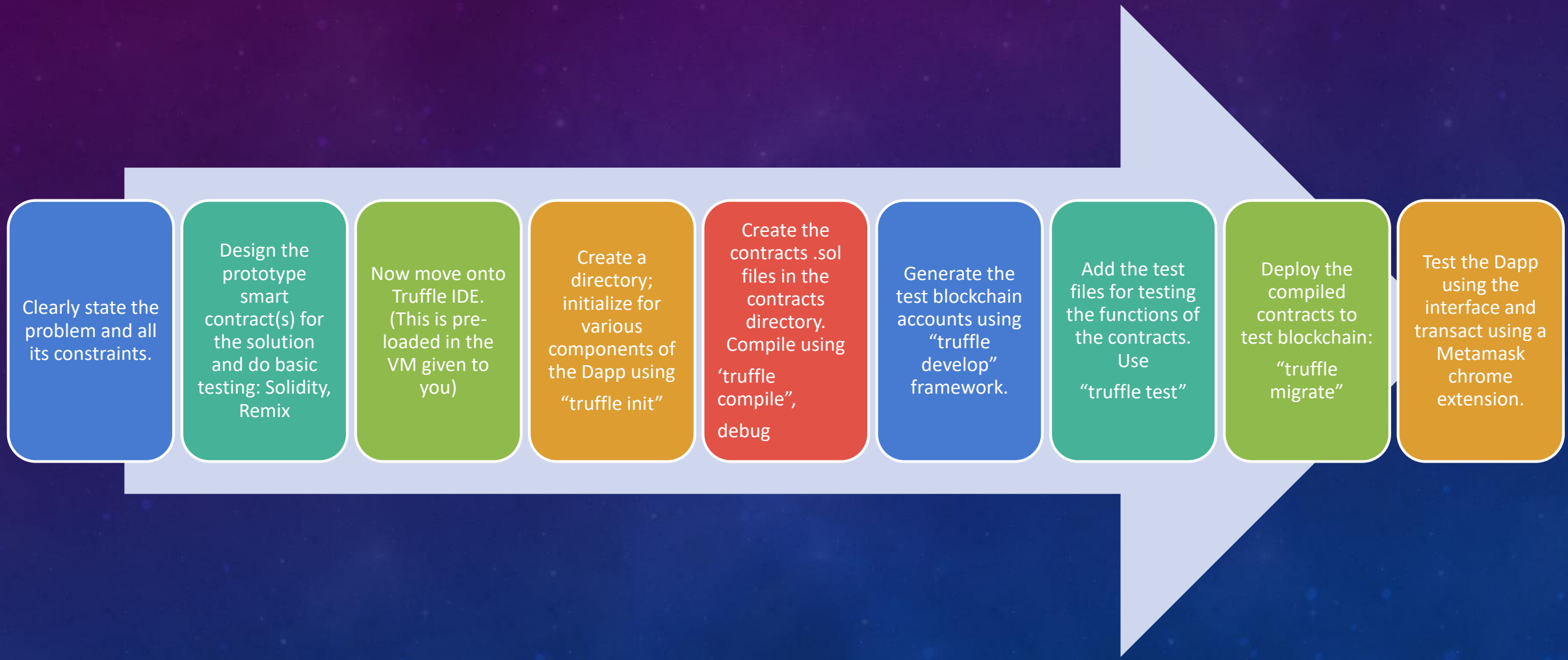
- Deploy the smart contract on the blockchain and test it
- Design and implement the front end and connect it to blockchain backend

Dapp Design Process & Demo



A METHOD FOR DEVELOPING A DAPP

Code Dapp, test—Truffle



B

BLOCKCHAIN OPPORTUNITIES

- Blockchain users, reporters
- Contributing Peer participants in blockchain application platforms
- Blockchain application development and education
- Disciplinary /vertical domain research in investigating application of blockchains (legal, healthcare, government, fintech…)
- Blockchain Dapp development tools and frameworks
- Blockchain protocol improvement research and decision makers
- Blockchain alternatives for decentralized systems
- Fundamental research in blockchain algorithms

UB INITIATIVES: HOW CAN WE HELP?

- UB SEAS is at the leading edge of educating people about this technology.
  - We are in the process of cresting a Coursera Blockchain “Specialization” that has 4 courses of 4 weeks of online instruction each with practical hands-on components.
- UB Blockchain Thinklab has been created with the support from President’s Circle Club funding
  - Premier event: UB Blockchain Buildathon April 13-15, 2018 See ethBuffalo.org
  - Educating UB students across the campus
  - Locating resources for your research and answering your blockchain questions
  - Consulting with regional and international businesses
  - Promoting Buffalo as a world renowned blockchain hub

SUMMARY

- Blockchain technology is not about cryptocurrency anymore.
  - It is used for broad range of applications across many industries: finance, healthcare, government, manufacturing, and distribution.
- Blockchain can enable an inclusive economy.
- Blockchain has created exciting new opportunities and innovative application models:
  - Global colloboration systems, self-governing systems, open government.
  - Private, public and permissioned (consortium) models to meet diverse business needs.
  - There is a role to play for each and everyone of you.

REFERENCES

1. S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. http://www.bitcoin.org, 2008.
2. Ethereum Project: Homestead and Platform. https://www.ethereum.org/, last view 2017.
3. Introduction to Smart Contract. https://solidity.readthedocs.io/en/develop/introduction-to-smart-contracts.html#. Last viewed 2017.
4. The Hyperlegder Project. https://www.hyperledger.org/, last viewed 2017.
5. Remix IDE: remix.ethereum.org, last viewed 2018.
6. Truffle IDE : http://truffleframework.com/docs/, last viewed 2018.