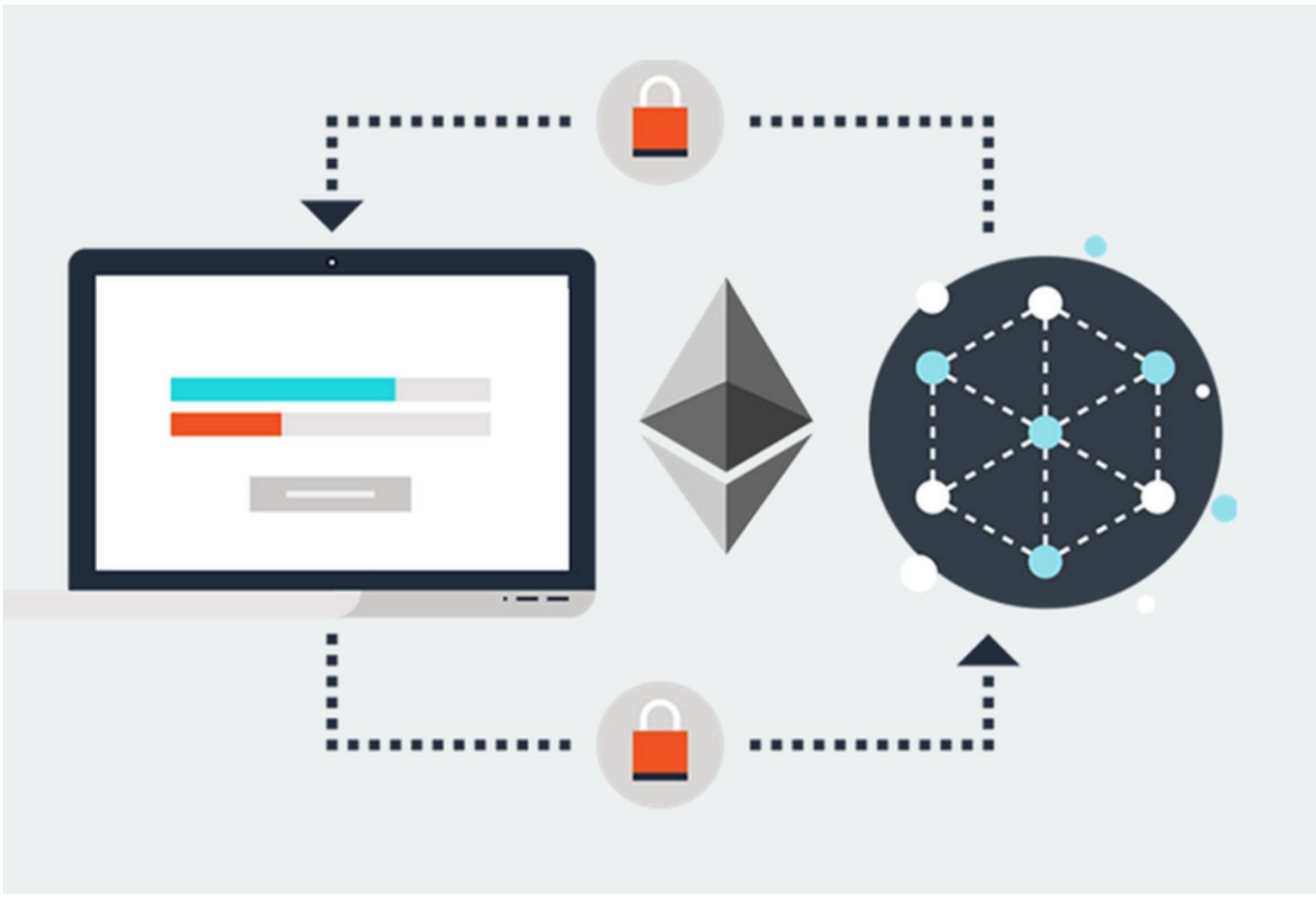
# Part 1: Preparing a Smart Contract for our dApp

- Learning Objective 1: Students will gain experience writing a moderately more complex smart contract that makes use of a few more Solidity programming language constructs.
- Learning Objective 2: Students will prepare their compiled contract into an ABI specification for use with Web3 in their dApp later this week.

---



Welcome back to more fun with Solidity!

Now that you've got your feet wet, it's time to dive in.

We'll start early, but the goal of this assignment is to start getting ready a moderately more complex Smart Contract than the variety we saw last week. We'll want to start this contract early because we will need it later this week for a dApp. Specifically we will be building a Smart Contract that has a few extra Solidity programming language constructs. Together we'll get to see:

- Structs
- Arrays
- The require statement
- and Modifiers

Let's expand our Bank contract from last week.

This time we'd like to include the functionality that users should be required to register before their allowed to make use of protected methods. To do so we'll use modifiers. To keep track of our user's we'll make use of of a counter, mappings, and an array.

Modifiers work much the same way as decorators do in Python, if you're familiar with them.

They allow us to enter a code-block before the function itself, allowing for further modularity and the reduction of repeated logic. This improves code quality, reduces opportunities for errors, simplifies testing and improves code readability.

We'd like to add a modifier that protects the withdraw(), transfer() and a get_balance() method we will add.

But how will we know which users are registered? We'll have to add a method for that too!

Let's keep track of our number of accounts with a property - number_of_accounts.

We can associate each address with a new integer incremented from this property.

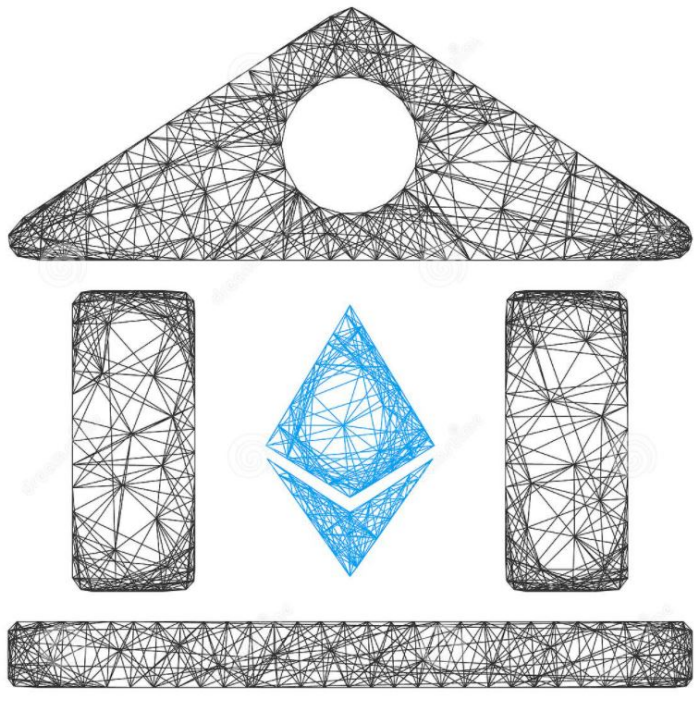
Why would we do that? Because we will place some KYC-esque information into a registration file stored in our Smart Contract.

How will we do that? With a struct!

We can easily define structs in Solidity with the keyword "struct", they're a lot like a class but without the methods and constructors. We will want to place the (wallet) address of the customer in this struct. But that would be super painful to lookup. To make life a lot easier we can keep a mapping handy for each record (struct) in our array, if a user's address has a non-zero value in our mapping, we know they're registered.

See where this is going yet?

---



## Step 1. Putting together our contract.

Let's get started!

We'll leave a few choices up to you. We'll also leave it to you to make the best choices regarding what should be public, private, etc

You'll need:

1. A fun name for your new Smart Contract

   Some Properties:

2. A KYC struct with a few fields, we'd recommend the following, but feel free to get creative:

1. A customer ID (uint)
2. A full name (string)
3. A profession (string)
4. Date of Birth (string)
5. Wallet Address

3. A counter property with an appropriate name and datatype, maybe number_of_accounts and uint?
4. A mapping of account balances, just like before.
5. An array of KYC Struct Records or similar as per above.
6. Our NEW mapping from uints to the location of the record in your array.

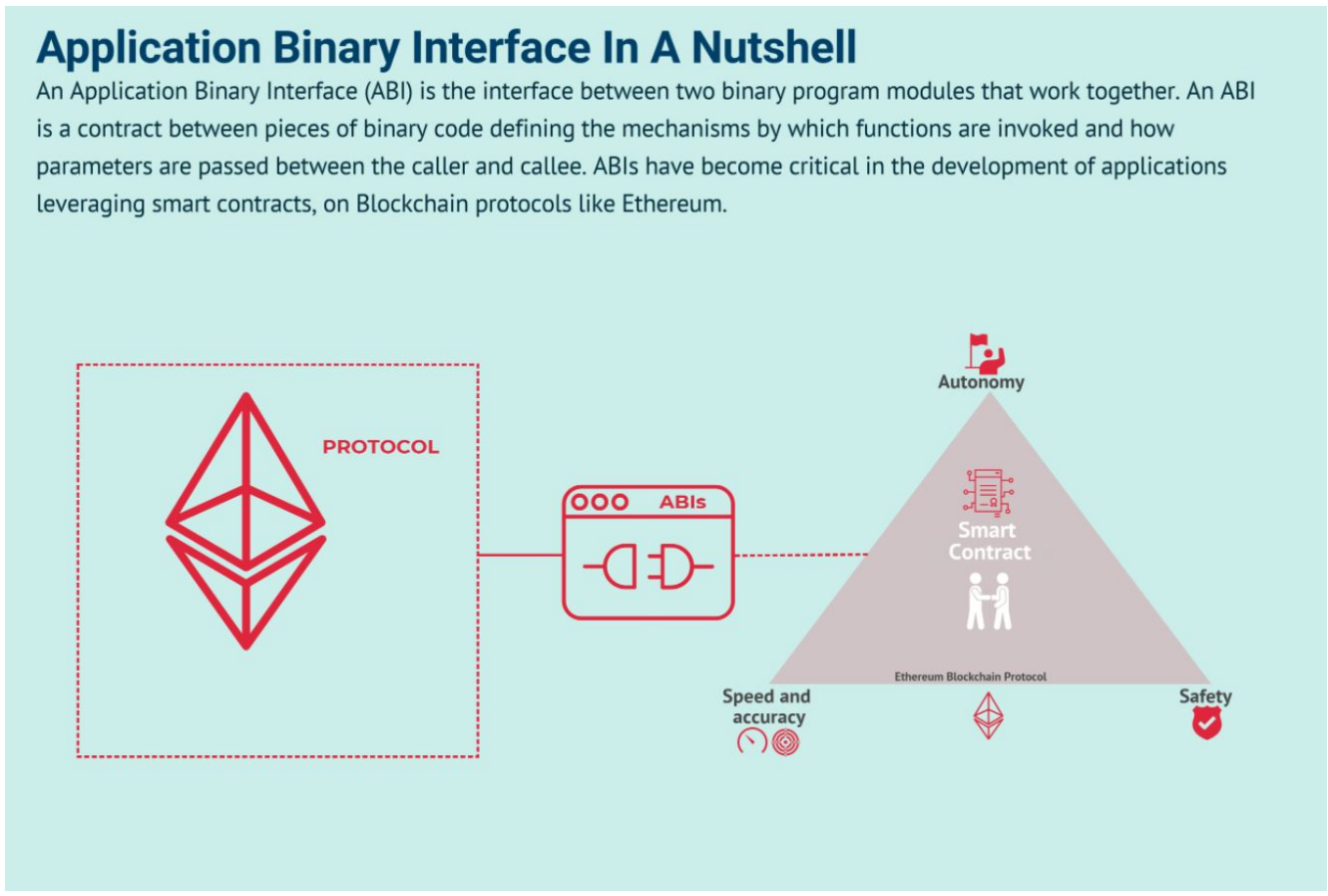
   Some Methods:

7. registerAccount would be a great one!
8. An onlyRegistered() modifier - how will you implement it?
9. getAccountInfo(), should probably be available to onlyRegistered
10. transfer(...), should probably be available to onlyRegistered
11. withdrawl(...), should probably be available to onlyRegistered
12. recieve()

You're welcome to add any other fun methods and properties you'd like to experiment with. Feel free to show off a bit, but please do cover the above bases.

Once you've written the above and tested it out a bit, have a TA or group member look it over before moving to the next part.

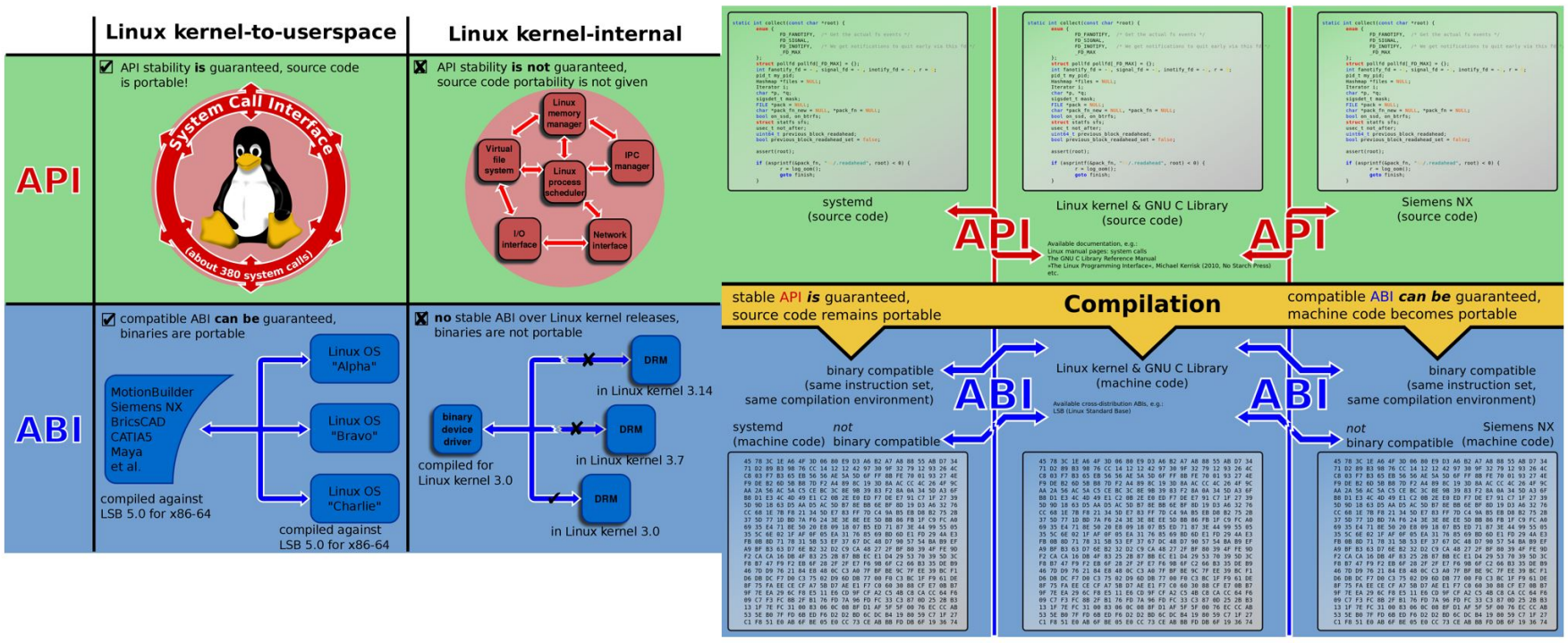
---



## Step 2. Generating our ABI

What is this ABI term doing here? What does it mean? Where did it come from?

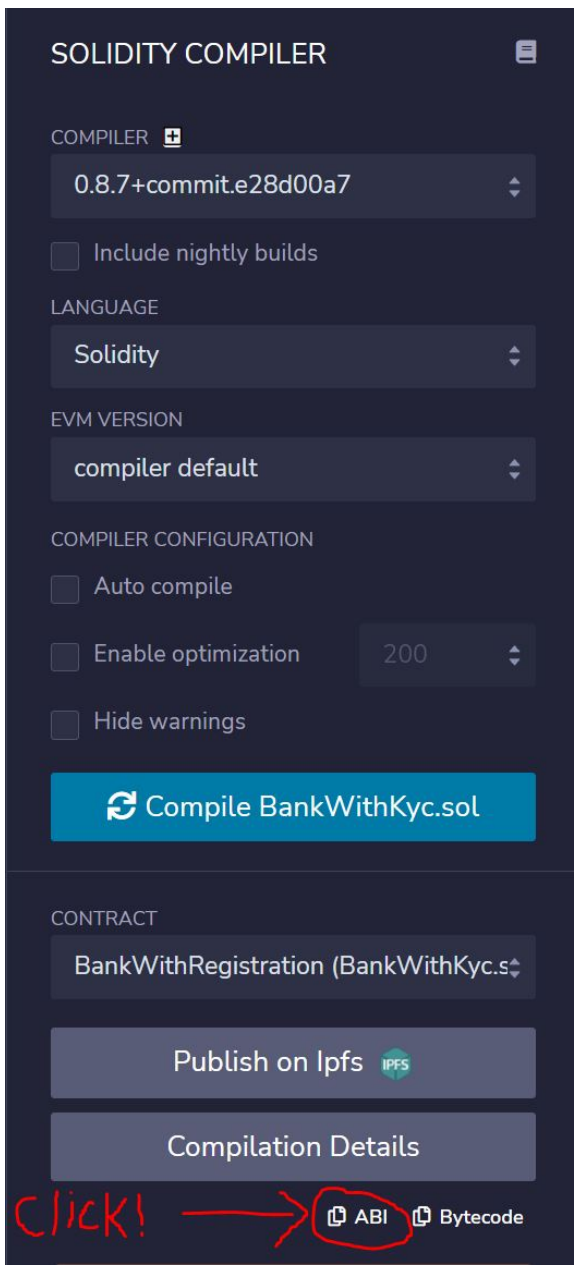
ABI stands for **Application Binary Interface.** They're finicky implementation specific specifications for how programs can talk to each other on a very granular basis. ABIs are not unique to Solidity. We find them in many other contexts in programming. For example see this fun Info Graphic below contrasting system calls with APIs with ABIs with Linux as an example. Thankfully things are a little gentler when working with virtual machines than they would be otherwise.




We can use our Smart Contracts ABI to communicate with Smart Contracts on EVM/blockchain from Web3.

We'll leave that fun part for later on. Your instructor in the meantime can show you how ABIs can be used to import and work with contracts found on the EVM for which we do not have or know the source code for, but which we may have an ABI from the authors.

We'll need the ABI from our contract later on in the week. So, after you're finished with all the changes you'd like to make on your contract above, if you would kindly compile it from the compiler tab and save a copy of the ABI into a .JSON file named after your clever smart contract name - that would be great!




---

## Final Step - Putting it all together.

If you haven't done so already, create a new GitHub repo for this week's assignment. Let's add our files (one .sol source file and one .JSON abi file), and add them to a new >**branch**< called "part-1". As you'll see, branches will play a star role in this week's work, and they're super-important on the job and in real life for getting the most out of Version Control.

Place the link to your GitHub repo in the usual spot and you're all done.

Great work!

- [ ] Grab your file from last week.
- [ ] Add the properties.
- [ ] Add the methods.
- [ ] Test it out a bit in Ganache.
- [ ] Add anything you'd like to experiment with.
- [ ] Test it some more.
- [ ] Generate the ABI.
- [ ] Make a new GitHub Repo
- [ ] Create a new Branch called "part-1".
- [ ] Place your files here.
- [ ] Paste the link below.
- [ ] All done! Congratulations and great work. We hope you had fun :D