CS 168: Blockchain and Cryptocurrencies

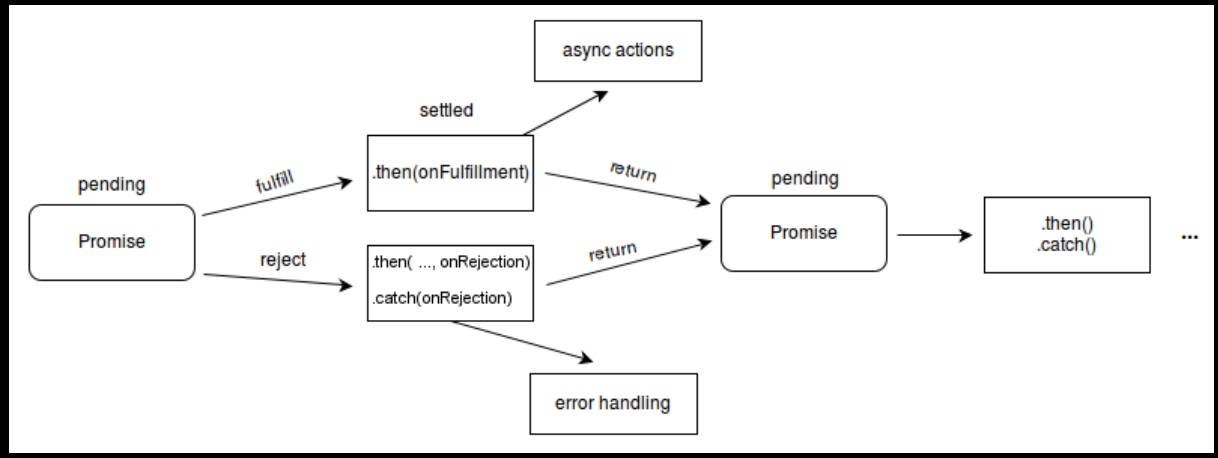
Decentralized Apps (DApps)

Prof. Tom Austin
San José State University

JavaScript Promises

- Promise: an object that *may* produce a value in the future.
- Similar to listeners, but
  - can only succeed or fail once
  - callback is called even if event took place earlier
- Simplify writing asynchronous code

Promise States



- *Pending*: neither fulfilled nor rejected.
- *Settled*, which is either:
  - *Fulfilled*: operation completed successfully.
  - *Rejected*: operation failed.
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

```
let fs = require('fs');
let p = new Promise((resolve, reject) => {
  //{ key: 'hello' }
  let f = fs.readFileSync('./test.json');
  resolve(f);
});

p.then(JSON.parse)
 .then((res) => res.key)
 .then((res) => console.log(res + " world!"));
```

```
let fs = require('fs');
let p = new Promise((resolve, reject) => {
  //{ key: 'hello' }
  let f = fs.readFileSync('./test.json');
  resolve(f);
});

p.then(JSON.parse)
 .then((res) => res.key,
       (err) => console.error(err)))
 .then((res) => console.log(res + " world!"));
```

```
let fs = require('fs');
let p = new Promise((resolve, reject) => {
  //{ key: 'hello' }
  let f = fs.readFileSync('./test.json');
  resolve(f);
});

p.then(JSON.parse)
 .then((res) => res.key,
 .then((res) => console.log(res + " world!"))
 .catch((err) => console.error(err));
```

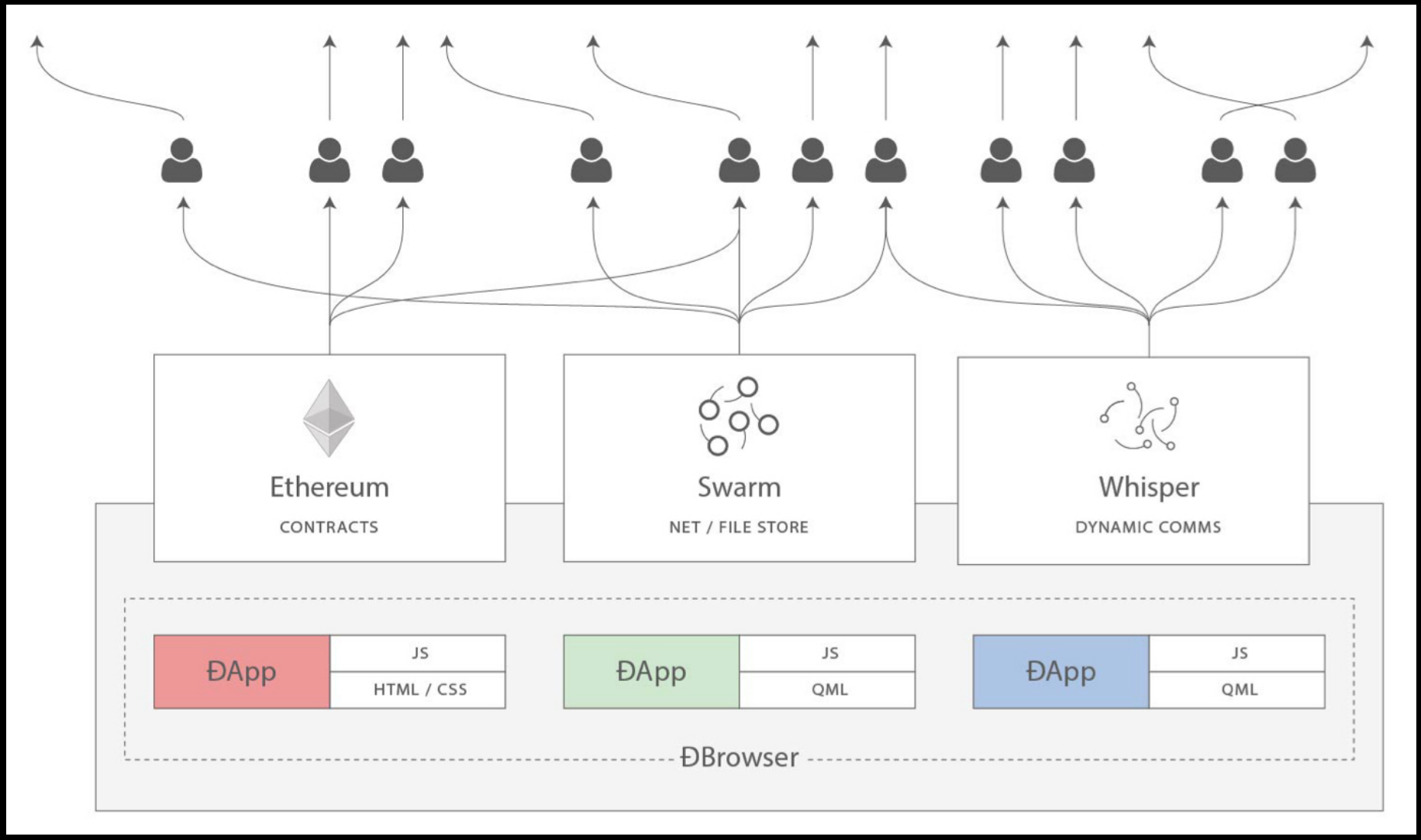
```
let fs = require('fs');
let p = new Promise((resolve, reject) => {
  //{ key: 'hello' }
  let f = fs.readFileSync('./test.json');
  resolve(f);
});

p.then(JSON.parse)
 .then((res) => res.key,
 .then((res) => console.log(res + " world!"))
 .catch((err) => console.error(err))
 .finally(() => console.log("All done!"));
```

Evolution of the Web

- Web 1.0
  - Few content creators
- Web 2.0
  - User-generated content
  - Social media
  - Applications still centralized
- Web 3.0
  - Decentralized applications

What Makes a DApp?

- Smart contracts
  - Business-logic, application state, etc.
  - Replace server-side layer
- Frontend
  - HTML, JavaScript, etc.
  - Web3.js – communication with smart contract
- Other components
  - Storage: IPFS or Swarm
  - Messaging: Whisper



Taken from *Mastering Ethereum*

Advantages of a DApp

- Advantages:
  - Censorship-resistance
  - Transparency
  - Resilience
- Disadvantages:
  - Pretty much everything else

Solidity Compiler Example

```
contract Vote {
  uint256 greenVotes; uint256 redVotes;
  function vote(bool votedGreen) public {
    if (votedGreen) greenVotes += 1;
    else redVotes += 1;
  }
  function getRedVotes() view public
                        returns (uint256) {
    return redVotes;
  }
  function getGreenVotes() view public
                          returns (uint256) {
    return greenVotes;
  }
}
```

DApp Development Process

1. Compile Solidity code
2. Run Ganache for testing
3. Deploy code
4. Interact through web3.js

DApp Development Process

1. **Compile Solidity code**
2. Run Ganache for testing
3. Deploy code
4. Interact through web3.js

Solidity Compiler

- Produces:
  - EVM bytecode
  - Application Binary Interface (ABI)
- Usage:
  ```
  $solcjs --bin --abi ContractFile.sol
  ```
- Installed through npm.
  - NOTE: There are other solidity compilers.

ABI

- Specifies interface for other tools to use.
- Produced by Solidity compiler.

Compiling `Vote.sol`

```
$ ls Vote*
Vote.sol
$ solcjs --abi --bin Vote.sol
$ ls Vote*
Vote.sol Vote_sol_Vote.abi
Vote_sol_Vote.bin
```

Vote_sol_Vote.abi

[{"inputs":[], "name":"**getGreenVotes**",
"outputs":[{"internalType":"uint256",
"name":"","type":"uint256"}],
"stateMutability":"view","type":"function"},
{"inputs":[],"name":"**getRedVotes**","outputs":
[{"internalType":"uint256","name":"","type":
"uint256"}],
"stateMutability":"view","type":"function"},
{"inputs":
[{"internalType":"bool","name":"votedGreen",
"type":"bool"}],"name":"**vote**","outputs":
[],"stateMutability":"nonpayable","type":"fu
nction"}]

Vote_sol_Vote.abi


```
[{"inputs":[], "name":"getGreenVotes",
"outputs":[{"internalType":"uint256",
"name":"","type":"uint256"}],
"stateMutability":"view","type":"function"},
{"inputs":[],"name":"getRedVotes","outputs":
[{"internalType":"uint256","name":"","type":
"uint256"}],
"stateMutability":"view","type":"function"},
{"inputs":
[{"internalType":"bool","name":"votedGreen",
"type":"bool"}],"name":"vote","outputs":
[],"stateMutability":"nonpayable","type":"fu
nction"}]
```

Source code

```
function vote(bool votedGreen) public {
    if (votedGreen) greenVotes += 1;
    else redVotes += 1;
  }
```

ABI

```
{"inputs":
  [{"internalType":"bool",
    "name":"votedGreen",
    "type":"bool"}],
 "name":"vote",
 "outputs":[],
 "stateMutability":"nonpayable",
 "type":"function"}
```

Source code

```
function vote(bool votedGreen) public {
    if (votedGreen) greenVotes += 1;
    else redVotes += 1;
  }
```

ABI

```
{"inputs":
  [{"internalType":"bool",
    "name":"votedGreen",
    "type":"bool"}],
 "name":"vote",
 "outputs":[],
 "stateMutability":"nonpayable",
 "type":"function"}
```

Source code

```
function vote(bool votedGreen) public {
    if (votedGreen) greenVotes += 1;
    else redVotes += 1;
  }
```

ABI

```
{"inputs":
  [{"internalType":"bool",
    "name":"votedGreen",
    "type":"bool"}],
 "name":"vote",
 "outputs":[],
 "stateMutability":"nonpayable",
 "type":"function"}
```

DApp Development Process

1. Compile Solidity code
2. **Run Ganache for testing**
3. Deploy code
4. Interact through web3.js

Ganache

- "One-click blockchain"
  - Blockchain emulator
- Part of Truffle tool suite
- Install through npm
- Creates 10 accounts with 100 ether

```
$ ./node_modules/.bin/ganache-cli
Available Accounts
==================
(0) 0xEb43… (100 ETH)
(1) 0xa869… (100 ETH)
… 10 addresses


Private Keys
==================
(0) 0x9fcf…
(1) 0x6818…
… 10 keys


… <Lots more stuff>


Listening on 127.0.0.1:8545
```

Leave the process running!

DApp Development Process

1. Compile Solidity code
2. Run Ganache for testing
3. **Deploy code**
4. Interact through web3.js

Web3.js

- Interacts with Smart Contract
- Works with either node or browser
- WARNING: a little buggy

Deploy.js

- Script uses Web3.js to deploy contract
- Choose an account from Ganache to use
- **<Review code in class>**

```
$
```

```
$ ./node_modules/.bin/ganache-cli
Available Accounts
==================
(0) 0xEb43… (100 ETH)
…
Listening on 127.0.0.1:8545
```

```
$ node deploy.js Vote_sol_Vote 0xEb43…
Contract address: 0xCD1F909f2F56…
$
```

```
$ ./node_modules/.bin/ganache-cli
Available Accounts
==================
(0) 0xEb43… (100 ETH)
…
Listening on 127.0.0.1:8545
```

```
$ node deploy.js Vote_sol_Vote 0xEb43…
Contract address: 0xCD1F909f2F56…
$
```

```
$ ./node_modules/.bin/ganache-cli
Available Accounts
==================
(0) 0xEb43… (100 ETH)
…
Listening on 127.0.0.1:8545
eth_sendTransaction
  Transaction: 0x2279…
  Contract created: 0xcd1f909f2f56…
  … <more stuff>
```

DApp Development Process

1. Compile Solidity code
2. Run Ganache for testing
3. Deploy code
4. **Interact through web3.js**

Interacting with node

```
$ node vote.js 0xcd1f… 0xEb43…
{ transactionHash: '0x1ae5…',
  …
}
$ node getVotes.js 0xcd1f… 0xEb43…
Green has 5 votes.
Red has 1 votes.
$
```

vote.js

```
const Web3 = require('web3');
const fs = require('fs')

let url = "http://localhost:8545";
let web3 = new Web3(url);
let cont = 'Vote_sol_Vote';

let contAddr = process.argv[1];
let from = process.argv[2];

let abi = JSON.parse(fs.readFileSync(
    `${cont}.abi`).toString());
let contract = new web3.eth.Contract(abi);
contract.options.address = contAddr;

// Voting green
contract.methods.vote(true).send({
  from: from,
}).then(console.log);
```

getVote.js

```
const Web3 = require('web3');
const fs = require('fs')
let url = "http://localhost:8545";
let web3 = new Web3(url);
let contractName = 'Vote_sol_Vote';
let contractAddress = process.argv[1];
let from = process.argv[2];

let abi = JSON.parse(fs.readFileSync(
    `${contractName}.abi`).toString());
let contract = new web3.eth.Contract(abi);
contract.options.address = contractAddress;

// Read-only -- no ether required.
contract.methods.getGreenVotes().call((_, greenVotes) => {
  console.log(`Green has ${greenVotes} votes.`);
});
contract.methods.getRedVotes().call((_, redVotes) => {
  console.log(`Red has ${redVotes} votes.`);
});
```

Using Web3.js with Webpage

Preamble for `index.js`

```
let web3 = new Web3(new
Web3.providers.HttpProvider(
      "http://localhost:8545"))
let account;
web3.eth.getAccounts().then((f) => {
  account = f[0];
});
let contract = new web3.eth.Contract(
      JSON.parse(/** ABI STRING **/));
contract.options.address = " 0xcd1f…";
```

Functions for `index.js`

```
function vote(votedGreen) {
  contract.methods.vote(votedGreen)
     .send({from: account}).then(updateResults);
}
function updateResults() {
  contract.methods.getGreenVotes().call().then((f) => {
    $("#greenVotes").html(f);
  });
  contract.methods.getRedVotes().call().then((f) => {
    $("#redVotes").html(f);
  });
}
updateResults();
```

index.html body

```
<h1>Vote Green or Red</h1>
<div id="results">
  <font size="+5">
    <font color="green">
      <span id="greenVotes">0</span>
    </font>
    &dash;
    <font color="red">
      <span id="redVotes">0</span>
    </font>
  </font>
</div>
<input id="greenButton" type="button" value="Vote Green"
       onclick="vote(true)" />
<input id="redButton" type="button" value="Vote Red"
       onclick="vote(false)" />
```

Note on Using Strings

- Many applications use `bytes32` rather than `string`.
  - Lower gas price.
- Web3.js offers utility functions to convert:
  - `web3.utils.hexToAscii` converts bytes to a string
  - `web3.utils.asciiToHex` converts a string to bytes

Lab: Auction DApp

- Build a DApp for an auction
- Details in Canvas