



# ethereum

## vienna

Workshop  
From Contract to DApp



# Workshops

Workshop #1: Contract Development for Beginners

Requirements: Basic Understanding of Ethereum

Solidity Basics

Workshop #2: From Idea to Contract

Requirements: Basic Understanding of Solidity

Mapping the real world to ethereum concepts

Advanced Solidity

**Workshop #3: From Contract to DApp**

**Requirements: Basic Understanding of Solidity, HTML/JS, node.js**

**Interfacing with Ethereum using web3.js**

**Auxiliary Technologies: IPFS, Whisper and Swarm**

# Agenda

1. Intro to the Ethereum Stack
2. About the workshop
3. The web3.js library
- 4. Implementing a simple frontend**
5. Swarm / IPFS
- 6. Deploying to IPFS**
7. Standard functionality in DApps
8. Open discussion

# Requirements

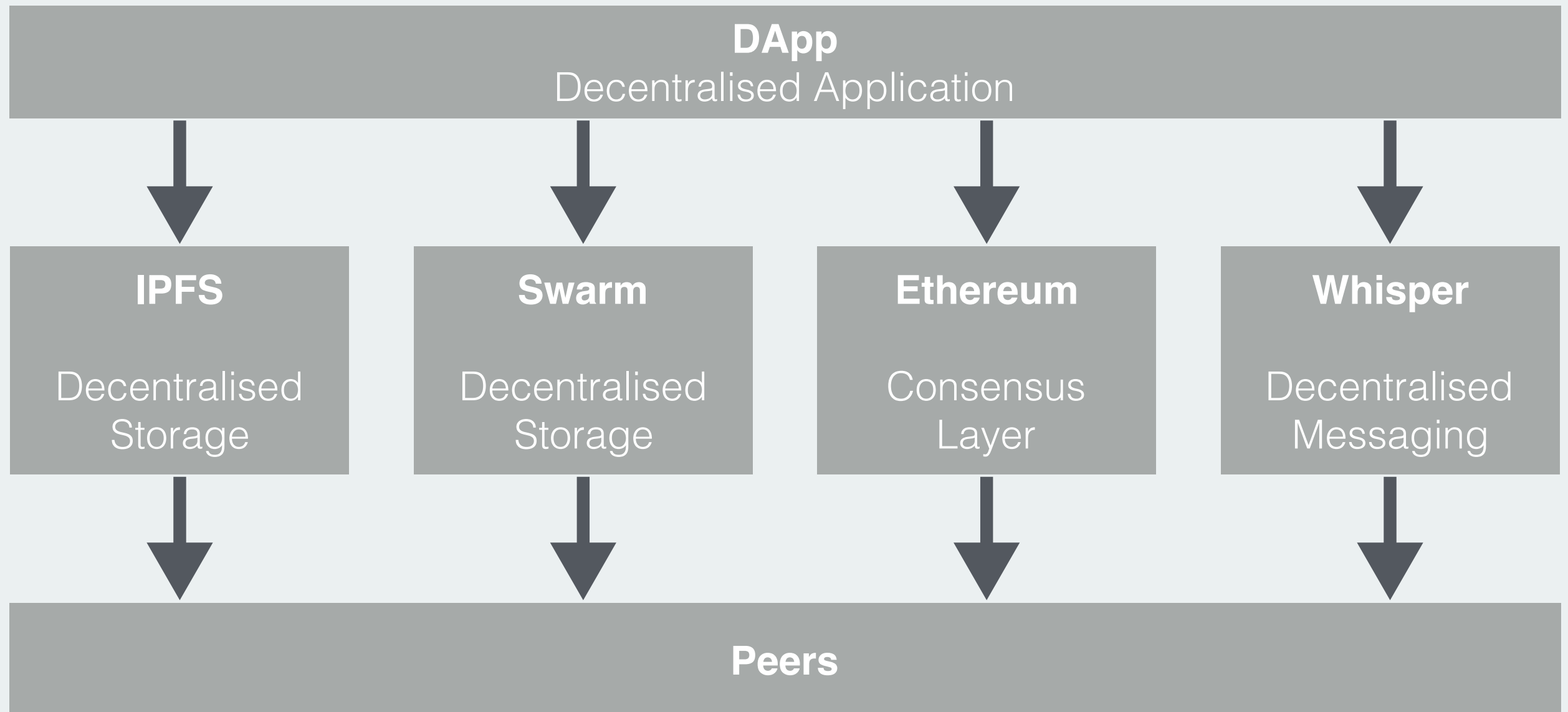
Requirements:

Knowledge of basic solidity (WS1)

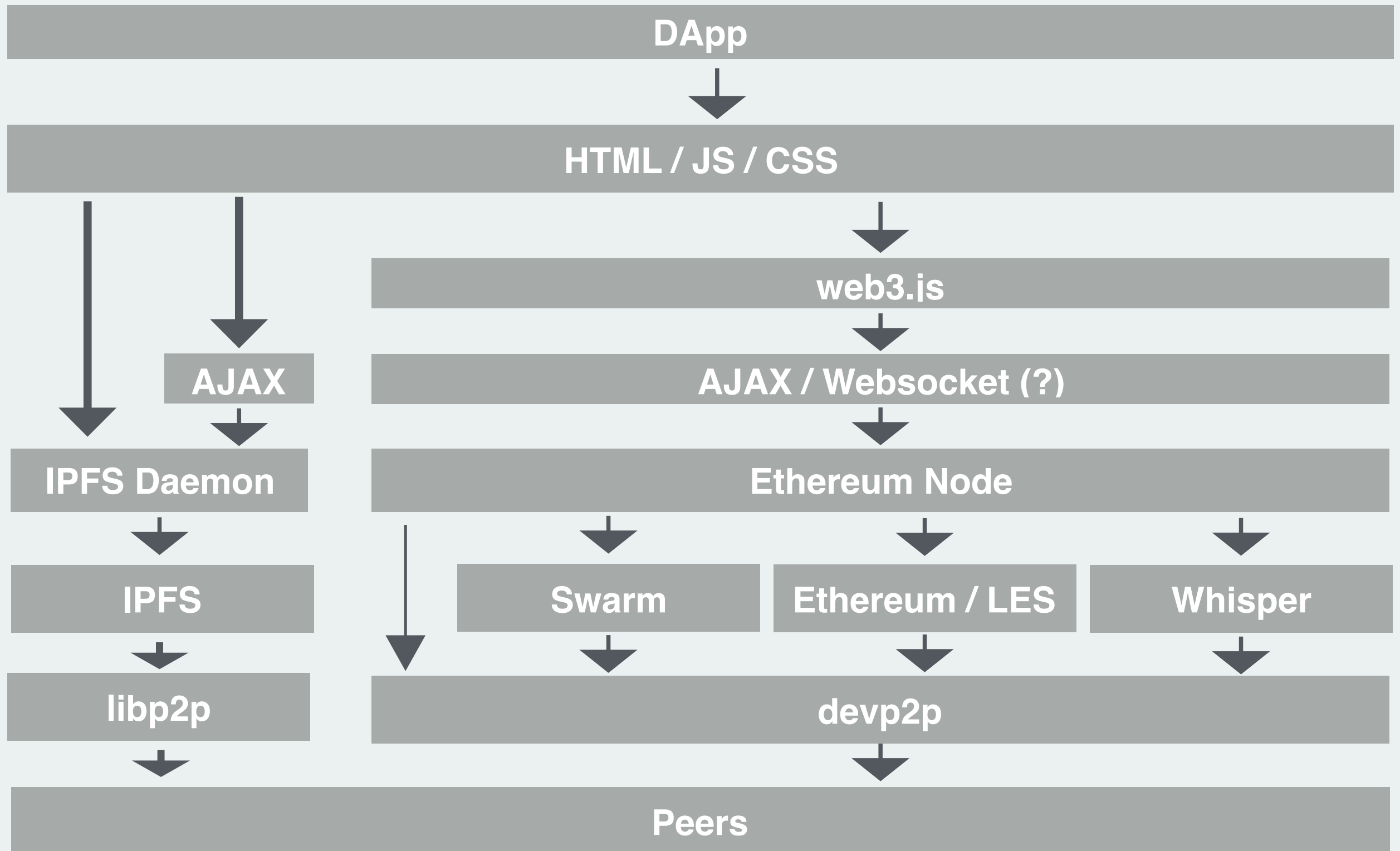
geth

Experience with HTML/JS

# The Ethereum Stack



# The Ethereum Stack



# The Ethereum Stack

Is also pretty much the entire stack

using anything else might introduce central points

devp2p not covered not yet directly accessible

In the future everything should be packaged with mist

# Ethereum Browser

## **Mist**

Standalone browser for ethereum

## **Metamask**

Browser plugin

both handle account management / unlocking

otherwise accounts need to be manually unlocked (in geth)



# About the workshop

No constraints on what you can use

No recommended Framework

Recommendation:

get web3.js with npm, bundle with webpack

boilerplate available

# geth

start geth with --dev for developer mode

private testchain

use --rpc to enable rpc

set the CORS domain with --rpccorsdomain '\*'

also the ipc path --ipcpath /path/to/standard/datadir/  
geth.ipc to be able to attach other instances

```
geth --dev --rpc --rpccorsdomain '*' --ipcpath /path/to/  
standard/datadir/geth.ipc console
```

# web3.js

Javascript Bindings for Ethereum

Works with all standard clients

Can connect with

- rpc

- websocket (probably not)

Same library as in geth console

web3.js is great

web3.js sucks

# web3.js

currently undergoing massive rework

next major version comes with

- promises
- event emitter

# web3.js installation

`npm install web3`

if used with webpack => json-loader required

`require('web3')`, regardless of used method

# web3.js configuration

```
var Web3 = require('web3') /* es6: import Web3 from 'web3' */  
  
/* create a object representing a connection to a node */  
  
var web3 = new Web3()  
  
/* set the provider to the default connection - port 8545 */  
web3.setProvider(new Web3.providers.HttpProvider('http://localhost:8545'))  
  
/* ready to use */  
  
  
/* for reuse as a module */  
module.exports = web3 /* es6: export default web3 */
```



# web3.js configuration

In Metamask / Mist: web3 already provided. Own web3.js, but keep the provider

```
window.addEventListener('load', function() {  
  if (typeof web3 !== 'undefined') {  
    web3 = new Web3(web3.currentProvider);  
  } else {  
    web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));  
  }  
  start()  
})
```

# web3 object structure

consists of several objects

Available in RPC:

**web3.eth**: interaction with ethereum

**web3.shh**: interaction with whisper

**web3.version**: version information about node

**web3.net**: information about peers

# web3 object structure

Not available in RPC:

**web3.personal**: creating and unlocking accounts

**web3.txpool**: information about pending transactions

**web3.admin**: admin interface to geth

**web3.miner**: mining interface

# web3 bignumber

Ethereum deals with huge numbers ( $2^{256}-1$ )

JS Integers too small

=> web3.js uses bignumber.js library

available through **web3.BigNumber**

# web3 bignumber

```
var a = new BigNumber(9)
```

```
=> { [String: '9'] s: 1, e: 0, c: [ 9 ] }
```

```
a.toString() == '9'
```

Use bignumber functions for arithmetic!

```
> a = new BigNumber(9)
9
> a + 3
"93"
> a.plus(3)
12
```

# web3 bignumber

conversion utilities available on web3 object

web3.toDecimal

web3.toBigNumber

web3.toHex

and many more

# web3 utilities

toUtf8, fromUtf8

toAscii, fromAscii

toWei, fromWei (to convert between ether and wei)

isAddress: check if valid address

isChecksumAddress: check if valid cs address

toChecksumAddress: converts address to cs address

# web3.js sync / async

most calls are synchronous by default

pass a callback as the last argument to make it async

`eth.getBlock(0)`

=> `eth.getBlock(0, function (err, block) { /**/ })`

`eth.accounts`

=> `eth.getAccounts(function (err, accounts) { /**/ })`



# web3.js sync / async

metamask demands async calls!

only a few selected sync calls allowed:

eth.accounts

eth.coinbase

...

# web3.personal

Account creation and unlocking

```
[> personal.newAccount()  
[Passphrase:  
[Repeat passphrase:  
"0x612cf3f4ec6a3e3d82d8da4e4820be1c8cadbbea"
```

Unlocks an account for a certain amount of time

```
[> personal.unlockAccount(eth.accounts[0])  
Unlock account 0x58ccc0432f9bd47d05079adda1b5199bb14144ab  
[Passphrase:  
true
```

console only. Mist / Metamask use different accounts

# web3.eth

accounts

```
> eth.accounts  
["0x58ccc0432f9bd47d05079adda1b5199bb14144ab"]
```

getBalance

```
> eth.getBalance(eth.accounts[0])  
9.79454623546e+21
```

```
> web3.fromWei(eth.getBalance(eth.accounts[0]))  
9859.54623546
```

# web3.eth

blockNumber

```
> eth.blockNumber  
1983
```

For metamask compatibility:

```
> eth.getBlockNumber(function(err, number) { /* */ })
```

# web3.eth

[illegible]

# web3.eth - sending value

```
> eth.sendTransaction({ from: eth.accounts[0], to: eth.accounts[0], value: web3.toWei(100) })
```

transactionObject has many optional fields:

from: sender (defaults to eth.defaultAccount)

to: destination (can be omitted in contract creation)

value: number of wei to be sent

gas: gaslimit

gasPrice (defaults to eth.gasPrice)

data: function arguments or contract code

# web3.eth - sending value

Sending will fail if sender account is not unlocked

In geth manual unlock necessary with  
`personal.unlockAccount(address)`

In Mist / Metamask, there will be a popup



# web3.eth

```
> eth.getTransaction('0x3a6f3359b1b1cf1a9abf9d100d542b8763dc38d314c159f74ddfdd437ec07bd6')  
{  
  blockHash: "0x9d0befcfc815de2126d387187cb2b198ae7fe98cef234ea49c43e8bb37f5255f",  
  blockNumber: 2086,  
  from: "0x58ccc0432f9bd47d05079adda1b5199bb14144ab",  
  gas: 90000,  
  gasPrice: 20000000000,  
  hash: "0x3a6f3359b1b1cf1a9abf9d100d542b8763dc38d314c159f74ddfdd437ec07bd6",  
  input: "0x",  
  nonce: 10,  
  r: "0xcceac75ab9440aaa40b16c7e01bb97dd01fb0e7535163507ed19e37d12596b05",  
  s: "0x1c8dc888dd8afd8e2e0d08160627ed4494267428c3efc84db6ec4d3d03fdd8c1",  
  to: "0x58ccc0432f9bd47d05079adda1b5199bb14144ab",  
  transactionIndex: 0,  
  v: "0x1c",  
  value: 1000000000000000000000  
}
```

blockHash / blockNumber may be null if not yet mined



# web3.eth

[illegible]

returns null if not yet mined

# web3.eth

'latest' filter fires for new blocks with their hash

```
[> var blockFilter = eth.filter('latest')
```

```
[> blockFilter.watch(function(err, blockHash) { })
```

```
[> blockFilter.stopWatching()
```

'pending' filter for recent pending transactions

can also be used for events (but there is a better api)

# solidity contract abi

JSON description of the interface of a contract

```
[{"constant":false,"inputs":  
[{"name":"a","type":"uint256"}],"name":"a","outputs":  
[{"name":"b","type":"uint256"}],"payable":false,"type":"function"},  
{"inputs":[{"name":"a","type":"uint256"},  
{"name":"b","type":"uint256"}],"payable":false,"type":"constructor"}]
```

web3.js can create JS objects that represents contract  
based on its abi

# web3.eth.contract

```
[> var Contract = eth.contract(abi)
```

Contract can now be used to create or reference contract instances

To reference an existing contract

```
[> var contract = Contract.at('0x6348ad8c4e02892cf011e222470055f076de1cb0')
```

# web3.eth.contract

To create a new contract:

```
> Contract.new(5,3, { data: bin }, function(err, contract) { })
```

First come all the constructor arguments

Afterwards the transactionObject (data is mandatory)

Callback will fire twice:

Once when transactionHash is know,

then when the transaction is included

# solidity contract abi

Calling functions on a contract is simply

```
contract.functionName(...arguments, txObject, cb)
```

'constant' functions will only be executed locally

other functions will result in a transaction

# solidity contract abi

For every event the contract object has a function  
`var filter = contract.MyEvent({v:27}, {fromBlock:0})`

```
filter.watch(function(err, result) { })
```

```
{
  "address": "0x39f0e4d64e2b935113b7d9d586dd47cdfef6a739",
  "args": {
    "v": "27"
  },
  "blockHash": "0x60268fb83dcfb7c5f40f27ab816860533f6a07651",
  "blockNumber": 2928,
  "event": "MyEvent",
  "logIndex": 0,
  "removed": false,
  "transactionHash": "0x834a4c0c0465e66d08976108b4c4329b442",
  "transactionIndex": 0
}
```

about reorganisations



Your turn:  
Implementing the frontend



# Whisper

Decentralised Messaging

Messages can be filtered by topics

Very flexible

- Messages can be encrypted

- Messages can be signed

- Broadcast

PoW for spam protection and priority

Not designed for real time communication





# Whisper

Not enabled by default

geth must be launched with --ssh

web3.js support through web3.ssh

-- dev already implies --ssh



# web3.shh

shh.newIdentity: create a new whisper identity

shh.post: send a whisper message

shh.filter: watch for messages

messages can have optional:

from, to, topics



# Swarm

Swarm (or IPFS)

Reverse Hash-table

Originator of source unknown

Low-latency

Incentivation model for storage

merged into geth (start with --bzz)

No web3.js support



# Intro to IPFS

Decentralised Hash Table

Maps hashes to files

Also support immutable directories (collection of files)

Hash size is not fixed. First bits of the hash indicate hashing algorithm. (so it's no uint256!)

No incentive structure yet (will be developed as a contract)

# IPFS from the CLI

ipfs binary from go-ipfs

or jsipfs from the javascript implementation

at least my jsipfs does not connect to anything

add files or directories:

ipfs add -r directory (-r for directories)

to read a file:

ipfs cat HASH

# IPFS Gateway

ipfs.io allows browsing hashes without an ipfs node

centralized

like onion.to and such



# IPFS from JS

Client library for Javascript

Server can also run in JS but doesn't have to (in dev)

```
var ipfs = window.IpfsApi()
```

```
ipfs.cat
```

```
ipfs.files.createAddStream
```

```
see ipfs-demo.js
```

Your turn:  
Deploy to IPFS

# Standard Functionality

In a DApp, much of the functionality of a regular app

must be implemented differently

does not scale as easily

might be less private by default

introduces trusted parties

# Standard Functionality

Computationally intensive calculation

Search

Privacy

Spam



1vieCmqYB3DE8StinXYBGGvgJ9hoXP1ib

# The End

0x50008dd0cc879e0341042f97541eb4870c9c8393

