Ethereum

The world computer

Dr. **Gavin Wood**
co-founder and lead developer, ethereum project

@gavofyork

Break it down

What, then why, then, maybe, How

Important Things to **Forget**

Proof of work

Ledger

Miner

Coin

Sign

Hash

Consensus

Curve

Currency

Block

Chain

Crypto

Bit

Transaction

ASIC

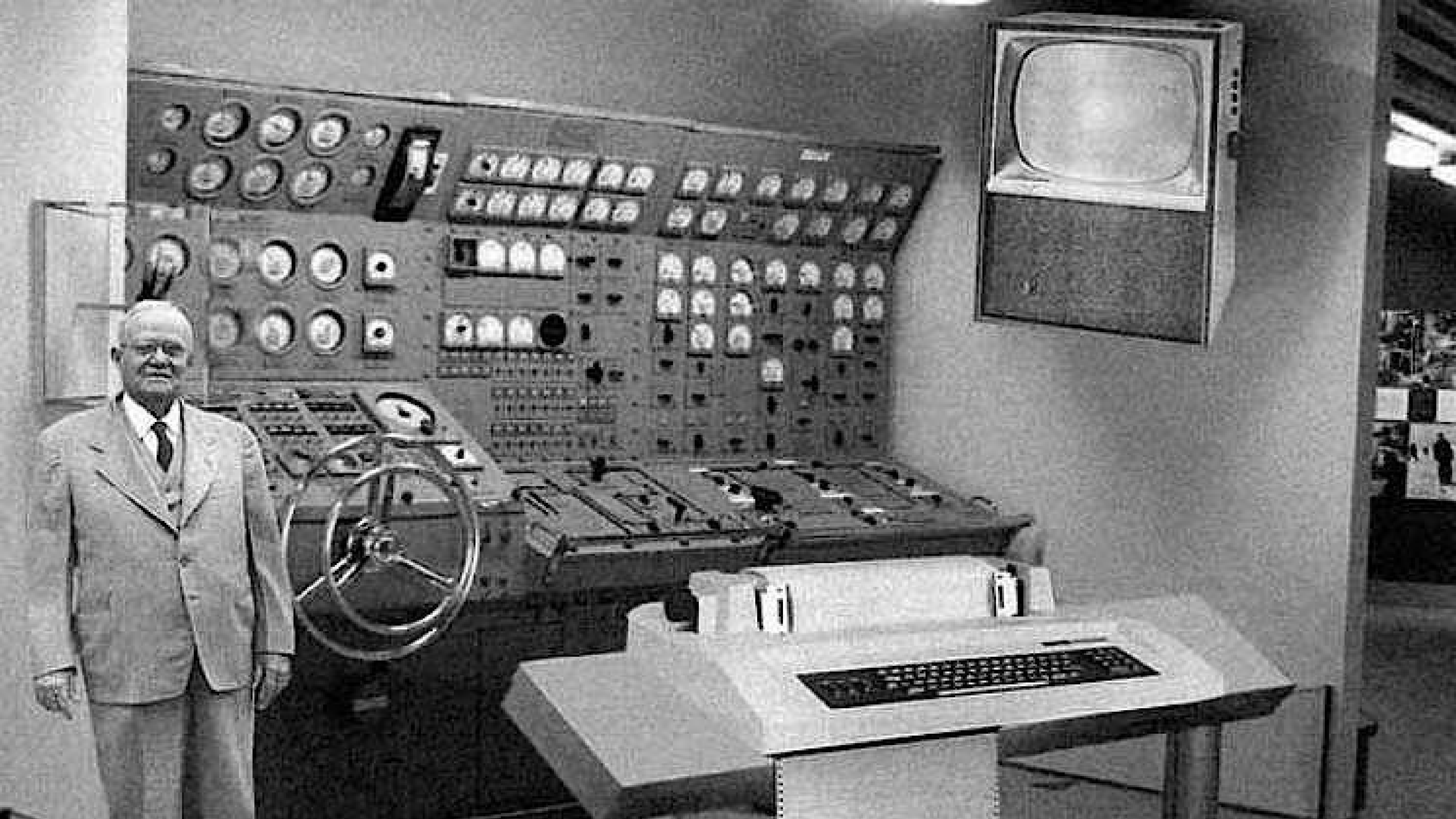
Proof of stake

Fork

Contract

What is it?

It’s a Computer, Silly!



It’s a Computer, Silly!

Slow

Code runs 5-100x slower that natively compiled

Expensive to use

Basic computation, memory and storage costs are ~1950s levels

Not always immediately decisive

Actions of last 60s may be reorganised

Sounds. Awesome.

Actually, it is.

Truly Global Singleton

One computer for the entire planet now and forever

Cannot Fail, be Stopped, be Censored

No authority, government or corporation behind it, resistant to attack

Ubiquitous

Where ever there’s Internet, there’s Ethereum

Natively Multi-User

Has as many accounts as is needed

Natively Object-Oriented

Encapsulation enforced in “virtual silicon”

Accessible

Where ever there’s Javascript, there’s Ethereum

Verifyable & Auditable

All code honoured now and forever

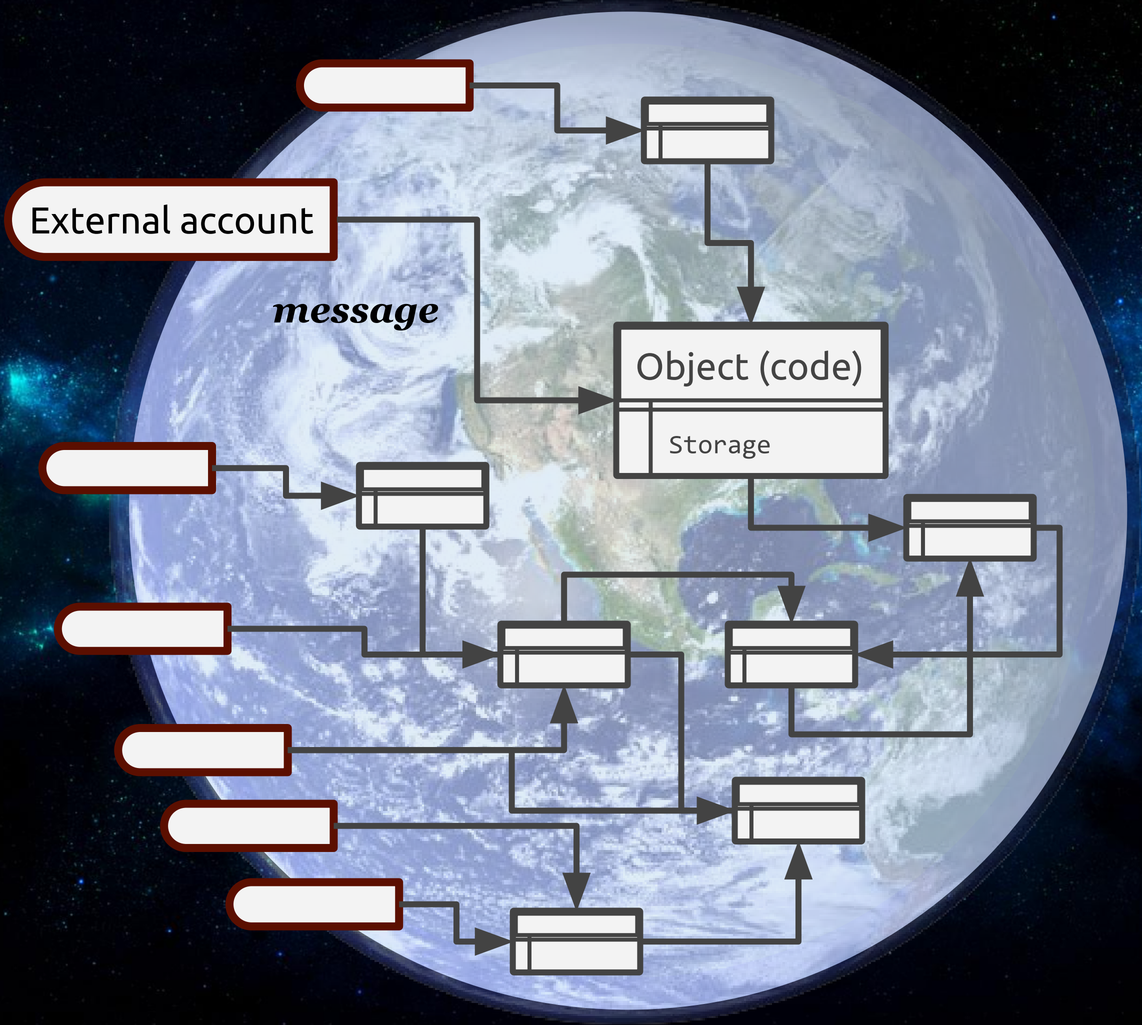
Simile

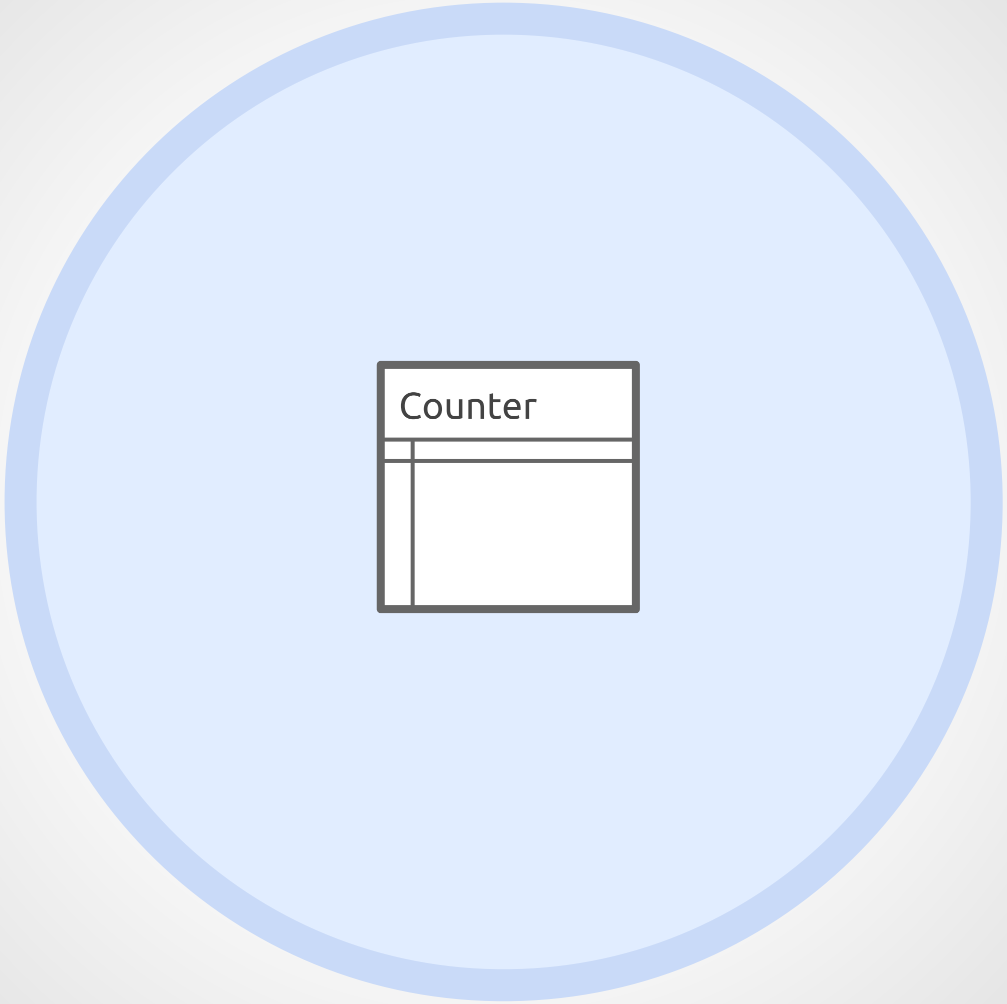
Internet is to communication

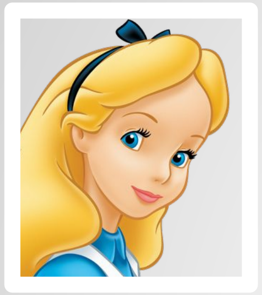
as

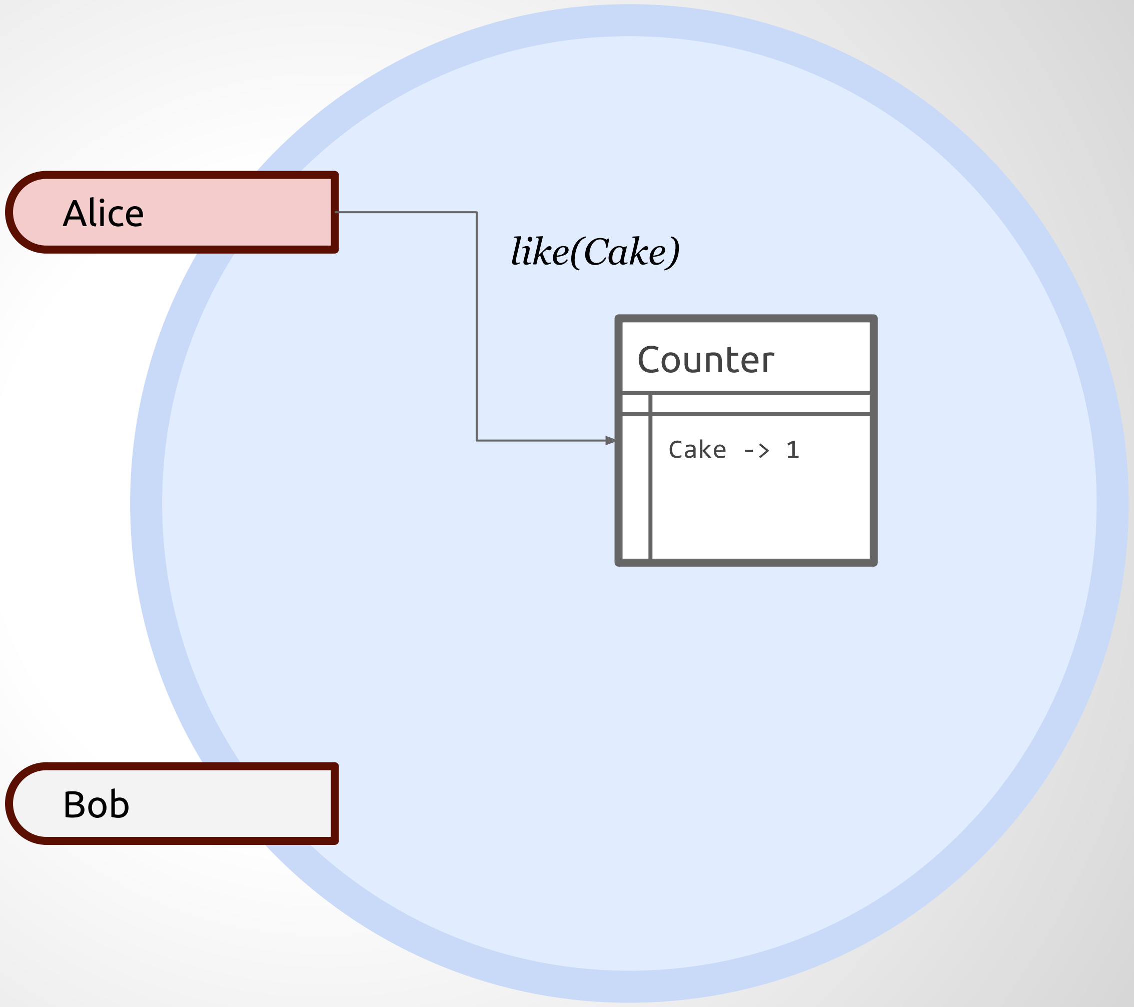
Ethereum is to computation

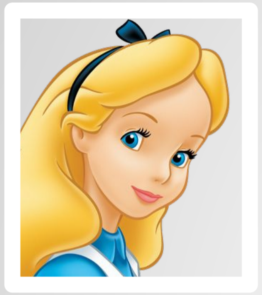
The **World** Computer

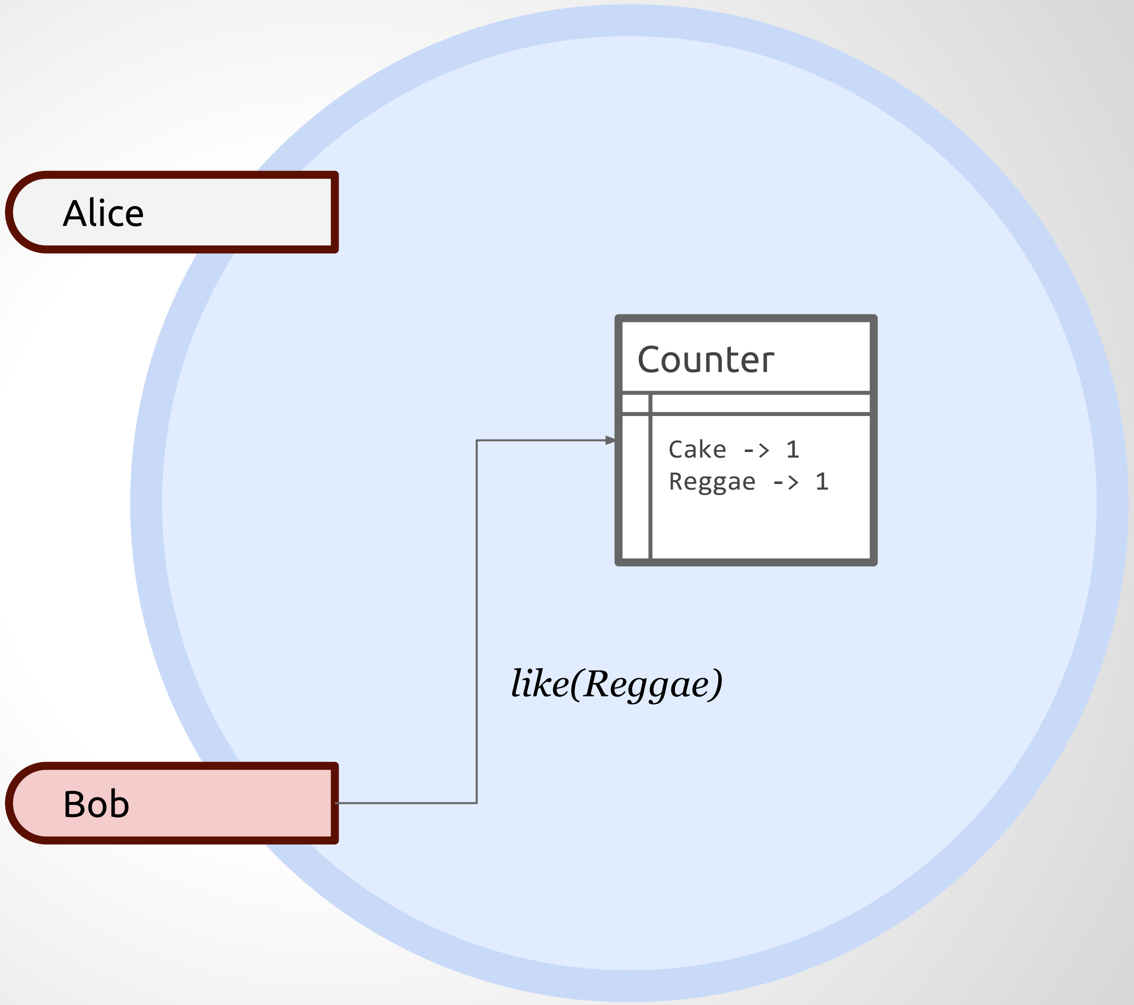


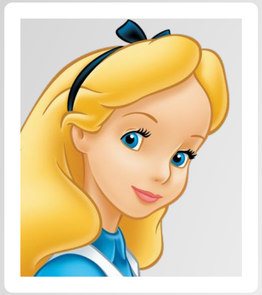

Counter

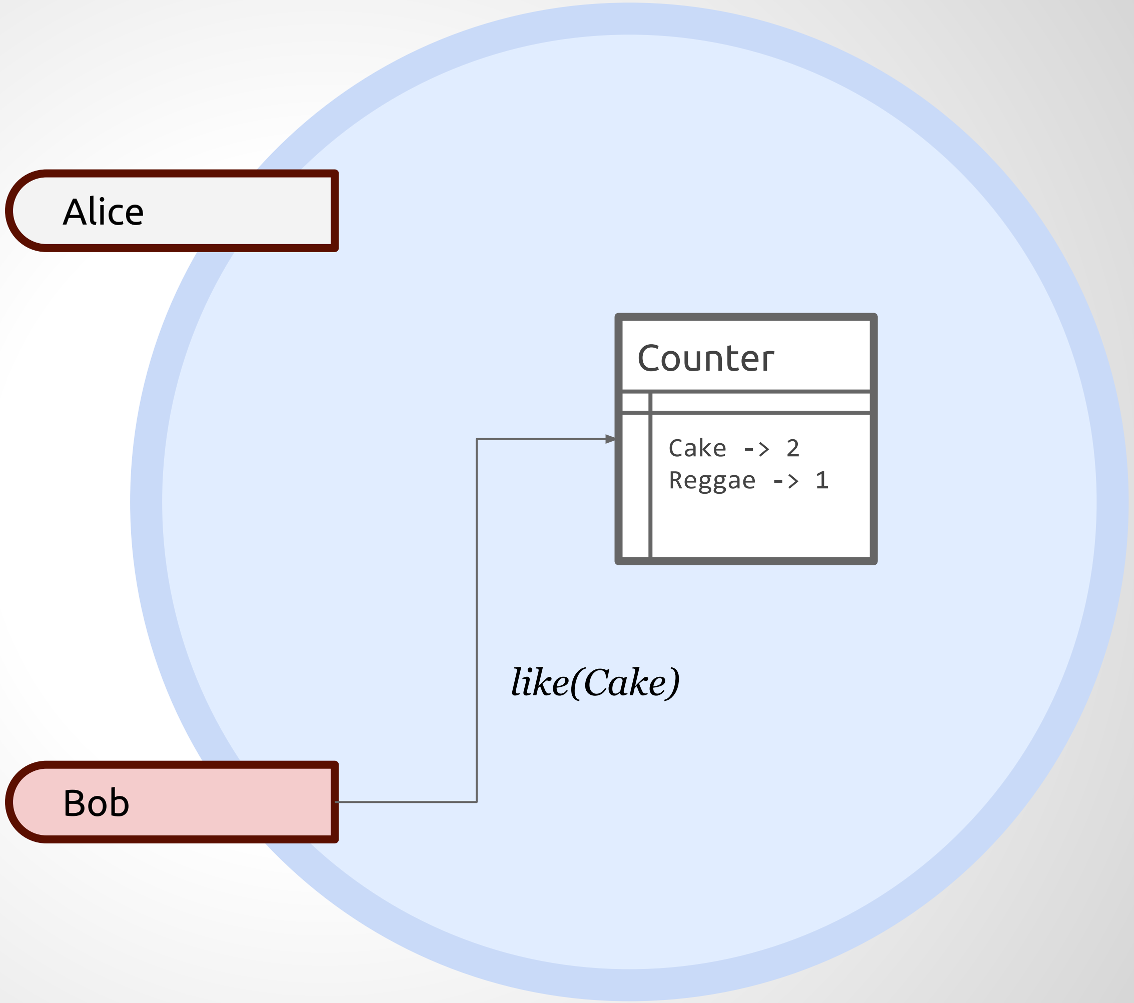



Alice
like(Cake)
Counter
Cake -> 1
Bob



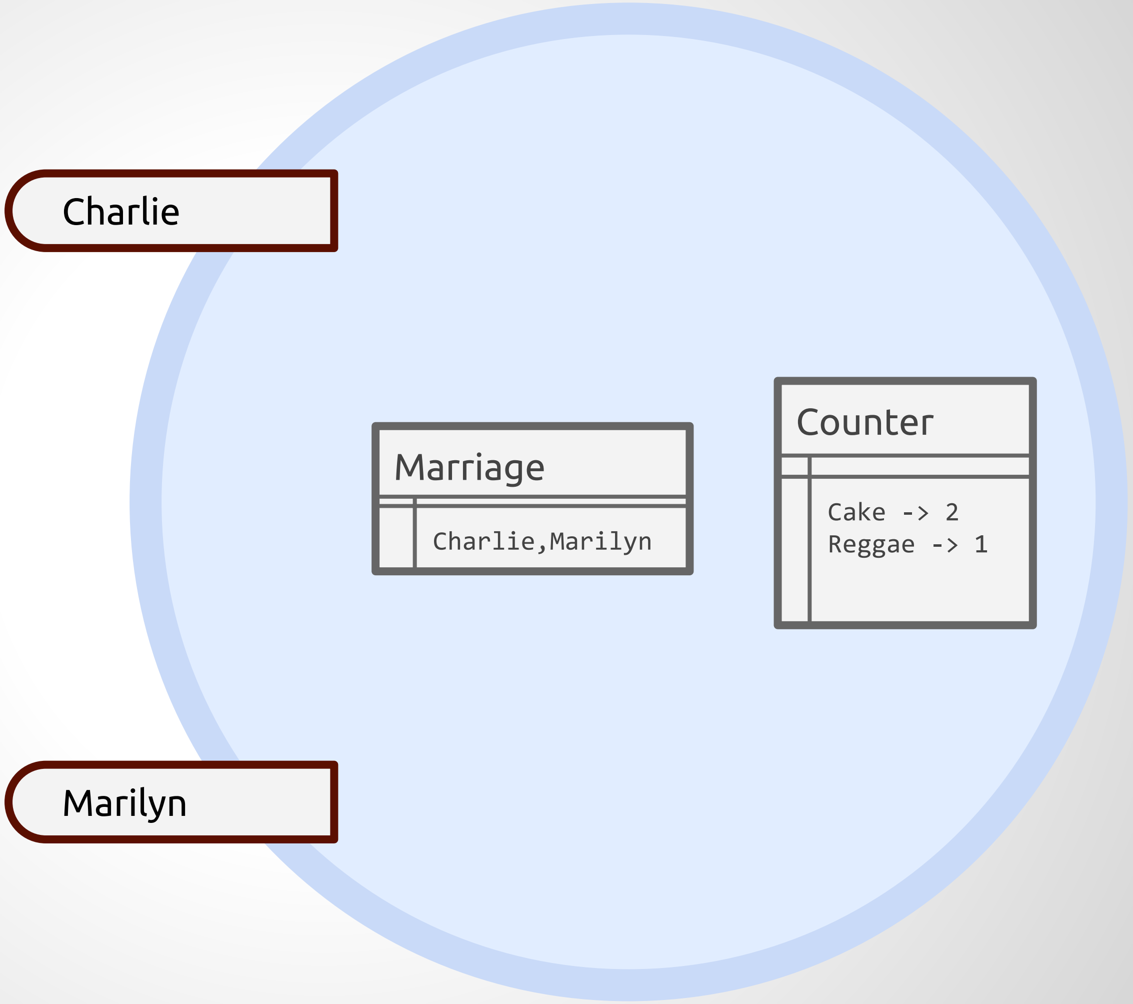

Alice
Counter
Cake -> 1
Reggae -> 1
like(Reggae)
Bob

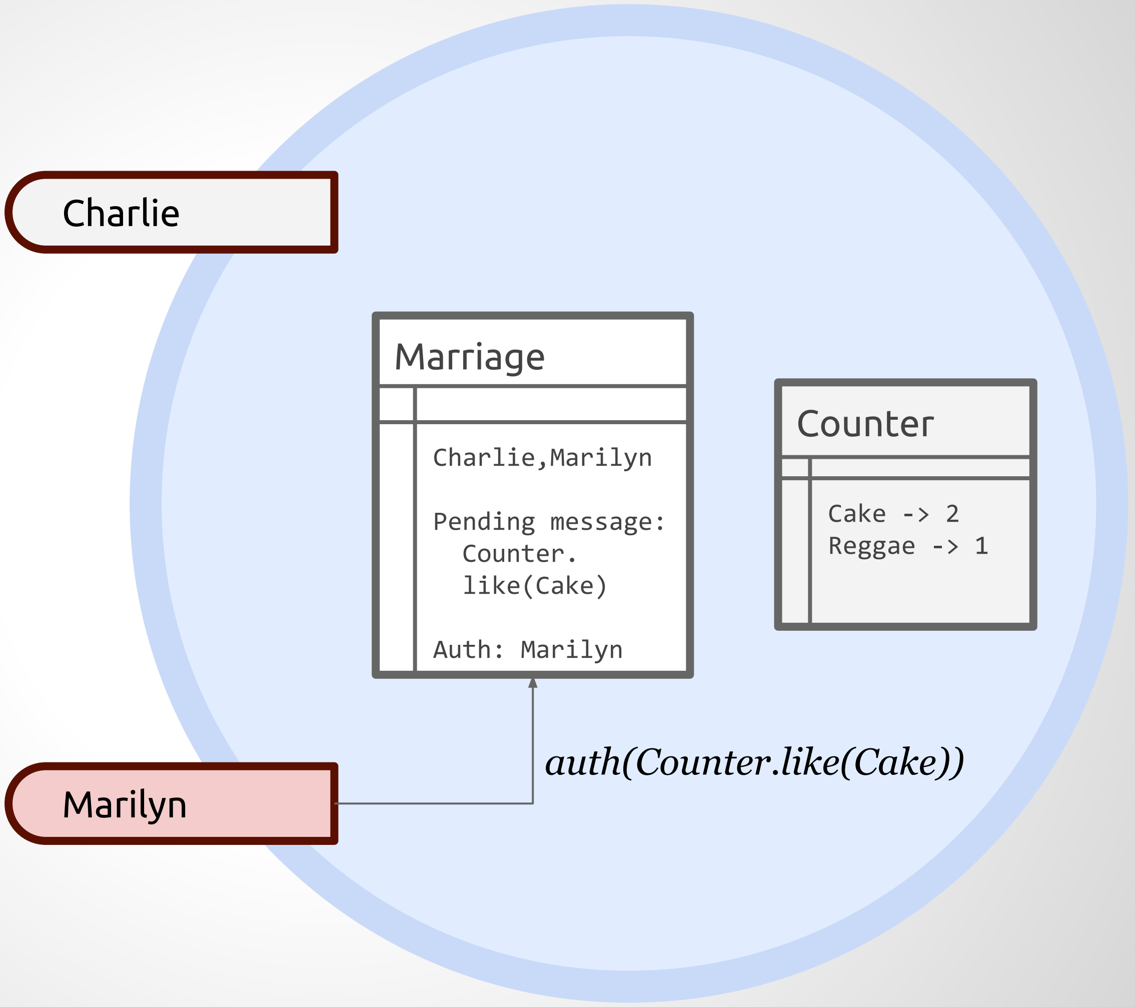



Alice
Counter
Cake -> 2
Reggae -> 1
like(Cake)
Bob

Objects can Call Each Other

e.g. Multi-signature “Marriage contract” could act as individual in terms of voting


Charlie
Marilyn
Marriage
Charlie,Marilyn
Counter
Cake -> 2
Reggae -> 1

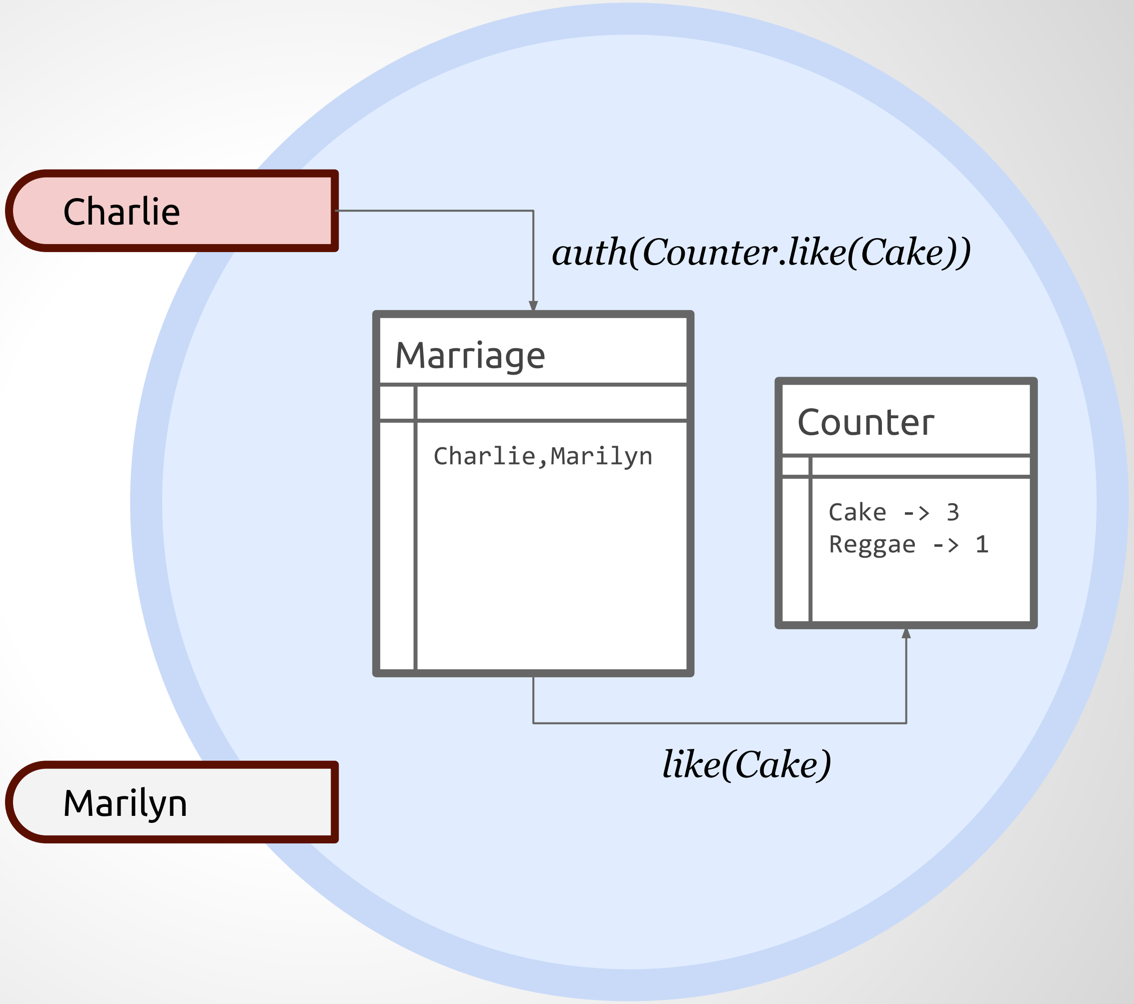


```
Charlie,Marilyn

Pending message:
  Counter.
  like(Cake)

Auth: Marilyn
```

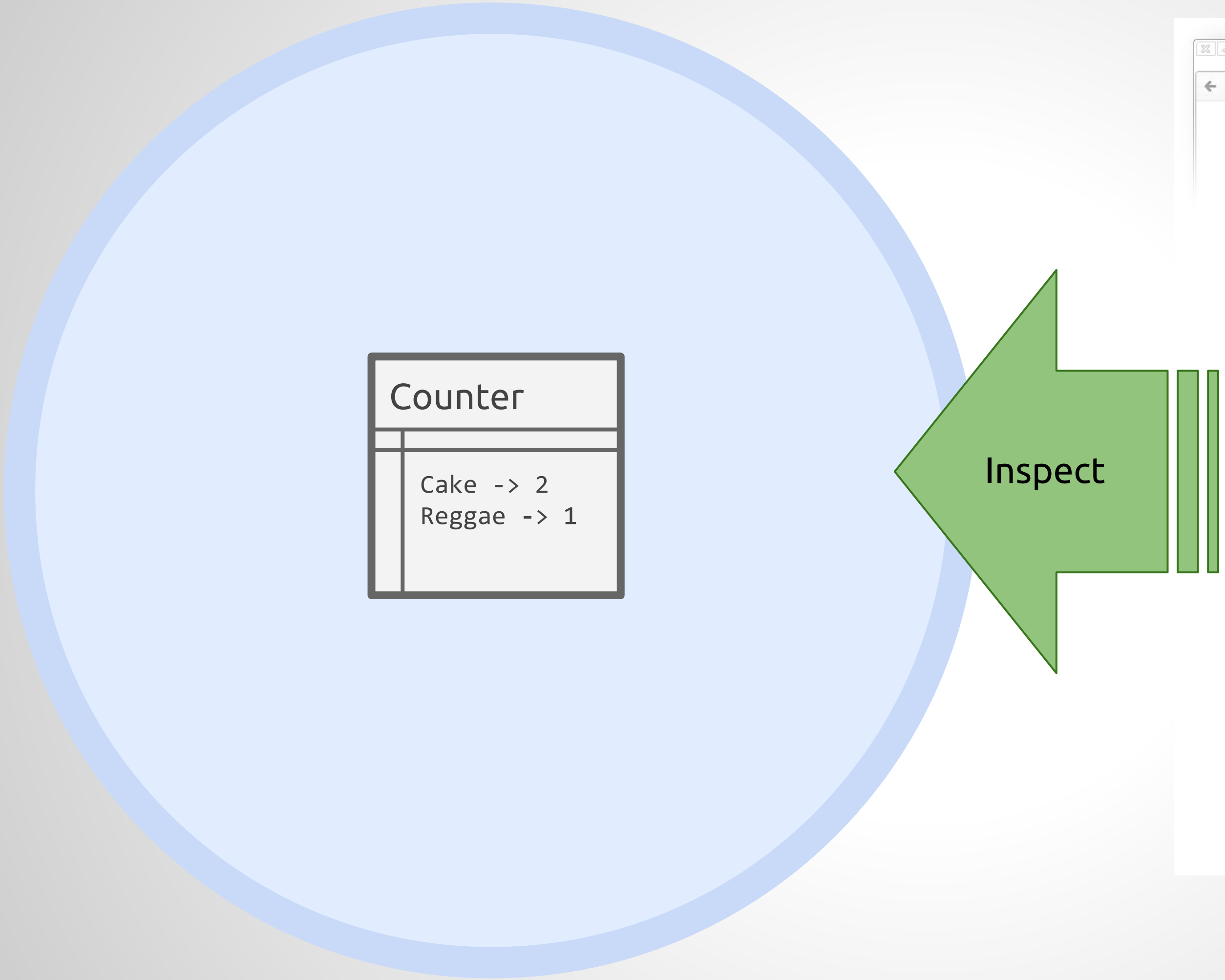
```
Cake -> 2
Reggae -> 1
```

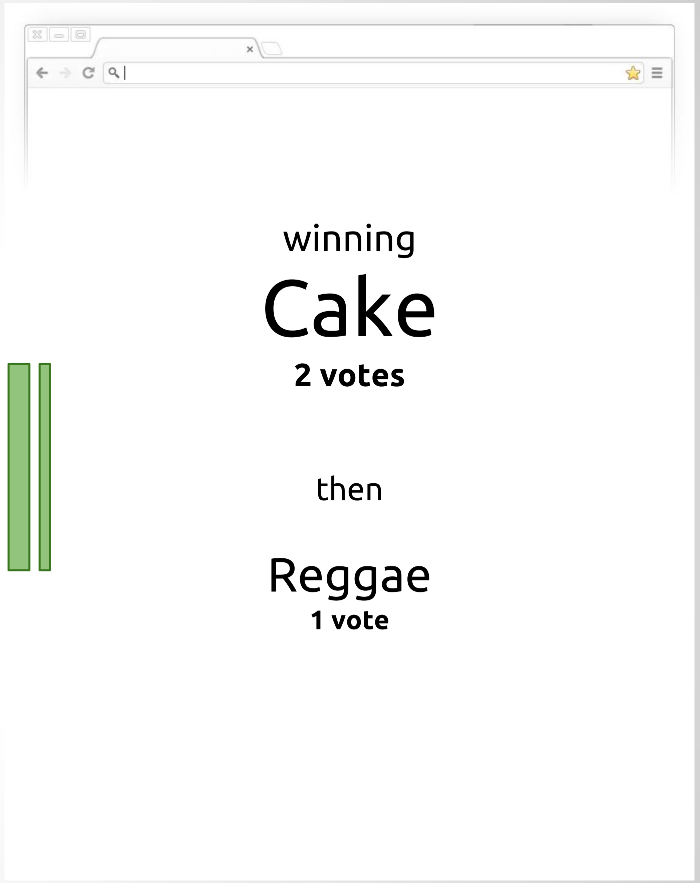

Charlie
auth(Counter.like(Cake))
Marriage
Charlie,Marilyn
Counter
Cake -> 3
Reggae -> 1
like(Cake)
Marilyn

Only Changes Cost

Network takes fees for operations that alter objects.

Operations that merely inspect the object are gratis.




winning

Cake

2 votes

then

Reggae

1 vote

Guarantees

Atomicity

Entire operation runs or nothing does

Synchrony

No two operations can interfere with each other

Provenance

All messages (method calls) can be inspected to determine caller address

Guarantees

Permanence

Object’s data are **permanent**

Immortality

Object can **never** be externally deleted - can only voluntarily commit suicide

Immutability

Object’s code can **never** be changed

Solidity Language

Familiar

Syntax similar to Javascript

Object Orientated

Classes, inheritance, methods, fields, events, modifiers

All-in-one: Database + Code

No need to introduce conceptual split between backing store & code

Counter

```
contract Counter {

function like( string _what ) {
    count[_what]++;
}

mapping ( string => int ) public count;

}
```

Counter (One Vote Only)

```
contract Counter {

function like( string _what ) {
    if (!voted[msg.sender]) {
        count[_what]++;
        voted[msg.sender] = true;
    }
}

mapping ( string => int ) public count;

mapping ( address => bool ) public voted;

}
```

Counter (Modifier)

```
contract onetime {

modifier once {
    if (!voted[msg.sender]) {
        voted[msg.sender] = true;
        _
    }
}

mapping ( address => bool ) public voted;

}
```

```
contract Counter is onetime {

function like( string _what ) once {
    count[_what]++;
}

mapping ( string => int ) public count;

}
```

Javascript API

Objects generally usable directly in JS.

E.g.:

```
theCounter.like(‘Cake’);
```

Counter (UI)

```
contract Counter is onetime {

function like( string _what ) once {
    count[_what]++;
}

mapping ( string => int ) public count;

}
```

```
<body>
<script>
Counter = /*special init code*/
theCounter = Counter.at(/*Counter’s address*/)
</script>
<button onclick=”theCounter.like(‘Cake’)”>
Cake!
</button>
<button onclick=”theCounter.like(‘Reggae’)”>
Reggae!
</button>
</body>
```

Counter (Payment)

```
contract costly {

modifier costs(uint _amount) {
    if (msg.value >= _amount) _
}

}
```

```
contract Counter is onetime, costly {

function like( string _what )
  once,
  costs(1 ether) {
    count[_what]++;
}

mapping ( string => int ) public count;

}
```

Javascript API

Value (in terms of ether) can be attached with `sendTransaction`

E.g.:

```
theCounter.like.sendTransaction(
‘Cake’, {value: 1000000000});
```

Counter (Payment UI)

```
contract Counter is onetime, costly {

function like( string _what )
  once,
  costs(1 ether) {
    count[_what]++;
}

mapping ( string => int ) public count;


}
```

```
<body>
<script>
Counter = /*special init code for the type*/
theCounter = Counter.at(/*the global address*/)
function like(what) {
    theCounter.like.sendTransaction(what,
        { value: web3.eth.toWei(1, ‘ether’) });
}
</script>
<button onclick=”like(‘Cake’)”>Cake!</button>
<button onclick=”like(‘Reggae’)”>Reggae!</button>
</body>
```

Counter (Events)

```
contract Counter is onetime, costly {

event NewLeader(string _what);

function like(string _what) once, costs(1 ether) {
    if (++count[_what] > count[leader]) {
        leader = _what;
        NewLeader(_what);
    }
}

mapping ( string => int ) public count;
string public leader;
}
```

Javascript API

Events happen through JS callback functions.

E.g.:

```
function cb(error, result) { /*result._what*/ }
theCounter.NewLeader(cb)
```

Counter (Events)

```
contract Counter is onetime, costly {

event NewLeader(string _what);

function like(string _what) once, costs(1 ether) {
    if (++count[_what] > count[leader]) {
        leader = _what;
        NewLeader(_what);
    }
}

mapping ( string => int ) public count;
string public leader;
}
```

```
<body>
<h1 id=”leader”></h1>with
<h2 id=”leadercount”></h2>
<script>
Counter = /*special init code*/
theCounter = Counter.at(/*Counter’s address*/)
l = document.getElementById(‘leader’)
lc = document.getElementById(‘leadercount’)
theCounter.NewLeader(function(e, r) {
    l.innerHTML = r._what
    lc.innerHTML = theCounter.count(r._what)
});
</script>
</body>
```

World Computer

Why?

Ethereum is an Innovation Commons

Compared to the walled garden of the server

Servers are Walled Gardens

Interoperability Difficult

Reliability, standards, trust, security collude to make it a nightmare

Increased Barriers

Naturally supportive of monopolies;

try integrating trade or payment without a third party

Expensive

Servers are expensive to set up and maintain;

Ethereum is always-on, always ready

Not to mention Privacy

Privacy

Less siloing of user-data; less intermediation; more privacy

Security

Security through nihilism; there’s no server to hack!

Authenticity

All interactions with the Global Computer are cryptographically signed:

Unauthorised Interactions are ***Impossible***

Bigger Picture

Commoditise Trust

Centralisation & Central Authorities

Single point of control

Single point of failure

Single bottleneck

Software Development

Individual coders

Strict hierarchy “cathedral”

Open-source hackers “bazaar”

Clones and forks (Github-style)

Communication

Word-of-Mouth (close to zero)

Press/Radio/Television

Internet

Mobile Mesh?

General Theme

...or natural order?

Nothing

*...strong individual imposes **order**; progress to...*

Centralisation

*...order enables **cooperation**; progress towords...*

Decentralisation

Efficient

Scalable

Resilient

Ethereum

Platform for ***Zero-trust Computing***

for

autonomous trading

smart contracts

interoperable infrastructure

permissions management

trust webs ...

Ethereum & Crypto-law

Uses **blockchain** to implement **arbitrary social contracts** without a central server

Ethereum & Web3

Infrastructure for the ITC revolution

Ethereum Zero-trust computing
Whisper Private asynchronous bulletins
Telehash Private realtime comms
IPFS/Swarm Decentralised data distribution

Basic Premise

“The truth is more common than any one lie”

Liars can try but, ultimately, they’ll be ignored

Where are we?

On the way

Timeline

Oct ‘13 Initial whitepaper written

Dec ‘13 Development begins

Jan ‘14 Public announcement

Apr ‘14 Formal specification written

Aug ‘14 Crowd sale generates $15m

Late ‘14 Development scales up

Mar ‘15 Pre-release testnet begins

At present

~30 devs around the globe

100% Free Software

Inclusive development, open source code.

Official C++, Go, Python implementations

Unofficial JS, Java, Haskell implementations

Plans

~~**Summer ‘15** v1.0 release~~

Winter ‘15-’16 PoS, light-client upgrades

Funding (eth**core**)

Summer ‘18 Tentative 2.0 release

“2.0”

Key differences:

Scalable

Currency agnostic

Hardware accelerated

Questions?

Ethereum

The world computer

Dr. Gavin Wood

@gavofyork