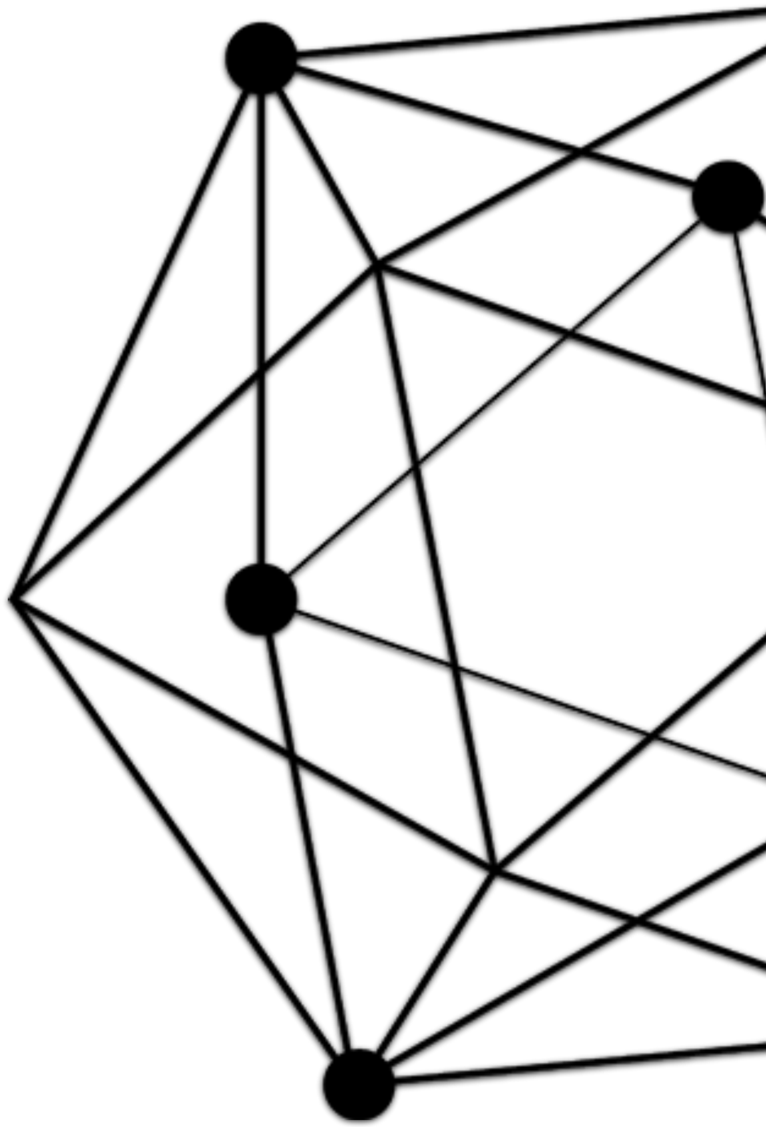
BLOCKCHAIN MONTREAL

Mimblewimble



Quentin Le Sceller - Catallaxy

q.lesceller@gmail.com

January 25, 2018

catallaxy

Outline

MIMBLEWIMBLE

- History
- What is Mimblewimble?
- Transactions
- Blockchain
- Scriptless Script

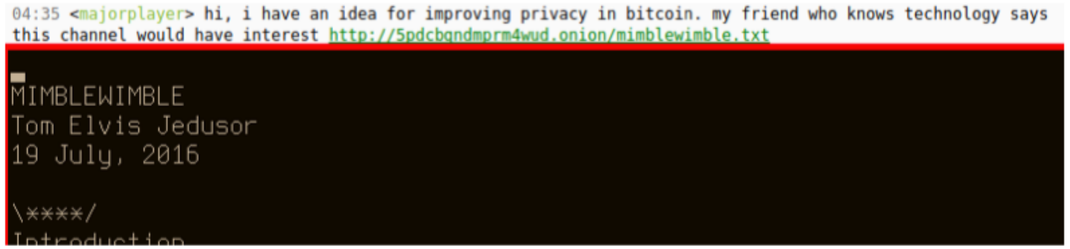
GRIN

- What is Grin?
- Proof-of-Work
- Characteristics
- What's next?
- Get Involved

History

2016

- On Tuesday, August 2 at 4:30 UTC, an individual logged on a Bitcoin IRC research channel. He dropped a paper on a tor hidden service: Mimblewimble by Tom Elvis Jedusor. He then signed off and he never came back.



- The next week, after discussion on Reddit between Andrew Poelstra, Gregory Sanders and others Bitcoin developers, the idea is considered worth pursuing.

- On October 10, Andrew Poelstra publish a follow-up paper explaining in details Mimblewimble.

History

2016

Initial Release
"Tom Elvis Jedusor" posts an .onion link on Bitcoin Wizard IRC leading to a paper: "Mimblewimble"

August 2

Follow-up Paper
Andrew Poelstra releases a paper explaining Mimwimble in details

October 10

First Implementation
Ignotus Peverell starts the first implementation of Mimblewimble: Grin

October 20

What is Mimblewimble?

- Mimblewimble is a completely new blockchain design that offers several benefits:
  - Privacy by default
  - Massively Prunable
  - Scalable
  - Relies on strong and proven cryptography (ECC)
- Can be implemented as a Bitcoin side chain or as an altcoin.
- Output-based like Bitcoin but without script

Mimblewimble Transactions

- In Bitcoin, every output has a script (script pub key) attached to it.
  In order to spend one of them, you must verify the conditions in the script.

- In Mimblewimble outputs only have public keys: no script.

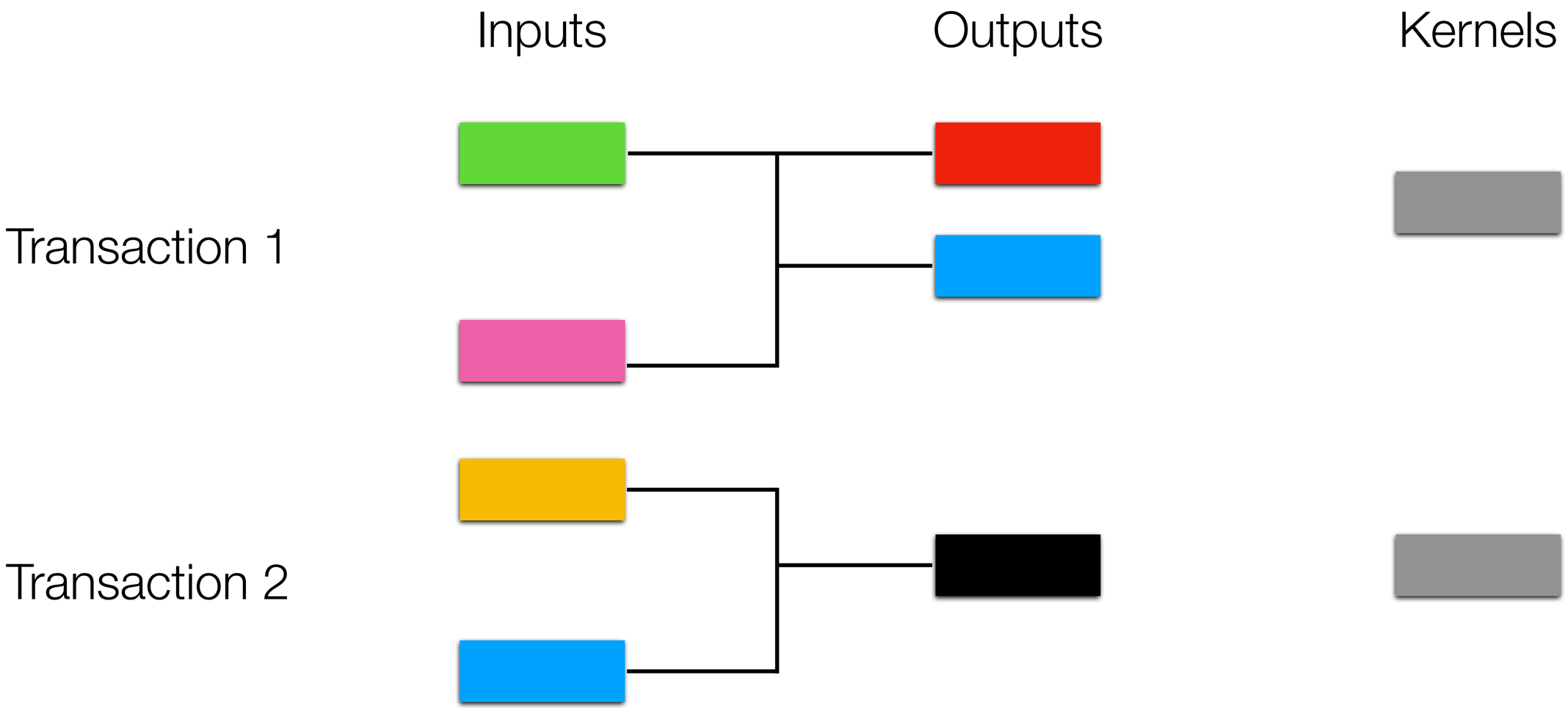
- Hence Mimblewimble transactions are scriptless
  => no payment channels, no atomic swap and multisignature at first glance

Mimblewimble Transactions

Mimblewimble transactions contains the following parts:

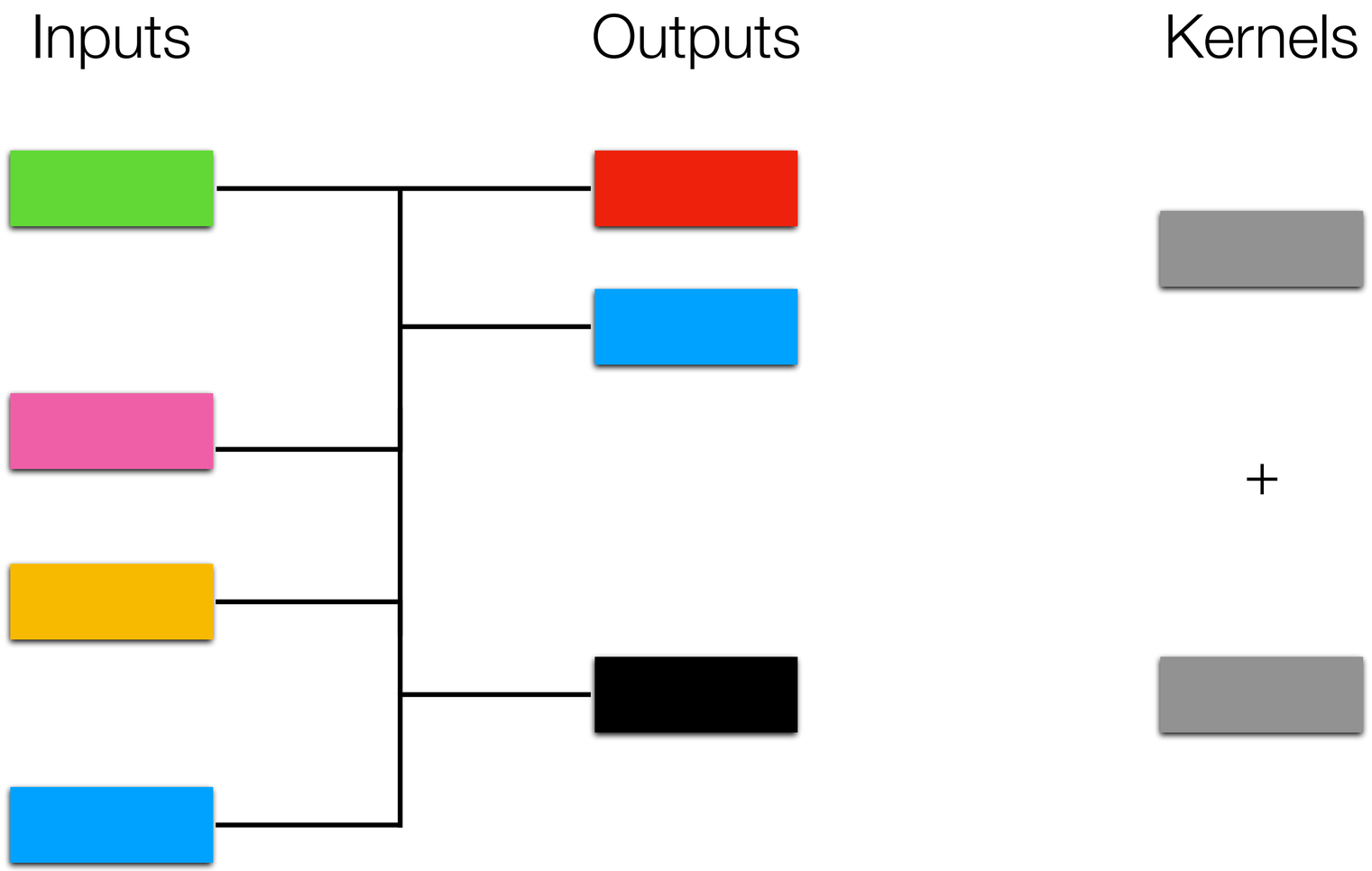
- Inputs (reference to old outputs)
- Outputs: confidential transactions (amounts are blinded) + rangeproof (cryptographic proof that no money was created)
- Kernel: difference between outputs and inputs + mining fee + signature

Mimblewimble Transactions



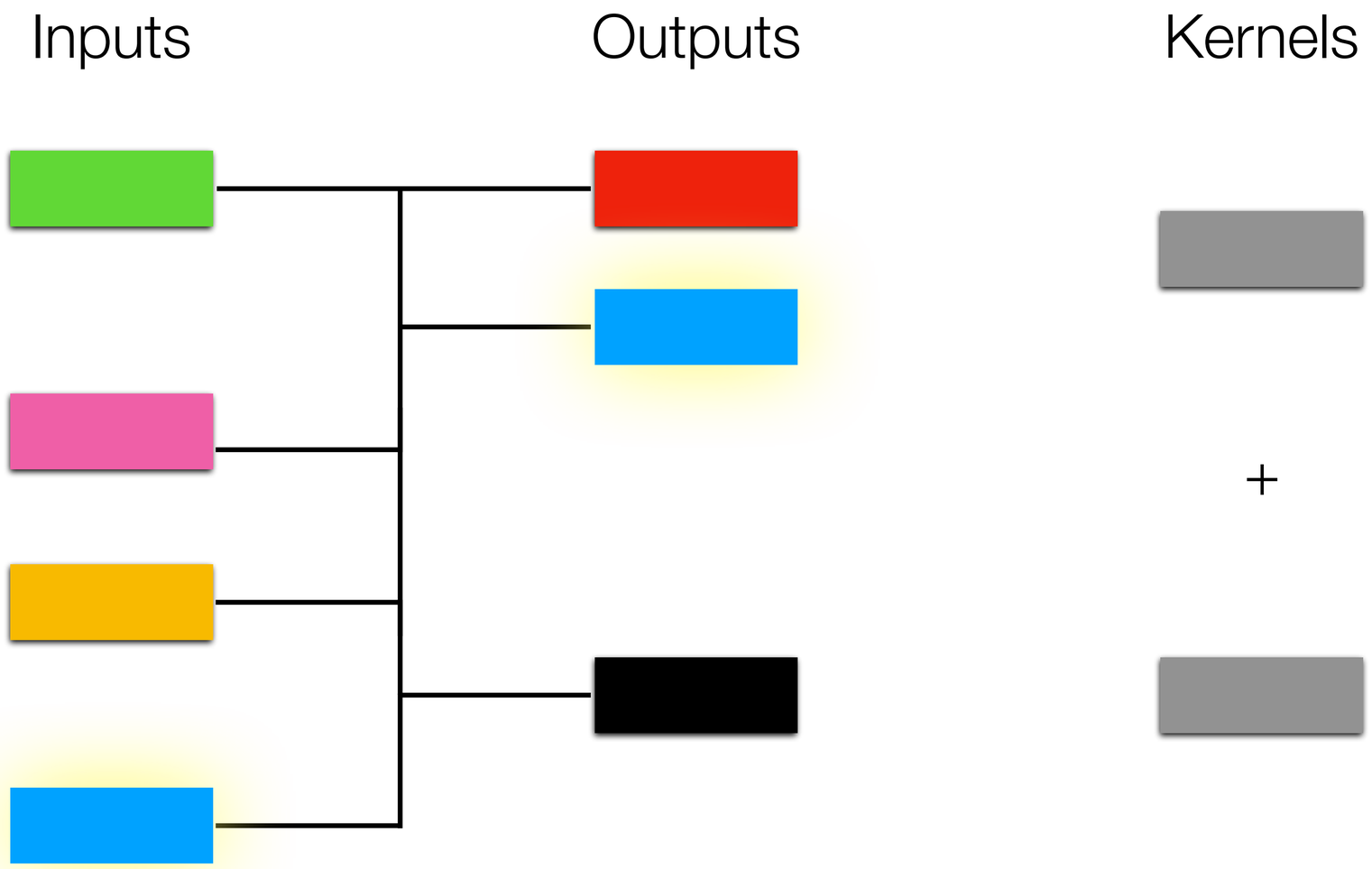
Mimblewimble Transactions

Transaction



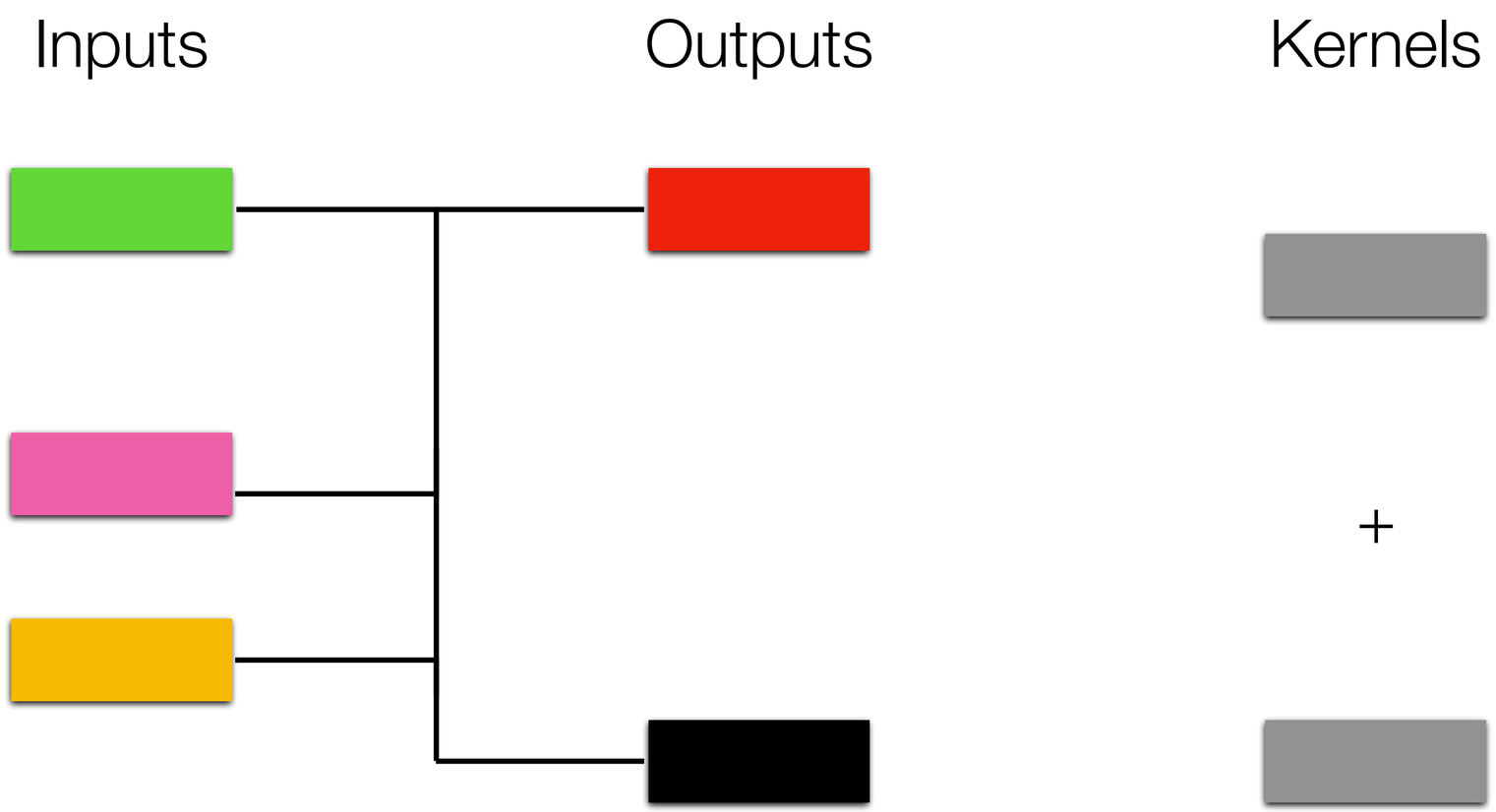
Mimblewimble Transactions

Transaction



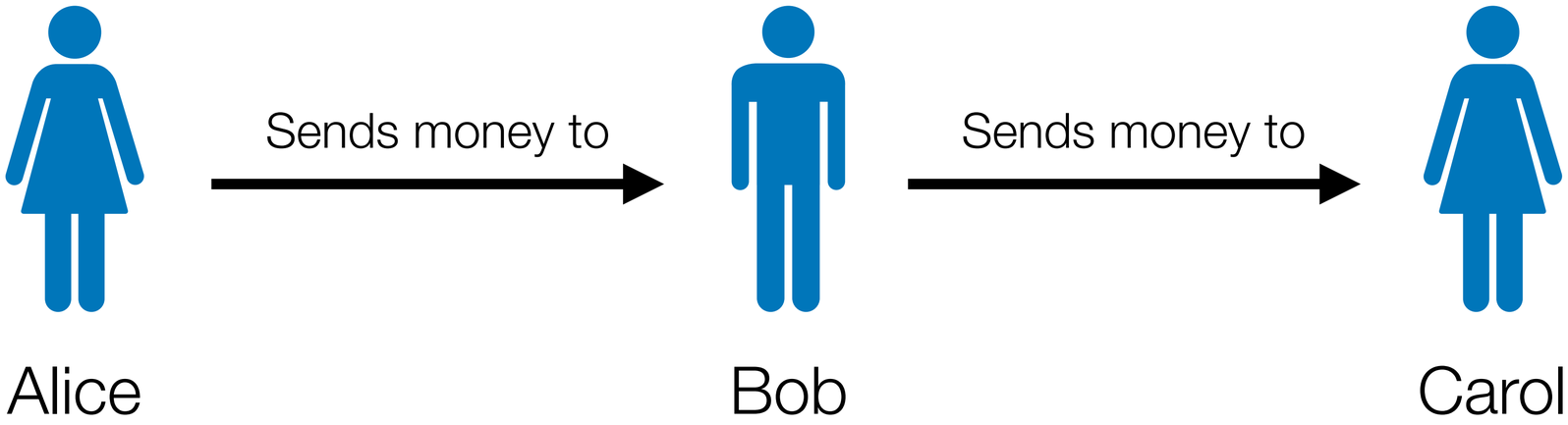
Mimblewimble Transactions

Transaction

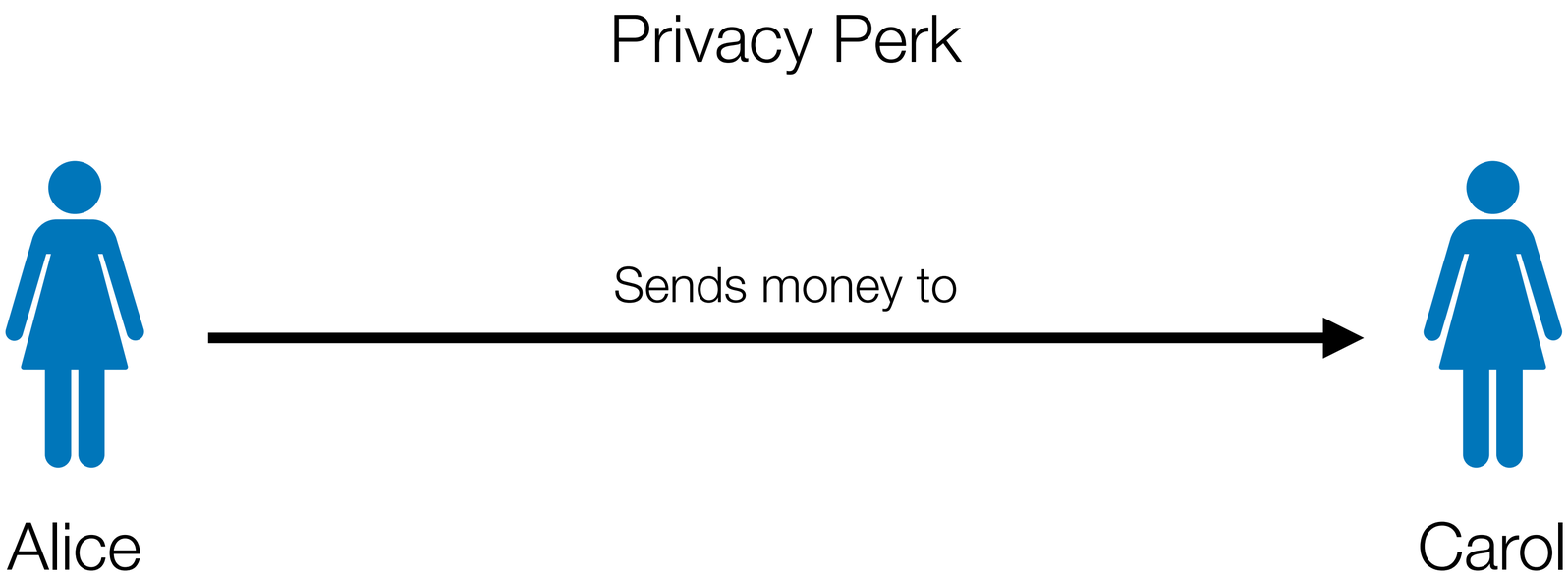


Mimblewimble Transactions

Privacy Perk



Mimblewimble Transactions



Mimblewimble Transactions

- Mimblewimble transactions are a variant of a non-interactive Coinjoin transaction (every participant doesn't need to be online in order to group the transactions)
- There is no addresses
- There is no amounts

Mimblewimble Blockchain

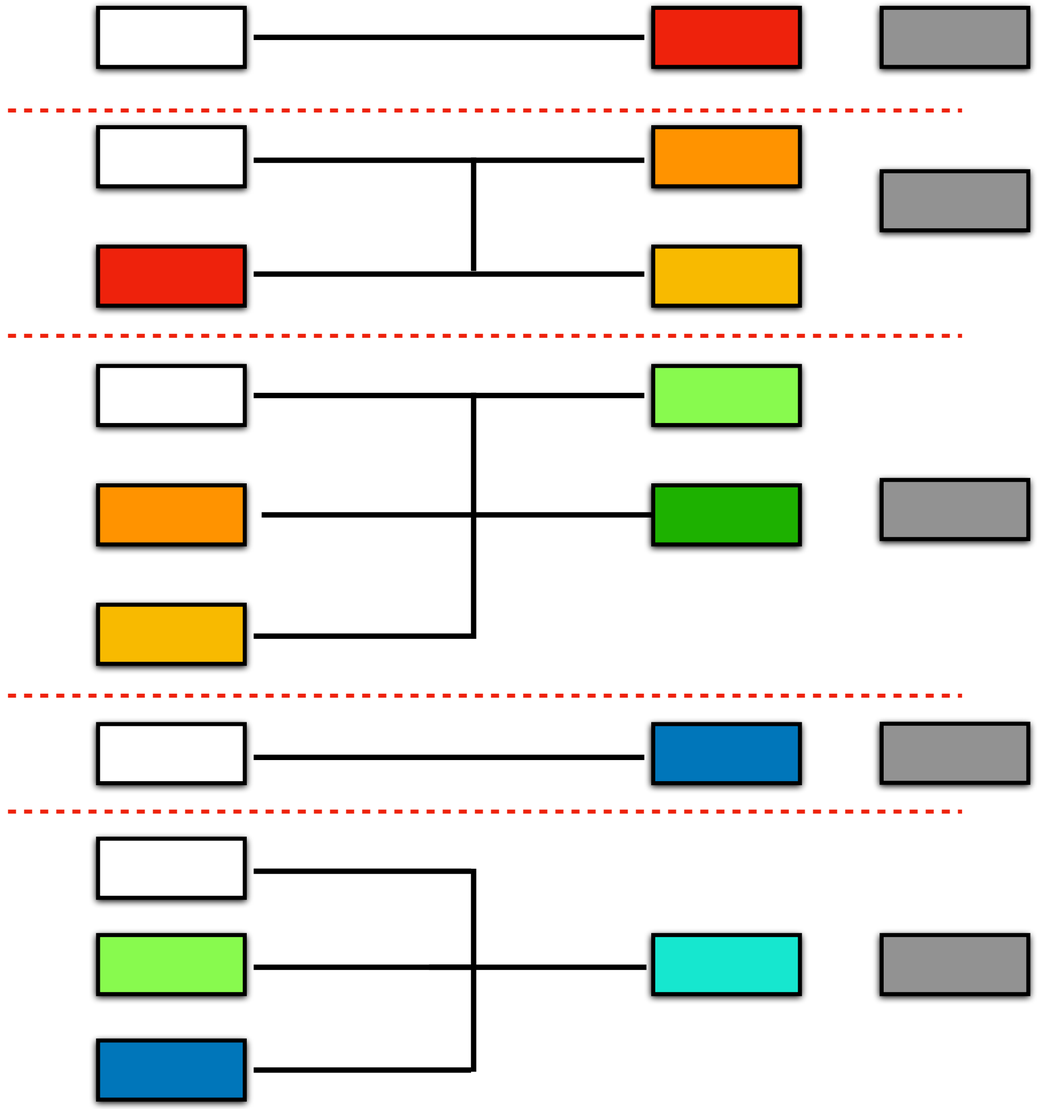
Block 1

Block 2

Block 3

Block 4

Block 5



Mimblewimble Blockchain

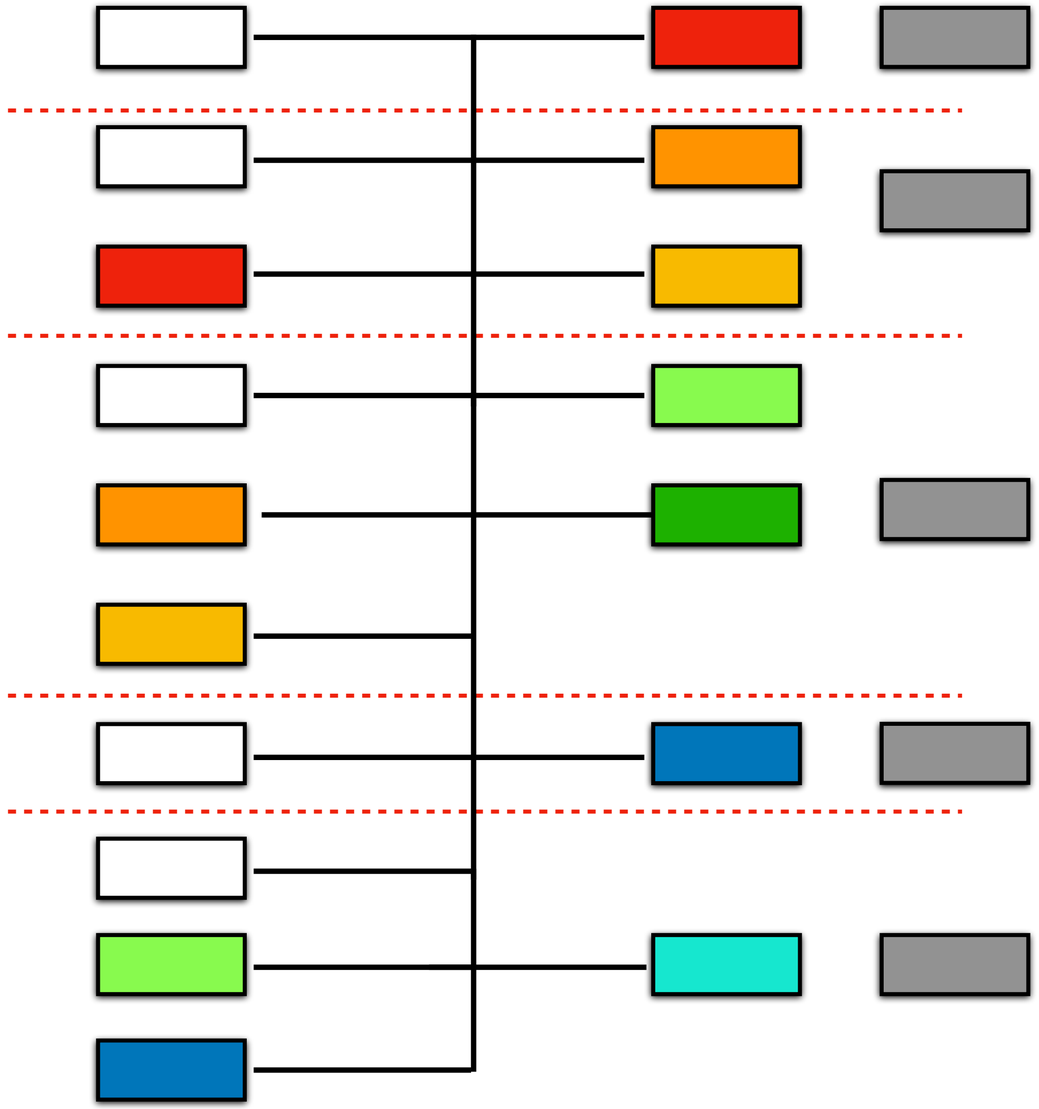
Block 1

Block 2

Block 3

Block 4

Block 5



Mimblewimble Blockchain

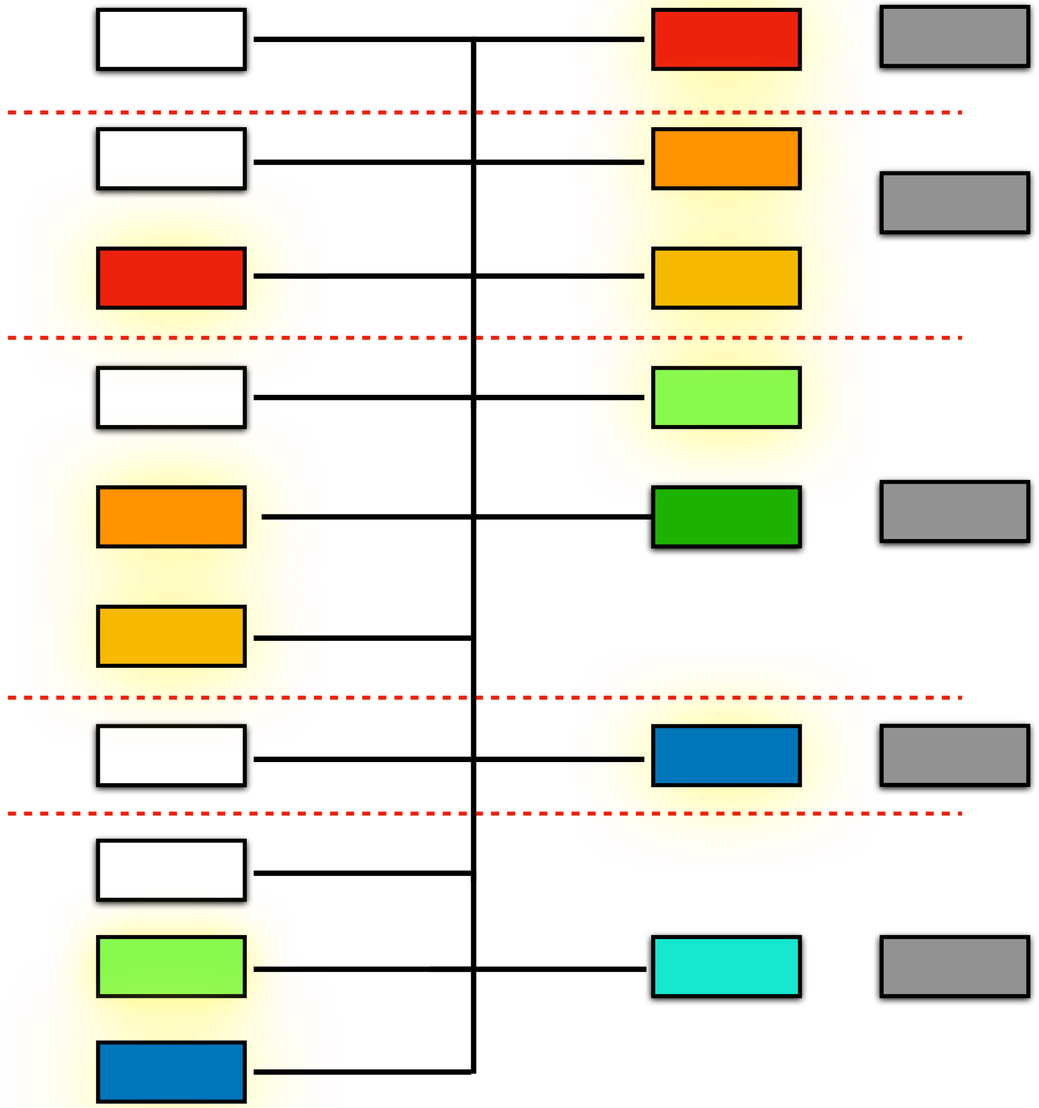
Block 1

Block 2

Block 3

Block 4

Block 5



Mimblewimble Blockchain

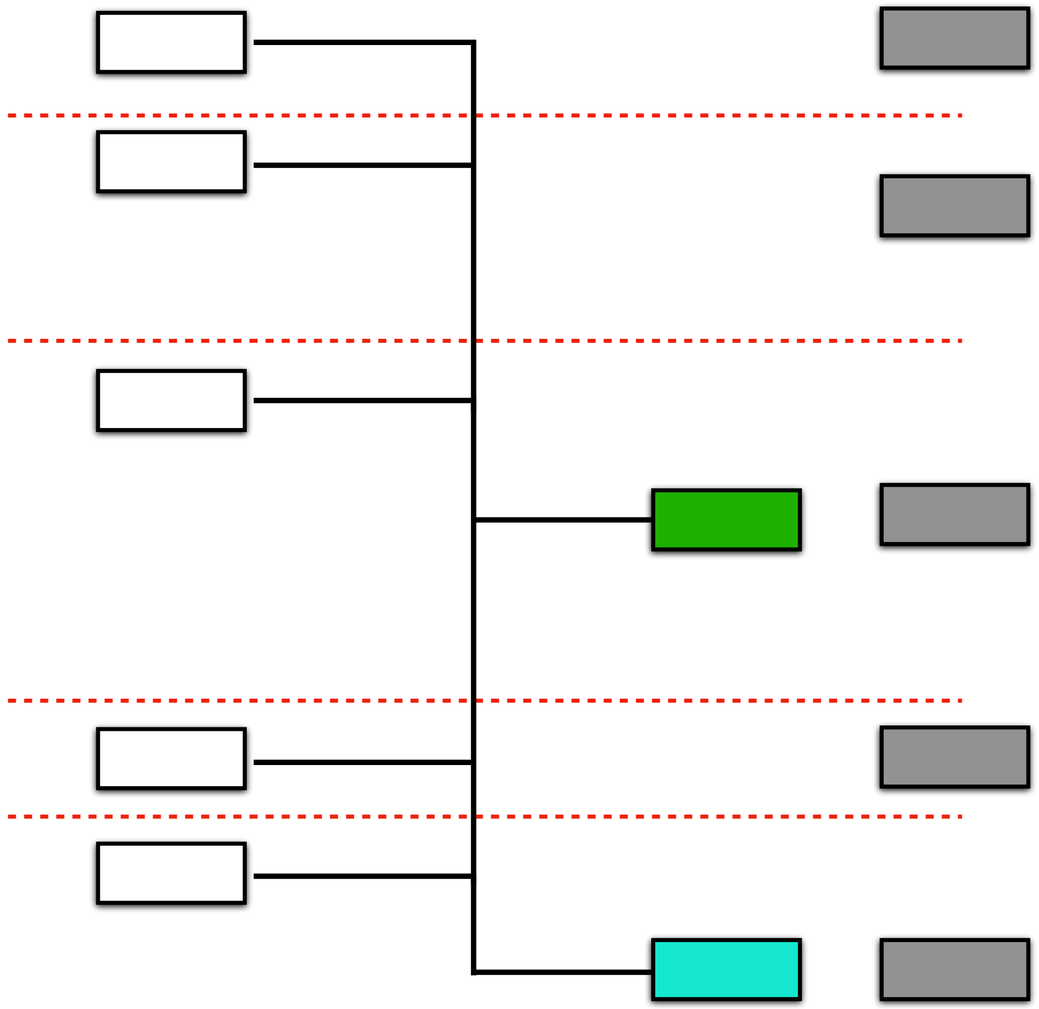
Block 1

Block 2

Block 3

Block 4

Block 5



Mimblewimble Blockchain

Applied to Bitcoin

- 150 millions transactions with approximately 400 millions outputs, 65 millions of which are unspent
- Currently 180 Gb; if you add CT 450 Gb.
- Mimblewimble with Bitcoin:
  18 Gb of transactions kernels, headers…
  2 Gb of UTXO
  45 Gb of UTXO rangeproofs.

Trust Model

- A transaction is valid if signed by the owner of the inputs and is non inflationary
- Once in a block, transactions cannot be reversed without doing enough work
- Current state reflects zero net theft and inflation
- No need to know the exact sequence of transaction in order to verify that the blockchain is correct

Scriptless Script

A first glance, scripts are not possible on the MW blockchain.
However, using some magic (with Adaptor Signature derived from Schnorr signature) its possible to do:

- Multi-signature transactions.
- Atomic swaps.
- Time-locked transactions and outputs.
- Lightning Network

Recap

- Mimblewimble is a new blockchain design proposed by an anonymous
- No amounts and no addresses
- Transactions are aggregated in block to form one unique transactions: Removing intermediaries.
- Resulting in a massively prunable blockchain

```
core::{Transaction, Input, Output, OutputFeatures, SwitchCommitHash, COINBASE_OUTPUT, DEFAULT_OUTPUT};
core::hash::Hash;
keychain;
keychain::{Keychain, BlindSum, BlindingFactor, Identifier};
util::LOGGER;

Context information available to transaction combinators.
struct Context<'a> {
    keychain: &'a Keychain,


Function type returned by the transaction combinators. Transforms a
(Transaction, BlindSum) pair into another, provided some context.
type Append = for<'a> Fn(&'a mut Context, (Transaction, BlindSum)) -> (Transaction, BlindSum);

Adds an input with the provided value and blinding key to the transaction
being built.
uild_input(
    value: u64,
    features: OutputFeatures,
    out_block: Option<Hash>,
    key_id: Identifier,
 Box<Append> {
    Box::new(move |build, (tx, sum)| -> (Transaction, BlindSum) {
            let commit = build.keychain.commit(value, &key_id).unwrap();
            let input = Input::new(
                    features,
                    commit,
                    out_block,
            );
            (tx.with_input(input), sum.sub_key_id(key_id.clone()))
    })


Adds an input with the provided value and blinding key to the transaction
being built.
fn input(
    value: u64,
    out_block: Hash,
    key_id: Identifier,
 Box<Append> {
```

Grin

What is Grin?

- On October 20, 2016, "Ignotus Peverell" advised Andrew Poelstra that he was working on an implementation of the Mimblewimble blockchain: Grin. https://github.com/mimblewimble/grin
- From Gringots, the wizard bank
- Shortly after, he was joined by "Antioch Peverell", "Garrick Olivander" and others HP characters. Other individual also joined, most notably Yeatsplume.

What is Grin?

- First implementation of Mimblewimble as an altcoin
- Minimal implementation of the Mimblewimble protocol
- In Rust (a programming language focused on safety, speed, and concurrency)

Not the actual Grin logo

Currently under development, slides might be obsolete soon

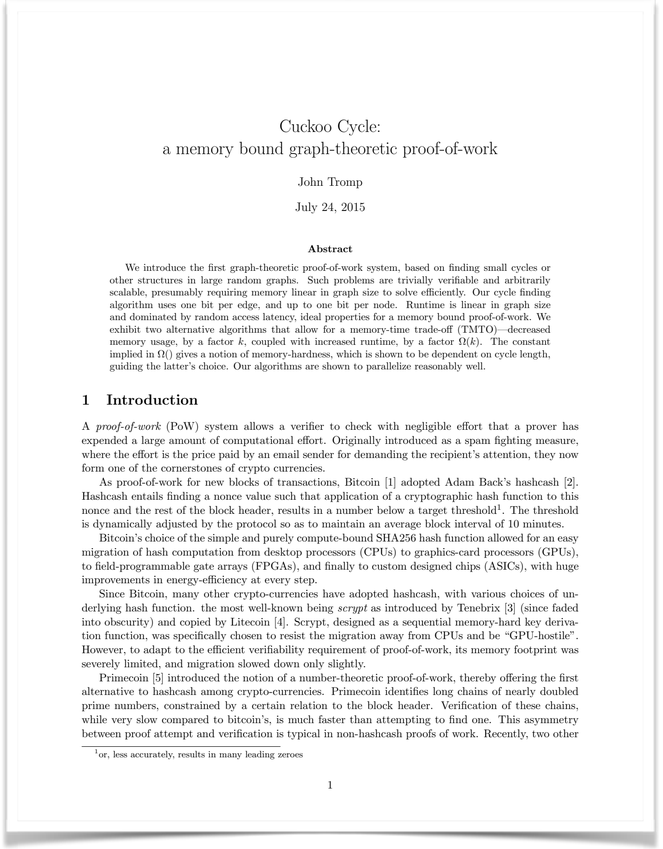
Development Funding

- Grin is a free open source software (FOSS)
- No Grin foundation and not a company, voluntary based development
- No ICO
- 100% community driven model
- Currently one developer full time funded Michael Cordner a.k.a Yeastplume

Grin Proof-of-Work

Cuckoo Cycle

- Created by John Tromp in July 2015
- First graph-theoritic proof-of-work system
- Memory-bound algorithm
- Designed to be ASIC resistant
- Only CPU and GPU (hopefully)



Cuckoo Cycle:
a memory bound graph-theoretic proof-of-work

John Tromp

July 24, 2015

Abstract

We introduce the first graph-theoretic proof-of-work system, based on finding small cycles or other structures in large random graphs. Such problems are trivially verifiable and arbitrarily scalable, presumably requiring memory linear in graph size to solve efficiently. Our cycle finding algorithm uses one bit per edge, and up to one bit per node. Runtime is linear in graph size and dominated by random access latency, ideal properties for a memory bound proof-of-work. We exhibit two alternative algorithms that allow for a memory-time trade-off (TMTO)—decreased memory usage, by a factor k, coupled with increased runtime, by a factor $\Omega(k)$. The constant implied in $\Omega()$ gives a notion of memory-hardness, which is shown to be dependent on cycle length, guiding the latter's choice. Our algorithms are shown to parallelize reasonably well.

1 Introduction

A *proof-of-work* (PoW) system allows a verifier to check with negligible effort that a prover has expended a large amount of computational effort. Originally introduced as a spam fighting measure, where the effort is the price paid by an email sender for demanding the recipient's attention, they now form one of the cornerstones of crypto currencies.

As proof-of-work for new blocks of transactions, Bitcoin [1] adopted Adam Back's hashcash [2]. Hashcash entails finding a nonce value such that application of a cryptographic hash function to this nonce and the rest of the block header, results in a number below a target threshold[1]. The threshold is dynamically adjusted by the protocol so as to maintain an average block interval of 10 minutes.

Bitcoin's choice of the simple and purely compute-bound SHA256 hash function allowed for an easy migration of hash computation from desktop processors (CPUs) to graphics-card processors (GPUs), to field-programmable gate arrays (FPGAs), and finally to custom designed chips (ASICs), with huge improvements in energy-efficiency at every step.

Since Bitcoin, many other crypto-currencies have adopted hashcash, with various choices of underlying hash function. the most well-known being *scrypt* as introduced by Tenebrix [3] (since faded into obscurity) and copied by Litecoin [4]. Scrypt, designed as a sequential memory-hard key derivation function, was specifically chosen to resist the migration away from CPUs and be "GPU-hostile". However, to adapt to the efficient verifiability requirement of proof-of-work, its memory footprint was severely limited, and migration slowed down only slightly.

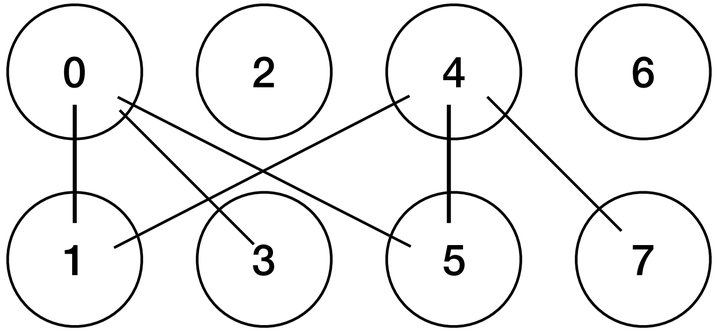
Primecoin [5] introduced the notion of a number-theoretic proof-of-work, thereby offering the first alternative to hashcash among crypto-currencies. Primecoin identifies long chains of nearly doubled prime numbers, constrained by a certain relation to the block header. Verification of these chains, while very slow compared to bitcoin's, is much faster than attempting to find one. This asymmetry between proof attempt and verification is typical in non-hashcash proofs of work. Recently, two other

[1] or, less accurately, results in many leading zeroes

1

Grin Proof-of-Work

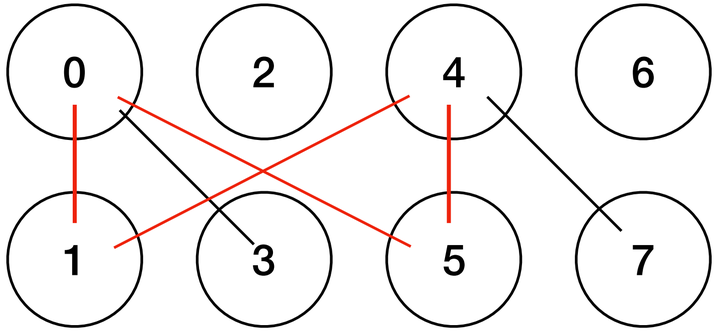
- Goal: find "cycle" in a graph
  Find a series of connected nodes starting and ending on the same node
  e.g. : find a cycle of length 4 in the following graph



Grin Proof-of-Work

- Goal: find "cycle" in a graph
  Find a series of connected nodes starting and ending on the same node
  e.g. : find a cycle of length 4 in the following graph

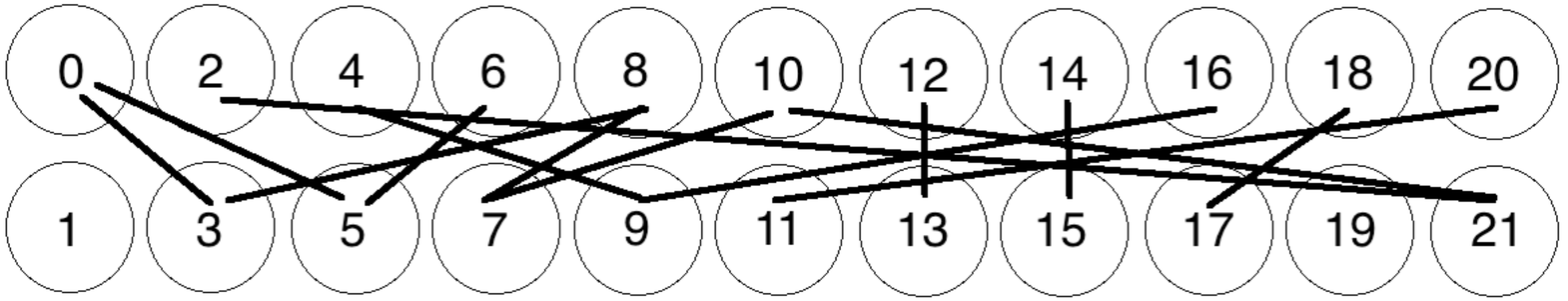
0 - 5 - 4 - 1 -0



Grin Proof-of-Work

Easy ?

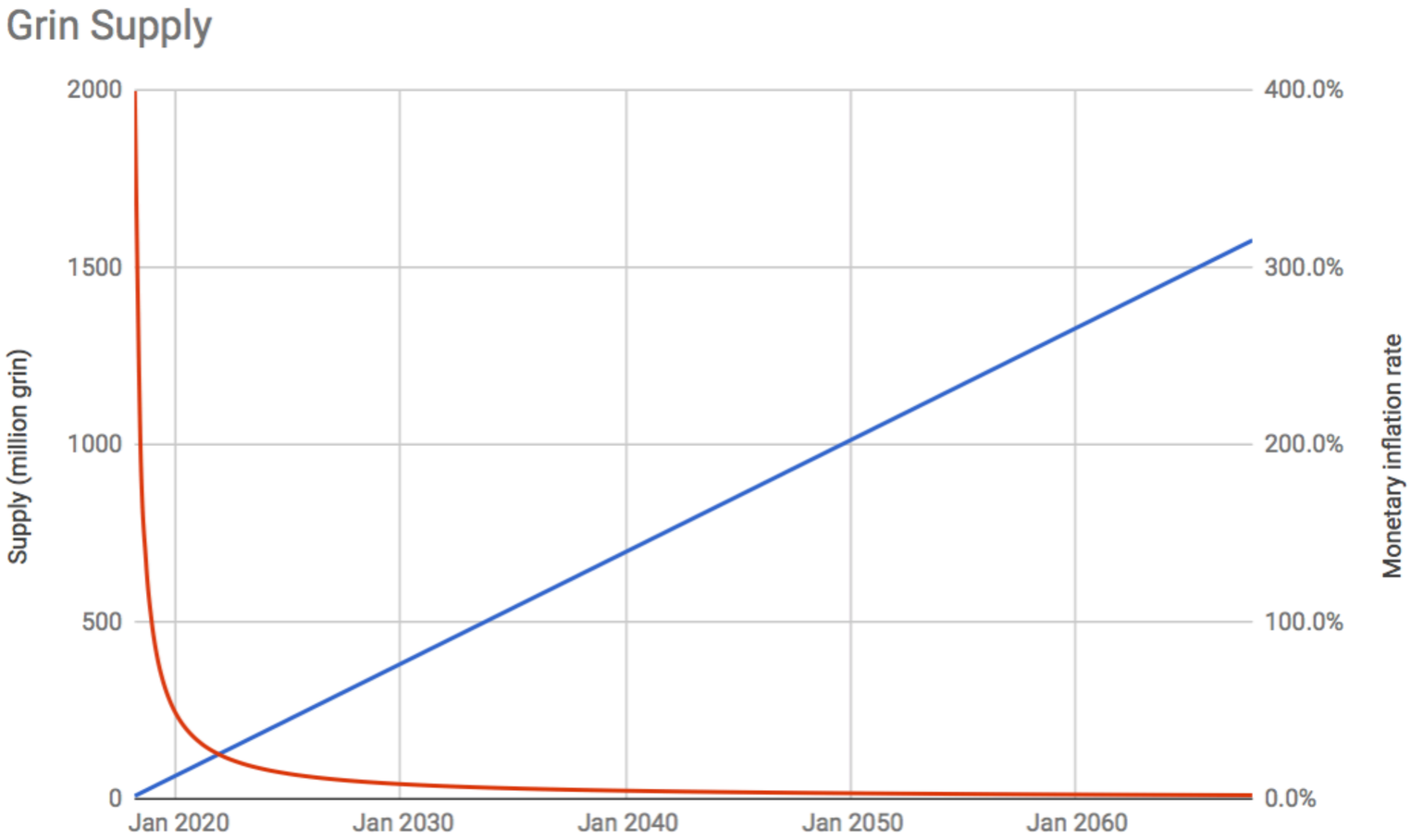
Try to find a cycle of length 8 here



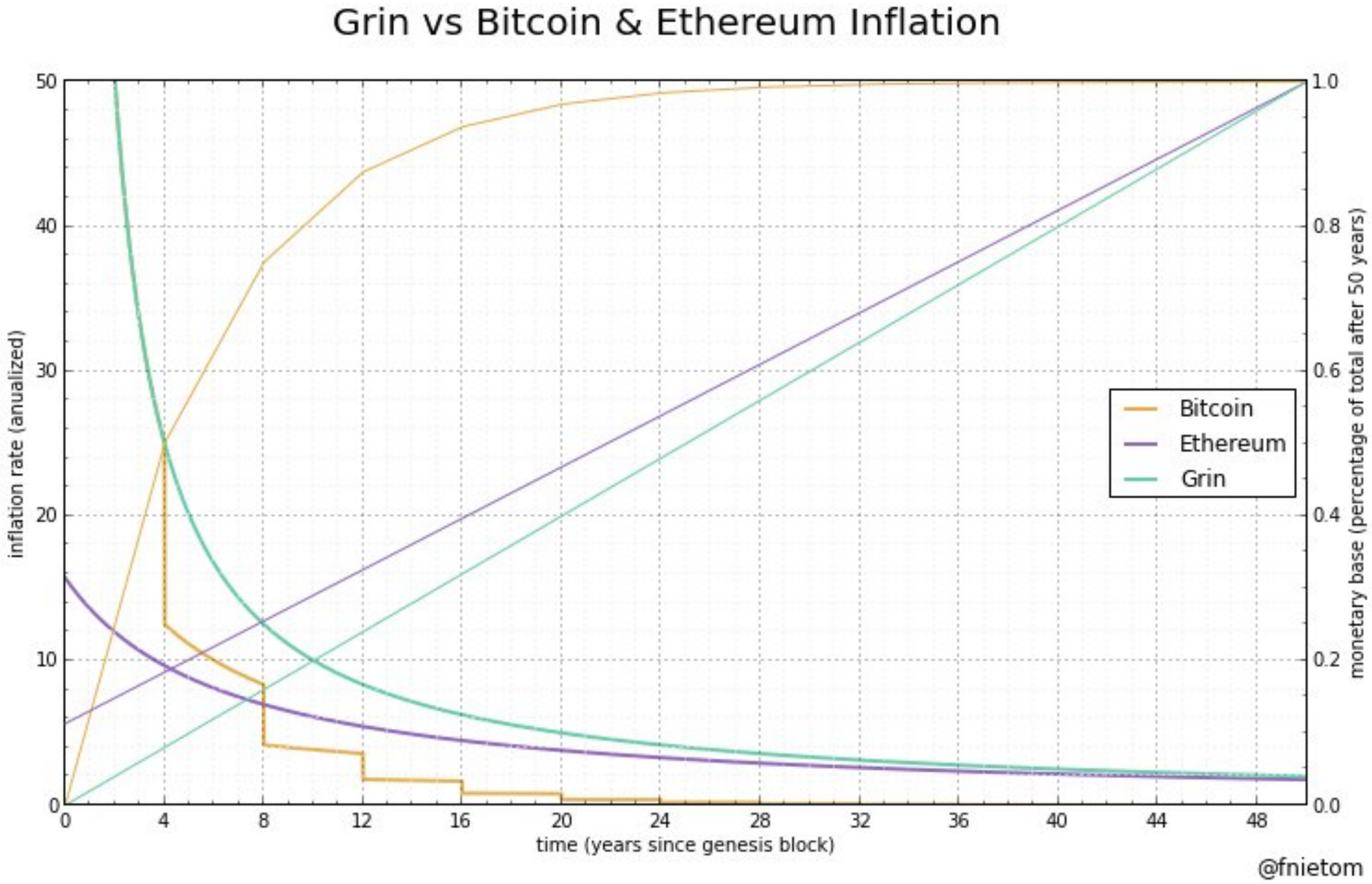
Grin Characteristics

- 60 grins per block
- Fast block time: 1 block every minute
- 1 grin per second
- Difficulty adjusted every 23 block with difficulty calculation is based on both Digishield (Digibyte) and GravityWave (Zcash)

Grin Characteristics



Grin Characteristics

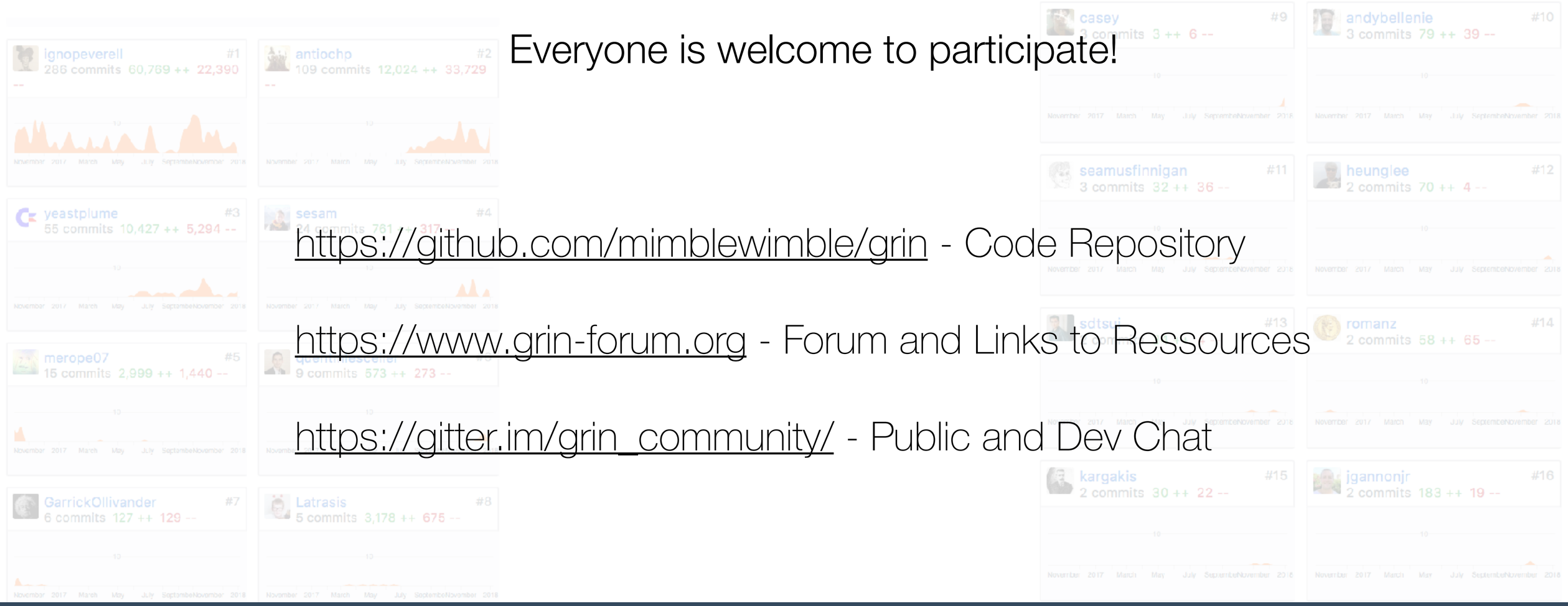


What's next?

No deadline

- Lot of development is still needed
- Currently on testnet1
- Soon™ testnet2
- Later this year (hopefully) mainnet

Get Involved



Everyone is welcome to participate!

https://github.com/mimblewimble/grin - Code Repository

https://www.grin-forum.org - Forum and Links to Ressources

https://gitter.im/grin_community/ - Public and Dev Chat