# Correct-by-Construction Asynchronous, Byzantine fault tolerant, Binary Casper Consensus Protocol

Vlad Zamfir

DEVCON2

# Table of Contents:

- Consensus: Background Knowledge

- Correct-by-Construction Binary Casper Consensus

- Relationship to Previously Existing Literature

- Future Work

# Consensus: Background Knowledge

- What is a Consensus Protocol?

- What is Having Consensus?

- What are safety and liveness?

- What are Asynchronous Networks?

- What are Byzantine faults?

# What are Consensus Protocols?

**Consensus protocols are used to guarantee that (protocol-following) nodes make the same decisions**

# What is having consensus?

Having consensus is having the protocol in a state that guarantees that...

All protocol-following nodes will make the same decision!

# What is safety?

**Safety is the property that all protocol-following nodes make the same decision, if/when they do make a decision**

# What is liveness?

**Liveness is the property that all protocol-following nodes are *guaranteed* to *eventually* make a decision.**

# What is an asynchronous network?

**The protocol has no assumptions about the reliability of the network.**

# What is an asynchronous network?

**Communications can arrive in any (causally consistent) order!**

**(Usually we do assume that they /eventually/ arrive.)**

# Asynchronous consensus is difficult!

The FLP impossibility theorem shows us that:

It's impossible to be live and safe in an asynchronous network (if communications can fail).

# What is a Byzantine fault?

Any node that is not protocol-following is called "Byzantine."

Byzantine nodes have arbitrary behaviour!

# Byzantine Fault Tolerant Consensus is also Difficult!

**Well known results:**

**Consensus safety can't tolerate 1/3 Byzantine faults (or more) in asynchronous networks**

**Consensus safety can't tolerate at most ½ Byzantine faults (or more) in synchronous networks**

# Correct-by-Construction Binary, Asynchronous Casper

- Approach Outline
- Data structures
- Definitions
- Correct-by-construction (safe) binary decisions

# Preface to Approach Outline

## Introducing *estimates*,
## he predecessors of *decisions*

## Estimates are
## “non-finalized decisions” or
## “decision proposals”

# Preface to Approach Outline

**Blockchains traditionally make estimates rather than decisions.**

**Only blockchains with “finality” make decisions.**

# Approach Outline

- Define safety of estimates

- Construct an ideal adversary who will not be able to induce nodes to change their estimates if they are safe

- Decide on an estimate when the ideal adversary fails to produce an attack on that estimate (in that view)

# Approach Motivation

**Determining the safety of decisions is hard because it's defined with respect to the decisions of *other* correct nodes**

# Approach Motivation

**(Required) Result:**
**If two nodes calculate that they have safe estimates… they must have the same estimate!**

# Approach Motivation

**This result guarantees that our decision rule (decide on an estimate when the estimate is safe) is safe by construction!!!**

# Data structures: Bets

## Bets are a triple:

## (estimate, justification, sender)

$$B = \{0,1\} \times P(B) \times V$$

**or**

$$B = \{0,1\} \times \{\} \times V$$

# Data structures: Validators

Validators are
a fixed subset of the names in V

Validators "have weights"
in (0, infinity)

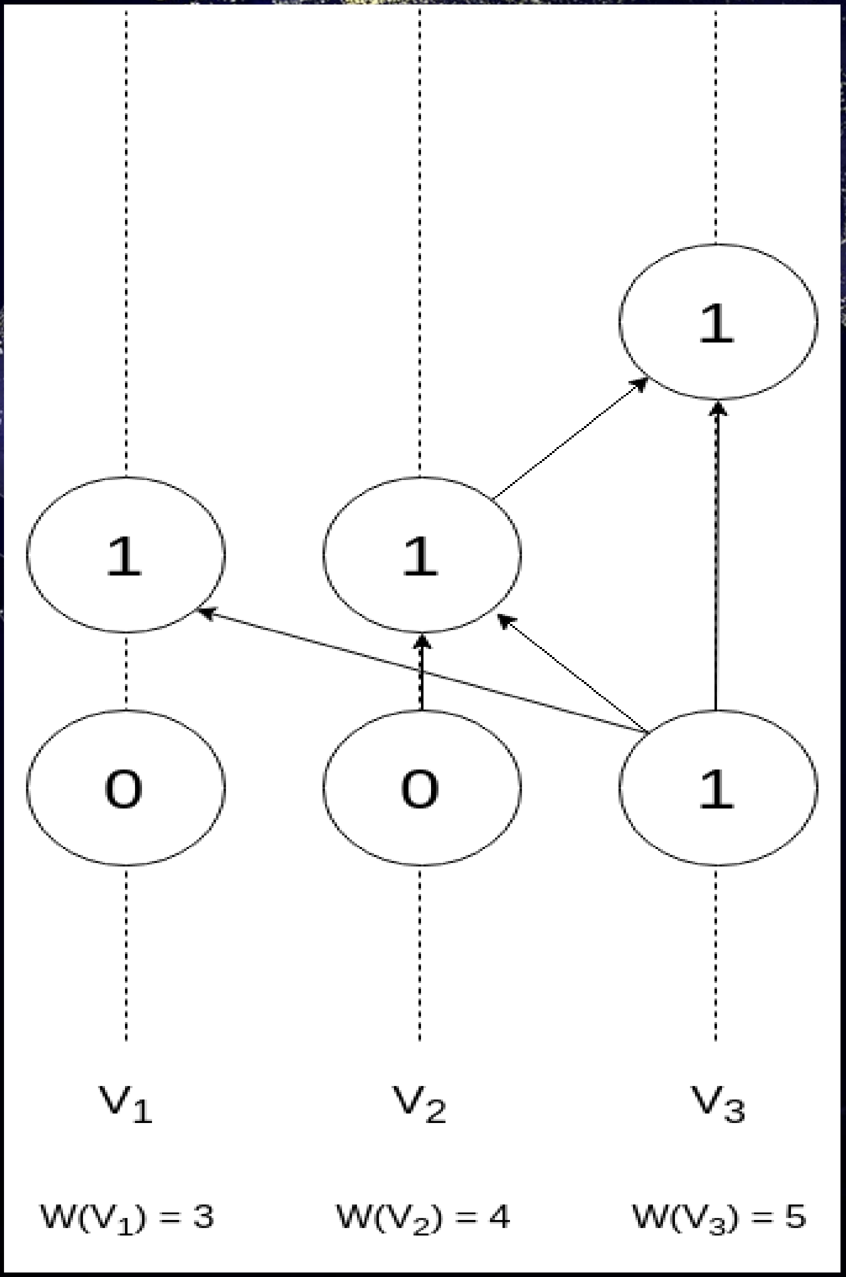
The weights have the
"tie-breaking property"

# Data structures: Views

## Views are sets of bets:

$$U = P(B)$$

# Data structures



# Definitions: Dependency

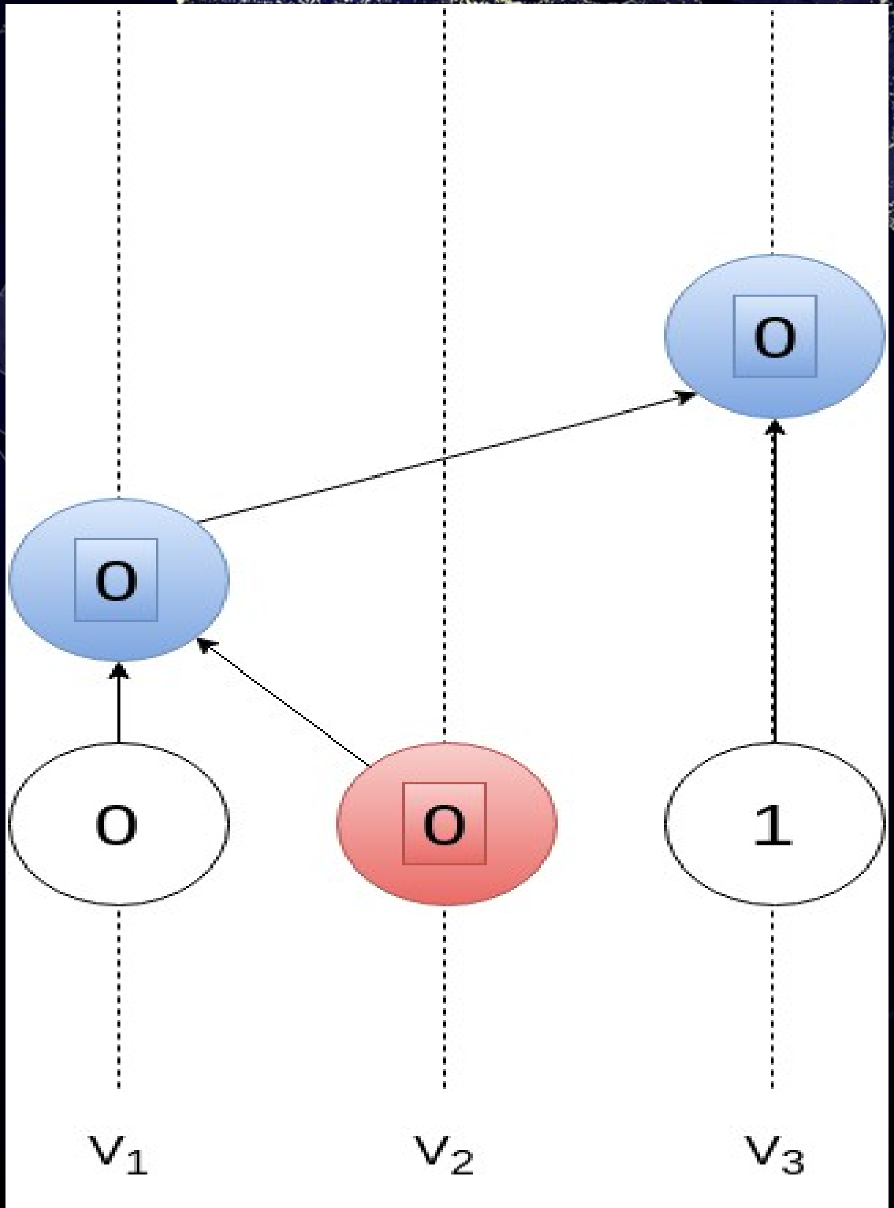
**bet A is a dependency of bet B if:**

**A is in justification(B)**

**OR**

**A is a dependency of C in justification(B)**

# Definitions: Dependency

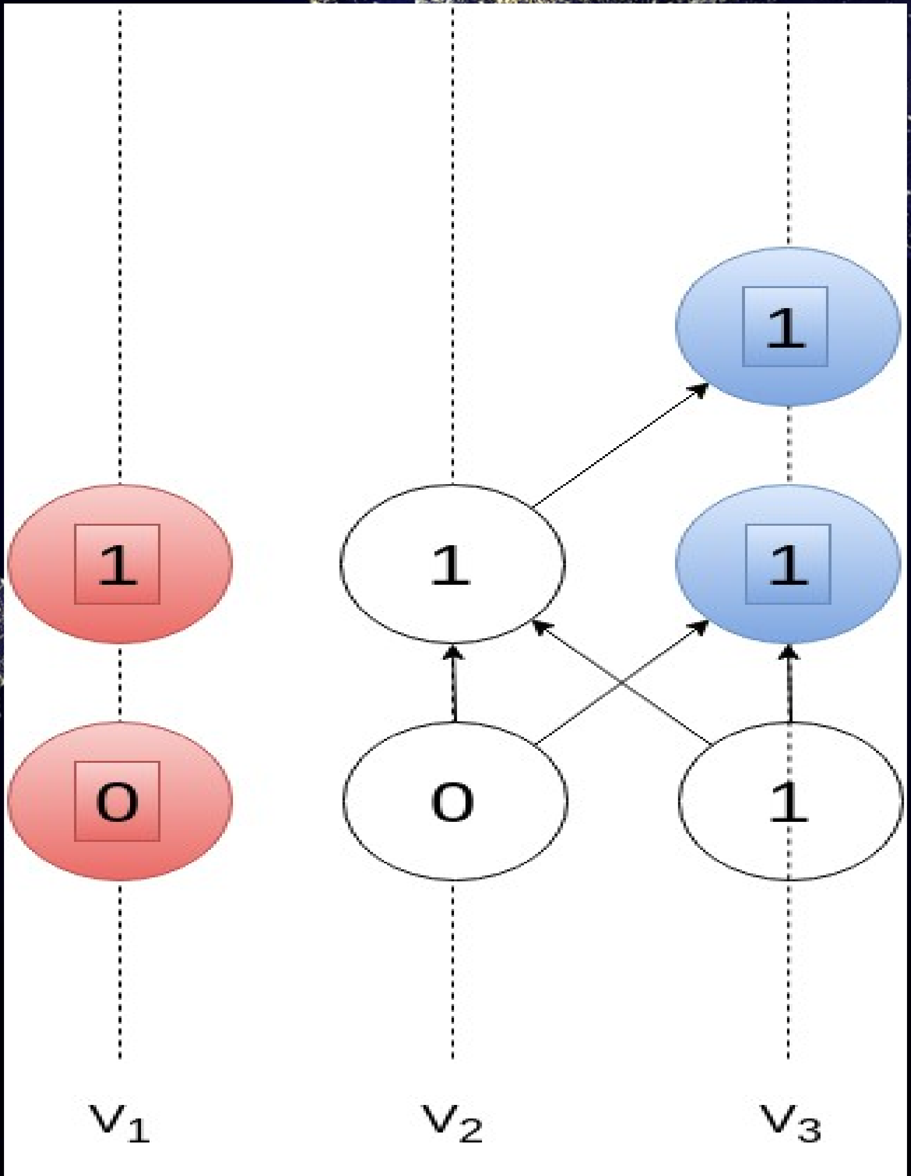


# Definitions: Equivocation

**bets A and B are an equivocation if:**

**A.sender = B.sender**
**A =/= B**
**A not dependency of B**
**B not dependency of A**

# Definitions: Equivocation

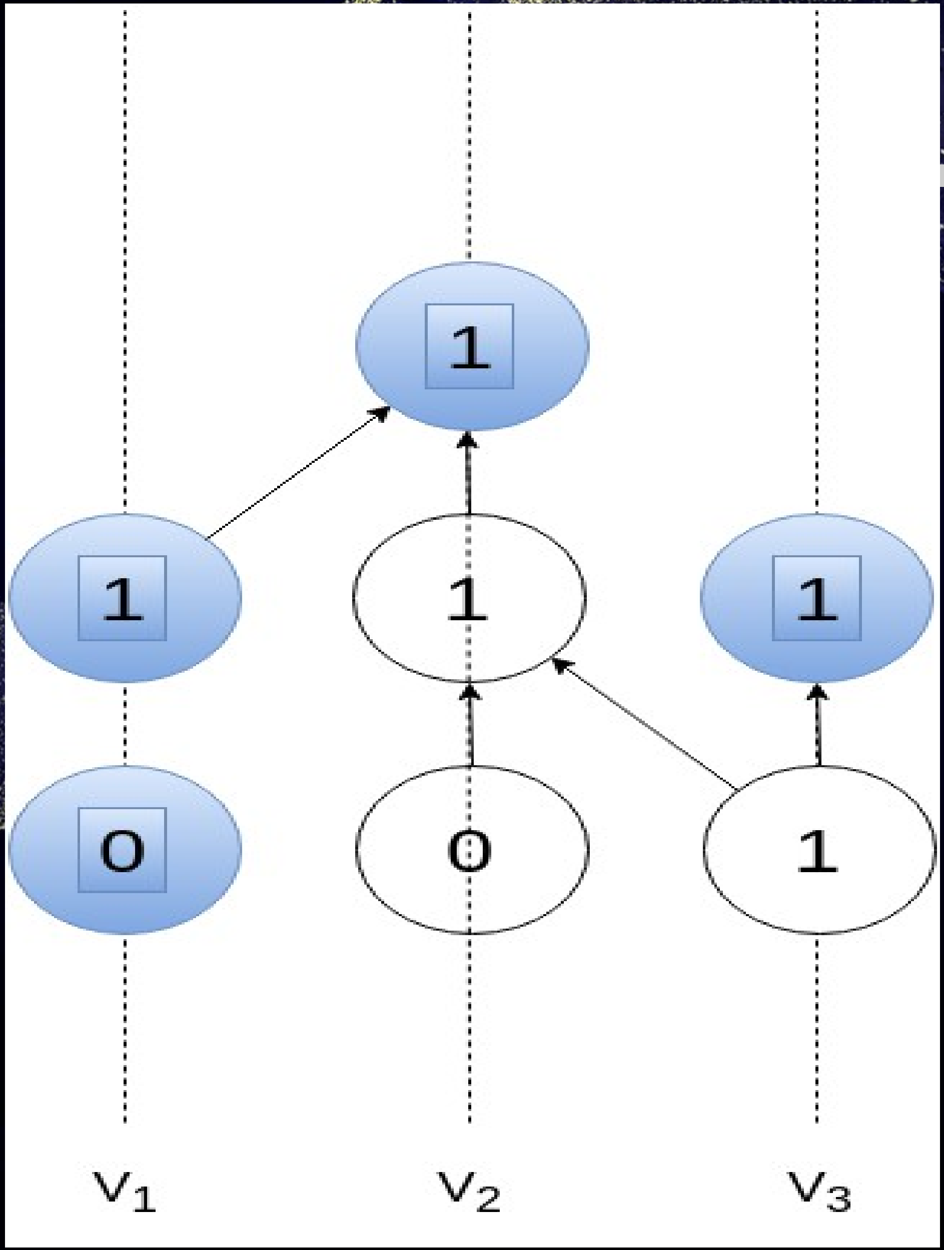


# Definitions: Latest Bets

## Bets are latest in a view…

## …if they are not in the dependency of other bets in that view!

# Definitions: Latest Bets



# Definitions: Invalid* Bets

**Bets are invalid* if**

**their estimate is NOT the "max-weight estimate"**

**in the estimates from latest bets in view given by the bet's justification (weighted by the sender's weight)**

# Definitions: Byzantine Validator

A validator is **Byzantine** in a view if in that view they:

- Produced an invalid* bet

OR

- Equivocated

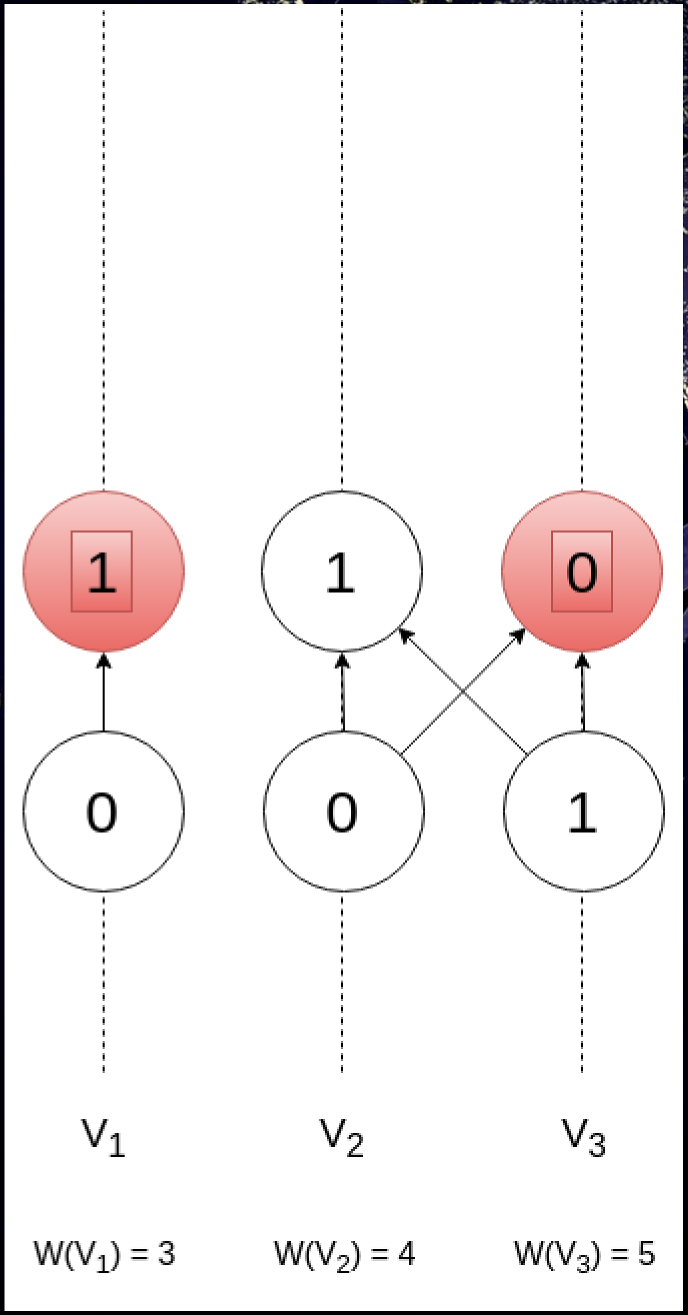
# Definitions: Invalid Bets

**Bets are invalid if**

**their estimate is the "Byzantine-free max-weight estimate"**

**in the latest bets in a view given by the bet's justification...**

# Definitions: Invalid Bets



# Definitions:
# Safety of Estimate

n estimate is safe in a view

n an asynchronous network
given a set of nodes marked
“Byzantine” if…

t is the Byzantine-free max-weight
estimate

# Definitions:
# Safety of Estimate

... and there is no “possible future” of this view where only the nodes marked “Byzantine” are observed to be Byzantine that has a **different canonical estimate**,

# Problem: Calculating safety.

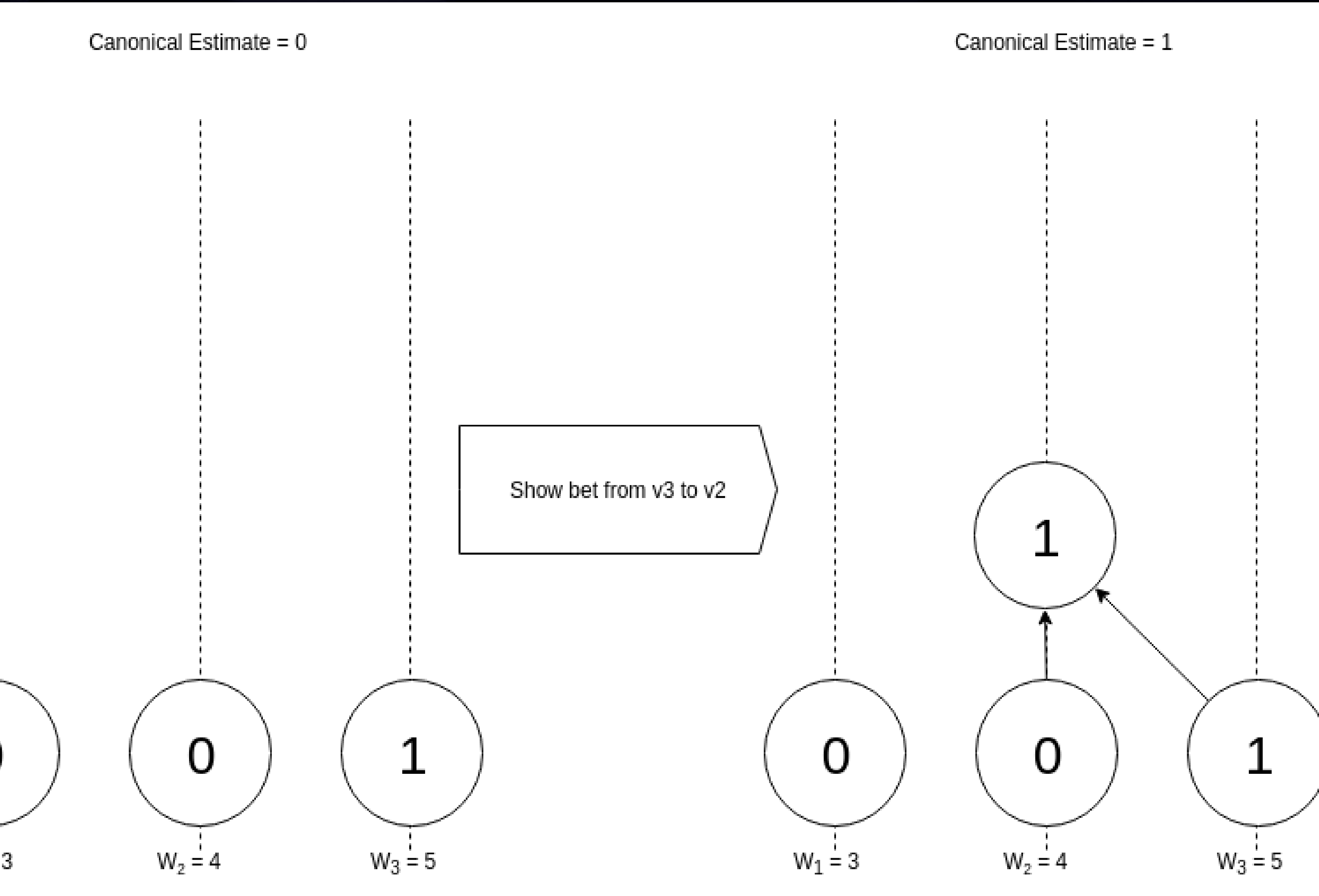
## Our definition is non-constructive

## nd the set of possible futures is large

# Problem: Calculating safety.

So we constructed an ideal adversary, which searches for a "possible future that changes the estimate"

by attacking the estimate in a view through the addition of new latest bets to that view


Canonical Estimate = 0
Canonical Estimate = 1
Show bet from v3 to v2
0
0
1
0
0
1
1
3
$W_2 = 4$
$W_3 = 5$
$W_1 = 3$
$W_2 = 4$
$W_3 = 5$

# Problem: Side effects

When the attacker shows a bet b1 from v1 to v2...

...they may introduce to v2's view a latest bet from v3 =/= v1

# Solution: Ignore side effects

# Now the ideal attacker is roviding a lower bound on safety

## f it fails to find an attack, we're safe

## it succeeds, we might be safe but we might not be safe

# Constructing the ideal attacker for a network-only attack (no equivocations)

- only add bets that don't have the victim estimate

# Constructing the ideal attacker for a network-only attack (no equivocations)

- only allow bets that don't have the victim estimate to cross the network

# afe-by-construction decision rule
# for asynchronous networks

**Protocol-following validators**
**mulate the ideal attacker on**
**heir estimate in their views..**

## afe-by-construction decision rule for asynchronous networks

… and if the ideal
ttacker fails to produce
an attack, then they
ecide on that estimate.

he ideal adversary for a network
ttack with equivocation faults is
surprisingly similar.

She adds and shows only bets
who don't have the victim
estimate...

...but she is able to add bets in
new places” for Byzantine nodes
(unf. no time for more details!
Ask me later!)

# Relation of this research to traditional consensus research.

## FLP Impossibility:

**his protocol provides safety and ault tolerance, but not liveness in an asynchronous network**

**The approach gives intuition on why FLP impossibility is a result:**

**An unsafe estimate cannot ecome safe in the presence of ar ideal asynchronous network attack (which would trivially prevent progress from being made!)**

# Safety, Liveness, and Byzantine faults:

Our approach focused exclusively on the ex-post measurement safety of estimates in views.

We have done nothing to reason about the liveness of the protocol, at all.

# Safety, Liveness, and Byzantine faults:

**There exist views where an estimate is safe against all-but-one nodes equivocating!**

**This is very safe!**

# Safety, Liveness, and Byzantine faults:

However, if all-but-one nodes are Byzantine, then there is no way to guarantee a transition to this state from lack of safety!

This is not live!

## Safety, Liveness, and Byzantine faults:

**Our approach provides an interesting view into the safety of Byzantine fault tolerant, asynchronous consensus protocols!**

**We aim to extend this to liveness!**

# Future Work

**Cover liveness with formal reatment in the same model we used to define *everything*.**

# Future Work

**Construct a conservative ideal equivocation attacker**

**One that equivocates with the inimum weight required to conduct a successful attack**

# Future Work

love from consensus on a bit
to consensus on the EVM

# Future Work

## Add validator rotation

# Future Work

## Add the economic security mechanisms

## So we can run Casper as a public consensus protocol

# Future Work

## Complexity and performance optimization

# Future Work

**mprove theory, specification, documentation and implementation of the correct-by-construction Casper**

**Complexity and performance**

# Thanks for listening!

<3 Vlad

```
ck unsuccessful...
 Byzantine fault context: set([])
-------------------------------------------------------


}, 1), (1, {}, 0)}, 1)



ck successful...
 Byzantine fault context: set([])
operations_log:
1), (1, {}, 0)}, 0)
1), (1, {}, 2), (0, {(0, {}, 1), (1, {}, 0)}, 0)}, 2)

-------------------------------------------------------





ck successful...
 Byzantine fault context: set([1])
operations_log:
(0, {}, 1) for validator 0
1), (1, {}, 0)}, 0)
(0, {}, 1) for validator 2
1), (1, {}, 2), (0, {(0, {}, 1), (1, {}, 0)}, 0)}, 2)

-------------------------------------------------------



}, 2), (1, {}, 0)}, 0)
}, 2), (1, {}, 0)}, 2)
}, 2), (1, {}, 0)}, 1)


ck unsuccessful...
 Byzantine fault context: set([2, 3])
amming/python/Casper$
```