Introduction to Cryptoeconomics

What is cryptoeconomics?

Cryptoeconomics is about...

- Building systems that have certain desired properties
- Use **cryptography** to prove properties about messages that happened *in the past*
- Use **economic** incentives defined inside the system to encourage desired properties to hold *into the future*

Bitcoin: Desired properties

- Create a chain of blocks
- Include transactions in each block
- Maintain a "state" (UTXO set)
    - Transactions affect state: `s' = STF(s, tx)`
- Maintain a clock

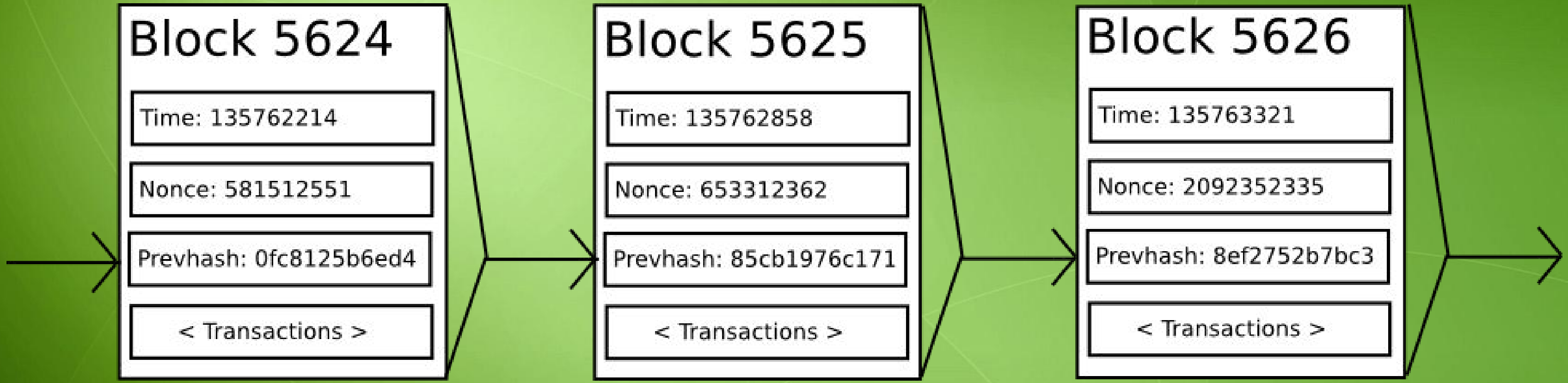
Bitcoin: Desired properties

- **Convergence**: new blocks can be added to the chain but blocks cannot be replaced or removed
- **Validity**:
  - Only txs that satisfy a predicate `VALID(s, tx)` with respect to the state at time of execution should be included in a block
  - Clock should be roughly increasing

Bitcoin: Desired properties

- **Data availability:** it should be possible to download full data associated with a block
- **Availability**: transactions should be able to get quickly included if they pay a reasonably high fee

Bitcoin: Desired properties



Bitcoin: Cryptography

- Proof of work
- Signatures (prove tx sender authenticity)
- Hashes
  - Ensure consistent total ordering of chain
  - Enable (limited) light client protocol via Merkle proofs

Bitcoin: Incentives

- Miner of block that gets into the chain gets 12.5 BTC, plus can extract rent from being “temporary dictator” over tx inclusion
- Miner of block that does not get into the chain gets nothing
- Difficulty adjustment: rewards are marginally long-run zero sum

Cryptography

Uses of cryptography

- **Hashes**: prove *topological order* of messages
- **Signatures**: prove the *identity of the sender* of a message
- **ZKPs**: prove arbitrary computable predicates on messages

More crypto

- **Proof of work:** prove that a certain amount of expected computational effort was expended
- **Erasure codes**: convert a 100% data availability requirement into a 50% data availability requirement
- **Timelock crypto / sequential PoW**: prove that some amount of time elapsed between messages A and B
- **Homomorphic encryption / obfuscation**: convert functions into isomorphic functions that are more privacy-preserving

Economics

Incentives

- **Tokens:** incentivize actors by assigning them units of a protocol-defined cryptocurrency
    - eg. block rewards
- **Privileges**: incentivize actors by giving them decision-making rights that can be used to extract rent
    - eg. transaction fees

Incentives

- **Rewards:** increase actors’ token balances or give them privileges if they do something good
- **Penalties**: reduce actors’ token balances or give them privileges if they do something bad

Concepts

- **Cryptoeconomic security margin:** an amount of money X such that you can prove "either a given guarantee G is satisfied, or those at fault for violating G are poorer than they otherwise would have been by at least X"
- **Cryptoeconomic proof**: a message signed by an actor that can be interpreted as "I certify that either P is true, or I suffer an economic loss of size X"

Concepts

- **Uncoordinated choice model:** a model that assumes that all participants in a protocol do not coordinate with each other and have separate incentives, and are all smaller than size X
- **Coordinated choice model**: a model that assumes that all actors in a protocol are controlled by the same agent (or coalition)

Concepts

- **Bribing attacker model:** a model that starts off with an uncoordinated choice assumption, but also assumes that there is an attacker capable of making payments to actors conditional of them taking certain actions
  - **Budget**: the amount that the briber must be willing to pay in order to execute a particular strategy
  - **Cost**: the amount that the briber actually does pay if the strategy succeeds

Example: Schellingcoin

- Property: provide the “true answer” to a given question
  - eg. who won the election?
- Algorithm:
  - Everyone votes A or B
  - Majority answer is taken as correct
  - Everyone who voted with majority given reward of P, all others get nothing

Example: Schellingcoin

- Uncoordinated choice: you have the incentive to vote the truth, because everyone else will vote the truth and you only get a reward of P if you agree with them
- Why will everyone else vote the truth? Because they are reasoning in the same way that you are!

P + epsilon attack

A bribing attacker can corrupt the Schellingcoin game with a **budget** of P + ε and zero **cost**!

Base game:

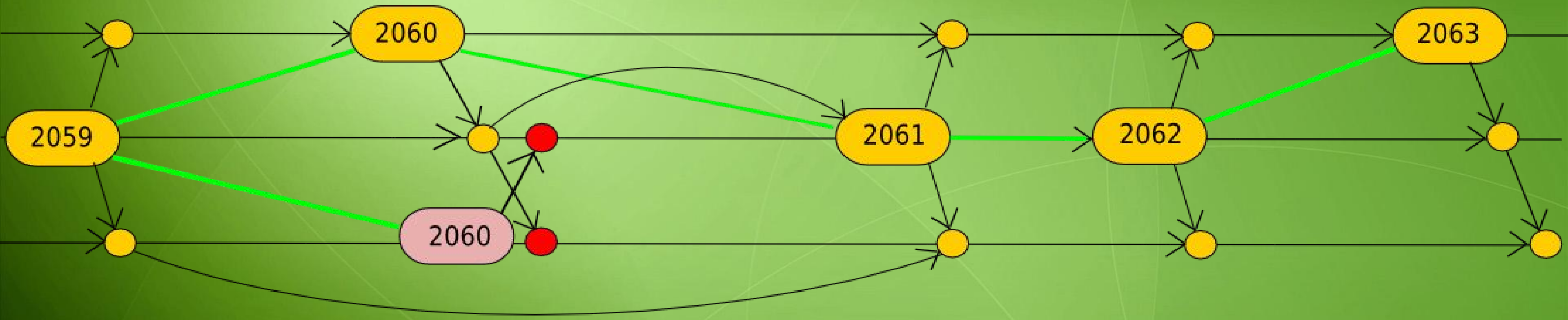
	You vote 0	You vote 1
Others vote 0	P	0
Others vote 1	0	P

With bribe:

	You vote 0	You vote 1
Others vote 0	P	P + ε
Others vote 1	0	P

Example: proof of work

- You get in the main chain, 12.5 BTC reward
- You do not get in the main chain, no reward.



- Same strategy as P+epsilon attack can be used in the bribing attacker model!

Griefing factors

- Even in an incentive-compatible protocol, there will almost always be opportunities to, at some cost to yourself, impose costs on others
- If it costs you $1 to harm someone else by $X, that’s a griefing factor of X

Griefing factors

- Griefing factors depend on:
  - Model (coordinated choice vs uncoordinated vs bribing)
  - Size of an actor
  - How often the griefing opportunities appear

Faults

- We can view faults at several levels:
  - Faults of the **protocol**, ie. the protocol not perfectly satisfying its desired properties
  - Faults of **individual actors** in the protocol
  - Faults of the **network**

Faults

- It is in many cases easy to measure protocol faults
  - Blockchain case: stale/uncle rate
- In some cases it’s possible with qualifications
  - Timestamps
- In some cases it’s not possible directly
  - Censorship (selectively denying transactions the ability to get included)

Categorization of faults

- Define a protocol as a function `P(M, aux) => msg`, where:
  - `M` is the set of protocol messages already received
  - aux is auxiliary data (eg. clock, real-world knowledge)
  - `msg` is the message that the node sends

Categorization of faults

- **Invalidity**: a message is not the result of `P(M, aux)` for any aux and any subset of the messages that the node saw
- **Equivocation**: two messages m1, m2 where:
  - `m1 = P(M1, aux1)`
  - `m2 = P(M2, aux2)`

Where M2 does not contain M1 + m1, and M1 does not contain M2 + m2

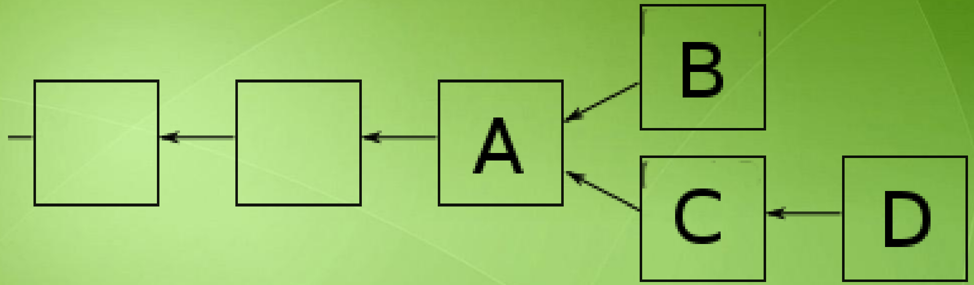
Categorization of faults

- **Ignoring/delaying inputs:** (consistently) pretending that some message that actually arrived at time T1 did not arrive until some later time T2 (possibly $T2 = \infty$)
- **Not sending/delaying outputs**: not sending a given message, or sending it later than intended
- Using false values of aux
    - Special case: **sending messages too early**
- **Network faults** (latency, dropping messages)

Fault assignment

- Given a protocol fault, we can often narrow down why the fault took place, at least to within one of several causes

Example: blockchain fork

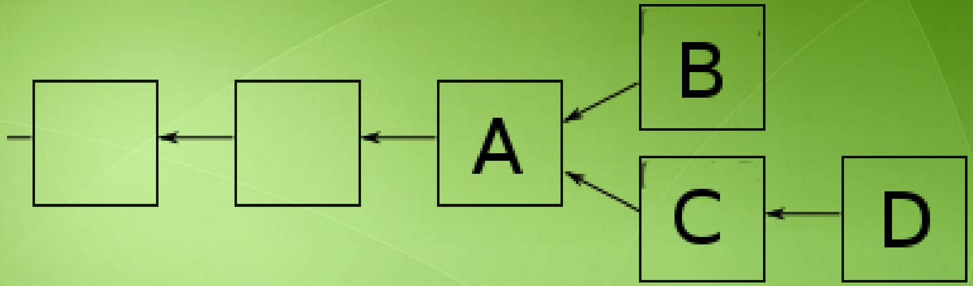


- Case 1: B ignored C and/or D
- Case 2: C ignored B
- Case 3: C did not send to B
- Case 4: B did not send to C
- Case 5: network fault

Principles of penalty assignment

1. **Maximum penalty upon conviction**: If you can unambiguously prove that one specific party is faulty, penalize that party maximally
2. **If you can't choose, penalize all (slightly)**: if fault assignment tells you that one of N parties are faulty, penalize all N (though not as much)
3. **Pay for performance**: total rewards should be an increasing function of some metric of "protocol quality"

Penalty assignment



- Example reward assignment:
  - A: +1
  - B: 0 (or -1 in a PoS protocol)
  - C: 0 (or -1 in a PoS protocol)
  - D: +1

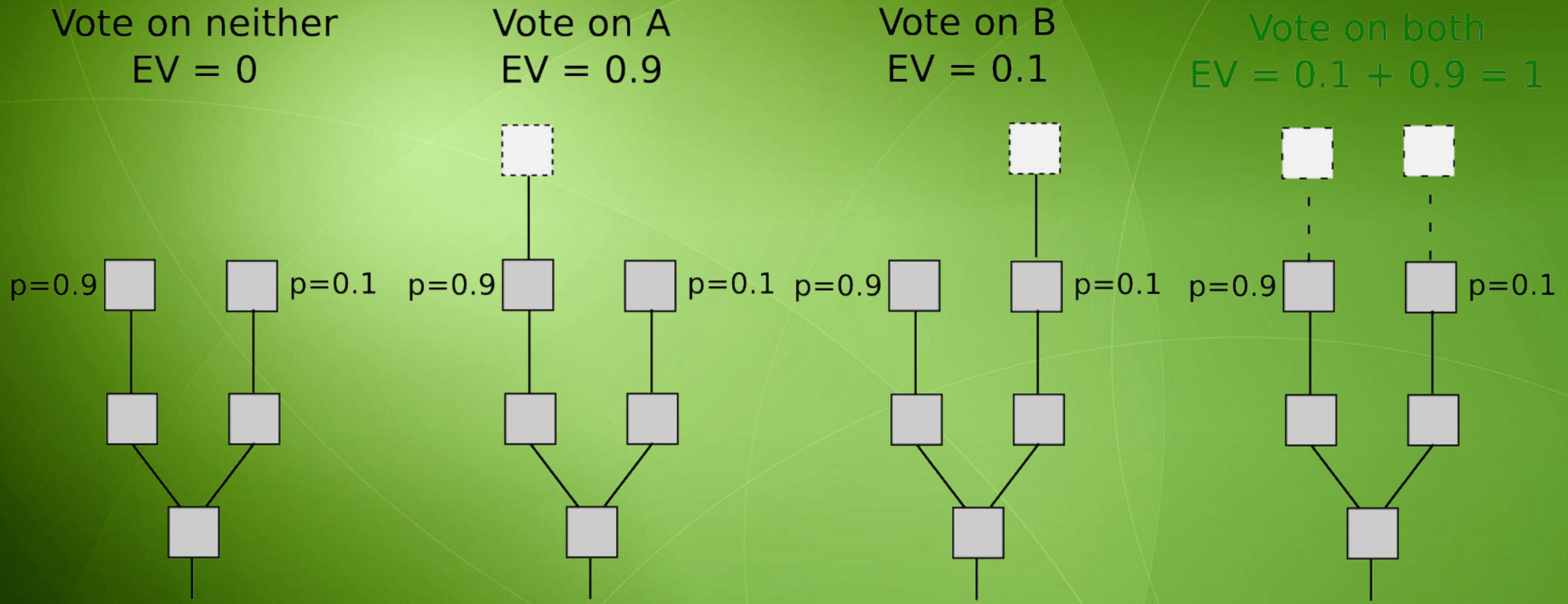
Benefits

- We know that if someone is faulty in a way that causes protocol failure, they will be punished
- Innocents may be punished too; bounding griefing factors is a good way to analyze to make sure this is not too much of a problem
- Selfish mining issues are much easier to resolve

Proof of stake

- Relies on a consensus algorithm using signatures signed by **bonded validators**
- Properties
    - Safety (finalized blocks don’t get un-finalized)
    - Liveness (as in proof of work)
- Cryptoeconomic security margin very high because we can rely on penalties, and not just rewards

Naive PoS: nothing at stake



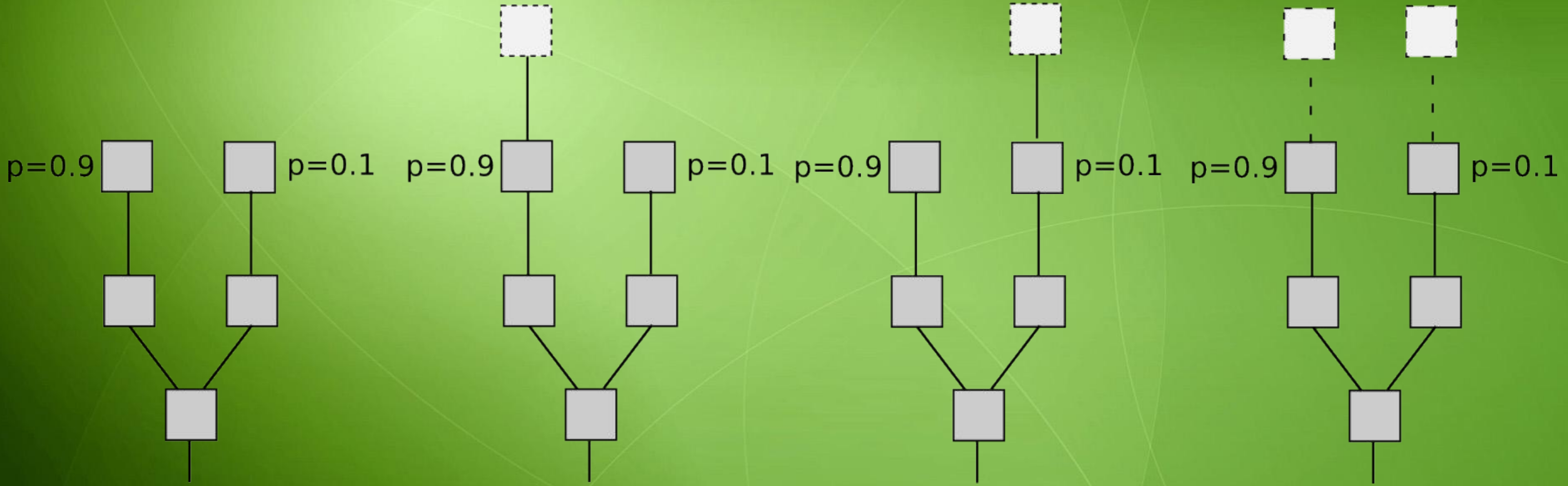
Slasher: solution 1 (penalize equivocation)

Vote on neither
EV = 0

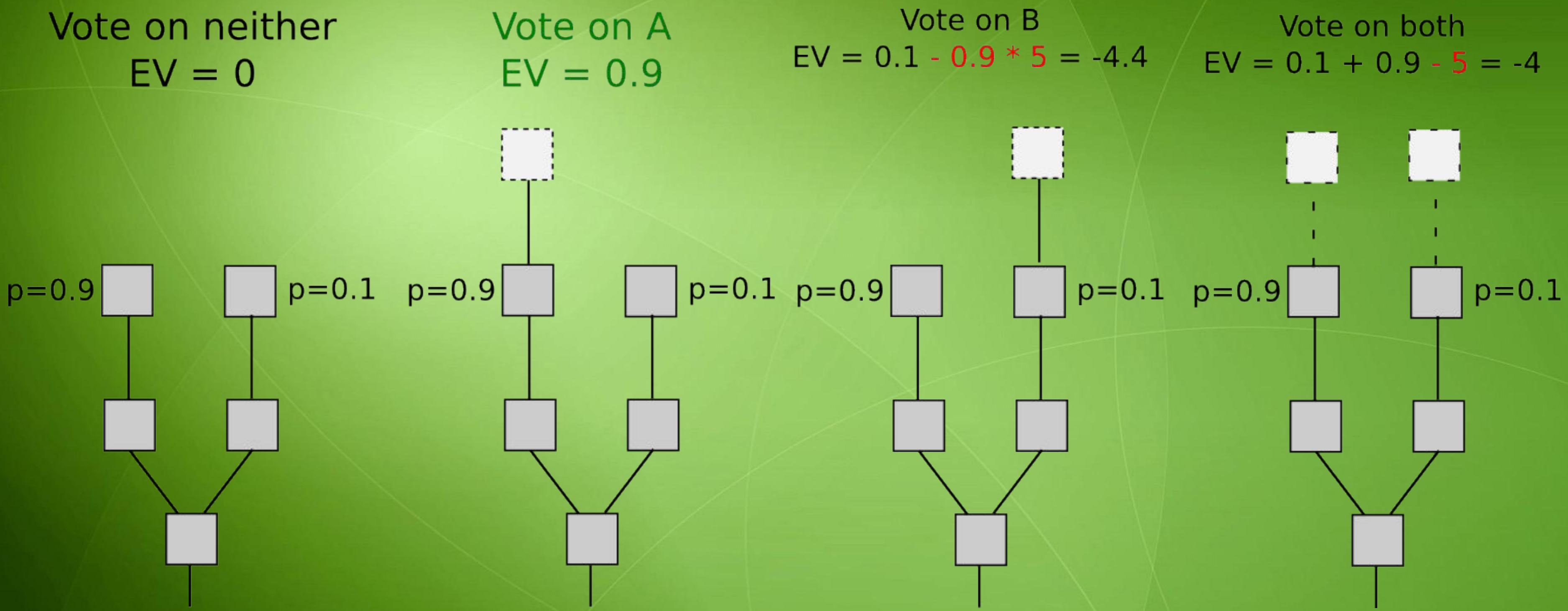
Vote on A
EV = 0.9

Vote on B
EV = 0.1

Vote on both
EV = 0.1 + 0.9 - 5 = -4



Slasher: solution 2 (penalize being wrong)



Auditable safety

- Cryptoeconomic version of BFT safety
- Definition: if conflicting values A and B are both finalized, then:
    - At least ⅓ of bonded validators were faulty
    - We know whom to blame

Auditable safety ~= asynchronous BFT safety

- Invalidity and equivocation are provable faults
- Message delays and lying about timestamps indistinguishable from latency
  - Sending/receiving messages too late ~= network delay
  - Sending messages too early ~= “stretch” all clocks by factor M, explain all newly created discrepancies with network delay

Plausible liveness

- Cryptoeconomic version of BFT liveness
- Definition: unless there already have been ⅓ provable faults, there exists a set of messages that ⅔ of validators could send to finalize one of a set of options

Plausible liveness ~= partially synchronous BFT liveness

- Same logic as before: all faults are either provable or indistinguishable from latency
- At any point, nodes can cease to be faulty (ie. latency back to normal), and finalize a block

Data availability

- Important in scalable/sharded blockchains, where individual nodes cannot download all data for themselves
- If data is available, then proving **correctness** is easy via interactive protocols
- But if data is **unavailable**, it is harder

Data availability

- Problem: you cannot unambiguously show fault
- The following are indistinguishable:
  - **Case 1**: node X published data D at time T3 instead of T1, node Y correctly notified you at time T2
  - **Case 2**: node X published data D at time T1, node Y lied at time T2 about the data's unavailability

Data availability

- Problem: how do we finalize a state when we can’t personally verify its correctness?
- Computation:
  - zk-SNARKs
  - Interactive games / challenge-response protocols
- Data-availability: ???

Data availability

- **Option 1: honest-minority assumption**
  - 15% of a randomly selected subset of a network can hold data back from being finalized indefinitely
  - We assume that these 15% are not colluding with the other 85% and are not taking bribes
  - If a conflict over whether or not data is available emerges, both publisher and challenger penalized
- **Option 2: erasure coding**

Open problems

- Optimal properties of consensus algorithms
- Censorship resistance
- Maximally accurate timestamping
- Scalable validation
- Optimal data availability solutions