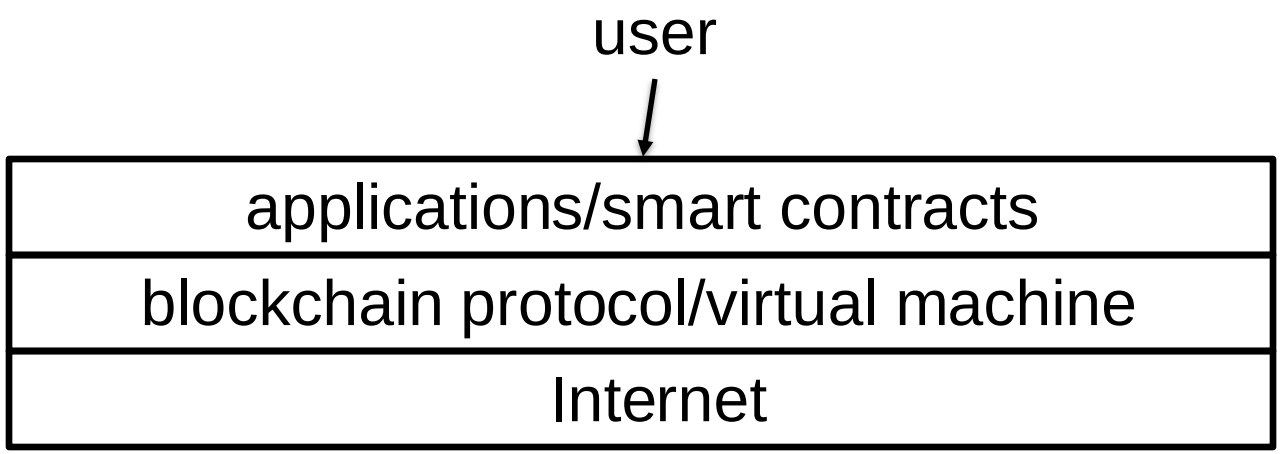
Lecture #2: State Machine Replication

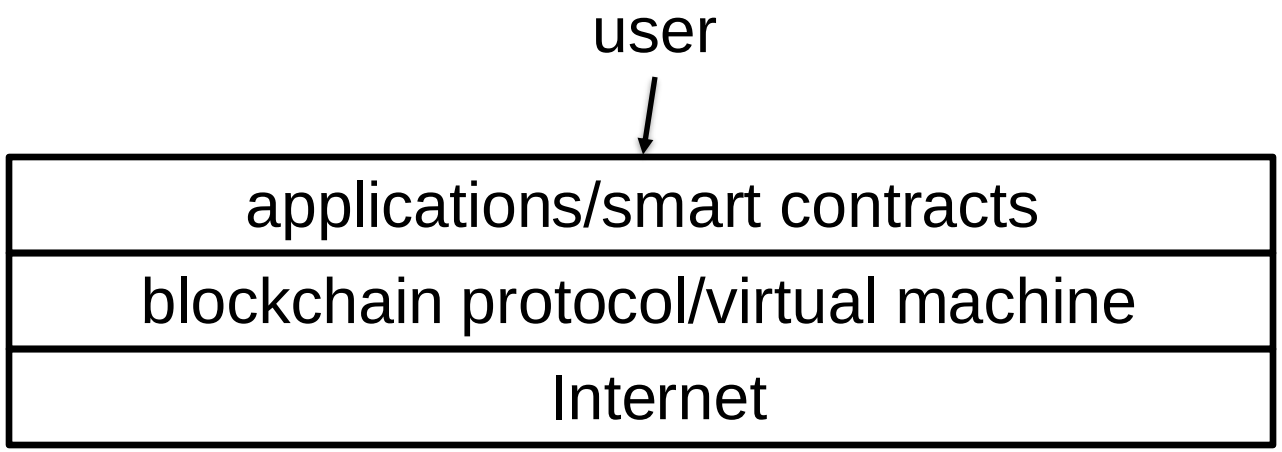
COMS 4995-001:
The Science of Blockchains
URL: https://timroughgarden.org/s25/

Tim Roughgarden

Recap: A Cartoon of Web3



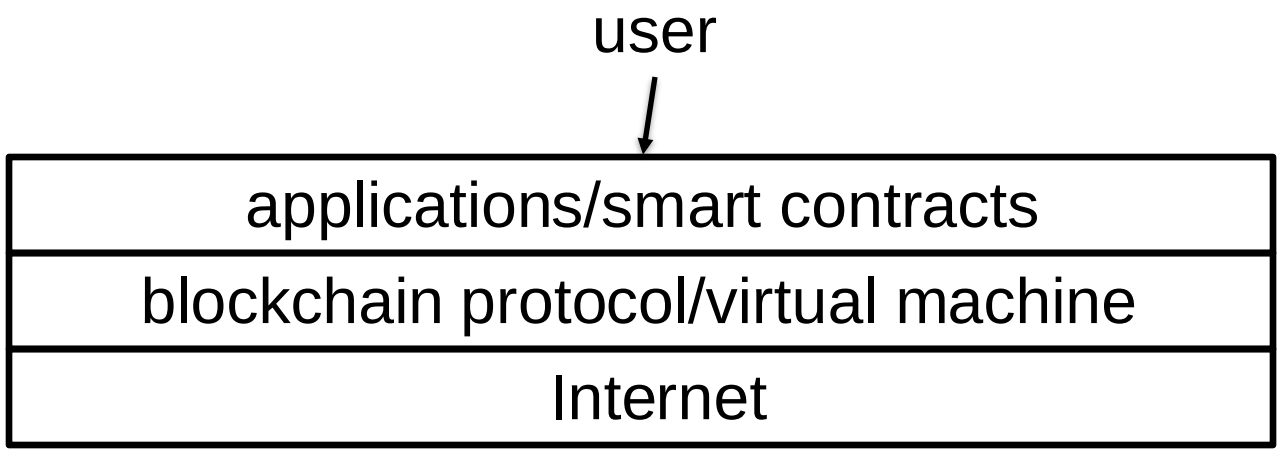
Recap: A Cartoon of Web3



Blockchain protocol:

- like an operating system, a blockchain protocol:
  - acts as a “master program” to coordinate all apps/smart contracts
  - provides a virtual machine to developers of applications

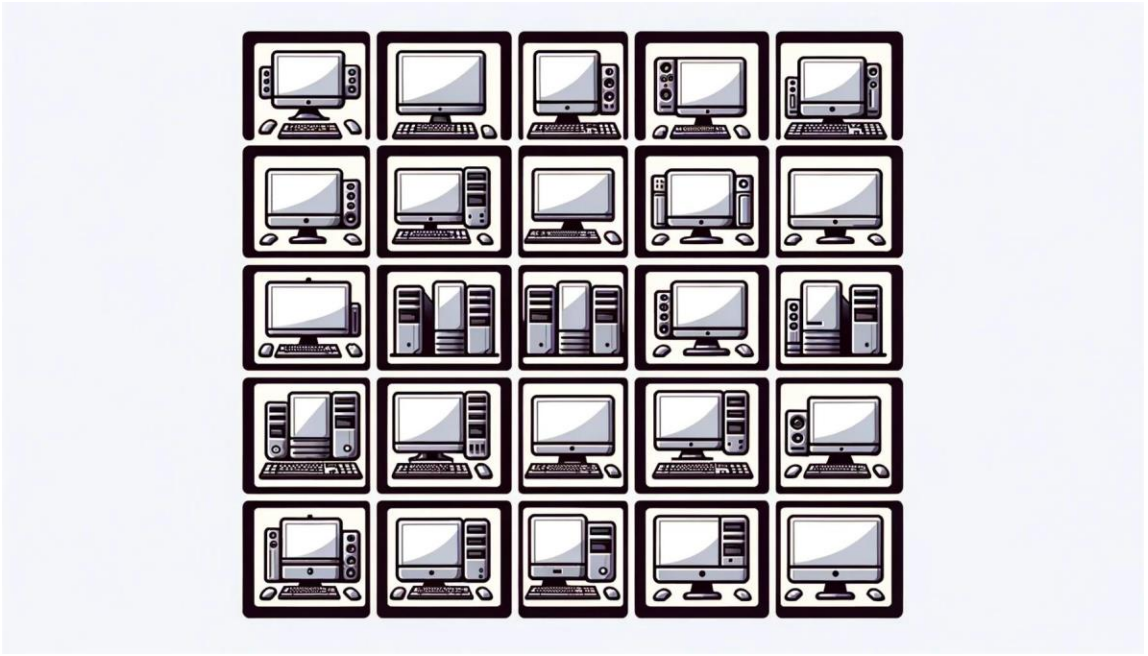
Recap: A Cartoon of Web3



Blockchain protocol:

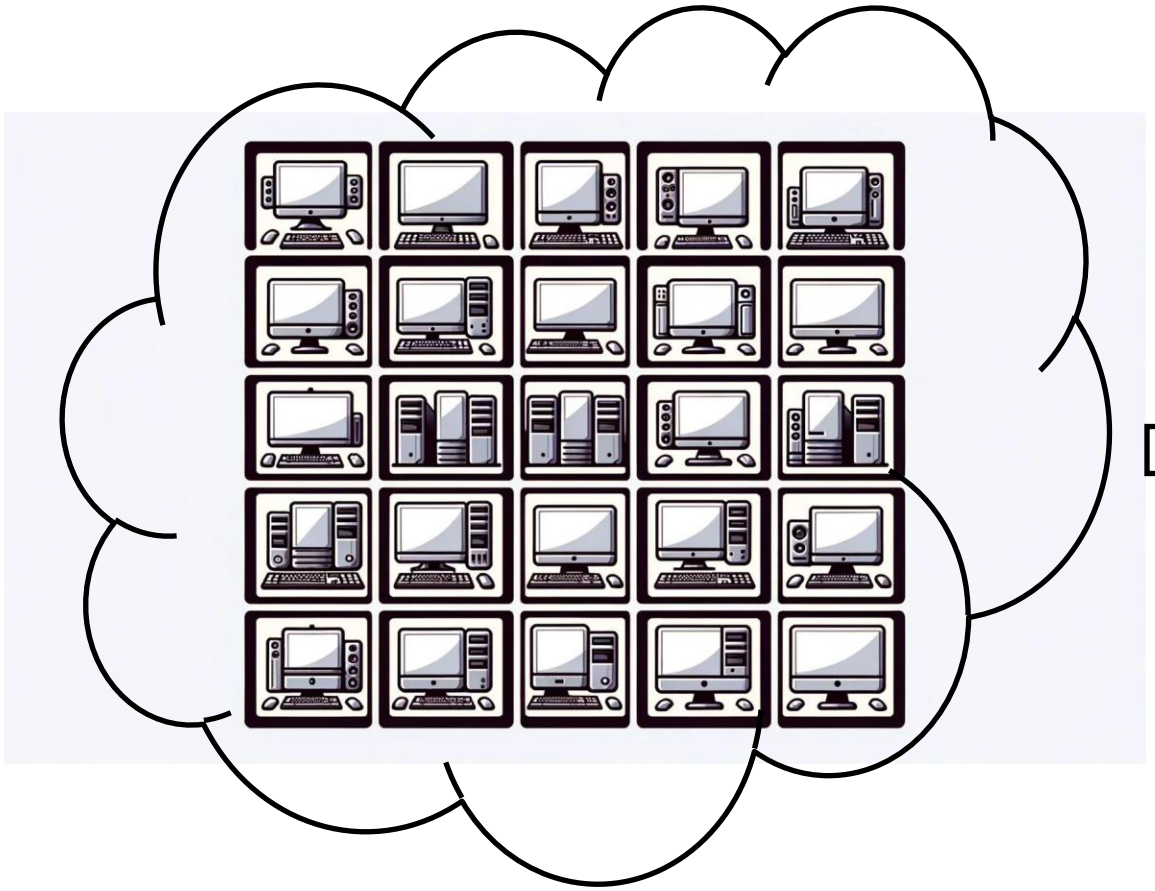
- like an operating system, a blockchain protocol:
  - acts as a “master program” to coordinate all apps/smart contracts
  - provides a virtual machine to developers of applications
- like the Internet, “decentralized” -- the product of collaboration between many physical machines, no one owner/operator

The Computer in the Sky



network of physical computers

The Computer in the Sky



network of physical computers
+ blockchain protocol

simulated (virtual) computer

Goals for Lecture #2

1. Consensus basics: validators, transactions, blocks.

Goals for Lecture #2

1. Consensus basics: validators, transactions, blocks.

2. State machine replication (SMR), consistency, liveness.
   - the problem we need to solve and the basic guarantees that we want

Goals for Lecture #2

1. Consensus basics: validators, transactions, blocks.
2. State machine replication (SMR), consistency, liveness.
   - the problem we need to solve and the basic guarantees that we want
3. Challenges to consensus: faulty validators (crash vs. Byanztine), message delays (synchronous vs. asynchronous).
   - validators, communication network might not behave as expected

Goals for Lecture #2

1. Consensus basics: validators, transactions, blocks.
2. State machine replication (SMR), consistency, liveness.
   - the problem we need to solve and the basic guarantees that we want
3. Challenges to consensus: faulty validators (crash vs. Byanztine), message delays (synchronous vs. asynchronous).
   - validators, communication network might not behave as expected
4. Security thresholds.
   - threshold of faulty validators at which consensus becomes impossible

Some Terminology

Validators: physical machines running a blockchain protocol.

- a.k.a. “nodes” (e.g., 22 or 100, communicating via Internet)

Some Terminology

Validators: physical machines running a blockchain protocol.

- a.k.a. “nodes” (e.g., 22 or 100, communicating via Internet)

Transaction: user-submitted action.

- e.g., make a payment or mint an NFT
- translates to snippet of low-level code to be executed in VM

Some Terminology

Validators: physical machines running a blockchain protocol.

- a.k.a. “nodes” (e.g., 22 or 100, communicating via Internet)

Transaction: user-submitted action.

- e.g., make a payment or mint an NFT
- translates to snippet of low-level code to be executed in VM

Block: sequence of transactions (➔ sequence of VM instructions).

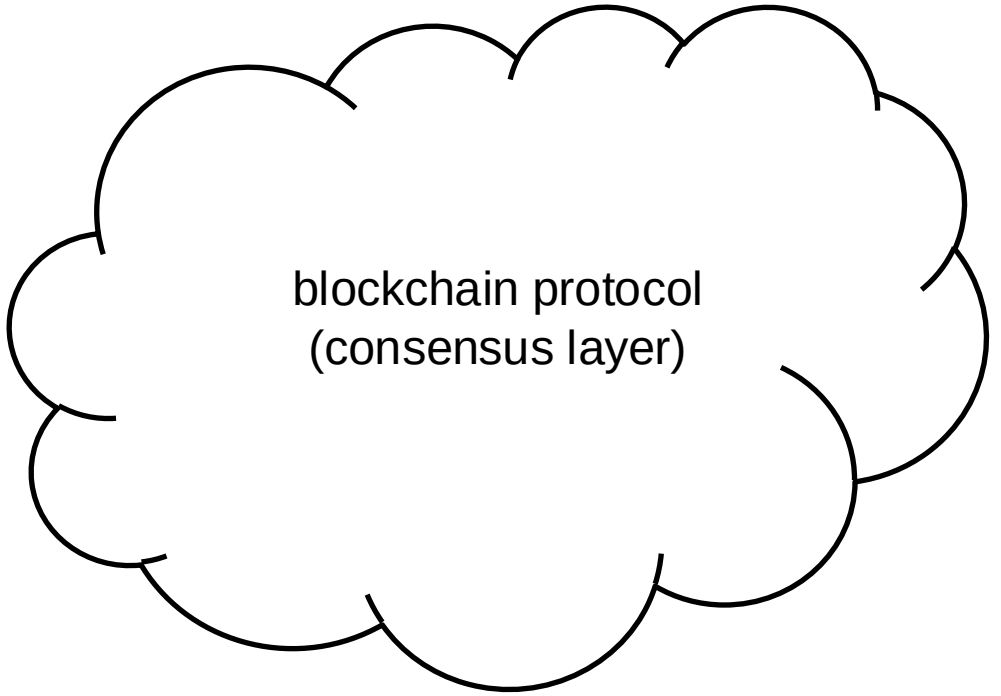
- for now, assume can be of unbounded size

Responsibilities of a Blockchain Protocol

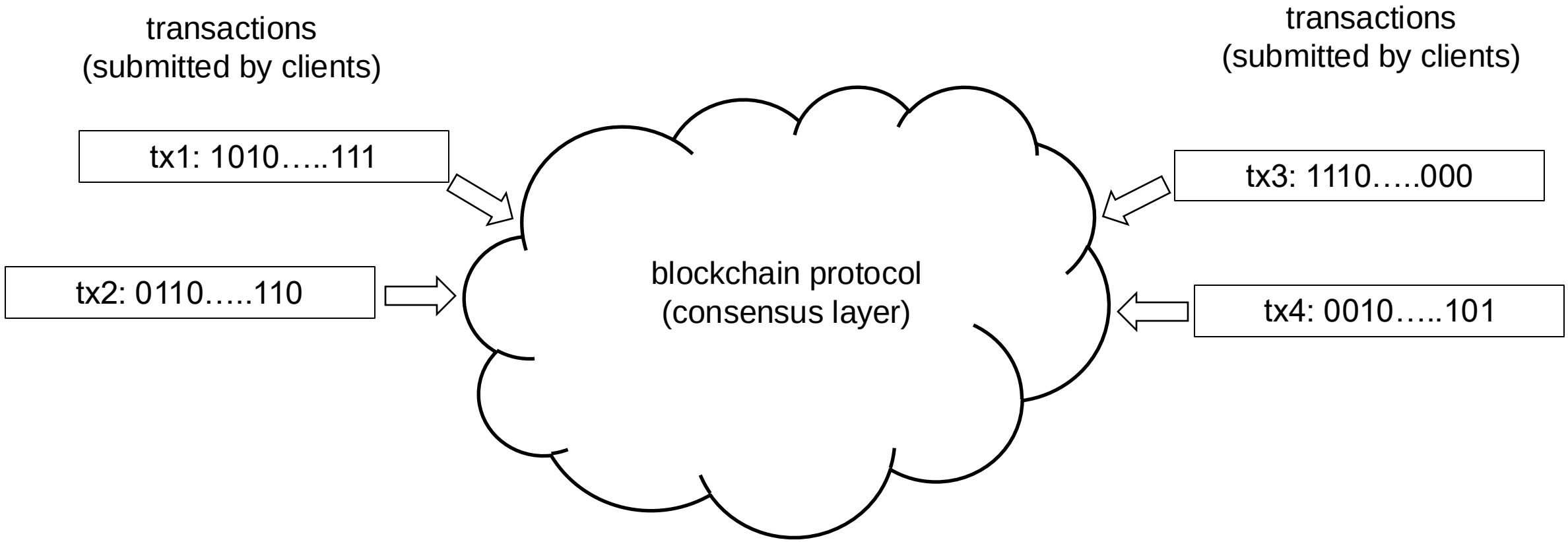
Consensus: decide on a sequence (aka “chain”) of blocks.

- note: all validators must agree on this sequence!
- blocks keeping getting added (one-by-one) as long as there are transactions to process
- not obvious how to do this, subject of next 5 lectures

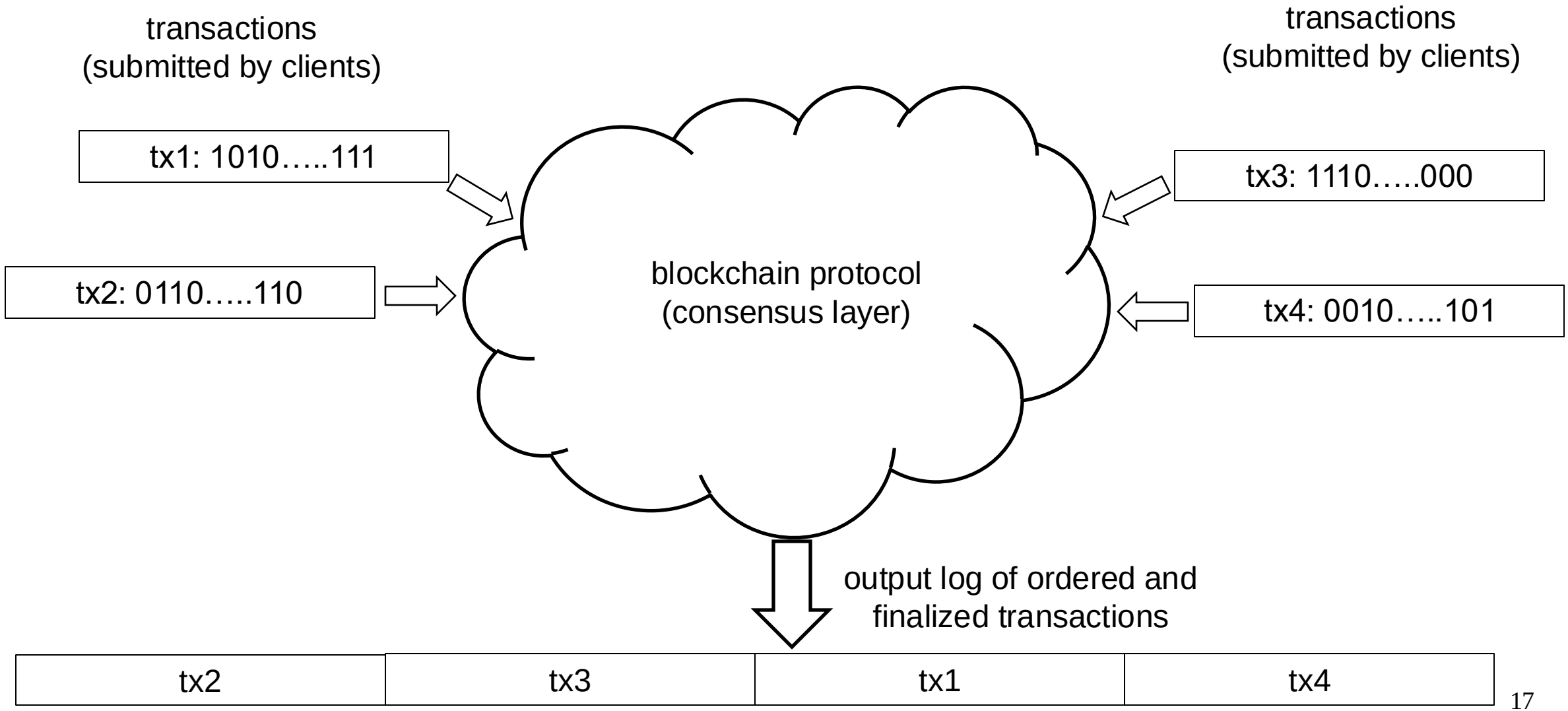
The Consensus Layer



The Consensus Layer



The Consensus Layer



Responsibilities of a Blockchain Protocol

Consensus: decide on a sequence (aka “chain”) of blocks.

- note: all validators must agree on this sequence!
- blocks keeping getting added (one-by-one) as long as there are transactions to process
- not obvious how to do this, subject of next 5 lectures

Execution: keep state of the virtual machine up-to-date.

- new block added ➔ execute the corresponding snippets of code (do computations, update variable values, etc.)
- subject of lectures 8+9 (concludes Part I of course)

Consensus: Getting Started

Consensus: keep validators in sync despite failures and attacks.

- fundamental problem that any blockchain protocol must solve
- glue between the “Internet hardware” and the app-facing VM

Consensus: Getting Started

Consensus: keep validators in sync despite failures and attacks.

- fundamental problem that any blockchain protocol must solve
- glue between the “Internet hardware” and the app-facing VM

Standing assumptions:

1. Fixed and known set of n validators. (E.g., n=22 or 100.)
   - each with known ID and IP address (will communicate over Internet)
   - a.k.a. “permissioned” or “proof of authority” blockchain protocol
   - i.e., randos can’t just join the validator set
   - will relax in Part III of course (”permissionless” protocols including Bitcoin, Ethereum, etc.)

Consensus: Getting Started

Consensus: keep validators in sync despite failures and attacks.

- fundamental problem that any blockchain protocol must solve
- glue between the “Internet hardware” and the app-facing VM

Standing assumptions:

1. Fixed and known set of n validators. (E.g., n=22 or 100.)
   - each with known ID and IP address (will communicate over Internet)
   - a.k.a. “permissioned” or “proof of authority” blockchain protocol
2. All validators have same notion of time (“synchronized clocks”).
   - approximately true in practice, won’t worry about this further

State Machine Replication (SMR)

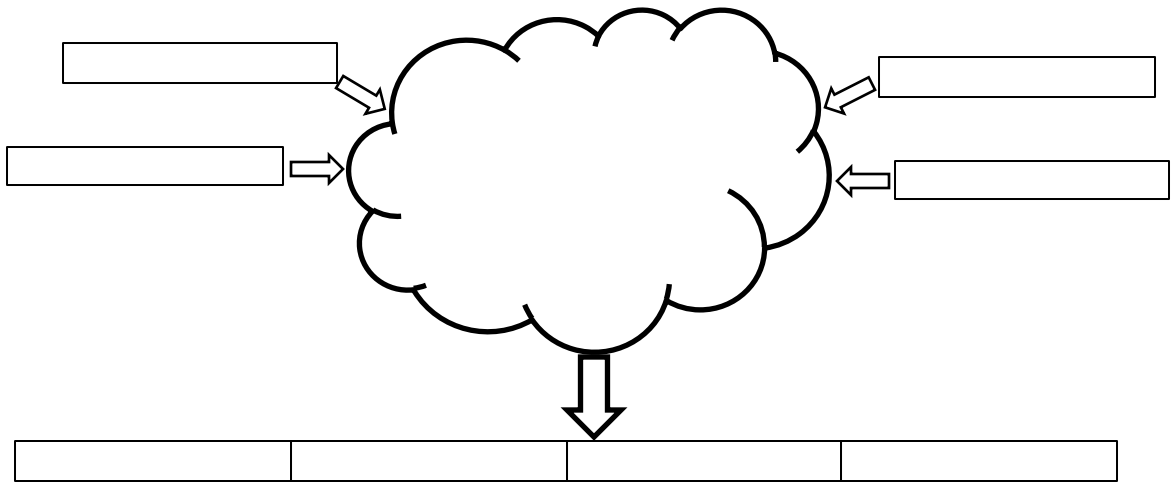
SMR: version of consensus appropriate for a blockchain protocol.

- “state machine” = for us, current state of virtual machine
- “replication” = all validators perform same state transitions

State Machine Replication (SMR)

SMR: version of consensus appropriate for a blockchain protocol.

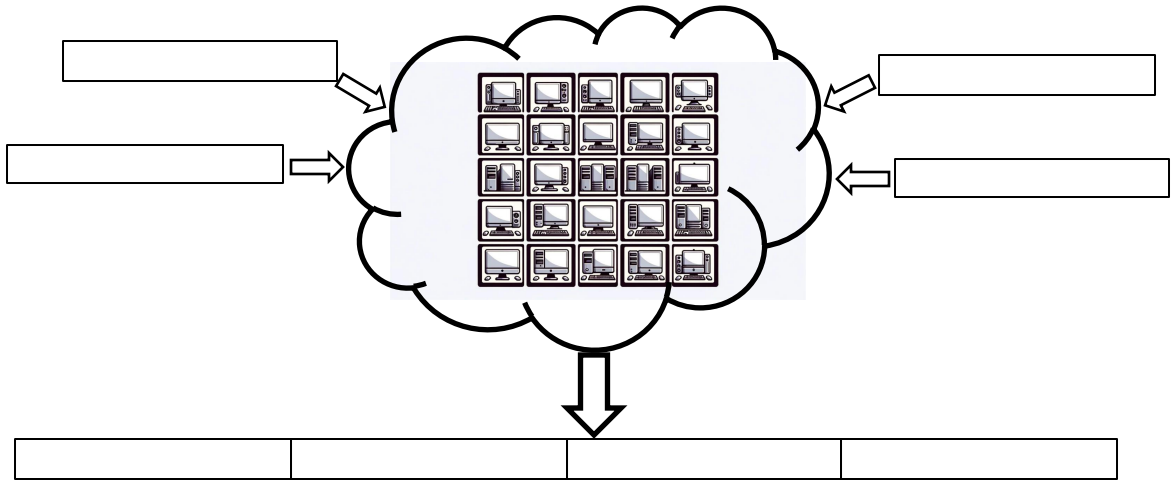
- “state machine” = for us, current state of virtual machine
- “replication” = all validators perform same state transitions
- “clients” submit transactions (“txs”) to validators
- each validator maintains an append-only list of finalized txs (a.k.a. “log” or “history”)



State Machine Replication (SMR)

SMR: version of consensus appropriate for a blockchain protocol.

- “state machine” = for us, current state of virtual machine
- “replication” = all validators perform same state transitions
- “clients” submit transactions (“txs”) to validators
- each validator maintains an append-only list of finalized txs (a.k.a. “log” or “history”)



State Machine Replication (SMR)

SMR: version of consensus appropriate for a blockchain protocol.

- “state machine” = for us, current state of virtual machine
- “replication” = all validators perform same state transitions
- “clients” submit transactions (“txs”) to validators
- each validator maintains an append-only list of finalized txs (a.k.a. “log” or “history”)

Goal: a protocol that satisfies consistency and liveness.

Protocols, Consistency, Liveness

Protocol: code run by each validator. Each time step:

- perform local computations
- receive messages from other validators and txs from clients
- send messages to other validators

Protocols, Consistency, Liveness

Protocol: code run by each validator. Each time step:

- perform local computations
- receive messages from other validators and txs from clients
- send messages to other validators

Consistency: all validators agree on a transaction sequence.

- ok if some lag behind, but no disagreements allowed!

Protocols, Consistency, Liveness

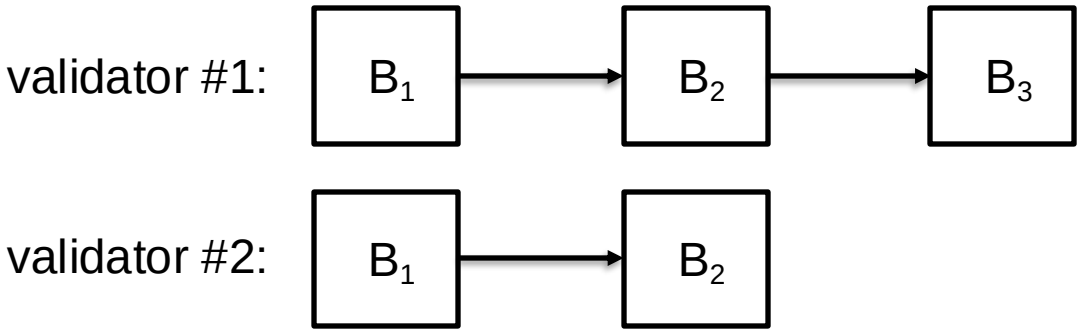
Protocol: code run by each validator. Each time step:

- perform local computations
- receive messages from other validators and txs from clients
- send messages to other validators

Consistency: all validators agree on a transaction sequence.

- ok if some lag behind, but no disagreements allowed!

acceptable:



Protocols, Consistency, Liveness

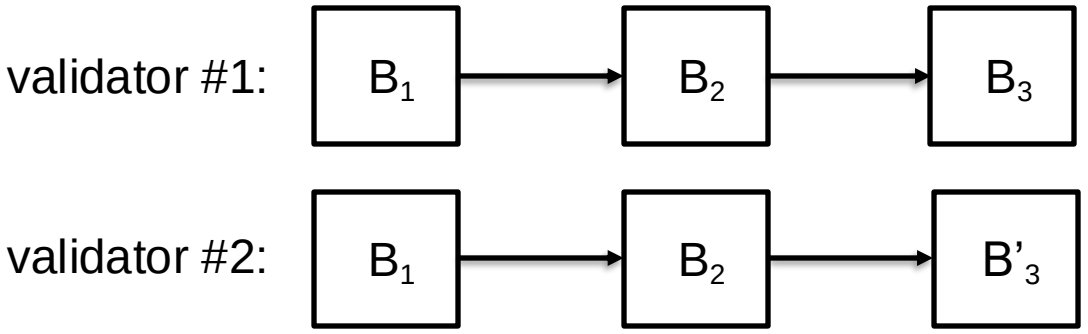
Protocol: code run by each validator. Each time step:

- perform local computations
- receive messages from other validators and txs from clients
- send messages to other validators

Consistency: all validators agree on a transaction sequence.

- ok if some lag behind, but no disagreements allowed!

unacceptable:



Protocols, Consistency, Liveness

Protocol: code run by each validator. Each time step:

- perform local computations
- receive messages from other validators and txs from clients
- send messages to other validators

Consistency: all validators agree on a transaction sequence.

- ok if some lag behind, but no disagreements allowed!
- equivalently: all validators' local chains should be prefixes of a single chain

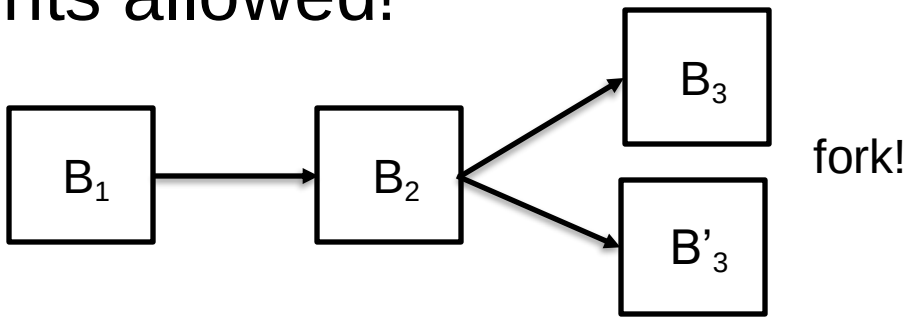
Protocols, Consistency, Liveness

Protocol: code run by each validator. Each time step:

- perform local computations
- receive messages from other validators and txs from clients
- send messages to other validators

Consistency: all validators agree on a transaction sequence.

- ok if some lag behind, but no disagreements allowed!
- equivalently: all validators' local chains should be prefixes of a single chain
  - i.e., no "forks" allowed



Protocols, Consistency, Liveness

Protocol: code run by each validator. Each time step:

- perform local computations
- receive messages from other validators and txs from clients
- send messages to other validators

Consistency: all validators agree on a transaction sequence.

- i.e., all validators' local chains are prefixes of a single chain

Protocols, Consistency, Liveness

Protocol: code run by each validator. Each time step:

- perform local computations
- receive messages from other validators and txs from clients
- send messages to other validators

Consistency: all validators agree on a transaction sequence.

- i.e., all validators' local chains are prefixes of a single chain

Liveness: every valid transaction submitted by a client eventually added to validators' local histories/chains.

Protocols, Consistency, Liveness

Protocol: code run by each validator. Each time step:

- perform local computations
- receive messages from other validators and txs from clients
- send messages to other validators

Consistency: all validators agree on a transaction sequence.

- i.e., all validators' local chains are prefixes of a single chain

Liveness: every valid transaction submitted by a client eventually added to validators' local histories/chains.

- practically relevant strengthening: also want bounded *latency*

Why Is Consensus Hard?

Protocol A:

- target one block per second (say)

Why Is Consensus Hard?

Protocol A:

- target one block per second (say)

1. validators take turns as “leader” (round-robin, one per second)
   - plays the role of a temporary dictator (to coordinate others)
   - recall assumptions of known validator set, shared global clock

Why Is Consensus Hard?

Protocol A:

- target one block per second (say)

1. validators take turns as “leader” (round-robin, one per second)
   - plays the role of a temporary dictator (to coordinate others)
   - recall assumptions of known validator set, shared global clock
2. current leader decides on a block of transactions
   - e.g., all new transactions it’s heard of, ordered by time of arrival

Why Is Consensus Hard?

Protocol A:

- target one block per second (say)

1. validators take turns as “leader” (round-robin, one per second)
   - plays the role of a temporary dictator (to coordinate others)
   - recall assumptions of known validator set, shared global clock
2. current leader decides on a block of transactions
   - e.g., all new transactions it’s heard of, ordered by time of arrival
3. leader sends block to all the other validators

Why Is Consensus Hard?

Protocol A:

- target one block per second (say)

1. validators take turns as “leader” (round-robin, one per second)
   - plays the role of a temporary dictator (to coordinate others)
   - recall assumptions of known validator set, shared global clock
2. current leader decides on a block of transactions
   - e.g., all new transactions it’s heard of, ordered by time of arrival
3. leader sends block to all the other validators

Question: what could go wrong?

Why Is Consensus Hard?

Protocol A: [target e.g. one block per second]

1. validators take turns as “leader” (round-robin, one per second)
2. current leader decides on a block of transactions
3. leader sends block to all the other validators

Question: what could go wrong?

Why Is Consensus Hard?

Protocol A: [target e.g. one block per second]

1. validators take turns as “leader” (round-robin, one per second)
2. current leader decides on a block of transactions
3. leader sends block to all the other validators

Question: what could go wrong?

- what if a validator doesn’t hear from the current leader within one second?

Why Is Consensus Hard?

Protocol A: [target e.g. one block per second]

1. validators take turns as “leader” (round-robin, one per second)
2. current leader decides on a block of transactions
3. leader sends block to all the other validators

Question: what could go wrong?

- what if a validator doesn’t hear from the current leader within one second?
  - perhaps due to problem with leader (e.g., might have crashed)

Why Is Consensus Hard?

Protocol A: [target e.g. one block per second]

1. validators take turns as “leader” (round-robin, one per second)
2. current leader decides on a block of transactions
3. leader sends block to all the other validators

Question: what could go wrong?

- what if a validator doesn’t hear from the current leader within one second?
  - perhaps due to problem with leader (e.g., might have crashed)
  - perhaps due to problem with network (e.g., congestion, or DoS attack)

Why Is Consensus Hard?

Key challenges: [unavoidable in practice]

Why Is Consensus Hard?

Key challenges: [unavoidable in practice]

1. Faulty validators (i.e., don't behave as expected).
   - and a protocol doesn't automatically know which ones are faulty

Why Is Consensus Hard?

Key challenges: [unavoidable in practice]

1. Faulty validators (i.e., don't behave as expected).
   - and a protocol doesn't automatically know which ones are faulty
2. Unreliable communication network (i.e., doesn't behave as expected).

Why Is Consensus Hard?

Key challenges: [unavoidable in practice]

1. Faulty validators (i.e., don't behave as expected).
   - and a protocol doesn't automatically know which ones are faulty
2. Unreliable communication network (i.e., doesn't behave as expected).

Revised goal: an SMR protocol that satisfies consistency and liveness, despite the presence of faulty validators and an unreliable communication network.

Why Is Consensus Hard?

Key challenges: [unavoidable in practice]

1. Faulty validators (i.e., don't behave as expected).
   - and a protocol doesn't automatically know which ones are faulty
2. Unreliable communication network (i.e., doesn't behave as expected).

Revised goal: an SMR protocol that satisfies consistency and liveness, despite the presence of faulty validators and an unreliable communication network.

- next: can make SMR easier/harder by allowing less/more severe validator faults and network unreliability

Faulty Validators, Unreliable Network

Faulty validators (easy mode): *crash faults.*

- every validator dutifully follows the protocol, but may crash (forever) at some point

Faulty Validators, Unreliable Network

Faulty validators (easy mode): *crash faults.*

- every validator dutifully follows the protocol, but may crash (forever) at some point

Faulty validators (hard mode): *“Byzantine” faults.*

- a faulty validator can behave arbitrarily
  - e.g., hard-to-model software bug, or maybe just a bad actor

Faulty Validators, Unreliable Network

Faulty validators (easy mode): *crash faults.*

- a faulty validator may crash (forever) at some point

Faulty validators (hard mode): *Byzantine faults.*

- a faulty validator can behave arbitrarily

Unreliable network (easy mode): *synchronous network.*

- for known parameter Δ, every msg delivered in $\leq \Delta$ time steps
  - e.g., 1 time step = 1 millisecond, Δ=2000 (i.e., $\leq$ 2 seconds)

Faulty Validators, Unreliable Network

Faulty validators (easy mode): *crash faults.*

- a faulty validator may crash (forever) at some point

Faulty validators (hard mode): *Byzantine faults.*

- a faulty validator can behave arbitrarily

Unreliable network (easy mode): *synchronous network.*

- for known parameter Δ, every msg delivered in $\leq \Delta$ time steps

Unreliable network (hard mode): *asynchronous network.*

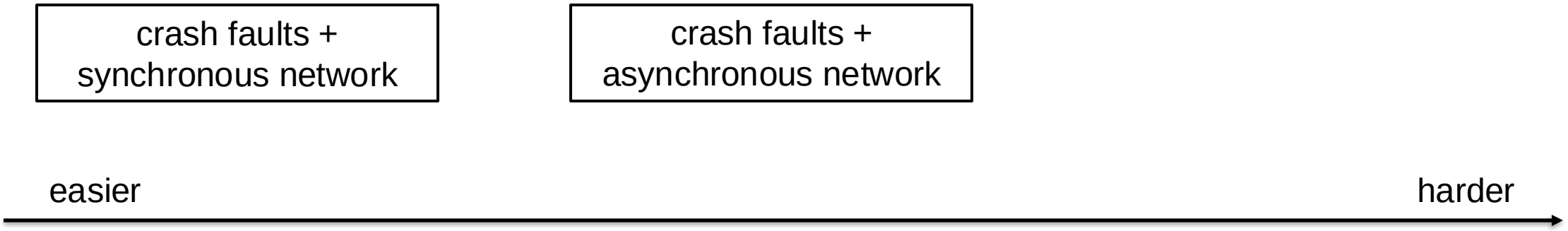
- all messages eventually delivered, but no bound on delay

A Road Map to Practical SMR Protocols

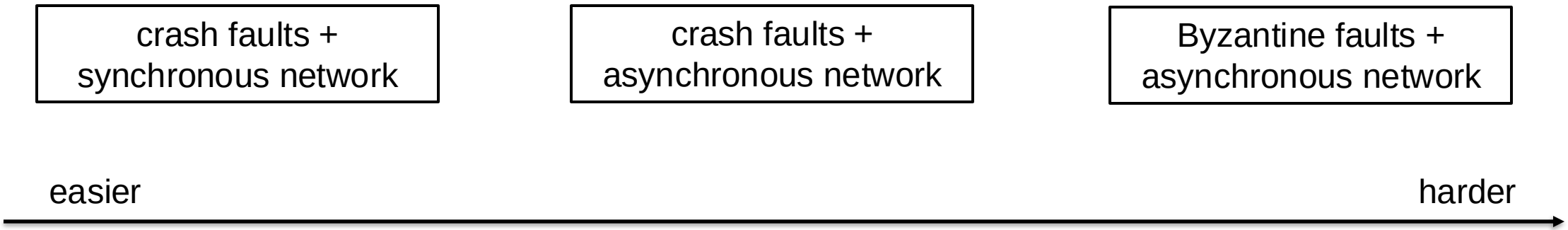
A Road Map to Practical SMR Protocols

crash faults +
synchronous network

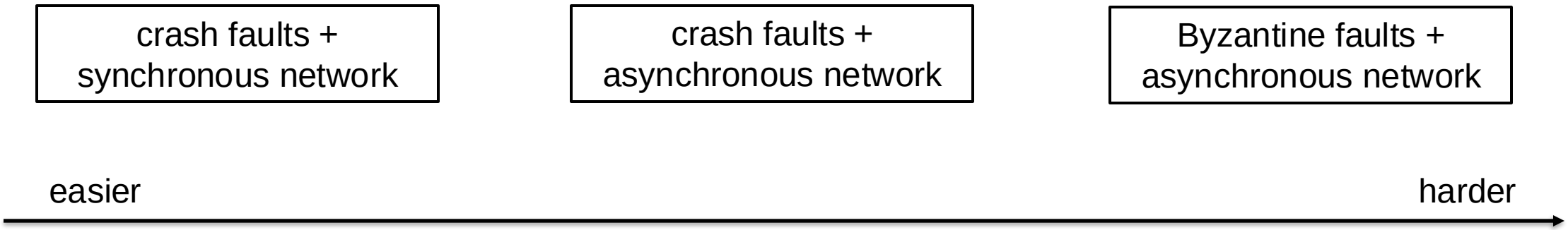
A Road Map to Practical SMR Protocols



A Road Map to Practical SMR Protocols



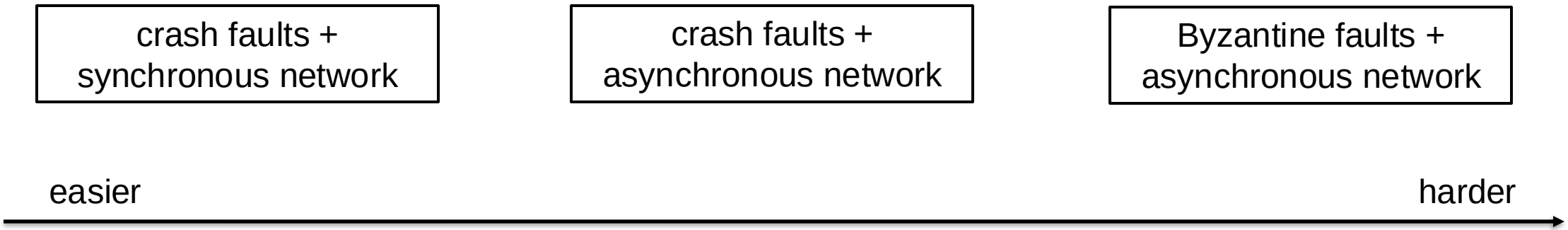
A Road Map to Practical SMR Protocols



Expectations:

1. More positive results (i.e., good SMR protocols) toward the left.

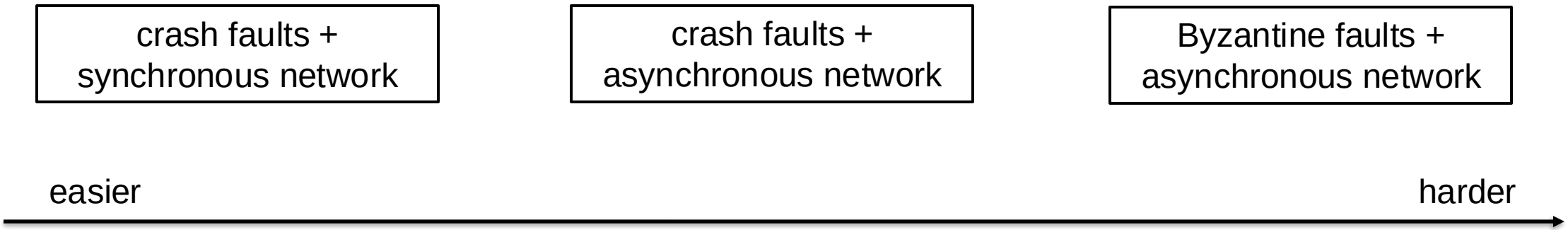
A Road Map to Practical SMR Protocols



Expectations:

1. More positive results (i.e., good SMR protocols) toward the left.
2. More impossibility results (i.e., SMR unsolvable) toward the right.

A Road Map to Practical SMR Protocols



Expectations:

1. More positive results (i.e., good SMR protocols) toward the left.
2. More impossibility results (i.e., SMR unsolvable) toward the right.
3. Simpler protocols toward the left, more complex toward the right.

Security Thresholds

Intuition: [mostly correct]

Security Thresholds

Intuition: [mostly correct]

1. Achieving consensus should be easy if 0% of the validators are faulty.
   - just wait for leaders' block proposals to (eventually) arrive

Security Thresholds

Intuition: [mostly correct]

1. Achieving consensus should be easy if 0% of the validators are faulty.
   - just wait for leaders' block proposals to (eventually) arrive
2. Achieving consensus should be impossible if ≈100% of the validators are faulty.
   - 98 colluding validators can convince the other 2 of different blocks

Security Thresholds

Intuition: [mostly correct]

1. Achieving consensus should be easy if 0% of the validators are faulty.
   - just wait for leaders' block proposals to (eventually) arrive
2. Achieving consensus should be impossible if ≈100% of the validators are faulty.
   - 98 colluding validators can convince the other 2 of different blocks

Security threshold: fraction of faulty validators at which achieving consensus flips from possible to impossible.

Security Thresholds

1. Achieving consensus should be easy if 0% of the validators are faulty.
2. Achieving consensus should be impossible if ≈100% of the validators are faulty.

Security threshold: fraction of faulty validators at which achieving consensus flips from possible to impossible.

Security Thresholds

1. Achieving consensus should be easy if 0% of the validators are faulty.
2. Achieving consensus should be impossible if ≈100% of the validators are faulty.

Security threshold: fraction of faulty validators at which achieving consensus flips from possible to impossible.

- threshold will depend on assumptions (on faults, network, etc.)
- typical values = 50% or 33%

Security Thresholds

1. Achieving consensus should be easy if 0% of the validators are faulty.
2. Achieving consensus should be impossible if ≈100% of the validators are faulty.

Security threshold: fraction of faulty validators at which achieving consensus flips from possible to impossible.

- threshold will depend on assumptions (on faults, network, etc.)
- typical values = 50% or 33%

Moral: crucial to have validator set of mostly reliable operators!