Bitcoin & RAFT

Distributed Systems

Nikita Borisov

Topics for Today

- Finish Bitcoin
  - Broadcast mechanism
- Overview of MP2
- Raft consensus

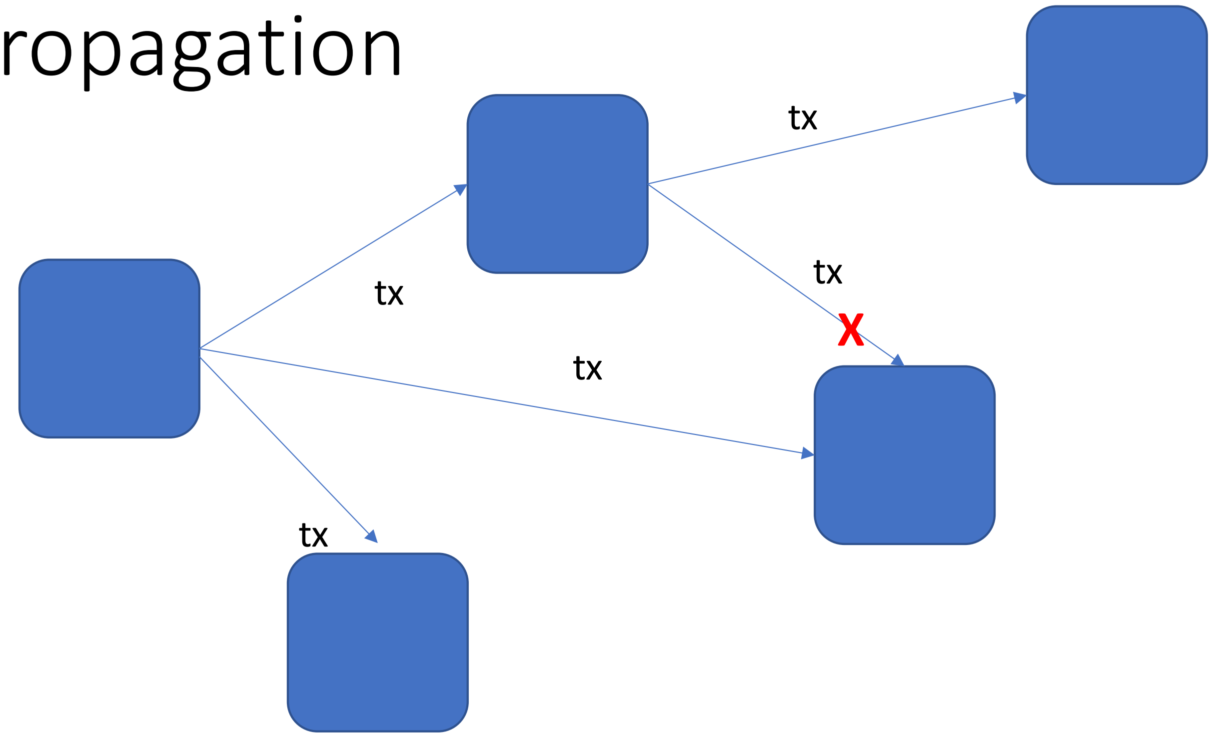
Bitcoin broadcast

- Need to broadcast:
  - Transactions to all nodes, so they can be included in a block
  - New blocks to all nodes, so that they can switch to longest chain
- Why not R-multicast?
  - Have to send O(N) messages
  - Have to *know* which nodes to send to

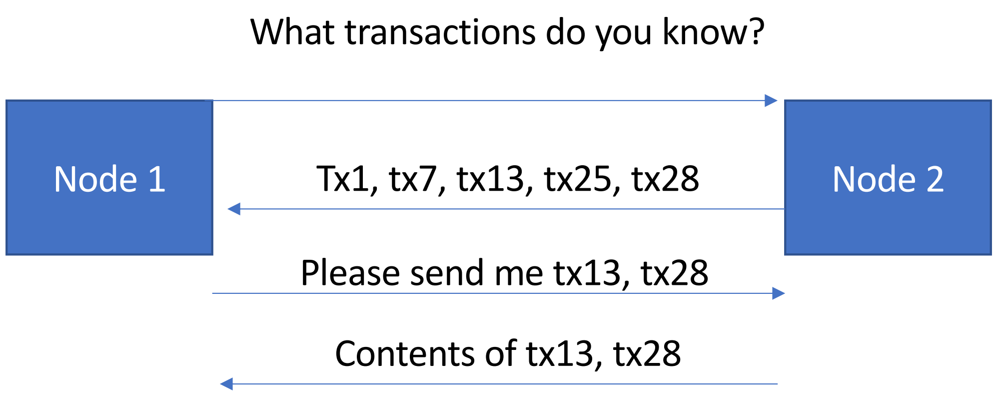
Gossip / Viral propagation

- Each node connects to a small set of *neighbors*
  - 10–100
- Nodes propagate transactions and blocks to neighbors
- Push method: when you hear a new tx/block, resend them to all (some) of your neighbors (flooding)
- Pull method: periodically poll neighbors for list of blocks/tx’s, then request any you are missing

Push propagation



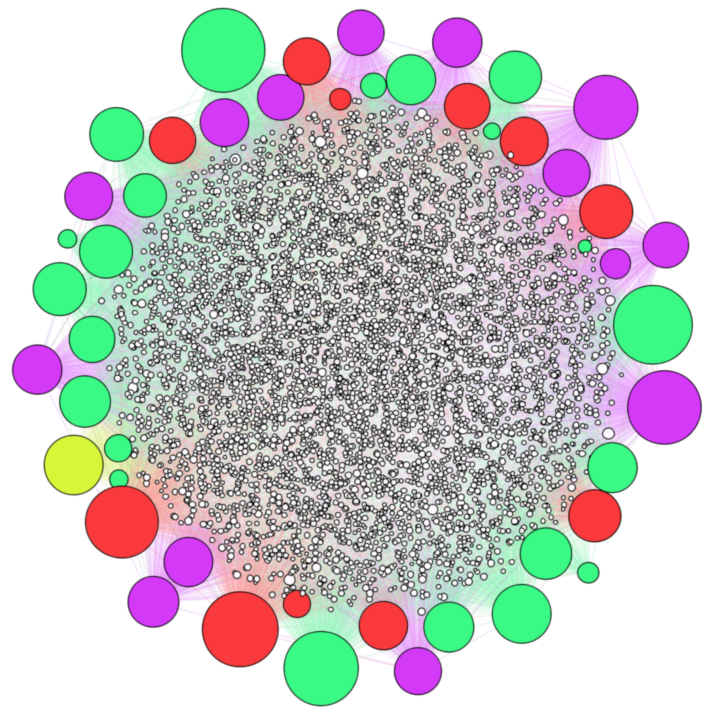
Pull propagation



Maintaining Neighbors

- A *seed* service
  - Gives out a list of random or well-connected nodes
  - E.g., seed.bitnodes.io

- Neighbor discovery
  - Ask neighbors about *their* neighbors
  - Randomly connect to some of them



MP2: Cryptocurrency

Implement a simple blockchain/cryptocurrency application

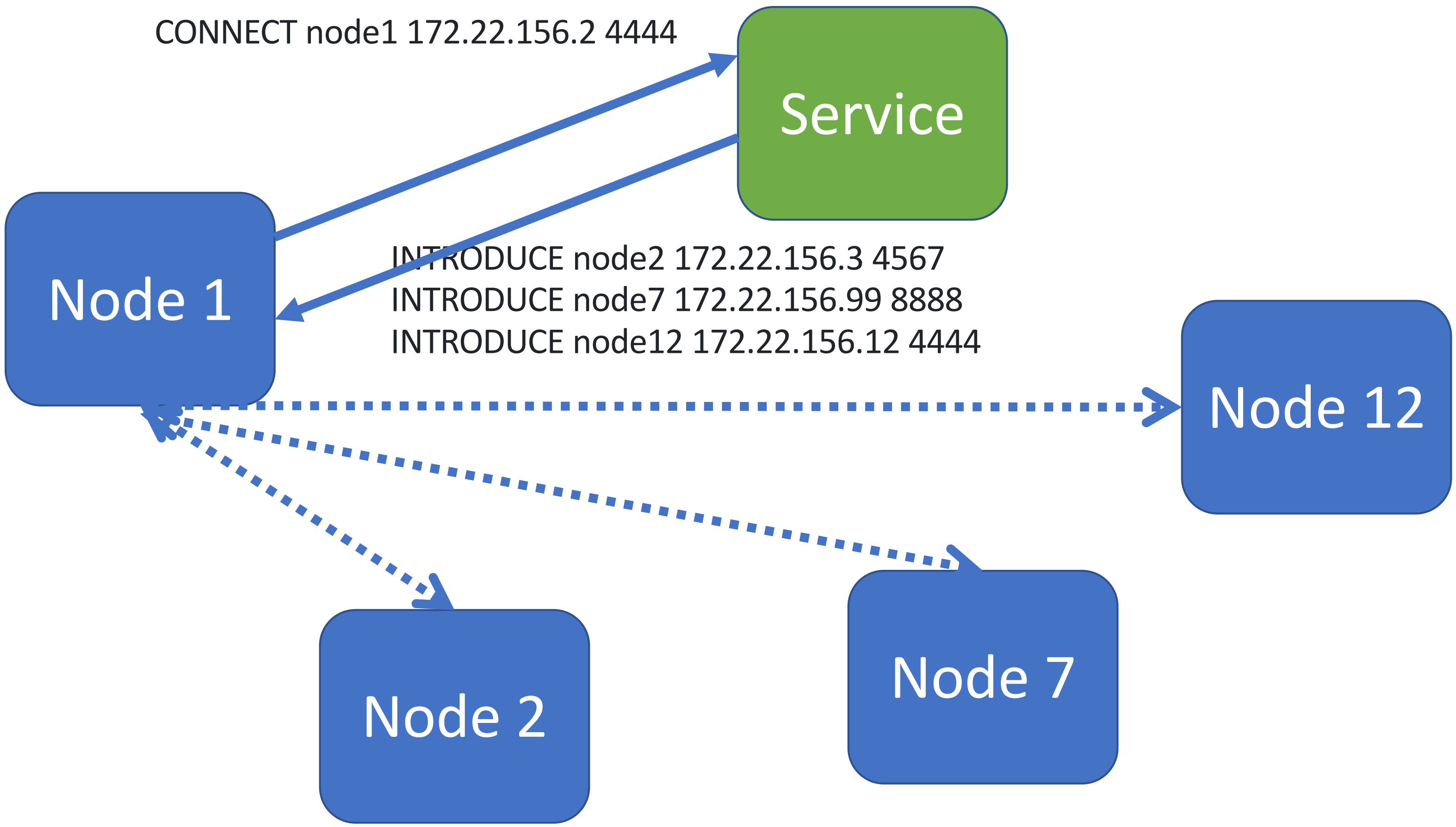
- Build a network of nodes
- Broadcast transactions
- “Mine” and broadcast blocks
- Validate blocks and enforce longest-chain rule

MP2 service

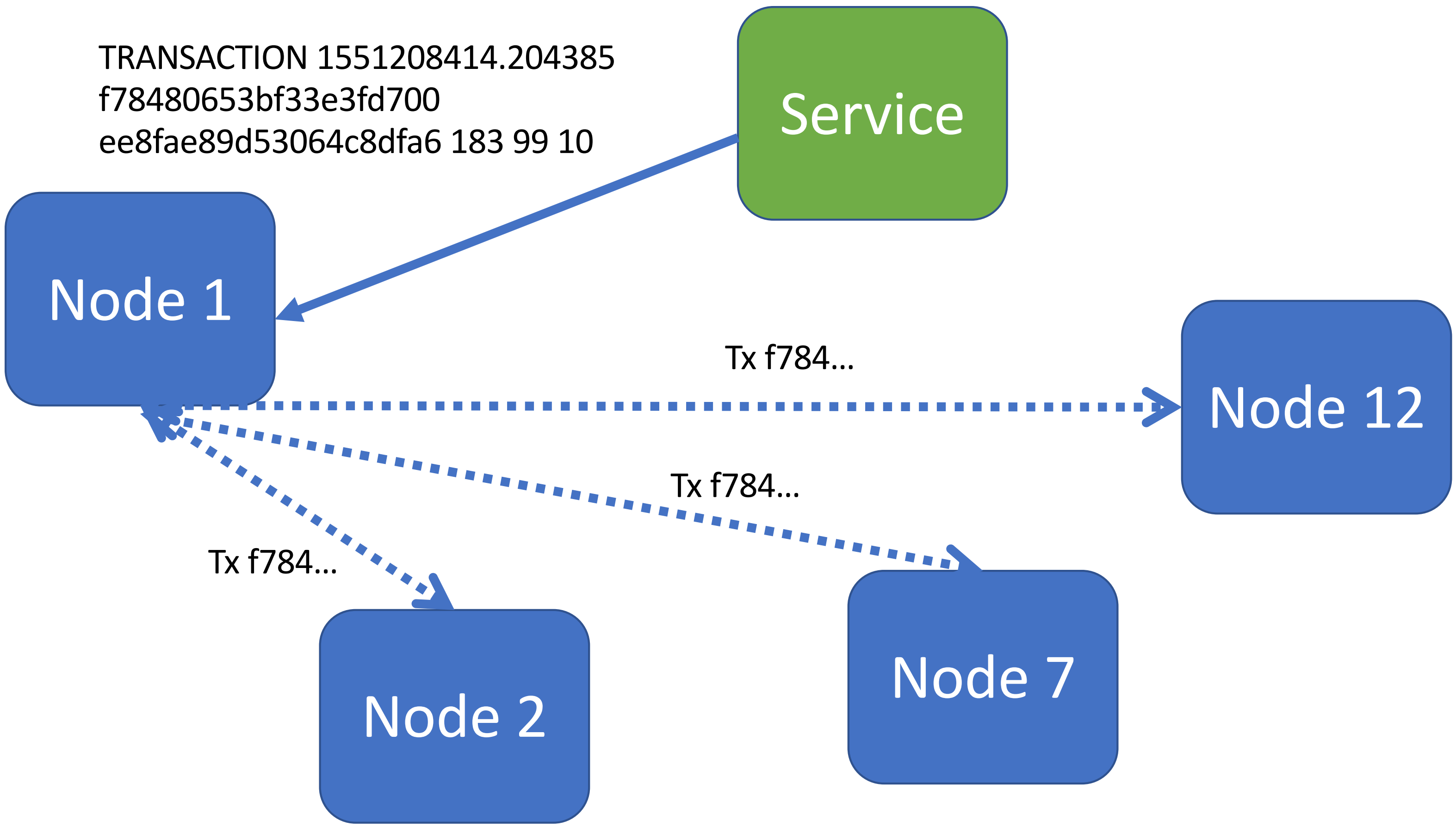
A network service to help with various aspects of the MP

- Introduce nodes to each other
- Generates transactions
- Simulates proof of work
- Tells nodes when to die

Part 1: Transaction broadcast



Part 1: Transaction broadcast



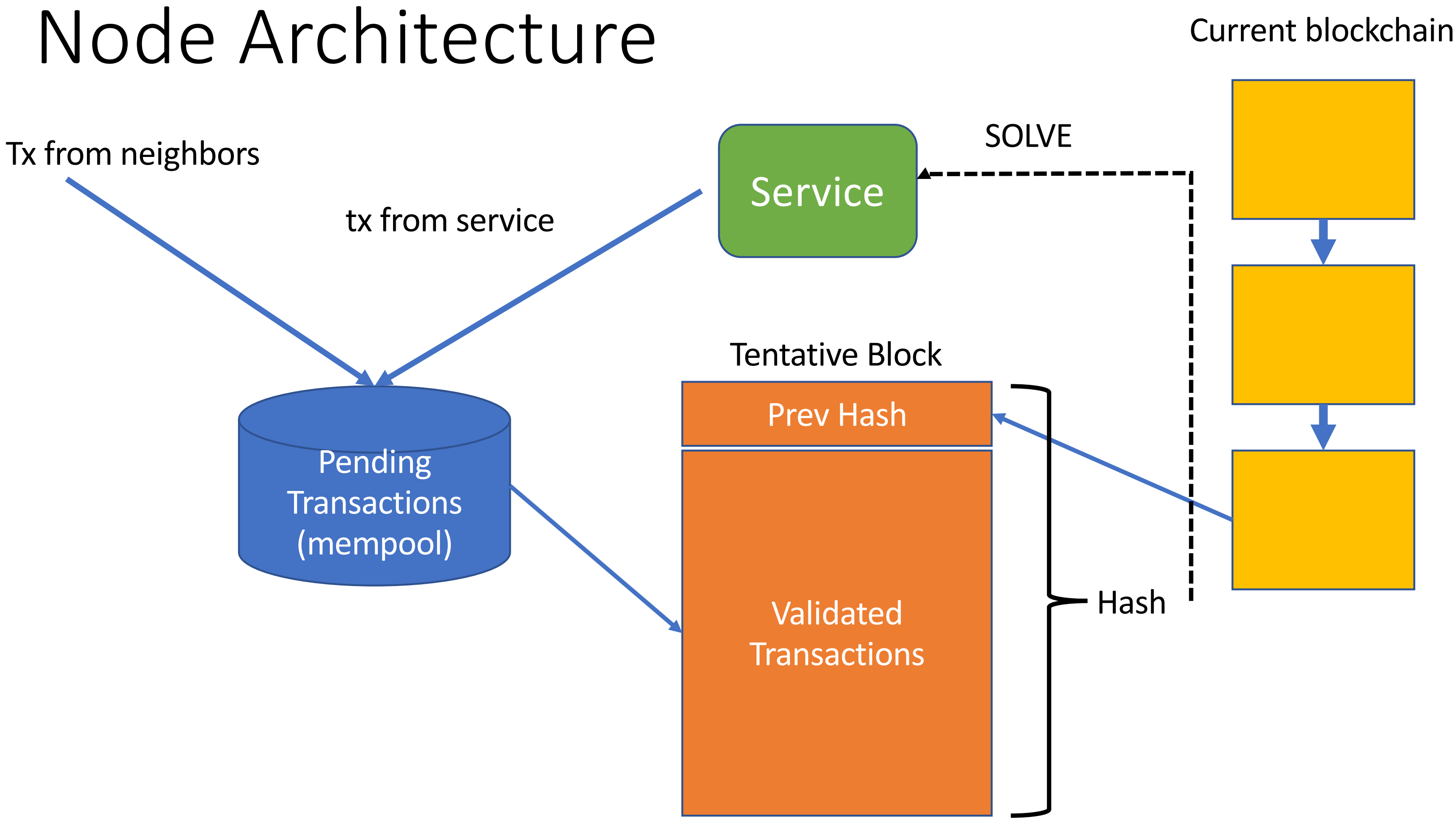
Tasks

- Maintain connectivity
  - As new nodes arrive
  - As existing nodes die
- Propagate transactions to **all** nodes
- Collect metrics
  - Transaction propagation delay
  - Aggregate bandwidth
- No efficiency target, but bonus marks for high performance!

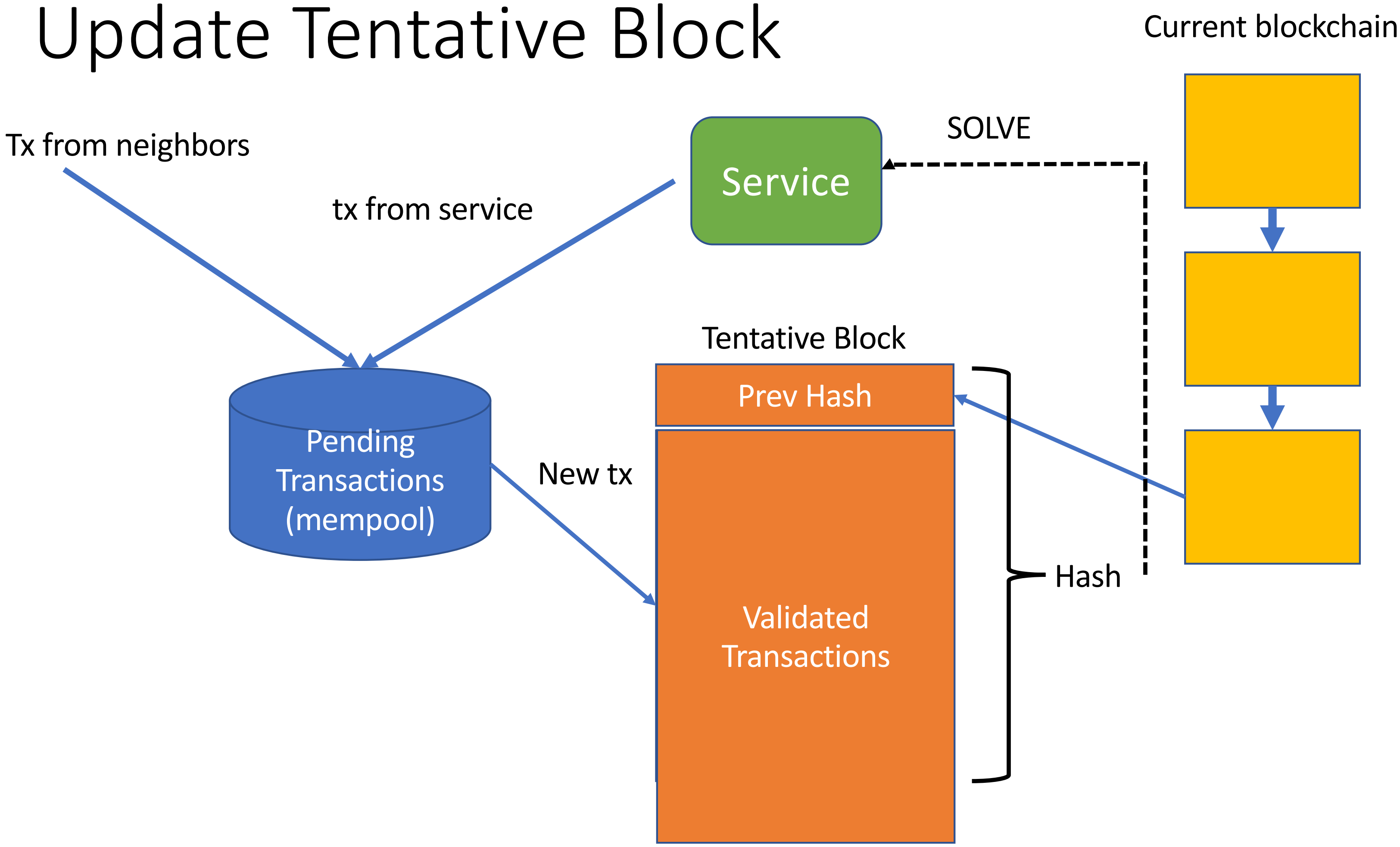
Part 2: Block creation and propagation

- Accumulate transactions into blocks
  - Enforce ordering
  - Prevent double-spending
- Use service to “solve” puzzles
- Propagate blocks to other nodes
- Import and verify blocks
- Resolve chain forks

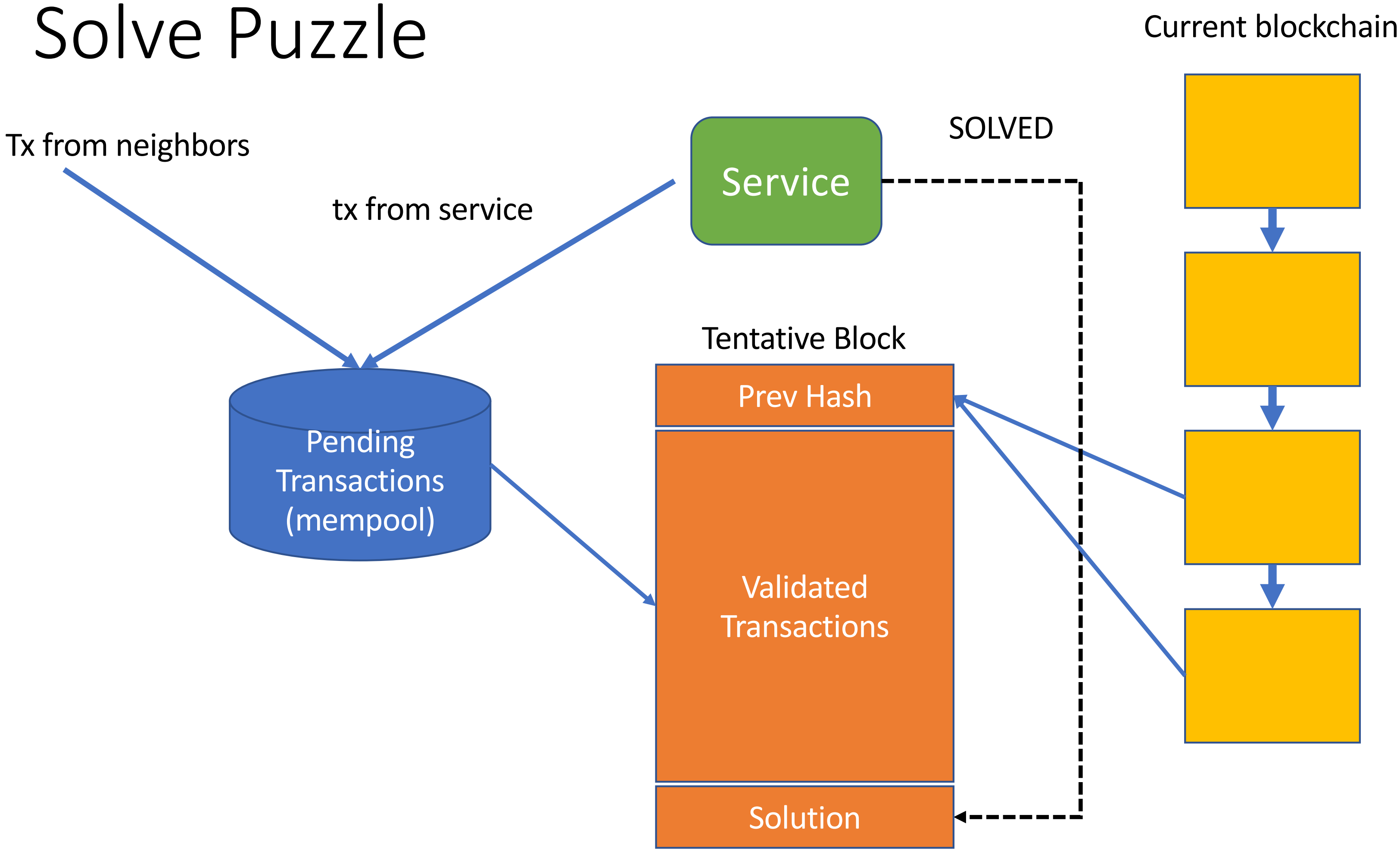
Node Architecture



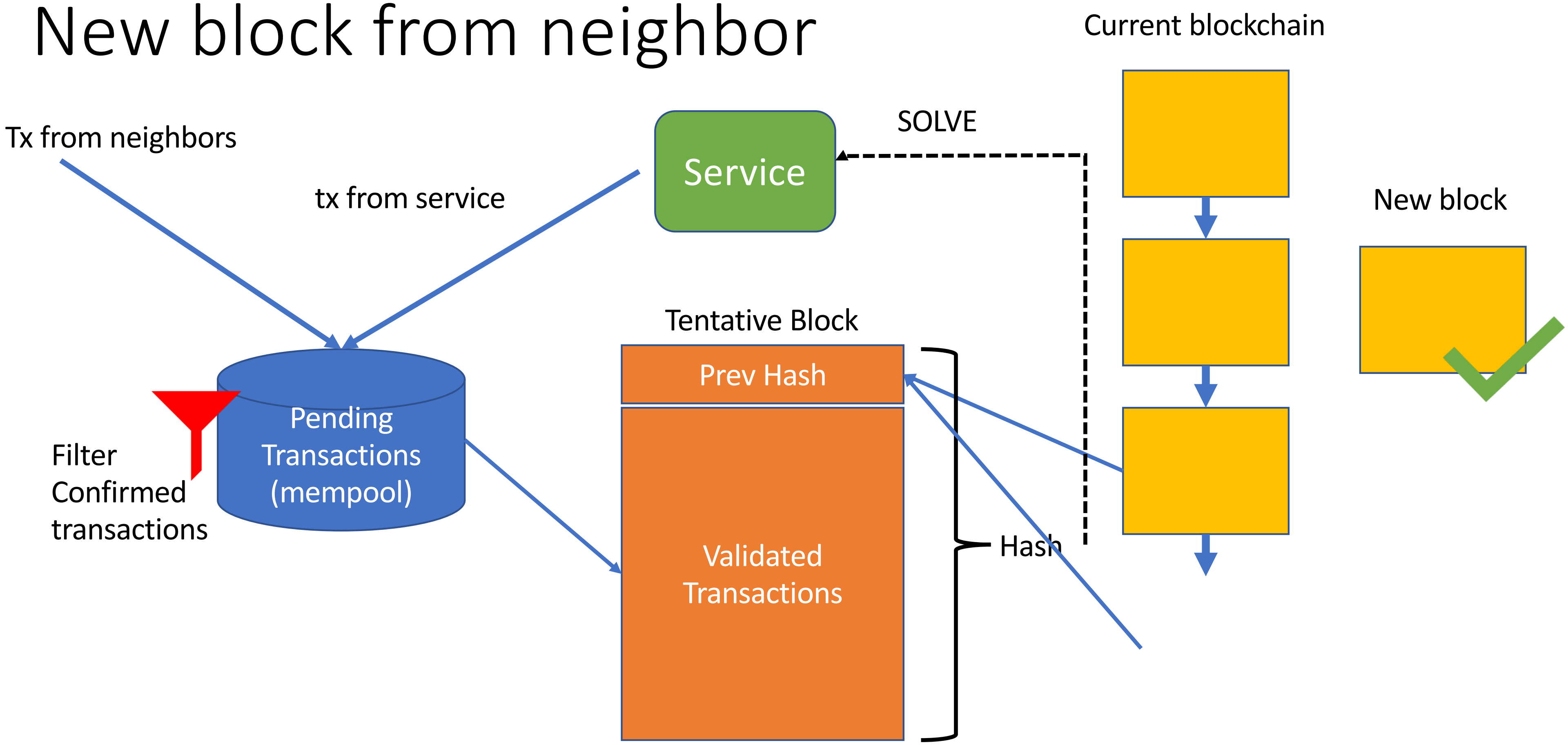
Update Tentative Block



Solve Puzzle



New block from neighbor



RAFT Consensus

Slide content borrowed from Diego Ongaro, John Ousterhout, and Alberto Montresor

Log Consensus

- Bit consensus: agree on a single bit, based on inputs
  - (0,1,0,0,1,0,0) -> 1

- Log consensus: agree on contents and *order* of events in a log
  - {A, B, Q, R, W, Z} -> [A, Q, R, B, Z]

Log-based

- Each *replica* maintains a *log* of events (from client(s))
- Replicas *apply* events in the log to update their *state*
- Same initial state + same order of events in the log => consistent final state

Log Consensus

- All replicas must agree on the order of events in the log
- Is this possible in asynchronous systems?

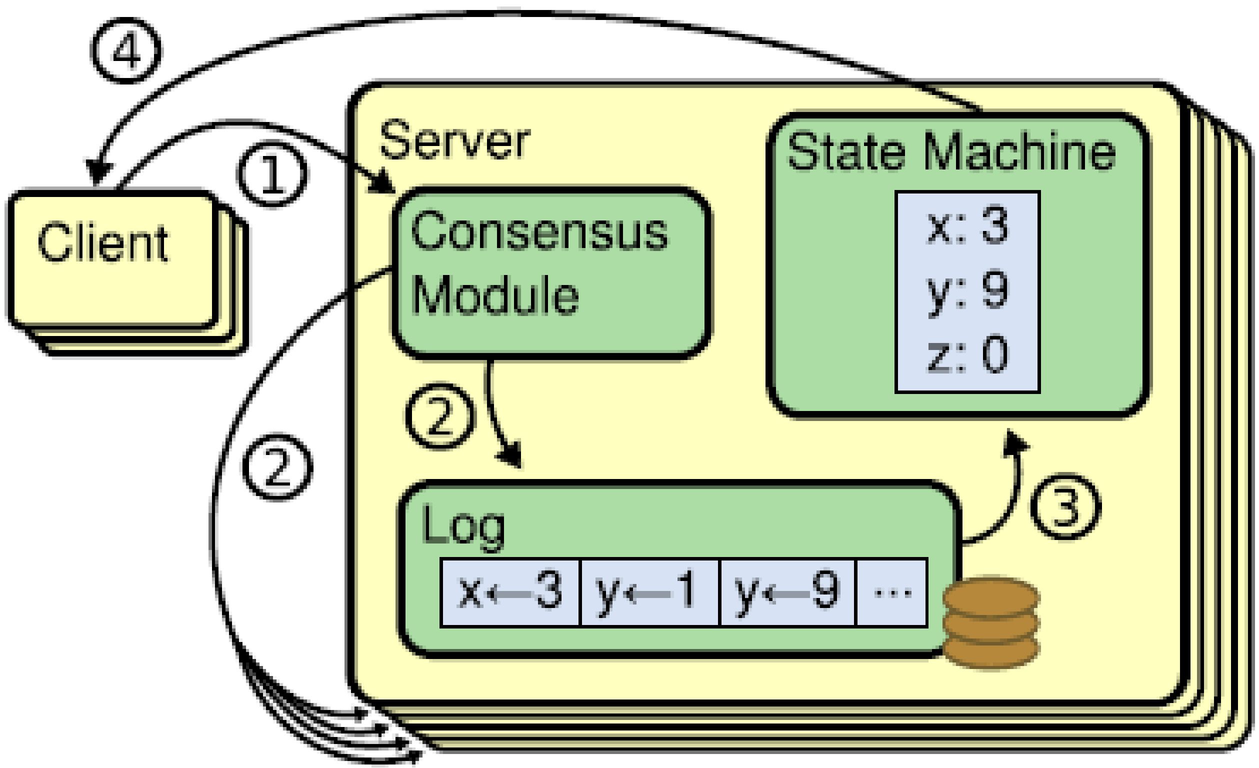
Log Consensus

- All replicas must agree on the order of events in the log
- Is this possible in asynchronous systems?
  - Totally correct implementation impossible (FLP)!
- Safety
  - Replicas always add events in consistent order
- Liveness
  - If a **majority** of nodes is **available**, they will eventually establish consistent log order
  - **Available** = not failed, and not delayed beyond a bound

The distributed log (I)

- Each server stores a log containing commands
- Consensus algorithm ensures that all logs contain the ***same commands*** in the same order
- State machines always execute commands ***in the log order***
  - They will remain consistent as long as command executions have ***deterministic results***

The distributed log (II)



The distributed log (III)

- Client sends a command to one of the servers
- Server adds the command to its log
- Server forwards the new log entry to the other servers
- Once a consensus has been reached, each server state machine process the command and sends it reply to the client

Paxos

Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state-machine approach to the design of distributed systems — an approach that has received limited attention because it leads to designs of insufficient complexity.

Paxos Timeline

- 1989: Lamport wrote 42 page (!) DEC technical report
- 1990: Submitted to and **rejected from** ACM Transactions on Computer Systems
- 1998: The original paper is resubmitted and accepted by TOCS.
- 2001 Lamport publishes “Paxos made simple” in ACM SIGACT News
- 2007 T. D. Chandra, R. Griesemer, J. Redstone. Paxos made live: an engineering perspective. PODC 2007, Portland, Oregon.

Paxos

- Google uses the Paxos algorithm in their Chubby distributed lock service. Chubby is used by BigTable, which is now in production in Google Analytics and other products
- Amazon Web Services uses the Paxos algorithm extensively to power its platform
- Windows Fabric, used by many of the Azure services, make use of the Paxos algorithm for replication between nodes in a cluster
- Neo4j HA graph database implements Paxos, replacing Apache ZooKeeper used in previous versions.
- Apache Mesos uses Paxos algorithm for its replicated log coordination

Paxos limitations (I)

- Exceptionally difficult to understand

"The dirty little secret of the NSDI community is that at most five people really, truly understand every part of Paxos ;-)."*
– Anonymous NSDI reviewer

*The USENIX Symposium on Networked Systems Design and Implementation

Paxos limitations (II)

- Very difficult to implement

"There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system…the final system will be based on an unproven protocol." – Chubby authors

Designing for understandability

- Main objective of RAFT
  - Whenever possible, select the alternative that is the easiest to understand
- Techniques that were used include
  - Dividing problems into smaller problems
  - Reducing the number of system states to consider
    - Could logs have holes in them? No

Raft consensus algorithm (I)

- Servers start by electing a ***leader***
  - Sole server habilitated to accept commands from clients
  - Will enter them in its log and forward them to other servers
  - Will tell them when it is safe to apply these log entries to their state machines

Raft consensus algorithm (II)

- Decomposes the problem into three fairly independent subproblems
  - ***Leader election:***
    How servers will pick a—***single***—leader
  - ***Log replication:***
    How the leader will accept log entries from clients, propagate them to the other servers and ensure their logs remain in a consistent state
  - ***Safety***