Distributed Hash Tables, Video, and Fun!

Thomas Gebert and Nicholas Misturak

February 14, 2020

Section 1

About Us

Who are we?

- Thomas Gebert
  - Software Engineer in New York, NY.
  - Certified Eccentric.
  - Mostly focuses on backend services.
- Nicholas Misturak
  - Front-End Web Developer in Orlando, FL.
  - Work in JavaScript and React.
  - Know some functional programming.

Not Experts

- We are hobbyists who want to build cool things.
- Possible we are doing things sub-optimally
  - If you see something we're doing wrong, let us know!

Section 2

Background

We want to make a video sharing platform

- YouTube has a near-monopoly on video-sharing on the web
- As a central entity, they have the benefit of complete control of their platform
- However, they also assume all of the cost of maintaining this platform
  - Video transcoding and storage are incredibly expensive
  - Completely infeasable to scale with the resources of most startups

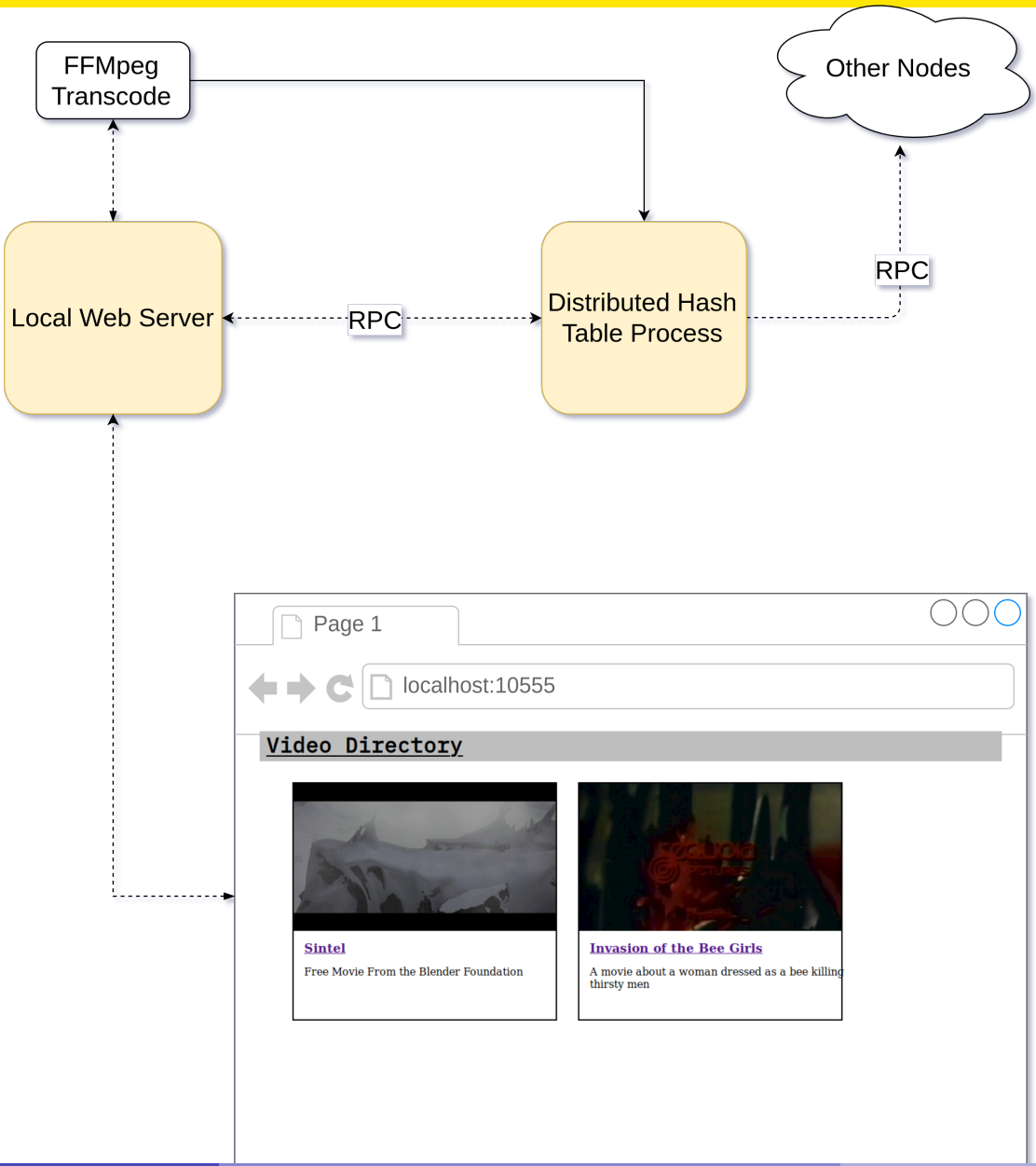
The Solution

- Don't write a centralized system, write a distributed one
- By distributing, we can ask participants in the platform to do the work of transcoding and storage
- In this way, we are sharing the cost of the platform

What we've built

- An efficient, peer-to-peer video sharing platform.
- Every node on the system is a client *and* a server.
- Communication happens over TCP. . . even for the local client.

What we've built



Prototype

- Original version built in early 2018 in Haskell.
- Code was a mess
  - Haskell's networking libraries can be very primitive.
  - There was difficulty in making Haskell's shared-memory-style system work with the model that I wanted for the project.
- Needed a rewrite in a language with better library support.

Rewrite

- Decided to rewrite prototype in Clojure.
- Wanted Go's CSP in a functional language.
- Needed mature and well-supported network libraries.
  - Could not be vanilla Java because life is too short as it is.
- Clojure also has Haskell-style transactional memory for non-reactive concurrency.
- Dynamic languages can be more-pleasant than typed languages for handling low-level networking stuff.

Software Used

Backend

- JeroMQ
  - Low-level socket library to handle TCP and UDP logic.
- MessagePack
- Incredibly fast binary-serialization library with bindings in virtually every language.
- core.async
  - Implementation of Go's CSP for Clojure.

Frontend

- Re-frame
  - React port to ClojureScript

HTTP Live Streaming

- The ability to partition videos and glue them together on the fly is an interesting Computer Science topic in itself.
- HTTP Live Streaming (HLS)
  - Part of the web standard.
  - Works by creating a Winamp Playlist file (.m3u8), and requesting chunks of the video on the fly.
  - This allows us to store any video across the whole network, increasing redundancy and parallelism.

Section 3

Distributed Hash Tables

What is a DHT?

- A data structure to store data and routing information across a cluster of nodes without a centralized server.
- Data is shared and replicated across nodes.
  - There are no central servers.
- Popular algorithms include Chord, Pastry, Tapestry, and Kademlia

Kademlia

- Most popular algorithm for distributed hash table programs
- Used by Facebook and other companies to distribute updates.
- Provides worst-case $\log_2(n)$ lookups

Section 4

Routing

What is a routing table?

A routing table provides a way for us to look up and store nodes that are of interest to us.

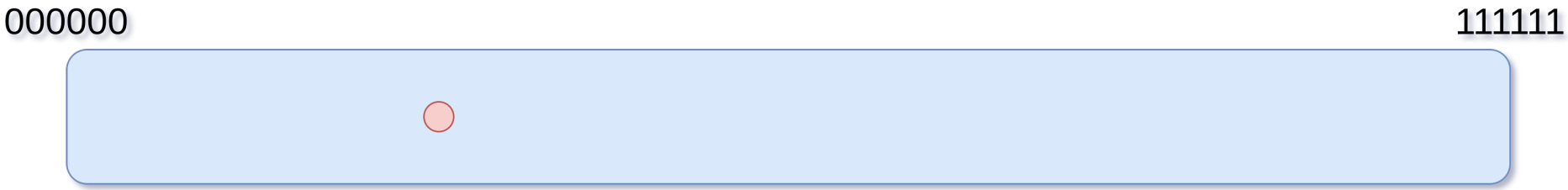
What's the point of a routing table?

- It would be infeasible to store reference to all the nodes in the cluster.
- Ideally, we want to store a small subset of all the nodes that will allow us to find any given value or node in the table eventually.
- It would be best if these lookups were faster than linear time and were likely to resolve.

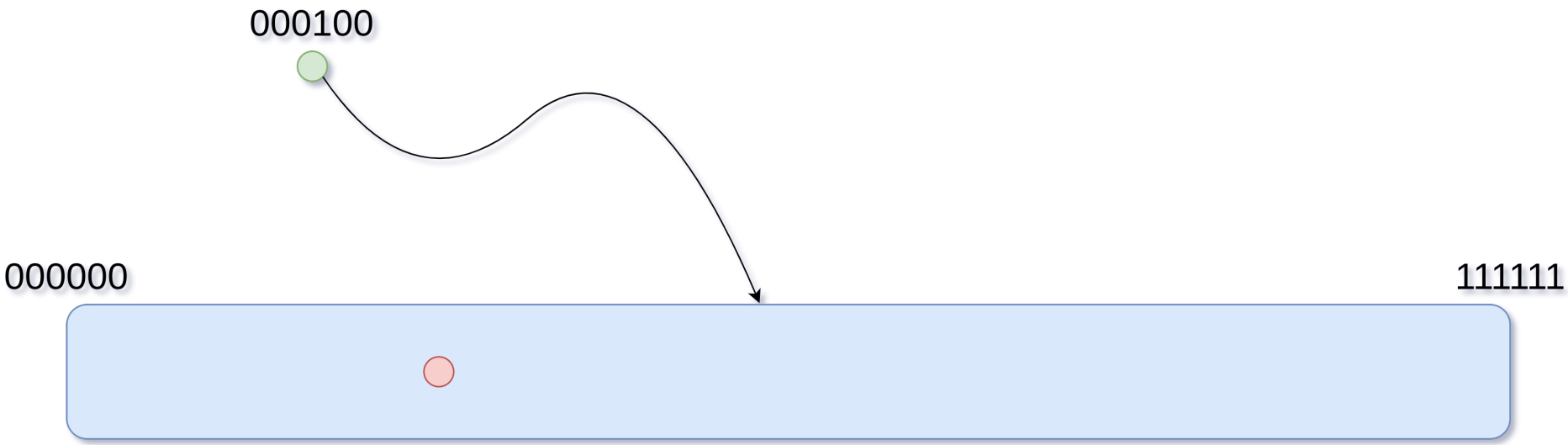
Routing Table

- Each machine participating in the DHT is a Node.
- Nodes are given random 160 bit IDs.
- "Distance" is defined by XORing two IDs.
  - Might seem weird to use as a distance metric but consider this.
    - $a \oplus b = 0 \iff a = b$
    - $a \oplus b = b \oplus a$
  - Gives us a deterministic way to figure out path to where data is stored without central synchronization.
- Each node has its own routing table.
- The table is divided into buckets of a set size.
  - For the sake of this demo, let's assume that the size is 5.

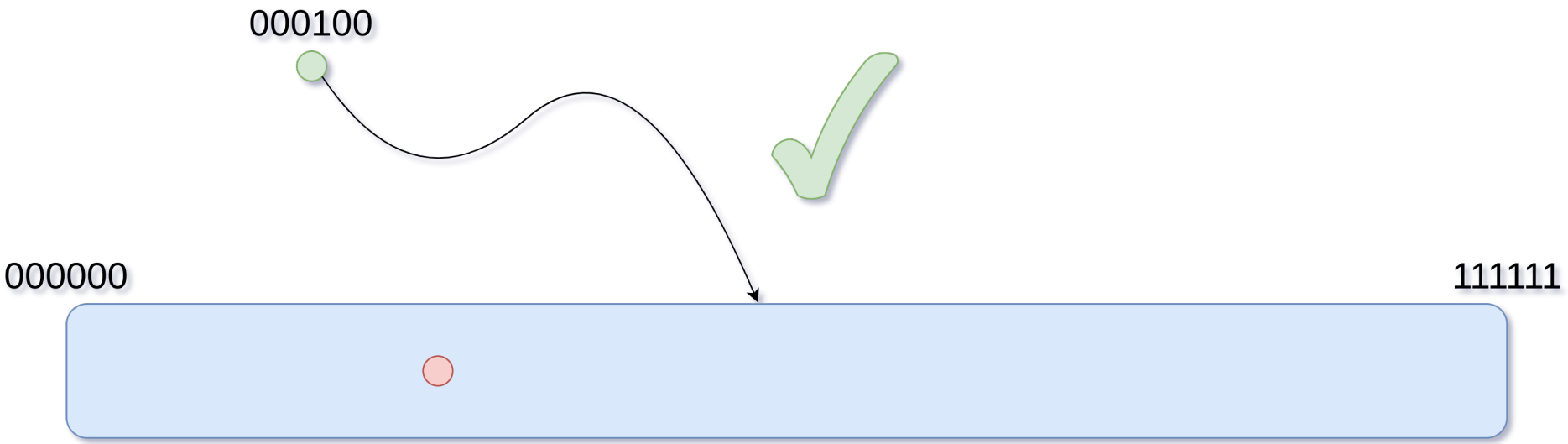
Routing Table



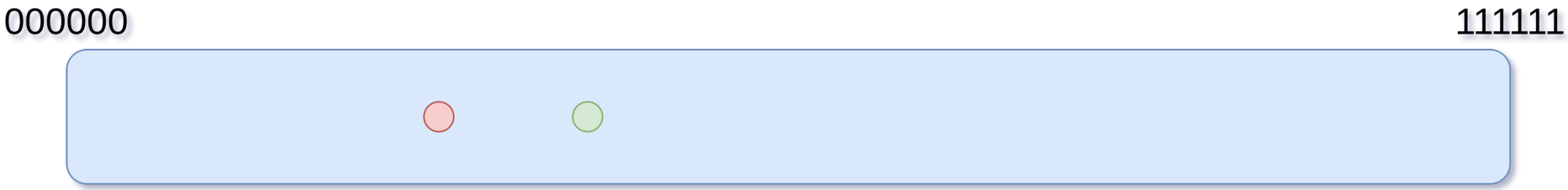
Routing Table



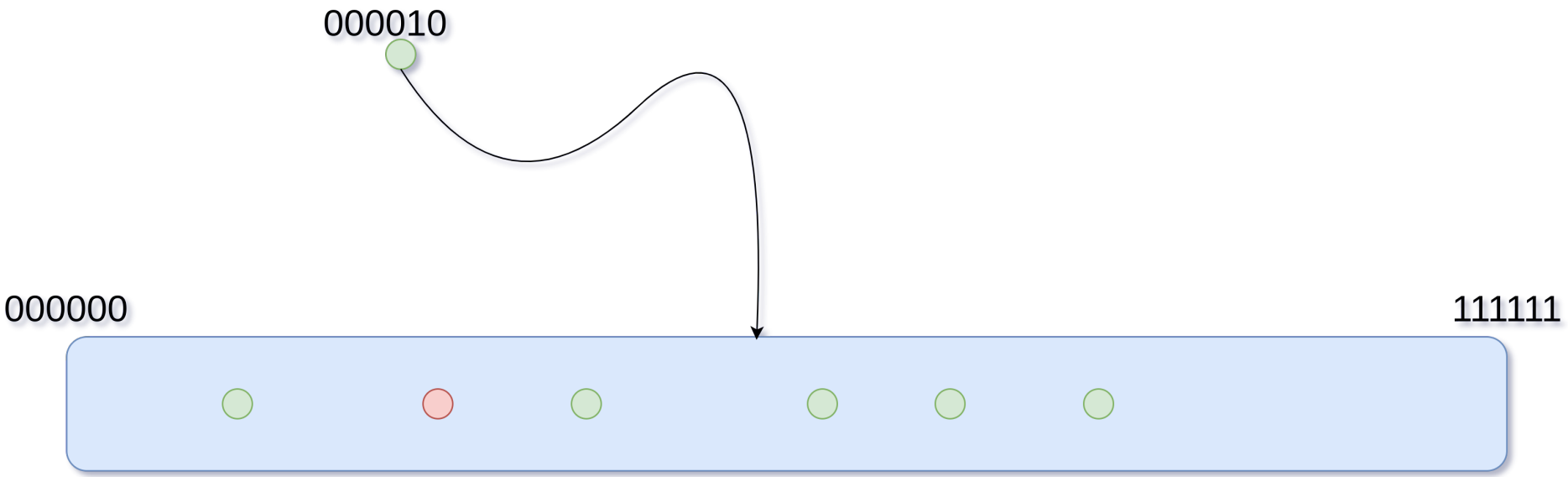
Routing Table



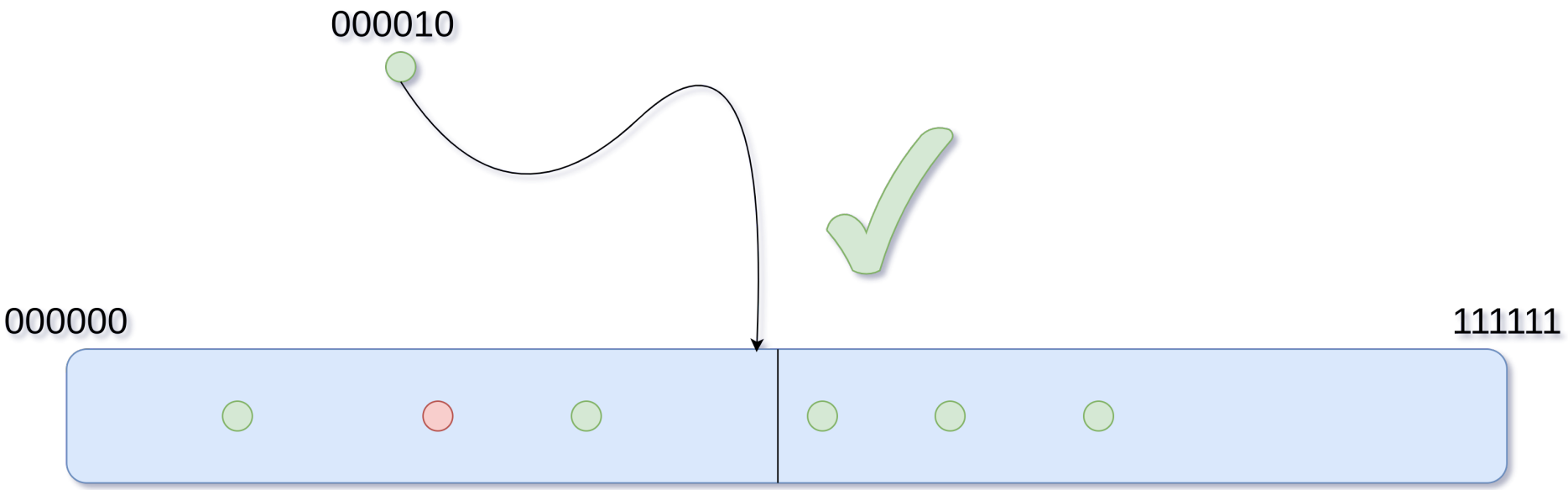
Routing Table



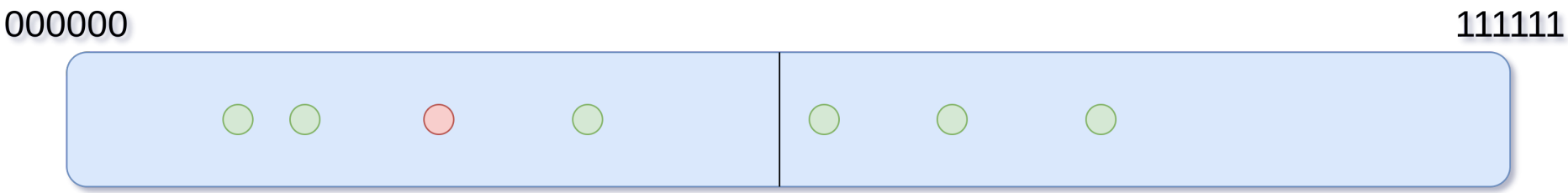
Routing Table



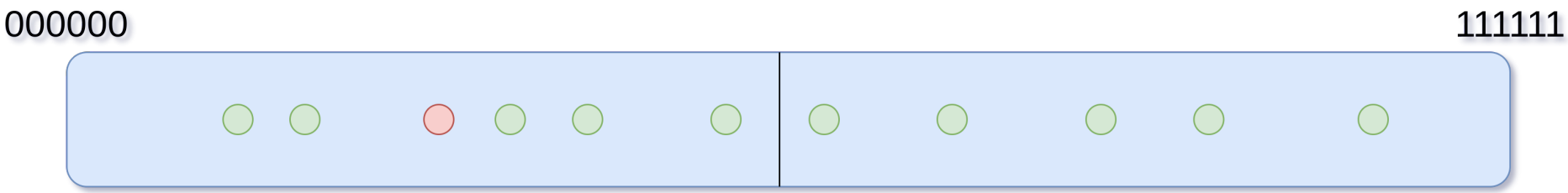
Routing Table



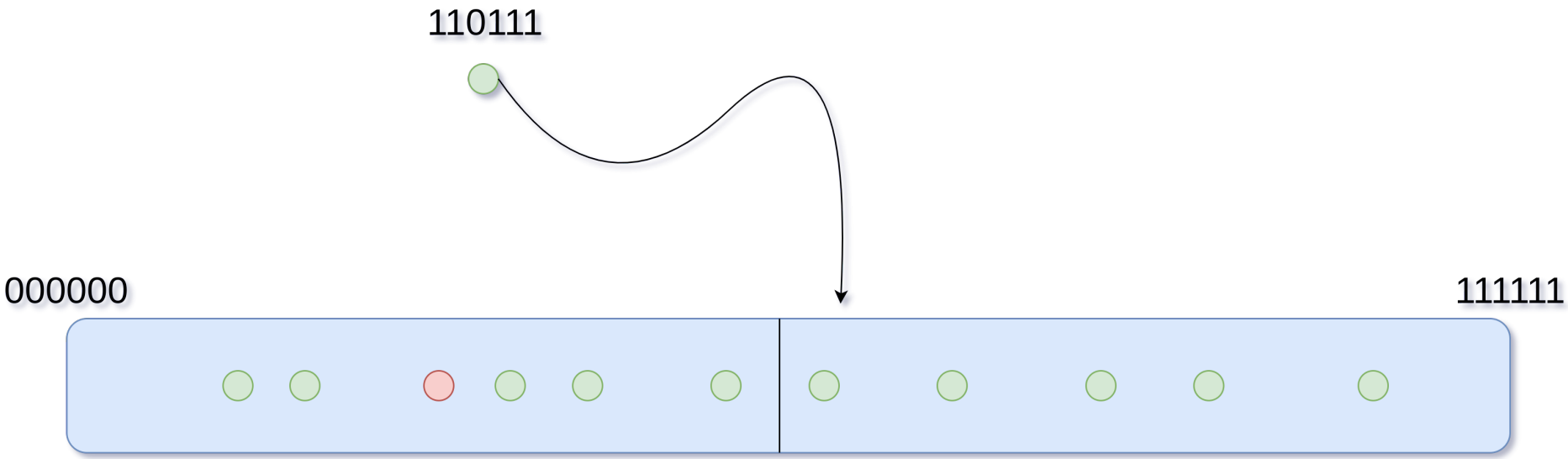
Routing Table



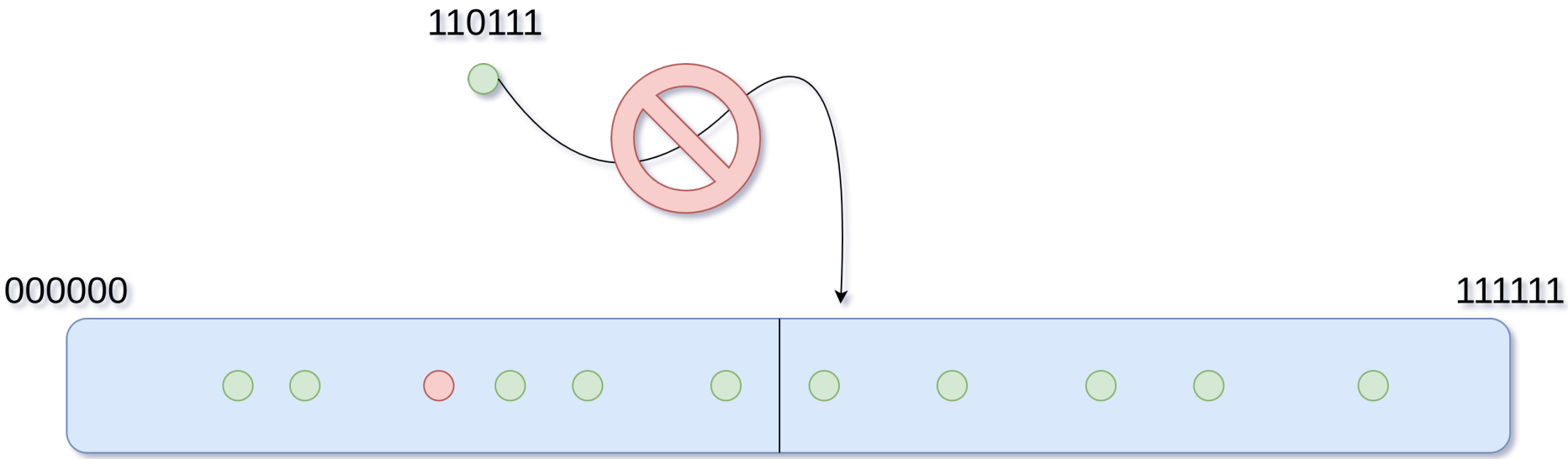
Routing Table



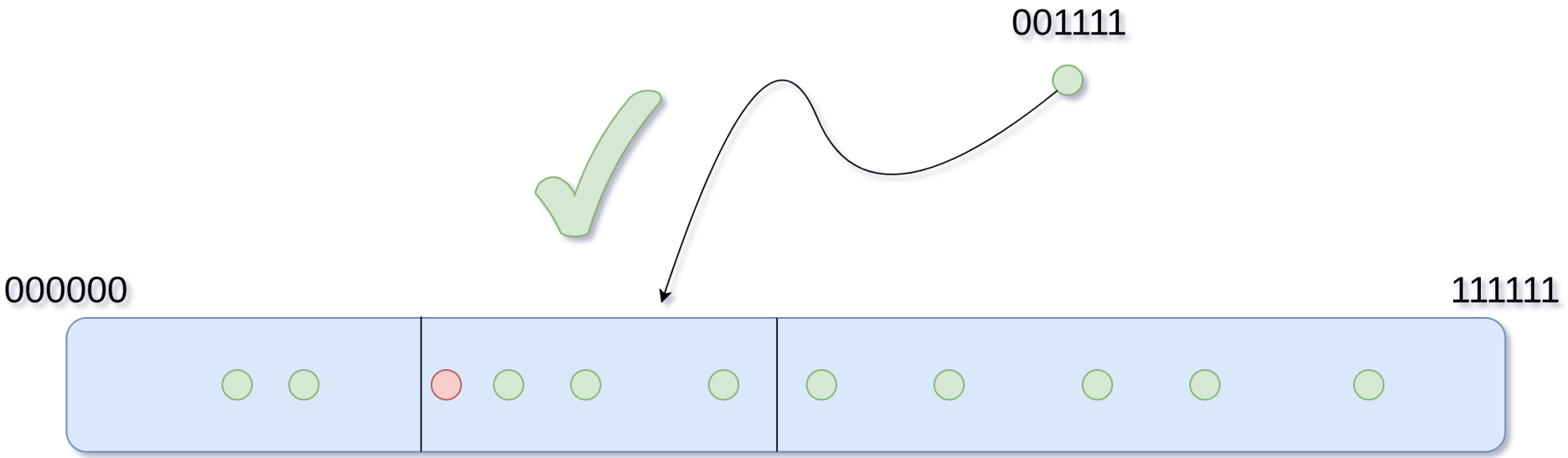
Routing Table



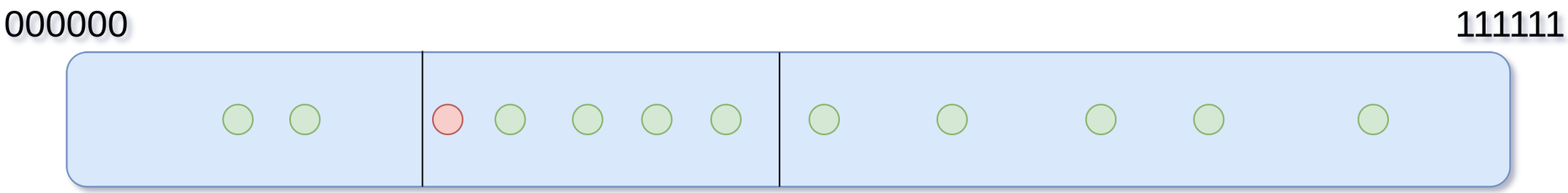
Routing Table



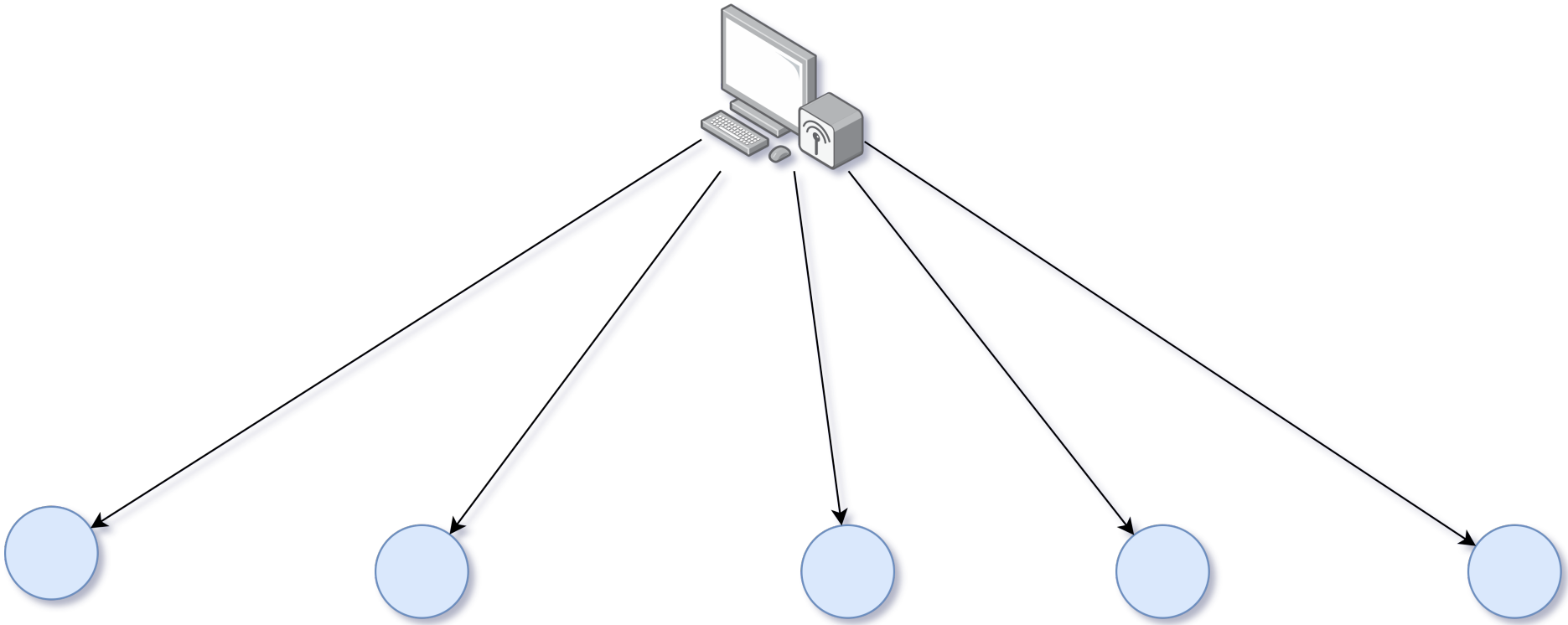
Routing Table



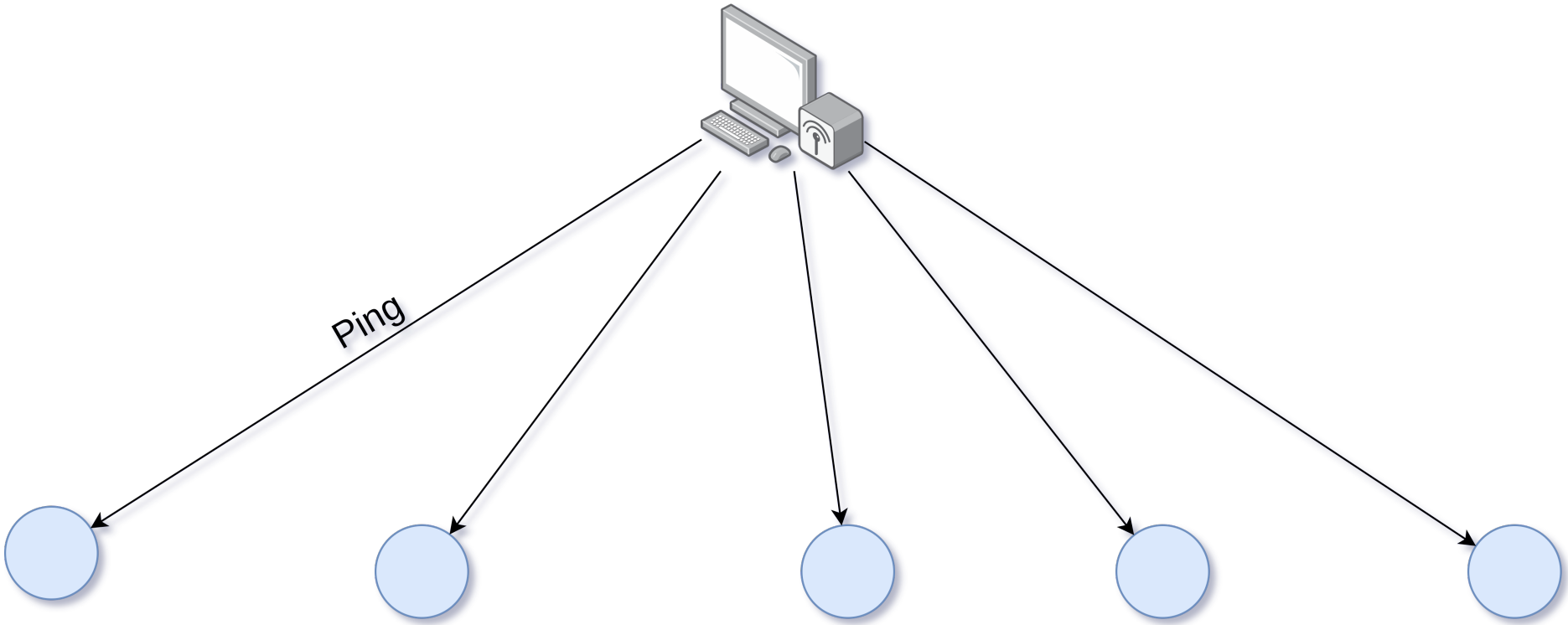
Routing Table



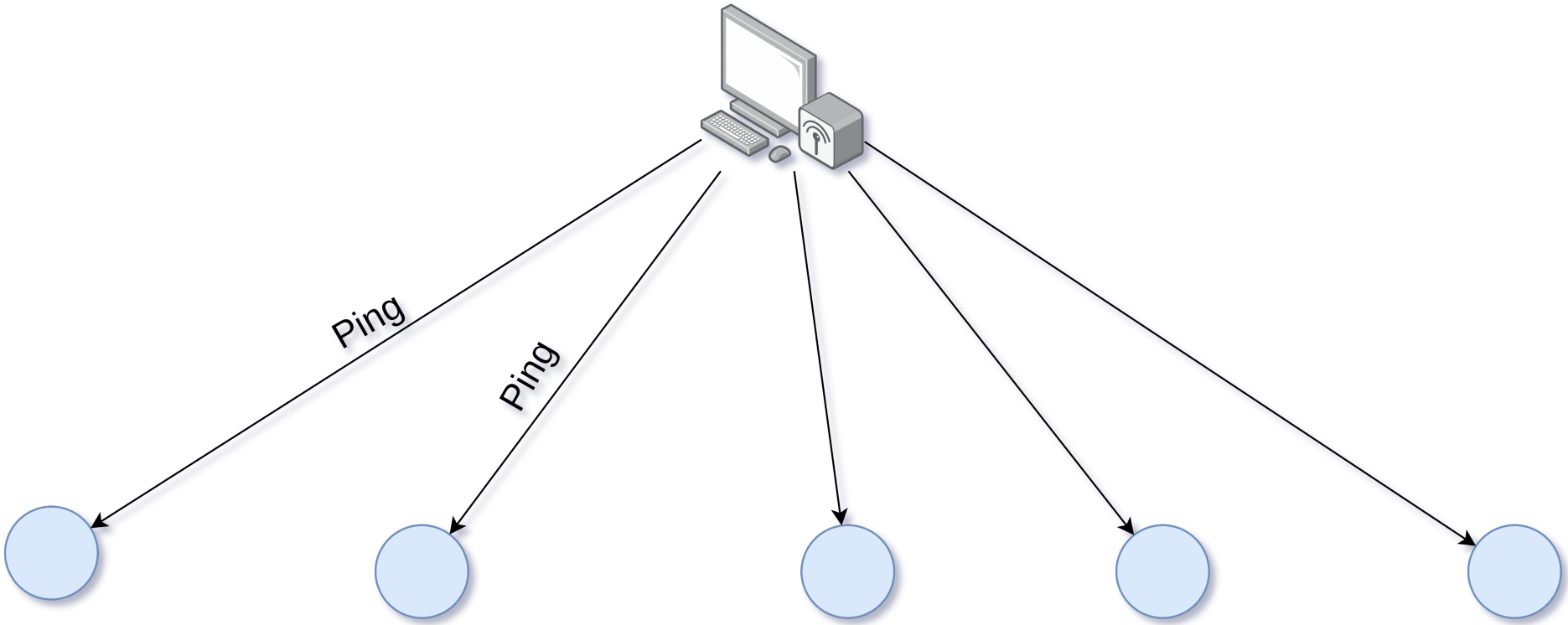
Eviction



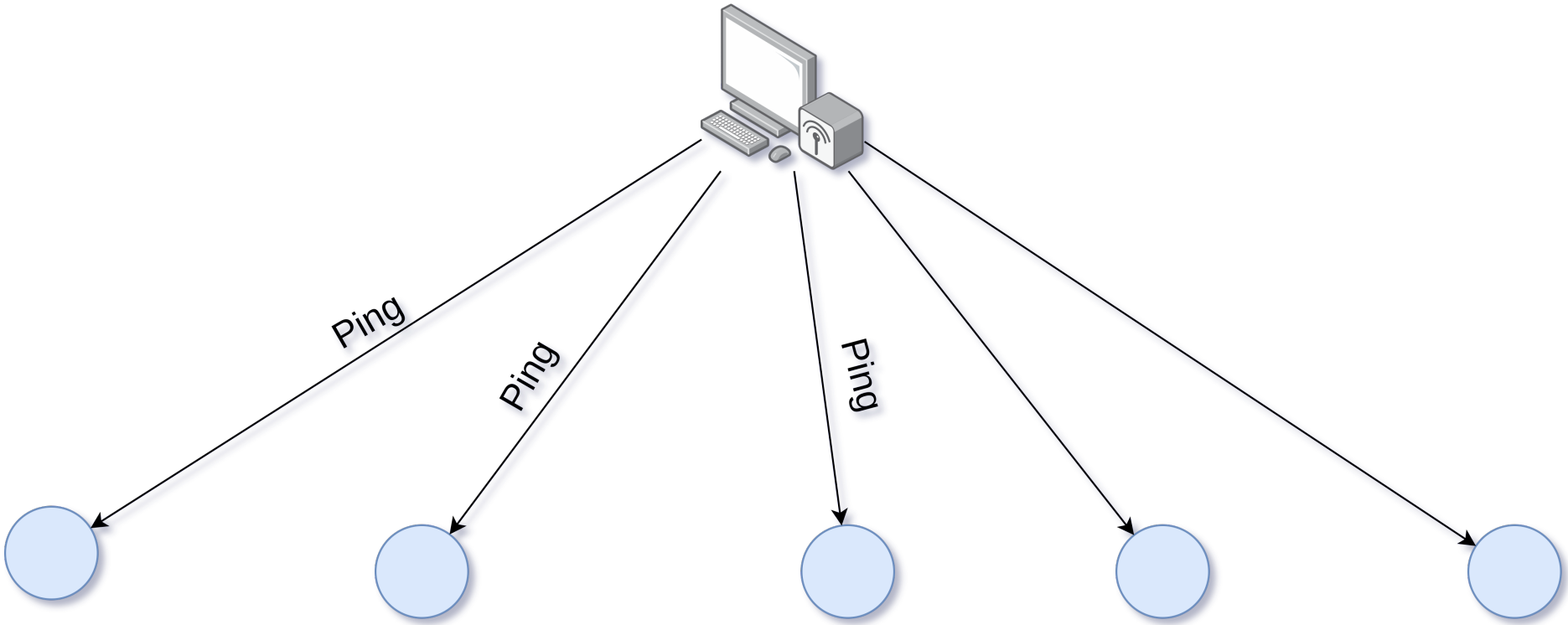
Eviction



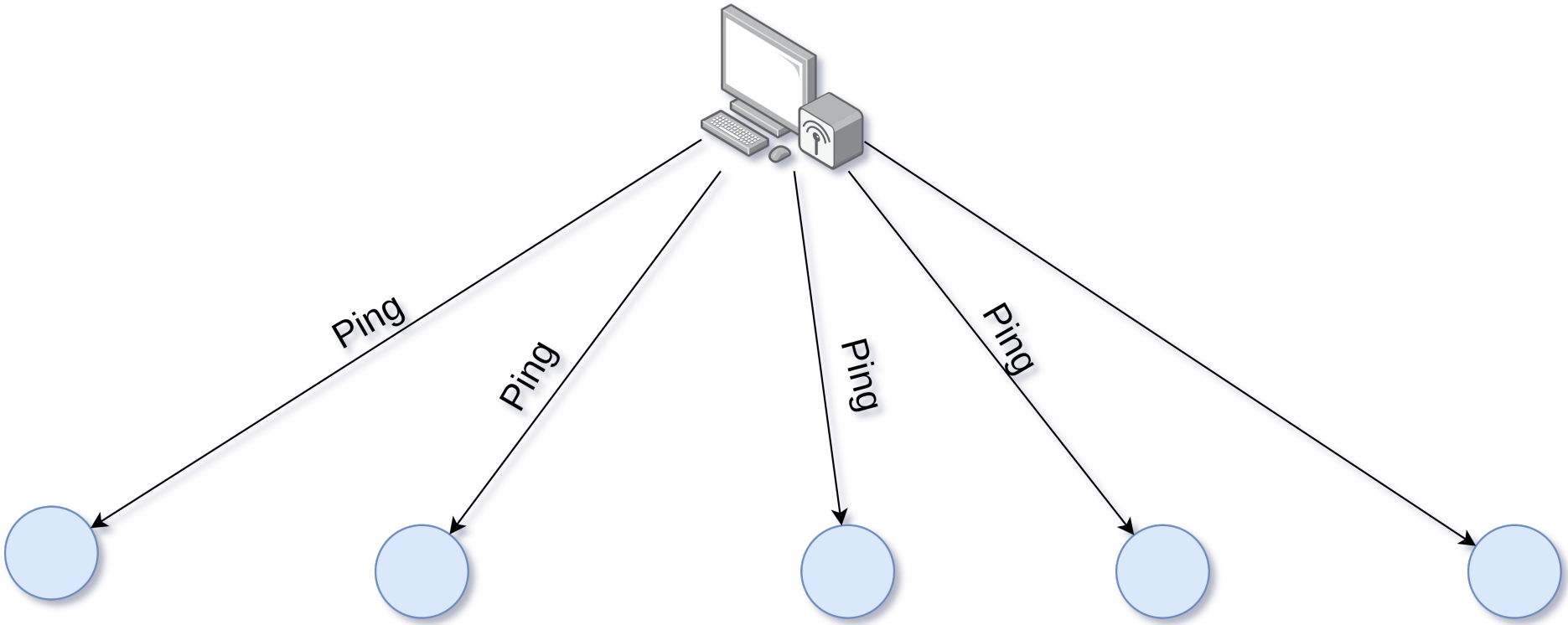
Eviction



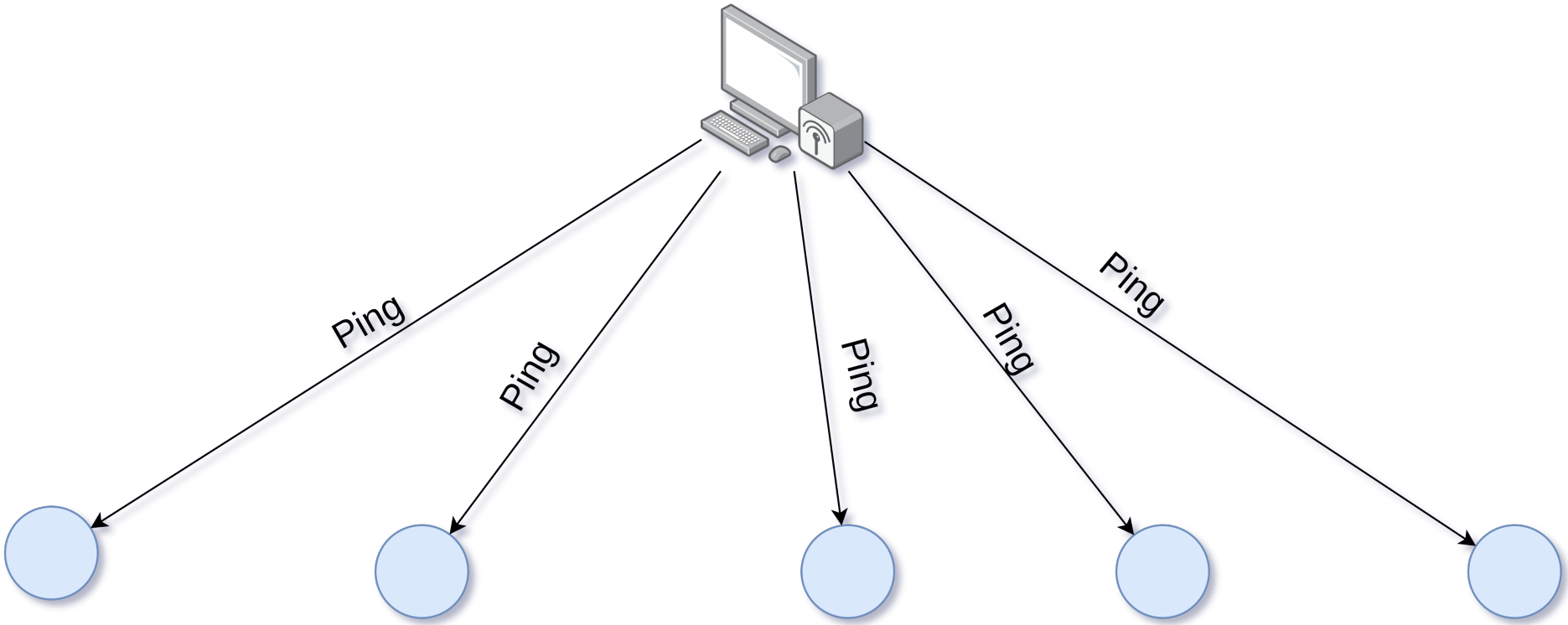
Eviction



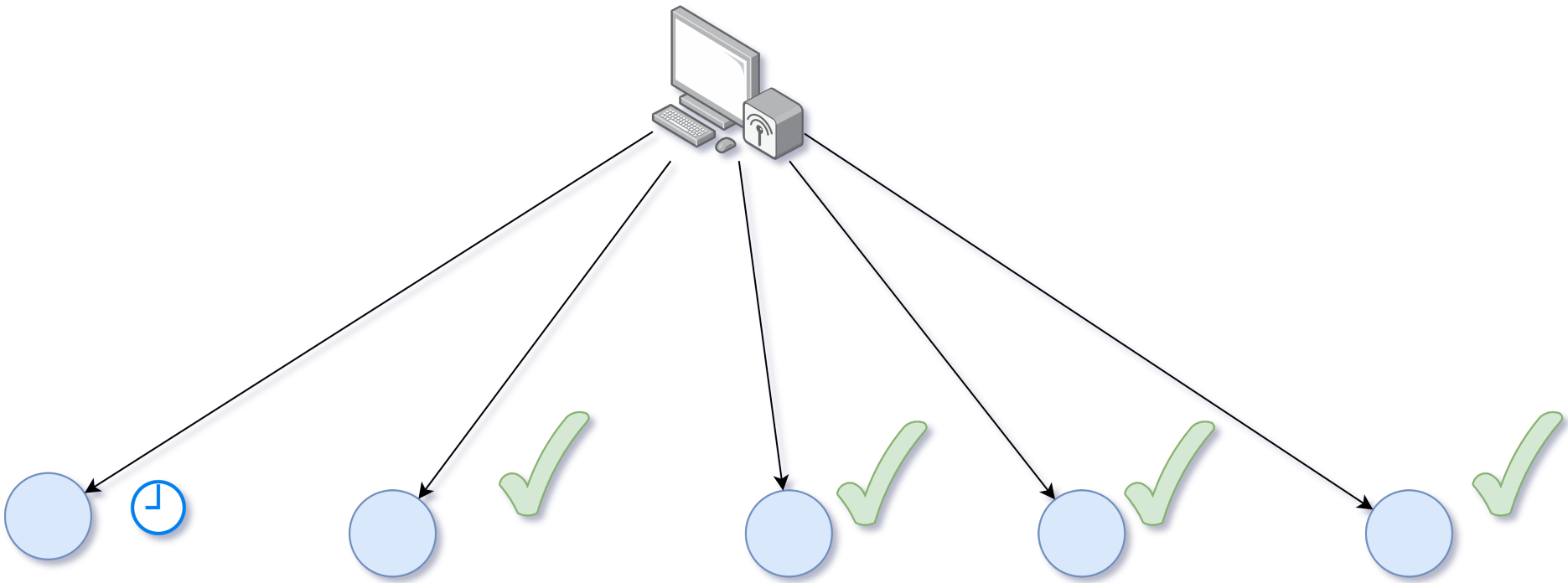
Eviction



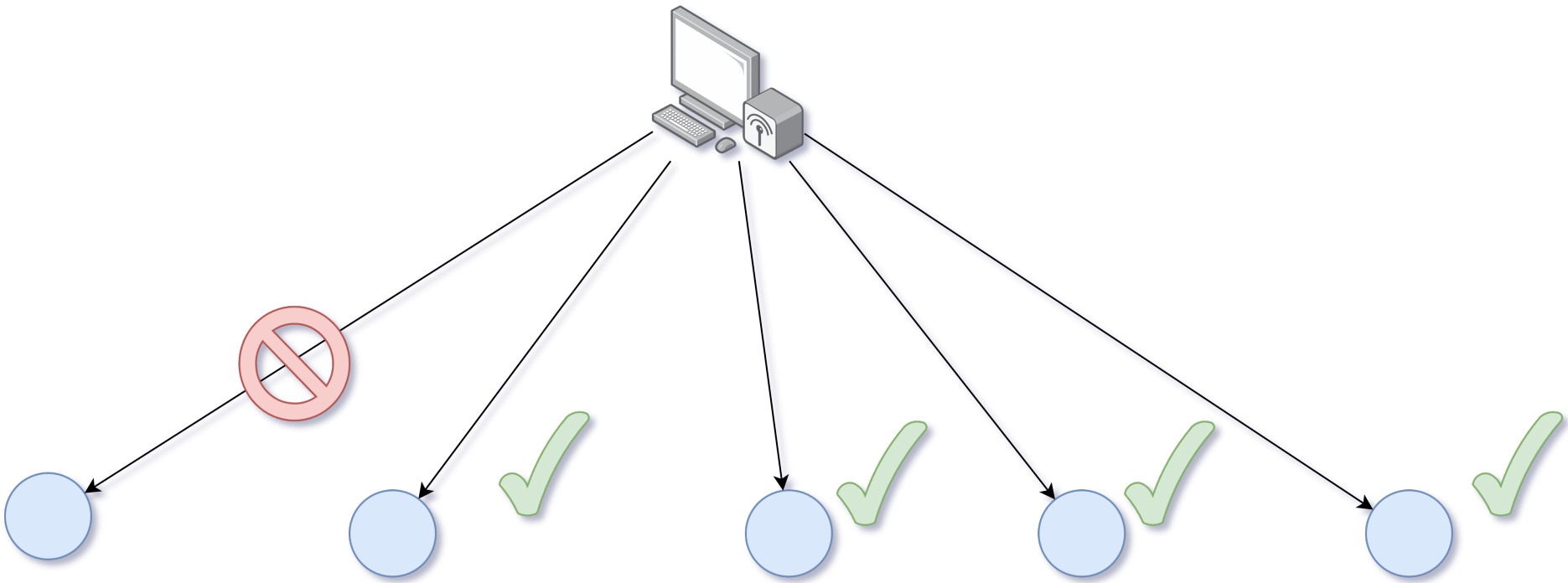
Eviction



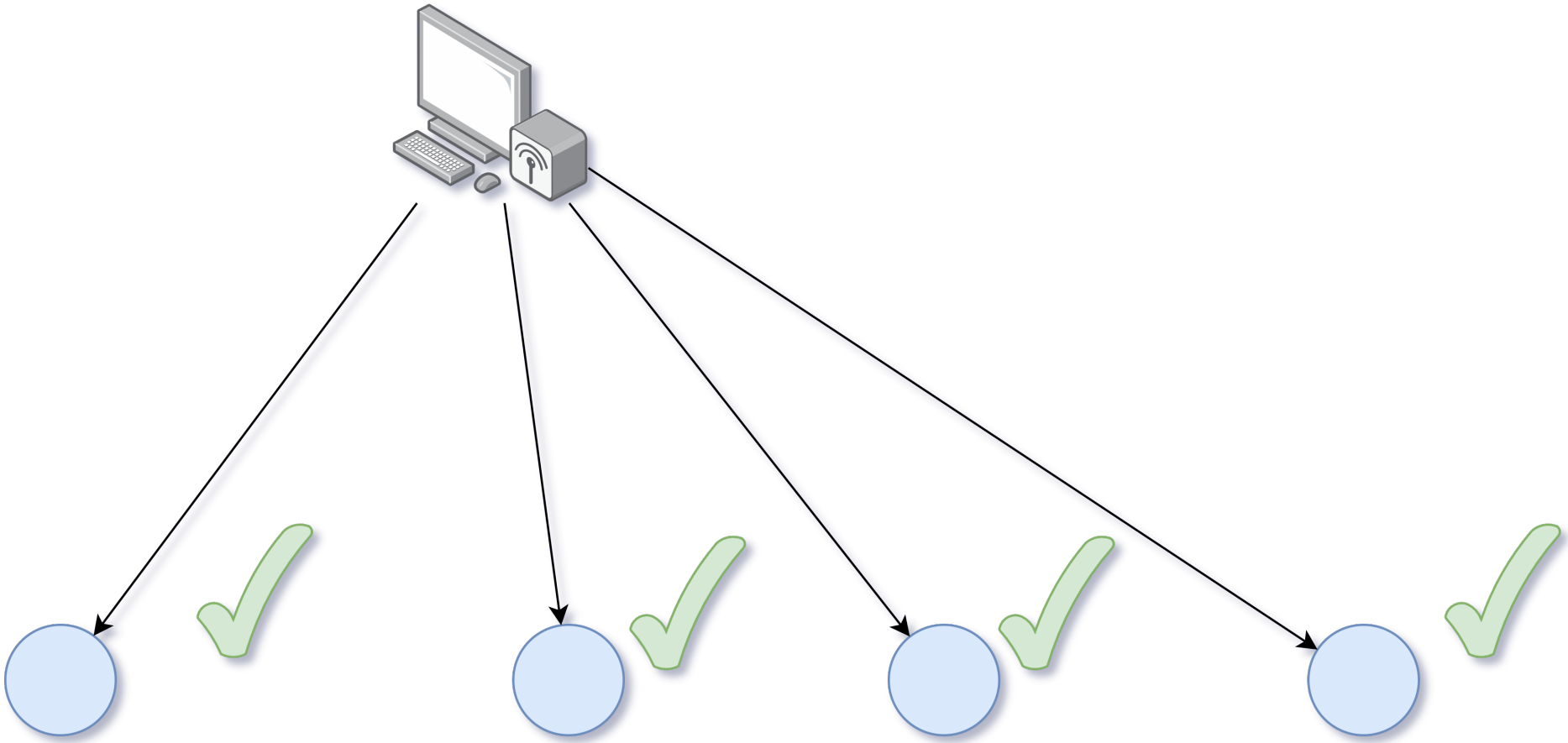
Eviction



Eviction



Eviction



Routing Table

- Storing nodes this way allows us to convert the address space into a directed graph.
- Lookups tend to be logarithmic.
- Allows a traversal accross the entire address space by storing a lot of information about our neighbors, but allowing a wide distribution to "far-away" nodes.

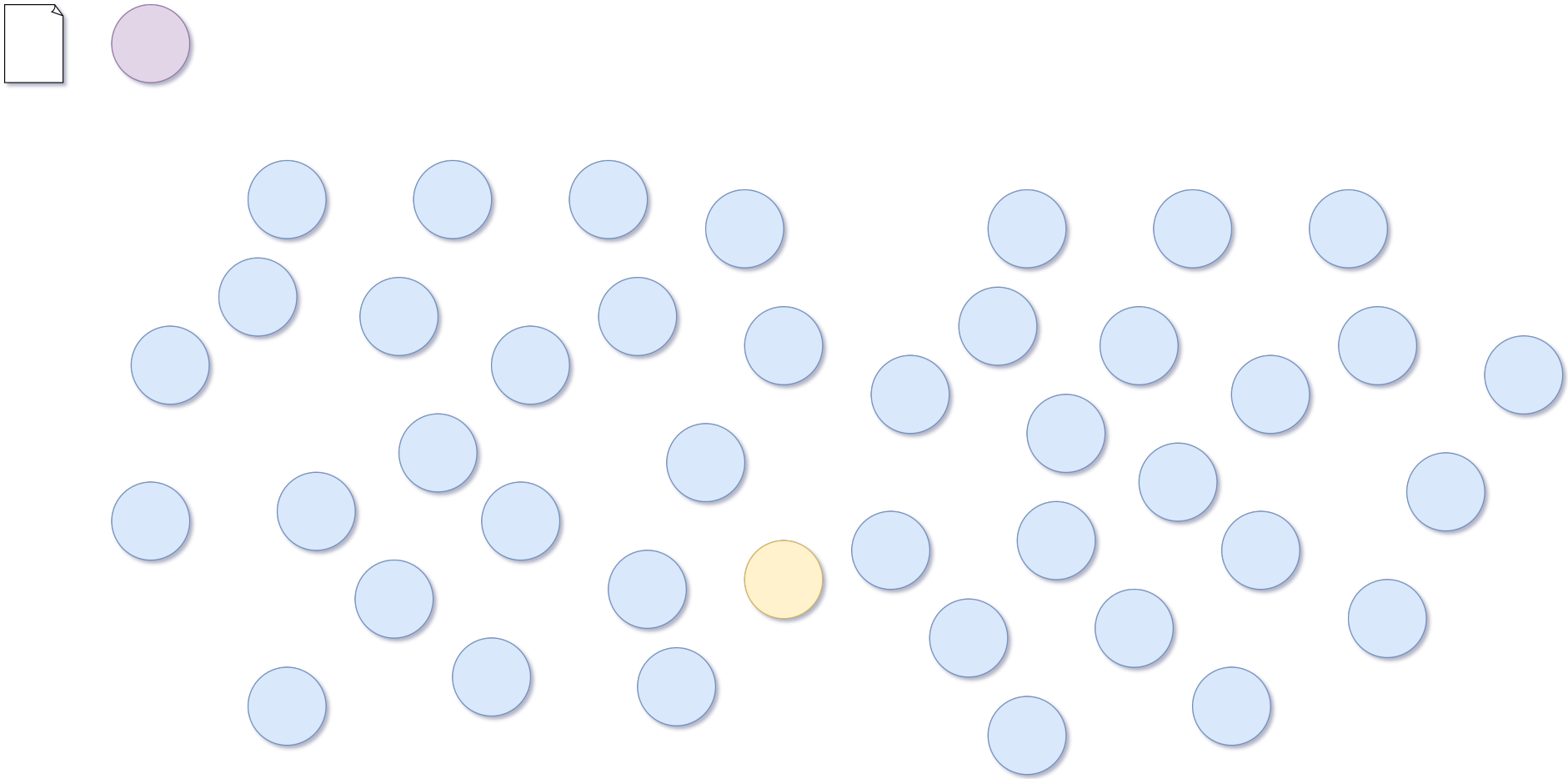
Section 5

Put and Get

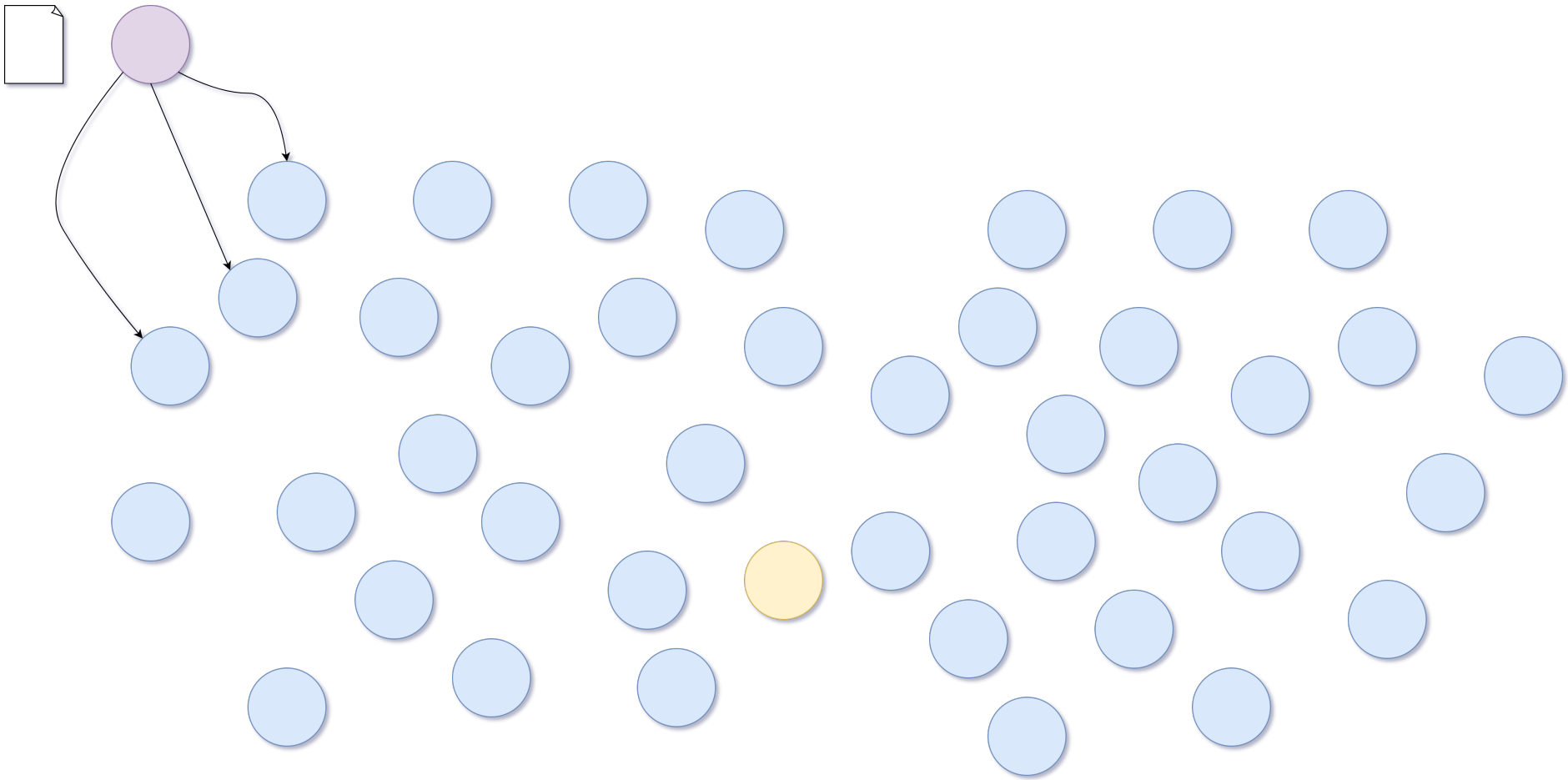
Hashing

- Like a normal hash table, keys are hashed before insertion.
- The hash length *must* be the same length as the node ID length.
  - By doing this, we keep our graph directed and allow for deterministic traversal.

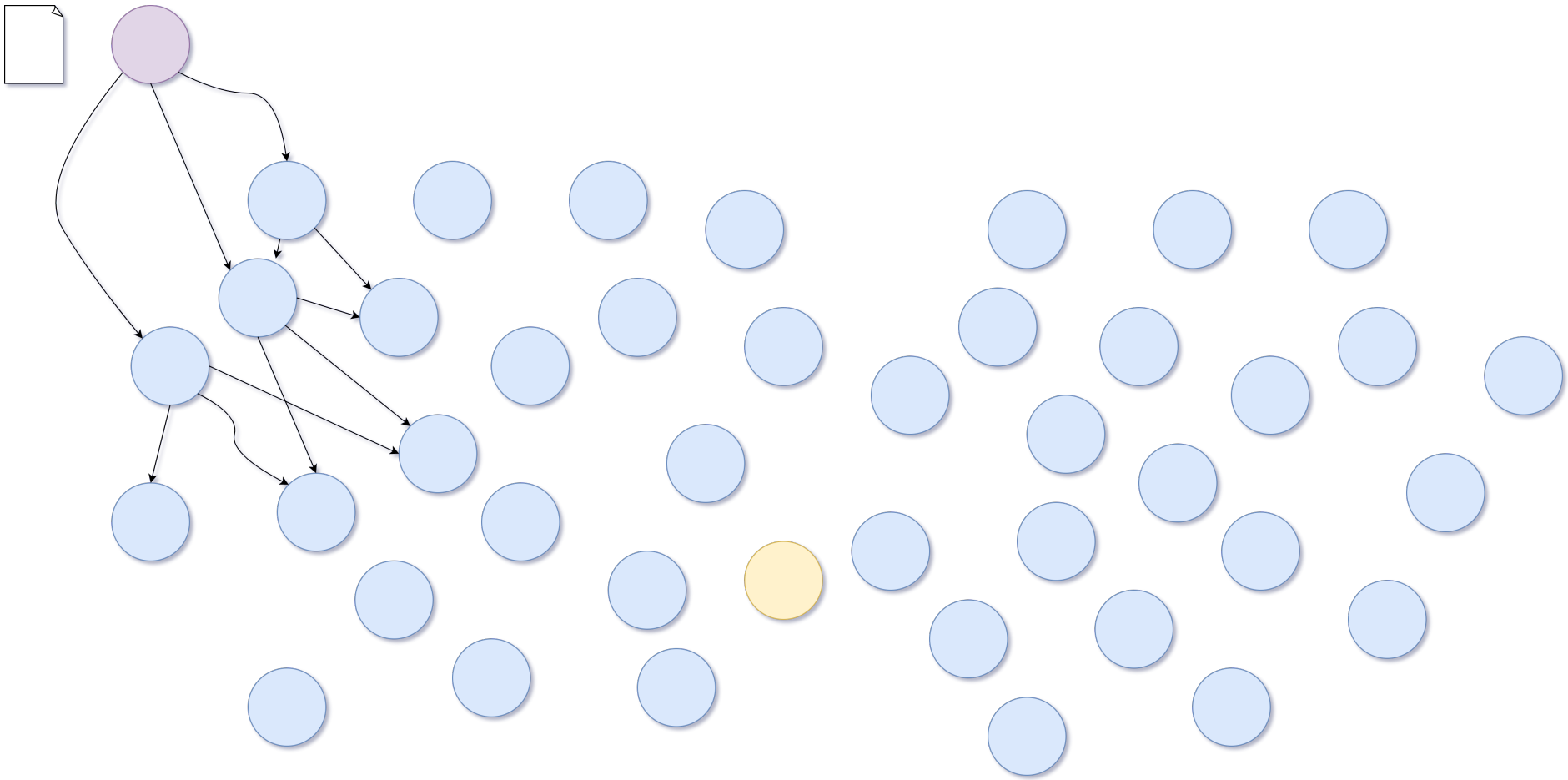
Put Logic



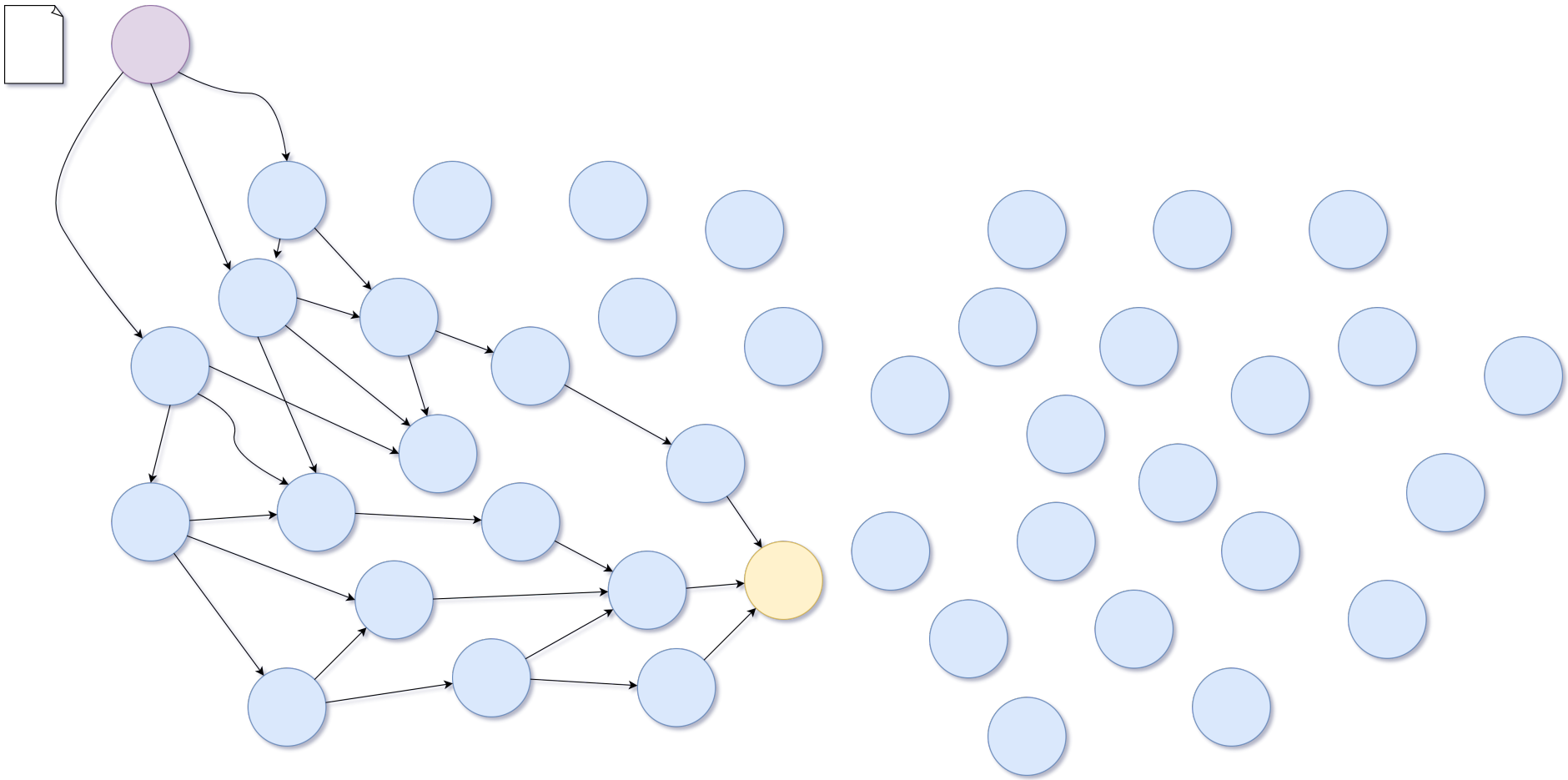
Put Logic



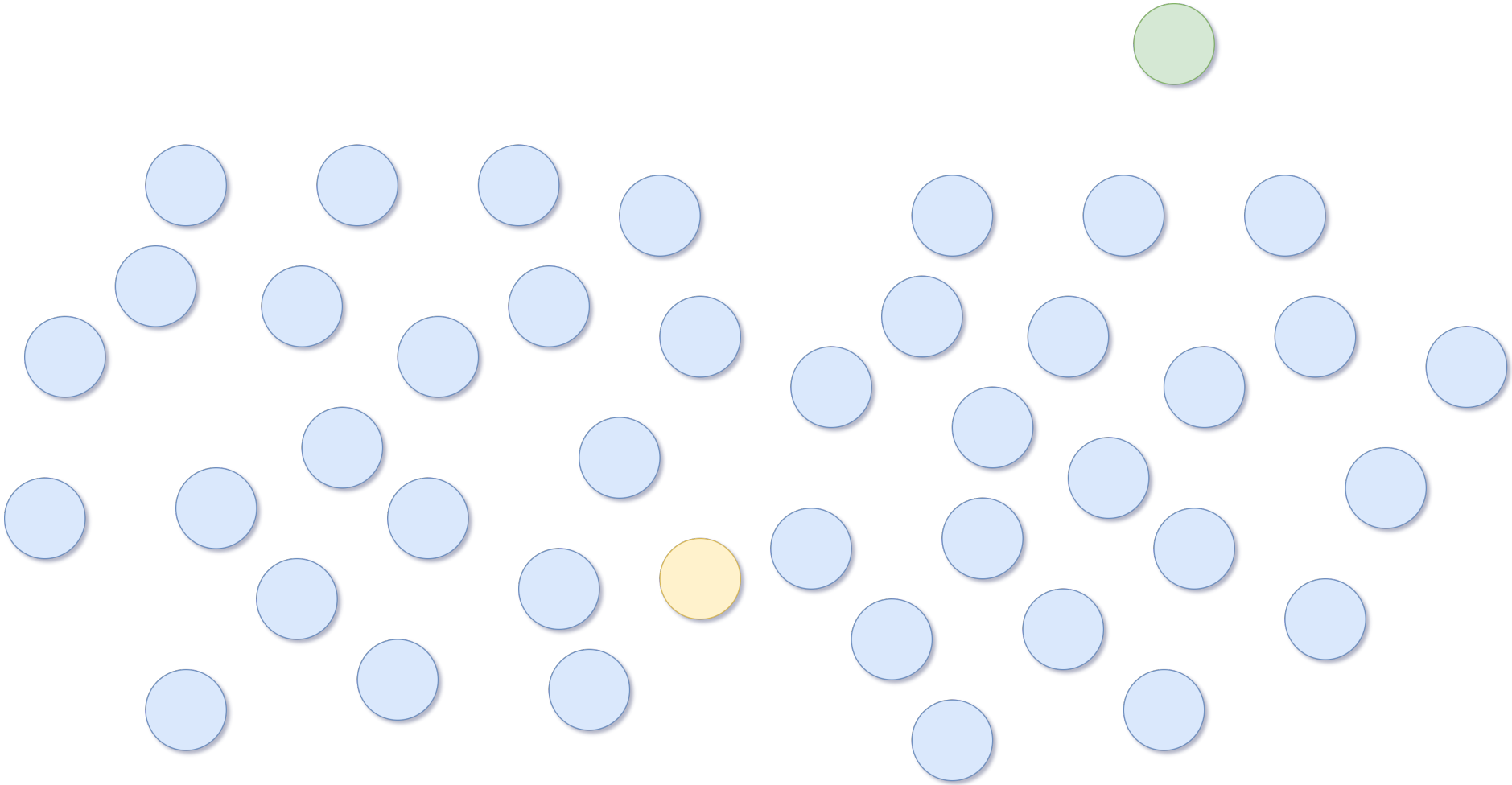
Put Logic



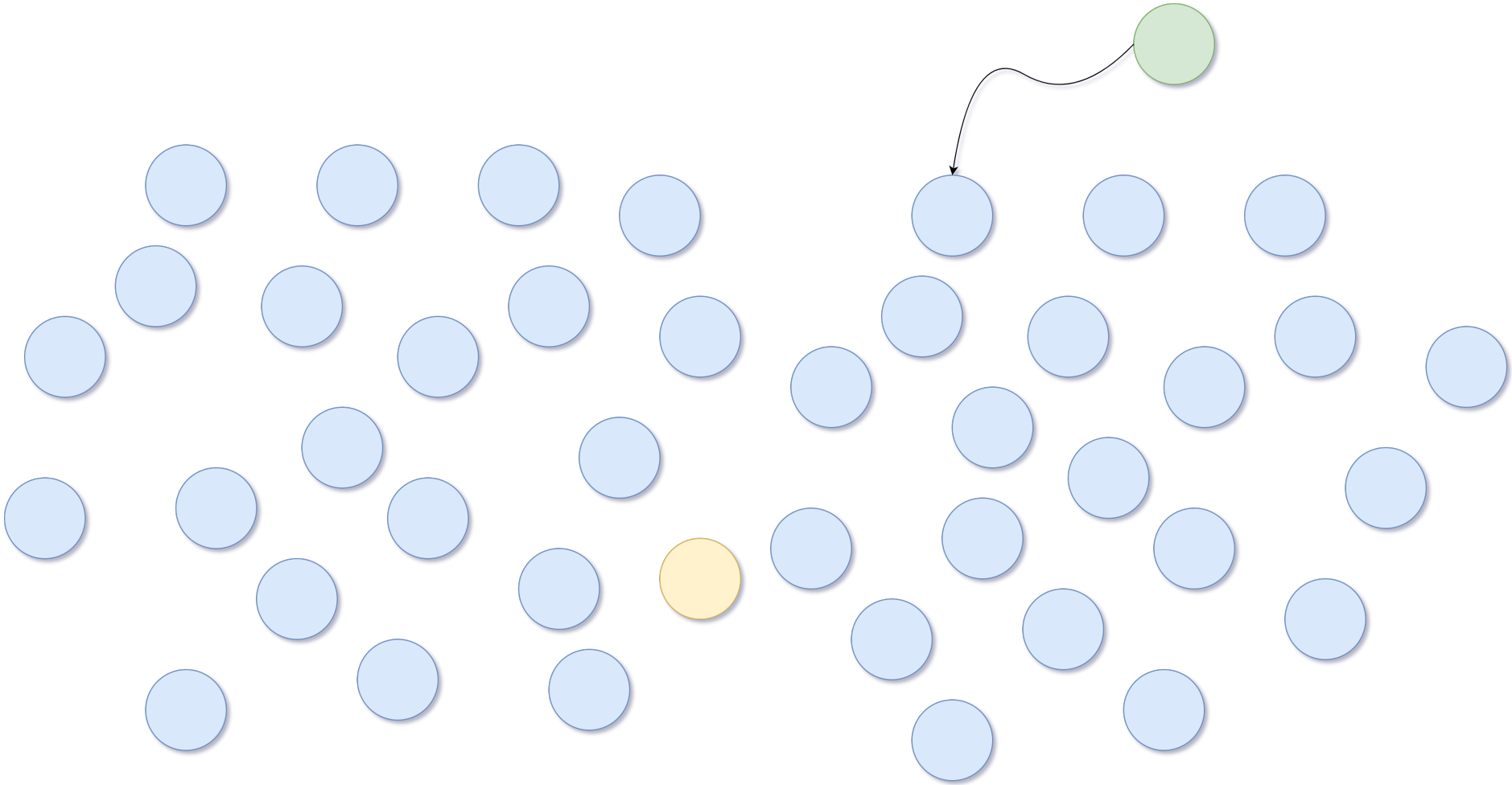
Put Logic



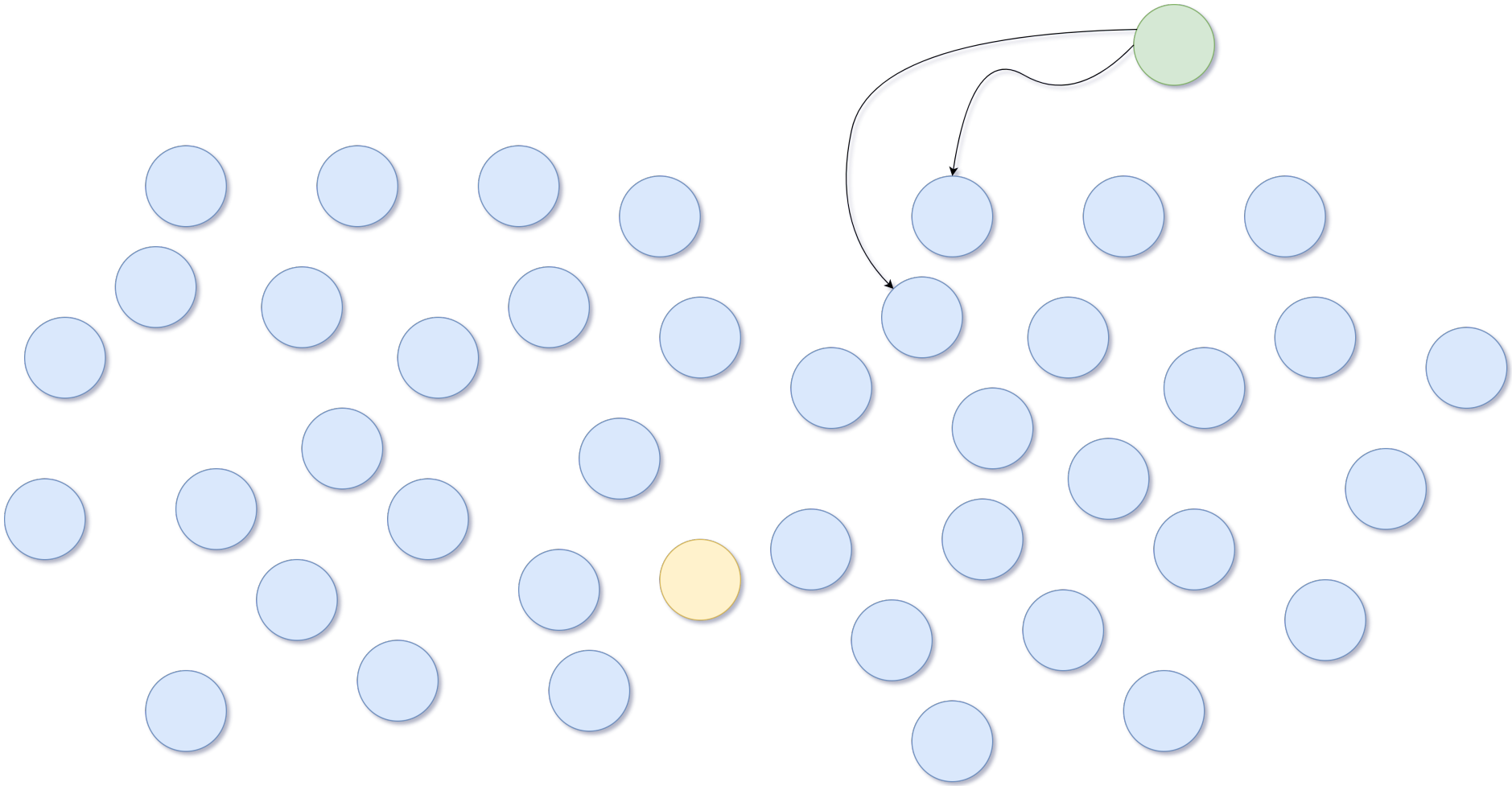
Get Logic



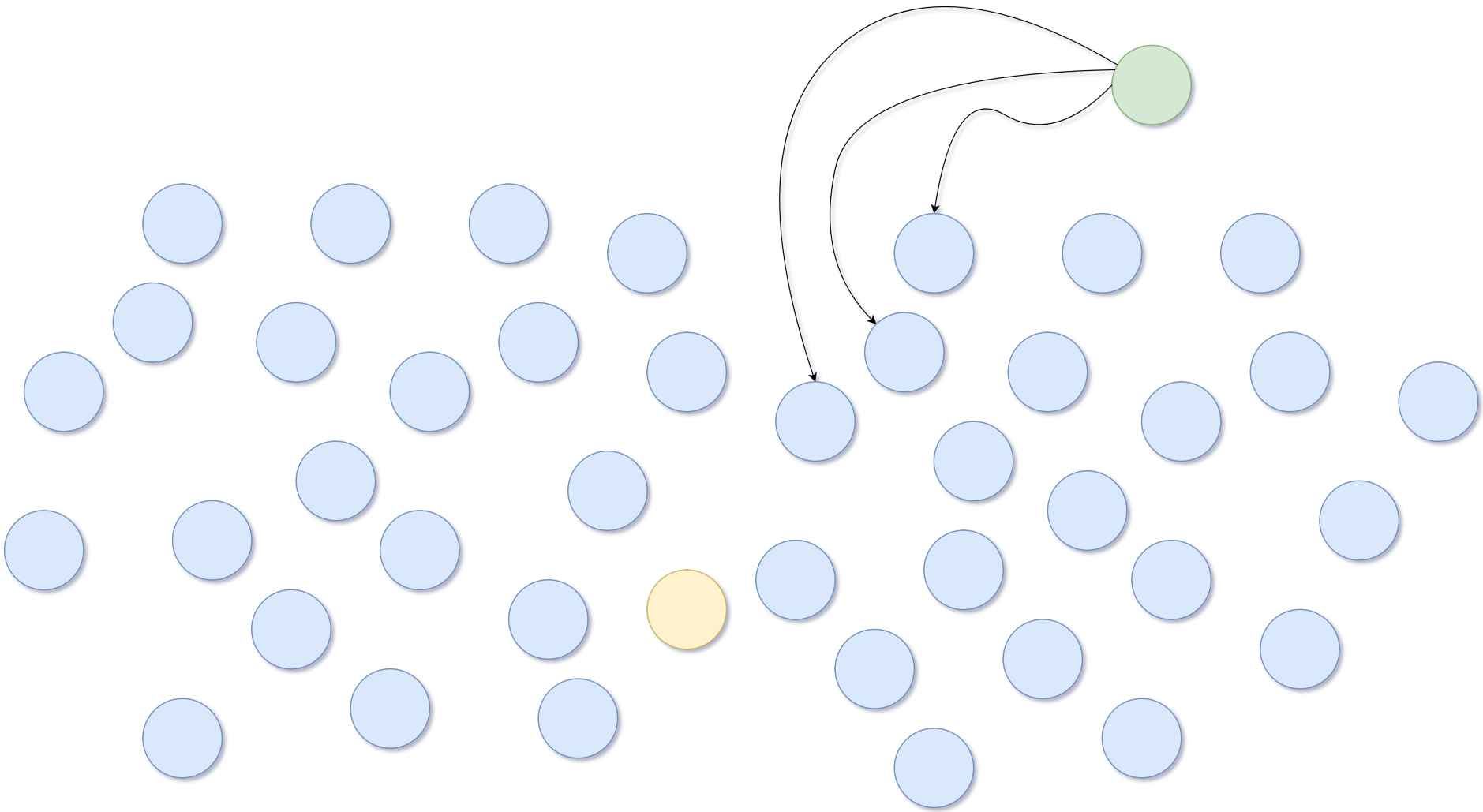
Get Logic



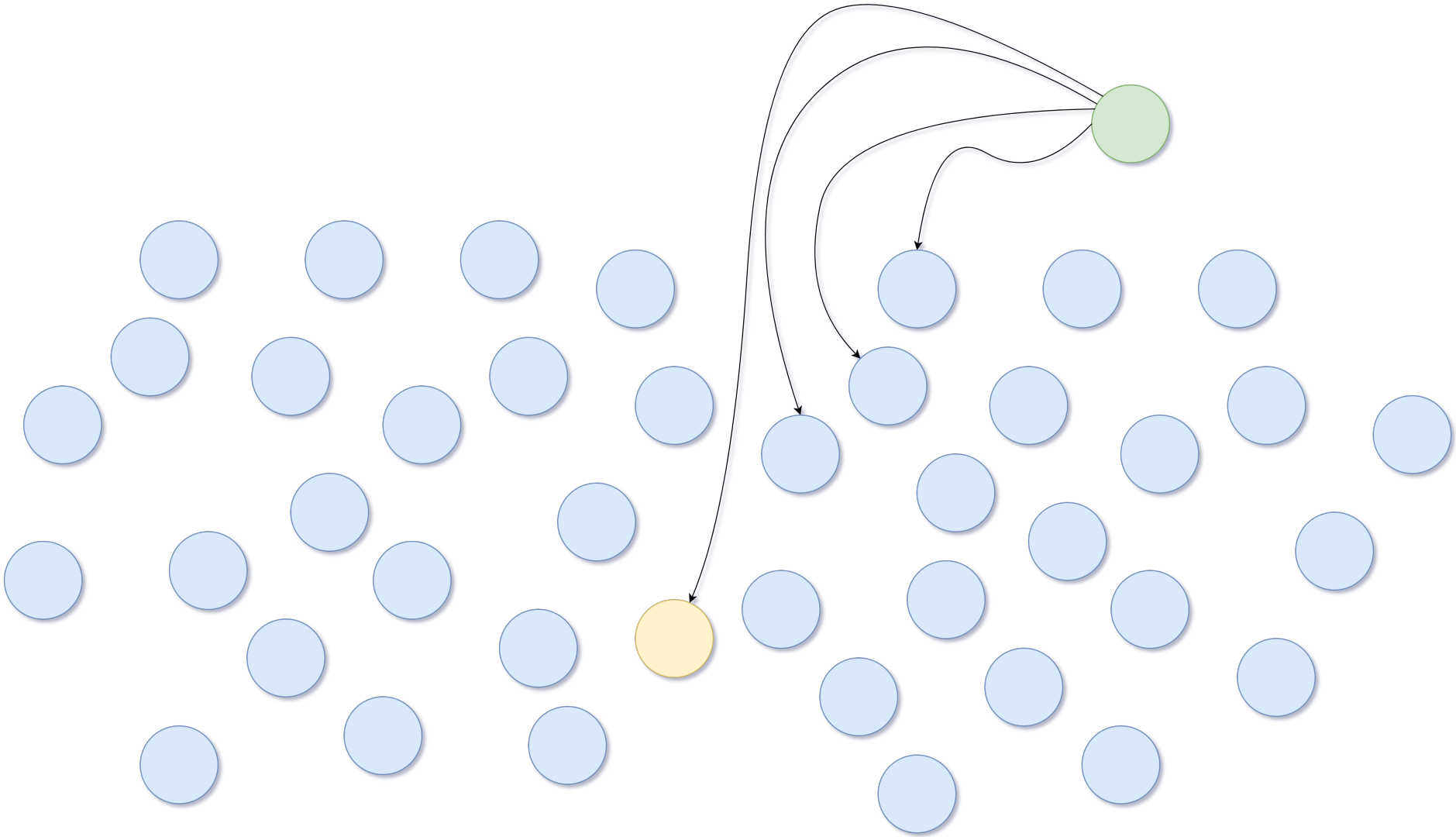
Get Logic



Get Logic



Get Logic



RPC

- For node communication, we defined an RPC protocol living on top of JeroMQ
- We define a multimethod to handle the dispatching of the message
- Kademlia mandates that you have the following RPC commands.
  - PING
  - STORE
  - GET
  - FIND

Section 6

NATs

NATs

- Network Address Translation
  - The feature of your router or firewall that allows you to share a single IP address across many devices.
  - Notoriously hard to work around when doing network applications.

NATs

Petar Maymounkov <petar@maymounkov.or… Mar 8, 2018, 5:53 PM

to me

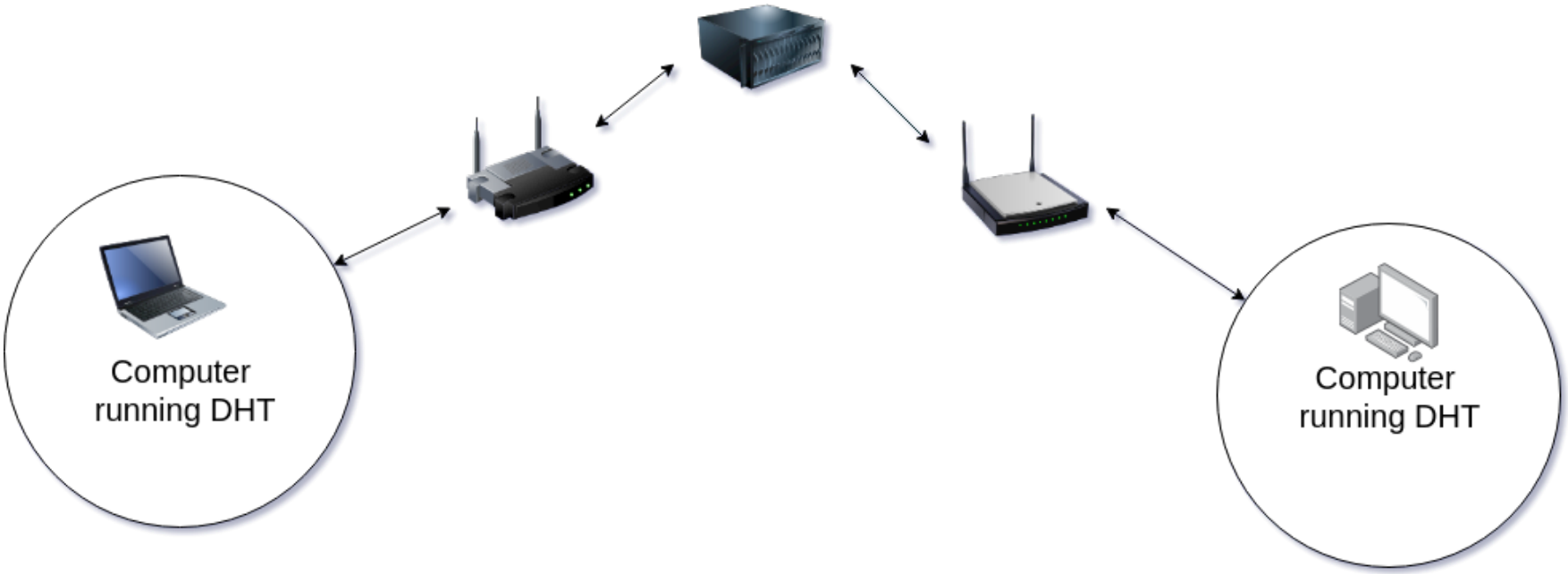
NATs are a hard problem. a few years ago there simply wasn't a good solution in oss. I don't know now. I actually still don't know of a solution on this.

Hole Punching

- External computers don't know how to talk directly to computers within NAT (without a port forward)

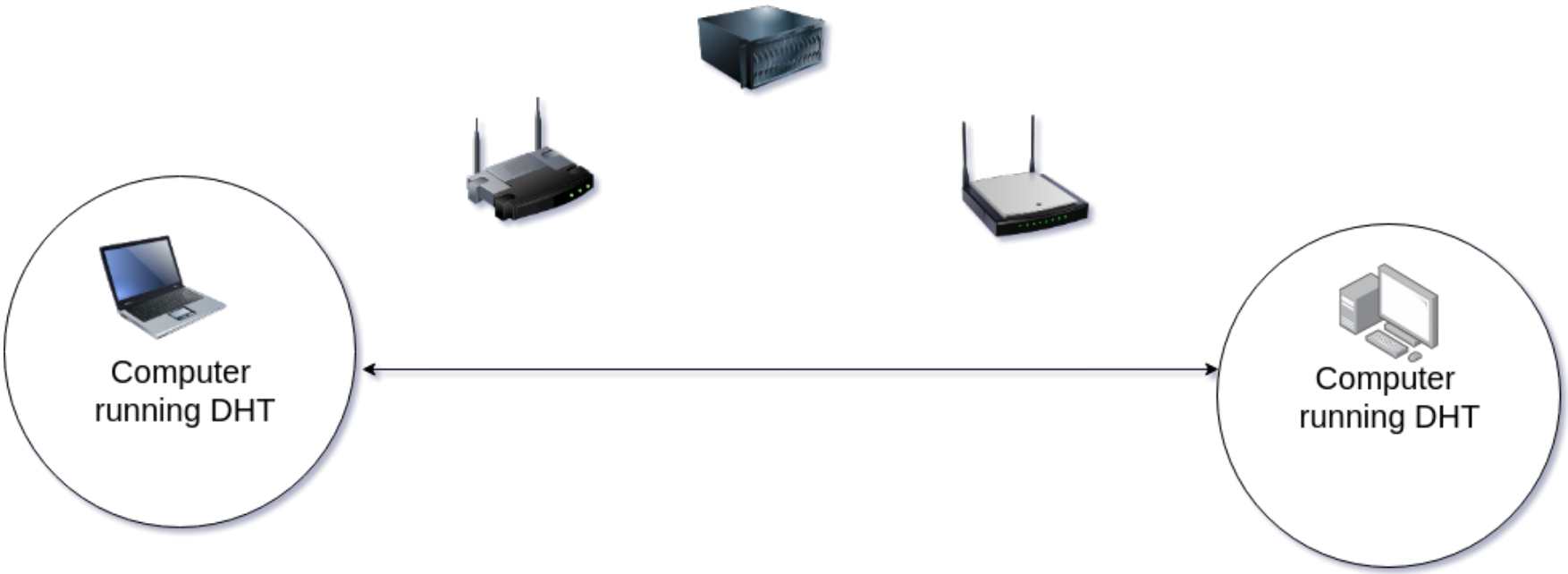
Hole Punching

- One solution, both computers connect to a server



Hole Punching

- Since the socket is open to both computers, the server can stich together a direct connection.



Hole Punching

- In order to keep this decentralized, we use the following algorithm.
  - Any node that has an open port gets priority on the routing table.
  - We maintain one constant connection to a node with an open port per bucket.
  - Each node will consistently publish any nodes with open ports that it has reference to as part of its metadata.
  - When trying to "get" from a closed node, a hole-punch will be attempted through a mutually connected node if it exists.

Changing Data

- It would be a bit of a lame user experience if we don't give the user the ability to update anything.
- Distributed systems can be really difficult to figure out which update happened most recently.

Vector Clock.

- Vector Clocks allow to create a partial ordering of data updates in a distributed system.
- Items are tagged with a counter (independent of wall-time), and this counter is incremented upon each time the information is touched by another entity.
- When two entities have created incompatible updates to data, a conflict is created and it must be resolved.
  - We cheat and during a conflict, we rely on wall-clock time and choose the latest from there.

Section 7

Frontend

A word on JavaScript

- I actually *like* JavaScript.
- JavaScript **punishes** the OOP programmer.
- Forced me to adopt functional programming techniques such as data/function seperation and composition.
- Has significant gaps though:
  - Large parts of the standard library focus around mutation
  - Missing pieces that should be a part of the language but aren't
  - "Pit of Despair" (classes, prototypical inheritence, null vs undefined)

Why ClojureScript?

- Fixes the problems I have with JavaScript while sharing similar philosophy and idioms:
  - Shared design ideology of working with many functions over relatively few data types
  - Embraces "dynamic programming" (dynamic typing, interactivity)
  - Has several useful features that JavaScript is trying to add

Future JavaScript today

- Optional Chaining (https://github.com/tc39/proposal-optional-chaining)
- Pattern Matching (https://github.com/tc39/proposal-pattern-matching)
- Pipeline Operator (https://github.com/tc39/proposal-pipeline-operator)
- Do Expressions (https://github.com/tc39/proposal-do-expressions)
- Protocols (https://github.com/michaelficarra/proposal-first-class-protocols)
- Macros (https://github.com/kentcdodds/babel-plugin-macros)

Other Niceties

- Immutability by default
- Expression-based
- Homoiconicity
- One null type
- No == vs ===

Reframe vs React

- HTML Templating:
  - React's JSX brings the pain of HTML into JavaScript
  - Awkward escapement rules trip up newbies
  - Reframe's use of native data structures largely sidesteps these issues
- State Management:
  - Reframe is a similar model to Redux, but with far less headaches
  - Out of the box support for returning effects, including async
  - Generally more consise and straightforward to write

Section 8

Roadmap

In Progress

- Formal verification of the system using TLA+.
- Tonika Routing to prevent against potential attacks.
- Allowing for "private" videos by employing the use of GPG.
- Using blockchain (or something similar) to incentivize people to open up ports and/or donate their CPU for more interesting computation tasks.

Section 9

Further Reading!

Links

Kademlia Paper

Feross Aboukhadijeh's talk

Contact

- Thomas Gebert
  - thomas@gebert.app
  - gitlab.com/tombert
- Nicholas Misturak
  - Twitter: @nrmisturak
  - gitlab.com/nrmisturak

Section 10

Demo

Demo