

Lecture 06: Peer-to-Peer Networking on Ethereum

Akash Khosla
Nick Zoghb



BLOCKCHAIN
AT BERKELEY



LECTURE OUTLINE

1



BUILDING A MENTAL MODEL

2



DISTRIBUTED SYSTEMS FAULTS AND ATTACKS

3



PEER-TO-PEER PROTOCOLS

4



DEV2P DEEP DIVE



ETHEREUM DECOMPOSITION

A MENTAL MODEL

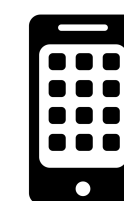
3



Transactions



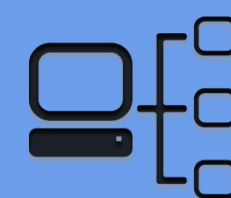
Identity



Application



Semantic



Propagation



Mining



Consensus

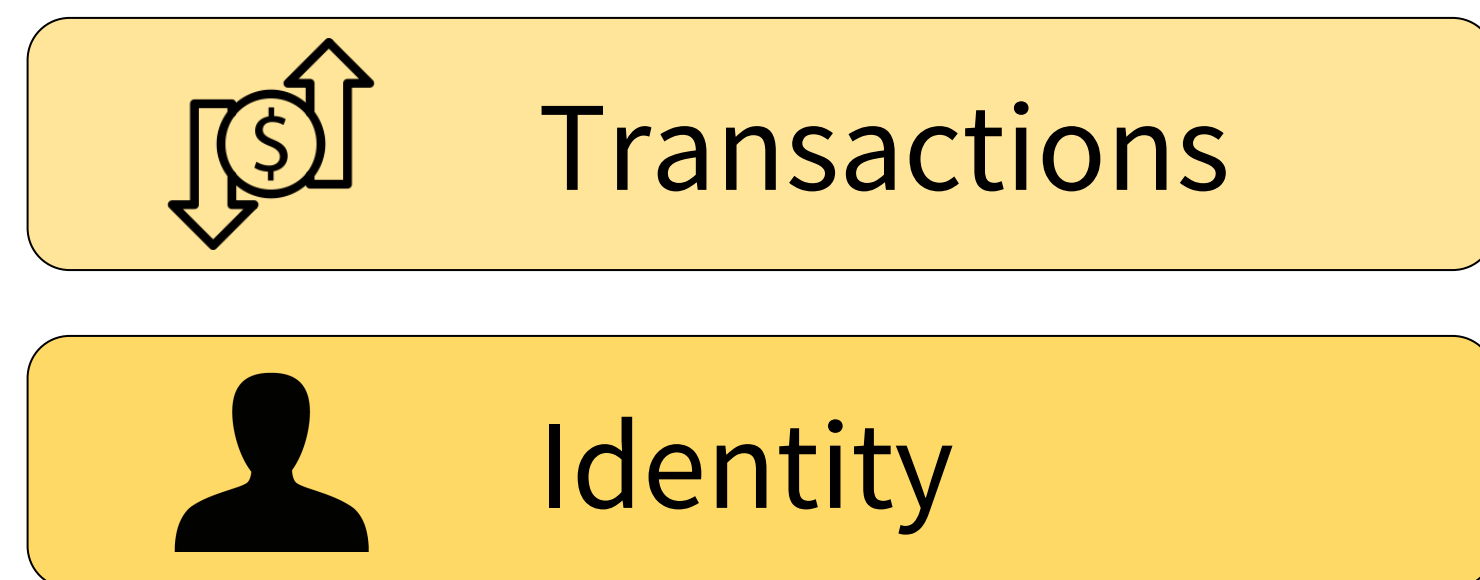




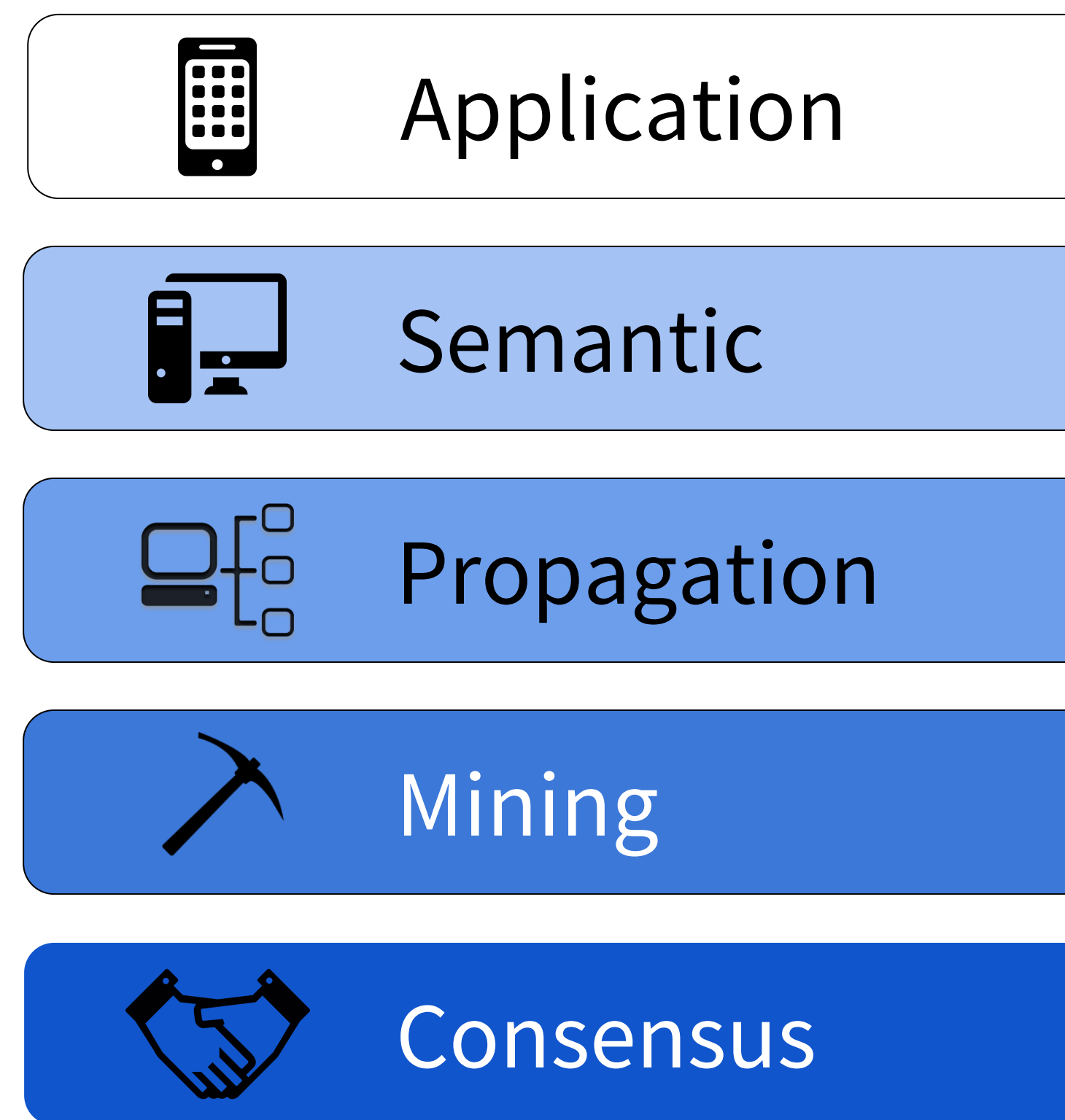
ETHEREUM DECOMPOSITION

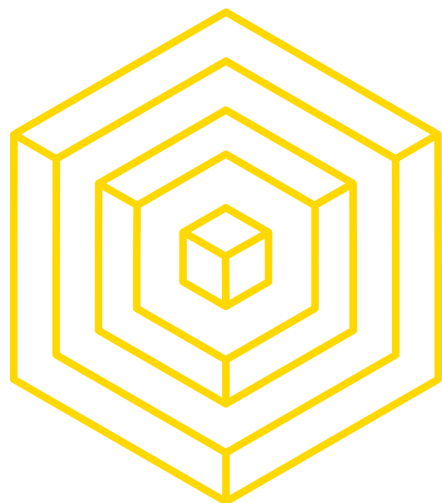
A MENTAL MODEL

4



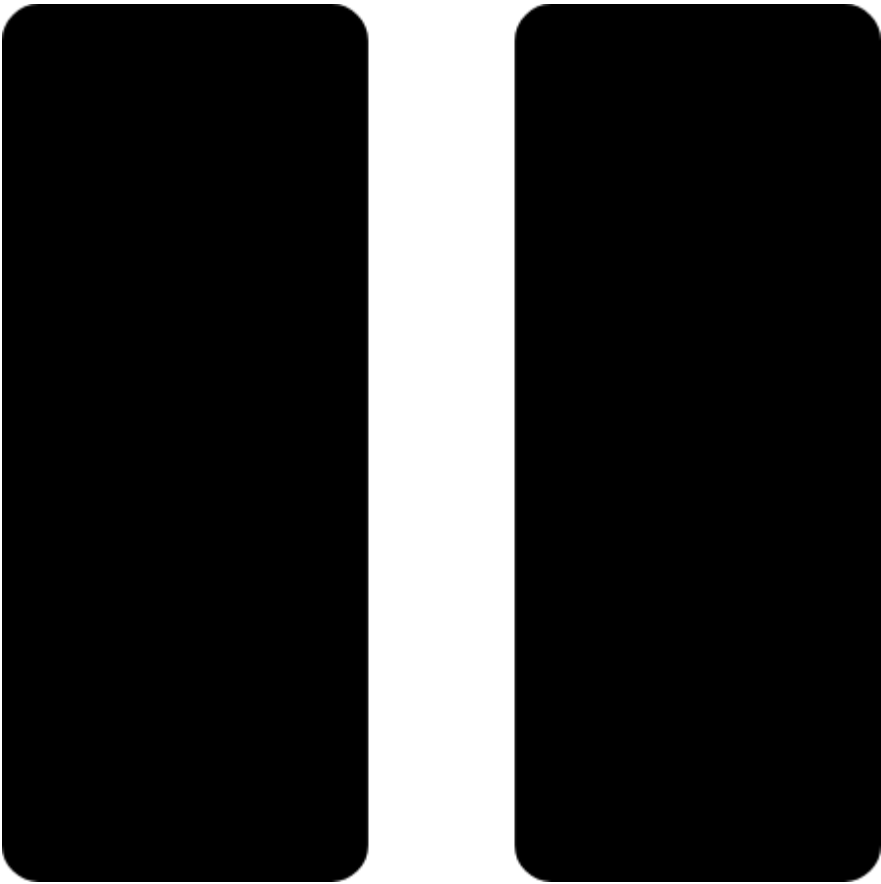
Let's look at
all this





PAUSE

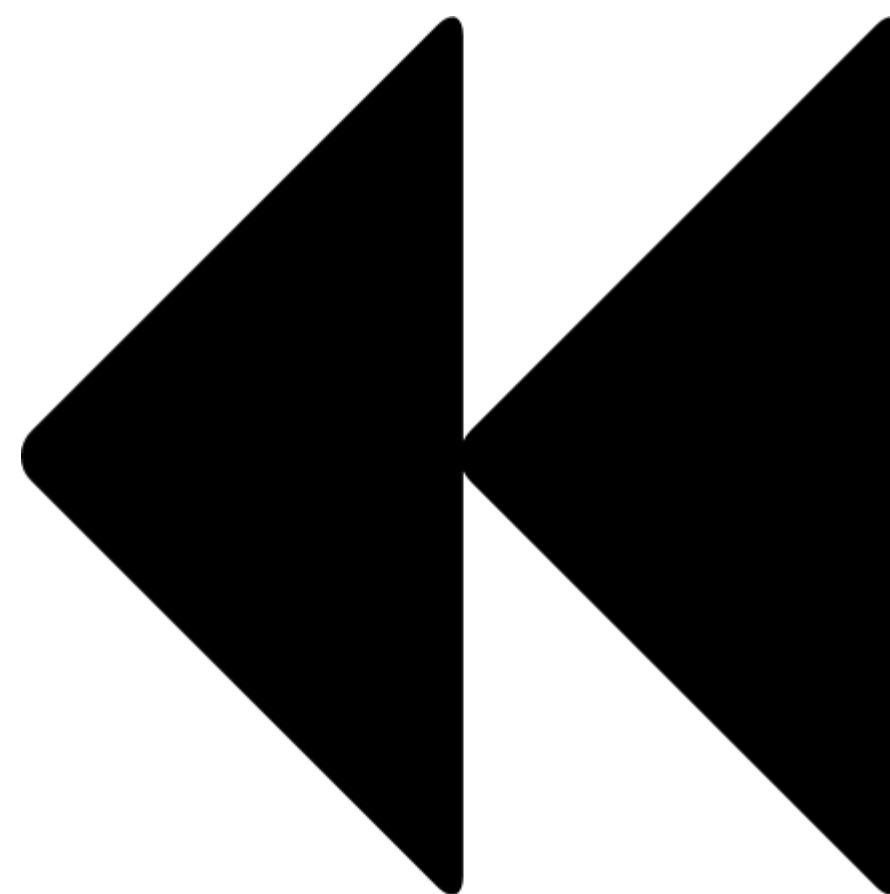
HOLD ON A SECOND





REWIND

LET'S TAKE A STEP BACK



1

BUILDING A MENTAL MODEL



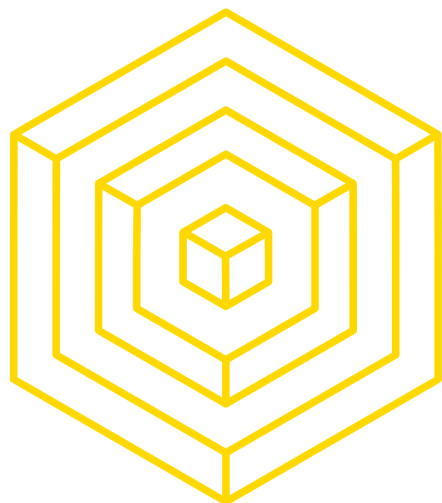
INTRODUCTION TO DISTRIBUTED SYSTEMS

WHAT ARE THEY?

Made up of a collection of entities that are:

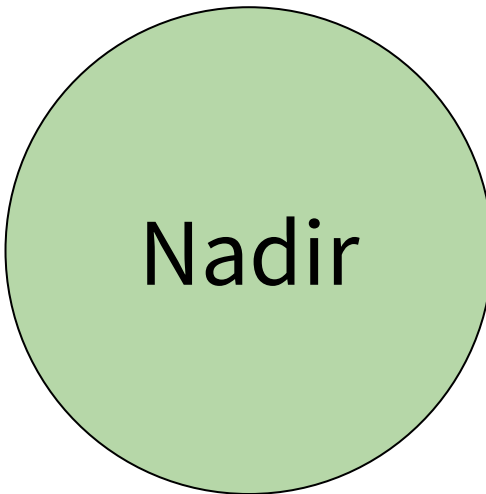
- Autonomous/Programmable
- Asynchronous
- Failure-prone
- Communicate through an unreliable medium

1.1 VISUAL EXAMPLES



A CENTRALIZED EXAMPLE

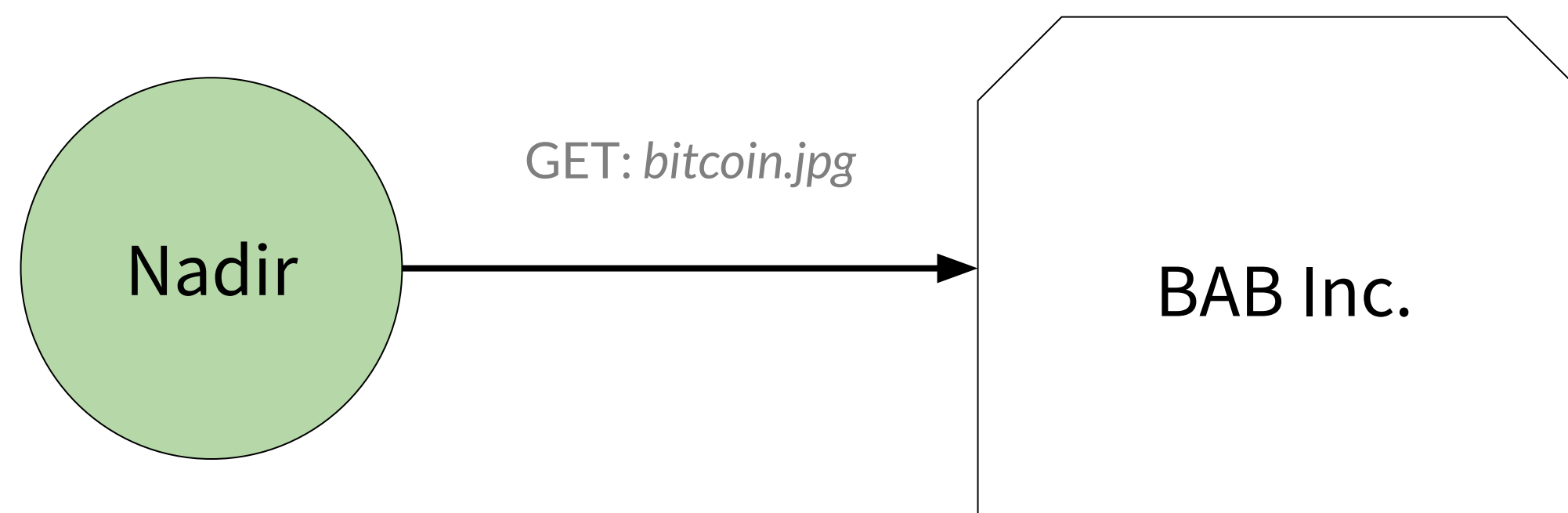
STARTING OFF SIMPLE





A CENTRALIZED EXAMPLE

STARTING OFF SIMPLE





A CENTRALIZED EXAMPLE

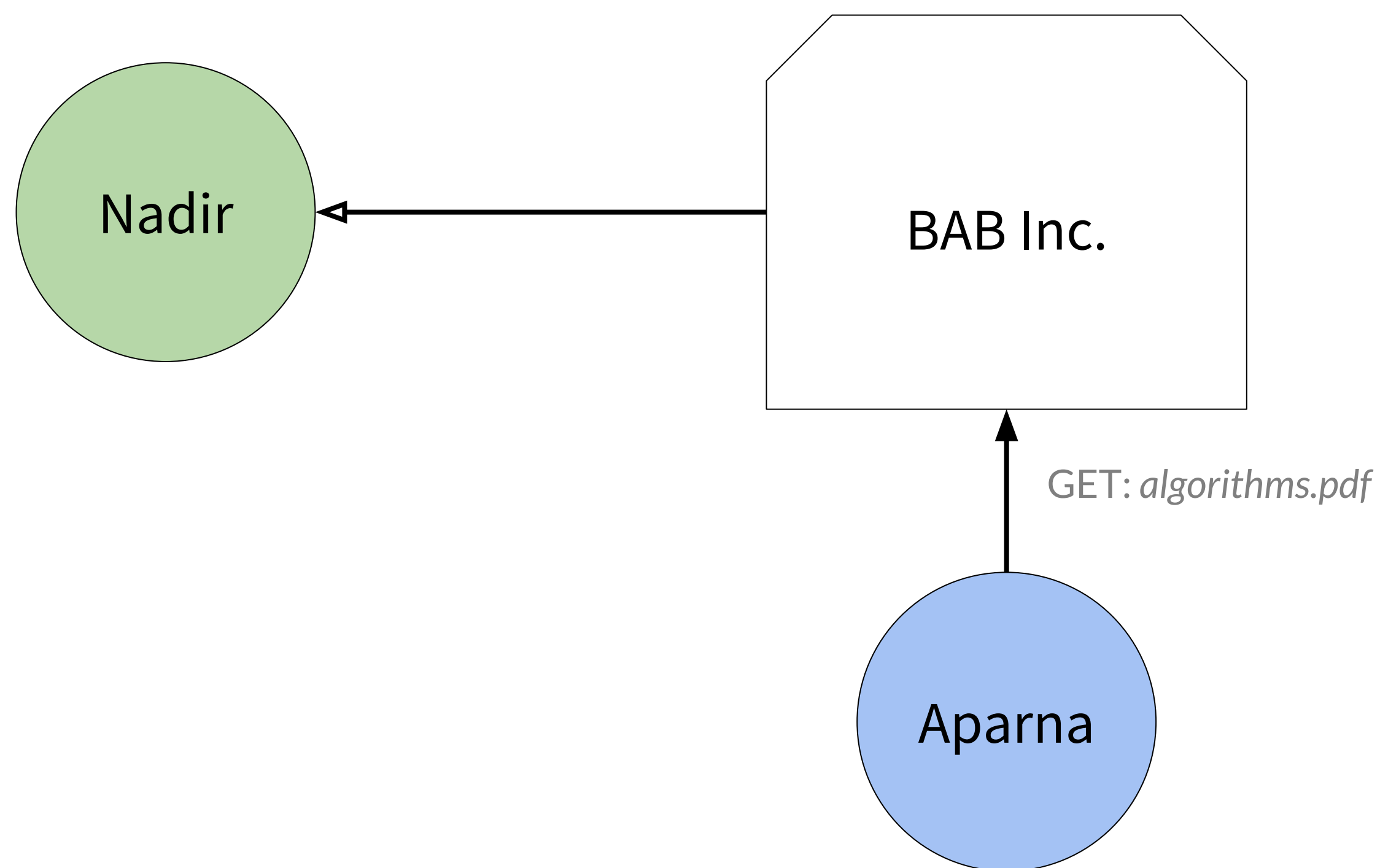
STARTING OFF SIMPLE





A CENTRALIZED EXAMPLE

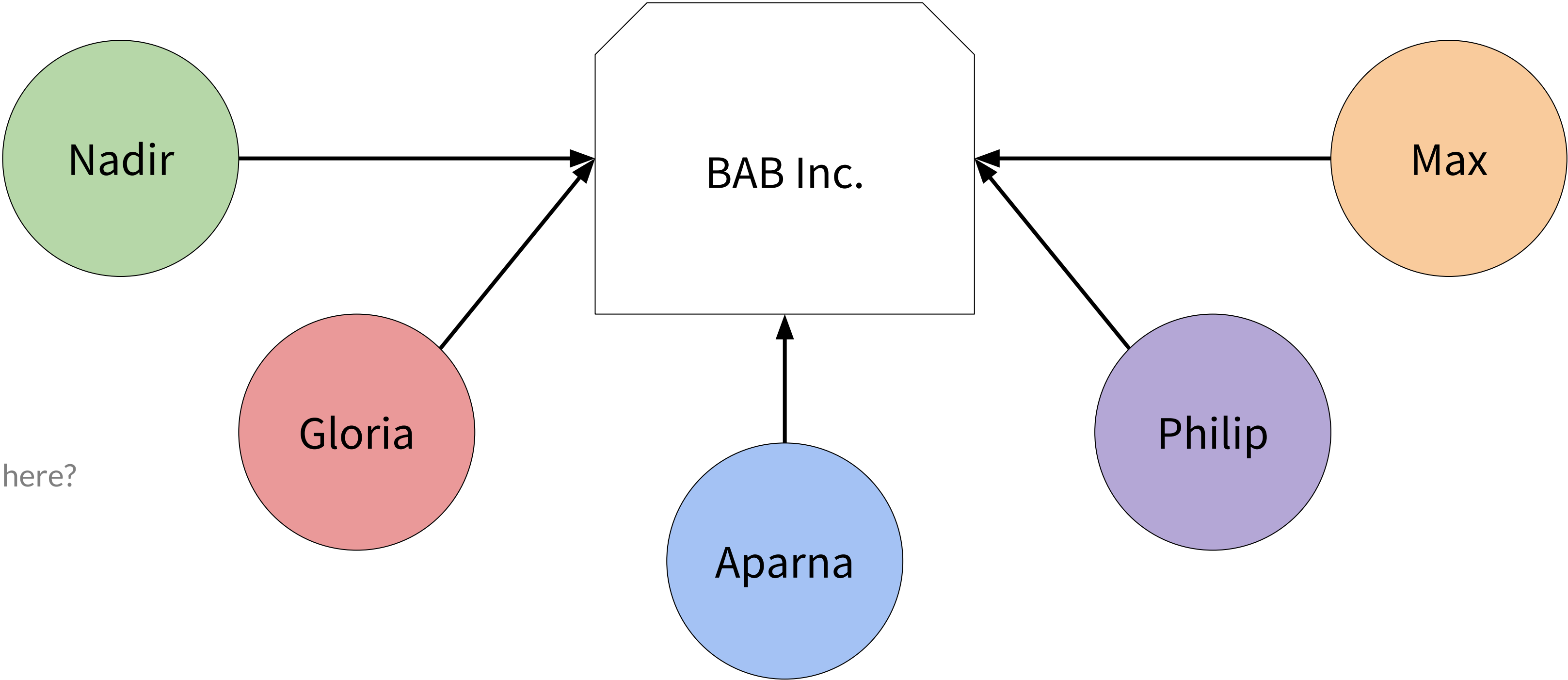
STARTING OFF SIMPLE





A CENTRALIZED EXAMPLE

STARTING OFF SIMPLE

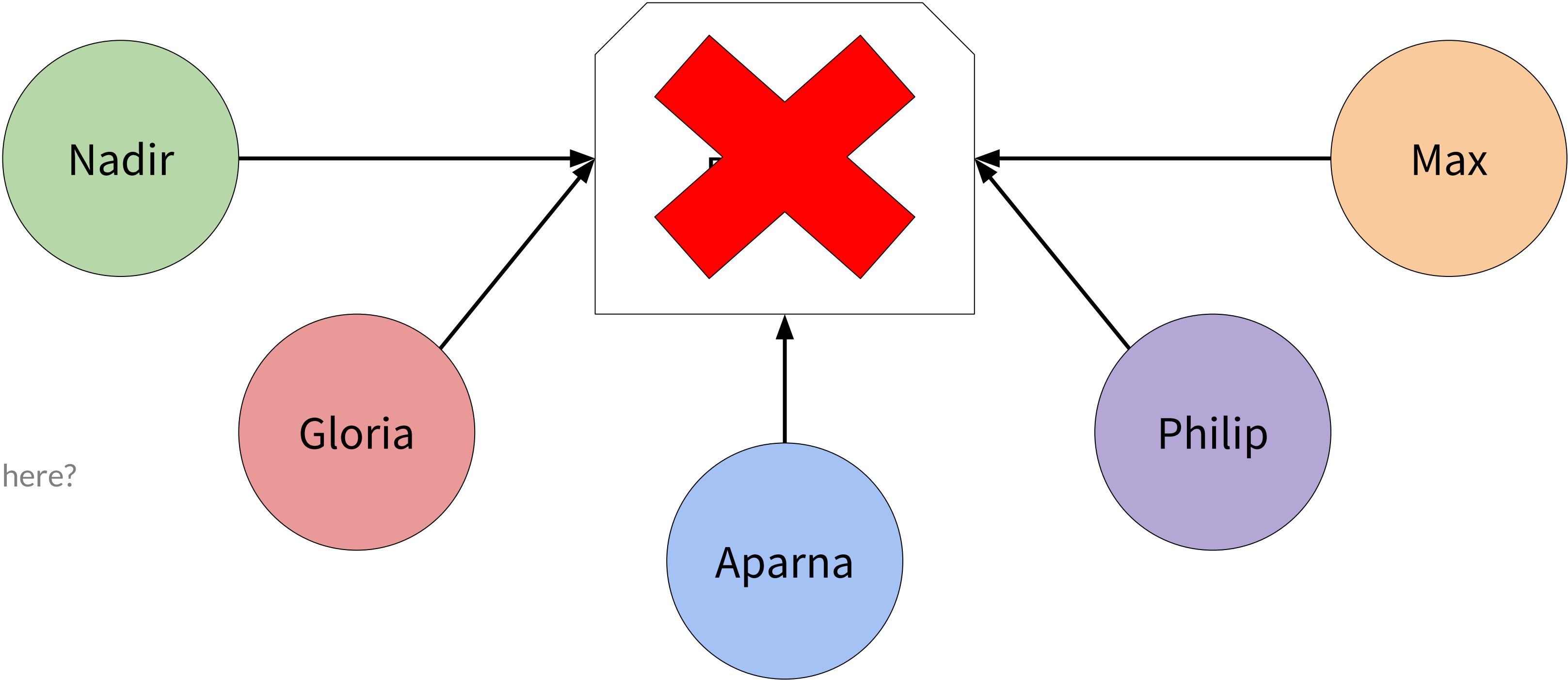


What are the problems here?

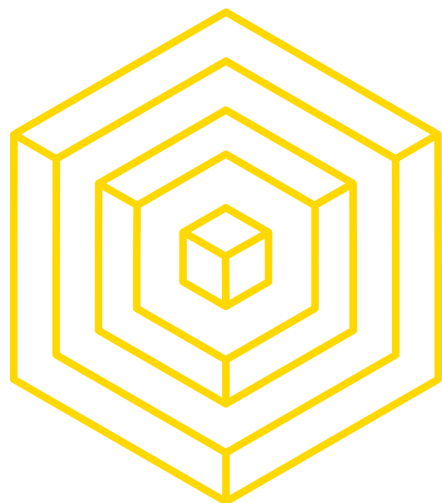


A CENTRALIZED EXAMPLE

STARTING OFF SIMPLE

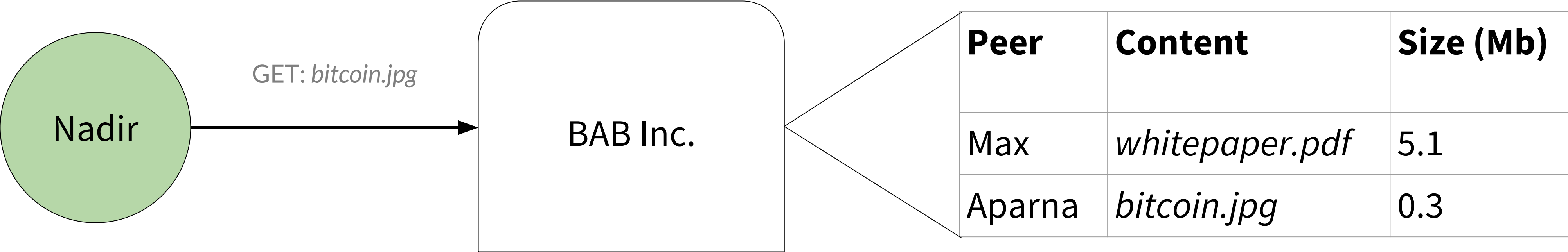


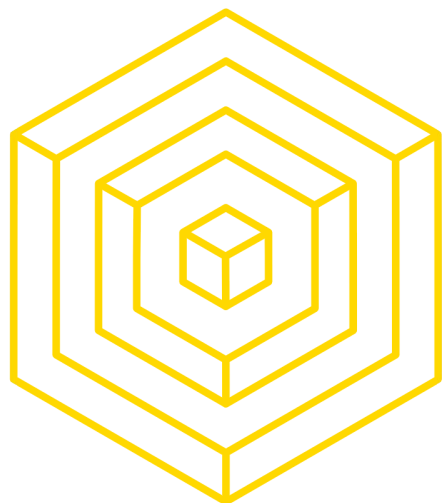
What are the problems here?



P2P EXAMPLE

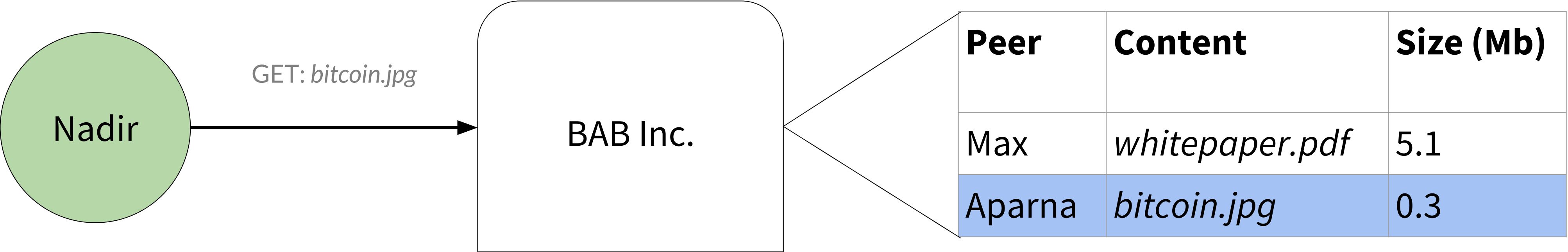
ANOTHER ONE

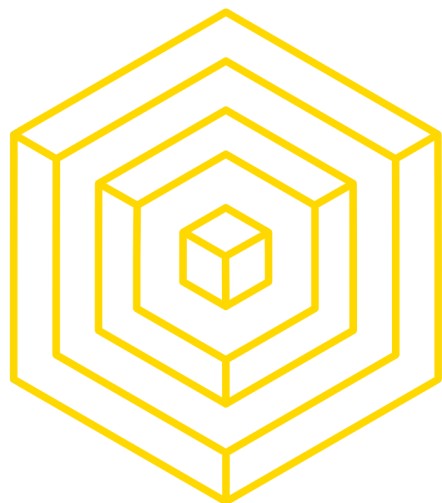




P2P EXAMPLE

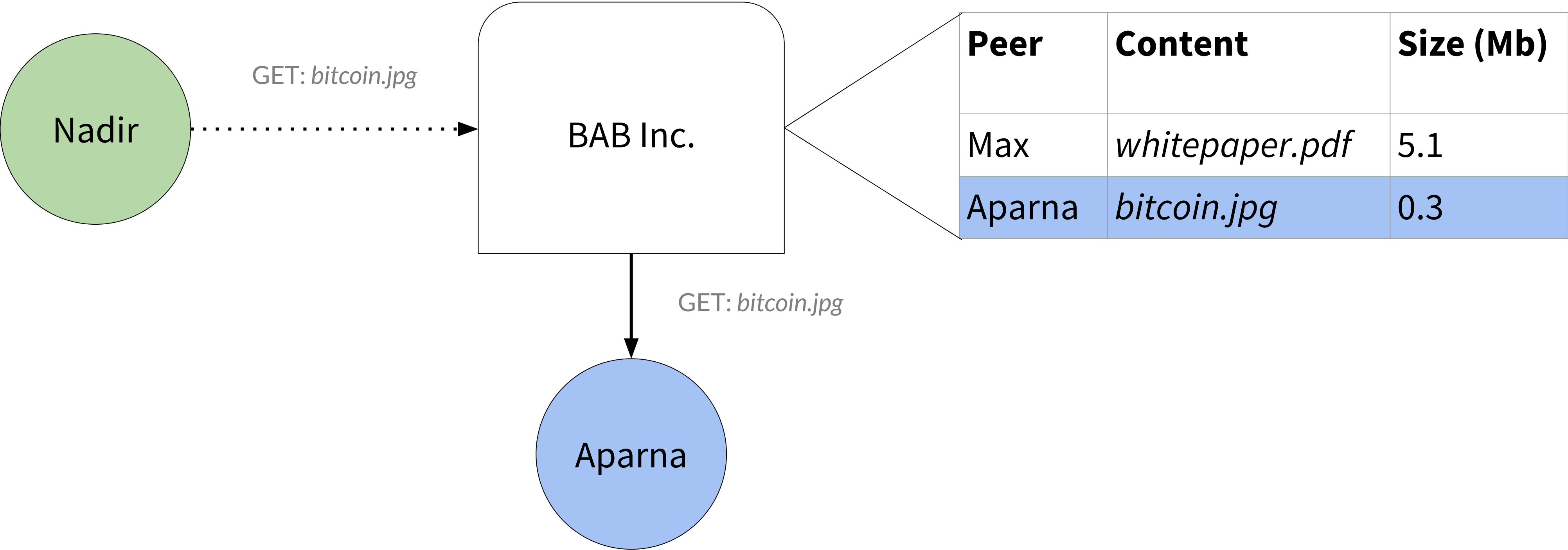
ANOTHER ONE

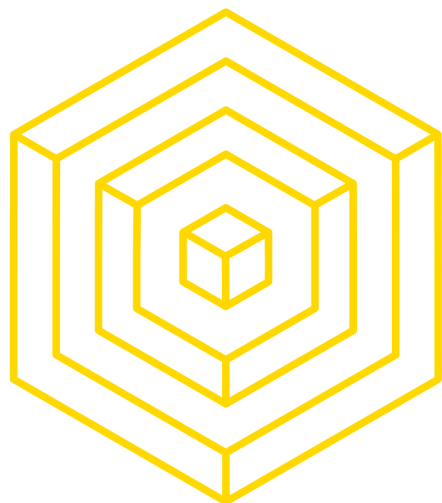




P2P EXAMPLE

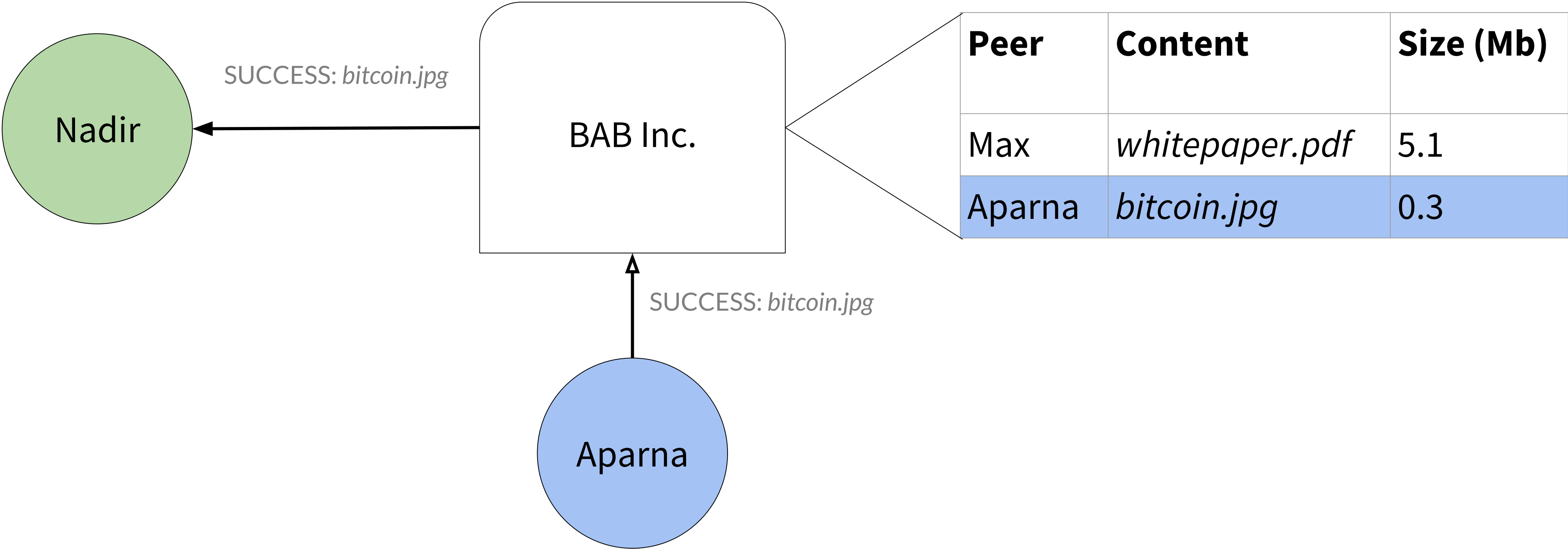
ANOTHER ONE





P2P EXAMPLE

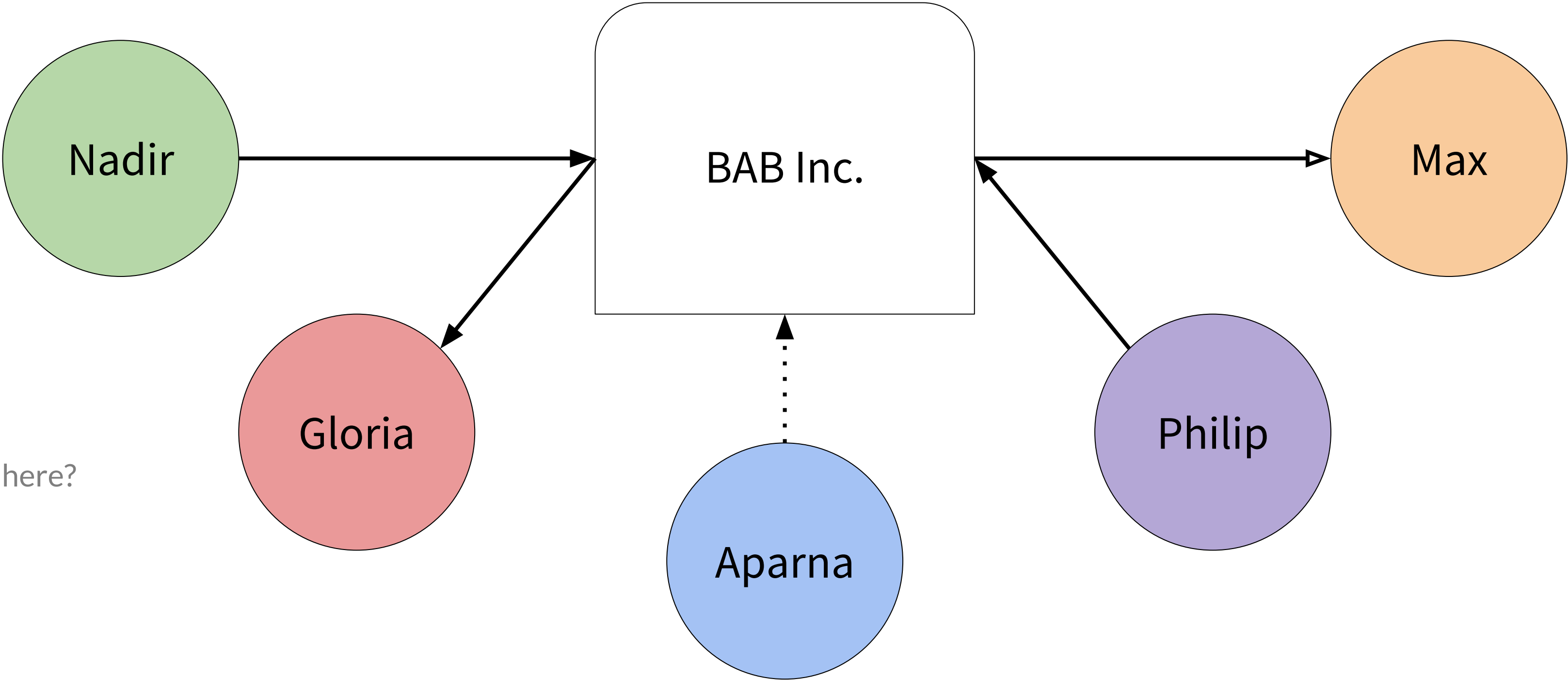
ANOTHER ONE





P2P EXAMPLE

STARTING OFF SIMPLE

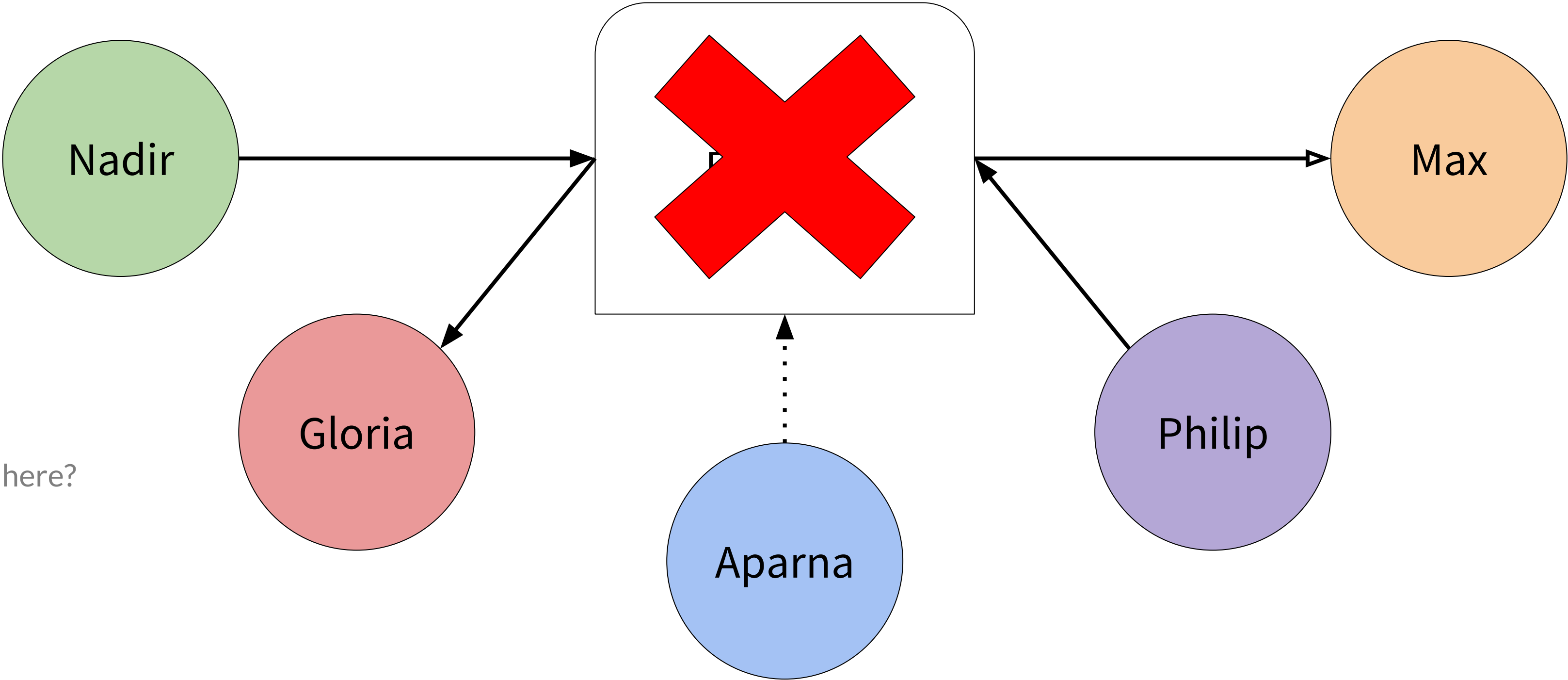


What are the problems here?



P2P EXAMPLE

STARTING OFF SIMPLE



1.2

BETTER P2P

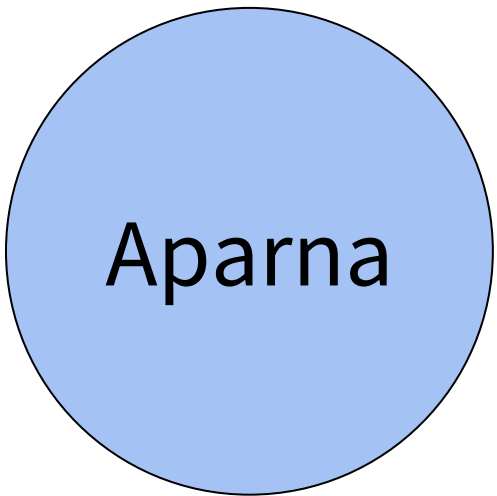
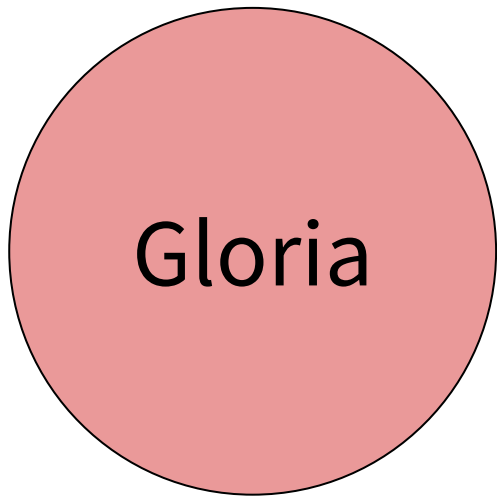
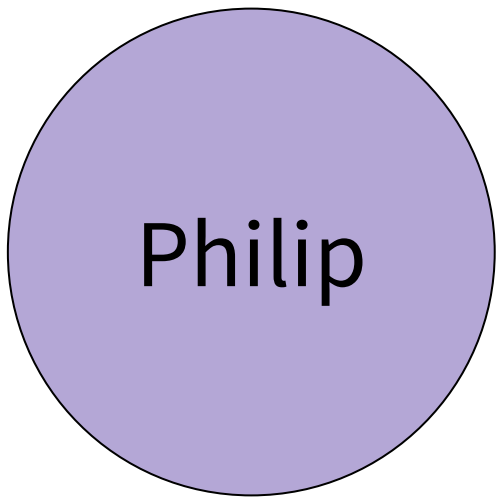
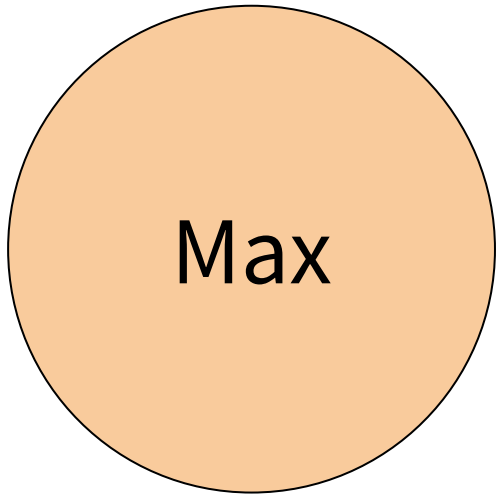
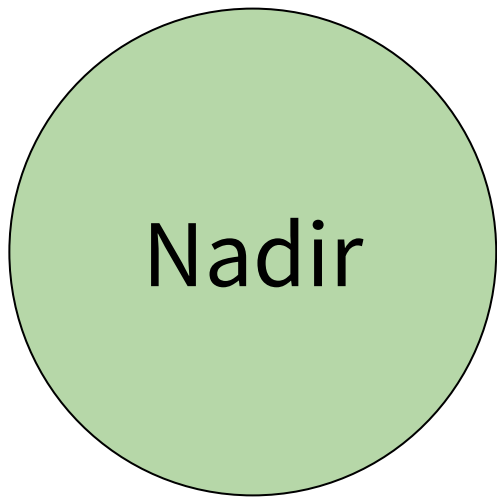


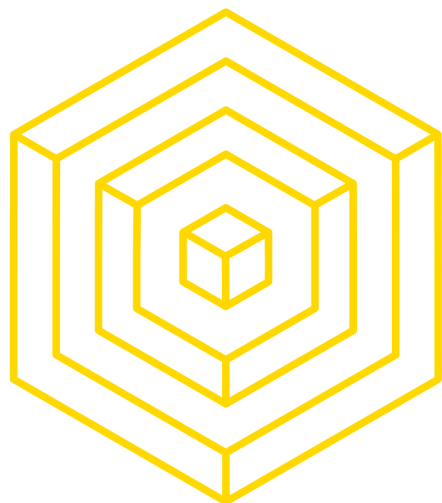
ALTERNATIVE P2P

FIRST ATTEMPT: LOOPING

Let's try looping through known peers

Peer	Address
Gloria	208.17.50.4
Aparna	208.17.50.5
Max	208.17.50.6
Philip	208.17.50.7



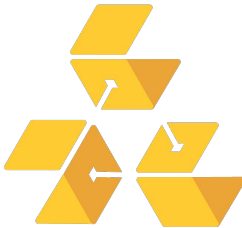
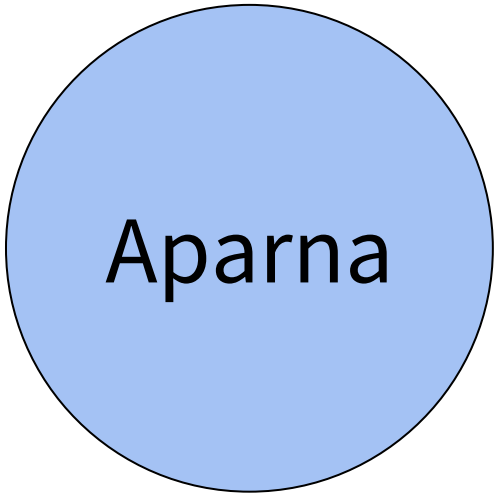
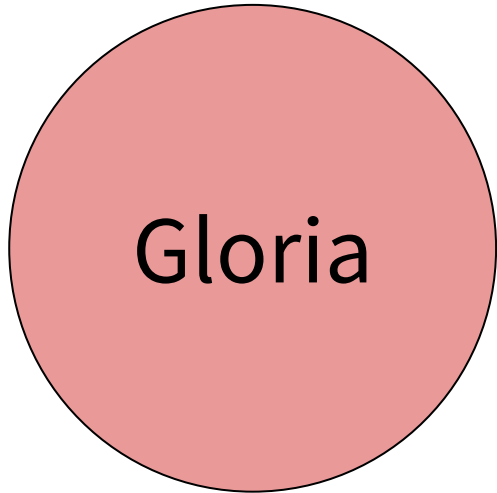
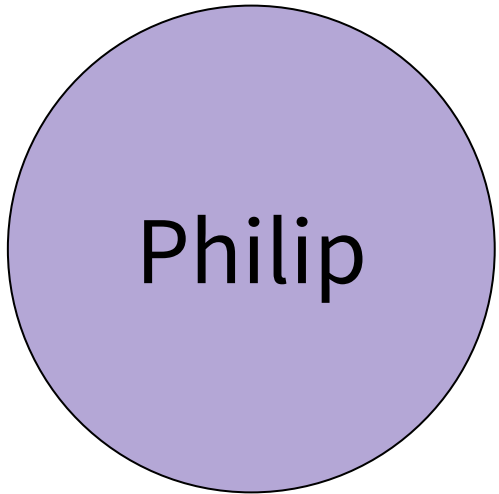
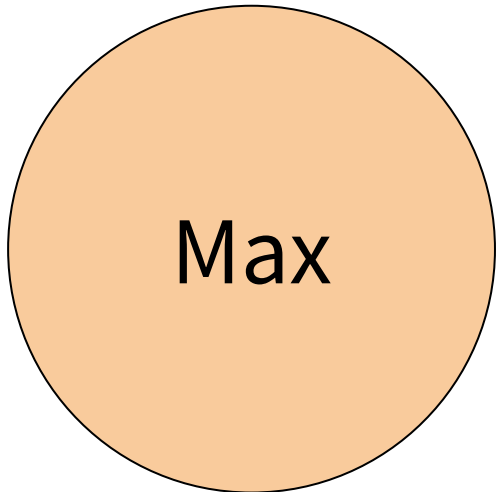
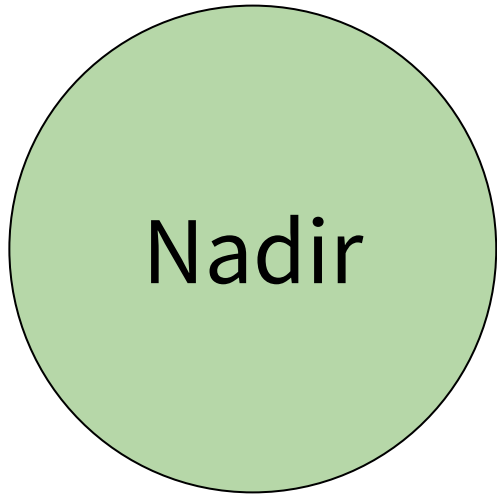


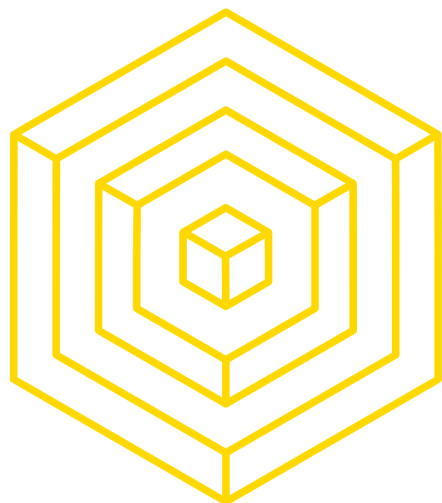
ALTERNATIVE P2P

FIRST ATTEMPT: LOOPING

QUERY: *bitcoin.jpg* ??????

Peer	Address
Gloria	208.17.50.4
Aparna	208.17.50.5
Max	208.17.50.6
Philip	208.17.50.7

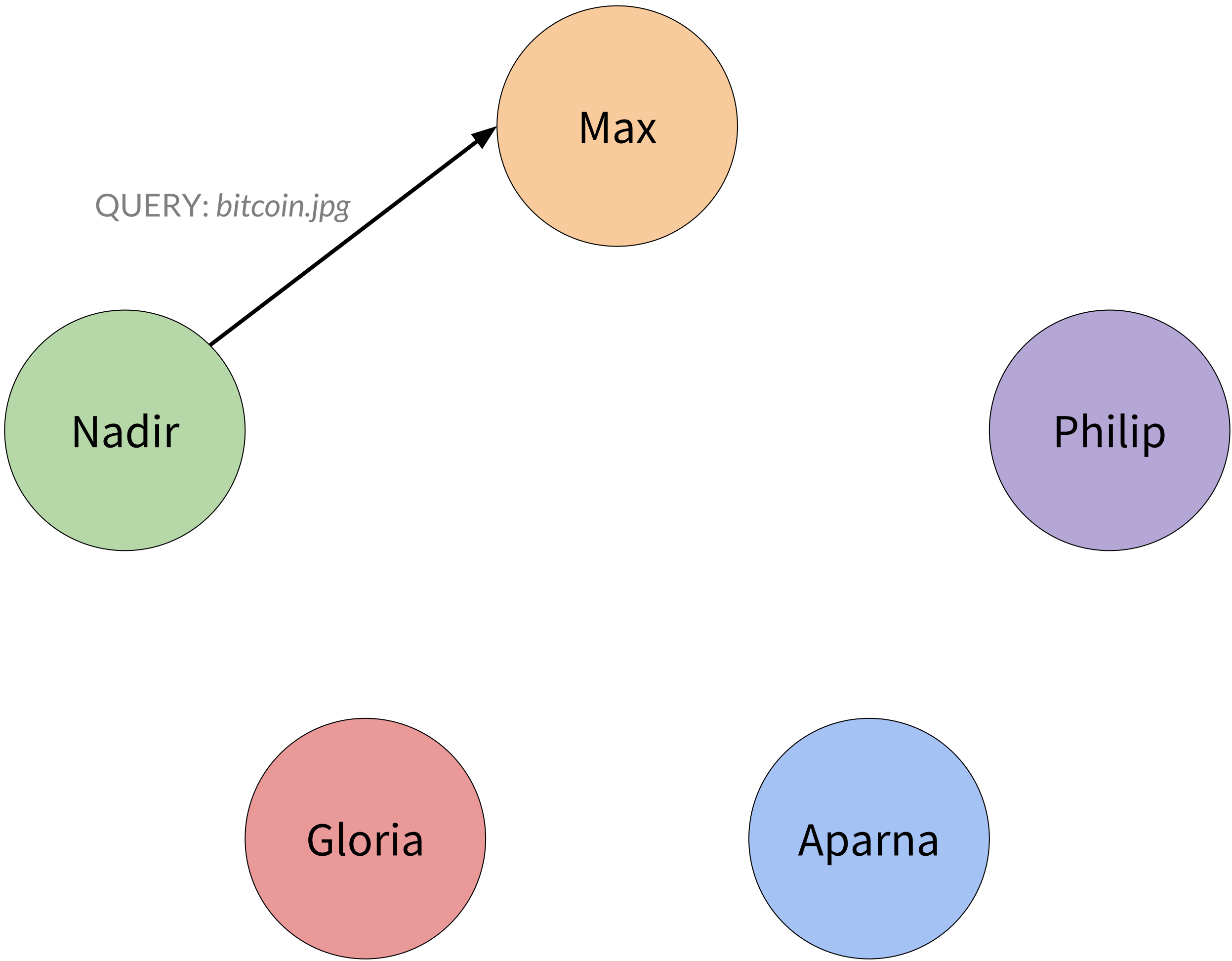




ALTERNATIVE P2P

FIRST ATTEMPT: LOOPING

Peer	Address
Gloria	208.17.50.4
Aparna	208.17.50.5
Max	208.17.50.6
Philip	208.17.50.7

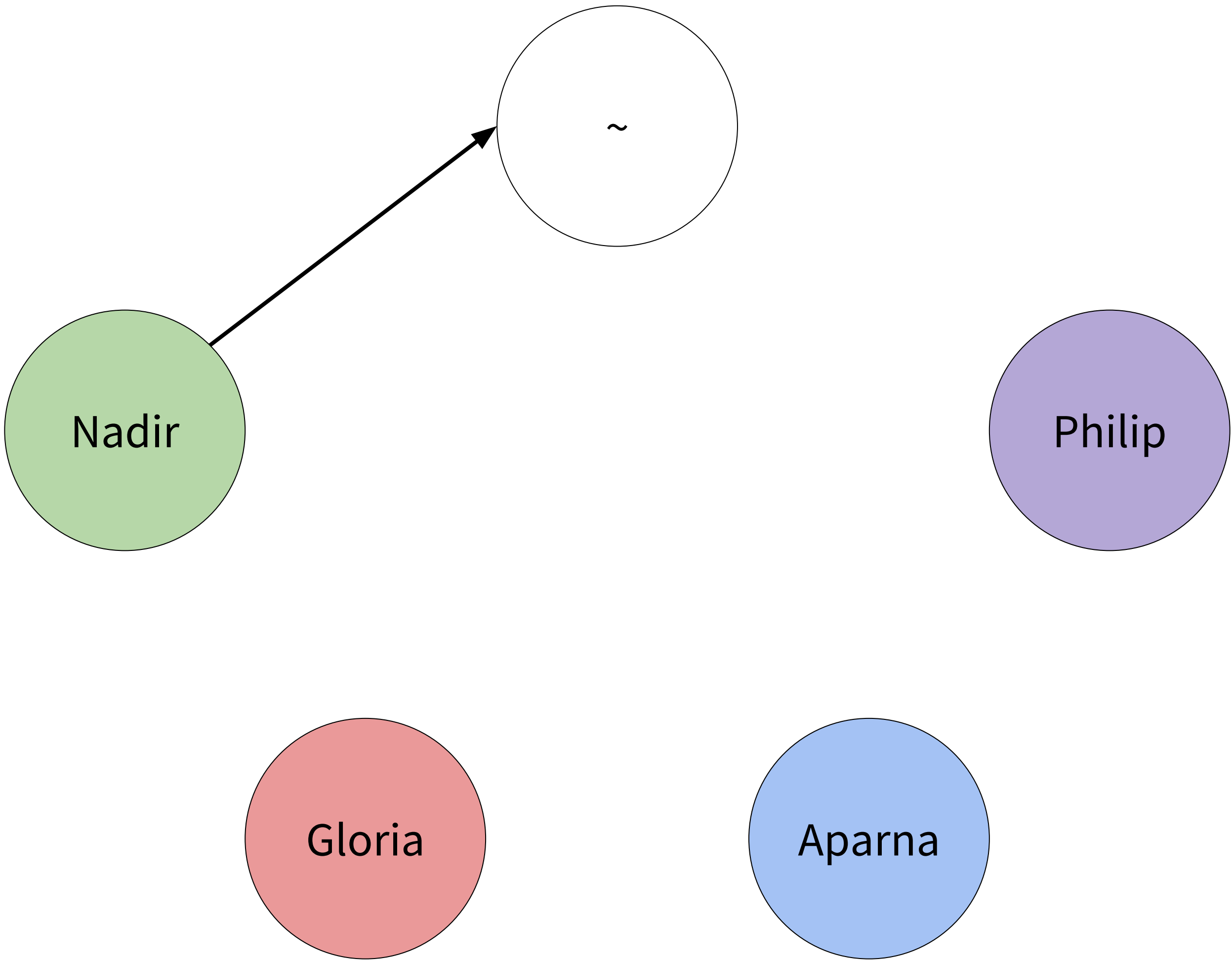




ALTERNATIVE P2P

FIRST ATTEMPT: LOOPING

Peer	Address
Gloria	208.17.50.4
Aparna	208.17.50.5
Max	208.17.50.6
Philip	208.17.50.7

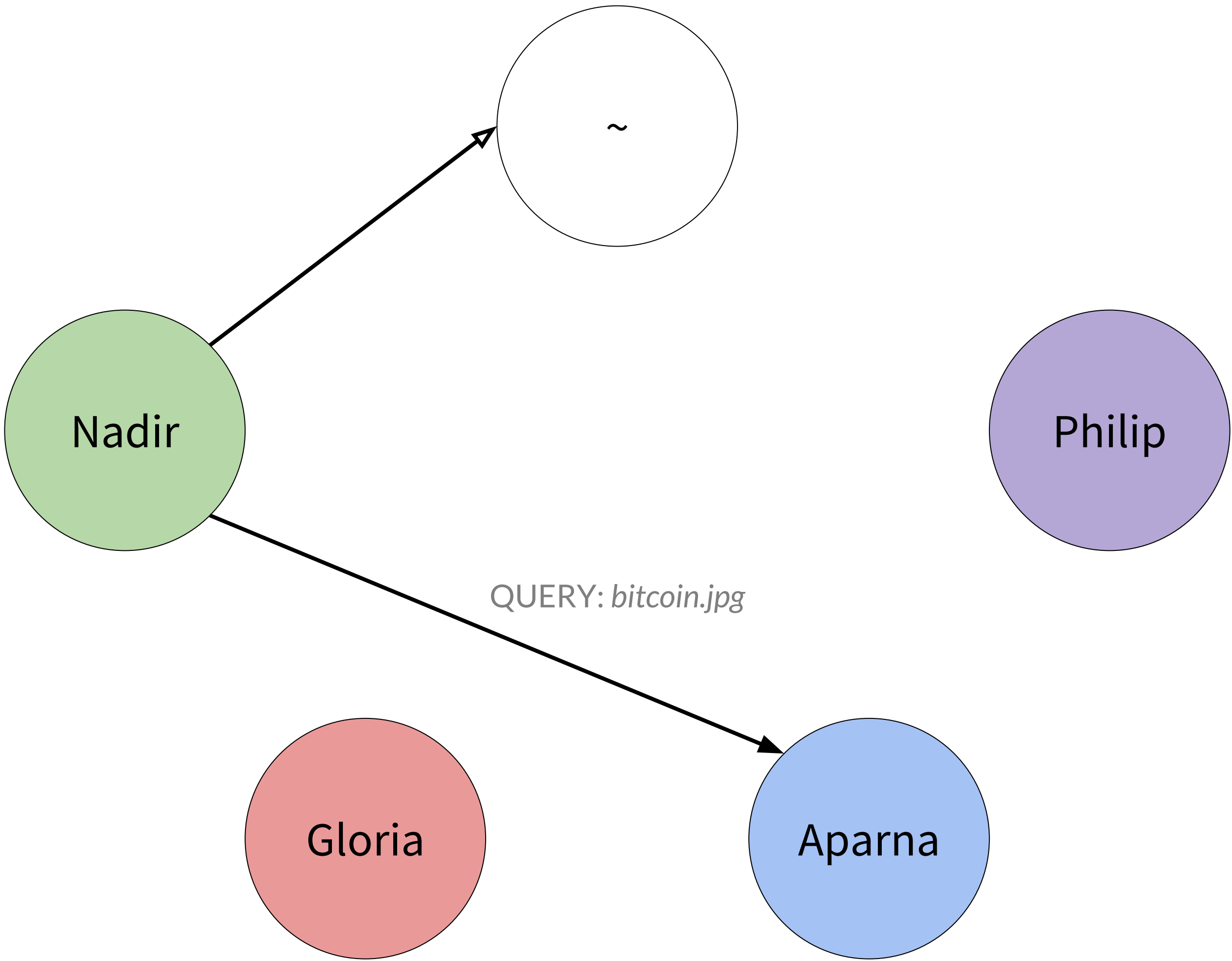


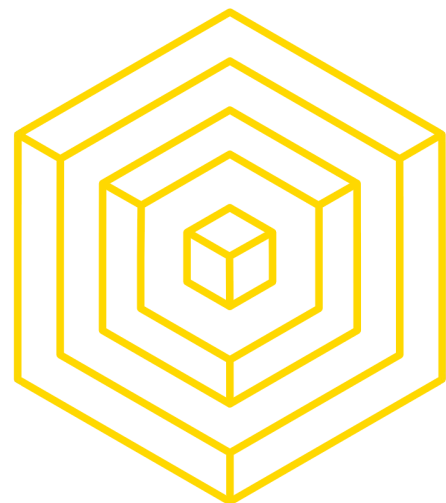


ALTERNATIVE P2P

FIRST ATTEMPT: LOOPING

Peer	Address
Gloria	208.17.50.4
Aparna	208.17.50.5
Max	208.17.50.6
Philip	208.17.50.7

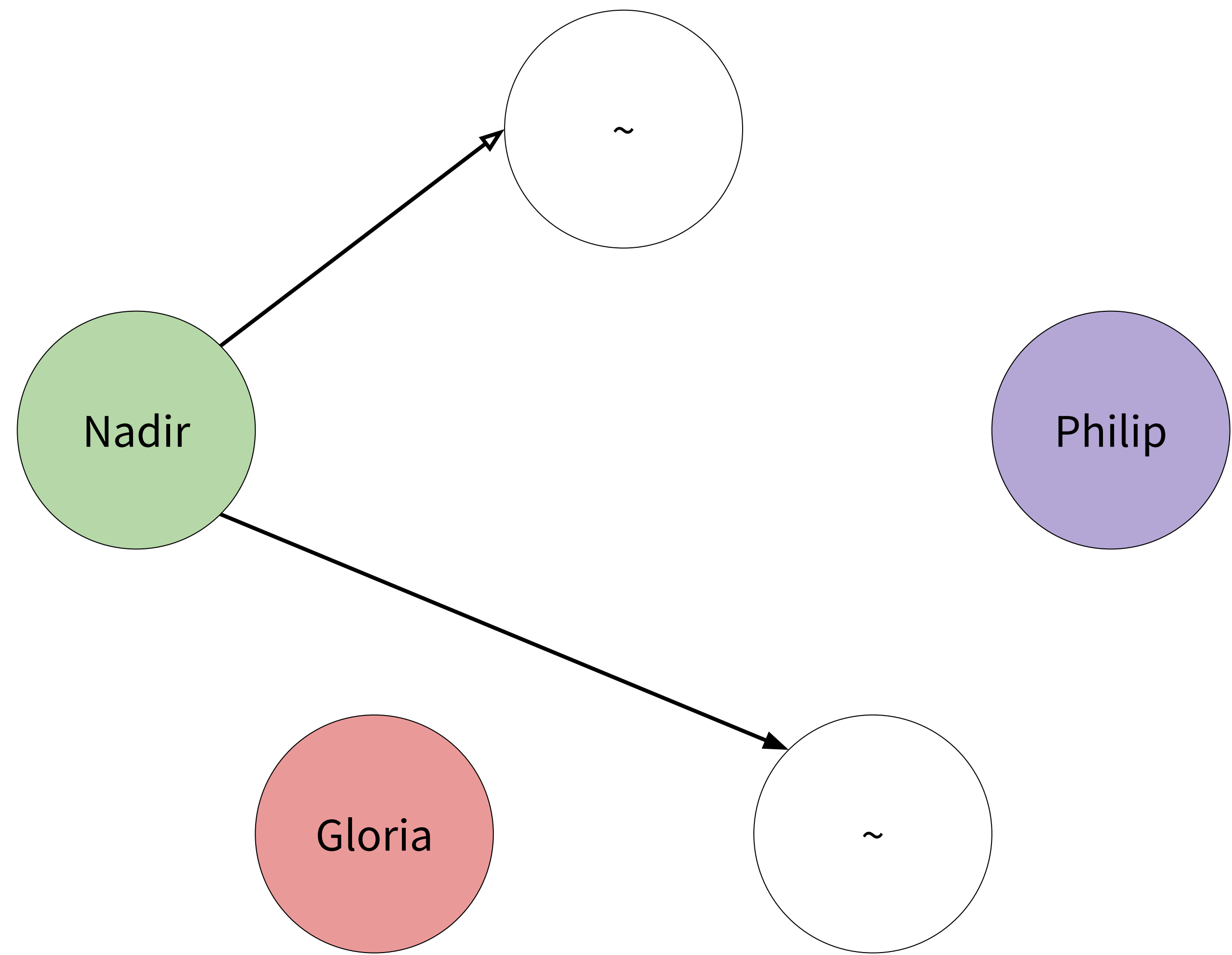




ALTERNATIVE P2P

FIRST ATTEMPT: LOOPING

Peer	Address
Gloria	208.17.50.4
Aparna	208.17.50.5
Max	208.17.50.6
Philip	208.17.50.7



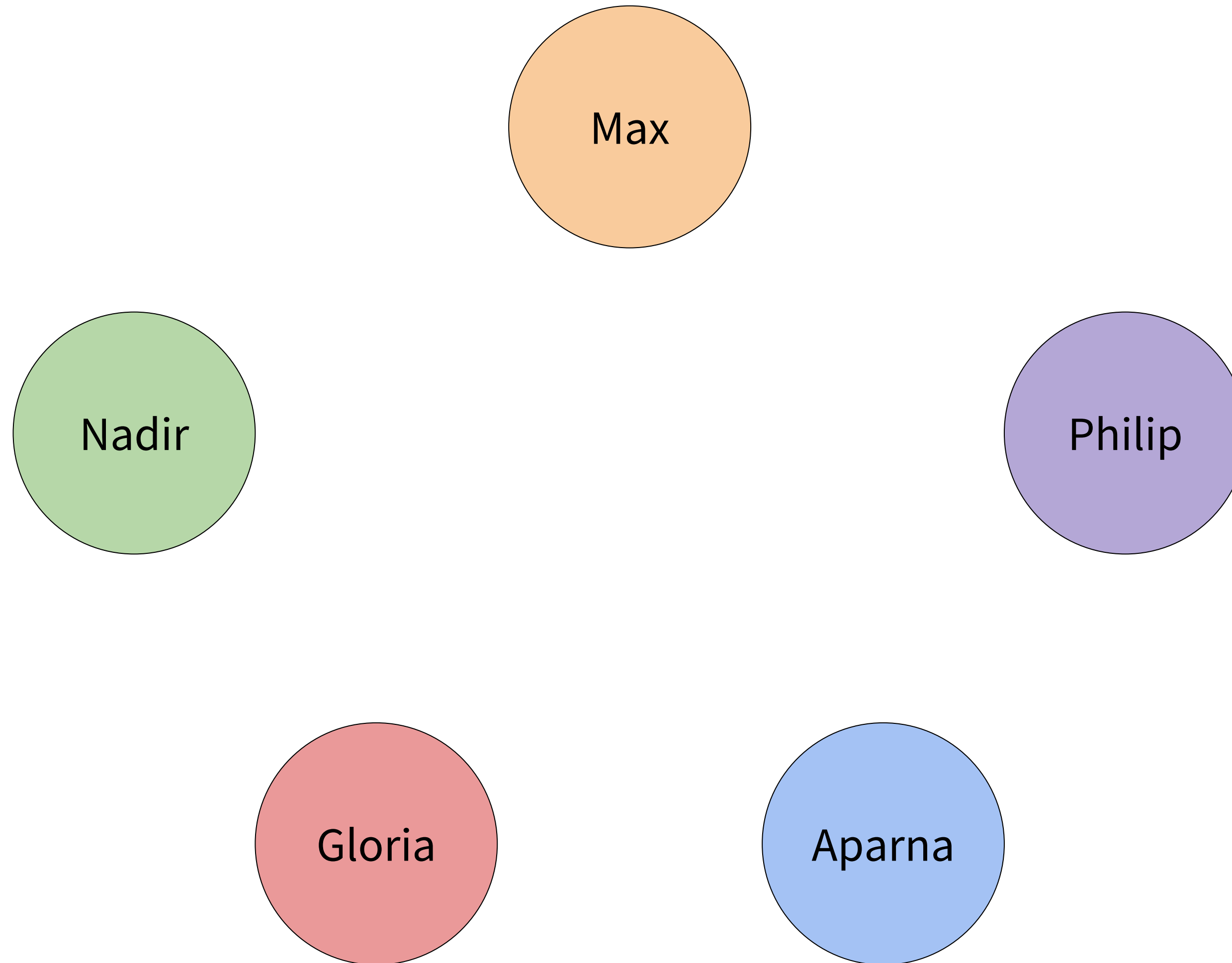


ALTERNATIVE P2P

SECOND ATTEMPT: MST

Let's try establishing a
routing table
(minimum spanning
tree)

- Logarithmic
overhead with
network growth!



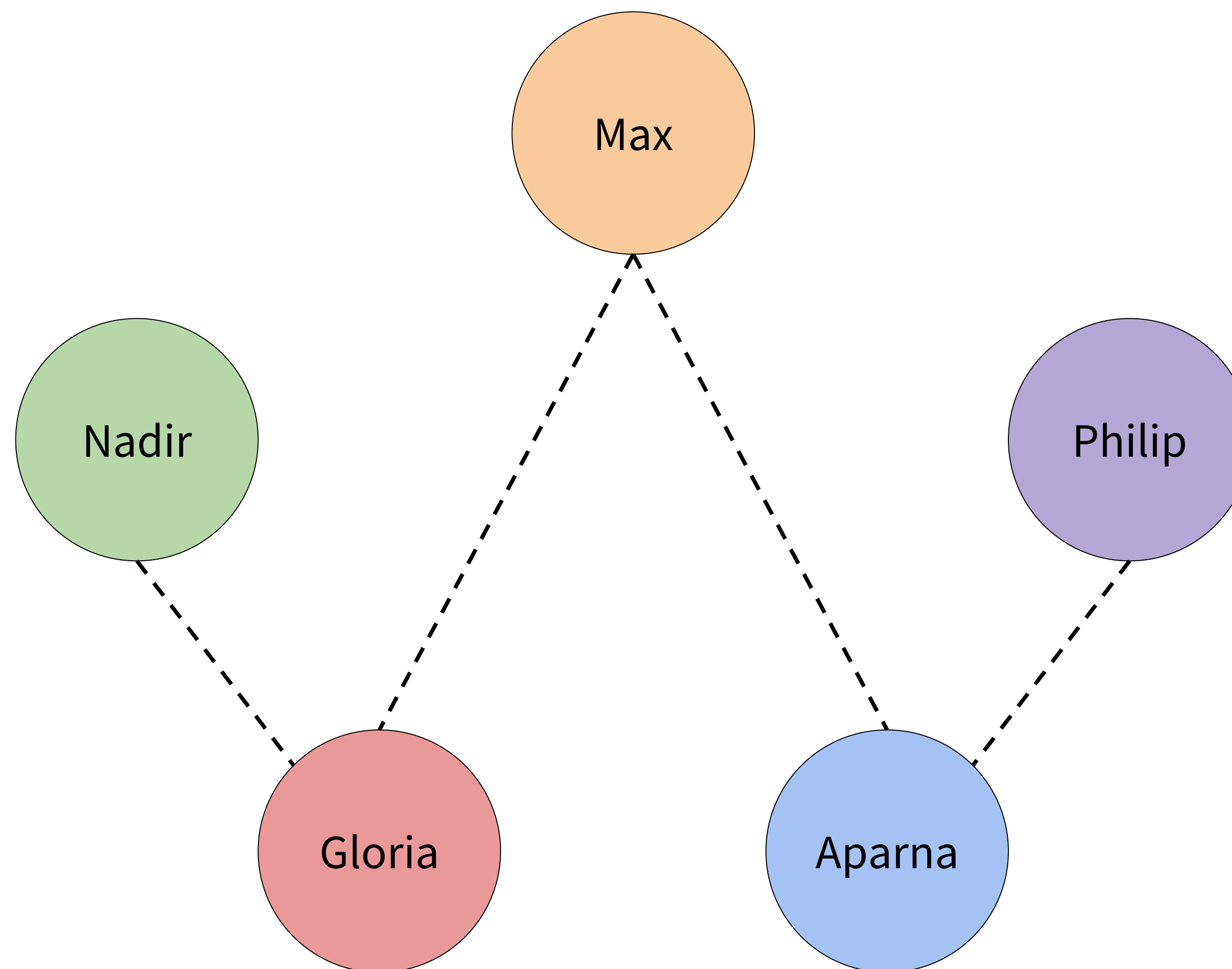


ALTERNATIVE P2P

SECOND ATTEMPT: MST

Let's try establishing a
routing table
(minimum spanning
tree)

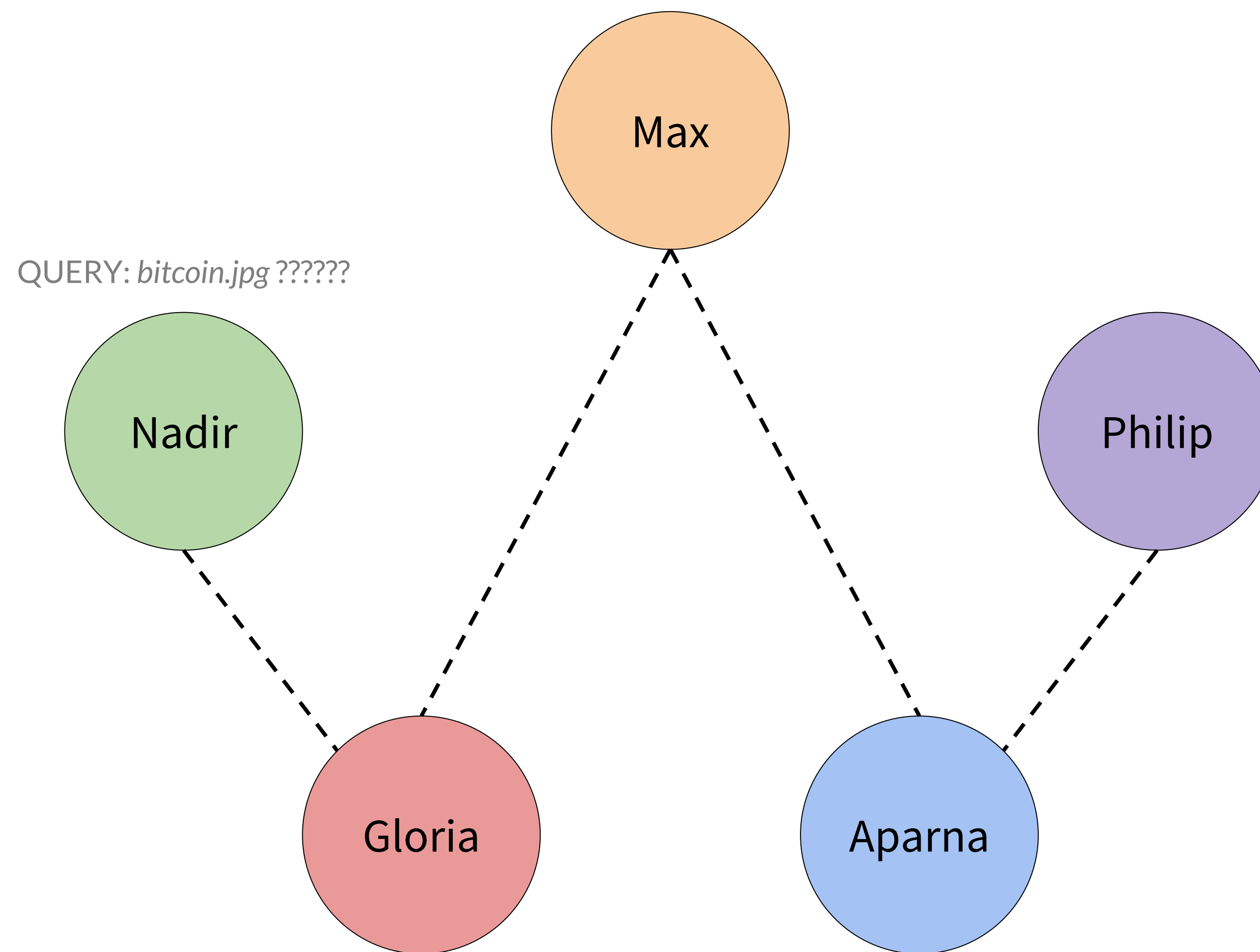
- Logarithmic
overhead with
network growth!





ALTERNATIVE P2P

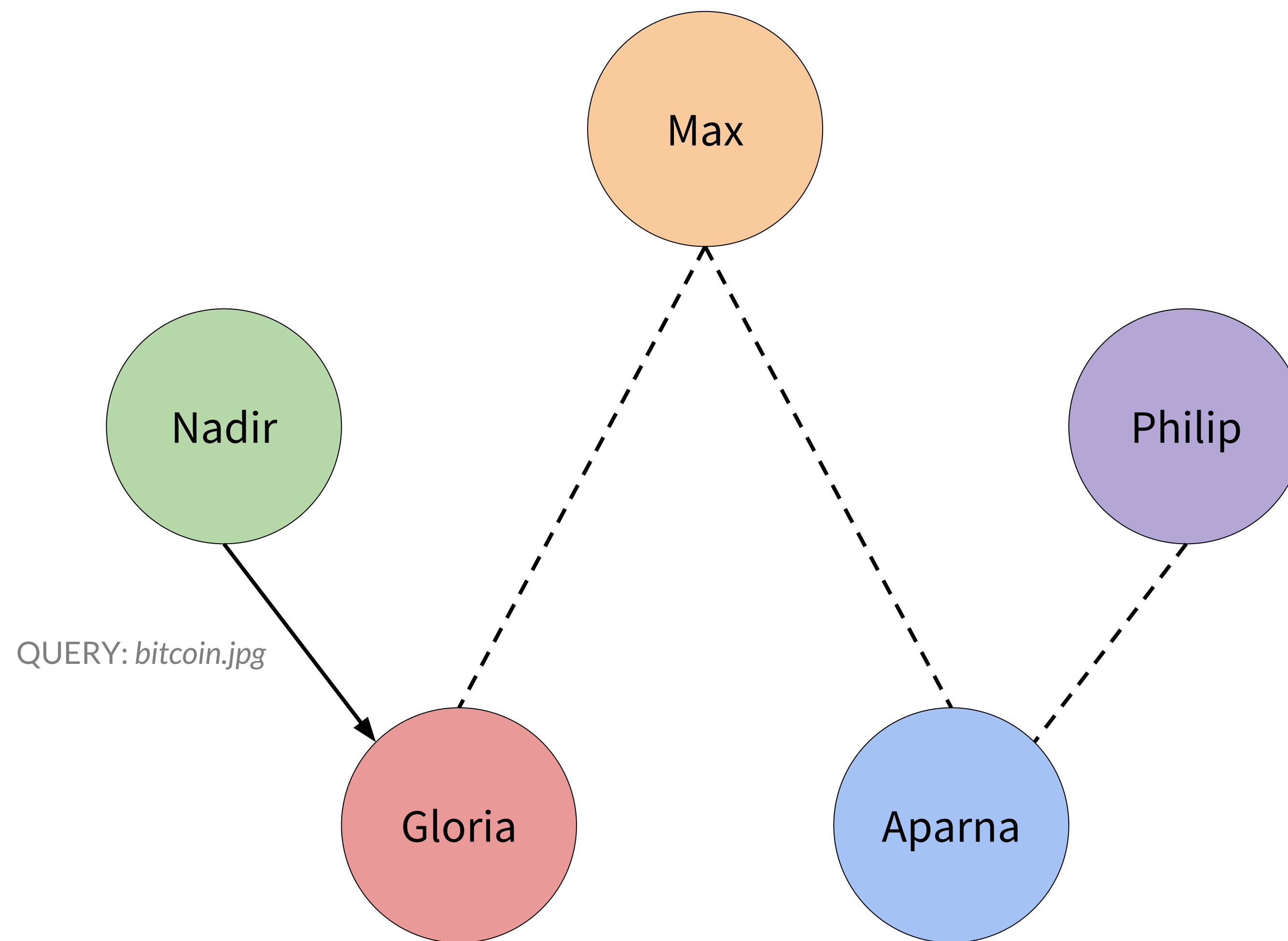
SECOND ATTEMPT: MST





ALTERNATIVE P2P

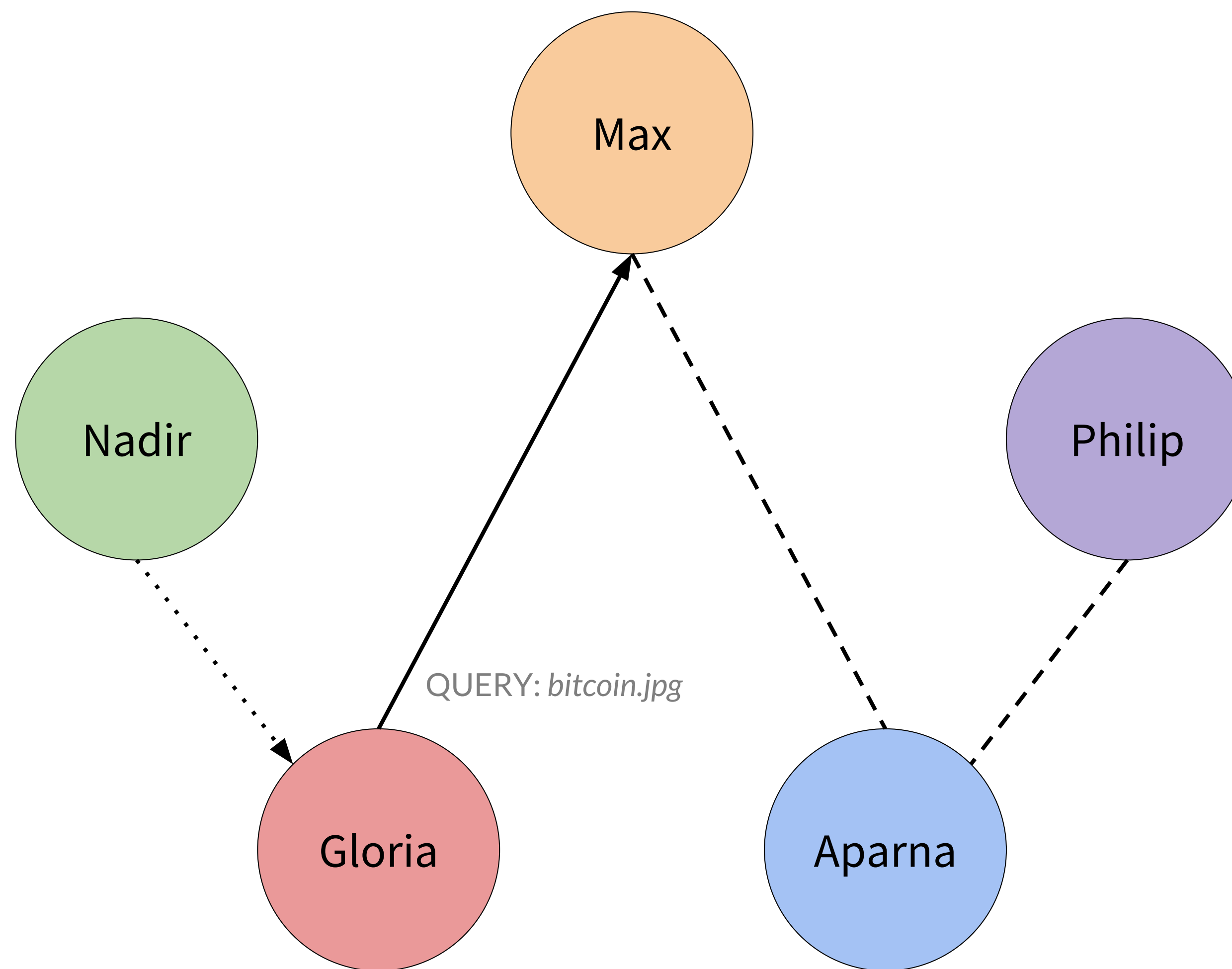
SECOND ATTEMPT: MST





ALTERNATIVE P2P

SECOND ATTEMPT: MST



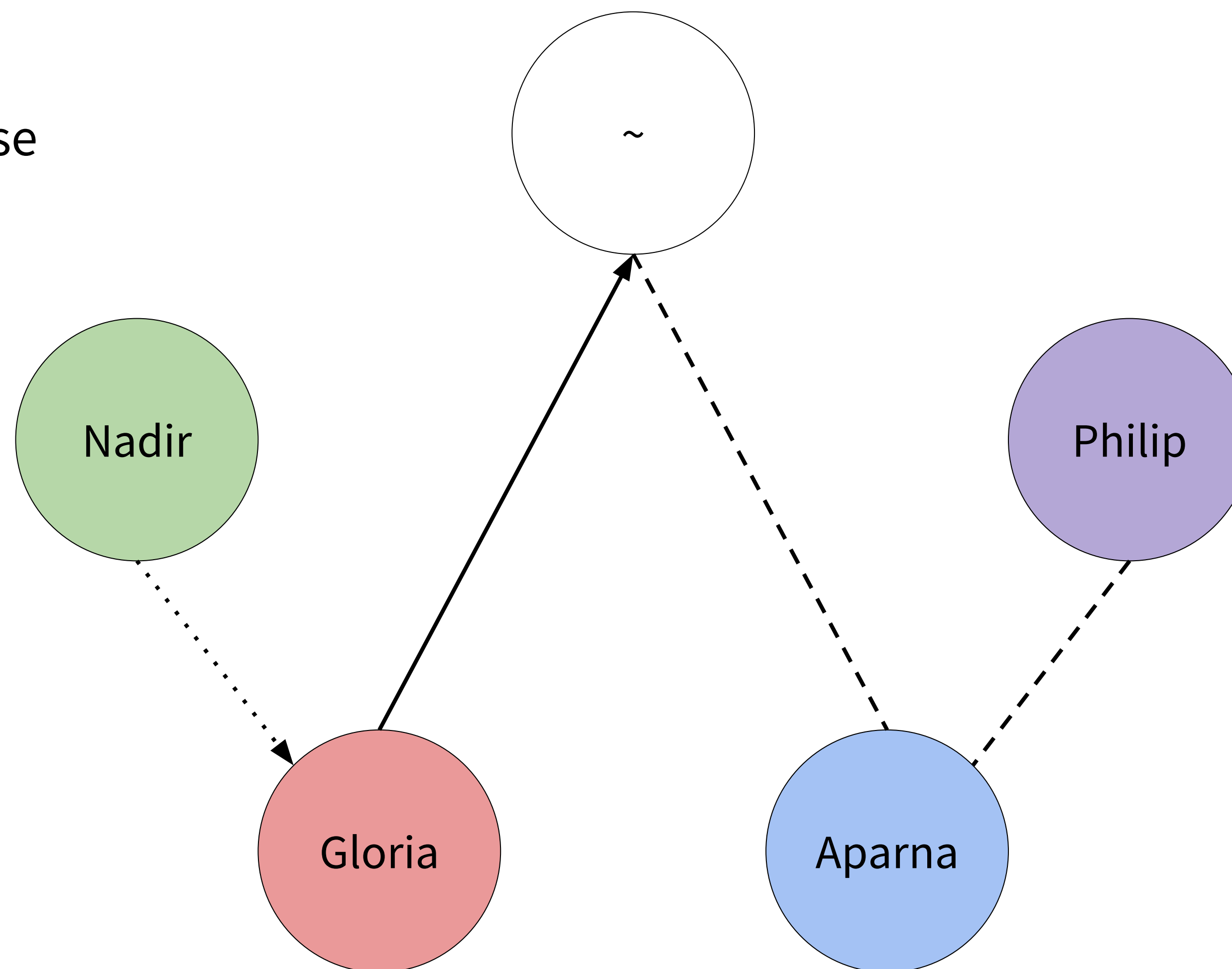


ALTERNATIVE P2P

SECOND ATTEMPT: MST

Scenario 1: No response

- Need to reroute





ALTERNATIVE P2P

SECOND ATTEMPT: MST

Scenario 2: Lossy channel

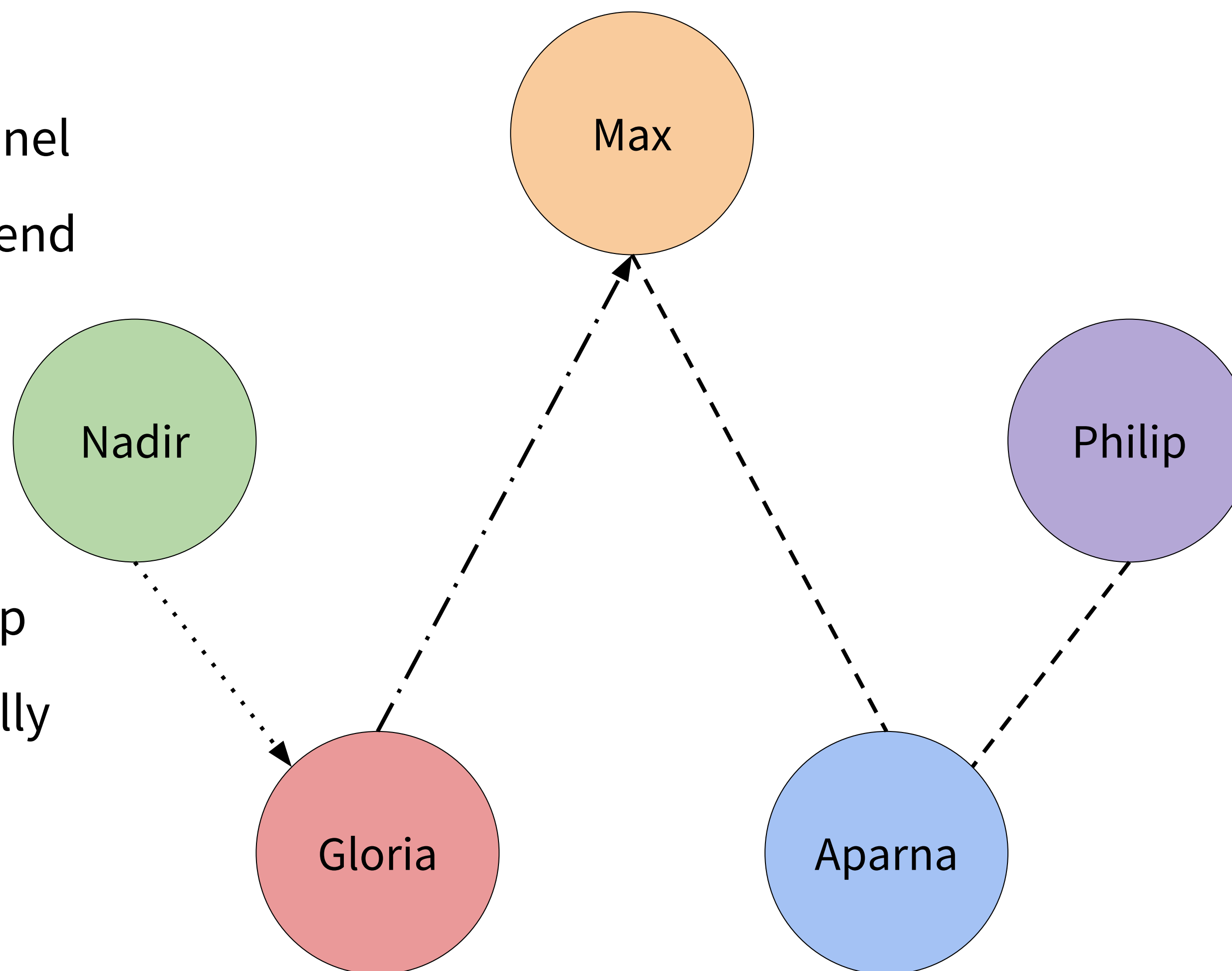
- Need to know that send failed
- Need to resend packets
 - No overall speedup

This solution is generally fine for

▼ fixed-infrastructure

▼ networks

AUTHOR: NICK ZOGHB



1.3

GOSSIP PROTOCOL

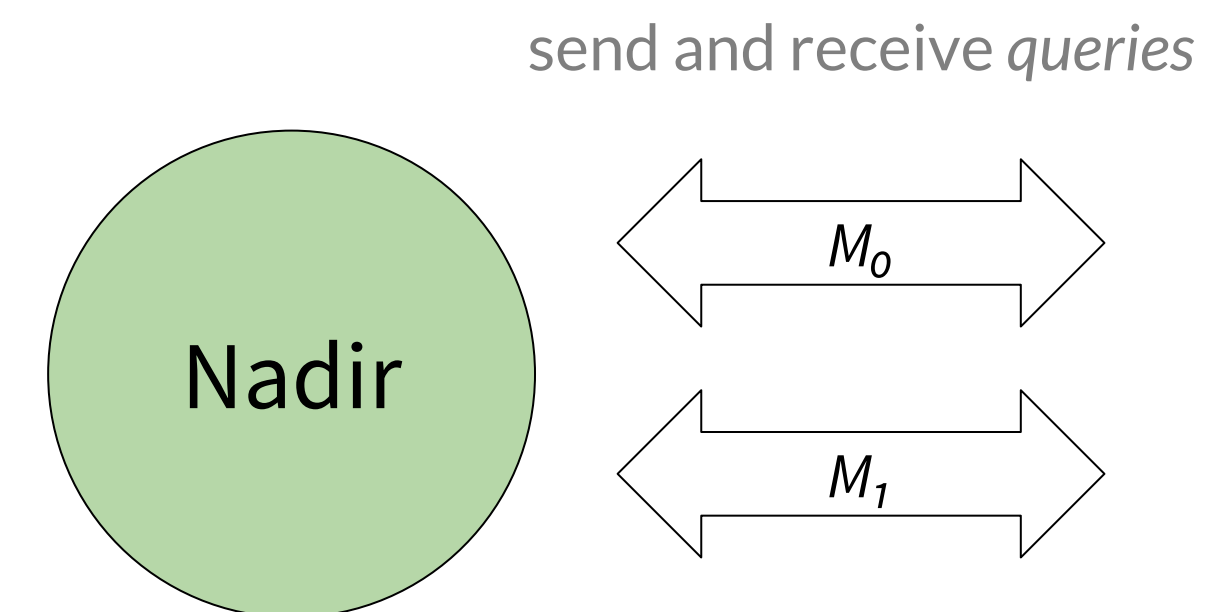


CASE STUDY: GNUTELLA

DEFINING FORMALISMS

Distributed systems are made up of a collection of entities that are:

- **Autonomous/Programmable**
- **Asynchronous**
- Failure-prone ✓
- Communicate through an unreliable medium ✓



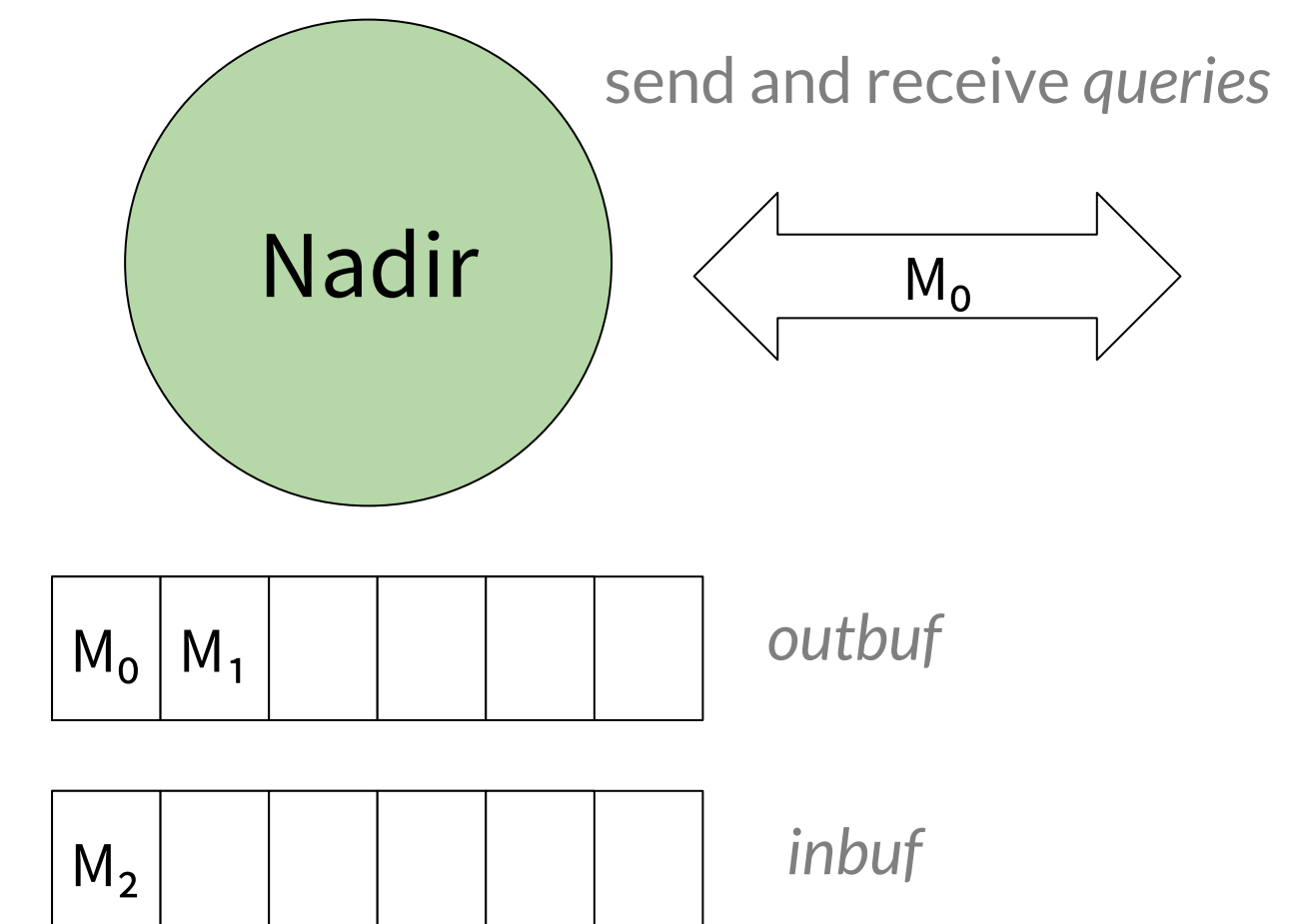


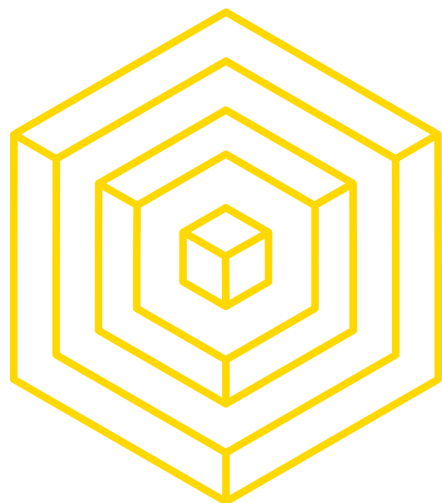
CASE STUDY: GNUTELLA

DEFINING FORMALISMS

Distributed systems are made up of a collection of entities that are:

- **Autonomous/Programmable**
- **Asynchronous**
- Failure-prone ✓
- Communicate through an unreliable medium ✓





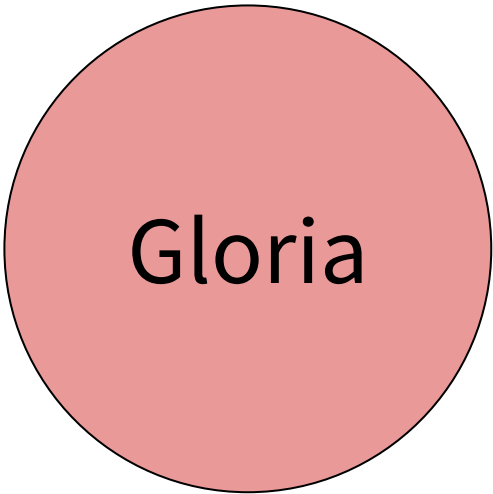
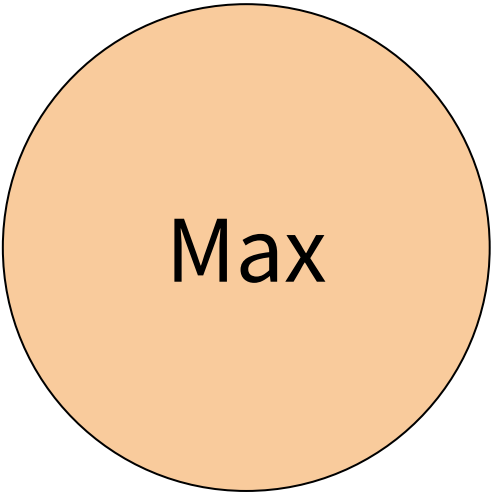
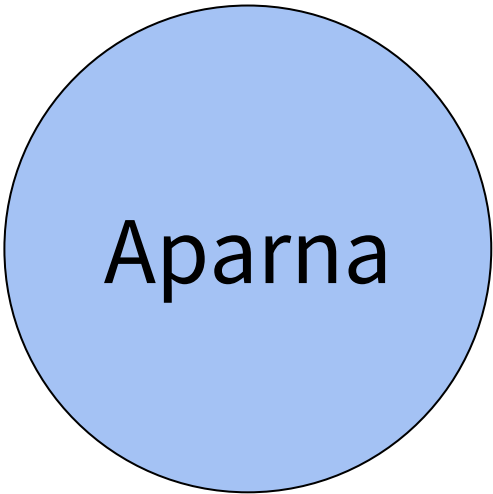
CASE STUDY: GNUTELLA

DEFINING FORMALISMS: PING/PONG & NETWORK DISCOVERY

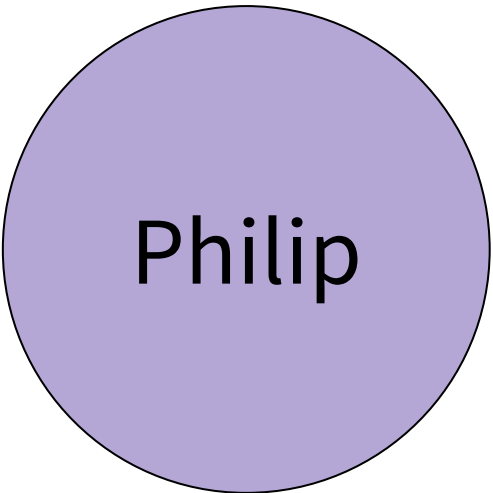
Peer	Gnutella Web Cached Address	Hops
Gloria	208.17.50.4	1
Aparna	208.17.50.5	1



Peers: 2



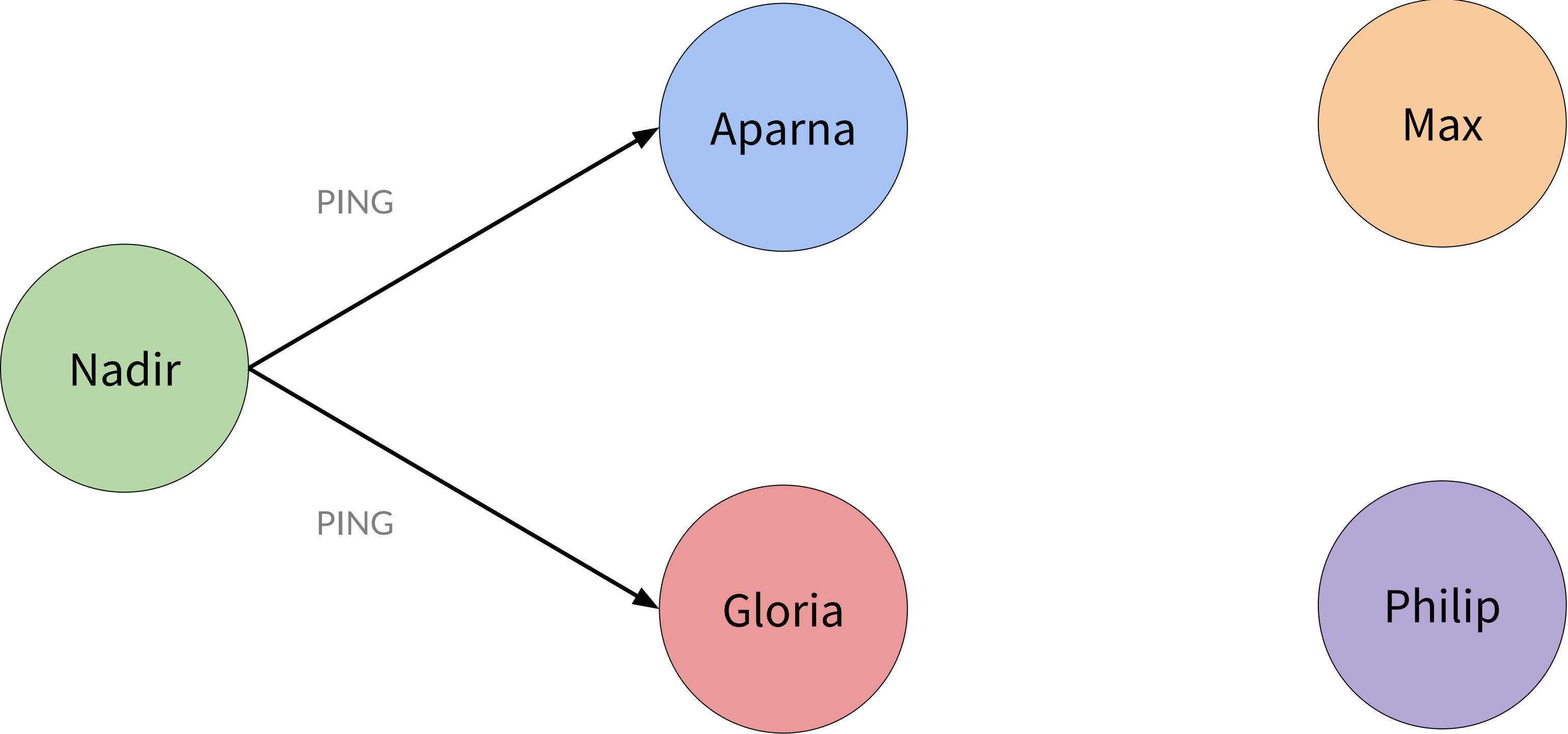
Peers: 0





CASE STUDY: GNUTELLA

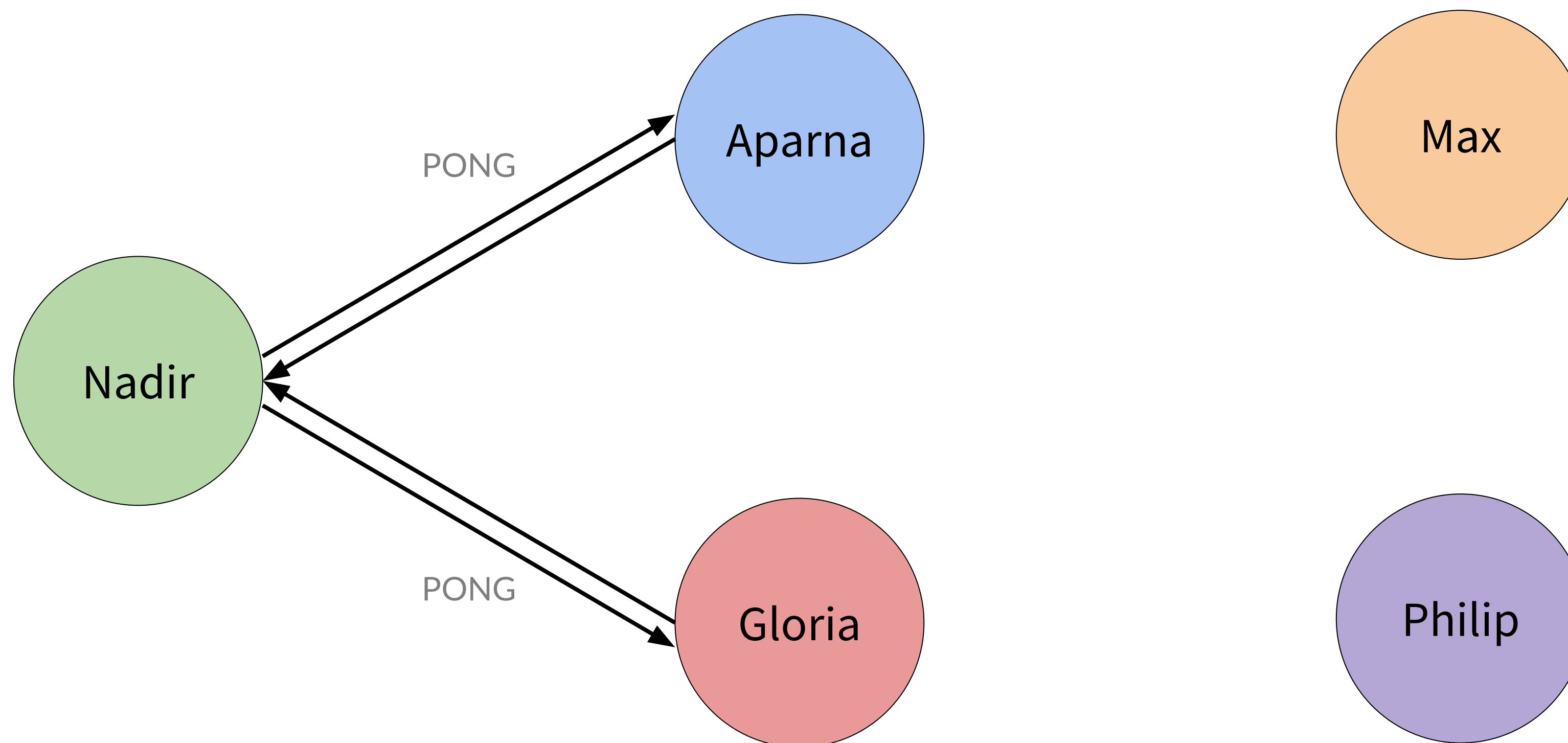
DEFINING FORMALISMS: PING/PONG & NETWORK DISCOVERY





CASE STUDY: GNUTELLA

DEFINING FORMALISMS: PING/PONG & NETWORK DISCOVERY





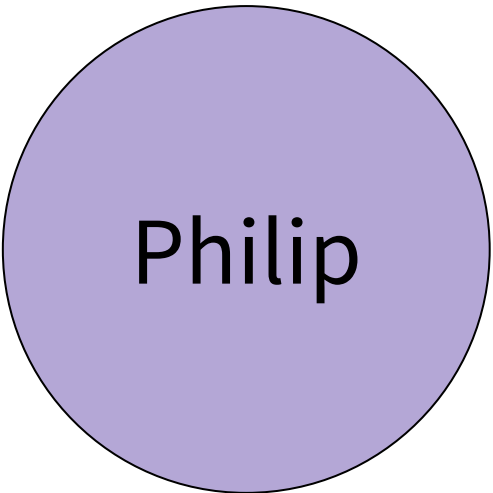
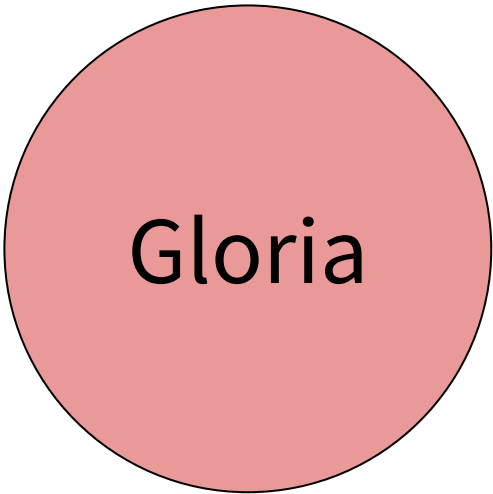
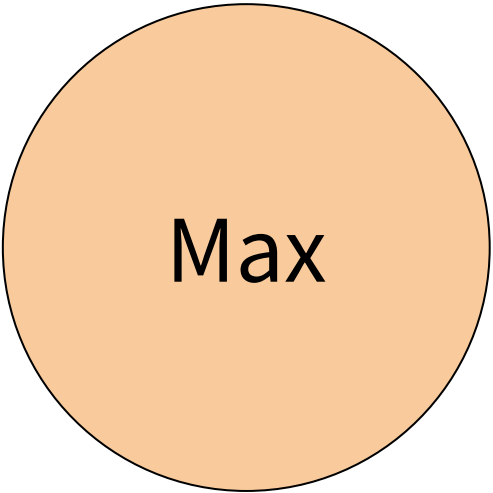
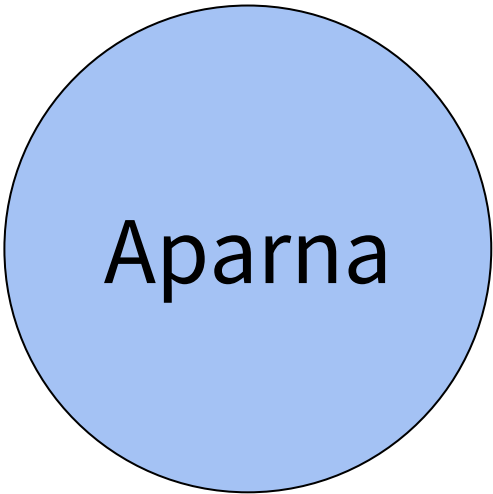
CASE STUDY: GNUTELLA

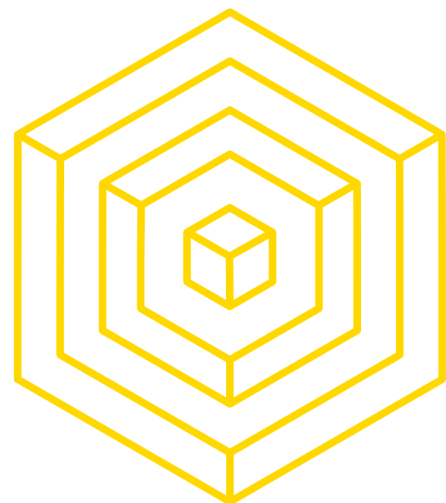
DEFINING FORMALISMS: QUERY/QUERYHIT

QUERY: *bitcoin.jpg* ??????



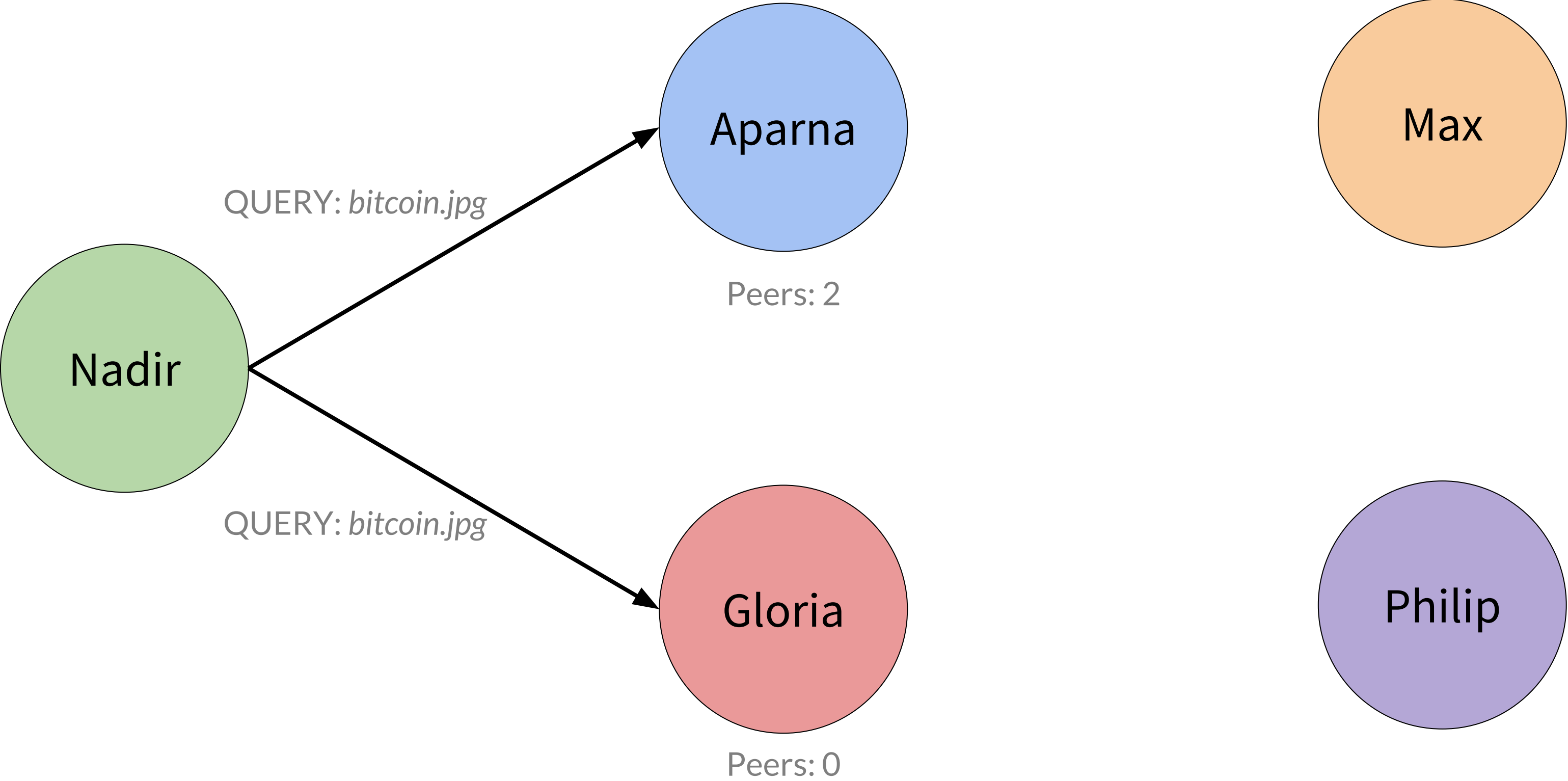
Peers: 2





CASE STUDY: GNUTELLA

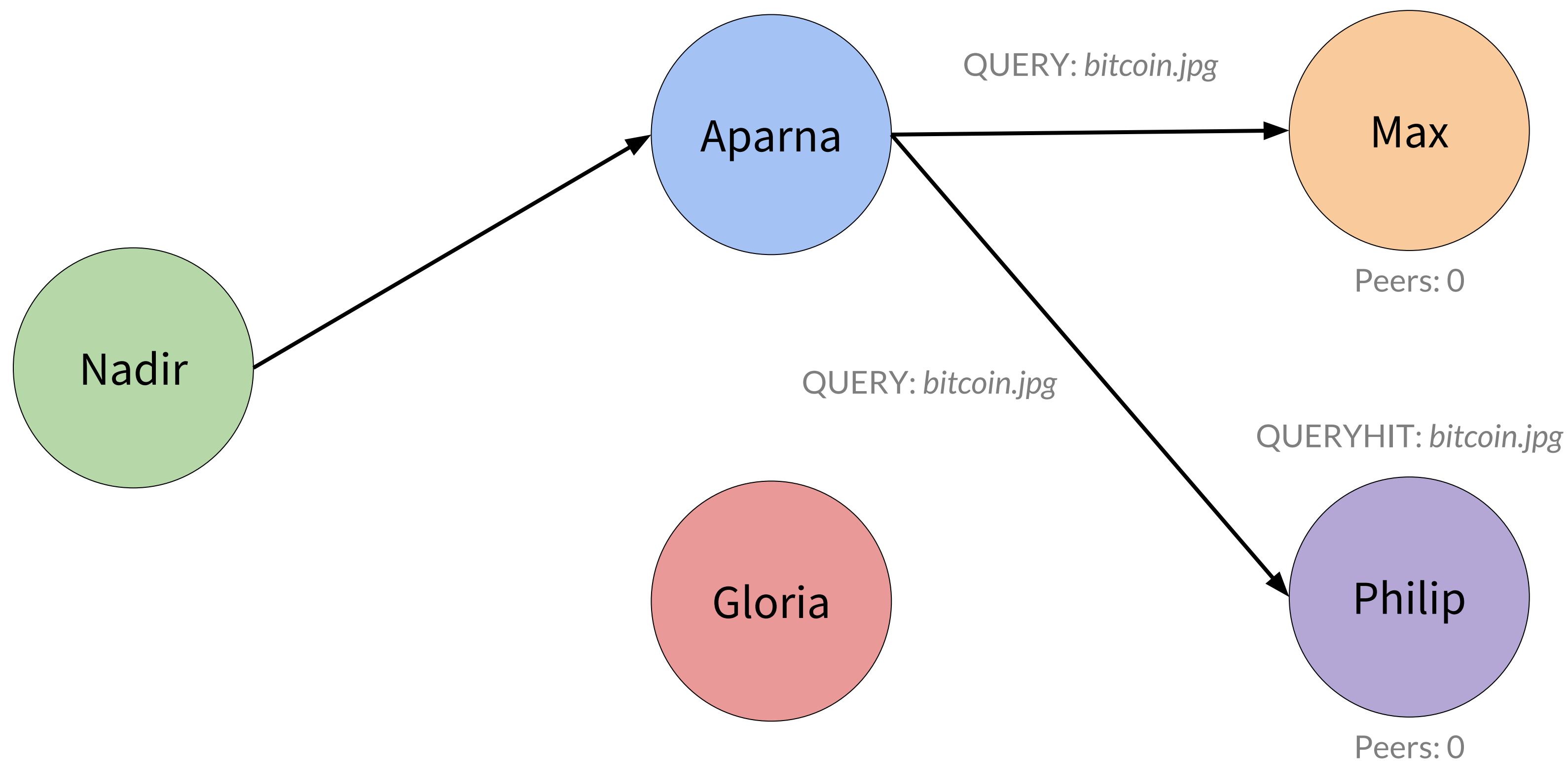
DEFINING FORMALISMS: QUERY/QUERYHIT





CASE STUDY: GNUTELLA

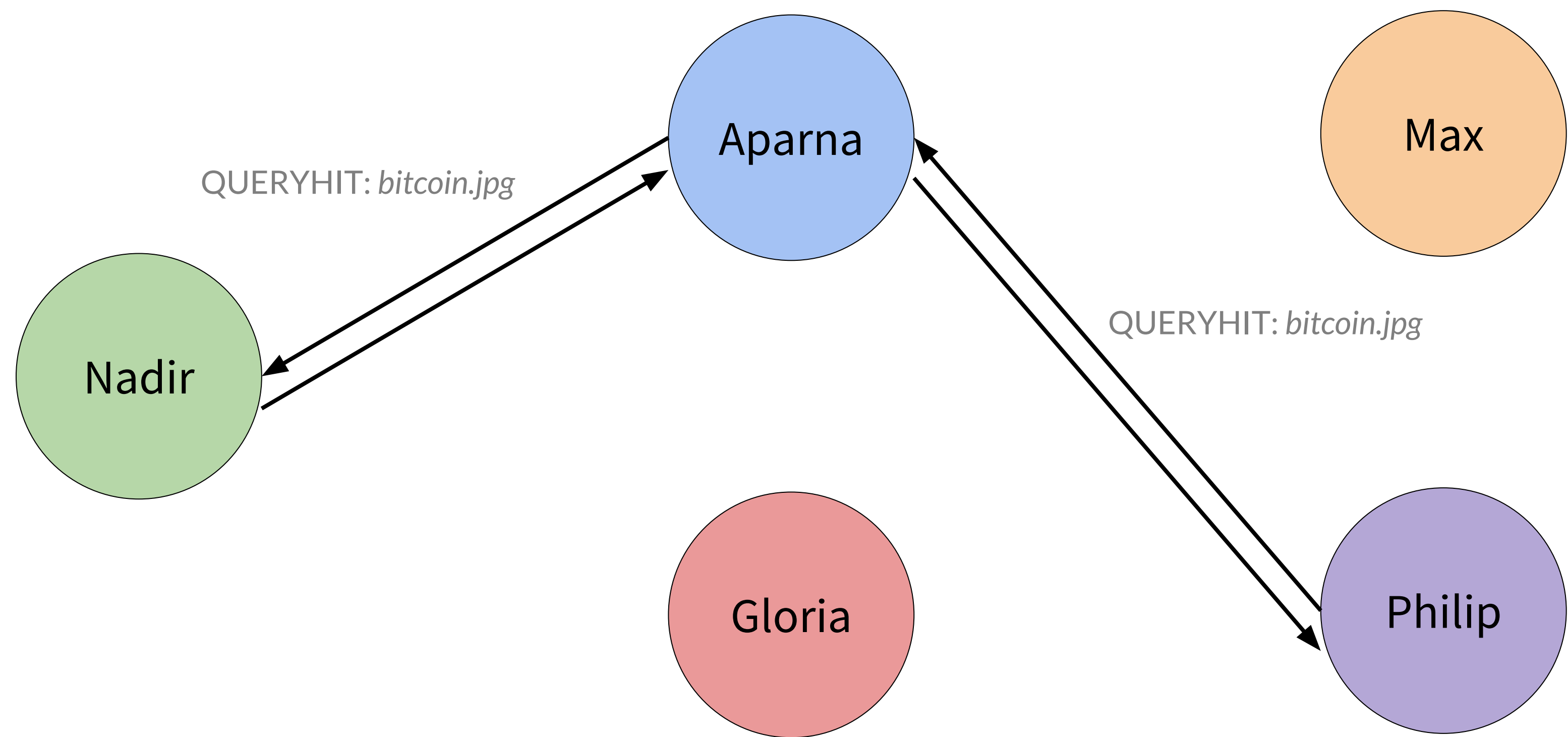
DEFINING FORMALISMS: QUERY/QUERYHIT





CASE STUDY: GNUTELLA

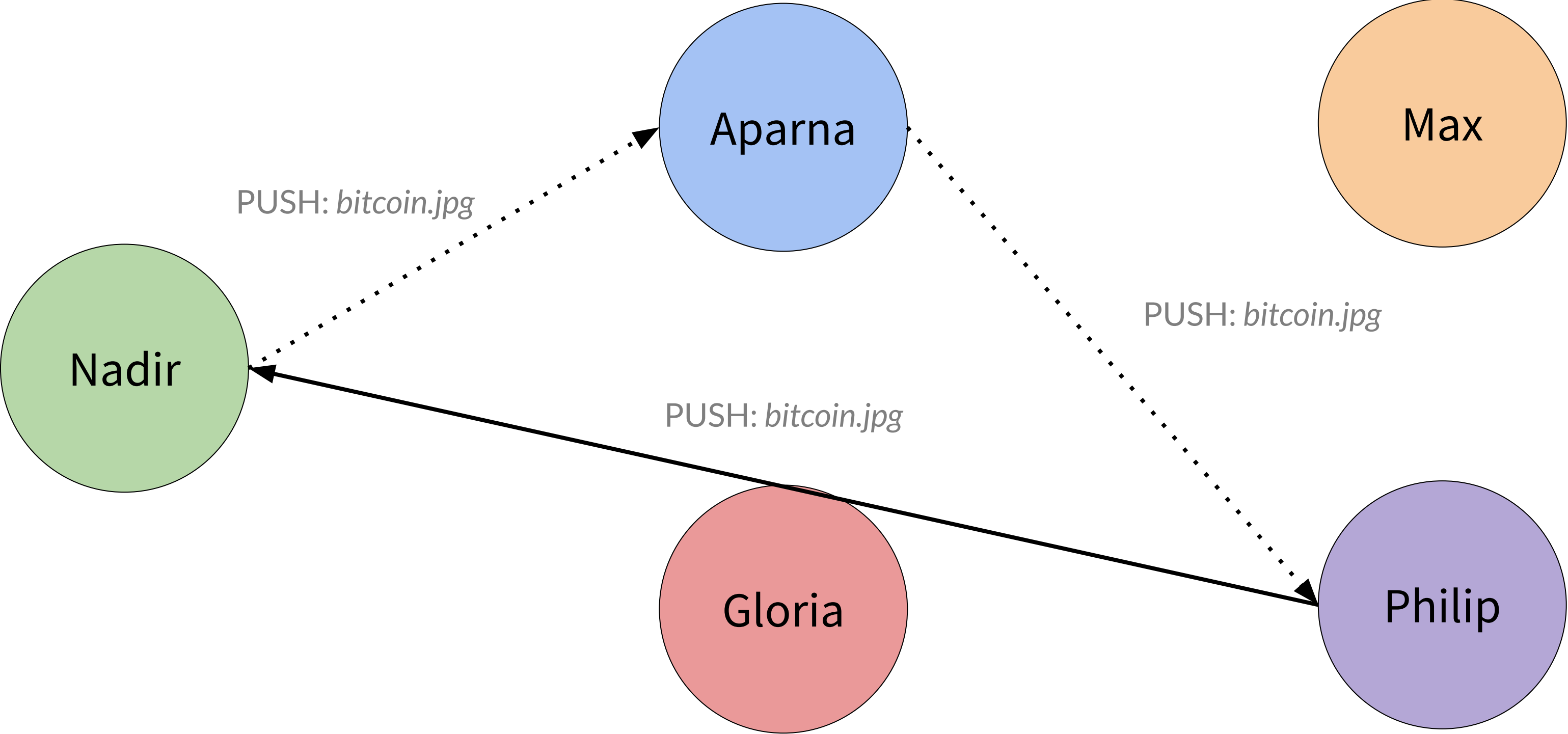
DEFINING FORMALISMS: QUERY/QUERYHIT





CASE STUDY: GNUTELLA

DEFINING FORMALISMS: PUSH



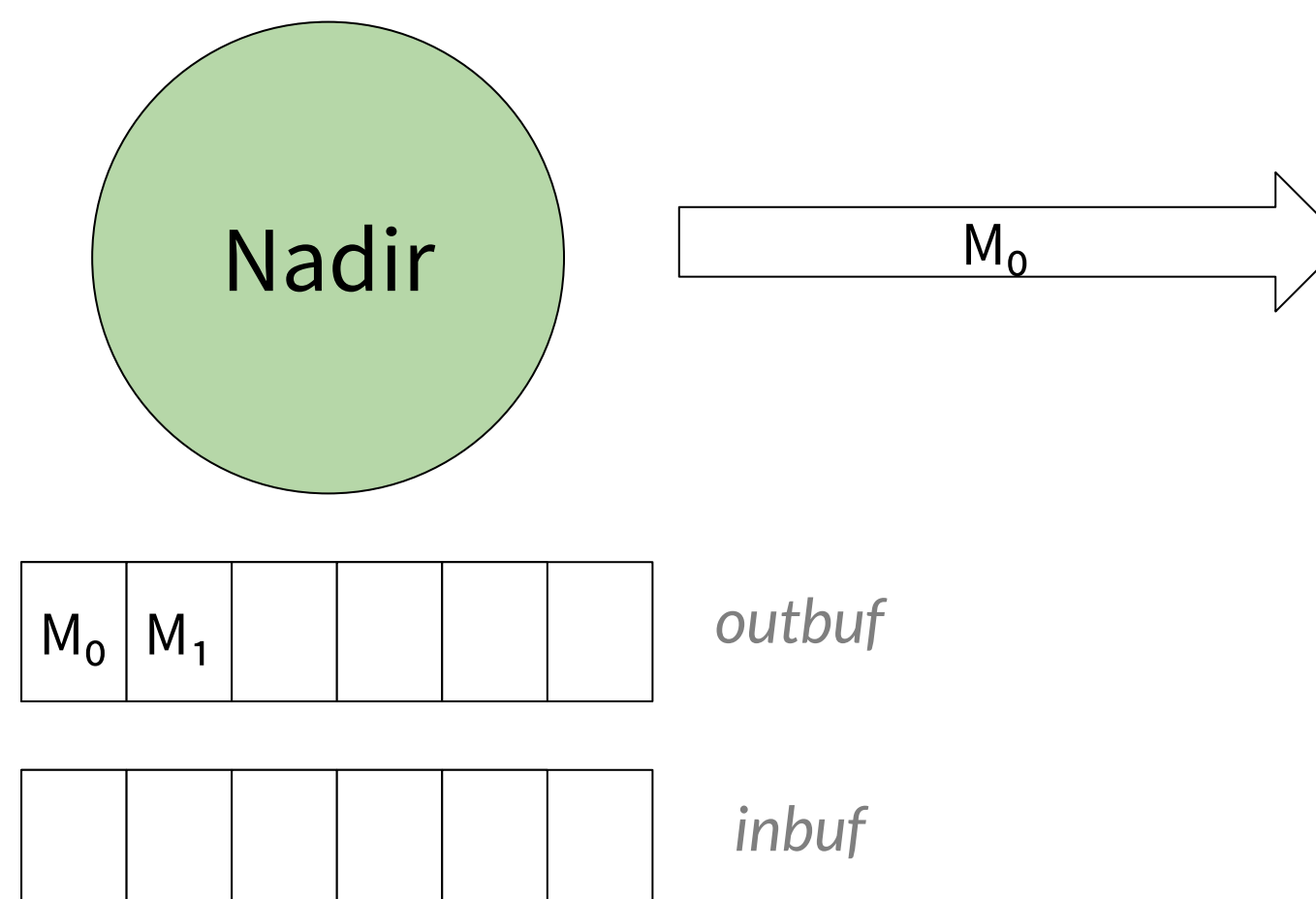


RE-EXAMINING GNUTELLA

ENFORCING FORMALISMS

Distributed systems are made up of a collection of entities that are:

- **Autonomous/Programmable**
- **Asynchronous**
- Failure-prone
- Communicate through an unreliable medium



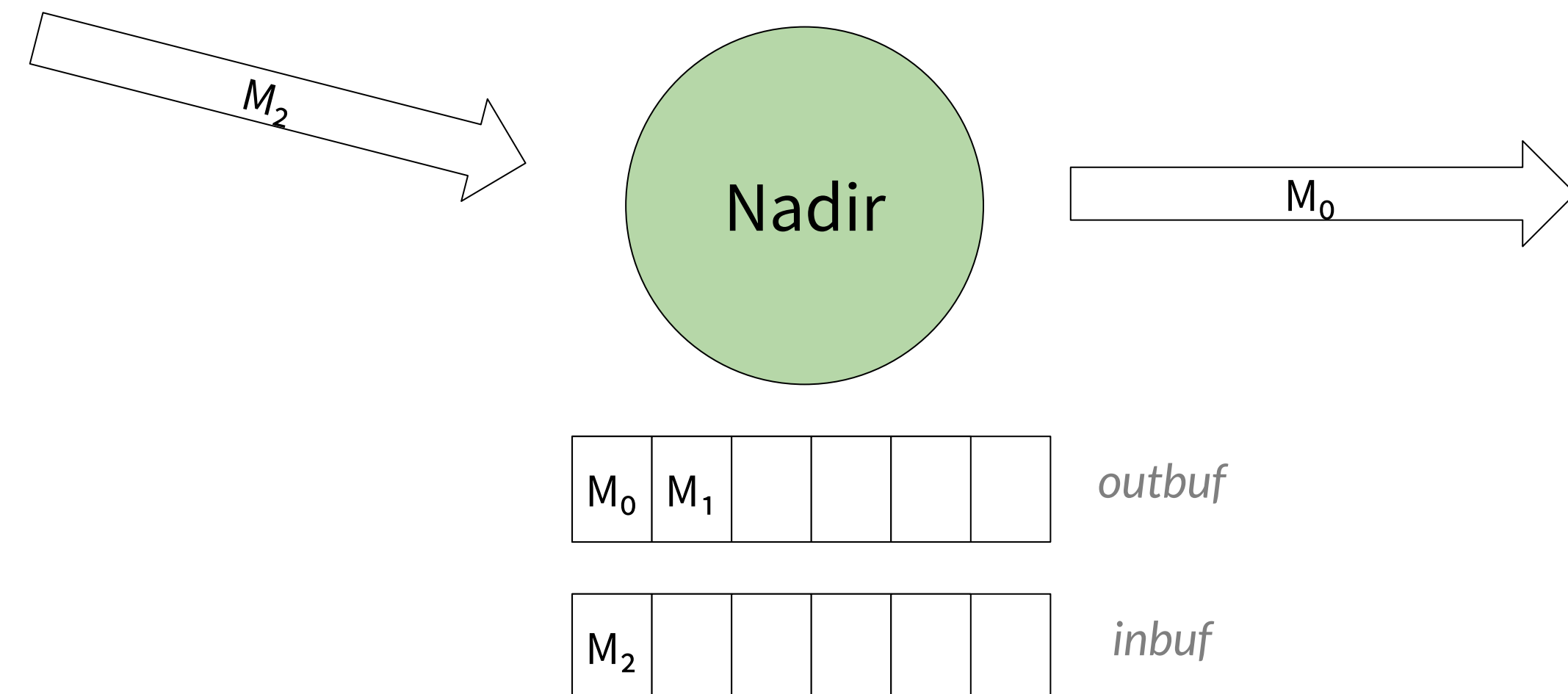


RE-EXAMINING GNUTELLA

ENFORCING FORMALISMS

Distributed systems are made up of a collection of entities that are:

- **Autonomous/Programmable**
- **Asynchronous**
- Failure-prone
- Communicate through an unreliable medium



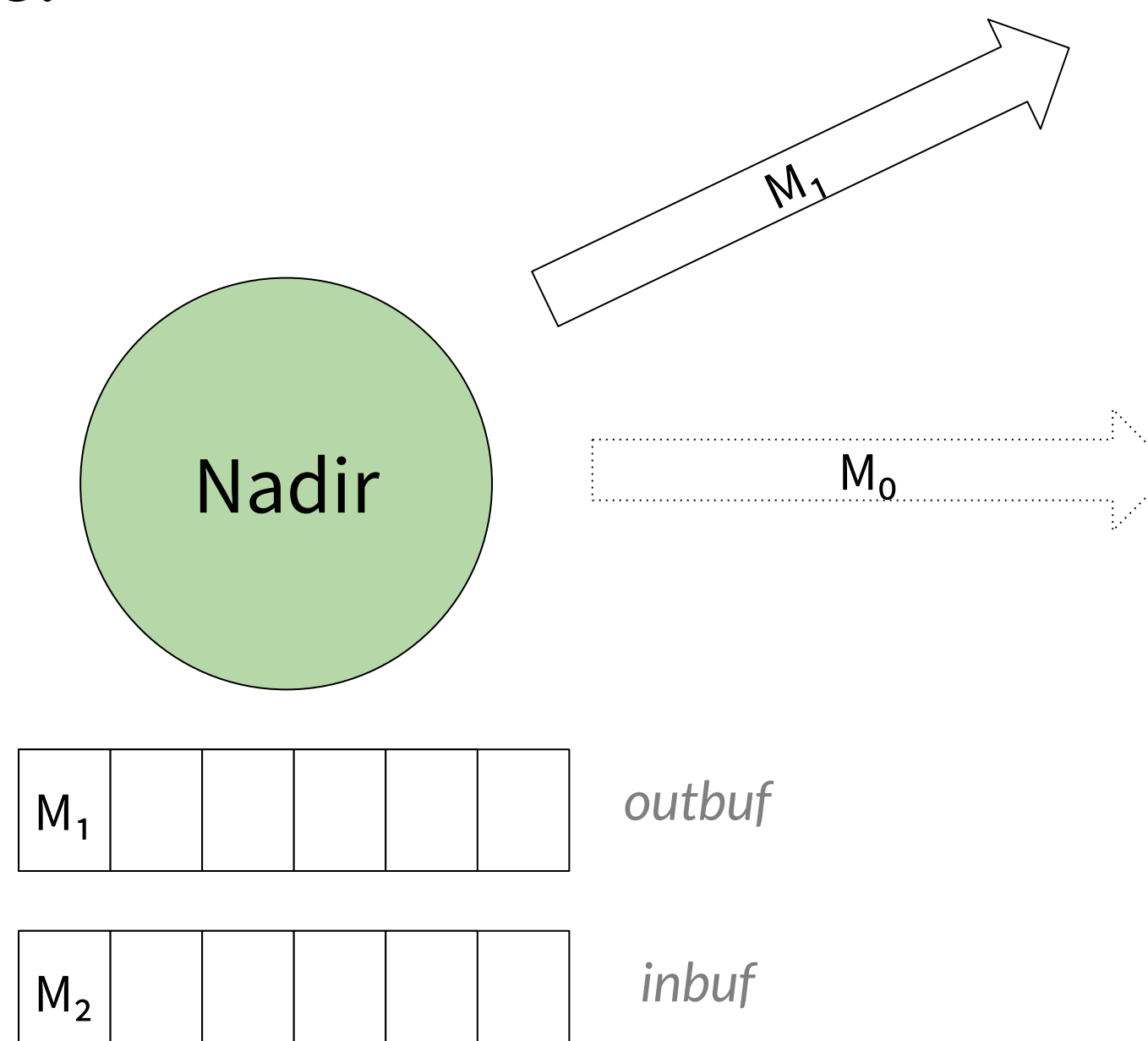


RE-EXAMINING GNUTELLA

ENFORCING FORMALISMS

Distributed systems are made up of a collection of entities that are:

- **Autonomous/Programmable**
- **Asynchronous**
- Failure-prone
- Communicate through an unreliable medium





UDP VS. TCP

PROTOCOL SMACKDOWN

50

User Datagram Protocol

- No error checking
- Fast

Transmission Control Protocol

- Guarantees receipt
- Error checking, tracking

Don't forget: **Internet Protocol (IP), Hypertext Transfer Protocol (HTTP)**

2

DISTRIBUTED SYSTEMS FAULTS AND ATTACKS

2.1

OVERVIEW



TYPES OF FAULTS

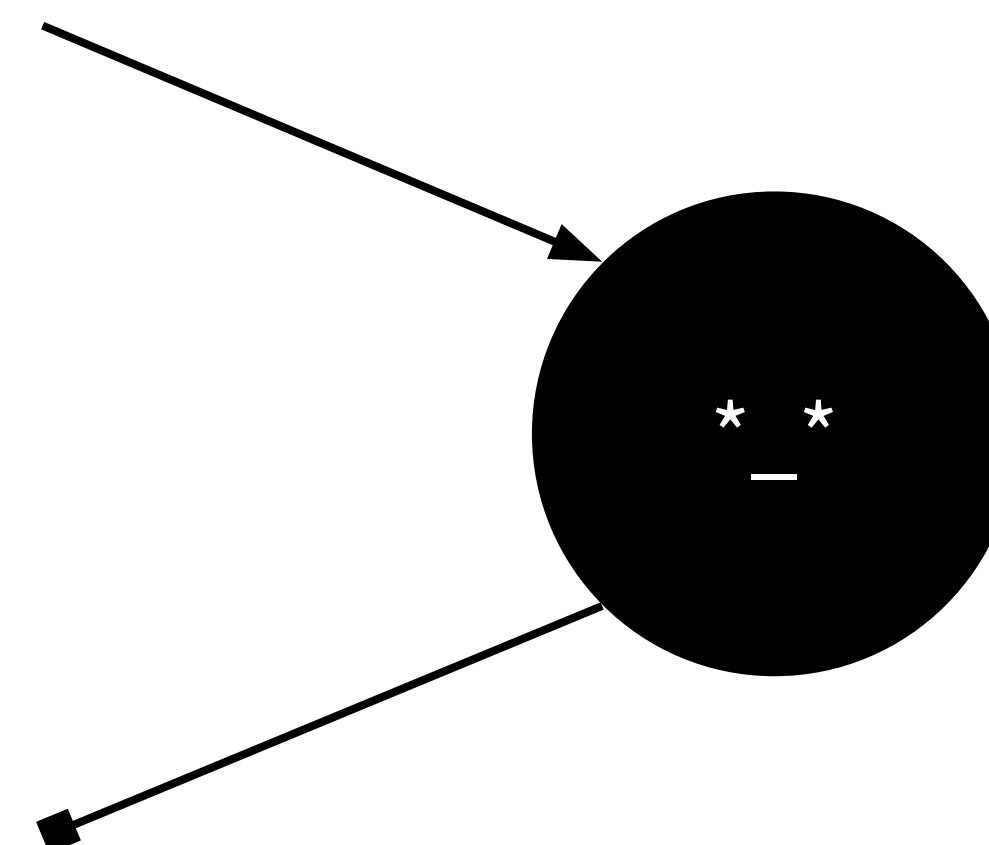
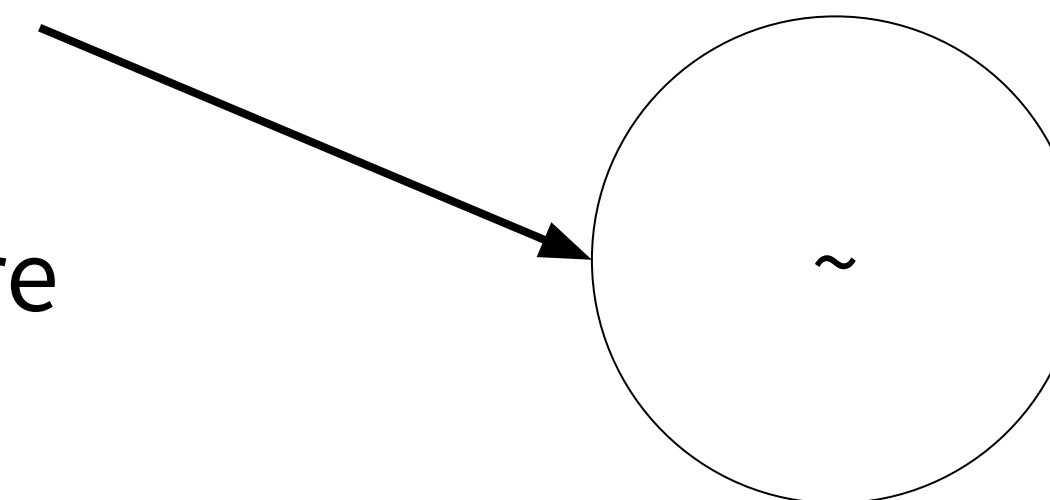
NOT THE ONES IN OUR STARS

Fail-Stop/Crash Failure

- Performance failure -> Omission failure -> Crash failure -> Fail-Stop Failure
- Can default to a certain action

Byzantine Failure

- Arbitrary behavior regardless of input based on the outlines protocol
- Can be malicious or benevolent

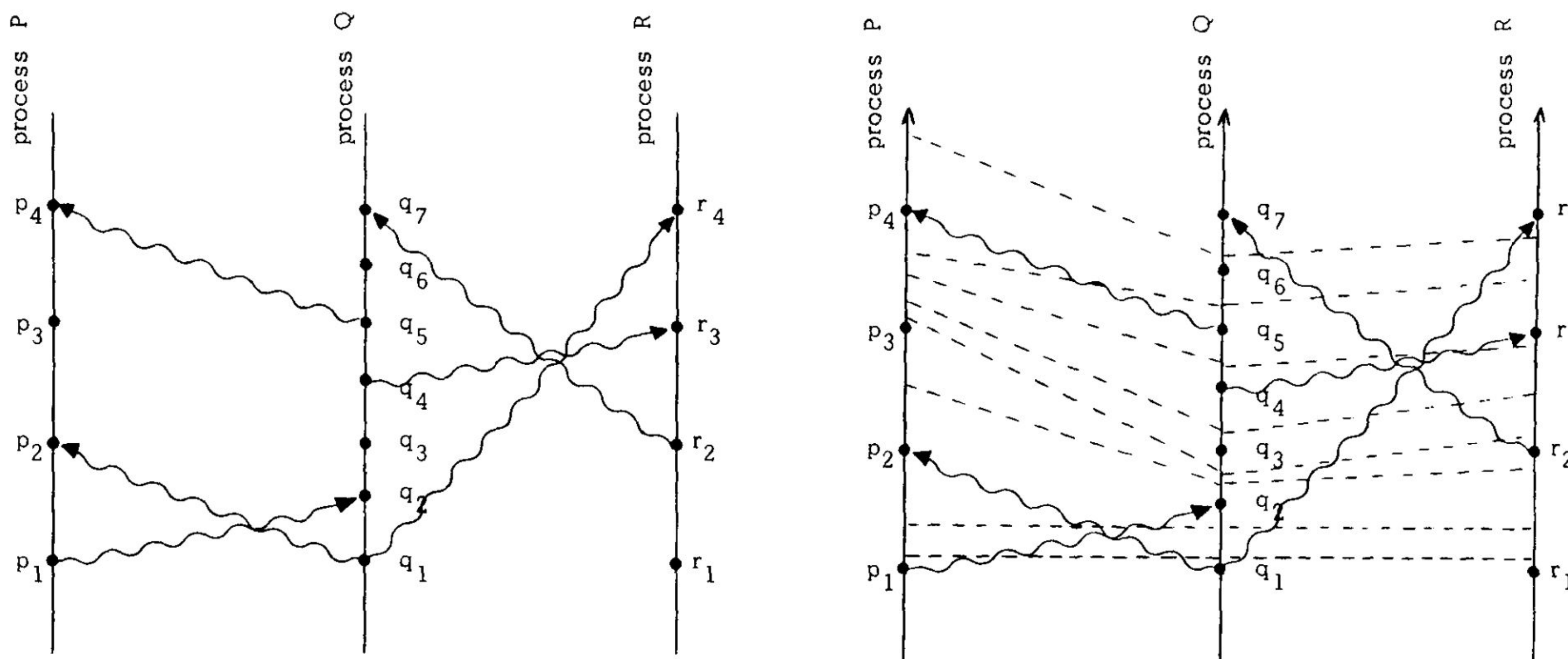




COPING WITH ASYNCHRONICITY

LOGICAL CLOCKS

“Time, Clocks, and the Ordering of Events in a Distributed System,” Leslie Lamport; 1978





BYZANTINE FAULT TOLERANCE

HEARD OF THE BYZANTINE GENERALS' PROBLEM?

Safety

- Results are valid and identical at every node

Liveness (guaranteed termination)

- Nodes that don't fail will produce a result

Correctness

Fault-Tolerance

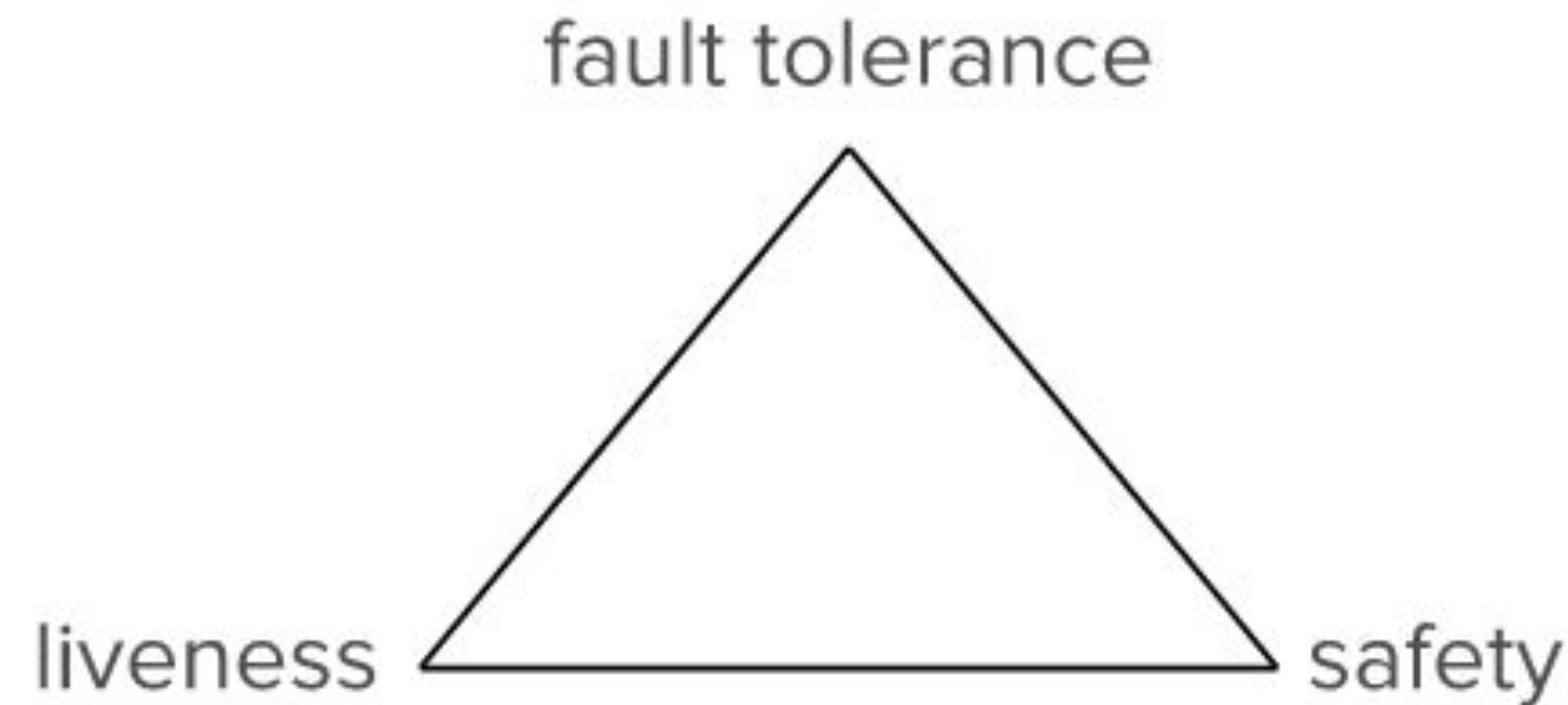
- The system can survive the failure of upto n nodes



FISCHER LYNCH PATERSON IMPOSSIBILITY

YOU CAN'T HAVE IT ALL

FLP impossibility states that asynchronous fault tolerant systems cannot simultaneously satisfy **liveness** and **safety** — thus, distributed systems cannot achieve **fault tolerance** AND **correctness**.





CAP THEOREM

YOU CAN'T HAVE IT ALL, RELOADED

For any distributed data store, the following three properties cannot hold simultaneously:

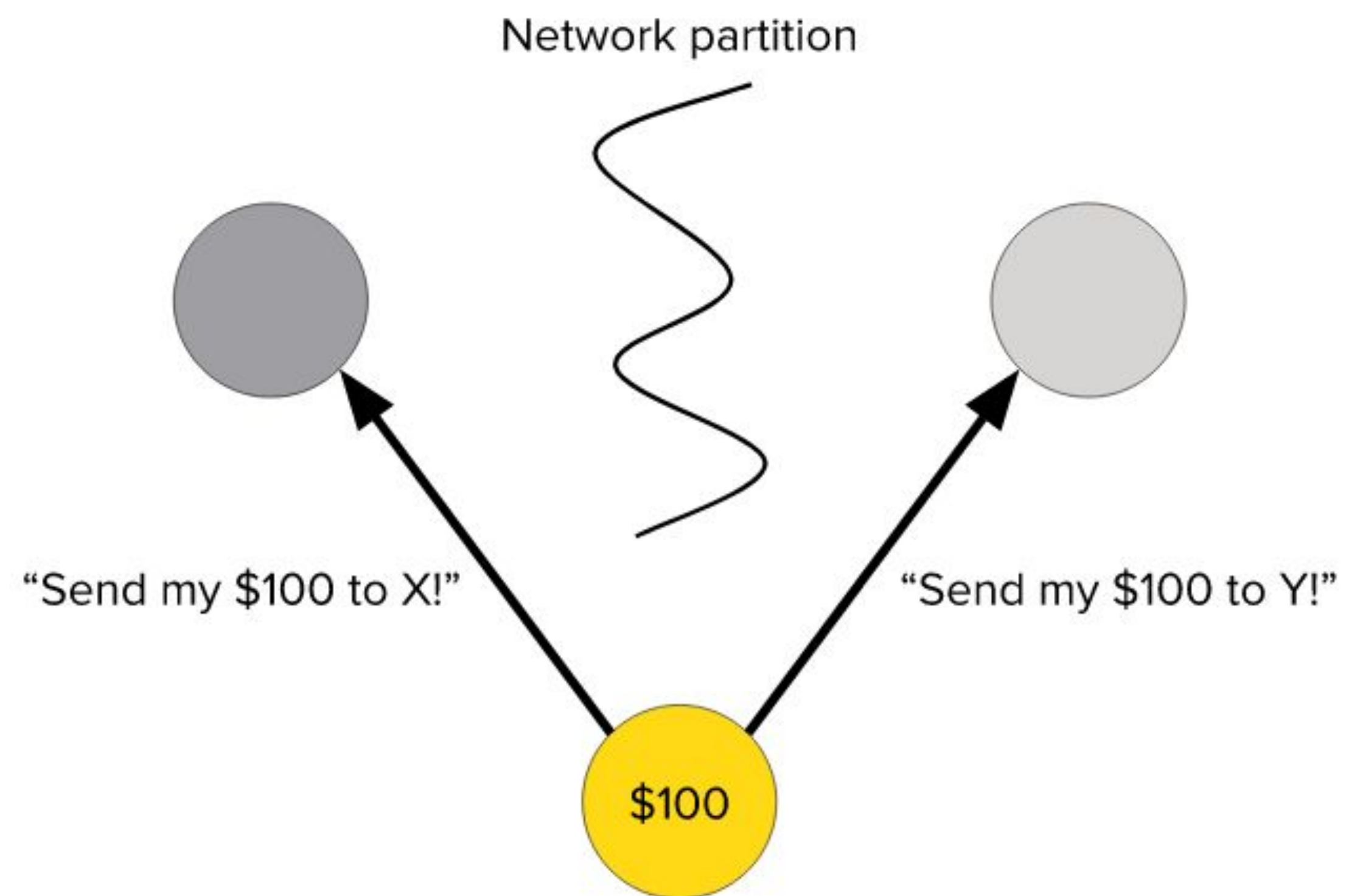
- **Consistency** is a guarantee that reading from each node will return the correct (most recent) write
- **Availability** is a guarantee that reading from any node will return some response
- **Partition tolerance** is a guarantee that the system will still operate in the face of a network partition, across which some messages between nodes cannot be delivered (*fault tolerance*)

More accurately, a choice must be made between **consistency** and **availability** under a network partition.



CAP THEOREM

YOU CAN'T HAVE IT ALL, RELOADED



2.2

FLP, AN APPETIZER



FLP RESULT

LEMMA 2

“Impossibility of Distributed Consensus with One Faulty Process,” Fischer, Lynch and Patterson; 1985

There is some initial configuration in which the decision is not predetermined, but in fact arrived as a result of the sequence of steps taken and the occurrence of any failure.



FLP RESULT

LEMMA 2

Contradiction: assume all initial configurations have predetermined executions

A = '0'

B = '1'



FLP RESULT

LEMMA 1

Contradiction: assume all initial configurations have predetermined executions

A = '0'

B = '1'

Possibilities:

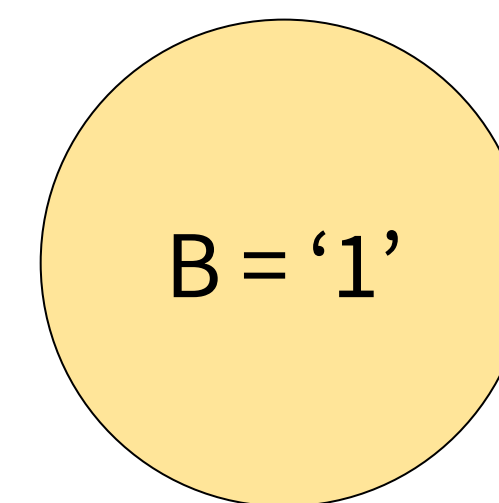
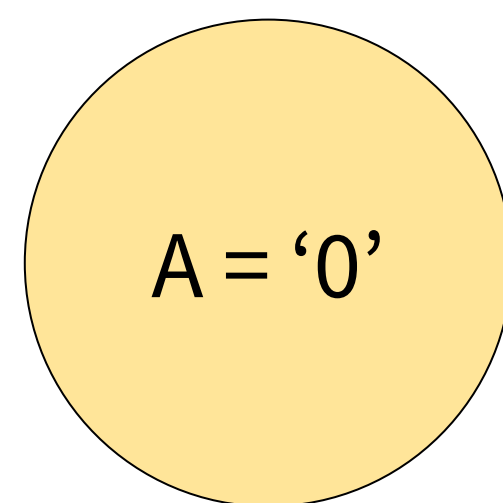
A	B	Observed State (arbitrary)
1	1	1
1	0	1
0	1	1
0	0	0



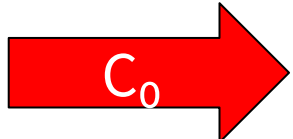
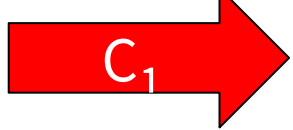
FLP RESULT

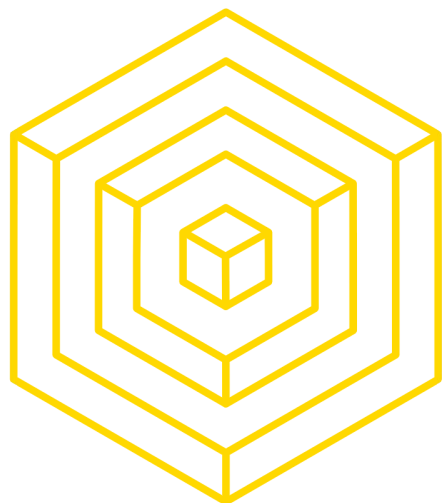
LEMMA 1

Contradiction: assume all initial configurations have predetermined executions



Possibilities:

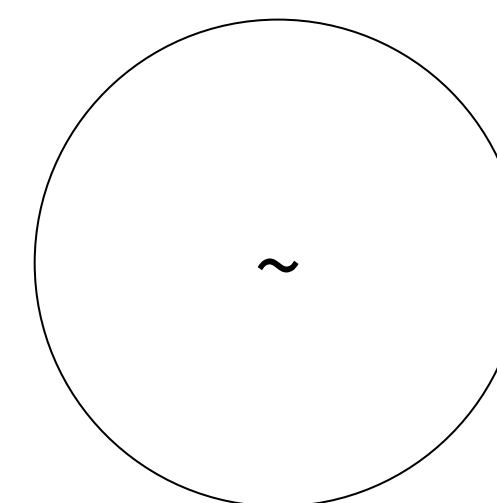
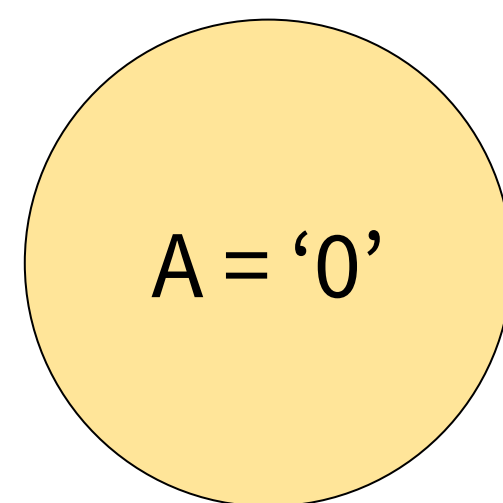
	A	B	Observed State (arbitrary)
	1	1	1
	1	0	1
	0	1	1
	0	0	0



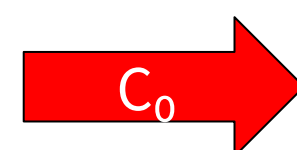
FLP RESULT

LEMMA 1

Contradiction: assume all initial configurations have predetermined executions



Possibilities:



A	B	Observed State (arbitrary)
1		1
1		1
0		1
0		0

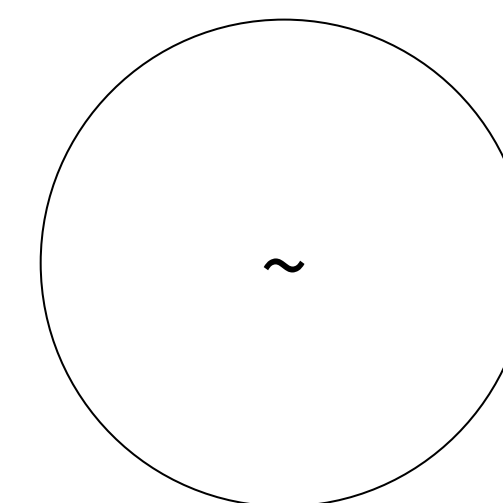
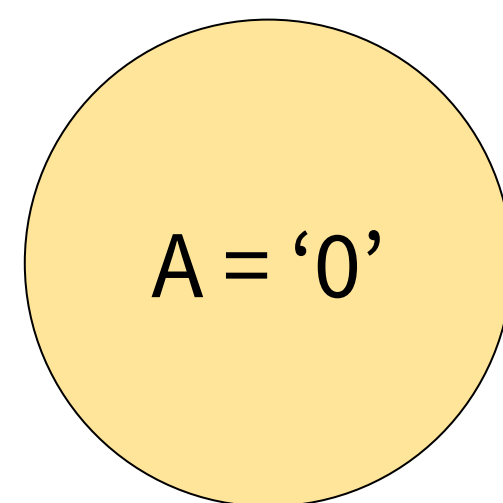


FLP RESULT

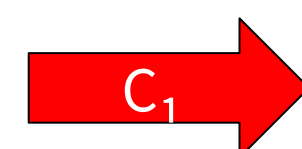
LEMMA 1

65

Contradiction: assume all initial configurations have predetermined executions



Possibilities:



A	B	Observed State (arbitrary)
1		1
1		1
0		1
0		0

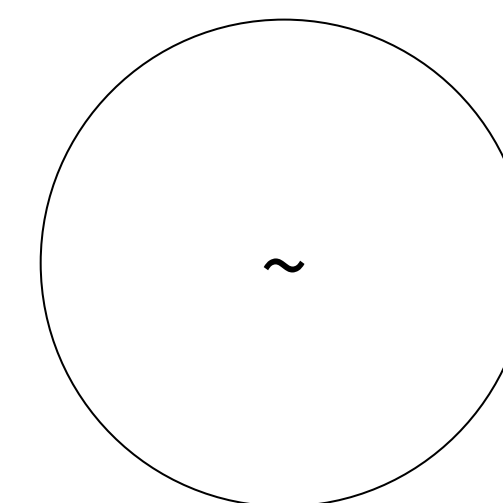
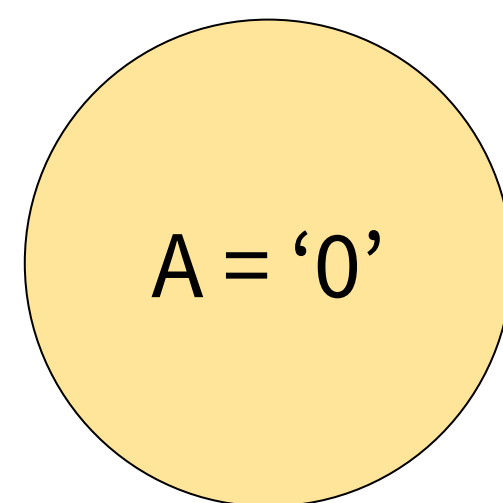


FLP RESULT

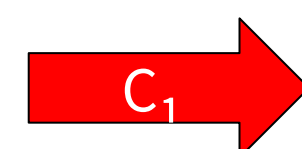
LEMMA 1 ✓

66

Contradiction: assume all initial configurations have predetermined executions ✕



Possibilities:



A	B	Observed State (arbitrary)
1		1
1		1
0		1
0		0

2.3

ECLIPSE ATTACKS

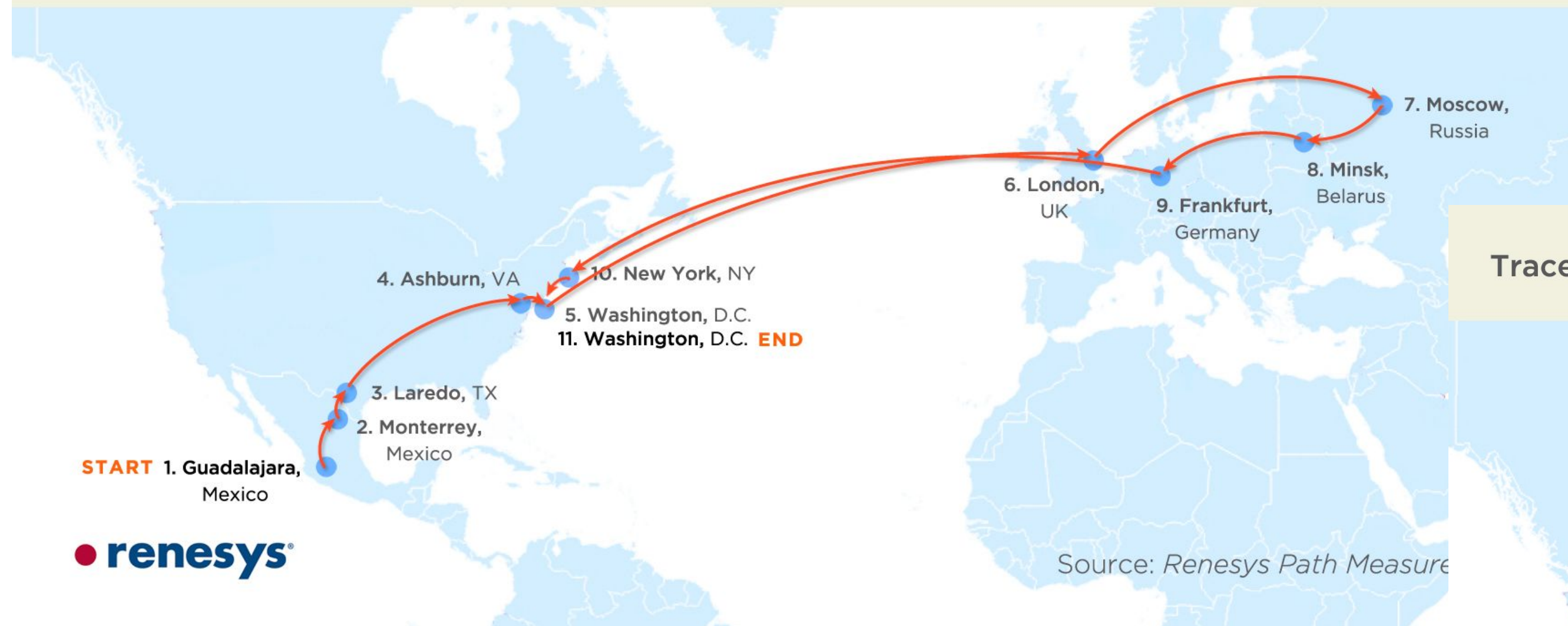




BORDER GATEWAY PROTOCOL HIJACKING

DATA REDIRECTION, 2016

Traceroute Path 1: from Guadalajara, Mexico to Washington, D.C. via *Belarus*



Read more [here](#)!

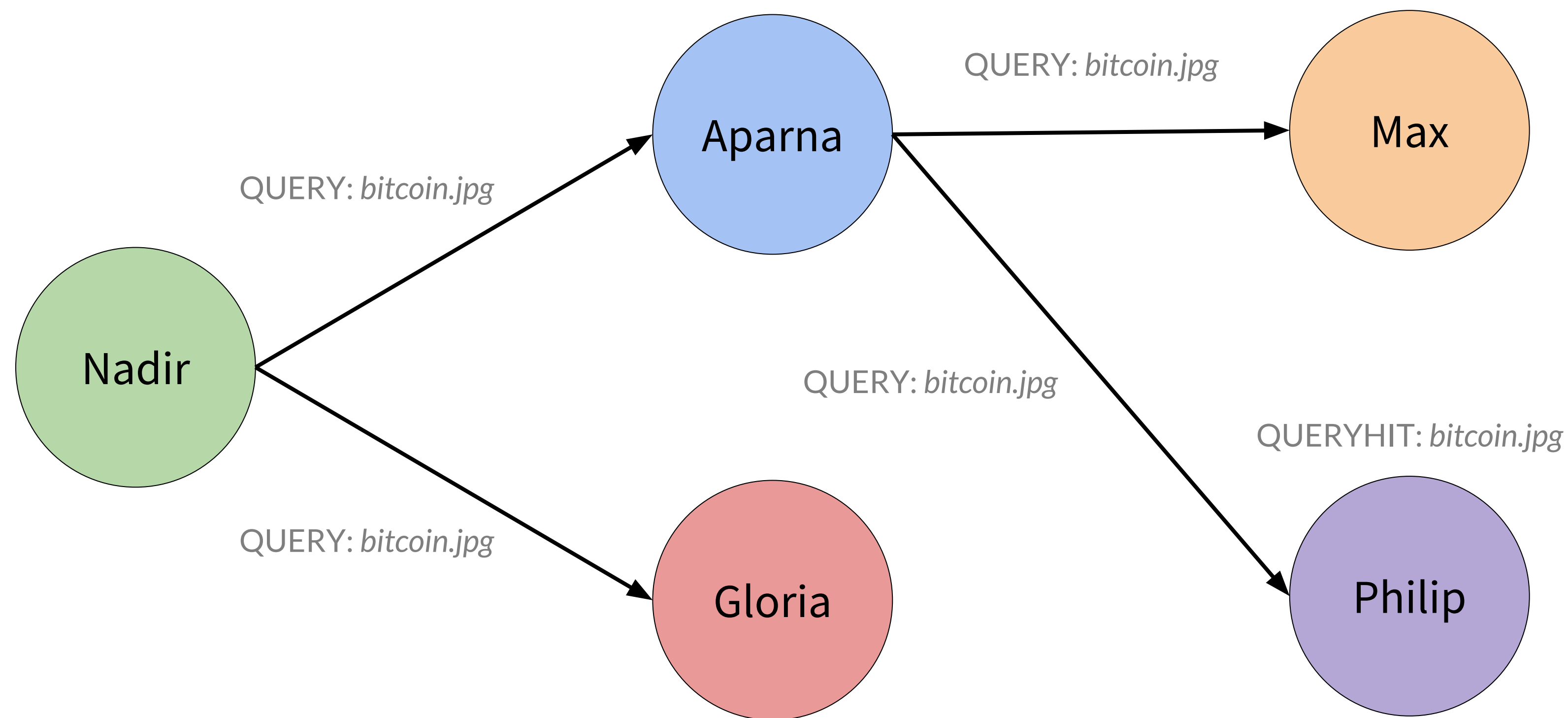
Traceroute Path 2: from Denver, CO to Denver, CO via *Iceland*





SYBIL ATTACKS

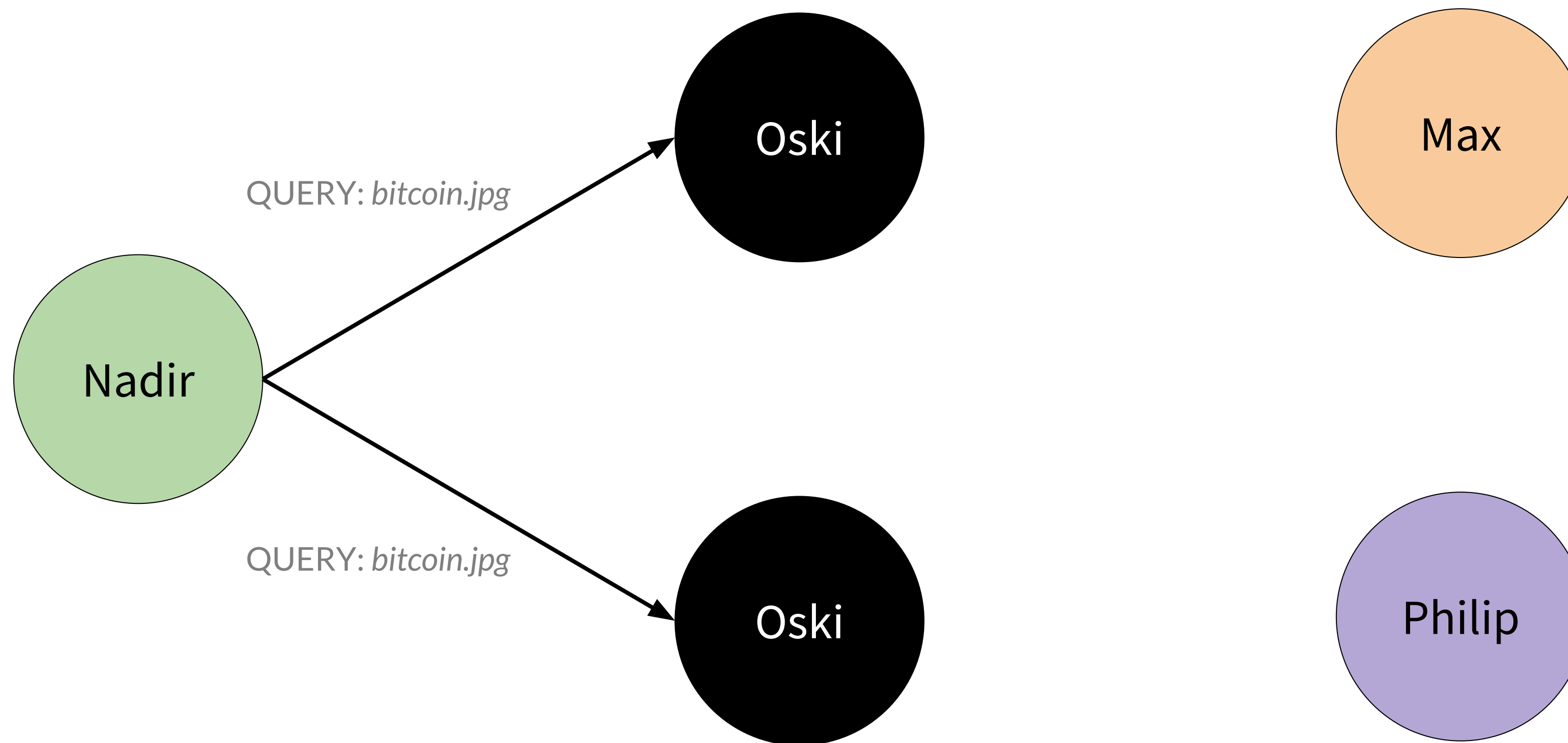
PSEUDOSPOOFING, 2015





SYBIL ATTACKS

PSEUDOSPOOFING

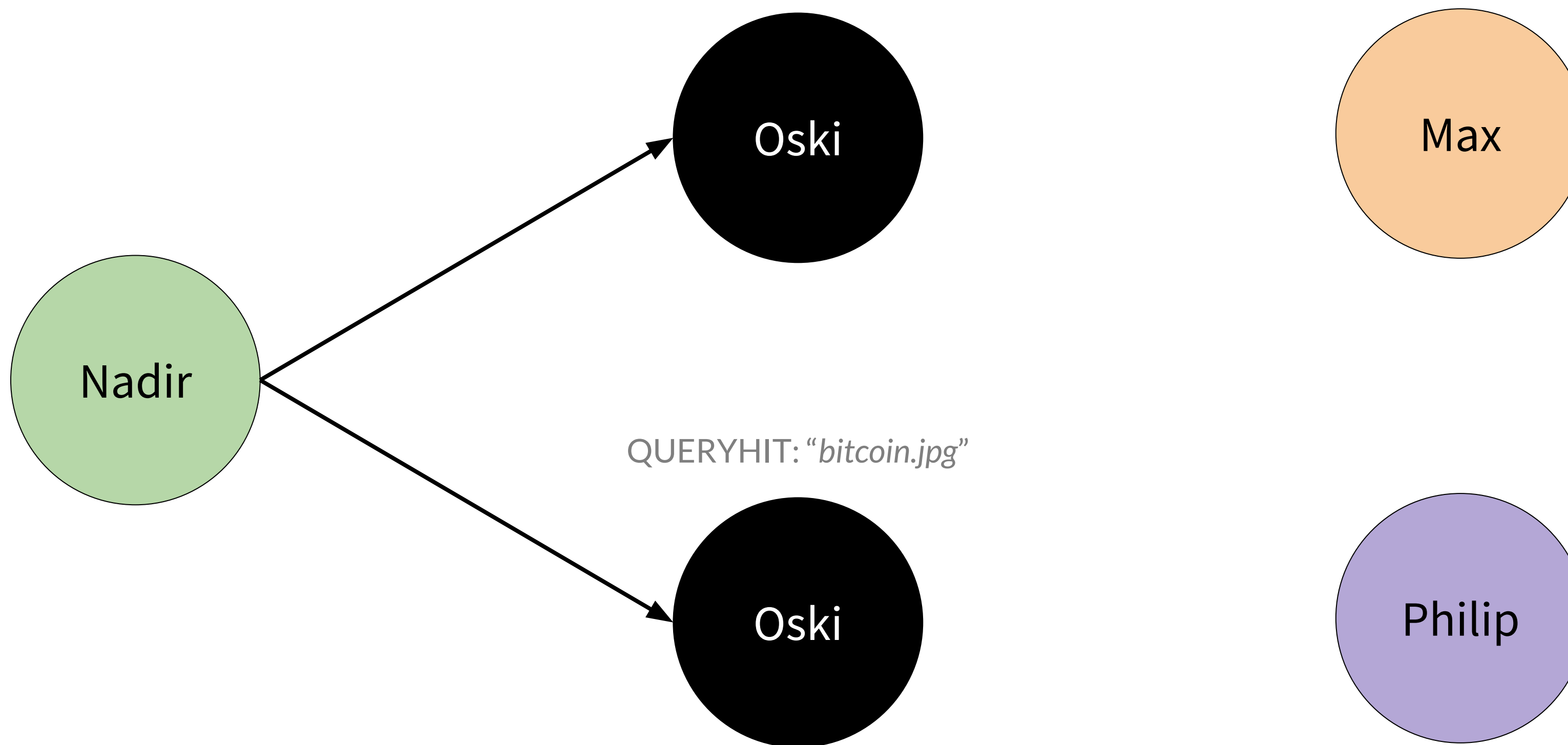




SYBIL ATTACKS

PSEUDOSPOOFING

72

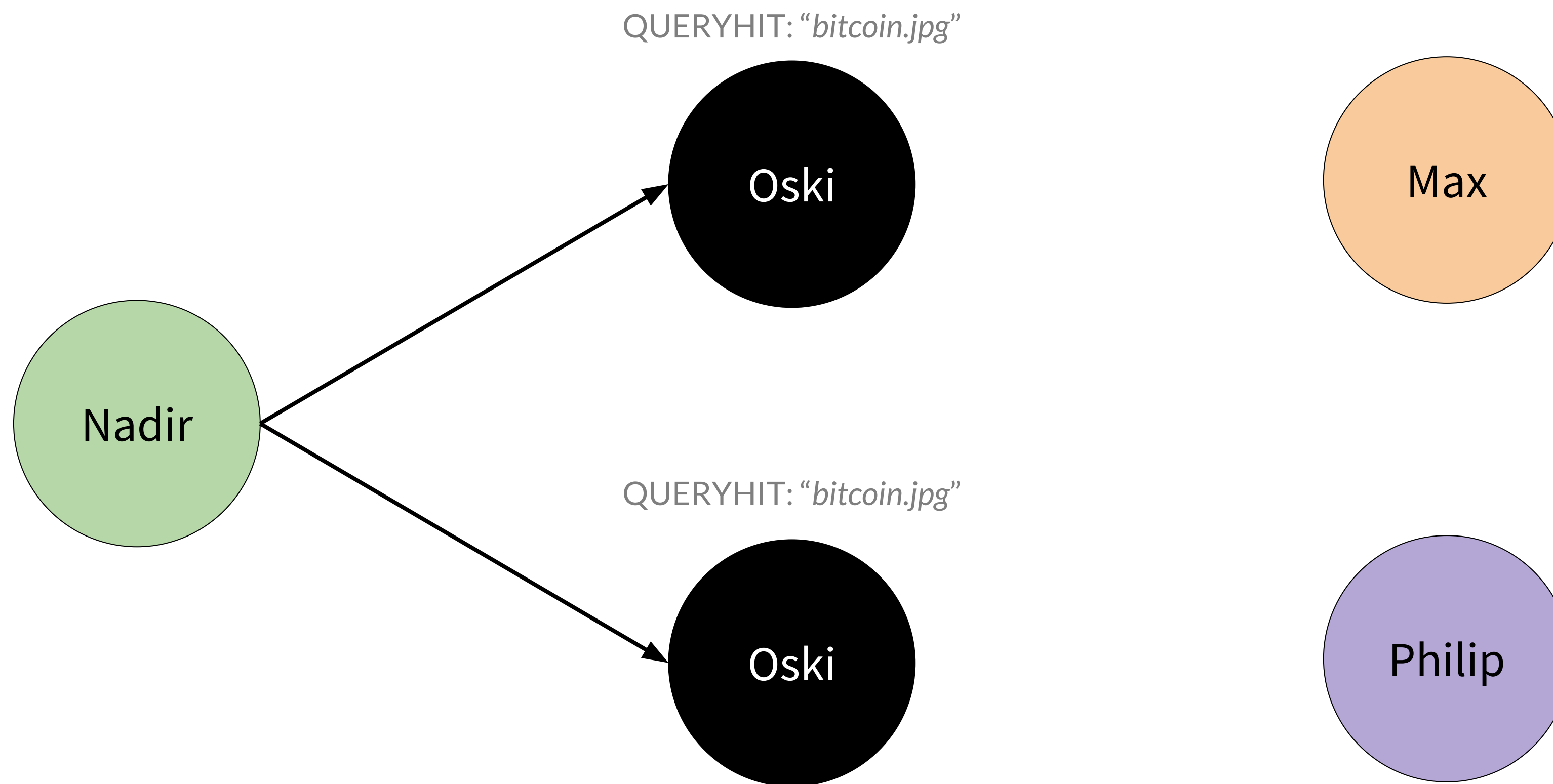




SYBIL ATTACKS

PSEUDOSPOOFING

73





SYBIL ATTACKS

ON BITCOIN

- The attacker can refuse to relay blocks and transactions from everyone, effectively disconnecting you from the network
- The attacker can relay only blocks that they create, effectively putting you on a separate network and then also leaving you open to double-spending attacks
- if you rely on transactions with 0 confirmations, the attacker can just filter out certain transactions to execute double-spending attack
- Slow-latency encryption/anonymization of Bitcoin's transmissions (with Tor, JAP, etc.) can be defeated relatively easily with a timing attack if you're connected to several of the attacker's nodes and the attacker is watching your transmissions at your ISP



TIME ATTACKS

ON ETHEREUM

75

- Set the target's system clock 20 seconds ahead
 - The Ethereum protocol rejects messages that are more than 20 seconds old
 - To prevent *replay attacks*
 - The target loses touch with all legitimate users
- Attackers use malicious nodes with the same clock time to connect to the target

3

PEER-TO-PEER PROTOCOLS



STAGES OF P2P

HOW P2P EVOLVED

- The first generation peer-to-peer file sharing networks, such as **Napster** relied on a **central database** to co-ordinate lookups on the network
- Second generation p2p such as **Gnutella** used flooding to locate files, which means they **searched every node on the network**
- Third generation p2p networks used **Distributed Hash Tables** to look up files on the network, which meant storing resource locations throughout the network



NAPSTER

HOW P2P EVOLVED

78

- The first large scale P2P content delivery system
- Required a central index server
- Each node upon joining would send a list of locally held files to the server which would perform searches and refer the queries to the nodes that held the results
- Vulnerable to attacks and lawsuits
- Motivated the need to do research on p2p networks, which included **Distributed Hash Tables** (DHT)
 - Decentralized indexing would fix the weakness of trusting a single party to be online



DISTRIBUTED HASH TABLE

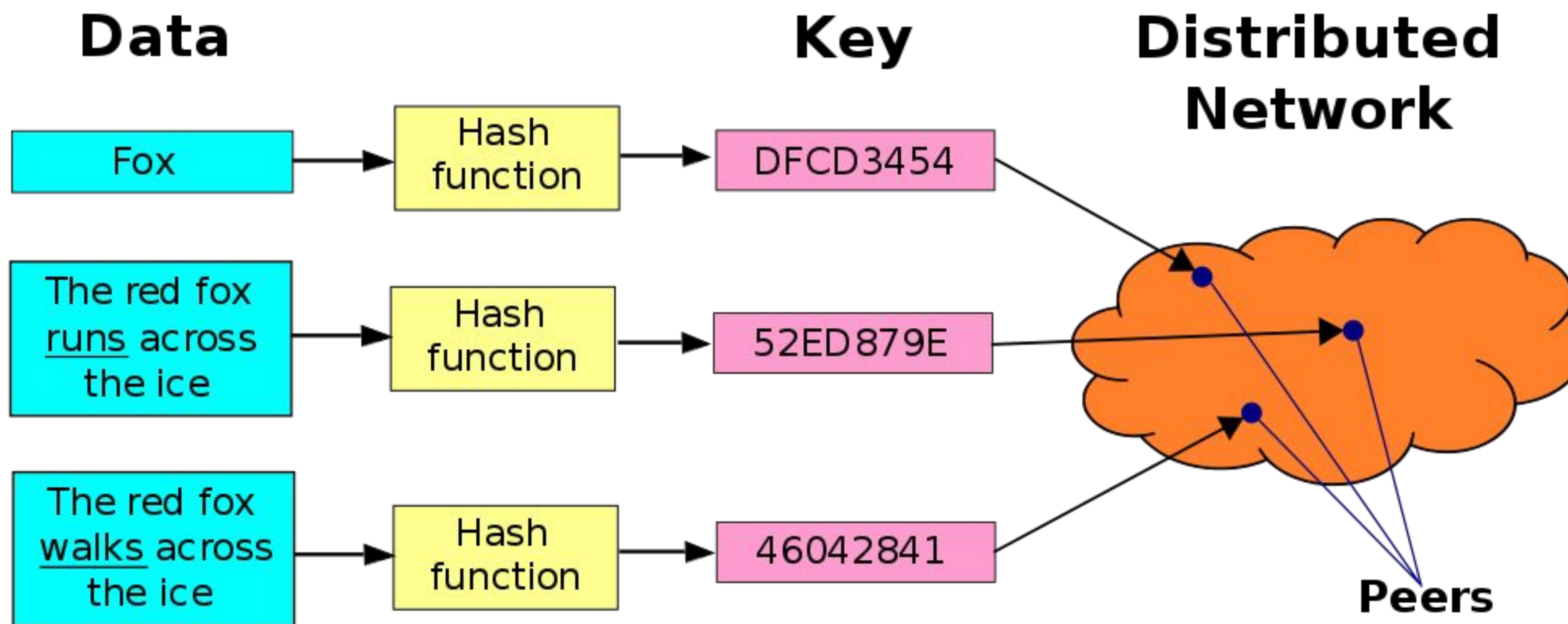
HIGH LEVEL

- A DHT stores key-value pairs by assigning keys to different computers (nodes)
- A node will store the values for all the keys for which it is responsible
- The protocol specifies how keys are assigned to nodes and how a node can discover the value for a given key by first locating the node responsible for that key
- The four original DHT protocols: Chord, CAN, Tapestry, Pastry (definitely read these papers if you're interested)



DISTRIBUTED HASH TABLE

HIGH LEVEL





DISTRIBUTED HASH TABLE

HIGH LEVEL

- Properties:
 - **Autonomy and decentralization:** The nodes collectively form the system without any central coordination
 - **Fault tolerance:** the system should be reliable even with the nodes continuously, joining, leaving and failing
 - **Scalability:** the system should function efficiently even with thousands or millions of nodes
- DHTs also must also deal with load balancing, data integrity and performance
- A major criterion is locating the desired nodes quickly, usually $O(\log n)$ time
- DHT that is carefully designed to have Byzantine Fault Tolerance can defend against a Sybil attack



BITTORRENT DHT PEER DISCOVERY

ABSTRACTING AWAY THE TECHNICALITIES OF A DHT

- BitTorrent has a bunch of clients (and servers) where clients download parts of files while uploading to other portions of the network based on its own fairness algorithm
 - Peer Optimization techniques such that most people get the best outcome and sharing happens in an effective and efficient way
- Goal: we need to match up with other peers, do uploads and downloads between each of them based on the file we want
- We find peers via a process called **bootstrapping**, and this is done via a **DHT** node in the network
- Is it decentralized? How do you really start the network?



BITTORRENT DHT PEER DISCOVERY

IS IT DECENTRALIZED?

- Difficult to have decentralization, because the internet is very **unicast** in nature
 - If you're connected to the web, you don't announce your presence to the billion of computer connected, that would be wasteful in bandwidth
- Your ISP's local router and destinations you connect to are the only ones that know you are online
 - **Multicast**: when your computer is made known to every other computer in the same subnet
 - But this is not what happens on the actual internet!
 - So the DHT Peer Discovery is not really decentralized for BitTorrent, because you need to hit up a single DHT at some point to find other peers, there is always some metadata you have hardcoded in the client because of the **unicast** nature of the internet



BITTORRENT DHT PEER DISCOVERY

WHAT HAPPENS WHEN DOWNLOAD A TORRENT FILE

- This is what is known as the “**bootstrapping**” process
- You connect to a set of DHTs, specifically router.bittorrent.com and router.utorrent.com on port 6881
 - Note that these don't have to be in tandem/agreement about what files are stored where, you select one server to bootstrap (Why might they do this?)
- The server will send out a handful of peer IP addresses that you'll also connect to
- You connect to those peers, they give you the address of peers they're connected to and so on until your peer list shows all the peers downloading (or uploading) the file you're trying to get
- Benefit: You only need a single peer address to get all the other peers, which saves DHT bandwidth

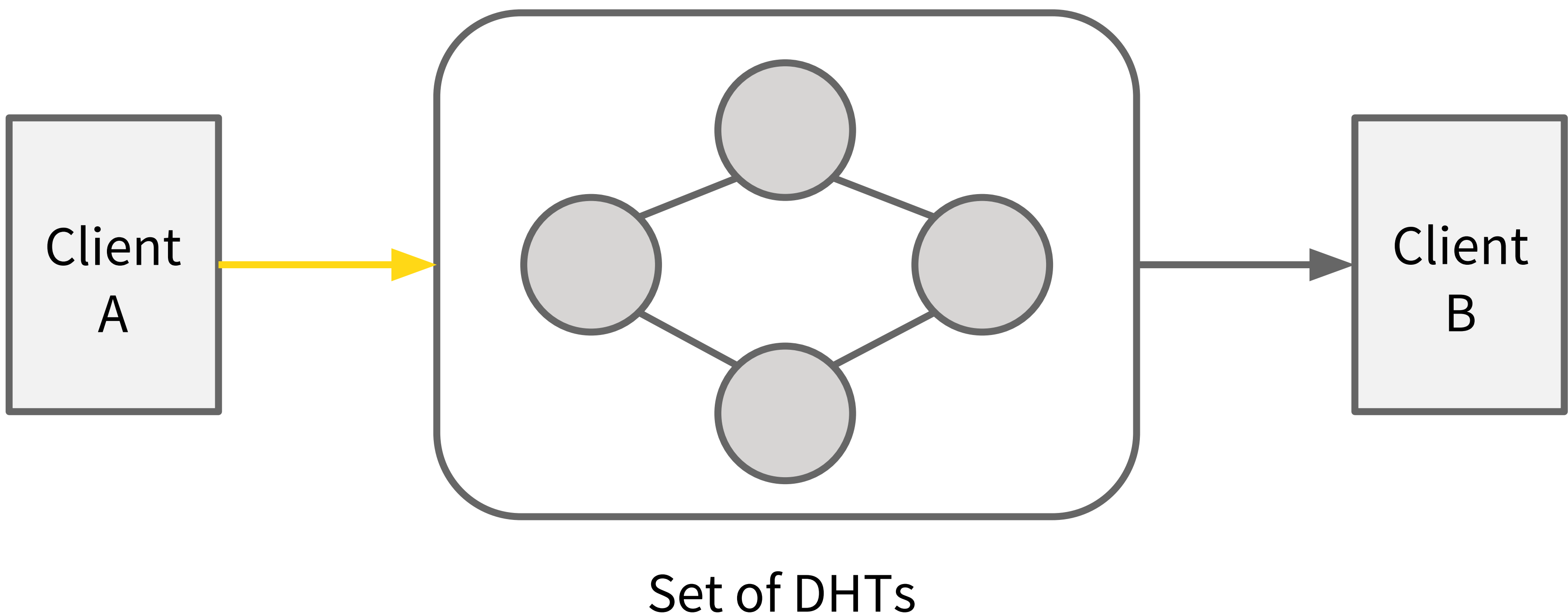


BITTORRENT DHT PEER DISCOVERY

WHAT HAPPENS WHEN DOWNLOAD A TORRENT FILE

Client A to one DHT in the set of DHTs

I'm on torrent 68f73bc839e7e9d4935808febca823e2 (infohash), you heard of anyone else on that one? Here's my <IP address, port number>

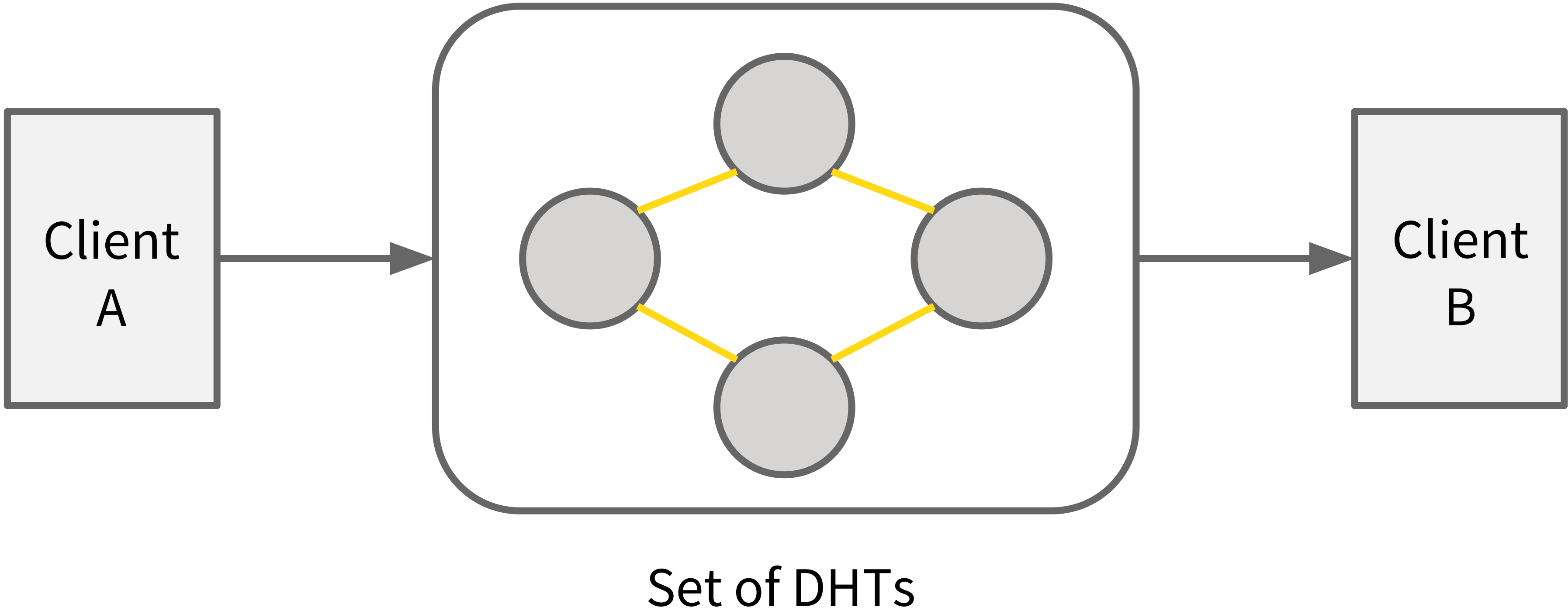




BITTORRENT DHT PEER DISCOVERY

WHAT HAPPENS WHEN DOWNLOAD A TORRENT FILE

The corresponding DHT broadcasts this infohash to other DHTs it knows of, and if we don't find that infohash within the current scope of DHTs, the DHTs that were broadcasted to do a broadcast on their end.

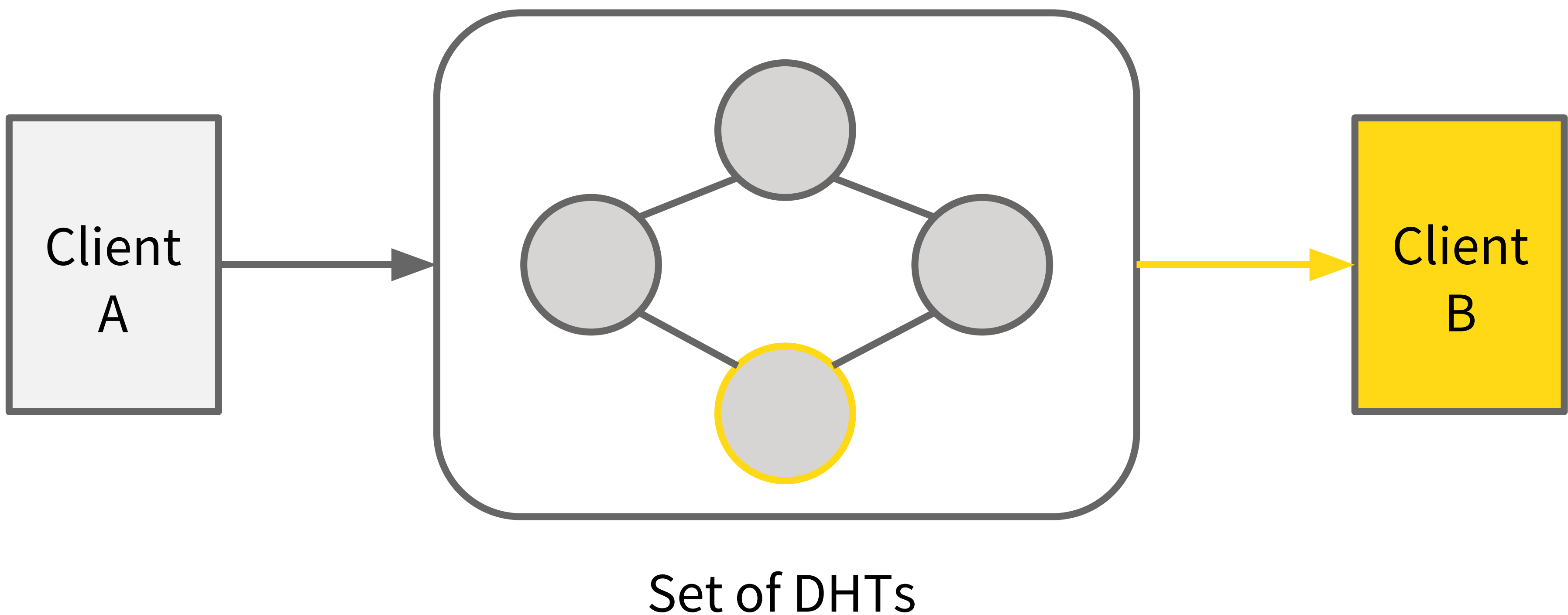




BITTORRENT DHT PEER DISCOVERY

WHAT HAPPENS WHEN DOWNLOAD A TORRENT FILE

One of the DHTs finds that Client B is currently working on the torrent via the infohash you sent.

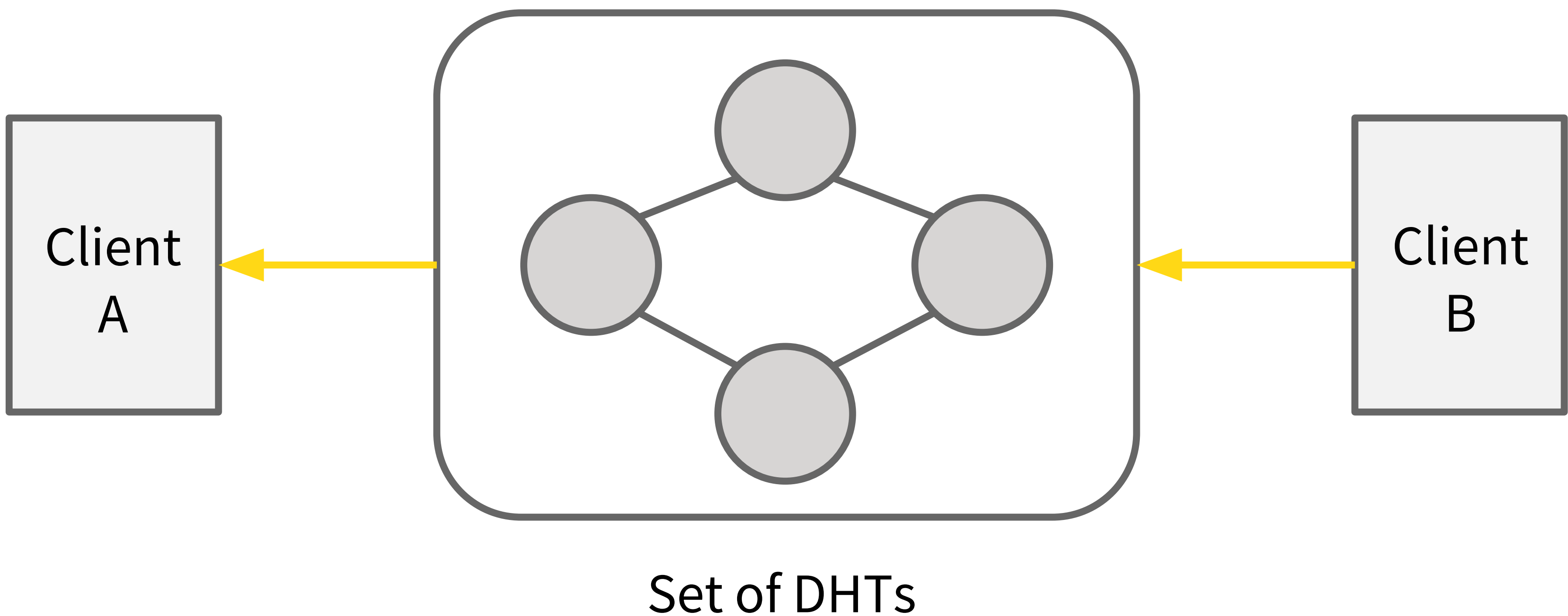


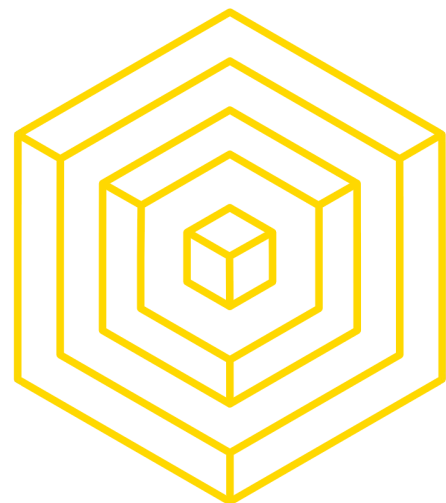


BITTORRENT DHT PEER DISCOVERY

WHAT HAPPENS WHEN DOWNLOAD A TORRENT FILE

Client A is aware of Client B, and Client A reaches out to Client B, and they do a “handshake” and start a file exchange. Note that this is simplified.

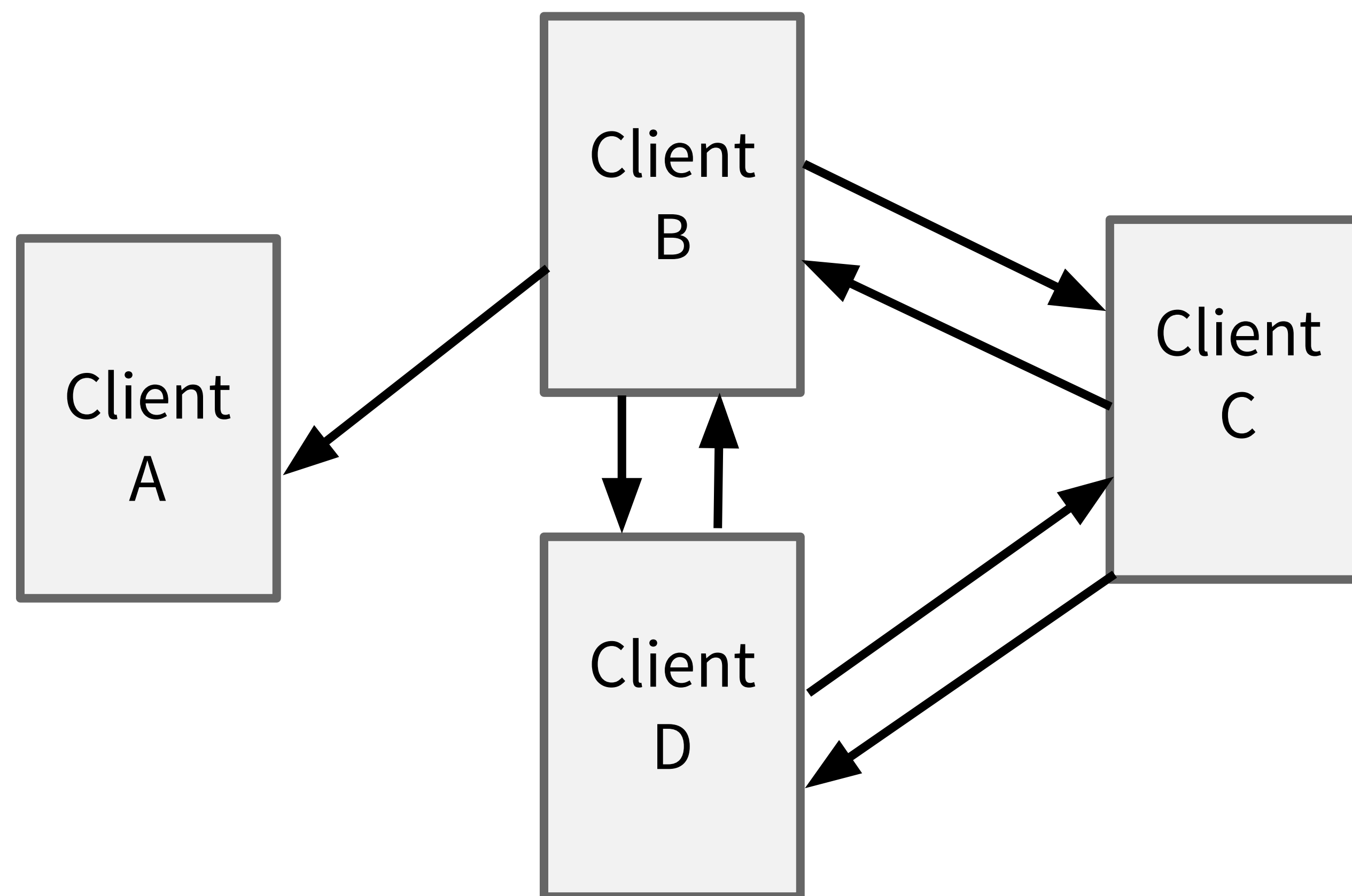




BITTORRENT DHT PEER DISCOVERY

WHAT HAPPENS WHEN DOWNLOAD A TORRENT FILE

Client A can't really contribute yet to Client B and C. But what if Client B gives them more data such that they can contribute to the rest of the network? And then more connections are formed? Then p2p file sharing works quite well!

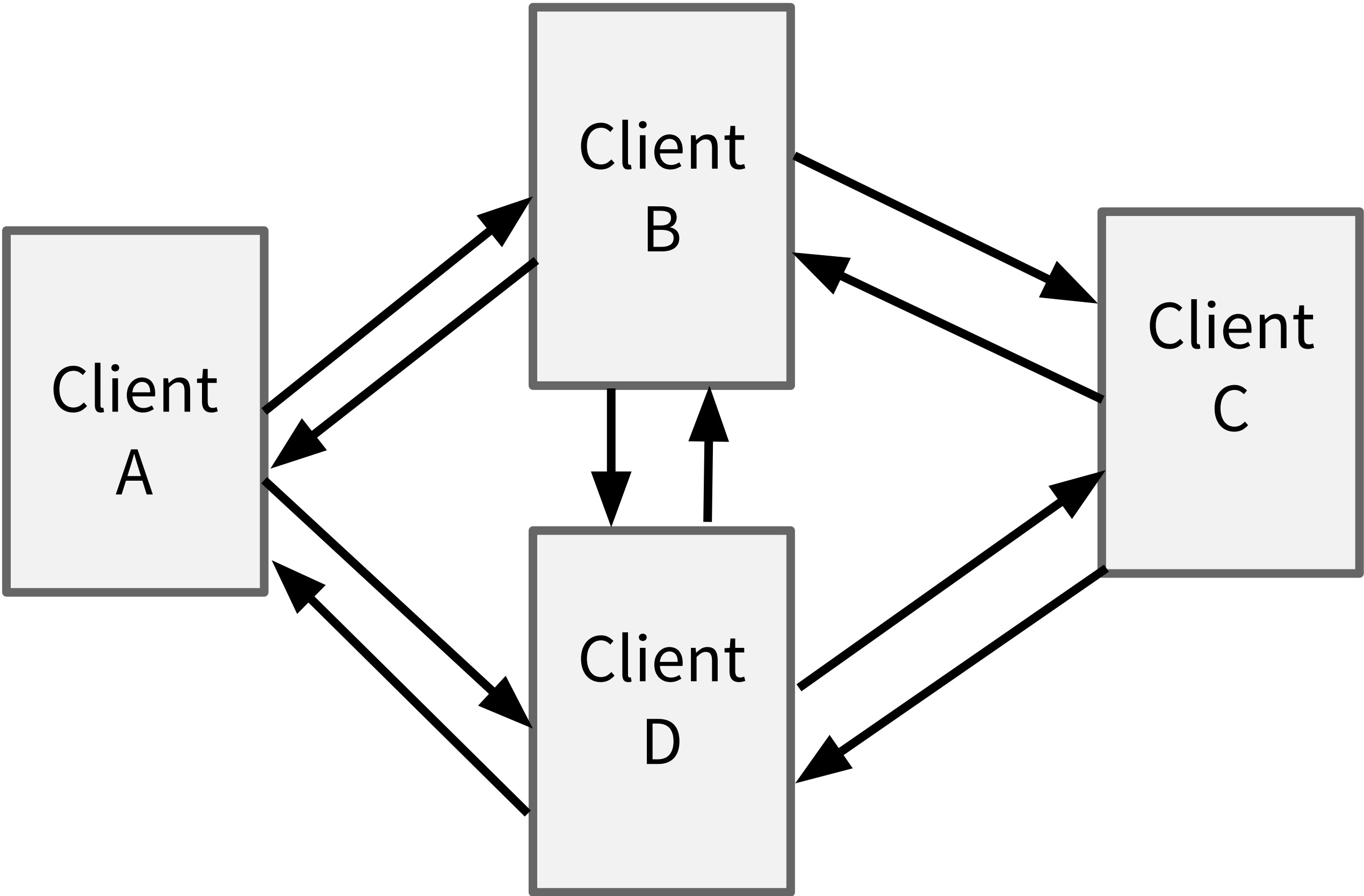




BITTORRENT DHT PEER DISCOVERY

WHAT HAPPENS WHEN DOWNLOAD A TORRENT FILE

Then p2p file sharing works quite well!





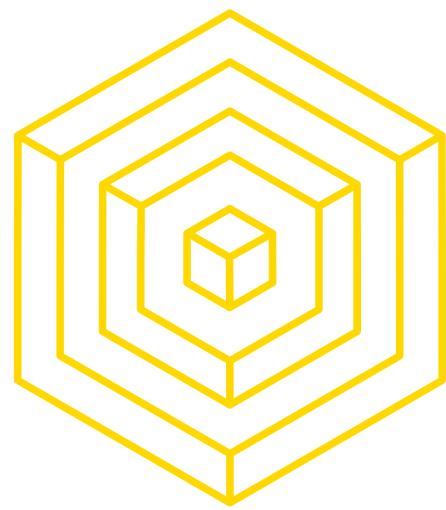
LATER ON

OTHER P2P PROTOCOLS WE WILL TALK ABOUT

- IPFS - Interplanetary File System
- Swarm - ETH Decentralized Storage Protocol

4

DEV P2P



WHAT DOES AN ENODE URL LOOK LIKE

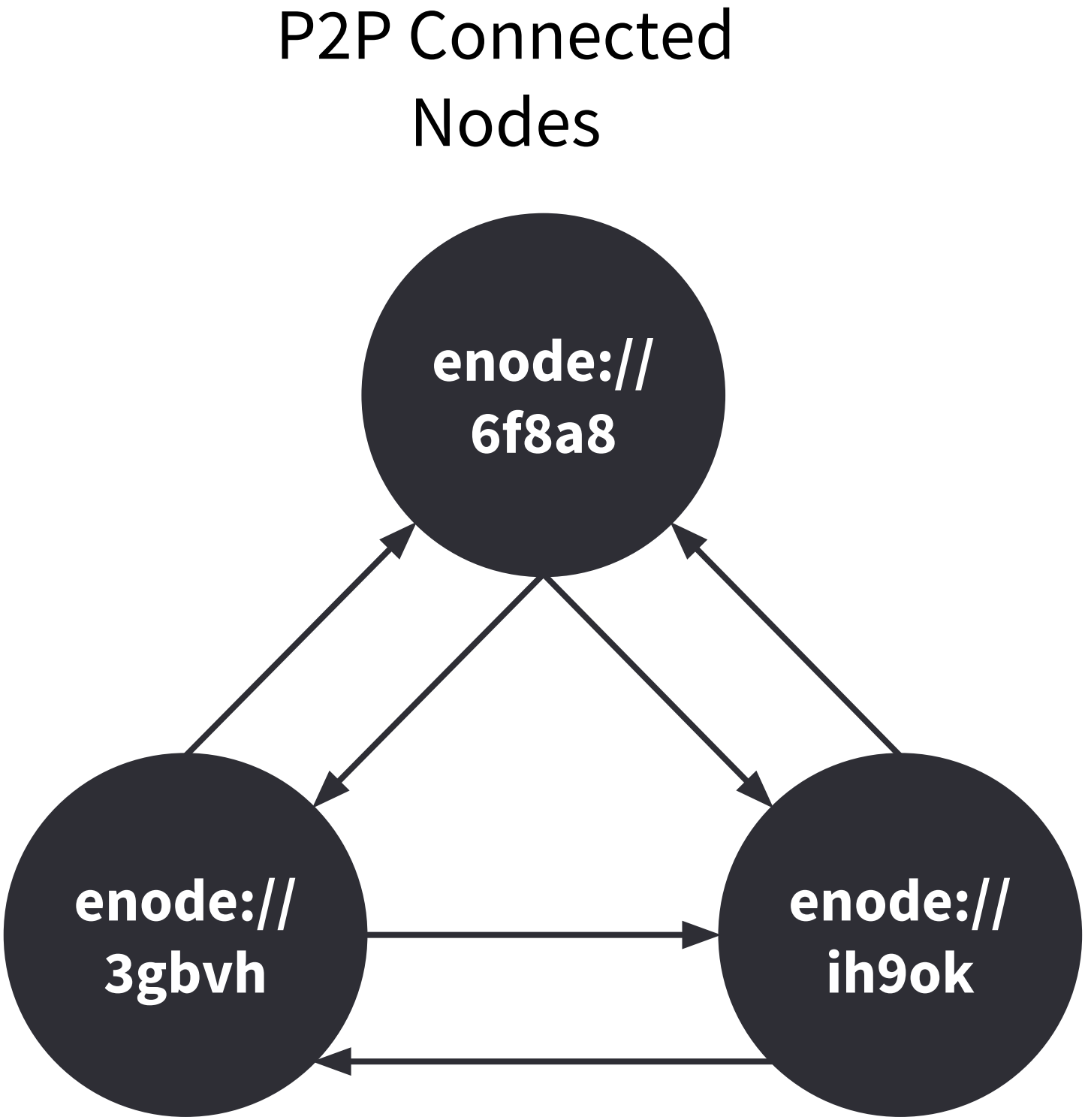
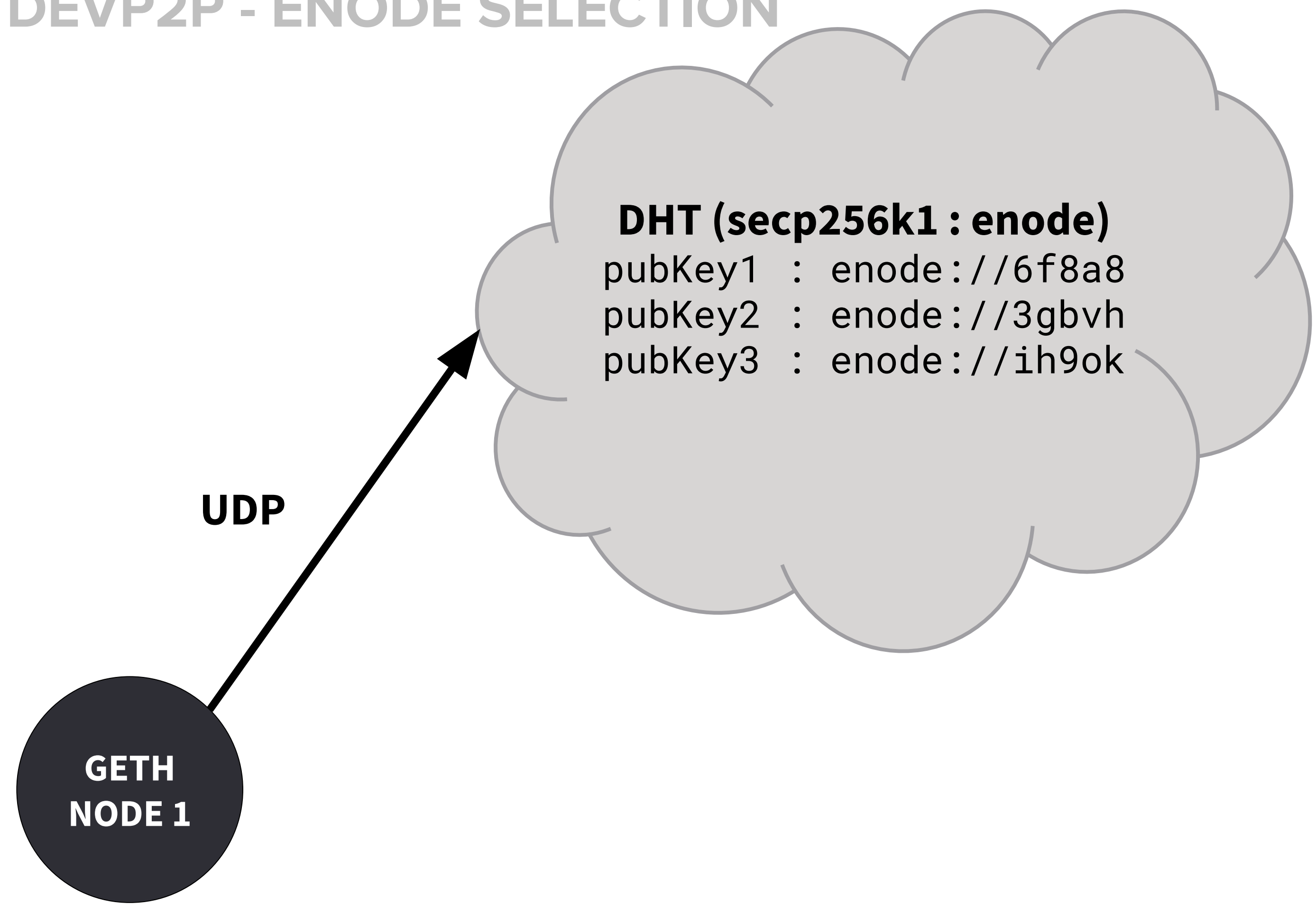
ENODE

- `enode://6f8a80d14311c39f35f516fa664deaaaaa13e85b2f7493f37f6144d86991ec012937307647bd3b9a82abe2974e1407241d54947bbb39763a4cac9f77166ad92a0@10.3.58.6:30303?discport=30301`
- The **hexadecimal node ID** is encoded in the username portion of the URL, separated from the host by an @ sign.
- The **hostname** can only be given as an IP address, DNS domain names are not allowed.
- The **port** in the host name section is the **TCP listening port**.
- If the TCP and UDP (discovery) ports differ, the UDP port is specified as query parameter "**discport**".
- In the following example, the node URL describes a node with IP address 10.3.58.6, TCP listening port 30303 and **UDP discovery port** 30301.



OVERVIEW

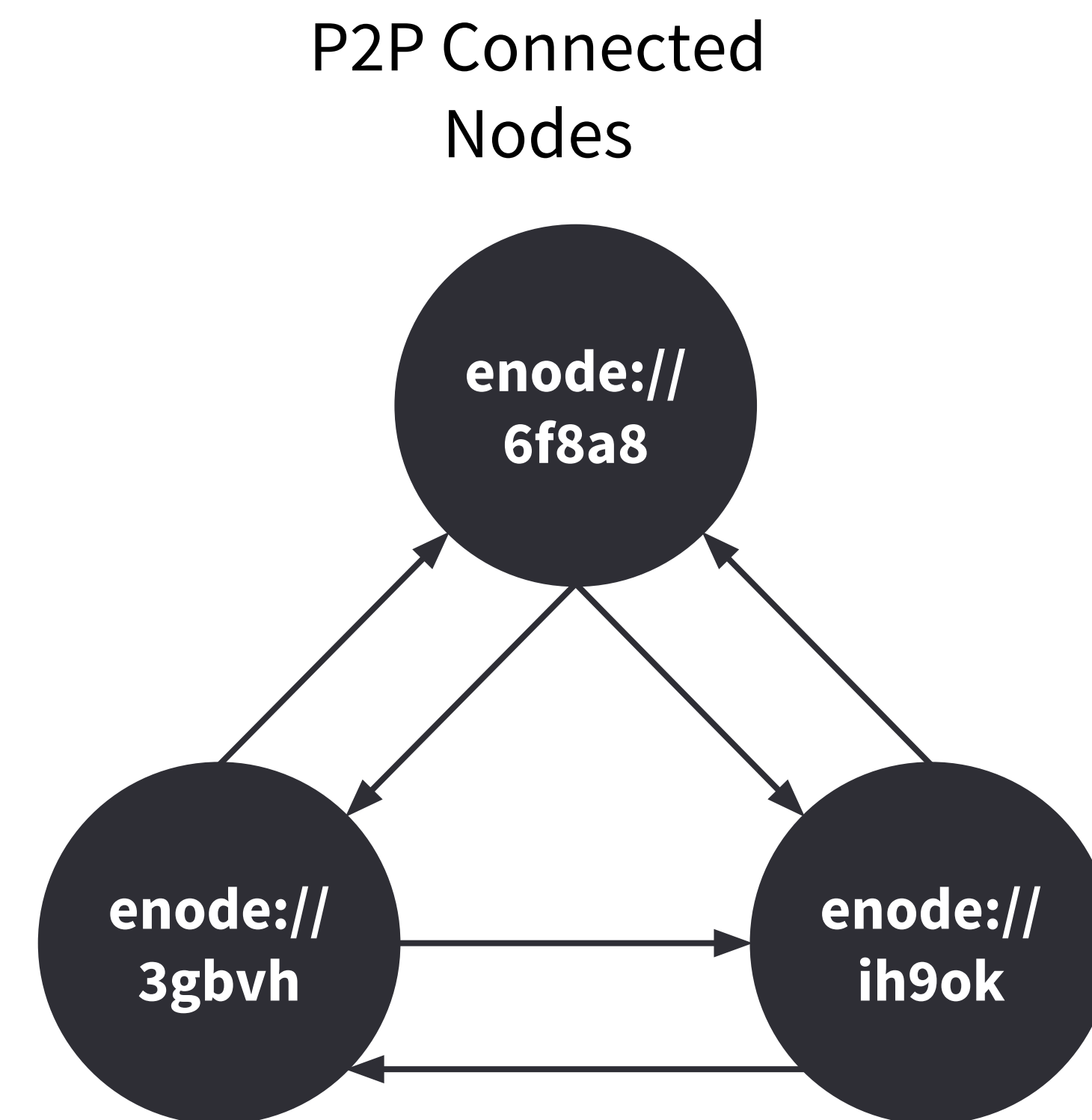
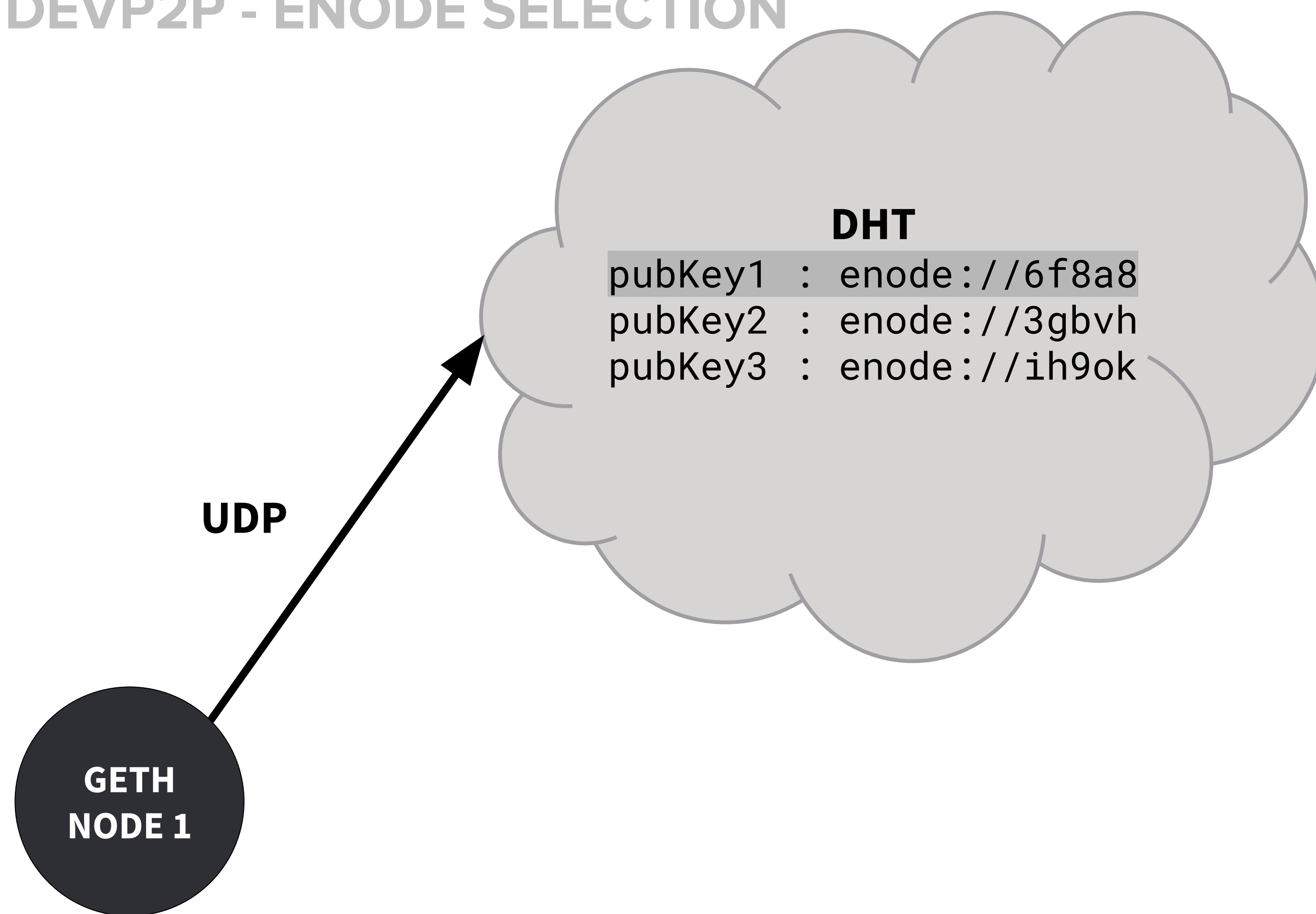
DEVP2P - ENODE SELECTION





OVERVIEW

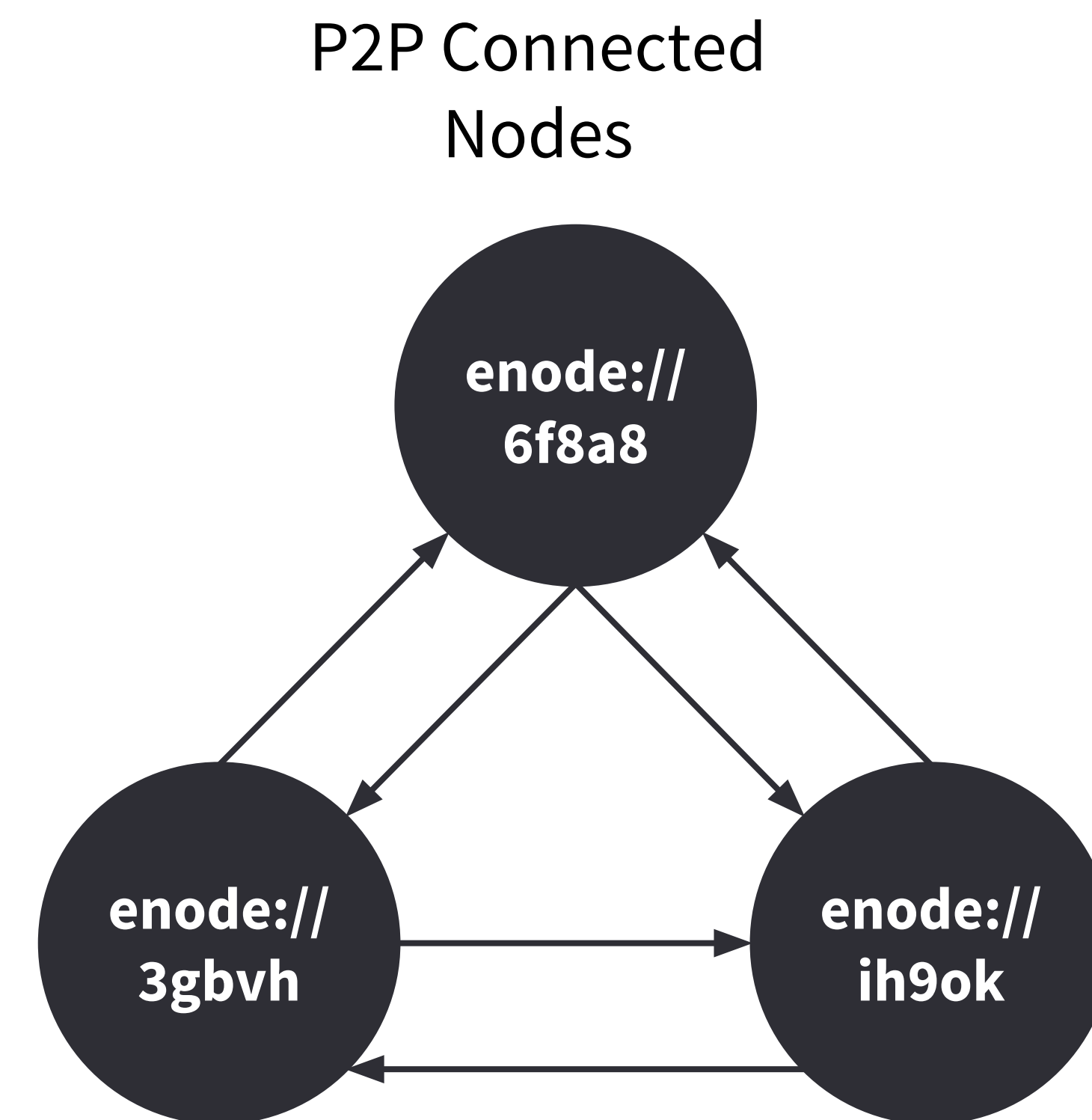
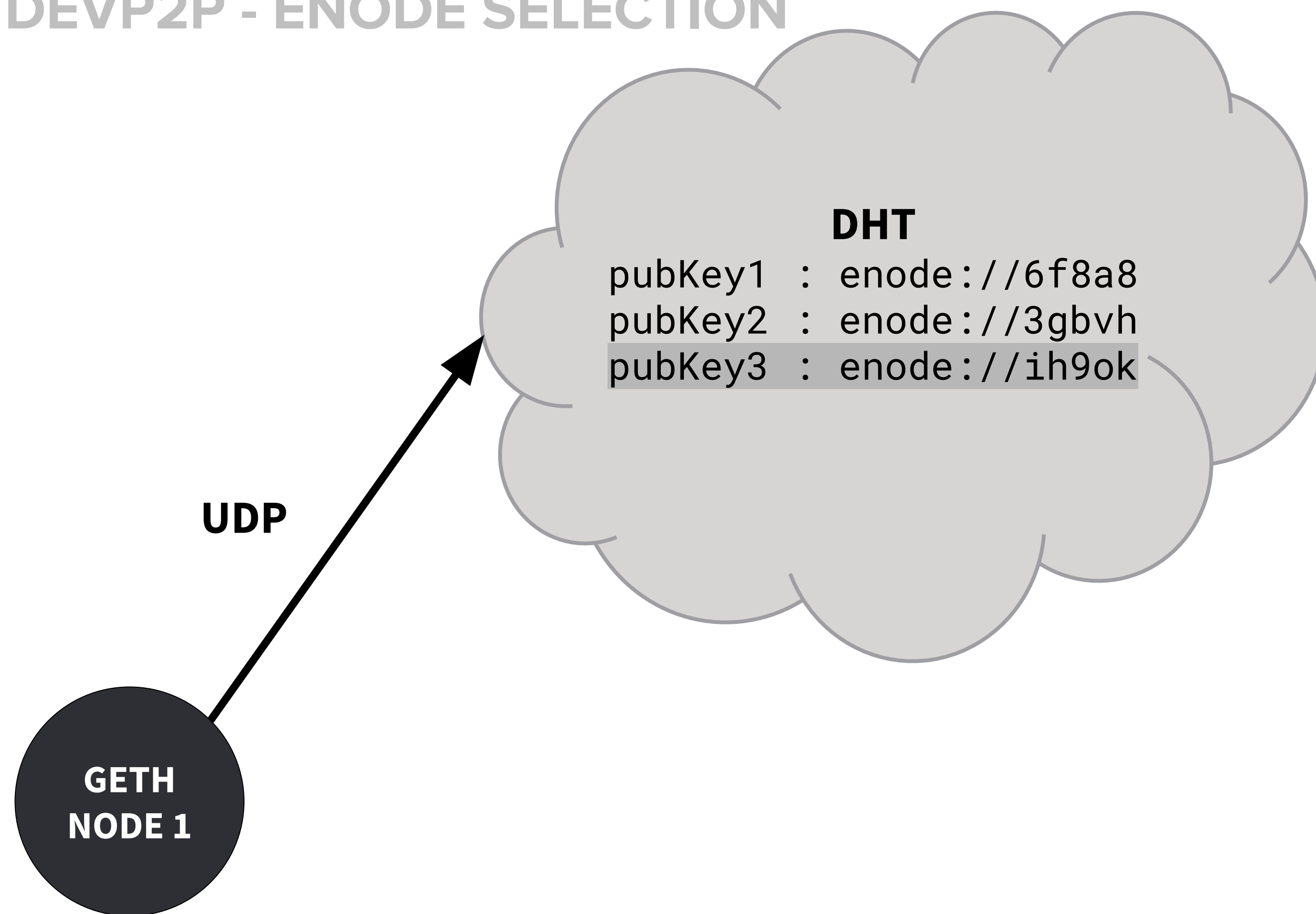
DEVP2P - ENODE SELECTION





OVERVIEW

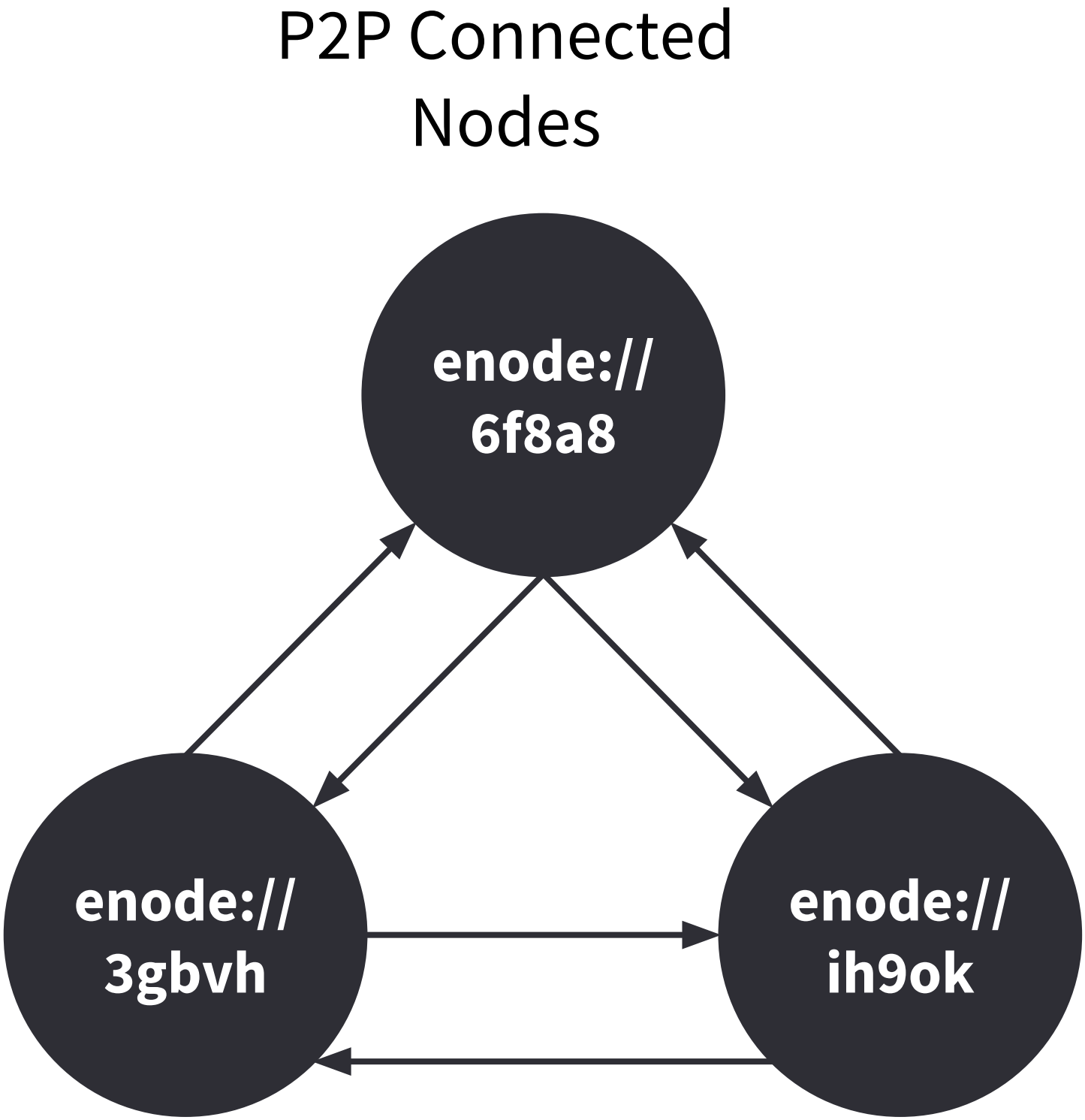
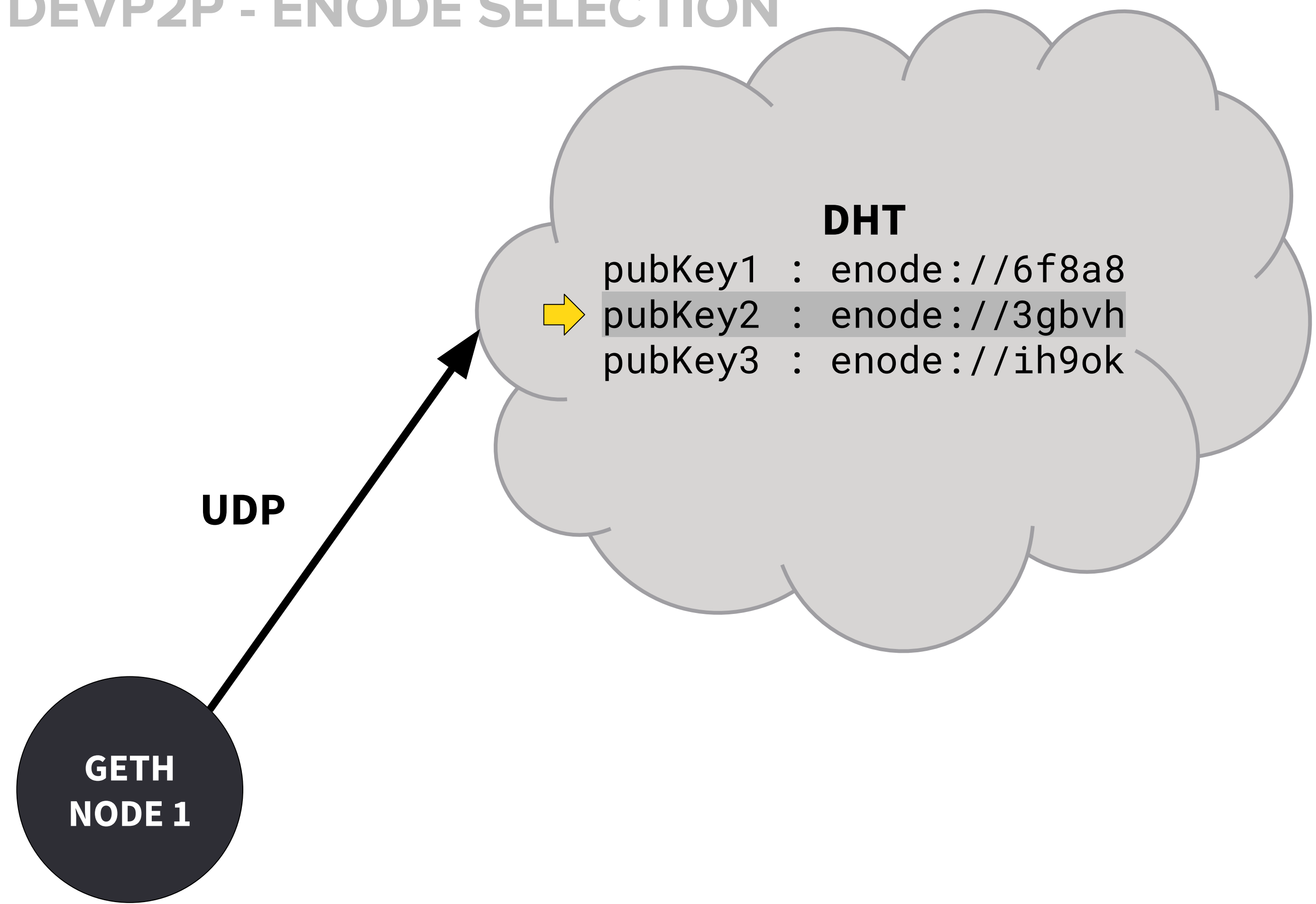
DEVP2P - ENODE SELECTION





OVERVIEW

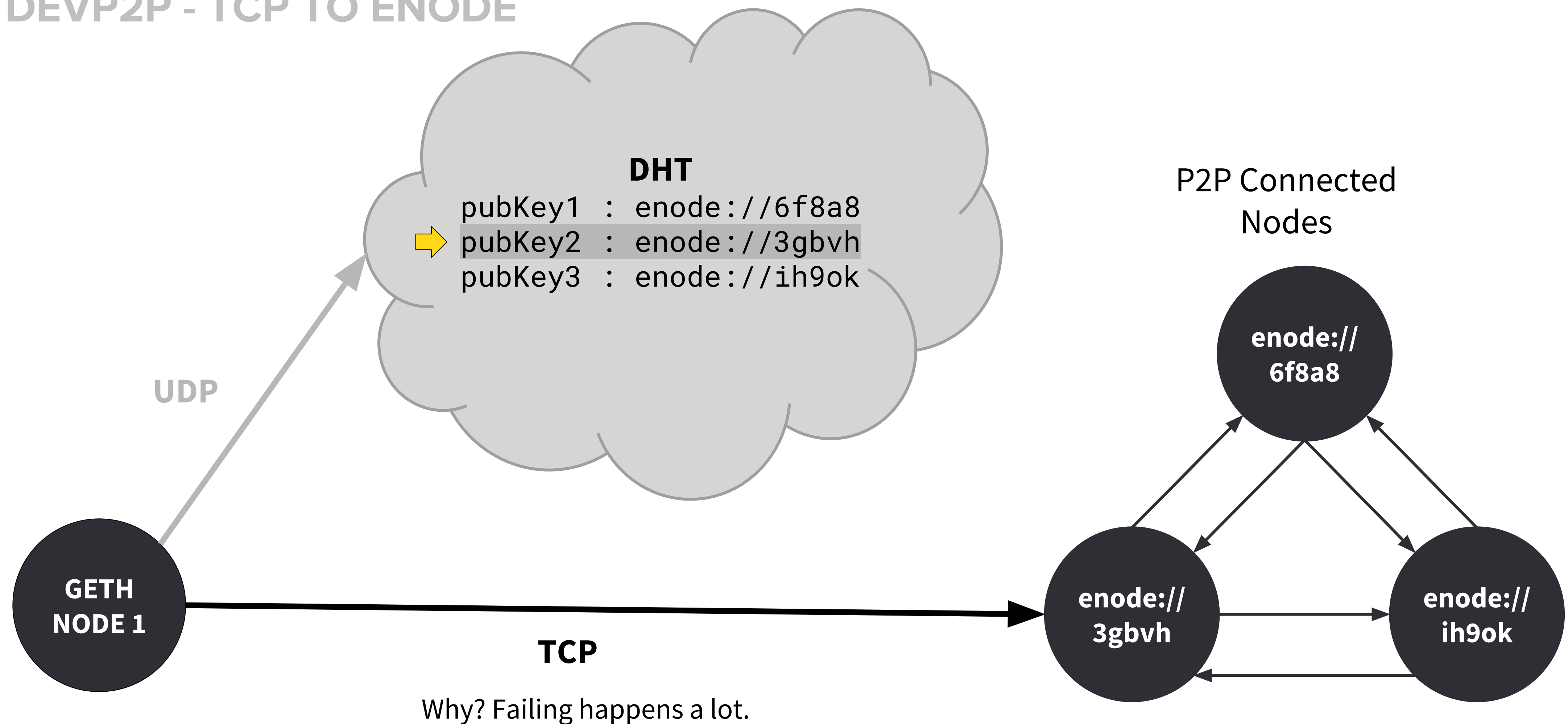
DEVP2P - ENODE SELECTION





OVERVIEW

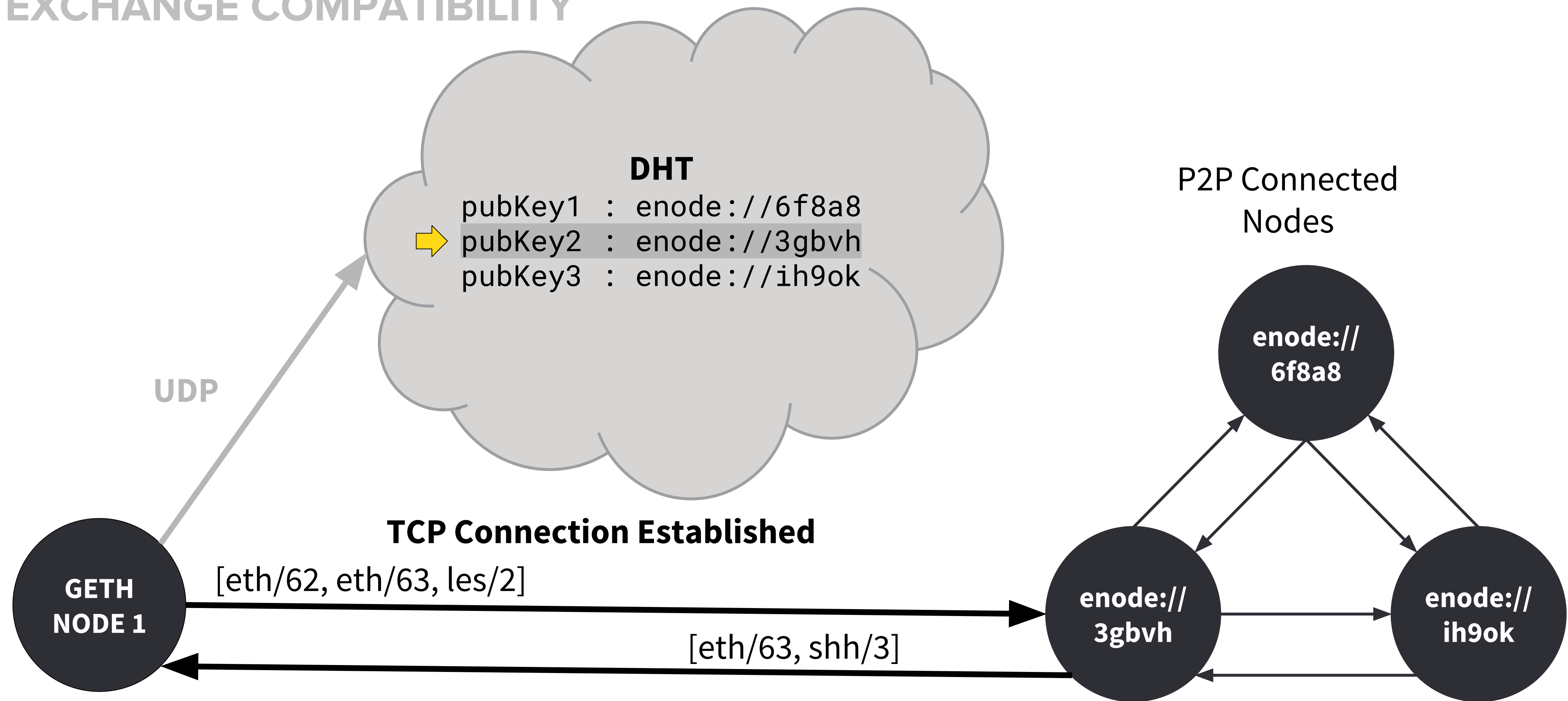
DEVP2P - TCP TO ENODE





OVERVIEW

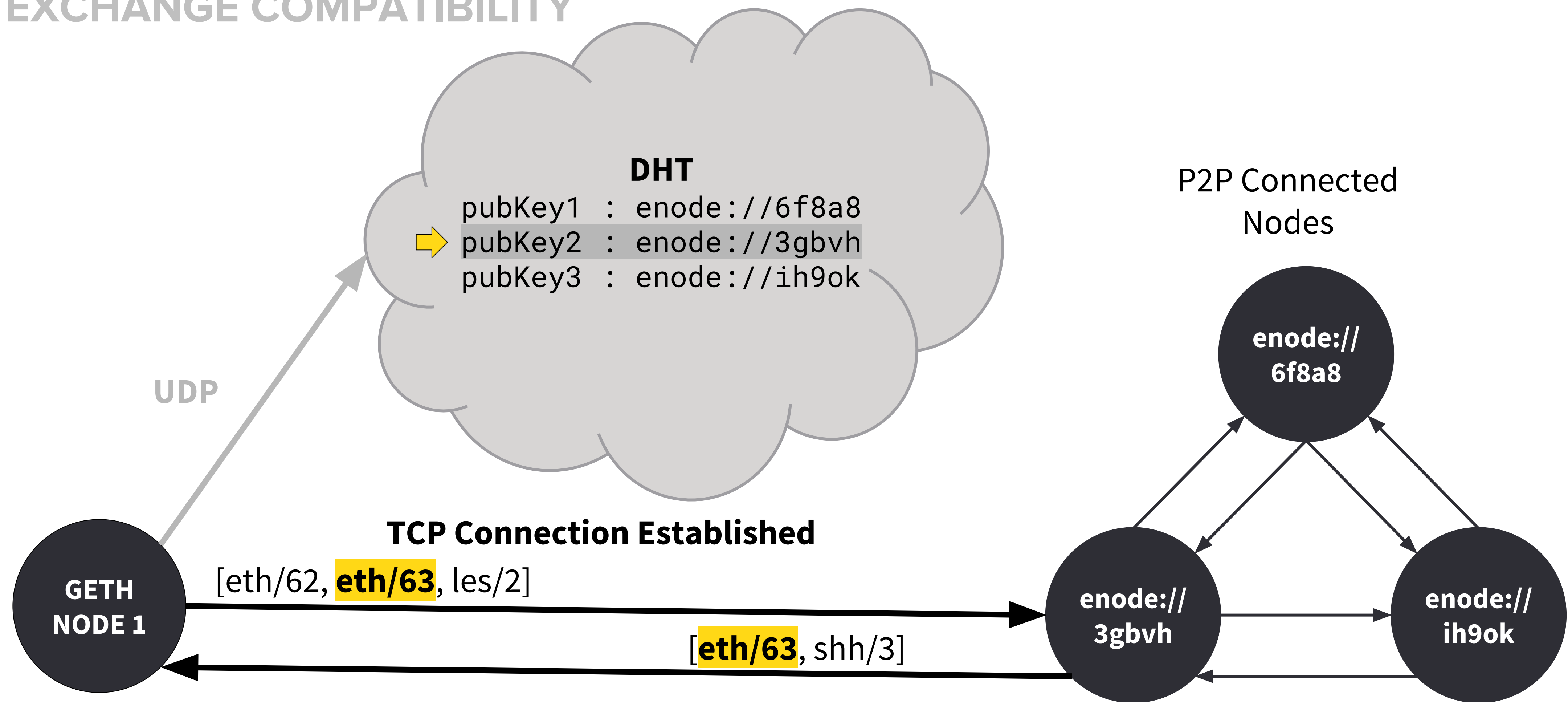
EXCHANGE COMPATIBILITY





OVERVIEW

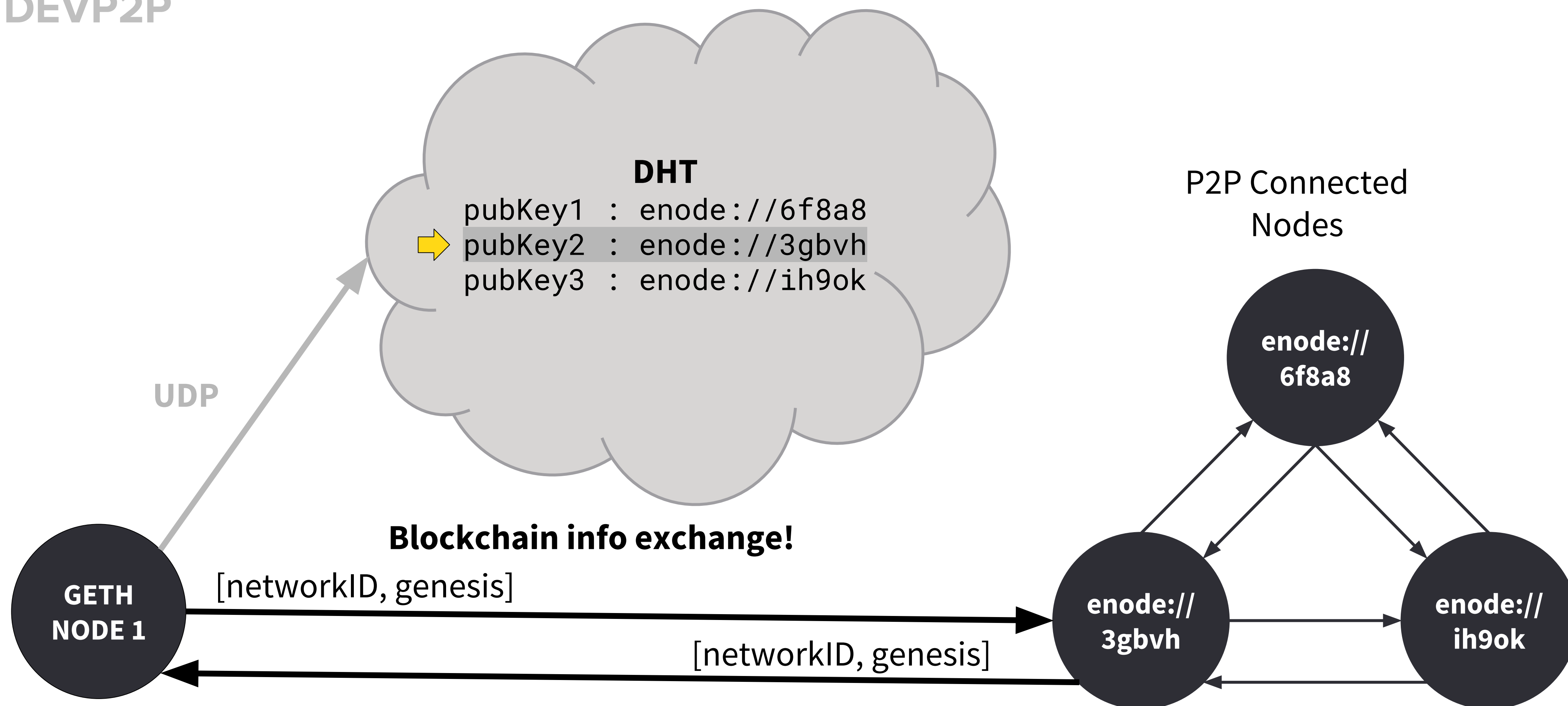
EXCHANGE COMPATIBILITY





OVERVIEW

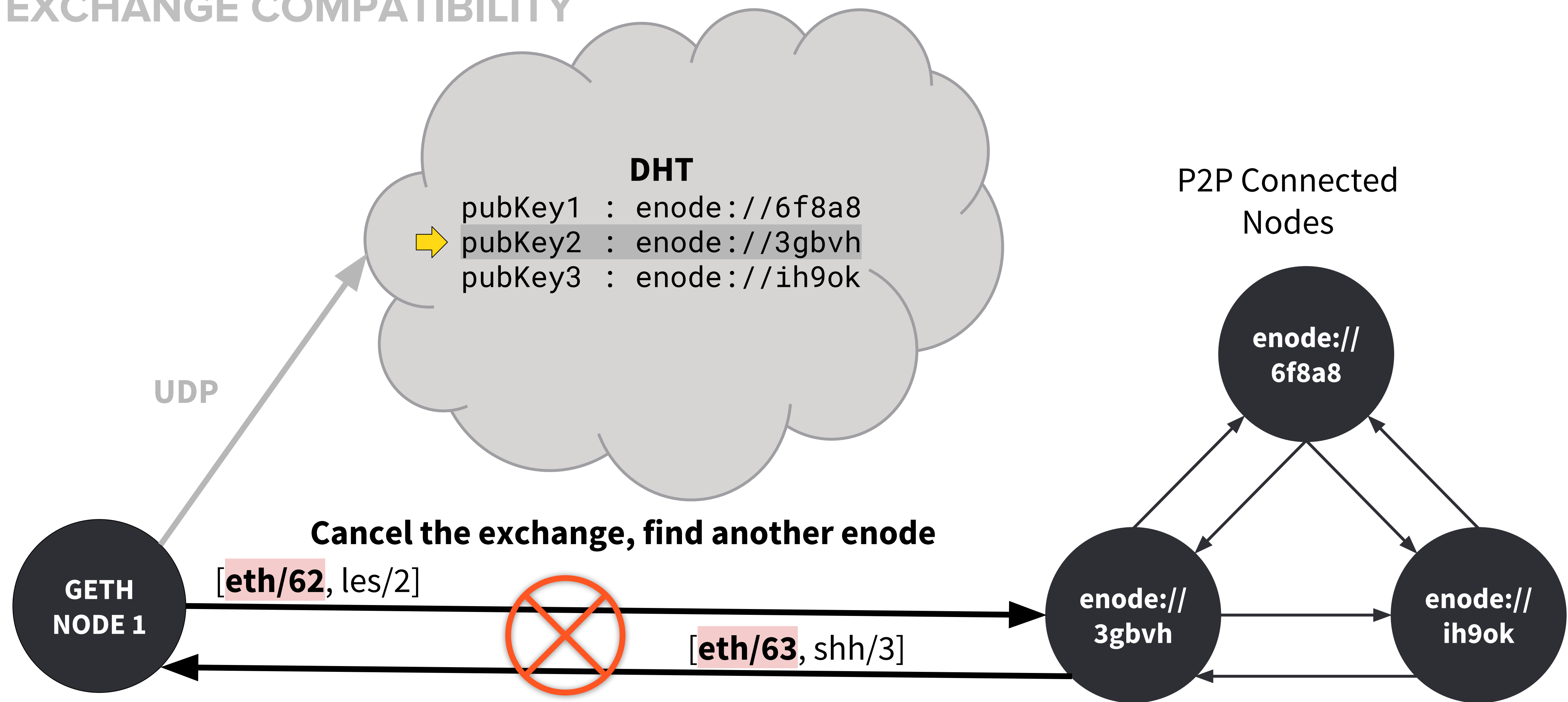
DEVP2P





OVERVIEW

EXCHANGE COMPATIBILITY



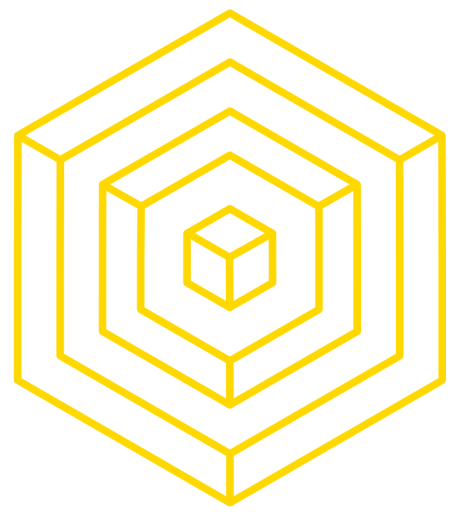


TRANSPORT PROTOCOL

HOW DOES COMMUNICATION BETWEEN NODES WORK

Note: All Ethereum clients work in tandem because they comply with $\mathfrak{DEVp2p}$. Without it, clients can't communicate correctly.

- In order to provide confidentiality and protect against network disruption, $\mathfrak{DEVp2p}$ nodes use **RLPx** messages, an encrypted and authenticated transport protocol
- **RLPx** utilizes **Kademlia**-like routing
 - Node discovery
 - Network formation
 - Multiple protocols over a single connection



KADEMLIA DHT

WHAT ETHEREUM'S P2P INFO SYSTEM IS BASED ON

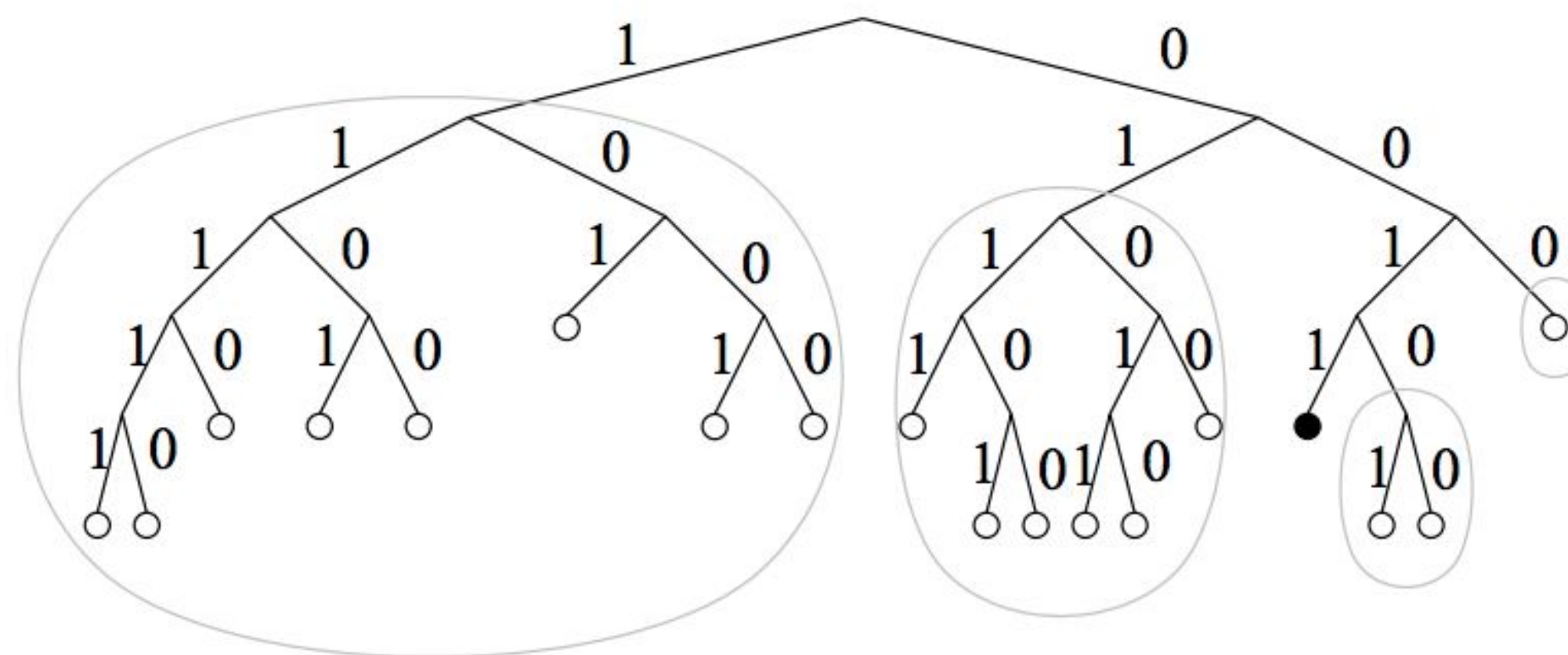
- A peer to peer, distributed hash table, great for discovery
- Configuration information spreads automatically as a side effect of key lookups
- Nodes have enough knowledge and flexibility to route queries through low latency paths
- Uses parallel, async queries to avoid timeout delays from failed nodes
- When searching for some value, the algorithm takes a key and explores the network in several steps.
 - Each step will find nodes that are closer to the key until the contacted node returns the value or no more closer nodes are found
 - Contacts only **$O(\log(n))$** nodes during the search out of a total of n nodes in the system.



KADEMLIA DHT

WHAT ETHEREUM'S P2P INFO SYSTEM IS BASED ON

- Nodes have **160 bit opaque IDs** and we want a lookup algorithm that **locates successively closer nodes to any desired ID**, converging to the lookup target in **logarithmic** steps.
- $\langle \text{key}, \text{value} \rangle$ pairs are stored on the nodes with IDs “close” to the key for some notion of closeness (i.e. latency)
- The protocol treats nodes as leaves in a binary tree, each node's position determined by the shortest unique prefix of its ID





KADEMLIA DHT

WHAT ETHEREUM'S P2P INFO SYSTEM IS BASED ON

- **The protocol ensures that every node knows of at least one node in each of each subtrees!**
- See node with prefix 0011 => for any node we divide the tree into a series of successively lower subtrees that don't contain the node.
- Highest subtree consists of half the binary tree not containing the node
- The next subtree consists of half of the remaining tree not containing the node, so on and so forth

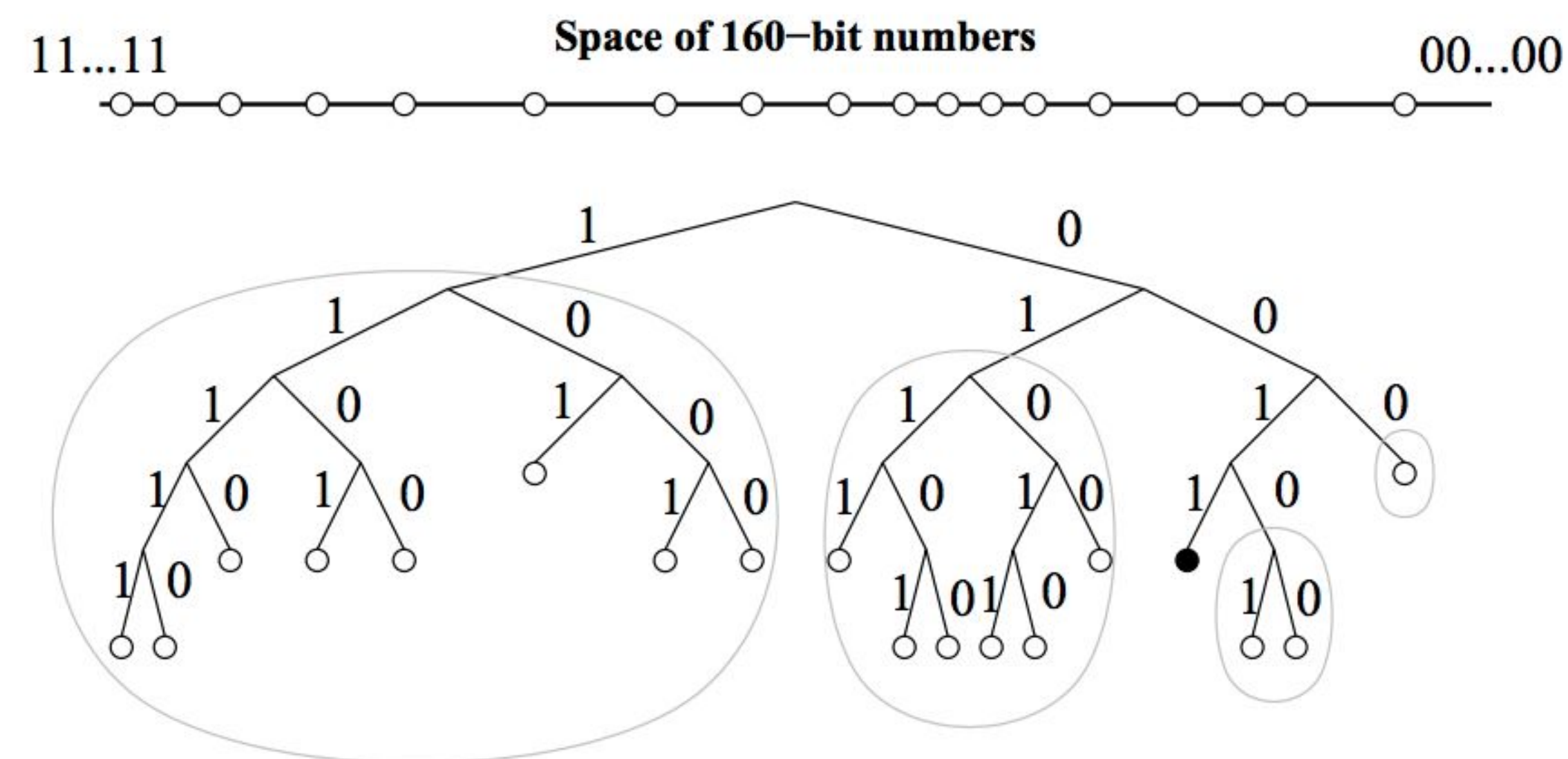


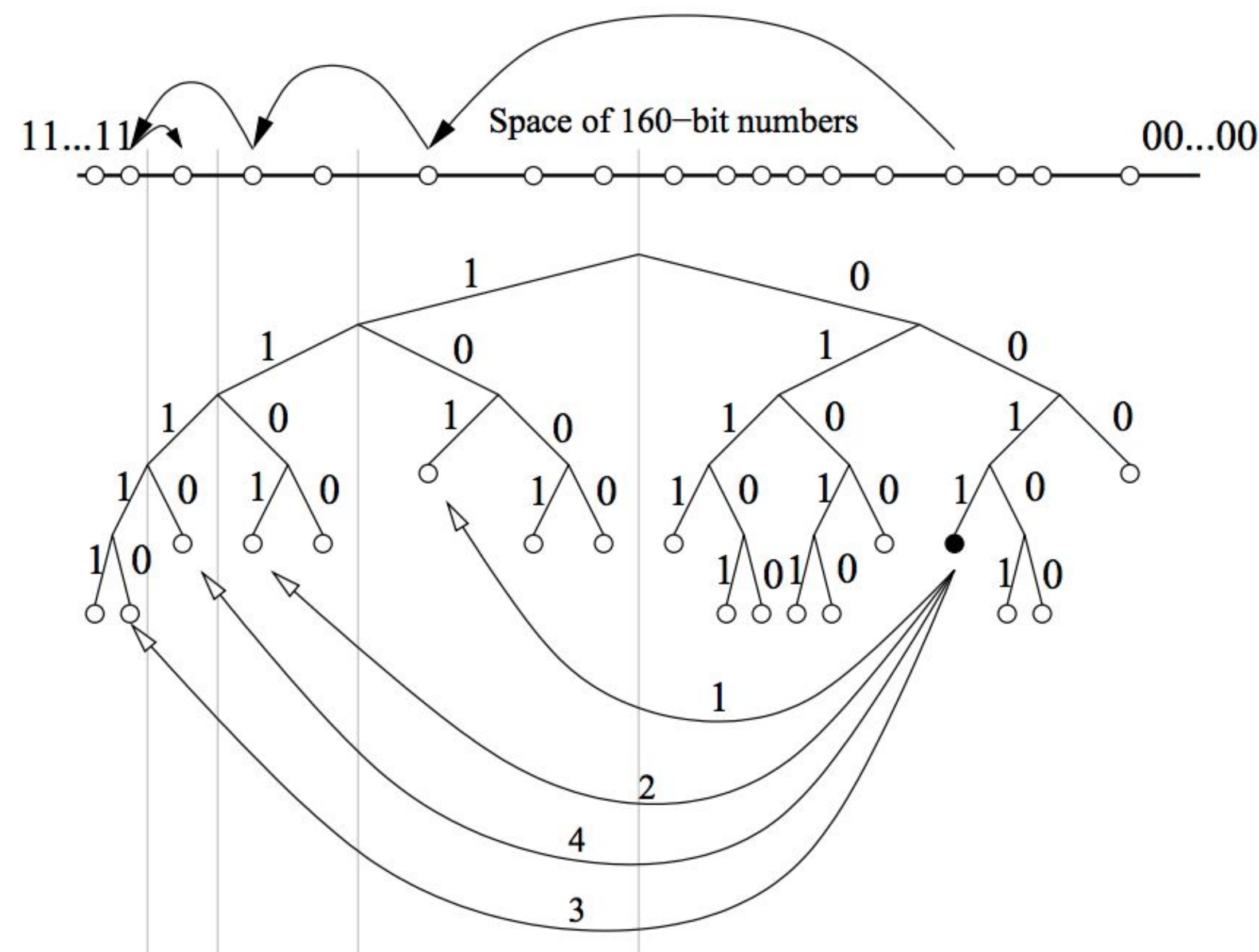
Fig.1: Kademlia binary tree. The black dot shows the location of node 0011... in the tree. Gray ovals show subtrees in which node 0011... must have a contact.



KADEMLIA DHT

WHAT ETHEREUM'S P2P INFO SYSTEM IS BASED ON

- **How do we locate other nodes by ID?**
 - Node with prefix 0011 finds node with prefix 1110 by successively learning of and querying closer and closer nodes
- Nodes have 160 bits IDs, and the keys for the KV pairs are also 160-bit IDs.
 - Because Kademlia relies on distance between two IDs, the distance is defined as: $d(x, y) = x \oplus y$ (**XOR**)

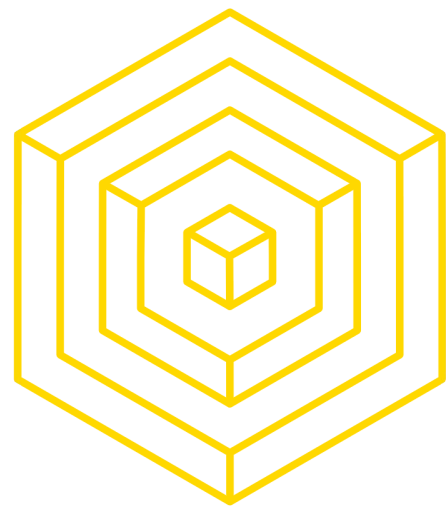




KADEMLIA DHT

WHAT ETHEREUM'S P2P INFO SYSTEM IS BASED ON

- **Why does XORing key and node id work?**
- In a fully populated binary tree of 160 bit IDs, the magnitude of distance between IDs is the height of the smallest subtree containing them both
- When a tree is not fully populated, the closest leaf to and ID **x** is the leaf who's ID shares the **longest common prefix** (LCP) of **x**
 - If there are empty branches, there might be more than one leaf with the LCP
 - In that case, the closest leaf to **x** will be the closest leaf to ID **x** produced by flipping the bits in **x** corresponding to the empty branches of the tree



KADEMLIA DHT

HOW ABOUT ROUTING TO OTHER NODES

- Each node keeps k-buckets in its routing table to store the peer node information (16 in Ethereum)
- Learn more about the routing tables [here](#) and how k-buckets works (maybe for lab!)



RLP ENCODING

HOW TO SERIALIZE DATA

- The purpose of **RLP (Recursive Length Prefix)** is to encode arbitrarily nested arrays of binary data, and RLP is the main encoding method used to serialize objects in Ethereum
- The only purpose of RLP is to encode structure; encoding specific data types (eg. strings, floats) is left up to higher-order protocol - there's also a way to decode data to use it
- Starts off with **byte value (0x80) + the length of the string**, and sometimes a **byte value (0xc0) + length of the list**
- Examples:
 - The string "dog" = [0x83, 'd', 'o', 'g']
 - The list ["cat", "dog"] = [0xc8, 0x83, 'c', 'a', 't', 0x83, 'd', 'o', 'g']

Worth reading the spec: <https://github.com/ethereum/wiki/wiki/RLP>



TRANSPORT PROTOCOL

HOW DOES COMMUNICATION BETWEEN NODES WORK

- When **DEVp2p** nodes communicate, they use **TCP** via the Internet
- But on top of that are the messages that are defined by **RLPx**, allowing them communicate the sending and receiving of packets
- Packets are dynamically framed, prefixed with an **RLP encoded** header, encrypted and authenticated
- **Multiplexing** is achieved via the **frame header** which specifies the **destination protocol** of a packet
 - Think about how packets might be received asynchronously (benefits of UDP's asynchronous transfer with TCP's reliability)



TRANSPORT PROTOCOL

WHAT IS MULTIPLEXING?

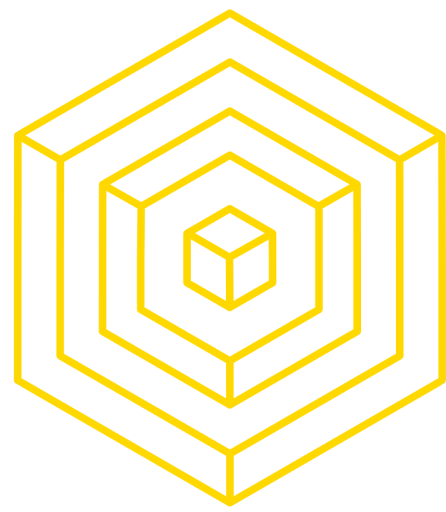
- In **statistical multiplexing**, a communication channel is divided into an arbitrary number of variable bitrate digital channels or **data streams**
- Each stream is divided into packets that normally are delivered asynchronously in a first-come first-served fashion
 - In alternative fashion, the packets may be delivered according to some scheduling discipline for fair queuing or differentiated and/or guaranteed quality of service
- Normally implies "on-demand" service rather than one that preallocates resources for each data stream



TRANSPORT PROTOCOL

ENCRYPTED HANDSHAKE

- Connections are established via cryptographic handshake, and once established, packets are encrypted and encapsulated as frames
- **Phase 1:** Peer authentication with an encryption handshake
 - **Peer authentication:** to establish a secure communication channel by setting up an encrypted, authenticated message stream
 - **Encryption handshake:** to exchange temporary keys to set initial values for this secure session
- **Phase 2:** The base protocol handshake
 - Negotiate supported protocols, checking versions and network IDs



TRANSPORT PROTOCOL

SECURE CONNECTIONS

114

Side Note: If the connection was initiated by a peer, we say that they are the **initiator**, and the other peer is **receiver**. The word **remote** is used to describe the 'other' peer in a connection when talking from a point of view of a node.

Creating a secure connection consists of the following steps:

1. **Initiator** sends an authentication message to **receiver**
2. **Receiver** responds with an authentication response message and sets up a secure session
3. **Initiator** checks receiver's response and establishes a secure session
4. **Receiver** and **Initiator** then send base protocol handshake on the established secure channel

Either side may disconnect if authentication fails or if the protocol handshake isn't appropriate.



TRANSPORT PROTOCOL

WHAT IF FAILURE?

115

Side Note: The other distinction is whether the remote peer is **known** or **new**. A **known** peer is one which has previously been connected and for which a corresponding session token is remembered.

- If the handshake fails, if and only if initiating a connection TO a known peer, then the nodes information should be removed from the node table and the connection **MUST NOT** be reattempted.
- Due to the limited IPv4 space and common ISP practices, this is likely a common and normal occurrence, therefore, no other action should occur.
- If a handshake fails for a connection which is received, no action pertaining to the node table should occur.



TRANSPORT PROTOCOL

WHAT IF SUCCESS?

116

Side Note: The other distinction is whether the remote peer is **known** or **new**. A **known** peer is one which has previously been connected and for which a corresponding session token is remembered.

- If the handshakes succeed, the fixed array of protocols supported by both peers will run on the connection parallelly to send and receive messages
- Once established, packets are encapsulated as frames which are encrypted using **AES-256 in CTR mode**
 - Initial values for the message authentication and cipher are never the same
 - Key material for the session is derived via a KDF (key derivation function) and ECDHE-derived (Elliptic-curve Diffie–Hellman) shared-secret
 - ECC uses secp256k1 curve (ECP).
- It is the purpose of the encryption handshake to negotiate these key values for a new secure session

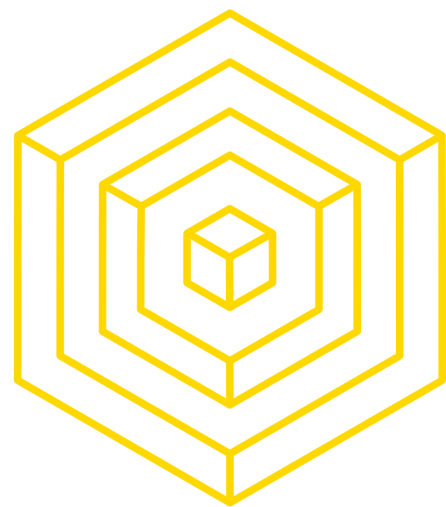
Read more here: <https://github.com/ethereumproject/go-ethereum/wiki/RLPx-Encryption>



TRANSPORT PROTOCOL

WHAT HAPPENS WITH TRANSACTIONS

- **DEVp2p** has a capabilities list as we saw in the earlier diagrams.
- With that two peers can decide to communicate with the ETH protocol capability
- The ETH protocol describes the forwarding of blocks and transactions
- If a user creates a transaction (that interacts with a smart contract) the transaction will be forwarded to all nodes/miners
- The miners will include it in a valid block and the nodes will then verify the block (including the execution of transactions) forward that block to every other node and add it to the chain



PROBLEMS

WHY DOES DEVP2P KINDA SUCK

- So many roundtrips just to figure out if someone is on the right blockchain
- Upgrades need tight coordination - the whole system is frozen - everything needs to be backward compatible
- Achieving upgrades have been tied to mainnet hard forks
- Everyone has to upgrade their own nodes and only speak their own protocol
- We can't make changes on an accelerated schedule
 - Heavily tied to RLPx protocol, secp256k1, keccak256

SEE YOU NEXT TIME

Down the Stack
Ethereum Storage
Ethereum Mining
Block Related Protocols