Algorithms and Distributed Systems 2019/2020 (Lecture Three)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Lecture structure:

- Unstructured Overlay Networks Usage
- Resource Location Problem (Exact Match Queries)
- Structured Overlay Networks.
- Partially Structured Overlay Networks.

Unstructured Overlay Networks

- Main properties:
  - Random topology.
  - Low maintenance cost.
  - Eventual Global Connectivity.

Unstructured Overlay Networks

- Main properties:
  - Random topology.
  - Low maintenance cost.
  - Eventual Global Connectivity.
- Why is this useful?

Unstructured Overlay Networks

- Main properties:
  - Random topology.
  - Low maintenance cost.
  - Eventual Global Connectivity.
- Why is this useful?
  - Message dissemination (Broadcast).

Unstructured Overlay Networks

- Main properties:
  - Random topology.
  - Low maintenance cost.
  - Eventual Global Connectivity.
- Why is this useful?
  - Message dissemination (Broadcast).
  - Replication of data across large number of nodes.
  - Monitoring.
  - Resource Location.

Unstructured Overlay Networks

- Main properties:
  - Random topology.
  - Low maintenance cost.
  - Eventual Global Connectivity.
- Why is this useful?
  - Message dissemination (Broadcast).
  - Replication of data across large number of nodes.
  - Monitoring.
  - **Resource Location.**

Resource Location

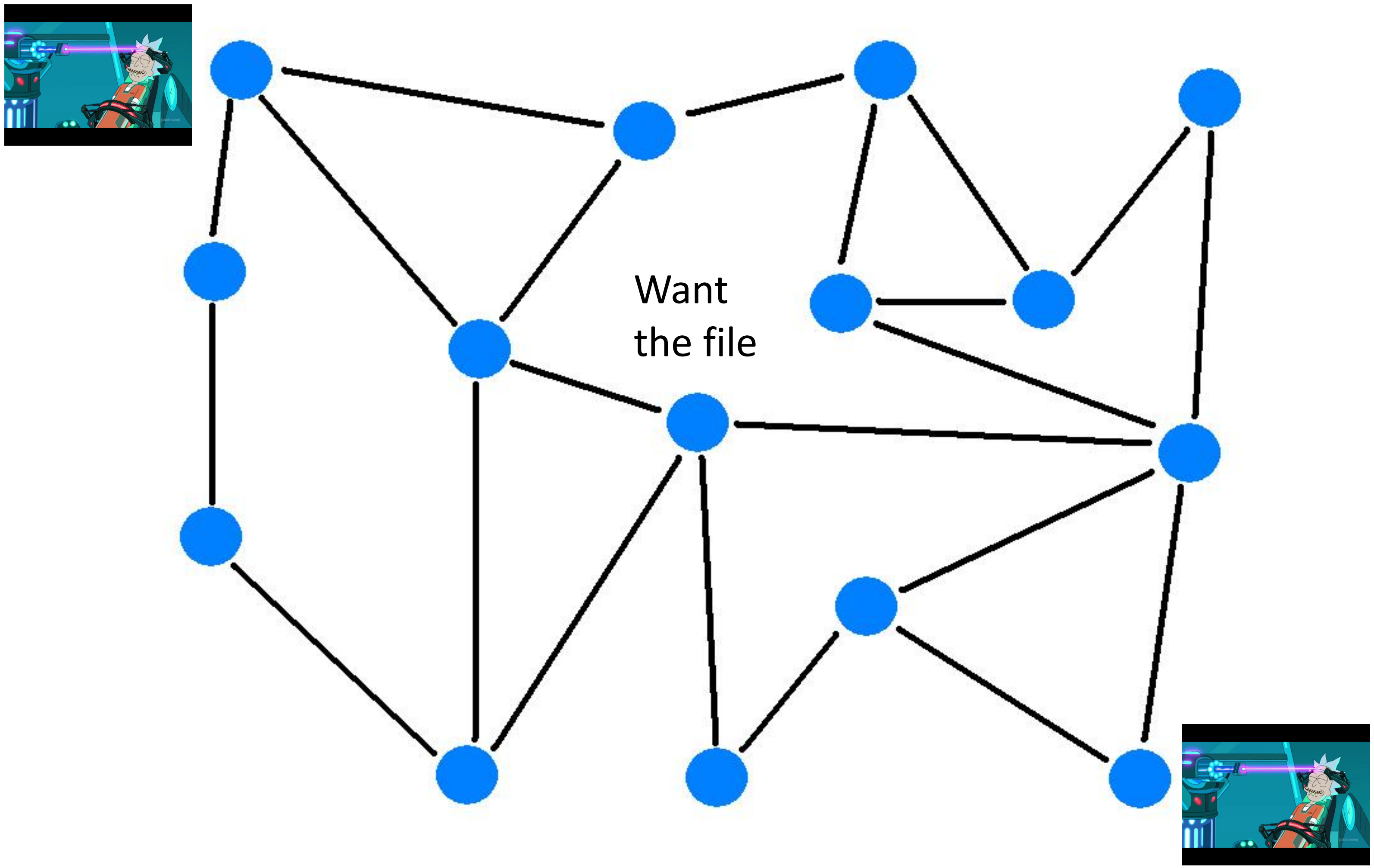
- A Definition:

Given a set of processes containing different sets of resources, locate the processes that contain resources with a given set of properties.

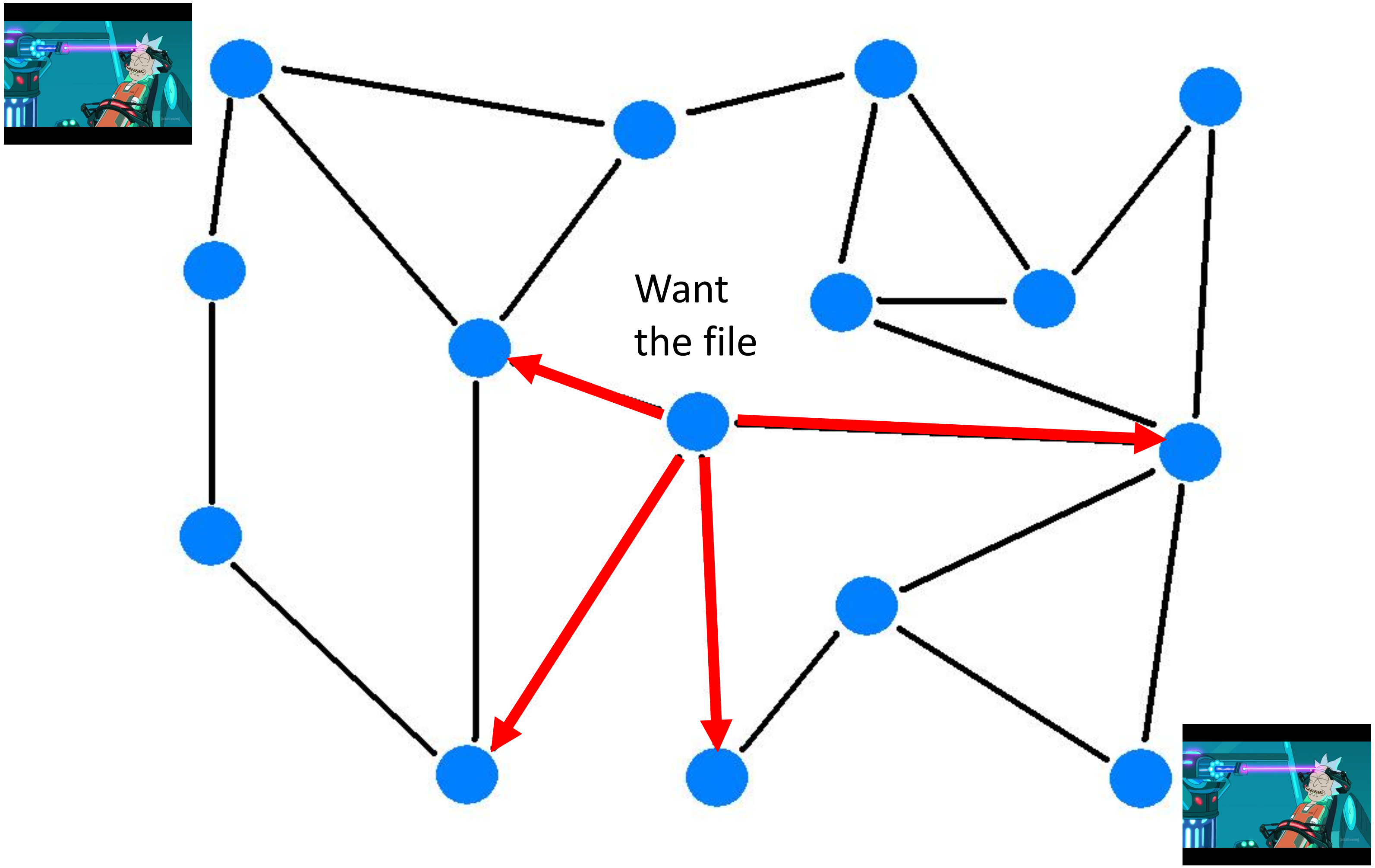
- One possible concretization:
  - File Sharing Applications.
  - Processes own a set of files that have properties.
  - Locate the processes (and files) that match a given set of criteria (e.g. Extension = mkv, size >= 1Gb, Name contains “Rick and Morty” and “S03E01”.

How do we solve the resource location problem?

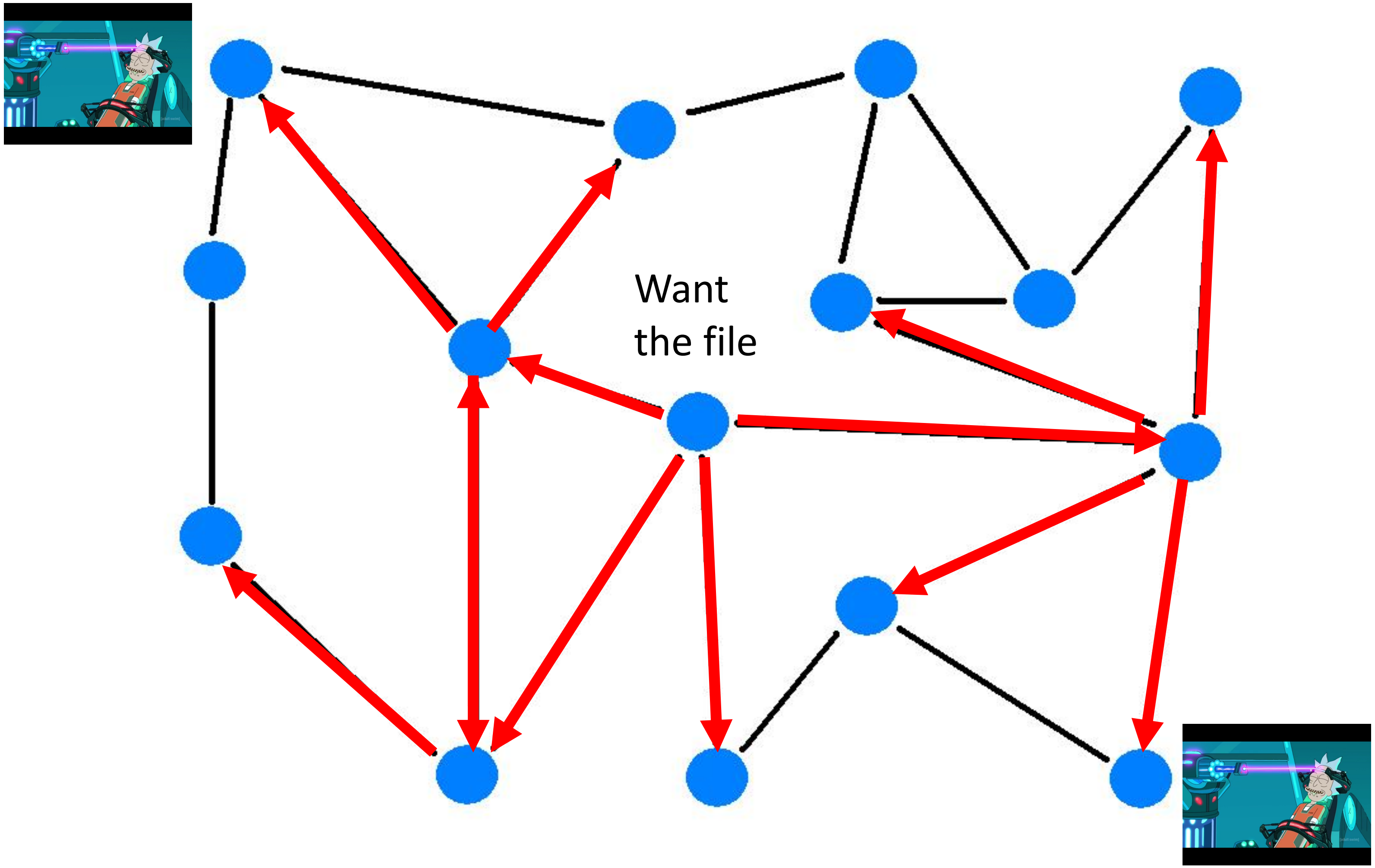
How do we solve the resource location problem?



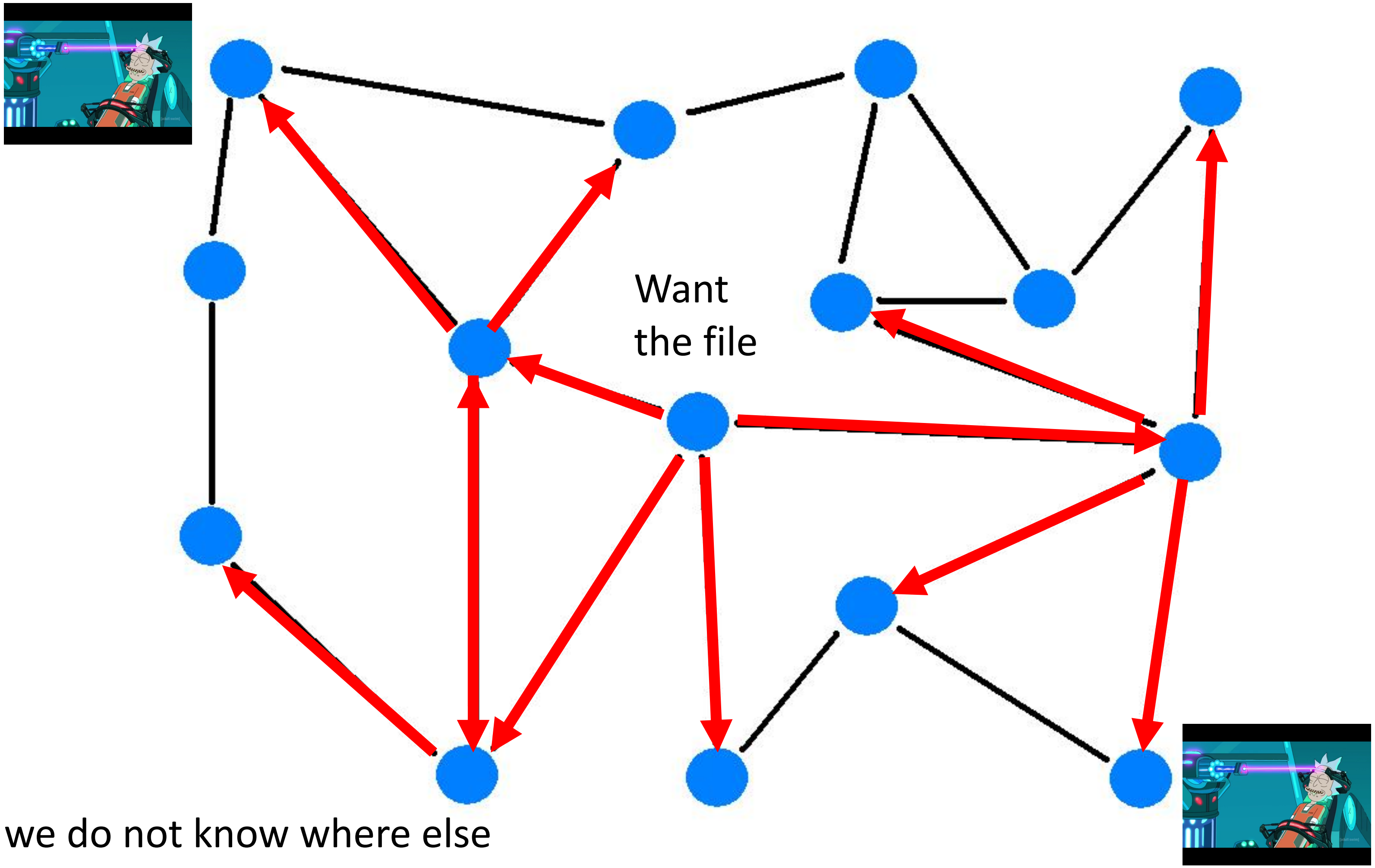
How do we solve the resource location problem?



How do we solve the resource location problem?

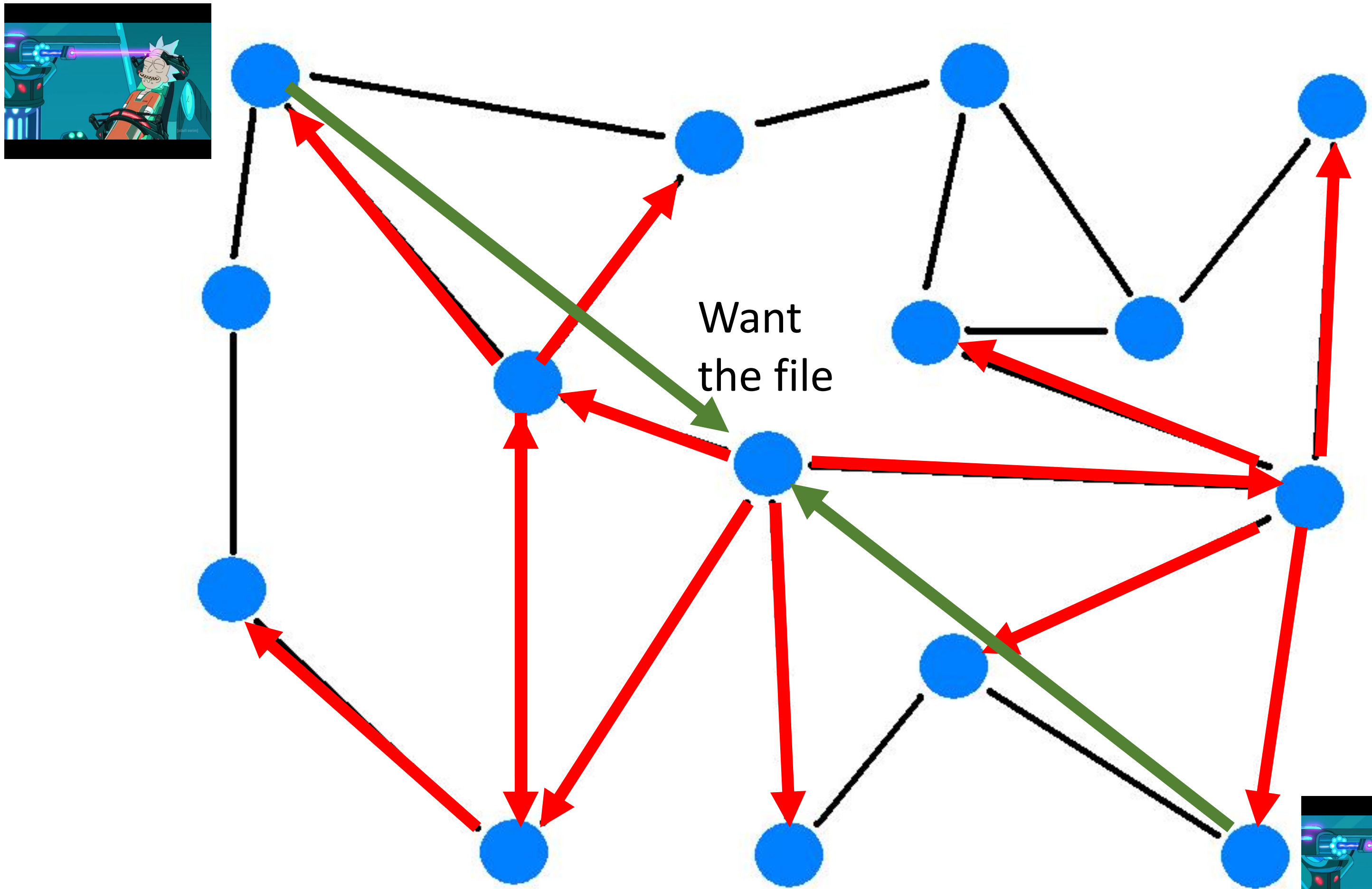


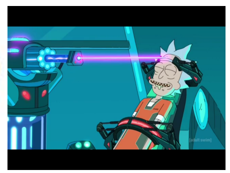
How do we solve the resource location problem?



Since we do not know where else other resources might be, we continue…

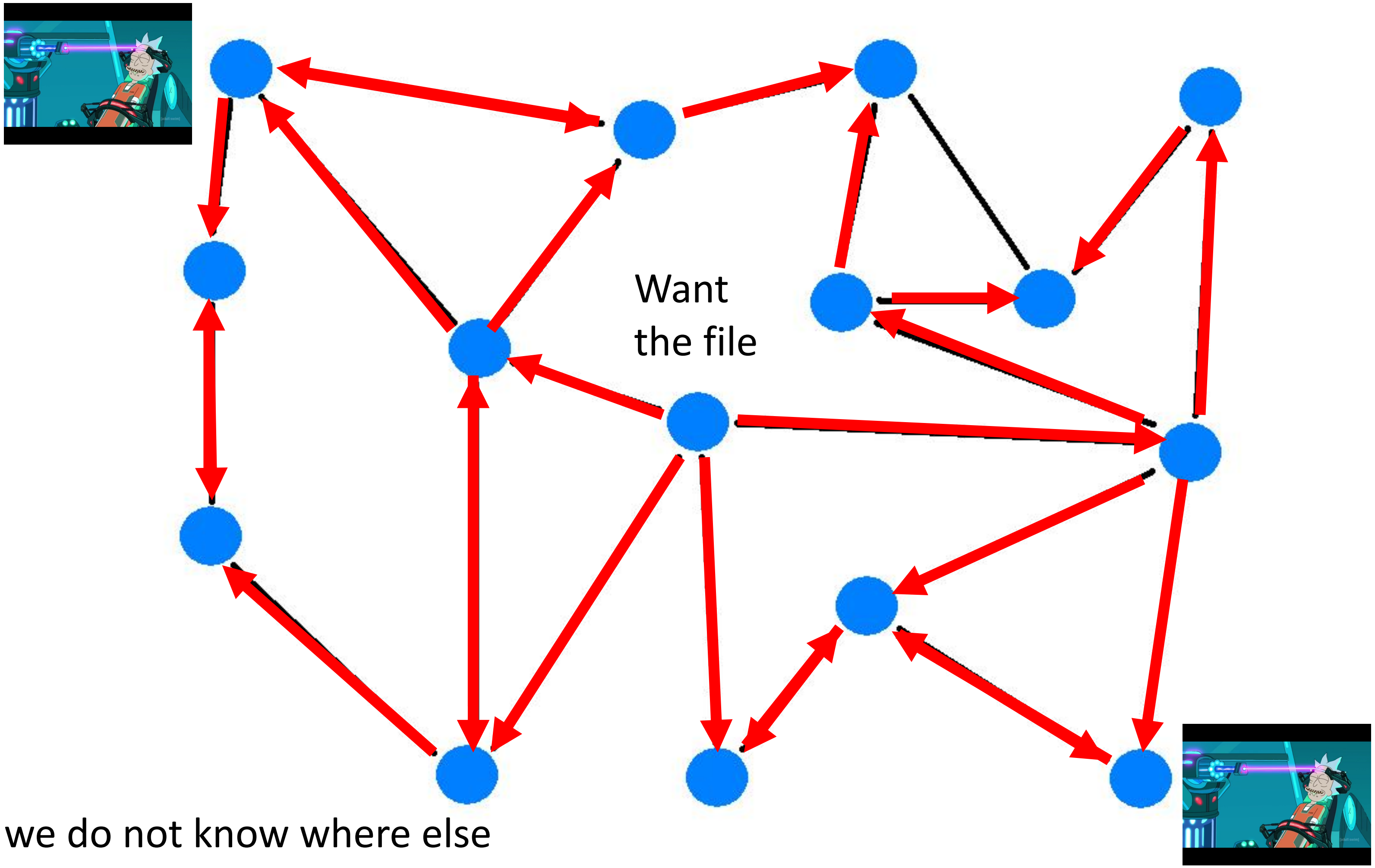
How do we solve the resource location problem?





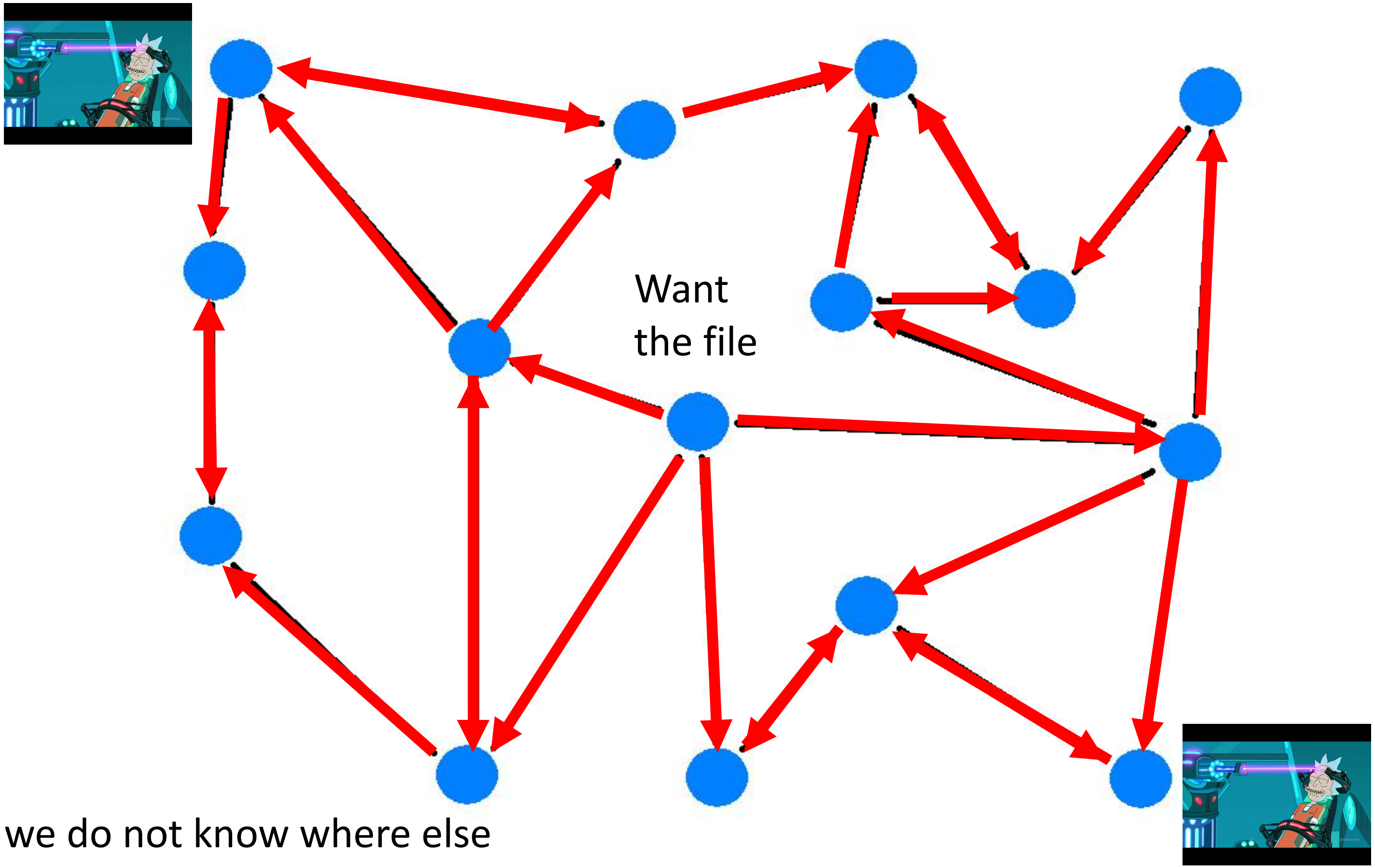
Since we do not know where else other resources might be, we continue…

How do we solve the resource location problem?



Since we do not know where else other resources might be, we continue…

How do we solve the resource location problem?



Since we do not know where else other resources might be, we continue…

Resource Location

- That is the gossip variant called Flooding.
- This was the solution implemented by a protocol named Gnutella (Version 1.0)
- It was used to support the search in file sharing applications in the first decade of 2000s (Similar to Shareaza, Limewire)
- What is the problem with this solution?

Resource Location

- That is the gossip variant called Flooding.
- This was the solution implemented by a protocol named Gnutella (Version 1.0)
- It was used to support the search in file sharing applications in the first decade of 2000s (Similar to Shareaza, Limewire)
- What is the problem with this solution?
    - Too many messages are generated and forwarded among processes, which might overload them…

Resource Location

- What could we do to address the problem of too many messages being disseminated?

Resource Location

- What could we do to address the problem of too many messages being disseminated?
- Two solutions were employed in the past:
  - Flooding with limited horizon.
  - Super-Peer Networks (Popularized in Gnutella V2).

Flooding with limited horizon

- When a query message is disseminated, it carries a value (e.g, hopCount) that is initially set to zero.
- This value is incremented whenever the message is retransmitted.
- Processes stop forwarding the message when the hopCount value reaches a given threshold.

Flooding with limited horizon

- When a query message is disseminated, it carries a value (e.g, hopCount) that is initially set to zero.
- This value is incremented whenever the message is retransmitted.
- Processes stop forwarding the message when the hopCount value reaches a given threshold.
- Why is this good -> Most messages are generated later in the dissemination (and most redundant messages also).
- Why is this bad -> You no longer have garantees of finding all relevant resources.

Unstructured Overlays based on Super-Peers.

- A small fraction of processes (those that have more resources, are more porwerfull, or simply more stable) are promoted to Super-Peers.
- Super-Peers form an unstructured overlay among them.
- Regular processes connect to a super-peer and transmit to it the index of its resources.
- Queries are forwarded to the super-peer and them disseminated only among super-peers.

Unstructured Overlays based on Super-Peers.

- A small fraction of processes (those that have more resources, are more porwerfull, or simply more stable) are promoted to Super-Peers.
- Super-Peers form an unstructured overlay among them.
- Regular processes connect to a super-peer and transmit to it the index of its resources.
- Queries are forwarded to the super-peer and them disseminated only among super-peers.
- Why is this good -> Significantly reduces the amount of messages.
- Why is this bad -> How do you decide which process should be a super-peer? Load in the system is highly unbalanced.

What if, you known exactly the resource you are looking for?

- Do we need all of this?
- Can we do something better?

Resource Location (Exact Match)

- A Definition:

Given a set of processes containing different sets of resources, locate the processes that contain a given resource given its unique identifier.

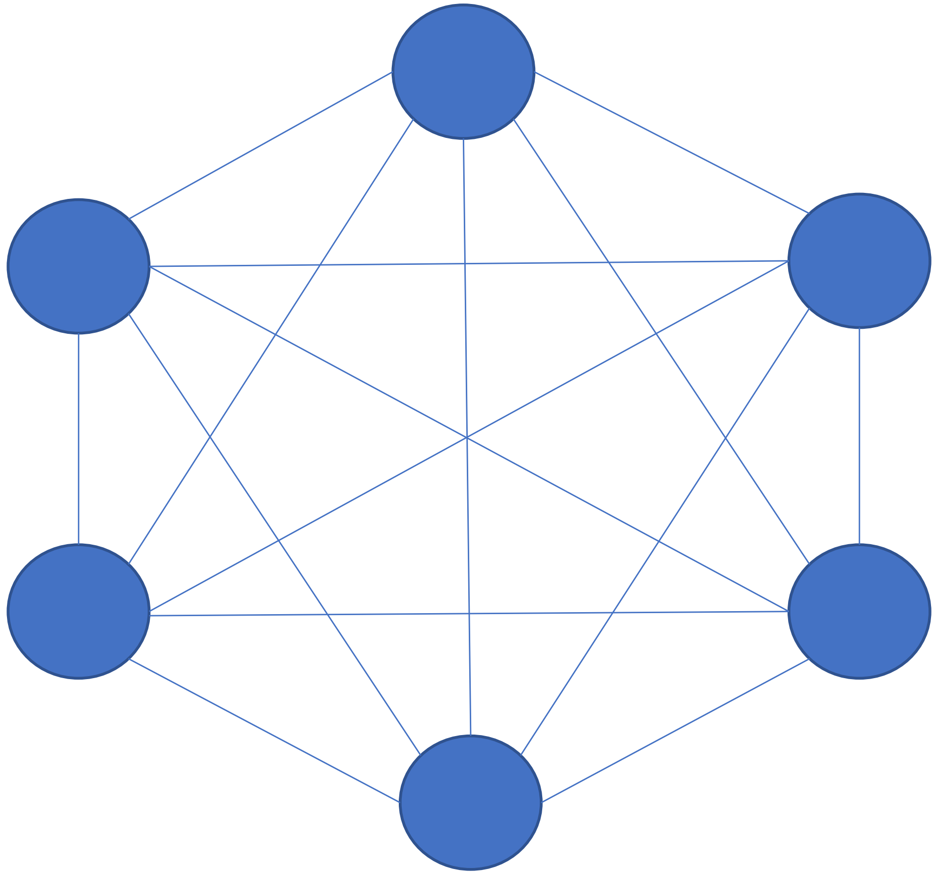
- One possible concretization:
  - File Sharing Applications.
  - Processes own a set of files that have properties.
  - Locate the processes that have a file named: “Rick and Morty - S03E01”.

Consistent Hashing

- We can build a distributed index of resources among all processes by doing the following.
- We pick a hash function that generates hash values in the interval [0, G].
- We attribute to each process an identifier within the interval [0, G] (all processes must have a different identifier.
- For each resource in the system, we compute the hash of its identifier, and store information about it in the process with the closest identifier.

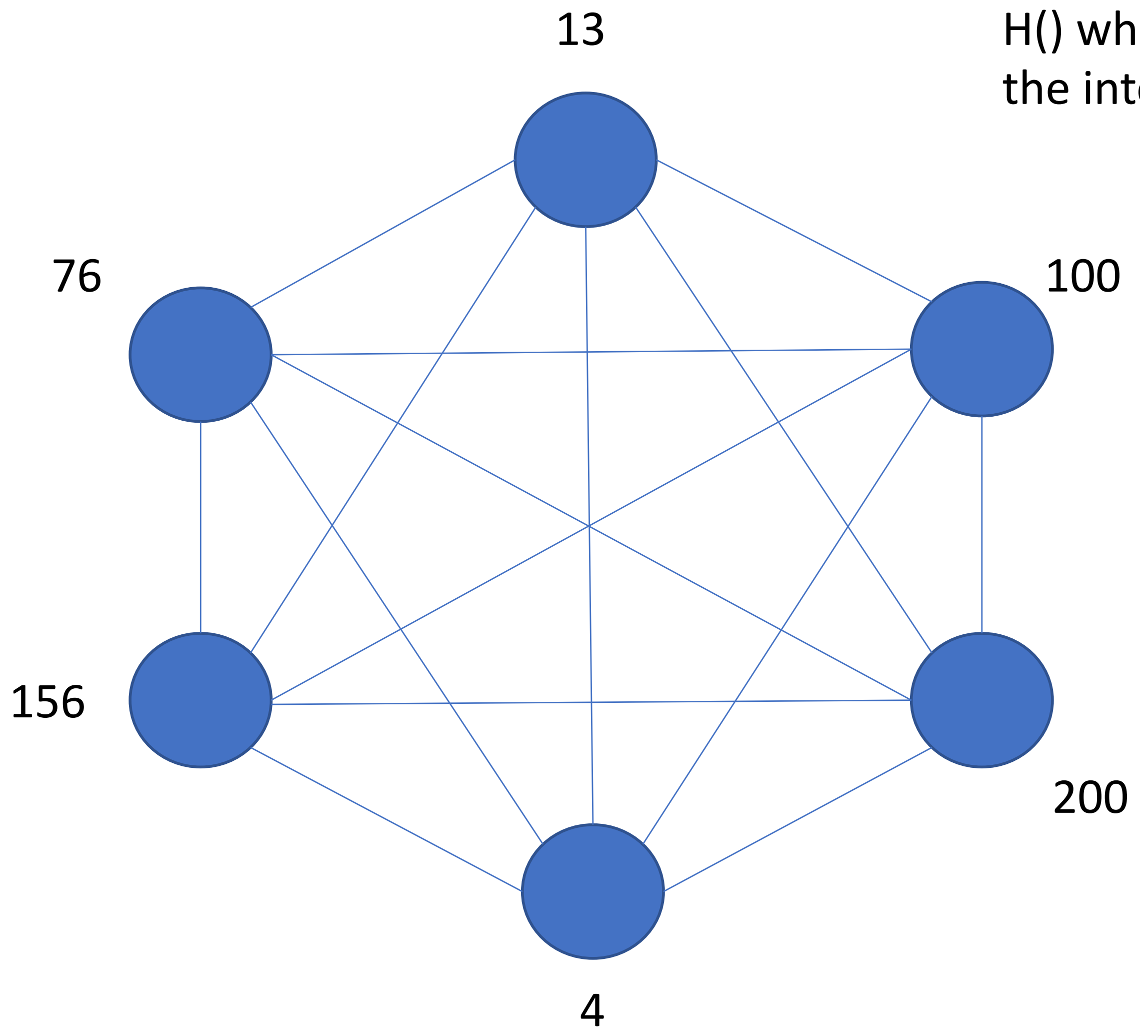
Consistent Hashing

Let's assume full membership information.



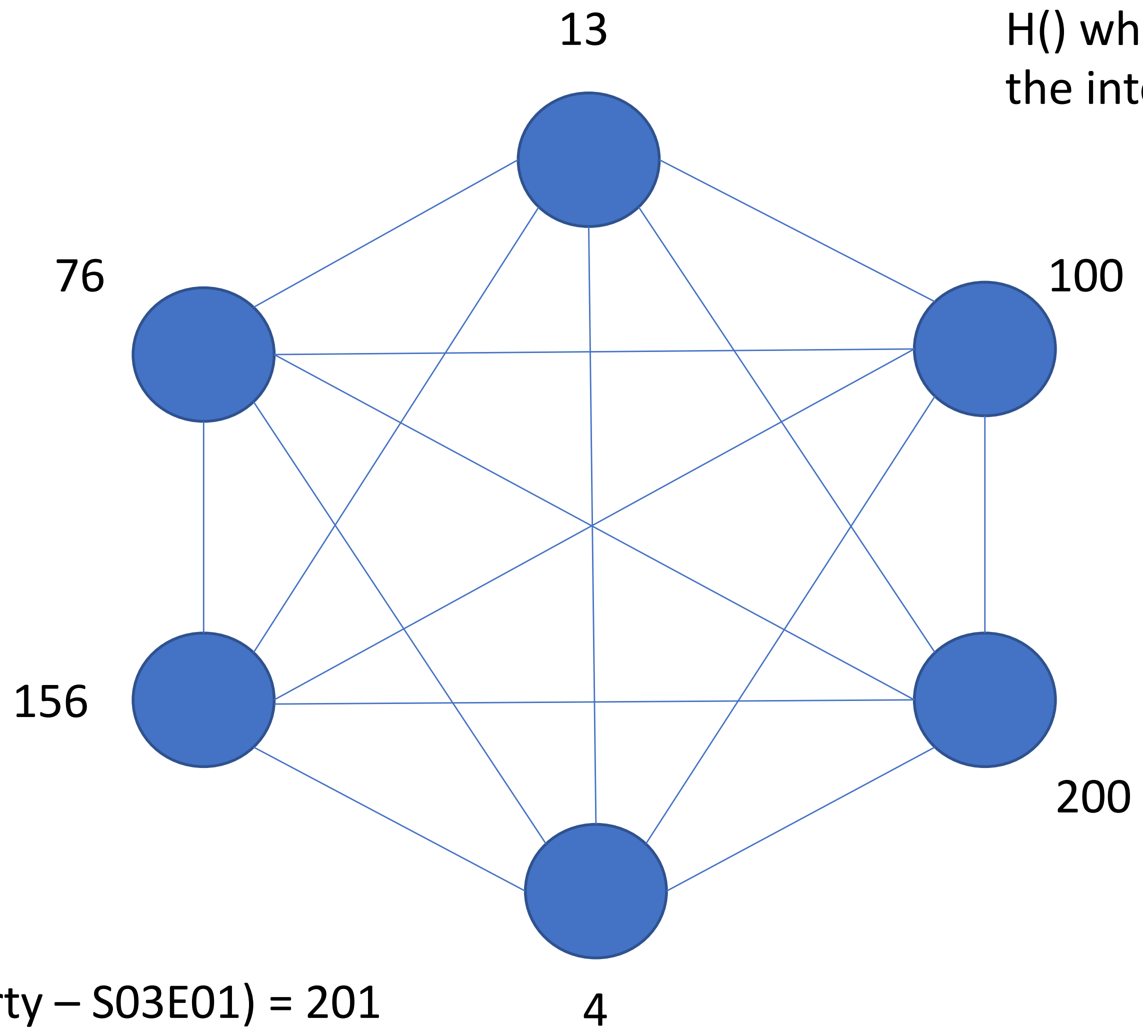
Consistent Hashing

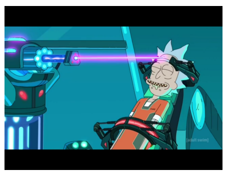
Assume a hash function H() whose result is within the interval: [0, 256].



Consistent Hashing

Assume a hash function H() whose result is within the interval: [0, 256].

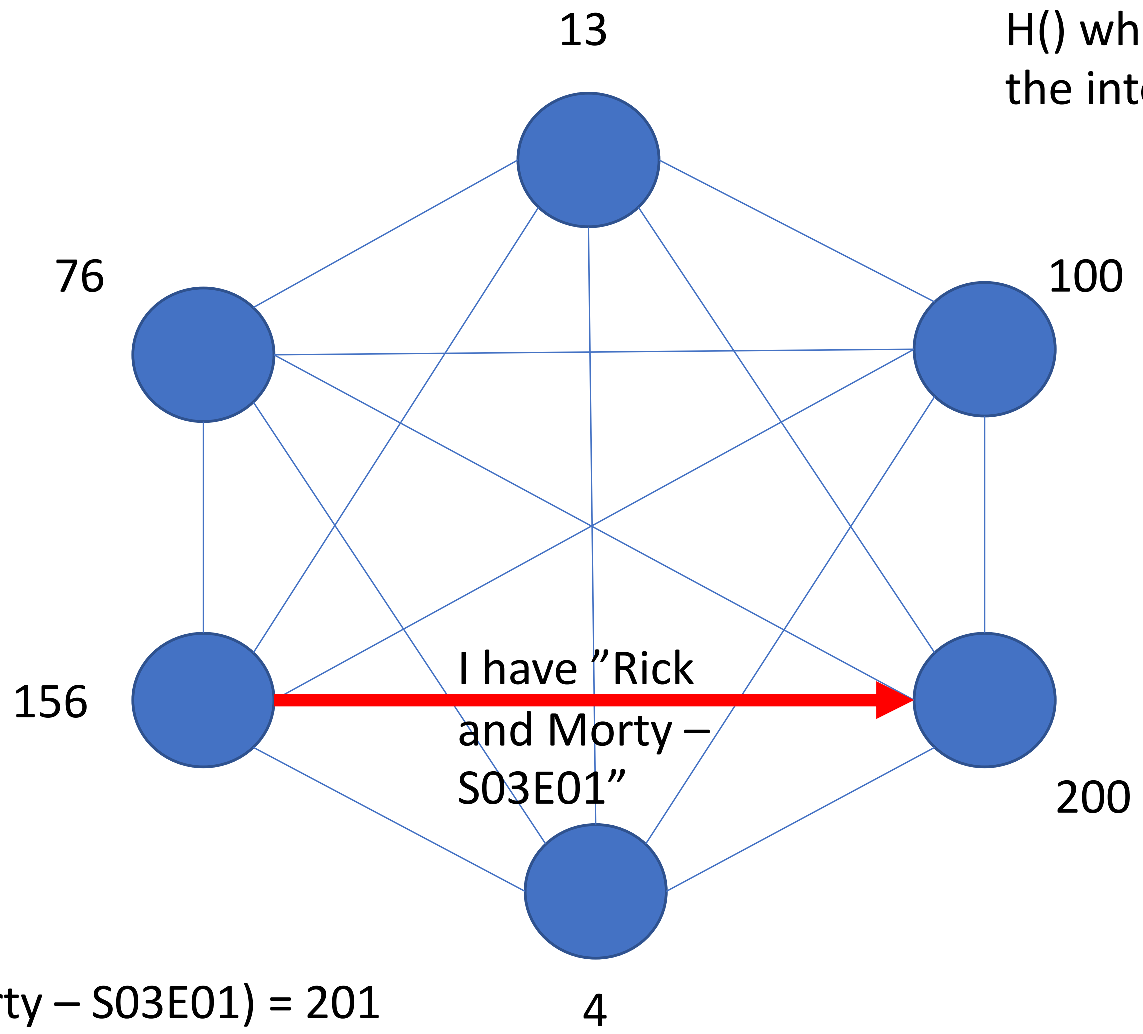


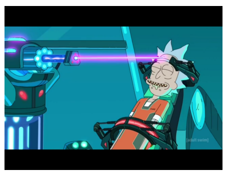


H(Rick and Morty – S03E01) = 201

Consistent Hashing

Assume a hash function H() whose result is within the interval: [0, 256].

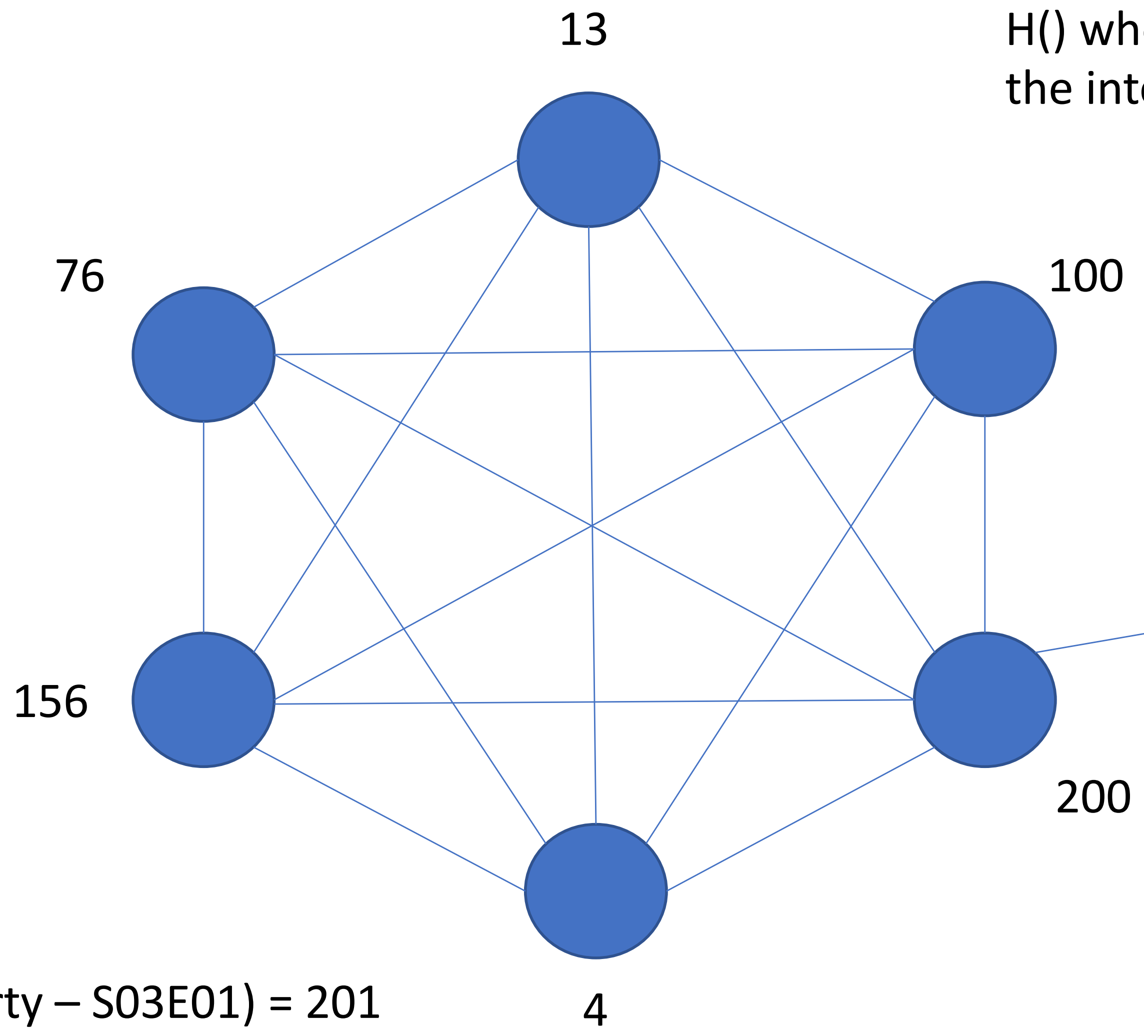




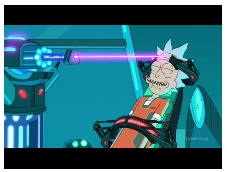
H(Rick and Morty – S03E01) = 201

Consistent Hashing

Assume a hash function H() whose result is within the interval: [0, 256].



“Rick and Morty – S03E01” @ 156

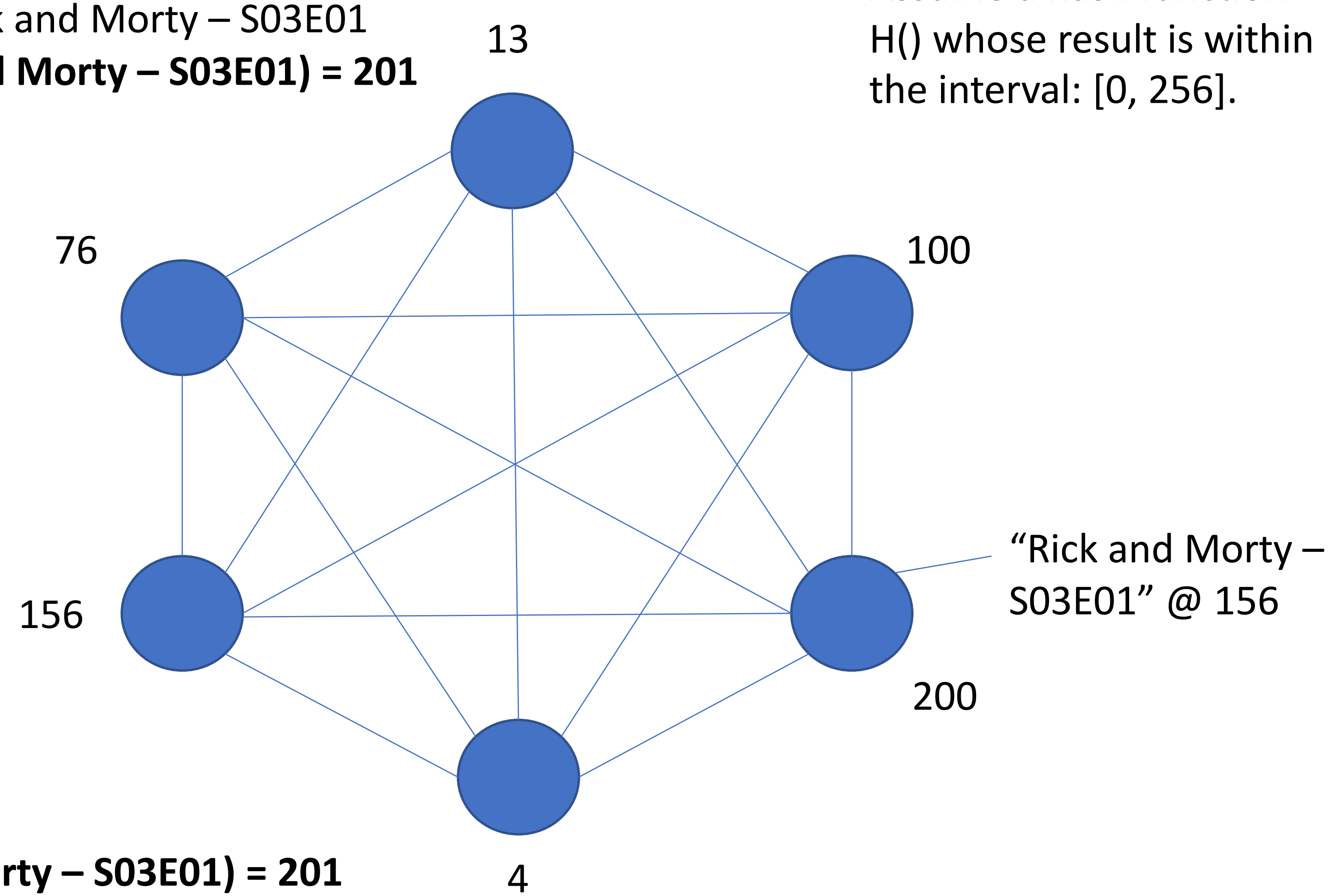


H(Rick and Morty – S03E01) = 201

Consistent Hashing

I want Rick and Morty – S03E01
H(Rick and Morty – S03E01) = 201

Assume a hash function H() whose result is within the interval: [0, 256].

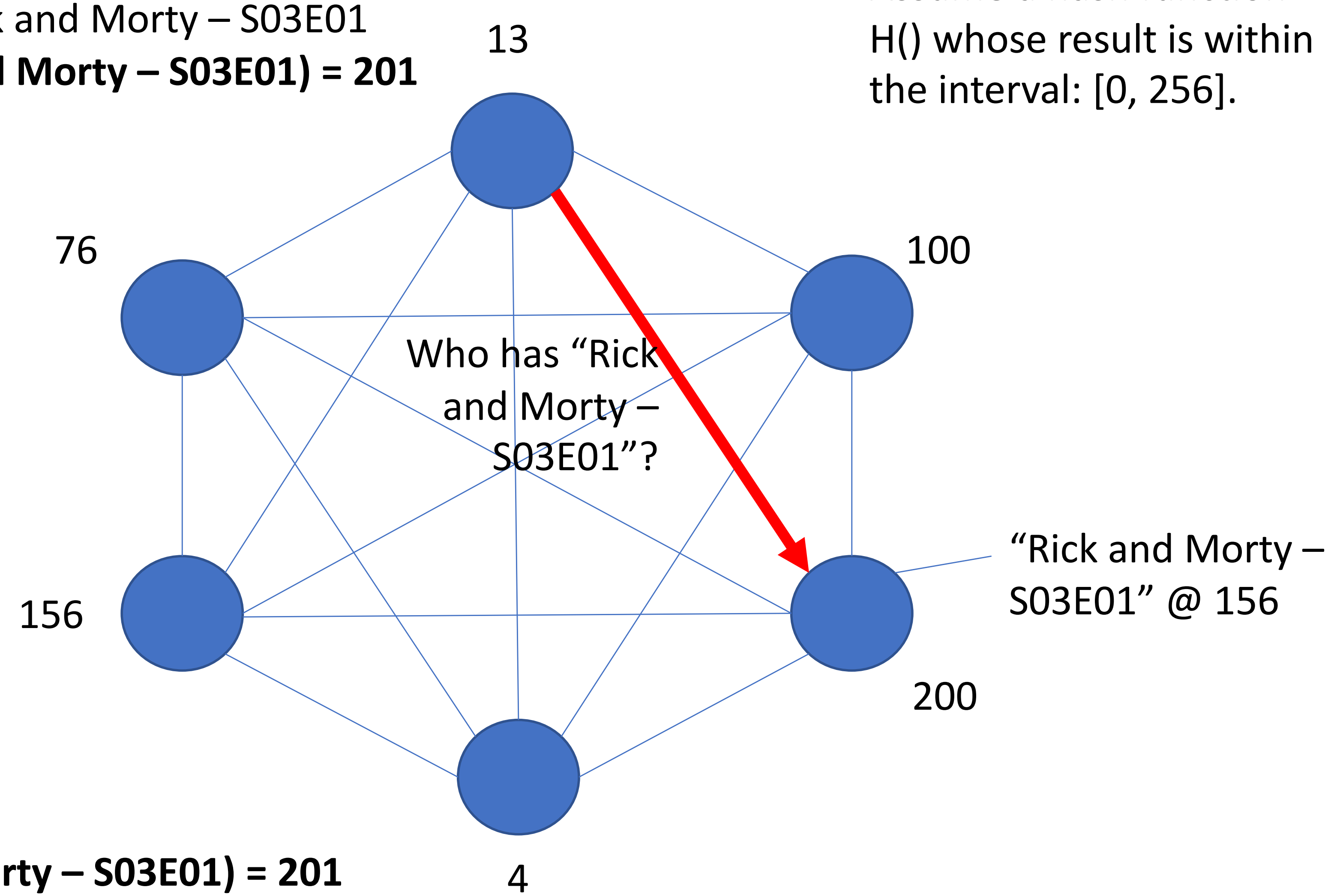


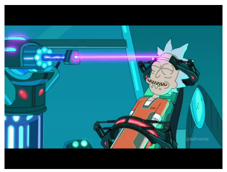
H(Rick and Morty – S03E01) = 201

Consistent Hashing

I want Rick and Morty – S03E01
H(Rick and Morty – S03E01) = 201

Assume a hash function H() whose result is within the interval: [0, 256].



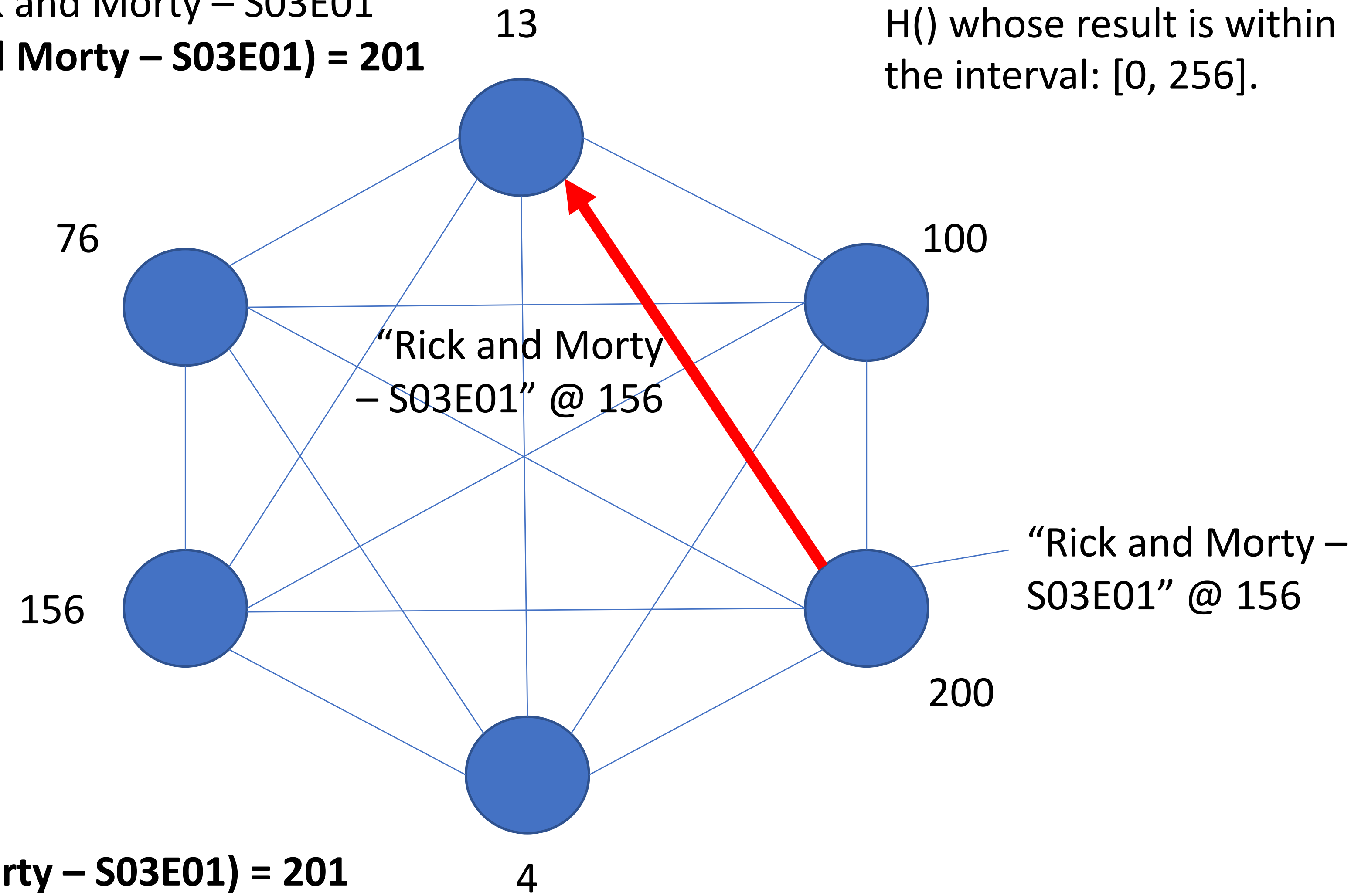


H(Rick and Morty – S03E01) = 201

Consistent Hashing

I want Rick and Morty – S03E01
H(Rick and Morty – S03E01) = 201

Assume a hash function H() whose result is within the interval: [0, 256].

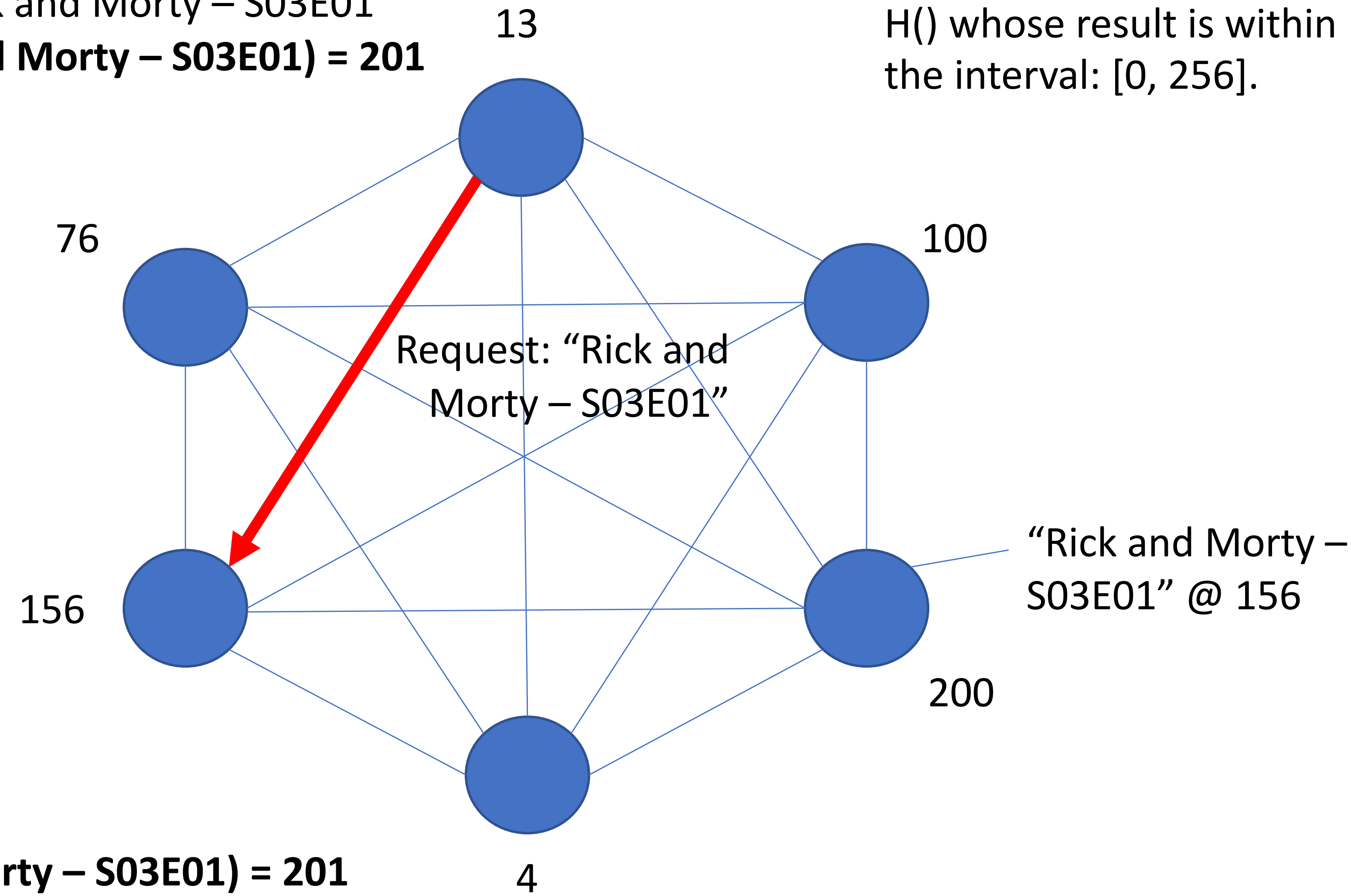


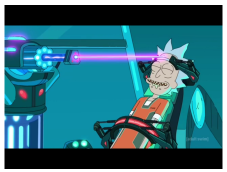
H(Rick and Morty – S03E01) = 201

Consistent Hashing

I want Rick and Morty – S03E01
H(Rick and Morty – S03E01) = 201

Assume a hash function H() whose result is within the interval: [0, 256].





H(Rick and Morty – S03E01) = 201

Consistent Hashing

- Consistent Hashing leverages the fact that independent processes will obtain the same value when appying a hash function to the same arbitrary input.
- When we have the full membership (i.e., every process in the system knowns all other) this allows to build a **One-Hop Distributed Hash Table (DHT).**
- One-Hop DHTs are one of the foundations of many modern NoSQL Datastores such as Cassandra, Dynamo, MongoDB, ...

What if we have a huge number of processes??

- Should we still use a one-hop DHT?
- Is there a better alternative?

What if we have a huge number of processes??

- Should we still use a one-hop DHT?
- Is there a better alternative?

- Problem: Full membership implies that every process has to know all other processes, if the system is large, changes in the membership might be frequent, and the cost to keep this information up-to-date becomes too expensive.
- Solution: partial views (e.g., Cyclon or HyParView).

What if we have a huge number of processes??

- Should we still use a one-hop DHT?
- Is there a better alternative alternative?

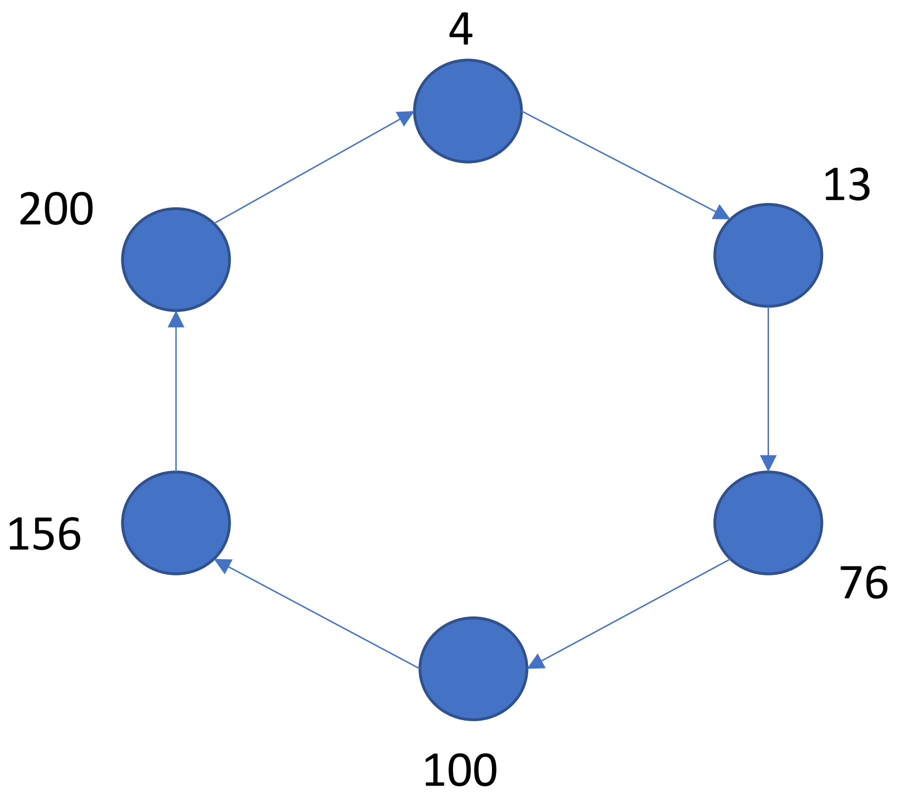
- Problem: Full membership implies that every process has to know all other processes, if the system is large, changes in the membership might be frequent, and the cost to keep this information up-to-date becomes too expensive.
- Solution: partial views (e.g., Cyclon or HyParView).
- **Caviat: You need to (efficiently) find a process given its identifier.**

Structured Overlay Networks

- An overlay network composed of logical links between processes, whose topology has properties known a-priori.
- Many times, these properties are related with the identifiers of nodes (but there are exceptions).

Structured Overlay Networks (first step)

- The most common overlay topology in structured overlay networks are rings, that connect nodes in order considering their identifiers.



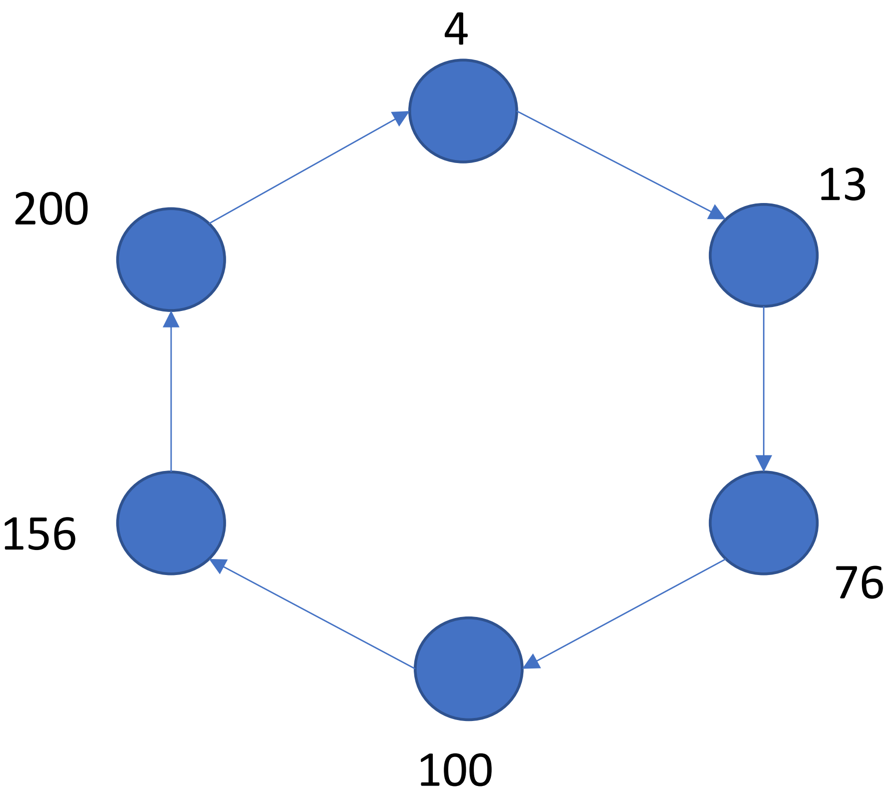
This is already very good:

- You can find the process that is responsible by a given number.
- Just go over the ring, until you find it.

In fact, the ring structure, in this type of structured overlay is the prime correctness criteria.

Structured Overlay Networks (first step)

Why is this not good enough?



This is already very good:

- You can find the process that is responsible by a given number.
- Just go over the ring, until you find it.

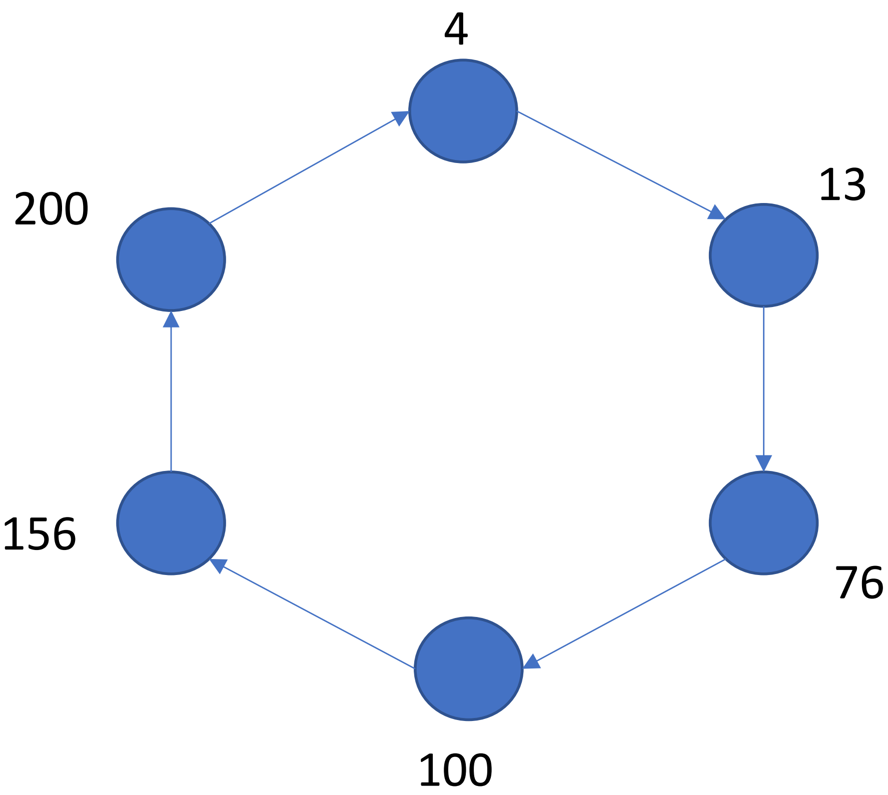
In fact, the ring structure, in this type of structured overlay is the prime correctness criteria.

Structured Overlay Networks (first step)

- T d
  c n
  c

Why is this not good enough?

Long paths between processes.



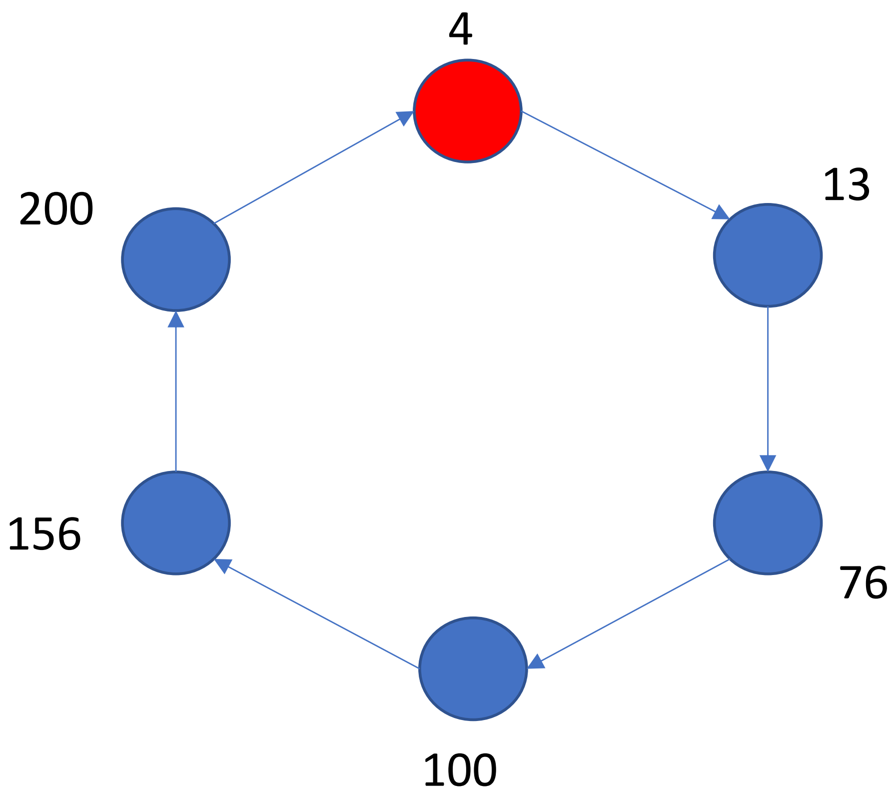
This is already very good:

- You can find the process that is responsible by a given number.
- Just go over the ring, until you find it.

In fact, the ring structure, in this type of structured overlay is the prime correctness criteria.

Structured Overlay Networks (first step)

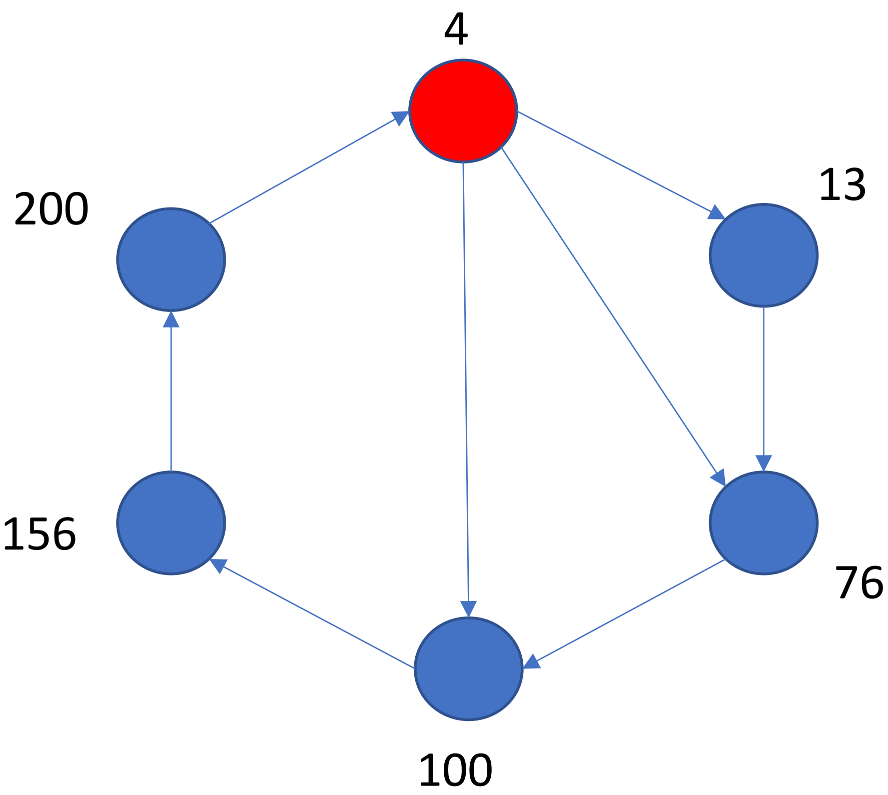
- The most common overlay topology in structured overlay networks are rings, that connect nodes in order considering their identifiers.



We add some additional overlay links (yes in the order of ln(# π)) to speed up things.

Structured Overlay Networks (first step)

- The most common overlay topology in structured overlay networks are rings, that connect nodes in order considering their identifiers.



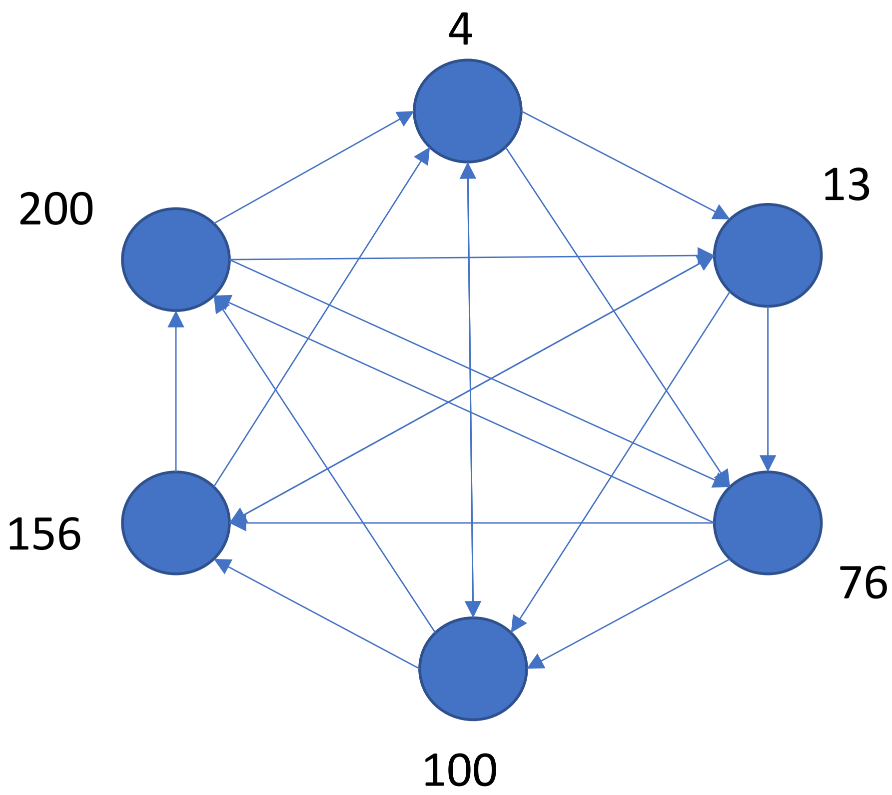
We add some additional overlay links (yes in the order of ln(# π)) to speed up things.

This solves two problems:

- **We now have information to deal with faults.**

Structured Overlay Networks (first step)

- The most common overlay topology in structured overlay networks are rings, that connect nodes in order considering their identifiers.



We add some additional overlay links (yes in the order of ln(# π)) to speed up things.

This solves two problems:

- **We now have information to deal with faults.**
- **But now, I can also reach any other process in a logarithmic number of hops (I can always reduce in half the distance to my target at each hop).**

Structured Overlay Networks: Relevant Examples

- There are a few relevant examples of such algorithms in the Literature:
- Chord (Canonical Academic Example)
- Pastry (Similar principles, different Algorithm)
- Kadmelia (and its famous implementation Kad)

Structured Overlay Networks: Relevant Examples

- There are a few relevant examples of such algorithms in the Literature:
- **Chord (Canonical Academic Example)**
- Pastry (Similar principles, different Algorithm)
- Kadmelia (and its famous implementation Kad)

The Chord Protocol

- Described here:

Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications

Ion Stoica[†], Robert Morris[‡], David Liben-Nowell[‡], David R. Karger[‡], M. Frans Kaashoek[‡], Frank Dabek[‡], Hari Balakrishnan[‡]

The Chord Protocol

- What is the state kept by each node?
- Assuming that process identifiers have m bits.

finger$[k]$	first node on circle that succeeds $(n + 2^{k-1}) \bmod 2^m, 1 \leq k \leq m$
successor	the next node on the identifier circle; *finger*$[1]$.*node*
predecessor	the previous node on the identifier circle

The Chord Protocol

- The key functionality of Chord (i.e., its interface), given an identifier, find the process responsible for managing that identifier.

```
// ask node n to find the successor of id
n.find_successor(id)
  if (id ∈ (n, successor])
    return successor;
  else
    n′ = closest_preceding_node(id);
    return n′.find_successor(id);

// search the local table for the highest predecessor of id
n.closest_preceding_node(id)
  for i = m downto 1
    if (finger[i] ∈ (n, id))
      return finger[i];
  return n;
```

The Chord Protocol

- How do we manage the finger table (that materialize the ring and extra paths)
- The “Init” step:

// *create a new Chord ring.*
$n.\mathbf{create}()$
  predecessor $= \mathbf{nil}$;
  successor $= n$;

// *join a Chord ring containing node* n'.
$n.\mathbf{join}(n')$
  predecessor $= \mathbf{nil}$;
  successor $= n'.find\_successor(n)$;

The Chord Protocol

- How do we manage the finger table (that materialize the ring and extra paths)
- Ensuring that the successor is correct:

// *called periodically. verifi es n's immediate*
// *successor, and tells the successor about* n.
n.**stabilize**()
    $x = \textit{successor}.\textit{predecessor}$;
    if $(x \in (n, \textit{successor}))$
        $\textit{successor} = x$;
    $\textit{successor}.\textit{notify}(n)$;

// n' *thinks it might be our predecessor.*
$n.\mathbf{notify}(n')$
    if ($\textit{predecessor}$ **is nil or** $n' \in (\textit{predecessor}, n))$
        $\textit{predecessor} = n'$;

The Chord Protocol

- How do we manage the finger table (that materialize the ring and extra paths)
- Ensuring that all other links are correct:

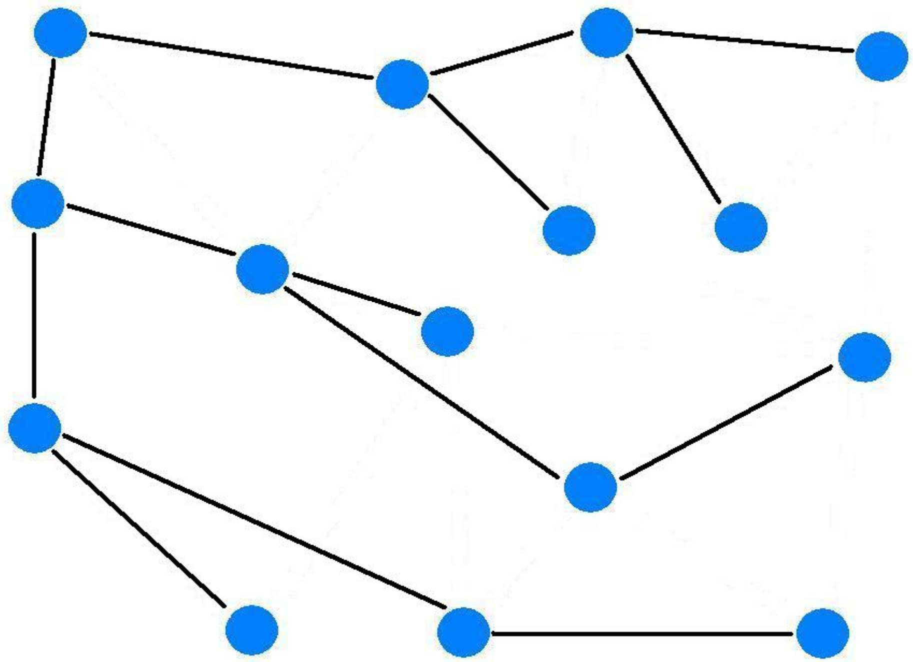
```
// called periodically. refreshes finger table entries.
// next stores the index of the next finger to fix.
n.fix_fingers()
    next = next + 1;
    if (next > m)
        next = 1;
    finger[next] = find_successor(n + 2^(next-1));

// called periodically. checks whether predecessor has failed.
n.check_predecessor()
    if (predecessor has failed)
        predecessor = nil;
```

Other Structured Overlay Networks.

- The definition only states that the topology of the overlay has some known property.
- One can be that nodes are organized in an ordered ring, which is good to find and route information among nodes.
- Can you think of another one?

Other Structured Overlay Networks.

- Tree-based overlay networks:
  - Good for disseminate messages with low overhead.
  - Also good to aggregate information.



Overview of Overlays:

- Unstructured (or Random):
  - + : Easy to build and maintain.
  - + : Robust to failures (any failed process can be replaced by any other failed process).
  - - : Limited efficiency for some use cases (locate a particular object or process for instance).
- Structured
  - + : Provides efficiency for particular types of applications (application-level routing, exact-search, broadcast).
  - - : Less robust to failures (a failed process can only be replaced – in another process partial view – by a limited number of other processes).
  - - : Somewhat more complex algorithms.

A final note on Gossip and Overlays

- Many of these works were originated by the Peer-to-Peer community, which might lead to the question "why does this still matters now-a-days?"

A final note on Gossip and Overlays

- Many of these works were originated by the Peer-to-Peer community, which might lead to the question “why does this still matters now-a-days?” or “why does this still matters now-a-days, netflix is a thing now, so who wants to share files online?”

A final note on Gossip and Overlays

- Many of these works were originated by the Peer-to-Peer community, which might lead to the question “why does this still matters now-a-days?” or “why does this still matters now-a-days, netflix is a thing now, so who wants to share files online?”
- All of these approaches have found new applications in the context of cloud-computing, namely to help manage very large systems running in the cloud.

A final note on Gossip and Overlays

- Many of these works were originated by the Peer-to-Peer community, which might lead to the question “why does this still matters now-a-days?” or “why does this still matters now-a-days, netflix is a thing now, so who wants to share files online?”
- All of these approaches have found new applications in the context of cloud-computing, namely to help manage very large systems running in the cloud.
- Now-a-Days we live in the Internet-of-Things and Edge Computing eras... Many of these solutions can be useful there.

Edge Computing Area

- Cloud datacenter can scale (virtual notion of infinite resources).
- Network link to cloud datacenters however cannot easily scale.

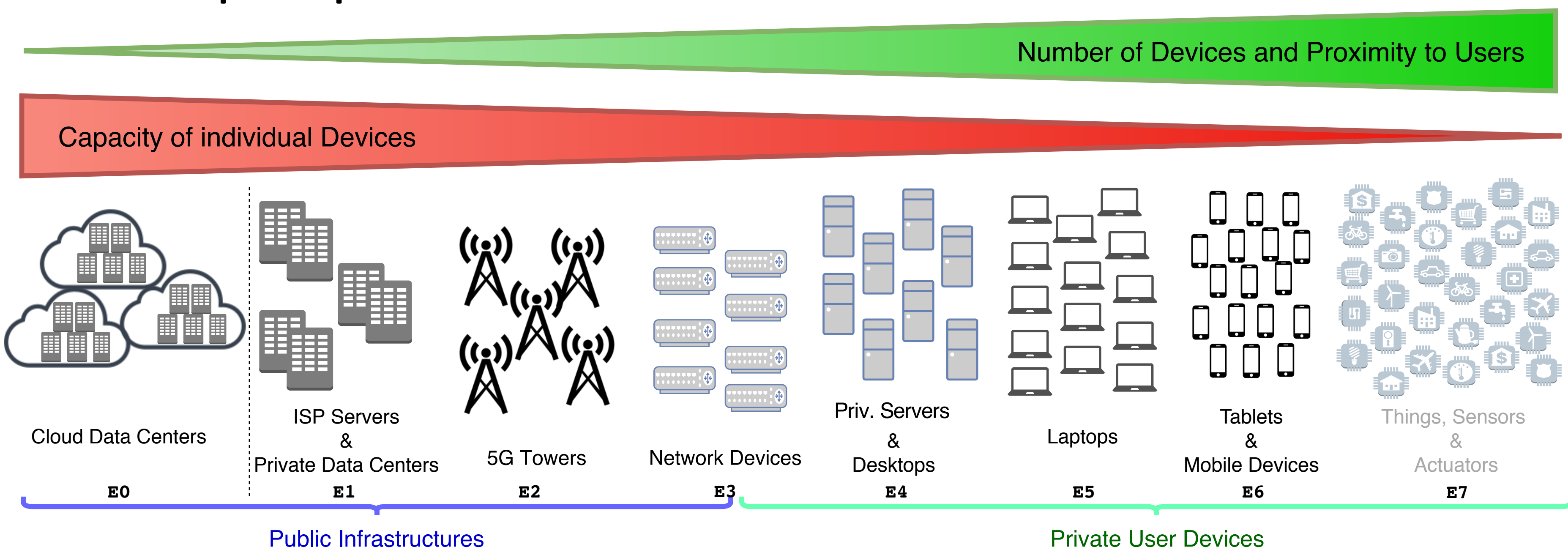
- Problem: Applications that generate lots of data might have problem in shipping all that data to cloud infrastructures and get answers in timely fashion

  (e.g., IoT, Mobile games)

- Solution: Put computations beyond the data center boundaries.

Edge Computing

Quick perspective:



Leitão et. al. 2018
(European project LightKone)

Some challenges

- Lots of devices to manage and different administrative domains.
- Need to make adequate choices regarding where to execute different computations.
- Computations need data, so data must be also managed on edge devices.
- Security: Data privacy, Data integrity
  (check CSD course)