UNIVERSITY OF
Southampton

UNIVERSITY OF Southampton

Peer-to-Peer Data Management

COMP3211 Advanced Databases

Dr Nicholas Gibbins – nmg@ecs.soton.ac.uk

2020-2021

Peer-to-Peer Characteristics

Autonomy

Query Expressiveness

Efficiency

Quality of Service

Fault-Tolerance

Security

Peer-to-Peer Characteristics

Autonomy

Query Expressiveness

Efficiency

Quality of Service

Fault-Tolerance

Security

- Peers may join or leave at any time without restriction
- Peers control the data they store
- Peers control which other peers store their data

Peer-to-Peer Characteristics

Autonomy

Query Expressiveness

Efficiency

Quality of Service

Fault-Tolerance

Security

- User may describe desired data at appropriate level of detail
  - Key lookup
  - Keyword search with ranking
  - Structured queries

Peer-to-Peer Characteristics

Autonomy

Query Expressiveness

Efficiency

Quality of Service

Fault-Tolerance

Security

- Efficient use of resources
  - Lower cost
  - Higher throughput

Peer-to-Peer Characteristics

Autonomy

Query Expressiveness

Efficiency

Quality of Service

Fault-Tolerance

Security

- User-perceived:
  - Completeness of results
  - Data consistency
  - Data availability
  - Query response time

Peer-to-Peer Characteristics

Autonomy

Query Expressiveness

Efficiency

Quality of Service

Fault-Tolerance

Security

- Efficiency and quality maintained despite peer failures
- Requires replication

Peer-to-Peer Characteristics

Autonomy

Query Expressiveness

Efficiency

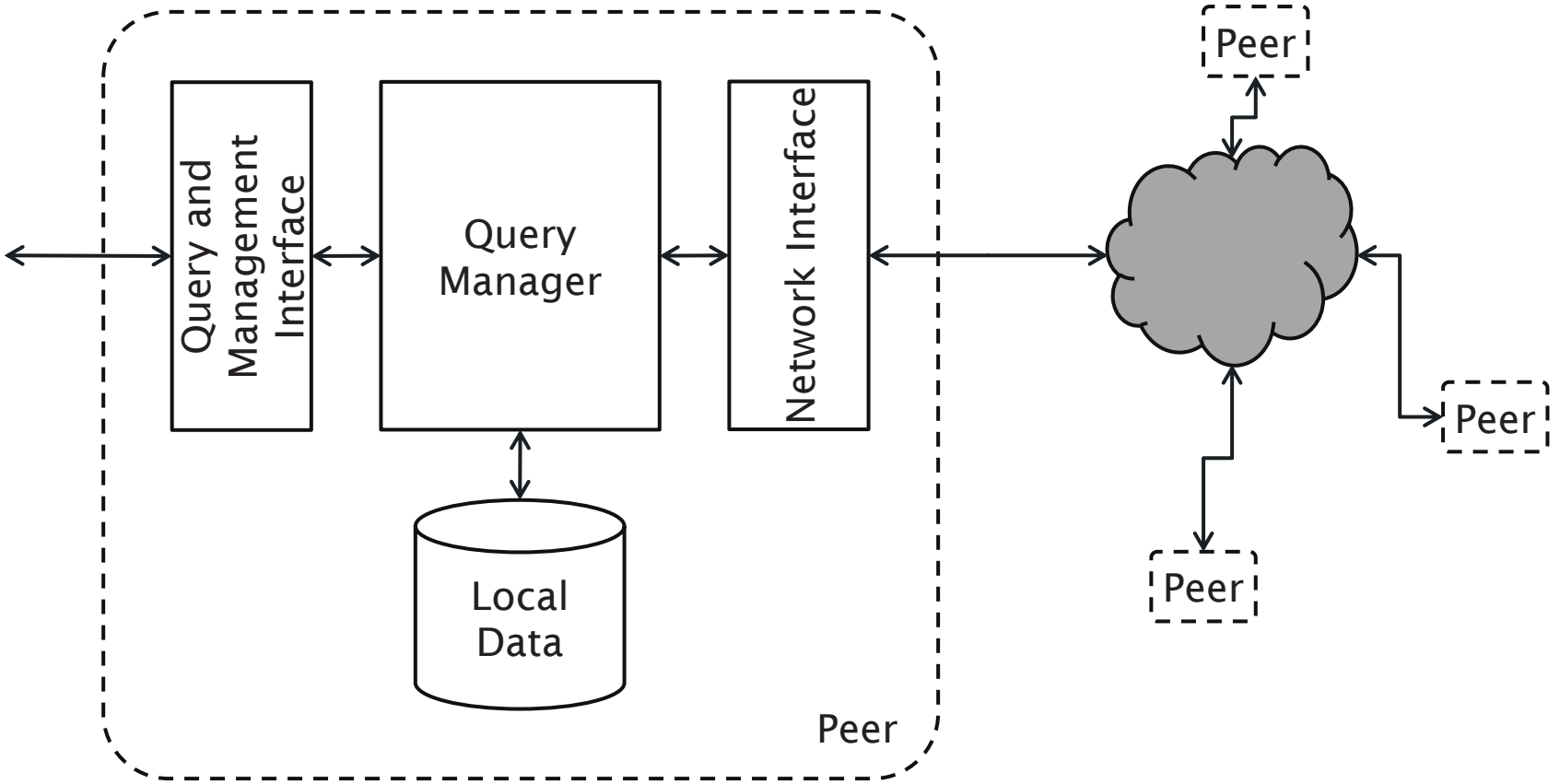
Quality of Service

Fault-Tolerance

Security

- Cannot rely on trusted servers!
- Access controls

Reference Peer-to-Peer Architecture



Data Management Issues

Data Location: peers must be able to refer to and locate data stored by other peers

Query Processing: given a query, discover peers that contribute relevant data and efficiently execute the query

Data Integration: access data despite heterogeneous schemas

Data Consistency: maintain consistency on duplicate data when replicating or caching

Overlay Networks

Peers connect to each other via an *overlay network*

- Usually has different topology to underlying physical network
- Concentrate on optimising communication over overlay network

Two common distinctions:

- Pure versus hybrid
- Structured versus unstructured

Pure versus Hybrid Networks

Pure Overlay Network

- No differentiation between peers – all are considered equal

Hybrid Overlay Network

- Some peers are given special tasks to perform
- Also referred to as *super-peer systems*

Structured versus Unstructured Networks

Unstructured Overlay Network

- Peers may communicate directly with any of their neighbours
- Peers may join network by attaching to any other peer

Structured Overlay Network

- Tightly controlled topology and message routing
- Peers may only communicate with certain other peers
- Peers may only join the network in certain places

Unstructured Peer-to-Peer Networks

Overlay network constructed in an ad hoc manner

Data placement is unrelated to overlay topology

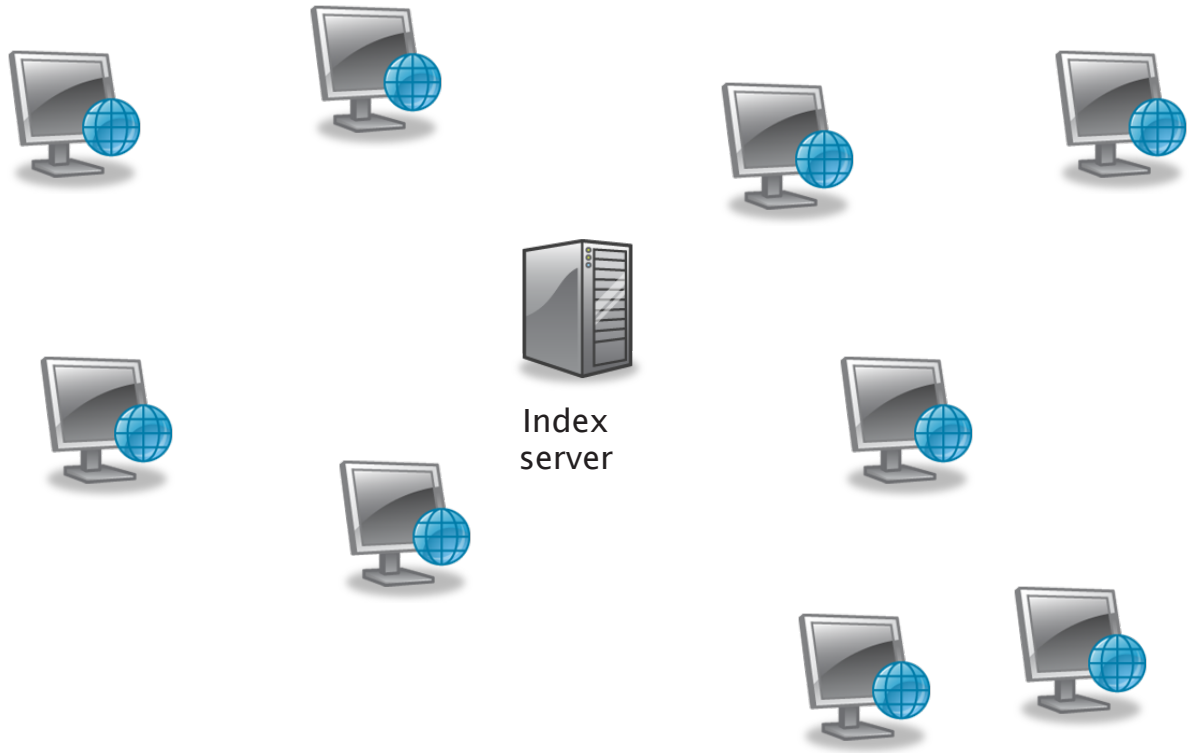
Examples:

- Gnutella, Freenet, BitTorrent, Kazaa

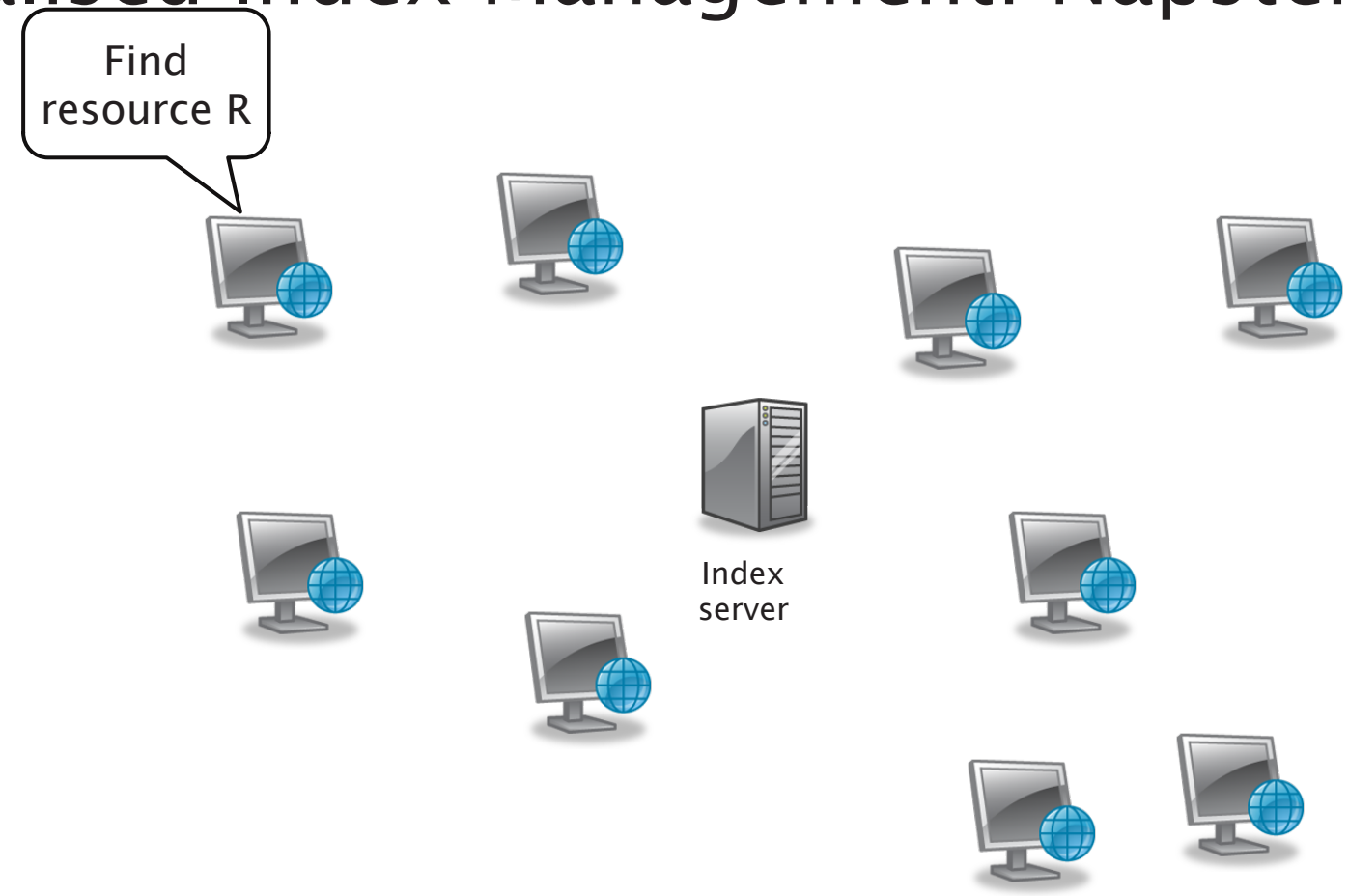
Fundamental issue of index management:

- Which peers hold which resources?
- Centralised versus distributed

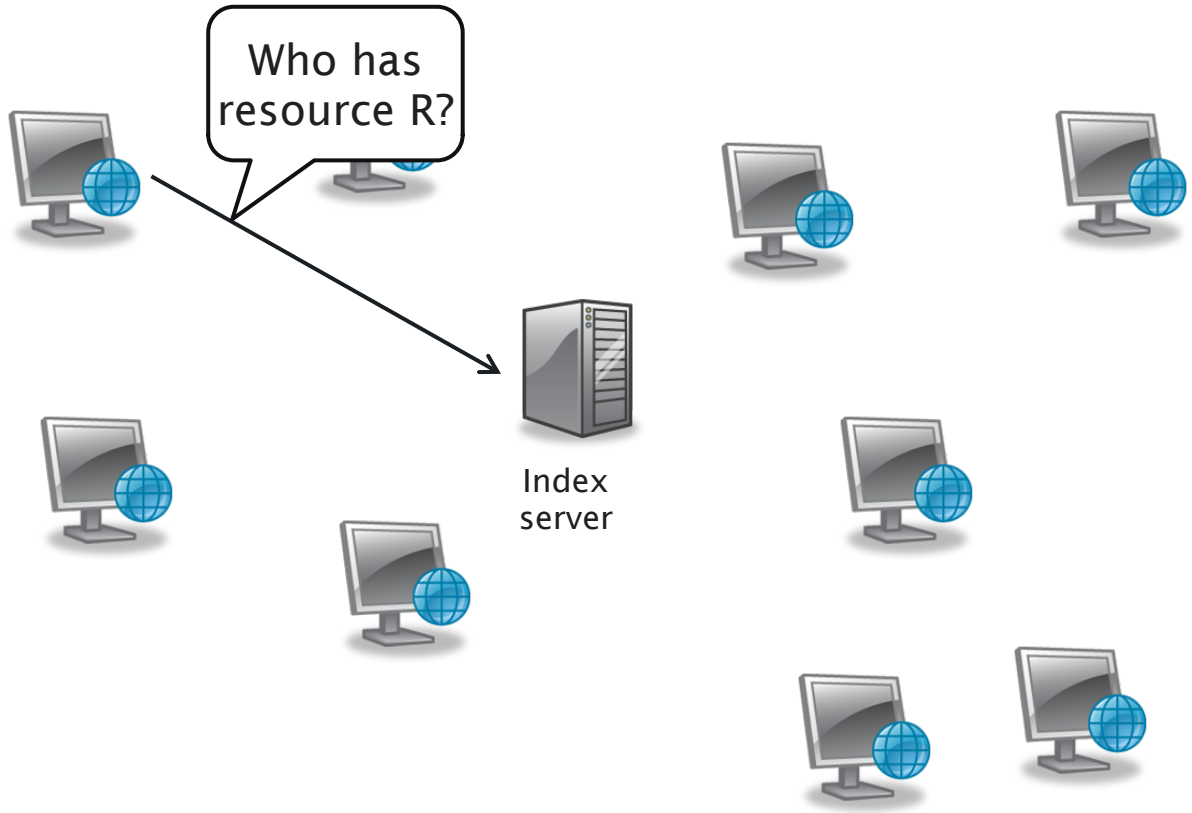
Centralised Index Management: Napster



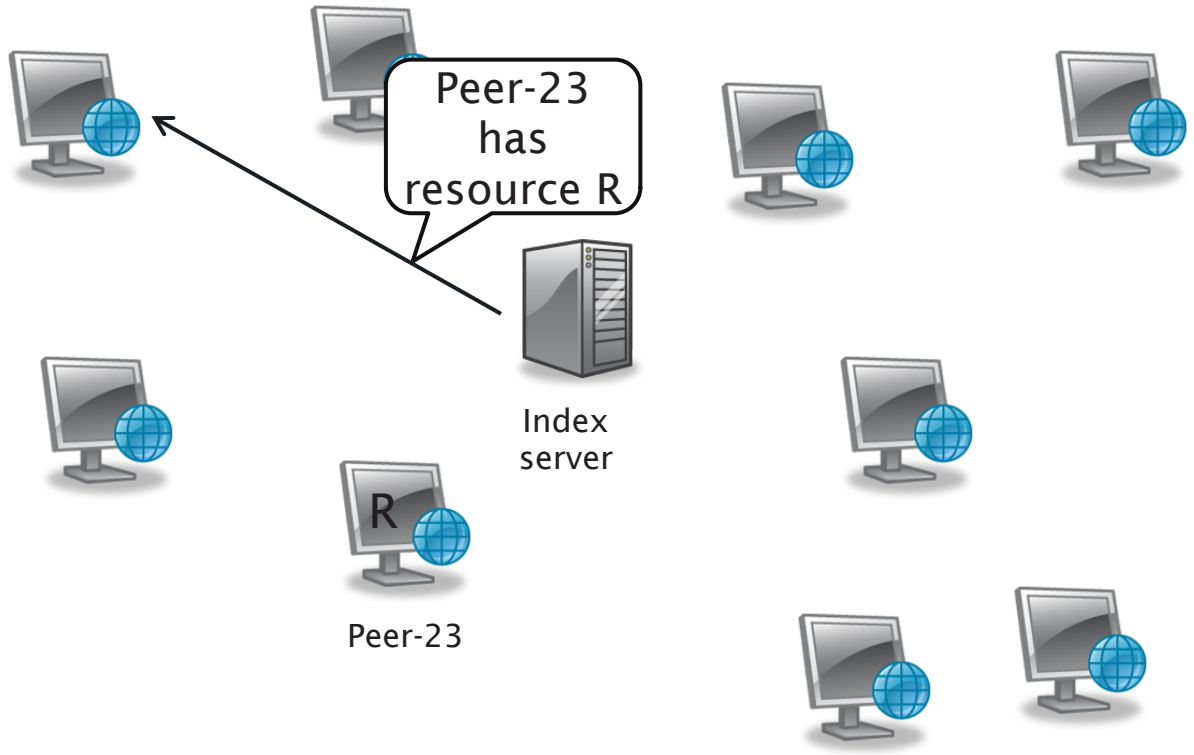
Centralised Index Management: Napster



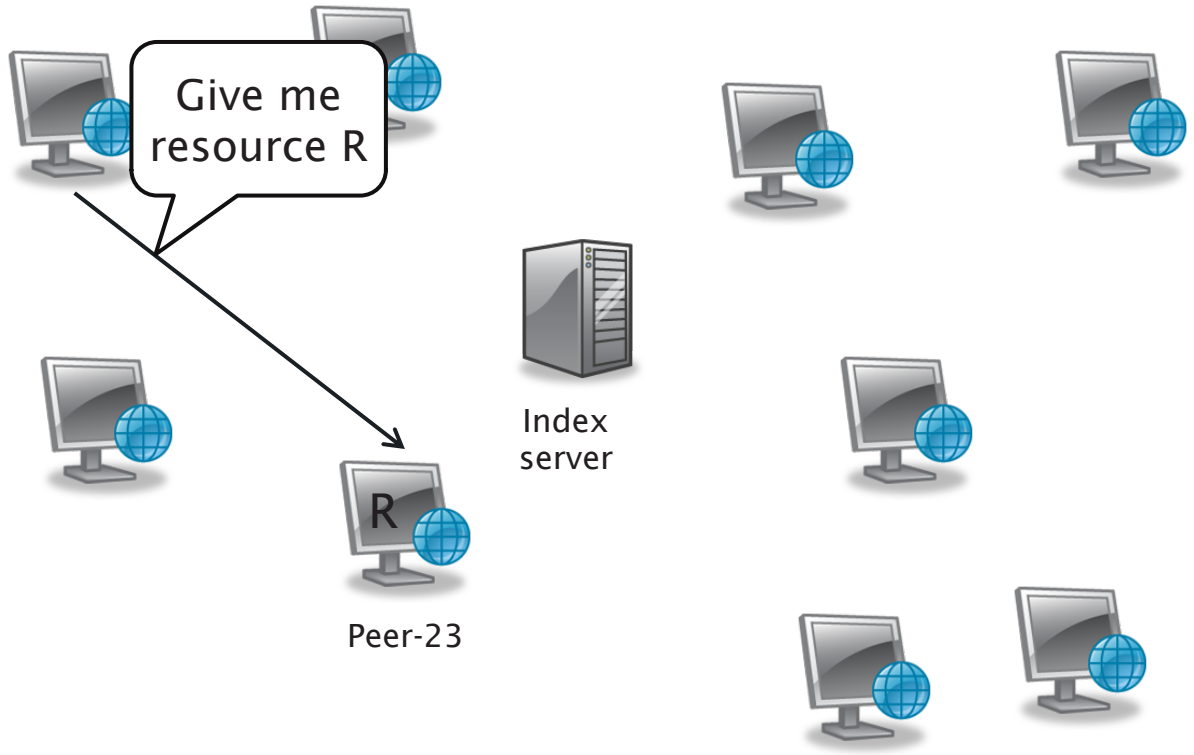
Centralised Index Management: Napster



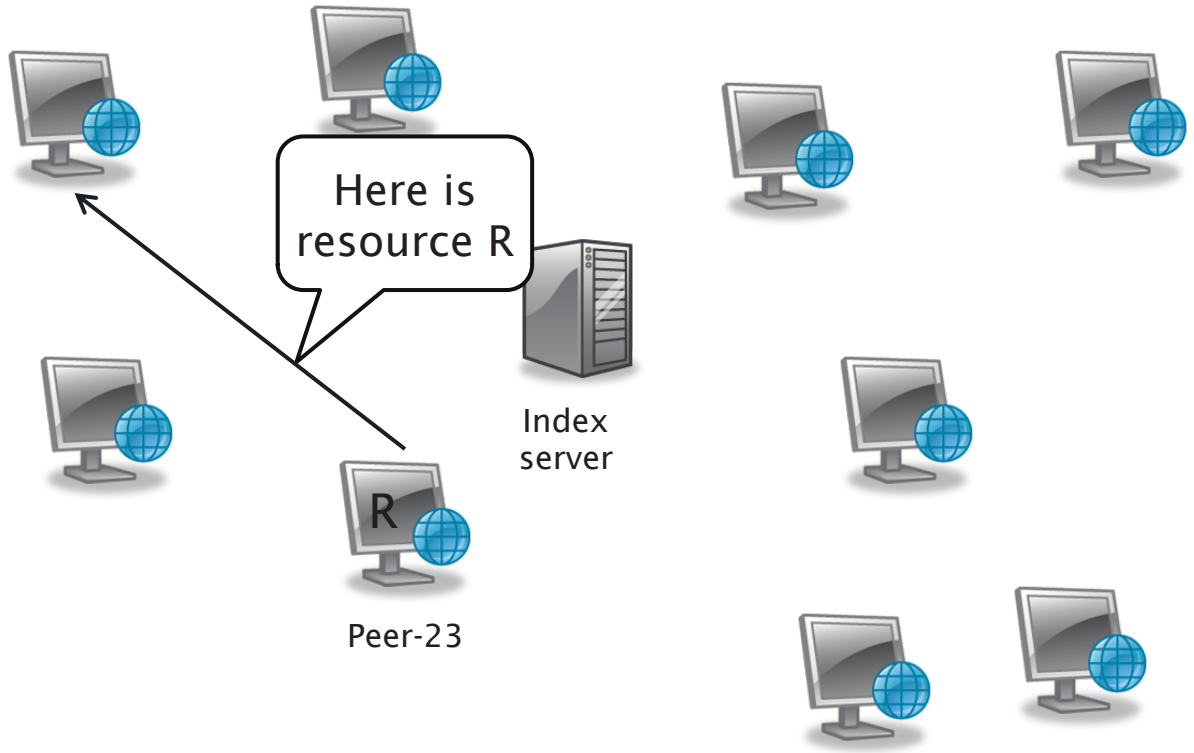
Centralised Index Management: Napster



Centralised Index Management: Napster



Centralised Index Management: Napster



Centralised Index Management

P2P networks with centralised indexes are *hybrid networks*

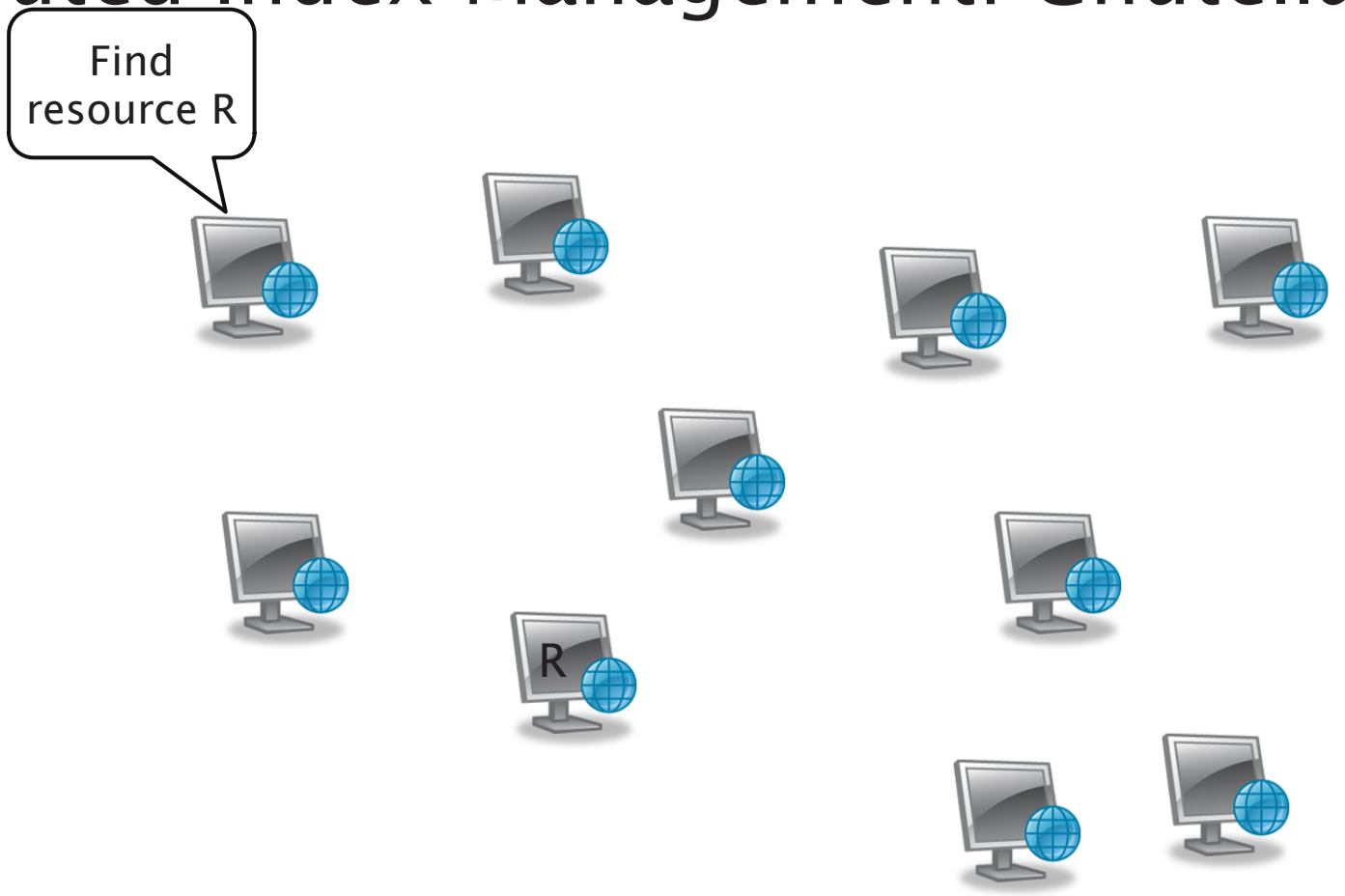
- Specialised peer for handling index
- Not all peers are equal

Centralised indexes are problematic

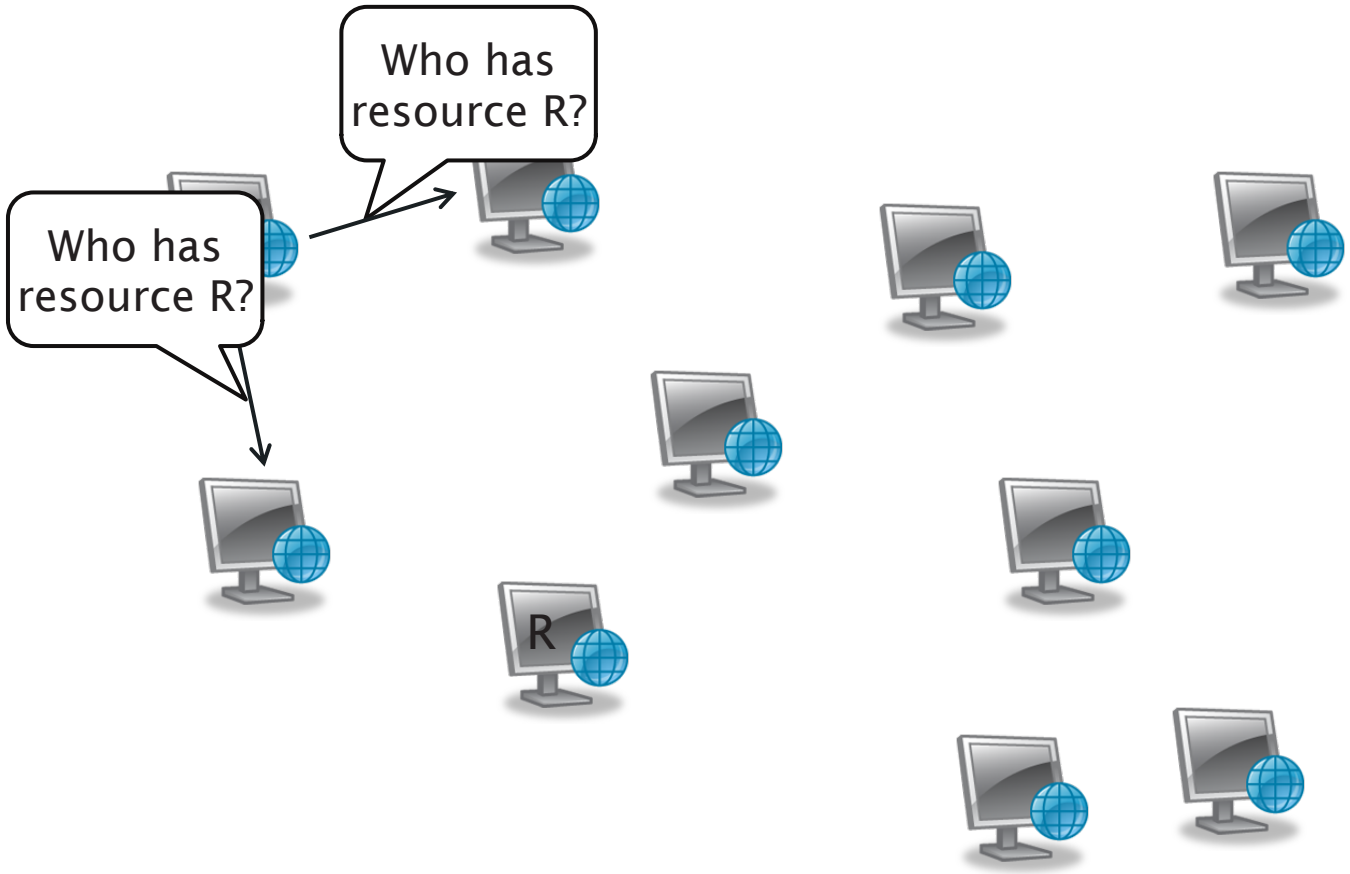
- Central point of failure (kill the index, kill the network)
- Bottleneck (limits throughput)

Distributed Index Management: Gnutella

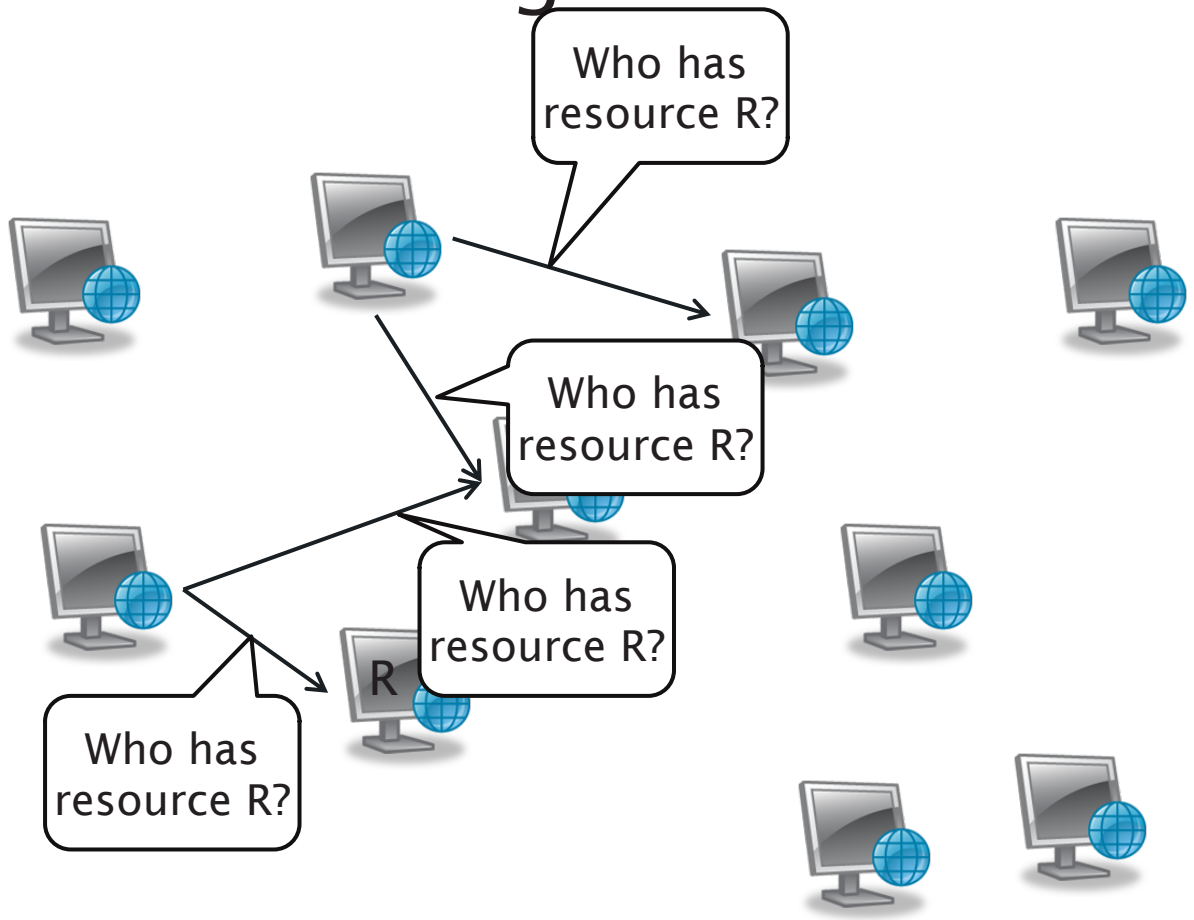
Distributed Index Management: Gnutella



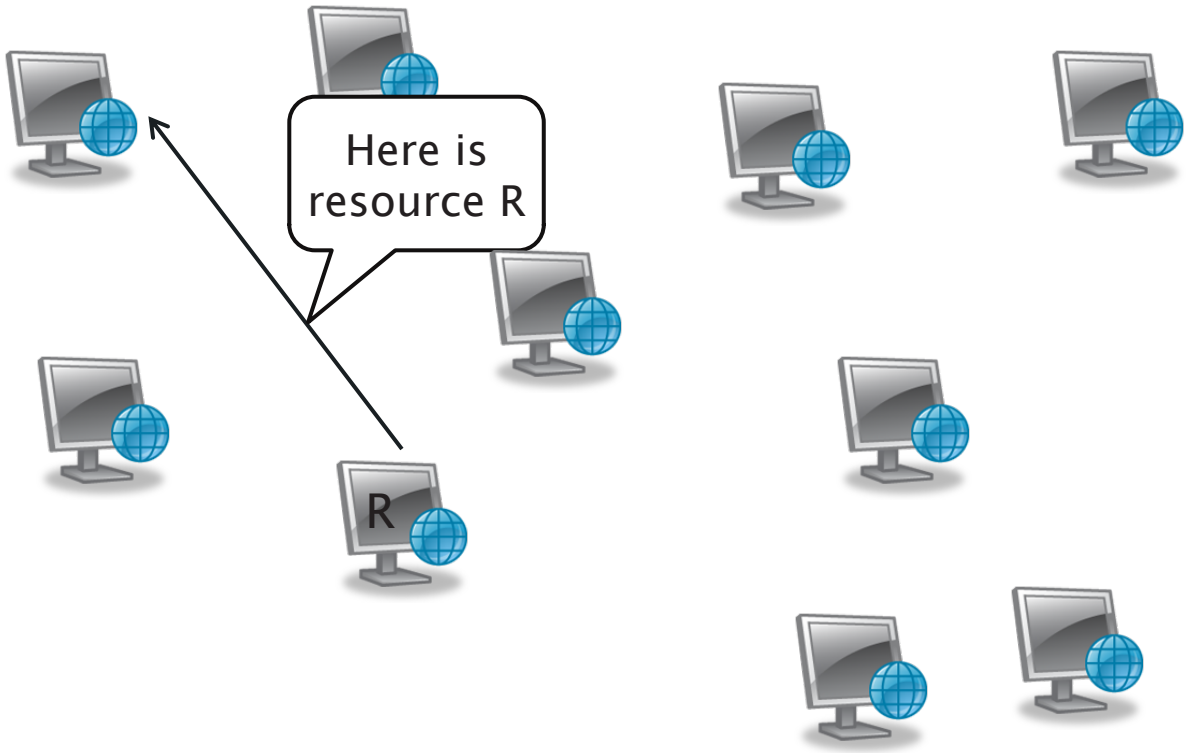
Distributed Index Management: Gnutella



Distributed Index Management: Gnutella



Distributed Index Management: Gnutella



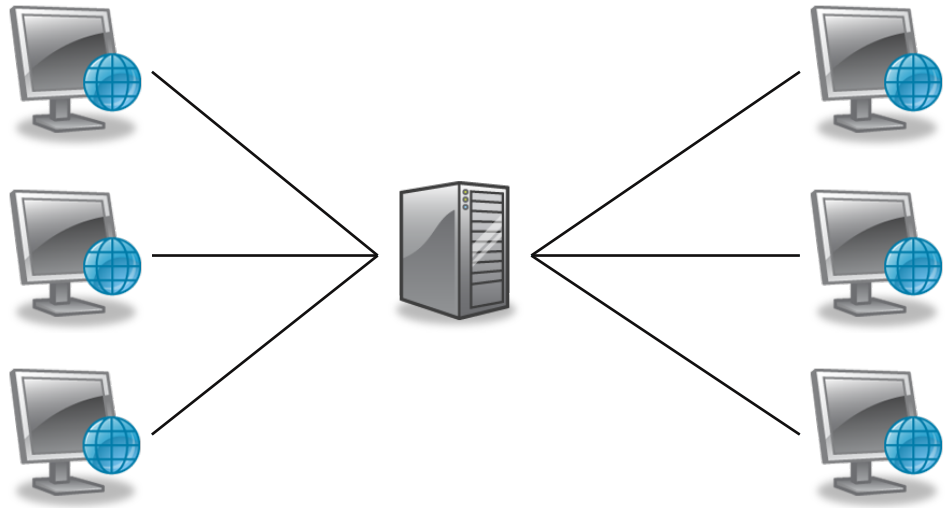
Distributed Index Management

Search neighbour using flooding or gossiping

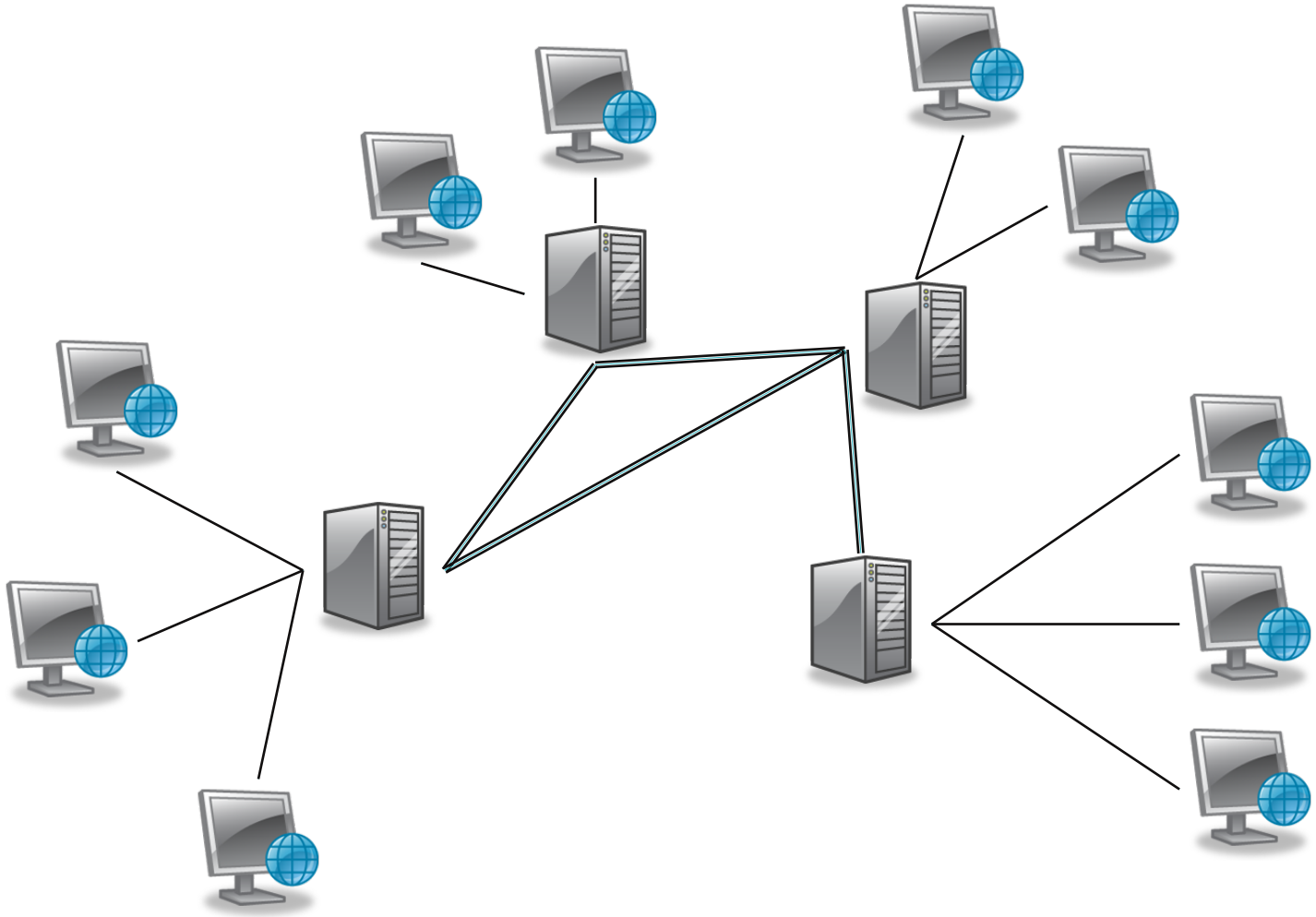
- Flooding
  - Each peer sends the request to all of its neighbours
  - Not scalable – heavy demands on network resources
  - Time-to-live (maximum number of hops) on messages
- Gossiping
  - Each peer knows all other peers in network
  - Chooses one peer to pass on request
  - Request eventually propagated to all nodes

Super-Peer Networks

- “Normal” peers index their content at super-peers
- Super-peers conduct neighbourhood search
- Single super-peer (original Napster, more or less)



Super-Peer Networks



Structured Peer-to-Peer Networks

Address scalability shortcomings of unstructured networks

Distributed Hash Tables are most popular approach

Range of applications:

- Peer-to-peer file sharing
- Content distribution networks
- Distributed file systems

UNIVERSITY OF Southampton

Distributed Hash Tables

Distributed Hash Tables

Provides a lookup service similar to a hash table

- put(key, resource)
- get(key) → resource

Hash of resource key used to locate peer holding that resource

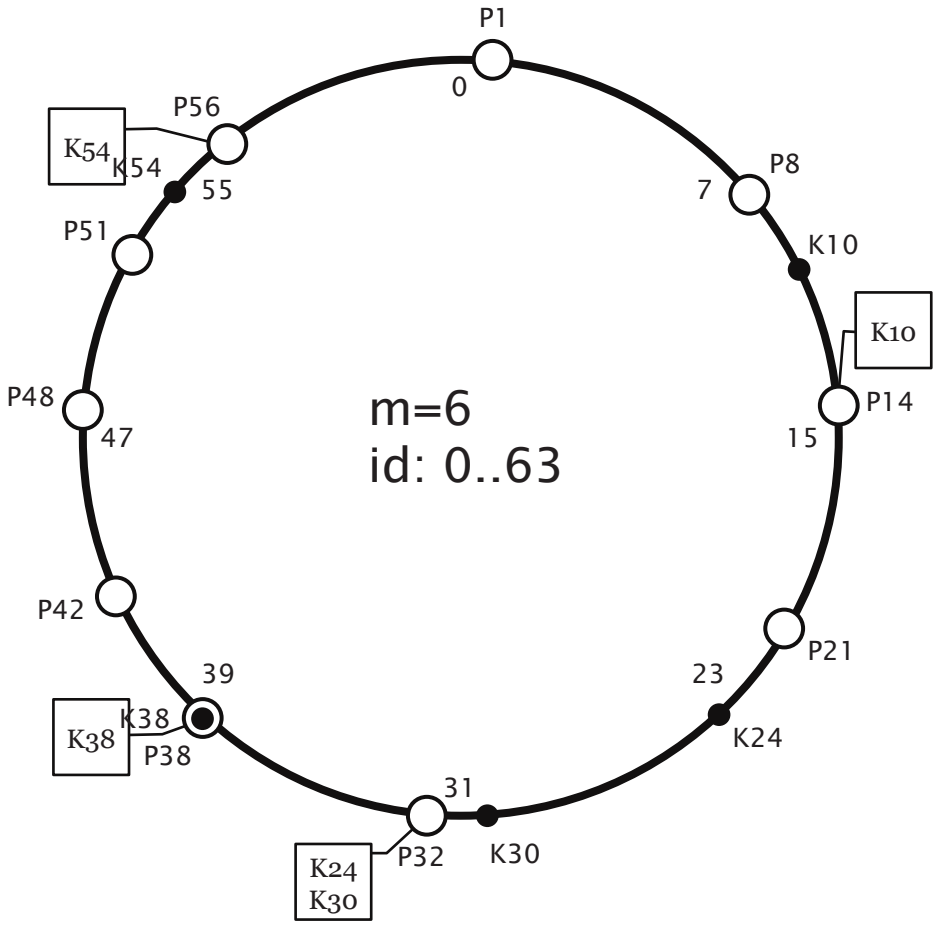
Routing protocol used to locate target peer (requires that peers share *routing information* between themselves)

Several approaches, based on different *routing geometries*:

- Ring, hypercube, tree

Chord: A ring geometry DHT

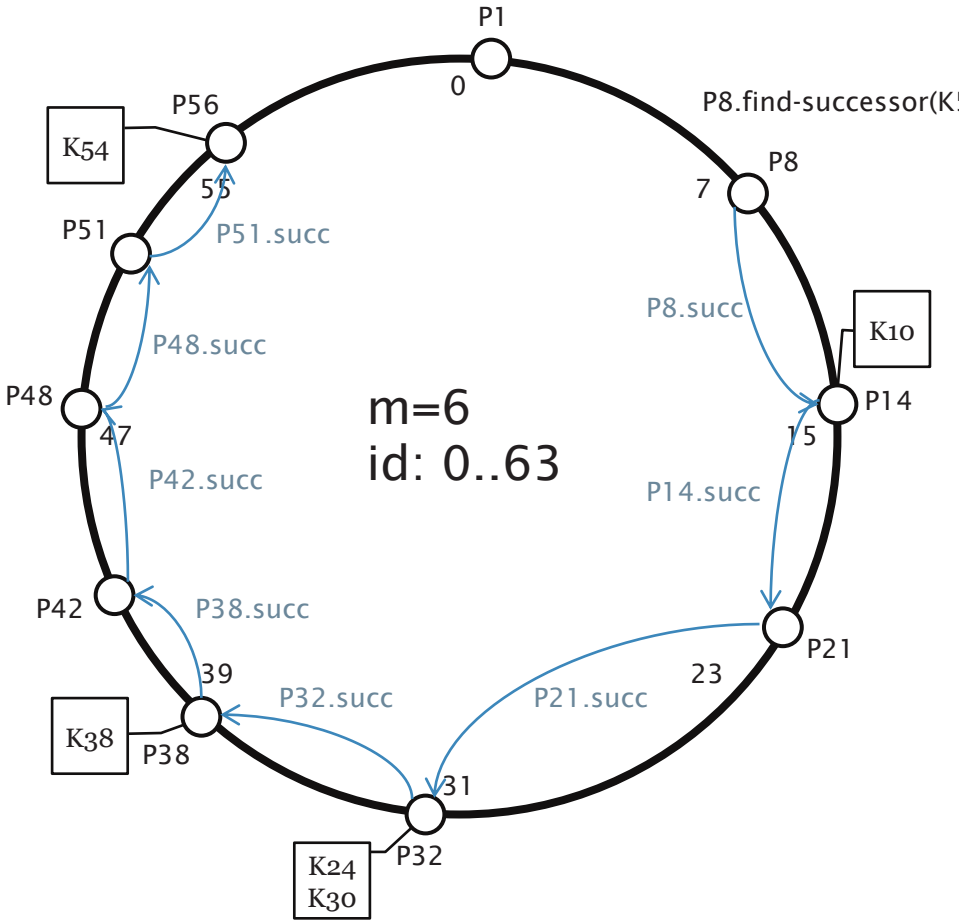
- Peers and keys are given m-bit hash identifiers
- Peers and keys are then arranged into an identifier circle with at most 2^m-1 nodes
- Keys are stored at the first peer whose identifier is greater than or equal to the key's identifier



Chord: A ring geometry DHT

- Each peer has a successor and predecessor
- Successor chain used to find peer holding resource

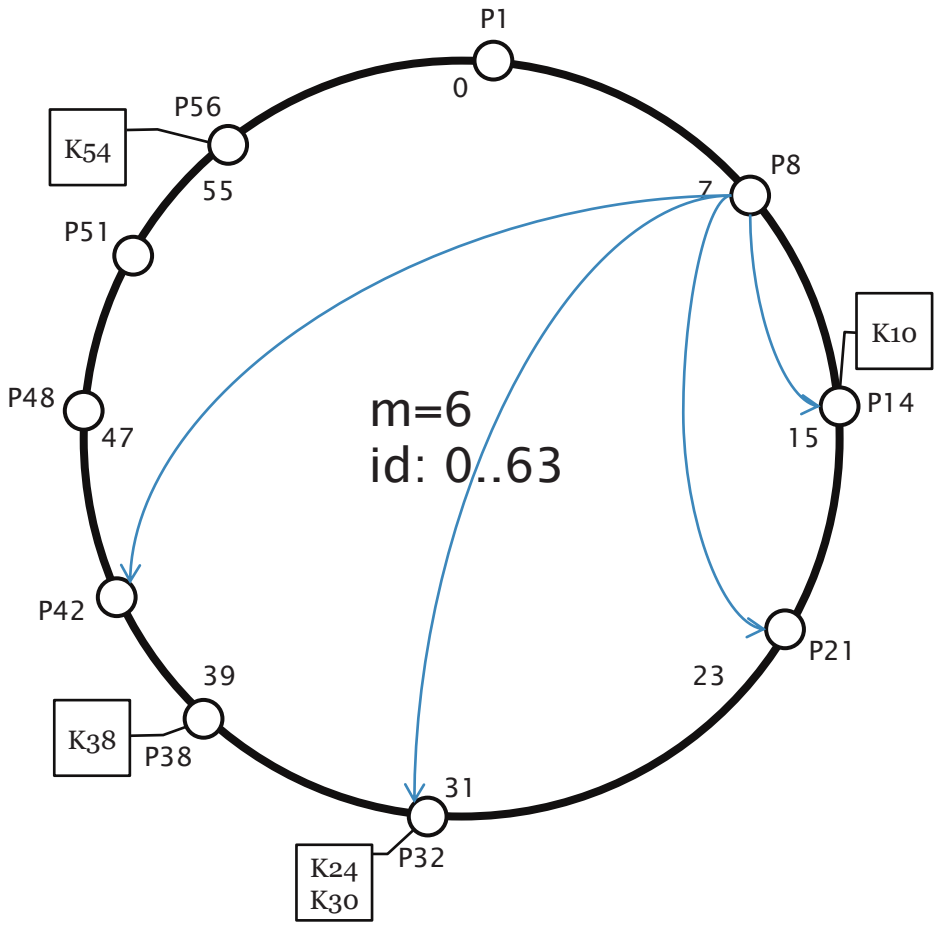
```
X.get(k) {
    peer = find-successor(k)
    resource = peer.lookup(k)
    return resource
}

X.find-successor(k) {
    if H(k) in (H(k), H(succ)]
        return succ
    else
        return succ.find-successor(k)
}
```



Chord: A ring geometry DHT

- Following successors gives linear search
- Improve using a *finger table*:
  - Contains up to m entries
  - i^{th} entry of peer n contains succ($(n+2^{i-1}) \bmod 2^m$)

P8+1	P14
P8+2	P14
P8+4	P14
P8+8	P21
P8+16	P32
P8+32	P42



UNIVERSITY OF Southampton

Peer to Peer Join Processing

PIERjoin – using DHTs for equi-joins

We wish to evaluate an equi-join query Q: $T = R \bowtie_A S$

- The tuples in R, S and T are stored in a DHT with hash function h()
- R_p, S_p, T_p are the horizontal fragments of R, S and T held by peer p of the DHT
- H_R, H_S are the sets of peers storing tuples of R, S
- H_T is the set of peers that will contain the output relation T

Three phases:

- Multicast
- Hash
- Probe/Join

Huebsch, R. et al (2003) *Querying the Internet with PIER*. Proceedings of the 29th VLDB Conference. pp.321-332.

PIERjoin: Multicast phase

At query originator, send Q to all peers in H_S and H_T

PIERjoin: Hash phase

```
foreach peer p in H_R that received Q in parallel do
    foreach tuple r in R_p do
        put(h(A), r)

foreach peer p in H_S that received Q in parallel do
    foreach tuple s in S_p do
        put(h(A), s)
```

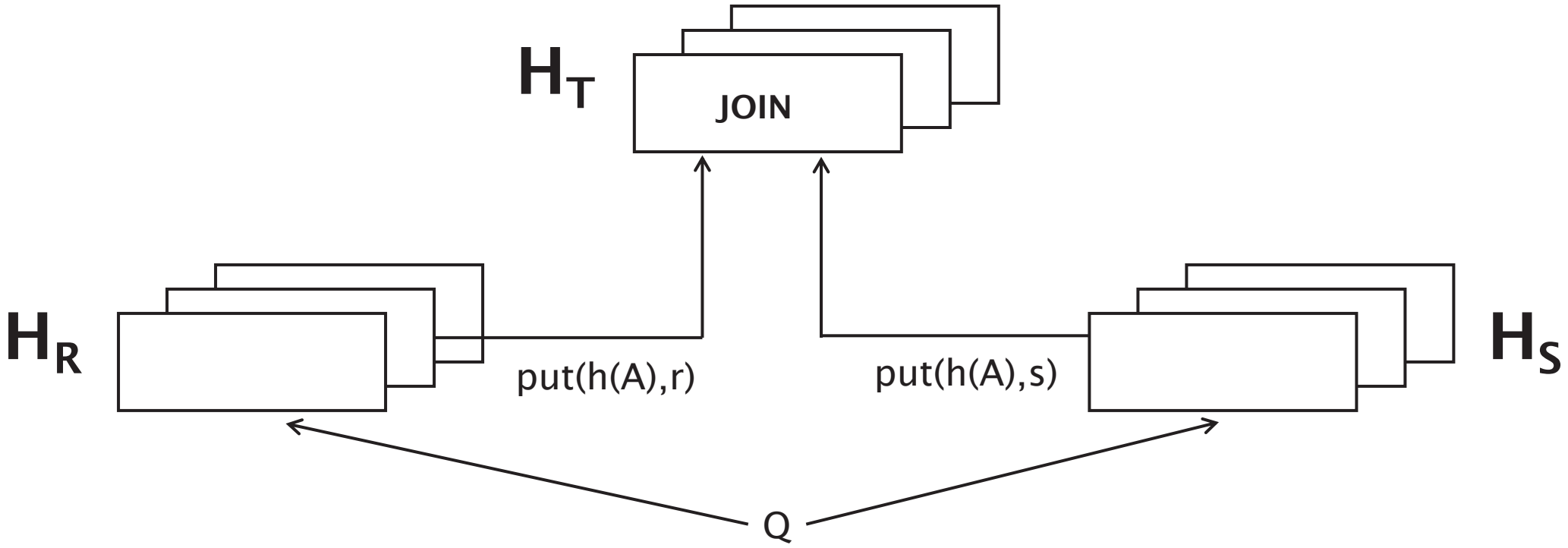
PIERjoin: Probe/join phase

```
foreach peer p in HT in parallel do
    if a new tuple i has arrived then
        if i is a tuple from R then
            probe for s tuples in Sp using h(A)
            Tp <- i ⋈ s
        else
            probe for r tuples in Rp using h(A)
            Tp <- r ⋈ i
```

PIERjoin

We've seen something similar before: *Repartitioned Join (AKA parallel hash join)*

(this is a pipelined variant without the final union)



UNIVERSITY OF Southampton

Summary

Comparison of Peer-to-Peer Systems

Requirement	Unstructured	Super-Peer	Structured
Autonomy	Low	Moderate	Low
Query Expressivity	High	High	Low
Efficiency	Low	High	High
Quality of Service	Low	High	High
Fault-Tolerance	High	Low	High
Security	Low	High	Low


That's all Folks!