Distributed Big Data Management Systems

Ioana Manolescu
INRIA Saclay & Ecole Polytechnique
ioana.manolescu@inria.fr
http://pages.saclay.inria.fr/ioana.manolescu/

M2 Data and Knowledge
Université de Paris Saclay

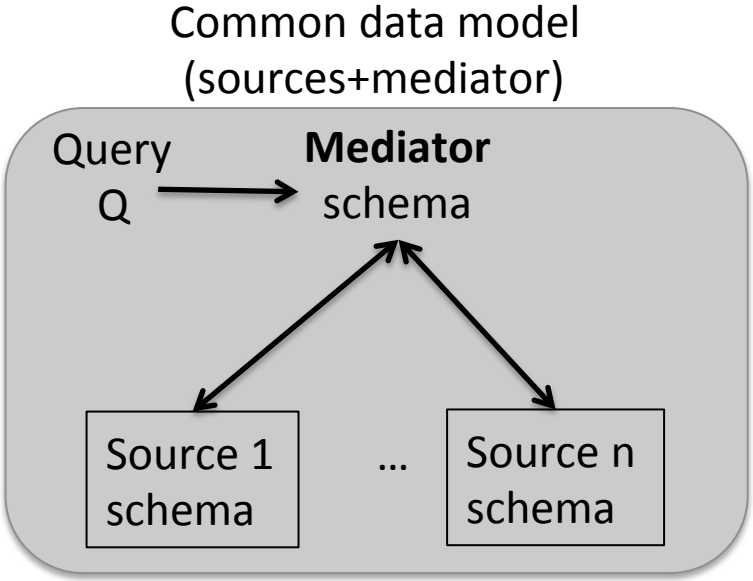
Dimensions of distributed data management systems

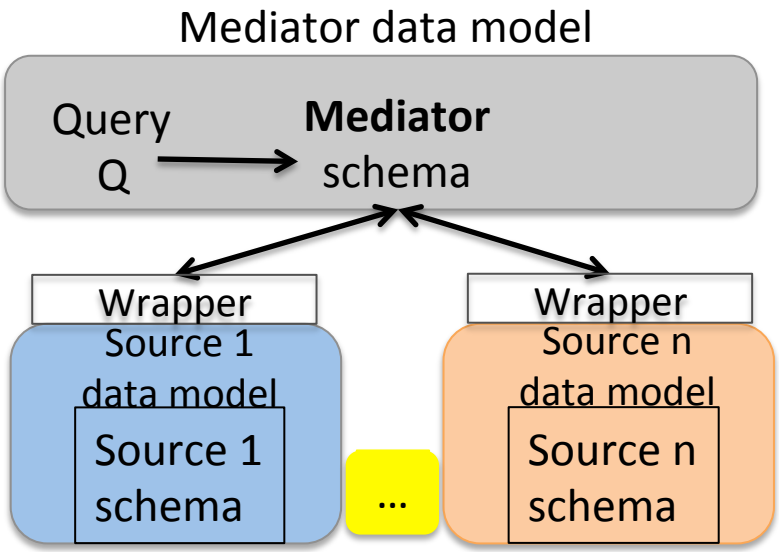
- **Data model**:
  - Relations, trees (XML, JSON), graphs (RDF, others…), nested relations
  - Query language
- **Heterogeneity** (DM, QL): none, some, a lot
- **Scale**: small (~10-20 sites) or large (~10.000 sites)
- **ACID** properties
- **Control**:
  - Single master w/complete control over N slaves (Hadoop/HDFS)
  - Sites publish independently and process queries as directed by single master/*mediator*
  - Many-mediator systems, or peer-to-peer (P2P) with *super-peers*
  - Sites completely independent (P2P)

MEDIATOR SYSTEMS

Mediator systems

- A set of **data sources**, of the same or different data model, query language; source schemas
- A **mediator** data model and mediator schema
- Queries are asked against the mediator schema

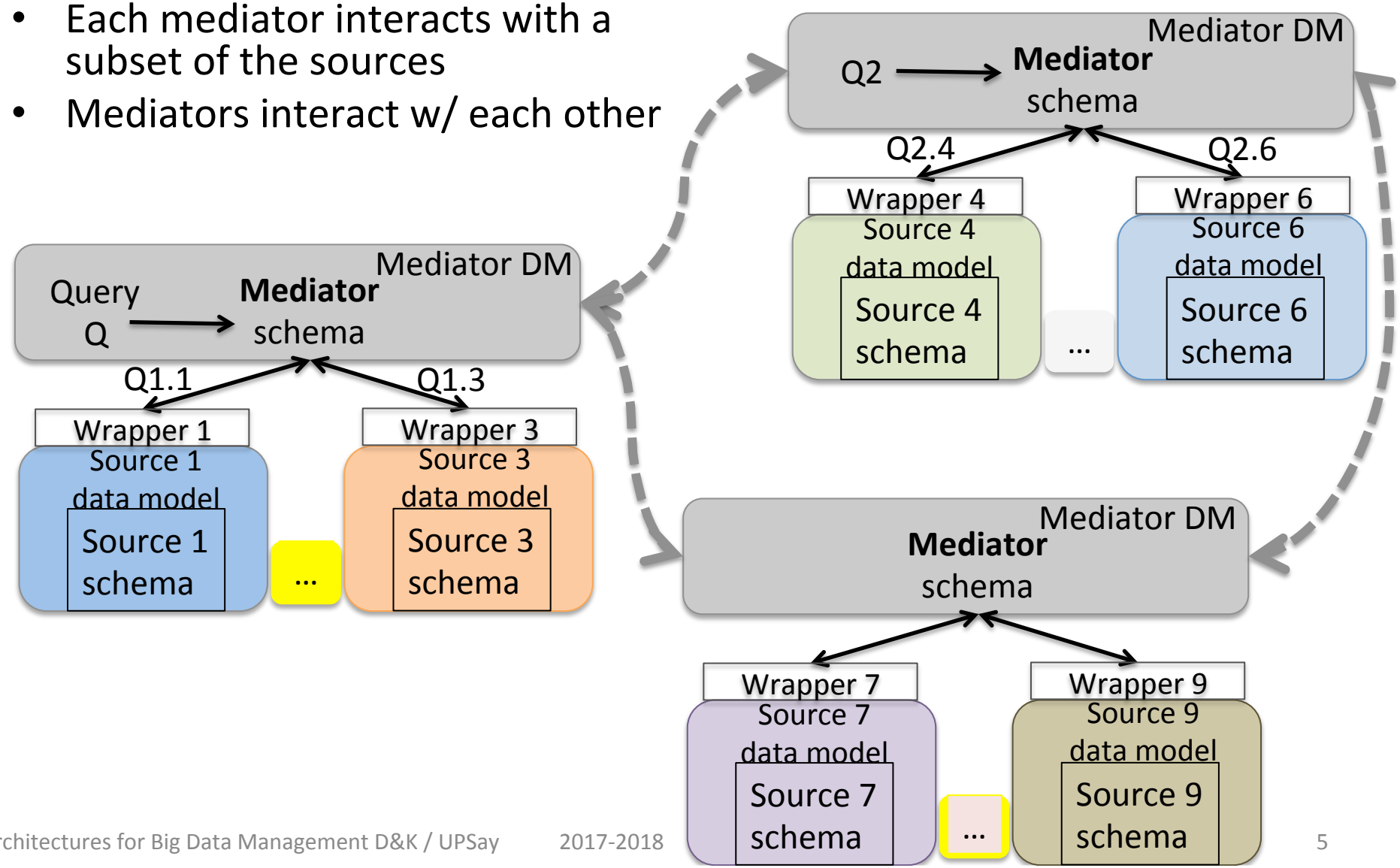




- **ACID**: mostly read-only; **size**: small
- **Control**: Independent publishing; mediator-driven integration

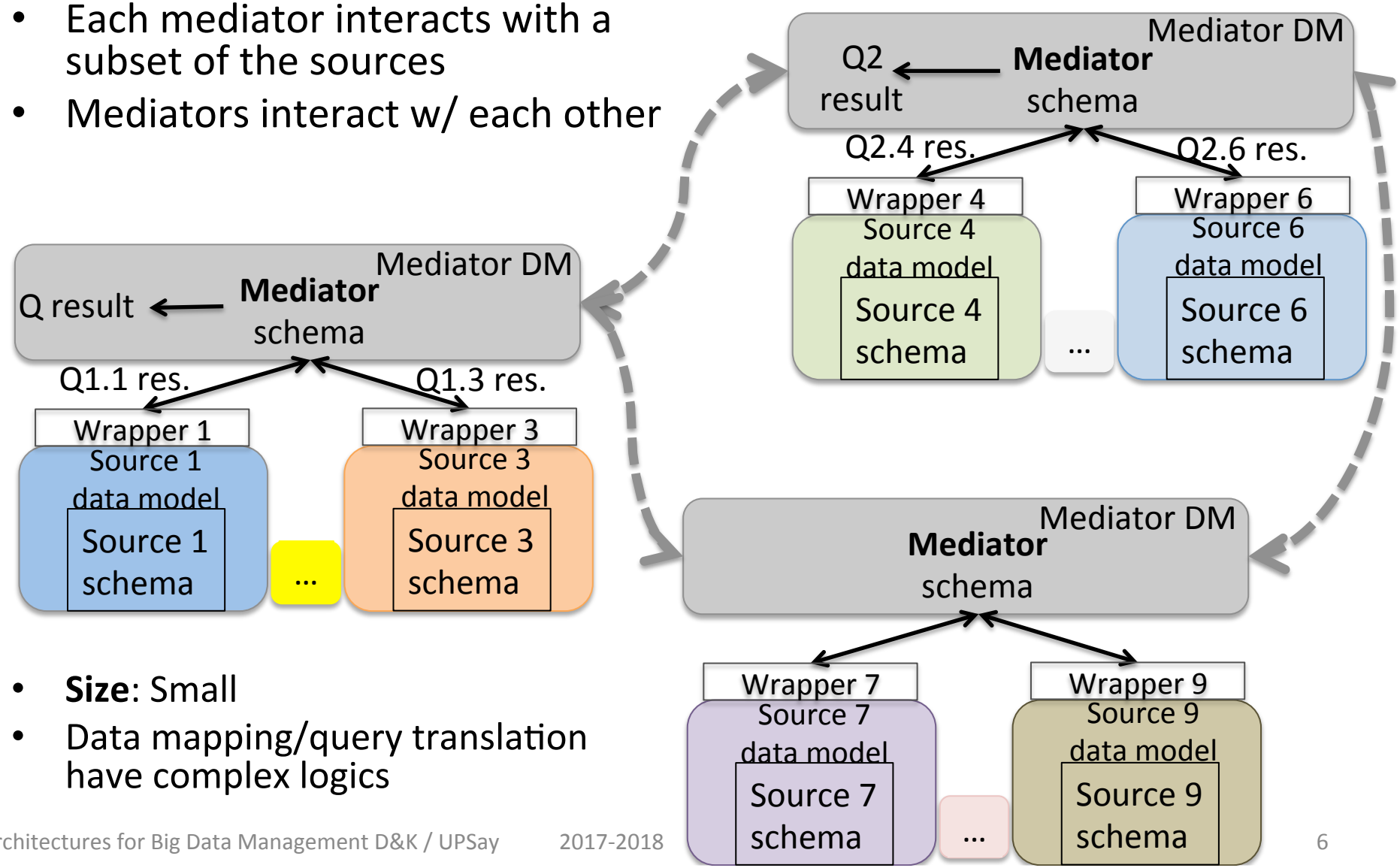
Many-mediator systems

- Each mediator interacts with a subset of the sources
- Mediators interact w/ each other



Many-mediator systems

- Each mediator interacts with a subset of the sources
- Mediators interact w/ each other



- **Size**: Small
- Data mapping/query translation have complex logics

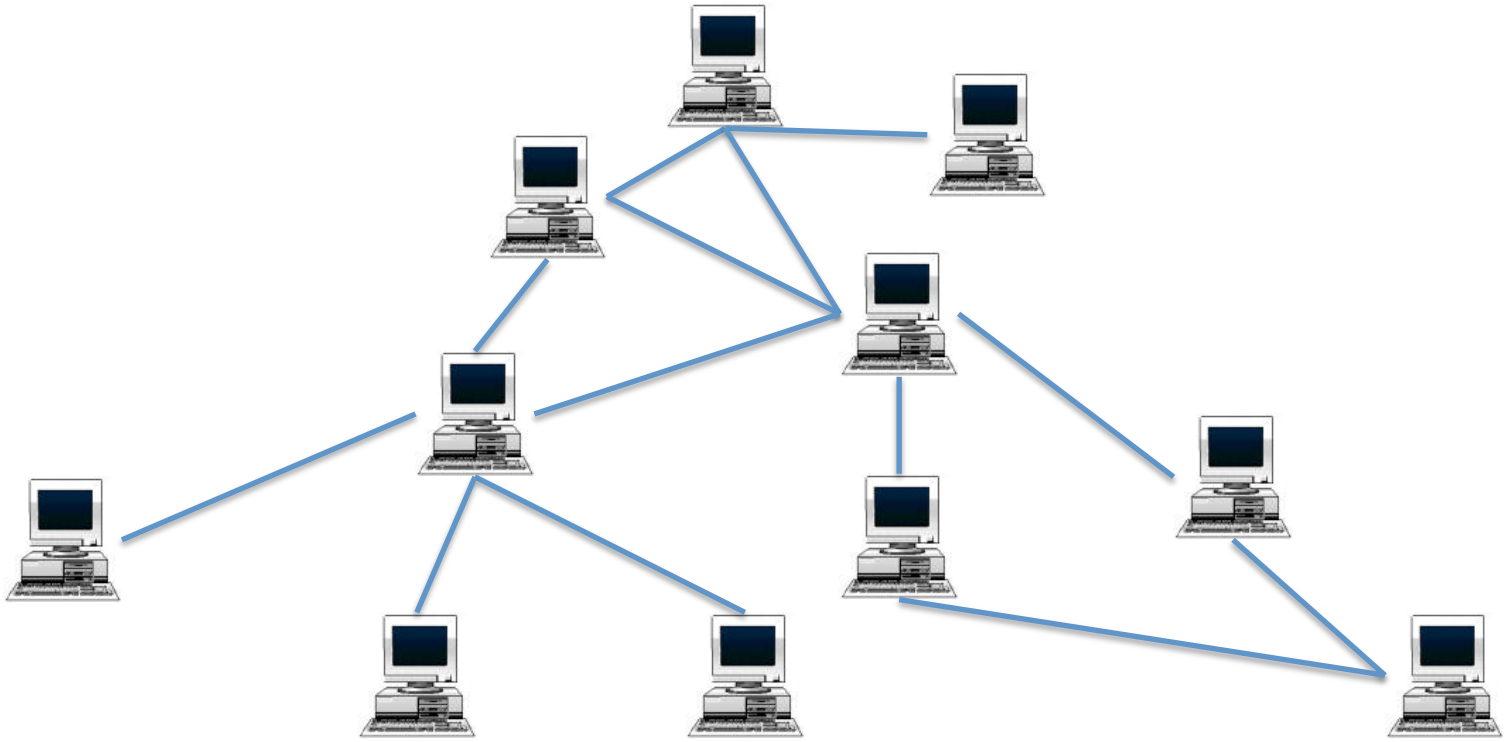
PEER-TO-PEER SYSTEMS

Peer-to-peer architectures

- Idea: easy, **large-scale** sharing of data with **no central point of control**
- **Advantages**:
  - Distribute work; preserve peer independence
- **Disadvantages**:
  - Lack of control over peers which may leave or fail → need for mechanisms to cope with peers joining or leaving (*churn*)
  - Schema unknown in advance; need for data discovery
- Two variants:
  - Unstructured P2P networks
    - Each peer is free to connect to other peers;
    - Variant: super-peer networks
  - Structured P2P networks
    - Each peer is connected to a set of other peers determined by the system

Unstructured P2P networks

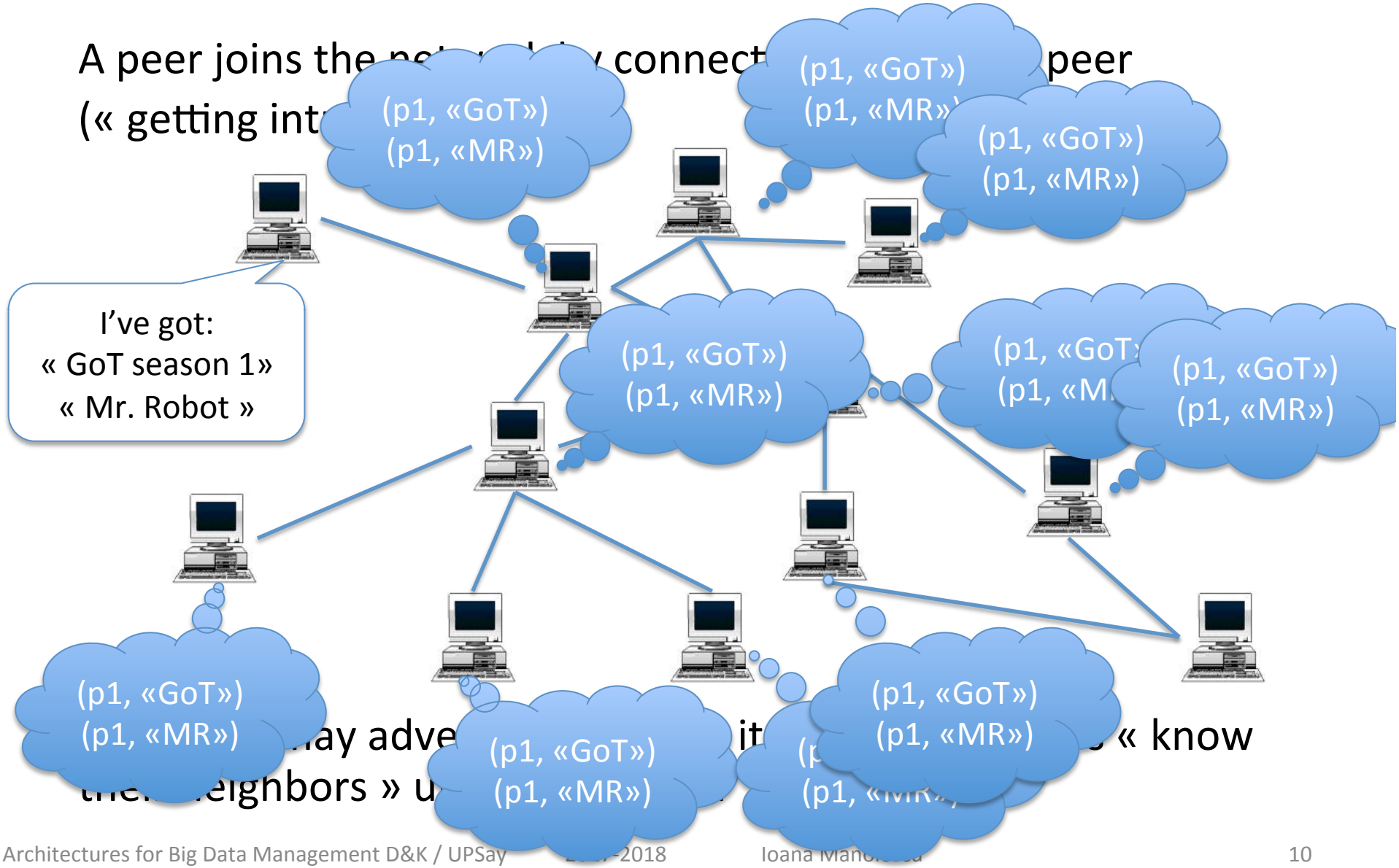
A peer joins the network by connecting to another peer
(« getting introduced »)



Each peer may advertise data that it publishes→peers « know their neighbors » up to some level

Unstructured P2P networks

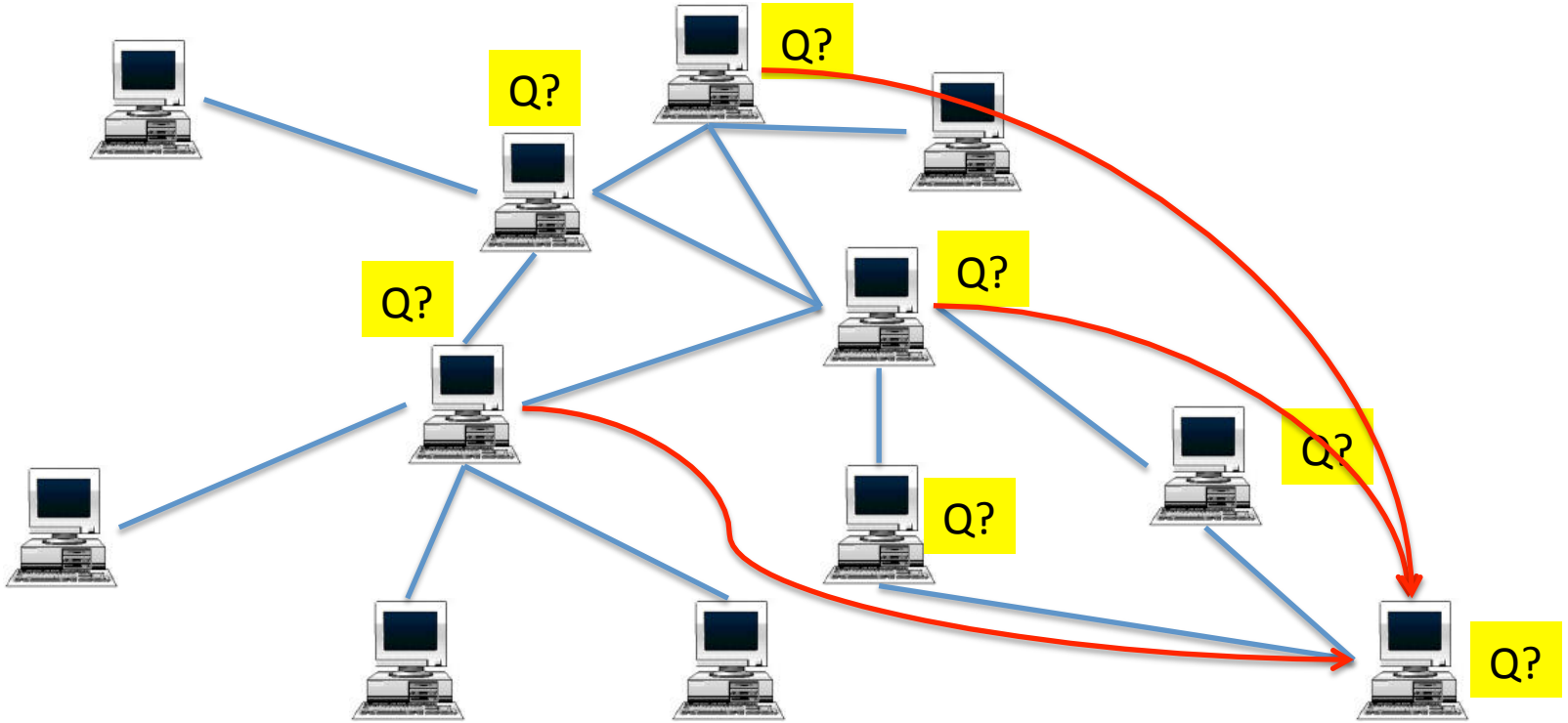
A peer joins the net… connect… peer
(« getting int…



…ay adve… it… « know
the… eighbors » u…

Unstructured P2P networks

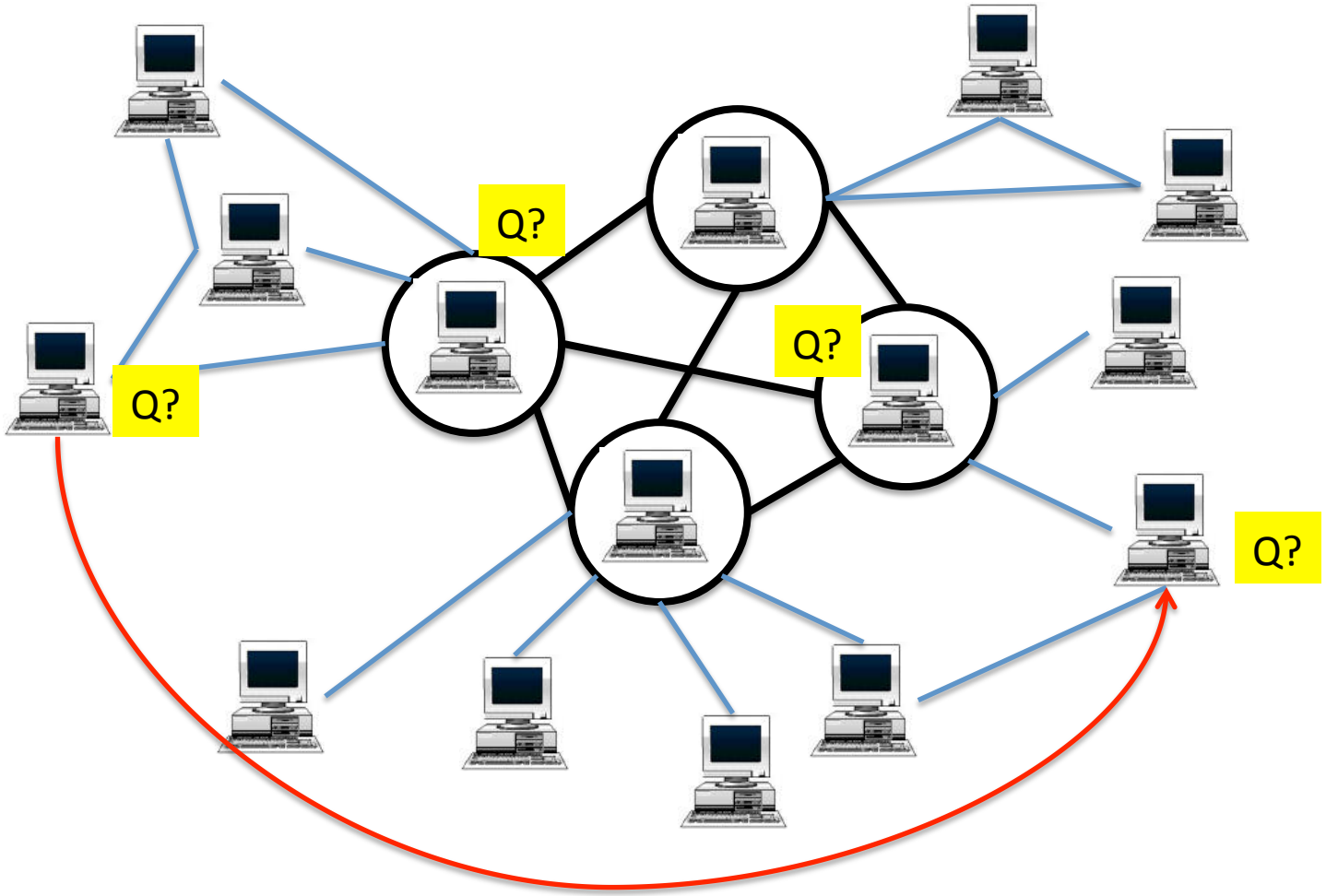
Queries are evaluated by propagation from the query peer to its neighbors and so on recursively (flooding)



To avoid saturating the network, queries have TTL (time-to-live)

This may lead to missing answers → a. replication; b. superpeers

Unstructured P2P with superpeers

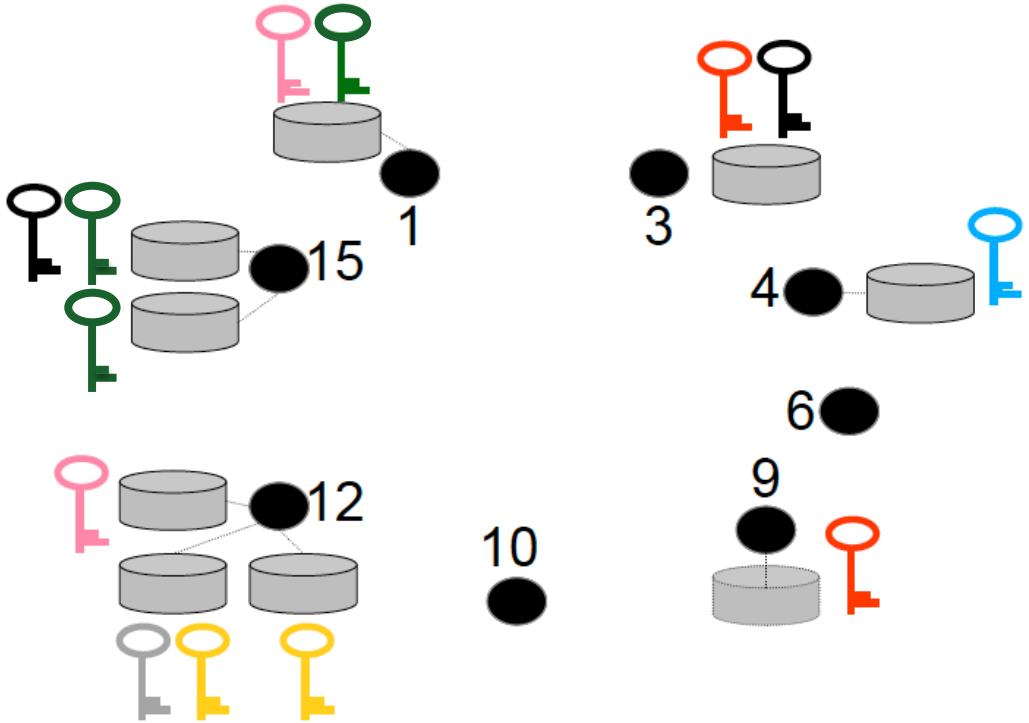


- Small subset of superpeers all connected to each other
- Specialized by data domain, e.g. [Aa—Bw], [Ca—Dw], … or by address space
- Each peer is connected at least to a superpeer, which routes the peer's queries

Catalog construction (indexing) in structured P2P networks

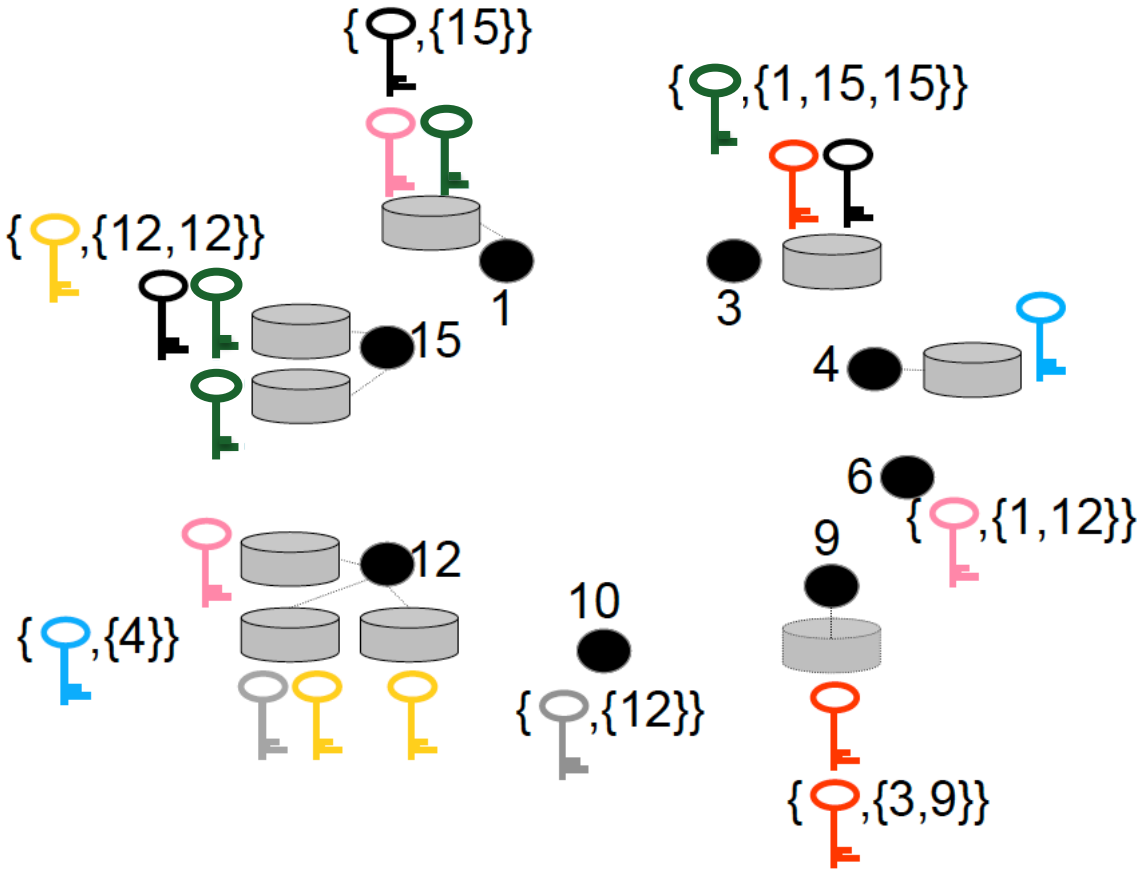
- Peers form a **logical address space** 0… 2^k-1
  - Some positions may be vacant
- The **catalog** is built as a set of key-value pairs
  - *Key*: expected to occur in search queries, e.g. «GoT », « Mr Robot »
  - **Value**: the address of content in the network matching the key, e.g. « peer5/Users/a/movies/GoT »
- A **hash function** is used to map every *key* into the address space; this distributes (*key*, value) pairs
  - H(*key*)=n → the (*key*, value) is sent to peer n
  - If n is unoccupied, the next peer in logical order is chosen
- The catalog is distributed across the peers
  (also the name: **distributed hash table, DHT**)

Catalog construction (indexing) in structured P2P networks



key	hash
	6
	2
	7
	1
	3
	8
	14

Catalog construction (indexing) in structured P2P networks



key	hash
	6
	2
	7
	1
	3
	8
	14

Searching in structured P2P networks

Locate all items characterized by  ?
Hash()=6
Peer 6 knows all the locations

Locate all items characterized by ?
Hash()=14
Peer 15 knows all the locations

How do we find peers 6 and 15?

Connections between peers in structured P2P networks

A peer's connections are dictated by the network organization and the logical address of each peer in the space 0… 2^k-1

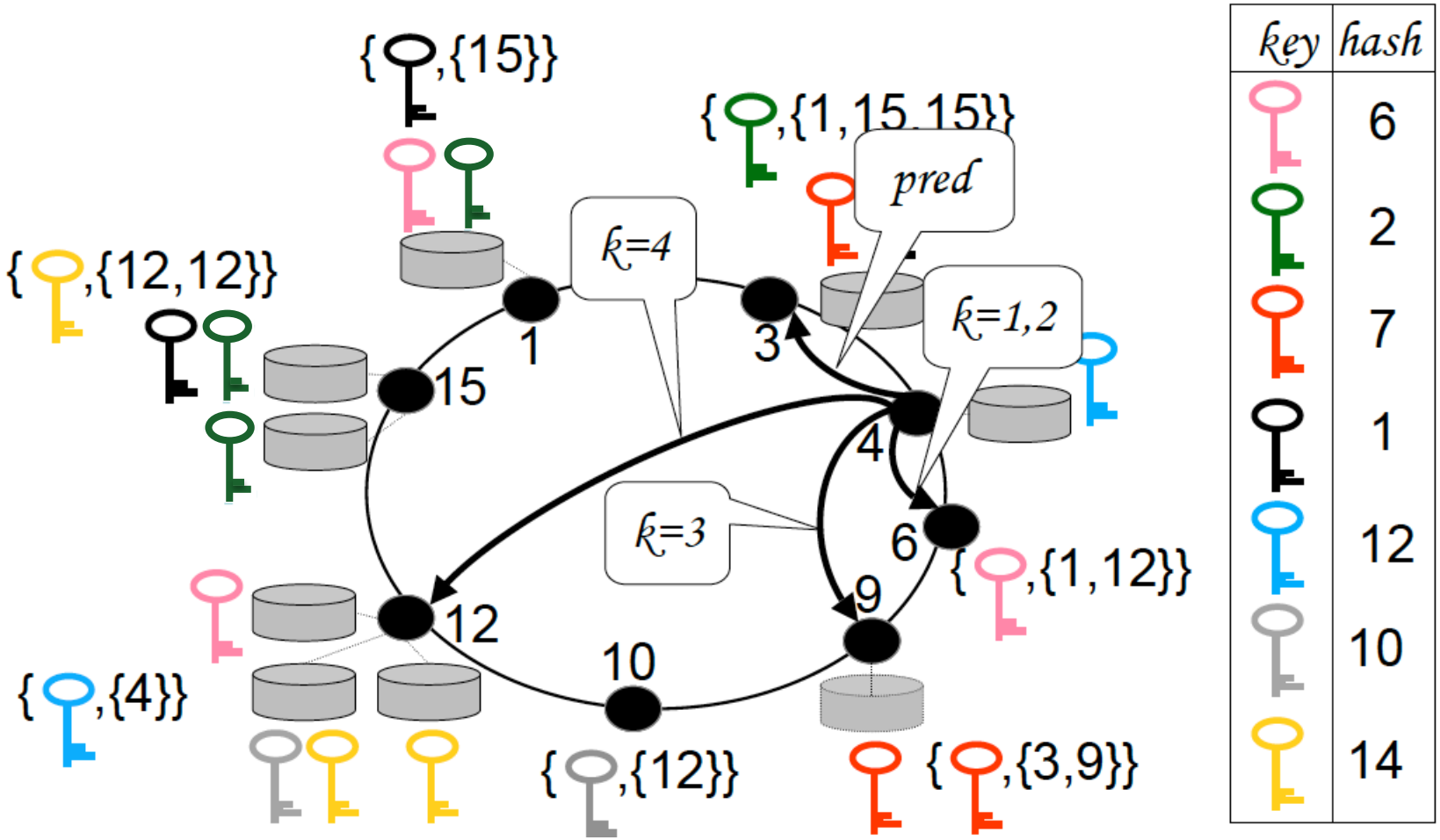
Example: Chord (MIT, most popular)

Each peer n is connected to

- *n+1, n+2, …, n+2^{k-1}*, or to the first peer *following* that position in the address space;
- The *predecessor of n*

The connections are called *fingers*

Connections between peers in Chord

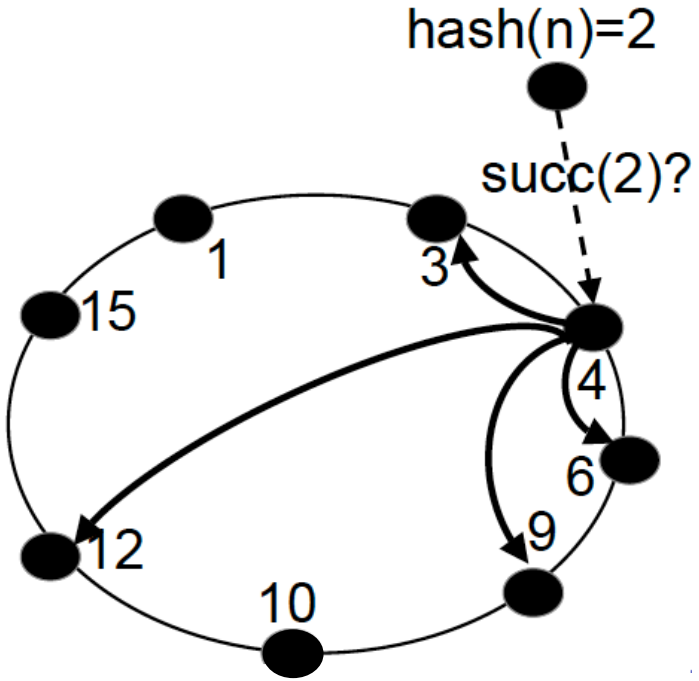


Peers joining in Chord

To join, a peer n must know (any) peer n' already in the network

Procedure **n.join(n')**:

```
s = n'.findSuccessor(n);
buildFingers(s);
successor=s;
```

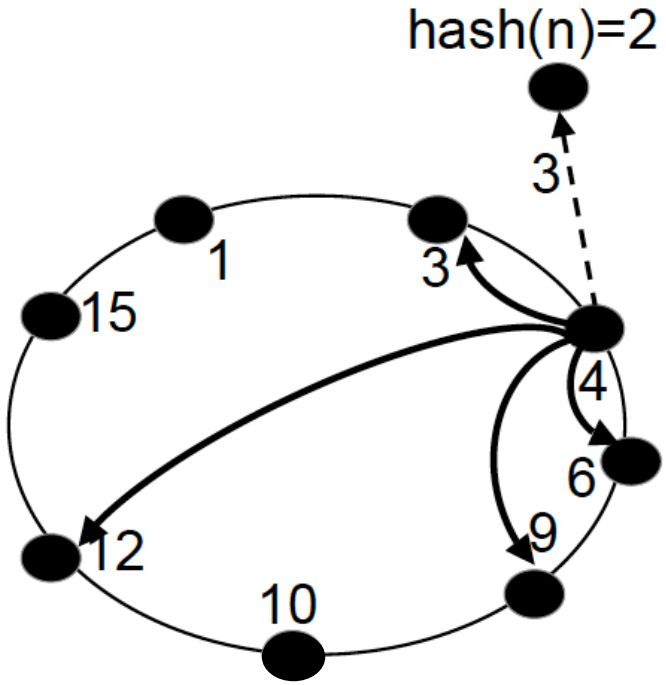


Peers joining in Chord

To join, a peer n must know (any) peer n' already in the network

Procedure **n.join(n')**:

s = n'.findSuccessor(n);
buildFingers(s);
successor=s;



Peers joining in Chord

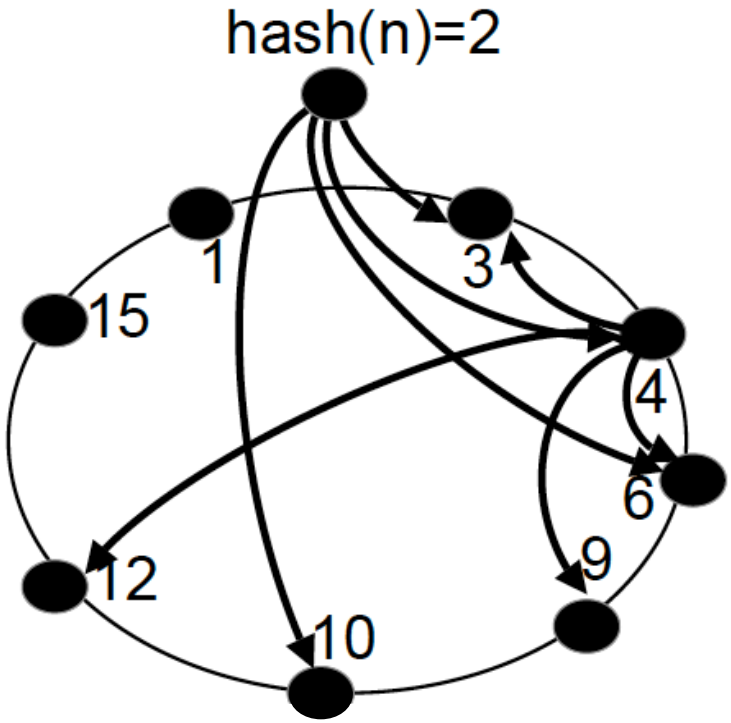
To join, a peer n must know (any) peer n' already in the network

Procedure **n.join(n')**:

```
s = n'.findSuccessor(n);
buildFingers(s);
successor=s;
```

If 3 had some key-value pairs for the key 2, 3 gives them over to 2

The network is not *stabilized* yet…



Network stabilization in Chord

Each peer periodically runs stabilize()

n.stabilize():

x = n.succ().pred()

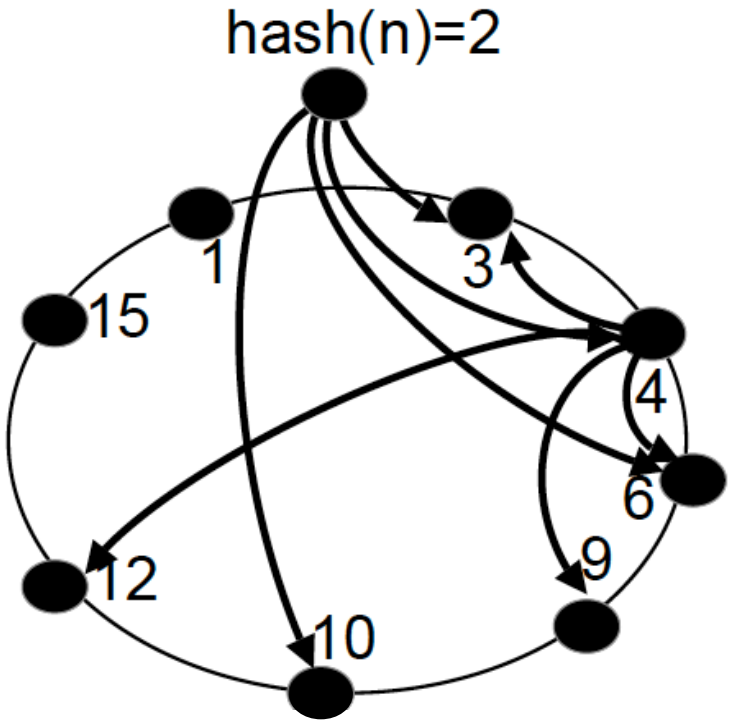
if (n < x < succ) then succ = x;

succ.notify(n)

n.notify(p):

if (pred < p < n)

then pred = p



Network stabilization in Chord

First stabilize() of 2: 3 learns its new predecessor

n.stabilize():

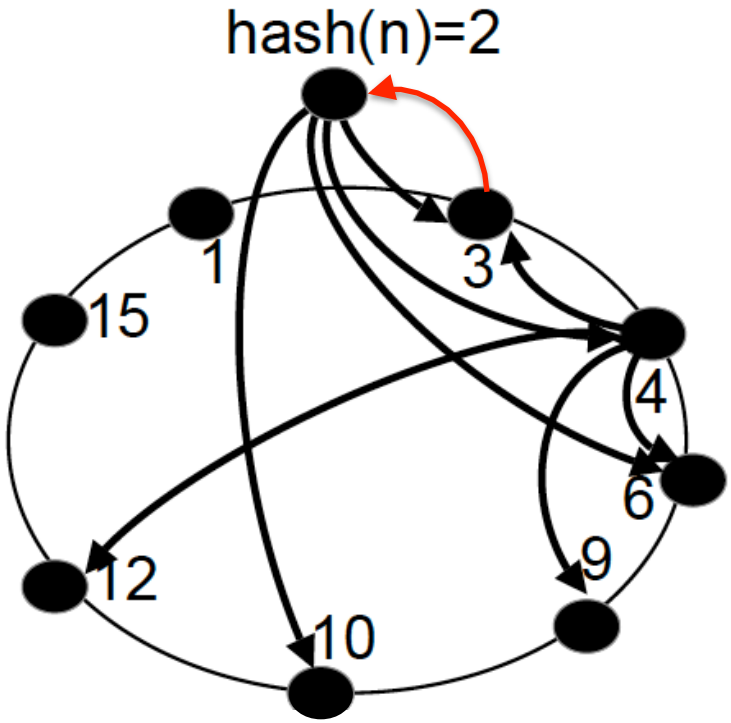
x = n.succ().pred()

if (n < x < succ) then succ = x;

succ.notify(n)

n.notify(p):

if (pred < p < n)
then pred = p



Network stabilization in Chord

First stabilize() of 1: 1 and 2 connect

n.stabilize():

x = n.succ().pred()

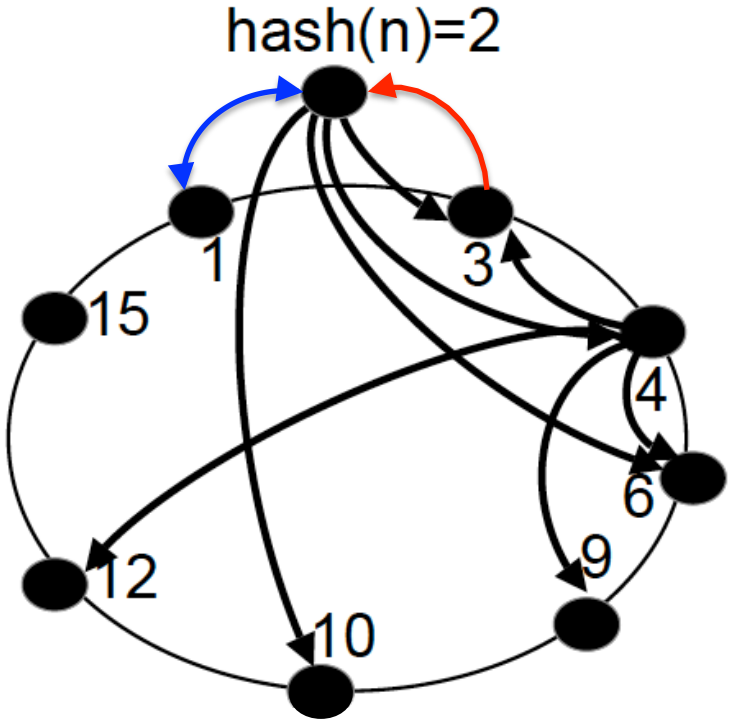
if (n < x < succ) then succ = x;

succ.notify(n)

n.notify(p):

if (pred < p < n)

then pred = p



Peer leaving the network

- The peer leaves (with some advance notice, « in good order »)

- Network adaptation to peer leave:
  - (key, value) pairs: those of the leaving peer are moved to its successor
  - Routing: P notifies successor and predecessor, which reconnect "over P"

Peer failure

- Without warning
- In the absence of replication, the (key, value) pairs held on P are lost
  - Peers may also re-publish periodically

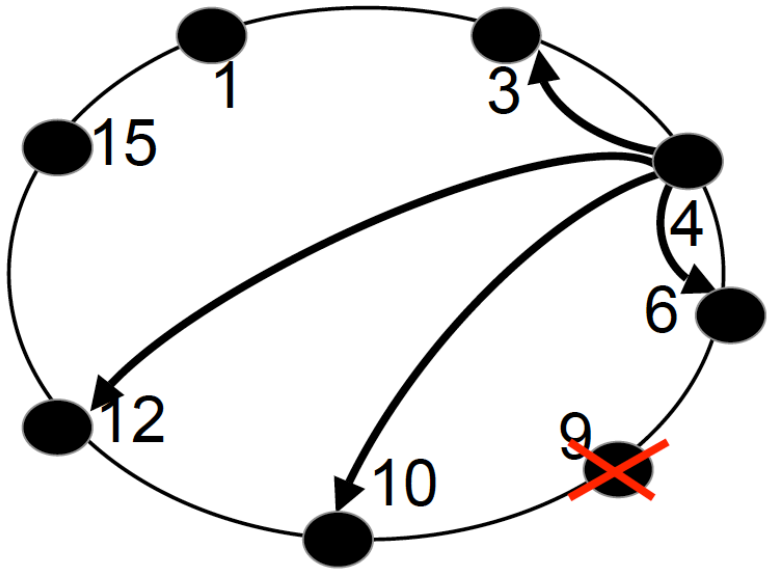
Example Running stab(), 6 notices 9 is down

6 replaces 9 with its next finger 10 →

all nodes have correct successors, but fingers are wrong

Routing still works, even if a little slowed down

Fingers must be recomputed



Peer failure

Chord uses successors to adjust to any change

- Adjustment may « slowly propagate » along the ring, since it is relatively rare

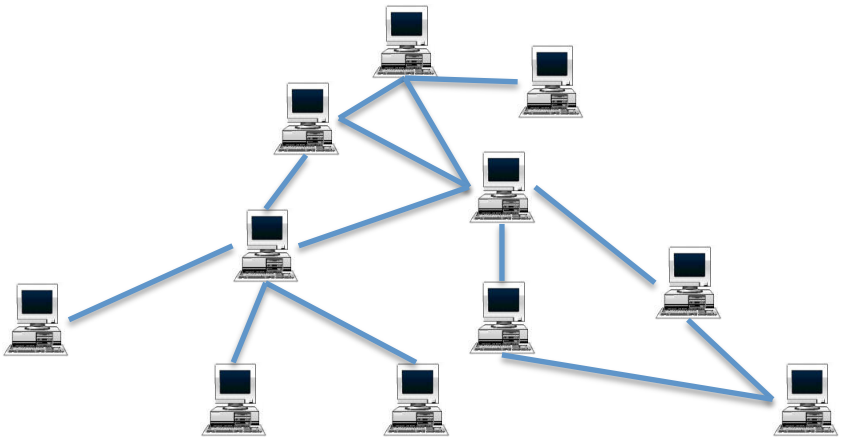
To prevent erroneous routing due to successor failure, each peer maintains a list of its *r* direct successors (*2 $log_2 N$*)

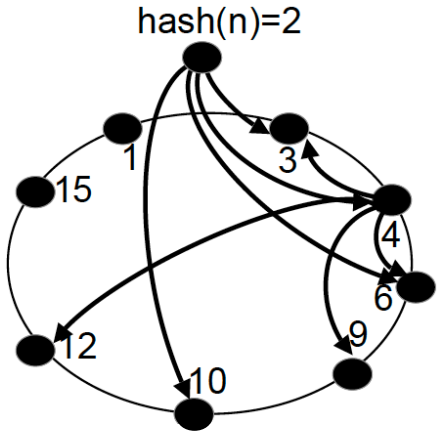
When the first one fails, the next one is used...

All *r* successors must fail simultaneously in order to disrupt search

Gossip in P2P architectures

- Constant, « background » communication between peers
- Structured or unstructured networks
- Disseminates information about peer network, peer data





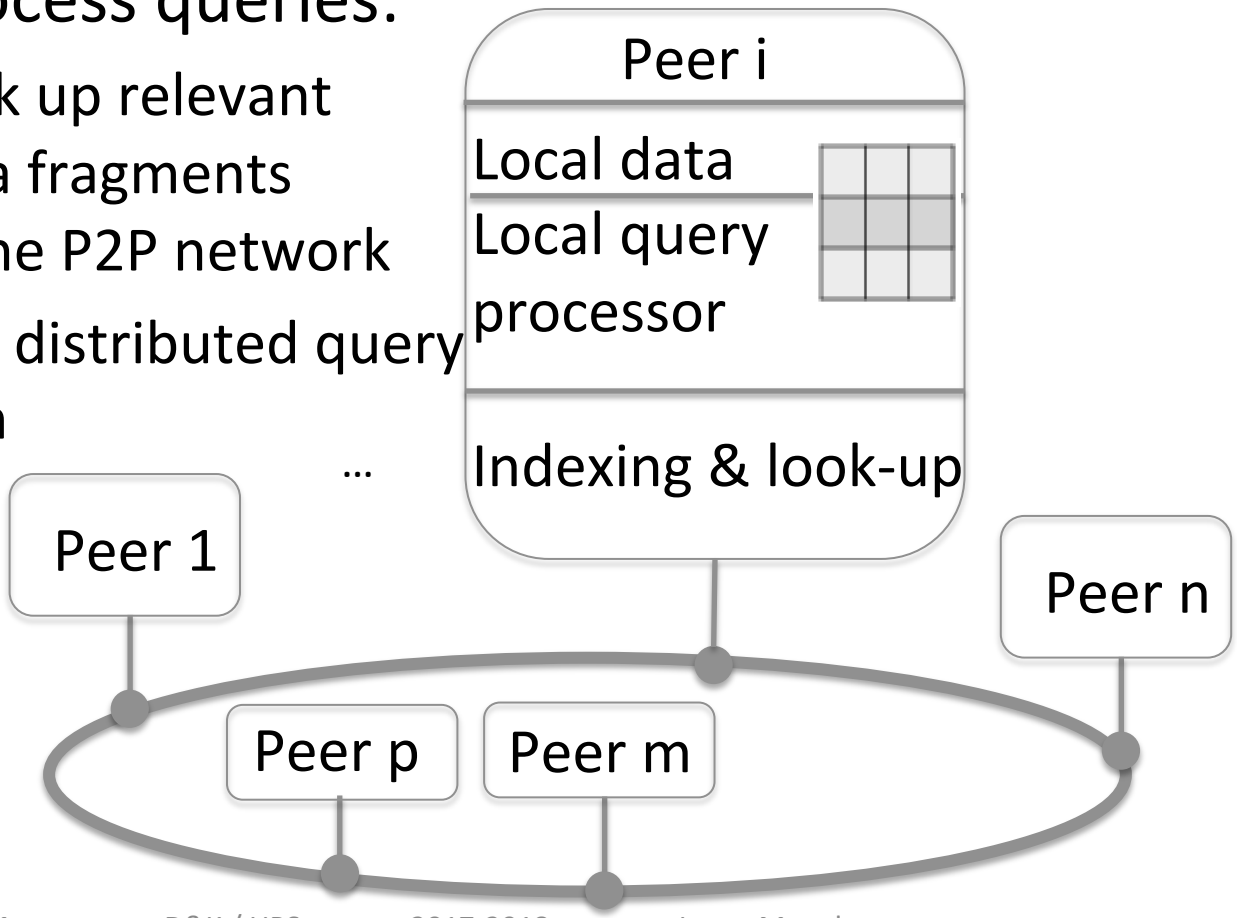
E.g. Cassandra (« Big Table » system):
« *During gossip exchanges, every node maintains a* ***sliding window of inter-arrival times of gossip messages from other nodes in the cluster****. Configuring the phi_convict_threshold property adjusts the sensitivity of the failure detector. Lower values increase the likelihood that an unresponsive node will be marked as down, while higher values decrease the likelihood that transient failures causing node failure.*
Use the default value for most situations, but ***increase it to 10 or 12 for Amazon EC2*** *(due to frequently encountered network congestion) to help prevent false failures.*
Values ***higher than 12 and lower than 5*** *are not recommended.* »

Peer-to-peer networks: wrap-up

- **Data model**:
  - Catalog and search at a simple key level
- **Query language**: keys
- **Heterogeneity**: not the main issue
- **Control**:
  - peers are autonomous in storing and publishing
  - query processing through symetric algorithm (except for superpeers)

Peer-to-peer data management

- Extract key-value pairs from the data & index them
- To process queries:
  - Look up relevant data fragments in the P2P network
  - Run distributed query plan



Example: storing relational data in P2P data management platform

- Each peer stores a horizontal slice of a table
- Catalog **at the granularity of the table**:
  - Keys: table names, e.g. Singer, Song
  - Value: peer1:postgres:sch1/Singer&u=u1&p=p1,
  - Query: select Singer.birthday
    from Singer, Song
    where Song.title= « Come Away » and
    Singer.sID=Song.singer
  - What can happen?
- Try other granularities