Kapitel 17: Verteilte Informationssysteme

17.1 Verteilte Systemarchitekturen

- **Software-Component-Oriented:**
  - **Client-Server Architecture**
  - **Multi-Tier Architecture**
  - **Federated Systems**
- **Data-Source-Oriented:**
  - **Data Warehouses (and Digital Libraries)**
  - **Wrapper-Mediator Architecture for Information Integration (Example: Internet Portals)**
- **Uncoordinated Decentralization:**
  - **Service-Oriented Architecture for Web Services or Grid**
  - **Peer-to-Peer Systems (P2P)**

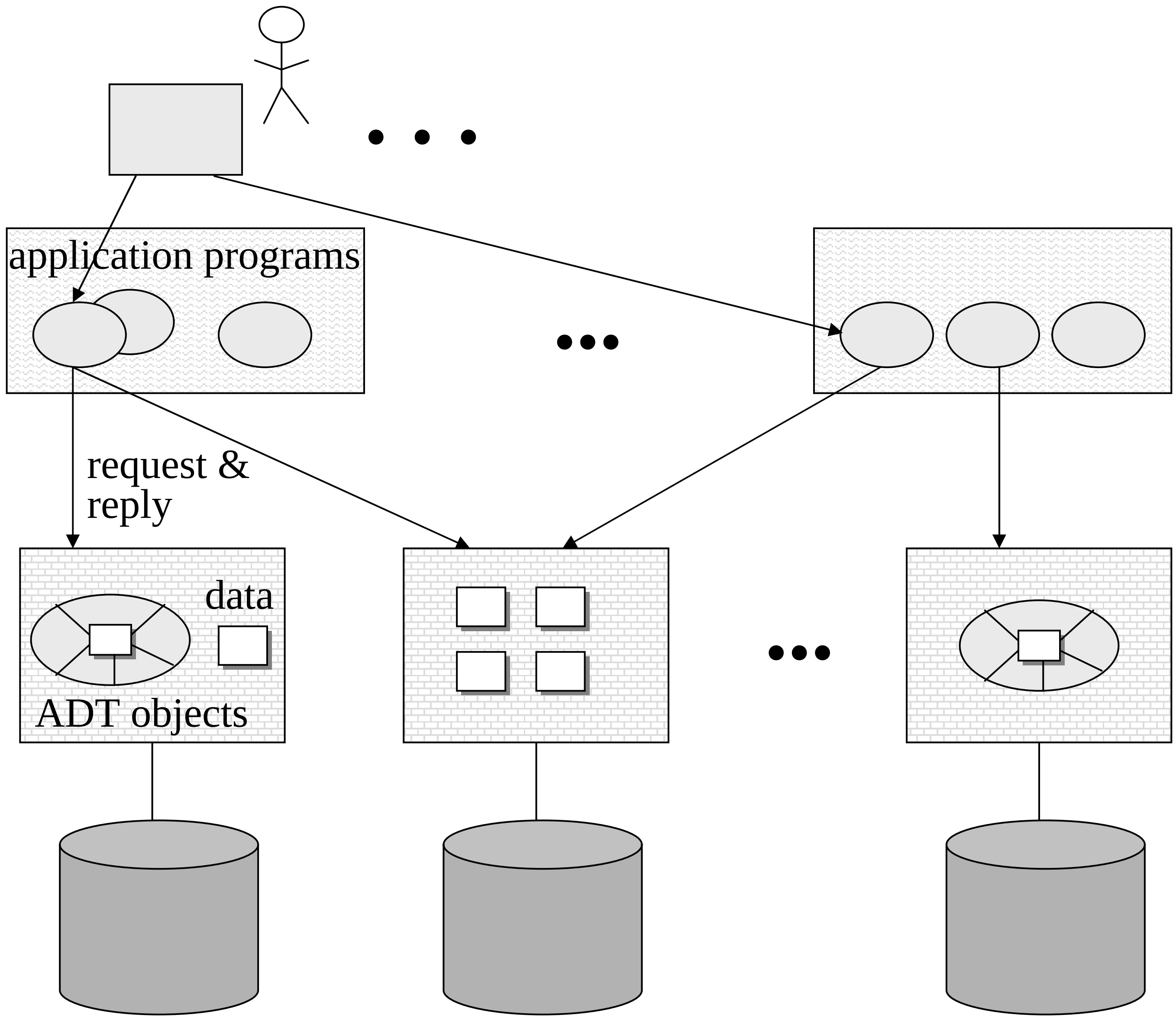
Client-Server and (Federated) Multi-Tier Systems

Users

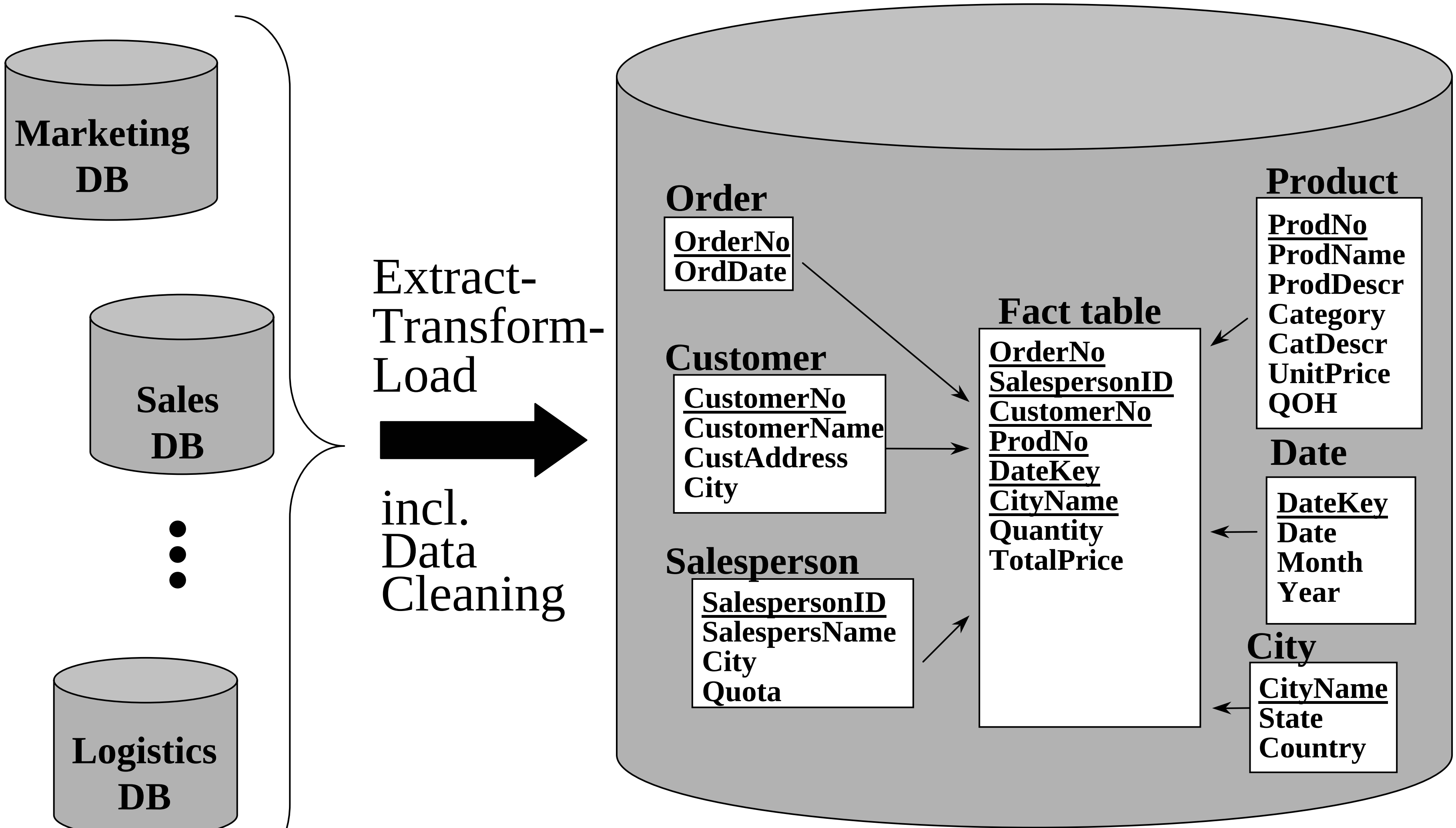
Clients

(Web) Application Servers

Data Servers (DB, IR, CMS, Mail, etc.)

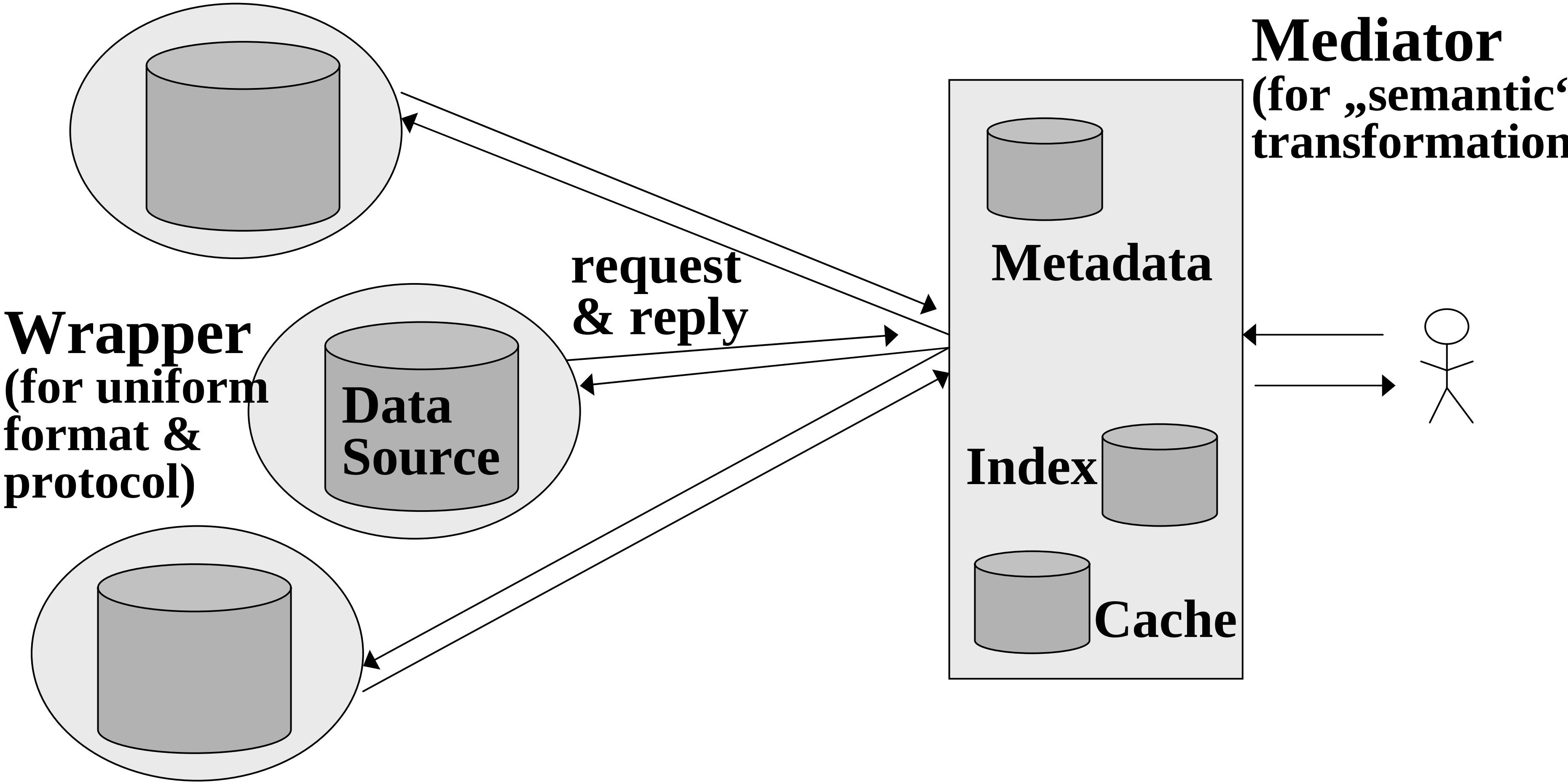


Data Warehouses (and Digital Libraries)



periodic or continuous, incremental ETL into
DW with star or snowflake schema (facts, dimensions)

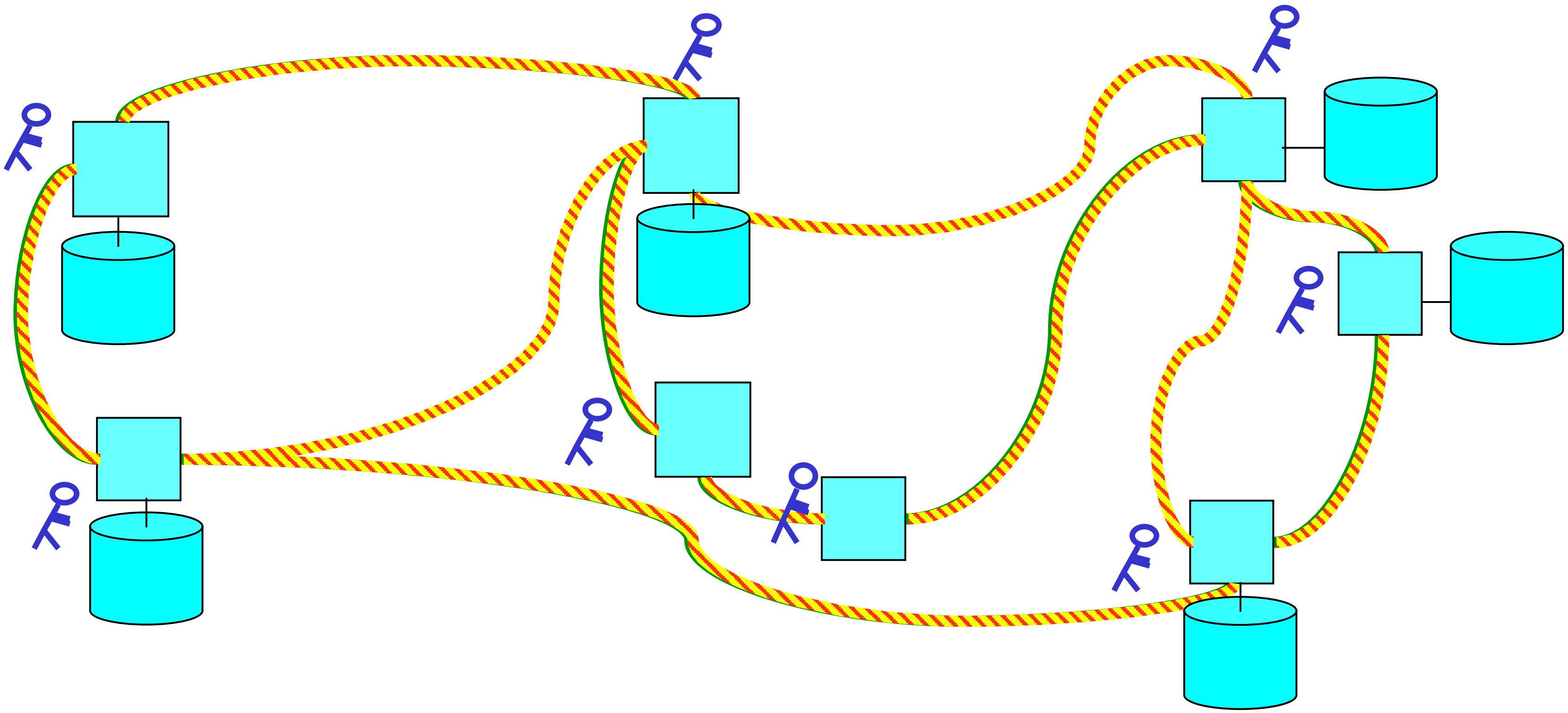
Wrapper-Mediator Architecture for Information Integration Systems



Wrappers may be hand-engineered or generated

Examples:
- **Yahoo for news, business, etc.**
- **SRS for bioinformatics & life sciences**
- **intranet portals for organizations**

Peer-to-Peer (P2P) Information Sharing



User post sharable files on autonomous computers (peers)
P2P system maintains (distributed) directory with file names
Users query file names and download files from other peers

Peer-to-Peer (P2P) Architectures

Decentralized, self-organizing, highly dynamic loose coupling of many autonomous computers

Applications:

- **Large-scale distributed computation (SETI, PrimeNumbers, etc.)**
- **File sharing (Napster, Gnutella, KaZaA, etc.)**
- **Publish-Subscribe Information Sharing (Marketplaces, etc.)**
- **Collaborative Work (Games, etc.)**
- **Collaborative Data Mining**
- **(Collaborative) Web Search**

Goals:

- **make systems ultra-scalable and completely self-organizing**
- **make complex systems manageable and less susceptible to attacks**
- **break information monopolies, exploit small-world phenomenon**

1st-Generation P2P

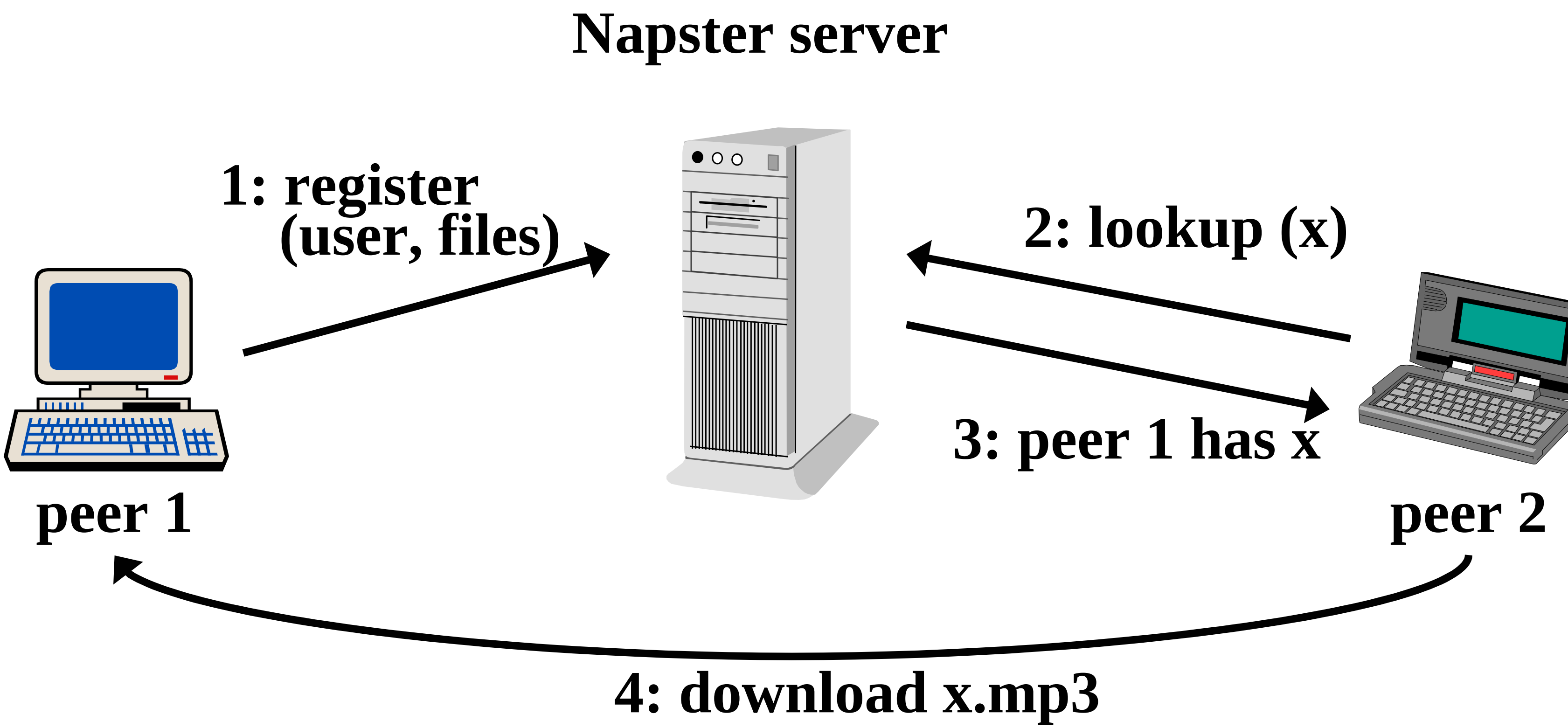
Napster (1998-2001) and Gnutella (1999-now):
driven by file-sharing for MP3, etc.
very simple, extremely popular

can be seen as a mega-scale but very simple publish-subscribe system:

- **owner of a file makes it available under name x**
- **others can search for x, find copy, download it**

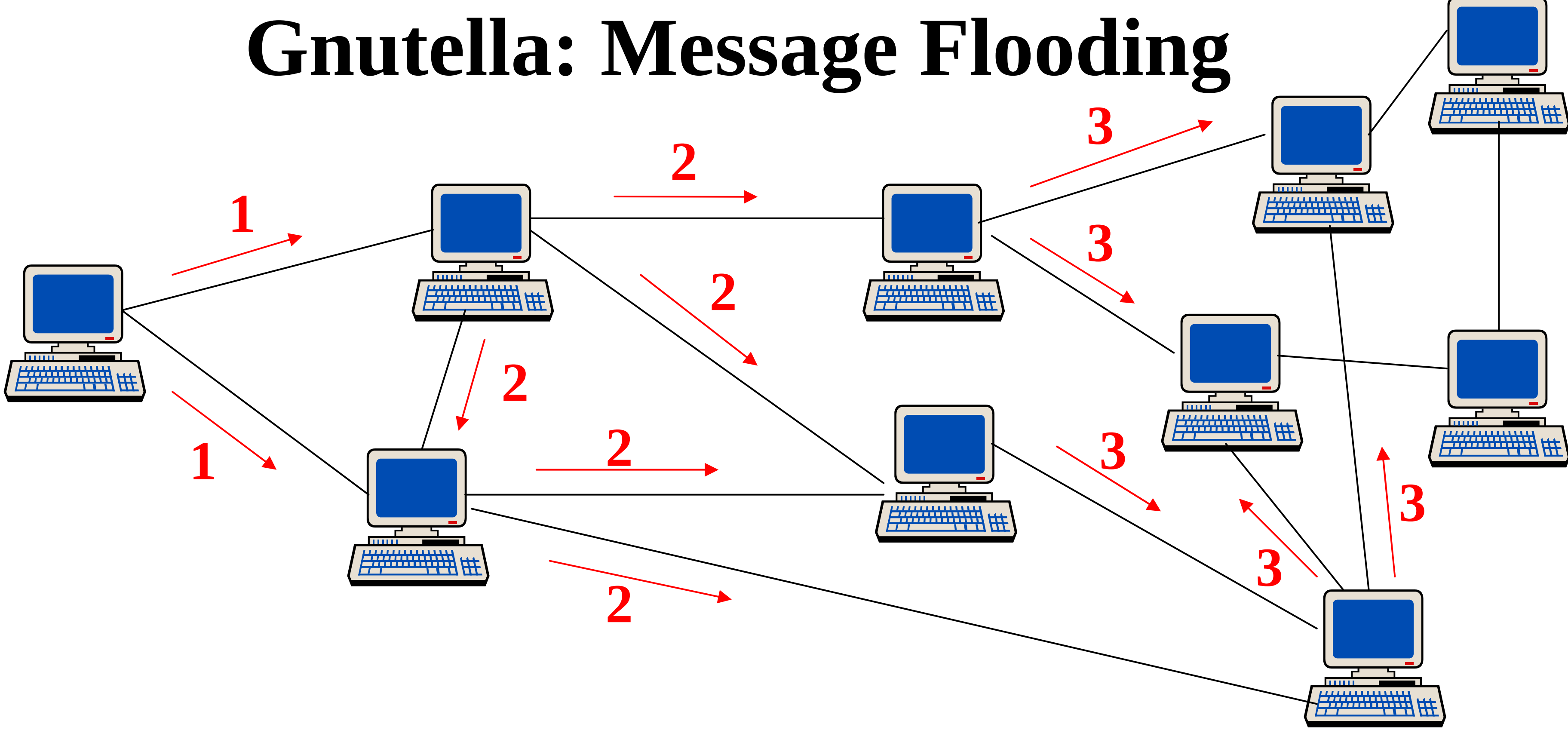
invitation to break the law (piracy, etc.) ?

Napster: Centralized Index



+ chat room, instant messaging, firewall handling, etc.

Gnutella: Message Flooding



all forward messages carry a TTL tag (time-to-live)

1) contact neighborhood and establish virtual topology (on-demand + periodically): *Ping, Pong*
2) search file: *Query, QueryHit*
3) download file: *Get* or *Push* (behind firewall)

2nd-Generation P2P

Freenet
emphasizes anonymity

eDonkey, KaZaA (based on FastTrack), Morpheus, MojoNation, AudioGalaxy, etc. etc.
commercial, typically no longer open source; often based on super-peers

JXTA
(Sun-sponsored) open API

Research prototypes (with much more refined architecture and advanced algorithms):
Chord (MIT), CAN (Berkeley), OceanStore/Tapestry (Berkeley), Farsite (MSR), Spinglass/Pepper (Cornell), Pastry/PAST (Rice, MSR), Viceroy (Hebrew U), P-Grid (EPFL), P2P-Net (Magdeburg), Pier (Berkeley), Peers (Stanford), Kademlia (NYU), Bestpeer (Singapore), YouServ (IBM Almaden), Hyperion (Toronto), Piazza (UW Seattle), PlanetP (Rutgers), SkipNet (MSR), Galanx (U Wisconsin), Minerva (MPII), etc. etc.

The Future of P2P: Challenging Requirements

Unlimited scalability with millions of nodes: O(log n) hops to target, O(log n) state per node

Failure resilience, high availability, self-stabilization (w.r.t. dynamics: many failures & high churn)

Data placement, routing, load management, etc. in overlay networks

Robustness to DoS attacks & other traffic anomalies

Trustworthy computing and data sharing

Incentive mechanisms to reconcile selfish behavior of individual nodes with strategic global goals

P2P-Related Technologies

Web Services (SOAP, WSDL, etc.)
for e-business interoperability (supply chains, etc.)

Grid Computing
for scientific data interoperability

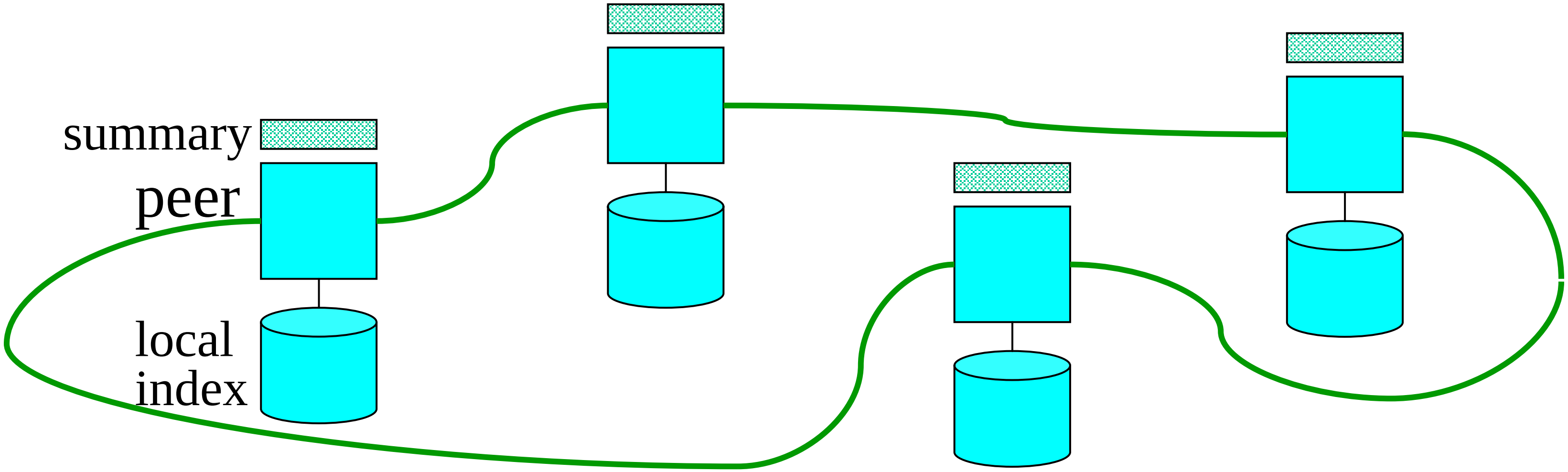
Autonomic / Organic / Introspective Computing
for self-organizing, zero-admin operation

Multi-Agent Technology
for interaction of autonomous, mobile agents

Sensor Networks
for data streams from measurement devices etc.

Content-Delivery Networks (e.g., Akamai)
for large content of popular Web sites

17.2 Anfrageausführung in verteilten IR-Systemen



- every peer is autonomous and has its own local search engine
- every peer posts (statistical) summary info about its contents (index lists, bookmarks, cached docs, QoS properties, ...)
- query routing is driven by similarity to summaries
- summaries are organized into a (distributed) directory
  - mapped onto DHT, random-graph overlay network, etc.
  - lazily replicated at additional peers (via „gossiping")

querying peer needs to

1. determine interesting peers (query routing)
2. plan, run, monitor, and adapt distributed top-k algorithm
3. reconcile results from different peers

Why Peer-to-Peer Web Search?

Goal: Self-organizing P2P Web Search Engine
with Google-or-better functionality

- Scalable & Self-Organizing Data Structures and Algorithms
  (DHTs, Semantic Overlay Networks, Epidemic Spreading, Distr. Link Analysis, etc.)

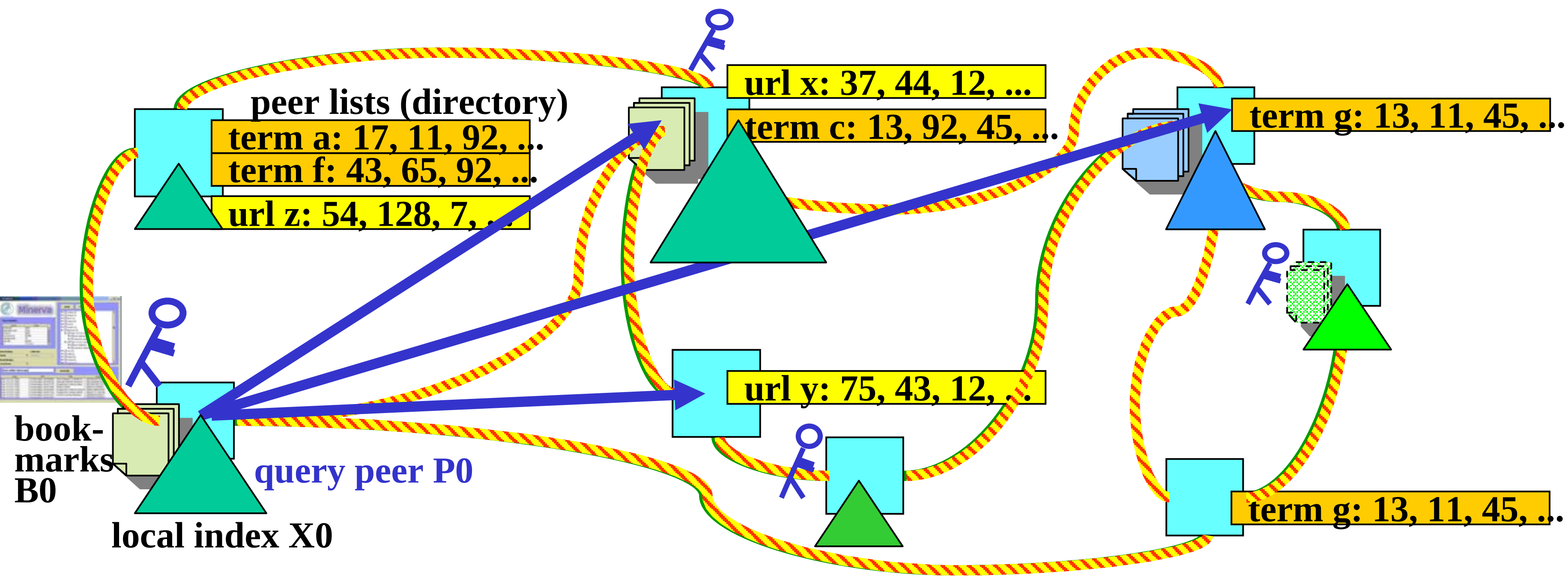
- Better Search Result Quality (Precision, Recall, etc.)
  - Powerful Search Methods for Each Peer
    (Concept-based Search, Query Expansion, Personalization, etc.)
  - Leverage Intellectual Input at Each Peer
    (Bookmarks, Feedback, Query Logs, Click Streams, Evolving Web, etc.)
  - Collaboration among Peers
    (Query Routing, Incentives, Fairness, Anonymity, etc.)

- Small-World Phenomenon
  Breaking Information Monopolies

Differences between Meta and P2P Search Engines

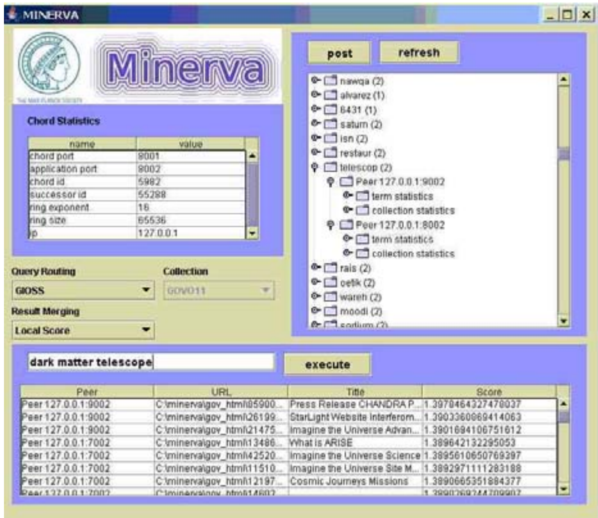
Meta Search Engine	P2P Search Engine
small # sites (e.g., digital libraries)	huge # sites
rich statistics about site contents	poor/limited/stale summaries
static federation of servers	highly dynamic system
each query fully executed at each site	single query may need content from multiple peers
interconnection topology largely irrelevant	highly dependent on overlay network structure

P2P Query Routing



Query routing aims to optimize benefit/cost
driven by distributed statistics on
peers' content similarity, content overlap,
freshness, authority, trust, performability etc.

Dynamically precompute „good peers"
to maintain a **Semantic Overlay Network**
using random but biased graphs



Framework and Parameters for Query Routing

M terms t_i, N documents d_j, P peers with N_k docs at peer p_k

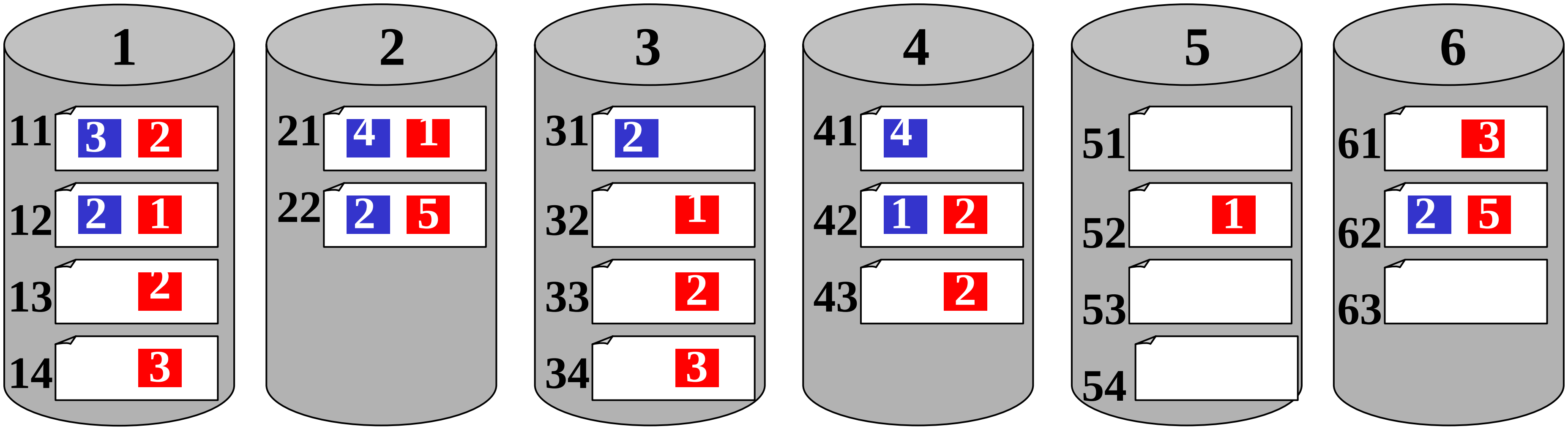
<u>local measures at peer k:</u>

$tf_i^{(k)}(d)$ – freq. of term i in doc d
$df_i^{(k)}$ - # docs with term i
$idf_i^{(k)}$ - inverse doc freq. of term i
$ttf_i^{(k)}$ – total freq. of term i

$mtf_i^{(k)}$ – max. term freq.
$mdf^{(k)}$ – max. doc freq.
$t^{(k)}$ – # distinct terms

<u>global measures:</u>

$gidf_i$ - inverse doc frequency of term i in P2P system
$gipf_i$ - inverse peer frequency of term i in P2P system



select best peers based on expected benefit/cost ratio

Query Routing based on IPF (PlanetP)

Every peer conceptually maintains the global inverse peer frequency (gipf) for each term i:

$$gipf_i = log\left(1 + \frac{\#\ peers}{\#\ peers\ \ with\ term\ i} \right)$$

For multi-keyword query q the quality of peer j is:

$$R_j(q) := \sum_{i \in q} gipf_i \cdot \begin{cases} 1\ if\ \ peer\ \ j\ contains\ term\ i \\ 0\ otherwise \end{cases}$$

To retrieve top k results for query q:

1. **rank peers in descending order of Rj(q)**
2. **contact peers in groups of m in rank order**
3. **merge results**
4. **iterate steps 2 and 3 until no peer contributes to top-k result**

Example: Query Routing based on IPF

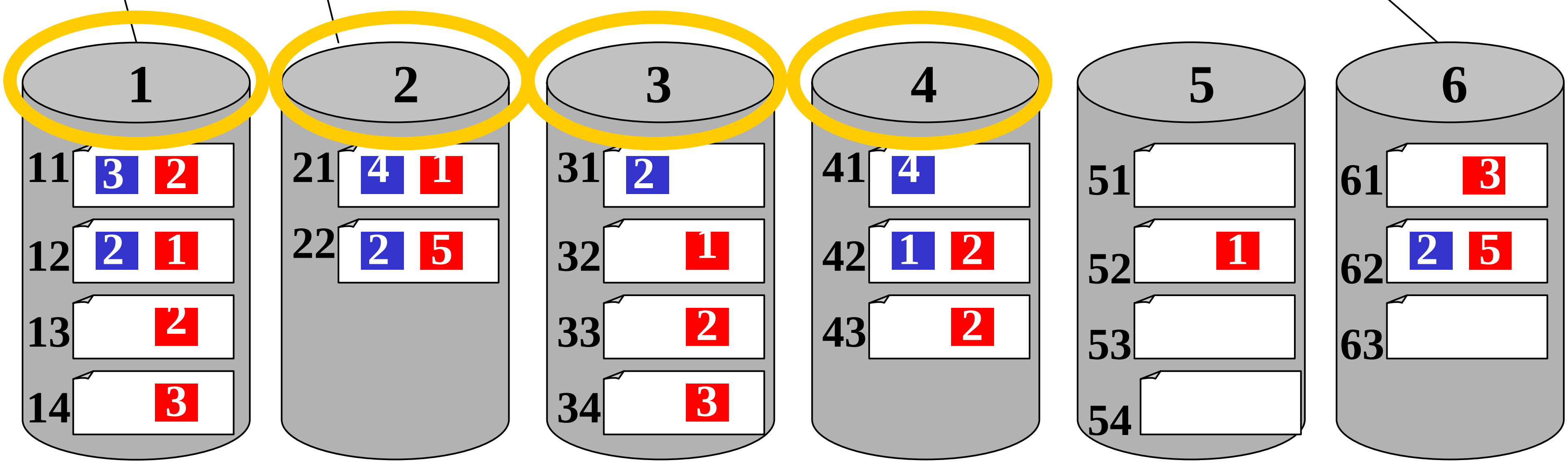
gipf = log (1 + 6/5)   gipf = log (1+1)

R_1( ) = log 11/5 * 1 + log 2 * 1

R_2( ) = log 11/5 * 1 + log 2 * 1

•••

R_6( ) = log 2 * 1



PlanetP Implementation

Each peer posts its summary in the form of a
***Bloom-filter signature*:**
- **bit vector S[1..s] of fixed length s, initially all bits zero**
- **if peer j has term i it sets bit h(i) to one using a hash function h**
- **other peers can test if peer j holds term set $\{q_1, ..., q_k\}$ by looking up $S[h(q_1)], ..., S[h(q_k)]$ or by computing a bit vector Q[1..s] for $\{q_1, ..., q_k\}$ and ANDing S with Q, both with the risk of *„false positives“***

Summaries are sent to other peers by asynchronous *gossiping* in a combined push/pull mode:
- ***push*: periodically send updates of global registry (small Δs) as „rumors“ to randomly chosen neighbors; stop doing so when n consecutive peers already know the update**
- **(anti-entropy) *pull*: periodically ask randomly chosen neighbor to send an updated summary of the global registry; alternatively ask push-recipient for recent rumors**

Query Routing based on Simple Heuristics

Consider df, ttf, mtf, gipf as quality measures of a peer

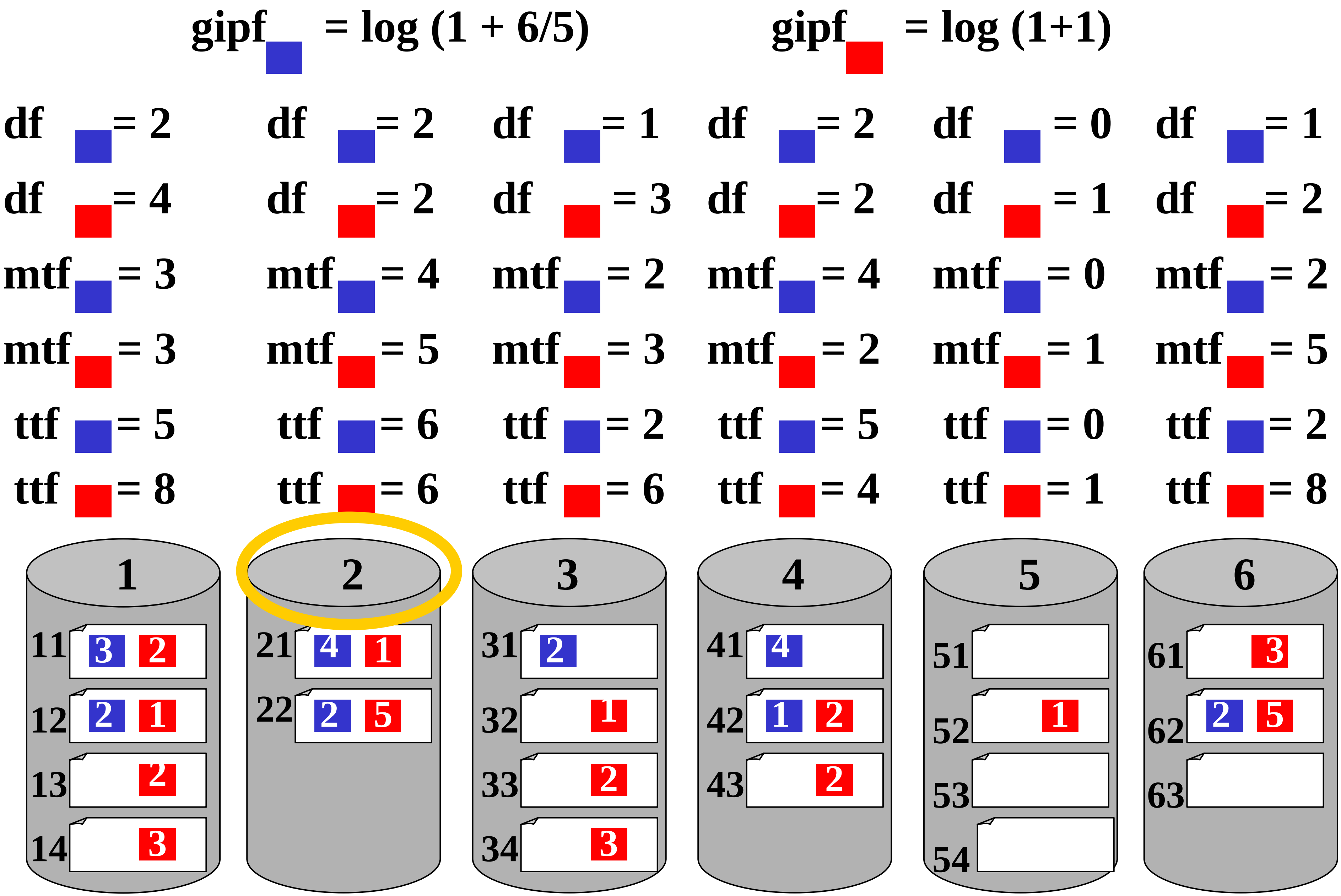
Choose peers in descending order of

$$\sum_{i \in q} \alpha_1 \log df_i^{(k)} + \alpha_2 \log mtf_i^{(k)} + \alpha_3 \log ttf_i^{(k)} \quad \textbf{or}$$

$$\sum_{i \in q} gipf_i \cdot \left(\alpha_1 \log df_i^{(k)} + \alpha_2 \log mtf_i^{(k)} + \alpha_3 \log ttf_i^{(k)} \right)$$

with tunable weights α_1, α_2, α_3 such that $\alpha_1 + \alpha_2 + \alpha_3 = 1$

Example: Query Routing Heuristics



Query Routing based on Statistical Similarity

For query q select peers p with highest value of sim(q, p), e.g., cosine(q, p) where p is represented by its centroid

Use statistical language model for similarity:

$$KL(q \,||\, p) = \sum_{t \in q} P[t\,|\,q] \; \log \frac{P[t\,|\,q]}{\lambda P[t\,|\,C_p] + (1-\lambda) P[t\,|\,G]}$$

where P[t|q], P[t|Cp], P[t|G] are the (estimated) probabilities that term t is generated by the language models for the query q, the corpus C_k of peer k, and the general vocabulary, and λ is a smoothing parameter between 0 and 1

Implementation may estimate $P[t|C_k] \approx ttf_t^{(k)} / \Sigma_i\, ttf_i^{(k)}$

The Kullback-Leibler divergence (aka. relative entropy) is a measure for the distance between two probability distributions:

$$KL(f \,\|\, g) := \sum_x f(x) \log \frac{f(x)}{g(x)}$$

CORI Query Routing

Apply probabilistic IR method with heuristic elements (Okapi BM25 term weighting) to peer selection by treating a peer‘s complete index contents as a „document“

$$P[\, q \mid p \,] \sim$$

$$\sum_{i \in q} \frac{ttf_i^{(k)}}{ttf_i^{(k)} + 0.5 + 1.5\, t^{(k)} / avg_\nu(t^{(\nu)})}$$

$$\cdot \frac{\log(gipf_t \cdot (0.5 + \#peers))}{\log(1 + \#peers)}$$

GlOSS Query Routing based on Goodness

Goodness (q, s, l) = $\sum$ {sim(q, d) | d $\in$ result(q, s) $\wedge$ lsim(q,d)>l}
for query q, source s, and score threshold l

GlOSS (Glossary Of Servers Server) aims to rank sources by goodness

Approximate goodness by using for source s:
- **$df_i(s)$: number of docs in s that contain term i**
- **$w_i(s)$: $\sum$ {$tf_i(d)*idf_i$ | d $\in$ s} (total weight of term i in s)**

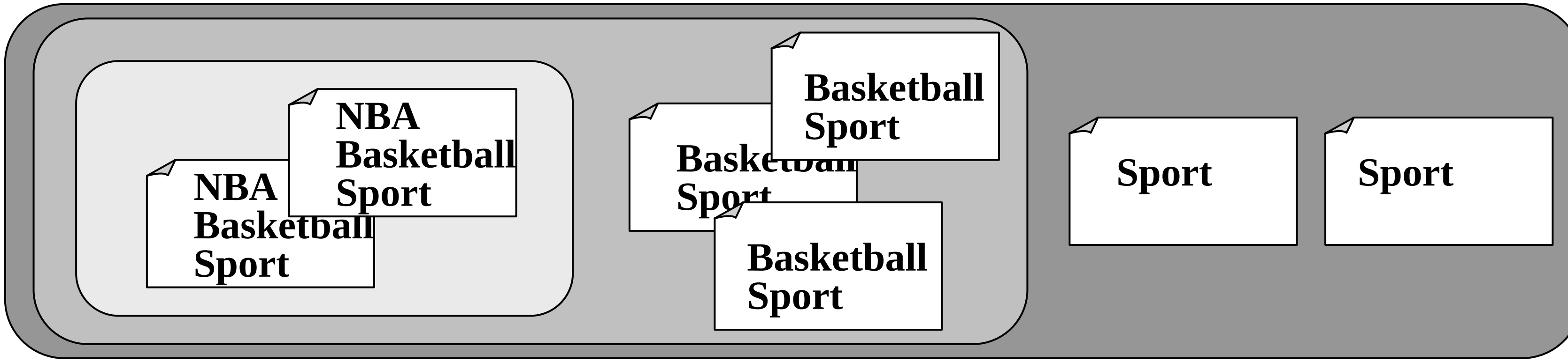
Uniformity assumption:
$w_i(s)$ is distributed uniformly over all docs in s that contain i

GlOSS Goodness with High-correlation Assumption

High-correlation assumption:

$df_i(s) \le df_j(s) \Rightarrow$ every doc in s that contains i also contains j

Example:



For fixed source s and query q = $t_1 \ldots t_n$ with $df_i \le df_{i+1}$ for i=1..n-1 consider subqueries $q_p = t_p \ldots t_n$ (p=1..n).
Every doc d in s that contains only $t_p \ldots t_n$ has query similarity

$$sim_p(q,d) = \sum_{i=p..n} t_i \frac{w_i(s)}{df_i(s)}$$

Find p such that $sim_p(q,d) > l$ and $sim_{p+1}(q,d) \le l$

EstGoodness(q,s,l) = $\sum_{i=1..p} (df_i(s) - df_{i-1}(s)) * sim_i$

GlOSS Goodness with Disjointness Assumption

Disjointness assumption:

$\{d \in s | d \text{ contains term } i\} \cap \{d \in s | d \text{ contains term } j\} = \emptyset$ for all $i,j \in q$

Uniformity assumption:

$w_i(s)$ is distributed uniformly over all docs in s that contain i

$$sim_i(q,d) = t_i \frac{w_i(s)}{df_i(s)}$$

EstGoodness(q,s,l) $= \sum_{i=1..n \wedge sim_i > l} df_i(s) \cdot t_i \frac{w_i(s)}{df_i(s)}$

$$= \sum_{i=1..n \wedge sim_i > l} t_i \; w_i(s)$$

Usefulness Estimation Based on MaxSim

Def.: A set S of sources is *optimally ranked* for query q in the order s1, s2, ..., sm if for every n>0 there exists k, $0<k\leq m$, such that s1, ..., sk contain the n best matches to q and each of s1, ..., sk contains at least one of these n matches

Thm.: Let MaxSim(q,s) = max{sim(q,d)|$q \in s$}.
s1, ..., sm are optimally ranked for query q if and only if MaxSim(q,s1) > MaxSim(q,s2) > ... > MaxSim(q,sm).

Practical approach („Fast-Similarity method“):
Capture, for each s, $df_i(s)$, $avgw_i(s)$, $maxw_i(s)$ as source summary.
Estimate for query q = t1 ... tk

$$MaxSim(q,s) := \max_{i=1..k} \{t_i * maxw_i(s) + \sum_{\nu \neq i} t_\nu * avgw_\nu(s)\}$$

estimation time linear in query size,
space for statistical summaries linear in #sources * #terms

Overlap Aware Query Routing

First execute q on initiator peer's local index X0,
then select peers Pi with highest benefit/cost ratio where

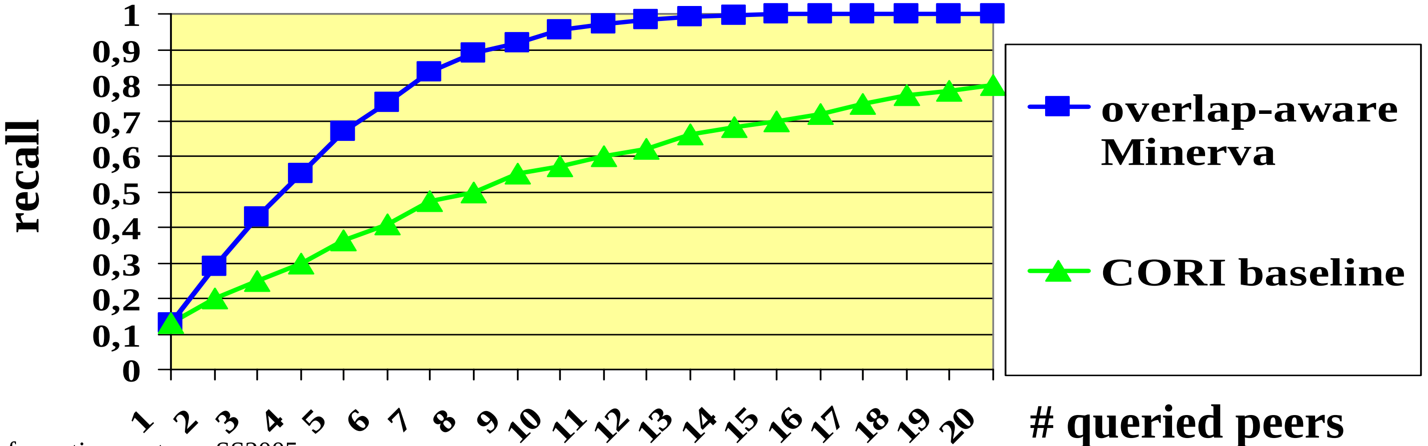
- **benefit(Pi) ~ sim (X0, Xi) and ~ 1/overlap(X0, Xi)**
- **cost(Pi) ~ estimated response time or communication costs**

precompute sim: $KL(X0, Xi) := \sum_{terms\ x} freq(x, X0)\ \log \frac{freq(x, X0)}{freq(x, Xi)}$

approximate overlap by Bloom filters, hash sketches, etc.

Experiments:

based on 100 .Gov partitions (1.25 Mio. docs), assigned to 50 peers,
with each peer holding 10 partitions and 80% overlap for P_i, P_{i+1}
with 50 TREC-2003 Web queries, e.g.: „juvenile delinquency"



Distributed Query Execution Issues

Algorithm:

• **Determine the number of results to be retrieved from each source a priori based on the source's content quality**

vs.

• **Run distributed version of top-k query processing algorithm**

Dynamic adaptation:

• **Plan query execution only once before initiating it vs.**

• **Dynamic plan adjustment based on sources' result quality and responsiveness (incl. failures)**

Parallelism:

• **Start querying all selected sources in parallel vs.**

• **Consider (initial) results from one source when querying the next sources**

Result Reconciliation

**Case 1: all peers use the same scoring function,
e.g. cosine similarities based on tf*idf weights**

**Case 2: peers may use different scoring functions
that are publicly known**

**Case 3: peers may use different & unknown scoring functions
but provide scored results**

Case 4: peers provide only result rankings, no scores

Baseline case:
**when gidf values are known at the query initiator,
we can recompute tf values from the different peers‘ result docs
and compute global scores based on gidf and tf values**

Techniques for Result Reconciliation (1)

for case 1:

local sim is

$$lsim(\vec{q},\vec{d}) = \sum_i \frac{q_i \cdot tf_i(\vec{d}) \cdot lidf_i}{\sqrt{\sum_i {q_i}^2} \cdot \sqrt{\sum_i tf_i(\vec{d})^2 \cdot {lidf_i}^2}}$$

global sim is

$$sim(\vec{q},\vec{d}) = \sum_i \frac{q_i \cdot tf_i(\vec{d}) \cdot gidf_i}{\sqrt{\sum_i {q_i}^2} \cdot \sqrt{\sum_i tf_i(\vec{d})^2 \cdot {gidf_i}^2}}$$

either recompute *tf* of result docs, infer *lidf* values, and compute *sim*

or submit additional single-term queries (one for each query term) such that each result d to the original query q is retrieved:

$$lsim(q_i,\vec{d}) = \frac{q_i \cdot tf_i(\vec{d}) \cdot lidf_i}{q_i \cdot \sqrt{\sum_j tf_j(\vec{d})^2 \cdot {lidf_j}^2}} = \frac{tf_i(\vec{d}) \cdot lidf_i}{\sqrt{\sum_j tf_j(\vec{d})^2 \cdot {lidf_j}^2}}$$

$$\Rightarrow \frac{lidf_i}{\sqrt{\sum_j tf_j(\vec{d})^2 \cdot {lidf_j}^2}} = \frac{lsim(q_i,\vec{d})}{tf_i(\vec{d})}$$

solve equation system for *tf* and *lidf* values (if possible) and compute *sim*

Techniques for Result Reconciliation (2)

for case 4:

set global score of doc j retrieved from source i to

$$g(d_j) := 1 - (r_{local}(d_j) - 1) \cdot \frac{r_{\min}}{m \cdot r_i}$$ **where**

- **r_{local}(dj) is the local rank of d_j,**
- **r_i is the score of source i among the queried sources,**
- **r_{min} is the lowest such score, and**
- **m is the number of desired global results**

Intuition:

- **initially local ranks are linearly mapped to scores**
- **the factor r_{min} / (m r_i) is the score difference for consecutive ranks from source i**

Wrap-Up: P2P Query Routing & Execution

Research on distributed IR has provided many approaches – principled as well as heuristic ones – that can be carried over to a P2P setting

However, the scale, dynamics, and usage patterns of a P2P search engine entail additional issues, many of which are widely open:

- **peer statistics collection and dissemination (frequency, quality, overlap, resource util., response times, etc.)**
- **precomputation of good peers → „semantic overlay network“**
- **consideration of strong correlations**
- **combination with PageRank-style authority etc.**
- **consideration of P0 query execution and feedback**
- **coping with tradeoffs in network bandwidth & latency, per-peer resource consumption, search result quality**

Literature

- Communications of the ACM, Vol 46, No. 2, Special Section on
- Peer-to-Peer Computing, Feb. 2003.
- Weiyi Meng, Clement Yu, King-Lup Liu: Building Efficient and Effective Metasearch Engines, ACM Computing Surveys 34(1), 2002
- F.M. Cuenca-Acuna, C. Peery, R.P. Martin, T.D. Nguyen: PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities, IEEE Symp. on HPDC, 2003
- Jie Lu, Jamie Callan: Content-Based Retrieval in Hybrid Peer-to-Peer Networks, CIKM Conference, 2003.
- Henrik Nottelmann, Norbert Fuhr: Evaluating Different Methods of Estimating Retrieval Quality for Resource Selection, SIGIR 2003
- M. Bender, S. Michel, P. Triantafillou, G. Weikum, C. Zimmer: Improving Collection Selection with Overlap Awareness in P2P Search Engines, SIGIR 2005
- M. Bender, S. Michel, G. Weikum, C. Zimmer: Das MINERVA-Projekt: Datenbankselektion für Peer-to-Peer-Websuche, Informatik Forschung und Entwicklung, 2005

17.3 Skalierbare verteilte Indexierung und Suche

Goals:

Decentralize

data store (MP3 files, Web documents, etc.) or
index (term-doc-score entries) or
directory (statistical info about peers)

across N peers with large N
and provide file-name / keyword lookup

with good properties:

- **scalability: throughput is proportional to N, response time is independent of N (or grows very slowly with N)**
- **efficiency: overhead should be O(1) or ≤ O(log N)**
- **load balance: factor between least loaded and most loaded peer should be O(log N) or even O(log log N)**
- **robustness to failures: peers being temporarily down**
- **robustness to churn: peers joining and leaving at high rate**
- **self-organization regarding**
  data growth, load growth, dynamics of system and load patterns

Structured P2P Interface

Basic operations:

store (key, data)	**- inserts a new data item**
lookup (key) returns data	**- finds a data item by its key**
delete (key)	**- removes a data item**
join (node)	**- new node joins the P2P network**
leave (node)	**- node leaves the P2P network**

Applications:

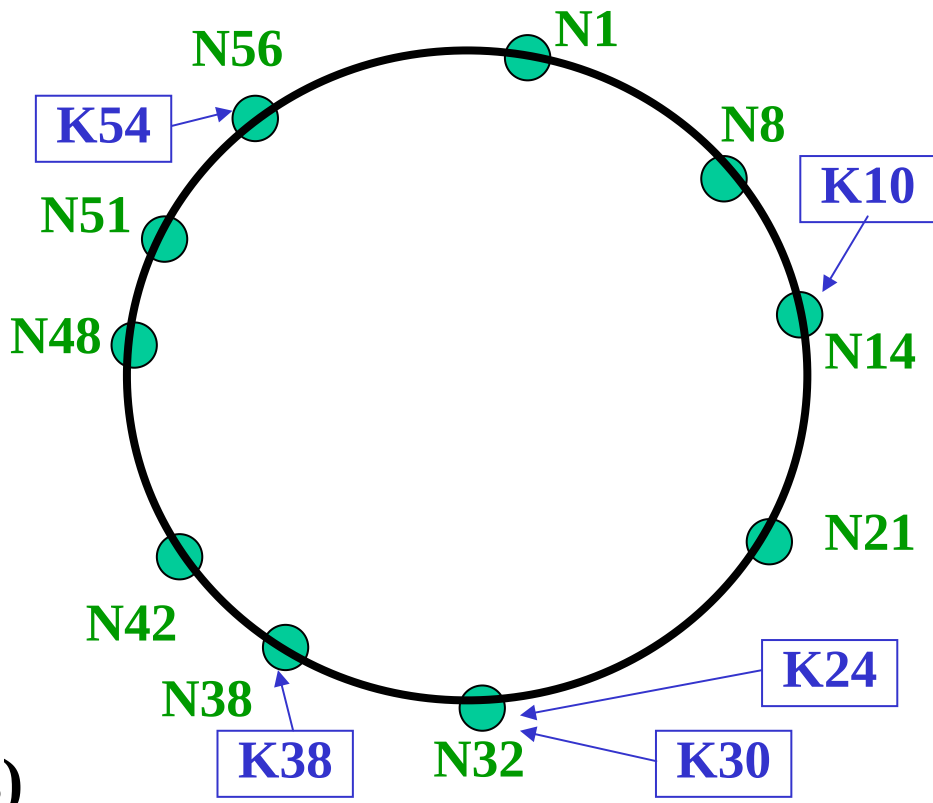
- **file sharing**
- **distributed storage (personal photo albums etc.)**
- **distributed caching of Web content**
- **DNS (domain name service)**
- **news and discussion forums (Usenet etc.)**
- **search engine directory and statistical info**
- **network monitoring (incl. anomaly detection)**

Structured P2P: Example Chord

<u>Distributed Hash Table (DHT):</u>

map strings (file names, keywords) and numbers (IP addresses) onto very large „cyclic" key space 0..2^m-1, the so-called *Chord Ring*

Key k (e.g., *hash*(file name)) is assigned to the node with key n (e.g., *hash*(IP address)) such that k ≤ n and there is no node n' with k ≤ n' and n'<n



<u>Properties:</u>

Unlimited scalability (> 10^6 nodes)

O(log n) hops to target, O(log n) state per node

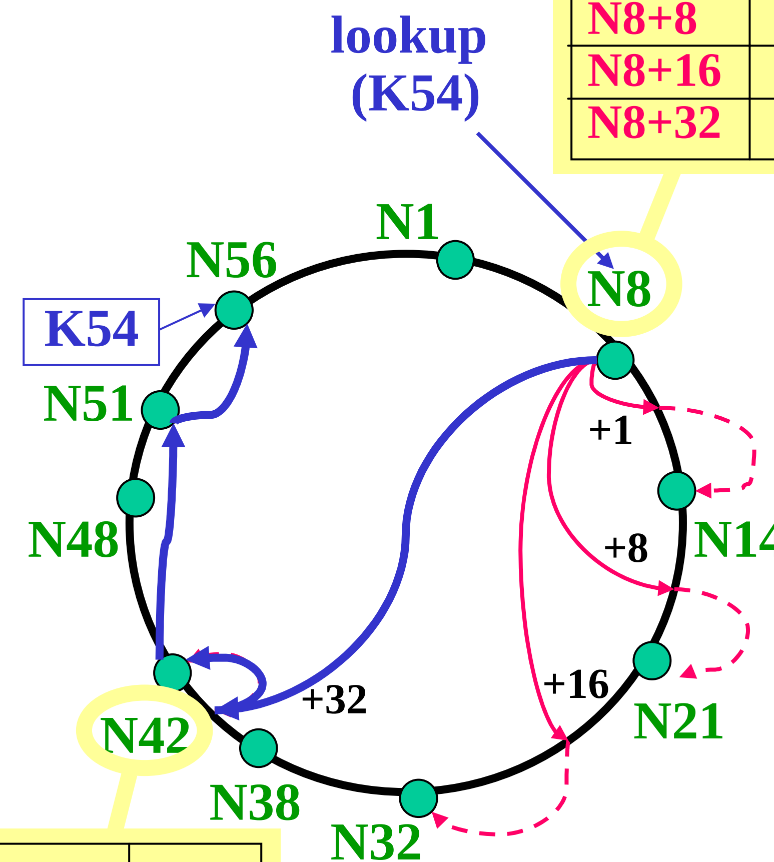
Self-stabilization (many failures, high dynamics)

Request Routing in Chord

Every node knows its pred/succ and has a finger table with log(n) pointers: finger[i] = successor (node id + 2^{i-1}) mod 2^m for i=1..m

Finger table:

N8+1	N14
N8+2	N14
N8+4	N14
N8+8	N21
N8+16	N32
N8+32	N42



N42+1	N48
N42+2	N48
N42+4	N48
N42+8	N51
N42+16	N1
N42+32	N14

For finding key k perform repeatedly: determine current node's largest finger[i] (modulo 2^m) with finger[i] ≤ k

pred/succ ring and finger tables require dynamic maintenance → stabilization protocol

Chord Operations (1)

```
lookup (key, n): //invoked by node n
  id := hash(key);
  if n.predecessor()<id ≤ n then return n.localfind(key).data
  else // determine closest preceding node
     for i:=m downto 1 do
        if n.finger[i] ≤ id then exit loop;
     lookup (key, n.finger[i]);
```

recursive lookup (see above) vs.
iterative lookup: finger[i] returned to original caller in each step

```
store (key, n):
  lookup + local store
```

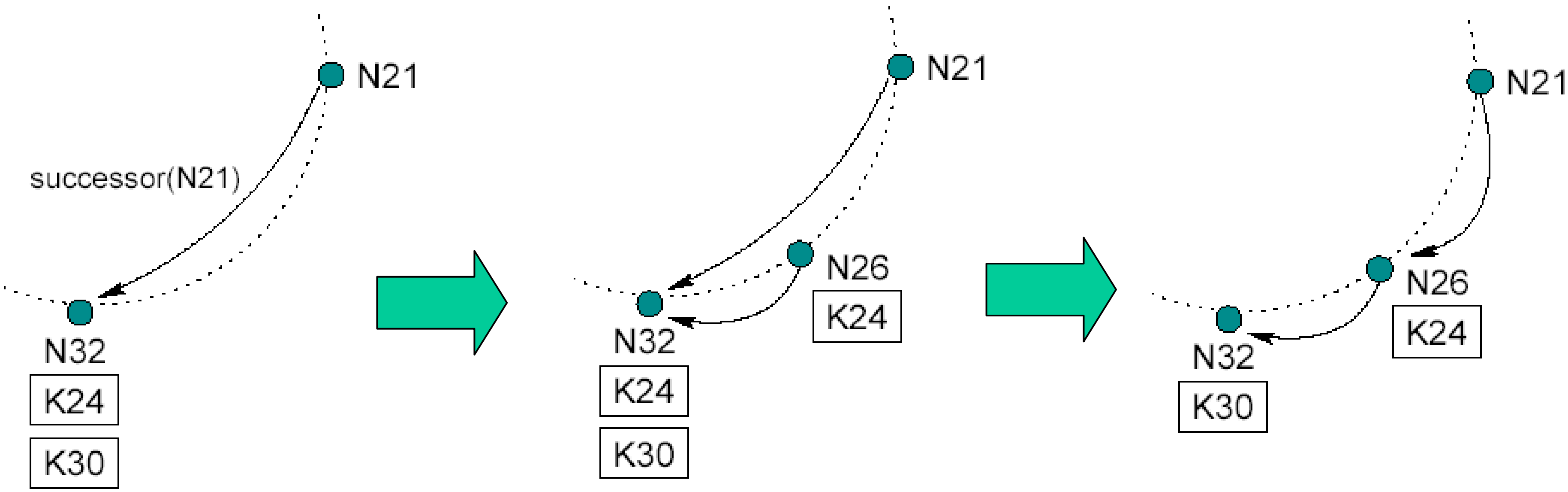
```
delete (key):
  lookup + local delete
```

Chord Operations (2)

```
join (n):
   id := hash(n);
   n.successor :=lookup (id, anyknownnode);  n.predecessor := nil;
   p := n.successor.predecessor;  p.successor := n;
   re-hash all keys (and data items) stored in n.successor;
   if hash value <= id then
       move key (and data item) to n;
   n.successor.predecessor := n; n.predecessor := p;
   for i:=0 to m do // build finger table
       n.finger[i] := lookup (n+2^i, n);
   stabilize (n);
```

```
leave (n):
   move all keys (and data items) owned by n to n.successor;
   n.predecessor.successor := n.successor;
   n.successor.predecessor := n.predecessor;
```

Example of Node Joining



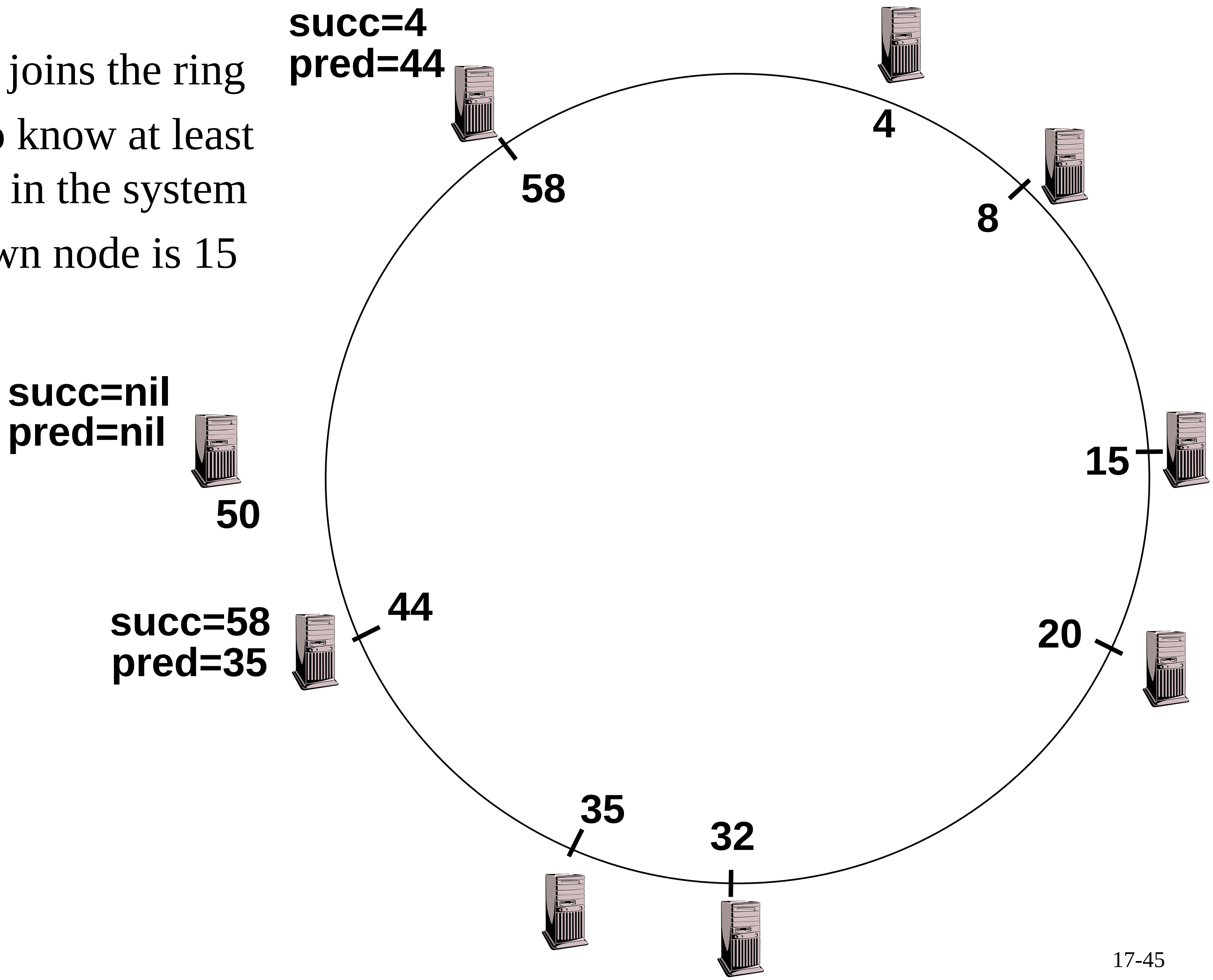
Chord Stabilization

every node must periodically check its successor & predecessor pointers and its finger table

```
stabilize (n):
  // find, verify & reconfirm/adjust successor
     s := lookup(n+1); x:= s.predecessor;
     if x.id > n.id then n.successor := x;  // test & adjust successor
     notify x:  // x learns about n
     if x knows no node between n and x then x.predecessor := n;
 // find, verify & adjust predecessor: analogously
 // test predecessor failure
     if probe message to n.predecessor times out
     then n.predecessor := nil;
 // refresh or fix finger table
     for i:=0 to m do
         n.finger[i] := lookup (n + 2^i, n);
```

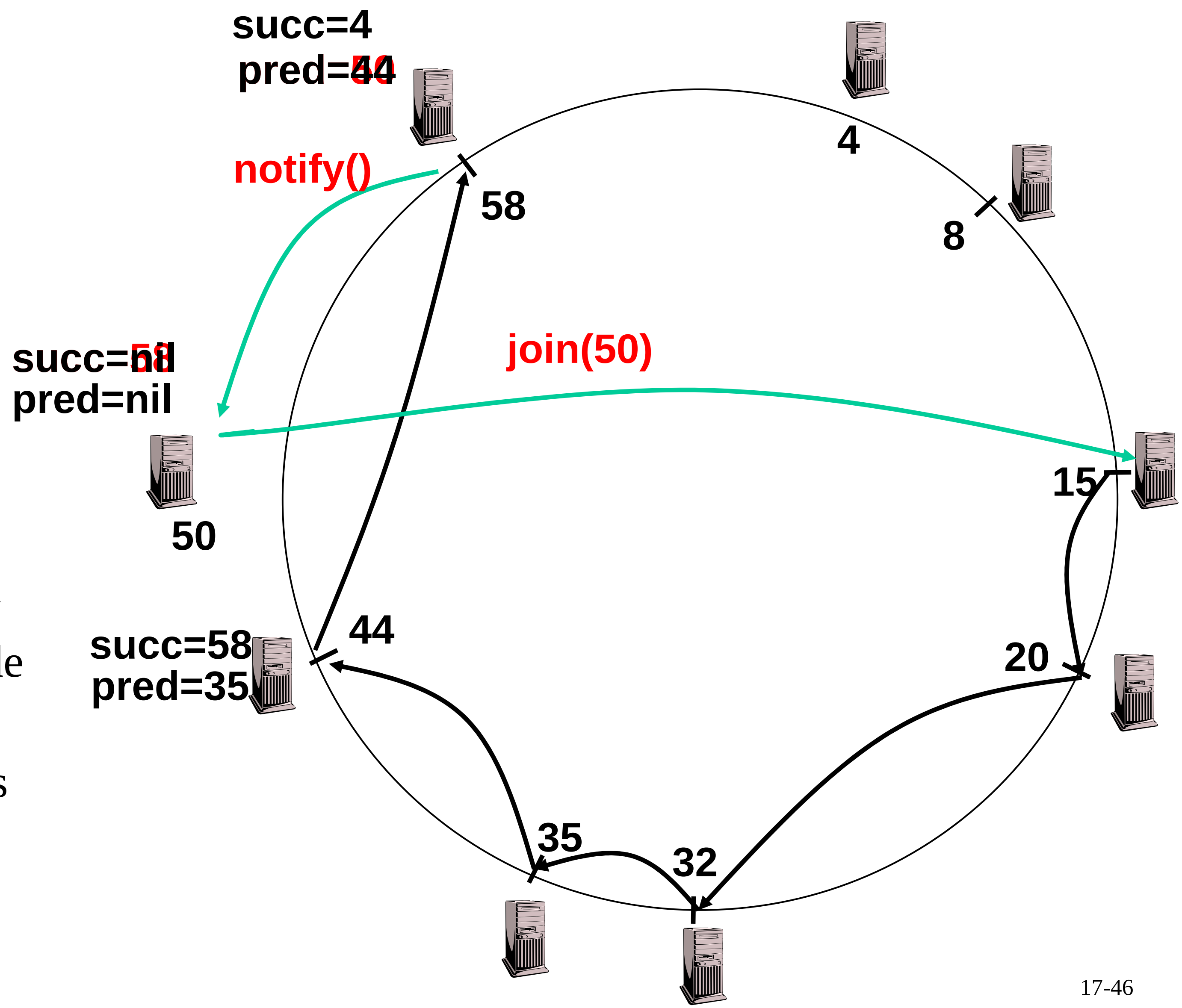
Example: Join & Stabilize (1)

- Node with id=50 joins the ring
- Node 50 needs to know at least one node already in the system
  - Assume known node is 15



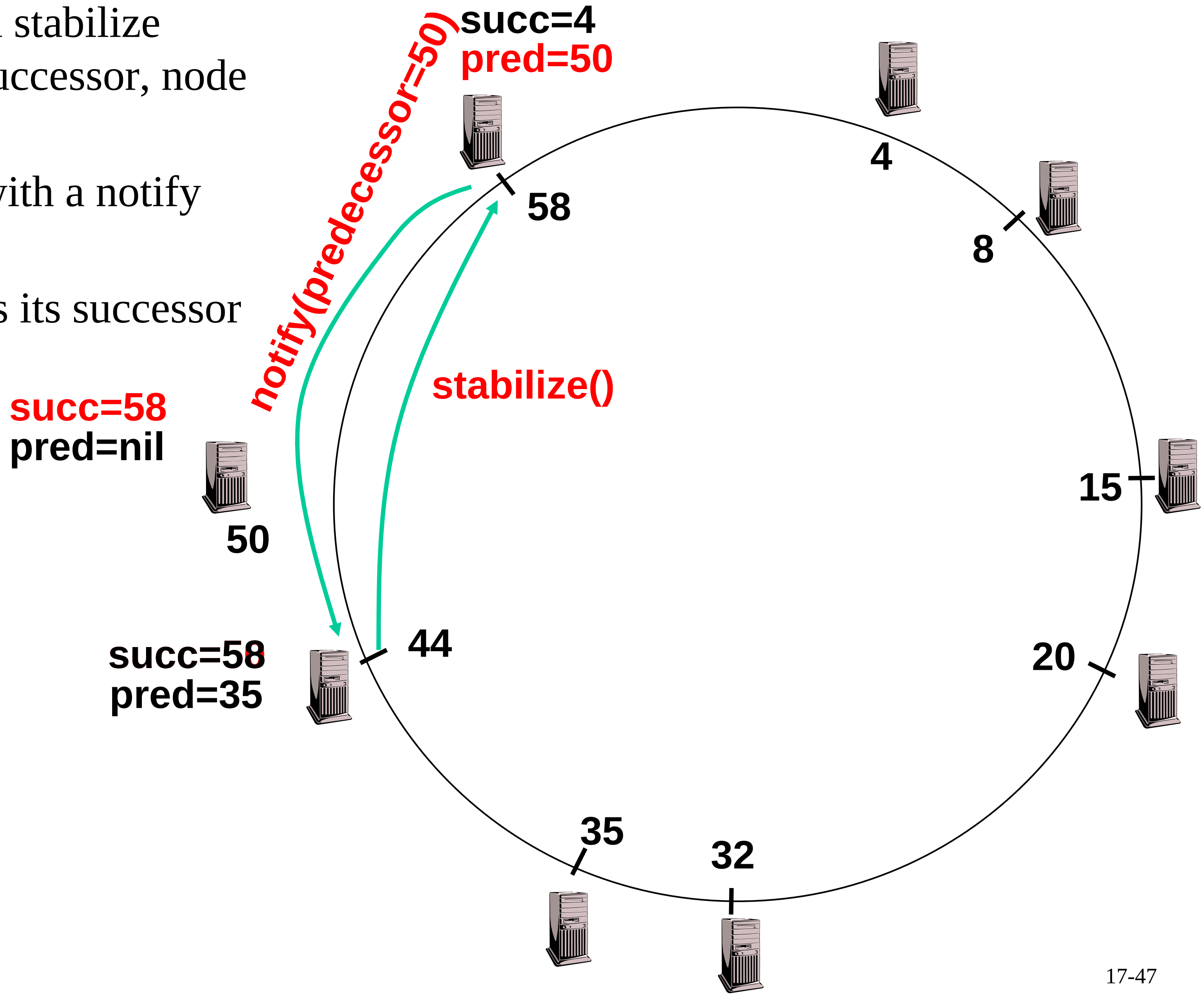
Example: Join & Stabilize (2)

- Node 50 asks node 15 to forward join message
- When join(50) reaches the destination (58), node 58
  1) updates its predecessor to 50,
  2) returns a notify message to node 50
- Node 50 updates its successor to 58



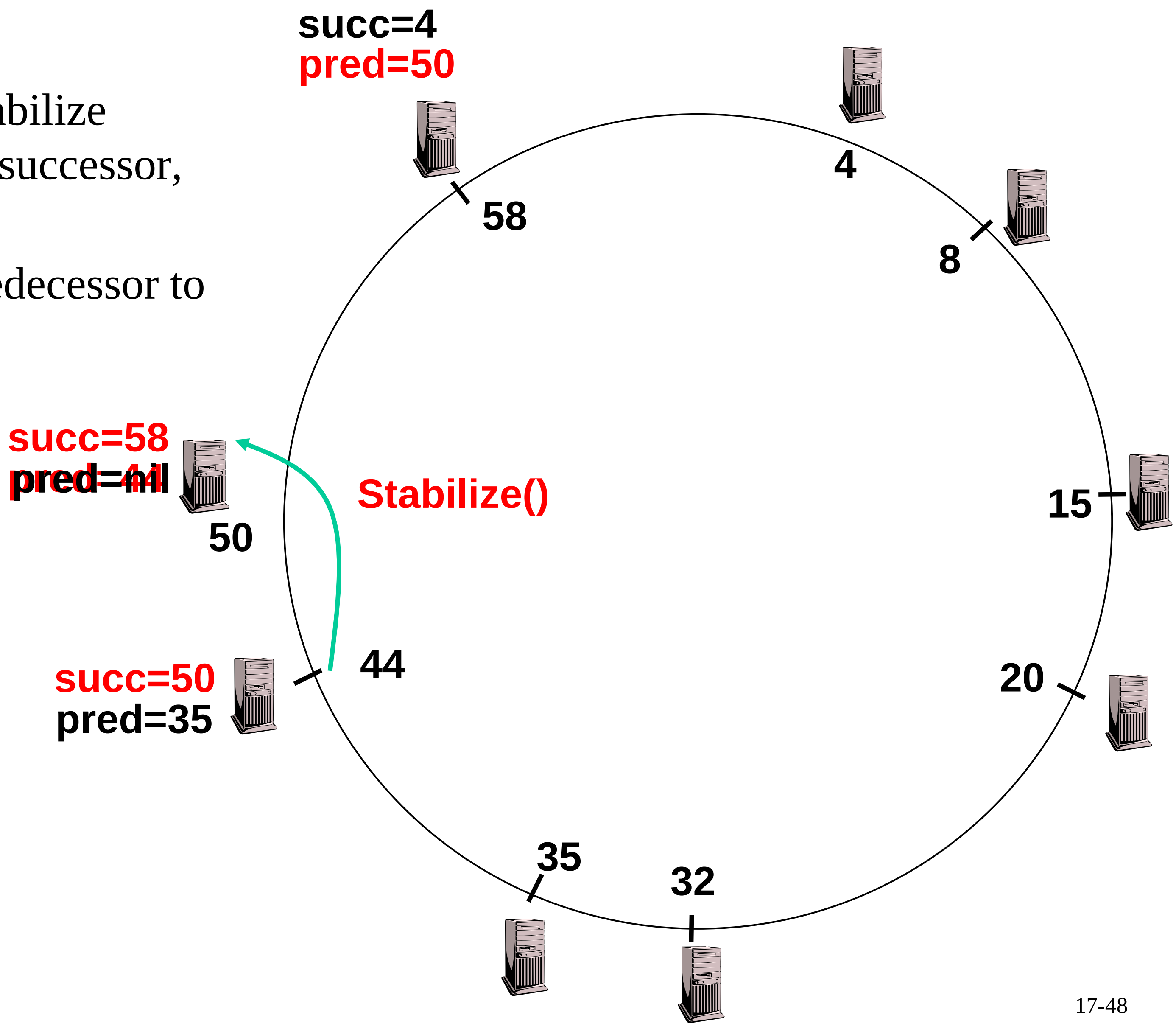
Example: Join & Stabilize (3)

- Node 44 sends a stabilize message to its successor, node 58
- Node 58 reply with a notify message
- Node 44 updates its successor to 50



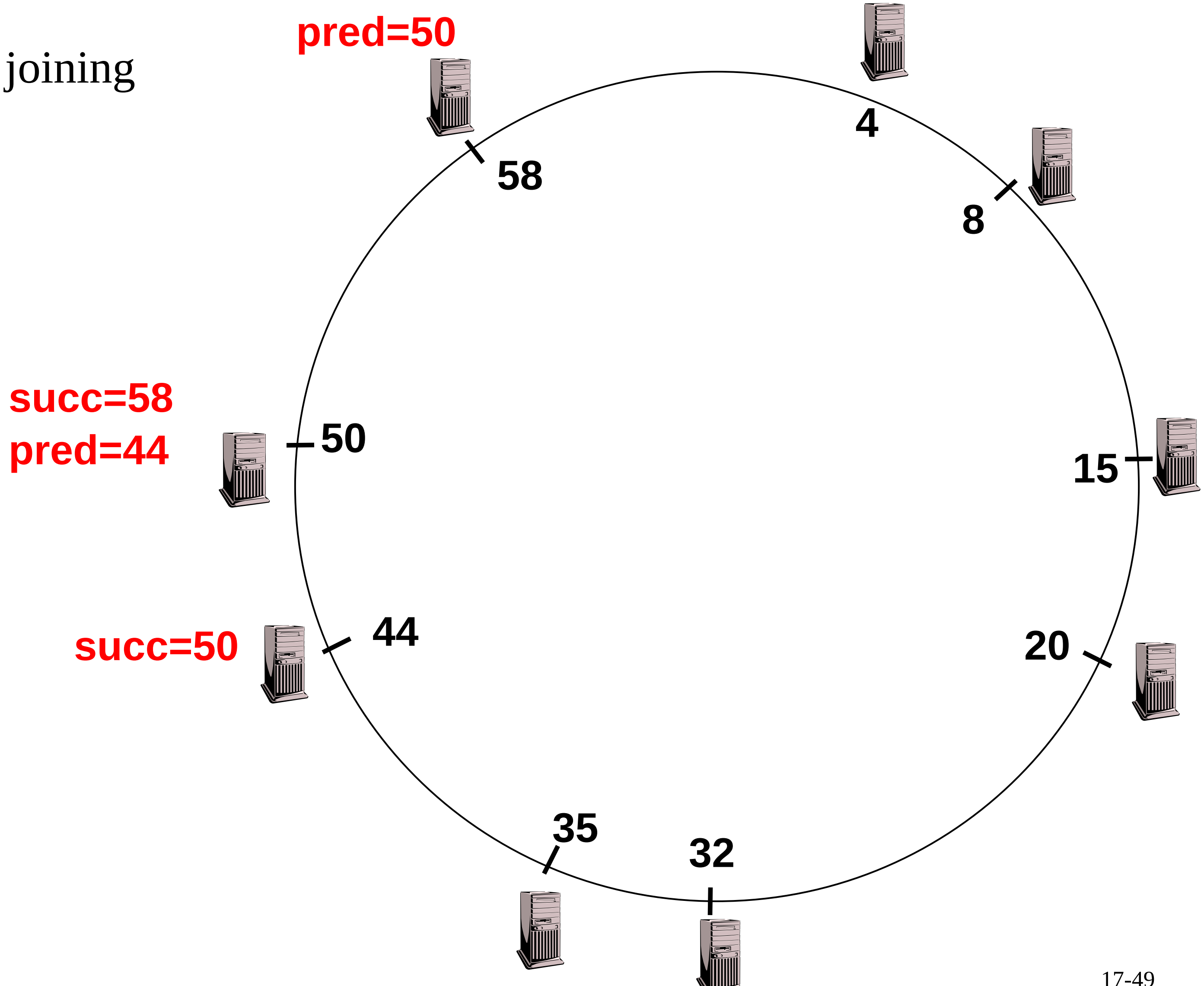
Example: Join & Stabilize (4)

- Node 44 sends a stabilize message to its new successor, node 50
- Node 50 sets its predecessor to node 44



Example: Join & Stabilize (5)

- This completes the joining operation!



Analysis of Chord Properties

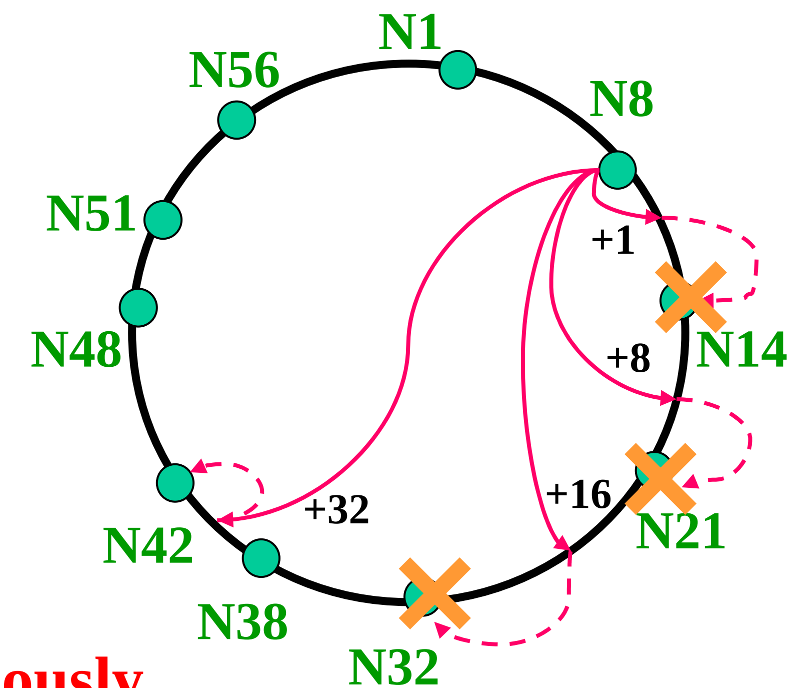
<u>Theorems</u> (with dynamics and stabilization disregarded):

- **distance between n and n.succ has expectation $2^m/n$**
- **distance is $\leq O(2^m/n * \log n)$ with probability $1 - n^{-c}$ (with $c > 1$)**
- **node density:**
  in interval of length $w*2^m/n$ there are with high prob.
  $\Theta(w)$ nodes if $w = \Omega(\log n)$ and $\leq O(w \log n)$ if $w = O(\log n)$
- **the number of nodes with finger to node x**
  has expectation $O(\log n)$ and is $\leq O(\log n)$ with high prob.

- **load balance:**
  each node holds $\leq (k/n * \log n)$ keys with high prob.
- **routing cost:**
  a lookup message has $O(\log n)$ hops with high prob.

Extensions of Chord: Failures (1)

Node failures can lead to incorrect behavior unless additional countermeasures are introduced (note that there is no global locking/synchronization among peers)



N8 could erroneously skip N38 during lookup(35)

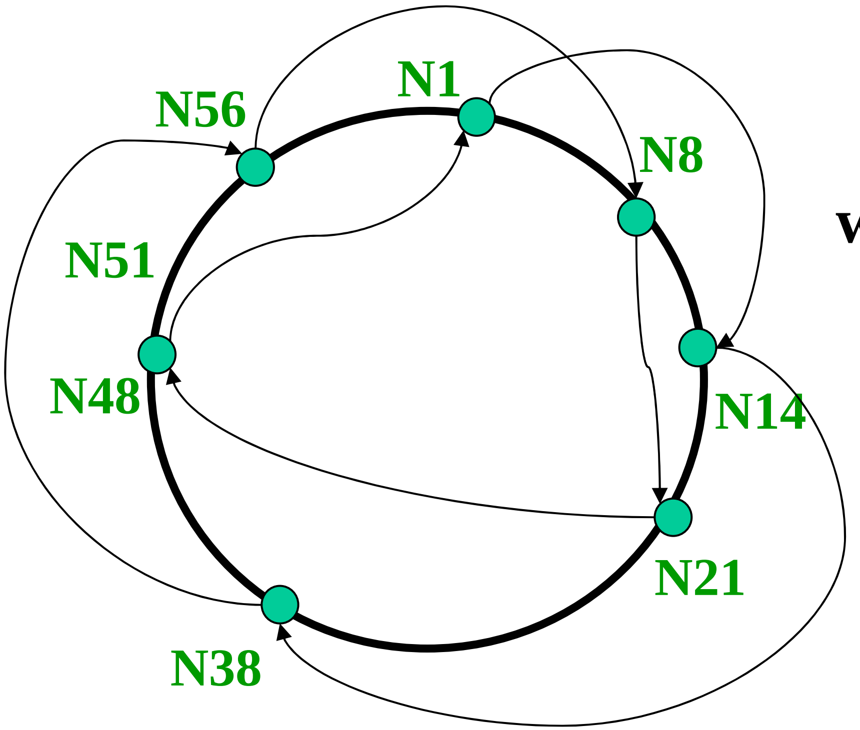
Extensions of Chord: Failures

Node failures can lead to violation of

strong consistency: all nodes form a single, doubly-linked ring, and for each n there is no x between n and n.succ

but stabilization eventually guarantees

weak consistency:
for each node n: n.succ.pred = n and n.pred.succ = n

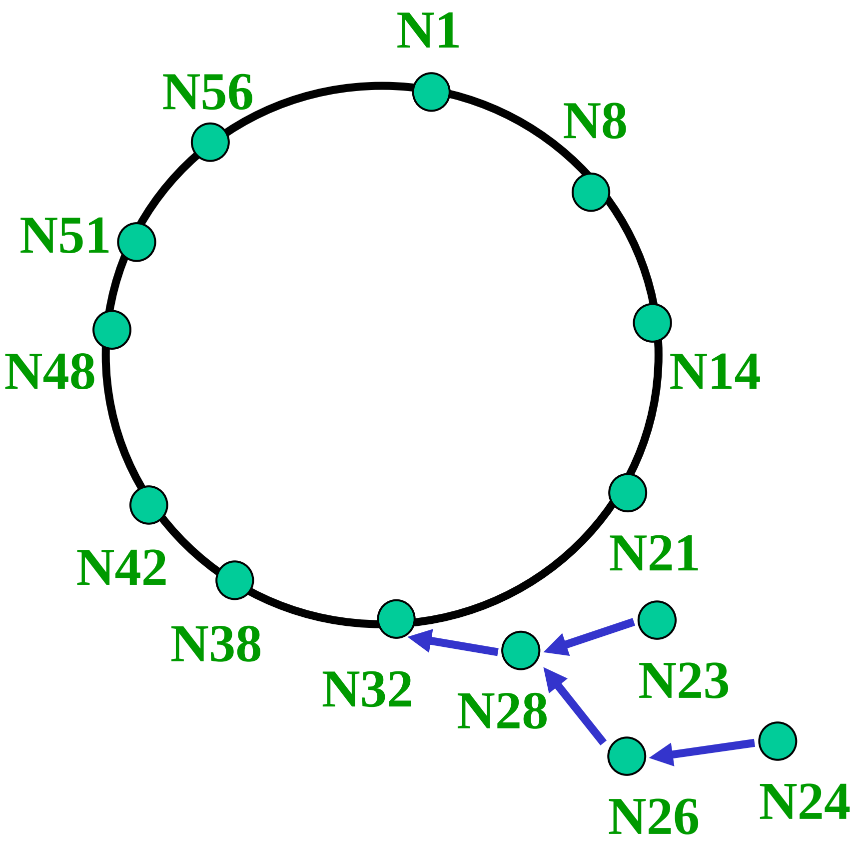


weakly, but not strongly consistent

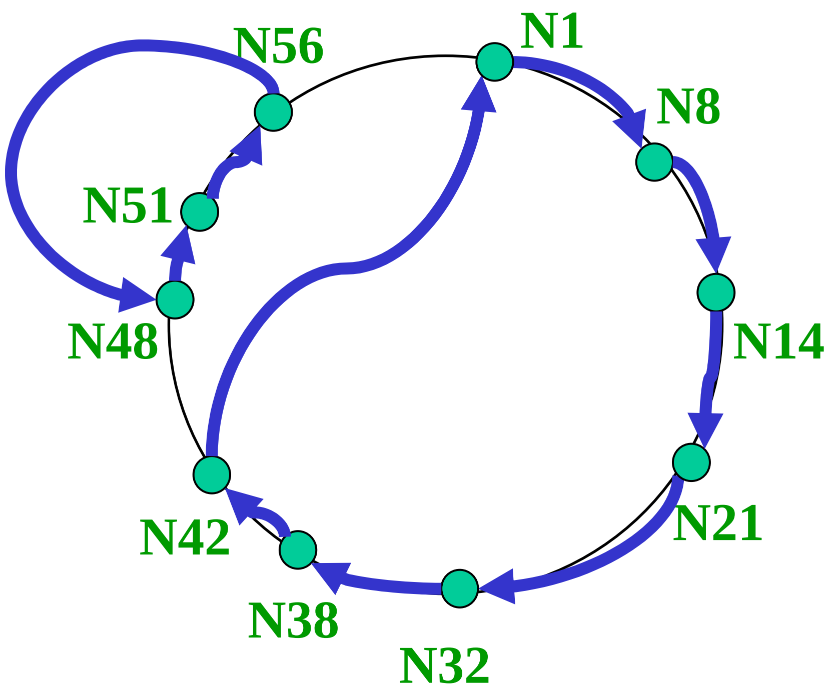
This anomaly can be avoided with high probability

Extensions of Chord: More Anomalies

Appendages



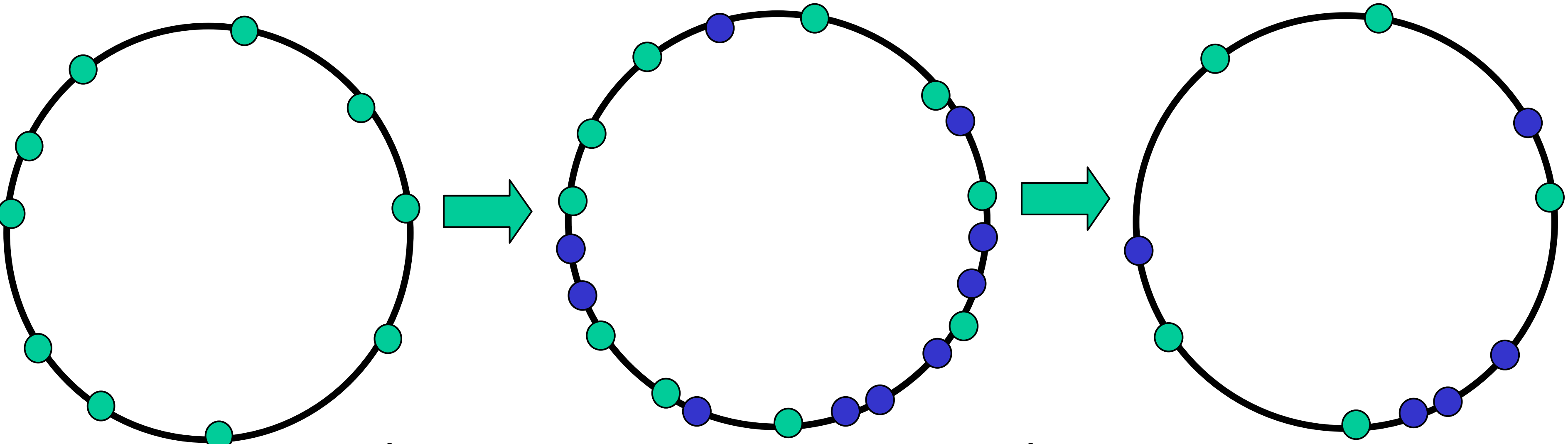
Partitioning



Problem potentially arises when churn/failure rate is high than the DHT maintenance rate

→ Solution is to ensure that stabilization runs at sufficient rate

Chord System Dynamics



Poisson process with rate λ for new nodes joining

Poisson process with rate μ for nodes leaving

$$P[\#\, new\ nodes\ per\ time\ unit = k] = e^{-\lambda}\frac{\lambda^k}{k!}$$

$$P[\#\, nodes\ leaving = k] = e^{-\mu}\frac{\mu^k}{k!}$$

$$P[time\ until\ next\ new\ node = t] = \lambda e^{-\lambda t}$$

$$P[time\ until\ next\ leaving = t] = \mu e^{-\mu t}$$

$$E[time\ until\ doubling] = \frac{1}{\lambda}N$$

$$E[time\ until\ halving] = \frac{1}{\mu}\ln 2$$

half-life τ

Limits of Self-Healing

Relationship of System Dynamics and Stabilization:

<u>Theorem:</u>
If a Chord ring runs fewer than k stabilizations per half-life τ (i.e., one node n receives $\leq$ k notifications) then it will become inconsistent (i.e., node n will be disconnected) with probability $\geq$ $\left(1-\frac{1}{e-1}\right)^k \approx 0.418^k$

<u>Corollary:</u>
A Chord ring with n nodes that stays connected with high prob. (i.e., becomes inconsistent with prob. O(1/n)) must notify Ω(log n) nodes per half-life τ

→ run stabilizations at sufficiently high rate !

Extensions of Chord: Failure Resilience and Handling of Churn

Each node n periodically checks successor s (and predecessor p) failures:
if no „heartbeat“ reply then assume that s (or p) has failed and adjust successor (pred.) pointer

For enhanced failure resilience each node maintains pointers to its next b successors (with b = Θ(log n))

For better handling of churn:
use small timeout values for assuming that non-replying node is failed, and use alternative route around presumed failures

Extensions of Chord: Data Replication

Goals:

Increase reliability:
prob. that no data will be lost by permanent failures

Increase availability:
prob. that data will be accessible despite temp. failures (or churn)

Techniques:

replicate data across independent peers by

- using multiple hash functions for assigning data items
- placing copies of the same items on b successive peers on the Chord ring
- chopping up data item into fragments and replicating fragments in random/combinatorial manner
- computing error correcting codes (ECC) for data items or fragments and carefully placing data + ECC

Extensions of Chord: Enhanced Routing

routing table size O(log n) is minimum
→ could keep routing entries for additional nodes

fingers may point to nodes that exhibit high latency (network time or node speed)
→ for forwarding lookup request, instead of using finger[i],

- **choose s (e.g., 16) random samples of nodes between finger[i-1] and finger[i],**
- **probe their IP packet round-trip time (RTT),**
- **and choose the fastest node**

(nodes can cache RTT info about sampled nodes, may form overlay network based on proximity neighbor selection)

Extensions of Chord: Combination with Random Graph or SON

Idea:

use Chord neighbors (fingers) as backbone
and add more „interesting nodes“ as neighbors

- randomly chosen nodes that yield good properties of network
  → Random (Expander) Graph
- nodes with short RTT for faster routing
- nodes with thematic similarity (w.r.t. contents or interests)
  → Semantic Overlay Network (SON)

use additional neighbors (and fingers) for message routing
(queries, postings, etc.)

Extensions of Chord: Combination with Random Graph (1)

Definitions:

An undirected graph $G=(V,E)$ is connected if there for all $x, y \in V$ there is a path $x \rightarrow^{+} y \in (E \cup E^{-1})^{+}$.

G is d-regular if every $x \in V$ has exactly $d > 1$ neighbors.

The edge boundary $\delta S \subseteq E$ of a node set $S \subset V$ is the set of edges that connect a node from S and a node from V-S.

G provides the expansion $\alpha > 0$ if for all node sets S with $|S| \le |V|/2$ the inequality $|\delta S| \ge \alpha |S|$ holds.

G is then called an expander graph.

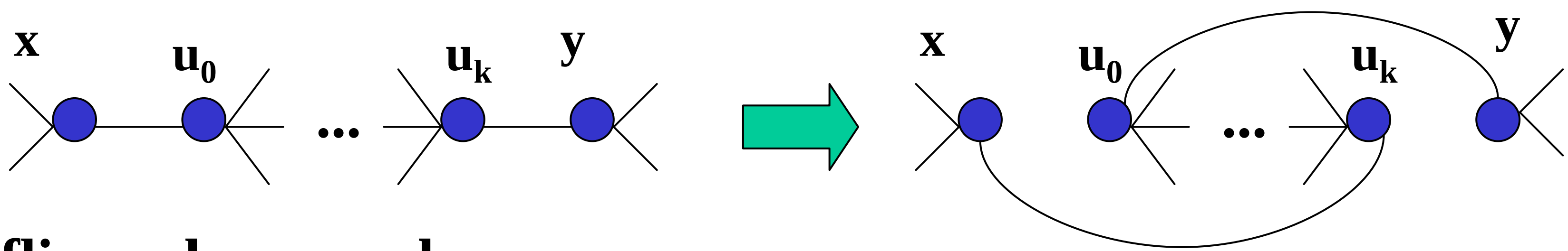
Theorem:

A random d-regular graph is a $\Theta(d)$-expander with high prob.

An expander graph has diameter $O(\log |V|)$ with high prob.

Extensions of Chord: Combination with Random Graph (2)

Start with any connected k-regular graph
Perform (uniformly chosen) random walks and apply the following random k-flipper operations:



flip nodes u_1 and u_k
(along with all their edges other than edges (x, u_0) and (u_k, y))

<u>Theorem:</u>
A series of O(dn) random k-flippers transform, with high prob. ($1-n^{-c}$ with $c>1$) any d-regular undirected graph into an expander graph for $k \in \Omega(d^2n^2 \log 1/\varepsilon)$ with any $\varepsilon>0$

Extensions of Chord: Combination with SON

Approach 1:

- **bias random walk by thematic similarity of peers**
- **apply random flipper only if newly neighboring nodes have thematic similarity above some threshold**

Approach 2:

- **remember good peers (query results & performance) in a local cache structure → *„friends“ list***
- **drop friends from or add new friends to k-neighbors list based on thematic similarity (and/or querying quality)**

Literature

- I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, H. Balakrishnan: Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications, ACM Transactions on Networking 11(1), 2003
- F. Dabek, B.Y. Zhao, P. Druschel, J. Kubiatowicz, I. Stoica: Towards a Common API for Structured Peer-to-Peer Overlays, IPTPS 2003
- D. Liben-Nowell, H. Balakrishnan, D.R. Karger: Analysis of the Evolution of Peer-to-peer Systems, PODC 2002
- F. Dabek, J. Li, E. Sit, J. Robertson, M.F. Kaashoek, R. Morris: Designing a DHT for Low Latency and High Throughput, NSDI 2004
- S.C. Rhea, D. Geels, T. Roscoe, J. Kubiatowicz: Handling Churn in a DHT, USENIX Conference, 2004
- S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, H. Yu: OpenDHT: A Public DHT Service and Its Uses, SIGCOMM Conf., 2005
- K.P. Gummadi, R. Gummadi, S.D. Gribble, S. Ratnasamy, S. Shenker, I. Stoica: Impact of DHT Routing Geometry on Resilience and Proximity, SIGCOMM 2003
- P. Linga, A. Crainiceanu, J. Gehrke, J.Shanmugasudaram: Guaranteeing Correctness and Availability in P2P Range Indices, ACM SIGMOD Conf., 2005
- P. Mahlmann, C. Schindelhauer: Peer-to-Peer Networks based on Random Transformations of Connected Regular Undirected Graphs, ACM Symp. on Parallel Algorithms, 2005
- J.X. Parreira, S. Michel, G. Weikum: p2pDating: Real Life Inspired Semantic Overlay Networks for Web Search, ACM SIGIR Workshop on Heterogeneous and Distributed Information Retrieval, 2005
- A.I.T. Rowstron, P. Druschel: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. Middleware Conf., 2001
- S. Iyer, A.I.T. Rowstron, P. Druschel: Squirrel: a Decentralized peer-to-peer Web Cache, PODC 2002
- S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker: A Scalable Content-addressable Network, ACM SIGCOMM Conf., 2001