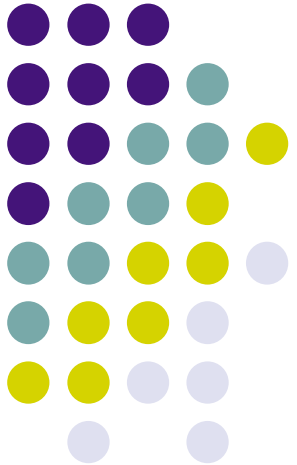
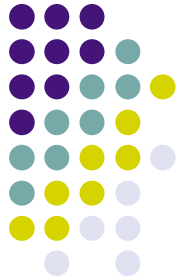
Peer 2 Peer Networks Seminar Wintersemester 06/07

System Kelips

Mykola Potapov
Institut für Informatik
Fakultät für Angewandte Wissenschaft
Freiburg, 01.03.2007



Struktur des Vortrags (1)



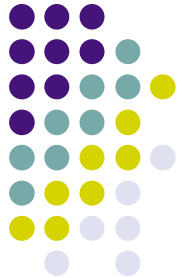
§1. Design

- 1.1 Entwickler
- 1.2 Analyse von Systemen
- 1.3 Neuer Designvorschlag

§2. Struktur von Kelips

- 2.1- 2.5 Interne Struktur
- 2.6 Platzbedarf
- 2.7 Beispiel

Struktur des Vortrags (2)



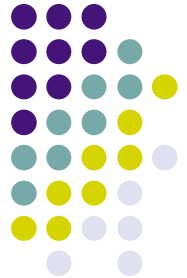
§3. Verbreitung der Information

- 3.1 Mechanismen
- 3.2 Messeging

§4. Lookup und Insertion

- 4.1 File Lookup
- 4.2 File Insertion

Struktur des Vortrags (4)



§5. Protokolle und Algorithmen

- 5.1 Kontakt im System
- 5.2 Anfragenumleitung

§7. Implementierung und Untersuchung

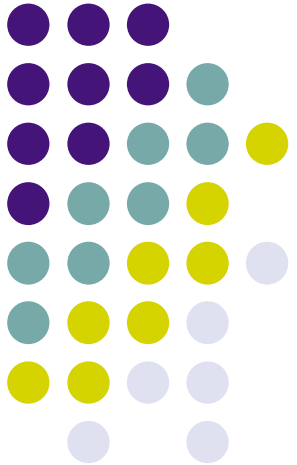
- 7.1 Simulation von Kepils
- 7.2 Ladebalance, Fault-tolerance

§8. Zusammenfassung, Literatur

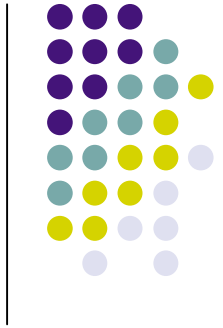
§1 Desing

Analyse von Systemen

Neuer Designvorschlag



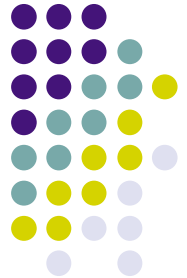
1.1 Entwickler



- Cornell University, Ithaca, NY, USA
  - Indranil Gupta
  - Ken Birman
  - Prakash Linga
  - Al Demers
  - Robbert van Renesse

{gupta, ken, linga, ademers, rvr}@cs.cornell.edu

1.2 Analyse von Systemen (1)

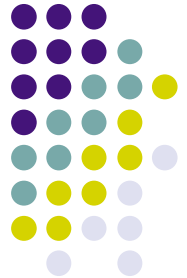


- Man betrachtet bei im Wesentlichen 3 Hauptkriterien:
  - Speicherverbrauch in jedem Knoten
    - ***In vielen Systemen logarithmisch***
  - Kommunikationskosten während der Arbeit
  - Kosten für Dateisuche
    - ***In vielen Systemen logarithmisch***

1.2 Analyse von Systemen (2)

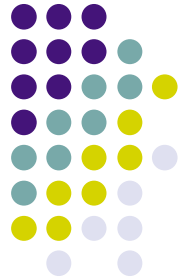
- Beispiel: Gnutella und Napster
  - ein wesentlicher Teil von Knoten kann nur mit der großen Latenz bzw. niedrigen Bandbreite verbunden werden
  - was am logarithmischen Pfad nicht erwünscht ist
    - das erhöht alle *lookup Kosten* im System
    - man möchte das vermeiden…

1.3 Designvorschlag (1)



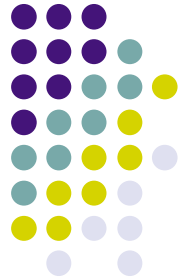
- 1. P2P Distributed Hash Table (DHT)
  - Erlaubt
    - den Hosts (Prozessen) sich leise am System an- und abmelden (an/abschließen)
    - Files (Objekten) mit bekannten Namen einzufügen bzw. diese rauszuholen
    - O(1) schnelle *lookups*

1.3 Designvorschlag (2)



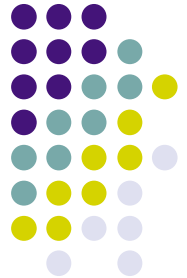
- 2. P2P Gossip (gossiping)
  - engl. für Klatsch
  - Der “Verbreitungsmechanismus“, der das teilweise Wiederholen bzw. das Kopieren der Fileindexinformation durchführt
  - Als Folge
    - ständige Kommunikation im Hintergrund (*background communication)*

1.3 Designvorschlag (3)



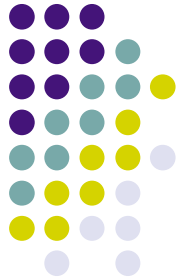
- P2P Gossip & P2P DHT
  - Indexstruktur höher Qualität
  - schnelle Konvergenz nach dem Knotenwechsel
  - *file lookup* mit O(1) Zeit und Komplexität
    - **unabhängig von der Systemgröße**
  - die Organisation von Knotenwechsel mit O(1) Zeit und Komplexität
    - **unabhängig von der Anzahl der verlorenen Knoten**

1.3 Designvorschlag (4)



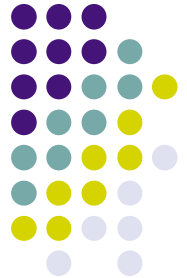
- P2P Gossip & P2P DHT
  - Somit kommen wir zur Idee
    - Speicherverbrauch etwas zu erhöhen
      - bis auf O(√n)
    - mit Kommunikationskosten im Hintergrund zu rechnen
      - diese sind aber konstant
    - Dafür aber das O(1) schnelle *lookups* zu erzielen

1.3 Designvorschlag (5)



- 3. Query Rerouting Mechanism
  - O(1) *file lookup* sogar bei Knotenverlusten (*failures*)
  - gute Ladebalance
    - *round trip time estimate* hilft die nah liegenden Peers zu finden

1.3 Designvorschlag (6)

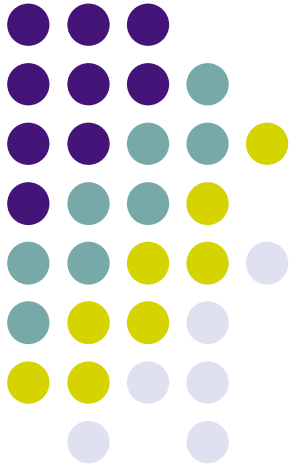


- 4. Lightweight Epidermic Multicast Protocol
  - Stabilität sogar bei Paketverlusten
    - verbreitet Informationen über
      - **Nachbarschaft**
      - **Indexierung von Files im System**
    - Als Folge „Elastizität“ des Systems

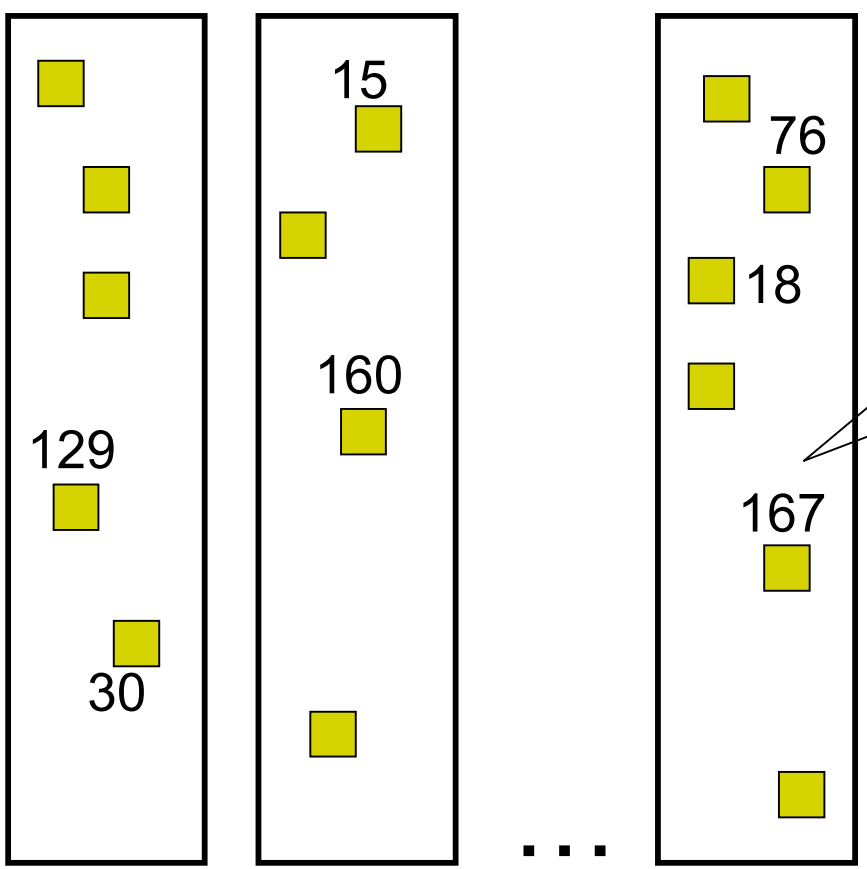
§3 Struktur von Kelips

Interne Struktur

Beispiel



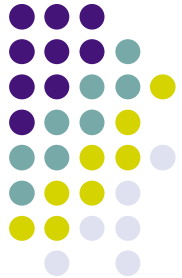
3.1 Affinity Group (1)



Gruppennummer der Knoten werden mittels einer Hashfunktion bestimmt

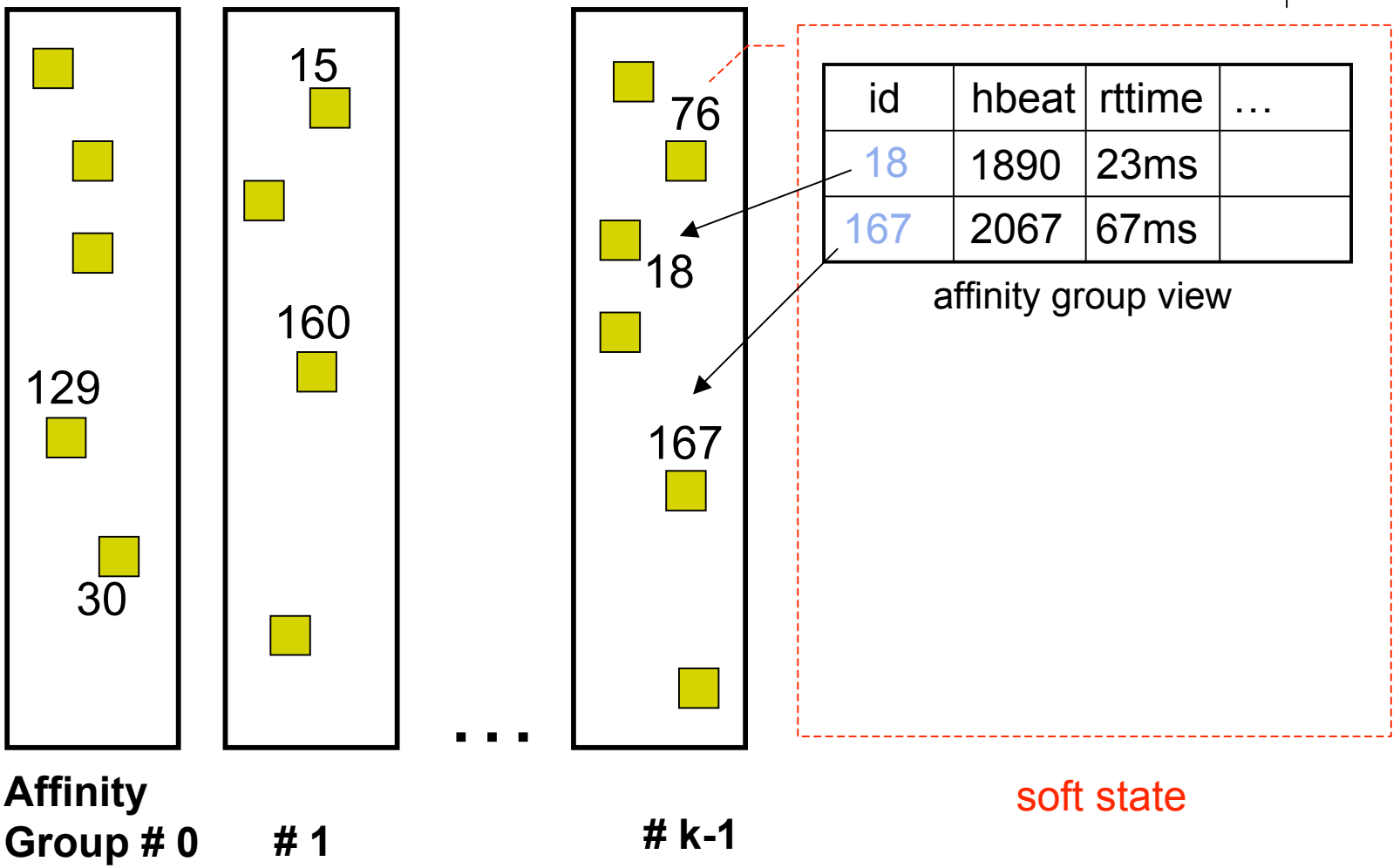
Port / IP-Adresse → int [0, k-1]

3.2 Affinity Group View (1)

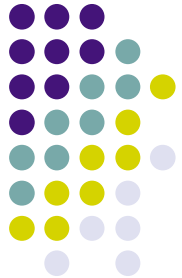


- Definition
  - Eine Menge von anderen Knoten aus der selben *affinity group*

3.2 Affinity Group View (2)

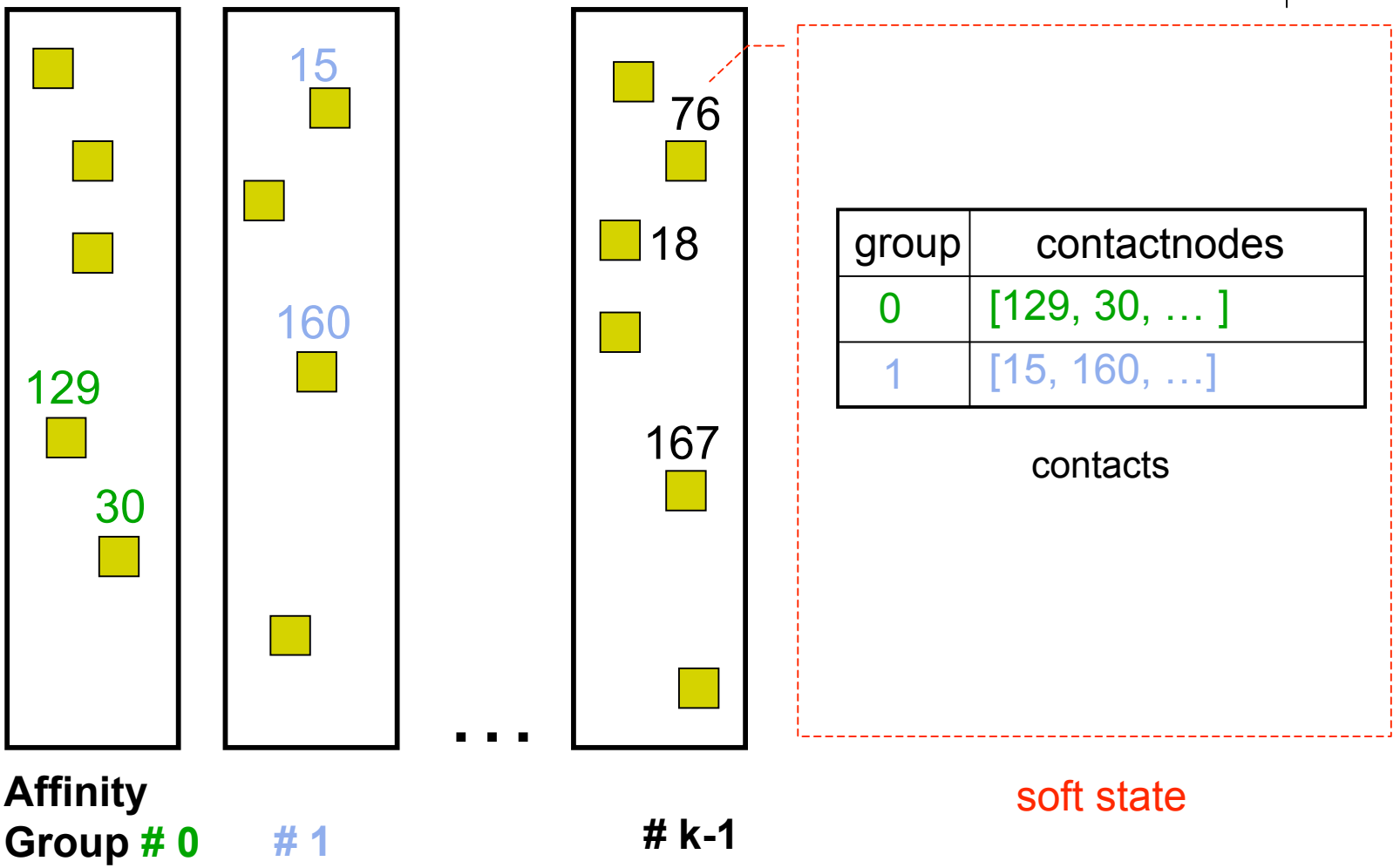


3.3 Contacts (1)

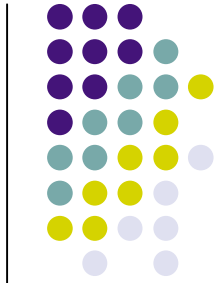


- Definition
  - Für jede *affinity group* im System eine kleine Menge von Knoten (konstanter Größe) aus der anderen *affinity group*

3.3 Contacts (2)

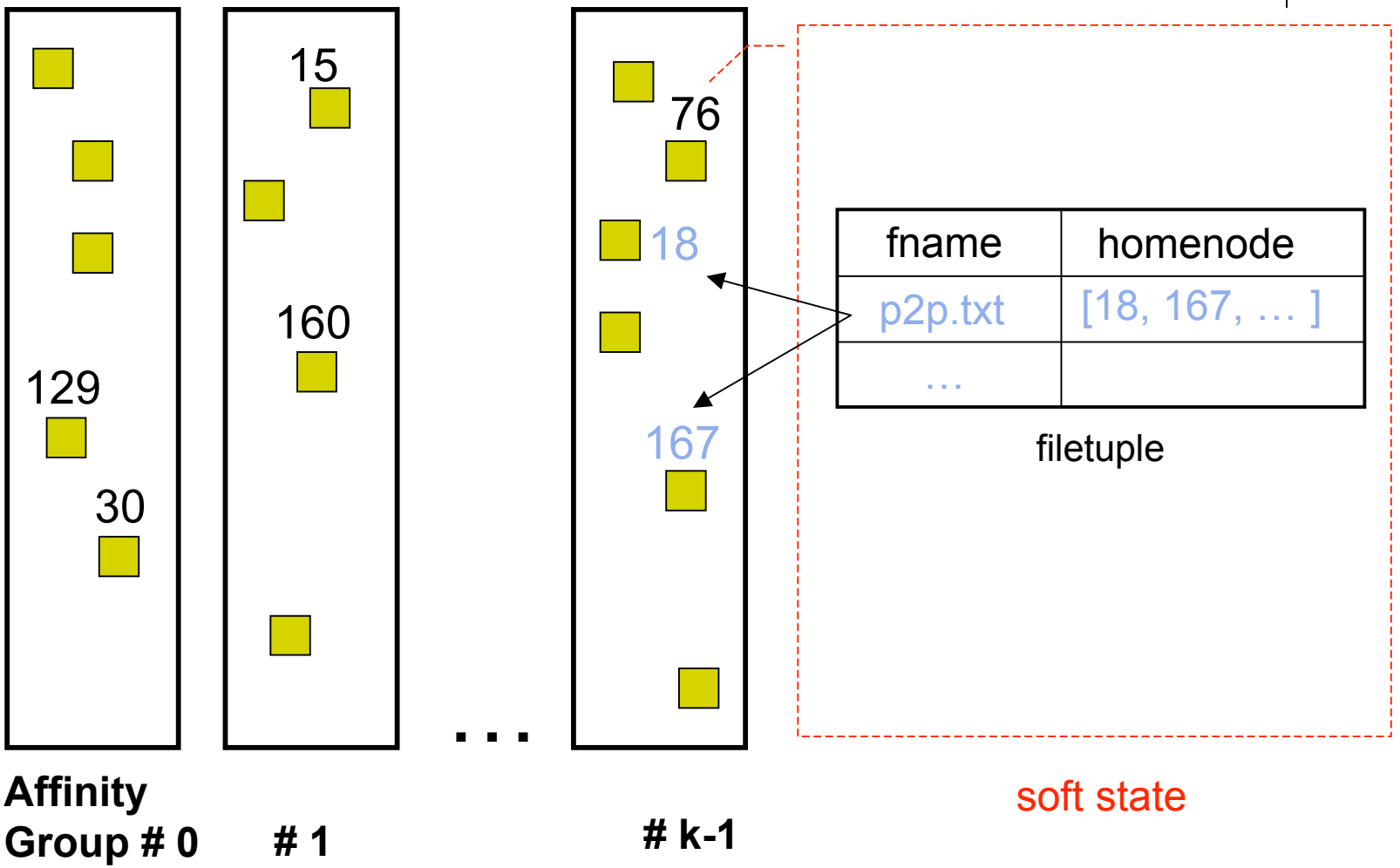


3.4 Filetuple (1)

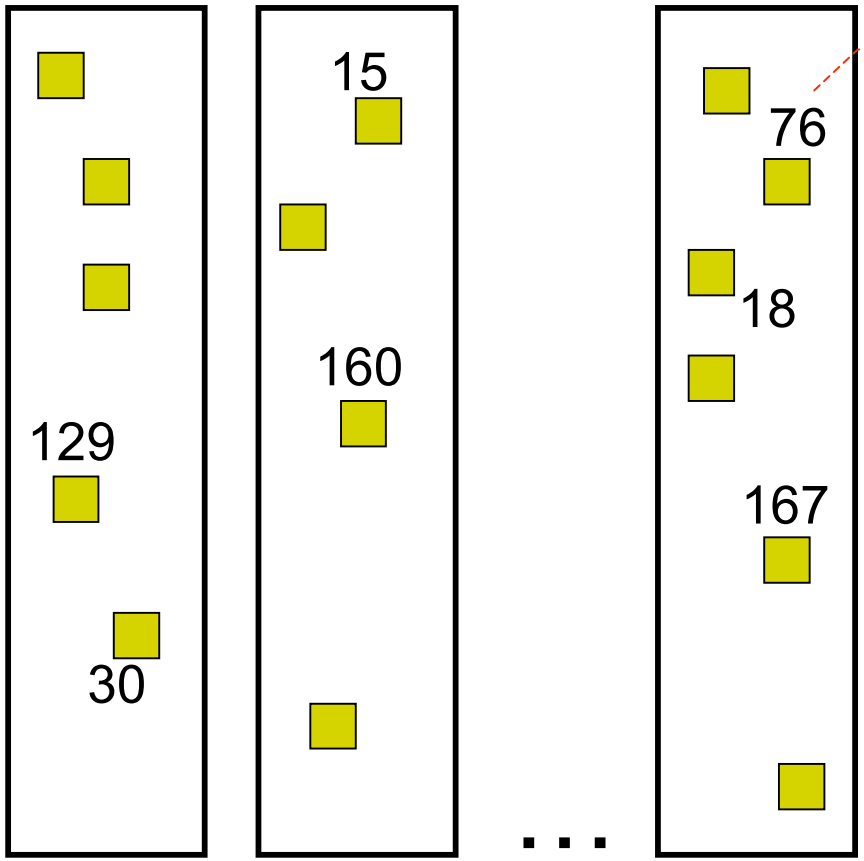


- Definition
  - Eine Menge von Tupeln mit der Information über Namen und die Platzierung von Files

3.4 Filetuple (2)



3.5 Soft State



soft state

affinity group view

id	hbeat	rttime	…
18	1890	23ms	
167	2067	67ms	

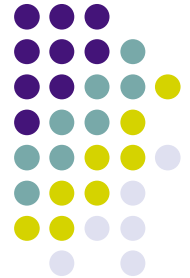
contacts

group	contactnodes
0	[129, 30, …]
1	[15, 160, …]

filetuple

fname	homenode
p2p.txt	[18, 167, …]
…	

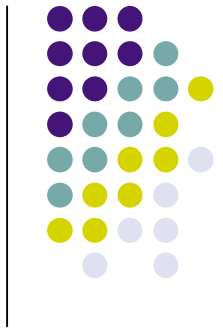
3.6 Platzbedarf (1)



- Implementierung
  - Einträge sind in *AVL-Bäumen* abgespeichert
    - → „logarithmisch-teuere“ Operationen
  - Anzahl der Einträge am Knoten beträgt

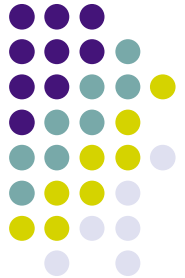
    $S(k, n) = n/k + c\cdot(k-1) + F/k$ wobei
    - n – Anzahl der Knoten im System
    - k – Anzahl der *affinity groups*
    - c – Anzahl der Kontakten aus der fremden *affinity group*
    - F – die Gesamtanzahl der Dateien im System ist

3.6 Platzbedarf (2)



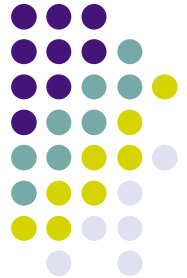
- Implementierung
  - Für ein festes n gilt $k = \sqrt{(n + F)/c}$
  - Sei die F proportional zu n und c steht fest, dann
    - $k \approx O(\sqrt{n})$
    - $S(k, n) \approx O(\sqrt{n})$
    - ein gutes Ergebnis für viele mittelgroße p2p Systeme!

3.7 Beispiel (1)



- Gegeben sei ein System mit
  - *n = 100 000 peers*
  - *k = √n = 317 affinity groups*
  - *60byte große filetuple entries*
  - *40byte große membership entries*
  - 2 Kontakten pro fremde *affinity group*
  - *10 Millionen files*
  - node soft state (Speicherverbrauch am Knoten*)*
    beträgt nur 1,93Mb

3.7 Beispiel (2)



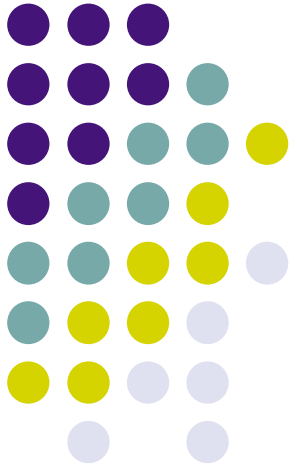
- Folge: Effizienz vs. Kosten
  - mit dem Speicherverbrauch am Knoten nur 1,93MB liefern die Suchanfragen die Platzierung (den Ort) der Datei mit O(1) Zeit und Komplexität

    → **schnell und nicht sehr teuer**

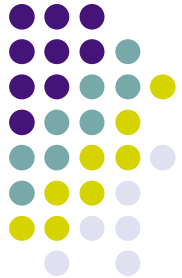
§3 Verbreitung der Information

Mechanismen

Messeging

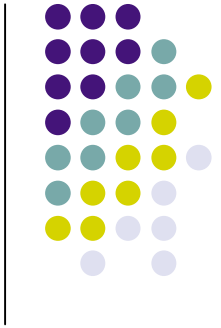


3.1 Mechanismen (1)



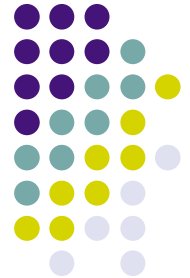
- 1. Heartbeat Mechanism:
  - Alle Einträge in *file tupels* werden periodisch innerhalb und außerhalb von Gruppen aktualisiert:
    - jeder im Knoten abgespeicherte Eintrag ist mit einem *integer heartbeat*-Zähler assoziiert

3.1 Mechanismen (2)



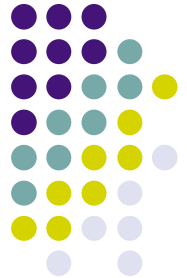
- Bedingung für jeden solchen Eintrag:
  - keine Veränderung des *heartbeat*-Zähler während einer vorher bestimmten Zeitperiode führt zum Löschen des Eintrags
  - Als Folge:
    - Hintergrundkommunikation (*background communication*) innerhalb der Gruppe
    - Das Miteinziehen ins System neuer Einträge für *group views, contacts, file tuples*

3.2 Mechanismen (3)



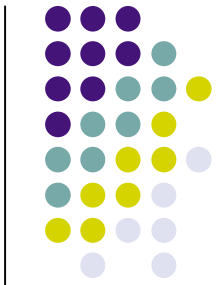
- 2. Gossiping, Multicasting
  - Der Knoten erhält Information und erzählt darüber eine bestimmte Anzahl von Runden (*rounds*):
    - selektiert eine kleine konstant große* Menge von Knoten aus der Nachbarschaft
    - jeder Nachbar erhält eine Kopie der Information
    - Bandbreite bleibt konstant (wegen*)
    - steigende Latenz als Folge (wächst polylogarithmisch mit der Gruppengröße)

3.2 Messeging (1)



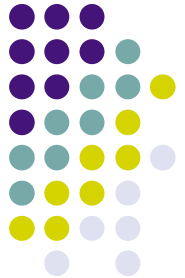
- *3. Lightweight Unreliable Protocol*
  - Wie UDP
  - Auswahl der Knoten passiert mittels *round-trip-time esimate (rtt)*:
    - wer ist topologisch am nähersten im Netzwerk

3.2 Messeging (2)



- Fakten
  - Experimentelle Studien über *gossip-style*-Protokolle zeigen:
    - sie sind robust zu plötzlichen Paket- und Knotenverlusten
    - Erzielen eine stabile Informationsverbreitung in *affinity groups*

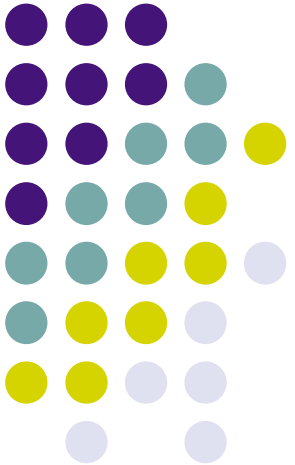
3.2 Messeging (3)



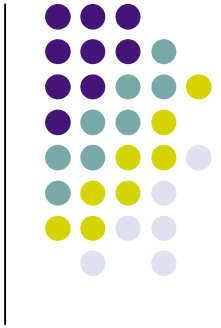
- Beim Knotenwechsel
  - Konvergenzzeiten:
    - Der Latenzfaktor ist O(√n) (Verzögerung)
    - O((√n) * log2(n)) für *group view*- und *filetuple*-
    - O((√n) * log3(n)) für *contact*-Einträge

§4 Lookup und Insertion

Funktionsweise

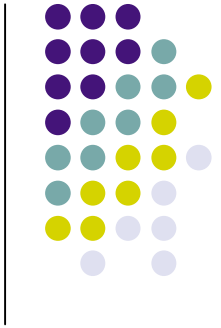


4.1 Lookup (1)



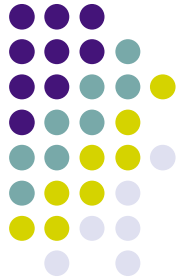
- 1. Zugriff auf Files
  - Der Knoten macht folgendes
    - Der Dateiname wird von dem suchenden Knoten in der entsprechenden (passenden) *affinity group* abgebildet
    - Dabei wird das *Hashing* benutzt
    - Die Suchanfrage wird an den nah liegenden Kontakt aus seiner *affinity group* geschickt

4.1 Lookup (2)



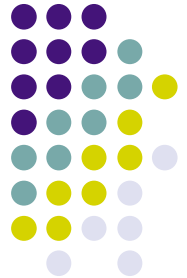
- 1. Zugriff auf Files
  - Der Knoten macht folgendes
    - Die Antwort auf die Anfrage wird in *filetuples* gesucht
    - und als die Adresse vom *homenode* der Datei geliefert
    - Der suchende Knoten kann auf die Datei zugreifen
    - *O(1) Zeit und Komplexität*

4.2 Insertion (1)



- 2. Das Einfügen von Files
  - Analog…
    - Der Kontakt holt (*randomisiert*) einen Knoten *h* aus seiner *affinity group* und übergibt ihm die Einfügeanfrage

4.2 Insertion (2)

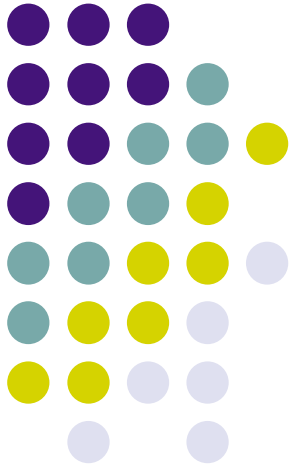


- 2. Das Einfügen von Files
  - *h* ist der *homenode* der Datei
  - die Datei *f* wurde erfolgreich an den *homenode* übertragen
  - Der neue *filetuple* wird erstellt um die neue Datei *f* im *homenode* abbilden zu können
    - *O(1) Zeit und Komplexität*

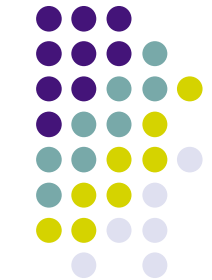
§5 Protokolle und Algorithmen

Kontakt im System

Anfragenumleitung

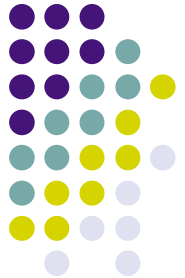


5.1 Kontakt im System (1)



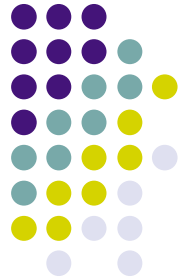
- Joing Protokoll
  - Der Kontakt passiert mittels eines bekannten introducer-Knotens
    - URL kann benutzt werden
  - Der joiner-Knoten nutzt Information des introducer-Knotens aus:
    - erneut *sein soft state (group view, filetuple, contacts)*
    - der *gossip-stream* verbreitet schnell Information über den joiner-Knoten im System

5.1 Kontakt im System (2)



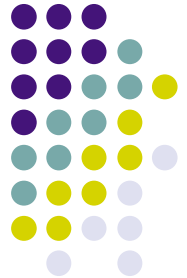
- Contact Policy
  - Die Anzahl von Kontakten steht fest
    - Es existiert eine *contact-replacement policy*
    - kann *lookup / insertion* beeinflussen
    - *proaktiv / reaktiv* arbeiten
    - achtet auf Faktoren wie Knotenentfernung und Zugänglichkeit (z.B. *firewalls* unterwegs)

5.2 Anfragenumleitung (1)



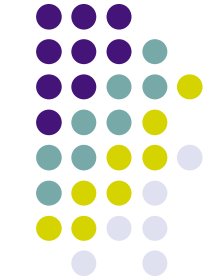
- Query Routing
  - Wiederholung der Anfrage beim Anfrageverlust (*query retrie*)
    - Abfrage von mehreren Kontaktknoten
    - *Query Forwarding* innerhalb *affinity groups der Kontakten*
    - Ausführung der Anfrage an einem anderen Knoten aus der selben *affinity group*

5.2 Anfragenumleitung (2)



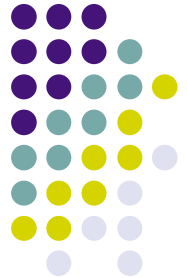
- Funktionsweise:
  - während einer TTL-Zeit
    - Randomisierter Durchlauf innerhalb der *file affinity group* in Kontakten (Fall 2)
    - Durchlauf innerhalb der *affinity group* eines anderen Knotens aus der *affinity group* des fragenden Knotens (Fall 3)

5.2 Anfragenumleitung (3)



- Funktionsweise
  - die TTL-Zeit hängt ab von
    - der maximalen Anzahl der Anfrageversuche zwischen den so genannten
      - *lookup query success rate* und
      - *maximum processing time*
    - die *lookup*- und *messeging*-Zeit bzw. Komplexität bleiben O(1)

5.2 Anfragenumleitung (4)

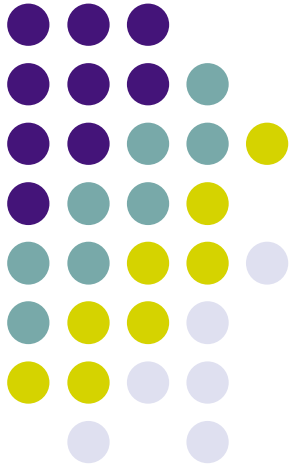


- Anfrageverlust beim *insert*
  - Funktioniert analog
  - Der Hauptunterschied:
    - die Datei wird an dem Knoten eingefügt, wo die TTL-Zeit abläuft
    - Gute Ladebalance wird erreicht
    - Die Zeit im Normalfall beträgt $O(\log \sqrt{n})$
      - das ist konkurrierbar mit den existierten Systemen

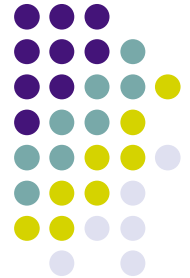
§6 Implementierung und Untersuchung

Simulation von Kelips

Interessante Ergebnisse

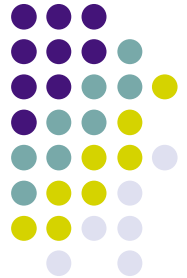


6.1 Simulation von Kelips (1)



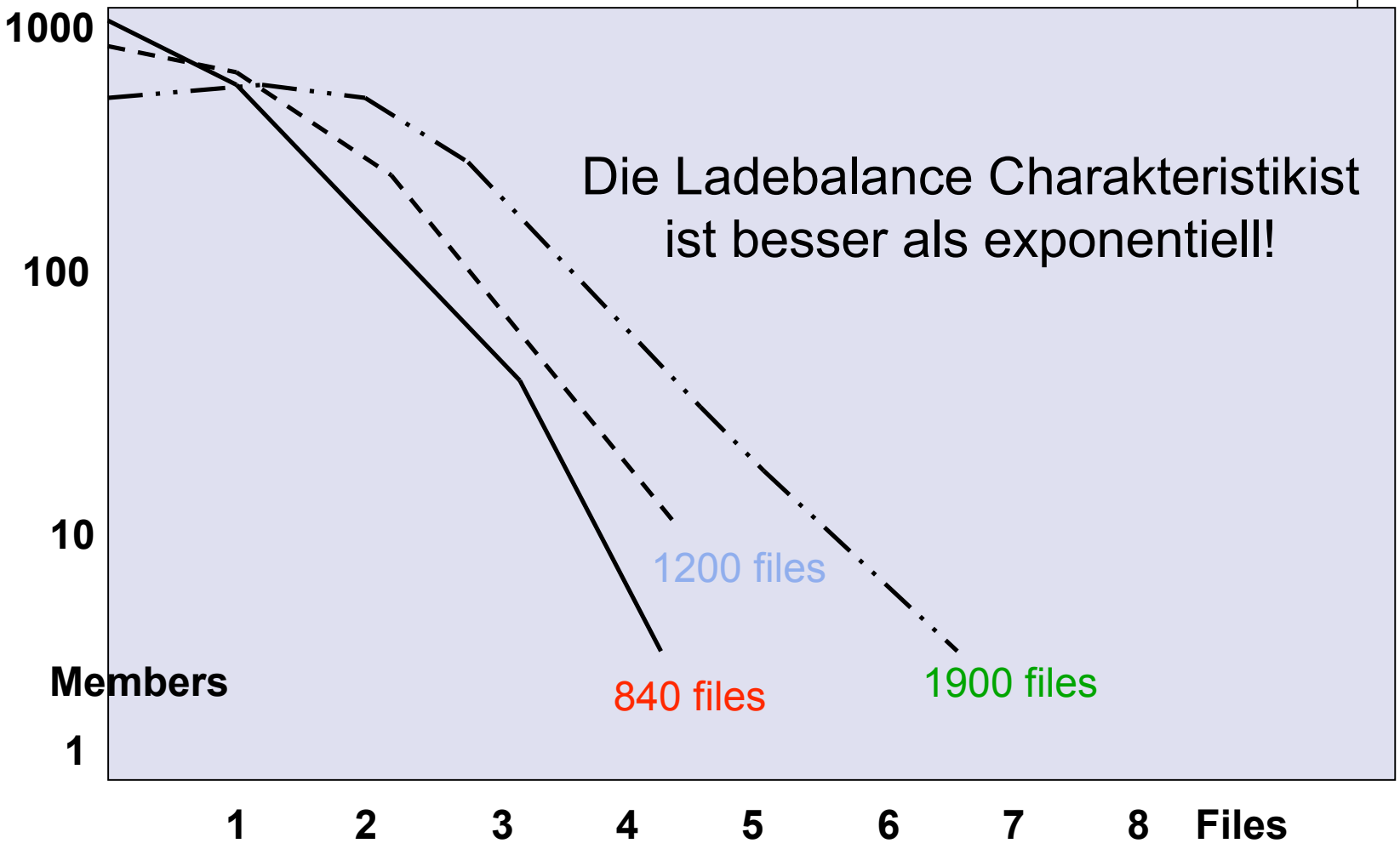
- C WinAPI Implementierung von Kelips
  - Mehrere Knoten werden an einem Rechner simuliert
    - 1GHz CPU, 1GB RAM, Win2K
    - emulierte topologische Netzwerkarchitektur
    - somit begrenzte Resoursen und Speicher
    - → Systemgröße nicht mehr als ein paar Tausend Knoten
    - *gossiping* passiert jede 2 Sekunden
    - Nachrichtgröße maximal 272byte

6.1 Simulation von Kelips (2)



- Ladebalance
  - Dateien werden in das System eingefügt
  - Dateinamendistribution:
    - es wird eine Reihe von anonymen URLs genommen

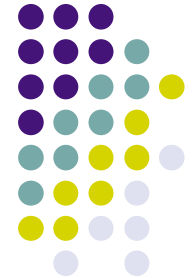
Ladebalance



6.1 Simulation von Kelips (3)

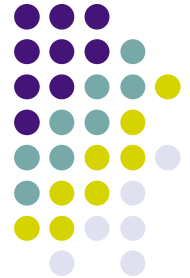
- *Multy-Try Multi-Hop Sheme*
  - Elastizität bei Abweichungen im System
  - Systemstabilität sogar im Falle der dynamischen Veränderungen
    - sogar wenn die hälfte der Knoten fällt
    - O(1)-schnelle *lookups* werden trotzdem erzielt

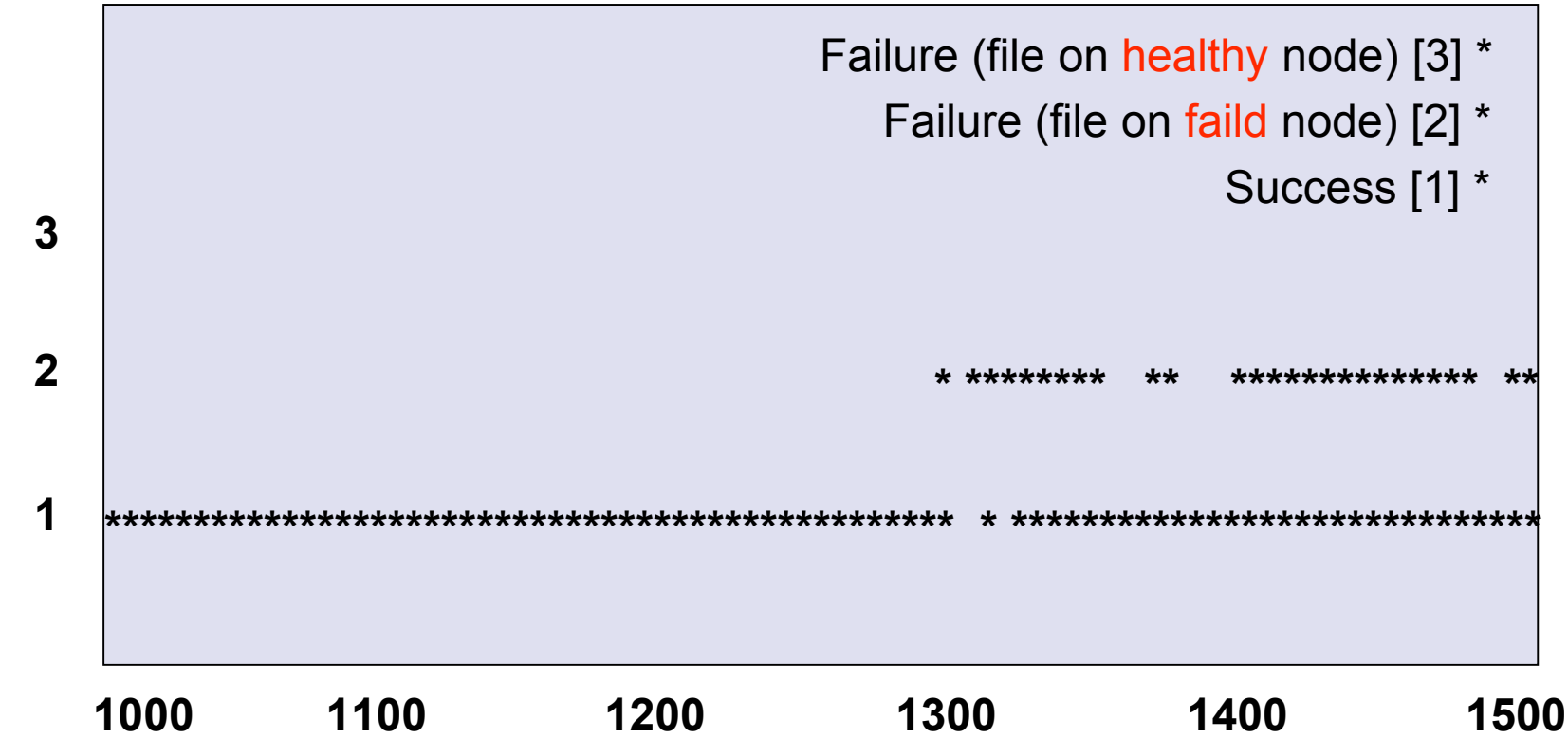
6.1 Simulation von Kelips (4)



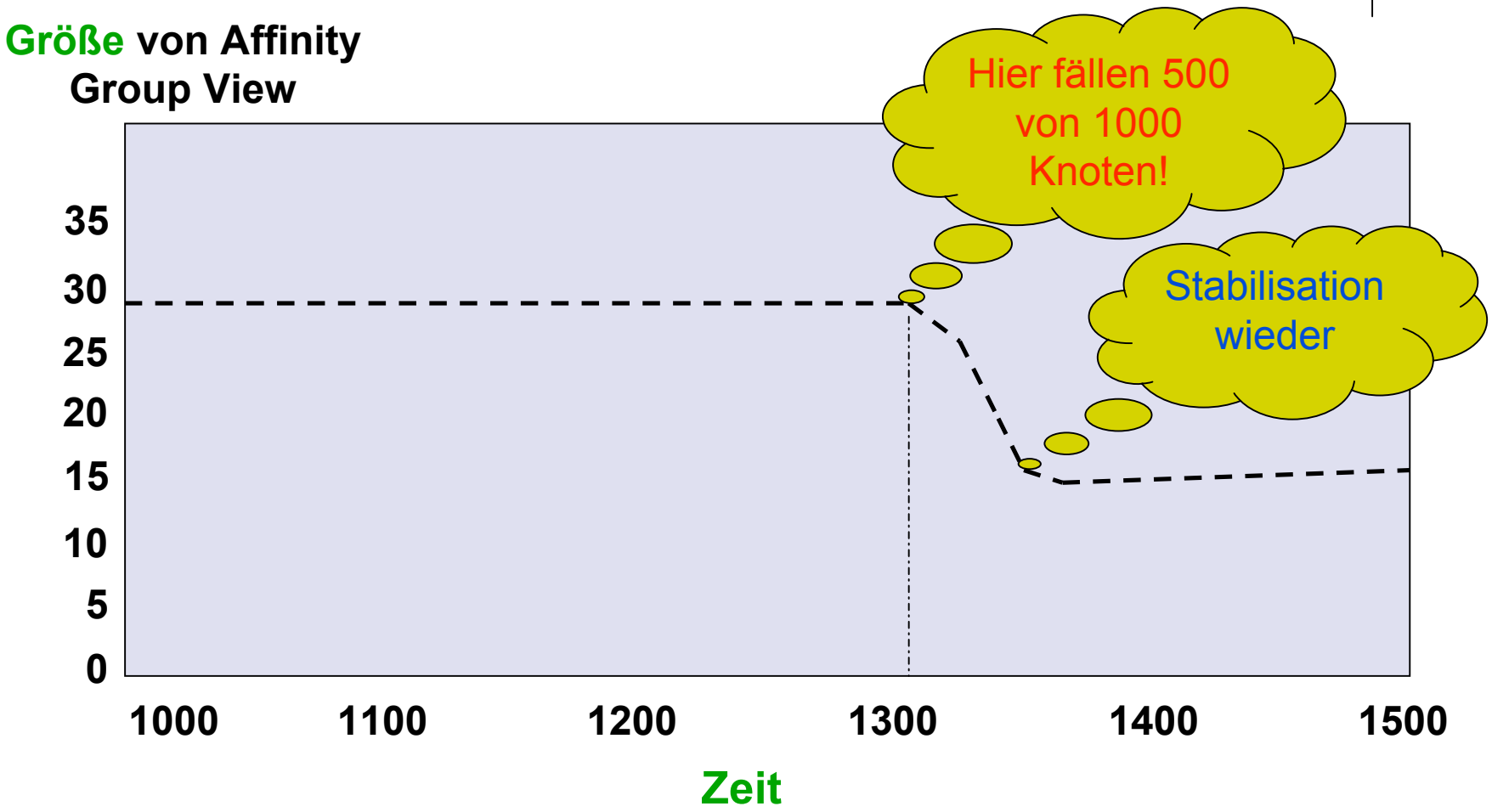
- File Insertion. Funktionsweise
  - *Mehrversuch-Mehrsprung-Schema*
    - max. 4 Versuche, TTL hat 3log(n) logische Sprunge
    - Das Einfügen von 1000 verschiedener Files zeigt:
      - 66,2% im ersten Versuch
      - 33% - im zweiten
      - 0,8% - im dritten
    - Nichts ging verloren
    - Nur 3 Versuche waren nötig

Fault-tolerance of Lookups (1)



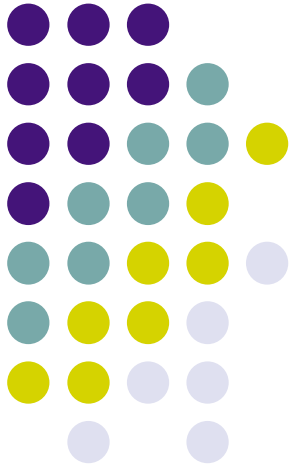


Fault-tolerance of Lookups (2)

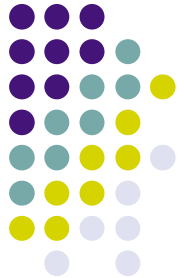


§7 Zusammenfassung

„Lange rede kurzer Sinn“

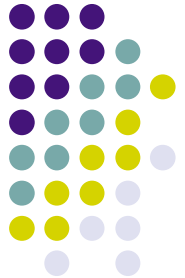


7.1 Zusammenfassung (1)



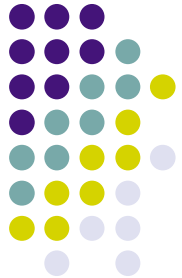
- Interner Design:
  - benutze Speicher des Knotens
  - rechne mit Kommunikationskosten im Hintergrund
  - erziele aber das O(1)-schnelle *lookups*

7.1 Zusammenfassung (2)



- Interner Design:
  - **p2p distributed hash table**
  - **p2p gossiping**
  - ***lightweight epidermic multicast protocol***
  - ***multi-hop-multi-try query routing***
  - erlauben stabiles, erfolgreiches und schnelles *lookup bzw. insertion* sogar bei der Bandbreitebegrenzung und den Verbindungsabbrüchen

Literatur (1)

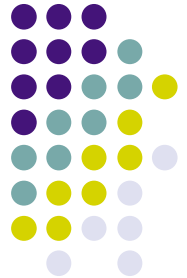


[1] N.T.J. Bailey, „Epidemic Theory of Infectious Diseases and its Applications", Hafner Press, Second Edition, 1975.

[2] K.P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, Y. Minsky, „Bimodal Multicast", ACM Trans. Comp. Syst., 17:2, pp. 41-88, May 1999.

[3] F. Dabek, E. Brunskill, M. F. Kaashoek, D.Karger, „Building peer-to-peer systems with Chord, a distributed lookup service", Proc. 8th Wshop. Hot Topics in Operating Syst., (HOTOS-VIII), May 2001.

Literatur (2)



[4] A. Demers, D.H. Greene, J. Hauser, W. Irish, J. Larson „Epidemic algorithms for replicated database maintenance", Proc. 6th ACM Symp. Principles of Distributed Computing (PODC), pp.1-12, 1987.

[5] D. Kempe, J. Kleinberg, A. Demers, „Spatial gossip and resource location protocols", Proc. 33rd ACM Symp. Theory of Computing (STOC), pp. 163-172, 2001.

[6] A. Rowstron, P. Druschel, „Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems", Proc. IFIP/ACM Middleware, 2001.

Vielen Dank für die Aufmerksamkeit

System Kelips

Mykola Potapov
Institut für Informatik
Fakultät für Angewandte Wissenschaft
Freiburg, 01.03.2007