Lectures in Wroclaw

- **Epidemic Algorithms**
  - Monday, April 6th, 2009, 3pm
- **Random Networks**
  - Monday, April 6th, 2009, 6pm
- **Distributed Heterogeneous Hash Tables**
  - Tuesday, April 7th, 2009, 3pm
- **Network Coding**
  - Wednesday, April 8th, 2009, 11am
- **Locality in Peer-to-Peer Networks**
  - Wednesday, April 8th, 2009, 3pm

ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Algorithms and Methods for Distributed Storage Networks

9 Analysis of DHT

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik

Wintersemester 2007/08

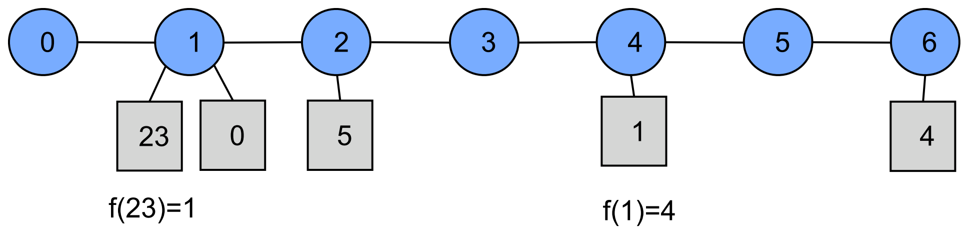
CoNe Freiburg

IIF INSTITUT FÜR INFORMATIK FREIBURG

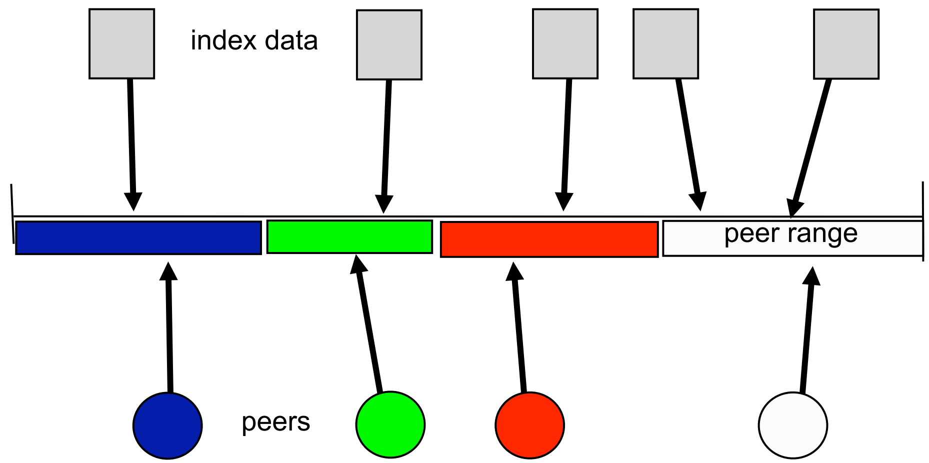
Distributed Hash-Table (DHT)

- **Hash table**
  - does not work efficiently for inserting and deleting
- **Distributed Hash-Table**
  - servers are „hashed“ to a position in an continuos set (e.g. line)
  - data is also „hashed“ to this set
- **Mapping of data to servers**
  - servers are given their own areas depending on the position of the direct neighbors
  - all data in this area is mapped to the corresponding server
- **Literature**
  - *“Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web”,* David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, Rina Panigrahy, STOC 1997

Pure (Poor) Hashing

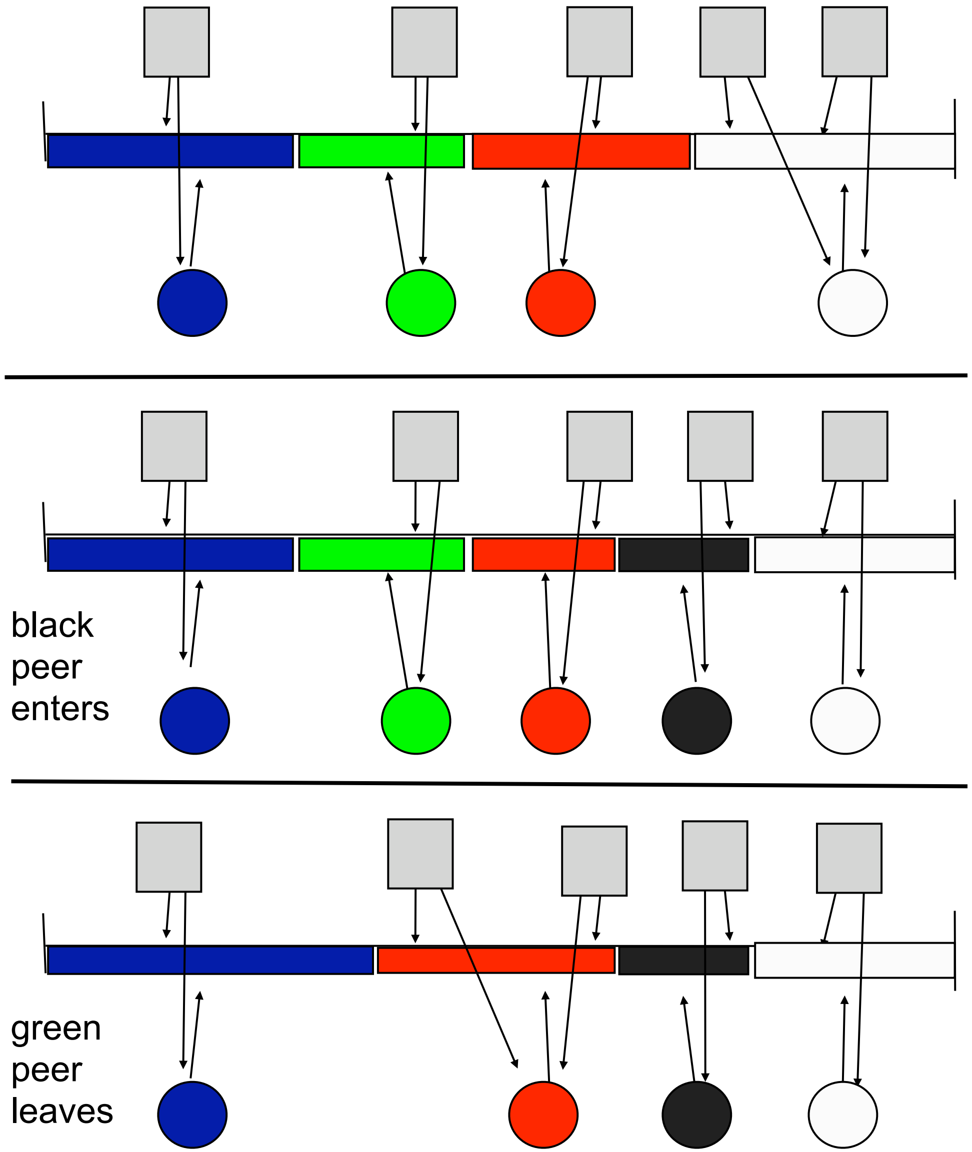


DHT

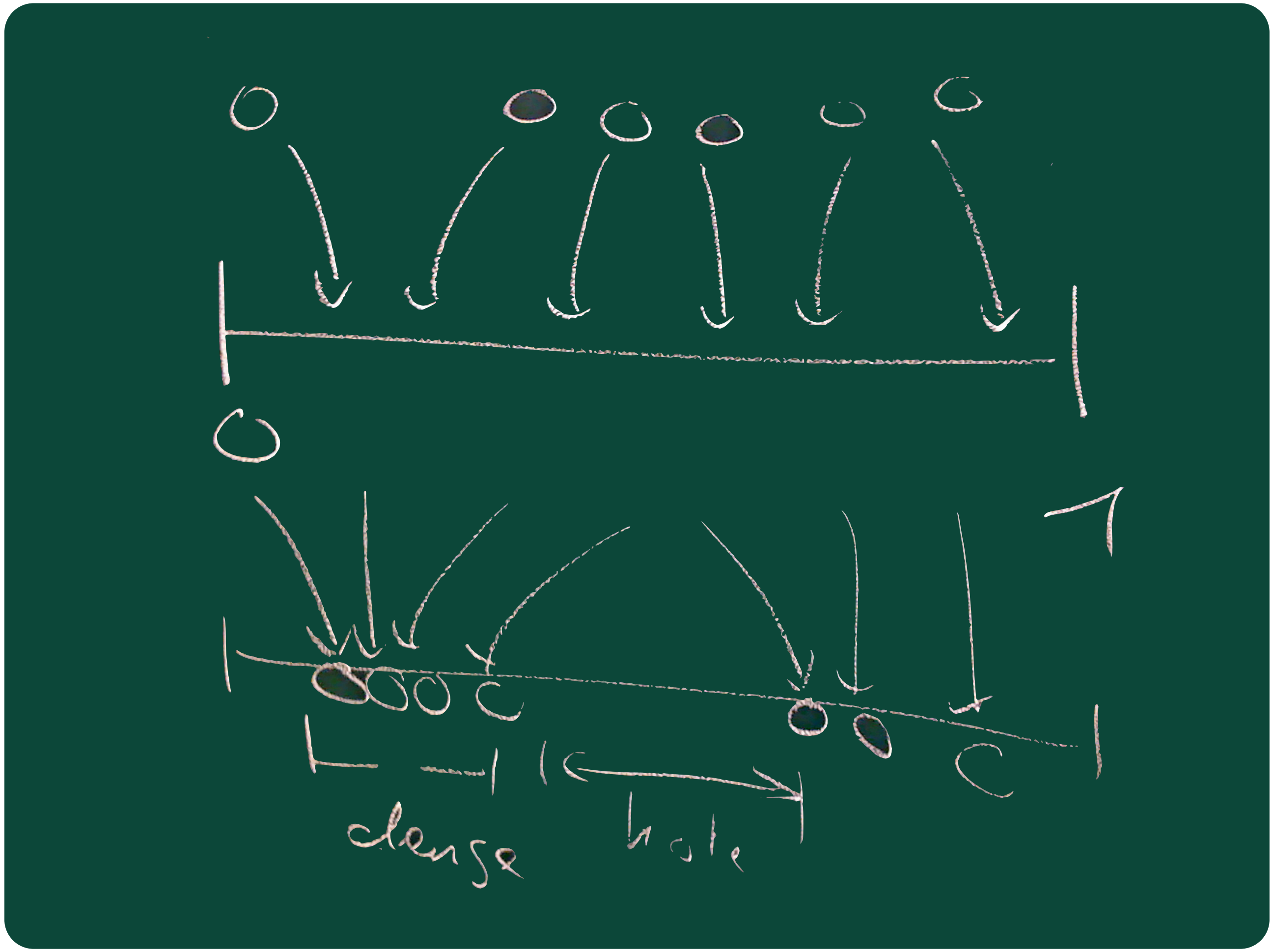


Entering and Leaving a DHT

- **Distributed Hash Table**
  - devices are hashed to to position
  - blocks are hashed according to the ID
- **When a device is added**
  - only blocks from neighbors have to be moved
- **When a device is deleted**
  - blocks are moved only to the neighbors



Holes and Dense Areas



Size of Holes

- **Theorem**
  - If n elements are randomly inserted into an array [0,1[then with constant probability there is a „hole“ of size Ω(log n/n), i.e. an interval without elements.
- **Proof**
  - Consider an interval of size log n / (4n)
  - The chance not to hit such an interval is (1-log n/(4n))
  - The chance that n elements do not hit this interval is

$$\left(1-\frac{\log n}{4n}\right)^n = \left(1-\frac{\log n}{4n}\right)^{\frac{4n}{\log n}\frac{\log n}{4}} \geq \left(\frac{1}{4}\right)^{\frac{1}{4}\log n} = \frac{1}{\sqrt{n}}$$

  - The expected number of such intervals is more than 1.
  - Hence the probability for such an interval is at least constant.

Proof of Dense Areas

$$\left(\frac{1}{4}\right)^{\frac{1}{4}\cdot\log n} = 2^{\left(\frac{1}{4}\log n\right)\overbrace{\log_2 \frac{1}{4}}^{-2}}$$

$$= 2^{\left(-\frac{1}{2}\right)\cdot\log n}$$

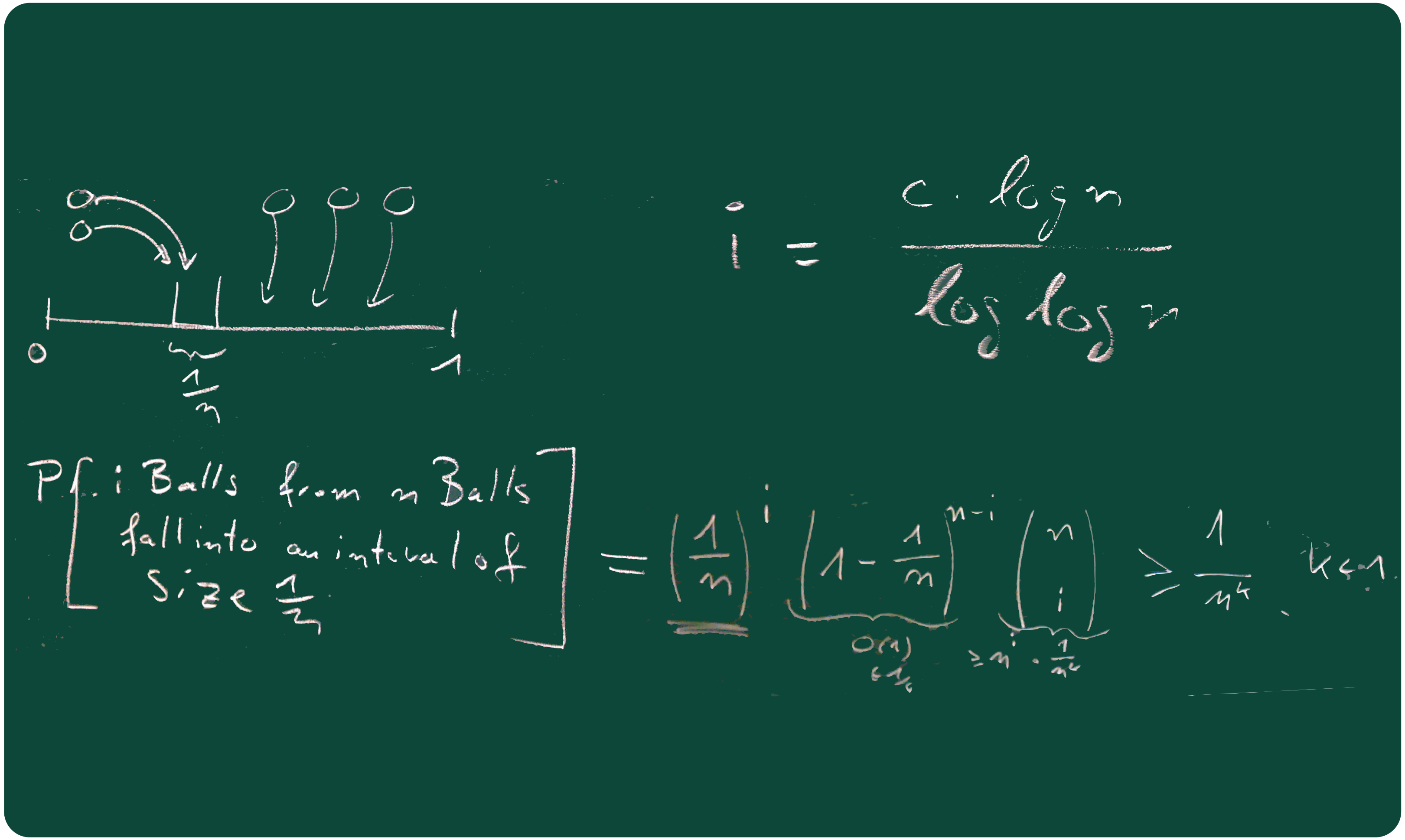
$$= n^{-\frac{1}{2}} = \frac{1}{\sqrt{n}}$$

Expectation: $\frac{4n}{\log n}\cdot\frac{1}{\sqrt{n}} = \frac{4\sqrt{n}}{\log n}$

Dense Spots

- **Theorem**
  - If n elements are randomly inserted into an array [0,1[then with constant probability there is a dense interval of length 1/n with at least Ω(log n/ (log log n)) elements.
- **Proof**
  - The probability to place exactly i elements in to such an interval is $$\left(\frac{1}{n}\right)^i \left(1-\frac{1}{n}\right)^{n-i} \binom{n}{i}$$
  - for i = c log n / (log log n) this probability is at least $1/n^k$ for an appropriately chosen c and k<1
  - Then the expected number of intervals is at least 1

Proof of Dense Areas



Proof of Dense Areas

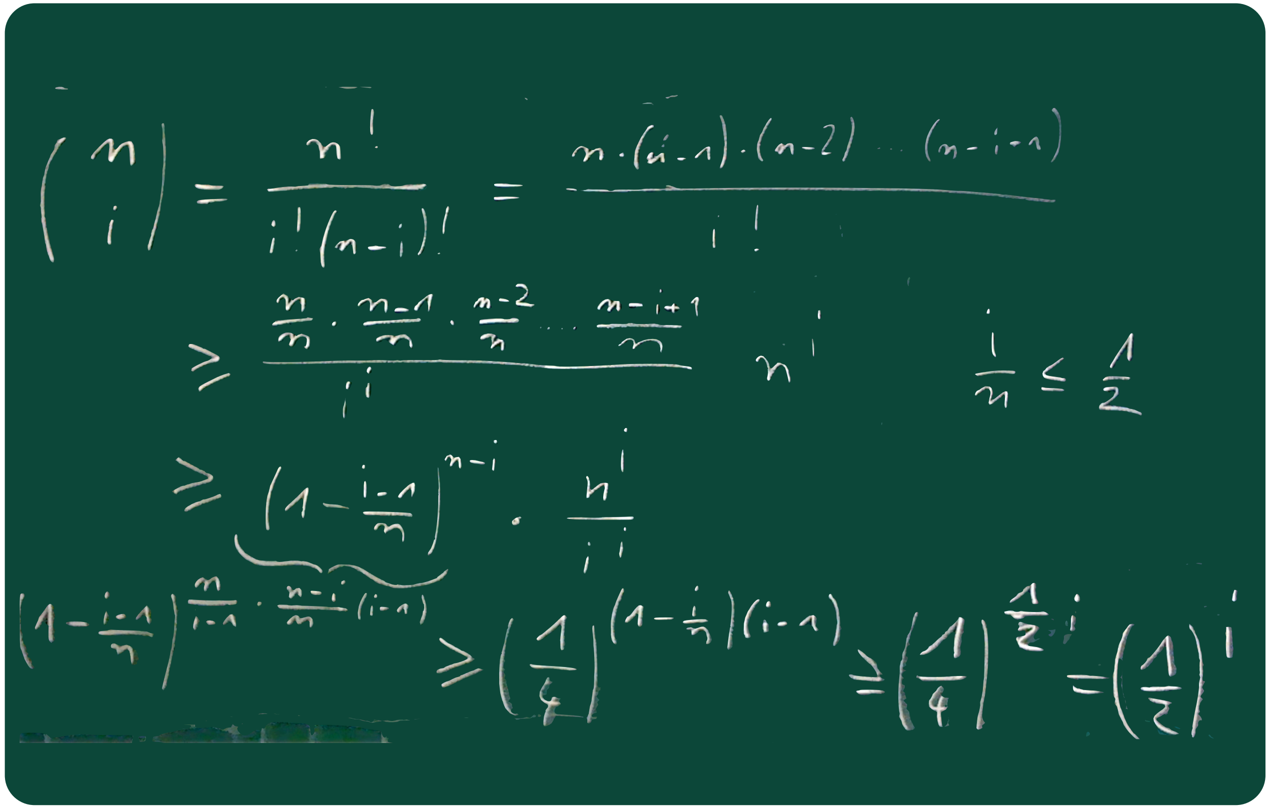
$$\frac{1}{4} \underset{m \geq 2}{\leq} \left(1 - \frac{1}{m}\right)^m \leq \frac{1}{e}$$

$$\left(1 - \frac{1}{n}\right)^{n-i} = \left(1 - \frac{1}{n}\right)^{n \frac{n-i}{n}}$$

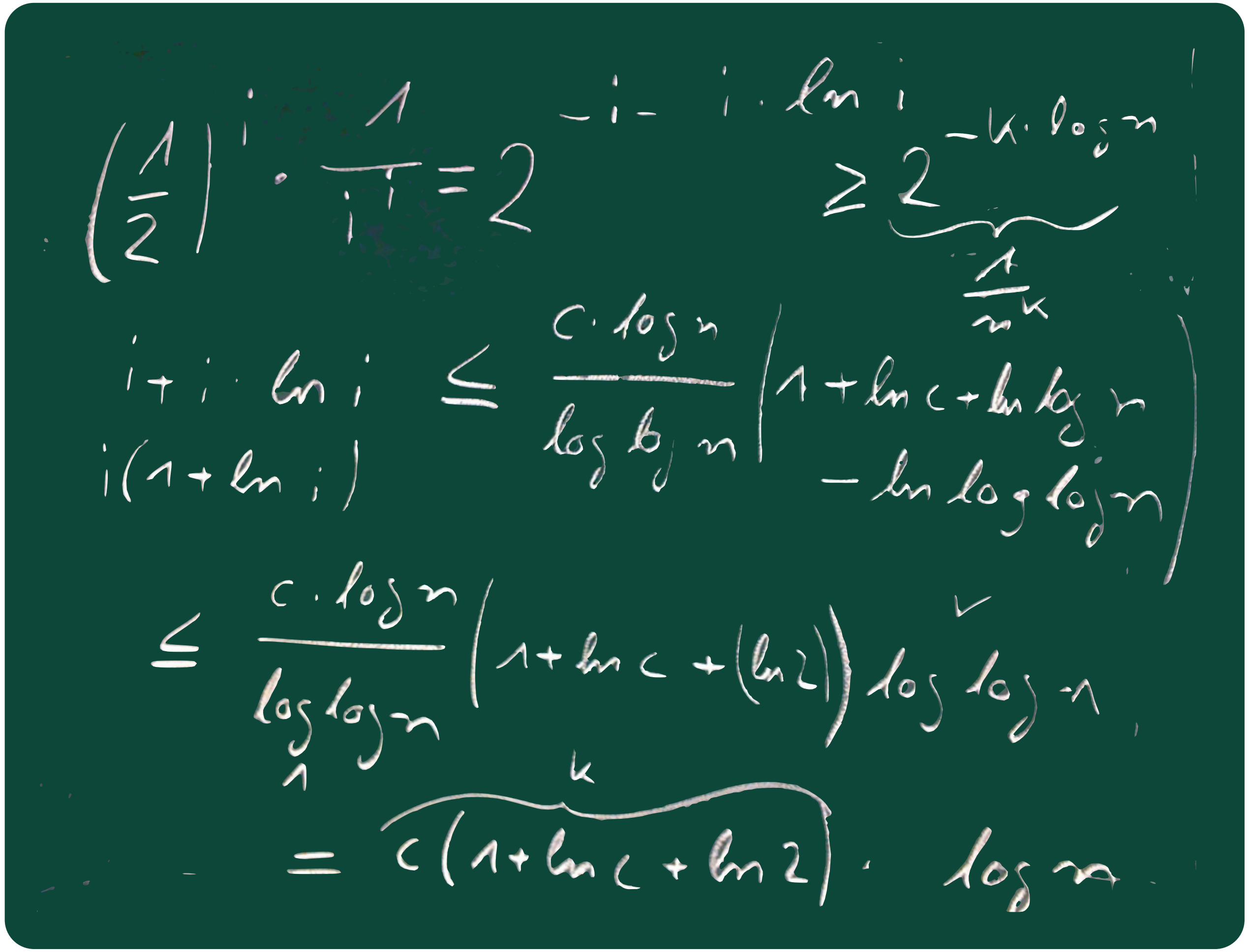
$$\geq \left(\frac{1}{4}\right)^{1 - \frac{i}{n}}$$

$$\geq \frac{1}{4}$$

Proof of Dense Areas



Proof of Dense Areas



Averaging Effect

- **Theorem**
  - If $\Theta(n \log n)$ elements are randomly inserted into an array [0,1[then with high probability in every interval of length 1/n there are $\Theta(\log n)$ elements.

Chernoff-Bound

- **Theorem Chernoff Bound**
  - Let $x_1,\ldots,x_n$ independent Bernoulli experiments with
    - $P[x_i = 1] = p$
    - $P[x_i = 0] = 1-p$
  - Let $$S_n = \sum_{i=1}^{n} x_i$$
  - Then for all c>0
$$\mathbf{P}[S_n \geq (1+c) \cdot \mathbf{E}[S_n]] \quad \leq \quad e^{-\frac{1}{3}\min\{c,c^2\}pn}$$
  - For $0 \leq c \leq 1$
$$\mathbf{P}[S_n \leq (1-c) \cdot \mathbf{E}[S_n]] \quad \leq \quad e^{-\frac{1}{2}c^2 pn}$$

Balls and Bins

Lemma

If m= k n ln n Balls are randomly placed in n bins:

1. Then for all c>k the probability that more than c ln n balls are in a bin is at most $O(n^{-c'})$ for a constant c'>0.
2. Then for all c<k the probability that less than c ln n balls are in a bin is at most $O(n^{-c'})$ for a constant c'>0.

Proof:

Consider a bin and the Bernoulli experiment B(k n ln n,1/n) and expectation: μ = m/n = k ln n

1. Case: c>2k
$$P[X \geq c \ln n] = P[X \geq (1+(c/k-1))k \ln n] \leq e^{-\frac{1}{3}(c/k-1)k \ln n} \leq n^{-\frac{1}{3}(c-k)}$$

2. Case: k<c<2k
$$P[X \geq c \ln n] = P[X \geq (1+(c/k-1))k \ln n] \leq e^{-\frac{1}{3}(c/k-1)^2 k \ln n} \leq n^{-\frac{1}{3}(c-k)^2/}$$

3. Case: c<k
$$P[X \leq c \ln n] = P[X \leq (1-(1-c/k))k \ln n] \leq e^{-\frac{1}{2}(1-c/k)^2 k \ln n} \leq n^{-\frac{1}{2}(k-c)^2/k}$$

Concept of High Probability

Lemma

If A(i) holds with **high** probability, i.e. $1-n^{-c}$, then

(A(1) and A(2) and ... and A(n)) with **high** probability,
i.e. $1-n^{-(c-1)}$

Proof:

- For all i: $P[\neg A(i)] \leq n^{-c}$
- Hence: $P[\neg A(1) \text{ or } \neg A(2) \text{ or } \ldots \neg A(n)] \leq n \cdot n^{-c}$

$$P[\neg(\neg A(1) \text{ or } \neg A(2) \text{ or } \ldots \neg A(n))] \leq 1- n \cdot n^{-c}$$

DeMorgan:

$$P[A(1) \text{ and } A(2) \text{ and } \ldots A(n)] \leq 1- n \cdot n^{-c}$$

Principle of Multiple Choice

- **Before inserted check c log n positions**
- **For position p(j) check the distance a(j) between potential left and right neighbor**
- **Insert element at position p(j) in the middle between left and right neighbor, where a(j) was the maximum choice**
- **Lemma**
  - After inserting n elements with high probability only intervals of size 1/(2n), 1/n und 2/n occur.

Proof of Lemma

1. Part: With high probability there is no interval of size larger than 2/n

follows from this Lemma

Lemma*

Let c/n be the largest interval. After inserting 2n/c peers all intervals are smaller than c/(2n) with high probability

From applying this lemma for c=n/2,n/4, ...,4 the first lemma follows.

Proof

- **2nd part: No intervals smaller than 1/(2n) occur**
  - The overall length of intervals of size 1/(2n) before inserting is at most 1/2
  - Such an area is hit with probability at most 1/2
  - The probability to hit this area more than c log n times is at least

$$2^{-c \log n} = n^{-c}$$

  - Then for c>1 such an interval will not further be divided with probability into an interval of size 1/(4m).

ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG

Algorithms and Methods for Distributed Storage Networks

9 Analysis of DHT

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik
Wintersemester 2007/08

CoNe Freiburg

IIF
INSTITUT FÜR INFORMATIK FREIBURG

ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Algorithms and Methods for Distributed Storage Networks

10 Distributed Heterogeneous Hash Tables

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik
Wintersemester 2007/08

CoNe Freiburg

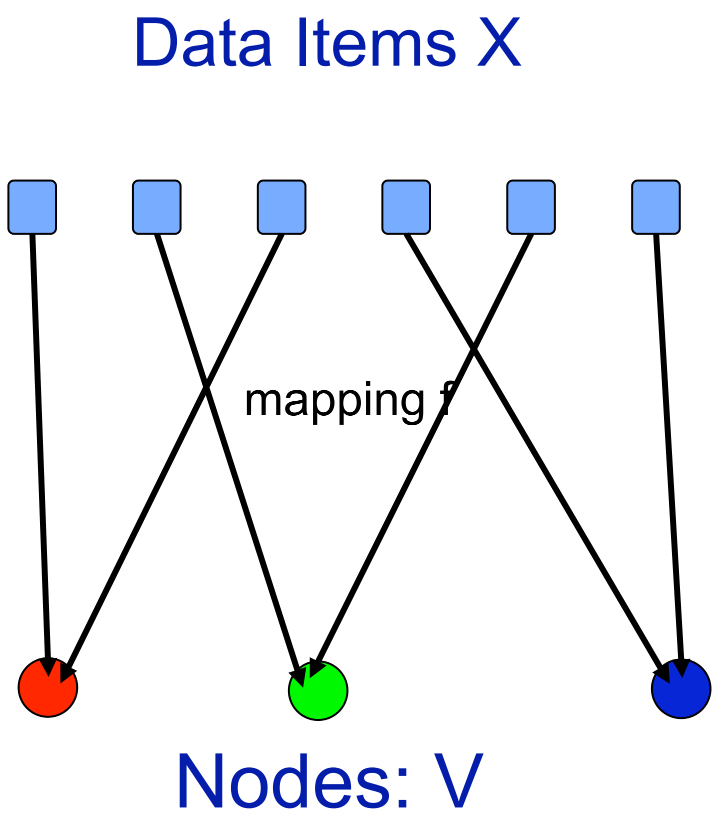
IIF
INSTITUT FÜR INFORMATIK FREIBURG

Literature

- André Brinkmann, Kay Salzwedel, Christian Scheideler, Compact, Adaptive Placement Schemes for Non-Uniform Capacities, 14th ACM Symposium on Parallelism in Algorithms and Architectures 2002 (SPAA 2002)
- Christian Schindelhauer, Gunnar Schomaker, Weighted Distributed Hash Tables, 17th ACM Symposium on Parallelism in Algorithms and Architectures 2005 (SPAA 2005)
- Christian Schindelhauer, Gunnar Schomaker, SAN Optimal Multi Parameter Access Scheme, ICN 2006, International Conference on Networking, Mauritius, April 23-26, 2006

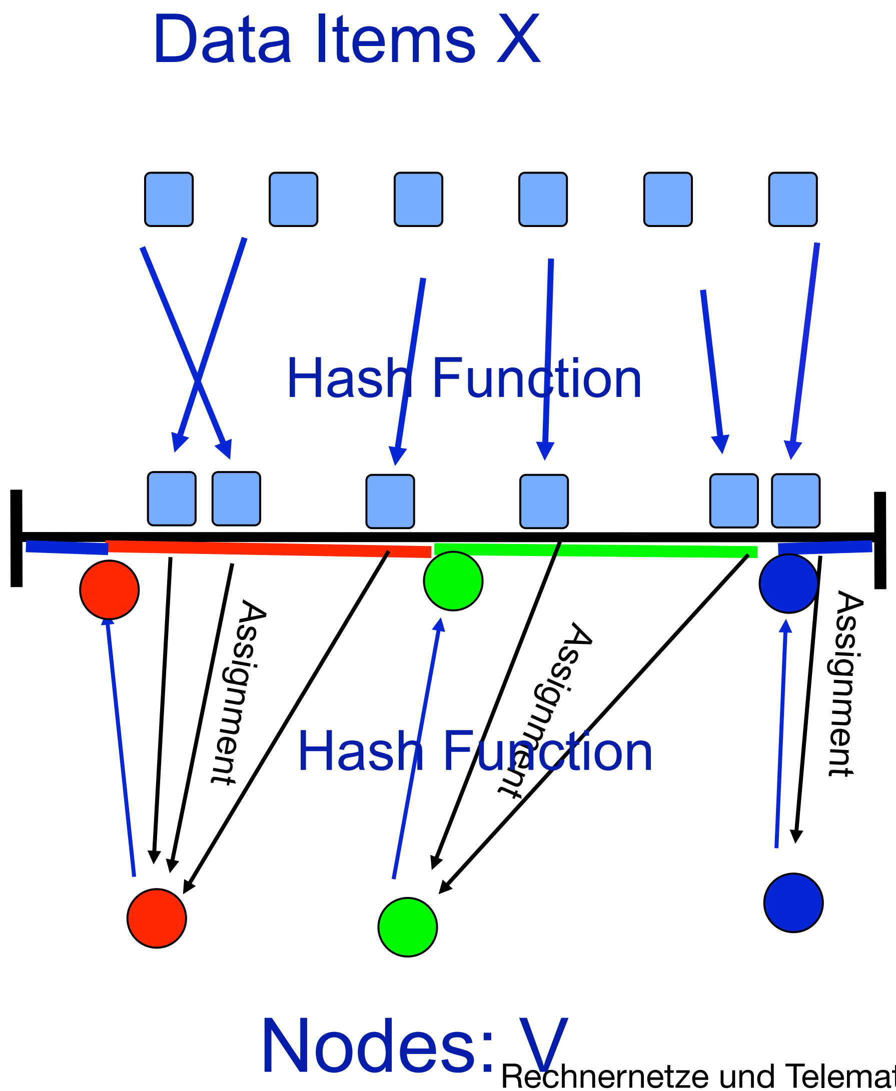
The Uniform Problem

- **Given**
  - a dynamic set of n nodes $V = \{v_1, \ldots, v_n\}$
  - data elements $X = \{x_1, \ldots, x_m\}$
- **Find**
  - a mapping $f_V : X \rightarrow V$
- **With the following properties**
  - The mapping is simple
    - $f_V(x)$ be computed using V and x
    - without the knowledge of $X \setminus \{x\}$
  - Fairness:
    - $|f_V^{-1}(v)| \approx |f_V^{-1}(v)|$
  - Monotony: Let $V \subset W$
    - For all $v \in V$: $f_V^{-1}(v) \supseteq f_W^{-1}(v)$
- **where** $f_V^{-1}(v) := \{x \in X : f_V(x) = v\}$



Distributed Hash Tables
THE Solution for the Uniform case

- **"Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web",**
  - David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, Rina Panigrahy, STOC 1997
  - Present a simple solution
- **Distributed Hash Table**
  - Chooose a space M = [0,1[
  - Map nodes v to M via hash function
    - $h : V \rightarrow M$
  - Map documents and servers to an interval
    - $h : X \rightarrow M$
  - Assign a document to the server which minimizes the distance in the interval
  - $f_V(x) = \text{argmin}\{v \in V: (h(x)-h(v)) \text{mod } 1\}$
    - where $x \text{ mod } 1 := x - \lfloor x \rfloor$

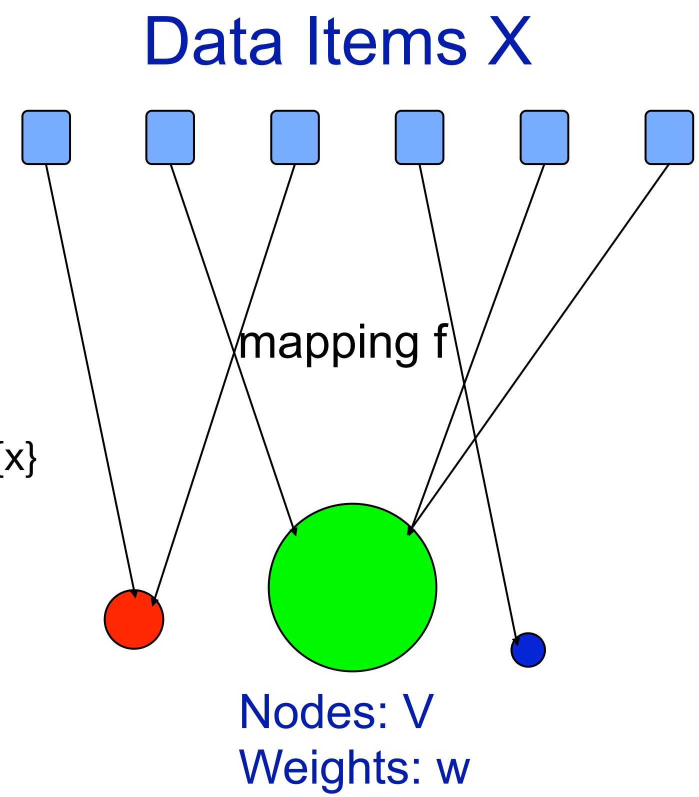


The Performance of Distributed Hash Tables

- **Theorem**
  - Data elements are mapped to node i with probability $p_i = 1/|V|$, if the hash functions behave like perfect random experiments
- **Balls into bins problem**
  - Expected ratio $\max(p_i)/\min(p_i) = \Omega(\log n)$
- **Solutions:**
  - Use O(log n) **copies** of a node
  - **Principle of multiple choices**
    - check at some O(log n) positions and choose the largest empty interval for placing a node,
  - **Cookoo-Hashing**
    - every node chooses among two possible position

The Heterogeneous Case

- **Given**
  - a dynamic set of n nodes $V = \{v_1, \dots, v_n\}$
  - dynamic weights $w : V \to R+$
  - dynamic set of data elements $X = \{x_1,\dots,x_m\}$
- **Find a mapping $f_{w,V} : X \to V$**
- **With the following properties**
  - The mapping is simple
    - $f_{w,V}(x)$ be computed using V, x, w without the knowledge of $X \setminus \{x\}$
  - Fairness: for all $u,v \in V$:
    - $| f_{w,V}^{-1}(u)|/w(u) \approx | f_{w,V}^{-1}(v)|/w(v)$
  - Consistency:
    - Let $V \subset W$: For all $v \in V$:
      - $f_{w,V}^{-1}(v) \supseteq f_{w,W}^{-1}(v)$
    - Let for all $v \in V\setminus\{u\}$: $w(v) = w'(v)$ and $w'(u)>w(u)$:
      - for all $v \in V\setminus\{u\}$: $f_{w,V}^{-1}(v) \supseteq f_{w',V}^{-1}(v)$ and $f_{w,V}^{-1}(u) \subseteq f_{w',V}^{-1}(u)$
- **where $f_{w,V}^{-1}(v) := \{ x \in X : f_{w,V}(x) = v \}$**



Some Application Areas

- **Proxy Caching**
  - Relieving hot spots in the Internet
- **Mobile Ad Hoc Networks**
  - Relating ID and routing information
- **Peer-to-Peer Networks**
  - Finding the index data efficiently
- **Storage Area Networks**
  - Distributing the data on a set of servers

Application
Peer-to-Peer Networks

- **Peer-to-Peer Network:**
  - decentralized overlay network delivering services over the Internet
  - no client-server structure
    - example: Gnutella
- **Problem: Lookup in first generation networks very slow**
- **Solution:**
  - Use an efficient data structure for the links and
  - map the keys to a hash space
- **Examples:**
  - **CAN**
    - maps keys to a d-dimensional array
    - builds a toroidal connection network,
      - where each peer is assigned to rectangular areas
  - **Chord**
    - maps keys and peers to a ring via **DHT**
    - establishes binary search like pointers on the ring

Application
Storage Area Networks (SAN)

- **Distribute data over a set of hard disks (like RAID)**
  - Nodes = hard disks
  - Data items = blocks
- **Problem**
  - Place copies of blocks for redundancy
  - If a hard disk fails other hard disk carry the information
  - Add or remove hard disks without unnecessary data movement
  - Hard disks may have different sizes

SAN Architecture

- **Avoid server based architectures**
  - Assignment of data is not flexible enough
  - High local storage concentration (for LAN traffic reduction)
  - Low availability of free capacity
- **Basic SAN concept**
  - Combine all available disks into a single virtual one
  - Server independent existence of storage

Challenges in SAN

- **Heterogeneity**
  - hard disks typically differ in capacity and speed
- **Popularity**
  - some data is popular and other not (e.g. movies, music :-)
  - their popularity rank varies over time
- **Consistency**
  - system changes by adding or re-placing/moving
  - preserving a fair share rate
  - only necessary data replacements must be done
- **Availability**
  - hard disks may fail, but data should not!
- **Performance**

Traditional Virtualization in SAN
waterproof definitions


Standalone


Cluster


Hot swap


RAID 0

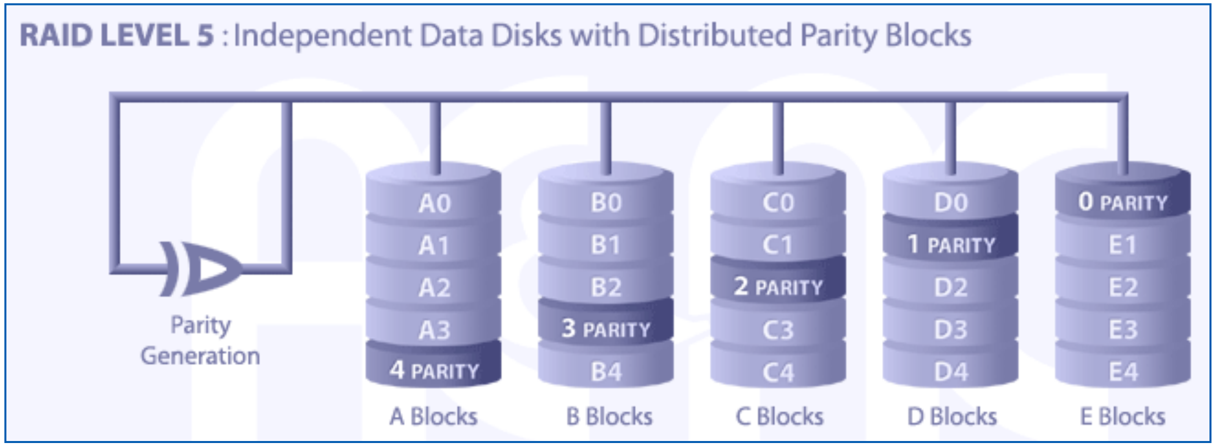

RAID 1


RAID 5


RAID 0+1

Deterministic Uniform SAN Strategies

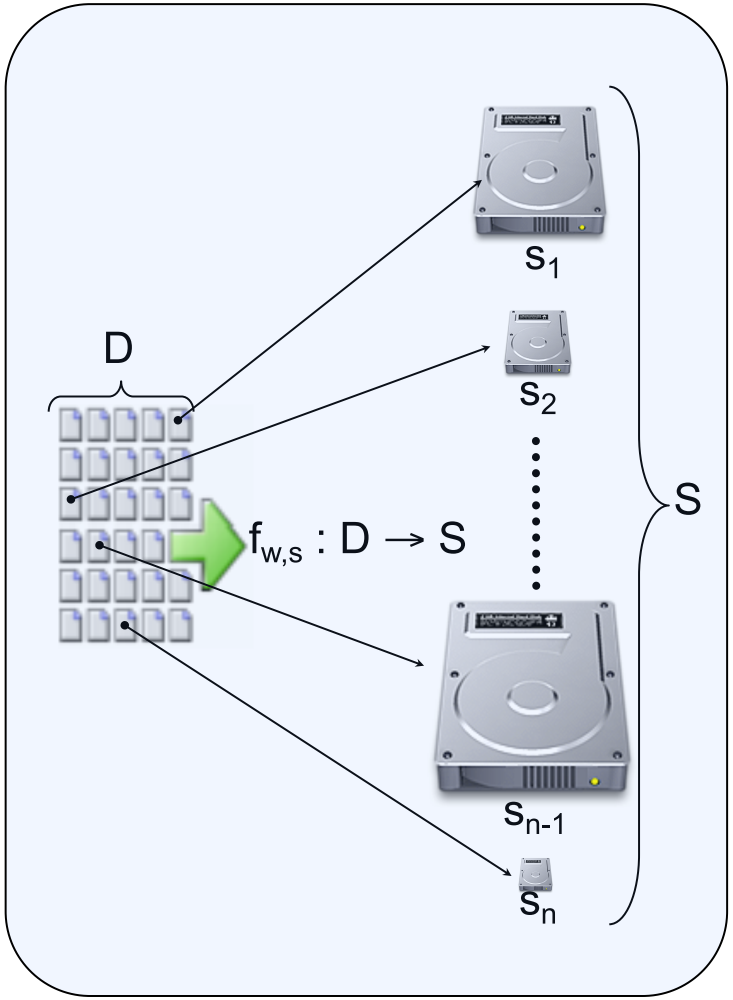
- **DRAID**
  - distributed Cluster Network for uniform storage nodes
  - uses RAID: striping/mirroring und Reed-Solomon encoding
  - organized in matrix rows => scalability only in groups of columns size



- **Good old stuff**
  - RAID 0, I, IV, V, VI (striping, mirroring, XOR, distributed XOR, XOR + Reed-Solomon)
- **Problems:**
  - scalability and availability is hard to combine
  - Re-Striping (time is money), huge offset tables (lookup is expansive),
  - storage concatenation without load balancing (disks are remaining full)
  - Only storage nodes with uniform capacities are allowed

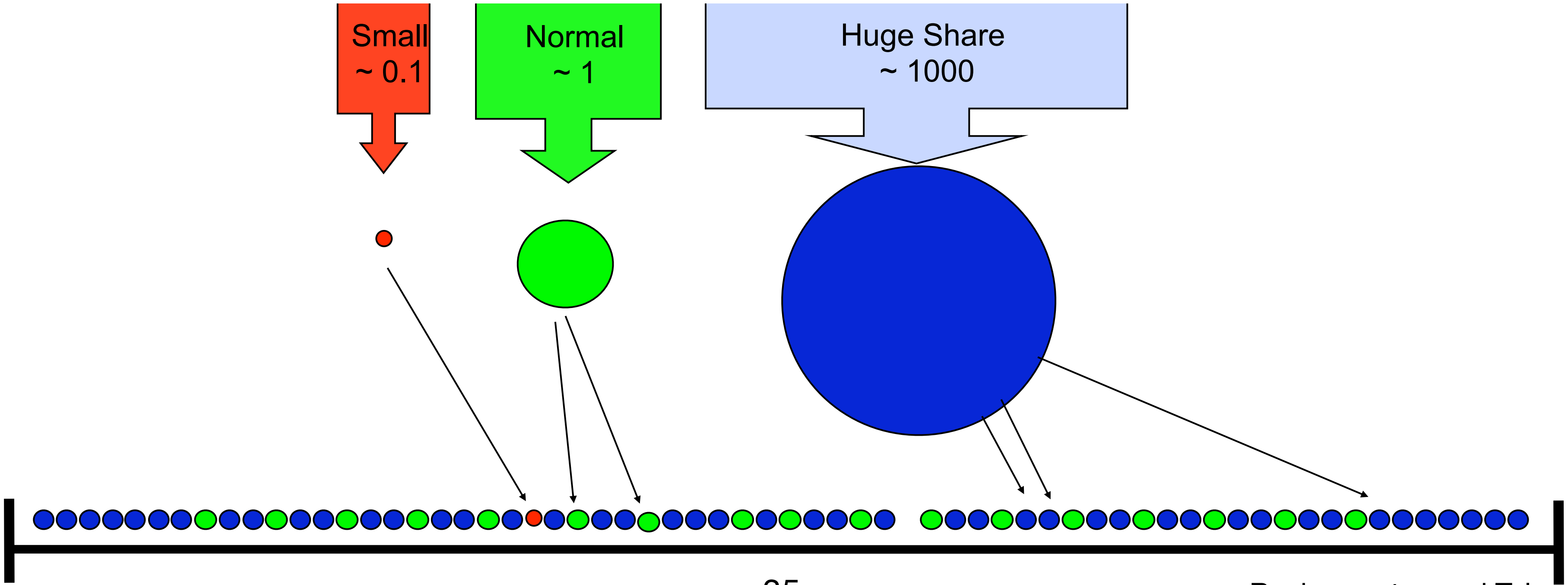
The Heterogeneous Case

- **Given**
  - a dynamic set of n nodes $V = \{v_1, \ldots, v_n\}$
  - dynamic weights $w : V \to \mathbf{R}^+$
  - dynamic set of data elements $X = \{x_1,\ldots,x_m\}$
- **Find a mapping $f_{w,V} : X \to V$**
- **With the following properties**
  - The mapping is **simple**
    - $f_{w,V}(x)$ be computed using V, x, w
    - without the knowledge of $X \backslash \{x\}$
  - **Fairness**: for all $u,v \in V$:
    - $| f_{w,V}^{-1}(u)|/w(u) \approx | f_{w,V}^{-1}(v)|/w(v)$
  - **Consistency**:
    - minimal replacements to preserve the data distribution
- **where $f_{w,V}^{-1}(v) := \{ x \in X : f_{w,V}(x) = v \}$**



The Naive Approach to DHT

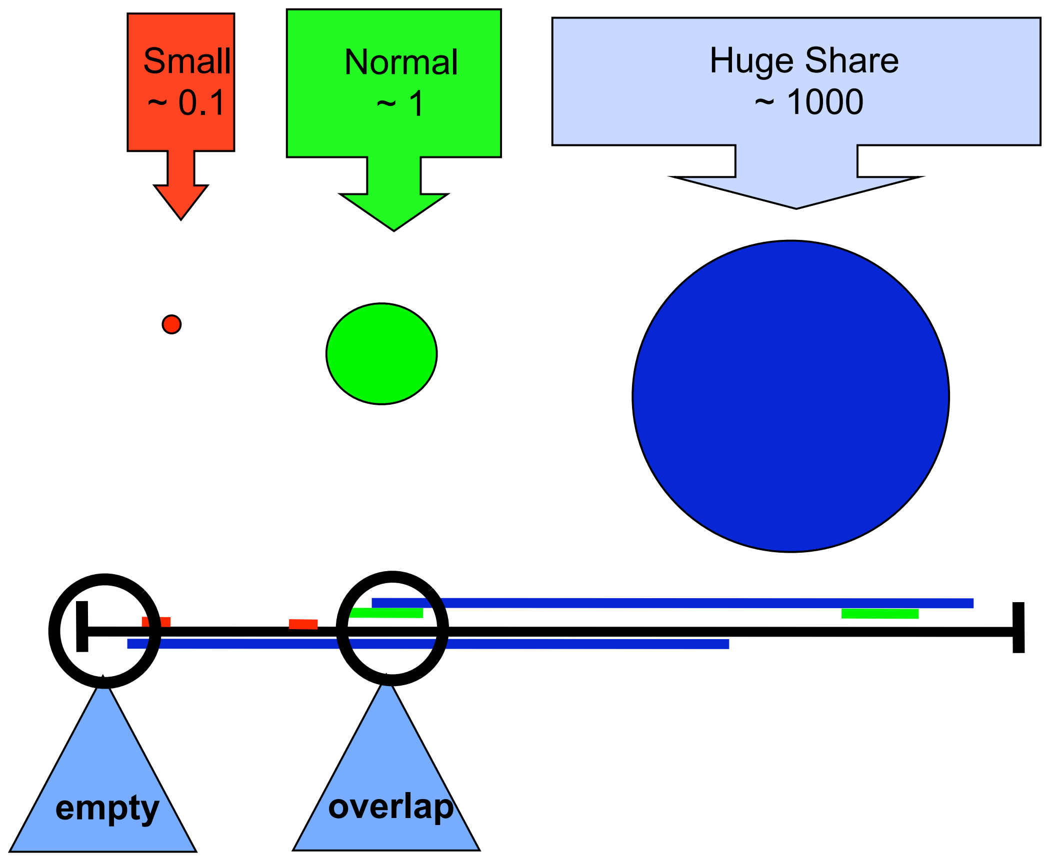
- Use $\left\lceil \frac{w_i}{\min_{j \in V}\{w_j\}} \right\rceil$ copies for each node w_i
- This is not feasible, if $\max_{j \in V}\{w_j\} / \min_{j \in V}\{w_j\}$ is too large
- Furthermore, inserting nodes with small weights increases the number of copies of all nodes.



Rechnernetze und Telematik
Albert-Ludwigs-Universität Freiburg
Christian Schindelhauer

SIEVE: Interval based consistent hashing

- **Interval based approach**
  - Brinkmann, Salzwedel, and Scheideler, SPAA 2000
- **Map nodes to random intervals (via hash function)**
  - interval length proportional to weight
- **Map data items to random positions (via hash function)**
- **Two problems**
  - What to do if intervals overlap?
  - What to do if the unions of intervals do not overlap the hash space M?

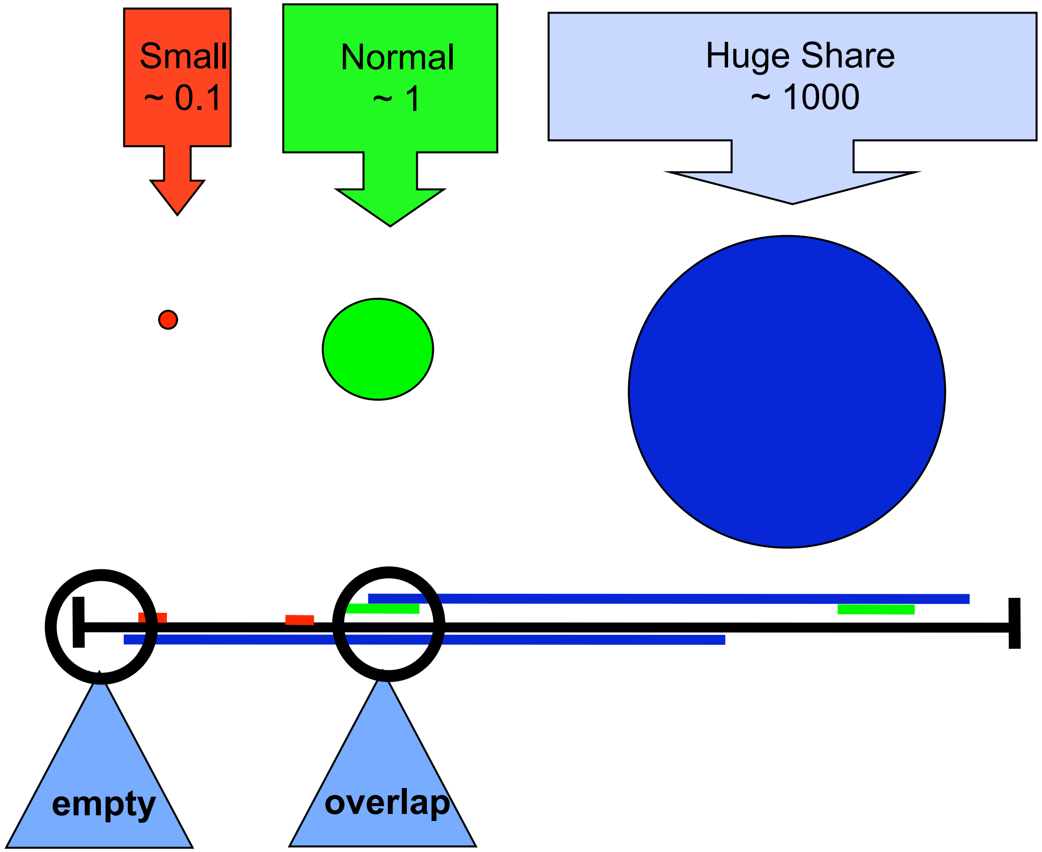


SIEVE: Interval based consistent hashing

1. **What to do if intervals overlap?**
   - Uniformly choose random candidate from the overlapping intervals
2. **What to do if the unions of intervals do not overlap the hash space M?**
   - Increase all intervals by a constant factor (stretch factor)
   - Use O(log n) copies of all nodes
     - resulting in O(n log n) intervals

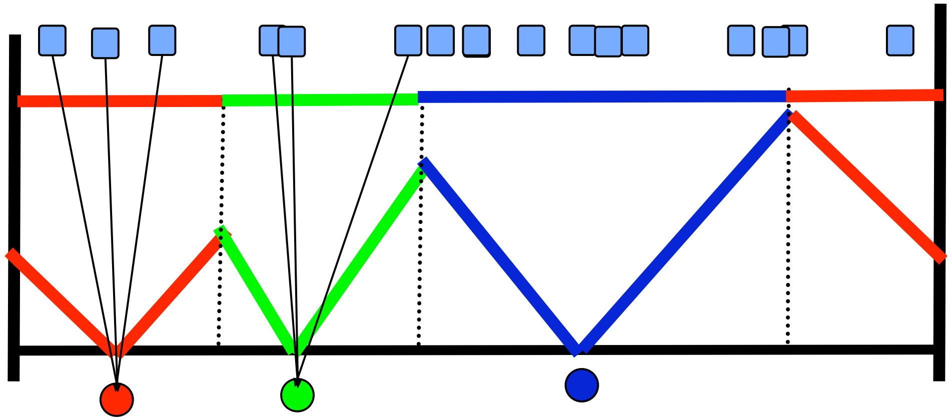
➢ **If more nodes appear**
   - then decrease all intervals by a constant factor

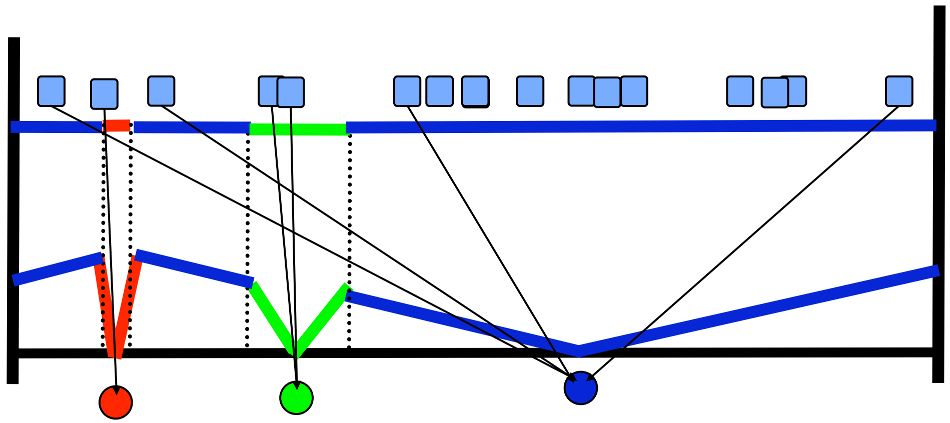
➢ **SIEVE is not providing monotony**
   - Re-stretching leads to unnecessary re-assignments



The Linear Method

- **Alternative presentation of (uniform) Consistent Hashing**
- **After "randomly" placing nodes into M**
  - Add cones pointing to the node's location in M
- **Compute for each data element x the height of the cones**
  - Choose the cone with smallest height
- **For the Linear Method**
  - Choose for each node i a cone stretched by the factor wi
- **Compute for each data element x the height of the cones**
  - Choose the cone with smallest height



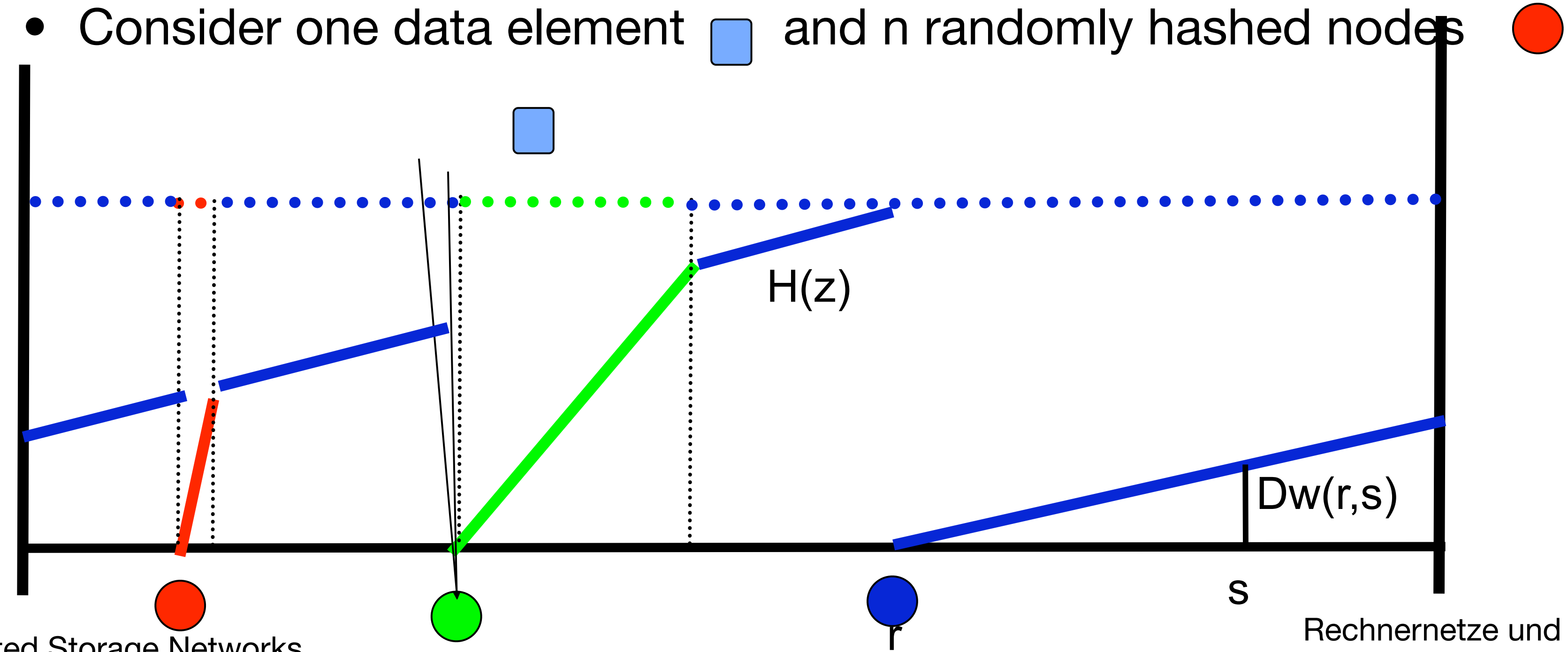


Literature

- André Brinkmann, Kay Salzwedel, Christian Scheideler, Compact, Adaptive Placement Schemes for Non-Uniform Capacities, 14th ACM Symposium on Parallelism in Algorithms and Architectures 2002 (SPAA 2002)
- Christian Schindelhauer, Gunnar Schomaker, Weighted Distributed Hash Tables, 17th ACM Symposium on Parallelism in Algorithms and Architectures 2005 (SPAA 2005)
- Christian Schindelhauer, Gunnar Schomaker, SAN Optimal Multi Parameter Access Scheme, ICN 2006, International Conference on Networking, Mauritius, April 23-26, 2006

The Linear Method: Basics

- **For easier description we use half-cones,**
  - the weighted distance is $D_w(r,s) := \frac{((s-r) \bmod 1)}{w}$
    - where x mod 1 := x - ⌊x⌋
- **Analyzing heights is easier as analyzing interval lengths!**
- **Define:** $H(z) := \min_{u \in V} D_{w_u}(z, s_u)$
  - Consider one data element and n randomly hashed nodes



The Linear Method: Basics

LEMMA 1. *Given n nodes with weights $w_1, \ldots, w_n$. Then the height $H(r)$ assigned to a position r in M is distributed as follows:*

$$P[H(r) > h] = \begin{cases} \prod_{i \in [n]} (1 - h w_i)\ , & \textit{if } h \leq \min_i \{\frac{1}{w_i}\} \\ 0\ , & \textit{else} \end{cases}$$

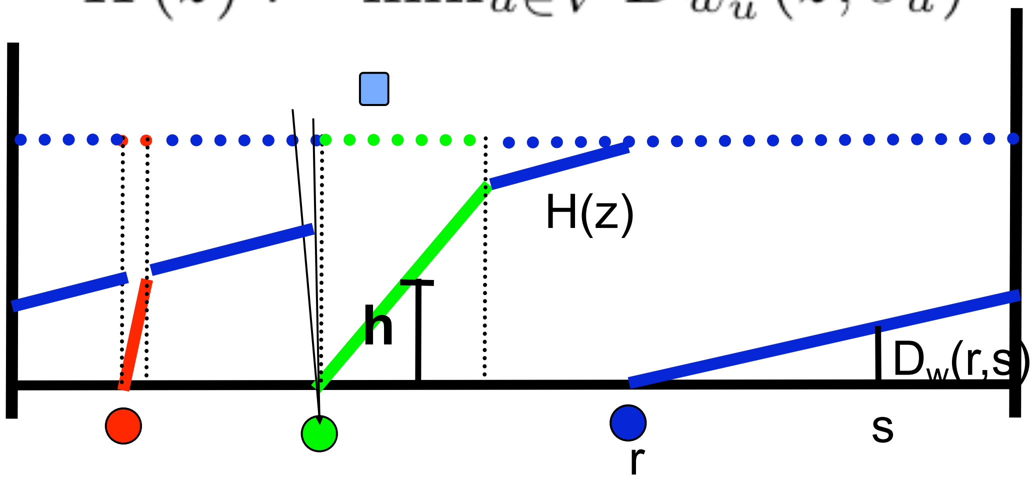
- **Proof:**
  - The probability of to receive height of at least h with respect to a node i is

    $1 - h\, w_i$

  - Since

$$\mathrm{P}[H_i \leq h] = \begin{cases} 1\ , & h \geq \frac{1}{w_i} \\ h \cdot w_i & \text{else.} \end{cases}$$

$$H(z) := \min_{u \in V} D_{w_u}(z, s_u)$$



∎

An Upper Bound for Fairness

THEOREM 1. *The Linear Method stores with probability of at most* $\frac{w_i}{W-w_i}$ *a data element at a node* i, *where* $W := \sum_{i=1}^{|V|} w_i$.

Proof:

From Lemma 1 follows

$$\mathrm{P}[H_i \in [h, h+\delta] \wedge \forall j \neq i : \; H_j > h] = \begin{cases} 0\,, & \exists j \; : \; h \geq \frac{1}{w_j} \\ \delta w_i \prod_{j \neq i}(1 - h w_j) & \text{else}\,. \end{cases}$$

We define $P_{i,h,\delta} := \delta w_i \prod_{j \neq i}(1 - h w_j)$

and the following term describes an upper bound

$$\sum_{m=1}^{\infty} P_{i,\delta m,\delta} \qquad \text{where} \qquad h = m\delta$$

An Upper Bound for Fairness (II)

THEOREM 1. *The Linear Method stores with probability of at most* $\frac{w_i}{W-w_i}$ *a data element at a node* i*, where* $W := \sum_{i=1}^{|V|} w_i$.

Proof (continued):

$$\begin{aligned} \lim_{\delta \to 0} \sum_{m=1}^{\infty} P_{i,\delta m,\delta} &\leq \lim_{\delta \to 0} \sum_{m=1}^{\infty} w_i \delta e^{-a\delta m} \\ &= \int_{x=0}^{\infty} w_i e^{-ax} dx = \frac{w_i}{a} \\ &= \frac{w_i}{\sum_{j \neq i} w_j} \end{aligned}$$

■

The Limits of the Linear Method

THEOREM 5. *The Linear Method (without copies) for n nodes with weights $w_1 = 1$ and $w_2, \ldots, w_{n-1} = \frac{1}{n-1}$ assigns a data element with probability $1 - e^{-1} \approx 0.632$ to node 0 when n tends to infinity.*

PROOF. We use Lemma 1 and reduce the probability to the following term.

$$\lim_{n \to \infty} \int_{x=0}^{1} x \left(1 - \frac{x}{n-1}\right)^{n-1} dx =$$

$$\int_{x=0}^{1} x e^{-x} dx = \left[-e^{-x}\right]_0^1 = 1 - e^{-1} .$$

Why does the biggest node win?

The small ones are competing against each other

The big one has no competitor in his league

The solution:

Use copies of each node

The Linear Method with Copies

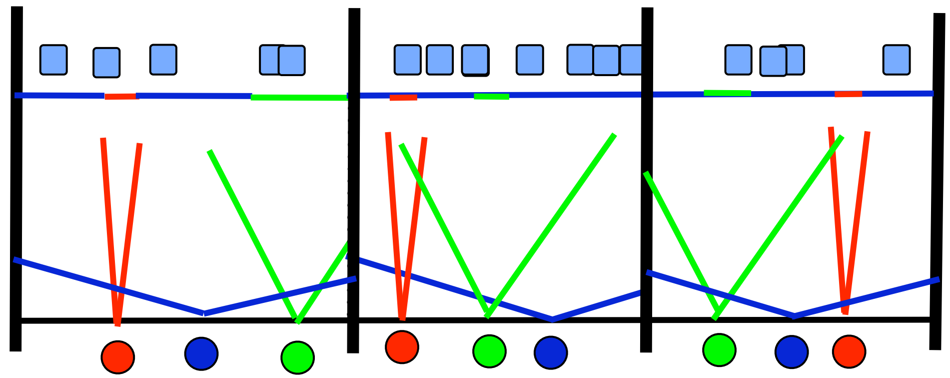
THEOREM 2. *Let $\epsilon > 0$. Then, the Linear Method using $\lceil \frac{2}{\epsilon} + 1 \rceil$ copies assigns one data element to node i with probability p_i where*

$$(1 - \sqrt{\epsilon}) \cdot \frac{w_i}{W} \le p_i \le (1 + \epsilon) \cdot \frac{w_i}{W} .$$

- **A constant number of copies suffice to “repair” the linear function**
- **This theorem works only for one data item**
  - If many data items are inserted, then the original bias towards some nodes is reproduced:
    - “Lucky” nodes receive more data items
- **Solution**
  - Independently repeat the game at least O(log n) times

Partitioning and the Linear Method

- **Partitions:**
  - Partition the hash range into sub-intervals
  - Map each data element into the whole interval
  - Map for each node 2/ε+1 copies into each sub-interval



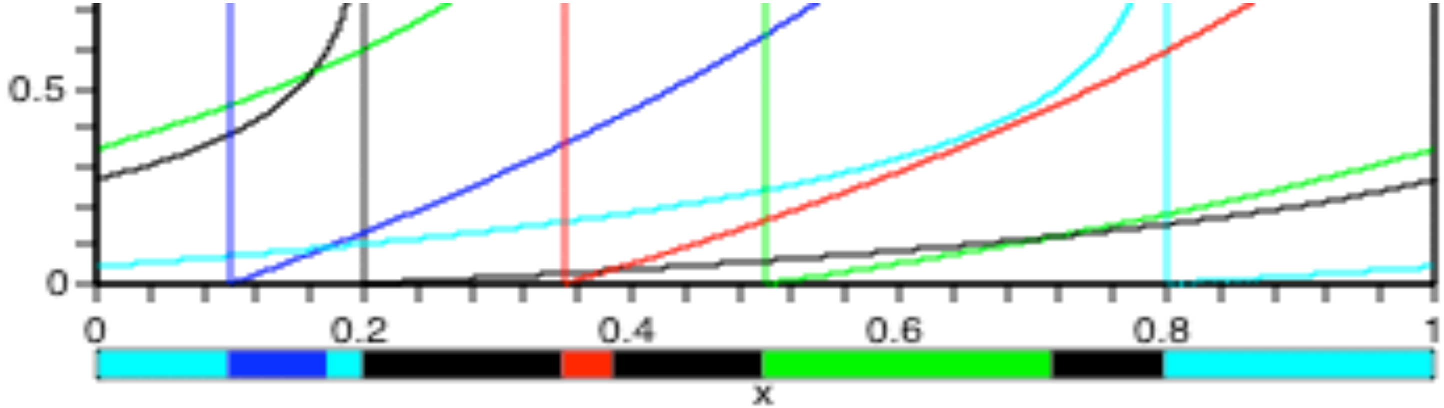
Theorem 3 *For all $\epsilon, \epsilon' > 0$ and $c > 0$ there exists $c' > 0$ such that when we apply the Linear Method to n nodes using $\lceil \frac{2}{\epsilon} + 1 \rceil$ copies and $c' \log n$ partitions, the following holds with high probability, i.e. $1 - n^{-c}$.*

Every node $i \in V$ receives all data elements with probability p_i such that

$$(1 - \sqrt{\epsilon} - \epsilon') \cdot \frac{w_i}{W} \leq p_i \leq (1 + \epsilon + \epsilon') \cdot \frac{w_i}{W} .$$

The Logarithmic Method

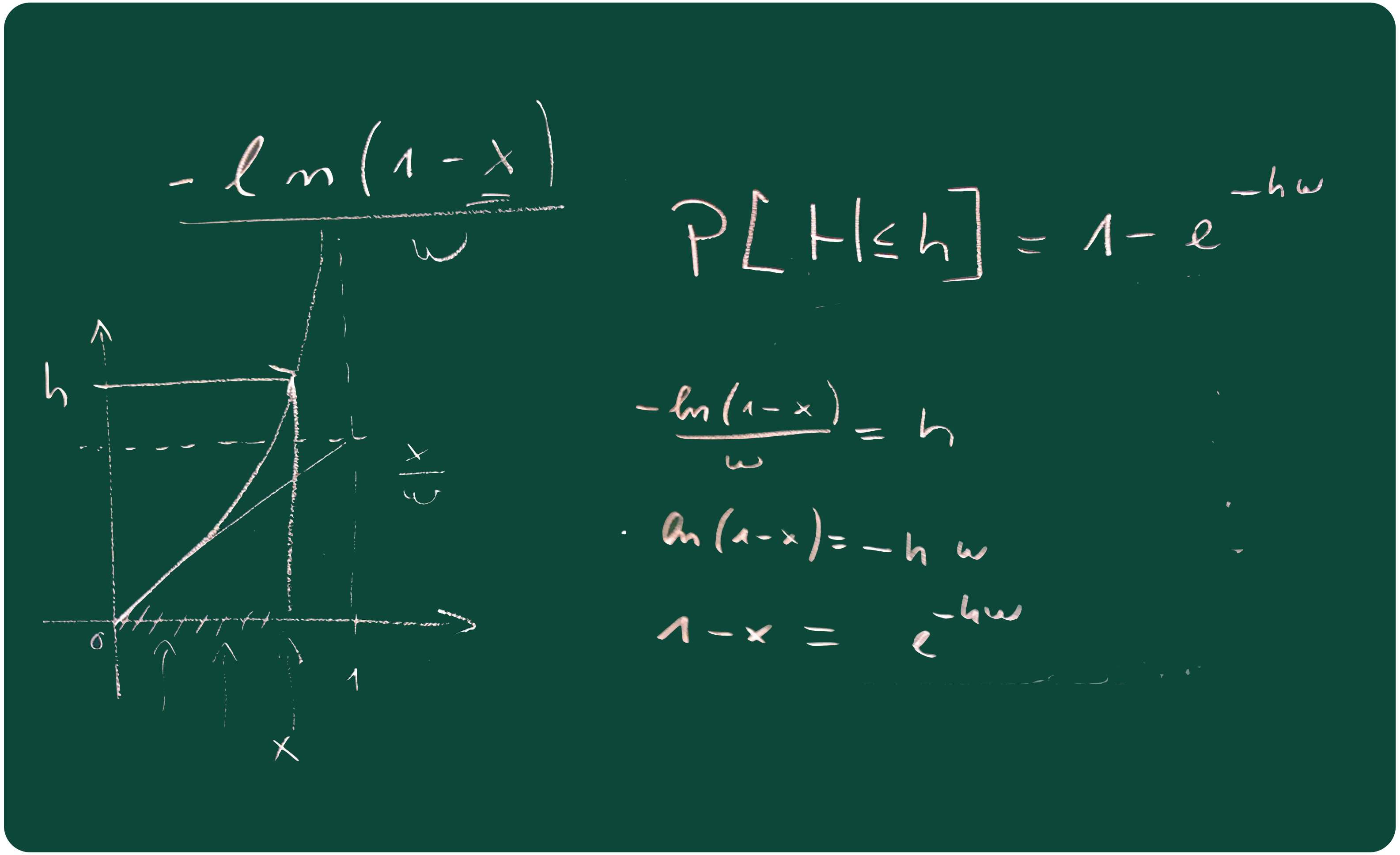
- **Replacing the linear function by** $L_w(r,s) := \frac{-\ln((1-(r-s)) \bmod 1)}{w}$
- **improves the accuracy**



FACT 2. *If in the Logarithmic Method (without copies and without partitions) a node arrives with weight w then the probability that data element x with previous height H_x is assigned to the new node is $1 - e^{-wH_x}$.*

THEOREM 6. Given n nodes with positive weights $w_1, \ldots, w_n$ the Logarithmic Method assigns a data element to node i with probability $\frac{w_i}{W}$, where $W := \sum_{i=1}^{|V|} w_i$.

Proof of Fact



Probability that a Height is in an Interval

$$P[H_i \geq h - \delta \wedge H_i < h]$$

$$= 1 - e^{-hw} - \left(1 - e^{-(h-\delta)\cdot w}\right)$$

$$= e^{-(h-\delta)w} - e^{-hw}$$

Proof of Theorem 2

Proof: Hence, the probability that a data element receives height in the interval [h–δ, h[and receives larger height than h for all other nodes is at most

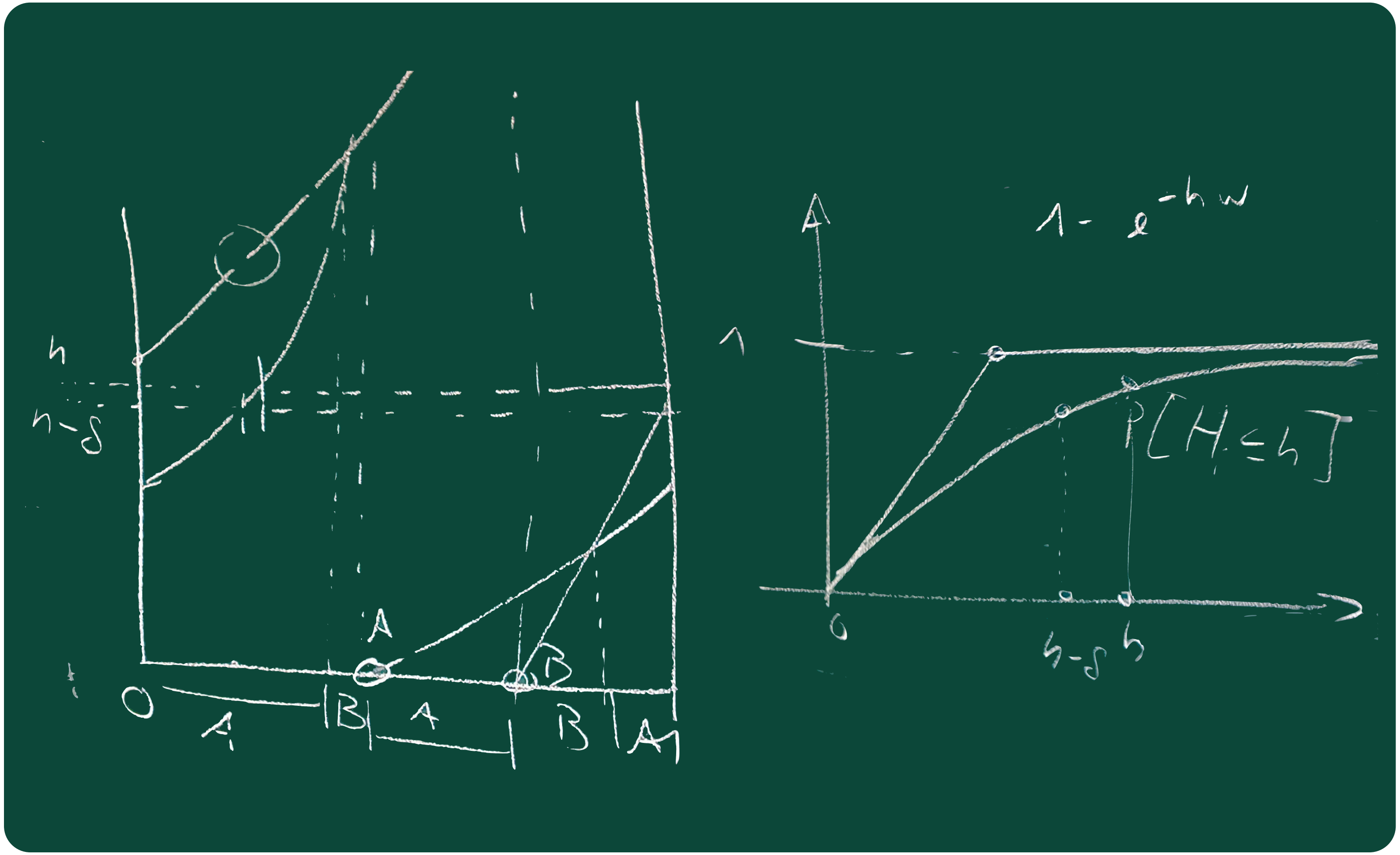
$$\mathbf{P}\left[H_i \geq h - \delta \;\wedge\; H_i < h \;\wedge\; \bigwedge_{j \neq i} H_j \geq h\right] =$$

$$\left(e^{-w_i(h-\delta)} - e^{-w_i h}\right) \prod_{j \neq i} e^{-w_j h} \quad =$$

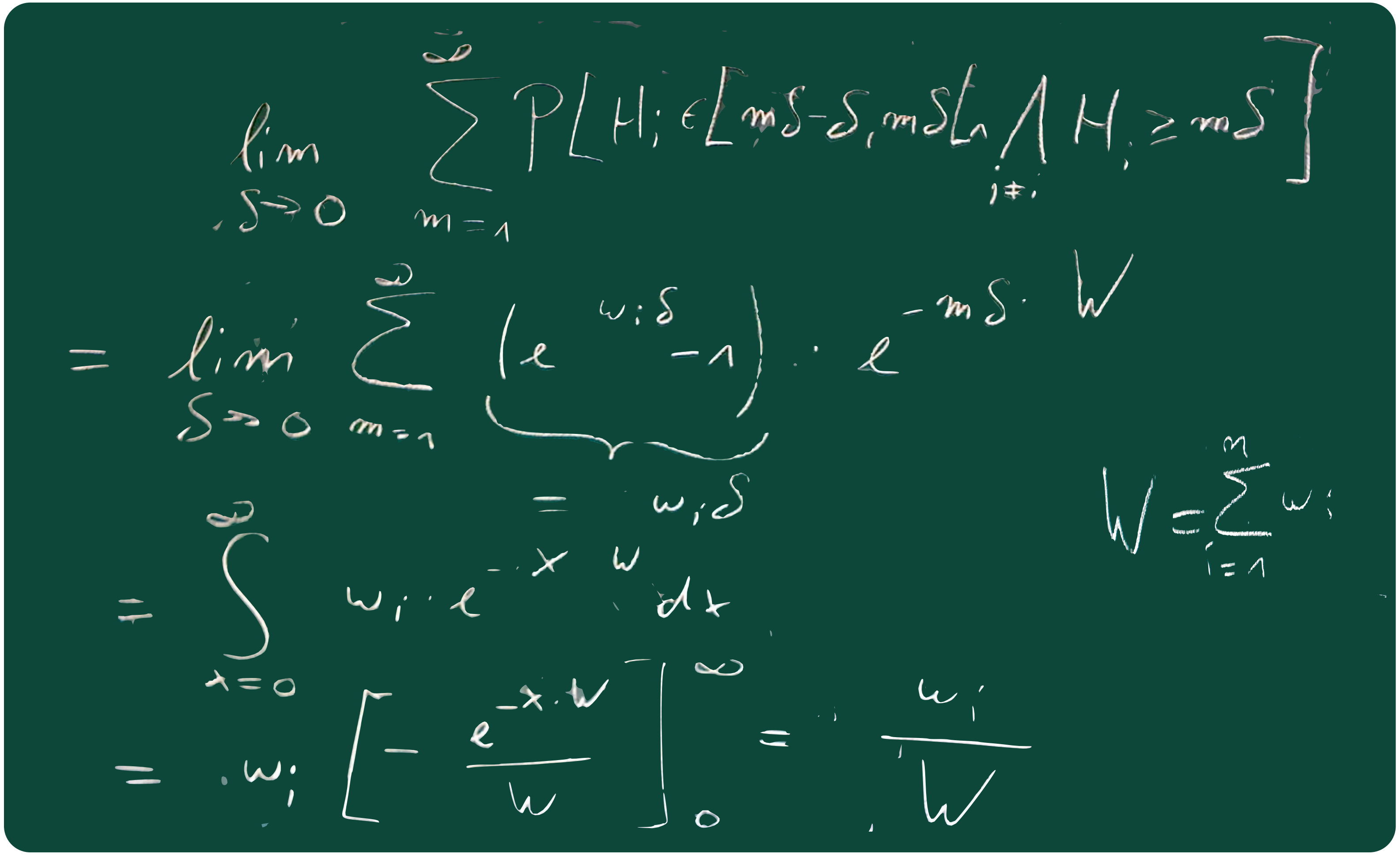
$$e^{-w_i h}\left(e^{w_i \delta} - 1\right) \prod_{j \neq i} e^{-w_j h} \quad =$$

$$\left(e^{w_i \delta} - 1\right) \prod_{j \in [n]} e^{-w_j h}$$

Proof of Theorem 2



Proof of Theorem 2



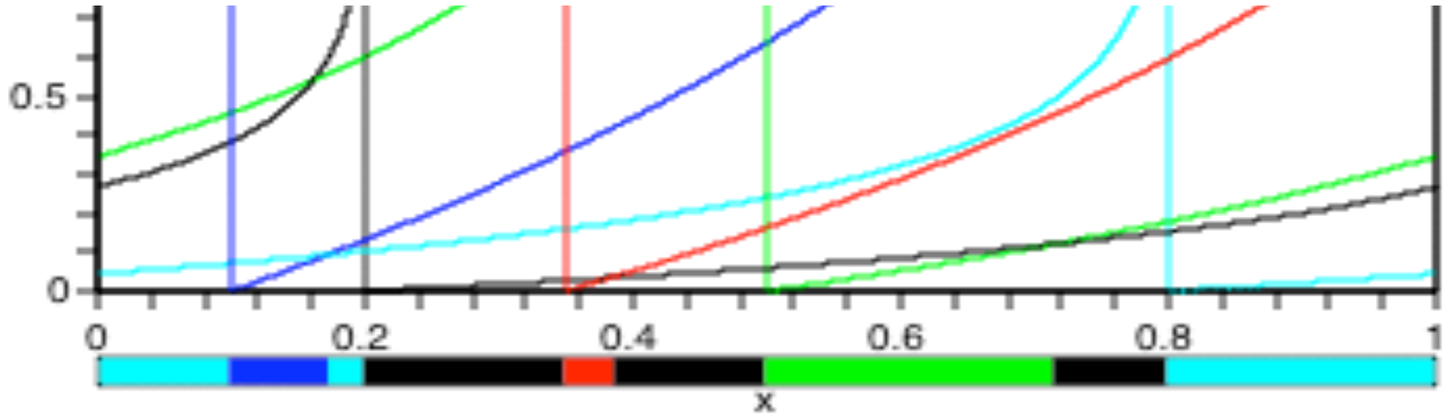
The Logarithmic Method

- **Replacing the linear function with $-\ln((1-d_i(x)) \bmod 1)/w_i$ improves the accuracy of the probability distribution**

Theorem 7 *For all $\epsilon > 0$ and $c > 0$ there exists $c' > 0$, where we apply the Logarithmic Method with $c' \log n$ partitions. Then, the following holds with high probability, i.e. $1 - n^{-c}$.*

Every node $i \in V$ receives data elements with probability p_i such that

$$(1-\epsilon) \cdot \frac{w_i}{W} \leq p_i \leq (1+\epsilon) \cdot \frac{w_i}{W} .$$



Rechnernetze und Telematik
Albert-Ludwigs-Universität Freiburg
Christian Schindelhauer

Further Features

- **Efficient data structure for the linear and logarithmic method**
  - can be implemented within O(n) space
  - Assigning elements can be done in O(log n) expected time
  - Inserting/deleting new nodes can be done in amortized time O(1)
- **Predicting Migration**
  - The height of a data element correlates with the probability that this data element is the next to migrate to a different server
- **Fading in and out**
  - Since the consistency works also for the weights:
  - Nodes can be inserted by slowly increasing the weight
  - No additional overhead
  - Node weight represents the transient download state
  - Vice versa for leaving nodes

Double Hashing

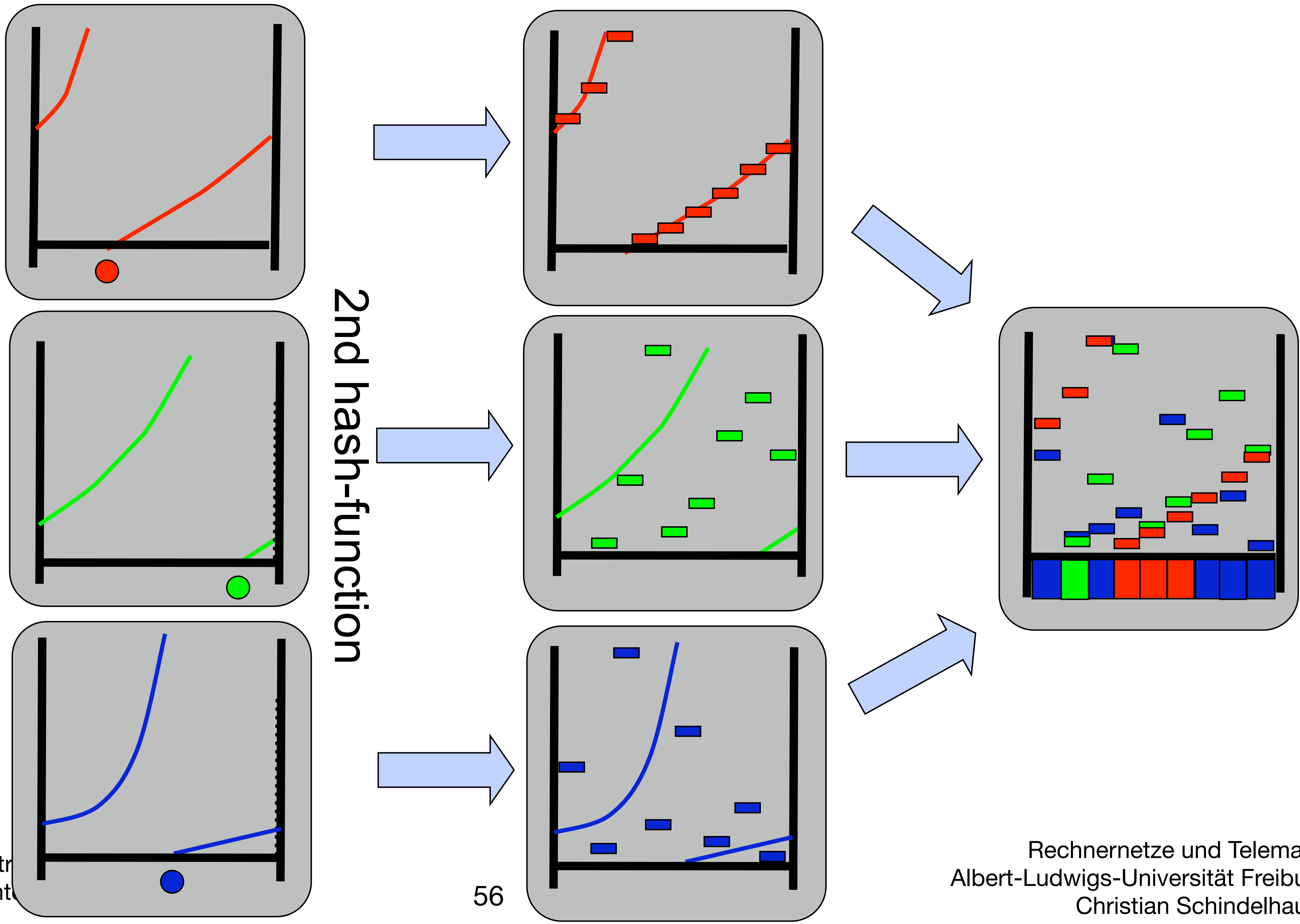
- **If every node uses a different hashing, then the logarithmic method can be chose without any copies**

For this, we apply for each node an individual hash function $h : V \times [0,1) \to [0,1)$. So, we start mapping the data element x to $r_x \in [0,1)$ as above and then for every node we compute $r_{i,x} = h(i, r_x)$. Now x is assigned to a node i which minimizes $r_{i,x}/w_i$ according the Linear Method. In the Logarithmic Method x is assigned to the node minimizing $-\ln(1 - r_{i,x})/w_i$.

- **Advantage:**
  - Perfect probability distribution
- **Disadvantage:**
  - Intrinsic linear time w.r.t. the number of servers

- **This is the method of choice for Storage Area Networks**

The Logarithmic Method with Double Hashing



Rechnernetze und Telematik
Albert-Ludwigs-Universität Freiburg
Christian Schindelhauer

Allocation Problem in Storage Networks

- **Given:**
  - S: set of servers with bandwidth b(s) and capacity |s| for each server s
  - D: set of documents with size |d| and popularity p(d) for each document
- **Find: $A_{d,s}$: Number of bytes of document d assigned to storage s**
- **Allocation using DHHT**
  - Use DHHT to split each document d into |S| sets of blocks according to weights $\mathbf{A_{d,s}}$
  - Store blocks of all corresponding |D| subsets on server s

The Problem in SAN

- **$A_{d,s}$: Number of bytes of document d assigned to storage s**
- **Distributed Algorithm:**
  - Use DHHT to split each document into |S| parts
  - Store corresponding blocks on the server
- **Can be also achieved by a centralized algorithm**
- **Straight forward generalization of fair balance**
  - Distribute data according to a (m x n) distribution matrix A where

$$\forall s : \sum_d A_{d,s} \leq |s| \quad \text{and} \quad \forall d : \sum_s A_{d,s} = |d|$$

- **DHHT**
  - assigns $A_{d,s}(1 \pm \varepsilon)$ elements of d ∈ D to s ∈ S
  - Information needed: File-IDs, Server-IDs, and matrix A
  - If matrix A changes to A´ $(1+\varepsilon)\sum_{d,s} \left|A_{d,s} - A'_{d,s}\right|$ data reassignments are needed

How to Balance

- **A fair balance like** $A_{d,s} = |d| \cdot \frac{|s|}{\sum_{s' \in S} |s'|}$ **is not always the best to do**
- **Servers are different in capacity and bandwidth**
- **Documents are different in size and popularity**

- **Goal: Optimize Time**

- **Assumption**
  - All sizes can be modeled as real numbers

Which Time ?

- **b(s) = bandwidth of server s**
  - b(s) = number of bytes per second
- **p(d) = popularity of document d**
  - p(d) = number of read/write accesses
- **Sequential time for a document d and an assignment A**

$$\mathrm{SeqTime}_A(d) := \sum_{s\in S} \frac{A_{d,s}}{b(s)}$$

- **Parallel time for a document d and an assignment A**

$$\mathrm{ParTime}_A(d) := \max_{s \in S}\left\{\frac{A_{d,s}}{b(s)}\right\}$$

- **Observation**
  - Popular bytes cause more traffic than less popular once
  - Costs are defined by the traffic per byte

Sequential Time

- **Sequential time**
  - load all parts of a document from all servers sequentially

$$\text{SeqTime}_A(d) := \sum_{s \in \mathcal{S}} \frac{A_{d,s}}{b(s)}$$

- **Worst case sequential time**

$$\text{WSeqTime} := \max_d \{\text{SeqTime}_A(d)\}$$

- **Average sequential time**

$$\text{AvSeqTime} := \sum_{d \in \mathcal{D}} p(d)\ \text{SeqTime}_A(d)$$

- **where**
  - S: set of servers with bandwidth b(s) and capacity |s| for each server s
  - D: set of documents with size |d| and popularity p(d) for each document

Parallel Time

- **Parallel time**
  - load all parts of a document from all servers simultaneously

$$\text{ParTime}_A(d) := \max_{s \in \mathcal{S}} \left\{ \frac{A_{d,s}}{b(s)} \right\}$$

- **Worst case parallel time**

$$\text{WParTime} := \max_d \{\text{ParTime}_A(d)\}$$

- **Average parallel time**

$$\text{AvParTime} := \sum_{d \in \mathcal{D}} p(d)\ \text{ParTime}_A(d)$$

- **where**
  - S: set of servers with bandwidth b(s) and capacity |s| for each server s
  - D: set of documents with size |d| and popularity p(d) for each document

Sequential Bandwidth

- **Sequential time**
  - load all parts of a document from all servers sequentially

$$\text{SeqTime}_A(d) := \sum_{s \in \mathcal{S}} \frac{A_{d,s}}{b(s)}$$

- **Sequential bandwidth**
  - download speed of a document d

$$\text{SeqBandwidth}_A(d) := \frac{|d|}{\text{SeqTime}_A(d)}$$

- **Worst case sequential bandwidth**

$$\text{WBandwidth} := \min_d \{\text{SeqBandwidth}_A(d)\}$$

- **Average sequential bandwidth**

$$\text{AvBandwidth} := \sum_{d \in \mathcal{D}} p(d)\ \text{SeqBandwidth}(d)$$

- **where**
  - S: set of servers with bandwidth b(s) and capacity |s| for each server s
  - D: set of documents with size |d| and popularity p(d) for each document

Parallel Bandwidth

- **Parallel time**
  - load all parts of a document from all servers in parallel

$$\text{ParTime}_A(d) := \max_{s \in \mathcal{S}} \left\{ \frac{A_{d,s}}{b(s)} \right\}$$

- **Parallel bandwidth**
  - download speed of a datum d

$$\text{ParBandwidth}_A(d) := \frac{|d|}{\text{ParTime}_A(d)}$$

- **Worst case parallel bandwidth**

$$\text{WParBandwidth} := \min_d \{\text{ParBandwidth}_A(d)\}$$

- **Average parallel bandwidth time**

$$\text{AvParBandwidth} := \sum_{d \in \mathcal{D}} p(d)\ \text{ParBandwidth}_A(d)$$

- **where**
  - S: set of servers with bandwidth b(s) and capacity |s| for each server s
  - D: set of documents with size |d| and popularity p(d) for each document

Most Reasonable Time Measures

- **Minimize the expected sequential time based on popularity of the document:**

$$\mathrm{AvSeqTime}(p, A) = \sum_{d \in \mathcal{D}} \sum_{s \in \mathcal{S}} p(d) \frac{A_{d,s}}{b(s)}$$

- **Minimize the expected parallel time based on the popularity of the document**

$$\mathrm{AvParTime}(p, A) = \sum_{d \in D} \max_{s \in S} \frac{A_{d,s}}{b(s)} p(d)$$

How to Describe AvParTime as a LP

Variables: $A_{d,s}, m_d$

Restraints: $\sum_s A_{d,s} = |d|$

$\sum_d A_{d,s} \leq |s|$

AvParTime

$$= \sum_{d \in D} p(d) \cdot \underbrace{\max_{s \in S} \frac{A_{d,s}}{b(s)}}_{m_d}$$

$$= \sum_{d \in D} p(d) \cdot m_d$$

$$m_d = \max_{s \in S} \frac{A_{d,s}}{b(s)}$$

Additional Restraints

$$m_d \geq \frac{1}{b(s_1)} \cdot A_{d,s_1}$$

$$m_d \geq \frac{1}{b(s_2)} \cdot A_{d,s_2}$$

$\vdots$

Solution by Linear Program

$$\forall s : \sum_d A_{d,s} \leq |s| \qquad \forall d : \sum_s A_{d,s} = |d|$$

Measure	Linear programm	Add. variables	Additional restraint	Optimize
AvSeqTime	yes	—	—	$\min \sum_{s \in \mathcal{S}} \sum_{d \in \mathcal{D}} p(d) \frac{A_{d,s}}{b(s)}$
WSeqTime	yes	m	$\forall d \in \mathcal{D} : \sum_{s \in \mathcal{S}} \frac{A_{d,s}}{b(s)} \leq m$	$\min m$
AvParTime	yes	$(m_d)_{d \in \mathcal{D}}$	$\forall s \in \mathcal{S}, \forall d \in \mathcal{D} : \frac{A_{d,s}}{b(s)} \leq m_d$	$\min \sum_{d \in \mathcal{D}} p(d) m_d$
WParTime	yes	m	$\forall s \in \mathcal{S}, \forall d \in \mathcal{D} : \frac{A_{d,s}}{b(s)} \leq m$	$\min M$
AvSeqBandwidth	no	—	—	$\max \sum_{d \in \mathcal{D}} \frac{p(d)\lvert d\rvert}{\sum_{s \in \mathcal{S}} \frac{A_{d,s}}{b(s)}}$
WSeqBandwidth	yes	m	$\forall d \in \mathcal{D} : \sum_{s \in \mathcal{S}} \frac{A_{d,s}}{\lvert d\rvert b(s)} \leq m$	$\min m$
AvParBandwidth	no	$(m_d)_{d \in \mathcal{D}}$	$\forall d \in \mathcal{D} : \sum_{s \in \mathcal{S}} \frac{A_{d,s}}{b(s)\lvert d\rvert} \leq m_d$	$\max \sum_{d \in \mathcal{D}} \frac{p(d)}{m_d}$
WParBandwidth	yes	m	$\forall s \in \mathcal{S}, \forall d \in \mathcal{D} : \frac{A_{d,s}}{\lvert d\rvert b(s)} \leq m$	$\min m$

Example

- **Storage device**
  - s_1: 500 GB, 100 MB/s
  - s_2: 100 GB, 50 MB/s
  - s_3: 1 GB 1000 MB/s
- **Documents**
  - d_1: 100 GB, popularity 1/111
  - d_2: 5 GB, popularity 100/111
  - d_3: 100 GB, popularity 10/111

$A_{d,s}$	s_1	s_2	s_3	Σ
d_1	100	0	0	100
d_2	2	2	1	5
d_3	2	98	0	100
Σ	≤ 500	≤ 100	≤ 1	

	SeqTime	SeqBand width	ParTime	ParBand width
d_1	1000	100	1000	100
d_2	61	82	40	125
d_3	1980	51	1960	51
Av	1864	121	1827	160
Worst case	1980	51	1960	51

Excursion: Linear Programming

- **Linear Program (Linear Optimization)**
- **Given:** m × n matrix A

  m-dimensional vector b

  n-dimensional vector c
- **Find:** n-dimensional vector $x=(x_1, ..., x_n)$
- **such that**
  - $x \geq 0$, i.e. for all j: $x_j \geq 0$
  - A x = b, i. e. $$\sum_{j=1}^{n}\sum_{i=1}^{m} A_{ij}x_j = b_j$$
  - $z = c^T x$ is minimized, i.e. $z = \sum_{j=1}^{n} c_j x_j$ is minimal

Linear Programming 2

- **Linear Programming (LP2)**
- **Given:** $m \times n$ matrix A

  m-dimensional vector b

  n-dimensional vector c
- **Find:** n-dimensional vector $x=(x_1, ..., x_n)$
- **such that**
  - $x \geq 0$
  - $A\, x \leq b$
  - $z = c^T x$ is maximal

LP = LP2

- **Lemma**
  - LP can be reformulated as an LP2 and vice versa.
  - The problem size increases only by a constant factor.
- **Proof:**

Geometric Interpretation

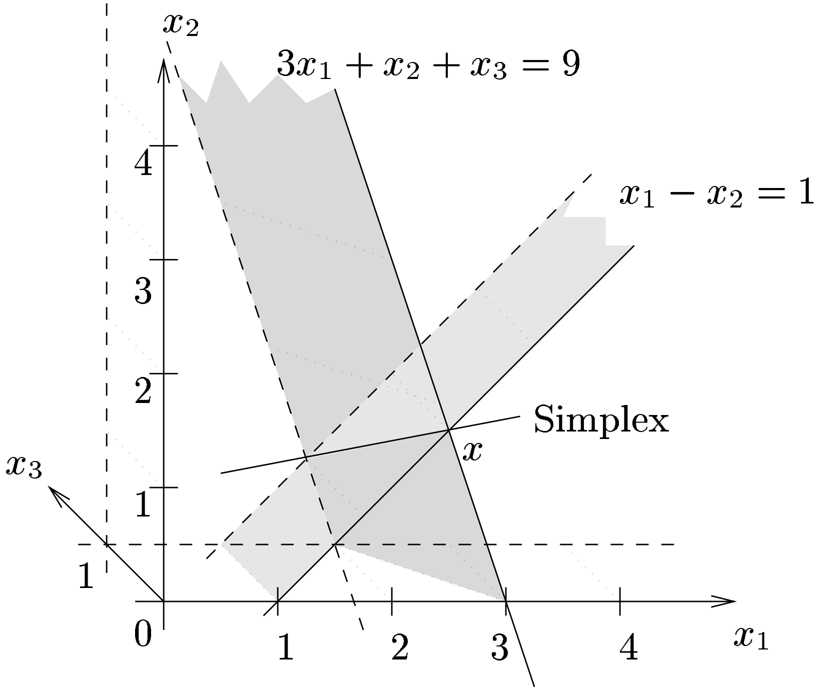
- **Example:**
  - A x = b
  - with $A = \begin{pmatrix} 1 & -1 & 0 \\ 3 & 1 & 1 \end{pmatrix}$

$$b = \begin{pmatrix} 1 \\ 9 \end{pmatrix}$$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

  - Minimize for x≥0 the term $c^T x$ where

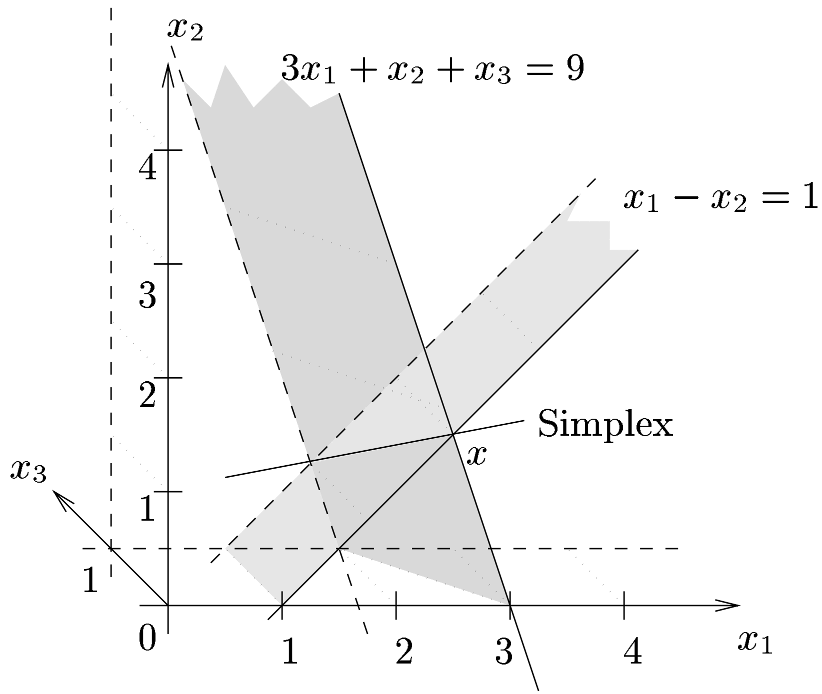
$$c^T = (0\ 0\ -1)$$



Rechnernetze und Telematik
Albert-Ludwigs-Universität Freiburg
Christian Schindelhauer

Simplex Algorithm

- **All solutions are in an intersection**
  - of hyper-planes (A x = b)
  - and half-planes x≥0
- **This is a simplex**
- **First construct a basis solution x on the vertices of the simplex**
  - x_i is called a basis variable
  - which suffices Ax=b and x≥0
  - but is not optimal
    - if x_i=0 it is called degenerated
- **Consider all edges of the simplex**
  - walk along the edge which improves the solution
  - until the next the next vertex
  - Choose it as new basis solution
- **Repeat until the optimum has been reached**



Intuition for the Simplex-Algorithm

$$A = \left(\underbrace{\begin{array}{|c|}\hline B \\ \hline\end{array}}_{m} \Big\}{m} \;\; \underbrace{\begin{array}{|c|}\hline N \\ \hline\end{array}}_{n-m} \right)$$

$$c = \begin{pmatrix} c_B \\ c_N \end{pmatrix} \begin{matrix} \}\, m \\ \}\, n-m \end{matrix}$$

A line in A describes the normal vector of the hyper-plane.

Computing the Parallel Vectors

$$M = \begin{pmatrix} B & N \\ 0 & E_{n-m} \end{pmatrix} \begin{matrix} \}\, m \\ \}\, n-m \end{matrix}$$

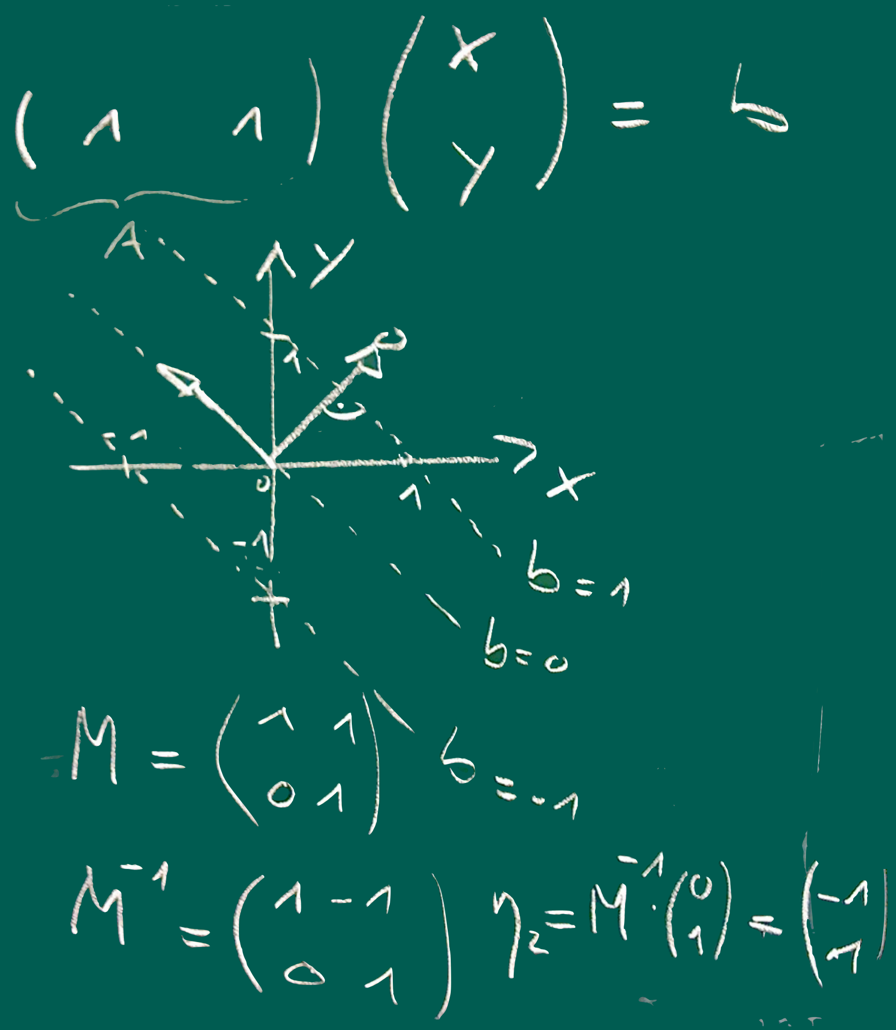
$$\underbrace{\quad}_{m} \quad \underbrace{\quad}_{n-m}$$

$$E_{n-m} = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix}$$

$$M^{-1} = \begin{pmatrix} B^{-1} & -B^{-1} \cdot N \\ 0 & E_{n-m} \end{pmatrix}$$

$$\eta_q = M^{-1} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \}\, q = M^{-1} \cdot e_q$$

2D Example



Rechnernetze und Telematik
Albert-Ludwigs-Universität Freiburg
Christian Schindelhauer

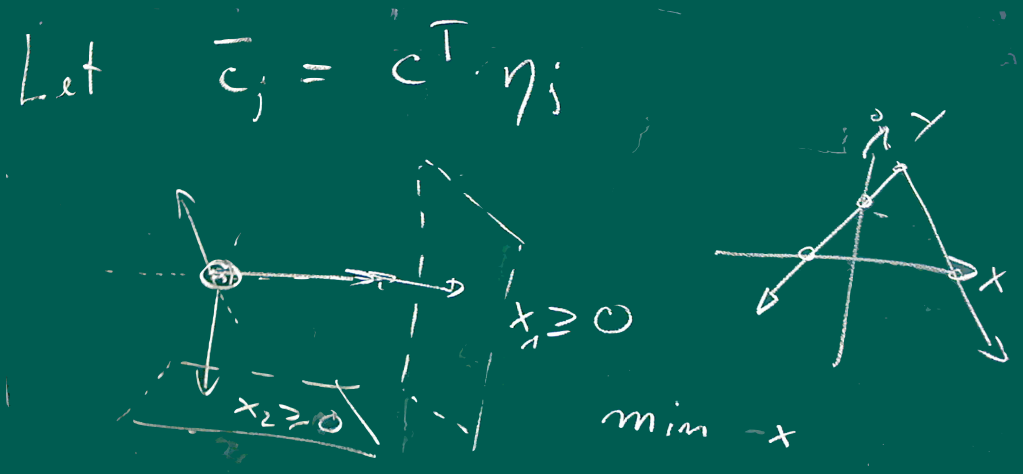
The Solution is in Sight

For $q \geq m$ η_q is a vector parallel to the $m-1$ hyper-planes which are <u>not</u> the q-th line of A.

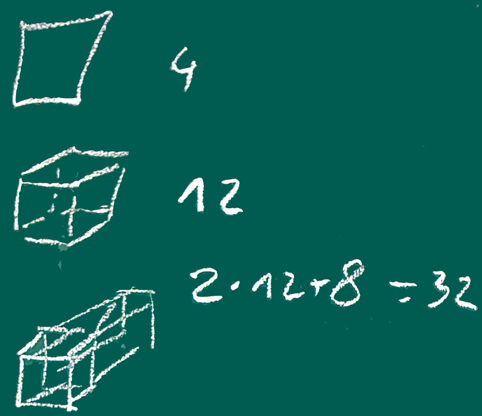
If x is a solution for $Ax = b$
Then every point y of the solution space is described by

$$y = x + \sum_{j=m+1}^{n} \eta_j \cdot z_j \; ; \; z_j \in \mathbb{R}$$

c gives the direction



too many edge in high dimensions



Simplex Algorithm

Simplex Algorithm

input: $m \times n$-matrix A,
m-dim. vector b
n-dim. vector c

$\{$ $I_B \leftarrow$ a set $\{j_1, \ldots, j_m\}$ of m positions with independent column vectors in A
$B \leftarrow (a_{j_1}, \ldots, a_{j_m})$
$x \leftarrow B^{-1}b$
$stop \leftarrow false$
while $\neg stop$ **do**
$\{$ $c_B \leftarrow (c_{j_1}, \ldots, c_{j_m})$
for all $j \notin I_B$ **do** $\overline{c_j} \leftarrow c_j - c_B B^{-1} a_j$
$optimal \leftarrow \bigwedge_{j \notin I_B} \overline{c_j} \geq 0$
$stop \leftarrow optimal$
if $\neg stop$ **then**
$\{$ $V \leftarrow \{j \notin I_B \mid \overline{c_j} < 0\}$
$q \leftarrow$ arbitrary element from V
$w \leftarrow B^{-1} a_q$
$stop \leftarrow (w \leq 0)$
if $\neg stop$ **then**
$\{$ Determine j_p such that $\frac{x_{j_p}}{w_p} = \min_{1 \leq i \leq m}\{\frac{x_{j_i}}{w_i} \mid w_i \geq 0\}$
$s \leftarrow \frac{x_{j_p}}{w_p}$
$x_q \leftarrow s$
for all $i \in \{1, \ldots m\}$ **do** $x_{j_i} \leftarrow x_{j_i} - s w_i$
$B \leftarrow$ replace column q by column j_p.
$I_B \leftarrow (I_B \setminus \{q\}) \cup \{j_p\}$
$j_p \leftarrow q$
$\}$

$\}$

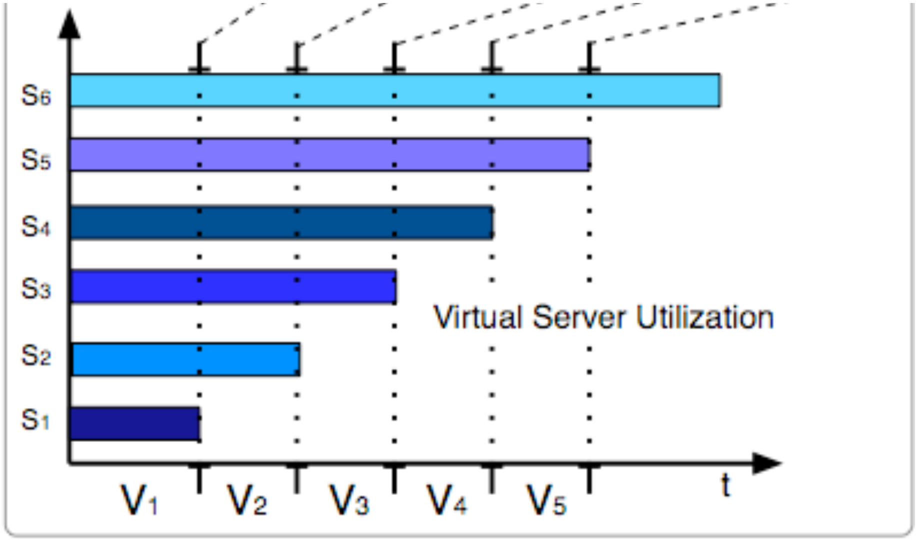
$\}$
if $optimal$ **then return** x
else return no lower bound
$\}$

Performance

- **Worst case time behavior of the Simplex algorithm is exponential**
  - A simplex can have an exponential number of edges
- **For randomized inputs, the running time of Simplex is polynomial on the expectation**
- **The Ellipsoid algorithm is a different method with polynomial worst case behavior**
  - In practice it is usually outperformed by the Simplex algorithm

ParTime = SeqTime with virtual servers

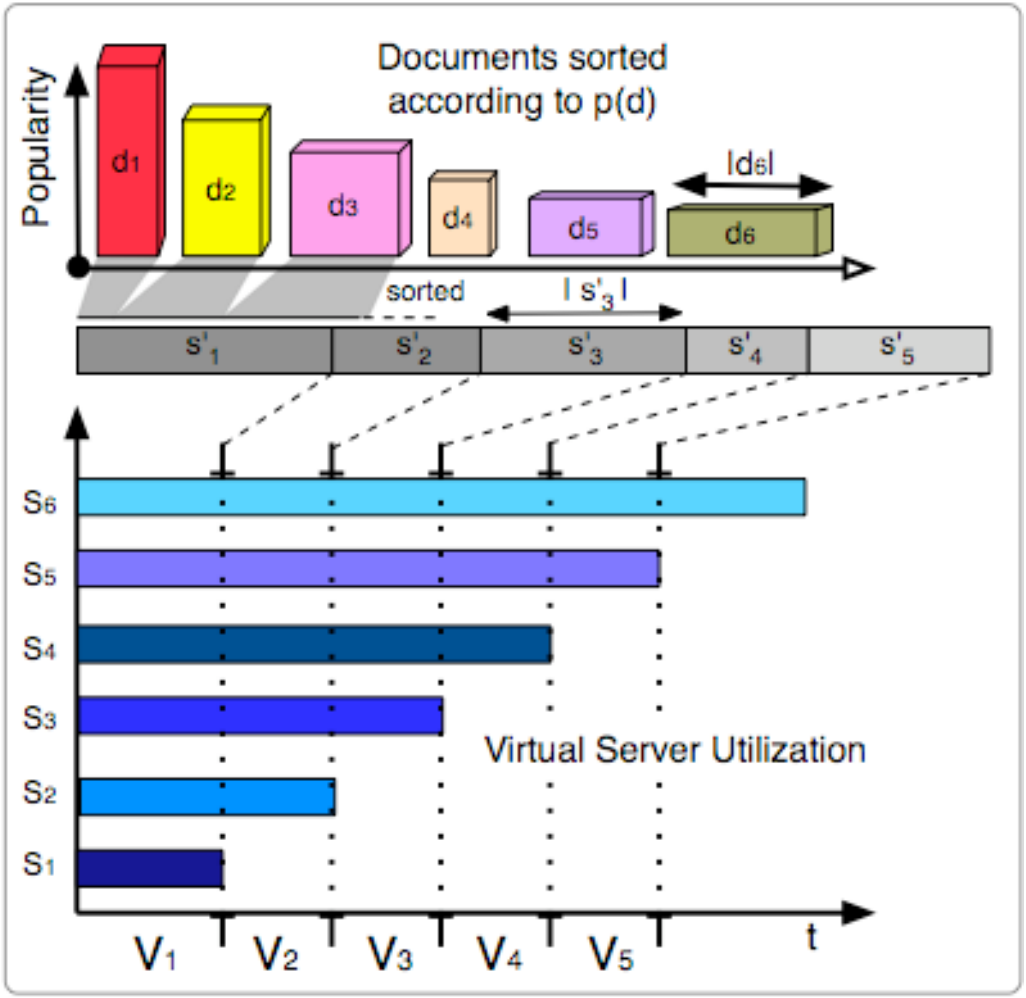
- **Reduce optimal solution for LP of ParTime to the optimal solution of LP of SeqTime**
  - Combining capacity of many disks in parallel
- **Define new sequential virtual servers $s'_1, \ldots, s'_m$**
  - Sort s_i such that $\frac{|s_j|}{b(s_j)} \leq \frac{|s_{j+1}|}{b(s_{j+1})}$
  - Server s'_j parallelizes servers $s_j, \ldots, s_{|S|}$
  - Virtual servers s'_i are then sorted such that $b(s'_i) > b(s'_{i+1})$
  - Size of s'_i:



$$t_j = \frac{|s_j|}{b(s_j)} - \sum_{i=1}^{j-1} t_i \qquad\qquad s'_j = b(s'_j) \cdot t_j$$

Solve the LP of AvSeqTime

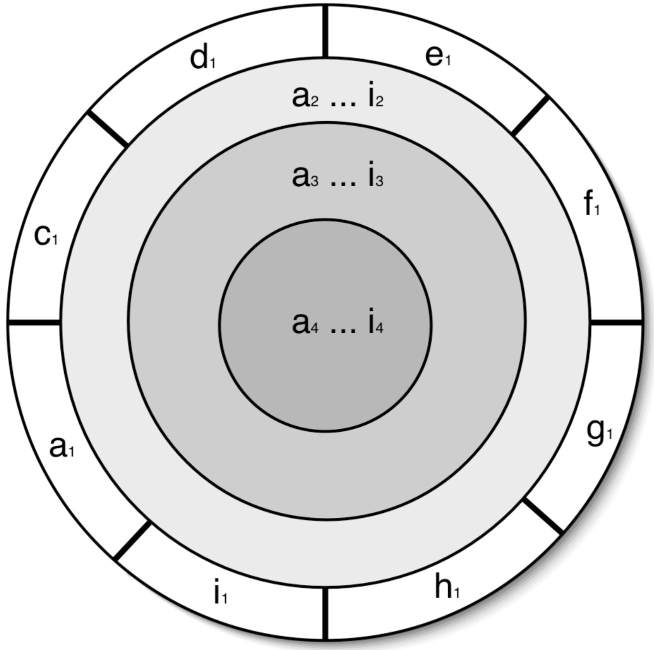
- **Simple optimal greedy solution**
- **Repeat until all documents are assinged:**
  - Assign most popular document on fastest sequential (virtual) server
  - Reduce the storage of the server by the document size and remove the document



Applications in SAN

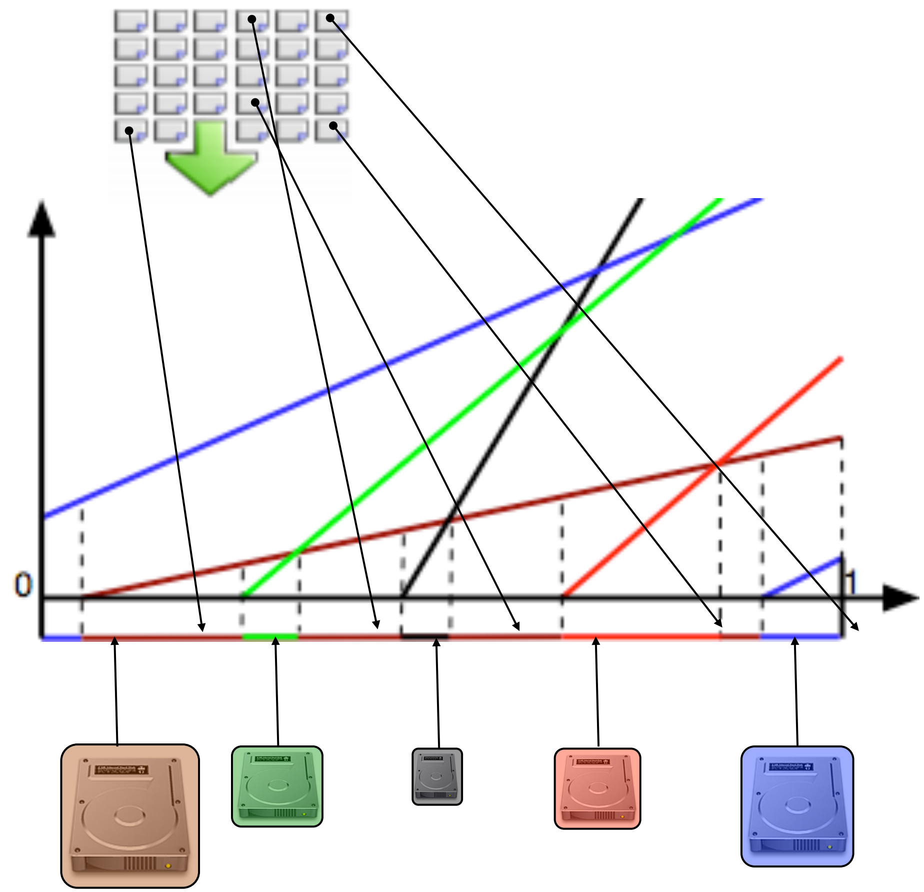
- **Object storage with different popularity zones**
  - e.g. movies with varying popularities over time
  - Fragmentation is done automatically
  - Includes dynamics for adding and removing documents
  - The same for servers

- **Use different bandwidth**
  - Each disk has different bandwidths
  - Exporting different zone classes as sequential servers



From DHT to DHHT

- **Distributed Heterogeneous Hash Table (DHHT)**
  - a straight-forward extension of the original DHT
  - efficient, fair
- **Linear Method**
  - Nice pictures
  - Performs quite well
  - Needs copies for fairness, and O(log n) partitions
- **Logarithmic Method**
  - Performs perfectly
  - Needs O(log n) partitions if more than one data item is used
  - is optimal when combined with double hashing
- **Applications of DHHT**
  - MANET, Peer-to-Peer-Networks
  - SAN: optimize time with very simple assignment rules



ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Algorithms and Methods for Distributed Storage Networks

10 Heterogeneous Virtualization Methods

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik
Wintersemester 2007/08

CoNe Freiburg

IIF
INSTITUT FÜR INFORMATIK FREIBURG