DAAD Summerschool Curitiba 2011

Aspects of Large Scale High Speed Computing Building Blocks of a Cloud

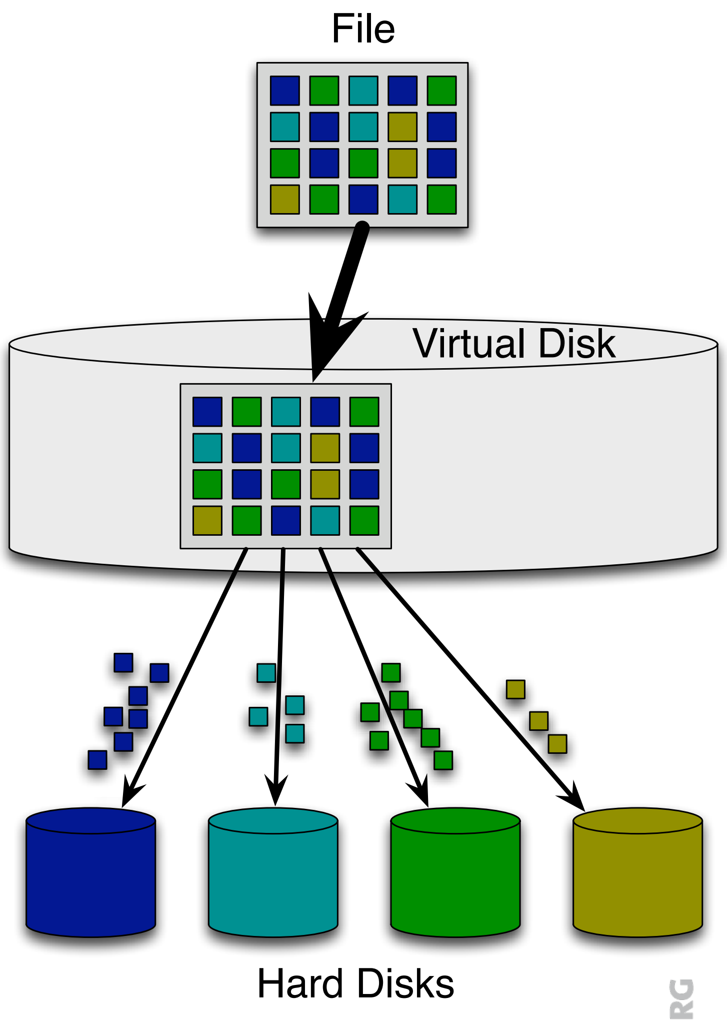
Storage Networks

3: Distributed Hash Tables - Virtualization without Index Database

Christian Schindelhauer
Technical Faculty
Computer-Networks and Telematics
University of Freiburg

Concept of Virtualization

- **Principle**
  - A virtual storage constitutes handles all application accesses to the file system
  - The virtual disk partitions files and stores blocks over several (physical) hard disks
  - Control mechanisms allow redundancy and failure repair
- **Control**
  - Virtualization server assigns data, e.g. blocks of files to hard disks (address space remapping)
  - Controls replication and redundancy strategy
  - Adds and removes storage devices

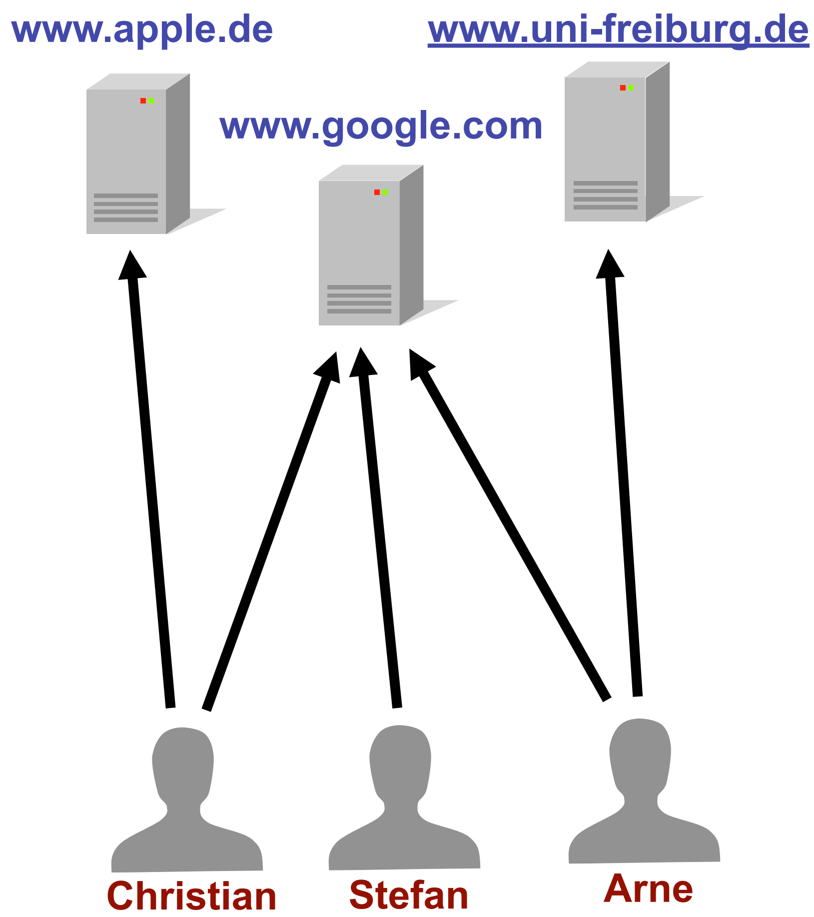


Distributed Wide Area Storage Networks

- Distributed Hash Tables
  - Relieving hot spots in the Internet
  - Caching strategies for web servers
- Peer-to-Peer Networks
  - Distributed file lookup and download in Overlay networks
  - Most (or the best) of them use: DHT

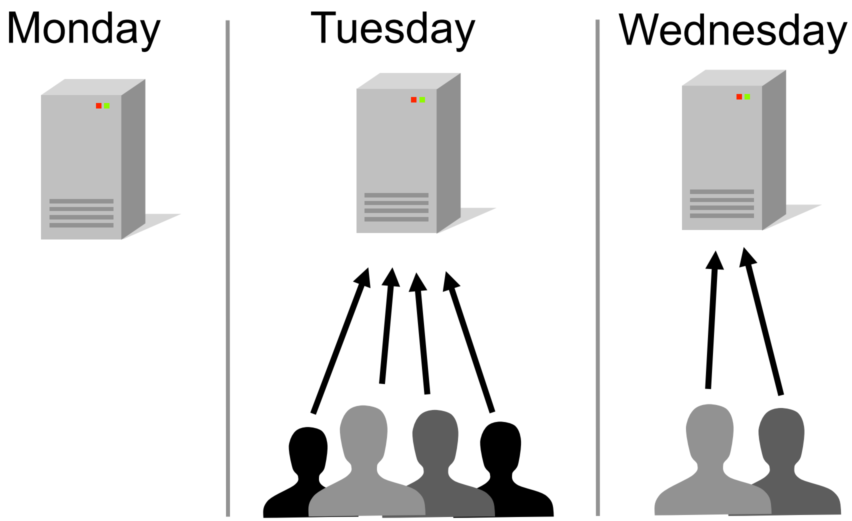
WWW Load Balancing

- Web surfing:
  - Web servers offer web pages
  - Web clients request web pages
- Most of the time these requests are independent
- Requests use resources of the web servers
  - bandwidth
  - computation time



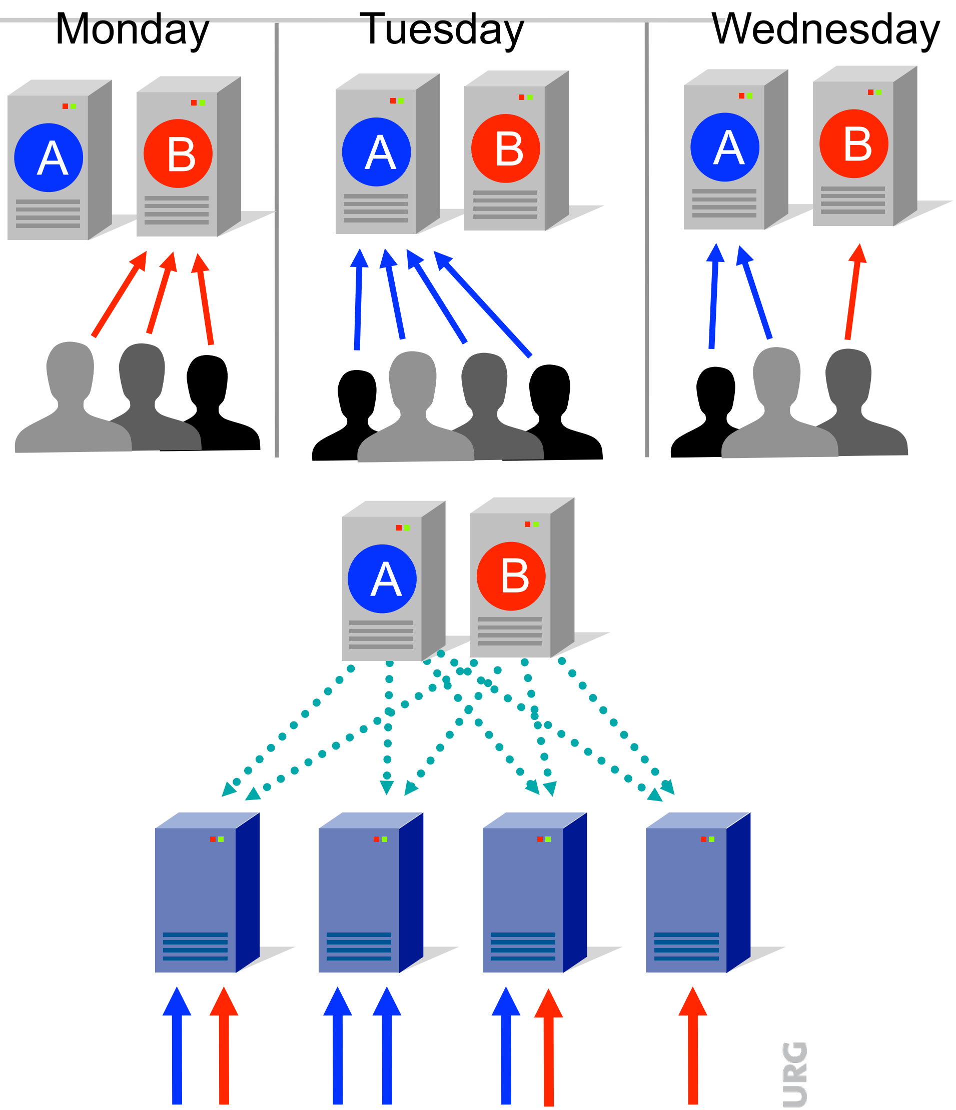
Load

- **Some web servers have always high load**
  - for permanent high loads servers must be sufficiently powerful
- **Some suffer under high fluctuations**
  - e.g. special events:
    - jpl.nasa.gov (Mars mission)
    - cnn.com (terrorist attack)
  - Server extension for worst case not reasonable
  - Serving the requests is desired



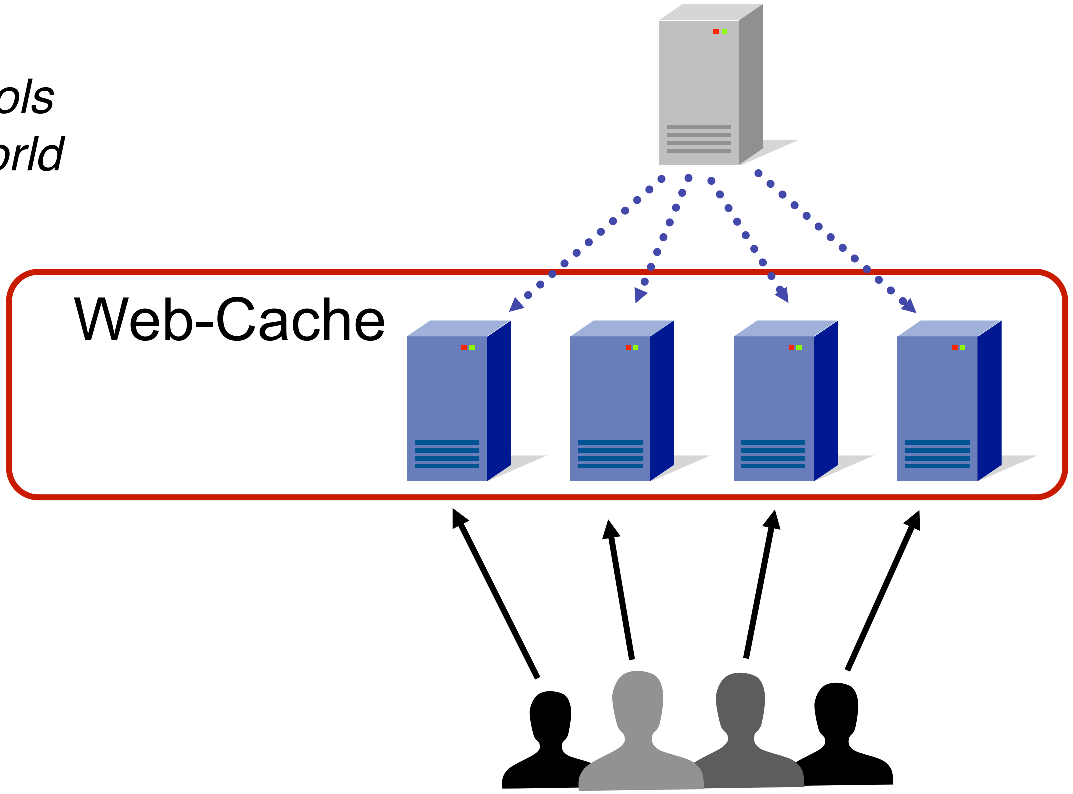
Load Balancing in the WWW

- Fluctuations target some servers
- (Commercial) solution
  - Service providers offer exchange servers an
  - Many requests will be distributed among these servers
- But how?



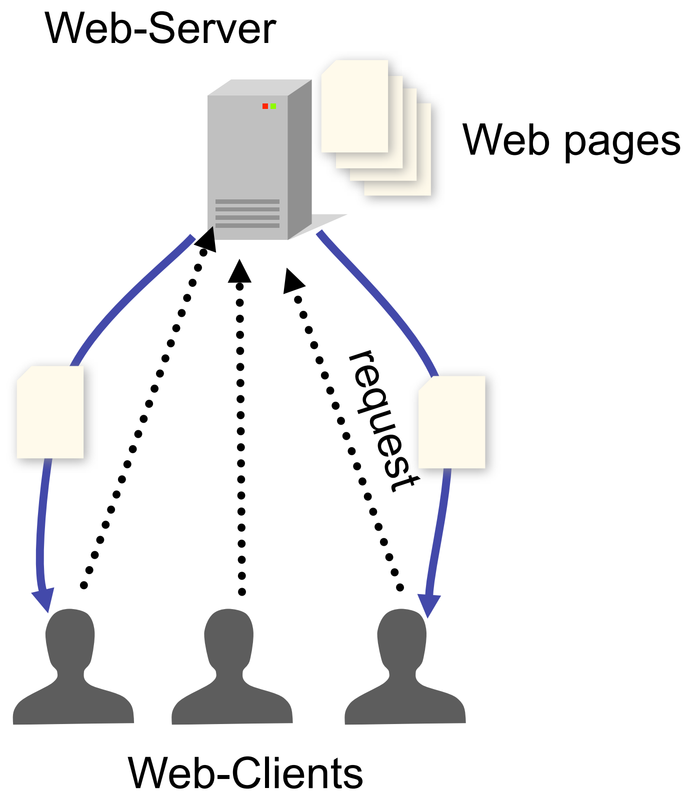
Literature

- **Leighton, Lewin, et al. STOC 97**
  - *Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web*
- **Used by Akamai (founded 1997)**



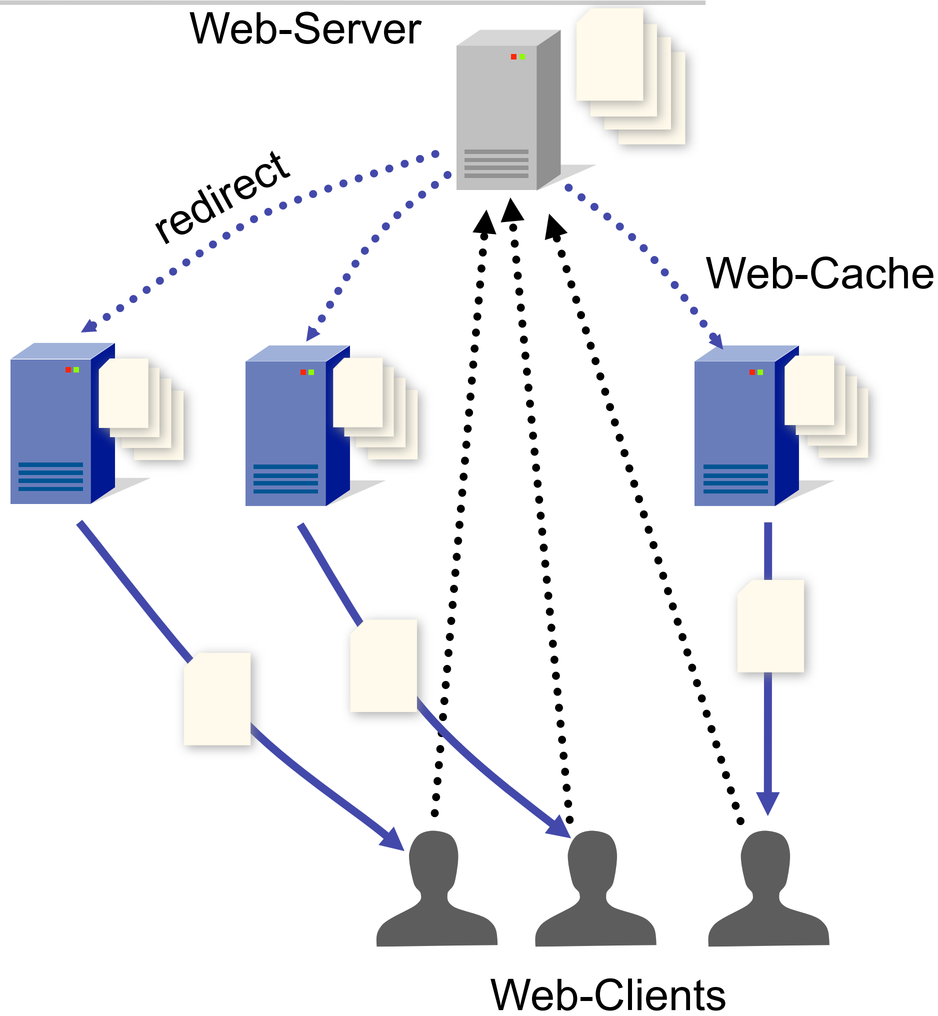
Start Situation

- **Without load balancing**
- **Advantage**
  - simple
- **Disadvantage**
  - servers must be designed for worst case situations



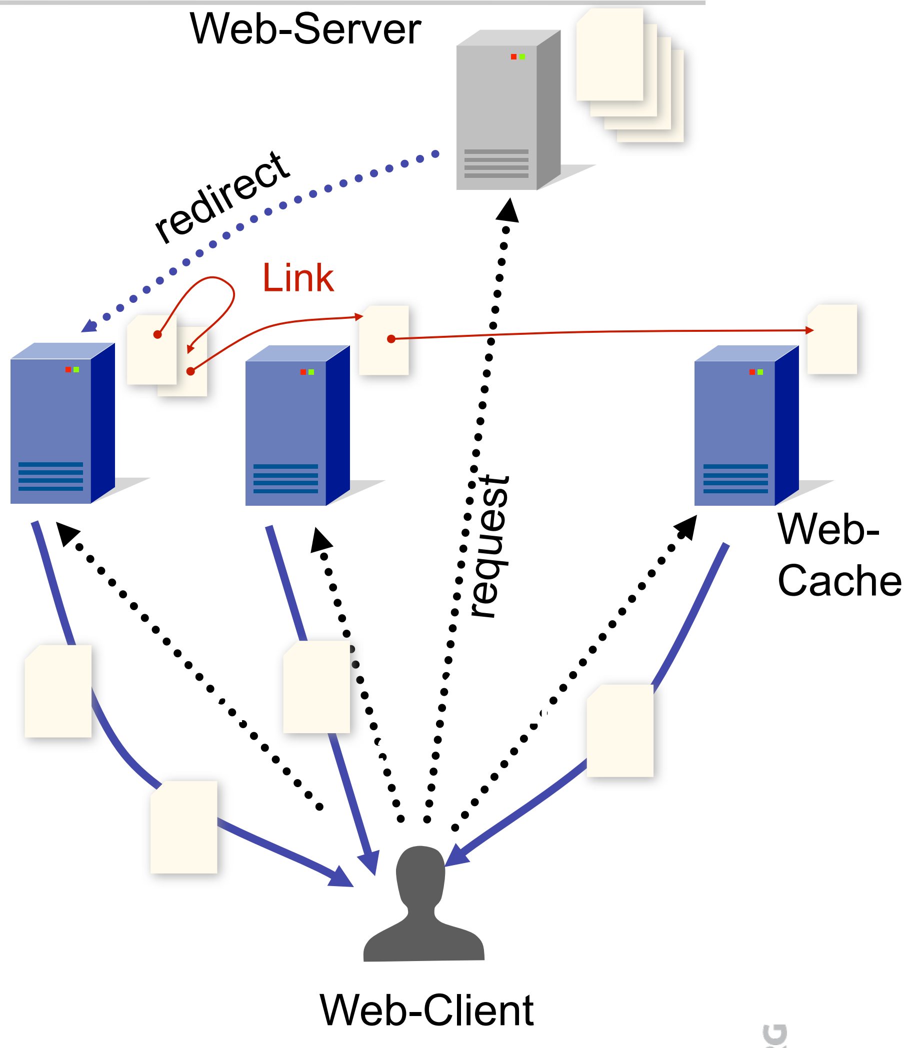
Site Caching

- **The whole web-site is copied to different web caches**
- **Browsers request at web server**
- **Web server redirects requests to Web-Cache**
- **Web-Cache delivers Web pages**
- **Advantage:**
  - good load balancing
- **Disadvantage:**
  - bottleneck: redirect
  - large overhead for complete web-site replication



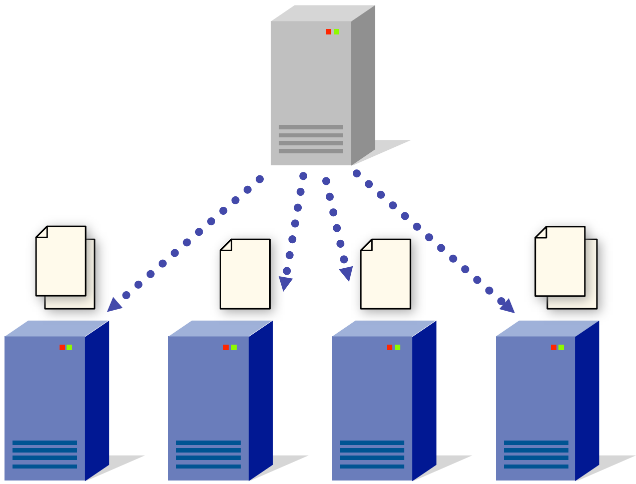
Proxy Caching

- **Each web page is distributed to a few web-caches**
- **Only first request is sent to web server**
- **Links reference to pages in the web-cache**
- **Then, web clients surfs in the web-cache**
- **Advantage:**
  - No bottleneck
- **Disadvantages:**
  - Load balancing only implicit
  - High requirements for placements

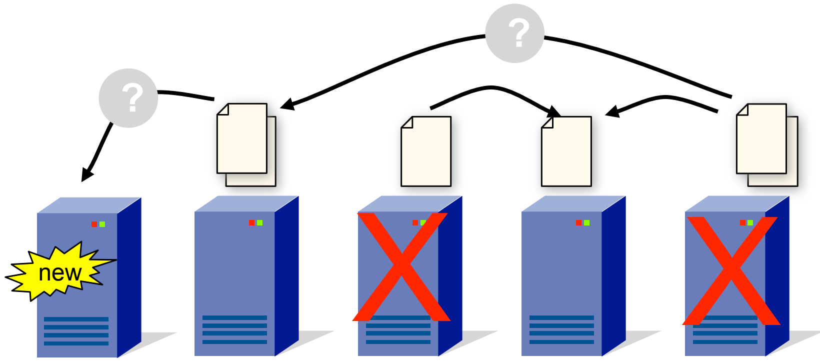


Requirements

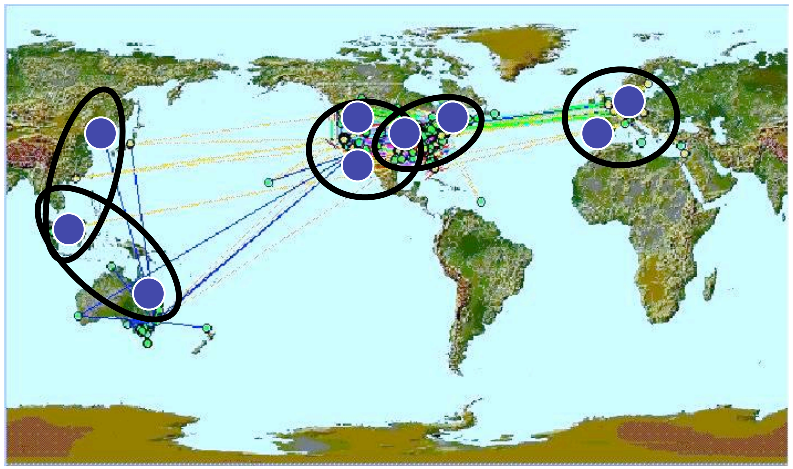
Balance
fair balancing of web pages



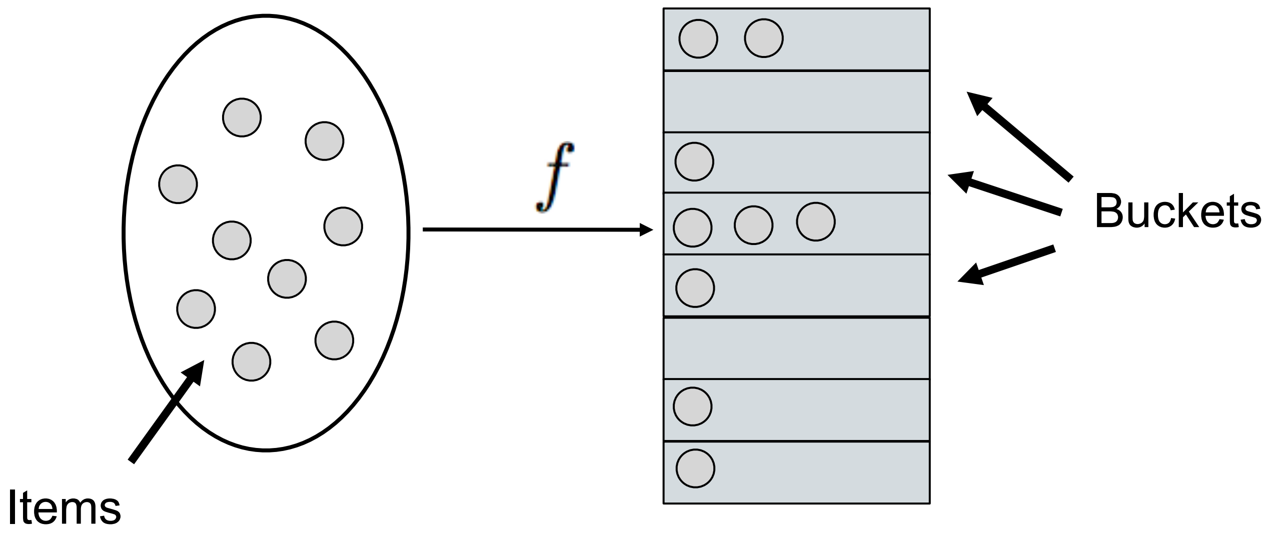
Dynamics
Efficient insert and delete of web-cache-servers and files



Views
Web-Clients „see“ different set of web-caches



Hash Functions



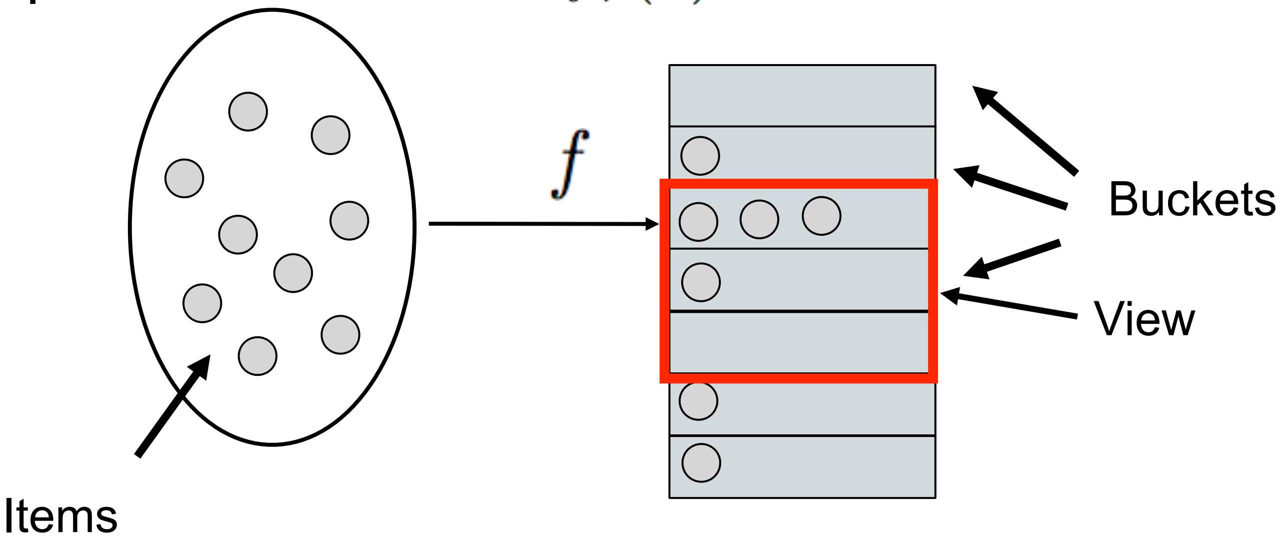
Set of Items: $\mathcal{I}$

Set of Buckets: $\mathcal{B}$

Example: $f(i) = ai + b \bmod n$

Ranged Hash-Funktionen

- Given:
  - Items $\mathcal{I}$, Number $I := |\mathcal{I}|$
  - Caches (Buckets), Bucket set: $\mathcal{B}$
  - Views $\mathcal{V} \subseteq 2^{\mathcal{B}}$
- Ranged Hash-Funktion:
  - $f : 2^{\mathcal{B}} \times \mathcal{I} \rightarrow \mathcal{B}$
  - Prerequisite: for alle views $f_{\mathcal{V}}(\mathcal{I}) \subseteq \mathcal{V}$

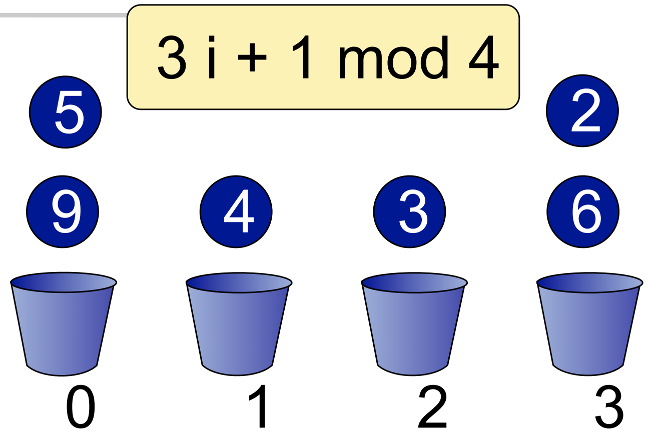


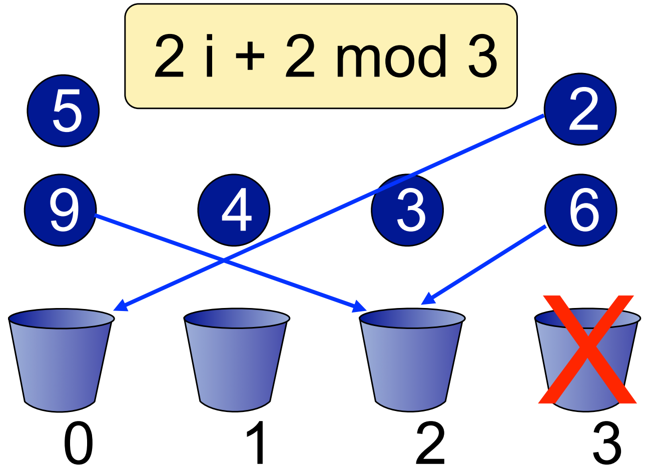
First Idea: Hash Function

- Algorithm:
  - Choose Hash funktion, e.g.

    $$f(i) = ai + b \bmod n$$

    n: number of Cache servers
- Balance:
  - very good
- Dynamics
  - Insert or remove of a single cache server
  - New hash functions and total re-hashing
  - Very expensive!!





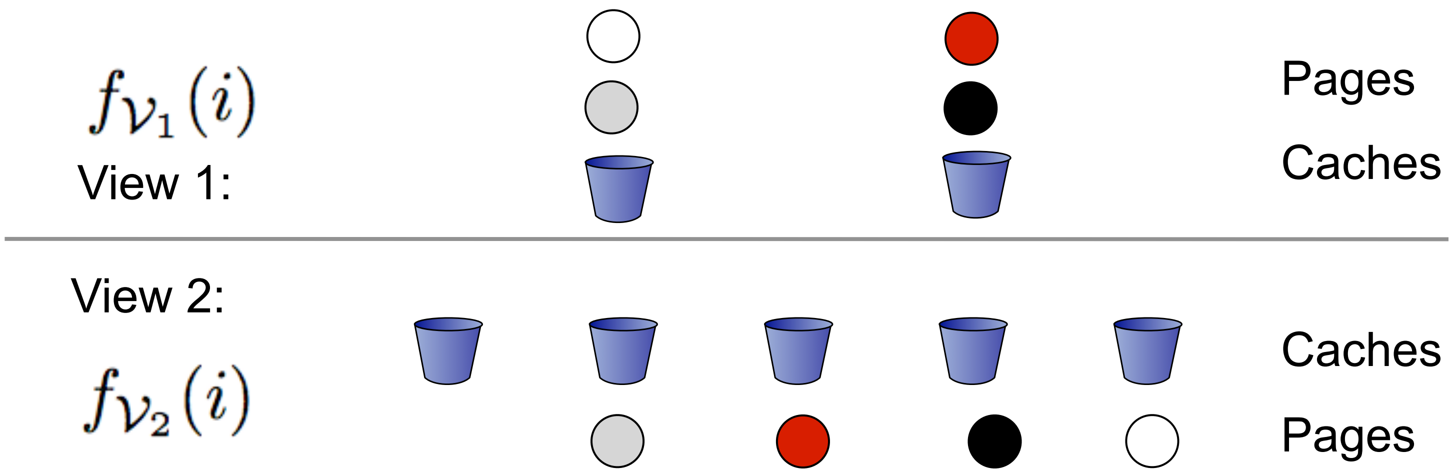
Requirements of the Ranged Hash Functions

- Monotony
  - After adding or removing new caches (buckets) no pages (items) should be moved
- Balance
  - All caches should have the same load
- Spread
  - A page should be distributed to a bounded number of caches
- Load
  - No Cache should not have substantially more load than the average

Monotony

- After adding or removing new caches (buckets) no pages (items) should be moved
- Formally: For all $\mathcal{V}_1 \subseteq \mathcal{V}_2 \subseteq \mathcal{B}$

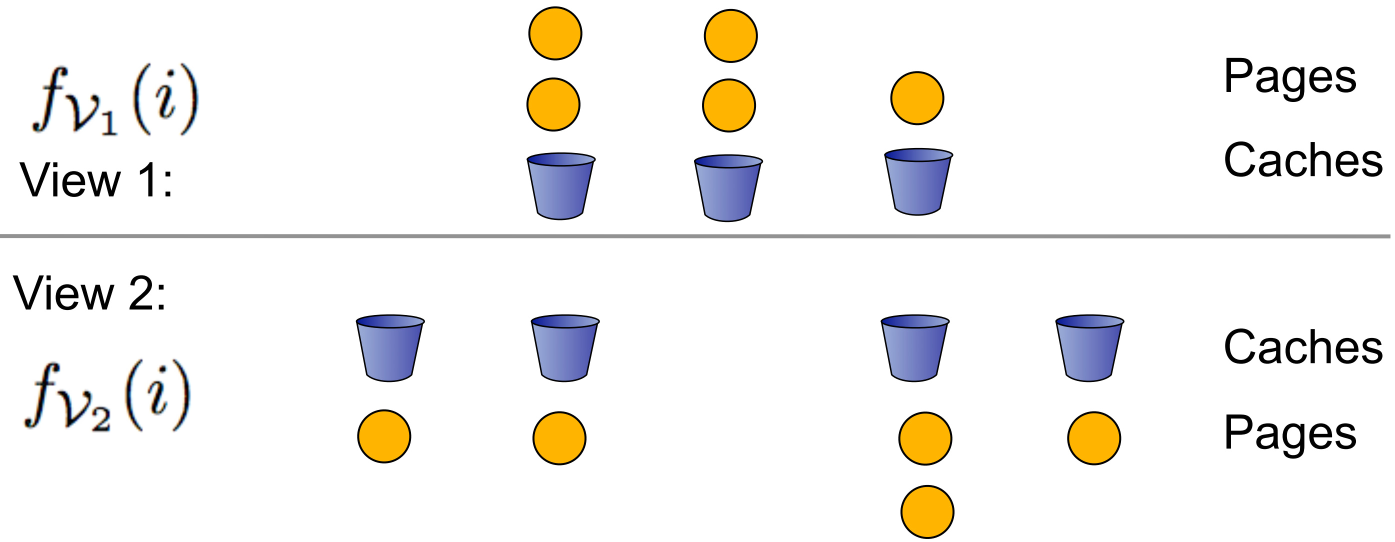
$$f_{\mathcal{V}_2}(i) \in \mathcal{V}_1 \;\Rightarrow\; f_{\mathcal{V}_1}(i) = f_{\mathcal{V}_2}(i)$$



Balance

- For every view V the is the $f_V(i)$ balanced
  For a constant c and all $\mathcal{V} \subseteq \mathcal{B}$:

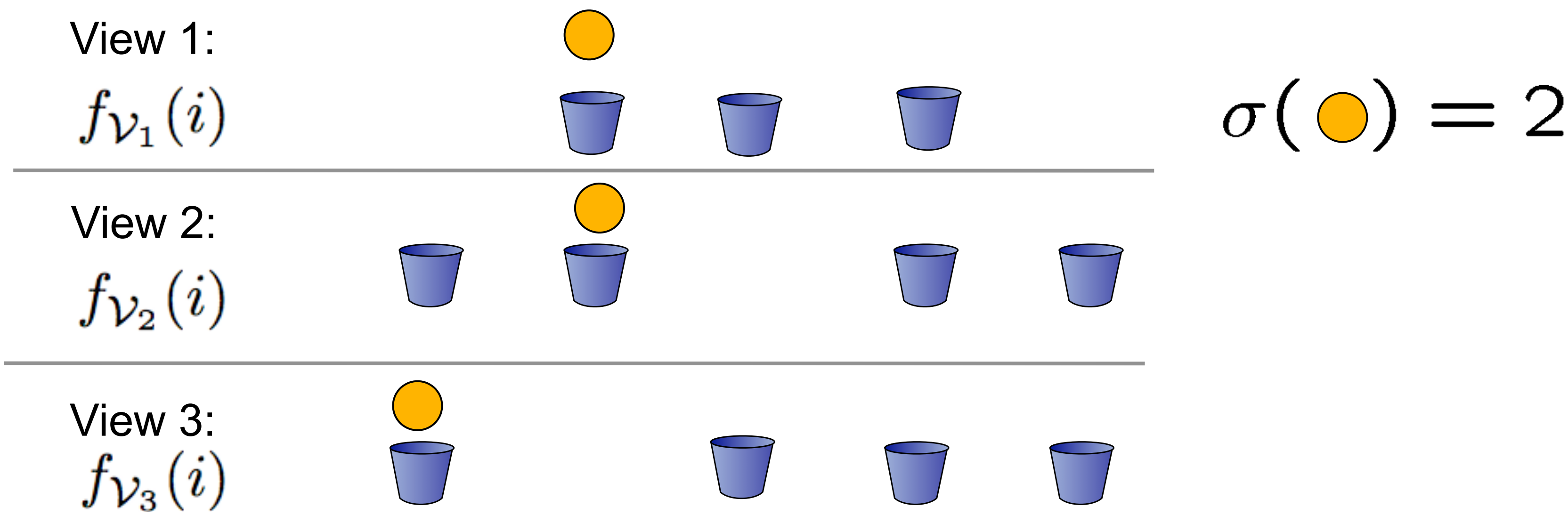
$$\Pr\left[f_{\mathcal{V}}(i) = b\right] \le \frac{c}{|\mathcal{V}|}$$



Spread

- The spread σ(i) of a page i is the overall number of all necessary copies (over all views)

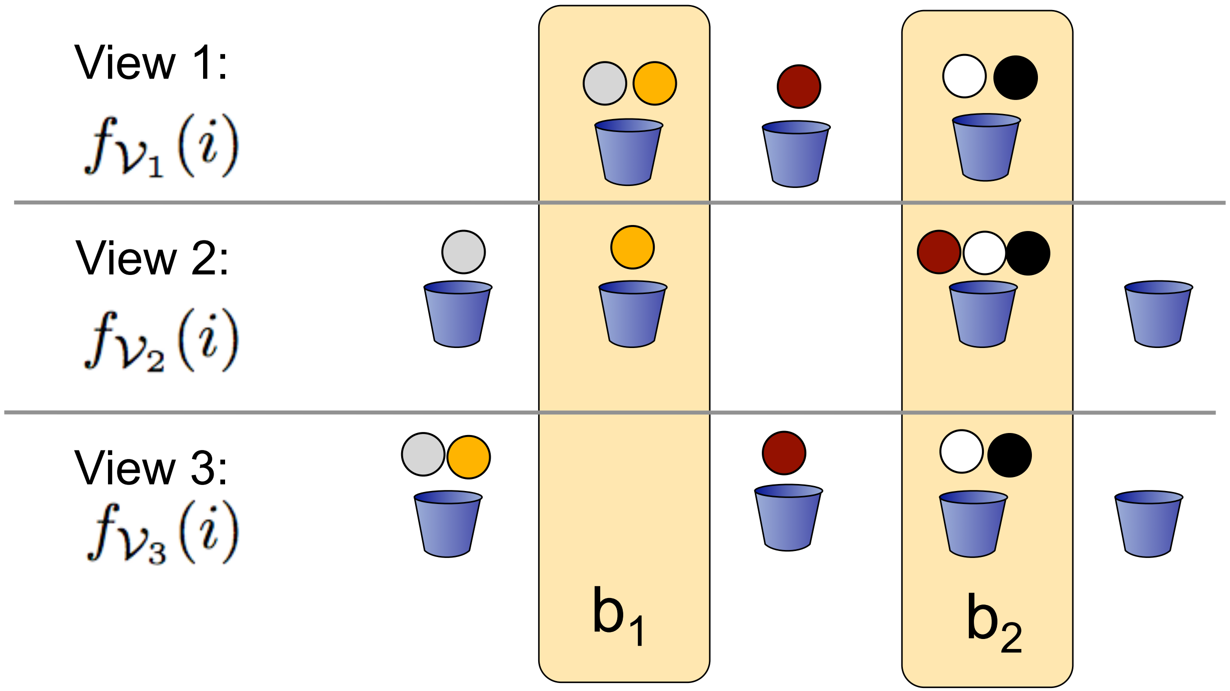
$$\sigma(i) := |\{f_{\mathcal{V}_1}(i), f_{\mathcal{V}_2}(i), \ldots, f_{\mathcal{V}_V}(i)\}|$$



Load

- The load $\lambda(b)$ of a cache b is the over-all number of all copies (over all views) $\lambda(b) := |\{ \cup_{\mathcal{V}} \ H_{\mathcal{V}}(b) \ \}|$,

  wher $H_{\mathcal{V}}(b)$:= set of all pages assigned to bucket b in View V



$\lambda(b_1) = 2$

$\lambda(b_2) = 3$

Distributed Hash Tables

C	number of caches (Buckets)
C/t	minimum number of caches per View
V/C = constant (#Views / #Caches)	
I = C	(# pages = # Caches)

Theorem

There exists a family of hash function with the following properties

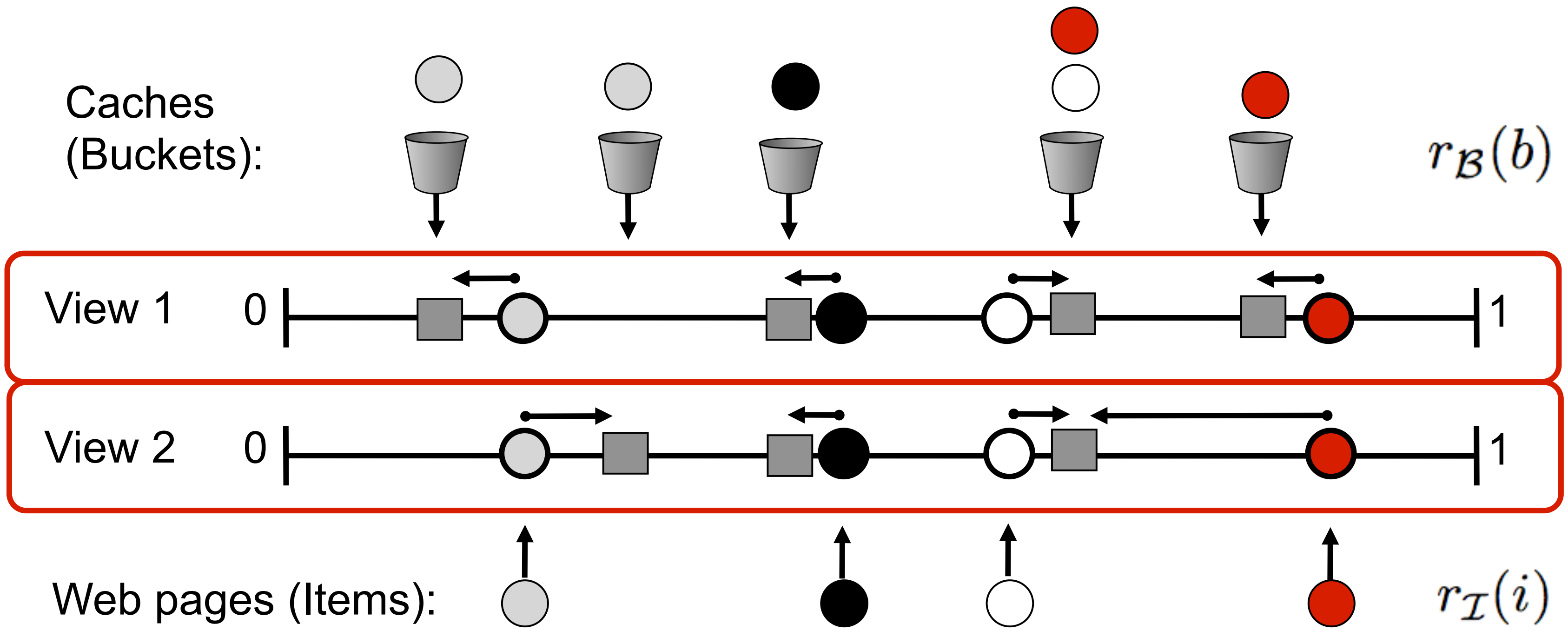
- Each function f∈F is **monotone**
- **Balance**: For every view $\Pr\left[f_{\mathcal{V}}(i) = b\right] \leq \frac{c}{|\mathcal{V}|}$
- **Spread**: For each page i $\sigma(i) = \mathcal{O}(t \log C)$

  with probability $1 - \frac{1}{C^{\Omega(1)}}$
- **Load:** For each cache b $\lambda(b) = \mathcal{O}(t \log C)$

  with probability $1 - \frac{1}{C^{\Omega(1)}}$

The Design

- 2 Hash functions onto the reals [0,1]
  - $r_{\mathcal{B}}(b)$ maps k log C copies of cache b randomly to [0,1]
  - $r_{\mathcal{I}}(i)$ maps web page i randomly to the interval [0,1]
- $f_{\mathcal{V}}(i)$:= Cache $b \in \mathcal{V}$ which minimizes $|r_{\mathcal{B}}(b) - r_{\mathcal{I}}(i)|$



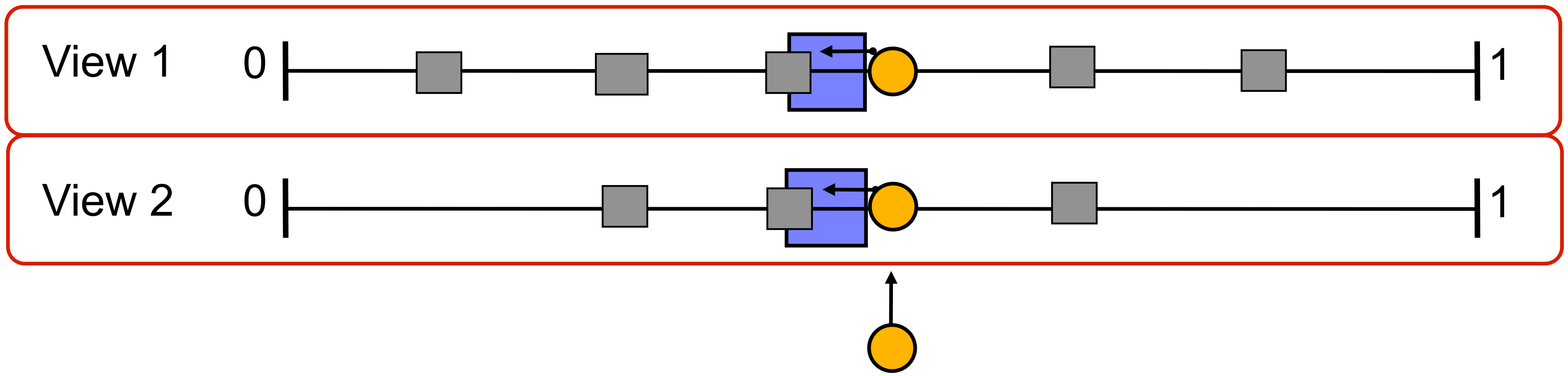
Monotony

- $f_{\mathcal{V}}(i)$:= Cache $b \in \mathcal{V}$ which minimizes $|r_{\mathcal{B}}(b) - r_{\mathcal{I}}(i)|$

For all $\mathcal{V}_1 \subseteq \mathcal{V}_2 \subseteq \mathcal{B}$:

$$f_{\mathcal{V}_2}(i) \in \mathcal{V}_1 \Rightarrow f_{\mathcal{V}_1}(i) = f_{\mathcal{V}_2}(i)$$

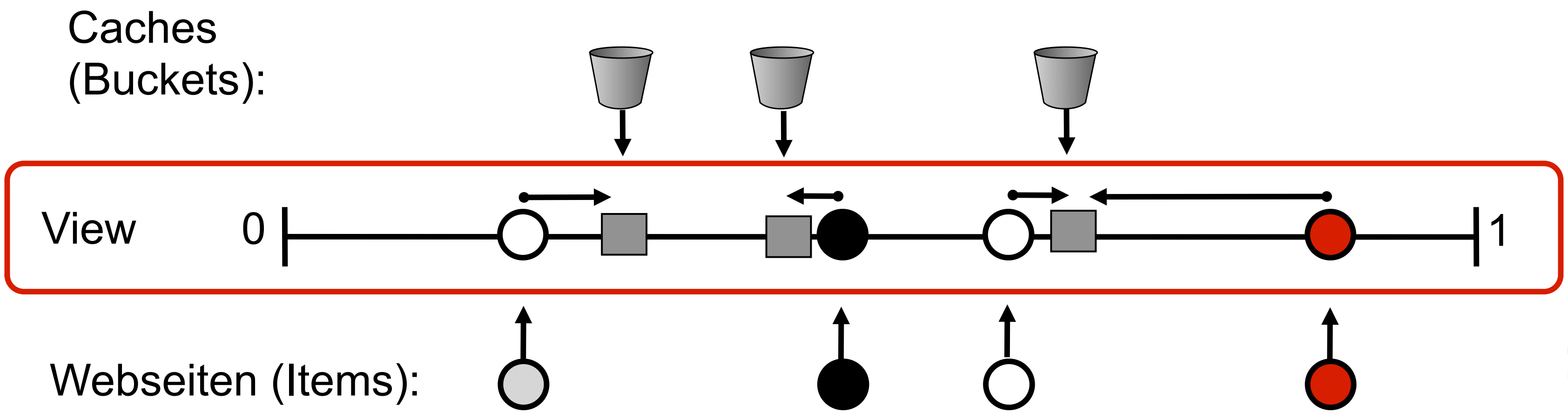
Observe: blue interval in V_2 and in V_1 empty!



2. Balance

Balance: For all views $\Pr\left[f_{\mathcal{V}}(i) = b\right] \leq \frac{c}{|\mathcal{V}|}$

- Choose fixed view and a web page i
- Apply hash functions $r_{\mathcal{B}}(b)$ and $r_{\mathcal{I}}(i)$.
- Under the assumption that the mapping is random
  - every cache is chosen with the same probability



3. Spread

σ(i) = number of all necessary copies (over all views)

$$\sigma(i) := |\{f_{\mathcal{V}_1}(i), f_{\mathcal{V}_2}(i), \ldots, f_{\mathcal{V}_V}(i)\}|$$

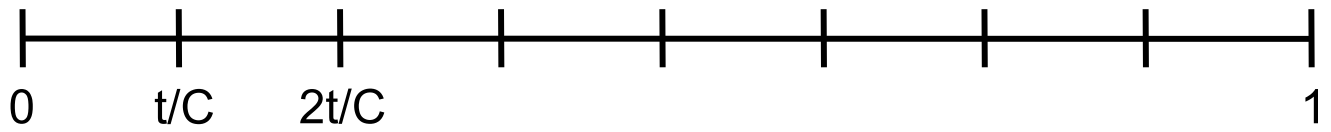
C	number of caches (Buckets)
C/t	minimum number of caches per View
V/C = constant (#Views / #Caches)	
I = C	(# pages = # Caches)

ever user knows at least a fraction of 1/t over the caches

For every page i $\sigma(i) = \mathcal{O}(t \log C)$ with prob. $1 - \frac{1}{C^{\Omega(1)}}$

Proof sketch:

- Every view has a cache in an interval of length t/C (with high probability)
- The number of caches gives an upper bound for the spread



4. Load

- Last (load): $\lambda(b)$ = Number of copies over all views

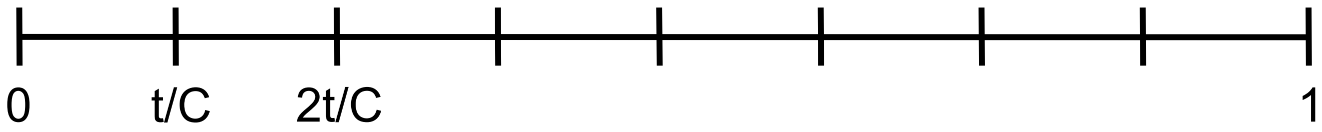
$$\lambda(b) := |\{ \cup_{\mathcal{V}} \; H_{\mathcal{V}}(b) \; \}|,$$

where $H_{\mathcal{V}}(b)$:= set of pages assigned to bucket b under view V

- For every cache be we observe $\lambda(b) = \mathcal{O}(t \log C)$ with probability $1 - \frac{1}{C^{\Omega(1)}}$

Proof sketch: Consider intervals of length t/C

- With high probability a cache of every view falls into one of these intervals
- The number of items in the interval gives an upper bound for the load



Summary

- Distributed Hash Table
  - is a distributed data structure for virtualization
  - with fair balance
  - provides dynamic behavior
- Standard data structure for dynamic distributed storages

DAAD Summerschool Curitiba 2011

Aspects of Large Scale High Speed Computing Building Blocks of a Cloud

Storage Networks

3: Distributed Hash Tables - Virtualization without Index Database

Christian Schindelhauer
Technical Faculty
Computer-Networks and Telematics
University of Freiburg