CoNe Freiburg

IIF INSTITUT FÜR INFORMATIK FREIBURG

Systeme II

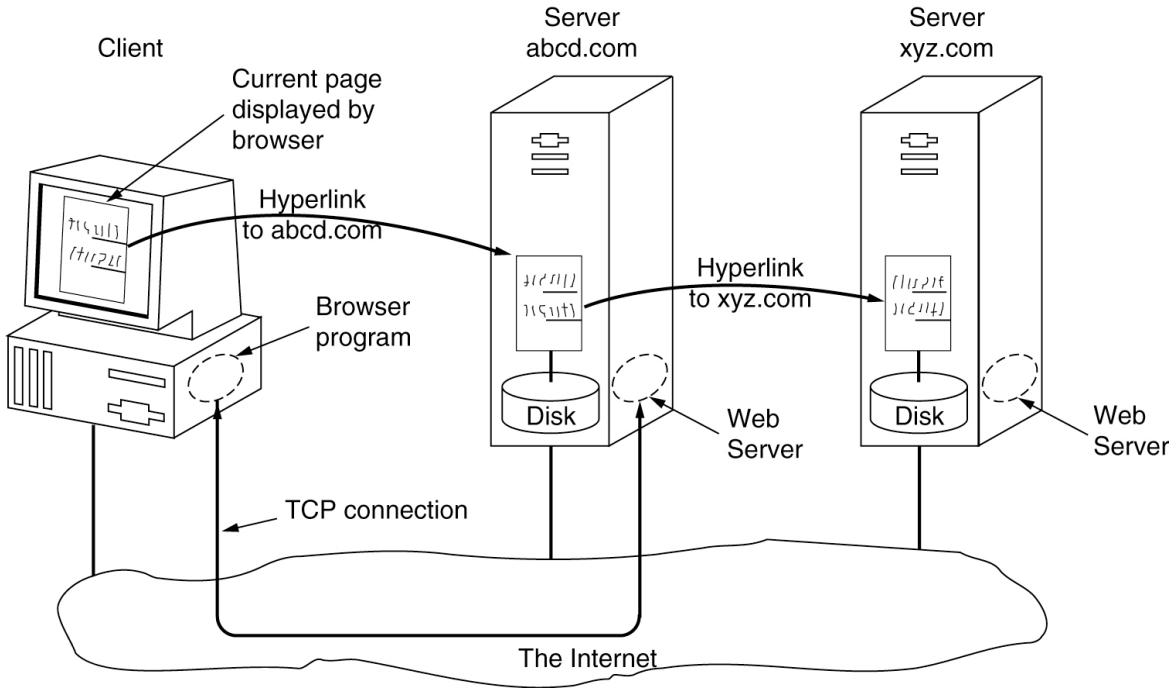
13. Woche Data Centers und Verteiltes Hashing

Christian Schindelhauer
Technische Fakultät
Rechnernetze und Telematik
Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

World Wide Web

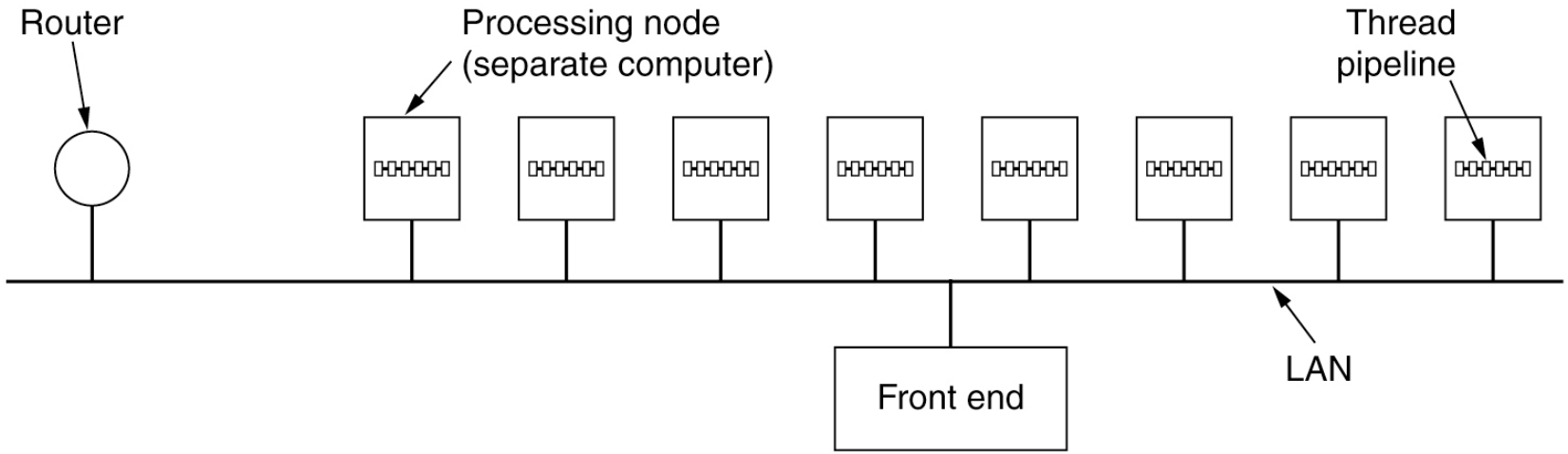
- Client-Server-Architektur
  - ***Web-Server*** stellt Web-Seiten bereit
    - Format: ***Hypertext Markup Language*** (HTML)
  - ***Web-Browser*** fragen Seiten vom Server ab
    - Server und Browser kommunizieren mittels ***Hypertext Transfer Protocol*** (HTTP)



CoNe Freiburg

Server-Farm

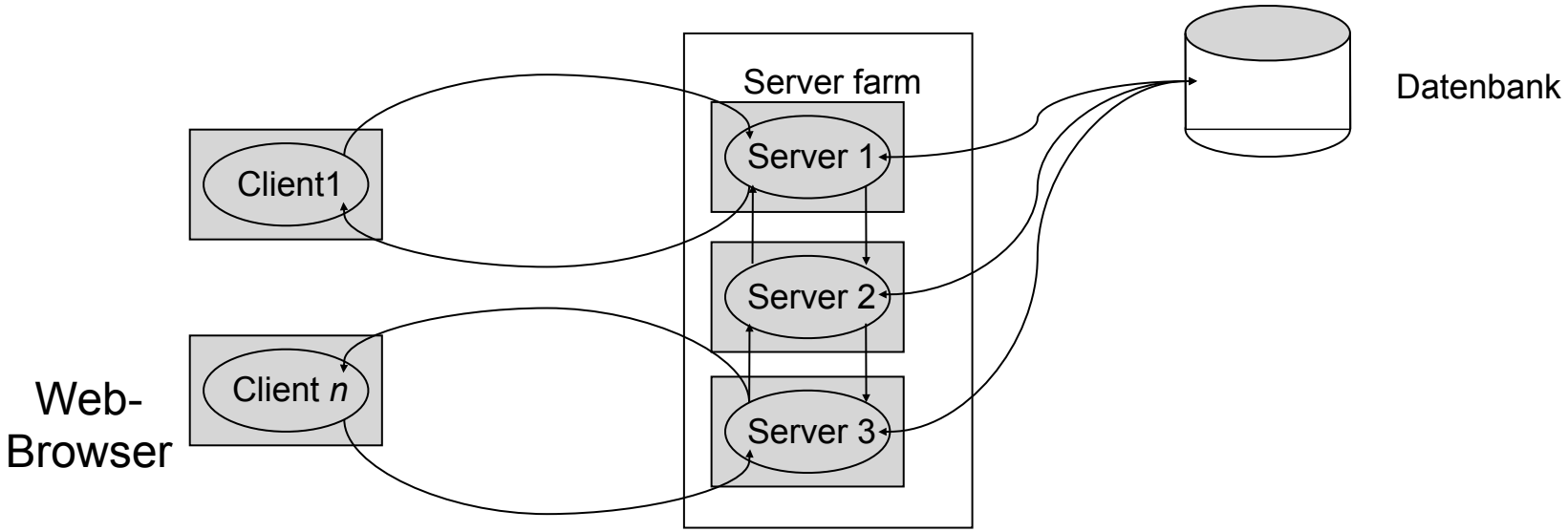
- Um die Leistungsfähigkeit auf der Server-Seite zu erhöhen
  - wird eine Reihe von Web-Server eingesetzt
- Front end
  - nimmt Anfragen an
  - reicht sie an separaten Host zur Weiterbearbeitung weiter



UNI FREIBURG

Web-Servers und Datenbanken

- Web-Server stellen nicht nur statische Web-Seiten zur Verfügung
  - Web-Seiten werden auch automatisch erzeugt
  - Hierzu wird auf eine Datenbank zurückgegriffen
  - Diese ist nicht statisch und kann durch Interatkionen verändert werden
- Problem:
  - Konsistenz
- Lösung
  - Web-Service und Daten-Bank in einer 3-Stufen-Architektur



CoNe Freiburg

Web-Cache

- Trotz Server-Farm ist die Latenzzeit häufig kritisch
- Lösung:
  - Cache (Proxy)
- Ort
  - Beim Client
  - Im lokalen Netzwerk (bei einem Proxy)
  - Beim Internet-Service-Provider
- Fragen
  - Plazierung, Größe, Aktualität
  - Entwertung durch Timeout

UNI FREIBURG

CoNe Freiburg

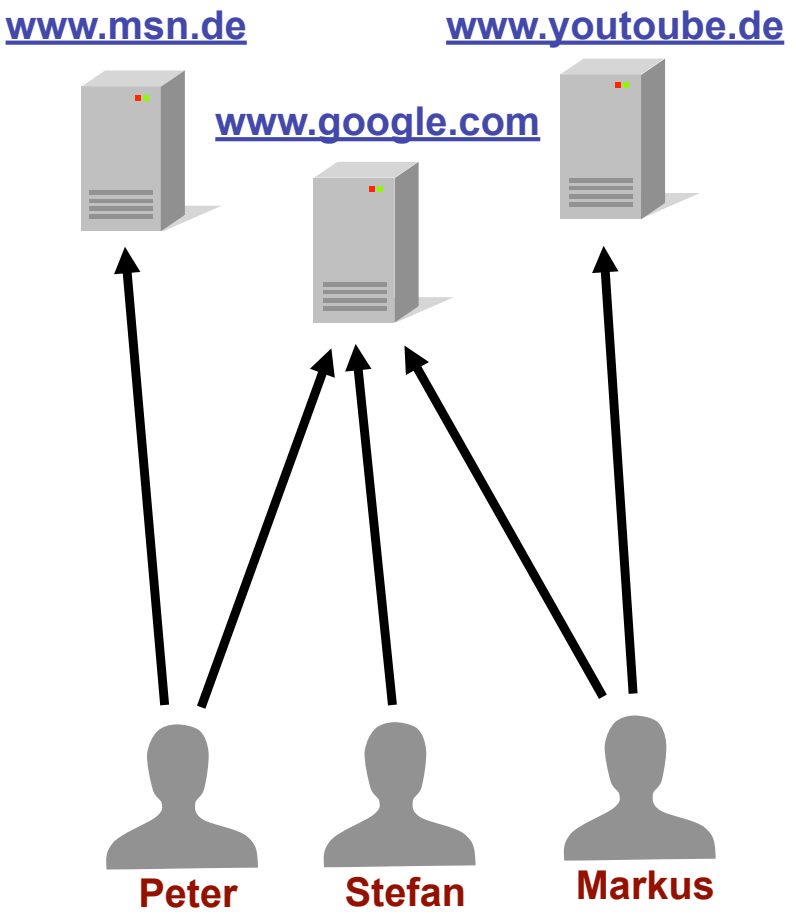
Content Distribution Networks (CDN)

- Eine koordinierte Menge von Caches
  - Die Last großer Web-Sites wird verteilt auf global verteilte Server-Farmen
  - Diese übernehmen Web-Seiten möglichst verschiedener Organisationen
    - z.B. News, Software-Hersteller, Regierungen
  - Beispiele: Akamai, Digital Island
  - Cache-Anfragen werden auf die regional und lastmäßig bestgeeigneten Server umgeleitet
- Beispiel Akamai:
  - Durch verteilte Hash-Tabellen ist die Verteilung effizient und lokal möglich

UNI FREIBURG

WWW-Lastbalancierung

- Für Surfen im Web typisch:
  - Web-Server bieten Web-Seiten an
  - Web-Clients fordern Web-Seiten an
- In der Regel sind diese Mengen disjunkt
- Eingehende Anforderungen belasten Web-Server hinsichtlich:
  - Übertragungsbandbreite
  - Rechenaufwand (Zeit,Speicher)

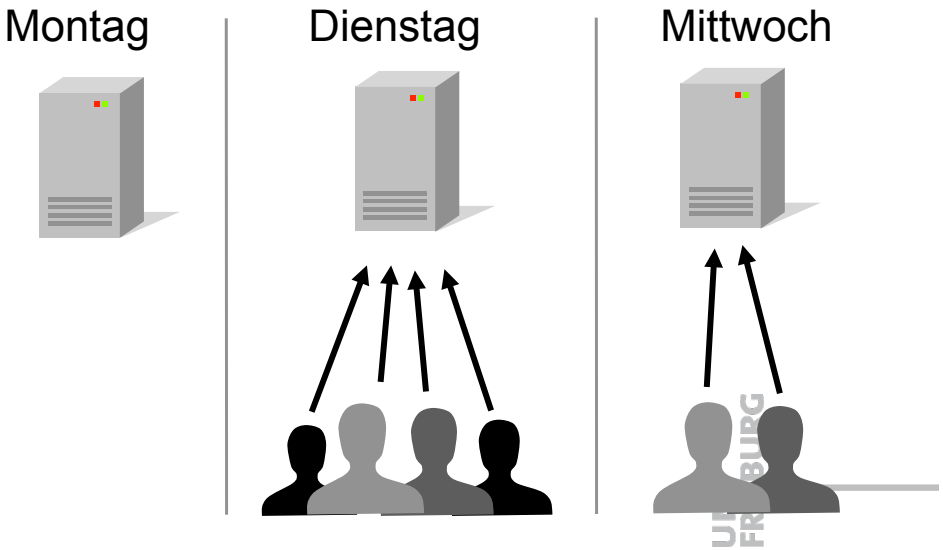


CoNe Freiburg

Lastanforderungen

- Einige Web-Server haben immer hohe Lastanforderungen
  - Z.B. Nachrichten-Sites, Suchmaschinen, Web-verzeichnisse
  - Für permanente Anforderungen müssen Server entsprechen ausgelegt werden

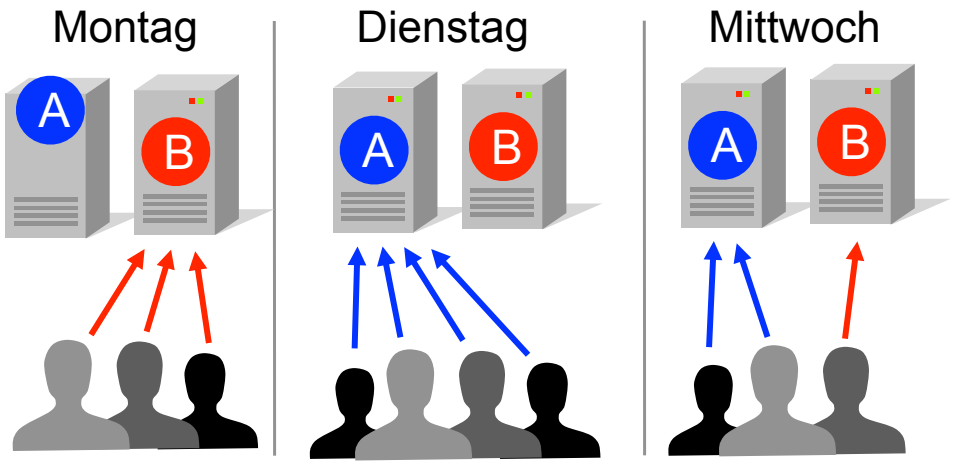
- Andere leiden unter hohen Fluktuationen
  - z. B. bei besonderen Ereignissen:
    - fifa.com (Fussball-WM)
    - t-mobile.de (iPhone 4 Einführung)
  - Server-Erweiterung nicht sinnvoll
  - Bedienung der Anfragen aber erwünscht



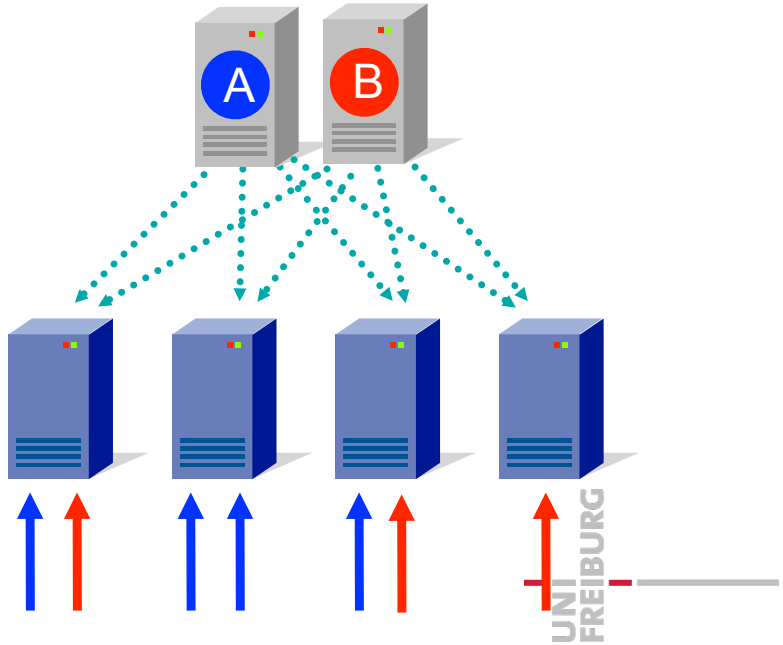
CoNe Freiburg

Lastbalancierung im WWW

- Fluktuationen betreffen meistens einzelne Server



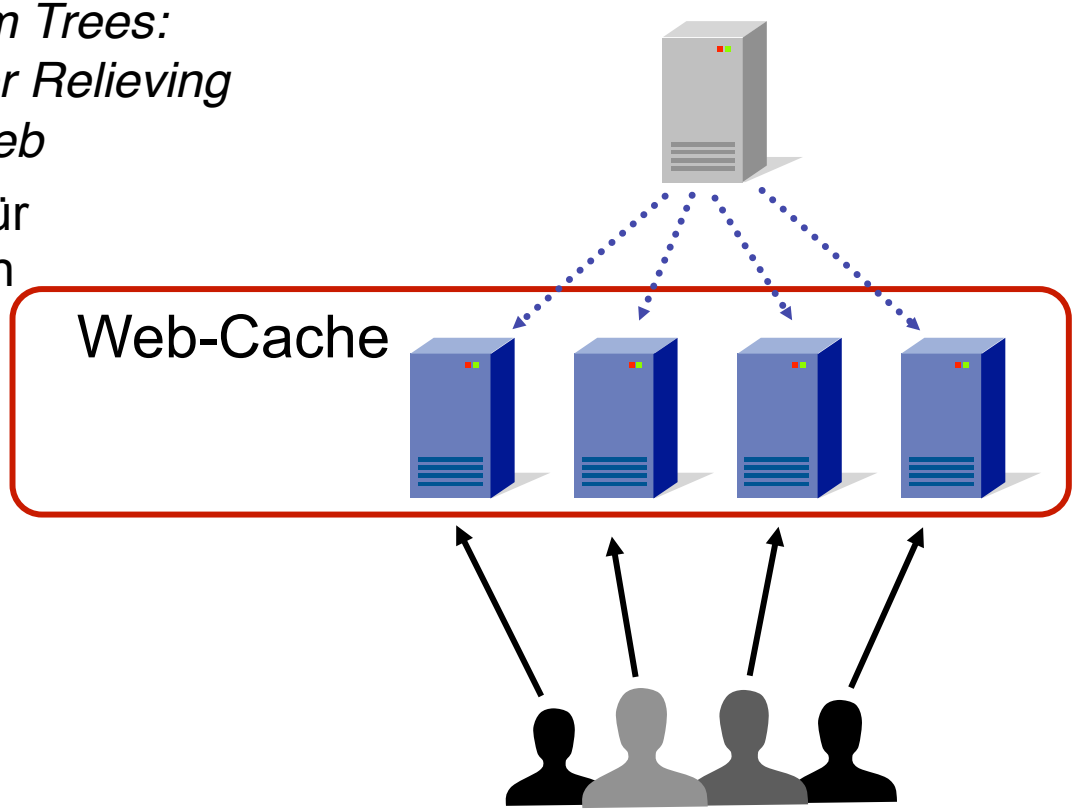
- (Kommerzielle) Lösung
  - Dienstleister bieten Ausweich-(Cache-)Server an
  - Viele Anforderungen werden auf diese Server verteilt
- Aber wie?



CoNe Freiburg

Web-Caching

- Leighton, Lewin, et al. STOC 97
  - *Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web*
  - Passen bestehende Verfahren für dynamische Hash-Funktionen an WWW-Anforderungen an



- Leighton und Lewin (MIT) gründen Akamai 1997

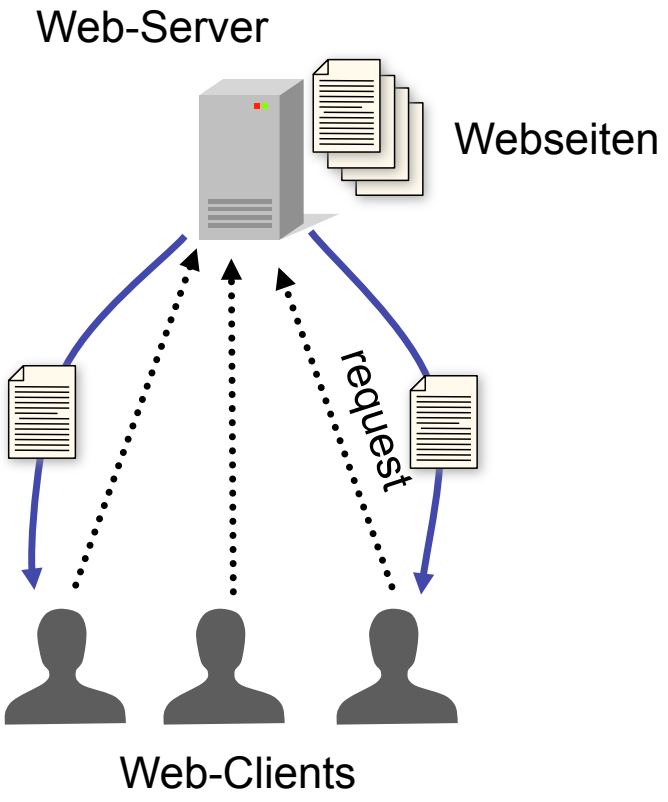
- Akaimai 2003:
  - 550 Angestellte
  - Ertrag 145 Mio. $ (2002)
  - 15.000 Server in 60 Ländern verbunden mit 1.100 lokalen Netzwerken

UNI FREIBURG

CoNe Freiburg

Ausgangssituation

- Ohne Lastbalancierung:
  - Jeder Browser (Web-Client) belegt einen Web-Server für eine Web-Site
- Vorteil:
  - Einfach
- Nachteil:
  - Der Server muss immer für den Worst-Case ausgelegt werden

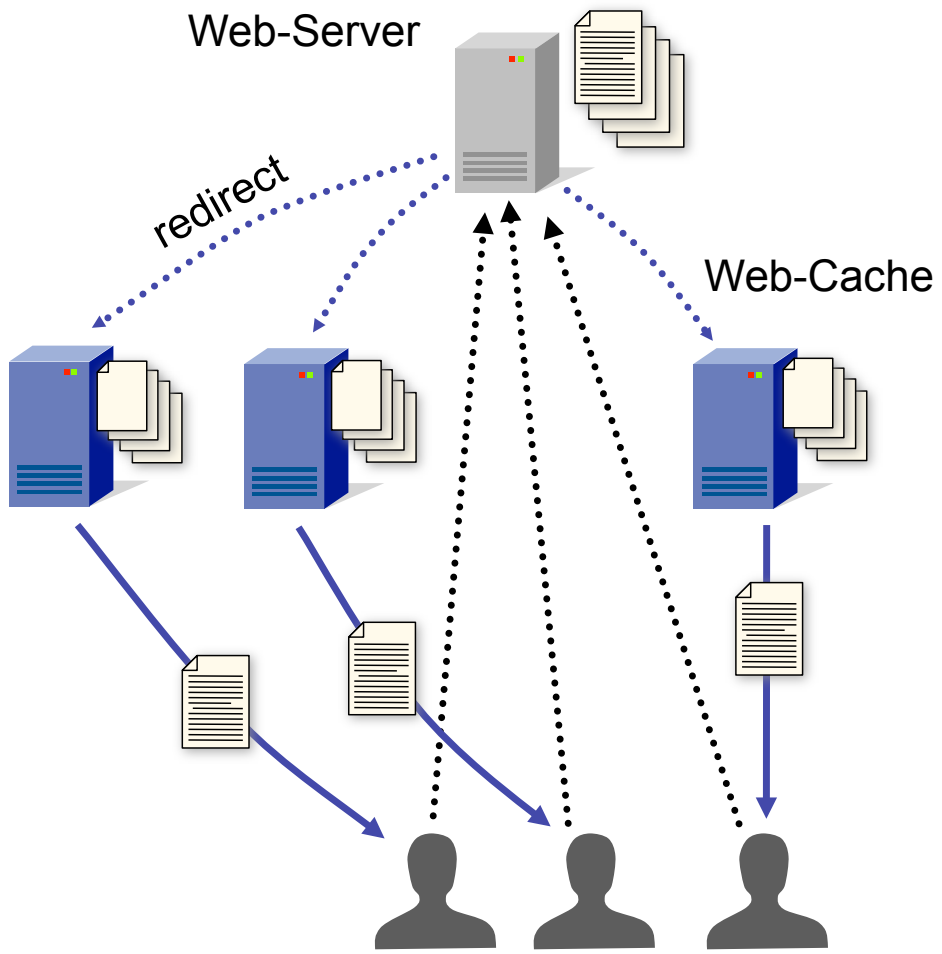


UNI FREIBURG

CoNe Freiburg

Site Caching

- Ganze Web-Site wird auf verschiedene Web-Caches kopiert
- Browser fragt bei Web-Server nach Seite
- Web-Server leitet Anfrage auf Web-Cache um (redirect)
- Web-Cache liefert Web-Seite aus
- Vorteil:
  - Gute Lastbalancierung für Seitenverteilung
- Nachteil:
  - Bottleneck: Redirect
  - Großer Overhead durch vollständige Web-Site-Replikationen

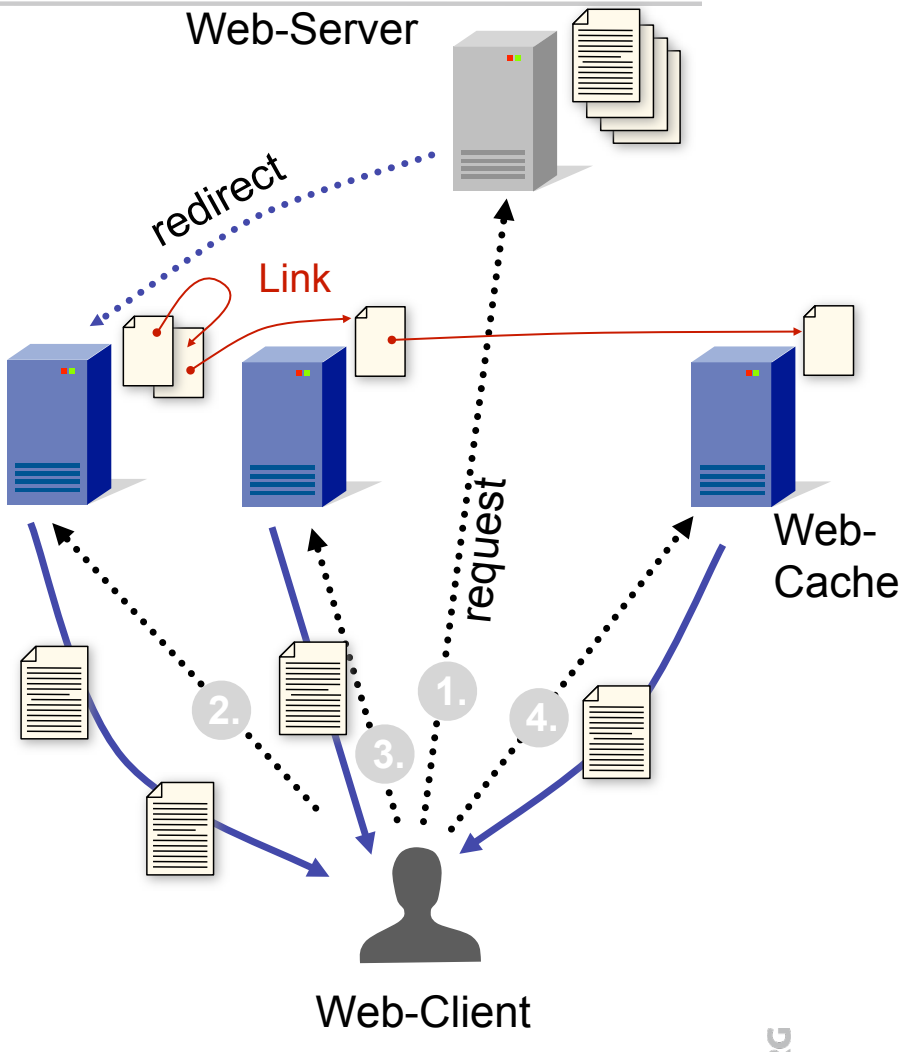


UNI FREIBURG

CoNe Freiburg

Proxy Caching

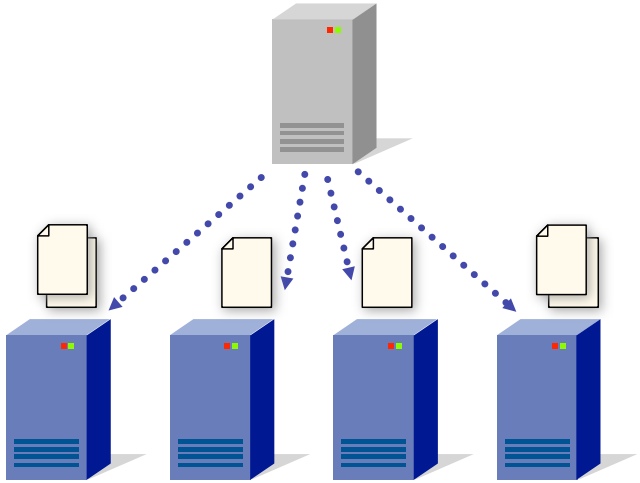
- Jede Web-Seite wird auf einige (wenige) Web-Caches verteilt
- Nur Startanfrage erreicht Web-Server
- Links referenzieren auf Seiten im Web-Cache
- Dann surft der Web-Client nur noch auf den Web-Cache
- Vorteil:
  - Kein Bottleneck
- Nachteil:
  - Lastbalancierung nur implizit möglich
  - Hohe Anforderung an Caching-Algorithmus



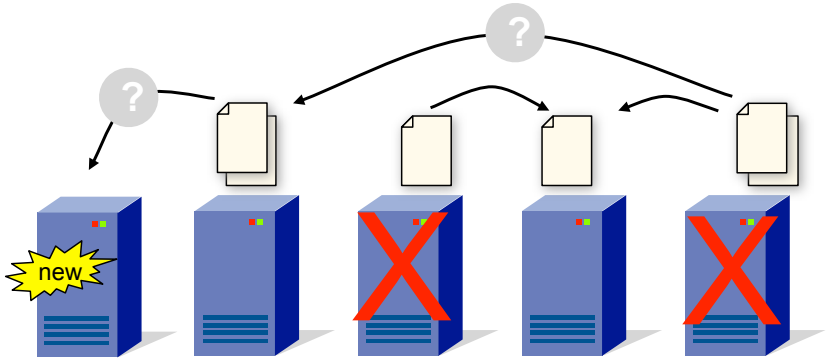
UNI FREIBURG

Anforderungen an Caching-Algorithmus

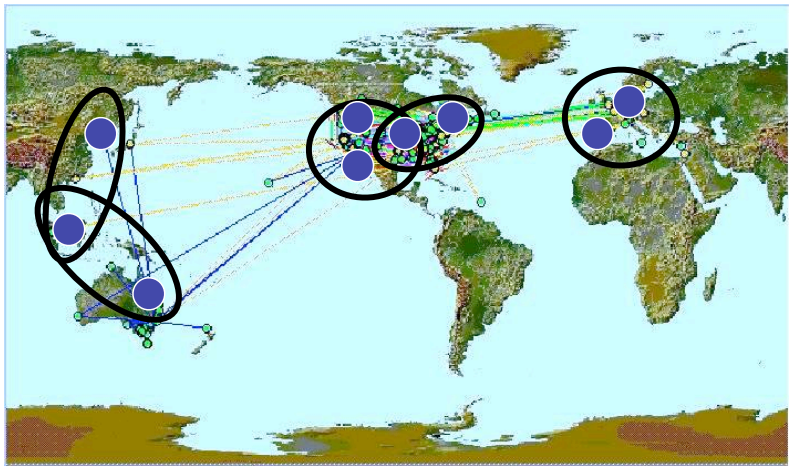
Balance
Gleichmäßige Verteilung der Seiten



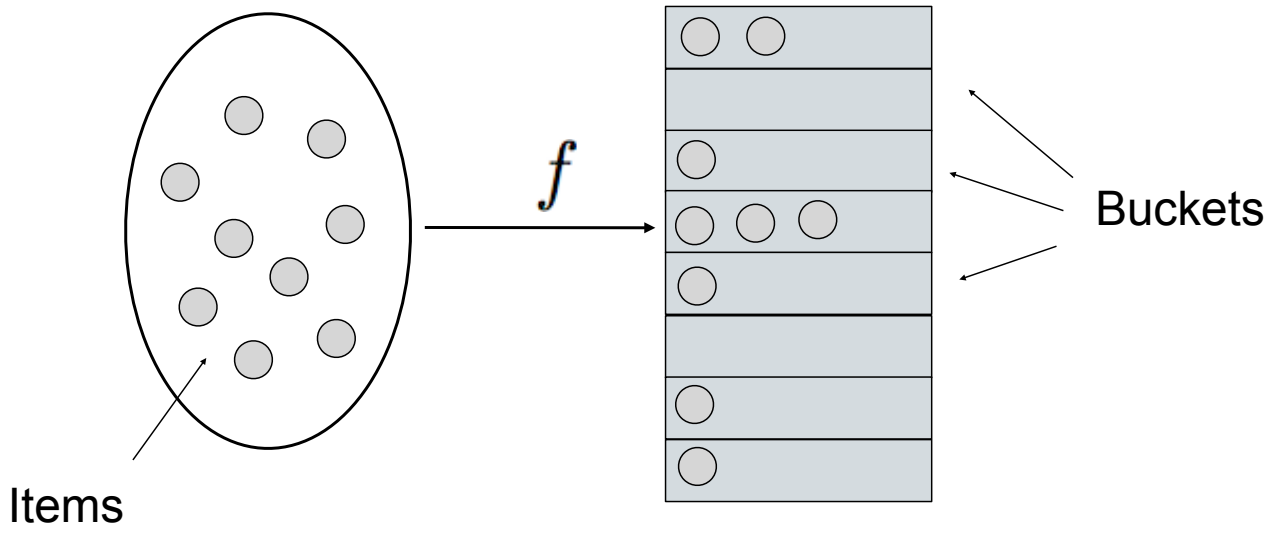
Dynamik
Effizientes Einfügen/Löschen von neuen Web-Cache-Servern



Views
Web-Clients „sehen“ unterschiedliche Menge von Web-Caches



Hash-Funktionen



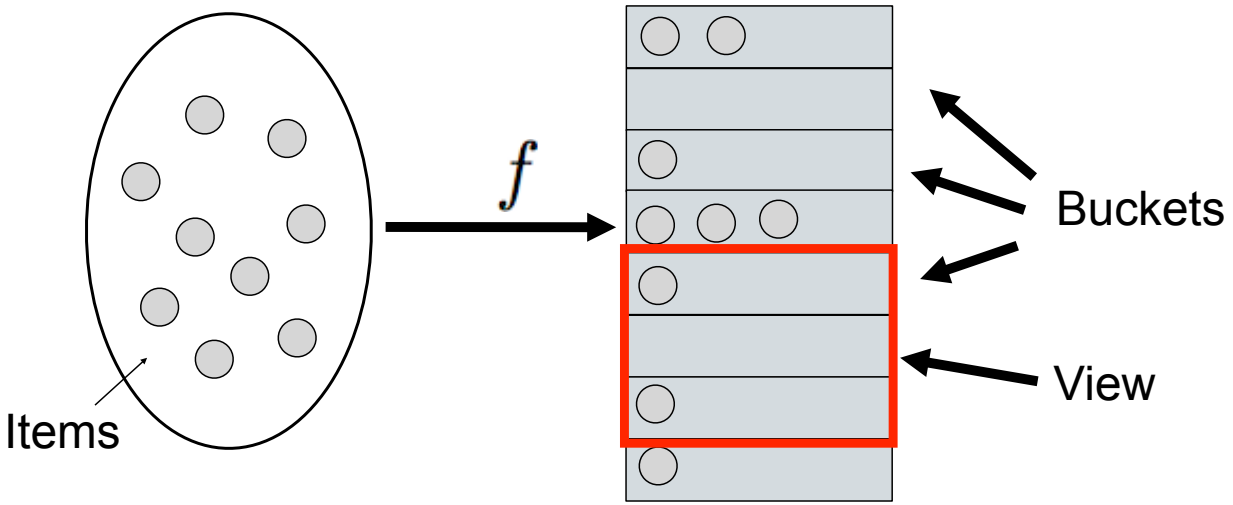
Menge der Items: $\mathcal{I}$

Menge der Buckets: $\mathcal{B}$

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

Ranged Hash-Funktionen oder Verteilte Hash-Funktionen

- Gegeben:
  - Elemente (Items) $\mathcal{I}$, Anzahl $I := |\mathcal{I}|$
  - Caches (Buckets), Menge der Buckets: $\mathcal{B}$
  - Views $\mathcal{V} \subseteq 2^{\mathcal{B}}$
- Ranged Hash-Funktion:
  - $f : 2^{\mathcal{B}} \times \mathcal{I} \rightarrow \mathcal{B}$
  - Voraussetzung: für alle Views gilt $f_{\mathcal{V}}(\mathcal{I}) \subseteq \mathcal{V}$



CoNe Freiburg

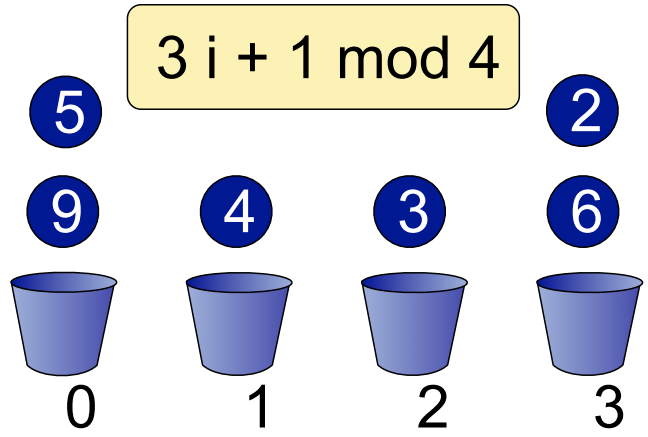
Erste Idee: Hash-Funktion

- Verfahren:
  - Wähle Hash-Funktion, z.B.

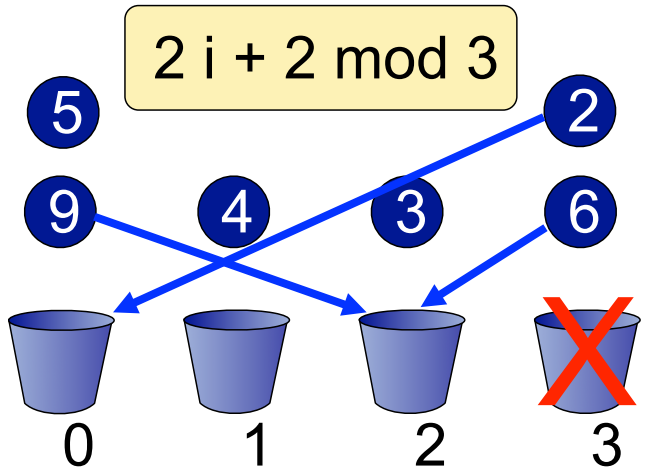
$$f(i) = ai + b \bmod n$$

n: Anzahl Cache-Server

- Balance:
  - Sehr gut!



- Dynamik
  - Einfügen/Löschen von nur einem Cache-Server
  - Neue Hash-Funktion und vollständige Neuzuweisung
  - Hoher Aufwand!



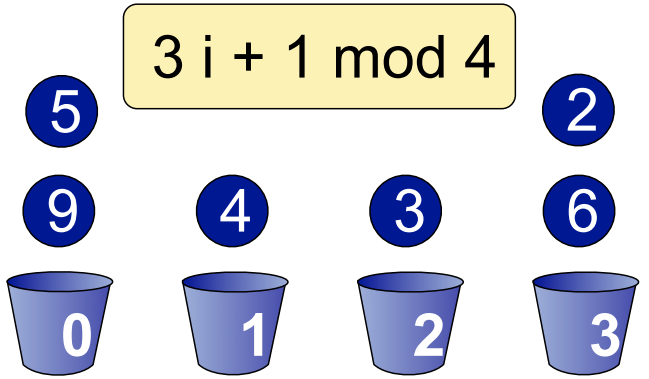
UNI FREIBURG

Erste Idee: Hash-Funktion (mit Views)

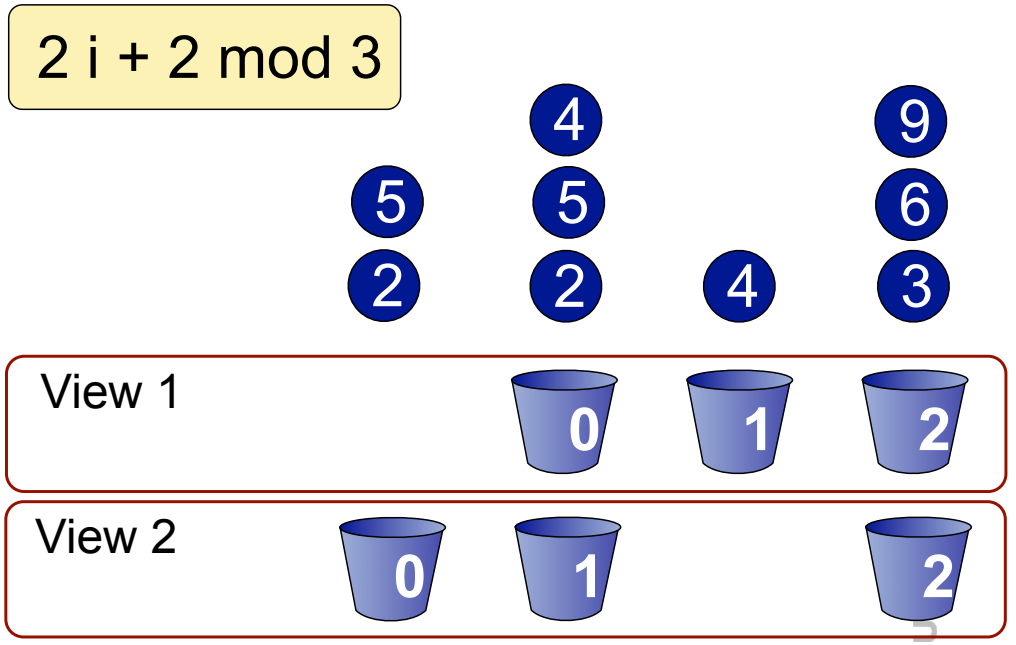
- Verfahren:
  - Wähle Hash-Funktion, z.B.

$$r(i) = ax + b \bmod n$$

n: Anzahl Cache-Server



- Views
  - Verschiedene Nummerierungen der Web-Cache notwendig
  - Anzahl der Duplikate proportional zu der Anzahl der Views



CoNe Freiburg

Anforderungen an Ranged Hash-Funktionen

- Monotonie
  - nach dem Hinzufügen neuer Caches (Buckets) sollten keine Seiten (Items) zwischen alten Caches verschoben werden
- Balance
  - Alle Caches sollten gleichmäßig ausgelastet werden
- Spread (Verbreitung,Streuung)
  - Eine Seite sollte auf eine beschränkte Anzahl von Caches verteilt werden
- Load
  - Kein Cache sollte wesentlich mehr als die durchschnittliche Anzahl von Seiten enthalten

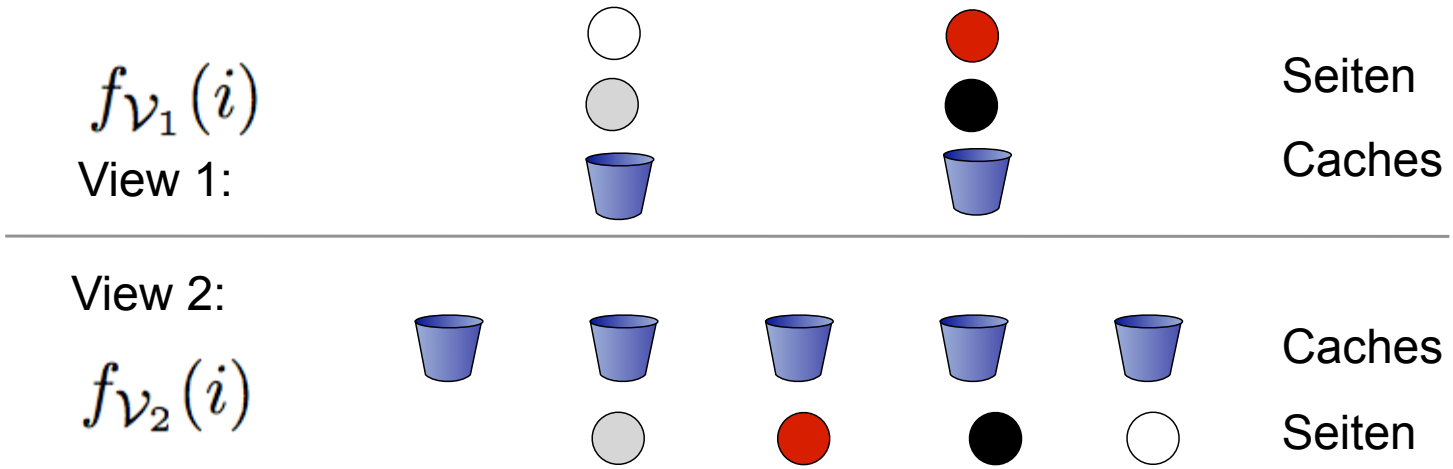
UNI FREIBURG

CoNe Freiburg

1. Monotonie

- Seiten, die im umfassenderen View einem Cache zugewiesen sind, werden nicht umorganisiert
- d.h. nach dem Hinzufügen neuer Buckets dürfen (alte) Seiten nur in neue Buckets verschoben werden
- Formal: Für alle $\mathcal{V}_1 \subseteq \mathcal{V}_2 \subseteq \mathcal{B}$ gilt:

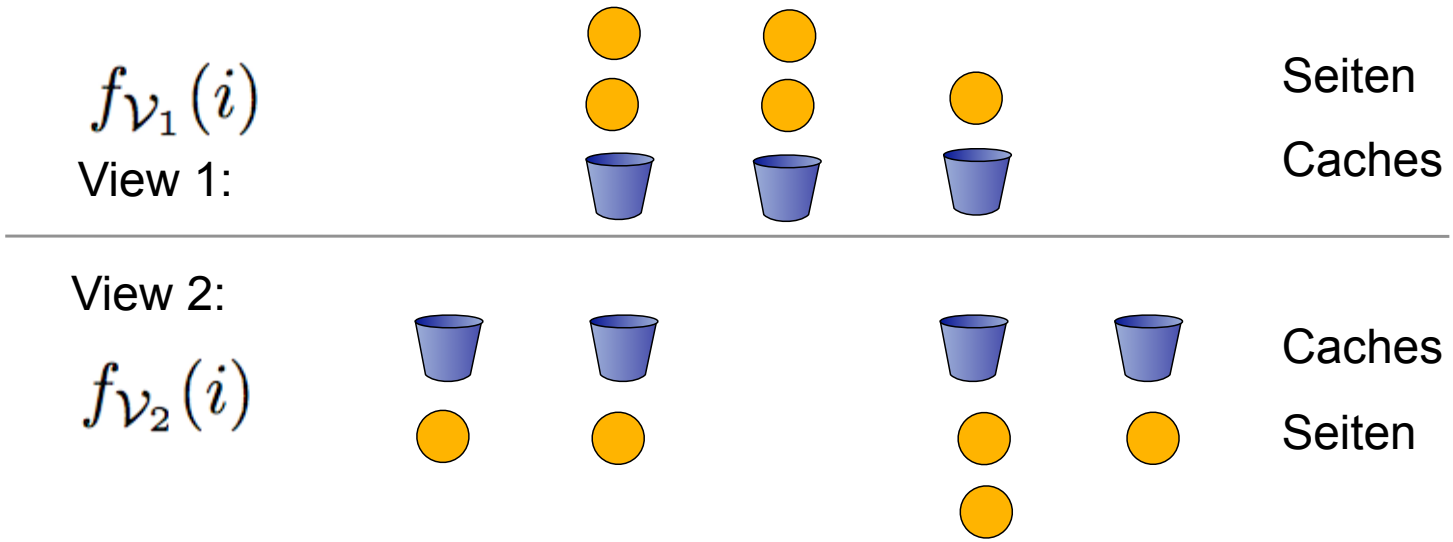
$$f_{\mathcal{V}_2}(i) \in \mathcal{V}_1 \;\Rightarrow\; f_{\mathcal{V}_1}(i) = f_{\mathcal{V}_2}(i)$$



2. Balance

- Für jeden View V ist die Hash-Funktion fV(i) balanciert
- Für eine Konstant c und alle $\mathcal{V} \subseteq \mathcal{B}$ gilt:

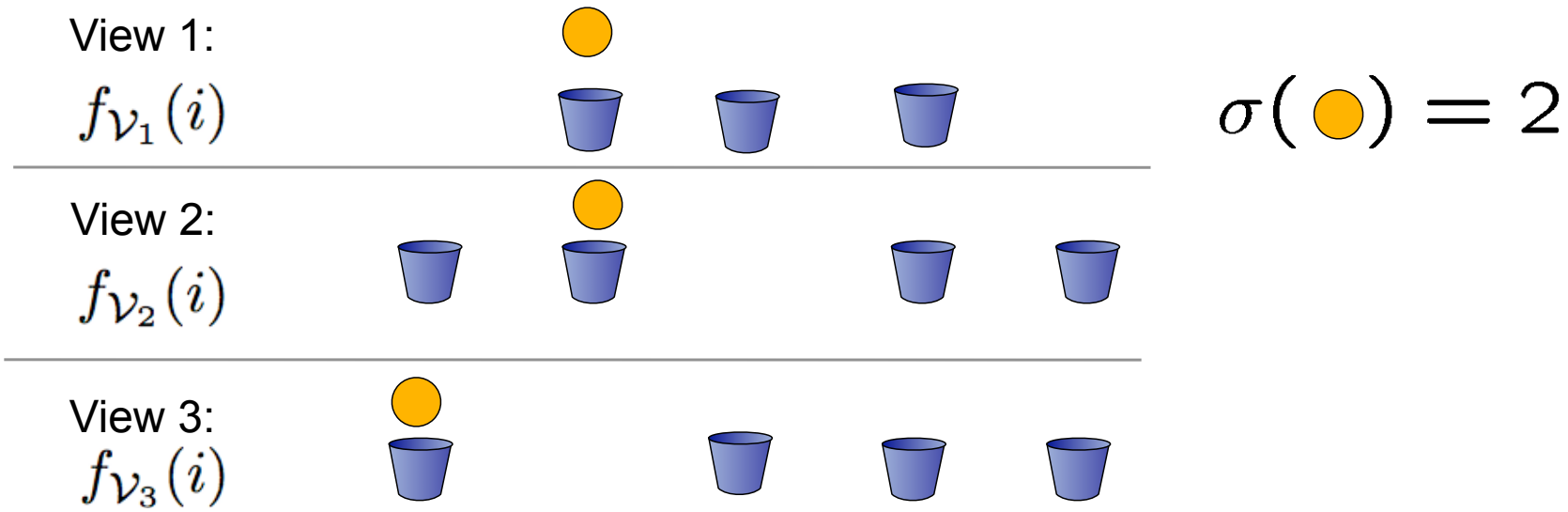
$$\Pr\left[f_{\mathcal{V}}(i) = b\right] \leq \frac{c}{|\mathcal{V}|}$$



3. Spread

- Die Verbreitung σ(i) (spread) einer Seite i ist die Gesamtanzahl aller notwendigen Kopien (über alle Views)

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

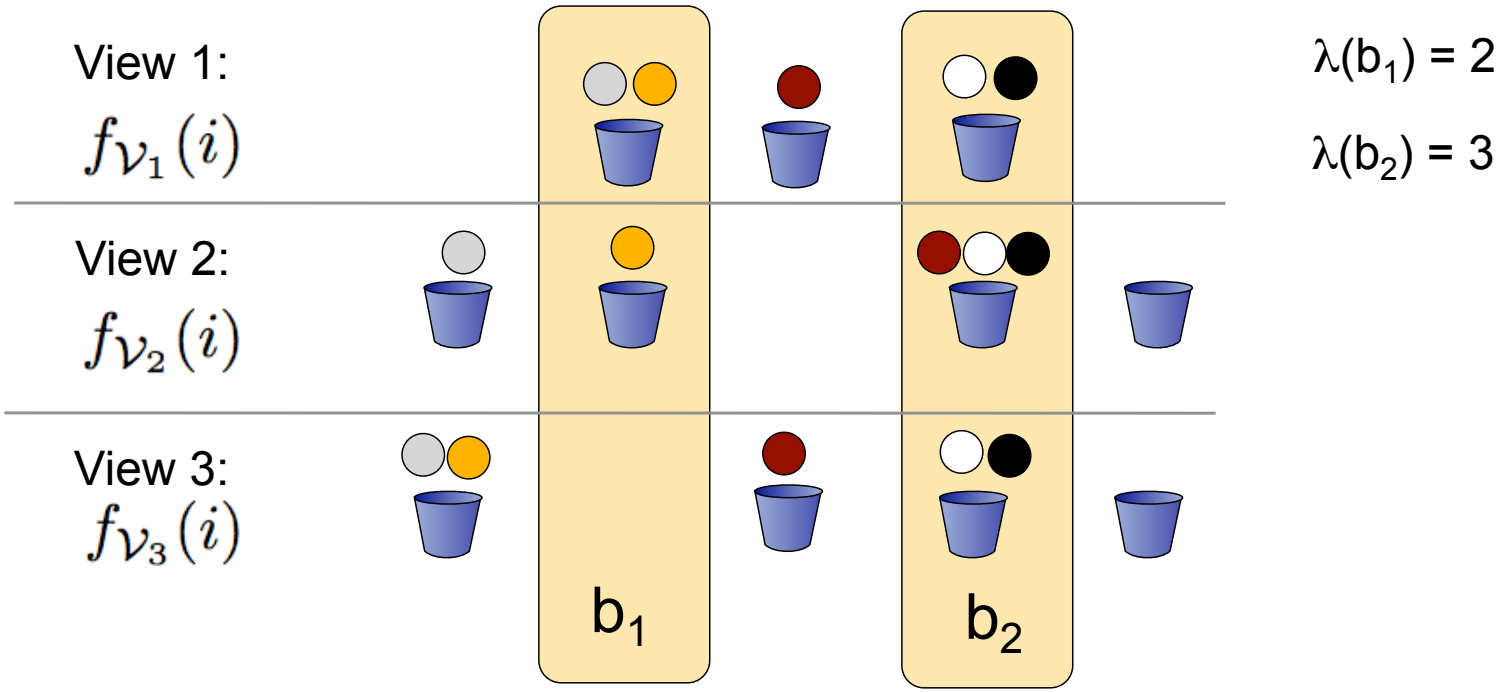


4. Load

- Die Last λ(b) (load) eines Caches b ist die Gesamtanzahl aller notwendigen Kopien (über alle Views)

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

wobei $H_{\mathcal{V}}(b)$:= Menge aller Seiten, die Bucket b zugewiesen werden (in View V)



CoNe Freiburg

Familien von Hash-Funktionen

- Für jede Hash-Funktion existiert eine Worst-Case-Eingabe
  - Daher betrachtet man grundsätzlich Familien von Hash-Funktionen
  - Genauso definieren wir Familie von Ranged-Hash-Funktionen für geg. Views und Caches

- Wir gehen im folgenden davon aus, dass eine Hash-Funktion sich verhält wie ein perfektes Zufallsereignis
  - Gleichwahrscheinlich
  - Unabhängig

- Die Elemente werden wie Bälle in Körbe verteilt.

UNI FREIBURG

CoNe Freiburg

Distributed Hash Tables (DHT)
Verteilte Hash-Funktionen

Theorem

Es gibt eine Familie von Ranged Hash-Funktionen F mit den folgenden Eigenschaften:

C	Anzahl aller Caches (Buckets)
C/t	Mindestanzahl Caches pro View
V/C = konstant	(#Views / #Caches)
I = C	(# Seiten = # Caches)

- Jede Funktion f∈F ist **monoton**
- **Balance**: Für jeden View gilt $\Pr\left[f_{\mathcal{V}}(i) = b\right] \leq \frac{c}{|\mathcal{V}|}$
- **Spread**: Für jede Seite i is $\sigma(i) = \mathcal{O}(t \log C)$

  mit W'keit $1 - \frac{1}{C^{\Omega(1)}}$
- **Load:** Für jeden Cache b ist $\lambda(b) = \mathcal{O}(t \log C)$

  mit W'keit $1 - \frac{1}{C^{\Omega(1)}}$

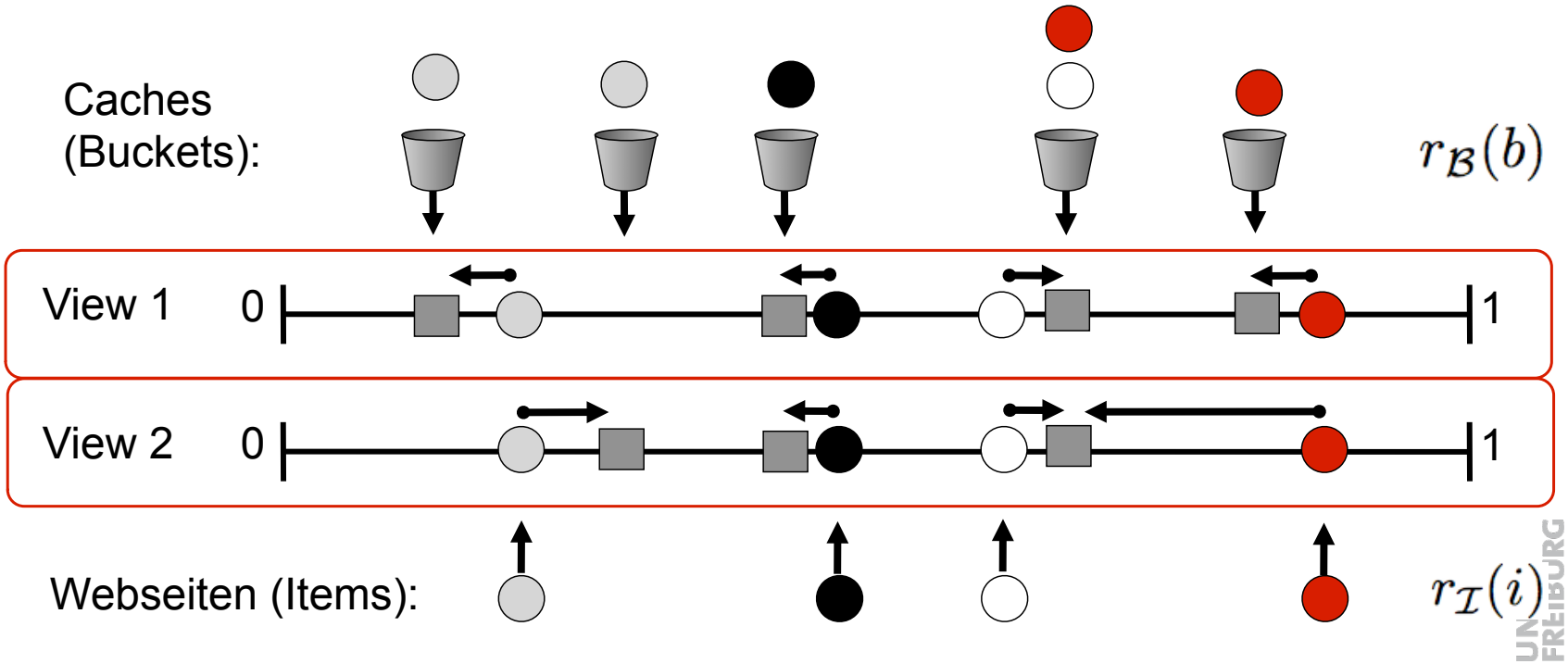
UNI FREIBURG

CoNe Freiburg

Die Konstruktion

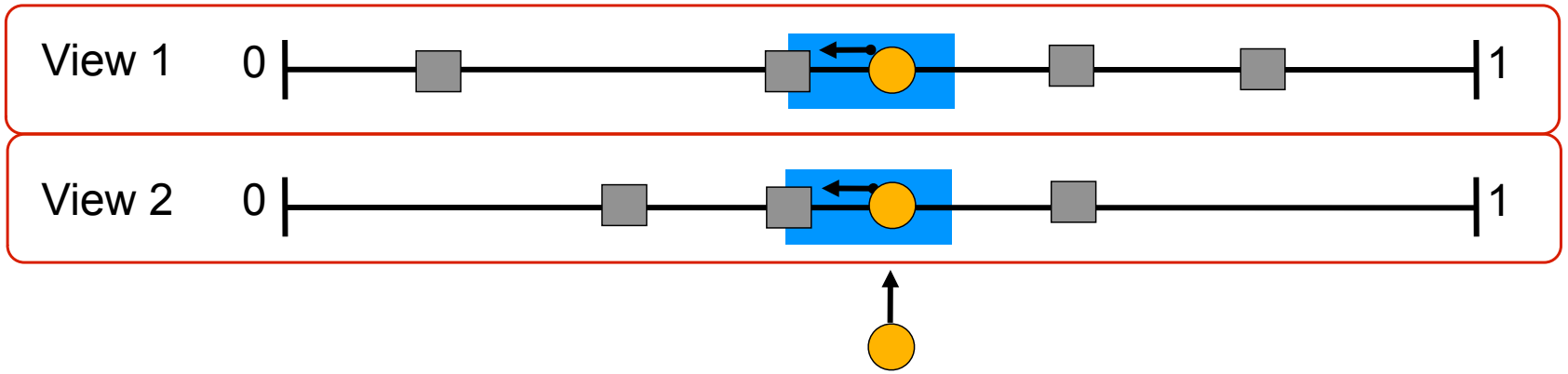
- 2 Hash-Funktionen auf das reelle Intervall [0,1]
  - $r_{\mathcal{B}}(b)$ bildet k log C Kopien des Caches b zufällig auf [0,1] ab
  - $r_{\mathcal{I}}(i)$ bildet Web-Seite i zufällig auf Intervall [0,1] ab

$f_{\mathcal{V}}(i)$ Cache $b \in \mathcal{V}$, der den Abstand $|r_{\mathcal{B}}(b) - r_{\mathcal{I}}(i)|$ minimiert.



1. Monotonie

- $f_{\mathcal{V}}(i)$:= Cache$b \in \mathcal{V}$, der den Abstand $|r_{\mathcal{B}}(b) - r_{\mathcal{I}}(i)|$ ninimiert.
- Für alle $\mathcal{V}_1 \subseteq \mathcal{V}_2 \subseteq \mathcal{B}$ gilt: $f_{\mathcal{V}_2}(i) \in \mathcal{V}_1 \Rightarrow f_{\mathcal{V}_1}(i) = f_{\mathcal{V}_2}(i)$
- Beobachtung: Blaues Intervall sowohl in V_2 als auch in V_1 leer!

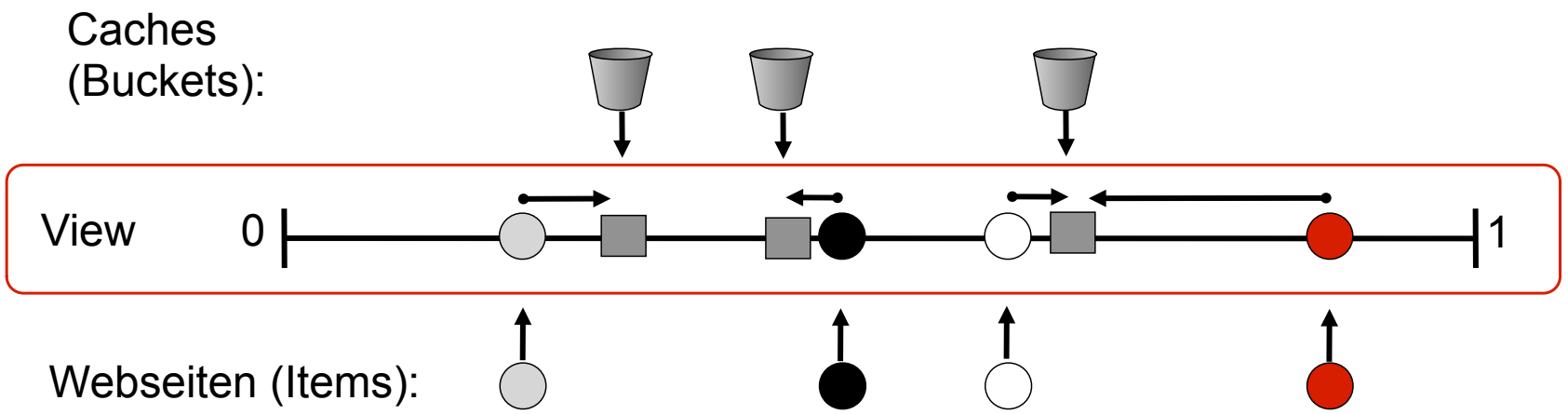


CoNe Freiburg

2. Balance

- Balance: Für jeden View gilt $\Pr[f_{\mathcal{V}}(i) = b] \leq \frac{c}{|\mathcal{V}|}$

- Wähle festen View und eine Web-Seite i
- Wende nun die Hash-Funktionen $r_{\mathcal{B}}(b)$ und $r_{\mathcal{I}}(i)$ an.
- Unter der Annahme, dass diese sich wie zufällige Abbildungen verhalten,
  - wird jeder Cache mit der gleichen Wahrscheinlichkeit ausgewählt.



CoNe Freiburg

3. Spread (I)

σ(i) = Gesamtanzahl aller notwendigen Kopien (über alle Views)

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

C	Anzahl aller Caches (Buckets)
C/t	Mindestanzahl Caches pro View
V/C = konstant	(#Views / #Caches)
I = C	(# Seiten = # Caches)

← jeder Server kennt mindestens einen Anteil von 1/t der Caches

Für jede Seite i ist $\sigma(i) = \mathcal{O}(t \log C)$ mit W'keit $1 - \frac{1}{C^{\Omega(1)}}$

UNI FREIBURG

CoNe Freiburg

4. Load (I)

- Last (load): λ(b) = Gesamtanzahl aller notwendigen Kopien (über alle Views)

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

wobei $H_{\mathcal{V}}(b)$:= Menge aller Seiten, die Bucket b zugewiesen werden (in View V)

- Für jeden Cache b ist $\lambda(b) = \mathcal{O}(t \log C)$

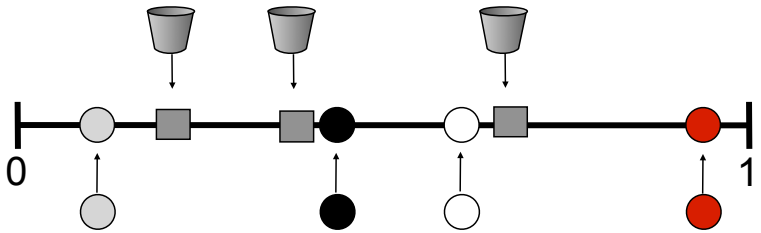
mit W'keit $1 - \frac{1}{C^{\Omega(1)}}$

UNI FREIBURG

CoNe Freiburg

Zusammenfassung

- Web-Caching durch konsistentes Hashing (verteilte Hash-Tabellen)
  - Seiten und Caches werden auf das Einheitsintervall abgebildet
  - Zuordnung durch minimalen Abstand



- Die Funktionen besitzen folgende Eigenschaften:
  - Jede Funktion aus dieser Familie ist monoton
  - Balance: Für jeden View gilt $\Pr[f_{\mathcal{V}}(i) = b] \leq \frac{c}{|\mathcal{V}|}$
  - Spread: Für jede Seite i ist $\sigma(i) = \mathcal{O}(t \log C)$ mit W'keit $1 - C^{-\Omega(1)}$
  - Load: Für jeden Cache b ist $\lambda(b) = \mathcal{O}(t \log C)$ mit $1 - C^{-\Omega(1)}$ W'keit

UNI FREIBURG

CoNe Freiburg

IIF INSTITUT FÜR INFORMATIK FREIBURG

Systeme II

13. Woche Data Centers und Verteiltes Hashing

Christian Schindelhauer
Technische Fakultät
Rechnernetze und Telematik
Albert-Ludwigs-Universität Freiburg

UNI FREIBURG