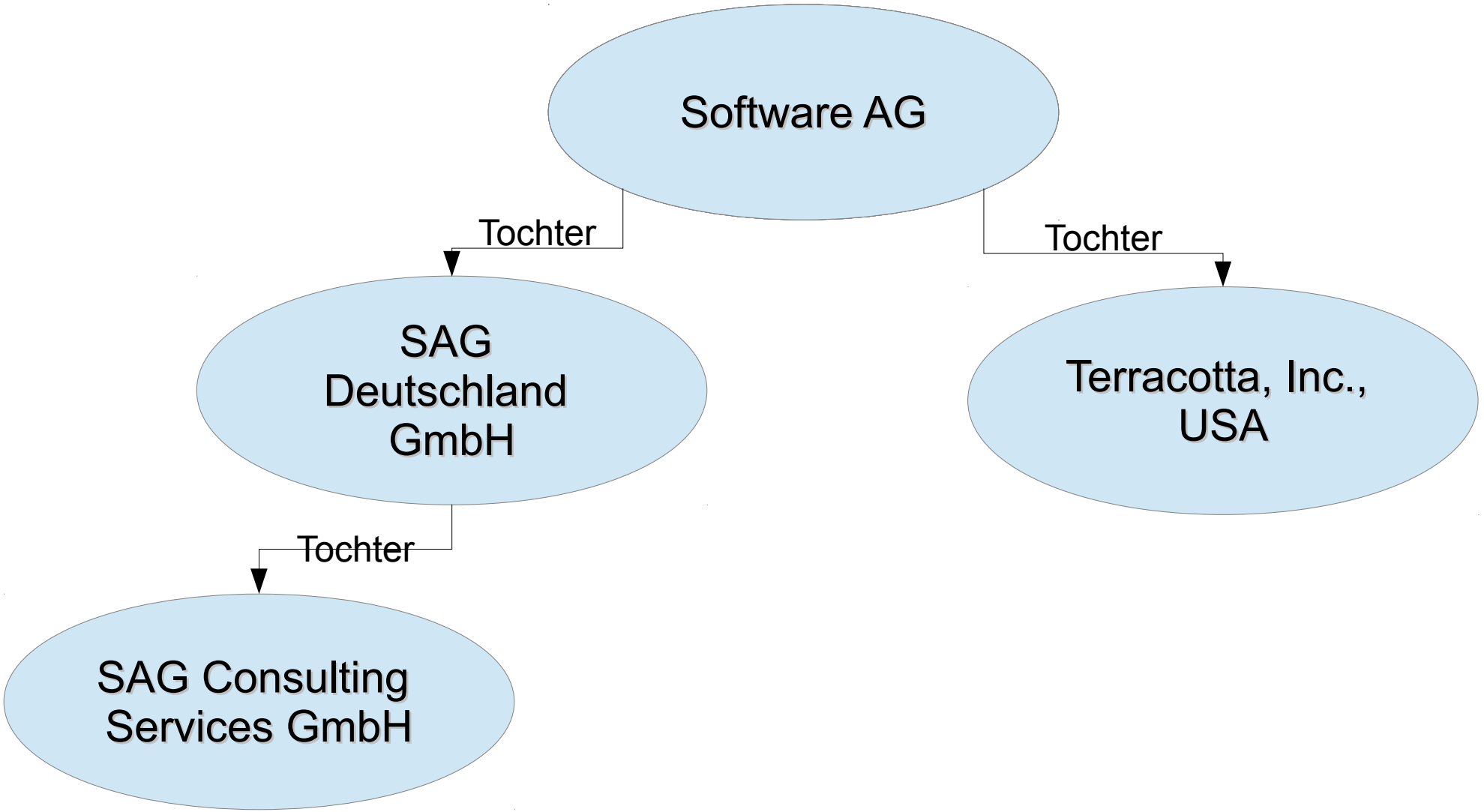
EHCache und Terracotta

Jochen Wiedmann,
Software AG

Autor

- Perl-Contributor
  - DBD::mySQL 2, DBI::Proxy, DBI::Shell, DBD::CSV, Net::Daemon, RPC::Pl(Client|Server) (Autor)
  - DBI (Developer)
- ASF-Member (Apache Software Foundation)
  - JaxMe, Commons FileUpload 2, XmlRpc 3 (Autor)
  - Commons, Creadur, Webservices (PMC-Mitglied)
  - James, Maven, Ant, Axis, ... (Developer bzw. Contributor)
- Senior Consultant Software AG Professional Services

Firma


Software AG
Tochter
Tochter
SAG Deutschland GmbH
Terracotta, Inc., USA
Tochter
SAG Consulting Services GmbH

EHCache

- API und minimale Implementation
  - JSR 107 (Co-Spec-Lead ist EHCache-Maintainer) wird unterstützt
- Open Source (ASL 2.0)
- Entwickelt und betreut von Terracotta, Inc. (Kein Apache-Projekt, aber auf SourceForge)
- Defacto-Standard (Hibernate, Spring, MediaWiki (Wikipedia))
- Unterstützt ColdFusion, Jruby/Rails, Google App Engine, Tomcat, Grails, Glassfish

EHCache-EE

- Proprietäre und kommerziell lizensierte Implementation des EHCache-API

Warum ein Cache?

- Performance: Zugriff erfolgt auf schnellstes Medium (Heap, RAM, Disk)
- Auslastung: Schont den Flaschenhals, d.h. die Datenbank.
- Skalierbarkeit: Wachstum bis 1 TB (BigMemory)

EHCache (2)

- EHCache ist ein Key-Value-Store

```
public class Element {
   public Object getKey(); // Normal ein String
   public Object getValue(); /* Normal ein Serializable */

}

public class Cache {
  public void put(Element pElement) throws CacheException;
  public Element get(Object pKey) throws CacheException;

    /* → evtl. null */

}
```

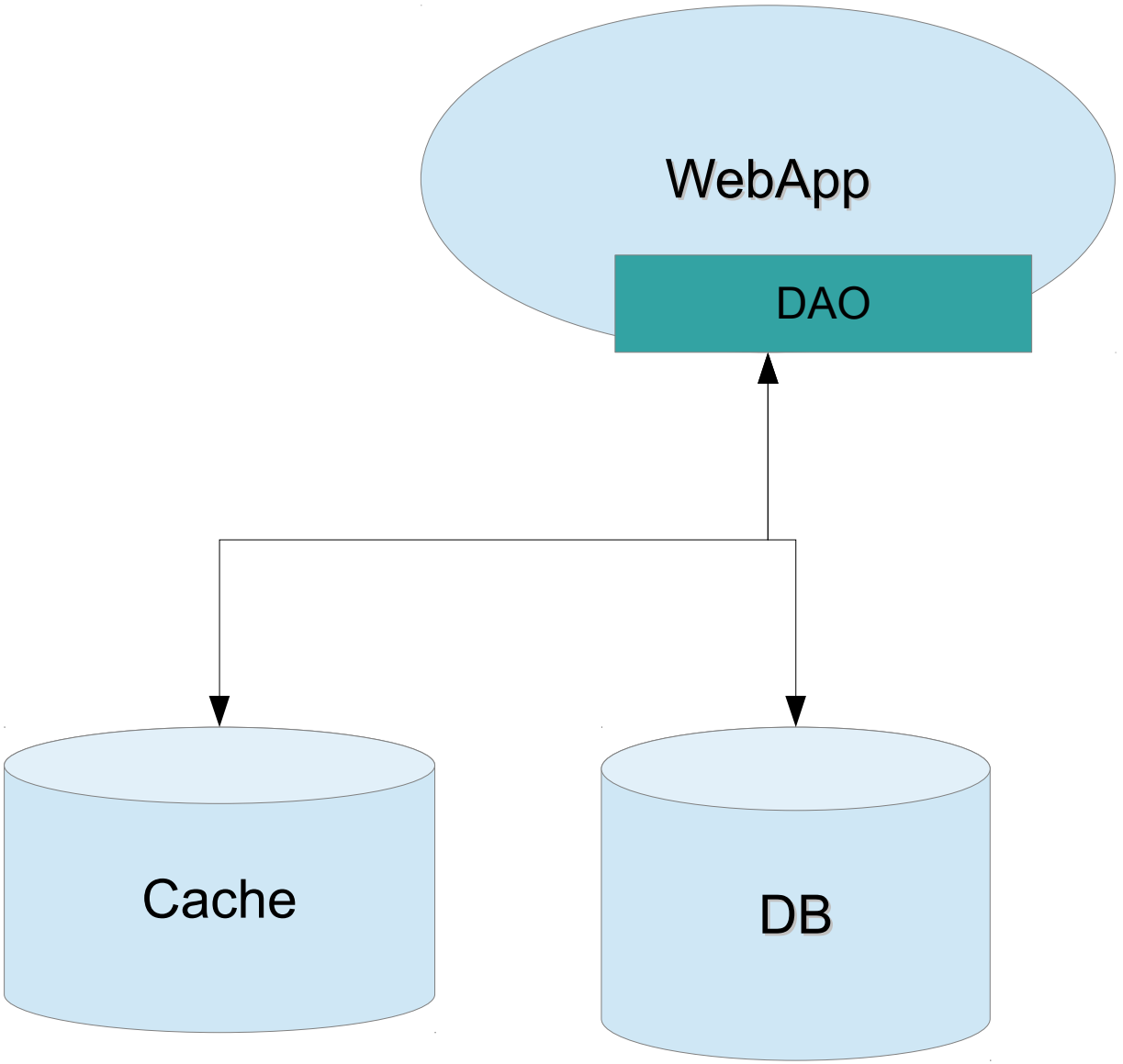
Installation

- Client: Jar-Files ins Projekt einbinden.
- Server:Skriptfähiger jar-Installer
- In jedem Fall: Konfigurationsdatei erstellen.
- Keine RPM's oder dergleichen.

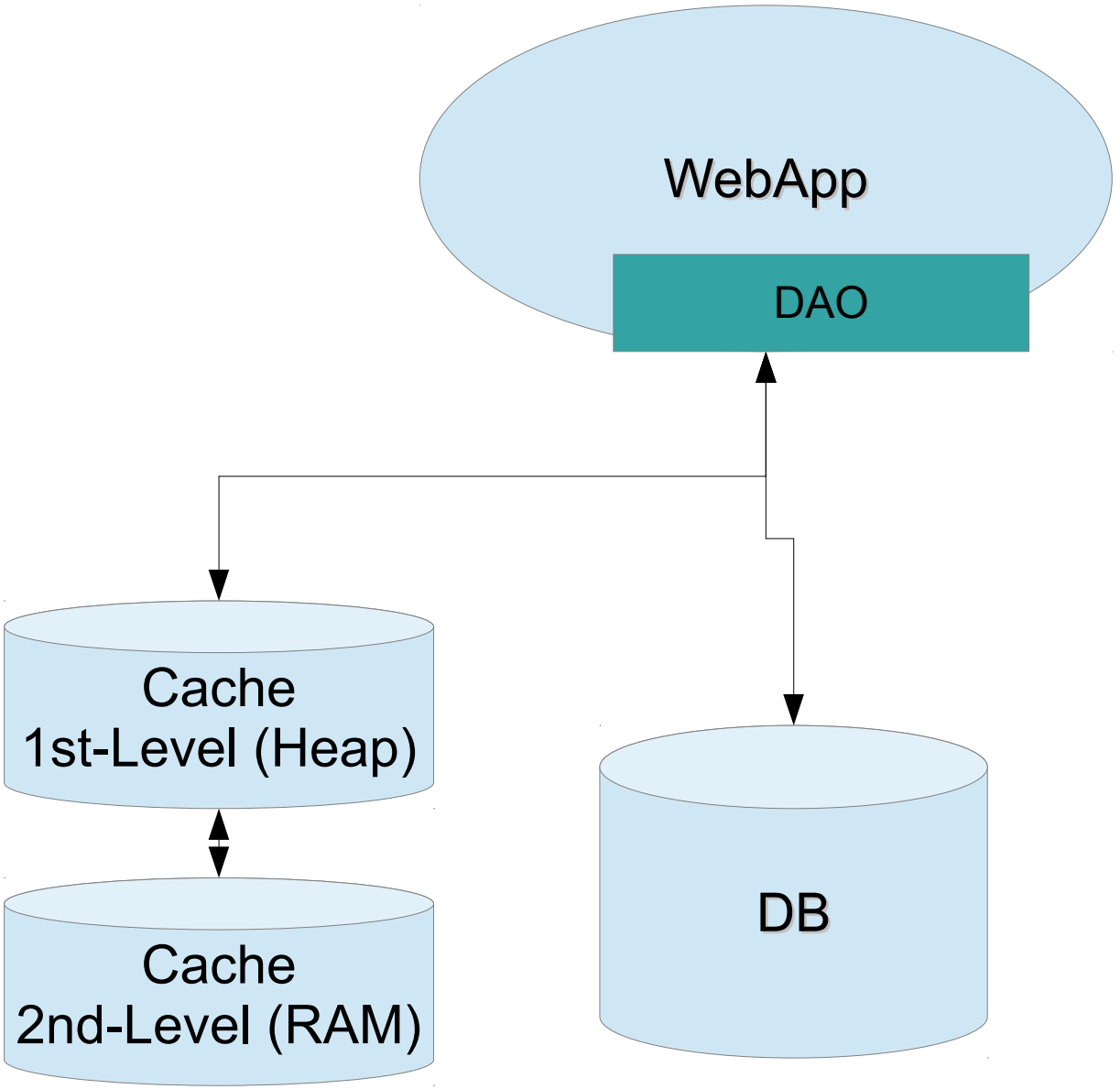
EHCache (3)

- Wie Map, nur etwas cleverer:
- Attribute:
  - MaxElementsInMemory
  - Eternal
  - TimeToIdleSeconds
  - TimeToLiveSeconds
  - OverflowToDisk (EE)
  - DiskPersistent (EE)
  - DiskExpiryThreadIntervalSeconds (EE)
  - MaxElementsOnDisk (EE)
  - Consistency (eventual, strong)
  - EvictionStrategy (fifo, lru)

Anwendungsszenario 1a

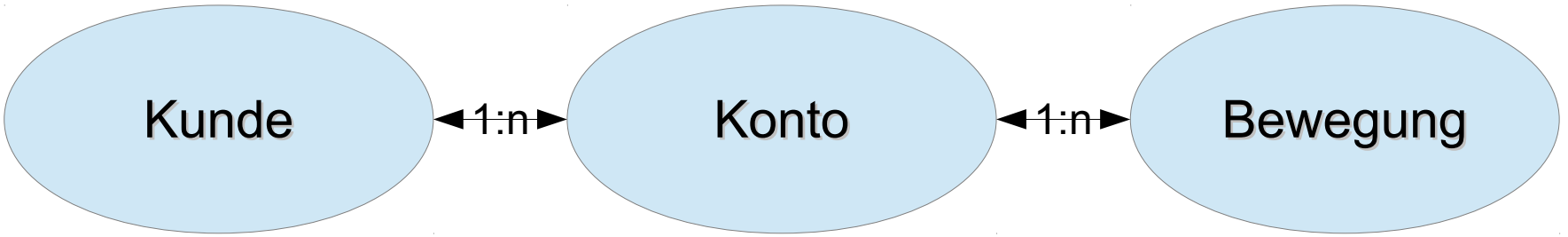


Anwendungsszenario 1b



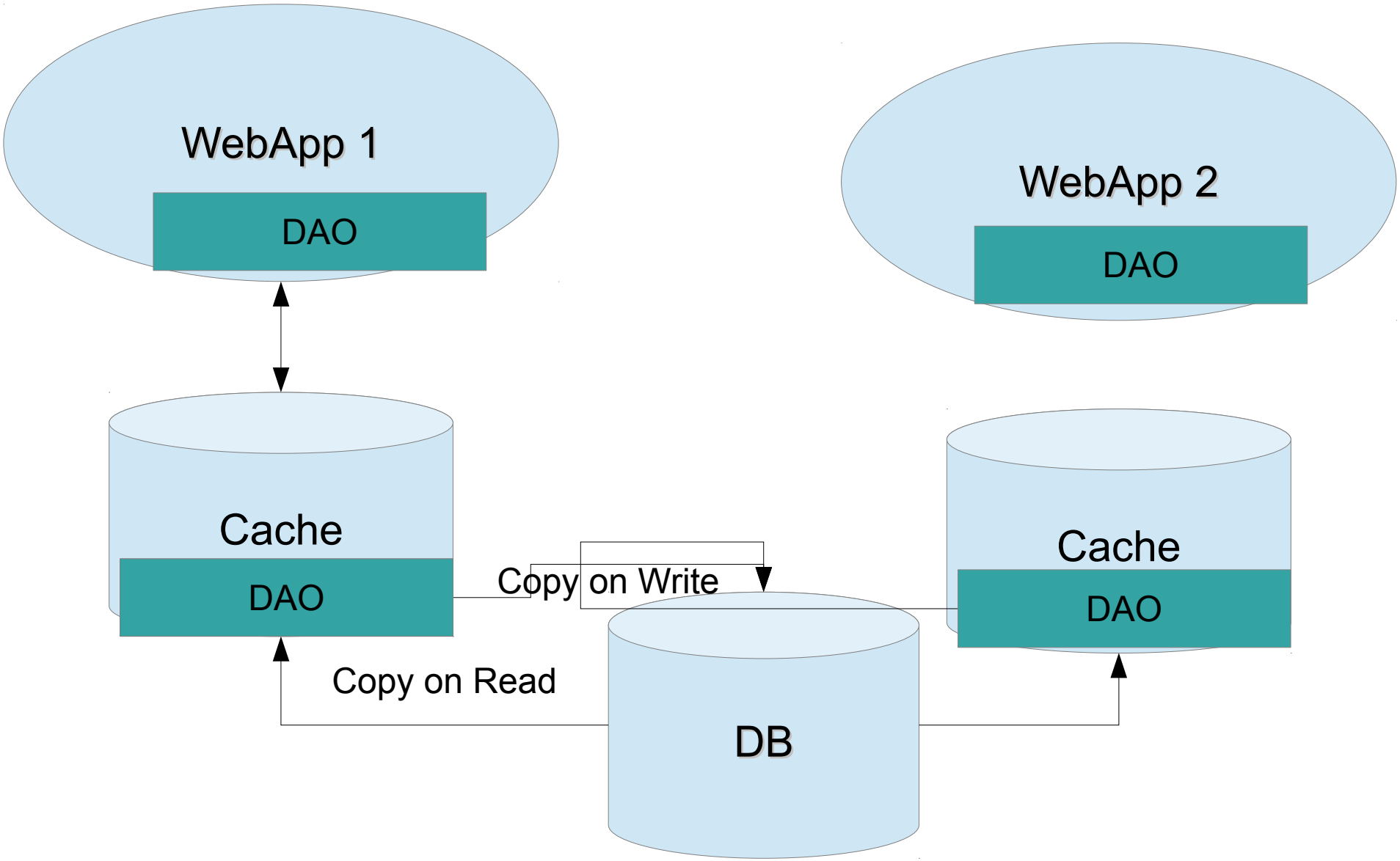
Daten im Cache

- Beispiel



- Kunde mit anhängenden Daten wird einmalig bei Anmeldung geladen und im Cache gespeichert.

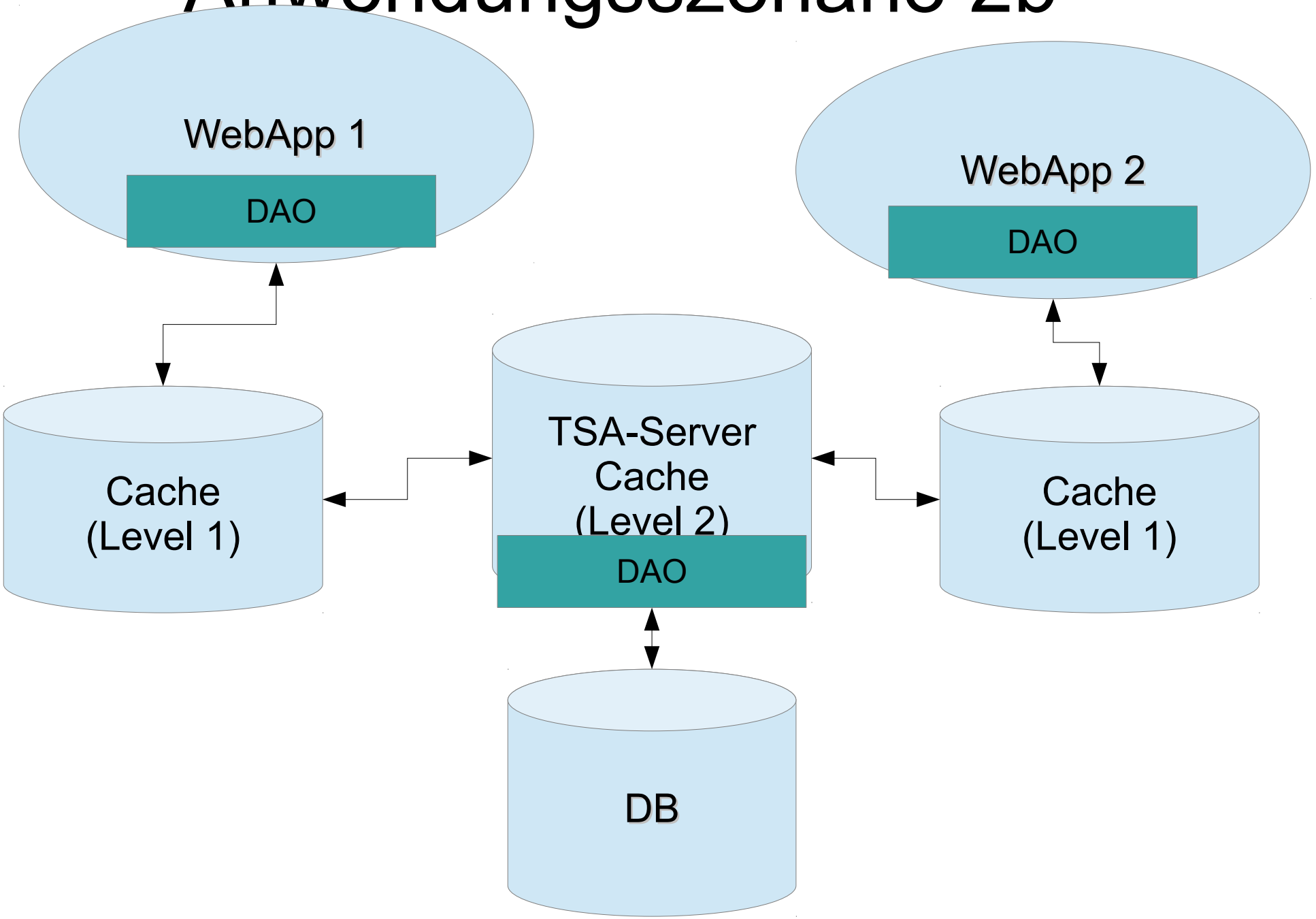
Anwendungsszenario 2a



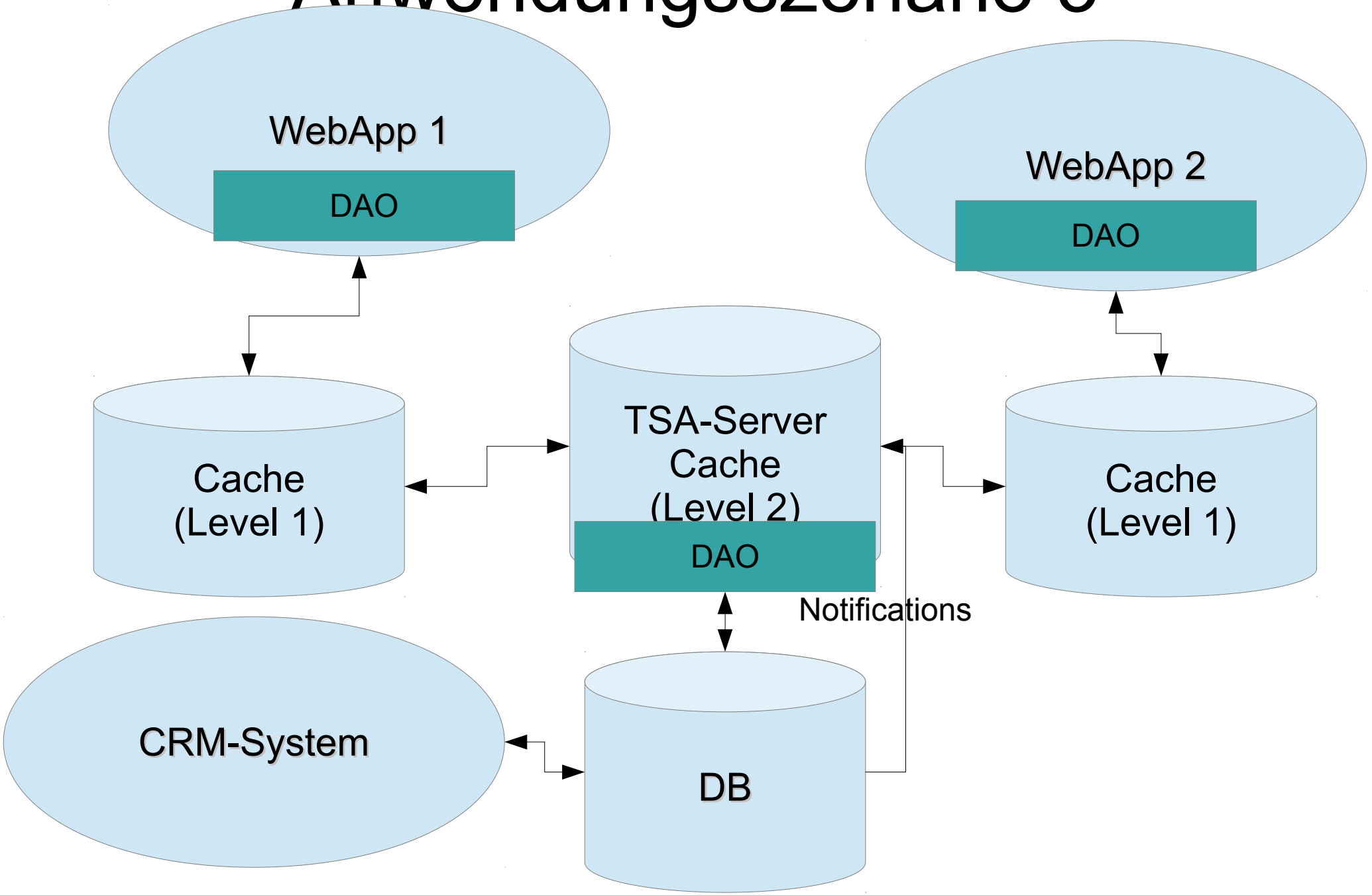
Konsistenz

- Einbindung in XA-Transaktionen
- Vermeidung von alternativen Zugriffswegen (Read-through=Copy on Read bzw. Write-through)
- Kostet Performance

Anwendungsszenario 2b

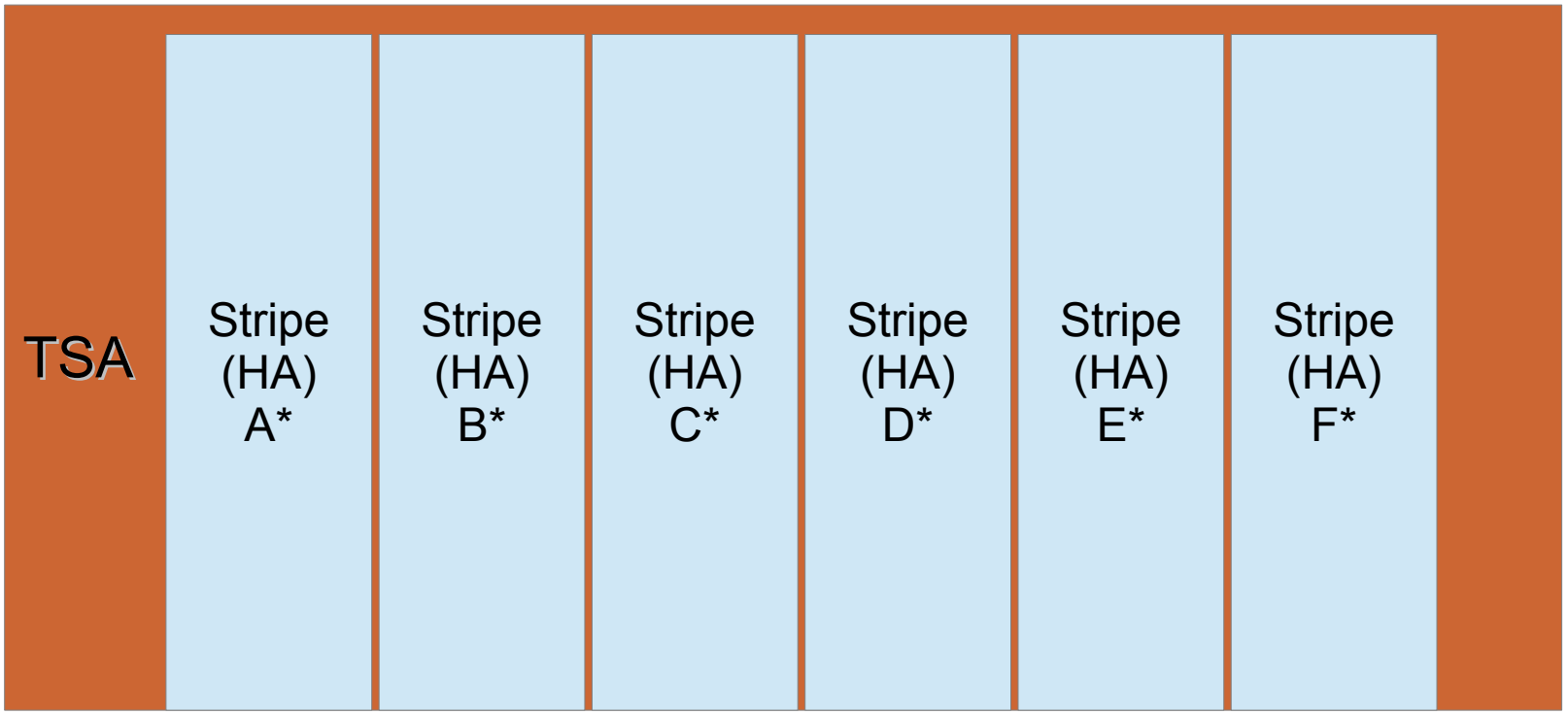


Anwendungsszenario 3



TSA-Skalierung

- Google-Modell: Commodity-Hardware, keine VM's, ganz schlecht mit NAS.
- Beliebige Skalierbarkeit durch Clustering: Jedes Stripe kann als HA-Cluster ausgelegt werden.
- Automatische Adressierung der Nodes



Konkurrenz belebt das Geschäft

- Cache-Lösungen
  - Oracle Coherence
  - MemCached
- NoSQL-Datenbanken
  - CouchDB
  - MongoDB
- Distributed Storage
  - Apache Hadoop
  - Apache Cassandra
  - Google BigTable
- Real Time Event Processing
  - Software AG CEP Server
  -

Über den Gartenzaun

- TSA und EHCache benötigen JVM.
- Direkt verfügbar nur unter Java, Groovy, Jruby, Scala, etc.
- TSA bietet SOAP- oder REST-API
- Dadurch ansatzweise auch in anderen Sprachen.

CAP- oder Brewer-Theorem

- Ein verteiltes System kann nur zwei der folgenden Eigenschaften haben:
- Konsistenz (C): Alle Knoten sehen zur selben Zeit dieselben Daten. Diese Konsistenz sollte nicht verwechselt werden mit der Konsistenz aus der ACID-Transaktionen, die nur die innere Konsistenz eines Datenbestandes betrifft.
- Verfügbarkeit (A): Alle Anfragen an das System werden stets beantwortet.
- Partitionstoleranz (P): Das System arbeitet auch bei Verlust von Nachrichten, einzelner Netzknoten oder Partition des Netzes weiter.
-