Der Middleware Cache mit seiner Architektur und Implementierung

ORACLE®

Cached Data Management Architektur

Wolfgang Weigend
Sen. Leitender Systemberater
Java Technologie und Architektur

JAVAFORUM stuttgart 2011 am 7. Juli 2011

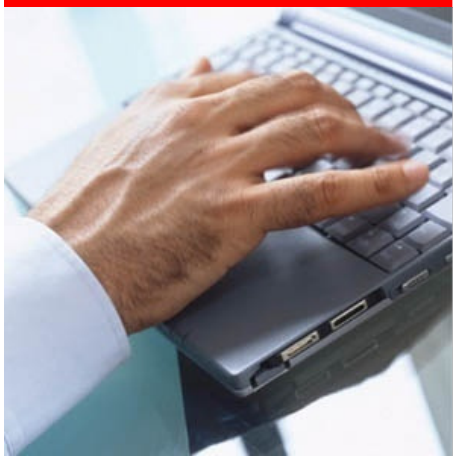
Agenda

- Anforderungen und Auswirkungen
- Maßnahmen und Lösungsansatz
- Coherence Funktionsweise und Deployment-Modelle (Clustered Caching und Zustandsverteilung)
- Eigenschaften von Coherence
- Partitionierte Topologie (Data Access, Data Update, Recovery)
- Architektur Integration Pattern (Read-Through, Write-Through, Write-Behind)
- Cache Architekturschichten (Coherence Integration mit TopLink/EclipseLink, Hibernate, und Spring)
- Einsatzfälle für Coherence
- Zusammenfassung

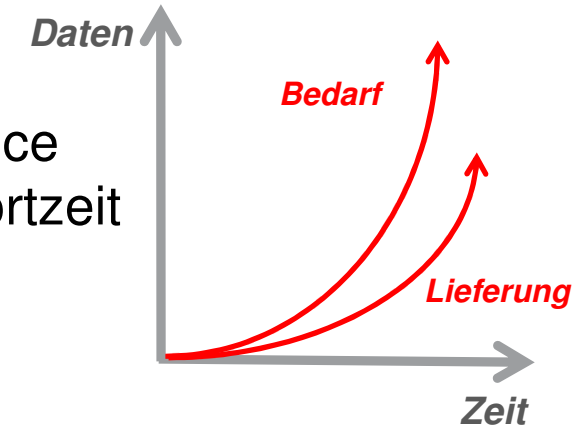
Geschäftliche Anforderungen und Auswirkungen

- **Skalieren durch dynamische Änderungen**
  - Dynamische, vorhersagbare Applikationsskalierbarkeit, angepasst an die Geschäftsanforderungen
  - Extremer Anstieg beim Datenzugriff, Volumen und Komplexität der Datennutzung
- **Kundenzufriedenheit erhöhen**
  - Bessere Applikationsleistung
  - Schneller Zugriff auf Daten, kürzere Antwortzeiten
- **Geschäftskontinuität ermöglichen**
  - Kontinuierliche Datenverfügbarkeit und Zuverlässigkeit
  - Service Level Agreements einhalten / überschreiten
  - Eindämmung von Kosten
- **Infrastruktur- und Entwicklungskosten reduzieren**
  - IT-Investitionen wirksam einsetzen, Cloud Computing
  - Build vs. Buy

Skalierbare Leistung und Transaktionsverarbeitung

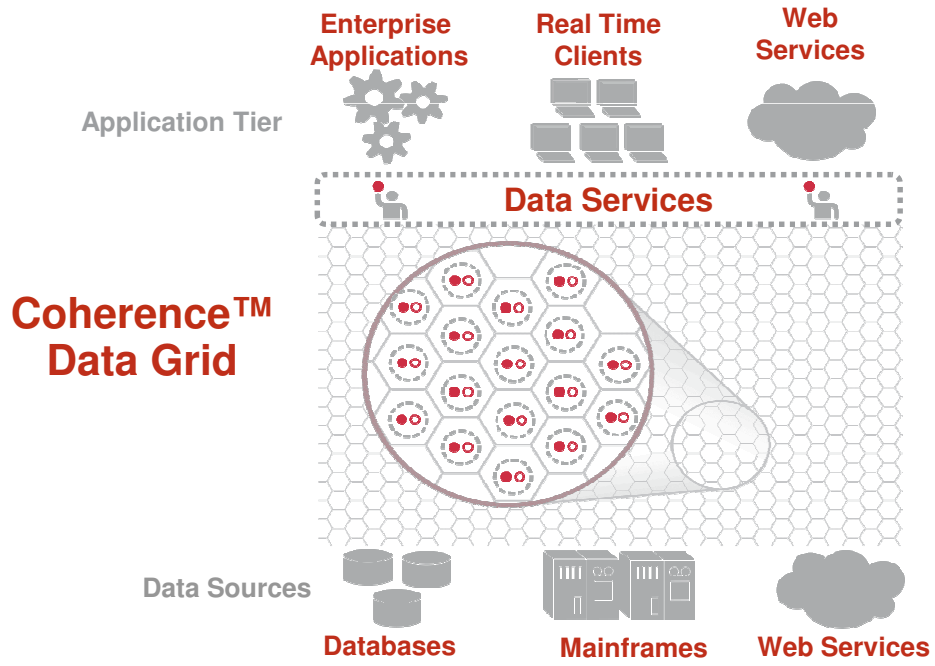


- **Services und Events stellen neue Anforderungen an die Transaktionsverarbeitung**
  - Unbekannte Anforderungen (Business Geschwindigkeit, neue Zuordnung, Innovation)
  - Komplexität von Services nimmt zu (State), Realtime-Analyse
- **Skalierbare Leistung (High Performance)**
  - Zielsetzung ist, sicherzustellen, dass ein Service innerhalb der erforderlichen, definierten Antwortzeit auch in extremen Situationen reagiert und bei Bedarf adhoc nahtlos skalierbar ist
- **Transaktionsplattform**
  - Verteilte Transaktionsverarbeitung (Distributed Transaction Processing for Open Systems)
  - Extreme Transaktionsverarbeitung (XTP)



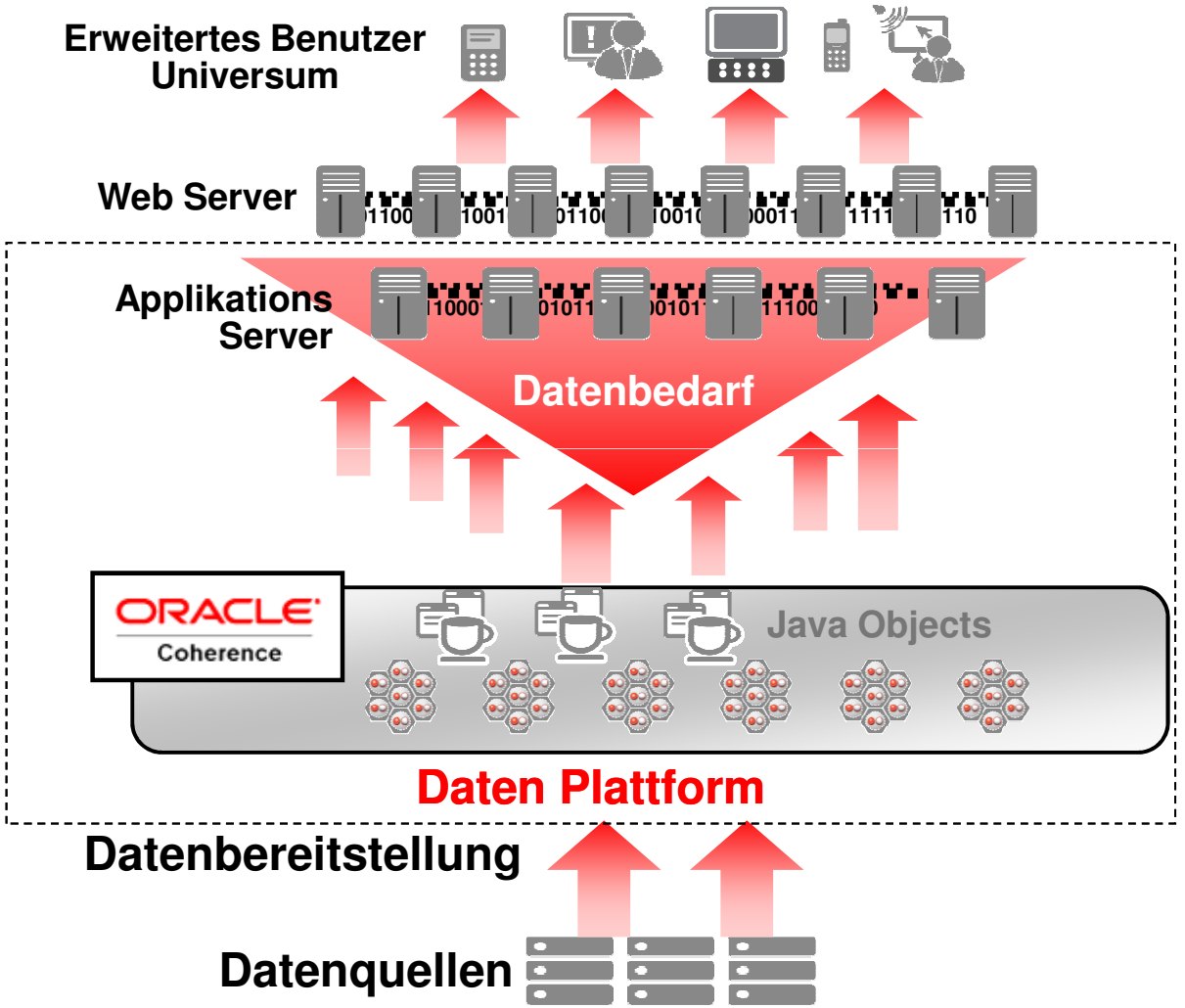
Geeignete Maßnahmen

- Coherence Data Grid
  - Liefert eine zuverläßige Datenschicht mit einer konsistenten Sichtweise und transaktionaler Integrität
  - Ermöglicht die dynamische Erhöhung der Datenkapazität mit Lastverteilung, kontinuierlicher Verfügbarkeit und Fehlertoleranz
  - Stellt die erforderliche Verarbeitung und Skalierbarkeit der benötigten Datenkapazität sicher mit extrem niedrigen Antwortzeiten (Low Latency)

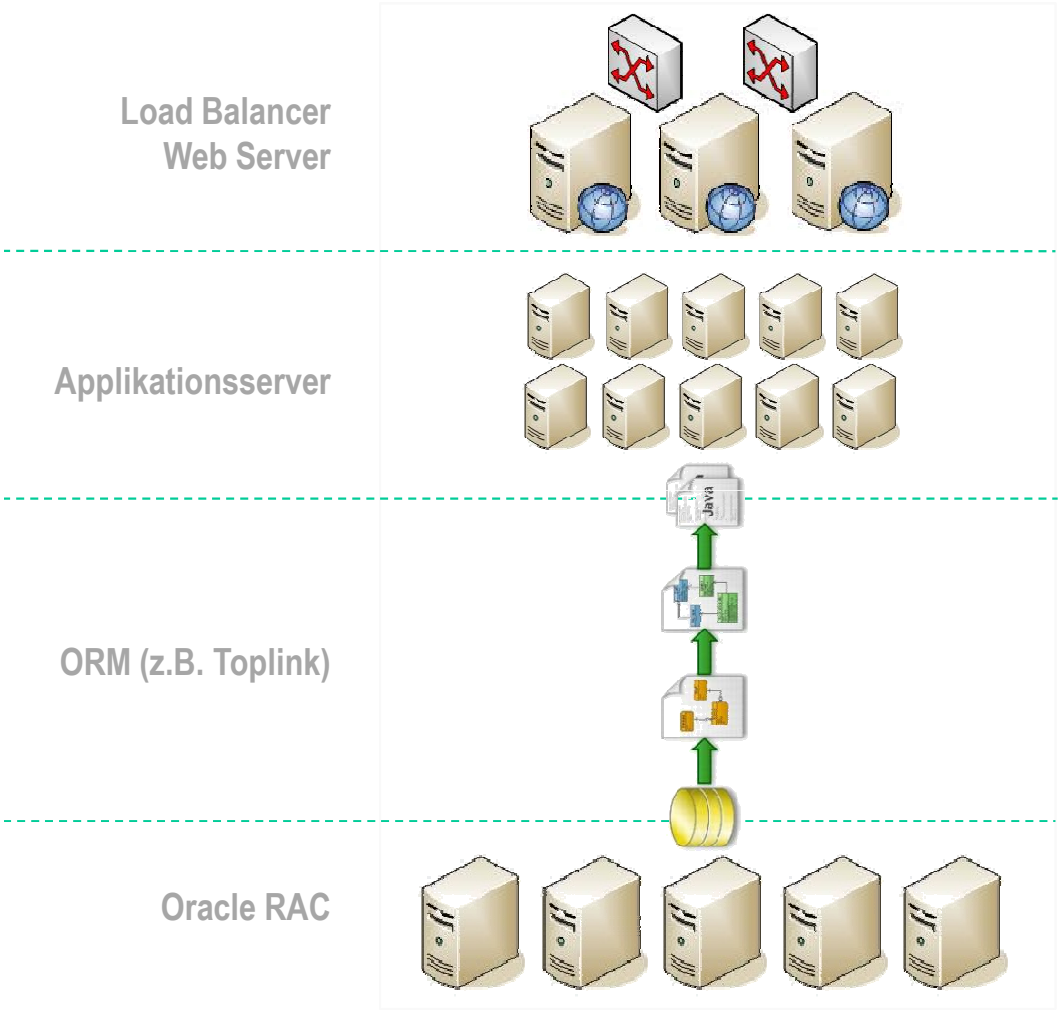


Lösungsansatz: Applikationsserver mit Coherence als Datenvermittler

- Coherence vermittelt die Datenversorgung mit dem Datenbedarf
- Data Grid im Middle-Tier mit Standard-Hardware skalieren

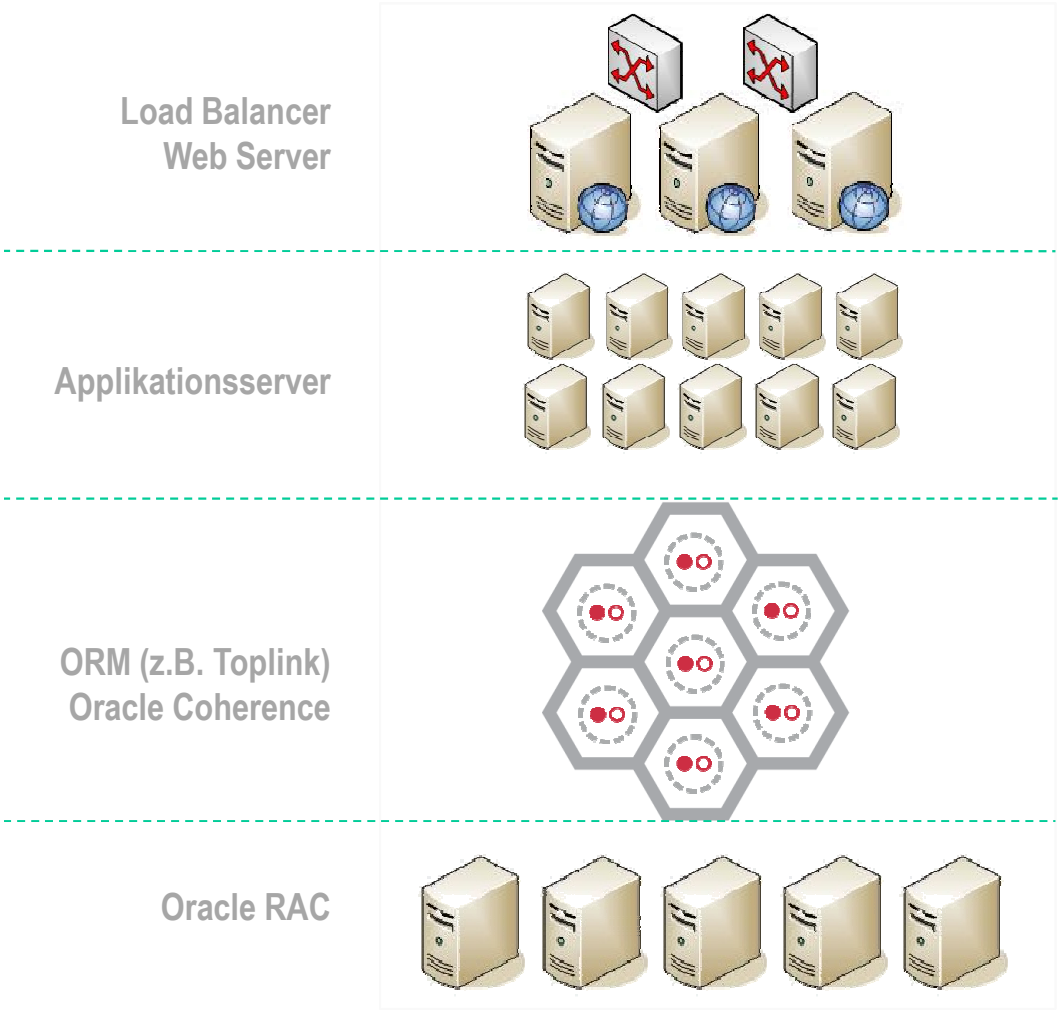


Herausforderungen



- Skalierbarkeit der Applikations-schicht ist Herausforderung
  - Loosely-Coupled = Split Brain?
  - Transaktionsverarbeitung von Java
- ORM (Java → SQL Mapping) ist ein Bottleneck (Latency)
- In-Memory-Ansatz erforderlich um Antwortzeiten zu erfüllen (Disk-I/O Bottleneck)
- Verlust auch nur eines einzelnen Transaktionstatus nicht akzeptabel (Finanzhandel, Flugbuchung)
- Session-States in heterogenen Umgebungen schwierig

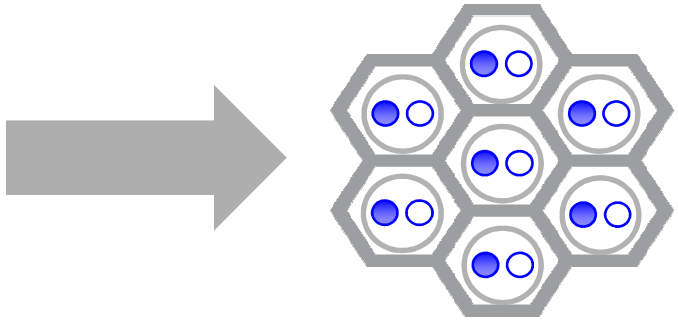
Oracle Coherence: Data Grid

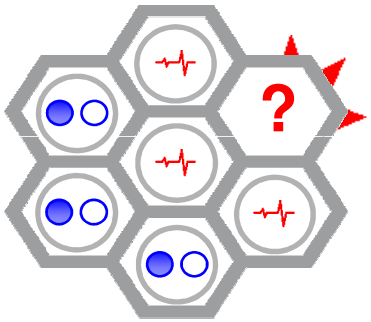


- Daten Virtualisierung mit Oracle Coherence: Caching von serialisierbaren Objekten im Grid
- Gemeinsame Nutzung von Daten (Read-Through, Read-Ahead)
- Parallele Abfragen, Indizes und Transaktionen im Cluster
- Entlastung der persistenten, unterliegenden Schicht (Write-Through, Write-Behind)
- Integriert in Fusion Middleware Toplink, Berkeley DB, TimesTen

Wie funktioniert Oracle Coherence?

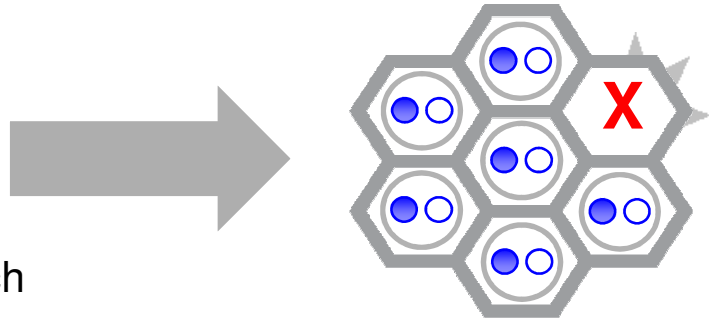
- Daten werden gleichmäßig auf Clusterknoten verteilt
- Backup der Daten wird ebenfalls auf Knoten verteilt
  - Daten werden automatisch und synchron auf mindestens einen weiteren Server repliziert
- Logisch konsistente Sicht auf Daten von allen Knoten



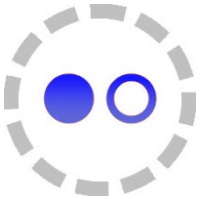


- Alle Knoten prüfen alle Knoten (Health-Check)
- Status bei Fehlern wird cluster-weit diagnostiziert

- Knoten wird aus dem Cluster entfernt
- Daten werden wieder gleichmäßig verteilt
- Kontinuierliche Operation: Kein Service-Abbruch oder Datenverlust wenn ein Server ausfällt



Oracle Coherence



Caching

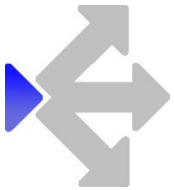
Applikationen nutzen Daten direkt aus dem verteilten, linear skalierbaren Data Grid. Anzahl der Zugriffe auf Backend-Systeme wird reduziert bzw. gezielt vermieden (Datenbanken, Mainframes, etc). Write-Queue reduziert Last auf Backend-Systeme

Analyse

Aggregationen, einfache und komplexe Abfragen können parallel im Data Grid clusterweit verteilt “In-Memory” ausgeführt werden

Transaktionen

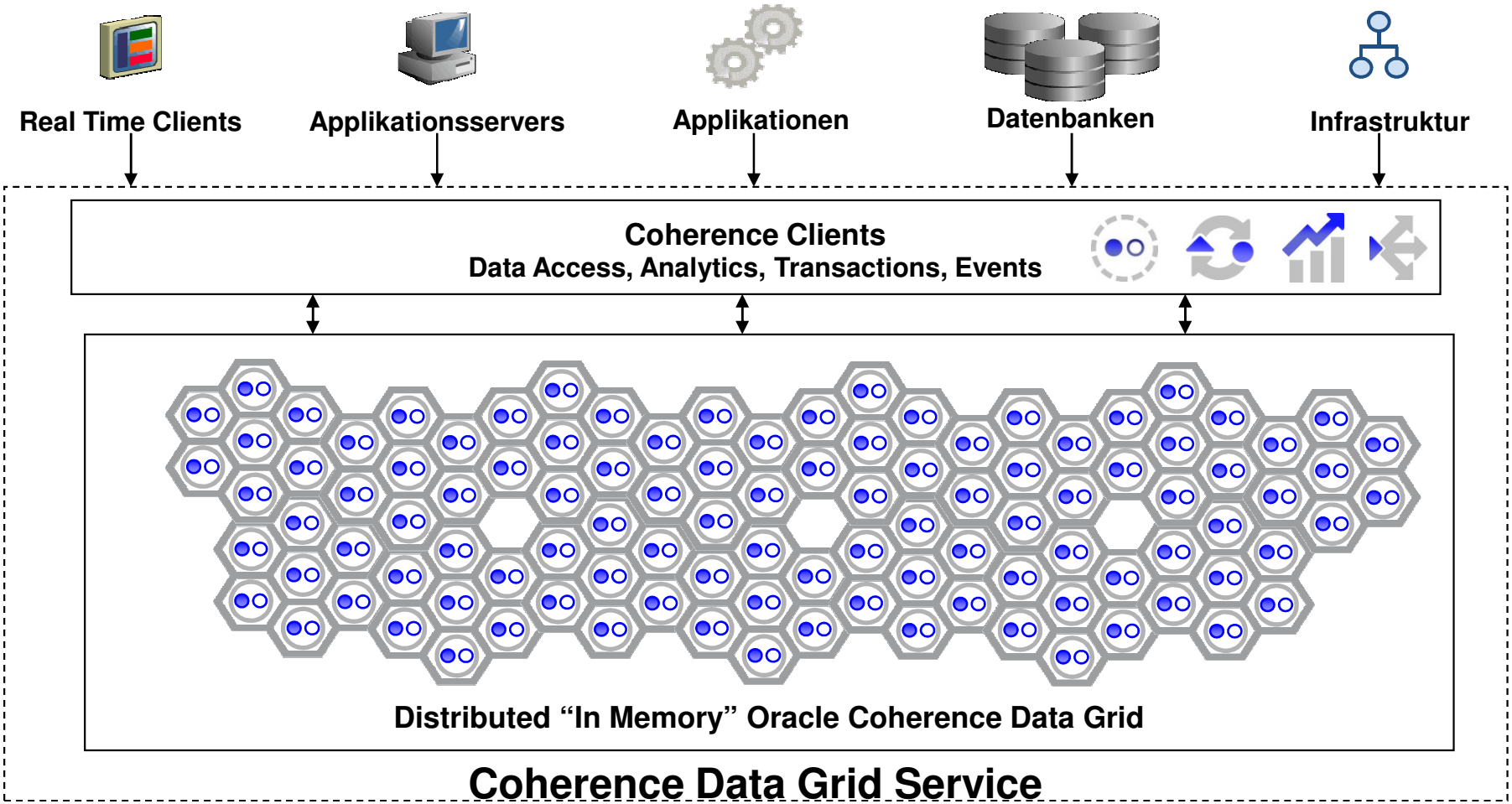
Applikationen können transaktionsrelevante Daten sicher, hochverfügbar und skalierbar im Data Grid verwalten



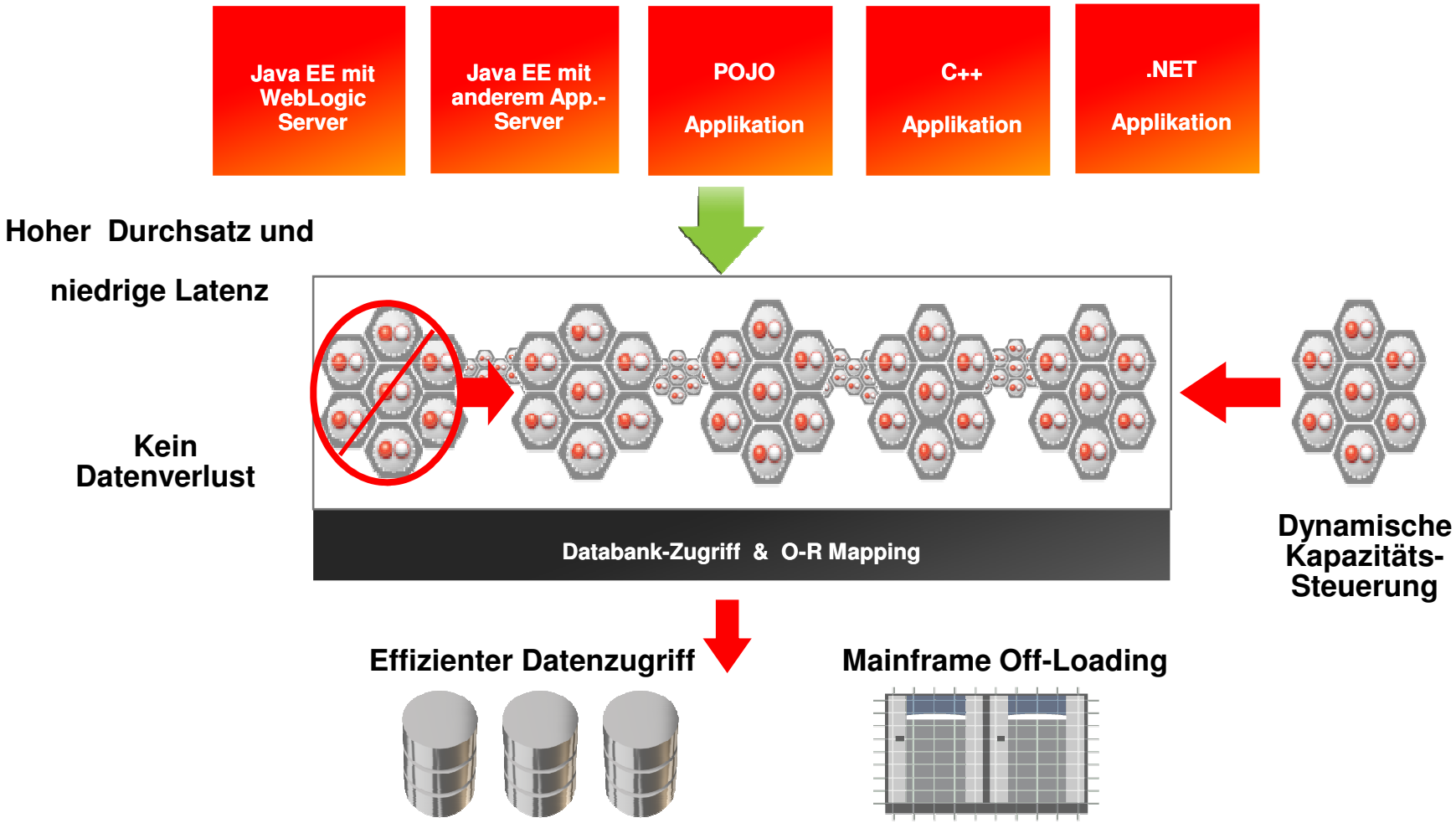
Events

Automatisches Event-Handling, Änderungsbenachrichtigungen

Verteilter “In Memory” Coherence Data Grid Service

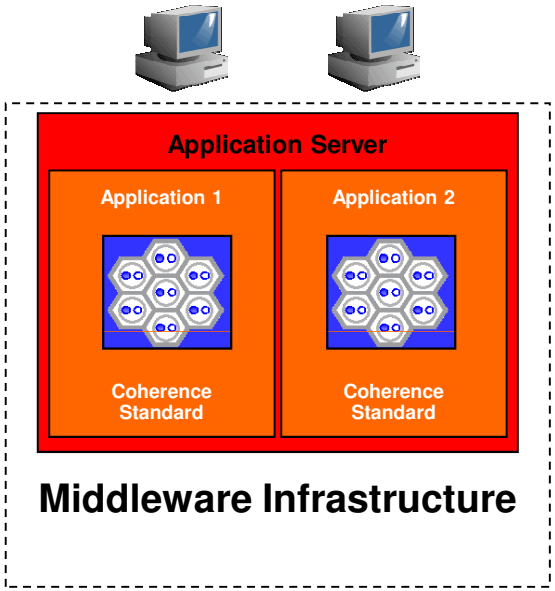


Verteilter “In Memory” Coherence Data Grid Service

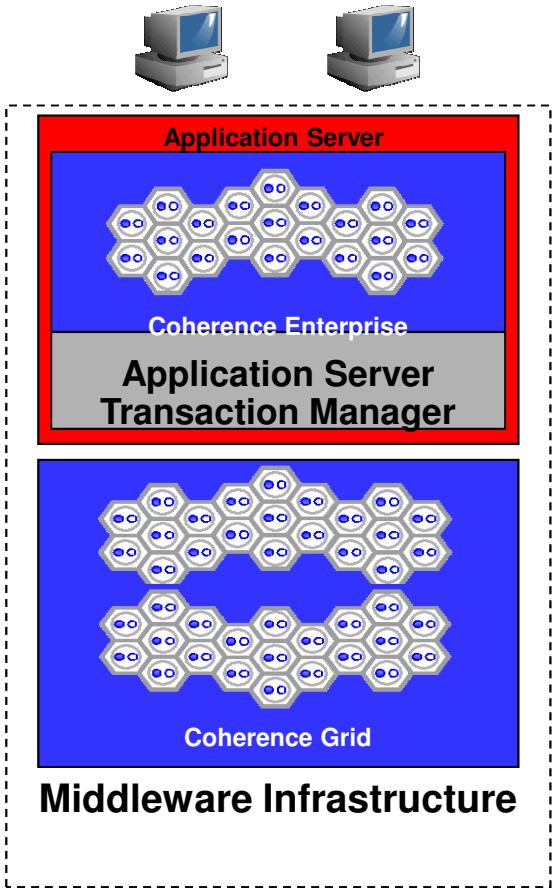


Coherence Betriebs-Modelle

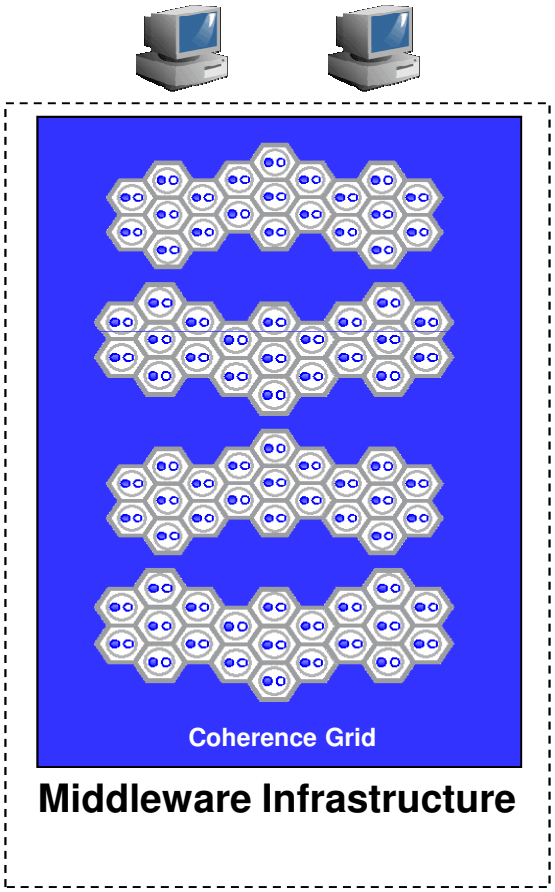
Applikationsserver Coherence Standard Edition



Hybrid-Deployment Coherence Enterprise Edition



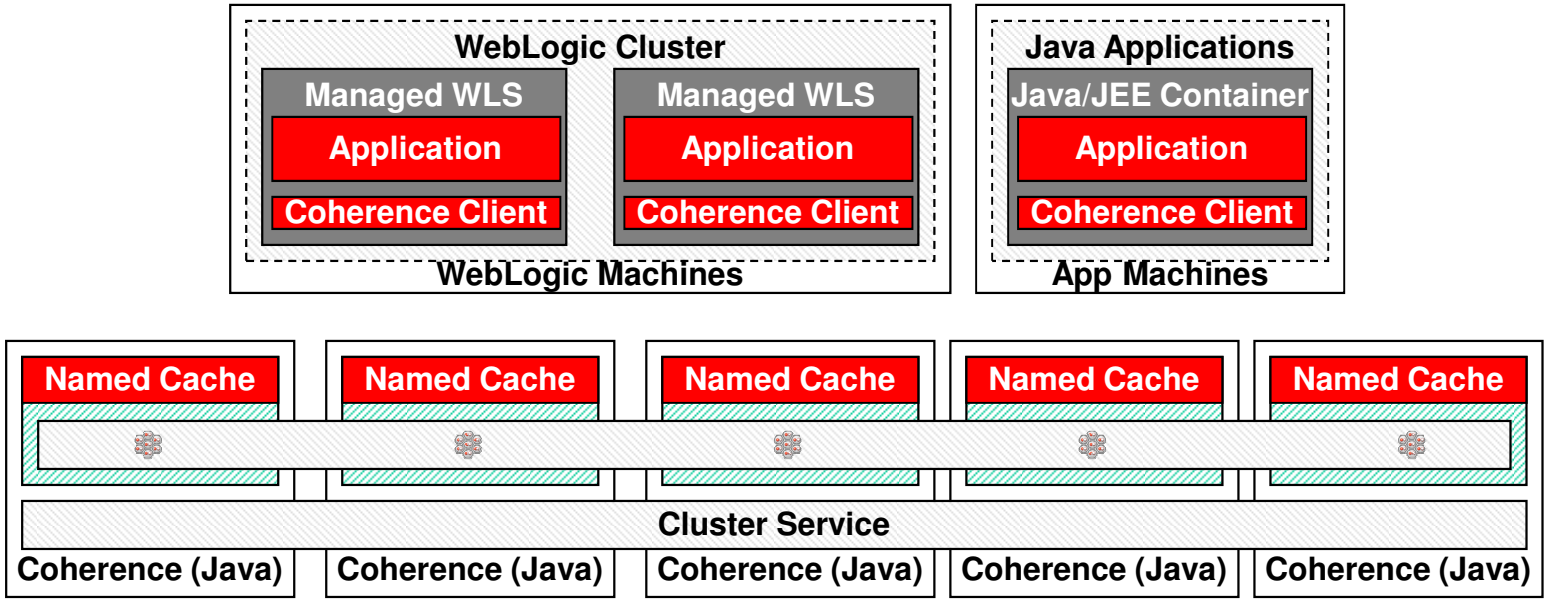
Data Grid Coherence Grid Edition



Kombination von Coherence und WebLogic

Clustered Caching und Zustandsverteilung

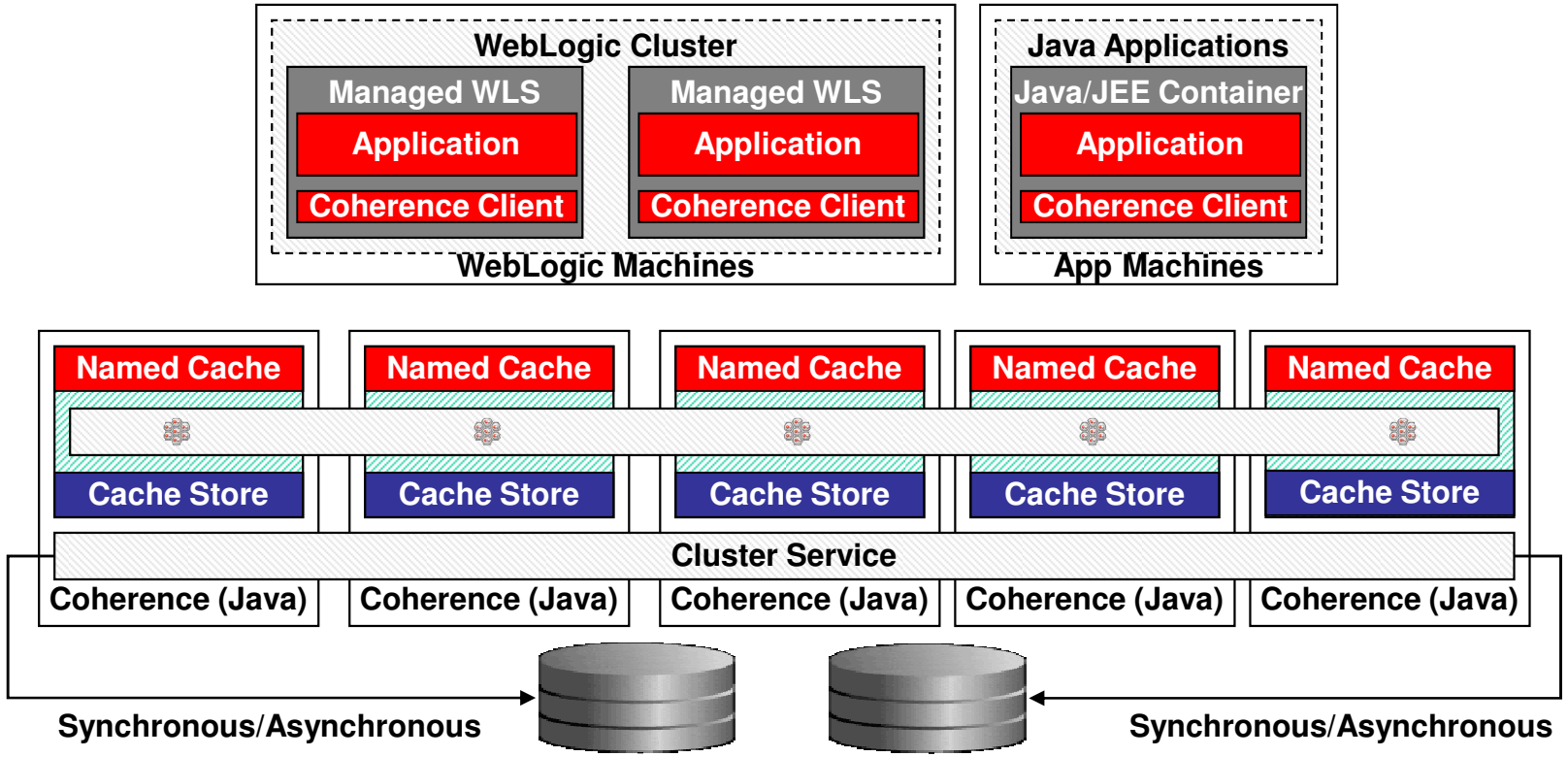
Verteilte Daten im Cache, unabhängige Zustandsverwaltung in heterogenen Java Umgebungen



Kombination von Coherence und WebLogic

Entlastung der Datenbank und Pufferfunktion

Read through und asynchrones ***write through*** zur Datenbank



Coherence Eigenschaften (1)

Datenmanagement zur Verbesserung der Applikationsleistung und Zuverlässigkeit

- Parallele Verarbeitung mit Daten- und Verarbeitungsaffinität
- Parallele Abfragen und Aggregation für Objekt-basierte Abfragen
- Unterstützt Replikation, Partitionierung, Read-through, Write-through, Write-behind und Refresh-ahead Caching
- Verwaltung von Daten über einen Cluster mit Coherence-Datenstrukturen: Sehr schnelles Objekt-Clustering-Verfahren
- Coherence verwendet UDP mit Unicast, kein TCP
  - TCP ist dafür zu schwerfällig
  - Das spezielle Tangosol Cluster Management Protocol (TCMP) verfügt über einen eigenen Serialisierungs-Mechanismus für den Netzwerk Transport
- Coherence verwendet Portable Object Format (POF) für optimierte Objekt-Serialisierung in Java, kein generisches “Java Serializable Object Format”

Coherence Eigenschaften (2)

Datenmanagement zur Verbesserung der Applikationsleistung und Zuverlässigkeit

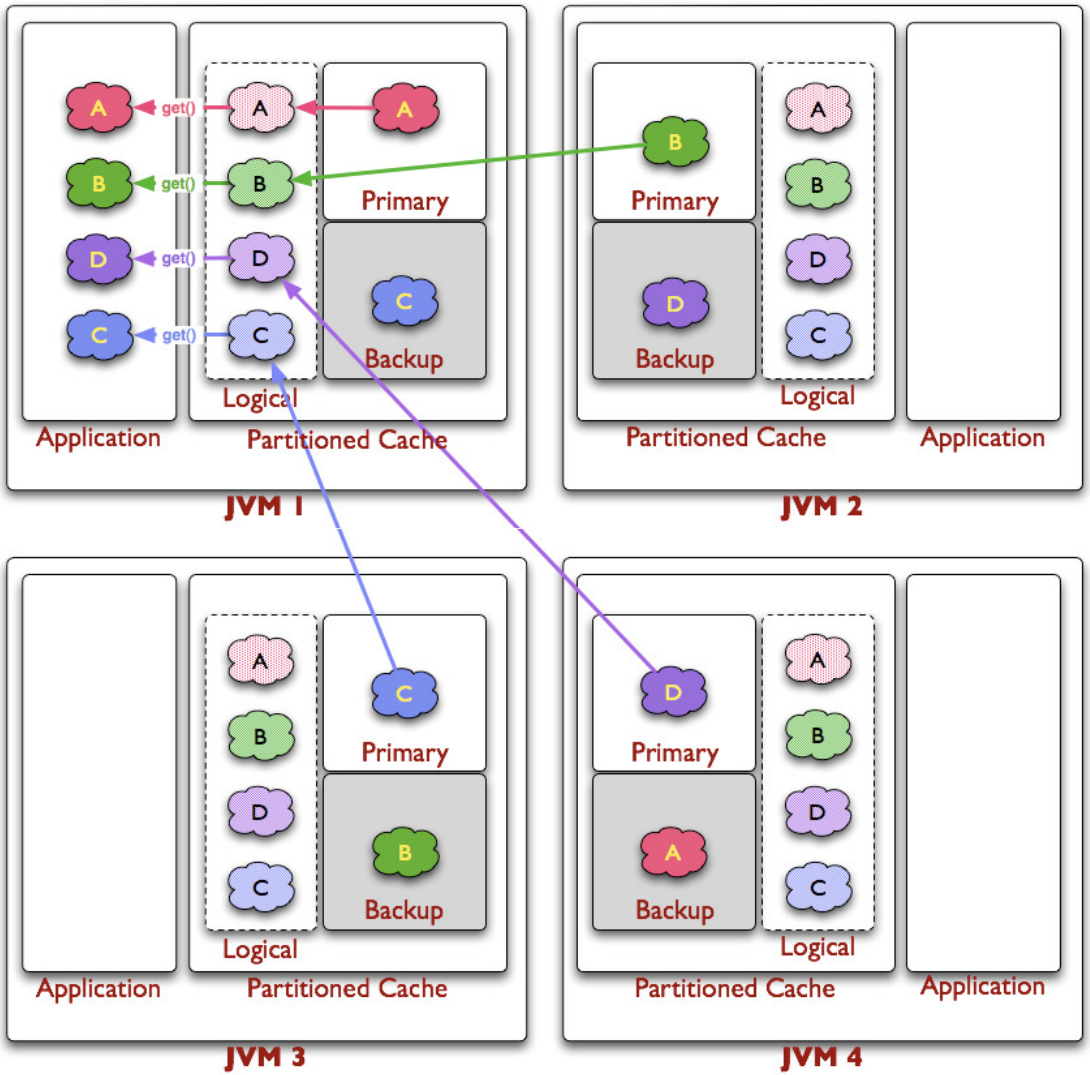
- MAN & WAN Daten- und Verarbeitungsdienste
- Failover und Recovery Management
- Transaktions Management
- Echtzeit-Ereignisüberwachung mit Listener Pattern (Cache Event Benachrichtigung)
- JAAS-basierte Authentication, Network Traffic Encryption, und Java Management Extensions (JMX)-based Monitoring und Management (Clustered JMX)
- JPA über Integration mit Oracle TopLink, Objekt/Relationale Mapping Technologie
- Kombinierbar mit Oracle WebLogic Server und anderen Java EE Applikationsservern
- Coherence API Unterstützung für C++ und .NET

Partitioned Topology: Data Access

- Coherence provides many Topologies for Data Management
- Local, Near, Replicated, Overview, Disk, Off-Heap, Extend (WAN), Extend (Clients)

Partitioned Topology

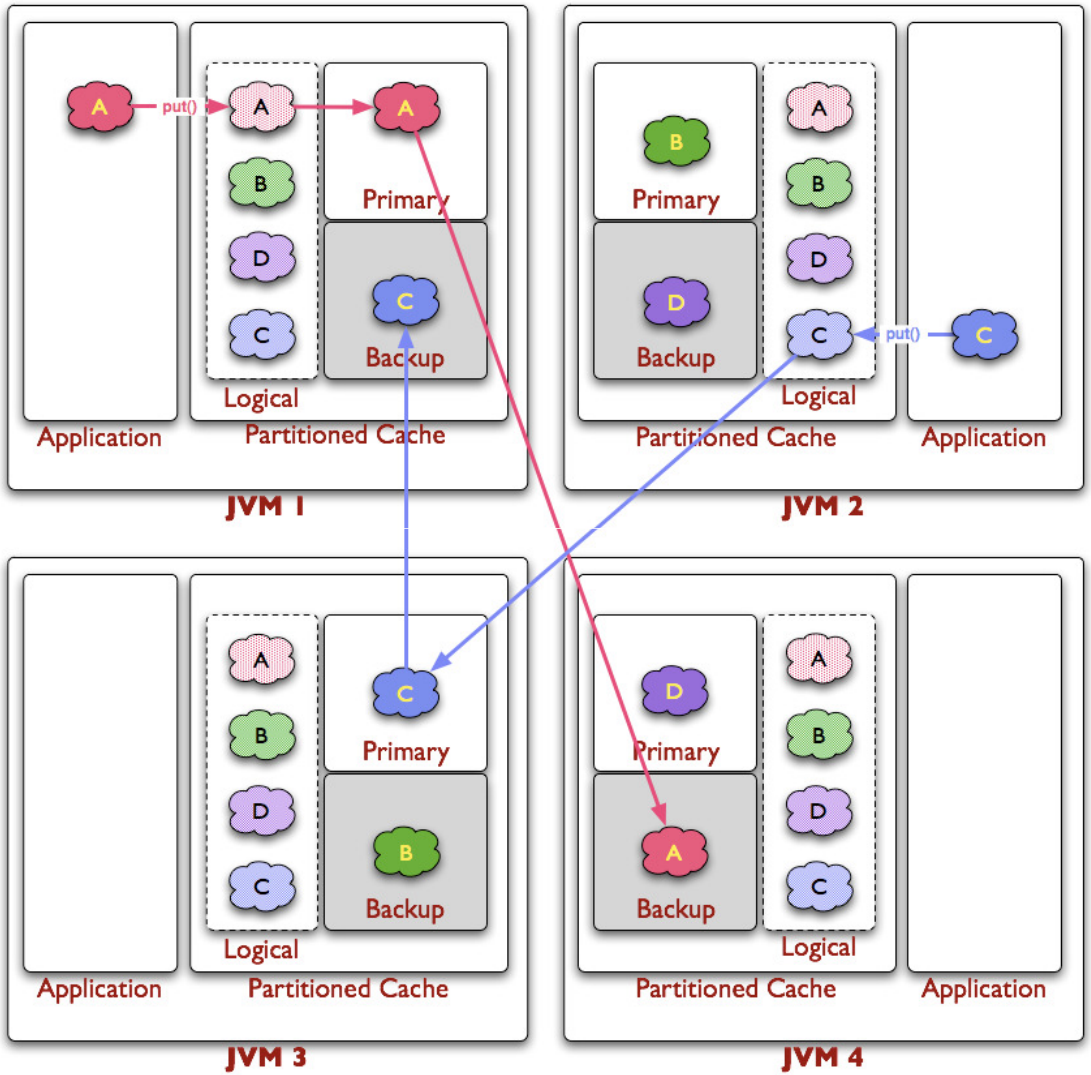
- Data spread and backed up across Members
- Transparent to Developers
- Members have access to all Data
- All Data locations are known – no lookup & no registry!



Partitioned Topology: Data Update

Partitioned Topology

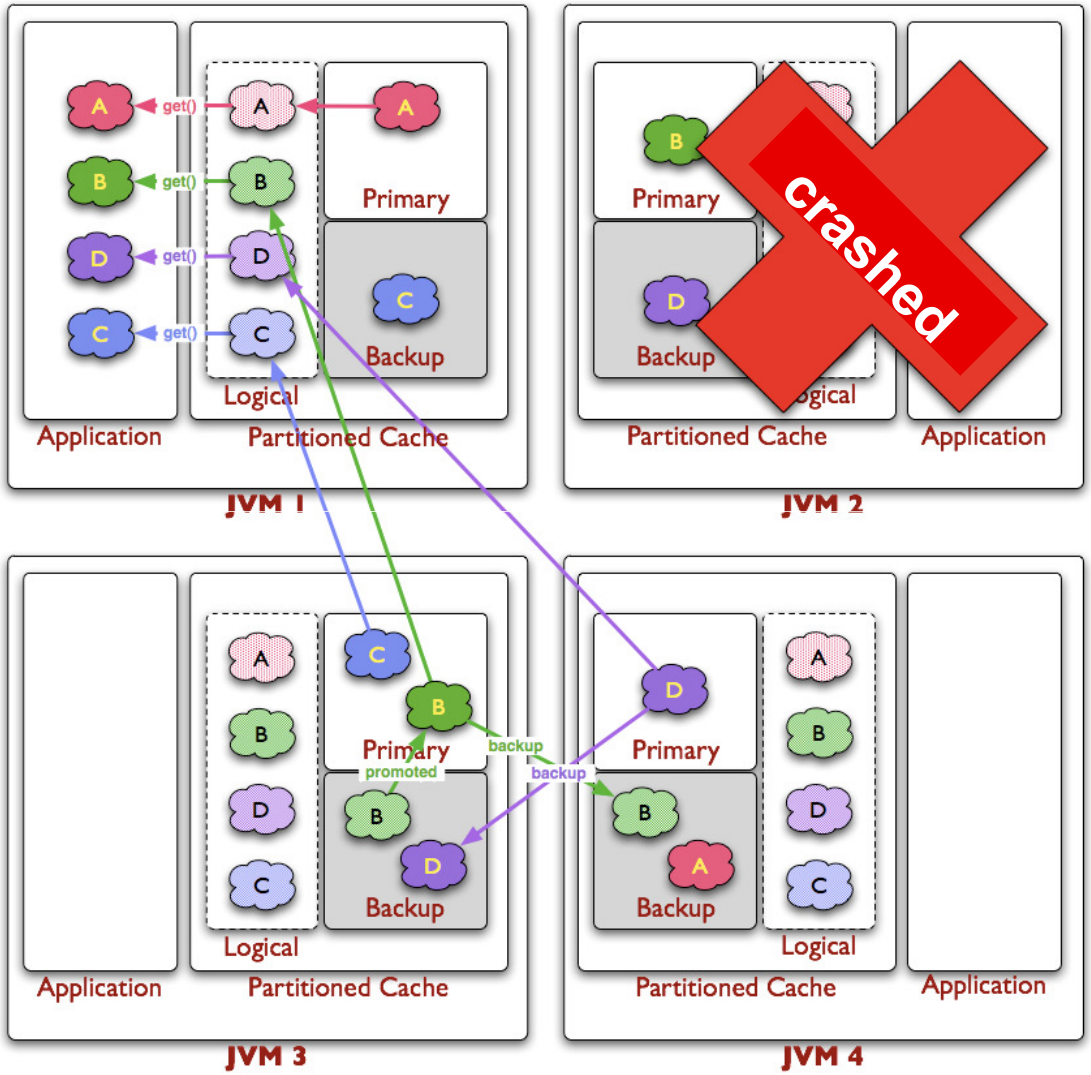
- Synchronous Update
- Avoids potential Data Loss & Corruption
- Predictable Performance
- Backup Partitions are partitioned away from Primaries for resilience
- No engineering requirement to setup Primaries or Backups
- Automatically and Dynamically Managed



Partitioned Topology: Recovery

Partitioned Topology

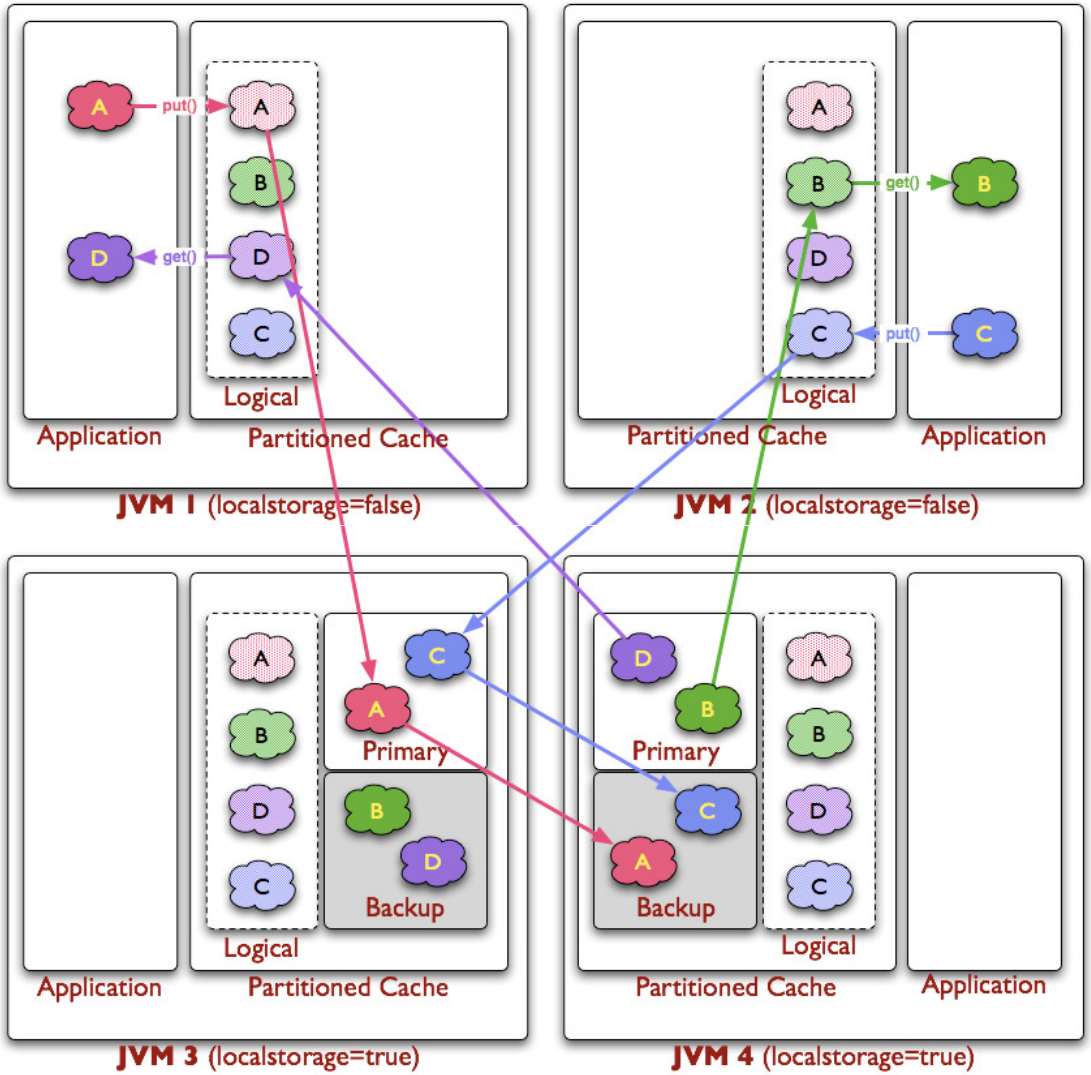
- Membership changes (new members added or members leaving)
- Other members, in parallel, recover / repartition
- No in-flight operations lost
- Some latencies (due to higher priority of recovery)
- Reliability, Availability, Scalability, Performance are the priority
- Degrade performance of some requests



Partitioned Topology: Local Storage

Partitioned Topology

- Some members are temporary in a cluster
- They should not cause repartitioning
- Repartitioning means work for the other members (and network traffic)
- May turn off storage ..

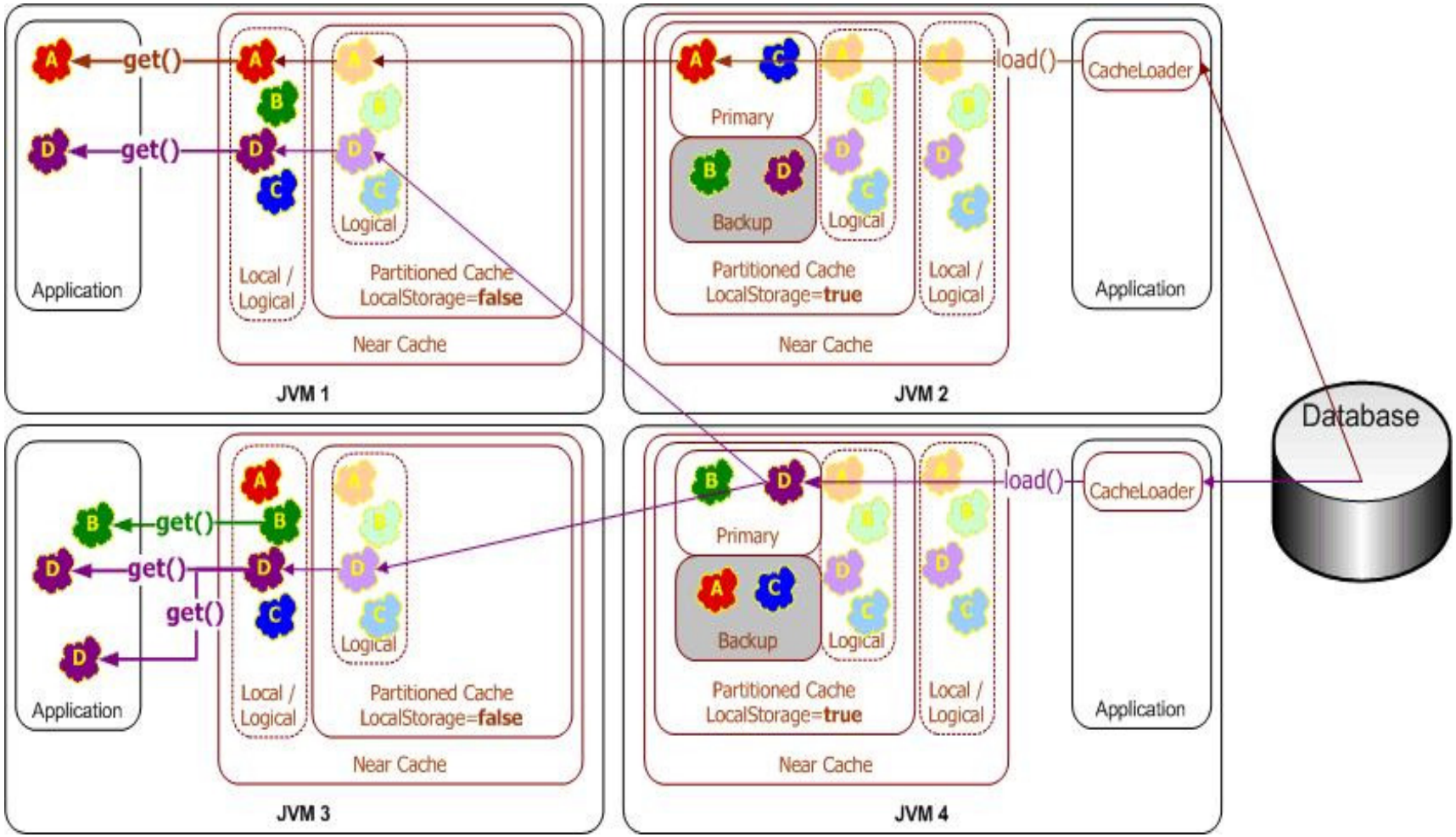


Architectural Integration Pattern

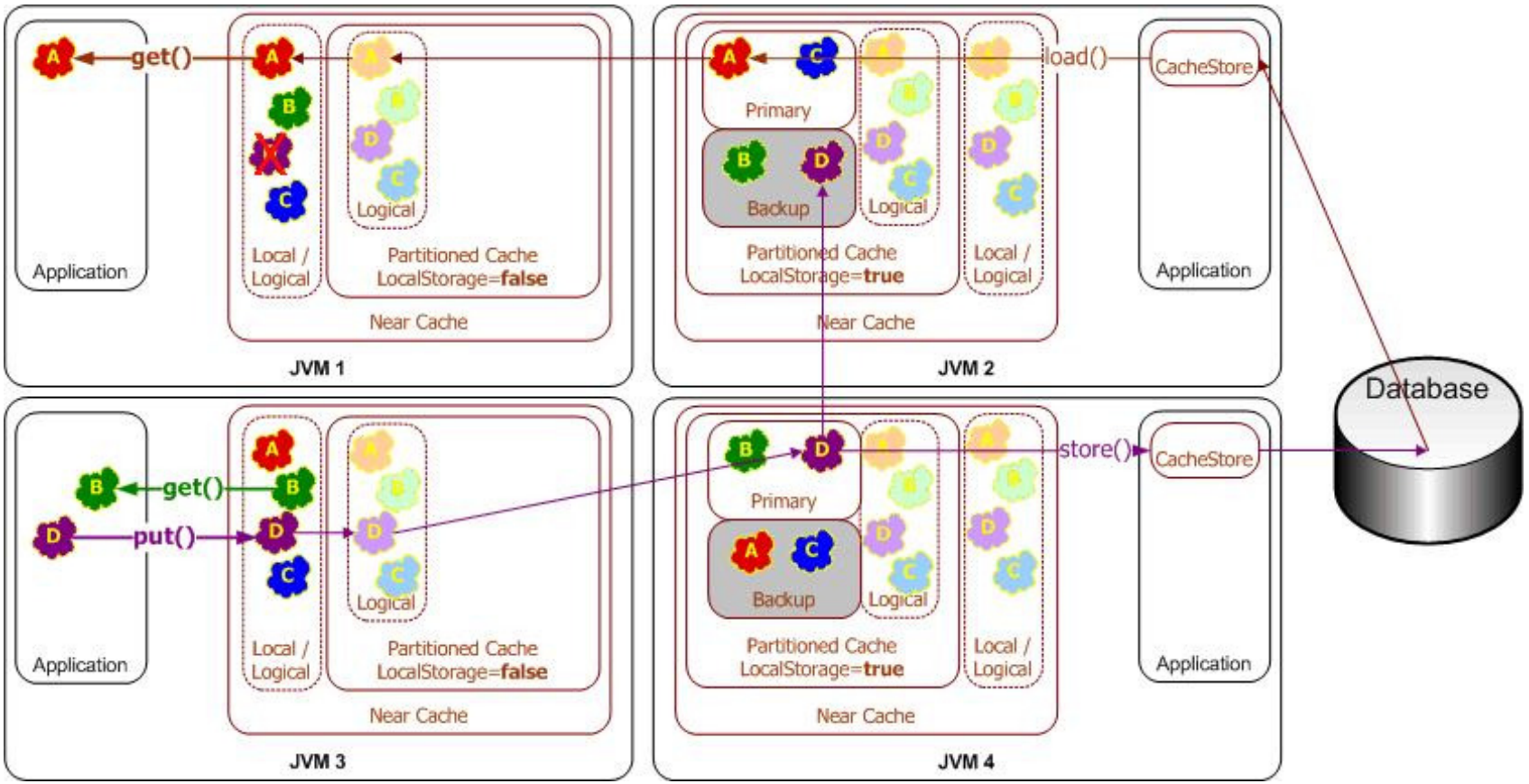
Data Source Integration (customizable per cache)

- **Read Through:** Read from CacheLoader when data not in grid
- **Write Through:** Write to CacheStore when data inserted, updated, removed in grid
- **Write Behind:** Asynchronous and coalesced updates to CacheStore when data inserted, updated, deleted in grid

Oracle Coherence Read-Through Caching

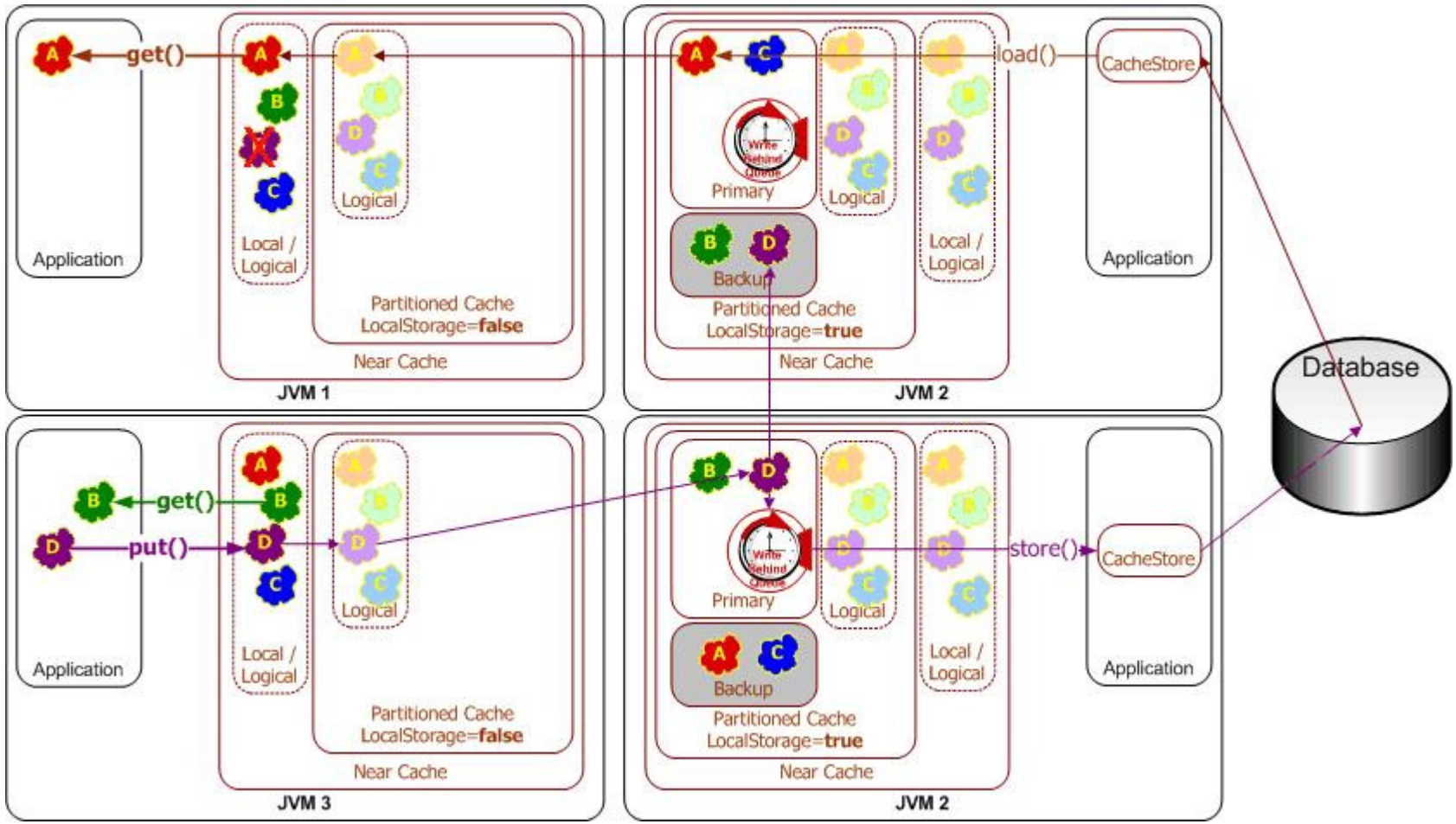


Write-Through Caching



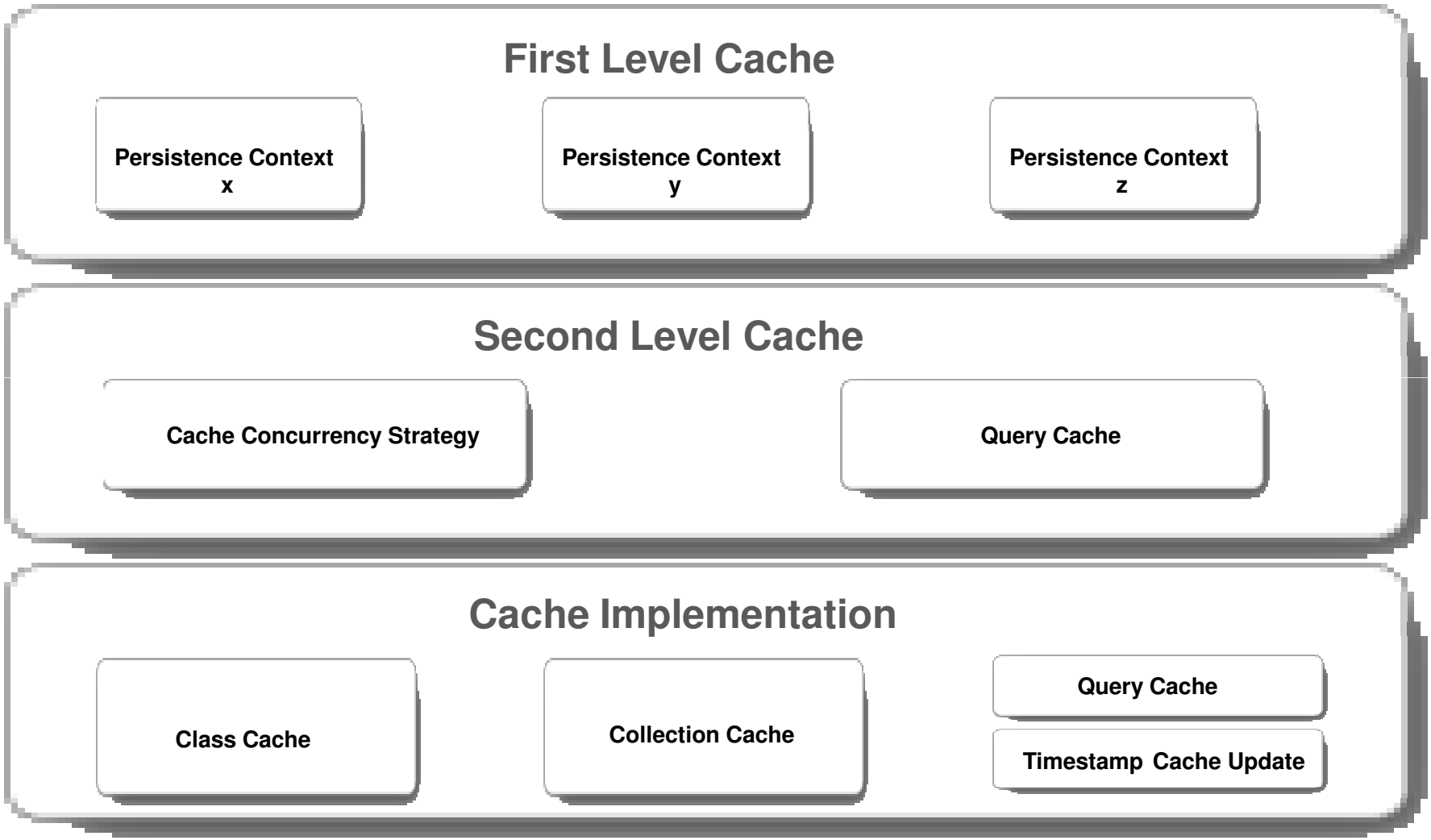
Write-Behind Caching

The result is a "read-once and write at a configurable interval" scenario



Cache Architekturschichten

Use Oracle Coherence as L2 cache for ORM



Integrating TopLink Grid and Coherence

Toplink - eclipseLink

- Using TopLink Grid as the CacheStore for Coherence
  - TopLink Grid provides a default entity-based CacheStore implementation, EclipseLinkJPACacheStore, and a corresponding CacheLoader implementation, EclipseLinkJPACacheLoader in the oracle.eclipselink.coherence.standalone package. These are optimized implementations for use with EclipseLink JPA. The classes and their Javadoc can be found in the toplink-grid.jar file.
  - Configuring an EclipseLinkJPACacheStore
- Using Coherence as the Toplink Cache
  - Coherence Shared L2 Cache - this configuration uses Coherence as the TopLink L2 (Shared) cache
  - Coherence Read - this configuration is optimal for entities that require fast access to large amounts of (fairly stable) data and must write changes synchronously to the database
  - Coherence Read/Write - this configuration is optimal for applications that require fast access to large amounts of (fairly stable) data and that perform relatively few updates

Integrating Hibernate and Coherence

Hibernate

Hibernate is an object/relational mapping tool for Java environments. The functionality in Oracle Coherence and Hibernate can be combined in several ways

- Using Hibernate as a CacheStore for Coherence
  - Coherence includes a default entity-based CacheStore implementation, HibernateCacheStore (and a corresponding CacheLoader implementation, HibernateCacheLoader) in the com.tangosol.coherence.hibernate package
- Using Coherence as the Hibernate L2 Cache
  - Coherence can be used as the L2 cache provider for Hibernate
  - Hibernate supports three primary forms of caching:
    - Session cache
    - L2 cache
    - Query cache

Integrating an Oracle Coherence CacheFactory with Spring

Coherence Transaction Framework (1)

- Provides ACID transaction guarantees across partitions and caches even in the event of failure
- Multi-Version Concurrency Control (MVCC)
- Supports the use of NamedCache operations, queries, aggregation, and entry processors within the context of a transaction
- Allows for consistent reads
  - Features
    - <transactional-scheme>
      - Allows for single statement (auto commit) transactions while using the standard Coherence API
    - Connection Based API
      - This API provides for more control over transactions
      - Allows for transactions that span multiple statements
      - Also introduces the OptimisticNamedCache interface which introduces functionality useful for optimistic transactions
    - XA Compliant Resource Adapter
      - Allows Coherence to participate in a distributed transaction as a fully compliant XA resource

Coherence Transaction Framework (2)

- **Transactions across caches**
  - Caches must be running on the same Distributed Service
- **Cache size overhead for MVCC**
  - MVCC stores additional versions of Cache values
  - Details and sizing directions in the documentation

```
Connection con = new DefaultConnectionFactory(). createConnection("TransactionalCache");
con.setAutoCommit(false);
try {
   OptimisticNamedCache cache = con.getNamedCache("MyTxCache");
   cache.insert(key, value);
   cache.insert(key2, value2);
   con.commit();
catch (Exception e) {
   con.rollback();
   throw e;
}
finally {
   con.close();
}
```

Coherence Query Language (CohQL)

- Coherence Query Language is a new light-weight syntax (in the tradition of SQL) that is used to perform cache operations on a Coherence cluster. The language can be used either programmatically or from a command-line tool
- CohQL is a Command Line Tool with SQL like Language
- Programmatic Filter and ValueExtractor building API that is similar to the SQL WHERE clause

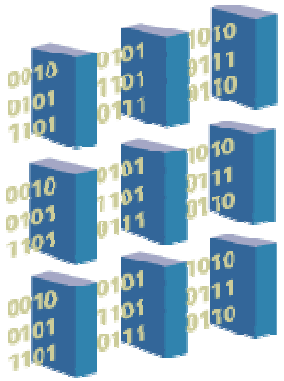
```
CohQL> select min(zipCodes.size()), min(zipCodes.size()) from cityinfo
Results:
0 , 0
CohQL> delete from cityinfo where zipCodes.size() is 0 or areaCodes.size() is 0
COHQL select areaCodes from cityinfo where name is “Dallas”
Results:
{214,972}
CohQL> update cityinfo set areaCodes = {214, 469, 972} where name is “Dallas” and
state is “TX”
```

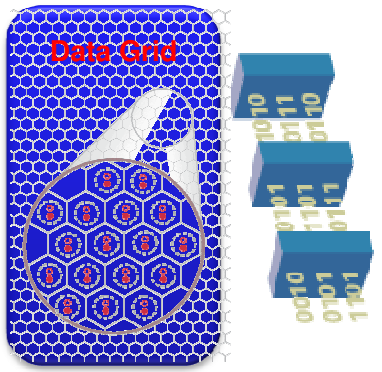
Einsatzfälle für Coherence

- Coherence*Web industrial session management
  - Already widely deployed (up to more than 100 servers) by leading e-tailers, travel sites, portals and banks
  - Out-of-the-box seamless integration with WebLogic Server and WebCenter Portal
  - Significant performance jump, particularly for large clusters, large sessions and session expiry
- Enabling Tera-Scale Data Grids
  - Support for multi-terabyte data grids, including configurable off-heap storage
  - Significant reduction in the number of JVMs required to support large data sets
  - Efficient querying and processing of massive data sets without de-serialization
- Ensuring continuous availability
  - New network dynamic configurability for capacity-on-demand Cloud environments
  - Guardian: A safeguard within each JVM that automatically detects and corrects fault
- Example repository of advanced patterns that leverage Coherence
  - Highlights Data Grid capabilities

Nahtlose Konsolidierung

Application Server Cluster with multiple JVMs / Managed





Application Server Cluster with less members and data grid cluster connected

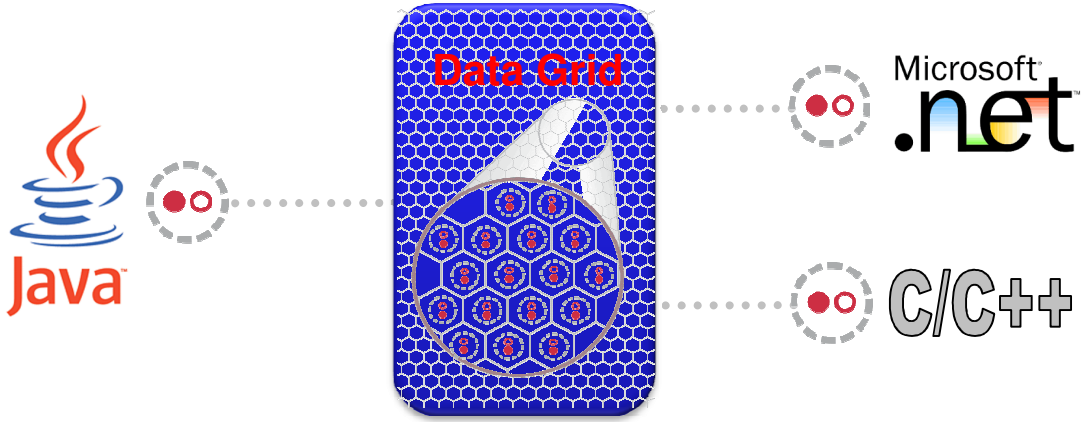
Challenges

- Application data can overwhelm available server memory.
- Adding more application servers only compounds problem.
  - Each app server requires fixed overhead.
  - Memory management issues.

Benefits

- Keep application data outside the application server.
- Instant 25% increase of users per WebLogic Server application.
- Scale application data linearly thru commodity hardware.
- Increase availability thru data redundancy.

Objekt Interoperabilität



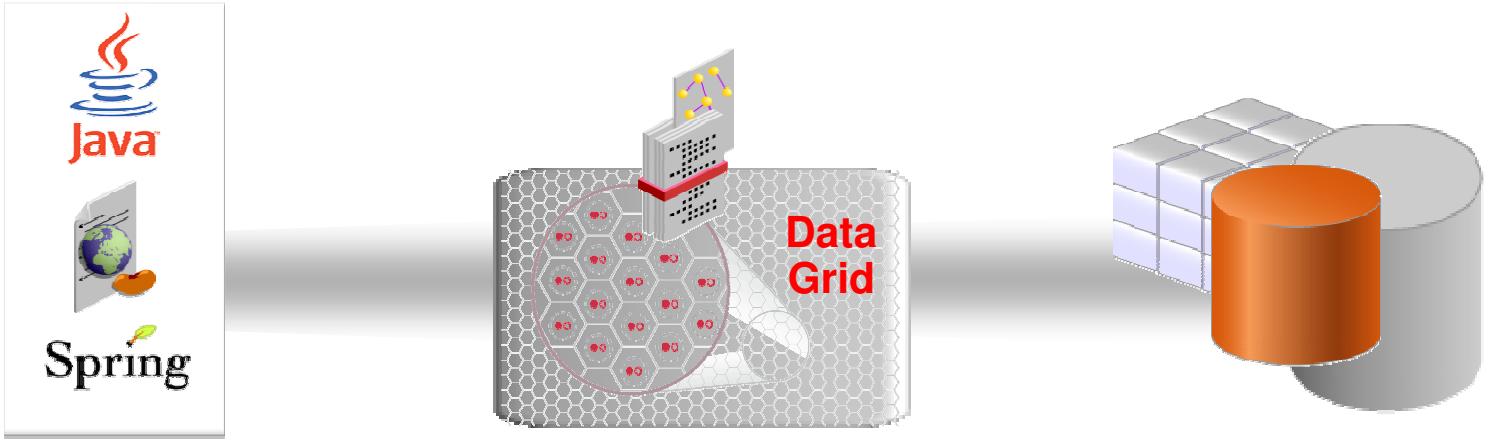
Challenges

- Share objects/behavior between different programming languages
- Web Services are good in interoperability bad in performance

Benefits

- Faster than Java and .Net serialization
- Pure .Net client (a single .DLL is needed)

Enhanced Data Mapping



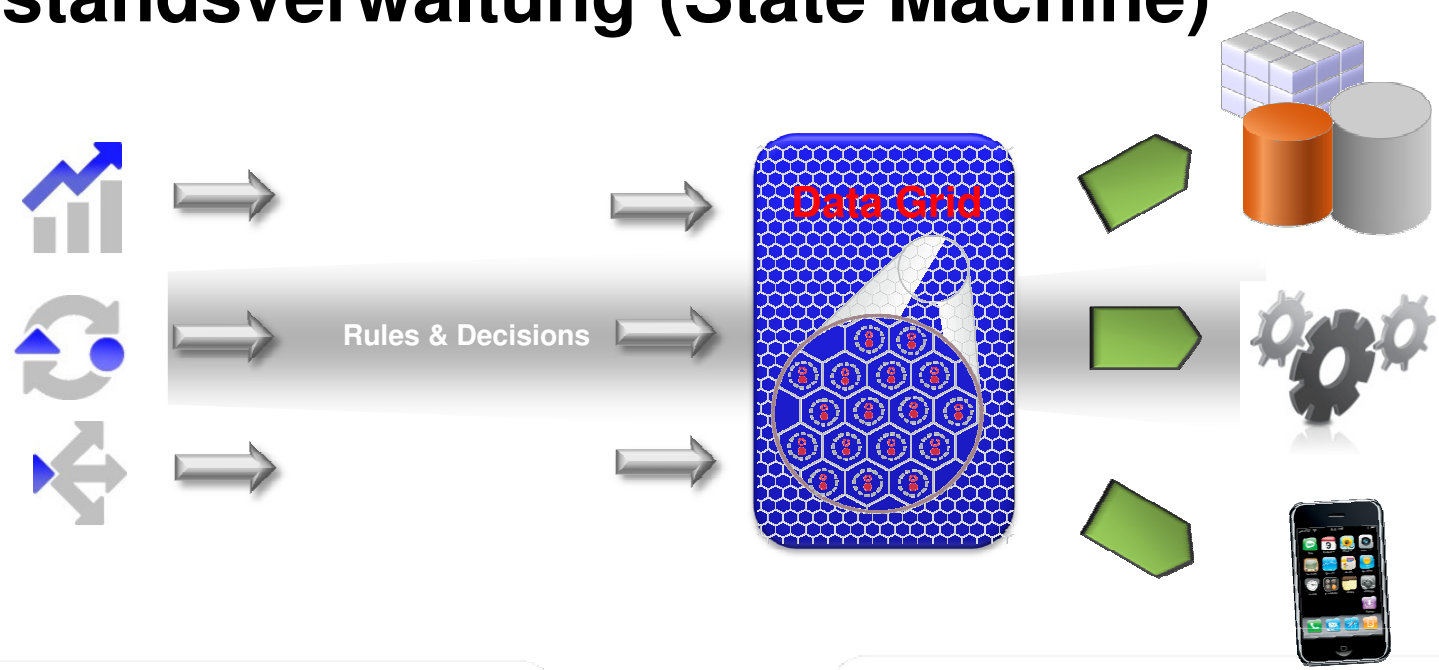
Challenges

- Object/Relational frameworks do not always generate optimal queries.
- Developer productivity increases at expense of performance and scale.
- Each framework solution has different caching abilities- not all cluster aware.

Benefits

- **Integrate O/R frameworks with Data Grid.**
- **TopLink pre-integrated- no code changes required.**
- **Support for Spring and Hibernate through interceptors.**

Zustandsverwaltung (State Machine)



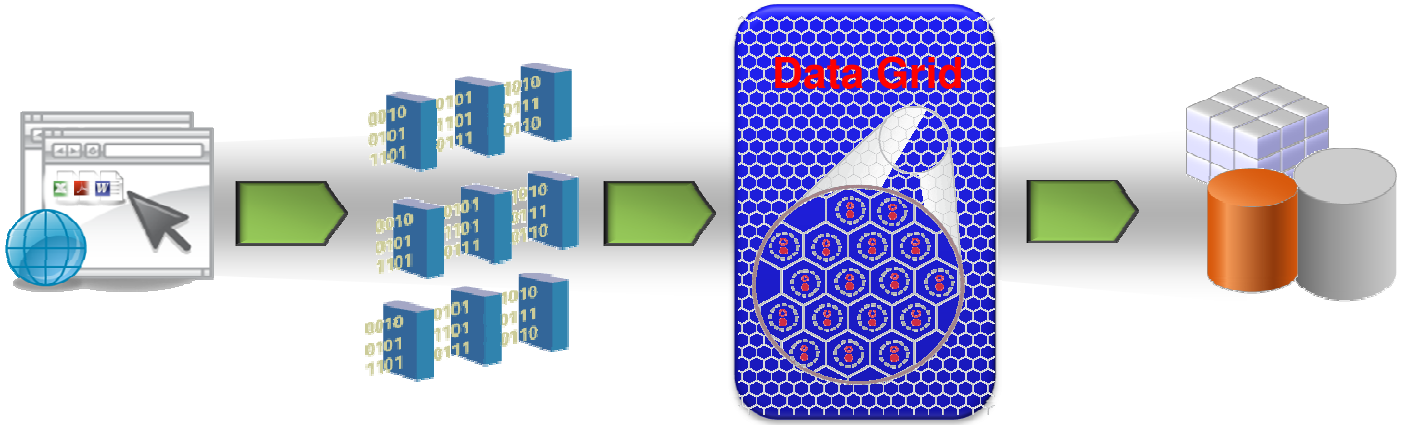
Challenges

- Keep a memory representation in real-time of the complete map of a logistics/transport network using commoditized infrastructure (hardware) and standard software

Benefits

- **Data grid layer to support low latency apps able to update the map in realtime**
- **100% transactional integrity and data reliability**
- **Application Grid for high availability and eXtreme transaction processing**

Skalierbarkeit für Web-Applikationen



Challenges

- Web Applications needs to be able to accept more load on a very short time

Benefits

- **Data grid layer to support low latency apps**
- **100% transactional integrity and data reliability**
- **Application Grid for high availability and eXtreme transaction processing**

Zusammenfassung: Vorteile von Oracle Coherence

Coherence In-Memory Data Grid ist ein Daten-Management-System für gemeinsam genutzte Anwendungs-Objekte auf mehreren Servern, mit geringer Reaktionszeit, sehr hohem Durchsatz, vorhersehbarer Skalierbarkeit, kontinuierlicher Verfügbarkeit und Zuverlässigkeit

Leistung

- **Extreme, vorhersagbare Reaktionsleistung**
- **Unbegrenzte lineare Applikationsskalierbarkeit**

Ressourcen-Management

- **Bessere Ausnutzung von Standard-Hardware und anpassbarer Applikations-Infrastruktur**
- **Rechenzentrums-Nutzen maximieren mit dynamischen SLAs & Ressourcen-Management**

Vielen Dank für Ihre Aufmerksamkeit!

Wolfgang.Weigend@oracle.com

ORACLE®