Persistence

Moving the Boundaries of Information

Data Services Architektur

Verbesserung der Performanz und Skalierbarkeit von verteilten Systemen

Rolf Knoll
Senior Technical Account Manager/Central Europe
Persistence Software
Rolf.Knoll@persistence.com

Agenda

- Überblick : Persistence Software
- Data Services Architektur (DSA)
  - Motivation
  - Produkt Portfolio
  - Features
  - Entwicklungsprozess
- Anwendungsfälle
- F&A

Persistence

Persistence Software

- Gegründet 1991
- Über 200 Kunden der Global 2000
  - Performanz
  - Kostenreduktion
  - Technologie erprobt/bewährt
- IBM Advanced Partner
- BEA Three Star Partner

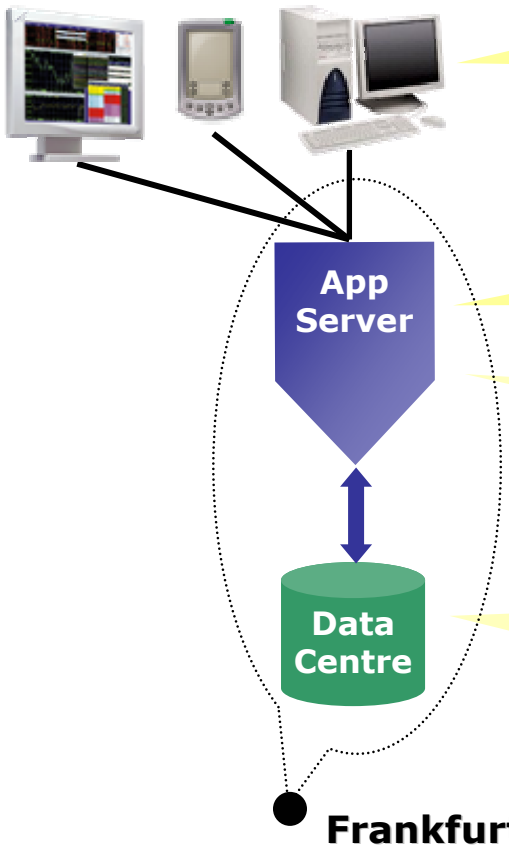
Persistence

Moving the Boundaries of Information

Data Services Architektur

Eine Architektur für verteilte Systeme unterschiedlicher Technologie und Sprachumgebungen zum Umgang mit Anwendungsdaten, die sich unvorhergesehen ändern.

Architekturszenarien



Client
- Applikationen (Java, VB, C#)
- Web-FrontEnds
- Mobile Devices

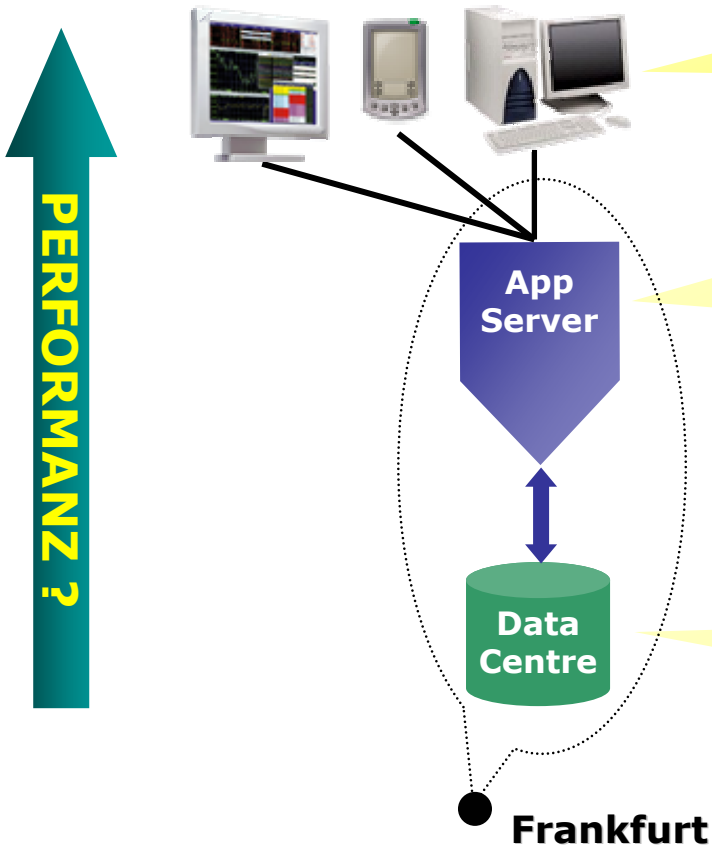
Technologie
- J2EE
- CORBA
- .NET

Sprache
- JAVA
- C++

RDBMS
- Oracle
- DB/2
- Sybase
- SQL Server
- …

Persistence

Architekturszenario I
- Kritische Punkte



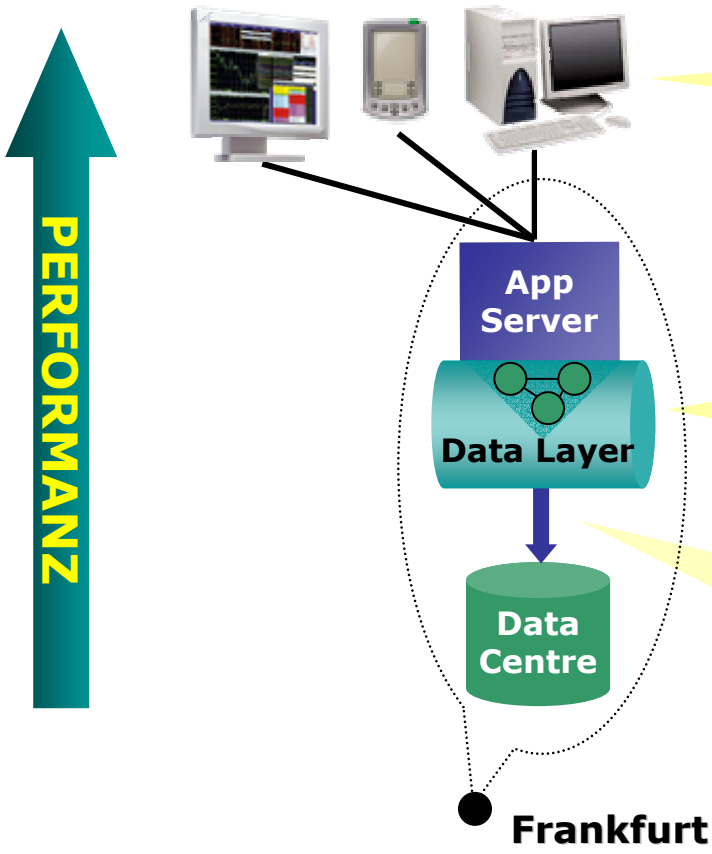
Werden Antworten auf dem Client schnell genug präsentiert ?

Werden vernetzte Geschäftsdaten auf dem Applikations-server schnell genug zur Ver-fügung gestellt ?

Werden DB Anfragen schnell genug abgearbeitet ?

Persistence

Architekturszenario I
- Lösung : Data Services



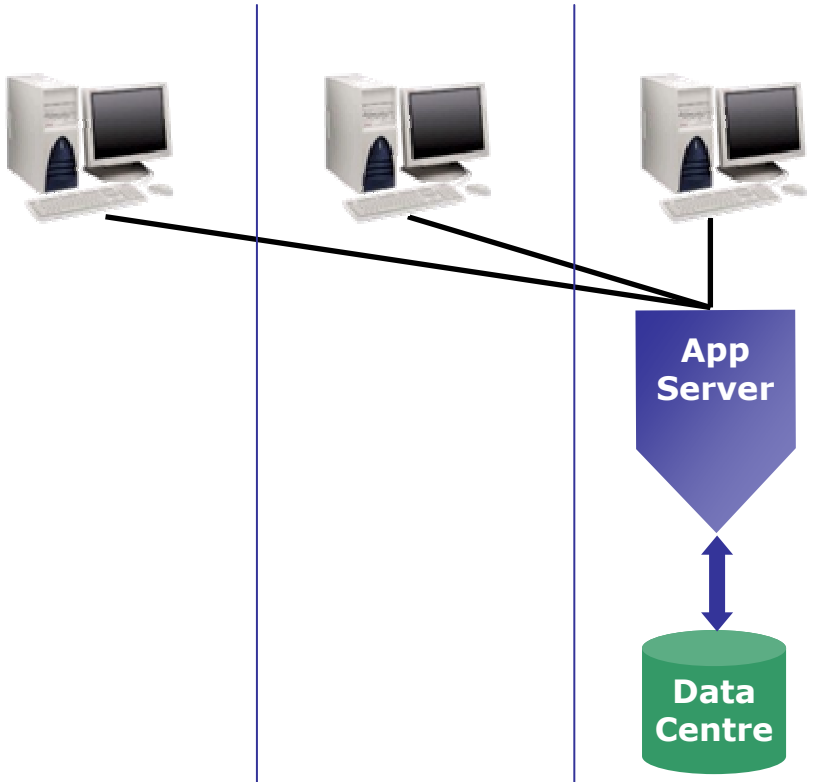
Antworten werden auf dem Client schneller präsentiert

Dem Applikations-server werden vernetzte Objektstrukturen in-memory zur Verfügung gestellt

Das Kommunikations-aufkommen zwischen App. Server und DB wird reduziert

Persistence

Architekturszenario II
- Kritische Punkte



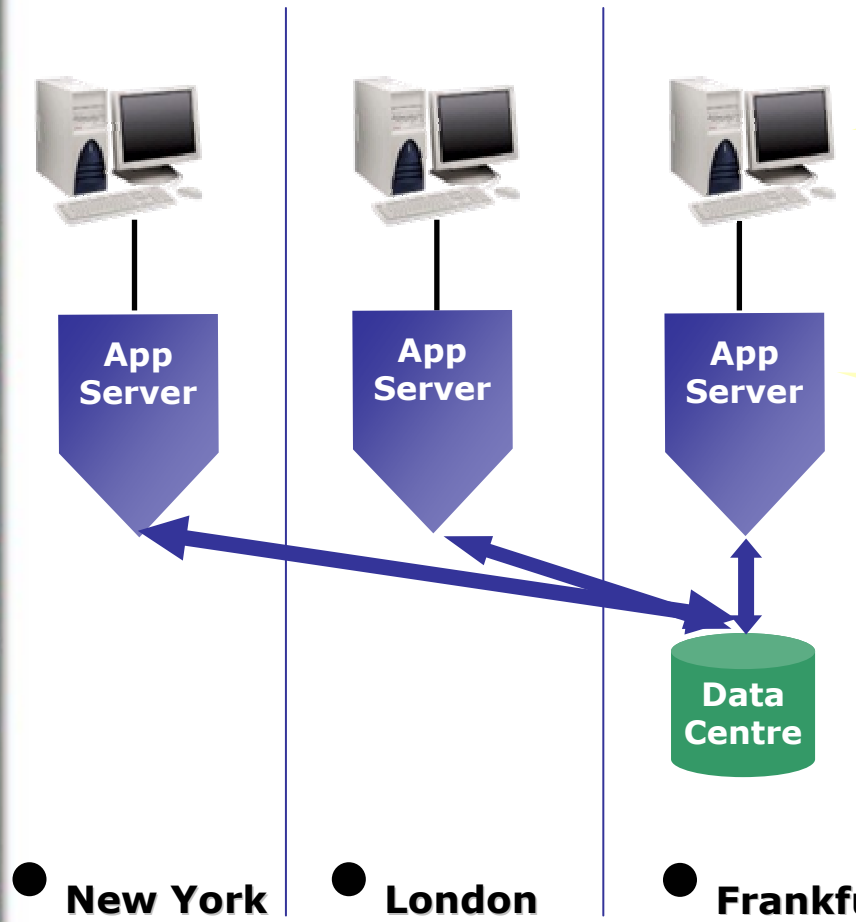
Können weit entfernte Clients mit identischen Antwortzeiten angeschlossen werden ?

- New York
- London
- Frankfurt

Skalierbarkeit ?

Persistence

Architekturszenario II
- Kritische Punkte



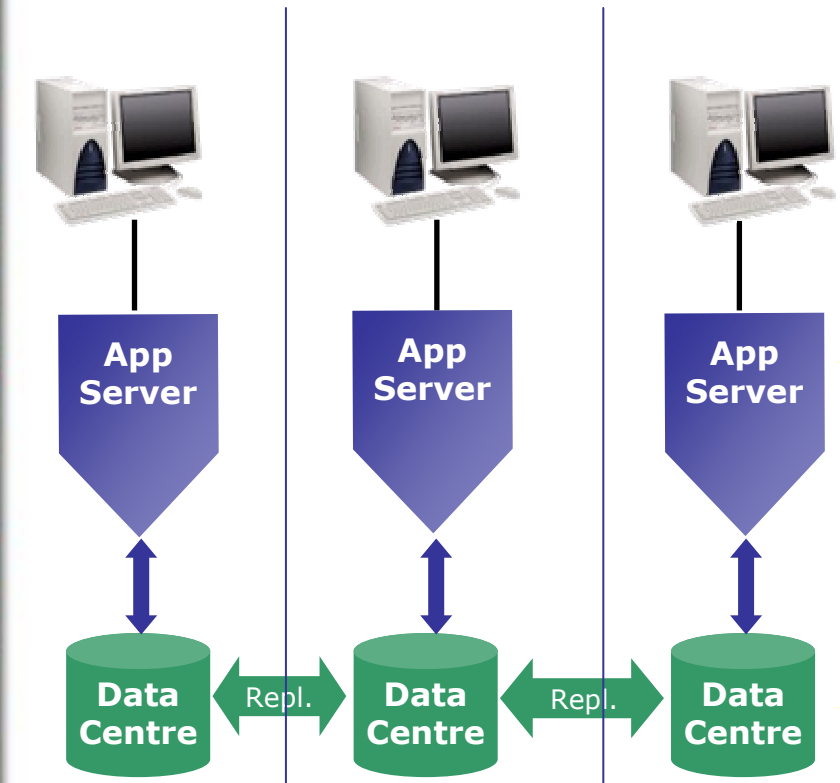
Können weit entfernte Clients mit identischen Antwortzeiten angeschlossen werden ?

Muss die Anwendungslogik verteilt zur Verfügung gestellt werden ?

Skalierbarkeit ?

Persistence

Architekturszenario II
- Kritische Punkte



Können weit entfernte Clients mit identischen Antwortzeiten angeschlossen werden ?

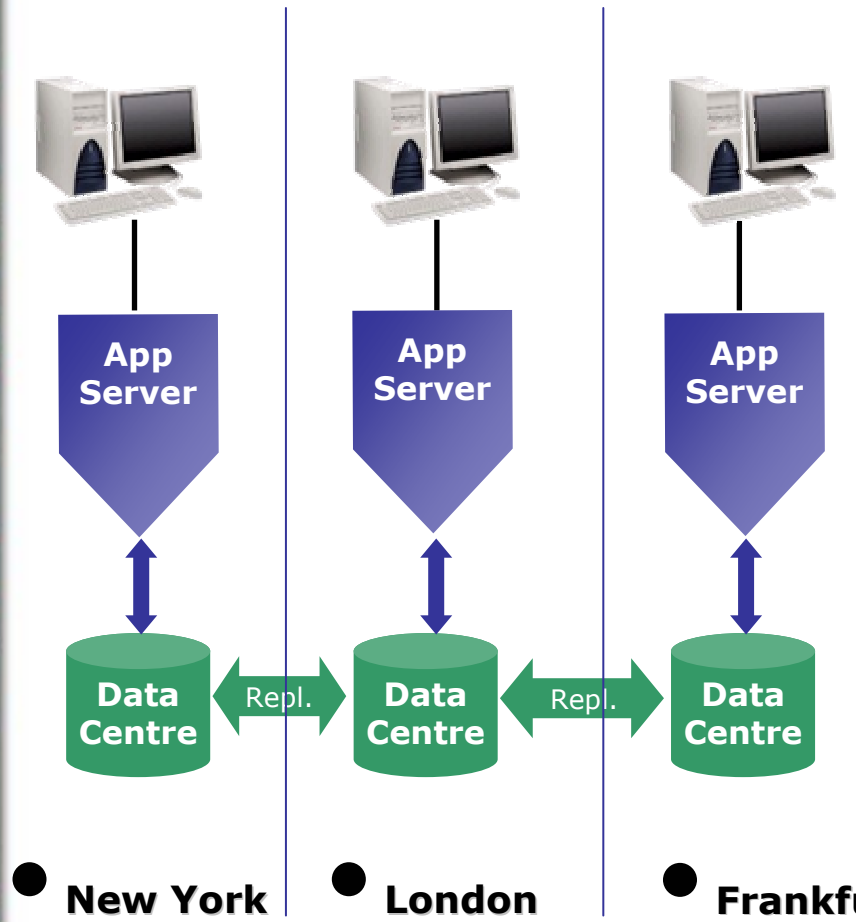
Muss die Anwendungslogik verteilt zur Verfügung gestellt werden ?

Werden replizierte Datenbestände benötigt ?

Skalierbarkeit ?

Persistence

Architekturszenario II
- Kritische Punkte



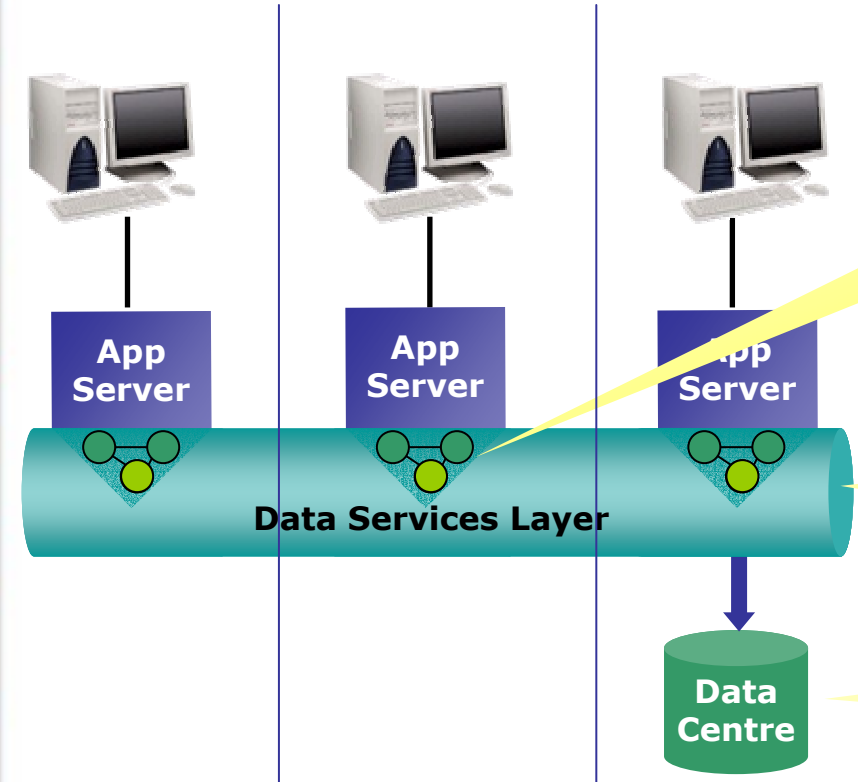
Zusammengefasst:

- Hohe Investitionskosten
- Hohe Betriebskosten
- Komplexe Gesamtarchitektur
- DB Replikation ist langsam und unzuverlässig
- Benutzung veralteter Daten nicht ausgeschlossen
- Systemerweiterung ist aufwendig

- **Dies ist keine brauchbare Lösung für den Unternehmenseinsatz**

Persistence

Architekturszenario II
- Lösung : Data Services



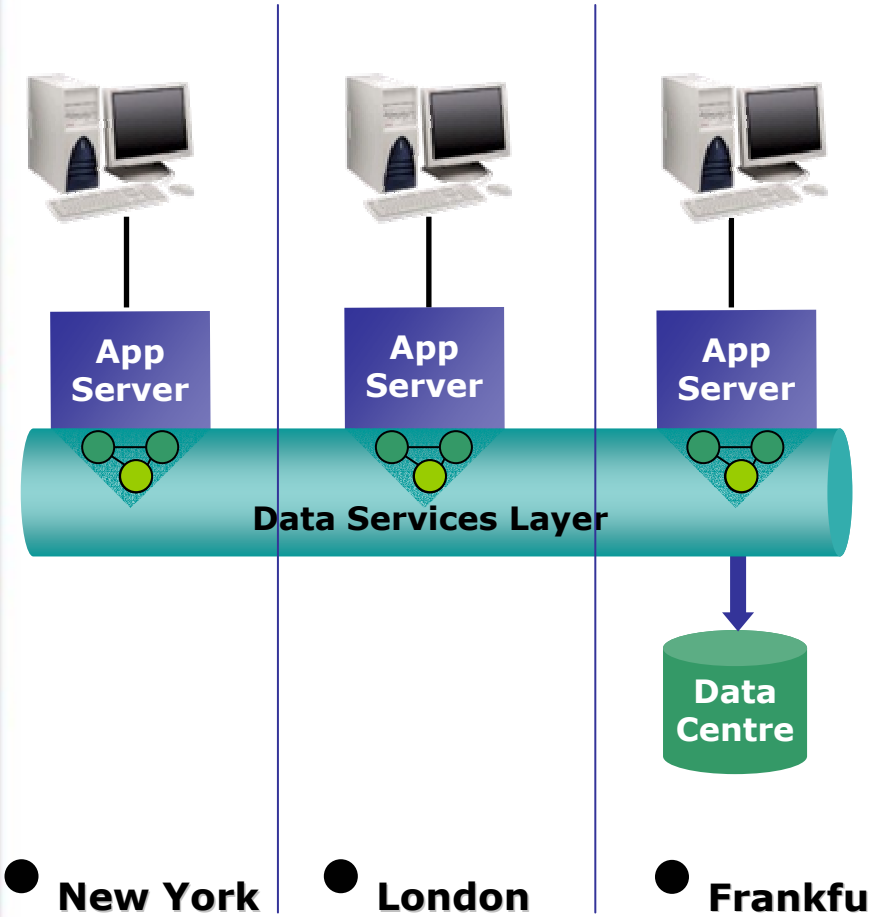
Objektstrukturen werden verteilt zur Verfügung gestellt

Objektänderungen werden durch asynchrones Messaging übertragen

Replizierte Datenbestände werden nicht mehr benötigt

Persistence

Architekturszenario II
- Lösung : Virtual Data Center



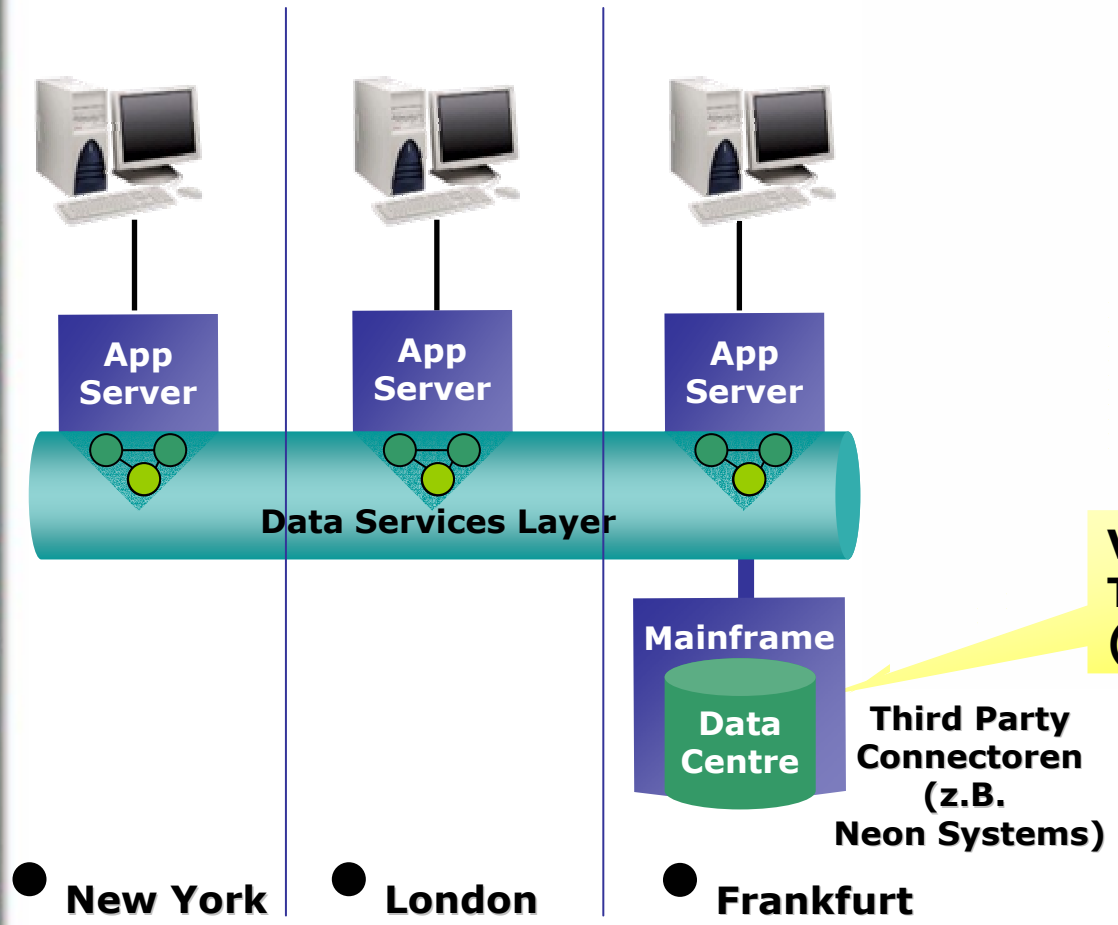
Zusammengefasst:

- Erhebliche Kostensenkung
  - bei Erstinstallation
  - und Betrieb
- Geschäftsobjekte sind mit minimaler Verzögerung im gesamten Unternehmen verfügbar
- Unternehmensweite Datenintegrität wird sichergestellt
- Gesamtarchitektur ist vereinfacht
- Erweiterbarkeit und Skalierbarkeit ist erhöht

- **Die Unternehmenskritischen Informationen sind verfügbar, wo sie benötigt werden**

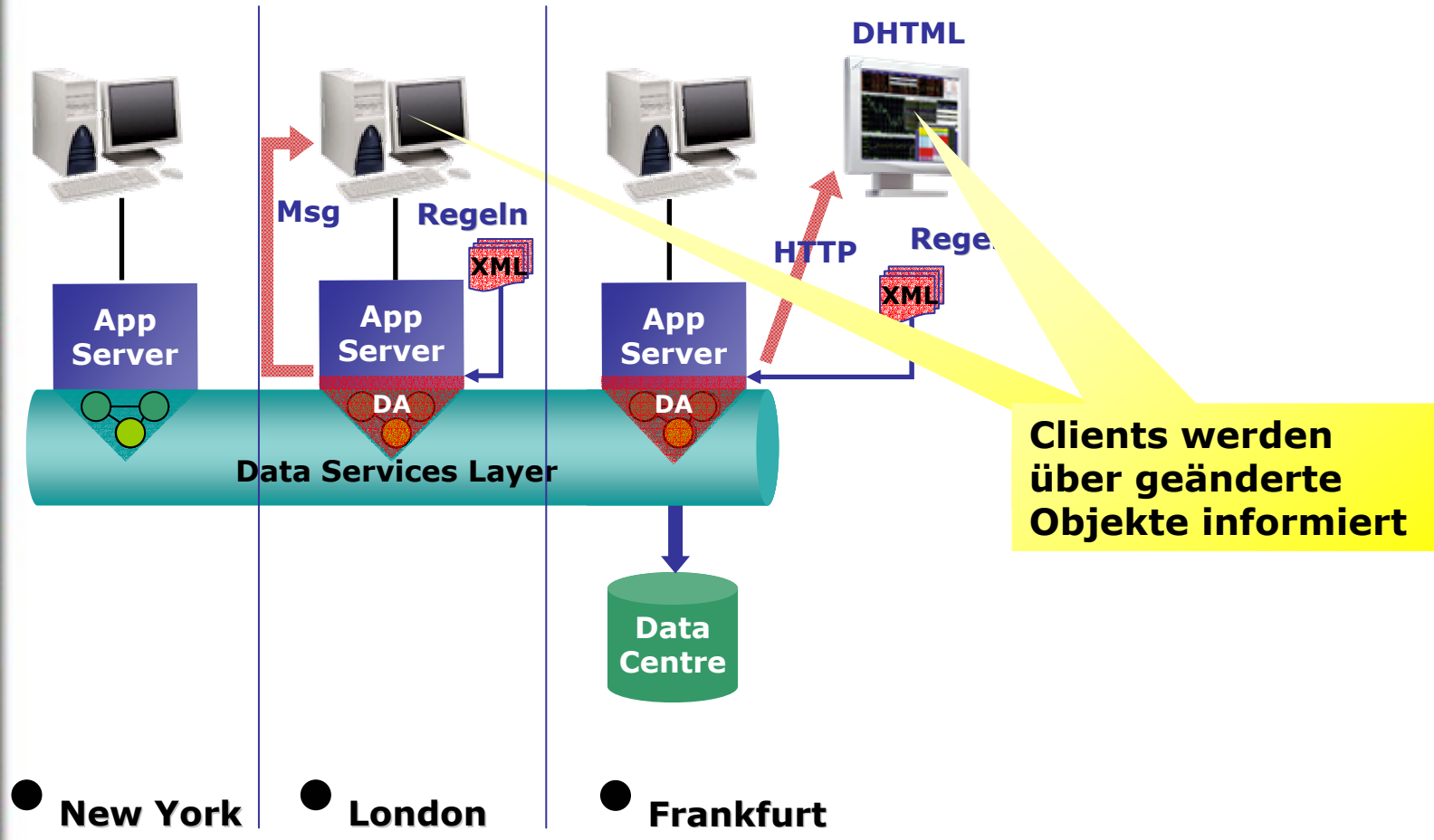
Persistence

Architekturoptionen
- Integration Datenquellen



Persistence

Architekturoptionen
- Notifizierung von Clients

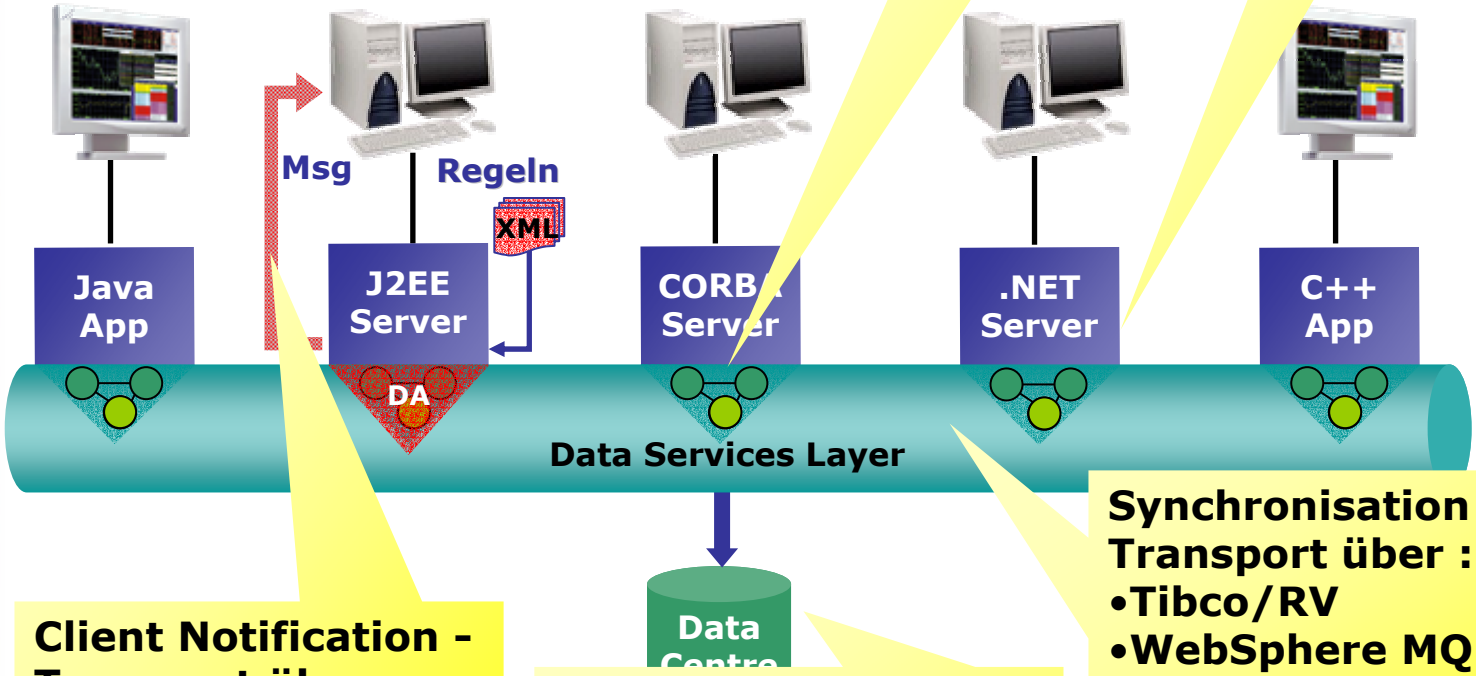


Persistence

Produkt Portfolio
- Plattformen

Middleware :
- J2EE
- CORBA
- .NET

Sprache :
- Java
- C++



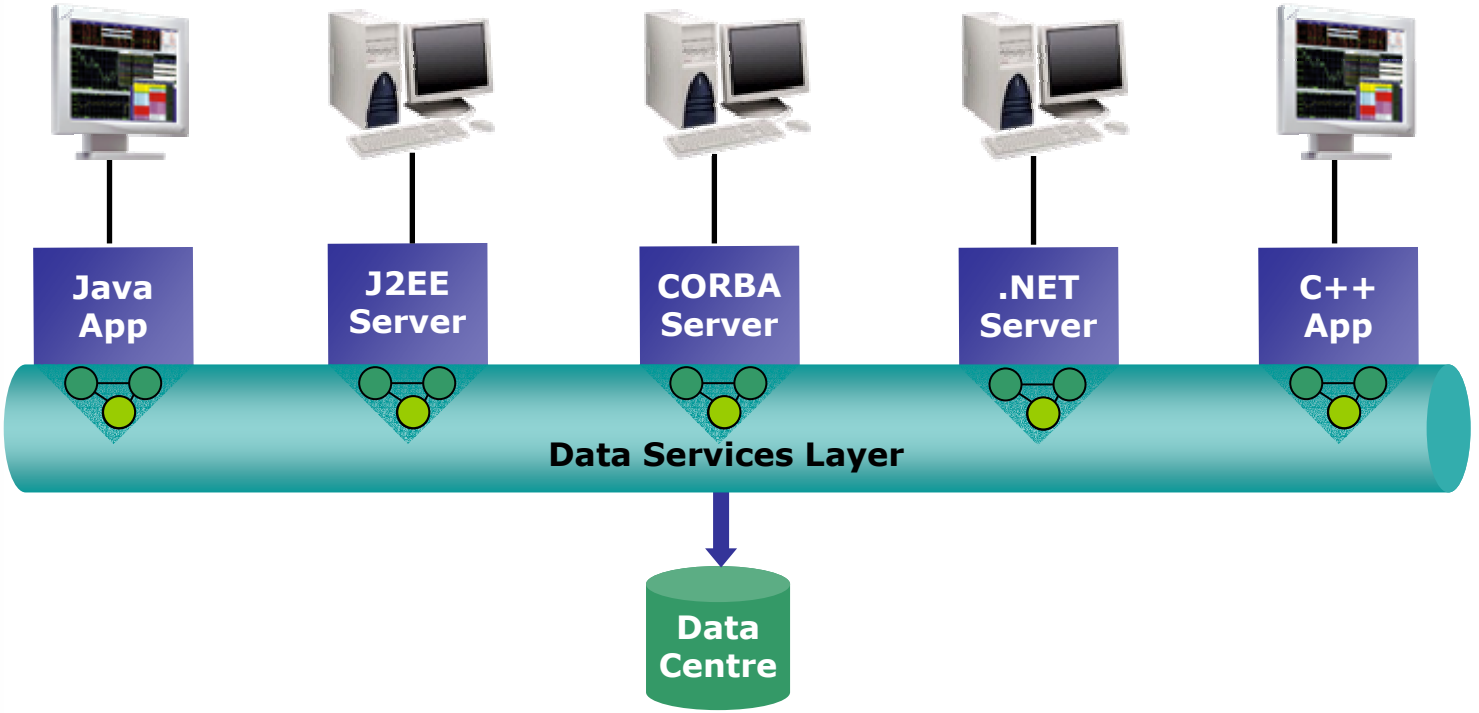
Client Notification - Transport über :
- Tibco/RV
- JMS
- Sockets
- benutzerdefiniert

DBMS :
- Oracle
- Sybase
- DB/2
- SQL Server
- ... (ODBC 3.0)
- IMS DC/CICS (über Neon Systems)

Synchronisation - Transport über :
- Tibco/RV
- WebSphere MQ
- Oracle AQ
- JMS
- Sockets
- benutzerdefiniert

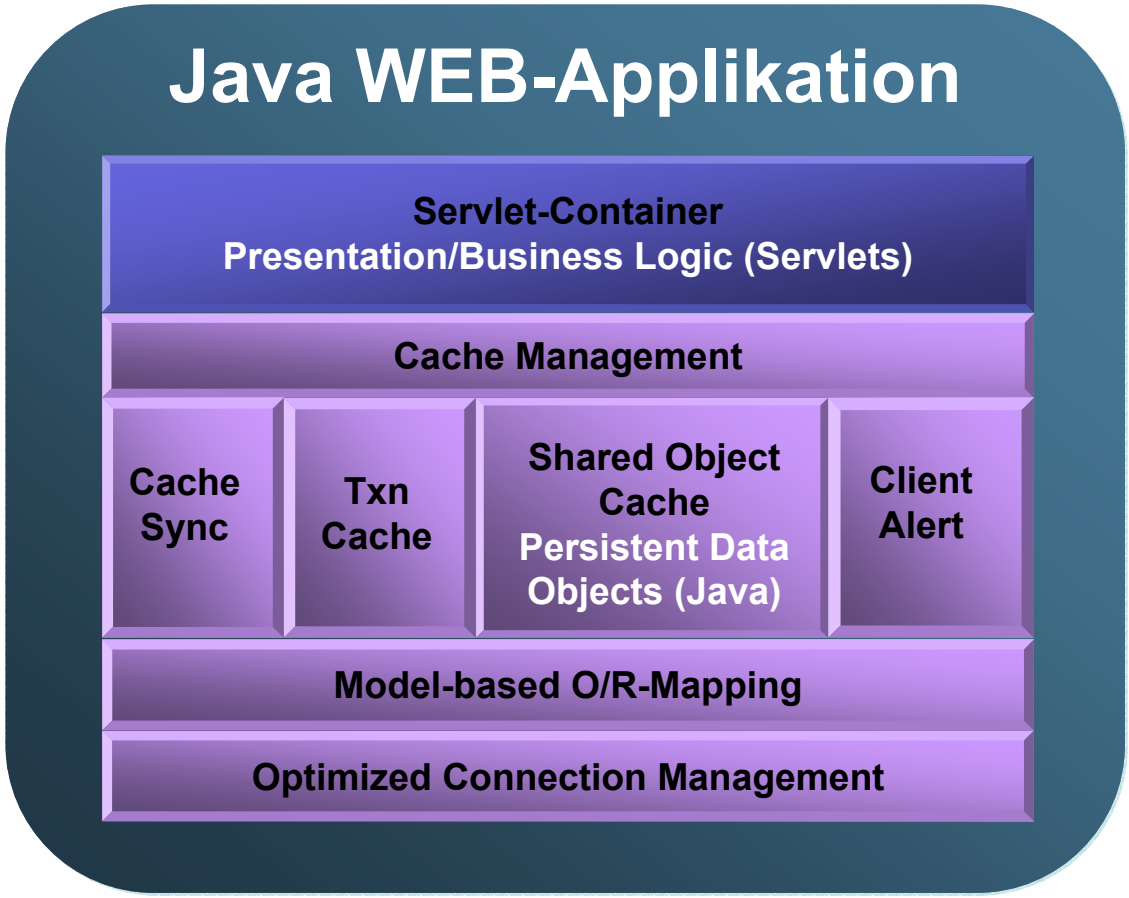
Persistence

Produkt Portfolio
- EdgeXtend



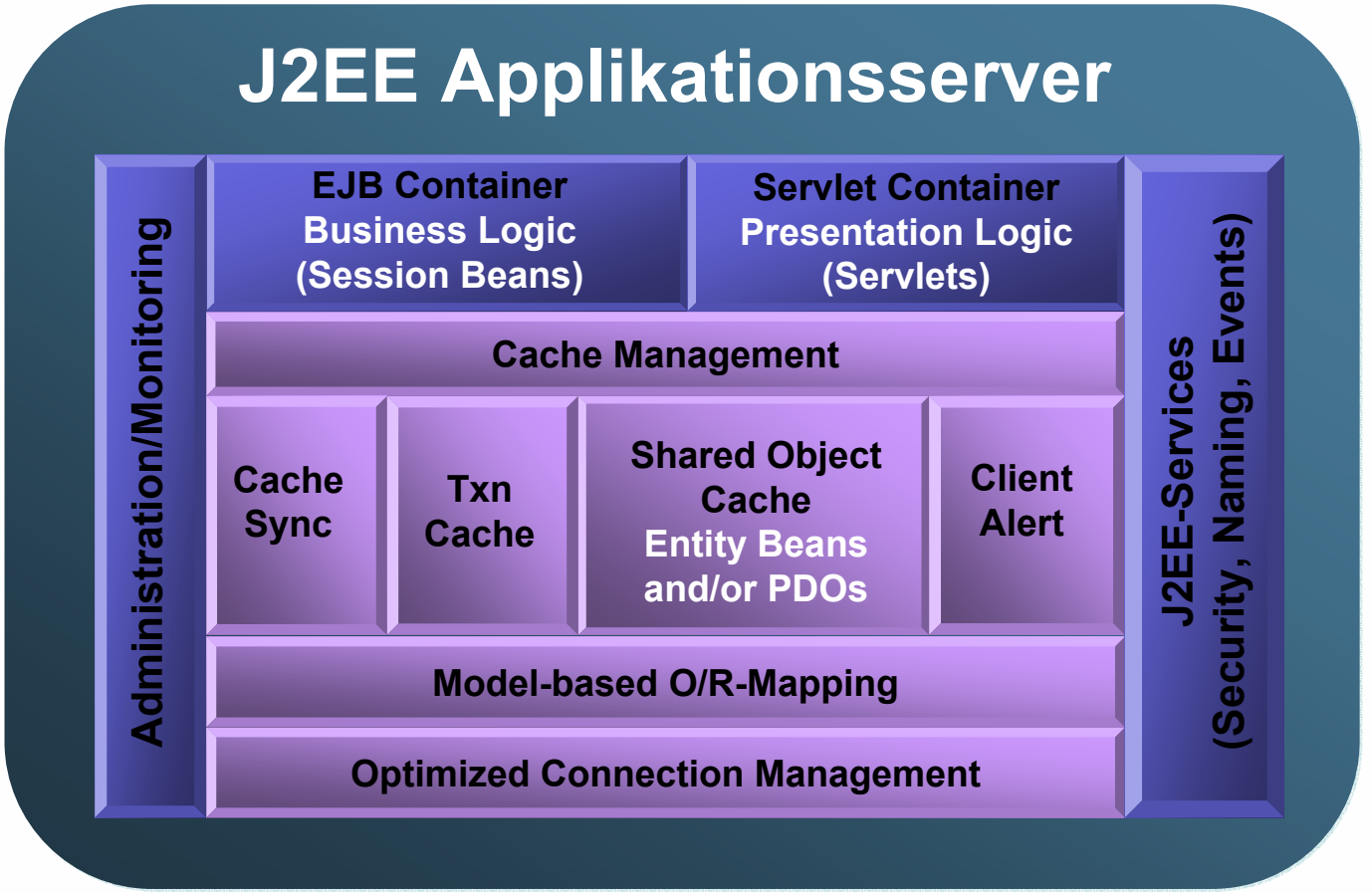
Persistence

Produkt Portfolio
- EdgeXtend for Java



Persistence

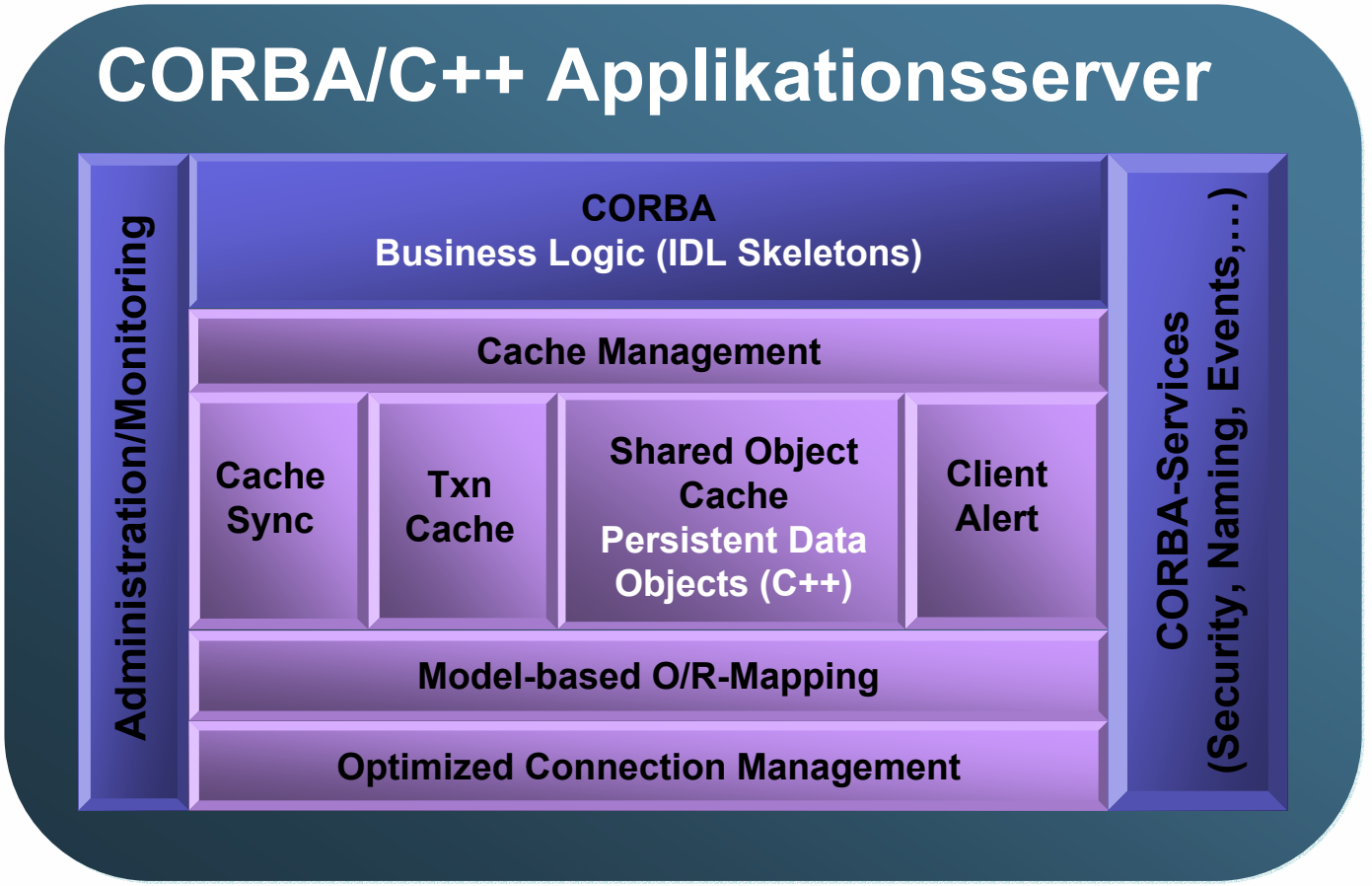
Produkt Portfolio
- EdgeXtend for J2EE



Unterstütze Applikationsserver

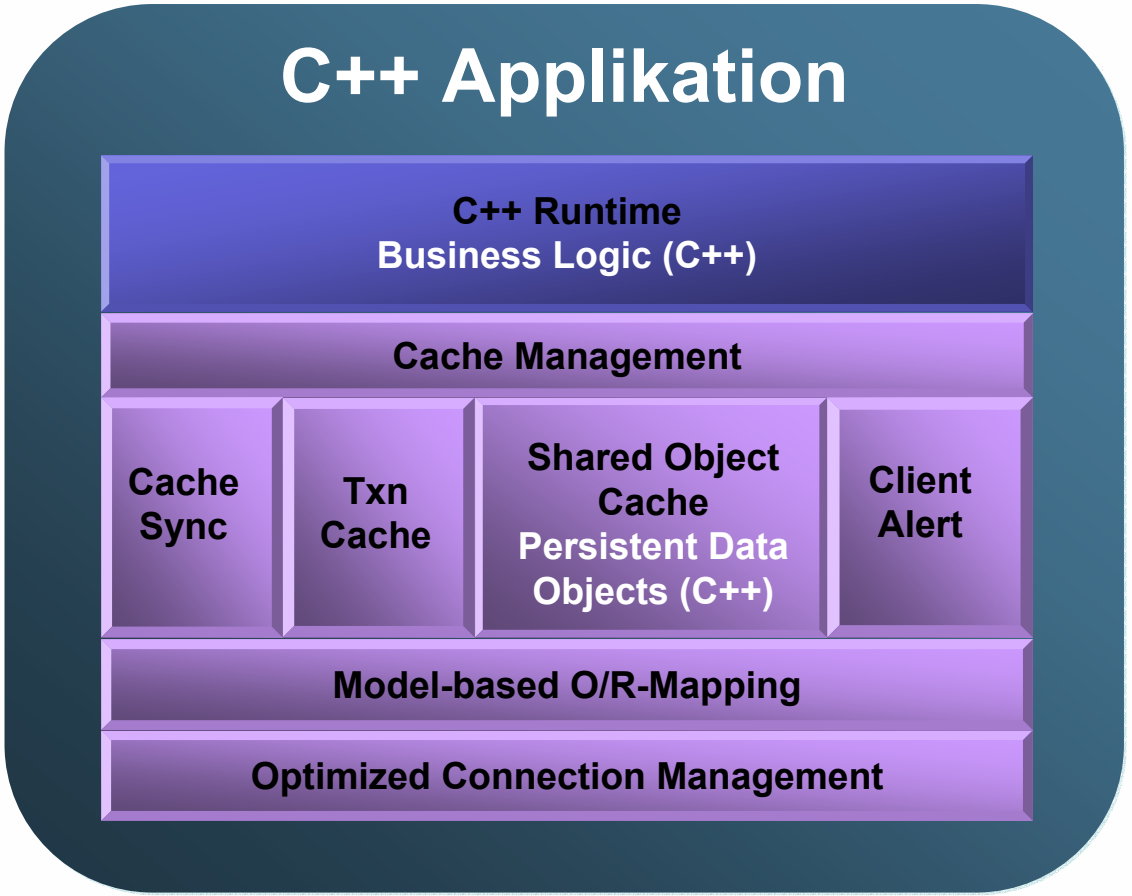
- **IBM WebSphere**
- **BEA Weblogic**

Produkt Portfolio
- EdgeXtend for C++/CORBA



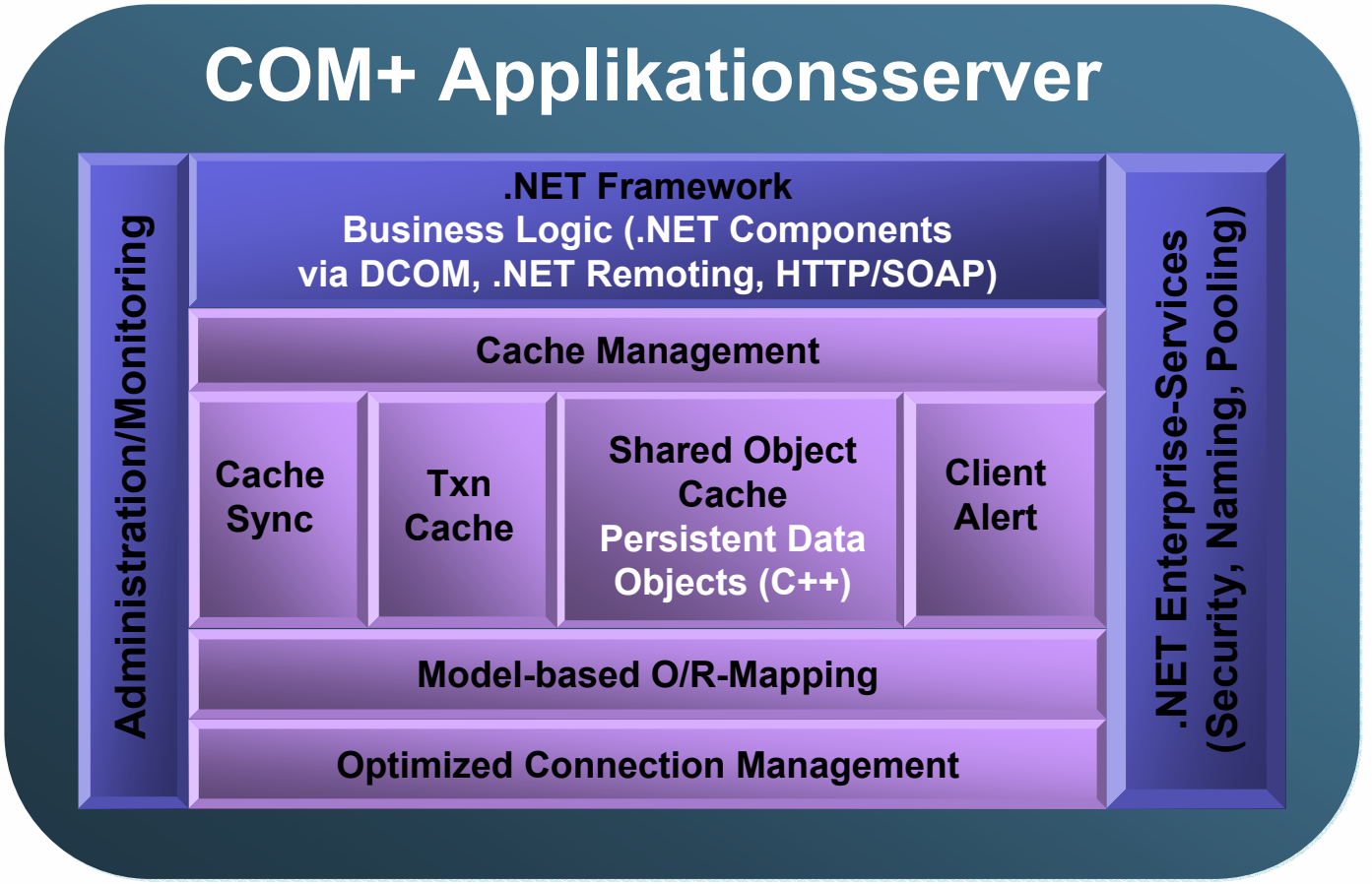
Persistence

Produkt Portfolio
- EdgeXtend for C++



Persistence

Produkt Portfolio
- EdgeXtend for .NET



Persistence

Persistence

Moving the Boundaries of Information

Data Services Architektur : Technischer Überblick

Caching (auch transaktional)

Cache-Synchronisation

Notifikation von Clients

O/R Mapping

Optimierte DB-Zugriffe

Performanz & Skalierbarkeit

Transaktions-management: zuverlässige Objekt-zustände

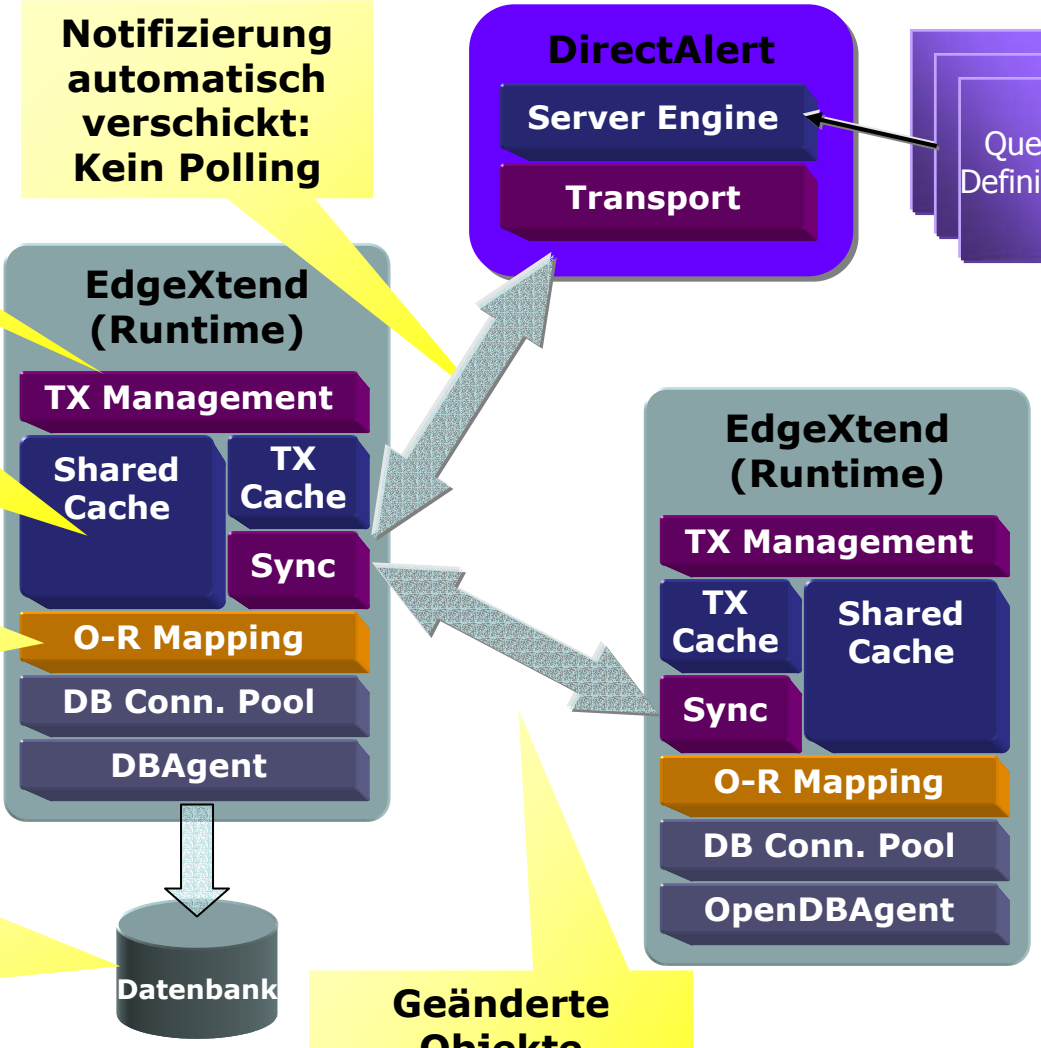
Direkter Zugriff auf Objekte im Speicher = massive Performanz-steigerung

Generiertes Mapping: Transparente Integration Objekt/RDBMS

Datenbank Optimierung:

- Native Treiber
- Ressourcen Pooling
- Locking (Opt/Pes)
- Schreibmodi (WI/WOC/WN)
- Routing

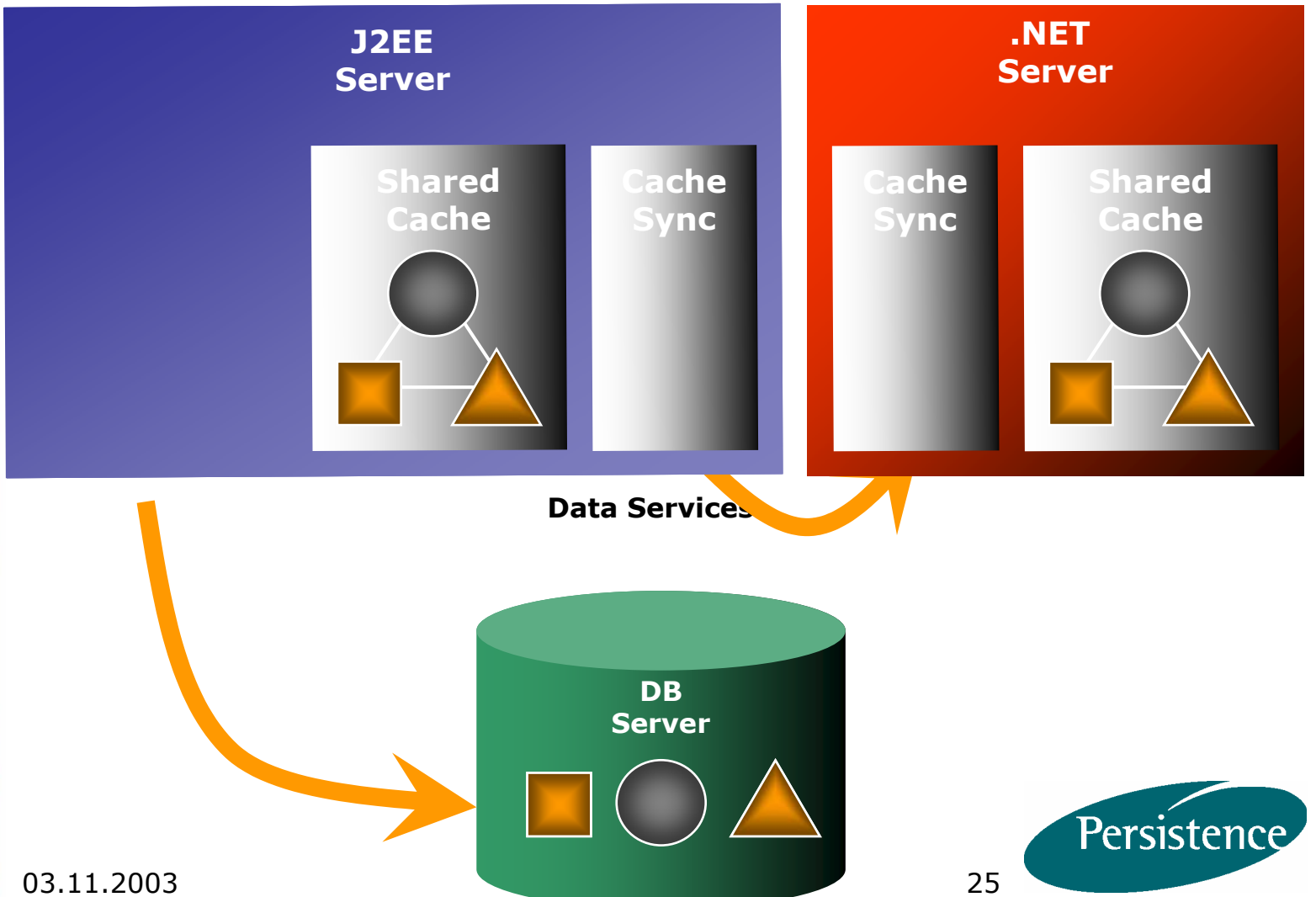
Notifizierung automatisch verschickt: Kein Polling



Geänderte Objekte automatisch synchronisiert: Kein DB-Zugriff

Persistence

Data Services
- Funktionsübersicht Caching



Persistence

Persistence

Moving the Boundaries of Information

Data Services Architektur
Technische Details

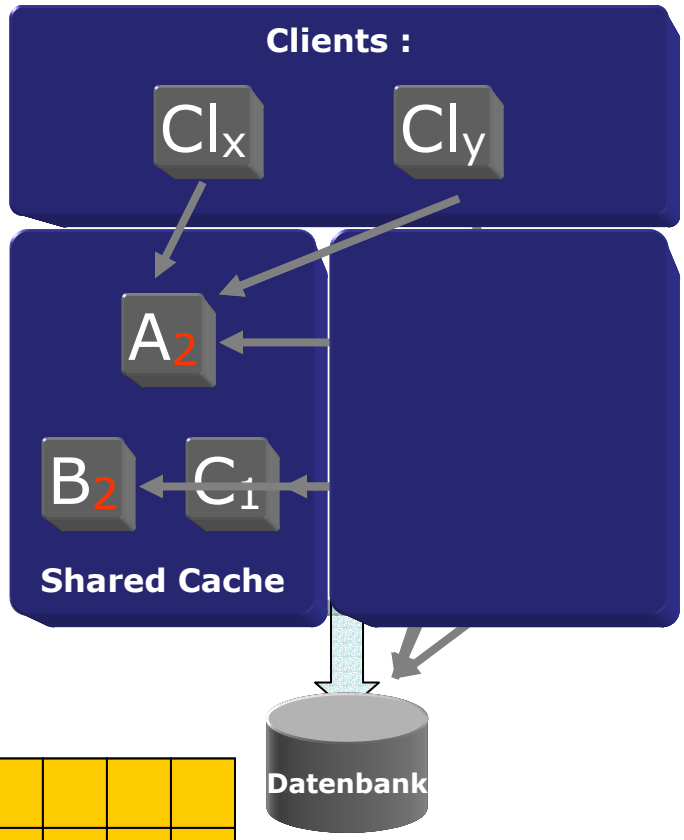
Transaktionales Caching

Optimistisches Locking

Cache-Synchronisation

Transaktionales Caching

Initialer Zustand :
- zwei Clients (Cly, Clx)
- ein Objekt (A) im Shared Cache



A	2		
B	2		
C	1		

(1) Start tx
- erzeugen tx Cache

(2) Verarbeitung tx Logik
- ändern Objekt A
- lesen Objekt B
- ändern Objekt B
- erzeugen Objekt C

(3) Commit tx
- schreiben Objekt A
- schreiben Objekt B
- einfügen Objekt C

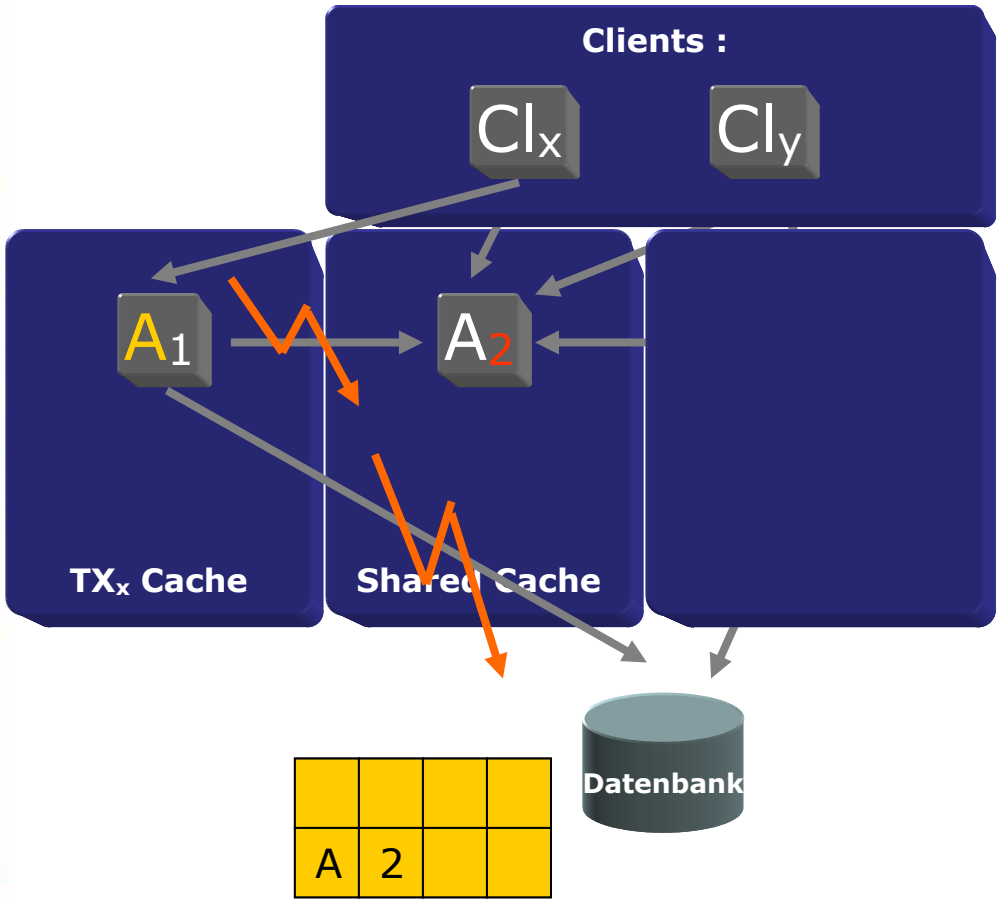
(4) sync. Shared Cache
- schreiben Objekt A
- schreiben Objekt B
- einfügen Objekt C

(5) entfernen tx Cache

Persistence

Optimistisches Locking

Initialer Zustand :
- zwei Clients (Cly, Clx)
- ein Objekt (A) im Shared Cache



(1) Start tx (y)
- erzeugen tx Cache
- ändern Objekt A

(2) Start tx (x)
- erzeugen tx Cache
- ändern Objekt A

(3) Commit tx (y)
- schreiben Objekt A

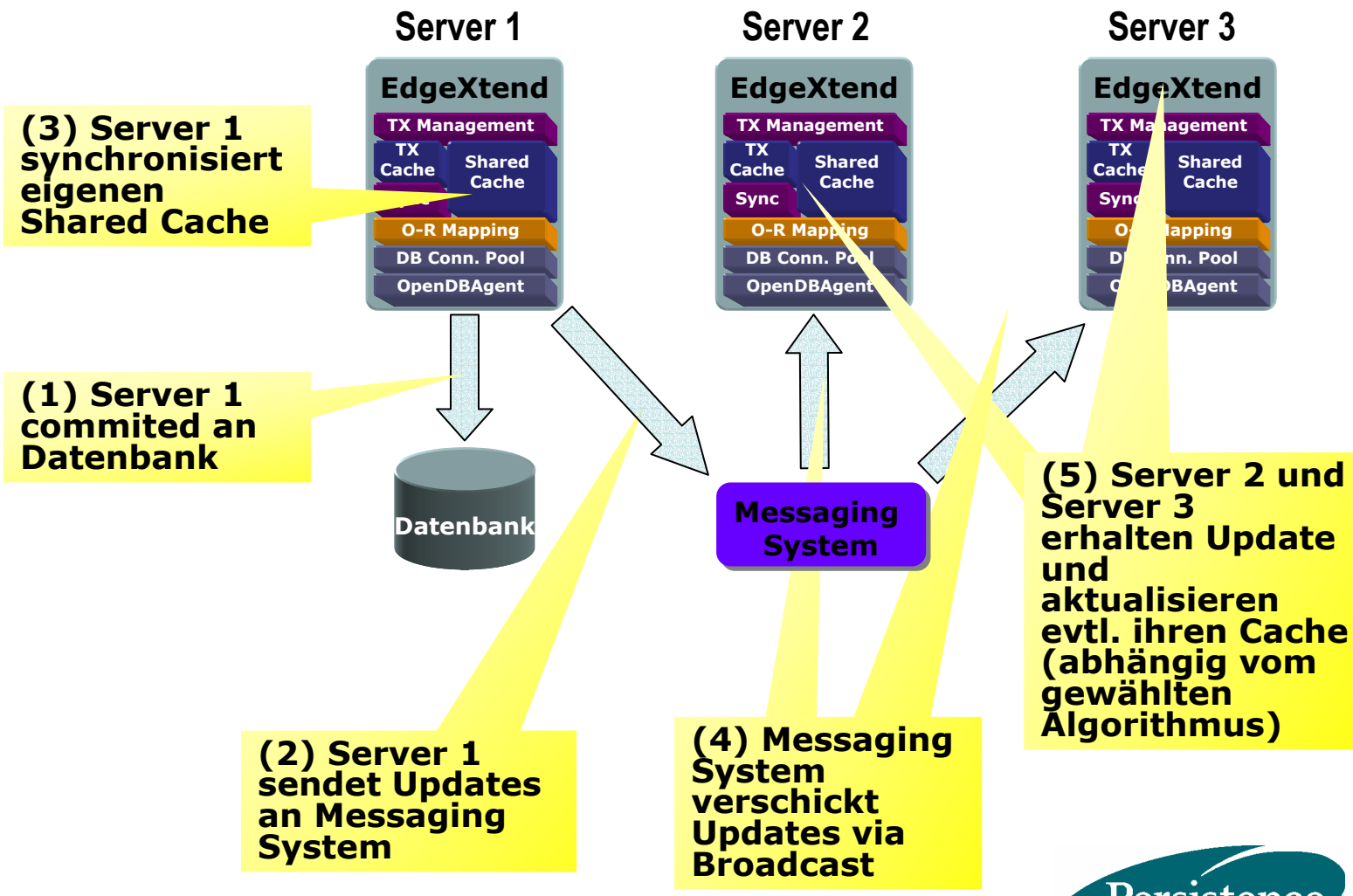
(4) sync. Shared Cache
- schreiben Objekt A

(5) entfernen tx Cache (y)

(6) Versuch Commit tx (x)
- Erhalt OC Exception

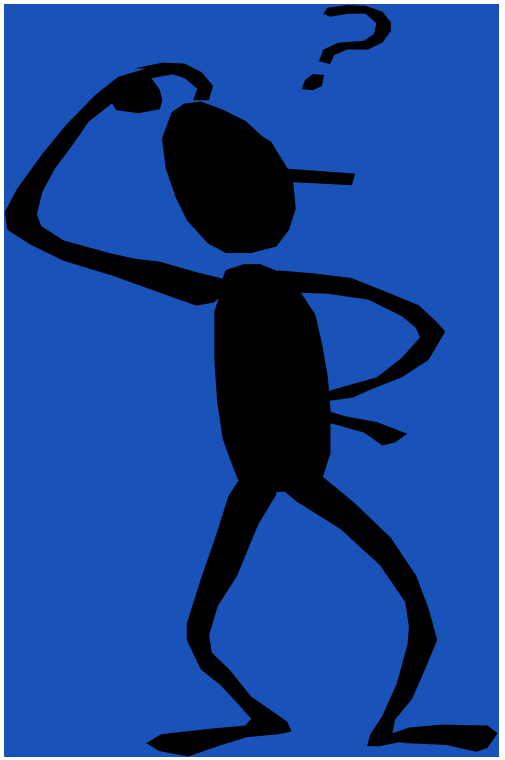
Persistence

Cache-Synchronisation



Persistence

Fragen ?



Persistence

Moving the Boundaries of Information

Data Services Architektur : O/R-Mapping

Mapping Features

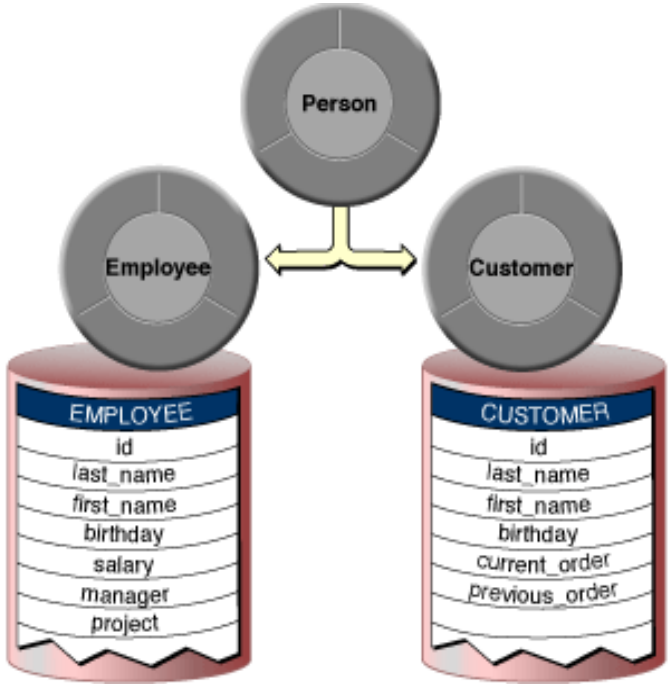
Vertikales Mapping

Wird von EdgeXtend nicht unterstützt

- Eine Tabelle je Klasse
- Die Attribute jeder Klasse sind genau in der zugeordneten Tabelle

Persistence

Horizontales Mapping



Vorteile:

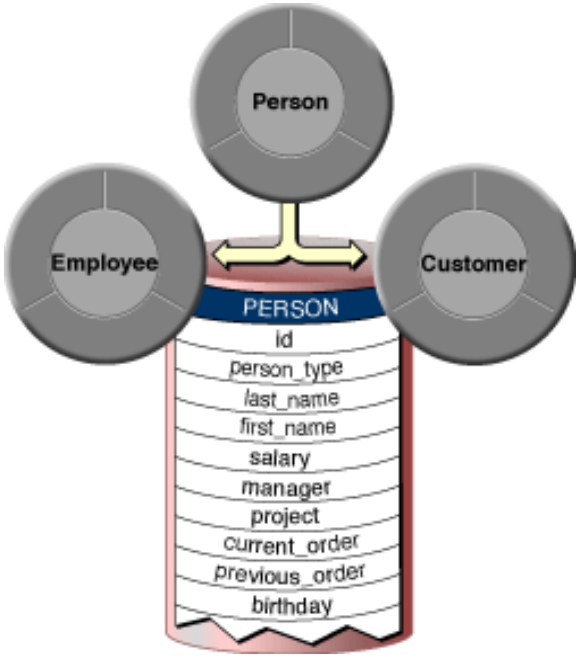
- Das System kann um eine neue Klasse erweitert werden, ohne bestehende Tabellen zu ändern
- Performant bei Suche bezüglich konkreter Klassen
- Auch bei tiefer Suche efizienter als vertikales Mapping, da keine Joins benötigt werden

Nachteile:

- Das Ändern von Oberklassenattributen führt zum Ändern aller Tabellen

- **Eine Tabelle je konkreter Unterklasse**
- **Die Attribute der (abstrakten) Oberklasse sind in jeder Tabelle aller konkreter Unterklassen**

Single-Table Mapping



Vorteile:

- Performant bei tiefer Suche (über die gesamte Hierarchie)
- Gut einsetzbar, wenn die konkreten Unterklassen wenig zusätzliche Attribute enthalten
- Änderungen des Objektmodells betreffen genau eine Tabelle

Nachteile:

- Führt zu großen, dünn besetzten Tabellen (insbesondere bei vielen stark unterschiedlichen Klassen)

- **Eine Tabelle für die gesamte Klassenhierarchie**
- **Alle Attribute aller Klassen sind in dieser Tabelle**

Persistence

Persistence

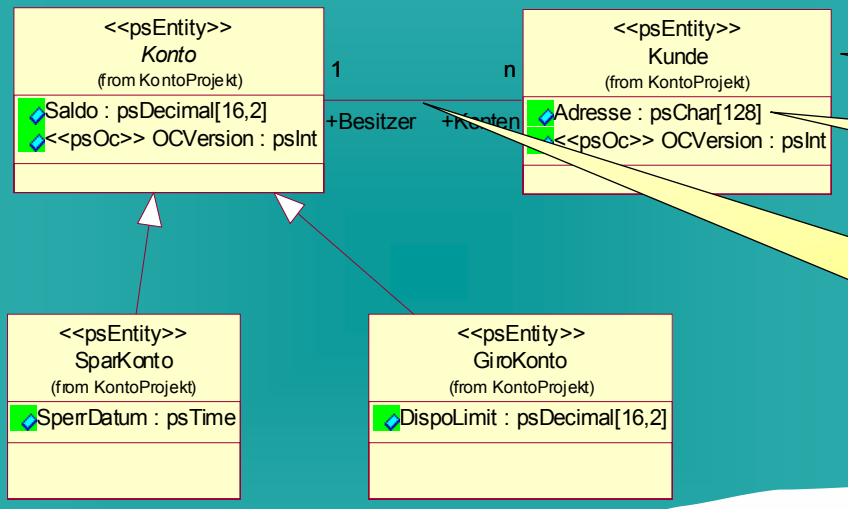
Moving the Boundaries of Information

Data Services Architektur : Entwicklungsprozess

Modellbasierte Generierung
von Code und Mapping

Entwicklungsprozess I

UML-Modell (inkl. Mappingspezifikation)



Modellierung von Klassen (inkl. Vererbung)

Spezifikation von Attributen

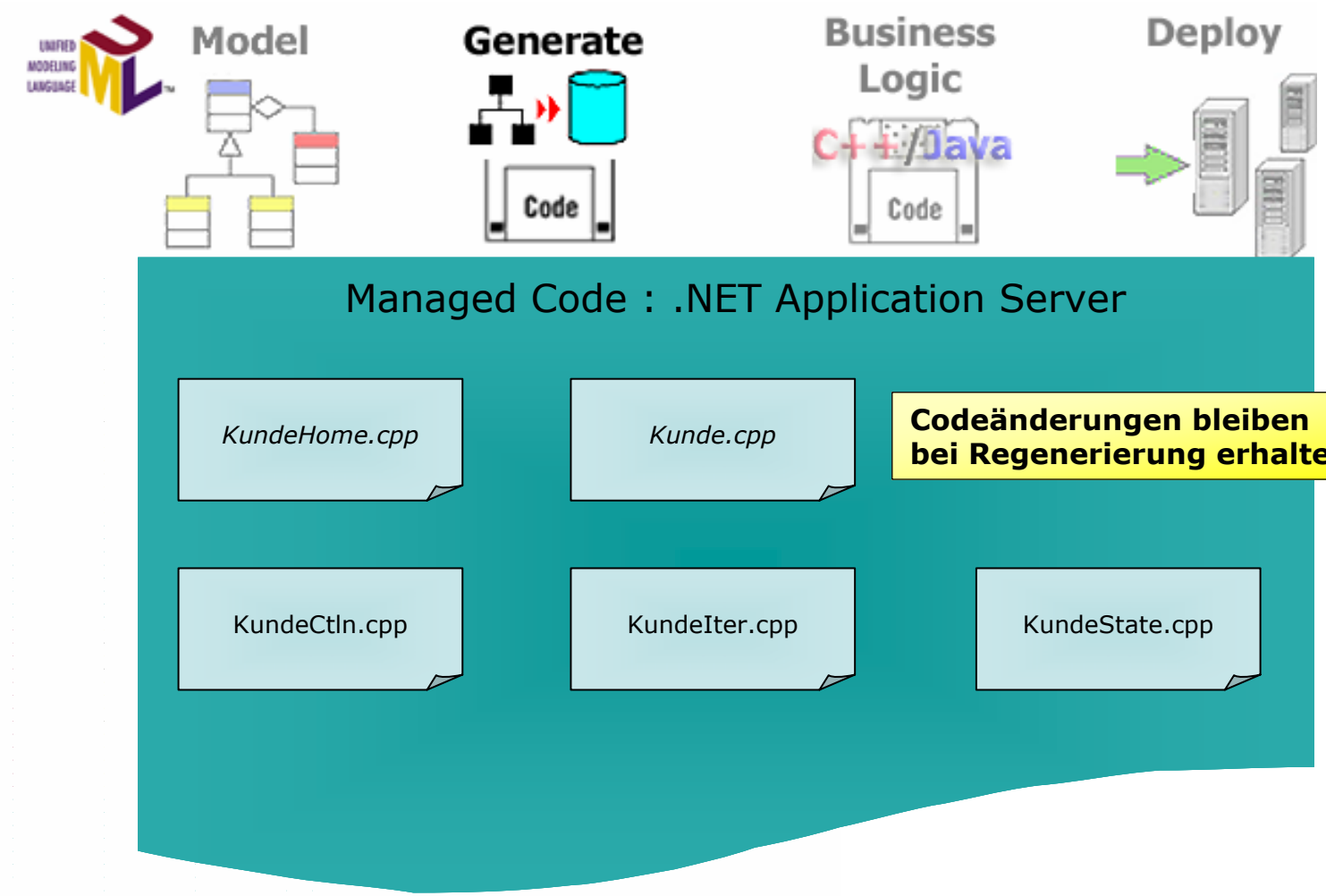
Spezifikation von Beziehungen :
- **Benennung**
- **Kardinalität**
- **Delete Actions**

Optional :
- **Generierung bzw.**
- **Import des DB-Schemas**

Modellbasierte Entwicklung

Persistence

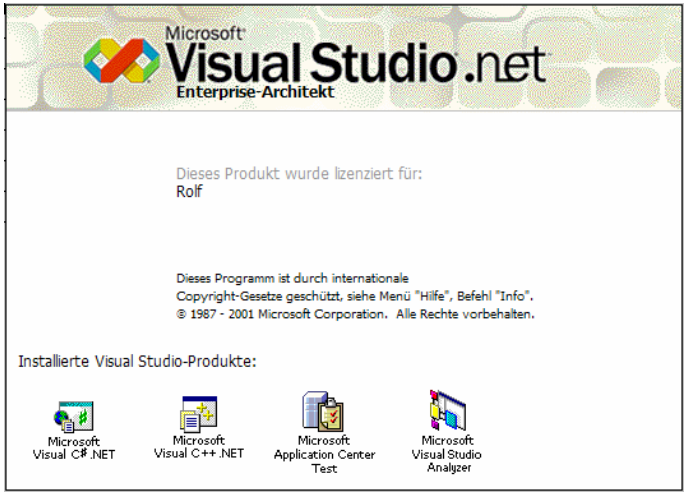
Entwicklungsprozess II



Generierung – .NET

Persistence

Entwicklungsprozess III



Entwicklung der Geschäftslogik

Persistence

Entwicklungsprozess III

Entwicklung der Geschäftslogik

- Saubere Abstraktion zwischen Geschäftslogik und Datenzugriffsschicht
- Geschützte Code-Bereiche für manuelle Erweiterungen der generierten Klassen.
- Unterstützung von inkrementeller Softwareentwicklung

Entwicklungsprozess IV

Deployment der Anwendung in den Zielcontainer bzw. in den Zielserver

- Einsatz der speziellen Tools des Container- bzw. Serverproviders

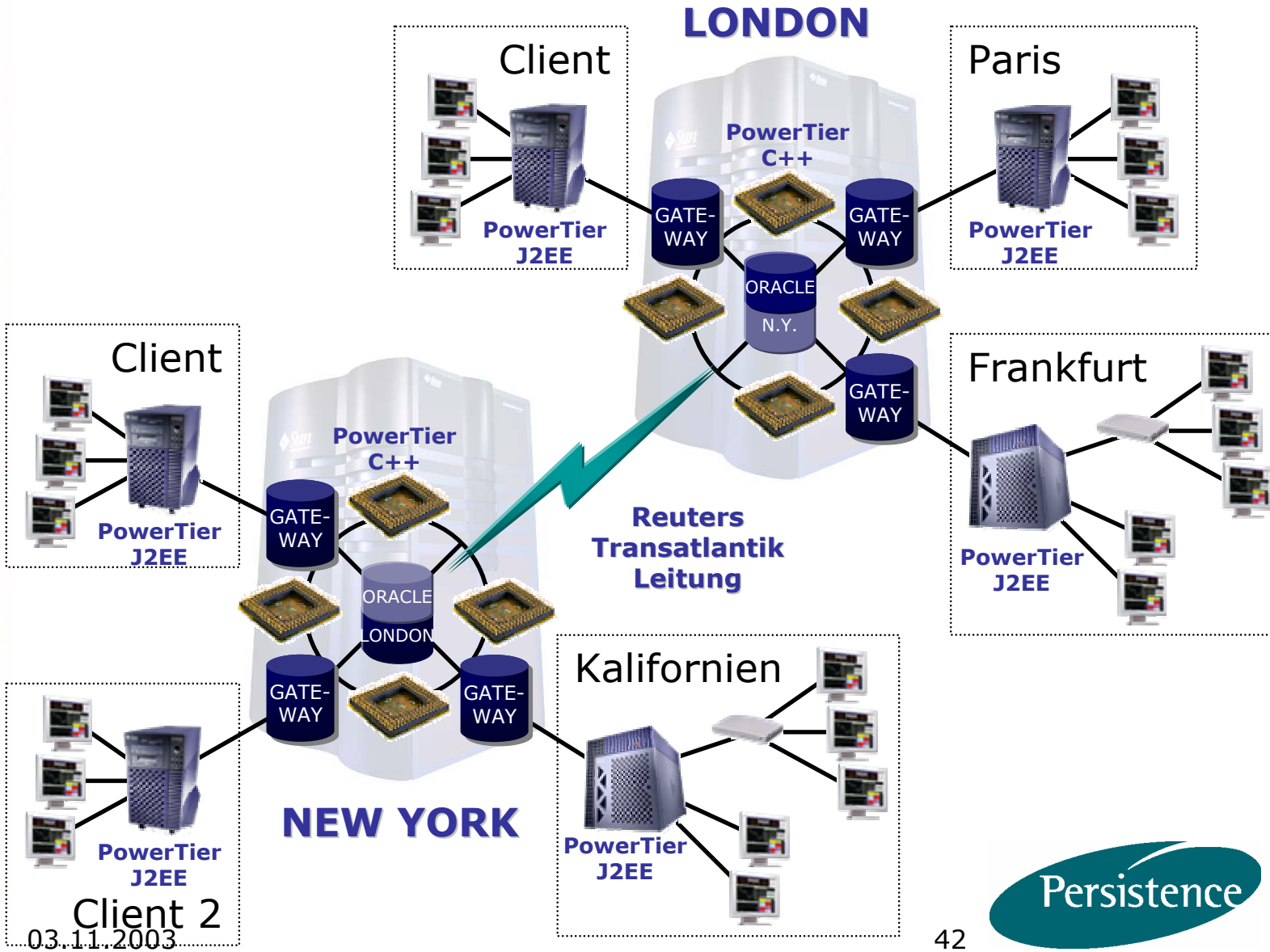
Persistence

Persistence

Moving the Boundaries of Information

Data Services Architektur : Anwendungsfälle

Anwendungsfall 1: Instinet

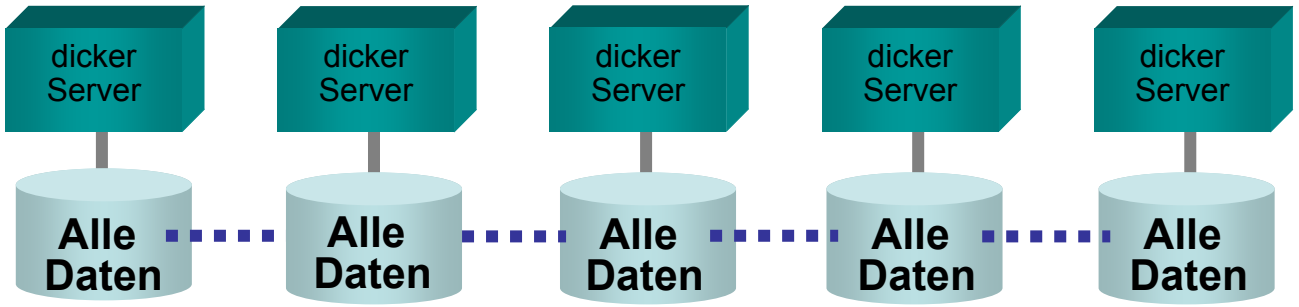


Persistence

Anwendungsfall 2: Kabel-TV

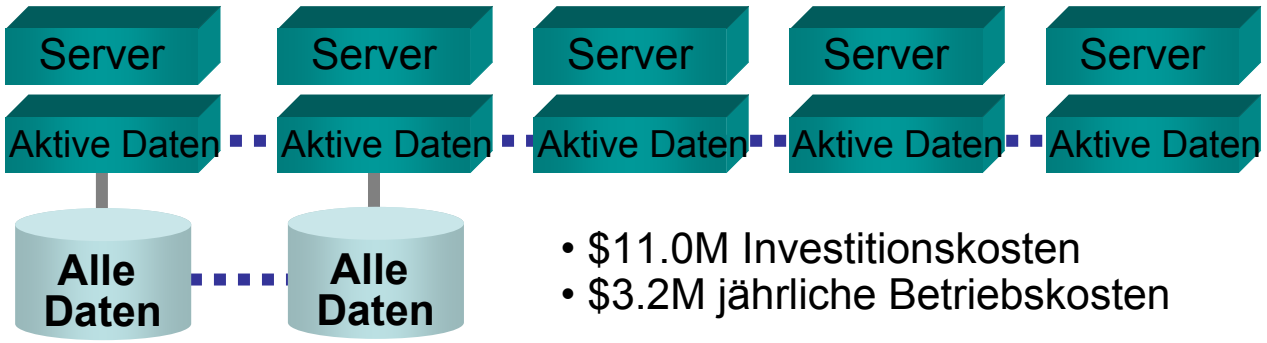
=Informationsfluss

Traditionelle Lösung: Replizierte Datenzentren



- $38.6M Investitionskosten
- $9.5M jährliche Betriebskosten

Persistence Lösung: “Schlanke” Datenzentren



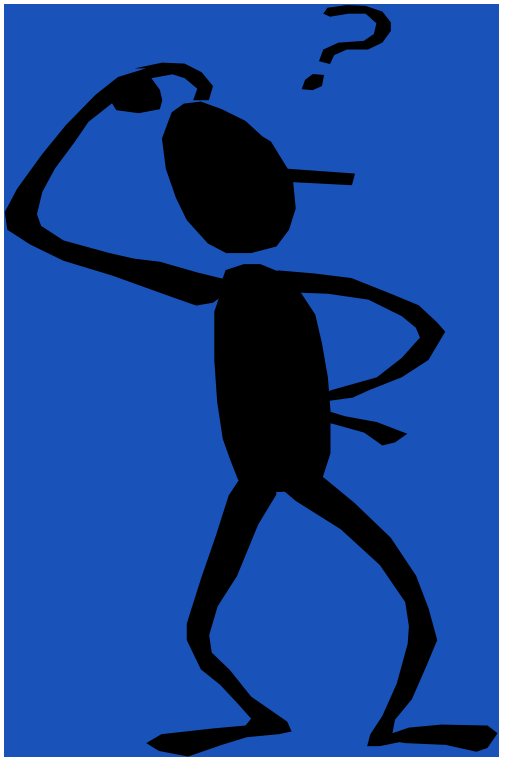
- $11.0M Investitionskosten
- $3.2M jährliche Betriebskosten

Nur die Verteilung der aktiven Daten allein brachte immense Effizienzsteigerung, >70% Kostenersparnis

Quelle: Fallstudie, Aberdeen Group, 2001

Persistence

Fragen ?



Persistence