ARS

Member of TIMETOACT GROUP SOFTWARE & CONSULTING

Enterprise Java Beans - Einführung

Java EE Grundlagen

ARS

Inhalte dieses Kapitels

Grundlagen

Architektur

EJB-Container

Arten von EJBs

Weitere Features

Verwendung von EJBs

Zusammenfassung

ARS

Grundlagen [1|2]

Enterprise Java Beans

- Standardisierte Komponenten für Geschäftslogik
- Nutzung von bereitgestellten Diensten
  - Persistenz
  - Transaktionen
  - verteilte Architekturen
  - Security
  - Timer
  - Clustering
- Verwaltung durch Container
- Bereitstellung über JNDI

ARS

Grundlagen [2|2]

- Thin-Client
  - Umfang des Clients geringer, da Anwendungslogik auf Server
  - Weniger benötigte Ressourcen
- Transparente Datenablage
  - Anwendungsschicht abstrahiert vom Zugriff auf die Daten
  - Anwendungsschicht kümmert sich um die Datenkonsistenz
- Bessere Skalierbarkeit und Ausfallsicherheit
  - Lastverteilung auf Ebene der Application Server
  - Routing von Anfragen bei Serverausfall
- Bessere Wiederverwendbarkeit und Wartung
  - Anwendungslogik/Dienst wird nur einmal programmiert
  - Änderungen nur an einer zentralen Komponente

ARS

Java Bean vs. Enterprise Java Bean

JavaBean

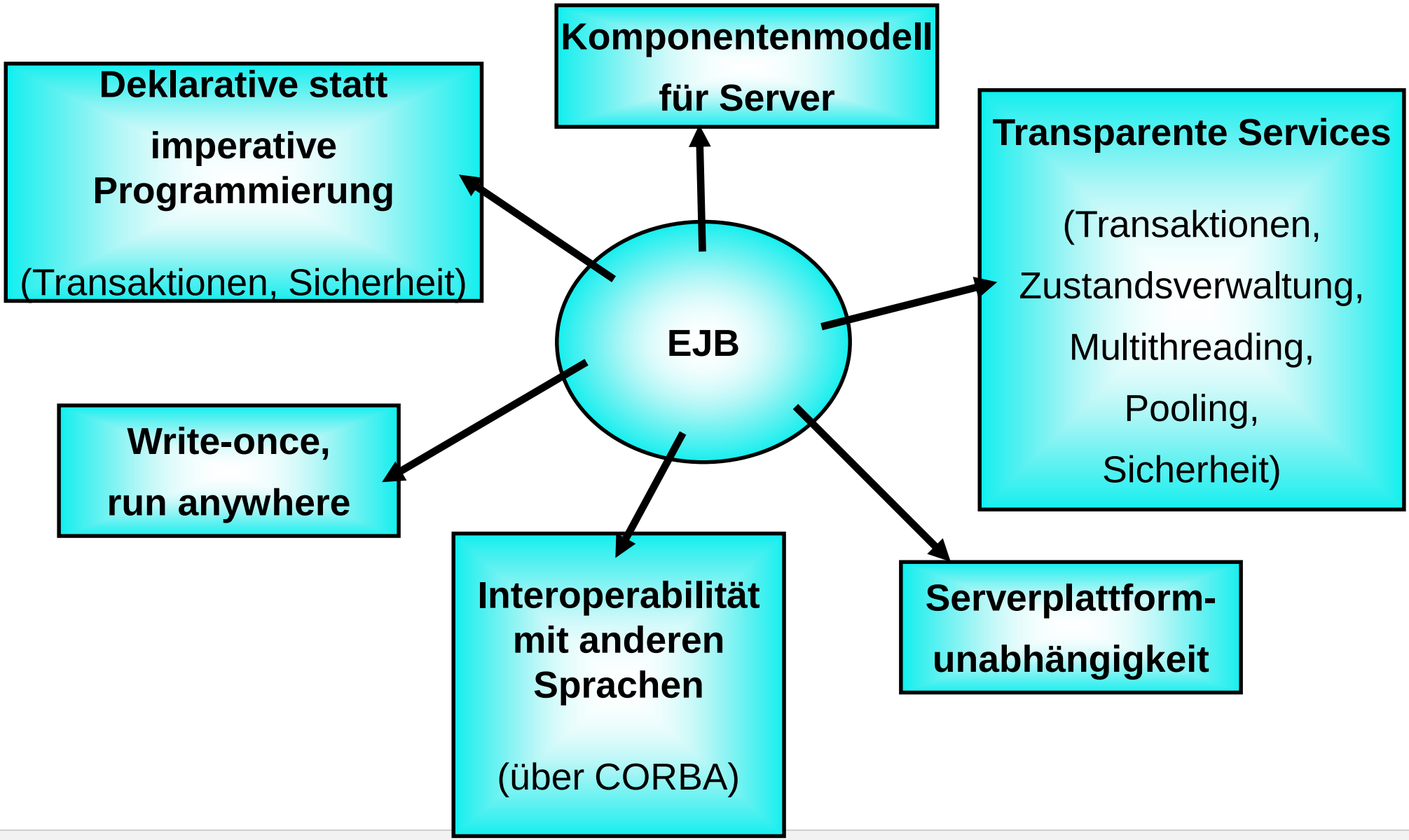
- Wiederverwendbare Komponente ohne verteilten Charakter, die mit Entwicklungswerkzeugen visuell bearbeitet werden kann
- Java-Klasse mit bestimmtem Aufbau (z.B. Getter/Setter)

Enterprise JavaBean

- Komponentenarchitektur für verteilte, serverseitige und transaktionsorientierte Komponenten
- Systemtechnisch orientiertes Komponentenmodell definiert Protokolle für die Verwaltung, Kooperation und Kommunikation zwischen Komponenten und die Nutzung durch den Client

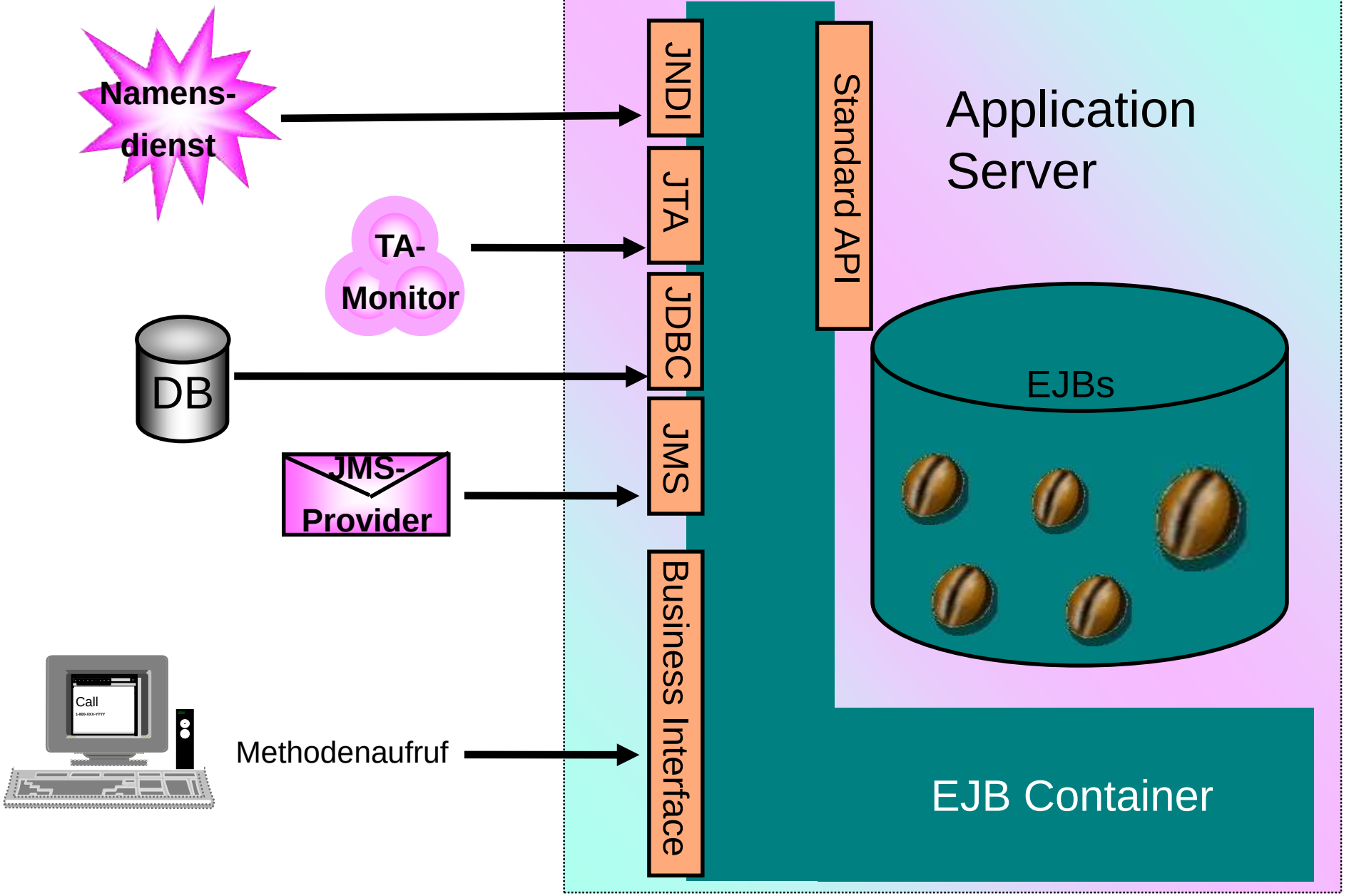
ARS

Ziele


Komponentenmodell für Server
Deklarative statt imperative Programmierung
(Transaktionen, Sicherheit)
Transparente Services
(Transaktionen, Zustandsverwaltung, Multithreading, Pooling, Sicherheit)
EJB
Write-once, run anywhere
Interoperabilität mit anderen Sprachen
(über CORBA)
Serverplattform-unabhängigkeit

ARS

Architektur [1|4]



ARS

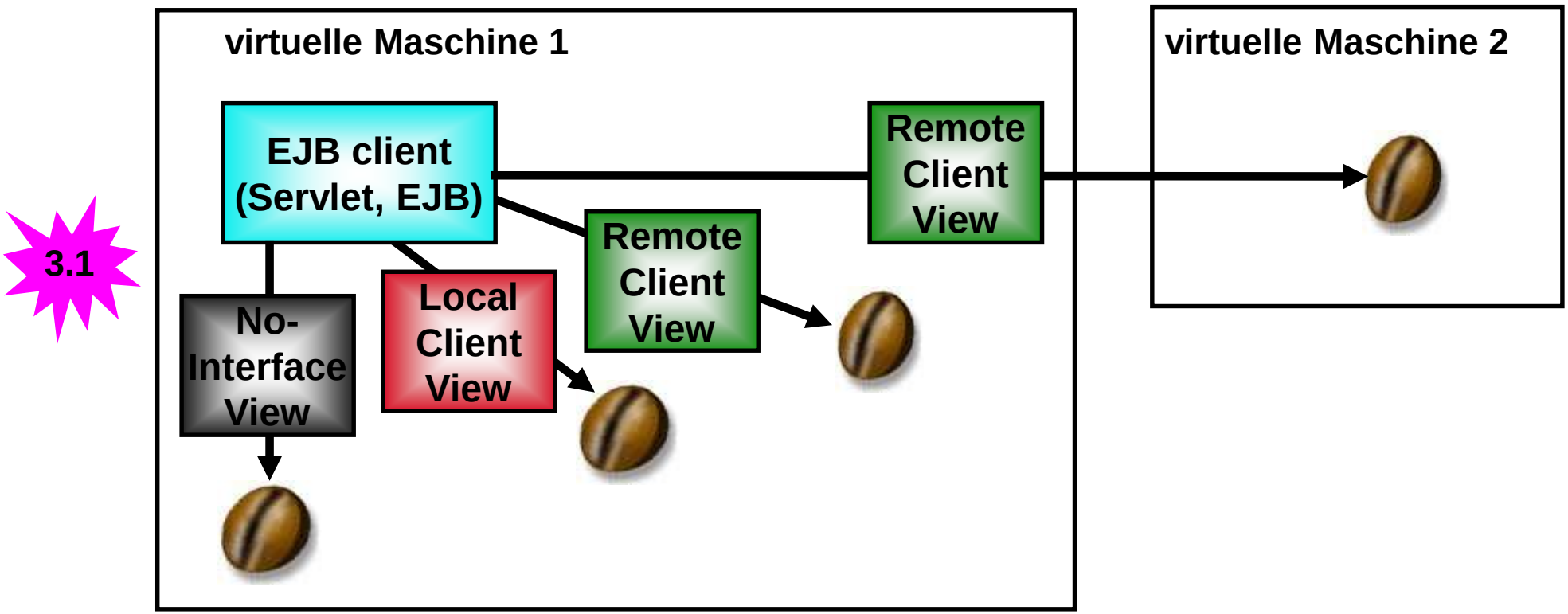
Architektur [2|4]

Sichtweisen des Clients auf eine EJB

- Local Business Interface: Für Clients, die in der gleichen virtuellen Maschine ablaufen wie die EJB selbst (`@Local`)
- Remote Business Interface: Für Clients, die in einer anderen (oder der gleichen) virtuellen Maschine ablaufen wie die EJB selbst (`@Remote`)
- No-Interface View, d.h. kein Interface nötig: EJB Klassentyp (`@LocalBean` bzw. überhaupt keine Annotation)

ARS

Architektur [3|4]



ARS

Architektur [4|4]

Aufruf über Remote Business Interface

- Entfernte Aufrufe (zeitintensiv)
- Unabhängig vom Ort der EJB
- Übergabeparameter werden mit "call by value"-Semantik übergeben
- Normales Java-Interface mit `@javax.ejb.Remote` annotiert

Aufruf über Local Business Interface

- Lokale Aufrufe (effizient)
- Aufrufer und EJB müssen in der selben virtuellen Maschine ablaufen
- Übergabeparameter werden mit "call by reference"-Semantik übergeben
- Normales Java-Interface mit `@javax.ejb.Local`

Aufruf über No-Interface View

- Gleiche Semantik und Verhalten wie bei Local Business Interface

ARS

EJB-Container [1|9]

Wichtige Aspekte der Laufzeitumgebung und Dienste

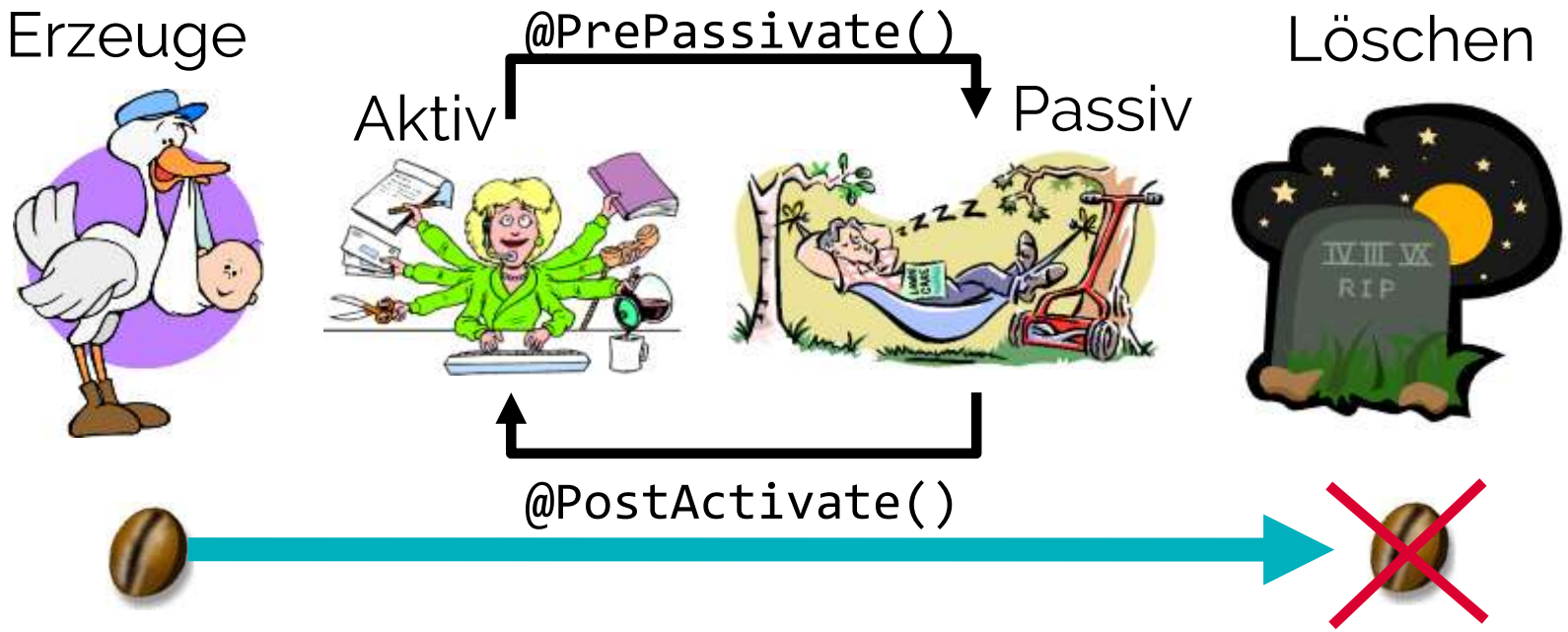
- Kontrolle des Lebenszyklus einer EJB (Laufzeitumgebung)
- Instanzen-Pooling und Passivierung bzw. Aktivierung (Laufzeitumgebung)
- Verteilung (Laufzeitumgebung)
- Namens- und Verzeichnisdienst (Dienst)
- Persistenz (Dienst)
- Transaktionen (Dienst und Laufzeitumgebung)
- Sicherheit (Laufzeitumgebung)

ARS

EJB-Container [2|9]

Kontrolle des Lebenszyklus einer Bean

- Container erzeugt Bean-Instanzen bei Aufforderung durch Client
- Container versetzt die Bean-Instanzen in verschiedene Zustände
- Container verwaltet die Bean-Instanzen in Pools, wenn sie nicht benötigt werden
- Container löscht Instanzen



ARS

EJB-Container [3|9]

Instanzen-Pooling und Passivierung bzw. Aktivierung

- Problem
  - Container muss mit großen Belastungen zurechtkommen
  - Client toleriert keine langen Antwortzeiten
  - Je mehr Clients angebunden sind, umso mehr erzeugte Objekte
- Abhilfe
  - Ständiges Erzeugen und Zerstören von Bean-Instanzen wird durch das Halten von Bean-Instanzen in einem Pool vermieden
  - Systemressourcen werden geschont, indem nicht benötigte Bean-Instanzen über Objektserialisierung persistent gemacht und aus Speicher entfernt werden
  - Bei Bedarf werden passivierte Instanzen wieder deserialisiert und im Speicher zur Verfügung gestellt

ARS

EJB-Container [4|9]

Verteilung

- Problem
  - Adressraum von Client und EJB-Container unterschiedlich
  - Ort, an dem EJB existiert, ist für den Client transparent
  - Wenn eine EJB eine andere EJB benutzt, ist sie auch ein Client
- Abhilfe
  - Verteilte Kommunikation wird über RMI abgewickelt
  - Für Kompatibilität mit CORBA wird RMI/IIOP eingesetzt (Remote Method Invocation via Internet Inter-ORB Protocol)
  - EJB wird nie direkt von außen angesprochen, sondern immer über den Container

ARS

EJB-Container [5|9]

Namens- und Verzeichnisdienst

- Client braucht zum Auffinden einer EJB einen Dienst
- Namensdienst
  - Namensdienst bildet Objektreferenzen auf Namen ab
  - Binding: Referenz wird unter einem Namen an einem definierten Platz hinterlegt
  - Lookup: Objekt wird über dessen Namen wieder gefunden
- Verzeichnisdienst
  - Verzeichnisdienst verwaltet zudem noch verteilte Objekte und andere Ressourcen (z.B. Datenquellen) in hierarchischen Strukturen
  - Verzeichnisdienst stellt zu jeder Referenz noch zusätzliche, beschreibende Informationen zur Verfügung

ARS

EJB-Container [6|9]

Persistenz

- EJB-Container
  - Stellt den EJBs über JNDI Referenzen auf Datenquellen zur Verfügung
  - Datenquellen kapseln den Zugriff auf die Datenhaltung
  - Datenquellen werden vom Application Server verwaltet
    - Verbindungsaufbau zur Datenhaltung
    - Verbindungsabbau zur Datenhaltung
    - Pooling der Verbindungen
- EJB-Spezifikation sieht JPA-Mechanismus vor
  - Zu persistenzierende Attribute werden im Deployment Descriptor oder über Annotations festgelegt
  - Quelltext zum Zugriff auf die Datenhaltung wird generiert
  - In der Regel Einschränkung auf relationale Datenbanksysteme

ARS

EJB-Container [7|9]

Transaktionen

- Transaktionen sind in verteilten Systemen unverzichtbar
- Transaktionen stellen ACID-Eigenschaft sicher
  - Atomicity: Unteilbarkeit der Aktion („Alles oder Nichts"-Prinzip)
  - Consistency: Nach Aktion ist Zustand konsistent
  - Isolation: Schutz vor gegenseitiger Beeinflussung
  - Durability: Dauerhaftigkeit der Aktion auch bei Systemausfällen
- EJB-Spezifikation unterstützt
  - Flache Transaktionen
  - Verteilte Transaktionen (2-Phasen-Commit-Protokoll)
  - Implizite Transaktionssteuerung: Deployment Descriptor oder Annotations
  - Explizite Transaktionssteuerung: Implementierung

ARS

EJB-Container [8|9]

Transaktions-einstellung	Transaktion des Clients	Transaktion der Business-Methode
Not Supported	Keine	Keine
	T1	Keine
Required	Keine	T2
	T1	T1
Supports	Keine	Keine
	T1	T1
Requires New	Keine	T2
	T1	T2
Mandatory	Keine	Fehler
	T1	T1
Never	Keine	Keine
	T1	Fehler

ARS

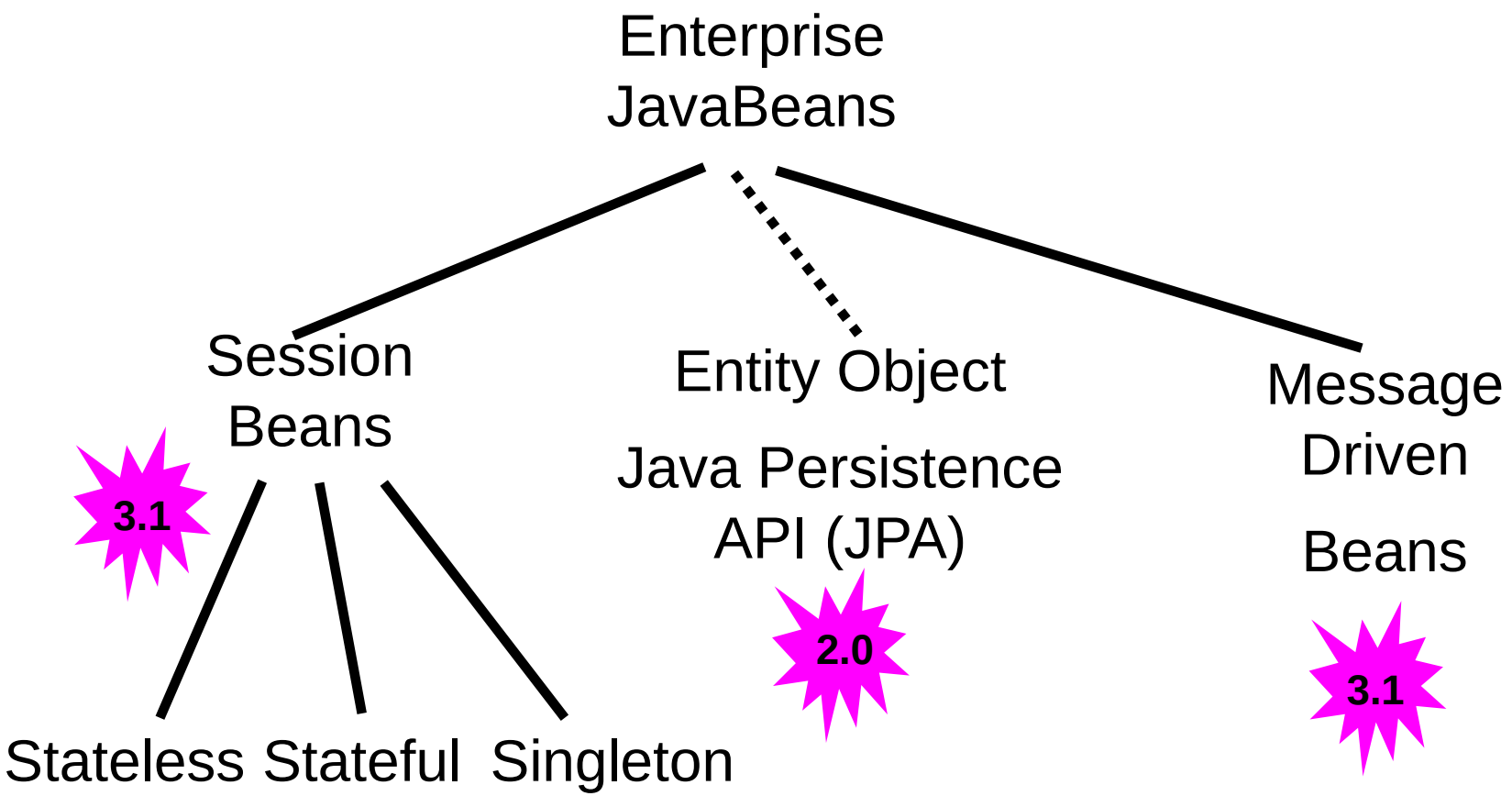
EJB-Container [9|9]

Sicherheit

- EJB-Container stellt den EJBs Infrastruktur für Sicherheits-management zur Verfügung
- Sicherheitsmechanismen werden für die EJB in der Regel transparent gehalten
  - Sicherheitsrollen
  - Berechtigungslisten für Methodenaufrufe
  - Zusätzlich Flexibilität durch Sicherheitsrollenreferenzen
- Programmatische Zugriffskontrolle über EJB-Kontext möglich
  - Wer ist Aufrufer: `ejbContext.getCallerPrincipal()`
  - Welche Rolle hat Aufrufer: `ejbContext.isCallerInRole()`
- Application Server
  - Authentifizierung durch Benutzerkennung und Passwort
  - Sichere Kommunikation (beispielsweise über SSL)

ARS

Arten von EJBs



ARS

Session Beans

- Transiente, kurzlebige Objekte
- Lebensdauer korreliert mit der Client-Sitzung
  - Stateless Session Beans
    - Haben keinen Zustand, das heißt jeder Aufruf ist wie der erste
    - Für Informationssysteme (Suchmaschine, Uhrzeitdienst)
  - Stateful Session Beans
    - Haben internen Zustand (Konversationsgedächtnis)
    - Sind einem Client zugeordnet
    - Können vom Server passiviert und wieder aktiviert werden
    - Für Geschäftsprozesse (Warenkorb)

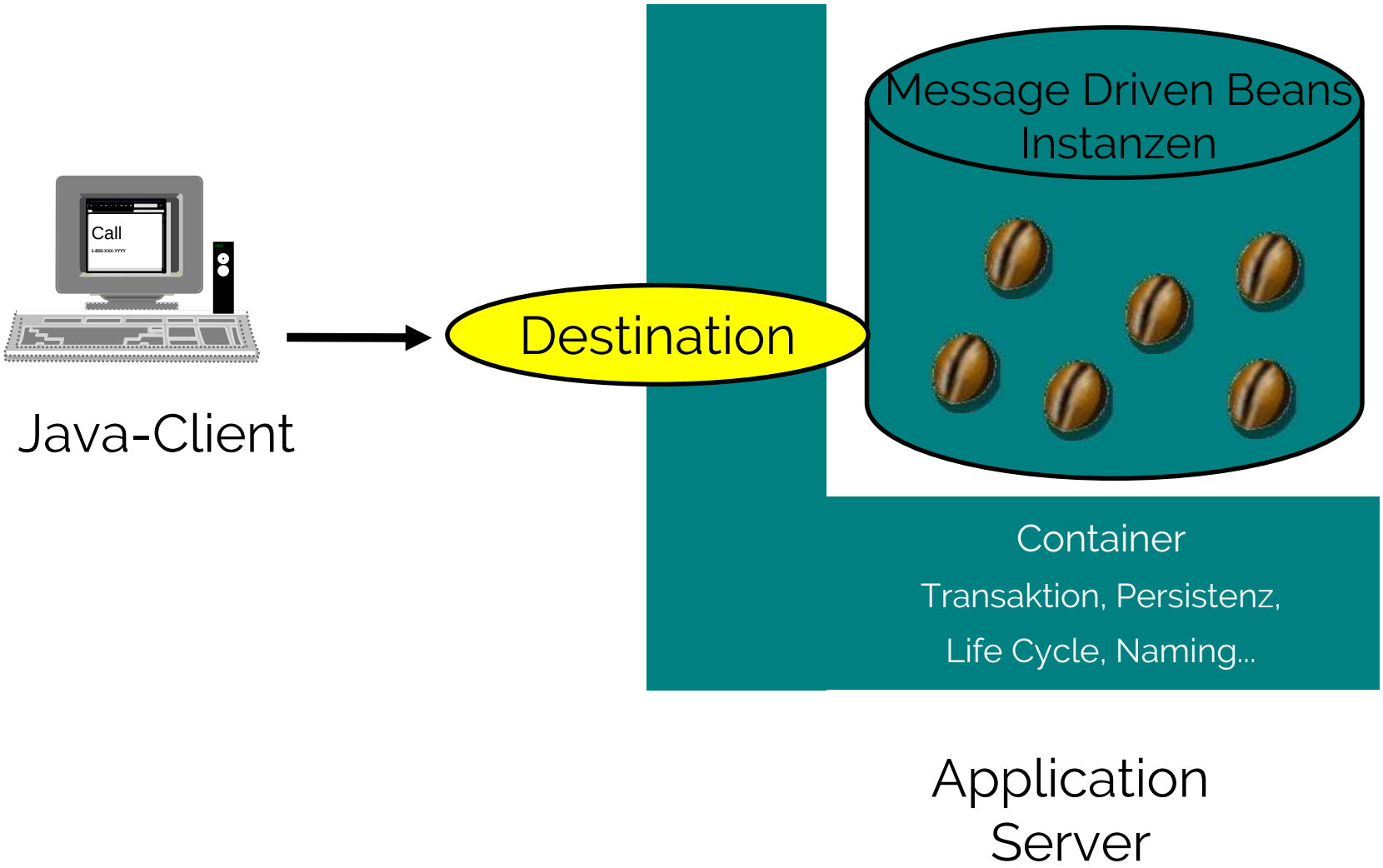
ARS

Message Driven Beans [1|2]

- Asynchron über Nachrichten ansprechbare Beans
- Integriert Java Message Service (JMS) und EJBs
- Sind aus Client-Sicht anonym
  - Keine Identität
  - Kein Konversationsgedächtnis
- Instanzvariablen zur Speicherung eines (globalen) internen Zustands möglich
  - Datenbankverbindung
  - Referenz zu einer EJB

ARS

Message Driven Beans [2|2]



ARS

Weitere Features [1|2]

Timer Service

- Möglichkeit, eine EJB zeitgesteuert zu starten/auszuführen
- Intervalle/Datum/Perioden
- Jeder Timer gehört zu einer EJB, eine EJB kann mehrere Timer haben
- Zwei Arten
  - Single-action Timer
  - Interval Timer
- Timer kann persistent oder nicht persistent sein

ARS

Weitere Features [2|2]

Web Services

- Zugriff auf Web Service aus einer EJB über Service Referenz
- EJB kann einfach zum Web Service "befördert" werden
- REST-Service (JAX-RS)
  - Annotation `@Path`
- SOAP-Service (JAX-WS)
  - Annotation `@WebService`
- Anwendbar auf:
  - Stateless Session Beans
  - Singleton Session Beans

ARS

Typisches Anwendungsszenario

- Beim Start der Anwendung wird die EJB beim Namensdienst registriert (Name der EJB ist per Annotation oder im Deployment Descriptor festgelegt)
- JavaEE-(Client-)Container lokalisiert die EJB durch eine Anfrage beim Namensdienst
- EJB-Container erzeugt EJB (sofern noch nicht geschehen/Pooling)
- Client erhält Referenz auf Proxy/View
- Client ruft Business-Methode(n) des Business-Interfaces auf
- EJB-Proxy delegiert Aufrufe an eigentliches EJB-Objekt (über Container)
- EJB-Container verwaltet EJB-Objekt selbständig (Eingriff mittels life-cycle Annotationen möglich)

ARS

Was eine EJB nicht darf

- Statische Variablen verwenden (Konstanten sind erlaubt)
- Thread-Manipulation
  - Starten und Stoppen
  - Synchronisierungsmechanismen
- Ausgaben an grafische Benutzungsoberflächen
- Auf Dateien oder Verzeichnisse mit dem Package `java.io` zugreifen
- Verwendung von Sockets
  - Auf Sockets hören
  - Verbindungen an Sockets akzeptieren
- Sicherheit
  - Laden von nativen Bibliotheken
  - Eigenen Class Loader erzeugen und verwenden
  - Durch Introspection und Reflection Sicherheitspolitiken umgehen
- JVM anhalten

ARS

Vorteile

- Entwickler muss sich damit nicht mehr um die Details der Implementierung von Diensten kümmern, zum Beispiel
  - Produktspezifische Transaktionsprogrammierung
  - Zustandsmanagement
  - Nebenläufigkeit
  - Performanzoptimierung durch Caching und Pooling
- Client-Programmierer kann EJBs wie lokale Objekte benutzen
- Deployer (Middleware-Spezialist) kann Anforderungen der Dienste der Laufzeitumgebung abstrakt definieren
  - Weniger Koordinationsaufwand
  - Leichtere Ersetzung einer Implementierung/Back-End-System
  - Leichteres Performance Tuning

ARS

Wann verwendet man EJBs? [1|2]

EJBs benötigen

- Gehobene IT-Infrastruktur
- Intensive Ausbildung und großes Know-how des Entwicklerteams

EJBs haben Einschränkungen

- Einbindung von nativem Code (z.B. C-Funktionen, Treiber)
- Keine eigene Thread-Verwaltung

EJBs sind nicht erforderlich, wenn

- Es um einfache und kleine Anwendungen geht
- Sie bereits eine funktionierende (erfolgreiche) Anwendung mit einer anderen Technologie besitzen
- Ihre Anwendung ein großes GUI-Front-End für eine Datenbank darstellt
  - Heavy on Data
  - Small on Logic

ARS

Wann verwendet man EJBs? [2|2]

EJBs bieten sich an, wenn

- Die Anwendung skalierbar sein muss, auch für (sehr) viele Anwender
- Viele Transaktionen verwaltet werden
- Die bereitgestellte Middleware (stark) genutzt wird
  - Komplexe Anwendungen können schneller erstellt werden
  - Stabilere Anwendungen
- Häufig neue Mitarbeiter eingearbeitet werden müssen (wegen Standard reduziert sich der Einarbeitungsaufwand)
- Wiederverwendbare Komponenten zum Einsatz kommen
  - Eigenentwickelte Geschäftsabläufe als EJBs
  - Zugekaufte EJB-Komponenten

ARS

Zusammenfassung

- Es gibt zwei Arten von EJBs
  - Session Beans (Stateful, Stateless und Singleton)
  - Message Driven Beans
- Ausgelagert in eigene Spec.: Java Persistence (als Ersatz für Entity-Beans)
- Das EJB-Modell hat ein Rollenmodell mit verschiedenen Rollen
- Enterprise Java Beans sind wiederverwendbare, verwaltete Komponenten
- Wichtige Dienste des Containers
  - Namens- und Verzeichnisdienst
  - Transaktion
  - Security