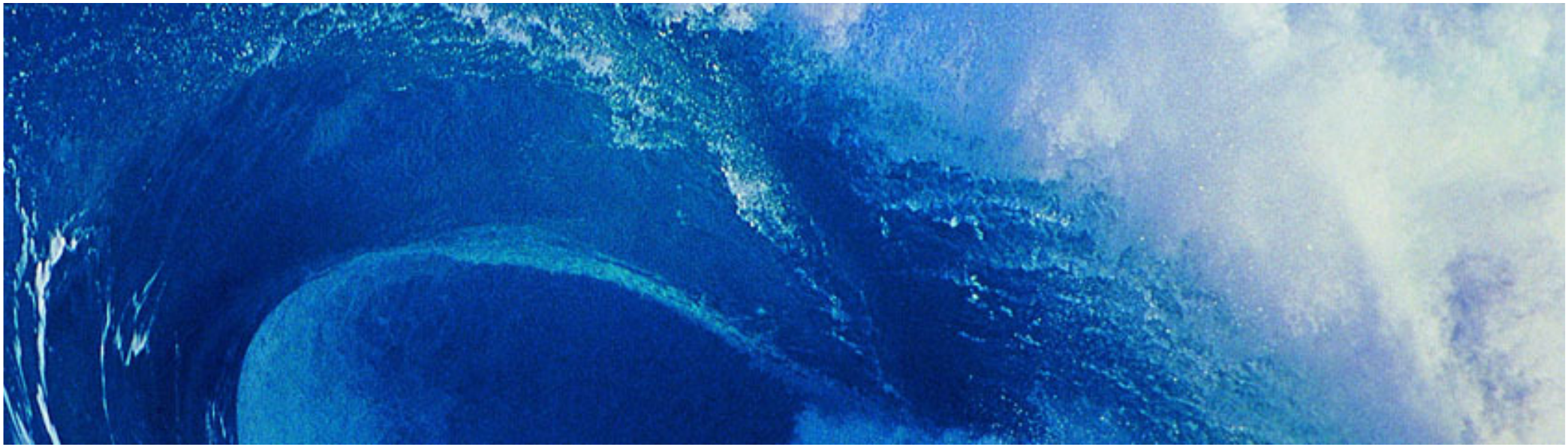
metafinanz



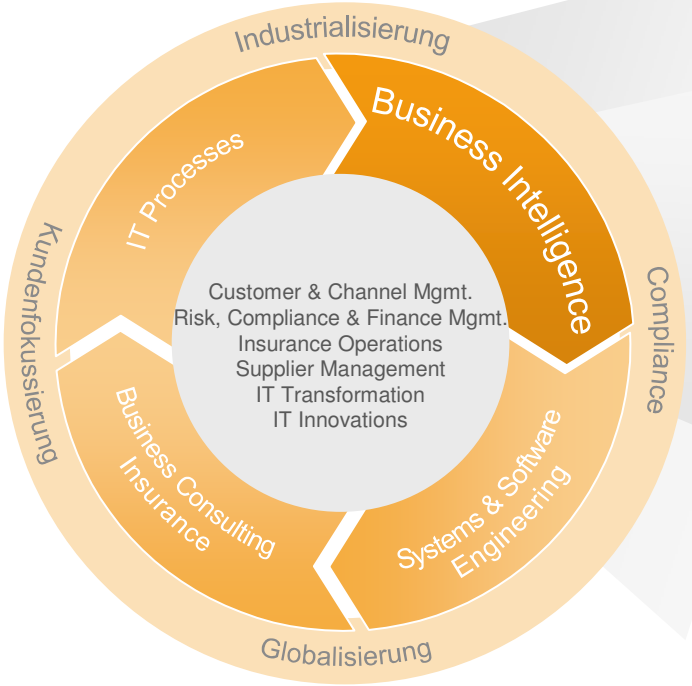
OWB Referenzarchitektur, Releasemanagement und Deployment

Carsten Herbe
metafinanz - Informationssysteme GmbH

Wir fokussieren mit unseren Services die Herausforderungen des Marktes und verbinden Mensch und IT

Facing the future

Die **Virtualisierung** und **Digitalisierung** der Welt verändert Wertschöpfungsketten und generiert neue Geschäftsmodelle. Ganzheitliche Strategien, das **Zusammenspiel zwischen Business und IT** sind dabei der Schlüssel zum wertorientierten Wachstumskurs. metafinanz gehört seit über 20 Jahren zu den erfahrensten Software-und Beratungshäusern am Markt. Wir entwickeln **intelligente zukunftsorientierte Lösungen** für komplexe Herausforderungen.



Reporting
- Customer Intelligence
- Controlling
- BI-Portale

Data Warehousing
- DWH-Architekturen & -Systeme
- Oracel Warehouse Builder
- Datenqualität

Schulungen
- OWB-Schulungen
- Oracle DWH-Schulungen

INFORMATICA

sas THE POWER TO KNOW.

COGNOS

Microsoft

ORACLE

Wir sind **Microsoft Solution Provider** sowie **Oracle-, Informatica- und SAS-Partner**.

Die metafinanz ist ein Unternehmen der Allianz Gruppe.

Beim Aufsetzen einer OWB Architektur in einer Umgebung mit mehreren Stages (Test, Integration, Produktion) ergeben sich viele Fragen zur Architektur und zum Deployment.

Fragen beim Aufsetzen einer OWB Architektur

- **Wie viele Repositories werden benötigt? Und wo?**
- **Was ist im RAC zu beachten?**
- **Wie deployt man in Integration und Produktion?**
- **Compliance-Anforderungen wie Nachvollziehbarkeit?**
- **Wie kriegt man Releases in die Versionsverwaltung?**
- **Was lässt sich automatisieren?**

Agenda

1. **Motivation**
2. **Architekturen**
3. **Configurations**
4. **Setup**
5. **Deploymentprozess**
6. **Implementierung Deploymentprozess**
7. **Fazit**

Agenda

1. **Motivation**
2. Architekturen
3. Configurations
4. Setup
5. Deploymentprozess
6. Implementierung Deploymentprozess
7. Fazit

Entwicklung, Betrieb und Governance stellen Anforderungen an Architektur und Prozesse.

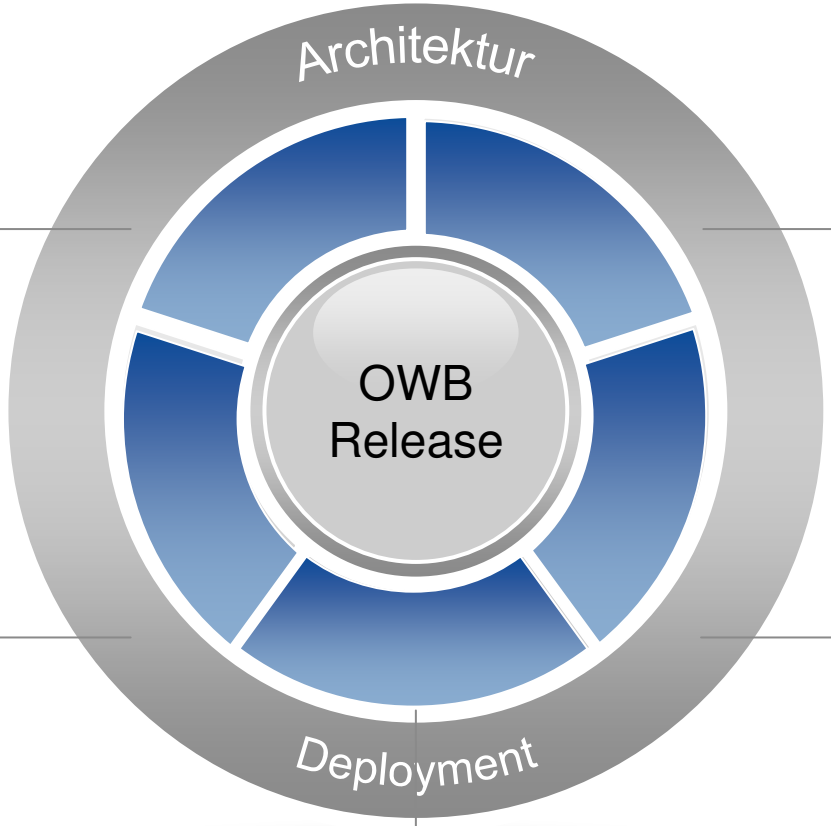
Anforderungen Architektur & Deploymentprozess

Automatisiert

Skript-basiertes Deployment vermeidet Fehler

Einfach

Wenig Aufwand für Entwicklung & Betrieb



Integriert

Nutzung bestehender Versionsverwaltungstools

Nachvollziehbar

Wann wurde was deployt?

Transparent

Auf welcher Stage ist was deployt?

Agenda

1 Motivation

2 **Architekturen**

3 Configurations

4 Setup

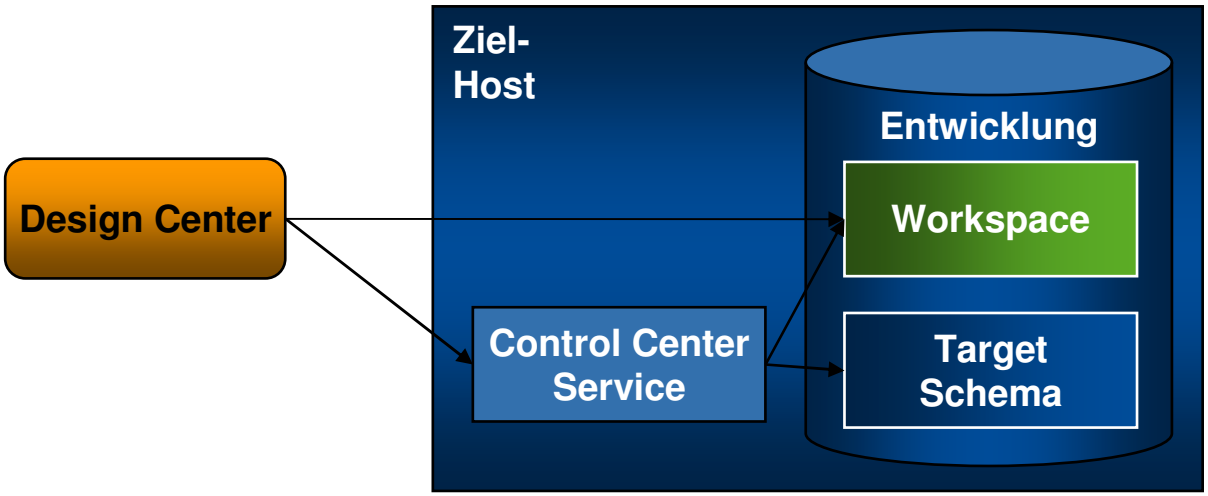
5 Deploymentprozess

6 Implementierung Deploymentprozess

7 Fazit

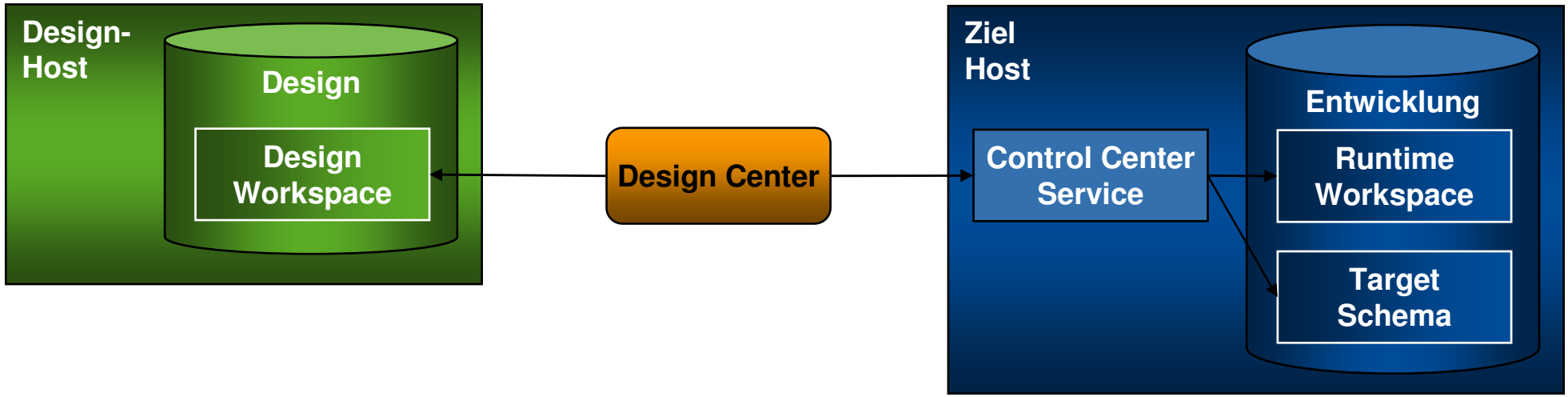
Einfache OWB Architektur, wie man sie z.B. aus Schulungen oder eigenen ersten Schritten her kennt. Es gibt nur eine Datenbank mit Design Daten und Target Schema.

First Step OWB Architektur



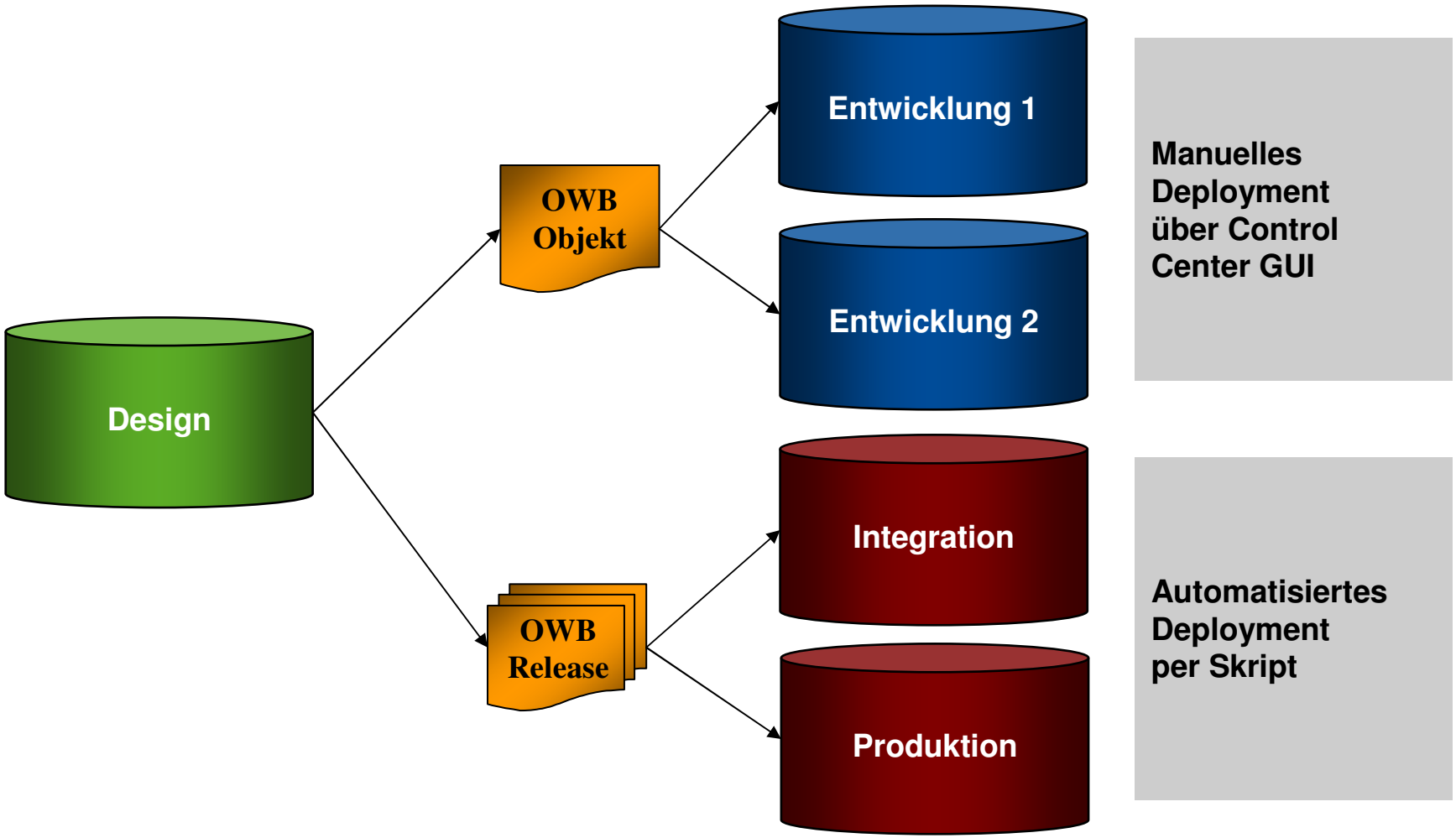
Für produktive Einsätze empfiehlt sich eine eigene Datenbank für den Workspace mit den Design Metadaten.

Einfache OWB Architektur



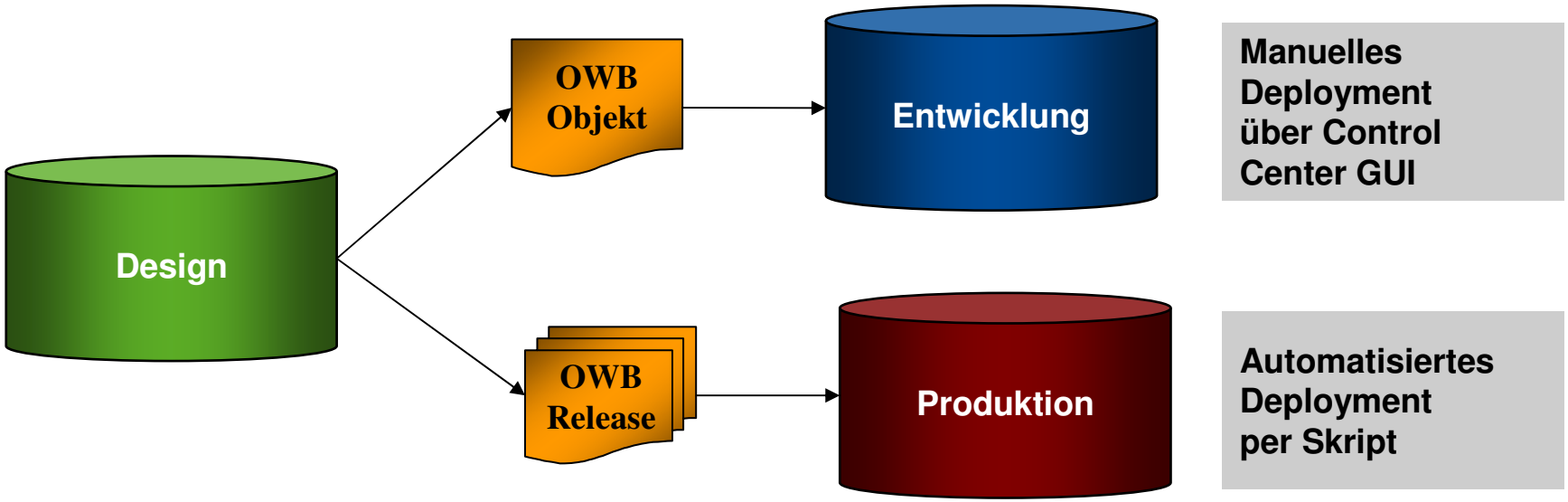
In die Entwicklung deployen die Entwickler einzelne OWB Objekte manuell. In Integration und Produktion sollen aber nur Releases geordnet und nachvollziehbar deployt werden.

Typische Stages



In unserer Beschreibung beschränken wir uns auf eine Entwicklung und die Produktion. Weitere Stages werden analog behandelt.

Typische Stages



Beim Aufsetzen einer OWB Architektur in einer Umgebung mit mehreren Stages (Test, Integration, Produktion) ergeben sich viele Fragen zur Architektur und zum Deployment.

Anforderungen an die OWB Architektur

1. **Manuelles Deployment auf Entwicklung**
2. **Automatisiertes Deployment von Releases auf Produktion**
3. **Transparenz: Was ist gerade wo deployt?**
4. **Nachvollziehbarkeit: Was wurde wann wo deployt?**

Für die Wahl der Architektur gibt es die Einfluss-Faktoren RAC, Enterprise ETL Option und Zugriff auf Produktion für Entwickler.

Einfluss-Faktoren

RAC & Enterprise ETL

- RAC ohne Enterpise ETL:
  - Deployment erlaubt
  - keine Design-Metadaten erlaubt

Zugriff auf Produktion

- In manchen Firmen ist der Zugriff auf Produktion für Entwickler nicht oder nur sehr eingeschränkt möglich

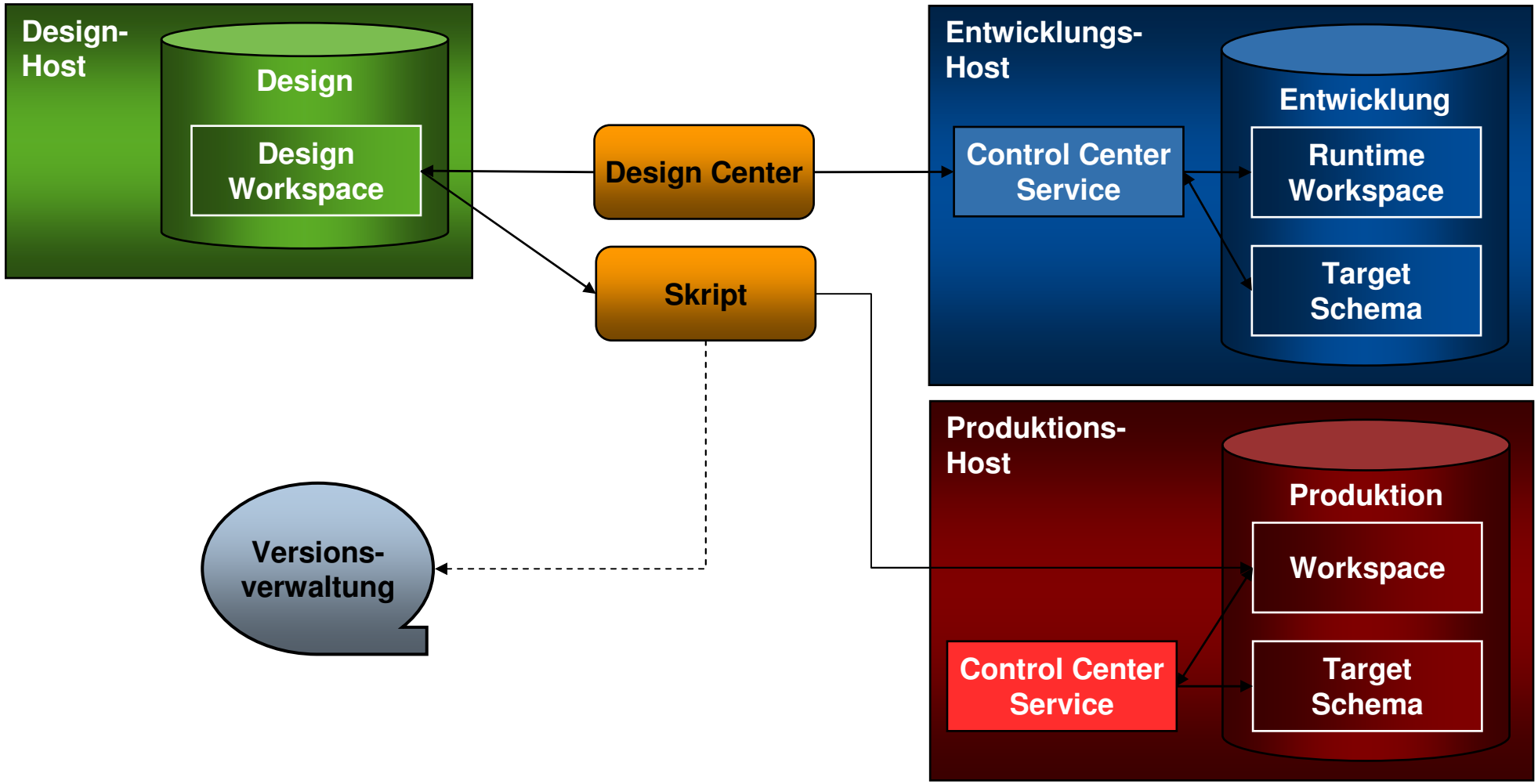
In Abhängigkeit von RAC oder Single Instance, Enterprise ETL oder nicht und Zugriff für Entwickler auf Produktion oder nicht kann man die Architekturvariante auswählen.

Entscheidungsmatrix Architekturvarianten

	Kein RAC oder RAC mit Enterprise ETL	RAC ohne Enterprise ETL
Zugriff Produktion	„Single Instance“ Architektur	„RAC“ Architektur
Kein Zugriff Produktion	„RAC“ Architektur	„RAC“ Architektur

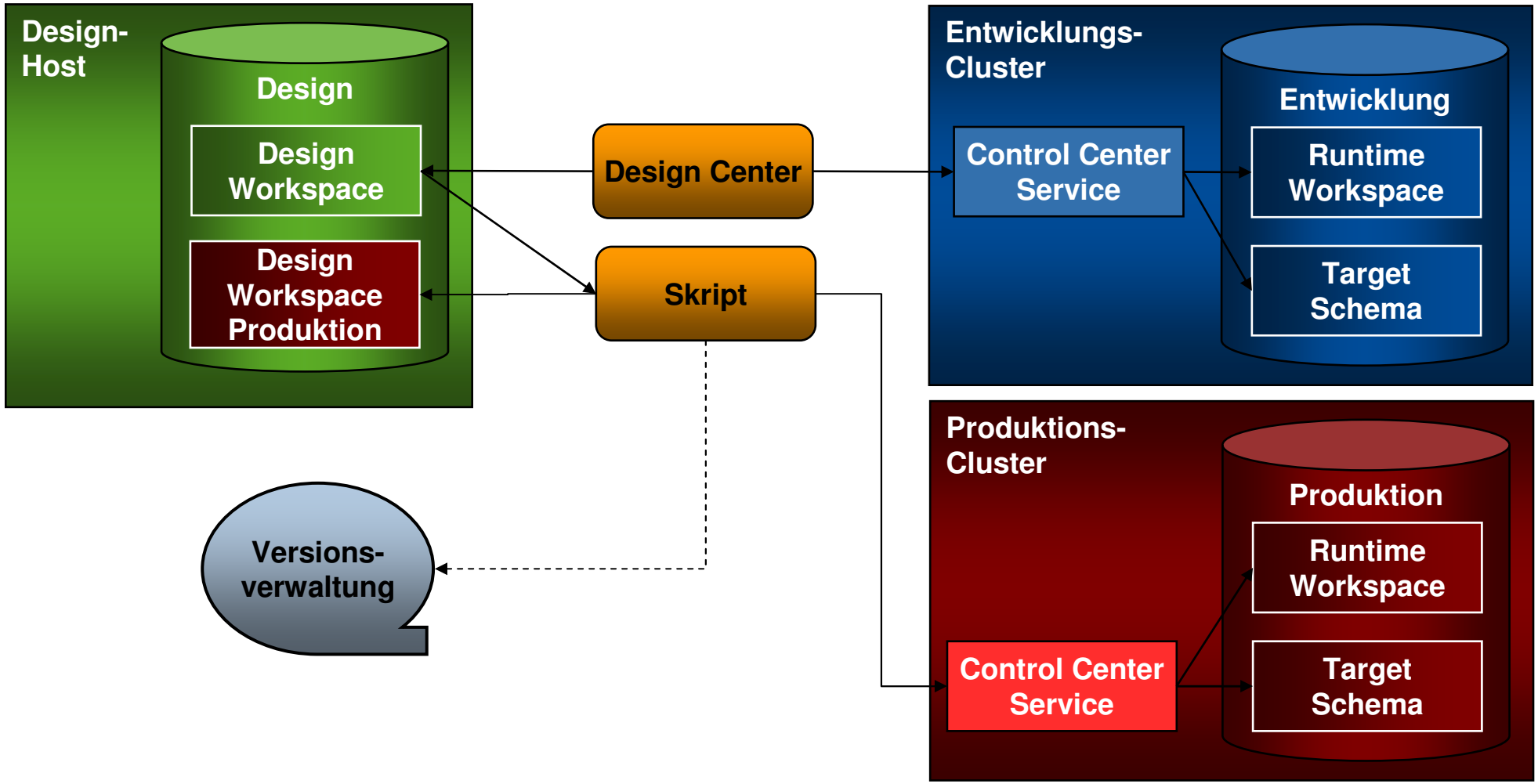
Diese Architektur bietet sich an für Single Instances oder RACs mit Enterprise ETL Option. Die Entwickler benötigen hier Zugriff auf den Produktions-Workspace.

OWB Architektur Single Instance



Diese Architektur bietet sich an für RACs ohne Enterprise ETL Option oder Umgebungen in denen die Entwickler keinen Zugriff auf das Produktions-Workspace haben können.

OWB Architektur RAC



Agenda

1 Motivation

2 Architekturen

3 **Configurations**

4 Setup

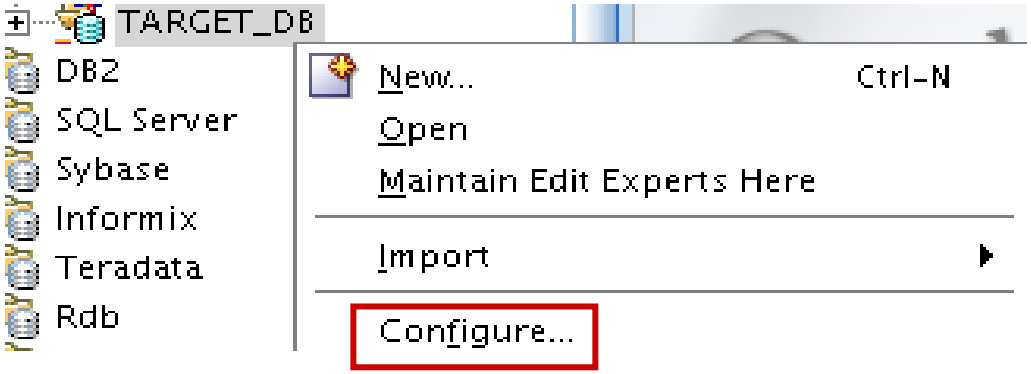
5 Deploymentprozess

6 Implementierung Deploymentprozess

7 Fazit

Was ist eine Configuration?

- Configuration = Satz von Eigenschaften aller Objekte
- Jedes Objekt hat mehrere Eigenschaften (Properties), welche über das Kontextmenü "Configure" zugänglich sind



  - Tabellen: Tablespaces, Parallelitätsgrad, …
  - Mappings: Generation Mode, Parallelität ein/aus, …
  - Datenbankmodule: Locations
- Diese Properties werden nur für die aktuelle Configuration bearbeit.
- Wechselt die Configuration, dann auch die Properties!

Wichtige Mapping Properties einer Configuration sind Enable Parallel DML, Generation Mode, Default Operating Mode sowie sämtliche Hints in den Table Operatoren.

Configuration: Mapping Properties

Start Page | Configuration of DIM_CHANNELS | **Configuration of MAP_DIM_CUSTOMERS**

Property	DEFAULT_CONFIGURATION	PRODUCTION
MAP_DIM_CUSTOMERS		
Deployable	true	true
Generation Comments		
Language	**PL/SQL**	**PL/SQL**
Referred Calendar		
Code generation options		
Analyze table statements	false	false
ANSI SQL Syntax	true	true
AUTHID option	None	None
Bulk processing code	true	true
Commit Control	Automatic	Automatic
Enable Parallel DML	false	false
Error Trigger		
Generation Mode	**Set based**	All Operating Modes
Optimized code	true	true
Use Target Load Ordering	true	true
Filter Operators		
Runtime parameters		
Sequence Operators		
Set Operators		
Splitter Operators		
Table Operators		

Die wohl wichtigsten Properties überhaupt sind die Locations der (Oracle) Module.

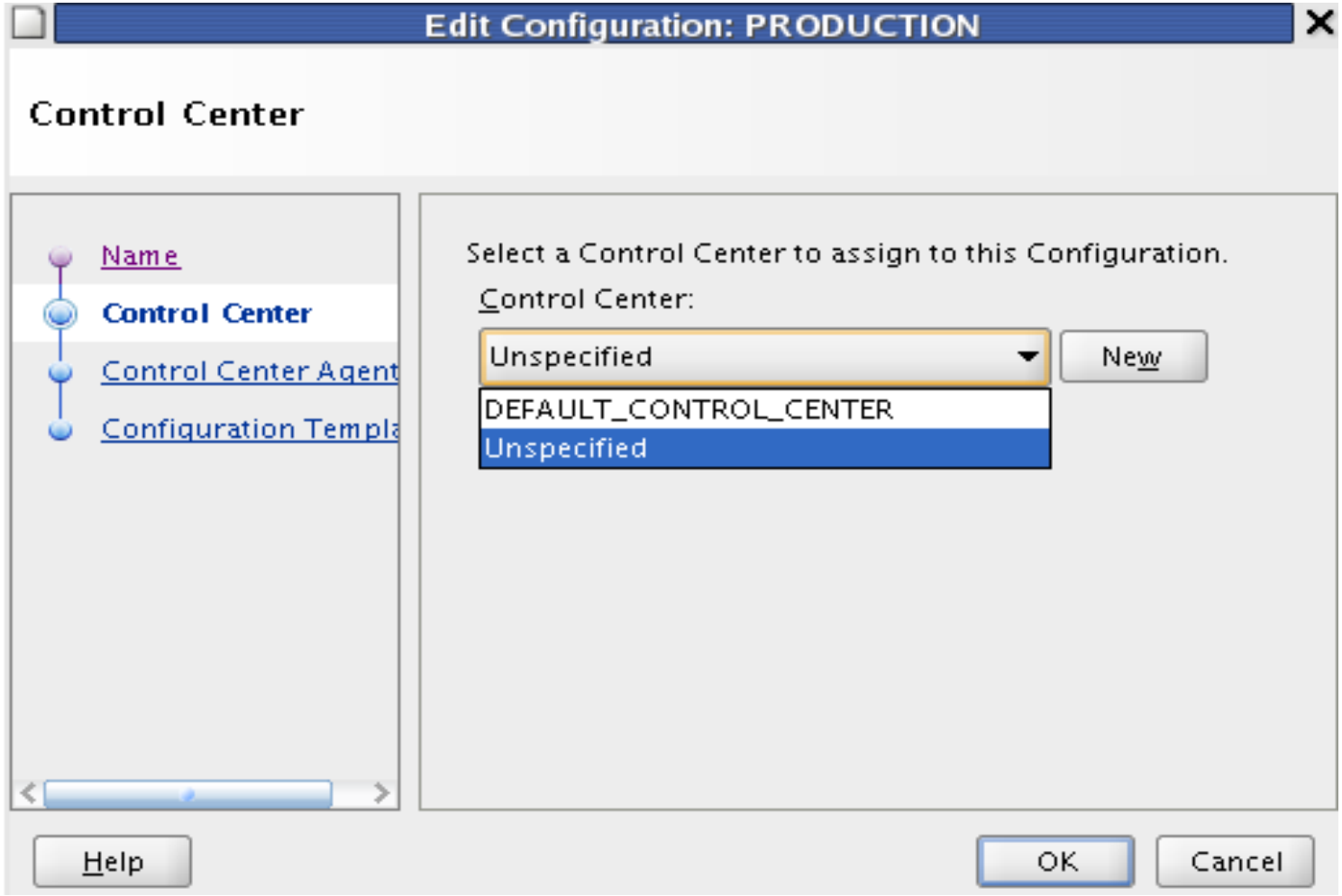
Configuration: Oracle Module Properties

Start Page | Configuration of TARGET_DB

Property	DEFAULT_CONFIGURATION	PRODUCTION
TARGET_DB		
Deployment System Type		
PL/SQL Generation Mode	Default	Default
Generation Preferences		
End of Line	\r\n	\r\n
Generation Target Directories		
Identification		
Application Short Name	WB	WB
Deployable	true	true
Location	**MFA_1_LOCATION(Default)**	MFA_1_LOCATION(Default)
Main Application Short Name	ora	ora
Streams Administrator Location	**MFA_1_LOCATION(Default)**	**MFA_1_LOCATION(Default)**
Top Directory	..\..\codegen\	..\..\codegen\
Run Time Directories		
Archive Directory	archive\	archive\
Input Directory	input\	input\
Invalid Directory	invalid\	invalid\
Log Directory	log\	log\
Receive Directory	receive\	receive\
Sort Directory	sort\	sort\
Work Directory	work\	work\
Tablespace Defaults		
Default Index Tablespace		
Default Object Tablespace		

Jeder Configuration ist ein Control Center und ein Control Center Agent zugeordnet.

Configuration: Control Center & Control Center Agent



Agenda

1 Motivation

2 Architekturen

3 Configurations

4 **Setup**

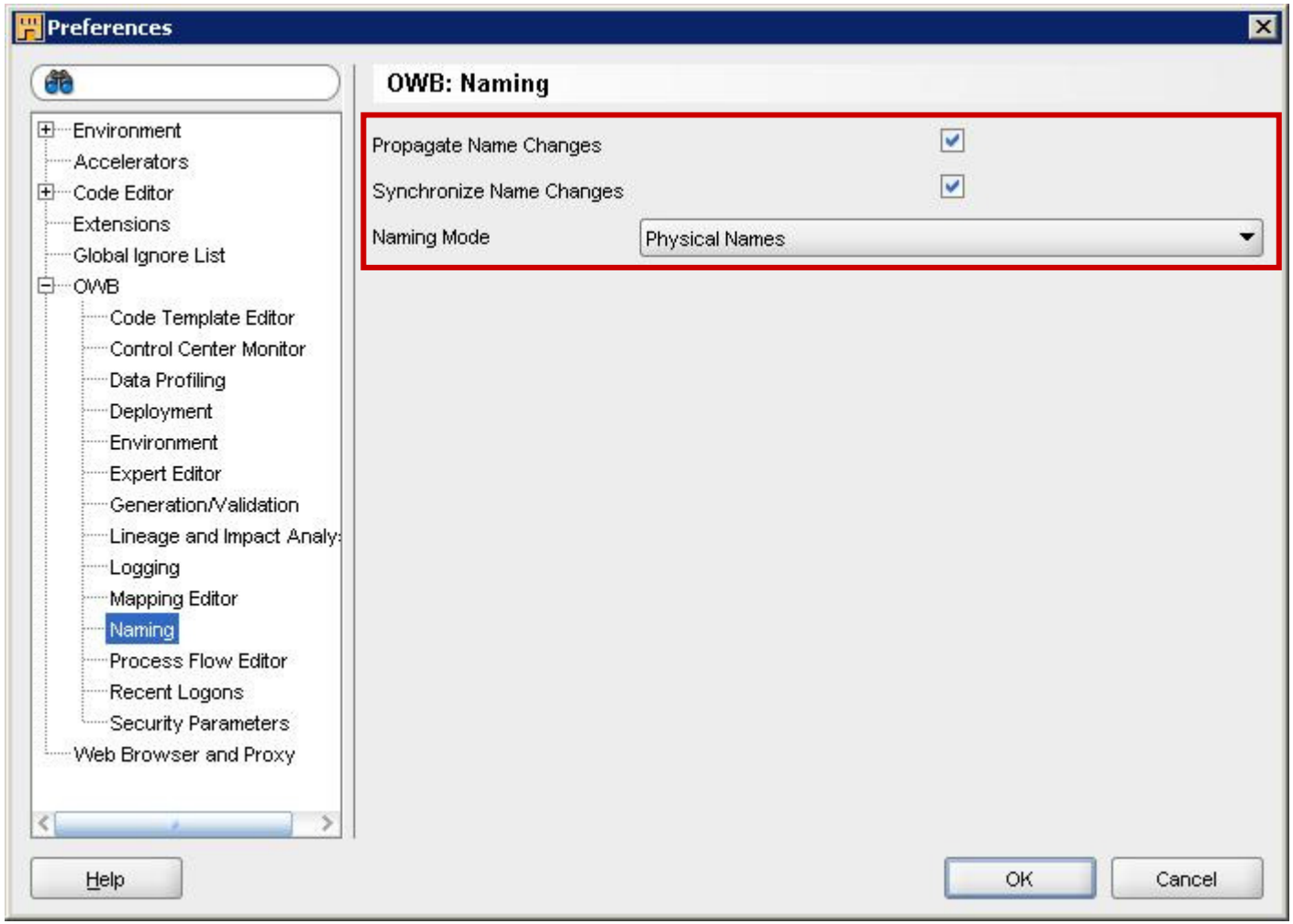
5 Deploymentprozess

6 Implementierung Deploymentprozess

7 Fazit

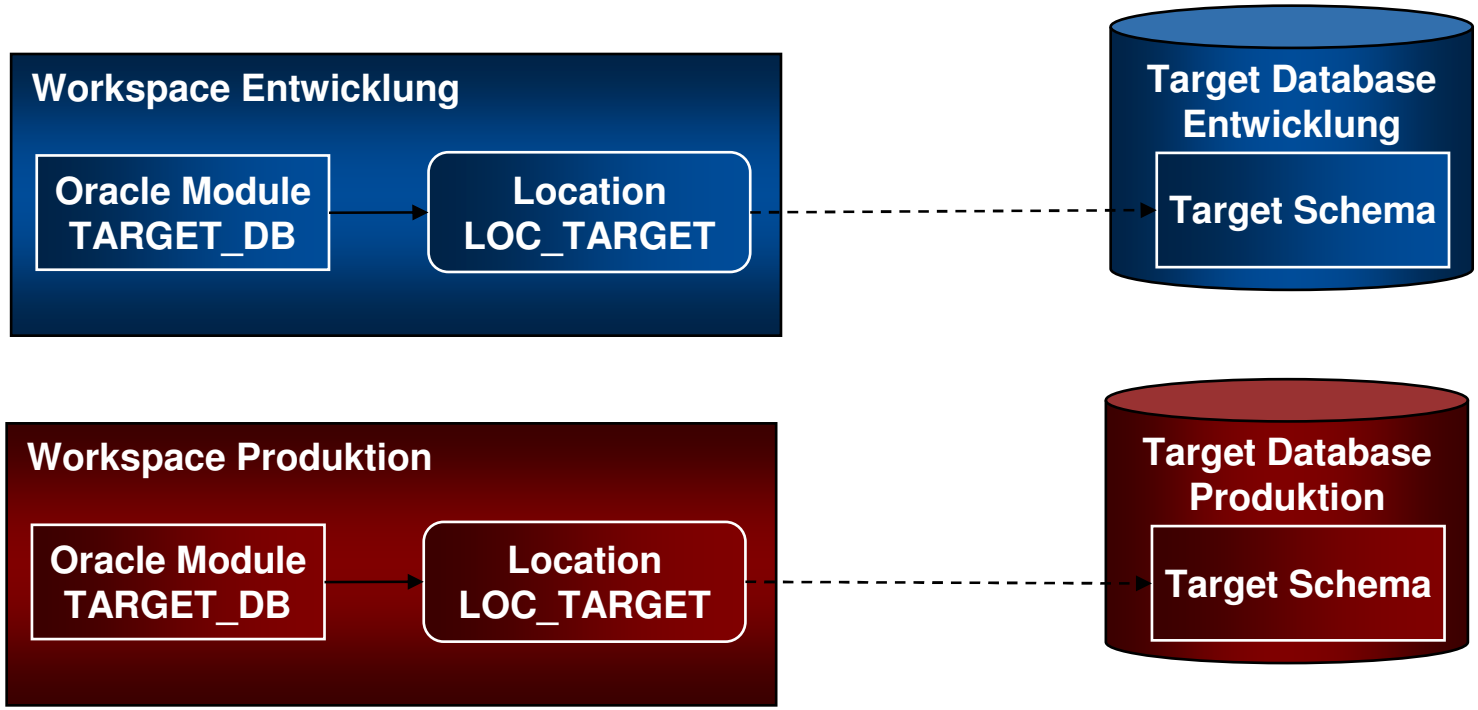
Für den Naming Mode muss Physical Names verwendet werden (inklusive Propagate & Synchronize Name Changes), damit der Import problemlos funktioniert.

Naming Mode



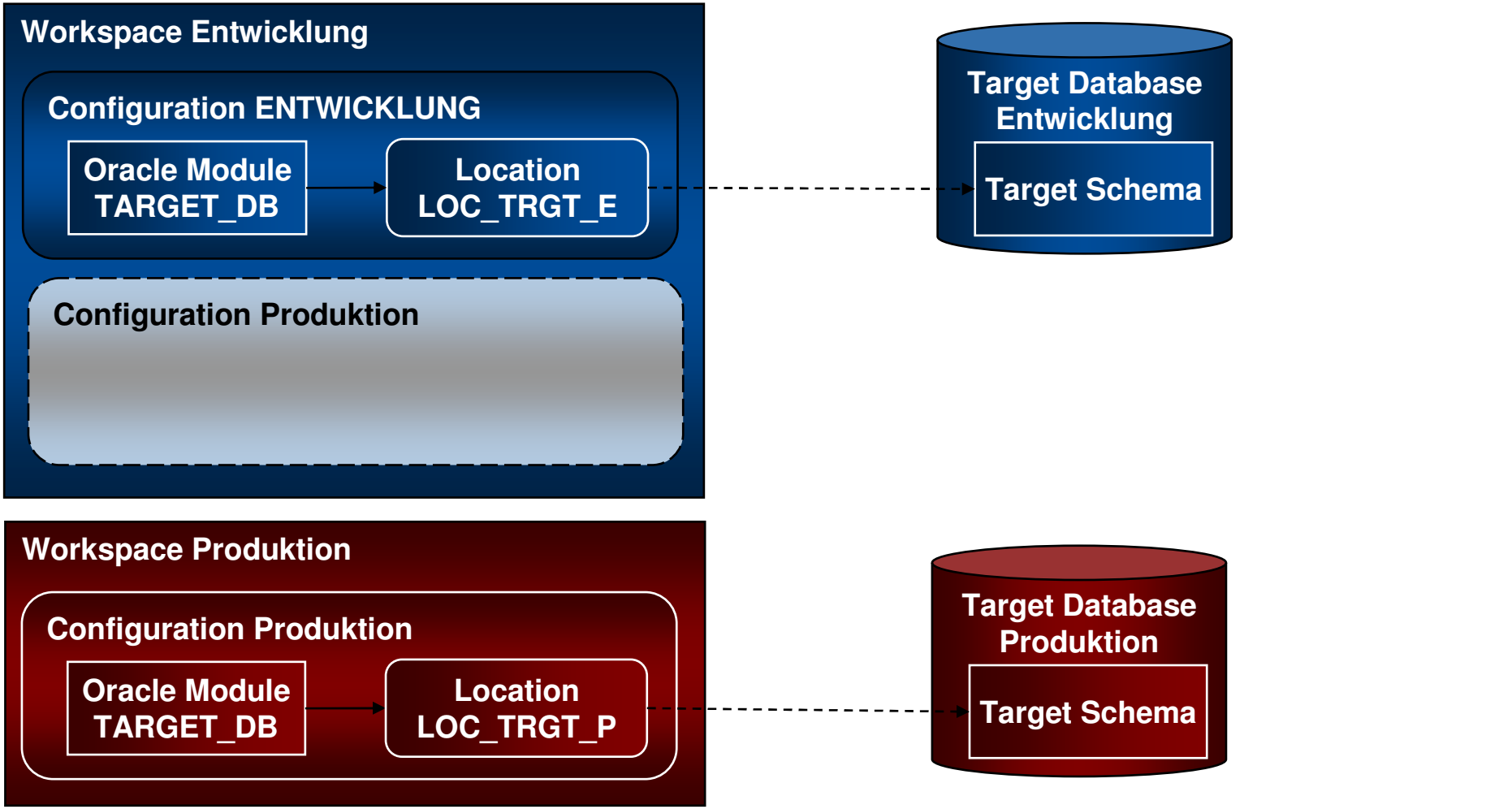
Locations haben auf Entwicklung und Produktion identische Namen, zeigen aber auf die entsprechende Zielumgebung, d.h. Entwicklung oder Produktion

Voraussetzung beim Arbeiten ohne Configurations



In der Entwicklungsumgebung gibt es eigene Locations für Entwicklung (registriert) und Produktion (nicht registriert). Auf den anderen Stages gibt es die jeweils entsprechende Location (registriert).

Voraussetzung beim Arbeiten mit Configurations



Agenda

1 Motivation

2 Architekturen

3 Configurations

4 Setup

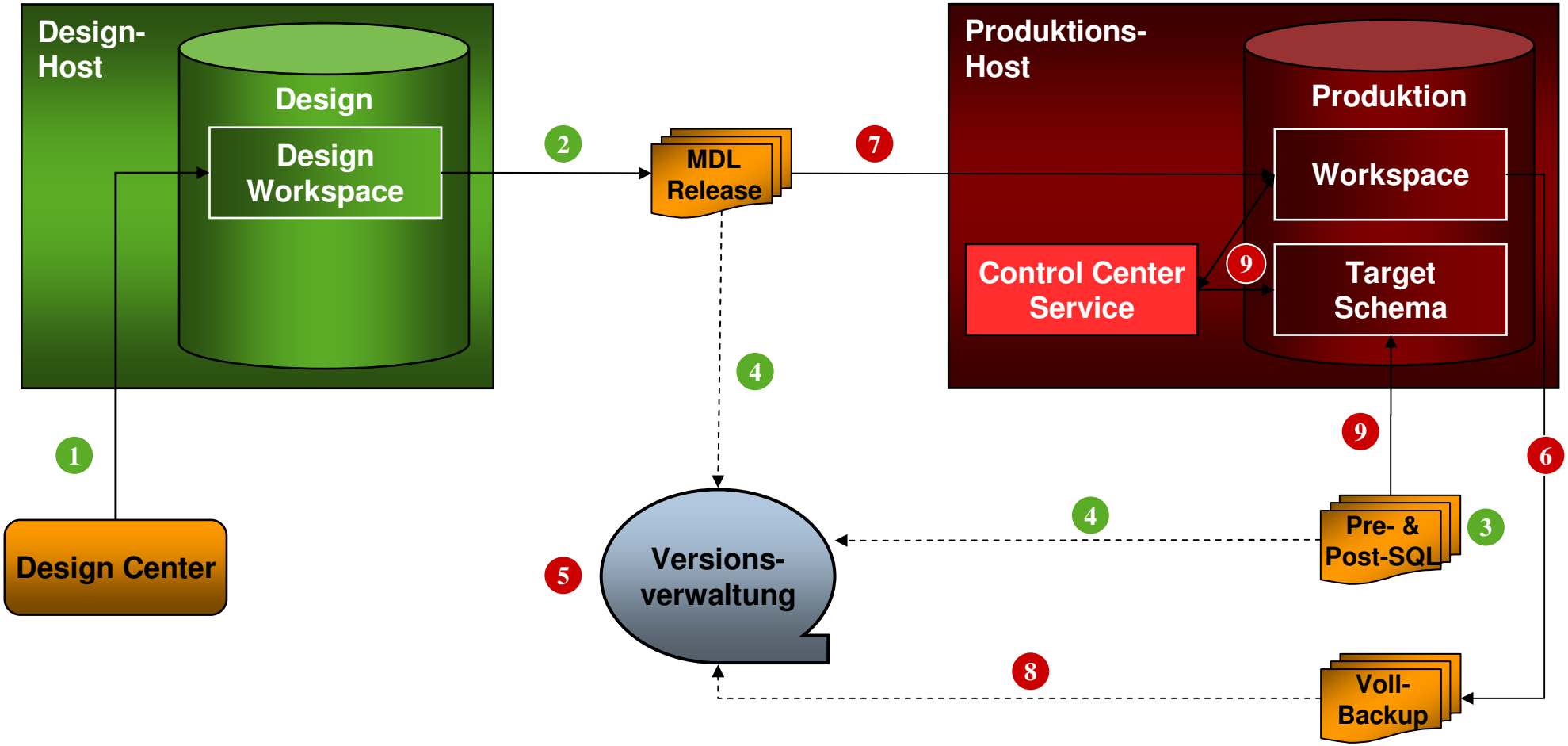
5 **Deployment prozess**

6 Implementierung Deployment prozess

7 Fazit

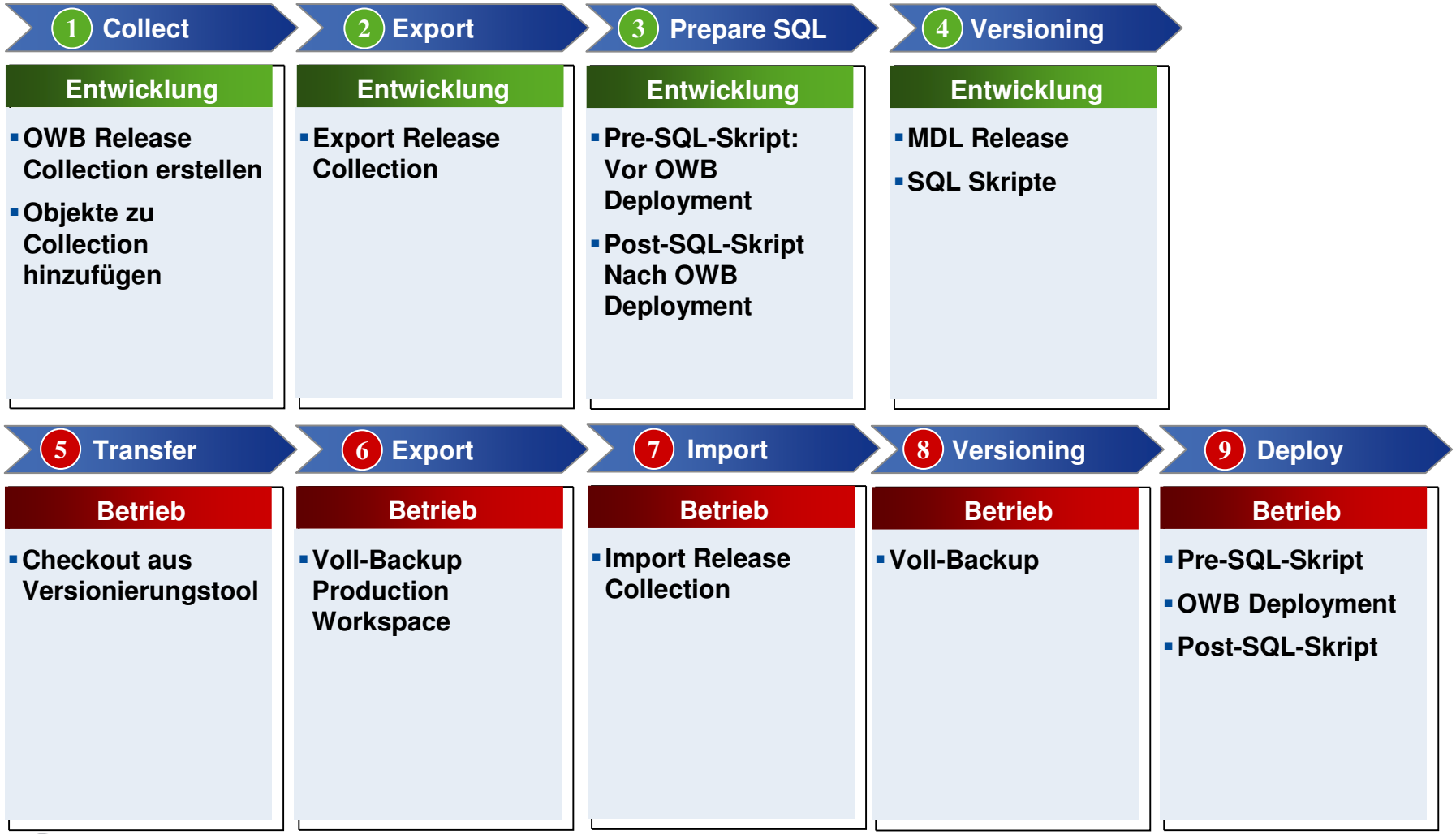
Diese Architektur bietet sich an für Single Instances oder RACs mit Enterprise ETL Option.
Die Entwickler sollten hier Zugriff auf den Produktions-Workspace haben.

Deployment Prozess Single Instance



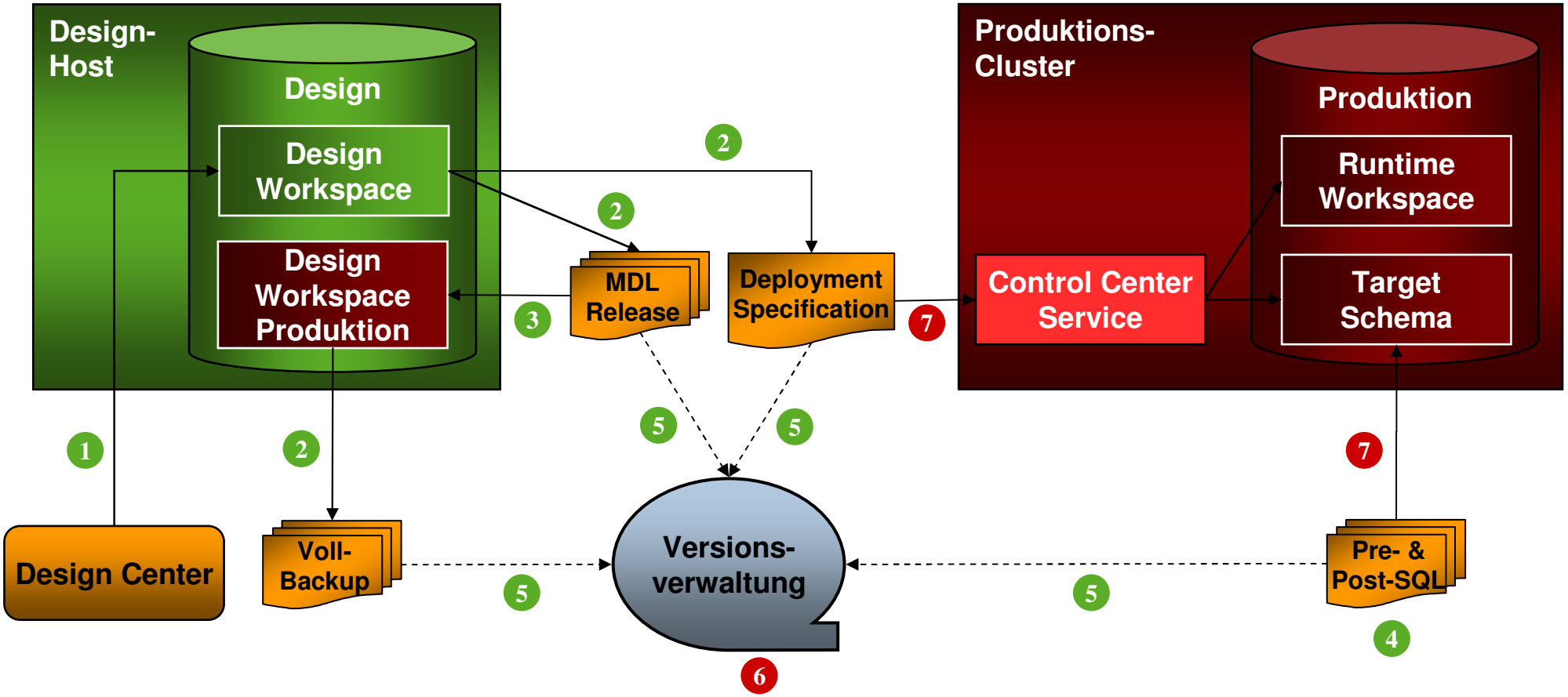
Der Deploymentprozess Single Instance teilt sich in zwei Teile auf: der erste Teil wird von der Entwicklung durchgeführt, der zweite Teil vom Betrieb.

Deploymentprozess Single Instance



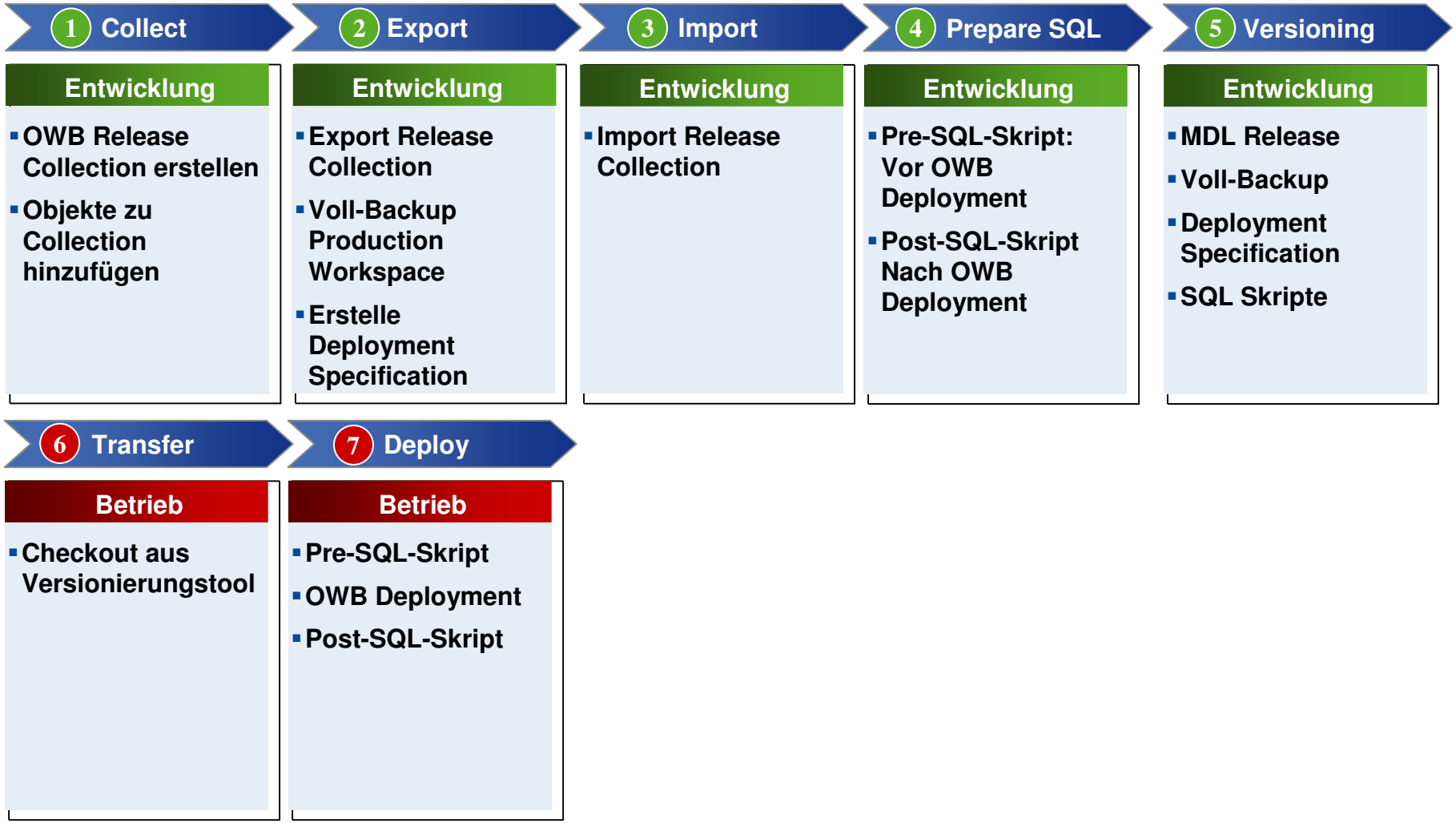
Diese Architektur bietet sich an für RACs ohne Enterprise ETL Option oder Umgebungen in denen die Entwickler keinen Zugriff auf den Produktions-Workspace haben können.

Deployment Prozess RAC



Der Deploymentprozess RAC teilt sich in zwei Teile auf: der erste Teil wird von der Entwicklung durchgeführt, der zweite Teil vom Betrieb.

Deploymentprozess RAC



Agenda

1 Motivation

2 Architekturen

3 Configurations

4 Setup

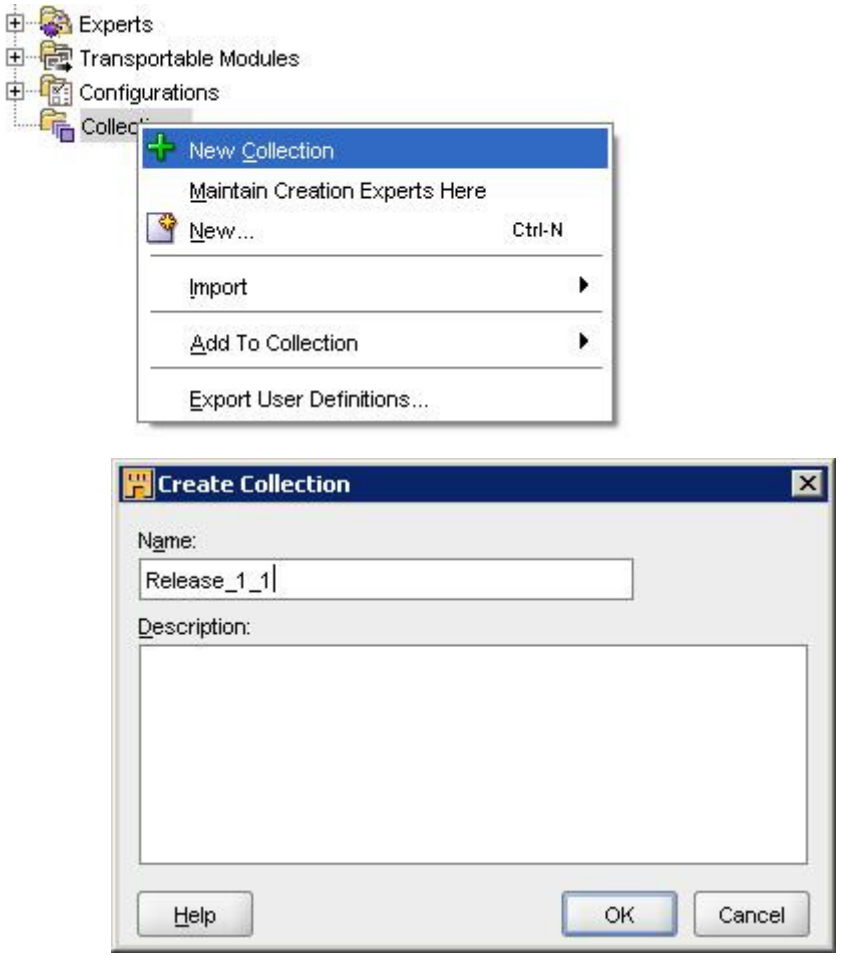
5 Deploymentprozess

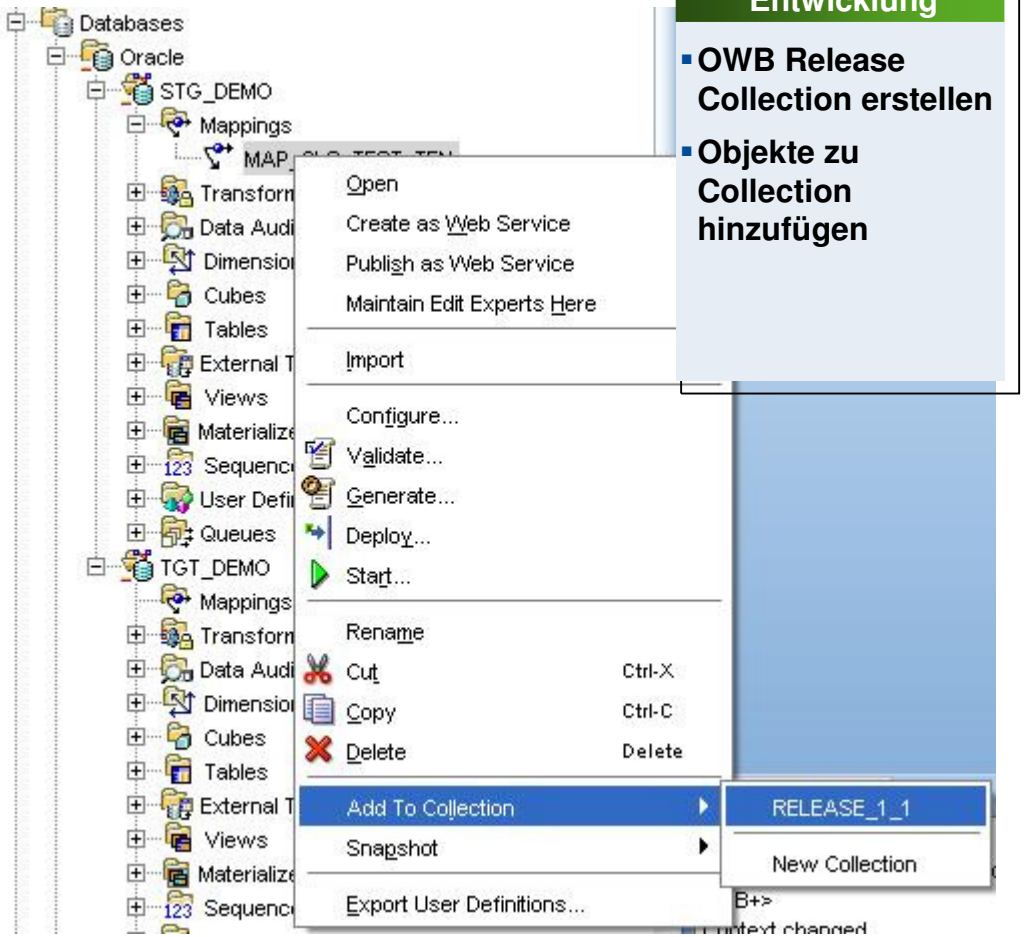
6 **Implementierung Deploymentprozess**

7 Fazit

Für jedes Release wird eine OWB Collection erstellt. Zu dieser werden alle Objekte des Releases hinzugefügt.

OWB Collection





1 Collect

Entwicklung

- **OWB Release Collection erstellen**
- **Objekte zu Collection hinzufügen**

Der Export lässt sich mit dem OMB Befehl OMBEXPORT automatisieren.

Export

Export aller Objekte einer Collection

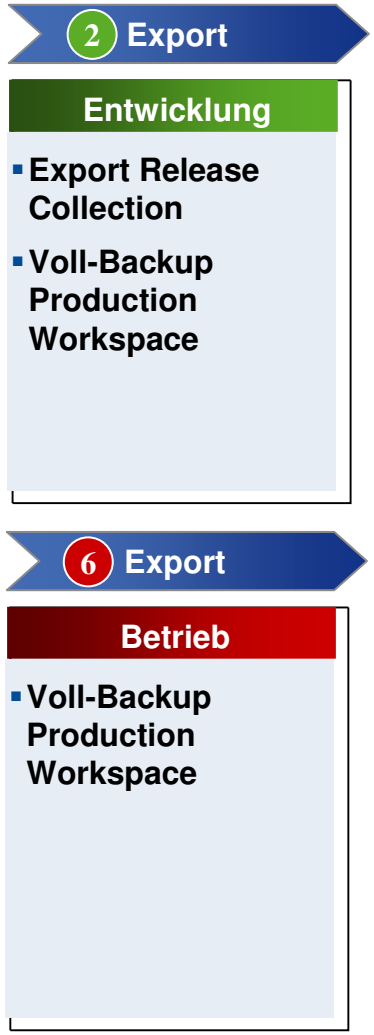
```
OMBEXPORT MDL_FILE '$file.mdl' \
          COMPONENTS (COLLECTION '/$project/$release') \
          DEPENDEE_DEPTH 0 \
          OUTPUT LOG '$file\_exp.log'
```

Export eines gesamten Projektes mit allen abhängigen Objekten

```
OMBEXPORT MDL_FILE '$file.mdl' \
          COMPONENTS (PROJECT '$project') \
          DEPENDEE_DEPTH MAX \
          OUTPUT LOG '$file\_exp.log'
```

file : Bsp: C:\\releases\\owb

release : Bsp: RELEASE_V1



Für die RAC Architektur wird eine Deploymentspecification erstellt und in diese „hineindeployt“. Das Ergebnis ist eine XML-Datei aus der heraus direkt in ein Target deployt werden kann.

Deploymentspecification

```
# Plan erstellen
OMBCREATE TRANSIENT DEPLOYMENT_ACTION_PLAN '$plan'

# Objekte hinzufügen
OMBALTER DEPLOYMENT_ACTION_PLAN '$plan' \
  ADD ACTION '$objectName' \
  SET PROPERTIES (OPERATION) VALUES ('REPLACE') \
  SET REFERENCE $objectType '$objectName'

# Deployment in Specification file
OMBDEPLOY DEPLOYMENT_ACTION_PLAN '$plan' \
  AS SPECIFICATION TO '$file.xml'

# Plan löschen
OMBDROP DEPLOYMENT_ACTION_PLAN '$plan'
```

Der Import lässt sich mit dem Befehl OMBIMPORT automatisieren. Wichtig sind die Einstellungen für den Import Mode (Update) und das Match-by Kriterium (Names).

Import

```
OMBIMPORT FROM MDL_FILE '$file' \
  USE UPDATE_MODE MATCH_BY NAMES \
  ALLOW_DIFFERENT_BASE_LANGUAGE \
  OUTPUT LOG '$file\_imp.log'
```

Es werden ein bis zwei SQL Skripte benötigt: eines vor und eines nach dem OWB Deployment.

Grants

Grants lassen sich per OWB nicht vergeben, dies muss per Skript erfolgen.

Tabellen Änderungen

Bei Änderungen an bestehenden (großen) Tabellen ist es oft sinnvoll, die Änderungen selbst durchzuführen: Neu-erstellen von Tabellen (CREATE TABLE AS SELECT …), usw.

Daten-korrekturen

Änderungen an den Daten müssen per Skript erfolgen. Diese sollten auch immer Teil eines Releases sein (Test, Nachvollziehbarkeit).

PL/SQL

PL/SQL Packages, Funktionen, Prozeduren, Table Functions lassen sich zwar im OWB entwickeln, deutlich komfortabler geht es aber mit anderen Werkzeugen. Die aktuelle PL/SQL-Spezifikation sollte natürlich im OWB importiert worden sein.

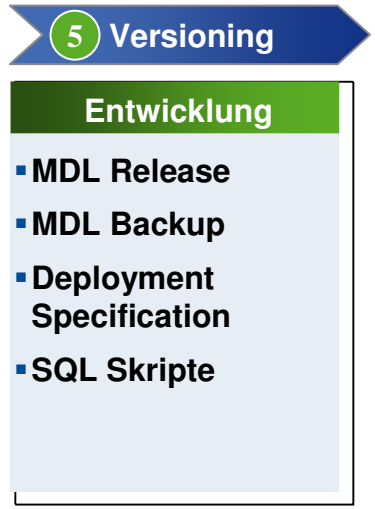
4 Prepare SQL

Entwicklung

- **Pre-SQL-Skript: Vor OWB Deployment**
- **Post-SQL-Skript Nach OWB Deployment**

Alle Dateien zu einem Release bilden ein Release-Package und werden in der Versionsverwaltung eingecheckt.

Versioning



Warum?

- Governance
- Sicherung
- Nachvollziehbarkeit

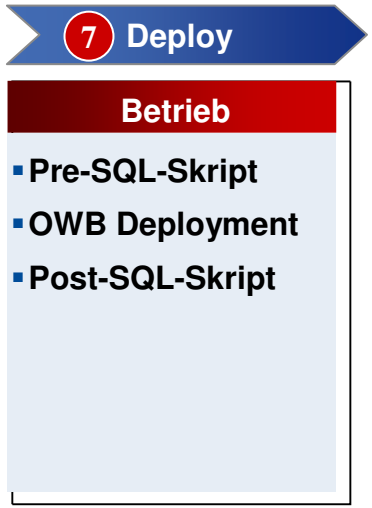
Was?

- OWB Design Daten
- SQL Skripte
- Deployment-Skripte
- Dokumentation (Release Notes)

Per OMB wird ein Deploymentplan erstellt und aus dem Design Repository in das Target deployt.

Deployment aus Repository

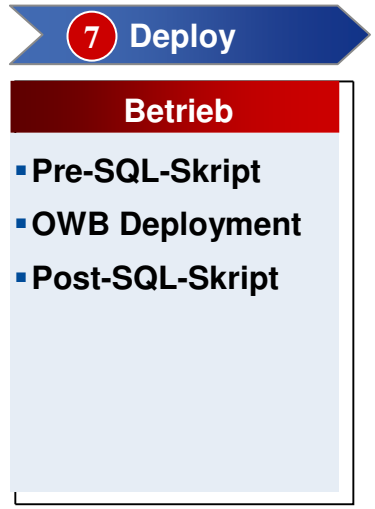
```
# Bei Multiple Configuration
OMBCAC '$configuration'

# Plan erstellen
OMBCREATE TRANSIENT DEPLOYMENT_ACTION_PLAN '$plan'

# Objekte hinzufügen
OMBALTER DEPLOYMENT_ACTION_PLAN '$plan' \
  ADD ACTION '$objectName' \
  SET PROPERTIES (OPERATION) VALUES ('REPLACE') \
  SET REFERENCE $objectType '$objectName'

# Deployment
OMBDEPLOY DEPLOYMENT_ACTION_PLAN

# Plan löschen
OMBDROP DEPLOYMENT_ACTION_PLAN '$plan'
```



Aus dem Specification File wird direkt in das Target deployt. Die Deployment-Actions für die einzelnen Objekte wurden schon bei Erstellung der XML-Datei spezifiziert.

Deployment aus Specification File

```
# Deployment
OMBDEPLOY SPECIFICATION FROM '$file.xml'
```



Agenda

1 Motivation

2 Architekturen

3 Configurations

4 Setup

5 Deploymentprozess

6 Implementierung Deploymentprozess

7 **Fazit**

Mit der richtigen Architektur und einem automatisierten Deploymentprozess werden alle Anforderungen erfüllt.

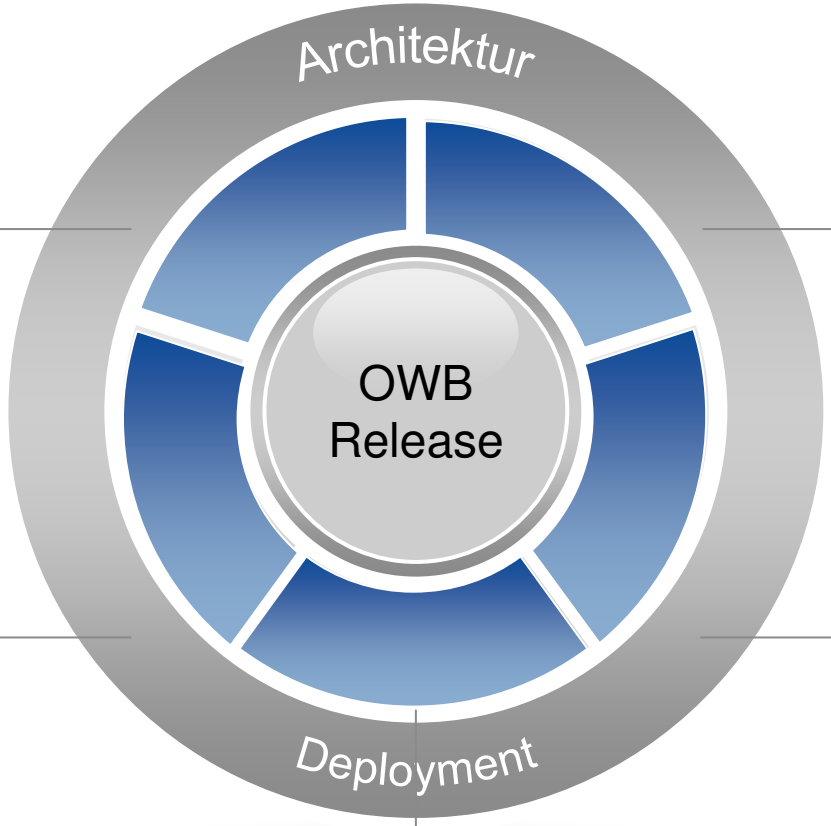
Fazit OWB Architektur & Deploymentprozess

Automatisiert

Skript-basiertes Deployment vermeidet Fehler

Einfach

Wenig Aufwand für Entwicklung & Betrieb



Integriert

Nutzung bestehender Versionsverwaltungstools

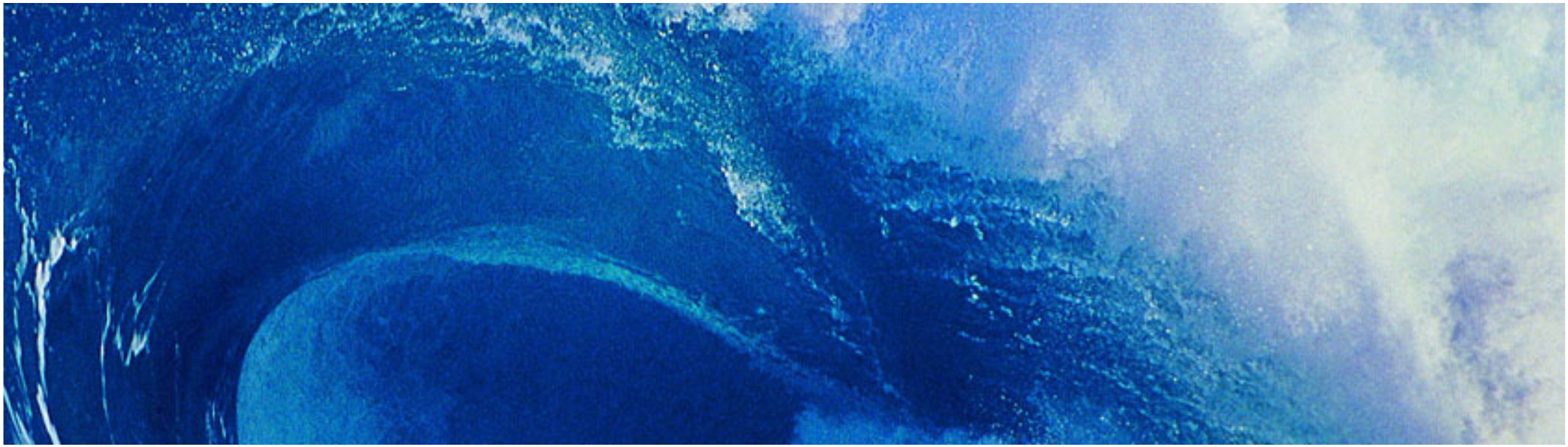
Nachvollziehbar

Wann wurde was deployt?

Transparent

Auf welcher Stage ist was deployt?

metafinanz



Herzlichen Dank!

metafinanz Informationssysteme GmbH
Leopoldstr. 146
Phone: +49 89 360531-0
Fax: +49 89 350531-5015
Email: kontakt@metafinanz.de
www.metafinanz.de