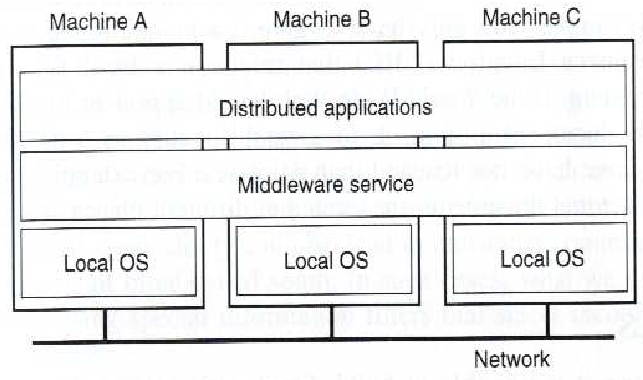
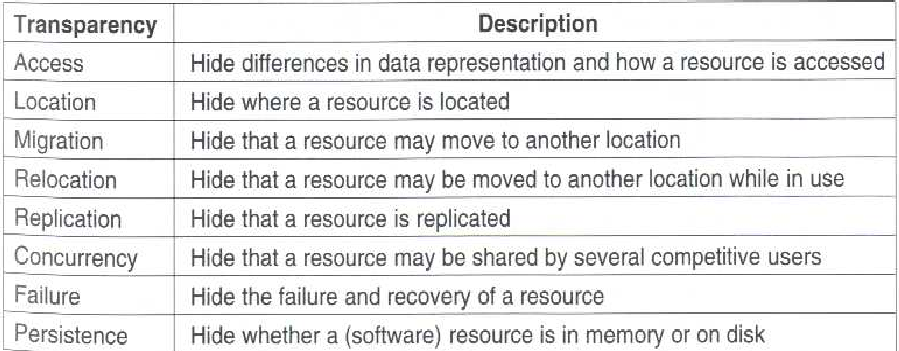
**Zusammenfassung VSY**

René Bernhardsgrütter – 19.05.2013

# Intro

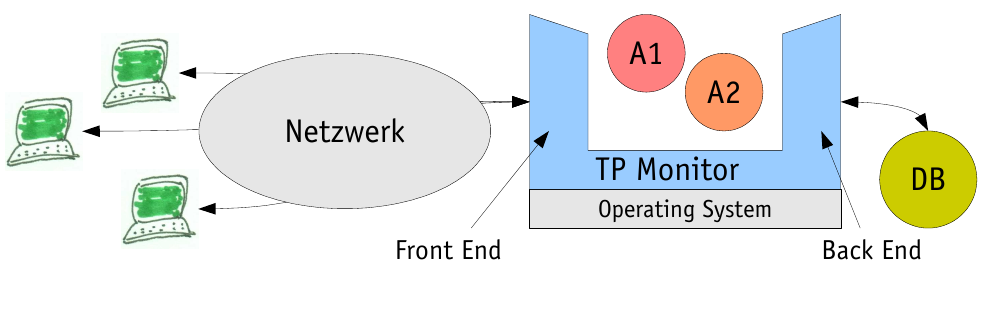
A distributed system is a collective of independent computers that appears to its users as a singe coherent system.

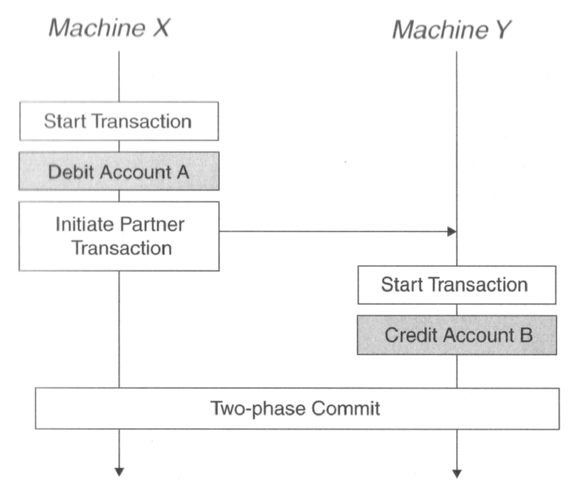




## Transaktioen

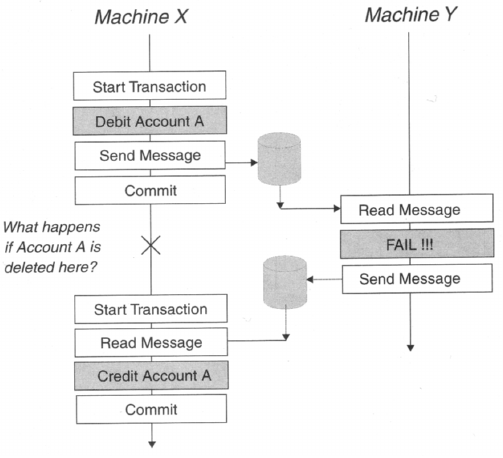
Mit dem „Teleprocessing“ entstand „Transaction Processing“

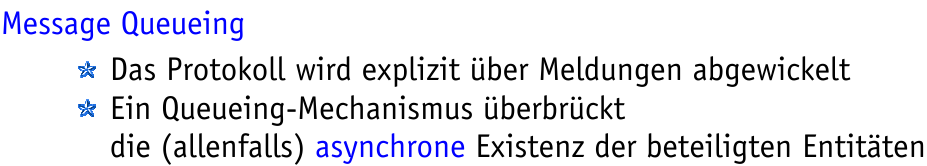




## Message Queuing

Entkoppelt von Anwendungen: Asynchron!

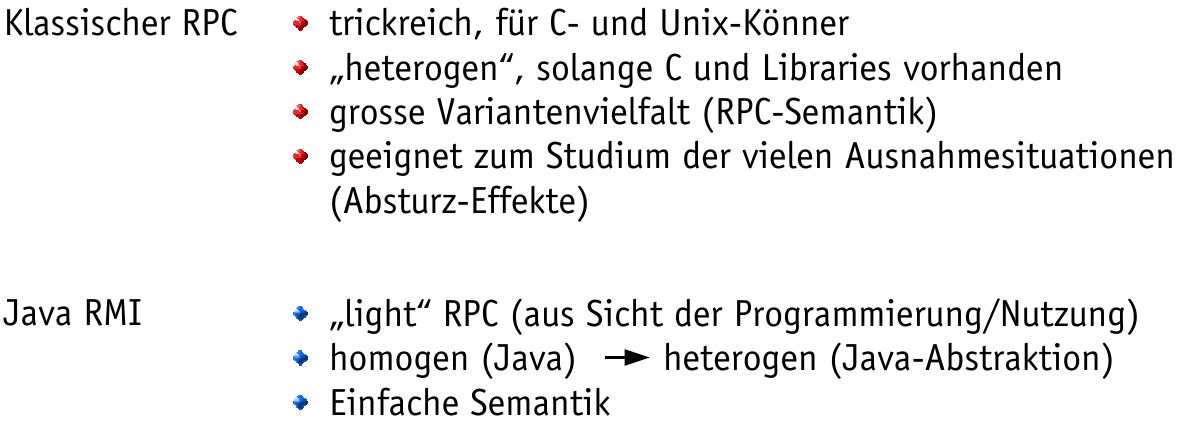


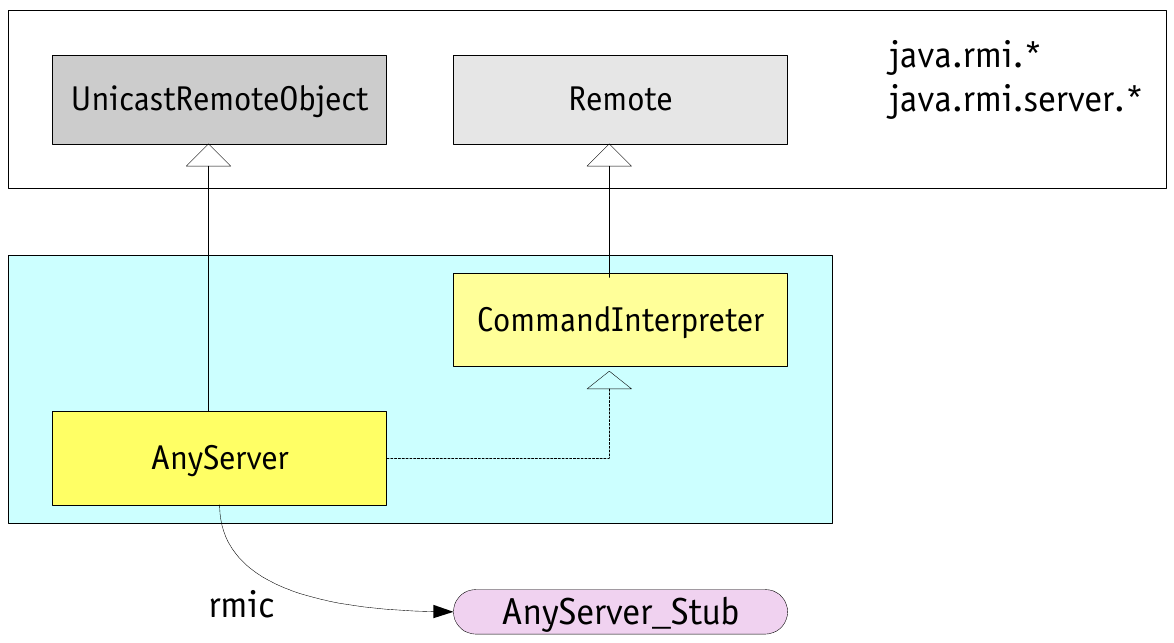


## Middleware…

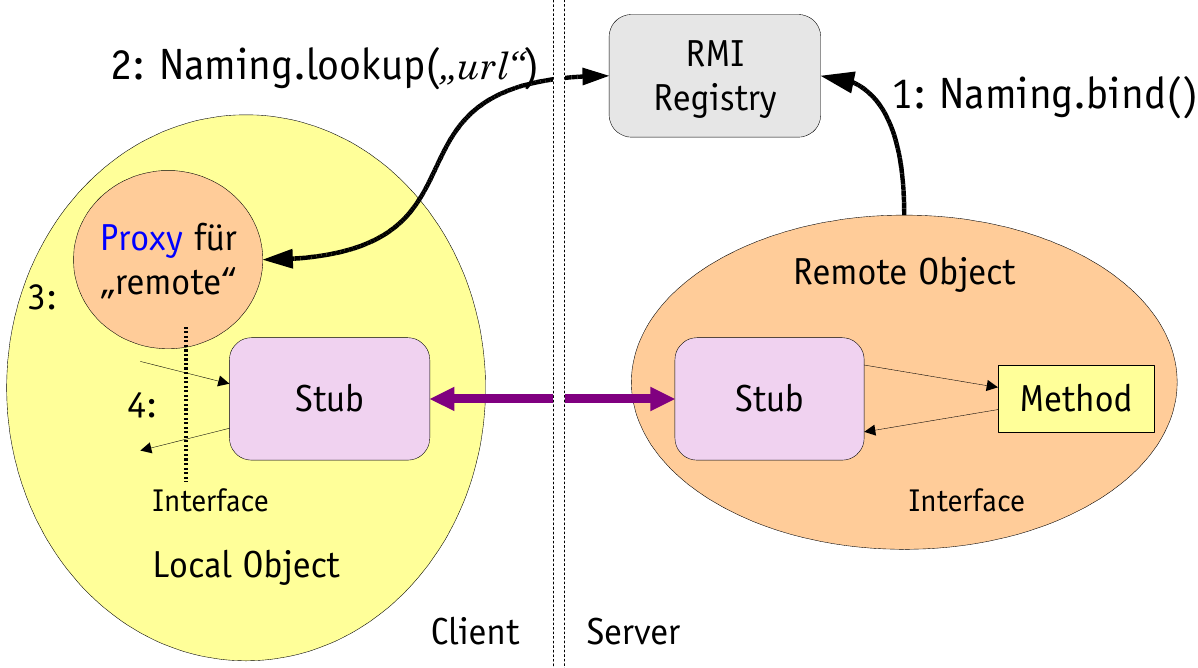
* …ist das, was macht, dass sich die Anwendung und damit die Anwendungsentwicklung auf die Business-Logik konzentrieren kann.
* …hilft zu einer Abstraktion der Business-Prozess-Topologie weg von Topologie der aktuellen Ressourcen und deren spezifischen Features.

# RMI

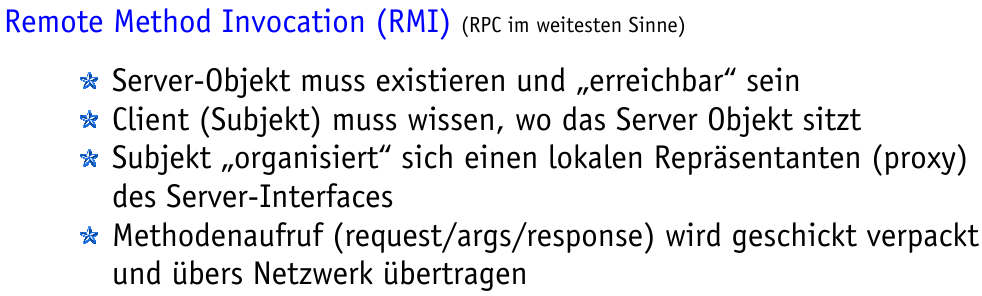




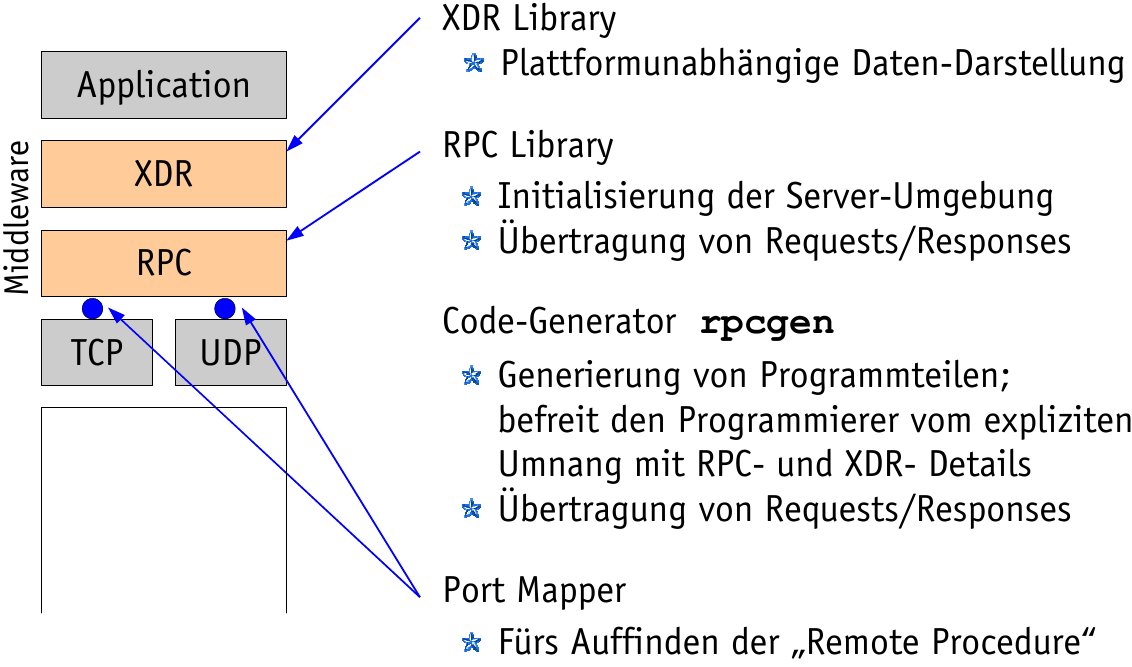
## RMI Verbindugsaufnahme



## Lebensräume



# RPC



## XDR (eXternal Data Representation)

**Gleich auf allen Systemen**:

* Zeichendarstellung
* Bit/Byte-Ordnung (little endian)
* Darstellung von Floatingpoints
* Alle Datentypen vielfaches von 4 Bytes
* Gibt flexiblen string-Type
* Structs wie in C



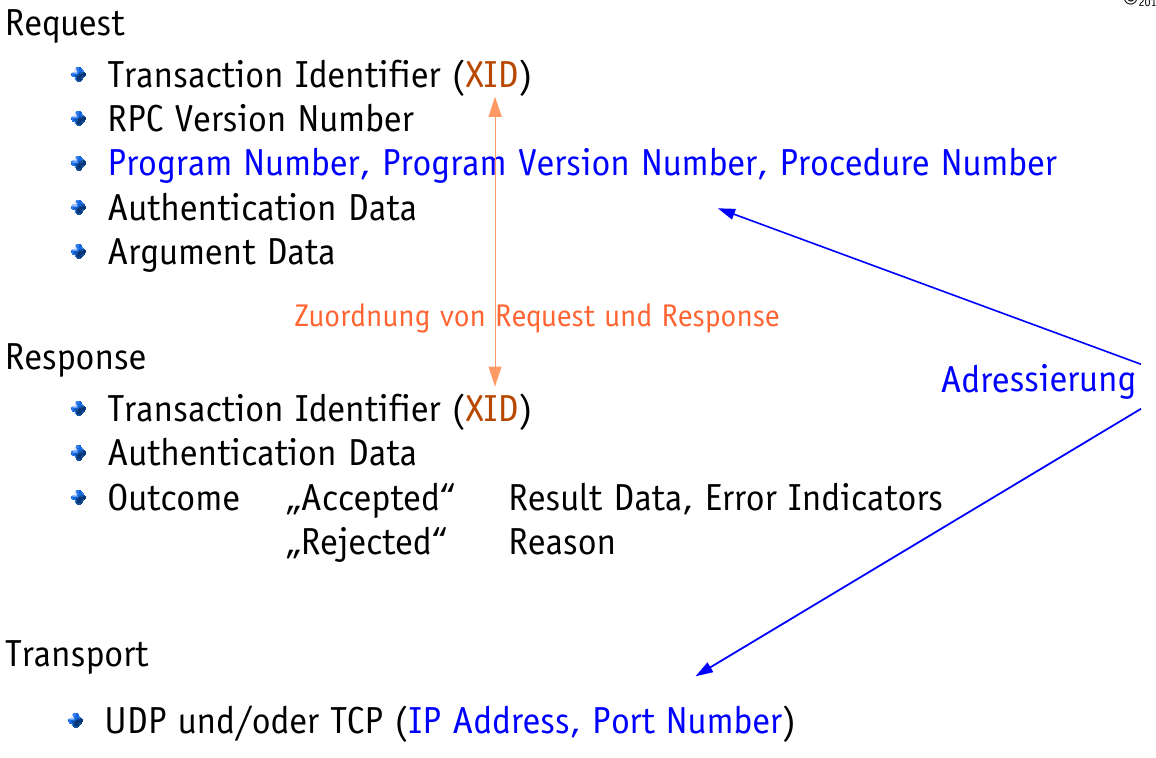
**Marshalling** (=Serialisierung)

Es kann sein, dass man, z. B. in C, die int konvertieren muss, um der XDR zu entsprechen!

## Adressierung von Funktionen

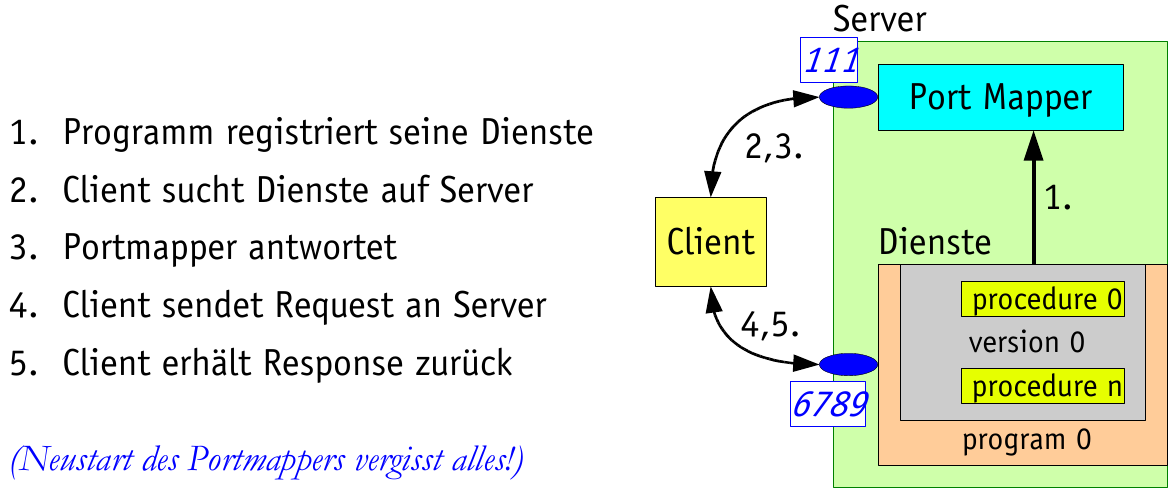
Dreistufiges Finden von Funkionen:

1. Service Identifier (Programm auf dem Server)
2. Programm Version Number
3. Procedure Number (Funktionsnummer)



Portmapper: Programm <=> UDP/TCP Port

*portmapper* unter Linux:



### rpcgen

Erzeugt Stub-Prozeduren für die jeweilige Architekturen:

* Adresseirung
* Verbindung
* Datenformate
* Überwachung

**Einer RPC-Transkation/-Funtionsaufruf kann höchstens ein Argument übergeben werden!**

Man kann aus .x-File …\_server und …\_client erzeugen, die man nachher ausprogrammieren muss.

*A* ***method stub*** *or simply* ***stub*** *in software development is a piece of code used to stand in for some other programming functionality. A stub may simulate the behavior of existing code (such as a procedure on a remote machine) or be a temporary substitute for yet-to-be-developed code.*

## .x-File

struct argumente {

int a;

int b;

};

program ADD\_PROG {

version ADD\_VERS {

int ADDIEREN ( struct arg ) = 1;

// procedure ID (above)

} = 1; // program Version

} = 0x31271966; // program ID

PC ruft remote die Nummern, nicht die Namen auf.

Konstanten, die von rpcgen in den Headerfiles gesetzt werden.

In C: ADDIEREN\_1, wobei das 1 die Version ist.

Alle Rückgabewerte der Prozeduren kommen als Pointer! Also bei ADDIEREN kommt ein int \* value an.

Die Agrumente sollten Pointer auf structs enhalten!

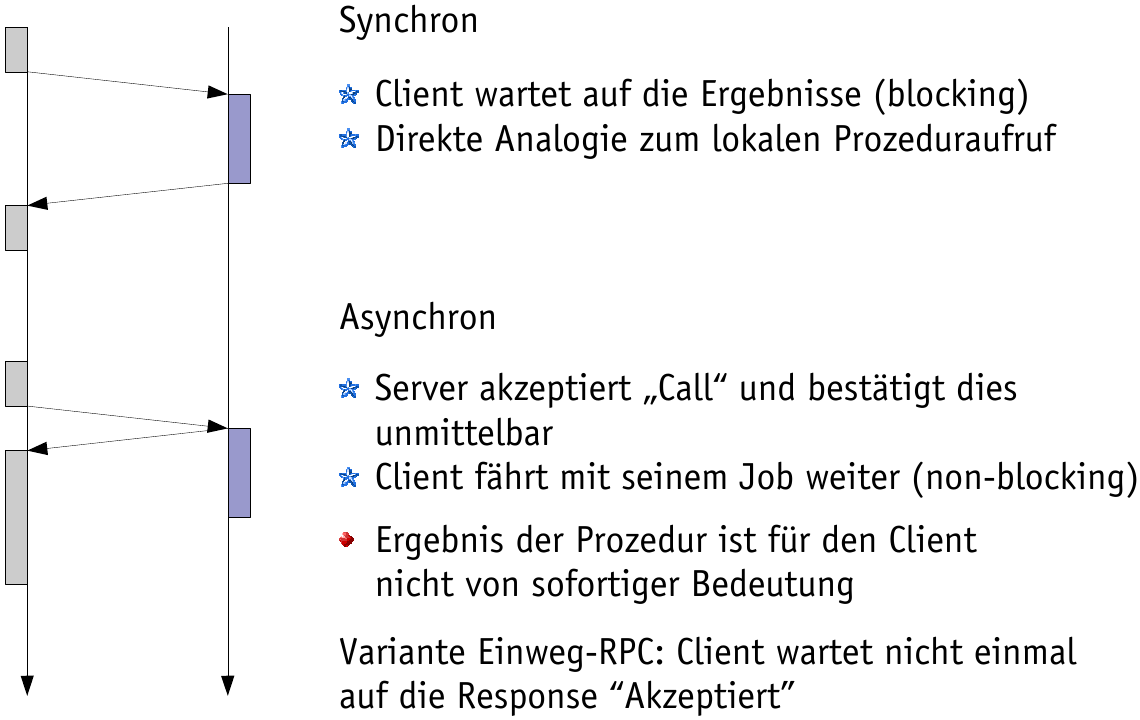
## Verbindung aufbauen

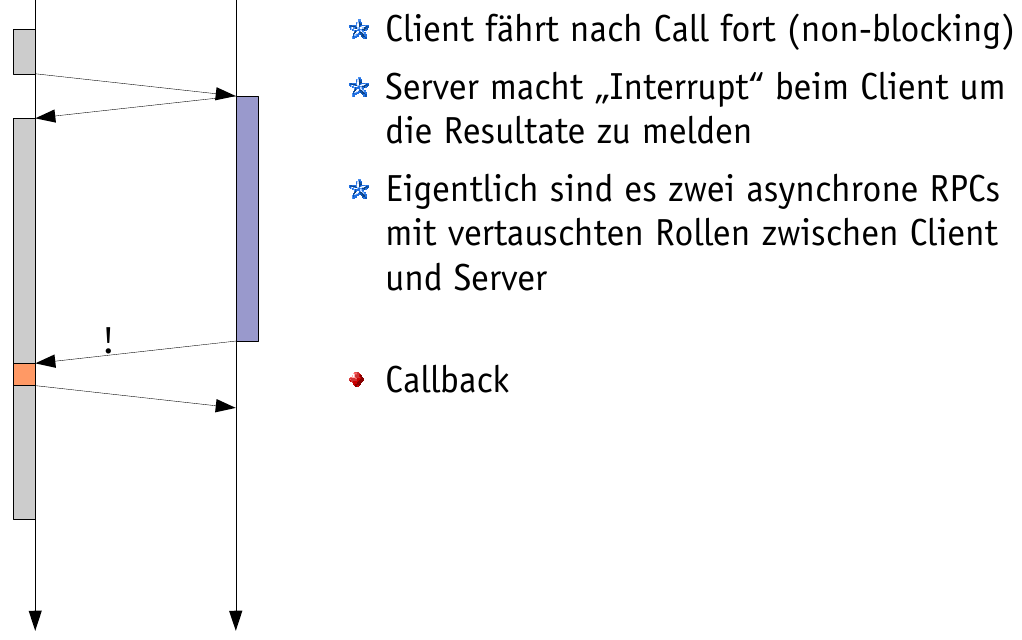
Bei Porgrammstart wird portmapper-Server-Adresse übergeben. Im Programm wird darauf zugegriffen und die Verbindung hergestellt. Die einzelnen RPC-Funktionen werden dann normal aufgerufen (z. B. int myResult = addieren\_1([1, 2], c1); wobei c1 vom Typ CLIENT ist (wird erzeugt)).

Server registriert Dienst für UDP- und TCP-Funktionen bei dem portmapper (Registrar) und startet dann den eigentlichen Server. Dort sind die eigentlichen Funktionen (z. B. addieren\_1(..) vorbereitet).

## Syc/Async

RPC unterstützt beides.



Kann auch „Verzögerter Synchroner RPC“:  


## Broadcast

Der Request wird vom Client an mehrere Server geschickt. Er kann auch mehrere Antworten handeln.

## Fehlerfall-Handling

* Maybe (wird ignoriert)
* At most once
* At least once
* Exactly once

Problem: Client kann Fehler auf der Serversiete nicht zuverlässig erkennen, denn:

* Kein Absturz: OK oder…
* … keine Response: Absturz nach der Ausführung oder…
* … keine Response: Absturz vor der Ausführung

Die letzten beiden Fälle unterscheiden sich vom Client aus nicht.

### At most once

* Client **sendet Request genau 1x**
* Responses ignorieren
* Es braucht keinen Timer

### Exactly once

* Request **sooft senden, bis Response eintrifft**
* Benötigt clientseitig einen Timeout, bis Wiederholung (im Fehlerfall)
* Braucht **Stateful Server**, damit mehrere gleiche Requests nur 1x ausgeführt werden

**Besondere Massnahme bei Fehlerfall**:

* Commit: Bestätigung bei vollständiger Erledigung (Server an Client)
* Rollback: Abbruch und Zurücknahme der Bearbeitung bei Fehler
* Checkpoints: Sicherung von definierten Zuständen zur Wiederaufnahme der Bearbeitung

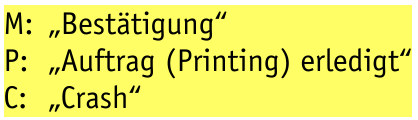
Bsp:

* Client sendet Text an Printserver
* Server sendet Empfangsbestätigung
* Server stürzt aber ab
* Nach Absturz/Neustart meldet Server „wieder online“
* Die Übertragung läuft das nächste Mal korrekt

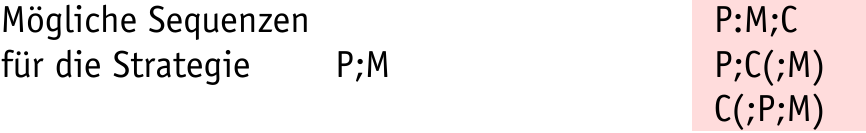
### At least once

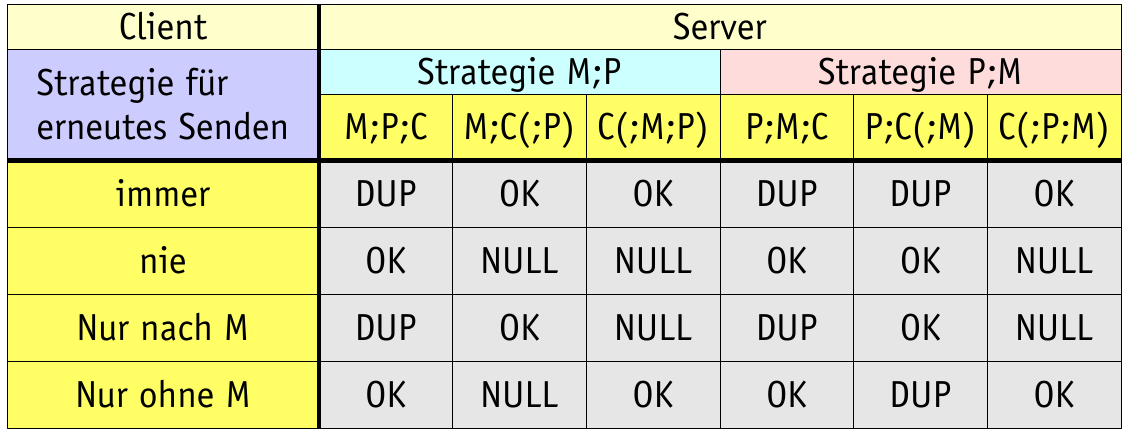
* Request **sooft senden, bis Response eintrifft**
* Benötigt clientseitig einen Timeout, bis Wiederholung (im Fehlerfall)

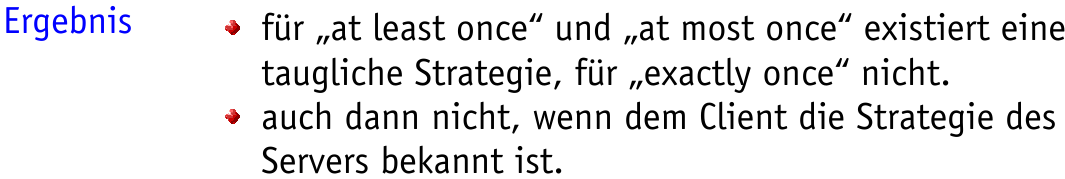
### Systematische Betrachtung

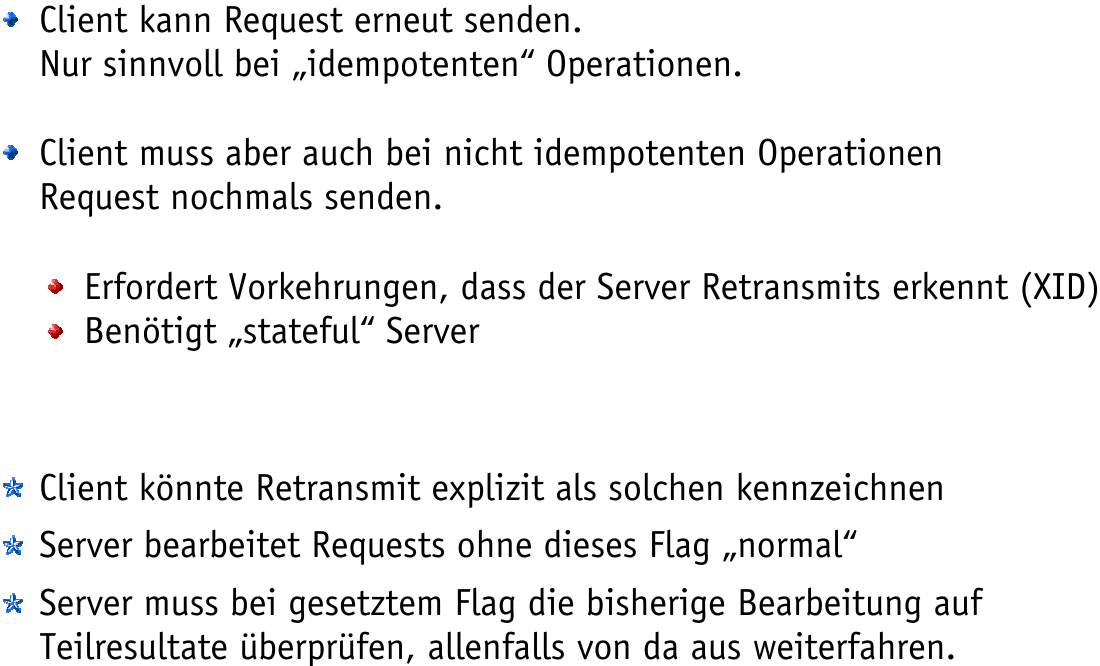












*In computing, an* ***idempotent*** *operation is one that has no additional effect if it is called more than once with the same input parameters.*

### Orphans

Sind “Waisenkinder” auf dem Server, wenn der Client abstürzt. Der Server kann dem Client nicht antworten.

### Nach Clientabsturz

**Reincarnation**: Neuer Sequenzieller Epoche (=Zeitabschnitt) beginnt (bei jedem Programmstart) und wird allen Servern mitgeteilt. Nur aktuelle Epoche ist gültig.

**Soft Reincarnation**: Server sucht zu Orphans den zugehörigen Client. Wenn gefunden, werden Orphans in neue Epoche überführt.

**Expiration**: Jeder Request hat einen Timeout-Wert τ. Sobald abgelaufen fordert Server bei Client eine Verlängerung an. Wenn dies nicht klappt, wird Operation gelöscht.

## RPC-ZUF

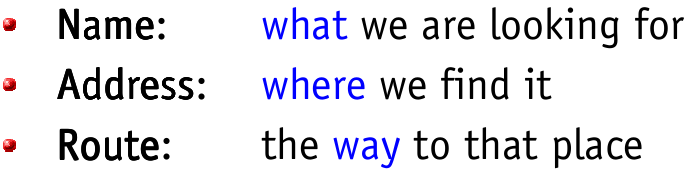
* Verteilte Apps oberhalb low-level Socket-Kommunikation
* Vor-Objektorientiert
* Heutiges (RMI, CORBA, SOAP) haben grundsätzlich selbe Fkt/Probleme

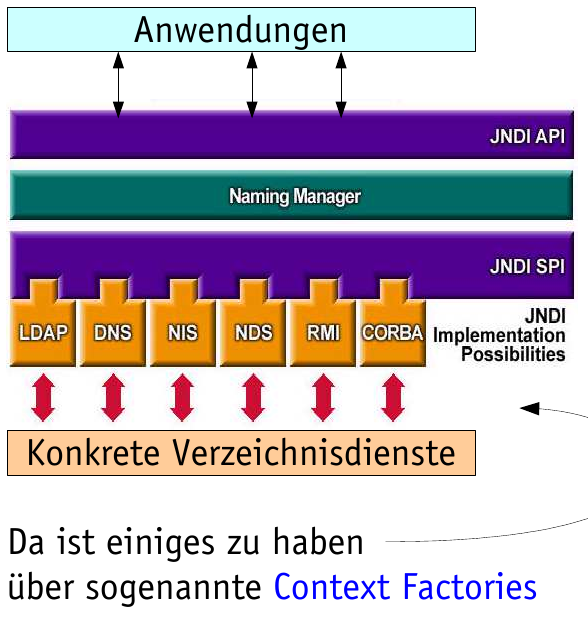
Bei Remote-Aufruf:

* Min. 3 Systeme involviert
* Keine gesicherte Information über Gesamtzustand gegeben
* Unsicherheit über andere Systeme bzgl. Kommunikations- und Überlebensstrategie bleibt

# JNDI Verzeichnisdienst in Java

=Java Naming and Directory Interface.





Bindings zwischen Adresse und Objekt sind in einem **Context** als konkrete (=instanziierte) Sammlung von Bindings haltbar.

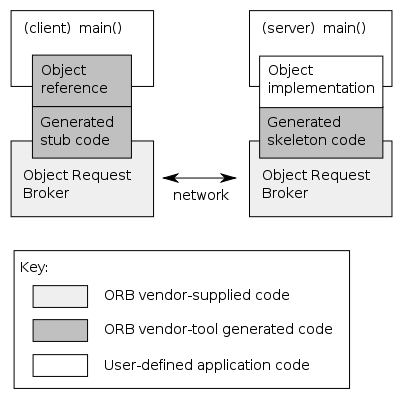
# Logging

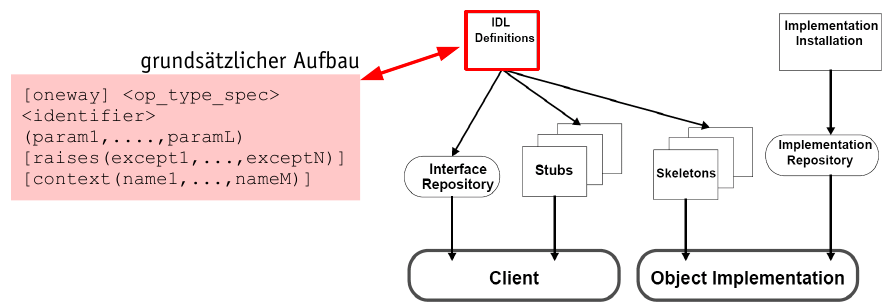
org.apache.log4j.Logger log;

log.debug(„{0} was called :-)”, “myMethod”);

* Beliebige Appender möglich (=wohin etwas geloggt werden soll, z. B. Logfile oder DB)
* Verschiedene Categories möglich (DEBUG, INFO, WARN, ERROR …)
* Hierarchisches Namenssystem (ch von ch.zhaw ist oberhalb von zhaw)
* Log-Level einschränkbar (z. B. auf ch.\* ERROR, auf ch.zhaw INFO)

# CORBA

Architektur:  


Oder auch so:  


### Objects By Reference

Standardmässig werden Objects By Reference gehandelt. Wenn der Client eine Referenz vom Server-Objekt holt, wird der Zugriff auf das konkrete Objekt, welches sich auf dem Server befindet, via **stub und Object Request Broker auf der Clientseite** und via **Object Request Broker und skeleton auf der Serverseite** gematched.

### Data By Value

Primitive Datentypen wie z. B. Integers oder Booleans werden als Values übergeben.

### Objects By Value

CORBA und RMI-IIOP definieren Objects By Value, wodurch ganze Objekte übertragen werden können. Der Methodencode wird normalerweise lokal, also auf dem Client ausgeführt. Der Code kann entweder beiden Seiten bekannt sein, oder der Client lädt den Code vom Server runter. Remote Methods sind aber dennoch möglich.

### Critics

#### Location transparency

CORBA's notion of location transparency has been criticized; that is, that objects residing in the same address space and accessible with a simple function call are treated the same as objects residing elsewhere (different processes on the same machine, or different machines). This notion is flawed if one requires all local accesses to be as complicated as the most complex remote scenario. However, CORBA does not place a restriction on the complexity of the calls. Many implementations provide for recursive thread/connection semantics. I.e. Obj A calls Obj B, which in turn calls Obj A back, before returning.

#### Design and process deficiencies

The creation of the CORBA standard is also often cited for its process of design by committee. There was no process to arbitrate between conflicting proposals or to decide on the hierarchy of problems to tackle. Thus the standard was created by taking a union of the features in all proposals with no regard to their coherence. This made the specification very complex, expensive to implement entirely and often ambiguous.

#### Problems with implementations

Through its history, CORBA has been plagued by shortcomings in its implementations. Often there have been few implementations matching all of the critical elements of the specification,[2] and existing implementations were incomplete or inadequate. As there were no requirements to provide a reference implementation, members were free to propose features which were never tested for usefulness or implementability. Implementations were further hindered by the general tendency of the standard to be verbose, and the common practice of compromising by adopting the sum of all submitted proposals, which often created APIs that were incoherent and difficult to use, even if the individual proposals were perfectly reasonable.

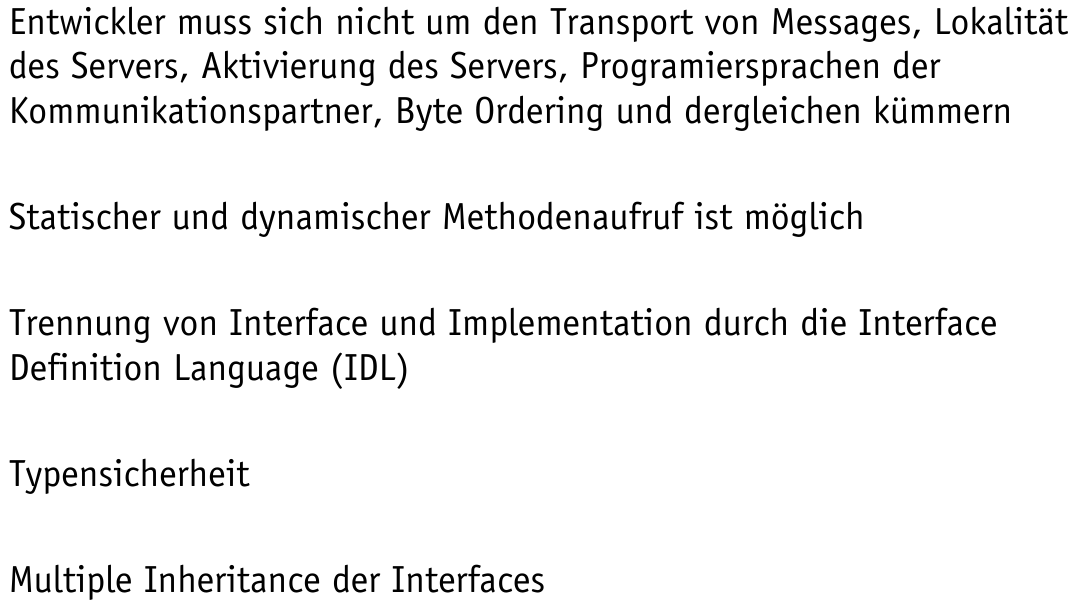
# CORBA IDL

= **Interface** Definition Language

* Mapping zu jeder Sprache definiert
* Von OMG definiert (Object Management Group)
* Aktuell
* **Objekte by reference, dort gespeichert, wo sind hergestellt wurden**

## Erzeugung

* Eine Deklaration, umwandelbar in „Target Code“
* Sieht syntaktisch „gewohnt“ aus, ist aber weder C/C++ noch Java.
* Methoden-Signaturen spezifizieren neben Parameter-Typen auch ob es sich um „**input**“ oder „**output**“ oder „**inout**“ handelt.



## Interfaces

interface Account {

struct personalInfo {

string name;

string age;

};

attribute string owner;

readonly attribute float balance;

const short MAX\_NAME\_LEN = 100;

void makeDeposit(in float amount);

boolean makeWithdrawal(in float

amount,

out float

blance);

oneway void notify(in string s1);

};

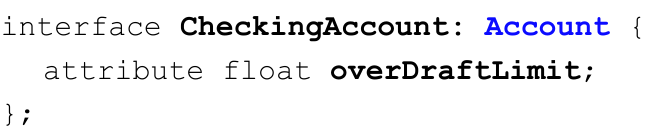
* Gibt Attribute und Operationen (Fkt)
* Attribute auch readonly möglich. Diese sind allerdings nicht zwingend konstant, denn balance kann von anderen Attributen beeinflusst werden.
* Operationen können Returnvalue und in und/oder out-Parameter haben
* Oneway ist **nonblocking/async**, der Aufrufende kann sofort weiterarbeiten. Können keine out oder inout Parameter haben, sie erfordern den Rückgabetyp void!
* interfaces sind (wie in Java) Typen. Man könnte z. B. ein interface Bank definieren, dass etwas vom Typ Account benötigt. Diese könnten über- und zurückgegeben werden.
* Konstanten mit const möglich!
* Structs mit struct möglich!

## Objektreferenz

Account newAccount(in string name);

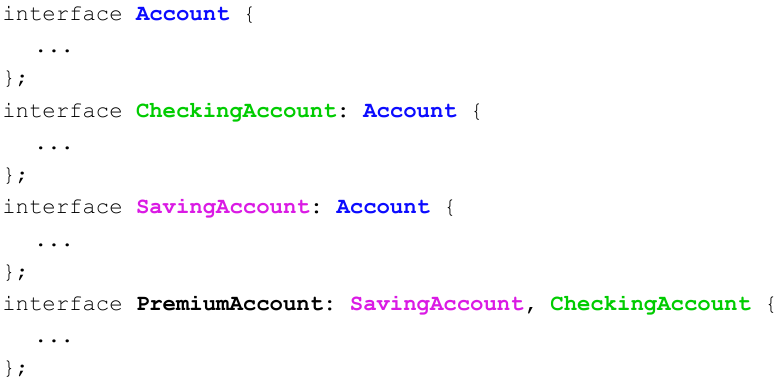
newAccount liefert eine Referenz auf ein Account-Objekt zurück.

## Vererbung



* Erbt alle Attribute und Operationen von Account. Wie CheckingAccount extends Account in Java.

### Mehrfachvererbung



**Achtung: Diamond-Inheritance** möglich durch Mehrfachvererbung!

*In Java:*

***Overloading****: Methoden haben unterschiedliche Signaturen, sind parallel möglich.*

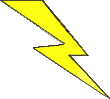
***Overriding****: Gleiche Signaturen, ‚überschreibend‘*

# RMI over CORBA IIOP

**IIOP = Internet Inter-ORB-Protokoll**

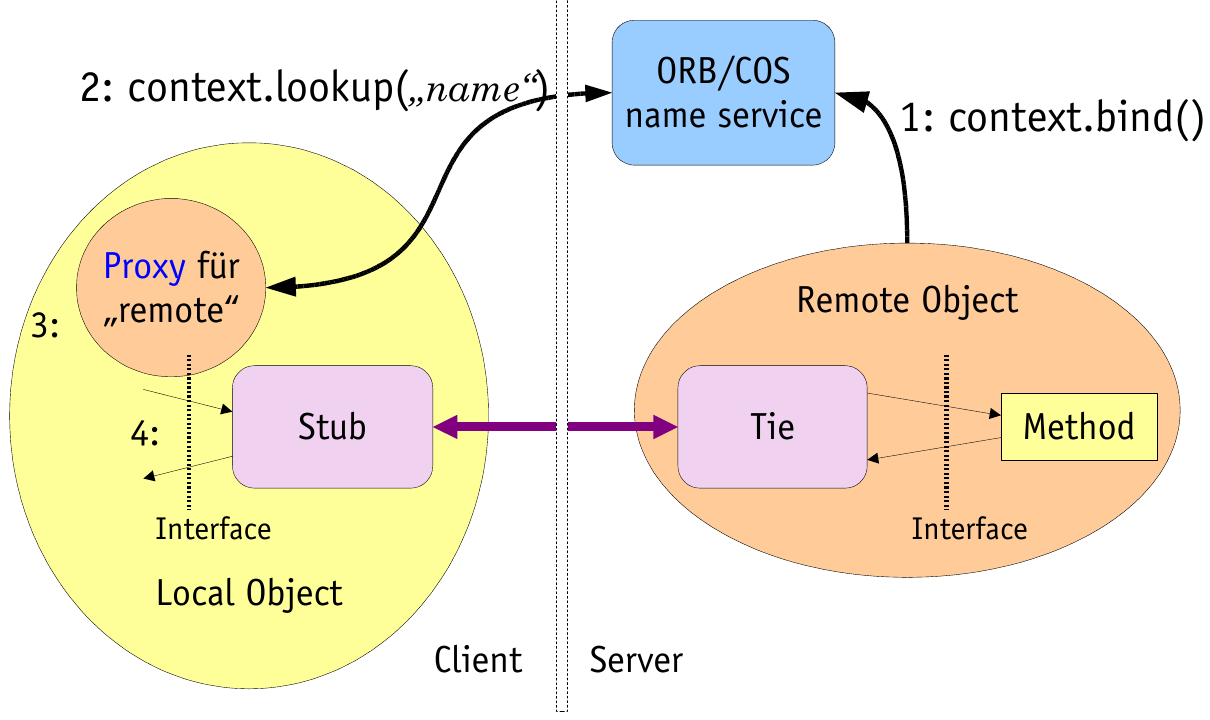
RMI-IIOP denotes the Java Remote Method Invocation (RMI) interface over the Internet Inter-Orb Protocol (IIOP), which delivers Common Object Request Broker Architecture (CORBA) distributed computing capabilities to the Java 2 platform.

* RMI = „normalen“ Methode via Proxy Objekt, mit Interface des Server Objekts à distance.
* Call By Value
* Klasse von Parameter-Objekten komplett (mit Methoden) **kopiert**



* CORBA = ortstransparente Original-Objekte, via Referenzen erreichbar, und IDL definiert, wie deren Methoden anzusprechen sind.

Mit RMI over CORBA IIOP kann Java mit CORBA arbeiten.



Braucht zudem ORB-Naming-Context.

RMI-IIOP is largely based on the Object by Value concept that serves as a container or direct replacement for CORBA structures, unions, sequences, arrays and strings. No separate IDL is necessary. Instead, the data structure definitions are "supposed" automatically, collecting the necessary data via reflection mechanisms.

Both CORBA and RMI-IIOP uses the same General Inter-ORB Protocol (=IIOP) communication standard. If required, it is possible to generate the IDL definitions for the involved RMI-IIOP data structures and use these definitions to arrange the interoperability between the RMI-IIOP and plain CORBA applications.

# REST

**Stateless**: Sessioninformationen nur beim Client, jeder Request kann einzeln behandelt werden.

**Uniform Interface**: Einzigartige Identifikation von Ressourcen, Manipulation möglich

**Layered System**: Jede Schicht sieht nur den nächsten Layer.

GET: Darf den Serverstatus nicht ändern

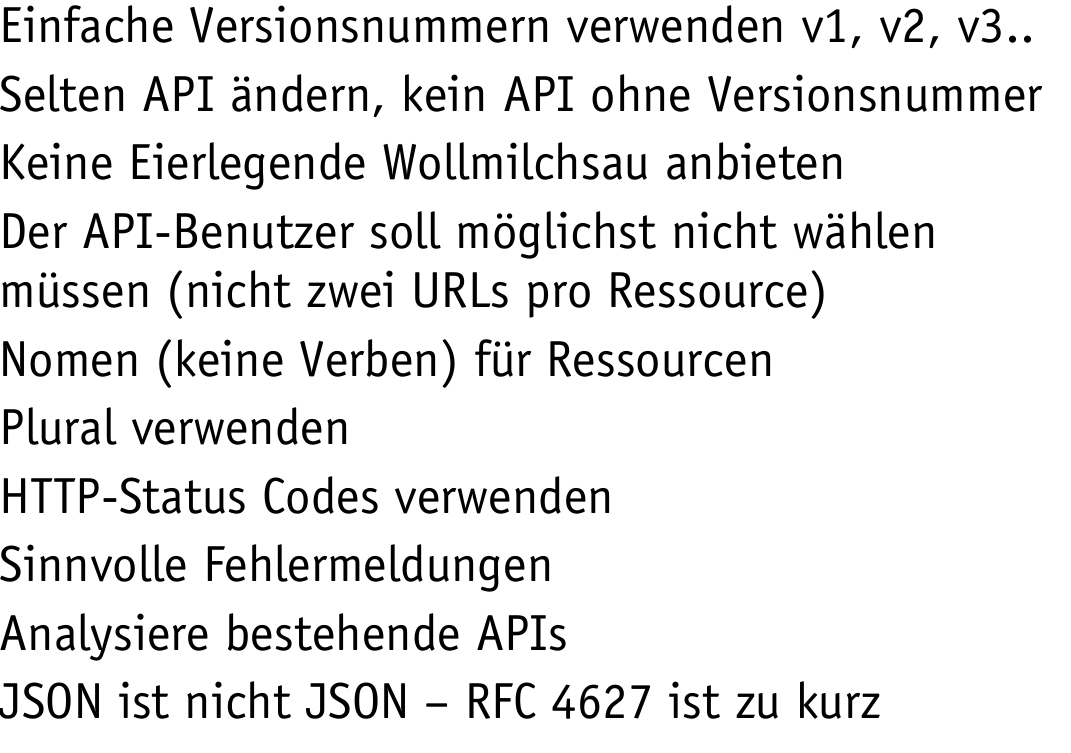
PUT: Ist idempotent, ersetzt allfällig existierende Ressource

POST: Ist nicht idempotent, Anlegen von neuen Ressourcen via Factory und untergeordnete Ressourcen hinzufügen.

DELETE: Ist idempotent (löschen ist löschen)



## Best Practice



# Zeit

BMC = Best Master Clock (Algorithm), um die beste Uhr zu finden

Grandmaster = Die beste Uhr = Zeitreferenz

PTP = Precision Time Protocol

## PTP-Ablauf

1. Slave schickt Delay-Request-Messages an Master
2. Master antwortet mit Delay-Reponse-Messages
3. Slave kann Roundtriptime berechnen.
4. Master schickt Zeitsignale an Slaves
5. Slave korrigiert Zeitsignale von Master mithilfe der berechneten Korrekturwerte.

## PTP Stratum

Gibt die maximale Abweichung zu UTC an:

Stratum 1: max. 25ns

Stratum 2: max. 100ns

Stratum 3: mehr als 100ns

Stratum 255: War nie synchronisiert

Mehr als 128ms Unterschied: Gilt als nicht synchronisiert

## NTP

Ziel ist die lokale Uhr an die **Coordinated Universal Time (UTC)** anzugleichen.

**Offset**: Differenz Serverzeit/lokale Uhr

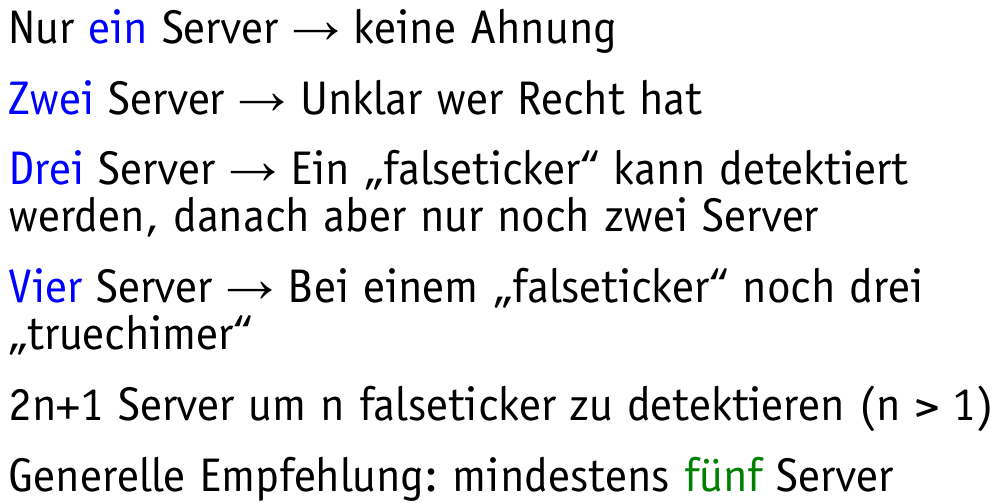
**Delay**: round-trip Server-Client

**Dispersion**: Max. Fehler lokale Uhr/Referenz

**Jitter**: Quadratisches Mittel der letzten Offsets

### Ablauf

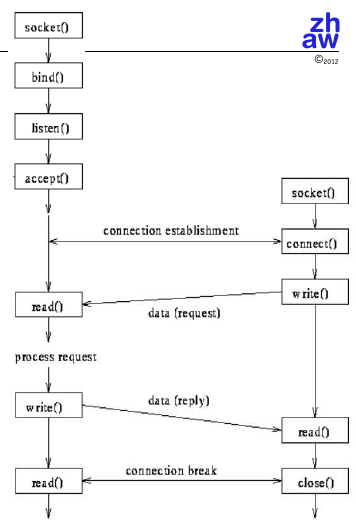
1. **Selektion**: Für alle Server Korrektheitsintervalle definieren. Die Mitte liegt beim Offset.
2. **Clustering**: Ergebnisse aus Selektion nach Qualität ordnen. Diese wird aus Distanz zum Root und Jitter berechnet. Top-Kandidat ist System-Peer.
3. **Combining**: Aus den restlichen Kandidaten werden die Systemwerte für Offset und Jitter ermittelt und die Qualität der Referenz-Uhr bestimmt.



## SNTP – Simple Network Time Pro…

* Wenn nur eine Referenz-Uhr da
* Einfache Algorithmen, Genauigkeit nicht sooo wichtig
* Einsatz bei Workstations am Ende der Stratumkette

# Sockets

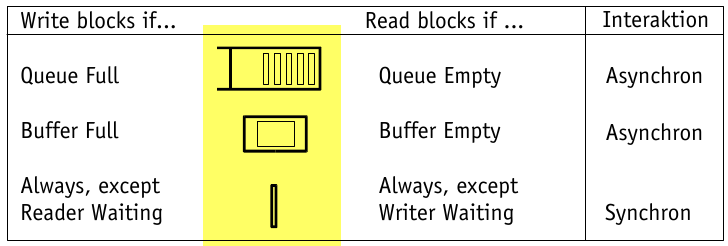
Verbindungsorientierter Dienst:  


**Funktionalität**: Zwei unabhängige, asynchron arbeitende Übertragungskanäle (in jede Richtung einen).

**Zuverlässigkeit**: kein Verlust; Reihenfolge OK.

**Leistung, Effizienz**: „Pipelining“ (Sliding Window Protocols).

**Flusskontrolle**: Keine, ist eine offene Warteschlange, die sich *unendlich* füllen lässt.



## Verbindungsaufbau

**Aktiv**, man verbindet zu einem Listener:

Socket c = new Socket(String host,

int port);

Passiv: Man wartet, bis jemand zu einem verbindet:

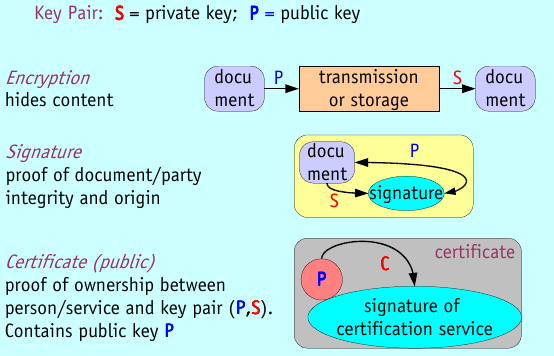
ServerSocket p = new ServerSocket(int port);

Socket s = p.accept();

## Datentransfer

* Socket-Konstruktor versucht die Verbindungsaufnahme.
* socket.getInputStream() bzw. socket.getOutputStream() liefern Anschlusspunkte für den Datenaustausch, die nach passender Veredelung sofort betriebsbereit sind.
* in/outputStream.close()/flush() um zu schliessen bzw. zu flushen
* socket.close() schliessent den Socket.

# SSL



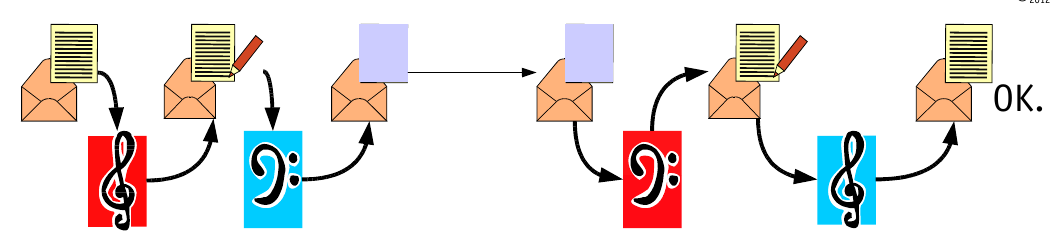
SSL kommt zwsichen Socket und TCP. Der Socket macht den Handshake und schützt die Verbindung. Rest normal.

## Private/Public-Key Cryptography

**Nachricht schicken**: Mit Public Key von Gegenüber eine Nachricht verschlüssen und diesem schicken. Nur er kann mit seinem Private Key die Nachricht entschlüsseln.

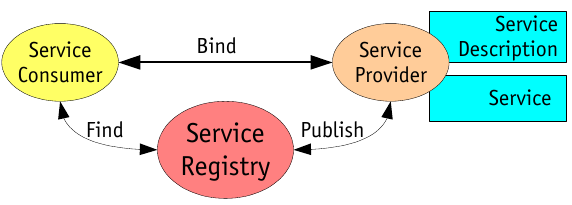
**Authentifikation**: Mit meinem Private Key etwas Hashen/Signieren und Gegenüber schicken. Er kann mit meinem Public Key die Nachricht entschlüsseln und weiss, dass ich den Private Key habe, also ich bin.

**Kombination**:



# UDDI

= Universal Description, Discovery and Integration



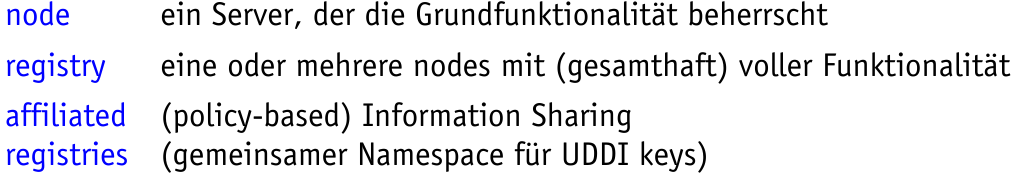
* Jemand betreibt UUID
* Service Provider können ihre Dienste bewerben
* Service Consumer finden, was sie wollen

Oft verwendet für:

* Veröffentlichung von Verbindungsinformationen, zum Finden von Web Services
* Nicht nur für Web Servcies

Oft redundant gehalten, da wichtige Infos online.

## Entitäten



## Schnittstellen

Publishing, Searching, …

## Zugriff

* Browsing
* Drill-Down: über etwas mehr Details erfahren
* Invocation: Mit Binding Templates versuchen, konkret an einen Dienst heran zu kommen (kA was das heisst…)

# Enterprise Messaging

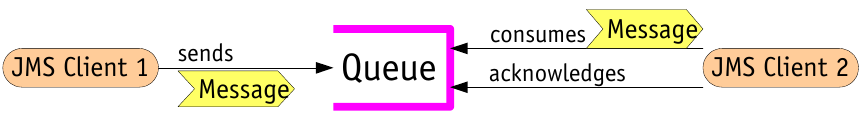
Alles async, zuverlässig, Messages = Requests/Respnses/Events, alles ziemlich anwendungsspezifisch, Client/Server-Unterschied aus Sicht des Message Services irrelevant da alles „Clients“

## JMS = Java Message Service

Administered Objects: Müssen registriert werden (bind), damit sie gefunden werden.

Der Rest wird über die Administered Objects gehandled.

### Point-to-Point Messaging



Queue ist das Ziel.

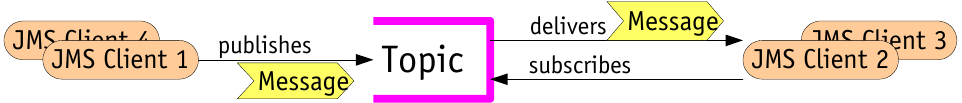
Eine Message wird höchstens von 1 Client abgeholt.

Das Abholen muss vom Empfänger initialisiert werden (oder er registriert einen Message Listener).

### Transactions

Für mehrere Messages wird Transaktionskontext festgelegt (alles oder nichts) => excatly once!

### Publish/Subscribe Messaging



Wie Mailinglist.