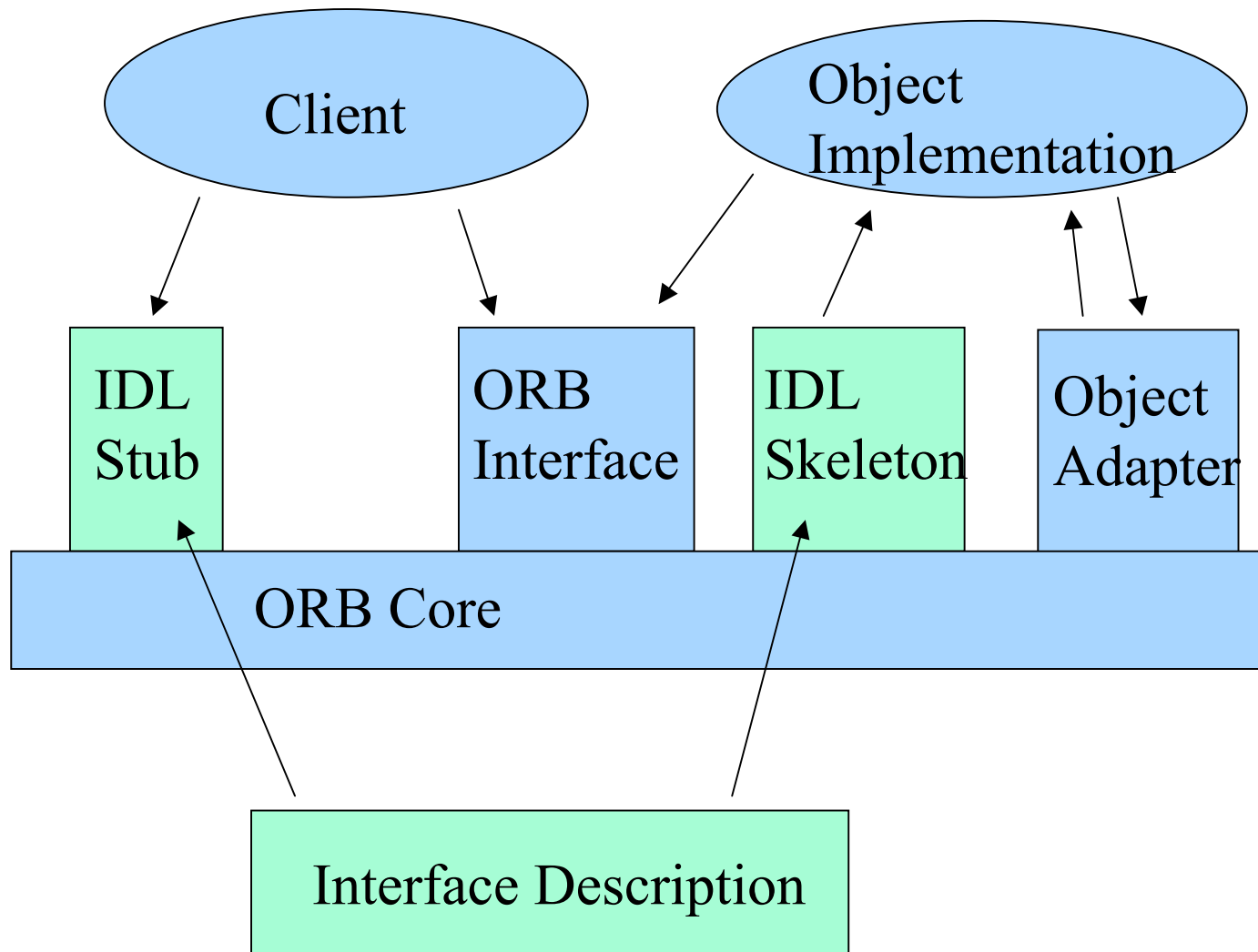




# CORBA

Static/Dynamic Invocation Interface (SII/DII),  
Interface Repository

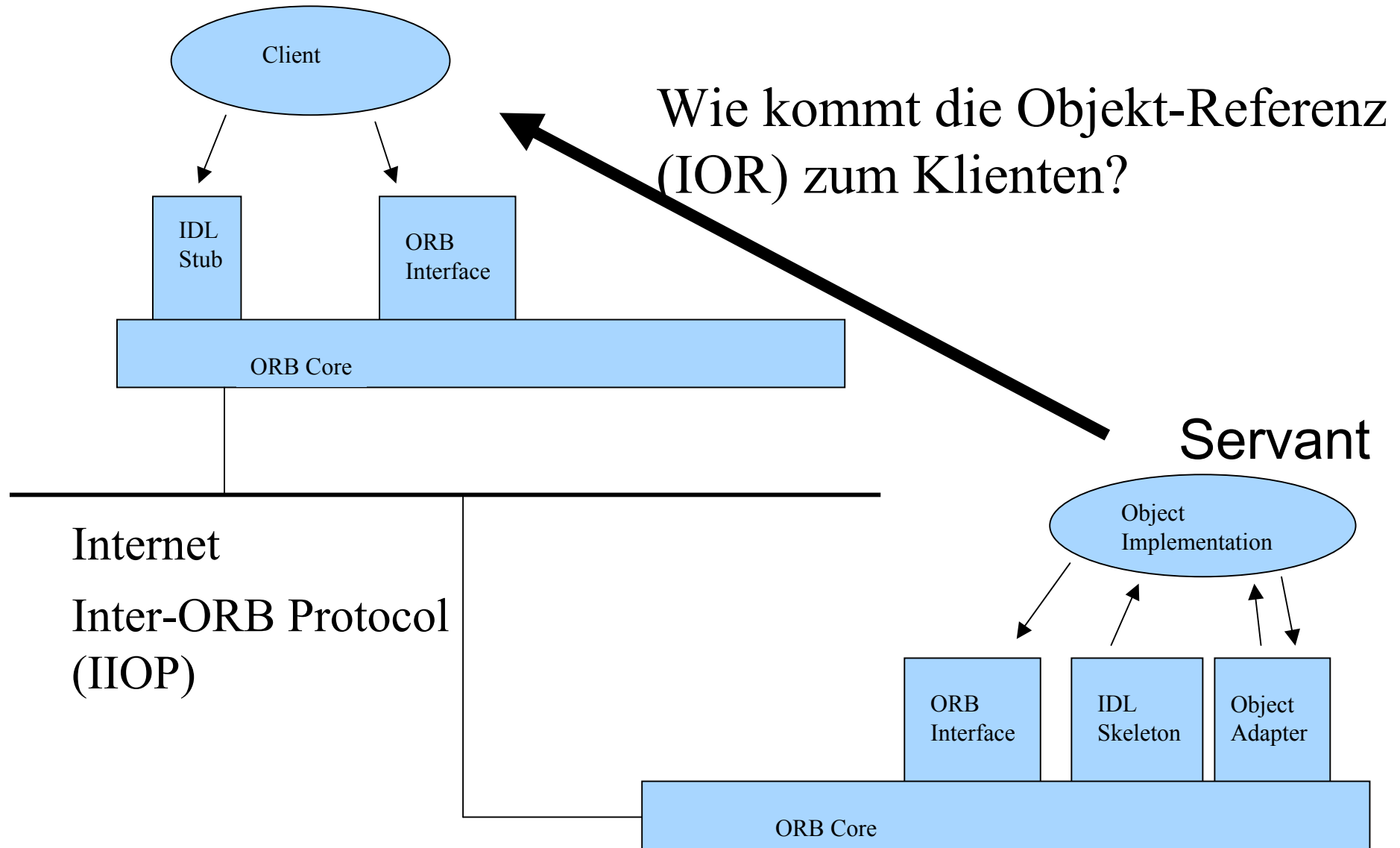
# Vereinfachtes Modell der ORB Architektur



# Verteiltes System (Beispiel x1)

- Server und Klient haben eine gemeinsame Schnittstellenbeschreibung in Form einer IDL-Datei
- Im Server-Prozess gibt es Servant-Objekte, die die gewünschte Funktionalität zur Verfügung stellen
- Objekt-Adapter verwalten die Servants (in diesem einfachen Beispiel wird das einzige Servant-Objekt vom Programmierer selbst aktiviert)
- Der Klient benötigt zum Aufruf von Methoden eines CORBA-Objektes dessen Objekt-Referenz (IOR)

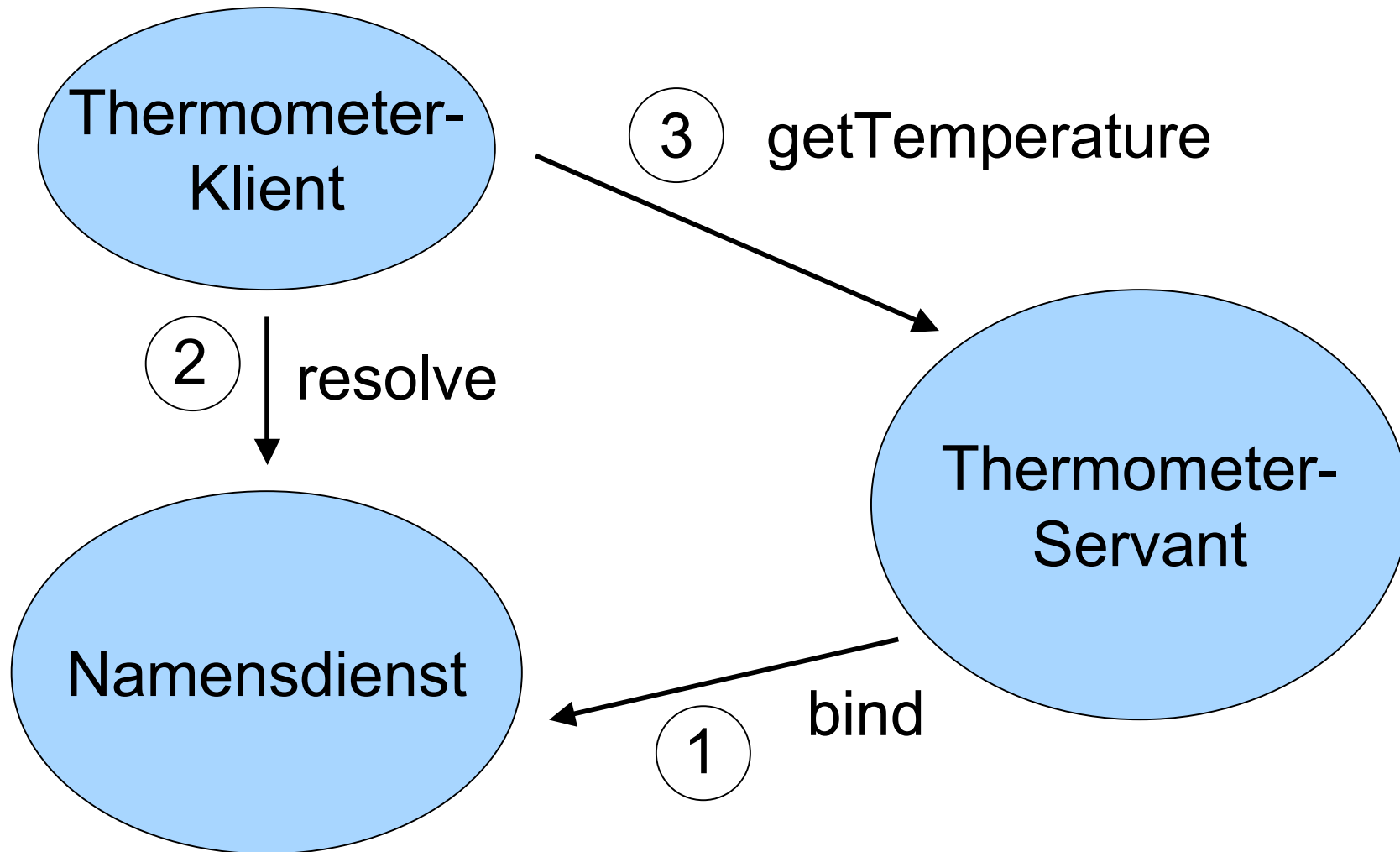
# Verteiltes System (Beispiel x1)



# Verteilung der Objekt-Referenz per Naming Service (Beispiel x2)

- unhandliche IOR-Strings werden durch kurze Namen in einem Namensdienst ersetzt
- der Namensdienst ist selbst ein CORBA-Objekt des Typs `CosNaming::NamingContext`
- Vorteil: man braucht nur noch die Objekt-Referenz des Namensdienstes an Server und Klienten zu übergeben
- Vorteil 2: im ORB sind Namensdienst-Abfragen schon eingebaut (INS):  
"corbaname::localhost:2809#Therm002"
- Java2 SDK1.4.0 beinhaltet ein Programm, das so einen Namensdienst realisiert (orbd)

# Nutzung eines Namensdienstes (Beispiel x2)



# Software-Entwicklungs-Zyklus

- zunächst wird eine Schnittstellen-Beschreibung erstellt und als IDL-Datei an die Programmierer verteilt
- danach findet kein weiterer Informationsaustausch statt, die Schnittstelle ist festgelegt
- Änderungen an der Schnittstelle sind mit hohem Aufwand verbunden, da sämtliche Klienten- und Server-Programme neu kompiliert werden müssen
- Erweiterungen der Funktionalität lassen sich in geringem Maße mit Hilfe von Vererbung realisieren

# Vererbung in IDL

```
#include "ExampleB.idl"

module ExampleC
{
    interface Thermometer : ExampleB::Thermometer
    {
        double getTemperatureInUnit
            (in ExampleB::TempUnit tUnit);
    };
};
```

- Syntax ist C++-ähnlich
- die IDL-Datei der abzuleitenden Klasse muss mit `#include` geladen werden

# Dynamisches Laden von Klassen in Java

- Klassenbeschreibungen sind Objekte (java.lang.Class)
- in Java lassen sich zur Laufzeit Klassen nachladen:

```
Class c = Class.forName("NeueKlasse");
```

- im diesem Fall muss die Datei "NeueKlasse.class" im CLASSPATH liegen
- Wie fängt man mit dieser Klasse etwas Sinnvolles an?  
Z.B. Casting:

```
Object obj = c.newInstance();
```

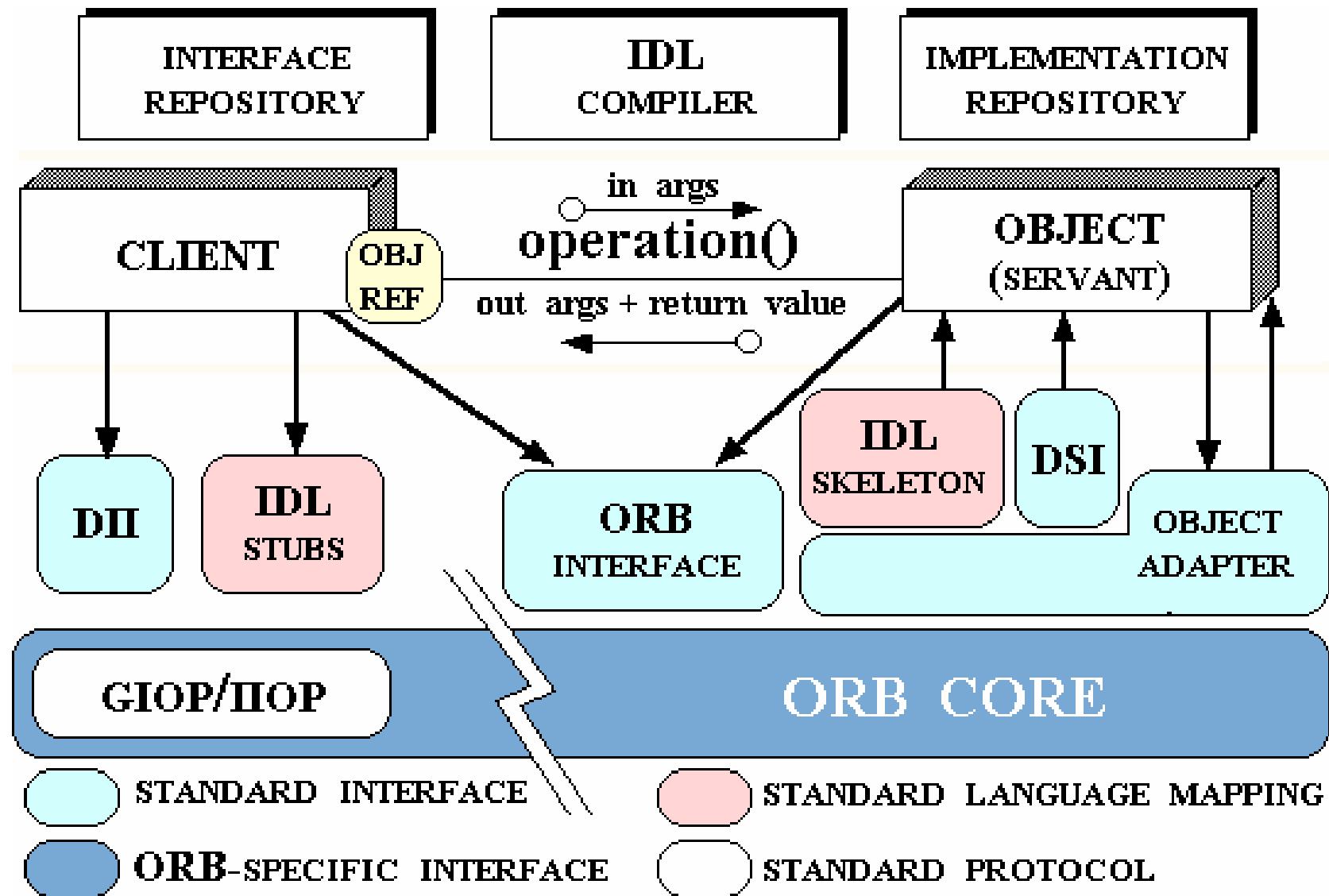
```
Thermometer t = (Thermometer) obj;
```

# Introspektion

- einfach in Java, Beispiel:

```
obj.getClass().getMethods()
```

- etwas schwieriger in CORBA:
  - Interface-Beschreibungen können in einem Interface-Repository gespeichert werden
  - ein Interface-Repository speichert den Inhalt von IDL-Dateien in maschinenlesbarer (kompilierter) Form
  - ein IR ist selbst ein CORBA-Objekt
  - Unterstützung für IR ist in vielen ORB-Implementationen bereits vorhanden



<http://www.cs.wustl.edu/~schmidt/corba-overview.html>