Netzwerktechnik und Verteile Systeme

Sebastian Simon

Teil 6:
Architekturen und Middleware

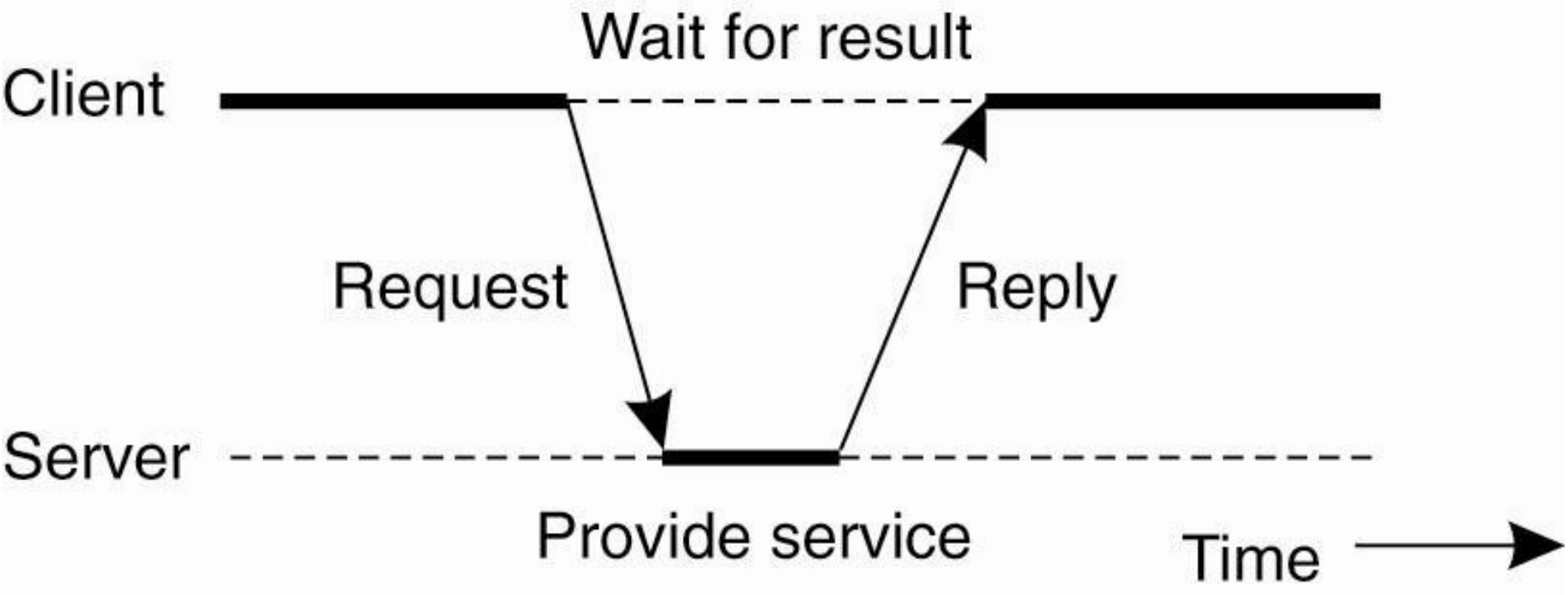
Bewältigung der Komplexität (1)

- Große Systeme haben Millionen Zeilen Code
- Es gilt den Überblick zu bewahren
  - Fehler müssen rasch befunden werden
  - Neue Features müssen laufend eingearbeitet werden
- Architektur befasst sich mit Aufteilung des Programmcodes in verschiedene Komponenten und deren Zusammenwirkung

Bewältigung der Komplexität (2)

- Seperation of Concerns
  - Einteilung in Client, Server, Service
  - Definieren von Schichten (Layering)
  - Implementierung aufteilen auf Komponenten
- Datenkapselung (Information hiding)
  - Zugriff nur über Schnittstellen
  - Direkter Zugriff auf Implementierung nicht möglich

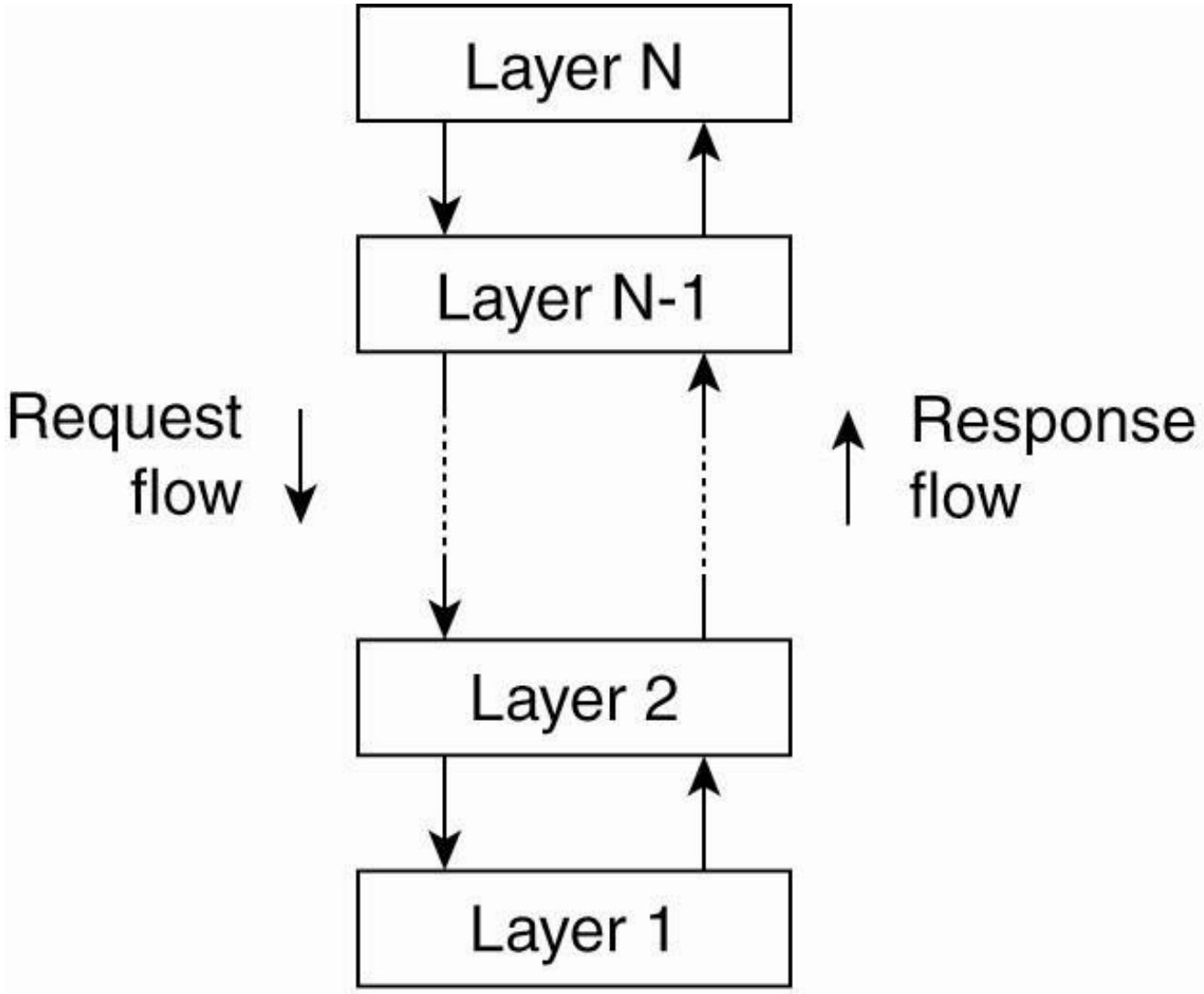
Client-Server Kommunikation (1)



Client-Server Kommunikation (2)

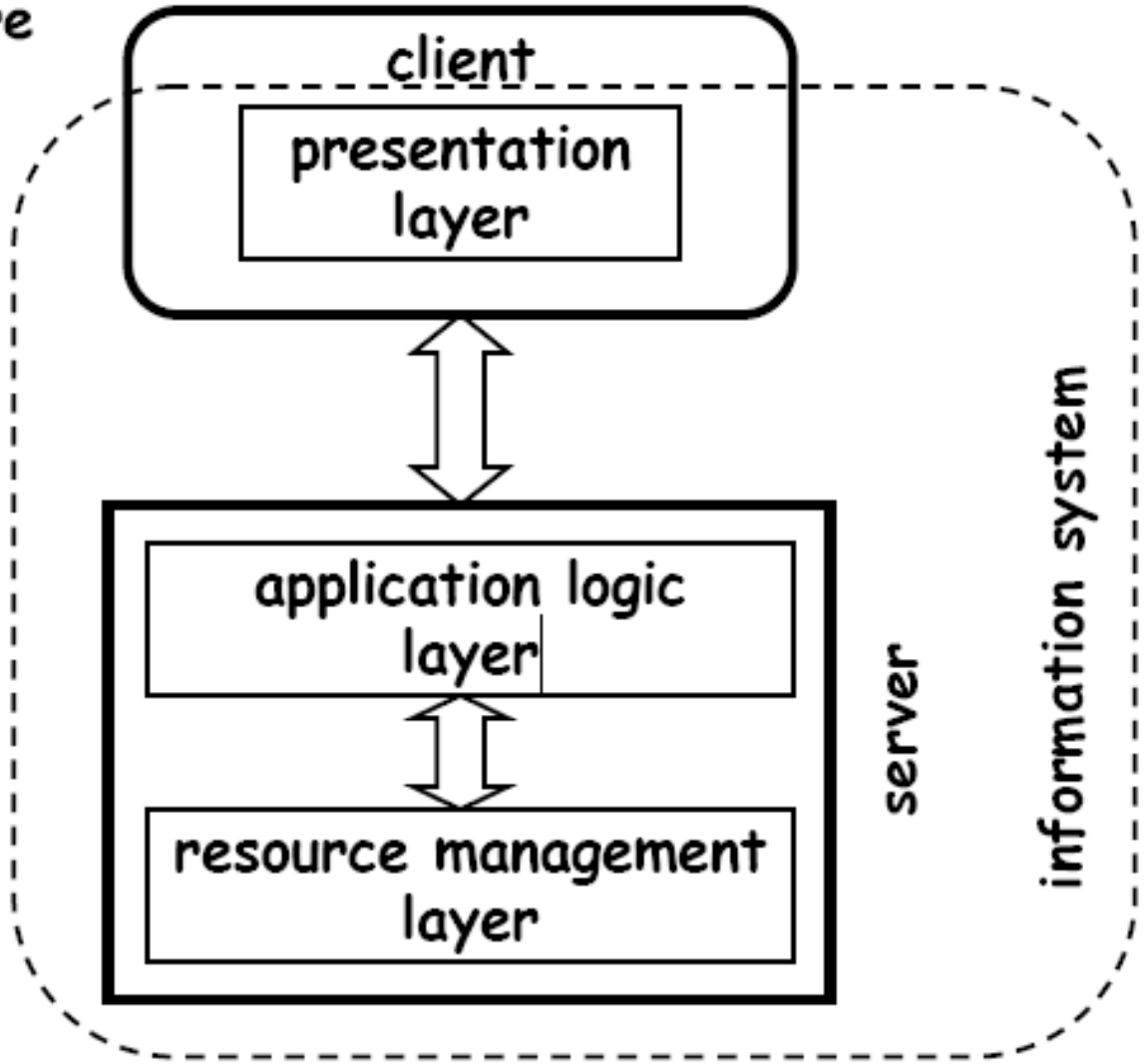
- Server wartet permanent auf Anfragen
- Client sendet Anfrage an Server und wartet
- Sobald eine Anfrage am Server eintrifft wird diese abgearbeitet und geantwortet
- Danach wartet der Server auf weitere Anfragen
- Sobald der Client die Nachricht empfängt, kann er weiterarbeiten

Schichtarchitektur



2-Schichten-Architektur

2-tier architecture



Wie Server-Call implementieren? (1)

1. Ansatz:

- Server lauscht auf TCP Port
- Client packt Parameter in einen String
- Client sendet gewünschten Funktionsnamen und Parameterstring
- Server entpackt Request und ruft gewünschte Funktion auf
- Server packt Antwort wieder in String und sendent zurück an Client
- Client entpackt Antwort und verarbeitet sie

Wie Server-Call implementieren? (2)

Verschiedene Punkte müssen dafür überlegt und implementiert werden:

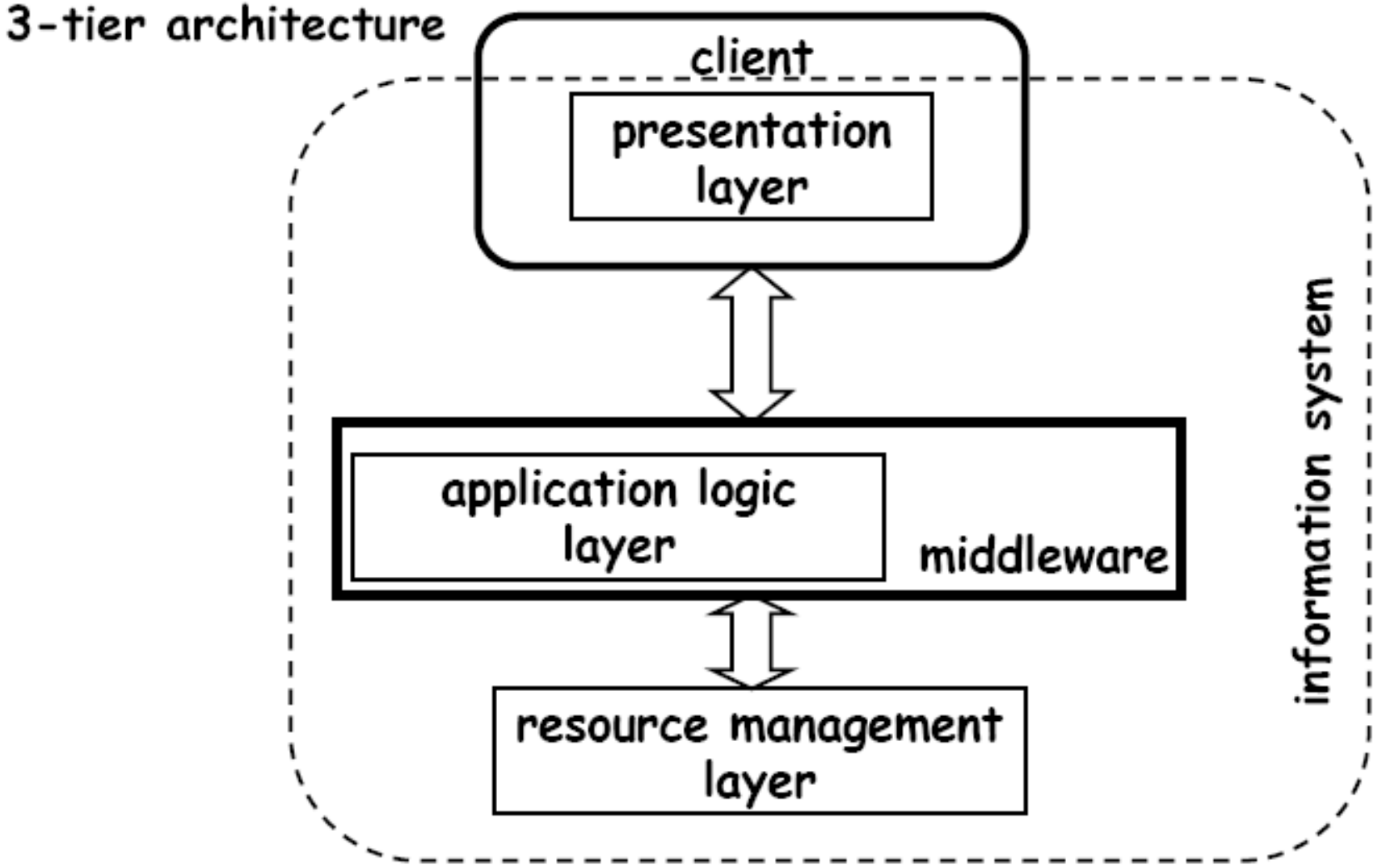
- Packen komplexer Objekte in einen String
- Exaktes Protokoll für Datenübertragen (wie beginnt/endet Aufruf, etc.)
- Unterstützung für Nebenläufigkeit
- Fehlerbehandlung
  - Server nicht erreichbar, Verbindung bricht ab, ...
  - Request vom Client fehlerhaft (zB falsche Parameter)
- Usw.

Middleware

- Typische Geschäftsanwendungen haben:
  - 70-80% Applikationsinfrastruktur die sich von Anwendung zu Anwendung kaum unterscheidet
  - 20-30% Geschäftslogik die spezifisch ist für die jeweilige Anwendung

- ►Middleware bietet Applikationsinfrastruktur für Anwendungen zur Laufzeit

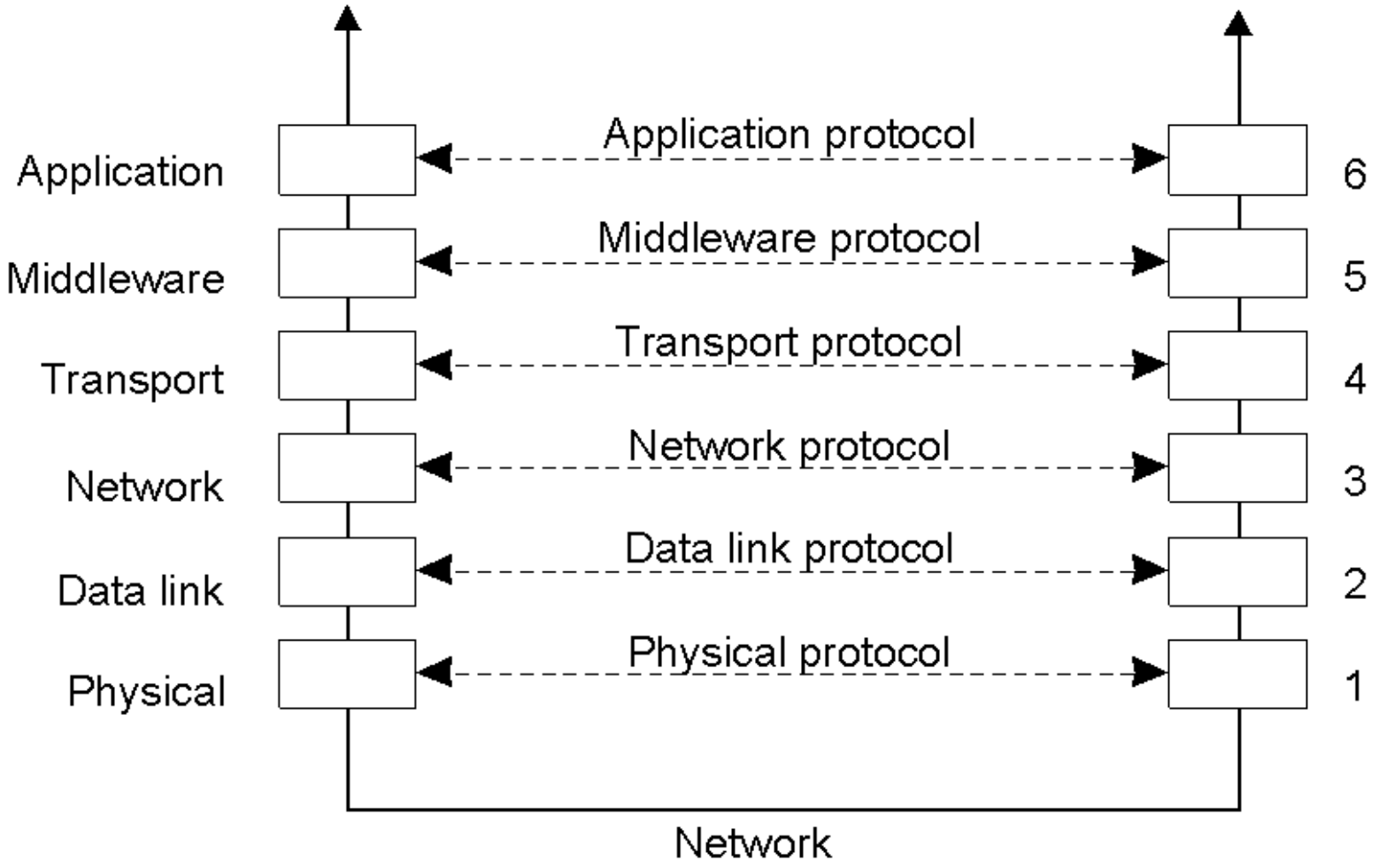
3-Schichten-Architektur mit Middlware



Middleware Services

- Erstellen von Schnittstellen (►Openess!)
- Bietet einfache Kommunikationsmöglichkeiten
  - zB Remote Procedure Call
- Unterstützung für Nebenläufigkeit
- Erleichtert Skalierbarkeit
- Services für Datenbankzugriff
- Unterstützung für Replikation
- Naming Services zum Auffinden der Server
- Security: Authenifizierung, Berechtigungen, …

Middleware Protokolle



Ein adaptiertes Referenzmodell für Netzwerkkommunikation

Remote Procedure Call (RPC)

- Ein normaler Funktionsaufruf:
  - String text = read("some_textfile.txt");
- Ziel des RPC ist es, eine Remote-Call aussehen zu lassen wie einen lokalen Aufruf
  - ►Transparenz!

Java Remote Method Invocation

- Java RMI ist die Defaultimplementierung eines Remote Procedure Calls in Java
- Beispiel - Aufruf der Servermethode “read” auf dem Interface FileReader von Server “Server1”:

```
try {
        Registry registry = LocateRegistry.getRegistry(“Server1”);
        FileReader stub = (FileReader) registry.lookup("FileReader");
        String response = stub.read("some_textfile.txt");
        System.out.println("File content: " + response);
    } catch (Exception e) {
        System.err.println("Client exception: " + e.toString());
    }
```