Technische Universität Braunschweig

Institut für Flugführung

Netzwerkkommunikation

Computerübergreifende Kommunikation im lokalen Netzwerk

Prof. Dr.-Ing. Peter Hecker, Dipl.-Ing. Paul Frost, 20. Juni 2017

Agenda

04. April	Kick-Off
11. April	Projektmanagement
18. April	Prozessmodelle
25. April	Versionsverwaltung
02. Mai	Einführung Arduino/Funduino
09. Mai	Entwicklungsumgebungen und Debugging
16. Mai	Dokumentation und Testing
23. Mai	Dateieingabe und -ausgabe
30. Mai	GUI-Erstellung mit Qt
06. Juni	Exkursionswoche
13. Juni	Bibliotheken
20. Juni	**Netzwerkkommunikation**
27. Juni	Projektarbeit + Pflichtenheft
04. Juli	Projektarbeit
11. Juli	Vorbereitung der Abgabe

Teil I

Wiederholung

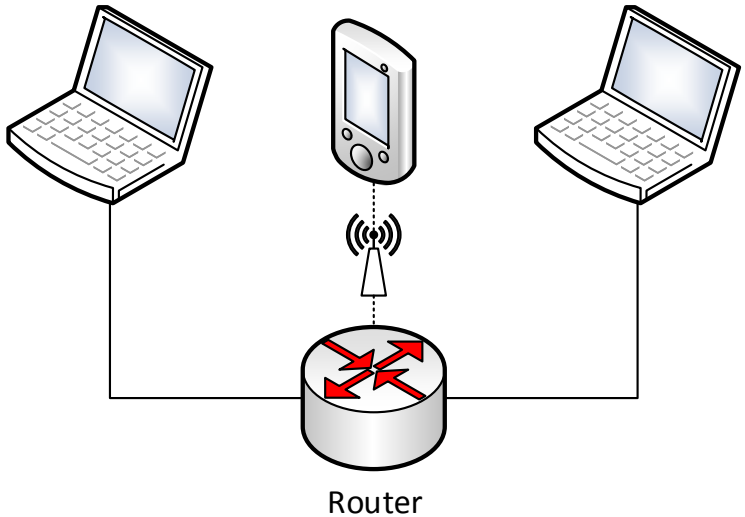
Teil II

Netzwerkkommunikation

Motivation

Geräteübergreifender Austausch von Informationen

- Abruf von HTML-Seiten
- Versand von Nachrichten
- Dateiaustausch
- Versand von Druckaufträgen
- Gerätesteuerung
- Spiele



Begrifflichkeiten

MAC-Adresse Media Access Control
Kennung der Netzwerkkarte

IP-Adresse Internet Protokoll-Adresse
Adresse im jeweiligen Netzwerk

Port Portnummer
Nummer für die Zuordnung von Paketen oder Protokollen innerhalb einer IP-Adresse

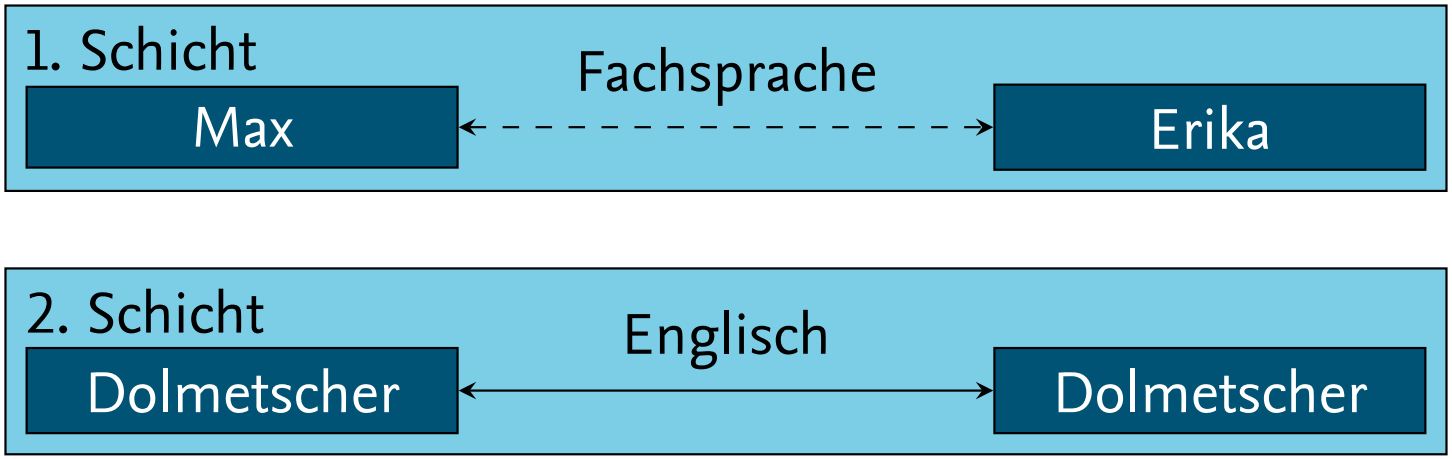
Socket Steckdose
Software-Schnittstelle für das Versenden von Informationen im Netzwerk

Server Abgeleitet von Diener
Stellt Informationen oder Leistungen zur Verfügung

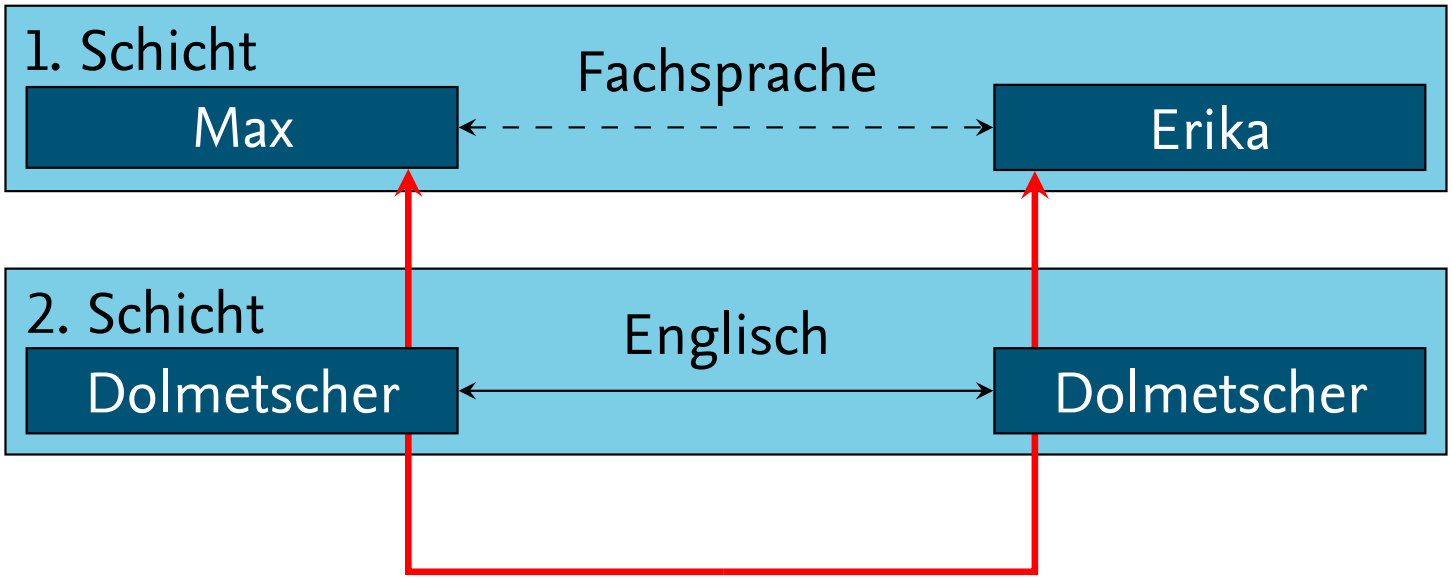
Client Abgeleitet von Kunde
Nutzt Informationen oder Leistungen eines Servers.

Schichtenmodell

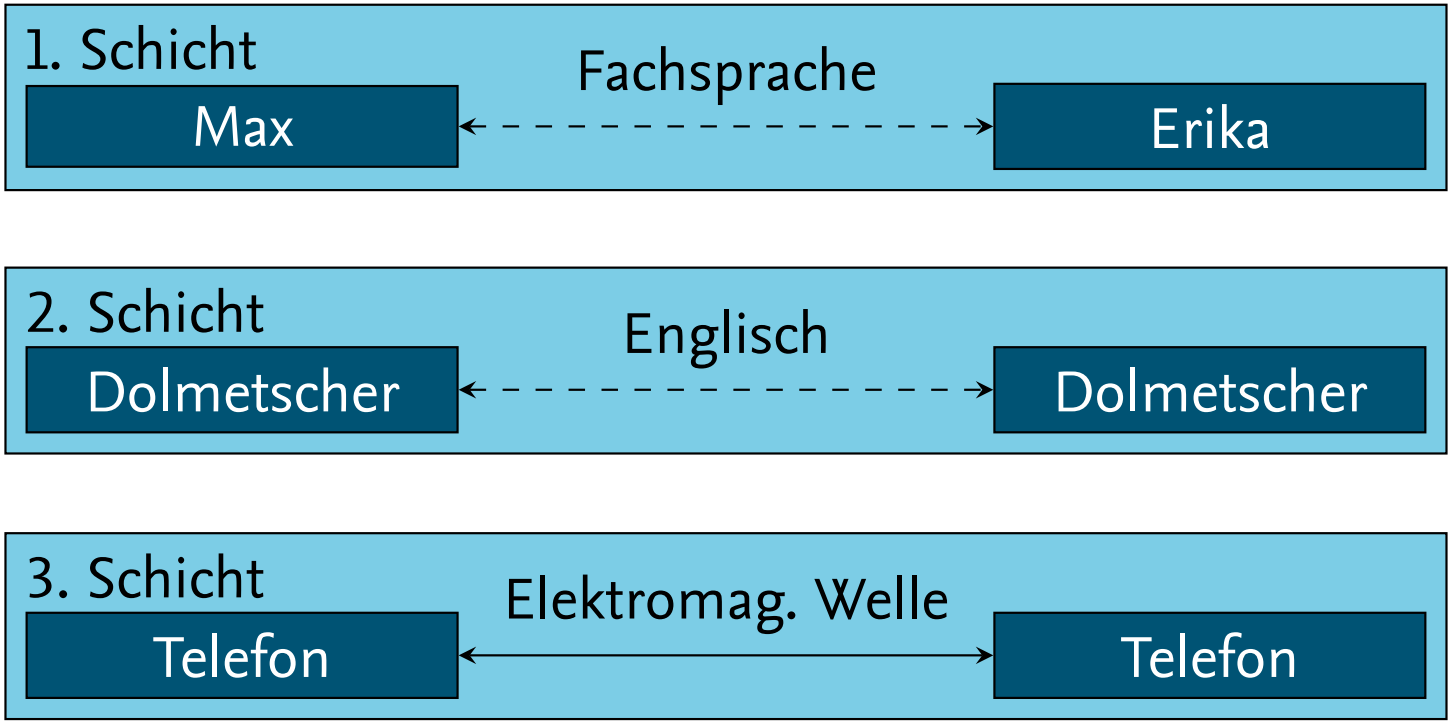
Schichtenmodell



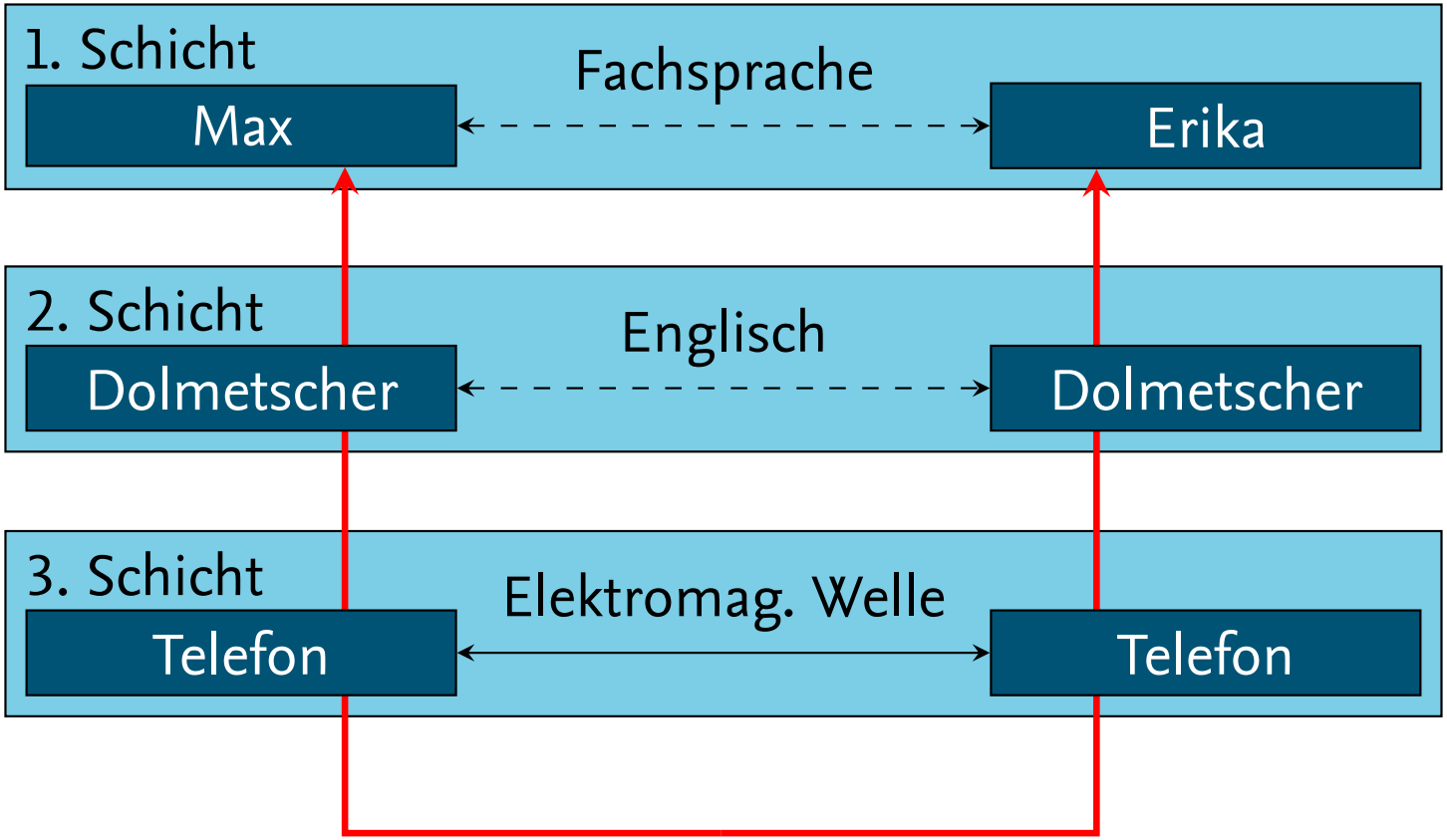
Schichtenmodell



Schichtenmodell



Schichtenmodell



Open Systems Interconnection (OSI)-Referenzmodell

PC 1

Anwendung

Open Systems Interconnection (OSI)-Referenzmodell

PC 1

Anwendung

Darstellung

Open Systems Interconnection (OSI)-Referenzmodell

PC 1

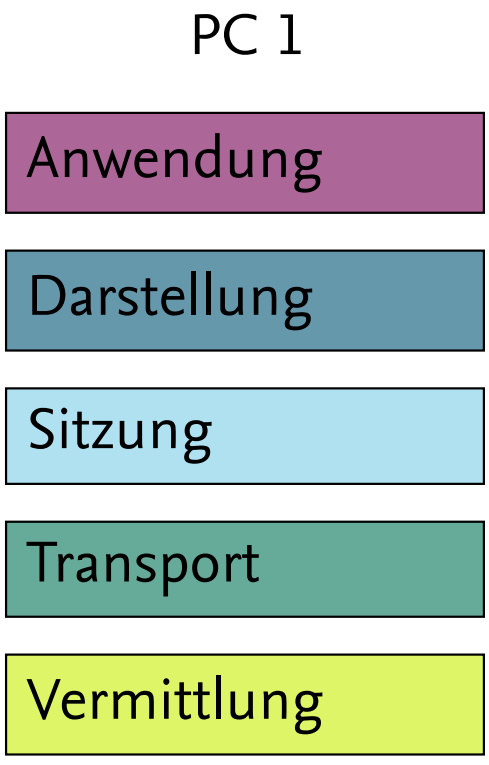
Anwendung

Darstellung

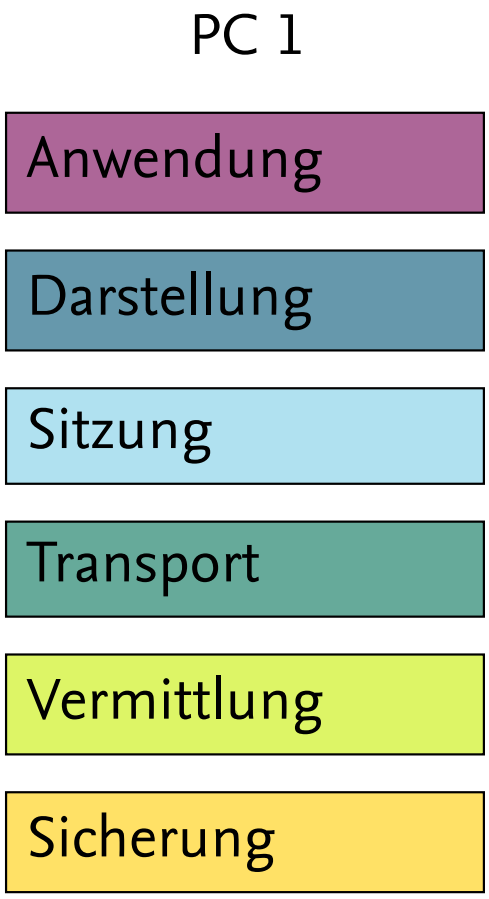
Sitzung

Open Systems Interconnection (OSI)-Referenzmodell

Open Systems Interconnection (OSI)-Referenzmodell

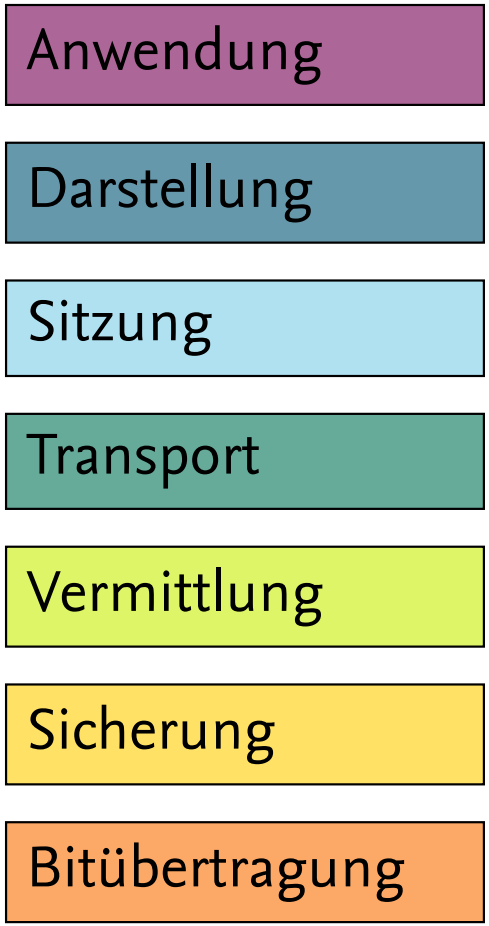


Open Systems Interconnection (OSI)-Referenzmodell

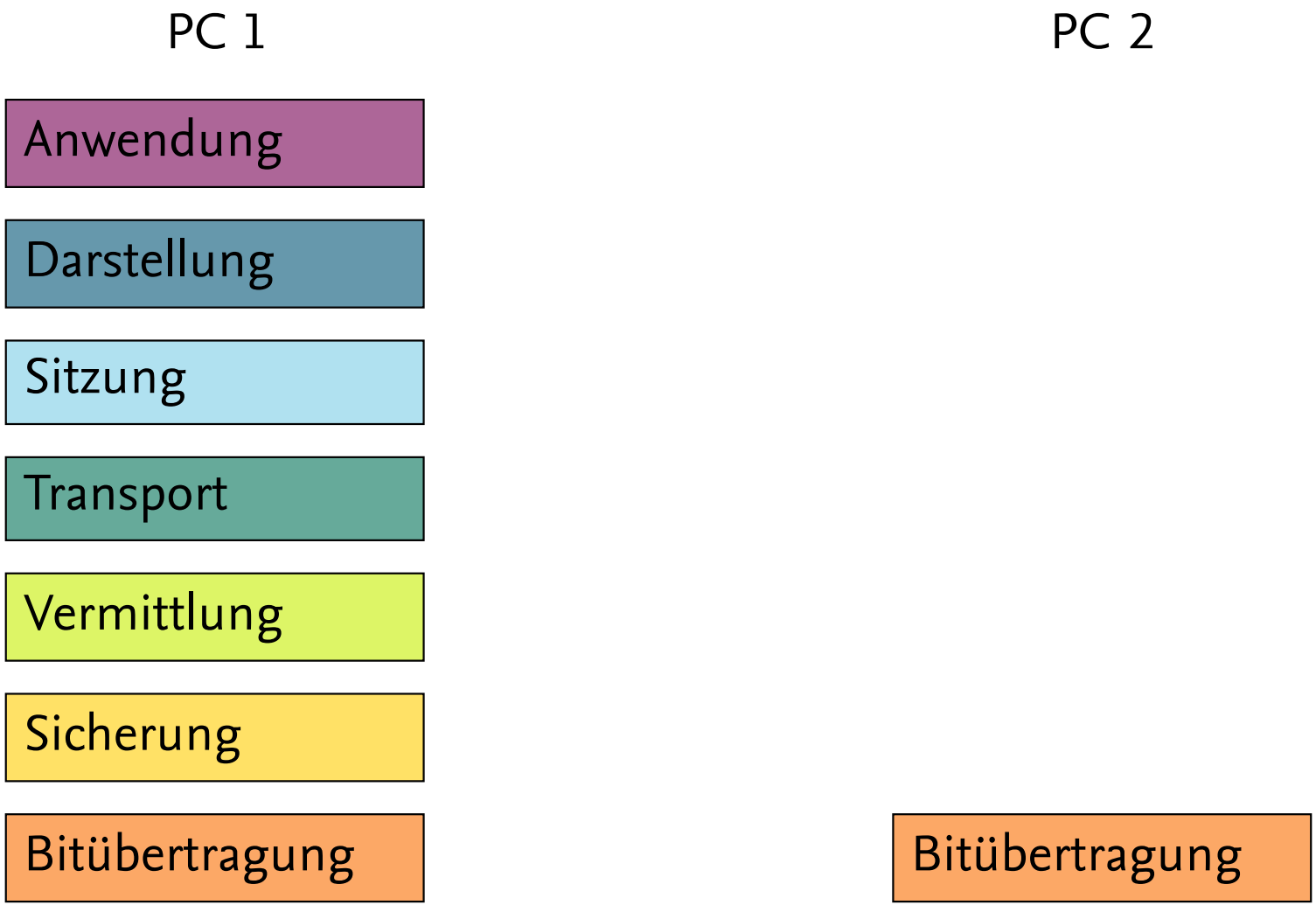


Open Systems Interconnection (OSI)-Referenzmodell

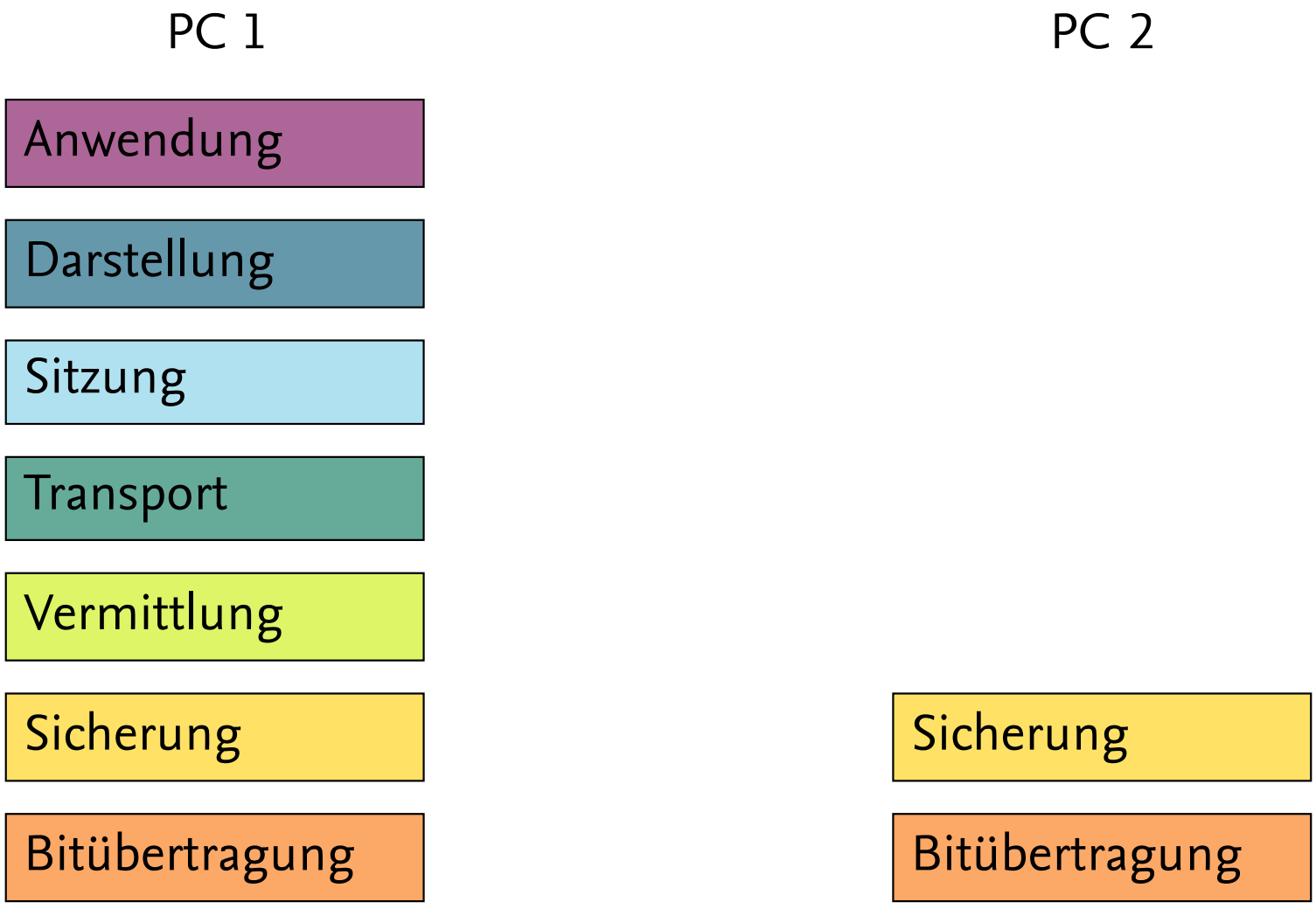
PC 1



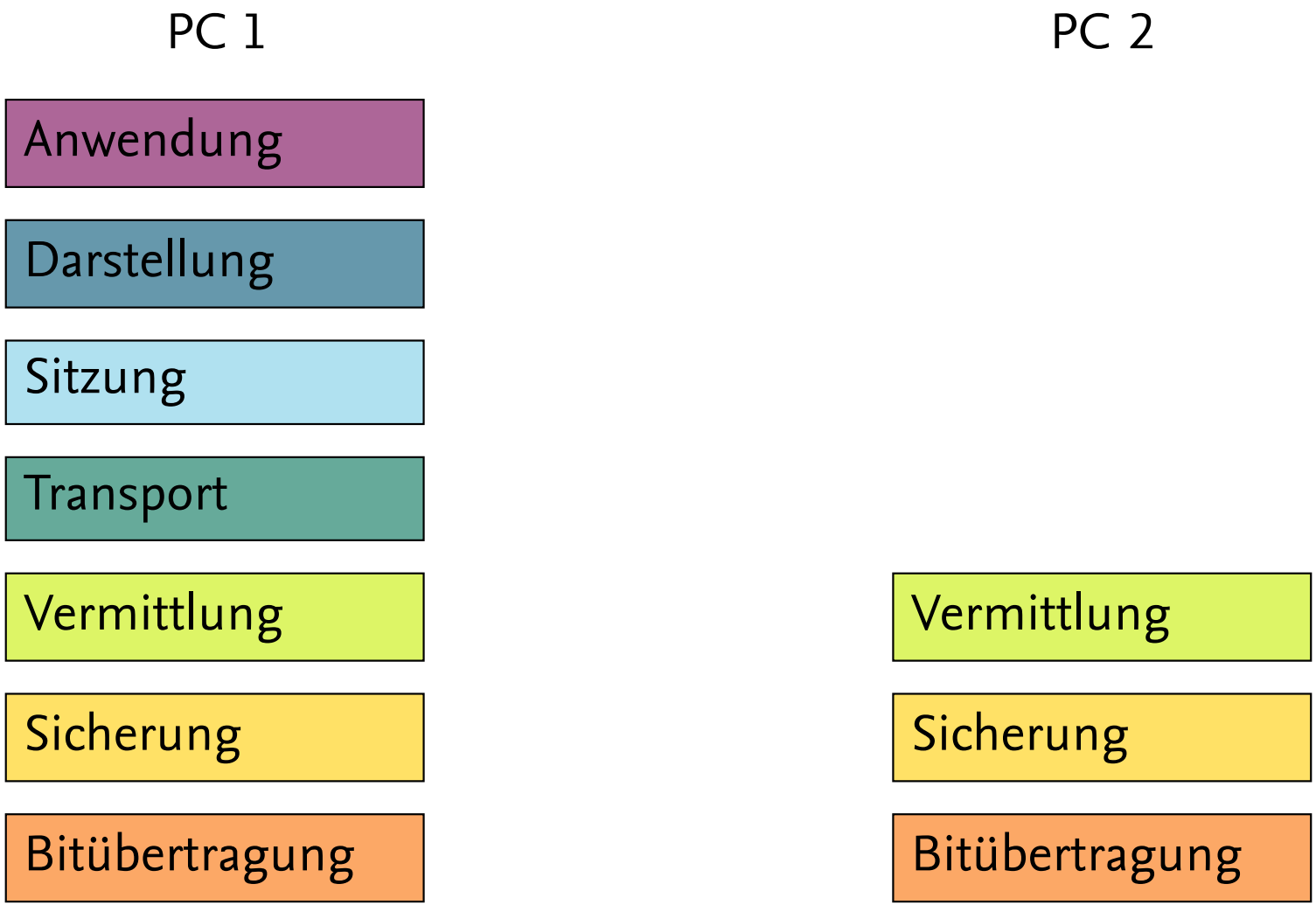
Open Systems Interconnection (OSI)-Referenzmodell



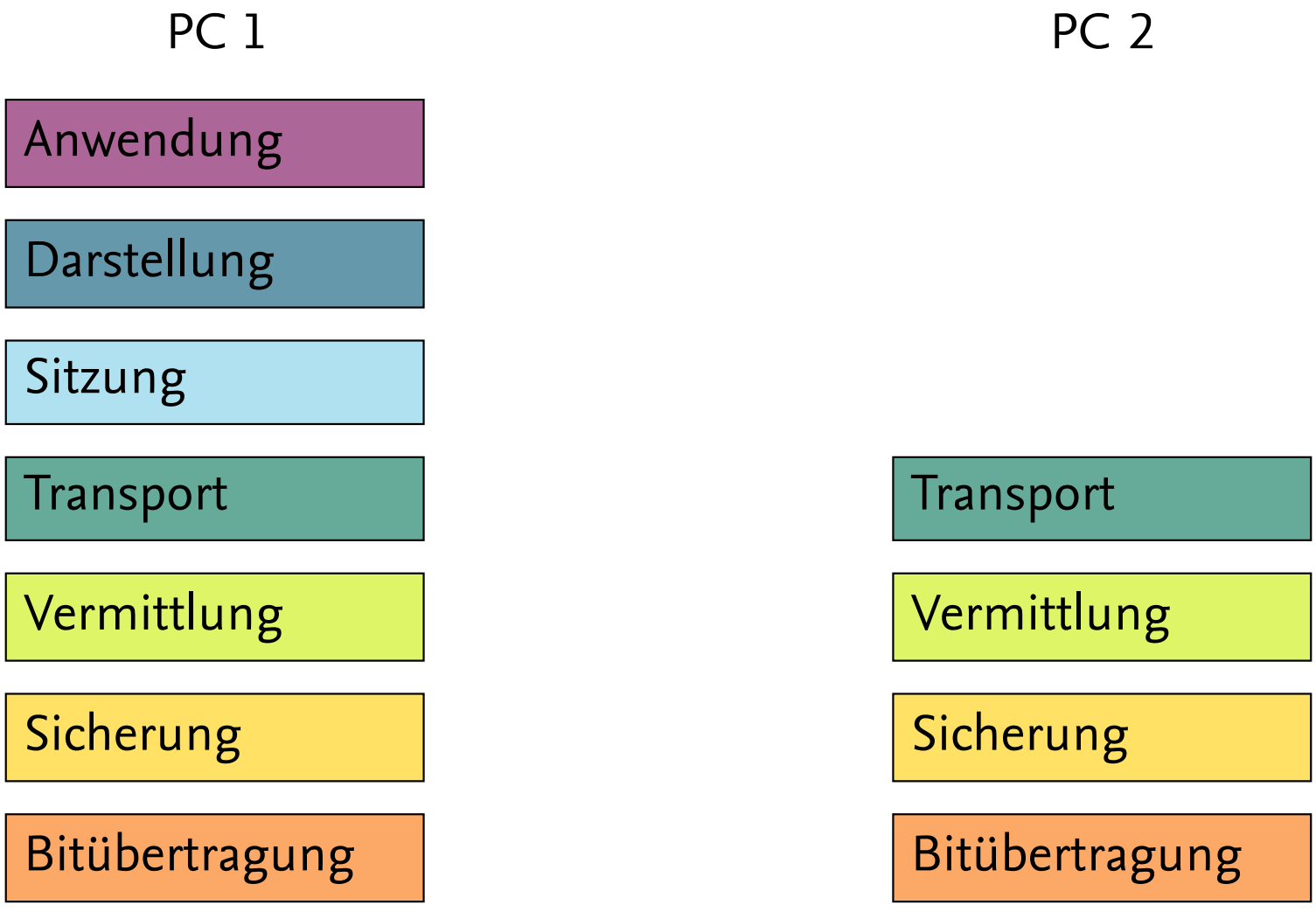
Open Systems Interconnection (OSI)-Referenzmodell



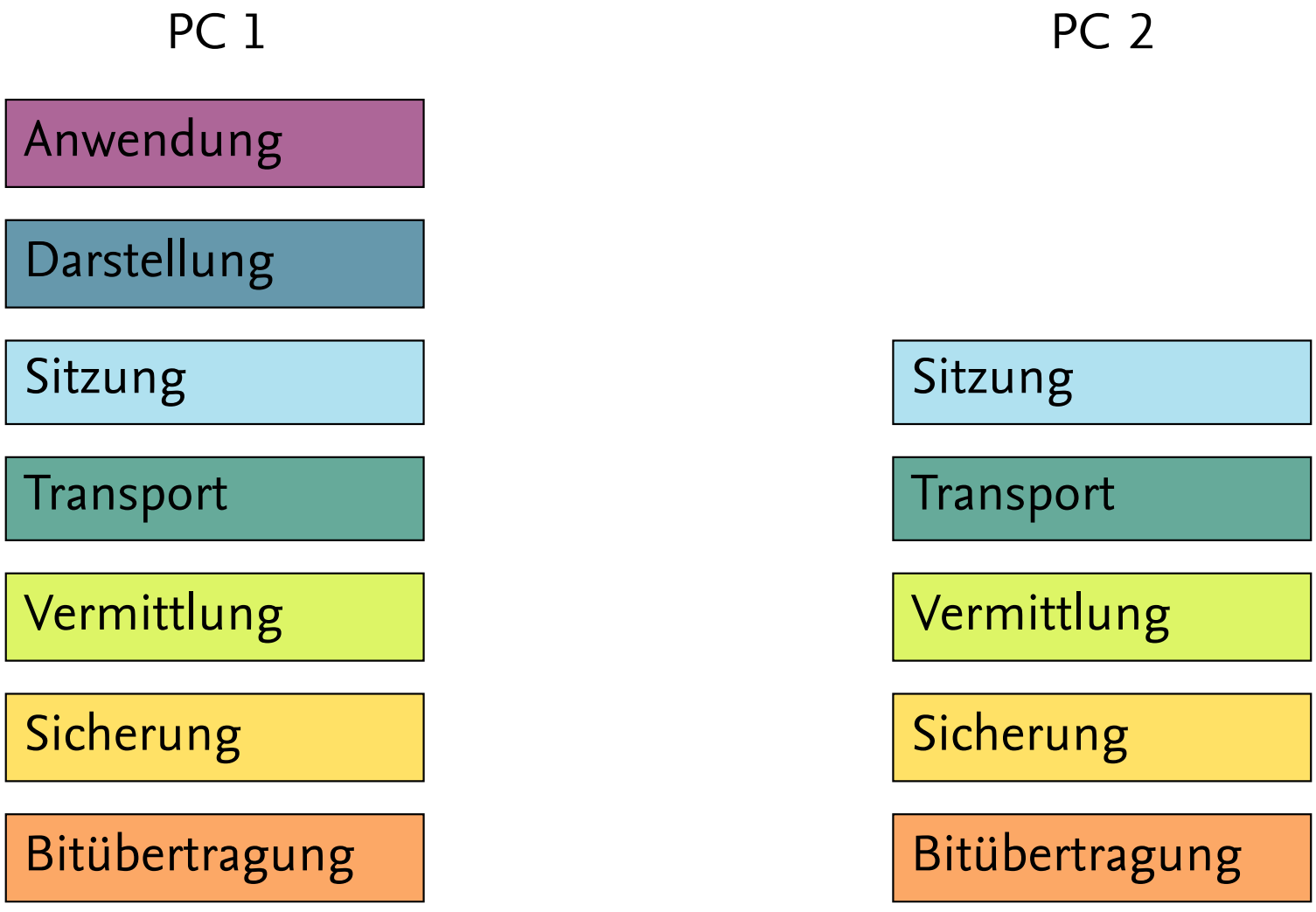
Open Systems Interconnection (OSI)-Referenzmodell



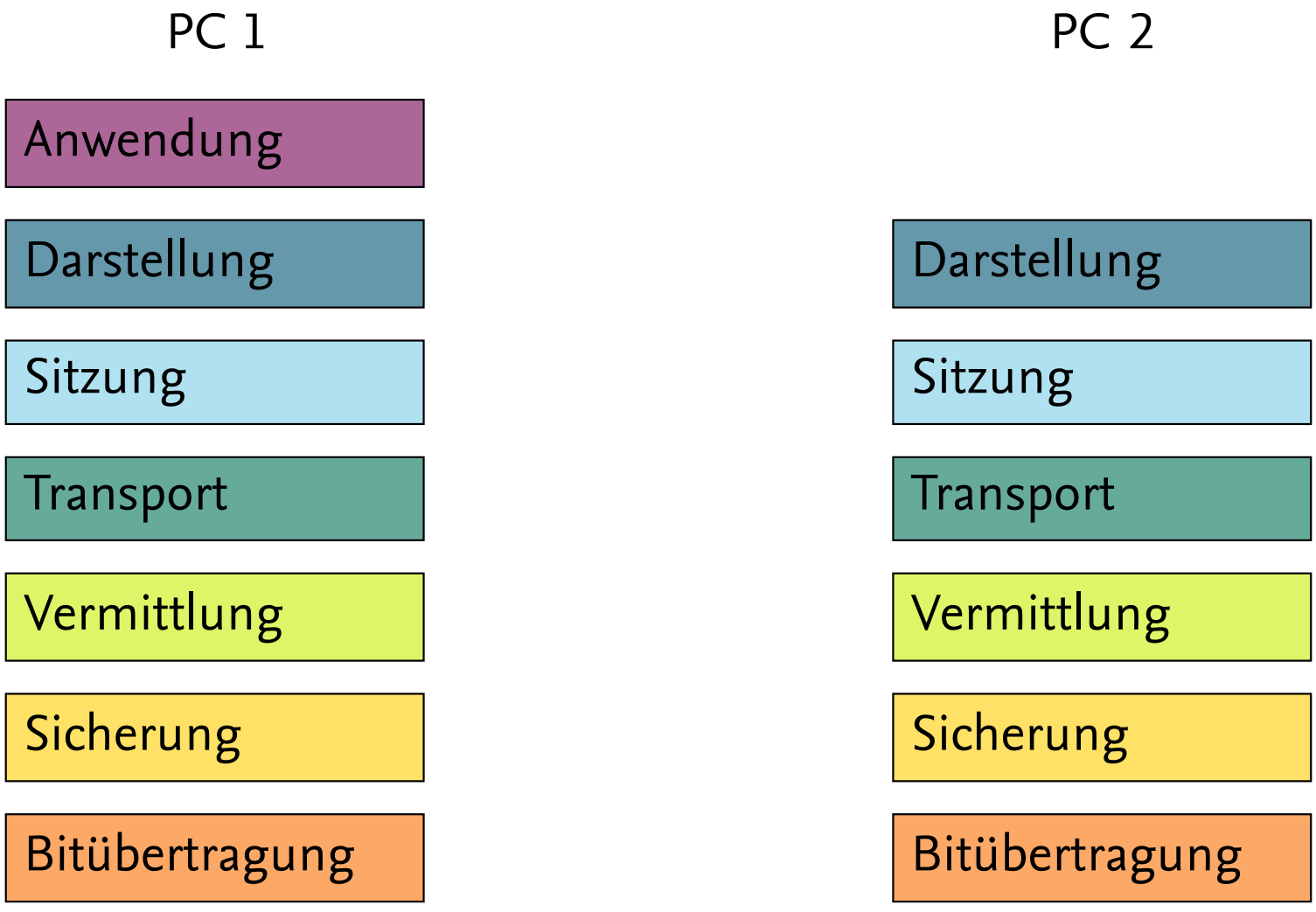
Open Systems Interconnection (OSI)-Referenzmodell



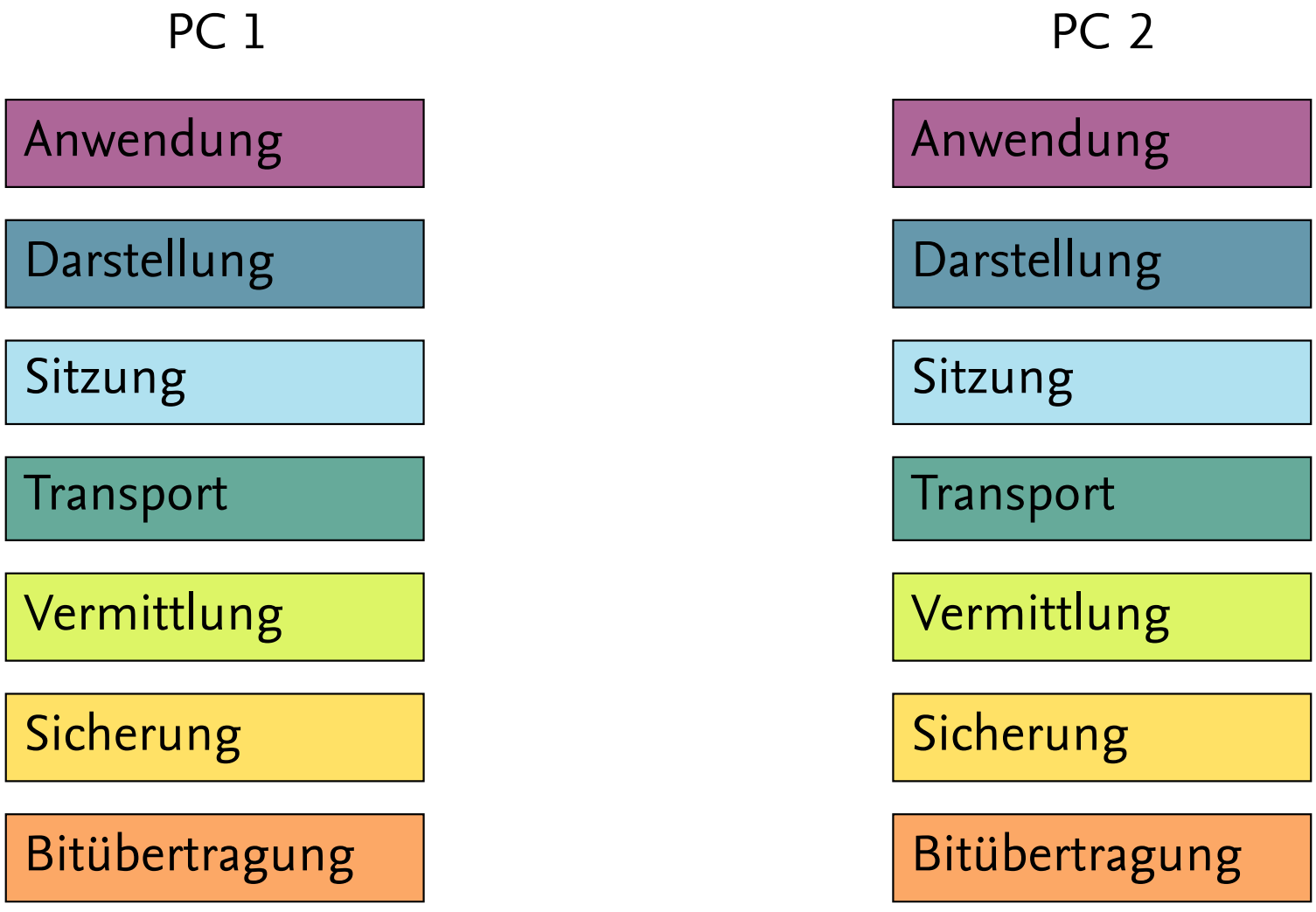
Open Systems Interconnection (OSI)-Referenzmodell



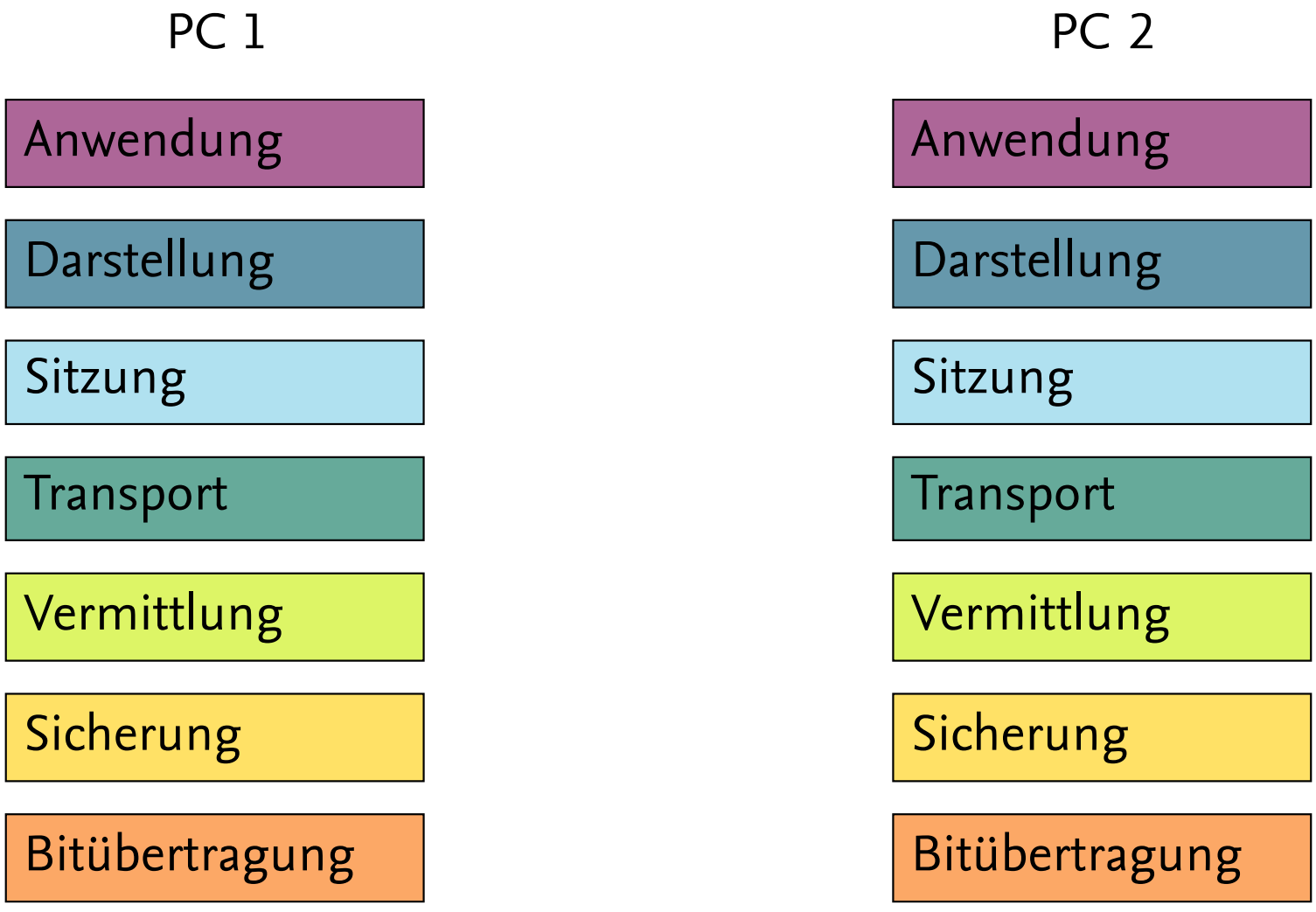
Open Systems Interconnection (OSI)-Referenzmodell



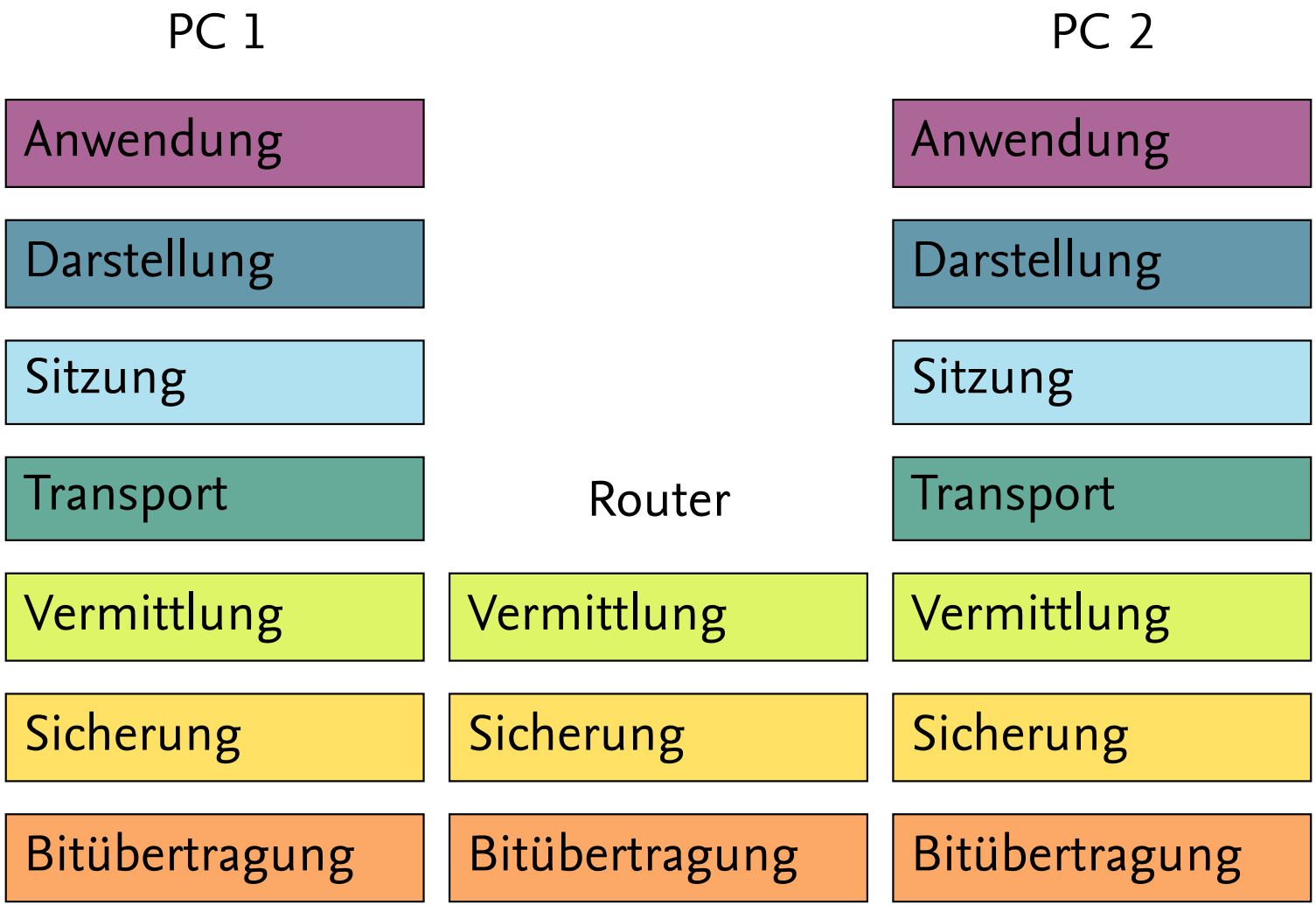
Open Systems Interconnection (OSI)-Referenzmodell



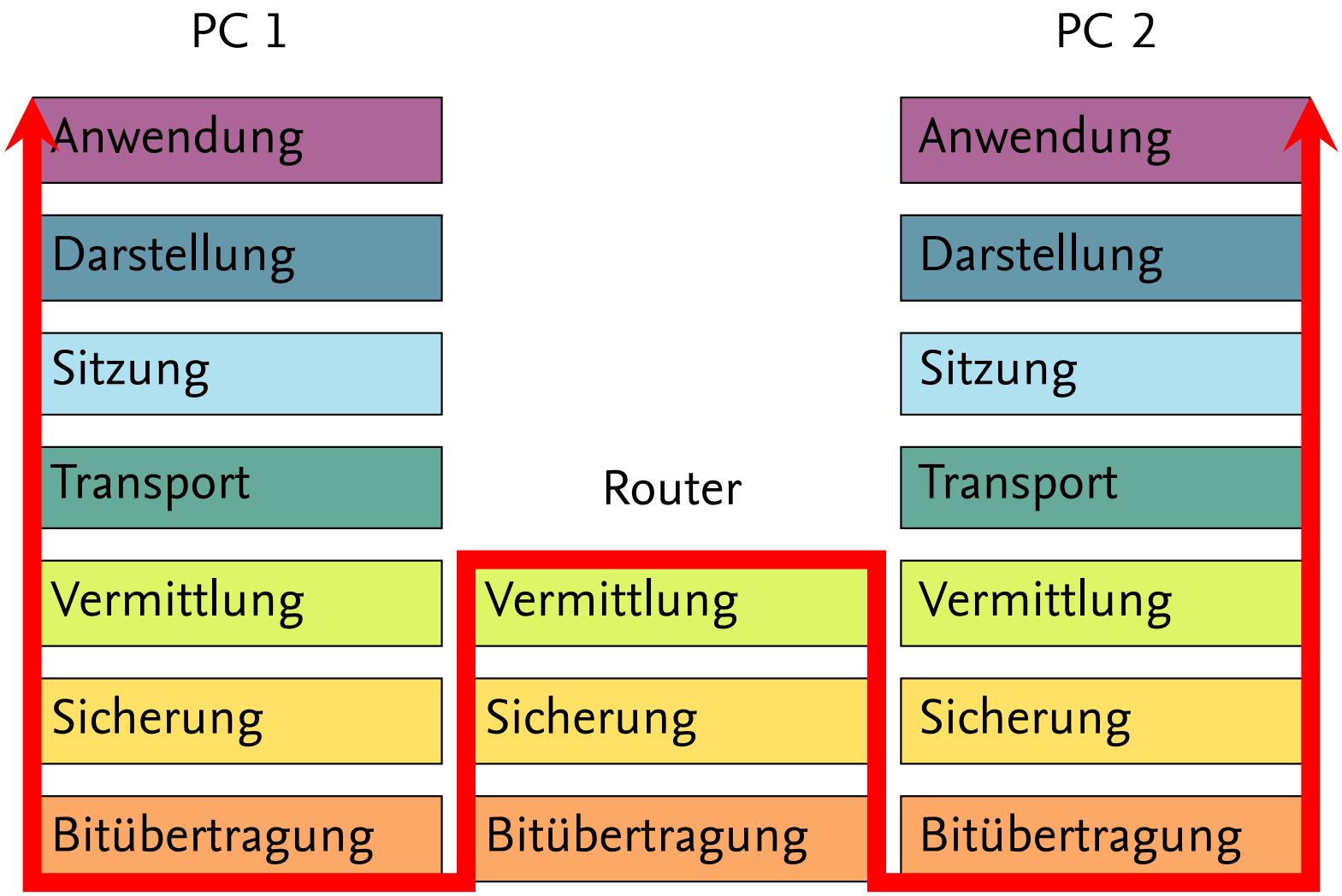
Open Systems Interconnection (OSI)-Referenzmodell



Open Systems Interconnection (OSI)-Referenzmodell



Open Systems Interconnection (OSI)-Referenzmodell



Open Systems Interconnection (OSI)-Referenzmodell

Schichten des OSI-Referenzmodells

7	Anwendung	Anwendungsorientiert	HTTP HTTPS FTP
6	Darstellung		
5	Sitzung		
4	Transport	Transportorientiert	TCP, UDP
3	Vermittlung		IP, IPX
2	Sicherung		Ethernet
1	Bitübertragung		Token Ring

[nach Haupt, M. (2008). *Grundlagen der automatischen Informationsverarbeitung für den Maschinenbau*]

TCP/UDP

TCP

- Verbindungsorientiert
- Reihenfolge der Nachrichten bleibt erhalten
- Erhöhter Aufwand durch die Sortierung und Sicherstellung der Verbindung

UDP

- Verbindungslos
- Reihenfolge wird nicht zwingend eingehalten
- Geringerer Aufwand, da die Daten verschickt werden und keine Sortierung oder Prüfung stattfindet

Verwendung einer TCP-Verbindung

Das TCP-Protokoll wird verwendet, wenn…

… sicher gestellt sein soll, dass jedes Paket ankommt, bzw. eine Nichtzustellung erkannt werden soll.

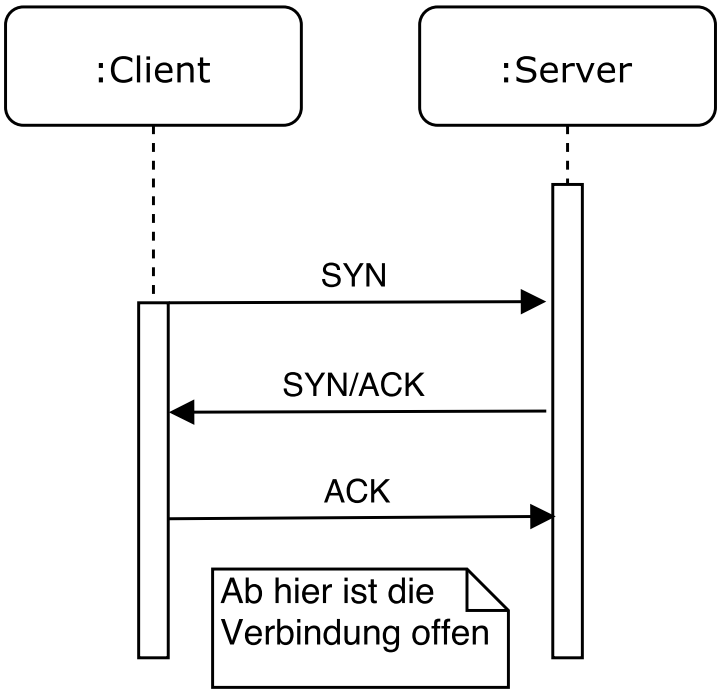
… die Reihenfolge der Datenpakete relevant ist.

… der Verbindungsstatus bekannt sein soll.

Beispiele

- Webseiten
- E-Mails
- SSH (Gesicherte Fernsteuerung eines Rechners)

TCP-Handshake



Verwendung einer UDP-Verbindung

Das UDP-Protokoll wird verwendet, wenn…

… Daten möglichst schnell übertragen werden sollen.

… nicht alle Datenpakete ankommen müssen.

… eine Nachricht an mehrere Empfänger versendet werden soll (Broad- oder Multicast).

Beispiele

- Videostream
- VoIP
- Simulationsschnittstellen

Verwendete QT-Netzwerk-Klassen

UDP

`QUDPSocket` Schnittstelle für die Nutzung des UDP-Protokolls

TCP

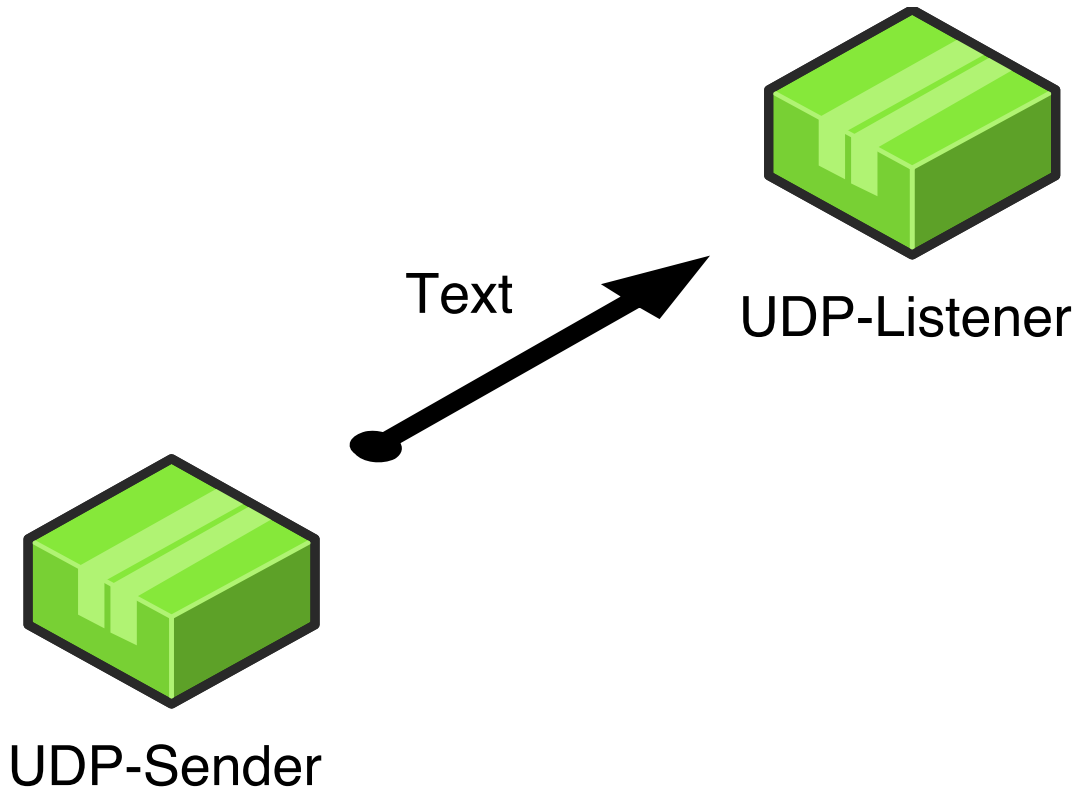
`QTCPServer` Wickelt Anfragen von Clients ab und liefert Sockets zu den jeweiligen Clients

`QTCPSocket` Schnittstelle für die Nutzung des TCP-Protokolls

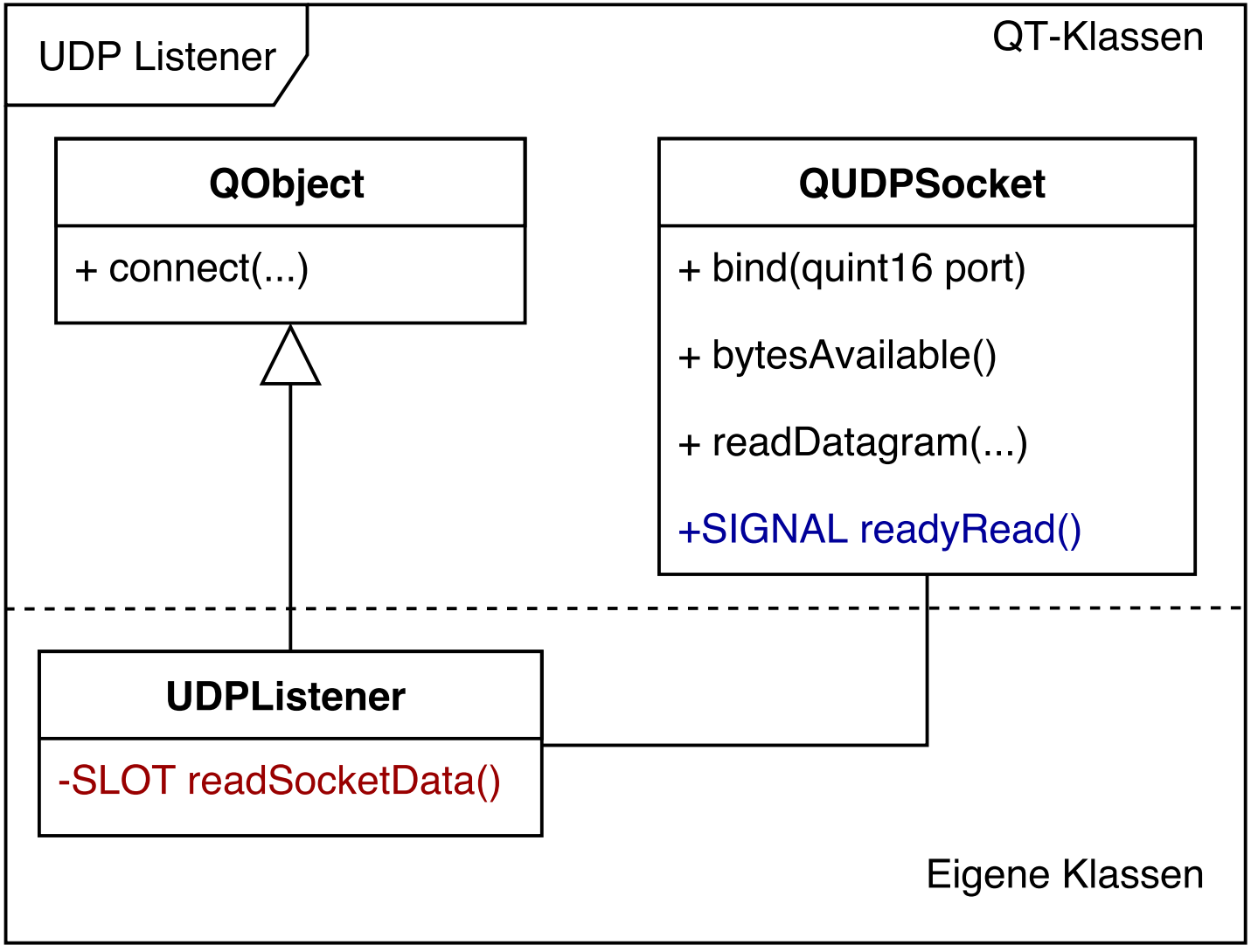
Sonstiges

`QByteArray` Enthält die empfangenen oder zu sendenden Daten.

UDP Beispiel



Klassendiagramm UDP-Listener



Erstellung eines UDP-Listener in QT

1. Neues Projekt erstellen
2. In der Projektdatei (*.pro) das `network`-Modul hinzufügen

   `QT += network`
3. Eine von `QObject` abgeleitete Klasse erstellen

   `UDPListener`
4. `QUDPSocket` als privates Objekt von `TCPServer` anlegen

   `m_socket`
5. Einen Slot in der Klasse `UDPListener` implementieren

   `void readSocketData()`
6. `m_socket` mit Slot verbinden und einem Port zuweisen

UDPListener Klasse

Listing 1: code/UDPListener/udplistener.h

```
1 private:
2   QUdpSocket m_socket; // #include <QUdpSocket>
3
4 private slots:
5   void readSocketData();
```

m_socket einrichten

- Mit der Methode `bind` kann einem Socket ein Port zugewiesen werden

Listing 2: code/UDPListener/udplistener.cpp

```
1 UDPListener::UDPListener(QObject *parent) : QObject(parent)
2 {
3   connect(&m_socket, SIGNAL(readyRead()), this, SLOT(readSocketData
      ()));
4   m_socket.bind(2006);
5 }
```

Slot für das Auslesen eines Sockets

- `qDebug()` erzeugt wie `std::cout` eine Ausgabe in der Konsole

Listing 3: code/UDPListener/udplistener.cpp

```
1  void UDPListener::readSocketData()
2  {
3    while (m_socket.bytesAvailable()) {
4      QByteArray byteArray;
5      byteArray.resize(m_socket.bytesAvailable());
6
7      m_socket.readDatagram(byteArray.data(), byteArray.size());
8
9      qDebug() << byteArray;
10   }
11 }
```

Erstellung eines UDP-Senders in QT

1. Neues Projekt erstellen
2. In der Projektdatei das `network`-Modul hinzufügen

   pro Datei:

   ```
   QT += network
   ```
3. `QUDPSocket` als Objekt in `main()` anlegen

   ```
   QUDPSocket socket;
   ```
4. Daten an IP-Adresse und Port schicken

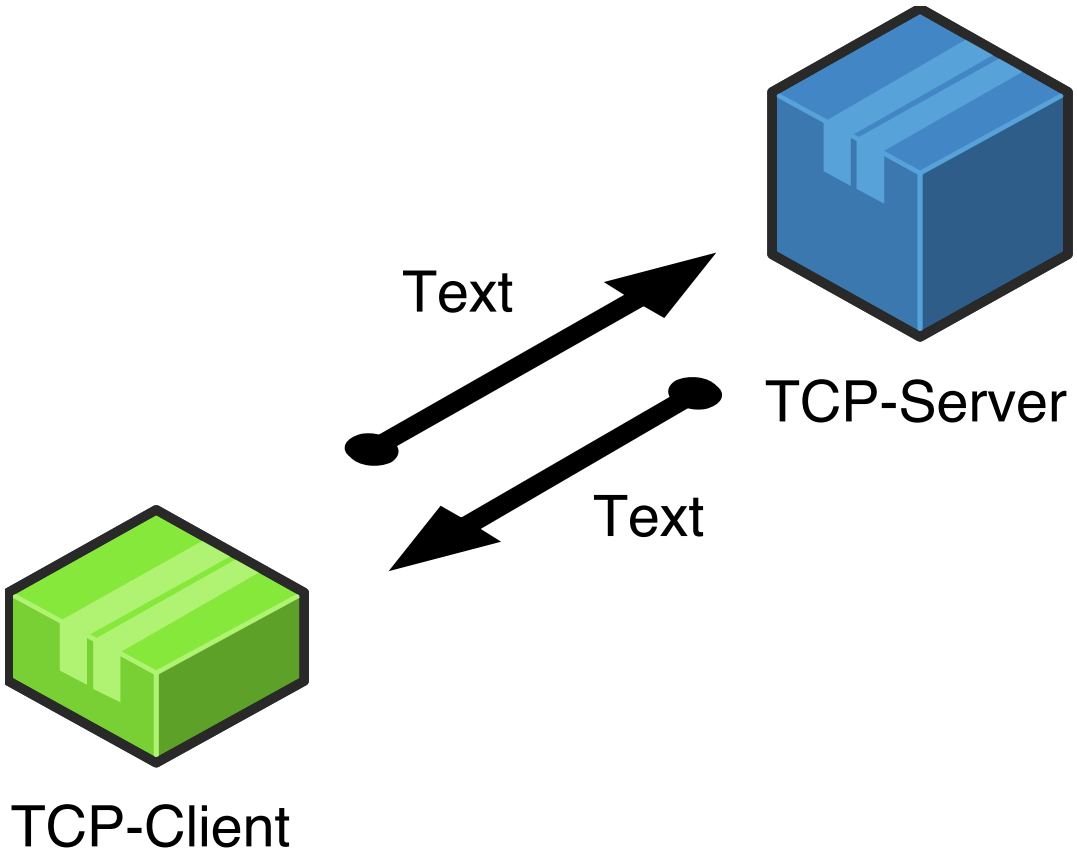
   ```
   socket.writeDatagram(...)
   ```

UDP-Sender Implementierung

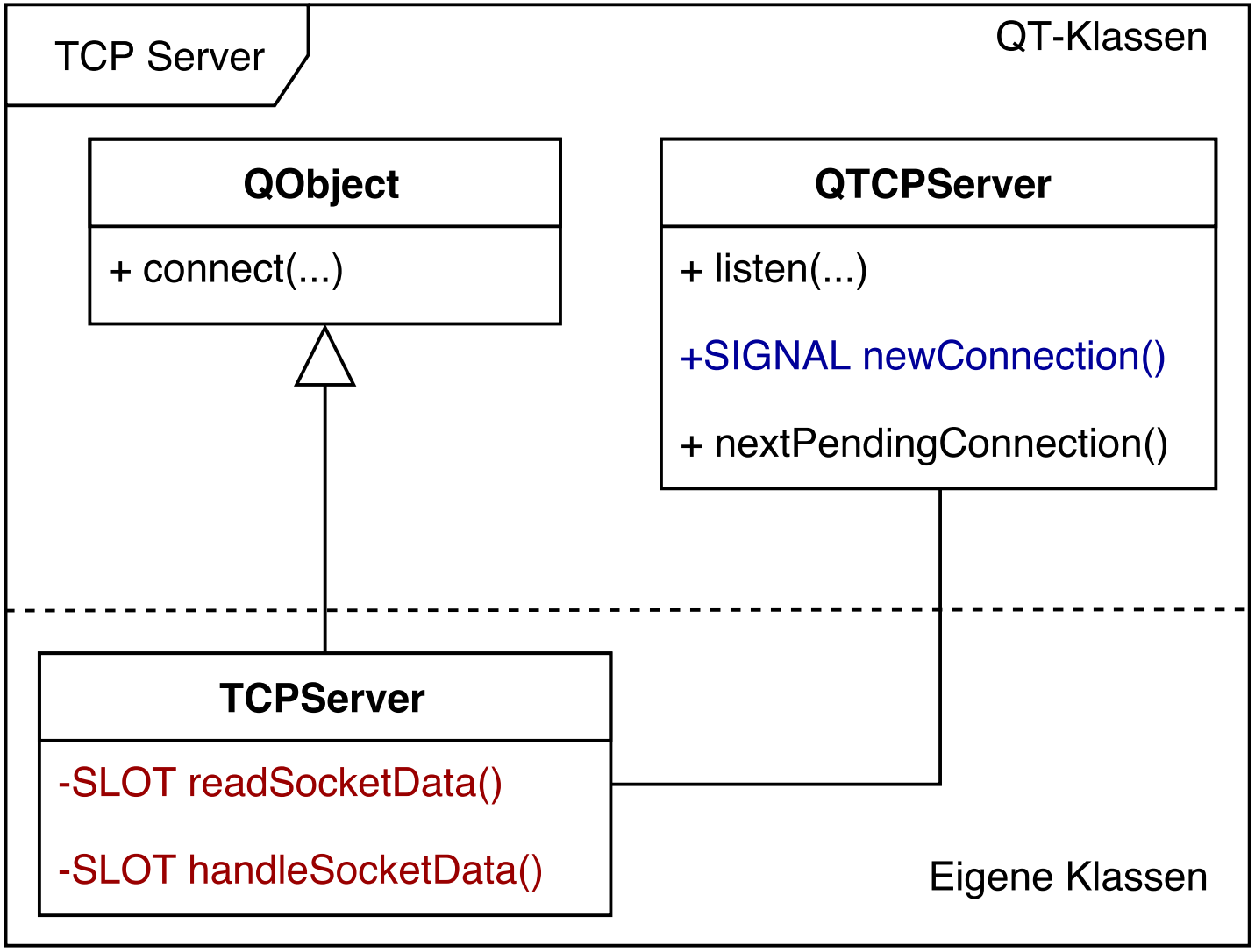
Listing 4: code/UDPSender/main.cpp

```
1 int main(int argc, char *argv[])
2 {
3   QCoreApplication a(argc, argv);
4
5   QUdpSocket socket;
6   socket.writeDatagram("Hello from the Sender side", QHostAddress::
      LocalHost, 2006);
7
8   return a.exec();
9 }
```

TCP Beispiel



Klassendiagramm TCP-Server



Erstellung eines TCP-Servers in QT

1. Neues Projekt erstellen
2. In der Projektdatei das `network`-Modul hinzufügen
   pro Datei: `QT += network`
3. Eine von `QObject` abgeleitete Klasse erstellen
   `TCPServer`
4. `QTCPServer` als privates Objekt von `TCPServer` anlegen
   `m_server`
5. Zwei Slots in der Klasse `TCP-Server` implementieren
   `void handleNewConnection()`
   `void readSocketData()`
6. `m_server` mit `handleNewConnection()` verbinden und die Funktion `listen` aufrufen

TCPServer Klasse

Die Klasse `QTCPServer` enthält alle relevanten Methoden für das Erstellen eines TCP-Servers.

Listing 5: code/TCPServer/tcpserver.h

```
1 private:
2   QTcpServer m_server; //#include <QTcpServer>
3
4 private slots:
5   void handleNewConnection();
6   void readSocketData();
```

m_Server einrichten

- Mit der Methode `listen` können IP-Adressen eingegrenzt und ein Port für eingehende Verbindungen festgelegt werden.
- `newConnection` wird aufgerufen, wenn ein Client sich mit dem Server verbinden möchte

Listing 6: code/TCPServer/tcpserver.cpp

```
1 TCPServer::TCPServer(QObject *parent) : QObject(parent)
2 {
3   connect(&m_server, SIGNAL(newConnection()),
4       this,SLOT(handleNewConnection()));
5
6   m_server.listen(QHostAddress::Any, 2006);
7 }
```

Slot für Verbindungsanfragen

- `nextPendingConnection()` gibt Sockets zu den anfragenden Clients zurück

Listing 7: code/TCPServer/tcpserver.cpp

```
1  void TCPServer::handleNewConnection()
2  {
3    QTcpSocket *tcpSocket = m_server.nextPendingConnection();
4
5    while(tcpSocket)
6    {
7      connect(tcpSocket, SIGNAL(readyRead()),
8          this, SLOT(readSocketData()));
9      tcpSocket = m_server.nextPendingConnection();
10   }
```

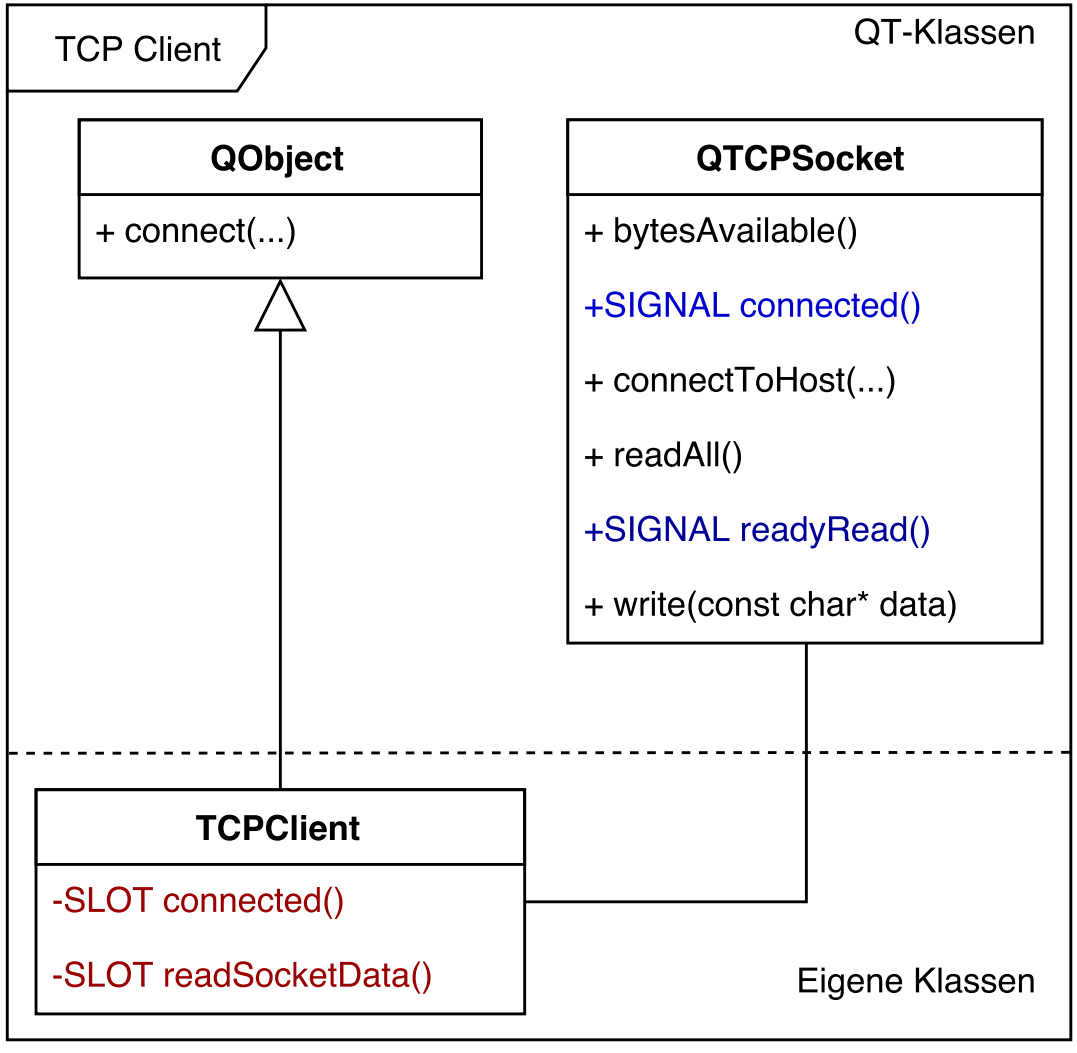
Slot für das Auslesen eines Sockets

- Weil ein Server mit mehreren Sockets verbunden sein kann, wird mit `this->sender()` das Objekt aufgerufen, welches das Signal emittiert hat
- `qDebug()` erzeugt wie `std::cout` eine Ausgabe in der Konsole

Listing 8: code/TCPServer/tcpserver.cpp

```
1 void TCPServer::readSocketData()
2 {
3   QTcpSocket *socket = (QTcpSocket*) this->sender();
4
5   while (socket->bytesAvailable()) {
6     qDebug()<< socket->readAll(); // #include <QDebug>
7   }
8 }
```

Klassendiagramm TCP-Client



Erstellung eines TCP-Clients in QT

1. Neues Projekt erstellen
2. In der Projektdatei das `network`-Modul hinzufügen
   pro Datei:
   ```
   QT += network
   ```
3. Eine von `QObject` abgeleitete Klasse erstellen
   ```
   TCPClient
   ```
4. `QTCPSocket` als privates Objekt von `TCPClient` anlegen
   ```
   QTCPSocket m_socket;
   ```
5. Zwei Slots in der Klasse `TCPClient` implementieren
   ```
   void connected();
   void readSocketData();
   ```
6. `m_socket` mit den Slots verbinden und eine Verbindung zum Server anfragen

TCPClient Klasse

Die Klasse `QTCPSocket` enthält die Schnittstelle zur Nutzung des TCP-Protokolls

Listing 9: code/TCPClient/tcpclient.h

```
1 private:
2   QTcpSocket m_socket; //#include <QTcpSocket>
3
4 private slots:
5   void connected();
6   void readSocketData();
```

m_socket verbinden

- Mit der Methode `connectToHost` kann die Verbindung zu einem TCP-Server aufgebaut werden

Listing 10: code/TCPClient/tcpclient.cpp

```
1 #include <QHostAddress>
2
3 TCPClient::TCPClient(QObject *parent) : QObject(parent)
4 {
5   connect(&m_socket, SIGNAL(connected()), this, SLOT(connected()));
6   connect(&m_socket, SIGNAL(readyRead()), this, SLOT(readSocketData
      ()));
7
8   m_socket.connectToHost(QHostAddress::LocalHost, 2006, QIODevice::
      ReadWrite);
9 }
```

Connected-Slot

- Der Slot `connected()` wird aufgerufen, falls eine Verbindung zum Server erfolgt ist
- Anschließend können Daten versendet werden

Listing 11: code/TCPClient/tcpclient.cpp

```
1 void TCPClient::connected()
2 {
3   m_socket.write("Hello from the Client");
4 }
```

Slot für das Auslesen eines Sockets

- Nur ein Socket vorhanden, welcher Teil der TCPClient-Klasse ist
- Ausleseroutine funktioniert wie auf der Server-Seite

Listing 12: code/TCPClient/tcpclient.cpp

```
1 void TCPClient::readSocketData()
2 {
3   while(m_socket.bytesAvailable())
4   {
5     qDebug() << m_socket.readAll();
6   }
7 }
```

Teil III

Projektarbeit

Anmeldung Projektmappe per E-Mail

Listing 13: Betreff der E-Mail

```
1 Anmeldung API-Projektmappe
```

Listing 14: Inhalt der E-Mail

```
1 Vorname:
2 Nachname:
3 Matrikelnummer:
4 GitHub-Benutzer:
5 GitHub-Team:
6 GitHub-Repository-URL: .git
7 GitHub-Wiki-URL: .wiki.git
```

An die folgende E-Mail-Adresse:
api-iff@tu-braunschweig.de

Anmeldung Projektarbeit per E-Mail

Anmeldung

- Jeder muss sich bis zum 30.06.2017 anmelden.
- Für die E-Mail sollen die Inhalte von Listing 13 und Listing 14 verwendet werden.
- Oder direkt den Link verwenden: api-iff@tu-braunschweig.de

3. Zyklus vom API-Spiralmodell beginnen

Aufgabe 1

Beginnen Sie den letzten Zyklus des API-Spiralmodells.

Externer-Code Review

Ankündigung

Am 27. Juni besteht die Möglichkeit, einen externen Code-Review durchzuführen. Diesbezüglich werden die anwesenden Gruppen vermittelt.

Wichtig

Es ist sinnvoll für das Review Quellcode zur Verfügung zu haben, da andere Gruppen wertvolles Feedback liefern können.

Fragen?

Gibt es noch Fragen?