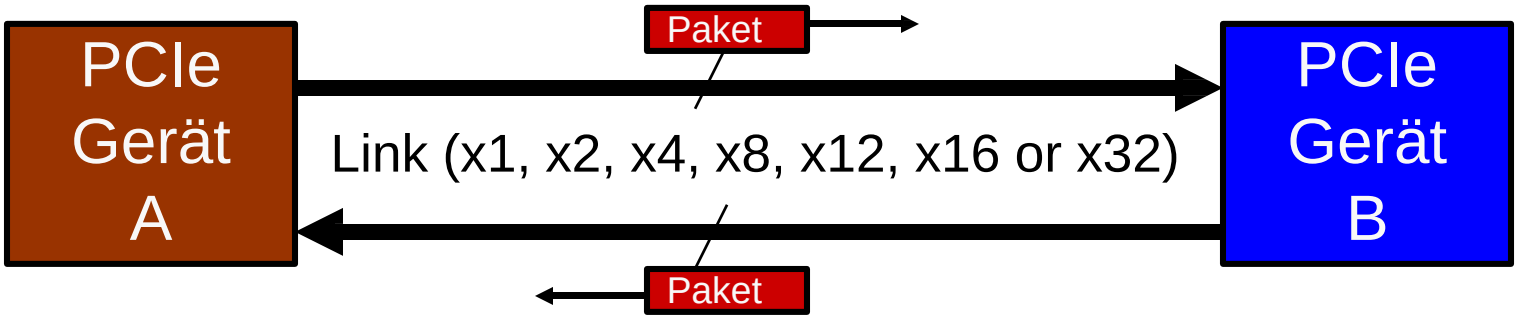
Kapitel 3

Rechnerarchitekturen für Echtzeitsysteme

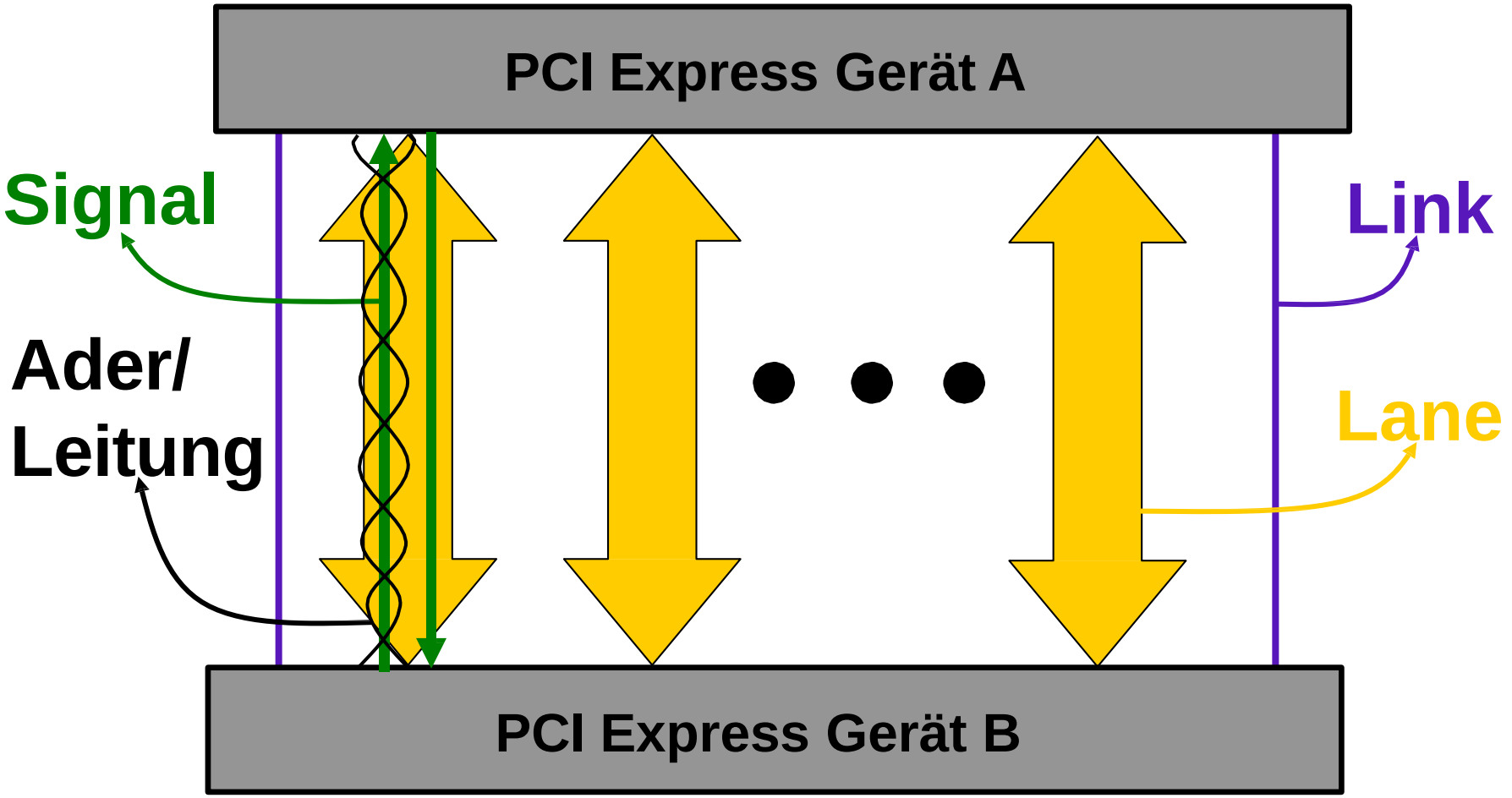
PCI Express Eigenschaften

- **Punkt-zu-Punkt-Verbindungen**
- **Weniger Verbindungsleitungen durch seriellen Bus**
- **Skalierbar: x1, x2, x4, x8, x12, x16, x32 lanes/link**
- **Doppelte Simplex-Verbindung**
- **Differentielle Übertragung**
- **2.5, 5.0 and 8.0 GT/s Transfer/Richtung/s**
- **Paketbasiertes Transaktionsprotokoll**



www.pcisig.com/developers/main/training_materials/

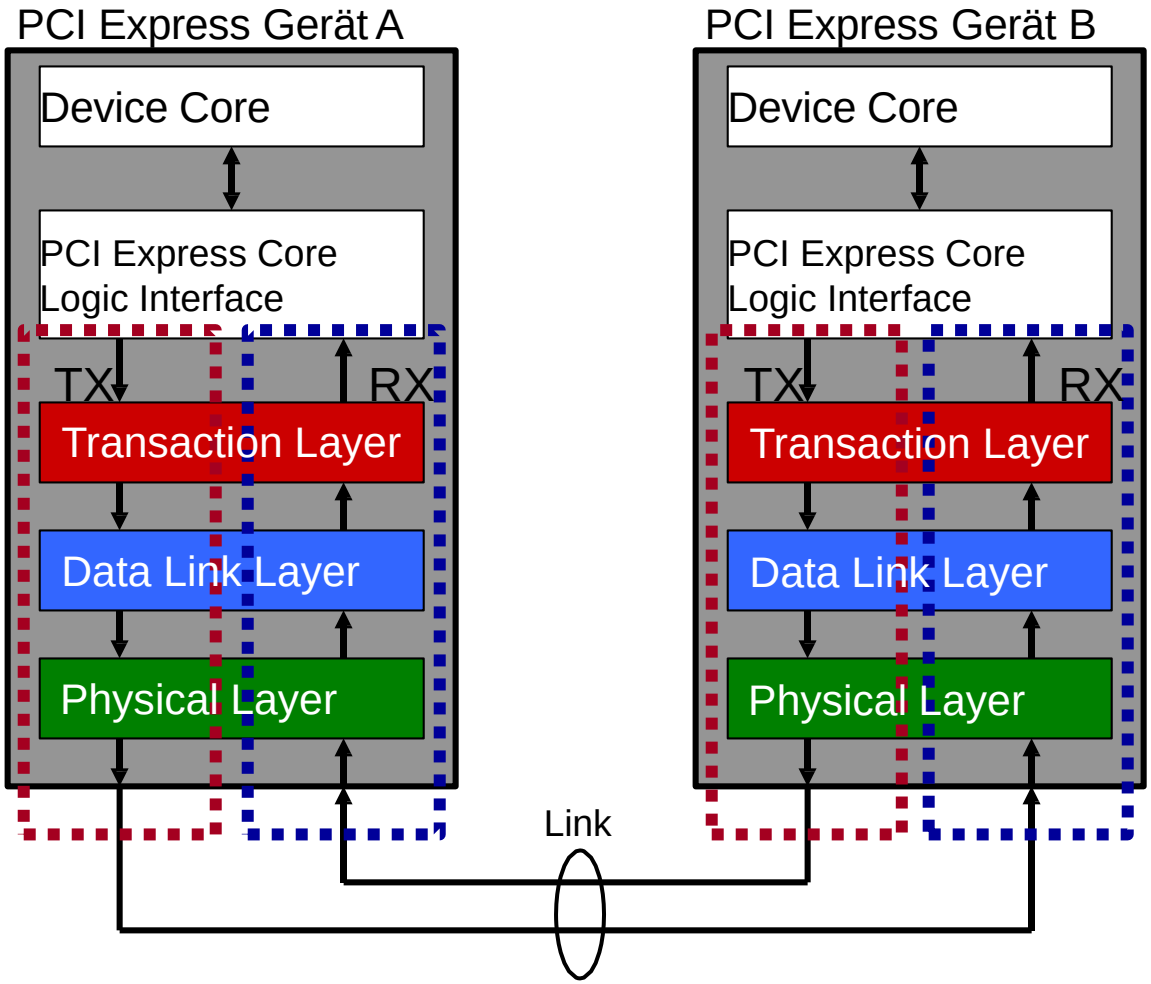
PCI Express Terminologie



www.pcisig.com/developers/main/training_materials/

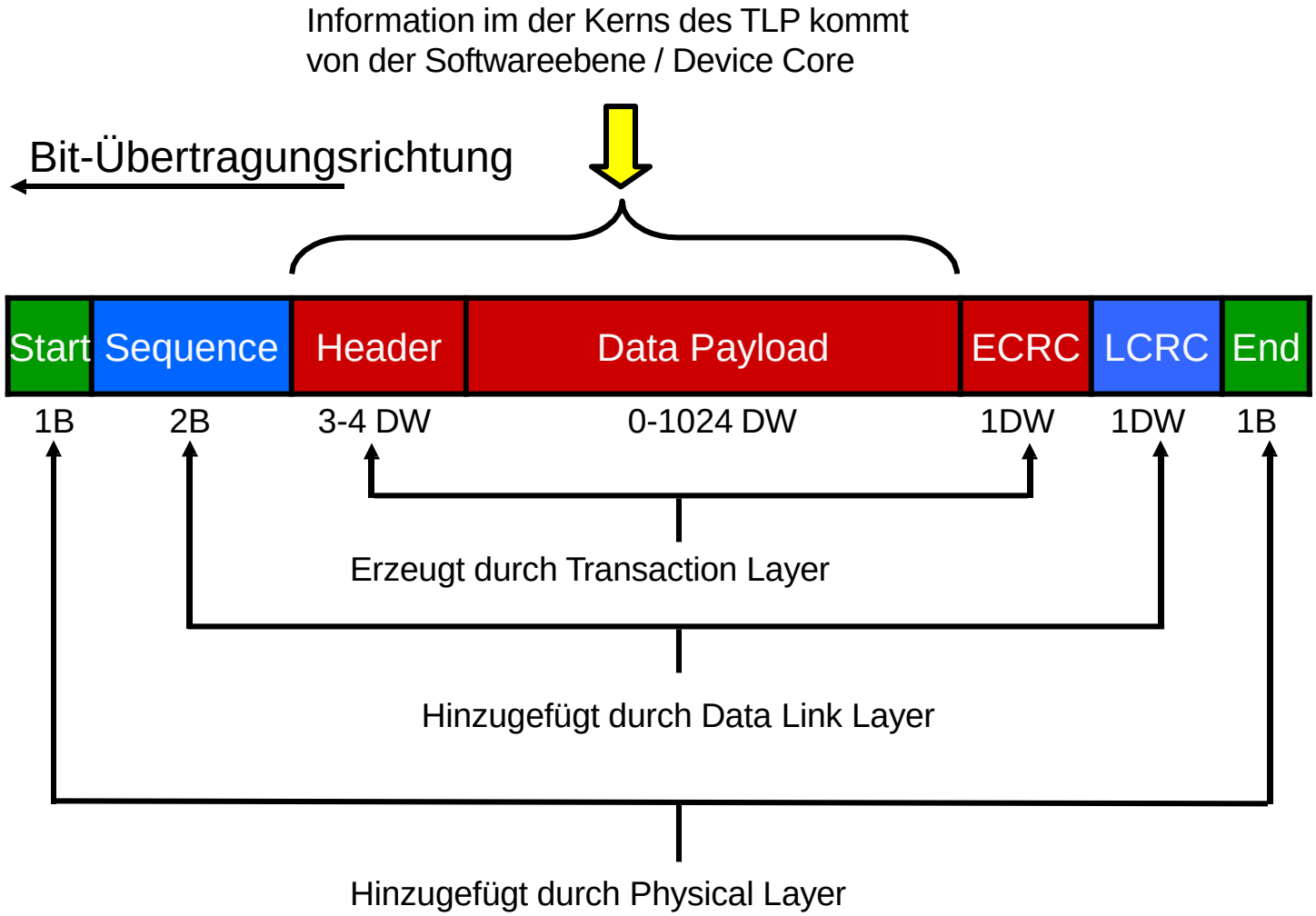
Intelligente Prozess-automation und Robotik

PCI Express Geräteebenen



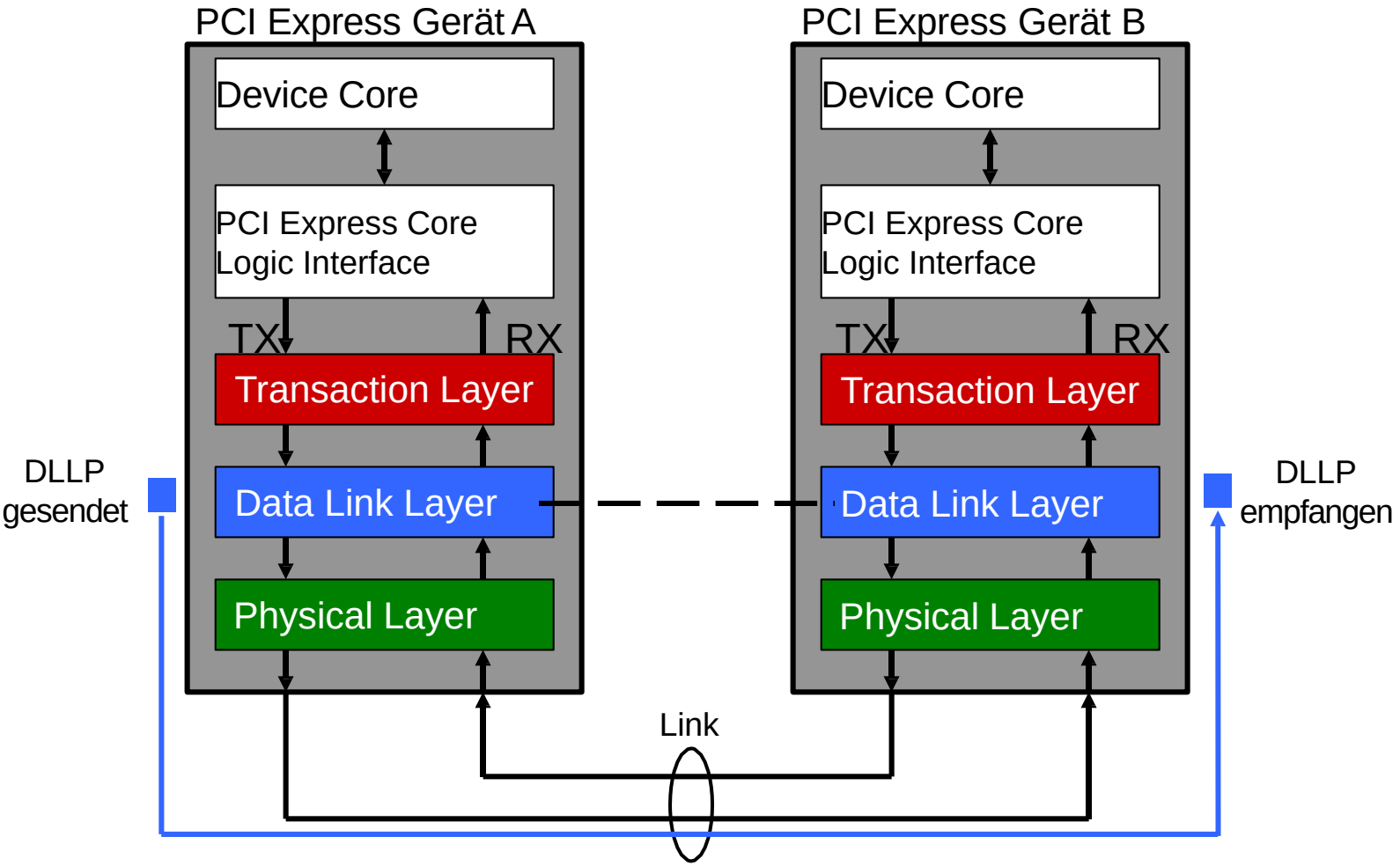
www.pcisig.com/developers/main/training_materials/

Aufbau TLP (transaction layer packet)



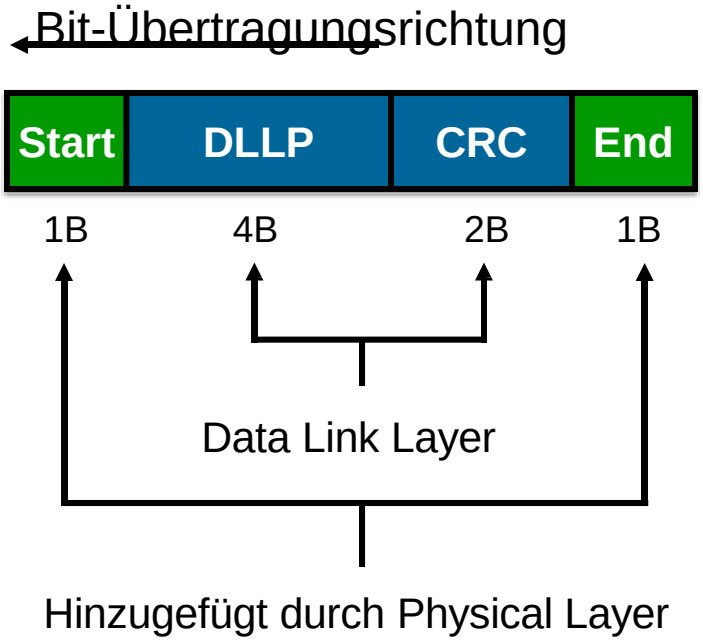
www.pcisig.com/developers/main/training_materials/

PCIe-Protokoll Verbindungsebene (DDLP)



www.pcisig.com/developers/main/training_materials/

Aufbau DDLP (data link layer packet)



- ACK/NAK-Pakete
- Flußkontrollpakete
- Energieverwaltungspakete
- Herstellerspezifische Pakete

www.pcisig.com/developers/main/training_materials/

Interrupt-Modell bei PCI Express

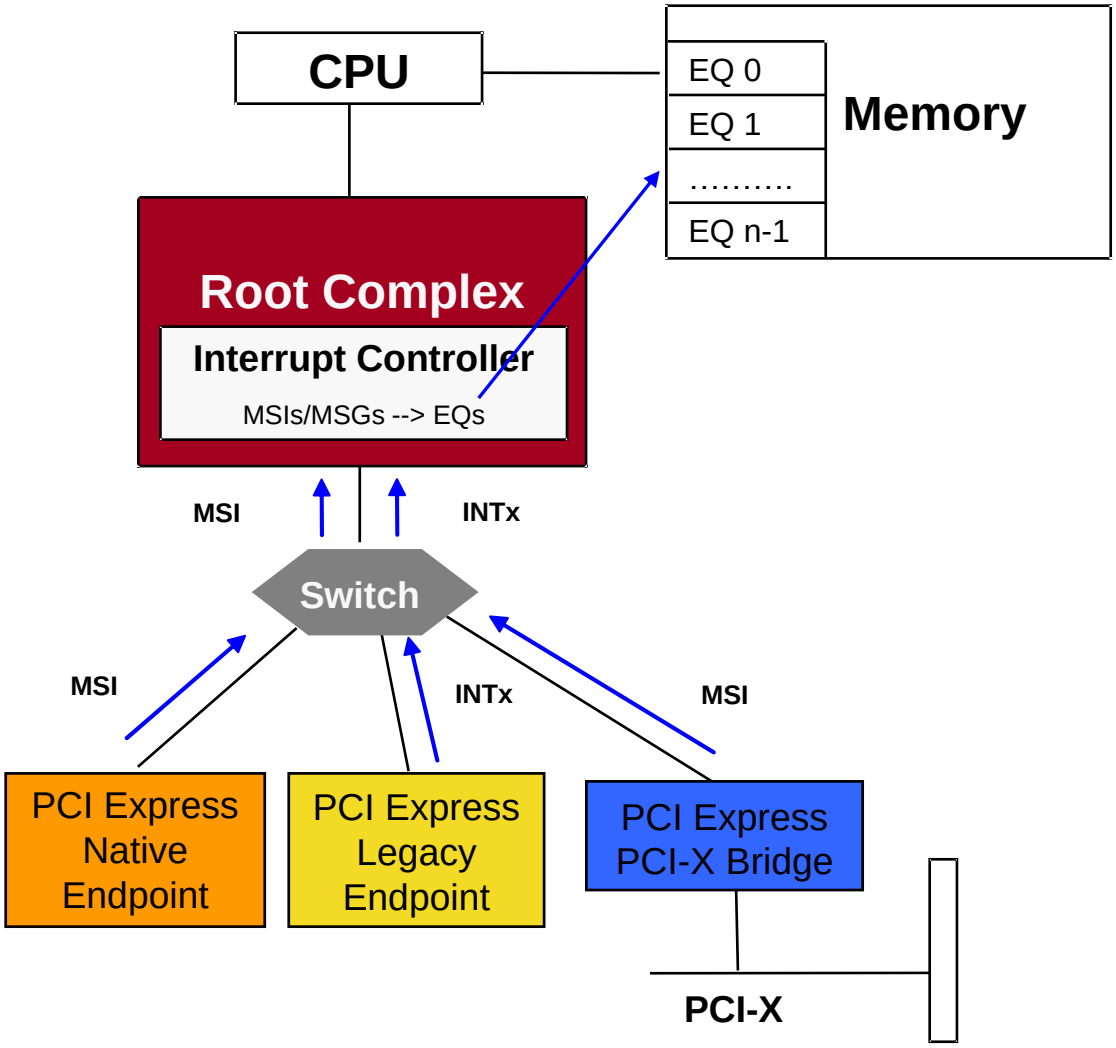
PCI Express unterstützt 3 Unterbrechungsmechanismen:

1. **Message Signaled Interrupts (MSI)**
   - Bereits im parallelen PCI ab Version 2.2
   - 1-32 Interrupts pro Gerät möglich, 1 Interruptvektor
   - Implementierung über MSI capability register mit 32/64-bit (Legacy-) oder 64-bit (Native PCI-Express Endpunkte) -> Schreibzugriff auf Adresse des Geräts

2. **Message Signaled Interrupts - X (MSI-X)**
   - Bis zu 2048 Interrupts pro Gerät möglich, minimal 64 unterstützt
   - Zusätzlich mehrere Interruptvektoren durch mehrere mögliche Einträge im 64-bit MSI-X capability register (Legacy und Native Endpunkte)

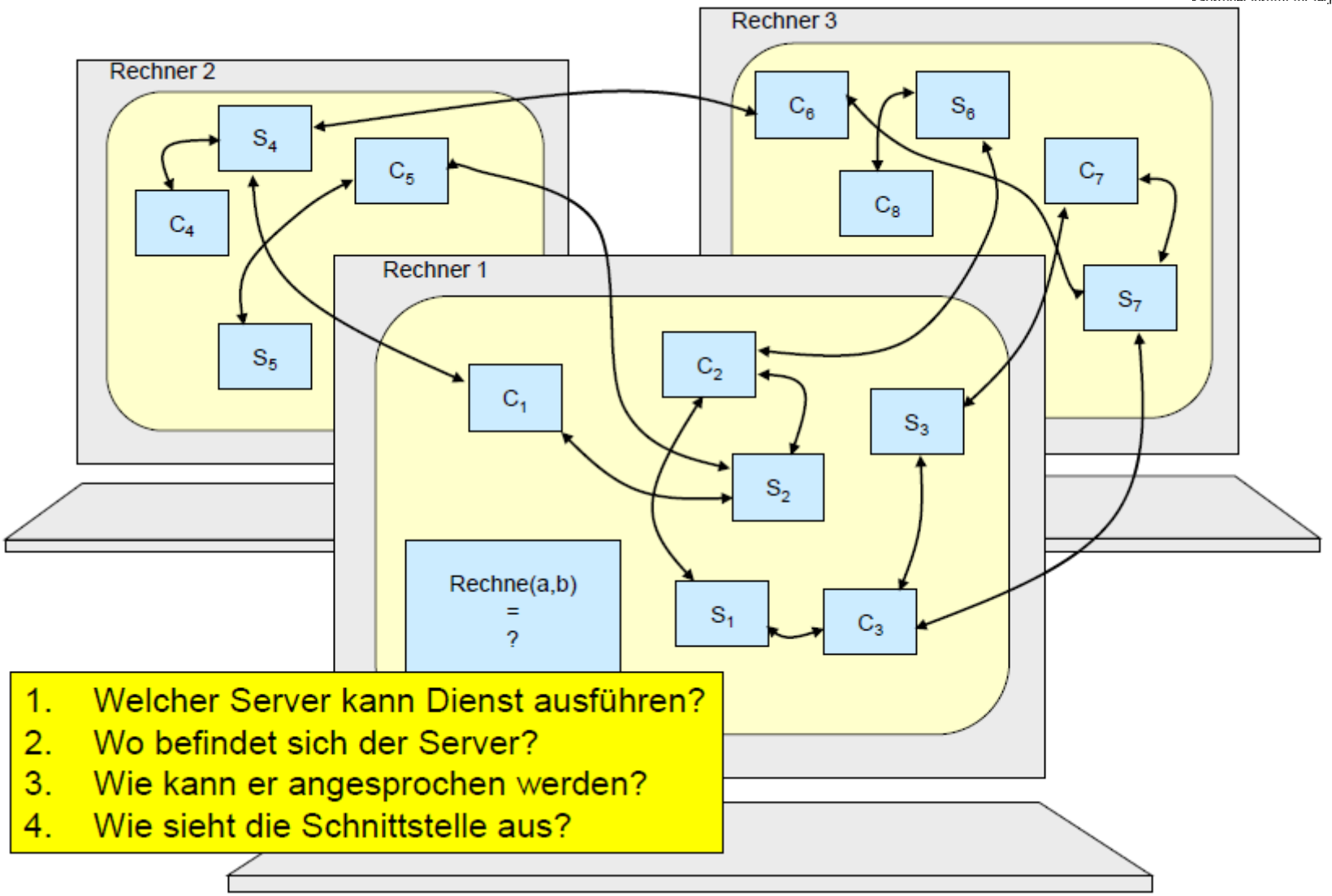
3. **INTx Emulation**
   - Native und Legacy-Endpunkte müssen Legacy-INTx-Emulation unterstützen
   - PCI Express definiert In-Band-Nachrichten, die die vier physikalischen Interrupt-Signale (INTA-INTD) zwischen PCI-Geräten und dem System-Interruptcontroller emulieren
   - Switches müssen Weiterleitung unterstützen

PCIe-Interrupt

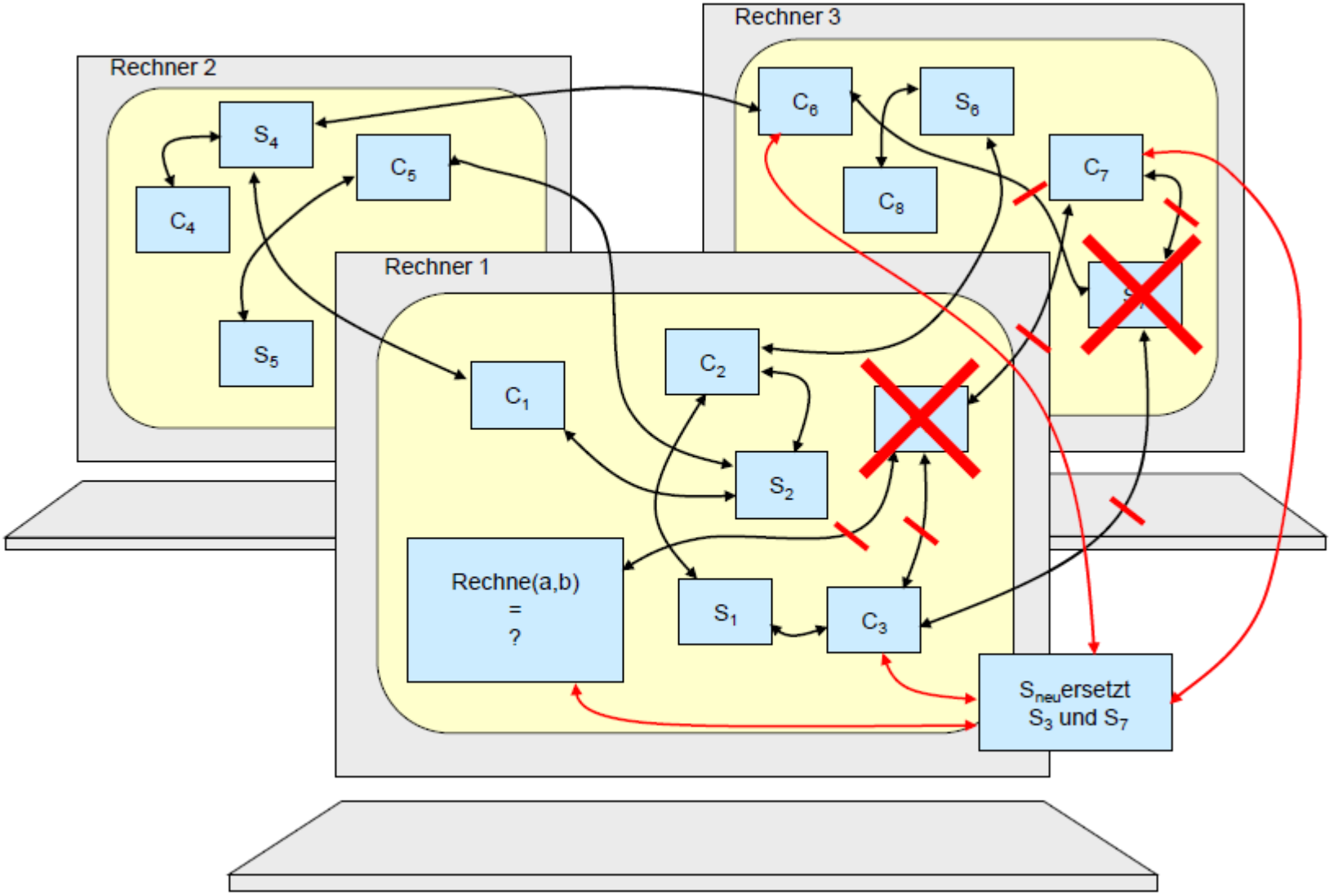


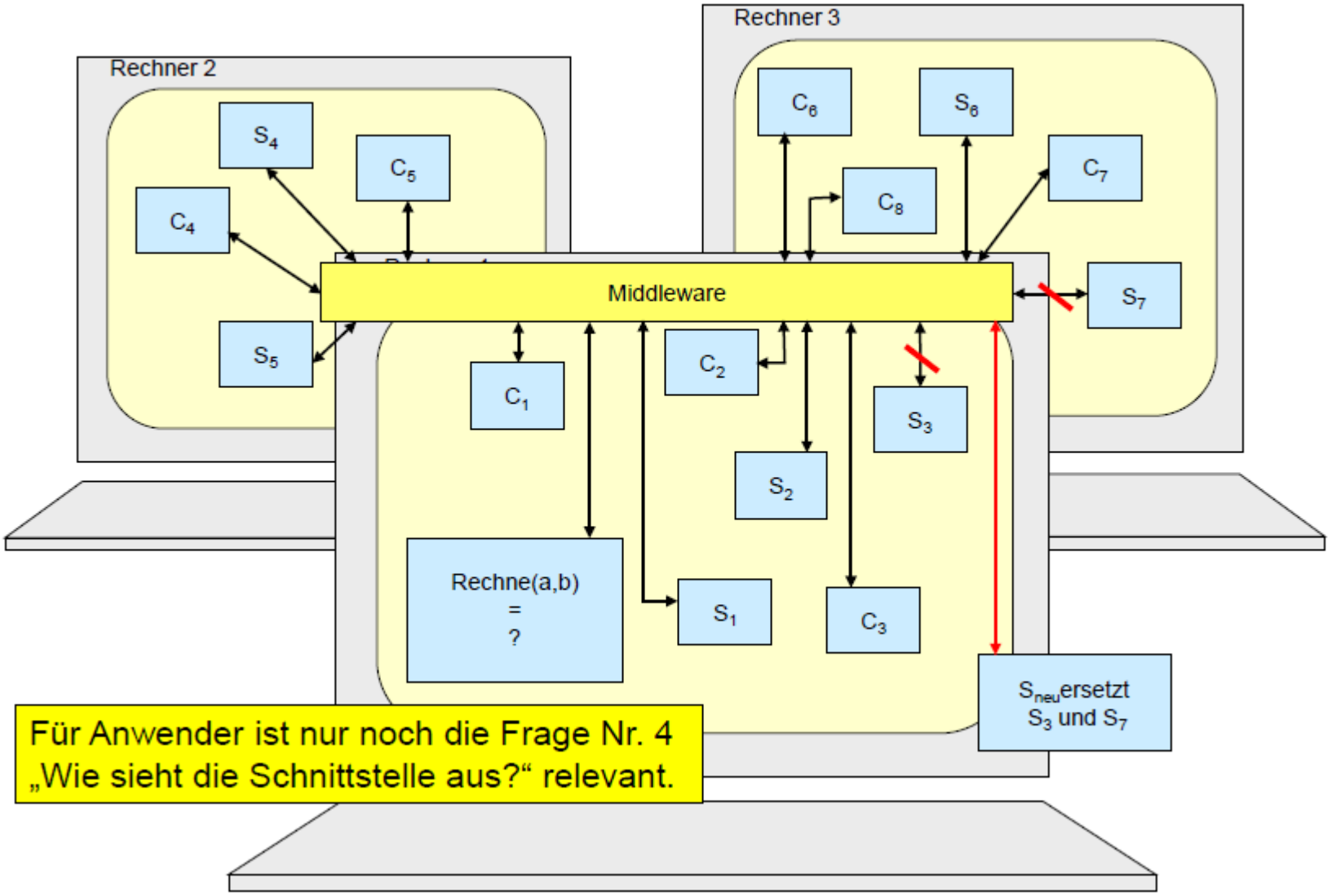
Kapitel 4

Echtzeitmiddleware



1. Welcher Server kann Dienst ausführen?
2. Wo befindet sich der Server?
3. Wie kann er angesprochen werden?
4. Wie sieht die Schnittstelle aus?


Rechner 3
Rechner 2
Rechner 1
C6
S6
C7
C8
S4
C5
C4
S5
C2
C1
S2
Rechne(a,b)
=
?
S1
C3
Sneu ersetzt
S3 und S7

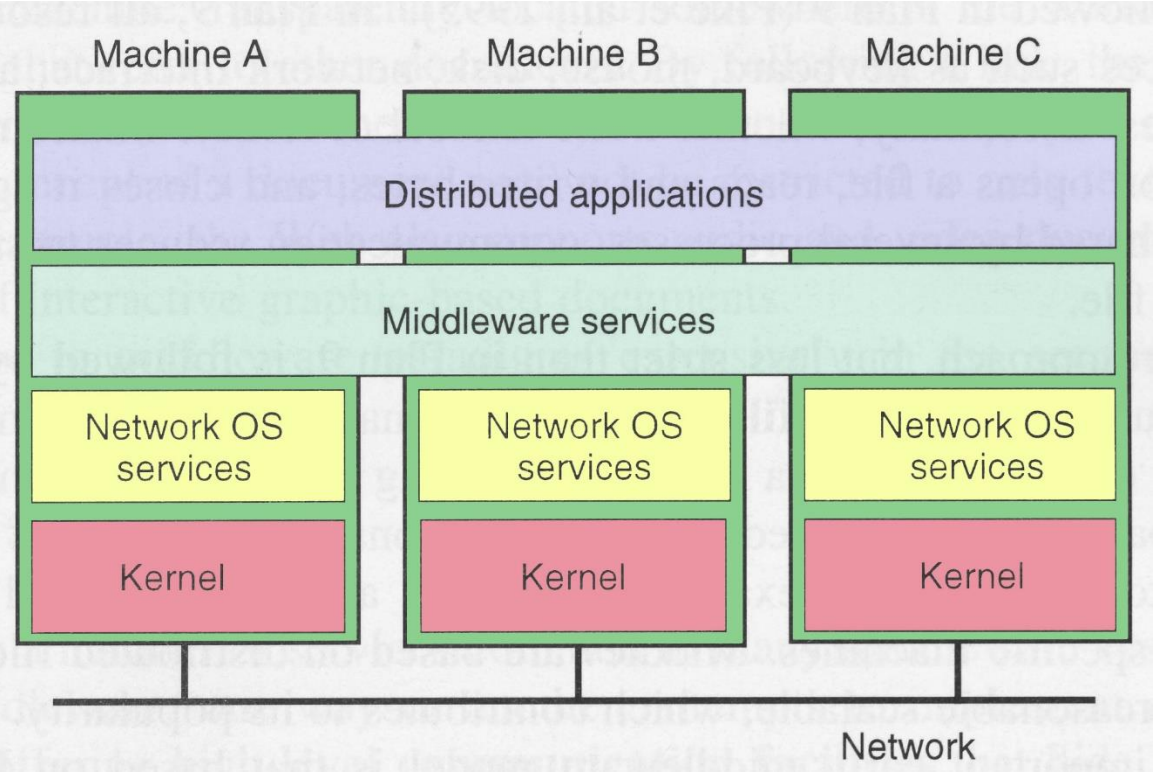

Rechner 2
Rechner 3
S_4
C_5
C_4
S_5
C_6
S_6
C_7
C_8
Middleware
S_7
C_1
C_2
S_3
S_2
Rechne(a,b)
=
?
S_1
C_3
S_{neu} ersetzt
S_3 und S_7
Für Anwender ist nur noch die Frage Nr. 4
„Wie sieht die Schnittstelle aus?“ relevant.

Zum Begriff “Middleware”

Middleware […] bezeichnet in der Informatik anwendungsneutrale Programme, die so zwischen Anwendungen vermitteln, dass die Komplexität dieser Applikationen und ihrer Infrastruktur verborgen wird. […] Im Gegensatz zu niveautieferen Netzwerkdiensten, welche die einfache Kommunikation zwischen Rechnern handhaben, unterstützt eine Middleware die Kommunikation zwischen *Prozessen*.

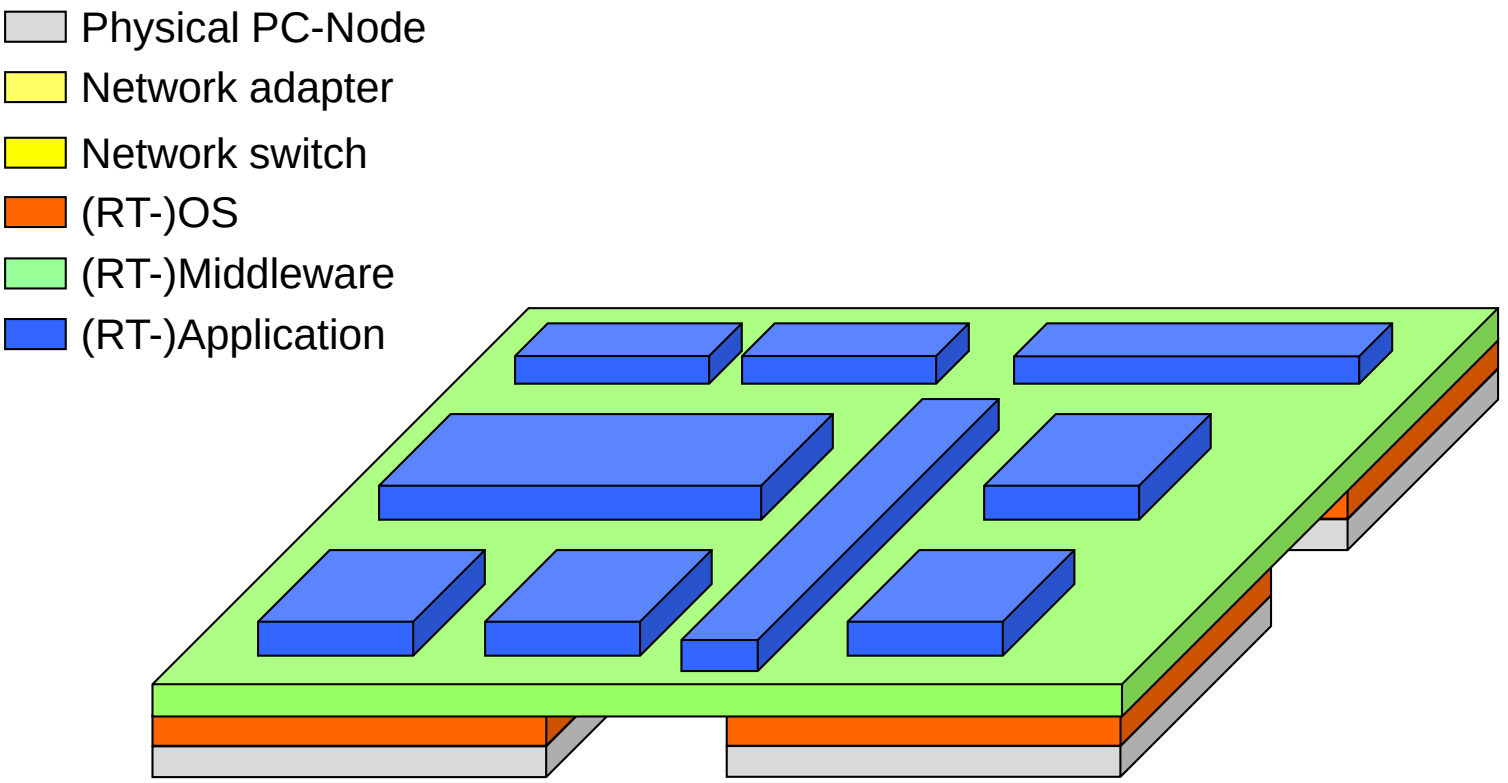
Quelle: W. R. Ruh, F. X. Maginnis, and W. J. Brown : Enterprise Application Integration. Wiley, 2001.

Zum Begriff “Middleware”

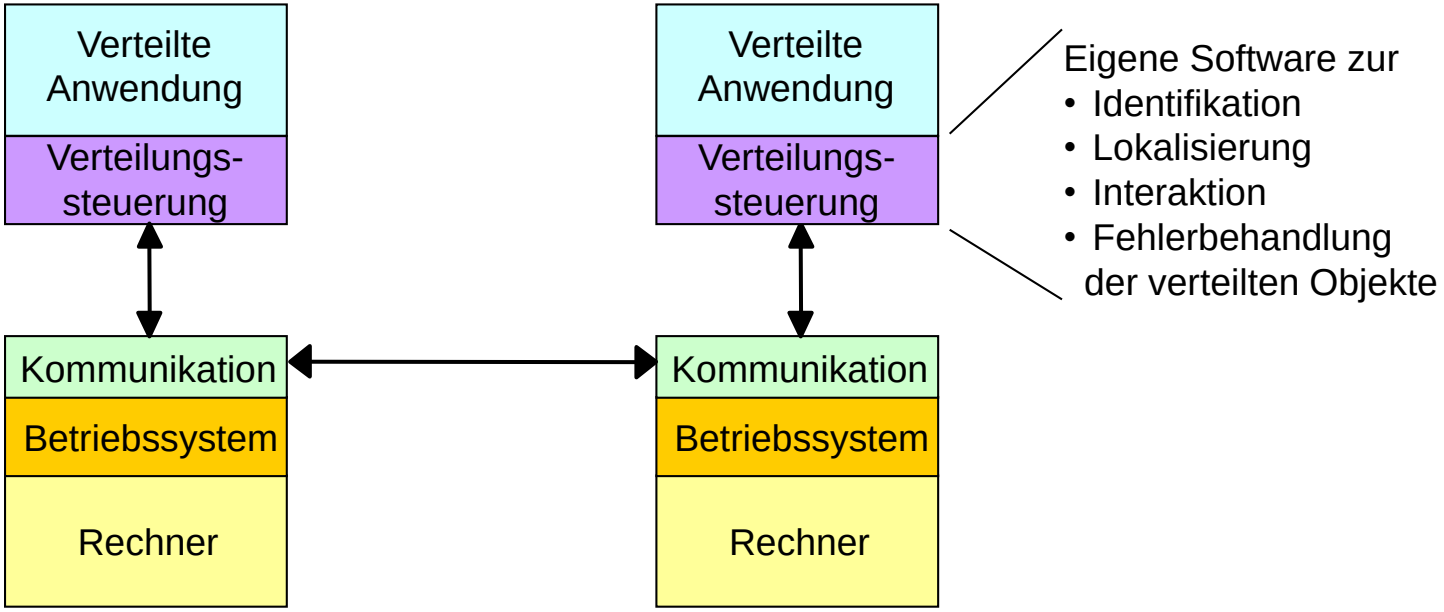


Source: Andrew S. Tannenbaum and Maarten van Steen. Distributed Systems, Prentice Hall, 2002.

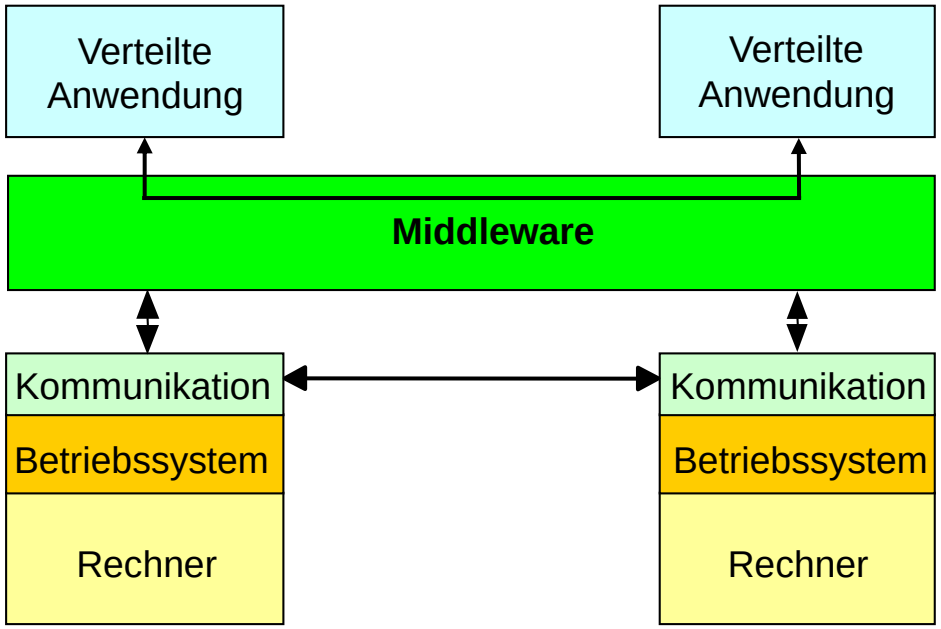
Zum Begriff “Middleware”



Eine verteilte Anwendung ohne Middleware



Eine verteilte Anwendung mit Middleware



Standardisierte Middleware
- Mittler zwischen verteilten Objekten
- Verbirgt die Verteilung
- Übernimmt die Aufgaben der Identifikation, Lokalisierung, Interaktion und Fehlerbehandlung

Aufgaben der Middleware:

- Identifikation (Verteilte Objekte über Namen, Referenz identifizieren)
- Lokalisierung (Ort eines Objektes)
- Interaktion (Aufträge, Nachrichten, entfernte Prozeduraufrufe)
- Erzeugung und Vernichtung
- Fehlerbehandlung

Vorteile durch den Einsatz von Middleware:

- die Anwendung muss sich nicht um die Verteilung kümmern
- verteilte und lokale Anwendungen können auf gleiche Weise entwickelt werden
- bei einer Änderung der Verteilung bzw. einer Rekonfiguration muss die Anwendung nicht geändert werden
- alle für die Verteilung notwendigen Aufgaben werden von einer einheitlichen, standardisierten Software übernommen

Vorteile durch den Einsatz von Middleware:

- die Portierbarkeit der Anwendung wächst
- eine heterogene Umgebung bleibt weitestgehend verborgen
- die Testbarkeit und Wartbarkeit der Anwendung wird verbessert

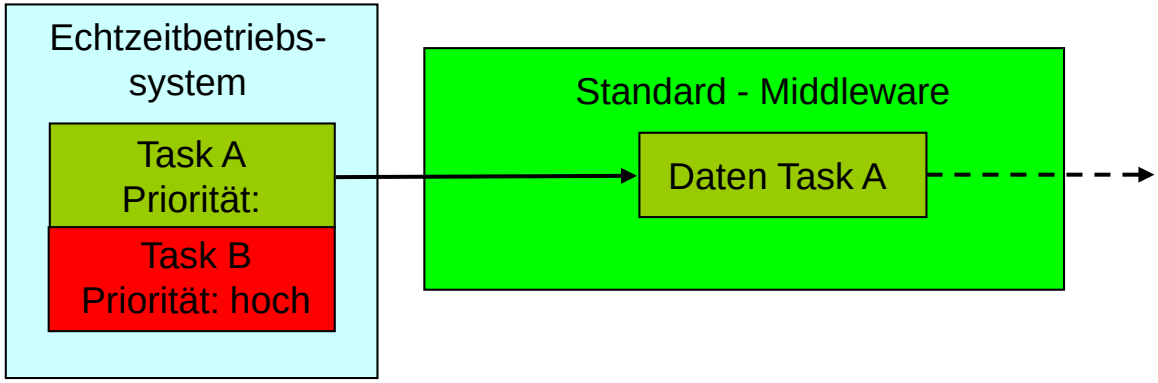
⇒ **die Entwicklungszeiten und -kosten werden reduziert**

Nachteile durch den Einsatz von Middleware:

- zusätzlicher Speicher-Overhead
- zusätzlicher Verarbeitungs-Overhead
- ‚Standard'-Middleware-Architekturen sind für Echtzeitanwendungen nicht geeignet
- Die Kombination einer Standard-Middleware mit einem Echtzeit-Betriebssystem ergibt kein echtzeitfähiges System

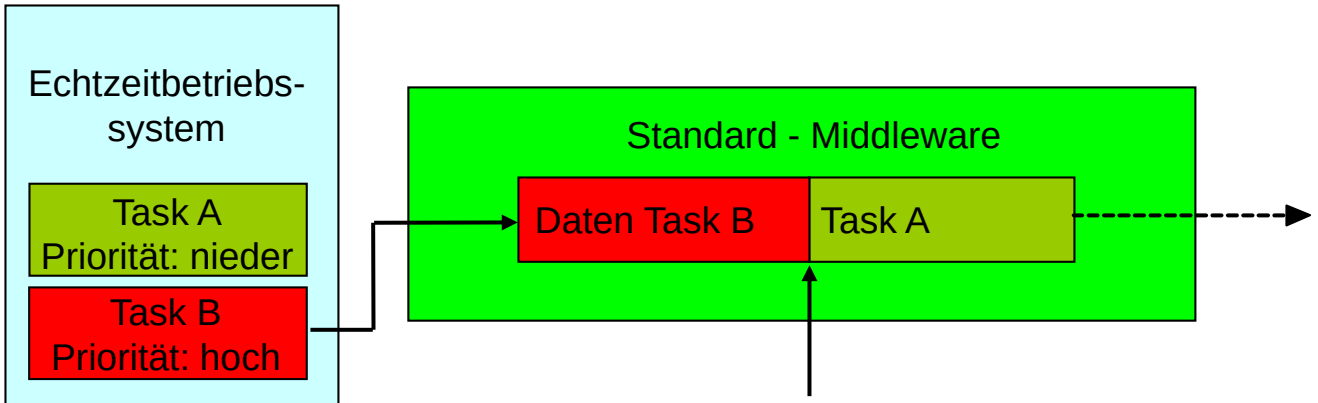
Prioritäteninversion durch Standard-Middleware in Echtzeitsystemen

Prioritäten-Inversion durch Unterbrechung der Prioritätenkette



Prioritäteninversion durch Standard-Middleware in Echtzeitsystemen

Prioritäten-Inversion durch Unterbrechung der Prioritätenkette



Die hochprioren Daten von Task B werden durch die niederprioren Daten von Task A blockiert
=> Prioritäten-Inversion

Anforderungen an eine Middleware für Echtzeitsysteme

- Wahrung der End-zu-End-Prioritäten
- Definition von Zeitbedingungen bzw. Zeitschranken
- Echtzeitfähige Funktionsaufrufe
- Echtzeitfähige Ereignisbehandlung
- Unterstützung des Echtzeitschedulings

  (Ankom. Daten von extern. Rechn. müssen entsprechend ihrer Priorität weiterverarbeitet werden)

Overview of OMG Data Distribution Service

High Performance real-time data-centric publish/subscribe middleware

- The right data, at the right place, at the right time—all the time
- Fully distributed, multicast-enabled, high performance, highly scalable, & high availability, hot-swap/hot-hot architecture

Blends data-centric & real-time publish/subscribe technologies

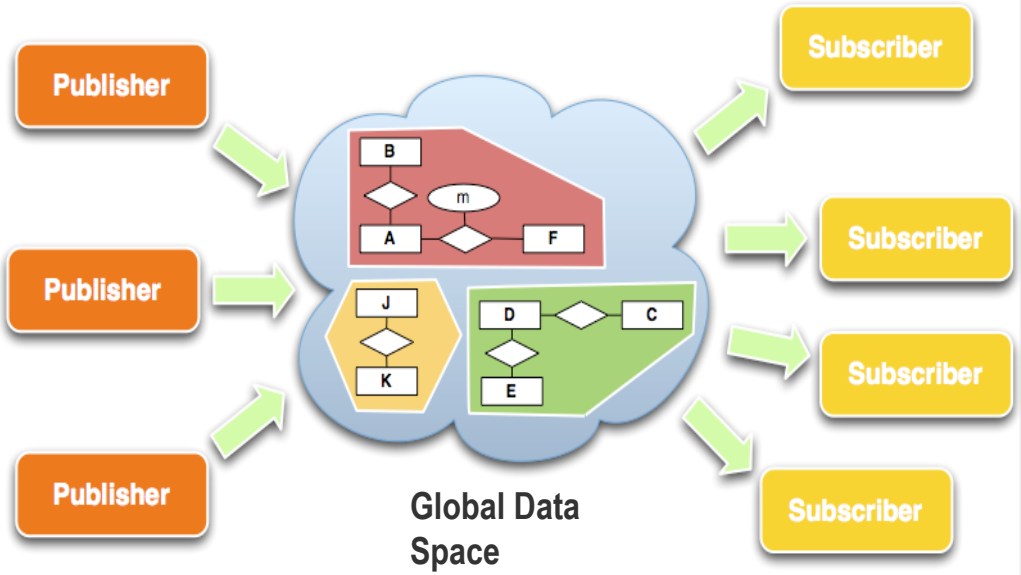
- Content-based subscriptions, (continuous) queries, & filters
- Fine-grained, policy-based tuning of resource usage, data delivery, availability QoS
- Optimal networking & computing resources usage

Loosely coupled

- Plug & play architecture with dynamic discovery
- Time & space decoupling

Open standard

- OMG DDS v1.2 latest version www.omg.org/dds



Literature: DDS

PrismTech Online Tutorial

- http://download.prismtech.com/docs/Vortex/html/ospl/DDSTutorial/index.html

OMG Website

- http://www.omg.org/dds/

RTI Website

- https://www.rti.com/gettingstarted
- https://community.rti.com/documentation

More details

- http://community.rti.com/docs/html/api_cpp/group__DDSQosTypesModule.html

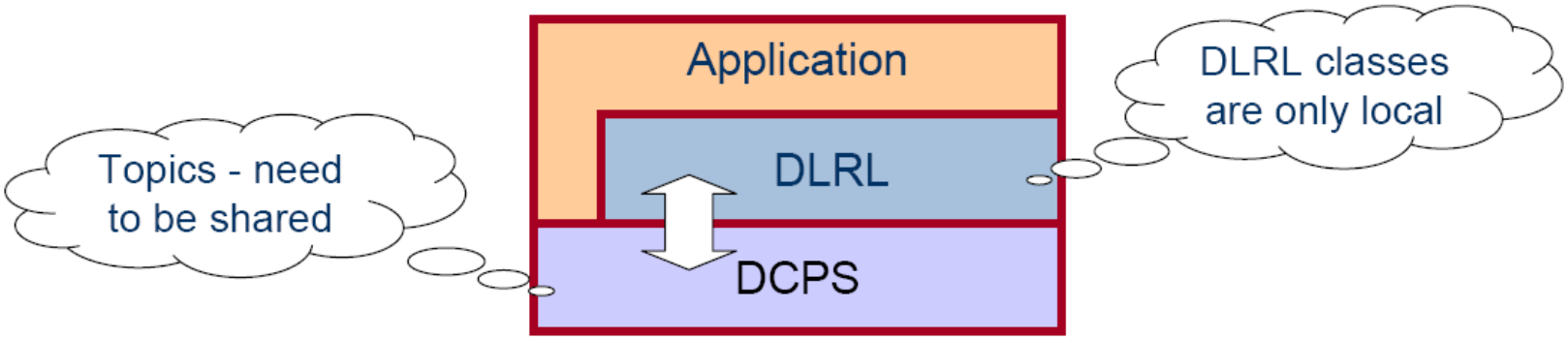
DDS Architectural Elements

Data-Centric Publish-Subscribe (DCPS)

- The lower layer APIs apps can use to exchange topic data with other DDS-enabled apps according to designated QoS policies

Data Local Reconstruction Layer (DLRL)

- The upper layer APIs that define how to build a local object cache so apps can access topic data as if it were local



DDS spec only defines policies & interfaces between application & service

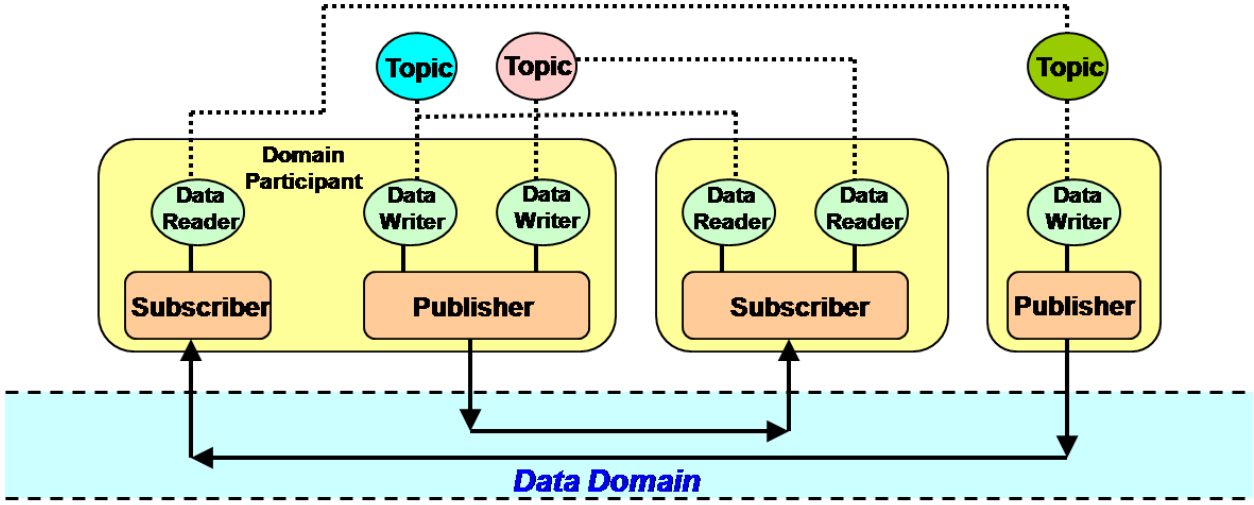
- Doesn't address protocols & techniques for different actors implementing the service
- Doesn't address management of internal DDS resources

Key DCPS Entities

DCPS Entities include

- Topics
  - Typed data
- Publishers
  - Contain DataWriters
- Subscribers
  - Contain DataReaders
- DomainParticipants
  - Entry points

- Data can be accessed in two ways
  - Wait-based (synchronous calls)
  - Listener-based (asynchronous callbacks)
- Sophisticated support for filtering
  - e.g., Topic, Content-FilteredTopic, or MultiTopic
- Configurable via (many) QoS policies



- **Information Model.** Defines the structure, relations, & QoS, of the information exchanged by the applications, & supports **flat**, **relational,** & **object-oriented modeling**

- **Typed Global Data Space.** A logical data space in which applications read & write data **anonymously** & **asynchronously**, decoupled in space & time

- **Publisher/Subscriber.** Produce/Consume information into/from the Global Data Space

- **QoS.** Regulates the non-functional properties of information in the Global Data Space, e.g., reliability, availability, & timeliness, etc.

Quality of Service

QoS Policy	Applicability	RxO	Modifiable	
DURABILITY	T, DR, DW	Y	N	Data Availability
DURABILITY SERVICE	T, DW	N	N	
LIFESPAN	T, DW	-	Y	
HISTORY	T, DR, DW	N	N	
PRESENTATION	P, S	Y	N	Data Delivery
RELIABILITY	T, DR, DW	Y	N	
PARTITION	P, S	N	Y	
DESTINATION ORDER	T, DR, DW	Y	N	
OWNERSHIP	T, DR, DW	Y	N	
OWNERSHIP STRENGTH	DW	-	Y	
DEADLINE	T, DR, DW	Y	Y	Data Timeliness
LATENCY BUDGET	T, DR, DW	Y	Y	
TRANSPORT PRIORITY	T, DW	-	Y	
TIME BASED FILTER	DR	-	Y	Resources
RESOURCE LIMITS	T, DR, DW	N	N	
USER_DATA	DP, DR, DW	N	Y	Configuration
TOPIC_DATA	T	N	Y	
GROUP_DATA	P, S	N	Y	

OMG DDS: QoS Data Timeliness

- The **DEADLINE QoS** policy allows applications to define the maximum inter-arrival time for data. DDS can be configured to automatically notify applications when deadlines are missed.

- The **LATENCY_BUDGET QoS** policy provides a means for applications to inform DDS of the urgency associated with transmitted data. The latency budget specifies the time period within which DDS must distribute the information. This time period starts from the moment the data is written by a publisher until it is available in the subscriber's data-cache ready for use by readers.

- The **TRANSPORT_PRIORITY QoS** policy allows applications to control the importance associated with a topic or with a topic instance, thus allowing a DDS implementation to prioritize more important data relative to less important data. These QoS policies help ensure that mission-critical information needed to reconstruct the shared operational picture is delivered in a timely manner.

OMG DDS: QoS Data Delivery

- The **PRESENTATION QoS** policy gives control on how changes to the information model are presented to subscribers. This QoS gives control of the ordering as well as the coherency of data updates. The scope at which it is applied is defined by the access scope, which can be one of **INSTANCE**, **TOPIC**, or **GROUP** level.

- The **RELIABILITY QoS** policy controls the level of reliability associated with data diffusion. Possible choices are **RELIABLE** and **BEST_EFFORT** distribution.

- The **PARTITION QoS** policy gives control over the association between DDS partitions (represented by a string name) and a specific instance of a publisher/subscriber. This association provides DDS implementations with an abstraction that allow segregation of traffic generated by different partitions, thereby improving overall system scalability and performance.

- The **DESTINATION_ORDER QoS** policy controls the order of changes made by publishers to some instance of a given topic. DDS allows the **ordering** of different changes according to source or destination timestamps.

- The **OWNERSHIP QoS** policy controls which writer ‘owns’ the write-access to a topic when there are multiple writers and ownership is **EXCLUSIVE**. Only the writer with the highest **OWNERSHIP_STRENGTH** can publish the data. If the **OWNERSHIP** QoS policy value is shared, multiple writers can concurrently update a topic. OWNERSHIP thus helps to manage replicated publishers of the same data.

Overview of OMG DDS Features

QoS-enabled information management for real-time applications

Conventional SOA middleware technologies are poorly suited for these types of systems due to insufficient QoS support

- Standard-based pub/sub
- Very low latency & predictable data dissemination
- Support for >>100K messages/sec
- Stability under overload condition
- Scalability
- Control over latency/throughput tradeoffs
- High performance persistence
- Event processing

Alternative: CORBA eine Standardmiddleware

CORBA steht für ***Common Object Request Broker Architecture*** und ist ein **Standard** für eine **objektorientierte Middleware-Architektur**. Sie wurde von der ***Object Management Group*** **(OMG)** entwickelt und wird von dieser Gruppe auch weiter gepflegt. Ziel der OMG ist die Entwicklung von Standards zur Kooperation verschiedenster heterogener Anwendungen über Rechnernetze, um verteilte, offene Systeme zu schaffen. CORBA ist nicht echtzeitfähig.

- **OMA**: ***Object Management Architecture***. Dies ist die **Architektur eines verteilten objektorientierten Systems** auf der Basis von CORBA.
- **ORB**: ***Object Request Broker***. Dies ist der **Kern der OMA**. Aufgabe des ORB ist es, die Anforderungen von verteilten Objekten weiterzuleiten und zu koordinieren.