

Einführung in die Java / Jakarta Enterprise Edition

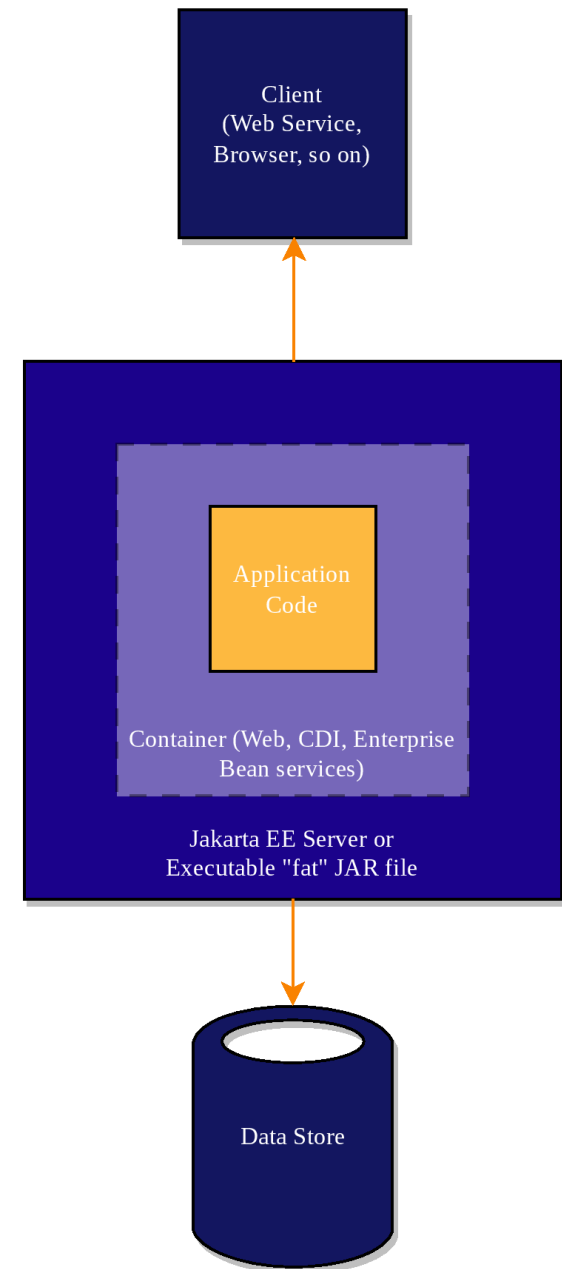
Kapitel 6 – Architektur & Design

6.1

BASISARCHITEKTUR

Basisdesign – Wiederholung 3 Schicht

- Standardarchitektur für viele Anwendungen
- z.B. Self-Contained-Systems



https://jakarta.ee/learn/docs/jakartaee-tutorial/current/intro/overview/overview.html#_footnotedef_3

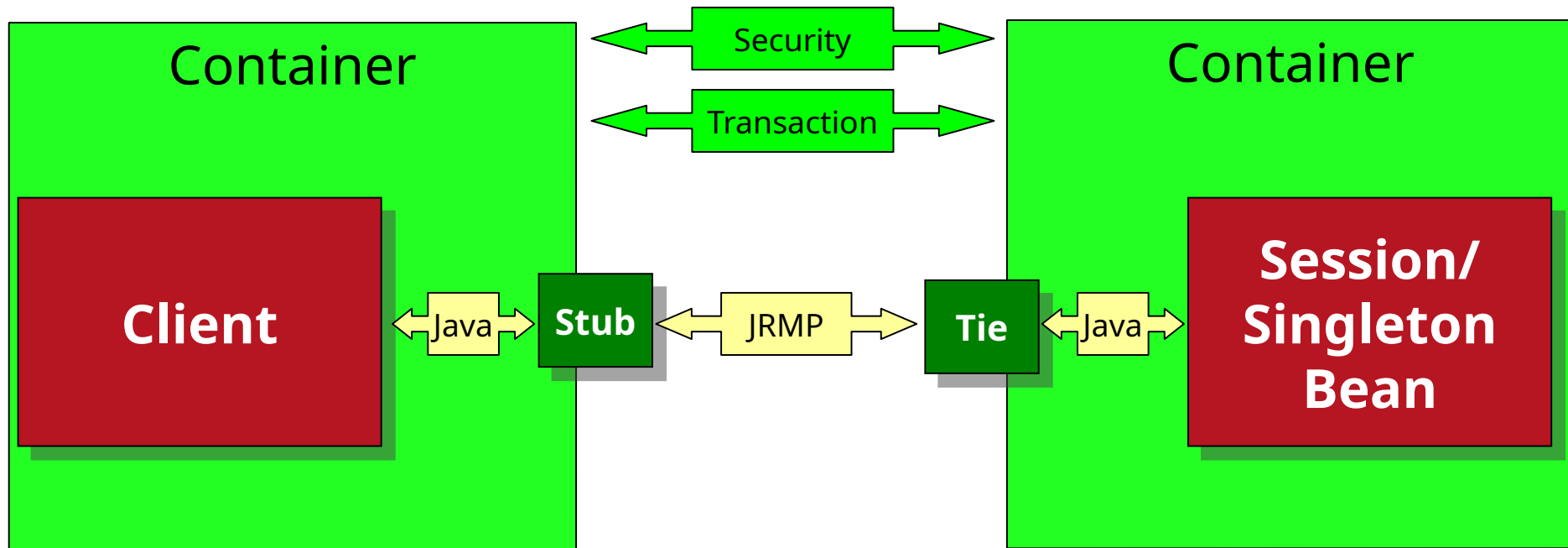
6.2

VERTEILTE ANWENDUNGEN

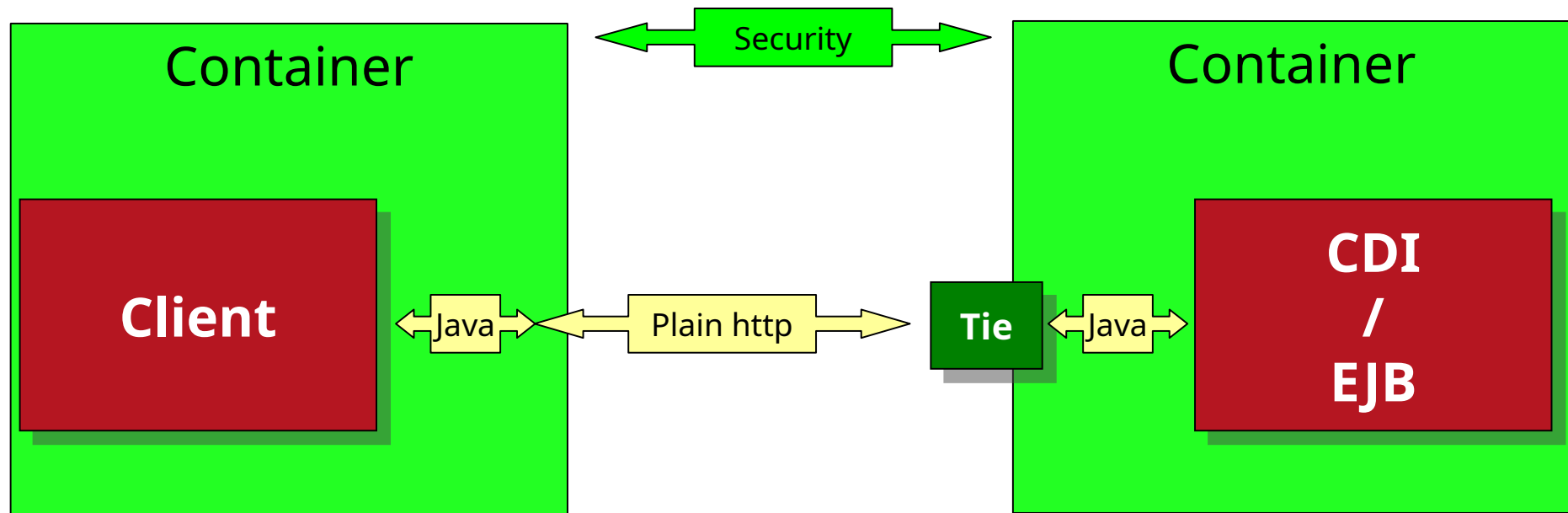
Remote Method Invocation

- Kommunikation zwischen zwei virtuellen Maschinen
 - Sehr hochwertig
 - Objekt-orientiert
 - Datenaustausch mit Serialisierten Java-Objekten
 - Sehr effizient und auf Java optimiert
 - Distributed Garbage Collection
- Definition des Server-APIs über ein Java-Interface
- Bereitstellung über eine Stateless oder Stateful SessionBean
 - Zusätzliche Annotation: `@Remote`

Java Remote Method Invocation

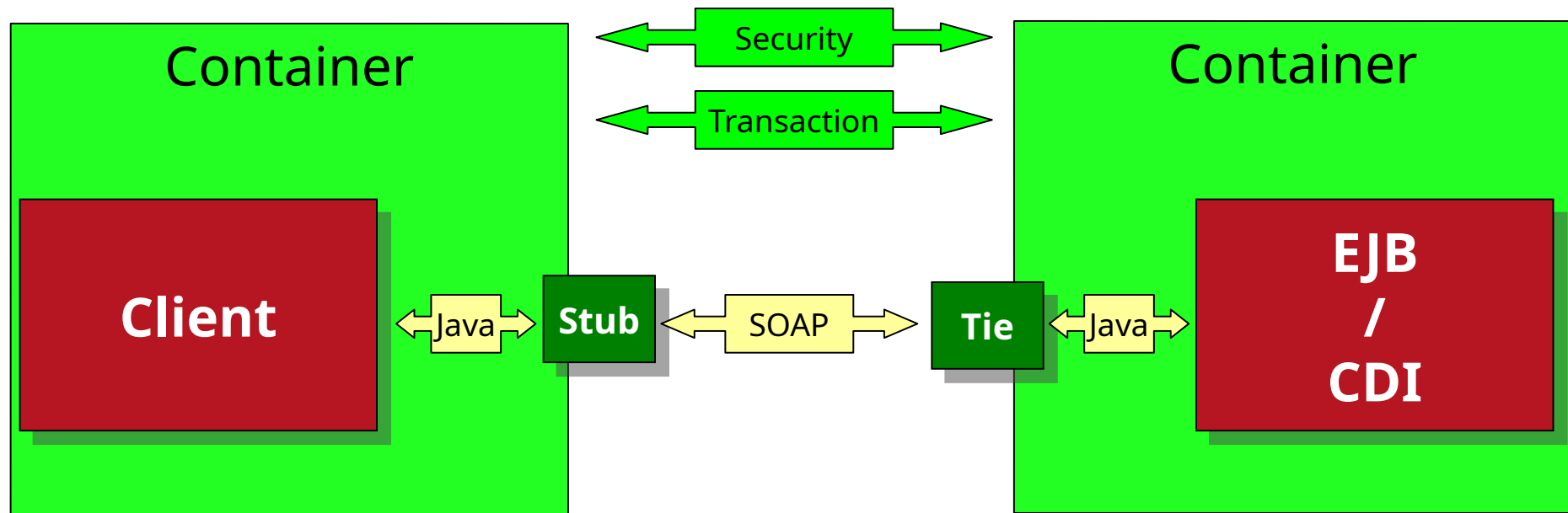


- Interoperable Kommunikation heterogener Plattformen durch Austausch von Standard-Dokumenten
 - Definiert über MIME-Types
- Der Server stellt für den Service eine Reihe von URL-Pfaden zur Verfügung
 - Als Operationen werden die http-Methoden benutzt
 - GET
 - PUT
 - POST
 - ...
- Die Bereitstellung erfolgt durch eine annotierte Klasse
 - CDI
 - EJB
 - Annotationen aus JAX-RS, z.B. @Path, @Produces, @Consumes



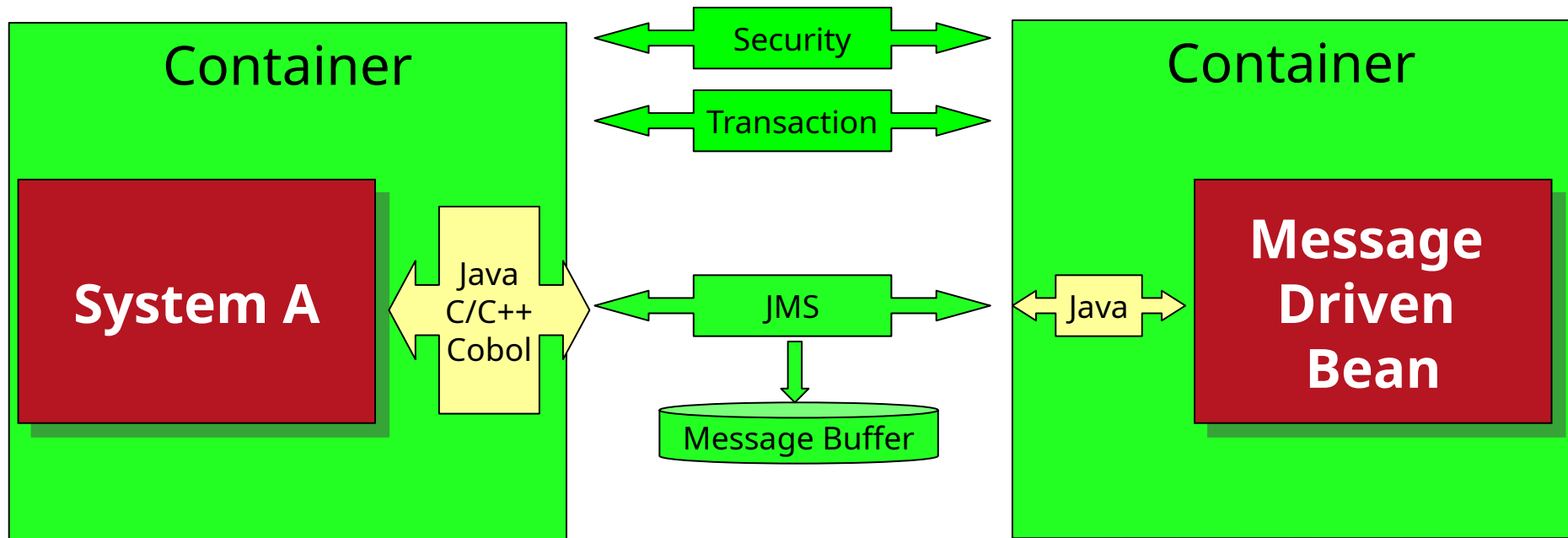
Security und die Propagierung einer Transaktion sind nicht verpflichtend unterstützt!

- Kommunikation heterogener Plattformen durch Austausch von XML-Dokumenten
 - Dies sind die SOAP-Envelopes
 - Dies sind XML-Dokumente
- Die Beschreibung des Services erfolgt über eine Schnittstelle formuliert in der Web Services Description Language (WSDL)
 - Contract First
- Alternativ hierzu kann die WSDL auch aus einer Java-Klasse erzeugt werden
 - Code First
 - Hierzu werden eine Vielzahl von Annotationen benutzt
 - JAX-WS, z.B. @WebService
 - JAXB, z.B. @XmlElement
- Über eine Code-Generierung werden Server-Rümpfe und Client-Stubs erzeugt
- Die Bereitstellung erfolgt über eine annotierte Klasse
 - CDI oder EJB



Security und die Propagierung einer Transaktion sind nicht verpflichtend unterstützt!

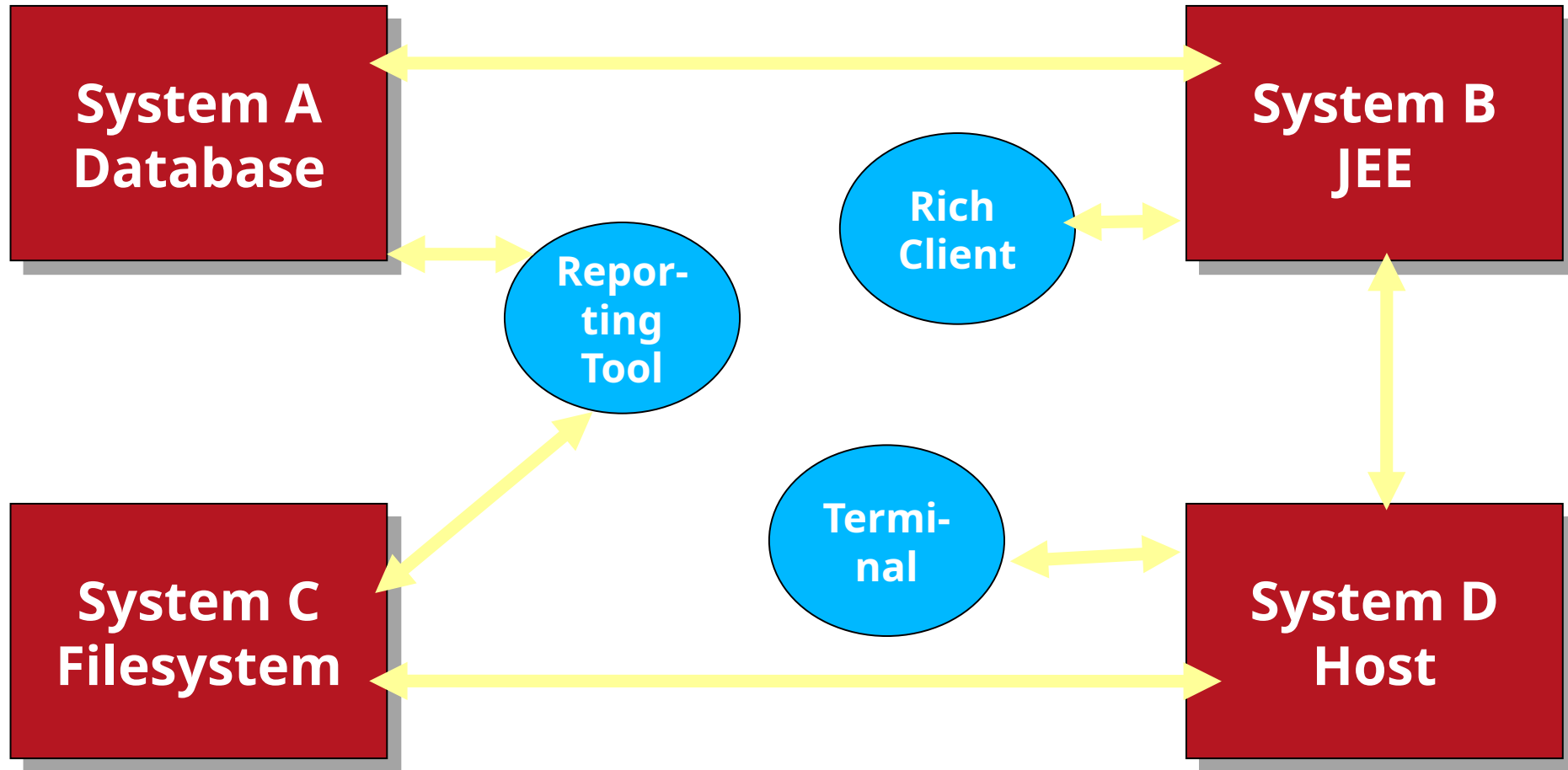
- Notwendig hierzu ist ein Messaging System
 - dazu kann der im Applikationsserver vorhandene genutzt werden
 - häufiger werden jedoch externe Messaging Systeme genutzt
 - z.B. Apache ActiveMQ
- Die Kommunikation erfolgt über das Versenden von Nachrichten an Destinations
 - Sender und Empfänger vereinbaren hierzu eine Nachrichten-Format
 - Die Validierung der Nachrichten bleibt größtenteils Aufgabe der Anwendung
- Bereitstellung über einen MessageListener oder eine MessageDrivenBean
 - Zusätzliche Annotation: @MessageDriven

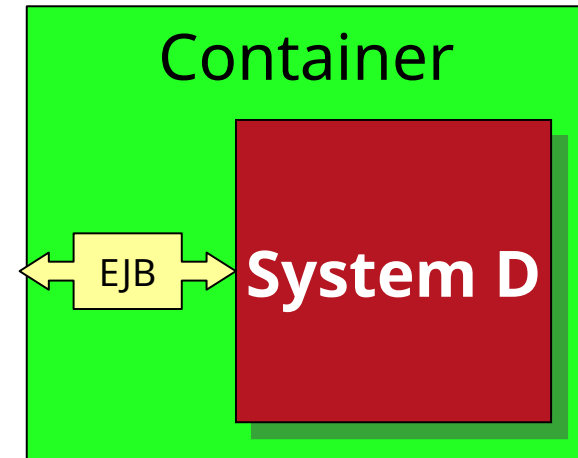
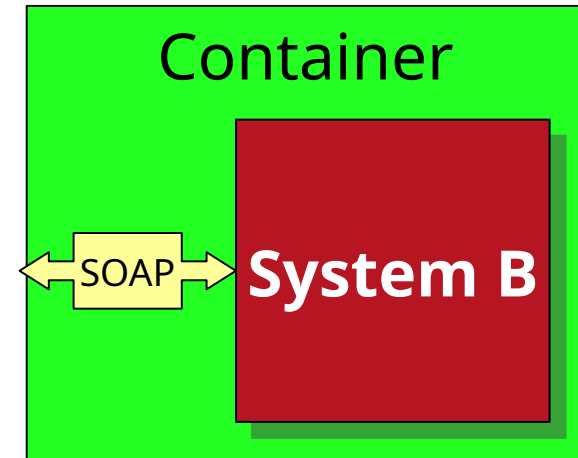
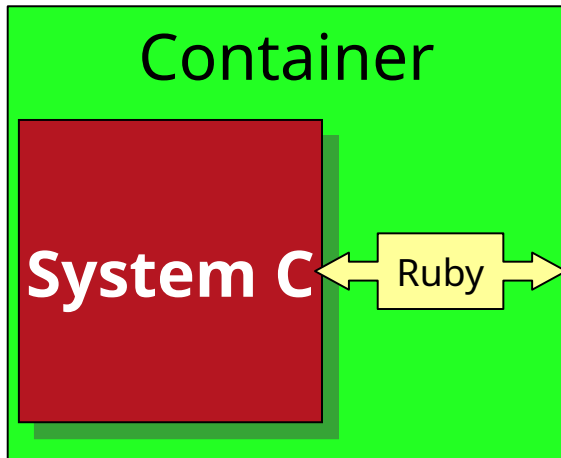
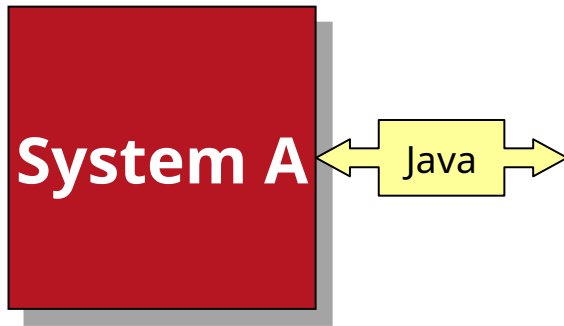


6.3

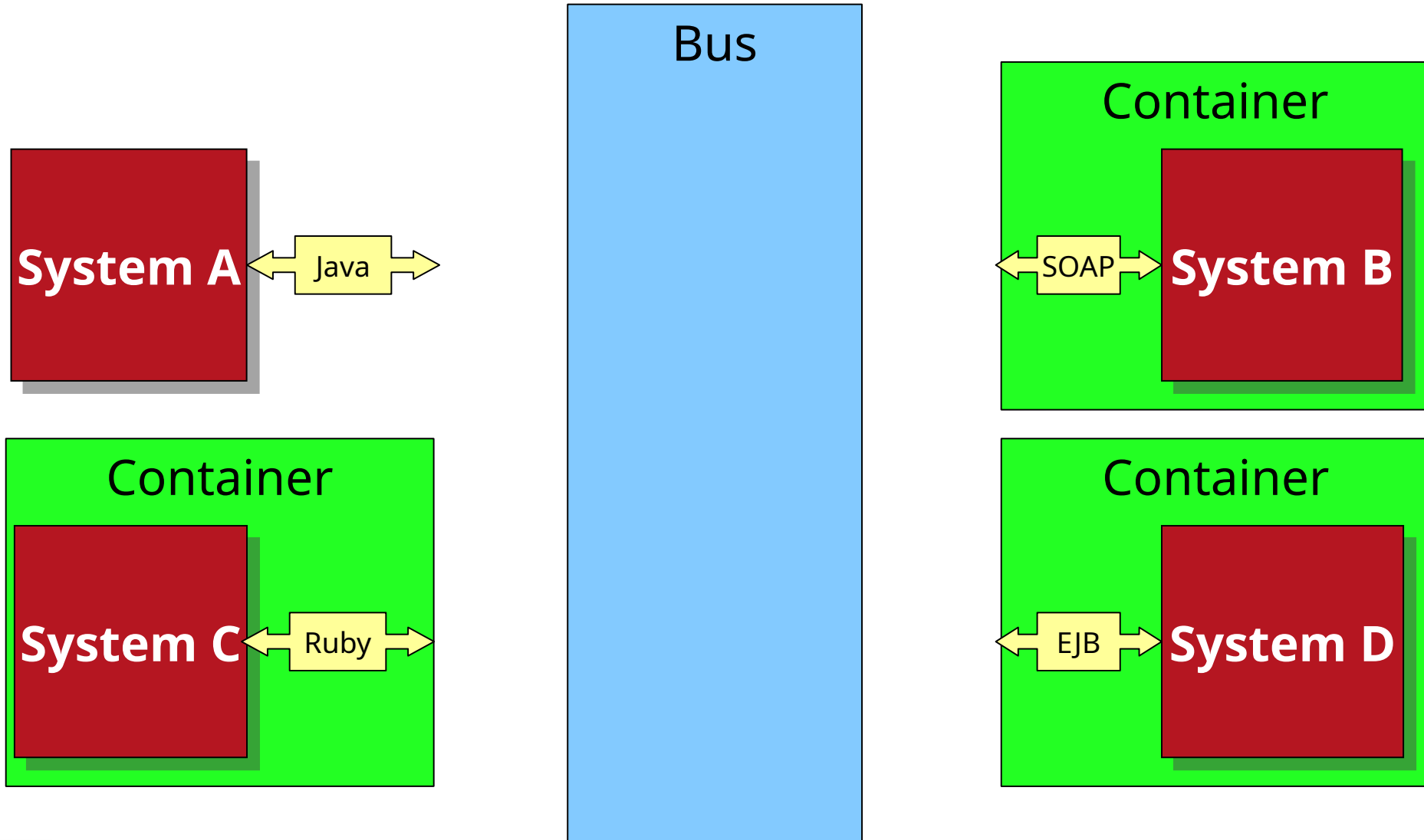
SERVICE ORIENTED ARCHITECTURE

- Service Oriented Architecture ist ein Begriff für eine Best Practice
 - Keine Spezifikation!
 - Keine Plattform!
 - Kein Komponentenmodell!
 - Kein Programmiermodell!
- Häufig auch noch verwechselt mit Web Services
 - Diese sind jedoch nur eine Möglichkeit von vielen, Services zu definieren und aufzurufen

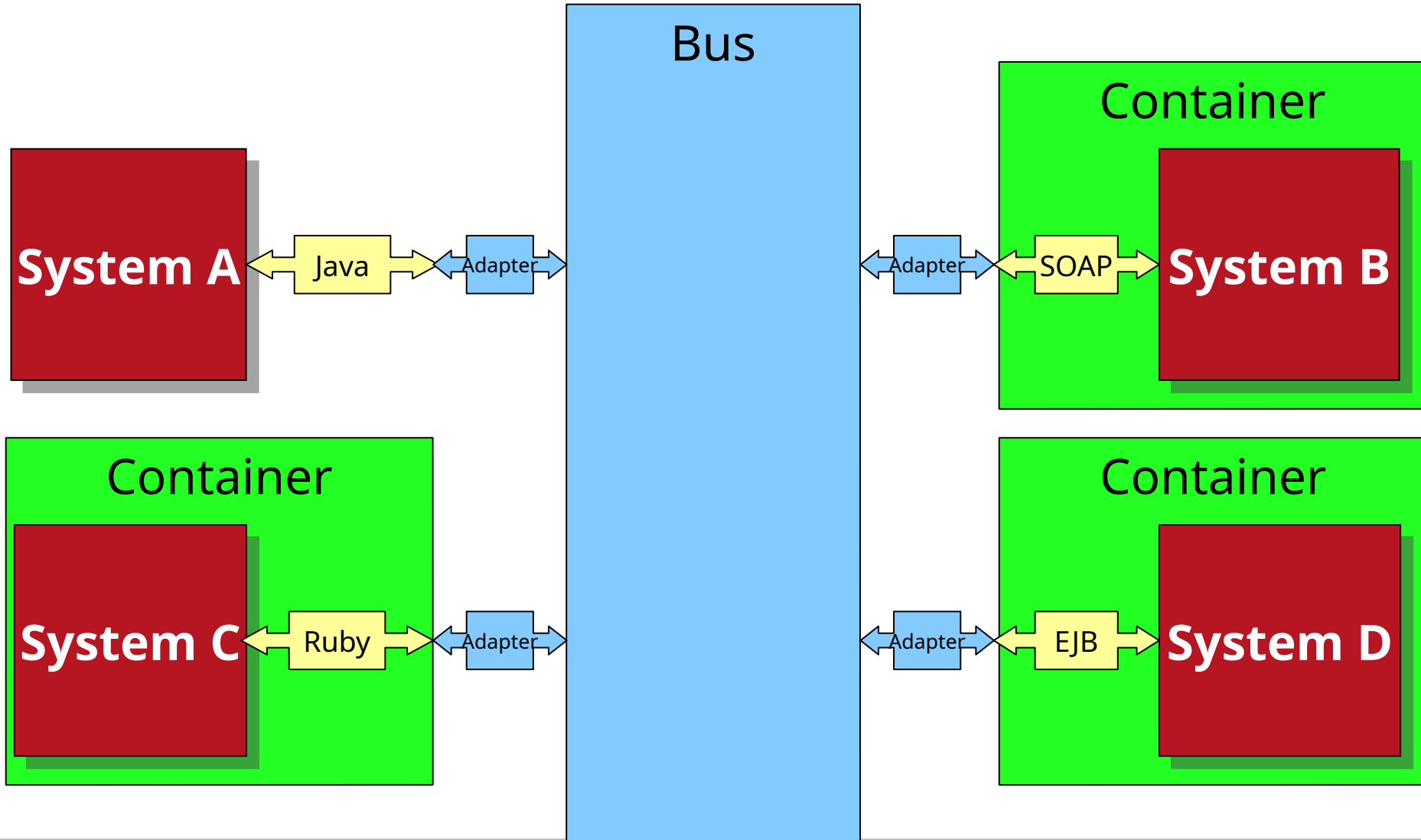


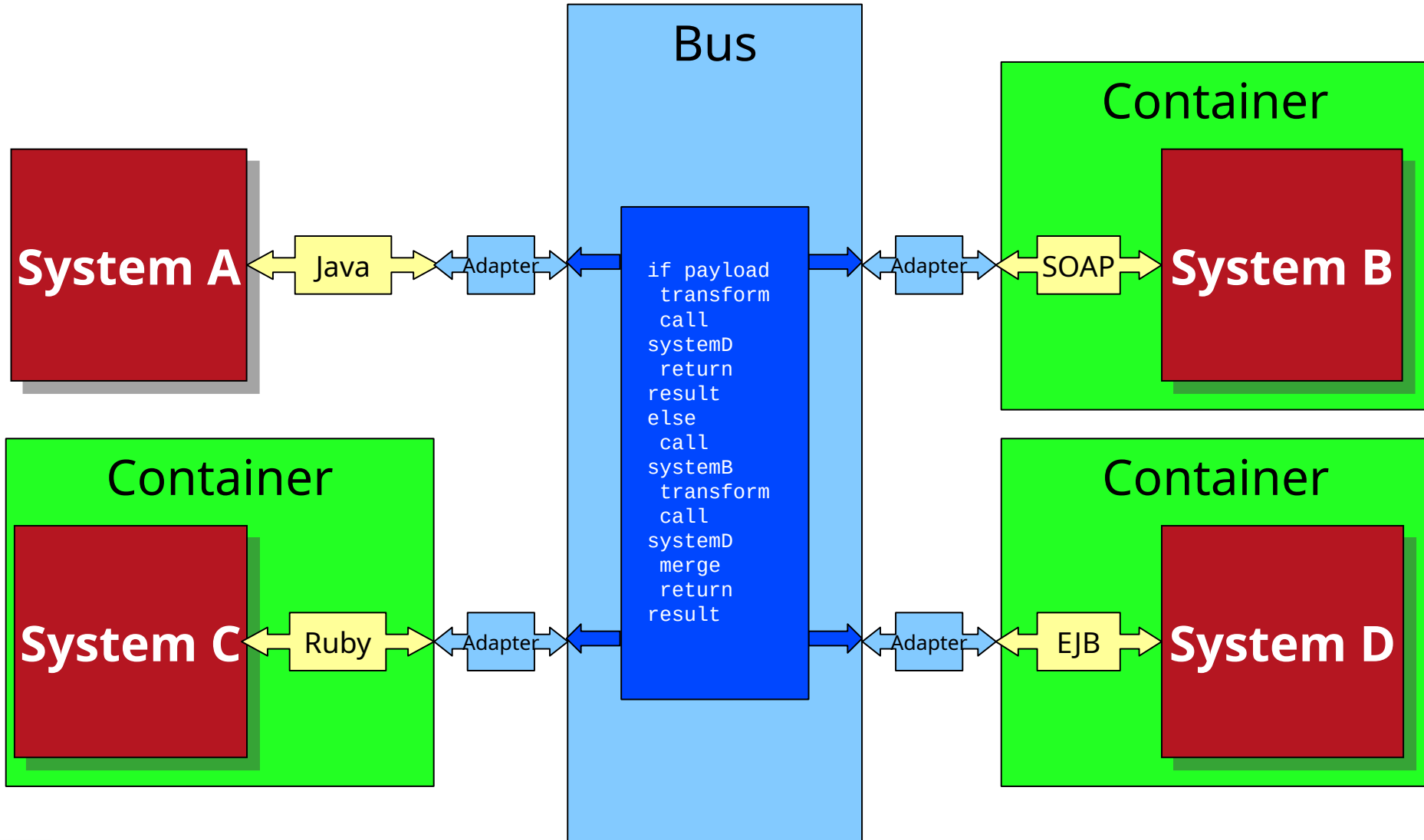


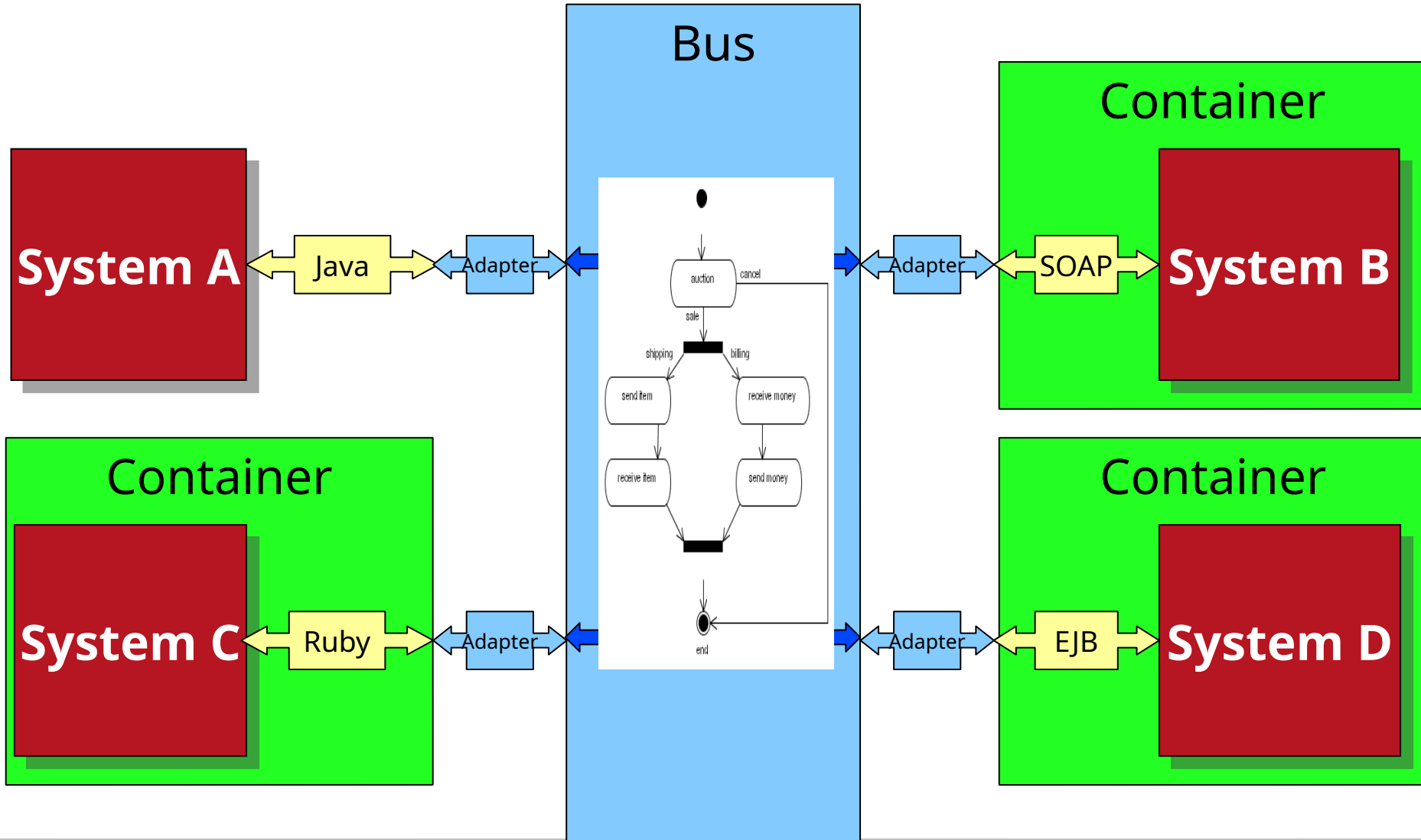
Einführung eines Bus-Systems



Kopplung über Adapter

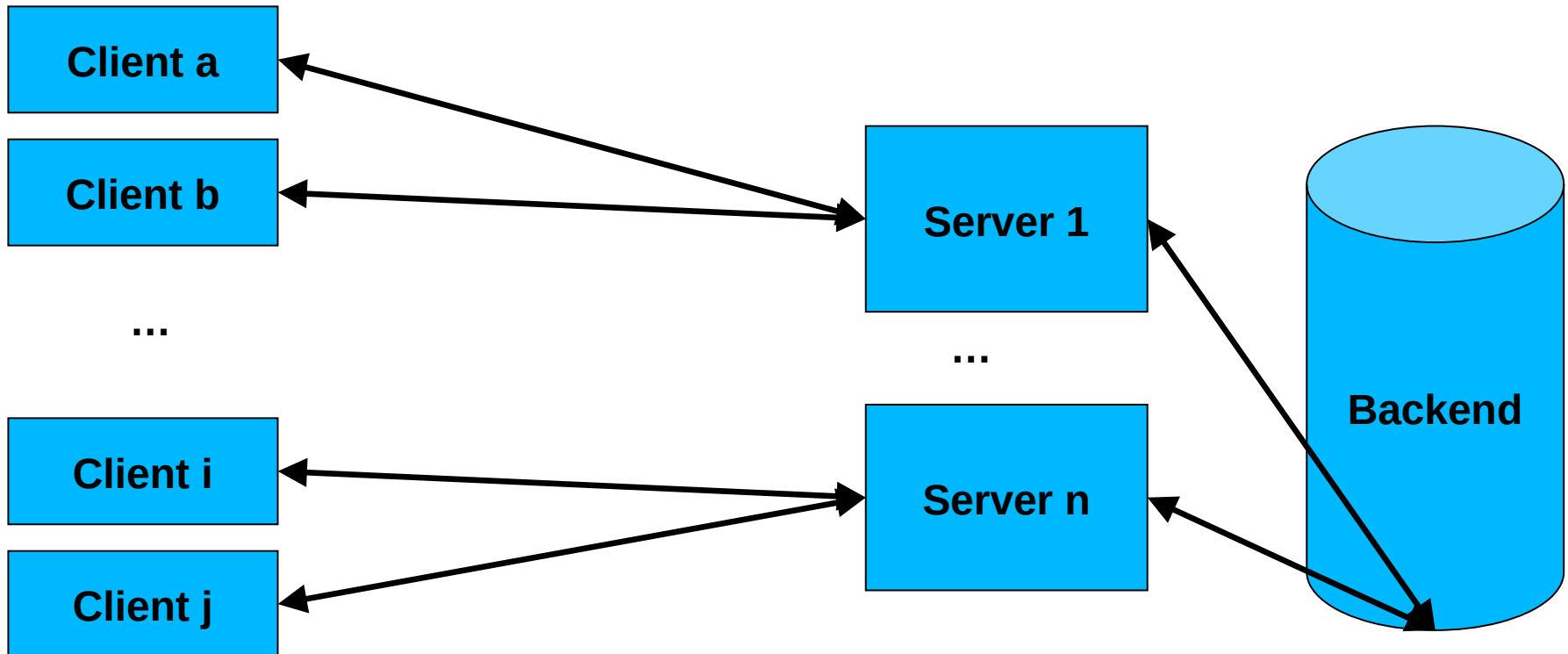


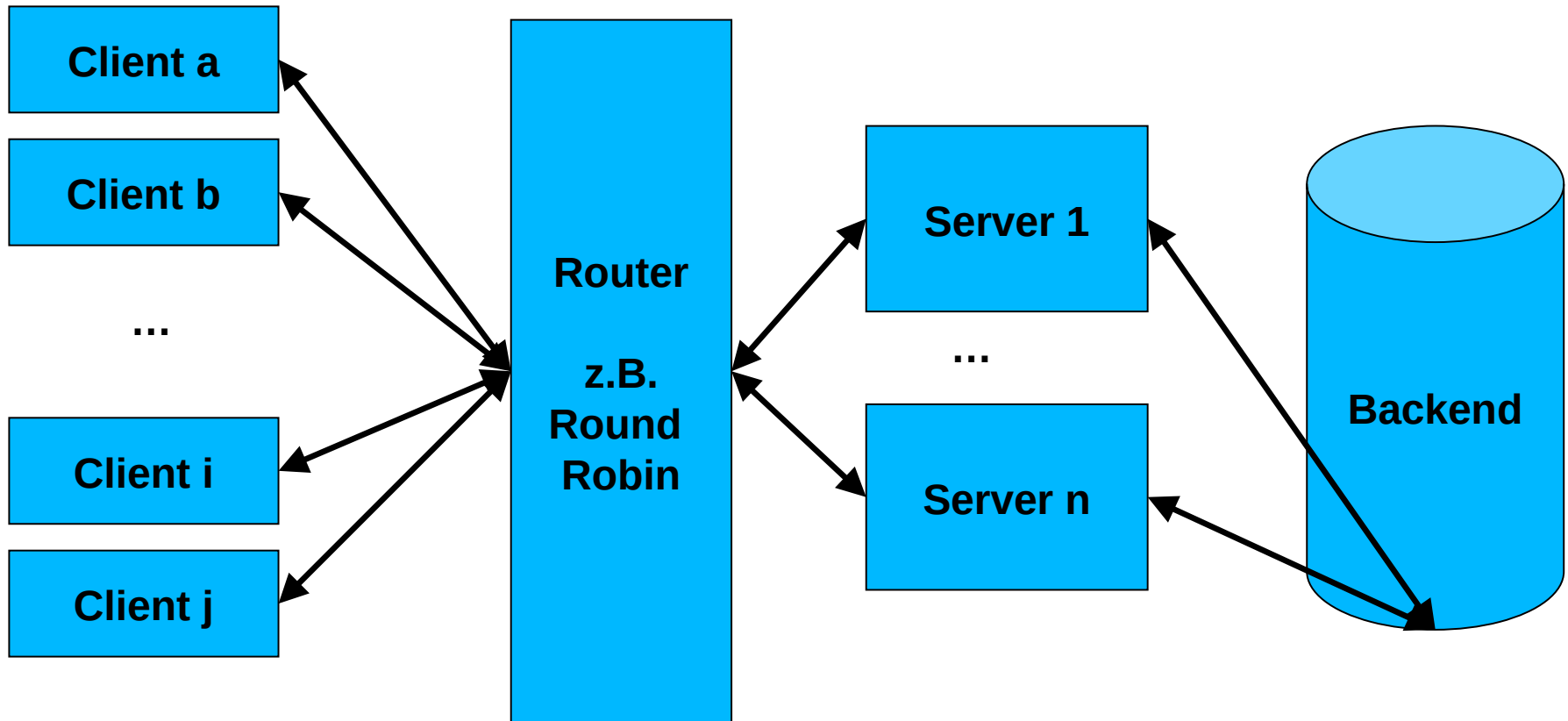




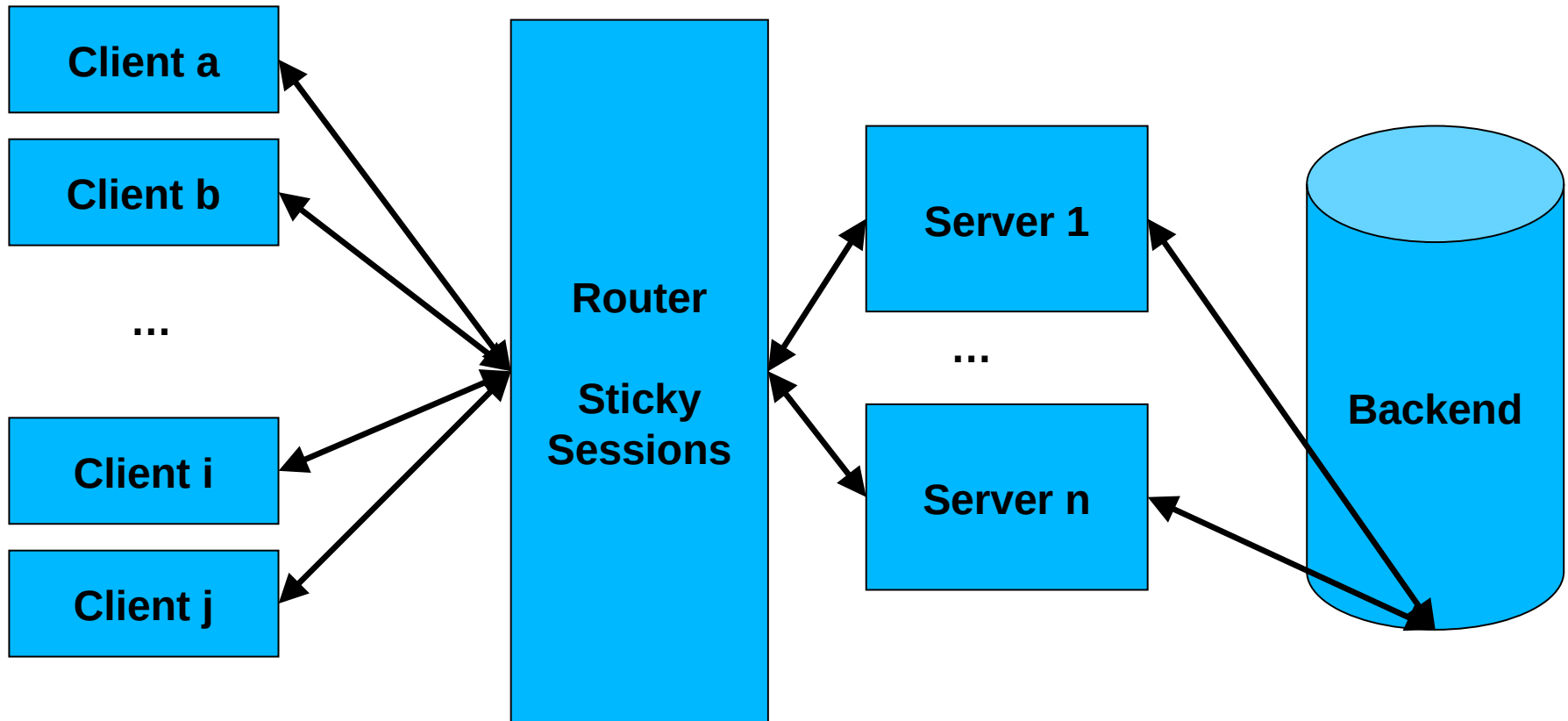
6.4

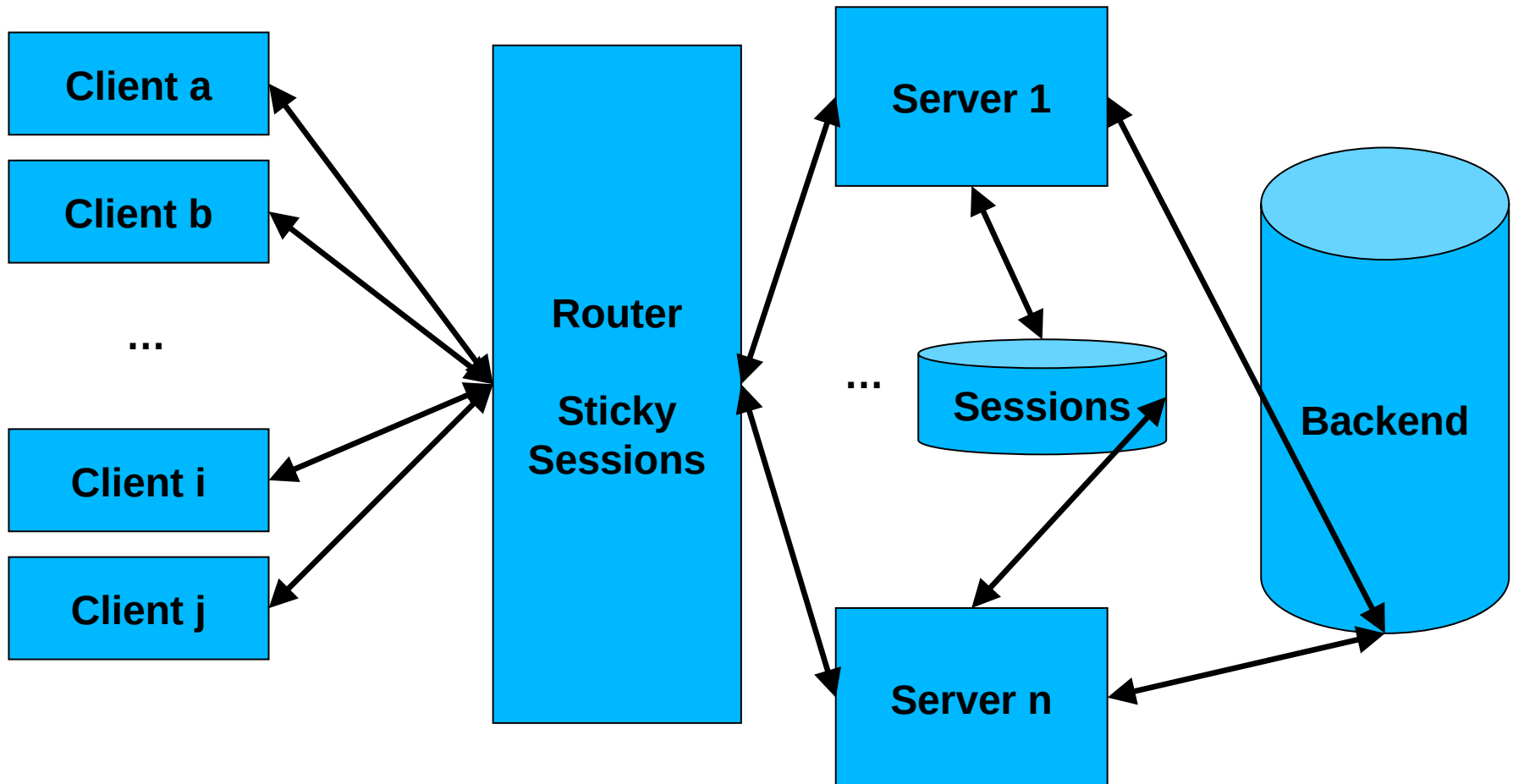
CLUSTER-SYSTEME





Zustandsbehaftete Anwendungen ohne Ausfallsicherheit





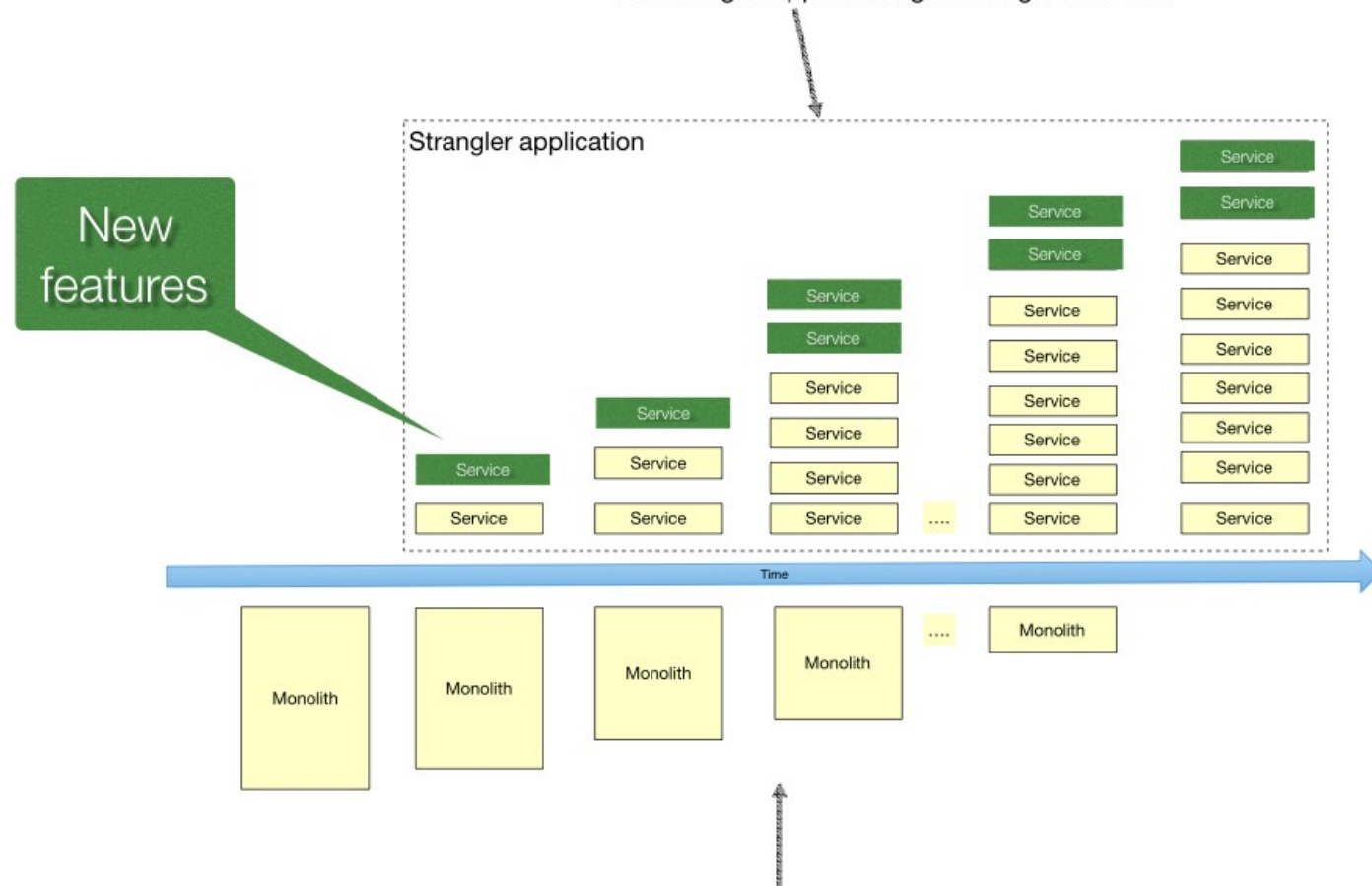
6.5

CONTAINER & MICROSERVICE ARCHITEKTUREN

Microservices, Monolyth neue und alte Welt – Welche Technologie ist die richtige?

Strangling the monolith

The strangler application grows larger over time



The monolith shrinks over time

Microservices, Monolith neue und alte Welt – Welche Technologie ist die richtige?

- <https://microservices.io/patterns/index.html>
- <https://microservices.io/patterns/monolithic.html>
- <https://microservices.io/patterns/microservices.html>
- <https://microservices.io/post/architecture/2023/07/31/how-modular-can-your-monolith-go-part-1.html>
- <https://microservices.io/articles/index.html> - Dark Energy and Dark Matter
- <https://microservices.io/patterns/service-template.html>

