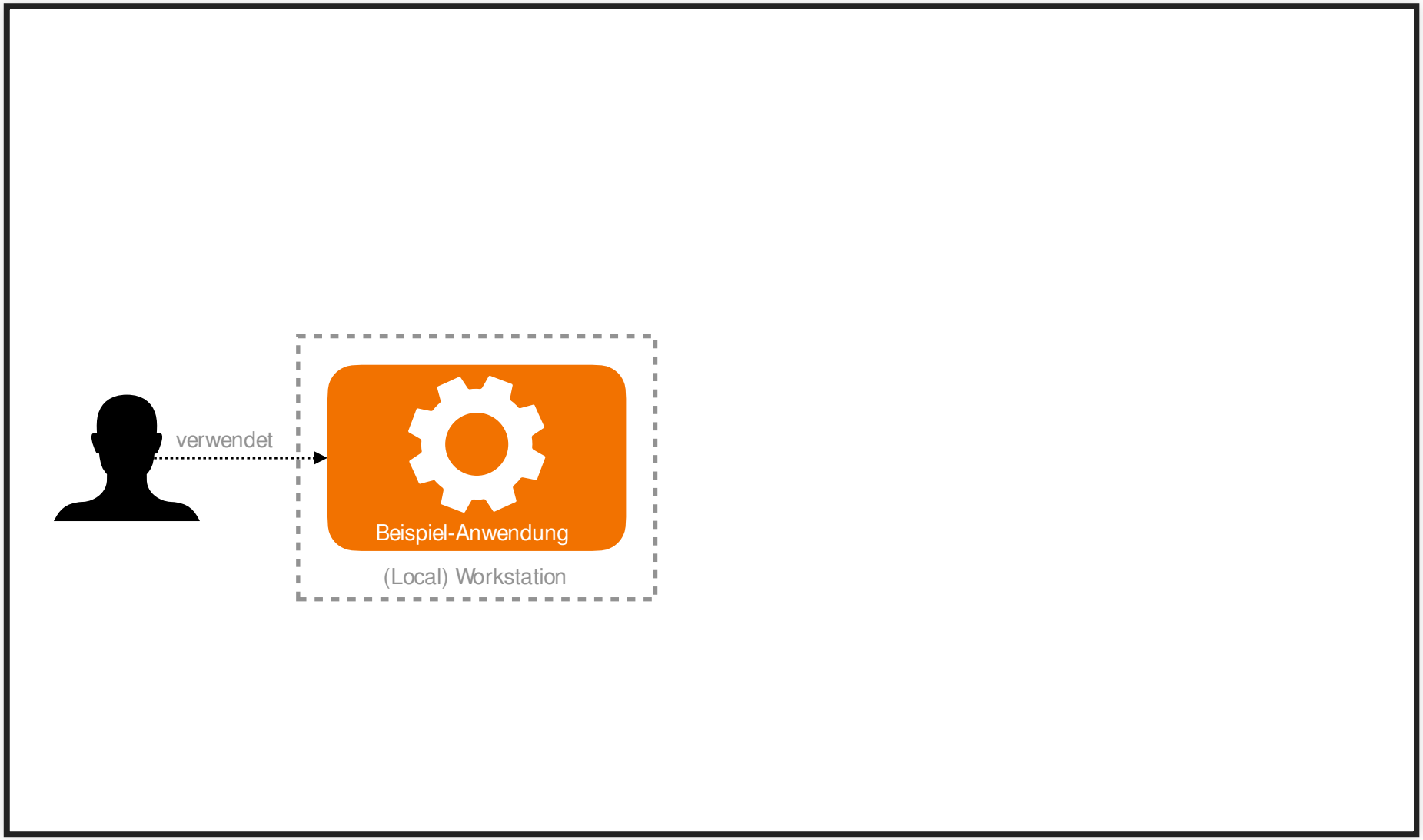
KAPITEL 1: EINLEITUNG

LERNZIELE

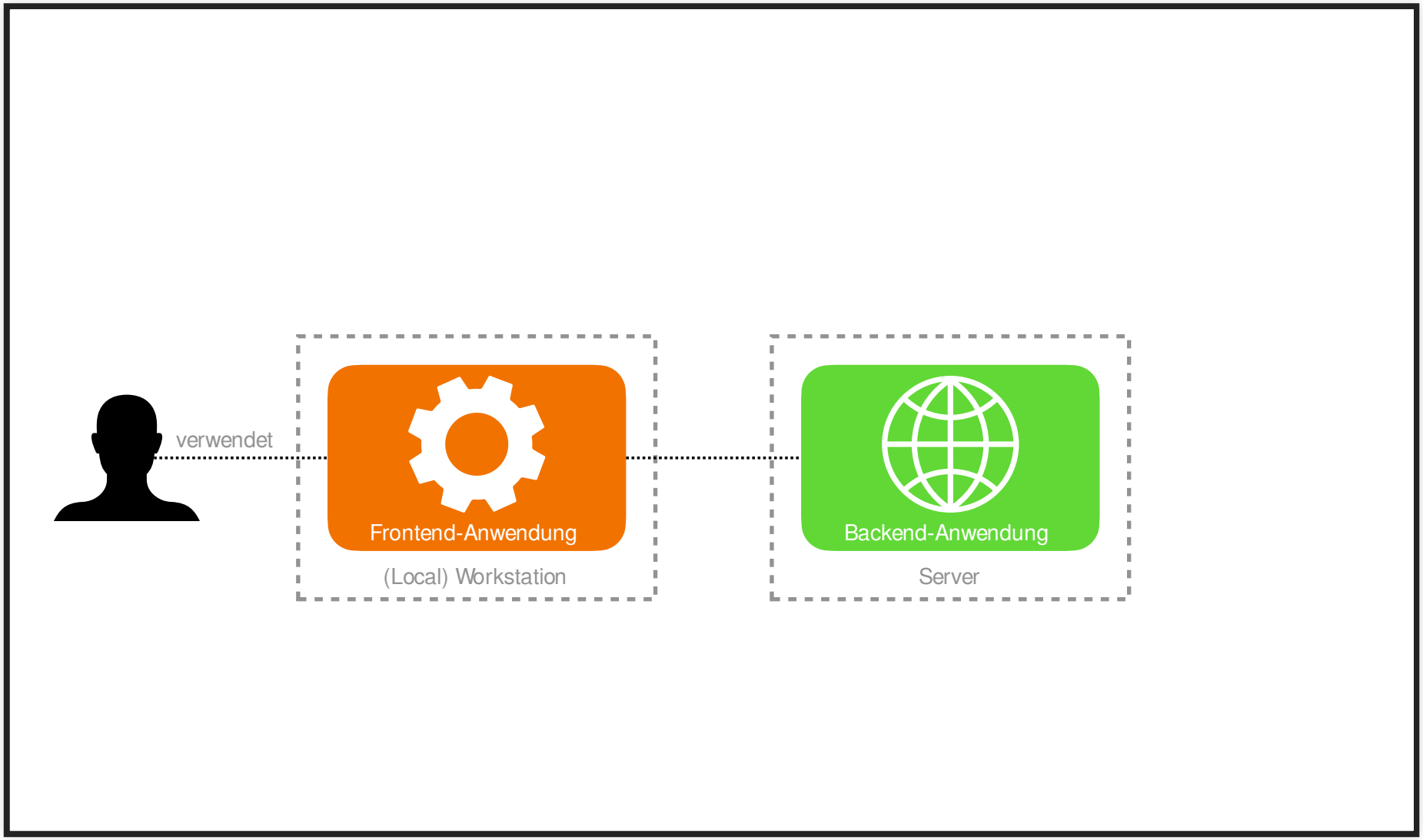
- Vorteile und Herausforderungen von verteilten Systemen nennen
- Typische Architekturmodelle in verteilten Systemen differenzieren
- Arten von Skalierbarkeit beschreiben

1.1 MOTIVATION

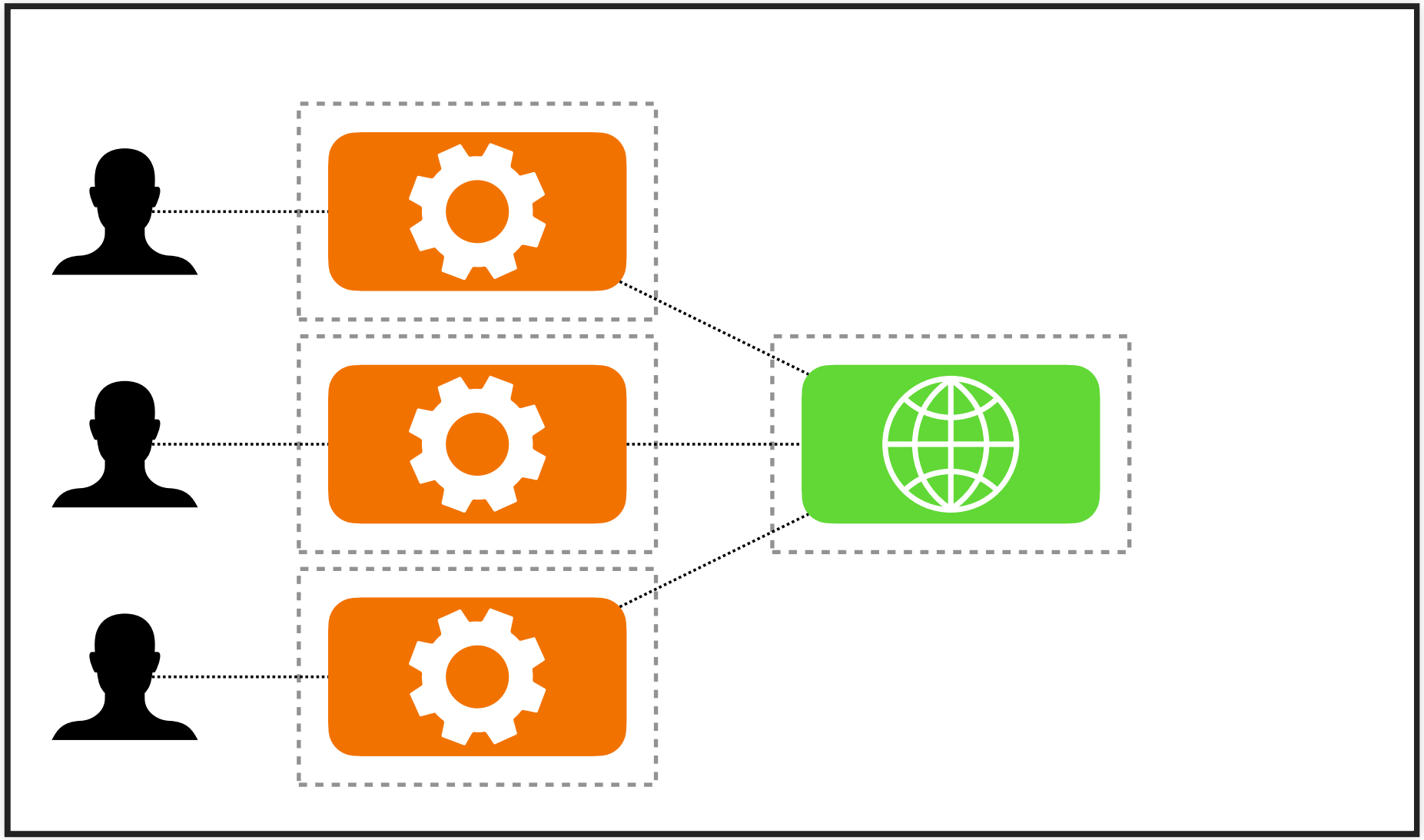
Ausgangssituation: Ein Nutzer nutzt eine Anwendung lokal



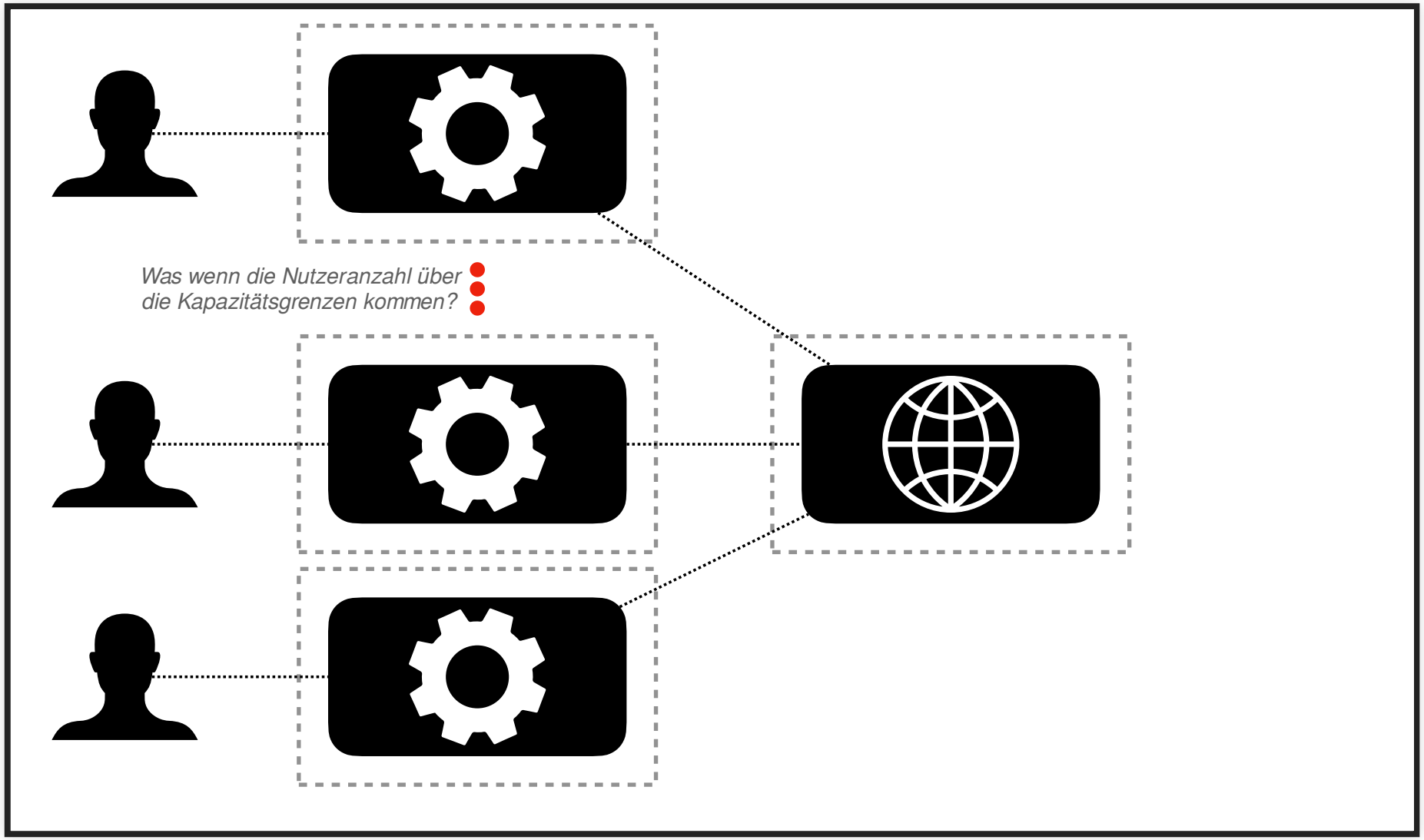
Was wenn diese Anwendung ein weiteres System nutzt?



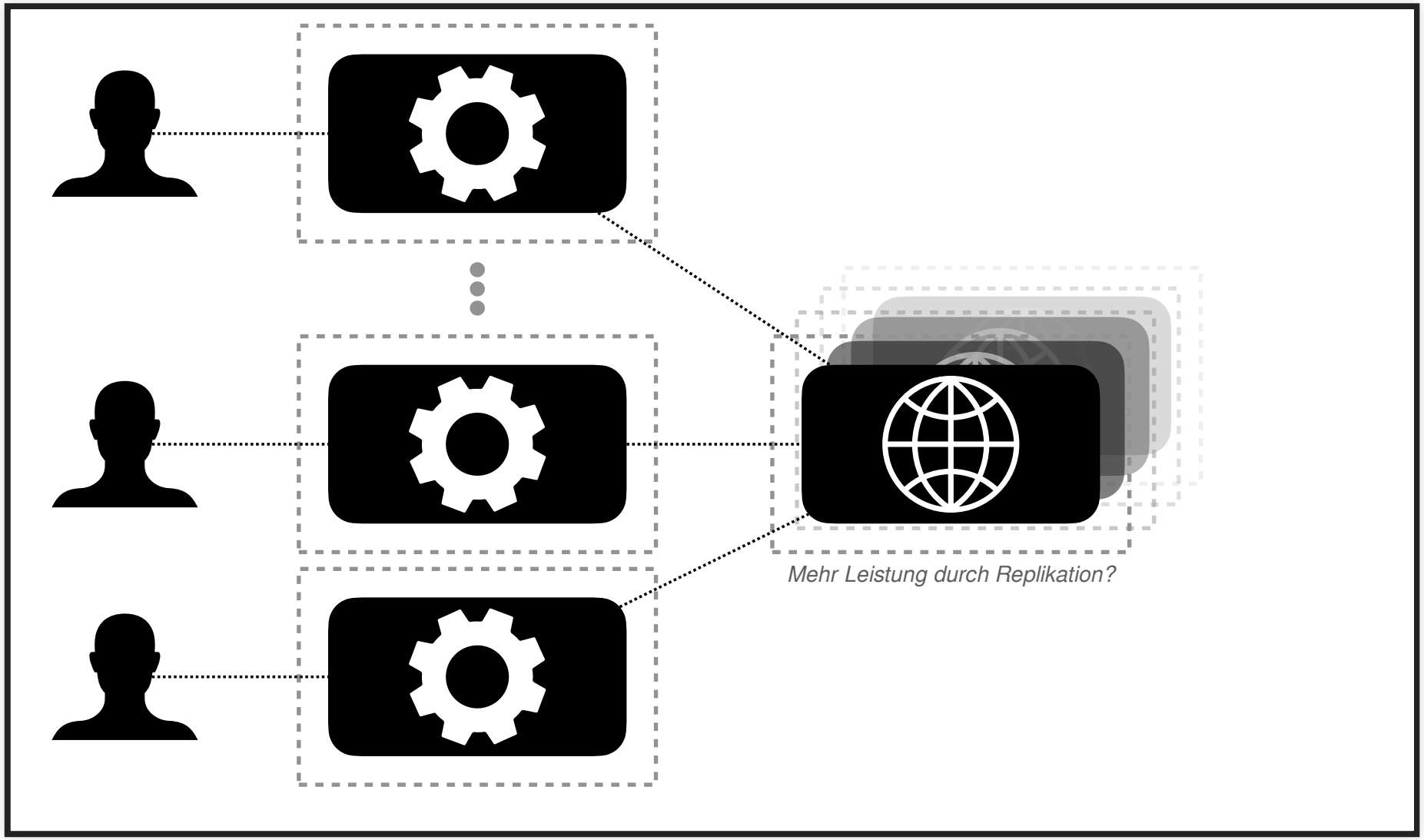
... und dieses System mehreren Nutzern zur Verfügung steht?



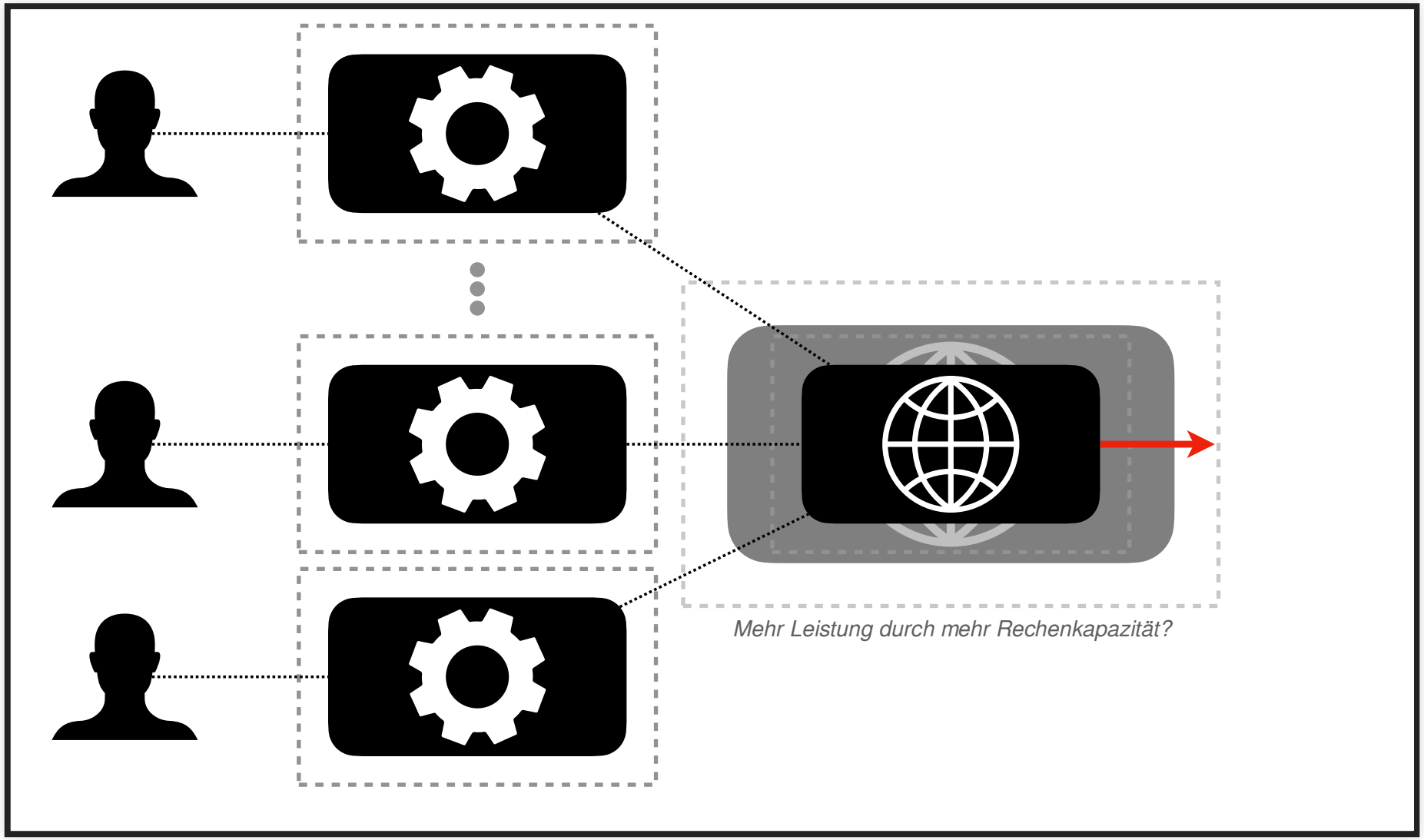
... und die Anzahl der Nutzer steigt?



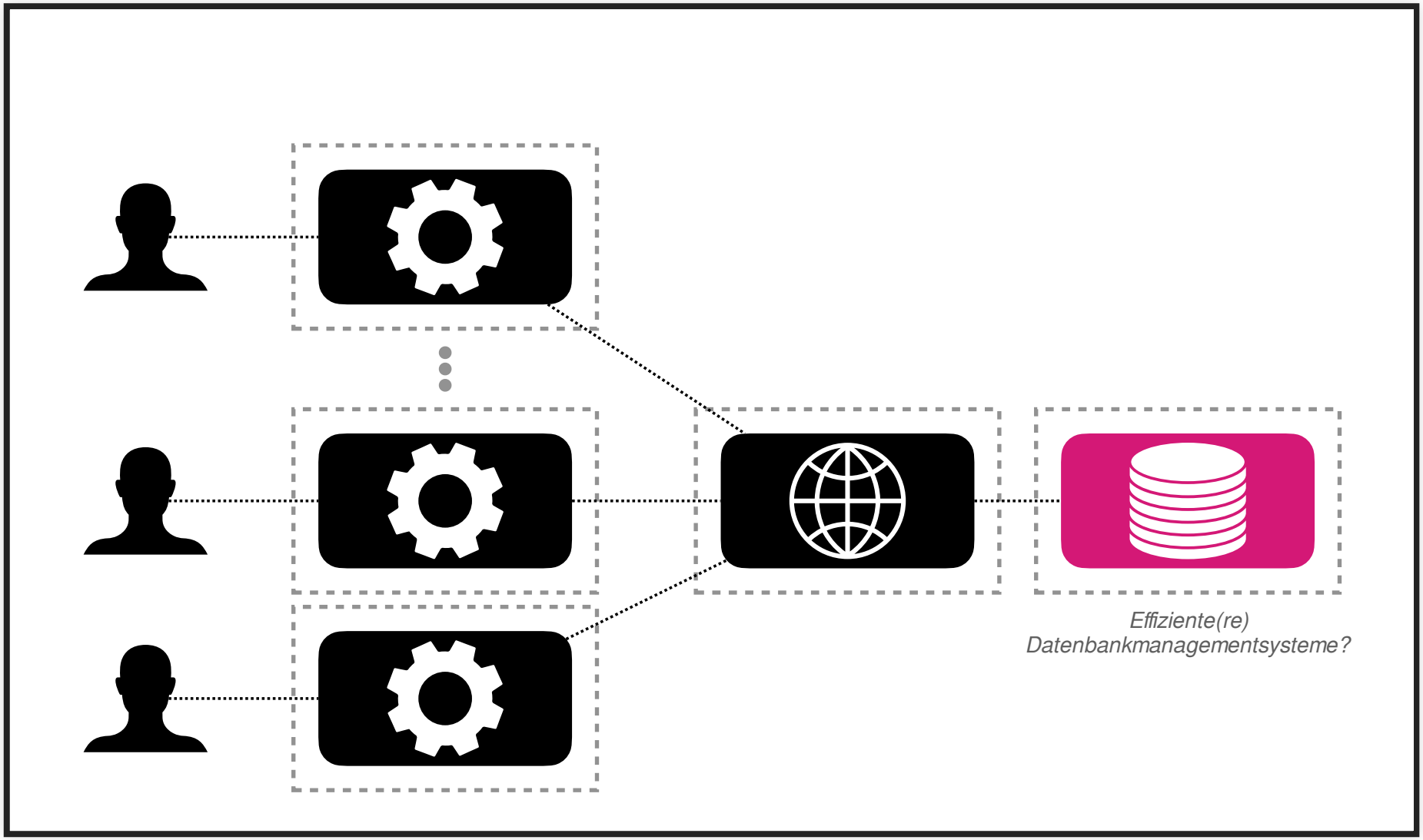
Könnte man das System vervielfältigen, um die steigende Last zu bedienen?



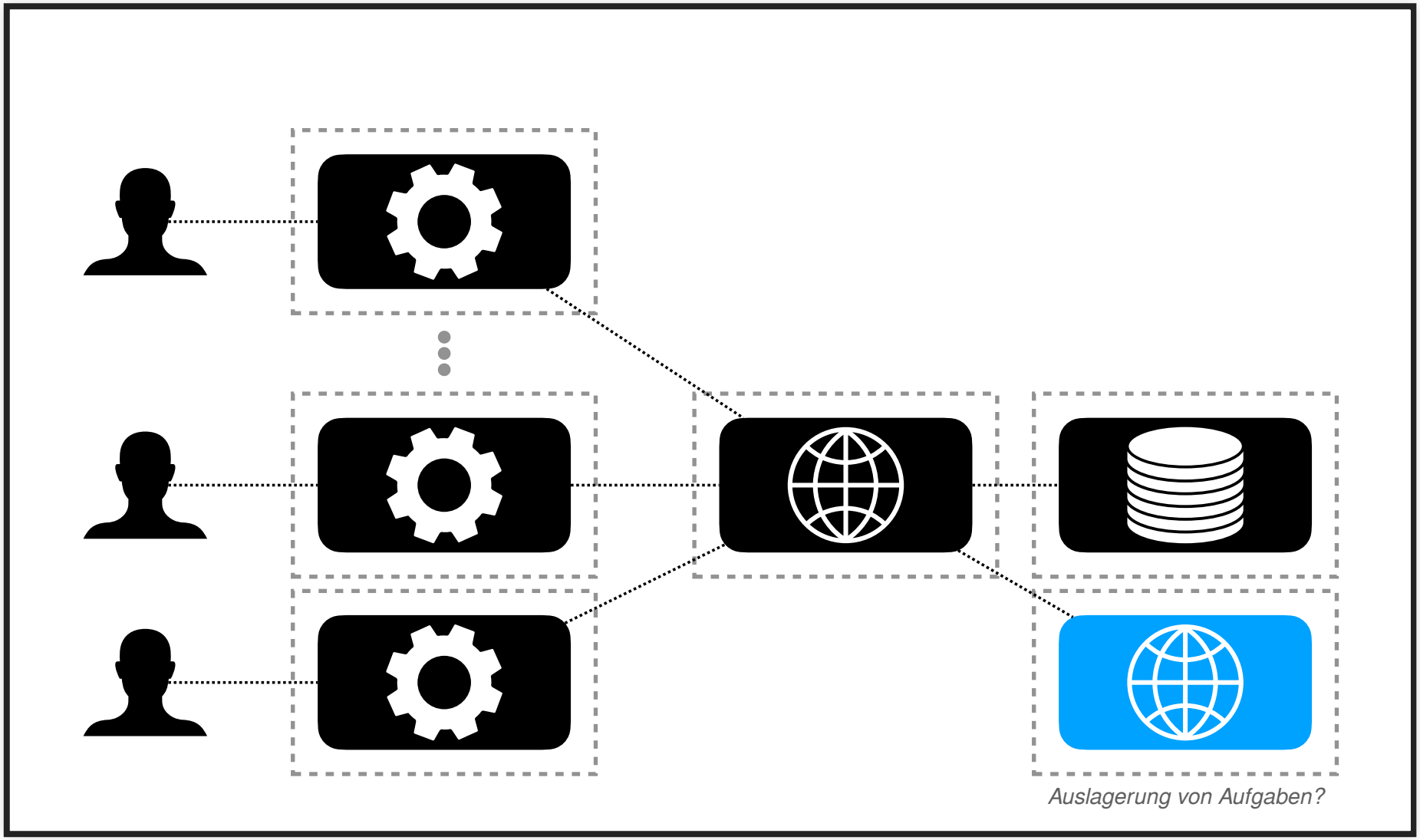
Könnte man das System größer dimensionieren?



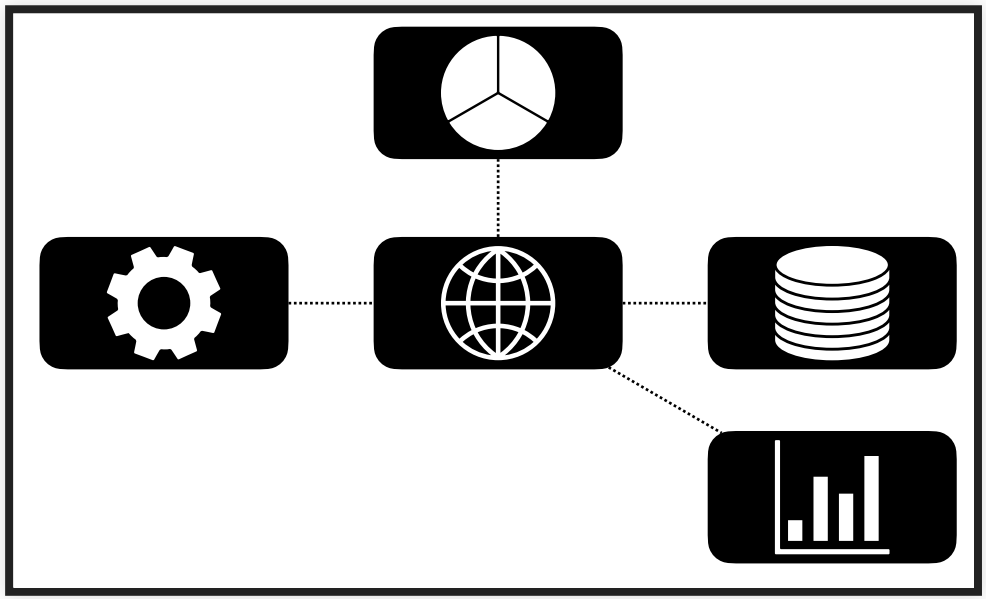
Lassen sich Systembestandteile wie eine Datenbanken auslagern?



Lassen sich abgrenzbare Aufgaben auslagern?



1.2 VERTEILTE (SOFTWARE-) SYSTEME



Ein verteiltes System ist ein System mit mehreren Komponenten auf verschiedenen Rechnern, die miteinander kommunizieren und ihre Aktionen koordinieren, um für den Endbenutzer als ein einziges kohärentes System zu erscheinen.

Vorteile von verteilten Systemen

- **Horizontale Skalierbarkeit**: Datenverarbeitung erfolgt unabhängig auf jedem Knoten, womit es im Allgemeinen kostengünstiger ist bei Bedarf weitere Knoten und Funktionen hinzuzufügen
- **Zuverlässigkeit**: Verteilte Systeme mit einem ausreichendem Maß an Fehlertoleranz können den Ausfall einzelner Recheninstanzen kompensieren
- **Leistung**: Aufgeteilte Arbeit und die Möglichkeit die Auslastung je Recheninstanz anzupassen erlauben eine hohe Effizienz

Herausforderungen von verteilten Systemen

- **Scheduling**: Ein verteiltes System muss entscheiden, welche Aufträge wann und wo ausgeführt werden sollen. Schedulers haben letztlich ihre Grenzen, was zu einer unzureichenden Auslastung des verteilten Systems und Unvorhersehbarkeiten zur Laufzeiten führt.
- **Latency**: Je weiter Ihr System verteilt ist, desto mehr Latenz kann bei der Kommunikation auftreten. Dies führt häufig dazu, dass Teams Kompromisse zwischen Verfügbarkeit, Konsistenz und Latenz eingehen müssen.
- **Beobachtbarkeit**: Das Erfassen, Verarbeiten, Darstellen und Überwachen von (Hardware-) Nutzungsmetriken für große Cluster ist eine große Herausforderung.

Typische Architekturmodelle in verteilten Systemen

- **Client-Server**: Clients fordern Daten beim Server an, formatieren sie und zeigen sie dem Endbenutzer an. Der Endbenutzer kann auch eine Änderung auf der Client-Seite vornehmen und sie an den Server zurückgeben, um sie dauerhaft zu persistieren.
- **Dreischichtige Architektur**: Informationen über den Client werden in einer mittleren Schicht und nicht auf dem Client gespeichert, um die Anwendungsbereitstellung zu vereinfachen (zum Beispiel bei Webanwendungen).

- **n-tier**: Anwendung wird in mehreren Schichten aufgeteilt. Wird in der Regel verwendet, wenn eine Anwendung oder ein Server Anfragen an zusätzliche Unternehmensdienste im Netzwerk weiterleiten muss.
- **Peer-to-Peer**: Es gibt keine zusätzlichen Rechner, die Dienste bereitstellen oder Ressourcen verwalten. Die Zuständigkeiten sind gleichmäßig auf die Rechner im System verteilt, die als Peers bezeichnet werden und entweder als Client oder als Server fungieren können.

1.3 SKALIERBARE ANWENDUNGEN

Scalability is a desirable attribute of a network, system, or process.

André B. Bondi: Characteristics of Scalability and Their Impact on Performance

Arten der Skalierbarkeit

- Last Skalierbarkeit
- Speicher Skalierbarkeit
- Speicher-Zeit Skalierbarkeit
- Strukturelle Skalierbarkeit

Last Skalierbarkeit

- Ein System ist skalierbar, bei geringer, mittlerer oder hoher Last funktioniert und die verfügbaren Ressourcen sinnvoll nutzt
- Nach Möglichkeit ohne unangemessene Verzögerung und ohne unproduktiven Ressourcenverbrauch oder Ressourcenkonflikte
- Faktoren, die die Last-Skalierbarkeit gefährden können
    - Gemeinsam genutzte Ressource
    - Resourcenklassen, die ihre eigene Nutzung erhöhen (Selbstexpansion)
    - unzureichende Ausnutzung der Parallelität

Speicher Skalierbarkeit

Ein System oder eine Anwendung gilt als skalierbar, wenn der Speicherbedarf mit zunehmender Anzahl der unterstützten Elemente nicht in "untragbare" Höhen steigt.

- Eine bestimmte Anwendung oder Datenstruktur ist skalierbar, wenn ihr Speicherbedarf höchstens sublinear mit der Anzahl der fraglichen Elemente ansteigt
- Um diese Skalierbarkeit zu erreichen, können verschiedene Programmiertechniken eingesetzt werden, wie Methoden für dünnbesetzte Matrizen oder Kompression
- Da z.B. Komprimierung Zeit in Anspruch nimmt, kann Speicher-Skalierbarkeit nur auf Kosten der Last-Skalierbarkeit erreicht werden

Speicher-Zeit Skalierbarkeit

Ein System ist Speicher-Zeit-Skalierbar, wenn es auch dann noch reibungslos funktioniert, wenn die Anzahl der Objekte, die es umfasst, um Größenordnungen zunimmt.

Datenstrukturen und Algorithmen, die zur Implementierung verwendet werden, ermöglichen einen reibungslosen und schnellen Betrieb, unabhängig davon, ob es sich um ein mittelgroßes oder großes System handelt.

- Nicht Speicher-Zeit-Skalierbar: Suchmaschine mit lineare skalierendem Algorithmus
- Speicher-Zeit-Skalierbar: Suchmaschine mit indizierten oder sortierten Datenstrukturen wie Hashtabellen oder binäre Bäume

Faktor die gegen Speicher-Zeit-Skalierbar sprechen:

1. Das Vorhandensein einer großen Anzahl von Objekten kann zu einer stärkeren Belastung führen.
2. Die Fähigkeit, eine schnelle Suche durchzuführen, kann durch die Größe einer Datenstruktur und deren Organisation beeinträchtigt werden.
3. Ein System oder eine Anwendung, die eine große Menge an Speicher belegt, kann einen erheblichen Paging-Overhead verursachen.

Strukturelle Skalierbarkeit

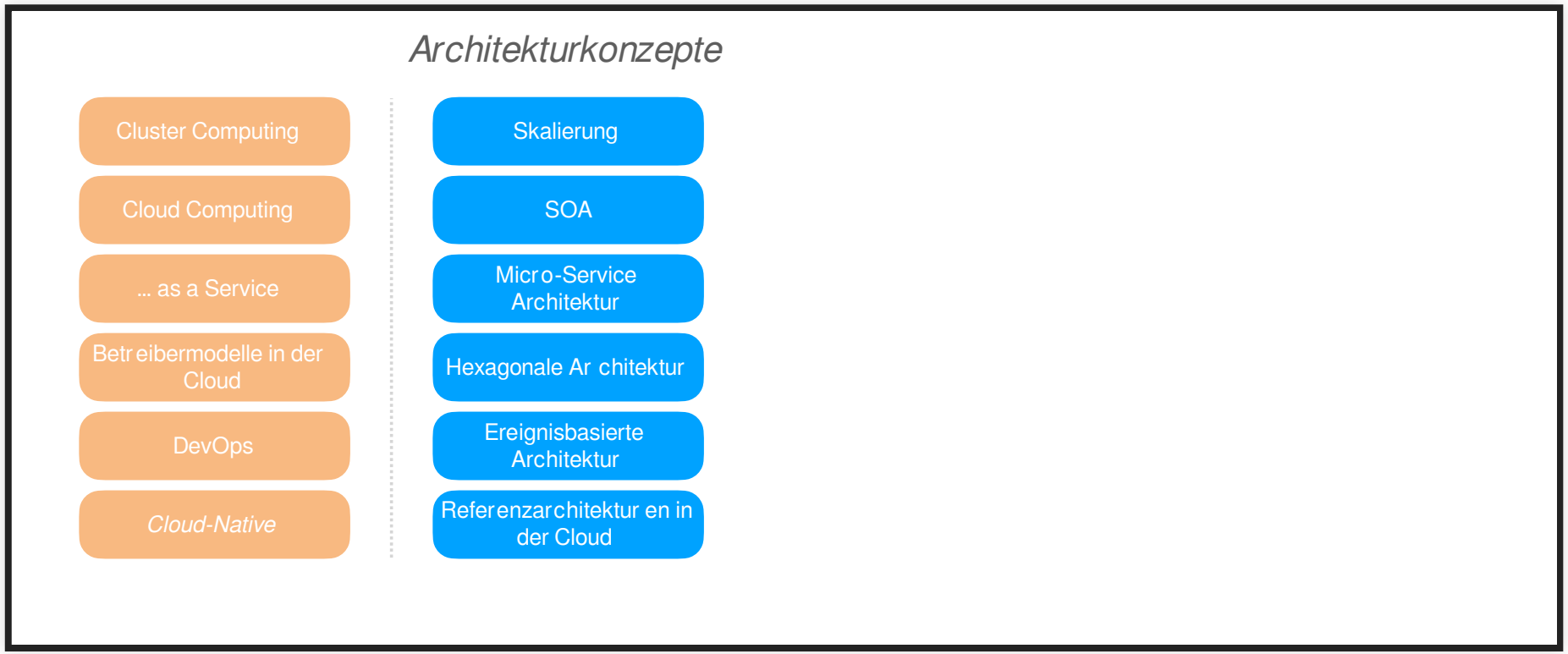
Ein System ist strukturell skalierbar, wenn seine Implementierung oder seine Standards das Wachstum der Anzahl von Objekten, die es umfasst, nicht behindern, oder zumindest nicht innerhalb eines bestimmten Zeitrahmens.

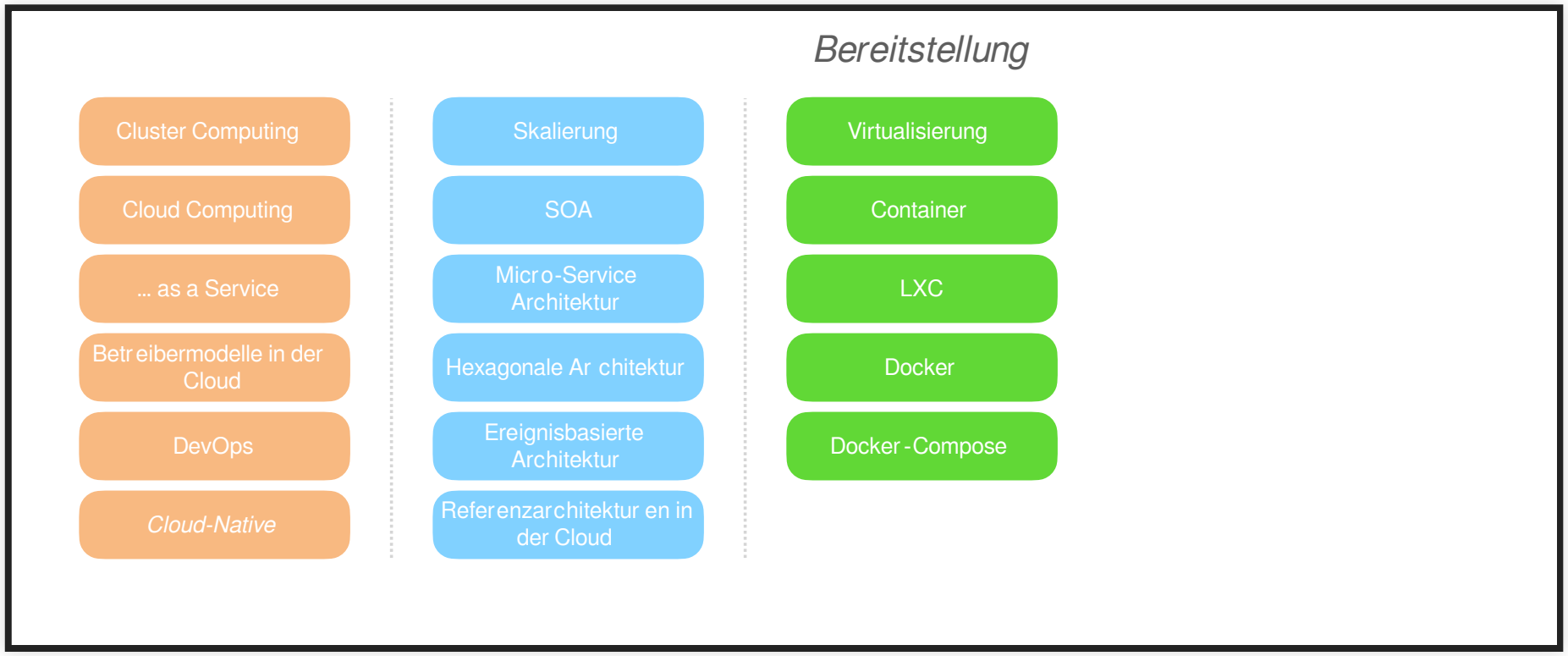
Dies ist ein relativer Begriff, da die Skalierbarkeit von der Anzahl der Objekte abhängt, die jetzt im Verhältnis zur Anzahl der späteren Objekte von Interesse sind.

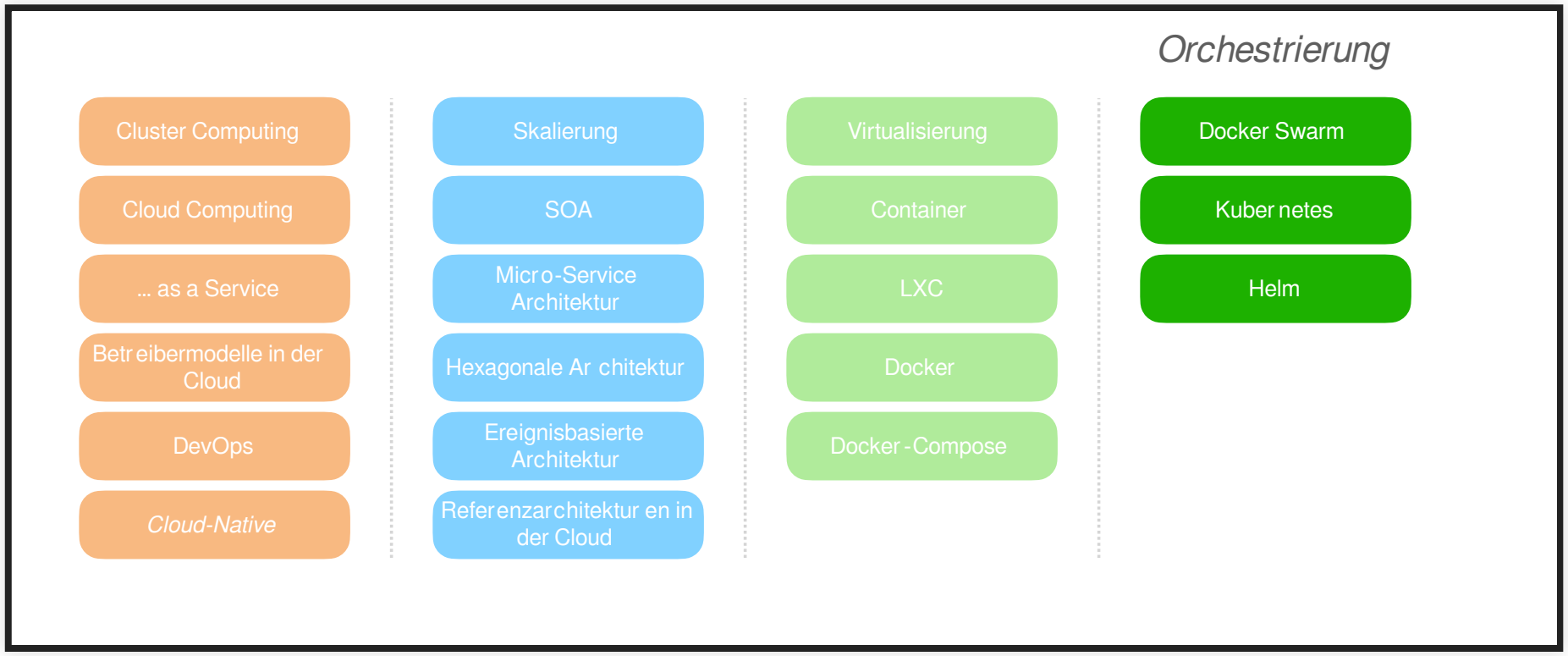
- Jedes System mit einem endlichen Adressraum hat Grenzen für seine Skalierbarkeit.
- Beispielsweise enthält ein Paket-Header normalerweise eine feste Anzahl von Bits.
    - Ist es ein Adressfeld, ist die Anzahl der adressierbaren Knoten begrenzt.
    - Ist es eine Fenstergröße, so ist die Menge der unbestätigten Daten begrenzt.
- Ein Telefonnummernschema mit einer festen Anzahl von Ziffern, ist nur insoweit skalierbar, als die maximale Anzahl unterschiedlicher Nummern deutlich größer ist als die Menge der zu vergebenden Nummern.

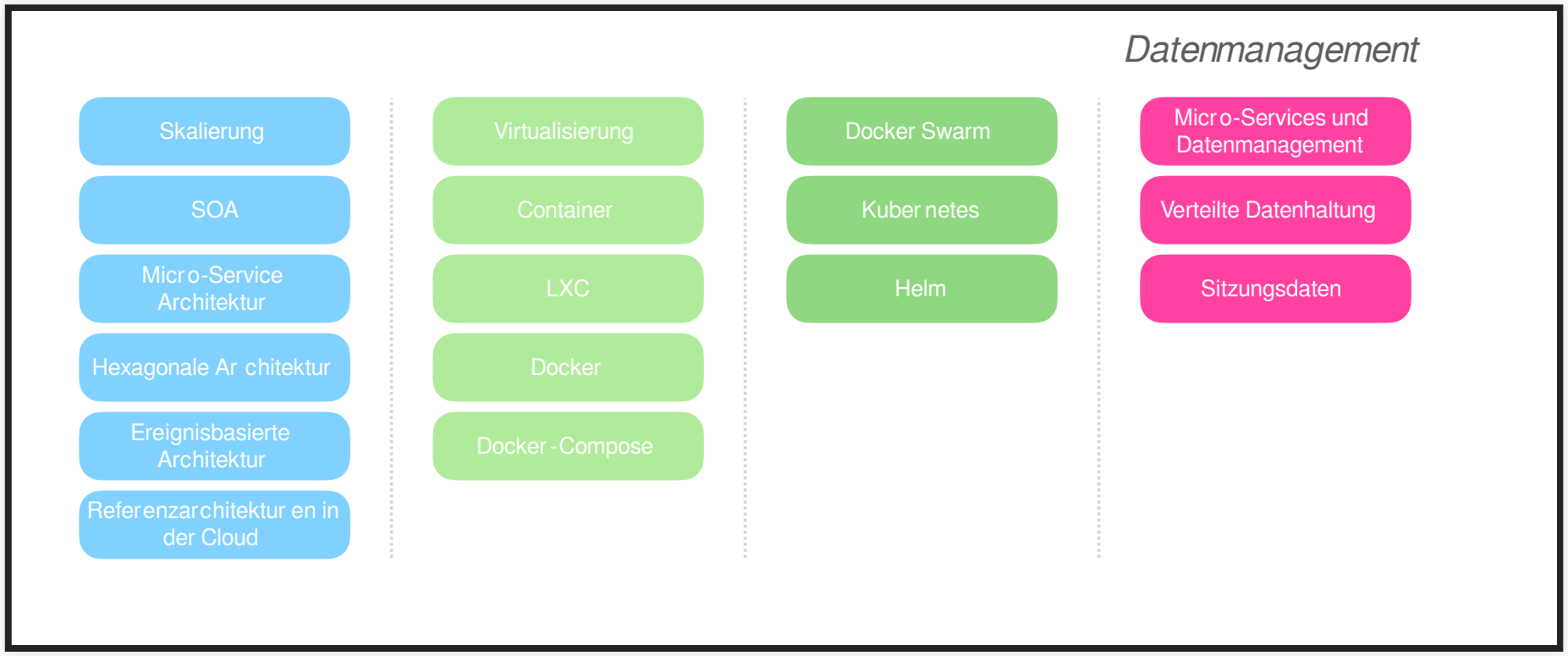
1.4 AUSBLICK

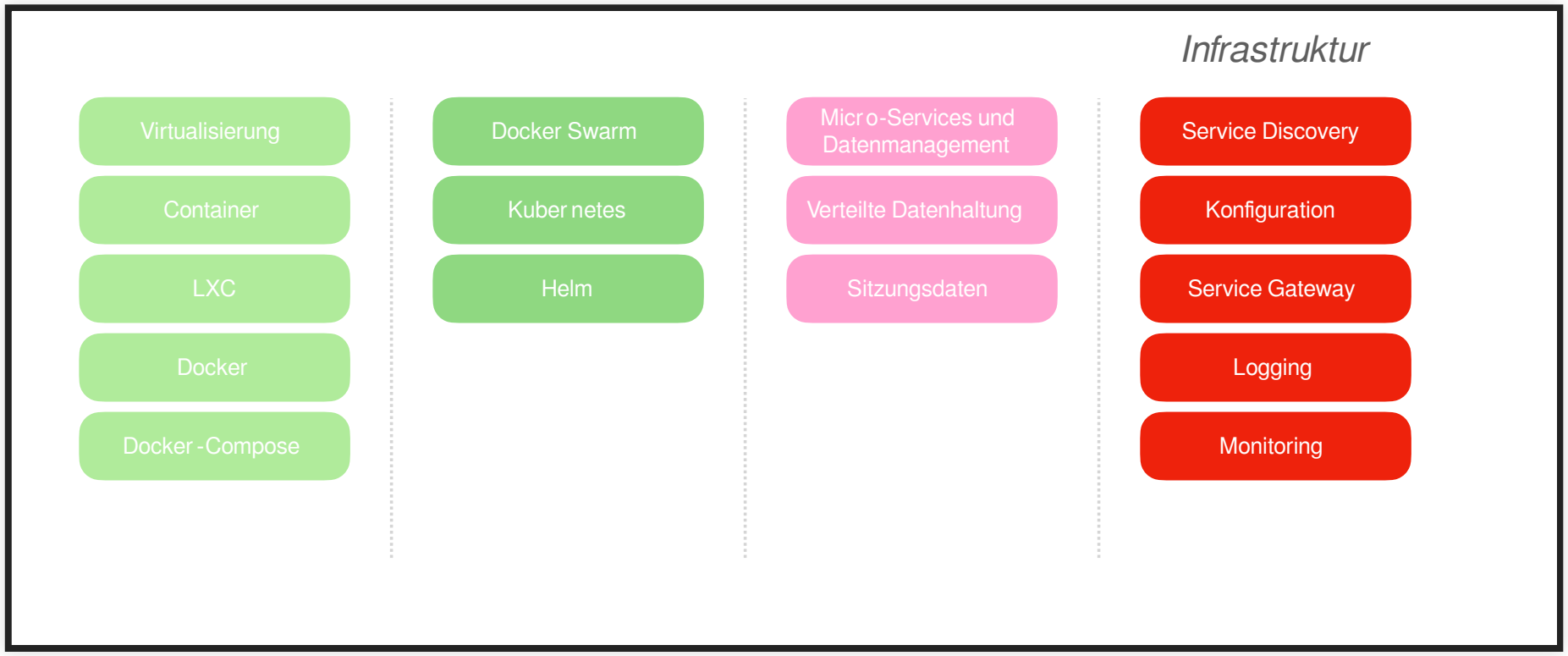

Was ist?
Cluster Computing
Cloud Computing
... as a Service
Betreibermodelle in der Cloud
DevOps
Cloud-Native


Architekturkonzepte
Cluster Computing
Cloud Computing
... as a Service
Betreibermodelle in der Cloud
DevOps
Cloud-Native
Skalierung
SOA
Micro-Service Architektur
Hexagonale Architektur
Ereignisbasierte Architektur
Referenzarchitekturen in der Cloud


Bereitstellung
Cluster Computing
Cloud Computing
... as a Service
Betreibermodelle in der Cloud
DevOps
Cloud-Native
Skalierung
SOA
Micro-Service Architektur
Hexagonale Architektur
Ereignisbasierte Architektur
Referenzarchitekturen in der Cloud
Virtualisierung
Container
LXC
Docker
Docker-Compose


Orchestrierung
Cluster Computing
Cloud Computing
... as a Service
Betr eibermodelle in der Cloud
DevOps
Cloud-Native
Skalierung
SOA
Micr o-Service Architektur
Hexagonale Ar chitektur
Ereignisbasierte Architektur
Referenzarchitektur en in der Cloud
Virtualisierung
Container
LXC
Docker
Docker -Compose
Docker Swarm
Kuber netes
Helm


Datenmanagement
Skalierung
SOA
Micro-Service Architektur
Hexagonale Architektur
Ereignisbasierte Architektur
Referenzarchitekturen in der Cloud
Virtualisierung
Container
LXC
Docker
Docker-Compose
Docker Swarm
Kubernetes
Helm
Micro-Services und Datenmanagement
Verteilte Datenhaltung
Sitzungsdaten


Infrastruktur
Virtualisierung
Container
LXC
Docker
Docker-Compose
Docker Swarm
Kubernetes
Helm
Micro-Services und Datenmanagement
Verteilte Datenhaltung
Sitzungsdaten
Service Discovery
Konfiguration
Service Gateway
Logging
Monitoring

*Ergänzend **Beispielanwendungen** in verschiedenen Ausprägungen sowie Experimente zur Ausführung der Anwendungen in verschiedenen Umgebungen.*

1.5 REFERENZEN

- André B. Bondi: Characteristics of Scalability and Their Impact on Performance. In Proceedings of the 2nd International Workshop on Software and Performance, New York, NY, USA. Pages 195-203. 2000.