

Apache Kafka

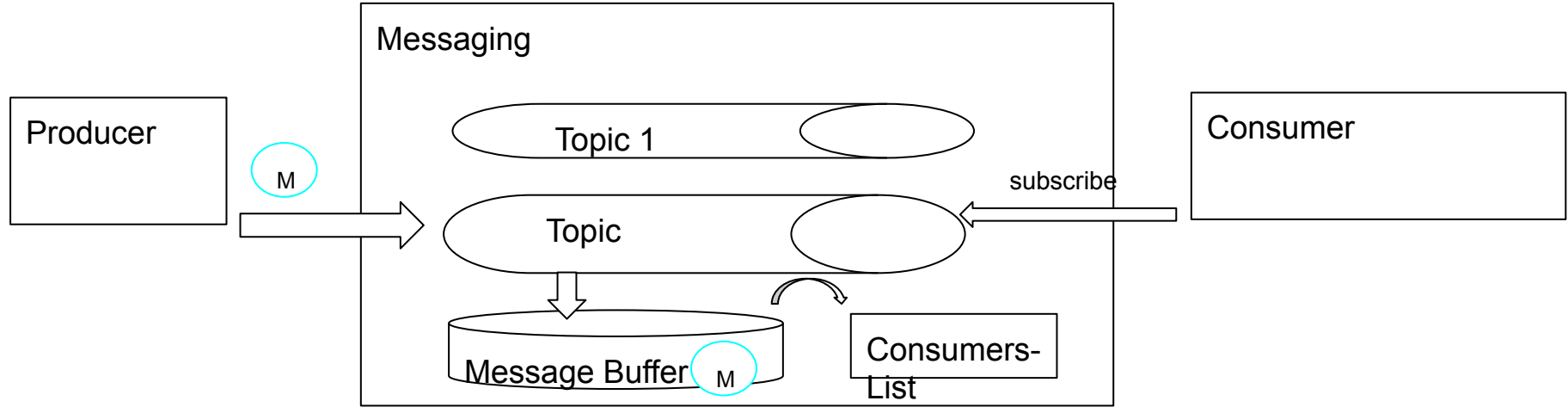
- Unternehmen?
- Ihre Rolle
 - Admin, Architect, Developer?
- Themenbezogene Vorkenntnisse
 - Selbsteinschätzung Java
- Individuelle Ziele für das Seminar
-

Ausgangssituation

- Komplexe Systemlandschaft mit einer Vielzahl Servern, von Anwendungen...
- Log-Dateien
- Status-Informationen
- Kommunikation zwischen den einzelnen Systemen
- Problemstellung
 - Speichern der Informationen
 - Verteilen der Informationen an die richtigen Zielsysteme

- Datenbank-Systeme zum Speichern aller möglicher Daten
 - NoSQL-Umfeld, insbesondere Key-Value-Store
- Messaging-System
 - Publish/Subscribe an einem Topic
- Was ist Apache Kafka?
 - Key-Value-Store zur Ablage von Datensätzen
 - Publish/Subscribe: Datensätze werden an Kafka-Consumer weitergeleitet

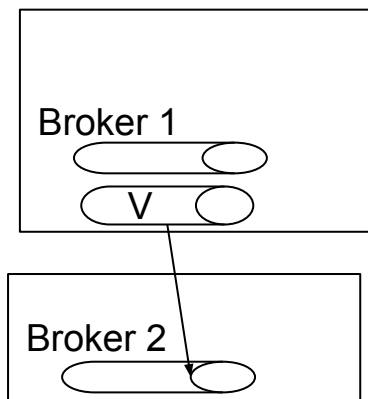
Warum ist ein Messaging-System keine Datenbank?



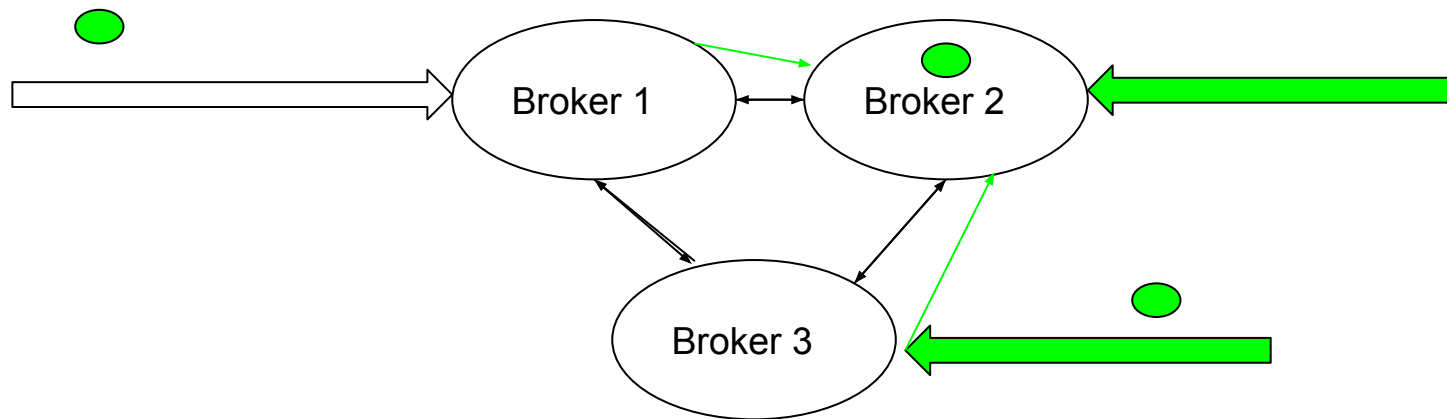
Garantien:

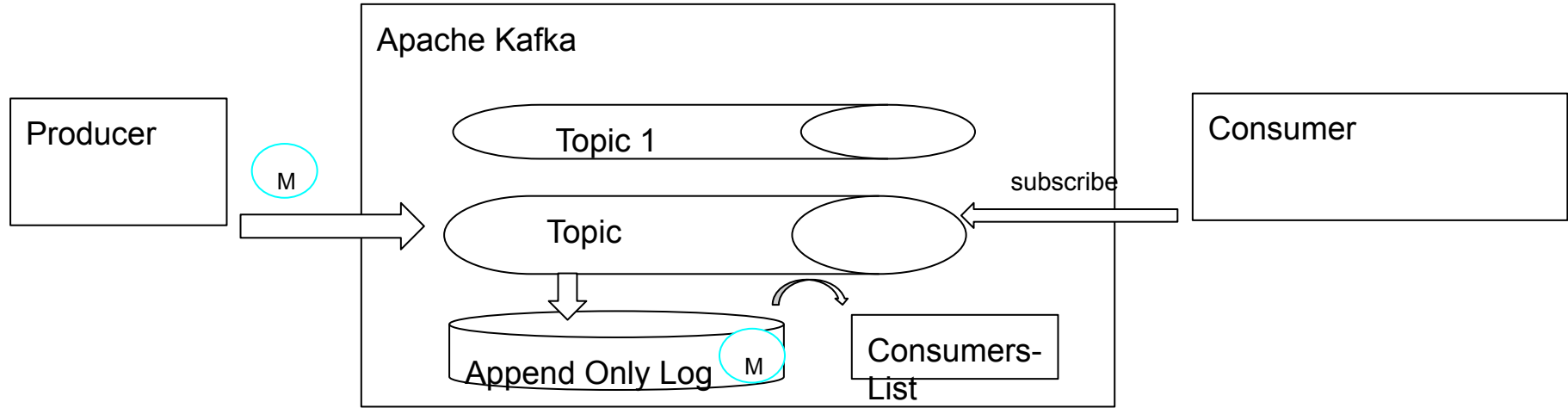
- Jede Nachricht garantiert ein Consumer
- Jede Nachricht erreicht alle aktiven Consumer
- Jede Nachricht erreicht alle registrierten Consumer

- Verarbeitete Nachrichten werden automatisch gelöscht
 - Eine “verspätete” Subscription findet keine Daten vor
- Clustering von Messaging-Systemen ist schwierig



- Topics sind aufzufassen als “Append Only” Log
- Damit kann Kafka sehr einfach im Cluster betrieben werden!
 - “Master”-less Ring-Cluster

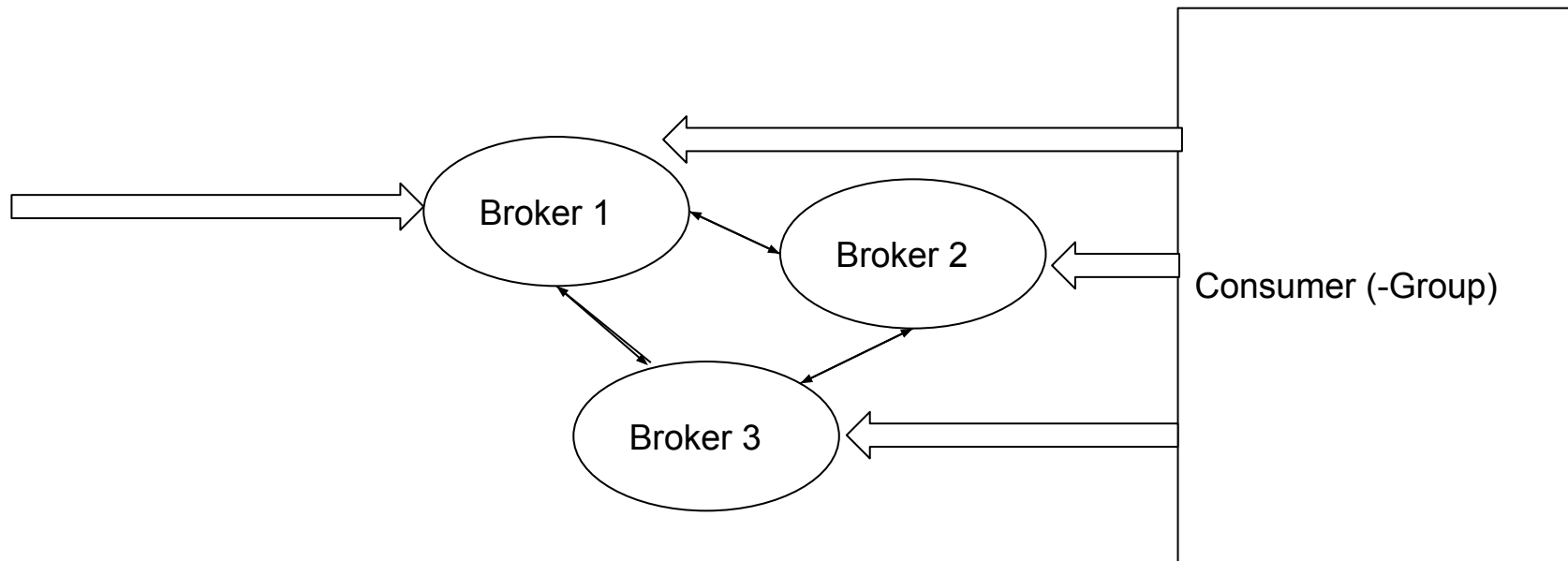




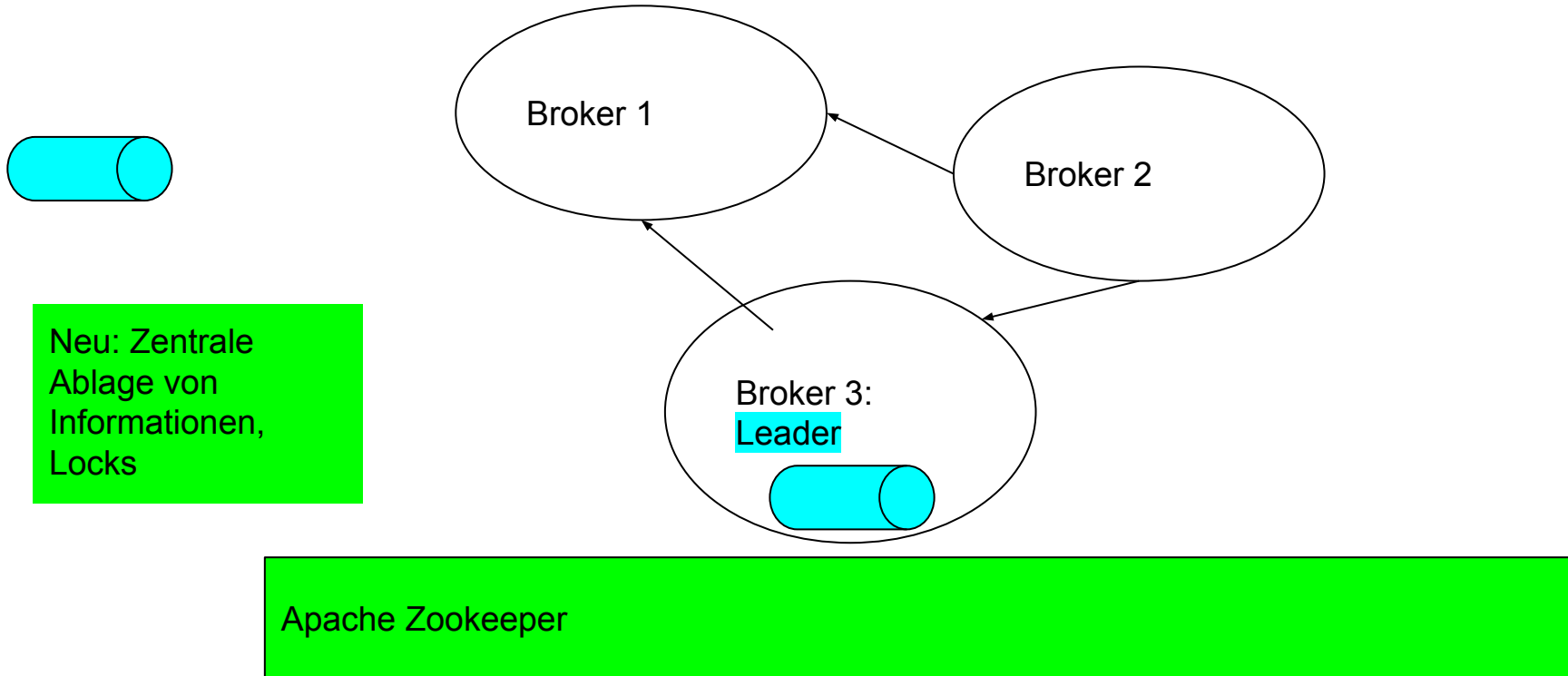
Garantien:

- At least once
- At most once
- Exactly Once (erfordert spezielle Producer und Consumer!)

- Consumer arbeiten mit dem Ring-Cluster zusammen
-

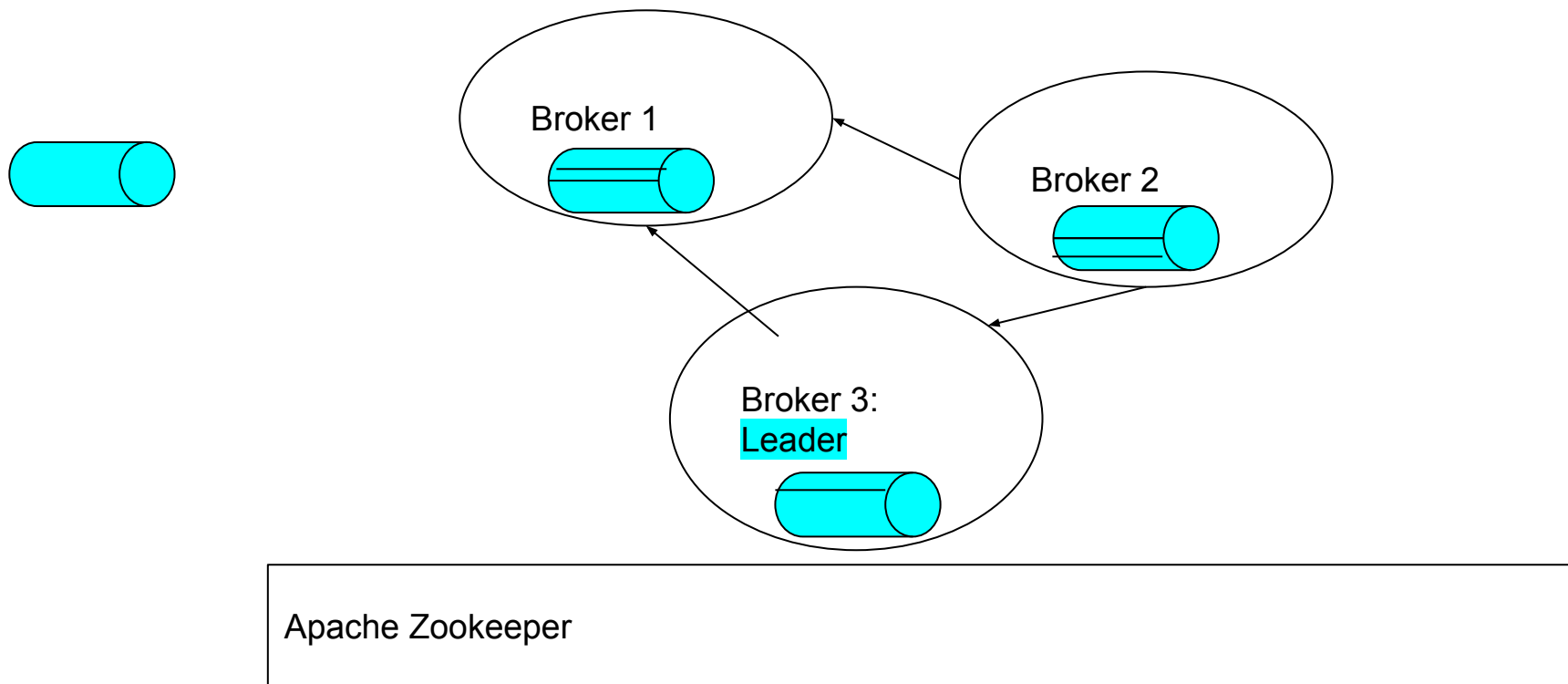


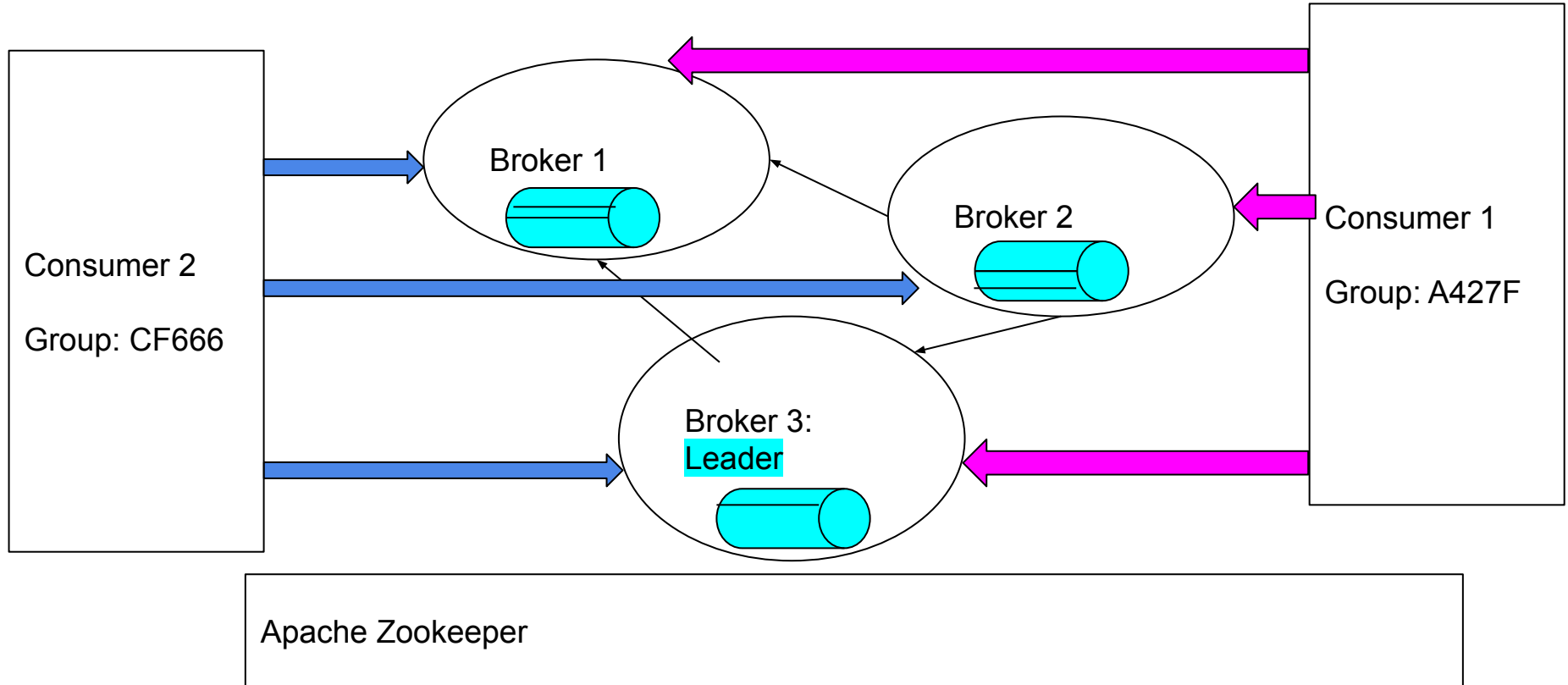
Verteilung der Topics auf den Ring



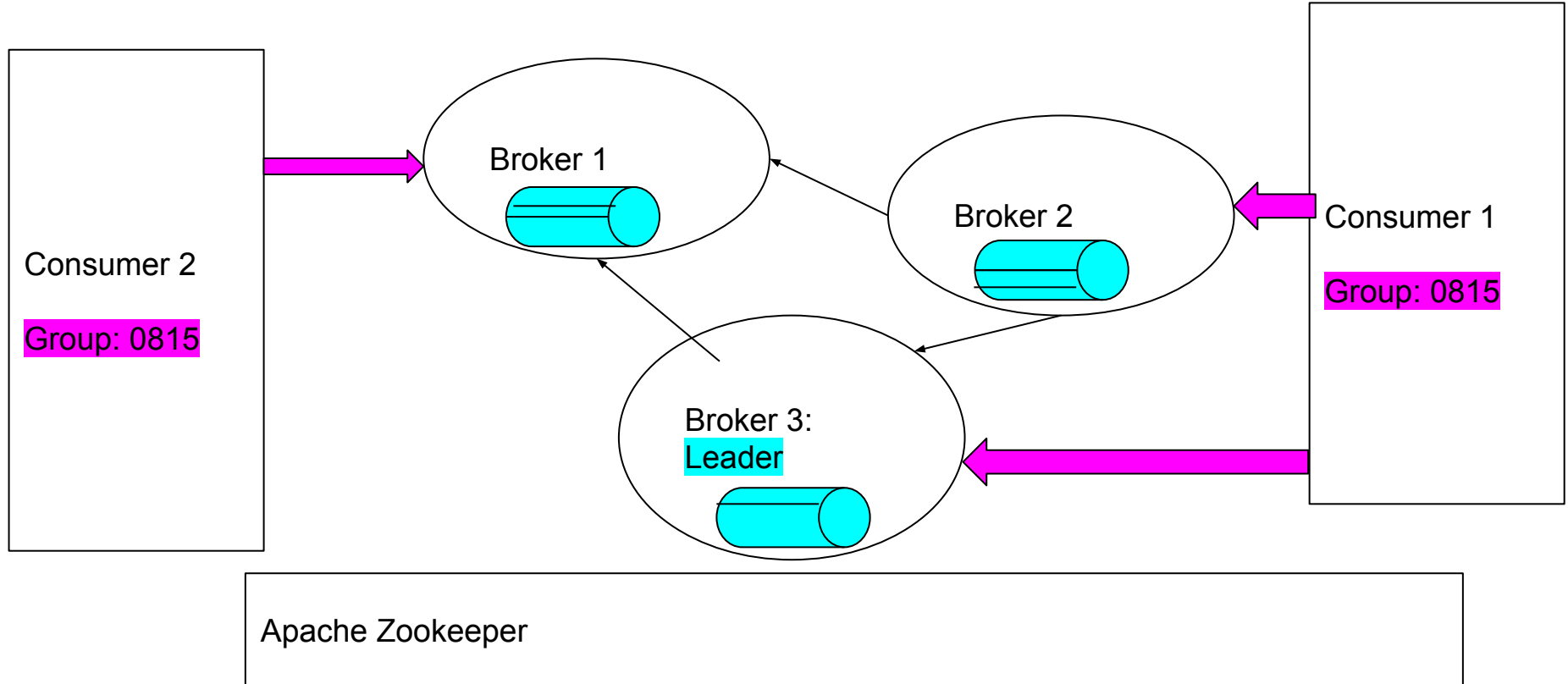
- Mit jedem Release von Kafka wird die Rolle des Zookeepers zurückgefahren
 - Roadmap: “Wir wollen die Zookeeper-Dependency entfernen”

Topics sind partitioniert



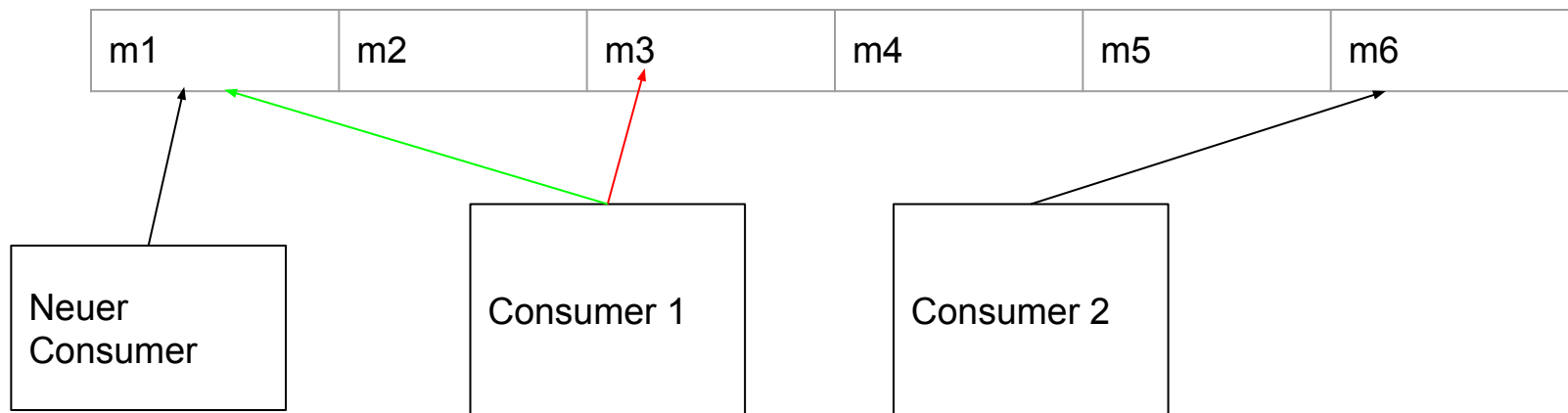


Partitionen und Consumer einer Group



- Beim Anlegen des Topics zusätzlich die Angabe eines Replikationsfaktors

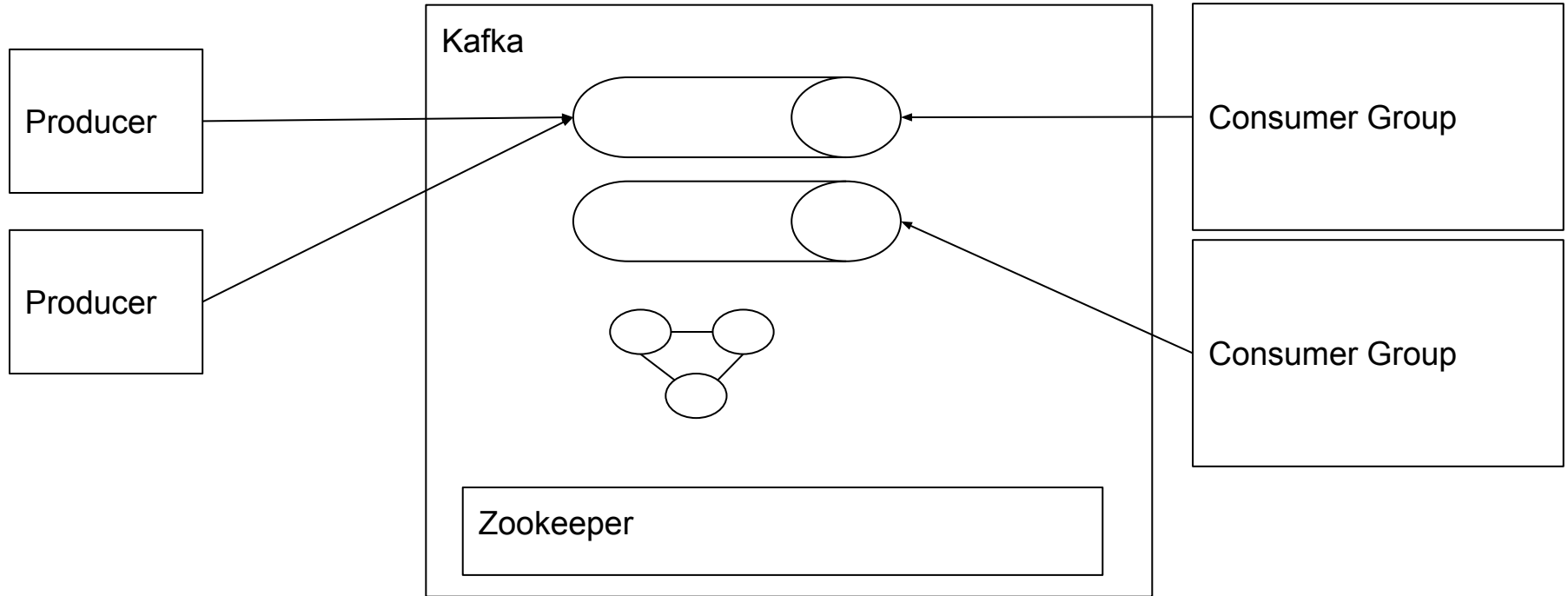
- Offset-Counter pro Consumer
 - Ablage ~~im Zookeeper~~ in einem speziellen internen Topic



- Alle Nachrichten werden beim Ablauf einer Verweilzeit automatisch gelöscht

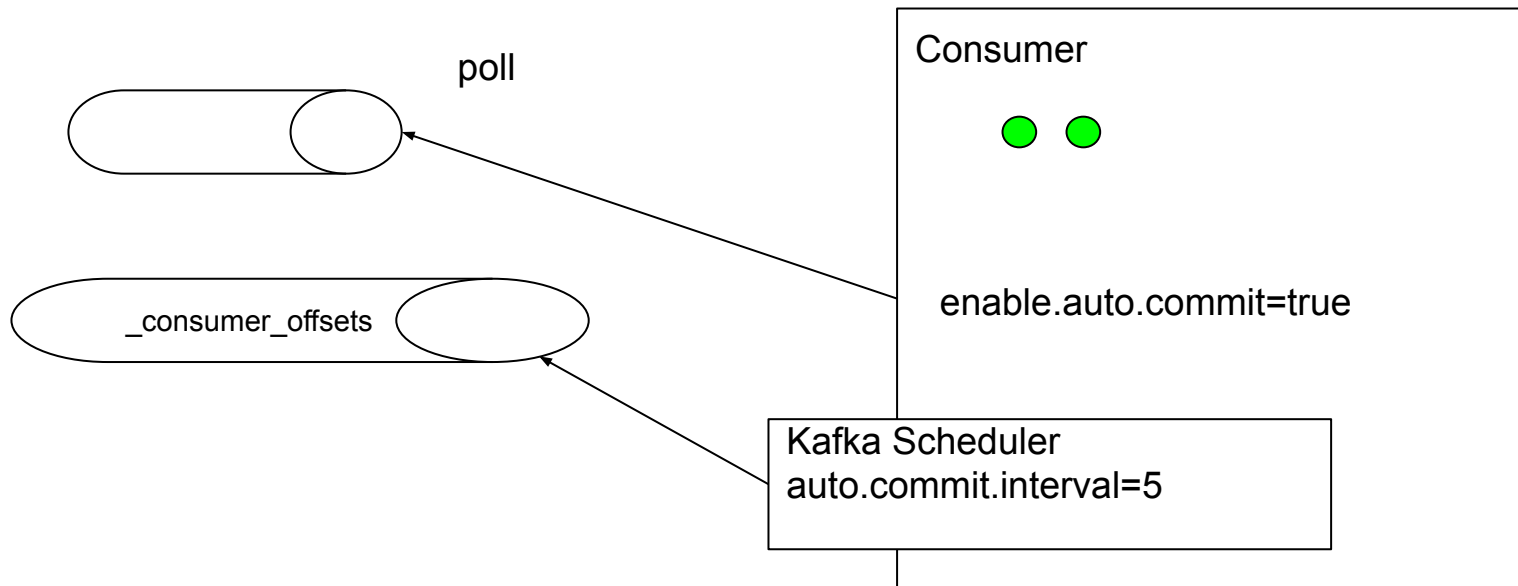
- Consistency
 - Der Gesamtbestand der Daten im Cluster ist stets konsistent
- Availability
 - Ein Broker nimmt Schreibvorgänge entgegen oder liefert Nachrichten aus
- Partition Tolerance
 - Partition Fault: Fehler in der Kommunikation zwischen den Brokern
- CAP-Theorem
 - Es gibt ausschließlich “2 aus 3”-Systeme
 - CA, CP, AP
 - Kafka: **AC** oder **AP**

Kafka und Anwendungen

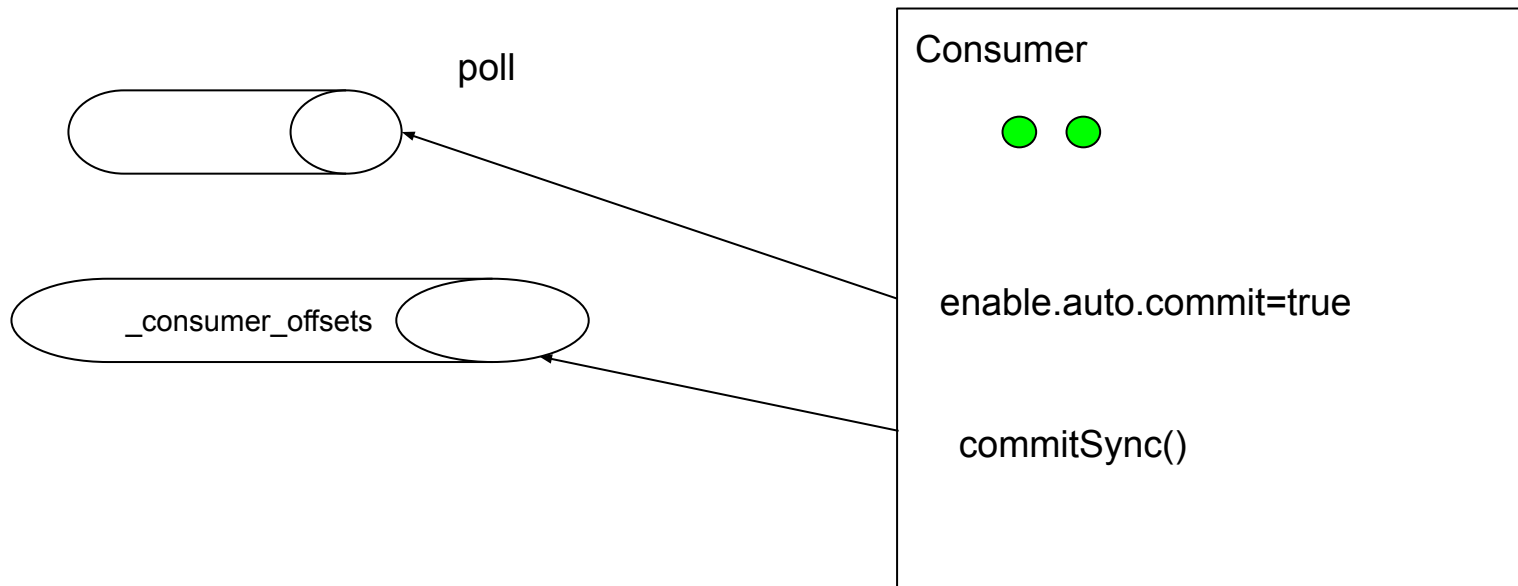


- Keine Garantie
 - Nachrichten können 0 - n mal verarbeitet werden
- At most once
 - 0 oder 1
- At least once
 - 1 bis n
- ~~Exactly~~ Effectively Once
 - 1

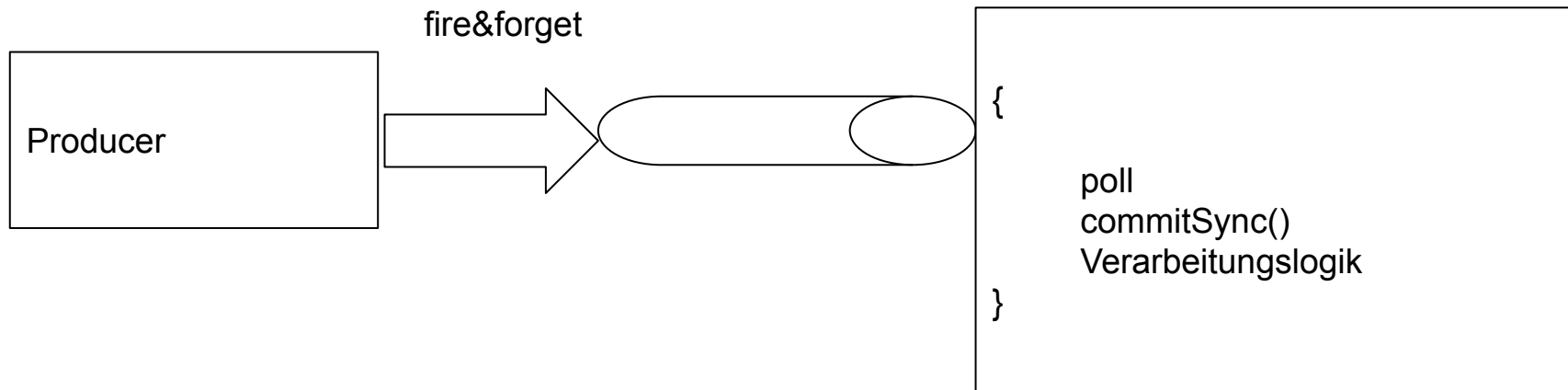
No guarantee



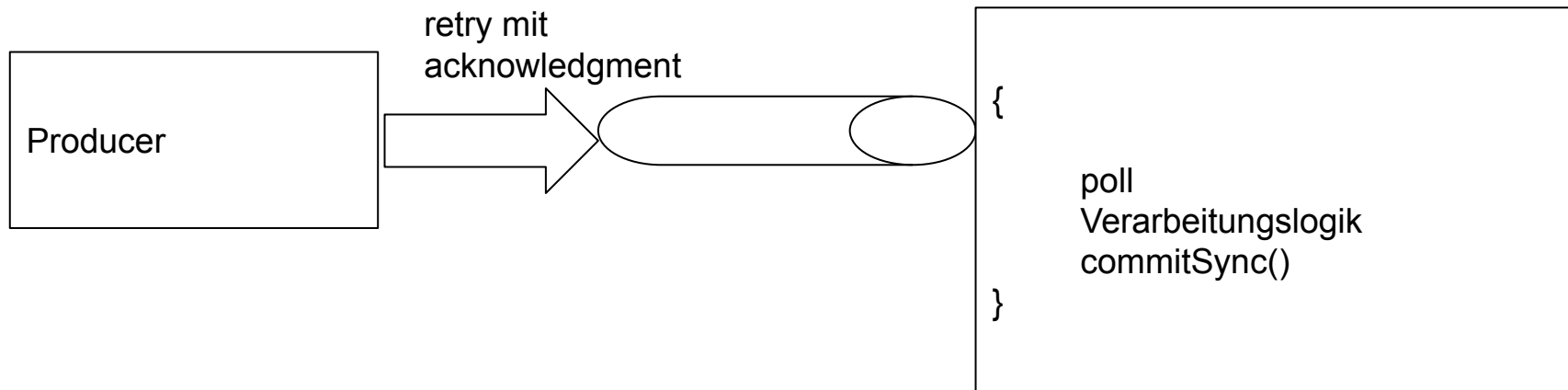
Alles weitere: `enable.auto.commit=false`



At most once

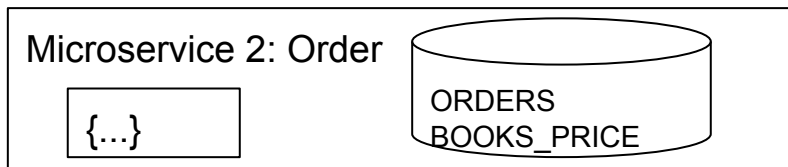
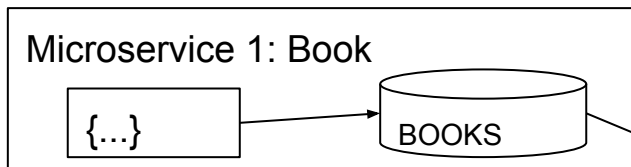


At least once

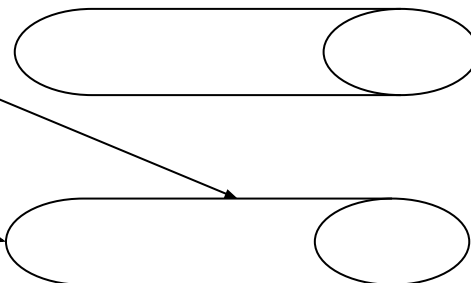


Warum war/ist Kafka hier so schlampig?

Microservice-Architektur



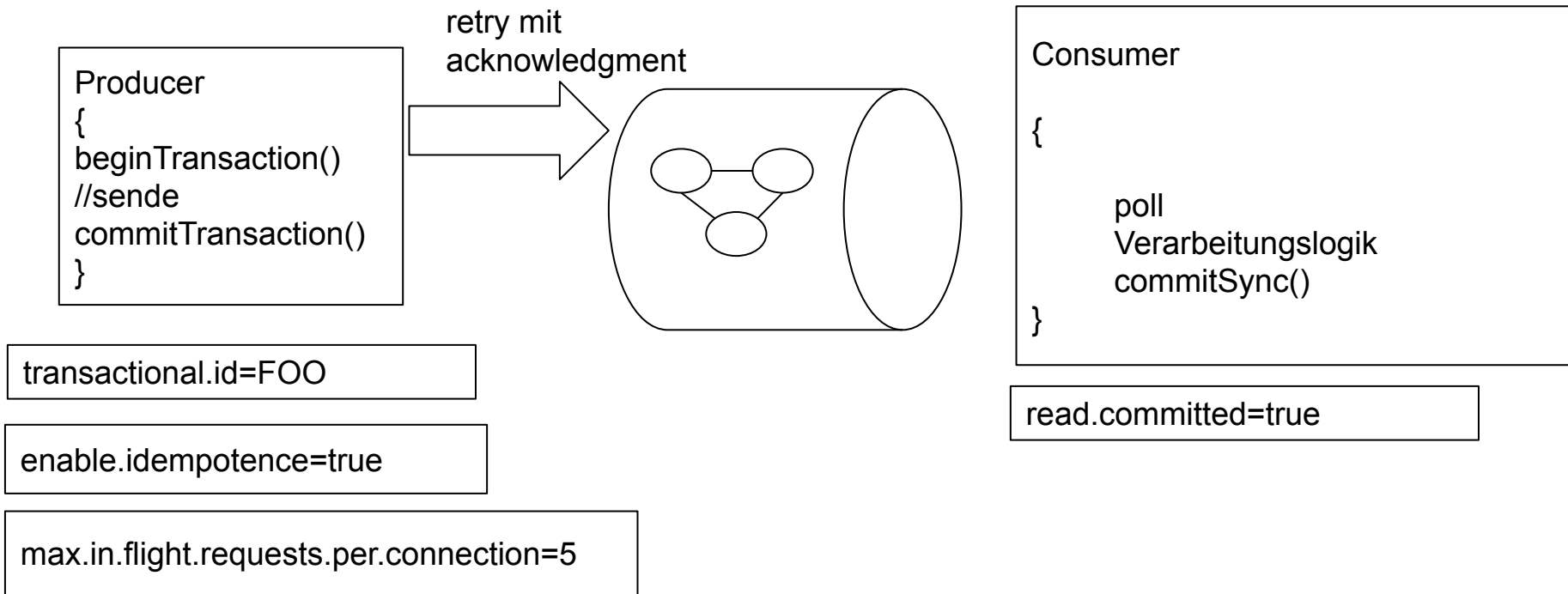
Blackboard/Whiteboard
Apache Kafka



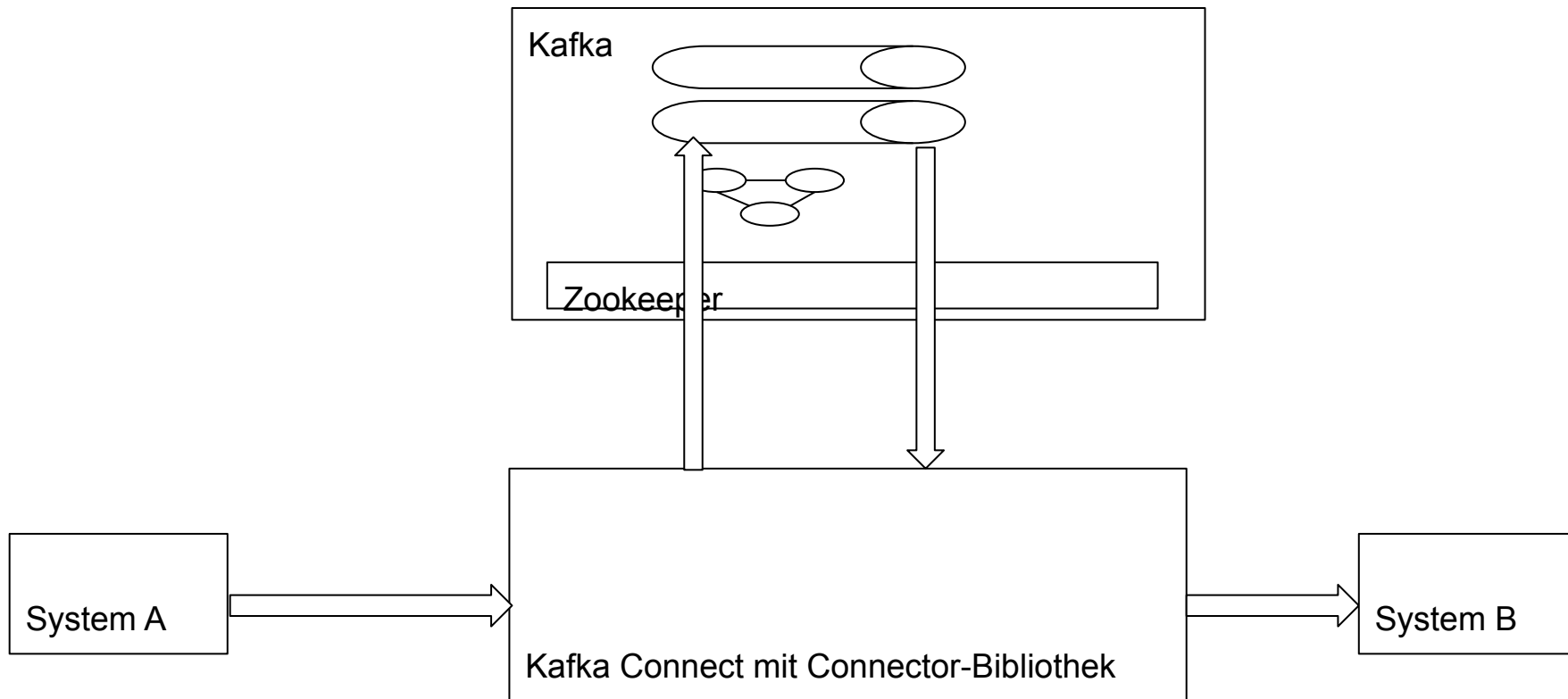
publish

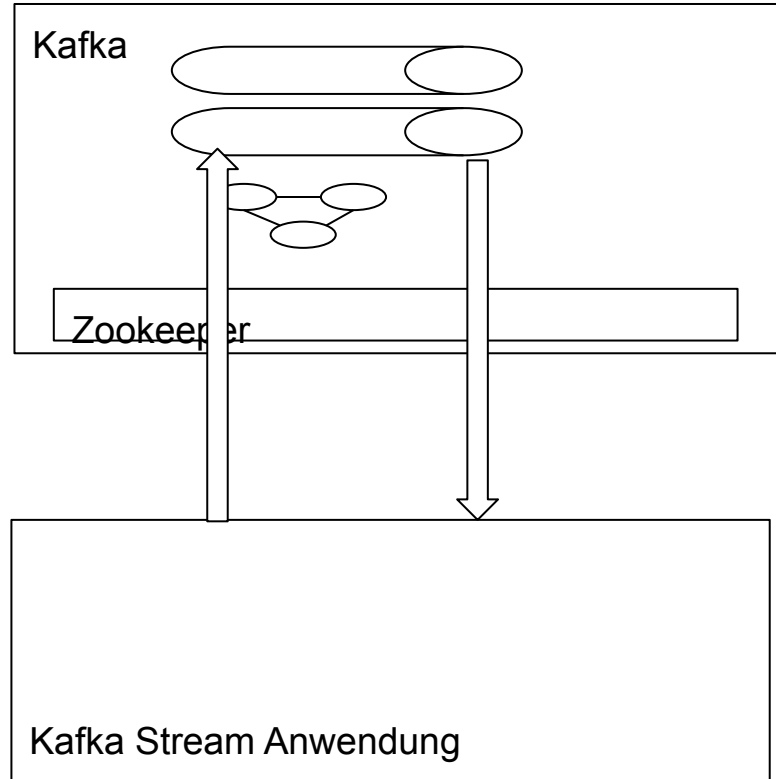
poll

Effectively Once

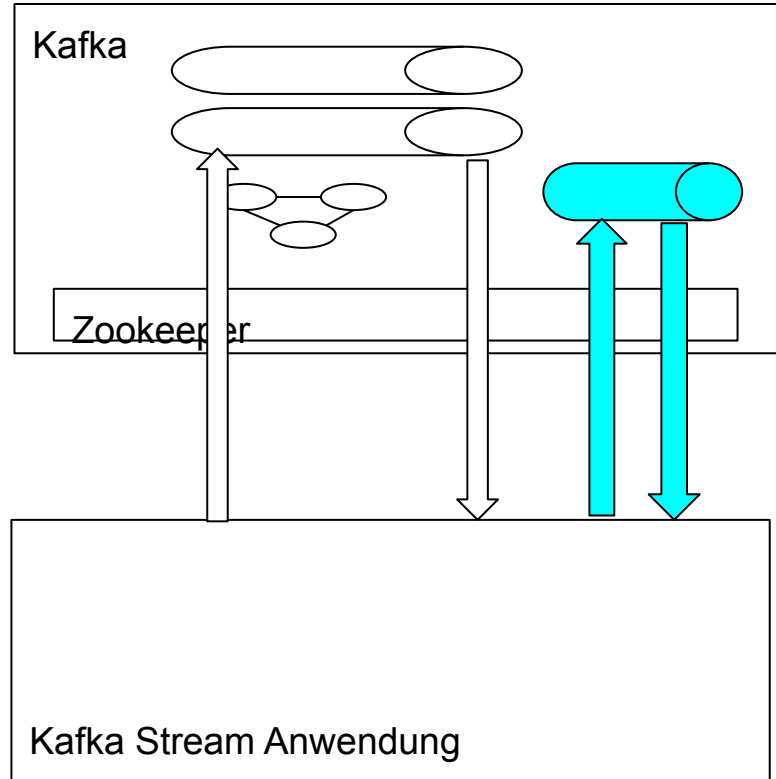


- Stream-orientierte Datenverarbeitung
 - Datenquelle
 - Filter, Transformieren
 - Datensenke





- Stream-orientierte Datenverarbeitung
 - Datenquelle
 - Filter, Transformieren, Aggregieren und Gruppieren
 - z.B. aktuelle Mittelwertbestimmung
 - Datensenke



Kafka-Clusters

- Vollkommen unabhängig voneinander
- Beide sind Java-Prozesse
- Kafka-Broker benutzt exzessiv das “Offheap”-Memory der Java Virtual Machine
- Beide sind Bestandteil der Kafka-Distribution
 - JAR-Dateien
 - Skripte zum Starten und Stoppen
 - Konfigurationsdateien
- Zusätzlich in der Distribution: Apache Kafka Connect Server

- Konsolen-Skripte
- Kein Web-Frontend etc.
- Sehr rudimentär...

Ein HelloWorld-Kafka-Cluster

