Apache Kafka

Vorstellungsrunde

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

Ausgangssituation

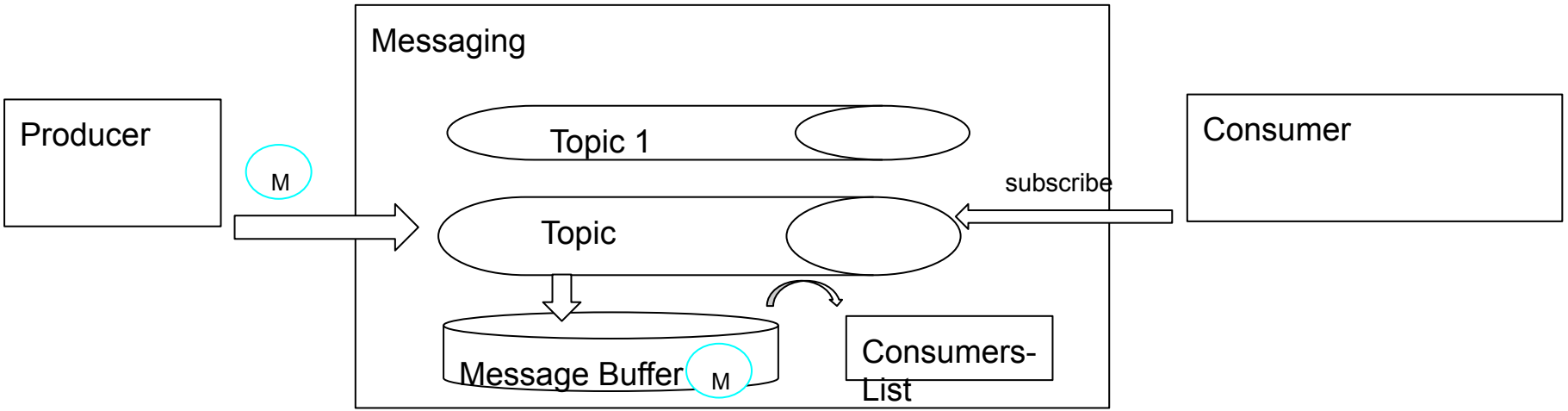
Informationsflut

- 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

Lösung

- 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?

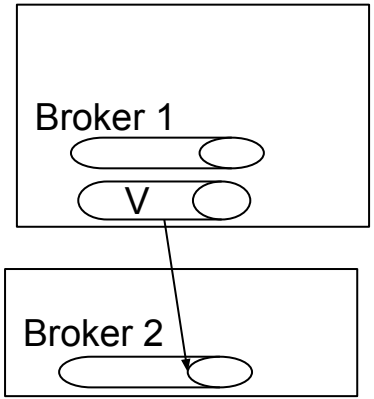


Garantien:

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

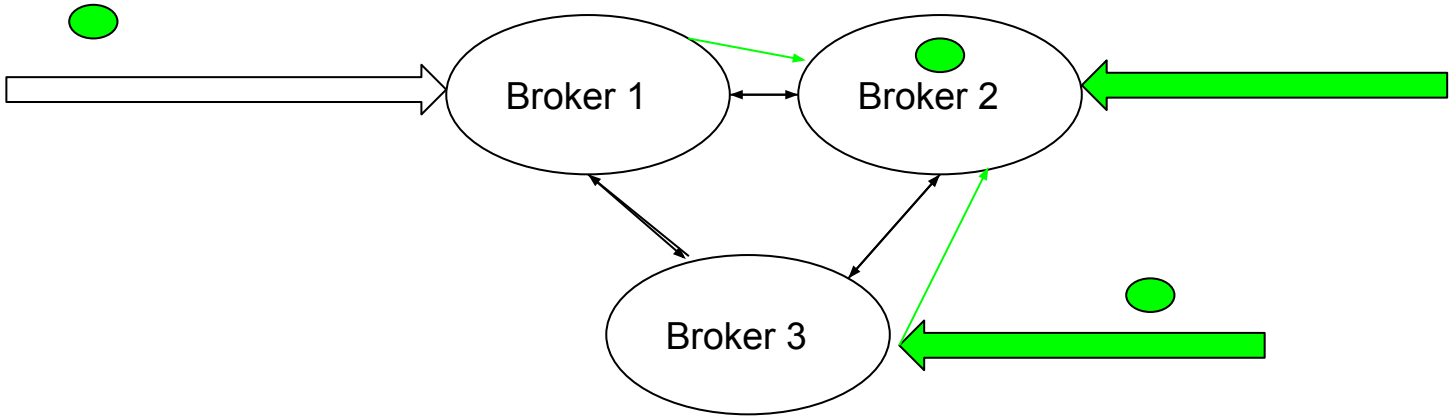
Konsequenzen

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

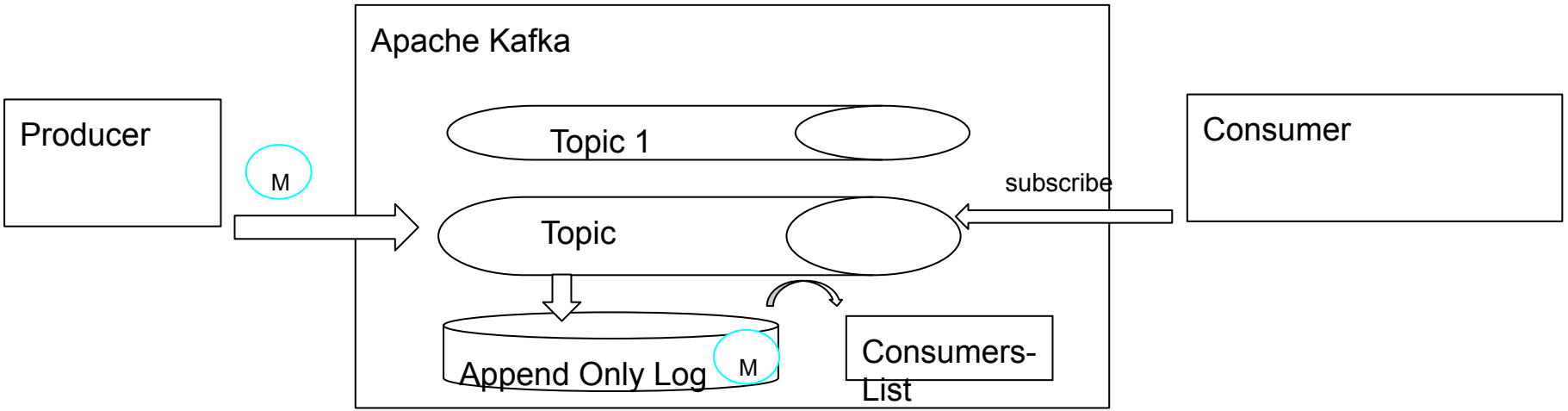


Wie arbeitet Kafka?

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



Kafka als Messaging-System

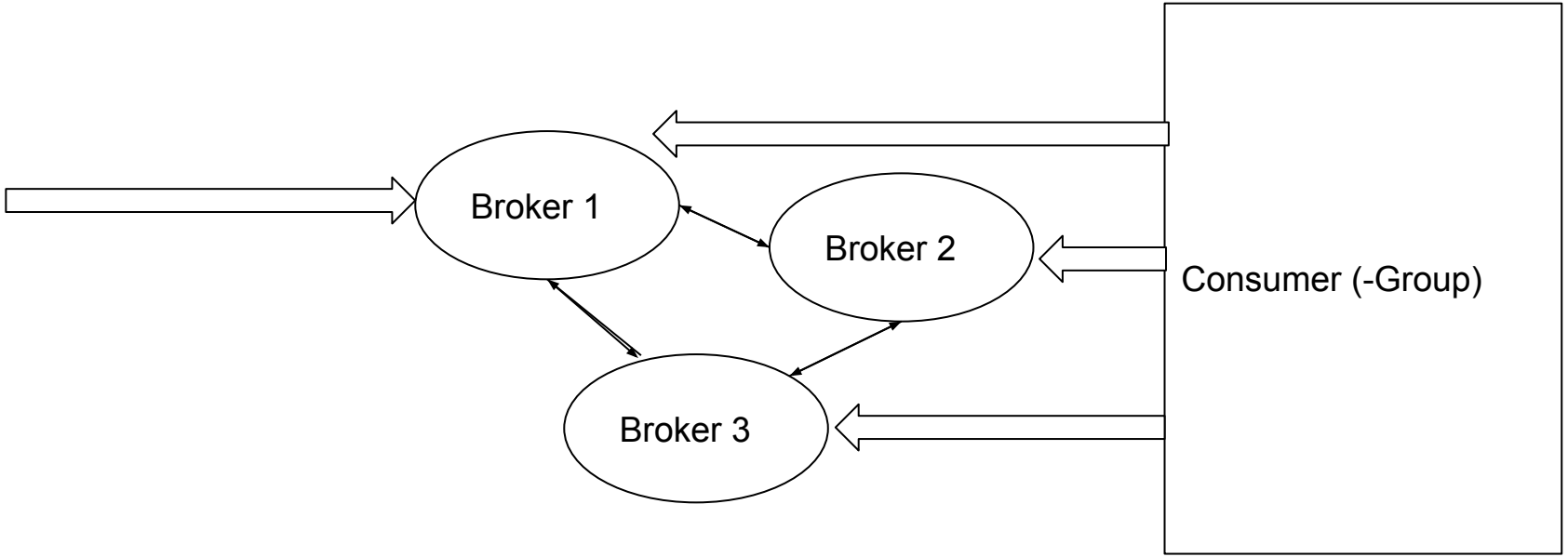


Garantien:

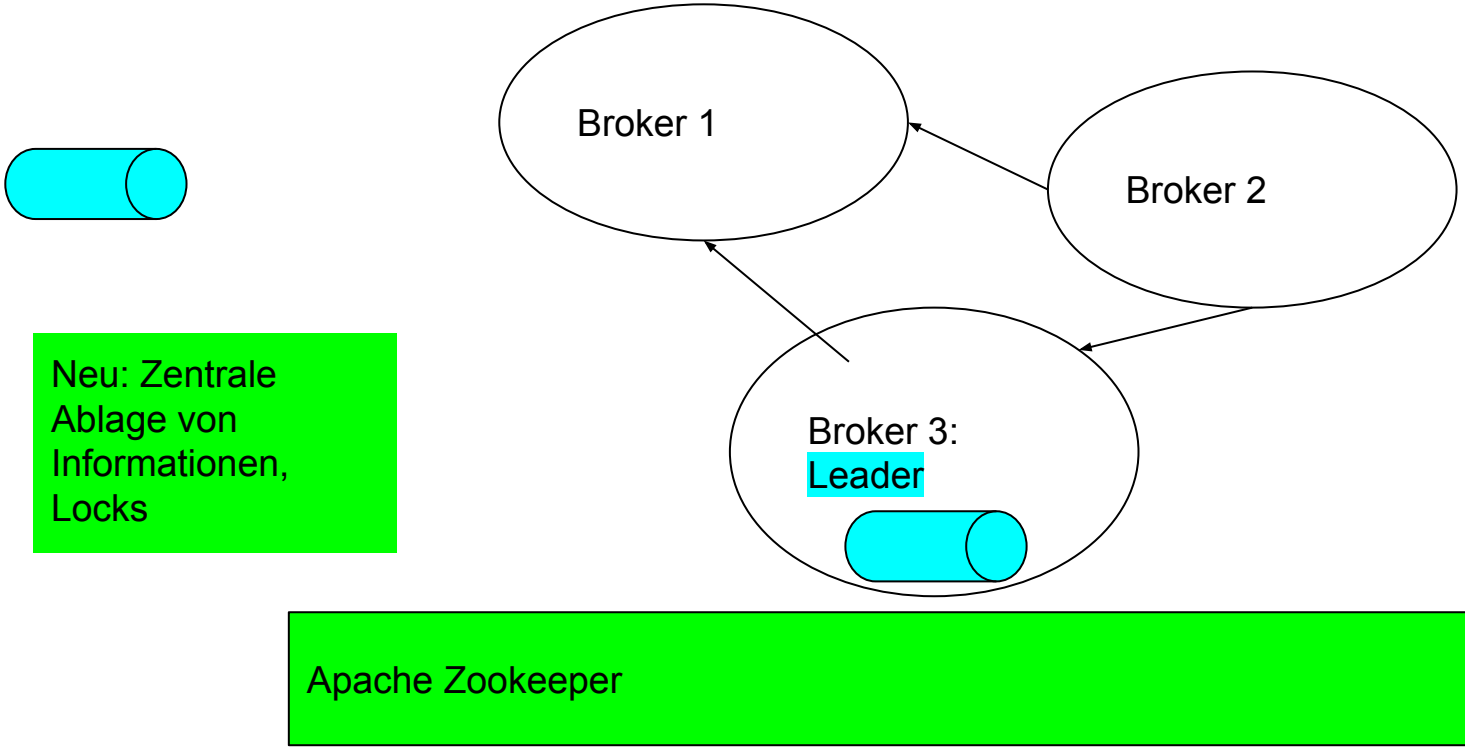
- At least once
- At most once
- Exactly Once (erfordert spezielle Producer und Consumer!)

Wie arbeitet Kafka?

- Consumer arbeiten mit dem Ring-Cluster zusammen
-



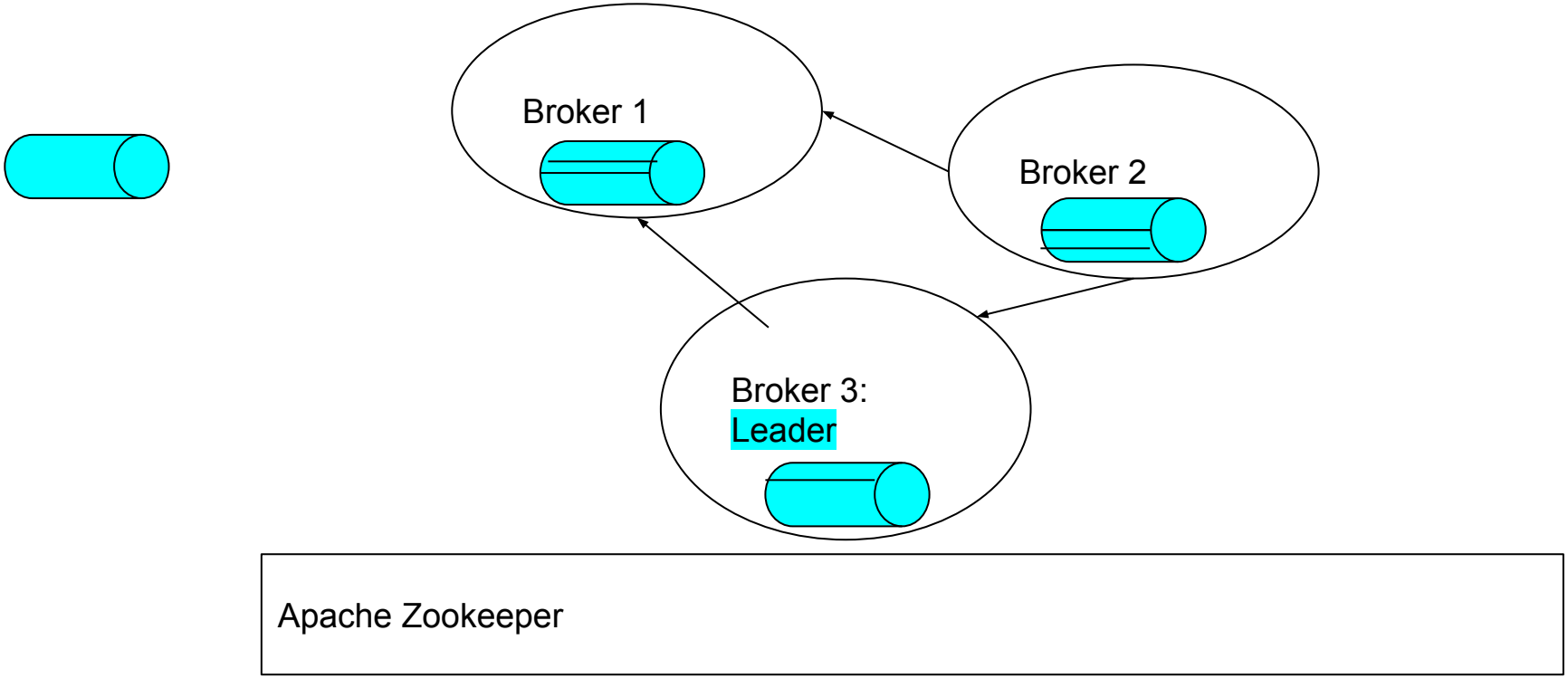
Verteilung der Topics auf den Ring



Nebenbemerkung

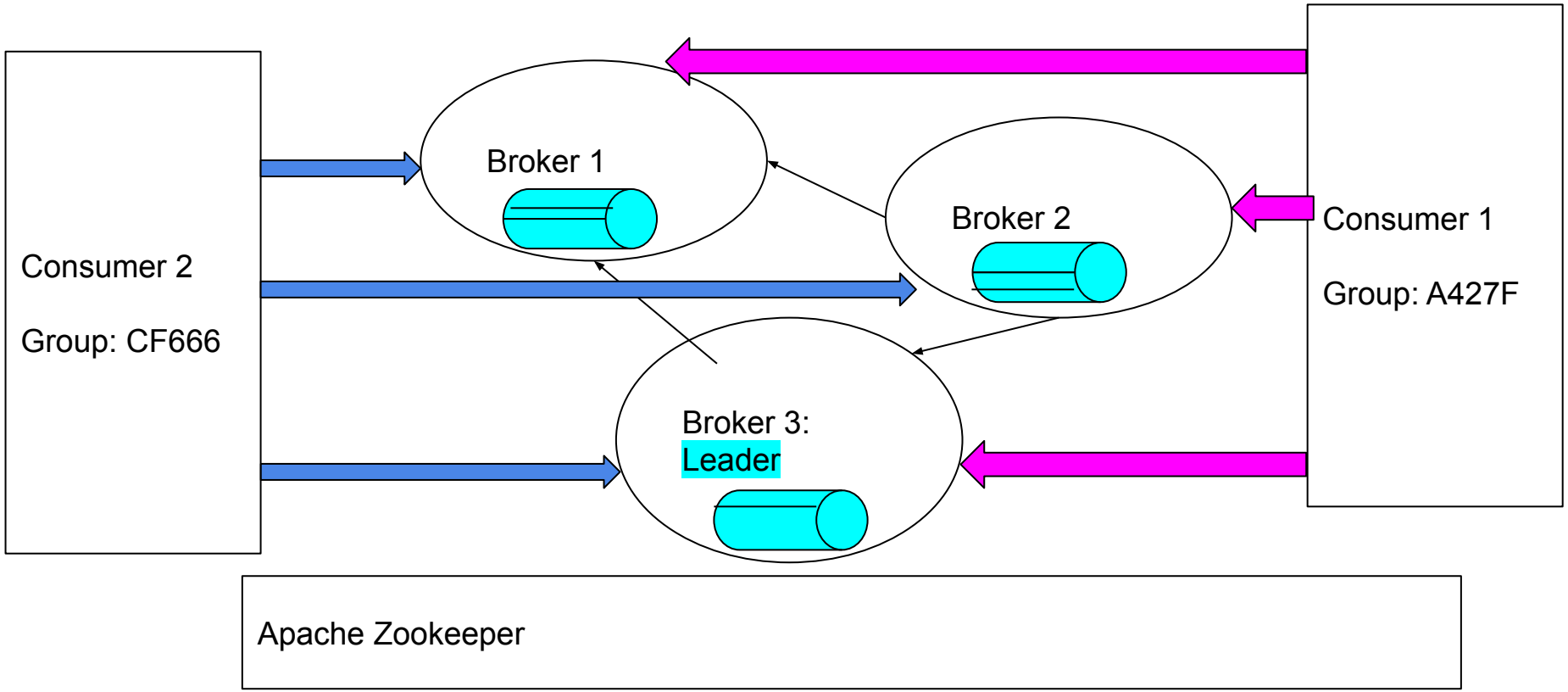
- Mit jedem Release von Kafka wird die Rolle des Zookeepers zurückgefahren
  - Roadmap: “Wir wollen die Zookeeper-Dependency entfernen”

Topics sind partitioniert



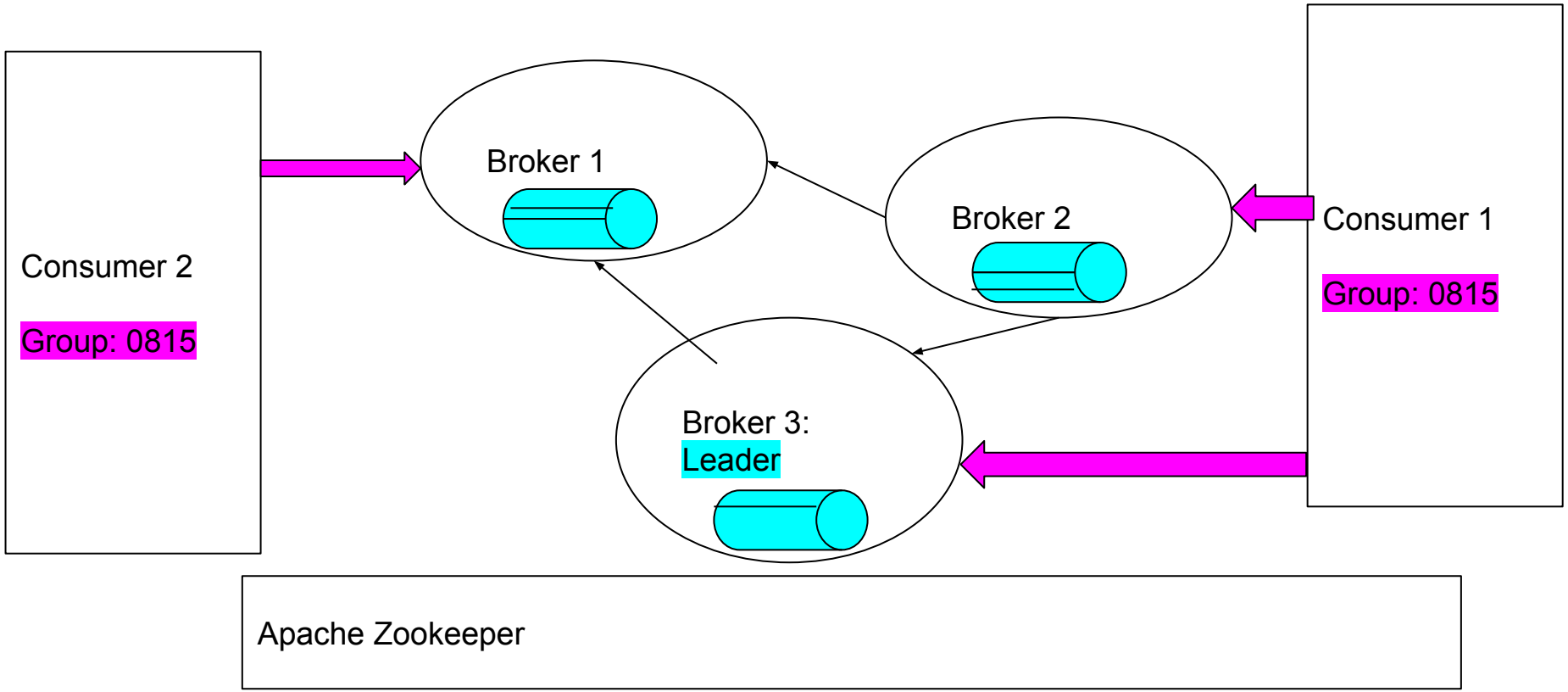
Partitionen und Consumer

integrata cegos



Partitionen und Consumer einer Group

integrata cegos

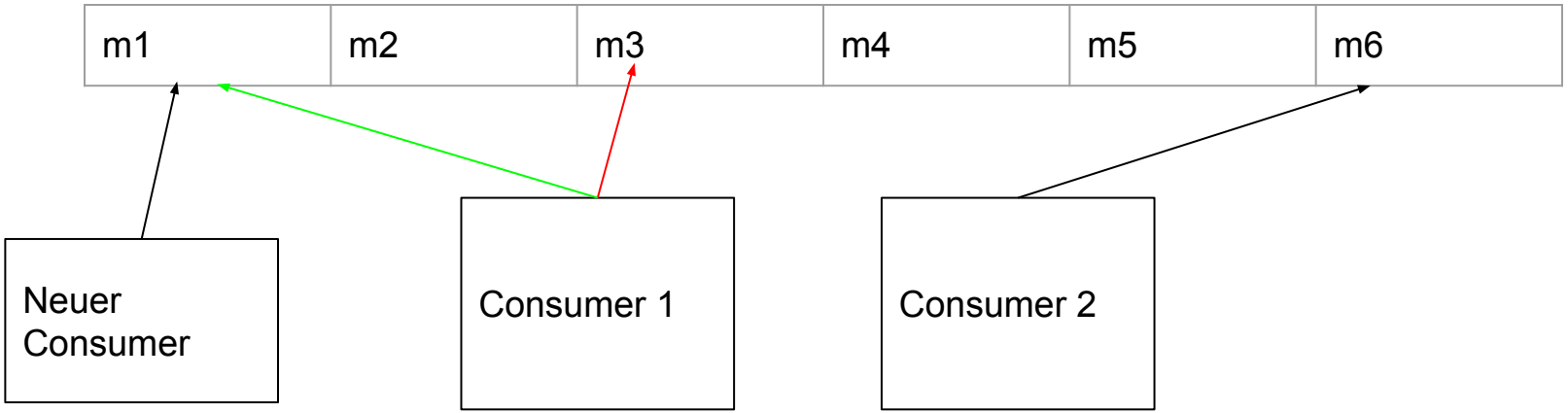


Ausfallsicherheit und Topics/Partitions

- Beim Anlegen des Topics zusätzlich die Angabe eines Replikationsfaktors

Topics/Partitions und Consumer

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



Konfigurierbare Retention

- Alle Nachrichten werden beim Ablauf einer Verweilzeit automatisch gelöscht

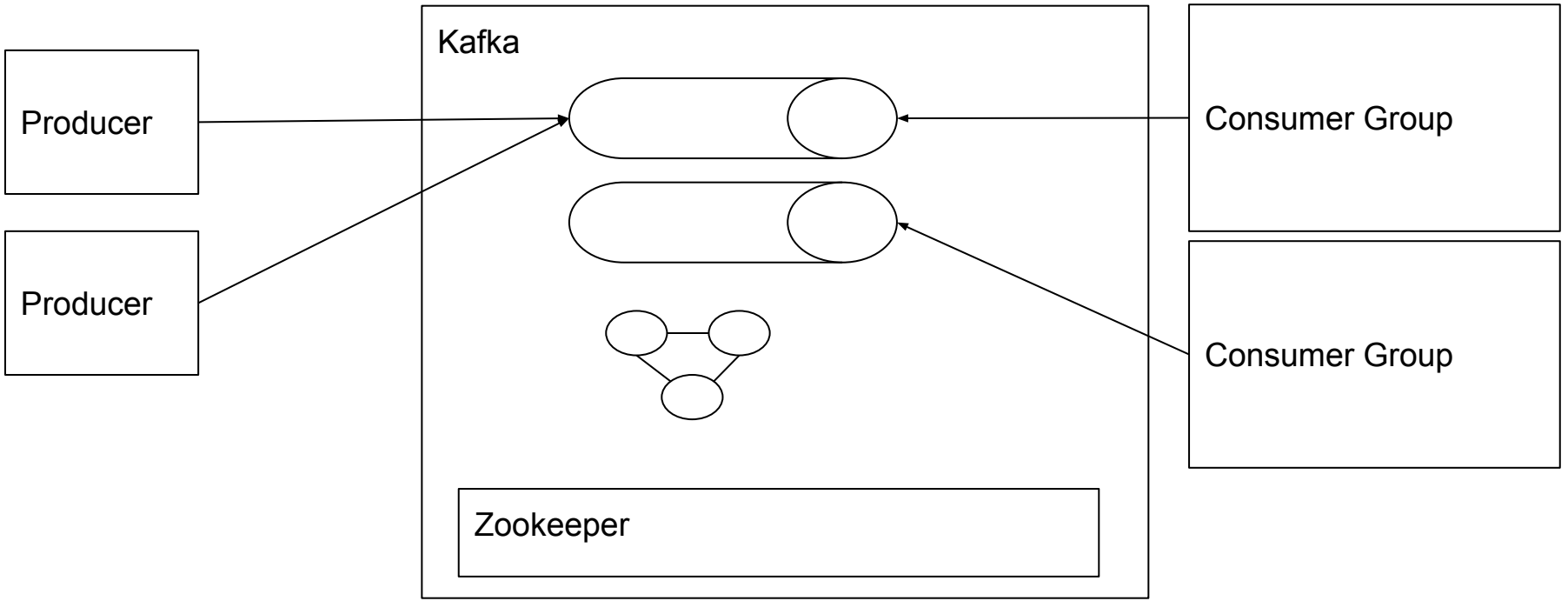
Daten-Konsistenz im Ring-Cluster

integrata cegos

- 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: **A**C oder **A**P

Kafka und Anwendungen

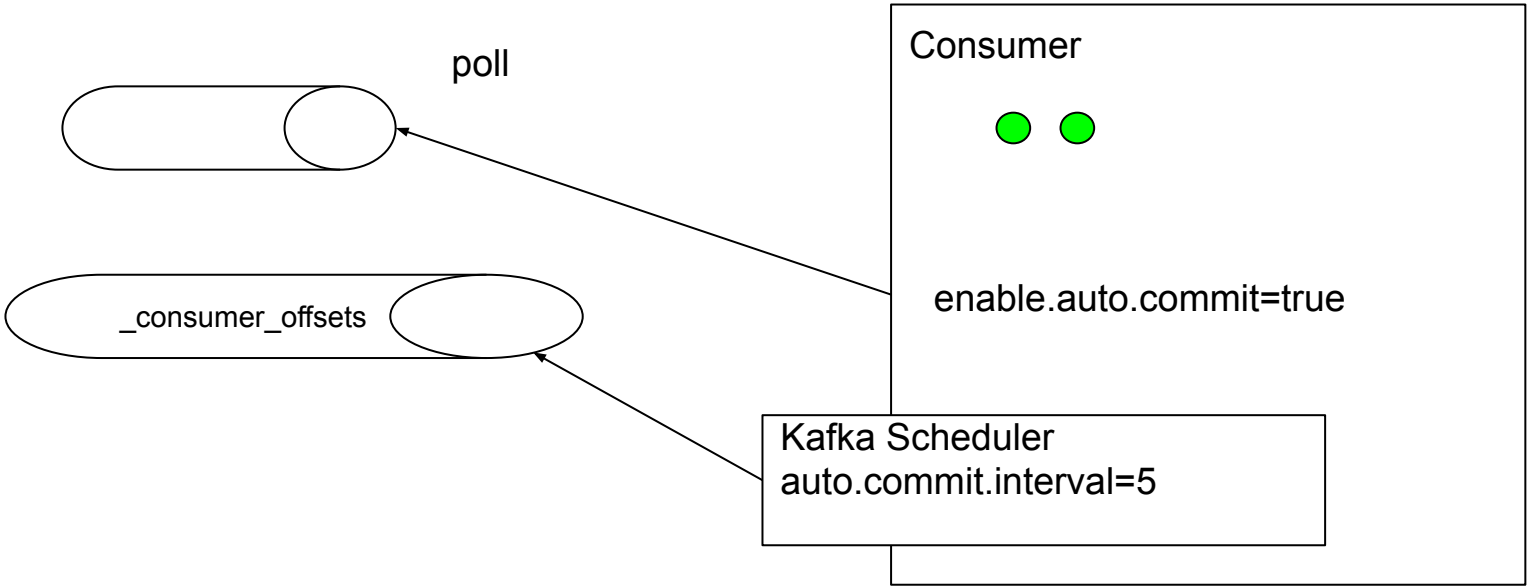
Einfachste Darstellung



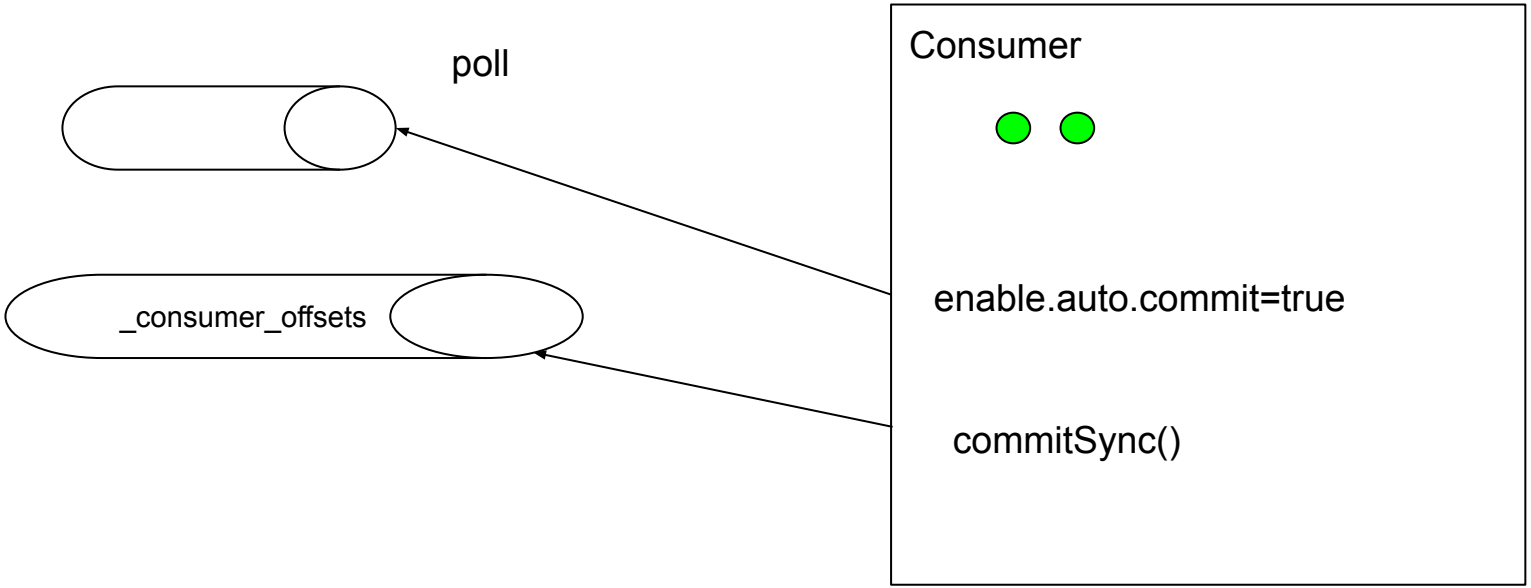
Garantien zur Message Übermittlung

- 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

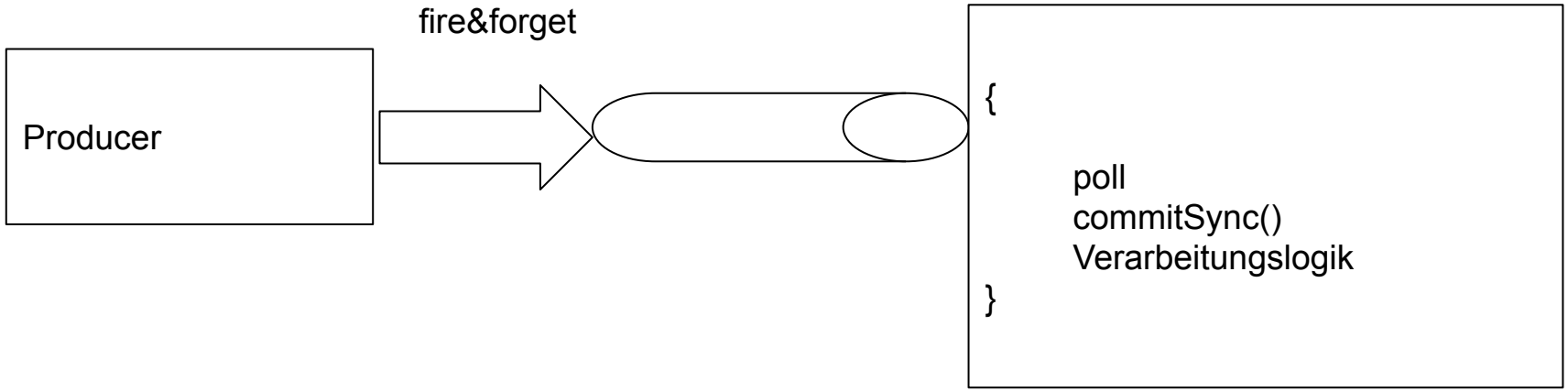
No guarantee



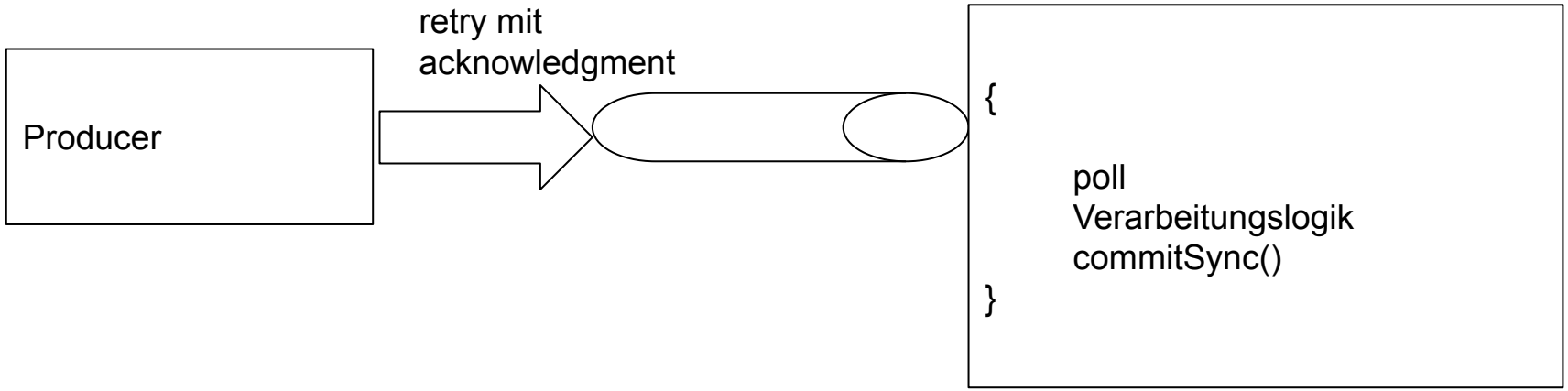
Alles weitere: enable.auto.commit=false



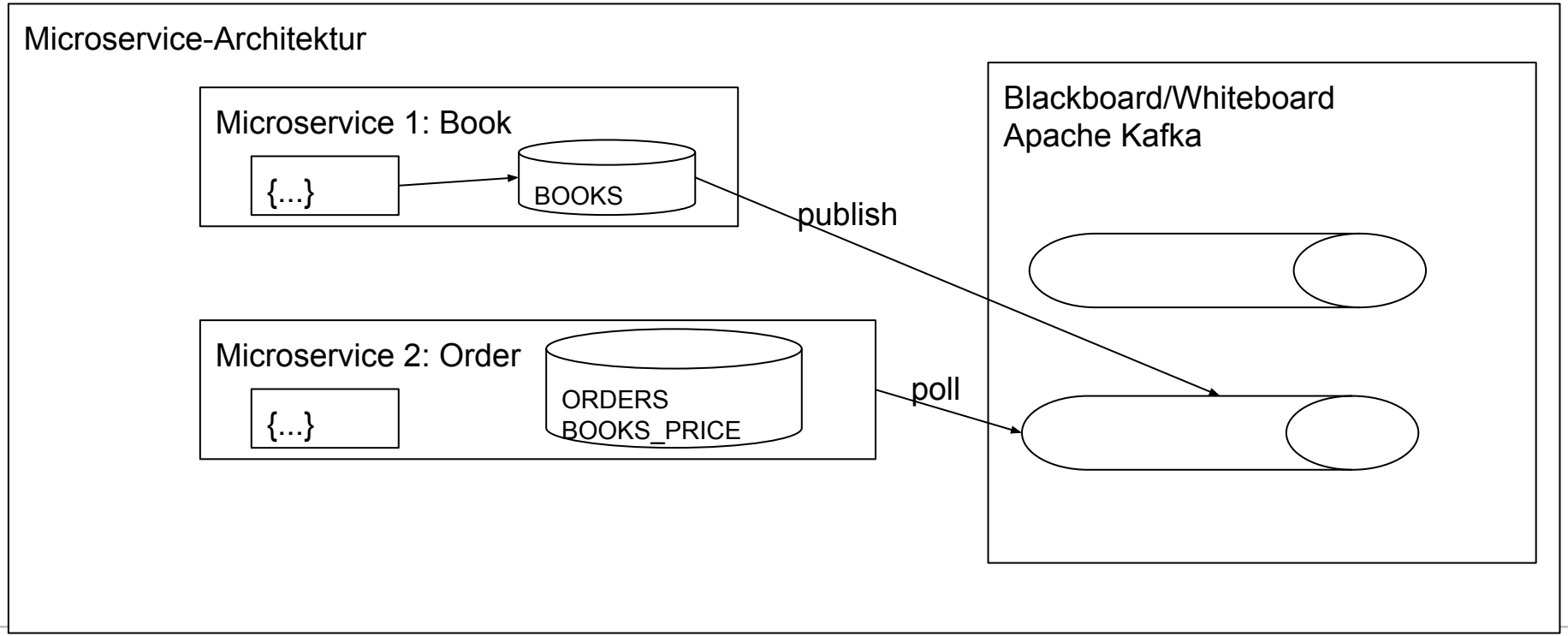
At most once


fire&forget
Producer
{
poll
commitSync()
Verarbeitungslogik
}

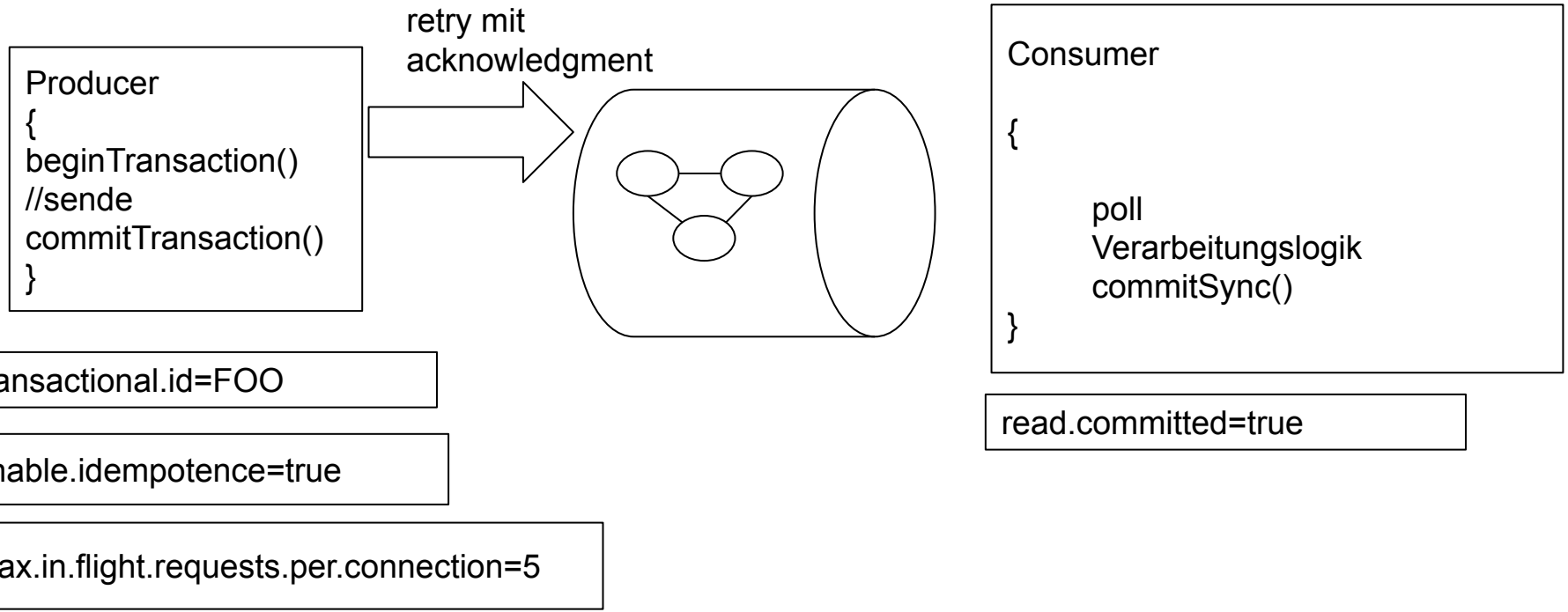
At least once



Warum war/ist Kafka hier so schlampig?



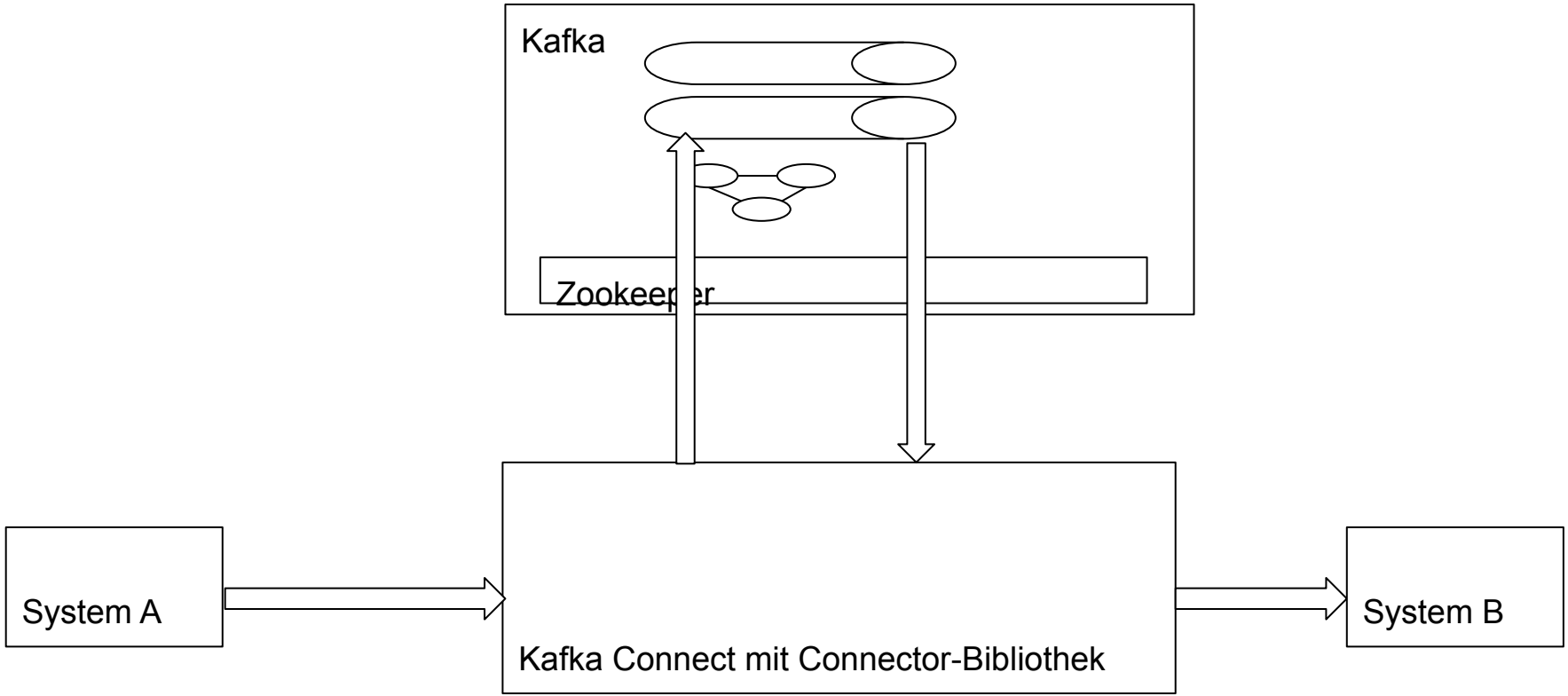
Effectively Once



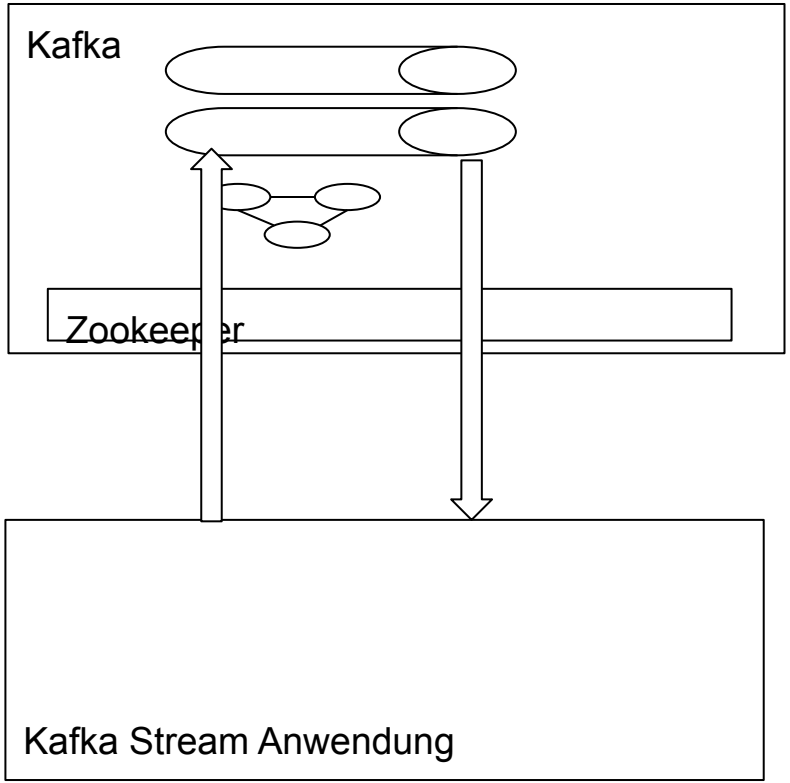
Kafka Streams

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

Kafka Connect



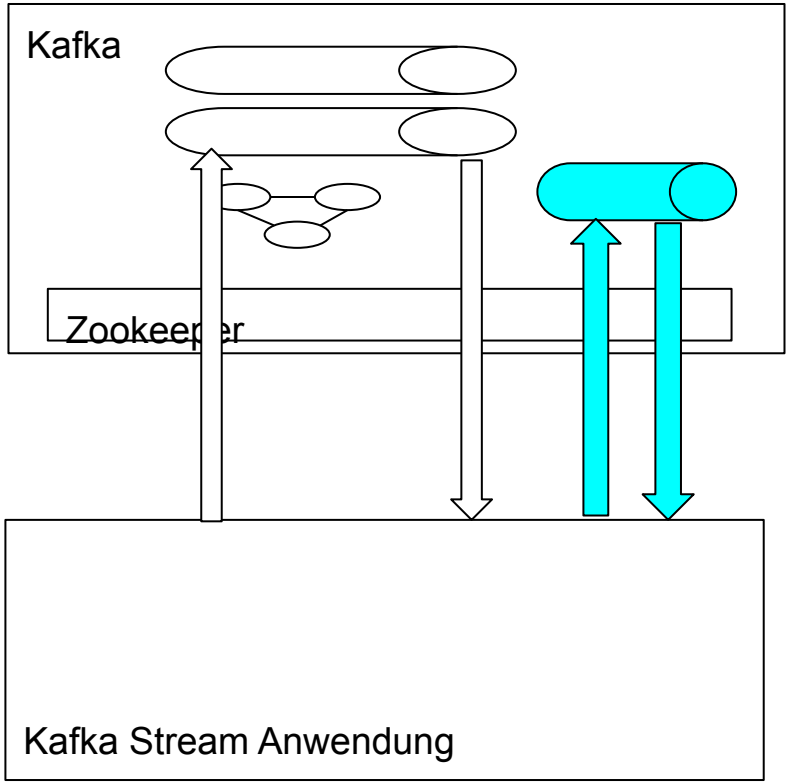
Kafka Streams



Kafka Streams

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

Kafka Streams



Kafka-Clusters

Zookeeper und Kafka-Broker

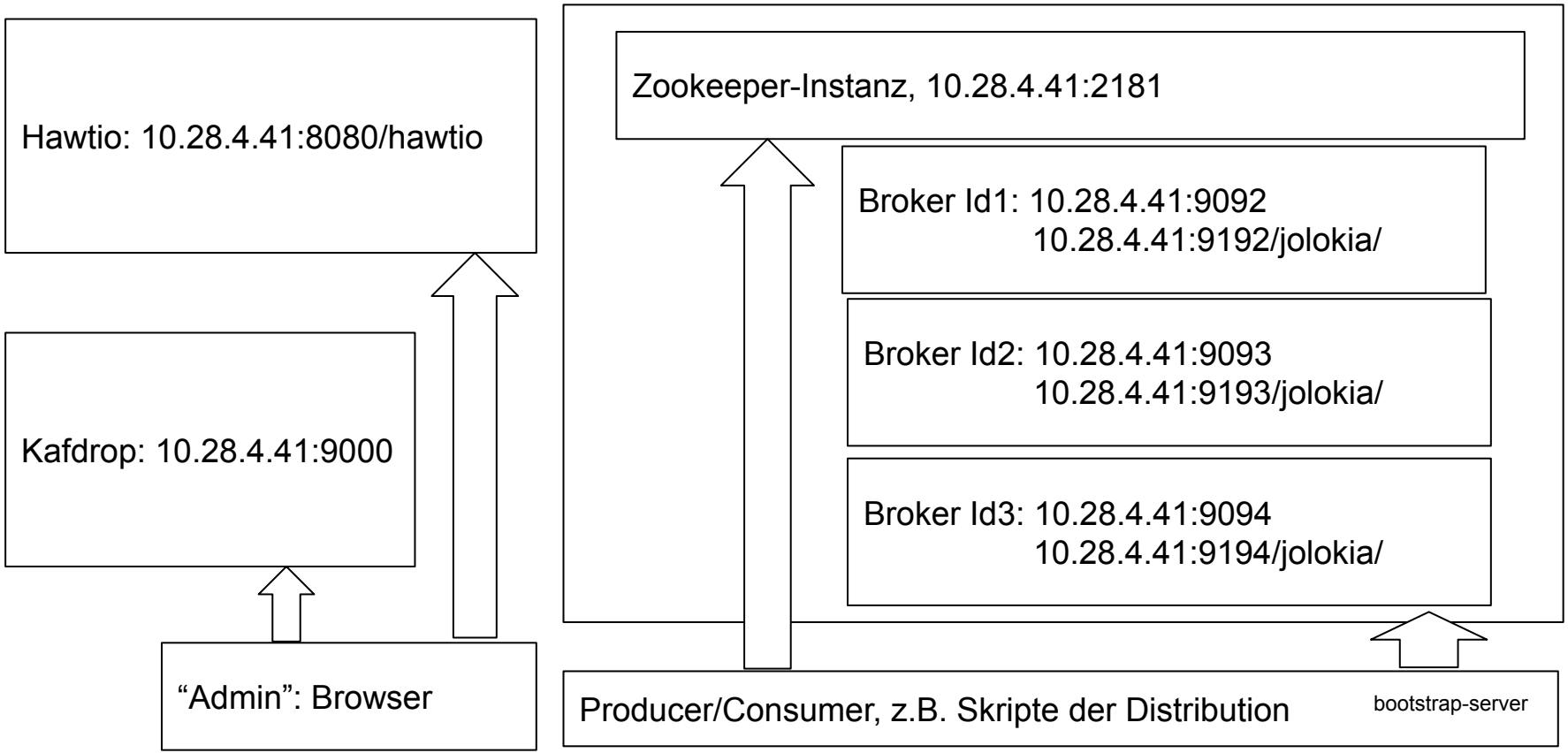
- Vollkommen unabhängig voneinander
- Beide sind Java-Prozesse
- Kafka-Broker benutzt exzessiv das “Offheap”-Memory der Java Virtual Machine
- Beide sind Bestandteil der Kafka-Distribution
  - JAR-Dateien
  - Skripte zum Starten und Stoppen
  - Konfigurationsdateien
- Zusätzlich in der Distribution: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

Weitere Tools der Distribution

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

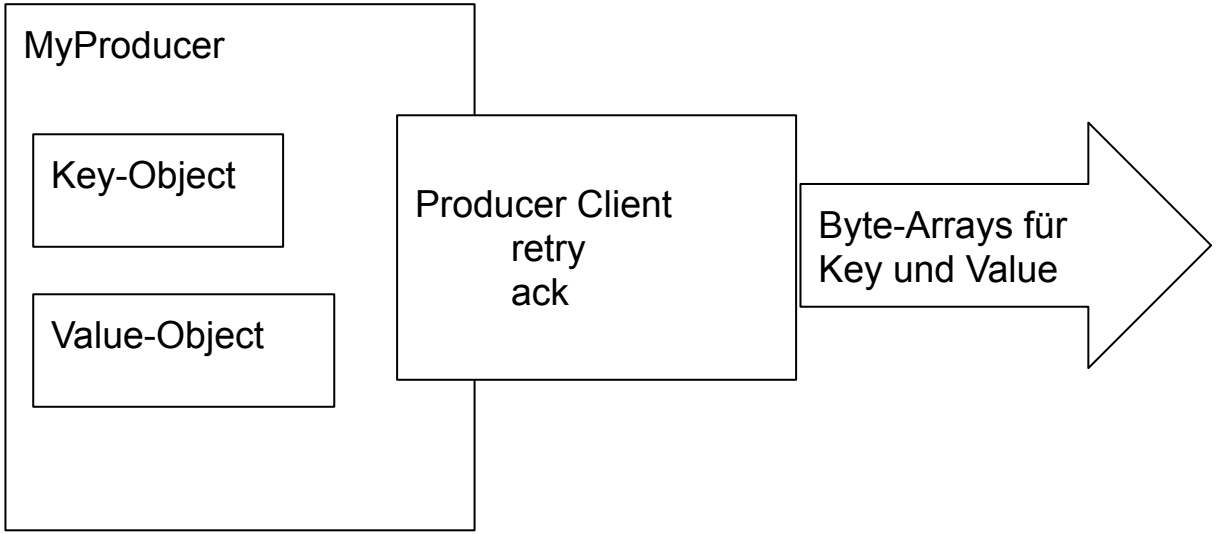
Ein HelloWorld-Kafka-Cluster

integrata cegos



Programmierung

Kafka Client



Acknowledgement

integrata cegos

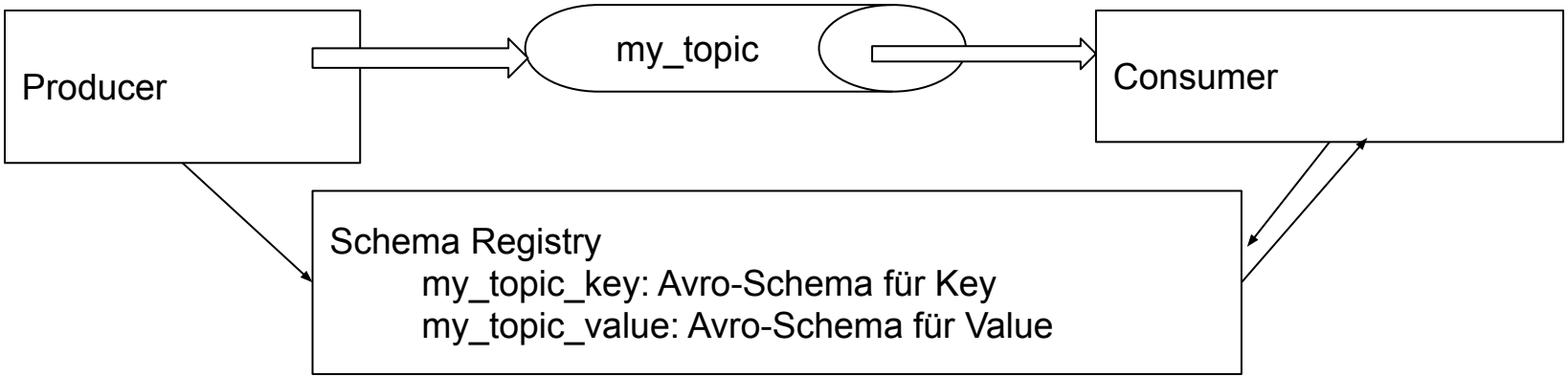
- Wann ist das Versenden der Message aus Sicht des Producers erfolgreich?
- 0
  - Fire & Forget
- 1 (Standard)
  - Die Nachricht ist in einem Broker eingetroffen
  - Nicht notwendigerweise repliziert
- all
  - Die Nachricht wurde auf alle Replikationsserver verteilt
- Was ist mit dem “Quorum”
  - ack = (Anzahl Broker + 1)/2

Zu den Serializers und Deserializer

- Basis-Datentypen werden im Kafka-Standard unterstützt
- (Custom-Serializer sind durch Implementierung einer Schnittstelle möglich
  - “Serde”: Klasse, die sowohl als Serializer oder auch als Deserializer genutzt werden kann)
- Dokumenten-Serializer
  - XML
  - JSON
    - Kein standardisiertes JSON-Schema
  - Apache Avro
    - Serializer
      - Confluent-Implementierung
        - Als Schema Registry kann ein Confluent-Produkt eingesetzt werden
      - Eigene Implementierung/Open Source

Was ist Apache Avro?

- Ein binäres Serialisierungs-Format
- Daten und Schema sind “getrennt”
  - Problematisch: Was passiert wenn Producer und Consumer unterschiedliche Versionen (!) des selben Schemas benutzen?



```
$ ./bin/kafka-consumer-groups.sh --bootstrap-server 10.28.4.41:9092
--all-groups --describe
```

https://blog.serverdensity.com/how-to-monitor-kafka/

Ideensammlung ToDo

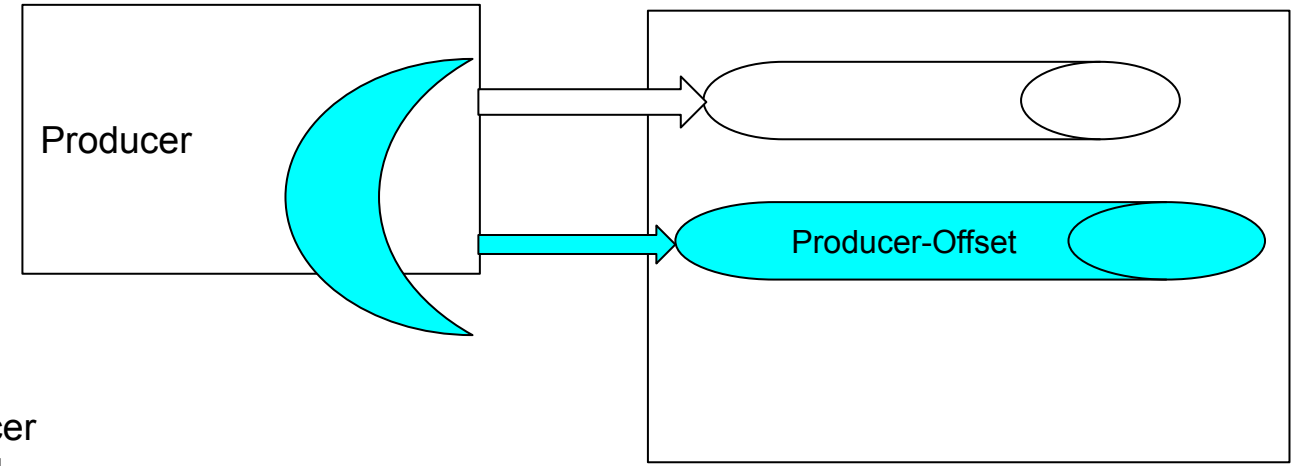
- Einblick ins JMX via Hawt in einen Broker
- Consumer-Prozess über die lokale jconsole überwachen

Zu Effectively Once

- Grundlegende Technologie: Transaktionen und Idempotenz
  - Neu mit dazu: Producer-ID
- Alle Nachrichten werden im Rahmen einer Transaktion sofort zum Kafka-System übertragen
  - Pro Nachricht: Eindeutige Id der Nachricht, Transaktions-ID + Producer-ID
    - Pro Producer-ID nur eine einzige Transaktion möglich
    - Falls unter dieser ID eine neue Transaktion gestartet wird wird Kafka die bisherigen, noch nicht committeten Messages verwerfen
- Transaktionen sind “Partitions-übergreifend”
  - beinhaltet: Topics

Blaupause für Producer

DataSet
Mehrere Datensätze
1.
2.
3.



Im Ausfall-Fall muss der Producer wieder von Vorne anfangen und überträgt Nachrichten doppelt
Transaktionen können garantieren, dass erst nach vollständigem Auslesen des DataSet die Nachrichten verarbeitet werden

Producer kann seinen State = Counter der erfolgreich übertragenen Nachrichten sichern

Blaupause für den Consumer

- Autocommit disabled
- READ_COMMITTED
- commitInTransaction

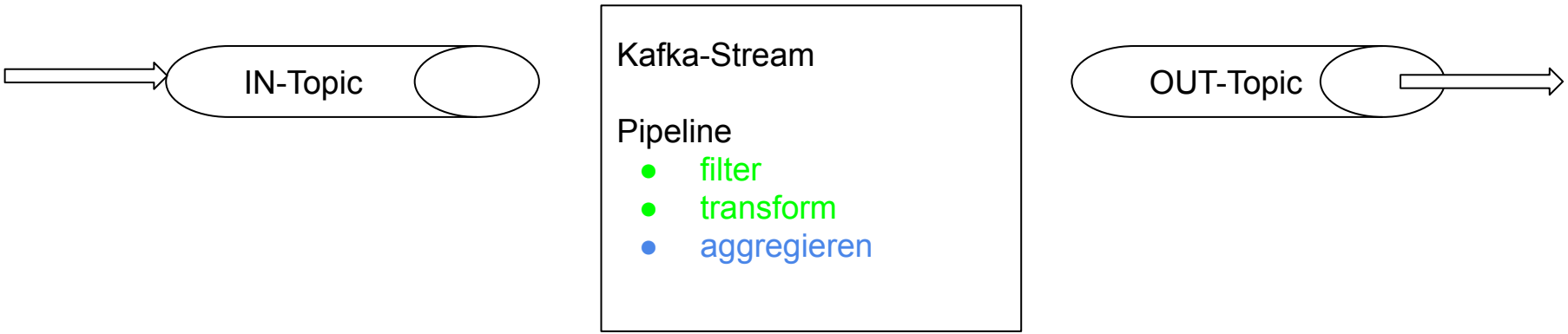
Kafka Connect

- Framework zur Implementierung von Zustands-behafteten Producern und transaktionellen Consumern
- Fertige Implementierungen
  - Kafka-Distribution enthält FileSource und FileSink
  - Confluent bietet einen reichhaltigen Satz Lizenz-pflichtiger Connectors
    - JDBC, JMS, …
  - Open Source Community

Kafka Streams

integrata cegos

- Neues API: Vereinigt Producer und Consumer



Pure Operations
benötigen keinen State

Stateful Pipeline
OUT-Topic hält das Ergebnis-Aggregat