

Apache Kafka

Apache Kafka, Linkedin fonlamasıyla beraber Jay Kreps önderliğinde geliştirilmiştir.

ilk sürüm 2011 yılında piyasaya çıkmıştır.

İsmini 2012 yılında Franz Kafka'dan almıştır. Jay Kreps, Franz Kafka'nın romanlarını yazarken uyguludağı disiplinini ve optimize olmuş çalışma biçimini benimsemiştir ve geliştirdiği ürüne Kafka ismini vermiştir.

Scala ve Java ile yazılmıştır.



Apache Kafka Nedir?

Apache Kafka tıpkı RabbitMQ gibi bir message broker sistemidir.

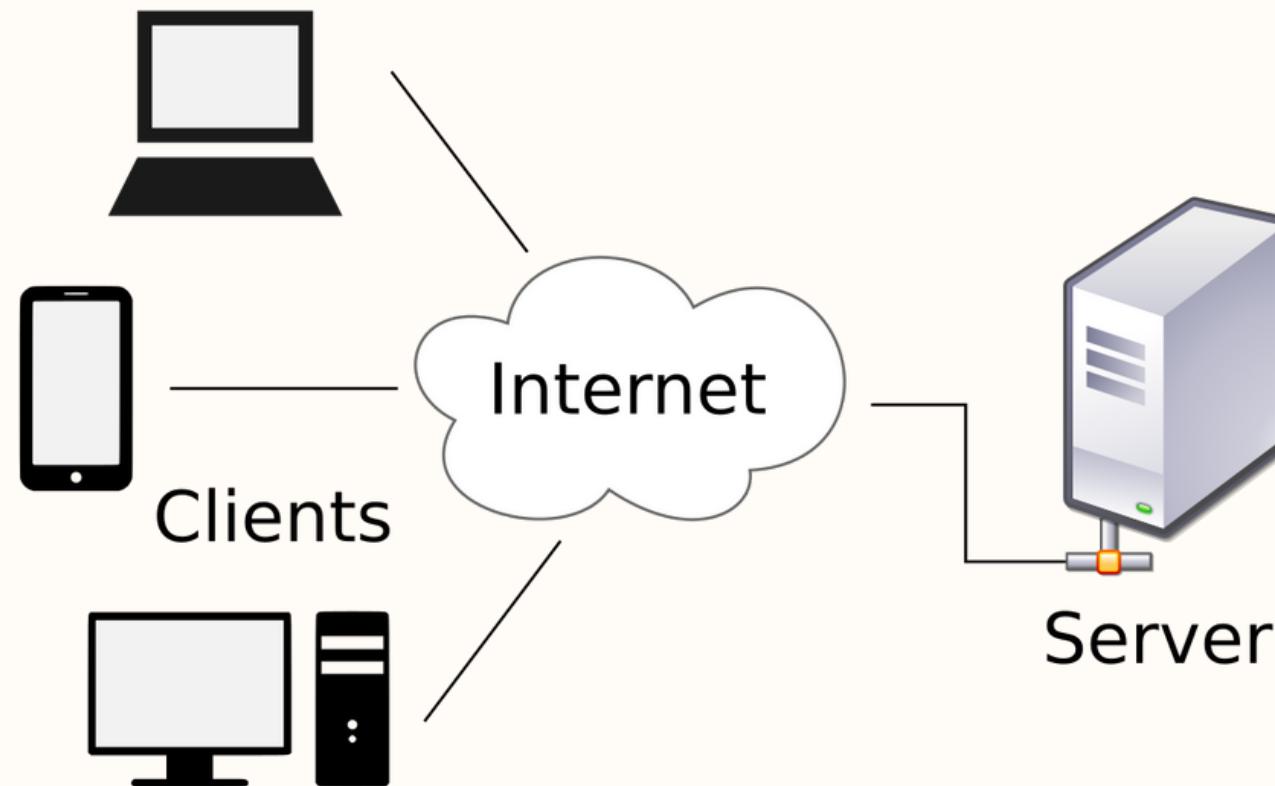
Bir mesajı alır ve bir yere depolar. Mesajı okuyacak kişiler de okur. Mesajı ilgili yerlere dağıtmaya, kuyruklama yapısıdır.

Apache Kafka'nın ne olduğunu daha iyi anlamak için geleneksel mimarilerden bahsedelim.

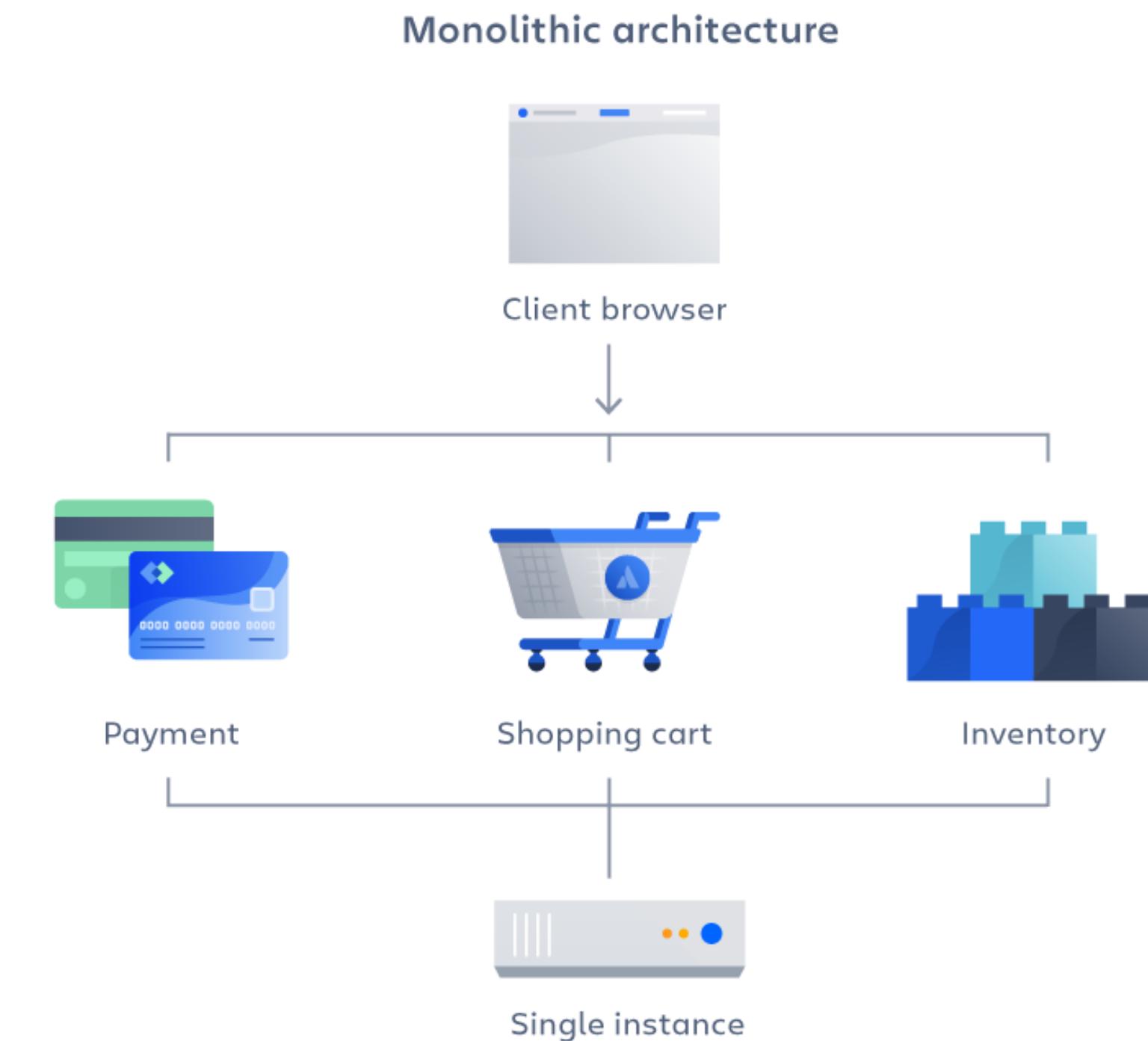
Apache Kafka Nedir?

1970-1980 mainframe devri olarak karşımıza çıkıyor.

90'lı yıllarda popüler mimari olarak client server mimarisi vardı. Bilgisayardaki bir client uygulama, server taraflı uygulamaya farklı protokoller kullanarak erişim sağlardı. Bu zamanlar da server taraflı uygulama monolith(tek parça) bir uygulamaydı.



Monolithic Architecture



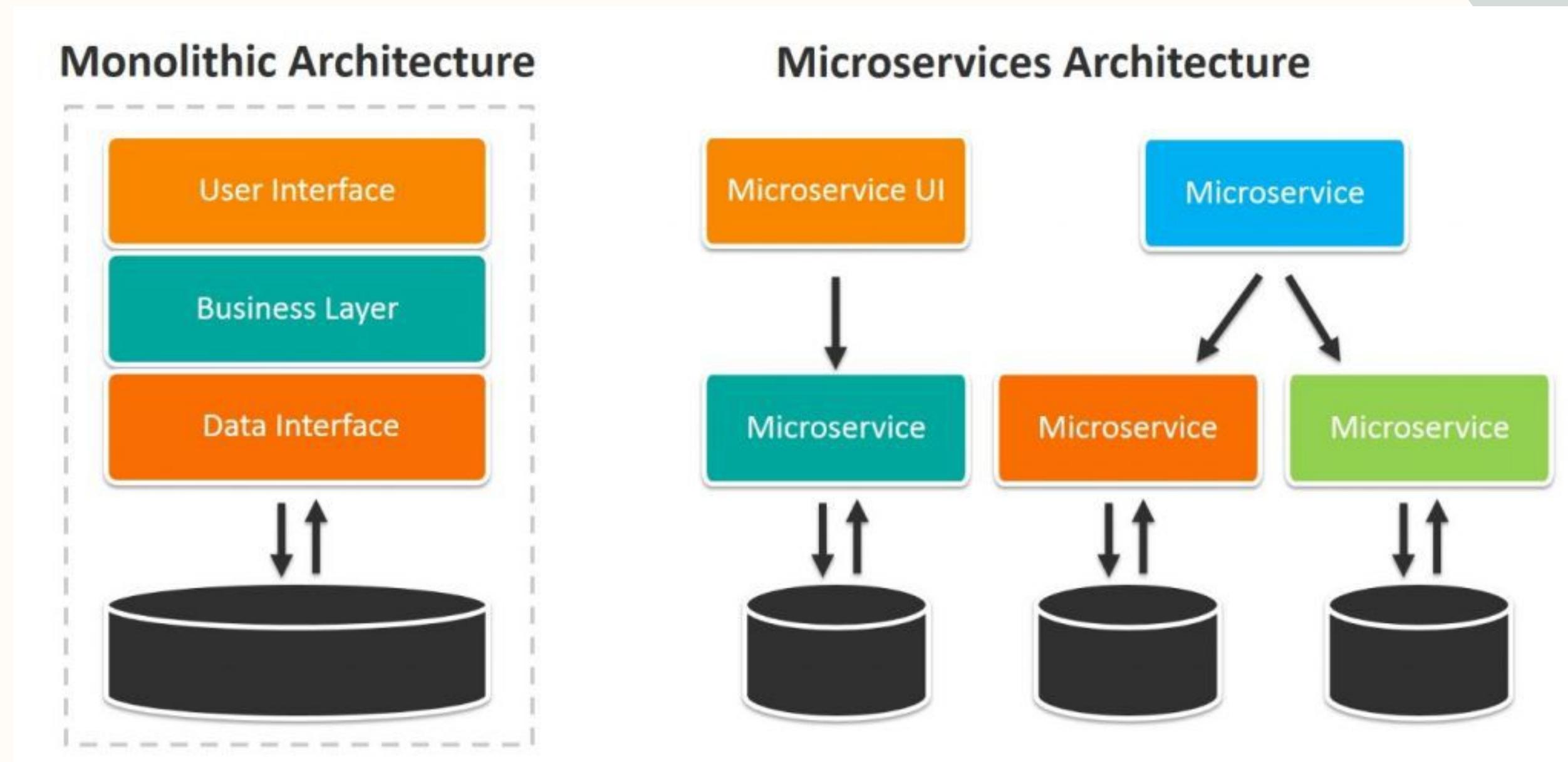
Microservice Architecture

İnternetin popüler hale gelmesi ve time to the market denilen pazara daha sık sürümler çıkarma gereksinimi ile monolith yapıdan uzaklaşıldı.

Artık uygulamalar küçük ve harici çalışan parçalar halinde tasarılmaya başlandı. Bunlardan en popülerleri mikroservislerdir.

Bu servisler dağıtık olarak aynı servler'larda çalışmaya başladı.

Microservice Architecture



Microservice Architecture

Eskiden tek bir uygulama artık dağıtık bir şekilde çalıştığı için farklı problemler oluşmaya başladı.

Peki bunca mikroservis birbiri ile birlikte nasıl çalışacaklar? Hani protokoller kullanılacak? (FTP, UDP,TCP,HTTP)

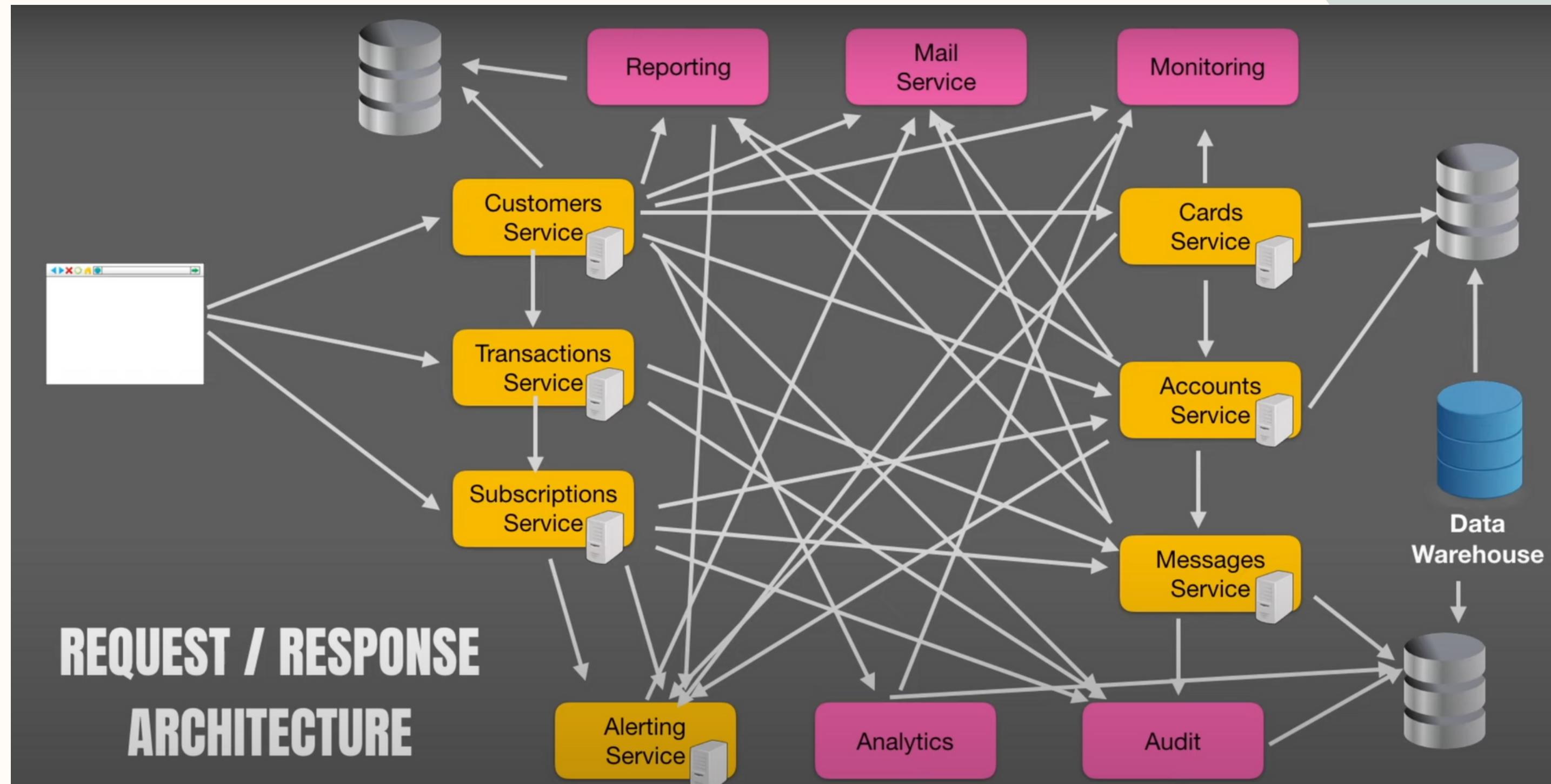
Bilgi akışı hangi formatta olacak? (XML, JSON, CSV)

Bunların yanında yazılım sektörü gelişikçe farklı gereksinimler ve ihtiyaçlar ortaya çıkmaya başladı. Bu ihtiyaçlar uygulamalara bazı komponentlerin eklenmesi gereksinimini doğurdu.

Bunlardan bazıları; reporting, mail service, monitoring, analytics, audit

Her komponent ve mikroservislerin birbirleri ile haberleşmesi uygulamaların içinden çıkışsız bir hale gelmesine yol açtı.

Request/Response Architecture



Event Driven Architecture

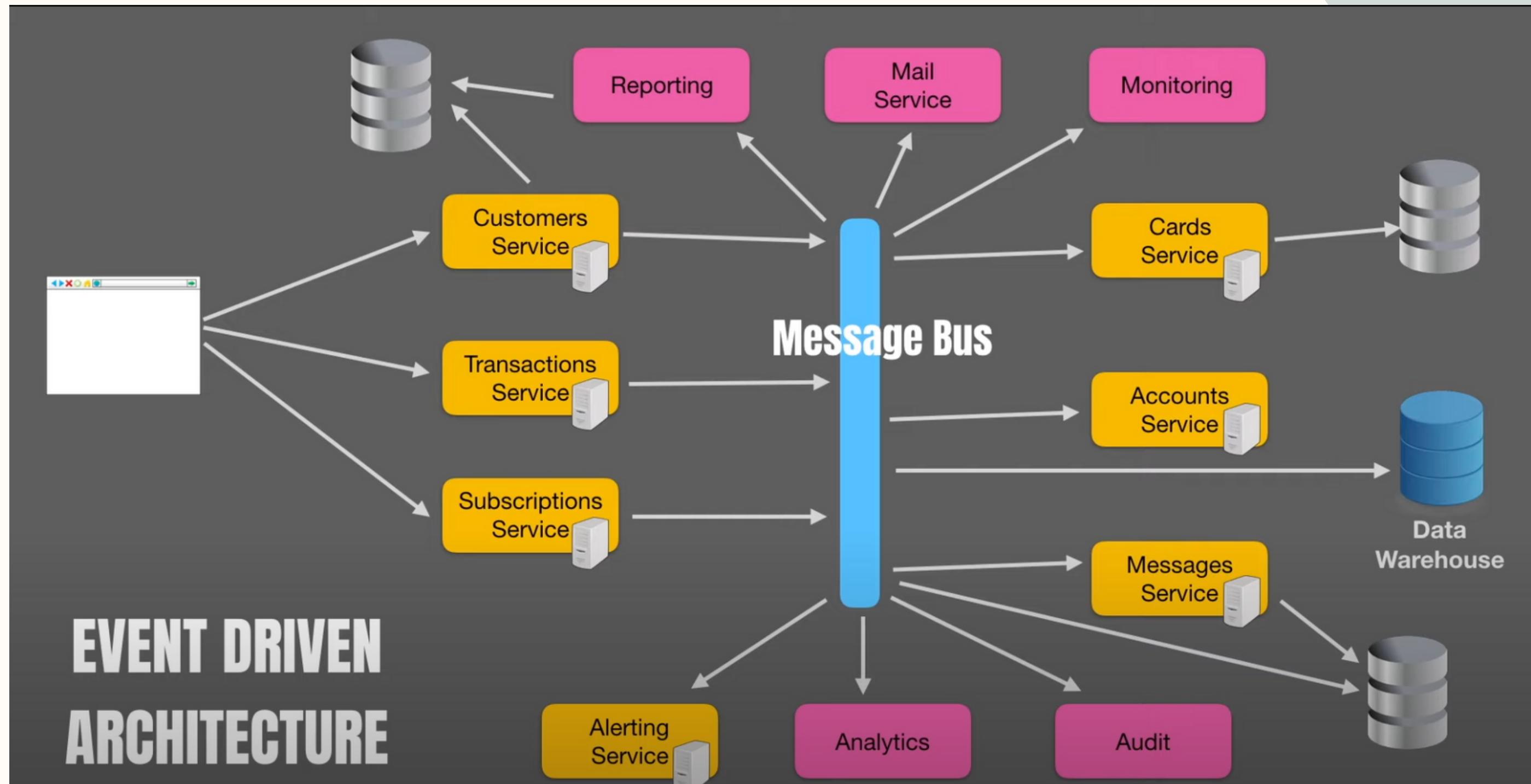
Bu mimari de bir servisin devre dışı kalması, zincirin zayıf halkası gibi ağdaki tüm servislerin devre dışı kalmasına yol açabilir.

Bu karışıklılığı önlemlemek için event driven architecture adında bir mimari ortaya çıktı. Bu uygulamanın mümkün kılınabilmesi içinde Message Bus uygulamaları geliştirilmeye başlandı.

Artık uygulamalar birbirleri ile doğrudan değil, Message Bus'lar üzerinden haberleşmeye başladı. Bunu da event denilen mesajlaşma kavramı ile yaptılar.

Event, geçmişte belirli bir zamanda oluşmuş bir şeyi tanımlamak için kullanılır.

Event Driven Architecture



Event Driven Architecture

Bu message bus adı verilen yapının sektörde bir çok karşılığı vardır. En popülerlerinden biri ise Apache Kafka'dır.

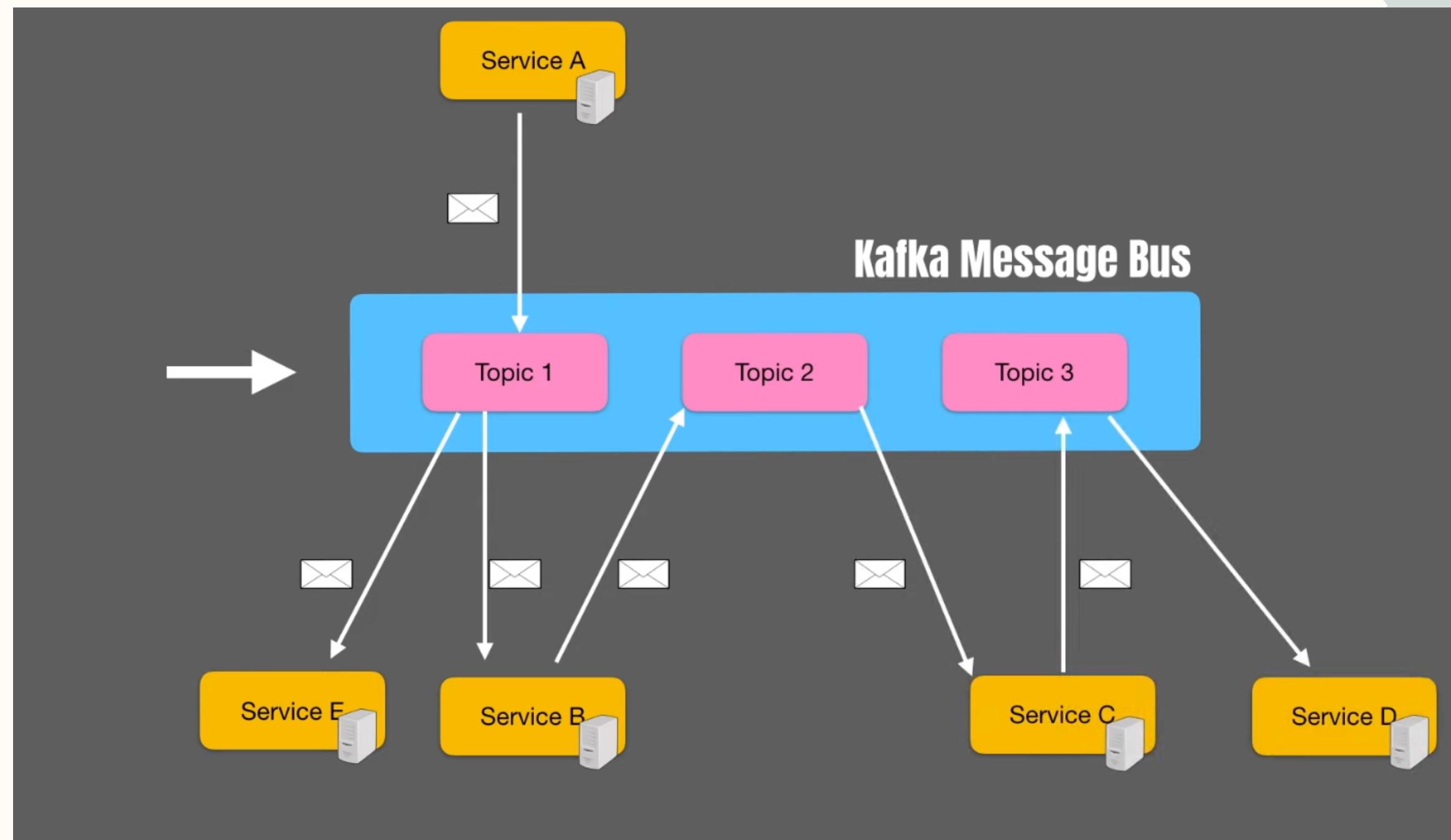
Apache kafka bir stream işleme platformudur. Tamamen distributed bir yapıdır.

Saklı tuttuğu eventleri kaybetmemek için arkasında resilient architecture kullanır ve retention ayarı yapmamıza imkan verir. (yani veri 3 gün saklansın, 5 gün saklansın gibi ayarlar yapabiliriz)

Ayrıca verinin saklanması sayısına da müdahale etmenize olanak verir. Yani veri 3 kere saklansın denilirse 3 farklı sunucu üzerinde aynı veri saklanır ve sunuculardan biri çökse dahi veri kaybolmamış olur.

Apache kafka yüksek derecede yüksek derecede ölçeklenebilirdir. (fully scalable)

Apache Kafka



Apache Kafka ile Neler Yapılabilir?

- Fraud and anomaly detection.
 - Müşterilerin yaptığı her hareketi event olarak kafka da saklarsanız ve fraud detection servisleriniz ile bütün aykırıları saptayabilirsiniz.
- Recommendation engine
 - Web sitenizde kullanıcıya tavsiyeler sunmak için kullanabilirsiniz.
- Monitoring / Metrics
 - Ağdaki bir çok sunucuyu hardware ve software olarak anlık monitörleyebiliriz.
- Activity Tracking
 - Her türlü aktiviyeyi izlemek için kullanabiliriz.
- Integrate systems
 - Ağdaki farklı sistemleri birbiri ile entegre edebiliriz.

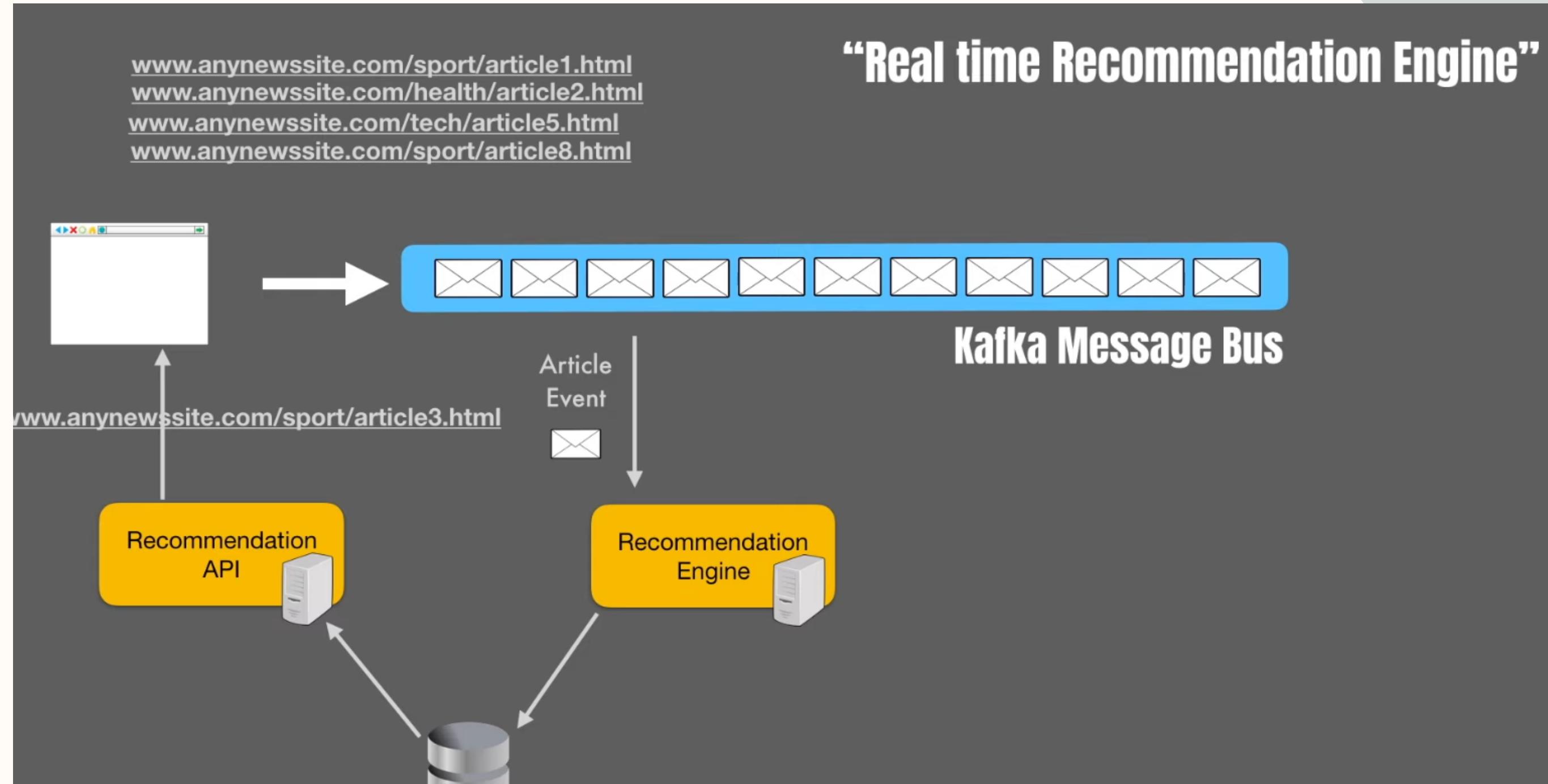
Real Time Recommendation Engine

Bir haber portalımız olduğunu düşünelim.

Kullanıcıların yaptığı tüm okumaları message bus üzerine kaydedelim.

Bu sayede kullanıcının yaptığı tüm okumaları okuyup makine öğrenmesi ile bizlere benzer haberler sunan bir servisten faydalananarak kullanıcıya real time olarak yeni ve benzer haber önerilerinde bulunabiliriz.

Real Time Recommendation Engine



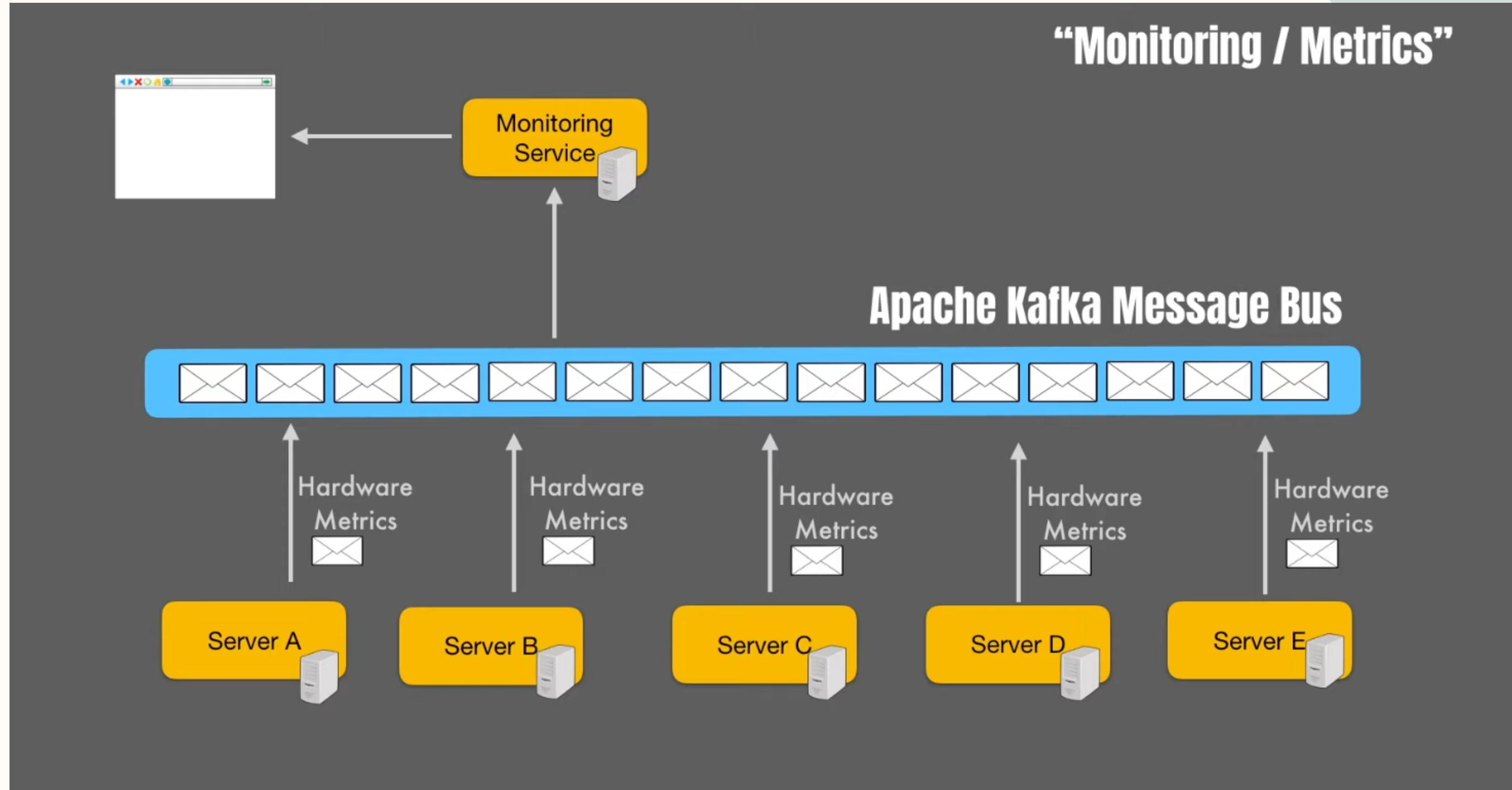
Monitoring / Metrics

Tüm sunuculara bir uygulama kurup her saniye içerisinde cpu, memory kullanım bilgilerini apache kafka message bus üzerine kaydedelim.

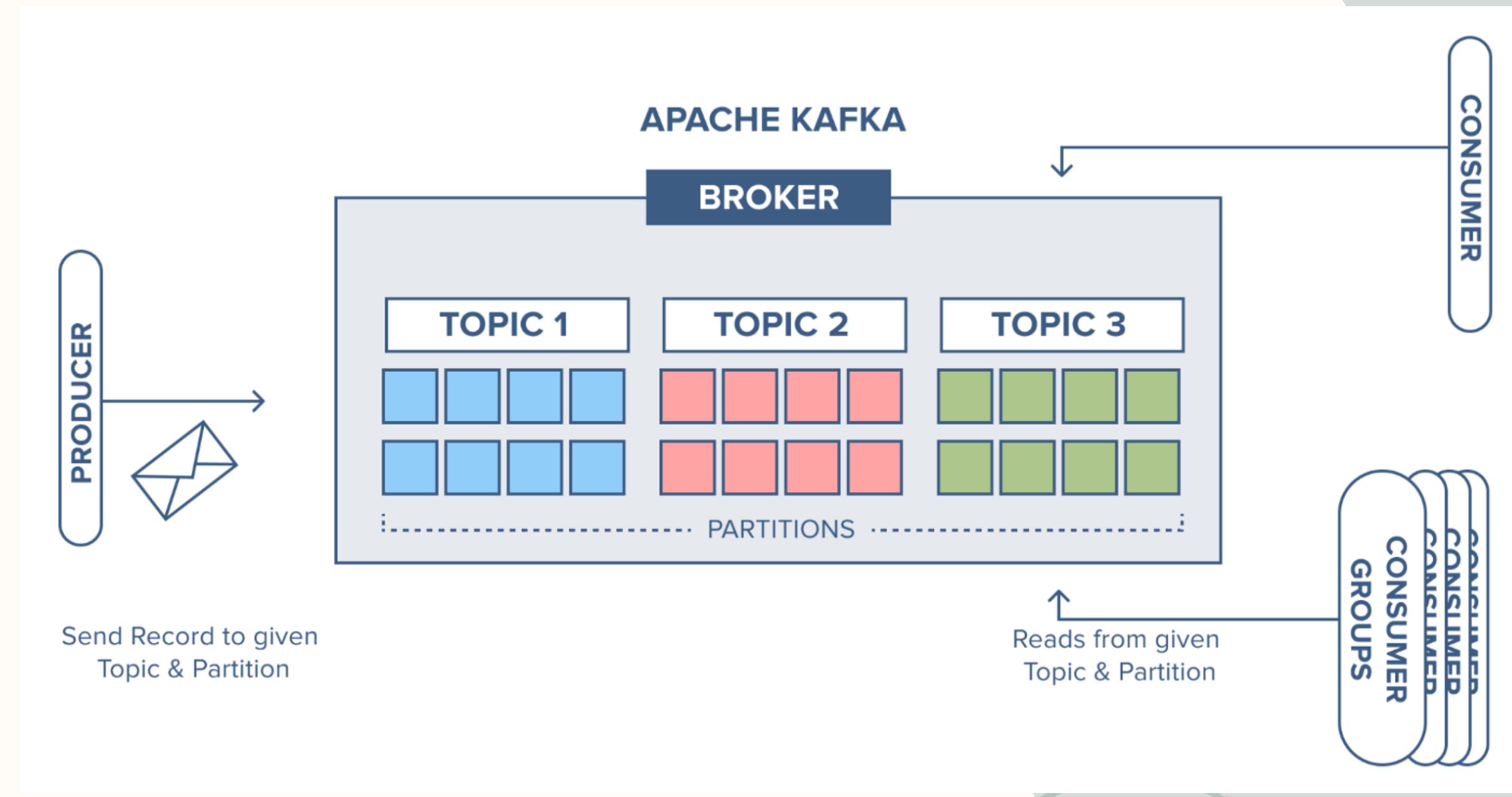
Monitoring servisi kafkaya yazılmış her bilgiyi okur ve analiz eder.

Okuyup analiz ettiği bilgileri ise farklı bir api yardımıyla kullanıcılar sunar.

Monitoring / Metrics



Apache Kafka Mimarisi



Apache Kafka Mimarisi

Kafkanın mimarisinde broker, zookeper, producer ve consumer bulunur.

Her kafka cluster bir veya daha fazla broker'dan oluşur. Her broker birbirine bağlı olarak çalışır ve hepsinin bir id'si bulunmaktadır. Gönderilen veriler broker'larda harddisklerde saklanır.

Zookeper ise apache lisanslı açık kaynak kodlu bir yazılımdır. Kafka bunu tüm broker'ları yönetmesi için kullanır. Gönderilen veriler burada saklanmaz. Zookeper'in görevleri arasında broker'ları koordine etmek, lider partition seçimi, broker'ların birbirini tanımı, yeni veya çökmüş brokerların veya silinen veya eklenen topiclerin keşfedilmesi gibi görevleri vardır.

Producer kafkaya veri yazar.

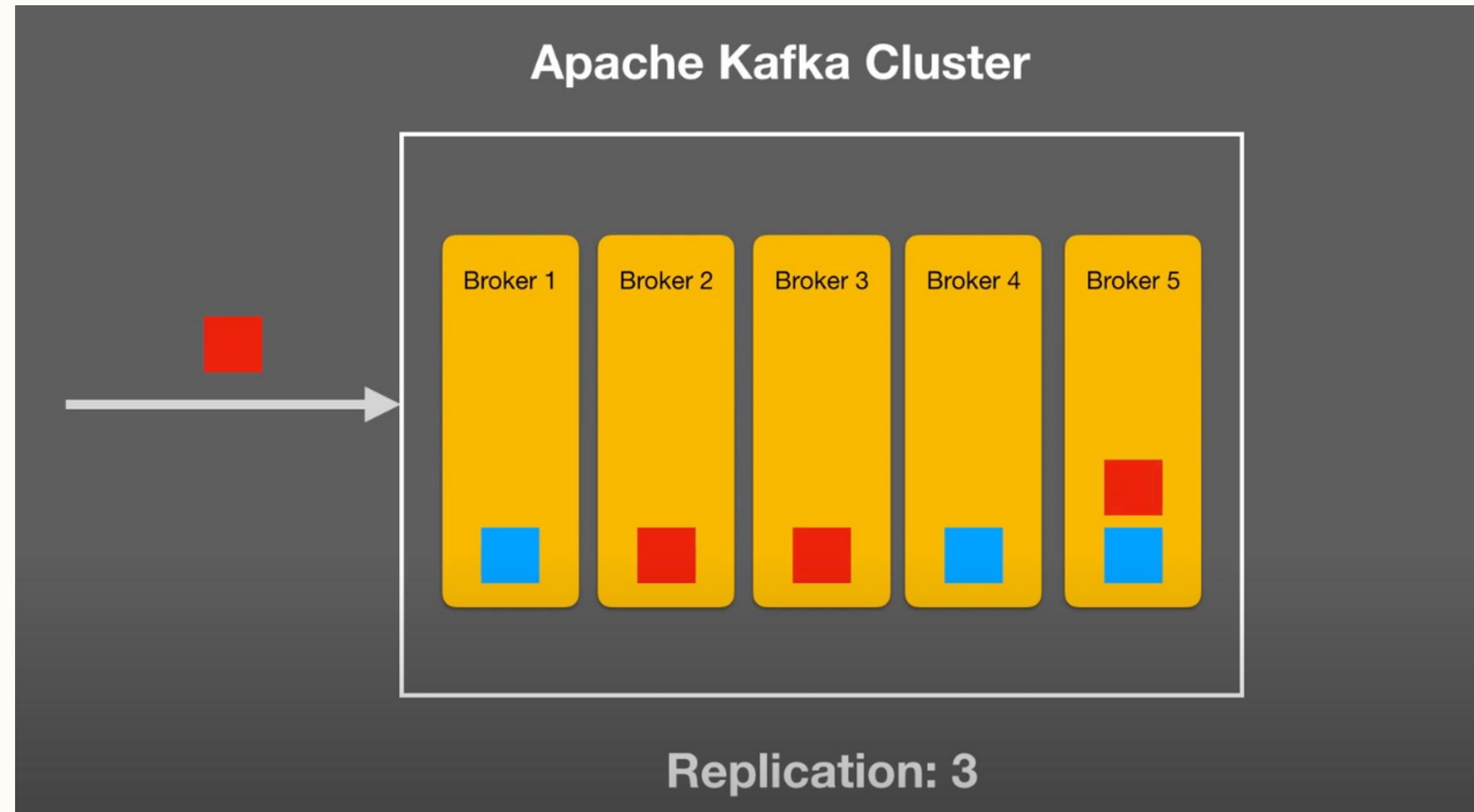
Consumer ise kafka daki veriyi okur.

Live ortamda tekil sayıda broker ve tekil sayıda zookeper kullanmak gereklidir. Bunun sebebi replikasyon işlemini sağlıklı yapabilmek ve split brain gibi sorunların önüne geçmek içindir.

Replication Problem

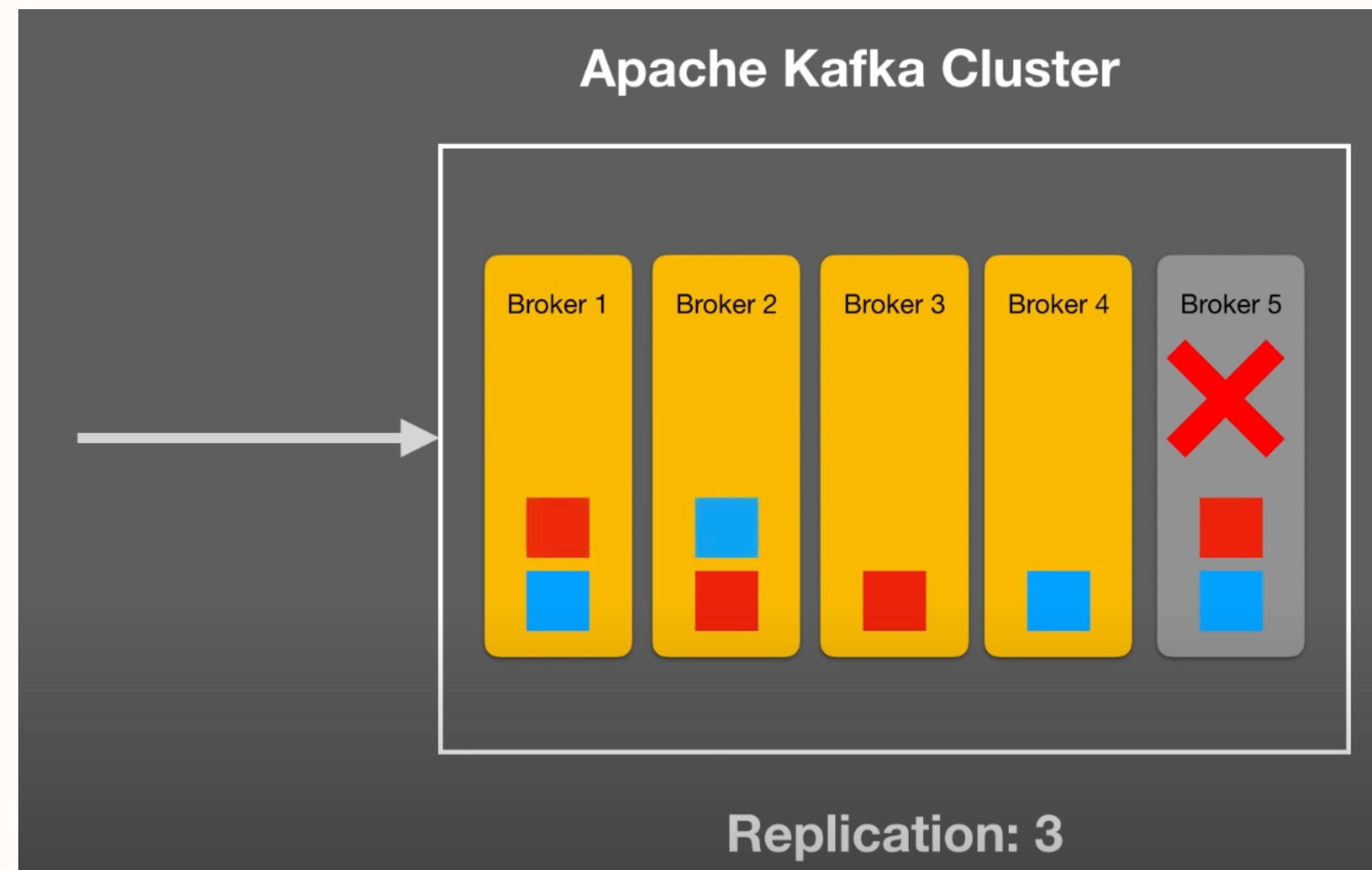
Replikasyon bir verinin kopyasının yapılp çoğaltılmasıdır.

5 broker'lı bir sistem düşünelim. Her verinin 3 replikası olsun yani her veri rastgele 3 broker'da tutulsun.



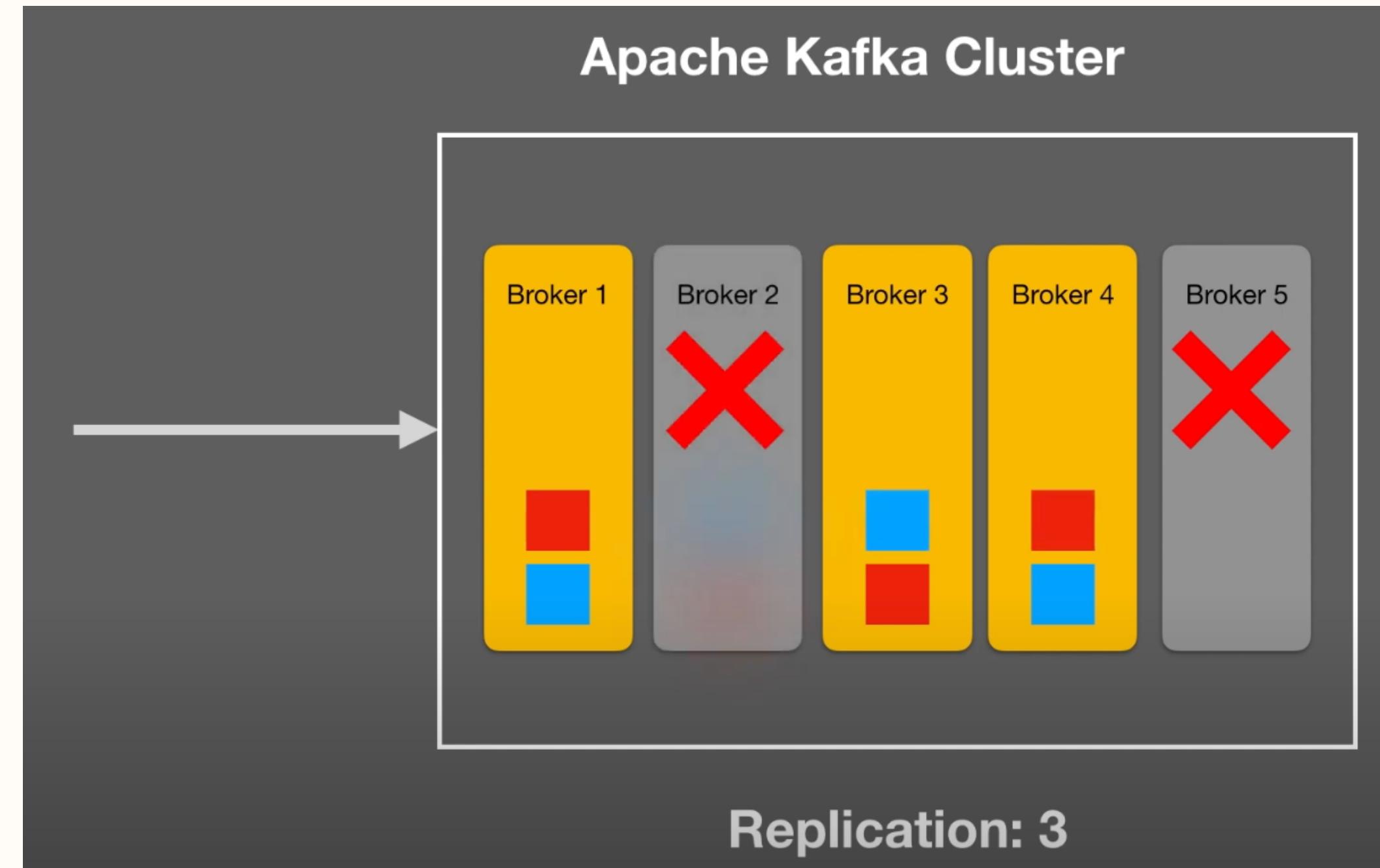
Replication Problem

Bir anda broker 5 çöksün. Bu durumda replikasyon sayımızın 3 olması gerektiği için, kafka hemen broker 5'deki verileri alıp replikasyon sayısına uygun şekilde başka broker'lara replikasyon yapar.



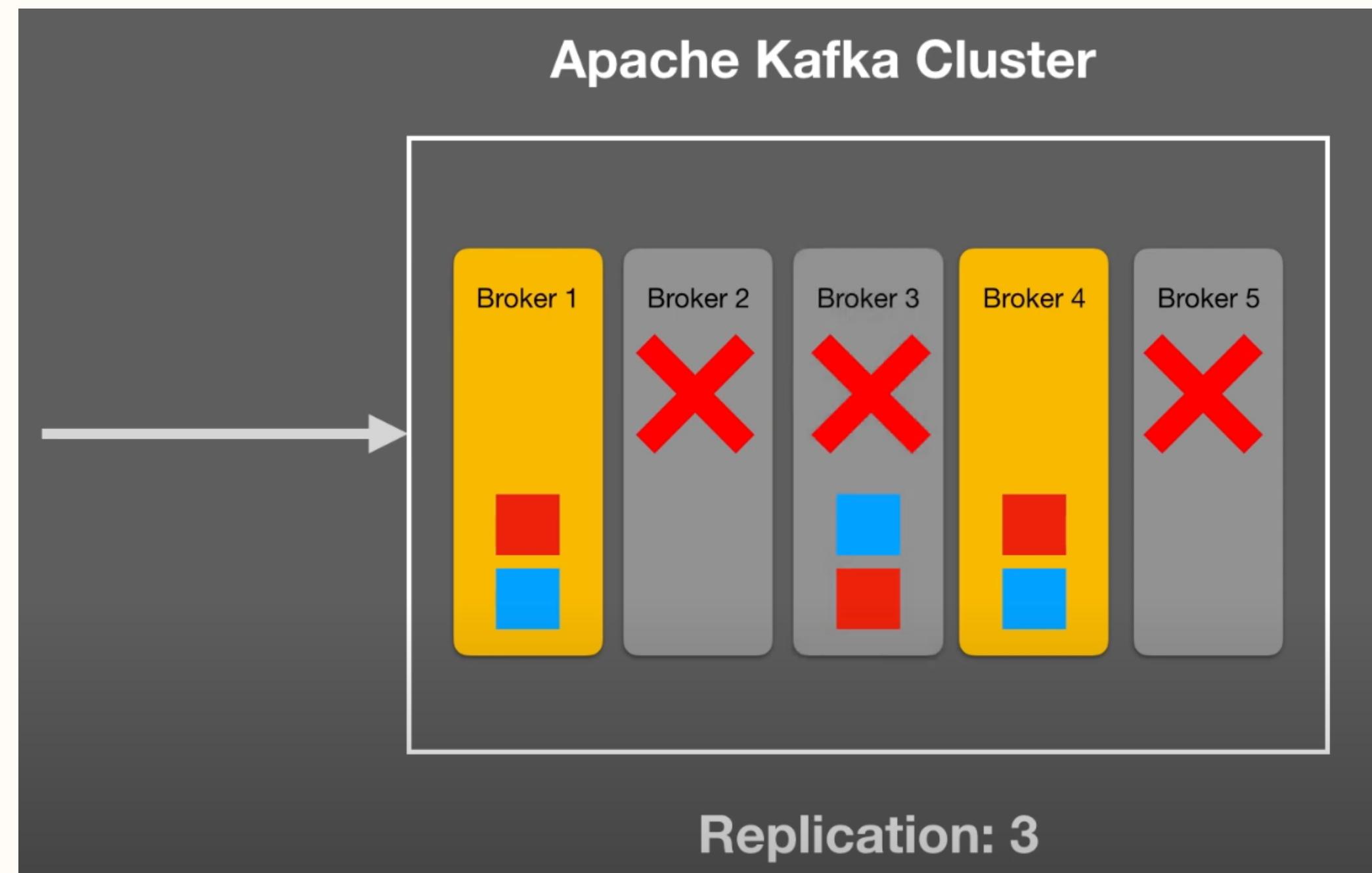
Replication Problem

Bir anda broker 2 de çöksün. Bu durumda replikasyon sayımızın 3 olması gerektiği için, kafka hemen broker 2'deki verileri alıp replikasyon sayısına uygun şekilde başka broker'lara replikasyon yapar.



Replication Problem

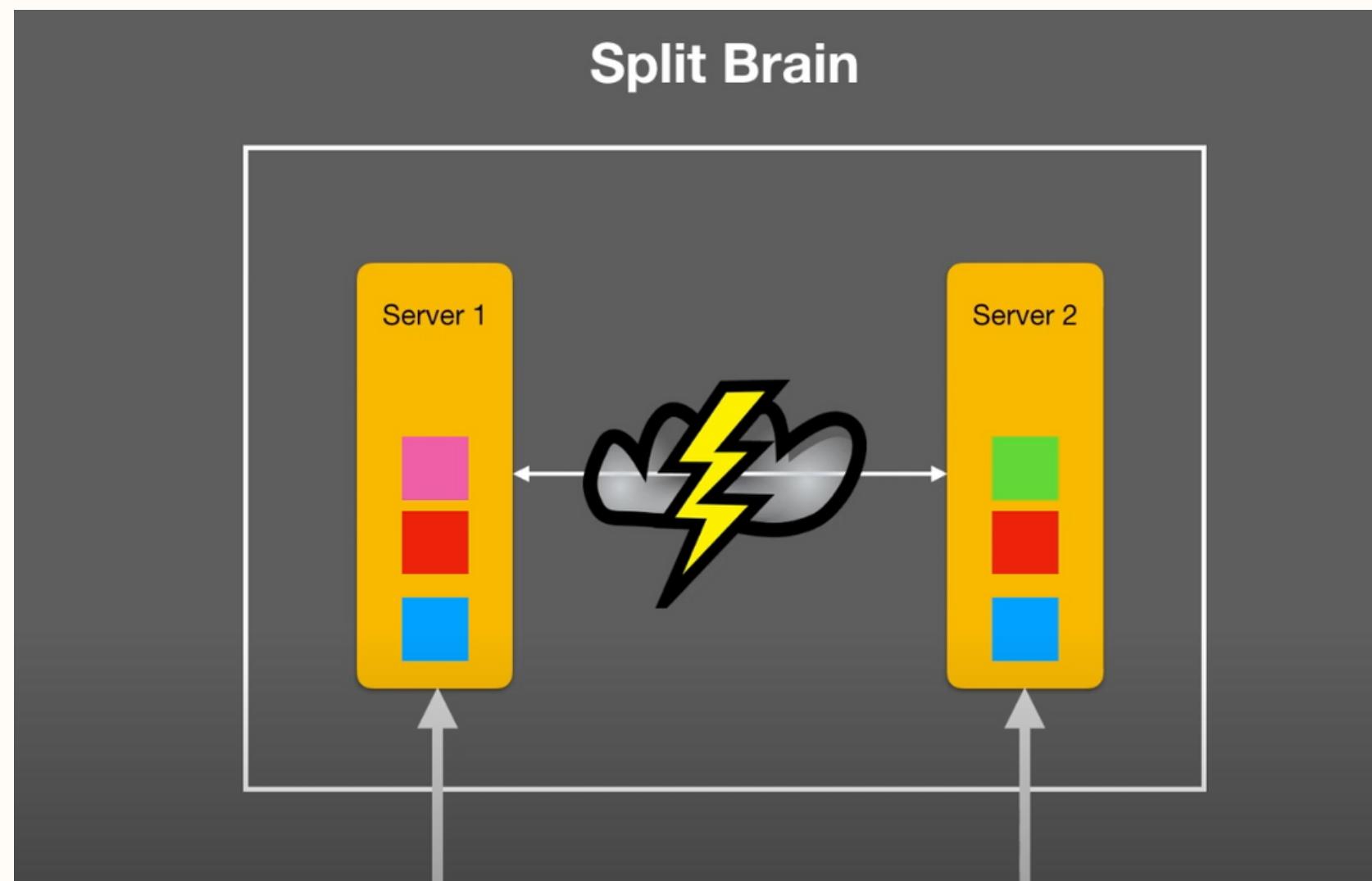
Şimdi'de broker 3 çöksün. Bu durumda kafka replikasyon sayısını 3'e tamamlayamayacağı için hata verecektir. Bu yüzden broker sayısı \geq replication sayısı olmalıdır.



Split Brain Problem

Farz edelim 2 brokerden oluşan bir cluster'ımız var. Replikasyon faktörü 1 olsun.

Bir nedenden dolayı aralarındaki bağlantı da bir sorun çıkıyor. O ana kadar senkron çalışan broker'lar, bu case oluştuğunda her broker diğer bir broker'ın devre dışı kaldığını düşünerek işlemleri yapmaya devam ediyor. Yani her iki broker da verileri yazmaya devam ediyor. Bu yüzden farklı veriler broker lara yazılmış oluyor.



Peki bu durumda hangi broker'daki veriler daha güncel, hangisini temel olarak almalıyız? İşte burada split brain sorunu ortaya çıkıyor. Bunu çözmek için tarih, saat, versiyon gibi farklı parametreler veri üzerinde saklanabilir. Ancak en önemli çözüm yöntemi Quorum'dır.

Split Brain Problem

Quorum, bir topluluğun geçerli olabilmesi için herhangi bir zamanda bulunması gereken asgari üye sayısıdır.

En az kaç futbolcu sahada olmalı ki futbol devam etsin. Burada quorum sayısı 7'dir. Beşinci kırmızı kartta oyun biter.

Birden fazla broker bulunan bir sistemde optimum quorum sayısı $n/2 + 1$ formülü ile bulunur.

3 broker bulunan bir sistemin formüle göre quoram sayısı 2'dir. Bu çökecek 1 brokerin tolere edilebileceği anlamına gelir.

Live ortamda en az 3 server ile çalışmak faydalıdır. 2 serverli bir sistemde quoram sayısı 2 olacağından çöken hiçbir sunucu tolere edilemez.

Kafka'ya Veri Nasıl Yazılır?

Apache Kafka verileri topic'ler de saklar, her topic kendine özgü bir isme sahiptir. Topicler brokerlar üzerinde saklanır.

Topicler partitionlardan oluşur. Verileri partitionlara yazarız. Her partition bir numaraya sahiptir.

Partition queue mantığı ile çalışır. Partitionın ortasına veya başına ekleme yapamayız

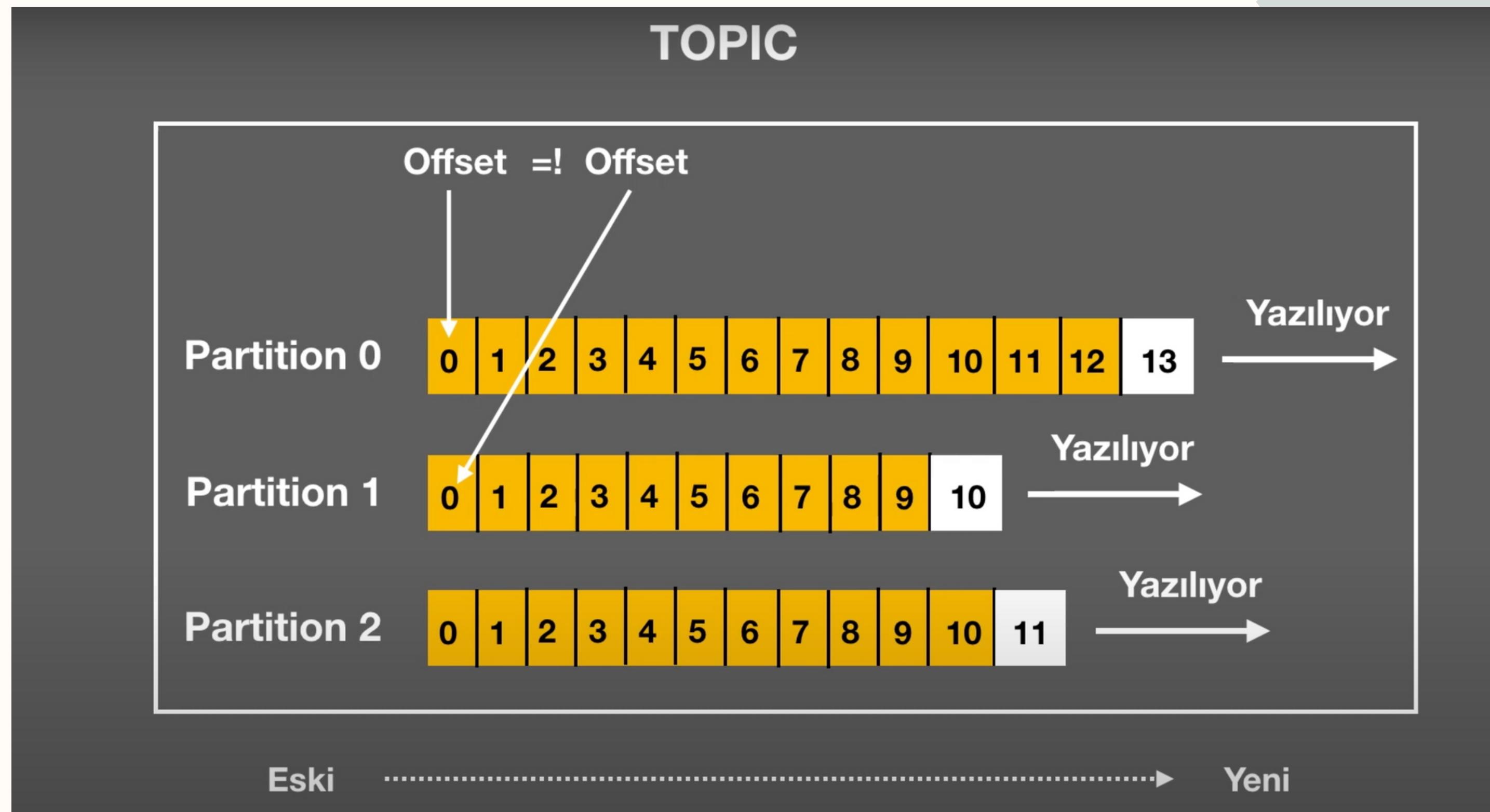
Yazılan bir veri bir daha değiştirilmez. Veriler harddiskte saklanır. Veriler sonsuza kadar saklanmaz.

Zaman bazlı olarak ayarlama yapabiliriz. Verinin partition içerisinde kalma süresi 7 günü geçtikten sonra veri partition üzerinden silinir.

Veri büyüklüğü bazlı ayarlama yapabiliriz. Örneğin bir topic içerisindeki veriler 100 GB'ı geçtiğinde eski verilerden başlayarak verileri siler.

Partition içerisindeki verilerin her birine bir offset tanımlanır. Verinin pozisyonunu belirler. Bu offset kullanılarak veri okunur.

Kafka'ya Veri Nasıl Yazılır?

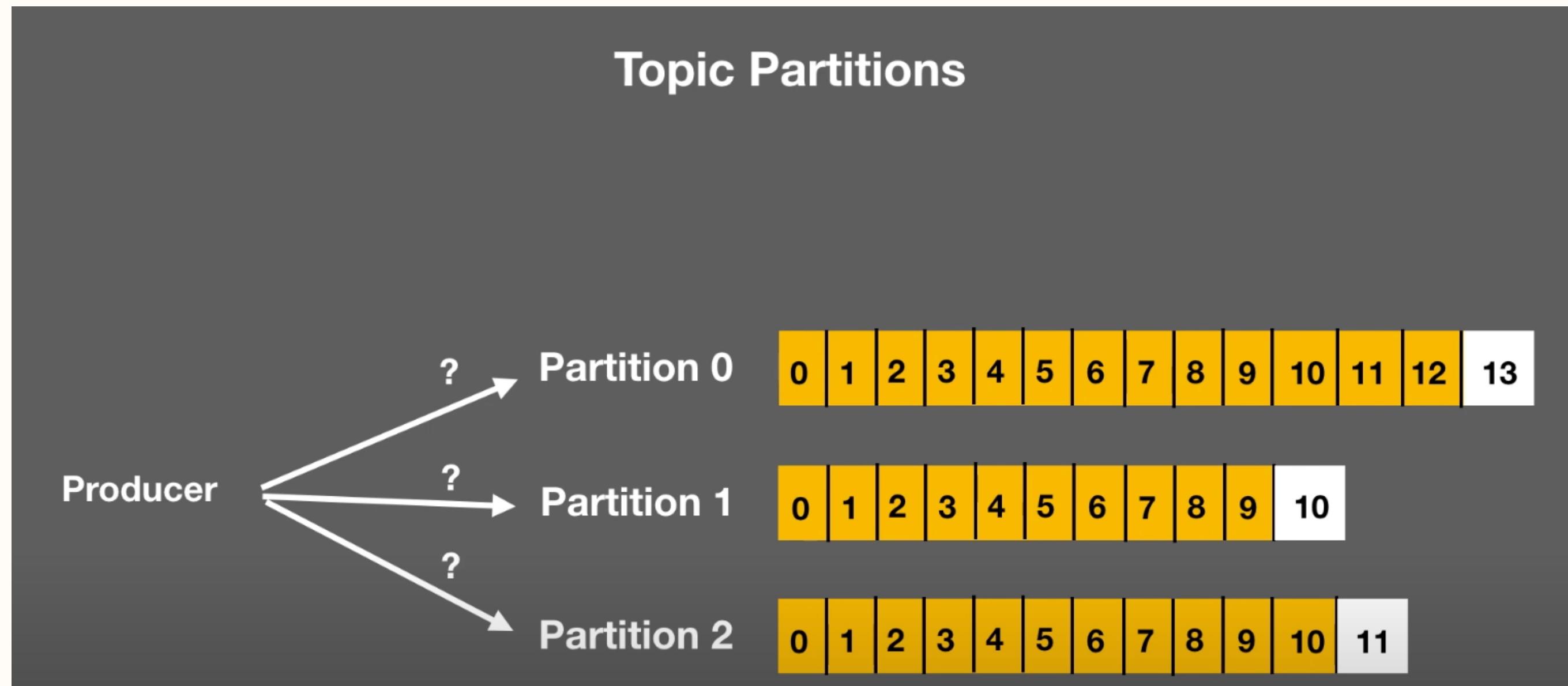


Neden Partition Oluştururuz?

- Yazılan verilerin belirli özelliklerine göre bir arada toplanması (aggregation)
- Verilerin sıralı bir şekilde toplanması (sorting - event sourcing)
- Daha hızlı okumak (parallelism)
- Verileri daha verimli saklamak (efficiency)

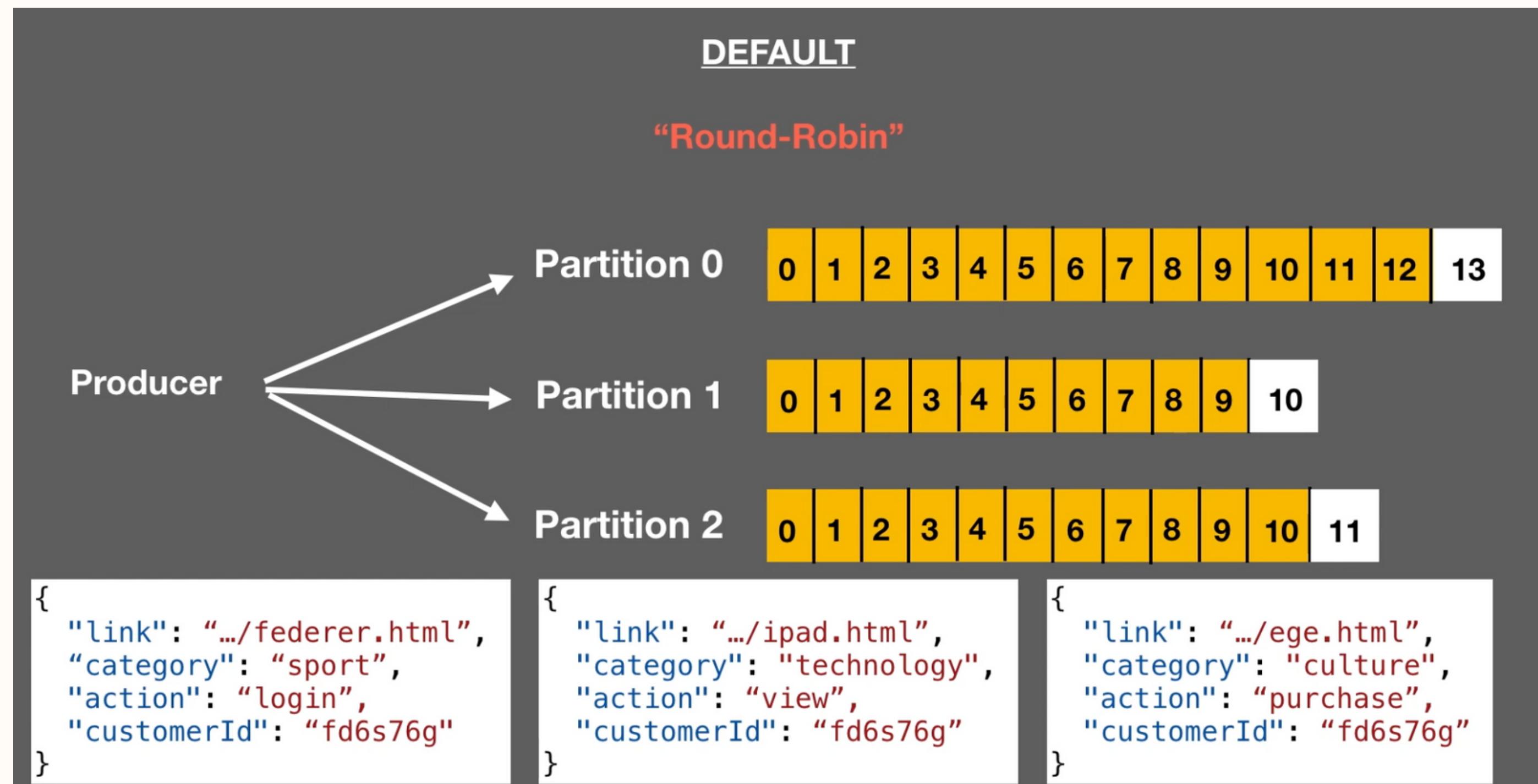
Veriler Partition'a Nasıl Yazılır?

1 veri 3 farklı partition'dan hangisine yazılır?



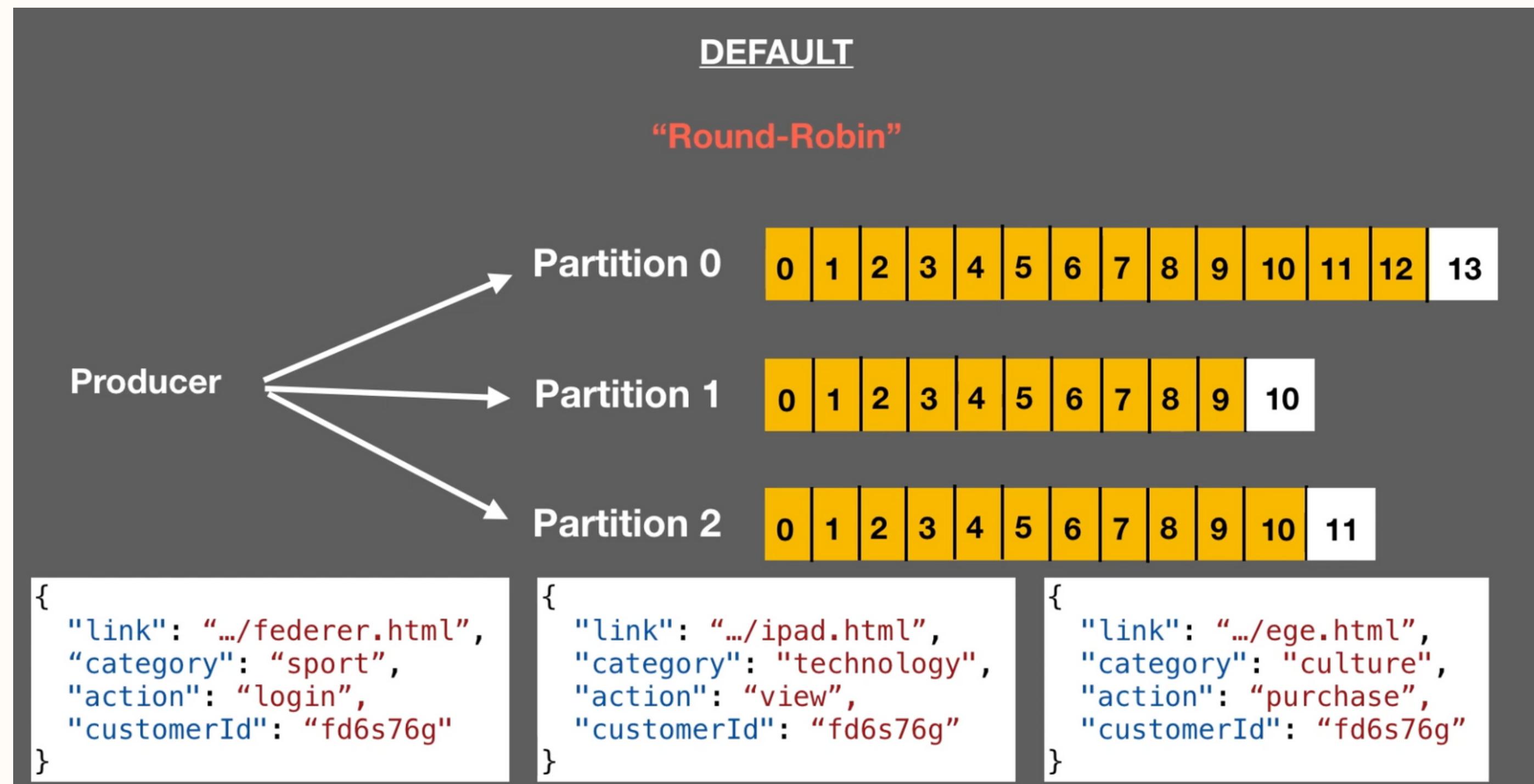
Veriler Partition'a Nasıl Yazılır?

Herhangi bir şey belirtilmediğinde kafka round-robin yöntemini kullanır. yani gelen verilerli, partition'lara bölüştürür.



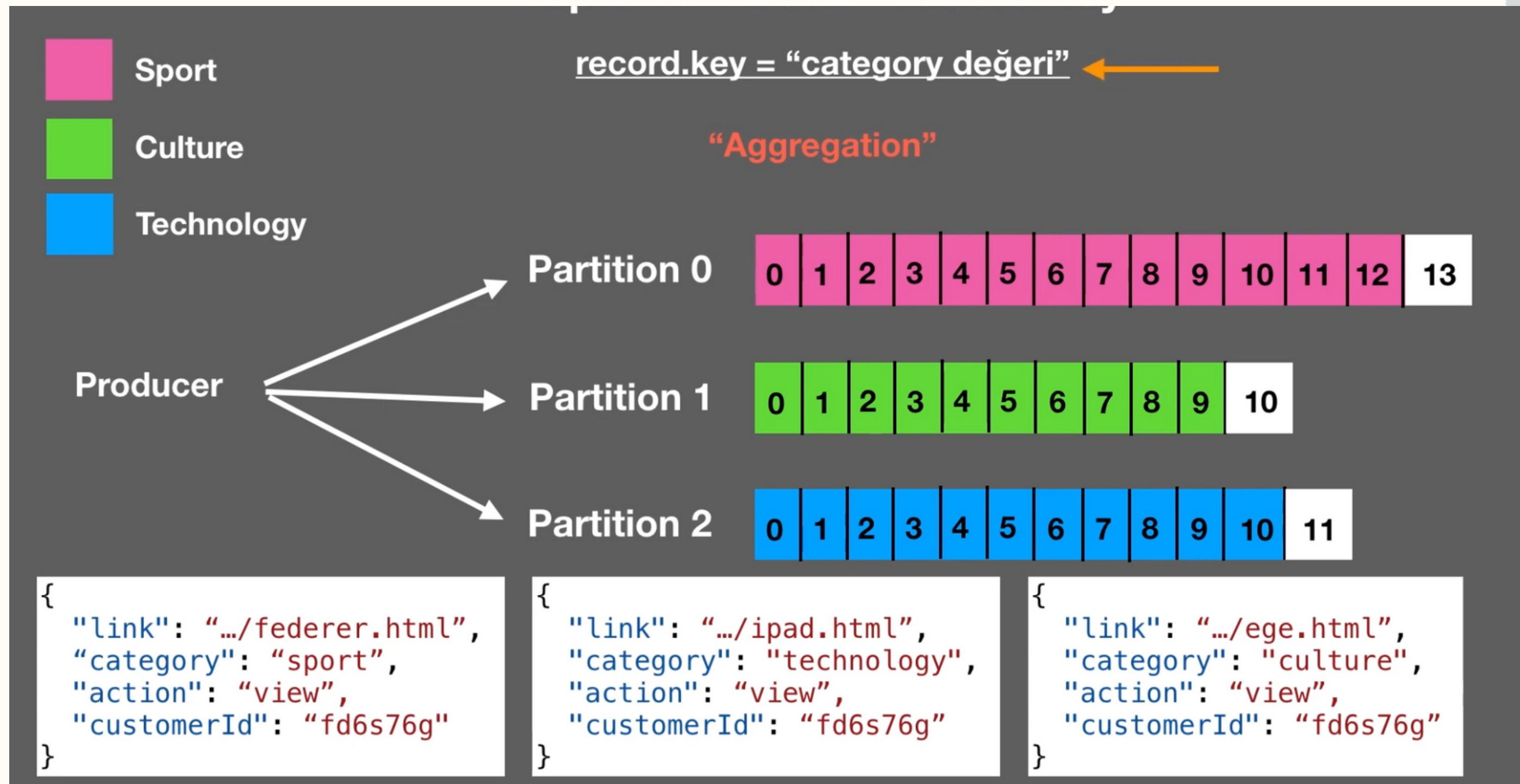
Veriler Partition'a Nasıl Yazılır?

Herhangi bir şey belirtilmediğinde kafka round-robin yöntemini kullanır. yani gelen verilerli, partition'lara bölüştürür.



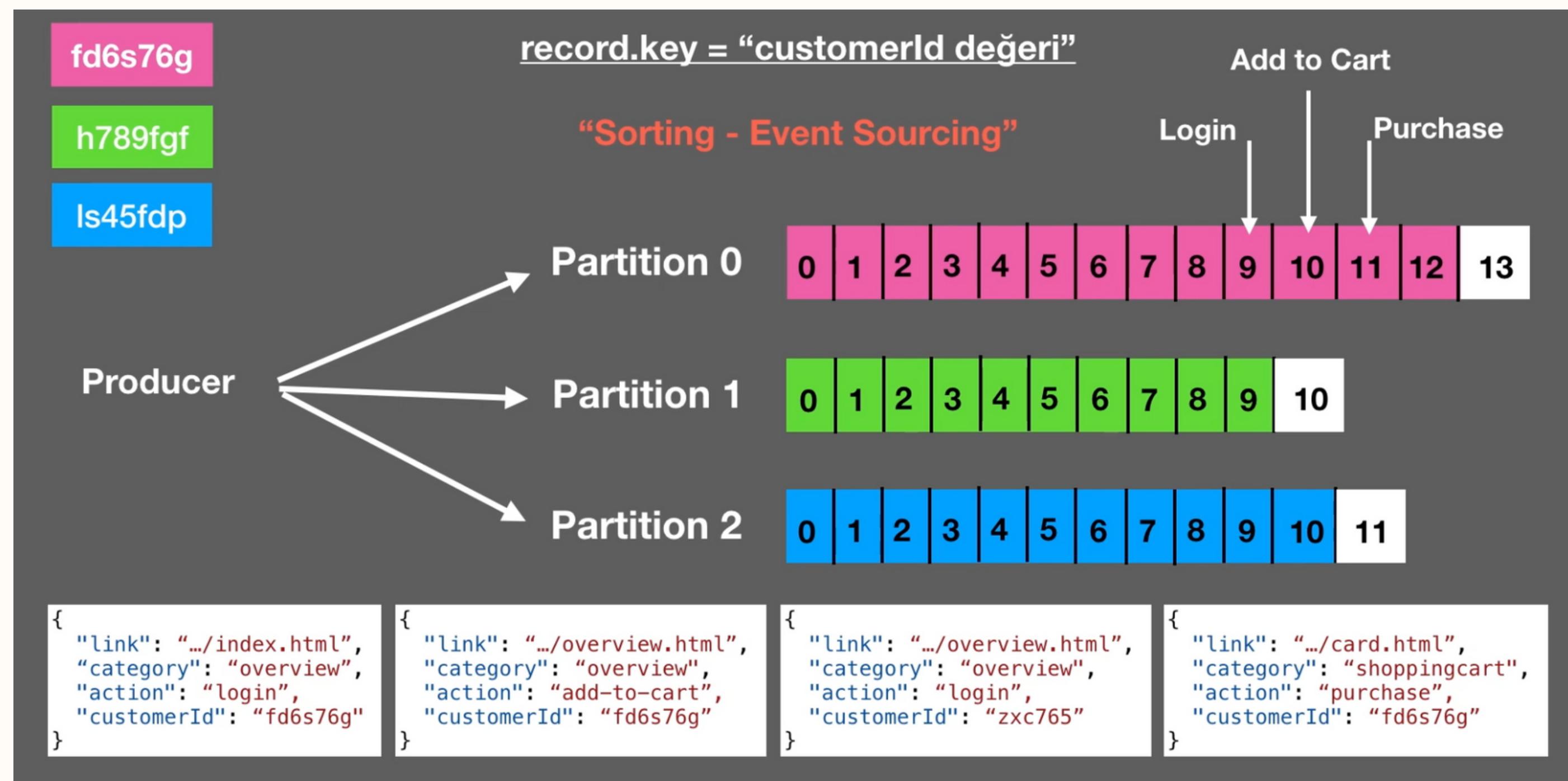
Veriler Partition'a Nasıl Yazılır?

Record.key olarak bir kategori belirler ise, bu sayede aynı kategoride olan mesajlar aynı partitionlara yazılır(Aggregation)



Veriler Partition'a Nasıl Yazılır?

Müşterinin müşteri id'sine göre partitionlara gönderilecek verileri kategorize edersek, müşterinin yaptığı her eventi sırasıyla saklamış oluruz. Buna da sorting-event sourcing denir. Record.key değiştirerek partitionları istediğimiz yöntem için kullanmış oluruz.



Veriler Partition'a Nasıl Yazılır?

Eğer sadece bir partition olsaydı okuma yazma performansı çok düşük olurdu.

Bunun nedeni bir partitiondan aynı anda aynı kimliğe sahip sadece bir consumerın okuma yapabilmesidir.

Leader ve Follower Partition

Replikasyon faktörü 3 olduğunda her partition replikasyon faktörü kadar çoğaltılar.
Replikasyon arttıkça partition sayısı da artmaktadır.

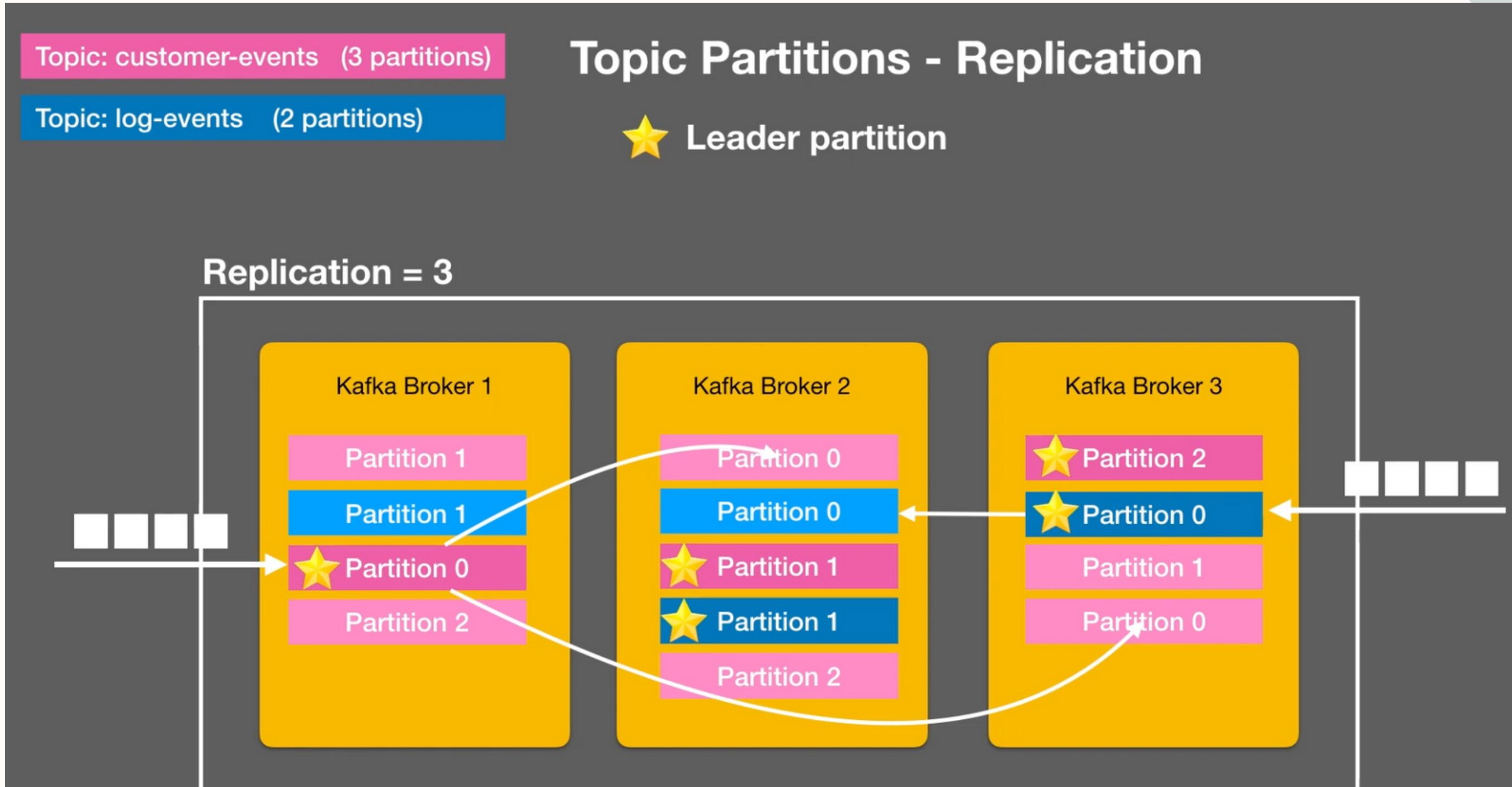
Peki verileri aynı veriyi tutan 3 partitiona birden mi yazacağız?

Burada leader ve follower partition kavramları karşımıza çıkar.

Veriler Leader partitiona yazılı ve follower partitionlara aynı veri leader partition tarafından gönderilir.

Eğer bir nedenden dolayı leader partition çöker ise, kafka hemen bir partitionı leader olarak seçer. Buna leader partition election denir

Leader ve Follower Partition



Kafka'ya Veri Yazımı

Kafkaya veri yazılımı 3 safhadan oluşmaktadır:

- Kafka'ya veriyi göndermek
- Verinin lider partitiona ulaşıp saklanması
- Lider partitionun kopya partitionlara veriyi iletip verinin onlarda da saklanması
ve böylelikle tam senkronlama durumuna ulaşmamız.

Producer Acknowledgment

Kafkaya veri gönderdiğimiz de bu üç safhanın hangilerinin bitmesini veya tastik edilmesini beklemek istedığınızı belirtebiliyoruz.

Burada da 3 seçenek var.

- En hızlı acks = 0 seçeneğidir ve ayrıca en risklidir. Kafkaya veri gönderildiğinde kafkaya verinin ulaşıp ulaşmadığını bilmeden diğer veri kafkaya yollanır. Bu durumda mesaj kaybolma riski yüksek olur.
- acks = 1, gönderilen veri kafkaya varır ve leader partitionda saklanır, senkronlama işlemi beklenmez ve diğer veri gönderilir. Orta derece hızlı ve güvenli, mesaj kaybolma şansı çok az.
- acks = all veya acks = -1, her gönderilen veri için 3 safhanın tamamlanması beklenir. En yavaş ve en güvenlidir. Mesaj kaybolma şansı yok.

Kafka'dan Veriler Nasıl Okunur?

Kafka'da veri okunan yapıya başta da belirtildiği gibi Consumer ismi verilir.

Consumer bir partitiona bağlanır ve okumaya başlanır.

Kafkanın okuma sırasında ilk baktığı verinin hangi pozisyondan - hangi offset'den okumaya başlanacağıdır.

Okuması yarı kalan bir consumerin zookeeper içerisinde son okuduğu offset belirlidir ve okuyucu oradan okumaya devam eder. Ancak ilk defa okumaya yapacak okuyucu şu şekilde çalışır:

Kafka'dan Veriler Nasıl Okunur?

- 1 - Event oku (offset 0)
- 2 - Event ile yapman gereken işlemini yap
- 3 - Diğer mesajı okuyabilmek için, kafka'ya okuduğumuz son mesaj ile işlemimizin bittiğini haber vermemiz gerekiyor. Buna commit deniyor. Commit et.
- 4- Kafka offset pozisyonunu bir sonraki pozisyon'a getiriyor ve offset bilgilerini zookeeper'a yazıyor.
- 5- Event oku (offset 1)
- 6- Commit et (offset 1)

İster 1 mesaj okunup commit edilir, ister 100 mesaj okunup commit edilir.

Kafka'dan Veriler Nasıl Okunur?

Kafkadan bilgi okumada da birden fazla seçenek vardır.

At Most once (en fazla bir kere)

- Okuyucu mesajı okur ve okur okumaz commit eder. Offset değeri 1 artar.
- Okumadan sonra veri üzerinde herhangi bir işlem yaparken bir sorun çıkarsa ve veri kaybolursa aynı veriyi tekrar okuma şansımız yoktur. Çünkü verinin işlemlerin tamamlanması beklenmeden commiti yapılır. Bir okuma daha gerçekleştirildiğinde bir sonraki veri okunur.

Kafka'dan Veriler Nasıl Okunur?

Kafkadan bilgi okumada da birden fazla seçenek vardır.

At Least Once (en az bir kere)

- Okuyucu veriyi okur ve belirli işlemlerden geçirerek veri üzerinde yaptığı işlemleri bitirir. Yapacağı işlemler bitmeden verinin okunduğunu zookeeper'a bildirmez. Offset değeri arttırmaz. Bu yüzden işlemler sırasında bir hata olussa da, veri kafkadan tekrar okunabilir.

Kafka'dan Veriler Nasıl Okunur?

Kafkadan bilgi okumada da birden fazla seçenek vardır.

Exactly Once (tam bir kere)

- Bu model sadece Kafka - Kafka arasındaki iş akışlarında Kafka Streams API ile sağlanıyor. Mesaj, Kafka topic'ler arasında transfer olurken ve işlenirken transactional producer ve consumer kullanılıyor. Bu sayede okunan mesaj tekrar okunmuyor ve transaction içerisinde bulunduğu için kaybolmuyor.

Consumer Groups

Consumerlar her zaman belirli bir consumer group içerisinde bağlıdır ve bu consumer group belirli bir isme sahiptir.

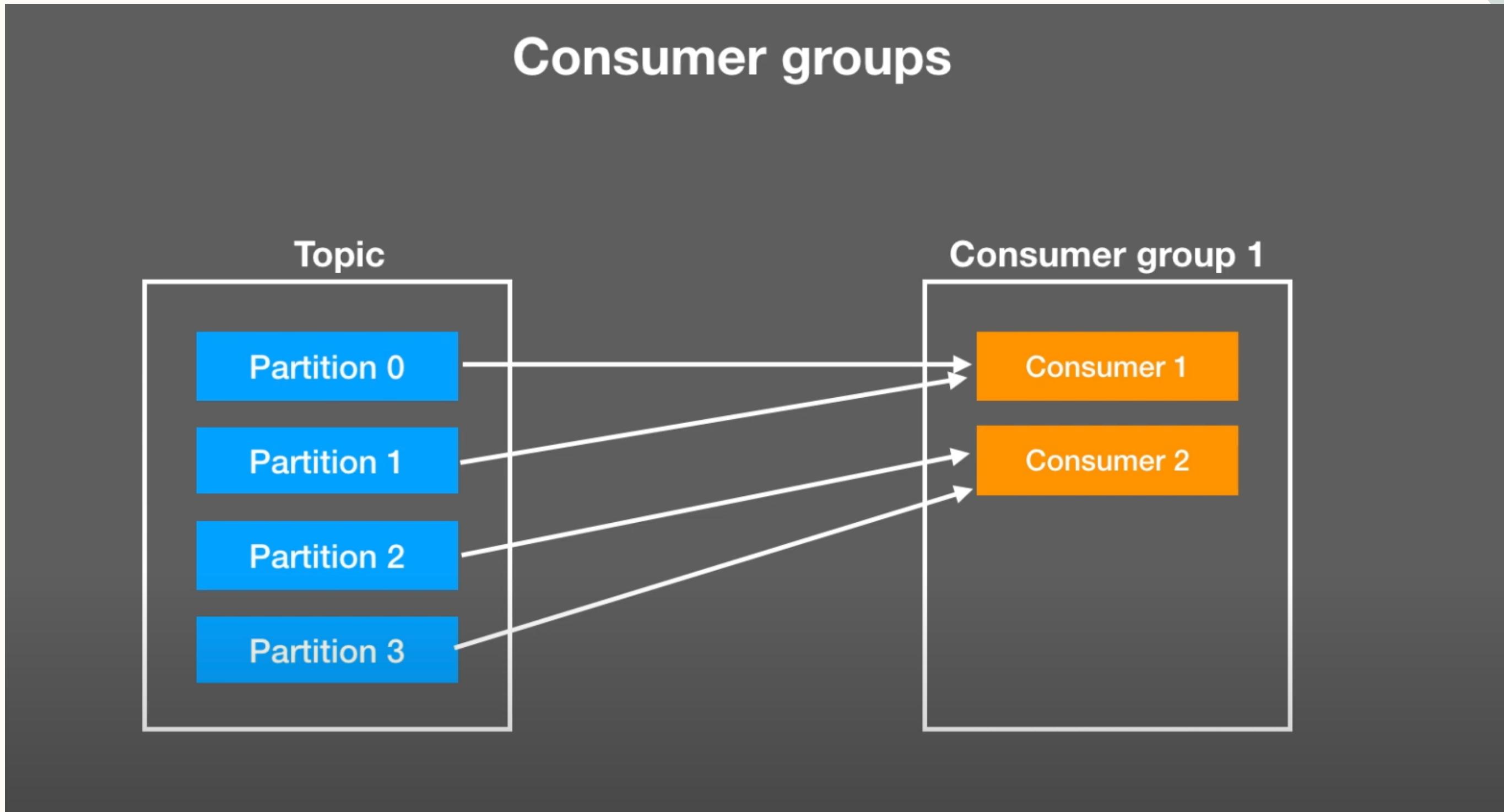
Consumer group içerisinde bulunan her bir consumer bu ismi kendisine alır.

Bu yüzden bir partitiondan aynı isme sahip sadece bir consumer okuma yapabilir.

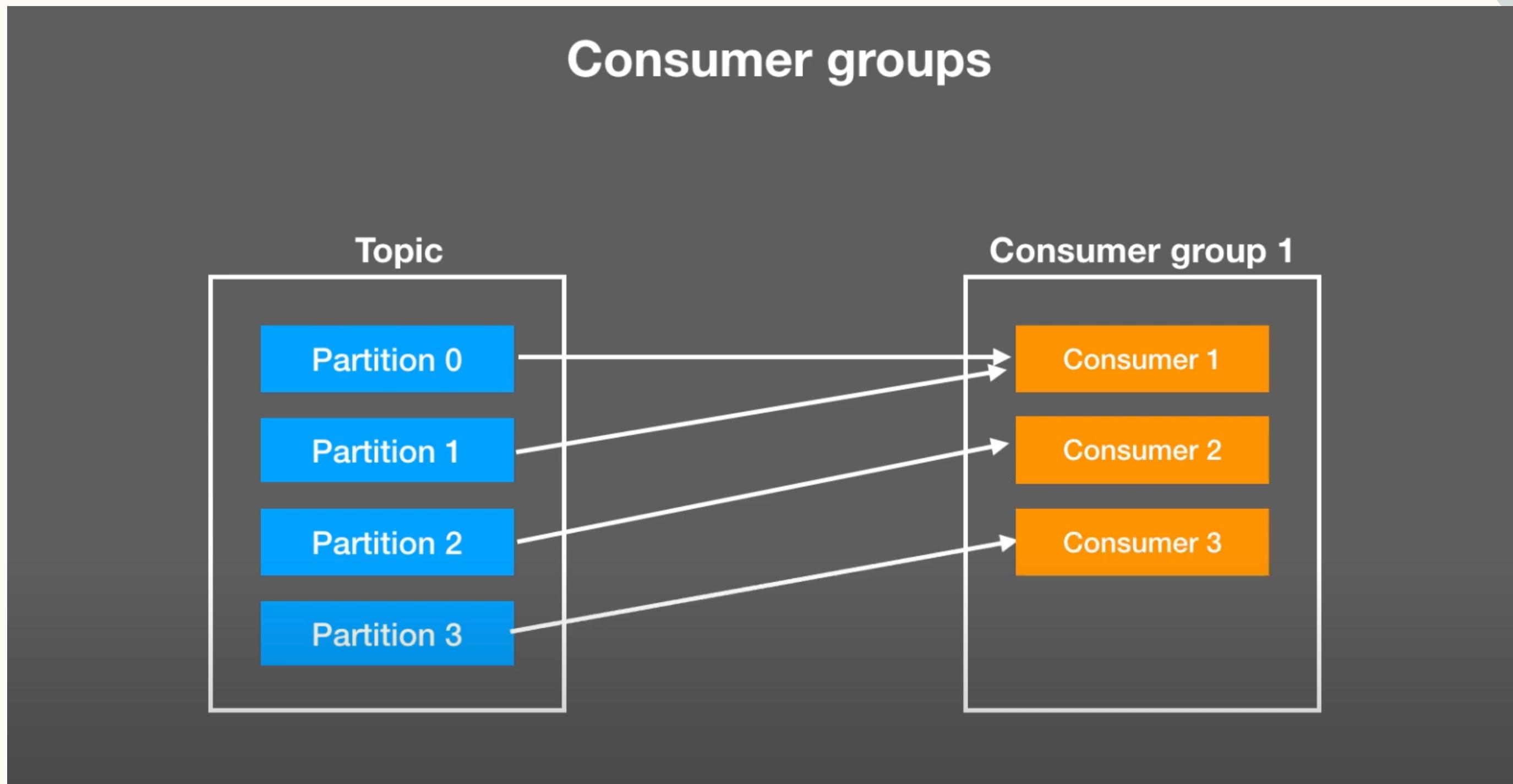
Tek consumer aynı anda birden fazla partitiondan okuma yapabilir. Ancak çok performanslı olmaz.

Kafka okuma yaparken Consumer group içerisinde bulunan consumerlara paylaştırma işlemi yapar.

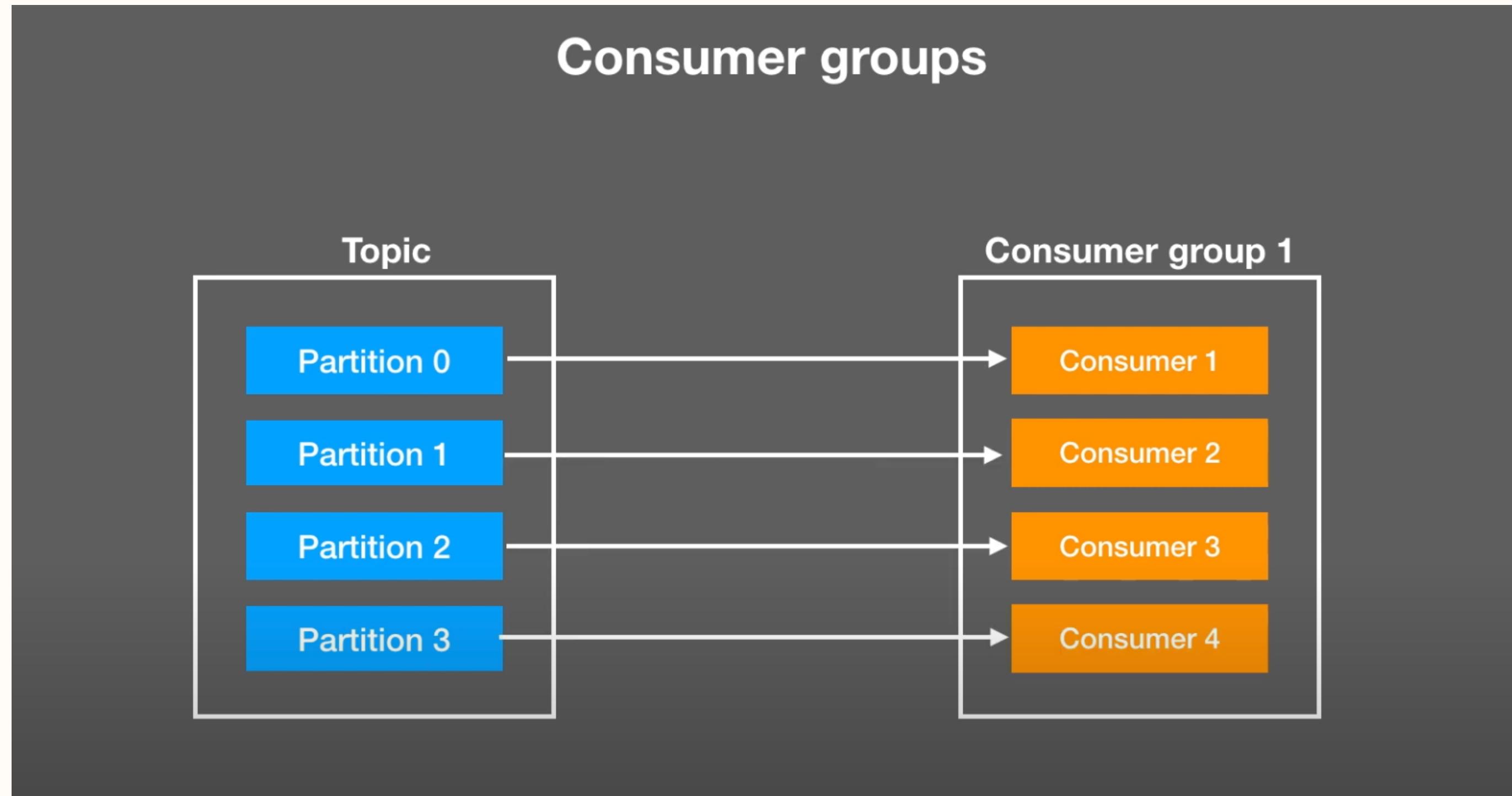
Consumer Groups



Consumer Groups

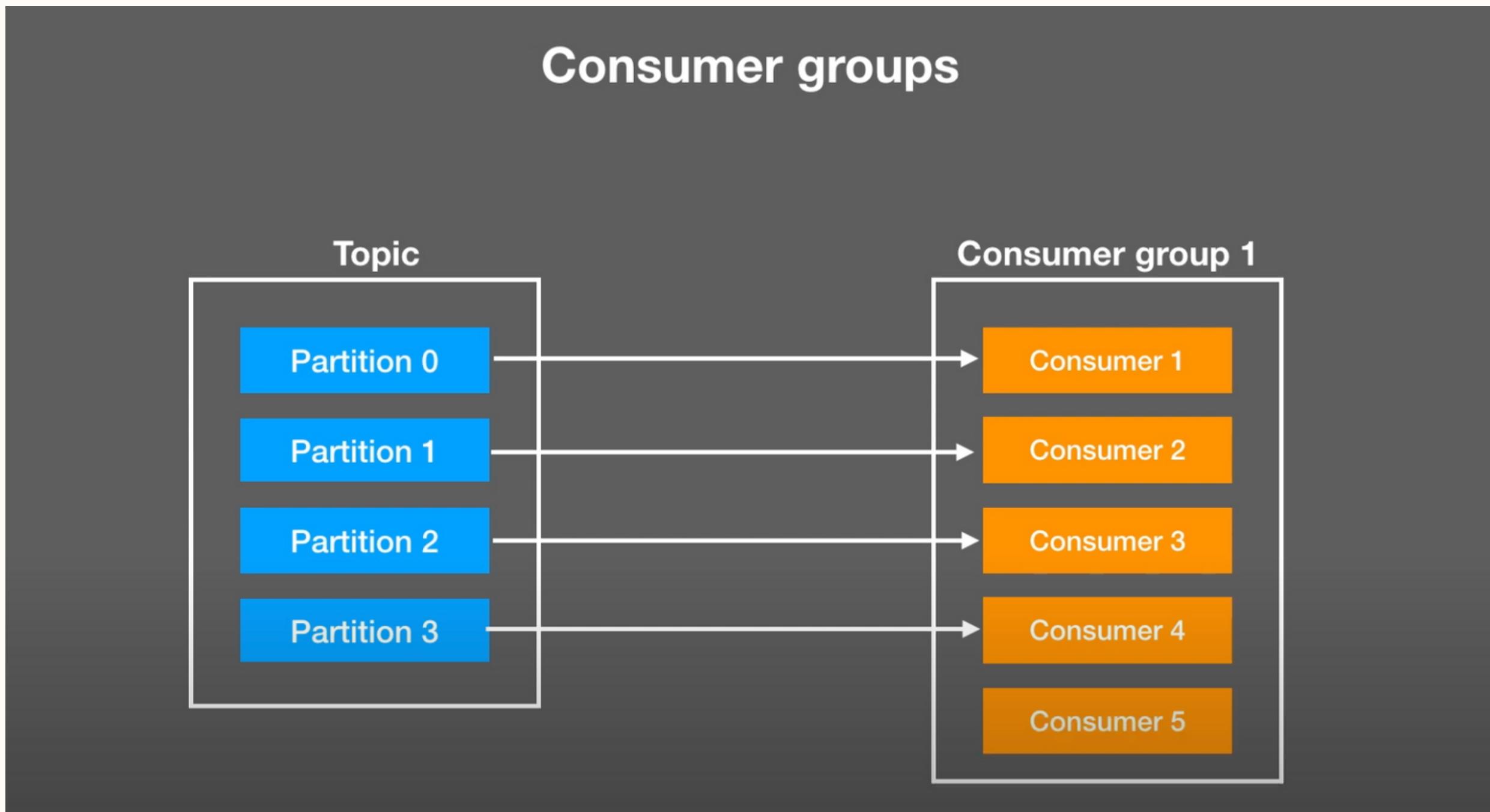


Consumer Groups



Queue Model

Eklenen son consumer okuma yapmaz, yapamaz. Consumer 5 pasif olarak bekler. Herhangi bir consumer'da problem meydana geldiğinde devreye girer ve problemli consumer yerine okuma yapar.



Pub/Sub Model

Burada iki farklı consumer group içerisinde bulunan consumerlar aynı partition üzerinden okuma gerçekleştirebilir. Çünkü iki farklı consumer groupda bulunan consumerlar farklı isimlere sahiptir.

