Apache Kafka is an open source distributed streaming platform designed for :

- building real-time
- data pipelines
- streaming applications

Kafka introduction

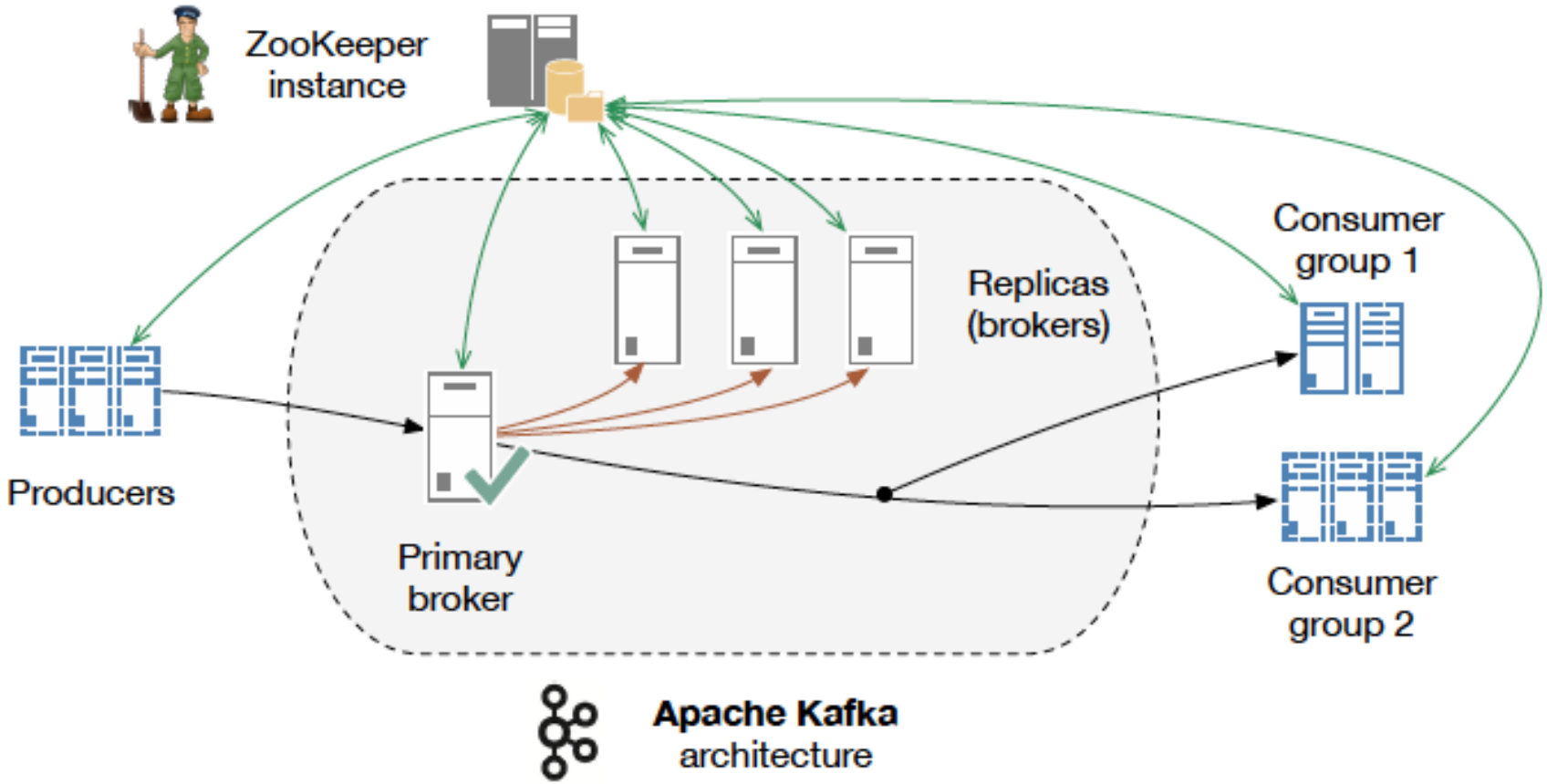
Apache kafka is a **distributed publish-subscribe messaging system**

It operates on a pub-sub model, where producers publish messages and consumers subscribe to topics to receive these messages.

Multiple consumers from a Consumer Group subscribe to topics allowing for load-balancing and parallel data consumption.

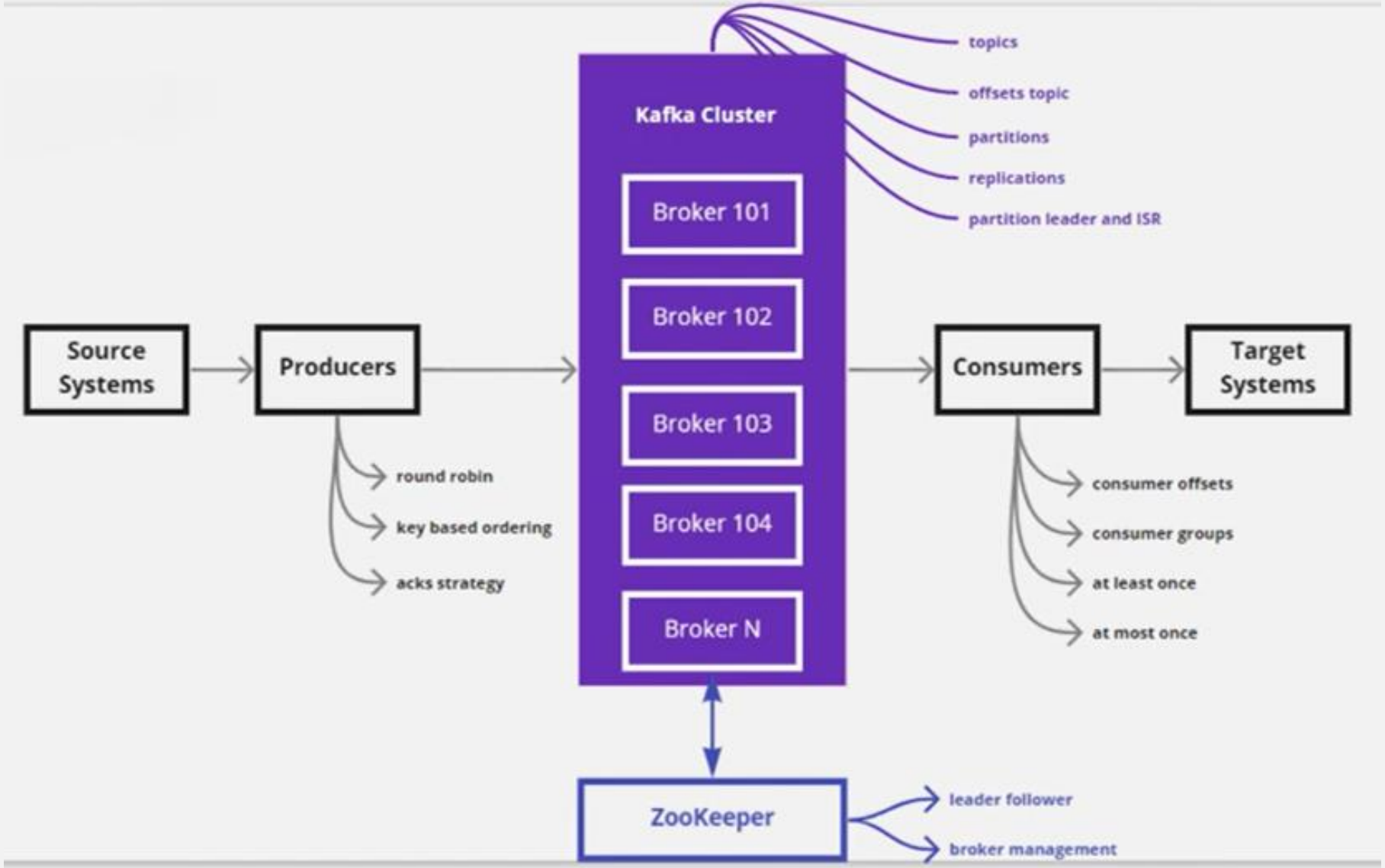
Kafka introduction

Apache Kafka که اولین بار برای linkedin پیاده‌سازی و در سال ۲۰۱۱ به صورت open source ارائه شد، امروزه بزرگترین data pipeline ها در جهان از کافکا استفاده می کنند. شکل زیر معماری این نرم‌افزار را نشان می‌دهد:

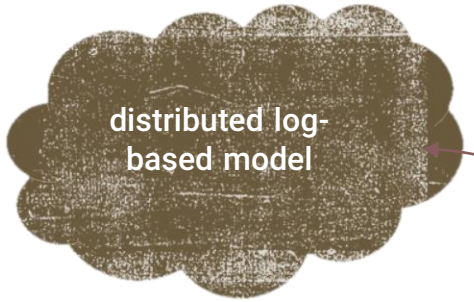


این نرم‌افزار که با زبان جاوا و اسکالا نوشته شده است دارای تکنولوژی فوق‌العاده‌ای است

Concepts in Kafka



Kafka Features



Fast

scalable

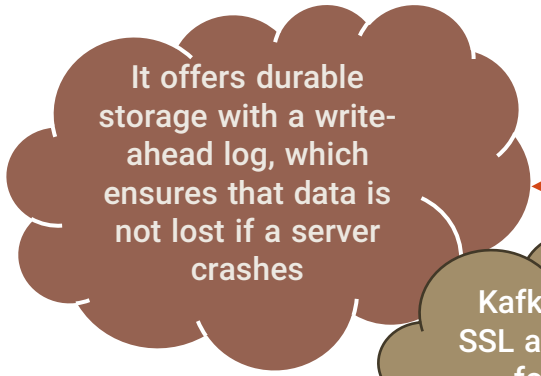
durable

fault-tolerant

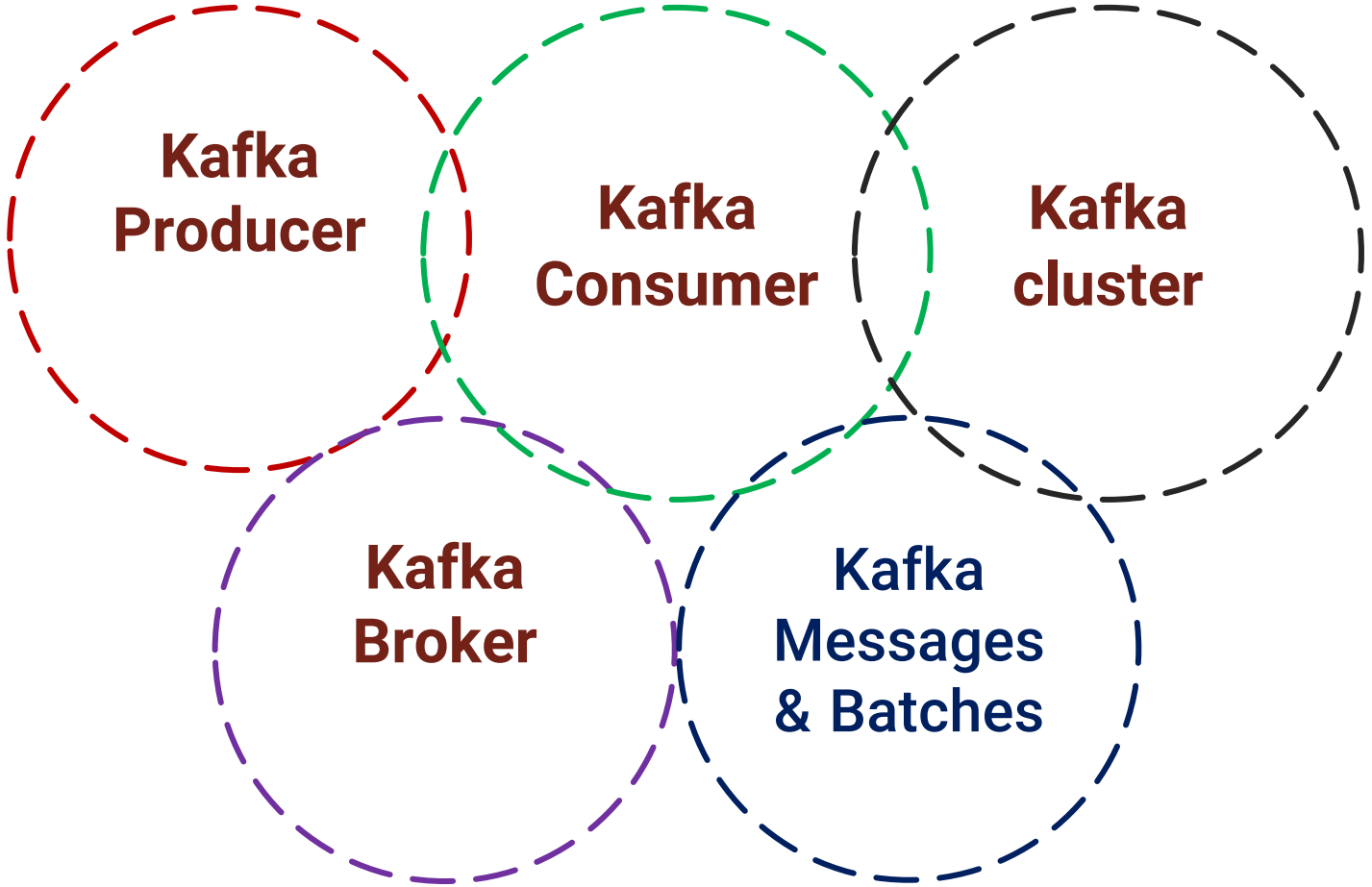
supports automatic recovery

Security

Dumb broker / smart consumer model



عملکرد کلی Kafka شامل مفاهیمی از قبیل:



Kafka Messages & Batches

کافکا از یک پروتکل binary TCP-based استفاده می کند که برای کارایی بهینه شده است و متکی بر یک انتزاع "message set" متکی است که به طور طبیعی پیام ها را گروه بندی می کند تا سربار رفت و برگشت شبکه را کاهش دهد.

واحد اصلی داده در کافکا یک پیام(Message) است

A **Message** is like a record in a DB table

به صورت آرایه ای از byte ها ارسال می شود و پیام ها را می توان فشرده (compress) کرد.
کافکا از فشرده سازی پیام برای کاهش ترافیک شبکه پشتیبانی می کند.

رویکرد دسته ای = افزایش توان عملیاتی

Messages are grouped into batches

لایه ذخیره سازی کافکا با استفاده از partitioned transaction log پیاده سازی شده

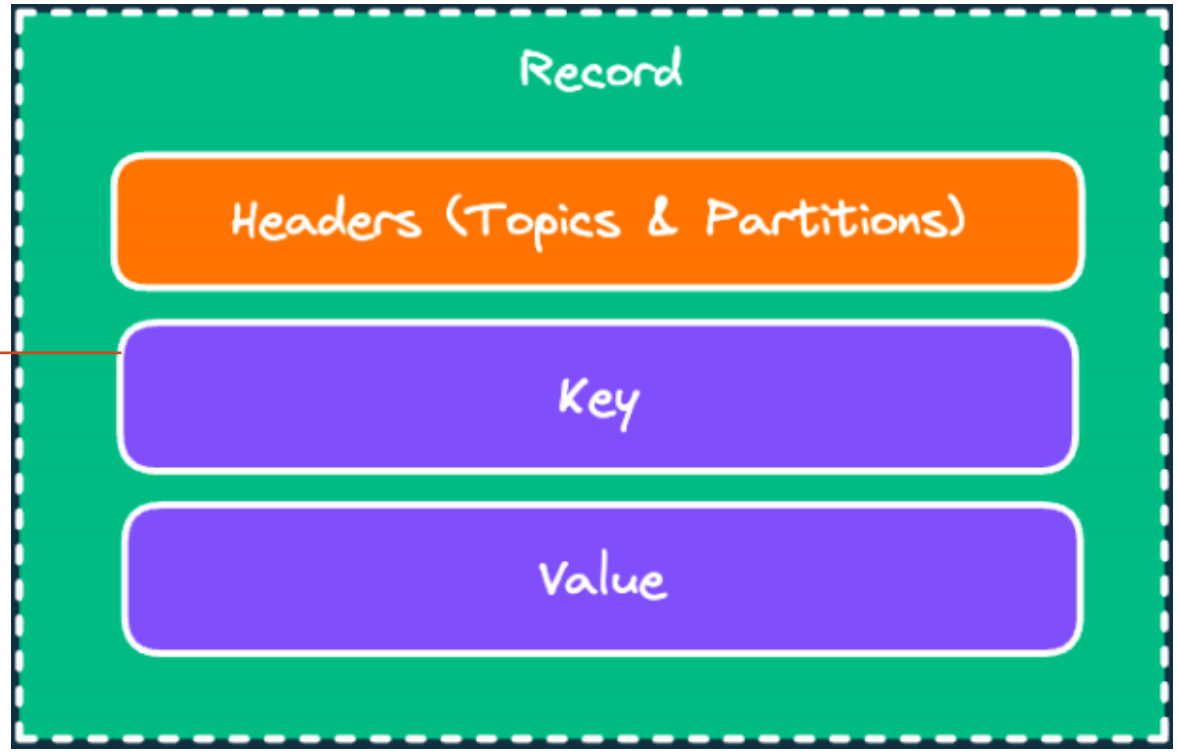
Once a batch is ready, it's sent to a **Topic** & **Partition**

Kafka Messages & Batches

کافکا جریان داده ها و رکورد ها را در ساختارهایی به نام topic ذخیره می کند.

هر رکوردی (Message) دارای یک کلید، یک مقدار و یک برچسب زمانی می باشد تا بصورت مجزا از سایر رکورد ها مشخص باشد.

Type key: int, string , ...



KAFKA TOPICS & PARTITIONS

Topics مانند یک جدول DB یا یک پوشه هستند

هر پیامی(Message) به Topic خاصی می رود

Topics نیز از چند پارتیشن (partition) تشکیل شده اند

هر پارتیشن از یک سری پیام‌های متوالی مرتب سازی شده، غیر قابل تغییر (زمانی که دیتا رو داخل آن می نویسیم غیر قابل تغییر) ، تشکیل یافته که هر پیامی بصورت پیوسته بهش الحاق میشه.

پارتیشن ها(partitions) به معنای افزونگی (redundancy) هستند و Topic را به صورت افقی (horizontally) مقیاس پذیر می کنند

پیام ها توسط producer(تولید کننده پیام) به یک پارتیشن (partition) append می شوند ، یعنی به انتهای لیست می رود.

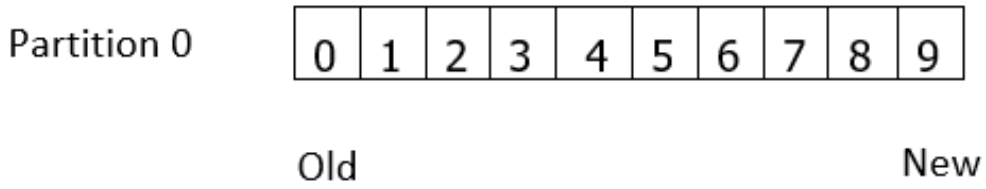
نکته ای که باید بهش توجه کرد اینه که Kafka بدون در نظر گرفتن اینکه آیا پیام‌ها consume شدن یا نه اونارو تا یک مدت زمانی (به صورت پیش فرض یک هفته) نگه میداره.

کافکا بصورت پیش فرض به منظور توزیع پیامها بصورت یکنواخت بین پارتیشن ها از round-robin partitioner استفاده میکنه.

KAFKA TOPICS & PARTITIONS

گفتیم که هر Topic از یک یا چندین Partition برای ذخیره داده‌ها استفاده می‌کند. تعریف درست تعداد Partitionها در یک Topic، تاثیر مستقیمی بر **درجه همزمانی و کارآیی** در آن Topic و کل سیستم دارد. در Kafka تمامی پیام‌ها به همان ترتیبی که وارد شده‌اند، در Partitionهای یک Topicذخیره می‌شوند و به همان **ترتیب** نیز توسط Consumer ها **pull** می‌شوند. **یعنی کافکا ترتیب پیام را در یک پارتیشن تضمین می کند، اما نه در بین پارتیشن های یک موضوع.**

بطور مثال فرض کنید تعداد Partitionهای Topic ، یک می‌باشد در این صورت تمامی پیامهای دریافتی تنها در یک Partition ذخیره می‌شوند.



هر خانه از یک Partition، توسط یک شناسه از نوع int و با نام offset در دسترس است. تمامی پیام‌های جدید ارسالی توسط Producer با offsetای بزرگتر از offsetموجود در این Partitionذخیره می‌شوند؛ یعنی در انتهای آن قرار می‌گیرند. در مثال فوق در صورت دریافت پیام جدید، offsetآن با عدد 10 مقداردهی می‌شود. همچنین عملیات خواندن نیز از کوچکترین offsetای که هنوز مقدار آن توسط **Consumerها** خوانده نشده است، انجام می‌شود. همانطور که مشخص است، بدلیل اینکه تعداد Partitionهای این مثال عدد یک می‌باشد، تمامی درخواست‌های Producerها در یک Partition قرار می‌گیرند و تمامی Consumerها نیز از طریق یک Partition به پیام‌ها دسترسی دارند؛ یعنی در صورت بالا بردن تعداد Producerها یا Consumerها، کارآیی بالا نمی‌رود.

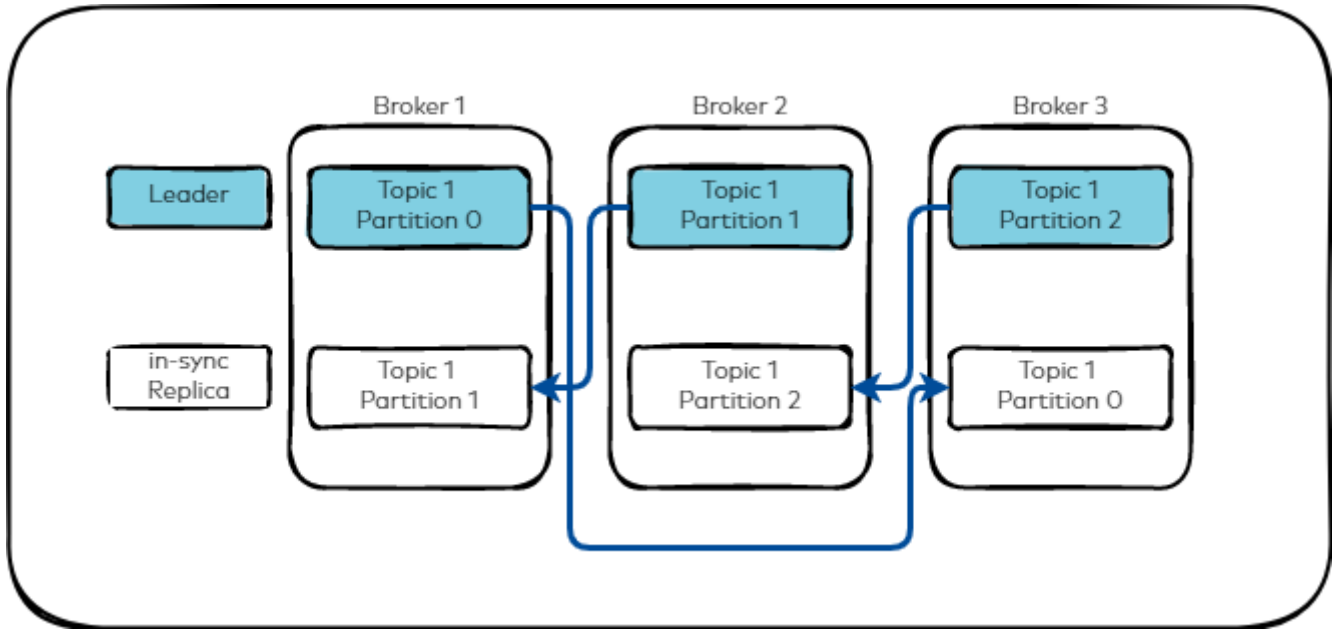
البته با اینکه کنترل مقدار اولیه offsetبرای شروع یک Consumer به دست خود Consumerو Zookeeperاست، اما در اکثر موارد تمامی Consumerهای یک Topic باید از یک نقطه، شروع به خواندن داده‌ها کنند. **در این حالت تا زمانیکه پیام با offset 1، توسط Consumerای خوانده نشود، هیچ Consumerای نمی‌تواند پیام شماره 2 را بخواند.**

KAFKA TOPICS & PARTITIONS

کافکا built-in durability را از طریق log replication ارائه می‌کند.

پارتیشن‌های یک موضوع در میان broker های Kafka cluster توزیع می‌شوند و هر broker یک یا چند پارتیشن را ذخیره می‌کند.کافکا به شما این امکان را می‌دهد که ضریب replication را برای یک Topic پیکربندی کنید تا کنترل کنید چند سرور دارای کپی از داده‌ها برای fault tolerance هستند. به عنوان مثال، اگر شما سه broker و ضریب تکرار 2 داشته باشید، حداکثر دو تا از آن broker می توانند بدون از دست دادن داده ها شکست بخورند.

ضریب replication باید حداقل دو باشد تا در صورت خرابی سرور، fault tolerance فراهم شود.

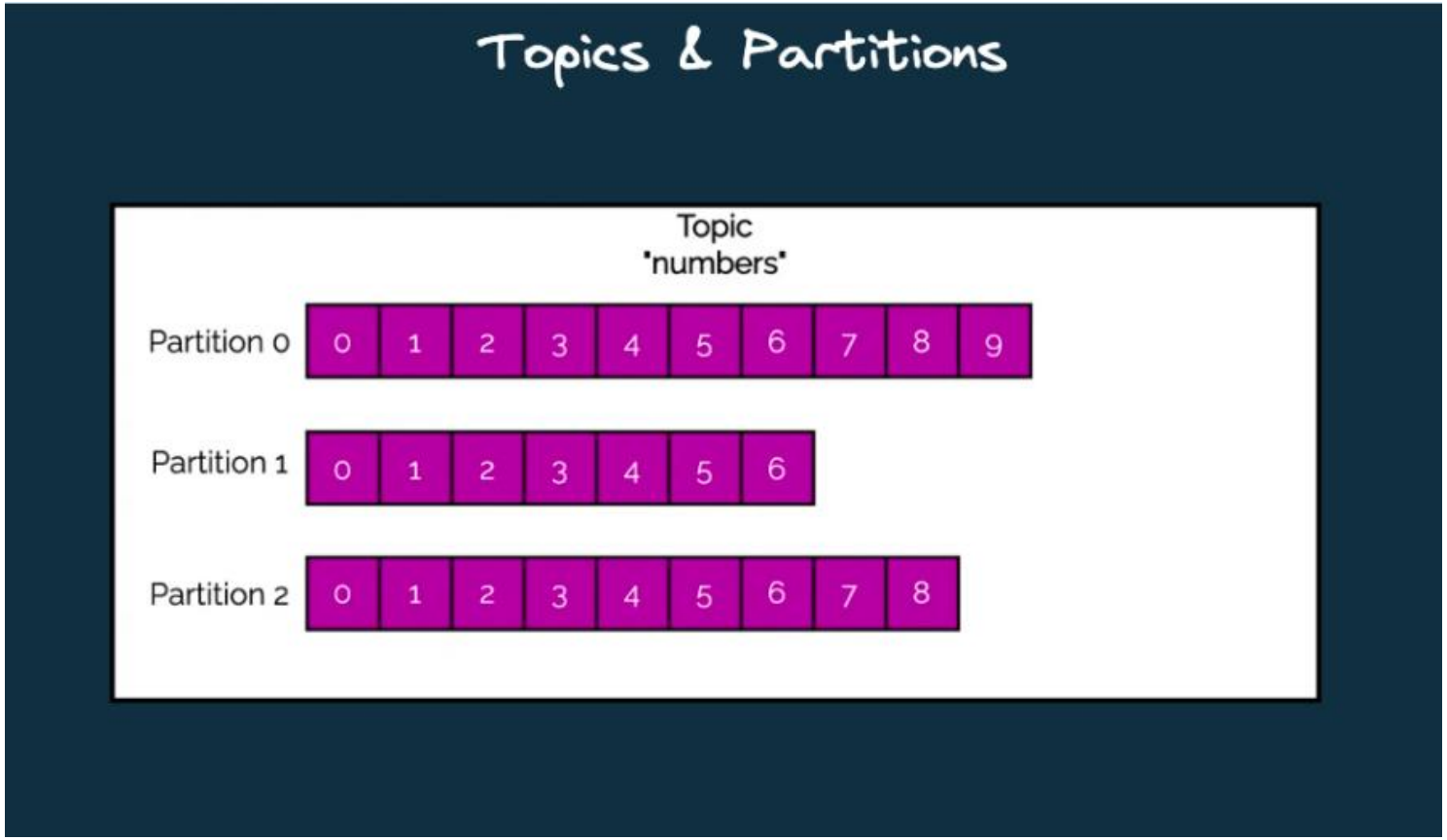


علاوه بر آن باید بدانید که هر Partitionدر هر زمان تنها توسط یک (leader)Primary Broker می‌تواند در دسترس سایر قسمت‌ها قرار بگیرد و تمامی عملیات خواندن و نوشتن در Partition توسط Kafka Server انجام می‌شود وسایر broker ها به صورت in-sync دیتا ها را خواهند داشت.

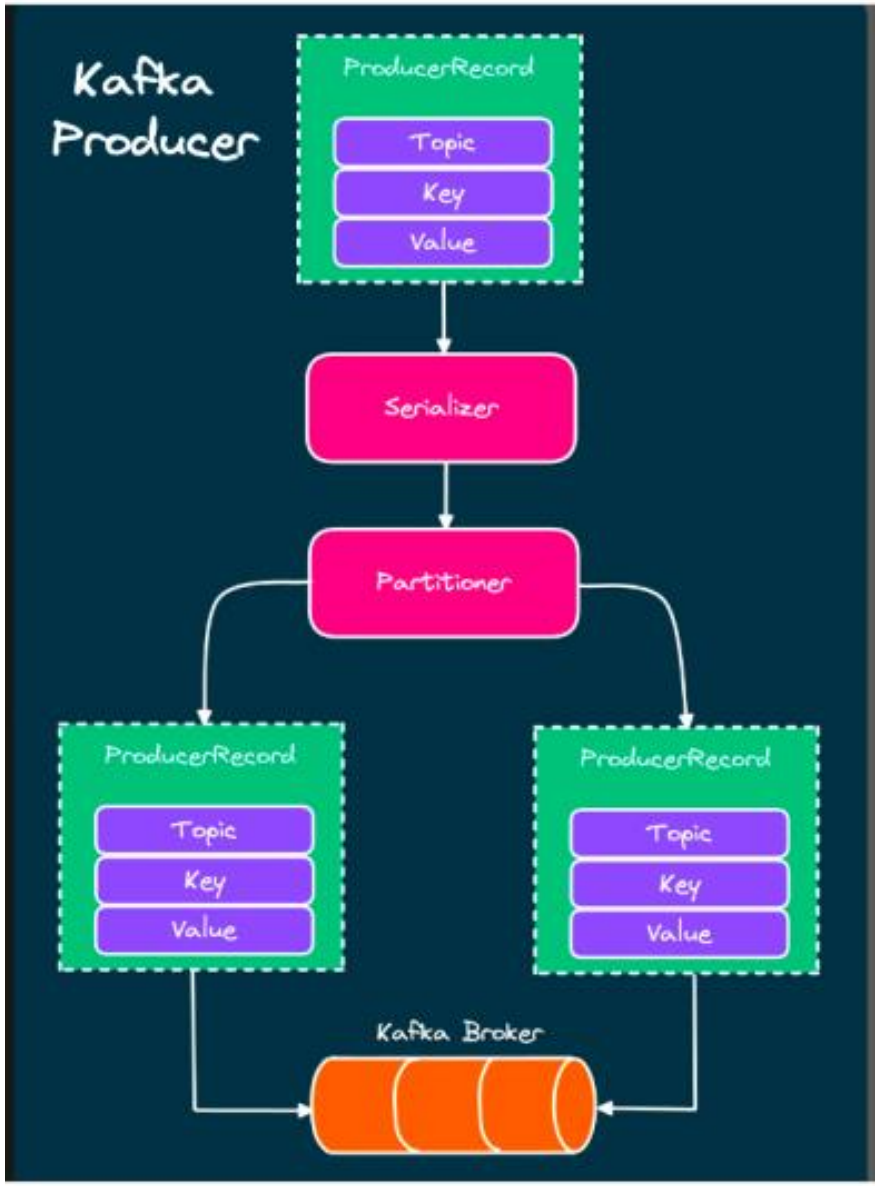
و در صورتیکه به هر دلیلی سرور leader از دسترس خارج شود، مدیریت این Partition به سرورهای دیگر (replica broker) داده می‌شود.

KAFKA TOPICS & PARTITIONS

باید در تعریف تعداد Partition های یک Topic این نکته را در نظر بگیرید که این تعداد کاملا به نیازمندی شما و کارآیی که شما مد نظر دارید، بستگی دارد.
مسیریابی پیام کافکا نسبتاً اساسی است و بر پارتیشن بندی مبتنی بر topic متکی است.



Producer



Producer می تواند یک اپلیکیشن ، سیستم یا زیرسیستم باشد.

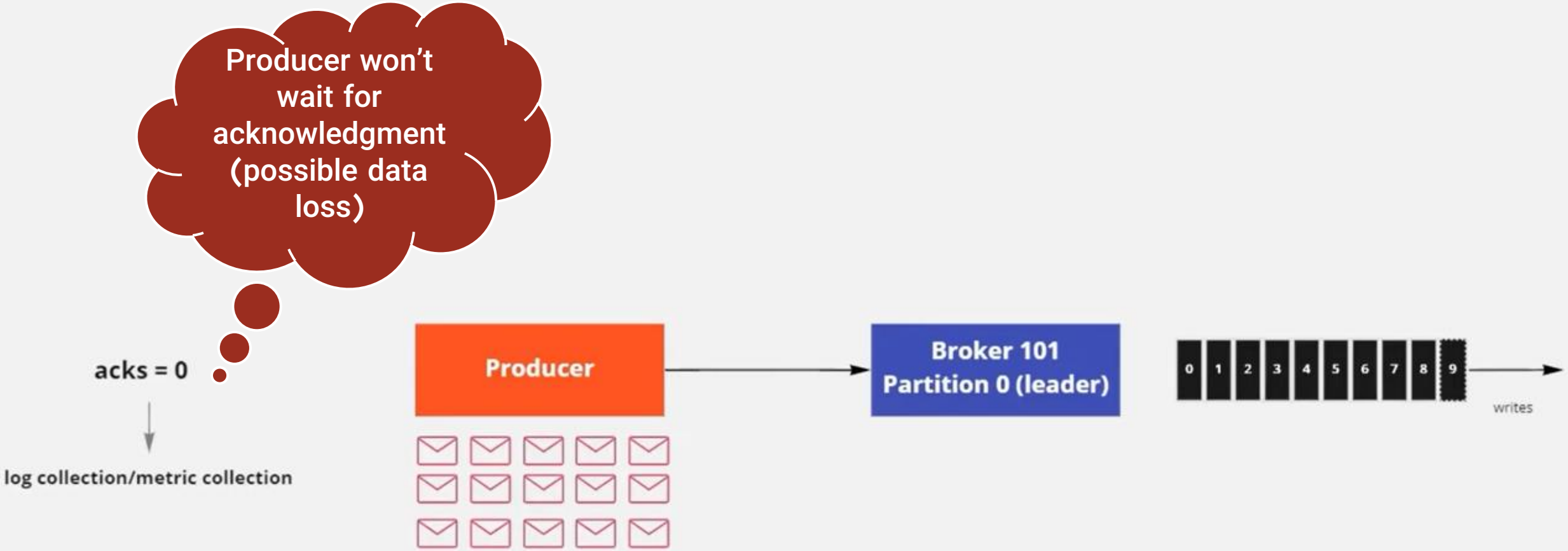
Producers پیام‌های جدیدی ایجاد می‌کنند، آن‌ها را دسته‌بندی می‌کنند و به یک موضوع Topic می‌فرستند.

با استفاده از TCP connection دو طرفه با broker حرف می زنه

A Producer also **balances** messages across all partitions of a topic.

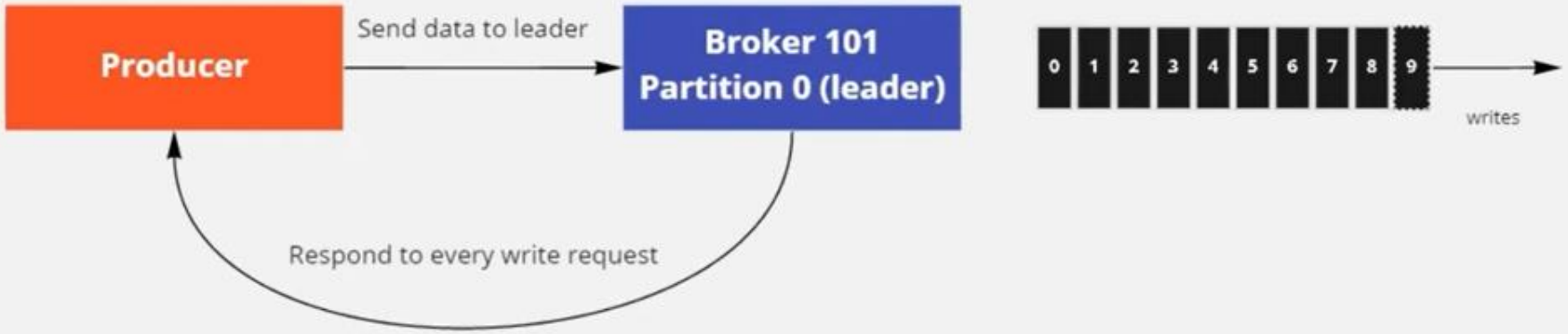
شما همچنین می توانید یک استراتژی پارتیشن بندی سفارشی ارائه دهید.
اگر message ها key=null باشد دیتا ها (message ها) به صورت round robin ارسال می شود اما اگر key بزاریم Kafka تضمین می کند همیشه داخل یک partition ذخیره بکنیم.
اما key اساسا زمانی ارسال میشه که ما احتیاج داریم message ها ترتیب داشته باشند.

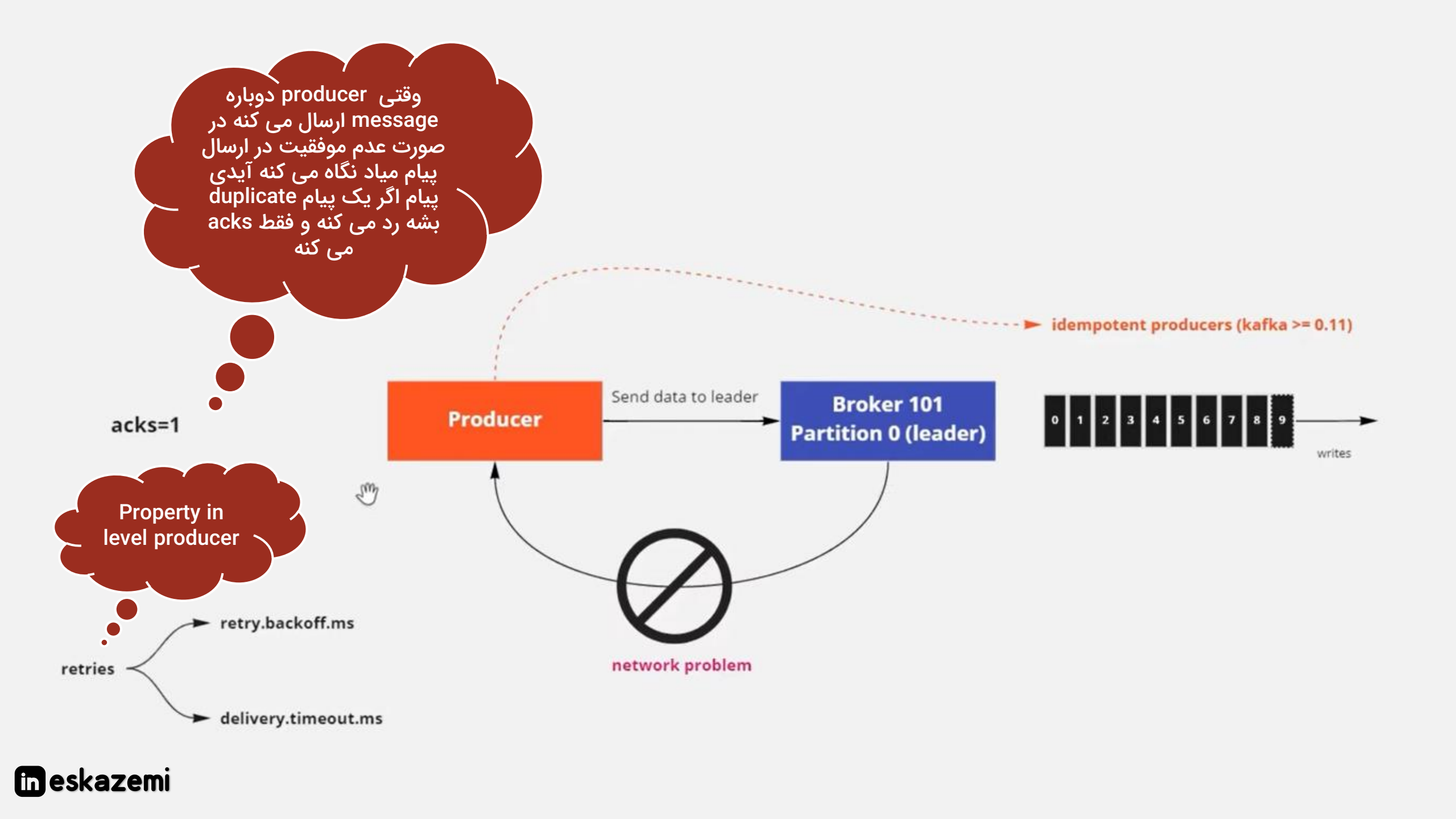
- ✓ زمانی که broker جدید start می شود همه producerها به طور خودکار شروع به جست و جو broker جدید کرده و پیام ها را به آن ارسال می کنند

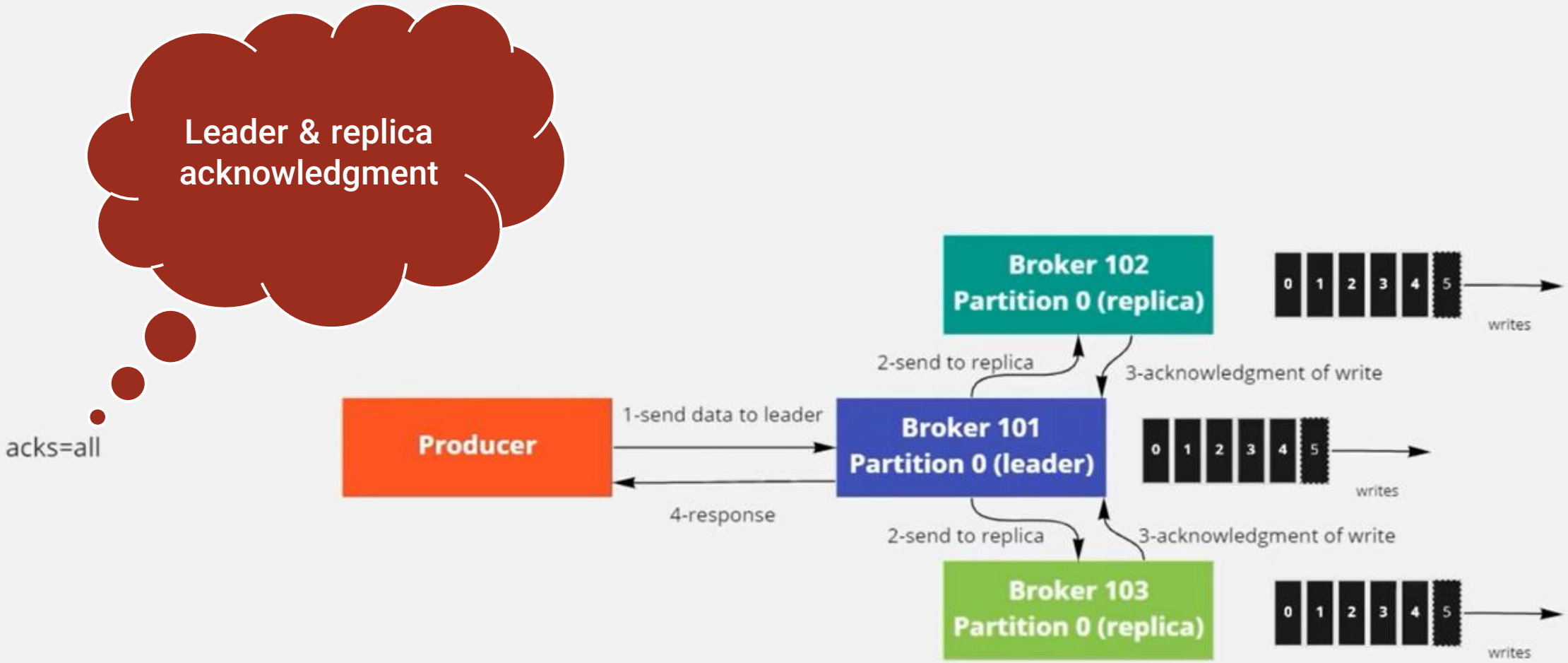

Producer won't wait for acknowledgment (possible data loss)
acks = 0
log collection/metric collection
Producer
Broker 101
Partition 0 (leader)
0 1 2 3 4 5 6 7 8 9
writes

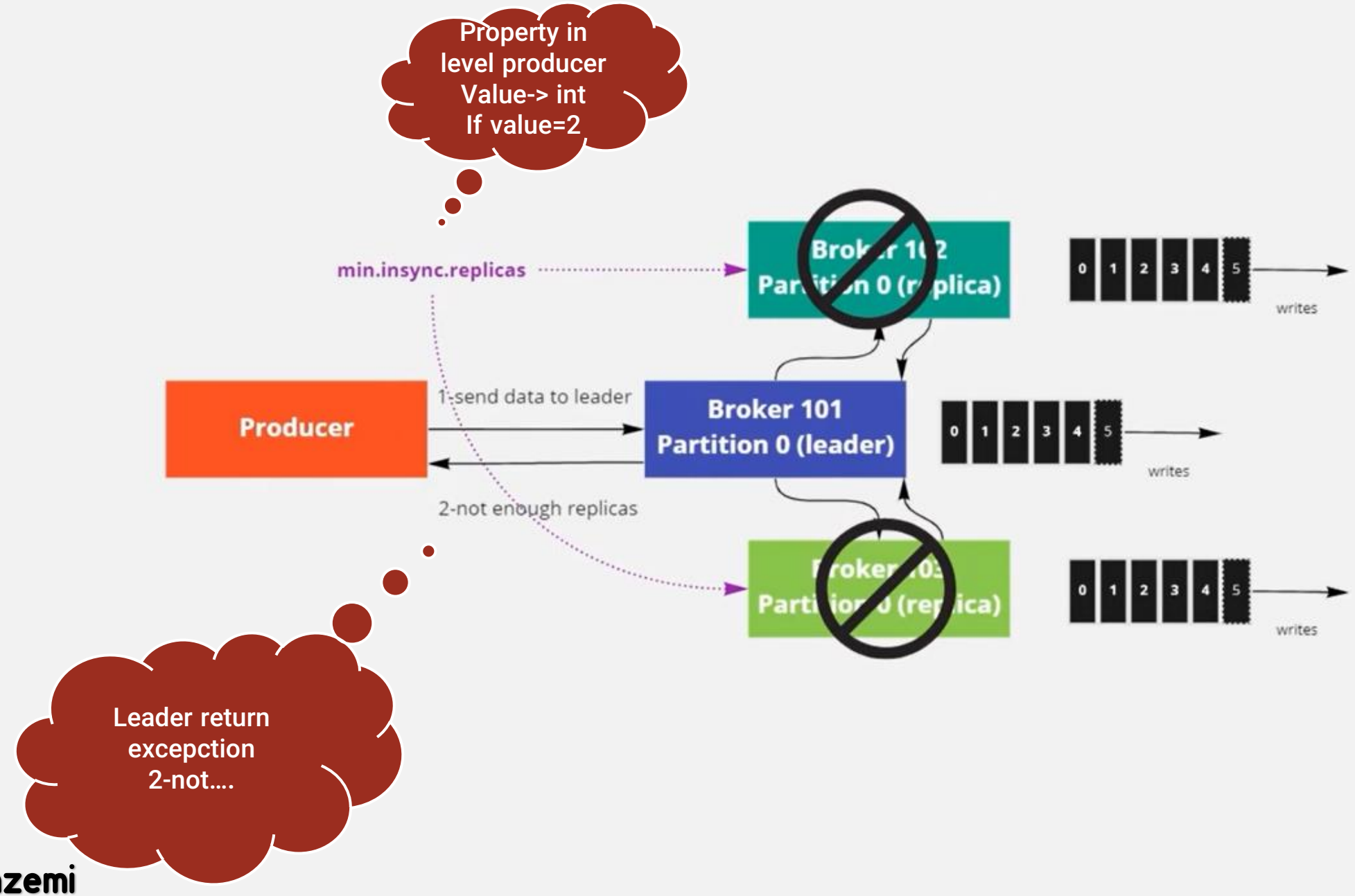
Default producer will wait for leader acknowledgment (limited data loss)

acks=1

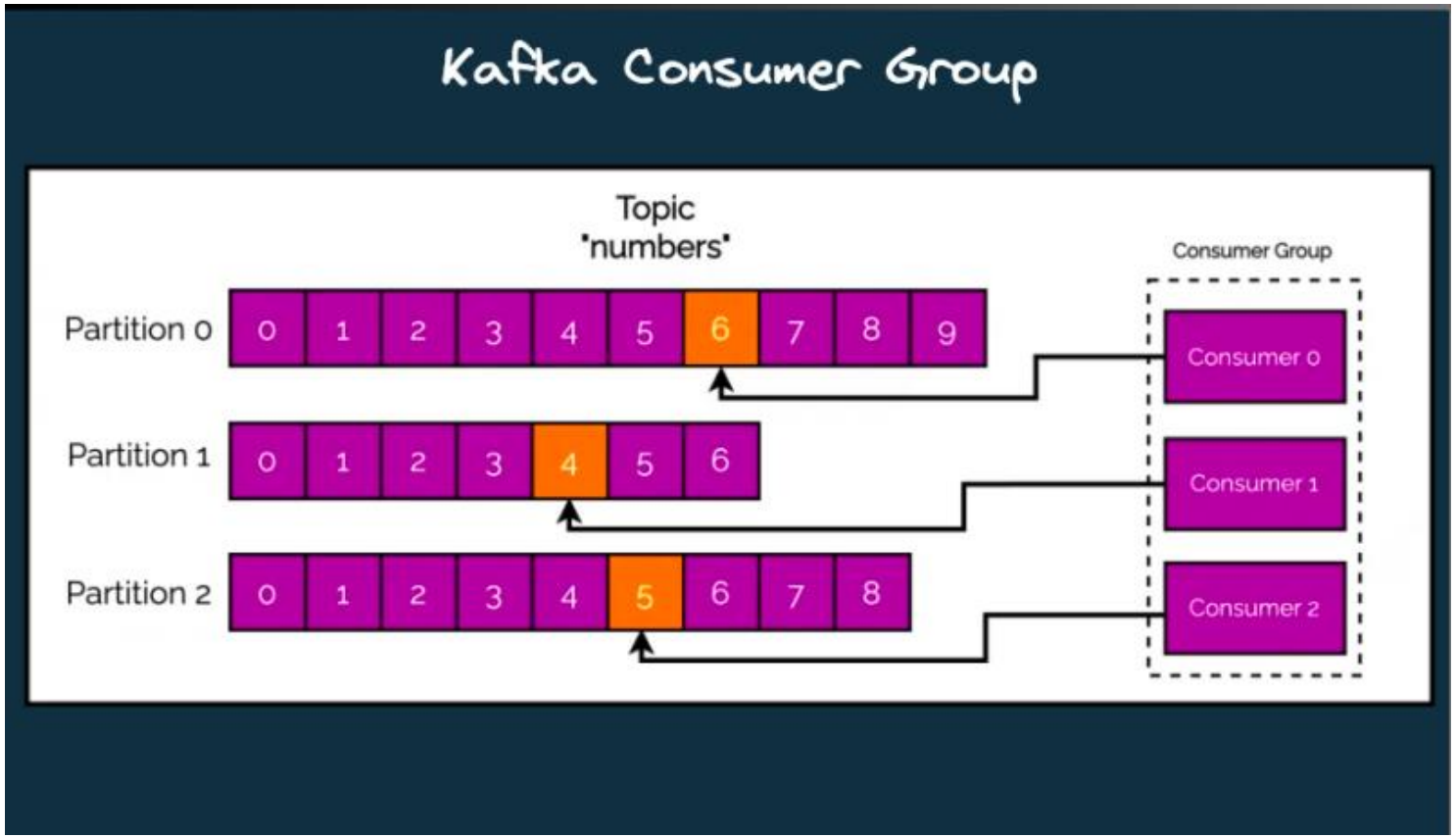



وقتی producer دوباره message ارسال می کنه در صورت عدم موفقیت در ارسال پیام میاد نگاه می کنه آیدی پیام اگر یک پیام duplicate بشه رد می کنه و فقط acks می کنه
acks=1
Property in level producer
retries
retry.backoff.ms
delivery.timeout.ms
Producer
Send data to leader
Broker 101
Partition 0 (leader)
idempotent producers (kafka >= 0.11)
0 1 2 3 4 5 6 7 8 9
writes
network problem


Leader & replica acknowledgment
acks=all
Producer
1-send data to leader
4-response
Broker 101
Partition 0 (leader)
Broker 102
Partition 0 (replica)
Broker 103
Partition 0 (replica)
2-send to replica
3-acknowledgment of write
2-send to replica
3-acknowledgment of write
0 1 2 3 4 5
writes
0 1 2 3 4 5
writes
0 1 2 3 4 5
writes


Property in
level producer
Value-> int
If value=2
min.insync.replicas
Broker 102
Partition 0 (replica)
0 1 2 3 4 5
writes
Producer
1-send data to leader
Broker 101
Partition 0 (leader)
0 1 2 3 4 5
writes
2-not enough replicas
Broker 103
Partition 0 (replica)
0 1 2 3 4 5
writes
Leader return
excepction
2-not....

Consumer

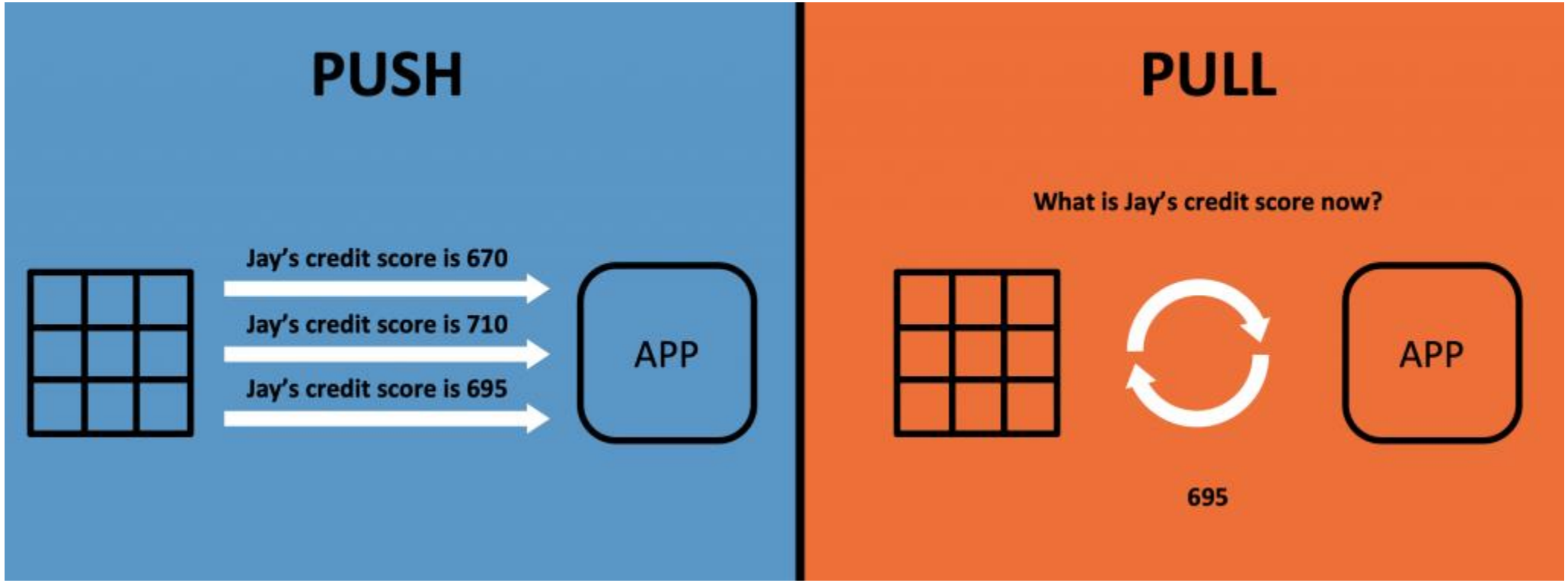


- دریافت کننده پیام یا رابط مصرف کننده Application، سیستم یا زیرسیستمی که بر روی یک یا چند Topicخاص، Subscribe کرده‌است به عبارتی دیگر به یک یا چند تاپیک متصل شده و به پردازش رکوردها می پردازد.
- با استفاده از TCP connection دو طرفه با broker حرف می زنه.
- **consumer group:** یک یا چند consumer به عنوان یک گروه consumer برای مصرف پیام های یک Topic کار می کنند. عموما این گروه شامل یک Replicate از یک Application است؛ مانند گروه ارسال کننده ایمیل
- یک نمونه consumer در یک گروه به یک پارتیشن خاص گره خورده است. به این معنی که consumer مالک پارتیشن است

Pull-based message/data consumption

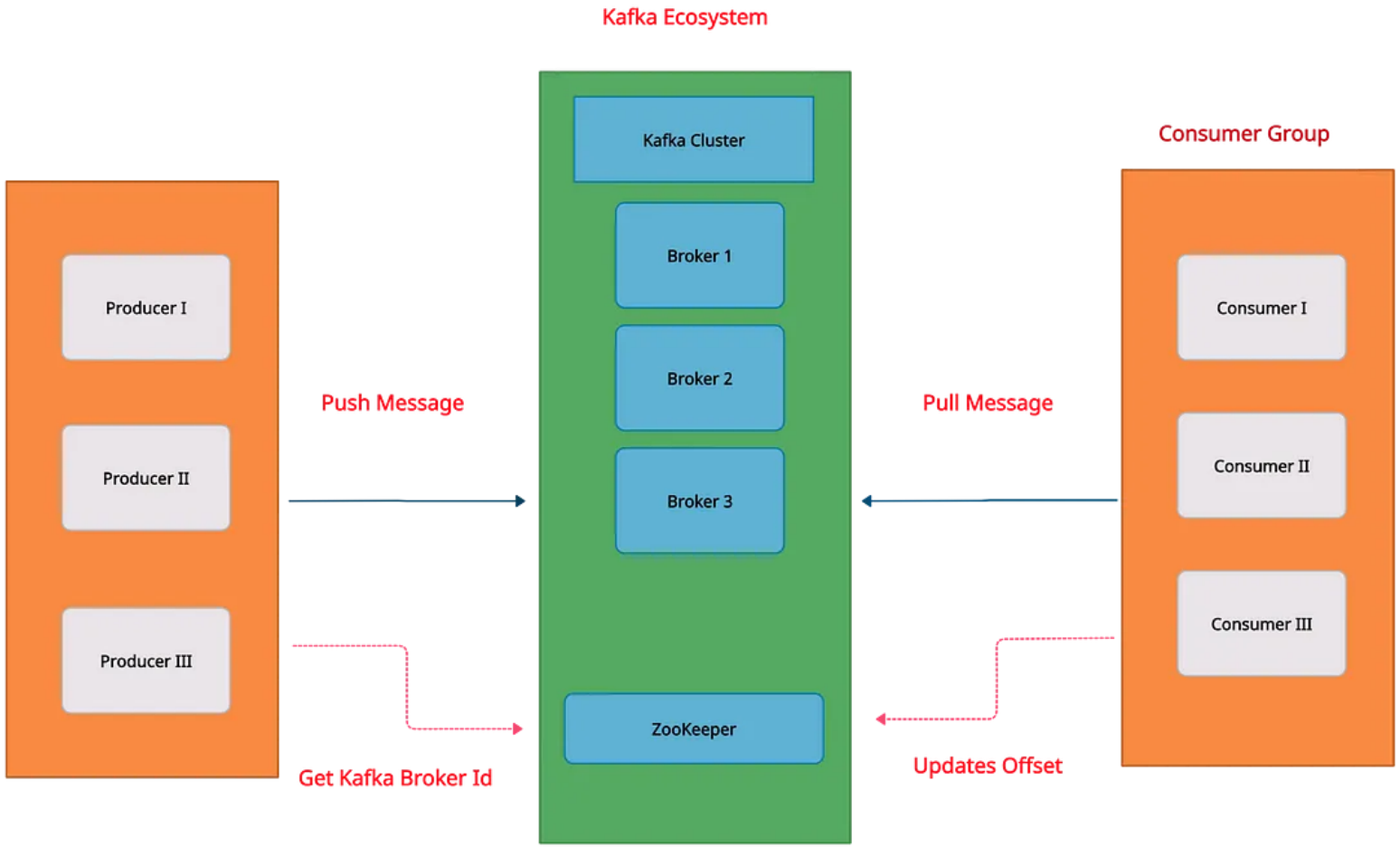
مصرف پیام Pull-based یک مدل مصرف داده است که در آن consumer به جای ارسال خودکار پیام ها توسط broker به consumer ها پیام ها را pull می کند (به طور فعال داده ها را درخواست می کند)

.در این مدل، consumer زمانی که آماده پردازش داده‌ها است، بازیابی داده‌ها را آغاز می‌کند و می‌تواند نرخ مصرف داده‌ها را کنترل کند.



Apache Kafka: Pull-based method

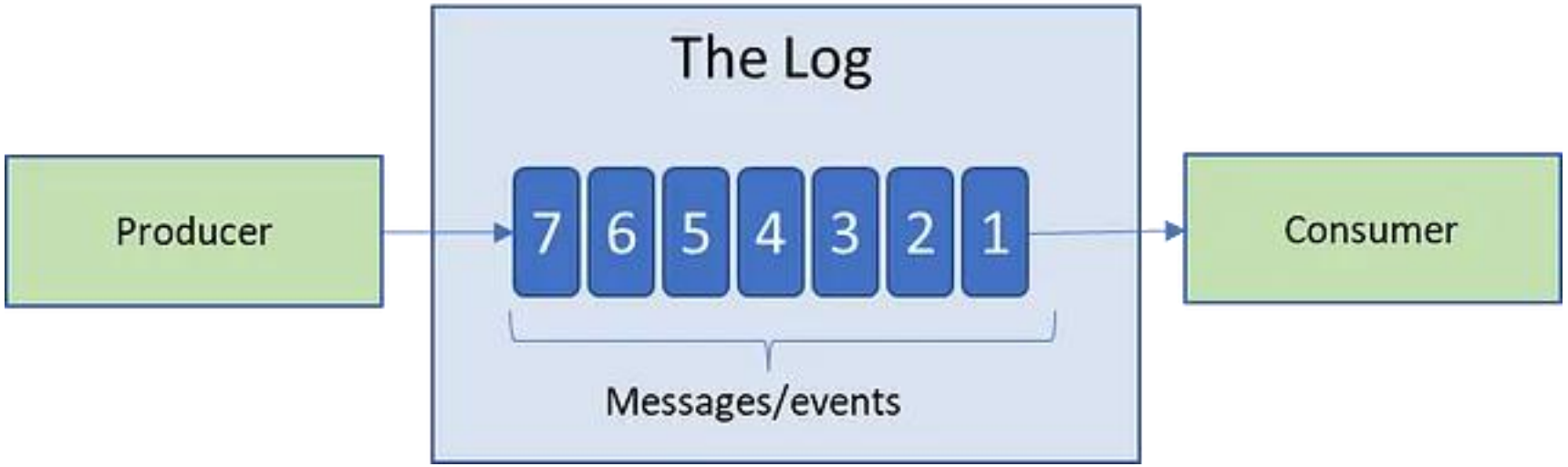
در سیستم‌های pull-based، broker ها منتظر می‌مانند تا consumer اطلاعات را بخواهد (" pull "). اگر consumerدیر کند، می تواند بعداً می تواند پیام ها را مصرف کند(pull).
در این سیستم ها Dumb broker / smart consumer مدل خواهیم داشت broker پیام هایی را که توسط مصرف کنندگان خوانده می شود ردیابی نمی کند و فقط پیام های خوانده نشده را نگه می دارد.
Consumer ها ایندکس پارتیشن هارو نگه میدارن و بصورت پی‌درپی پیام‌هارو میخونن

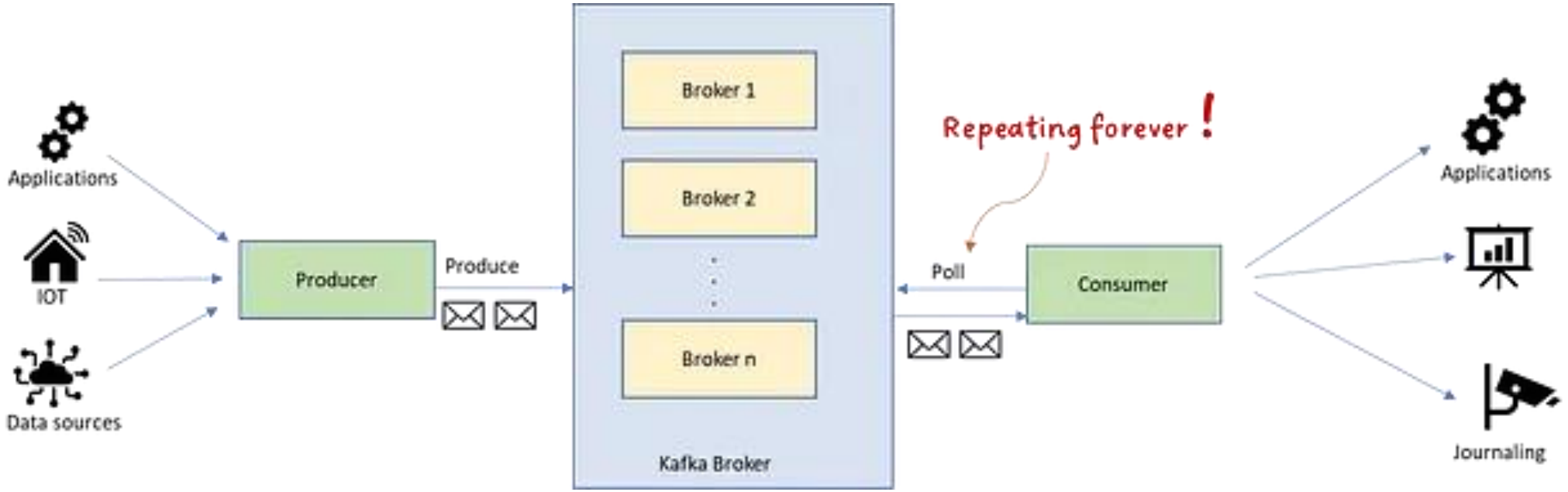


Why does Kafka use a pull-based message consumer model?

کافکا یک queue نیست، یک log است - log از پیام ها/رویدادها. Producer ها به log اضافه می‌کنند و مصرف‌کنندگان از log می‌خوانند.

log تغییر ناپذیر است اما نمی تواند تعداد بی نهایت داده را ذخیره کند، بنابراین یک زمان پیکربندی شده برای زنده ماندن سوابق در log وجود دارد. از سوی دیگر در یک queue، پیام ها پس از پردازش/دریافت توسط مصرف کننده حذف می شوند.

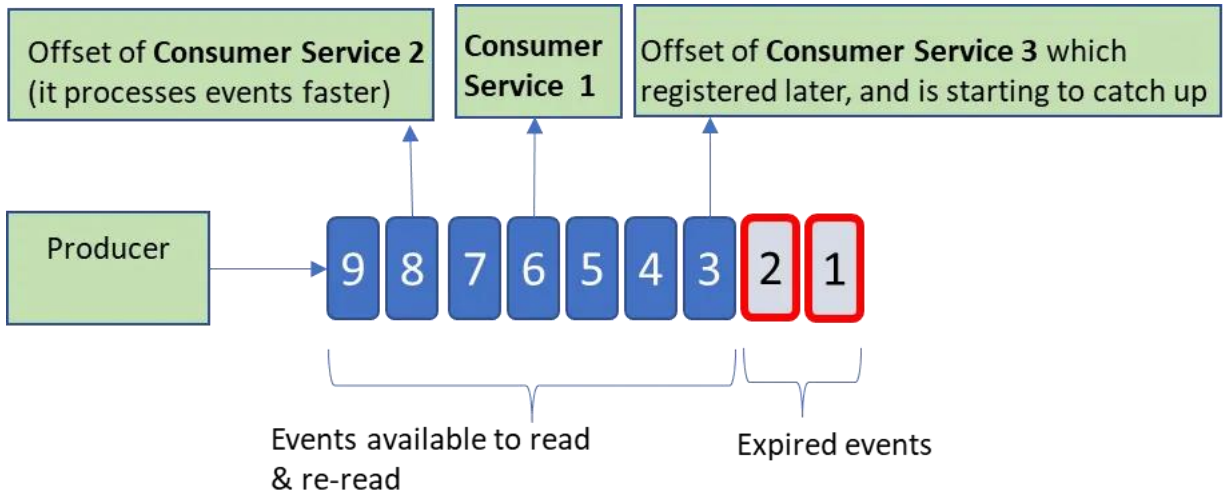




Producer ها یک درخواست تولید همراه با رکوردها را به log ارسال می‌کنند و به هر رکورد، با رسیدن، همان طور که گفتیم شماره خاصی به نام offset داده می‌شود که موقعیت logical آن رکورد در log است. مصرف کنندگان یک درخواست واکشی برای خواندن سوابق ارسال می کنند، و آنها از offsetها برای bookmark، مانند متغیرهای مکان، استفاده می کنند.

Now let’s see how the pull model helps Kafka.

Kafka is deployed with consumers varying in processing capacities & requirements. For simplicity concept of partition is not shown in this image.



1. Diverse consumers

Schema های log به چندین گروه consumer مستقل اجازه می‌دهد تا با سرعتی که برای گروه consumer امکان‌پذیر است، از log بخوانند.همانطور که در تصویر بالا نشان داده شده است، مدل consume Pull-based consumer به consumer های متنوع با نرخ های مصرف متفاوت اجازه می دهد. برای مصرف رکورد ها.

2 - مدل pull-based consumption کافکا کنترل دقیق تری بر پردازش پیام، مدیریت خطا، acknowledgement و مدیریت offset و امکان مصرف مجدد داده ها در صورت نیاز به consumers می دهد.

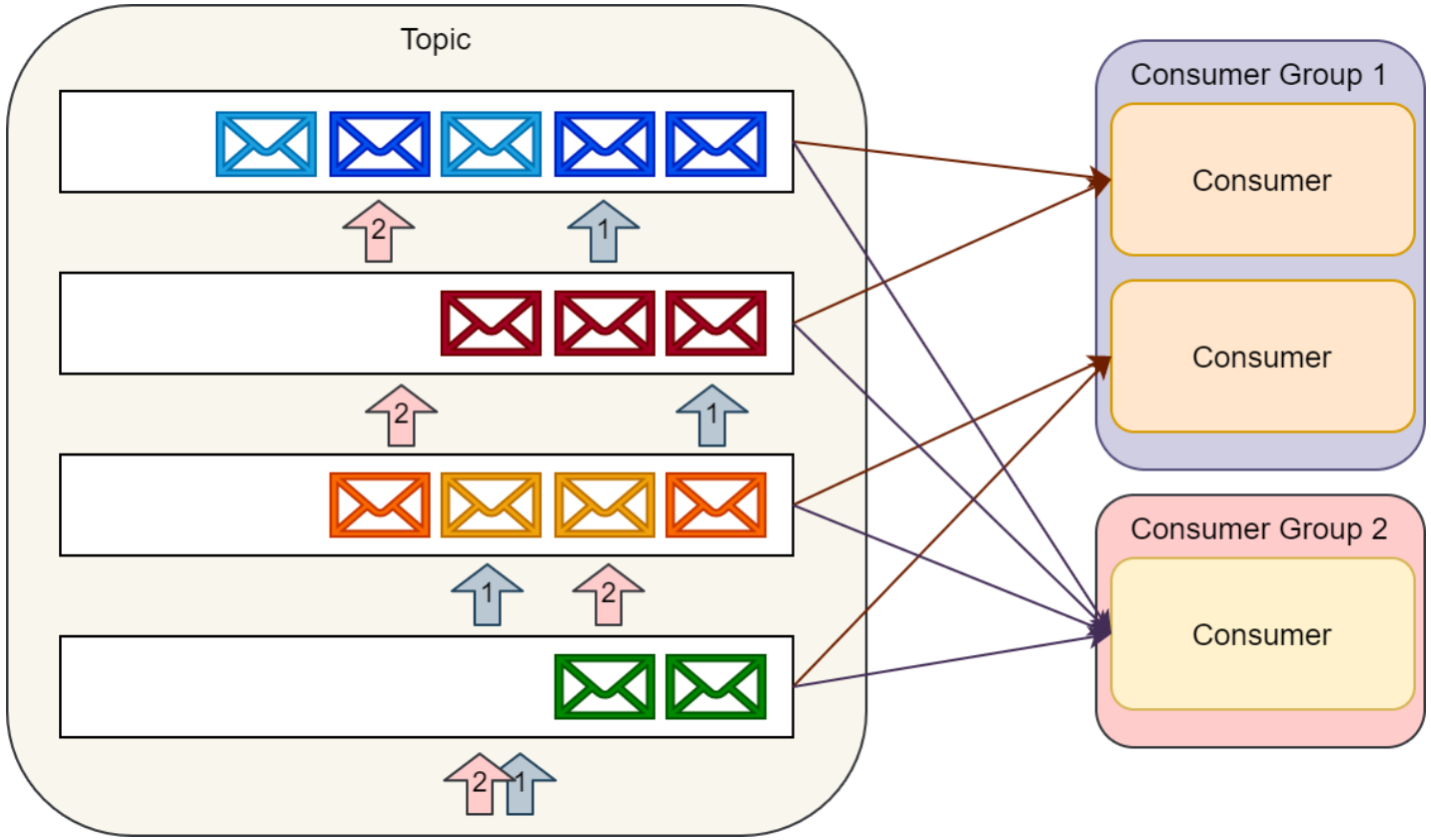
مزیت دیگر یک سیستم pull-based این است که به دسته بندی کارآمد داده های ارسال شده به consumer کمک می کند. در حالی که یک سیستم push-based باید انتخاب کند که یا درخواستی را فوراً ارسال کند یا داده های بیشتری را جمع آوری کند و بعداً بدون اطلاع از اینکه downstream consumer قادر به پردازش فوری آن خواهد بود یا خیر ارسال کند.

نقص کلی یک سیستم pull-based ساده این است که اگر broker داده ای نداشته باشد،

the consumer may end up polling in a tight loop wasting resources

منابع را در حالی که منتظر رسیدن داده است هدر دهد.

برای جلوگیری از این امر، کافکا پارامترهایی در درخواست واکشی خود دارد که به درخواست consumer اجازه می‌دهد تا در یک «long poll»، منتظر بمانند تا داده‌ها در دسترس باشند یا تا زمان انقضا، مسدود شود.



هر consumer به تنهایی میتونه دیتای چندین topic رو بخونه.

همچنین میشه تعداد consumerها رو به تعداد پارتیشن های موجود scaleکرد. درنتیجه زمانی که میخوایم یک topicبسازیم باید به دقت ، توان messaging ای که از topic انتظار داریم رو در نظر داشته باشیم.

از اونجایی که consumerها ایندکس پارتیشن رو نگه میدارن پس می‌تونن مشخص کنن که subscriptionشون بصورت durableباشه که بعد از هر بار ریستارت ادامه پیام‌هارو پردازش کنن یا بصورت ephemeralباشه و با هر بار ریستارت از آخرین پیامی که وارد پارتیشن شده کارشونو شروع کنن.

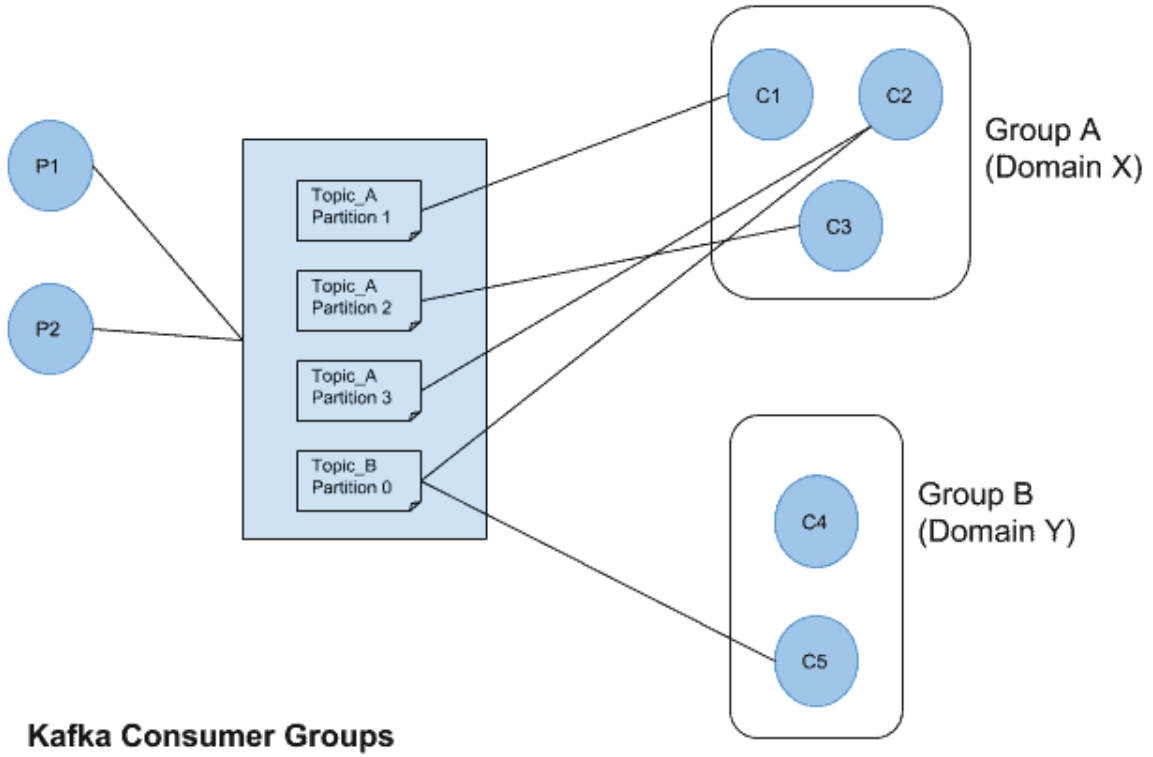
کافکا همچنین دارای پشتیبانی داخلی برای فیلتر کردن پیام است. این به consumer ها اجازه می‌دهد فقط پیام‌هایی که به آن‌ها علاقه دارند subscribe کنند. برای مثال، یک consumer می‌تواند فقط پیام‌هایی با یک کلید یا مقدار خاص را subscribe کنند.

هر partition حداکثر توسط یک consumer از **یک گروه consume میشه** یعنی یک partition نمی تواند همزمان توسط دو consumer از یک گروه consume بشه.

دو معماری message-queue و pub/sub قابل پیاده سازی در kafka هستند

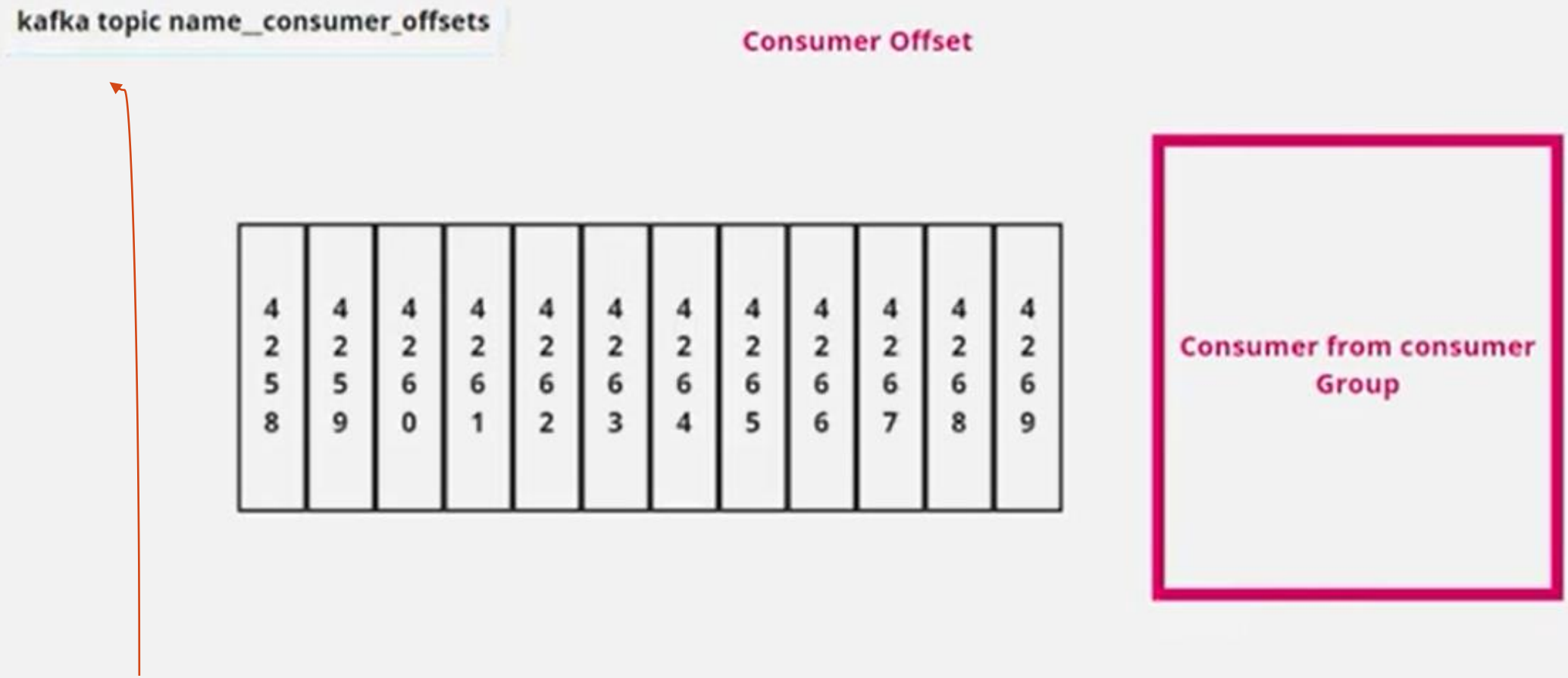
زمانی که بخواهیم پترن Queue داشته باشیم همه consumer ها را داخل یک گروه قرار می دهیم و هر consumer وظیفه consume هر partition داره ولی خب این پترن می‌تونه یه سری مشکلات داشته باشه.

وقتی که پترن pub/sub می خواهیم داشته باشیم هر consume داخل گروه uniqe قرار می دهیم در نتیجه partition توسط چندین consumer در گروه های مختلف می تواند consume شود.

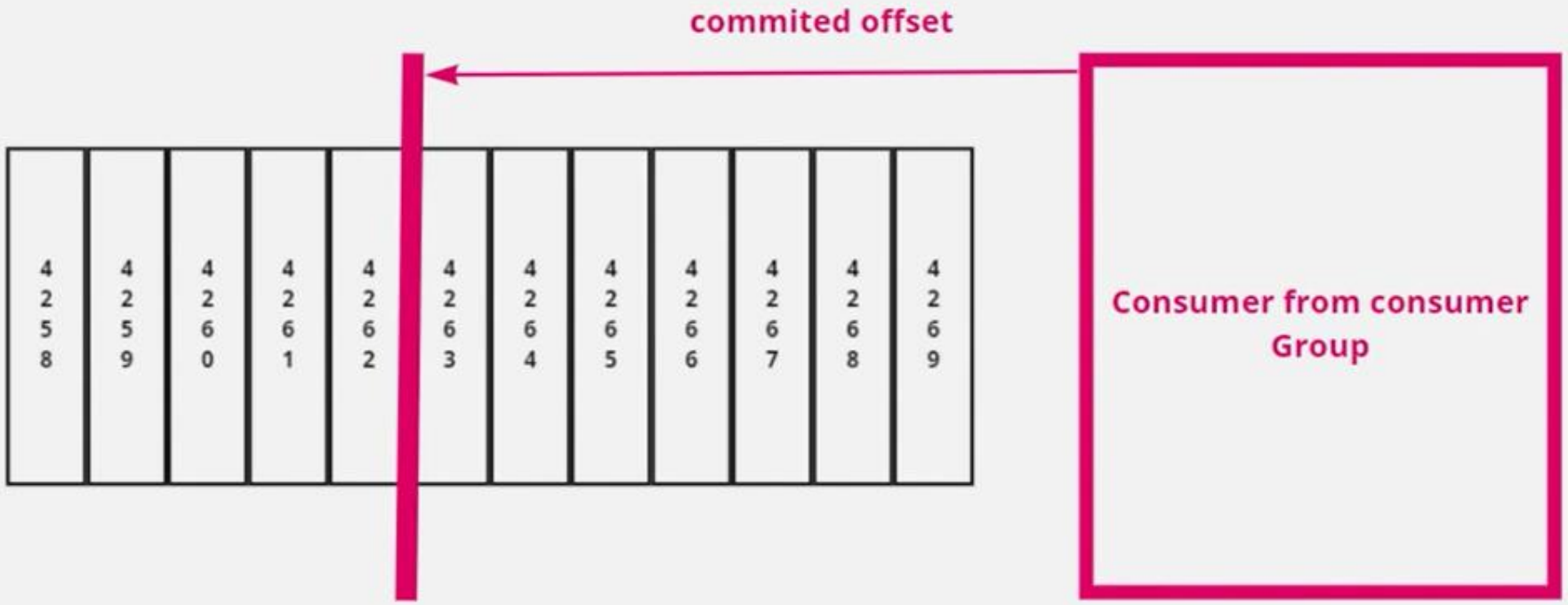


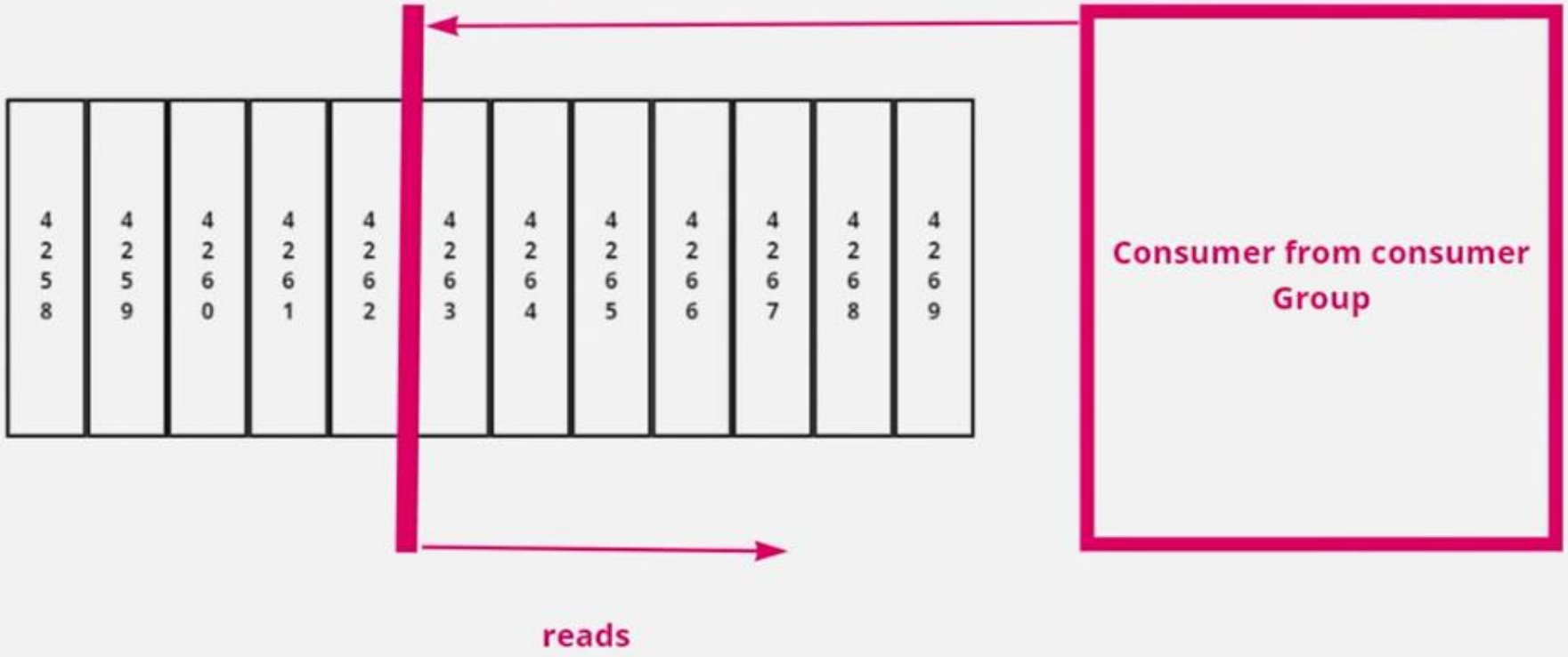
Kafka Consumer Groups

همان طور که گفتیم consumer ها توانایی دارن که offset ها تا جایی که read کردن restore کنن.



زمانی که consumer داره پیام ها رو process می کنه می آید offset را commitمی کنه یعنی action صدا می زنه.


commited offset
4258
4259
4260
4261
4262
4263
4264
4265
4266
4267
4268
4269
Consumer from consumer Group



زمانی که consumer (مصرف کننده) down بشه این اتفاق باعث میشه که بتوانیم Read back کنیم از جایی که Left off شده.

Commit offset تاثیر داره روی Deliver semantics

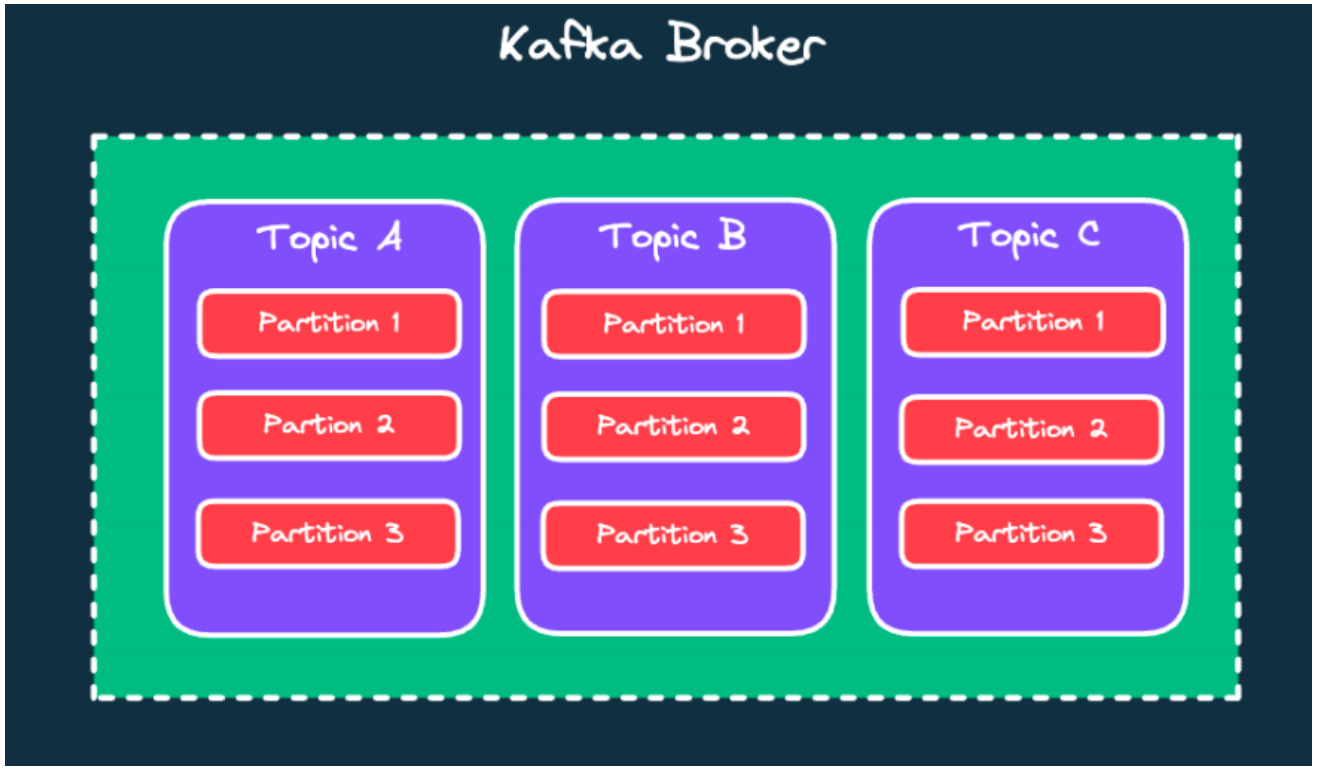
به محض رسیدن پیام به consumer
Offset = commited
این خوب نیست چون اگر processing پیام به هر دلیلی به مشکل بر بخوره پیام و دیتا ما loss میشه

Offset زمانی commit میشه که process پیام تمام بشه در نتیجه پیام از بین نمی ره فقط مشکلی که پیش میاد idempotent

Idempotent -> برای مدیریت پردازش پیام تکراری بهتره این رویکرد را در نظر بگیریم
پردازش یک پیام تکراری نباید تاثیری توی سیستم ما داشته باشد.

Broker

A single Kafka server is called a Broker
A broker is part of a Kafka cluster
Port default : 9092



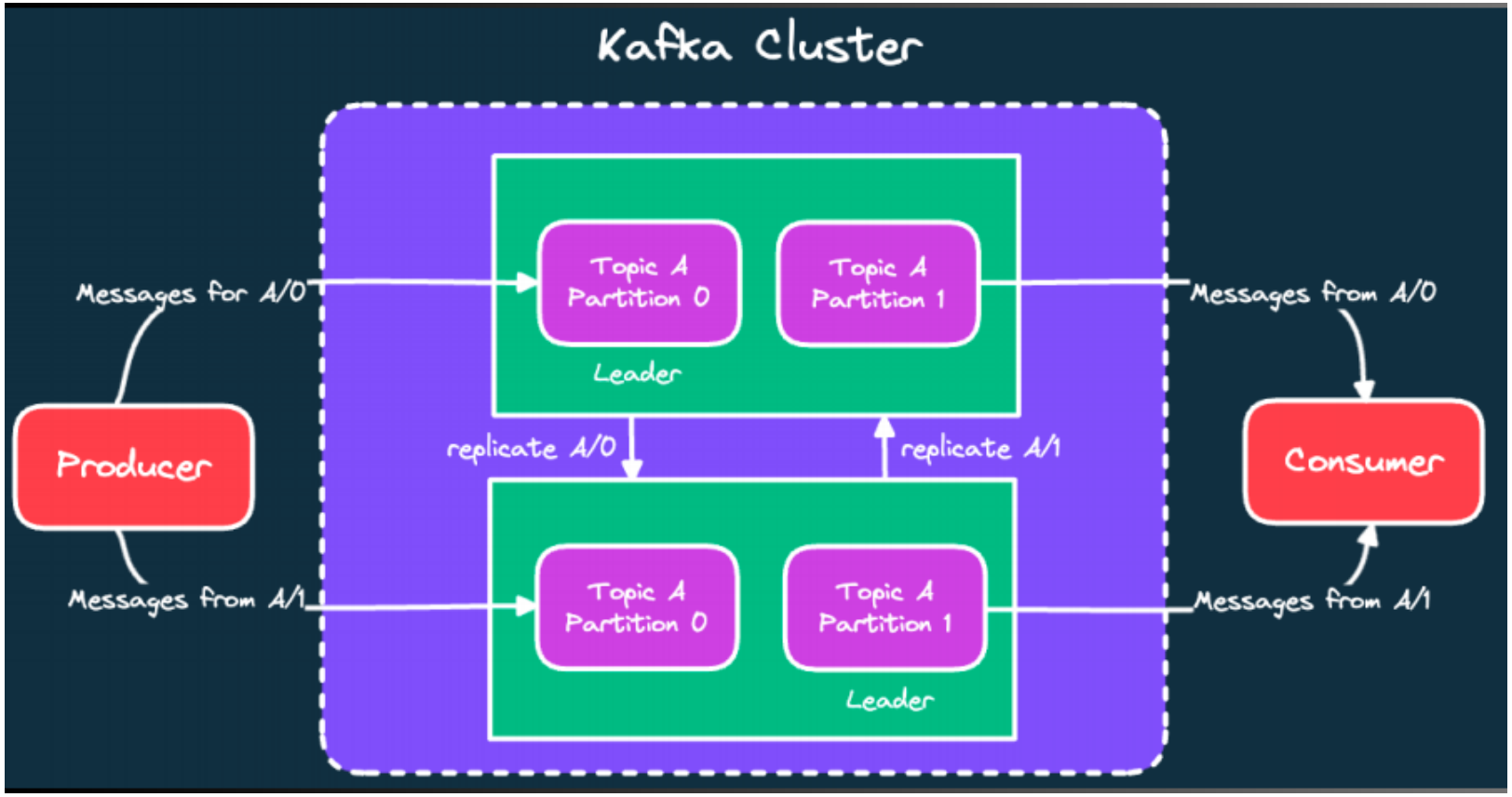
یک Broker می تواند هزاران پارتیشن و میلیون ها پیام را در هر ثانیه مدیریت کند

یک Broker برای برقراری ارتباط روی TCP ها listen می کنه

Broker مانند یک پل است چون قسمتی که تمامی پیام‌ها را از Producer دریافت می‌کند، سپس آن‌ها را در Log مربوط به Topicمشخص شده ذخیره می‌کند

Cluster

مجموعه ای از Brokerها می‌باشد که بصورت یک Cluster اجرا شده‌اند. این کار باعث بالا رفتن کارآیی و تحمل خطا می‌شود.



Kafka brokerها stateless هستند

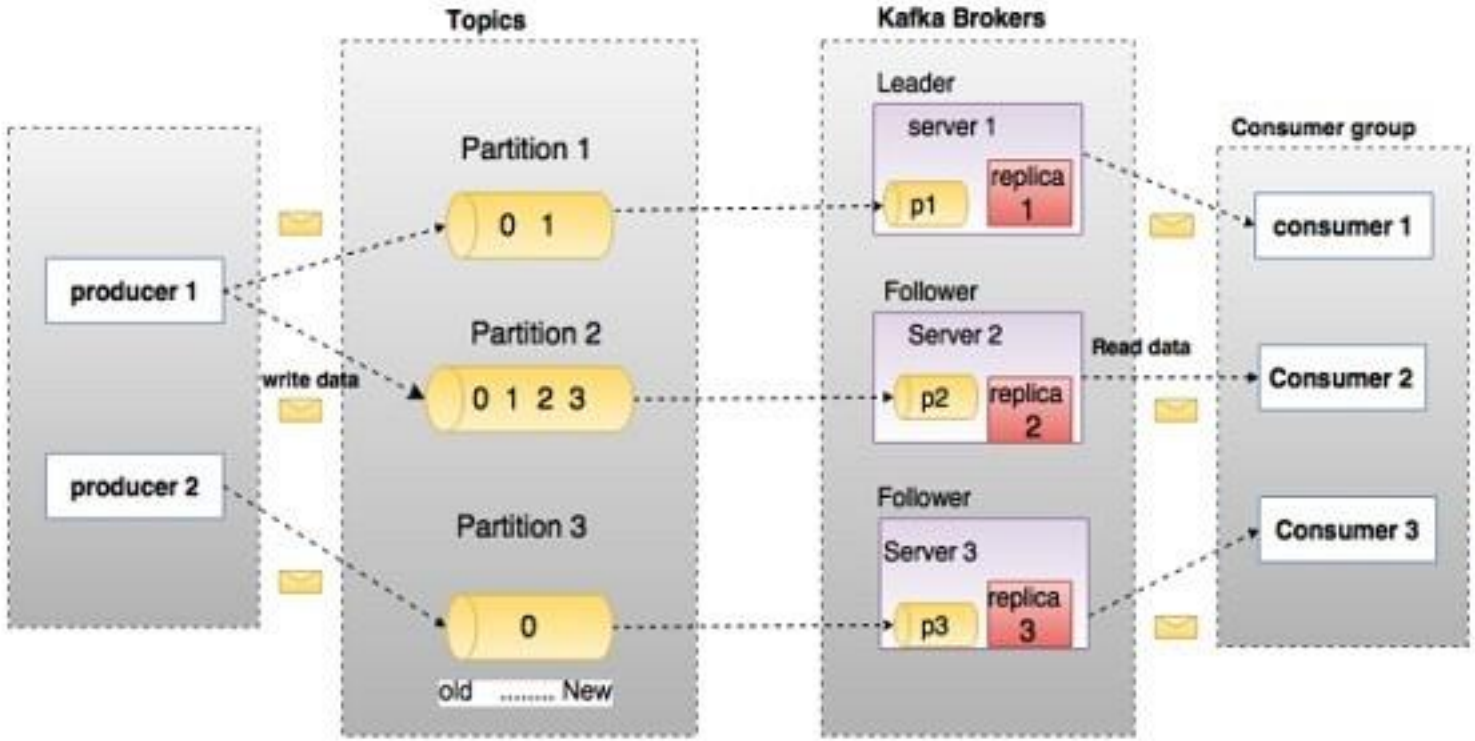
یک Kafka Cluster به شما امکان می دهد پیام ها را در یک پارتیشن مشخص تکرار کنید و در نتیجه این پارتیشن در چندین broker تکرار می شود

یک Broker کنترل کننده Cluster است و به صورت خودکار(توسط zookeeper) انتخاب می شود

Replicas Brokers

Kafka Serverهایی هستند که شامل یک کپی از Partition می‌باشند. عملیات خواندن و نوشتن در Partitionتوسط Primaryانجام می‌شود. در صورتیکه Primaryاز دسترس خارج شود، ZooKeeper یکی از Replicas Brokerها را بعنوان Primaryدر نظر می‌گیرد. همچنین این نکته را باید در نظر بگیرید که هر Replicateهمزمان می‌تواند Primary پارتیشن‌های دیگر باشد.

انتخاب Kafka broker leaderتوسط zookeeperانجام می شود.



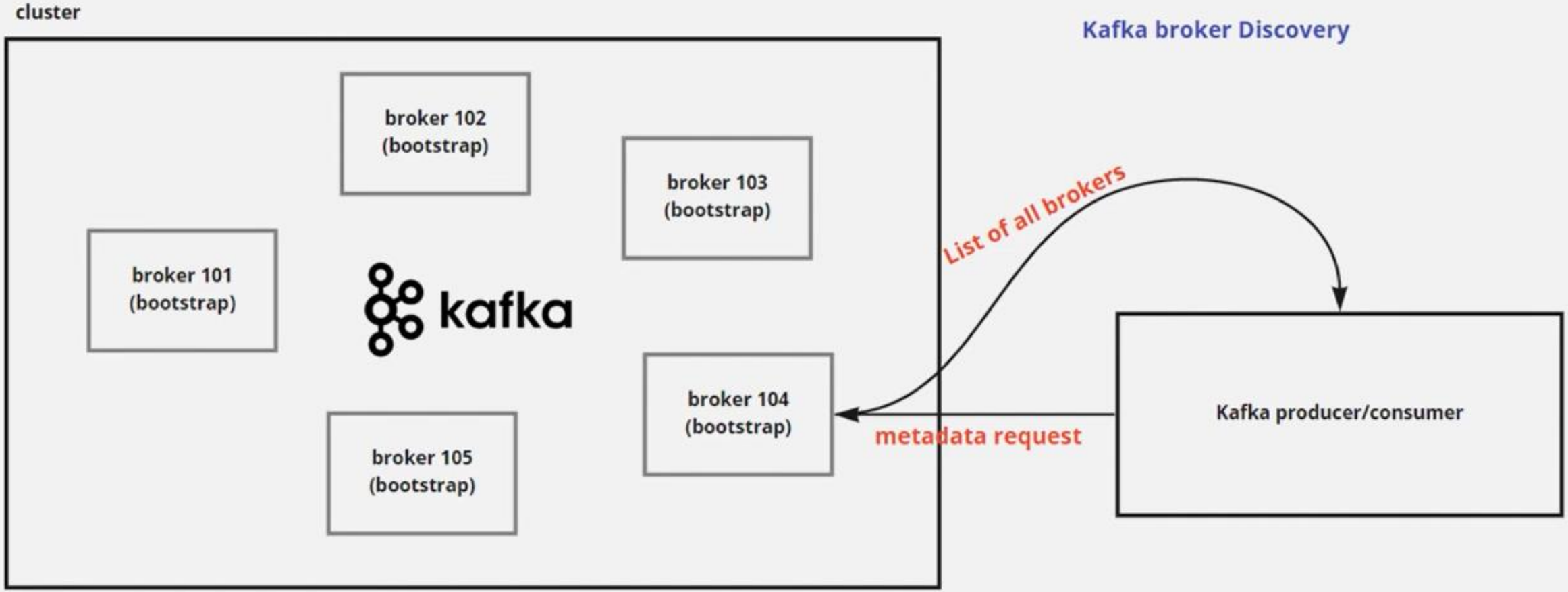
Replication Factor

این خصوصیت احتمال از دست دادن داده‌های یک Topicرا به حداقل می‌رساند؛ به این صورت که هر پیام از یک Topic، در چندین سرور مختلف که تعداد آنها توسط این خصوصیت مشخص می‌شود، نگهداری می‌شود.

Primary Broker

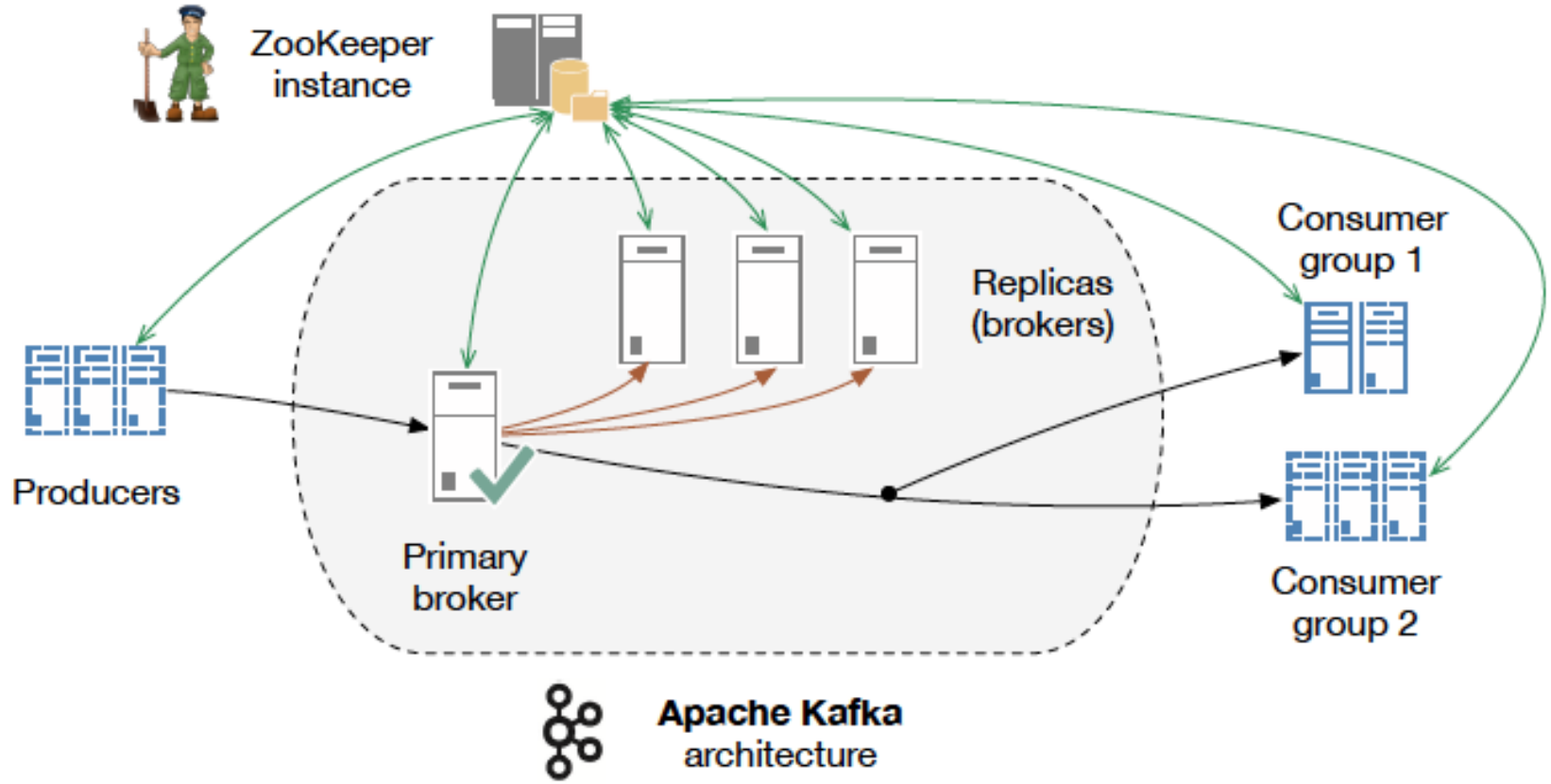
یک Kafka Server که مسئول خواندن و نوشتن در یک Partition است. در یک Clusterهر Partitionدر یک زمان تنها یک Primary Broker دارد. این Primary Brokerهمزمان می‌تواند برای Partitionهای دیگر نقش Replicas Brokerرا بازی کند. انتخاب یک Primary Broker برای یک Partition توسط ZooKeeper انجام می‌شود.

در cluster هر broker در مورد سایر broker ها و topic ها و
partition ها اطلاعات دارد که بهش می گوییم Meta data



Apache ZooKeeper

Kafkaهیچ Stateای را نگه نمی‌دارد (اصطلاحا statelessمی‌باشد). برای ذخیره کردن و مدیریت تمامی Stateها از جمله اینکه درحال حاضر Primary Broker برای یک Partitionچه سروری است، یا اینکه پیام‌های یک Partitionتا کدام offsetتوسط Consumer ها خوانده شده‌اند یا اینکه کدام Consumer در حال حاضر در یک Consumer Groupمسئول یک Partition می‌باشد، توسط Apache Zookeeper انجام می‌شود.



ضمانت های کافکا

کافکا سه ضمانت برای کار خودش می‌دهد که عبارت اند از :

1- تمامی پیام‌های دریافتی در یک Partitionاز یک Topic، به همان ترتیبی که دریافت می‌شوند ذخیره می‌شوند.
2- در یک Partitionفعال ، consumerها تمامی پیام‌ها را به همان ترتیبی که ذخیره شده‌اند، دریافت می‌کنند.
3- در یک Topicبا Replication Factor ای با مقدار N، درجه تحمل خطا N - 1 می‌باشد.

Advantages of Kafka

1. Handling multiple producers with ease
2. Support multiple consumers without interference
3. Disk-based retention of data
4. Highly scalable
5. built-in partitioning
6. fault tolerant
7. replication

Disadvantages of Kafka

- Overwhelming number of configuration options
- Lack of mature client libraries other than Java or C (but things are improving)

Why is Kafka So Fast?

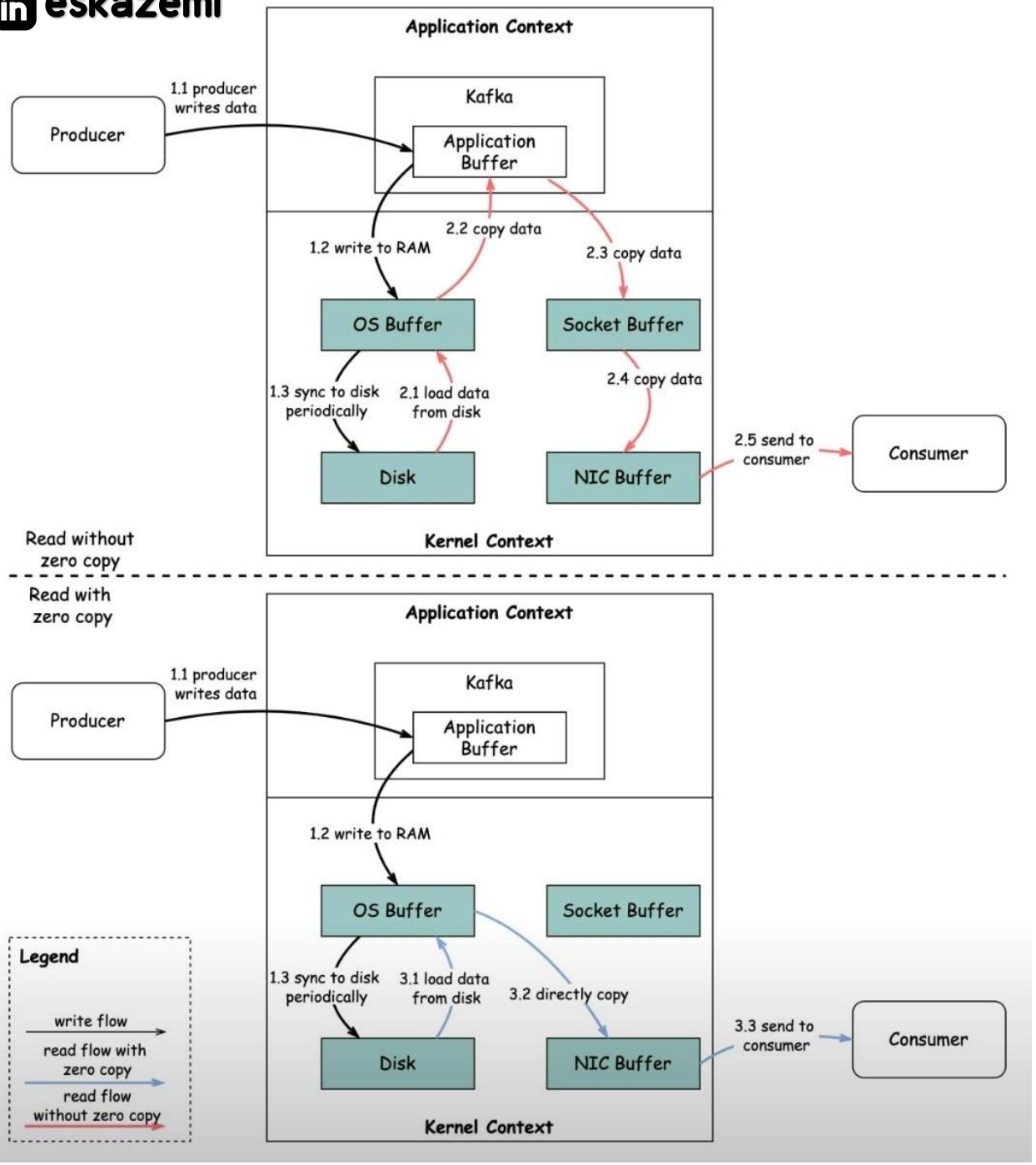
Kafka is known for its high performance
and speed due to several reasons:

The two most important reasons are its low latency message delivery through Sequential I/O and Zero Copy Principle.

Sequential IO

Kafka relies heavily on the filesystem to store and cache messages. The general perception is that "disks are slow, "meaning high seek time. Imagine if we can avoid seeking time. We can achieve low latency as low as RAM here. Kafka does this through Sequential I/O. Kafka efficiently uses a log, an append-only, totally ordered data structure.

eskazemi



ZERO COPY

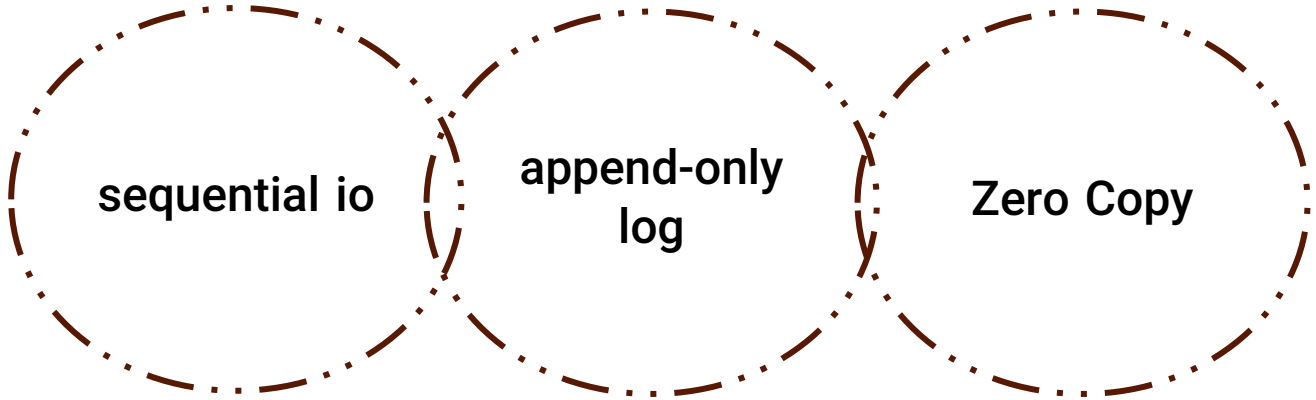
To read a file from a disk, and then send it over the network, a traditional data transfer would require four context switches between user and kernel modes, making the data copied four times.

Kafka uses the zero-copy principle by requesting the kernel to move the data to nic rather than via the application. (Zero-copy optimization consists of removing the second and third data copies and transferring the data directly from the OS buffer to the nic buffer.)

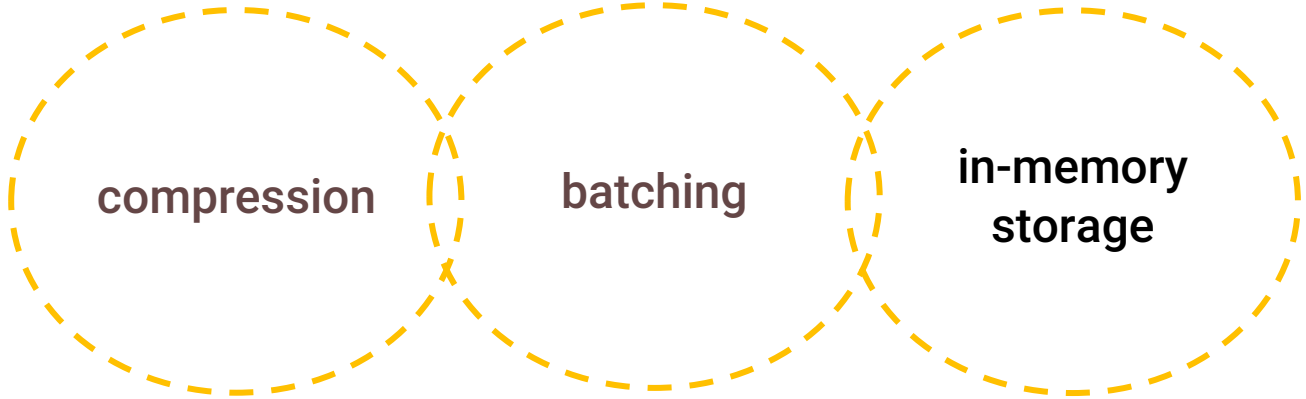
Message Compression

Kafka allows for message compression, which reduces messages' size and enables faster data transmission and processing. Message Batching: Kafka can batch messages together, which reduces the overhead of individual message processing and improves throughput.

به طور کلی، سریع بودن کافکا ناشی از موارد زیر
1- معماری بهینه شده آن شامل 3 مورد زیر



2- فشرده سازی 3 - دسته بندی 4 -ذخیره سازی



Use-Cases of Kafka

- Real-Time Data Strooming
- Batch Data Processing
- Messaging
- Event Sourcing and CQRS

Implementing event-driven architectures for maintaining the state of distributed systems and supporting complex business workflows.

- data integration

Whether it's ETL pipelines, data lakes, or log aggregation; ingesting, organizing, transforming, and distributing data are core functions of Kafka.

- Complex Event Processing(CEP)
- Change Data Capture(CDC)
- log aggregation

Collecting and analyzing logs from distributed systems for monitoring, troubleshooting, and performance analysis.

eskazemi

در مباحث Big data معمولا با دو چالش اصلی مواجهیم

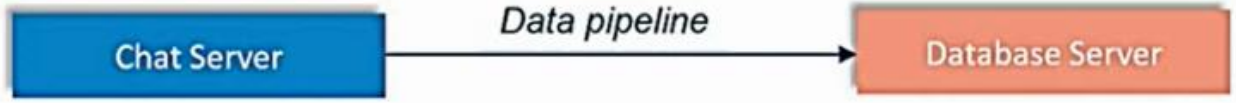
1- جمع آوری حجم زیادی از داده ها
داده ها شامل :
داده های مربوط به رفتار user ها
Event messages
و...
2- تحلیل داده های جمع آوری شده به صورت real-time می باشد.

لازمه ی غلبه بر این دو چالش استفاده از یک **messaging system** **همچون kafka** است.

Real-time streaming data pipelines

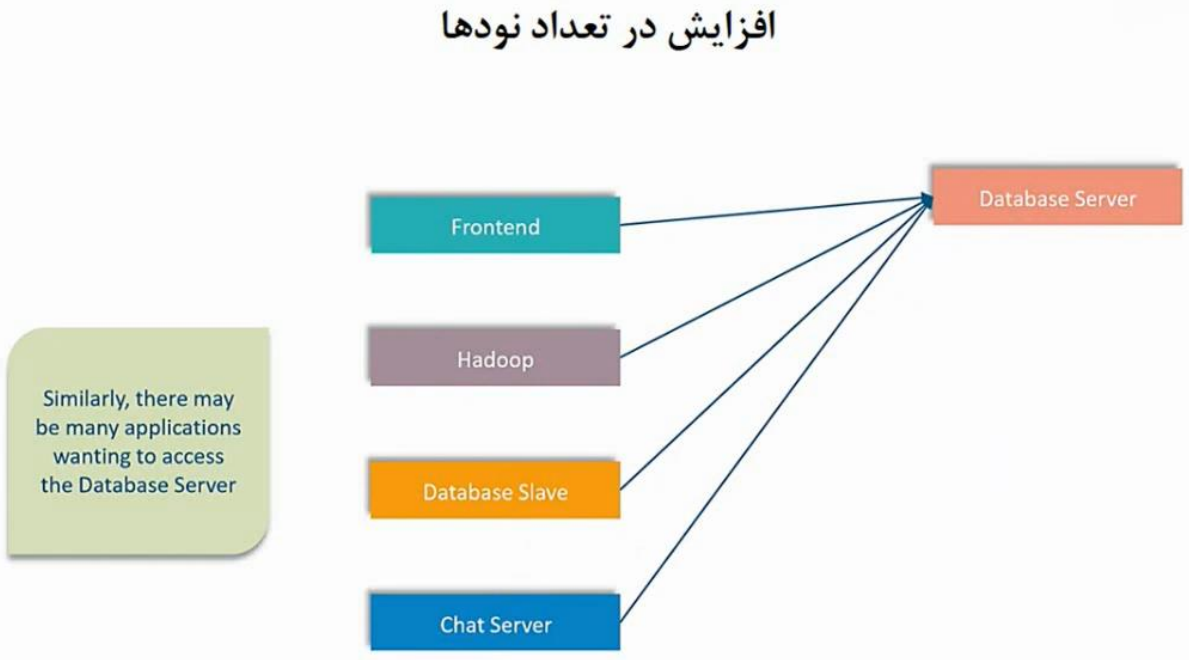
پایپ لاین داده

Communication is required between different systems in the real-time scenario, which is done by using data pipelines.

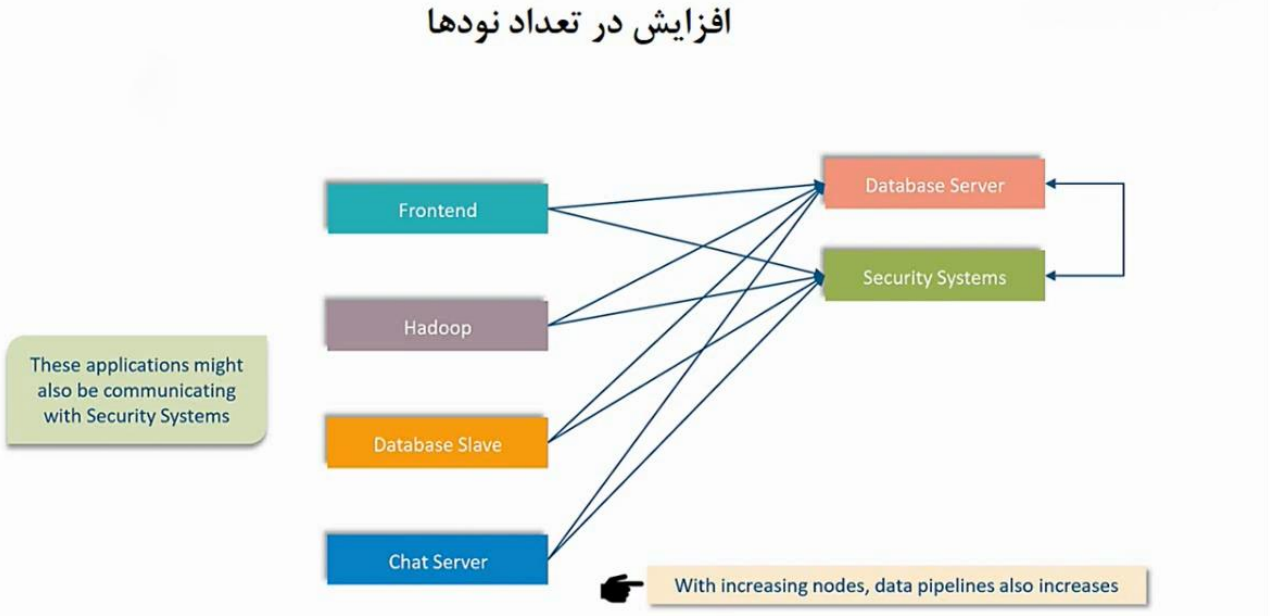


For Example: Chat Server needs to communicate with Database Server for storing messages

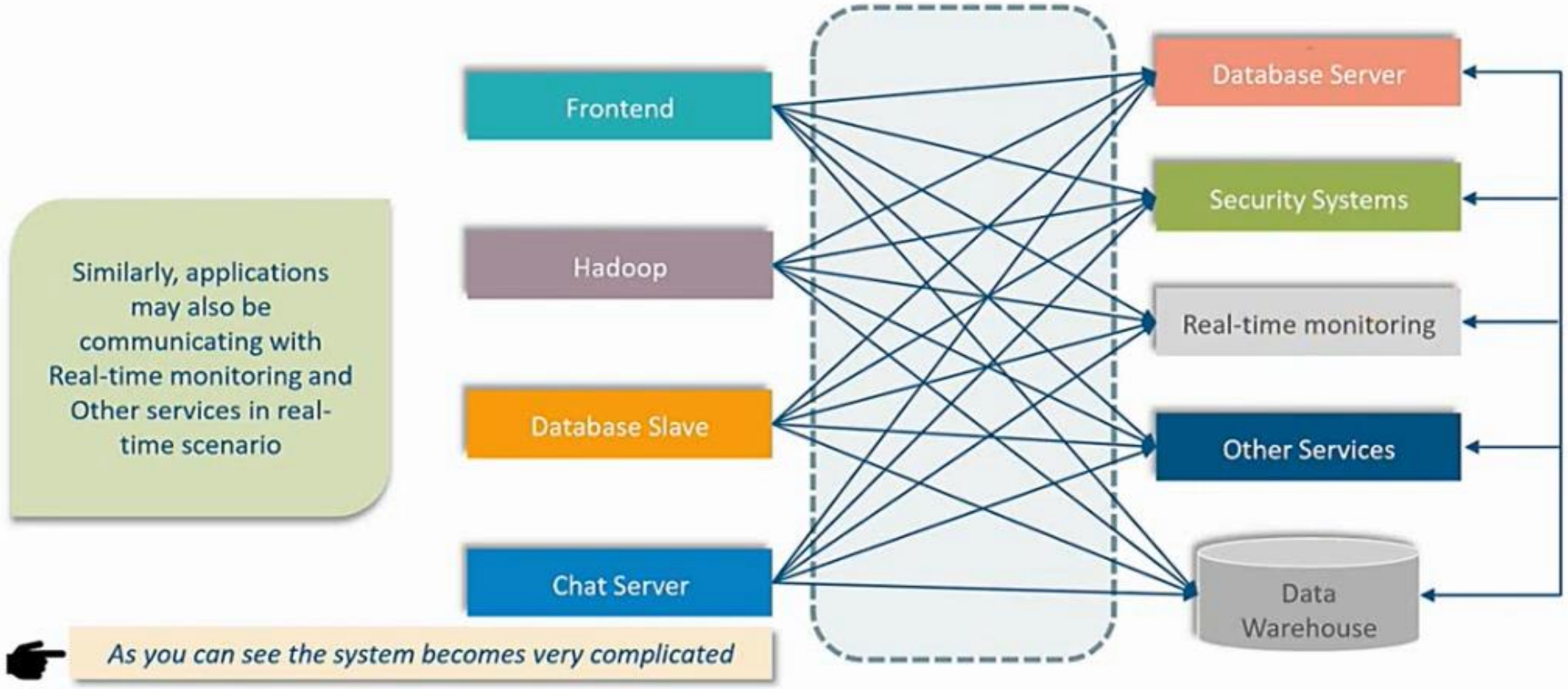
فرض کنید شما تعداد زیادی سیستم و زیرسیستم مختلف را داشته باشید که هر کدام از آنها نیازمند ارتباط با برخی از قسمتهای دیگر است. در این صورت شما دو راه دارید: اول اینکه در هر قسمت سرویس‌هایی را برای ارتباط با سایر قسمت‌ها پیاده سازی کنید یا اینکه هر قسمت بصورت مستقیم با سایر قسمتها در ارتباط باشد.

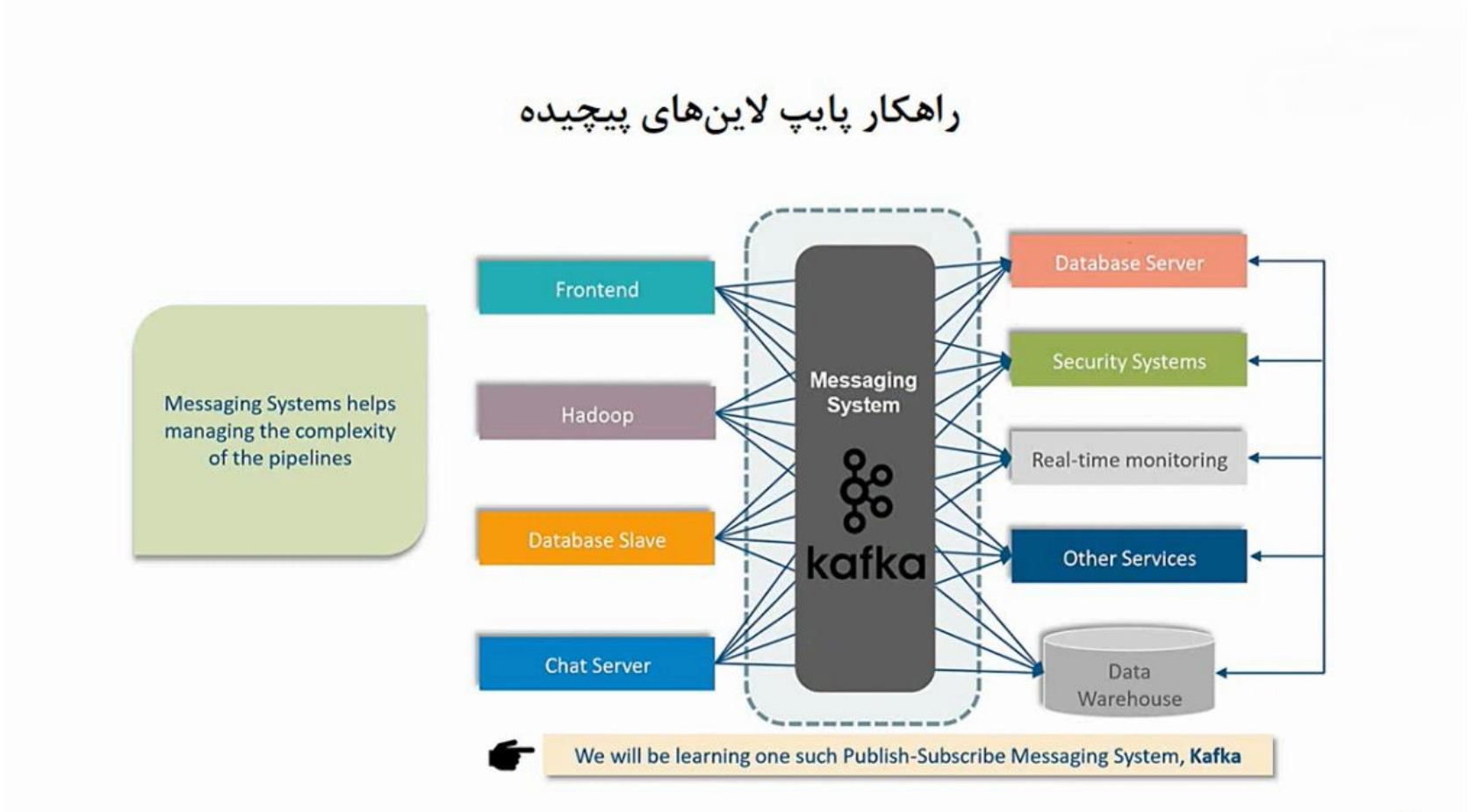


مشخصا کنترل و مدیریت جریان اطلاعاتی در این پیاده سازی کار بسیار دشواری است. تغییر هر قسمت، تاثیر مستقیمی بر روی سایر قسمتها دارد و در صورتی که هریک از قسمتها با مشکلی روبرو شوند، سایر قسمتهای مرتبط نیز با مشکل روبرو می‌شوند. این مشکل زمانی بسیار نمایان می‌شود که در معماری‌هایی مانند میکروسرویس، بدلیل بالا رفتن تعداد زیرسیستم‌ها و ارتباطات آنها، مدیریت این ارتباطات کار بسیار دشوار، پرهزینه و پیچیده‌ای می‌شود.

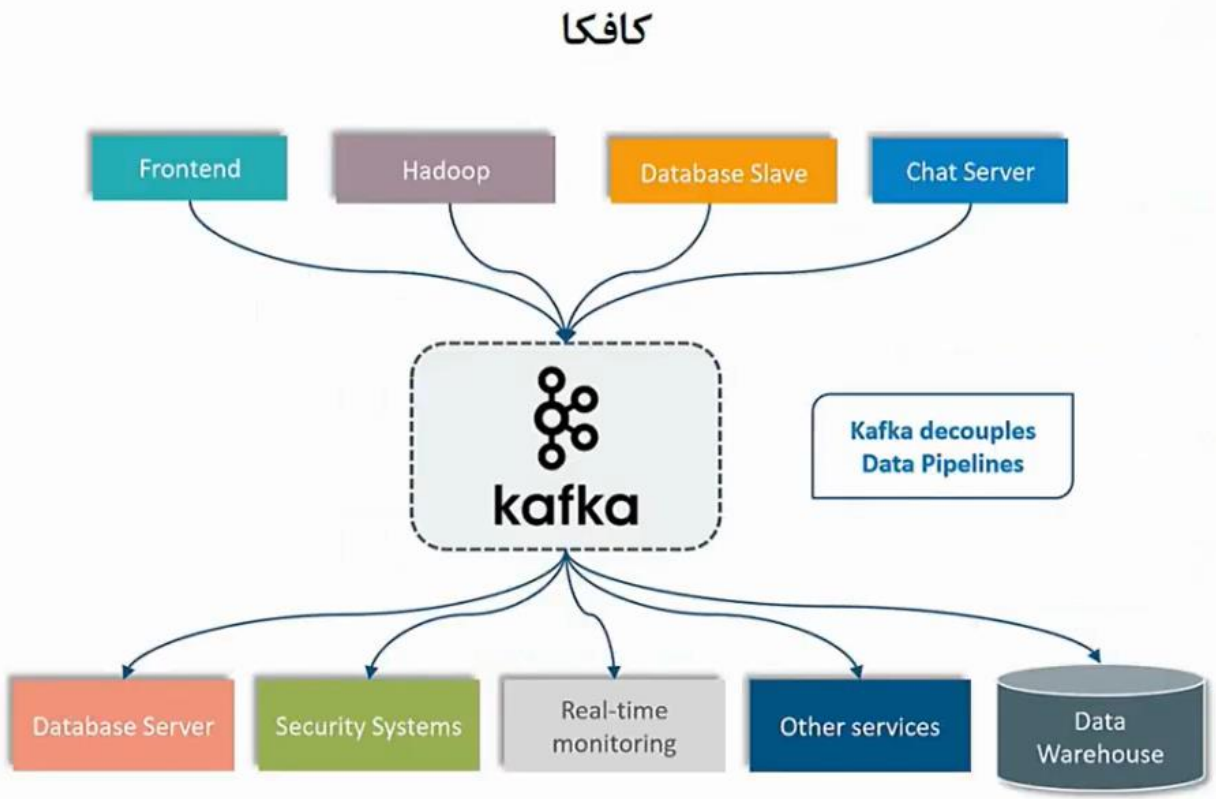


پایپ لاین‌های پیچیده





روش Apache Kafka برای رفع مشکل پایپ لاین های پیچیده به این صورت است که ارائه یک بستر برای مدیریت و کنترل جریان‌های اطلاعاتی با کارآیی بسیار بالا، در سیستم‌ها و زیرسیستمهای مختلف است. یعنی شما می‌توانید با ایجاد کردن یک Pipeline برای جریان اطلاعات خود، وابستگی مستقیم سیستمها و زیرسیستمها را از بین ببرید؛ آن هم به صورتی که بروز مشکلی در هر قسمت، کمترین میزان تاثیر را در سایر قسمتها داشته باشد.



همانطور که می‌بینید دیگر نیازی نیست تا قسمتهای مختلف بصورت مستقیم با یکدیگر در ارتباط باشند؛ تمامی ارتباطات از طریق Kafkaانجام می‌شود. تغییر یک قسمت، تاثیر زیادی بر روی سایر قسمتها ندارد یعنی دسترس خارج شدن یا بروز هر گونه مشکلی در یک قسمت، بر روی کل سیستم تاثیر زیادی ندارد. پیام‌های مربوط به یک قسمت تا زمانی که پردازش نشده‌اند از بین نمی‌روند؛ پس سیستم‌ها می‌توانند در حالت Offlineنیز به کار خود ادامه دهند. شما می‌توانید در این روش تمامی قسمت‌های سیستم را بصورت یک Clusterپیاده سازی کنید. در اینصورت احتمال از دسترس خارج شدن هر قسمت به کمترین میزان می‌رسد. حتی درصورتی که یک قسمت بصورت موقت از دسترس خارج شود، پیام‌های مرتبط با آن قسمت تا زمانی که دوباره به جریان پردازش بازگردد، از بین نمی‌روند بلکه پس از اضافه شدن قسمت از دسترس خارج شده، بلافاصله تمامی پیامهای مرتبط با آن قسمت برایش ارسال می‌شوند.

in eskazemi