

# Kafkaを使った マイクロサービス基盤 part2 +運用して起きたトラブル集

@matsu\_chara 2016/5/31

Apache Kafka Meetup Japan #1 at Yahoo! JAPAN

今日のスライド

[http://www.slideshare.net/matsu\\_chara/kafka-part2](http://www.slideshare.net/matsu_chara/kafka-part2)

part1のスライド

<http://xuwei-k.github.io/slides/kafka-matsuri/#1>

# Apache Kafka を使った マイクロサービス基盤

[2016/01/31 Scala Matsuri](#)



# 自己紹介



- @matsu\_chara
- Ponylang非公式エバンジェリスト活動
- Scala新卒研修用テキスト

# 話すこと

- Kafkaを使ったイベントハブについて
  - イベントハブとしてのKafka
  - 現在のシステム構成
  - Kafkaの設定
- Kafka運用時辛かった事例
  - TopicとPartition数増大による性能劣化
  - FullGC発生によるPublish失敗
  - Raidコントローラエラー発生事件

# 話すこと

- 利用用途の違いでKafkaのチューニングはどう変わるのか
- 運用・性能面で困ったことを共有

Kafkaを使ったEventHubについて

# よくあるKafkaの使われ方

- ユーザーアクティビティログ・メトリクスの集約

=> availability重視

- イベントハブ(受け取ったデータをロストしないことが最重要)

=> durabilityを重視



# よくあるKafkaの使われ方

- ユーザーアクティビティログ・メトリクスの集約

=> availability重視

- イベントハブ(受け取ったデータをロストしないことが最重要)

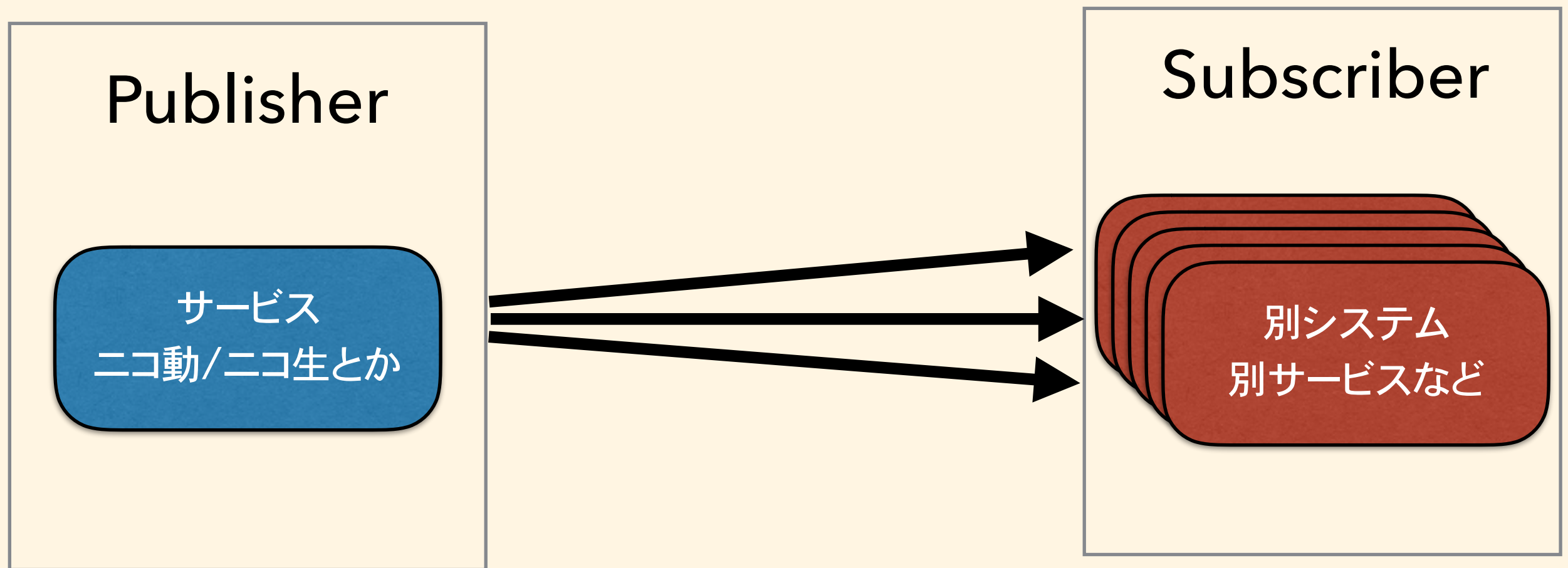
=> durabilityを重視

# イベントハブとしてのKafka

- 社内システム連携・メッセージングのための基盤

# イベントハブとしてのKafka

- 社内システム連携・メッセージングのための基盤

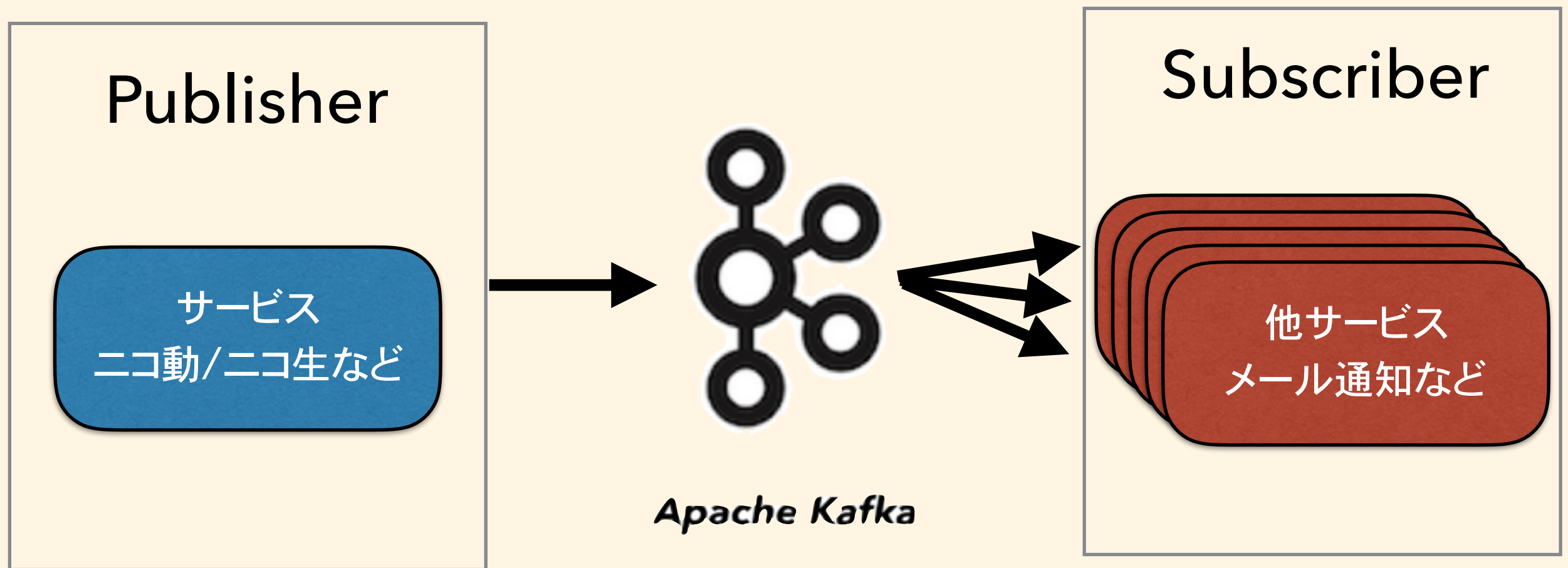


# イベントハブとしてのKafka

- Publisherが直接1:Nで配信するのは大変
  - 様々な温かみが生まれた歴史…
- 各種サービスから情報を集約したいチームが出てきた時に対応するコスト
- 性能を各サービスでスケールさせるコスト

# イベントハブとしてのKafka

- 社内システム連携・メッセージングのための基盤



# イベントハブとしてのKafka

- Kafkaを中心にしてデータを集約
- Kafkaのスケーラビリティにより、色々なサービスが情報をsubscribe可能になる
- publisherのシステムの都合にsubscriberが影響されない(密結合を防ぐ)

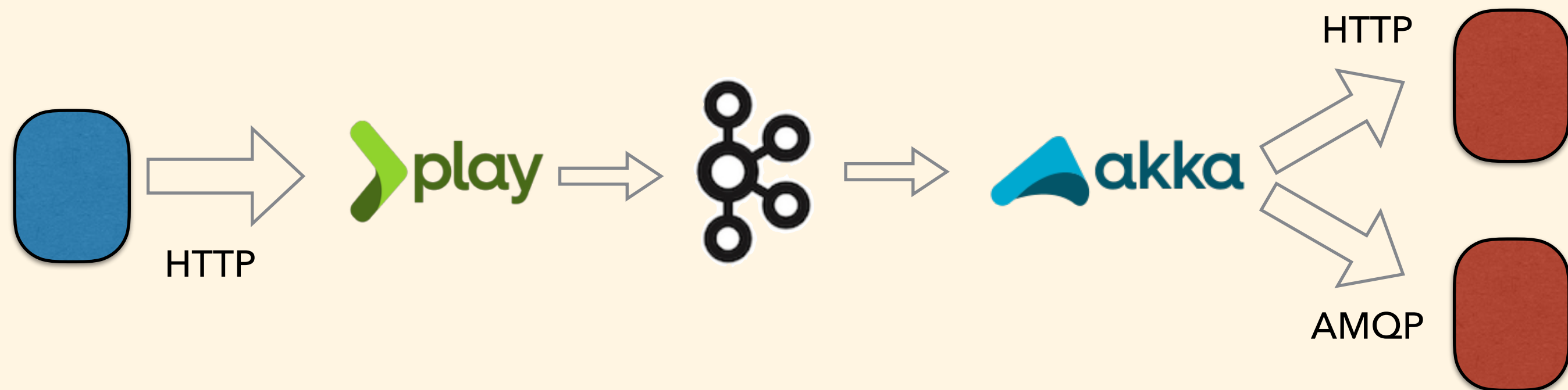
# 現在のシステム

- Scala/Play/akka
- 運用開始から半年ちょっと
- Kafka 0.9(クラスタは一つ。まだあまり大きくない)




# 現在のシステム

- HTTPでイベントを受け取りKafkaへpublish
- KafkaからsubscribeしHTTP/AMQPで通知





# Protocol Buffers on Kafka

- イベントのシリアライザは  **protobuf**  
Protocol Buffers
- 社内システム間連携の基盤として、**メッセージの互換性**を保障・調整する役割も担いたい
- 互換性維持のやりやすさを考慮して採用
- grpcも併せて社内のデータ交換形式の統一をしていきたい

# Kafkaの設定

- データを失わないことを重視
- Netflixの事例と方向性が異なる

項目名	default値	Netflix	設定値
acks	1	1	all
replication.factor	-	2	3
min.insync.replica	1	?	2

# Kafkaの設定

その他の設定はpart1で紹介。

もっとチューニングしたいけど機能追加の兼ね合いがあるので隙を見てやっていきたい

もっと詳細な情報

<http://xuwei-k.github.io/slides/kafka-matsuri/#34>

clouderaの資料

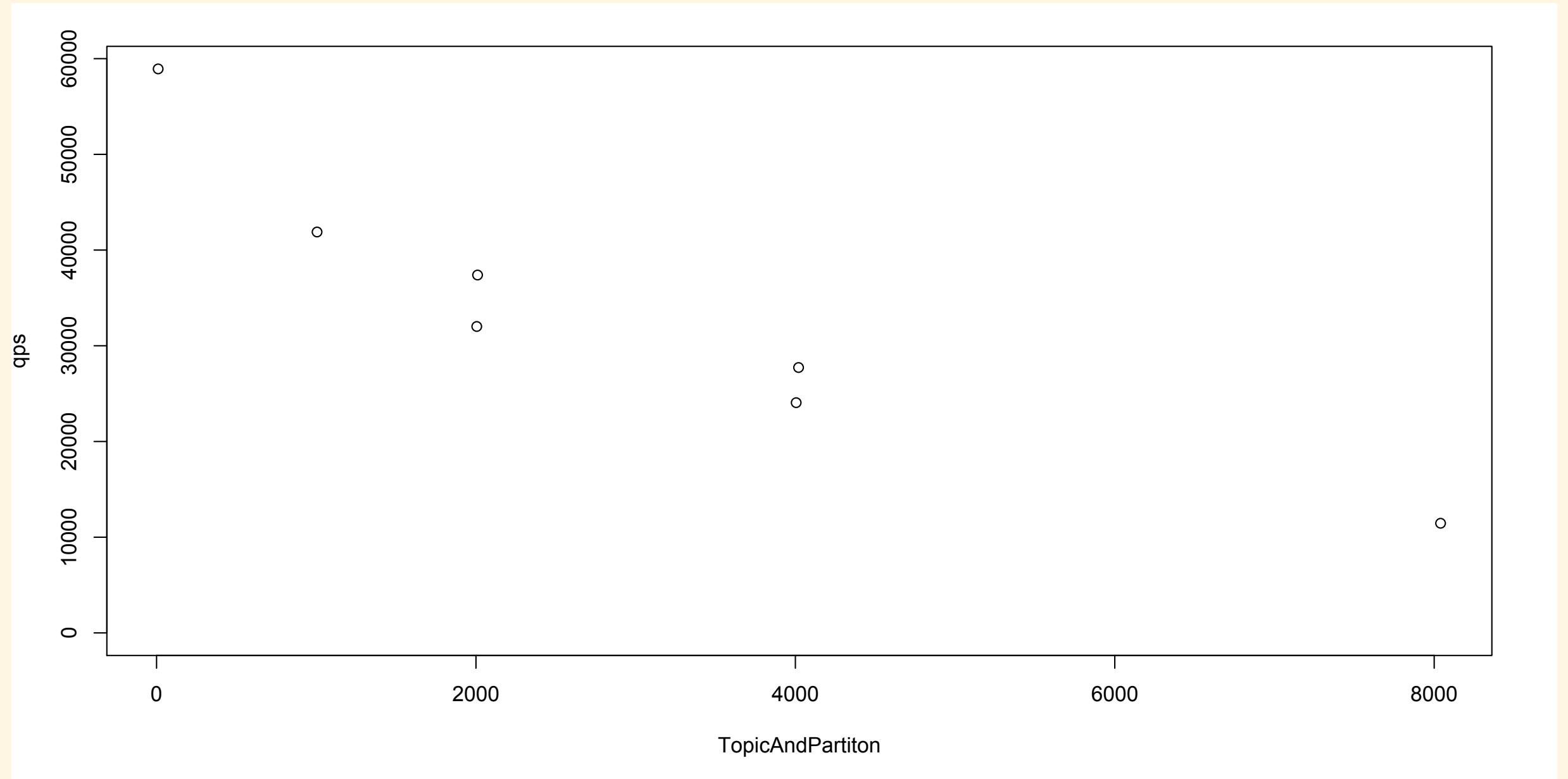
[http://www.cloudera.com/documentation/kafka/latest/topics/kafka\\_ha.html](http://www.cloudera.com/documentation/kafka/latest/topics/kafka_ha.html)

# Kafka運用辛かった事例

# TopicAndPartition増大による 性能劣化

- partitionが増えるとPublish完了までの時間が悪化
- replication factorにも依存
- レプリケーションが主な原因のようなので  
num.replica.fetchers などをチューニングする

# TopicAndPartition増大による 性能劣化



topicをたくさん作り、1 topicにのみ100万件publishしたときのqps

- グラフはHDDで計測したもの。SSDでも傾向自体は変化なし。

# TopicAndPartition増大による 性能劣化

- 現在はイベント頻度が高すぎないものに関しては partition数を1にして対処(必要に応じて増やす)
- partition数の目安は1 brokerあたり  
(100 \* broker台数 \* replication factor) 程度?  
(記事参照)

詳細

<http://www.confluent.io/blog/how-to-choose-the-number-of-topicspartitions-in-a-kafka-cluster/>

# TopicAndPartition増大による 性能劣化

- Netflixも抑えているが、そちらは可用性に関するチューニング？
- 故障時のオーバーヘッドを減らす

企業	目安	参考元
confluent	2000~4000 partitions/broker 10K~ partitions/cluster	<a href="http://www.confluent.io/blog/how-to-choose-the-number-of-topicspartitions-in-a-kafka-cluster/">http://www.confluent.io/blog/how-to-choose-the-number-of-topicspartitions-in-a-kafka-cluster/</a>
Netflix	200 broker/cluster 以下 10K partition/custer 以下	<a href="http://techblog.netflix.com/2016/04/kafka-inside-keystone-pipeline.html">http://techblog.netflix.com/2016/04/kafka-inside-keystone-pipeline.html</a>



# FullGC発生によるPublish失敗

- 負荷試験中に発生。
- メッセージサイズによる。(Kafka的には1KB程度が最も性能がでてGCにも優しいらしい)
- Javaパフォーマンスに書いてあるようなことをひたすらやっていく。

clouderaの資料

[http://www.cloudera.com/documentation/kafka/latest/topics/kafka\\_performance.html](http://www.cloudera.com/documentation/kafka/latest/topics/kafka_performance.html)

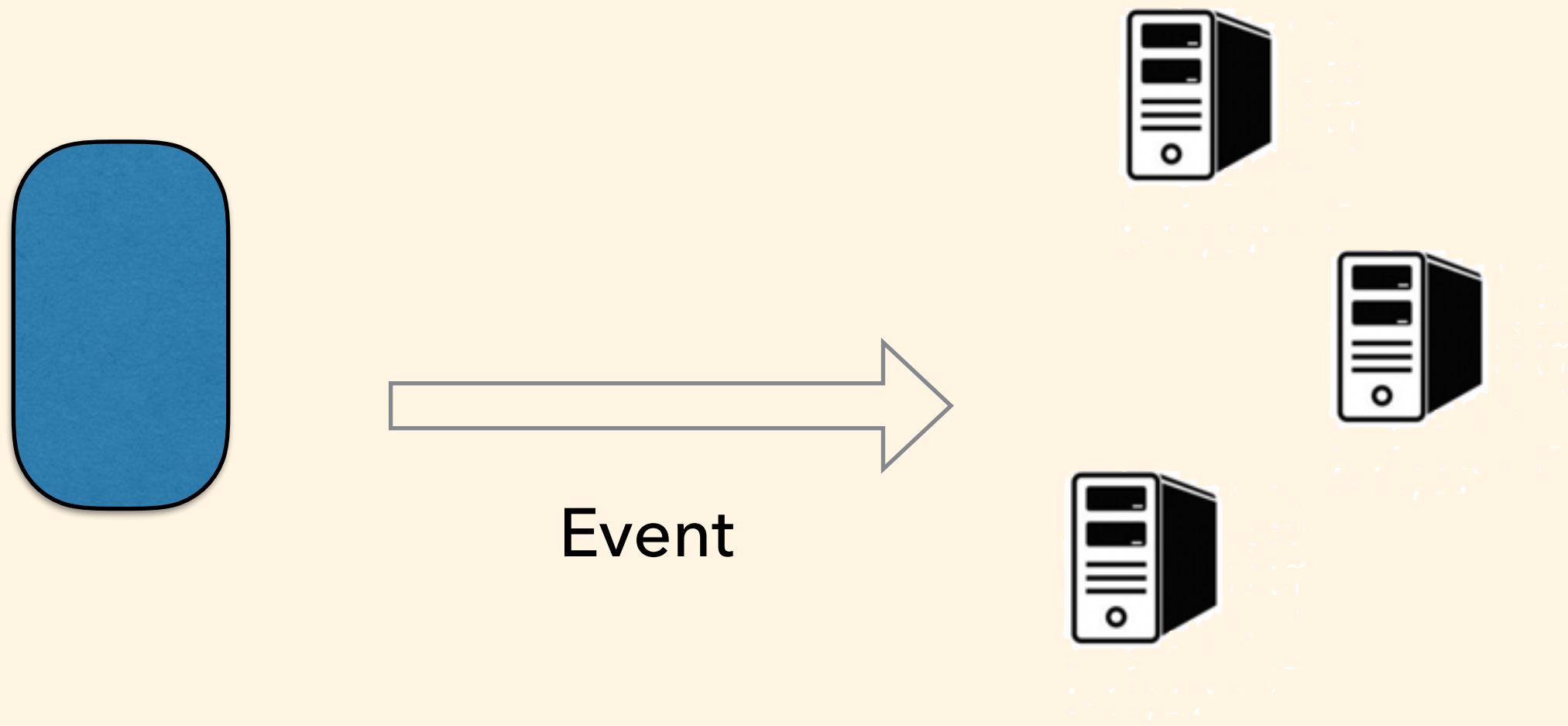
実際にやったチューニング

<http://xuwei-k.github.io/slides/kafka-matsuri/#61>

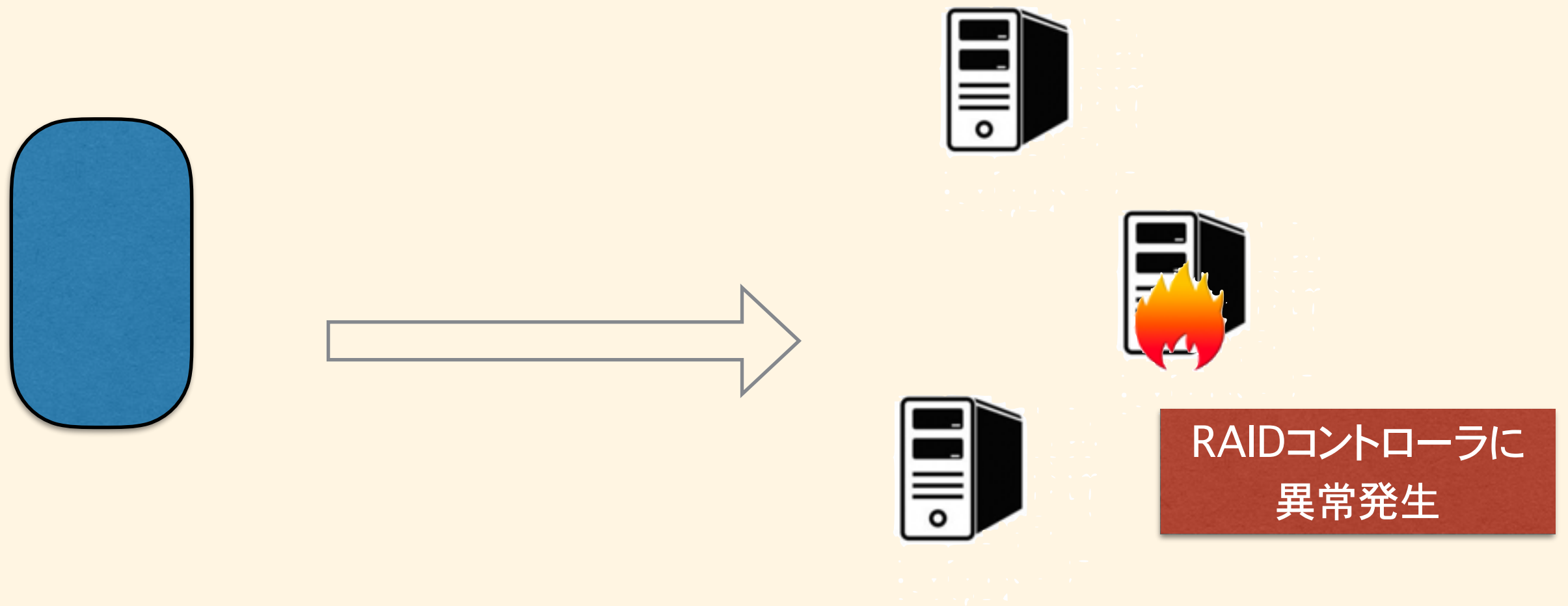
# RAIDコントローラエラー発生事件

- 突然Kafkaへのpublishがタイムアウトし始める
- ログを見るとRAIDコントローラが再起動していた
- RAIDコントローラ再起動後のbrokerは正常に動作
- 最近の出来事で調査・対策の方針がまだ立っていない

# RAIDコントローラエラー発生事件

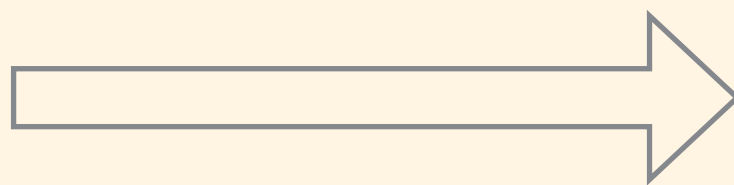


# RAIDコントローラエラー発生事件



# RAIDコントローラエラー発生事件

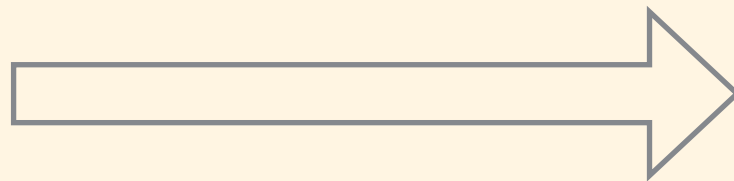
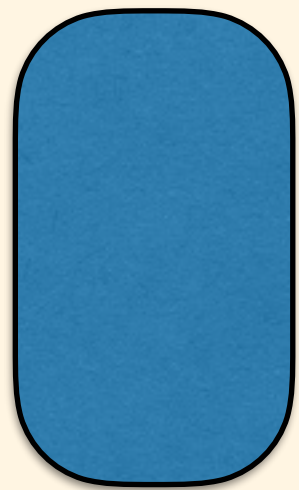
想定



RAIDコントローラに  
異常発生

# RAIDコントローラエラー発生事件

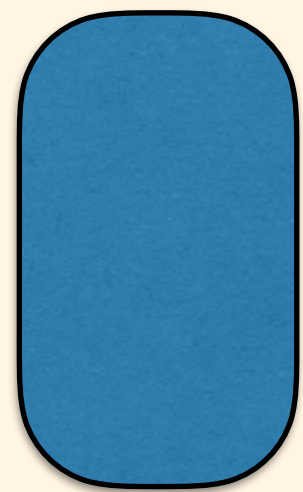
想定



in-sync replicaから離脱

# RAIDコントローラエラー発生事件

想定

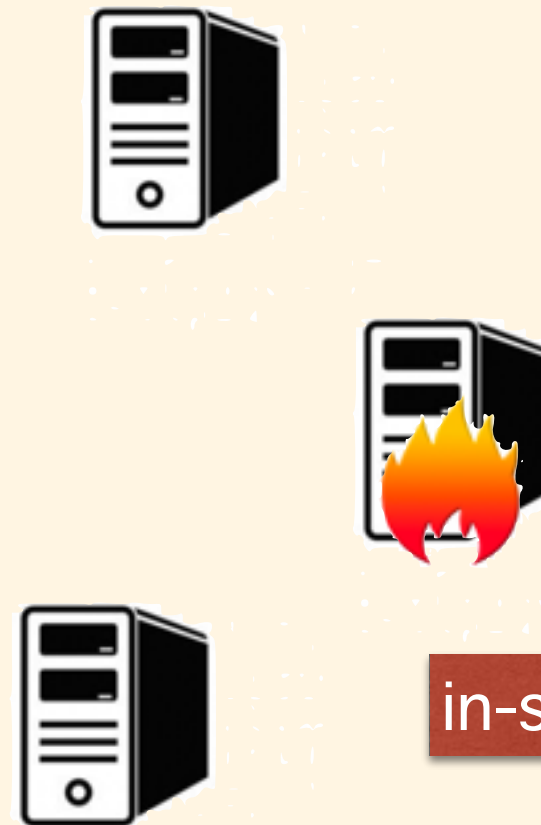
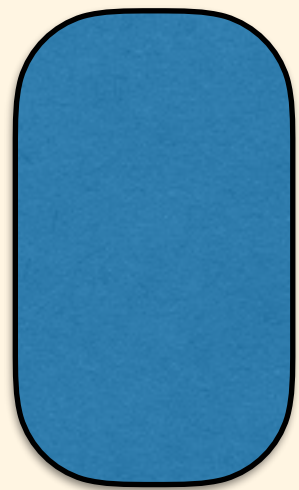


残った2台でack



# RAIDコントローラエラー発生事件

現実

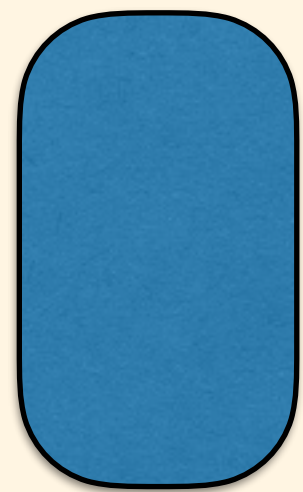


in-sync replicaのまま

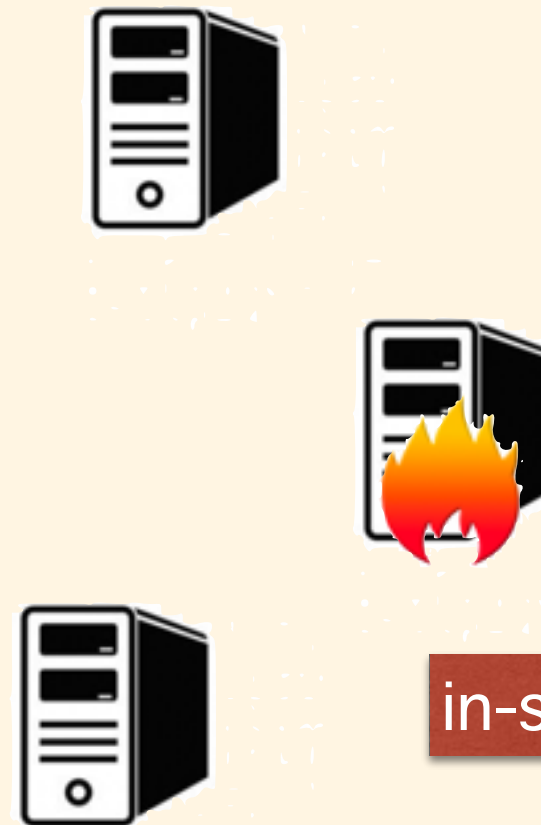
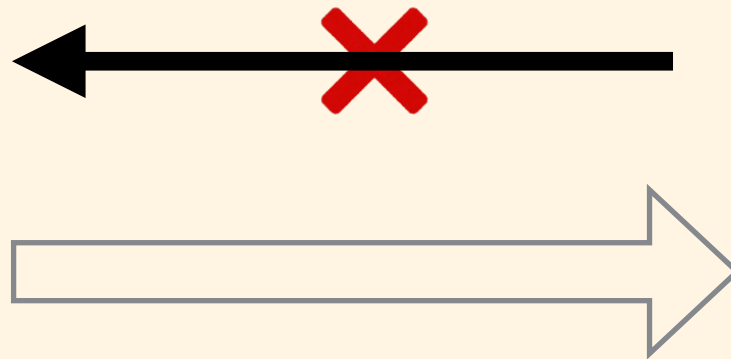


# RAIDコントローラエラー発生事件

現実



acks=allを待って  
タイムアウト



in-sync replicaのまま

# RAIDコントローラエラー発生事件

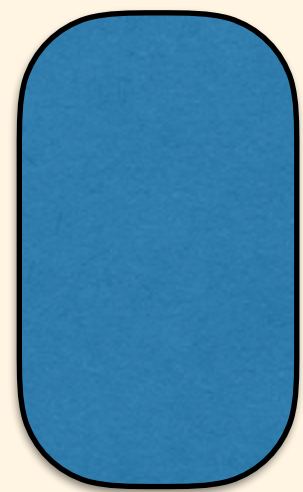
現実



しばらく経った後  
RAIDコントローラ  
再起動

# RAIDコントローラエラー発生事件

現実



3台でack



しばらく経った後  
RAIDコントローラ  
再起動

# RAIDコントローラエラー発生事件

- `min.insync.replica=2`なので1台落ちてもpublishできるという想定だった。
- しかし「**brokerがackを返せない状態**」で「**クラスタから離脱しなかった**」ため、「**acks=all**」の設定によりpublishできなかったと思われる
- brokerはzookeeperのハートビートには応答するが、ackは返せないという状態になりうる？

# RAIDコントローラエラー発生事件

- acks=2はkafka 0.9からは出来なくなっている
- RAIDを使わない方針も考えられる？
- RAID以外のエラーでも同じような現象は起きうるのか？
- 自動で離脱しないなら、brokerを停止させる外部機構が必要？

# RAIDコントローラエラー発生事件

- Netflixのようにcold standbyなクラスタを用意するのはどうなのか、調子の悪いbrokerを停止させるだけでは不十分？
- 再現できていないので仮説ベースな部分あり
- 意見募集

# まとめ

- 事例紹介
- 用途の違いを意識したチューニングが必要になる
  - Netflixのようなavailabilityを重視
  - イベントバスとしてdurabilityを重視
- 運用トラブルが起きる前に、confluent/linkedin/clouderaなどの資料は一通り目を通しておく後悔が少ない。
- 実際の運用時の環境を想定した負荷試験を試してみる