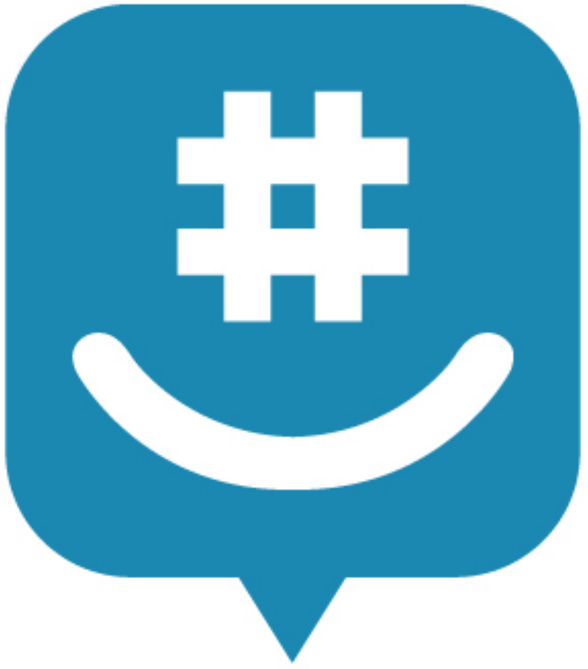
the great logfile in the sky

@jpignata

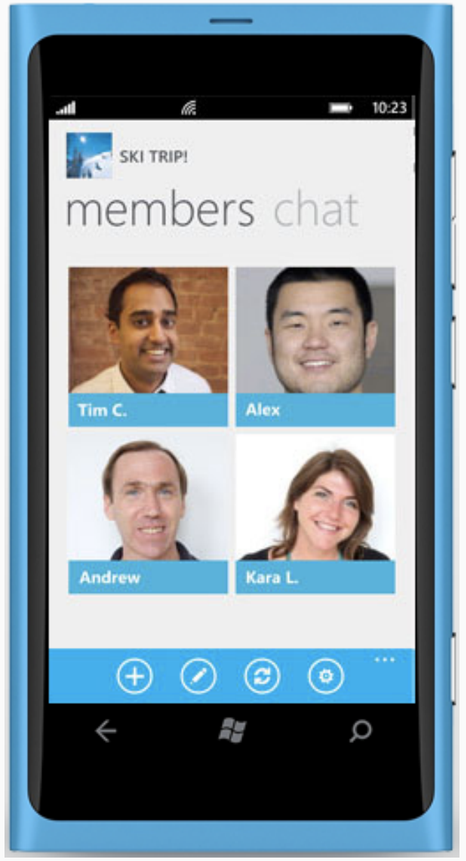

NYC



groupme


11:45 AM
Home
Brunch Club
Chat
Map
Members
Feb 6, 2012 1:06 PM
Brandon: Steak and eggs @ Le Barricou. Time to plan next week's destination.
Feb 6, 2012 2:30 PM
Kevin: How about Walter Foods? I hear it's good.
Tara: Walter Foods is amazing! Their chorizo scramble is my favorite!
Send


9:36
Friday Night Drinks
Chat
Map
Members
Chris: Guys, it's FRIDAY!!! Where should we go tonight?
Mon 4:15 PM
Jeremy: How about Veselka? I heard they have good specials.
Mon, 4:17 PM
Kevin: I'm down. I have been there before!
4:20 PM
Tara: Awesome! I get out at 7. See you guys there.
4:20 PM
Send!


10:23
SKI TRIP!
members chat
Tim C.
Alex
Andrew
Kara L.


C
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
0 +


Kafka

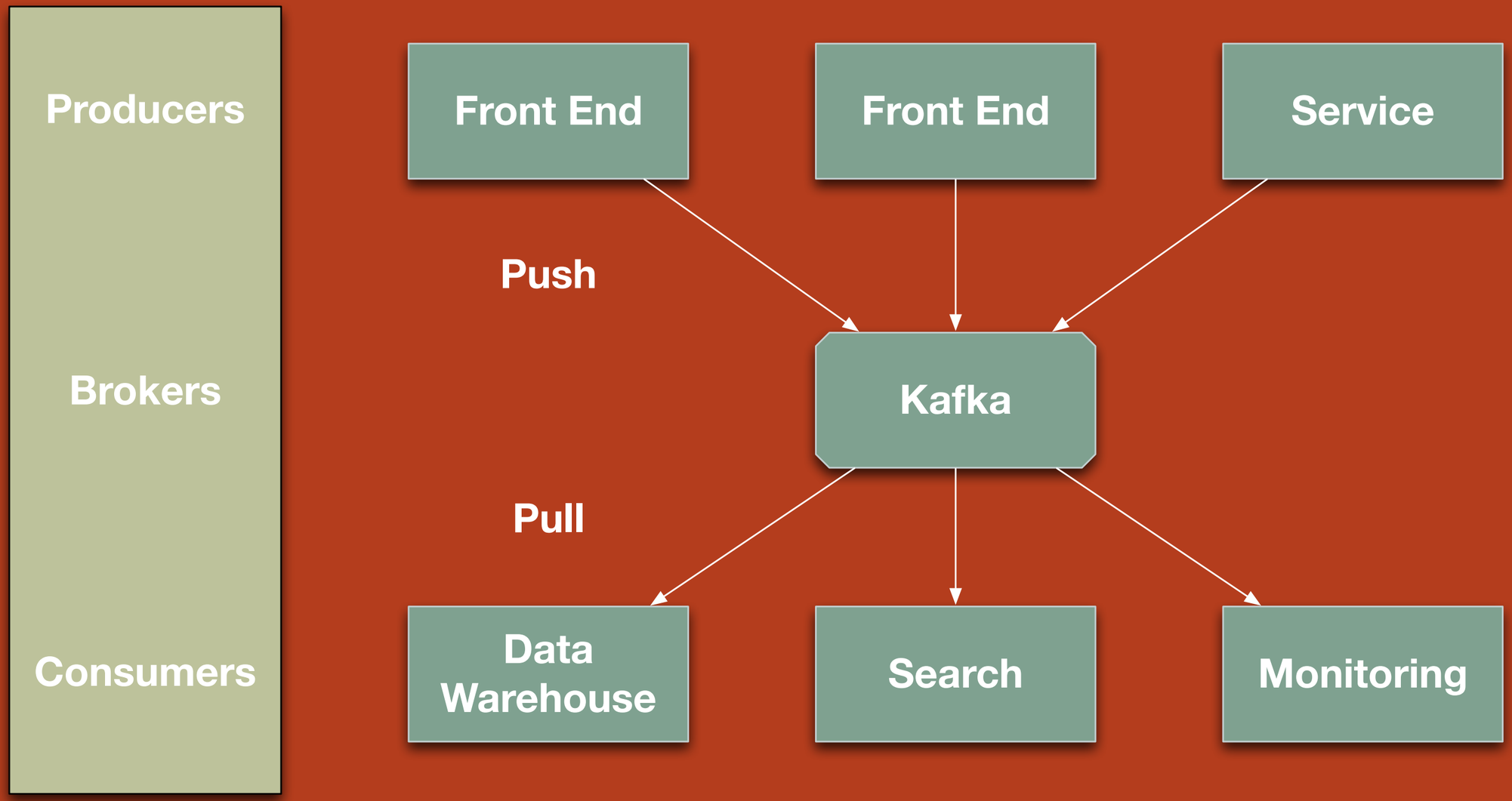
Kafka is a persistent publish/subscribe messaging system designed to broker high-throughput, data streams for multiple consumers.


LinkedIn

Kafka is a persistent, publish/subscribe

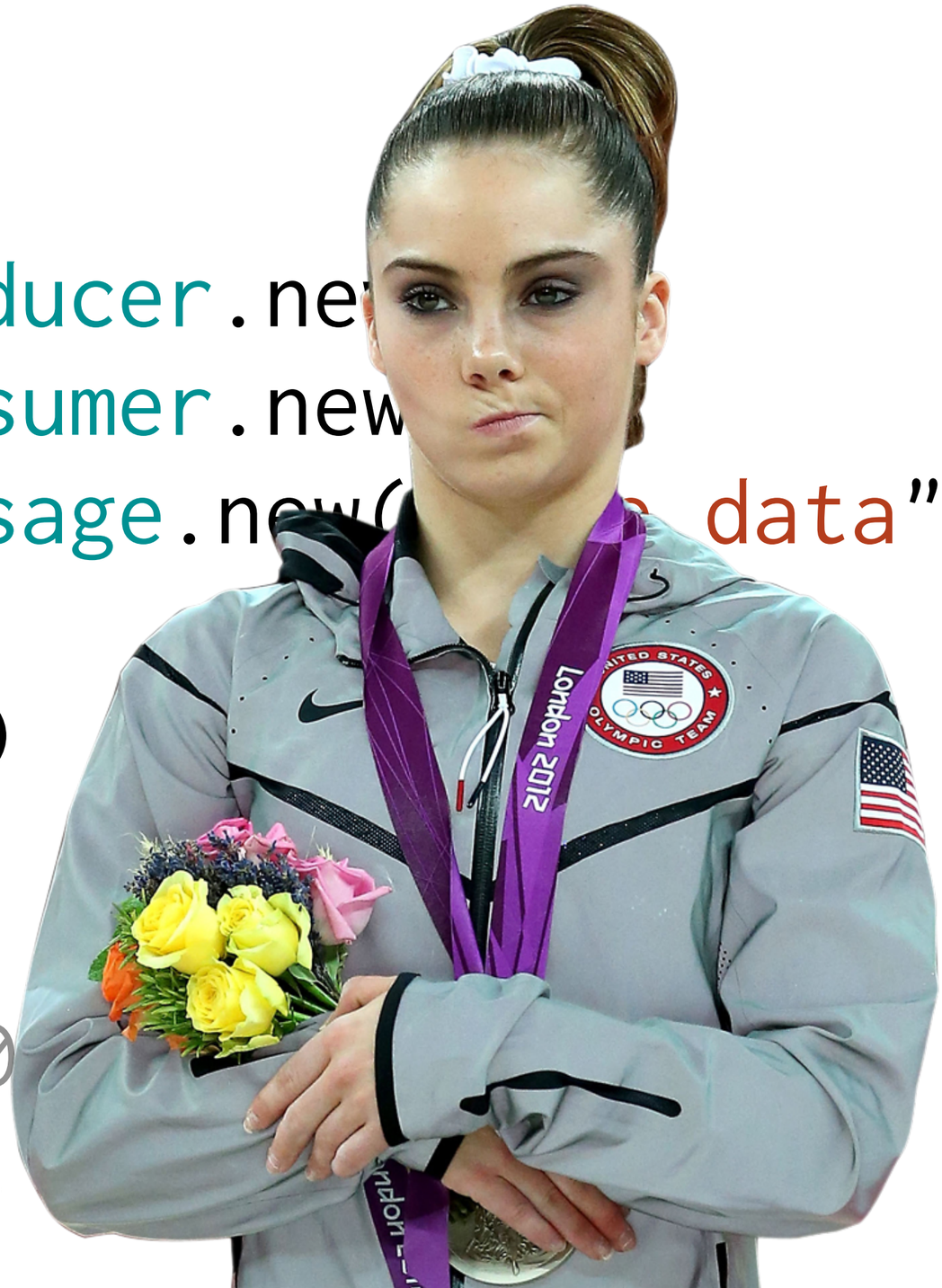
messaging system

designed to broker high-throughput data streams for multiple consumers.



```
require "kafka"

producer = Kafka::Producer.new
consumer = Kafka::Consumer.new
message  = Kafka::Message.new("Some data")

producer.send(message)
consumer.consume

=> [#<Kafka::Message:0x007fee51f83a80
@payload="Some data" ...>]
```

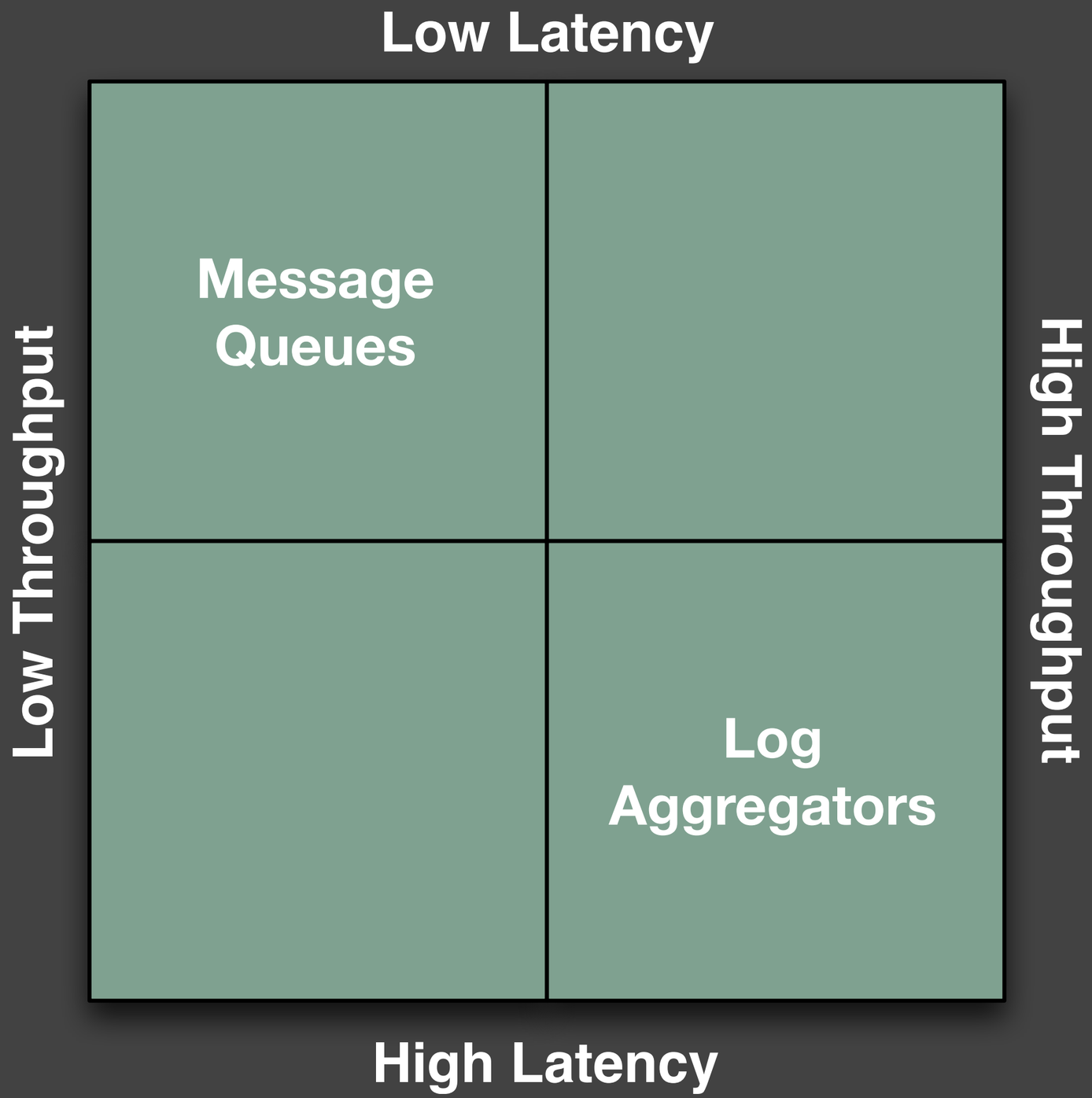
```
require "kafka"

producer = Kafka::Producer.ne
consumer = Kafka::Consumer.new
message  = Kafka::Message.new(      data")

producer.send(message)
consumer.consume

=> [#<Kafka::Message:0
@payload="Some data" .
```

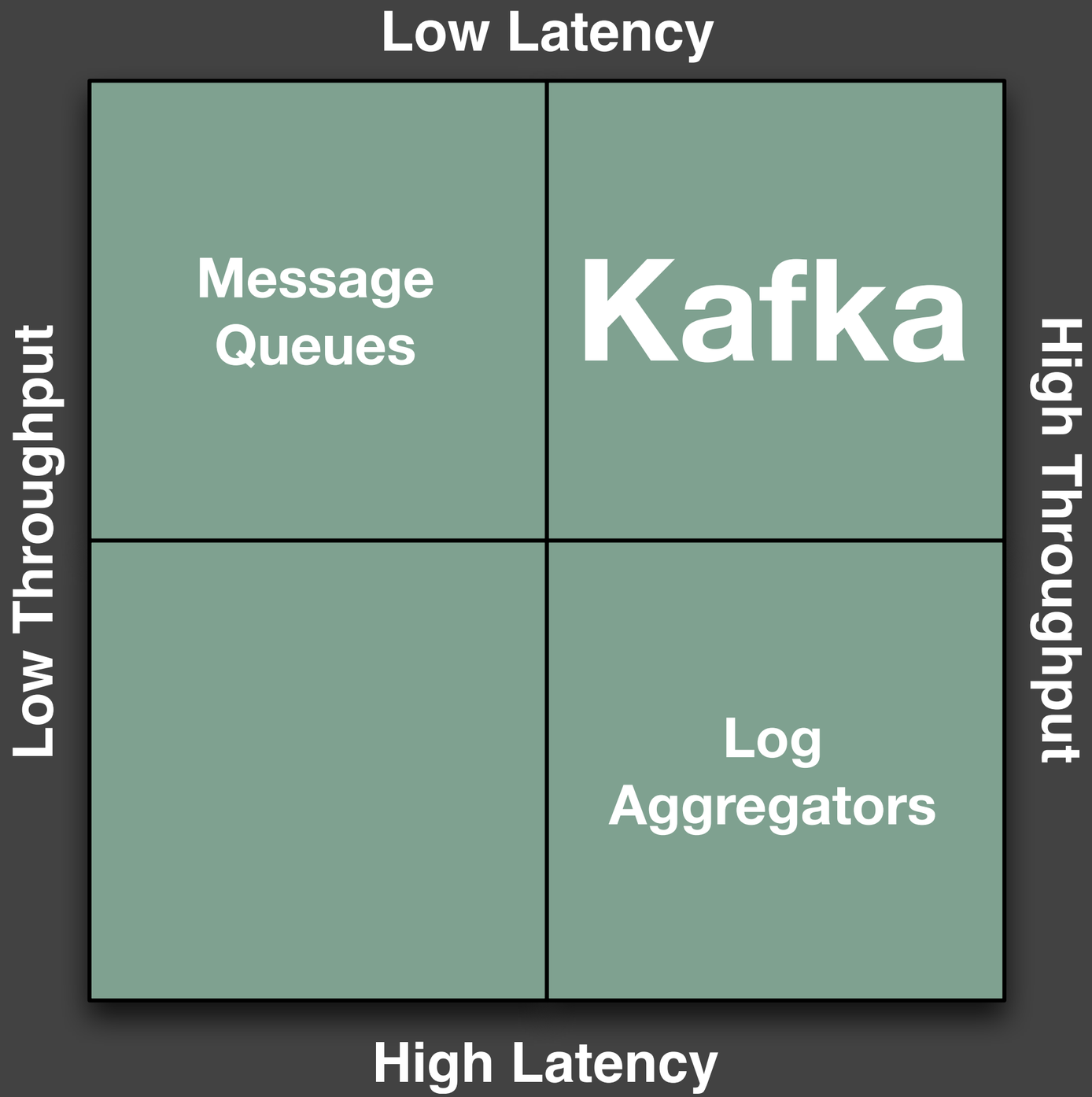


WHY?

Log Aggregators

Message Queues


Low Latency
Message Queues
Log Aggregators
Low Throughput
High Throughput
High Latency


Low Latency
Message Queues
Kafka
Low Throughput
High Throughput
Log Aggregators
High Latency

Apache Kafka is a

persistent,

publish/subscribe messaging system designed to broker high-throughput data streams for multiple consumers.

```
$ ls -l /opt/kafka/logs/page_views-0/
-rw-r--r-- 1 kafka kafka 536870926 Jul 25 21:17 00000000215822159191.kafka
-rw-r--r-- 1 kafka kafka 536870922 Jul 25 23:27 00000000216359030117.kafka
-rw-r--r-- 1 kafka kafka 536871053 Jul 26 01:38 00000000216895901039.kafka
-rw-r--r-- 1 kafka kafka 536871062 Jul 26 03:51 00000000217432772092.kafka
-rw-r--r-- 1 kafka kafka 536871084 Jul 26 06:09 00000000217969643154.kafka
-rw-r--r-- 1 kafka kafka 368959329 Jul 26 08:38 00000000218506514238.kafka

$ ls -l /opt/kafka/log/analytics-5/
-rw-r--r-- 1 kafka kafka 536871090 Jul 26 04:58 00000000032212266086.kafka
-rw-r--r-- 1 kafka kafka 536871130 Jul 26 22:00 00000000032749137176.kafka
-rw-r--r-- 1 kafka kafka 536870939 Jul 27 15:49 00000000033286008306.kafka
-rw-r--r-- 1 kafka kafka 536871063 Jul 28 01:28 00000000033822879245.kafka
-rw-r--r-- 1 kafka kafka 424050131 Jul 28 21:18 00000000034359750308.kafka
```

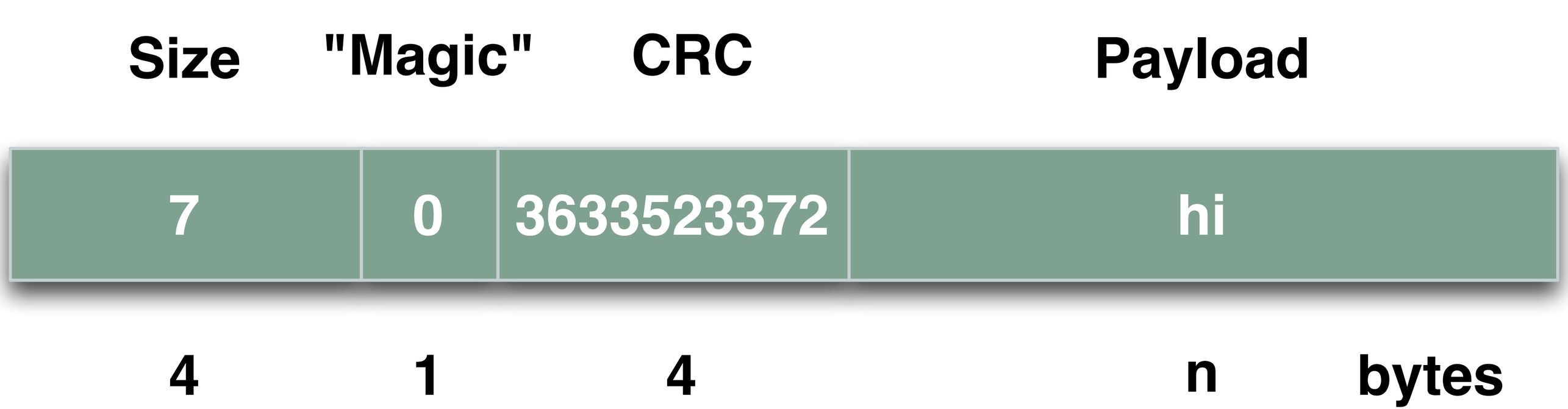
TOPIC

page_views
ad_clicks
service_logs

PARTITION

0..n

```
producer.send(Message.new("hi"))
```



```
%w(hi hi hi hello goodday hi).each do |payload|
  producer.send(Message.new(payload))
end
```

0 hi	11 hi	22 hi
33 hello	47 goodday	63 hi

```
$ ls -l /opt/kafka/logs/page_views-0/
-rw-r--r-- 1 kafka kafka 536870926 Jul 25 21:17 00000000215822159191.kafka
```

```
{
   topic:     page_views,
   partition: 0,
   offset:    215822159191
}
```

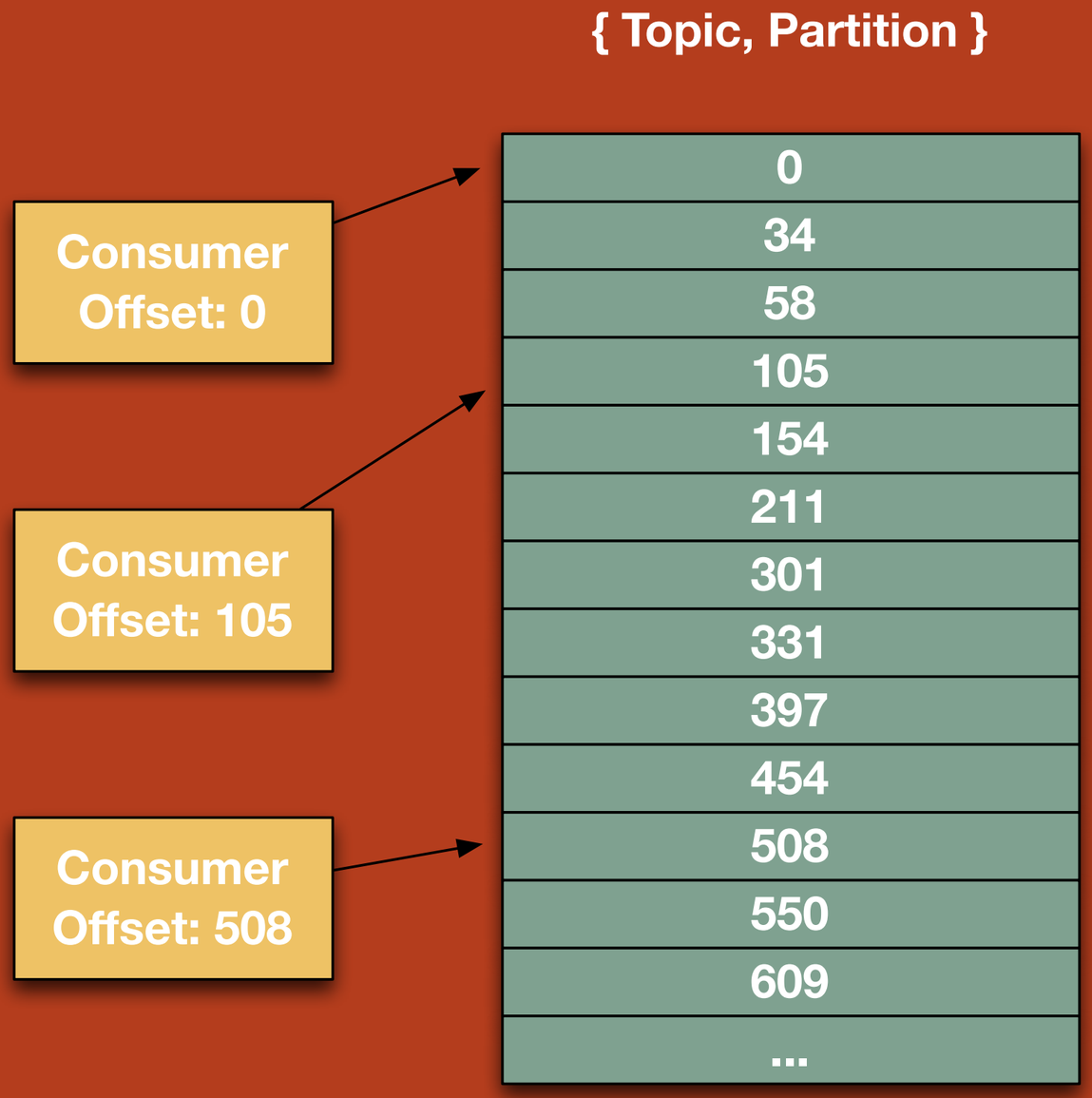
```
producer = Kafka::Producer.new(
  topic:     "letters",
  partition: 0
)

%w(a b c d e).each do |letter|
  message = Kafka::Message.new(letter)
  producer.send(message)
end
```

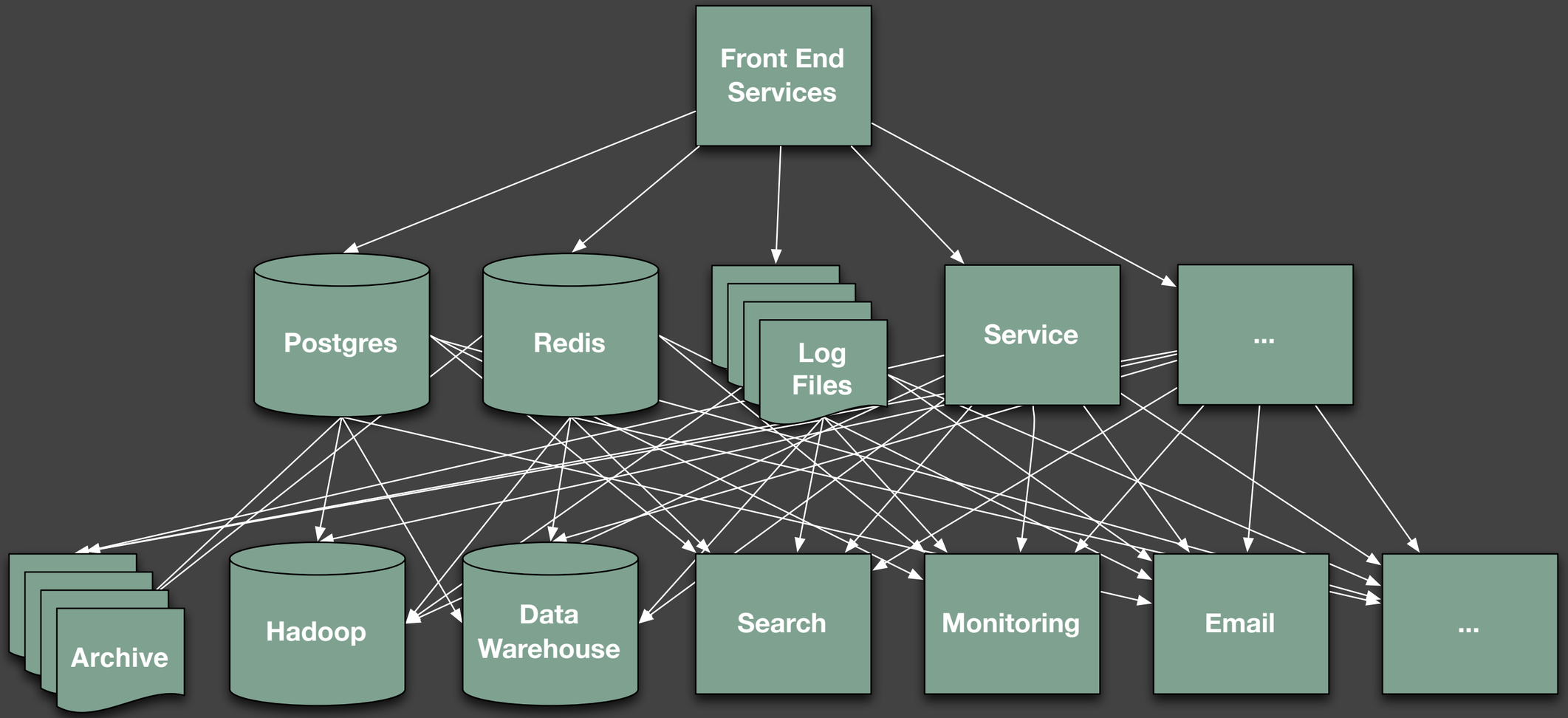
```
consumer = Kafka::Consumer.new(
  offset:    10,
  topic:     “letters”,
  partition: 0
)

consumer.offset
=> 10

consumer.consume.map(&:payload)
=> [“b”, “c”, “d”, “e”]

consumer.offset
=> 50
```

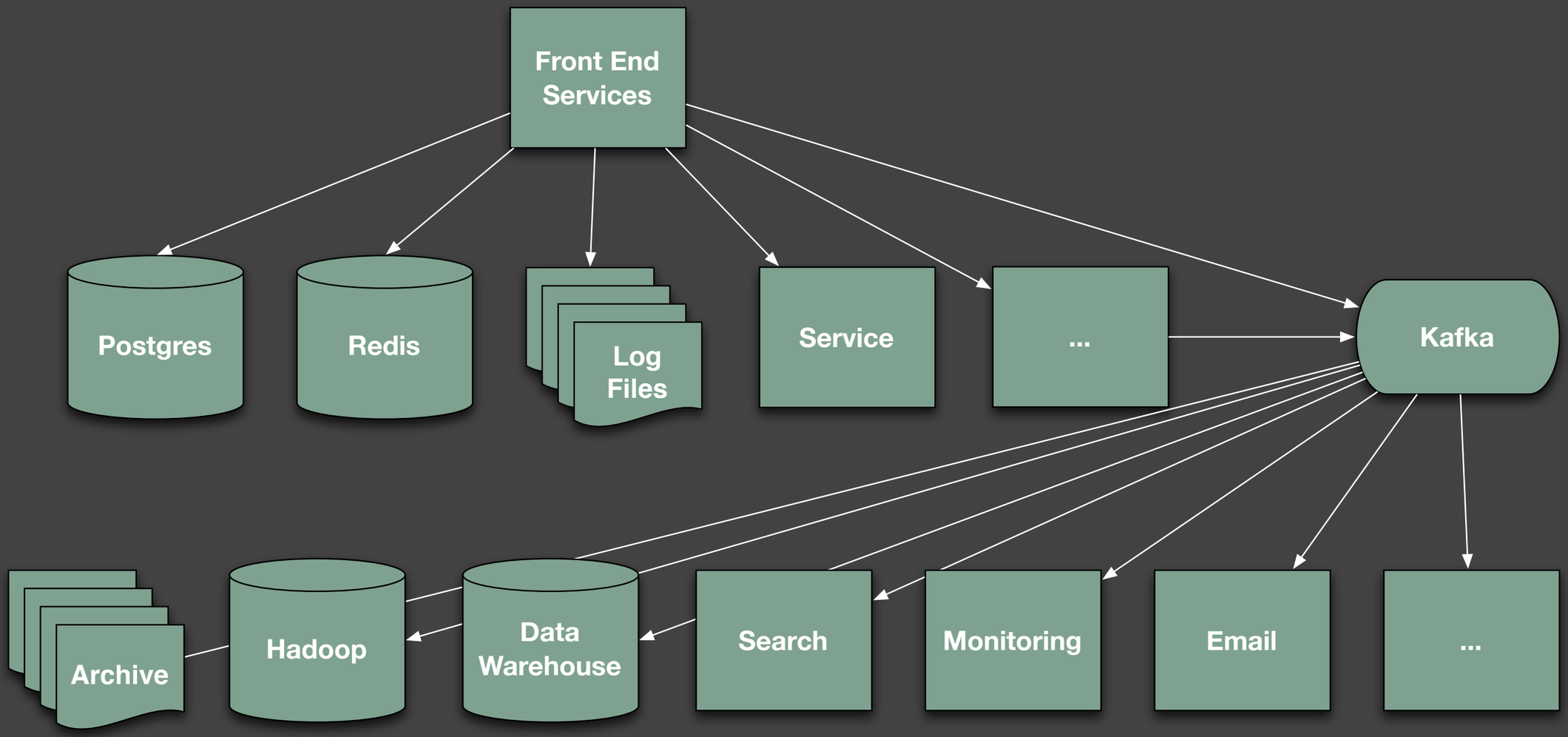
```
$ ls -l /opt/kafka/logs/page_views-0/
-rw-r--r-- 1 kafka kafka 536870926 Jul 25 21:17 00000000215822159191.kafka
-rw-r--r-- 1 kafka kafka 536870922 Jul 25 23:27 00000000216359030117.kafka
-rw-r--r-- 1 kafka kafka 536871053 Jul 26 01:38 00000000216895901039.kafka
-rw-r--r-- 1 kafka kafka 536871062 Jul 26 03:51 00000000217432772092.kafka
-rw-r--r-- 1 kafka kafka 536871084 Jul 26 06:09 00000000217969643154.kafka
-rw-r--r-- 1 kafka kafka 368959329 Jul 26 08:38 00000000218506514238.kafka

$ ls -l /opt/kafka/log/analytics-5/
-rw-r--r-- 1 kafka kafka 536871090 Jul 26 04:58 00000000032212266086.kafka
-rw-r--r-- 1 kafka kafka 536871130 Jul 26 22:00 00000000032749137176.kafka
-rw-r--r-- 1 kafka kafka 536870939 Jul 27 15:49 00000000033286008306.kafka
-rw-r--r-- 1 kafka kafka 536871063 Jul 28 01:28 00000000033822879245.kafka
-rw-r--r-- 1 kafka kafka 424050131 Jul 28 21:18 00000000034359750308.kafka
```

Kafka is a persistent, publish/subscribe messaging system designed to broker high-throughput data streams for

multiple consumers.



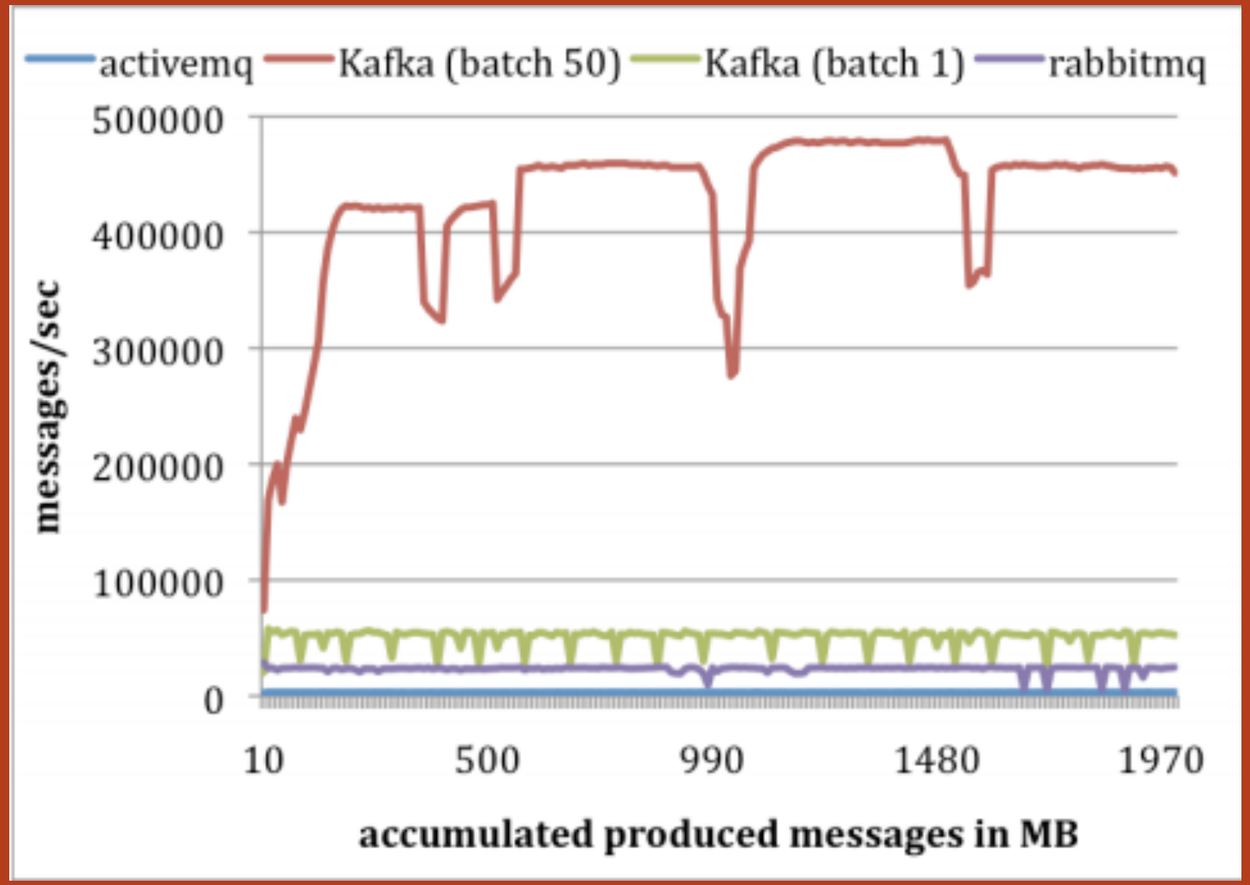

Front End Services
Postgres
Redis
Log Files
Service
...
Archive
Hadoop
Data Warehouse
Search
Monitoring
Email
...

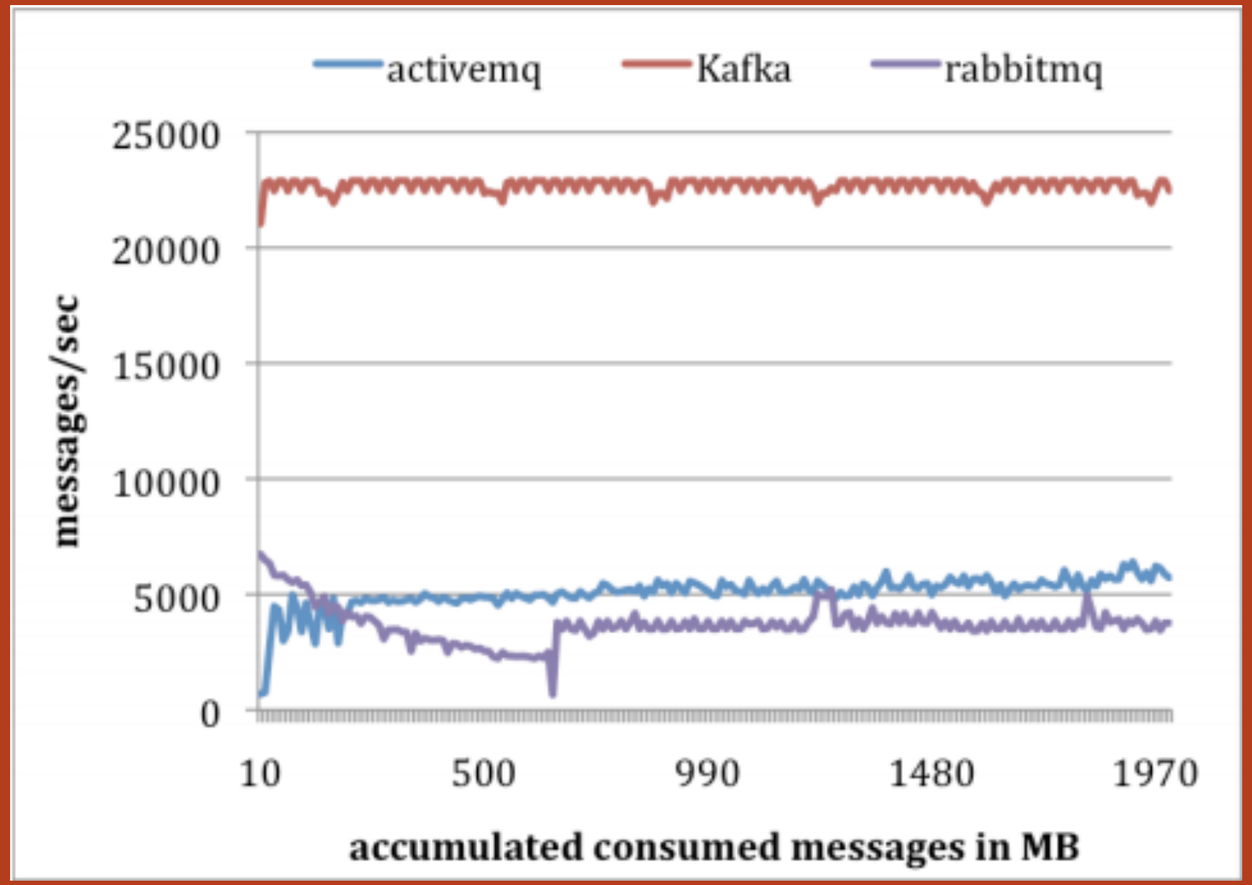

Front End Services
Postgres
Redis
Log Files
Service
...
Kafka
Archive
Hadoop
Data Warehouse
Search
Monitoring
Email
...

Kafka is a persistent, publish/subscribe messaging system designed to broker

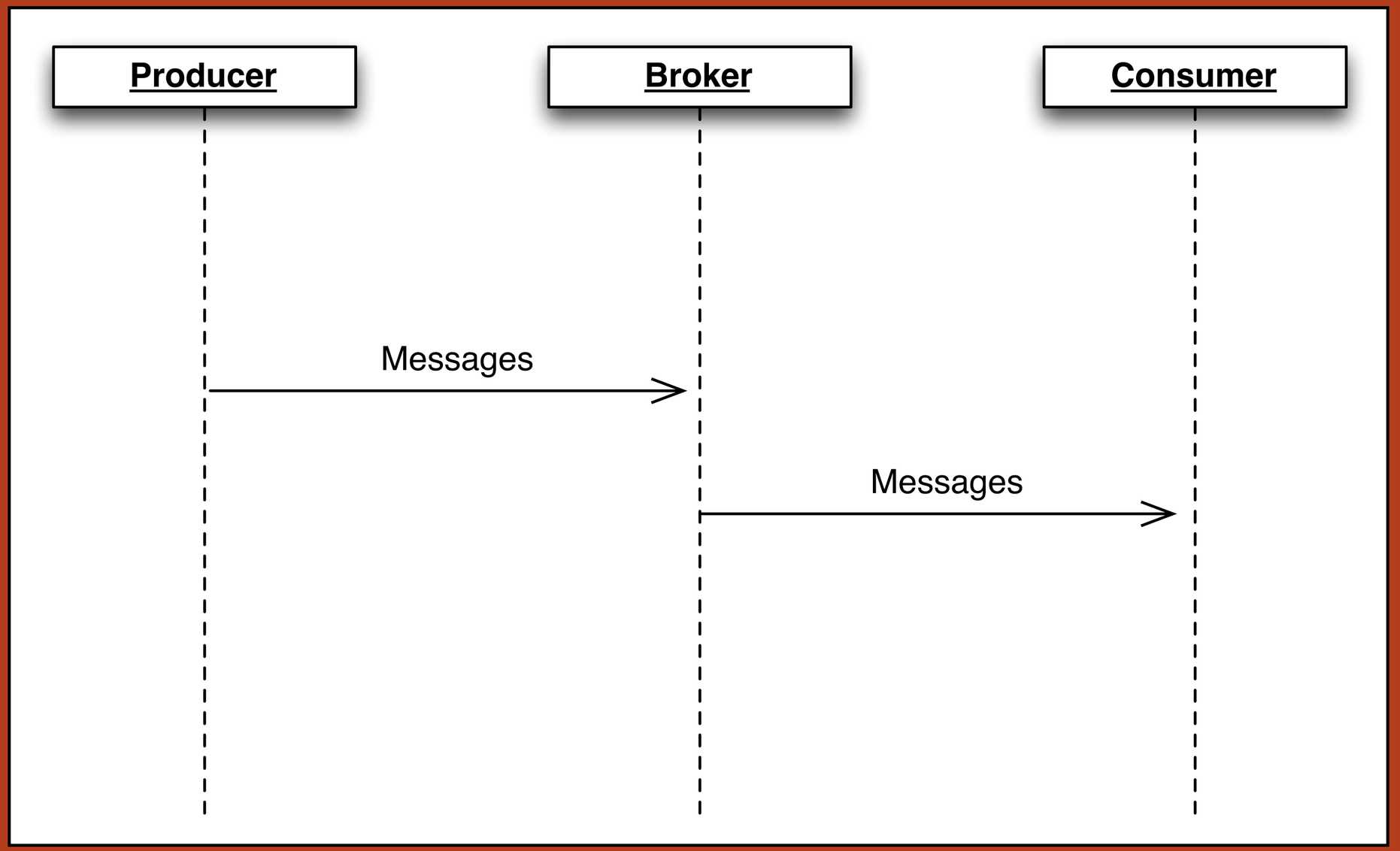
high-throughput,

data streams for multiple consumers.

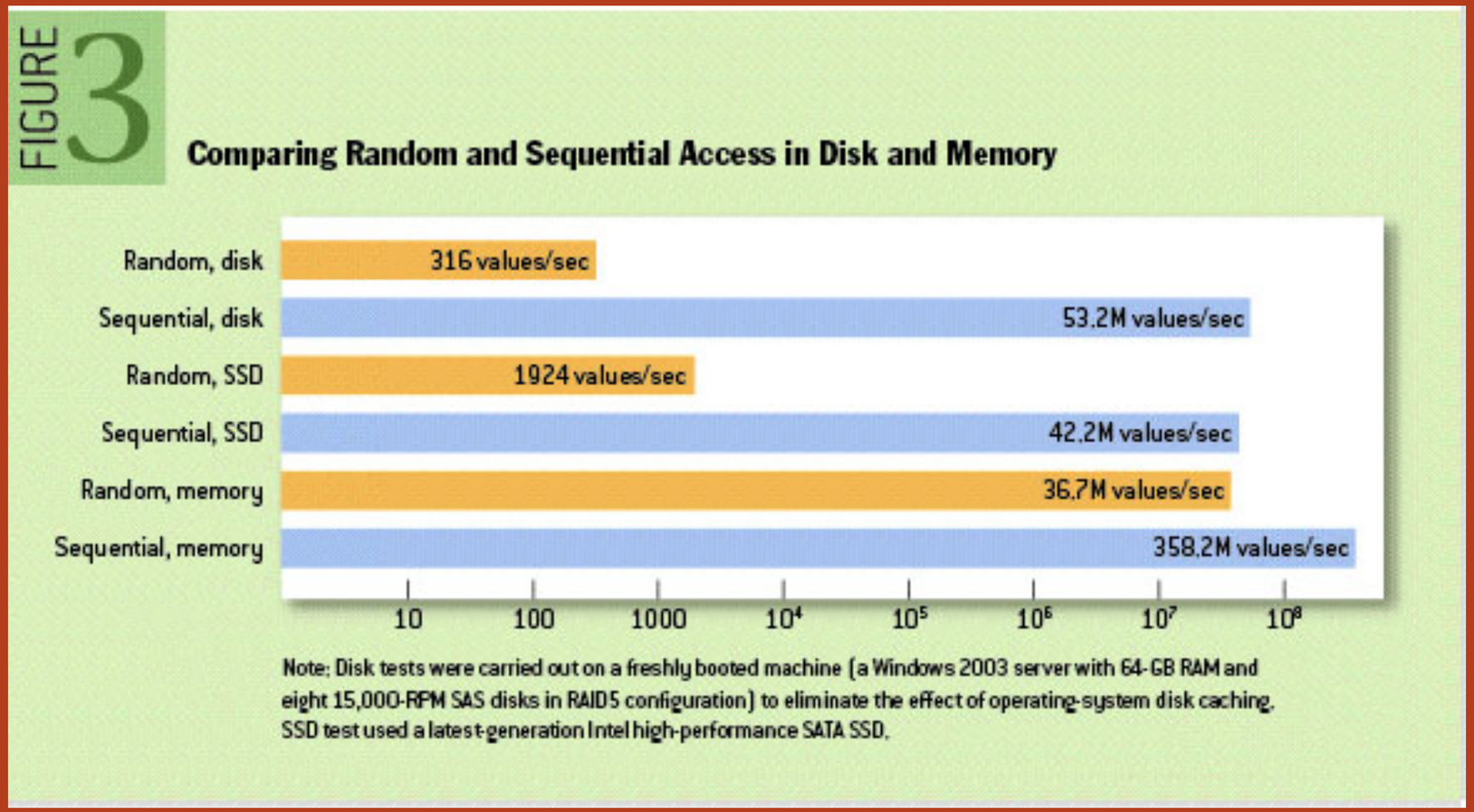




API Simplicity



Linear Disk Access



"[it's] widely underappreciated: in modern systems, as demonstrated in the figure, random access to memory is typically slower than sequential access to disk. Note that random reads from disk are more than 150,000 times slower than sequential access"

Adam Jacobs "The Pathologies of Big Data."
ACM Queue, July 2009

Page Cache

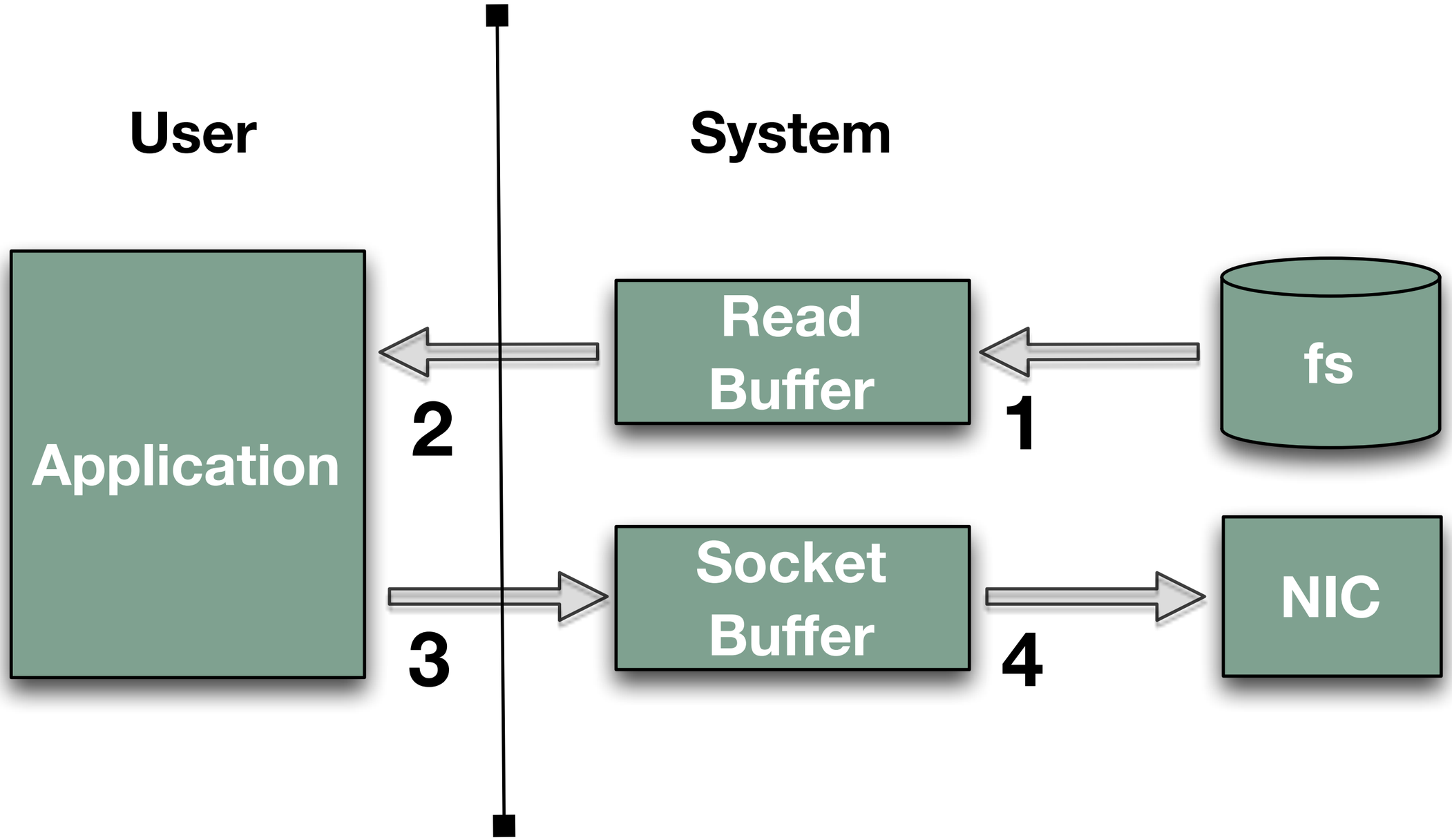
```
$ free
             total       used       free     shared    buffers     cached
Mem:          7450       7296        154          0        150       4916
-/+ buffers/cache:       2229       5220
Swap:            0          0          0
```

Write Behind

Read Ahead

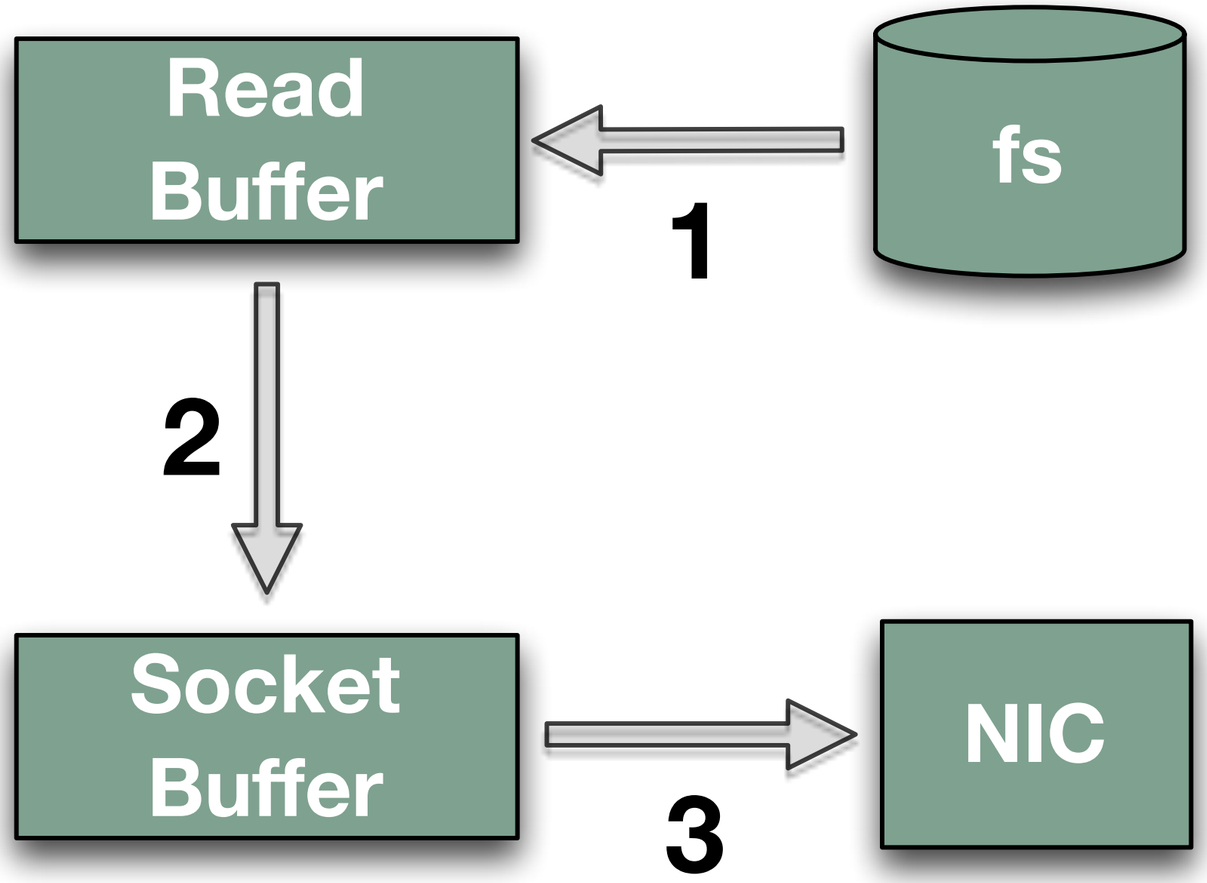
sendfile(2)

```
pread(file, buffer, size, offset);
// do something with the buffer
write(socket, buffer, size);
```


User
System
Application
Read Buffer
fs
2
1
Socket Buffer
NIC
3
4

```
sendfile(socket, file, offset, size);
```

System



Durability Concessions

Stateless Broker

Simple Schema-less Log Format

RECAP

Kafka is a persistent, publish/subscribe messaging system designed to broker high-throughput, data streams for multiple consumers.

Messaging System

- Cherry pick characteristics of log aggregation systems (performance) and message queues (semantics)

Persistent

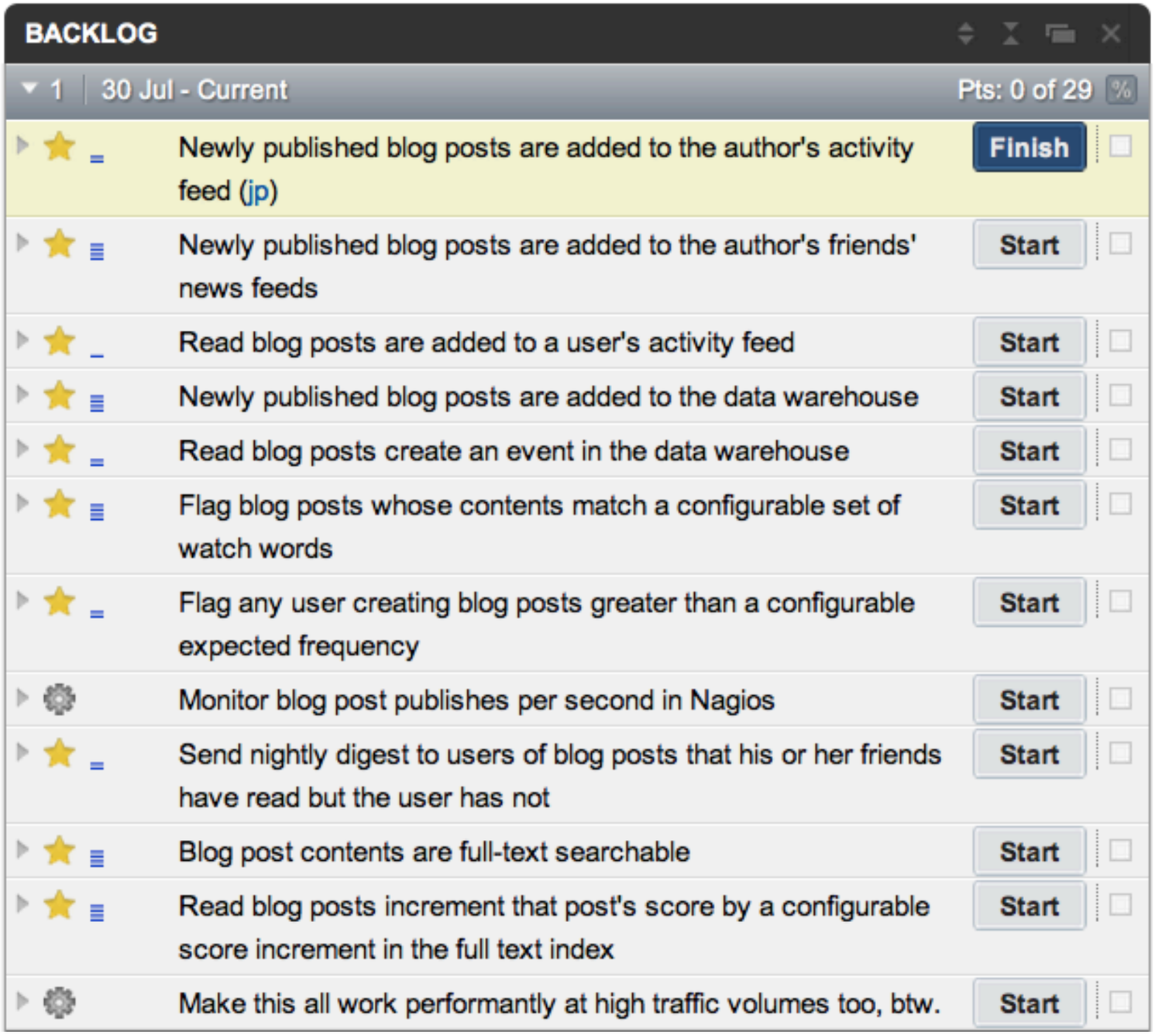
- Maintain a rolling time-based window of the stream
- Don’t fear the filesystem

High-Throughput

- Performance over features and durability
- Rely on operating system features
- Eschew user-land caching

Multiple Consumers

- Push data in, pull data out
- Support parallel consumers with varying rates from offline to realtime


BACKLOG
1
30 Jul - Current
Pts: 0 of 29
Newly published blog posts are added to the author's activity feed (jp)
Finish
Newly published blog posts are added to the author's friends' news feeds
Start
Read blog posts are added to a user's activity feed
Start
Newly published blog posts are added to the data warehouse
Start
Read blog posts create an event in the data warehouse
Start
Flag blog posts whose contents match a configurable set of watch words
Start
Flag any user creating blog posts greater than a configurable expected frequency
Start
Monitor blog post publishes per second in Nagios
Start
Send nightly digest to users of blog posts that his or her friends have read but the user has not
Start
Blog post contents are full-text searchable
Start
Read blog posts increment that post's score by a configurable score increment in the full text index
Start
Make this all work performantly at high traffic volumes too, btw.
Start

Publishing content to feeds based upon events

Data warehouse ETL of event data

Spam flagging of user-generated content

System monitoring

Full text search

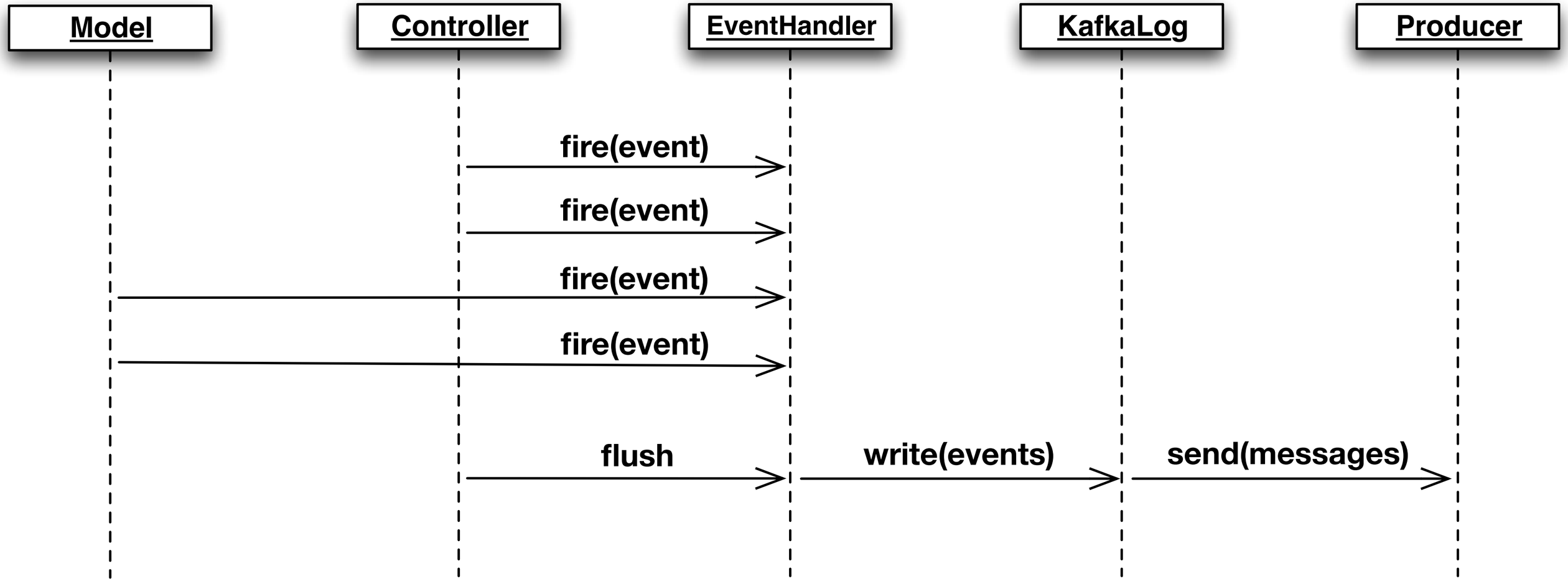
Trigger email newsletters

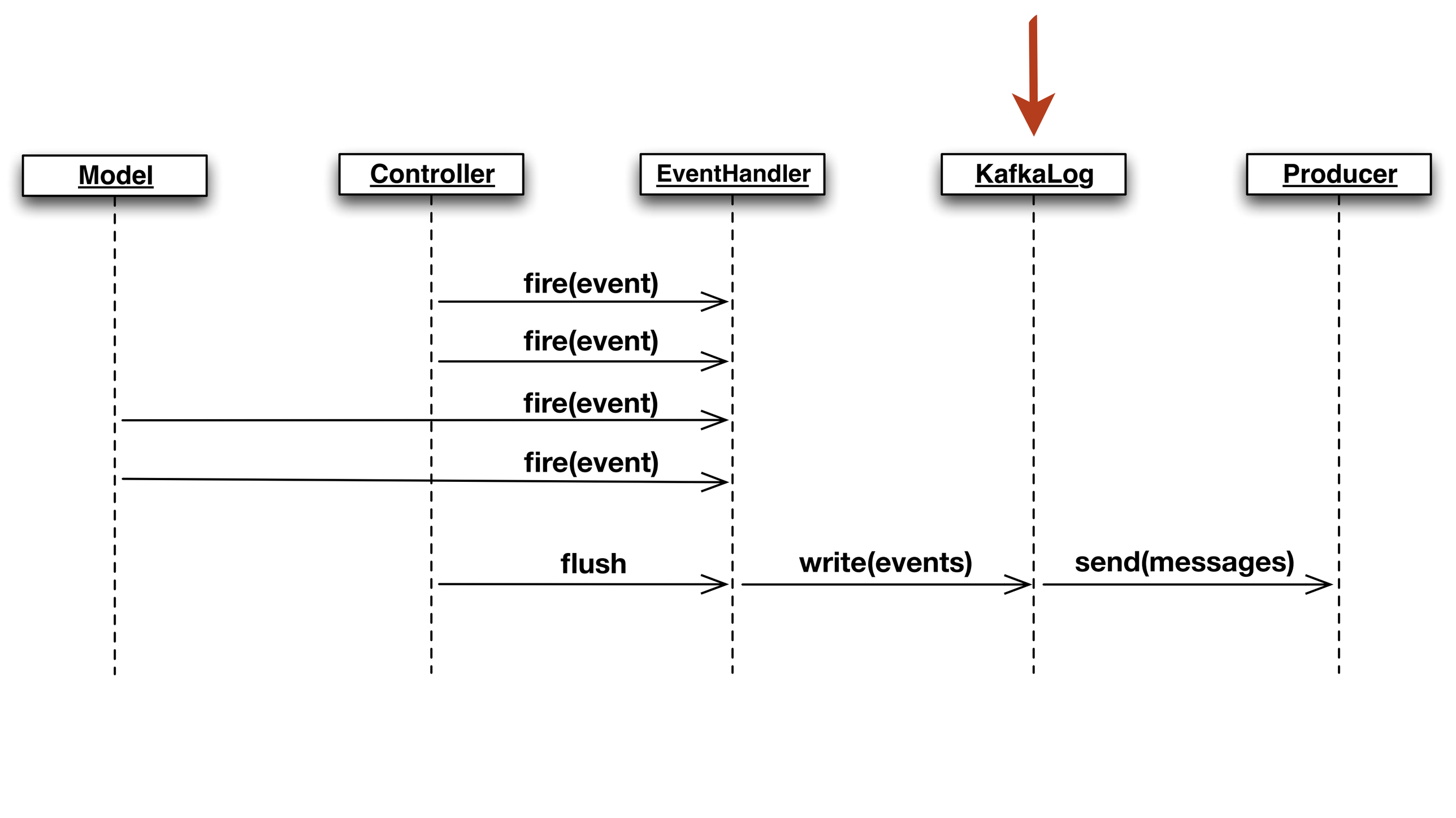
Message Requirements

1) Provide each message as a uniform JSON payload containing:
- Event name
- Timestamp of the event's occurrence
- Actor User ID and created_at timestamp
- Attributes

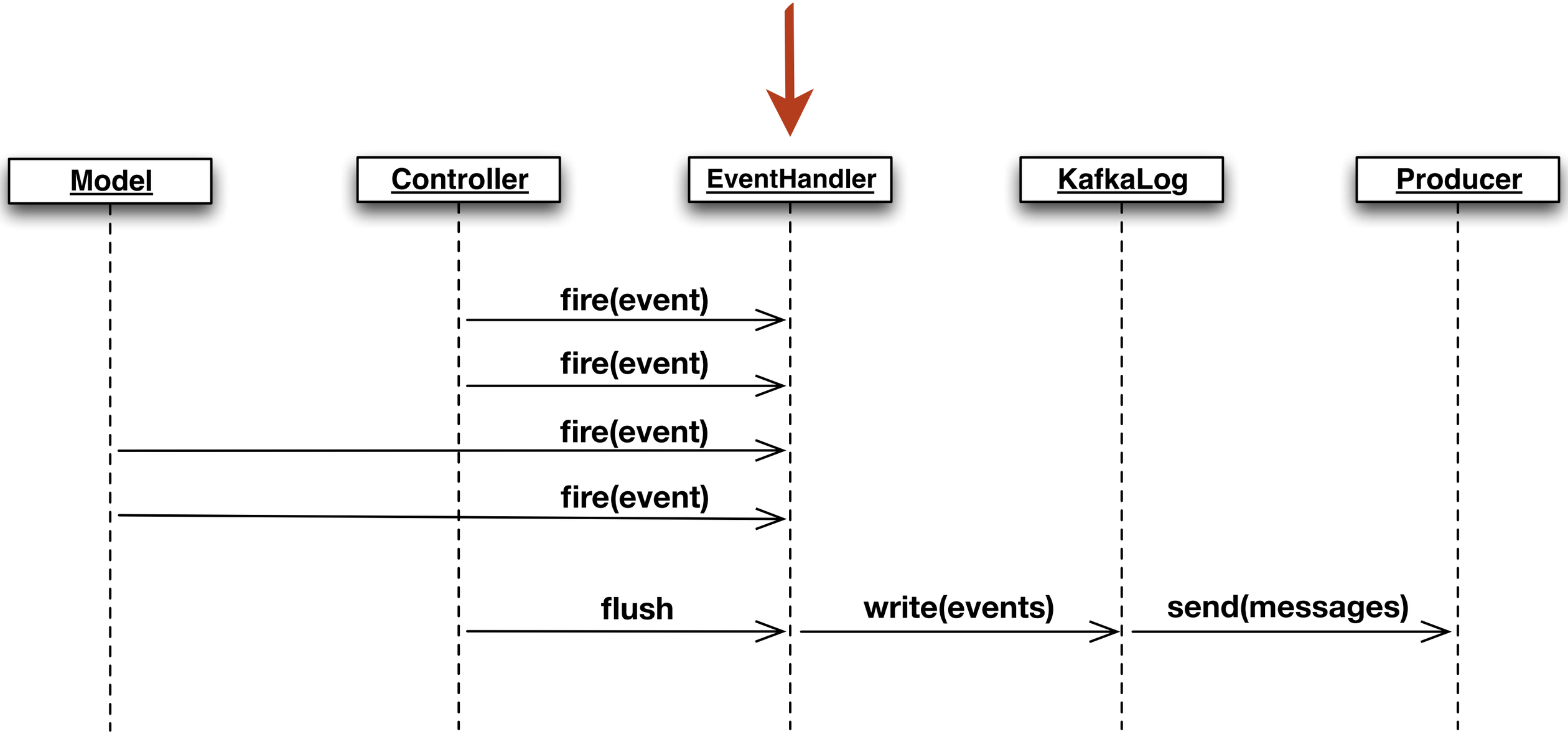
2) Transmit messages to Kafka asynchronously

3) Maximize producer performance by batching messages together when possible

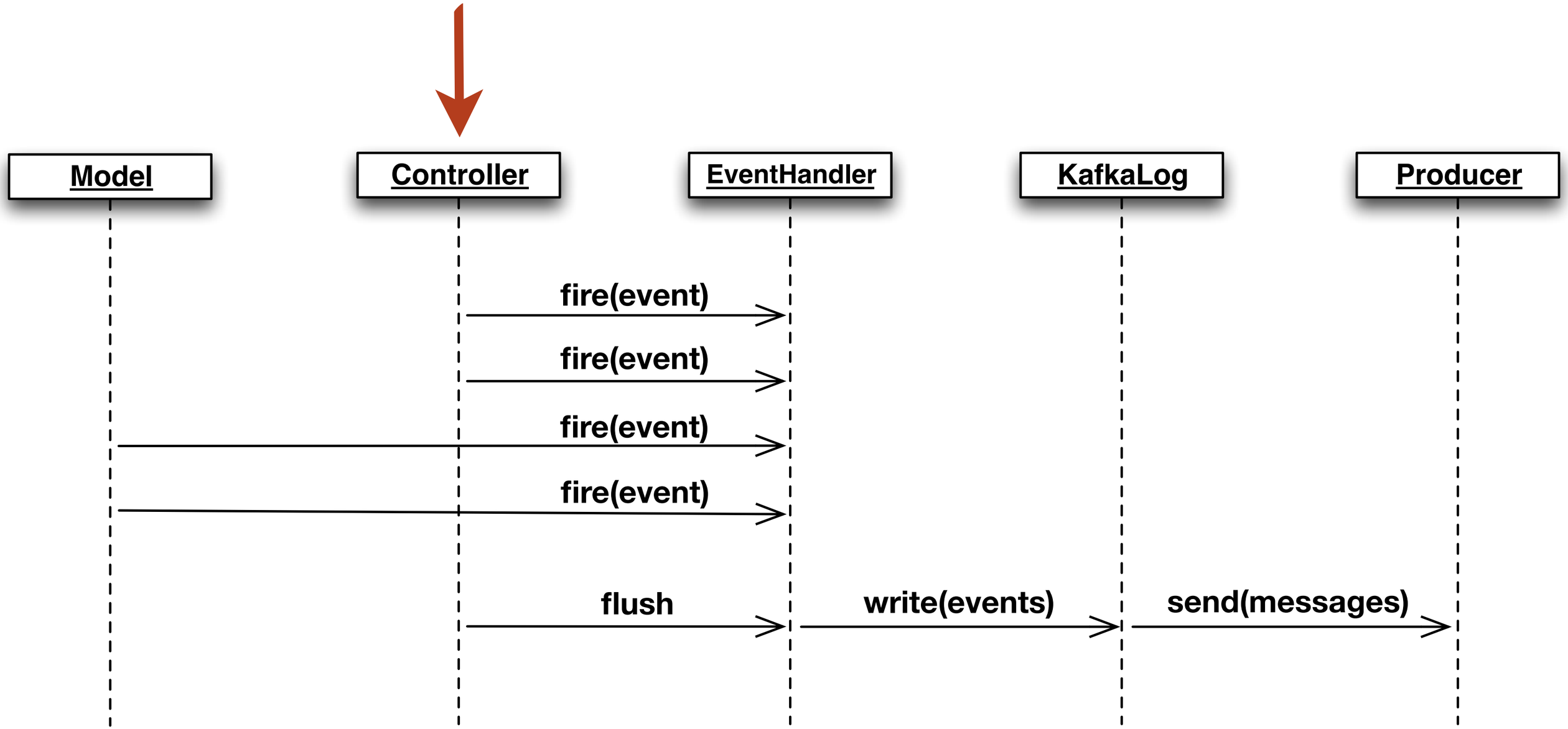

Model
Controller
EventHandler
KafkaLog
Producer
fire(event)
fire(event)
fire(event)
fire(event)
flush
write(events)
send(messages)


Model
Controller
EventHandler
KafkaLog
Producer
fire(event)
fire(event)
fire(event)
fire(event)
flush
write(events)
send(messages)

```ruby
class KafkaLog
  include Singleton

  def initialize
    @queue = Queue.new
  end

  def write(messages)
    @queue.push(messages)
  end

  def start(producer)
    Thread.new do
      while batch = @queue.pop
        producer.batch do
          batch.each do |message|
            producer.send(Kafka::Message.new(message))
          end
        end
      end
    end
  end
end
```


Model
Controller
EventHandler
KafkaLog
Producer
fire(event)
fire(event)
fire(event)
fire(event)
flush
write(events)
send(messages)

```ruby
class EventHandler
  def initialize(logger)
    @logger   = logger
    @messages = []
  end

  def fire(event, user, attributes={})
    payload = {
      event:      event,
      timestamp:  Time.now.to_f,
      attributes: attributes,
      user: {
        id:         user.id,
        created_at: user.created_at.to_f
      }
    }

    @messages.push(payload.to_json)
  end

  def flush
    @logger.write(@messages) if @messages.present?
  end
end
```


Model
Controller
EventHandler
KafkaLog
Producer
fire(event)
fire(event)
fire(event)
fire(event)
flush
write(events)
send(messages)

```ruby
class ApplicationController < ActionController::Base
  after_filter :flush_events_to_log

  def event_handler
    @event_handler ||= EventHandler.new(KafkaLog.instance)
  end

  def flush_events_to_log
    @event_handler.flush
  end
end
```

```ruby
class PostsController < ApplicationController
  def create
    @post = Post.new(params[:posts])

    if @post.save
      event_handler.fire("post.create", current_user,
        id:    @post.id,
        title: @post.title,
        body:  @post.body
      )
    end
  end
end
```

```ruby
class PostsController < ApplicationController
  def show
    @post = Post.find(params[:id])

    event_handler.fire("post.show", current_user,
      id:    @post.id,
      title: @post.title
    )
  end
end
```

```ruby
# config/initializers/kafka_log.rb

producer = Kafka::Producer.new(topic: “blog_log”)
KafkaLog.instance.start(producer)
```

```ruby
  desc "Tail from the Kafka log file"
  task :tail, [:topic] => :environment do |task, args|
    topic    = args[:topic].to_s
    consumer = Kafka::Consumer.new(topic: topic)

    puts "==> #{topic} <=="

    consumer.loop do |messages|
      messages.each do |message|
        json = JSON.parse(message.payload)
        puts JSON.pretty_generate(json), "\n"
      end
    end
  end
end
```

THANK YOU!  @jpignata

Slides
https://speakerdeck.com/u/jpignata/p/kafka-the-great-logfile-in-the-sky

Video of Presentation @ Pivotal Labs
http://www.livestream.com/pivotallabs/video?clipId=pla_edbd81df-89ec-4933-8295-42bf91a9d301

Demo Application Repo
http://github.com/jpignata/kafka-demo/

Apache Incubator: Kafka
http://incubator.apache.org/kafka/

Kafka Papers & Presentations
https://cwiki.apache.org/KAFKA/kafka-papers-and-presentations.html

Kafka Design
http://incubator.apache.org/kafka/design.html

Kafka: A Distributed Messaging System for Log Processing
http://research.microsoft.com/en-us/um/people/srikanth/netdb11/netdb11papers/netdb11-final12.pdf

IEEE Data Engineering Bulletin (July, 2012): Big Data War Stories
http://sites.computer.org/debull/A12june/A12JUN-CD.pdf