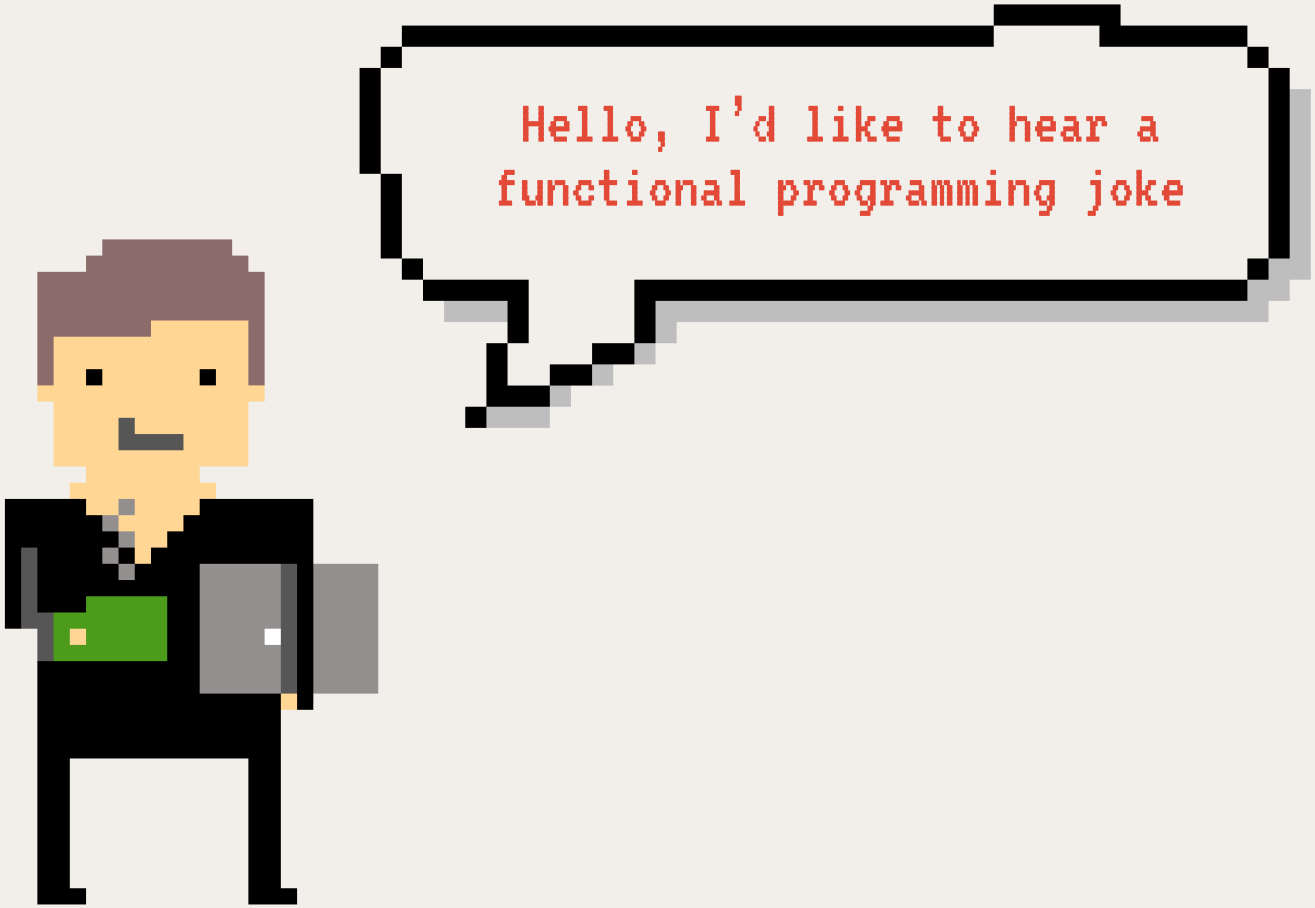
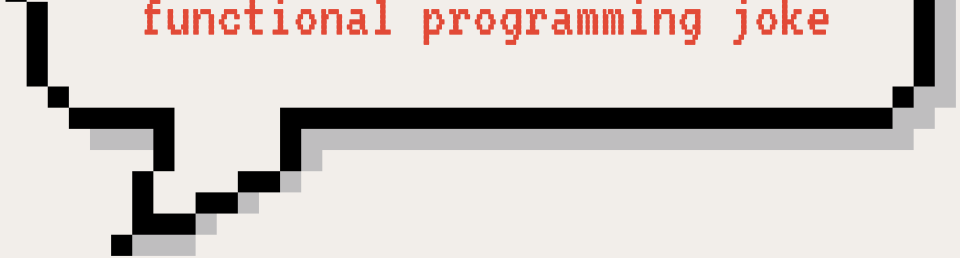
# FAST & FUNCTIONAL

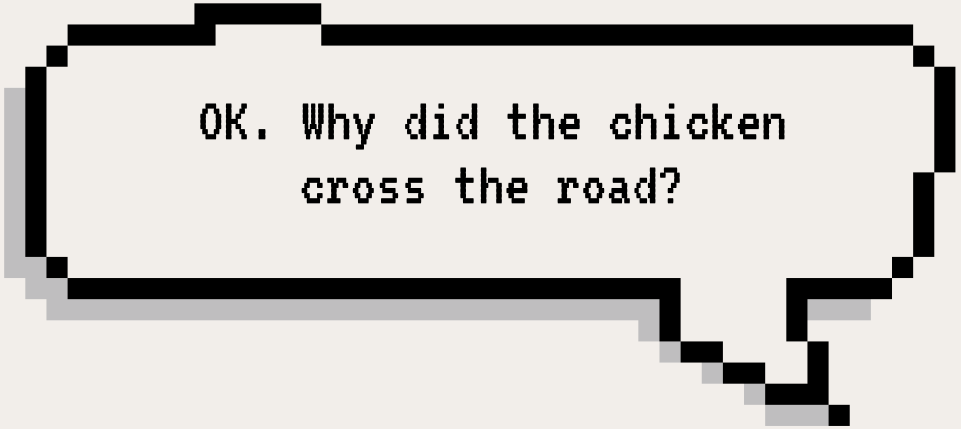
REALITY GAMES

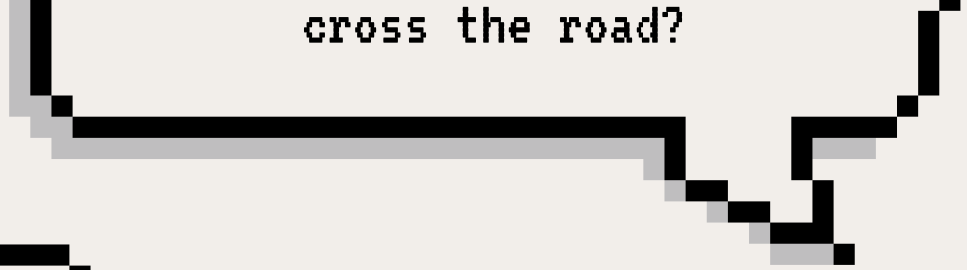
Michał Płachta
@miciek
www.michalplachta.com

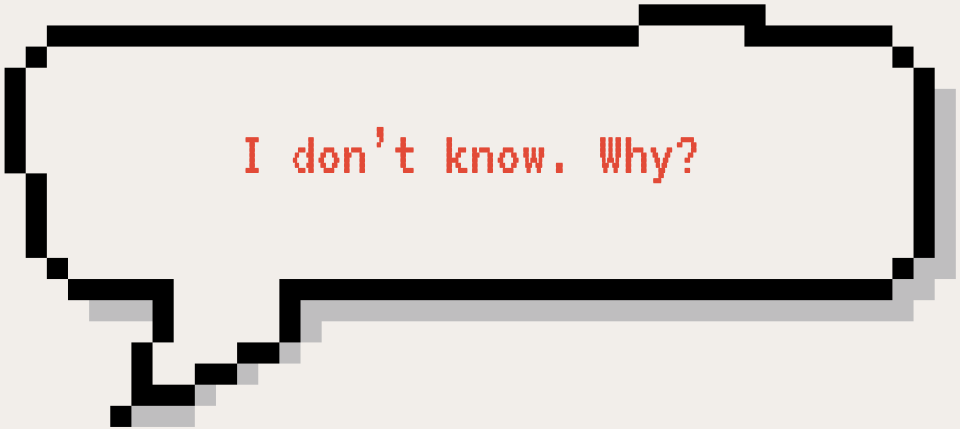


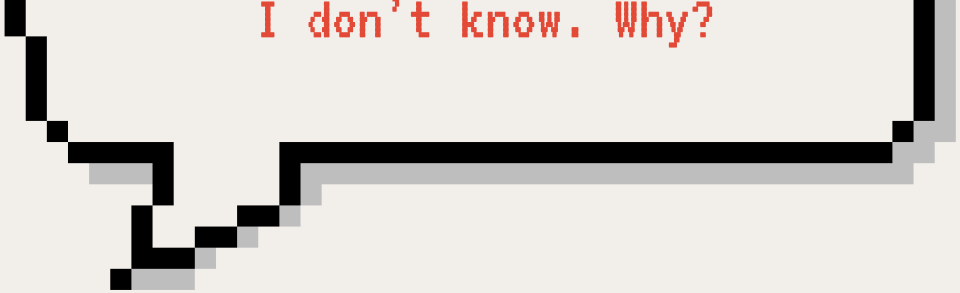

@miciek

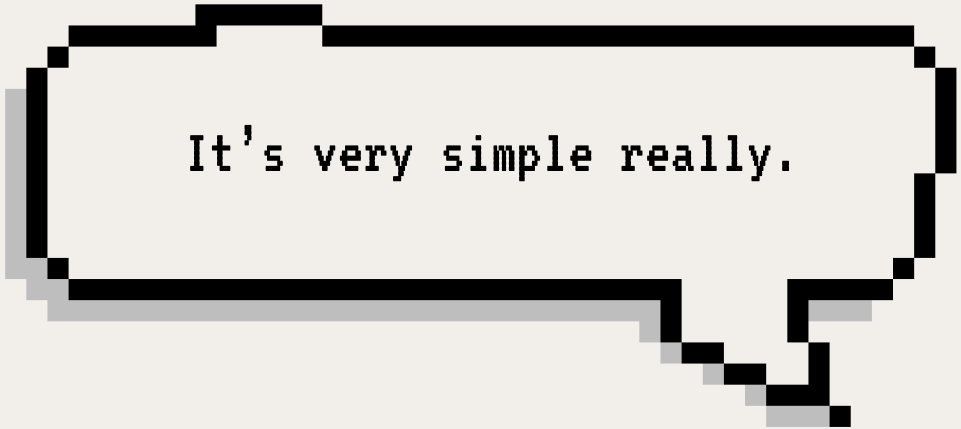

functional programming joke

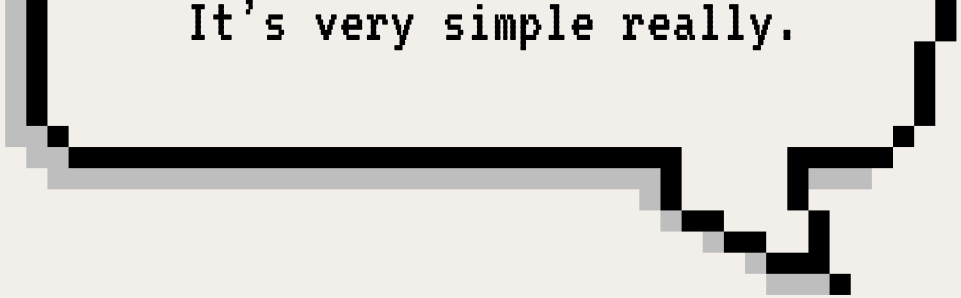

OK. Why did the chicken
cross the road?

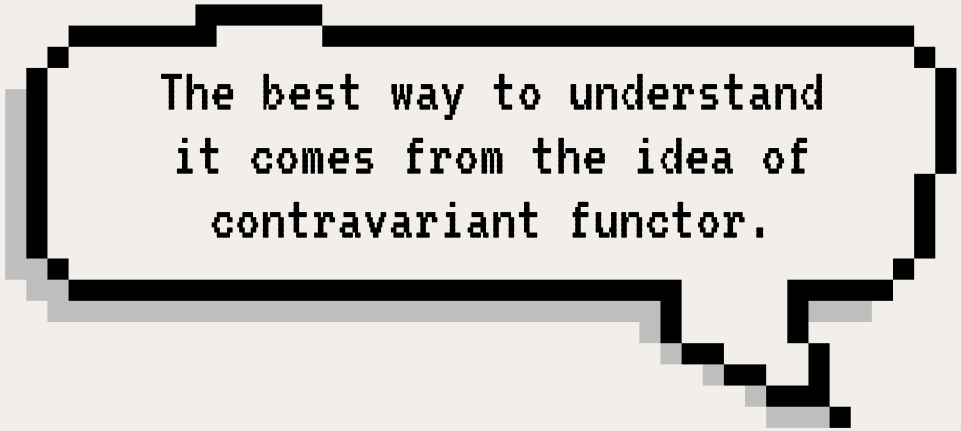

cross the road?


I don't know. Why?


I don't know. Why?


It's very simple really.


It's very simple really.


The best way to understand
it comes from the idea of
contravariant functor.

contravariant functor.

As you probably know, a functor is just a mapping between categories that respects the category structure; a homomorphism, if you will.

will.

Now, some constructions that we want to express look like functors but in some sense “turn morphisms around”. We call these contravariant functors.

As you have probably noticed, we can indeed define a contravariant functor as simply a covariant functor on opposite categories, invoking the obvious correspondence. Of course you have already noticed the obvious connection this has with the category theoretic generalization of topological presheaves - a presheaf on a topology is just a contravariant functor from the…


Igal Tabachnik
@hmemcpy
Hello, I would like to hear a Haskell joke.
User: Hello, I would like to hear a Haskell joke.
Haskell: Okay. Why did the chicken cross the road?
User: I don't know, why?
Haskell: It's very simple really. The best way to understand it comes from the idea of a contravariant functor. As you probably know, a functor is just a mapping between categories that respects the category structure; a homomorphism between categories, if you will. Now, some constructions that we want to express look like functors but in some sense "turn morphisms around". We call these contravariant functors. As you have probably noticed, we can indeed define a contravariant functor as simply a covariant functor on opposite categories, invoking the obvious correspondence. Of course, you've already noticed the obvious connection that this has with the category theoretic generalization of topological presheaves -- a presheaf on a topology is just a contravariant functor from the poset of open sets to the category of sets. Now, it's obvious that
8:23 AM · Jan 8, 2019 · Twitter Web Client
257 Retweets
573 Likes
Igal Tabachnik @hmemcpy · Jan 8
Replying to @hmemcpy
Aside: if you ever use the words "obviously, simply, probably" to describe a subject, don't.
4
4
63

# Learning FP


I Am Devloper @iamdevloper · Mar 14
Every functional programming tutorial...
How to draw an owl
1.
2.
1. Draw some circles
2. Draw the rest of the owl
61
3.2K
5.3K


@miciek

# Functional

**simple real-world service in Scala**

very modular

using functional mechanics

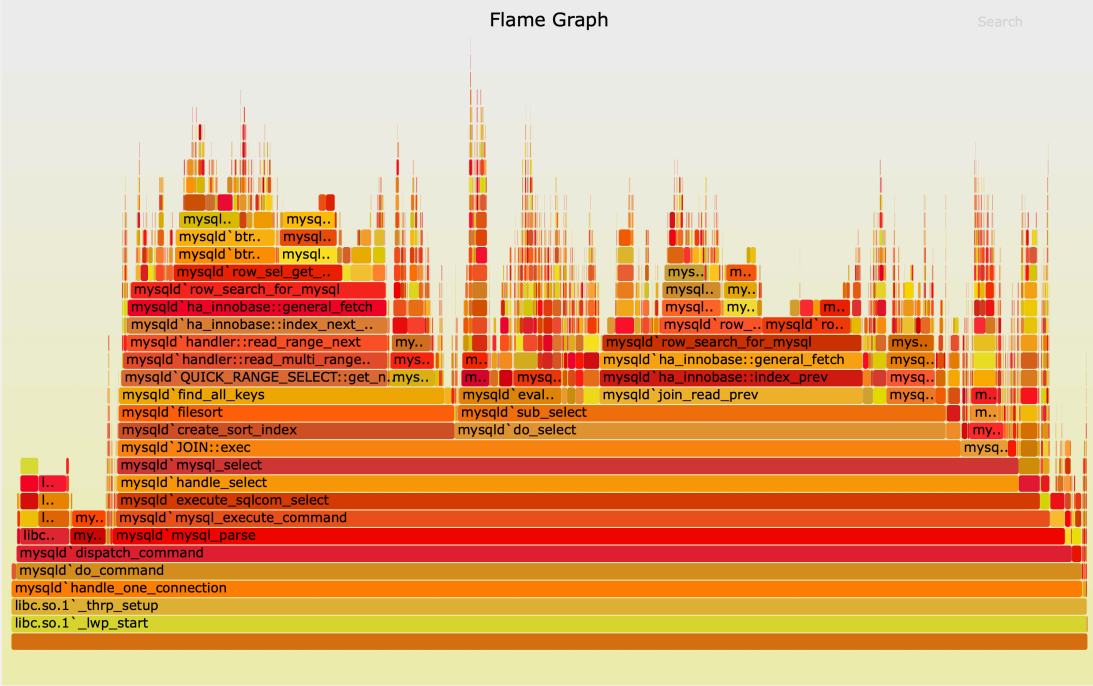
plug & play

quickly replacing modules without affecting the logic

# FAST

1st meaning:

**PERFORMANCE**



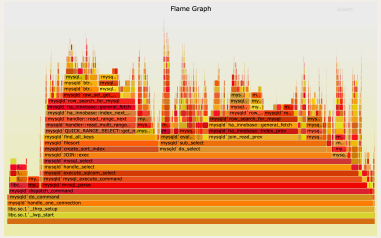
# FAST

plug & play

very modular

2nd meaning:

**DEV SPEED**




# In this talk…

modularity (the λ way!)

**FAST & FUNCTIONAL**

plug & play

**simple real-world service in Scala**

# Influencer Stats

# Marketing

# Old Ways Don't Work

Remember when...

...people watched TV

# Influencers

22 237 likes

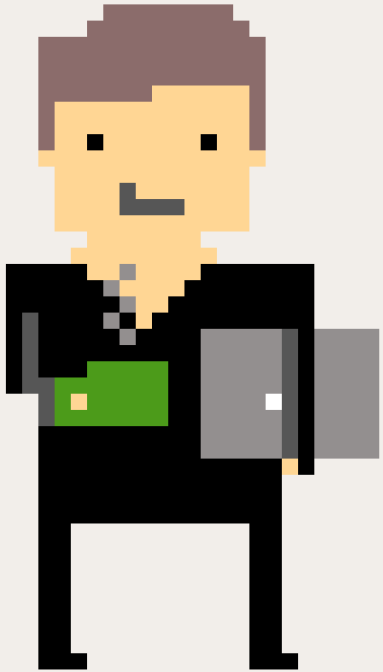
2302 likes, 70 retweets

...and more

201 232 views, 23987 likes


502 123 views, 2012 likes


# Influencers items aggregation



**1 201 882 views, 423 987 likes**

...and more

All tweets, youtube videos, etc for **one product**

# Influencer Stats service

**collection of videos**
how many **total views**
how many **total likes**

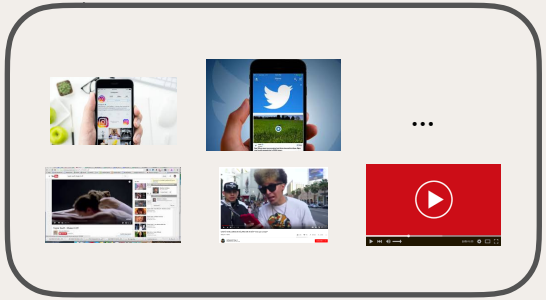
# Requirement #1

## Collections

ability to

add & update

`Collections` of videos




COLLECTION 1

...

COLLECTION 2

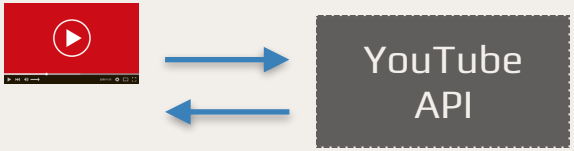
# Requirement #2

## Fetch YouTube Stats

ability to

fetch individual video stats through

YouTube API




20 views, 2 likes

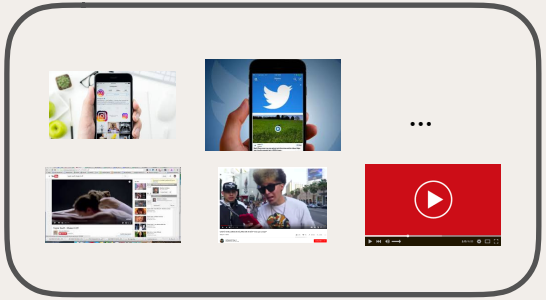
# Requirement #3

## Calculate Stats

ability to

calculate & return

Collection statistics




**1 201 882 views, 423 987 likes**

# Non-functional requirements

- ✓HTTP Web Server
- ✓HTTP Client
- ✓Collection Storage
- ✓Logging

## PURE

✓**Calculate Stats**

## Side-effect'y

✓**Save & Load Collections**
✓**Fetch YouTube Stats**
✓HTTP Web Server
✓HTTP Client
✓Collection storage
✓Logging

# Modular programming using algebras

@miciek

# Model

```
final case class Collection(videos: List[String])
```

```
final case class InfluencerItem(views: Int,
                                likes: Int,
                                dislikes: Int,
                                comments: Int)
```

```
final case class CollectionStats(impressions: Int,
                                 engagements: Int,
                                 score: Int)
```

# Pure logic

```
def calculate(items: List[InfluencerItem]): CollectionStats = {


}
```

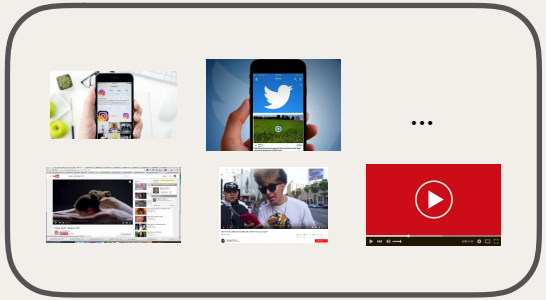
# Pure logic

```
def calculate(items: List[InfluencerItem]): CollectionStats = {
  items.foldLeft(CollectionStats.empty) { (results, item) =>
    
    
    
    
    
    
    
  }
}
```

# Pure logic

```
def calculate(items: List[InfluencerItem]): CollectionStats = {
  items.foldLeft(CollectionStats.empty) { (results, item) =>
    CollectionStats(
      impressions = results.impressions + item.views,
      engagements = results.engagements + item.comments +
                    item.likes + item.dislikes,
      score = results.engagements + item.likes
    )
  }
}
```

# Pure logic




**1 201 882 views, 423 987 likes**

```
def calculate(items: List[InfluencerItem]): CollectionStats = {
```

easy to reason about

easy to test

# Can it be as easy as this?

```
def getCollectionResults(collectionId: String): CollectionStats = {




}
```

# Can it be as easy as this?

```
def getCollectionResults(collectionId: String): CollectionStats = {
```

1. get collection `collectionId`
2. get `videoIds` from it
3. for each `videoId`, call YouTube API
4. call our pure `calculate` function

(and log some things in between)

```
}
```

# Can it be as easy as this?

```
def getCollectionResults(collectionId: String): CollectionStats = {
  for {
```

1. get collection `collectionId`
2. get `videoIds` from it
3. for each `videoId`, call YouTube API
4. call our `calculate` function

```
  } yield
}
```

# Side-effect'y stuff


```
public CompletableFuture<CollectionStats> getStats(String collectionId) {
    return getCollection("collectionId")
            .thenCompose(Statistics::fetchVideoStatistics)
            .thenCompose(collection ->
                logging.info("fetched collection: " + collection)
                        .thenCompose(ignored -> {
```

# Side-effect'y stuff

```
def getStats(collectionId: String): Future[CollectionStats]
  for {
    1. get collection collectionId
    2. get videoIds from it
    3. for each videoId, call YouTube API
    4. call our pure calculate function
  } yield
}
```

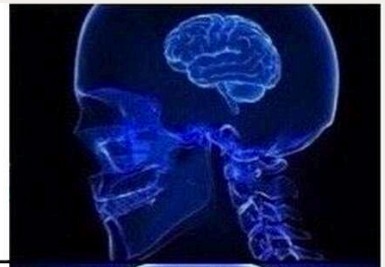
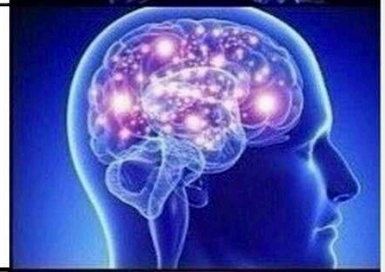
# Side-effect'y stuff

```
def getStats(collectionId: String): IO[CollectionStats]
  for {
    1. get collection collectionId
    2. get videoIds from it
    3. for each videoId, call YouTube API
    4. call our pure calculate function
  } yield
}
```

F doesn't do anything
F is chosen by user

# Side-effect'y stuff

```
def getStats[F[_]: Monad](collectionId: String): F[CollectionStats] =
  for {
    1. get collection collectionId
    2. get videoIds from it
    3. for each videoId, call YouTube API
    4. call our pure calculate function
  } yield
}
```

| | |
|---|---|
| synchronous |  |
| asynchronous |  |
| `IO` effect type |  |
| `F[_]` |  |

# Constrained?

```
public static <A> A f(List<A> list)
```

# Constrained?

```
public static <A> A f(List<A> list) {
    return list.get(0);
}
```

# We can't constrain ourselves

## What we have

```
public static CompletableFuture<Collection> getCollection(String id) {
```

## What we want

```
public static <F<_>> F<Collection> getCollection(String id) {
```

F doesn't do anything
F is chosen by user

# Constraints Liberate

```
def getStats[F[_]: Monad                                    ]
            (collectionId: String): F[CollectionStats] = {
  for {

  }
}
```

# Get collection `collectionId`

```
def getStats[F[_]: Monad: CollectionView                    ]
             (collectionId: String): F[CollectionStats] = {
  for {
    collection <- CollectionView[F].fetchCollection(collectionId)



  }
}
```

# Get collection `collectionId`

```
def getStats[F[_]: Monad: CollectionView                    ]
            (collectionId: String): F[CollectionStats] = {
  for {
    collection <- CollectionView[F].fetchCollection(collectionId)



  }
}
```

just an algebra

```
trait CollectionView[F[_]] {
  def fetchCollection(id: String): F[Option[Collection]]
}
```

# Get videoIds from it

```
def getStats[F[_]: Monad: CollectionView                    ]
            (collectionId: String): F[CollectionStats] = {
  for {
    collection <- CollectionView[F].fetchCollection(collectionId)
    videoIds   = collection.map(_.videos).getOrElse(List.empty)


  }
}
```

# For each `videoId` call YouTube API

```
def getStats[F[_]: Monad: CollectionView: VideoClient         ]
             (collectionId: String): F[CollectionStats] = {
  for {
    collection <- CollectionView[F].fetchCollection(collectionId)
    videoIds    = collection.map(_.videos).getOrElse(List.empty)
    responses  <- videoIds.map(VideoClient[F].fetchVideoListResponse).sequence
    items       = responses.flatMap(responseToItems)
  }
}
```

# For each `videoId` call YouTube API

```
def getStats[F[_]: Monad: CollectionView: VideoClient          ]
            (collectionId: String): F[CollectionStats] = {
  for {
    collection <- CollectionView[F].fetchCollection(collectionId)
    videoIds    = collection.map(_.videos).getOrElse(List.empty)
    responses  <- videoIds.map(VideoClient[F].fetchVideoListResponse).sequence
    items       = responses.flatMap(responseToItems)
  }
}
```

```
trait VideoClient[F[_]] {
  def fetchVideoListResponse(videoId: String): F[VideoListResponse]
}
```

# Call our pure `calculate` function

```
def getStats[F[_]: Monad: CollectionView: VideoClient          ]
             (collectionId: String): F[CollectionStats] = {
  for {
    collection <- CollectionView[F].fetchCollection(collectionId)
    videoIds   = collection.map(_.videos).getOrElse(List.empty)
    responses  <- videoIds.map(VideoClient[F].fetchVideoListResponse).sequence
    items      = responses.flatMap(responseToItems)
  } yield calculate(items)
}
```

# (and log some things in between)

```
def getStats[F[_]: Monad: CollectionView: VideoClient: Logging]
            (collectionId: String): F[CollectionStats] = {
  for {
    _          <- Logging[F].info(s"trying to fetch $collectionId")
    collection <- CollectionView[F].fetchCollection(collectionId)
    _          <- Logging[F].info(s"fetched collection: $collection")
    videoIds   = collection.map(_.videos).getOrElse(List.empty)
    _          <- Logging[F].info(s"going to make ${videoIds.size} fetches")
    responses  <- videoIds.map(VideoClient[F].fetchVideoListResponse).sequence
    _          <- Logging[F].info(s"got responses: $responses")
    items      = responses.flatMap(responseToItems)
    _          <- Logging[F].info(s"got list of influencer items: $items")
  } yield calculate(items)
}
```

# (and log some things in between)

```
def getStats[F[_]: Monad: CollectionView: VideoClient: Logging]
             (collectionId: String): F[CollectionStats] = {
  for {
    _           <- Logging[F].info(s"trying to fetch $collectionId")
    collection <- CollectionView[F].fetchCollection(collectionId)
    _           <- Logging[F].info(s"fetched collection: $collection")
    videoIds    = collection.map(_.videos).getOrElse(List.empty)
    _           <- Logging[F].info(s"going to make ${videoIds.size} fetches")
    responses   <- videoIds.map(VideoClient[F].fetchVideoListResponse).sequence
    _           <- Logging[F].info(s"got responses: $responses")
    items       = responses.flatMap(responseToItems)
    _           <- Logging[F].info(s"got list of influencer items: $items")
  } yield calculate(items)
}
```

```
trait Logging[F[_]] {
  def info(msg: String): F[Unit]
}
```

## PURE

- ✓**Calculate Stats**
- ✓Collections **algebra**
- ✓HTTP Client **algebra**
- ✓Logging **algebra**
- ✓**getStats algorithm on F**

## Side-effect'y

- ✓Collections **interpreter**
- ✓HTTP Web Server
- ✓HTTP Client **interpreter**
- ✓Collection storage **interpreter**
- ✓Logging **interpreter**

## algebras

- ✓Collections **algebra**
- ✓HTTP Client **algebra**
- ✓Logging **algebra**

## interpreters

- ✓Collections **interpreter**
- ✓HTTP Client **interpreter**
- ✓Collection storage **interpreter**
- ✓Logging **interpreter**

# algebra

```
trait VideoClient[F[_]] {
  def fetchVideoListResponse(videoId: String): F[VideoListResponse]
}
```

# interpreter

```
class AkkaHttpVideoClient(youtubeUri: String, youtubeApiKey: String)
                         (implicit system: ActorSystem)
    extends VideoClient[IO] {
  def fetchVideoListResponse(videoId: String): IO[VideoListResponse] = {
    implicit val materializer: ActorMaterializer = ActorMaterializer()
    implicit val ec: ExecutionContextExecutor    = system.dispatcher

    IO.fromFuture(IO {
      Http()
        .singleRequest(
          HttpRequest(
            uri = s"$youtubeUri?part=statistics" +
            s"&id=$videoId&key=$youtubeApiKey"
          )
        )
        .flatMap(Unmarshal(_).to[youtube.VideoListResponse])
    })
  }
```

## algebra

```
trait VideoClient[F[_]] {
  def fetchVideoListResponse(videoId: String): F[VideoListResponse]
}
```

```
trait CollectionView[F[_]] {
  def fetchCollection(id: String): F[Option[Collection]]
}
```

```
trait Logging[F[_]] {
  def info(msg: String): F[Unit]
}
```

## interpreters

- AkkaHttp
- Hammock (Apache)
- Http4s

- InMem LinkedList
- InMem TrieMap

- Log All
- Max 1k/sec (dropping rest)

plug & play

# Make it fast

# Warning

Do not perform any of the following stunts
on **PRODUCTION.**

# We will use wrk

16
connections

```
wrk -t1 -c16 -d30s --latency URL
```

1 thread

30 sec
duration

# We will use async-profiler

```
jps
cd async-profiler
./profiler.sh -d 10 -f <FILE> <PID>
```

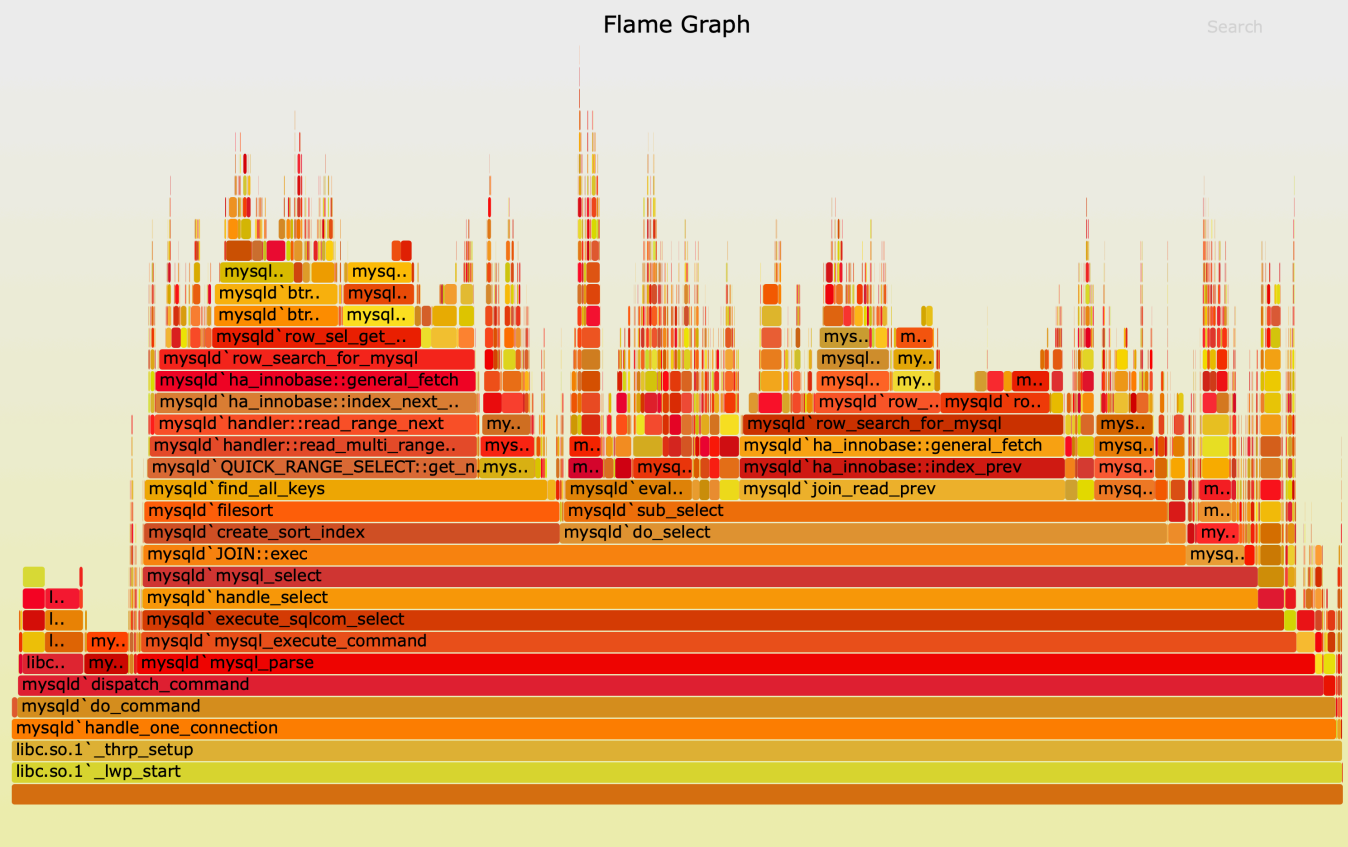
## async-profiler

This project is a low overhead sampling profiler for Java that does not suffer from Safepoint bias problem. It features HotSpot-specific APIs to collect stack traces and to track memory allocations. The profiler works with OpenJDK, Oracle JDK and other Java runtimes based on HotSpot JVM.



https://github.com/jvm-profiling-tools/async-profiler

# We will use flame graphs




read from bottom



http://www.brendangregg.com/flamegraphs.html

# Version 1

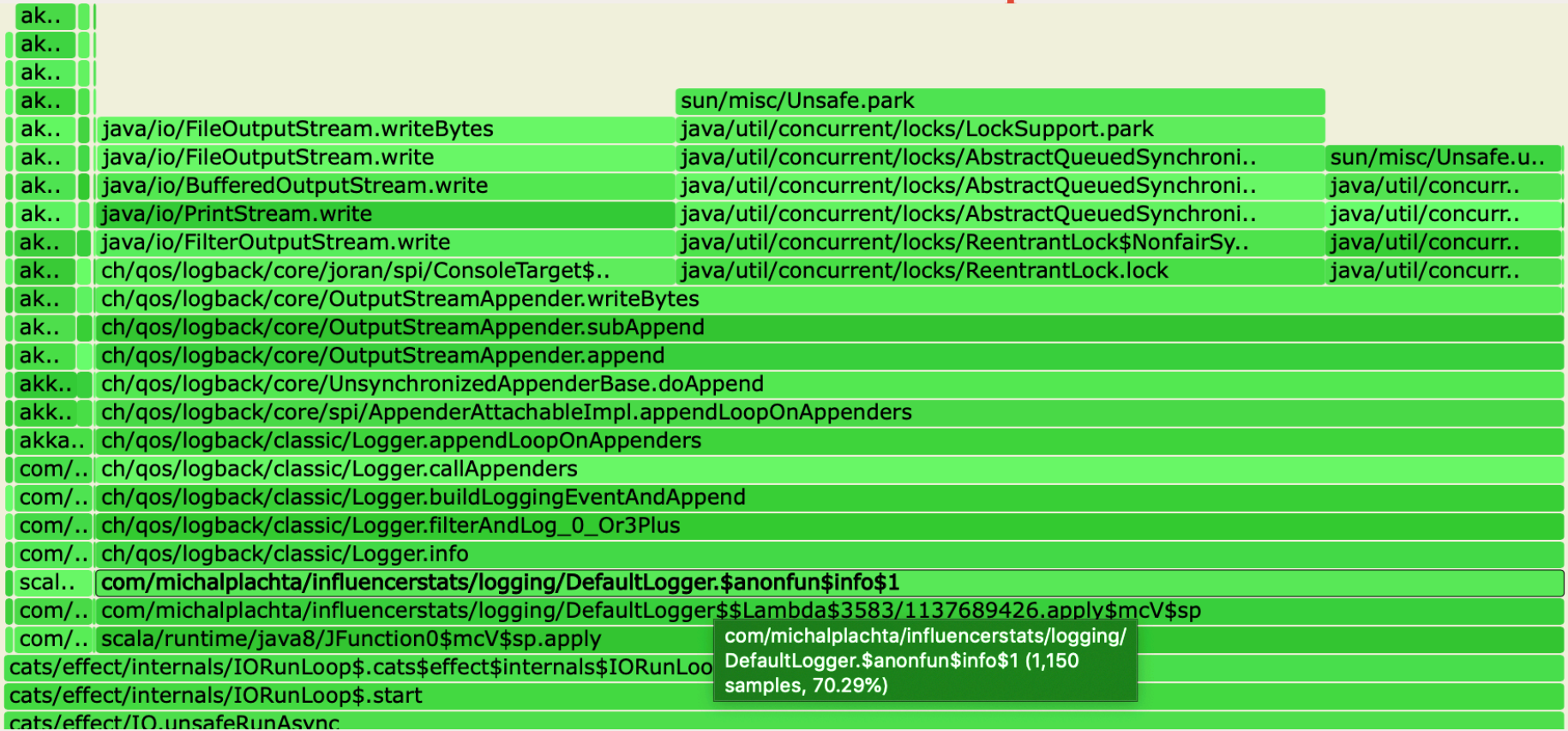
- akka-http (client + server)
- log all things
- in mem linked list state

| | # requests | 99% latency | avg latency | req/s |
|---|---|---|---|---|
| **Version 1** | ? | ?ms | ?ms | ? |

# Version 1 Results

| | # requests | 99% latency | avg latency | req/s |
|---|---|---|---|---|
| **Version 1** | 8510 | 82.41ms | 56.47ms | 283.12 |

# Version 1 Flame Graph



# Version 2: Dropping logs

- don't change the logic
- just change the `Logger` interpreter
- dropping logs if the rate is > 1k per second

```
implicit val logging: Logging[IO] = new DefaultLogger[IO]
```

```
implicit val logging: Logging[IO] = new DroppingLogger[IO]
```

# Version 2 Results

DroppingLogger

| | # requests | 99% latency | avg latency | req/s |
|---|---|---|---|---|
| **Version 1** | 8510 | 82.41ms | 56.47ms | 283.12 |
| **Version 2** | 89 757 | 62.53ms | 6.94ms | 2986.26 |

# Version 2 Flame Graph

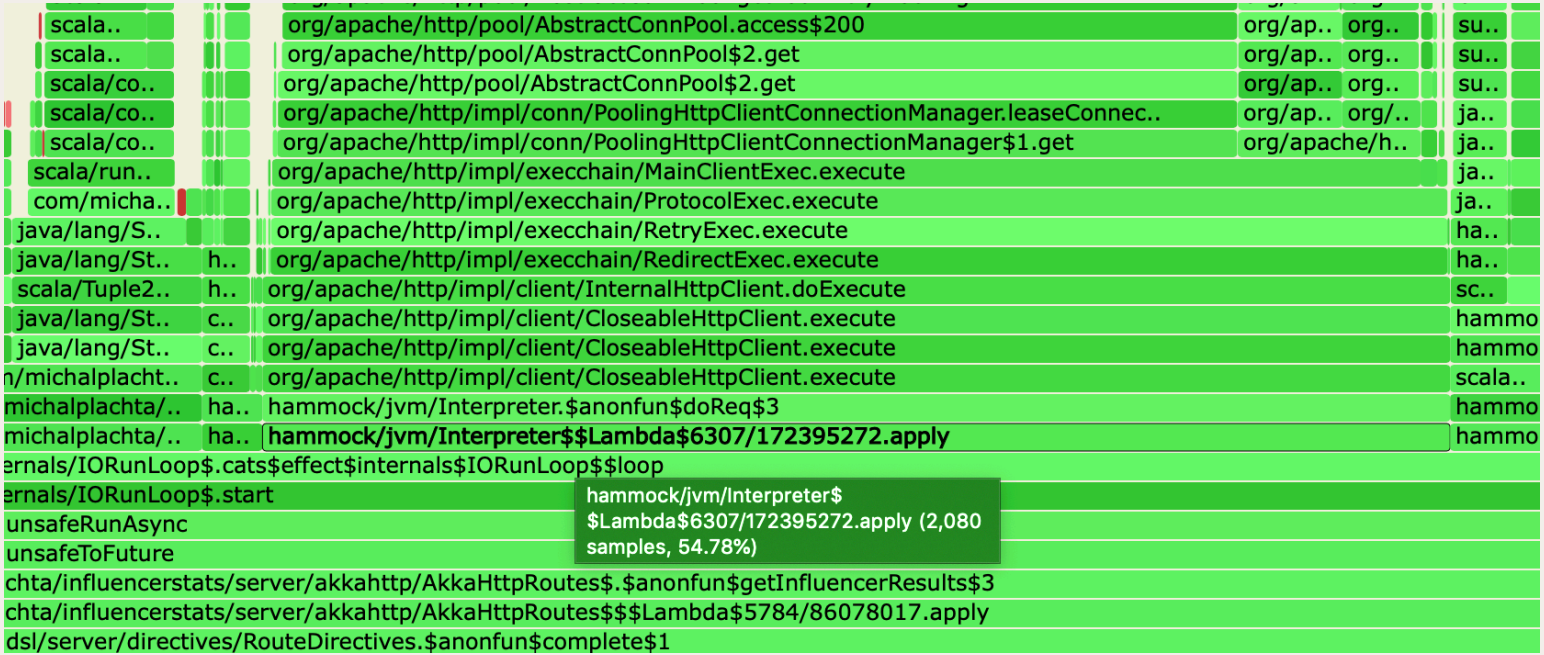
# Version 3: Change the Video Client

- don’t change the logic
- change the `VideoClient` type class instance

# Version 3 Results

| | | # requests | 99% latency | avg latency | req/s |
|---|---|---|---|---|---|
| DroppingLogger | Version 1 | 8510 | 82.41ms | 56.47ms | 283.12 |
| | Version 2 | 89 757 | 62.53ms | 6.94ms | 2986.26 |
| Hammock (Apache) | Version 3 | 96 564 | 13.05ms | 5.06ms | 3218.48 |

# Version 3 Flame Graph



# Version 4: Statistics Caching

- don’t change the logic
- cache results and call logic in separate thread

# Version 4 Results

| | | # requests | 99% latency | avg latency | req/s |
|---|---|---|---|---|---|
| | Version 1 | 8510 | 82.41ms | 56.47ms | 283.12 |
| DroppingLogger | Version 2 | 89 757 | 62.53ms | 6.94ms | 2986.26 |
| Hammock (Apache) | Version 3 | 96 564 | 13.05ms | 5.06ms | 3218.48 |
| Stats Caching | Version 4 | 933 303 | 62.86ms | 2.87ms | 31 081.82 |

# Version 4 Flame Graph


a..
sun/misc..
sun/misc/..
akka/dis..
akka/disp..
akka/act..
akka/acto..
sun/nio/ch/FileDispatcher..
sun/nio/..
sun/nio/ch/SocketDispatch..
sun/nio/ch/IOUtil.writeFr..
akka/str..
akka/stre..
sun/nio/ch/IOUtil.write
akka/stream/i..
sun/nio/ch/SocketChannelI..
akka/stream/impl/fusing/..
akka/io/TcpConnection$Pending..
akka/io/TcpC..
akka/io/TcpConnection.doWrite
akka/io/TcpCon..
akka/io/TcpConnection$$anonfu..
sun..
scala/PartialFunction$OrElse.applyOrElse
akka/actor/Actor.aroundR..
sun/n..
akka/actor/Actor.aroundReceive
sun/ni..
akka/actor/Actor.aroundReceive$
akka/io/TcpConnection.aroundReceive
akka/actor/ActorCell.receiveMessage
akka/io..
akka/actor/ActorCell.invoke
akka/dispatch/Mailbox.processMailbox
akka/dispatch/Mailbox.run
akka/ut..
akka/dispatch/Mailbox.exec
s..
akka/dispatch/forkjoin/ForkJoinTask.doExec
akka/dispatch/forkjoin/For..
sun/misc/Unsafe.park
akka/di..
akka/dispatch/forkjoin/ForkJoinPool$WorkQueue.runTask
akka/dispatch/forkjoin/ForkJoinPool.scan
java/uti..
akka/dispatch/forkjoin/ForkJoinPool.runWorker
akka/dispatch/forkjoin/ForkJoinWorkerThread.run
java/lan..
all

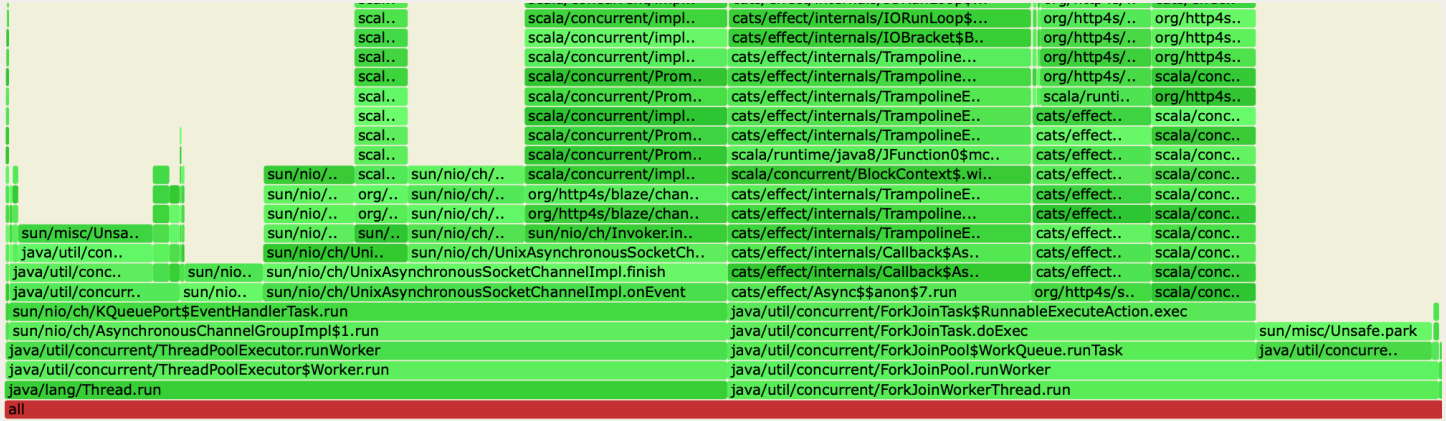
# Version 5: Change the Server

- don’t change the logic
- change the underlying server from Akka HTTP to Http4s

# Version 5 Results

| | | # requests | 99% latency | avg latency | req/s |
|---|---|---|---|---|---|
| | **Version 1** | 8510 | 82.41ms | 56.47ms | 283.12 |
| DroppingLogger | **Version 2** | 89 757 | 62.53ms | 6.94ms | 2986.26 |
| Hammock (Apache) | **Version 3** | 96 564 | 13.05ms | 5.06ms | 3218.48 |
| Stats Caching | **Version 4** | 933 303 | 62.86ms | 2.87ms | 31 081.82 |
| Http4s Server | **Version 5** | 1 158 664 | 9.05ms | 555.80µs | 38 596.24 |

# Version 5 Flame Graph



# Inventions That Already Exist

# Introducing...

**Pause Pod**


@miciek

**Antoine Linguine**
@aklingus

This is a tent. You invented a tent.

Przetłumacz Tweeta

**INSIDER** @thisisinsider · 26 wrz 2017
Enter the Pause Pod. 😴


5:58 AM · 29 wrz 2017 · Tweetbot for iOS

# Introducing...

**Cheaper taxi that drives on <u>fixed</u> routes!**

Cheaper taxi that drives on fixed routes!

**This is a BUS. You invented a BUS.**

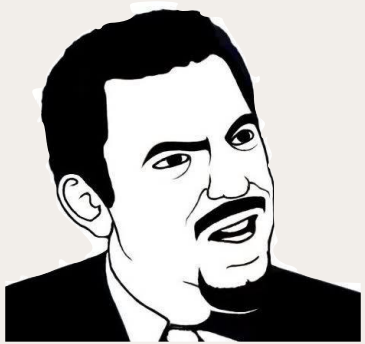
# Introducing...

## Algebras & Interpreters

```
def getStats[F[_]: Monad: CollectionView: VideoClient: Logging]
             (collectionId: String): F[CollectionStats] = {
  for {
    _          <- Logging[F].info(s"trying to fetch $collectionId")
    collection <- CollectionView[F].fetchCollection(collectionId)
    _          <- Logging[F].info(s"fetched collection: $collection")
    videoIds   = collection.map(_.videos).getOrElse(List.empty)
    _          <- Logging[F].info(s"going to make ${videoIds.size} fetches")
    responses  <- videoIds.map(VideoClient[F].fetchVideoListResponse).sequence
    _          <- Logging[F].info(s"got responses: $responses")
    items      = responses.flatMap(responseToItems)
    _          <- Logging[F].info(s"got list of influencer items: $items")
  } yield calculate(items)
}
```

Algebras & Interpreters

**This is an INTERFACE. You invented an INTERFACE.**



# Scala FP

```scala
def getStats[F[_]: Monad: CollectionView: VideoClient: Logging]
            (collectionId: String): F[CollectionStats] = {
  for {
    _           <- Logging[F].info(s"trying to fetch $collectionId")
    collection <- CollectionView[F].fetchCollection(collectionId)
    _           <- Logging[F].info(s"fetched collection: $collection")
    videoIds    = collection.map(_.videos).getOrElse(List.empty)
    _           <- Logging[F].info(s"going to make ${videoIds.size} fetches")
    responses   <- videoIds.map(VideoClient[F].fetchVideoListResponse).sequence
    _           <- Logging[F].info(s"got responses: $responses")
    items       = responses.flatMap(responseToItems)
    _           <- Logging[F].info(s"got list of influencer items: $items")
  } yield calculate(items)
}
```

# Java

```
public class Statistics {
    private CollectionView collectionView;
    private VideoClient videoClient;
    private Logging logging;

    public Statistics(CollectionView collectionView, VideoClient videoClient, Logging logging) {
        this.collectionView = collectionView;
        this.videoClient = videoClient;
        this.logging = logging;
    }

    public CompletableFuture<CollectionStats> getStats(String collectionId) {
        return logging.info("trying to fetch collection with id " + collectionId) CompletableFuture<Void>
                .thenCompose(ignored ->
                        collectionView.fetchCollection(collectionId)
                ) CompletableFuture<Optional<Collection>>
                .thenCompose(collection ->
                    logging.info("fetched collection: " + collection)
                            .thenCompose(ignored -> {
                                List<String> videoIds = collection.map(c -> c.videos).orElse(emptyList());
                                return logging.info("going to make " + videoIds.size() + " fetches")
                                        .thenCompose(ignored_ ->
                                            sequence(videoIds
                                                .stream()
                                                .map(videoClient::fetchVideoListResponse).collect(toList())));
                            })
                ) CompletableFuture<List<VideoListResponse>>
                .thenCompose(responses ->
                    logging.info("got responses: " + responses)
                        .thenCompose(ignored -> {
                            List<InfluencerItem> items = responses.stream().flatMap(r -> responseToItems(r).stream()).collect(toList());
                            return logging.info("got list of influencer items: " + items)
                                    .thenApply(ignored_ -> calculate(items));
                        })
                );
    }

    private static <T> CompletableFuture<List<T>> sequence(List<CompletableFuture<T>> futures) {
        return futures.stream() Stream<CompletableFuture<T>>
                .map(f -> f.thenApply(Stream::of)) Stream<CompletableFuture<Stream<T>>>
                .reduce((a, b) -> a.thenCompose(xs -> b.thenApply(ys -> concat(xs, ys)))) Optional<CompletableFuture<Stream<T>>>
                .map(f -> f.thenApply(s -> s.collect(toList()))) Optional<CompletableFuture<List<T>>>
                .orElse(completedFuture(emptyList()));
    }
```

@miciek

# Interfaces


```
public interface VideoClient {
    CompletableFuture<VideoListResponse> fetchVideoListResponse(String videoId);
}
```

```
public interface CollectionView {
    CompletableFuture<Optional<Collection>> fetchCollection(String id);
}
```

```
public interface Logging {
    CompletableFuture<Void> info(String msg);
}
```

# Interfaces


```
public interface VideoClient {
    CompletableFuture<VideoListResponse> fetchVideoListResponse(String videoId);
}
```

```
public interface CollectionView {
    CompletableFuture<Optional<Collection>> fetchCollection(String id);
}
```

```
public interface Logging {
    CompletableFuture<Void> info(String msg);
}
```

**difference:**

we need to choose the effect type 😞

# Algebras

```scala
trait VideoClient[F[_]] {
  def fetchVideoListResponse(videoId: String): F[VideoListResponse]
}
```

```scala
trait CollectionView[F[_]] {
  def fetchCollection(id: String): F[Option[Collection]]
}
```

```scala
trait Logging[F[_]] {
  def info(msg: String): F[Unit]
}
```

# Dependencies


```
public class Statistics {
    private CollectionView collectionView;
    private VideoClient videoClient;
    private Logging logging;

    public Statistics(CollectionView collectionView, VideoClient videoClient, Logging logging) {
        this.collectionView = collectionView;
        this.videoClient = videoClient;
        this.logging = logging;
    }
```

# Dependencies

```
def getStats[F[_]: Monad: CollectionView: VideoClient: Logging]
```

# Return type


```
public CompletableFuture<CollectionStats> getStats(String collectionId) {
    return logging.info("trying to fetch collection with id " + collectionId)
            .thenCompose(ignored ->
                    collectionView.fetchCollection(collectionId)
            ) CompletableFuture<Optional<Collection>>
            .thenCompose(collection ->
```

**difference:**

we need to choose the effect type 😞

# Abstracted effect type

```
def getStats[F[_]: Monad: CollectionView: VideoClient: Logging]
            (collectionId: String): F[CollectionStats] = {




}
```

# Abstracted effect type

```
def getStats[F[_]: Monad: CollectionView: VideoClient: Logging]
            (collectionId: String): F[CollectionStats] = {
  for {
    _          <- Logging[F].info(s"trying to fetch $collectionId")
    collection <- CollectionView[F].fetchCollection(collectionId)
    _          <- Logging[F].info(s"fetched collection: $collection")
    videoIds   = collection.map(_.videos).getOrElse(List.empty)
    _          <- Logging[F].info(s"going to make ${videoIds.size} fetches")
    responses  <- videoIds.map(VideoClient[F].fetchVideoListResponse).sequence
    _          <- Logging[F].info(s"got responses: $responses")
    items      = responses.flatMap(responseToItems)
    _          <- Logging[F].info(s"got list of influencer items: $items")
  } yield calculate(items)
}
```

# Specific effect type


```
public CompletableFuture<CollectionStats> getStats(String collectionId) {
    return logging.info("trying to fetch collection with id " + collectionId)
            .thenCompose(ignored ->
                    collectionView.fetchCollection(collectionId)
            ) CompletableFuture<Optional<Collection>>
            .thenCompose(collection ->
```

# Specific effect type


```
public CompletableFuture<CollectionStats> getStats(String collectionId) {
    return logging.info("trying to fetch collection with id " + collectionId)
            .thenCompose(ignored ->
                    collectionView.fetchCollection(collectionId)
            ) CompletableFuture<Optional<Collection>>
            .thenCompose(Statistics::sendHumanToMars)
            .thenCompose(collection ->
```

# Too powerful effect type :(


```
public CompletableFuture<CollectionStats> getStats(String collectionId) {
    return logging.info("trying to fetch collection with id " + collectionId)
            .thenCompose(ignored ->
                    collectionView.fetchCollection(collectionId)
            ) CompletableFuture<Optional<Collection>>
            .thenCompose(Statistics::sendHumanToMars)
            .thenCompose(collection ->
```

# Algebras & Interpreters

**Not an INTERFACE.**

**major difference:**
we can abstract over the effect type
and constrain ourselves 🎊

using `F` instead of `CompletableFuture`...

# Conclusions

# What have we done?

- changed infrastructure details
- ...but our core logic (and tests) didn't need to change!

plug & play

`quickly replacing non-core-logic modules`

# Modularity / Separation of concerns

if we can..

```
replace the server implementation
replace the state implementation
replace the logging implementation
replace the video client implementation
add caching
```

without

```
changing the core logic (& tests)
```

## PURE

- ✓**Calculate Stats**
- ✓Collections **algebra**
- ✓HTTP Client **algebra**
- ✓Logging **algebra**
- ✓**getStats algorithm on** F

just `F[_]`

very constrained

**unit-tested**

## Side-effect'y

- ✓Collections **interpreter**
- ✓HTTP Web Server
- ✓HTTP Client **interpreter**
- ✓Collection storage **interpreter**
- ✓Logging **interpreter**

`Future`, `IO`, ...

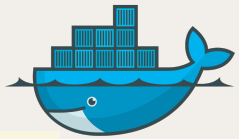
no constraints

**integration-tested**

# Interpreters integration test


https://www.testcontainers.org/

# Interpreters integration test



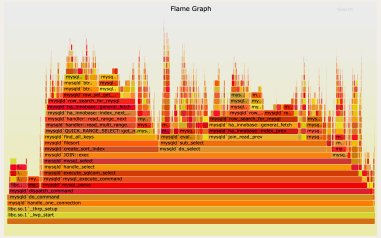
```
override val container =
  GenericContainer("miciek/influencer-stats-youtube:v1",
    exposedPorts = Seq(80), waitStrategy = Wait.forListeningPort)

"AkkaHttpVideoClient" should {
  "be able to get response from Youtube" in new WithYoutubeServer {
    val videoClient: AkkaHttpVideoClient =
      new AkkaHttpVideoClient(youtubeUri, "test")(ActorSystem("test"))

    videoClient
      .fetchVideoListResponse("-4lB5EKS5Uk")
      .unsafeRunSync() should be(
      VideoListResponse(
        List(
          Video(VideoStatistics(viewCount = 261734, likeCount = 14237))
        )
      )
    )
  }
```

interpreter

response



https://www.testcontainers.org/




You learned…

modularity (the λ way!)

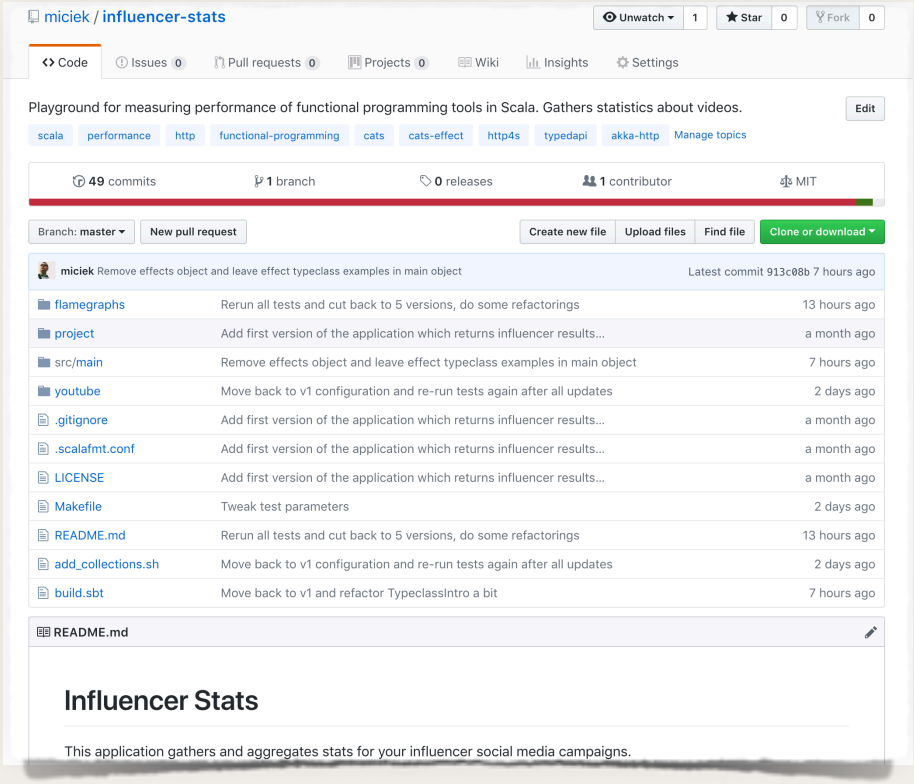
FAST & FUNCTIONAL

plug & play

**simple real-world service in Scala**

# Learn More

https://github.com/miciek/influencer-stats



# FAST & FUNCTIONAL

Thanks!

REALITY GAMES

Michał Płachta
@miciek
www.michalplachta.com