- Building reactive real-time *data pipeline* at FPT

by @tantrieuf31

- **Basic info**
  - I work at FPT Telecom, tech lead of the team ***FA²*** (Rich Media Analytics and Advertising)
  - My focus about data infrastructure, backend system and reactive system architecture.
  - My blog at nguyentantrieu.info/blog
  - Founding a small Data Lab at mc2ads.org and has started R&D about Fast Data since 2014

Note:

- All contents and thoughts in this slide are my subjective ideas and work experience.
- The slide is written for meetup event at grokkingengineering.org

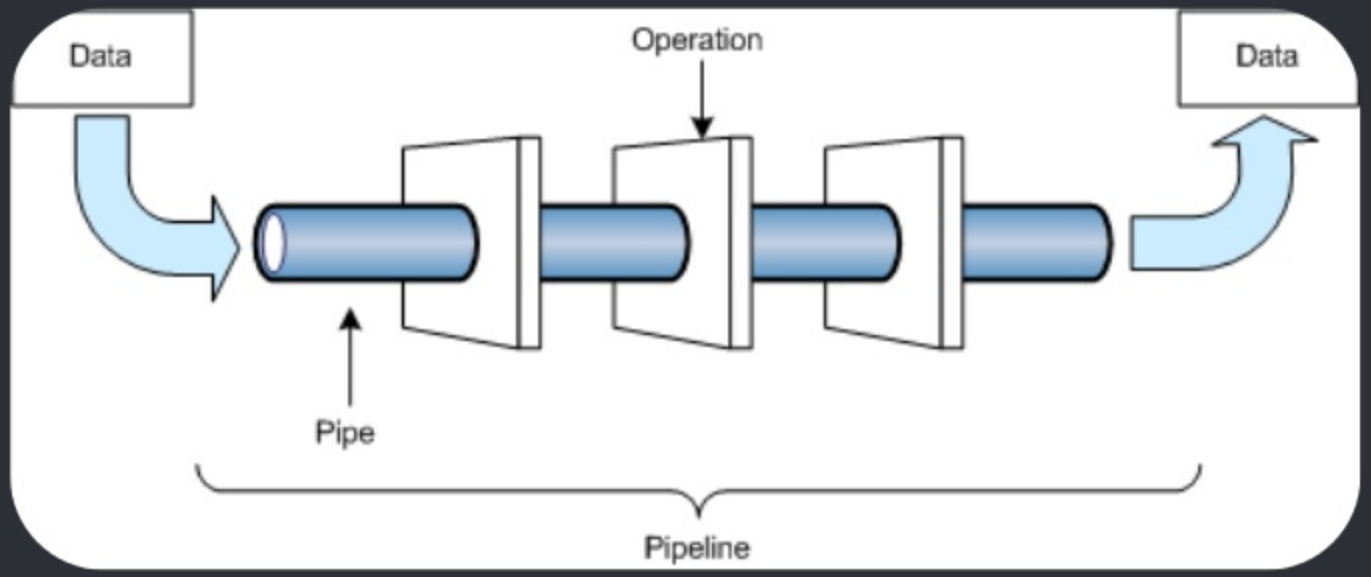
Contents

1. What is “Data Pipeline” ?
2. Big Data Problems at FPT
   a. VnExpress: pageview and heat-map
   b. eClick: real-time reactive advertising
3. Solutions and Patterns
4. Fast Data Architecture at FPT
5. Wrap up

1) What is “Data Pipeline” ?

“Intelligence is not only the ability to reason; it is also the ability to find relevant material in memory and to deploy attention when needed.”

— Daniel Kahneman, Thinking, Fast and Slow



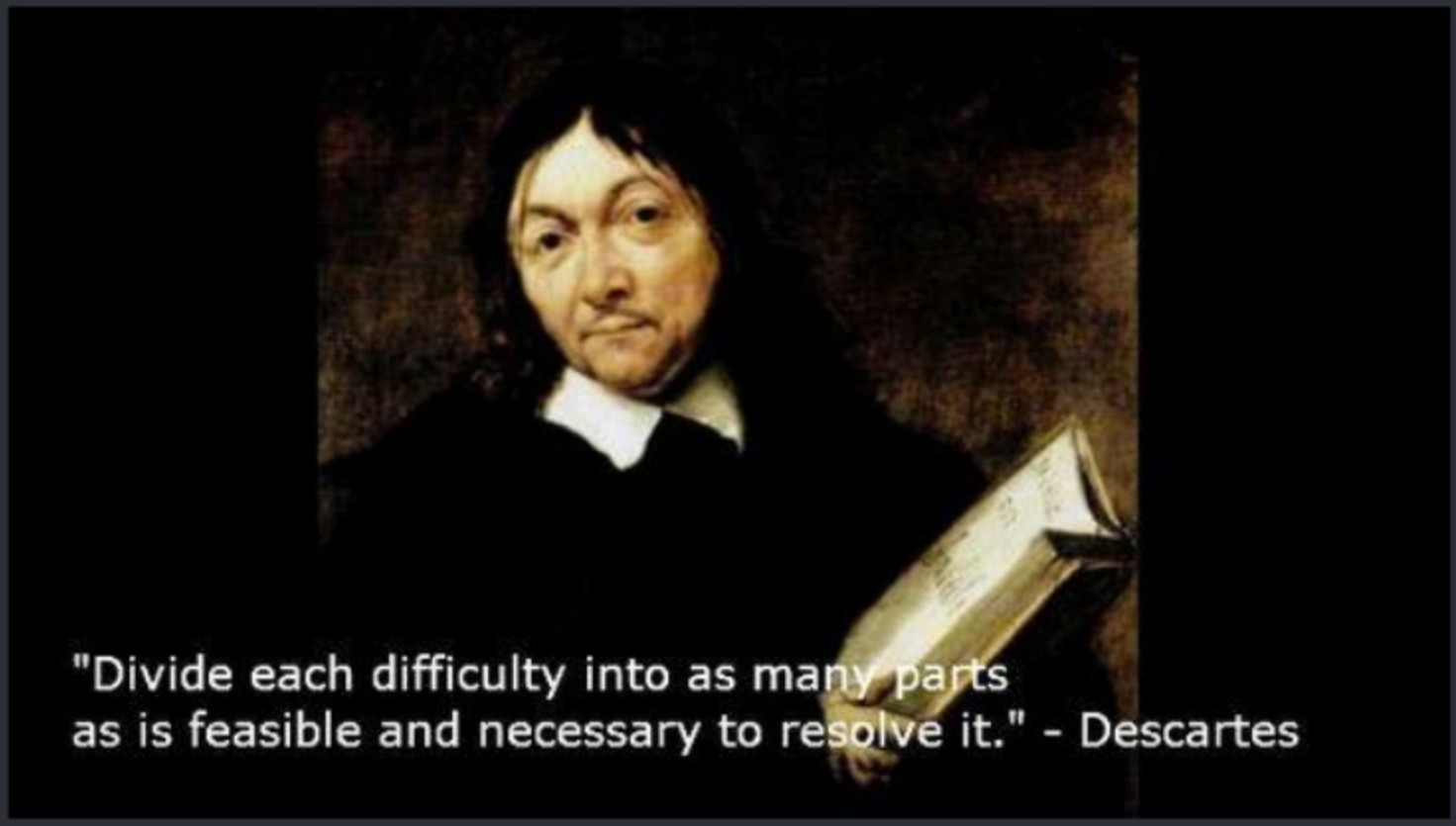
Data Pipeline

a set of data processing elements connected in series, where the output of one element is the input of the next one

2) Big Data Problems at FPT

2.1 VnExpress: scaling Analytics system

2.2 eClick: real-time data pipeline



about Big Data System at FPT Online

50 GB Log/day for real-time processing
300 GB Log/day for batch processing
processing nearly 5000 events in 1 second !

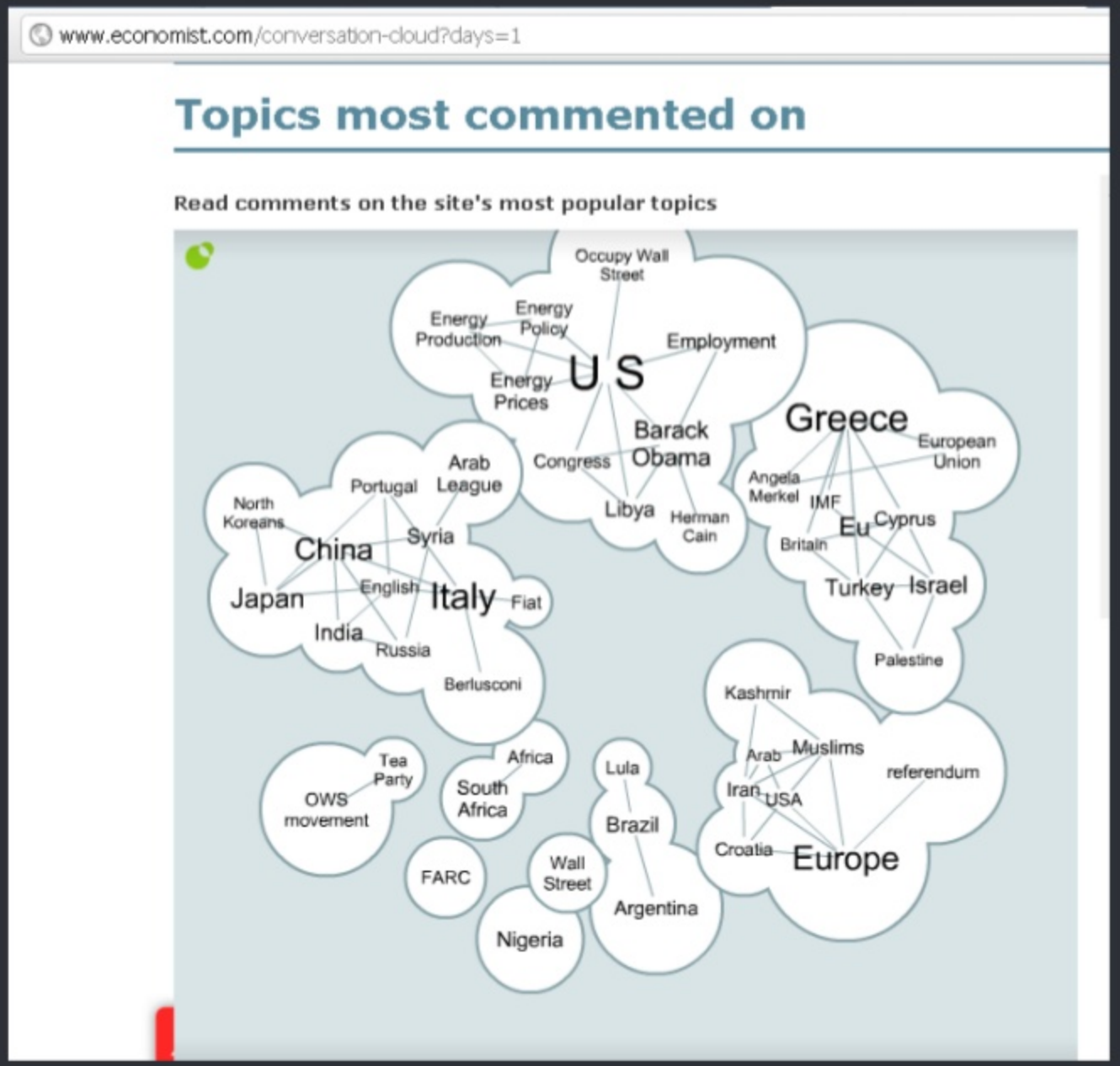
20 millions Vietnamese online user
that's the number of active Vietnamese user from around the World

less than 5 seconds for critical KPI
is the max latency from tracking to reacting !

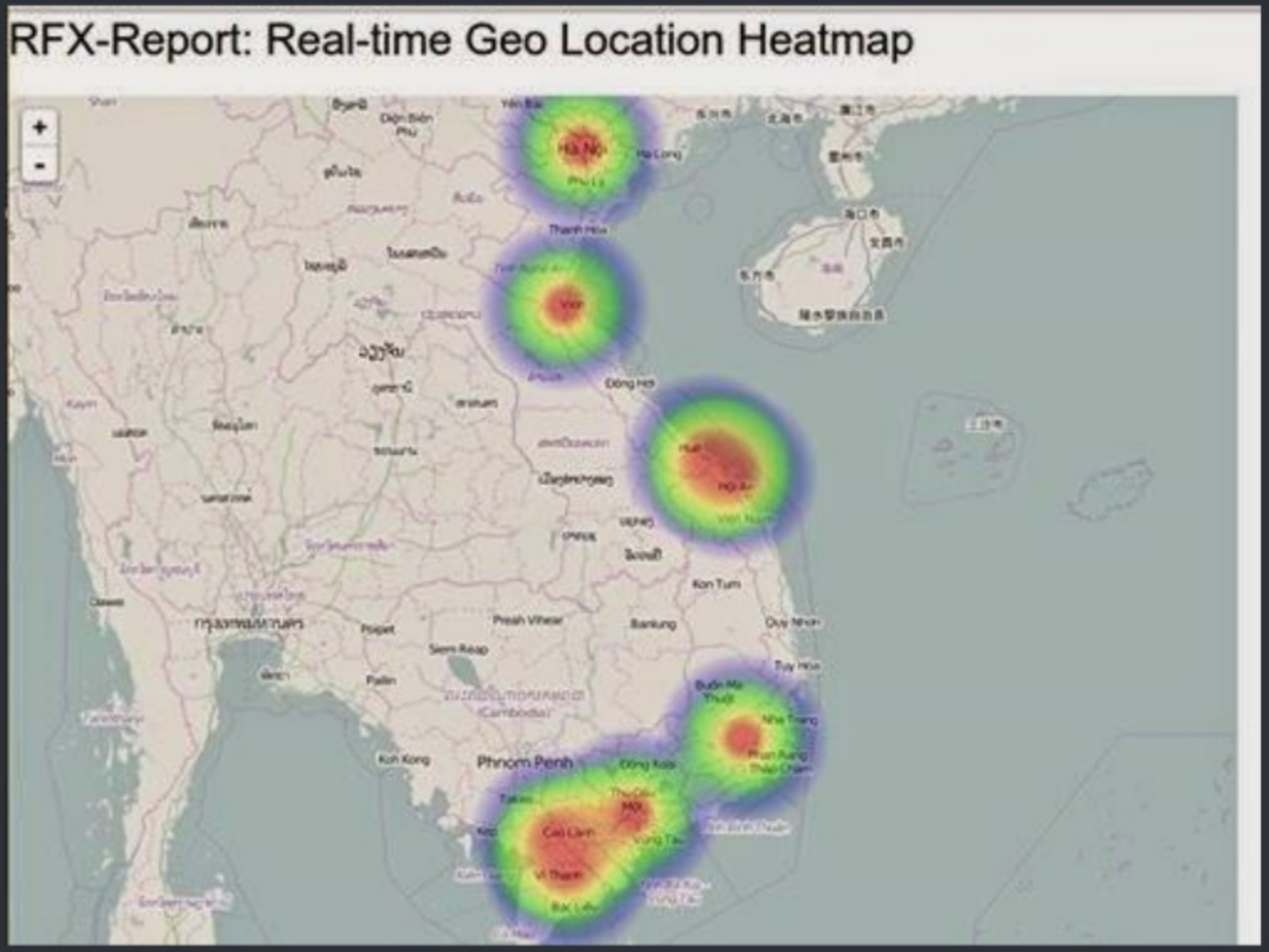
Problems at VnExpress

- Counting page-view for all articles
- Predicting the reach number for a specific topic, keyword, URL.
- Computing the social trending from Facebook
- Classifying positive/negative comments
- Viral score for journal topics
- Smarter content editor
- Real-time newsroom

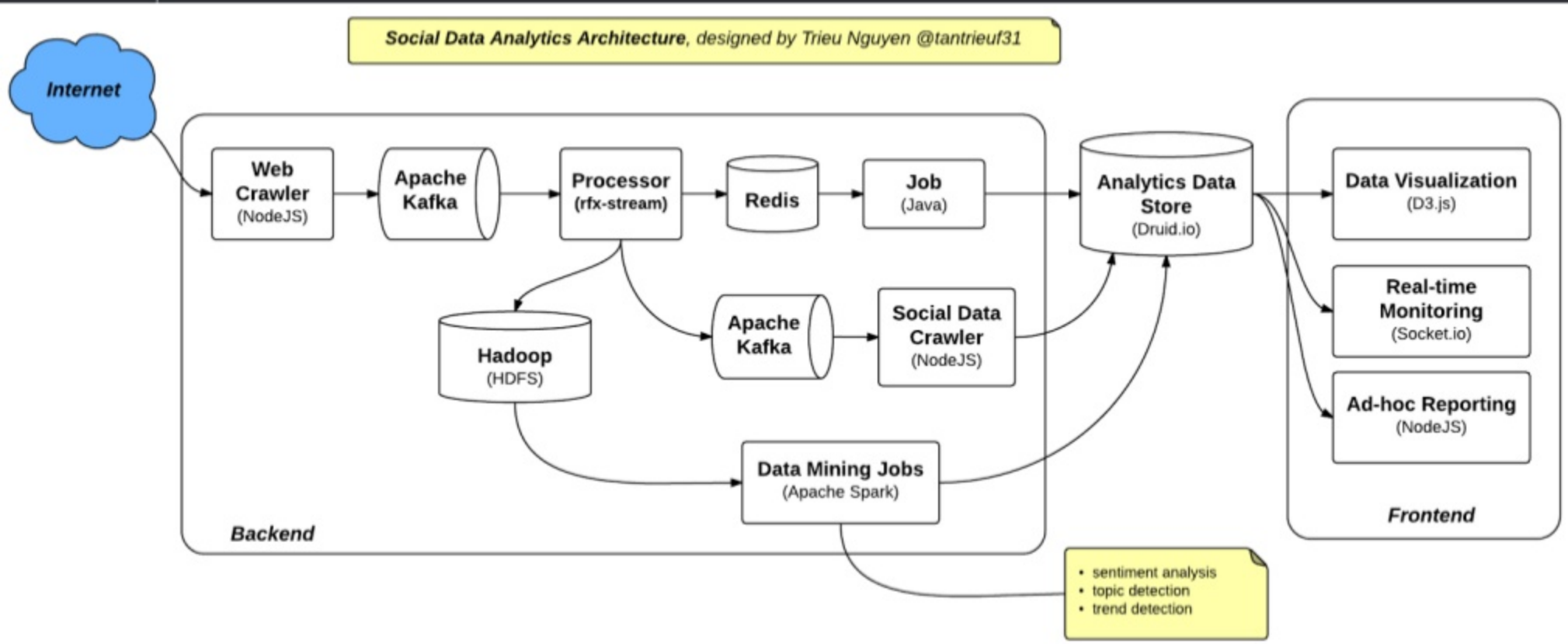
Problem: Which topics most commented on



- Problem: Measure the heatmap of User Activity



The architecture to Social Media Analytics

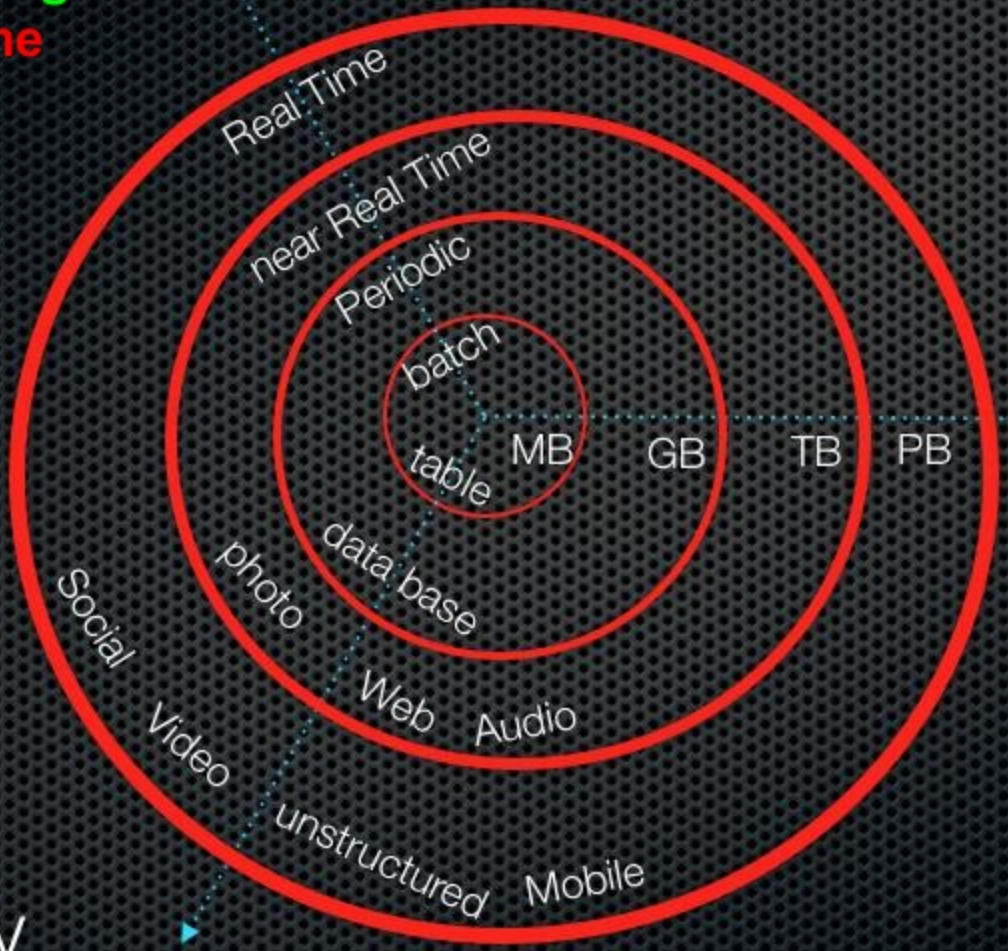


- **Challenges at eClick's backend system**
  - Should be **Faster**
  - Must be **Scalable**
  - Must be **"anti-fraud"**
  - Should be **Maintainable**
  - More **Agile** in Data Analytics
  - **Reactive** for important events in real-time
  - Connecting the Seller Side and Buyer Side better

Big Data: Expanding on 3 fronts at an increasing rate.

Data Velocity

at eClick we must check campaigns in near-real-time (seconds) !



Data Volume

at eClick we have ~50 GB Log for Stream ~300 GB Log for Batch just for tracking user actions (click, impression,...) in *ONE* day !

Data Variety

at eClick we have many types of log (video, web, mobile, ad-campaigns, articles, …)

- How problem was solved

Great things are done by a series of small things brought together.

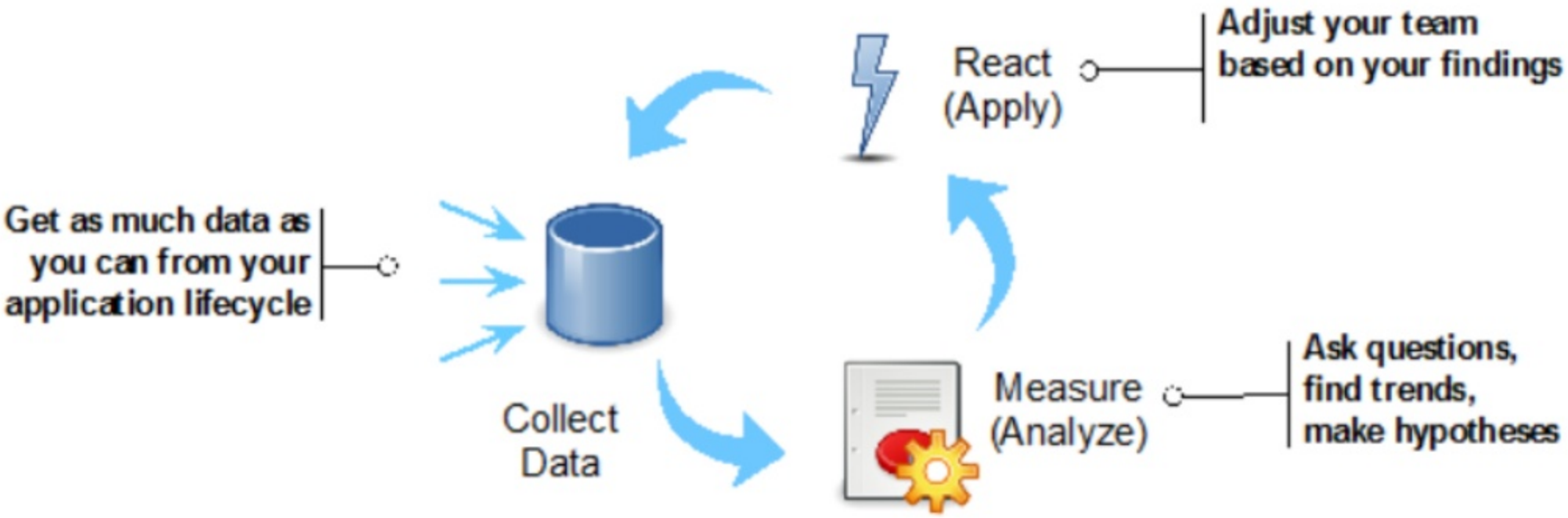
- Vincent van Gogh, Dutch painter (1853-1890)

Finding a framework

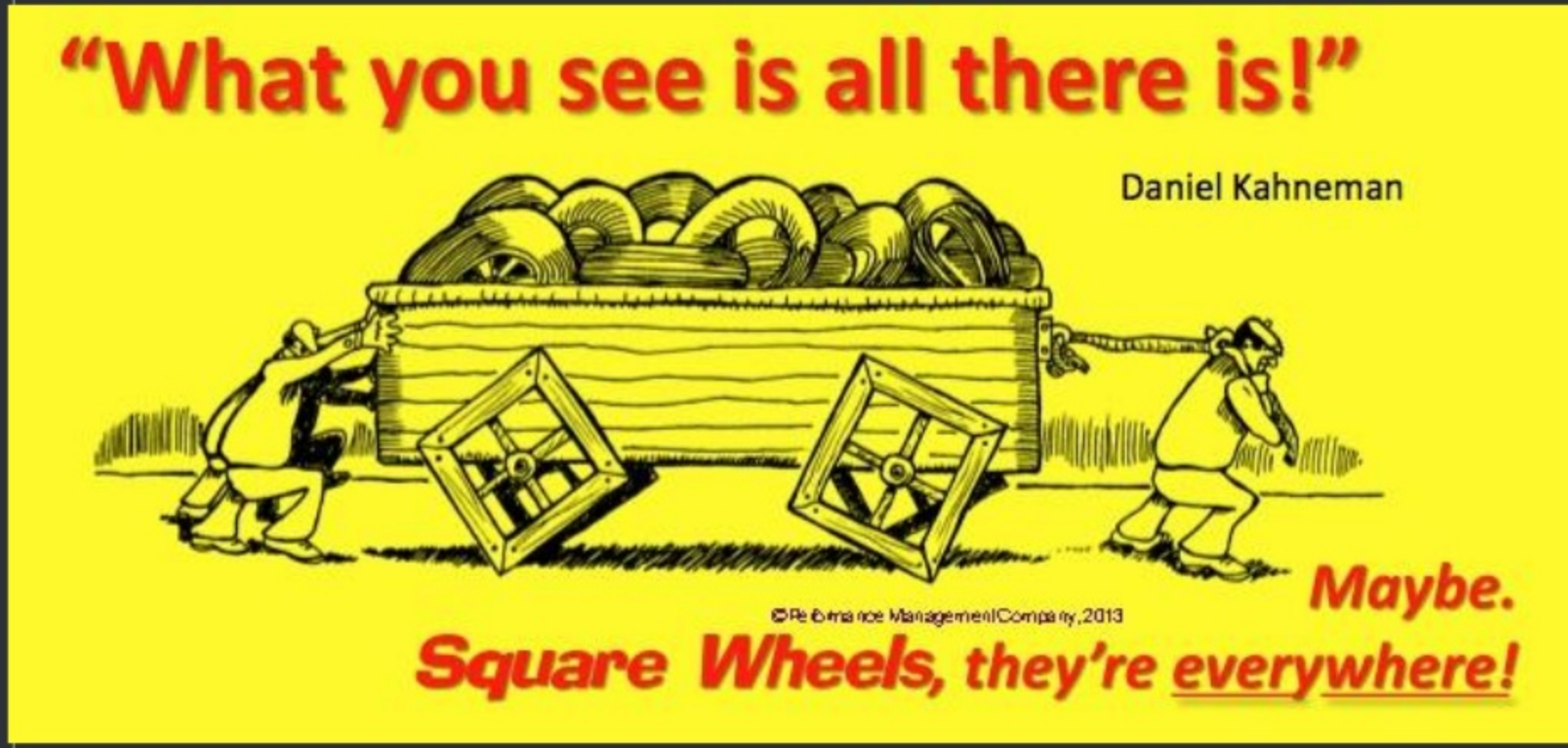
Solution	Developer	Type	Description
Storm	Twitter	Streaming	Twitter's new streaming big-data analytics solution
S4	Yahoo!	Streaming	Distributed stream computing platform from Yahoo!
Hadoop	Apache	Batch	First open source implementation of the MapReduce paradigm
Spark	UC Berkeley AMPLab	Batch	Recent analytics platform that supports in-memory data sets and resiliency
Disco	Nokia	Batch	Nokia's distributed MapReduce framework
HPCC	LexisNexis	Batch	HPC cluster for big data

in 2012, that's all I have ! And I chose the Storm

Learn & Build the system in agile way "Reacting to change quickly"

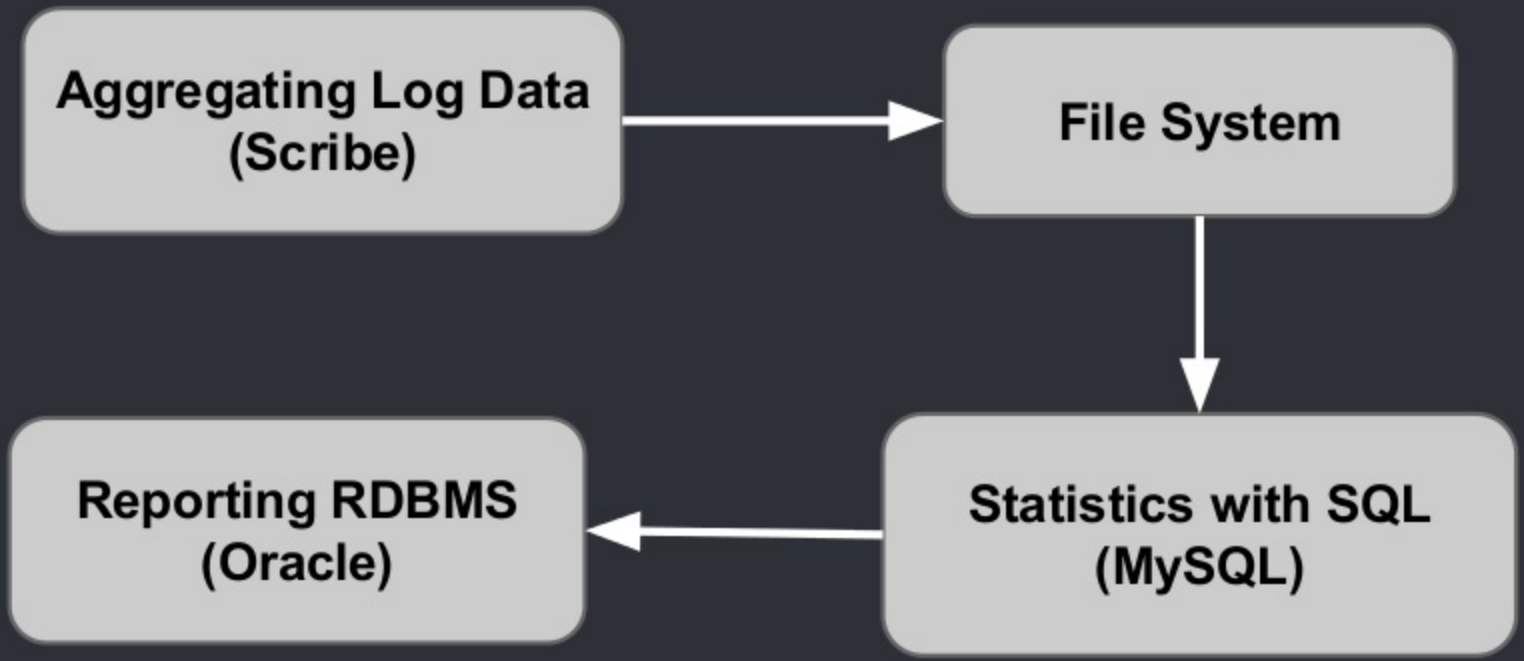


- Before implementing Lambda Architecture at FPT



That's the Batch Processing Architecture

Before implementing Lambda Architecture at FPT (Before May 1, 2013)

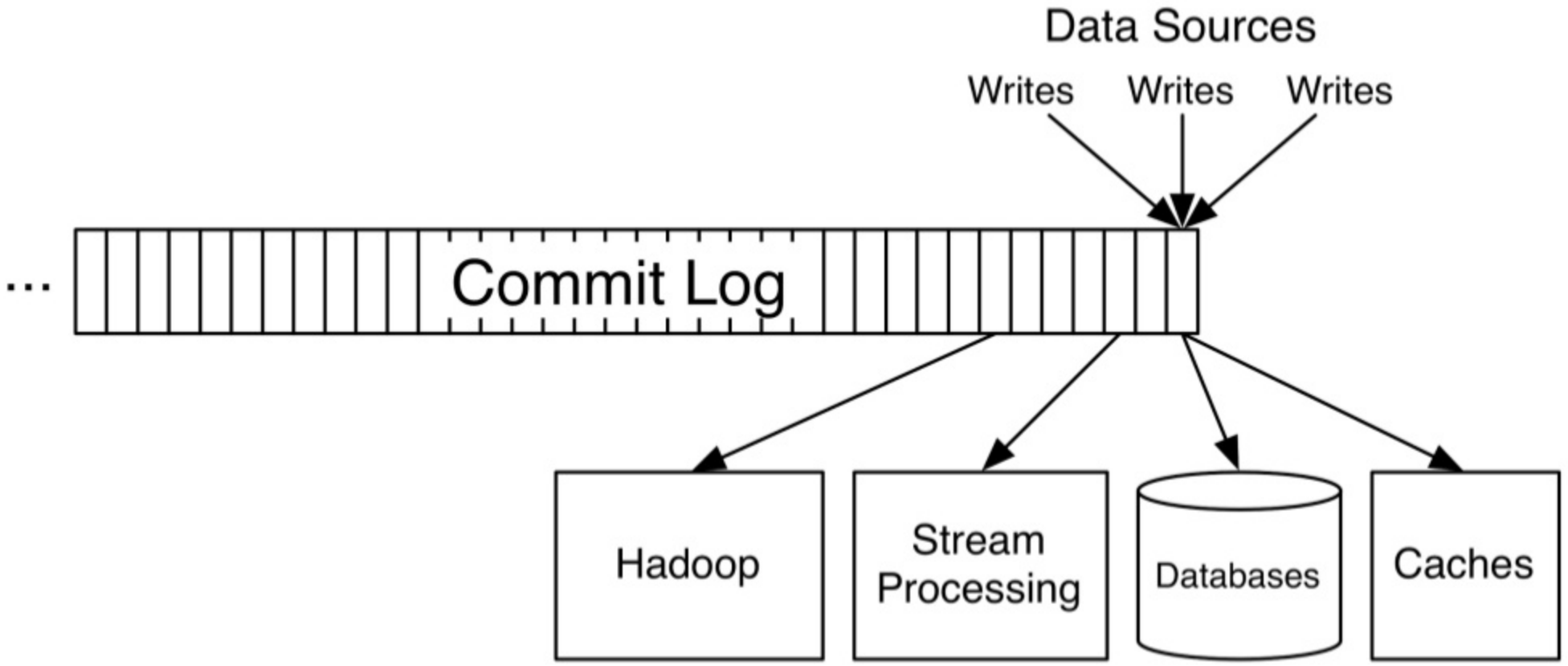


Problems with Batch Processing Architecture

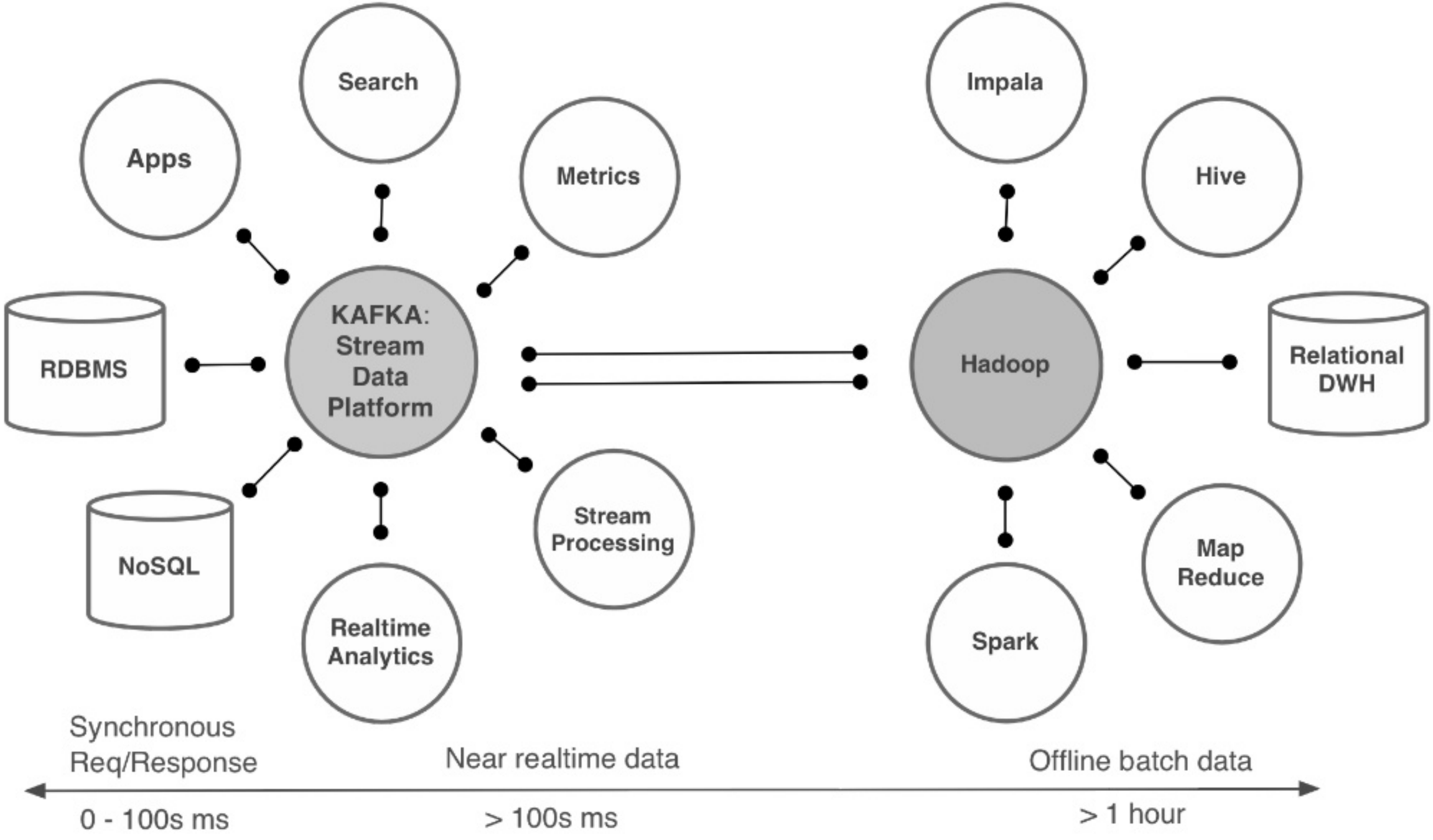
- Checking advertising campaign takes more than 5 minutes
- Hard to scale for 20 million users
- Not fault tolerant (by File System)
- More than 15 minutes for reporting (File → MySQL + Python → Oracle)

- Lambda Architecture 1.0 at FPT

Key idea: Adapting Apache Kafka for distributed messaging and event queue

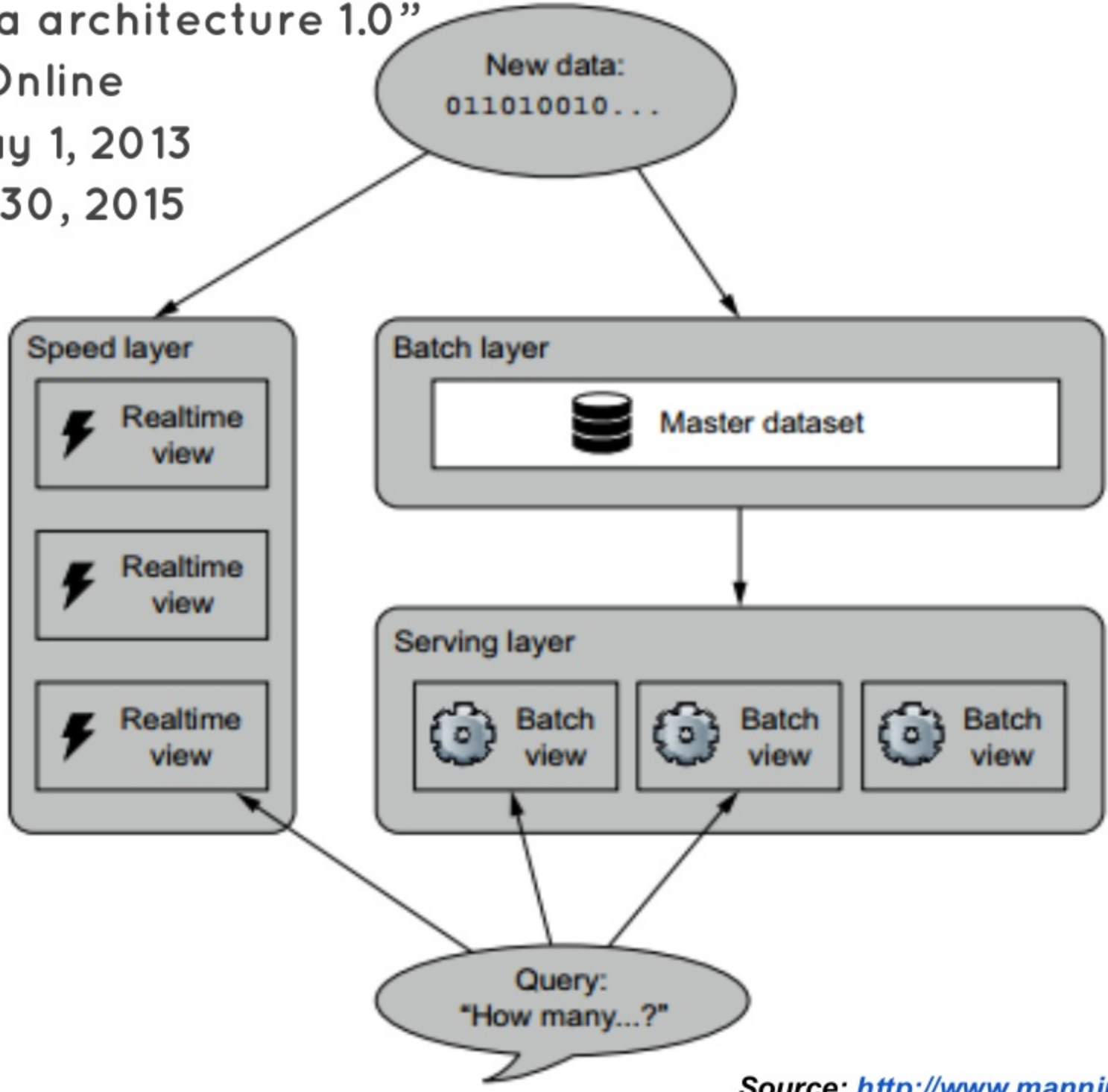


Key idea: data as streams



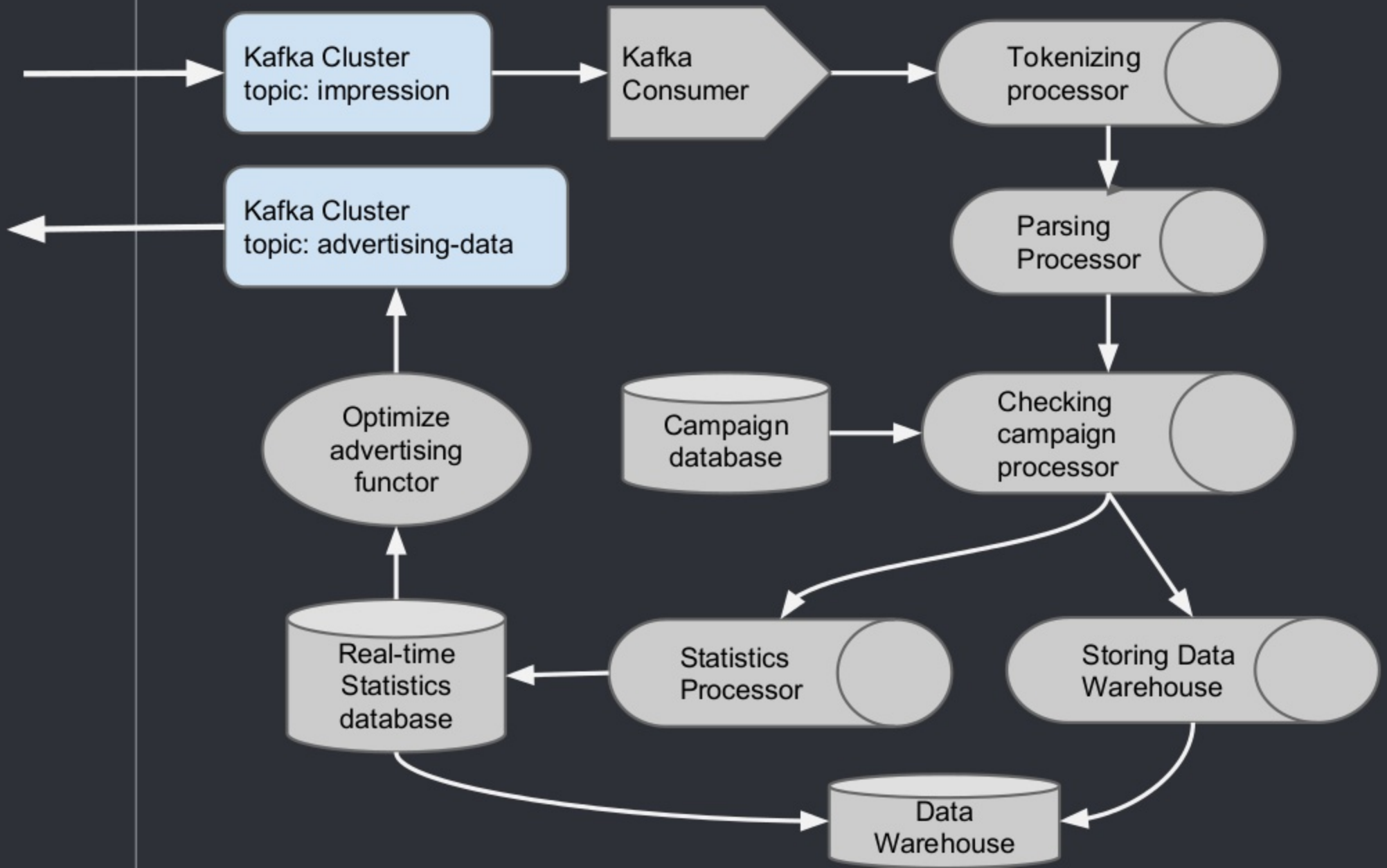
http://blog.confluent.io/2015/02/25/stream-data-platform-1/

"Lambda architecture 1.0"
at FPT Online
from May 1, 2013
to April 30, 2015



Source: http://www.manning.com/marz/

Processing Topology at eClick



Stream Processing System Dashboard

Monitor | Workers | System Events | Deployment | Settings | Topology | About

The list of reactive analytical workers, actions: **Refresh data** **Restart system**

Worker Info	Memory Usage	Topology Name	Kafka Topic	Partition Range	Kafka Brokers	Seeding Counter	Uptime	Topology Actors	Total Actors	Actions
[redacted]:14004 ALIVE	Used: 160.2 MB Limit: 1.9 GB	**TrueImpressionTopology**	**true-impression**	0-7	[redacted]:9081, :9082, :9083, :9084]	**2080023**	359:34:10	1. TokenizingFunctor=> {TrueImpressionParsingActor} 2. TrueImpressionParsingActor=> {TrueImpressionRawLogActor,CheckCpmActor} 3. CheckCpmActor=>{} 4. TrueImpressionRawLogActor=>{}	20000	setting restart
[redacted]:14006 ALIVE	Used: 285.5 MB Limit: 1.9 GB	**UrlPageviewTopology**	**eclick-pageview**	0-1	[redacted]:9091, :9092, :9093, :9094]	**93281441**	359:34:35	1. TokenizingFunctor=> {PageviewLogParsingActor} 2. PageviewLogParsingActor=>{}	10000	setting restart
[redacted]:14003 ALIVE	Used: 355.9 MB Limit: 1.9 GB	**PageviewImpressionTopology**	**pageview-impression**	8-15	[redacted]:9081, :9082, :9083, :9084]	**1508869**	359:33:55	1. TokenizingFunctor=> {PageviewImpressionParseActor} 2. PageviewImpressionParseActor=> {PageviewRawLogActor} 3. PageviewRawLogActor=>{}	15000	setting restart
[redacted]:14005 ALIVE	Used: 438 MB Limit: 1.9 GB	**TrueImpressionTopology**	**true-impression**	8-15	[redacted]:9081, :9082, :9083, :9084]	**1390051**	359:34:10	1. TokenizingFunctor=> {TrueImpressionParsingActor} 2. TrueImpressionParsingActor=> {TrueImpressionRawLogActor,CheckCpmActor} 3. CheckCpmActor=>{} 4. TrueImpressionRawLogActor=>{}	20000	setting restart
[redacted] ALIVE	Used: 78.3 MB Limit: 1.9 GB	**PageviewImpressionTopology**	**pageview-impression**	0-7	[redacted]:9081, :9002, :9083, :9084]	**2258310**	359:34:0	1. TokenizingFunctor=> {PageviewImpressionParseActor} 2. PageviewImpressionParseActor=> {PageviewRawLogActor} 3. PageviewRawLogActor=>{}	15000	setting restart
[redacted]:14001 ALIVE	Used: 424.8 MB Limit: 1.9 GB	**ClickTopology**	**click**	0-7	[redacted]:9081, :9082, :9083, :9084]	**12244**	359:34:15	1. TokenizingFunctor=>{ParsingClickLogActorV3} 2. ParsingClickLogActorV3=> {CheckAndSaveClickActor} 3. CheckAndSaveClickActor=>{}	15000	setting restart

Improvements after applying Lambda Architecture version 1

- Checking advertising campaign in less than 5 seconds (near real-time)
- Scalability for 20 million users
- Fault tolerance (thank to Kafka)
- Take less than 3 minutes for reporting (Kafka → Redis → Oracle)
- Take less than 1 hour for Data Mining Jobs (with HBase and Java code)

Analytics' Dashboard

Analytics' Dashboard > Overview

True Impression

Dữ liệu hiện tại: **09:14:23** - True Impression: **16,544,199,471** - Lượt click: **6,731,462** - CTR: **0.0004%**

Hiện tại

1,383,783

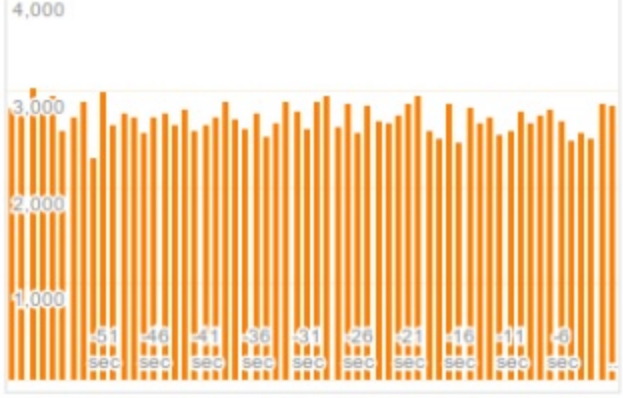
người đang truy cập

Người dùng mới
Người dùng cũ

70% 30%

True Impression

True Impression



Lambda Architecture 2.0 at FPT

Make everything as simple as possible, but not simpler.

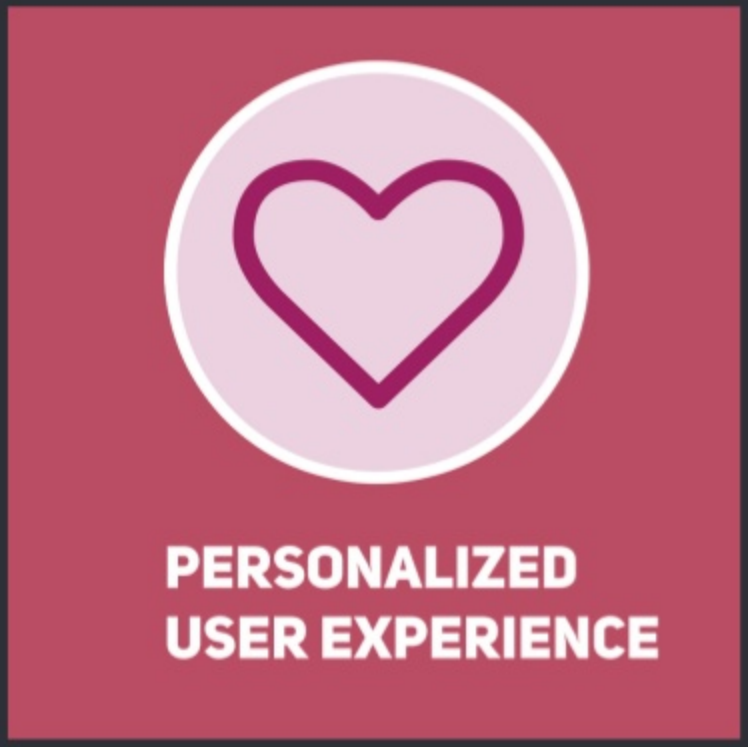
In version 2, we want to react to the World faster

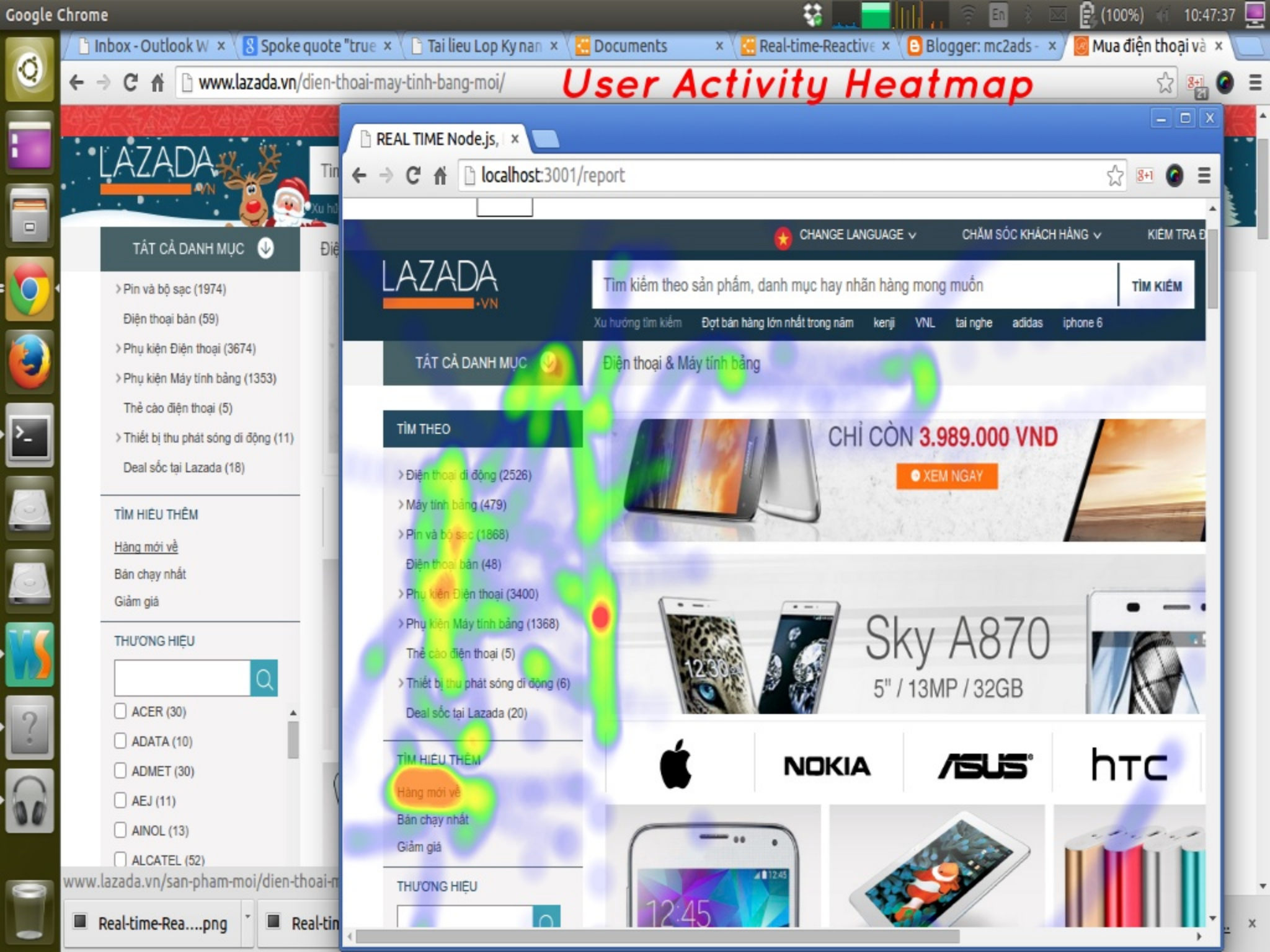
Why is λ 2.0?

Because λ 1.0 does not deliver

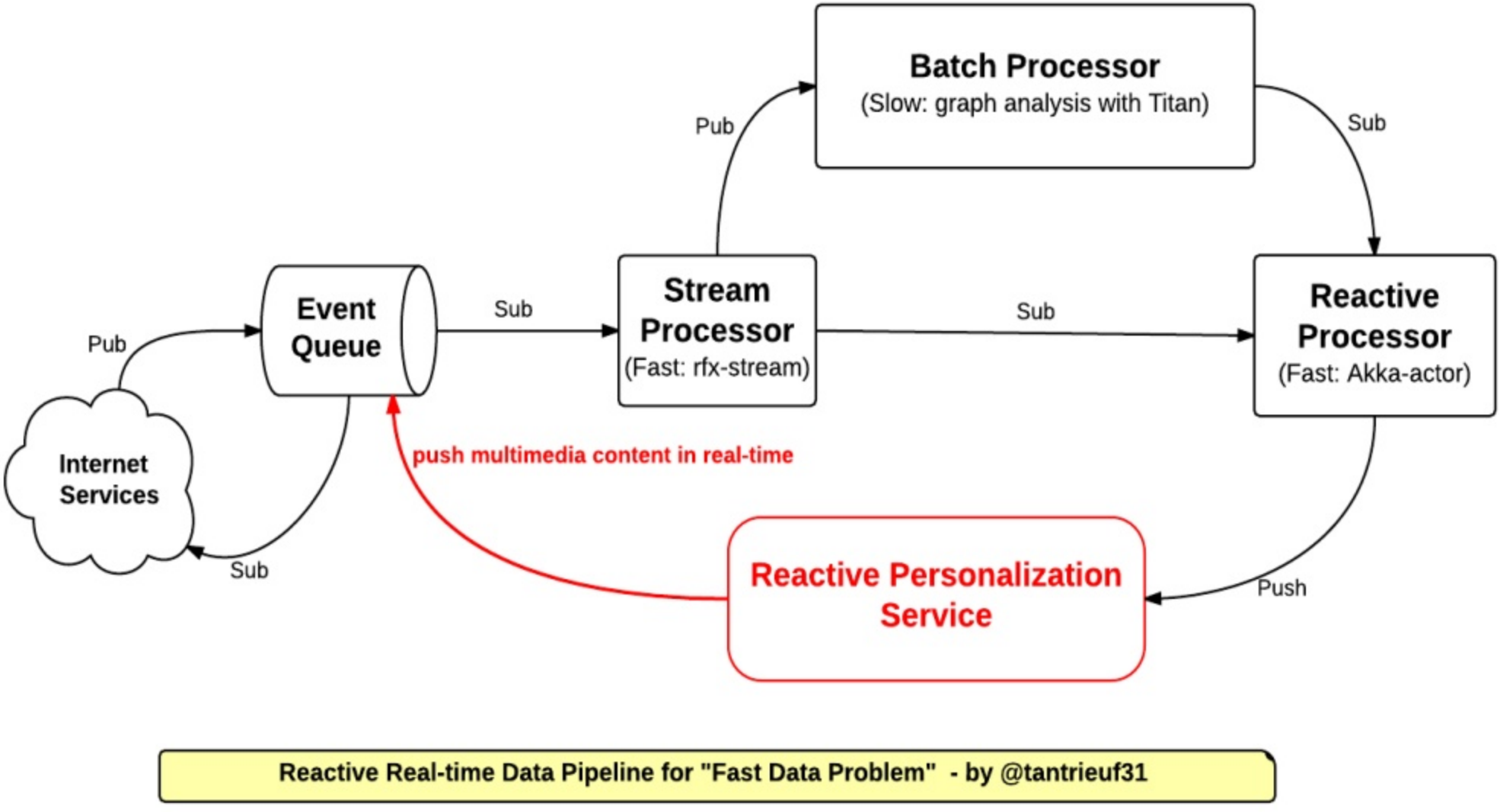
"Fast Personalized UX" for Agile Business

→ *Reactive* Advertising for Digital Media

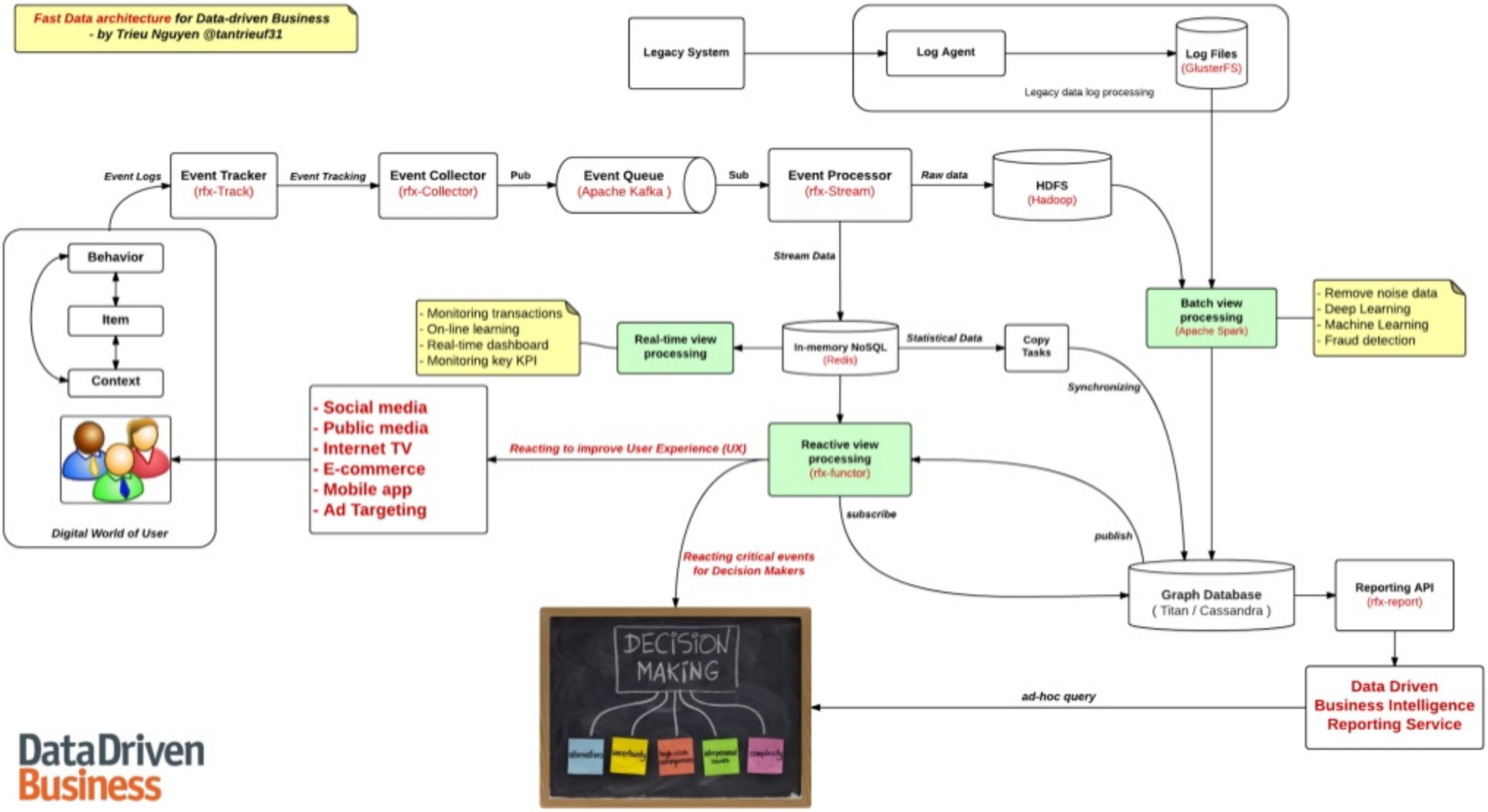



Google Chrome
User Activity Heatmap
REAL TIME Node.js,
localhost:3001/report
www.lazada.vn/dien-thoai-may-tinh-bang-moi/
LAZADA
CHANGE LANGUAGE
CHĂM SÓC KHÁCH HÀNG
Tìm kiếm theo sản phẩm, danh mục hay nhãn hàng mong muốn
TÌM KIẾM
TẤT CẢ DANH MỤC
Điện thoại & Máy tính bảng
TÌM THEO
CHỈ CÒN 3.989.000 VND
XEM NGAY
Sky A870
5" / 13MP / 32GB
NOKIA
ASUS
htc
TÌM HIỂU THÊM
Hàng mới về
Bán chạy nhất
Giảm giá
THƯƠNG HIỆU

- **Improvements in Lambda Architecture version 2**
  - Fast Processing for Data Mining / Learning
  - Fast Agile for Data Science
  - Fast Reactive for critical event in real-time
  - Fast Connecting from demand to supply
  - Fast Sharing information among the "Actor"

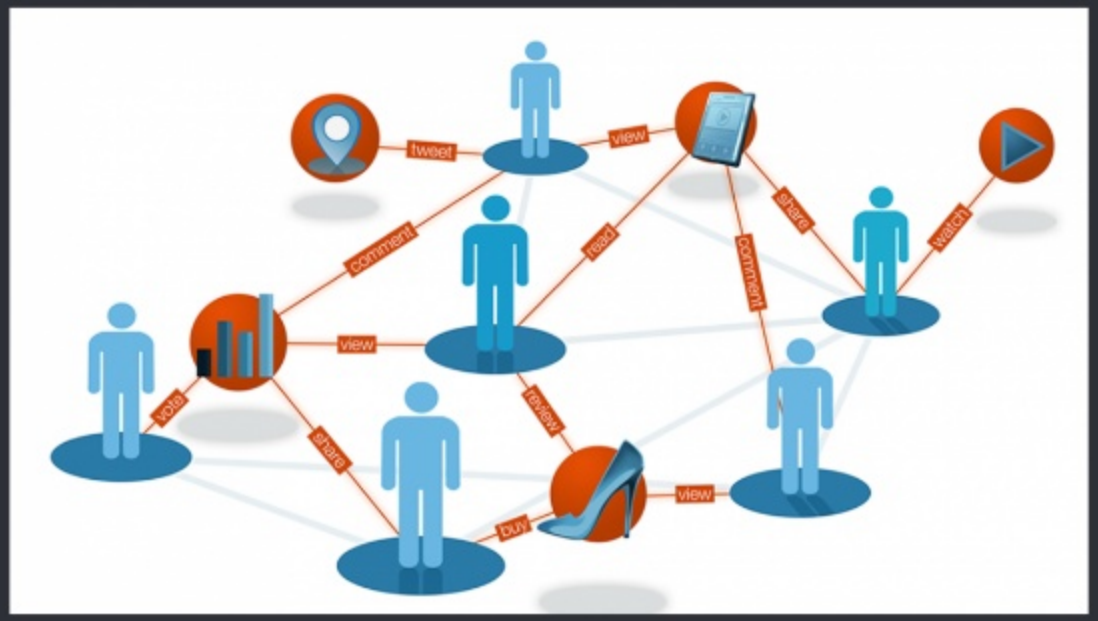

Batch Processor
(Slow: graph analysis with Titan)
Pub
Sub
Event Queue
Sub
Stream Processor
(Fast: rfx-stream)
Sub
Reactive Processor
(Fast: Akka-actor)
Pub
Internet Services
Sub
push multimedia content in real-time
Reactive Personalization Service
Push
Reactive Real-time Data Pipeline for "Fast Data Problem" - by @tantrieuf31

"Lambda Architecture 2.0"
will be "In Production" after June 1, 2015



DataDriven Business

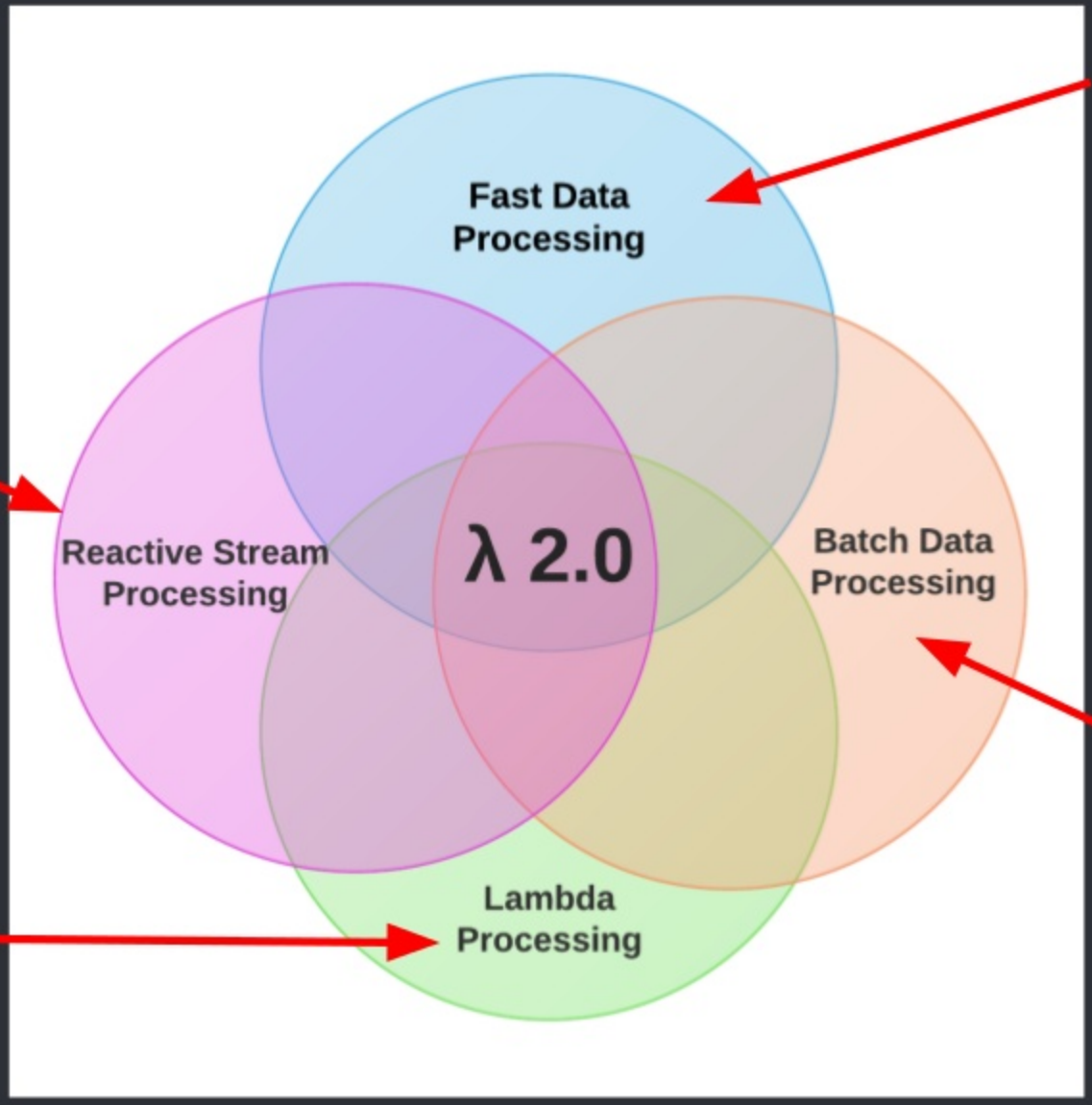
Problem: tracking, crawling and measuring social trends and user behaviors from social media




```
14 /**
15  * tracking user activity and finding social trends (keywords, topics) in real-time
16  * @author trieu
17  */
18 public class TrackingUserTopology extends BaseTopology  {
19     private static final String TOPIC_SOCIAL_ACTIVITY = "social-activity";
20
21     @Override
22     public BaseTopology buildTopology(){
23         return Pipeline.create(this)
24                 .apply(HttpEventLogTokenizing.class).apply(ParsingLog.class)
25                 .applyFromXtoY(ParsingLog.class, FindingSocialTrends.class )
26                 .applyFromXtoY(ParsingLog.class , FindingKeywords.class)
27                 .joinAndApply(FindingSocialTrends.class , FindingKeywords.class, AlertingUserAboutHotContents.class )
28                 .done();
29     }
30
```

4) Fast Data Architecture at FPT

Built with Lambda Architecture 2.0



for observable data

reacting to critical event faster

for data mining and machine learning on Big Data

SQL querying (SQL is true lambda language !?)

We use new CONCEPTS for abstracting all things in system

Data Actor

Is the source for all data

Data Pipeline

just special data queue

Data Event

is what's happened in specific context, emitted by data actor

Processing Topology

is a directed graph of event processors

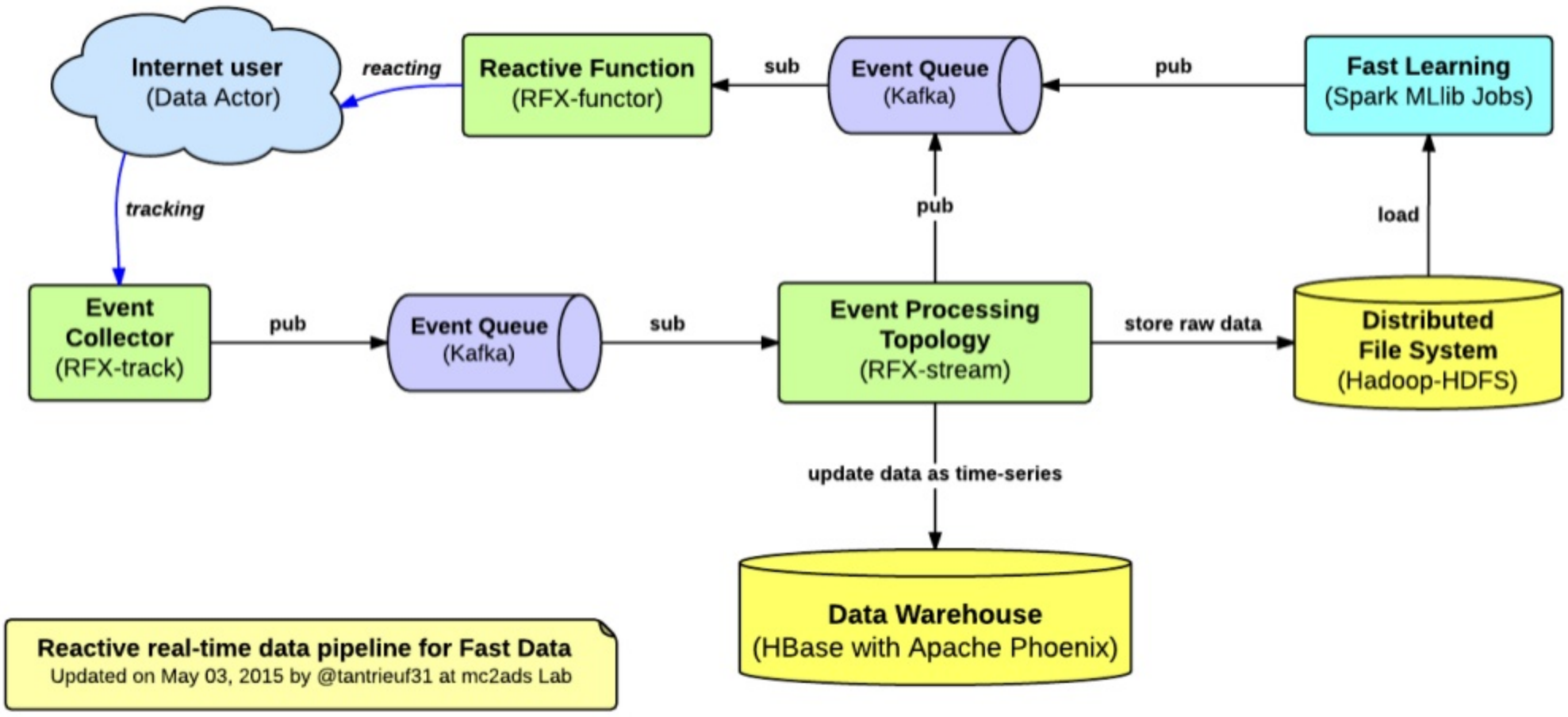
Event processor

is the logical processor for data pipeline

Reactive functor

is the is a functional actor that responds (reacts) to external data events

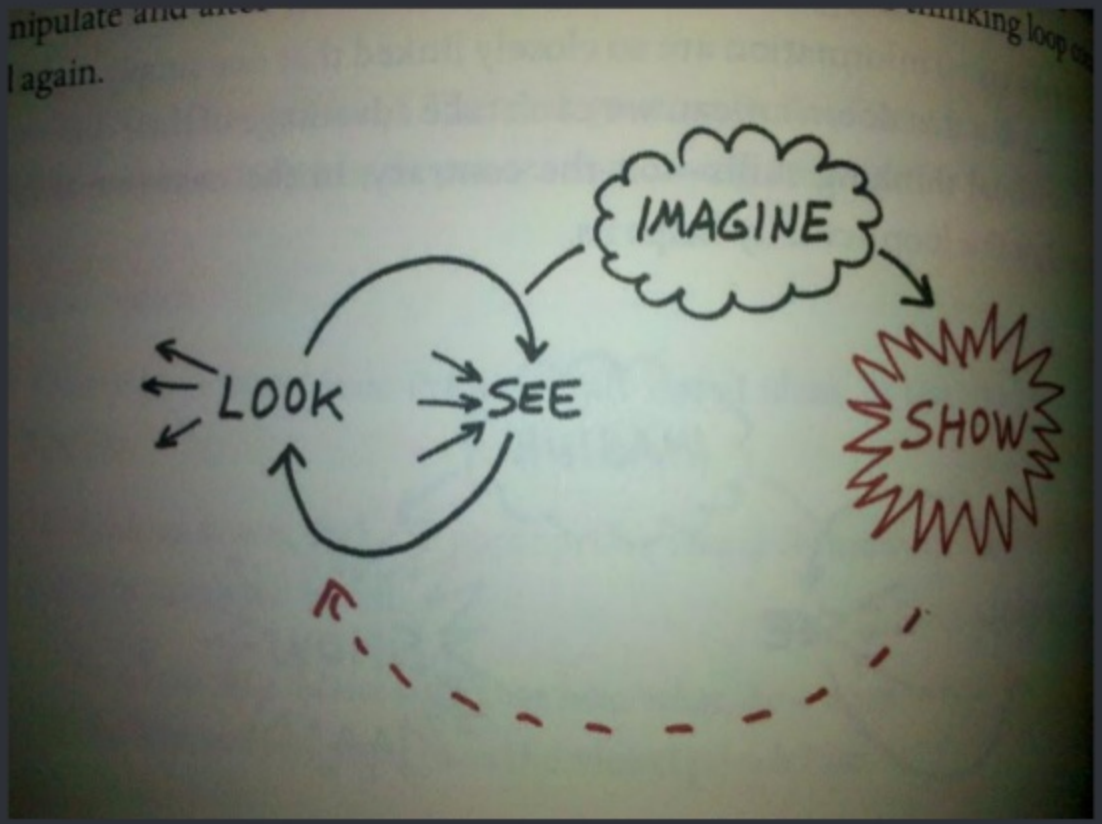
The architecture for Fast Data: Reactive real-time data pipeline



Technology stack (5D model)

1) **Data collector** (I/O networking)
   - Netty for event collector and HTTP server (rfx-track)
2) **Data persistence** (aka: data storage)
   - Kafka for distributed message queue(Apache Kafka)
   - HBase for scalable big table
3) **Data processing**
   - RFX with stream processor (RFX framework)
   - Apache Spark for in-memory batch processing
   - RxJava + Akka for reactive processor (reactivex)
4) **Data analysis**
   - measures of uncertainty
     - Python Dempster-Shafer theory → **anti-fraud**
5) **Data reporting**
   - SQL with Phoenix (http://phoenix.apache.org)
   - real-time frontend report: NodeJs, SocketIO

5) What we have learned



1) Understanding problems
2) Pick right framework
3) ***Build***
4) ***Test***
5) ***Measure***
6) ***Learn***
7) Go back to step 3

It's time to solve *practical problem "Lambda for Vacuum Cleaner"*

1) push the "Hello" message into vacuum cleaner
2) take it out !
3) display message "HELLO" in readable and ordered format?

Question 1: Which *data structure* could be used to implement these tasks ?

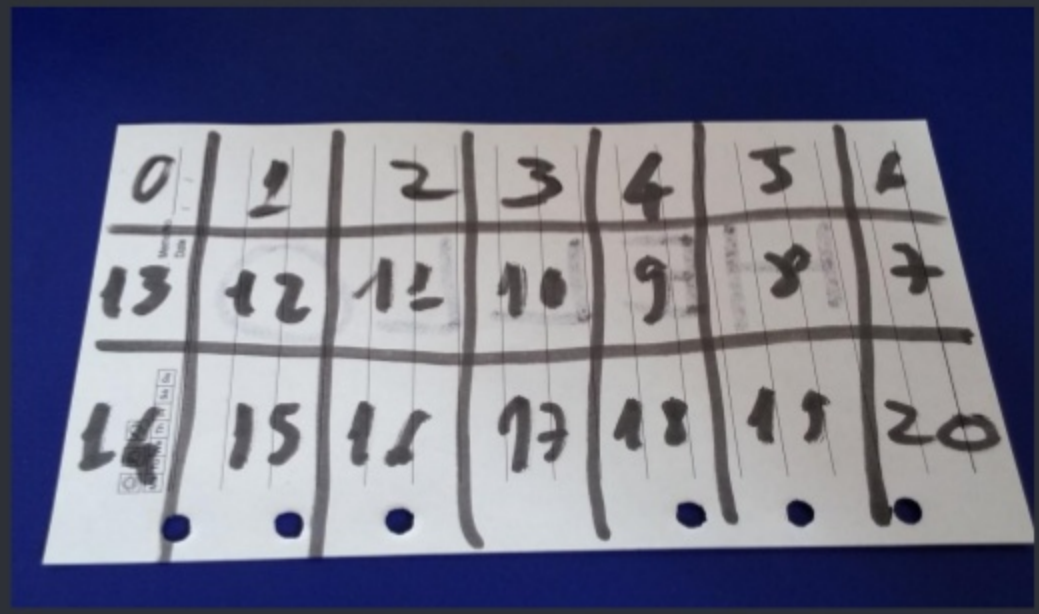
Question 2: Can you solve it using a functional programming language?

Question 3: Write code to prove your solution

Solution code for problem "Lambda for Vacuum Cleaner"

```
5   case class DataEvent(c: Char, i: Int) {
6     var _c: Char = c
7     var _i: Int = i
8     override def toString = "%s at %d".format(_c, _i)
9     def getData = "%s".format(_c)
10  }
11
12  def toElement(c: Char, i: Int): DataEvent = {
13    return new DataEvent(c,i)
14  }
15
16  def pushIntoVacuumCleaner(elements: Array[DataEvent]): Seq[DataEvent] = {
17    var stream = util.Random.shuffle(elements.toSeq)
18    // blah blah tasks will be done
19    return stream;
20  }
21
22  def main(args: Array[String]) {
23    //this is a paper we have
24    val paper = "*****HELLO******"
25
26    //transform into a set of elements
27    val paperSet = paper.toCharArray()
28    val elements = Array.tabulate(paper.length) { i => toElement(paperSet(i), i) }
29
30    //treat data as stream
31    val stream = pushIntoVacuumCleaner(elements)
32    println( stream.foldLeft("")((x,y) => x + "\n" + y  ) )
33
34    //print it out as unordered list
35    println( stream.map { x:DataEvent => x.getData  } )
36
37    //sorting the list and print it out as ordered list
38    val newStream = stream.sortWith(_._i < _._i)
39    println( newStream.map { x:DataEvent => x.getData  } )
40  }
```

```
L at 8
* at 15
H at 5
* at 10
* at 11
E at 6
* at 12
* at 2
* at 14
O at 9
* at 3
* at 4
* at 1
L at 7
* at 13
* at 0
RAW message: *****HELLO******
After take out from VacuumCleaner: ArrayBuffer(L, *, H, *, *, E, *, *, *, O, *, *, *, L, *, *)
Sorted data event: ArrayBuffer(*, *, *, *, *, H, E, L, L, O, *, *, *, *, *, *)
```



"In a concurrent world, imperative is the wrong default !"
- Tim Sweeney, Epic Games

Hey, back-end engineers and
front-end engineers

Want to join team FA^2 at FPT?
Please submit your CV to
tantrieuf31@ gmail.com