Processamento de Dados em Tempo Real

Jorge Cavalcanti Barbosa Fonsêca
(jorge.fonseca@upe.br)

Processamento de Dados na IoT

Jorge Cavalcanti Barbosa Fonsêca
(jorge.fonseca@upe.br)

UNIVERSIDADE DE PERNAMBUCO

Big IoT

Jorge Cavalcanti Barbosa Fonsêca
(jorge.fonseca@upe.br)

Processamento de Dados em Tempo Real

Jorge Cavalcanti Barbosa Fonsêca
(jorge.fonseca@upe.br)

Quem sou eu?

Jorge Fonsêca

- Professor UPE
- Filho do CIn
(Graduação, Mestrado, Doutorado)
- Criado no CESAR (10 anos)

TV Digital/Smart TV, Mobile, Web (anos atrás), SD
Requirement Analyst, Developer, Tester, Technical Coordinator, Team Leader
Recrutador ☺

Quem são vocês?

© Can Stock Photo - csp12854623

O que vocês esperam desse módulo?

Agenda

Fim de semana 1

- ✓ Sistemas Distribuídos
  - ✓ conceitos principais
  - ✓ *Middleware (Prática)*
- ✓ Data Stream
  - ✓ Teoria
  - ✓ Prática (*Storm local*)
  - ✓ Zookeeper

Fim de semana 2

- ✓ Data Stream (Storm)
  - ✓ *Acknowledgment*
  - ✓ *Window*
  - ✓ *Storm remoto/Cluster*
- ✓ CEP
  - ✓ Teoria e prática

Agenda…

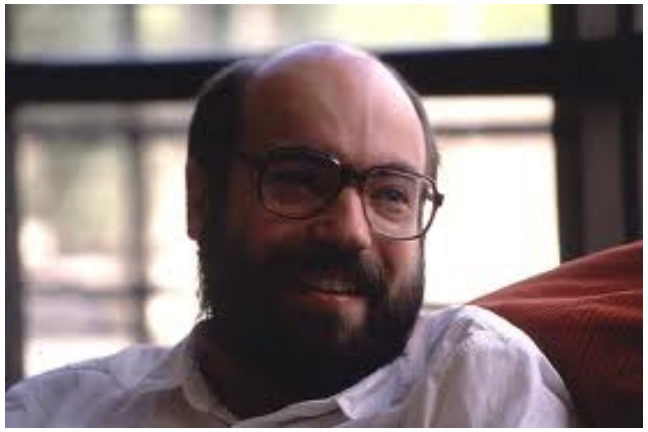
hands on

Agenda...

Introdução / Contextualização

Computação Ubíqua

"*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.*"



(Mark Weiser – The Computer for the 21st Century)

Computação Ubíqua

09-91 SCI AMER WEISER *** 1

Scientific American Ubicomp Paper after Sci Am editing

one more final edit from me to go

The Computer for the 21st Century

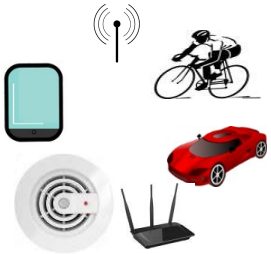
Mark Weiser

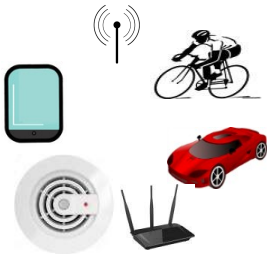
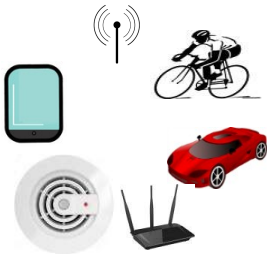
The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.

Consider writing, perhaps the first information technology: The ability to capture a symbolic representation of spoken language for long-term storage freed information from the limits of individual memory. Today this technology is ubiquitous in industrialized countries. Not only do books, magazines and newspapers convey written information, but so do street signs, billboards, shop signs and even graffiti. Candy wrappers are covered in writing. The constant background presence of these products of "literacy technology" does not require active attention, but the information to be conveyed is ready for use at a glance. It is difficult to imagine modern life otherwise.

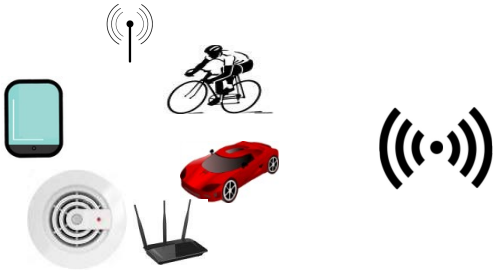
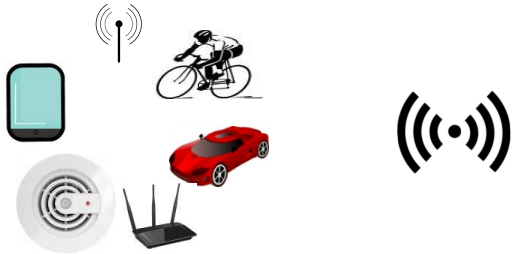
Computação Ubíqua

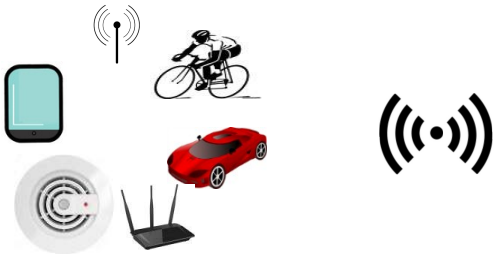
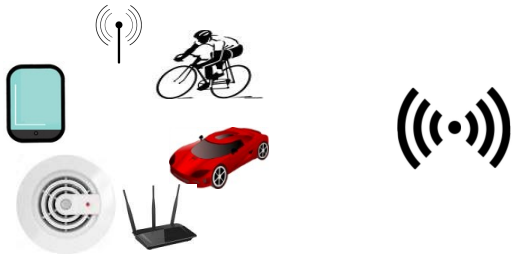
Objetos Físicos → Objetos Eletrônicos

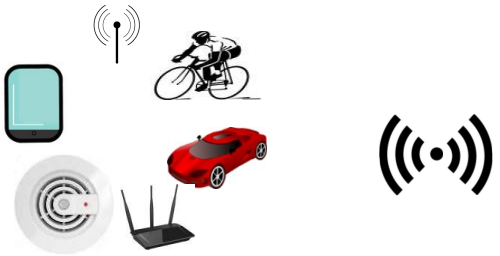
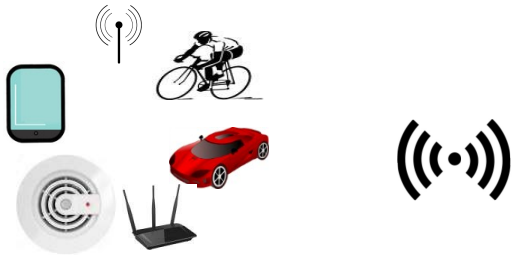



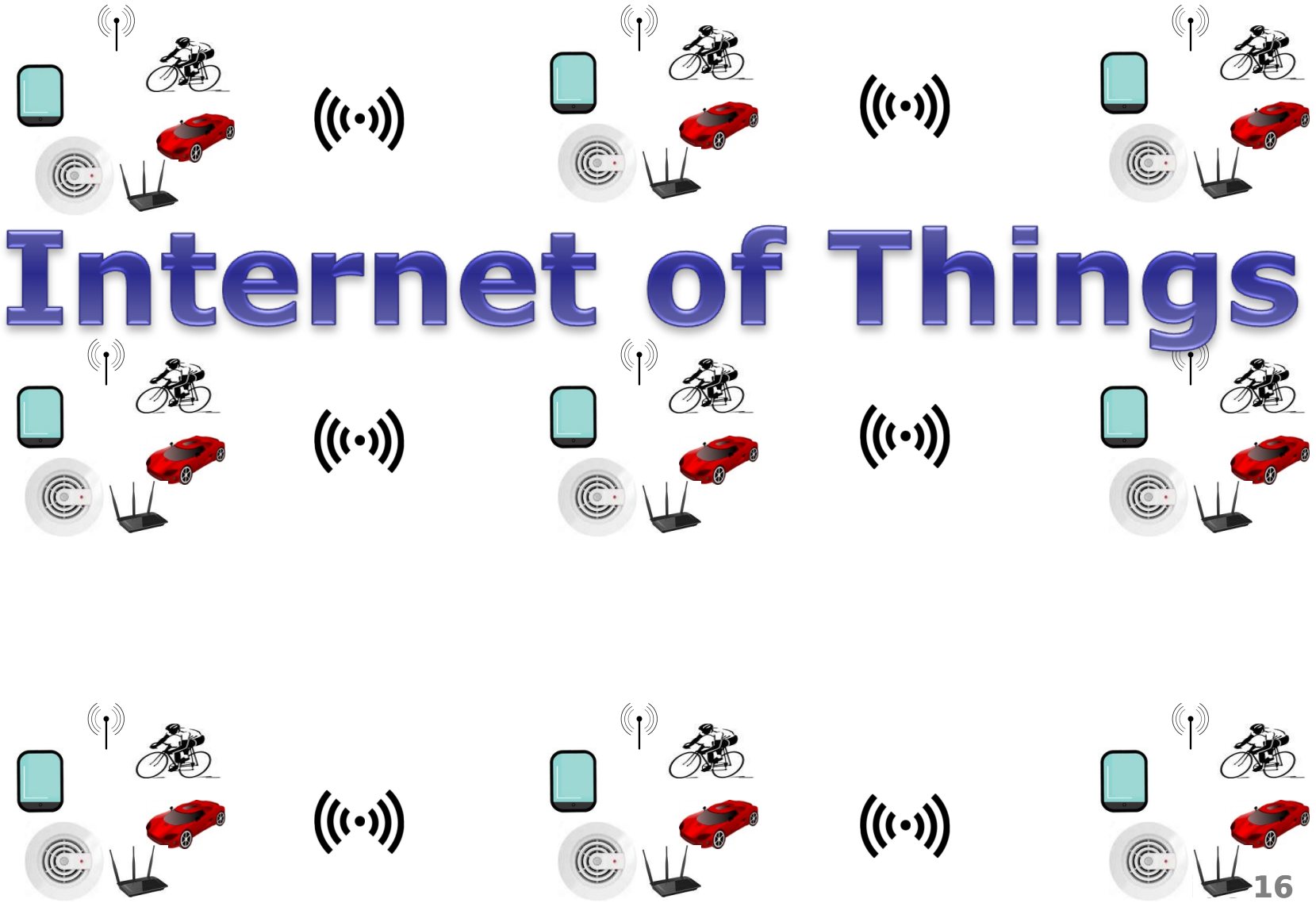





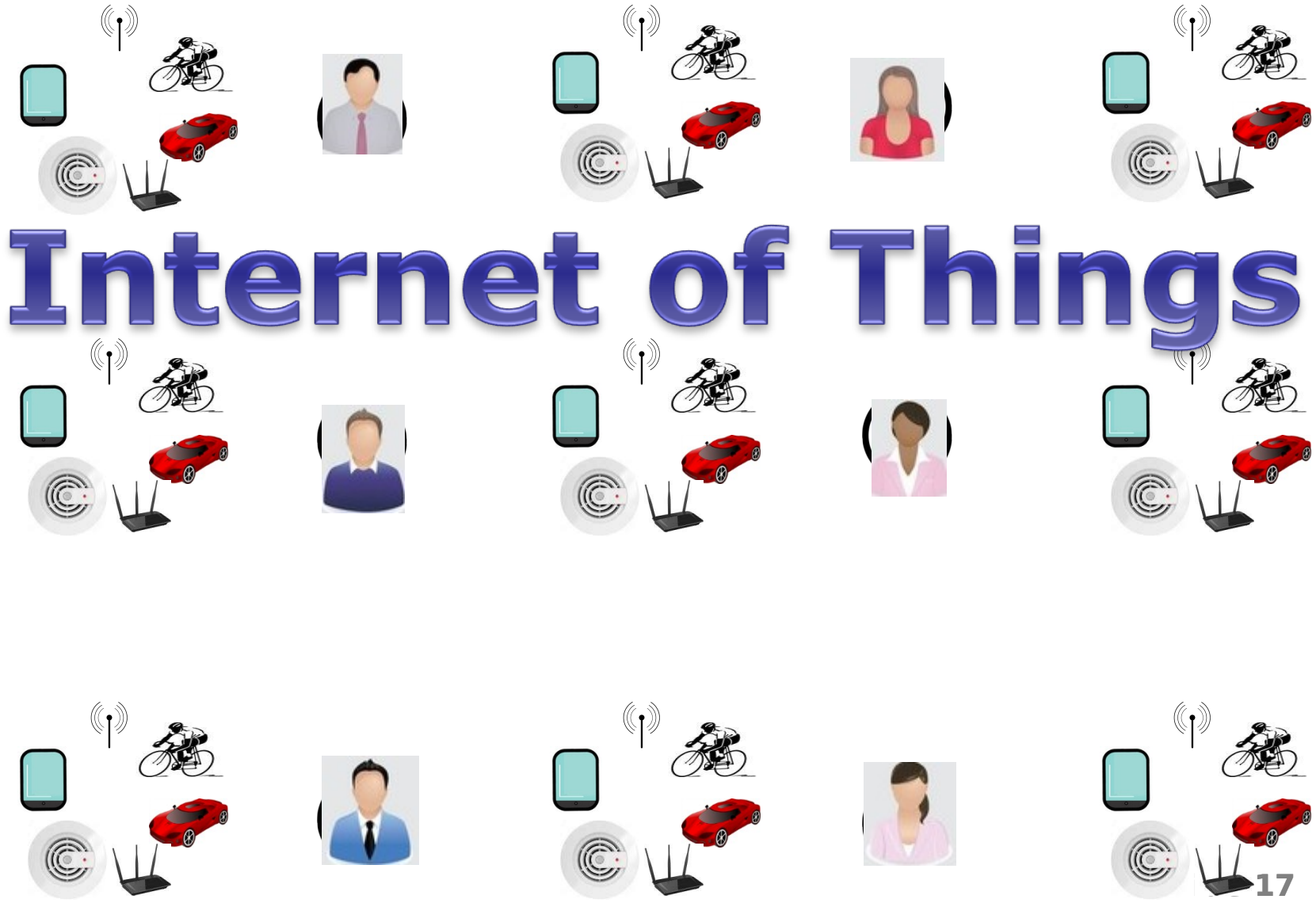



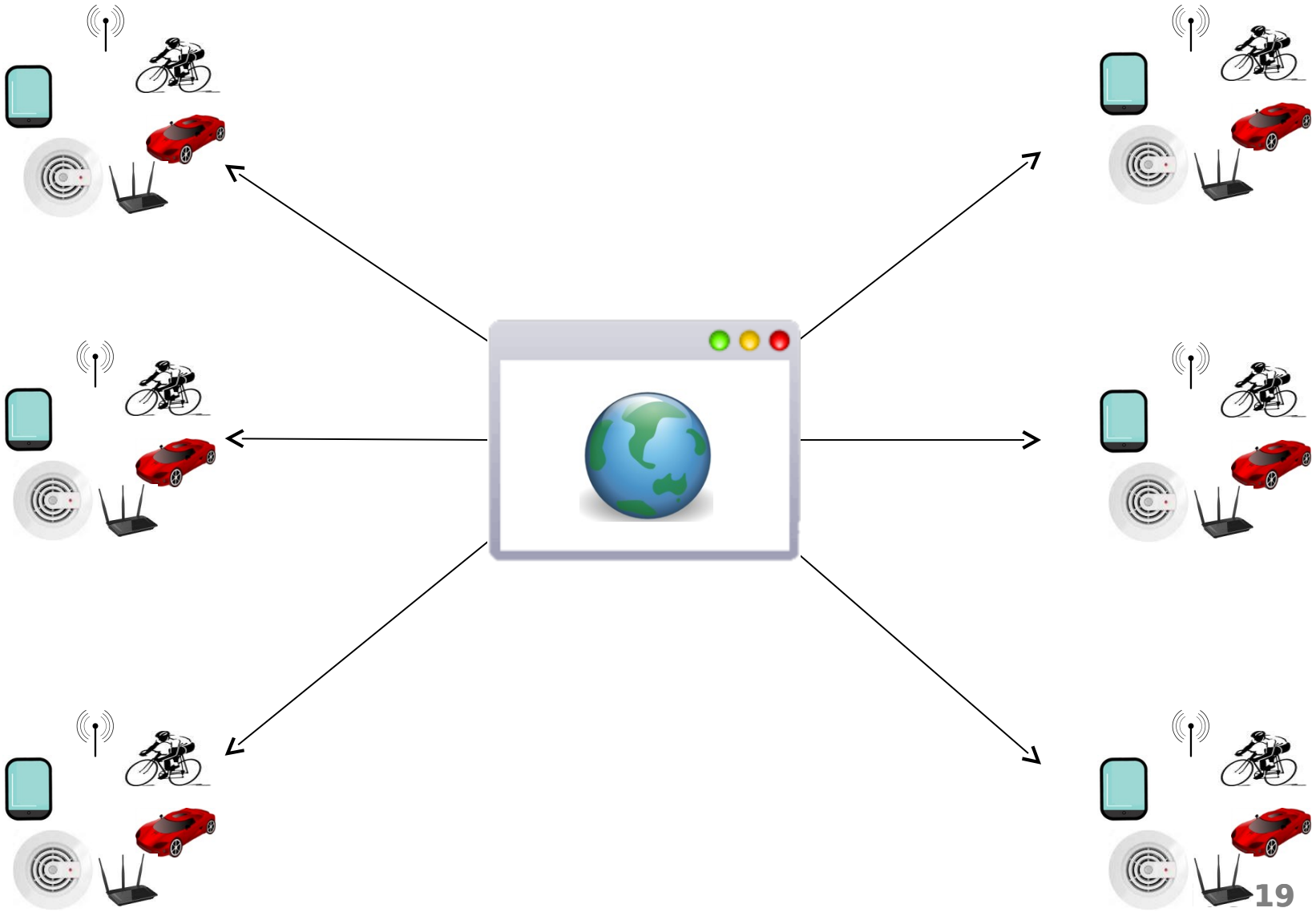


Internet of Things

IoT



IoT

IoT

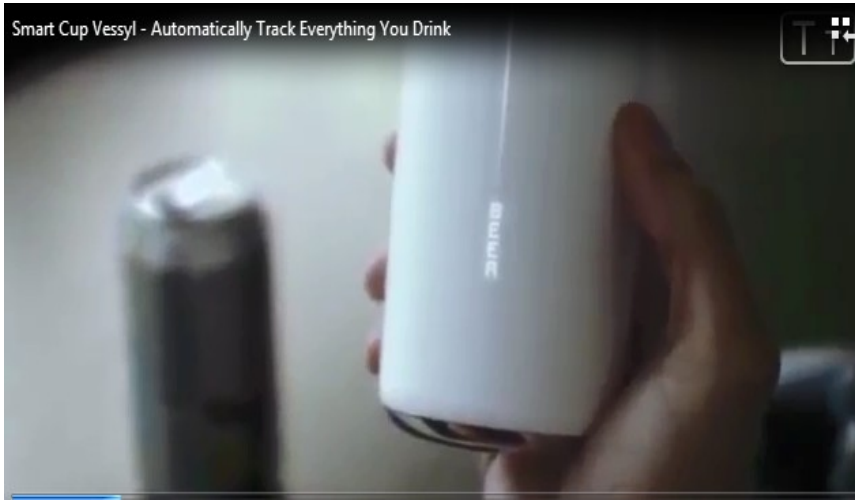


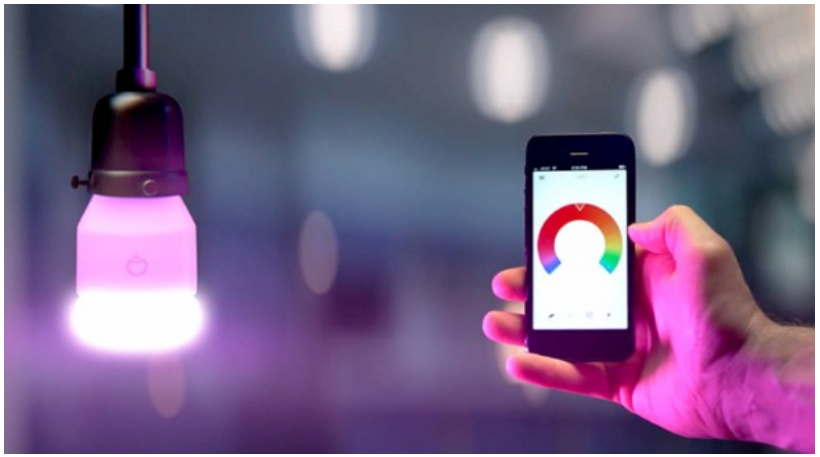
IoT

Fonte: (Bonomi et al; 2014)

IoT

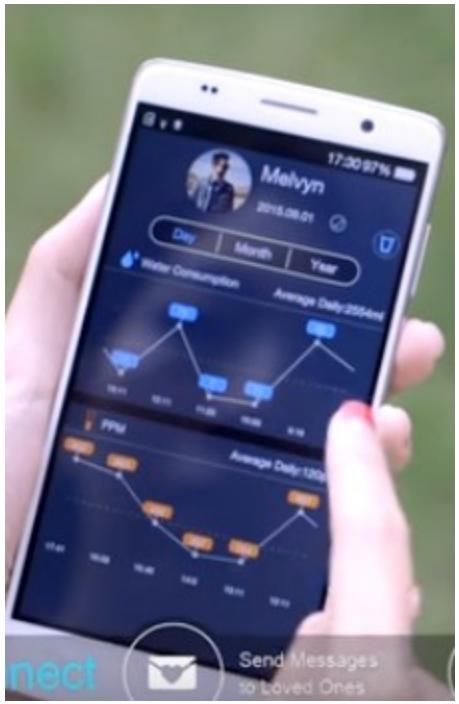
Não é apenas...

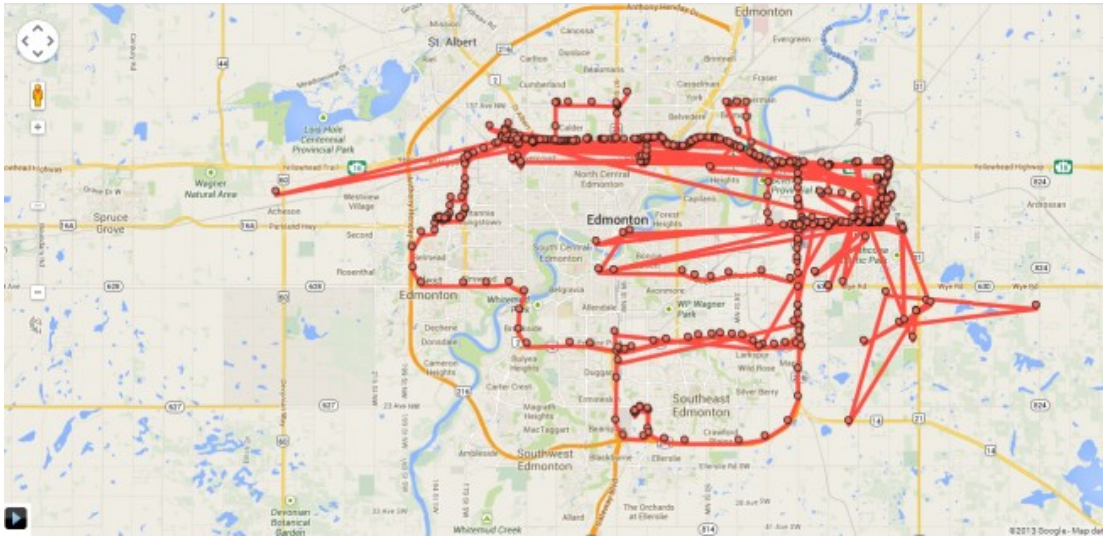




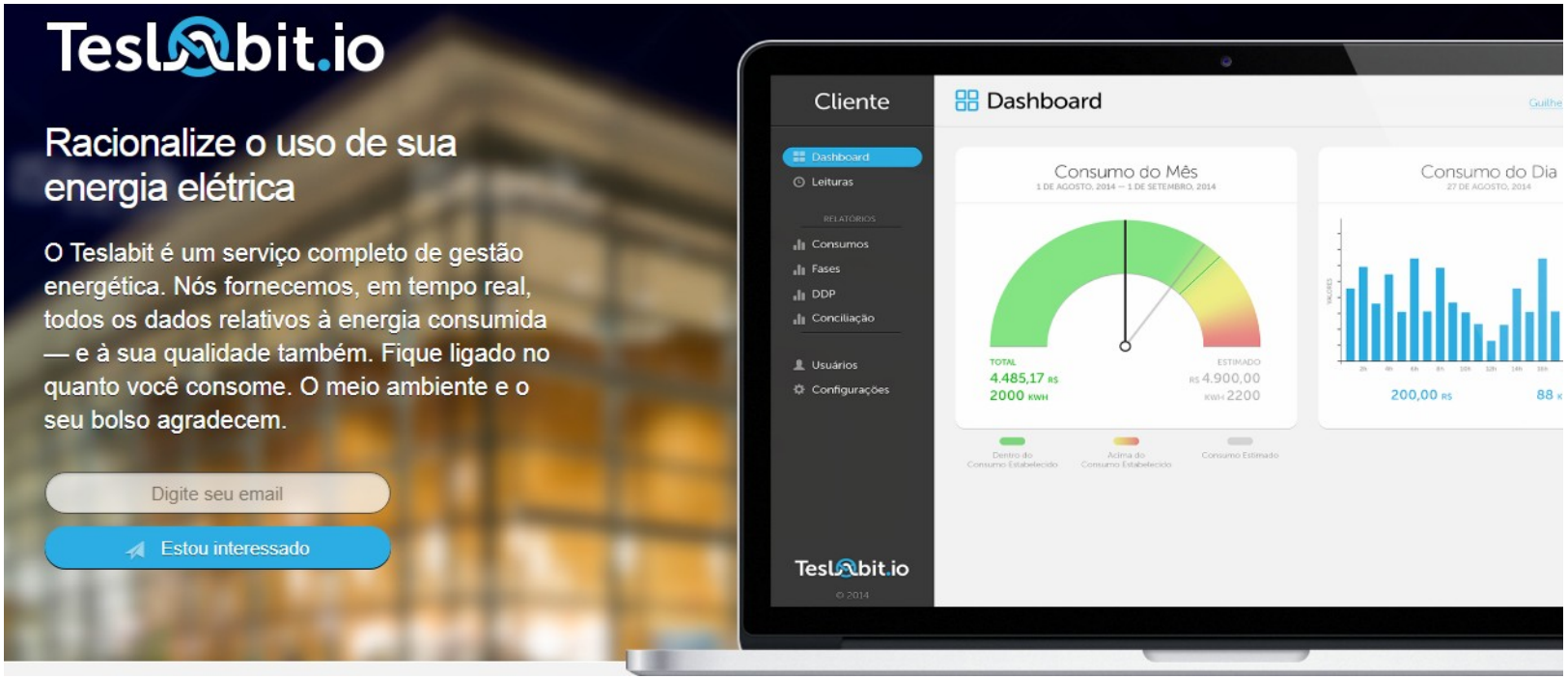
IoT

Precisamos...

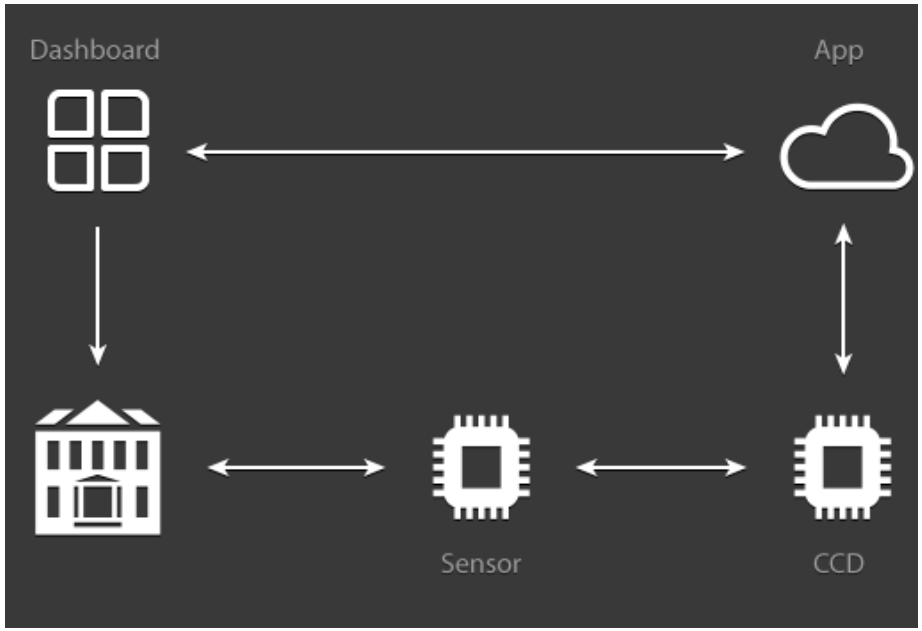




IoT – Exemplo Analytics



IoT – Exemplo Analytics



Como funciona?

1. Instalamos sensores em cada circuito e capturamos dados relativos à energia elétrica.
2. Os dados são enviados por meio de tecnologias redundantes para os servidores na internet.
3. O cliente pode visualizar seus dados através da Web, a qualquer hora, em qualquer lugar.

IoT – Exemplo Analytics

O AWS IoT Analytics é um serviço totalmente gerenciado que facilita a execução de análises sofisticadas em volumes massivos de dados da IoT, sem necessidade de se preocupar com todo o custo e a complexidade normalmente associados à criação da sua própria plataforma de análises da IoT. Esta é a forma mais fácil de executar análises de dados da IoT e obter insights que ajudam a tomar decisões melhores e mais precisas para

Internet of Things

Fonte: http://www.3g.co.uk

Internet of Things

Desafios:

Heterogeneidade de dispositivos

Escalabilidade

Troca de dados ubíqua

Otimização de energia

Localização e rastreamento

Autogerenciamento

Gerenciamento de dados

Semântica de interoperabilidade

Segurança e privacidade

Mobilidade

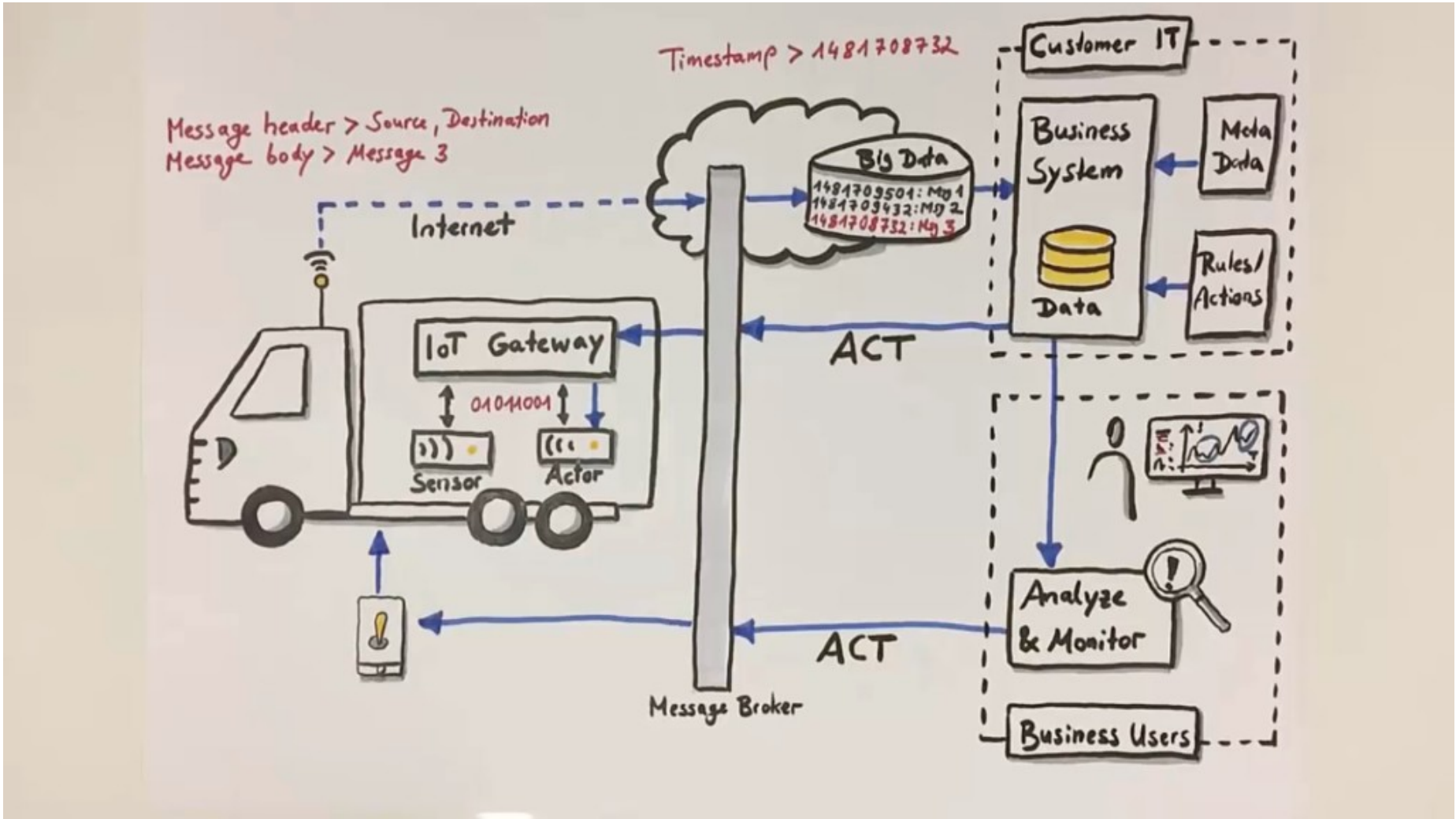
Fonte: http://www.123rf.com

IoT e Big Data = IoT Data

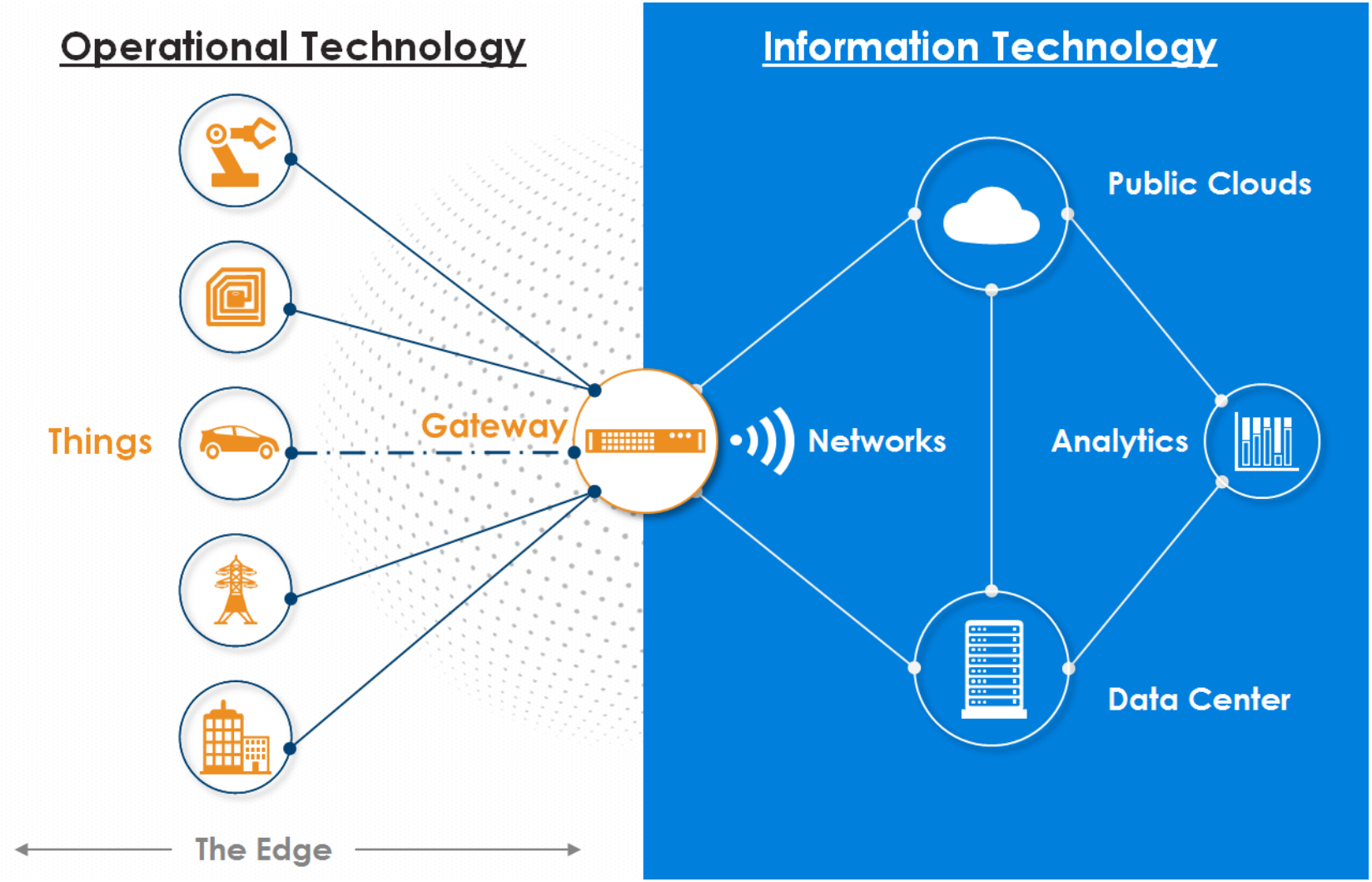
Typical data flow in an IoT solution

https://www.youtube.com/watch?v=y9KMkjAn_b0

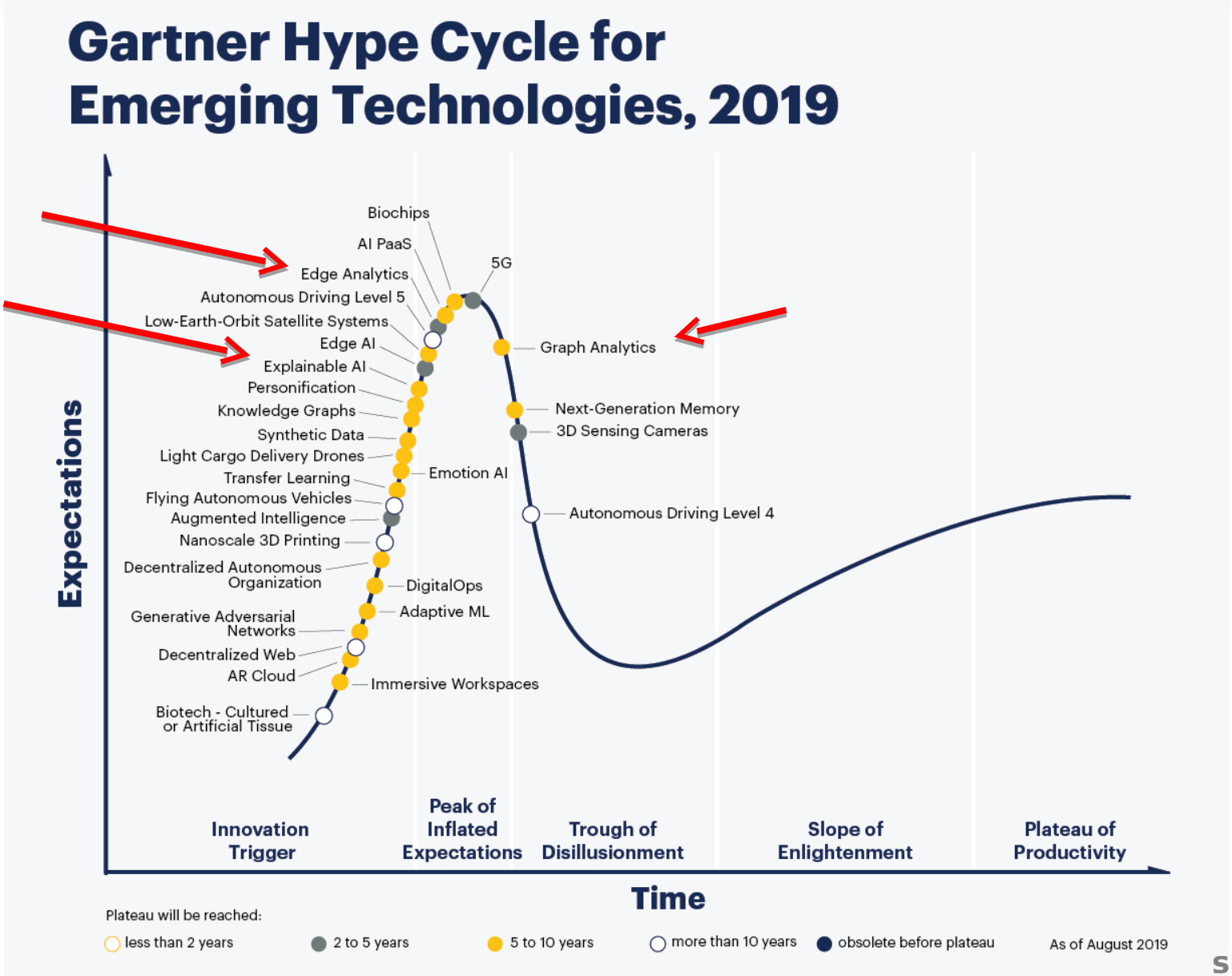
IoT


Timestamp > 1481708732
Customer IT
Message header > Source, Destination
Message body > Message 3
Big Data
1481709501: Msg 1
1481709432: Msg 2
1481708732: Msg 3
Business System
Meta Data
Internet
Data
Rules/ Actions
IoT Gateway
ACT
01011001
Sensor
Actor
Analyze & Monitor
ACT
Message Broker
Business Users

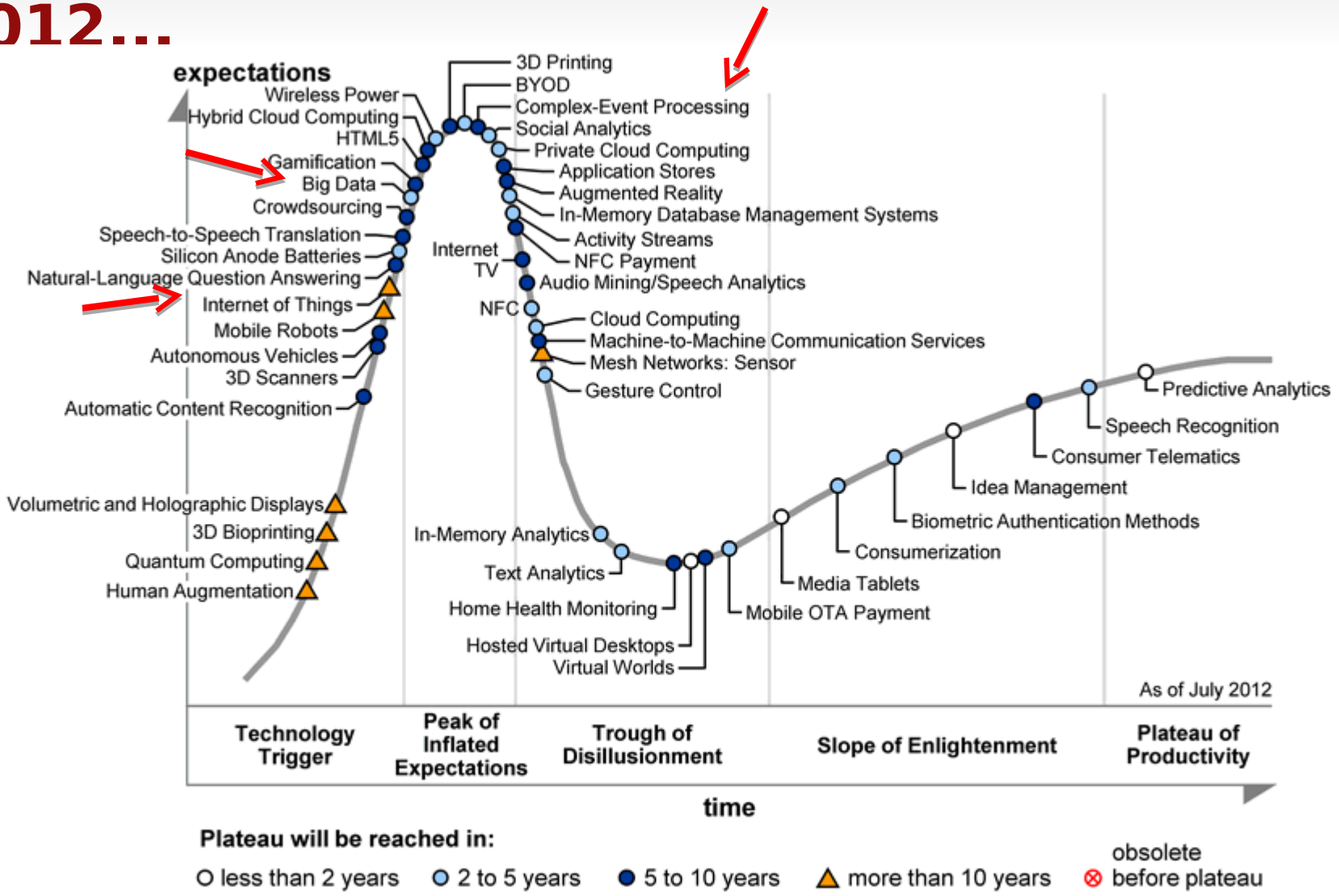
IoT Solution Architecture



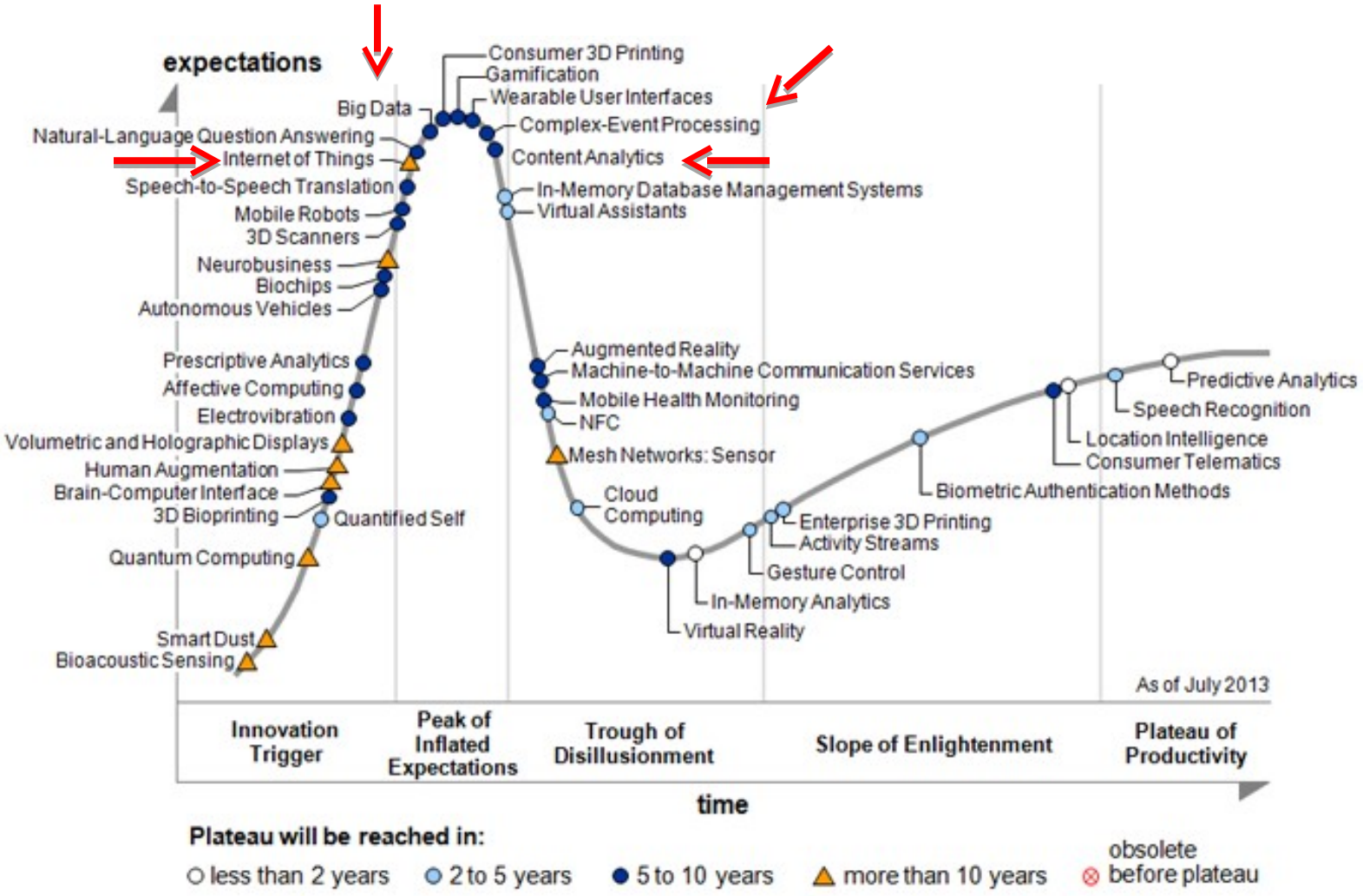
Gartner Hype Cycle - 2019



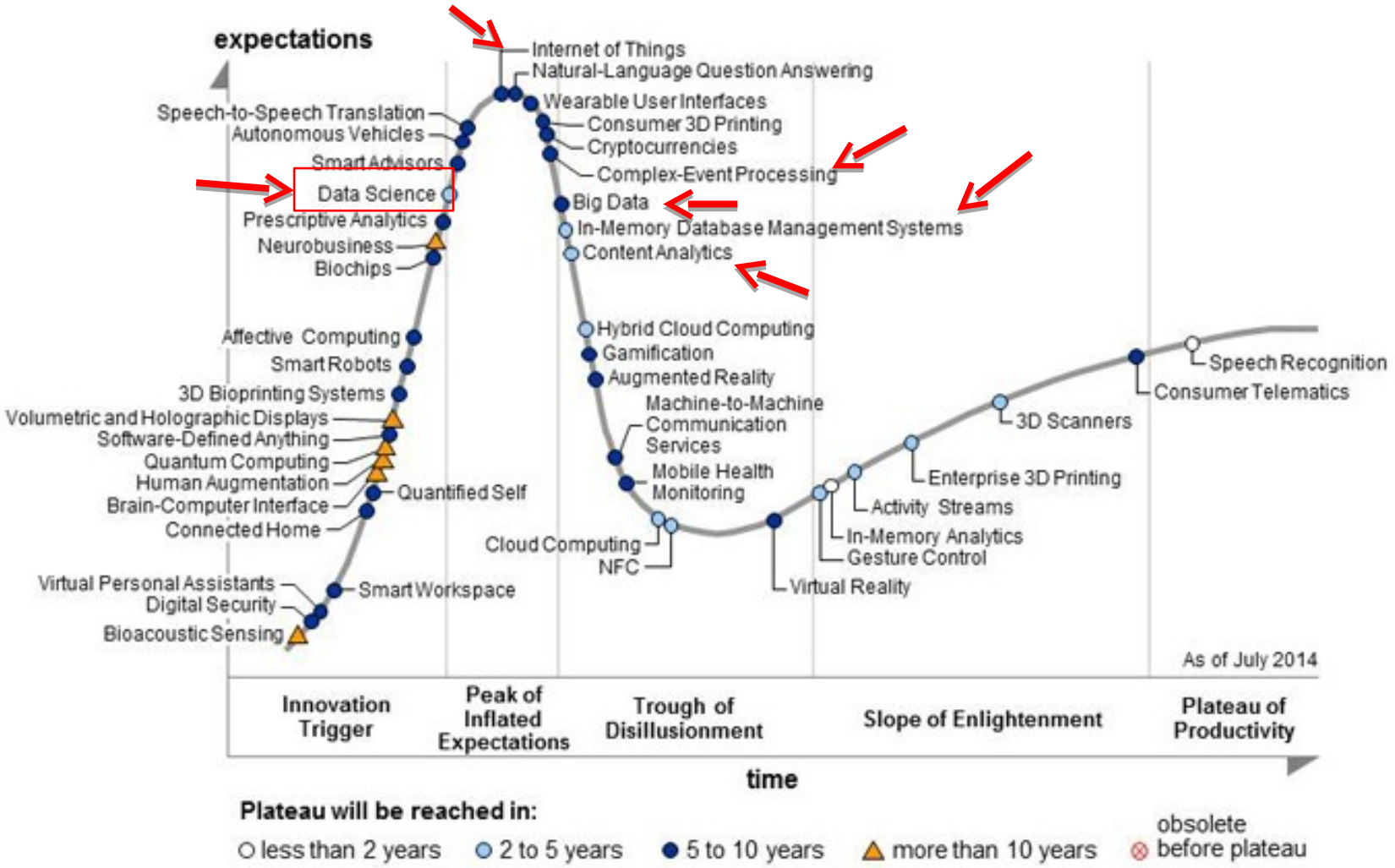
Falando em Hype Cycle – 2012...



Falando em Hype Cycle - 2013...



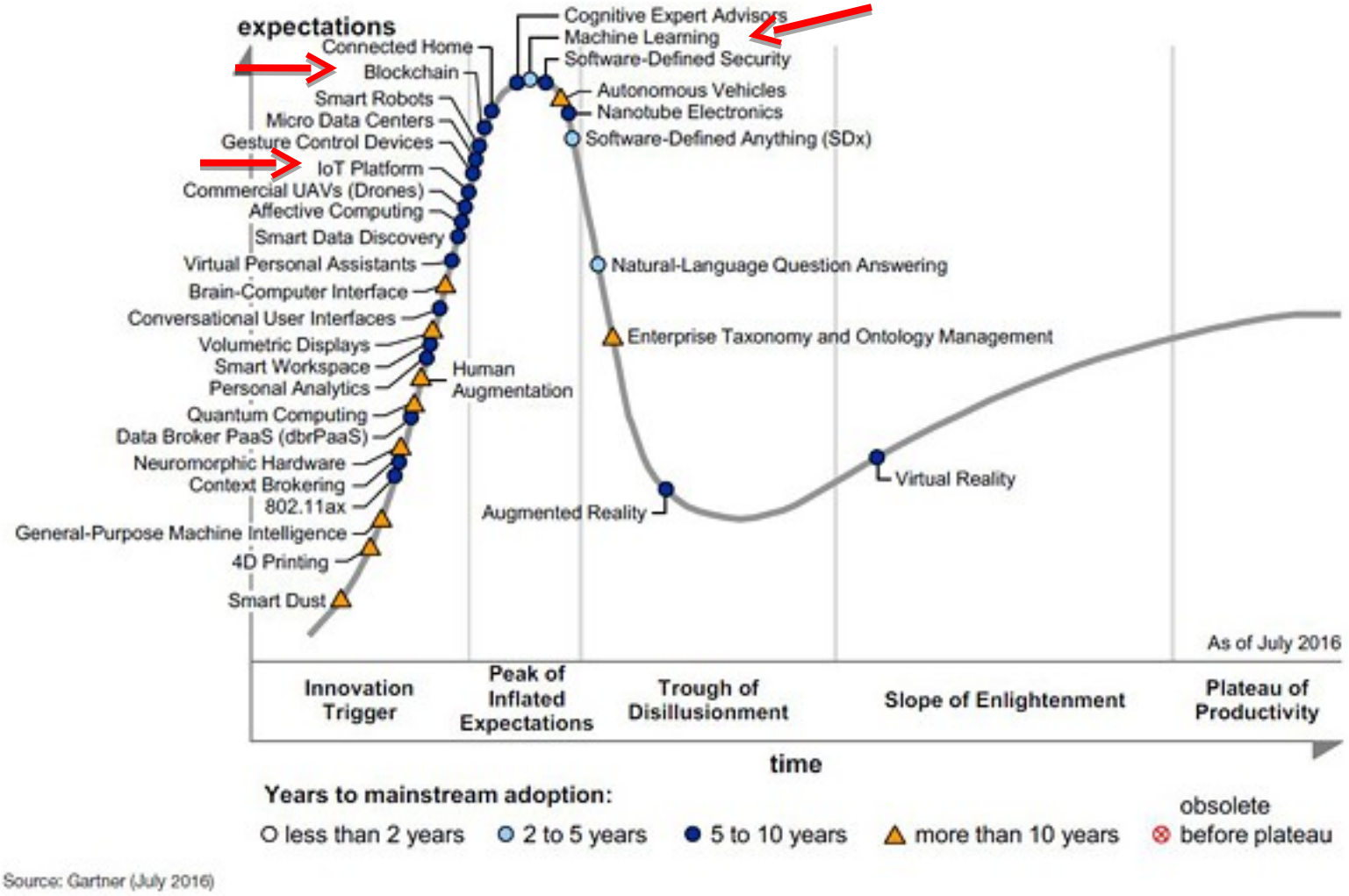
Falando em Hype Cycle - 2014...



Falando em Hype Cycle 2015...

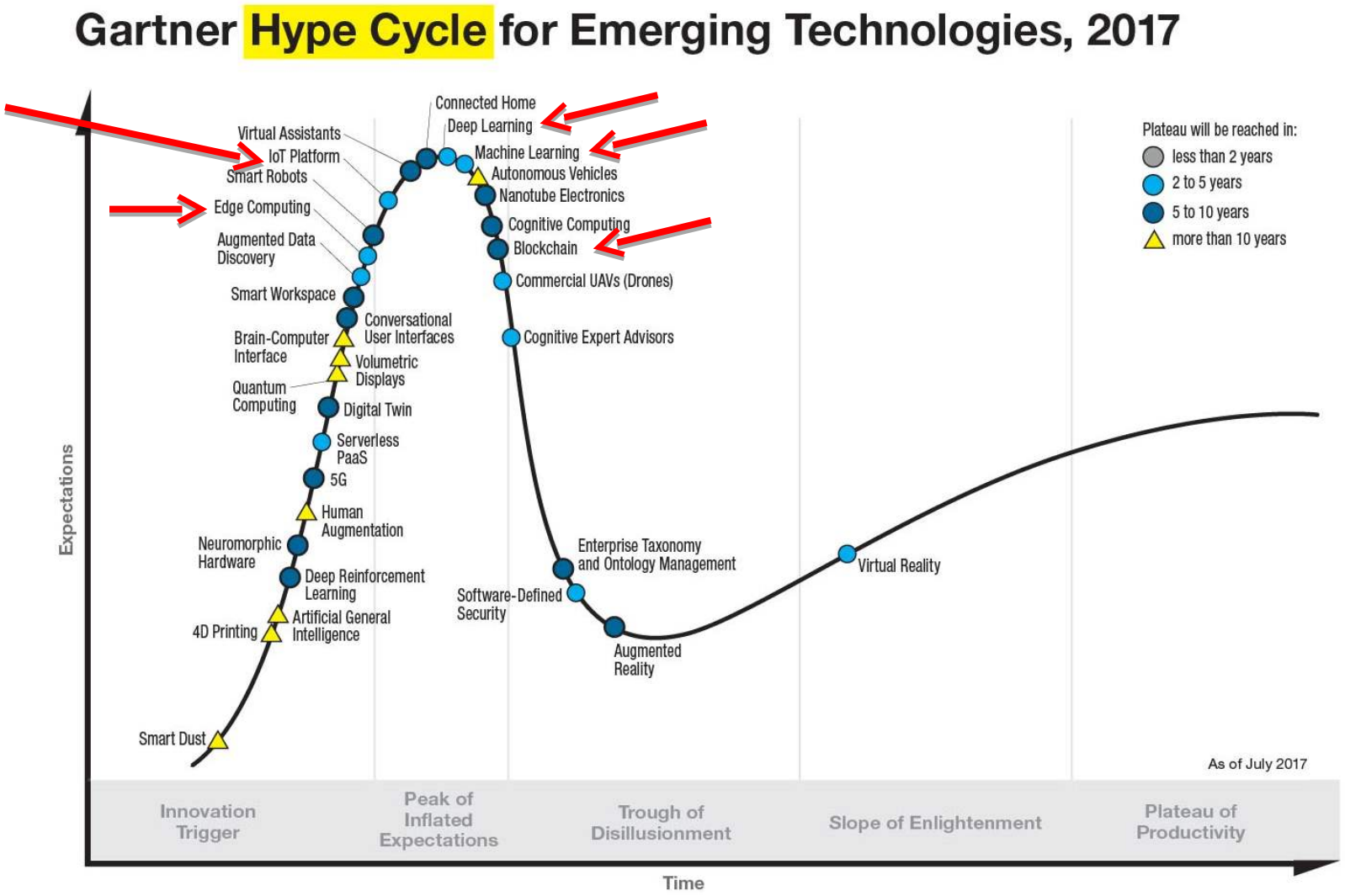
Figure 1. Hype Cycle for Emerging Technologies, 2015

Gartner Hype Cycle - 2016

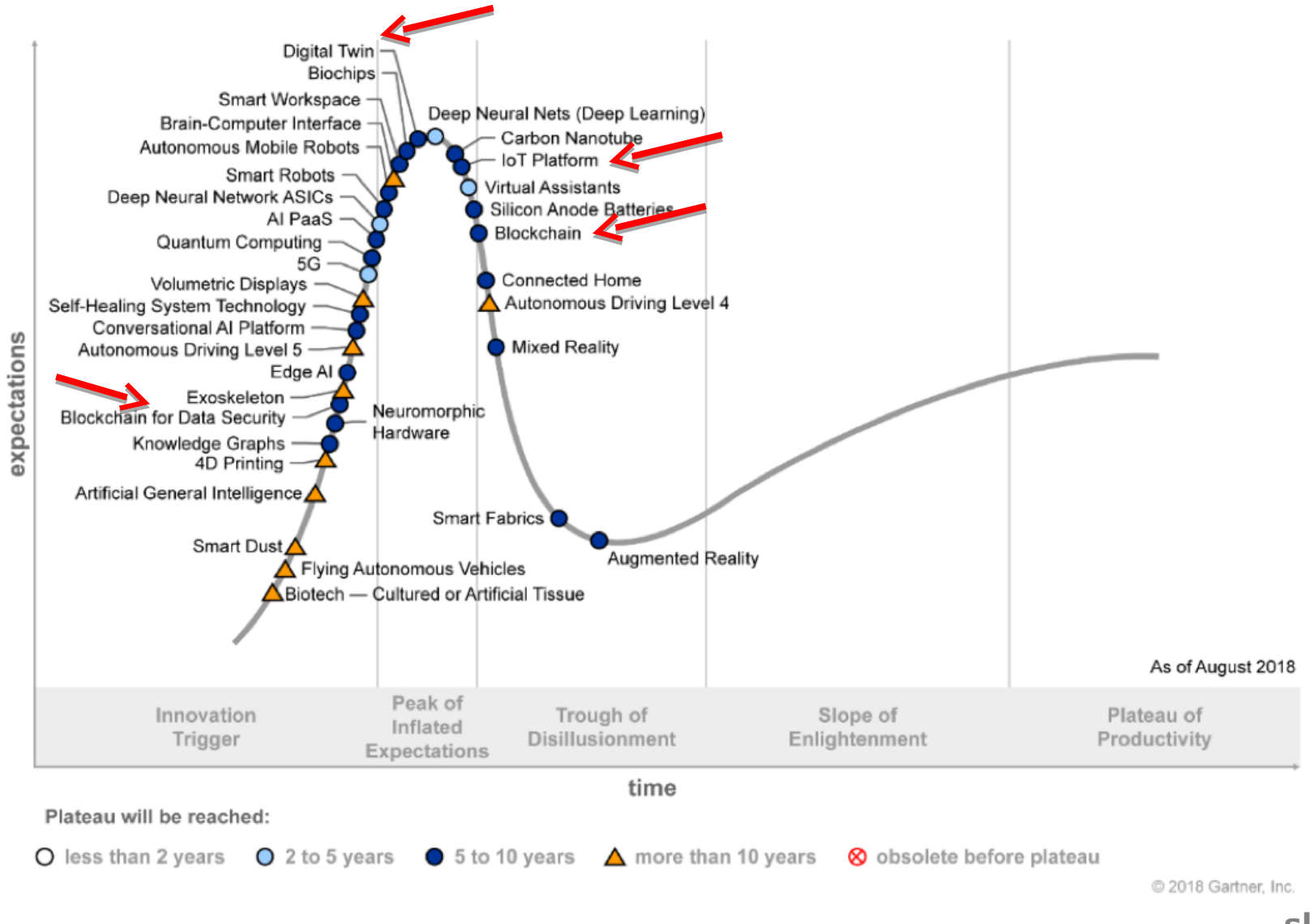


Source: Gartner (July 2016)

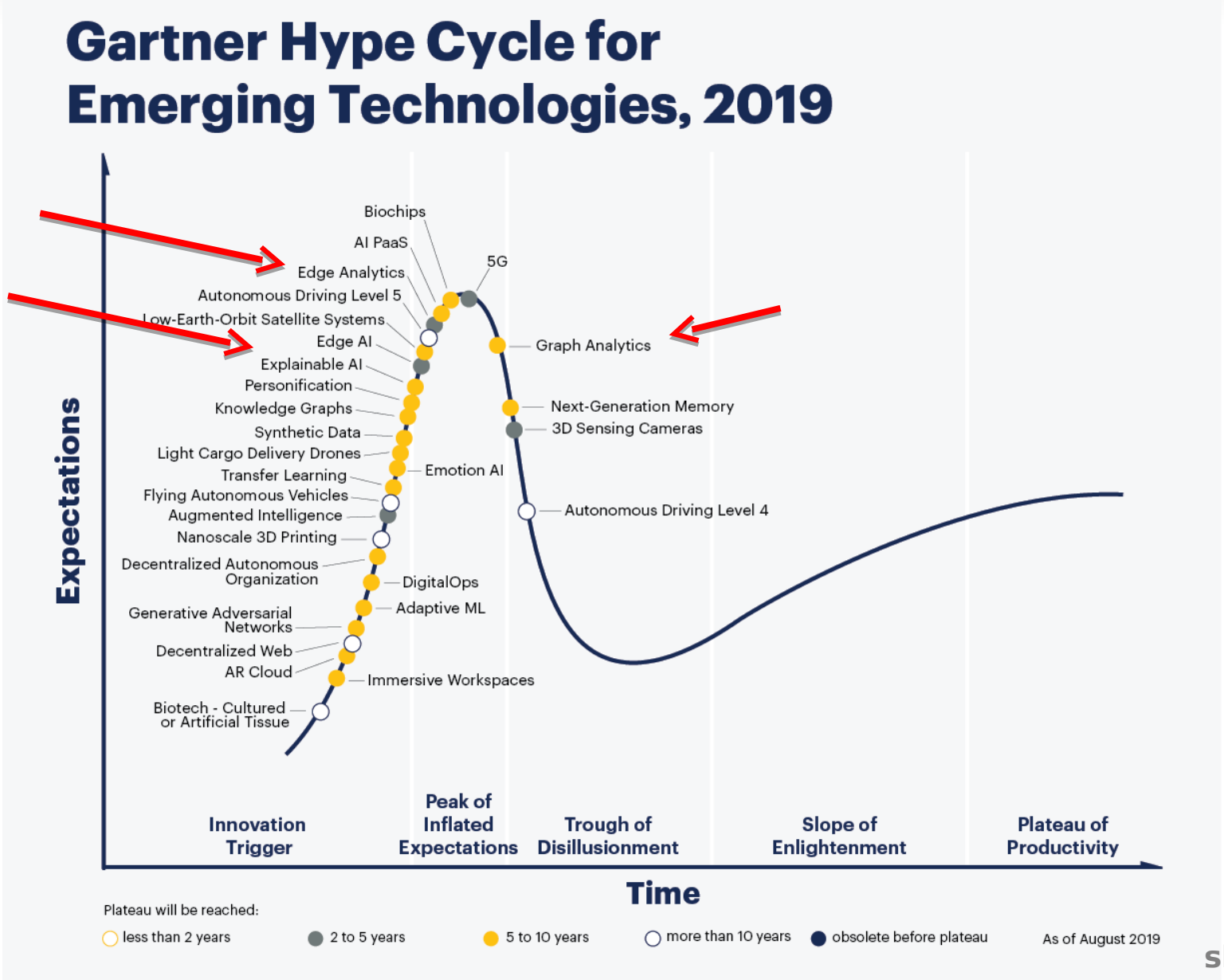
Gartner Hype Cycle - 2017



Gartner Hype Cycle - 2018



Gartner Hype Cycle - 2019



Uma nova profissão...

Data Scientist

vs.

Data Engineer

Uma nova profissão...

Data Scientist

A data scientist is the alchemist of the 21st century: someone who can turn raw data into purified insights. Data scientists apply statistics, machine learning and analytic approaches to solve critical business problems. Their primary function is to help organizations turn their volumes of big data into valuable and actionable insights

"vs."

Data Engineers are the data professionals who prepare the "big data" infrastructure to be analyzed by Data Scientists. They are software engineers who design, build, integrate data from various resources, and manage big data.

Data Engineer

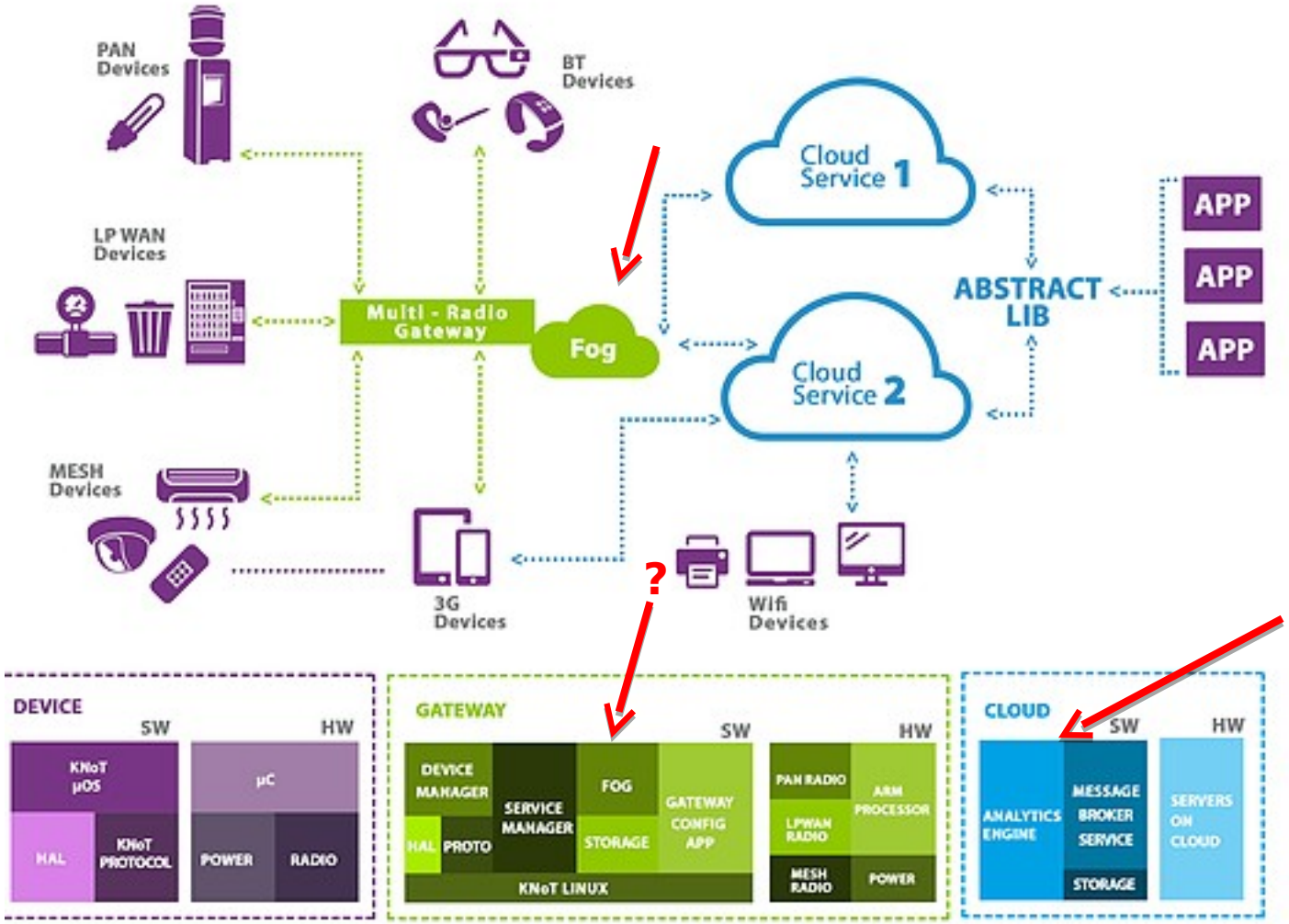
https://cognitiveclass.ai/blog/data-scientist-vs-data-engineer/

KNoT

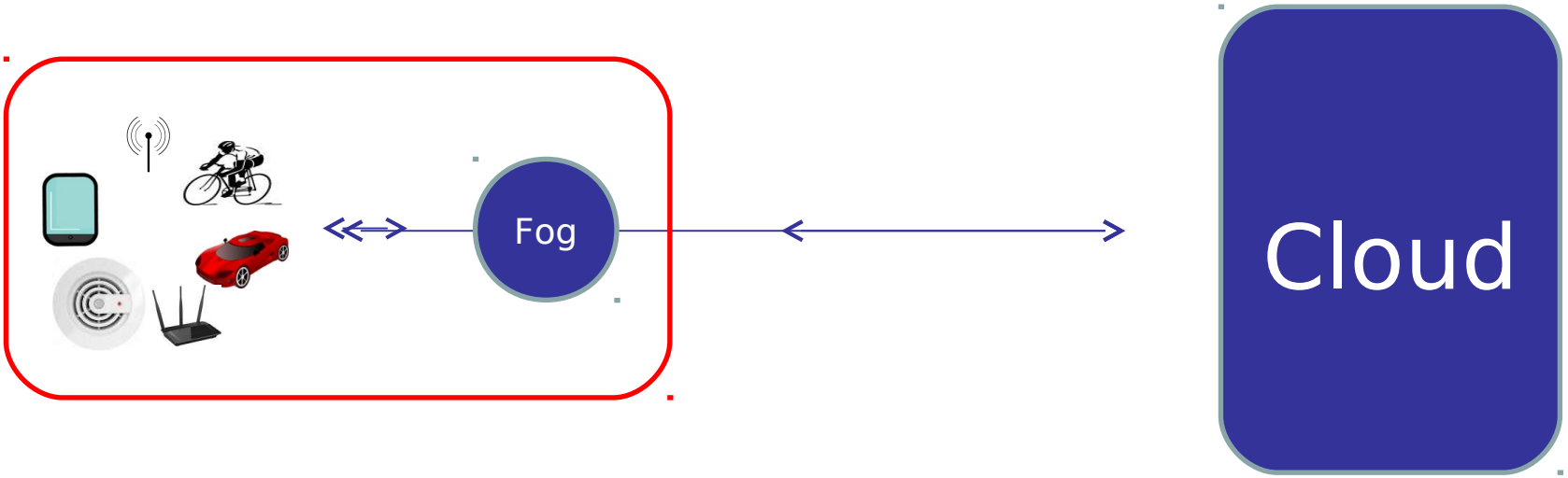
POWERED BY

C . E . S . A . R

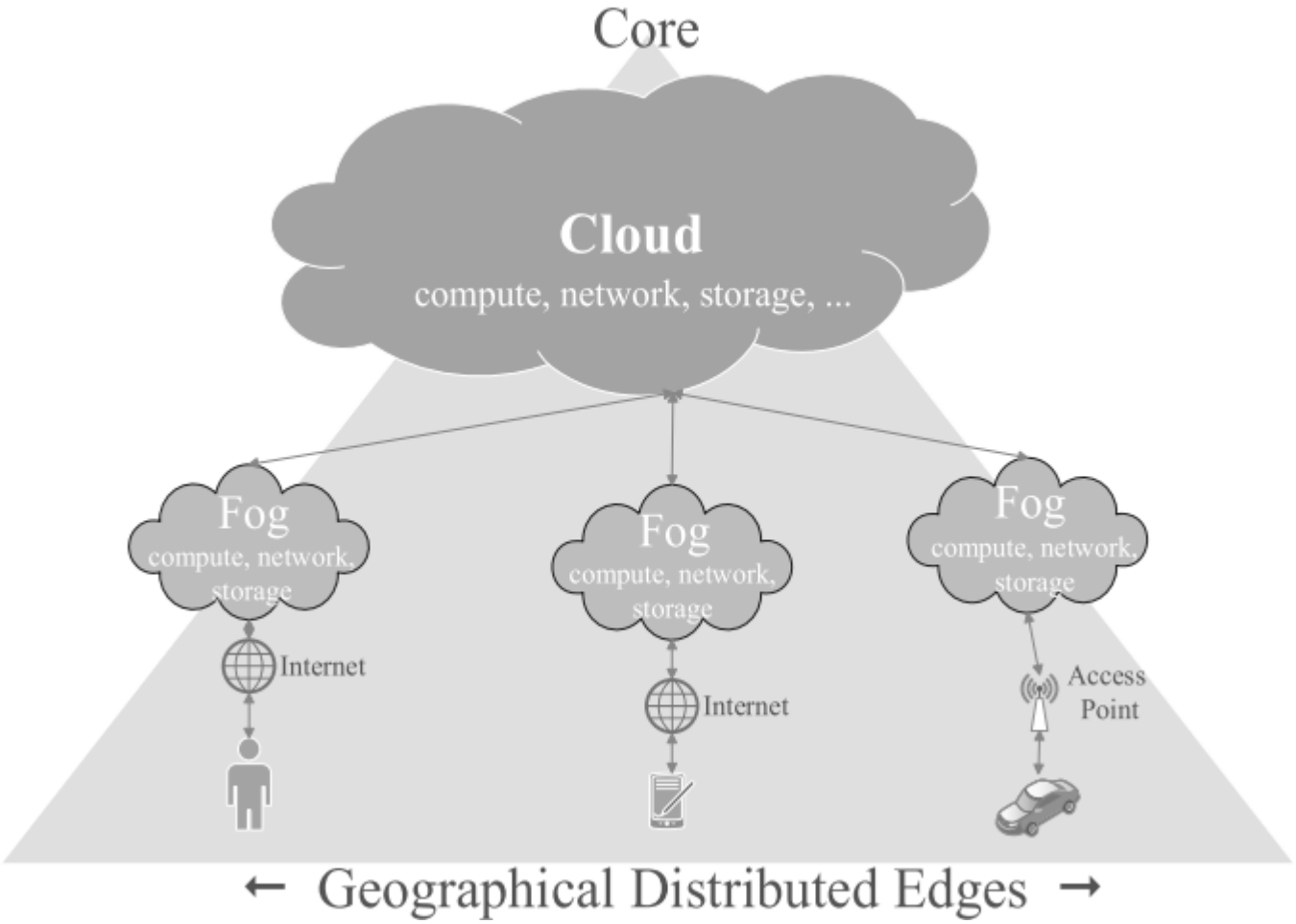
KNoT


PAN Devices
BT Devices
LP WAN Devices
Multi - Radio Gateway
Fog
Cloud Service 1
Cloud Service 2
ABSTRACT LIB
APP
APP
APP
MESH Devices
3G Devices
Wifi Devices
?
DEVICE
SW
HW
KNoT µOS
HAL
KNoT PROTOCOL
µC
POWER
RADIO
GATEWAY
SW
HW
DEVICE MANAGER
HAL
PROTO
SERVICE MANAGER
FOG
STORAGE
GATEWAY CONFIG APP
KNoT LINUX
PAN RADIO
LPWAN RADIO
MESH RADIO
ARM PROCESSOR
POWER
CLOUD
SW
HW
ANALYTICS ENGINE
MESSAGE BROKER SERVICE
STORAGE
SERVERS ON CLOUD

Fog Computing



Fog Computing



Fonte: (Bonomi et al; 2014)

Fog Computing - Características

1. Edge location, location awareness, and low latency
2. Geographical distribution
3. Large-scale sensor networks to monitor the environment
4. Support for mobility
5. Real-time interactions
6. Predominance of wireless access
7. Heterogeneity
8. Interoperability
9. Support for on-line analytic and interplay with the Cloud.

Fog Computing - articles

Fog Computing and Its Role in the Internet of Things

Flavio Bonomi, Rodolfo Milito, Jiang Zhu, Sateesh Addepalli
Cisco Systems Inc.
170 W Tasman Dr. San Jose, CA 95134, USA
{flavio, romilito, jiangzhu, sateeshk}@cisco.com

2012

Fog Computing: A Platform for Internet of Things and Analytics

Flavio Bonomi, Rodolfo Milito, Preethi Natarajan and Jiang Zhu

2014

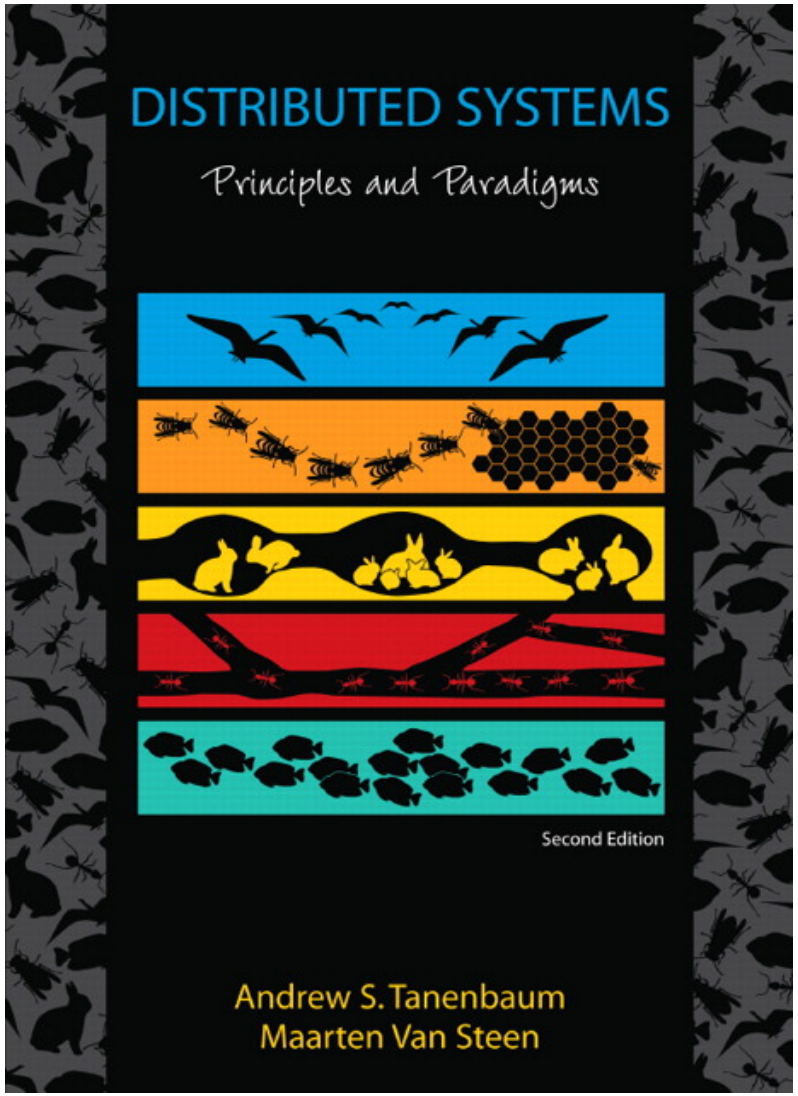
....Conclusão...

O mundo é (e será cada vez mais) distribuído

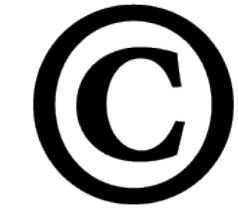
Dados de todos os lugares...

Como processar esses dados?

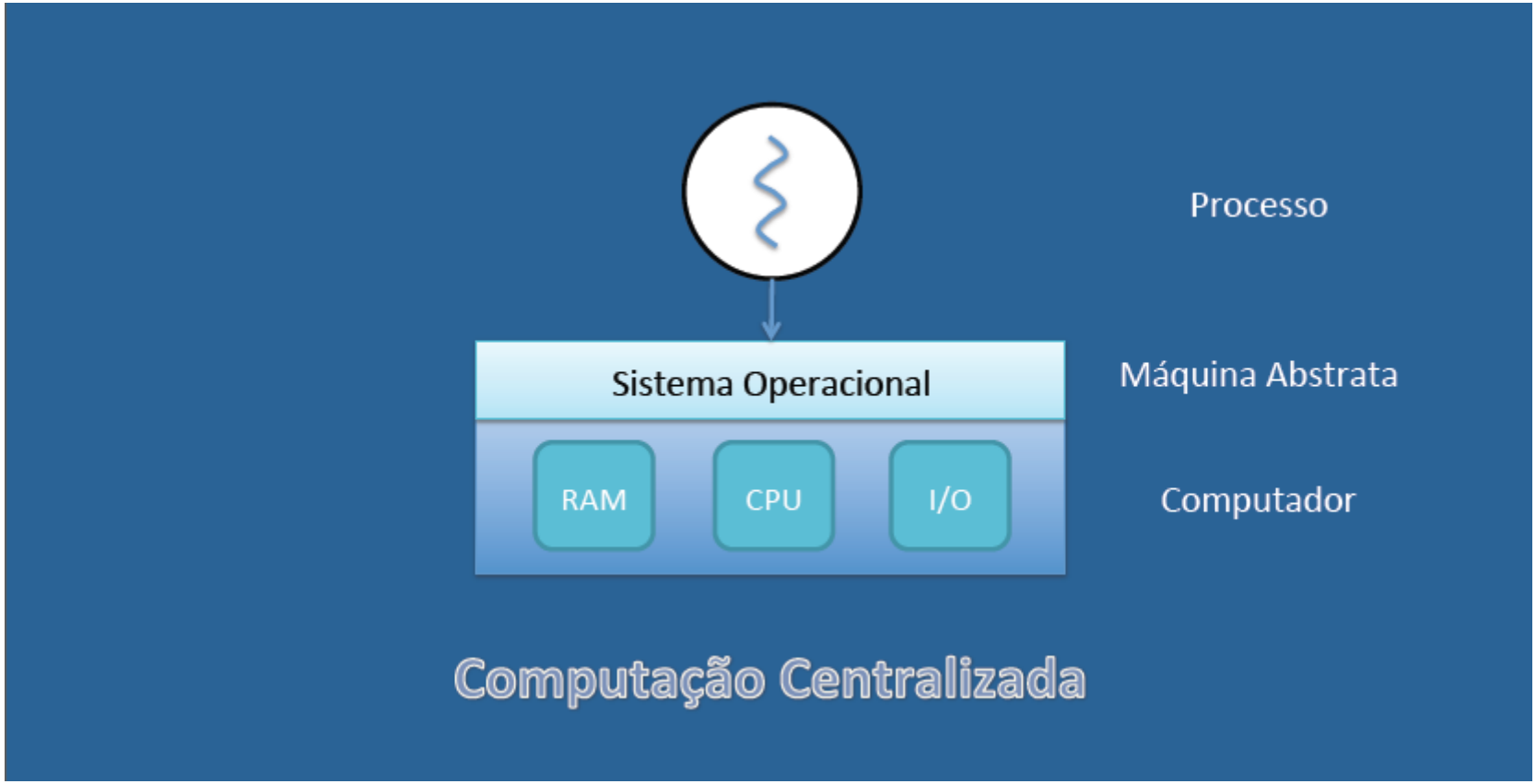
Sistemas Distribuídos



SD - Evolução

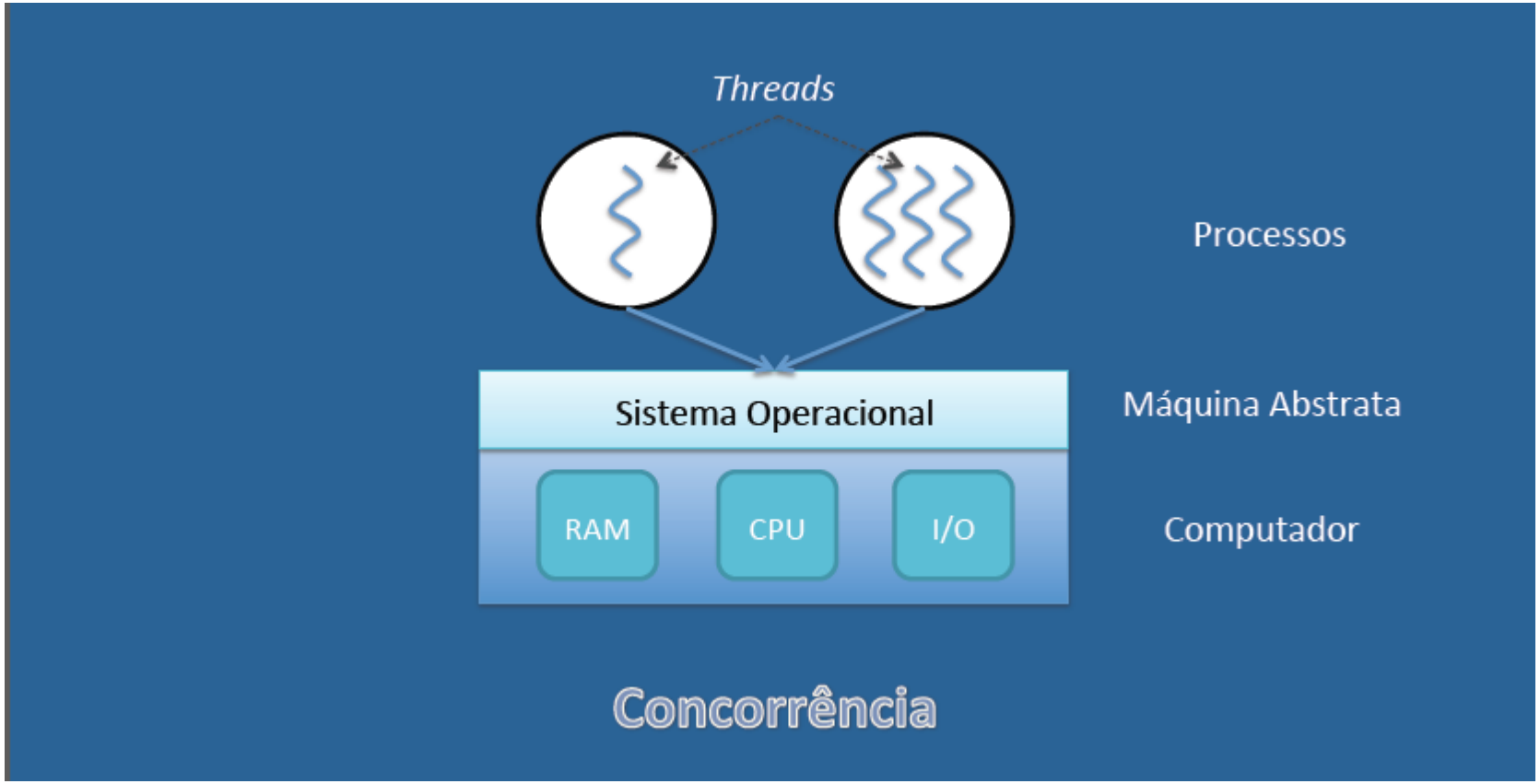


SD - Evolução

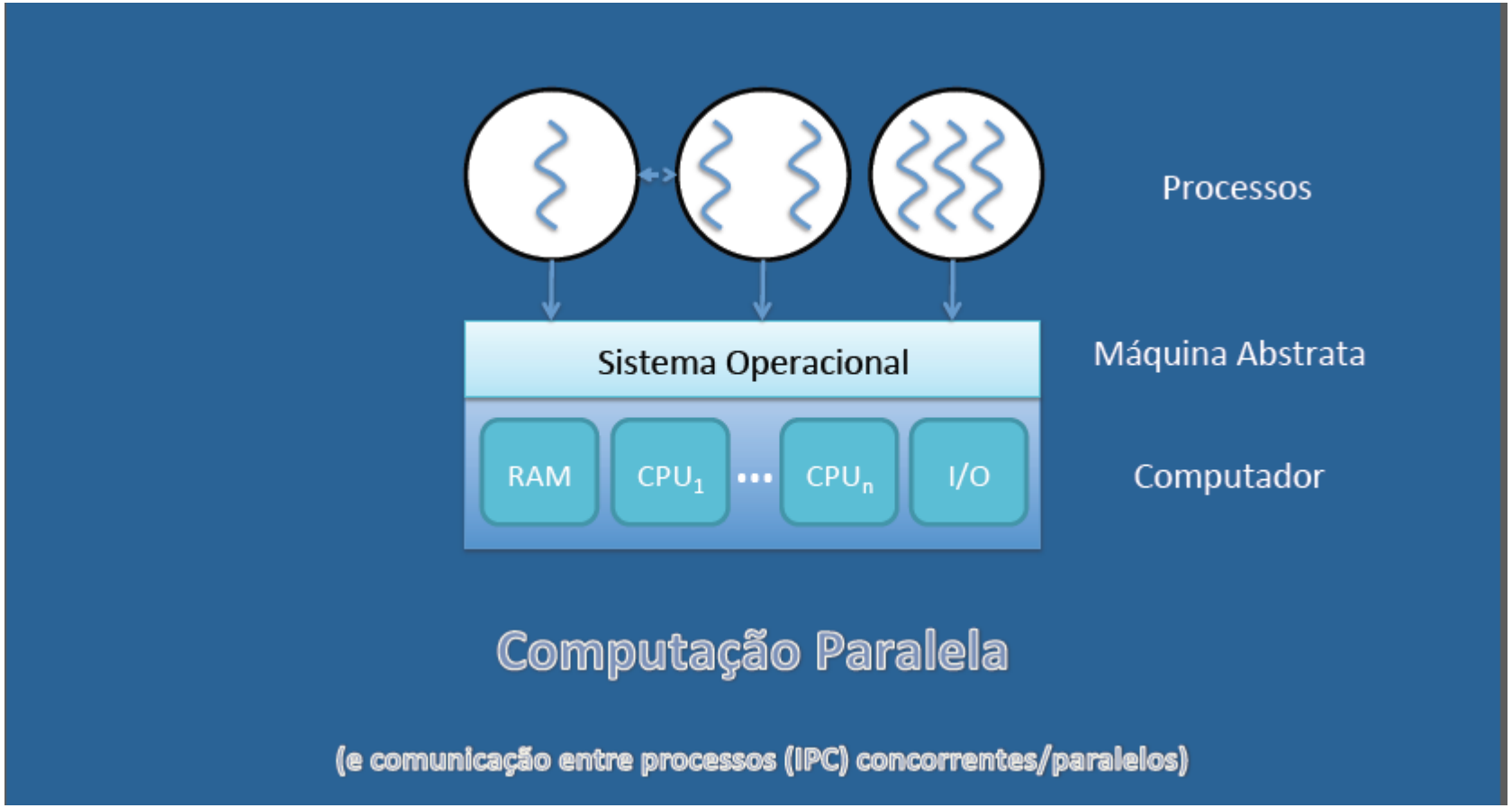


O que é uma Thread?

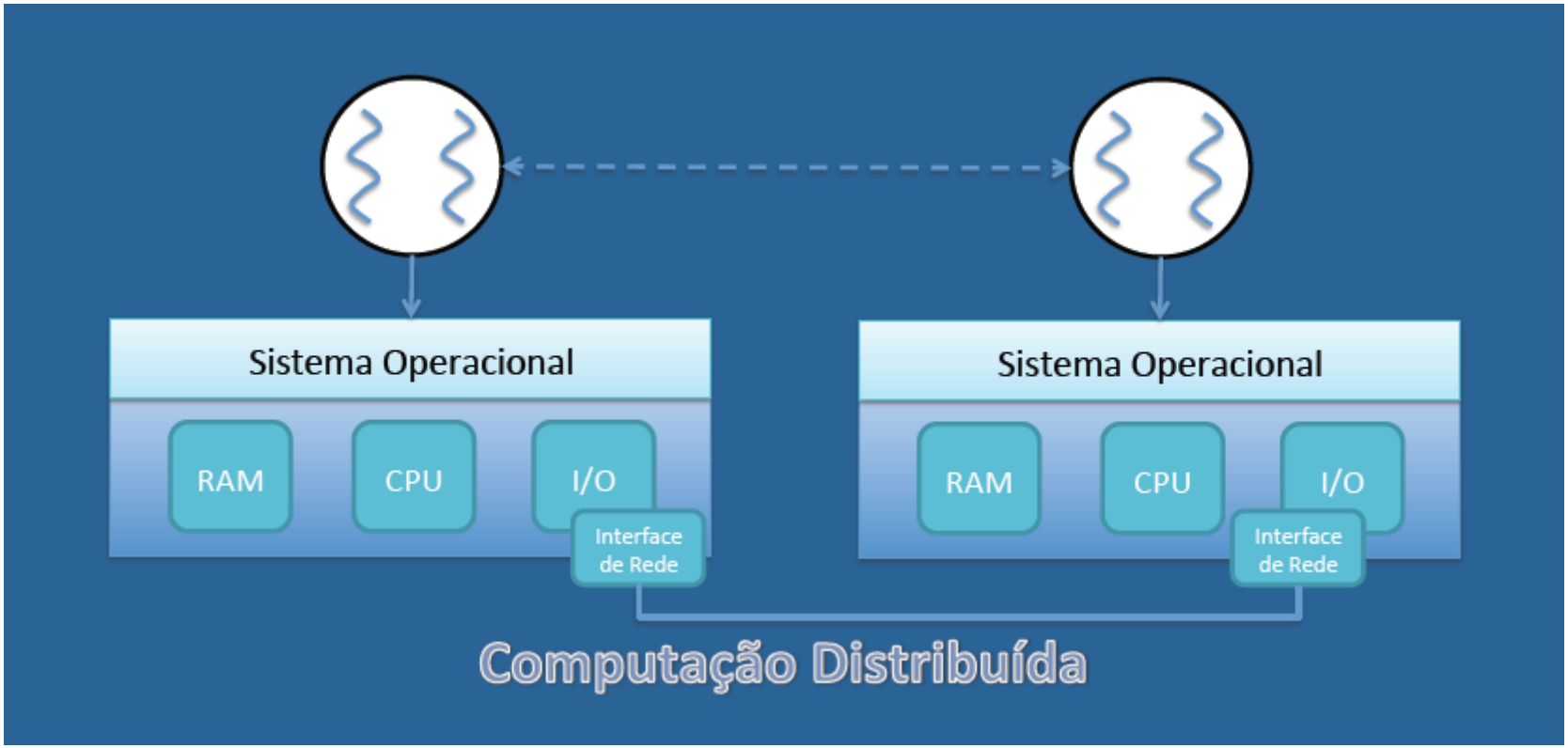
SD - Evolução


Threads
Processos
Sistema Operacional
Máquina Abstrata
RAM
CPU
I/O
Computador
Concorrência

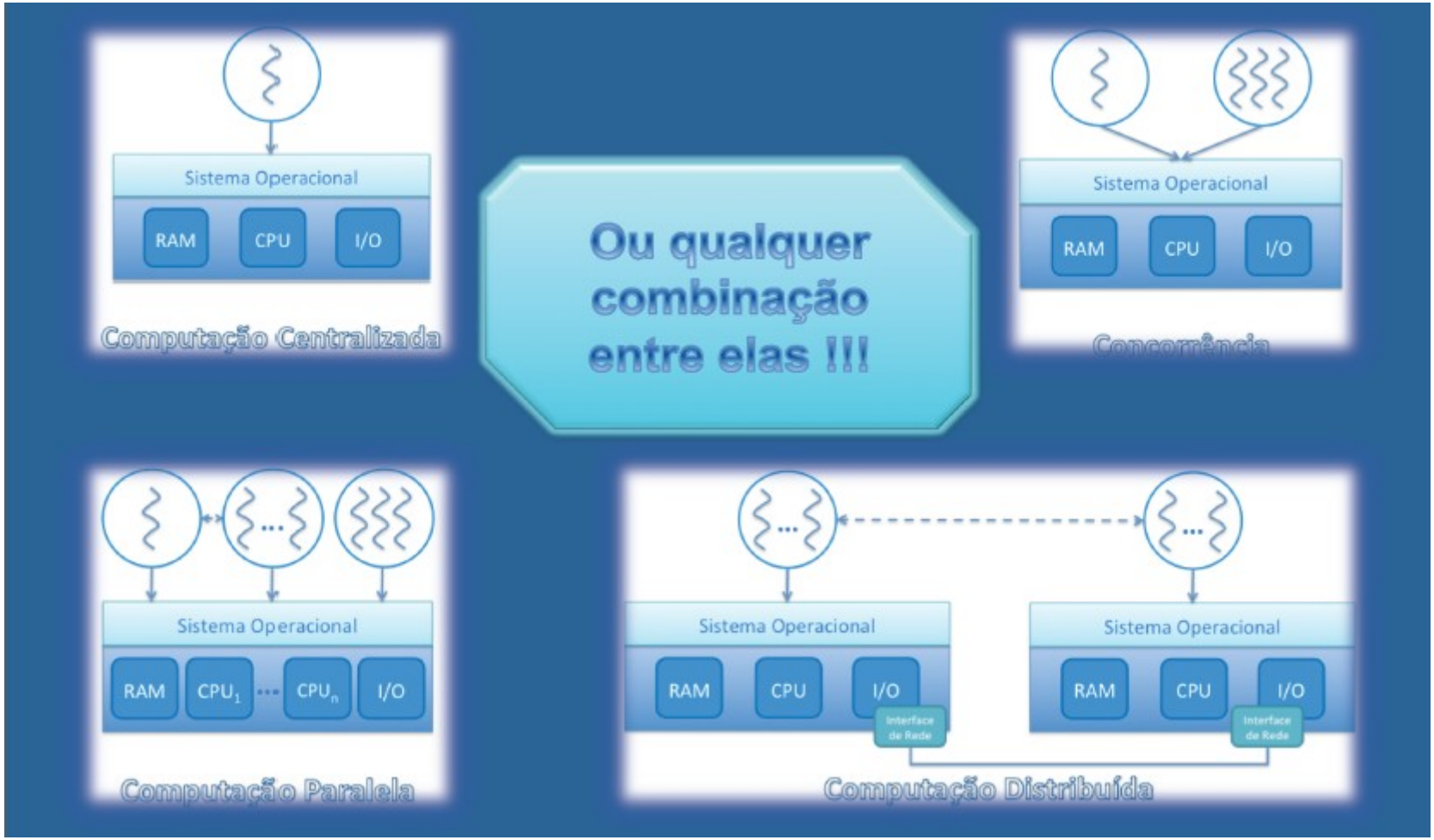
SD - Evolução


Processos
Sistema Operacional
Máquina Abstrata
RAM
CPU1
CPUn
I/O
Computador
Computação Paralela
(e comunicação entre processos (IPC) concorrentes/paralelos)

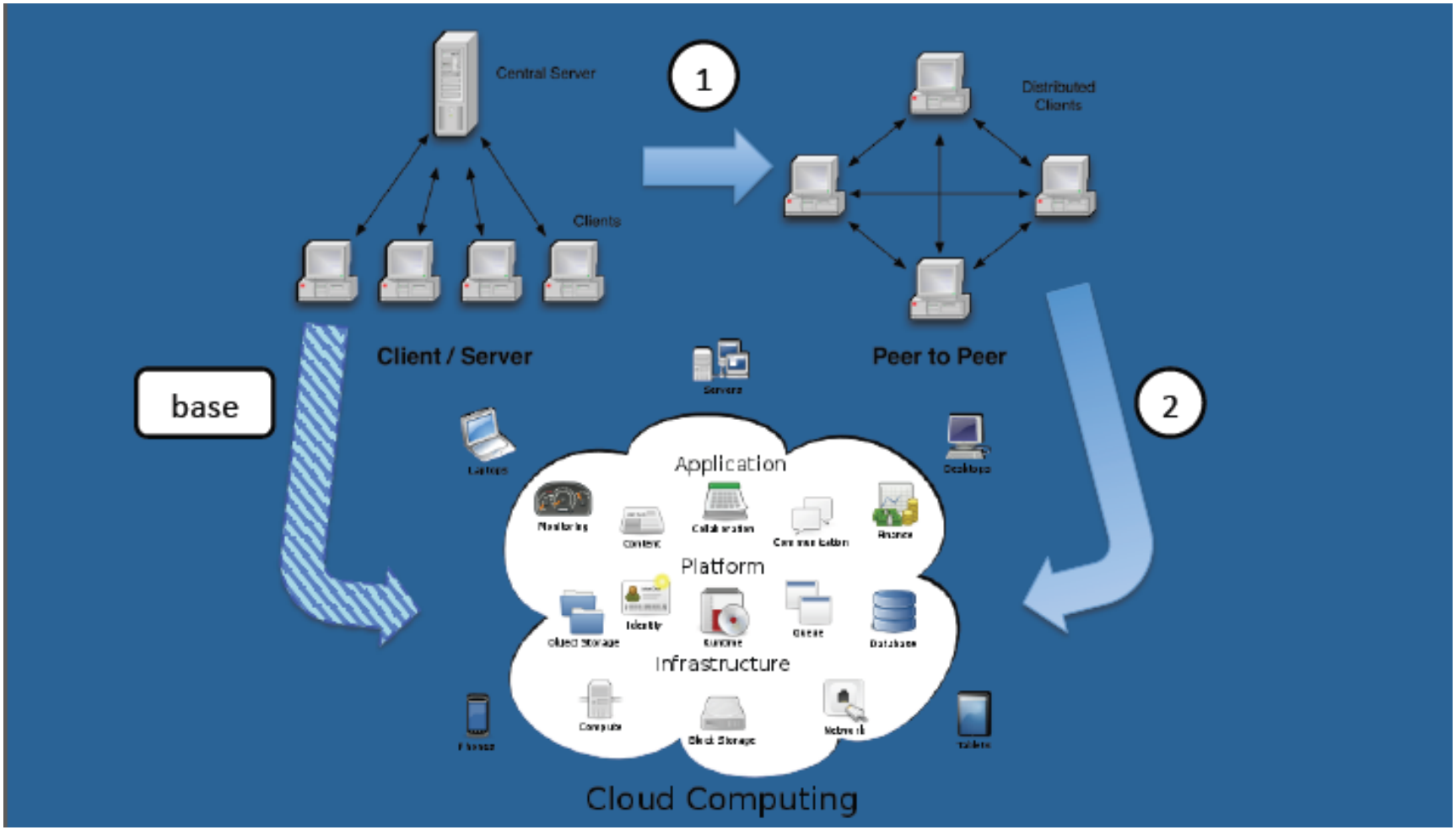
SD - Evolução


Sistema Operacional
RAM
CPU
I/O
Interface de Rede
Sistema Operacional
RAM
CPU
I/O
Interface de Rede
Computação Distribuída

SD - Evolução


Sistema Operacional
RAM
CPU
I/O
Computação Centralizada
Ou qualquer combinação entre elas !!!
Sistema Operacional
RAM
CPU
I/O
Concorrência
Sistema Operacional
RAM
CPU1
CPUn
I/O
Computação Paralela
Sistema Operacional
RAM
CPU
I/O
Interface de Rede
Sistema Operacional
RAM
CPU
I/O
Interface de Rede
Computação Distribuída

Computação Distribuída



Computação Distribuída

SD - Definição

*Um **sistema distribuído** é aquele no qual componentes localizados em uma **rede de computadores** se comunicam e coordenam suas ações através da passagem (troca) de mensagens.*

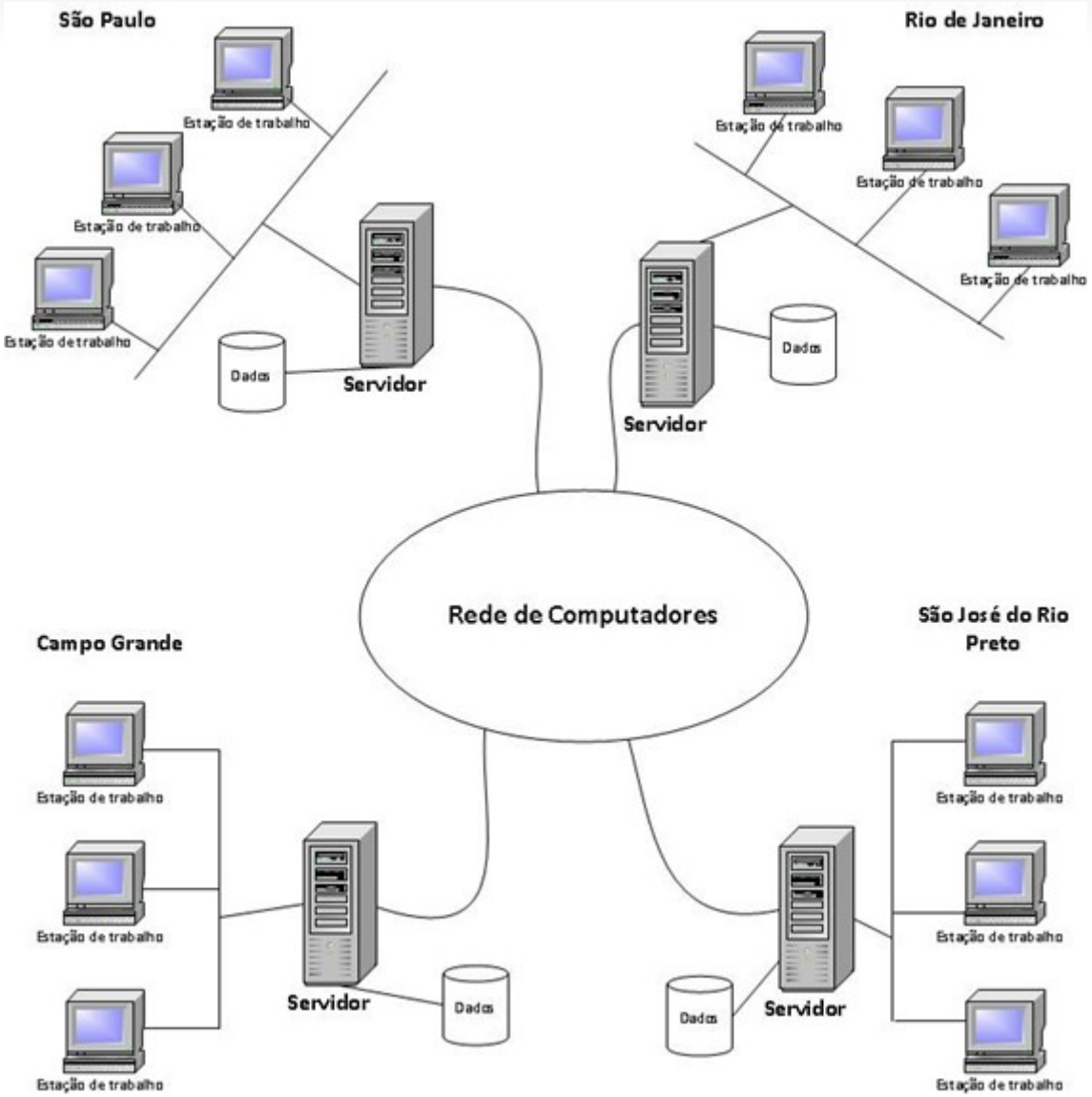
Coulouris et al., 2012

SD - Definição

Coleção de computadores independentes que aparecem para os usuários do sistema como um único computador.

Tanenbaum & van Steen

Sistemas Distribuídos...



Características Marcantes

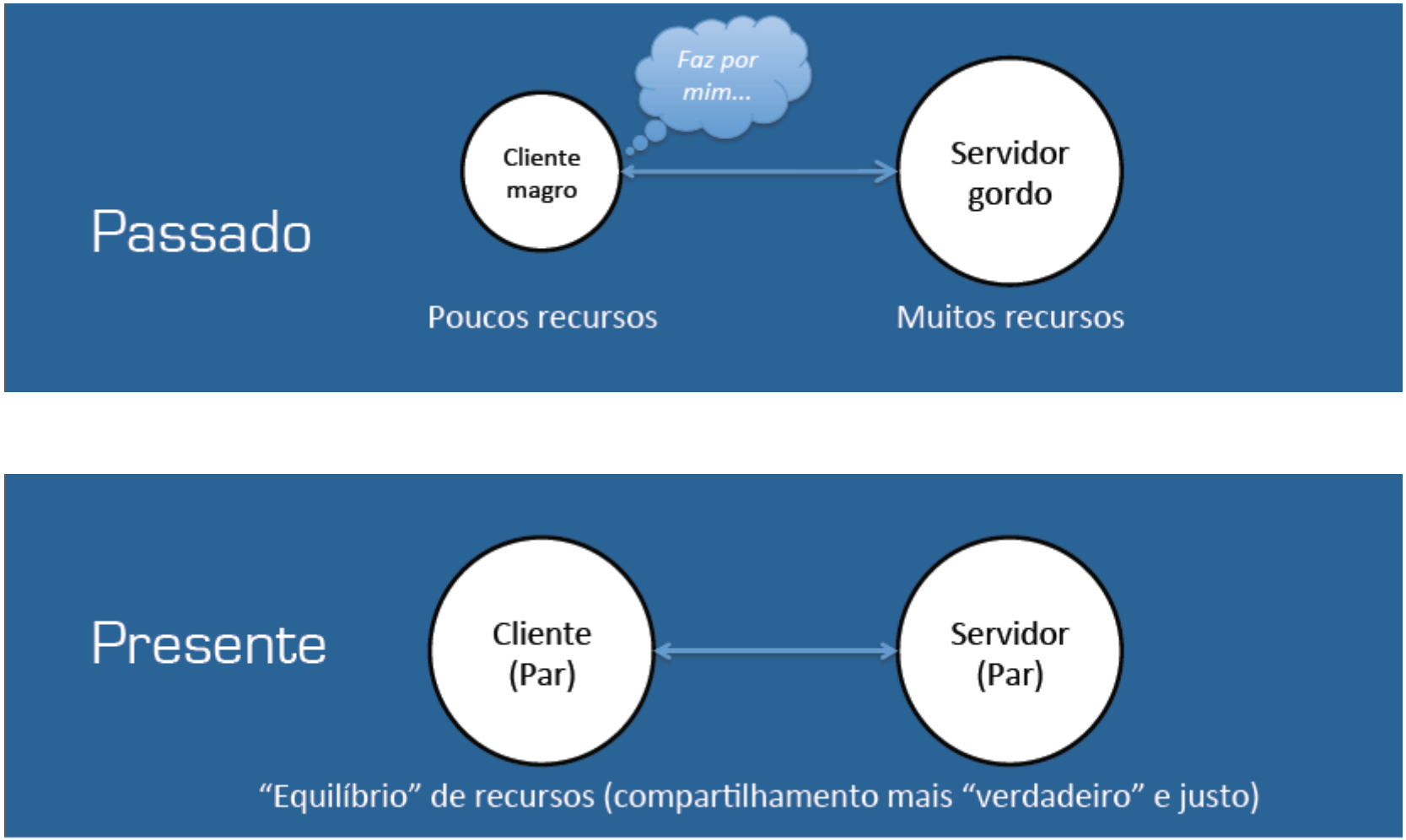
- Concorrência de componentes
- Falta de um relógio global
- Componentes falham de forma independente –falha parcial

Tendências-Chave

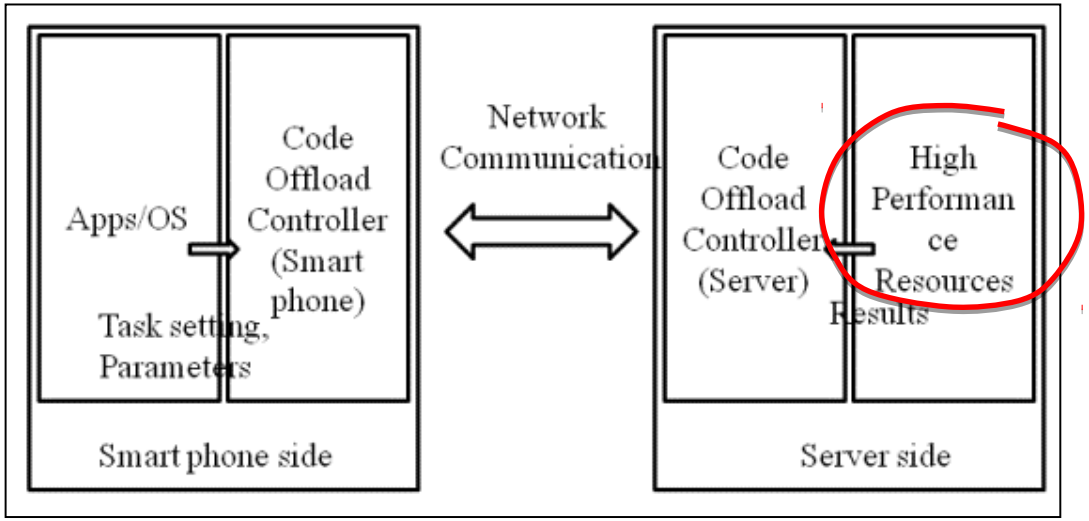
- O que move os Sistemas Distribuídos hoje?
  - **Pervasividade** das redes
  - Computação móvel e **ubíqua**
  - Importância crescente de sistemas **multimídia** (distribuídos)
  - Sistemas distribuídos como **utilidade (serviço)** – *Cloud Computing* e seus modelos (*IaaS, PaaS, SaaS*)
  - **Analytics**

- Principal motivação: **compartilhamento de recursos**

Compartilhamento de Recursos

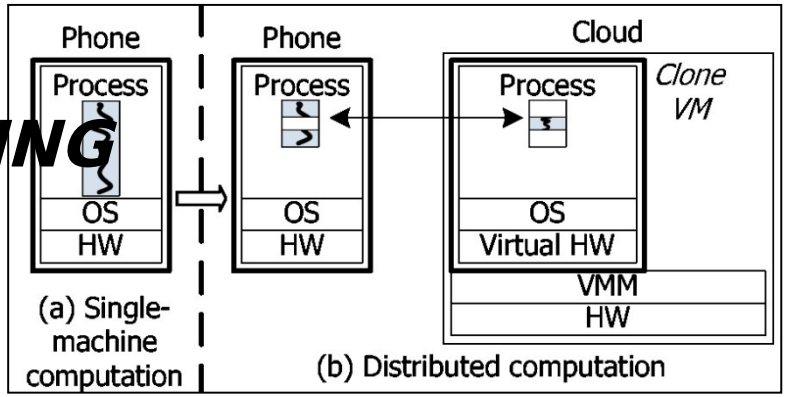


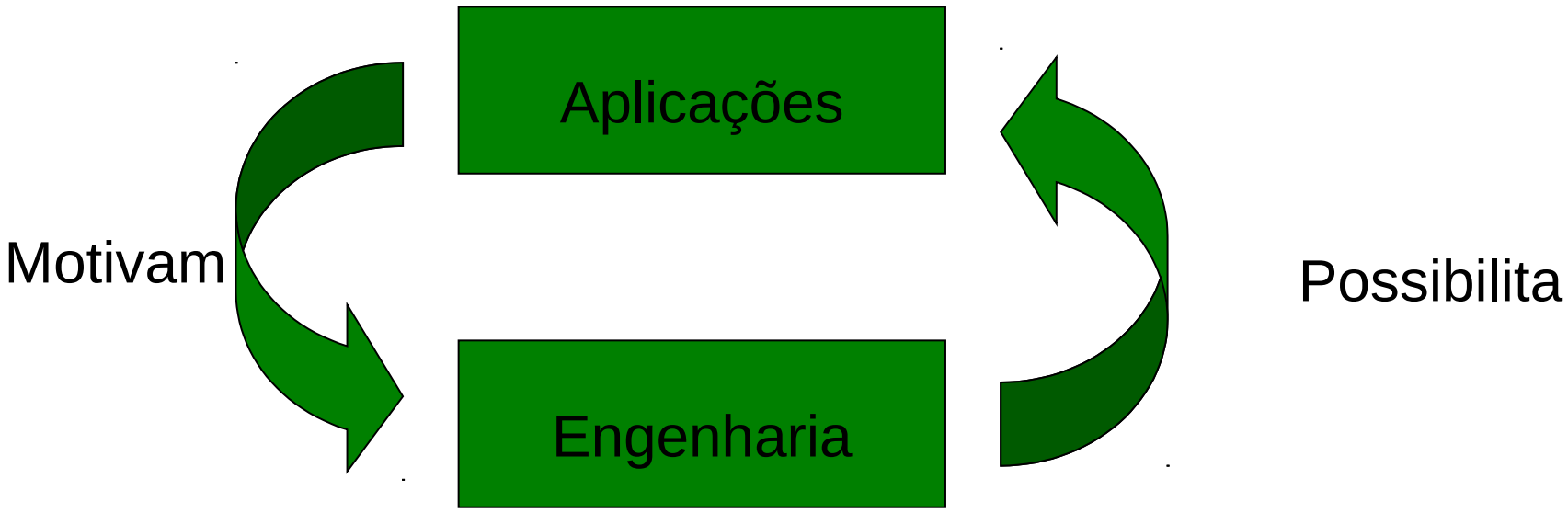
Computação *offloading*



MOBILE CLOUD COMPUTING

Como se faz?




Aplicações
Motivam
Possibilita
Engenharia

Objetivos de um projeto de SD

- Eficiência – desempenho produtivo
- Robustez – resistência a falha
  - Disponibilidade: o sistema está no ar quando preciso
  - Confiabilidade: o sistema não falha por um longo período de tempo
    - E quando falha, a falha não provoca uma “catástrofe”
- Consistência – mesma visão de dados

Desafios

- Heterogeneidade
- Abertura
- Segurança
- Escalabilidade / Elasticidade
- Tratamento de falha
- Concorrência
- Transparência

Conceitos-Chave

- Distribuição
- Comunicação
- Complexidade
- Heterogeneidade

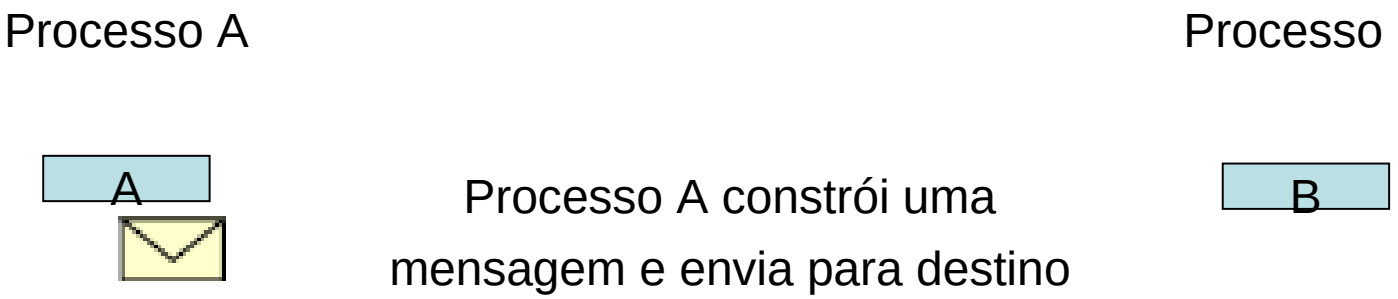
Transparência

Transparências

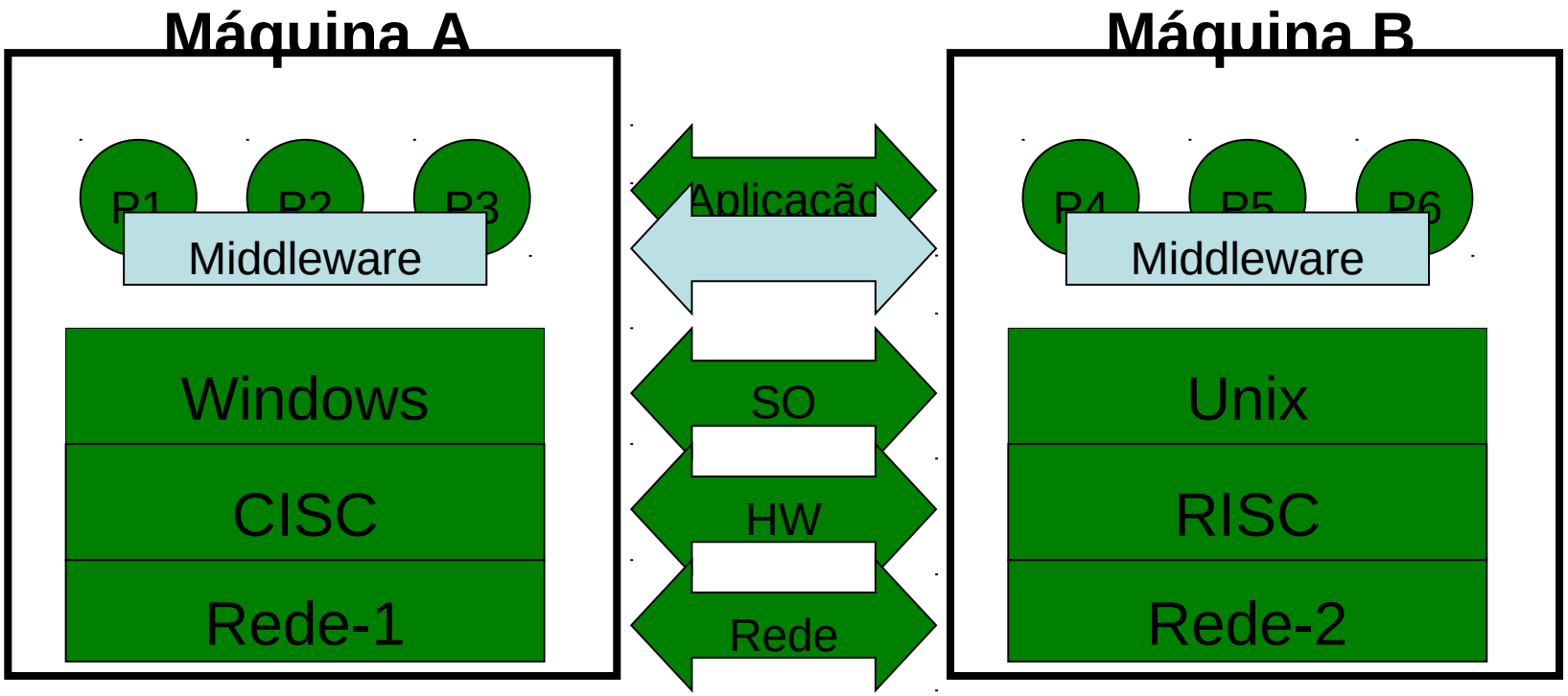
- **Localização**: esconde onde o recurso está localizado
- Acesso: operações idênticas para acesso local e remoto
- Migração: esconde que um recurso pode se mover para outra localização
- Relocação: esconde que um recurso pode ser movido para outra localização enquanto está em uso
- Concorrência: compartilhamento de recursos sem interferência entre processos concorrentes
- **Falha**: esconde a falha e recuperação de um recurso
- Replicação: esconde de usuários ou programadores de aplicação a existência de réplicas de recursos

Localização + acesso = transparência de rede

Comunicação



- O que é enviado/recebido pela rede?
  - Pacotes (sequência de bytes)
- Como o processo B entende os dados que chegam pela rede?
  - Protocolo / Regras (processos falam o “mesmo idioma”)



Infraestrutura de Software

Middleware - Definição

Middleware pode ser visto como...

um conjunto reusável, expansível de serviços e funções ...

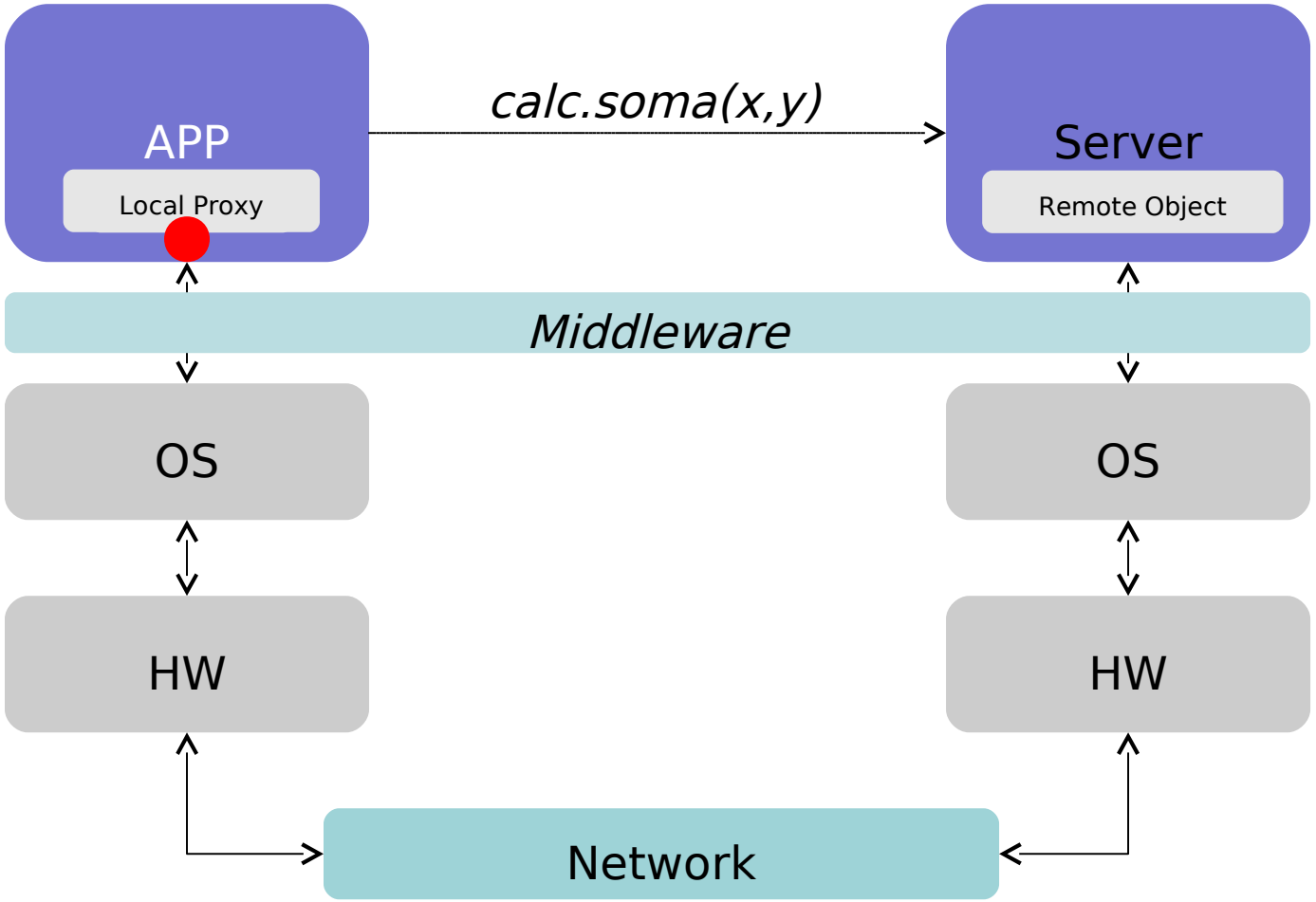
que são comumente necessários por parte de várias aplicações distribuídas ...

para funcionarem bem em um ambiente de rede.

Invocação Remota e Comunicação Indireta

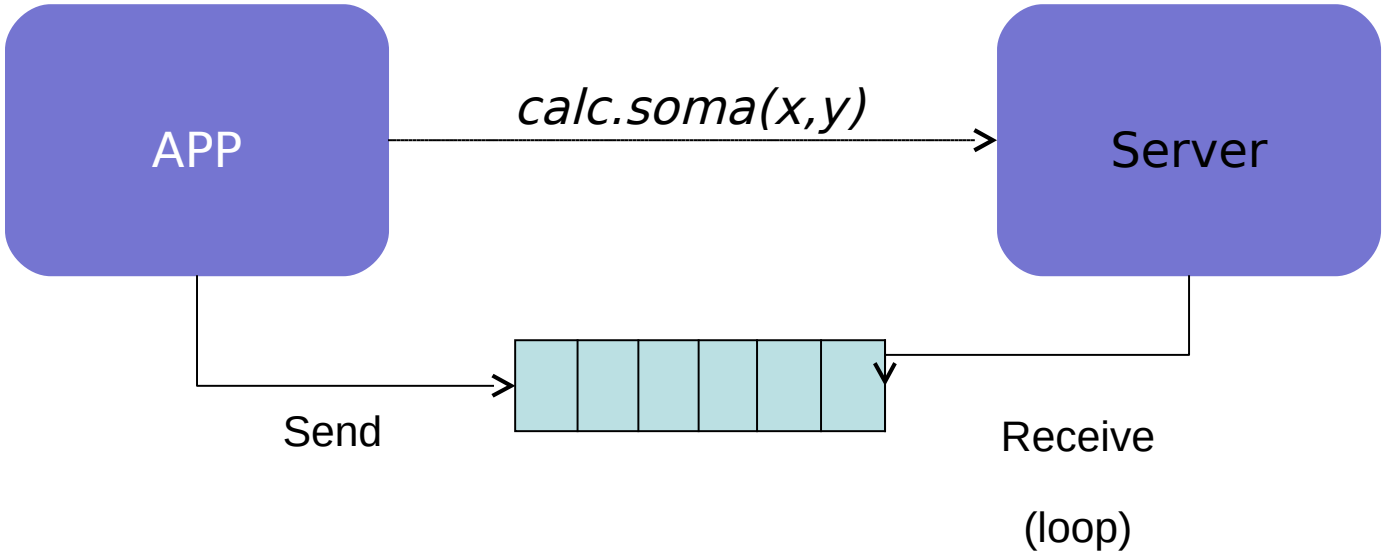
Middleware

```
int soma(int x, int y){
  Connection to Server
  ...
}
```



Middleware Orientado à Objetos (MOO)

Middleware

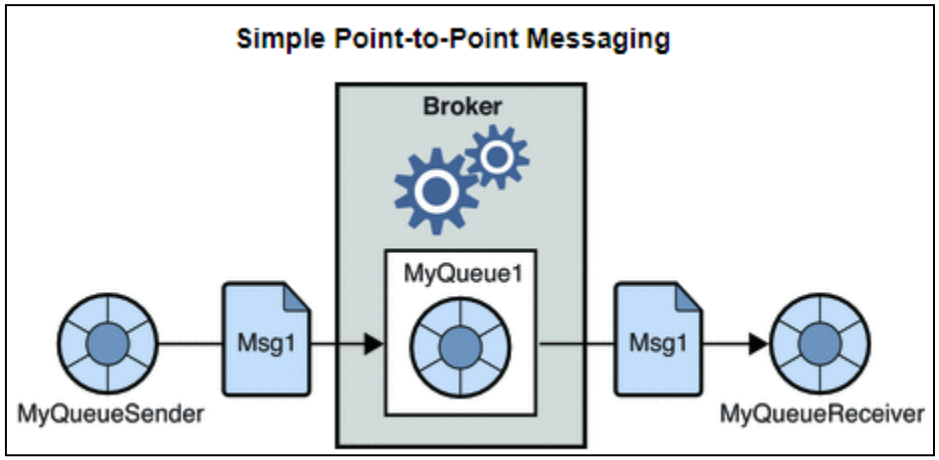


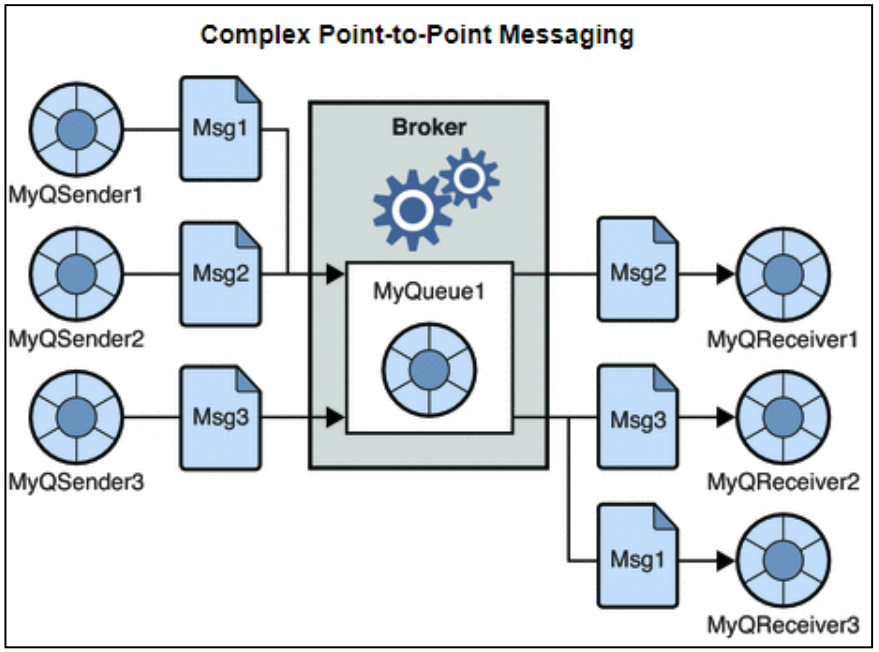
Protocolos: *mqtt, amqp*

Chamadas Assíncronas → “livre” para executar outras ações

Middleware Orientado à Mensagens (MOM)

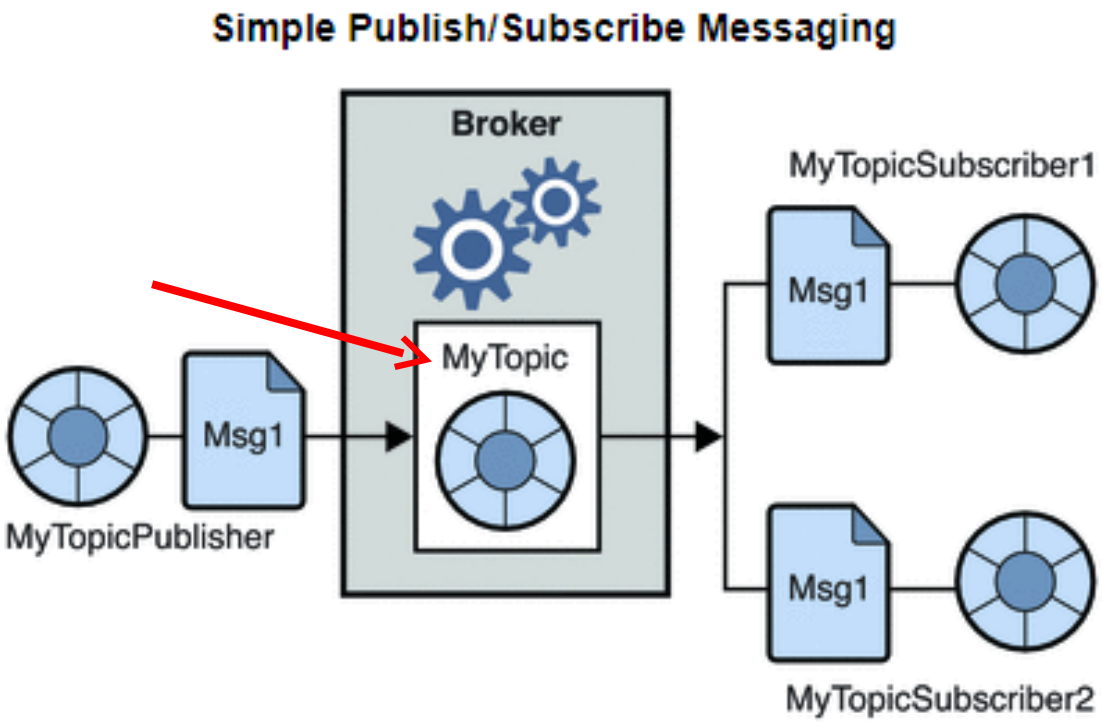
MOM


Simple Point-to-Point Messaging
Broker
MyQueue1
Msg1
Msg1
MyQueueSender
MyQueueReceiver


Complex Point-to-Point Messaging
Broker
MyQueue1
Msg1
Msg2
Msg3
Msg2
Msg3
Msg1
MyQSender1
MyQSender2
MyQSender3
MyQReceiver1
MyQReceiver2
MyQReceiver3

MOM - Publish/Subscribe

Pattern: Publish-Subscribe



Topic = named logical channels

MOM – Topic vs. Queue

Queues

*A single message will be received by exactly **one consumer**. If there are no consumers available at the time the message is sent it **will be kept until a consumer is available** that can process the message. If a consumer receives a message and does not acknowledge it before closing then the message will be redelivered to another consumer.*

Topics

*Topic implements publish and subscribe semantics. When you publish a message it **goes to all the subscribers who are interested** - so zero to many subscribers will receive a copy of the message. Only subscribers who had an active subscription **at the time the broker receives the message** will get a copy of the message.*

Fonte: JMS specification

MOM – Message Queue

MOM - Publish/Subscribe

ActiveMQ™

The Apache Software Foundation
http://www.apache.org/

Index | Download | JavaDocs More... | Source | Forums | Support

Download ActiveMQ 5.15.0 Today!

Apache ActiveMQ ™ is the most popular and powerful open source messaging and Integration Patterns server.

Apache ActiveMQ is fast, supports many Cross Language Clients and Protocols, comes with easy to use Enterprise Integration Patterns and many advanced features while fully supporting JMS 1.1 and J2EE 1.4. Apache ActiveMQ is released under the Apache 2.0 License

Grab yourself a Download, try our Getting Started Guide, surf our FAQ or start Contributing and join us on our Discussion Forums.

Overview

- Index
- News
- New Features
- Getting Started
- FAQ
- Articles
- Books
- Download
- License

Search

value
Search

Sub Projects

- Artemis
- Apollo
- CMS
- NMS

Community

ActiveMq

1. activemq start
    1. activemq.xml
    - *netstat –na | find “61616”*
    - disable "Internet Connection Sharing (ICS)" service
2. http://localhost:8161/admin (admin:admin)
3. bin/activemq producer --message "My_message" --destination queue://UPE-CECDA --messageCount 1
4. activemq consumer --messageCount 50 --destination queue://UPE-CECDA

ActiveMq – Producer/Publisher

1. Create a *ConnectionFactory*
2. Create a Connection
3. Start a connection
4. Create a Session
5. Create the destination (Topic or Queue)
6. Create a *MessageProducer* from the Session to the Topic or Queue
7. Create a Message
8. Tell the producer to send the message

ActiveMq – Producer/Publisher

```
// Create a ConnectionFactory
ActiveMQConnectionFactory connectionFactory = new
ActiveMQConnectionFactory("tcp://localhost:61616");

// Create a Connection
Connection connection = connectionFactory.createConnection();
connection.start();

// Create a Session
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

// Create the destination (Topic or Queue)
Destination destination = session.createQueue("UPE-QUEUE");

// Create a MessageProducer from the Session to the Topic or Queue
MessageProducer producer = session.createProducer(destination);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);

TextMessage message = session.createTextMessage("Hello Poli");

// Tell the producer to send the message
producer.send(message);
```

Thread...

ActiveMq – Consumer/Subscriber

1. Create a *ConnectionFactory*
2. Create a *Connection*
3. Create a *Session*
4. Create the destination (Topic or Queue)
5. Create a *MessageConsumer* from the Session to the Topic or Queue
6. Set a *listener* to this consumer
7. Start the *connection*

ActiveMq – Consumer/Subscriber

```
// Create a ConnectionFactory
ActiveMQConnectionFactory connectionFactory = new
ActiveMQConnectionFactory("tcp://localhost:61616");

// Create a Connection
Connection connection = connectionFactory.createConnection();

// Create a Session
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

// Create the destination (Topic or Queue)
Destination destination = session.createQueue("UPE-QUEUE");

// Create a MessageConsumer from the Session to the Topic or Queue
MessageConsumer consumer = session.createConsumer(destination);

consumer.setMessageListener(new MessageListener() {
     @Override
     public void onMessage(Message m) {
          try {
              System.out.println(((TextMessage)m).getText());
              //m.acknowledge();
          } catch (JMSException e) {}

     }
     });

connection.start();
```

Thread...

ActiveMq – BlobMessage

Blob Messages

A common requirement these days is to send around massive files for processing by consumers. Folks want to take advantage of the message broker's features such as reliable, transactional load balancing of queues with smart routing but still manage to deal with huge logical files.

So we are introducing a BlobMessage API which allows massive BLOBs (Binary Large OBjects) to be sent around in some out-of-band transport mechanism. Possible out-of-band mechanisms could be HTTP or FTP or SCP or some other point-to-point protocol.

There are now new createBlobMessage() methods on the ActiveMQSession that you can use for sending BLOBs.

Sending BlobMessages

You can send a URL around the JMS network, such as a file or URL which exists on some shared file system or web server using the following code

```
BlobMessage message = session.createBlobMessage(new URL("http://some.shared.site.com");
producer.send(message);
```

Or if you are creating files or streams dynamically on the client you may want to upload the file to the broker or some server (Jetty, FTP, WebDav or whatever). In which case you'd use one of the following methods

```
// lets use a local file
BlobMessage message = session.createBlobMessage(new File("/foo/bar");
producer.send(message);
```

```
// lets use a stream
InputStream in = ...;
BlobMessage message = session.createBlobMessage(in);
producer.send(message);
```

Receiving BlobMessages

A BlobMessage is a regular JMS message so it can be received just like any other message...

```
public class MyListener implements MessageListener {
  public void onMessage(Message message) {
    if (message instanceof BlobMessage) {
      BlobMessage blobMessage = (BlobMessage) message;
      InputStream in = blobMessage.getInputStream();

      // process the stream...
    }
  }
}
```

http://activemq.apache.org/blob-messages.html

ActiveMq – Clustering

How do I configure distributed queues or topics

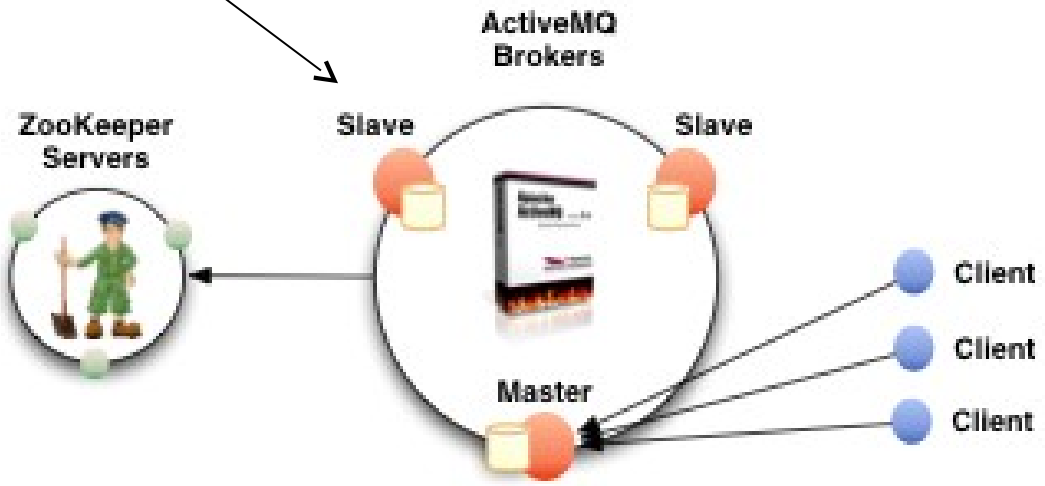
You don't need to explicitly configure distributed queues or topics as any queue or topic is automatically distributed across other brokers when the brokers are configured in either a store and forward network or a master/slave cluster.

So you just need to connect brokers together to form either

- a Store and Forward Network of Brokers, which means the messages travel from broker to broker until they reach a consumer; with each message being owned by a single broker at any point in time
- a Master/Slave Cluster, which means all messages are replicated across each broker in the master/slave cluster

Also see

- How do I configure the queues I want
- How do distributed queues work
- Networks of Brokers
- MasterSlave



Exercício – Now !

T-Drive trajectory data sample

August 12, 2011

Download PDF

BibTex

Authors

Yu Zheng

Published In

T-Drive sample dataset

Abstract Related Info

This is a sample of T-Drive trajectory dataset that contains a one-week trajectories of 10,357 taxis. The total number of points in this dataset is about 15 million and the total distance of the trajectories reaches 9 million kilometers.

Please cite the following papers when using the dataset:
[1] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. Driving with knowledge from the physical world. In The 17th ACM SIGKDD international conference on Knowledge Discovery and Data mining, KDD'11, New York, NY, USA, 2011. ACM.

ActiveMq

1. Um pouco mais de ActiveMq...
    1. Acknowledge
    2. Properties of the Message
        - createConsumer(destination, "(idade < 50)")
    3. DurableSubscriber
    4. Message Expiration time / setTimeToLive

1. Ok. Sabemos o que é um SD.
2. Sabemos usar um middleware (*message queue)*

Mas... Como processar os dados?

Como processar esses dados?

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

2004

Abstract

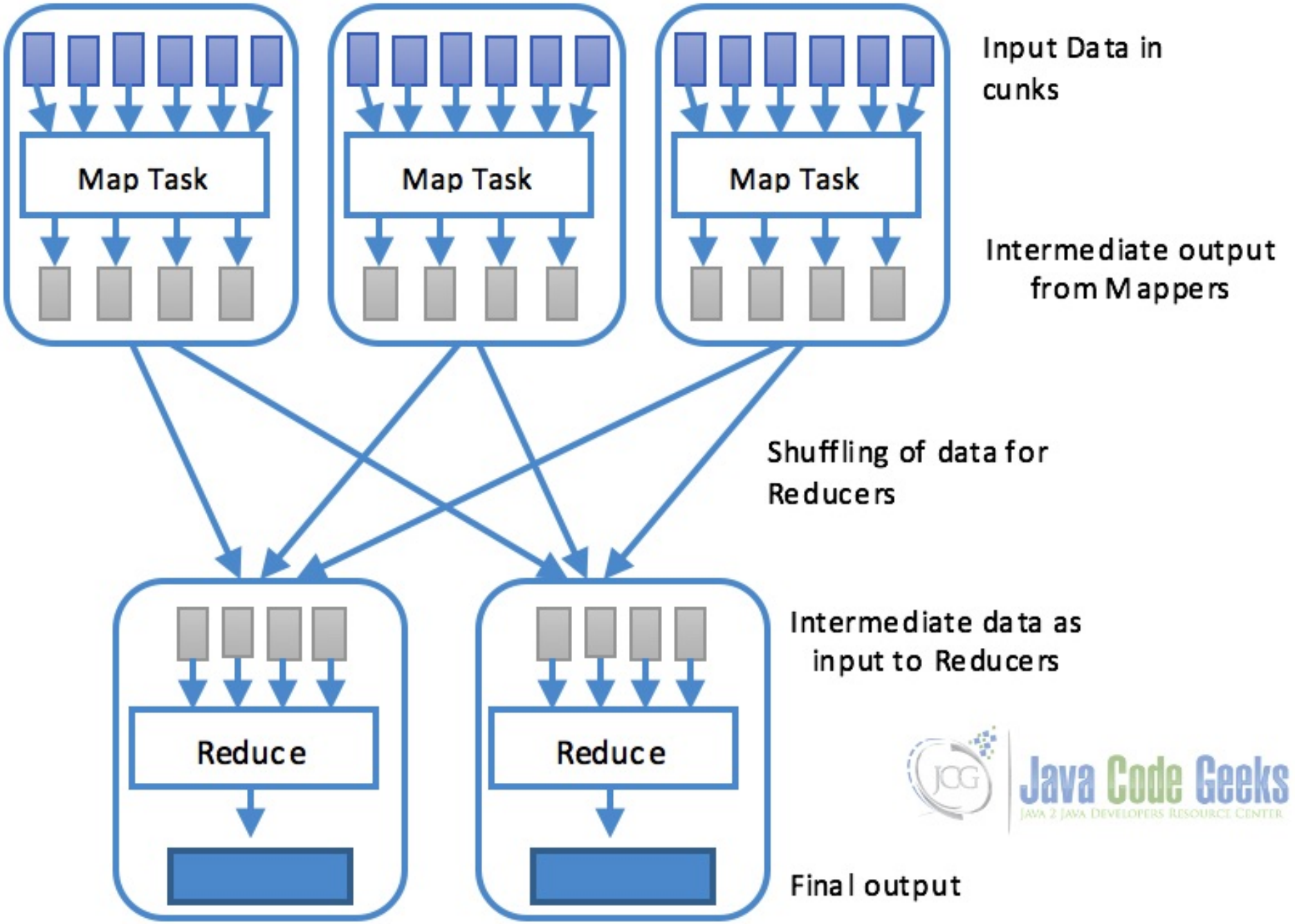
MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling ma-

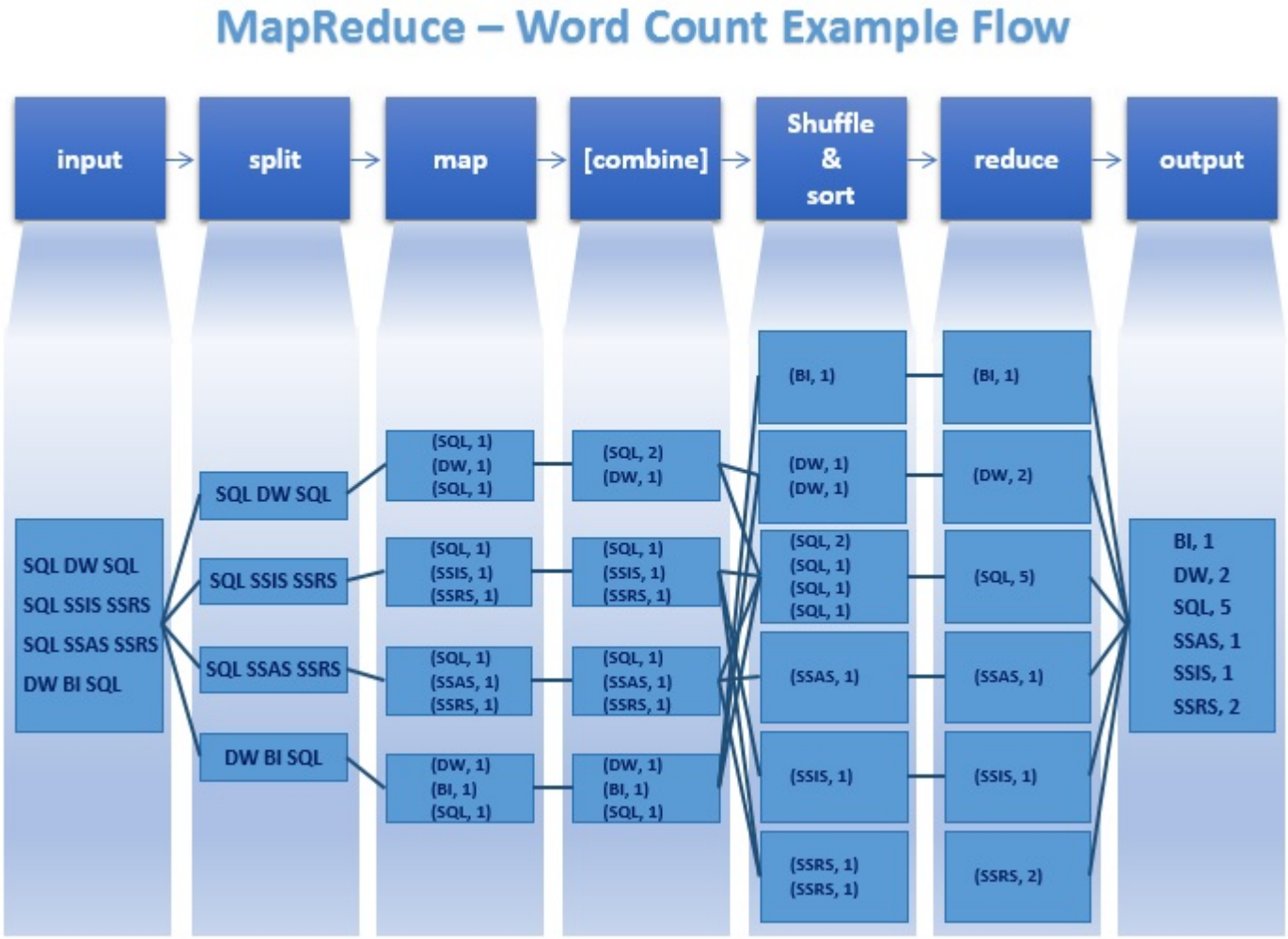
given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp

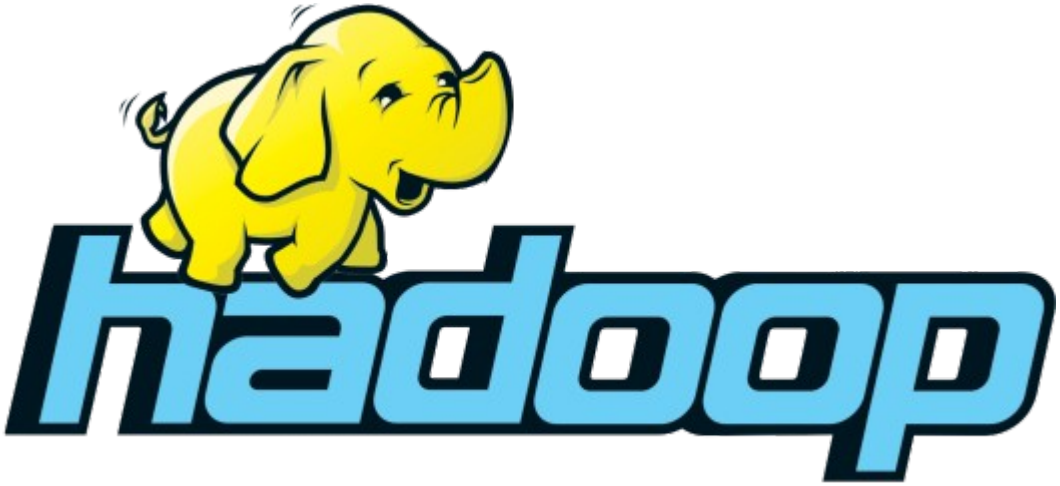
MapReduce é um modelo de programação, e framework introduzido pelo Google para suportar computações paralelas em grandes coleções de dados em clusters de computadores


Map Task
Map Task
Map Task
Input Data in cunks
Intermediate output from Mappers
Shuffling of data for Reducers
Reduce
Reduce
Intermediate data as input to Reducers
Final output
Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

Map Reduce – Word Count Example



Map Reduce



(In-memory)

Map Reduce

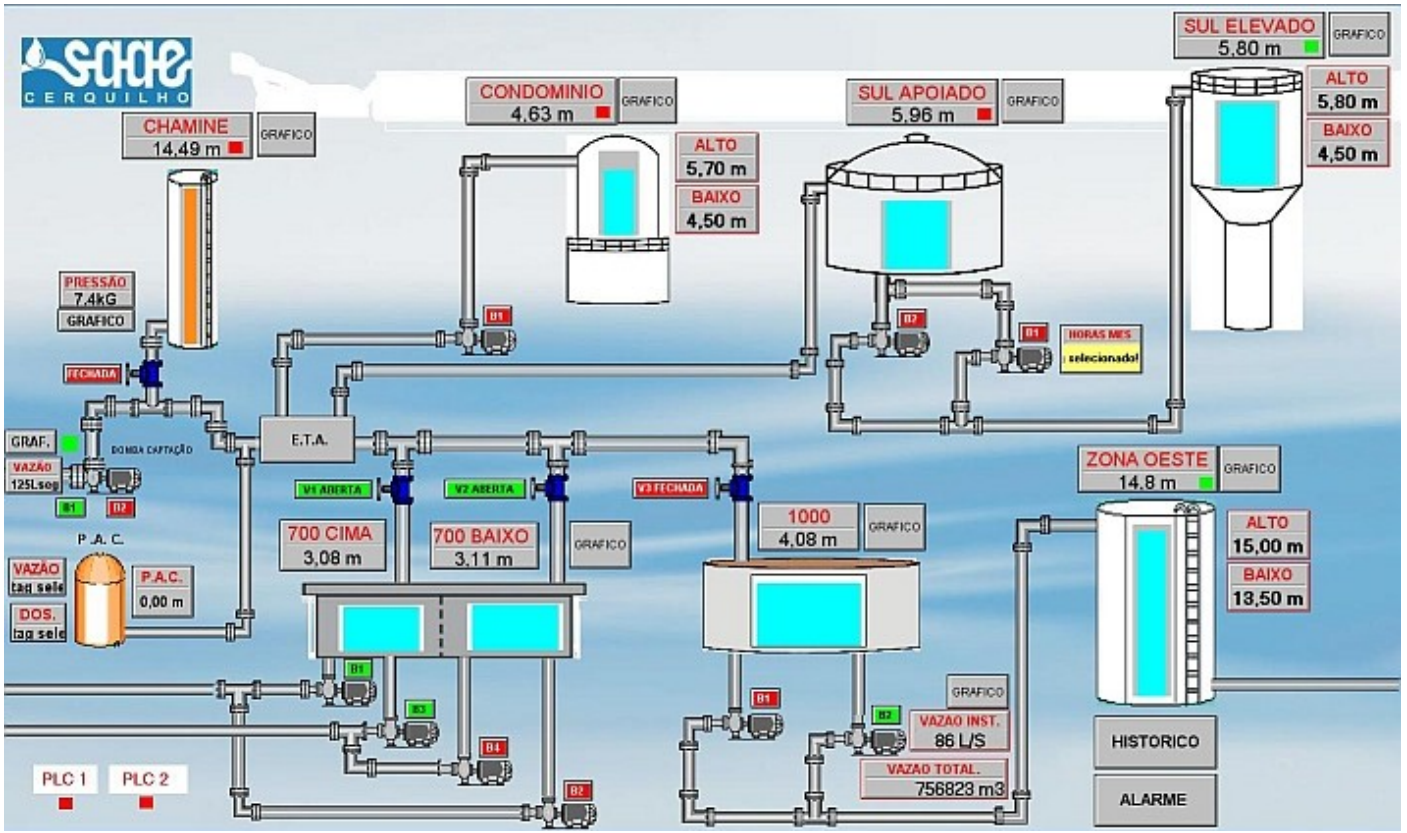
- But MR is Batch Processing…
  - V = Volume
  - Time is not an issue
  - Has access to all data
  - Might compute something big and complex
  - Is generally more concerned with throughput than latency of individual components of the computation
  - **Has latency measured in seconds/minutes**
    - Depends on the size of the data

LinkedIn

LinkedIn began processing “big data” on Apache Hadoop six years ago. As time passed, we recognized that some of our use cases couldn’t be implemented in Hadoop due to the large turn-around time that batch processing needed. We wanted our results to be calculated incrementally and available immediately at any time.

Around the same time, LinkedIn developed Apache Kafka, which is a low-latency distributed messaging system. Unlike Hadoop, which is optimized for throughput, Kafka is optimized for low-latency messaging. We built a processing system on top of Kafka to allow us to react to the messages- to join, filter, and count the messages. The new processing system, Apache Samza, solved our batch processing latency problem and has allowed us to process data in near real-time.

Cenários



SAAE Cerquilho

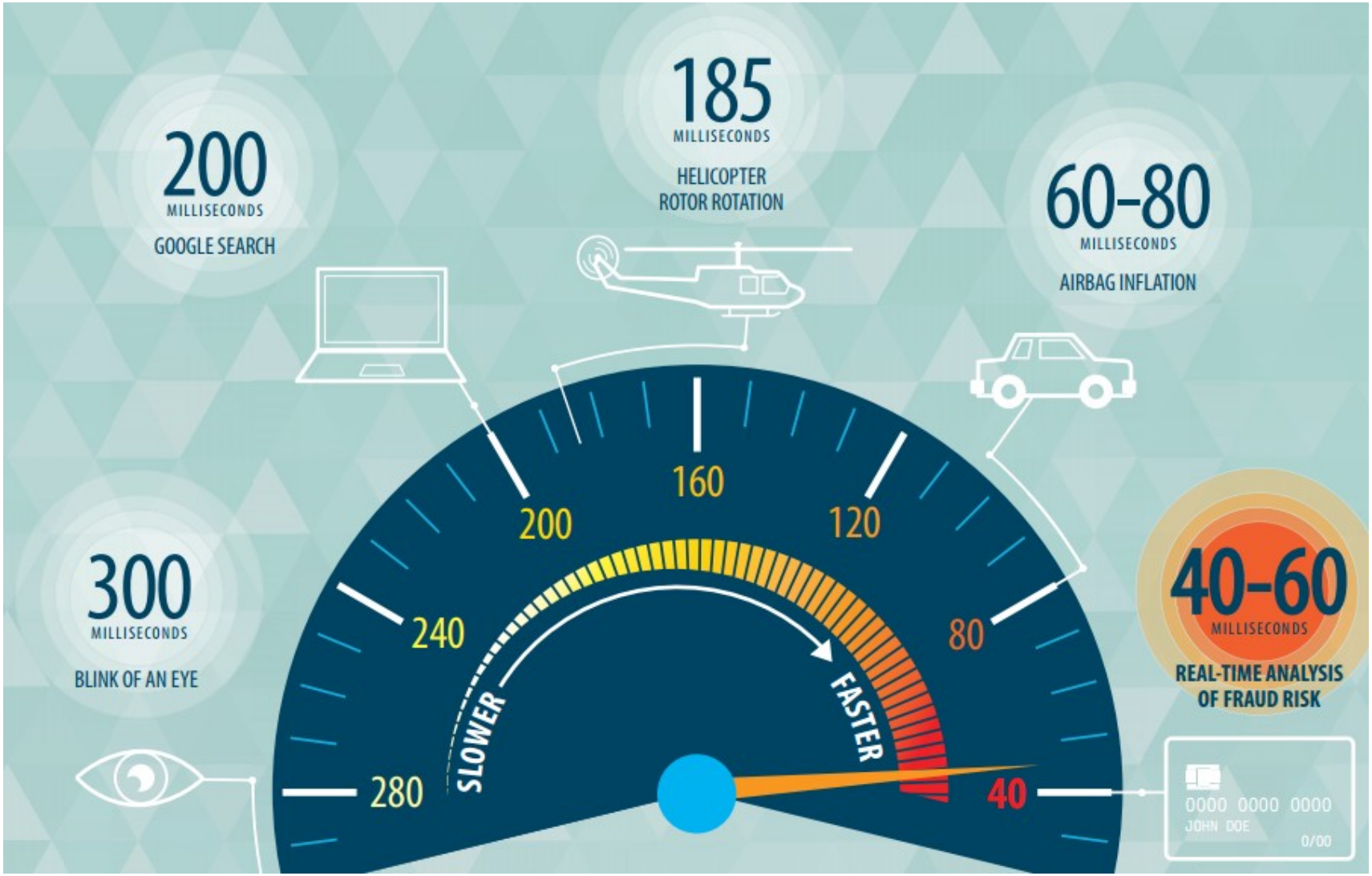
Cenários

Cenários

Network Intrusion Detection

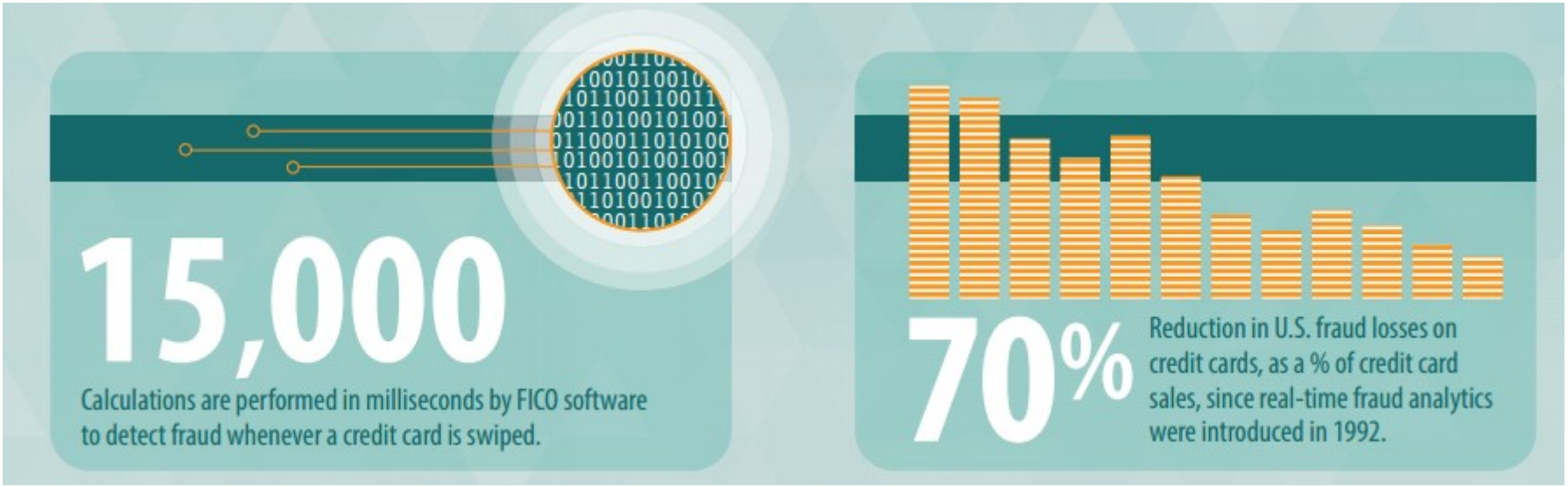
Cenários

Cenários - Números



fico.com

Cenários



fico.com

Cenários

O que esses cenários tem em comum?

Fluxo contínuo de dados

Information Flow Processing (IFP)

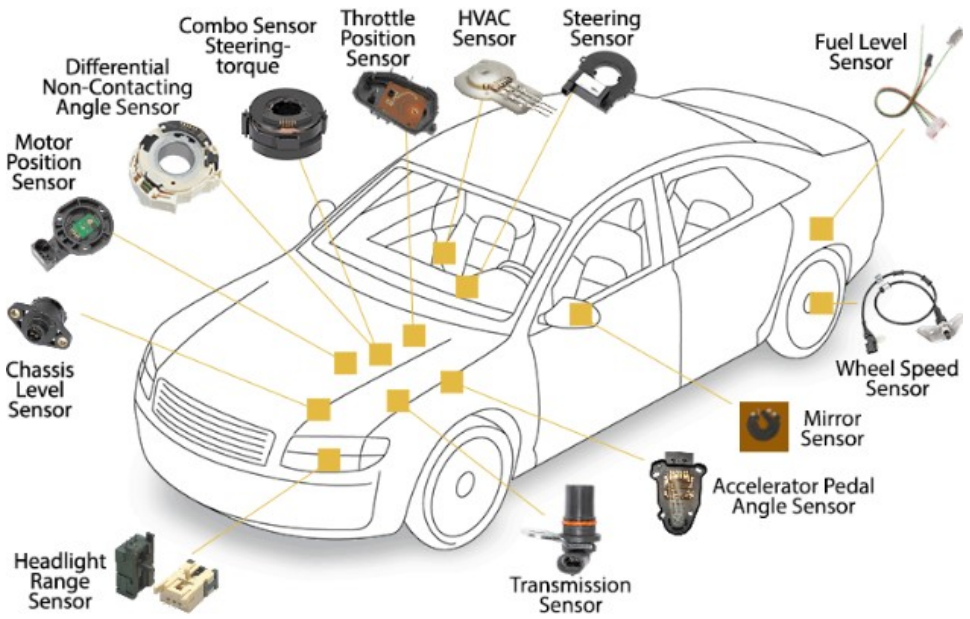
Batch resolve?

Falando em *Streams* (Fluxos)

Stream

Data Stream

Outros exemplos...



Streams...

- A large number of distributed applications requires *continuous* and *timely processing* of information as it flows from the periphery to the center of the system
  - Sequence is potentially infinite

- DBMS can hardly fullfil *timeliness requirements*
  - Data be stored and indexed...
  - ... To be processed when explicity asked by the users
    - Asynchronously with respect to its arrival
  - Timeliness and “Flow processing” justify the need of a new class of systems
    - **Data Stream Management Systems (DSMS)**

DBMS vs. DSMS

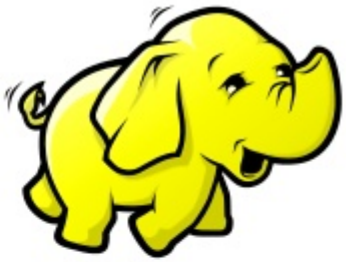
- Persistent relations
- Relatively low update rate
  - All data is stored
- One-time queries
- Random access
- “Unbounded” disk store
- No real-time services
- Data at any granularity

Human Active
Database-Passive
(HADP)

- Transient streams
- Continuously updated
  - Data processed and discarded
- Continuous queries
- Sequential access
- Bounded main memory
- Real-time requirements
- Data at fine granularity

Database Active
Human-Passive
(DAHP)

Latência



RPC

Stream Processing

Response latency

Milliseconds to minutes

Synchronous

Later. Possibly much later.

História

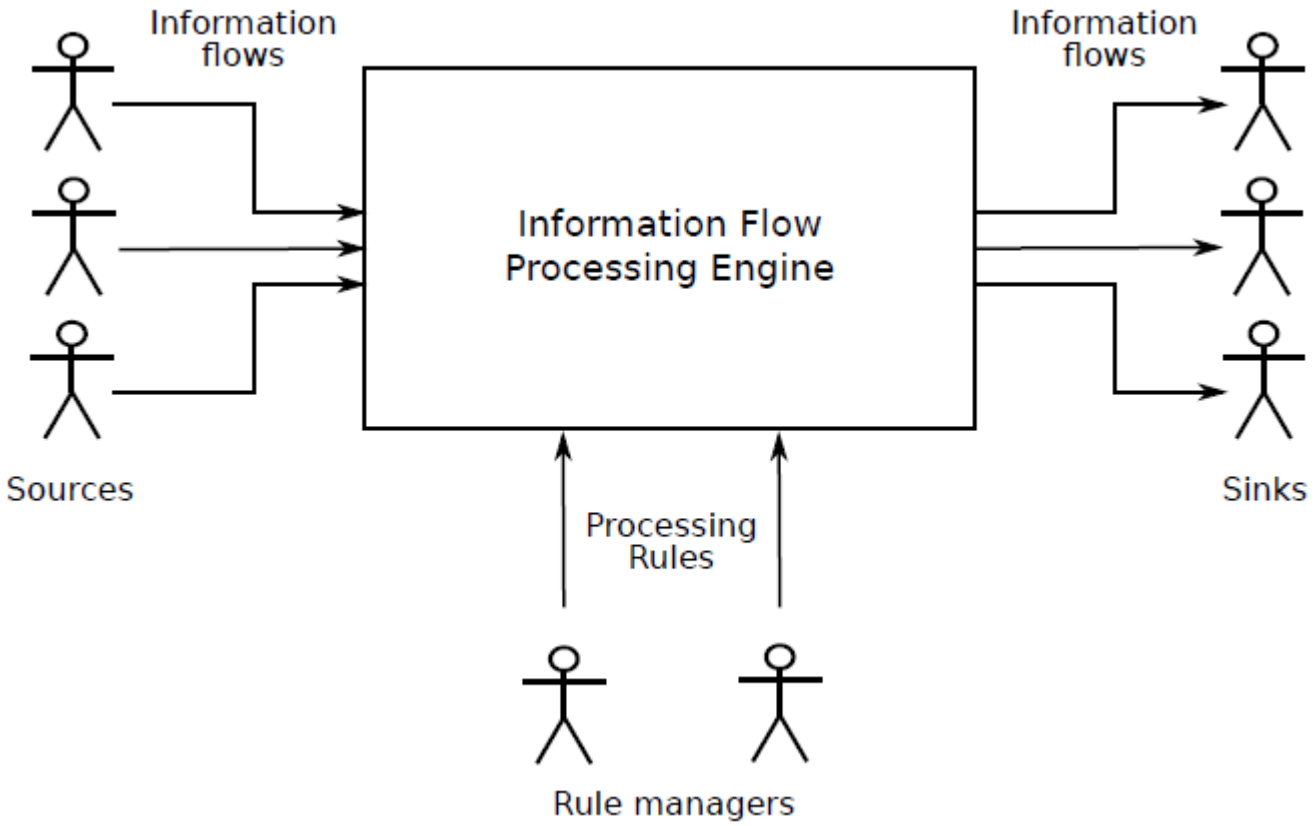
DBMS → Active DBMS → DSMS

all the relevant data is kept, and whose updates are relatively infrequent

- Differently from tables, streams are usually unbounded
- one-time processing is the typical mechanism used to deal with streams
  - size and time constraints make it difficul to store and process data stream elements after their arrival

IFPs

IFP (Information Flow Processing) Systems



selections, aggregates, joins

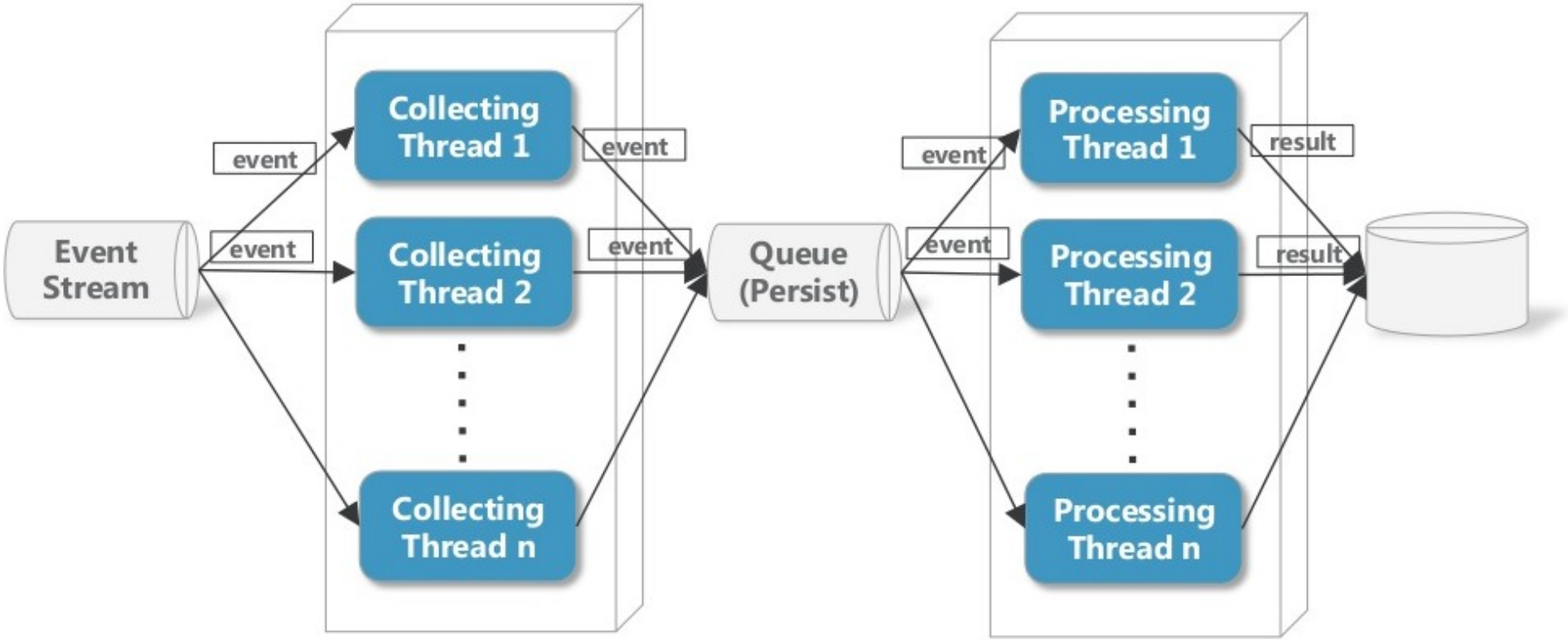
IFPs


Event Stream
event
Collecting/ Processing
result

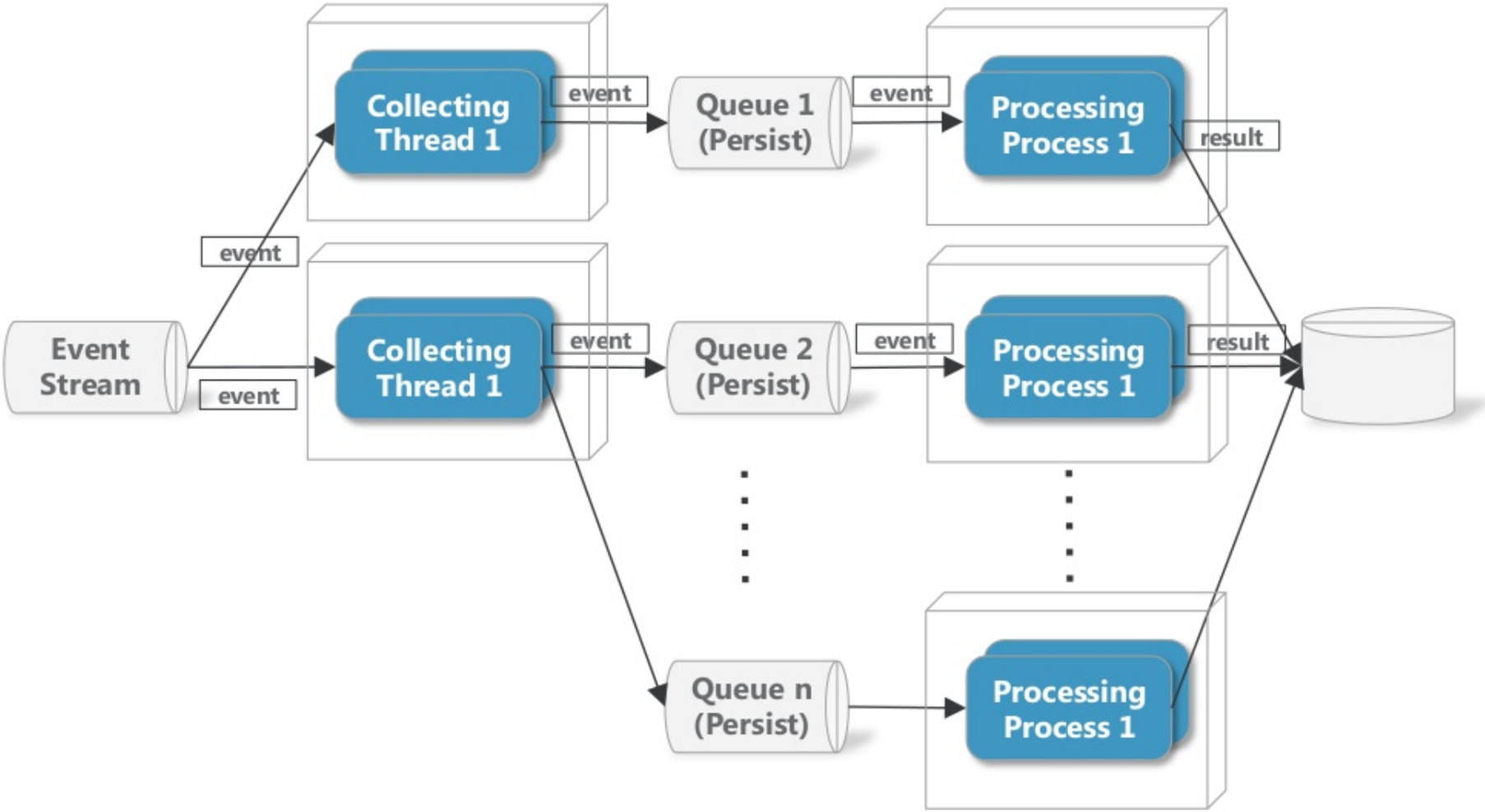

Event Stream
event
Collecting
event
Processing
result


Event Stream
event
Collecting
event
Queue (Persist)
event
Processing
result

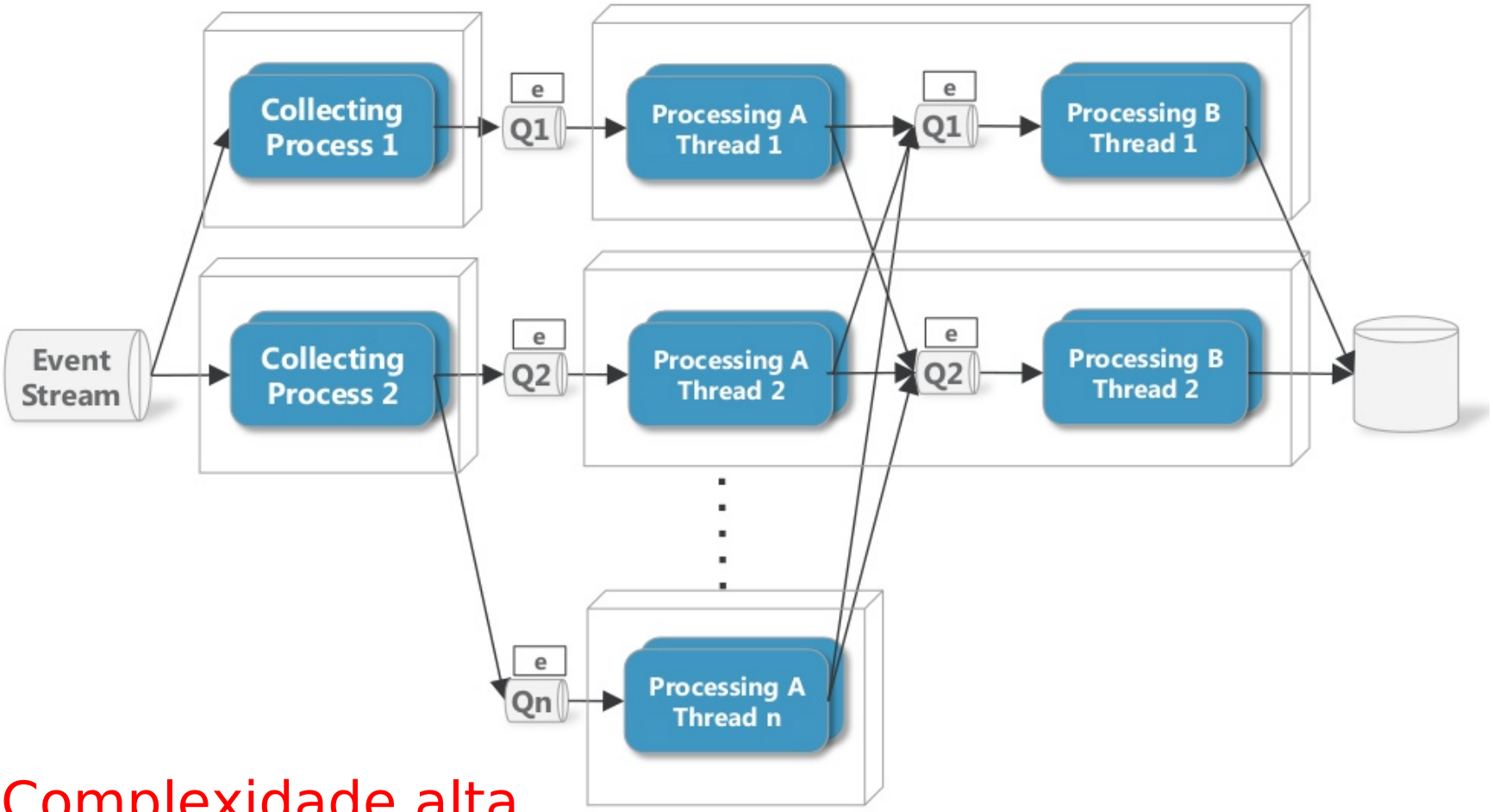
IFPs


Event Stream
event
event
Collecting Thread 1
Collecting Thread 2
Collecting Thread n
event
event
Queue (Persist)
event
event
Processing Thread 1
Processing Thread 2
Processing Thread n
result
result

IFPs


Event Stream
event
event
Collecting Thread 1
Collecting Thread 1
event
event
Queue 1 (Persist)
Queue 2 (Persist)
Queue n (Persist)
event
event
Processing Process 1
Processing Process 1
Processing Process 1
result
result

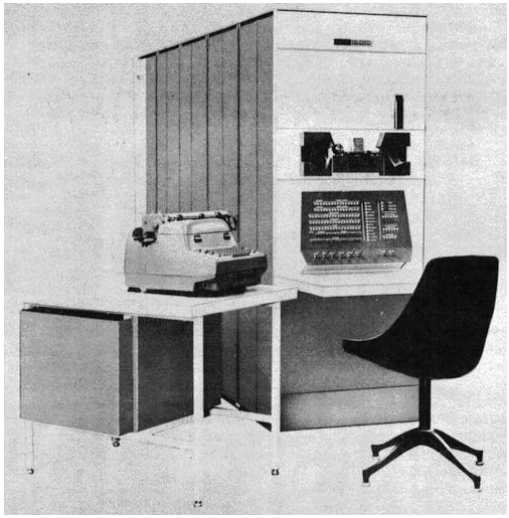
IFPs

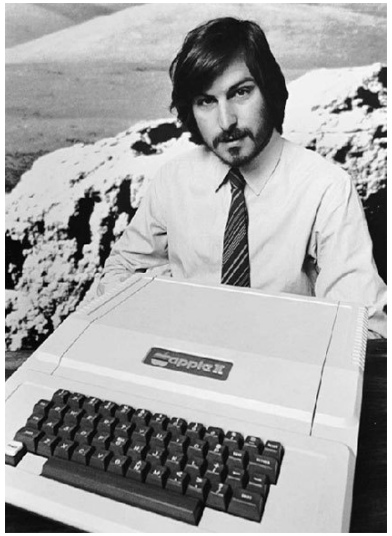


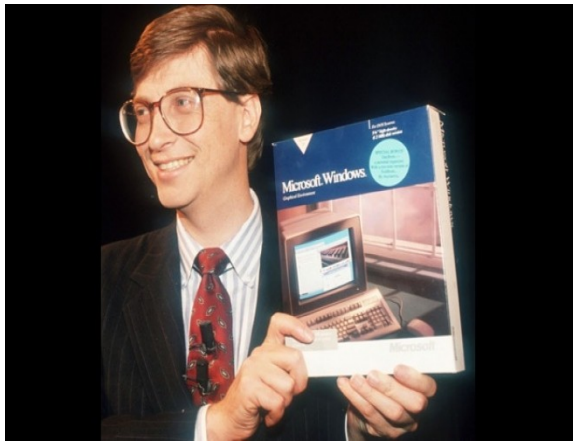
Complexidade alta

(Complexidade)

Sistemas Operacionais – Introdução ENIAC, 1946







Complexidade

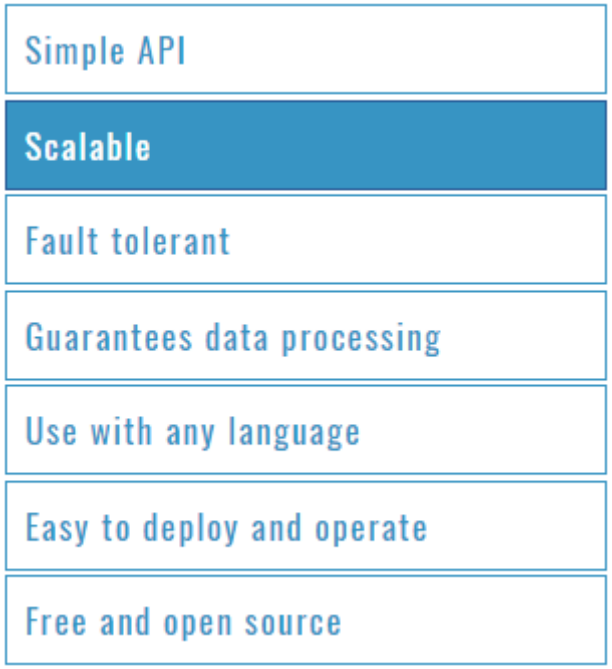
Precisamos de plataformas

Precisamos de infraestrutura

Apache Storm

Apache Storm is a free and open source distributed ***realtime computation system****. Storm makes it easy to reliably process unbounded streams of data,* ***doing for realtime processing what Hadoop did for batch processing****.*

Apache Storm



Storm topologies are inherently parallel and run across a cluster of machines. Different parts of the topology can be scaled individually by tweaking their parallelism. The "rebalance" command of the "storm" command line client can adjust the parallelism of running topologies on the fly.

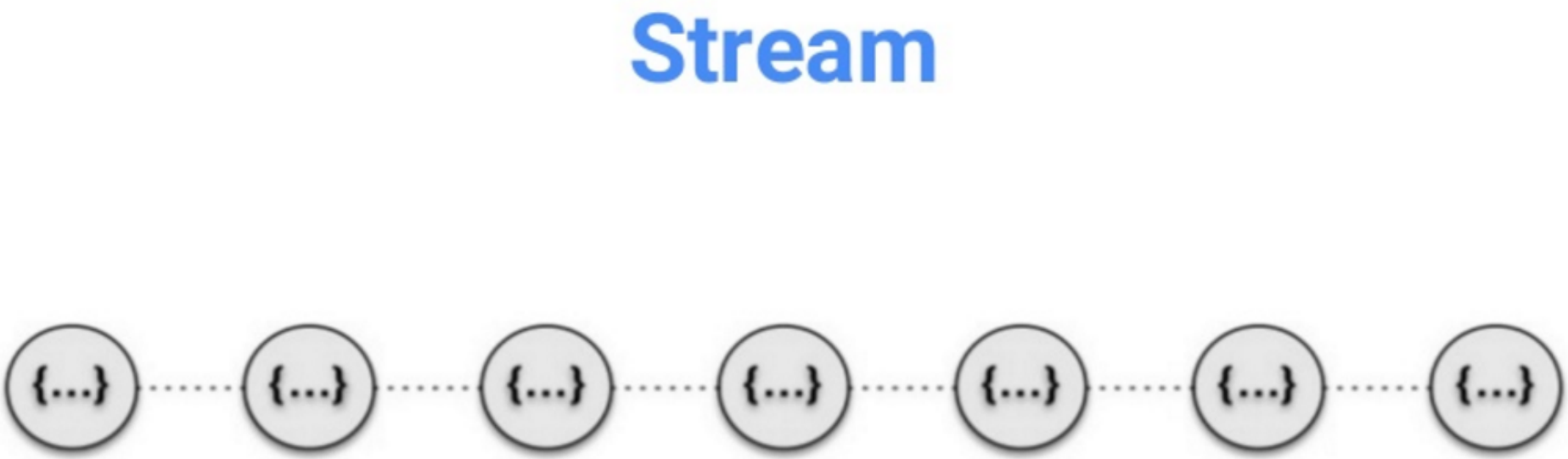
Storm's inherent parallelism means it can process very high throughputs of messages with very low latency. **Storm was benchmarked at processing one million 100 byte messages per second per node on hardware with the following specs**:

Processor: 2x Intel E5645@2.4Ghz

Memory: 24 GB

Apache Storm

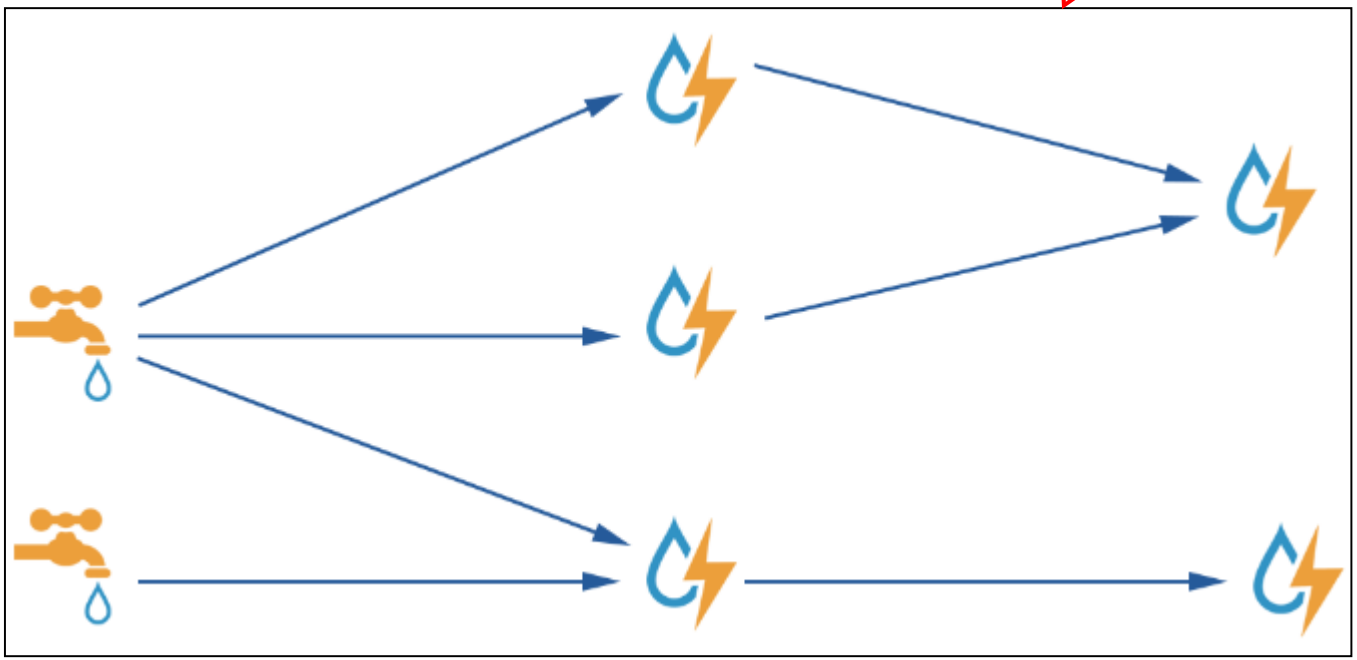
Apache Storm



Fonte: Sapienza University of Rome

Apache Storm

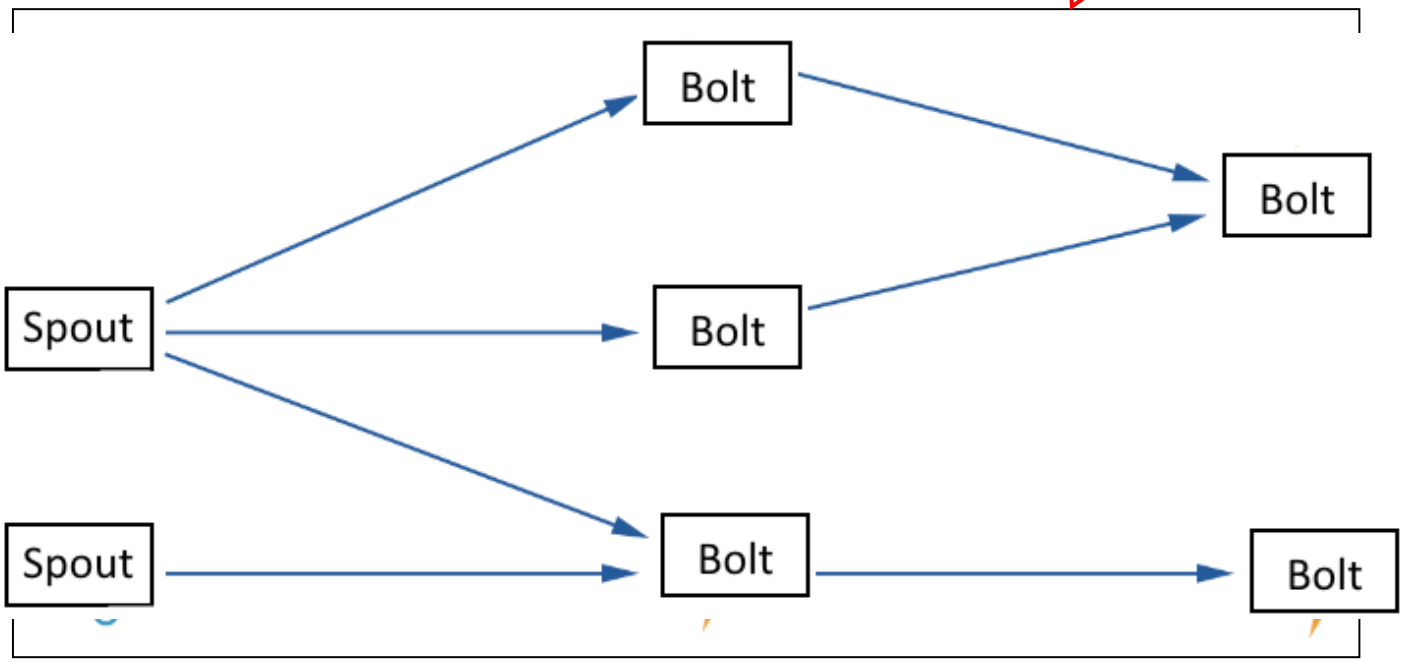
An application is defined in Storm through a **Topology** that describes its logic as a **DAG** of operators and streams



A Storm topology is analogous to a MapReduce job. One key difference is that a MapReduce job eventually finishes, whereas a **topology runs forever** (or until you kill it, of course)

Apache Storm

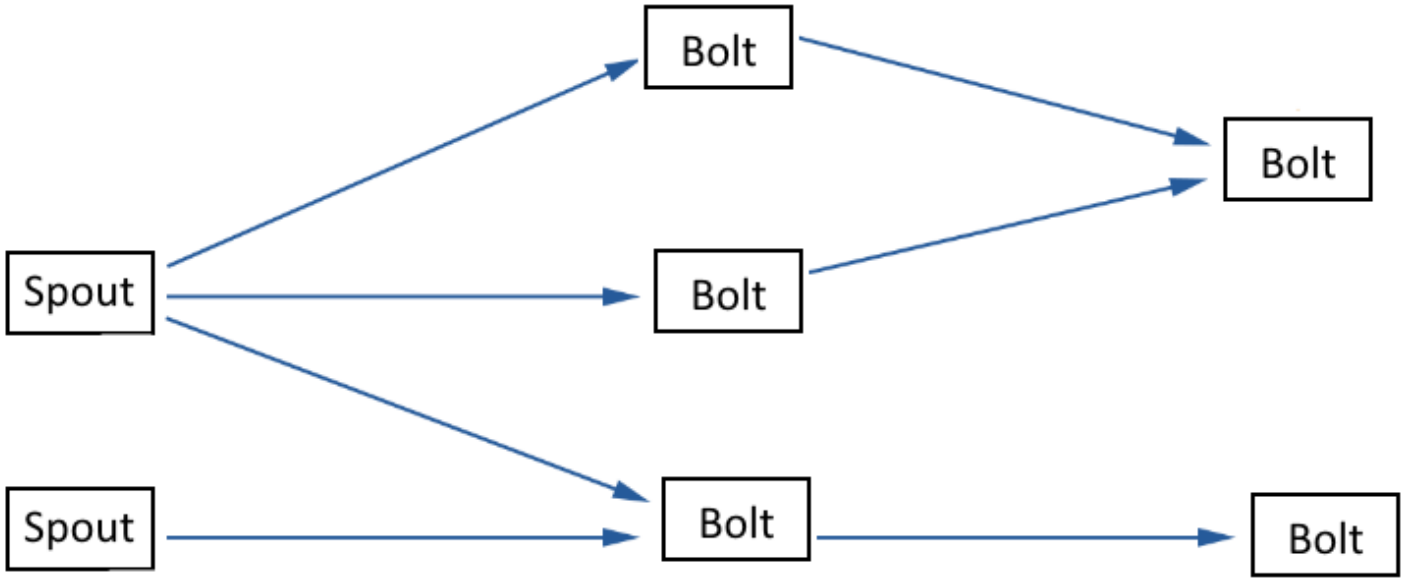
An application is defined in Storm through a **Topology** that describes its logic as a **DAG** of operators and streams



A Storm topology is analogous to a MapReduce job. One key difference is that a MapReduce job eventually finishes, whereas a **topology runs forever** (or until you kill it, of course)

Apache Storm

An application is defined in Storm through a **Topology** that describes its logic as a **DAG** of operators and streams



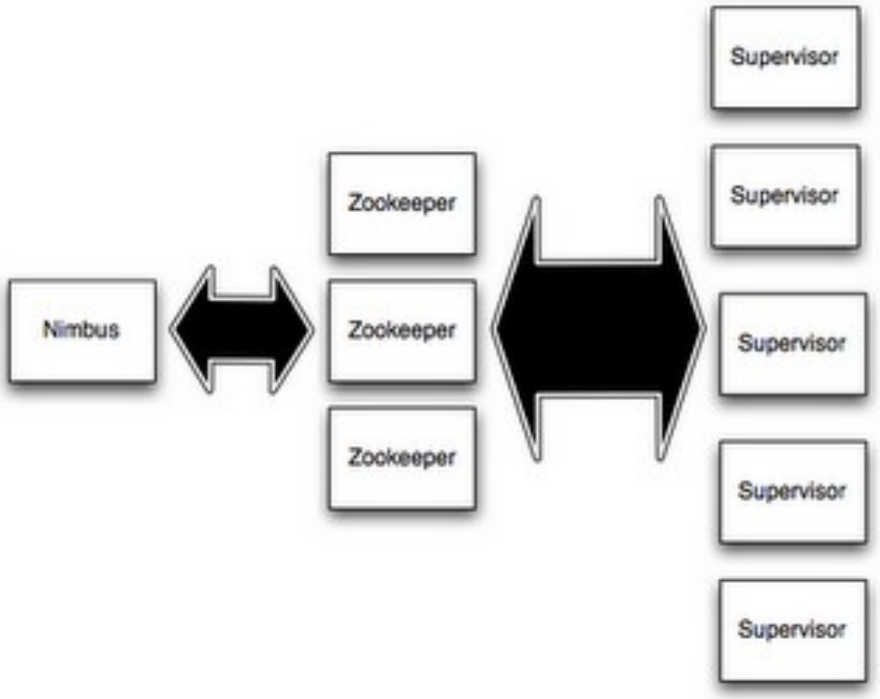
- Generally **spouts** will read tuples from an external source and emit them into the topology
- All processing in topologies is done in **bolts**. Bolts can do anything from filtering, functions, aggregations, joins, talking to databases, and more.

Apache Storm - Architecture

Two kinds of nodes on a Storm cluster:

- The **Master** node runs a daemon called "**Nimbus**" that is similar to Hadoop's "*JobTracker*". **Nimbus** is responsible for distributing code around the cluster, assigning tasks to machines, and monitoring for failures.

- Each **worker (slave)** node runs a daemon called the "**Supervisor**". The supervisor listens for work assigned to its machine and starts and stops worker processes as necessary based on what Nimbus has assigned to it. Each worker process executes a subset of a topology.

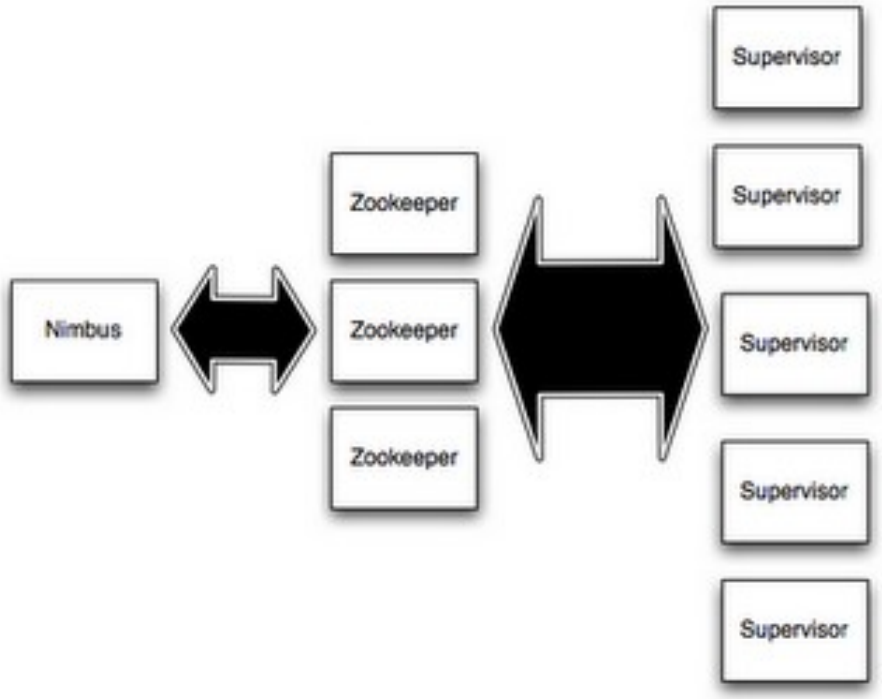


Apache Storm - Architecture

All coordination between Nimbus and the Supervisors is done through a **Zookeeper cluster**.

Additionally, the Nimbus daemon and Supervisor daemons are **fail-fast and stateless**; all state is kept in Zookeeper or on local disk.

This means you can kill -9 Nimbus or the Supervisors and **they'll start back up like nothing happened**. This design leads to Storm clusters being incredibly stable.



Coordenação em um SD

- *Coordination*: An act that multiple nodes must perform together

- Examples:

  Group membership

  Locking

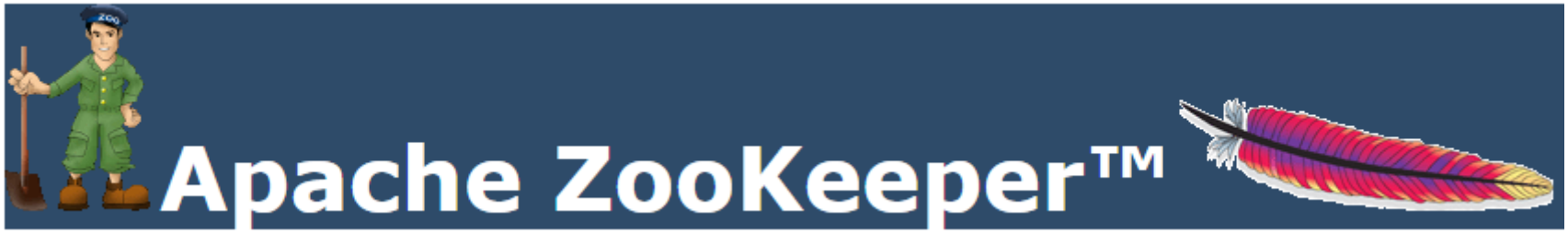
  Publisher / Subscriber

  Leader Election

  Synchronization

- Getting node coordination correct is very hard

Coordenação em um SD



ZooKeeper is a high-performance coordination service for distributed applications

Recipes

Apache Storm - Spout

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

org.apache.storm.topology

Interface IRichSpout

All Superinterfaces:

IComponent, ISpout, Serializable

All Known Implementing Classes:

BaseRichSpout, BlobStoreAPIWordCountTopology.RandomSentenceSpout, BucketTestHiveTopology.UserDataSpout, CheckpointSpout, ClojureSpout, ConstSpout, DRPCSpout, EsIndexTopology.UserDataSpout, EventHubSpout, FastWordCountTopology.FastRandomSentenceSpout, FeederSpout, FileReadSpout, FixedTupleSpout, FluxShellSpout, HdfsFileTopology.SentenceSpout, HdfsSpout, HiveTopology.UserDataSpout, HiveTopologyPartitioned.UserDataSpout, InOrderDeliveryTest.InOrderSpout, JmsSpout, KafkaSpout, KafkaSpout, KinesisSpout, MasterBatchCoordinator, MetricGenSpout, MqttSpout, PythonShellMetricsSpout, RandomIntegerSpout, RandomSentenceSpout, RandomSentenceSpout.TimeStamped, RawInputFromCSVSpout, RichShellSpout, RichSpoutBatchTriggerer, SequenceFileTopology.SentenceSpout, SolrFieldsSpout, SolrJsonSpout, SpoutTracker, StringGenSpout, TestEventLogSpout, TestPlannerSpout, TestWordSpout, ThroughputVsLatency.FastRandomSentenceSpout, TransactionalSpoutCoordinator, UserSpout, WordCountTopologyNode.RandomSentence, WordSpout, WordSpout, WordSpout

```
public interface IRichSpout
extends ISpout, IComponent
```

When writing topologies using Java, `IRichBolt` and `IRichSpout` are the main interfaces to use to implement components of the topology.

Method Summary

Methods inherited from interface org.apache.storm.spout.ISpout

ack, activate, close, deactivate, fail, nextTuple, open

Methods inherited from interface org.apache.storm.topology.IComponent

declareOutputFields, getComponentConfiguration

Apache Storm - Spout

```
@Override
public void open(Map conf, TopologyContext context, SpoutOutputCollector
output) {
}

@Override
public void nextTuple() {
}


@Override
public void declareOutputFields(OutputFieldsDeclarer d) {
}
```

Apache Storm - Bolt

```
public interface IRichBolt
extends IBolt, IComponent
```

When writing topologies using Java, `IRichBolt` and `IRichSpout` are the main interfaces to use to implement components of the topology.

Method Summary

Methods inherited from interface org.apache.storm.task.IBolt

`cleanup`, `execute`, `prepare`

Methods inherited from interface org.apache.storm.topology.IComponent

`declareOutputFields`, `getComponentConfiguration`

Apache Storm - Bolt

```
@Override
public void prepare(Map arg0, TopologyContext arg1, OutputCollector arg2) {
}

@Override
public void execute(Tuple arg0) {
}

@Override
public void declareOutputFields(OutputFieldsDeclarer arg0) {
}
```

Apache Storm - Topology

```
public static void main(String[] args) {
    Config config = new Config();

    TopologyBuilder builder = new TopologyBuilder();
    builder.setSpout("first_spout", new FirstSpout());
    builder.setBolt("splitter_bolt", new SplitterBolt())
            .shuffleGrouping("first_spout");
    builder.setBolt("counter_bolt", new CounterBolt())
            .shuffleGrouping("splitter_bolt");


    LocalCluster local = new LocalCluster();
    local.submitTopology("upe-poli", config, builder.createTopology());
}
```

Apache Storm – Hands-on

Processamento de Dados em Tempo Real

Jorge Cavalcanti Barbosa Fonsêca
(jorge.fonseca@upe.br)