Introductuion to High Performance Reactive Architectures, ML, Robotics, IoT

Disclaimer

All information presented in this document and all supplementary materials and programming code represent only my personal opinion and current understanding and has not received any endorsement or approval by IPT - Intellectual Products and Technologies or any third party. It should not be taken as any kind of advice, and should not be used for making any kind of decisions with potential commercial impact.

The information and code presented may be incorrect or incomplete. It is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement. In no event shall the author or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the information, materials or code presented or the use or other dealings with this information or programming code.

About Me

Trayan Iliev

– CEO of IPT – Intellectual Products & Technologies

– Oracle® certified programmer 15+ Y

– End-to-end reactive fullstack apps with Go, Python, Java, ES6/7, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer

– Voxxed Days, jPrime, Java2Days, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

– Lecturer @ Sofia University – courses: Fullstack React, Multimedia with Angular & TypeScript, Spring 5 Reactive, Internet of Things (with SAP), Multiagent Systems and Social Robotics, Practical Robotics & Smart Things

– Robotics / smart-things/ IoT enthusiast, RoboLearn hackathons organizer

Where to Find The Code and Materials?

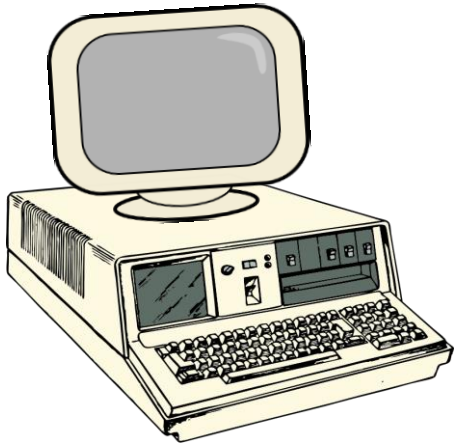
https://github.com/iproduct/course-ml

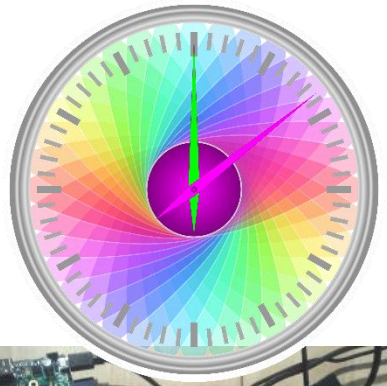
Machine Learning + Big Data in Real Time + Cloud Technologies

=> The Future of Intelligent Systems

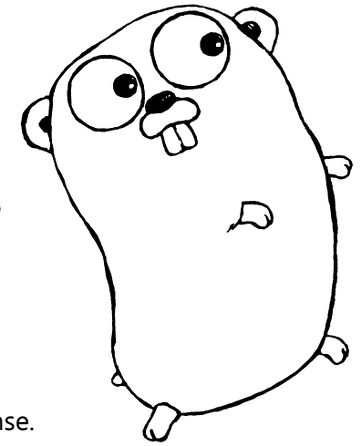
Asynchronous Event Streams in Real-time

Need for Speed :)

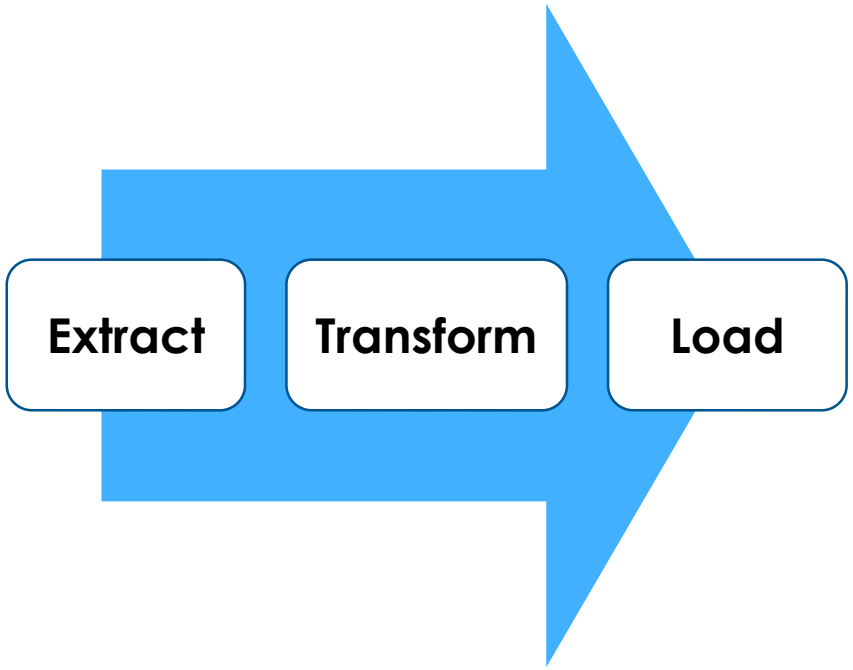


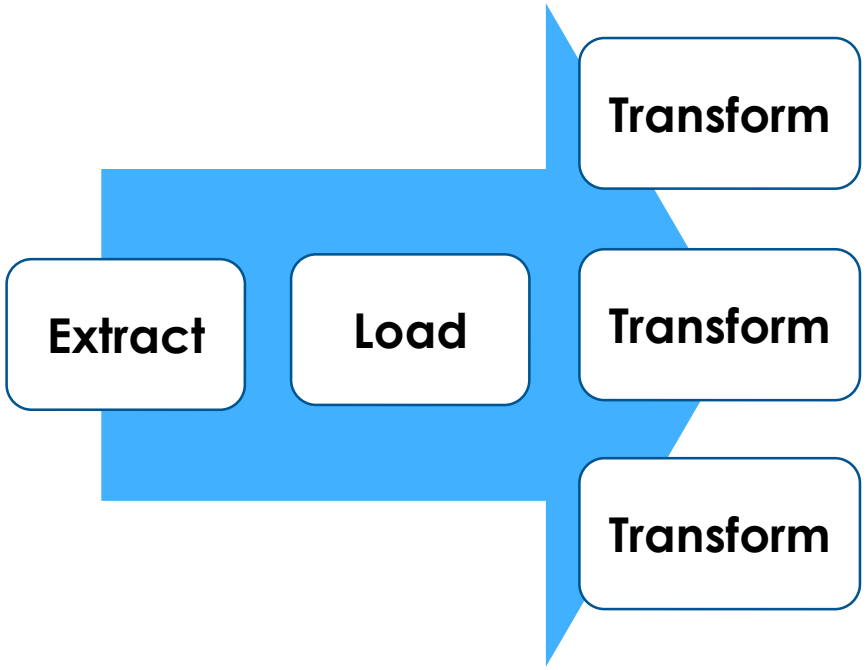


?

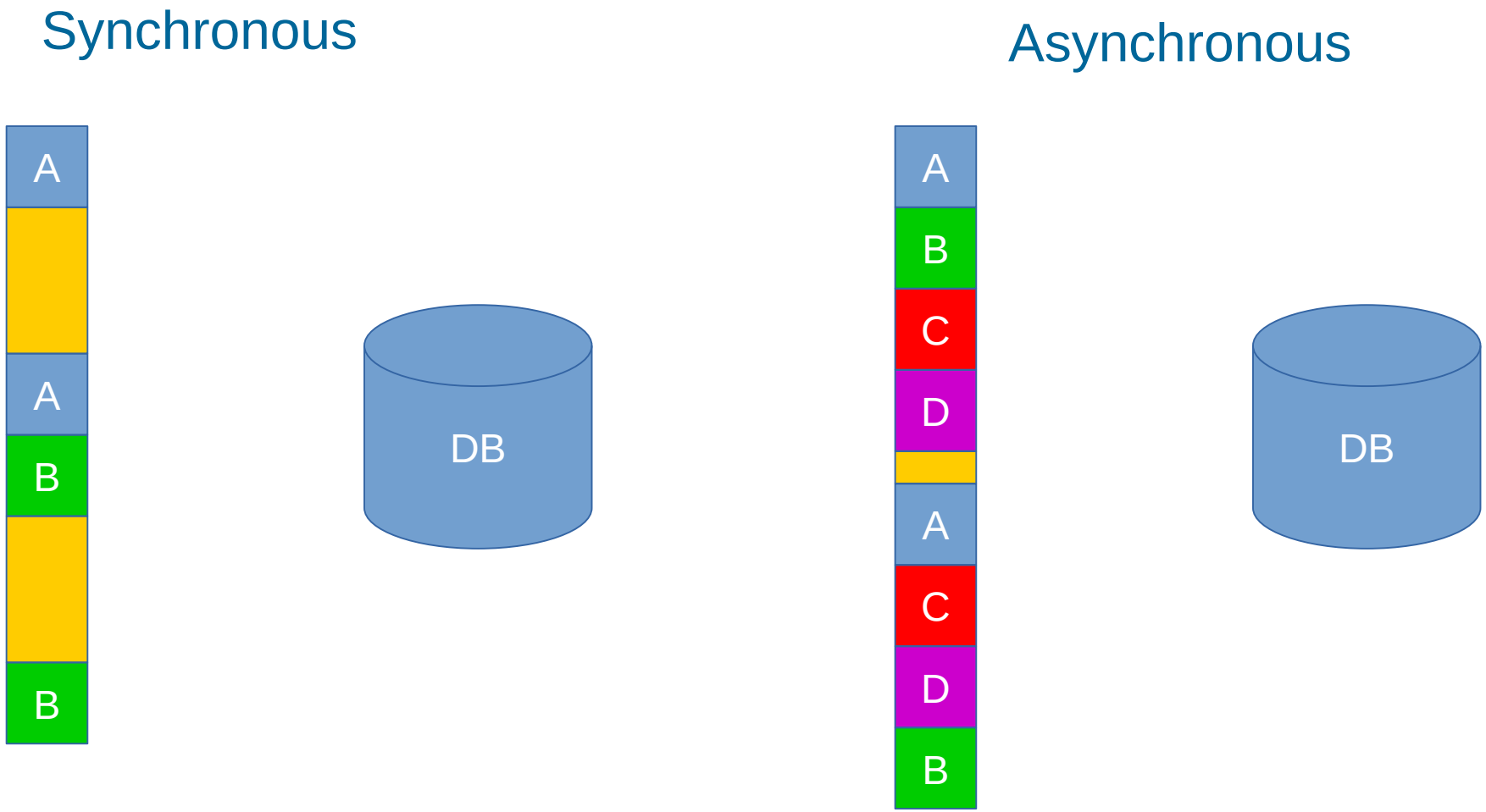


Batch Processing



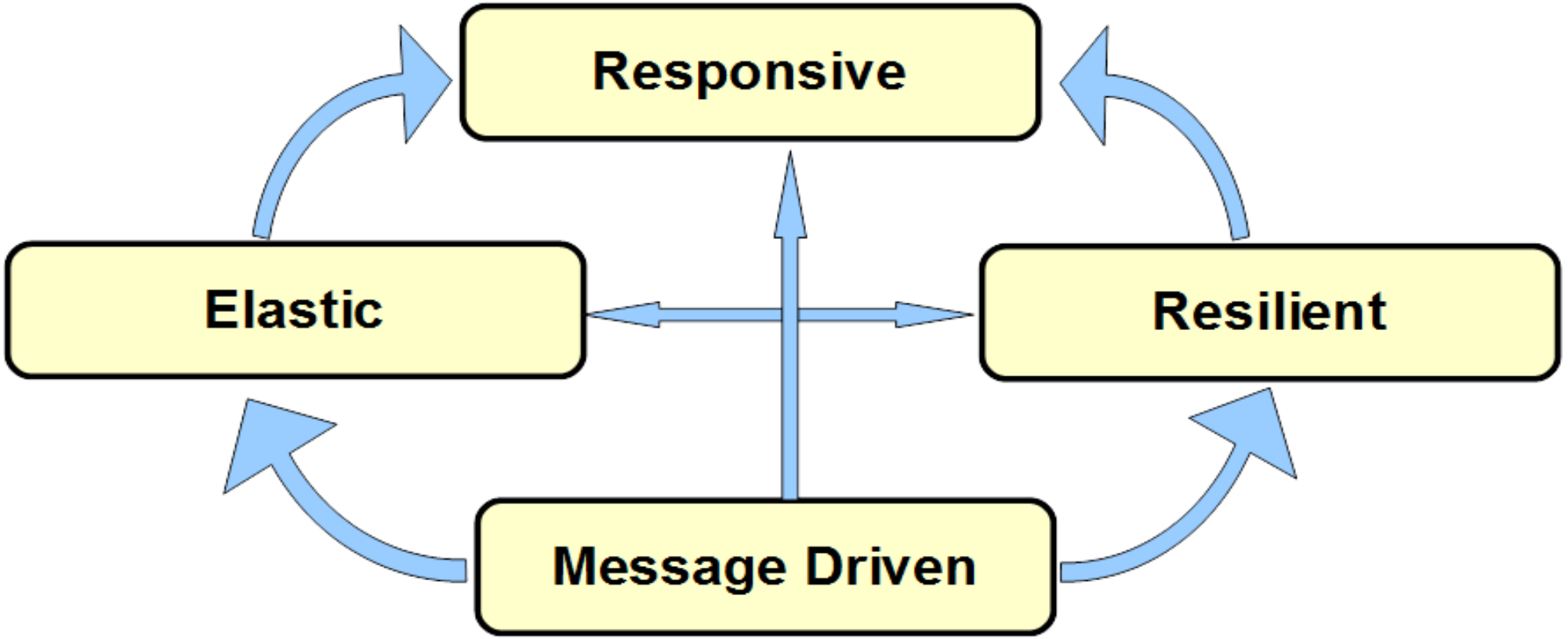


Synchronous vs. Asynchronous IO



Reactive Manifesto

http://www.reactivemanifesto.org



Scalable, Massively Concurrent

- **Message Driven** – asynchronous message-passing allows to establish a boundary between components that ensures loose coupling, isolation, location transparency, and provides the means to delegate errors as messages [Reactive Manifesto].
- The main idea is to separate concurrent producer and consumer workers by using **message queues.**
- **Message queues** can be **unbounded** or **bounded** (limited max number of messages)
- **Unbounded** message queues can present memory allocation problem in case the producers outrun the consumers for a long period → **OutOfMemoryError**

What's High Performance?

- **Performance** is about 2 things (Martin Thompson – http://www.infoq.com/articles/low-latency-vp):
  - **Throughput** – units per second, and
  - **Latency** – response time
- **Real-time** – time constraint from input to response regardless of system load.
- **Hard real-time system** if this constraint is not honored then a total system failure can occur.
- **Soft real-time system** – low latency response with little deviation in response time
- **100 nano-seconds** to **100 milli-seconds**. [Peter Lawrey]

Data / Event / Message Streams

“Conceptually, a stream is a (potentially never-ending) flow of data records, and a transformation is an operation that takes one or more streams as input, and produces one or more output streams as a result.”

Apache Flink: Dataflow Programming Model

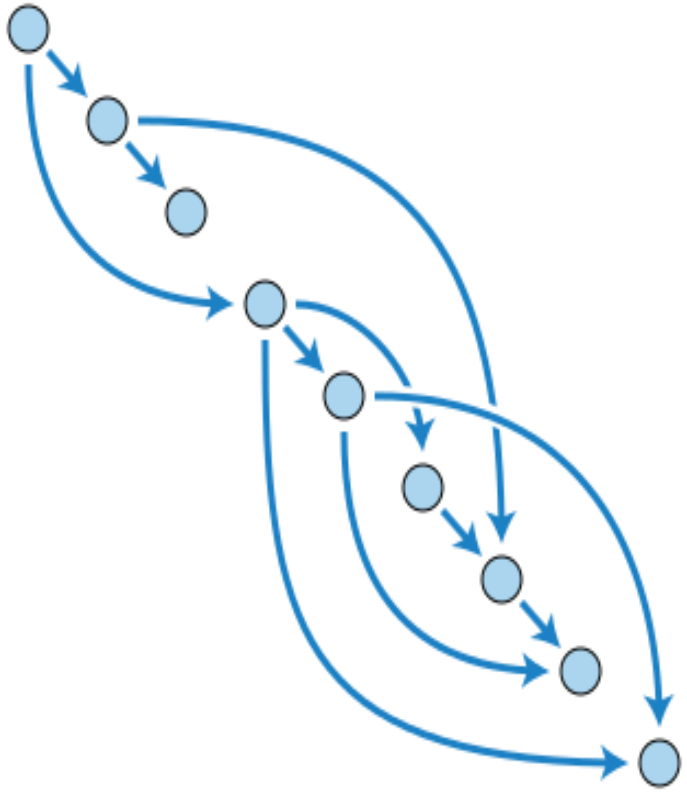
Data Stream Programming

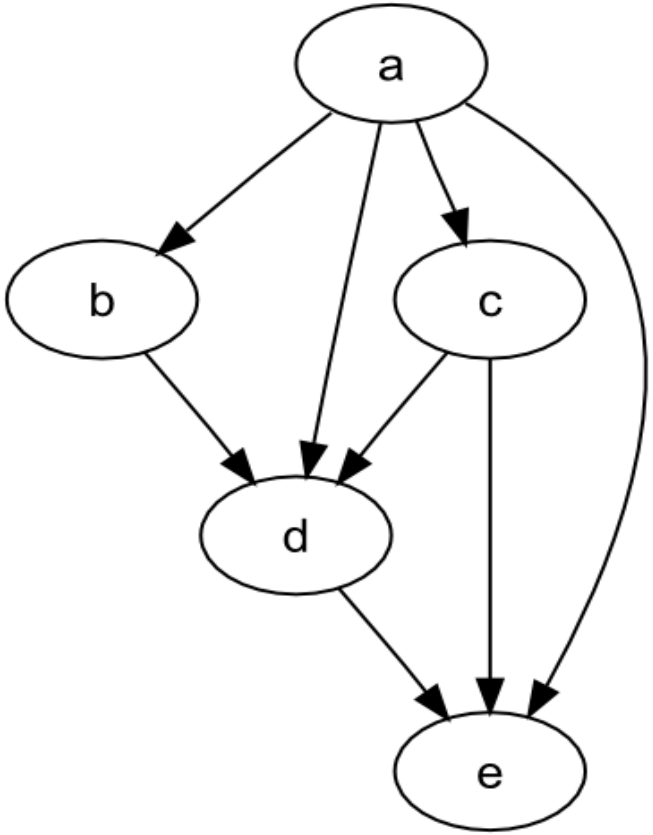
The idea of abstracting logic from execution is hardly new -- it was the dream of SOA. And the recent emergence of microservices and containers shows that the dream still lives on.

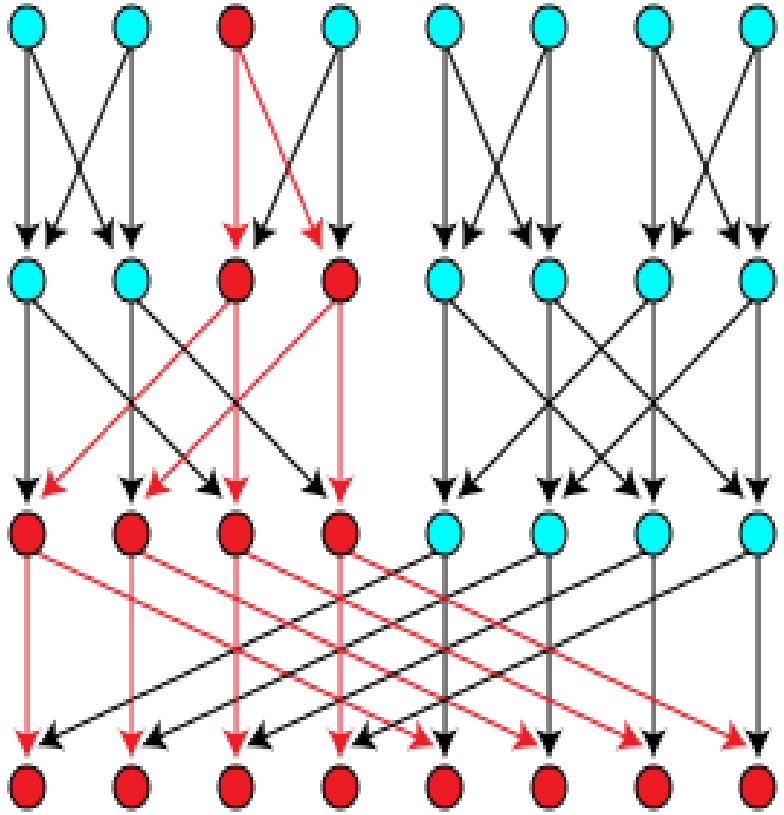
For developers, the question is whether they want to learn yet one more layer of abstraction to their coding. On one hand, there's the elusive promise of a common API to streaming engines that in theory should let you mix and match, or swap in and swap out.

Tony Baer (Ovum) @ ZDNet - Apache Beam and Spark: New coopetition for squashing the Lambda Architecture?

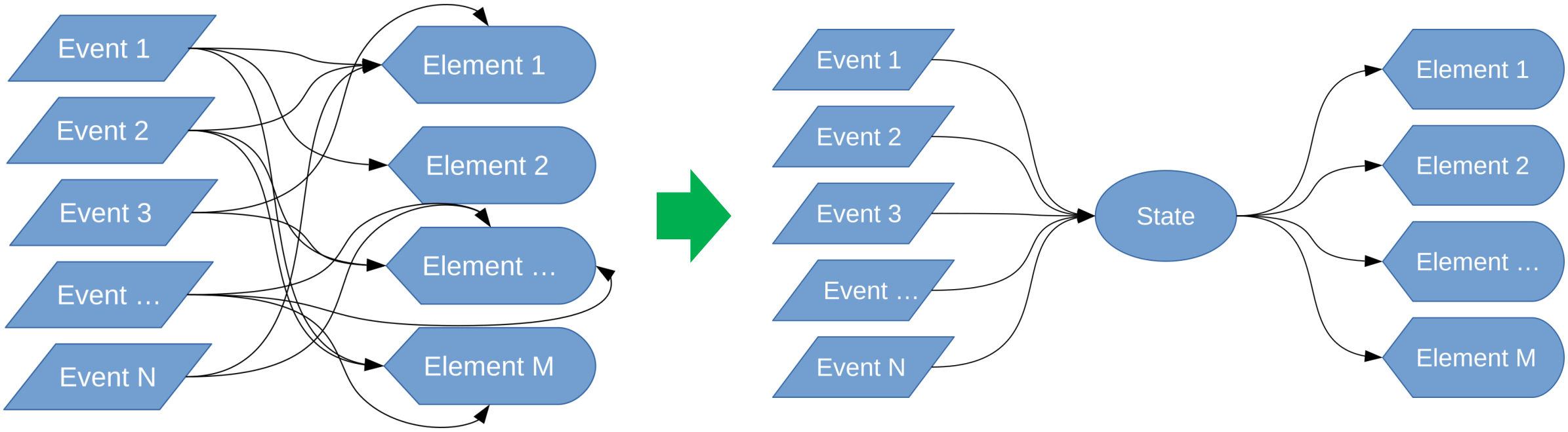
Direct Acyclic Graphs - DAG





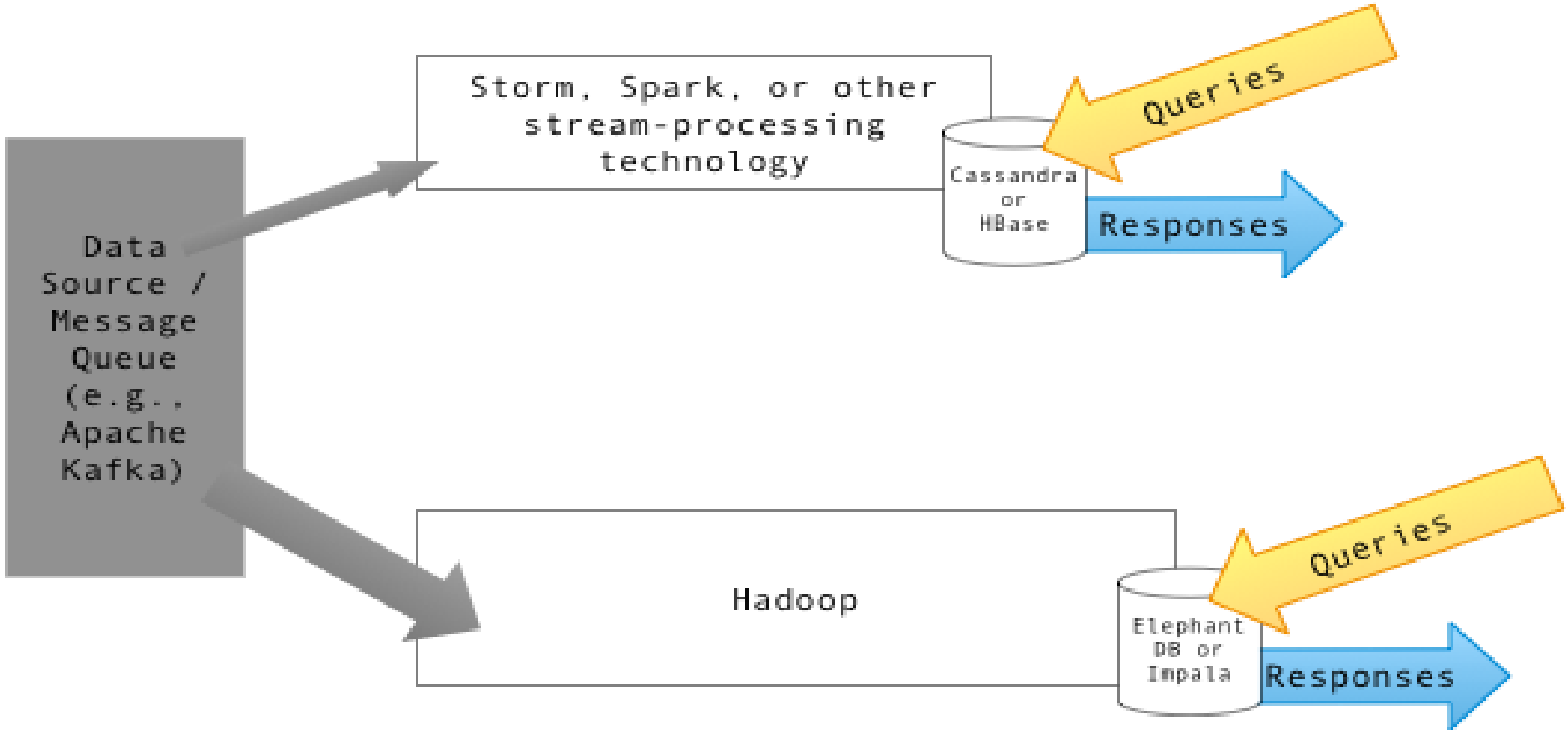


Event Sourcing – Events vs. Sate (Snapshots)



Lambda Architecture - I

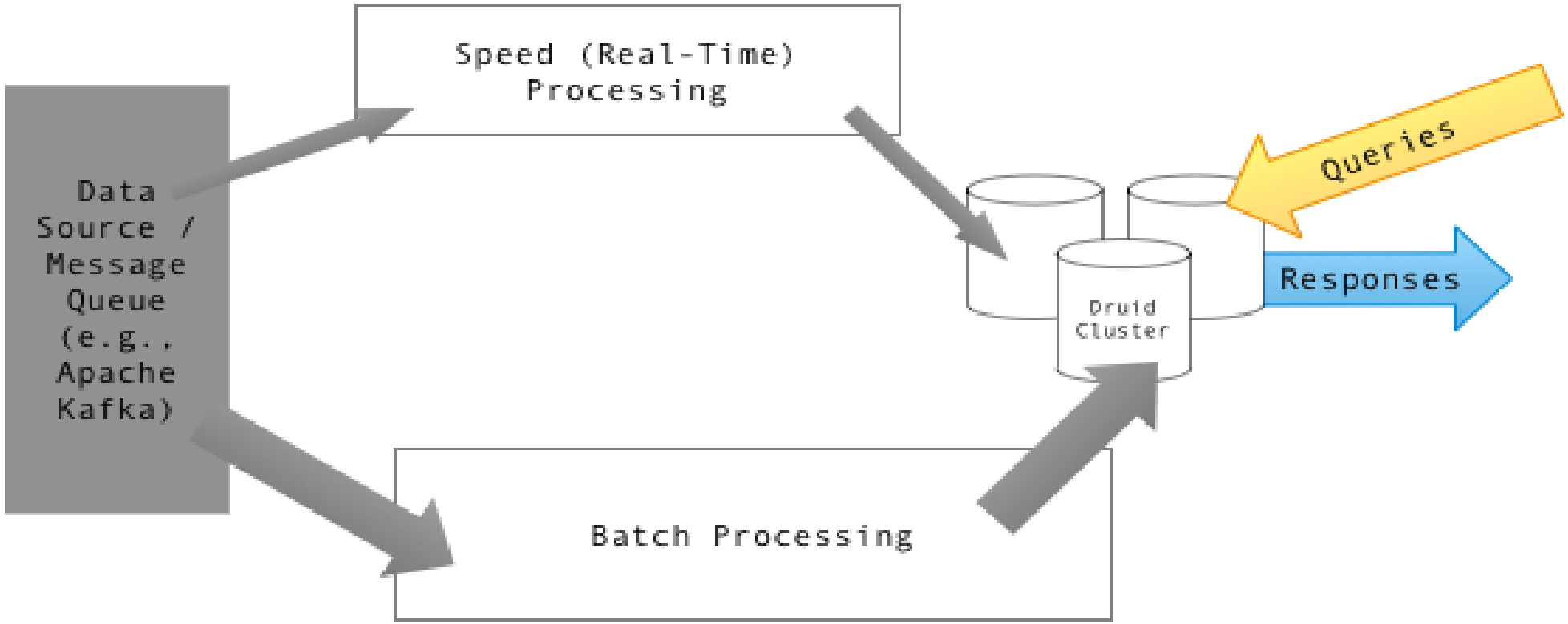
Query = λ (Complete data) = λ (live streaming data) * λ (Stored data)



https://commons.wikimedia.org/w/index.php?curid=34963987, By Textractor - Own work, CC BY-SA 4

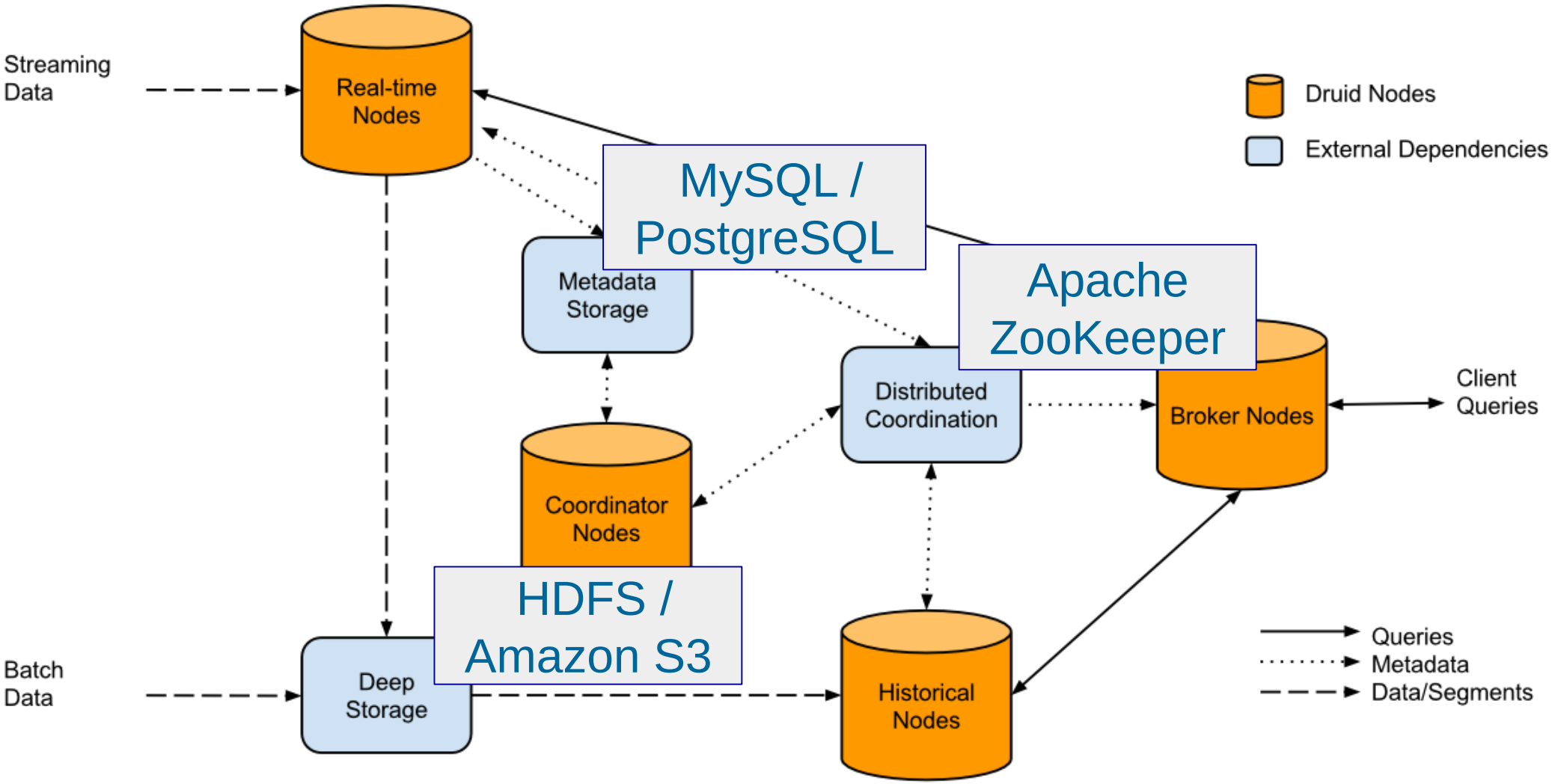
Lambda Architecture - II

Query = λ (Complete data) = λ (live streaming data) * λ (Stored data)



https://commons.wikimedia.org/w/index.php?curid=34963986, By Textractor - Own work, CC BY-SA 4

Lambda Architecture - Druid Distributed Data Store



https://commons.wikimedia.org/w/index.php?curid=33899448 By Fangjin Yang, GFDL

Kappa Architecture

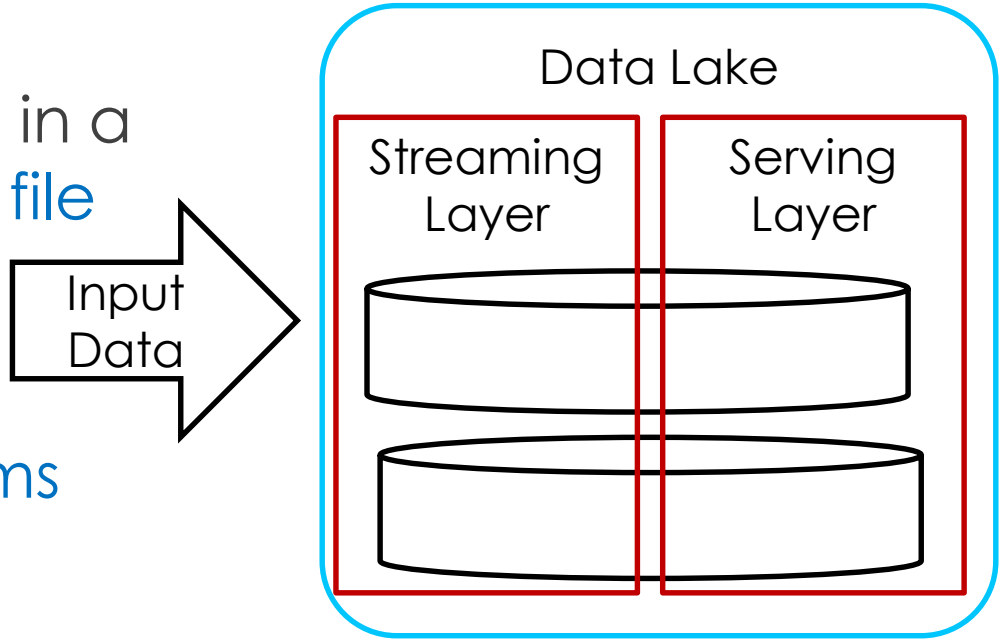
Query = K (New Data) = K (Live streaming data)

- Proposed by Jay Kreps in 2014
- Real-time processing of distinct events
- Drawbacks of Lambda architecture:
  - It can result in coding overhead due to comprehensive processing
  - Re-processes every batch cycle which may not be always beneficial
  - Lambda architecture modeled data can be difficult to migrate
- Canonical data store in a Kappa Architecture system is an append-only immutable log (like Kafka, Pulsar)

Kappa Architecture II

Query = K (New Data) = K (Live streaming data)

- Multiple data events or queries are logged in a queue to be catered against a distributed file system storage or history.
- The order of the events and queries is not predetermined. Stream processing platforms can interact with database at any time.
- It is resilient and highly available as handling terabytes of storage is required for each node of the system to support replication.
- Machine learning is done on the real time basis



Zeta Architecture

- Main characteristics of Zeta architecture:
  - file system (HDFS, S3, GoogleFS),
  - realtime data storage (HBase, Spanner, BigTable),
  - modular processing model and platform (MapReduce, Spark, Drill, BigQuery),
  - containerization and deployment (cgroups, Docker, Kubernetes),
  - Software solution architecture (serverless computing – e.g. Amazon Lambda)
- Recommender systems and machine learning
- Business applications and dynamic global resource management (Mesos + Myriad, YARN, Diego, Borg).

Distributed Stream Processing – Apache Projects:

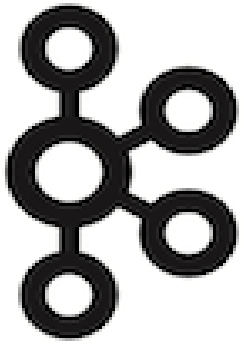
- Apache Spark is an open-source cluster-computing framework. Spark Streaming, Spark Mllib

- Apache Storm is a distributed stream processing – streams DAG

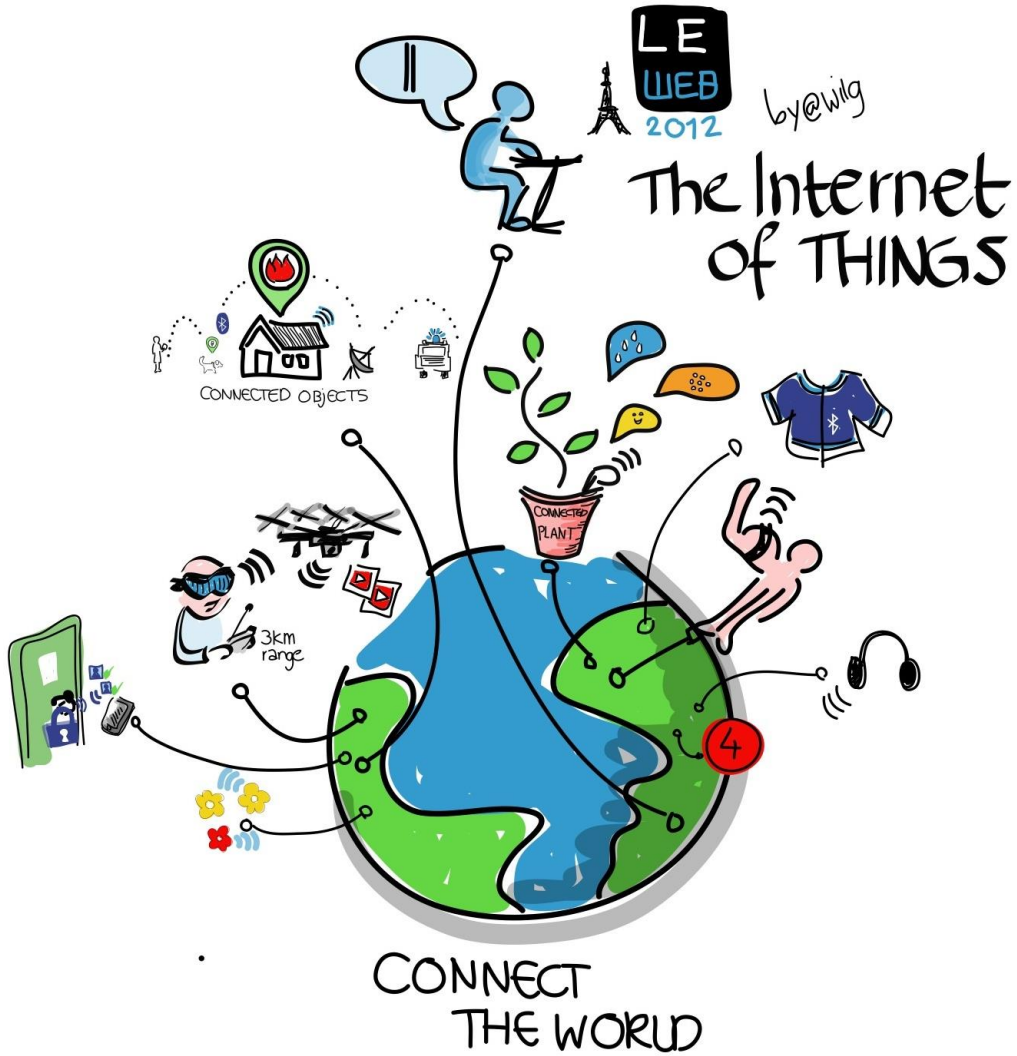
- Apache Samza is a distributed real-time stream processing framework.

Distributed Stream Processing – Apache Projects II

- Apache Flink - open source stream processing framework – Java, Scala
- Apache Kafka - open-source stream processing (Kafka Streams), real-time, low-latency, high-throughput, massively scalable pub/sub
- Apache Beam – unified batch and streaming, portable, extensible

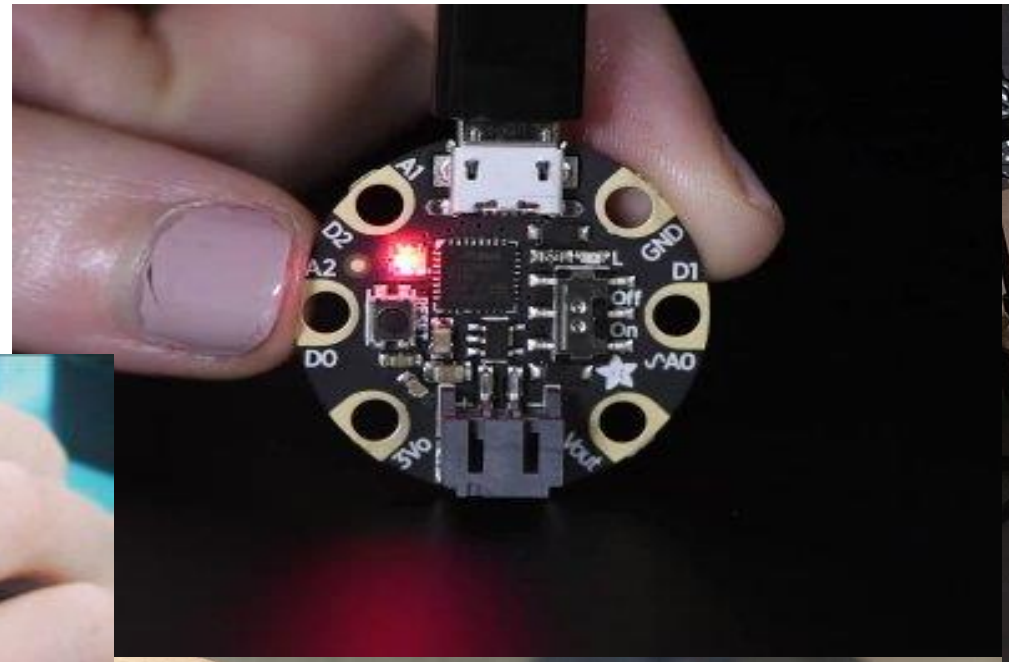


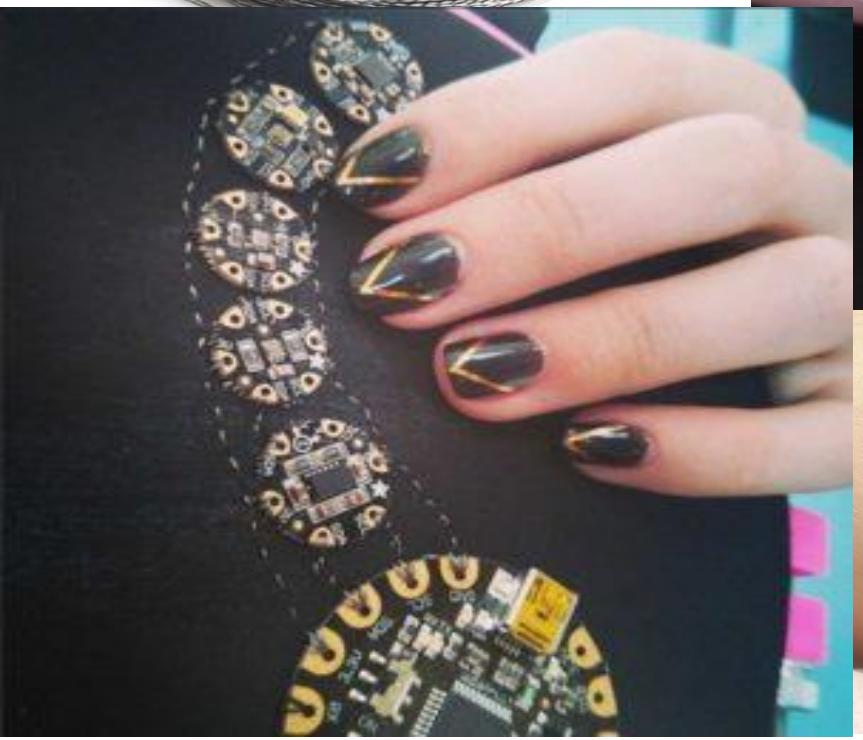
Example: Internet of Things (IoT)



CC BY 2.0, Source: https://www.flickr.com/photos/wilgengebroed/8249565455/ ,
Radar, GPS, lidar for navigation and obstacle avoidance (2007 DARPA Urban Challenge)

Wearable Electronics :)





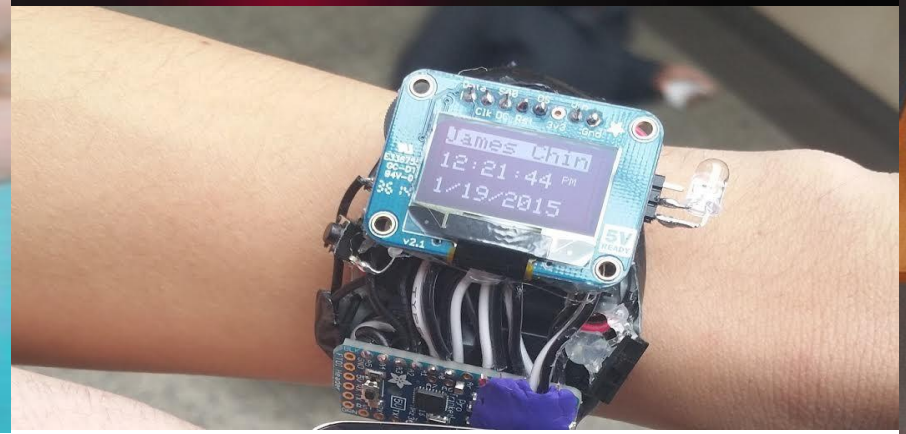
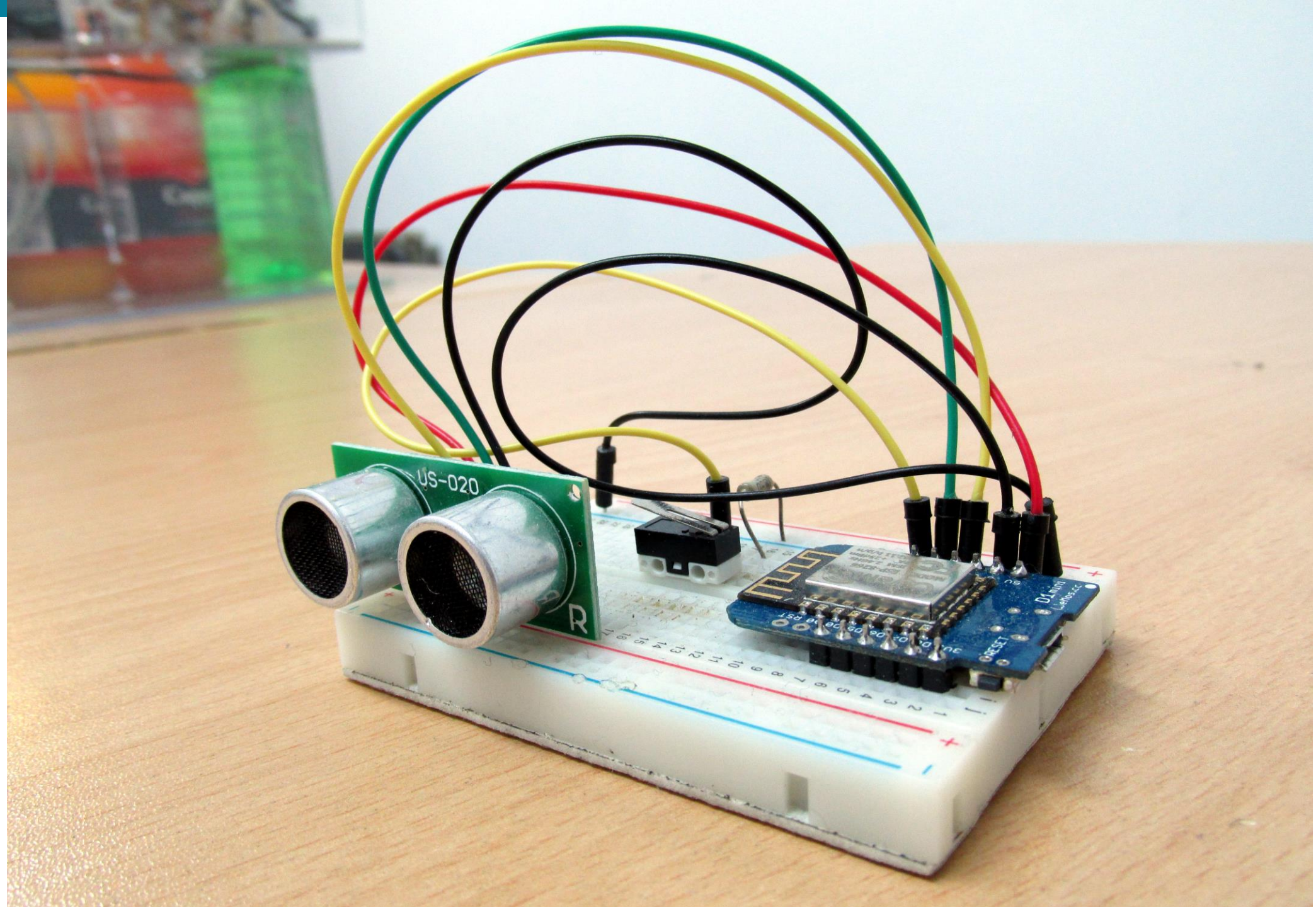
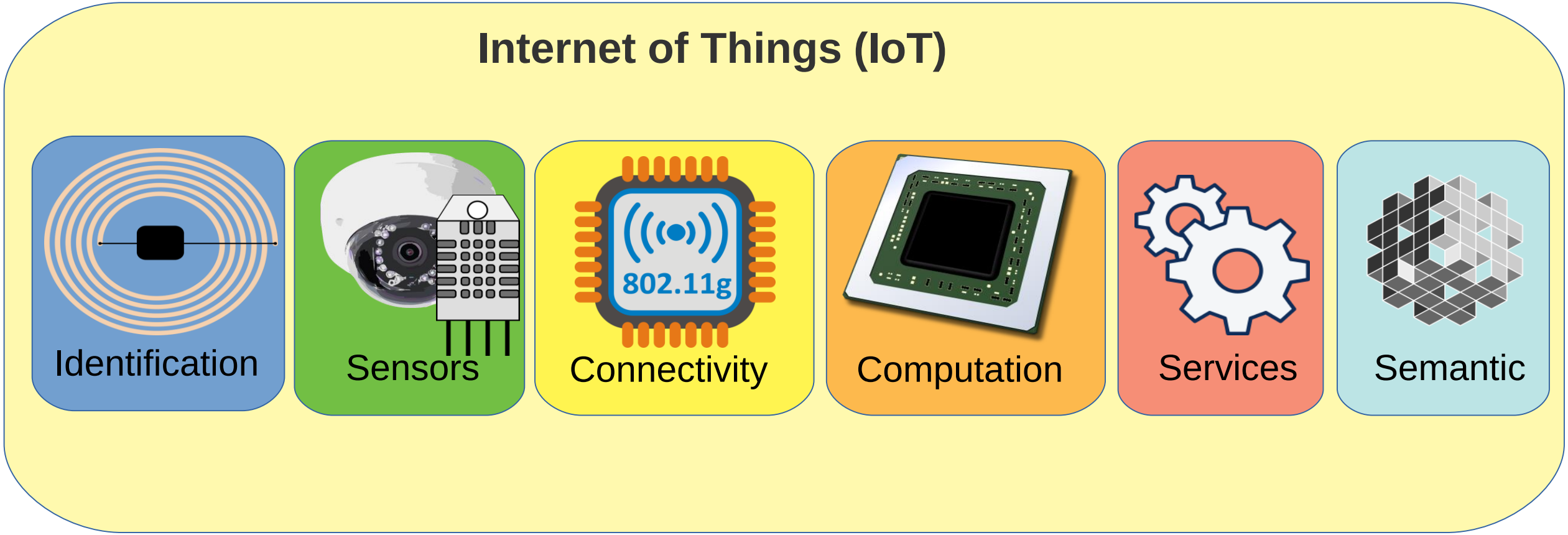


Photo credit: Adafruit, Becky Stern and others

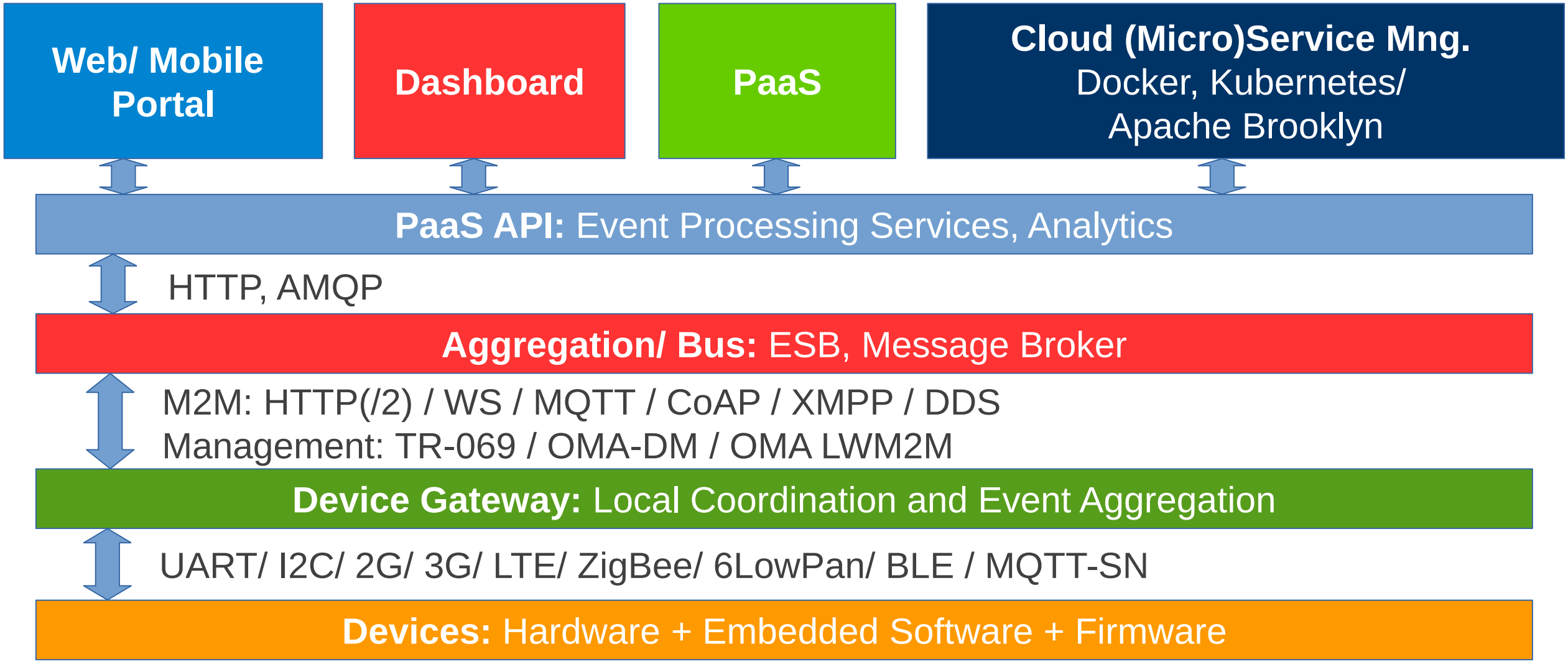

US-020
intel
JOULE
570x


US-020
R
D1 mini
RESET

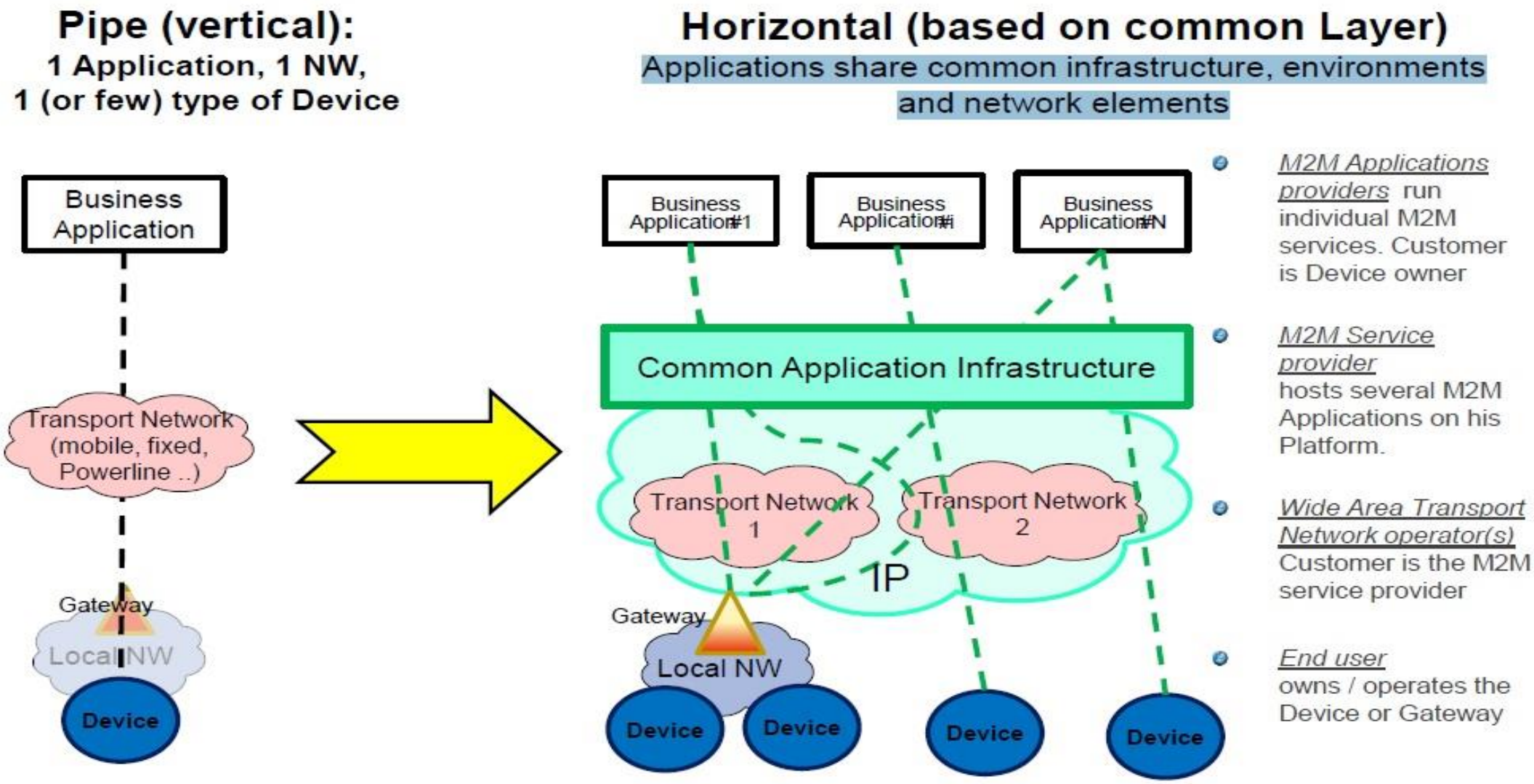
Key Elements of IoT



IoT Services Architecture



Vertical vs. Horizontal IoT



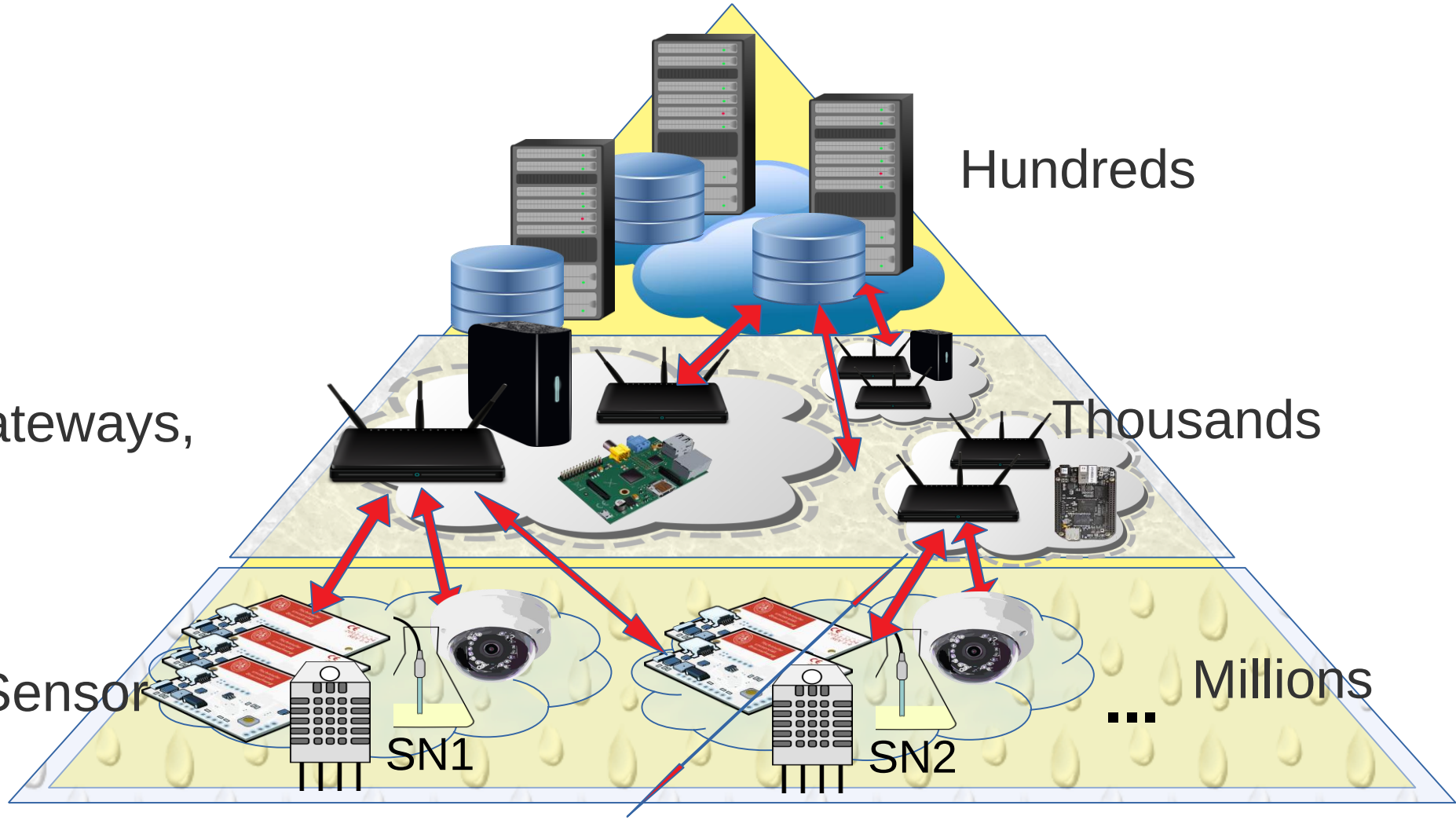
http://www.iot-week.eu/overview_2013/iot-week-2012/programme-1/tuesday-1/presentations/semantic-interoperability/K1_ETSI-M2M-oneM2M-and-the-need-for-semantics-IoTWeek2012.pdf, By Joerg Swetina (NEC)- ETSI M2M / oneM2M and the need for semantics

Cloud, Fog and Mist Computing

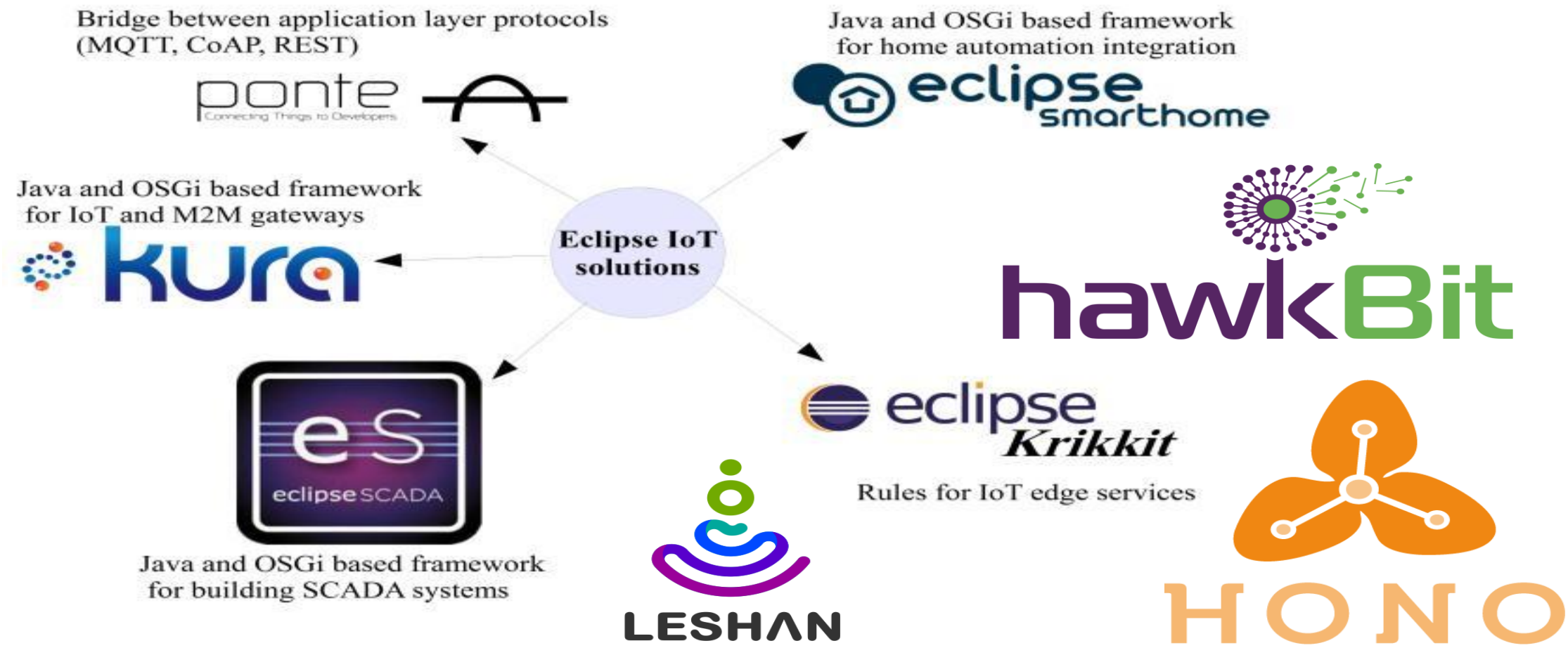
Cloud Computing
(Data-centers)

Fog Computing
(Fog Nodes, Edge Gateways, Cloudlets)

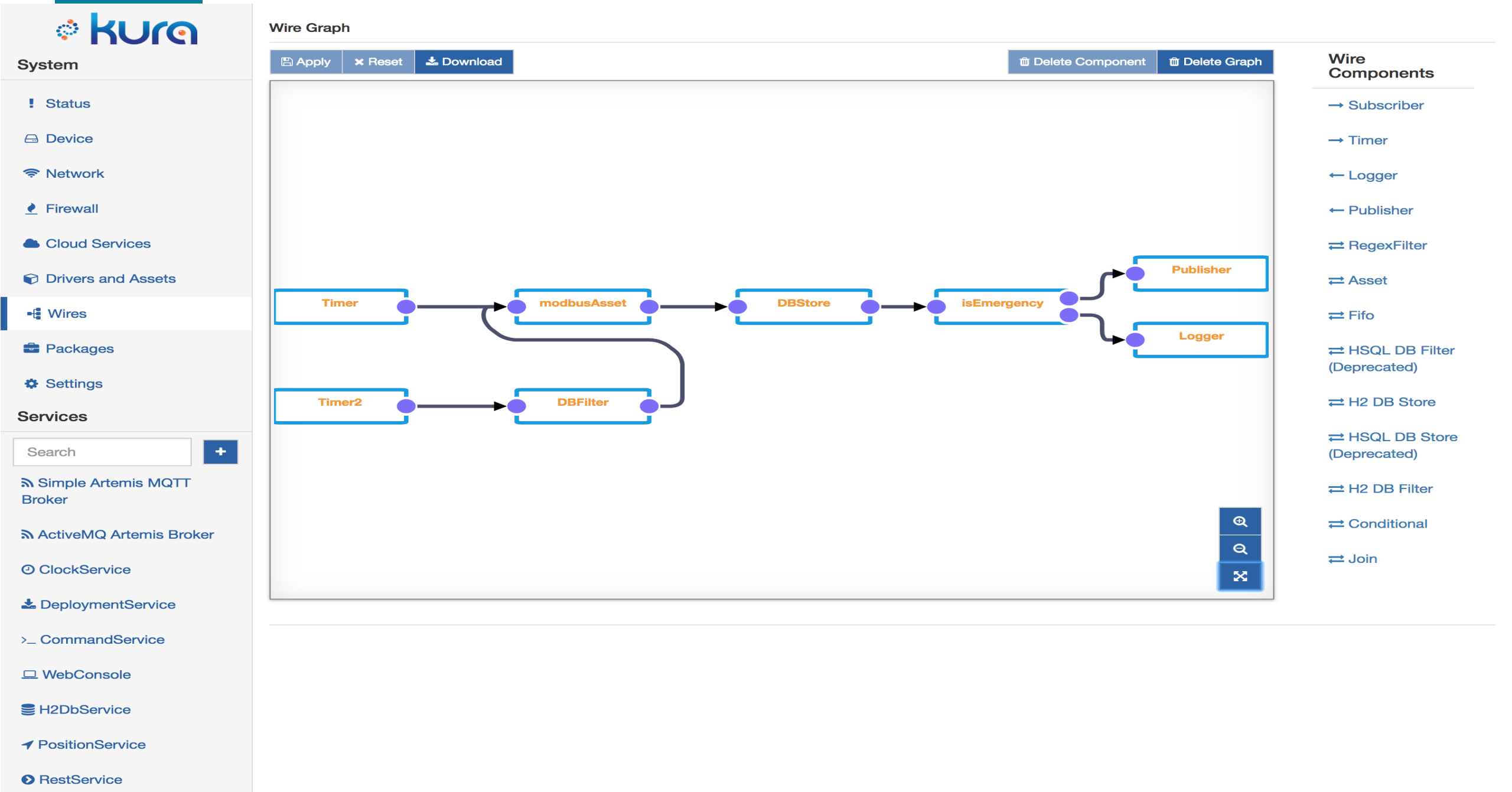
Mist Computing
(Smart IoT Devices, Sensor Networks)



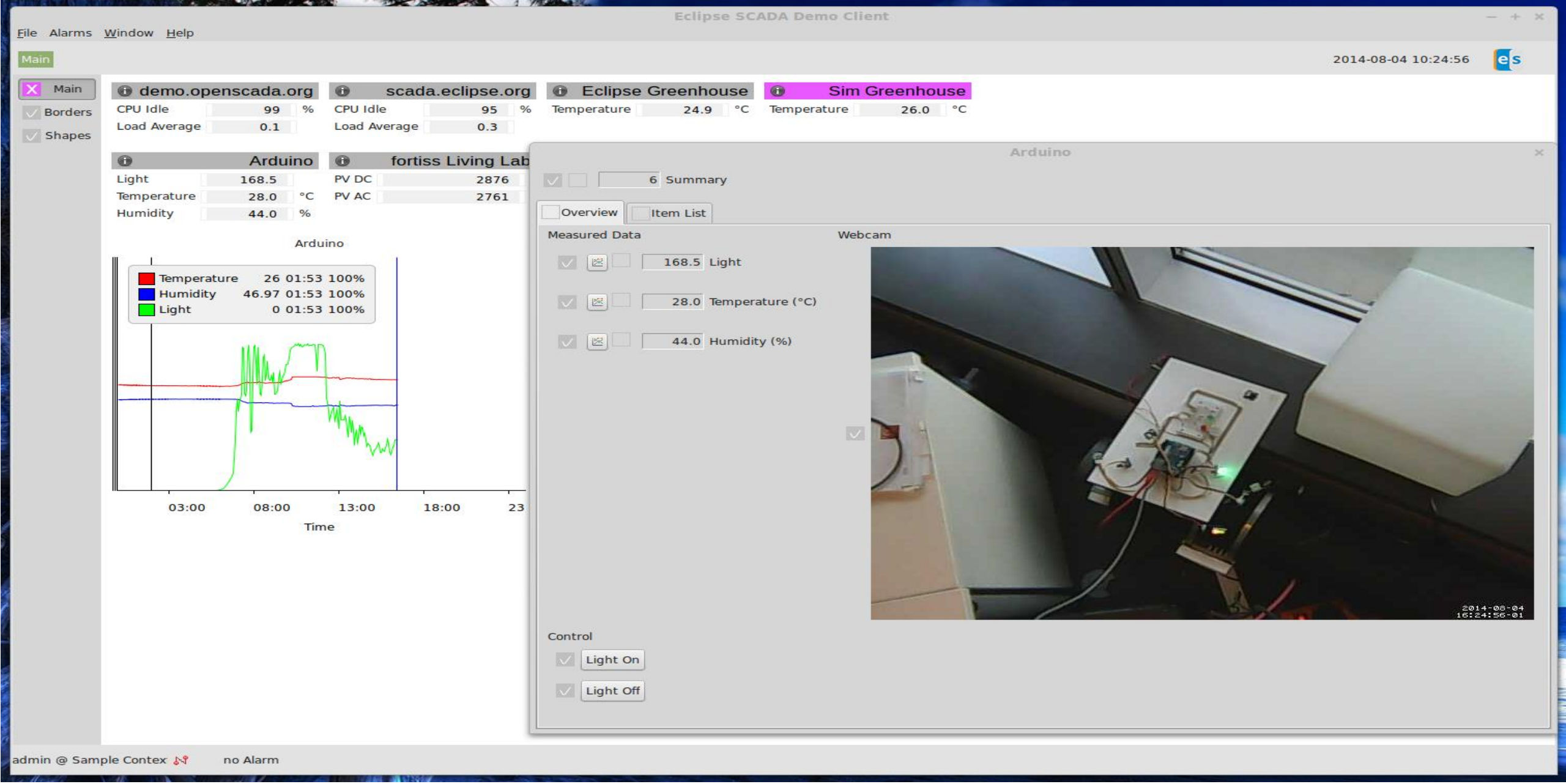
Eclipse IoT Platform



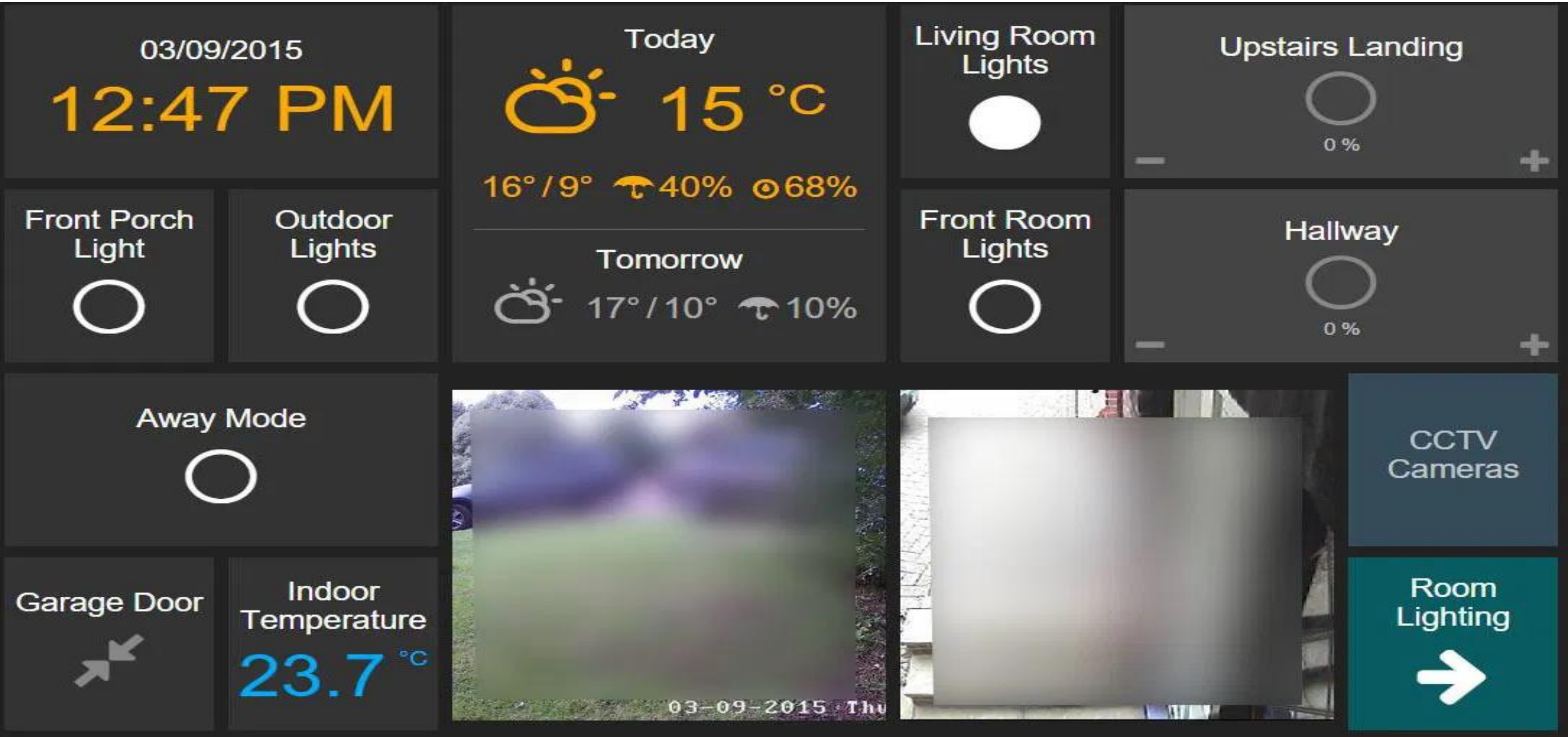
Based on: https://www.researchgate.net/publication/279177017_Internet_of_Things_A_Survey_on_Enabling_Technologies_Protocols_and_Applications, By Ala Al-Fuqaha et al. - Internet of Things: A Survey on Enabling Technologies, Protocols and Applications



Source: https://www.eclipse.org/kura/



Eclipse Smart Home – Open HAB



01/09/2017

1:40 AM

57.4 °F

Humidity: 52 %

Pressure: 30.12 in

Today	Tuesday	Wednesday
		
Mostly Cloudy	Clear	Clear
75 °F / 46 °F	70 °F / 45 °F	66 °F / 45 °F

Utility

Hallway

Family Room

Dining

Kitchen

Stereo Volume

17%

Stereo

34%

73.0 °F

Office: 77°

70

Front Yard

Porch

Driveway

Garage

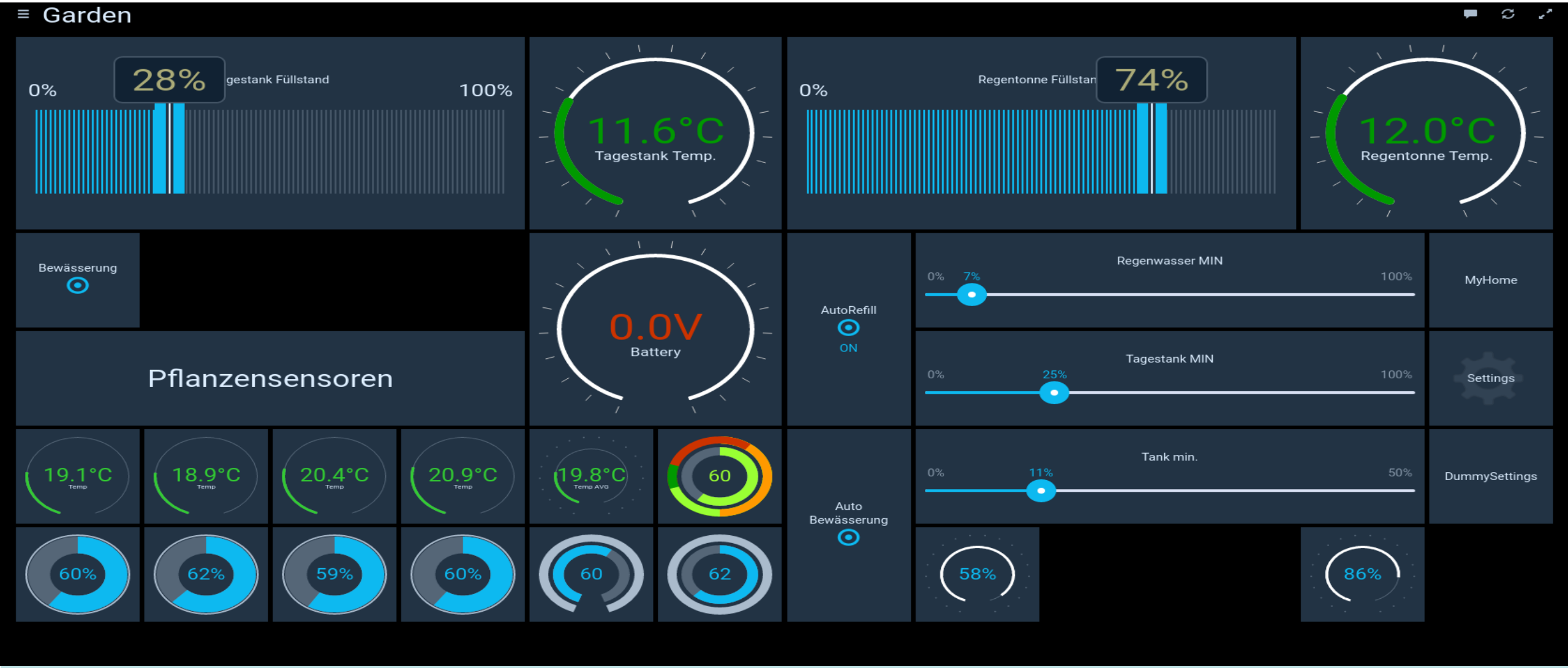
Living Room

Living Room Lights

Back Yard

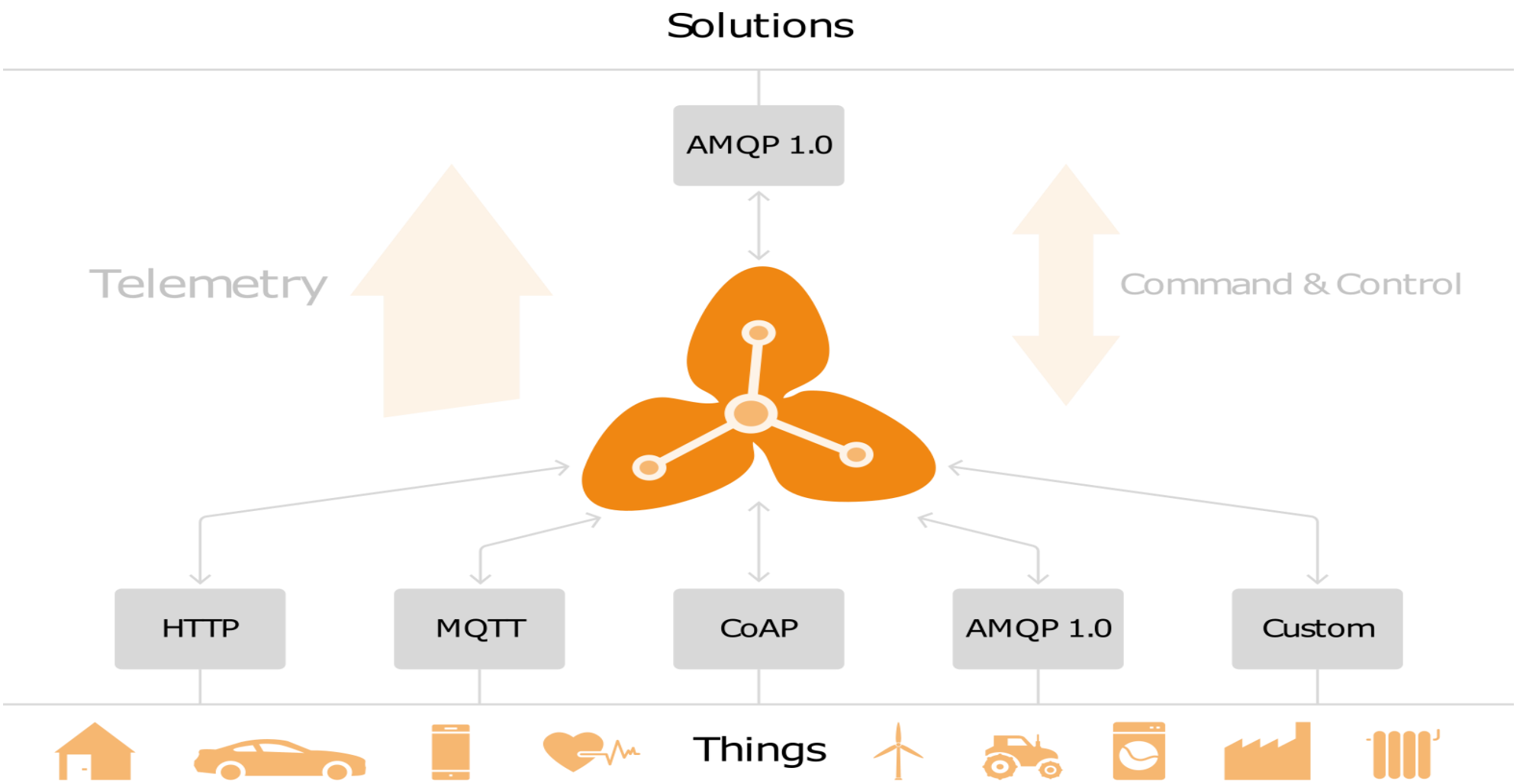
Wake Up

Open HAB: Hydroponics Dashboard



Source: https://community.openhab.org/t/aquaponics-goes-openhab/26456

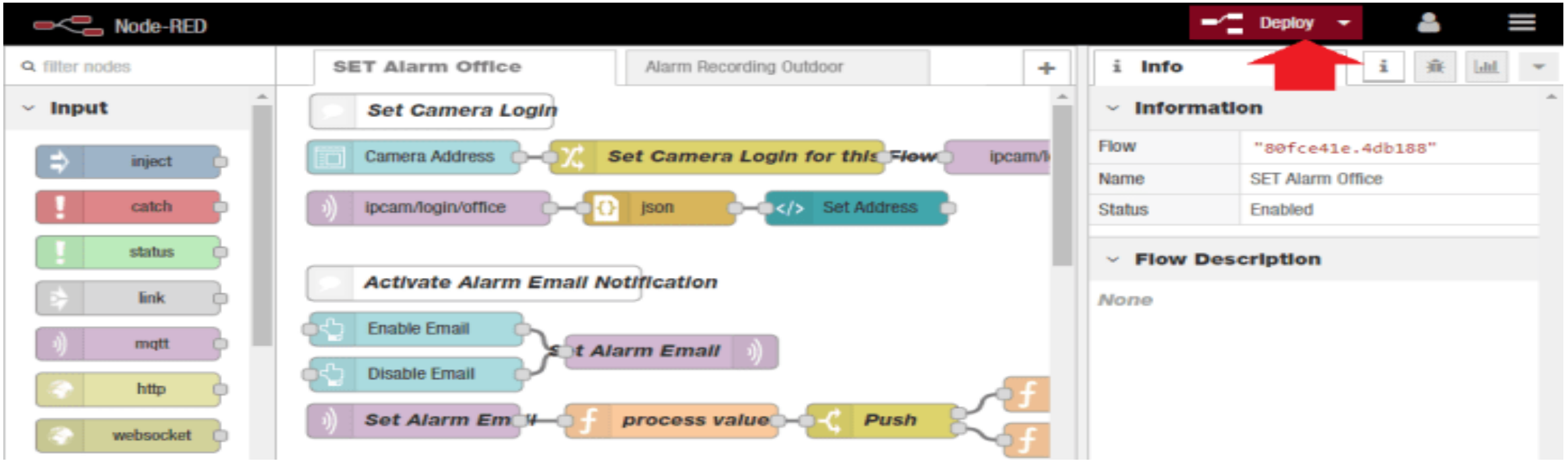
Eclipse Hono: Connect. Command. Control.



Source: https://www.eclipse.org/hono/

Node Red

Node-RED is a flow-based development tool for visual programming developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things.



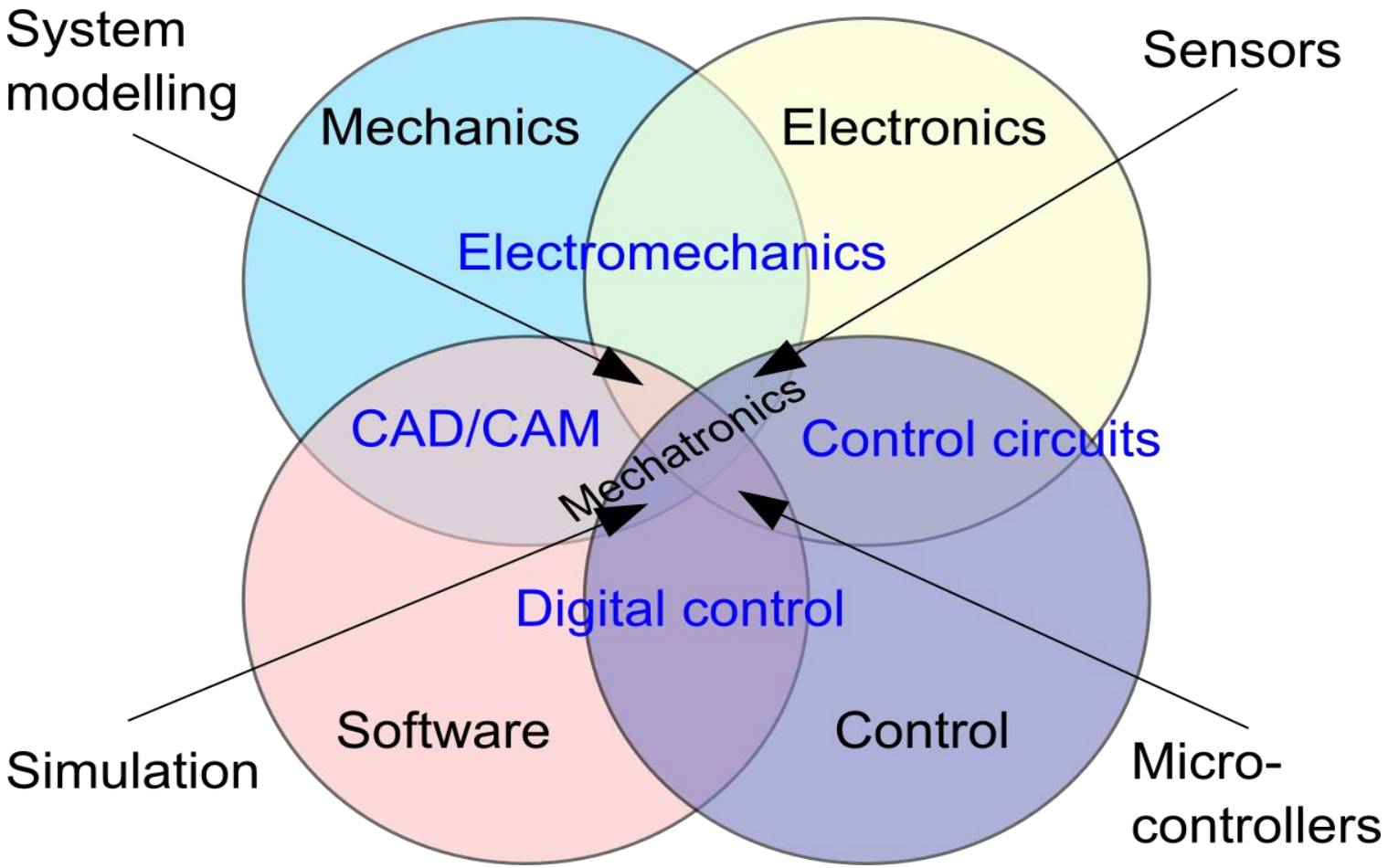
Source: https://nodered.org/

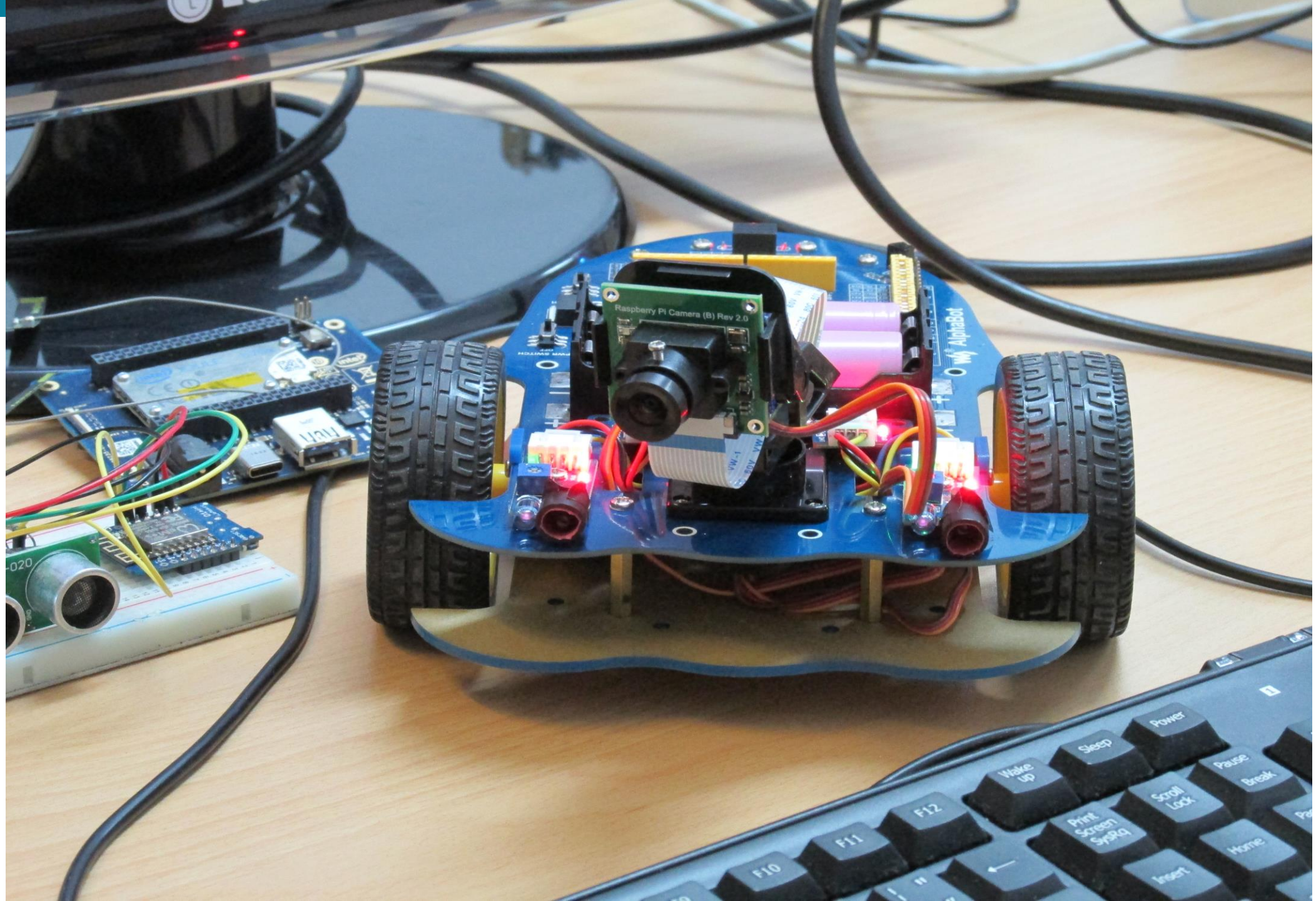
Robotics

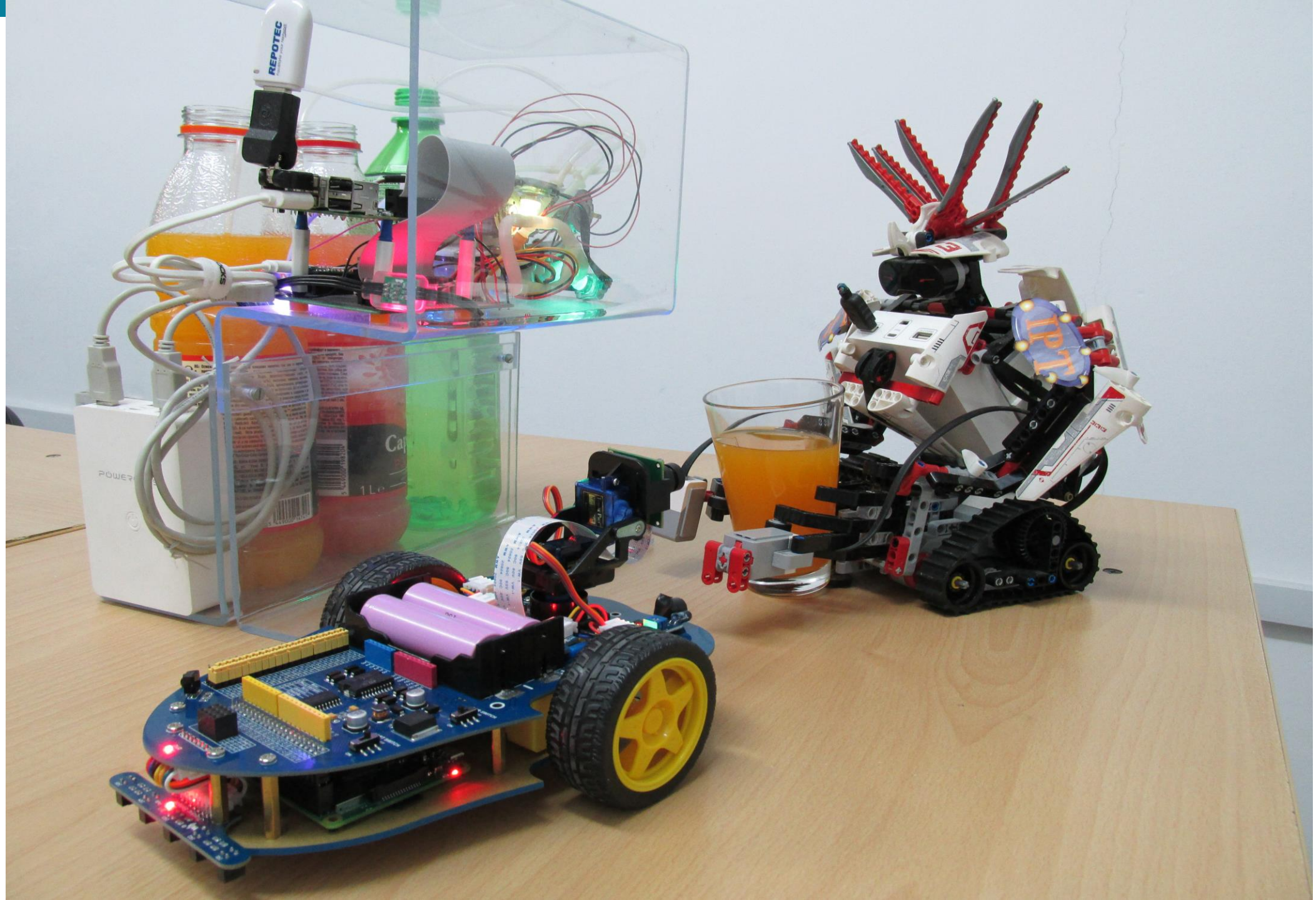
Роботиката е пресечна точка на много диспилини:

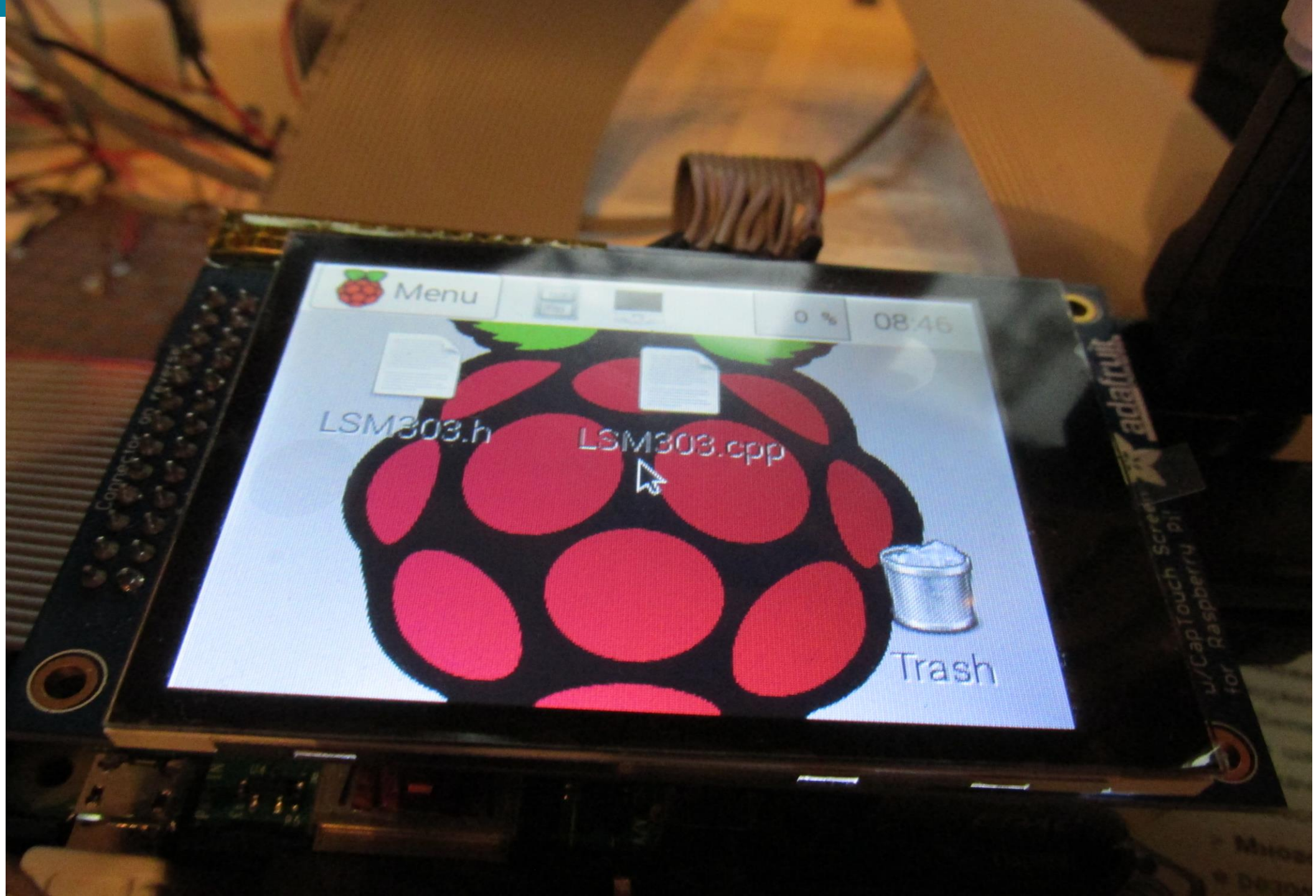
- Механика
- Електроника и компютърни науки
- Софтуерно инженерство
- Изкуствен интелект (AI)
- Човеко-машинни интерфейси
- Социология и психология
- Дизайн

Инженерство, наука и изкуство




Raspberry Pi Camera (B) Rev 2.0
AlphaBot
Power
Sleep
Wake up
F12
F11
F10
Scroll Lock
Pause Break
Print Screen SysRq
Home
Insert


REPOTEC
IPT

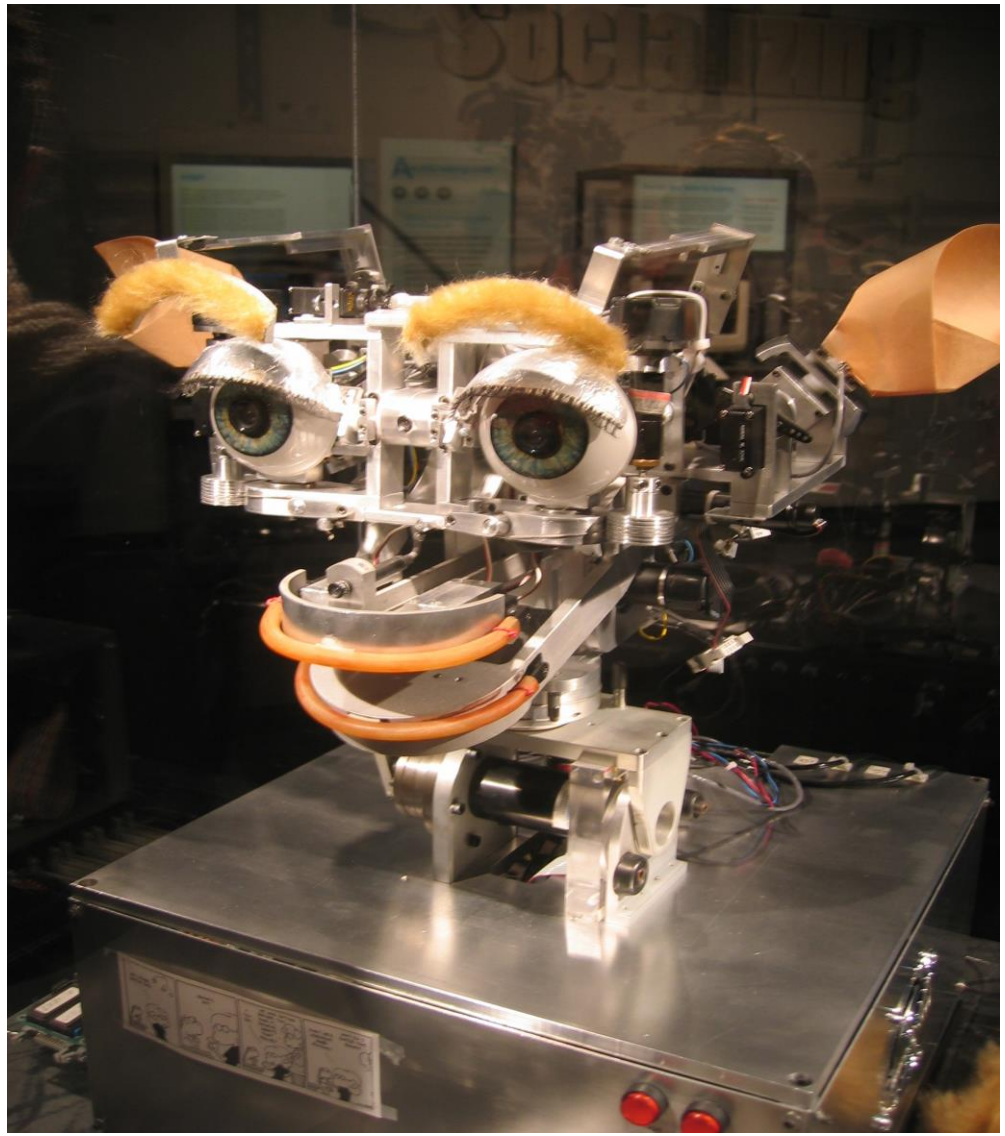

Menu
0 %
08:46
LSM303.h
LSM303.cpp
Trash


EEG-SMT
http://openEEG.
urceforge.net
OLIMEX
w.olimex.com

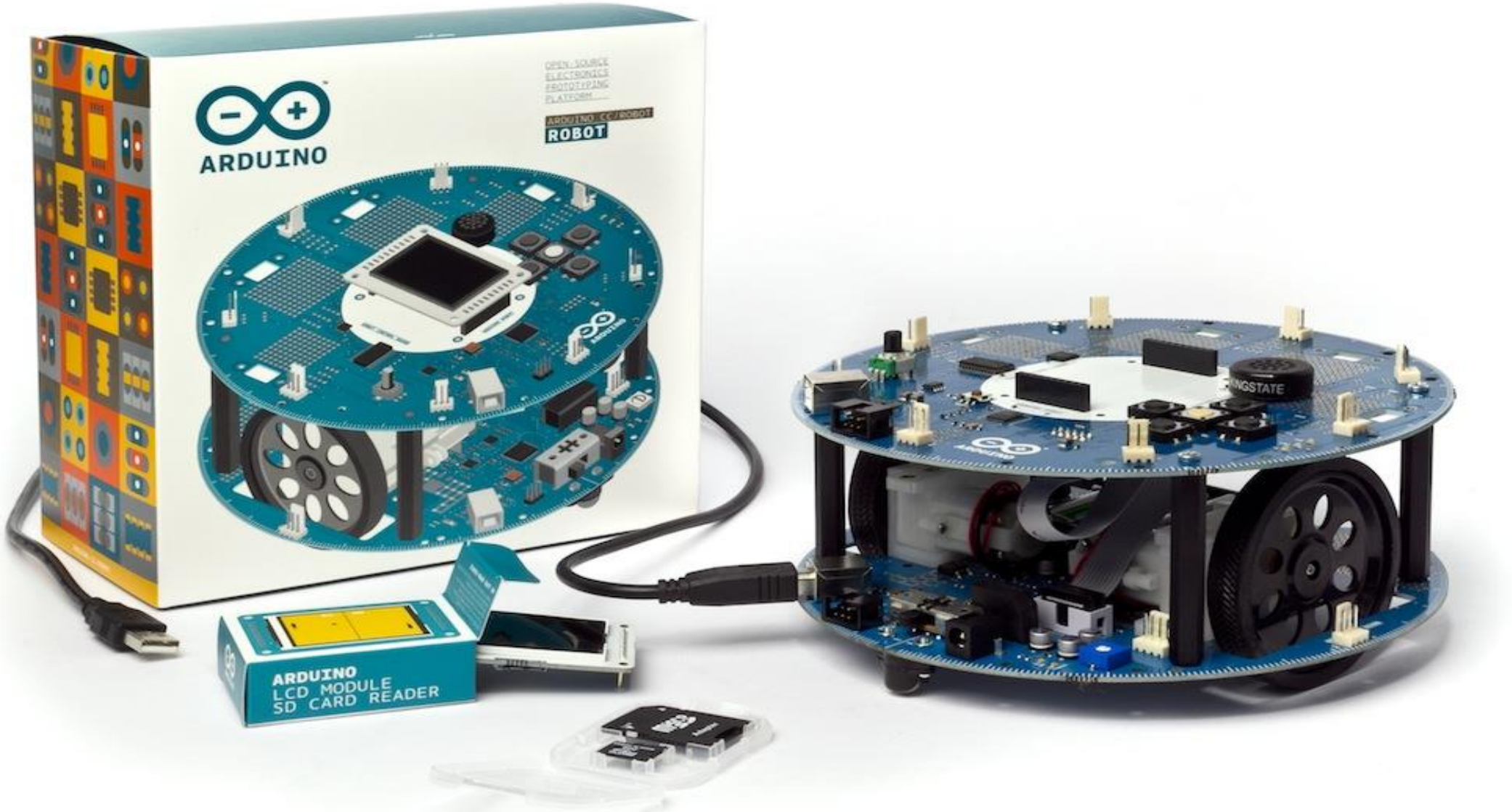
Роботите са вече у дома ...

Source: By Nohau - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=17987534

Social Robotics:)

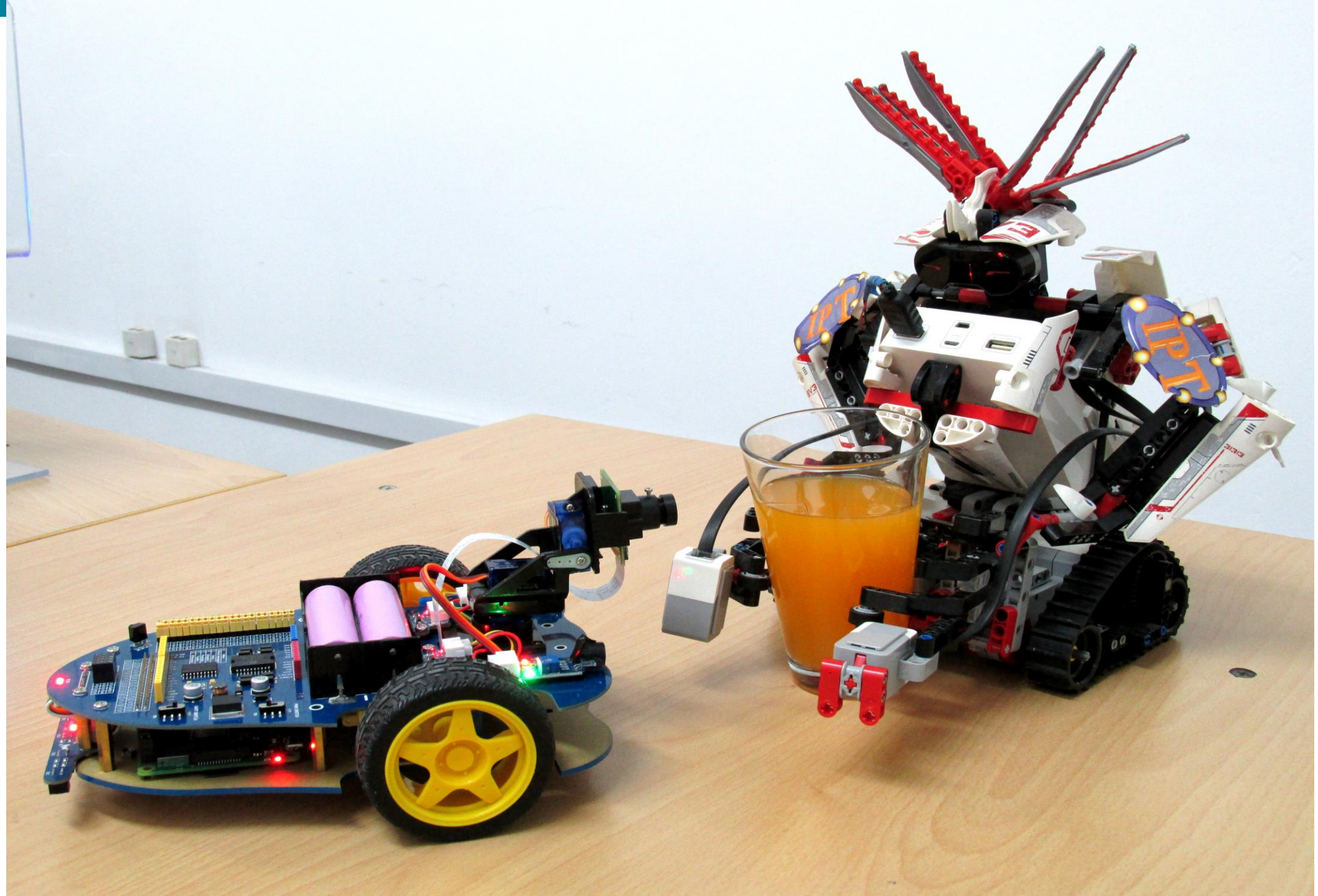


Обучение чрез програмиране на малки роботи

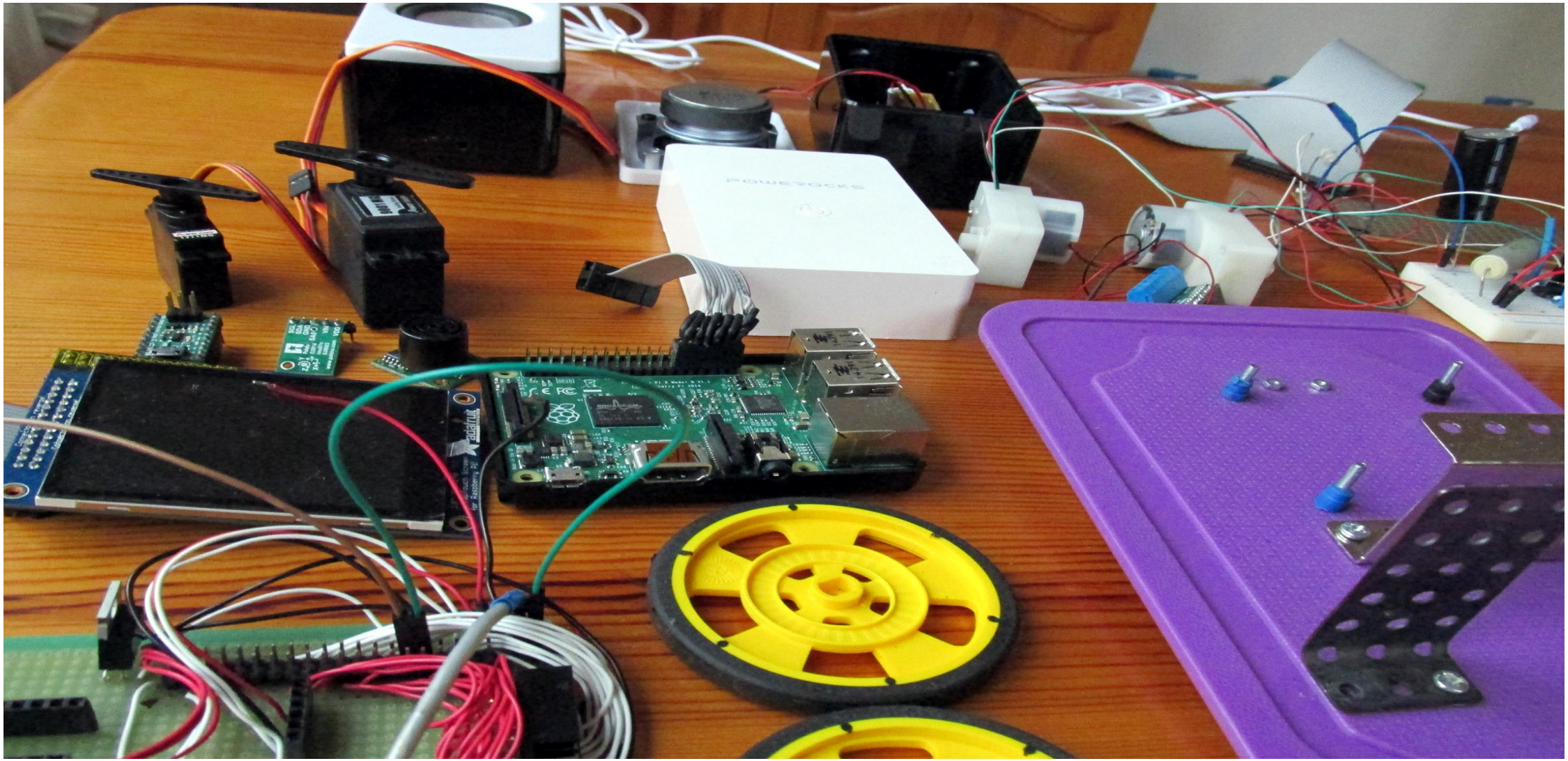


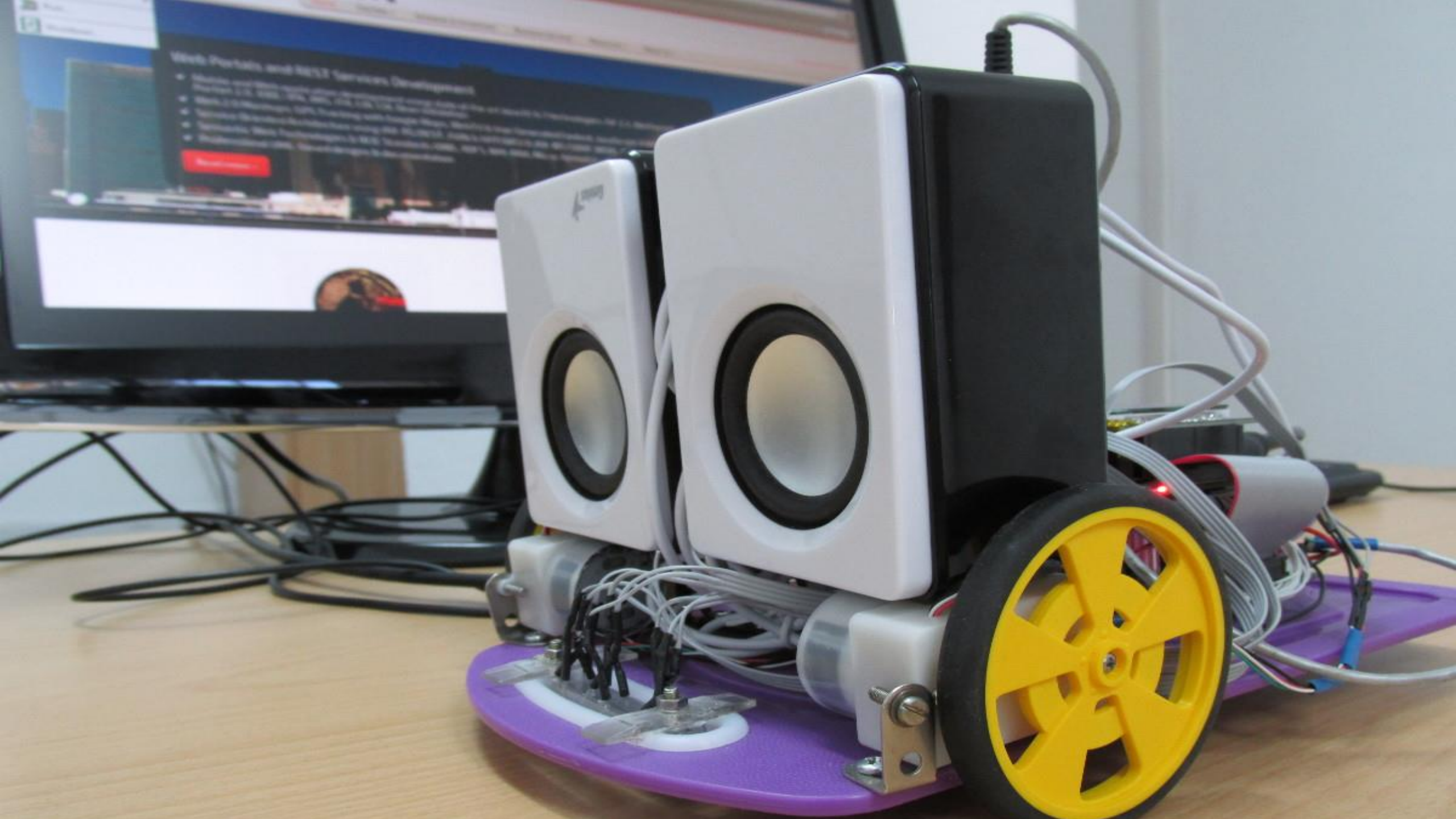
LeJaRo: Lego® Java Robot

- Modular – *3 motors (with encoders)* – one driving each track, and third for robot clamp.
- Three sensors: *touch sensor* (obstacle avoidance), *light color sensor* (follow line), *IR sensor* (remote).
- LeJaRo is programmed in Java using ***LeJOS*** library.
- More information about LeJaRo: http://robolearn.org/lejaro/
- Programming examples available @GitHub: https://github.com/iproduct/course-social-robotics/tree/master/motors_demo

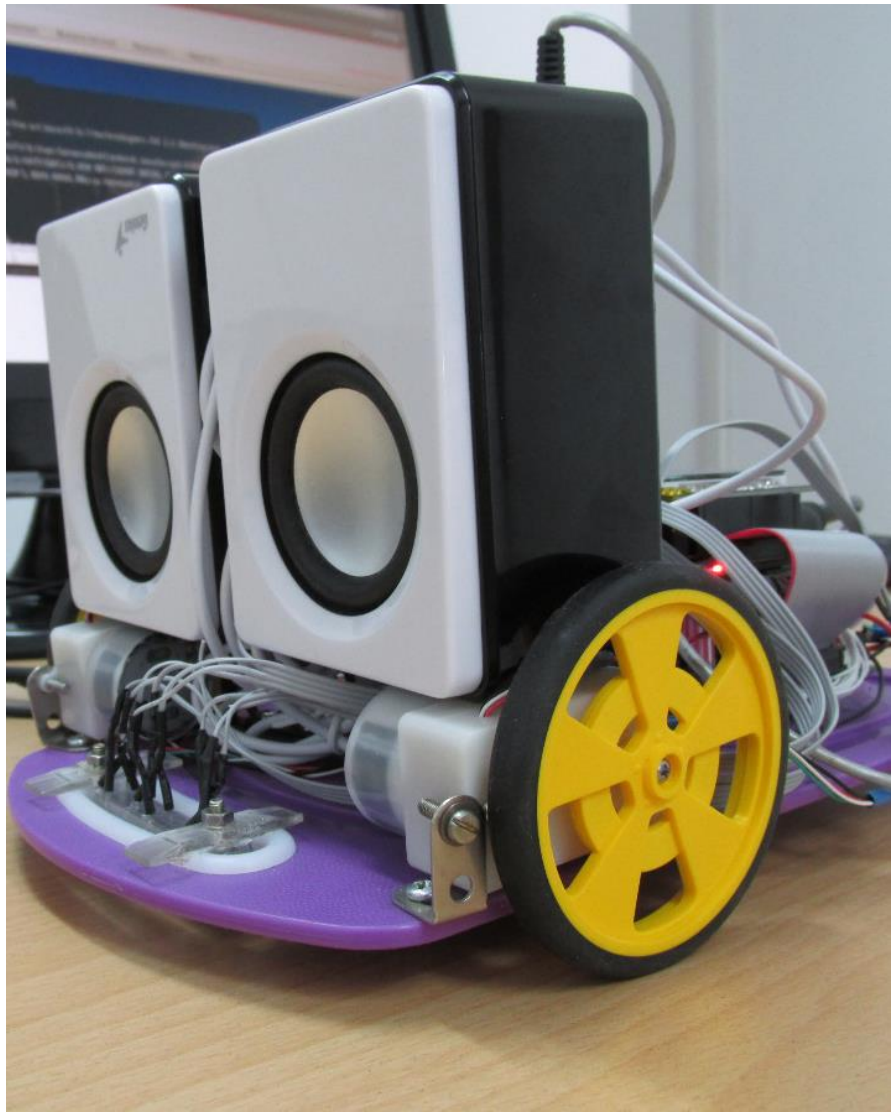

IPT
IPT

Meet IPTPI :)



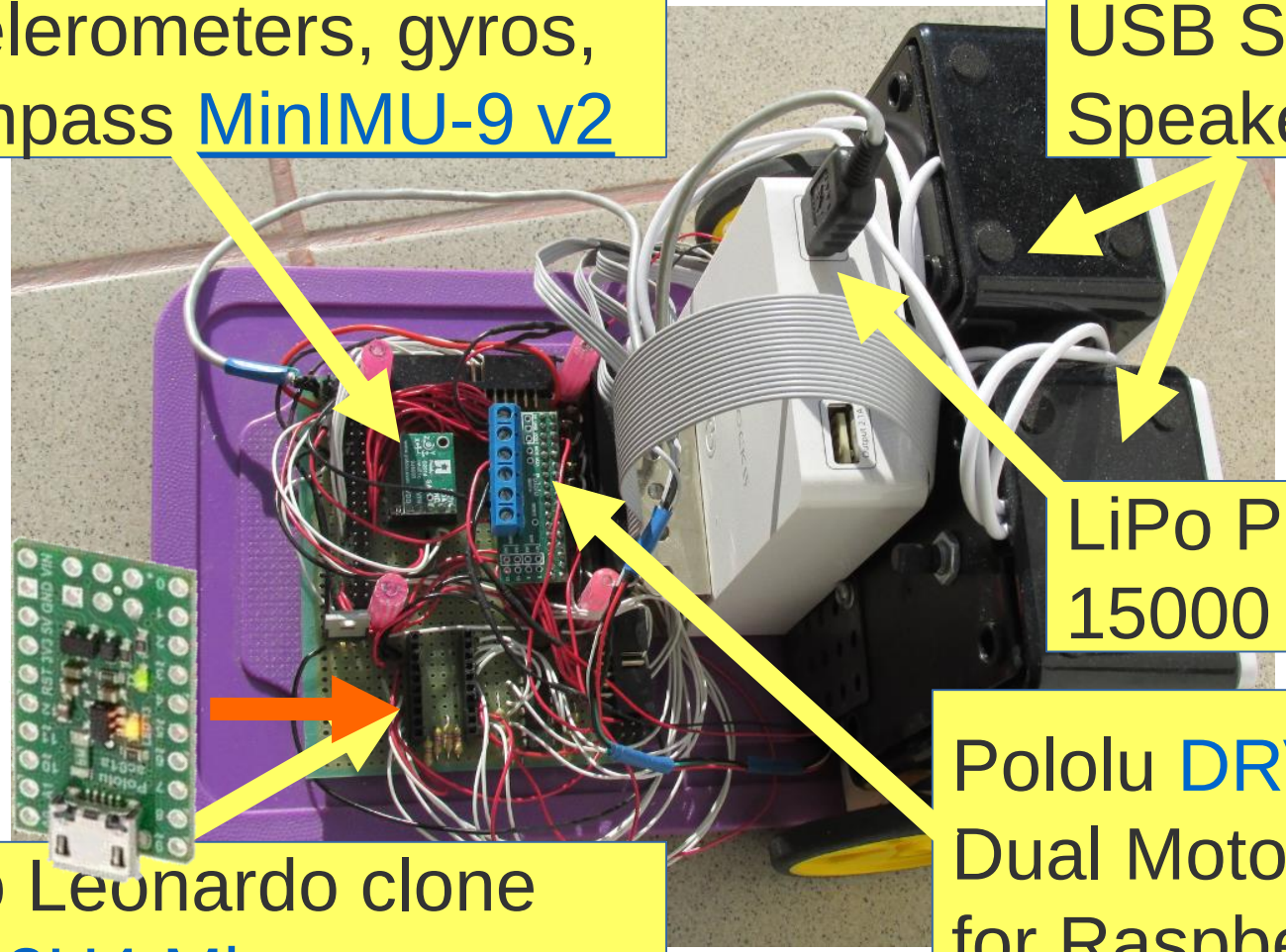


IPTPI: Raspberry Pi + Ardunio Robot



- Raspberry Pi 2 (quad-core ARMv7 @ 900MHz) + Arduino Leonardo clone A-Star 32U4 Micro
- *Optical encoders* (custom), IR optical array, 3D accelerometers, gyros, and compass MinIMU-9 v2
- **IPTPI** is programmed in Python, Java and Go using: Wiring Pi, Numpy, Pandas, Scikit-learn, Pi4J, Reactor, RxJava, GPIO(Go)

IPTPI: Raspberry Pi + Ardunio Robot



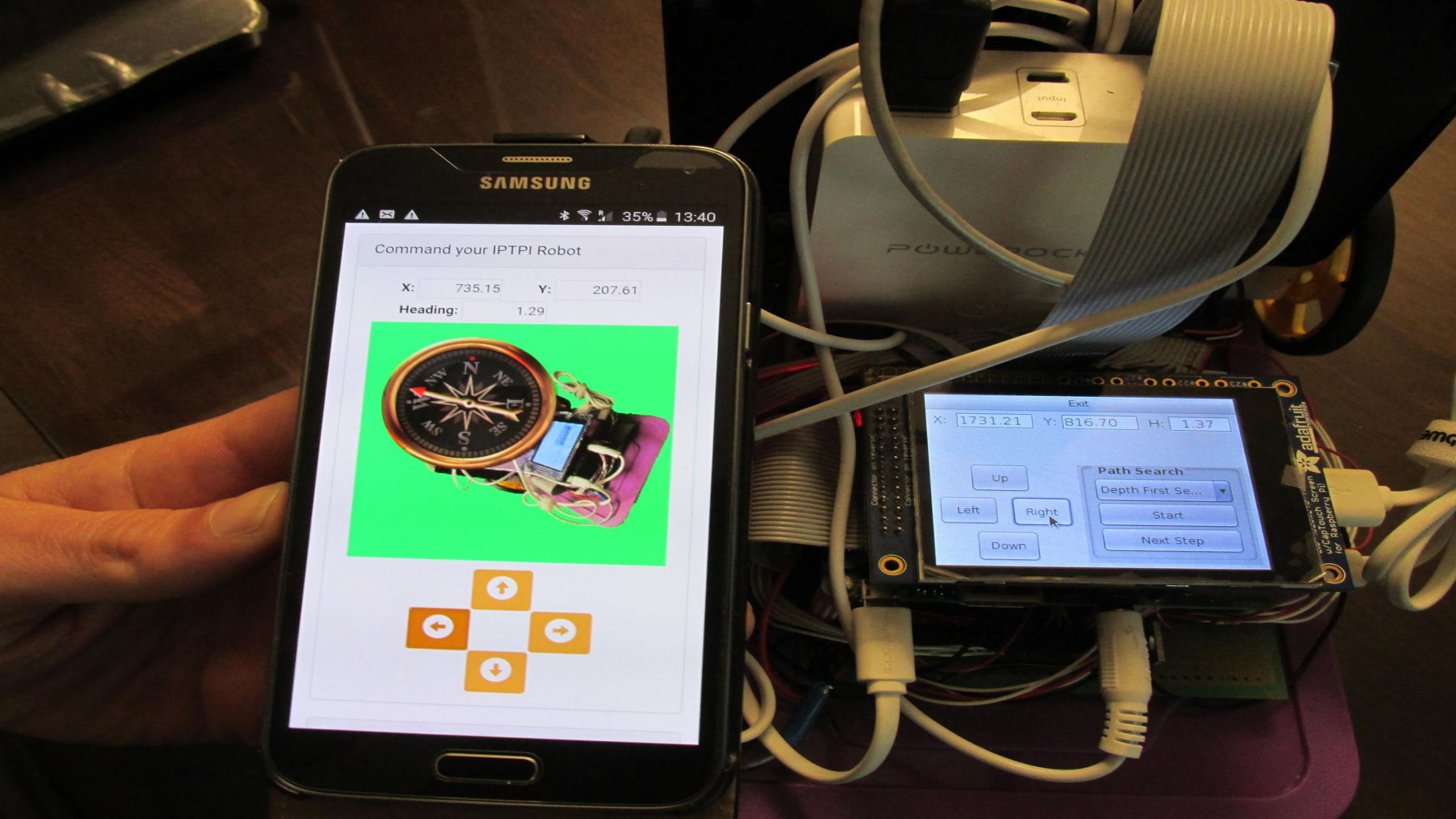
3D accelerometers, gyros, and compass MinIMU-9 v2

USB Stereo Speakers - 5V

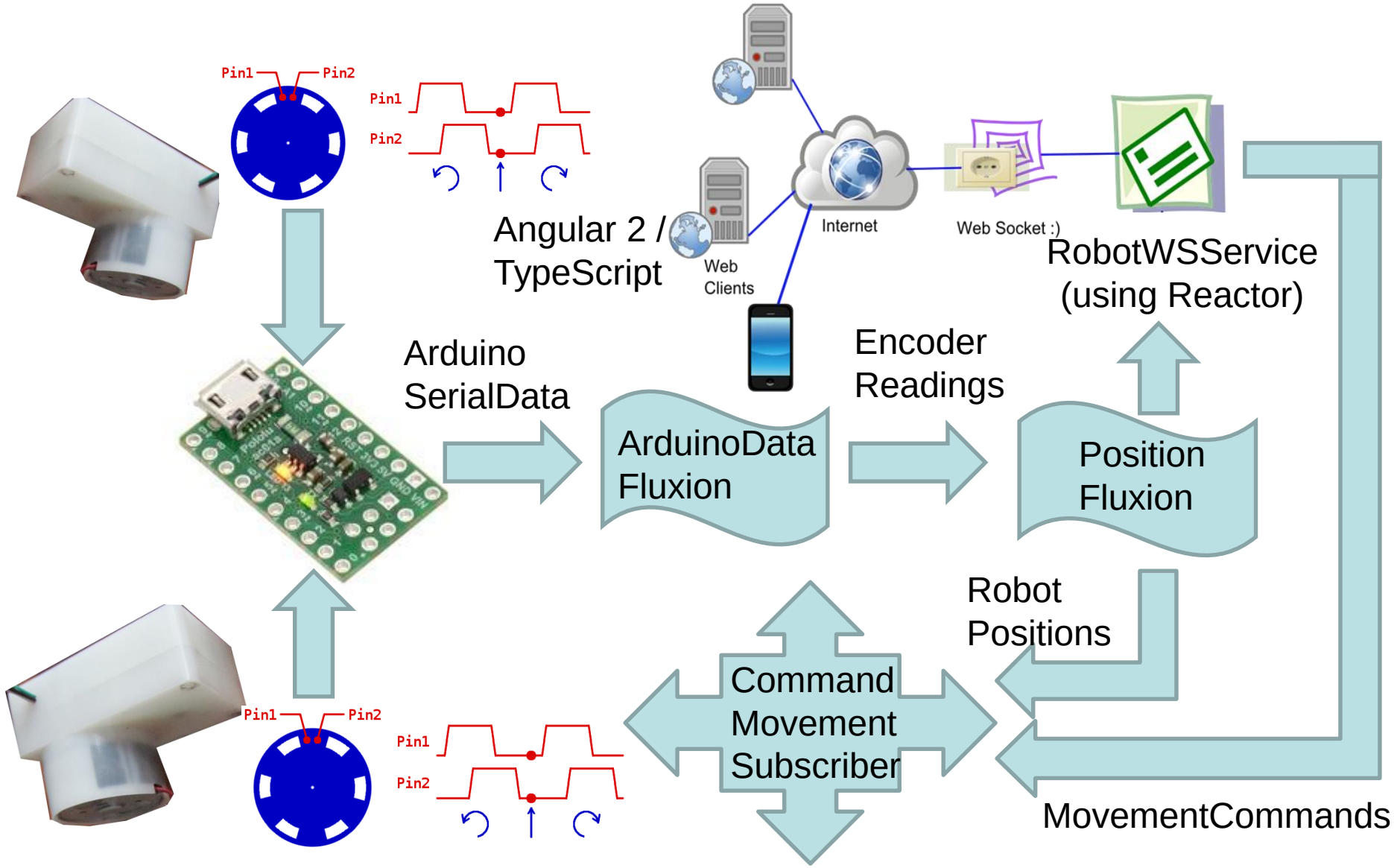
LiPo Powebank 15000 mAh

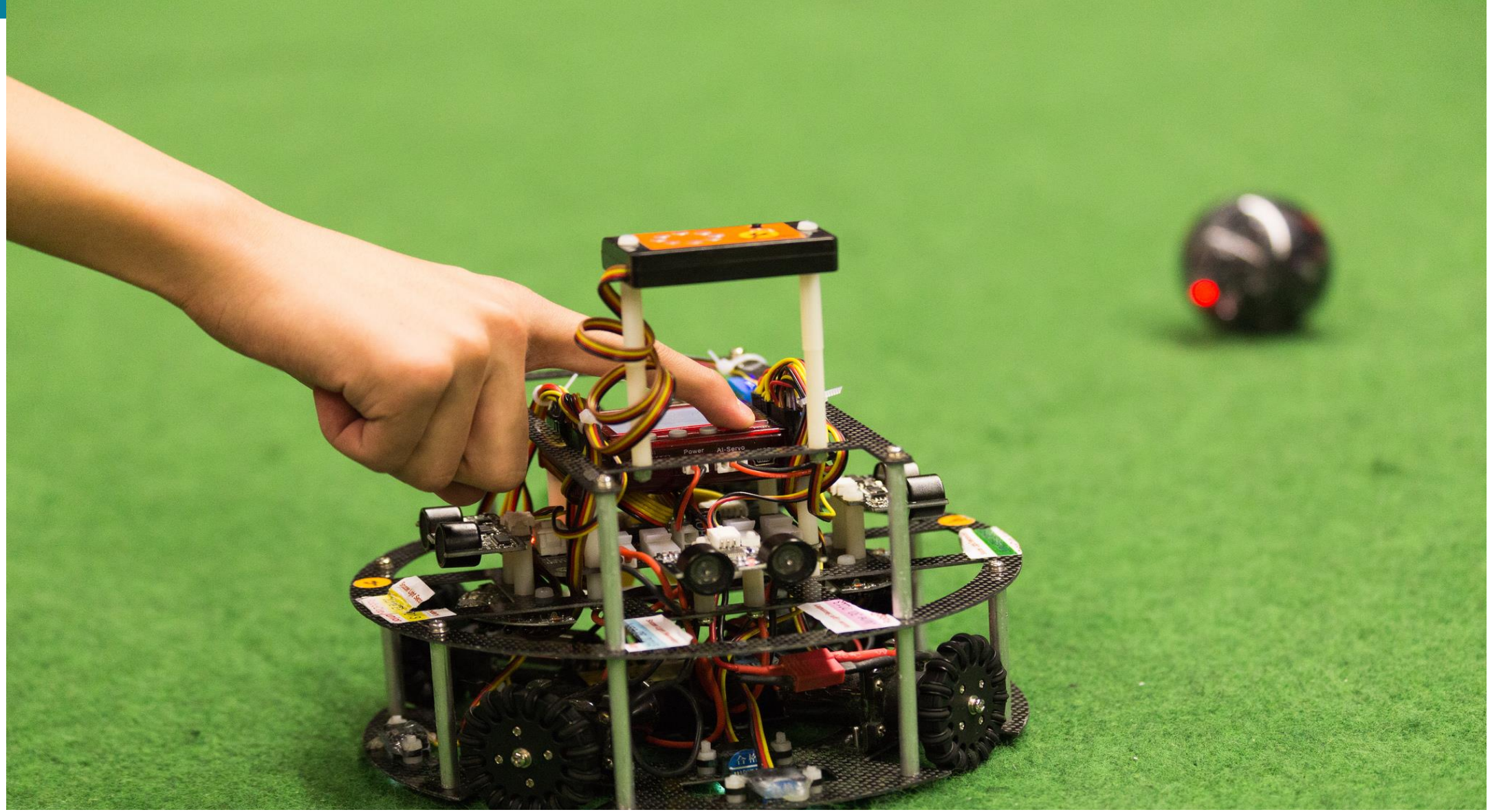
Pololu DRV8835 Dual Motor Driver for Raspberry Pi

Arduino Leonardo clone A-Star 32U4 Micro


SAMSUNG
35% 13:40
Command your IPTPI Robot
X: 735.15
Y: 207.61
Heading: 1.29
Exit
X: 1731.21
Y: 816.70
H: 1.37
Up
Left
Right
Down
Path Search
Depth First Se...
Start
Next Step

IPTPI Reactive Streams





RoboCup 2013 – Eindhoven by Albert van Breemen
[https://www.flickr.com/photos/robocup2013/10151792836]

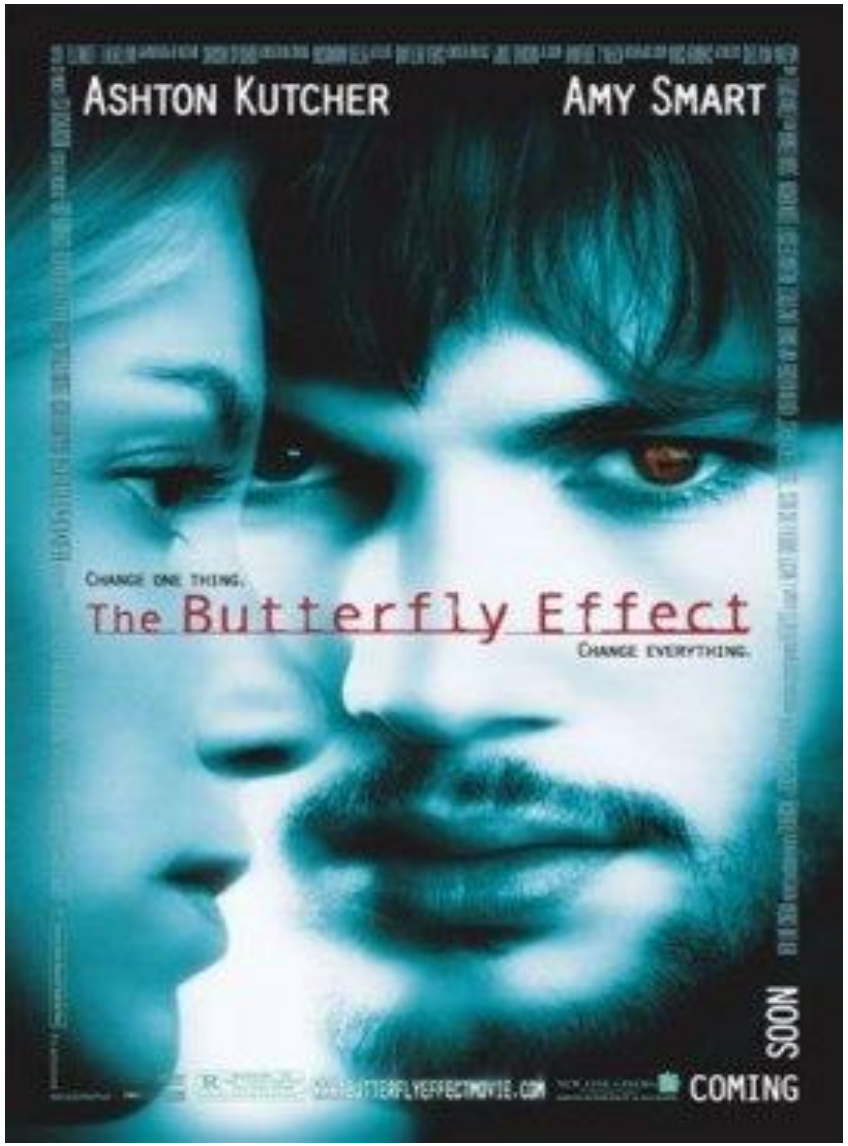
Functional Reactive Programming

Imperative and Reactive

We live in a Connected Universe

... there is hypothesis that all the things in the Universe are intimately connected, and you can not change a bit without changing all.

Action – Reaction principle is the essence of how Universe behaves.



Source: https://en.wikipedia.org/wiki/The_Butterfly_Effect#/media/File:Butterflyeffect_poster.jpg
Fair use, File:Butterflyeffect poster.jpg, Uploaded by Yaminator, Uploaded: 8 July 2008

Imperative vs. Reactive

- Reactive Programming: using static or dynamic data flows and propagation of change
- Example: a := b + c
- Functional Programming: evaluation of mathematical functions, avoids changing-state and mutable data, declarative programming
- Side effects free => much easier to understand and predict the program behavior.

Ex. (RxGo):

```
observable := rxgo.Just("Hello", "Reactive", "World", "from", "RxGo")().
        Map(ToUpper). // map to upper case
        Filter(LengthGreaterThan4) // greaterThan4 func filters values > 4
for item := range observable.Observe() {
        fmt.Println(item.V)
}
```

Functional Reactive Programming (FRP)

- According to Connal Elliot's (ground-breaking paper at Conference on Functional Programming, 1997), FRP is:

  (a) Denotative

  (b) Temporally continuous

- FRP is asynchronous data-flow programming using the building blocks of functional programming (map, reduce, filter, etc.) and explicitly modeling time

Reactive Programming

- ReactiveX (Reactive Extensions) - open source polyglot (http://reactivex.io):

  Rx = Observables + Flow transformations + Schedulers

- Go: RxGo, Kotlin: RxKotlin, Java: RxJava, JavaScript: RxJS, Python: RxPY, C#: Rx.NET, Scala: RxScala, Clojure: RxClojure, C++: RxCpp, Ruby: Rx.rb, Python: RxPY, Groovy: RxGroovy, JRuby: RxJRuby, ...
- Reactive Streams Specification (http://www.reactive-streams.org/):
  - Publisher – provider of potentially unbounded number of sequenced elements, according to Subscriber(s) demand (backpressure): onNext* (onError | onComplete)
  - Subscriber, Subscription
  - Processor = Subscriber + Publisher

ReactiveX: Observable vs. Iterable

Example code showing how similar high-order functions can be applied to an Iterable and an Observable

`Iterable`

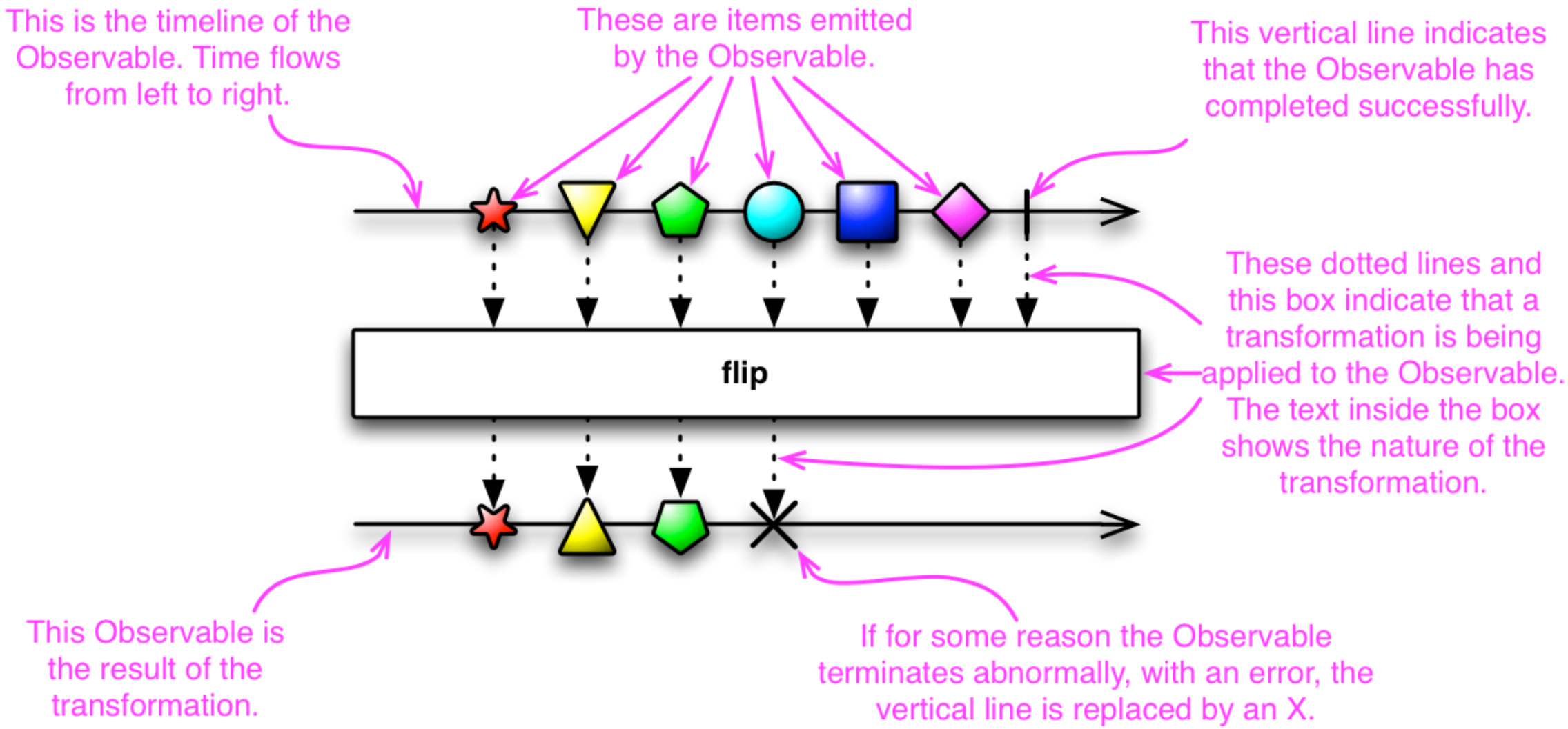
```
getDataFromLocalMemory()
  .skip(10)
  .take(5)
  .map({ s -> return s + " transforme
d" })
  .forEach({ println "next => " + it
})
```

`Observable`

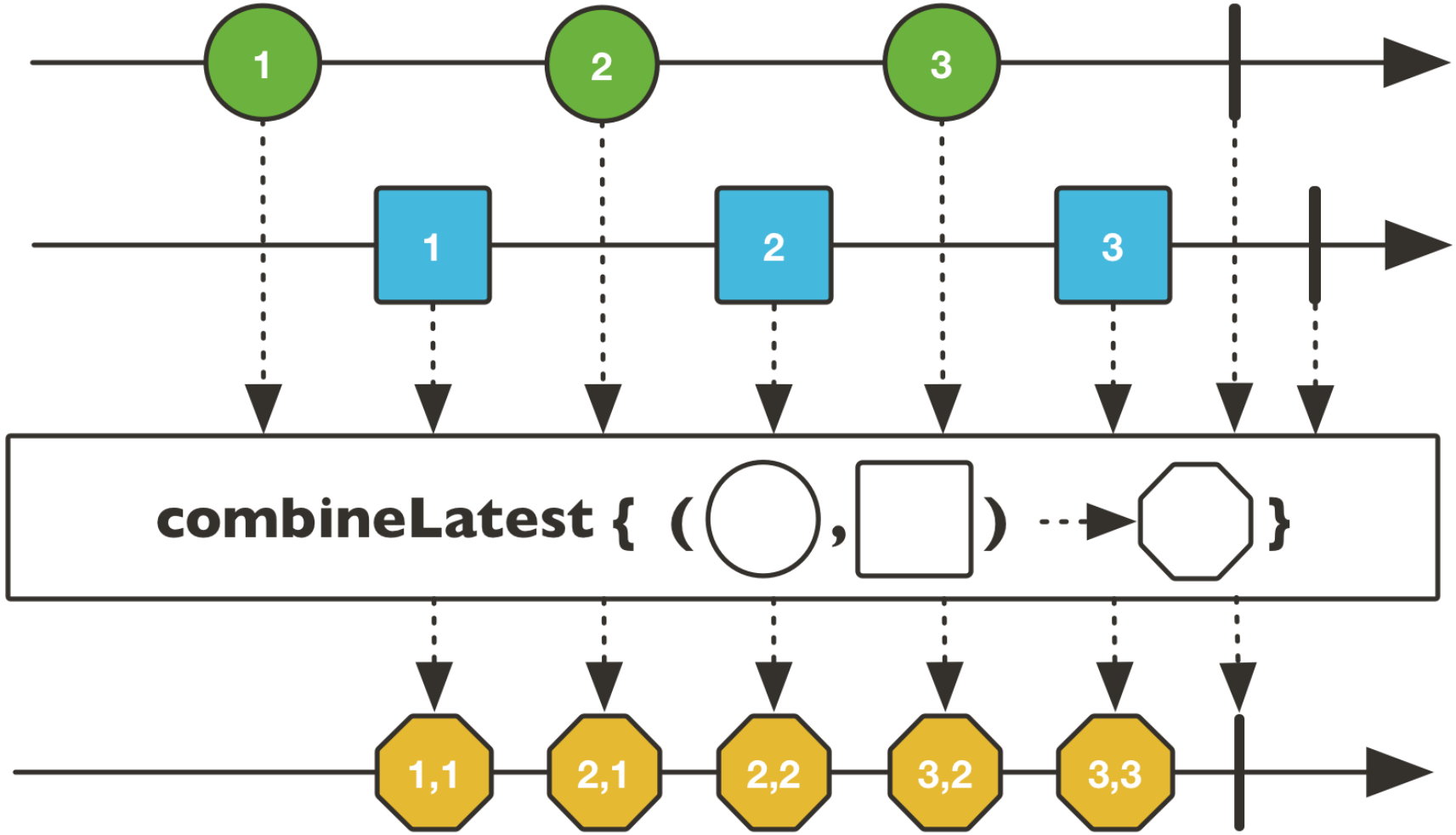
```
getDataFromNetwork()
  .skip(10)
  .take(5)
  .map({ s -> return s + " transformed"
})
  .subscribe({ println "onNext => " + it
})
```

You can think of the Observable class as a **"push"** equivalent to Iterable, which is a **"pull."** With an Iterable, the consumer pulls values from the producer and the thread blocks until those values arrive. By contrast, with an Observable the producer pushes values to the consumer whenever values are available. This approach is more flexible, because values can arrive synchronously or asynchronously.

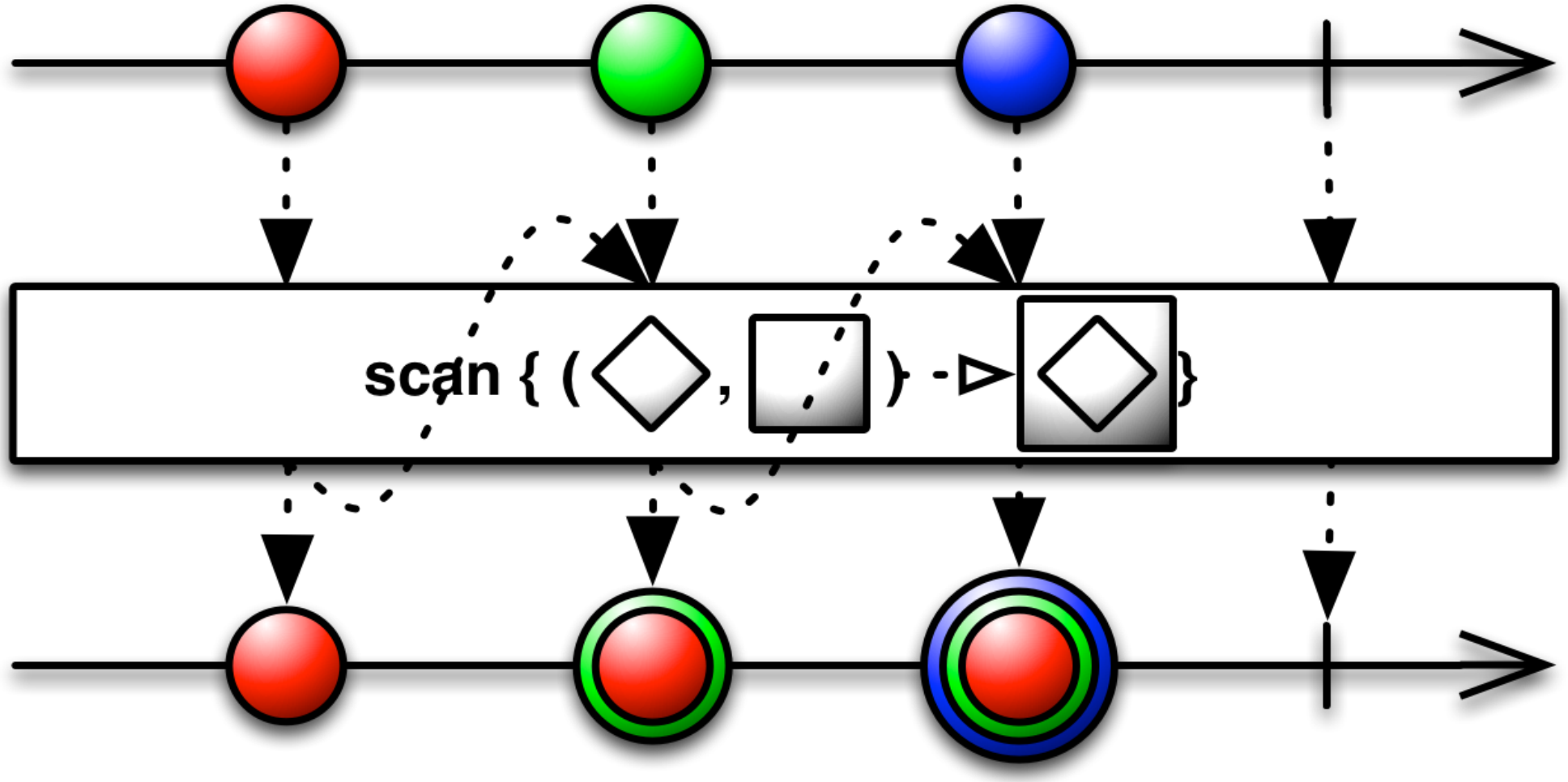
ReactiveX Observable – Marble Diagrams



Example: CombineLatest



Redux == Rx Scan Opearator



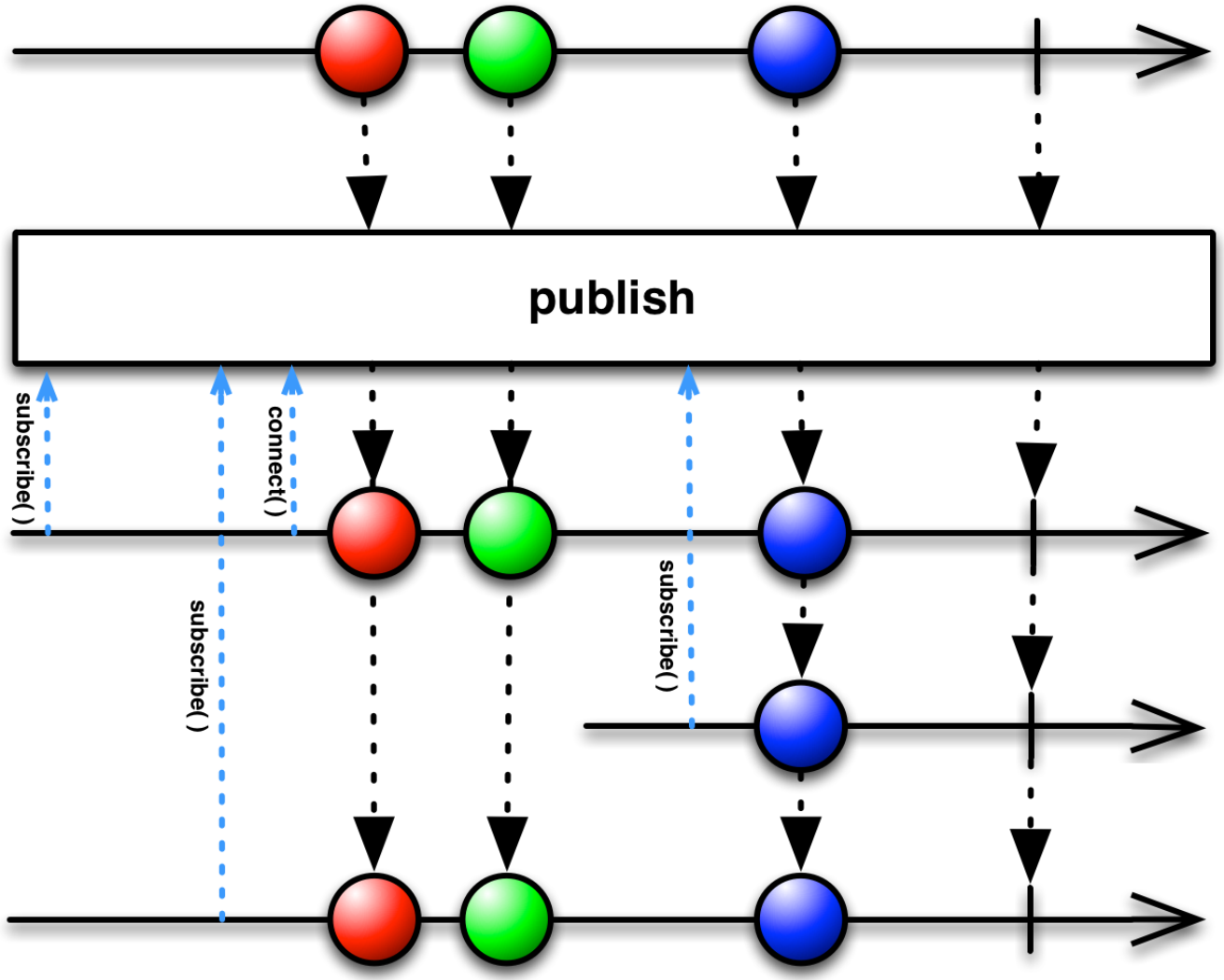
Source: RxJava 2 API documentation, http://reactivex.io/RxJava/2.x/javadoc/

Hot and Cold Event Streams

- PULL-based (Cold Event Streams) – Cold streams (e.g. RxJava Observable / Flowable or Reactor Flow / Mono) are streams that run their sequence when and if they are subscribed to. They present the sequence from the start to each subscriber.
- PUSH-based (Hot Event Streams) – emit values independent of individual subscriptions. They have their own timeline and events occur whether someone is listening or not. When subscription is made observer receives current events as they happen.
- *Example:* mouse events

69

Converting Cold to Hot Stream



Source: RxJava 2 API documentation, http://reactivex.io/RxJava/2.x/javadoc/

Reactive Programming

❖ Microsoft® opens source polyglot project ReactiveX (Reactive Extensions) [http://reactivex.io]:

Rx = Observables + LINQ + Schedulers :)

Java: RxJava, JavaScript: RxJS, C#: Rx.NET, Scala: RxScala, Clojure: RxClojure, C++: RxCpp, Ruby: Rx.rb, Python: RxPY, Groovy: RxGroovy, JRuby: RxJRuby, Kotlin: RxKotlin ...

❖ Reactive Streams Specification [http://www.reactive-streams.org/] used by:

❖ (Spring) Project Reactor [http://projectreactor.io/]

❖ Actor Model – Akka (Java, Scala) [http://akka.io/]

Reactive Streams Spec.

- Reactive Streams – provides standard for asynchronous stream processing with non-blocking back pressure.
- Minimal set of interfaces, methods and protocols for asynchronous data streams
- April 30, 2015: has been released version 1.0.0 of Reactive Streams for the JVM (Java API, Specification, TCK and implementation examples)
- Java 9: java.util.concurrent.Flow

Reactive Streams Spec.

- **Publisher** – provider of potentially unbounded number of sequenced elements, according to Subscriber(s) demand.

Publisher.subscribe(Subscriber) => **onSubscribe onNext* (onError | onComplete)?**

- **Subscriber** – calls **Subscription.request(long)** to receive notifications
- **Subscription** – one-to-one **Subscriber** ↔ **Publisher**, request data and cancel demand (allow cleanup).
- **Processor** = **Subscriber** + **Publisher**

FRP = Async Data Streams

- **FRP** is asynchronous data-flow programming using the building blocks of functional programming (e.g. map, reduce, filter) and explicitly modeling time
- Used for GUIs, robotics, and music. Example (RxJava):

```
Observable.from(
     new String[]{"Reactive", "Extensions", "Java"})
    .take(2).map(s -> s + " : on " + new Date())
    .subscribe(s -> System.out.println(s));
```

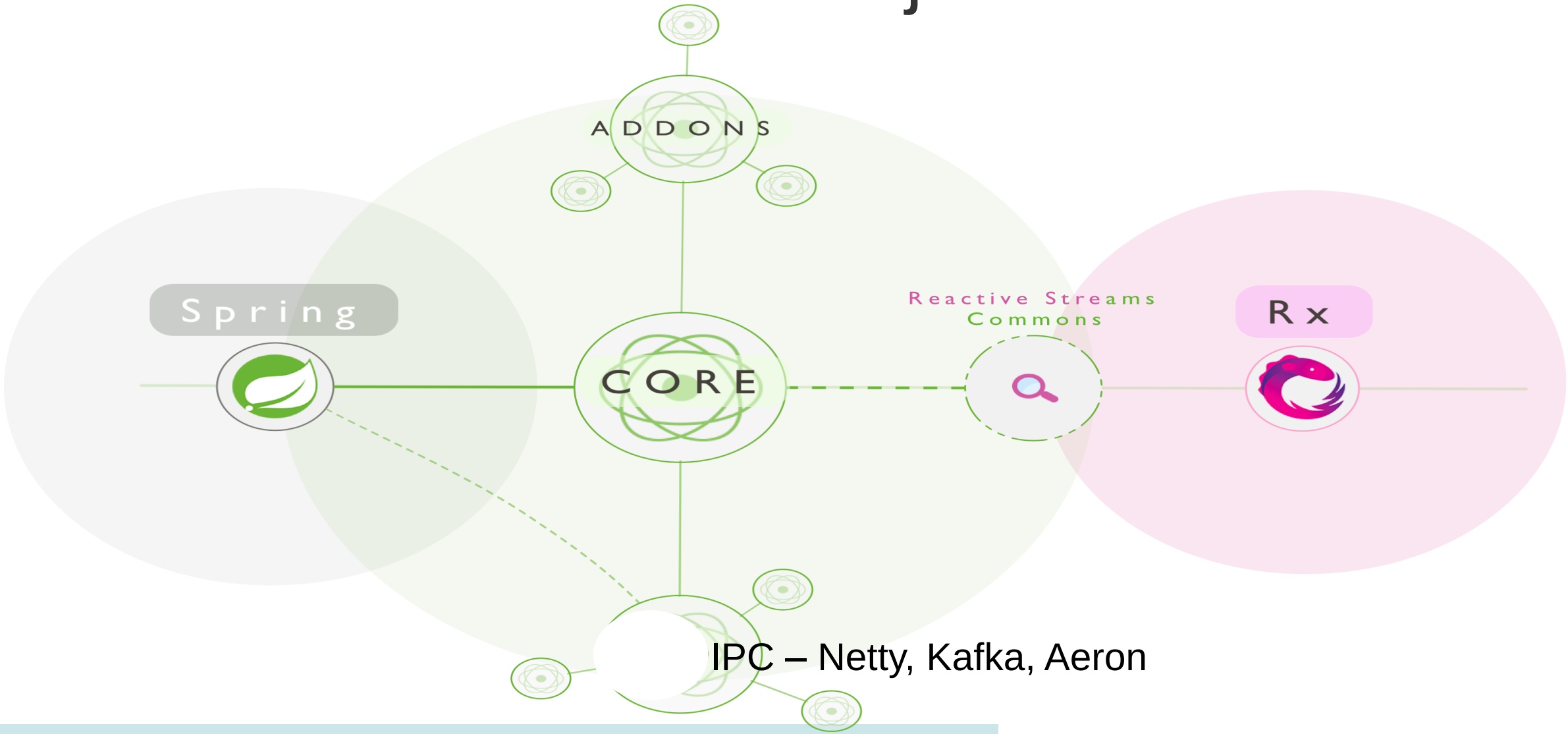
Result:

```
Reactive : on Wed Jun 17 21:54:02 GMT+02:00 2015
Extensions : on Wed Jun 17 21:54:02 GMT+02:00 2015
```

Project Reactor

- Reactor project allows building high-performance (low latency high throughput) non-blocking asynchronous applications on JVM.
- Reactor is designed to be extraordinarily fast and can sustain throughput rates on order of 10's of millions of operations per second.
- Reactor has powerful API for declaring data transformations and functional composition.
- Makes use of the concept of Mechanical Sympathy built on top of Disruptor / RingBuffer.

Reactor Projects



https://github.com/reactor/reactor, Apache Software License 2.0

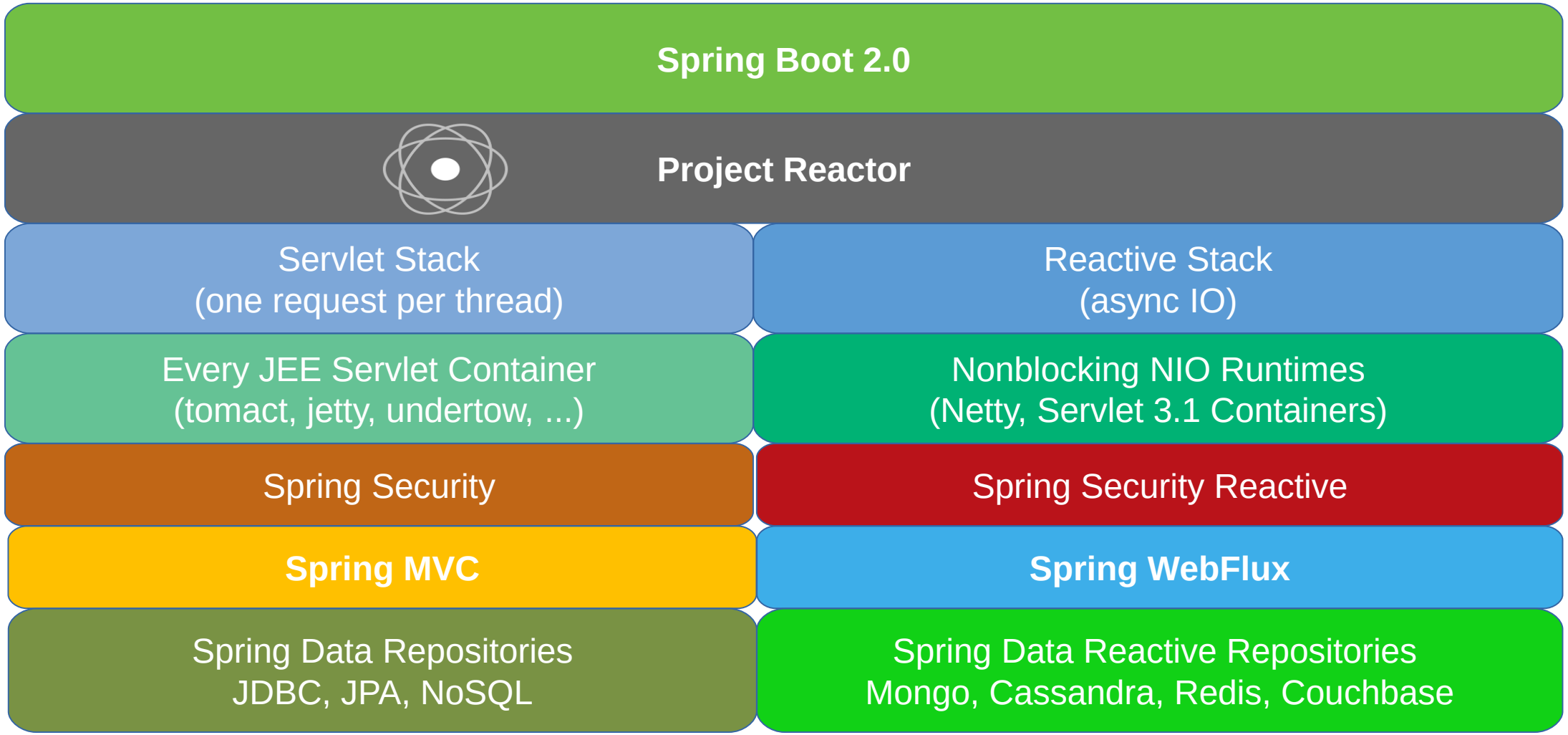
Hot Stream Example - Reactor

```
EmitterProcessor<String> emitter =
        EmitterProcessor.create();
FluxSink<String> sink = emitter.sink();
emitter.publishOn(Schedulers.single())
        .map(String::toUpperCase)
        .filter(s -> s.startsWith("HELLO"))
        .delayElements(Duration.of(1000, MILLIS))
.subscribe(System.out::println);
sink.next("Hello World!"); // emit - non blocking
sink.next("Goodbye World!");
sink.next("Hello Trayan!");
Thread.sleep(3000);
```

Top New Features in Spring 5

- Reactive Programming Model
- Spring Web Flux
- Reactive DB repositories & integrations + hot event streaming: MongoDB, CouchDB, Redis, Cassandra, Kafka
- JDK 8+ and Java EE 7+ baseline
- Testing improvements – WebTestClient (based on reactive WebFlux WebClient)
- Kotlin functional DSL

Spring 5 Main Building Blocks



Source: https://spring.io

Introduction to Machine Learning

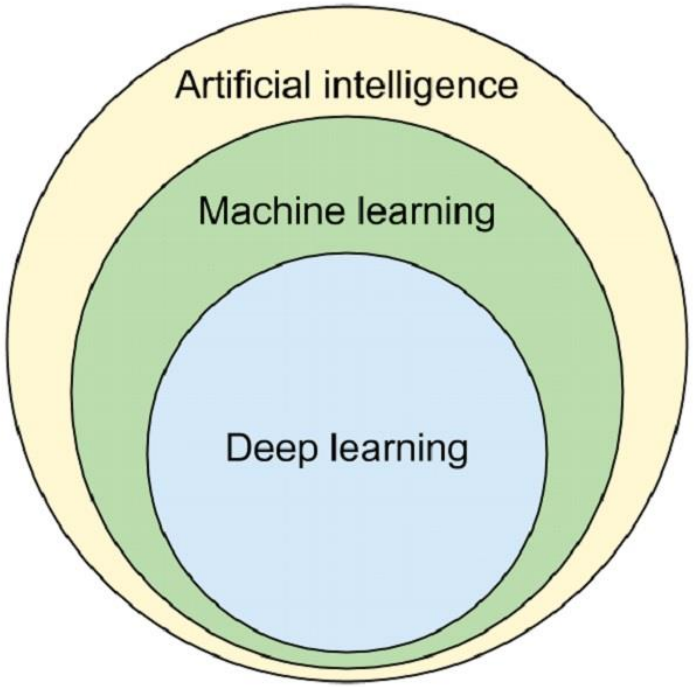
What Is Machine Learning?

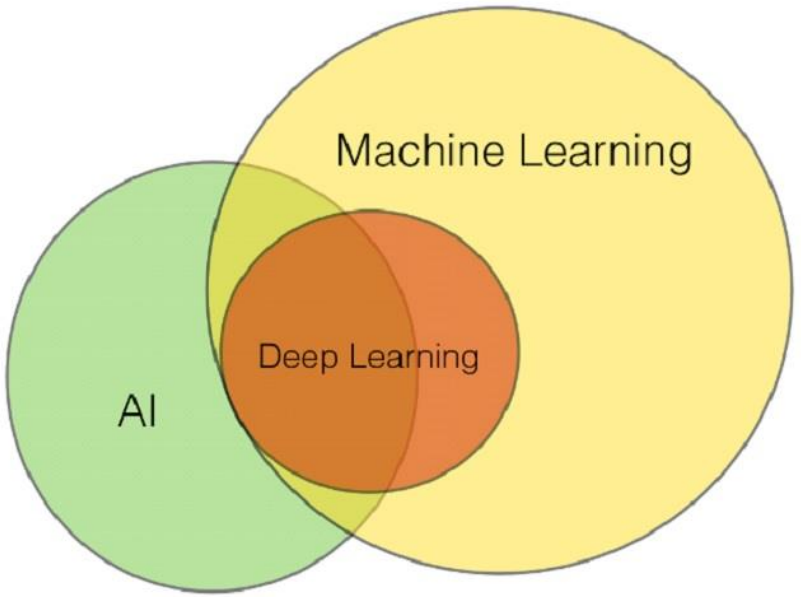
- **Machine Learning (ML)** –the study of computer algorithms that improve automatically through experience. A subset of Artificial Intelligence (AI).
- ML algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so.
- ML algorithms are used in a wide variety of applications, where it is difficult or unfeasible to develop conventional algorithms for the task – e.g. computer vision, email filtering, digital assistants, natural language processing (NLP), recommendations, personalized online advertising, chatbots, fraud detection, cybersecurity, medical image analysis, self-driving cars, self-driving databases, and much, much more ...
- ML is closely connected with computational statistics, mathematical optimization, data mining, predictive analytics.

Machine Learning and Artificial Intelligence

- ML learns and predicts based on passive observations, whereas AI implies an agent interacting with the environment to learn and take actions that maximize its chance of successfully achieving its goals.

Judea Pearl, The Book of Why, 2018





Pictures by: Yakoove - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=95589683
By Yakoove - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=95589710

Types of Learning Algorithms: Supervised Learning

- **Supervised learning** – build a mathematical model of a set of data that contains both the inputs and the desired outputs.
- Training data – a set (matrix) of training examples (feature vectors), having one or more inputs and the desired output. Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new inputs.
- Examples: active learning, classification and regression.
  - Active learning – learning algorithm can interactively query a user (teacher or oracle) to label new data points with the desired outputs (optimal experimental design).
  - Classification algorithms – when the outputs are restricted to a limited set of values
  - Regression algorithms – when the outputs may have any numerical value within a range.
  - Similarity learning – using a similarity function measuring how similar two objects are – ranking, recommender systems, visual identity tracking, face and speaker verification.

Types of Learning Algorithms: Unsupervised Learning

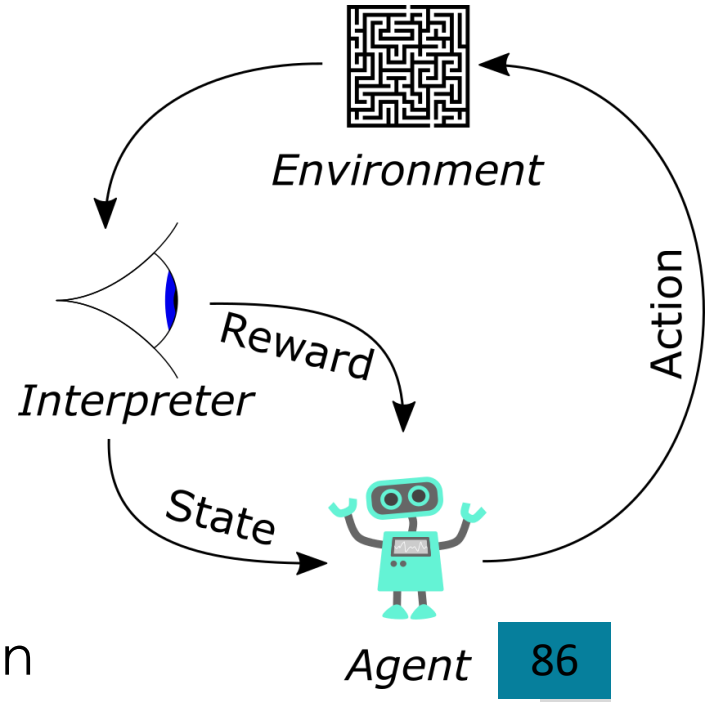
- **Unsupervised learning** – takes a data set that contains only inputs, and find structure in the data, like grouping or clustering of data points by identifying data commonalities
- Algorithms learn from test data that has not been labeled, classified or categorized.
- Applications: density function estimation, summarizing and explaining data features.
- Cluster analysis is the assignment of a set of observations into subsets (called clusters) so that observations within the same cluster are similar according to one or more predesignated criteria, while observations drawn from different clusters are dissimilar.
- Different clustering techniques make different assumptions on the structure of the data, often defined by some similarity metric and evaluated, for example, by internal compactness, or the similarity between members of the same cluster, and separation, the difference between clusters. Other methods are based on estimated density and graph connectivity.

Types of Learning Algorithms: Semi-supervised Learning

- **Semi-supervised learning** – falls between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data). During training, it uses a smaller labeled data set to guide classification and feature extraction from a larger, unlabeled data set.
- In **weakly supervised learning**, the training labels are noisy, limited, or imprecise; however, these labels are often cheaper to obtain, resulting in larger effective training sets.

Types of Learning Algorithms: Reinforcement Learning

- **Reinforcement learning** – concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward
- Due to its generality, the field is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics and genetic algorithms.
- In ML, the environment is typically represented as a Markov decision process (MDP). Many reinforcement learning algorithms use dynamic programming techniques.
- Reinforcement learning algorithms do not assume knowledge of an exact mathematical model of the MDP, and are used when exact models are infeasible.
- Examples: reinforcement learning algorithms are used in autonomous vehicles or in learning to play a game against human



Types of Learning Algorithms: Feature learning – I

- **Feature learning (representation learning algorithms)** – discover better representations of the inputs provided during training. Examples: principal components analysis and cluster analysis.
- Attempt to preserve the information in their input but also transform it in a way that makes it useful, often as a pre-processing step before performing classification or predictions.
- Allows a machine to both learn the features and use them to perform a specific task.
- Feature learning can be either supervised or unsupervised. In supervised feature learning, features are learned using labeled input data. Examples: artificial neural networks, multilayer perceptrons, and supervised dictionary learning. In unsupervised feature learning, features are learned with unlabeled input data. Examples: dictionary learning, independent component analysis, autoencoders, matrix factorization, and clustering.

Types of Learning Algorithms: Feature learning – II

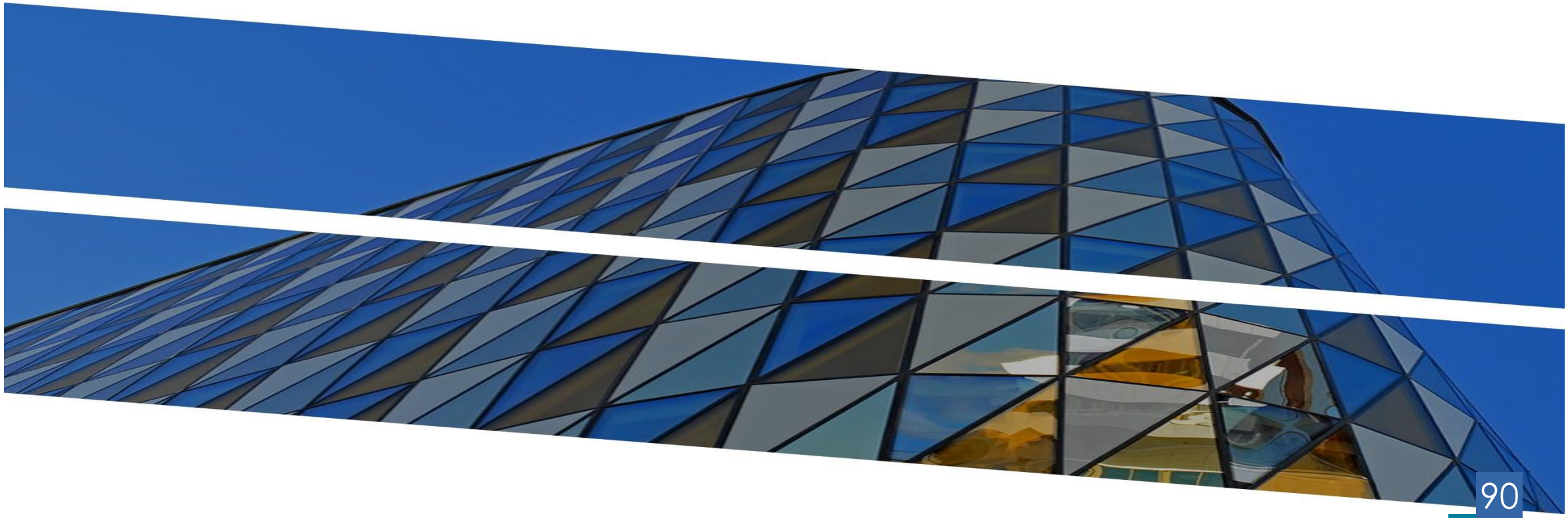
- Manifold learning algorithms attempt to do so under the constraint that the learned representation is low-dimensional.
- Sparse coding algorithms attempt to do so under the constraint that the learned representation is sparse, meaning that the mathematical model has many zeros.
- Multilinear subspace learning algorithms aim to learn low-dimensional representations directly from tensor representations for multidimensional data, without reshaping them into higher-dimensional vectors.
- Deep learning algorithms discover multiple levels of representation, or a hierarchy of features, with higher-level, more abstract features defined in terms of (or generating) lower-level features. It has been argued that an intelligent machine is one that learns a representation that disentangles the underlying factors of variation that explain the observed data.

MACHINE LEARNING

Supervised Learning		Unsupervised Learning		Deep Learning (semi-supervised)		
Regressors	Classifiers	Dimension Reducers	Clustering Methods	Unsupervised Pretrained Networks	Convolutional Neural Networks	Recurrent Neural Networks
Wireline log prediction Spatial interpolation Sesimic inversion (with numeric labels)	Automatic facies prediction from wireline logs Seismic inversion (with categ-orical labels)	Compressing high-dimensional data Increasing signal-to-noise ratio in data	AVO class prediction Seismic inversion (uncalibrated)	Seismic denoising Seismic multiple removal Seismic migration	Seismic structural feature/fault detection Automatic facies prediction from borehole imagery	Microseismic analysis Sedimentary process modelling Earthquake prediction

Picture by: https://wiki.seg.org/wiki/Machine_learning

Machine Learning Libraries



Top Machine Learning Libraries - I

- Apache Spark Mllib – fast, in-memory, big-data processing, which has also made it a go-to framework for ML, highly scalable, can be coupled with any Hadoop data source, supports Java, Scala, R and Python, new ML algorithms are constantly being added, and existing enhanced.

- H_20 + AutoML - open source, in-memory, distributed, fast, and scalable machine learning and predictive analytics platform allowing to learn from big data, written in Java, uses Distributed Key/Value store for accessing data, models, objects, etc. across all nodes and machines, uses distributed Map/Reduce, utilizes Java Fork/Join multi-threading, data read in parallel, distributed across the cluster, and stored in memory in a columnar format in a compressed way, data parser has built-in intelligence to guess the schema of the incoming dataset and supports data ingest from multiple sources in various formats, APIs: REST, Flow UI, H2O-R, H2O-Python, supports Deep Learning, Tree Ensembles, and Generalized Low Rank Models (GLRM).

Top Machine Learning Libraries - II

- Microsoft Azure ML Studio - Microsoft's ML framework runs in their Azure cloud, offering high-capacity data processing on a pay-as-you-go model. Its interactive, visual workspace can be used to create, test and iterate ML "experiments" using the built-in ML packages; they can then be published and shared as web services. Python or R custom coding is also supported. A free trial is available for new users.

- Amazon Machine Learning, Sagemaker - like Azure, the Amazon Machine Learning framework is cloud-based, supporting Amazon S3, Redshift and RDS. It combines ML algorithms with interactive tools and data visualizations which allow users to easily create, evaluate and deploy ML models. These models can be binary, categoric or numeric – making them useful in areas such as information filtering and predictive analytics. Includes support for labeling, data preparation, feature engineering, statistical bias detection, auto-ML, training, tuning, hosting, explainability, monitoring, and workflows. Provides integrated development environment (IDE) for ML.

Top Machine Learning Libraries - III

- Microsoft Distributed Machine Learning Toolkit (DMTK) - uses local data caching to make it more scalable and efficient on computing clusters that have limited resources on each node. Its distributed ML algorithms allow for fast training of gradient boosting, topic and word-embedding learning models:
  - LightLDA: Scalable, fast and lightweight system for large-scale topic modeling.
  - LightGBM: LightGBM is a fast, distributed, high performance gradient boosting (GBDT, GBRT, GBM or MART) framework based on decision tree algorithms, used for ranking, classification and many other machine learning tasks.
  - Distributed word embedding: a parallelization of the Word2Vec algorithm
  - Distributed Multi-sense Word Embedding (DMWE) - a parallelization of the Skip-Gram Mixture algorithm used for polysemous words.
- Google TensorFlow - open-source ML toolkit that has been applied in areas ranging from machine translation to biomedical science. Data flows are processed by a series of algorithms described by a graph. This allows for a flexible, multi-node architecture that supports multiple CPUs, GPUs, and TPUs. The recently released Tensorflow Lite adds support for neural processing on mobile phones.

Top Machine Learning Libraries - IV

- Caffe - developed by Berkeley AI Research, Caffe claims to offer one of the fastest implementations of convolutional neural networks. It was originally developed for machine vision projects but has since expanded to other applications, with its extensible C++ code designed to foster active development. Both CPU and GPU processing are supported.

- Veles - developed and released as open source by Samsung, Veles is a distributed ML platform designed for rapid deep-learning application development. Users can train various artificial neural net types – including fully connected, convolutional and recurrent. It can be integrated with any Java application, and its "snapshot" feature improves disaster recovery.

Top Machine Learning Libraries - V

- Massive Online Analysis (MOA) - developed at the University of Waikato in New Zealand, this open-source Java framework specializes in real-time mining of streaming data and large-scale ML. It offers a wide range of ML algorithms, including classification, regression, clustering, outlier detection and concept drift detection. Evaluation tools are also provided.

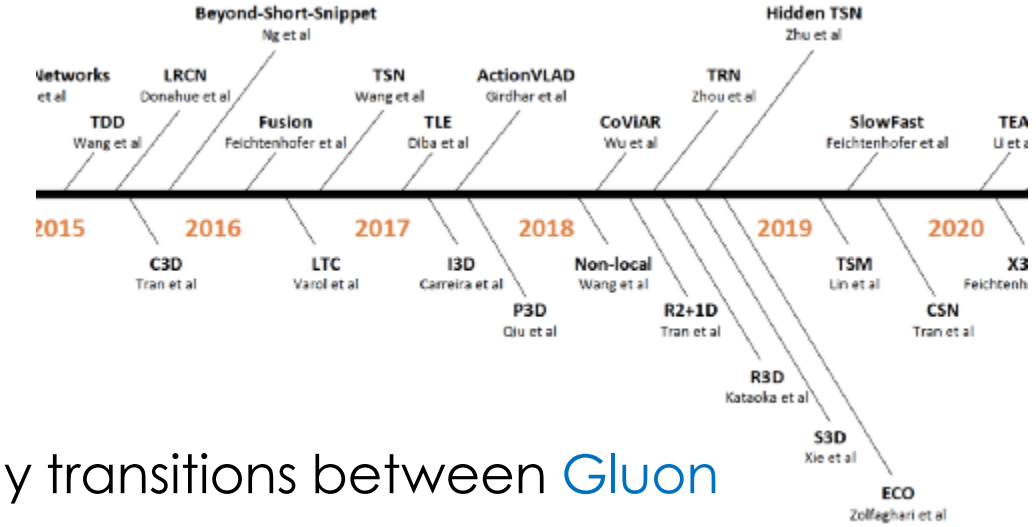
- Torch - Torch ML library makes ML easy and efficient, thanks to its use of the Lua scripting language and a GPU-first implementation. It comes with a large collection of community-driven ML packages that cover computer vision, signal processing, audio processing, networking and more. Its neural network and optimization libraries are designed to be both accessible and flexible.

Top Machine Learning Libraries - VI

- Keras – deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research. Follows the principle of progressive disclojure of complexity: it makes it easy to get started, yet it makes it possible to handle arbitrarily advanced use cases, only requiring incremental learning at each step – e.g. customizing the training loop by combining Keras functionality with the TensorFlow GradientTape:

```
optimizer = tf.keras.optimizers.Adam()
loss_fn = tf.keras.losses.kl_divergence
with tf.GradientTape() as tape:
    predictions = model(inputs)
    loss_value = loss_fn(targets, predictions)
   gradients = tape.gradient(loss_value, model.trainable_weights)
   optimizer.apply_gradients(zip(gradients, model.trainable_weights))
   …
```

Top Machine Learning Libraries



Apache MXNet:

- Hybrid Front-End – a hybrid front-end seamlessly transitions between Gluon eager imperative mode and symbolic mode to provide both flexibility and speed.
- GluonCV – a computer vision toolkit with rich model zoo – from object detection to pose estimation.
- GluonNLP – provides state-of-the-art deep learning models in NLP – for engineers and researchers to fast prototype research ideas and products.
- GluonTS – Gluon Time Series is the Gluon toolkit for probabilistic time series modeling, focusing on deep learning-based models.

Picture source: https://medium.com/apache-mxnet/which-one-is-the-best-algorithm-for-video-action-recognition-298fb5c4ad4f

Deep Learning

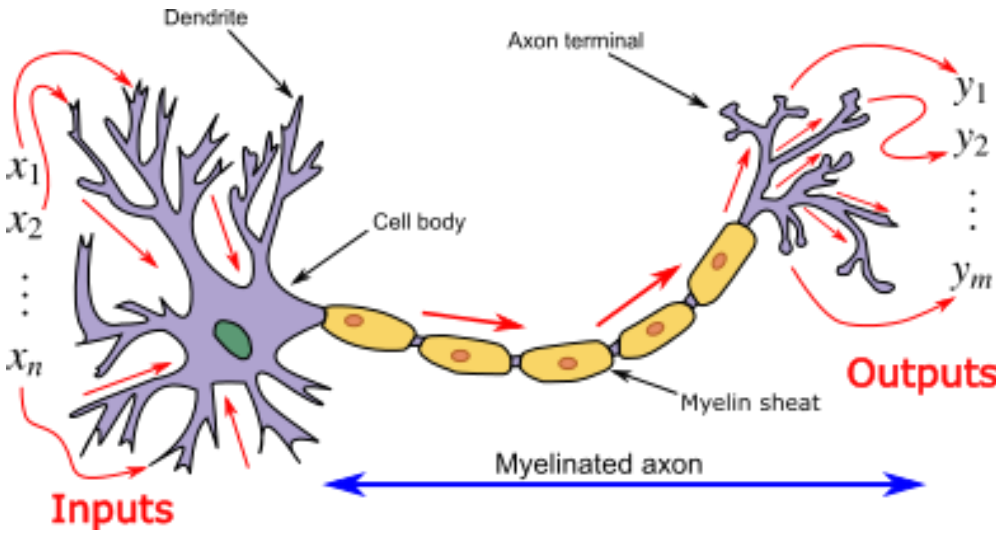
Convolutional Neural Networks

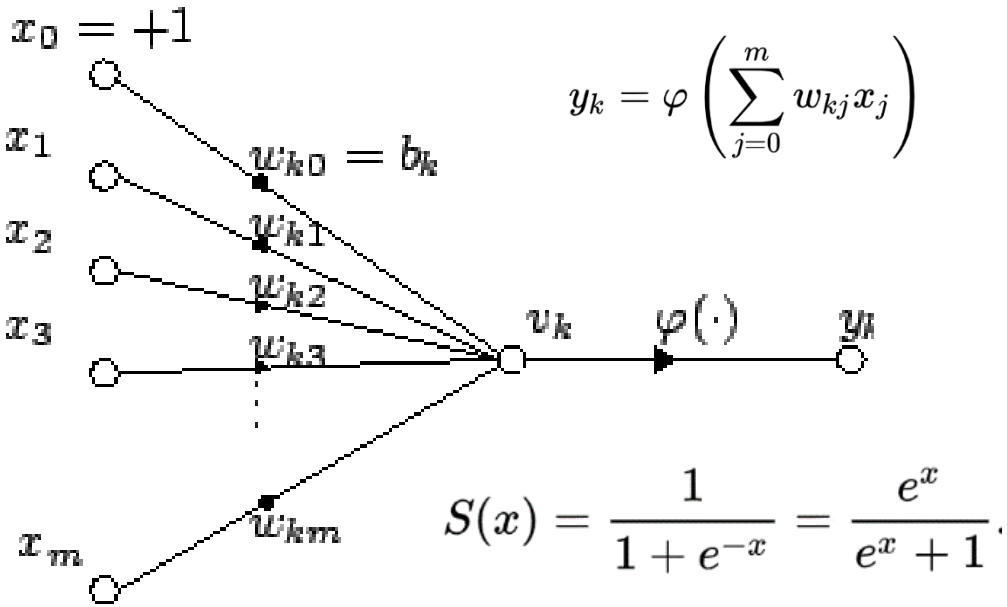
Deep Learning

- Deep learning (deep structured learning) - artificial neural networks with representation learning – supervised, semi-supervised or unsupervised.
- Deep-learning architectures such as deep neural networks, deep belief networks, recurrent neural networks (RNN) and convolutional neural networks (CNN) have been applied to fields including computer vision, machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and game programing, where they have produced results comparable to and in some cases surpassing human expert performance.
- Artificial neural networks (ANNs) were inspired by information processing and distributed communication nodes in biological systems. ANNs have various differences from biological brains. Specifically, neural networks tend to be static and symbolic, while the biological brain of most living organisms is dynamic (plastic) and analog.

Artificial Neural Networks (ANNs)

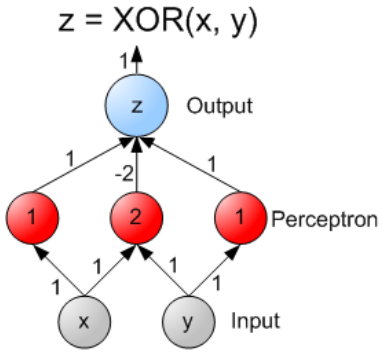
- ANN is based on a collection of connected units or nodes called artificial neurons, which model the neurons in a biological brain.
- Synapses can transmit a signal to other neurons, "signal" is a real number, output is computed by some non-linear function of the sum of its inputs -> threshold.
- The connections are called edges - typically have weight that adjusts as learning proceeds.
- Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

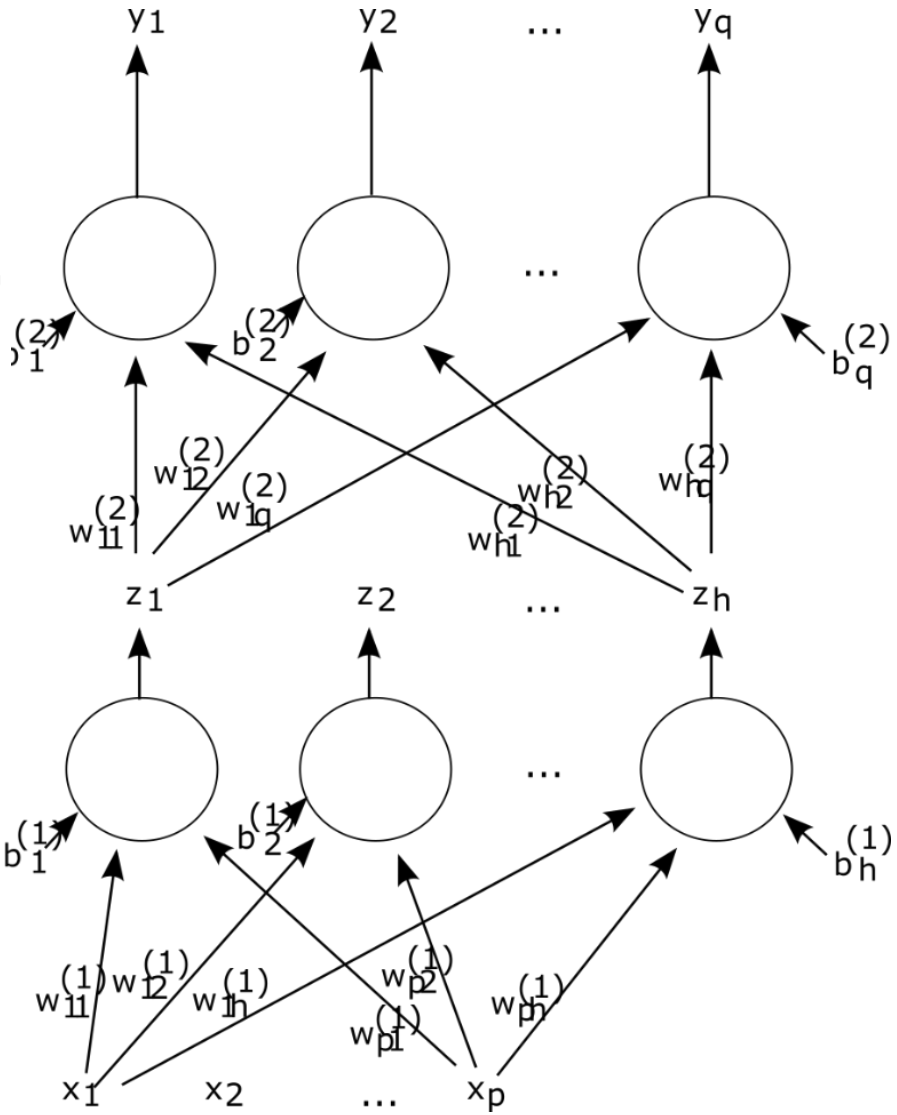


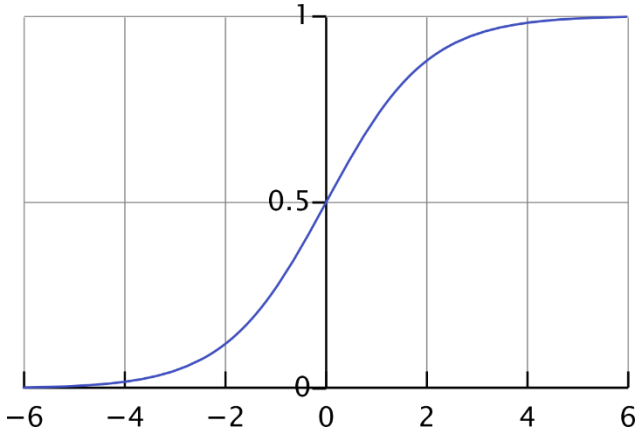


Feed Forward Networks (FFN)

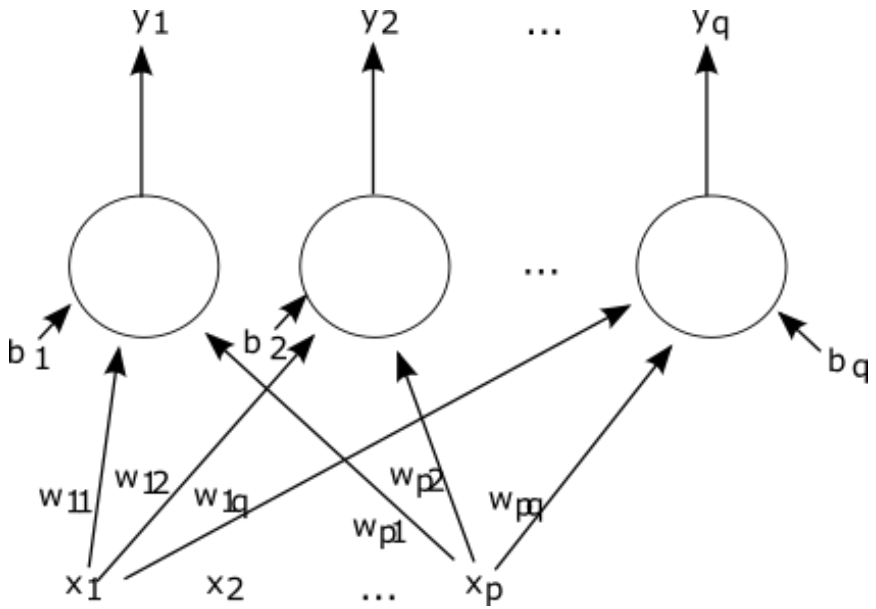
- FFN - artificial neural network wherein connections between the nodes do *not* form a cycle. As such, it is different from its descendant: recurrent neural networks.







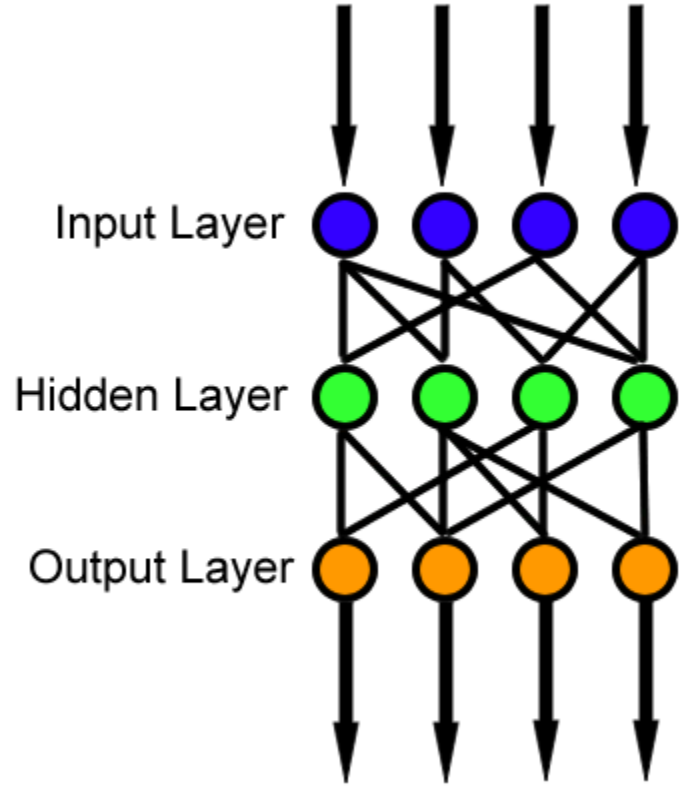
$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

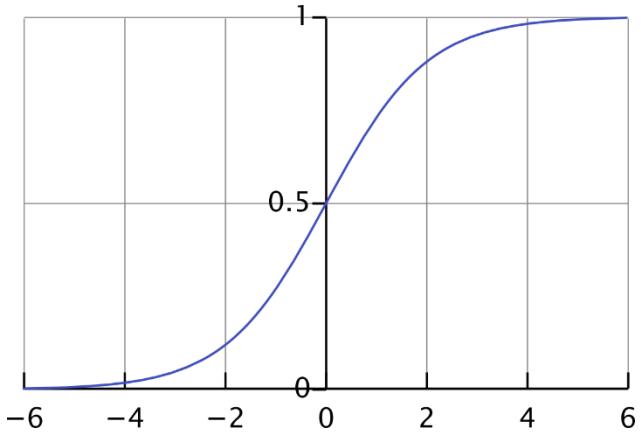


$$y_q = K*(\sum(x_i * w_{iq}) - b_q)$$

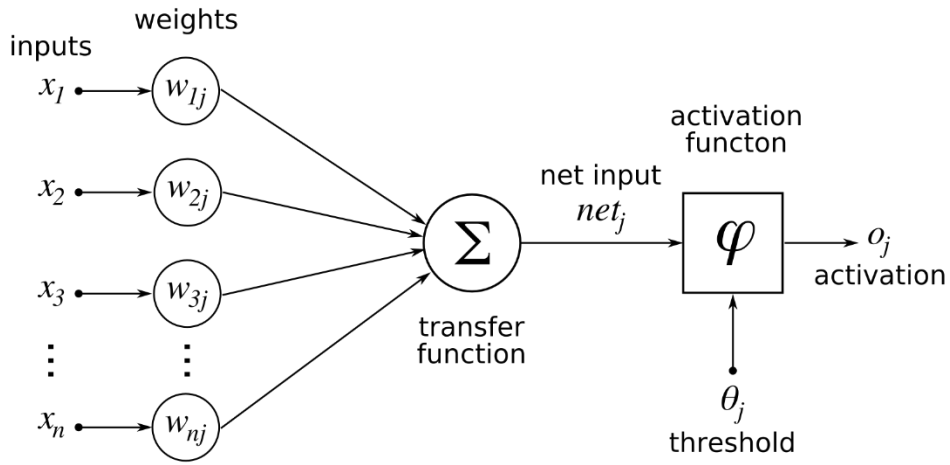
ANNs, Multilayer Perceptron (MLP)

- MLP - class of feedforward artificial neural network (ANN), three layers: input layer, hidden layer, and output layer, nonlinear activation function, supervised learning technique called backpropagation for training.





$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}.$$



Error Backprpagation

Learning occurs in the perceptron by changing connection weights after each piece of data is processed, based on the amount of error in the output compared to the expected resu
This is an example of supervised learning, and is carried out through backpropagation, a generalization of the least mean squares algorithm in the linear perceptron.

We can represent the degree of error in an output node j in the nth data point (training example) by $e_j(n) = d_j(n) - y_j(n)$, where d is the target value and y is the value produced by the perceptron. The node weights can then be adjusted based on corrections that minimize the error in the entire output, given by

$$\mathcal{E}(n) = \frac{1}{2}\sum_j e_j^2(n).$$

Using gradient descent, the change in each weight is

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} y_i(n)$$

where y_i is the output of the previous neuron and η is the *learning rate*, which is selected to ensure that the weights quickly converge to a response, without oscillations.

The derivative to be calculated depends on the induced local field v_j, which itself varies. It is easy to prove that for an output node this derivative can be simplified to

$$-\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = e_j(n)\phi'(v_j(n))$$

where ϕ' is the derivative of the activation function described above, which itself does not vary. The analysis is more difficult for the change in weights to a hidden node, but it can b shown that the relevant derivative is

$$-\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = \phi'(v_j(n))\sum_k -\frac{\partial \mathcal{E}(n)}{\partial v_k(n)} w_{kj}(n).$$

This depends on the change in weights of the kth nodes, which represent the output layer. So to change the hidden layer weights, the output layer weights change according to the derivative of the activation function, and so this algorithm represents a backpropagation of the activation function.[5]

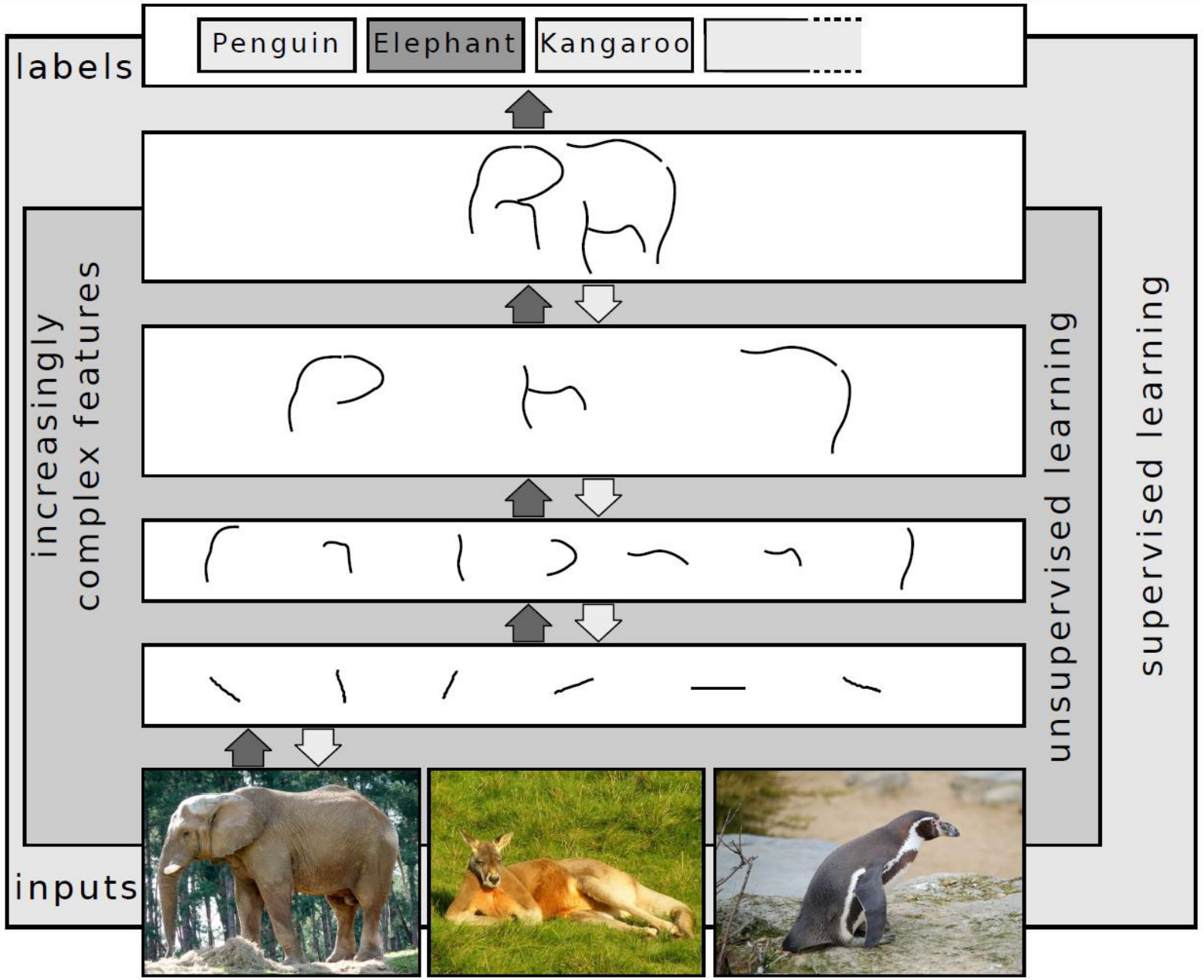
Source: https://en.wikipedia.org/wiki/Multilayer_perceptron

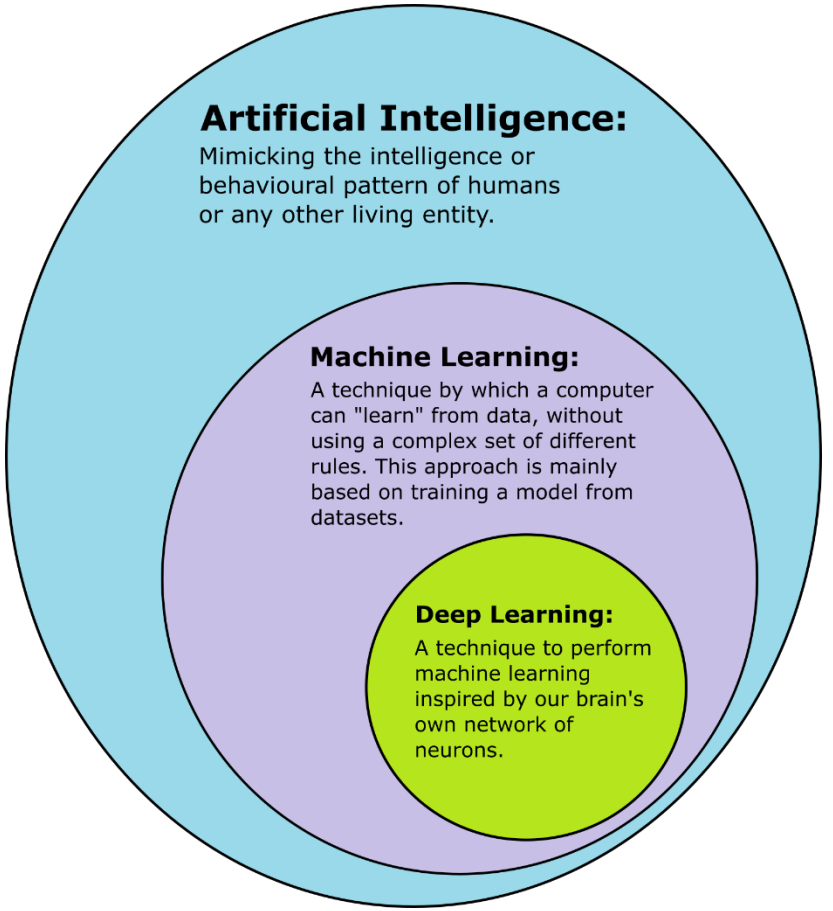
Deep Learning

- The adjective "deep" in deep learning comes from the use of multiple layers in the network. Early work showed that a linear perceptron cannot be a universal classifier, and that a network with a nonpolynomial activation function with one hidden layer of unbounded width can be so.
- Deep learning – unbounded number of layers of bounded size, which permits practical application and optimized implementation, while retaining theoretical universality under mild conditions.
- Layers can be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability, whence the "structured" part.

Convolutional Neural Networks (CNN)

- Convolutional Neural Network (CNN, or ConvNet) - a class of deep neural networks, most commonly applied to analyzing visual imagery, also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics, have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.
- CNNs are regularized versions of multilayer perceptrons – MPLs usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer -> overfitting, typical ways of regularization include adding measurement of weights to the loss function.
- CNNs take a different approach towards regularization – they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns -> lower connectedness and complexity.



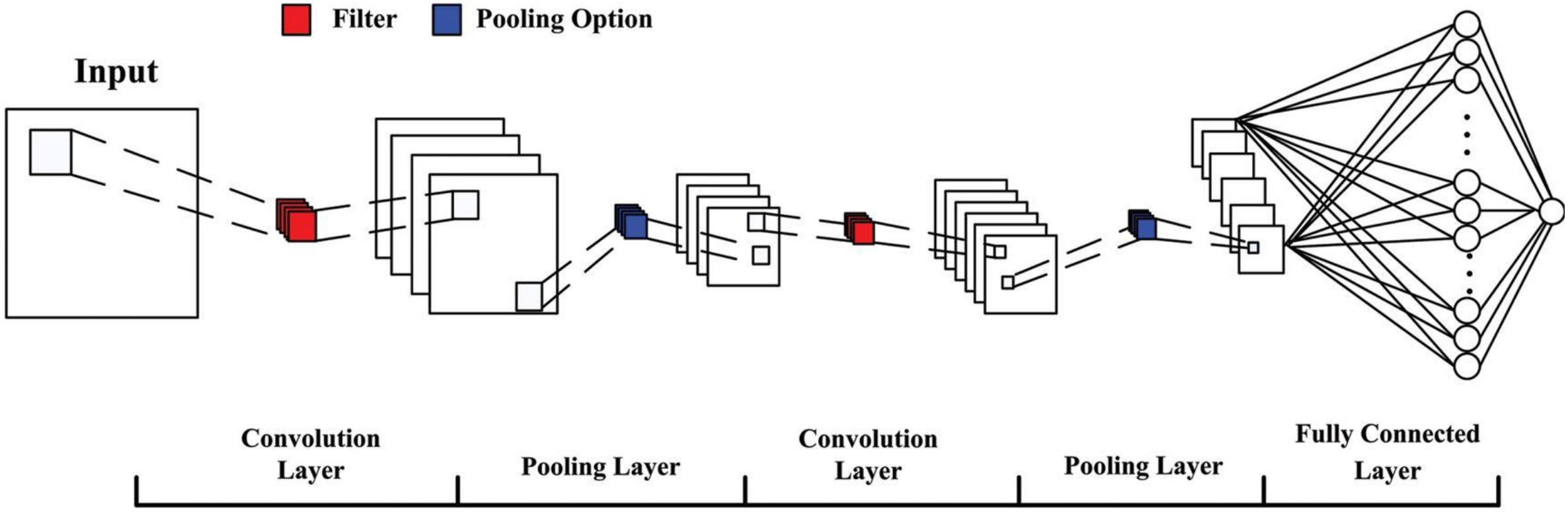


Convolutional Neural Networks (CNN)

- Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

- CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

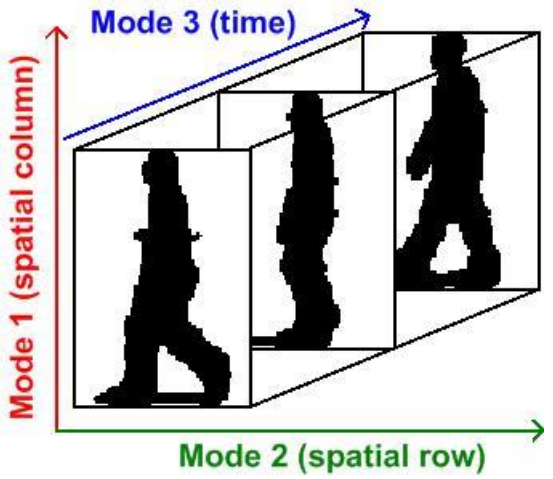
Convolutional Neural Networks (CNN)



Picture by: https://wiki.seg.org/wiki/Machine_learning

Tensors

- **Tensor** – algebraic object that describes a (multilinear) relationship between sets of algebraic objects related to a vector space. Objects that tensors may map between include vectors and scalars, and even other tensors. Tensors can take several different forms – for example: scalars and vectors (which are the simplest tensors), dual vectors, multilinear maps between vector spaces, and even some operations such as the dot product. Tensors are defined independent of any basis, although they are often referred to by their components in a basis related to a particular coordinate system.



  - Examples:
    - Vector data – 2D tensors of shape (samples, features)
    - Timeseries data or sequence data— 3D tensors of shape (samples, timesteps, features)
    - Images – 4D tensors of shape (samples, height, width, channels) or (samples, channels, height, width)
    - Video - 5D tensors of shape (samples, frames, height, width, channels) or (samples, frames, channels, height, width)

Source: https://en.wikipedia.org/wiki/Tensor
Image by WikiMSL - Make it in Microsoft Visio myself, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=23571011

Tensor Operations

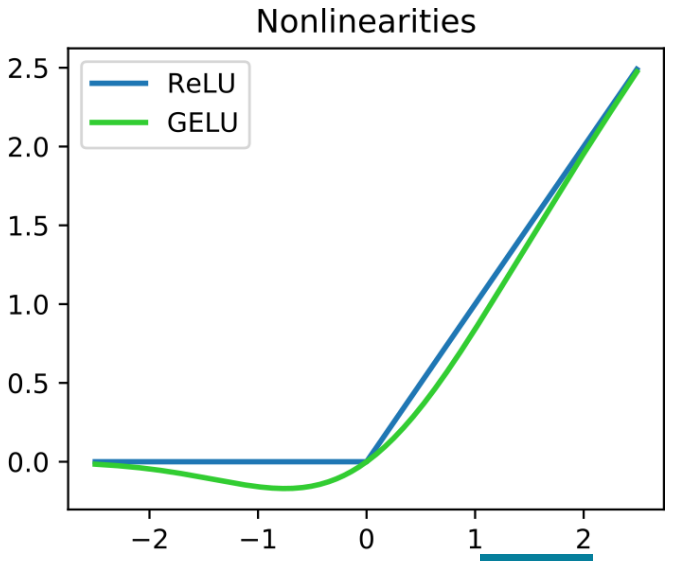
All transformations learned by deep neural networks can be reduced to a number of tensor operations applied to tensors of numeric data. For instance, it's possible to add tensors, multiply tensors, etc. For example:

keras.layers.Dense(512, activation='relu')

can be interpreted as a function, which takes as input a 2D tensor and returns another 2D tensor – as a function of the input tensor. If **W** is a 2D tensor and **b** is a vector (both are attributes of the layer):

output = relu(dot(W, input) + b) – there are 3 tensor operations here:

1) dot product (**dot**) between the input tensor and **W** tensor;
2) an addition (**+**) between the resulting 2D tensor and vector **b**;
3) a **ReLU** operation: **relu(x)** = **max(x, 0)**



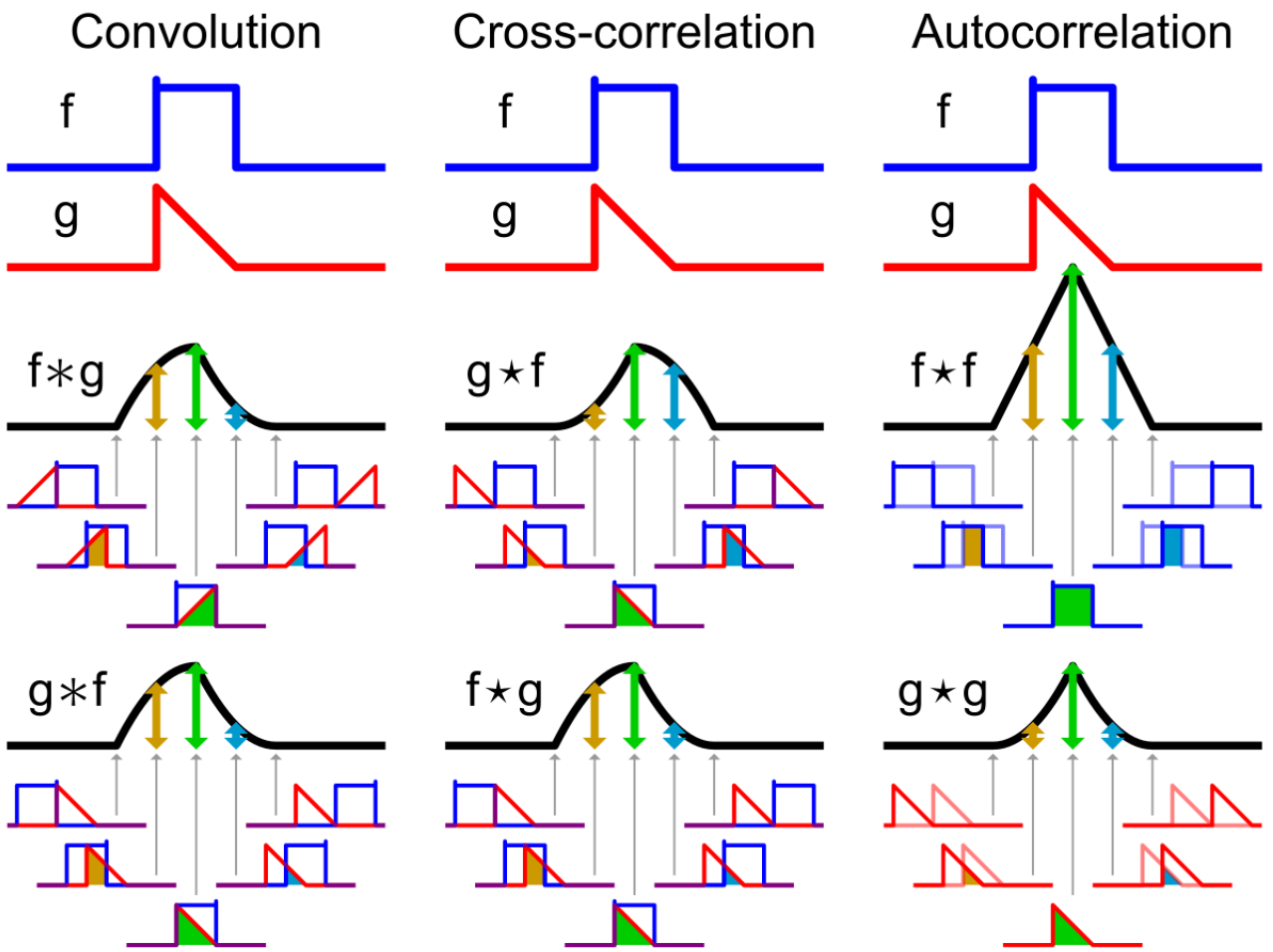
Sources: https://en.wikipedia.org/wiki/Tensor, https://en.wikipedia.org/wiki/Activation_function
Image by Ringdongdang - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=95947821

Convolution

$$f(t) * g(t) := \underbrace{\int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau}_{(f*g)(t)}$$

$$f(t) * g(t) := \underbrace{\int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau}_{(f*g)(t)}$$

- Convolution – a mathematical operation on two functions (**f** and **g**) that produces a third function **f*g** that expresses how the shape of one is modified by the other.
- The term convolution refers to both the result function and to the process of computing it. It is defined as the integral of the product of the two functions after one is reversed and shifted.
- The integral is evaluated for all values of shift, producing the convolution function.



Source: https://en.wikipedia.org/wiki/Convolution
Image by Cmglee - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=20206883

Multidimensional Discrete Convolution

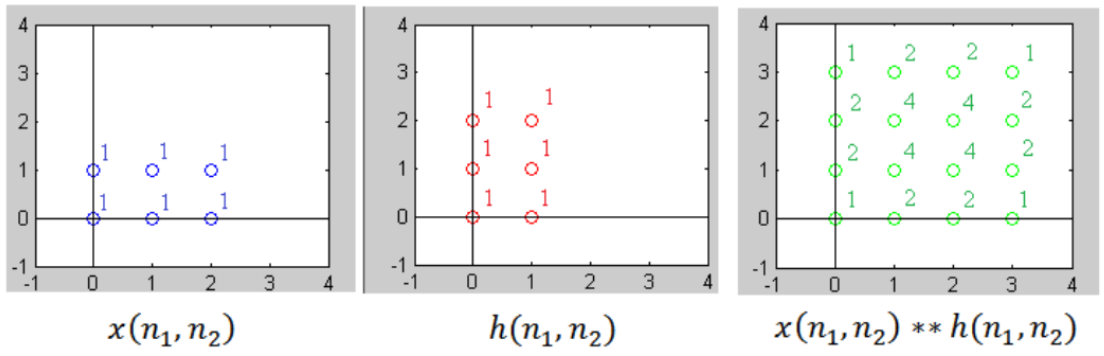
Similar to the one-dimensional case, an asterisk is used to represent the convolution operation. The number of dimensions in the given operation is reflected in the number of asterisks. For example, an *M*-dimensional convolution would be written with *M* asterisks. The following represents a *M*-dimensional convolution of discrete signals:

$$y(n_1, n_2, \ldots, n_M) = x(n_1, n_2, \ldots, n_M) *^{\cdots M \cdots} h(n_1, n_2, \ldots, n_M)$$

For discrete-valued signals, this convolution can be directly computed via the following:

$$\sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \cdots \sum_{k_M=-\infty}^{\infty} h(k_1, k_2, \ldots, k_M) x(n_1 - k_1, n_2 - k_2, \ldots, n_M - k_M)$$

The resulting output region of support of a discrete multidimensional convolution will be determined based on the size and regions of support of the two input signals.



Visualization of Convolution between two Simple Two-Dimensional Signals

Explained by example:
https://www.allaboutcircuits.com/technical-articles/two-dimensional-convolution-in-image-processing/

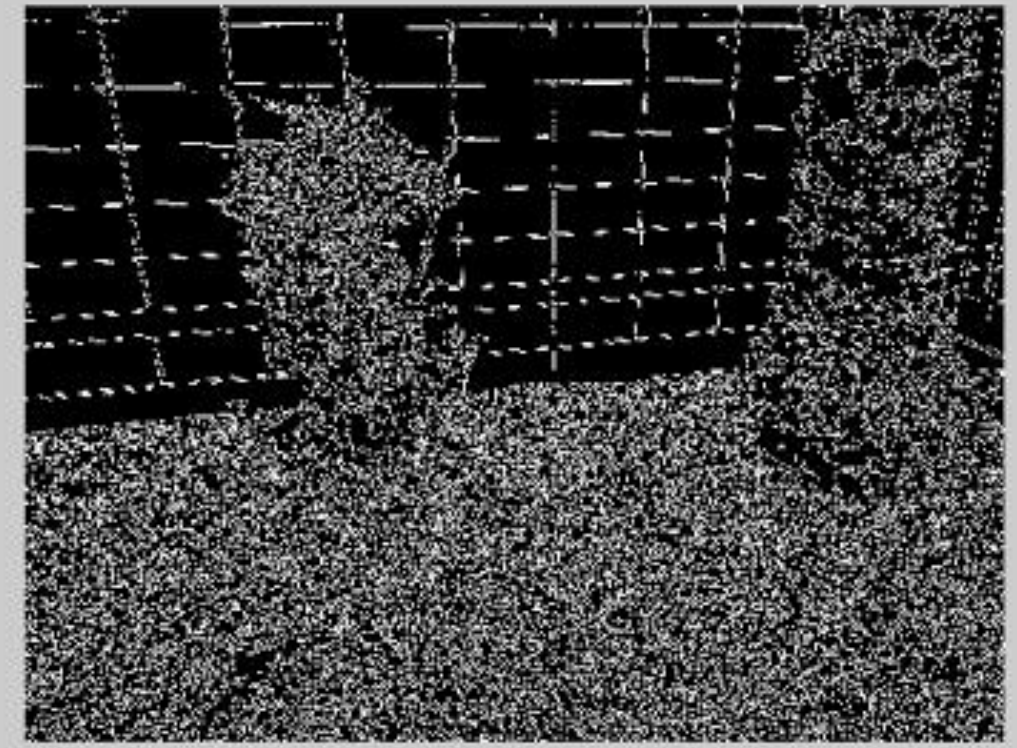
Listed are several properties of the two-dimensional convolution operator. Note that these can also be extended for signals of N-dimensions.

Source: https://en.wikipedia.org/wiki/Multidimensional_discrete_convolution
Image by Shaddowffax - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44909356

2D Convolution Example – Edge Detection

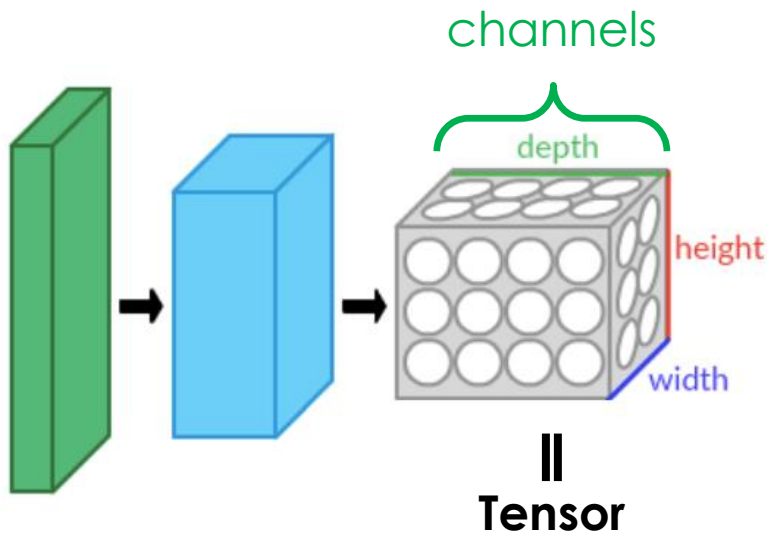
0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

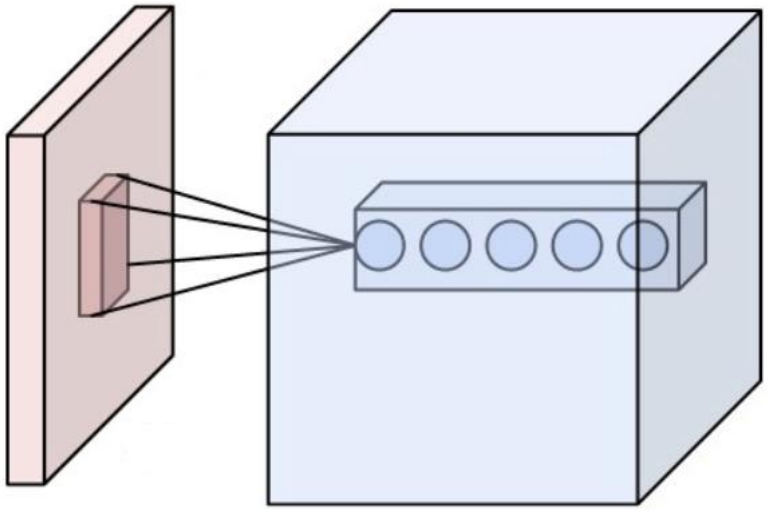


CNNs Specific Features - I

- **3D volumes of neurons** - the layers of a CNN have neurons arranged in 3 dimensions: width, height and depth where each neuron inside a convolutional layer is connected to only a small region of the layer before it, called a receptive field. Distinct types of layers, both locally and completely connected, are stacked to form a CNN architecture.



- **Local connectivity** - CNNs exploit spatial locality by enforcing a local connectivity pattern between neurons of adjacent layers. The architecture thus ensures that the learned "filters" produce the strongest response to a spatially local input pattern. Stacking many such layers leads to non-linear filters that become increasingly global (i.e. responsive to a larger region of pixel space) so that the network first creates representations of small parts of the input, then from them assembles representations of larger areas.



Source: https://en.wikipedia.org/wiki/Convolutional_neural_network, By Aphex34 - CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45661858
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

CNNs Specific Features - II

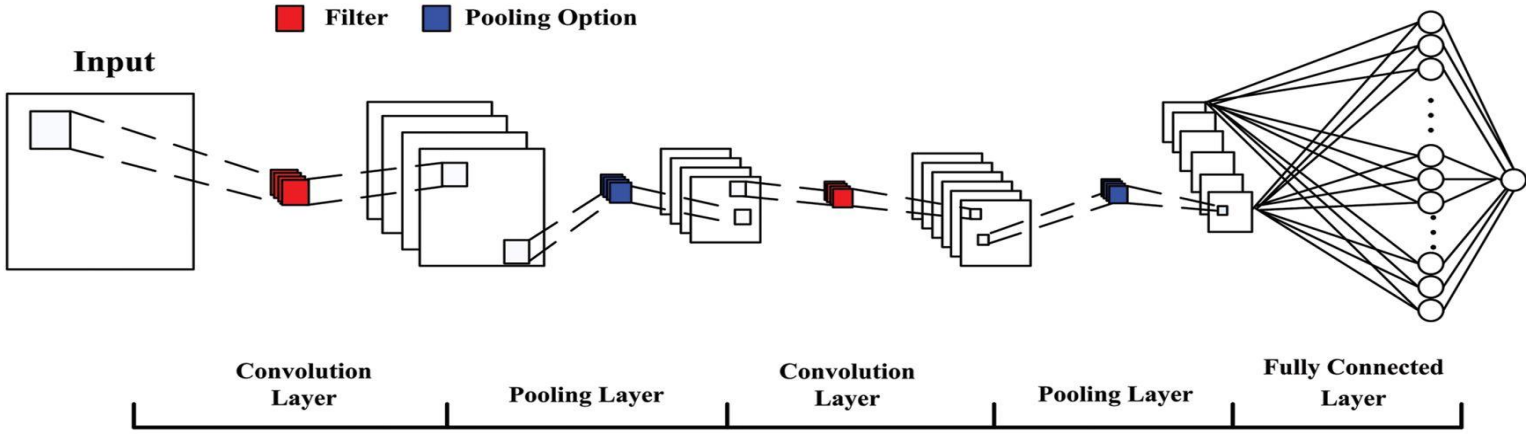
- **Shared weights** – in CNNs, each filter is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a feature map. This means that all the neurons in a given convolutional layer respond to the same feature within their specific response field. Replicating units allows for the feature map to be equivariant under changes in the locations of input features in the visual field -> translational equivariance.

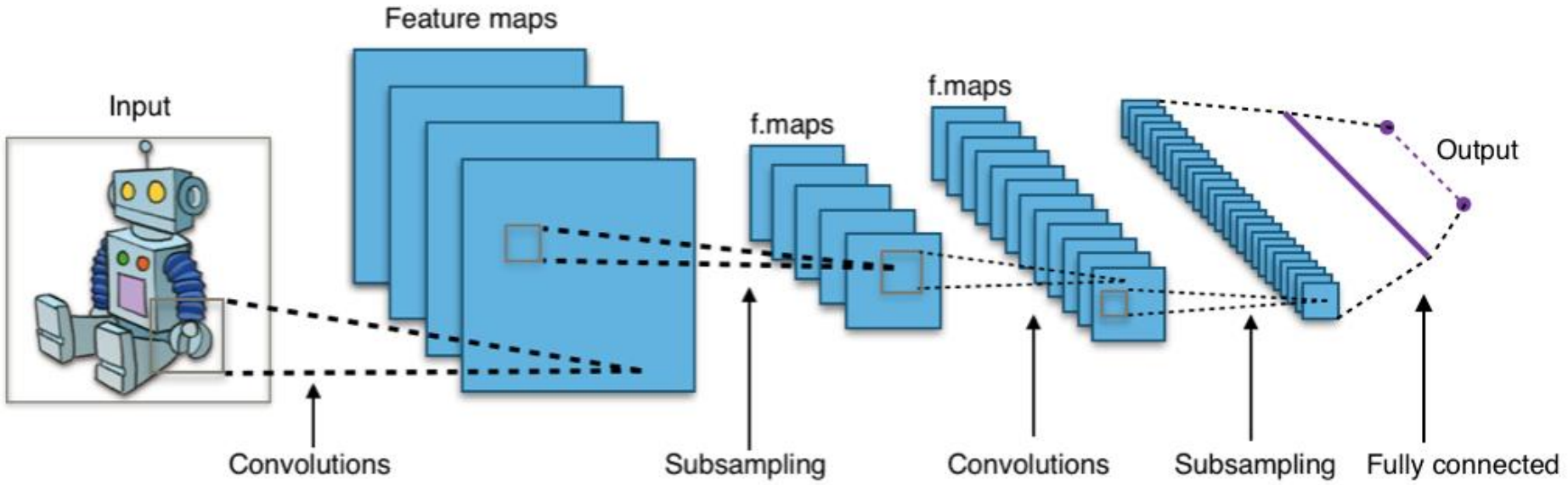
- **Pooling** – in a CNN's pooling layers, feature maps are divided into rectangular sub-regions, and the features in each rectangle are independently down-sampled to a single value, commonly by taking their average or maximum value. In addition to reducing the sizes of feature maps, the pooling operation grants a degree of translational invariance to the features contained therein, allowing the CNN to be more robust to variations in their positions.

- Together, these properties allow CNNs to achieve better generalization on vision problems. Weight sharing dramatically reduces the number of free parameters learned, thus lowering the memory requirements for running the network and allowing the training of larger, more powerful networks.

Source: https://en.wikipedia.org/wiki/Convolutional_neural_network

Convolutional Neural Networks (CNN)

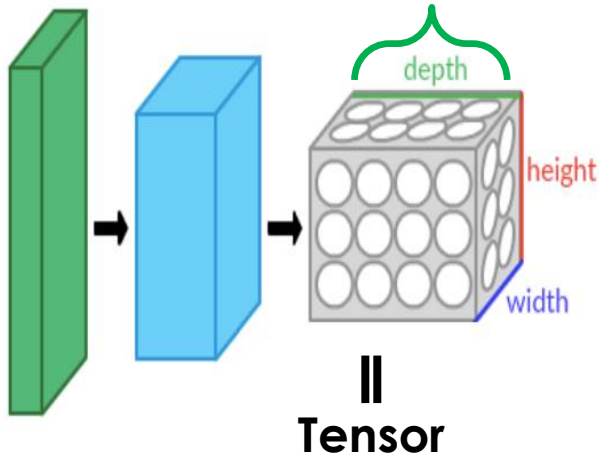


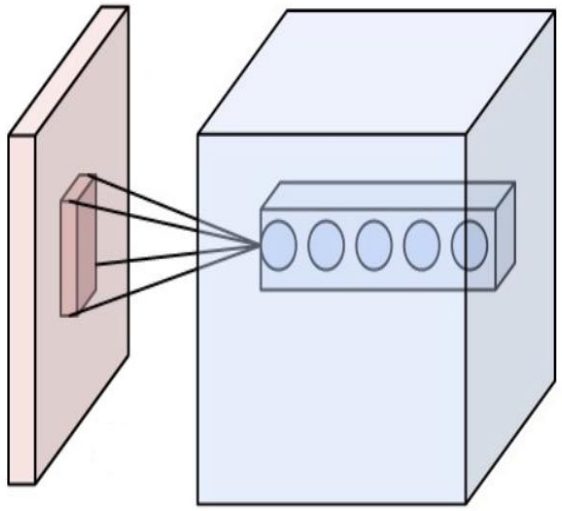


Picture by: https://wiki.seg.org/wiki/Machine_learning
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

CNN – Types of Layers – Convolutional layer

- Local connectivity - the extent of this connectivity is a hyperparameter called the receptive field of the neuron - connections are local in space (along width and height), but always extend along the entire depth of the input volume.
- Spatial arrangement – depth, stride and zero-padding – depth of the output volume controls the number of neurons in a layer that connect to the same region of the input volume. These neurons learn to activate for different features in the input (e.g. various oriented edges, or blobs of color); 2) stride – 1 meaning we move the filters one pixel at a time; 3) padding input with zeros on the border of the input volume.
- Parameter sharing - used in convolutional layers to control the number of free parameters. It relies on the assumption that if a patch feature is useful to compute at some spatial position, then it should also be useful to compute at other positions., it is common to refer to the sets of weights as a filter (or a kernel), which is convolved with the input. -> translation invariance of CNN.

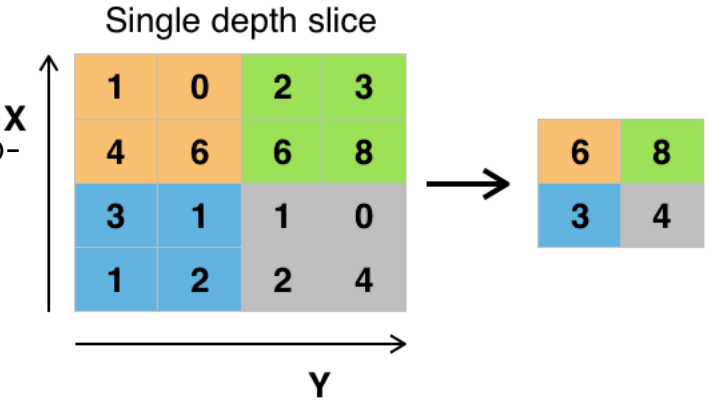




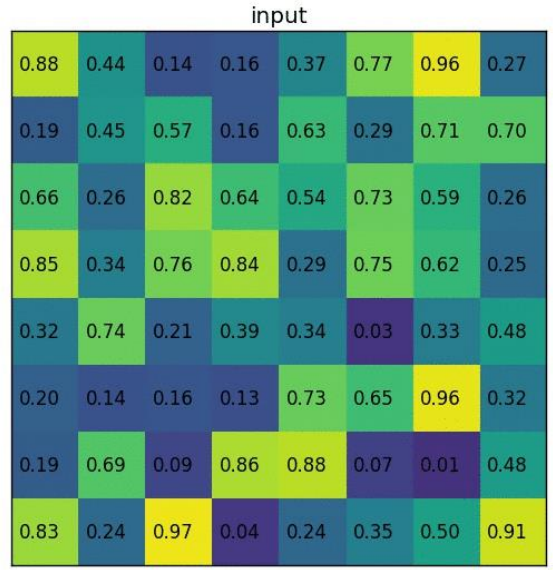
Source: https://en.wikipedia.org/wiki/Convolutional_neural_network, By Aphex34 - CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45661858
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

CNN – Types of Layers - Pooling layer

- Pooling = non-linear down-sampling. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum - location of a feature is less important than its rough location relative to other features.



- The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters, memory footprint and amount of computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers (each one typically followed by a ReLU layer) in a CNN architecture.
- The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 downsamples at every depth slice in the input by 2 along both width and height, discarding 75% of the activations.
- ℓ2-norm pooling, Region of Interest (RoI) pooling - output size is fixed and input rectangle is a parameter – most often pooling pooling to size 2x2. In this example region proposal (an input parameter) has size 7x5.



Source: By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45673581
By Tomasz Grel (deepsense.io) - https://deepsense.io/region-of-interest-pooling-explained/, CC BY 4.0

CNN Example with Keras and TensorFlow (MNIST Dataset)

```
from keras import layers
from keras import models
from keras.datasets import mnist
from keras.utils import to_categorical


model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

CNN Example with Keras and TensorFlow - Summary

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
```

CNN Example with Keras – Training & Testing (with MNIST)

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test Accuracy: ${test_acc}')
print(f'Test Loss: ${test_loss}')
print('Demo finished')
```

CNN Example with Keras – Training & Testing (CPU only)

```
938/938 [==============================] - 18s 14ms/step - loss: 0.3878 - accuracy: 0.8740
Epoch 2/5
938/938 [==============================] - 13s 14ms/step - loss: 0.0502 - accuracy: 0.9852
Epoch 3/5
938/938 [==============================] - 13s 14ms/step - loss: 0.0326 - accuracy: 0.9897
Epoch 4/5
938/938 [==============================] - 13s 14ms/step - loss: 0.0248 - accuracy: 0.9928
Epoch 5/5
938/938 [==============================] - 13s 14ms/step - loss: 0.0175 - accuracy: 0.9945
313/313 [==============================] - 1s 2ms/step - loss: 0.0286 - accuracy: 0.9911
Test Accuracy: 0.991100013256073
Test Loss: 0.028569826856255 53
Demo finished
```

CNN Example with Keras – Training & Testing (using GPU)

```
938/938 [==============================] - 7s 4ms/step - loss: 0.3917 - accuracy: 0.8749
Epoch 2/5
938/938 [==============================] - 4s 4ms/step - loss: 0.0490 - accuracy: 0.9851
Epoch 3/5
938/938 [==============================] - 3s 3ms/step - loss: 0.0334 - accuracy: 0.9897
Epoch 4/5
938/938 [==============================] - 3s 3ms/step - loss: 0.0242 - accuracy: 0.9928
Epoch 5/5
938/938 [==============================] - 3s 3ms/step - loss: 0.0183 - accuracy: 0.9946
313/313 [==============================] - 1s 2ms/step - loss: 0.0213 - accuracy: 0.9930
Test Accuracy: 0.9930000305175781
Test Loss: 0.021334622055292 13
Demo finished
```

Search nodes. Regexes supported.

Fit to Screen

Download PNG

Run (4) 20210109-050204\train

Tag (3) Default

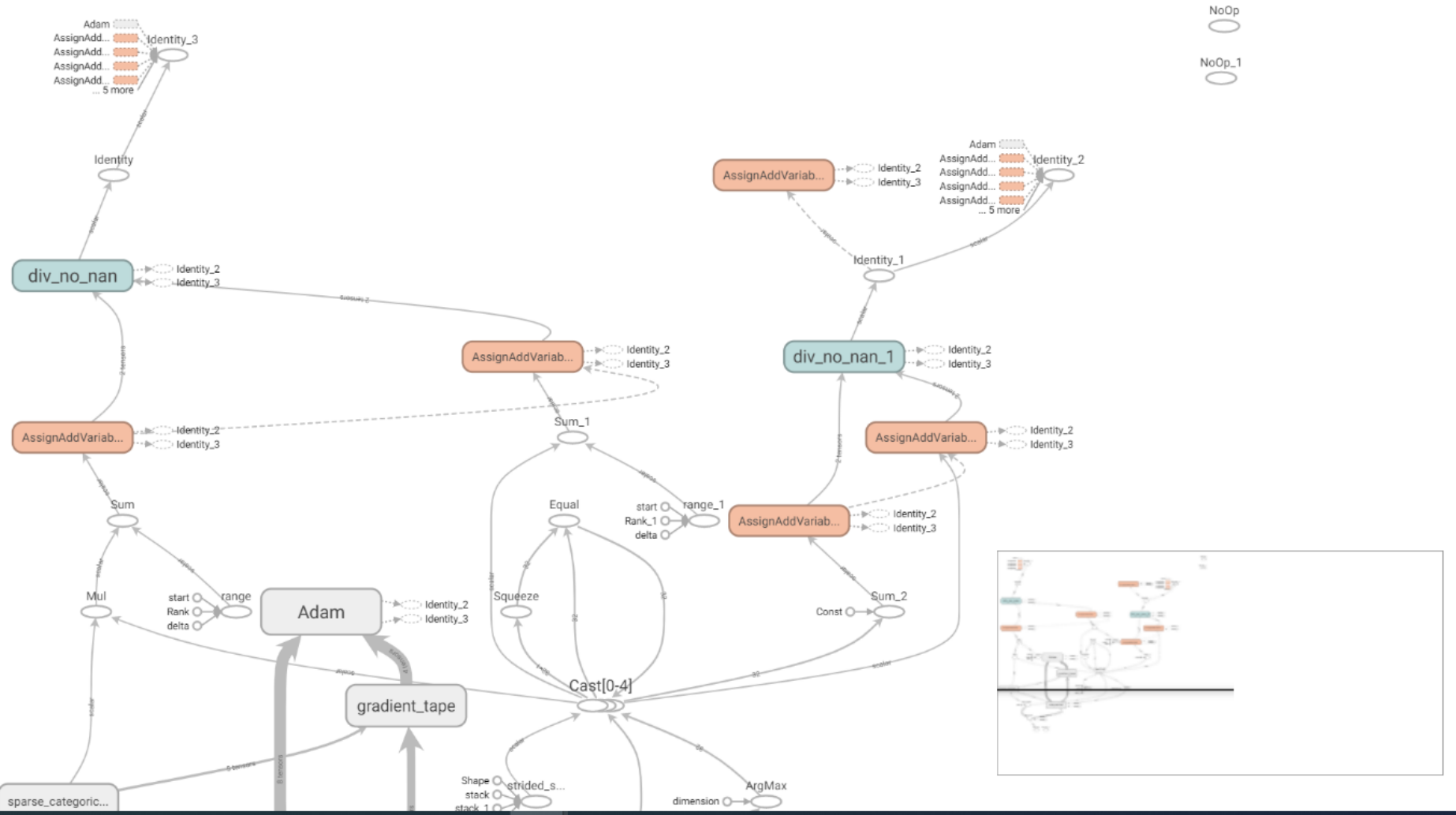
Upload Choose File

- Graph
- Conceptual Graph
- Profile

Close legend.

Graph (* = expandable)

- Namespace* ?
- OpNode ?
- Unconnected series* ?
- Connected series* ?
- Constant ?
- Summary ?
- Dataflow edge ?
- Control dependency edge ?
- Reference edge ?



Histogram mode

OVERLAY OFFSET

Offset time axis

STEP RELATIVE WALL

Runs

Write a regex to filter runs

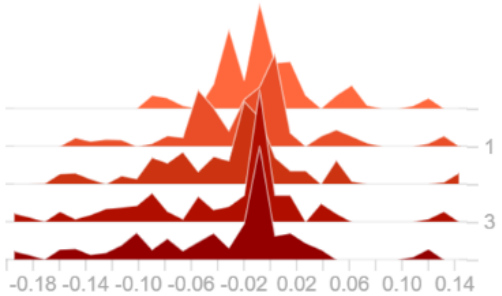
- [x] 20210109-050204\train
- [x] 20210109-050204\validation
- [x] 20210109-051457\train
- [x] 20210109-053307\train
- [x] 20210109-053933\train

TOGGLE ALL RUNS

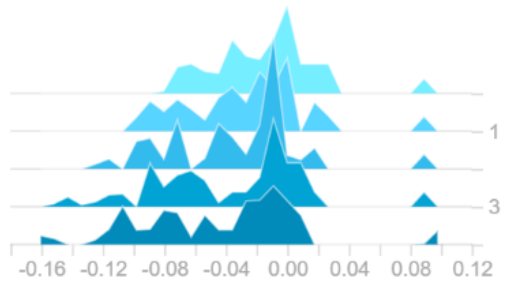
./logs

conv2d

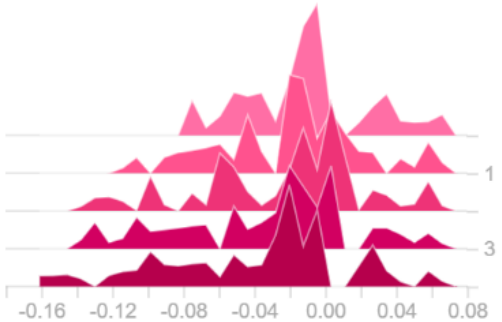
conv2d/bias_0 20210109-051457\train



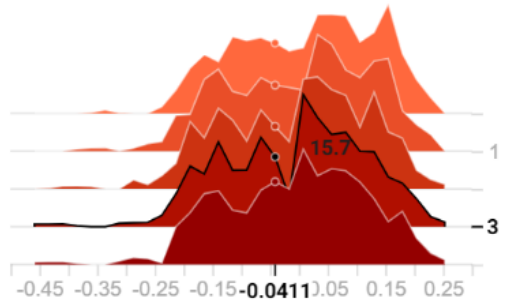
conv2d/bias_0 20210109-053307\train



conv2d/bias_0 20210109-053933\train



conv2d/kernel_0 20210109-051457\train



conv2d/kernel_0 20210109-053307\train

conv2d/kernel_0 20210109-053933\train

CAPTURE PROFILE

Runs (4)

20210109-053933\train\2021_01_...

Tools (8)

overview_page

Hosts

OFFICE27

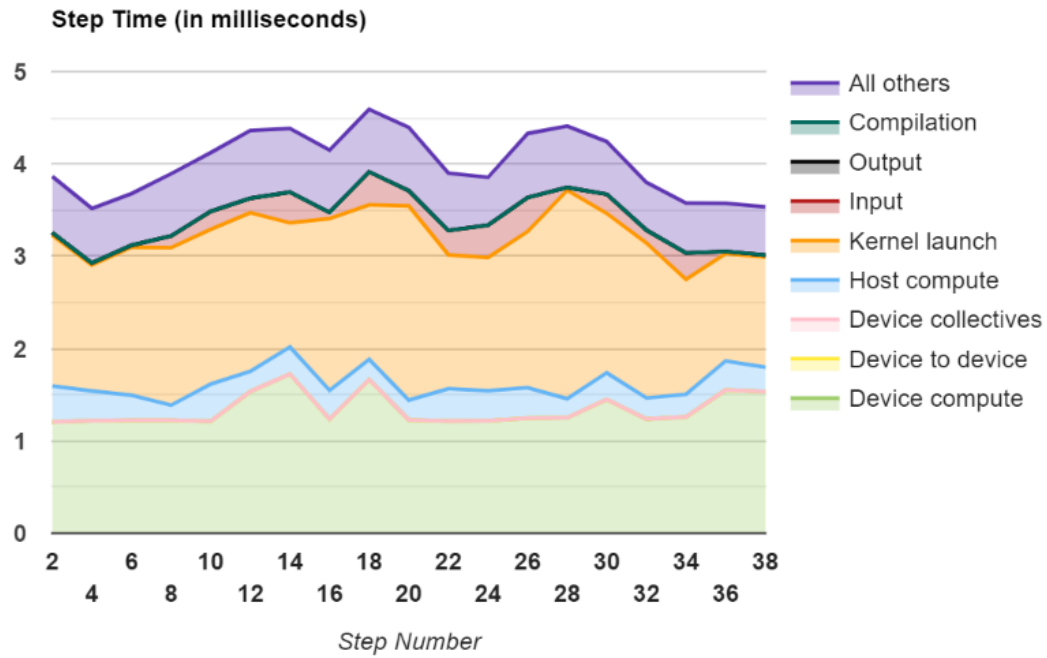
Performance Summary

Average Step Time — 4.0 ms
lower is better
(σ = 0.3 ms)

- Idle: ms
- Input: ms
- Compute: ms
- Idle: ms
- Input: ms
- Compute: ms

- All Others Time — 0.6 ms
  (σ = 0.1 ms)
- Compilation Time — 0.0 ms
  (σ = 0.0 ms)
- Output Time — 0.0 ms
  (σ = 0.0 ms)
- Input Time — 0.2 ms
  (σ = 0.1 ms)
- Kernel Launch Time — 1.6 ms
  (σ = 0.3 ms)

Step-time Graph



Recommendation for Next Step

- Your program is POTENTIALLY input-bound because 15.4% of the total step time sampled is spent on 'All Others' time (which could be due to I/O or Python execution or both).

CAPTURE PROFILE

Runs (4)

20210109-053933\train\2021_01_...

Tools (8)

tensorflow_stats

Hosts

OFFICE27

TensorFlow Stats

(1) In the charts and table below, "IDLE" represents the portion of the total execution time on device (or host) that is idle.
(2) In the pie charts, the "Other" sector represents the sum of sectors that are too small to be shown individually.

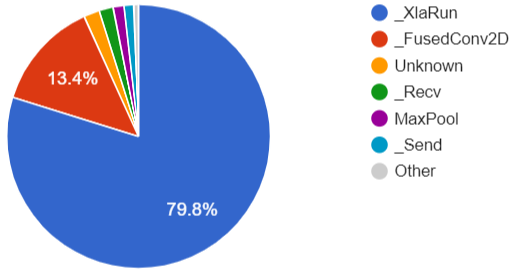
Export as CSV

Include IDLE time in statistics

No

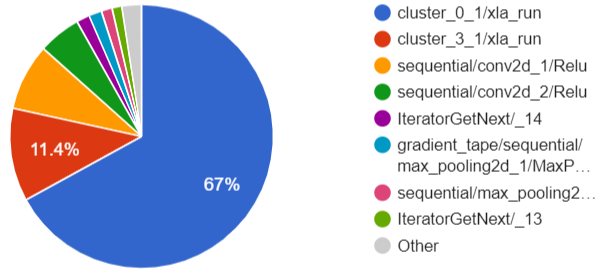
ON DEVICE: TOTAL SELF-TIME (GROUPED BY TYPE)

(in microseconds) of a TensorFlow operation type



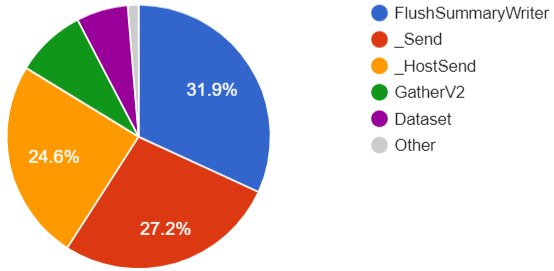
ON DEVICE: TOTAL SELF-TIME

(in microseconds) of a TensorFlow operation



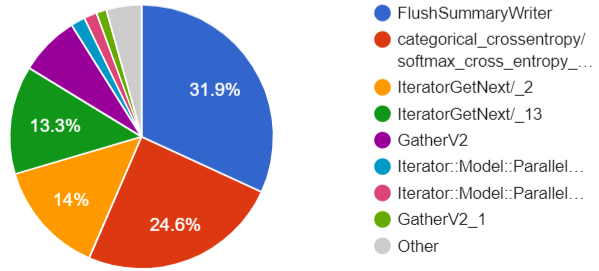
ON HOST: TOTAL SELF-TIME (GROUPED BY TYPE)

(in microseconds) of a TensorFlow operation type



ON HOST: TOTAL SELF-TIME

(in microseconds) of a TensorFlow operation



CAPTURE PROFILE

Runs (4)

20210109-053933\train\2021_01_...

Tools (8)

kernel_stats

Hosts

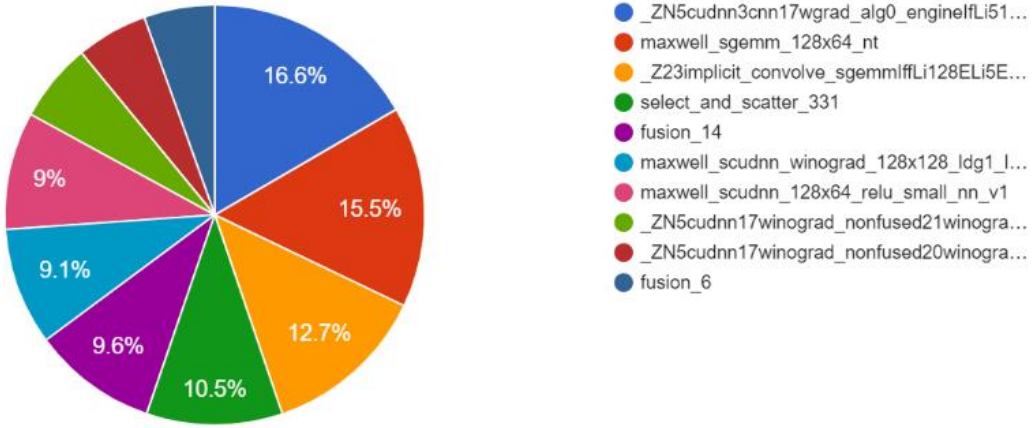
OFFICE27

GPU Kernel Stats

Export as CSV

Top 10 Kernels with highest Total Duration

Show top 10 Kernels



GPU Kernels

Kernel Name Op Name

CAPTURE PROFILE

Runs (4)

20210109-053933\train\2021_01_...

Tools (8)

overview_page

Hosts

OFFICE27

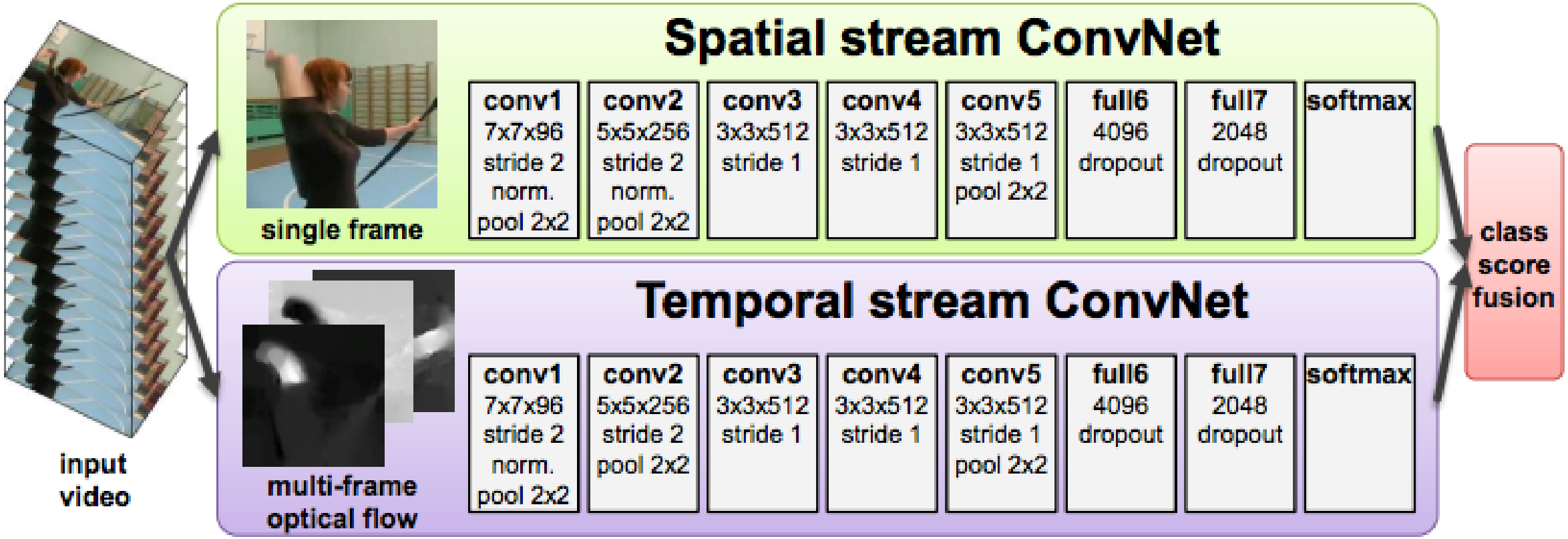
Device type: Nvidia GPU (Pascal)

Number of device cores: 1

Top 10 TensorFlow operations on GPU

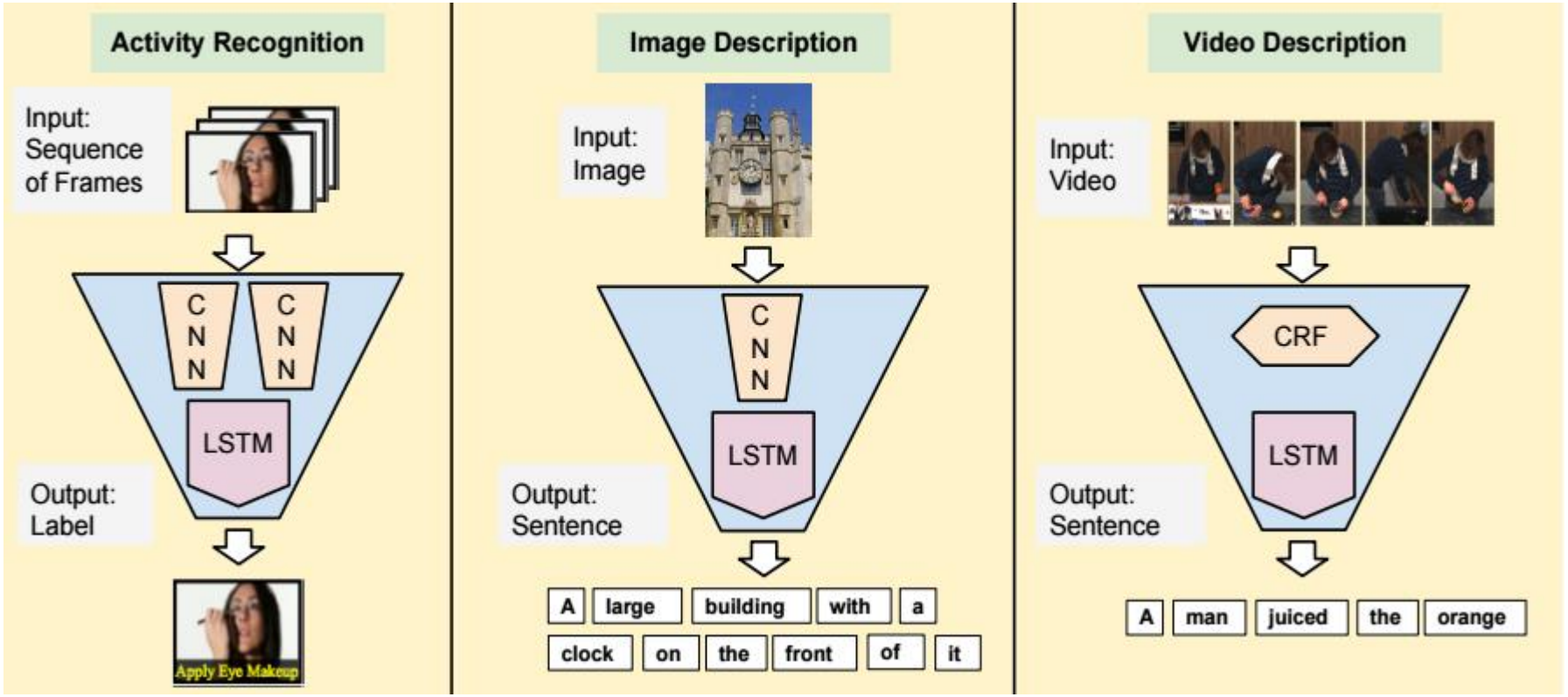
Time (%)	Cumulative time (%)	Category	Operation	TensorCore eligibility	Op is using TensorCore
67%	67%	_XlaRun	cluster_0_1/xla_run	X	X
11.4%	78.4%	_XlaRun	cluster_3_1/xla_run	X	X
8.2%	86.6%	_FusedConv2D	sequential/conv2d_1/Relu	X	X
5.2%	91.8%	_FusedConv2D	sequential/conv2d_2/Relu	X	X
1.6%	93.5%	_Recv	IteratorGetNext/_14	X	X
1.6%	95%	Unknown	gradient_tape/sequential/max_pooling2d_1/MaxPool/MaxPoolGrad-0-TransposeNHWCToNCHW-LayoutOptimizer:Transpose	X	X
1.3%	96.4%	MaxPool	sequential/max_pooling2d_1/MaxPool	X	X
1.2%	97.6%	_Send	IteratorGetNext/_13	X	X
0.9%	98.5%	_XlaRun	cluster_1_1/xla_run	X	X
0.5%	99%	_XlaRun	cluster_2_1/xla_run	X	X

Recurrent Neural Networks (RNN)



Picture by: https://wiki.tum.de/display/lfdv/Video+Analysis

RNN for Video Analysis



Picture by: https://wiki.tum.de/display/lfdv/Video+Analysis

Deep Leaning with H2O - I

- purely supervised training protocol for regression and classification tasks
- fast and memory-efficient Java implementations based on columnar compression and fine- grain Map/Reduce
- multi-threaded and distributed parallel computation to be run on either a single node or a multi-node cluster
- fully automatic per-weight adaptive learning rate for fast convergence
- optional specification of learning rate, annealing and momentum options
- regularization options include L1, L2, dropout, Hogwild! and model averaging to prevent model overfitting
- grid search for hyperparameter optimization and model selection

Deep Leaning with H2O - II

- model checkpointing for reduced run times and model tuning
- automatic data pre- and post-processing (one-hot encoding and standardization) for categorical and numerical data
- automatic imputation of missing values
- automatic tuning of communication vs computation for best performance
- model export in plain java code for deployment in production environments
- export of weights and biases as H2O frames
- additional expert parameters for model tuning
- deep autoencoders for unsupervised feature learning and anomaly detection capabilities

H2O- DeepLear | pandas.DataFra | python - Creatir | Python String sp | Using Flow - H2 | Intellij Idea 201 | Reading a CSV | OrdinalModelPr | Deep Learning | The Definitive P

localhost:54321/flow/index.html#

convert 1/2

H2O FLOW Flow Cell Data Model Score Admin Help

DeepLearning_FaceRecognition

262ms

Model

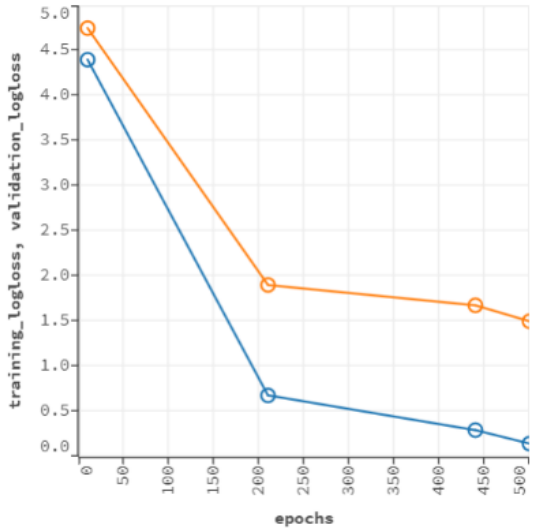
Model ID: deeplearning-699a1500-b9e4-44af-8e89-772f3b6628e3

Algorithm: Deep Learning

Actions: Refresh | Predict... | Download POJO | Download Model Deployment Package (MOJO) | Export | Inspect | Delete | Download Gen Model

▸ MODEL PARAMETERS

▾ SCORING HISTORY - LOGLOSS



▾ VARIABLE IMPORTANCES

OUTLINE FLOWS CLIPS HELP

Help

PACK

examples

- GBM_Example.flow
- DeepLearning_MNIST.flow
- GLM_Example.flow
- DRF_Example.flow
- K-Means_Example.flow
- Million_Songs.flow
- KDDCup2009_Churn.flow
- QuickStartVideos.flow
- Airlines_Delay.flow
- GBM_Airlines_Classification.flow
- GBM_GridSearch.flow
- RandomData_Benchmark_Small.flow
- GBM_TuningGuide.flow
- XGBoost_Example.flow

Ready

Connections: 0

deeplearning_699....java

Show all

Type here to search

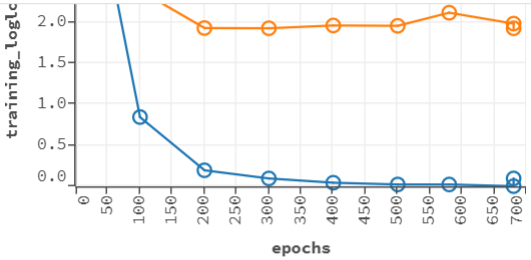
ENG 16:14 21.12.2020 г.

H2O- DeepLear | pandas.DataFra | python - Creatir | Python String sp | Using Flow - H2 | Intellij Idea 201 | Reading a CSV | OrdinalModelPr | Deep Learning | The Definitive P

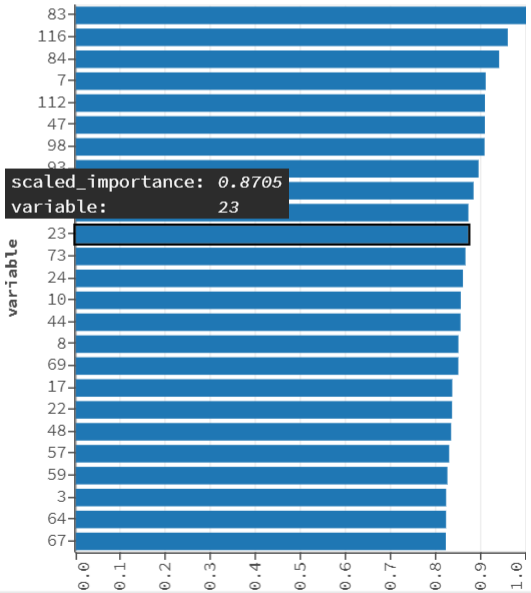
localhost:54321/flow/index.html#

H2O FLOW Flow Cell Data Model Score Admin Help

DeepLearning_FaceRecognition



VARIABLE IMPORTANCES



OUTLINE FLOWS CLIPS HELP

Help

PACK

examples

- GBM_Example.flow
- DeepLearning_MNIST.flow
- GLM_Example.flow
- DRF_Example.flow
- K-Means_Example.flow
- Million_Songs.flow
- KDDCup2009_Churn.flow
- QuickStartVideos.flow
- Airlines_Delay.flow
- GBM_Airlines_Classification.flow
- GBM_GridSearch.flow
- RandomData_Benchmark_Small.flow
- GBM_TuningGuide.flow
- XGBoost_Example.flow

Ready

Connections: 0

deeplearning_699....java | deeplearning_699....java | deeplearning_699....java | deeplearning_699....java

Show all

Type here to search

ENG 17:26 21.12.2020 г.

H2O- DeepLear | pandas.DataFra | python - Creatir | Python String sp | Using Flow - H2 | Intellij Idea 201 | Reading a CSV | OrdinalModelPr | Deep Learning | The Definitive P

localhost:54321/flow/index.html#

H2O FLOW Flow Cell Data Model Score Admin Help

DeepLearning_FaceRecognition

scaled_importance

TRAINING METRICS - CONFUSION MATRIX ROW LABELS: ACTUAL CLASS; COLUMN LABELS: PREDICTED CLASS

1	0	87	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0.0440	4 / 91	1.0
2	0	0	39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 / 39	1.0
3	0	0	0	82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 / 82	0.85
4	0	0	0	4	97	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0.0490	5 / 102	0.99
5	0	0	0	0	0	56	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 / 56	0.98
6	0	0	0	0	0	0	92	0	0	0	0	0	0	0	0	0	6	3	0	0	0.0891	9 / 101	1.0
7	0	0	0	0	1	0	0	34	0	0	0	0	0	0	0	0	0	0	0	0	0.0286	1 / 35	1.0
8	1	0	0	0	0	0	0	0	75	0	0	3	0	0	0	0	0	0	0	0	0.0506	4 / 79	0.99
9	1	0	0	0	0	0	0	0	0	100	0	1	0	0	0	0	0	0	0	0	0.0196	2 / 102	0.91
10	0	0	0	0	0	0	0	0	0	7	72	1	0	0	0	0	0	0	0	0	0.1000	8 / 80	1.0
11	0	0	0	0	0	0	0	0	0	0	0	101	0	0	0	0	0	0	0	0	0	0 / 101	0.89
12	1	0	0	2	0	0	0	0	0	0	0	5	87	0	0	0	0	0	0	0	0.0842	8 / 95	1.0
13	1	0	0	6	0	0	0	0	1	1	0	2	0	68	0	0	0	0	0	0	0.1392	11 / 79	1.0
14	1	0	0	0	0	0	0	0	0	0	0	0	0	0	36	0	0	0	0	0	0.0270	1 / 37	1.0
15	1	0	0	0	0	1	0	0	0	2	0	0	0	0	0	39	1	1	0	0	0.1333	6 / 45	1.0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	103	0	0	0	0	0 / 103	0.92
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	79	0	0	0.0125	1 / 80	0.84
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	7	88	0	0.0833	8 / 96	1.0
19	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	84	0.0233	2 / 86	1.0
Total	100	87	39	96	98	57	92	34	76	110	72	113	87	68	36	39	112	94	88	84	0.0442	70 / 1Â 582	
Recall	1.0	0.96	1.0	1.0	0.95	1.0	0.91	0.97	0.95	0.98	0.90	1.0	0.92	0.86	0.97	0.87	1.0	0.99	0.92	0.98			

OUTLINE FLOWS CLIPS HELP

Help

PACK

examples

- GBM_Example.flow
- DeepLearning_MNIST.flow
- GLM_Example.flow
- DRF_Example.flow
- K-Means_Example.flow
- Million_Songs.flow
- KDDCup2009_Churn.flow
- QuickStartVideos.flow
- Airlines_Delay.flow
- GBM_Airlines_Classification.flow
- GBM_GridSearch.flow
- RandomData_Benchmark_Small.flow
- GBM_TuningGuide.flow
- XGBoost_Example.flow

Ready

Connections: 0

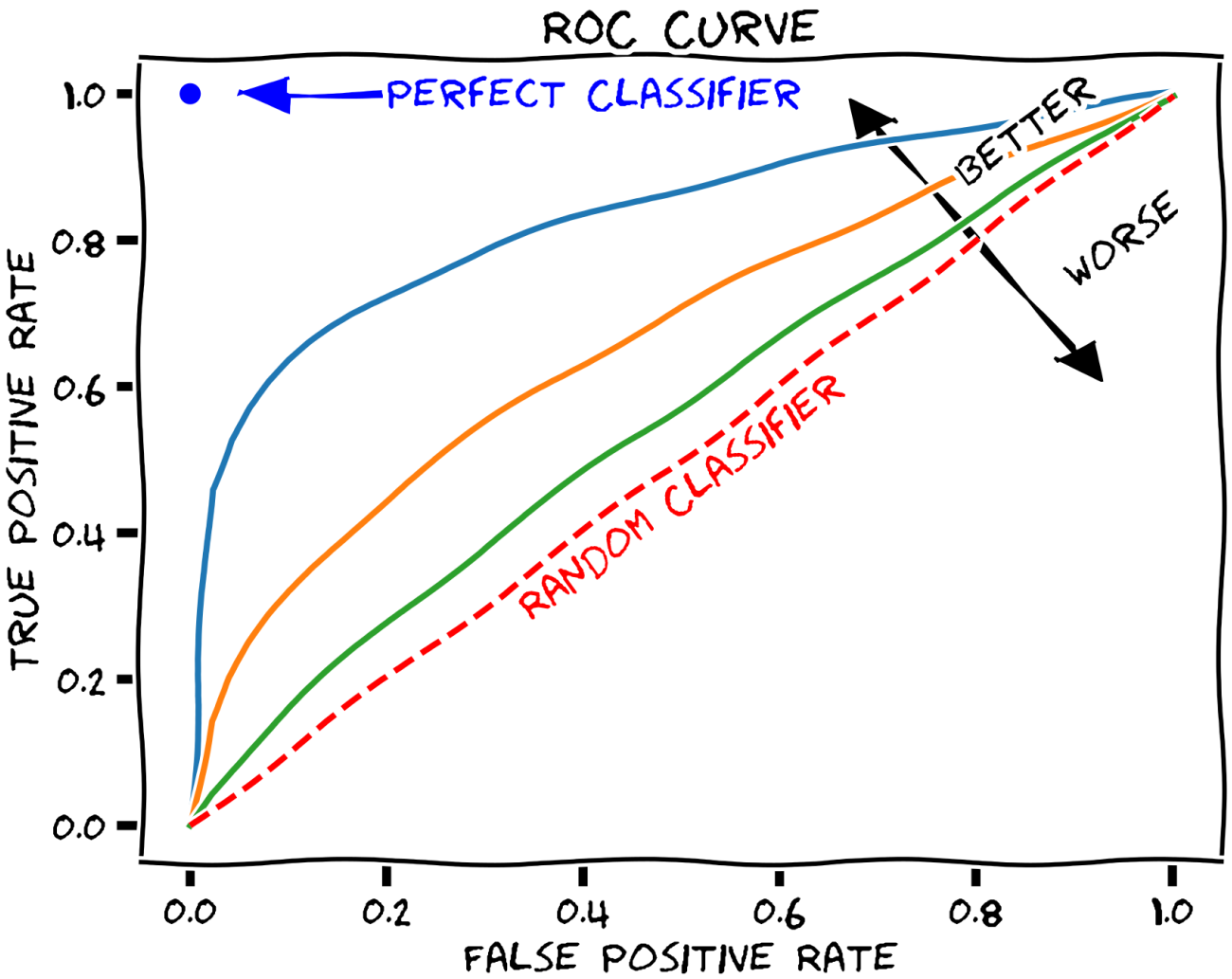
deeplearning_699....java

Show all

Type here to search

ENG 16:17 21.12.2020 г.

Receiver Operating Characteristic (ROC)



Receiver operating characteristic (ROC) curve - graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

Receiver Operating Characteristic (ROC)

- condition positive (P) / negative (N) - the number of real positive / negative cases in the data
- true positive (TP) - eqv. with hit, true negative (TN) - eqv. with correct rejection
- false positive (FP) - eqv. with false alarm, Type I error, false negative (FN) - eqv. with miss, Type II error

		True condition				
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$		

Distributed Learning: Ensemble Learning

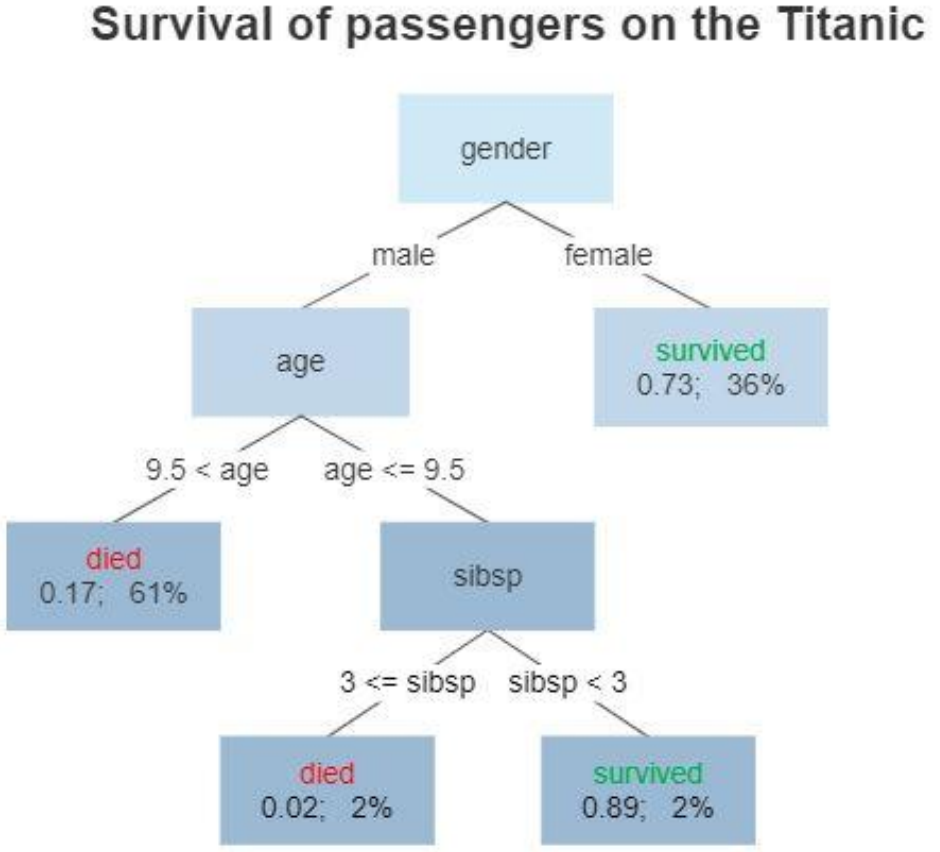
Ensemble Learning

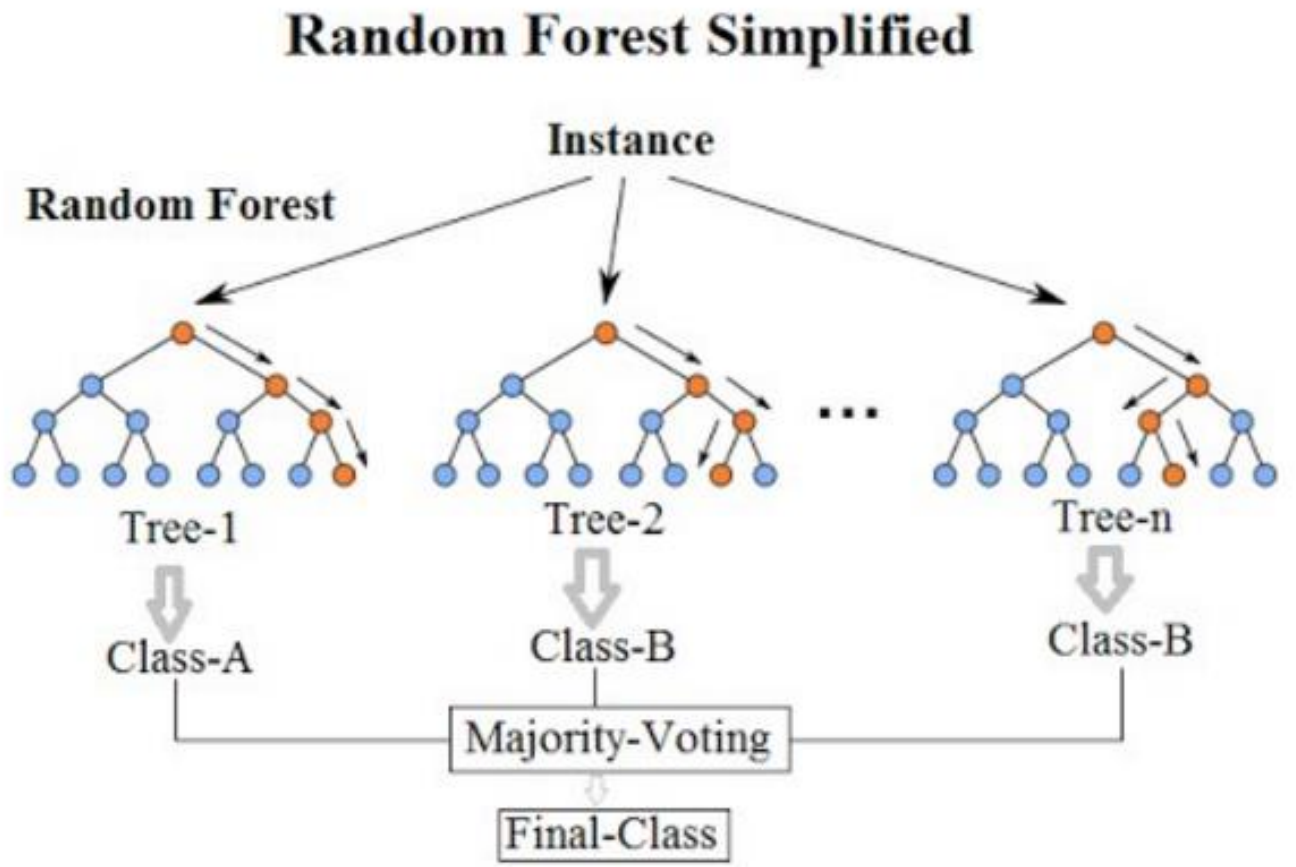
- Ensemble methods – using multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. The term ensemble usually means using the same type of base learners.
- Even if the hypothesis space contains hypotheses that are very well-suited for a particular problem, it may be very difficult to find a good one. Ensembles combine multiple hypotheses to form a better hypothesis. Resultant hypothesis, is not necessarily contained within the hypothesis space of the building models – can be more complex.
- Ensemble over-fits the training data easier than a single model -> Bagging.
- Ensembles yield better results when there is a diversity among the models. Using variety of strong learning algorithms, has been shown to be more effective than using techniques that attempt to dumb-down the models in order to promote diversity.
- number of independent classifiers == number of classes => highest accuracy

Bagging – e.g. Random Forest

- Bagging (Bootstrap AGGregatING) – involves having each model in the ensemble vote with equal weight. In order to promote model variance, bagging trains each model in the ensemble using a randomly drawn subset of the training set.
- Example: Random Forest algorithm combines random decision trees with bagging to achieve very high classification accuracy. Given a training set X = x1, ..., xn with responses Y = y1, ..., yn, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples + feature bagging:
  1. For b = 1, ..., B:
  2. Sample, with replacement, n training examples from X, Y; call these Xb, Yb.
  3. Train a classification or regression tree fb on Xb, Yb, by selecting a limited number of features (ayttributes) to be used during the training (feature bagging).
  4. After training, predictions for unseen samples x' can be made by averaging the predictions / taking the majority vote from all the individual regression trees on x‘.

Decision Trees. Random Forests

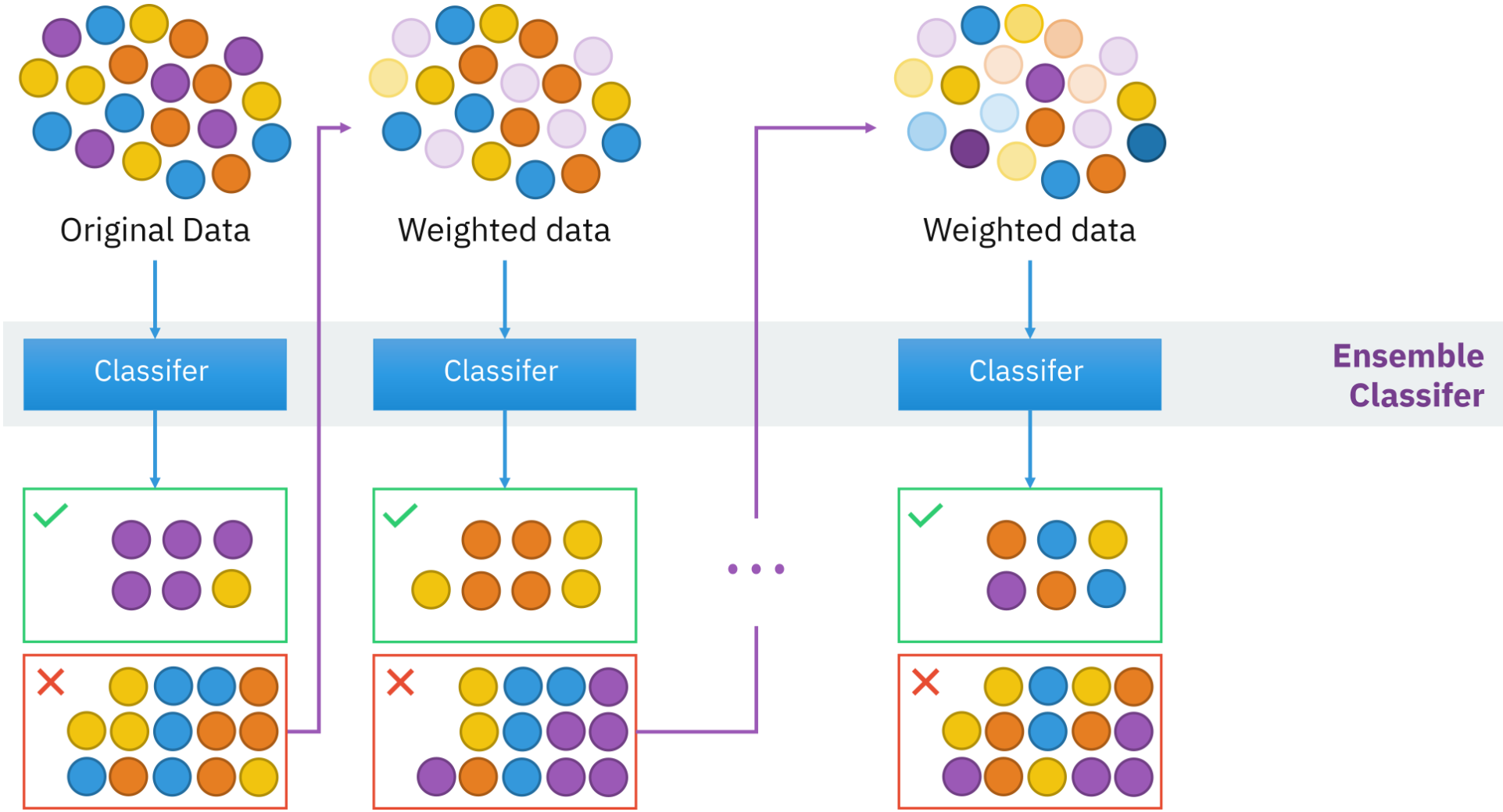




Boosting

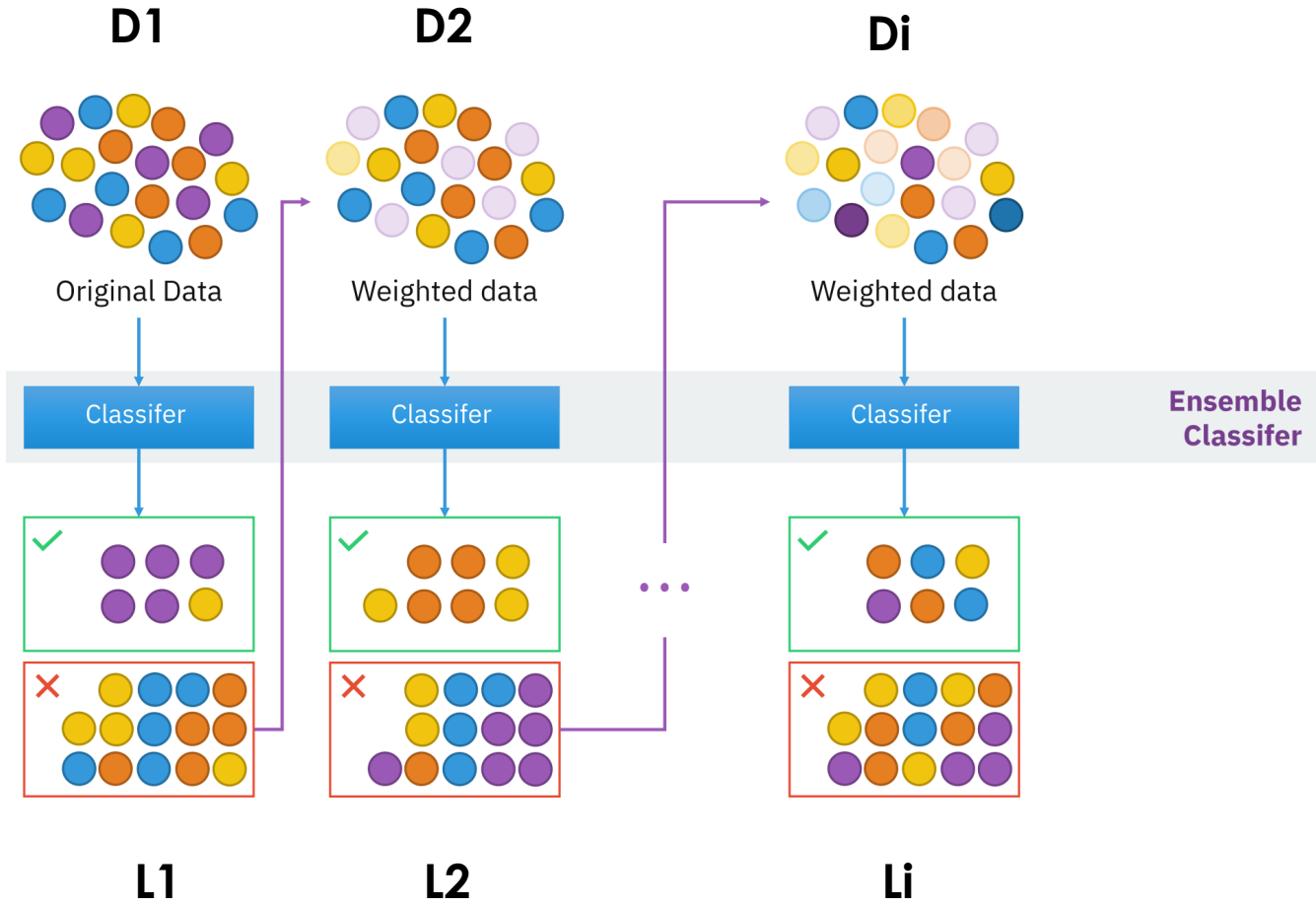
- Boosting - ensemble meta-algorithm for primarily reducing bias, and also variance, based on the question posed by Kearns and Valiant (1988, 1989"Can a set of weak learners create a single strong learner?"
- In supervised learning, and a family of machine learning algorithms that convert weak learners to strong ones
- Weak learner – a classifier that is only slightly correlated with the true classification (it can label examples better than random guessing). In contrast, a strong learner is a classifier that is arbitrarily well-correlated with the true classification.
- Informally, [the hypothesis boosting] problem asks whether an efficient learning algorithm […] that outputs a hypothesis whose performance is only slightly better than random guessing [i.e. a weak learner] implies the existence of an efficient algorithm that outputs a hypothesis of arbitrary accuracy [i.e. a strong learner]."

Boosting - Basic Algorithm II



Boosting – Basic Algorithm

1. Equal weight (uniform probability distribution) is given to the sample training data (say D_1) initially. Let **i = 1**
2. This data (D_i) is then given to a base learner (say L_i).
3. Mis-classified instances by ensemble $L_1, L_2, \ldots L_i$ are assigned a weight higher than the correctly classified instances (keeping the total probability distribution = 1) => D_{i+1} – boosted data
4. Let **i = i + 1**. Go to step 2 until error becomes acceptable (or other finish criteria is met).



AdaBoost

- AdaBoost - the output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to be strong learner.
- Every learning algorithm tends to suit some problem types better than others, and typically has many different parameters and configurations to adjust before it achieves optimal performance on a dataset. AdaBoost (with decision trees as the weak learners) is often referred to as the best out-of-the-box classifier. When used with decision tree learning, information gathered at each stage of the AdaBoost algorithm about the relative 'hardness' of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples.

Discrete AdaBoost

With:

- Samples $x_1 \dots x_n$
- Desired outputs $y_1 \dots y_n, y \in \{-1, 1\}$
- Initial weights $w_{1,1} \dots w_{n,1}$ set to $\frac{1}{n}$
- Error function $E(f(x), y, i) = e^{-y_i f(x_i)}$
- Weak learners $h: x \to \{-1, 1\}$

For t in $1 \dots T$:

- Choose $h_t(x)$:
  - Find weak learner $h_t(x)$ that minimizes ϵ_t, the weighted sum error for misclassified points $\epsilon_t = \sum_{\substack{i=1 \\ h_t(x_i) \neq y_i}}^{n} w_{i,t}$
  - Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$
- Add to ensemble:
  - $F_t(x) = F_{t-1}(x) + \alpha_t h_t(x)$
- Update weights:
  - $w_{i,t+1} = w_{i,t} e^{-y_i \alpha_t h_t(x_i)}$ for i in $1 \dots n$
  - Renormalize $w_{i,t+1}$ such that $\sum_i w_{i,t+1} = 1$
  - (Note: It can be shown that $\frac{\sum_{h_{t+1}(x_i)=y_i} w_{i,t+1}}{\sum_{h_{t+1}(x_i)\neq y_i} w_{i,t+1}} = \frac{\sum_{h_t(x_i)=y_i} w_{i,t}}{\sum_{h_t(x_i)\neq y_i} w_{i,t}}$ at every step, which can simplify the calculation of the new weights.)

Source: https://en.wikipedia.org/wiki/AdaBoost

Gradient Boosting

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
    1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

    2. Fit a base learner (or weak learner, e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
    3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

    4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Hastie, T.; Tibshirani, R.; Friedman, J. H. (2009). "10. Boosting and Additive Trees". The Elements of Statistical Learning (2nd ed.). New York: Springer. pp. 337–384. ISBN 978-0-387-84857-0. Archived from the original on 2009-11-10.

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD