# Reactive Manifesto

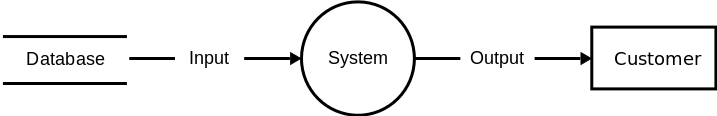
* Reactive Systems are **Responsive**, **Resilient**, **Elastic** and **Message Driven**.
  + **Responsive**: The system responds in a timely manner if at all possible.
  + **Resilient:** The system stays responsive in the face of failure.
  + **Elastic:** The system stays responsive under varying workload.
  + **Message Driven:** Reactive Systems rely on asynchronous message-passing to establish a boundary between components that ensures loose coupling, isolation and location transparency.
* This means that they are:
  + Flexible
  + Loosely-coupled
  + Scalable
  + Tolerant to failures
  + Highly responsive

Sources:

<https://www.reactivemanifesto.org/>

# Dataflow Programming (DFP)

* Represents applications as a directed graph, similarly to a dataflow diagram.



Data flow diagram with data storage, data flows, function and interface

(A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system)

* Applications are represented as a set of nodes (also called blocks) with input and/or output ports in them.
* These nodes can either be sources, sinks or processing blocks to the information flowing in the system.
* Nodes are connected by directed edges that define the flow of information between them.
* (Saul’s note) Visual programming languages take advantage of DFP.
* Implicit achievement of concurrency
  + Each node is an independent processing block working independently from any others.
  + Such execution model allows nodes to execute as soon as data arrives to them, without the possibility of creating deadlocks, as there are no data dependencies in the whole system
* Introduced by Gilles Kahn, the Kahn Process Networks approached this problem by having sequential processes (nodes) to communicate via unbounded FIFO queues as message passing protocol.
  + Whenever the entry FIFO queue of a node was not empty, the first value would be processed by the node and outputted into the FIFO belonging to the next node in the chain.

Sources:

<https://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf>

<https://en.wikipedia.org/wiki/Data-flow_diagram>

# Functional Programming

* Functional programming is a programming paradigm — a style of building the structure and elements of computer programs — that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.
* A pure function is
  + Deterministic
  + Doesn’t cause any side effects
* Benefits of a pure function
  + Easier to test (No need to mock)
  + Leads to declarative programs, which can simplify the systems
  + Referential transparency (Defined as the fact that an expression, in a program, may be replaced by its value). This makes the code context independent.
  + Pure functions can always be **parallelized**.
  + **Memoization:** Because pure functions are referentially transparent, we only need to compute their output once for given inputs.
  + **Laziness:** Delay the evaluation of an expression until its value is needed.
* Refactoring to functional: One fruitful approach is to separate the pure, functional, value based core of your application from an outer, imperative shell.
* Avoid mutability
  + Whenever you need to model a state change, you pass the previous value to a function that returns a new value. Don’t change the old value just return a new one.

Sources:

<https://medium.freecodecamp.org/an-introduction-to-the-basic-principles-of-functional-programming-a2c2a15c84>

<https://www.sitepoint.com/functional-programming-pure-functions/>

<https://www.sitepoint.com/what-is-referential-transparency/>

TODO: <https://medium.com/javascript-scene/curry-and-function-composition-2c208d774983>