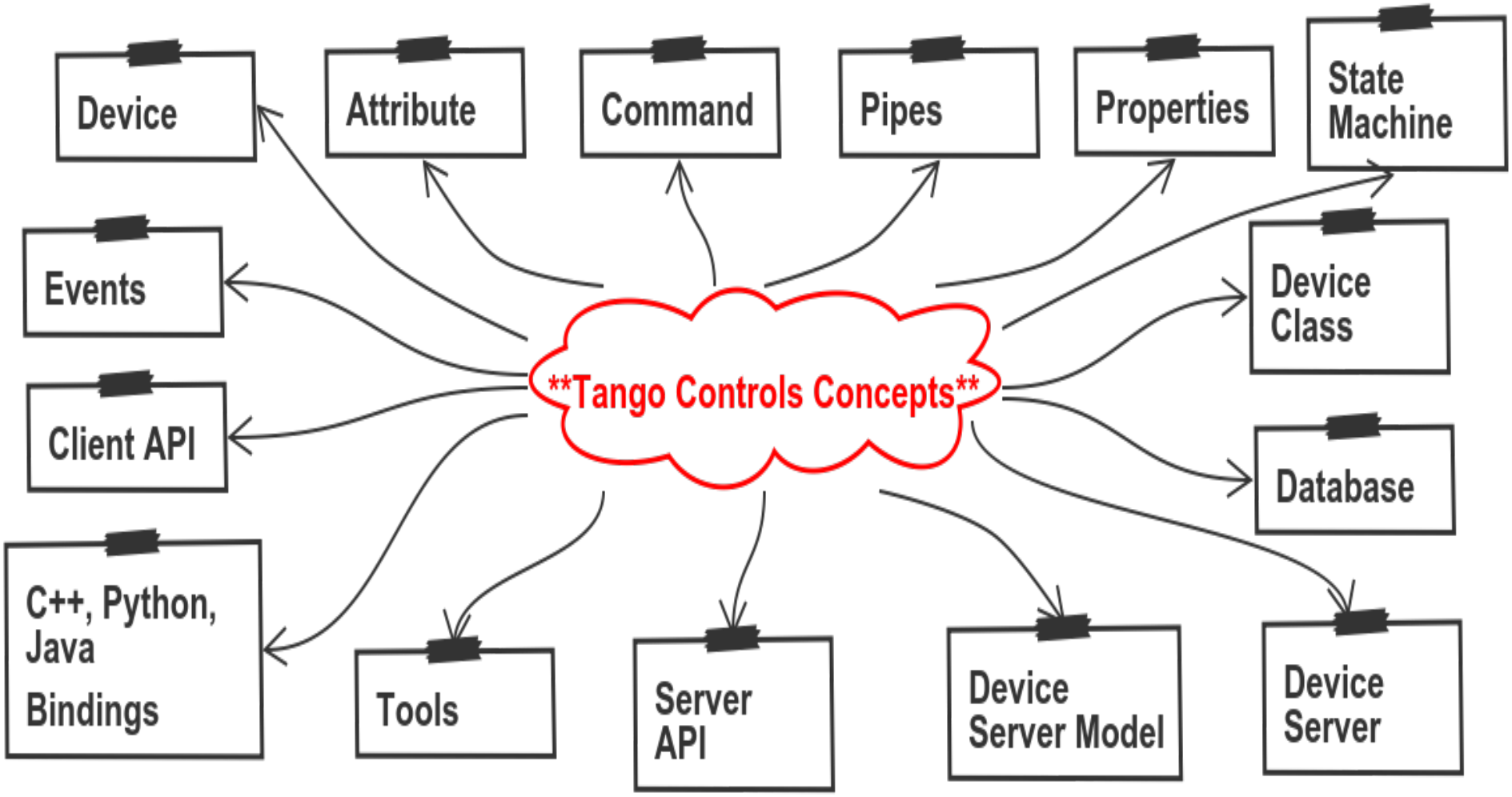

TANGO
controls

TANGO CONTROLS CONCEPTS

A brief introduction to Tango Controls Concepts

Andy Götz

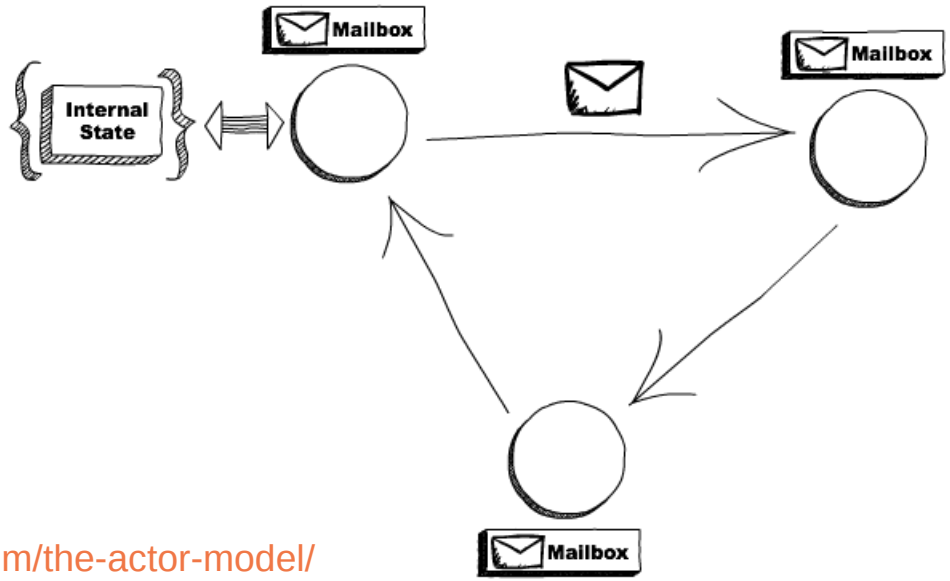
TANGO HAS A NUMBER OF CONCEPTS



TANGO = ACTORS + MICROSERVICES

- Tango is based on the concept of **Distributed Devices**
- This is an implementation of the **Actor Model**
- Device servers implement **Microservices**
- **Actors** + **Microservices** are *a la mode*
- TANGO is based on MODERN concepts !

ACTOR MODEL



http://www.brianstorti.com/the-actor-model/

The actor model in computer science is a mathematical model of concurrent computation that treats "actors" as the universal primitives of concurrent computation. In response to a message that it receives, an actor can: make local decisions, create more actors, send more messages, and determine how to respond to the next message received. Actors may modify private state, but can only affect each other through messages (avoiding the need for any locks).

Proposed in ***1973 by Carl Hewitt and others***

https://en.wikipedia.org/wiki/Actor_model

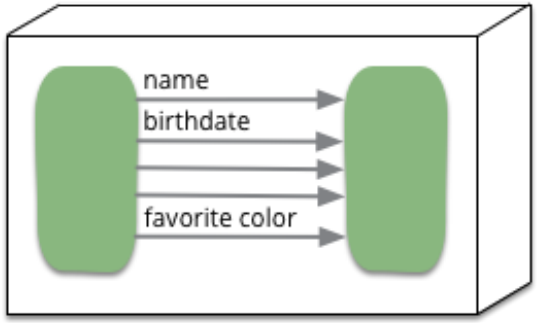
DISTRIBUTED OBJECTS

When I wrote Patterns of Enterprise Application Architecture, I coined what I called the First Law of Distributed Object Design: "don't distribute your objects". In recent months there's been a lot of interest in microservices, which has led a few people to ask whether microservices are in contravention to this law, and if so why I am in favor of them?

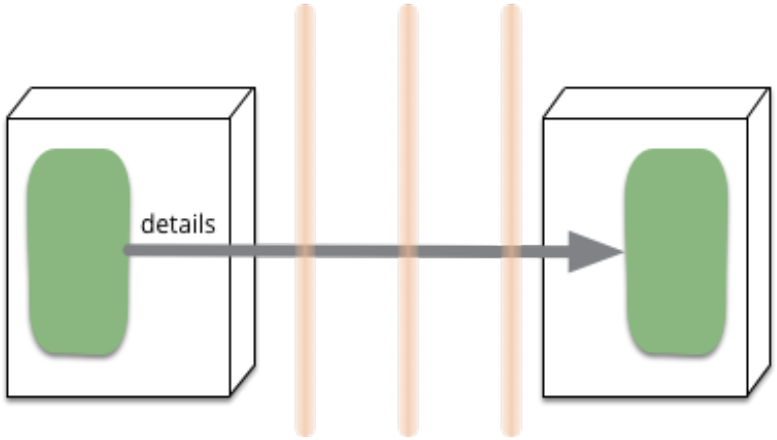
Martin Fowler

https://martinfowler.com/articles/distributed-objects-microservices.html

DISTRIBUTED OBJECTS



With two modules in the same process, it's best to use many fine-grained calls…



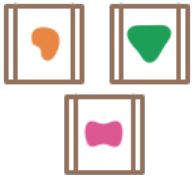
…but when modules are remote, then favor few coarse-grained calls.

Tango implements Microservices not Distributed Objects !

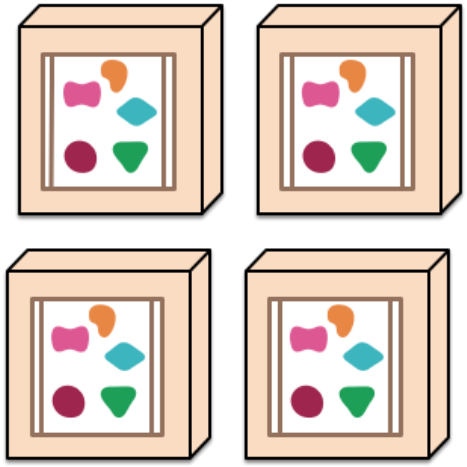
MICROSERVICES

A monolithic application puts all its functionality into a single process...

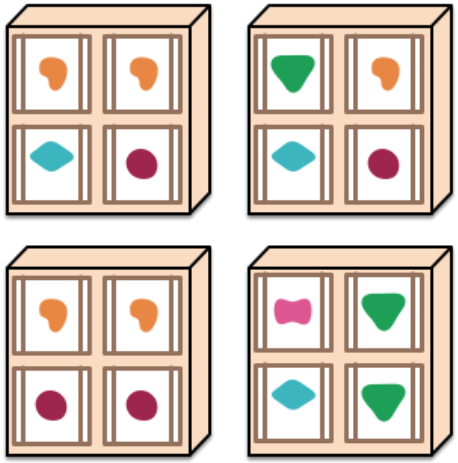
A microservices architecture puts each element of functionality into a separate service...



... and scales by replicating the monolith on multiple servers

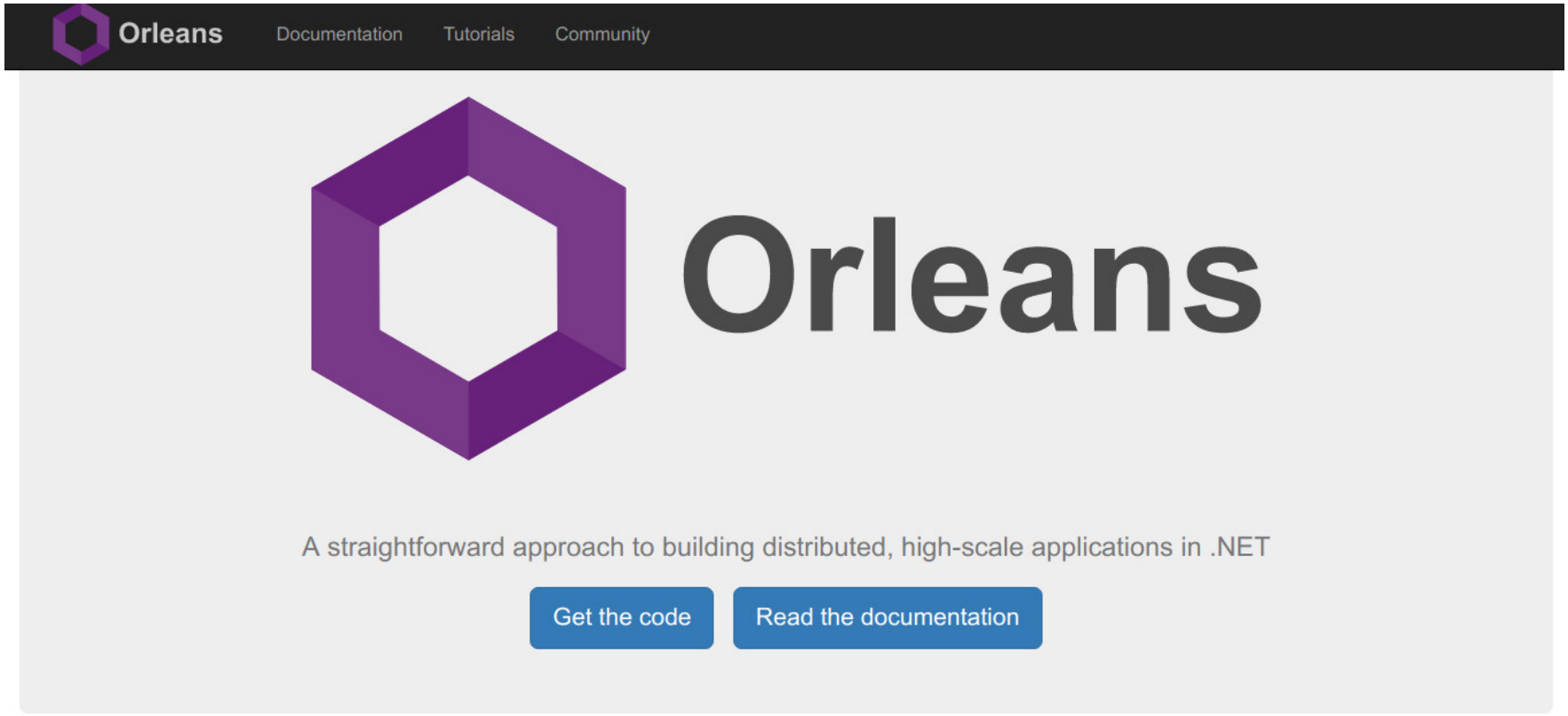


... and scales by distributing these services across servers, replicating as needed.



https://martinfowler.com/articles/microservices.html

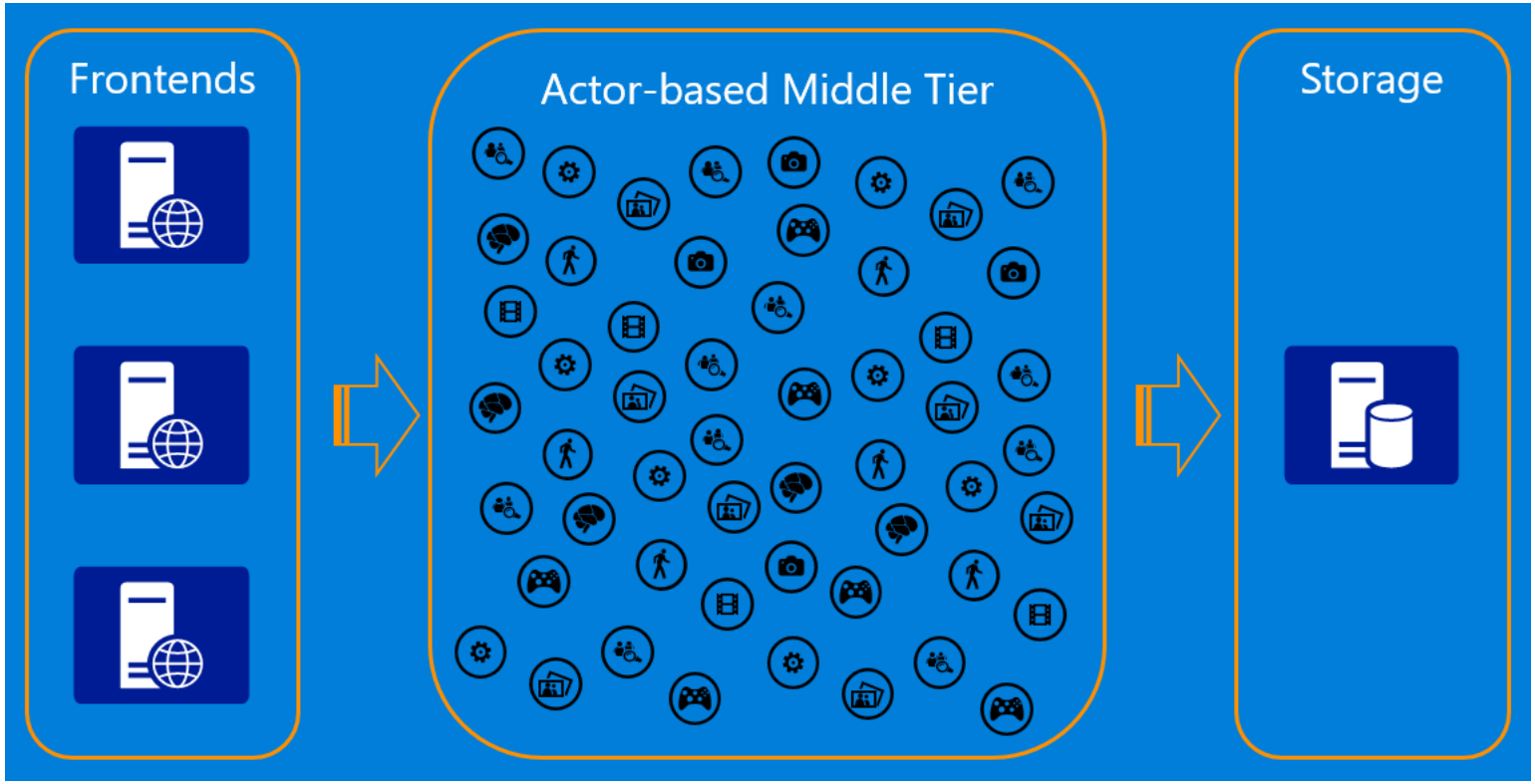
MICROSERVICES BY MICROSOFT



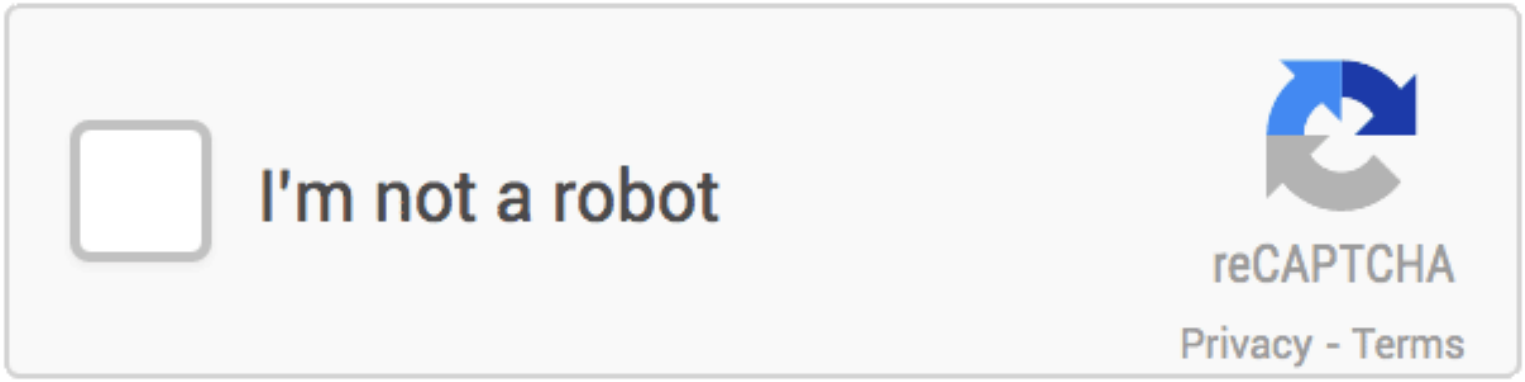
Orleans is a framework that provides a straightforward approach to building distributed high-scale computing applications, without the need to learn and apply complex concurrency or other scaling patterns. It was created by Microsoft Research and designed for use in the cloud.

Orleans has been used extensively in Microsoft Azure by several Microsoft product groups, most notably by 343 Industries as a platform for all of Halo 4 and Halo 5 cloud services, as well as by a growing number of other companies.

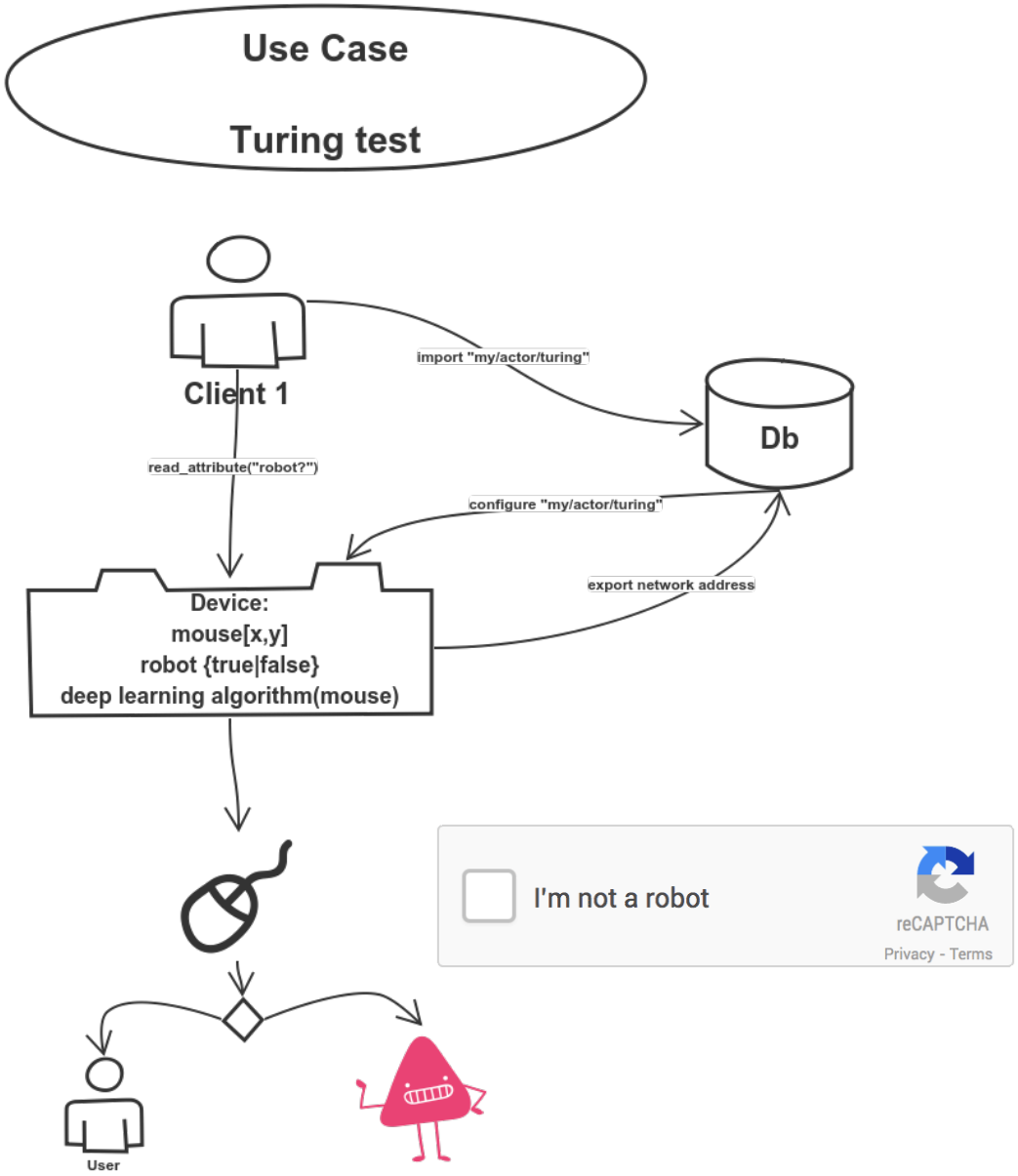
MICROSERVICES BY MICROSOFT



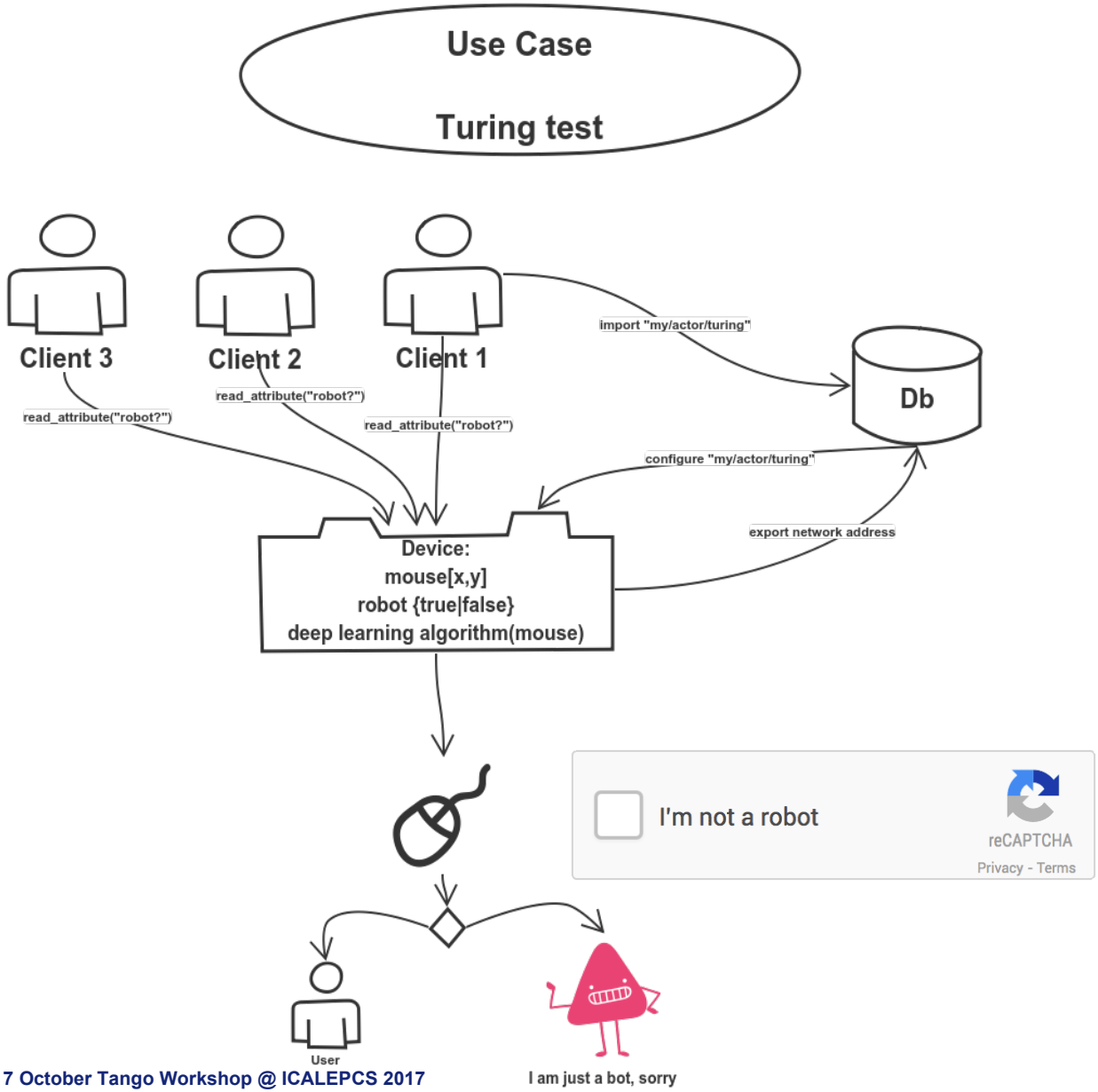
TANGO TURING TEST



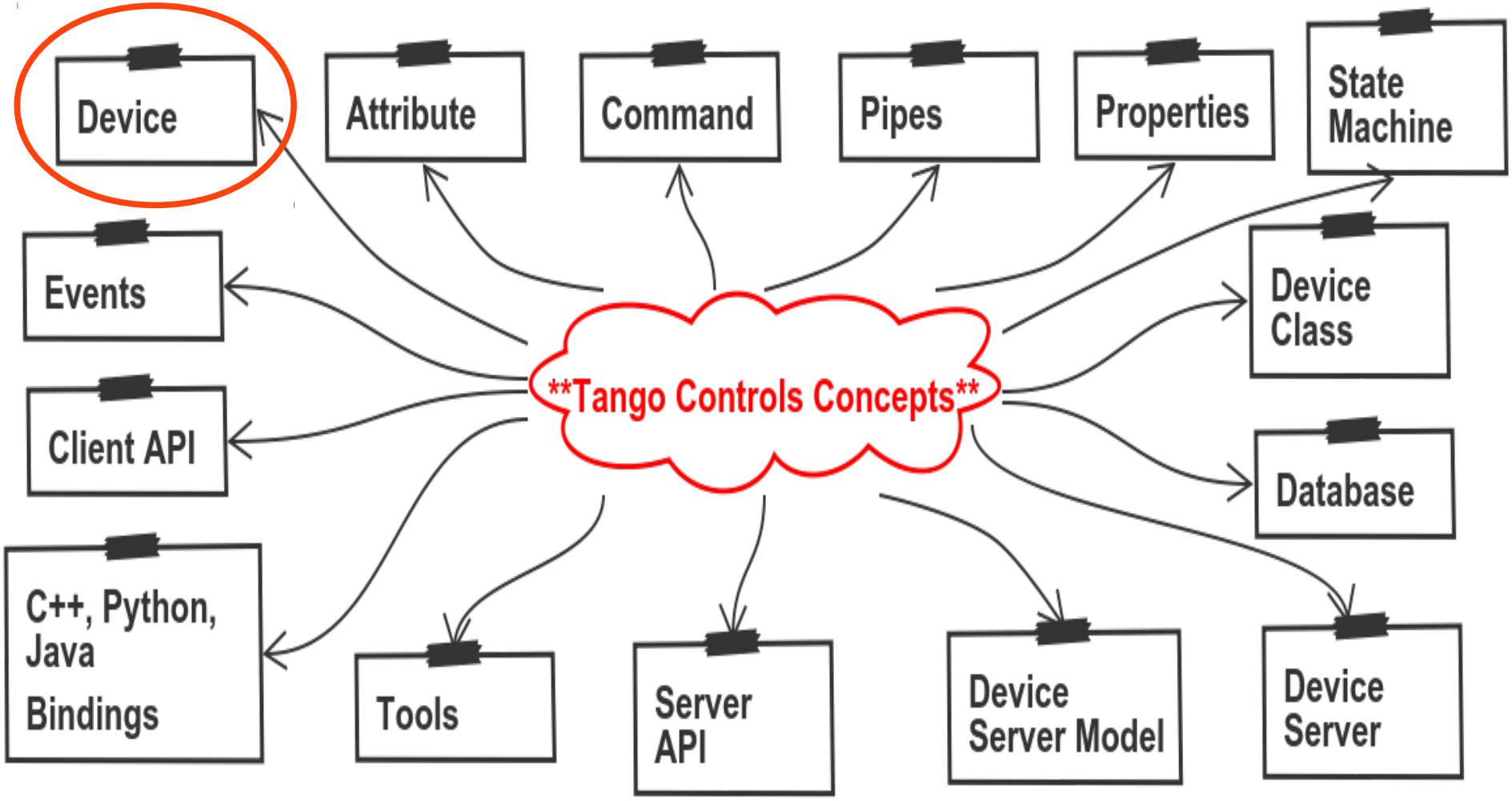
TANGO TURING TEST



TANGO TURING TEST - MULTIPLE CLIENTS



DEVICE CONCEPT



DEVICE CONCEPT

- Tango **Devices** are the objects which implement the microservices of a Tango System. Devices implement the sensors and actuators. Devices can be any piece of hardware or software.

- *Examples : motor, powersupply, camera, data analysis service, ...*

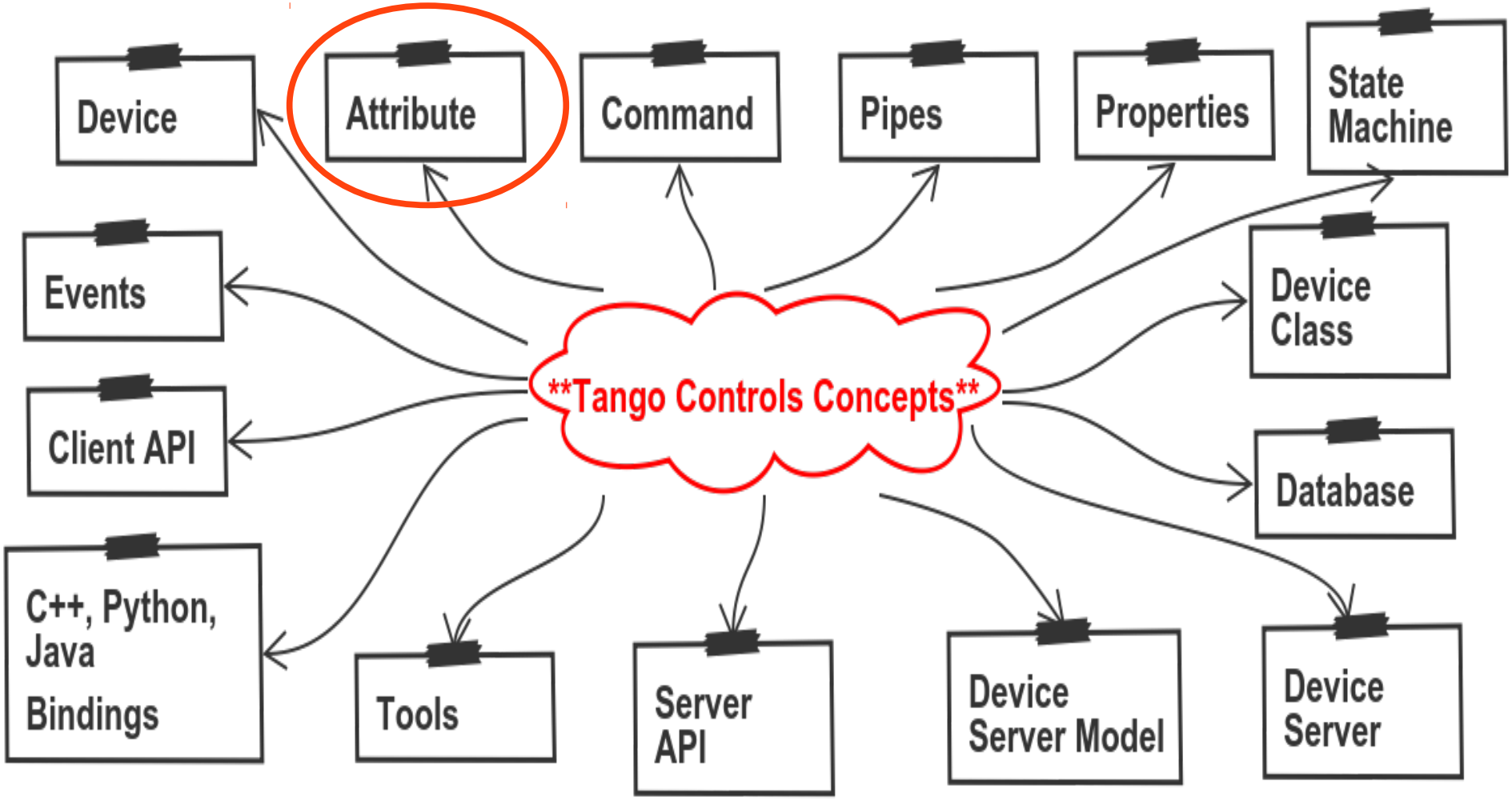
- Devices belong to a Device Class and are in a Device Server. They are stateful i.e. have State. Accessed via a common API. Have a unique 3 field name (D/F/M)

- Device Classes can be implemented in Python, C++ or Java

DEVICE THOUGHT EXPERIMENT

- How would you decompose your system into **Devices**?
- What **naming convention** to use? How many **hierarchies**?

ATTRIBUTE CONCEPT



Attribute concept

- Tango **Attributes** represents the data fields a Device wants clients to Read or Write or receive Events.

- *Examples : interlock bit, read/set value, spectrum, image, …*

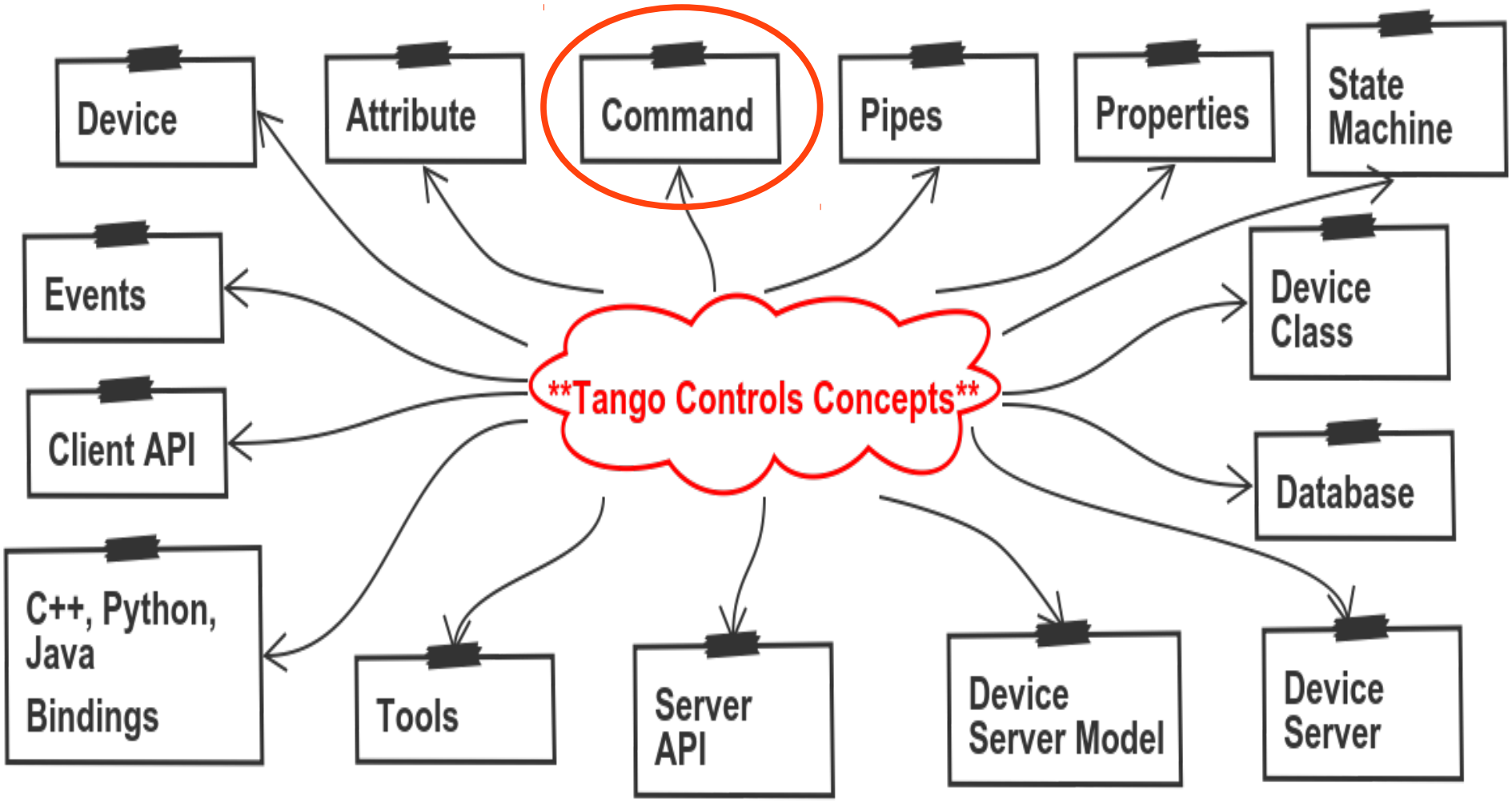
- Attributes can be **scalar**, **spectrum** (1D) or **images** (2D) and are **self describing** (units, min, max, alarms, display,…)

- **All** Device data should be provided as attributes (*well almost all!*). Attributes can be read one by one or many. Device developers have hooks for optimising attributes. Attributes respect the **State Machine**.

Attribute thought experiment

- What **Attributes** would you implement? Number? Size?
- What are the (network) **limitations** of attributes?

COMMAND CONCEPT



COMMAND CONCEPT

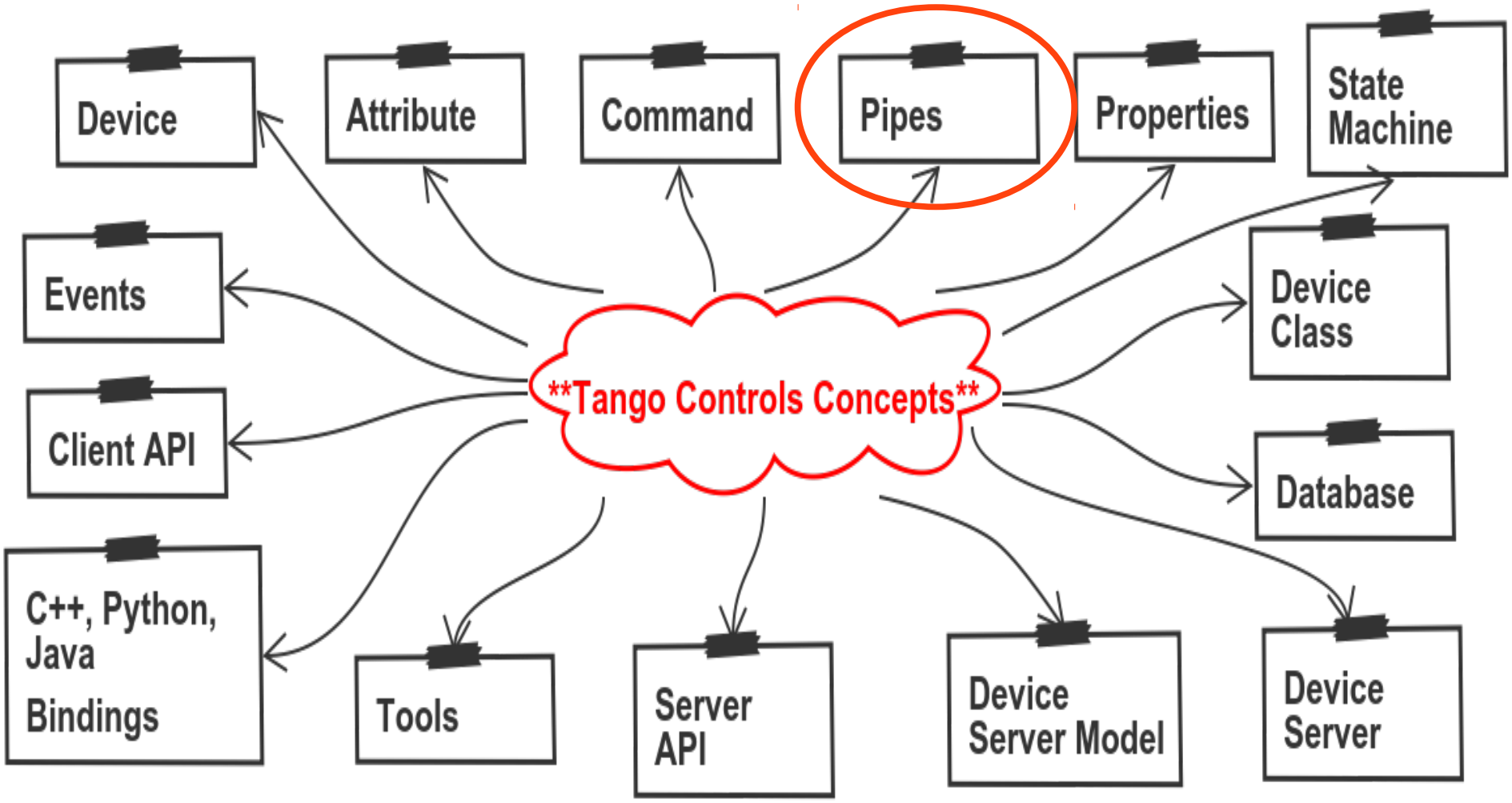
- Tango **Commands** are the actions of a Device the clients needs to execute. Commands affect State.
- *Examples : On, Off, Calibrate, Move, ...*
- Commands take **one input** and **one output parameter**. Parameters can be of any Tango data type.
- Commands always call the **State Machine**.

COMMAND THOUGHT EXPERIMENT

- What **Commands** would you implement? Are you sure your command should not be an attribute (get/set)?

- Are there any **Tango Data Types** missing for your case?

PIPE CONCEPT

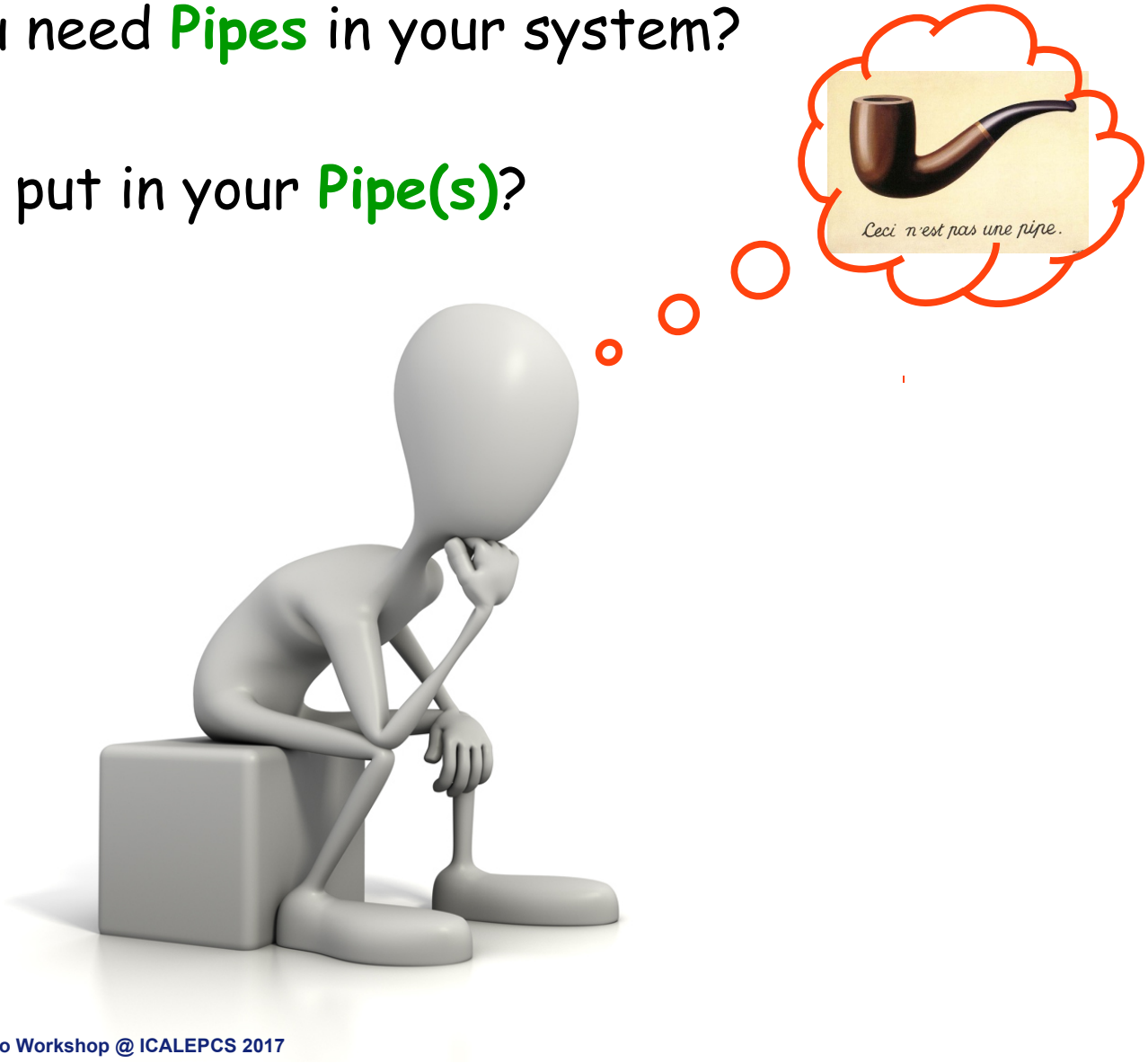


PIPE CONCEPT

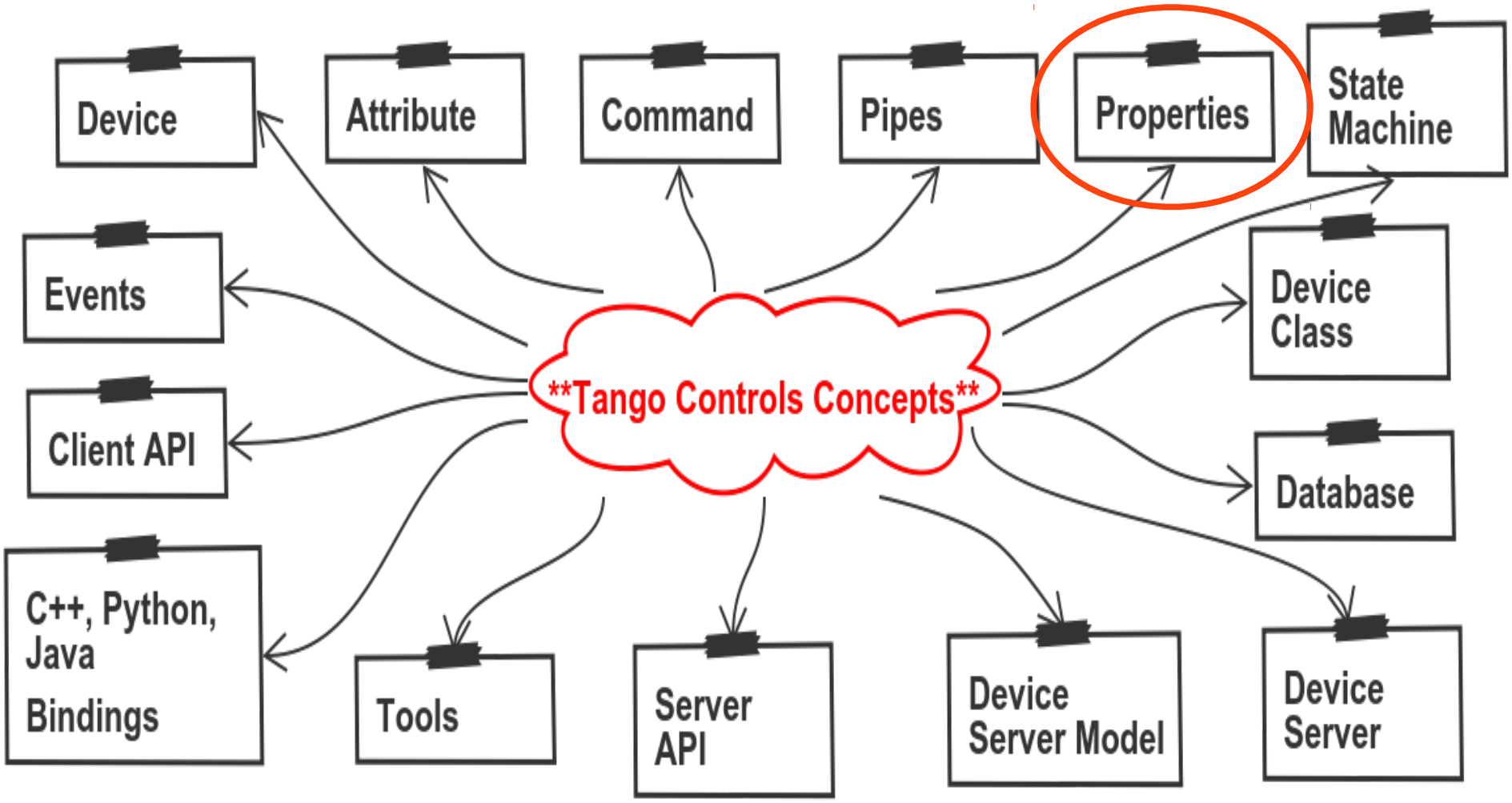
- Tango **Pipes** are data streams or channels for exchanging a stream of any Tango data type. Data types can be sent individually or grouped together in a Blob.
- *Examples : scanning data stream of mixed data types*
- Also used to circumvent the fixed data type set of Tango by sending mixed data types or a JSON blob.
- **DO NOT** only use Pipes (except in special cases)!
- **DO** use Attributes!

PIPE THOUGHT EXPERIMENT

- Where do you need **Pipes** in your system?
- What will you put in your **Pipe(s)**?



PROPERTIES CONCEPT



PROPERTIES CONCEPT

- Tango **Properties** are data stored in the database and used to configure Devices at startup. Properties can be any Tango Data types. Properties enable Device Classes to be generic. Properties are edited with Jive usually.

- *Examples : channel address, initial or current settings, sub-device names, ...*

- Changes to Properties can be persisted in the Database.

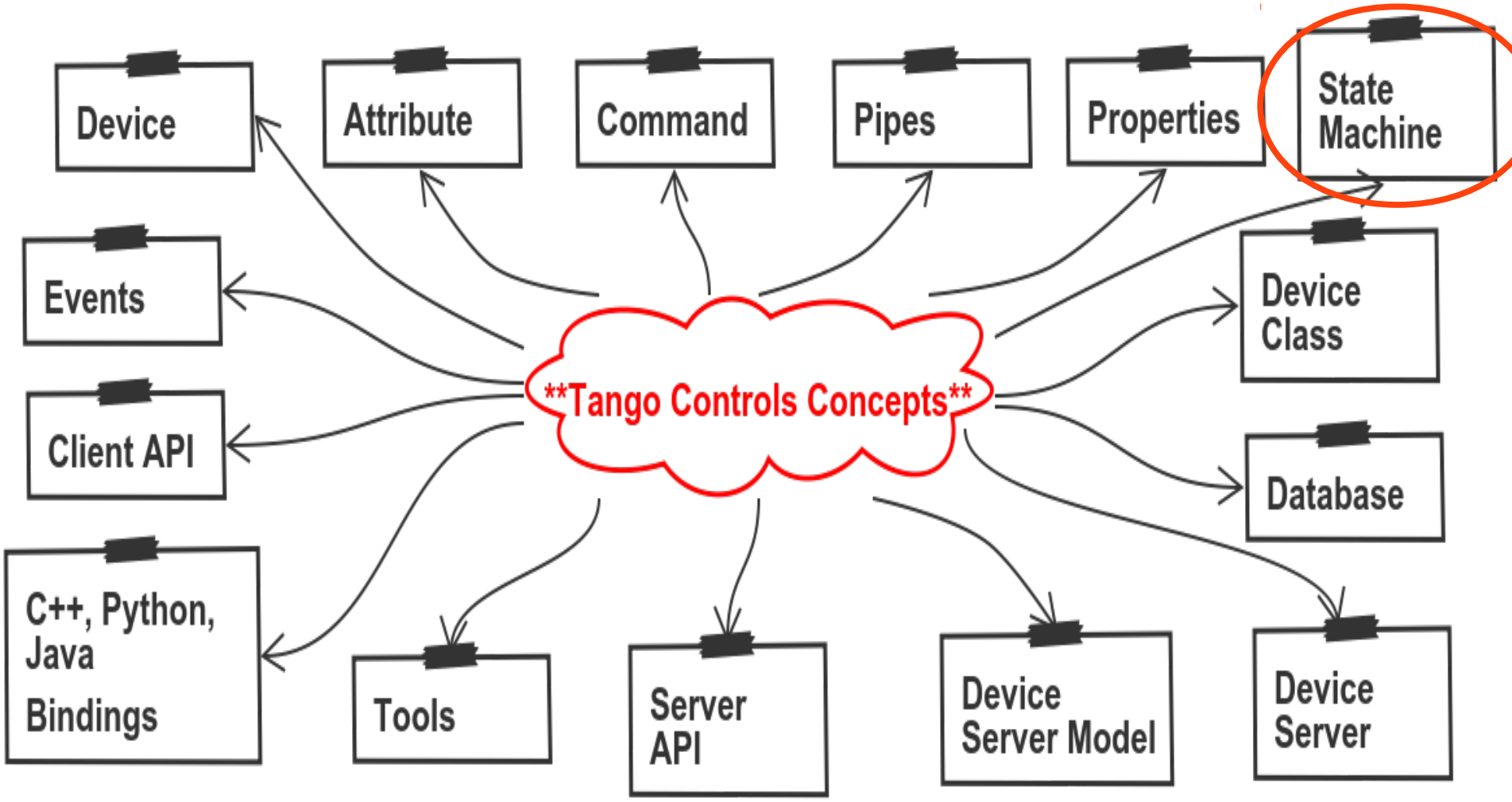
- **DO NOT** exit if Properties are wrong!

- **DO** use sensible default Properties!

PROPERTIES THOUGHT EXPERIMENT

- What **Properties** will you implement for your Devices?

STATE MACHINE CONCEPT



STATE MACHINE CONCEPT

- All Tango Devices have State. Tango States are limited to 14 discrete values. Each Tango Device Class **State Machine** implements the state transitions.

- *Examples : ON, OFF, FAULT, MOVING, OPEN, CLOSED, STANDBY, UNKNOWN*

- State is a very powerful mechanism for protecting Devices and for communicating changes to clients or servers.

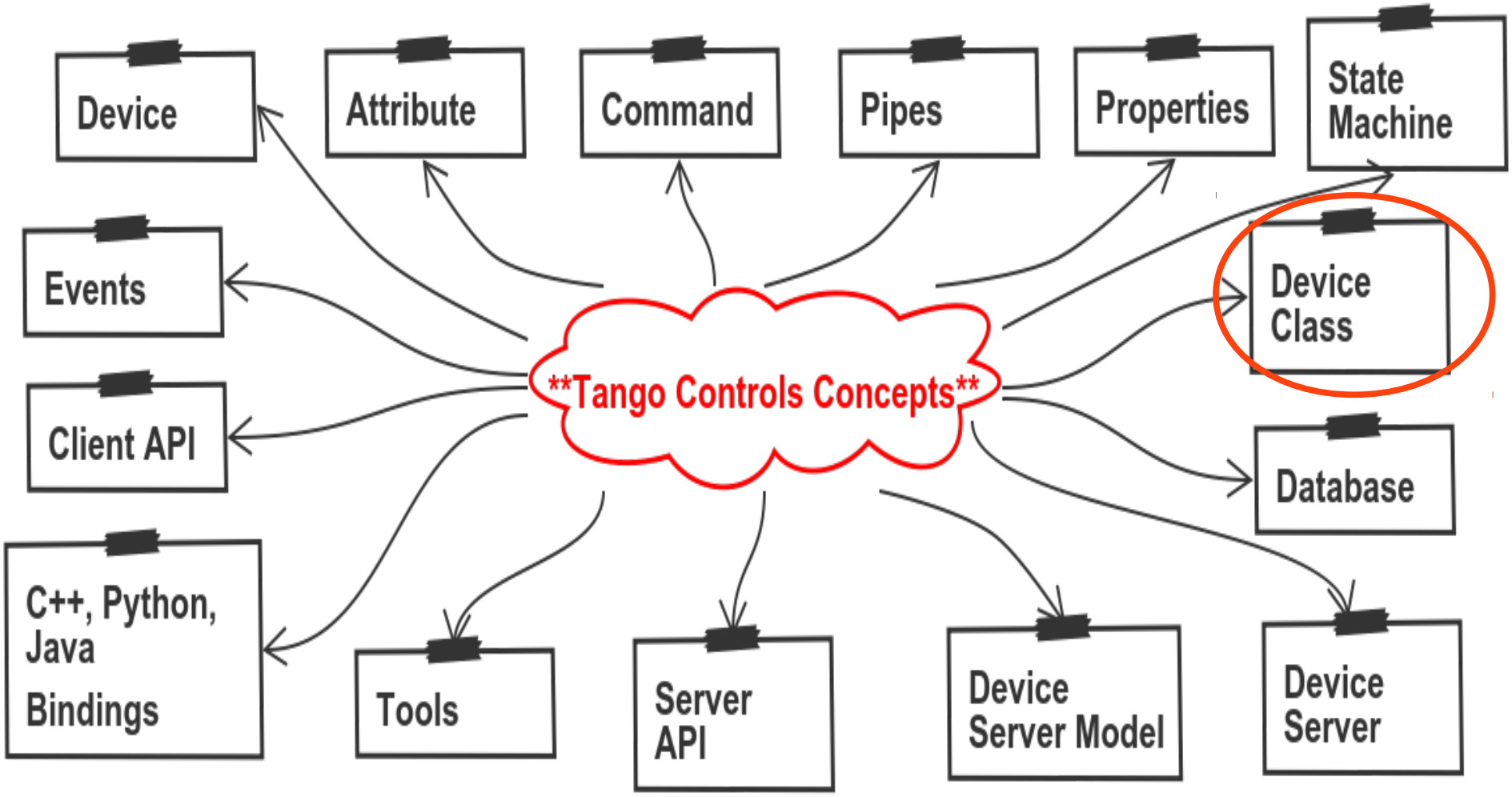
- **DO NOT** ignore State !

- **DO** set a default State!

State Machine Thought Experiment

- How will you map your **States** to **Tango States**?
- Do you really need more **States** or can they be implemented as attributes of enum type?

DEVICE CLASS CONCEPT



DEVICE CLASS CONCEPT

- All Tango Devices are implement by a **Device Class**. The Device Class implements a generic Device behaviour. Properties are used to configure the specific Device

- *Examples : MyPowerSupply, SerialLine, Polly*

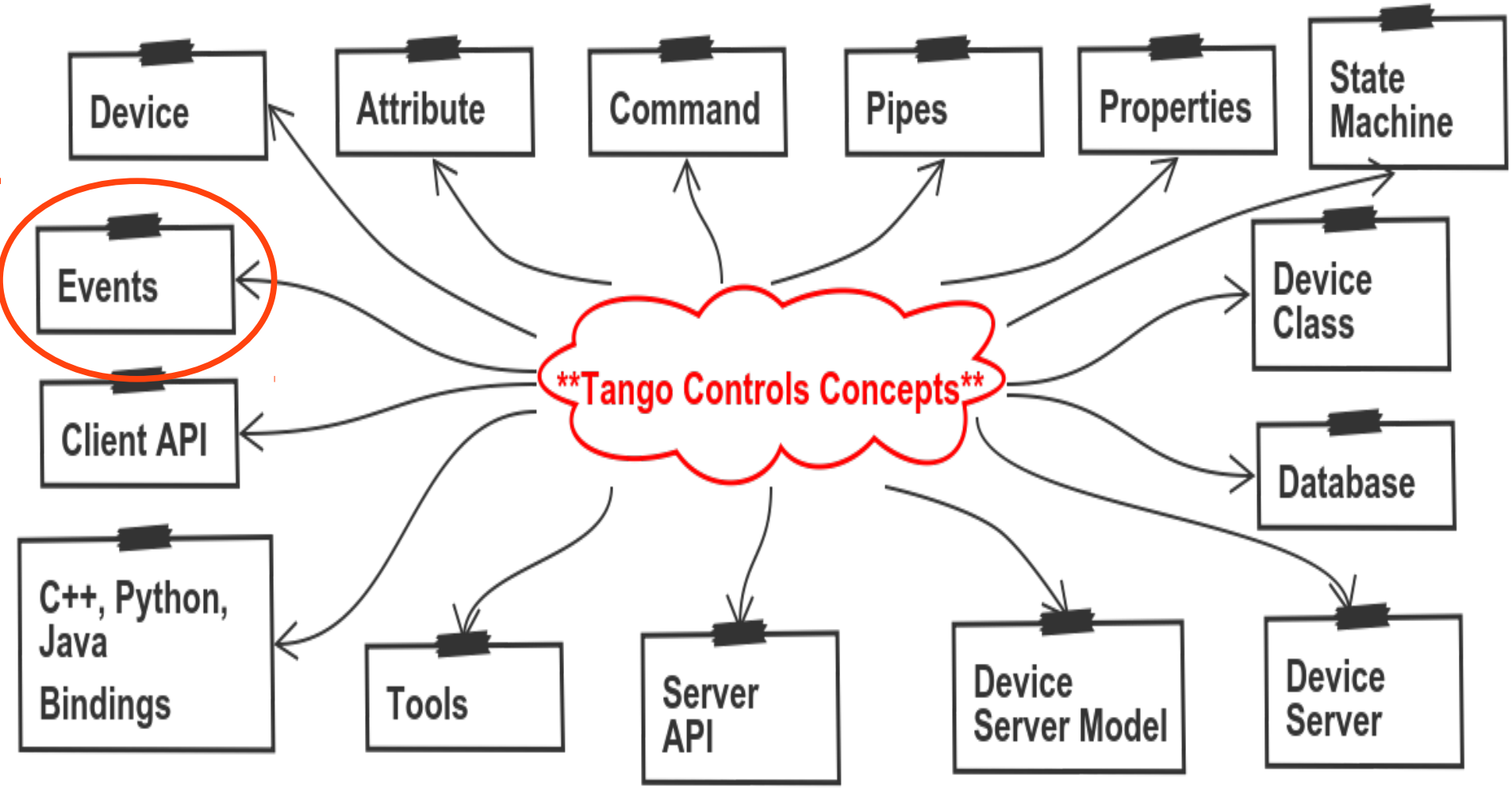
- Device Server developers are in fact developing Device Classes

- C++ developers have an extra class to develop – the so-called DeviceClassClass e.g. MyPowerSupplyClass. This is uses one of the Gang of Four patterns. Python and Java have only the DeviceClass.

DEVICE CLASS THOUGHT EXPERIMENT

- How many Tango Device Classes do you need?
- How would you encourage sharing of Device Classes?

EVENTS CONCEPT



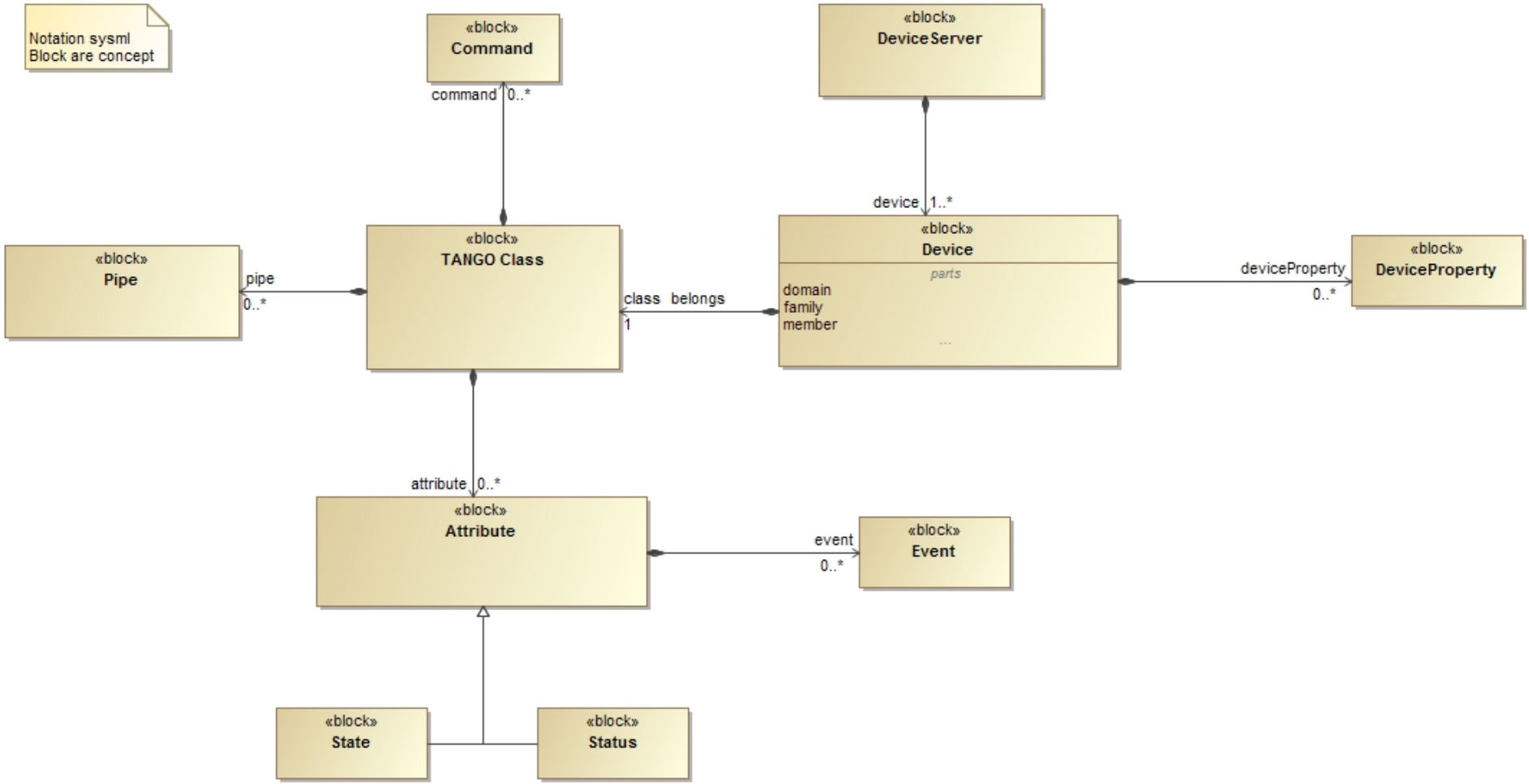
EVENTS CONCEPT

- Tango **Events** are a **Pub-Sub** communication between clients and servers. Events are only supported for **Attributes** and **Pipes**. Multiple Event types are supported – **Change, Periodic, Archive, User**, ...
- *Examples : Send Event if Attribute changes by x%*
- Events use ZMQ + are the most **efficient** way to communicate. Events rely on **Polling** to be triggered.
- **DO** understand and use Events fully
- **DO NOT** only use Polling

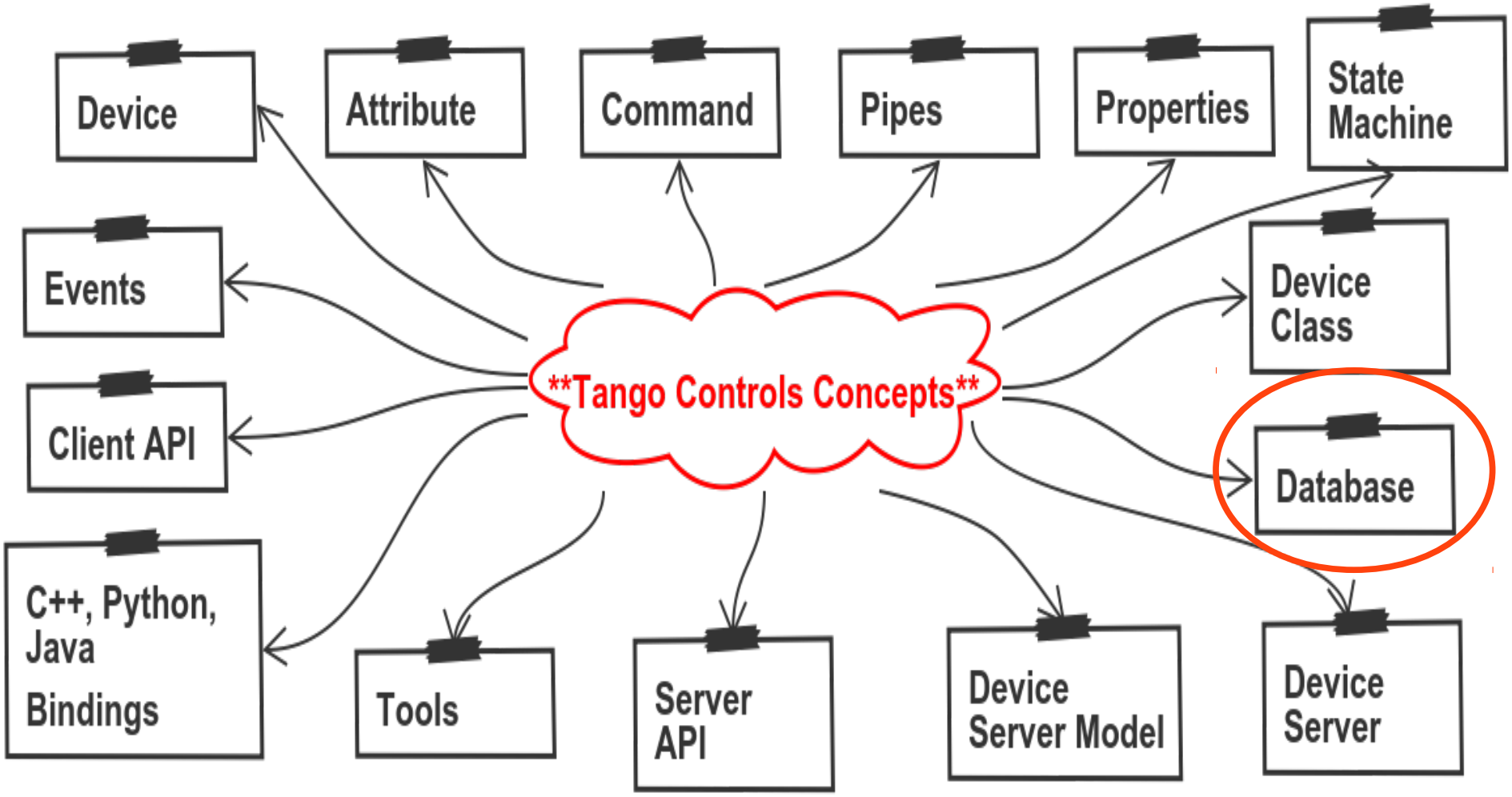
EVENTS THOUGHT EXPERIMENT

- Have you understood how Polling triggers Events?
- Will you use standard Events or User Events?

TANGO SIMPLE DEVICE MODEL



DATABASE CONCEPT



DATABASE CONCEPT

- Tango **Database** implements the Configuration and Naming Service for Tango. It can also persist set values.

- *Examples : configuration properties, export/import*

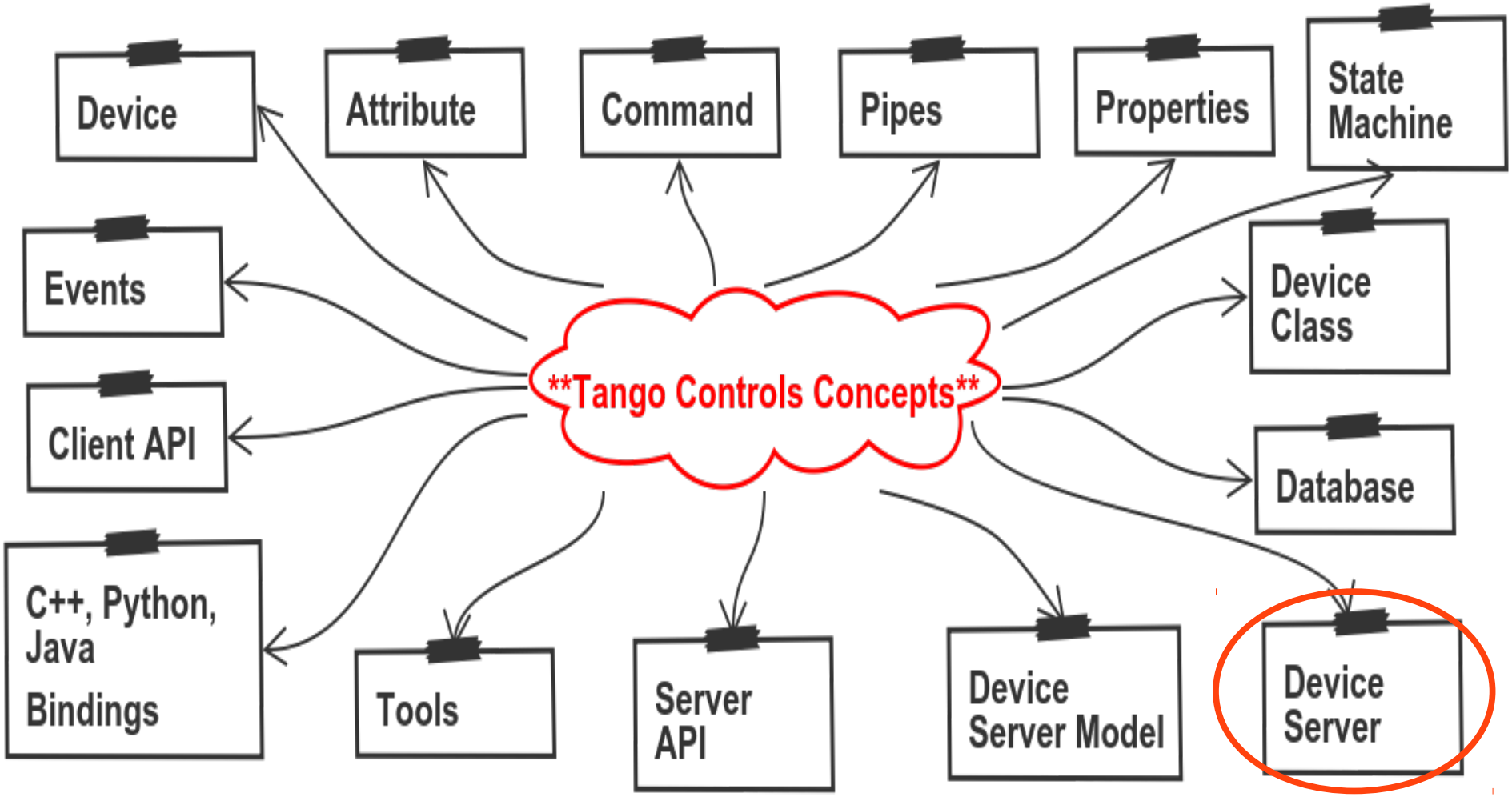
- Tango Database is implemented as a Device Server. Clients use the Tango Client API and Data Types to access the Database. Only MySQL is supported.

- Database is only fixed address (TANGO_HOST=host:port) environment variable. Multiple Databases supported.

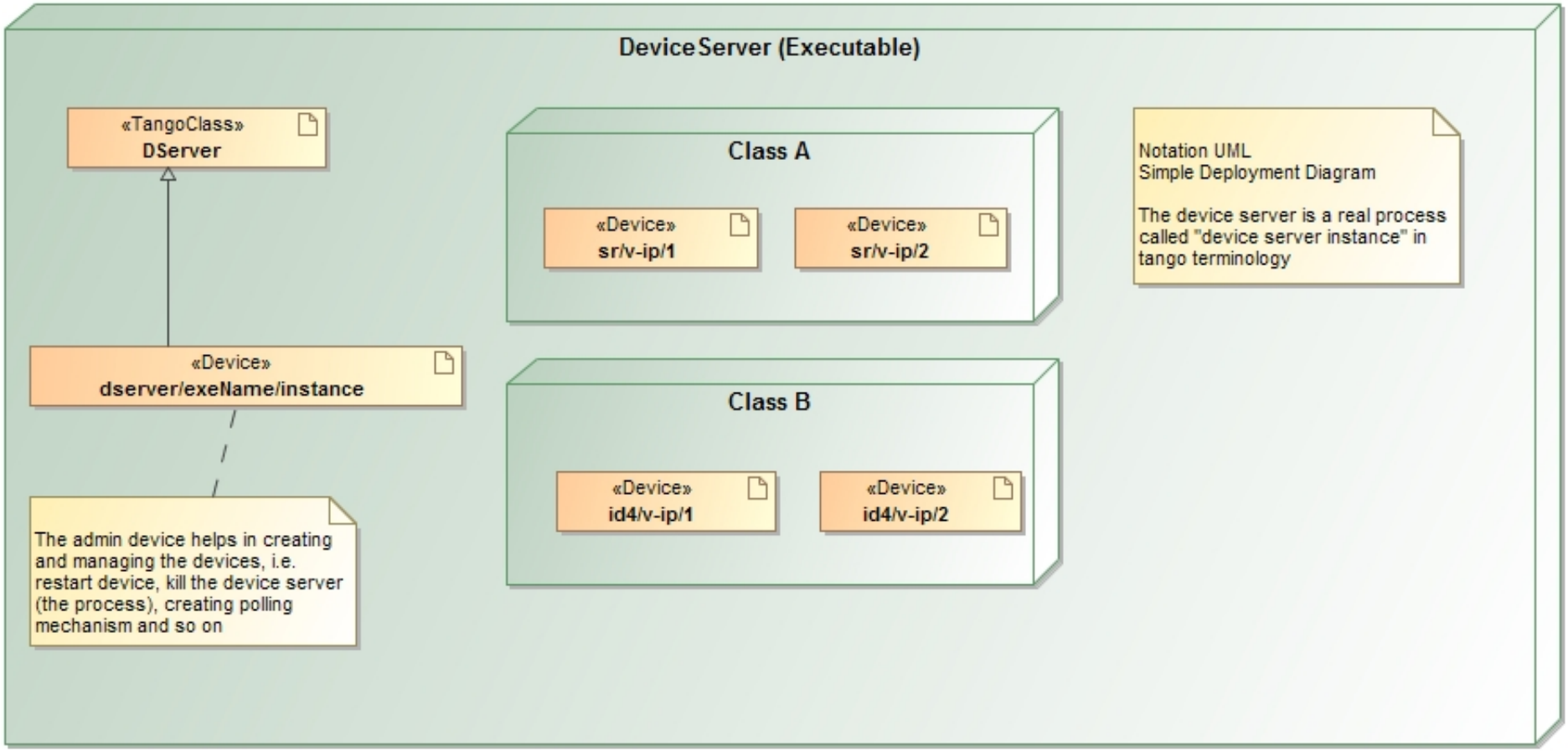
DATABASE THOUGHT EXPERIMENT

- Have you understood how multiple Database support systems of systems?
- How to reduce the single point of failure?

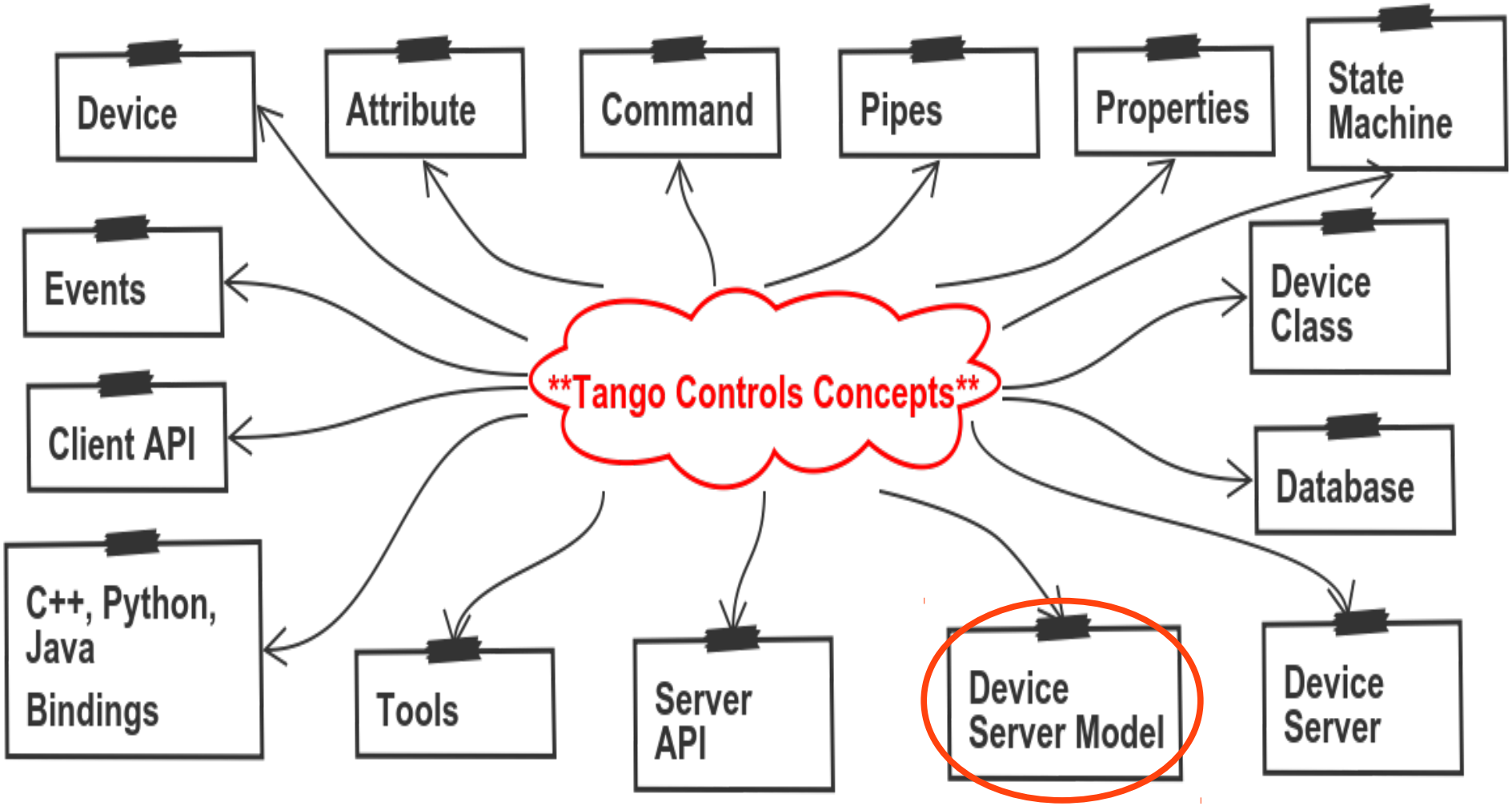
DEVICE SERVER CONCEPT



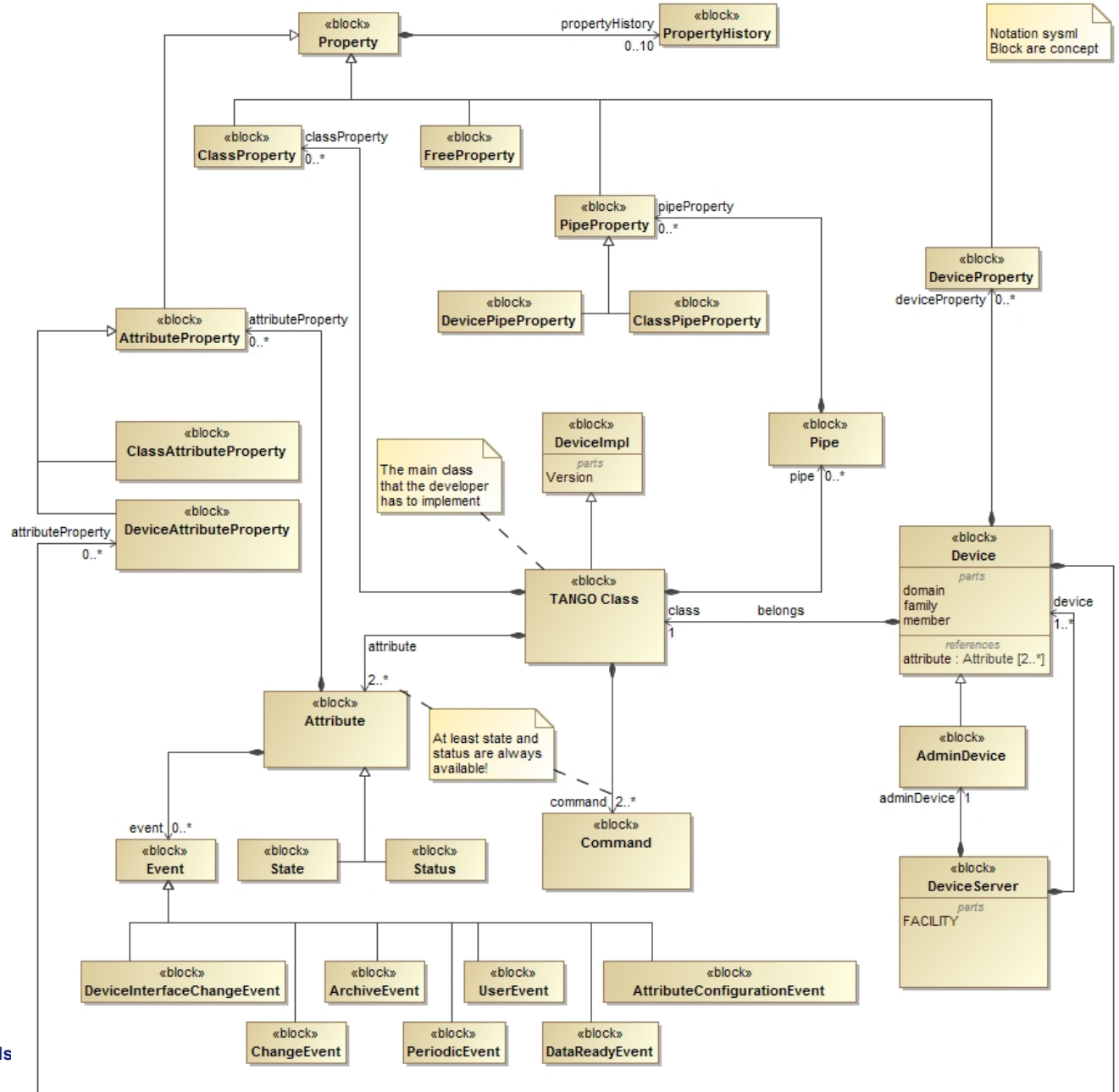
TANGO RUNTIME DEVICE MODEL



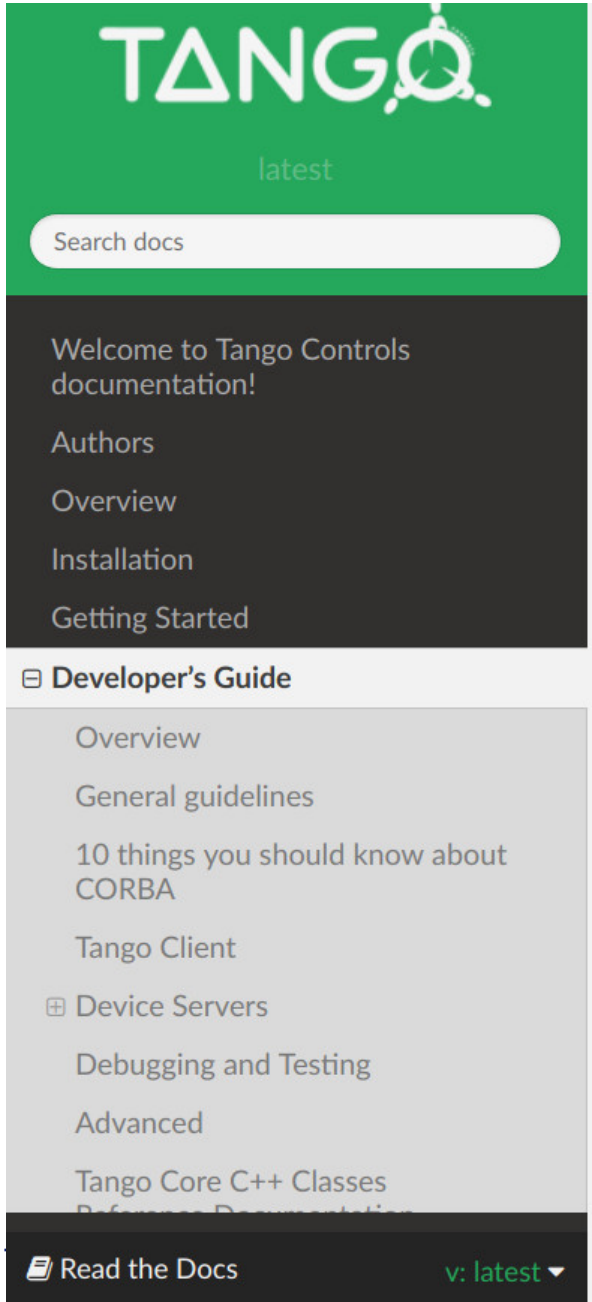
DEVICE SERVER MODEL CONCEPT



TANGO FULL DEVICE MODEL



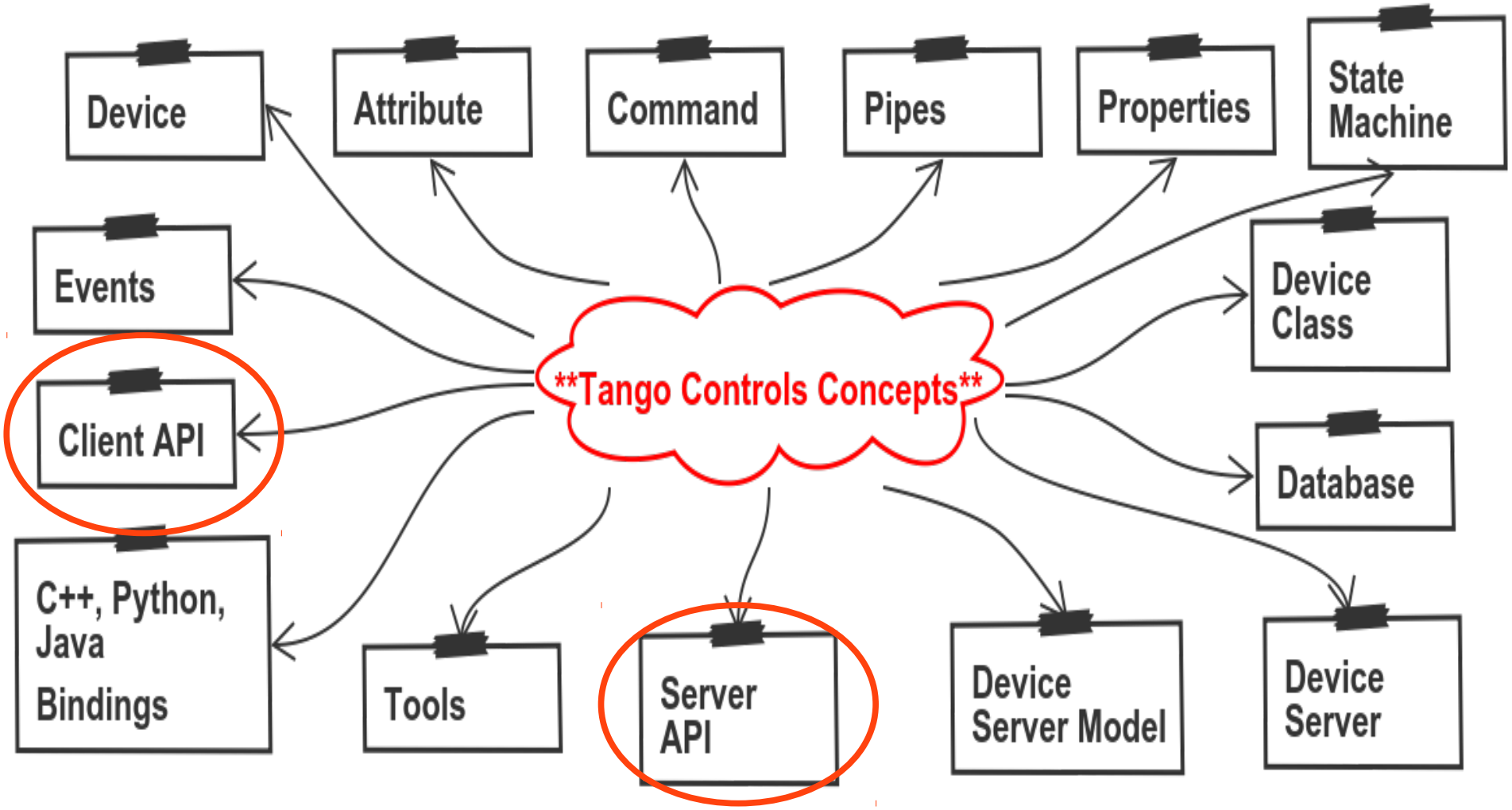
TANGO DEVELOPERS GUIDELINES



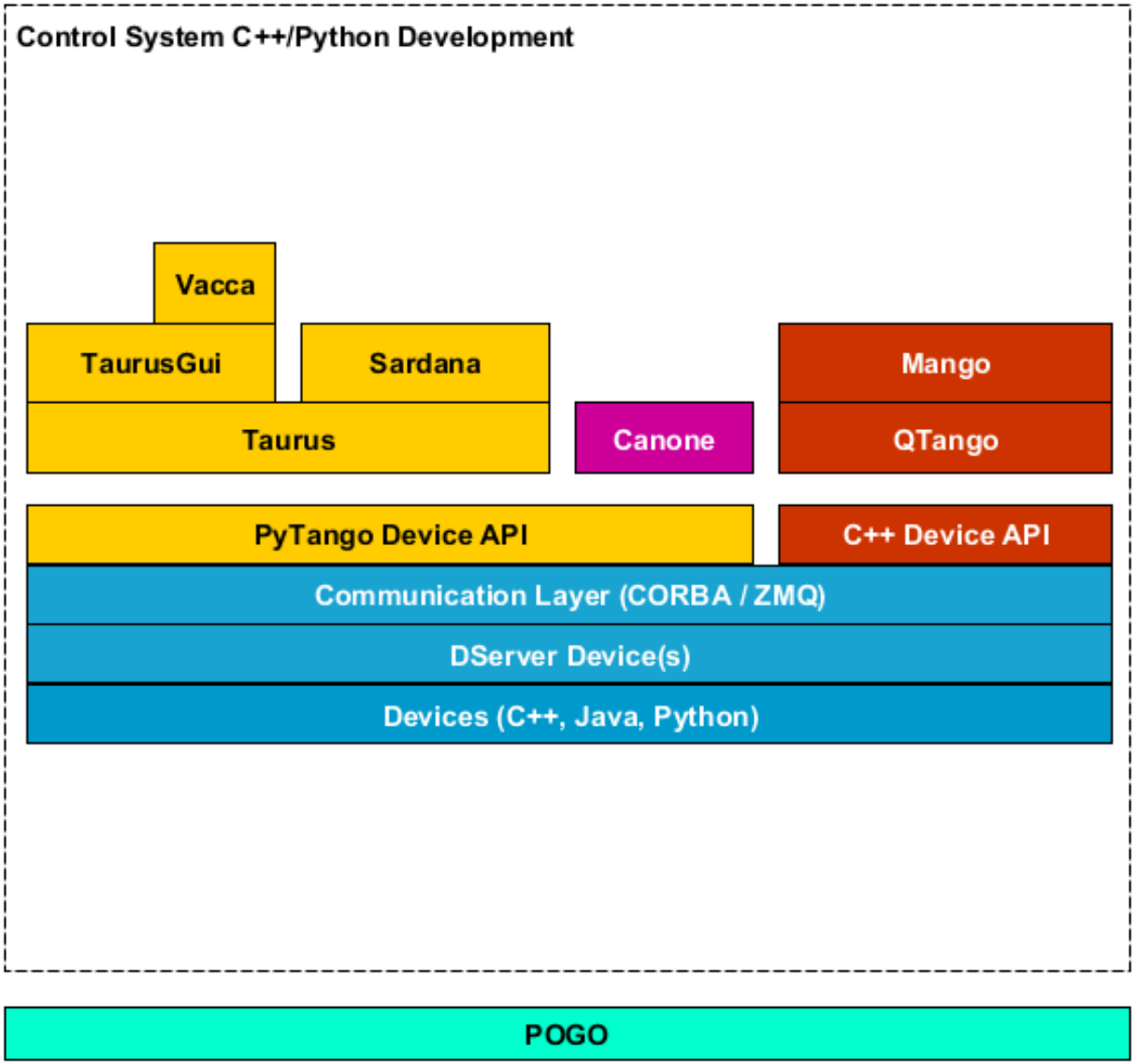
TANGO Device Server Guidelines

Contents:

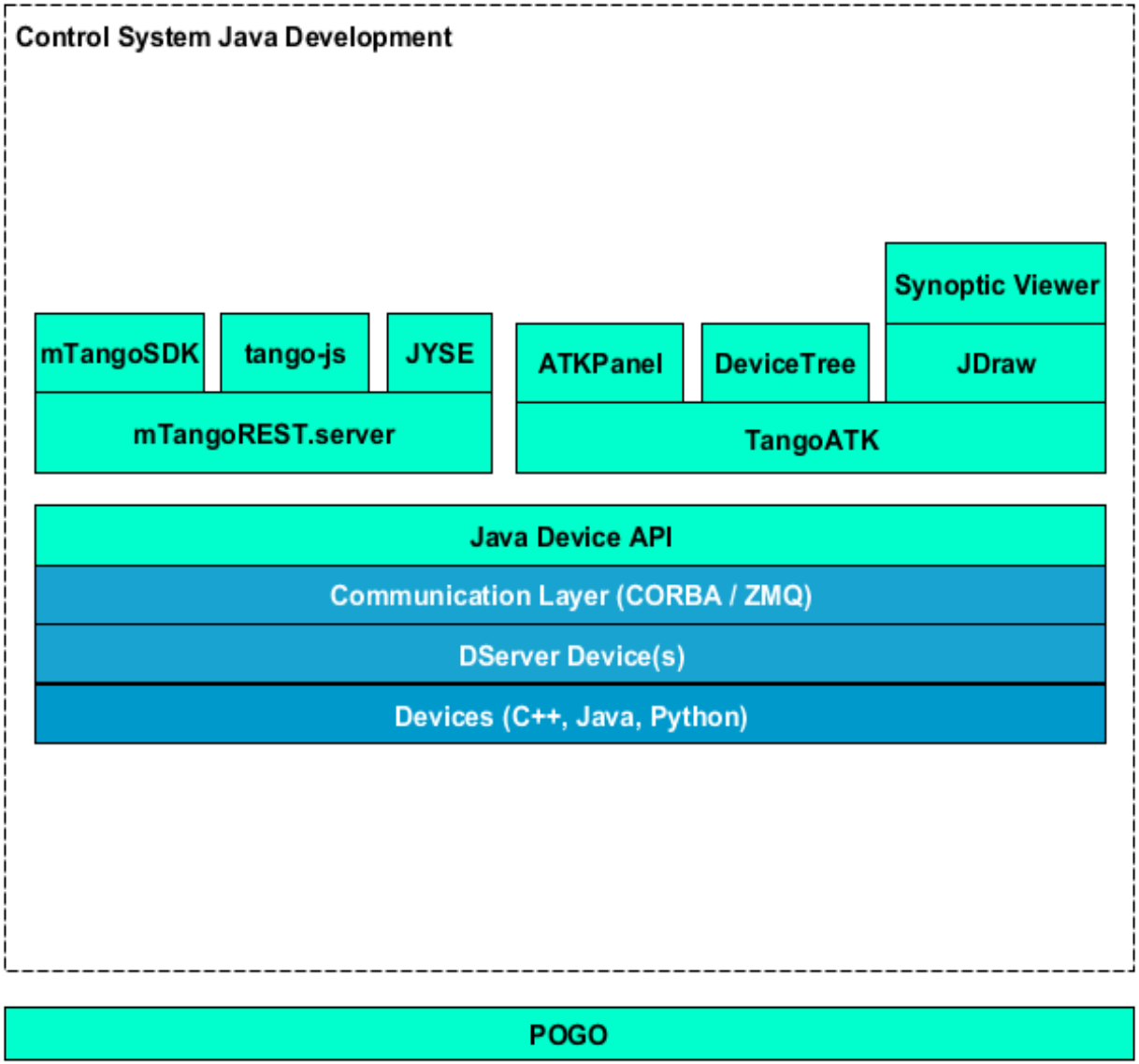
SERVER CLIENT API CONCEPT



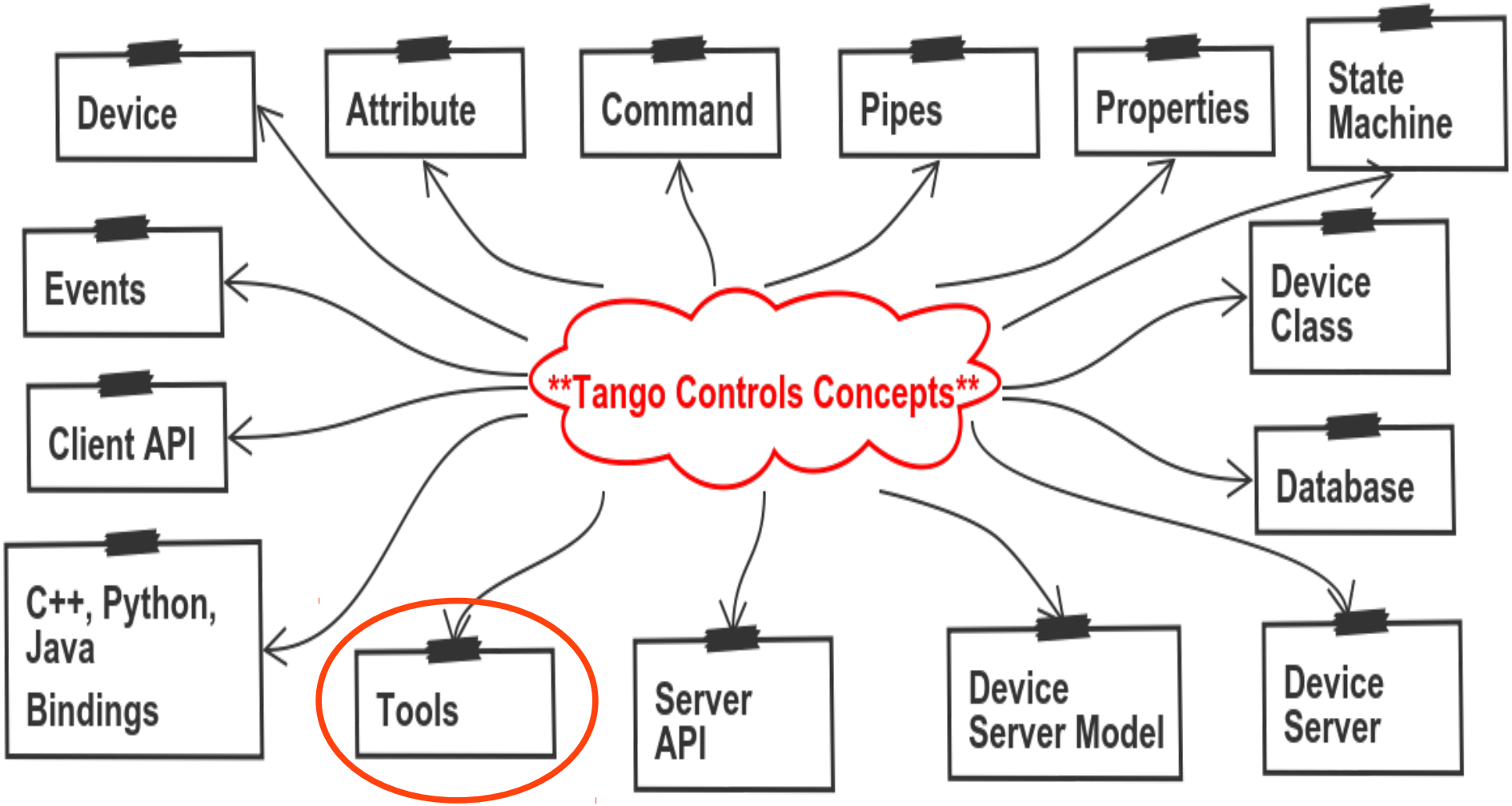
TANGO DEVELOPERS MAP



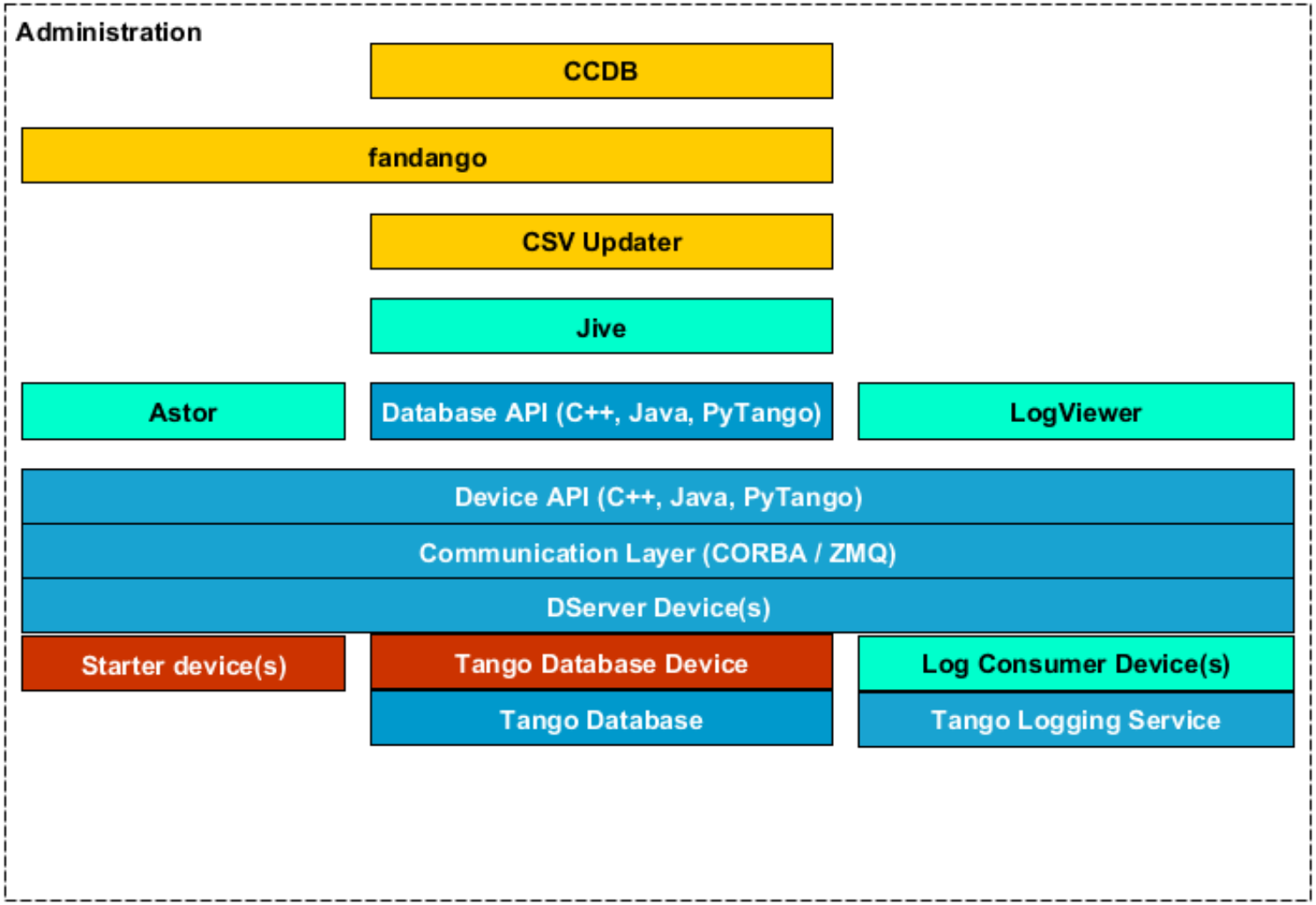
TANGO DEVELOPERS MAP



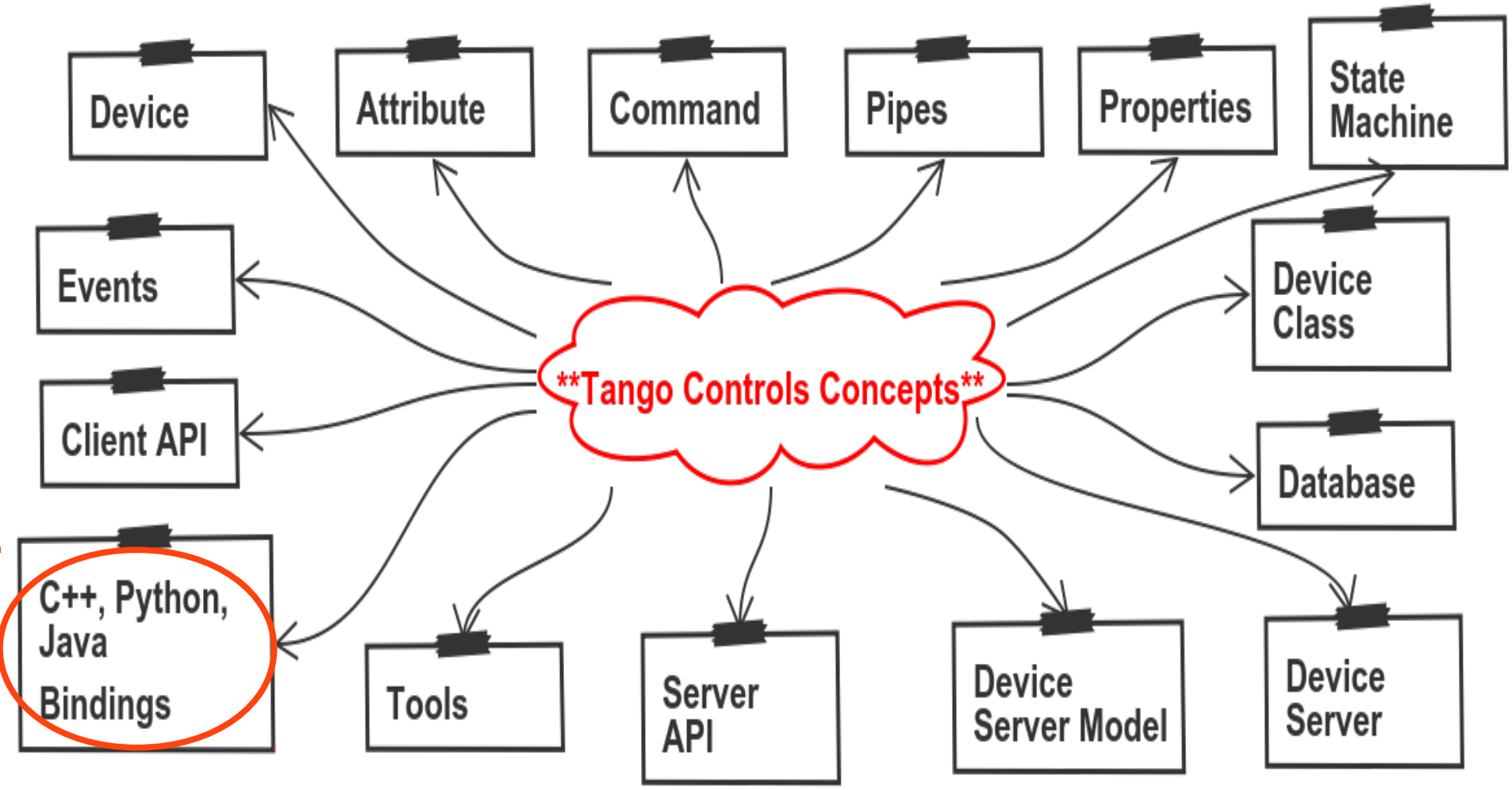
TOOLS CONCEPT



TANGO TOOLS MAP



BINDINGS CONCEPT



TANGO BINDINGS MAP