

TANGO CONTROLS CONCEPTS

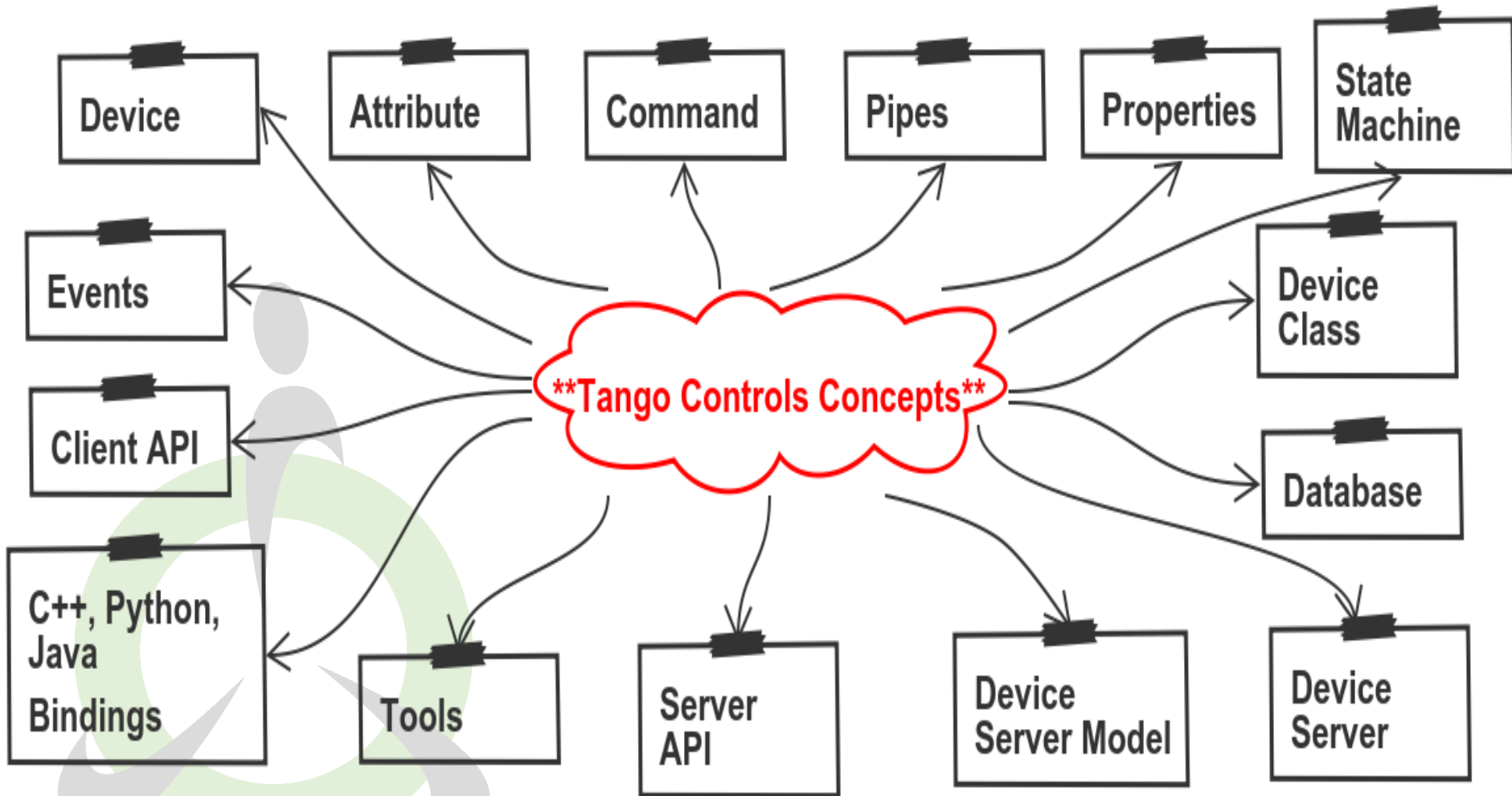
***A brief introduction to the
Tango Controls Concepts***

by

Andy Götz (vocals) +
Reynald Bourtembourg (keyboards)



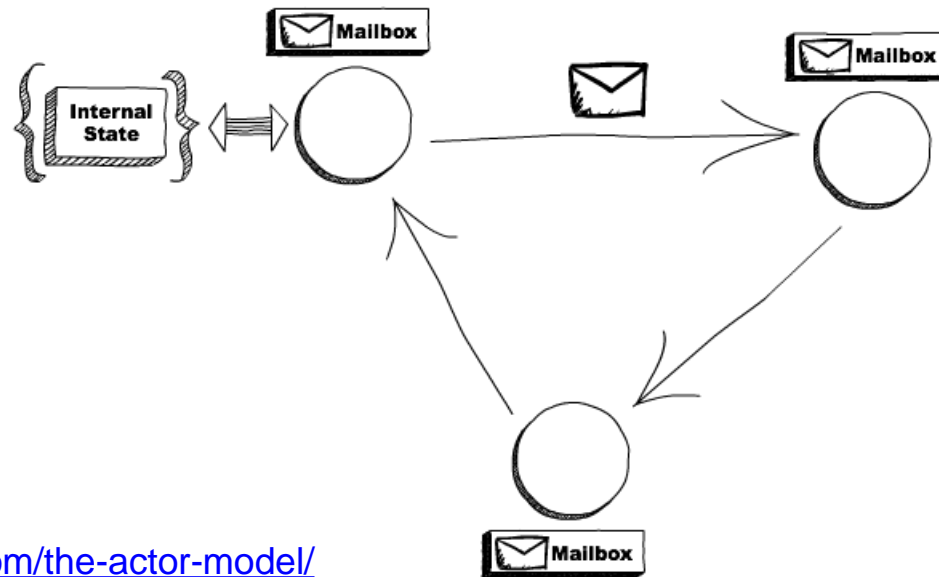
Tango basic concepts



Tango = actors + microservices

- Tango is based on the concept of **Distributed Devices**
- This is an implementation of the **Actor Model**
- Device servers implement **Microservices**
- **Tango = Actors + Microservices**
- **Actors + Microservices** are in fashion today
- **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

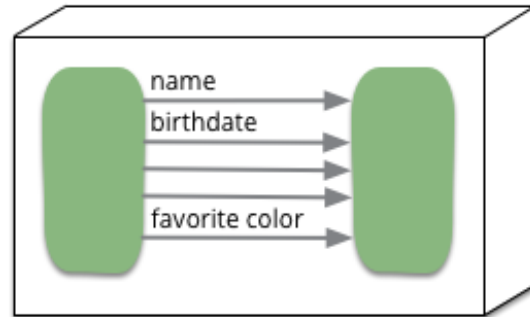
When I wrote [Patterns of Enterprise Application Architecture](#), I coined what I called the First Law of Distributed Object Design: "don't distribute your objects". In recent months there's been a lot of interest in [microservices](#), which has led a few people to ask whether microservices are in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

Martin Fowler

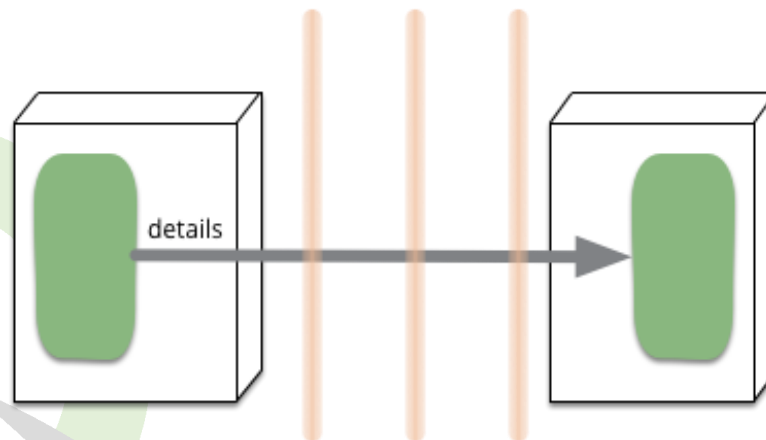


<https://martinfowler.com/articles/distributed-objects-microservices.html>

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.

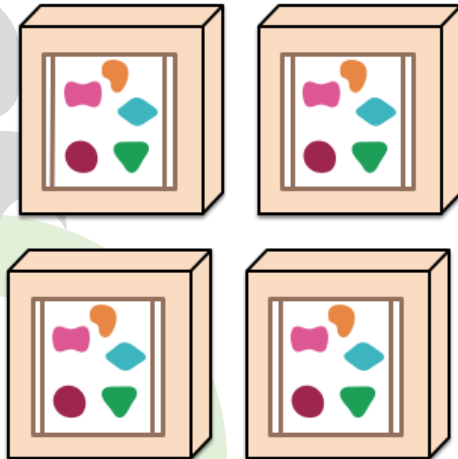
Tango implements Microservices not Distributed Objects !

Microservices

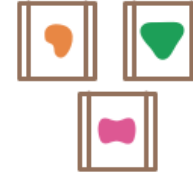
A monolithic application puts all its functionality into a single process...



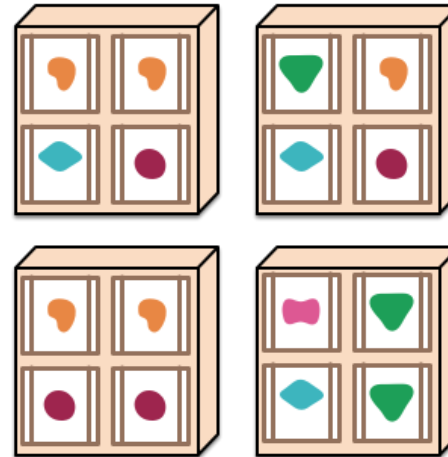
... and scales by replicating the monolith on multiple servers



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



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



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

Microservices by Microsoft



Orleans

[Documentation](#)

[Tutorials](#)

[Community](#)



Orleans

A straightforward approach to building distributed, high-scale applications in .NET

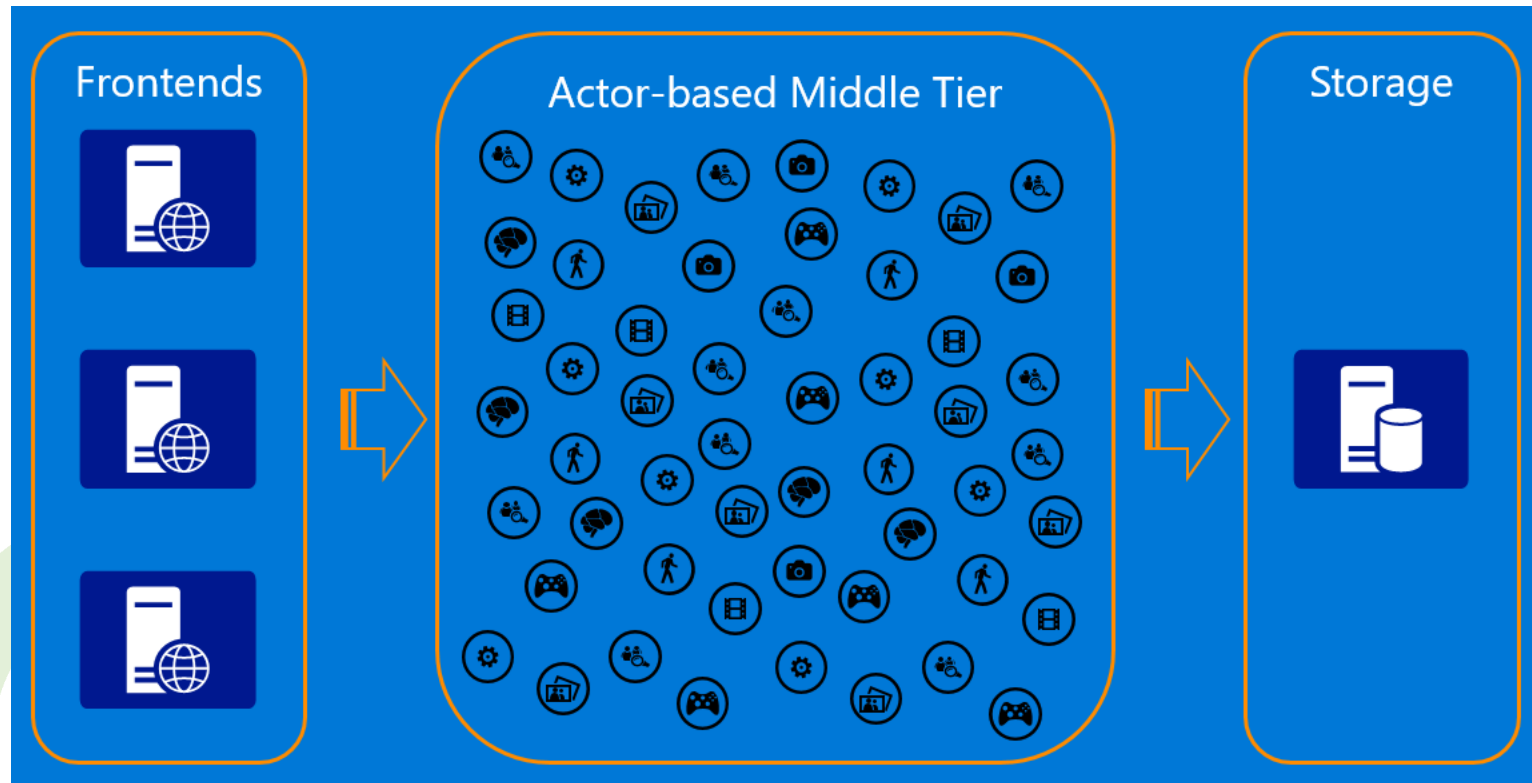
[Get the code](#)

[Read the documentation](#)

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.

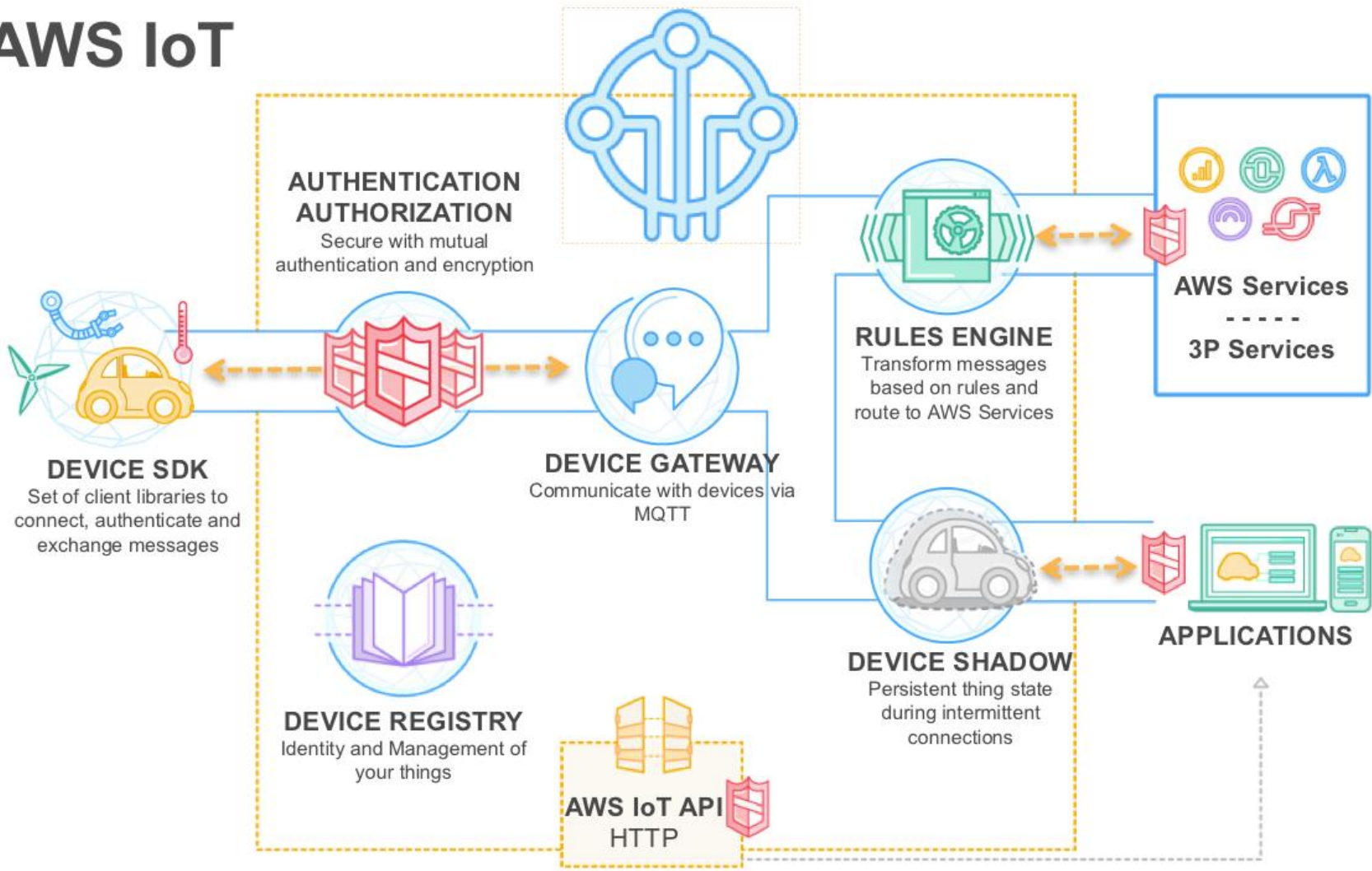
Orleans has been used extensively in Microsoft Azure by several Microsoft product groups, most notably by 343 Industries as a platform for all of Halo 4 and Halo 5 cloud services, as well as by a growing number of other companies.

Microservices by Microsoft



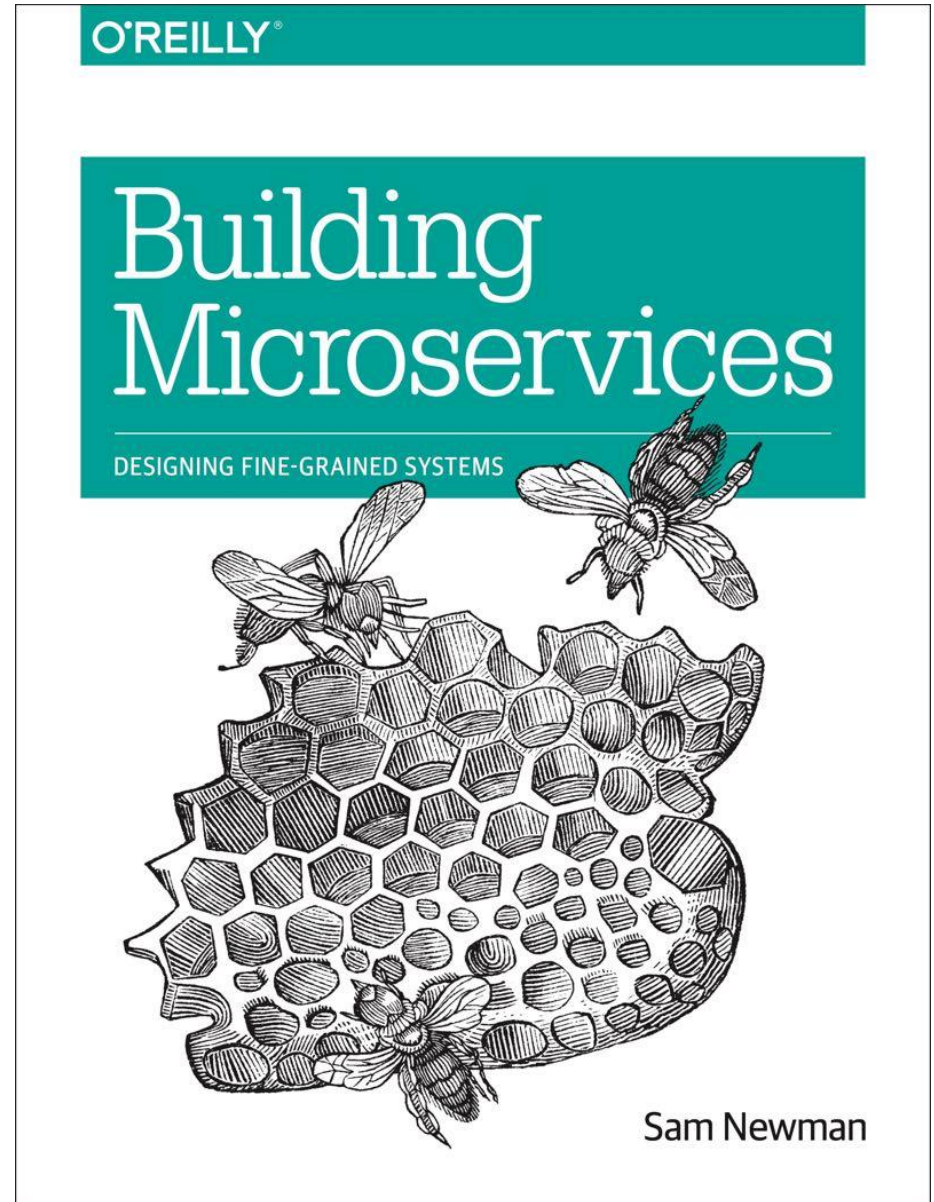
Microservices by AWS (but no Agents)

AWS IoT



Building Microservices

Sam Newman's book reads as if it was describing Tango Controls systems !



Tango Turing test



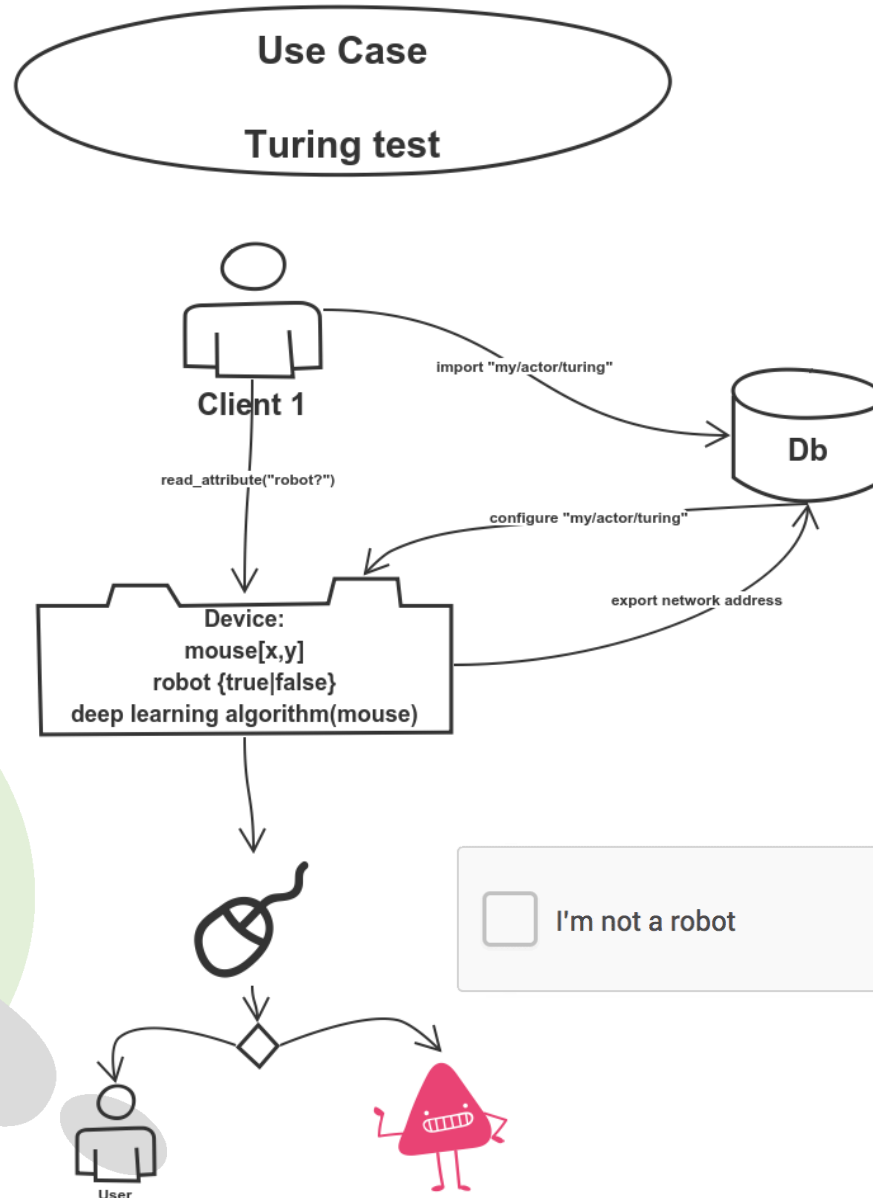
I'm not a robot



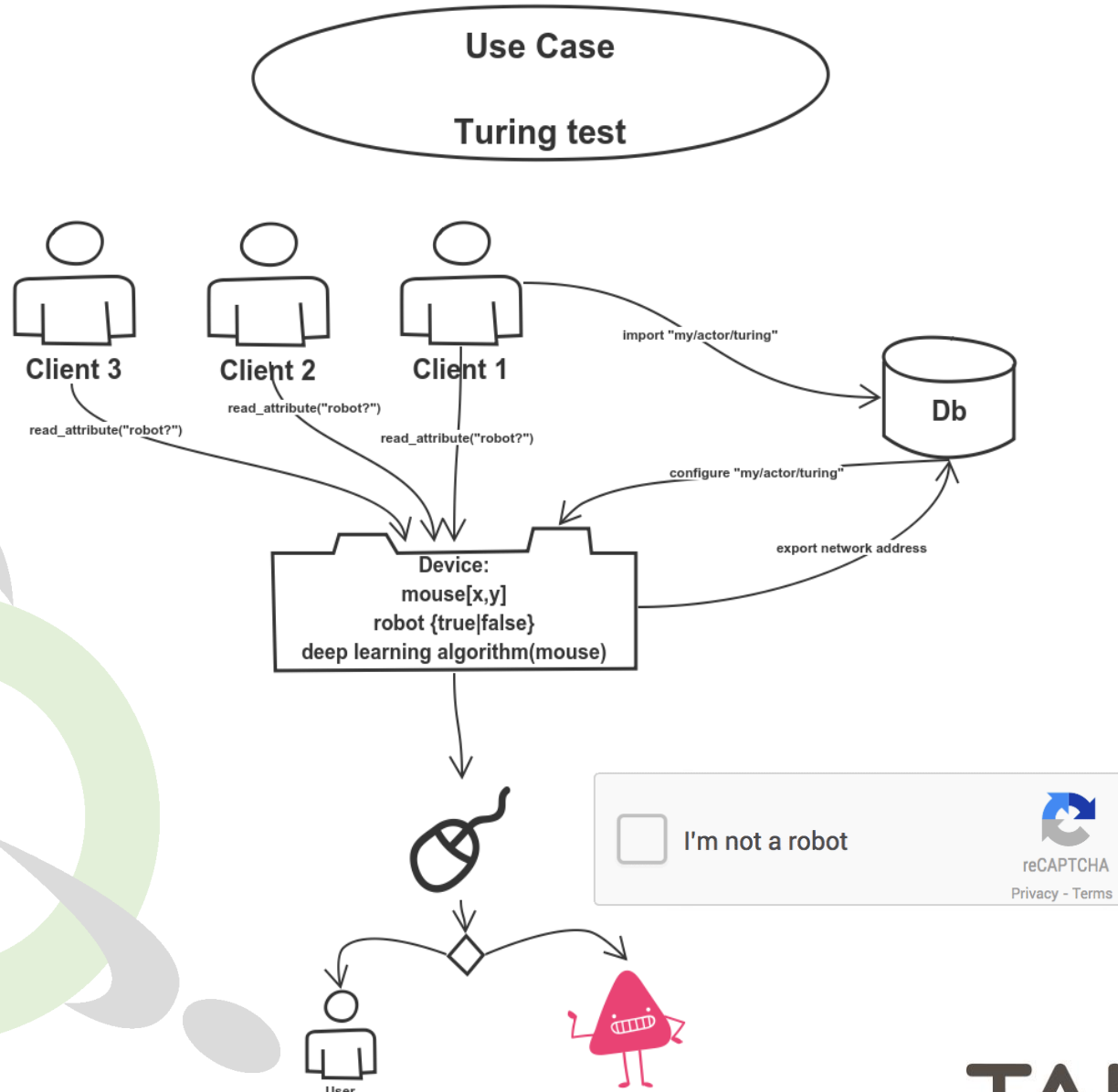
reCAPTCHA

[Privacy](#) - [Terms](#)

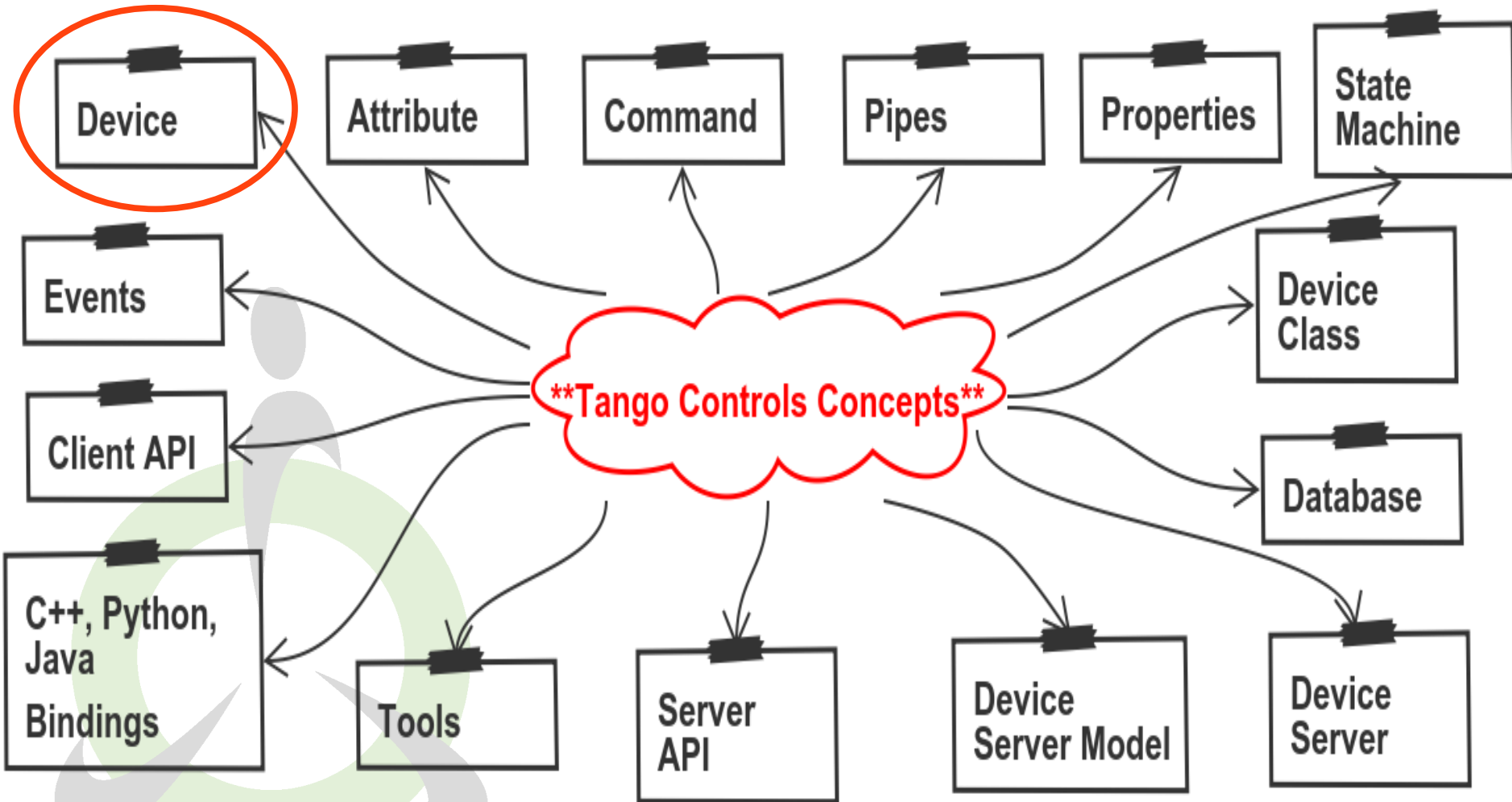
Tango Turing test



Tango Turing test - MULTIPLE clients



Device concept #1



Device concept #1

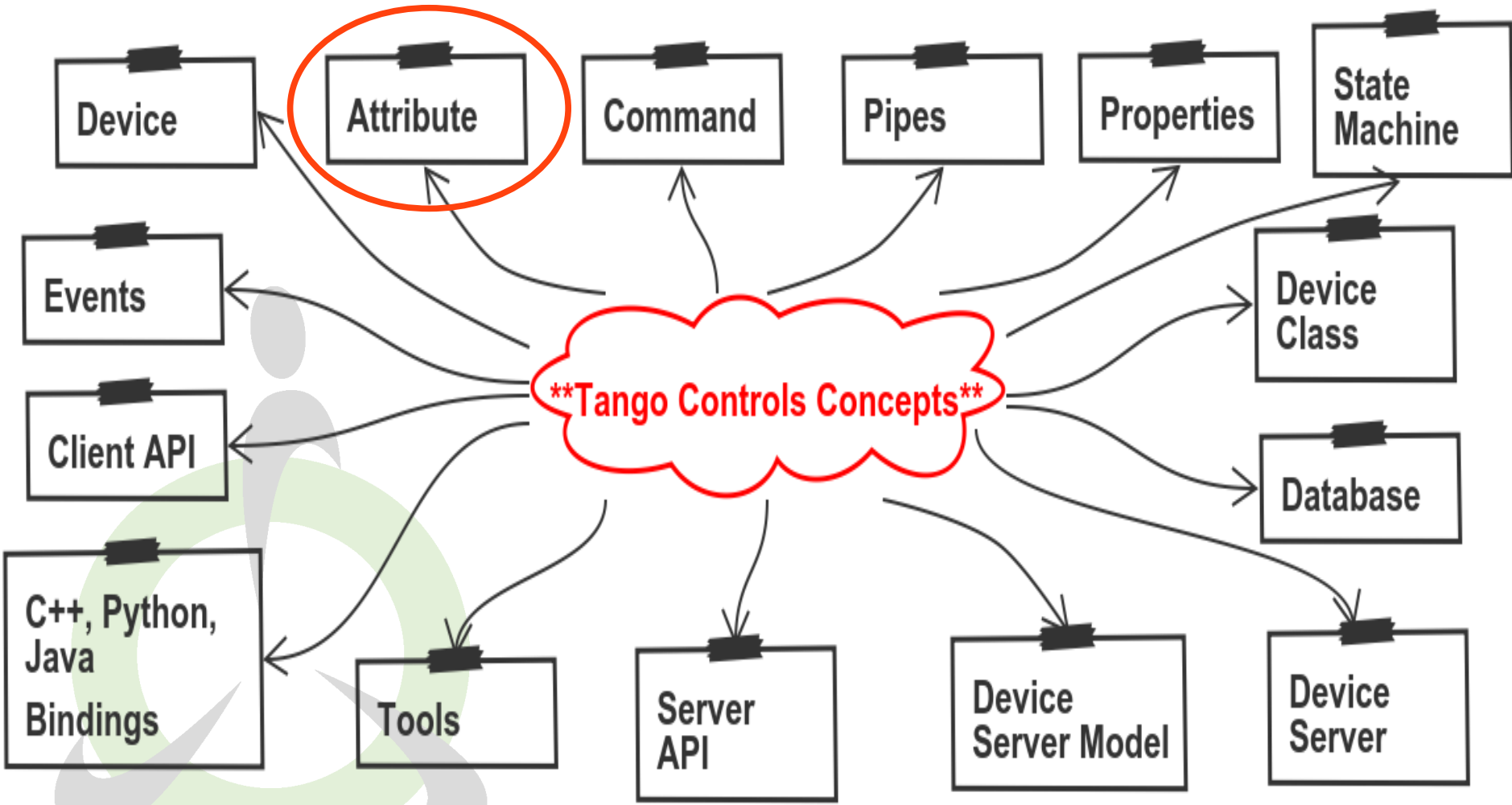
- Tango **Devices** are the objects which implement the microservices of a Tango System. Devices implement the **sensors** and **actuators**. Devices can be any piece of hardware or software.
- *Examples : Modbus controller, motor, powersupply, camera, data analysis service, ...*
- **Devices** belong 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
- Devices can be built on top of other Devices

Device thought experiment

- How would you decompose your system into **Devices**?
- What **naming convention** to use?
- How many **hierarchies** of Devices? Is there a limit?
- What are the equivalent of Devices in other systems e.g. EPICS, Spring Boot, your system ?
- How would you run a control system in the **cloud**?
- **QUESTIONS?**



Attribute concept #2



Attribute concept #2

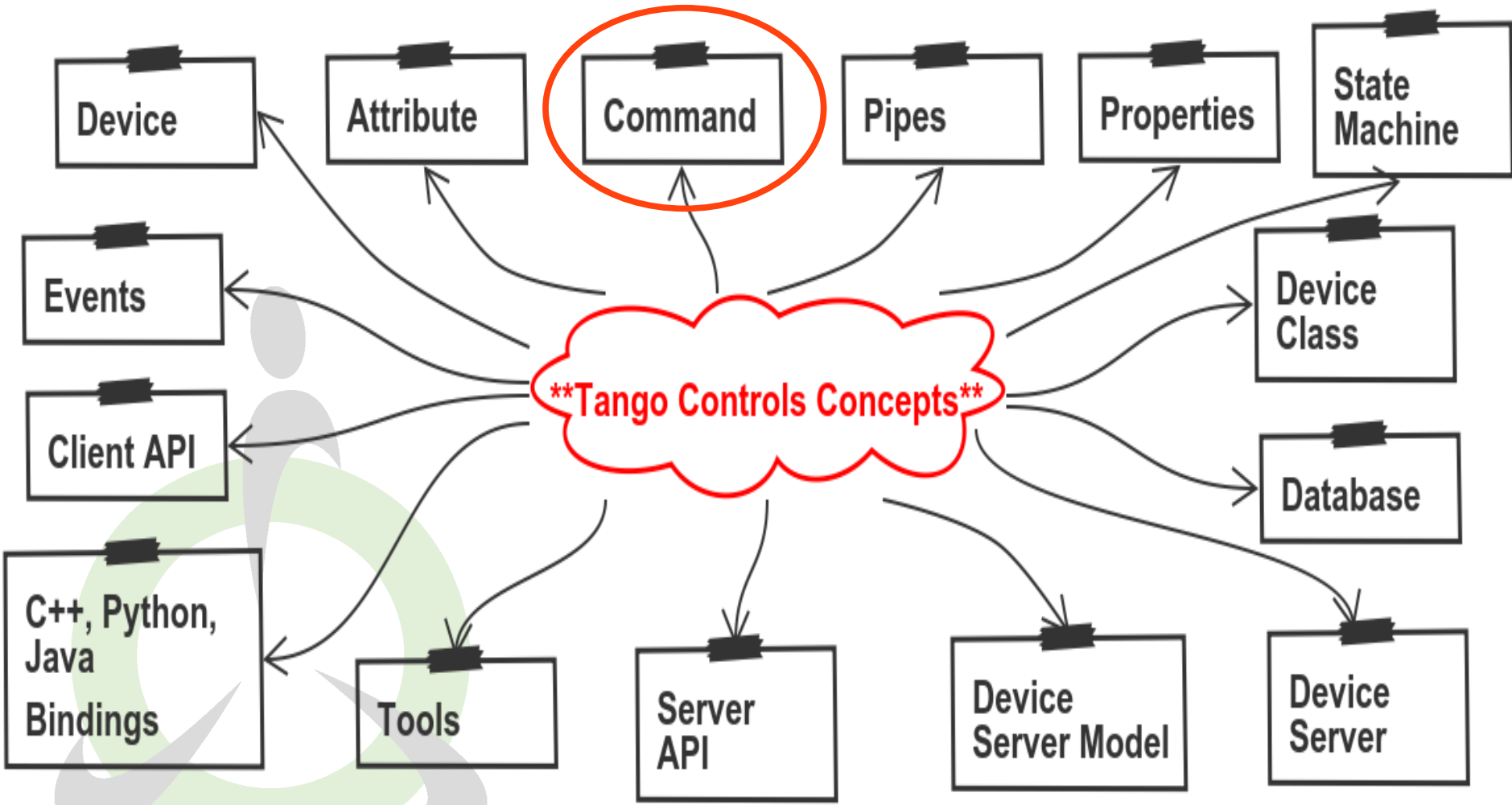
- Tango **Attributes** represents the data fields a Device wants clients to **Read** or **Write** or receive **Events**.
- *Examples : modbus register, interlock bit, read/set value, spectrum, image, ...*
- Attributes can be **scalar**, **spectrum** (1D) or **images** (2D) and are **self describing** (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 read/write check the **State Machine**.

Attribute thought experiment

- What **Attributes** would you implement? Number? Size?
- What are the **limitations** of attributes?
- How would you name your Attributes?
- Would it be useful to have serverless Attributes?
- **QUESTIONS?**



Command concept #3



Command concept #3

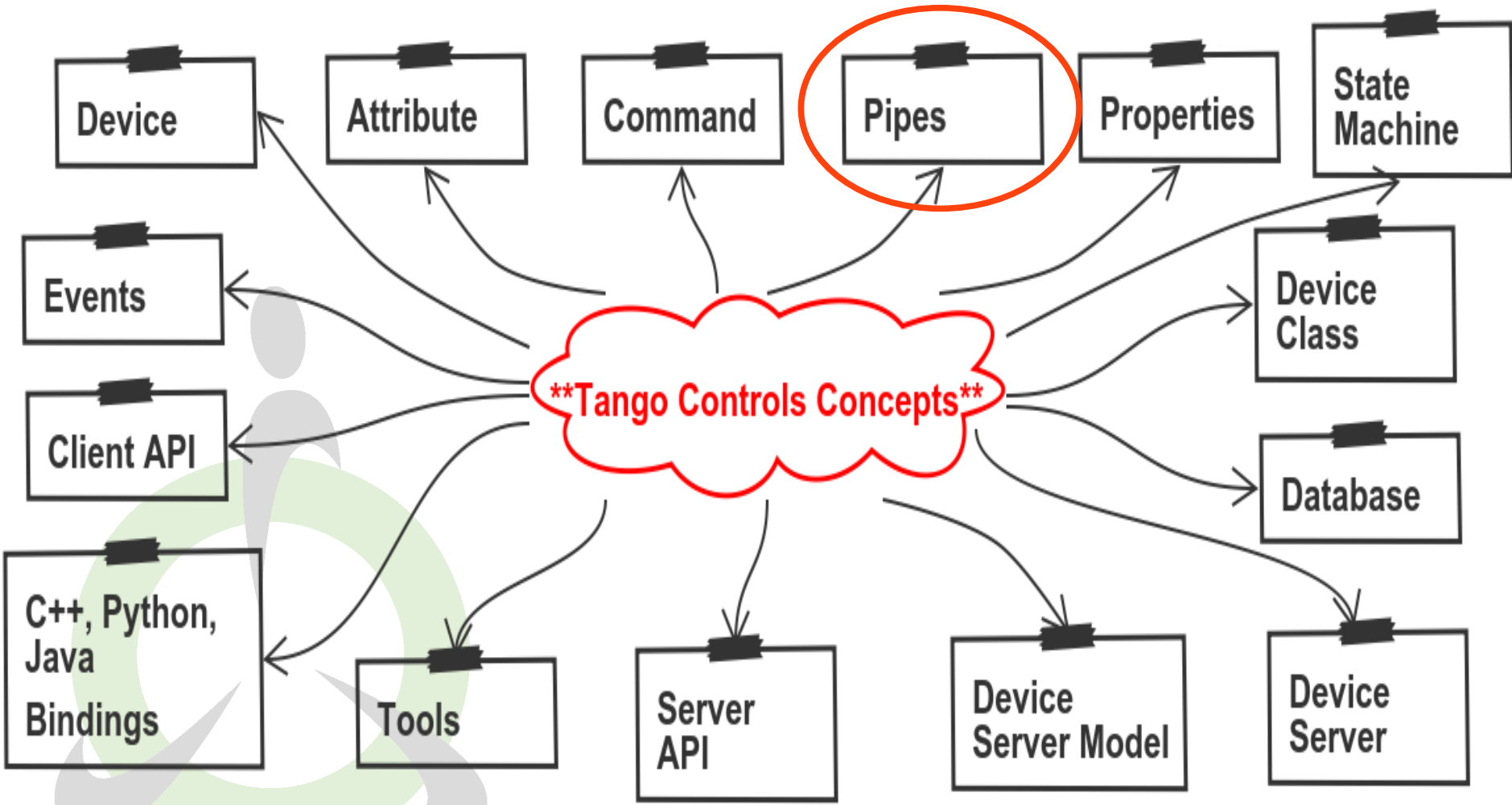
- Tango **Commands** are the actions of a Device the clients needs to execute. Commands can change the State of a Device (Attributes don't)
- *Examples : On, Off, Calibrate, Move, ...*
- Commands take **one input** and **one output parameter**. Parameters can be of any of the 20+ Tango data types.
- Commands always check the **State Machine** before and after execution (Attributes only before).

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?
- **QUESTIONS?**



Pipe concept #4



Pipe concept #4

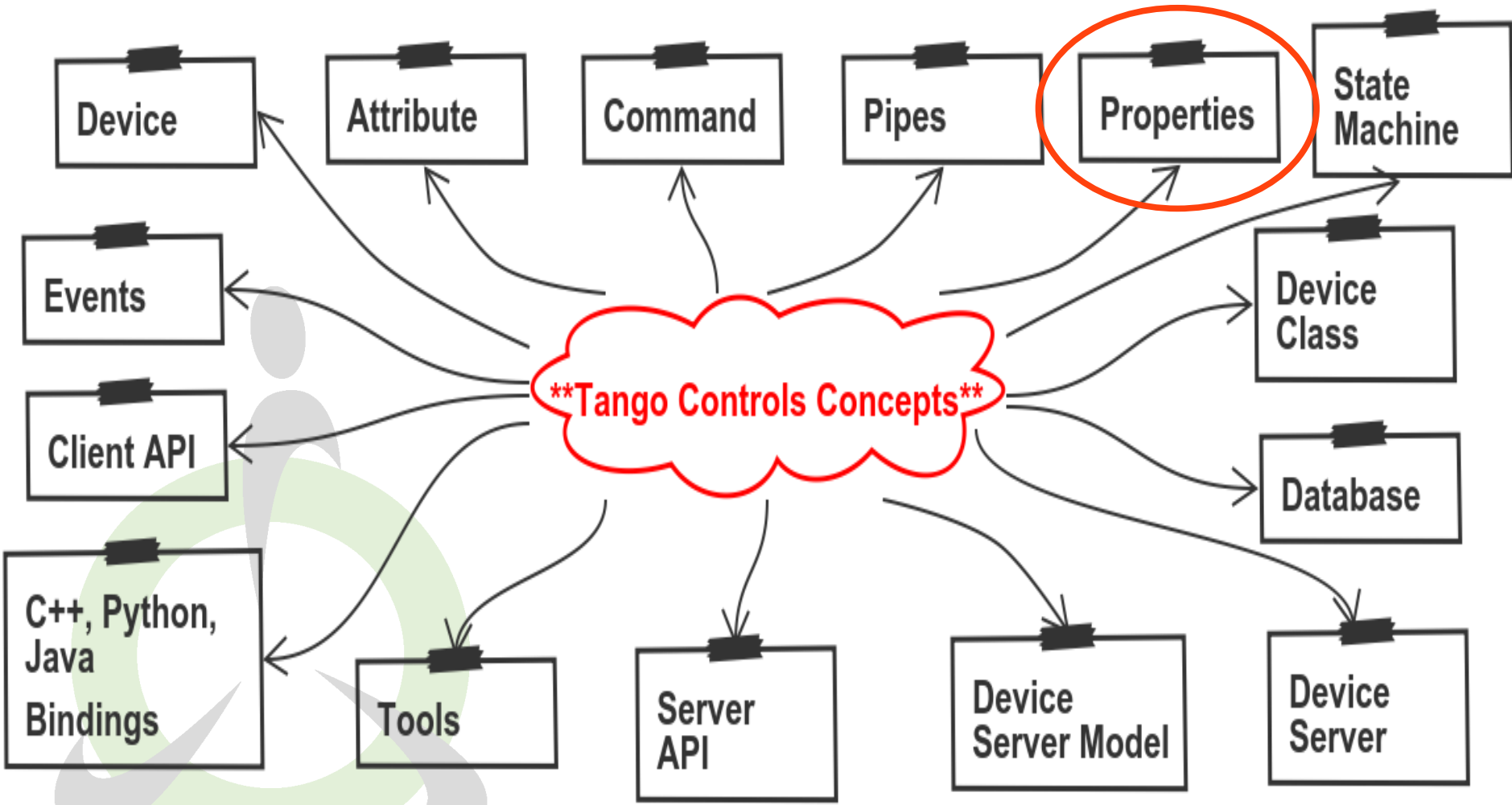
- Tango **Pipes** are data streams or channels for exchanging a stream of any Tango data type. Data types can be sent individually or grouped together in a Blob.
- *Examples : scanning data stream of mixed data types*
- Also used to circumvent the fixed data type set of Tango by sending mixed data types or a JSON blob.
- **DO NOT** only use Pipes (except in special cases)! Pipes were added to Tango in V9.x
- **DO** use Attributes!

Pipe thought experiment

- Where do you need **Pipes** in your system?
- What will you put in your **Pipe(s)**?
- Do you need JSON data types?
- Connect Pipes to Kafka?
- **QUESTIONS?**



Properties concept #5



Properties concept #5

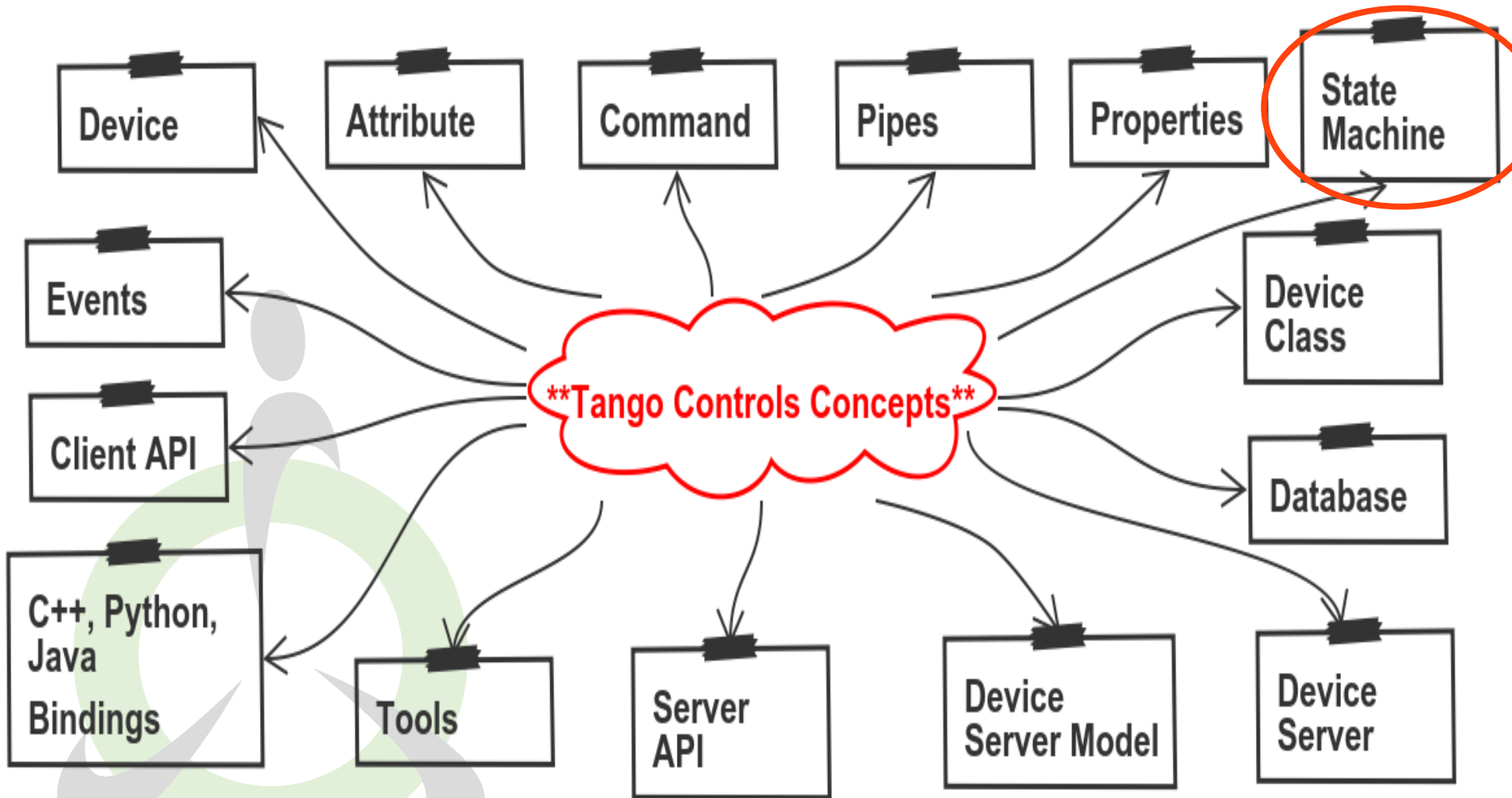
- Tango **Properties** are data stored in the database and used to configure Devices at startup. Properties can be any Tango Data types. Properties enable Device Classes to be generic. Properties are edited with Jive usually.
- *Examples : channel address, initial or current settings, sub-device names, ...*
- 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?
- Properties can be for Device Classes, Devices, free.
- Complex properties are not easy to edit with Jive
- **QUESTIONS?**



State machine concept #6

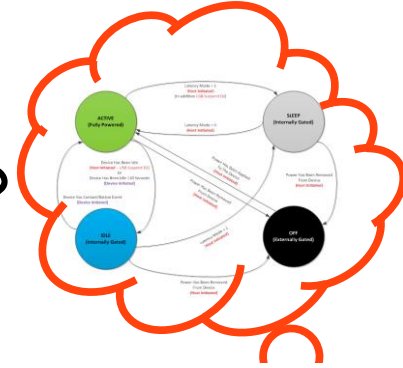


State machine concept #6

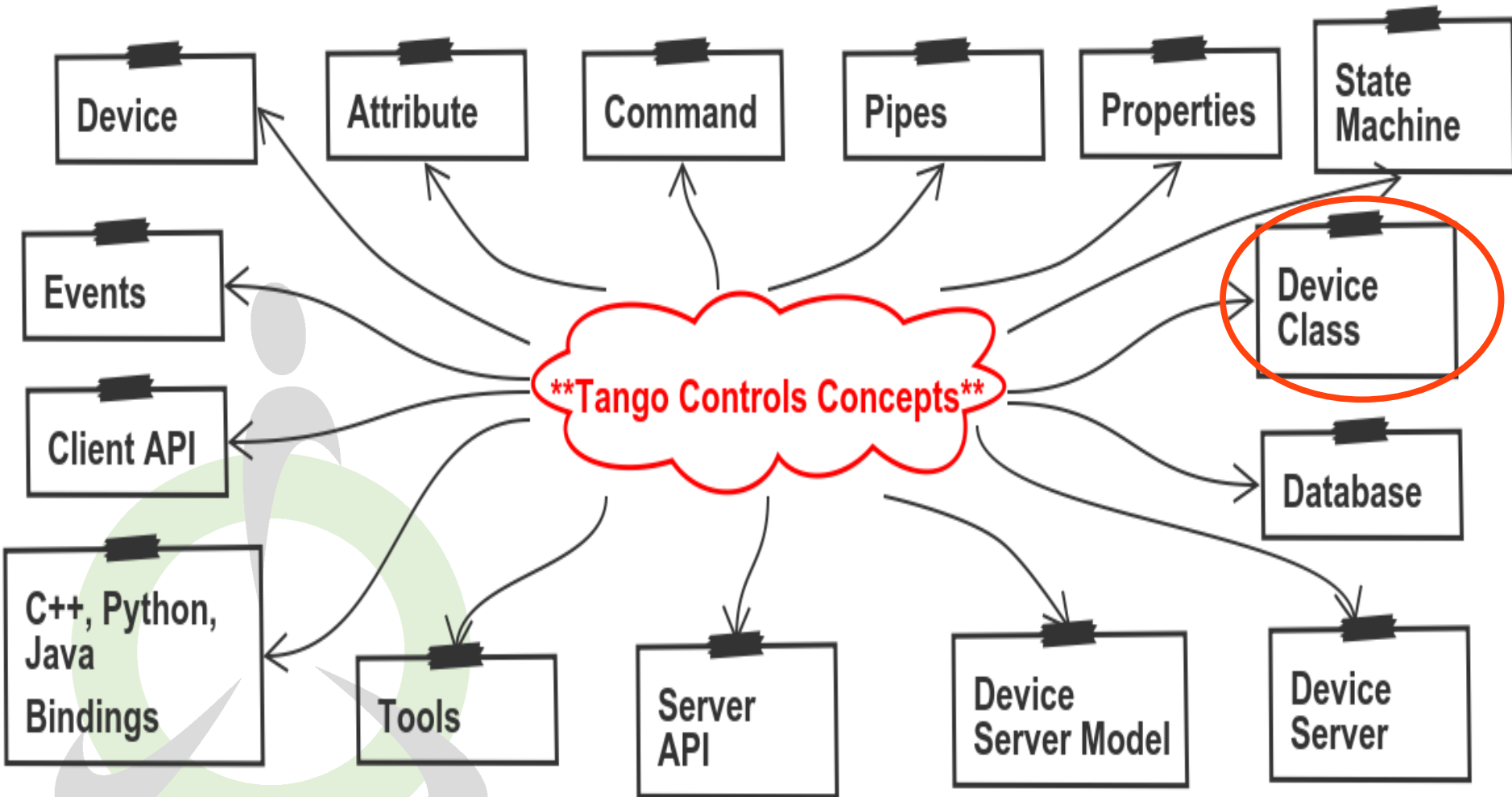
- All Tango Devices have State. Tango States are limited to 14 discrete values. Each Tango Device Class **State Machine** implements the state transitions.
- **ON, OFF, CLOSE, OPEN, INSERT, EXTRACT, MOVING, STANDBY, FAULT, INIT, RUNNING, ALARM, DISABLE, and UNKNOWN**
- State is a very powerful mechanism 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?
- **QUESTIONS?**



Device class concept #7



Device class concept #7

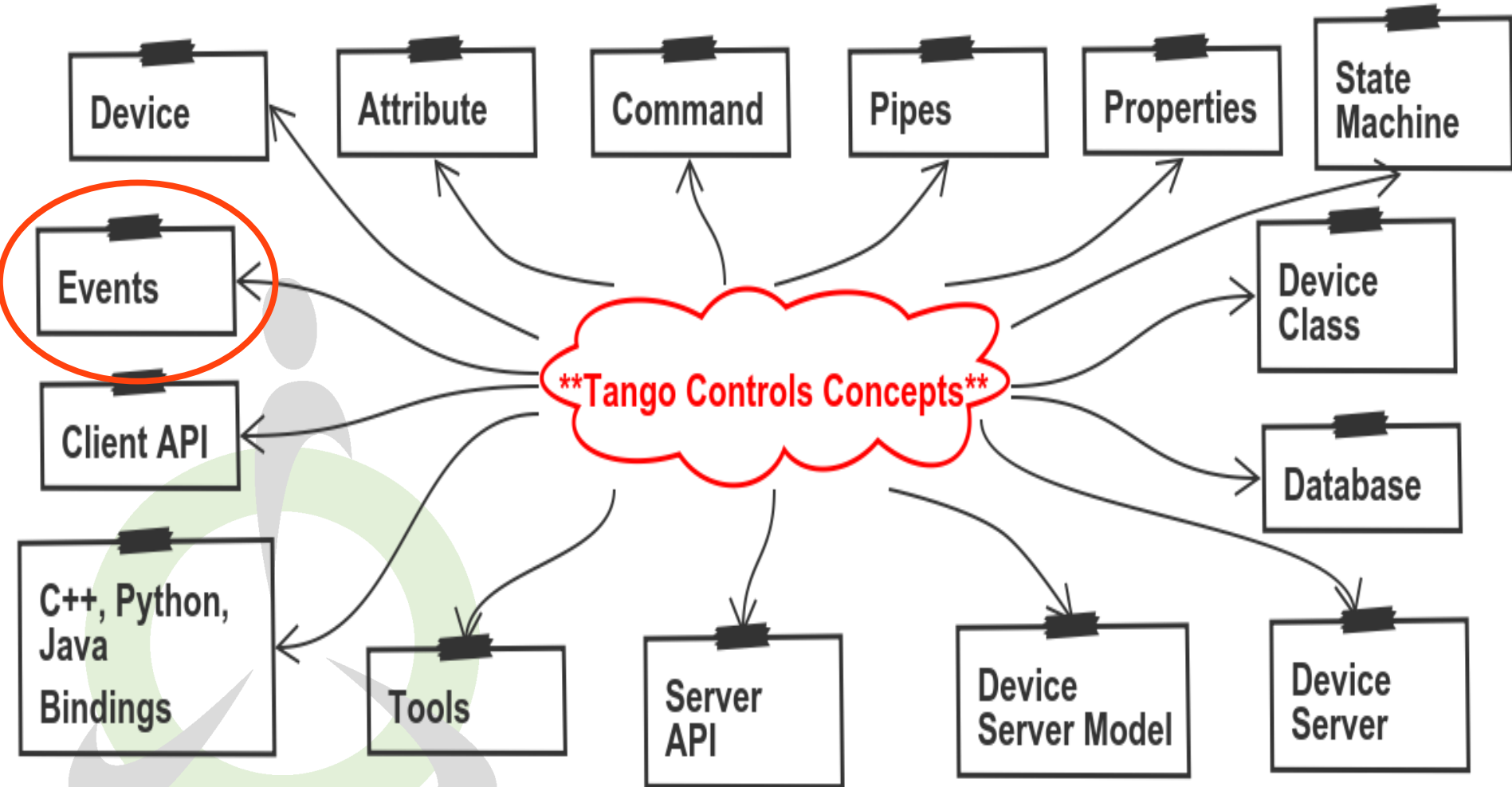
- All Tango Devices are implemented by a **Device Class**. The Device Class implements a generic Device behaviour. Properties are used to configure the specific Device
- *Examples : PowerSupply, SerialLine, Polly*
- Device Server developers are in fact developing Device Classes
- C++ developers have an extra class to develop - the so-called **DeviceClassClass** e.g. MyPowerSupplyClass. This uses one of the Gang of Four patterns. Python and Java have only the DeviceClass.

Device class thought experiments

- How many Tango Device Classes do you need?
- Share your classes on GitHub / GitLab!
- To **POGO** or not to **POGO** that is the question?
- Find classes in the Device classes catalogue
- There is not too much sharing of classes - too easy to write your own?
- How to package Device classes - Debian, Docker, Conda, ...
- **QUESTIONS?**



Events concept #8

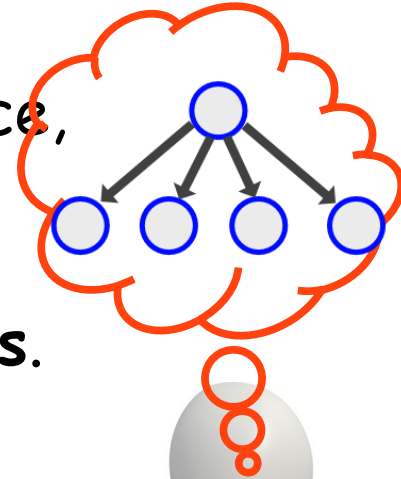


Events concept #8

- 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 **ZeroMQ** + are the most **efficient** way to communicate. Events rely on **Polling** to be triggered.
- **Events** are configured in the database or code
- Tango implements a Polling algorithm to trigger events.
- **DO NOT** only read Attributes, use Events

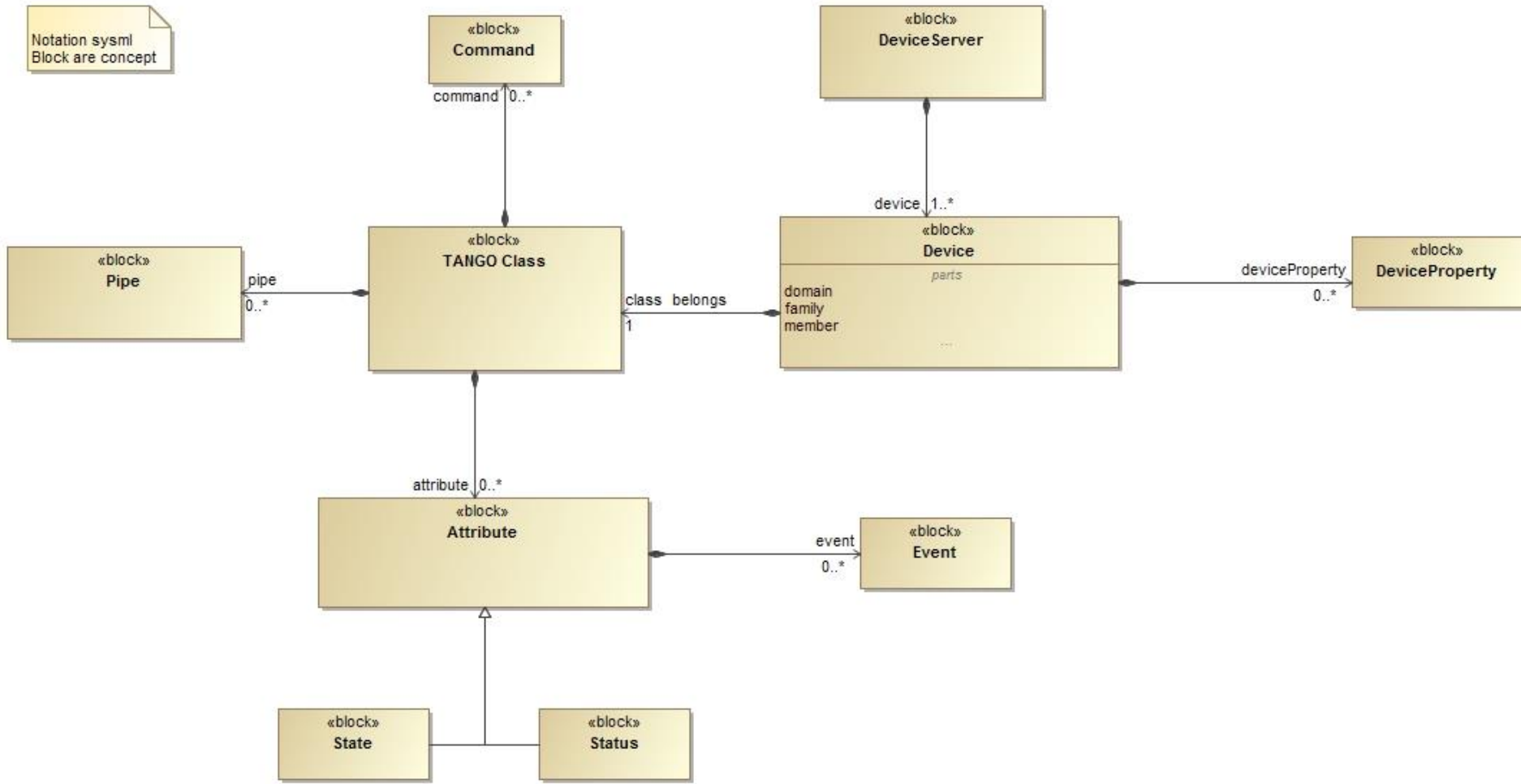
Events thought experiments

- Have you understood how **Polling** triggers **Events**?
- Multiple polling threading models - per Device, Poll single or multiple attributes
- Understand standard **Events** vs. **User Events**.
- Documentation describes multicast Events. No-one is using multicast!
- Events performance scale per device server and inversely per client (see paper by Michal + Piotr)
- **QUESTIONS ?**

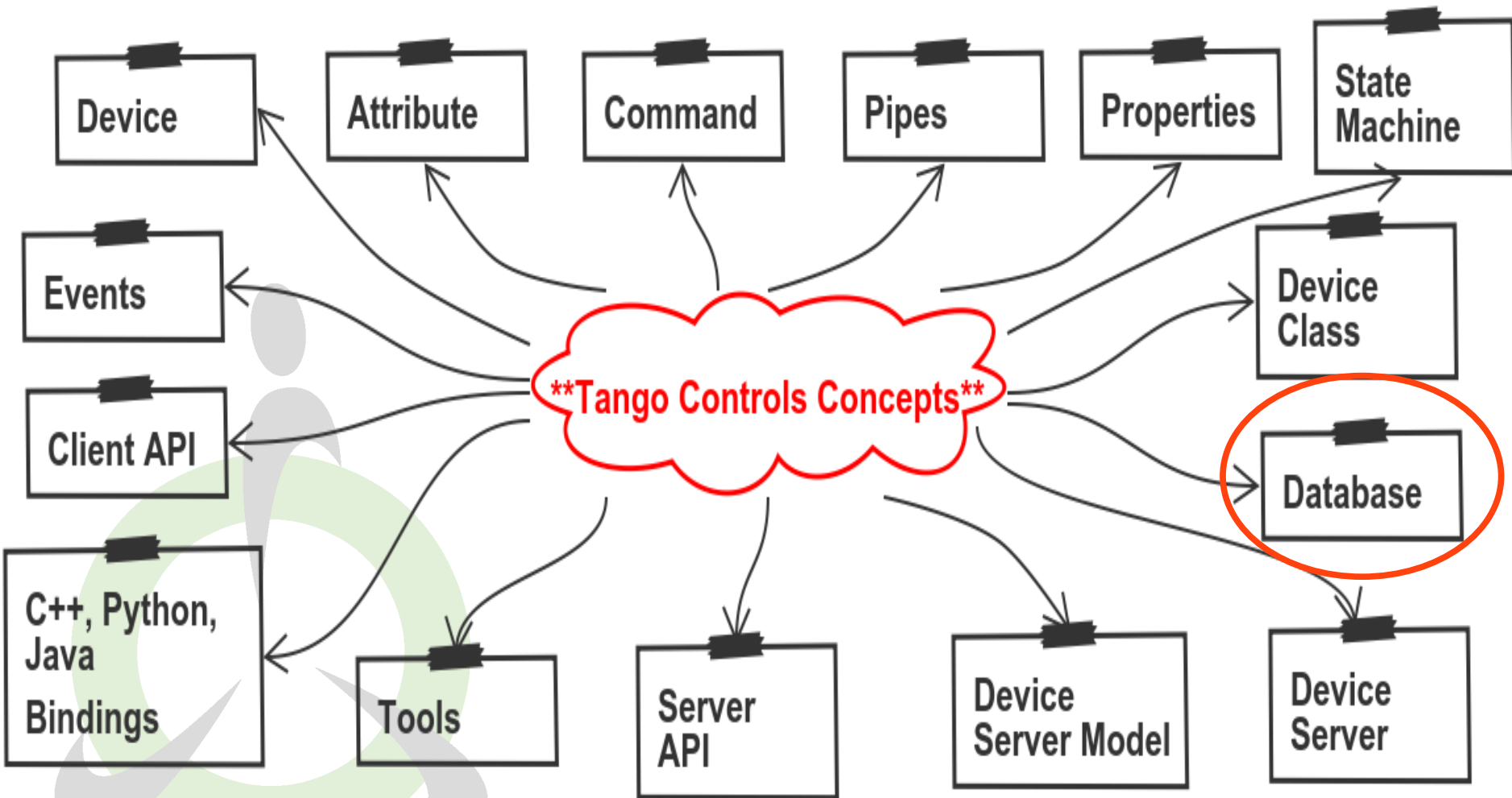


Tango simple Device Model (concepts #1 to #8)

Notation sysml
Block are concept



Database concept #9

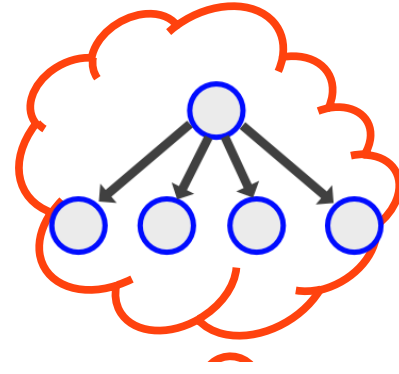


Database concept #9

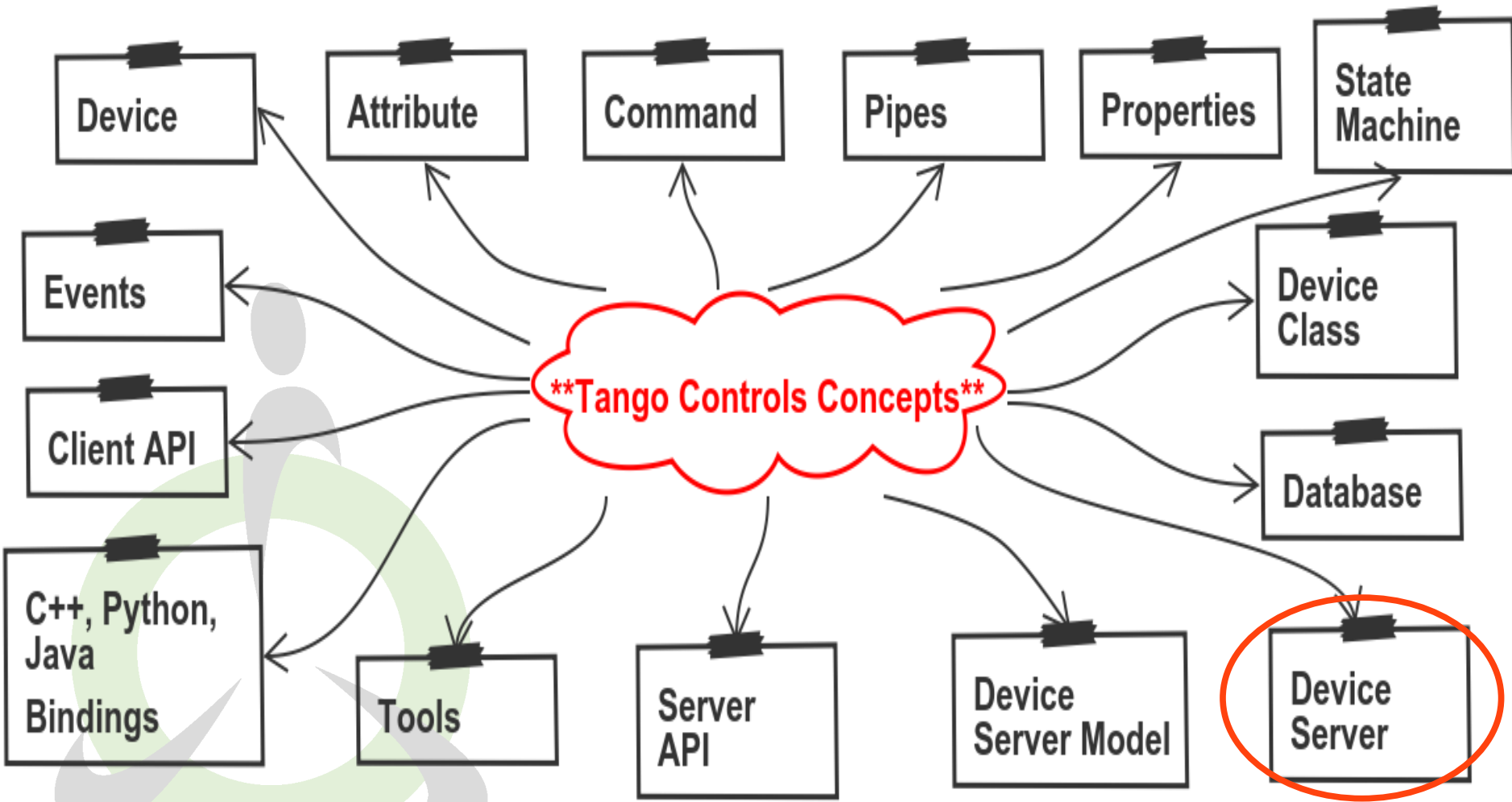
- Tango **Database** implements the **Configuration and Naming Service** for Tango. It can also persists settings values.
- *Examples : configuration properties, export/import*
- Tango Database is implemented as a Device Server. Clients use the Tango Client API and Data Types to access the Database. Only MySQL is supported. A (non-official) version exists for SQLite + yaml.
- Database is only fixed address **TANGO_HOST=host:port** or **/etc/tango.rc** environment variable.
- Multiple Databases supported.

Database thought experiment

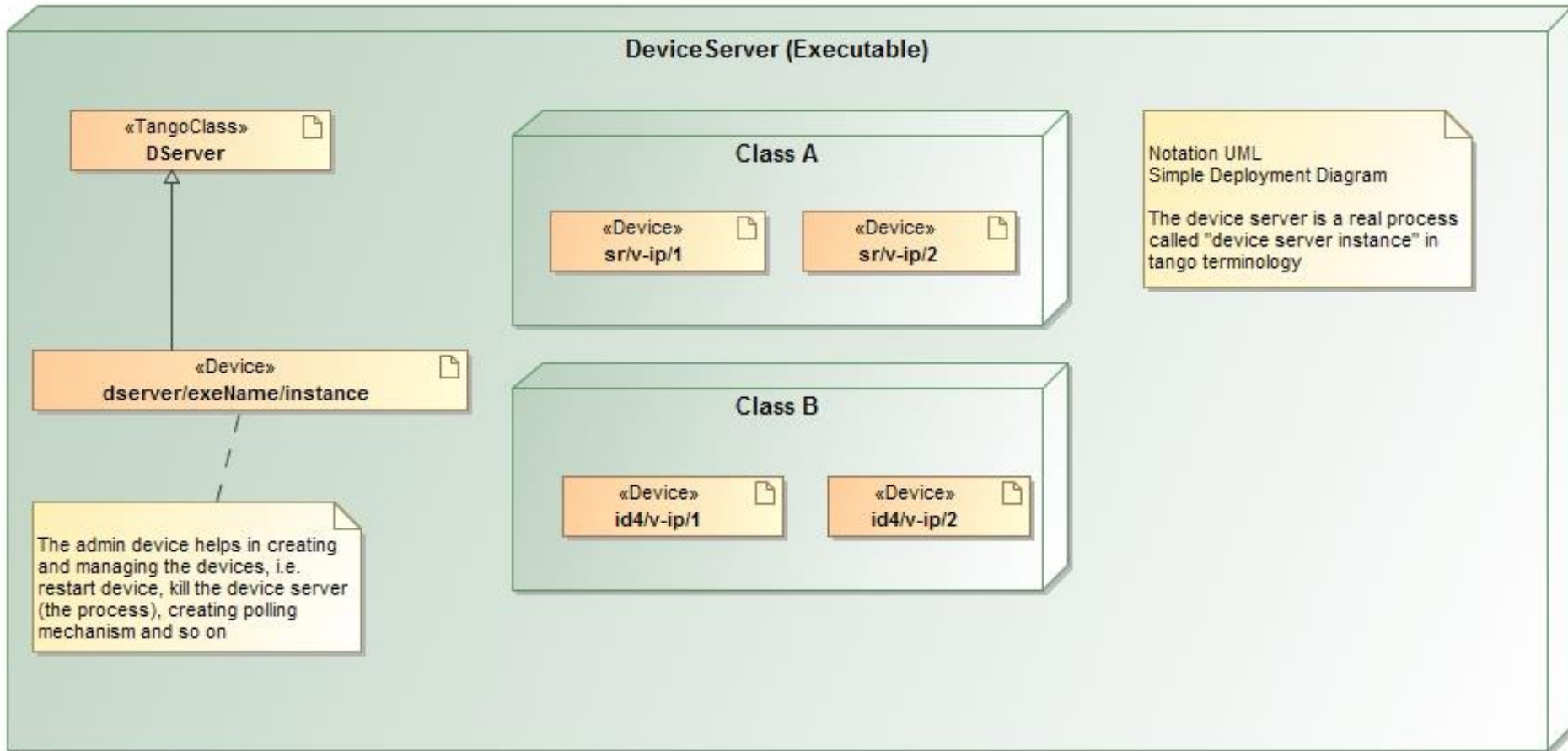
- Have you understood how multiple Database support systems of systems?
- Is the Database a Single Point of Failure? Multiple schemes to reduce the single point of failure (see Tango RTD)
- Who needs in a non-MySQL version of the Database?
- **QUESTIONS?**



Device server concept #10

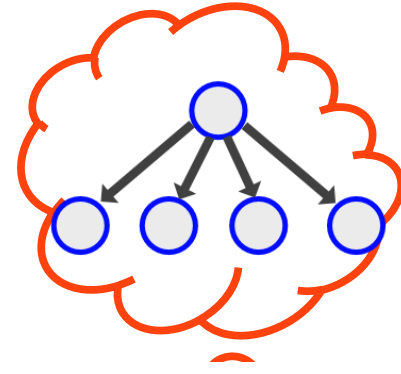


Tango runtime Device Model

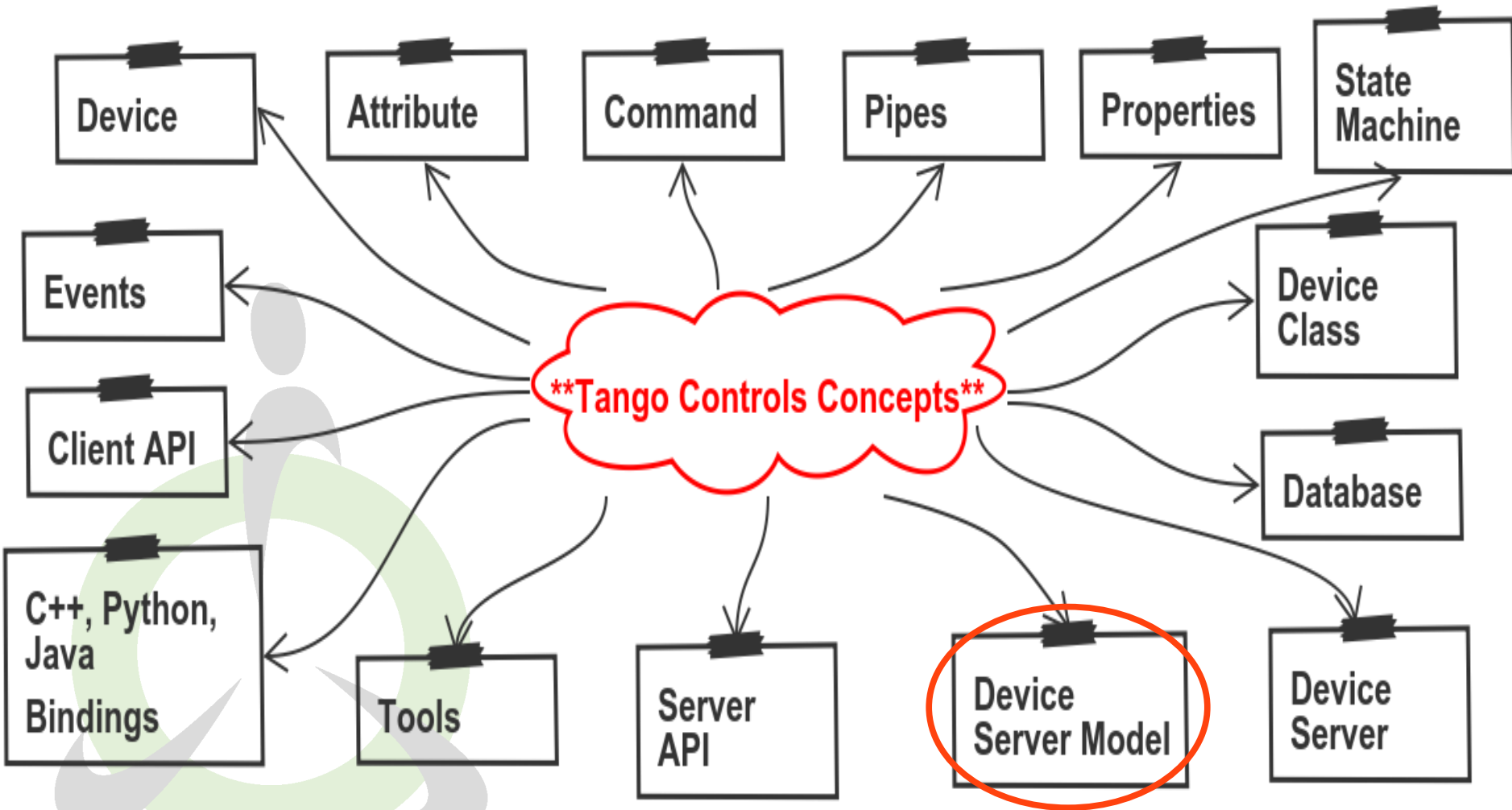


Device Server thought experiment

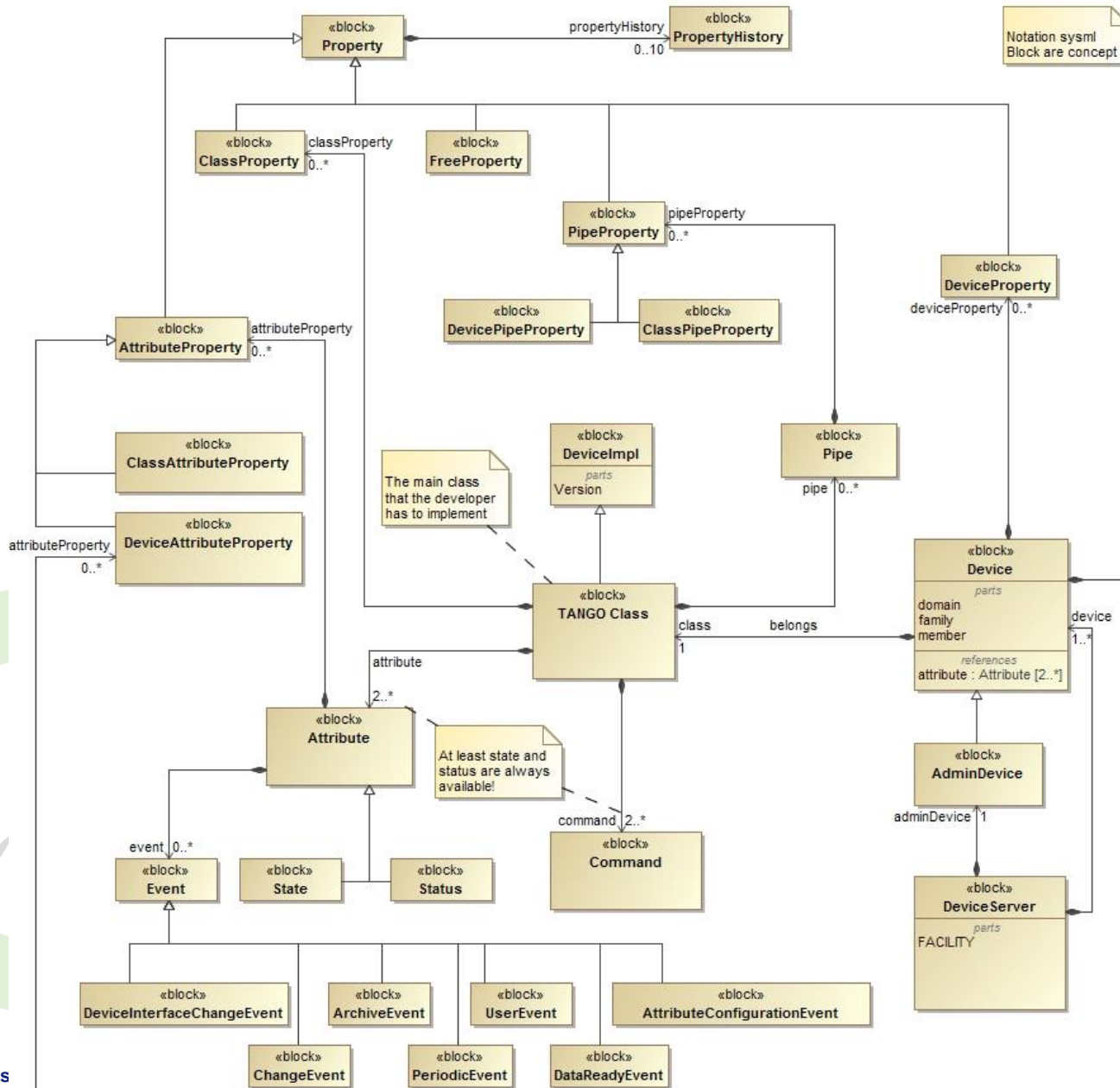
- Important to understand difference between **Device vs. Device Server vs. Device Server Instance Name vs. Device Server process?**
- Probably the most difficult concept but which gives Tango its flexibility to adapt to large numbers of Devices
- Is this similar to an IOC ?
- QUESTIONS?



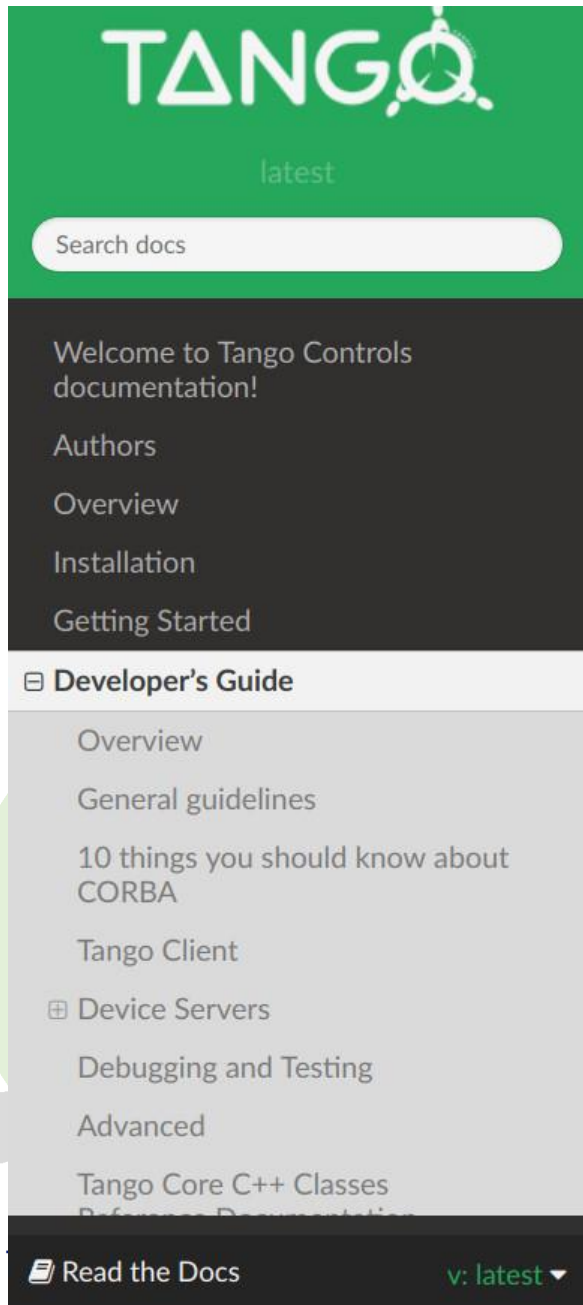
Device server model concept #11



Tango full Device Model



Tango Device Server developers guidelines



TANGO
latest

Search docs

Welcome to Tango Controls documentation!

Authors

Overview

Installation

Getting Started

☰ **Developer's Guide**

- Overview
- General guidelines
- 10 things you should know about CORBA
- Tango Client
- ☰ Device Servers
 - Debugging and Testing
 - Advanced
- Tango Core C++ Classes
- Performance Documentation

📖 Read the Docs v: latest ▾

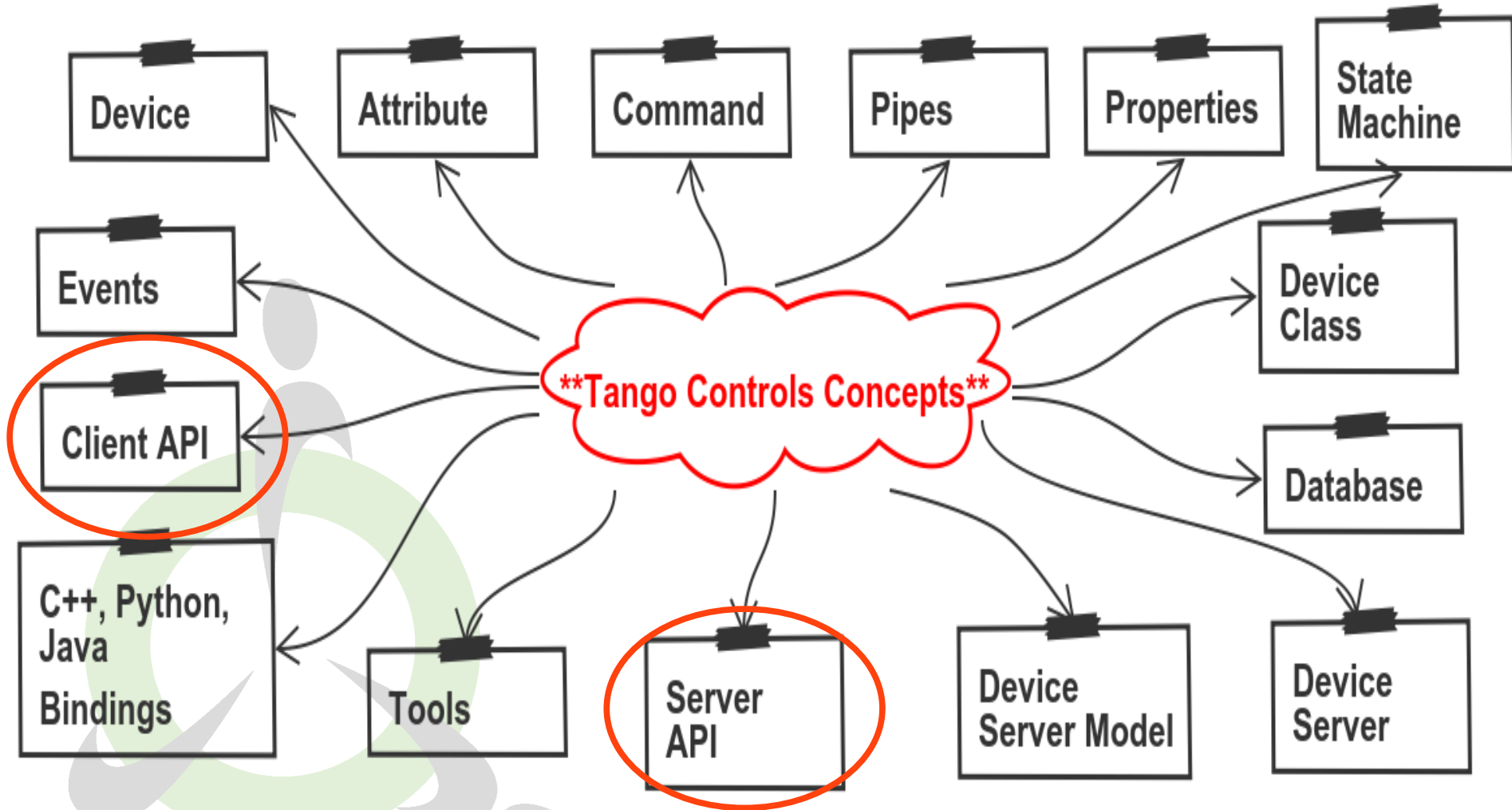
TANGO Device Server Guidelines

Contents:

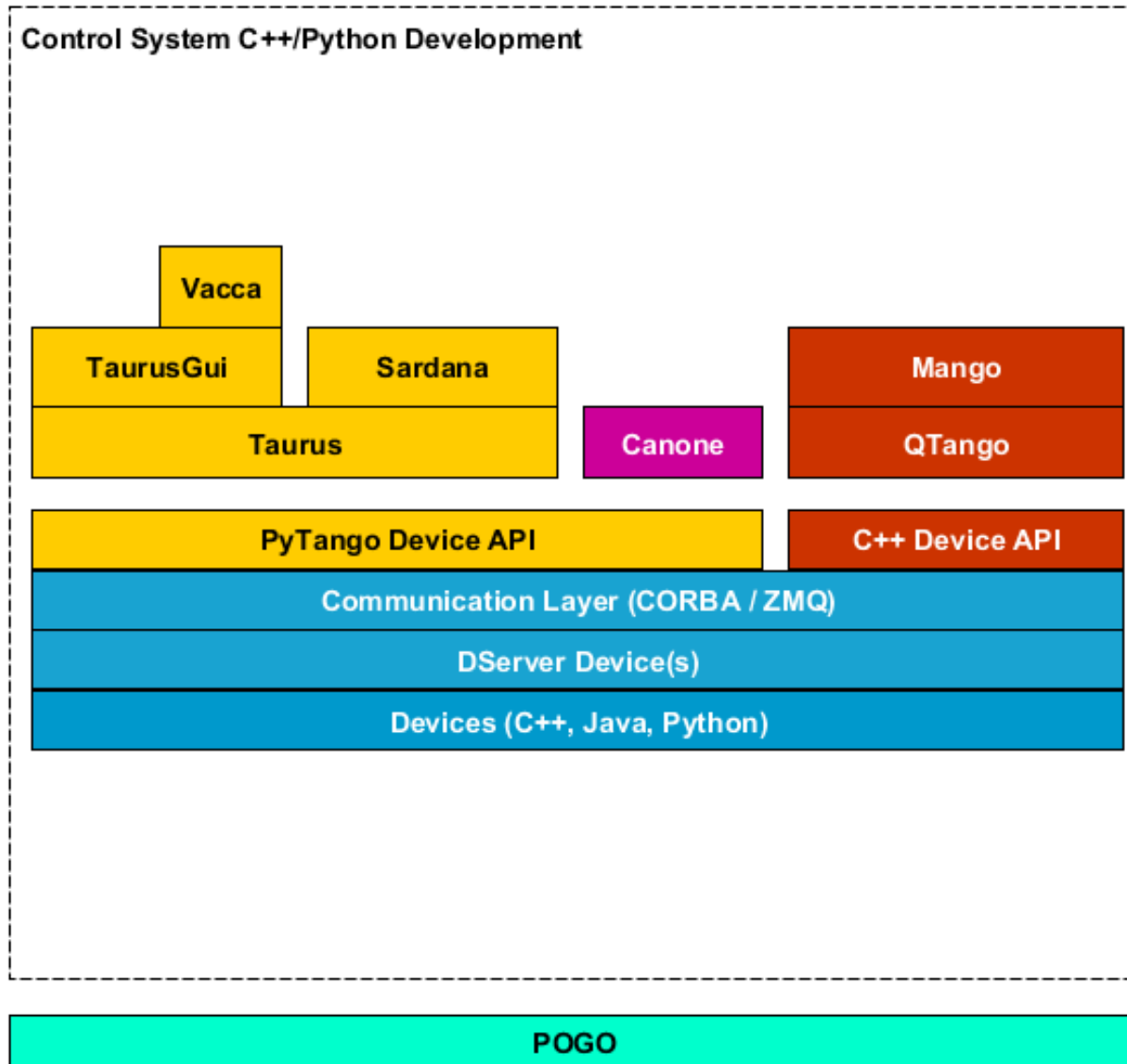
- Guidelines
 - About this document
- Tango Concepts
 - Tango Control system
 - Device concept
 - Hierarchy
 - Communication paradigms
 - Class, Device and Device Server
- Tango Device Design
 - Elements of general design
 - Device interface definition
 - Service availability
- Tango device implementation
 - General rules
 - Device interface
 - Pogo use
 - Internal device implementation
 - Device state management
 - Logging management
 - Error handling
- Appendices
 - Appendix 1 – Code Quality Checklist
 - Appendix 2 – Full code samples



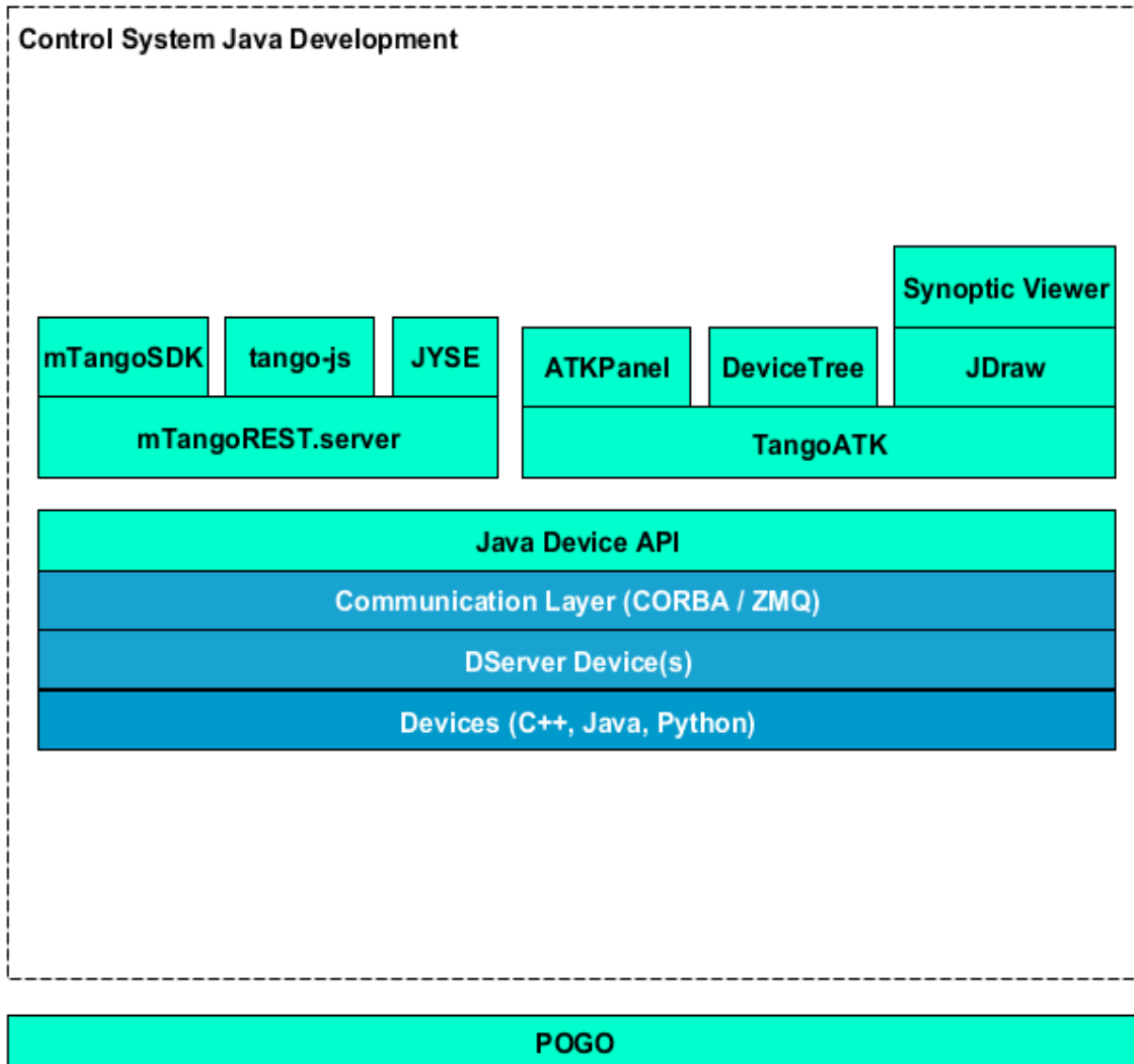
Server + Client api concept #12



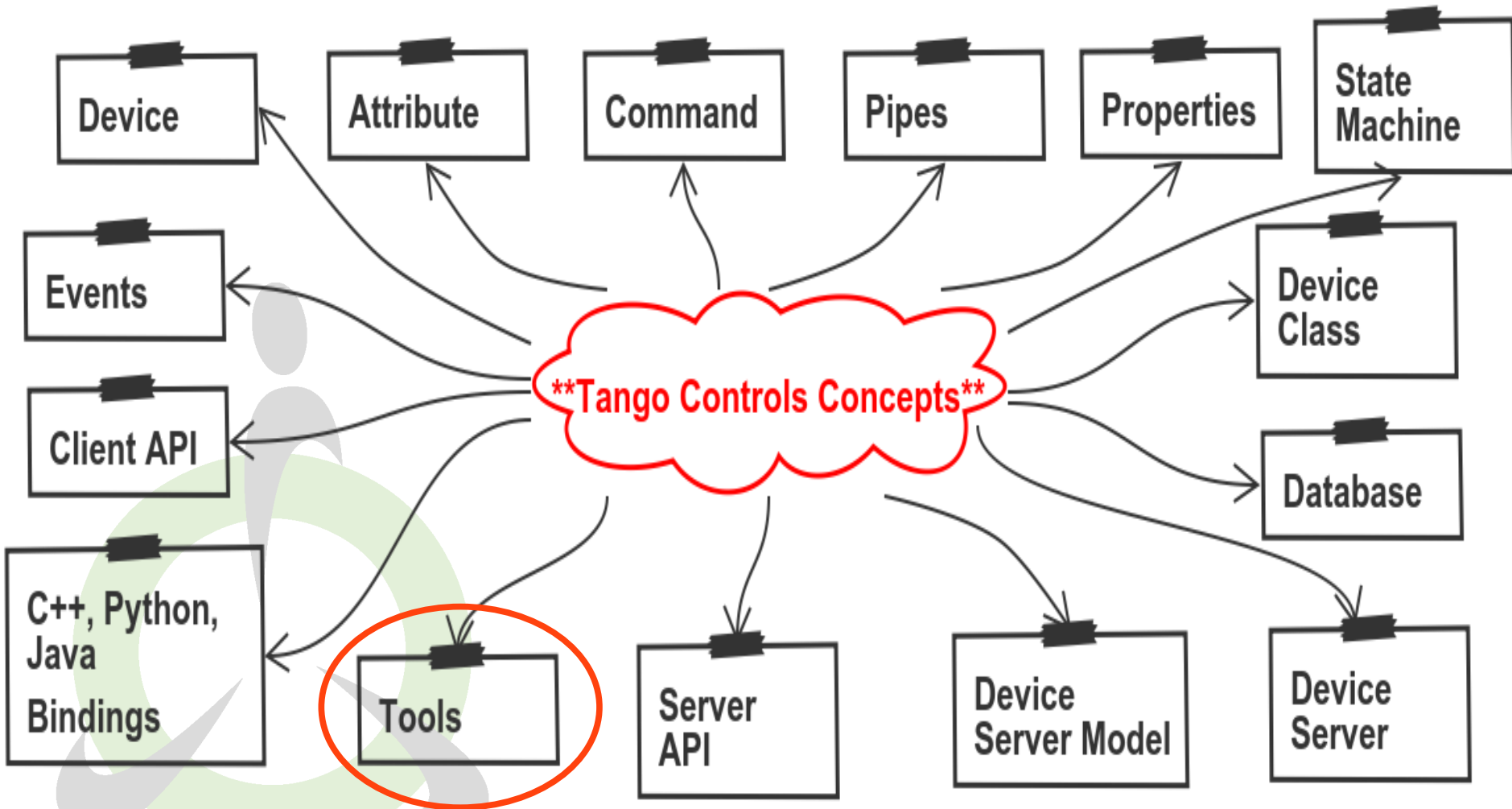
Tango Developers map



Tango Developers map

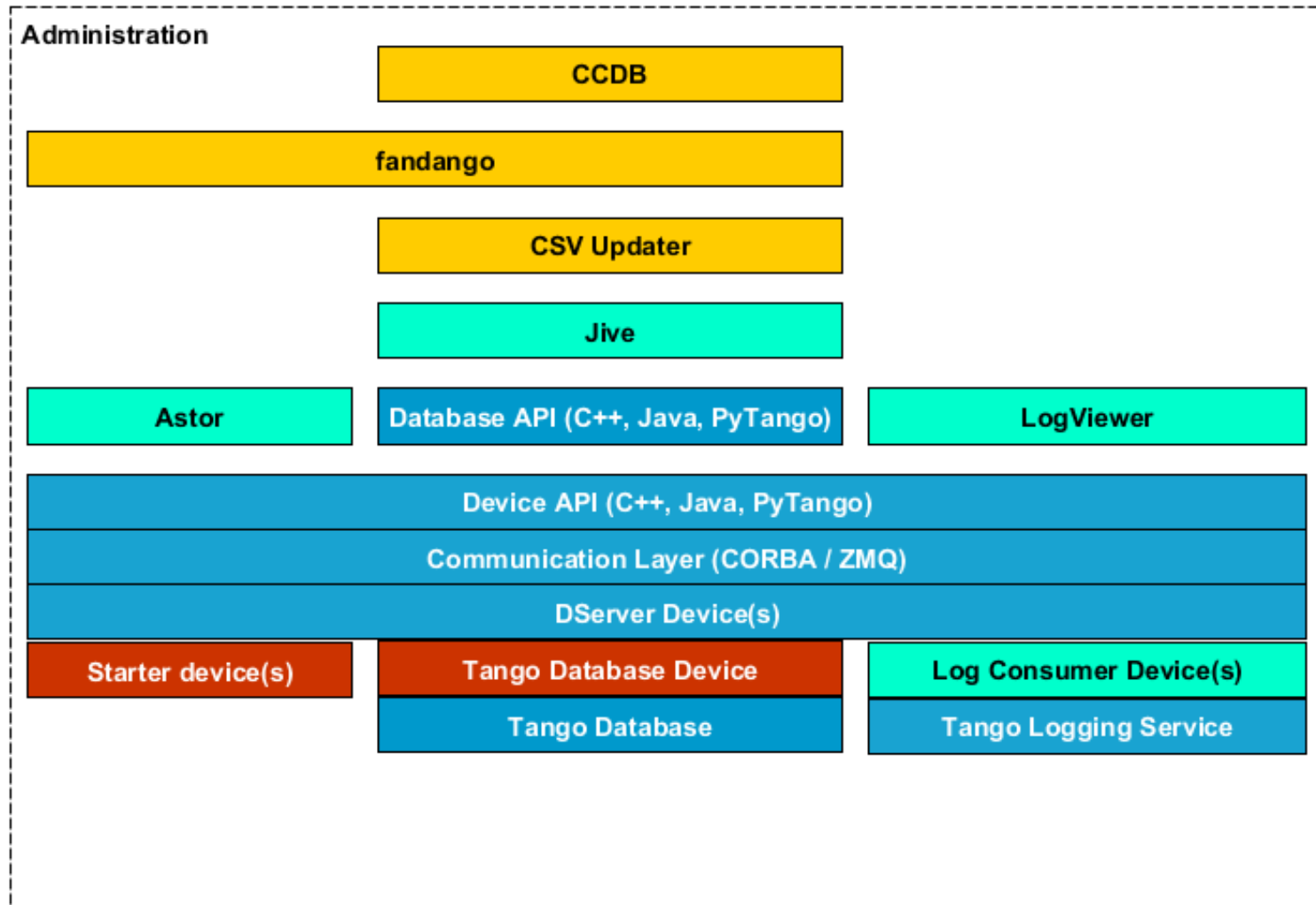


Tools concept #13

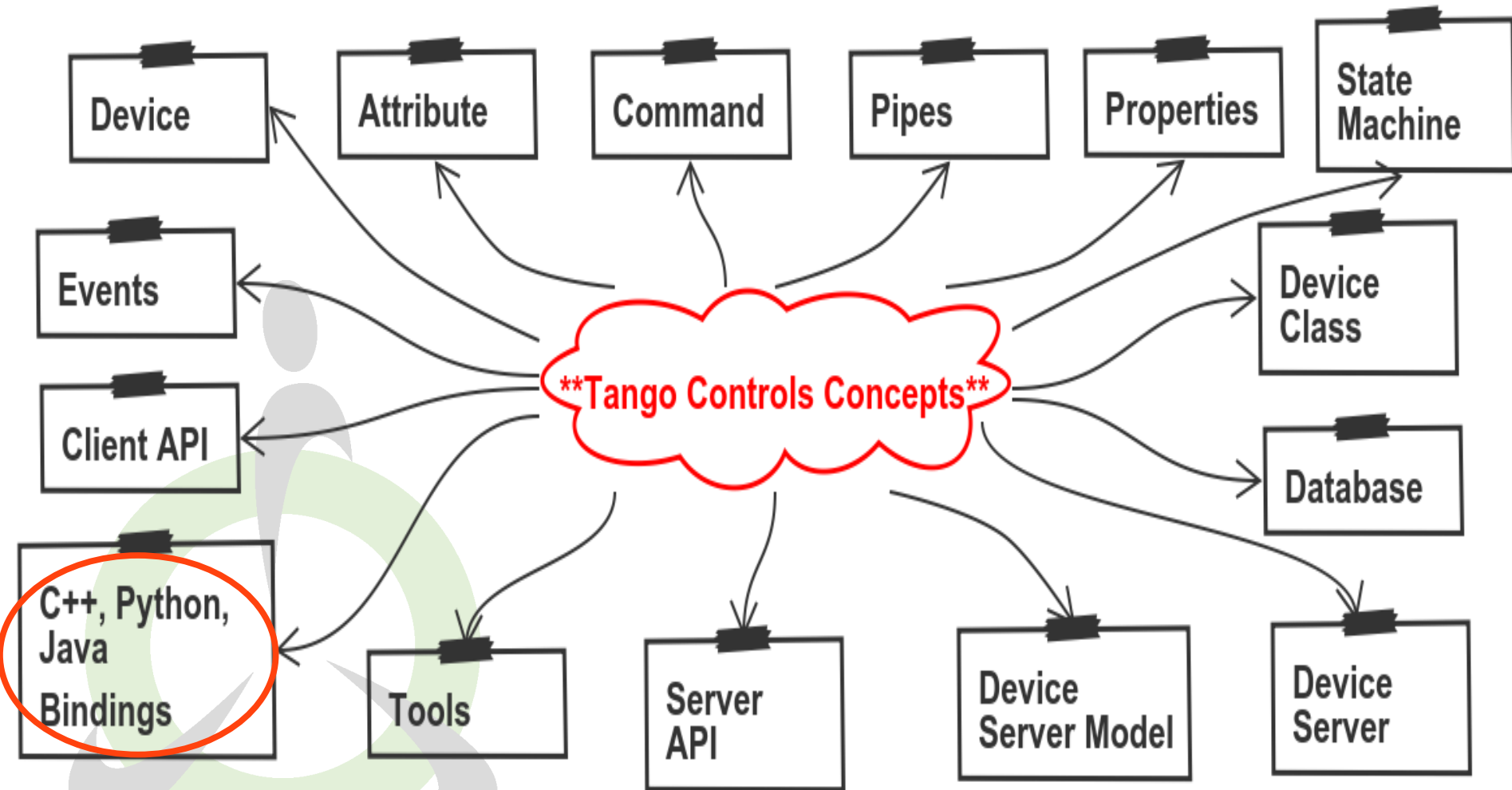


- Many tools exist for Tango to configure + test Devices
- Configuration tool = **Jive** (alternatives are **dsconfig**, **Waltz**, **WebJive**)
- Start/stop control system = **Astor** (alternatives **system 5**, **Supervisord**)
- Test + monitor Device tool = **ATKPanel** (alternatives are **PyTango**, **Taurus**, **Waltz**, **WebJive**, **Jyse**, ...)
- Archiver - **HDB++** (alternatives are **HDB**)
- View logs - **Logviewer** (alternatives **Elasticsearch**)

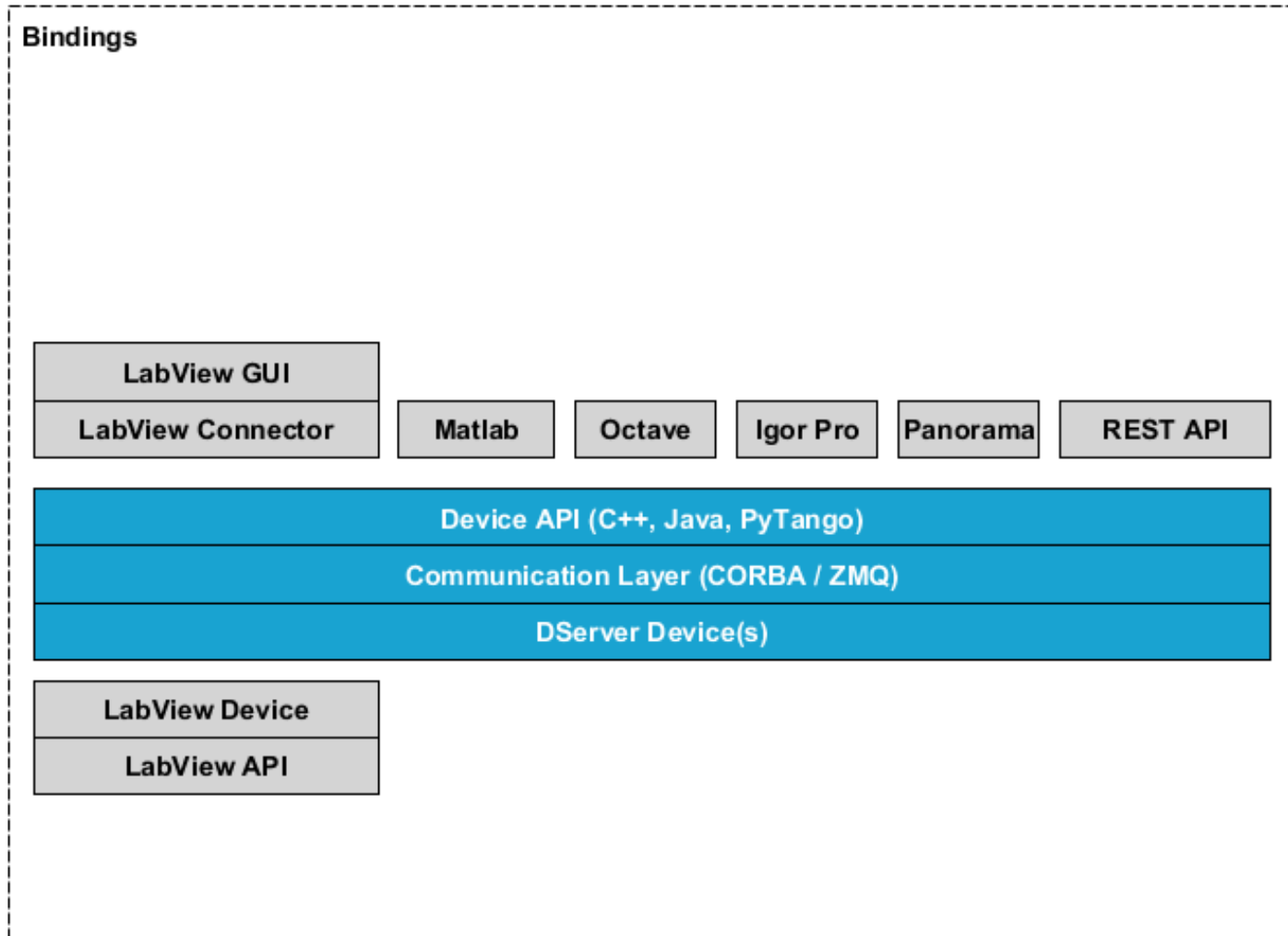
Tango Tools map



Bindings concept #14



Tango Bindings map



Bindings concept #14

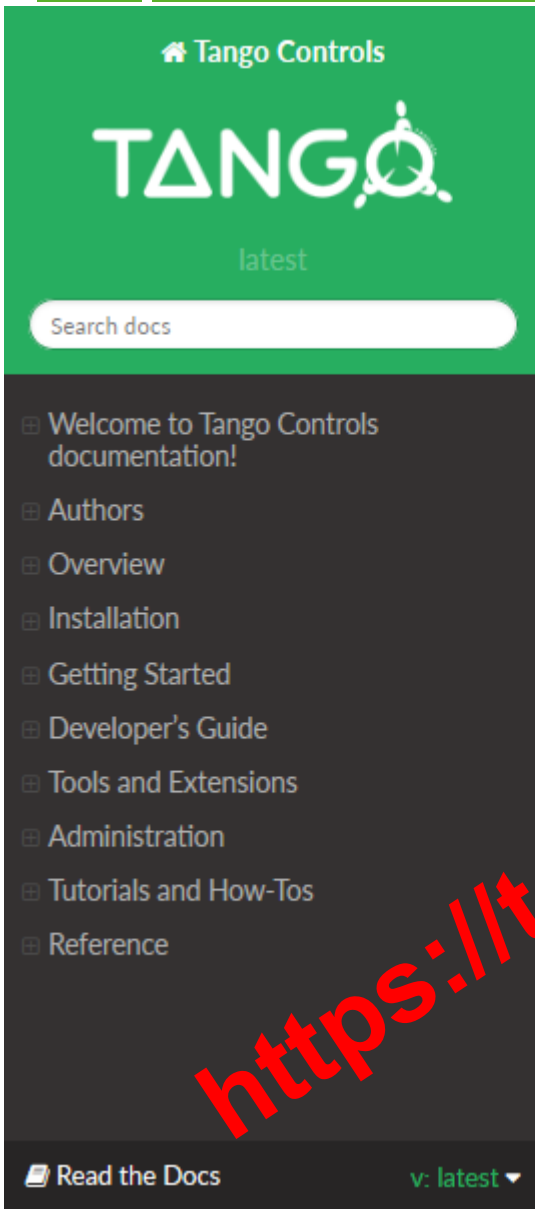
- Many language bindings exist - main ones are **C++**, **Python** and **Java**
- **C++** is core implementation language, support **C++11** (with some arcane **CORBA** memory management specifics)
- **Python** binding builds on **C++** using **Boost Python** (new implementation with **PyBind11** in progress). "Low-level" and "High-level" versions exist. HL-API is more Pythonic
- **Java** implementation based on **Jacorb**. Currently supports Java 8 but will work with Java 11 (13?) with extra **CORBA** libraries.
- Bindings for **Labview**, **Matlab**, **Octave**, **Igorpro** available.

Bindings thought experiment

- **PyTango** is the most used language binding by newcomers to Tango. The High Level API has made it extremely simple to use. Python can be used from Jupyter notebooks + web easily.
- Which **language** do you plan to use?
- **QUESTIONS?**



Read the Documentation !



🏠 Tango Controls

TANGO

latest

- Welcome to Tango Controls documentation!
- Authors
- Overview
- Installation
- Getting Started
- Developer's Guide
- Tools and Extensions
- Administration
- Tutorials and How-Tos
- Reference

Read the Docs v: latest

The documentation is organized in the following categories (some of them overlap):

- **Overview** will give you a quick overview of what Tango Controls is, its origins and who uses it. Start reading here.
- **First steps** will lead you through getting started with Tango Controls. This category includes an overview of Tango Controls concepts, procedures for installation and starting the system as well as *Getting started* tutorials.
- **Developer's Guide** documents the API and information for **Developers** needed for development of **Device Servers** and client applications.
- **Administration** section is important mainly for **System Administrators**. However, it may provide some information for both **End Users** and **Developers**, too. It contains useful information on Tango Controls system development, startup and maintenance.
- **Tools and extensions** Tango comes with rich set of command line tools, graphical toolkits and programming tools for management, developing graphical applications and connecting with other systems and applications. All, **End Users**, **Developers** and **System Administrators** should take a look at the toolkits' manuals.
- **Tutorials and HOWTOs** give step by step guidance and teach you how to work with Tango Controls.
- **Table of Contents** provides access to all documents.
- If you want to contribute to the documentation please read the document **How to work with Tango Controls documentation** and the **Documentation workflow tutorial**.

Indices and tables

- [Table of Contents](#)
- [Index](#)

GitHub – Tango Controls organisation



Tango Controls Core Projects

The official place for Tango Controls core projects

<http://www.tango-controls.org> info@tango-controls.org

Repositories 61

Packages

People 38

Teams 4

Projects 1

Settings

Pinned repositories

Customize pinned repositories

TangoTickets

The official place to create an issue/ticket related to Tango when it impacts several repositories or you don't know where to create your issue

★ 3

cppTango

TANGO kernel C++ implementation

★ 30 🍴 25

pytango

Python binding to Tango C++

Python ★ 36 🍴 37

rest-api

Tango REST API specification

★ 8 🍴 3

JTango

TANGO kernel Java implementation

Java ★ 7 🍴 9

tango-doc

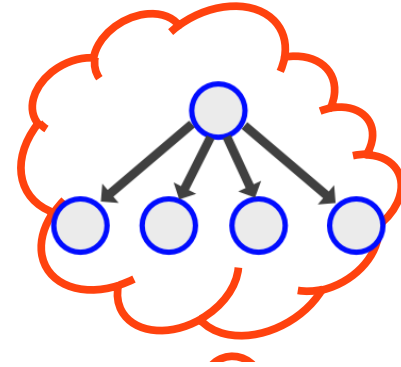
The official documentation for Tango controls

TeX ★ 5 🍴 13

<https://github.com/tango-controls>



- **QUESTIONS**
- **REACTIONS**
- **REQUESTS**



Thank you for your interest !



I think I will
try Tango
Controls

