PUMA

Reliable, secure, scalable and user-oriented design of a multi platform framework based on the most advanced stage of web technologies *

Giacomo Strangolino
Lucio Zambon

Elettra Sincrotrone Trieste

* inspired by an idea of Alessio I. Bogani

Elettra Sincrotrone Trieste

HISTORY

from canone to canone3 (PUMA)

- 2006 - Canone - Python server, drag and drop designer
- 2016 - ElettrApp - Responsive, Cordova, JQuery, Bootstrap
- 2017 - PWMA - C++ server + Websocket (named Canone 2), React, React Native
- 2020 – PUMA - NChan server + SSE (named Canone 3), interactive SVG, designer tool for responsive pages
- 2022 – PUMA – NChan + SSE (caserver) **more robust, modular and distributed architecture**

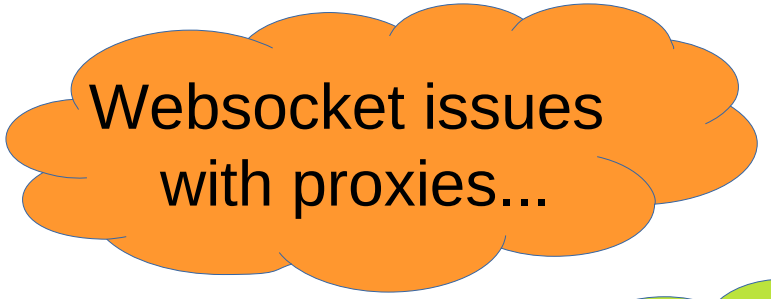
Elettra
Sincrotrone
Trieste

SECTION I

THE SERVICE

DESIGN RATIONALE

1. RELIABLE



<u>The system must work</u>

- From any place, time, platform
- Always, regardless the number of clients
- Always, included when part of the system is unavailable
- Included when network performance is slow (or subject to charges!)

Performance independent of the number of clients

DESIGN RATIONALE

2. SECURE

1) From external attacks (DoS, intrusion)
2) From ill designed clients flooding the service
3) Protect the downstream Control System Engine (Tango, EPICS)

- Network architecture
- Security-oriented **OS**es (BSD!?)
- Framework design, database and code

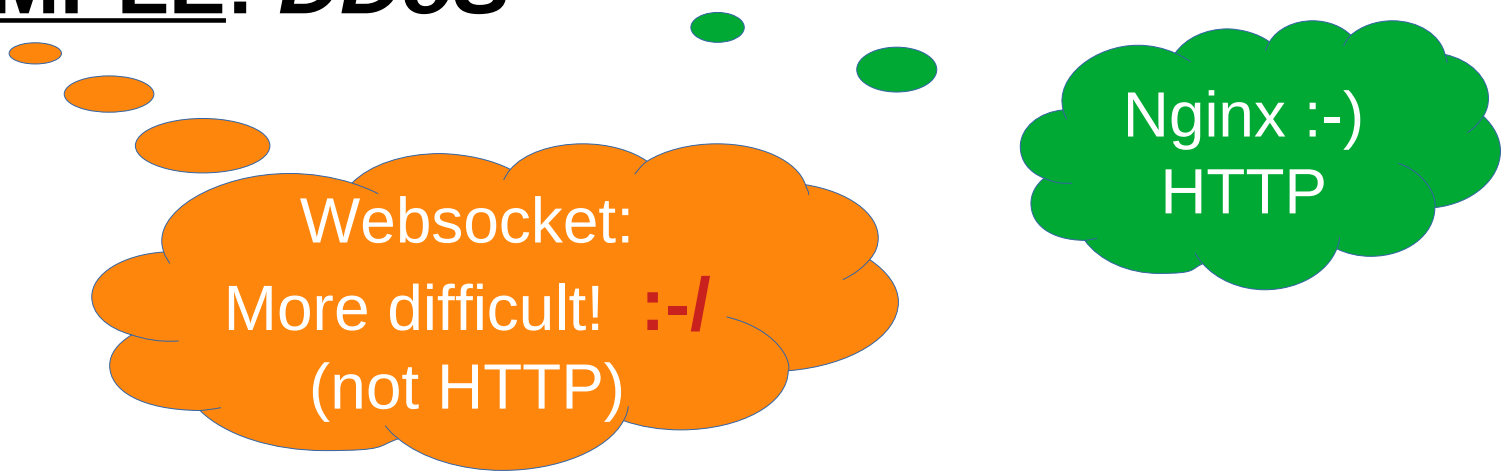
- No service performance decay
- Hamper unruly clients

DESIGN RATIONALE

2. SECURE (II)

GOAL secure enough as to avoid VPN or other additional client-side configuration hindering *usability*, especially on mobile devices

EXAMPLE: ***DDoS***

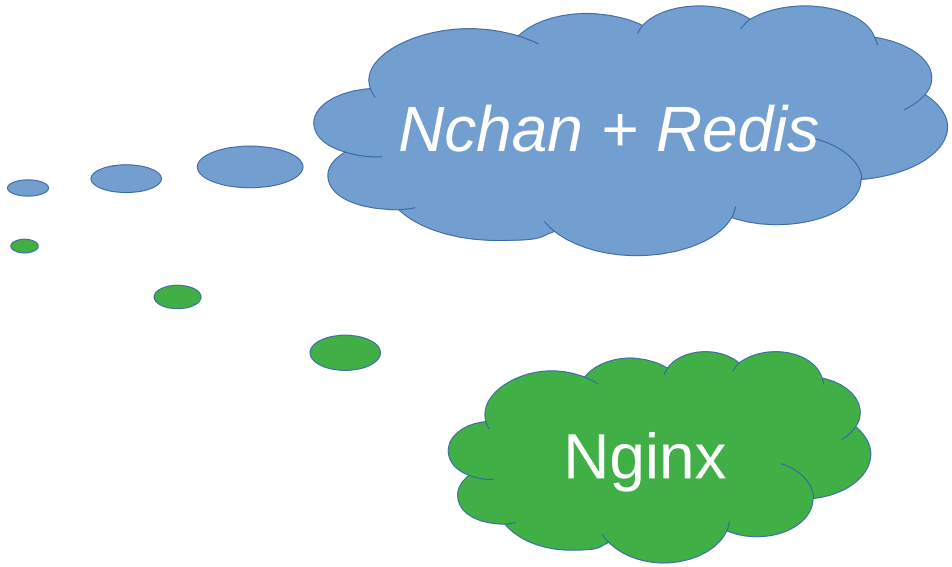


DESIGN RATIONALE

3. SCALABILITY

A good infrastructure must be designed with scalability in mind. It reinforces security and reliability.

1) <u>Horizontal</u>
2) Vertical



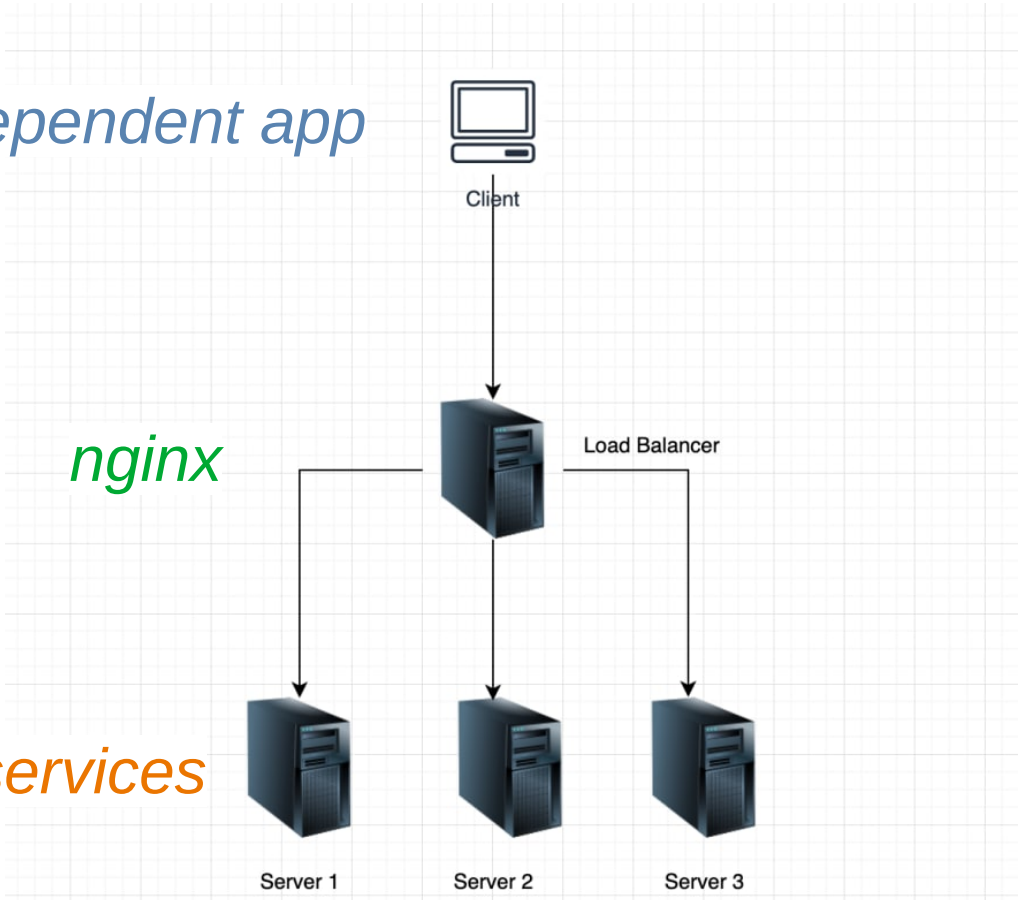
DESIGN RATIONALE

3. SCALABILITY (II)

Platform independent app

nginx

PUMA (canone3) services



Tango, EPICS, …

Elettra Sincrotrone Trieste

DESIGN RATIONALE

4. GENERIC API

Must serve

1) The web
2) Mobile applications
3) Desktop applications *

* Cumbia libs already support the API so that any Qt application can be instructed to rely on either the native control system engine or the HTTP service at runtime

Elettra
Sincrotrone
Trieste

DESIGN RATIONALE

Nginx + nchan + http/SSE * = ?

* inspired by an intuition of Alessio I. Bogani

Elettra Sincrotrone Trieste

Nginx + nchan + http/SSE = ?

- ✓ **NGINX**: high performance load balancer, web server and reverse proxy *https://www.nginx.com/*
- ✓ **NCHAN**: flexible *pubsub* for the modern web
- ✓ **SSE**: a server *push* technology: a client receives automatic updates from a server via HTTP connection

=

A scalable, secure, efficient service with multiplexing for web, mobile and desktop applications

Elettra Sincrotrone Trieste

Nginx + Nchan + Redis

Redis (Remote Dictionary Server) is an in-memory data structure project implementing a distributed, in-memory key–value database with optional durability.

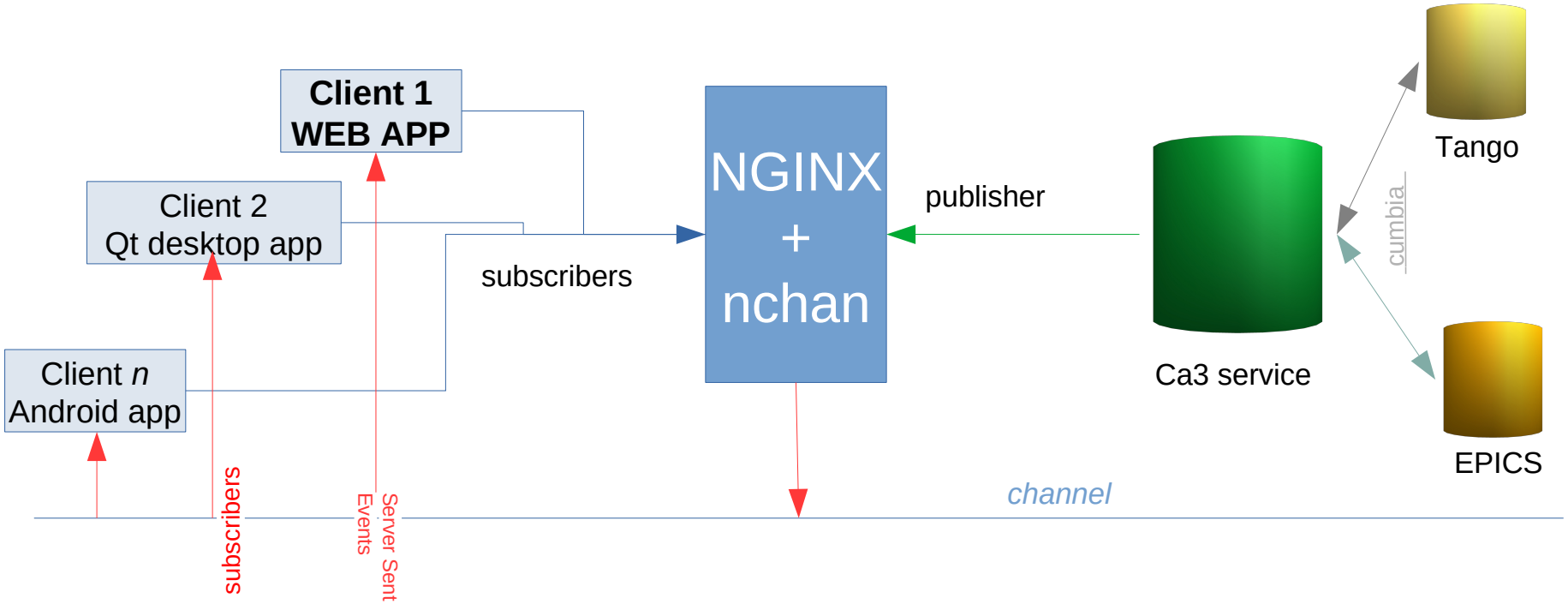
✔ Redis can be used to add data persistence and horizontal scalability, failover and high availability to a Nchan setup.

✔ *Redis Cluster* provides a way to run a Redis installation where data is automatically sharded across multiple Redis nodes.

Nchan + Redis

✔ add scalability via sharding channels among cluster nodes.
✔ *Redis cluster* provides automatic failover, high availability,
✔ *Redis cluster* eliminates the single point of failure of one shared Redis server

Nginx + nchan + http/SSE = ?

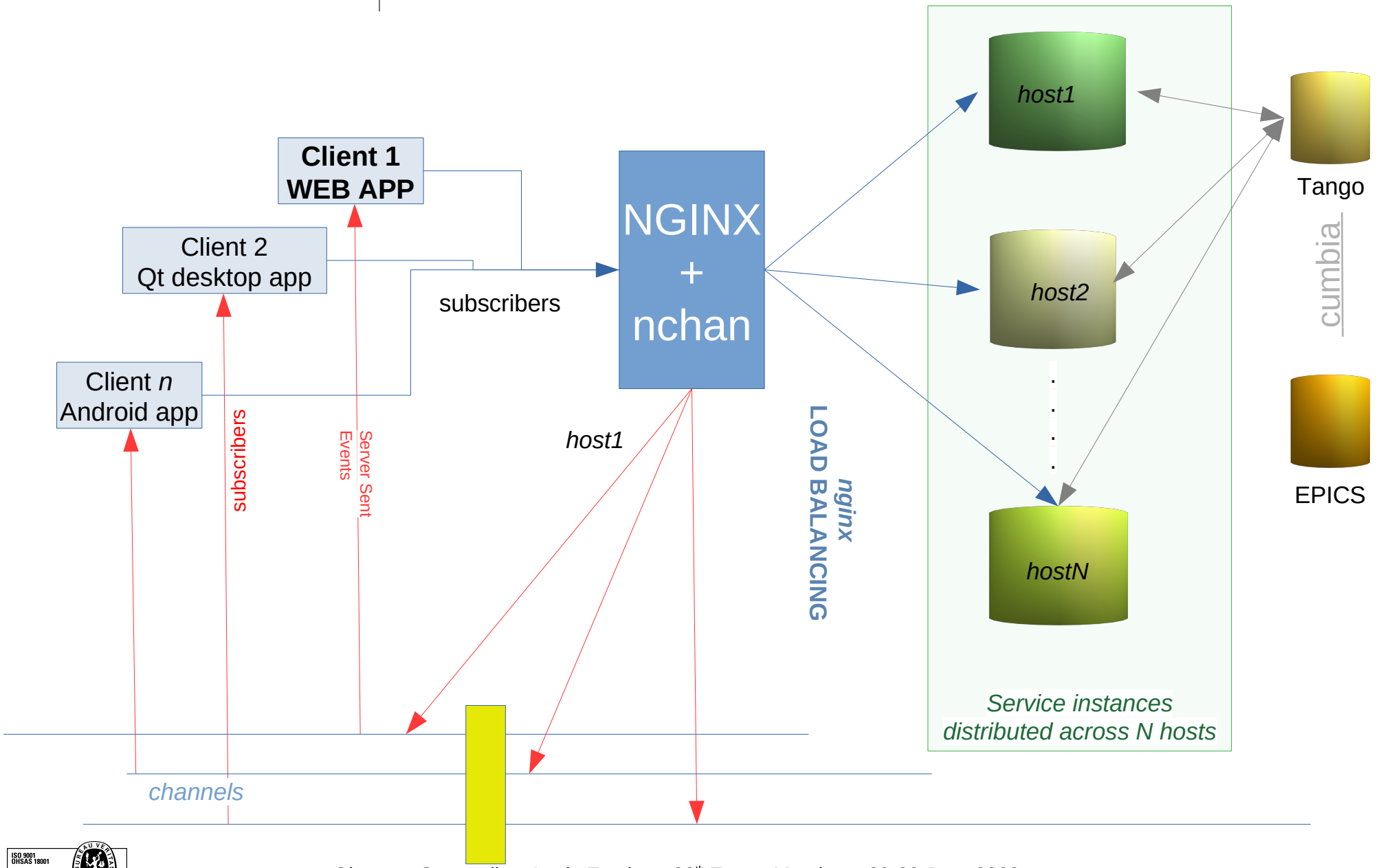


Additionally

- Synchronous readings
- Synch database property fetch
- Authenticated synchronous writings

Multiplexing: *n* clients reading *x* ⇒ *1* reader to the native engine

Nginx + nchan – scalability and load balancing



Nchan + Redis = additional scalability

DESIGN RATIONALE

NEW SERVICE ARCHITECTURE – v1.3.6 (I)

Monitoring over time is a complementary task compared to a single (first) read and configuration phase.

- Monitoring is asynchronous by nature (channels)
- Set up is synchronous: a client subscribes using an *http* request and expects some kind of reply, possibly with *source* configuration and current value.

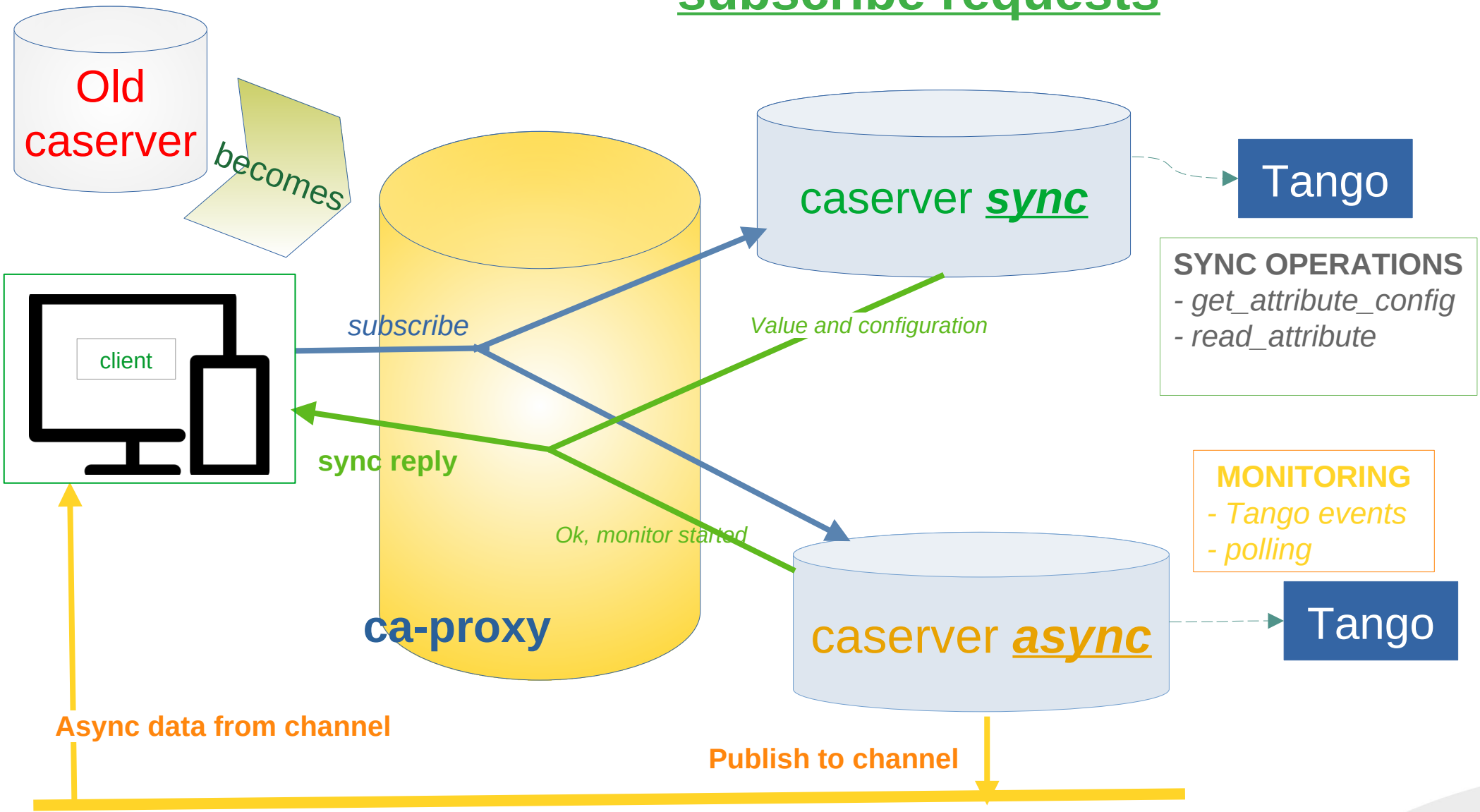
DESIGN RATIONALE – NEW SERVICE ARCHITECTURE – v1.3.6 (II)

Splitting a single one into two services mirrors real functionality:

- A single service is dedicated to a specific task
- Each single service is *much simpler* because it does not need implement *complementary* operations
- Each service is *much* easier to maintain, and so is debugging
- *caserver-async* and *caserver-sync* are the respective names
- A third, intermediate service is needed to *proxy* (split) the client requests and forward them to either or both services (*caserver-proxy)*

DESIGN RATIONALE – NEW SERVICE ARCHITECTURE – v1.3.6 (III)

subscribe requests



Caserver new architecture (I)

Subscribe requests from clients (method: s) imply

✔ an immediate synchronous reply containing the value and configuration of the source

✔ a subscription for updates through an nchan channel on which the client shall listen

✔ a subscribe request shall be forwarded both to the sync and async caserver instances.

✔ The replies from the caservers are then merged together by caserver proxy and sent back to the client as an http reply to the client's http request.

Caserver new architecture (II)

Handle unsubscribe requests (method: u)

✔ *ca-proxy* forwards requests to *all* known *caservers* because it's unaware of which one is monitoring the source (due to *nginx* load balancing across multiple upstream services)

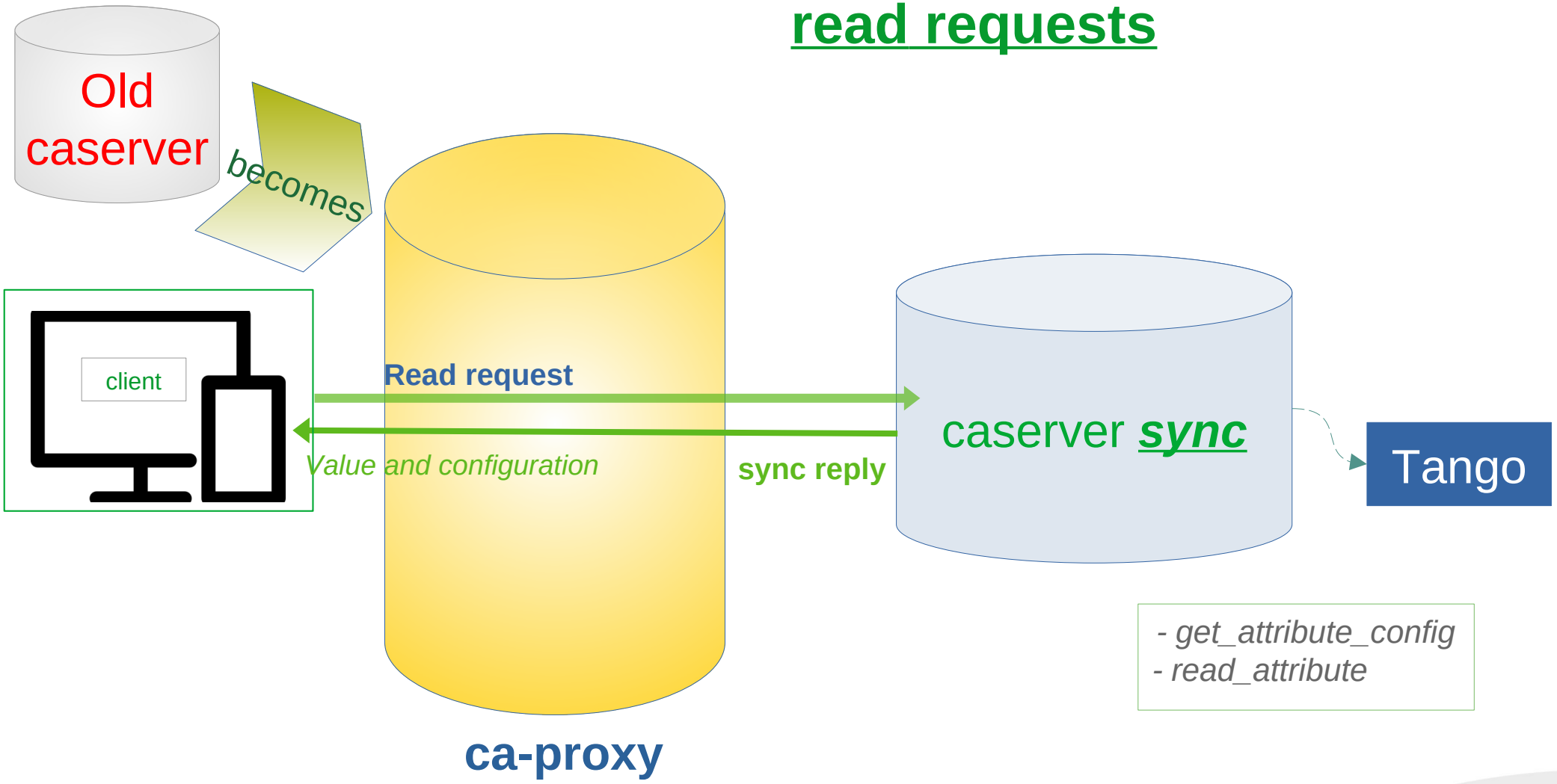
Handle ungracious / vanished clients

✔ clients must periodically send a *keepalive* message in order to receive data from channels;

✔ *ca-proxy* shall periodically check a database to stop monitoring sources linked to *vanished* clients.

DESIGN RATIONALE – NEW SERVICE ARCHITECTURE – v1.3.6 (IV)

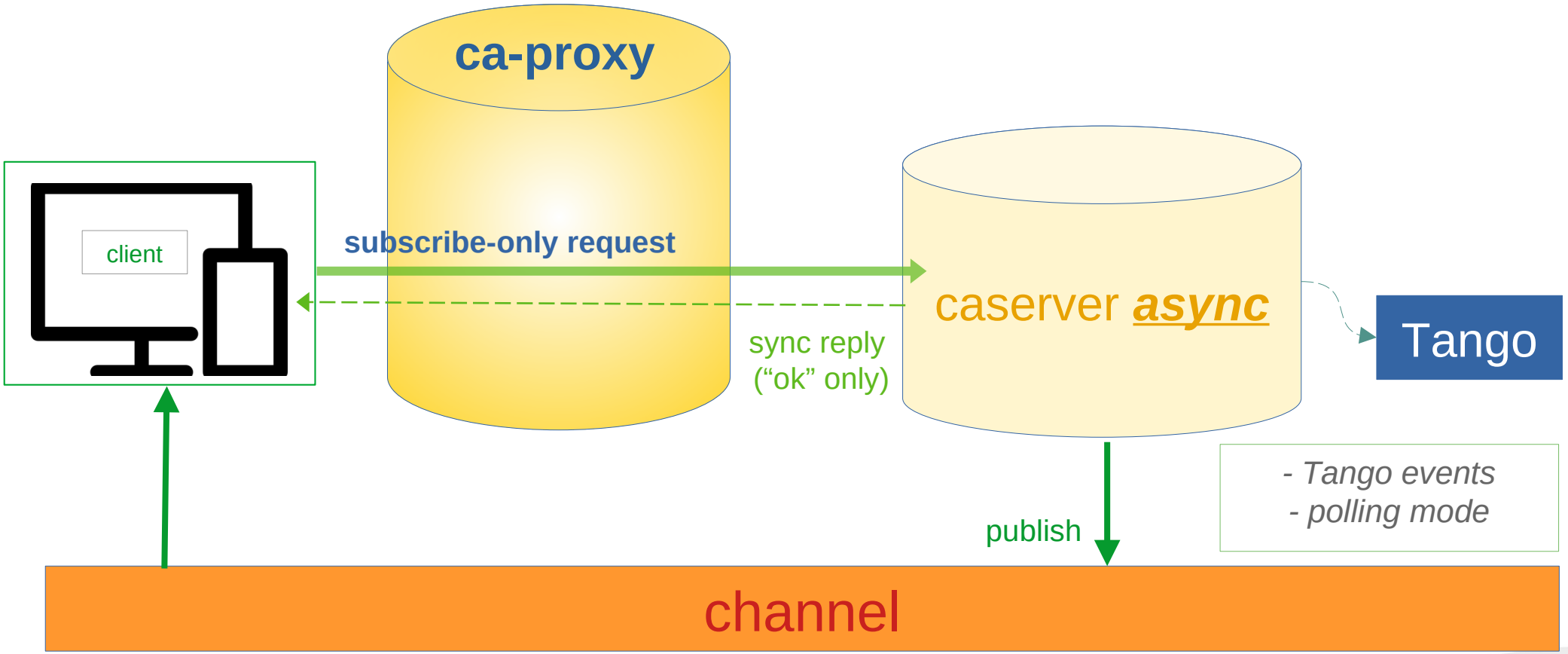
read requests



Read requests are forwarded to caserver-sync only

DESIGN RATIONALE – NEW SERVICE ARCHITECTURE – v1.3.6 (IV)

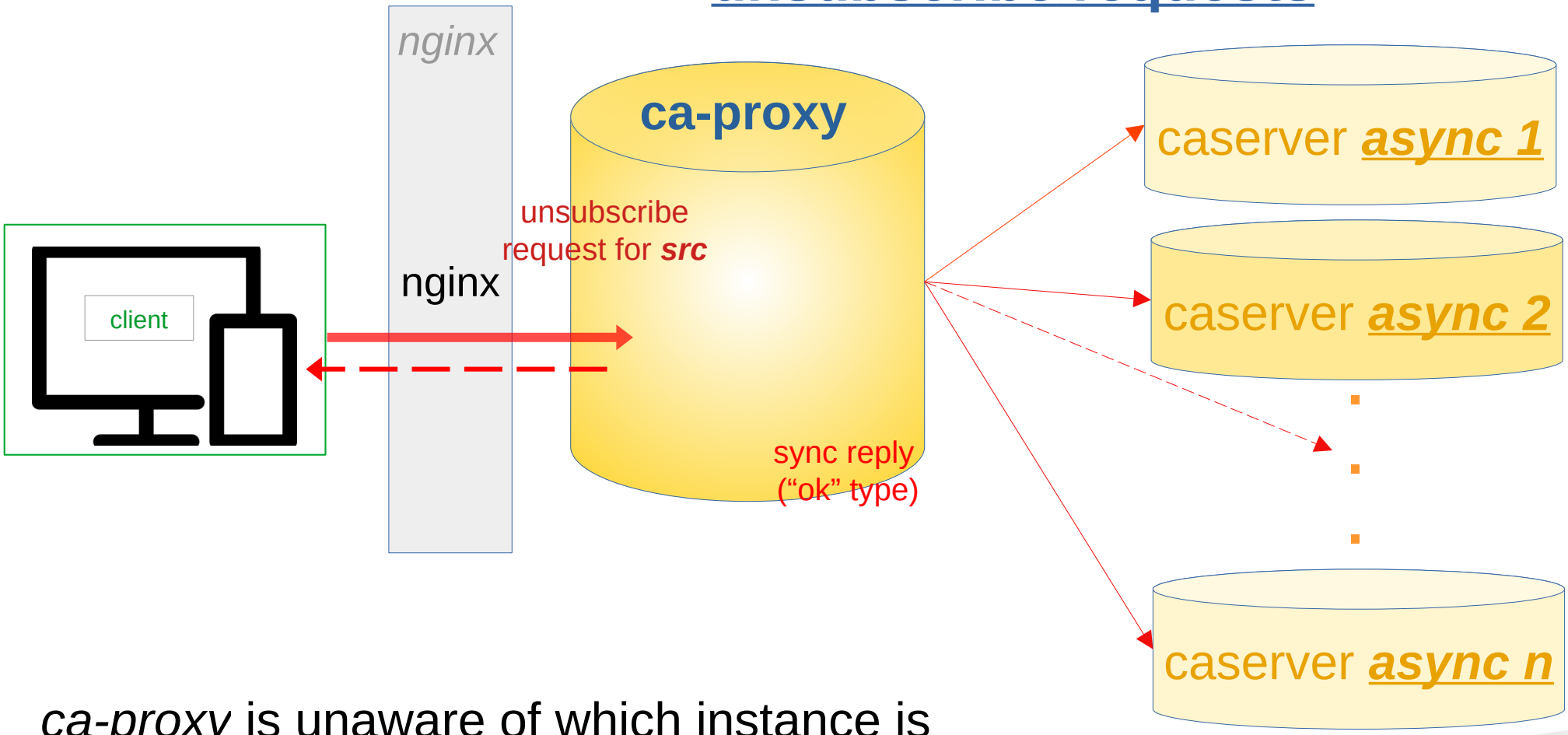
subscribe only requests



subscribe-only requests are forwarded solely to caserver-async

DESIGN RATIONALE – NEW SERVICE ARCHITECTURE – v1.3.6 (V)

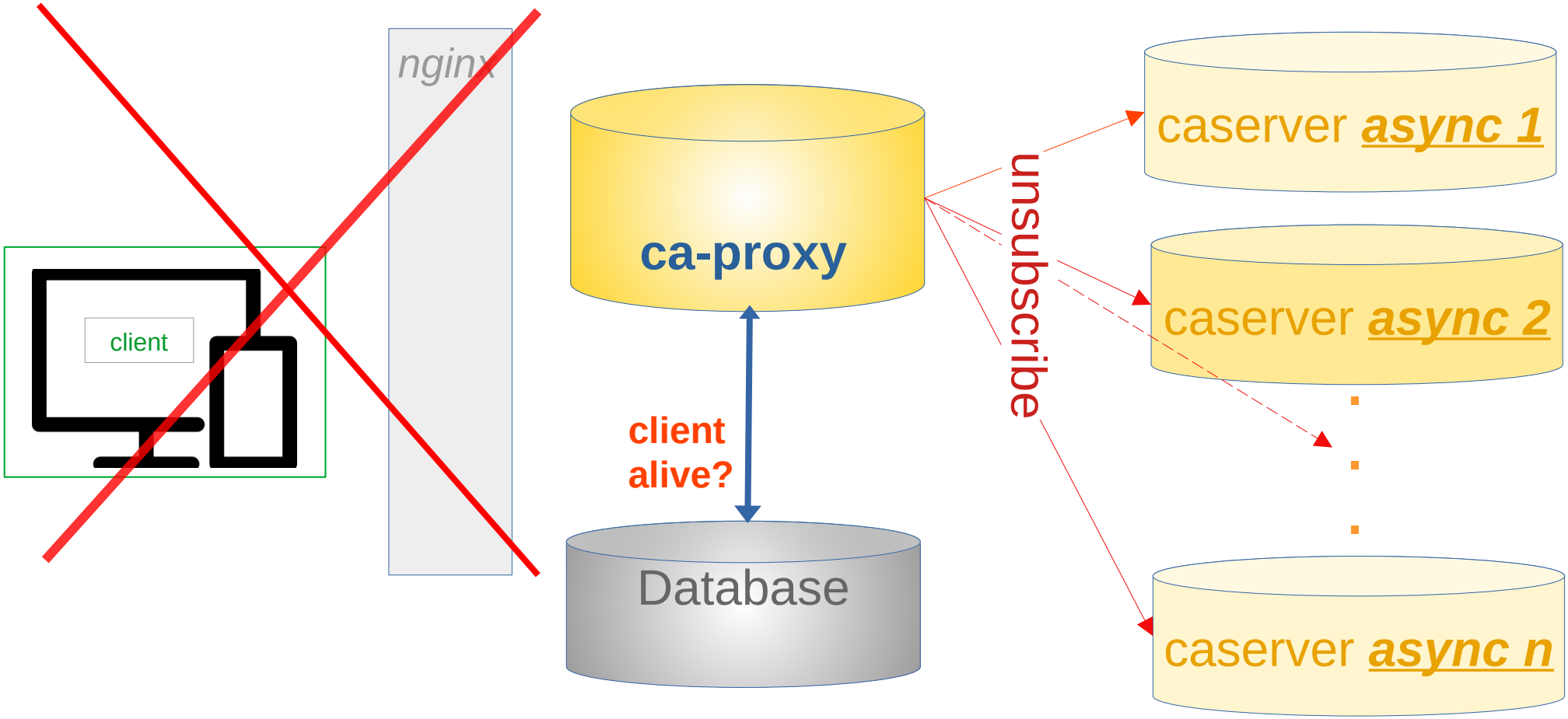
unsubscribe requests



ca-proxy is unaware of which instance is monitoring *src*: broadcast *unsubscribe*

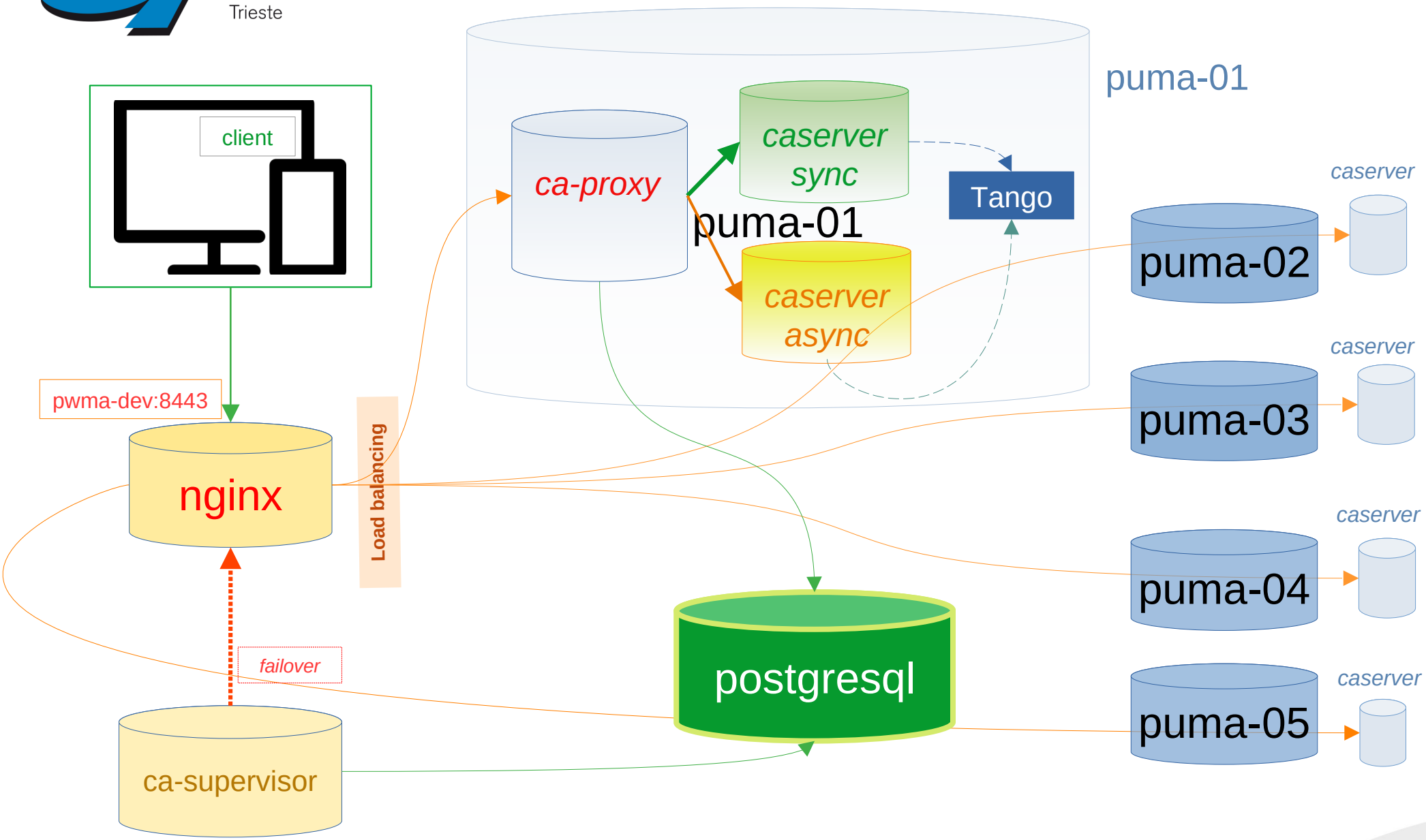
DESIGN RATIONALE – NEW SERVICE ARCHITECTURE – v1.3.6 (VI)

robustness: **vanished clients**



ca-proxy checks for missing clients (not updating their *keep-alive* messages)

DESIGN RATIONALE - DEPLOYMENT



Elettra Sincrotrone Trieste

DESIGN RATIONALE

TEST ENVIRONMENT – LOAD BALANCING

The test environment is pictured in the previous slide. A query to the database shows the the distribution of 1255 readings across six services:

id	started	expected	state	addr	count
311	2021-09-13 13:35:18.331211	2021-09-13 14:20:23.492731	ACTIVE	192.168.205.159	212
320	2021-09-13 13:35:17.06742	2021-09-13 14:20:30.02976	ACTIVE	192.168.205.106	222
326	2021-09-13 13:35:17.653089	2021-09-13 14:20:21.798637	ACTIVE	192.168.205.149	235
327	2021-09-13 13:35:17.54277	2021-09-13 14:20:30.455891	ACTIVE	192.168.205.158	215
324	2021-09-13 13:35:16.584578	2021-09-13 14:20:20.77197	ACTIVE	192.168.205.157	161
328	2021-09-13 13:35:18.383527	2021-09-13 14:20:26.159456	ACTIVE	192.168.205.166	210

(6 rows)

1255

DESIGN RATIONALE

TEST ENVIRONMENT – LOAD BALANCING (II)

Load balancing is accomplished by *nginx.* Take a look at *nginx.conf*:

```
http {
        upstream caserver {
#               hash $http_x_channel consistent;
#
#               A request is sent to the server with the least number of active connections
#               https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/#c
                least_conn;

                server puma-01.elettra.trieste.it:9292 fail_timeout=20s;
                server puma-02.elettra.trieste.it:9292 fail_timeout=20s;
                server puma-03.elettra.trieste.it:9292 fail_timeout=20s;
                server puma-04.elettra.trieste.it:9292 fail_timeout=20s;
                server puma-05.elettra.trieste.it:9292 fail_timeout=20s;
                # -dev hosts serve as test nodes
                server pwma-dev.elettra.trieste.it:9292 fail_timeout=20s;
                server puma-dev-01.elettra.trieste.it:9292 fail_timeout=20s;

        }
```

DESIGN RATIONALE

TEST ENVIRONMENT – FAILOVER

Simulate a server failure in one of the hosts:

```
puma@puma-04:~$ host puma-04
puma-04.elettra.eu has address 192.168.205.159
puma@puma-04:~$ sudo killall -SEGV caserver
```

Take a look at the *casupervisor* logs:

```
[Mon Sep 13 14:44:17 2021] [thread:0x7f7fc717d700] ca-supervisor:  #  ACTIVE     "pwma-dev" 192.168.205.106:9292 | expected:   13s  [Mon Sep 13 15:44:30 2021]|started: Sun Sep 13 13:35:17 2021
[Mon Sep 13 14:44:17 2021] [thread:0x7f7fc717d700] ca-supervisor:  #  ACTIVE     "puma-01" 192.168.205.149:9292 | expected:   15s  [Mon Sep 13 15:44:32 2021]|started: Sun Sep 13 13:35:17 2021
[Mon Sep 13 14:44:17 2021] [thread:0x7f7fc717d700] ca-supervisor:  #  ACTIVE     "puma-02" 192.168.205.157:9292 | expected:   14s  [Mon Sep 13 15:44:31 2021]|started: Sun Sep 13 13:35:16 2021
[Mon Sep 13 14:44:17 2021] [thread:0x7f7fc717d700] ca-supervisor:  #  ACTIVE     "puma-03" 192.168.205.158:9292 | expected:   14s  [Mon Sep 13 15:44:31 2021]|started: Sun Sep 13 13:35:17 2021
[Mon Sep 13 14:44:17 2021] [thread:0x7f7fc717d700] ca-supervisor:  #  ACTIVE     "puma-05" 192.168.205.166:9292 | expected:   19s  [Mon Sep 13 15:44:36 2021]|started: Sun Sep 13 13:35:18 2021
[Mon Sep 13 14:44:17 2021] [thread:0x7f7fc717d700] ca-supervisor:  #  ZOMBIE     "puma-04" 192.168.205.159:9292 | expected: -103s  [Mon Sep 13 15:42:34 2021]|started: Sun Sep 13 13:35:18 2021
```

Elettra Sincrotrone Trieste

DESIGN RATIONALE

TEST ENVIRONMENT – FAILOVER (II)

Execute the same query as in the first *load balancing* inspection:

id	started	expected	state	addr	count
320	2021-09-13 13:35:17.06742	2021-09-13 14:47:00.696596	ACTIVE	192.168.205.106	434
326	2021-09-13 13:35:17.653089	2021-09-13 14:46:52.57544	ACTIVE	192.168.205.149	235
327	2021-09-13 13:35:17.54277	2021-09-13 14:47:01.1764	ACTIVE	192.168.205.158	215
324	2021-09-13 13:35:16.584578	2021-09-13 14:47:01.49247	ACTIVE	192.168.205.157	161
328	2021-09-13 13:35:18.383527	2021-09-13 14:46:56.885659	ACTIVE	192.168.205.166	210

(5 rows)

1255

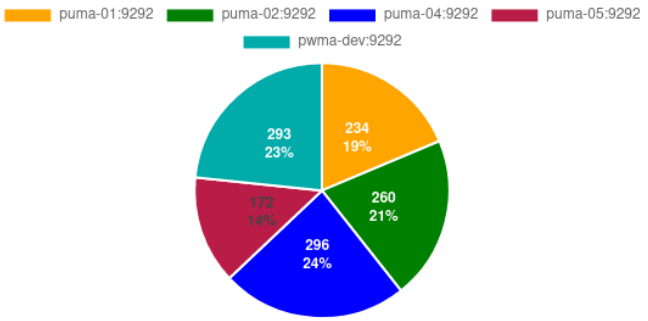
Please note that load redistribution after a failure is always administered by *nginx*

TEST ENVIRONMENT

Load balancing and service status: web view

https://pwma-dev.elettra.eu/status.php

server:port load balancing 2021-09-15 10:45:45 src n: 1255



service history activity history

id	started	expected	state	conf_id	srvnam	addr	port
320	2021-09-15 10:44:12.239299	2021-09-15 10:45:08.870399	ACTIVE	1	pwma-dev	192.168.205.106	9292
317	2021-07-20 10:09:22.534603	2021-07-20 12:27:27.964855	INACTIVE	4	pwma-dev	192.168.205.106	19292
326	2021-09-15 10:44:13.767563	2021-09-15 10:45:05.899086	ACTIVE	5	puma-01	192.168.205.149	9292
324	2021-09-15 10:44:12.994295	2021-09-15 10:45:11.993982	ACTIVE	7	puma-02	192.168.205.157	9292
333	2021-09-14 16:23:37.325481	2021-09-14 16:24:48.04815	ACTIVE	9	puma-03	192.168.205.158	9292
332	2021-09-14 16:23:37.907864	2021-09-15 10:45:11.991821	ACTIVE	10	puma-04	192.168.205.159	9292
328	2021-09-14 16:23:38.346319	2021-09-15 10:45:10.384647	ACTIVE	12	puma-05	192.168.205.166	9292
300	2021-07-06 14:53:22.219985	2021-07-06 15:05:30.619484	INACTIVE	18	woody	192.168.1.159	9292

Elettra
Sincrotrone
Trieste

DESIGN RATIONALE (II)

SECTION II

NATIVE CLIENTS

DESIGN RATIONALE (II)

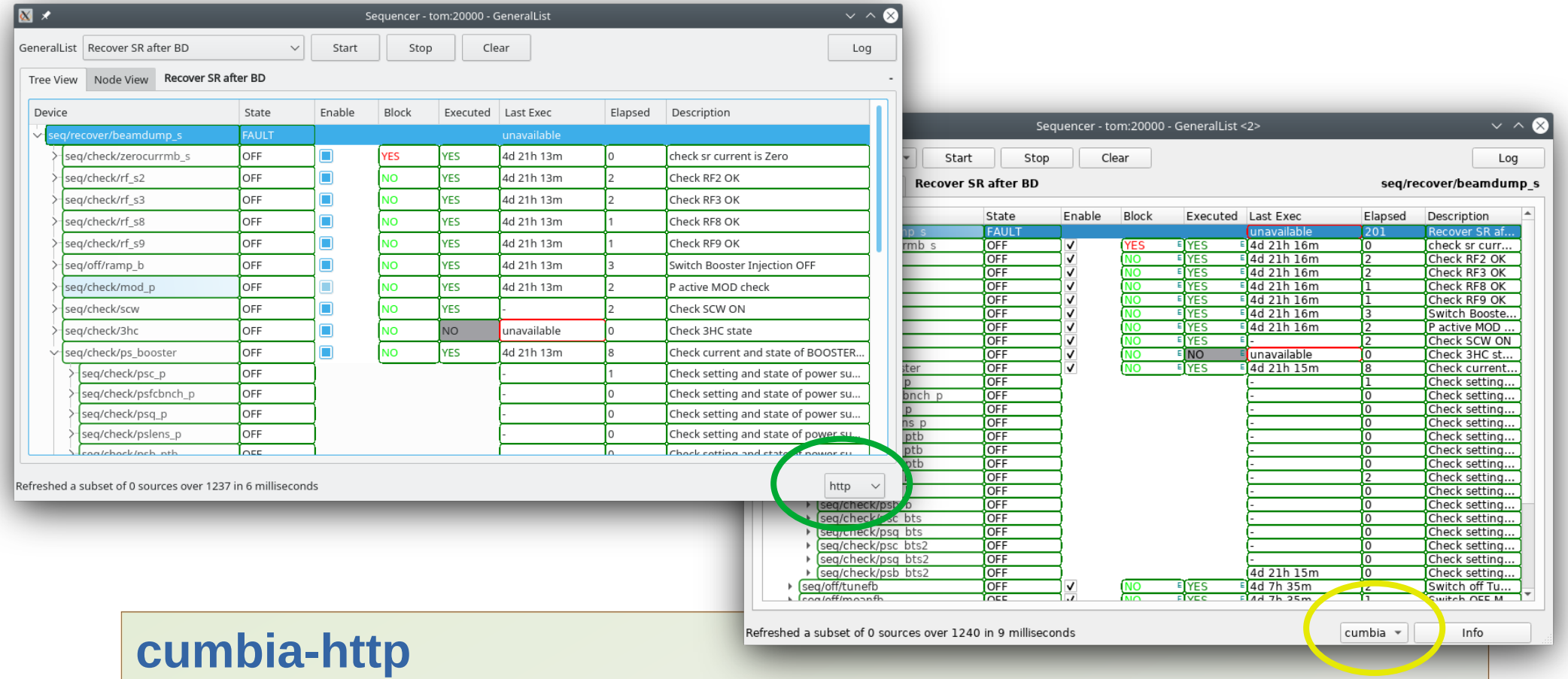
1. MULTI PLATFORM

We deem *Telegram* a perfect example of multi platform application:

1. A *native* app on all mobile devices
2. A *desktop* applications for all platforms (even *FreeBSD*)
3. A *web* interface
4. Has a simple and open API to create clients, bots, …
5. Efficiency, security and privacy centered

These traits have continuously inspired the development of *canone3*

Clients – Qt desktop apps



cumbia-http

- ✗ the same app is *designed* and *run* transparently regardless the engine in use (*native Tango* or *http/SSE*)
- ✗ No *engine-specific* coding
- ✗ Run with the same command line!
- ✗ A virtual machine run at home can reproduce exactly the control room desktop

Clients – Control Room apps

Approaches to running control room applications remotely

Now

✗ Control room virtual machines run on the server side

✗ Although optimized, a full graphical session is streamed, with a strong impact on *bandwidth* and *battery life* of portable devices. If the former today may not be considered a critical limit (however, estimate the load of a server streaming to hundreds of clients), the latter is indeed a precious resource for portable devices and laptops. *The only relevant piece of information is* ***data****.*

Then

✗ Control room virtual machines run on the client

✗ The user runs only the apps he needs

✗ The apps run natively (i.e. the fastest possible on the client device), look exactly as in the control room (they are the same) without lags

✗ *Only data through the network*, for native and web apps alike

Clients – Control Room apps

Approaches to running control room applications remotely (II)

Additionally

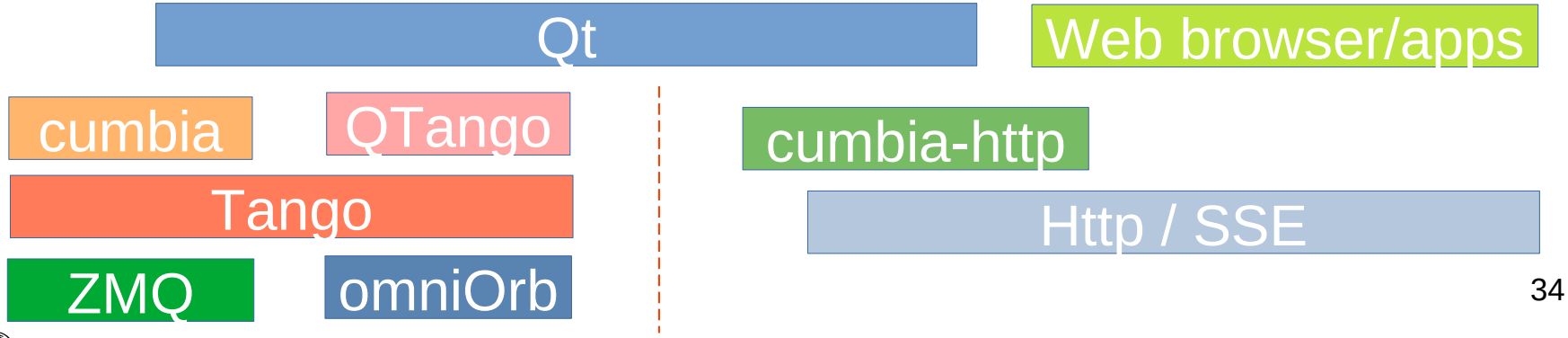
✗ Either *virtual* or *physical* machines do not need Tango + dependencies

✗ They need only a web browser and Qt for native apps

✗ Events only

✗ Tango control system potentially *highly relieved* (remember: *N* clients reading *x*, 1 read to the control system)

+ all benefits discussed in the *design rationale* section



Elettra
Sincrotrone
Trieste

SECTION III

WEB CLIENTS

1) JavaScript
2) Decentralization

Elettra Sincrotrone Trieste

WEB INTERFACES

Does ES6 make JavaScript frameworks obsolete?

byTheodoros 'Theo' Karasavvas

https://stackoverflow.blog/2021/11/10/does-es6-make-javascript-frameworks-obsolete/

Elettra
Sincrotrone
Trieste

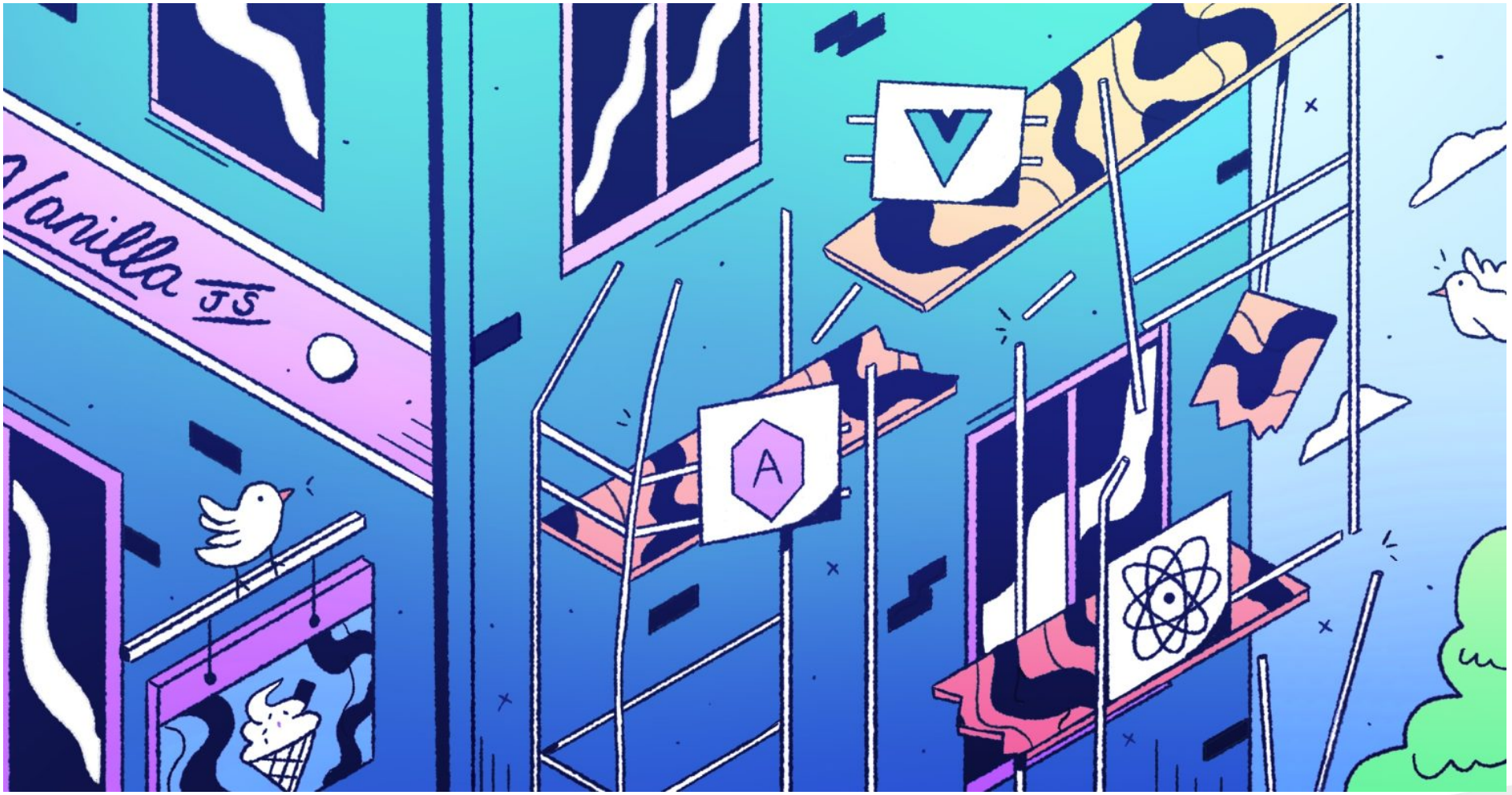
WEB INTERFACES

"Every time JavaScript goes through a major update, we seem to repeat the same cycle. At first, developers are delighted by the new features. They move back to coding directly in JavaScript, and frameworks become less popular. Then, in the relatively long periods between releases, frameworks begin to offer new features and tempt developers back. Repeat."

[React] "building a complex stack of components that you then have to constantly maintain and manage"

Elettra
Sincrotrone
Trieste

WEB INTERFACES



Elettra Sincrotrone Trieste

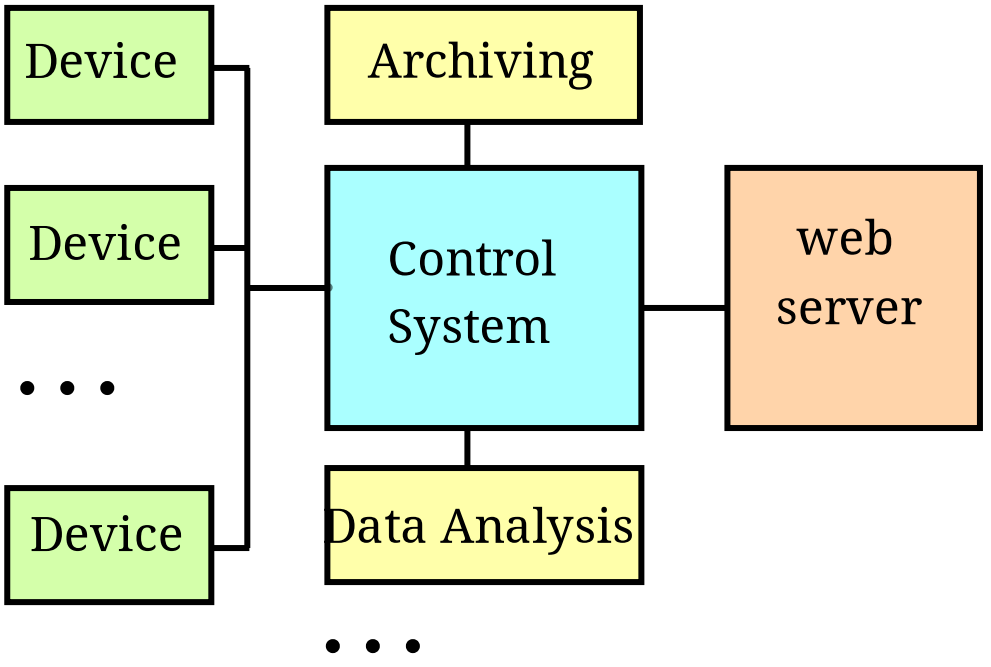
WEB INTERFACES

"In short, ES6 brought a host of syntactic changes to JavaScript that greatly reduce the need for most frameworks. Coupled with the fact that most of the frameworks we are using at the moment obscure JavaScript code and add an additional dependency, we could see a tangible and permanent reduction in the use of frameworks over the next few years.
Or perhaps the cycle will simply repeat, and we'll have only a few years of learning how to write better JavaScript before we retreat into our frameworks again."

WEB INTERFACES

2. Decentralization

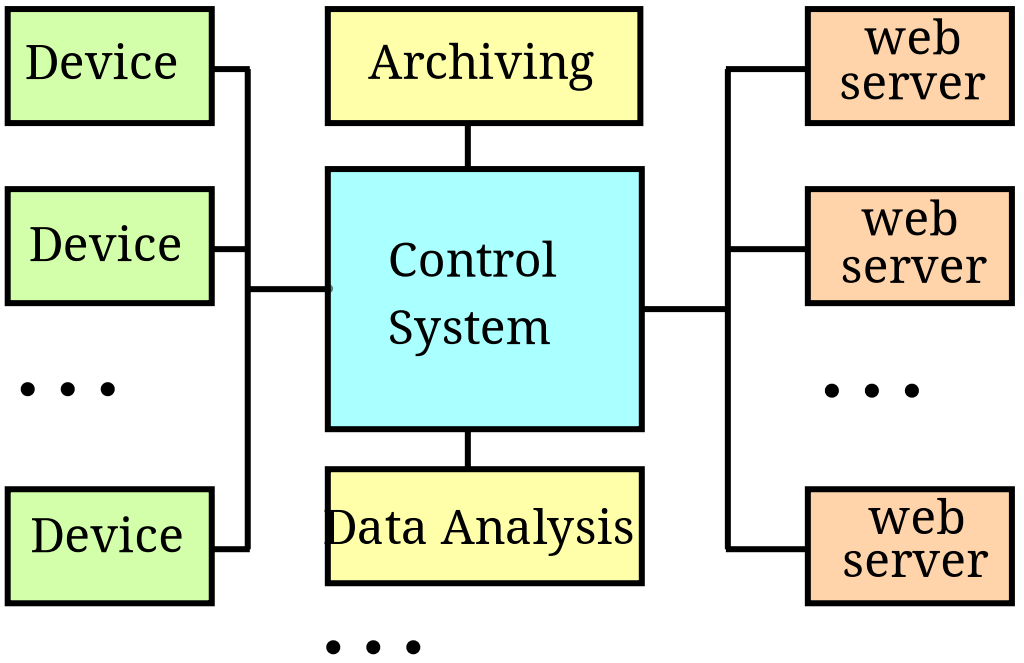
- We started with a centralized python server from Tango to socket and a PHP server from socket to web (from 2006 to 2017)



- than we moved to a C++ server from Tango or Epics to web with websocket (2017 - 2020)
- NGINX NChan server + SSE (2020 - 2022)

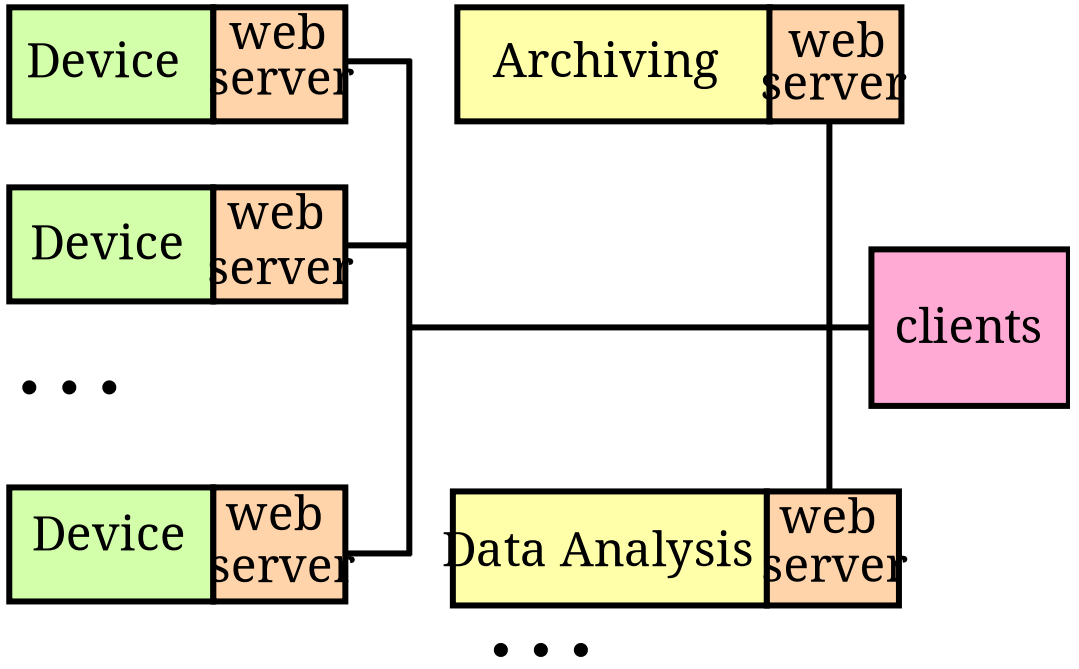
WEB INTERFACES

We changed again Nchan based server to make it more robust, modular and distributed (since 2022)

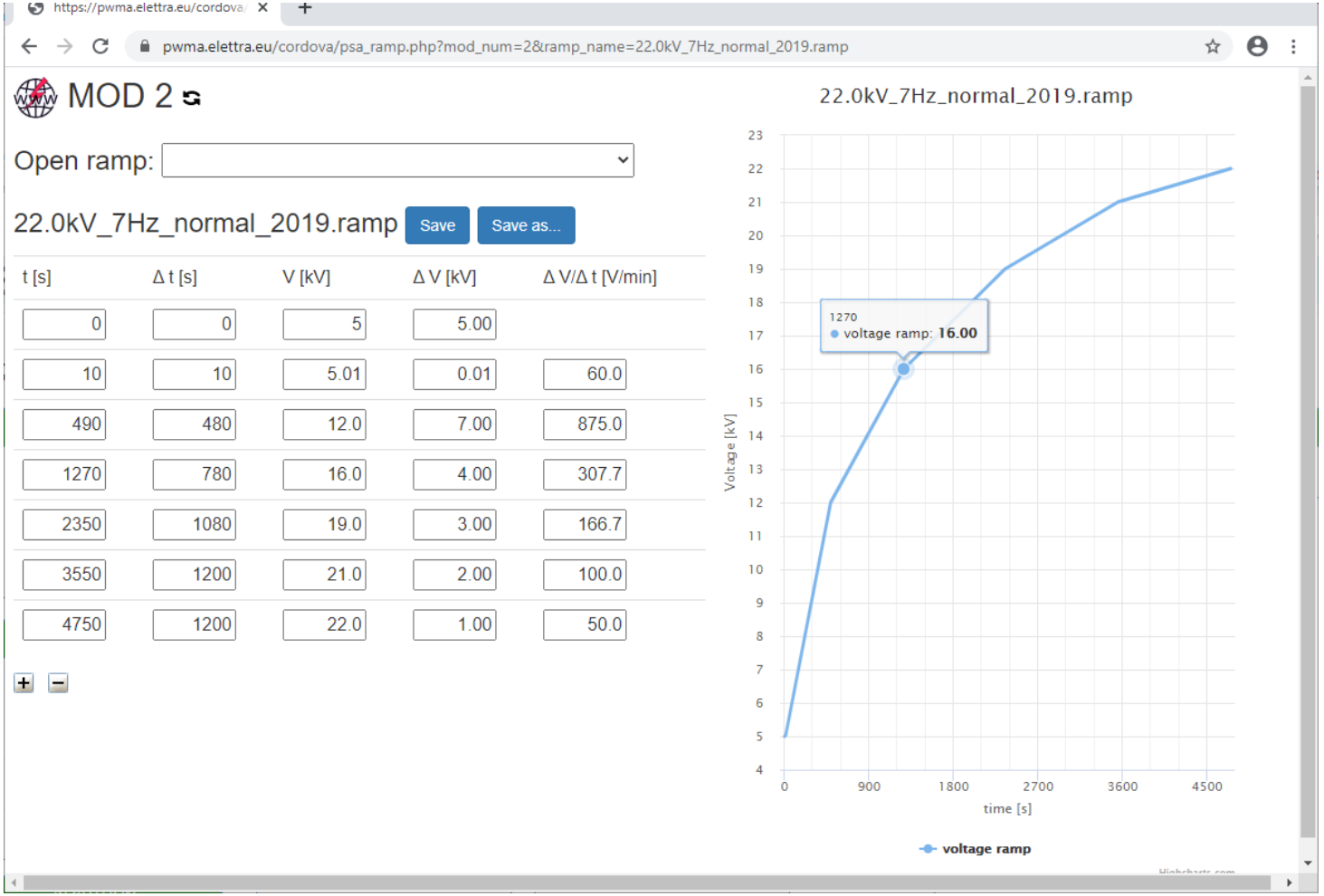


WEB INTERFACES

An other possible configuration could be a web server on any "intelligent" device, for example a websocket server has been implemented on a beaglebone

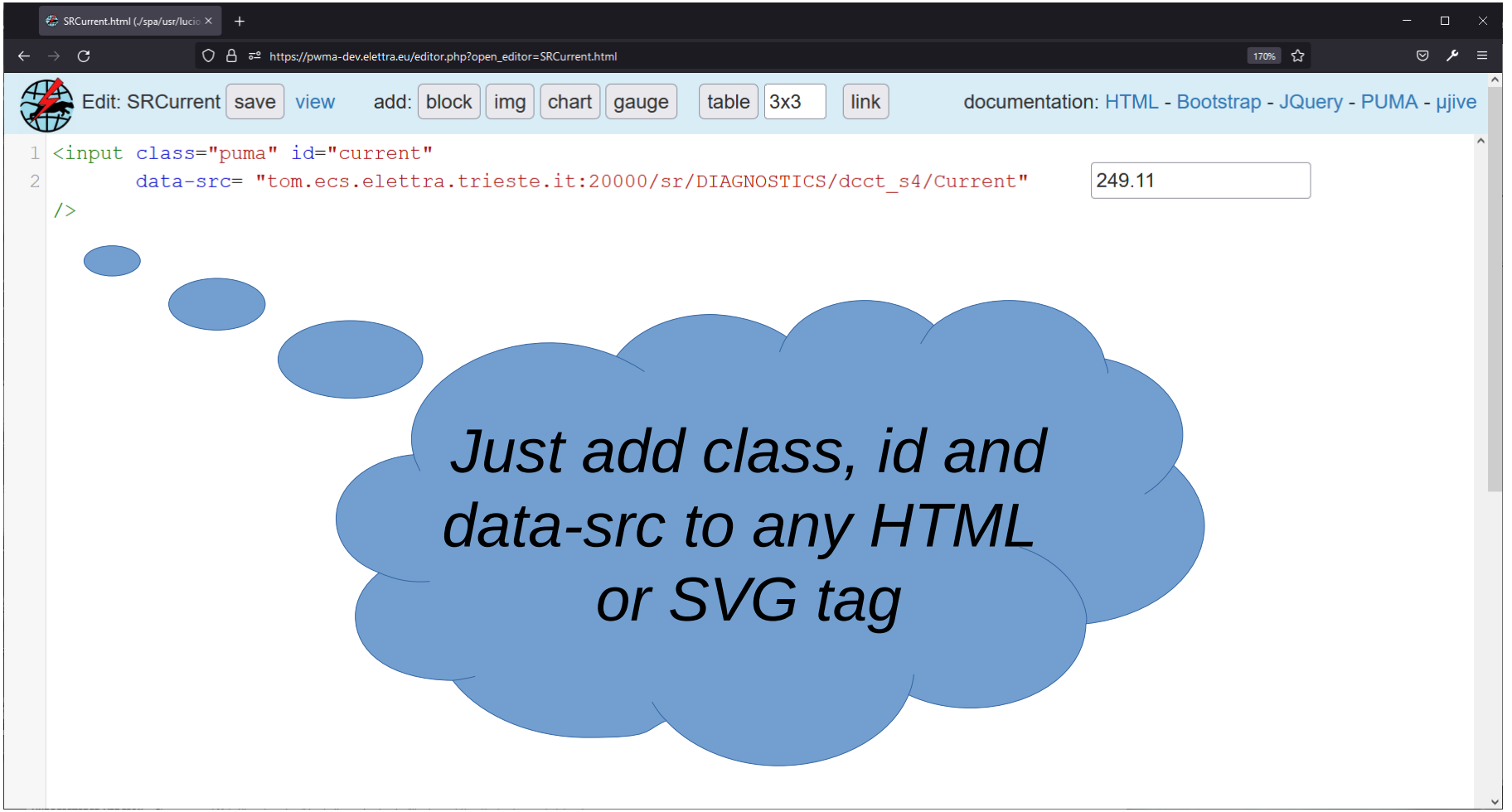


WEB INTERFACES



youtube.com/watch?v=9TI2S3mThmQ

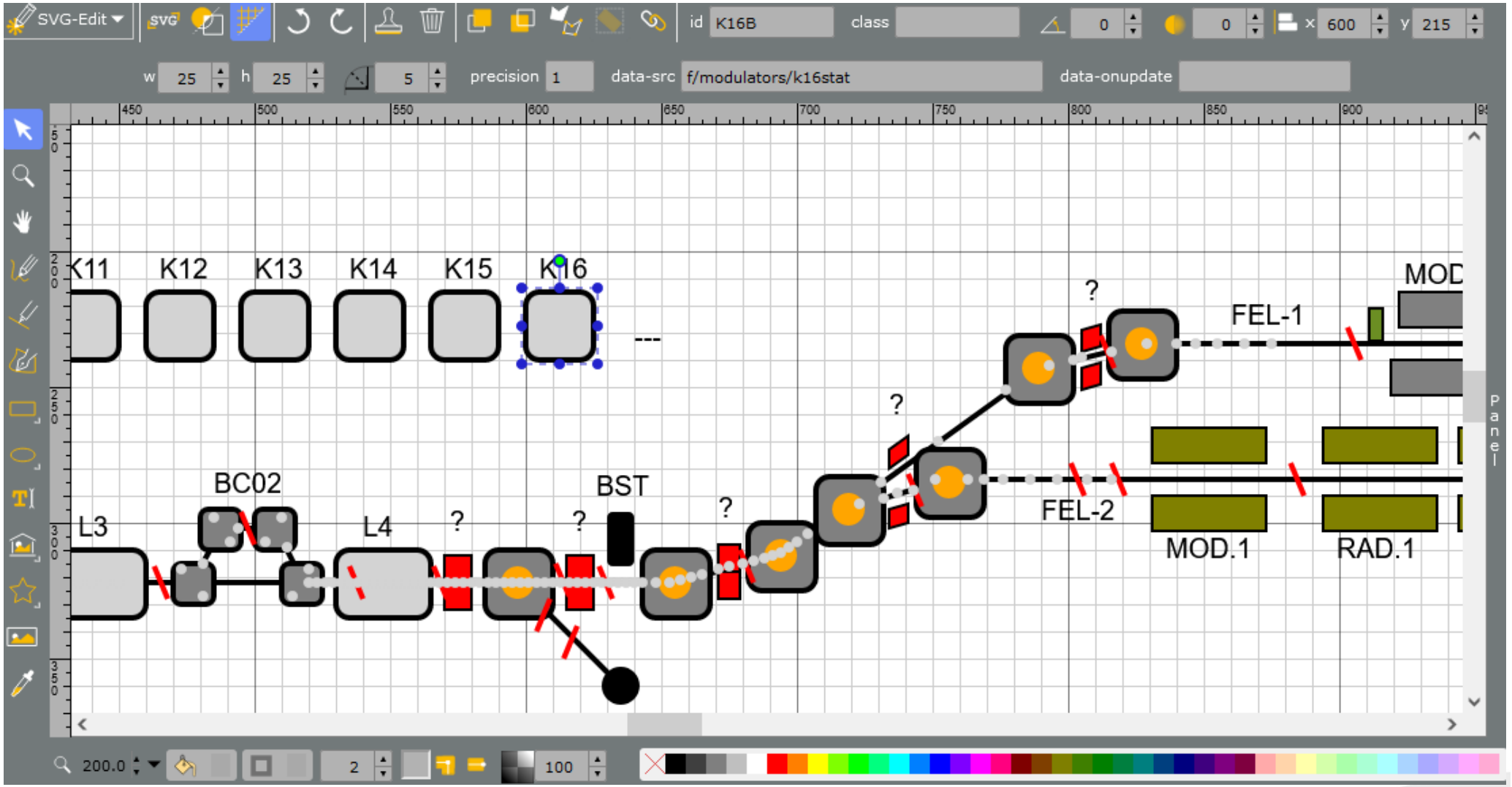
WEB INTERFACES



youtube.com/watch?v=psj4ZOz7ThA

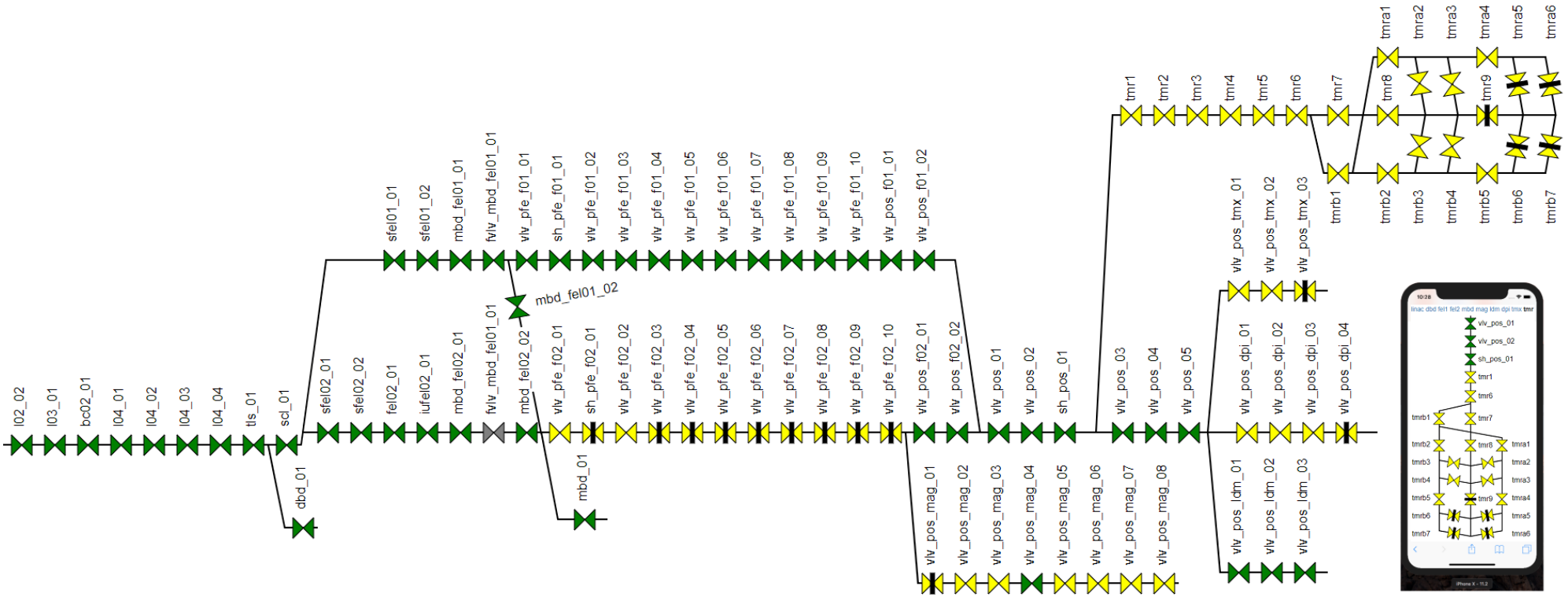
WEB INTERFACES

Elettra
Sincrotrone
Trieste



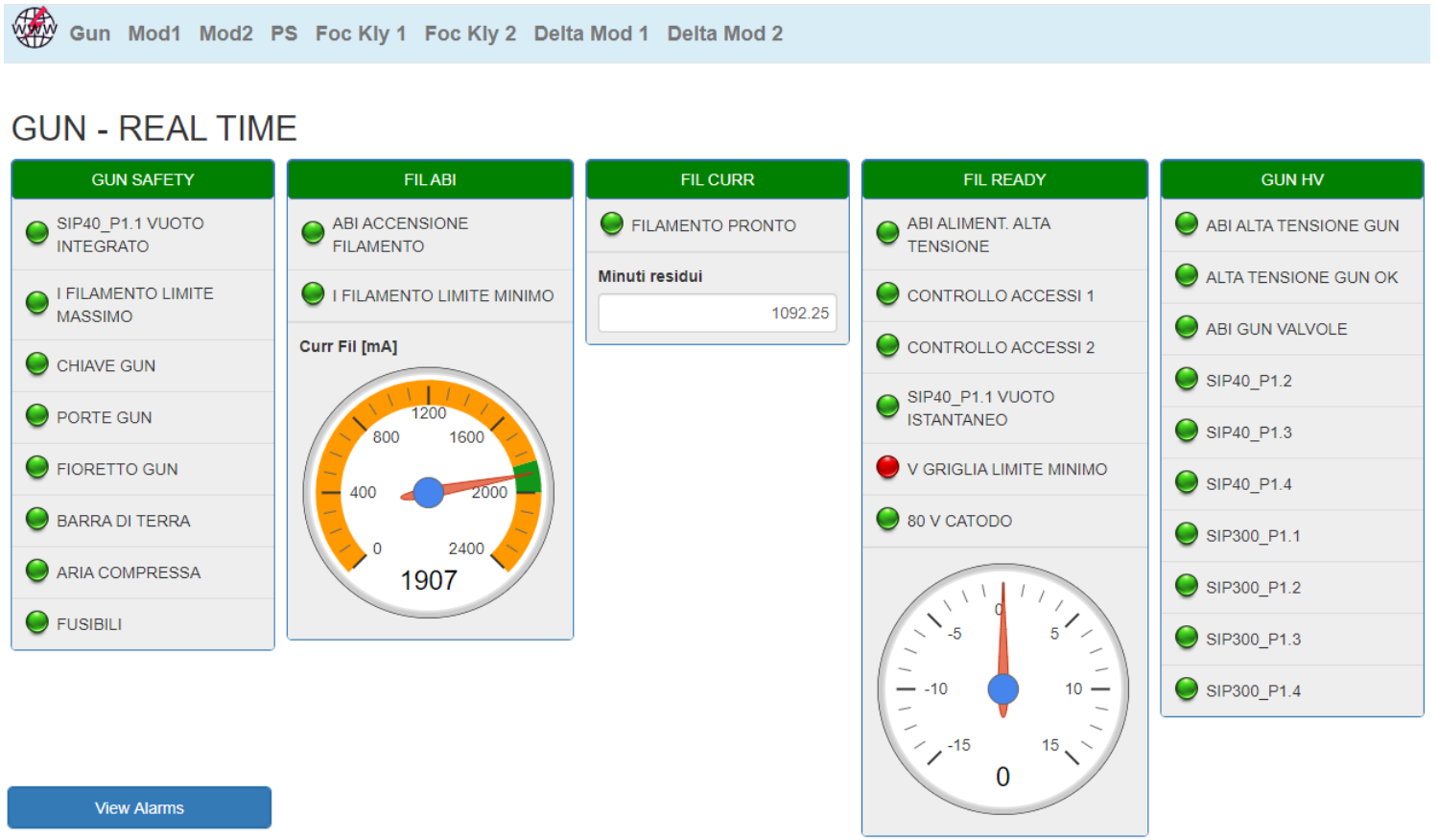
youtube.com/watch?v=vfe_YrOJvtA

WEB INTERFACES



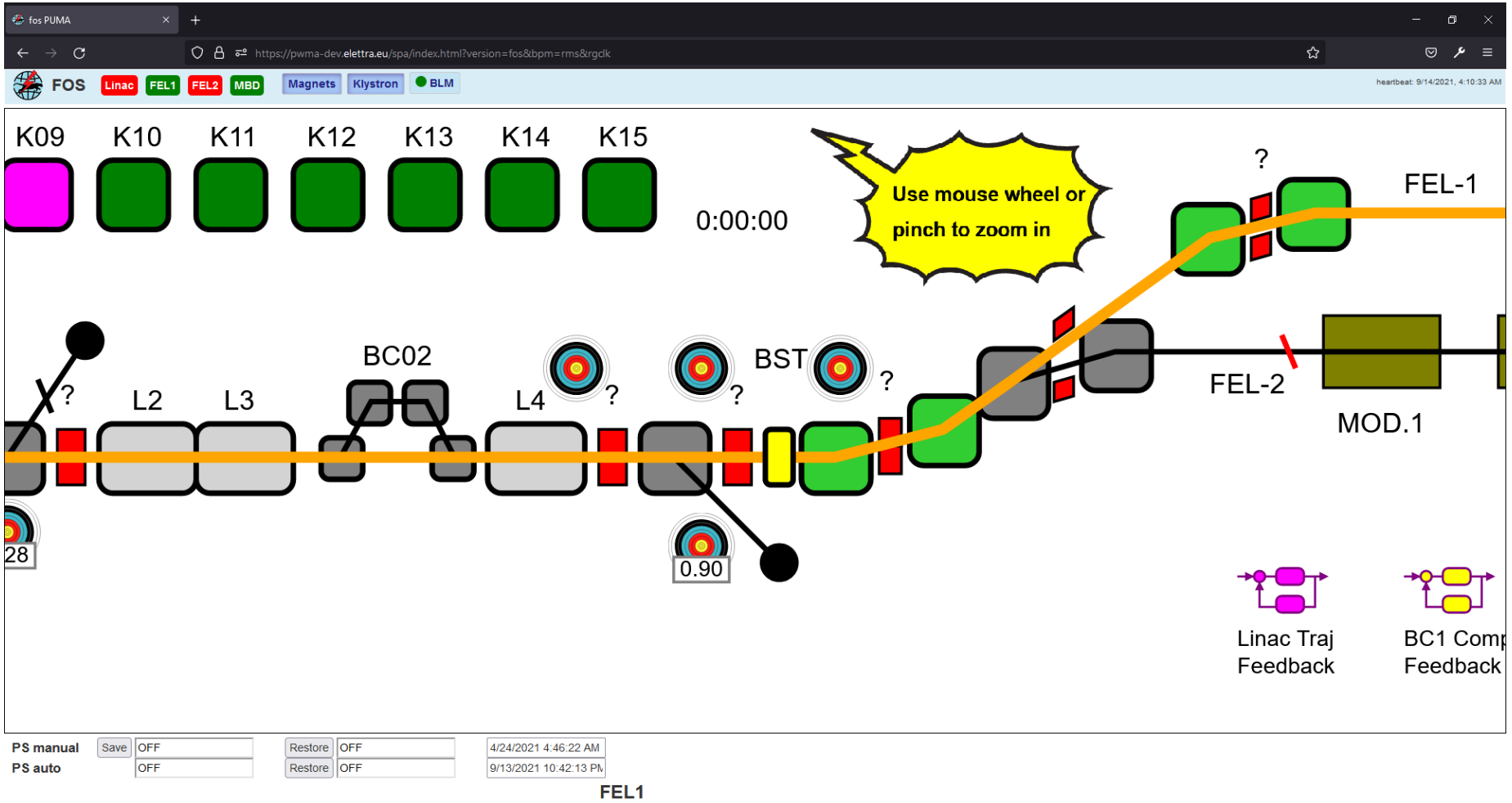
youtube.com/watch?v=I-3L0CaJ5X8

WEB INTERFACES



youtube.com/watch?v=9TI2S3mThmQ

WEB INTERFACES



youtube.com/watch?v=z7FUDB7w2aw

Elettra
Sincrotrone
Trieste

Thank you!

ISO 9001
OHSAS 18001
BUREAU VERITAS
Certification

Elettra
Sincrotrone
Trieste

www.elettra.eu