😠X(MLRPC) GON' GIVE IT TO YA🥊

Using Django and Embedded Python to Build and Test Electrical Substation Computers

WHO AM I?!

- Jason Devore
- Software Developer at NovaTech for 1.33 years
- BS CIS, MBA (😭) but coding since 10 y.o.
- PythonKC attendee for ~3 years
- Started with Python in 2005, Django in 2009
- Mostly wrote C#, VB, PHP, and JS for ~7 years until 2017
- Volunteer mentor at DjangoGirlsKC 2017, 2018
- **Loves Python and Django!**

NovaTech.

Build Smarter Infrastructure

ch is a U.S.-based supplier of automation and engineering solutions for the electric utilit
manufacturing industries. Our products and services are used by hundreds of utilities a
us Fortune 100 process manufacturers worldwide to simplify complex technical challeng
nal risk, and deliver the highest long term value.

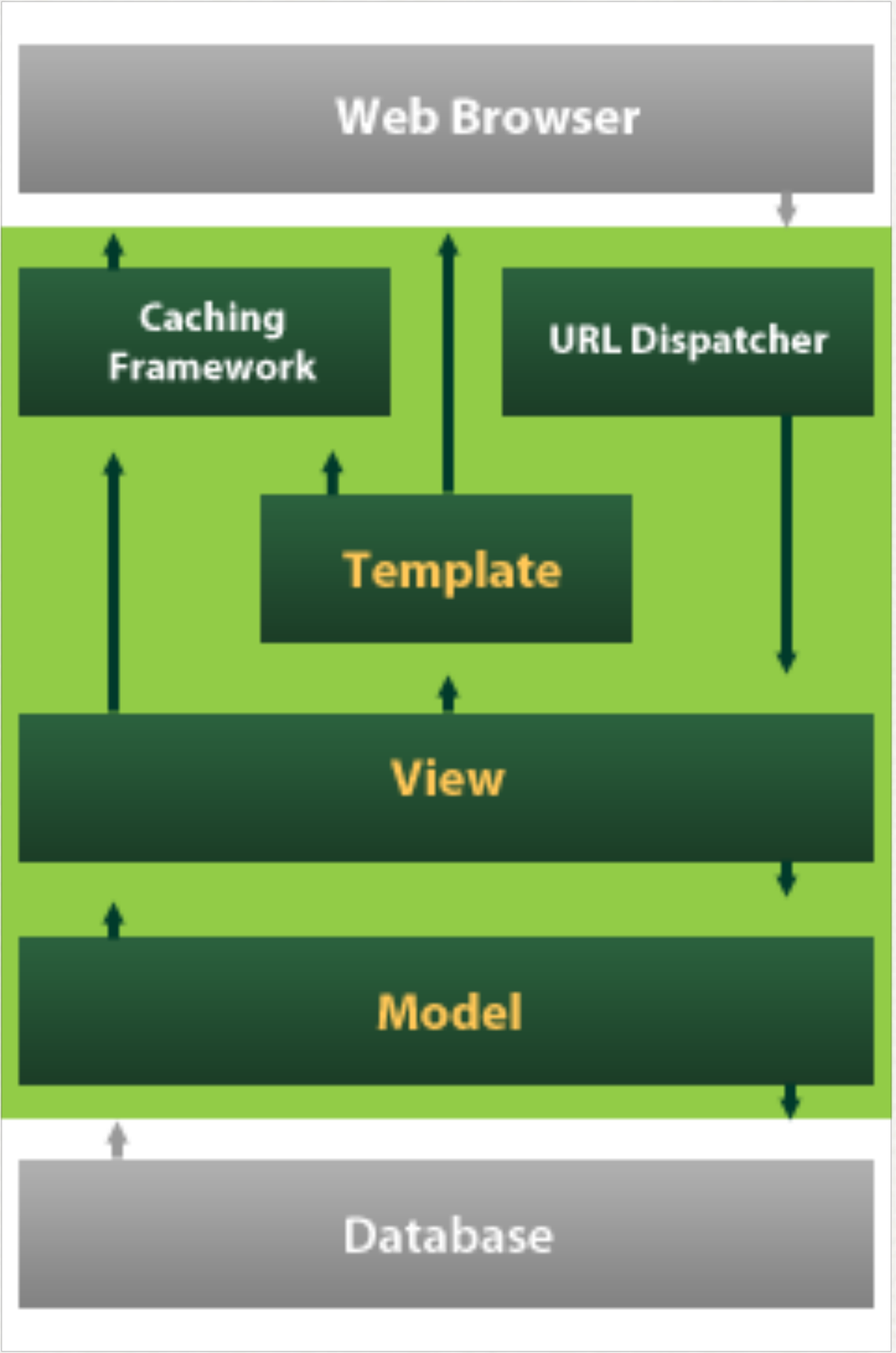
ee core product families – the Orion Substation Automation Platform, D/3 Distributed Co

ABOUT NOVATECH

- Electrical Engineering firm which makes computers and other devices for electric utility substations and manufacturers (oil, gas, beer)
- Custom hardware, software, and SaaS-like services (infrastructure, hosting, support)
- Orion devices automate power flow (e.g. when the electricity comes back on after a <1 sec outage)
- OS built from Open Embedded Linux
- Applications written in C, C++, Python
  - Support for hundreds of protocols (MODBUS, Protocol Buffers) and supports Lua for logic scripting in the field

CASE STUDY

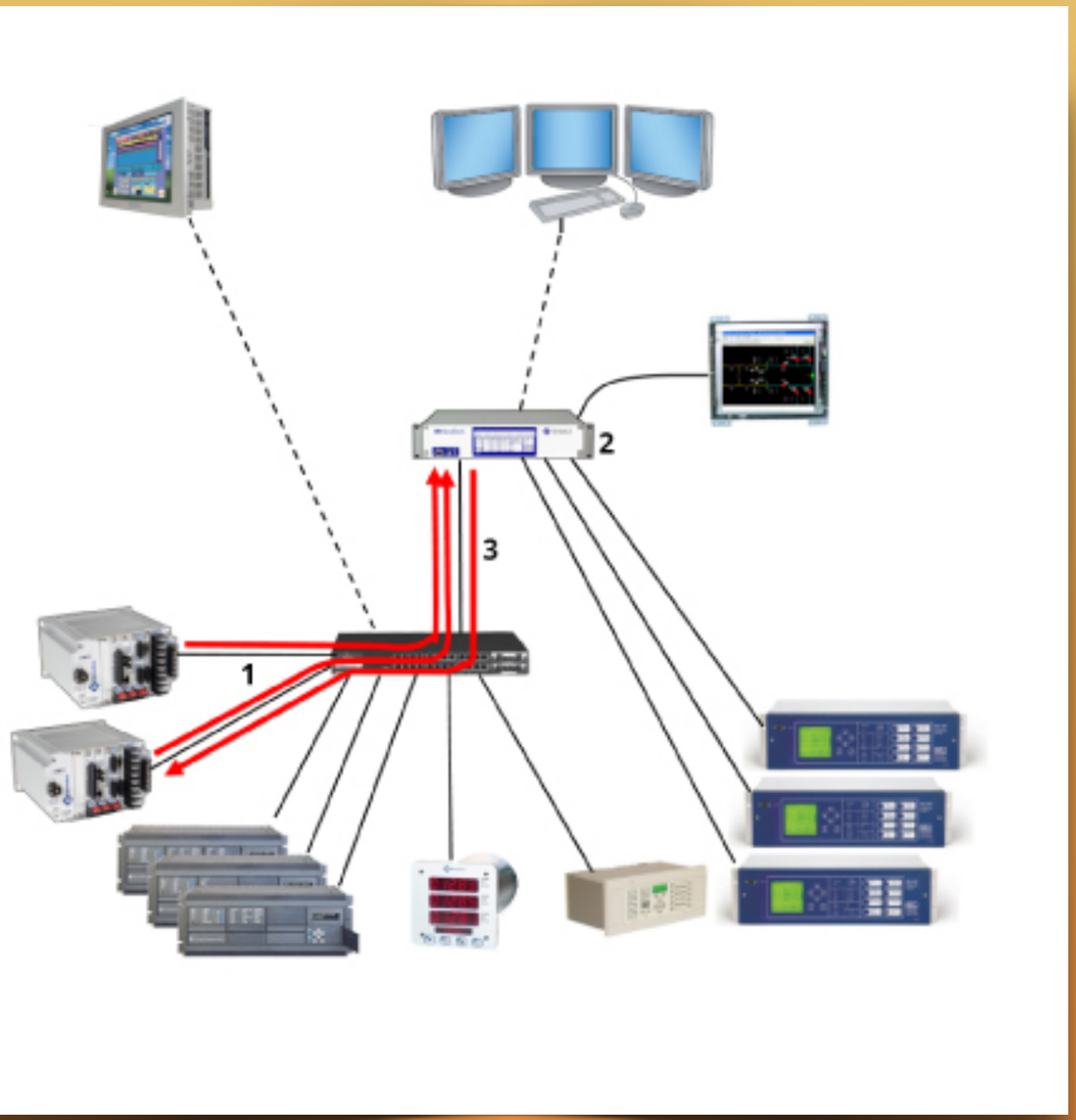
Using Django as an XMLRPC server and hardware testing suite



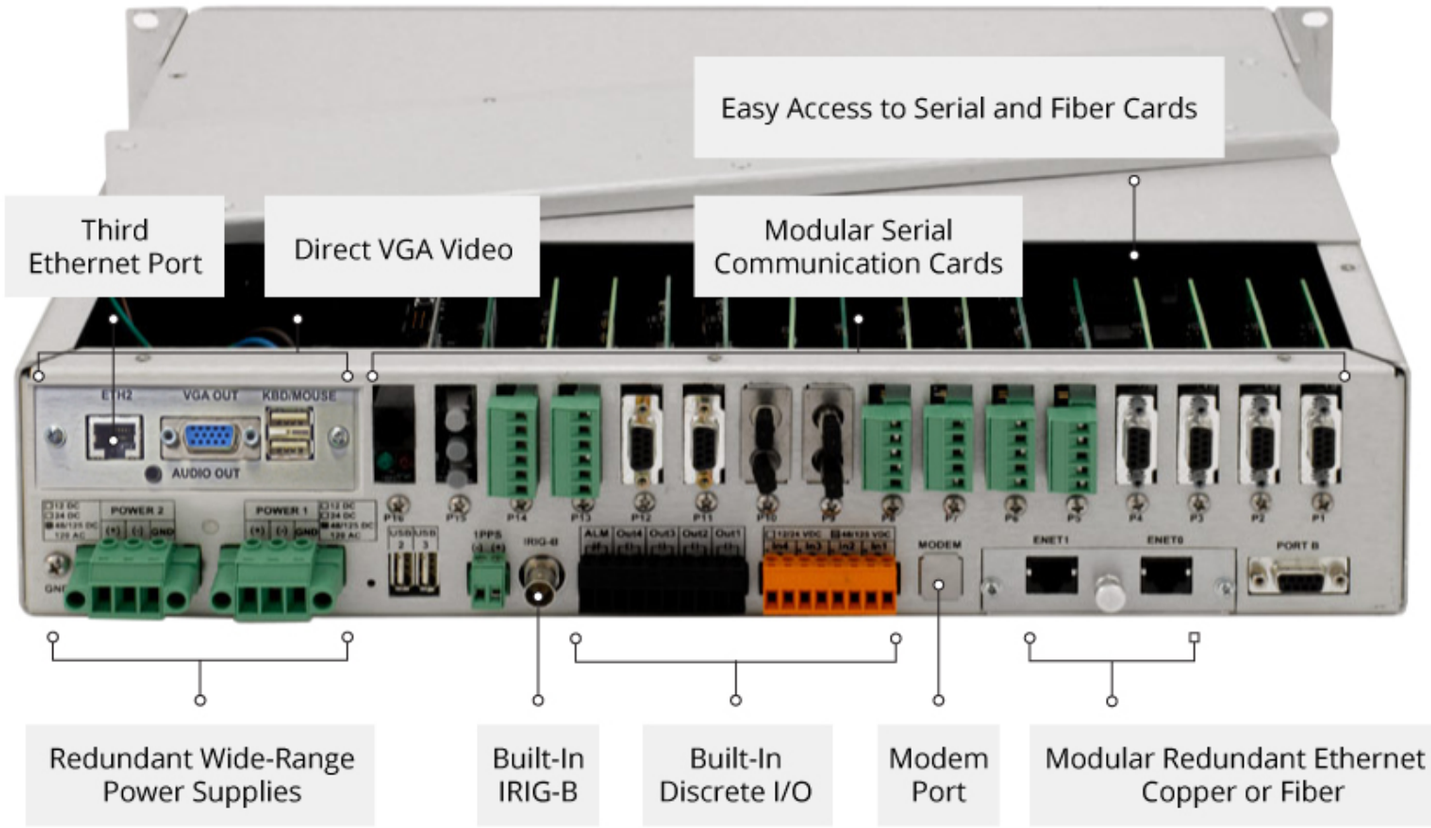
BUILDSYSTEM

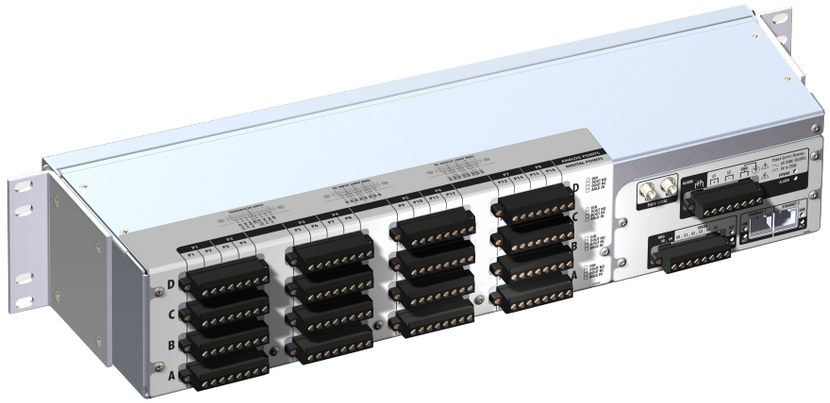
- "Web" Application
  - Docker containers (HTTP/ XMLRPC, MySQL, and TFTP) on CentOS server.
  - Segmented into development and production servers
  - Created and customized using Ansible scripts
  - Server accessed on TestStations
    - Restricted via hardcoded IP addresses

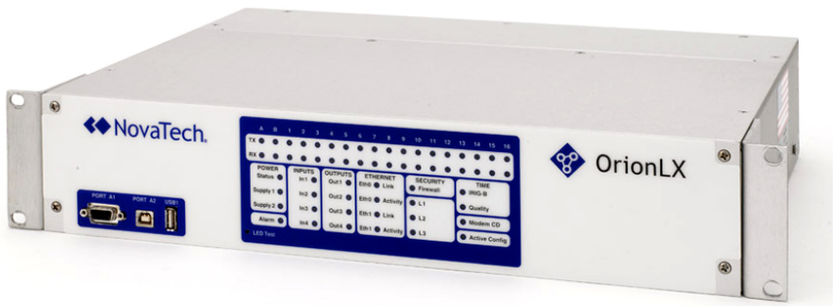
TEST STATIONS



- Work benches used to setup and test new and RMA Orion devices
- Master and slave Orions networked to LAN and to each other
- Slave device is “Device Under Test” (DUT) and tests include…
  - Ethernet
  - Modem
  - Clock (IRIG-B)
  - I2C (temperature)
  - USB, RS232
  - VGA and Audio (Alarm)
  - Electrical I/O with relays to open/close circuits







Example: OrionLX

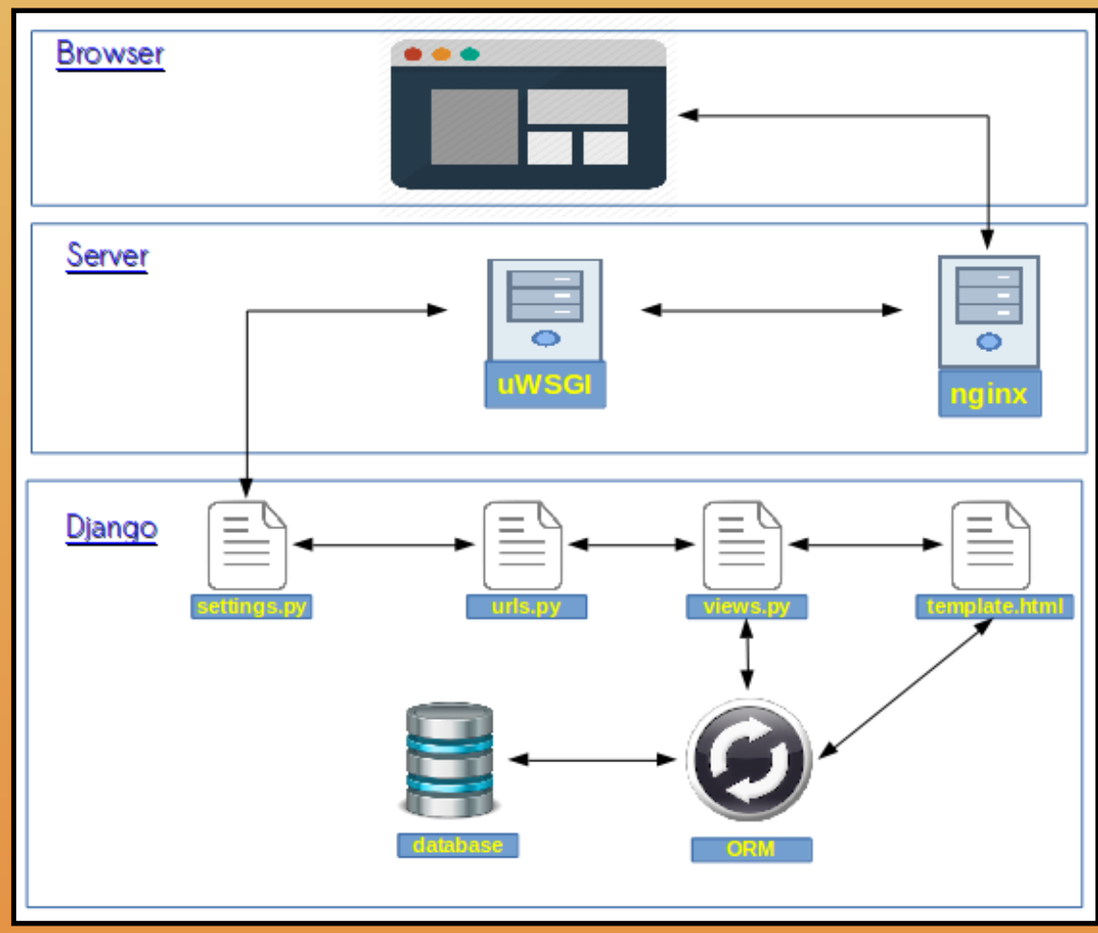
SPECIAL TECH PRESENTATION SURPRISE...

NO LIVE CODE EXAMPLES!

(Source code is property of NovaTech, LLC.)

But I have code mockups!

EVENT DIAGRAM OF THE TEST PROCESS



*BECAUSE WHO DOESN'T **LOVE** UML?*😏

Web Browser | Django Server | XMLRPC Server | Master Device | Slave Device

Request to view device record in test app

Looks up record via id and responds with rendered HTML and JSON via Django templates

Request to run test(s) on device under test (DUT)

Transforms request into commands and arguments

Serializes XML to send to master

Runs control tests on slave (timing, electrical IO, serial, etc.)

Serializes XML to send to slave

Runs tests on itself (memory, timing, RNG, etc.)

Deserializes XML results from master

Interprets results and converts data to XML

Reports back its results to master and to HTTP/XMLRPC server

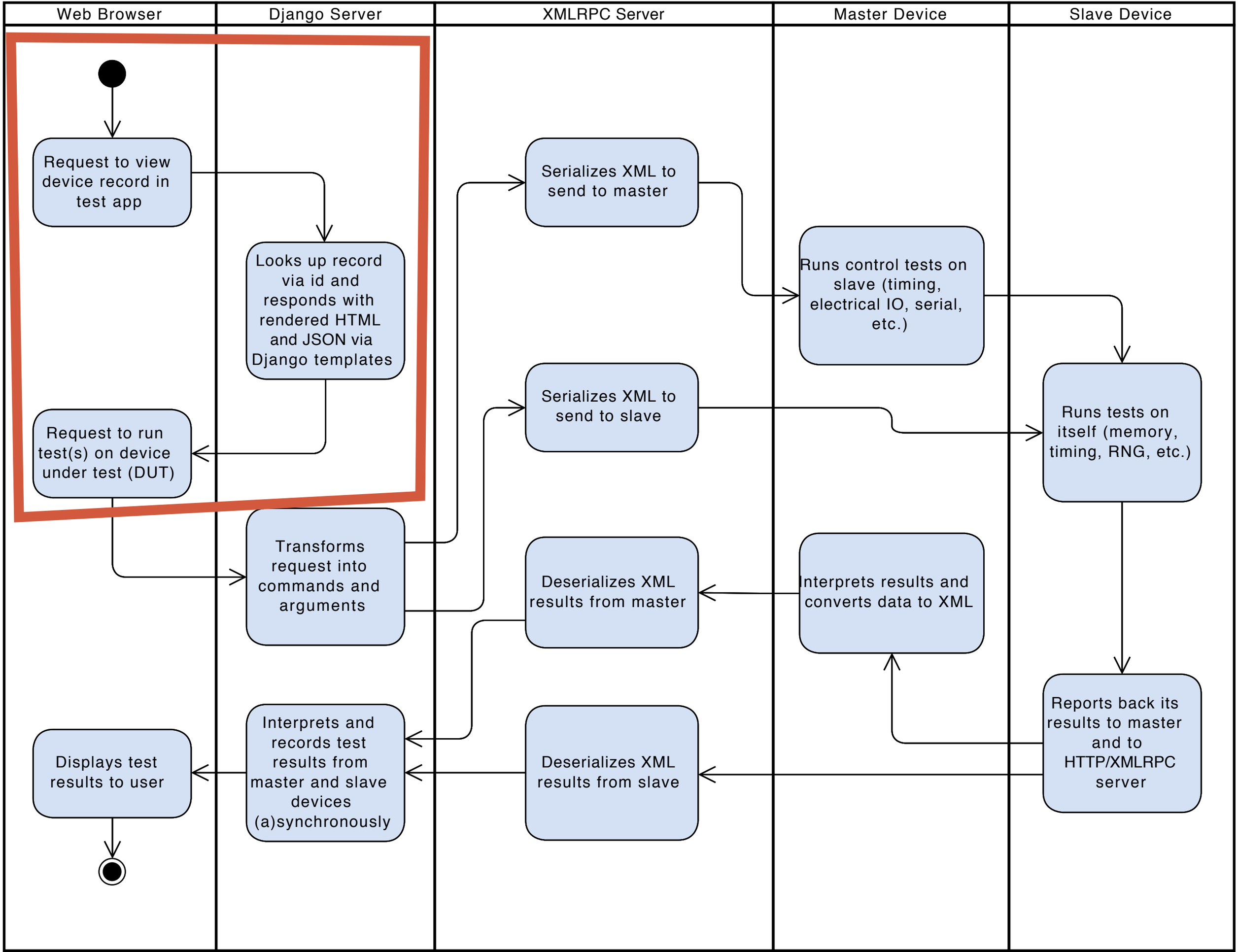
Interprets and records test results from master and slave devices (a)synchronously

Deserializes XML results from slave

Displays test results to user

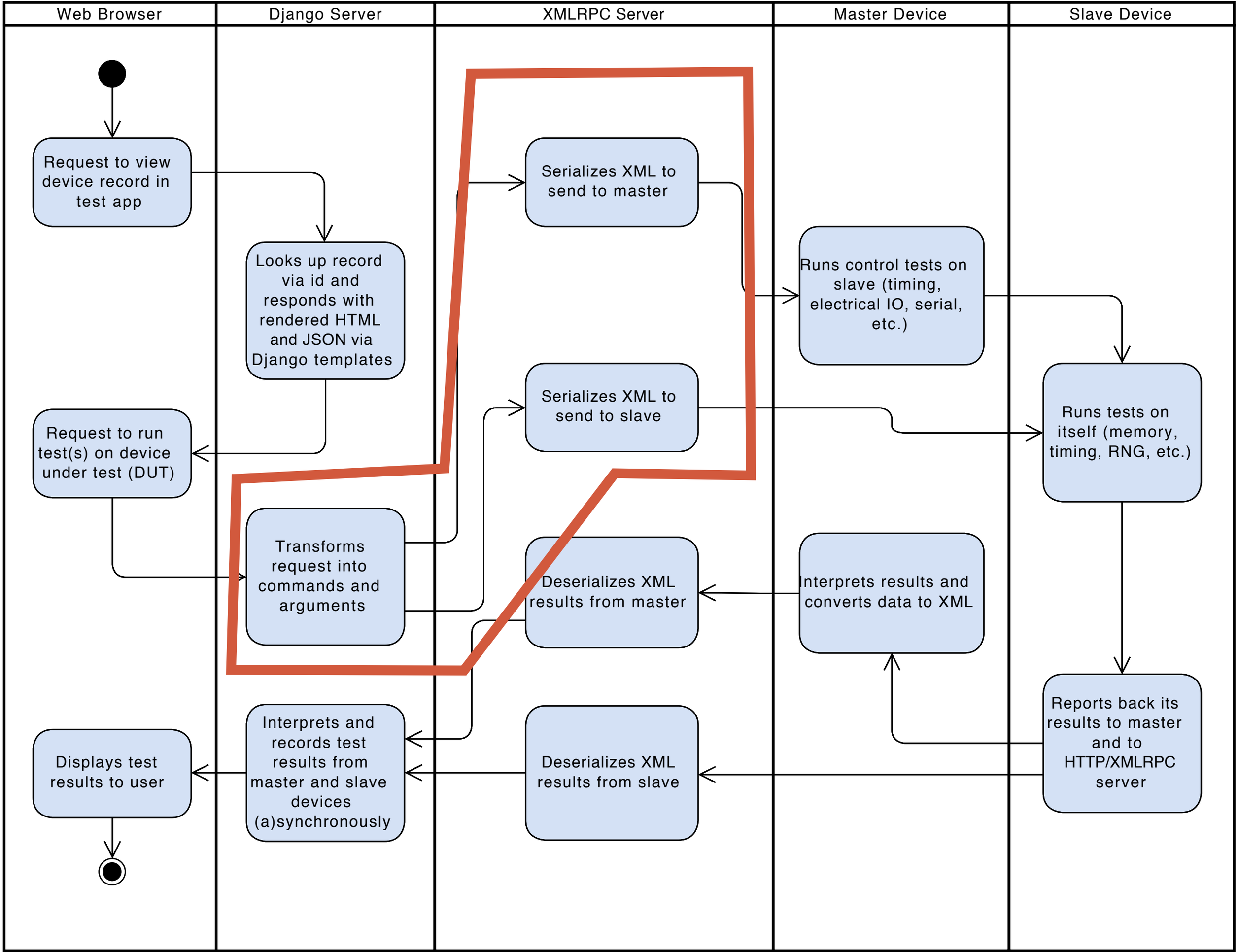
~/URLS.PY

```
from django.conf.urls import url
from django.contrib.auth import views
from testing import views

urlpatterns = [
    url(r’^testing/(\d+)$', testing.views.test_by_id),
]
```


Web Browser
Django Server
XMLRPC Server
Master Device
Slave Device
Request to view device record in test app
Looks up record via id and responds with rendered HTML and JSON via Django templates
Request to run test(s) on device under test (DUT)
Transforms request into commands and arguments
Serializes XML to send to master
Serializes XML to send to slave
Runs control tests on slave (timing, electrical IO, serial, etc.)
Runs tests on itself (memory, timing, RNG, etc.)
Interprets results and converts data to XML
Deserializes XML results from master
Deserializes XML results from slave
Reports back its results to master and to HTTP/XMLRPC server
Interprets and records test results from master and slave devices (a)synchronously
Displays test results to user

~/TESTING/VIEWS.PY

```
from devices.models import DeviceRecord

def test_by_id(request, device_id):
    # create request context
    context = RequestContext(request, {})

    # lookup device
    try:
        device = DeviceRecord.objects.get(pk=device_id)
    except:
        return Http404

    # get device's specific information for testing
    property_bag = device.get_properties()

    # render test app front-end with context
    return render_to_response(
        'testing/test_page.html',
        property_bag,
        context)
```


Web Browser
Django Server
XMLRPC Server
Master Device
Slave Device
Request to view device record in test app
Looks up record via id and responds with rendered HTML and JSON via Django templates
Request to run test(s) on device under test (DUT)
Transforms request into commands and arguments
Serializes XML to send to master
Serializes XML to send to slave
Runs control tests on slave (timing, electrical IO, serial, etc.)
Runs tests on itself (memory, timing, RNG, etc.)
Deserializes XML results from master
Interprets results and converts data to XML
Reports back its results to master and to HTTP/XMLRPC server
Displays test results to user
Interprets and records test results from master and slave devices (a)synchronously
Deserializes XML results from slave

~/TESTING/VIEWS.PY (CONTINUED)

```
from xmlrpc_server import tests_available
from json import dumps

def run_a_test(request, test_to_run):
    # create request context, do device lookup, etc.
    …

    # convert JSON data into Python dict
    kwargs = {}
    for k, v in request.POST.items():
        kwargs[k] = _fictional_value_conversion(v)

    # find and run test
    xmlrpc_func = getattr(tests_available, test_to_run, None)
    if xmlrpc_func:
        test_info = xmlrpc_func(**kwargs)

    # present results to jQuery-based front-end script
    return HttpResponse(dumps(test_info))
```

What is XML-RPC?

- XML-RPC implements RPC using open Web standards.
- Data is encoded ("marshalled") in XML using a special DTD for XML-RPC.
- The RPC call is made using HTTP.
- The application exporting the function must implement or be attached to a HTTP server.

Web Browser | Django Server | XMLRPC Server | Master Device | Slave Device

Request to view device record in test app

Looks up record via id and responds with rendered HTML and JSON via Django templates

Request to run test(s) on device under test (DUT)

Transforms request into commands and arguments

Serializes XML to send to master

Serializes XML to send to slave

Runs control tests on slave (timing, electrical IO, serial, etc.)

Runs tests on itself (memory, timing, RNG, etc.)

Interprets results and converts data to XML

Reports back its results to master and to HTTP/XMLRPC server

Deserializes XML results from master

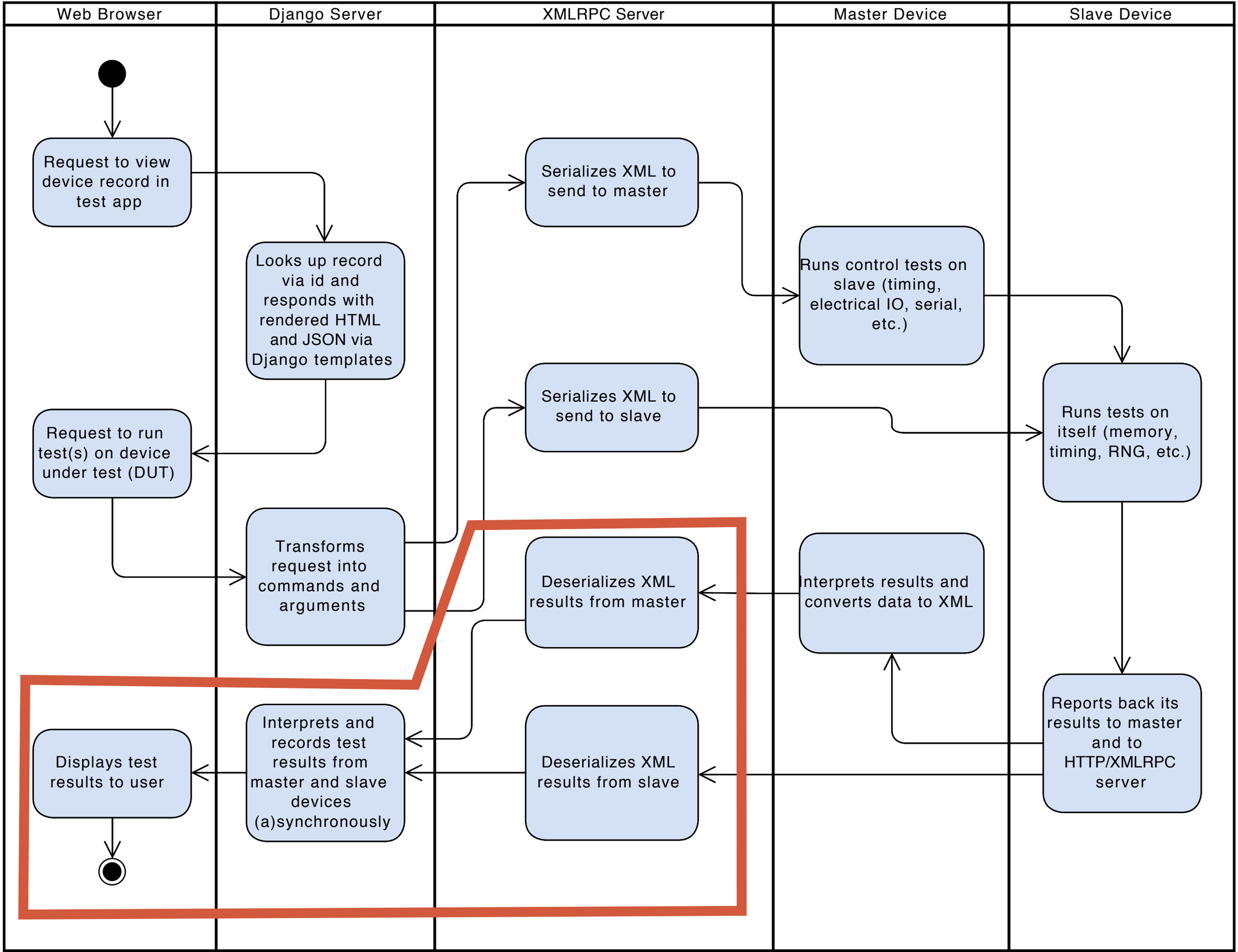
Deserializes XML results from slave

Interprets and records test results from master and slave devices (a)synchronously

Displays test results to user

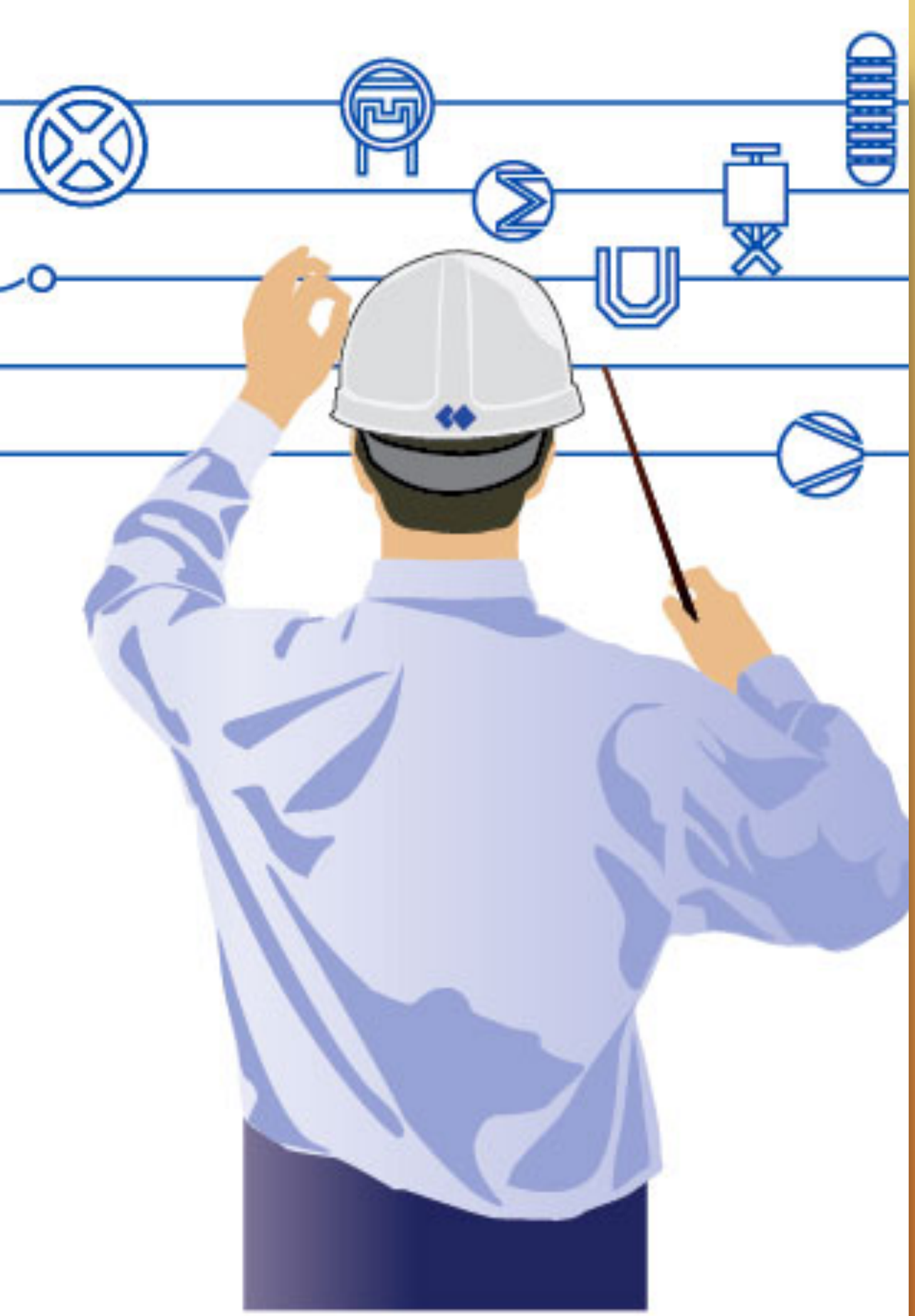
~/XMLRPC_SERVER/TESTS_AVAILABLE.PY

```
From testing.models import TestRecord
from xmlrpclib import dumps, loads

def irig_test(targets, data, cli_args, etc):
    # Convert kwargs dict into XMLRPC
    command_set = dumps(kwargs)

    # Record intended directions for device
    input = _create_input_(command_set)

# Send to device (steps skipped for brevity)
output = _run_xmlrpc(input)

# Record actual events from device
test_data = TestRecord(xml_results=loads(output))

# Save to db
test_data.save()

# Send data back to front-end
return test_data.convert_to_json()
```


Web Browser
Django Server
XMLRPC Server
Master Device
Slave Device
Request to view device record in test app
Looks up record via id and responds with rendered HTML and JSON via Django templates
Request to run test(s) on device under test (DUT)
Transforms request into commands and arguments
Serializes XML to send to master
Runs control tests on slave (timing, electrical IO, serial, etc.)
Serializes XML to send to slave
Runs tests on itself (memory, timing, RNG, etc.)
Deserializes XML results from master
Interprets results and converts data to XML
Reports back its results to master and to HTTP/XMLRPC server
Interprets and records test results from master and slave devices (a)synchronously
Deserializes XML results from slave
Displays test results to user

WHAT ABOUT THE EMBEDDED PYTHON?🐍

- Some Orion computers are low resource
  - (No SSD, 256mb RAM)
- OS customized to be more efficient
  - Removed man pages and unneeded tools:
    - No vim, but vi
    - No htop, but top
- Python 2.7.x included
- Test Client application receives XMLRPC and runs shell commands
  - Too big to cover in this presentation… until next time!



CONCLUSION

- Roadmap for further development:
  - Convert to Django 2.x/ Python 3.x
  - Refactor business logic on the back-end
  - Use React to generate HTML elements on page to control tests
  - Practice Test-Driven Development!😖

QUESTIONS, COMMENTS, SUGGESTIONS?

(Ha;lp plox)

~BAI~

Thank you, PythonKC and Lawrence, KS!

jason.devore@pm.me