Apex II + FORTE: Data Acquisition Software for Space Surveillance

Vladimir Kouprianov

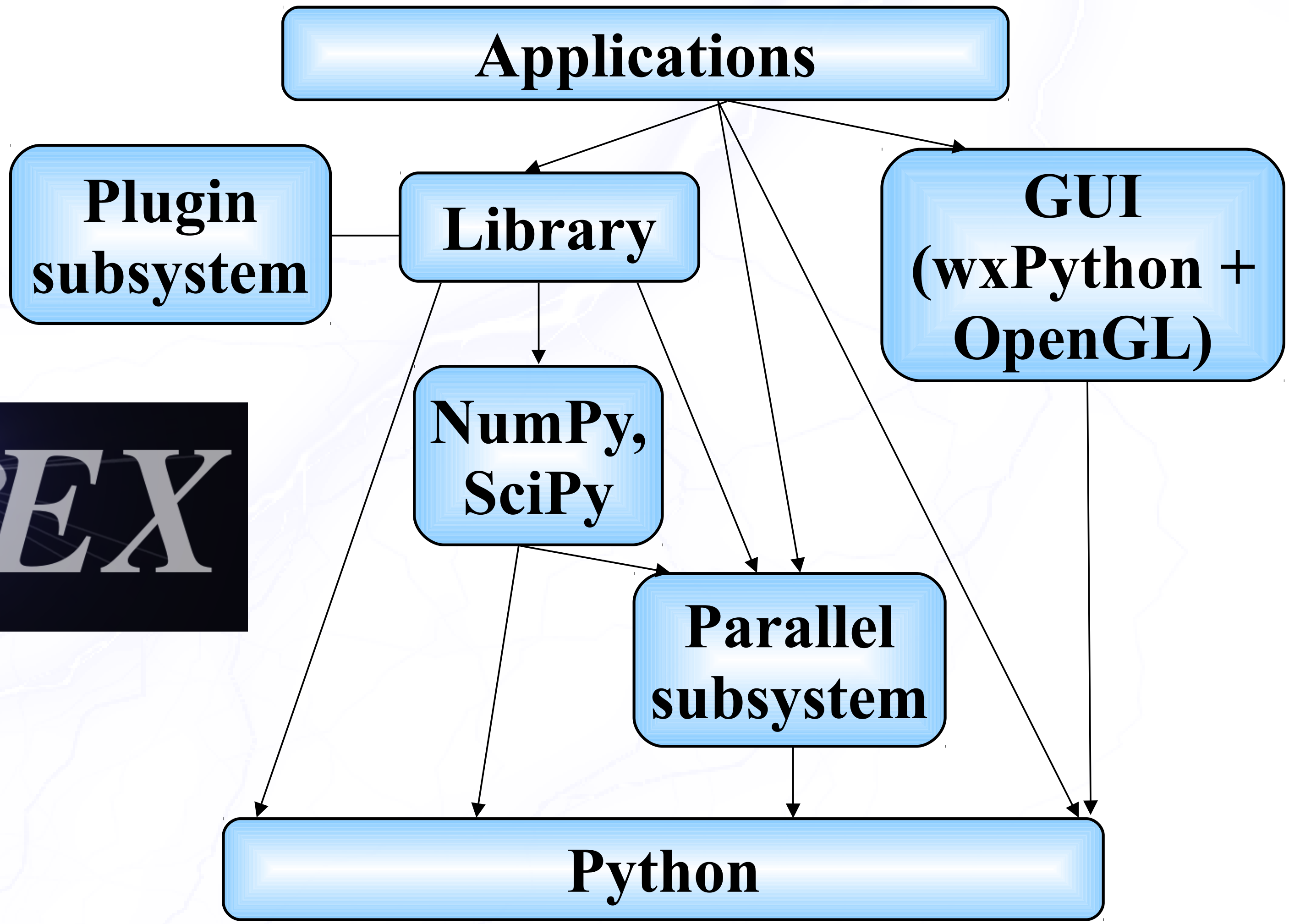
Central (Pulkovo) Observatory
Russian Academy of Sciences
Saint Petersburg

V.K@BK.ru

Apex II: Motivation

- No general-purpose packages suitable for high-precision **astrometry** were available
- Demand for high **flexibility** due to the diversity of instruments and tasks in ISON
- Demand for high **accuracy** for fast wide-FOV sensors and undersampled images
- Fully **unattended** operation, scripting
- Existing packages (IRAF, MIDAS, IDL, …) — deprecated software technologies, hard to adapt, closed source

Apex II: Outline



Apex II: Recent Developments

Sensor performance depends on the fast and reliable detection of space objects

1. GEO survey mode: ~3–4 frames per minute
   Before 2010: 10s per 1K×1K frame for image analysis — real time broken even for 2K×2K
   → **parallel computing** for fast image analysis without loss of sensitivity
2. Fast apparent motion of Earth-orbiting objects with respect to background stars → use **mathematical morphology** for fast and robust detection without loss of sensitivity

Apex II: Recent Developments
1. Parallel Computing

- Old *Apex II* parallel subsystem: relies on OS processes → utilizes multiple CPUs → can process many images in parallel
- New parallel core: relies on OpenCL → utilizes CPUs and GPGPUs (now under testing) → accelerating pixel operations, object detection and measurement
- Works on the KIAM cluster: detection of faint space debris beyond the sensitivity limit (*Yanagisawa et al.*, Proc. 4th European Conf. Space Debris, Darmstadt, 2005)

Apex II: Recent Developments 2. Mathematical Morphology

Traditional approach to detection of fast-moving objects:

- Difference images → noise, false detections.
- Compare coordinates of all detections → bad performance.
- Both are unsuitable for space surveillance.

Our approach: binary morphological filtering — distinguish Earth-orbiting objects from field stars by shape (*Kouprianov*, Adv. Space Res., 2008, **41**(7), 1029–1038):

- Can detect objects in a single frame.
- Fast.

FORTE

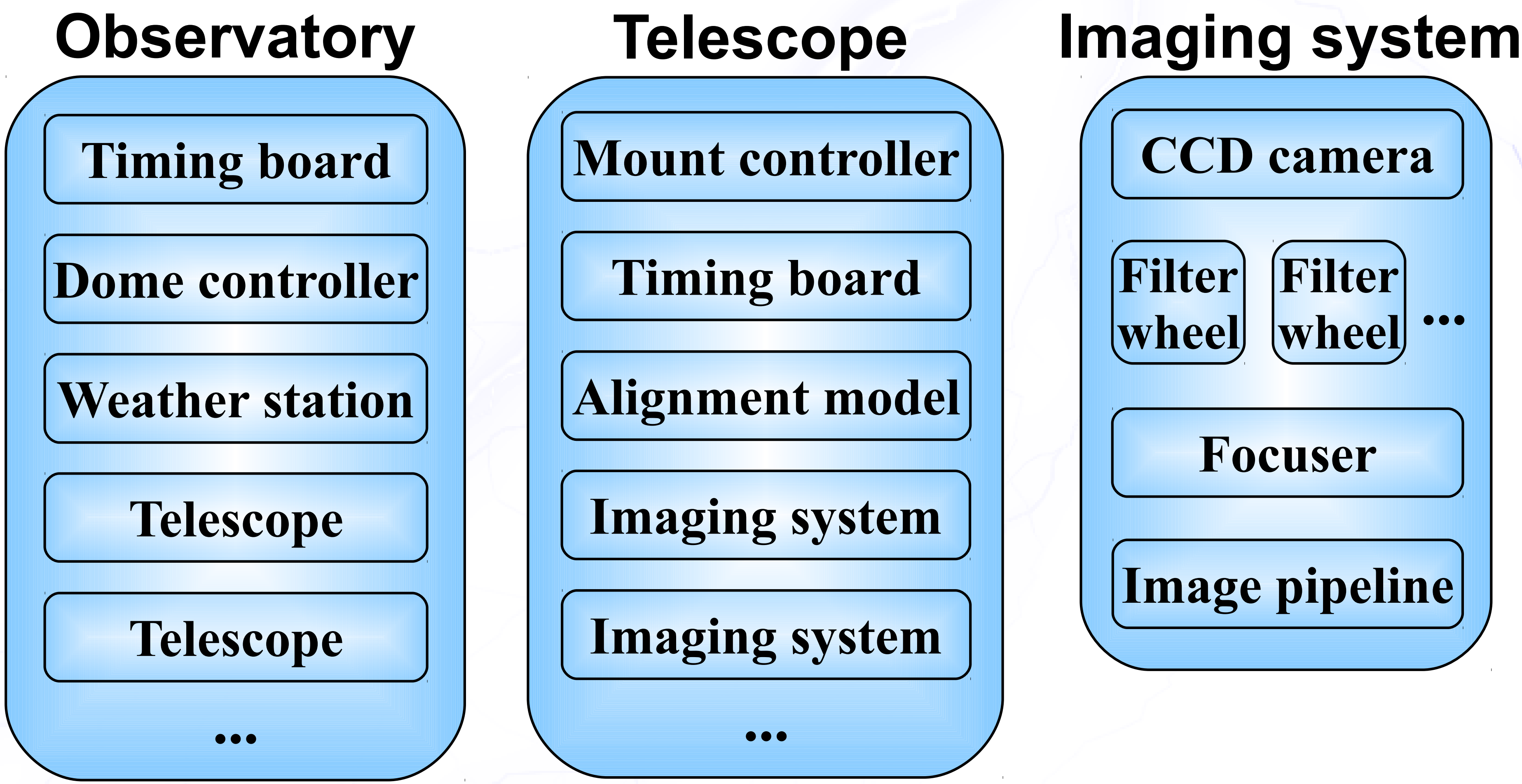
Facility for ***O***perating

Robotic ***T***elescope ***E***quipment

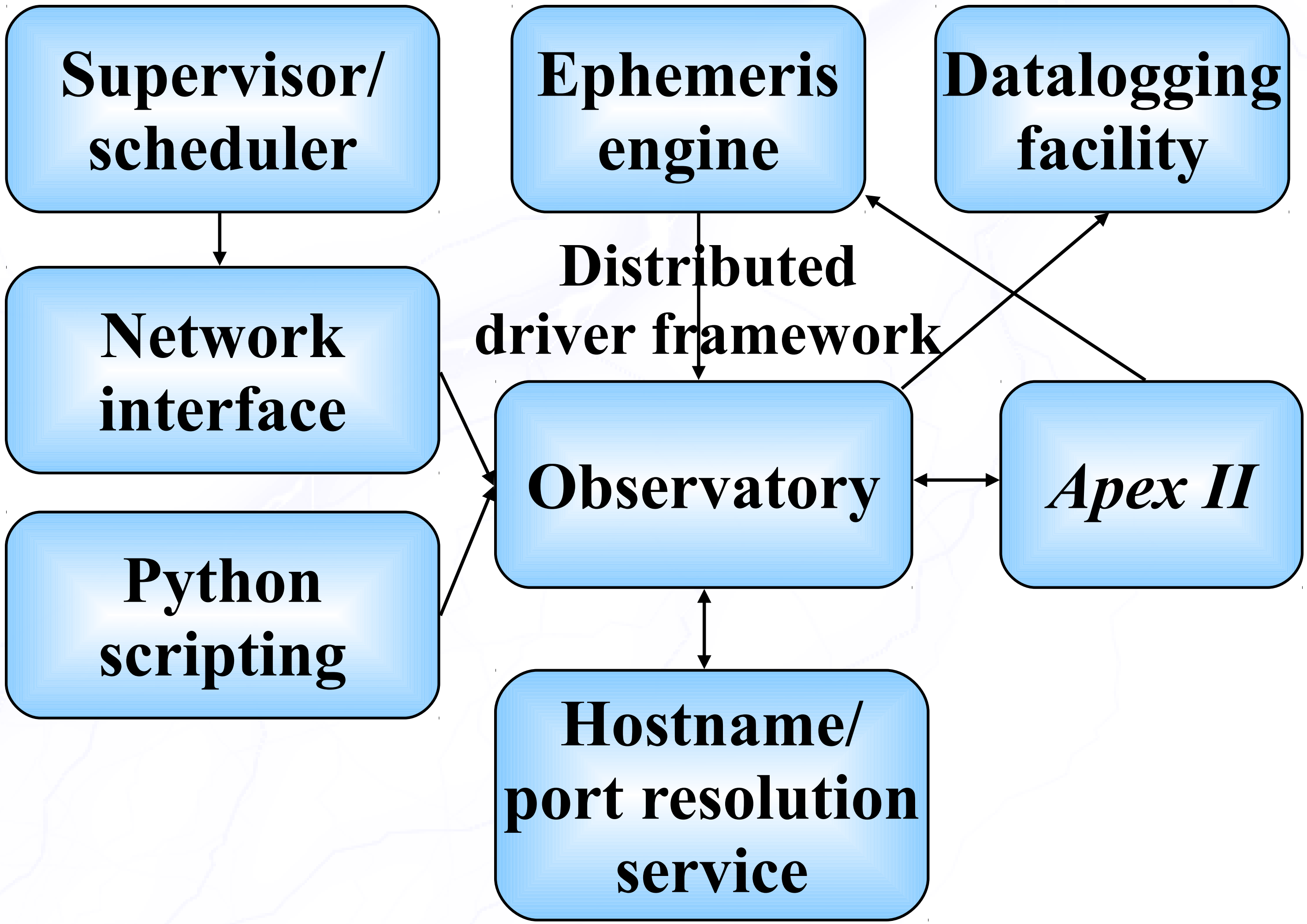
- Written in Python
- Tight integration with *Apex II*
- Distributed
- Flexible
- Extensible

f

FORTE: Abstract Approach



FORTE: Basic Structure



Hardware states

- **offline** – ready for poweroff (TE cooling disabled, scope in safe position, ...)
- **suspend** – long delay in operation, e.g. due to weather conditions (only hardware sensors working, roof closed, ...)
- **standby** – ready for normal operation (TE cooling stabilized, roof open, ...)
- **online** – system is fully operational

Types of commands

- **Synchronous** – simple actions (e.g. hardware sensor queries) that require immediate reply
- **Asynchronous** – complex long-lasting actions (e.g. pointing) that expect premature abort:
  - exclusive – one task of the given kind at a time
  - nonexclusive (e.g. image analysis)

Asynchronous actions are also implicitly used by more complex actions for more efficient operation by doing several sub-commands in parallel.

Remote Procedure Call

- **Internal** – communication between devices working on the same or on separate TCS workstations; transport based on Python serialization over TCP/IP
- **External** – high-level TCS control via the Observatory interface; transport based on XML packets over TCP/IP

FORTE RPC supports transparent remote actions on any Python data structures, including IPC synchronization primitives.

XML RPC: get scope position

Query:

```
<call>
  <target>scope.mount</target>
  <name>get_hadec</name>
  <args></args>
</call>
```

Response:

```
<result>
  <tuple>
    <item>E</item>
    <item><float>0.12345</float></item>
    <item><float>-1.23456</float></item>
  </tuple>
</result>
```

XML RPC: observe an object

```
<task>
  <target>scope</target>
  <name>observe</name>
  <args>
    <arg name=”target”>90</arg>
    <arg name=”ephem_provider”>apex</arg>
    <arg name=”apex_catalog”>EPOS01</arg>
    <arg name=”tracking”>auto</arg>
    <arg name=”exposures”><dict>
      <item name=”texp”><float>30</float></item>
      <item name=”nexp”><int>2</int></item>
      <item name=”filter”>R</item>
    </dict></arg>
  </args>
</task>
```

Python Scripting

```
from forte.net.xml_rpc import *
o = XMLRPCProxy('observatory', server_address =
    ('192.168.1.22', 2011))

o.start_task('set_state', 'online').join()
print 'Coordinates:', o.scope.mount.get_hadec()
t = o.scope.start_task('observe', target='90',
    ephem_provider='apex', apex_catalog='EPOS01',
    tracking='auto', exposures=dict(texp=30,
        nexp=2, filter='R'))
print t.is_alive()   # still running?
t.abort()            # abort observation
t.join()             # wait until finished
```

Image Pipelines

- Run in parallel, sequentially, or in any combinations
- Fully customizable by the user
- Can be dynamically overridden for each exposure
- Run asynchronously just after image acquisition
- Examples: data storage, image examination, image analysis

Events (under construction)

- Generated by all TCS components at different important moments (state transitions, change of conditions, end of exposure, ...)
- Customizable event parameters
- Customizable actions assigned to each event
- Examples: stand by if too cloudy; suspend if humidity above 95%; re-focus if ambient temperature changes by 10º

Datalogging

- Uses built-in Python logging facility
- Backends: disk files (incl. auto-rotation), syslog daemon, NT event log, sockets
- Customizable logging destinations and formats separately for every logging channel
- Collect hardware statistics (shutter cycles, motor revolutions, voltages, …) for scheduling maintenance

Other features

- Automatic focusing and scope alignment
- Sophisticated soft limits with auto-recovery
- Support for various hardware timing modes
- High-level web interface with modules for automatic scheduling of different kinds of observations, incl. GEO survey mode
- Interoperability with *Apex II* via web interface: new images are placed on the processing queue; all detections, incl. uncorrelated tracks, are displayed immediately

New ISON Measurement Format


```
<meas>
  <sensor>12345</sensor> <id>12001002</id>
  <filename>/.../25.20120101T001122345.fit</filename>
  <utc>2012-01-01T00:11:22.345678</utc>
  <ra_j2000>1.2345678</ra_j2000>
  <dec_j2000>-2.345678</dec_j2000>
  <ra_j2000_error>0.123</ra_j2000_error>
  <dec_j2000_error>0.234</dec_j2000_error>
  <mag>15.678</mag> <mag_error>0.05</mag_error>
  <snr>5.678</snr> <x>123.456</x> <y>789.012</y>
  <x_error>0.0234</x_error> <y_error>0.0345</y_error>
  <vel_ha>-0.123</vel_ha> <vel_dec>1.234</vel_dec>
  <length>39.7</length> <width>2.5</width> <rot>43</rot>
      ...
</meas>
<meas>
  ...
</meas>
```

First ROSCOSMOS Sensor

Conclusions

- Among other factors, performance of ISON sensors was formerly limited by non-realtime image analysis and its weak integration with hardware control loop.
- During the years 2010–2011, *Apex II* parallel subsystem and extensive use of mathematical morphology for object detection lead to much higher performance of data reduction.
- A new observatory control system, *FORTE*, is tightly integrated with *Apex II* and is expected to significantly improve the ISON space debris discovery rate.

Thank you!