SeisComP, its data model and how it is used in Python

Joachim Saul, Andres Heinloo
Helmholtz Center Potsdam, German Center for Geosciences (GFZ)

Jan Becker
Gempa GmbH

saul@gfz-potsdam.de, becker@gempa.de

ORFEUS Observatory Meeting 2015, Bucharest
SeisComP meets ObsPy

What is SeisComP?

- Software package for seismological data acquisition, distribution, quality control, archival and analysis in real time
- GUIs for quick manual interaction, event visualization and state-of-health monitoring
- Modular architecture, plugins, QuakeML-like data model
- Distributed processing through messaging
- Support for standards: SEED, SeedLink, QuakeML, FDSN StationXML, FDSN web services
- Framework for seismological software development
- Nowadays 99 percent written in C++ (≈490 000 lines)
- Easily extensible through its Python interface

Evolution of the SeisComP software package

- Originally designed as acquisition and archiving software for the GEOFON data center at GFZ Potsdam (Germany)
- SeedLink as core protocol and software has become a de-facto standard in Europe since 2001 and is adopted world-wide
- Since 2003 (after the Algeria earthquake) development of simple automatic analysis tools
- Since 2005 (version 2) with multi-channel picker, global associator/locator, since 2006 prototype version of mB magnitude no integrated interactive analysis
- ArcLink for distributed archives (waveforms and meta data)
- Mid of 2006 start of SeisComP 3 core (communication, database)
- SeisComP 3 GUI development starting in end of 2006

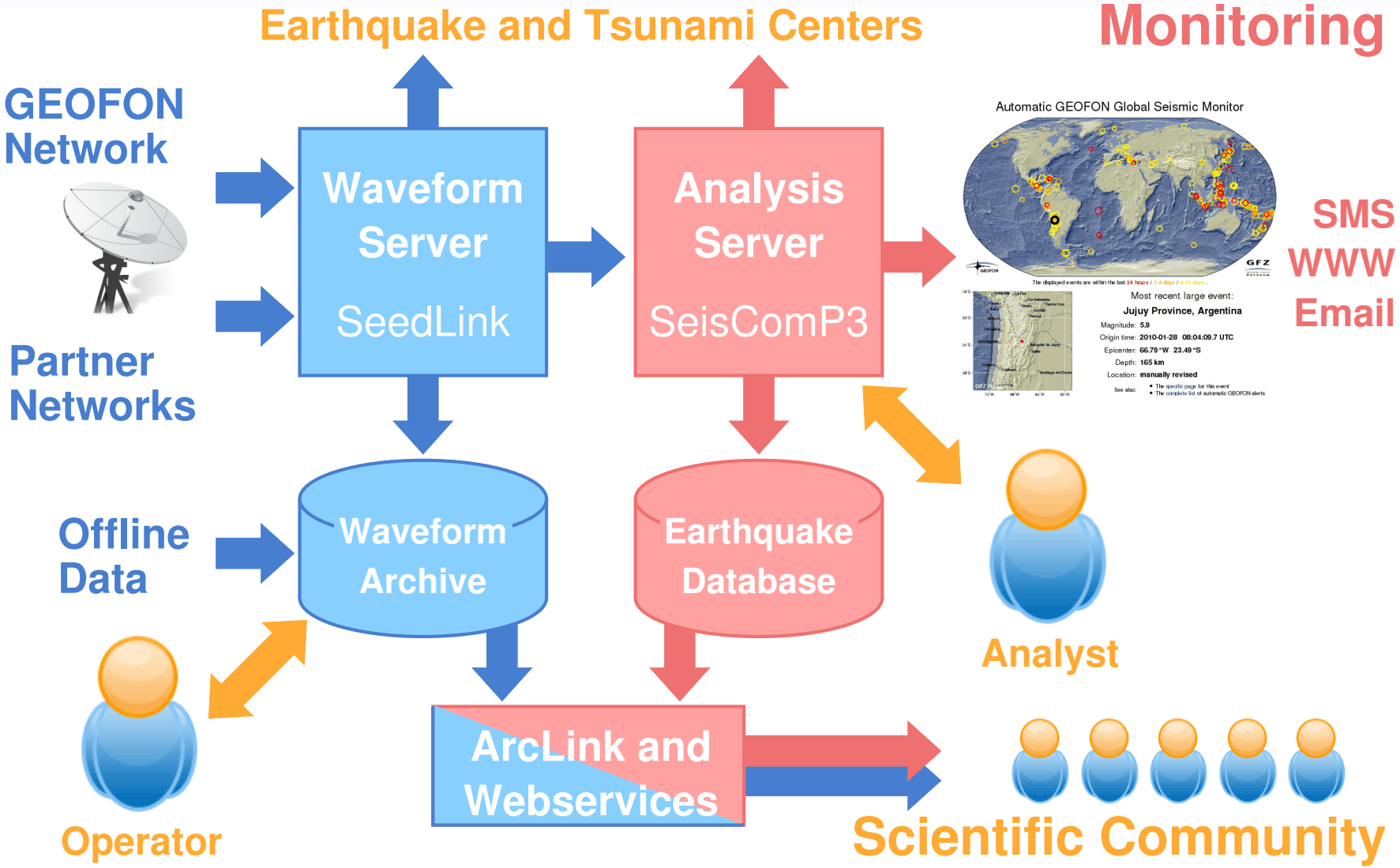
Evolution of the SeisComP software package

- May 2007 deployment of first SeisComP3 prototype in Indonesia
- May 2008 first SeisComP 3 User Group (SC3UG) meeting, software training and release (Barcelona)
- May 2009 further training and public release (Erice) incl. improved database schema
- Sept. 2010 release "Potsdam" at SC3UG 2010
- Sept. 2011 release "Zürich" at SC3UG 2011 incl. add-ons for local monitoring sponsored by ETH Zürich
- Jan. 2013 release "Seattle" at SC3UG 2013 incl. add-ons sponsored by IRIS (scconfig, fdsnws)
- Sept. 2014 release "Jakarta" at SC3UG 2014

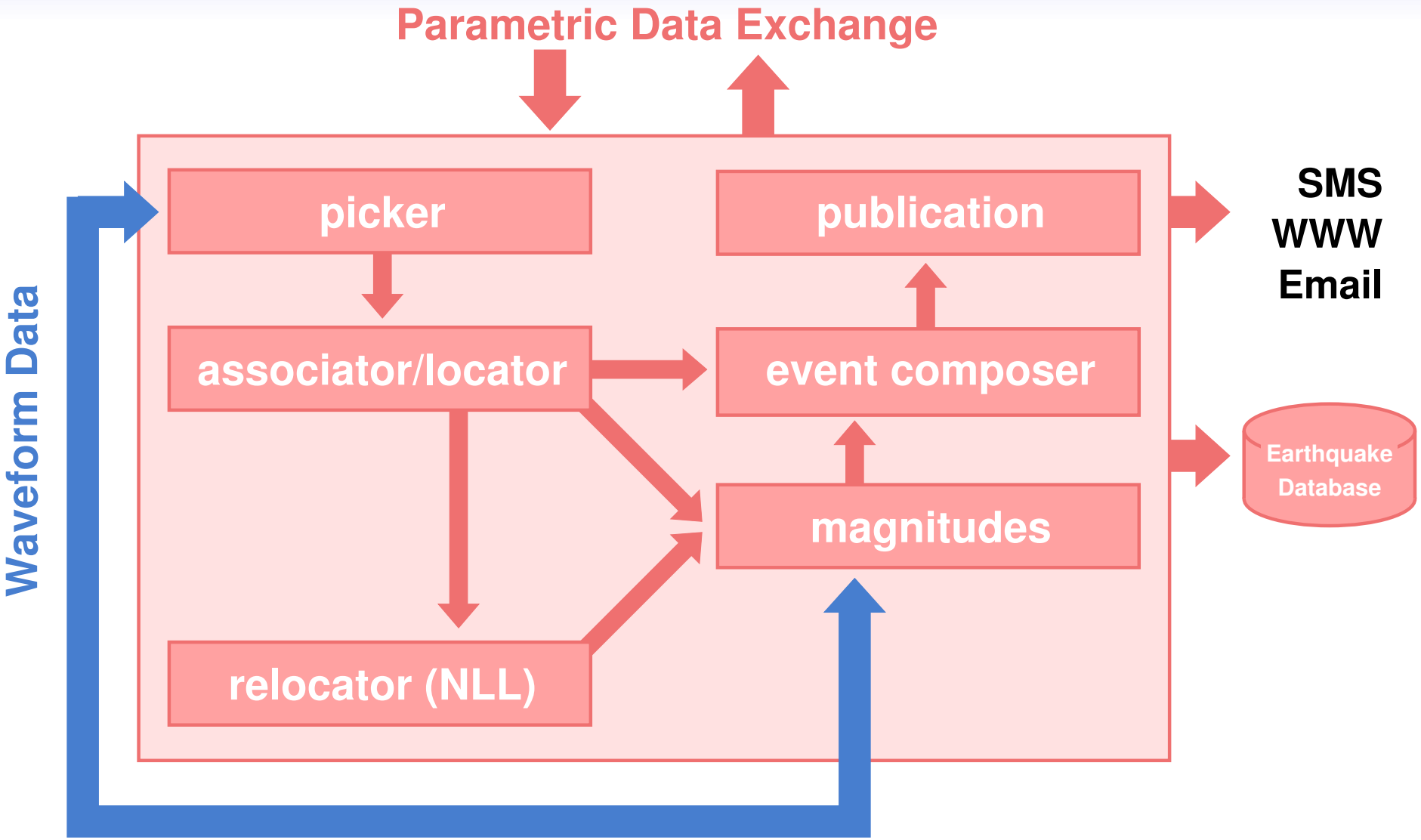
Main features of SeisComP

- Distributed processing within TCP/IP network
- Data acquisition using SeedLink
- QuakeML-derived data model used for storage and communication
- Database support for MySQL, PostgreSQL and SQLite
- Automatic phase picker, global phase associator/locator
- Magnitudes implemented: ML, mb, mB, Mw(mB),Mwp
  recently: mBc
- PGA/PGV/response spectra computation (e.g. for ShakeMaps)
- GUIs for data quality and SOH monitoring and event visualization
- GUIs for near-real-time manual analysis
- Emphasis on simplicity and speed. SeisComP 3 must allow timely tsunami warnings!

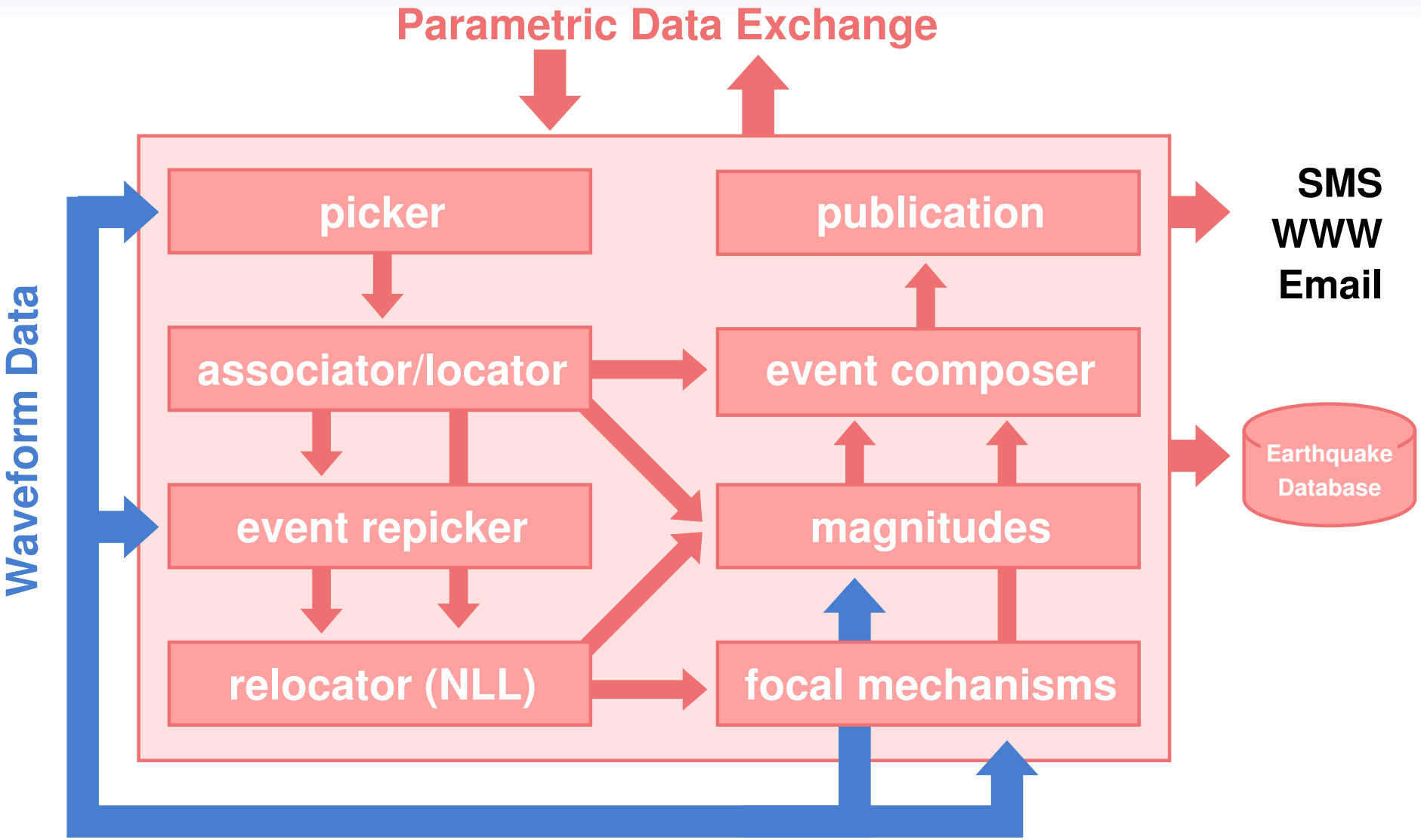
SeisComP within a typical data center



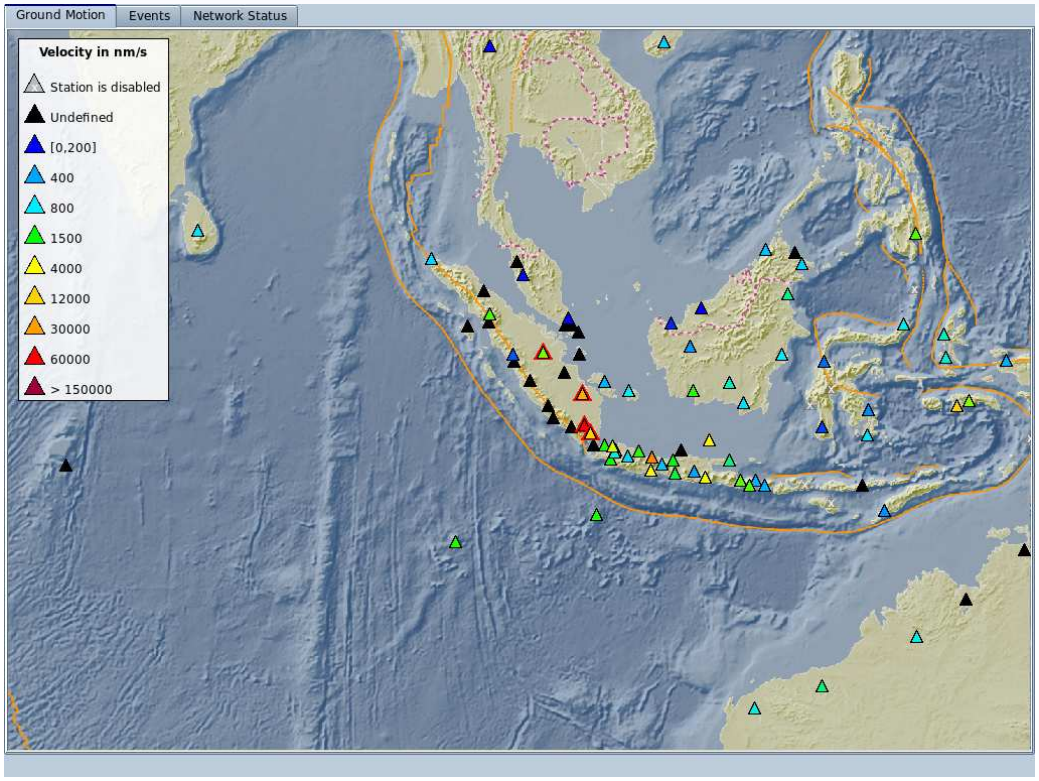
SeisComP processing data flow



SeisComP processing data flow



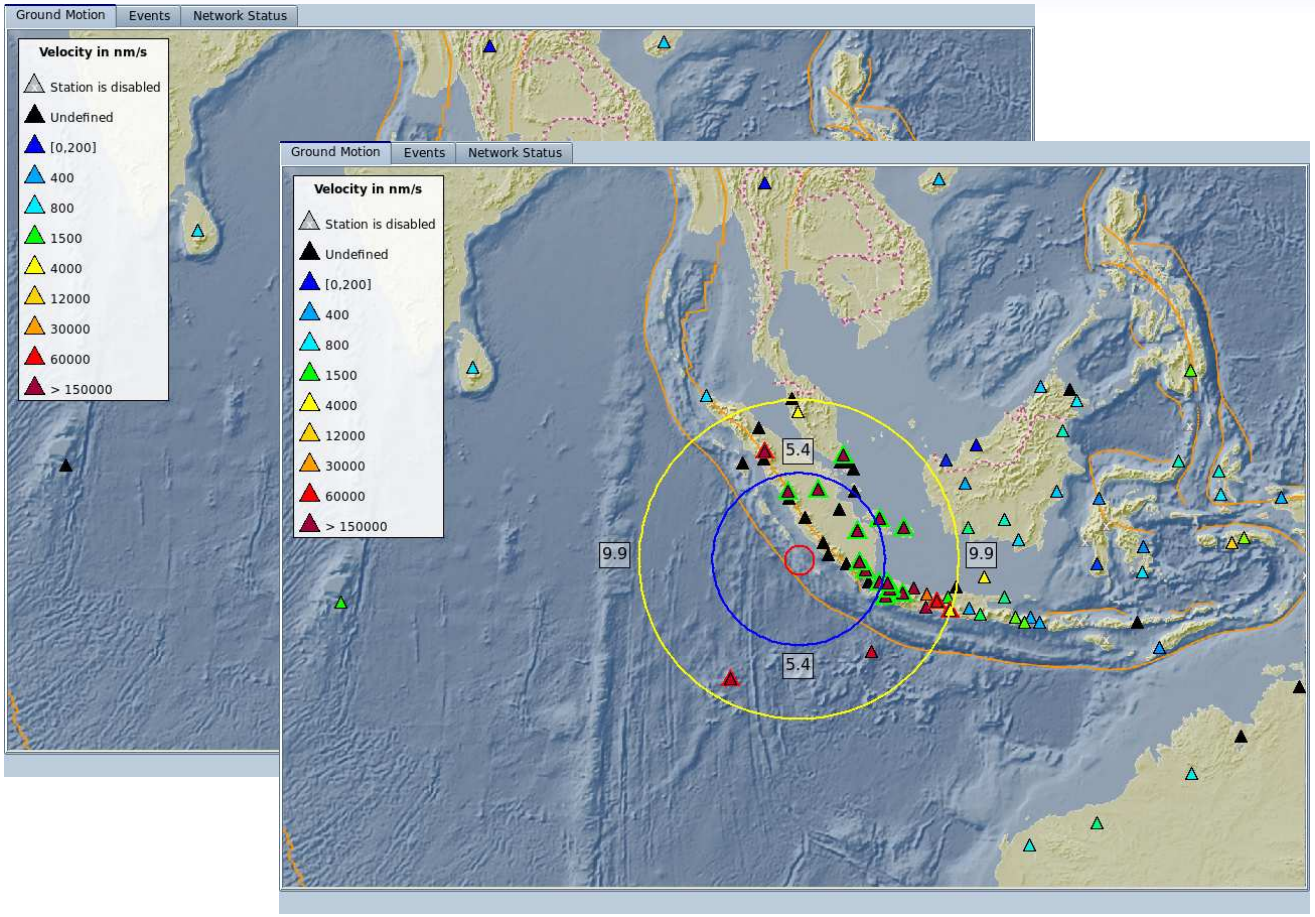
SeisComP3 GUIs - MapView



+1 min.

From playback!

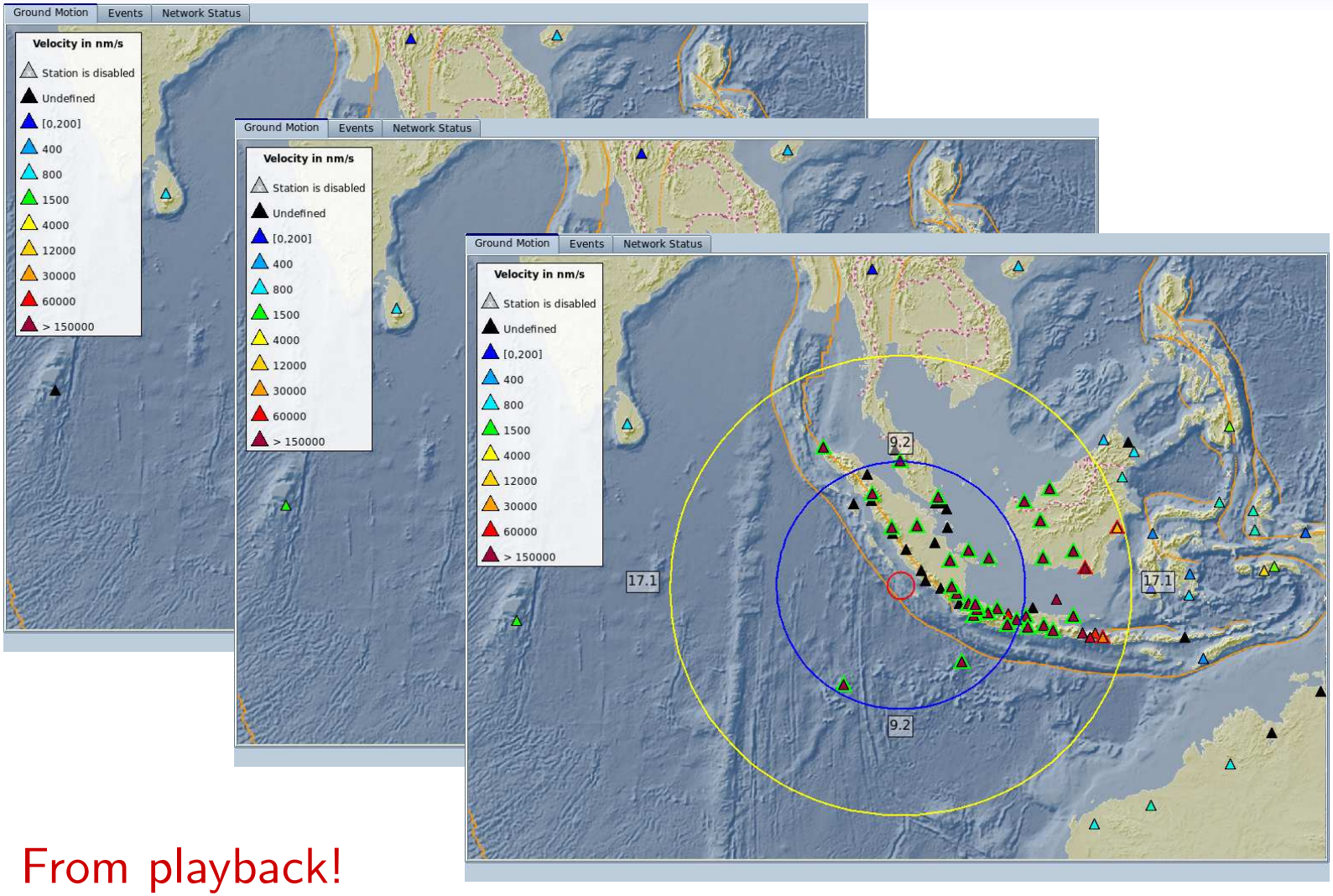
SeisComP3 GUIs - MapView



+2.5 min.

From playback!

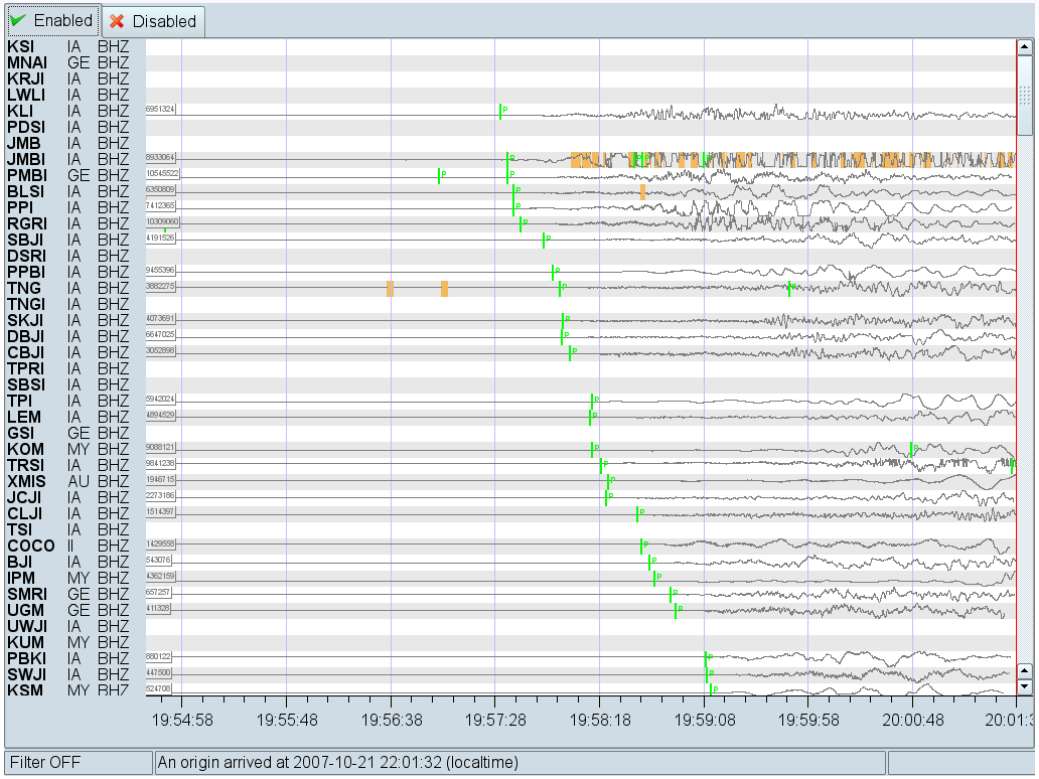
SeisComP3 GUIs - MapView



From playback!

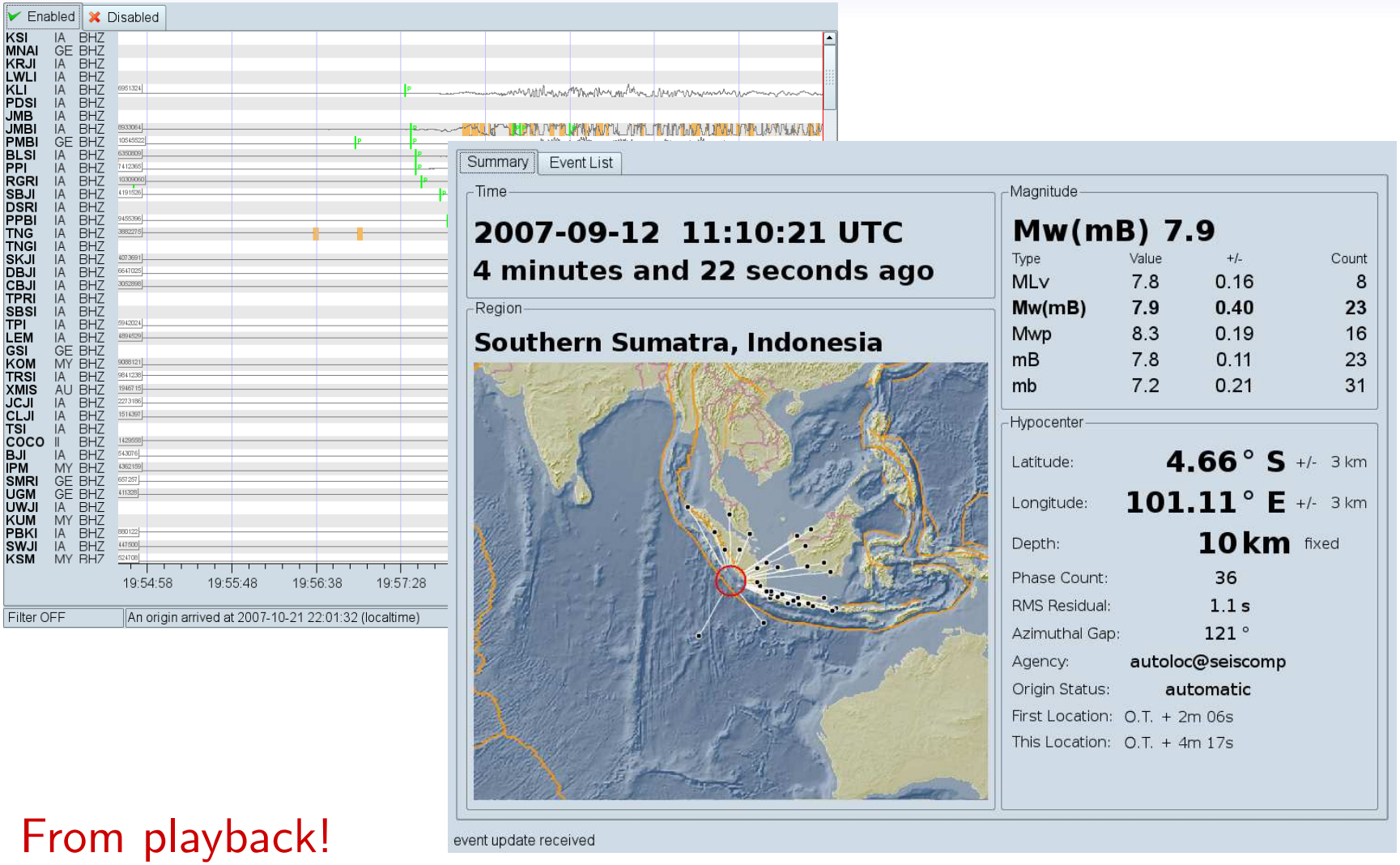
+4 min.

SeisComP3 GUIs



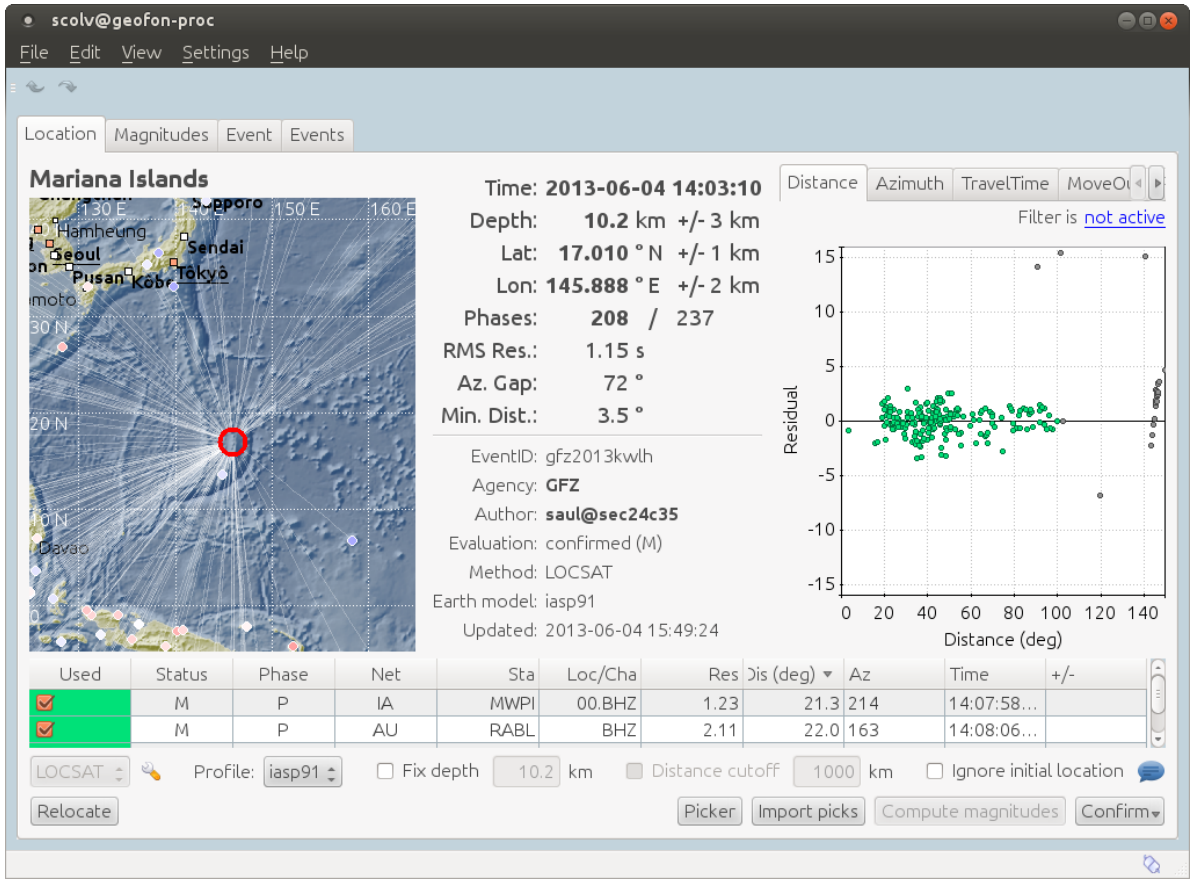
From playback!

SeisComP3 GUIs

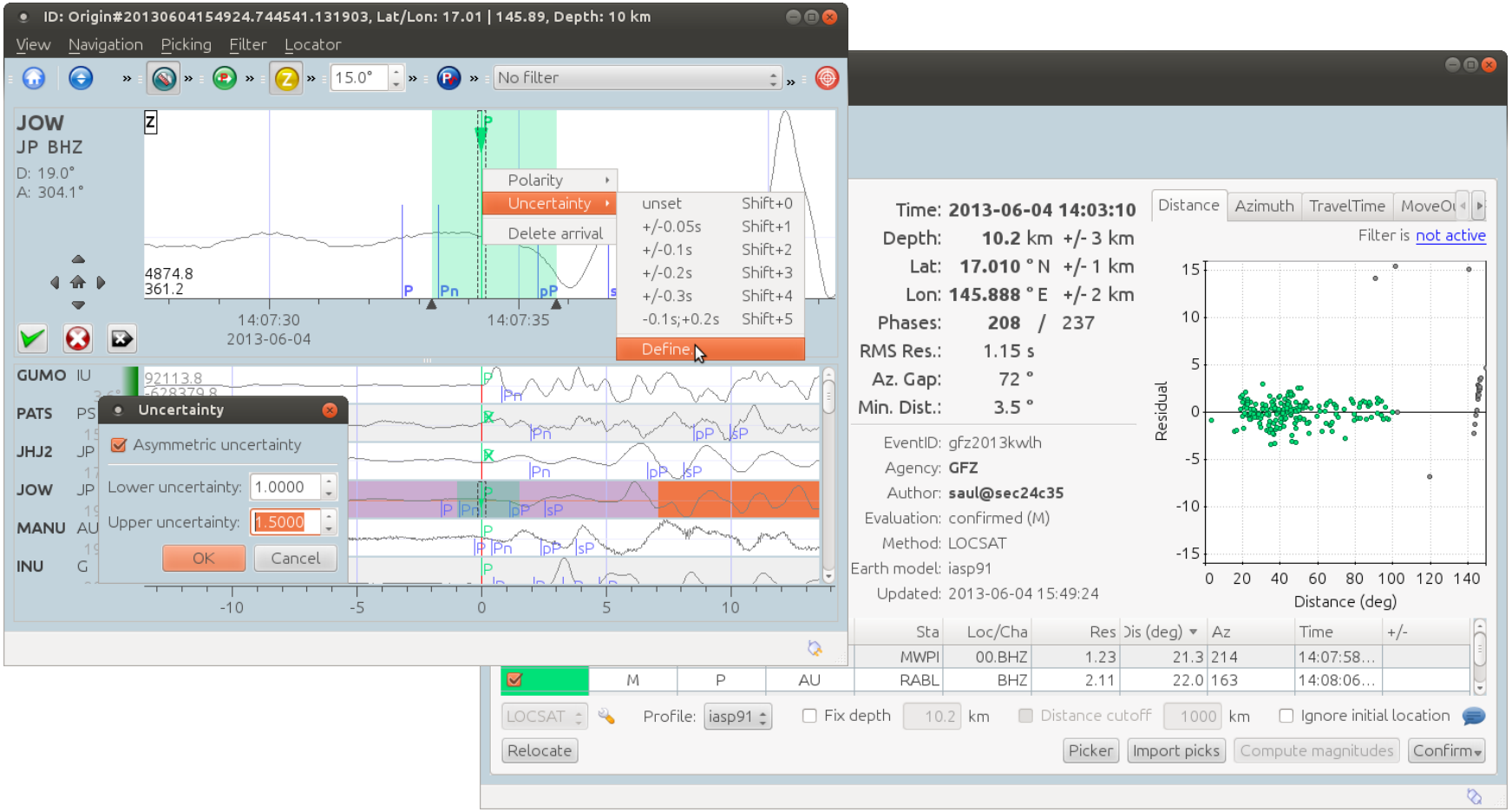


From playback!

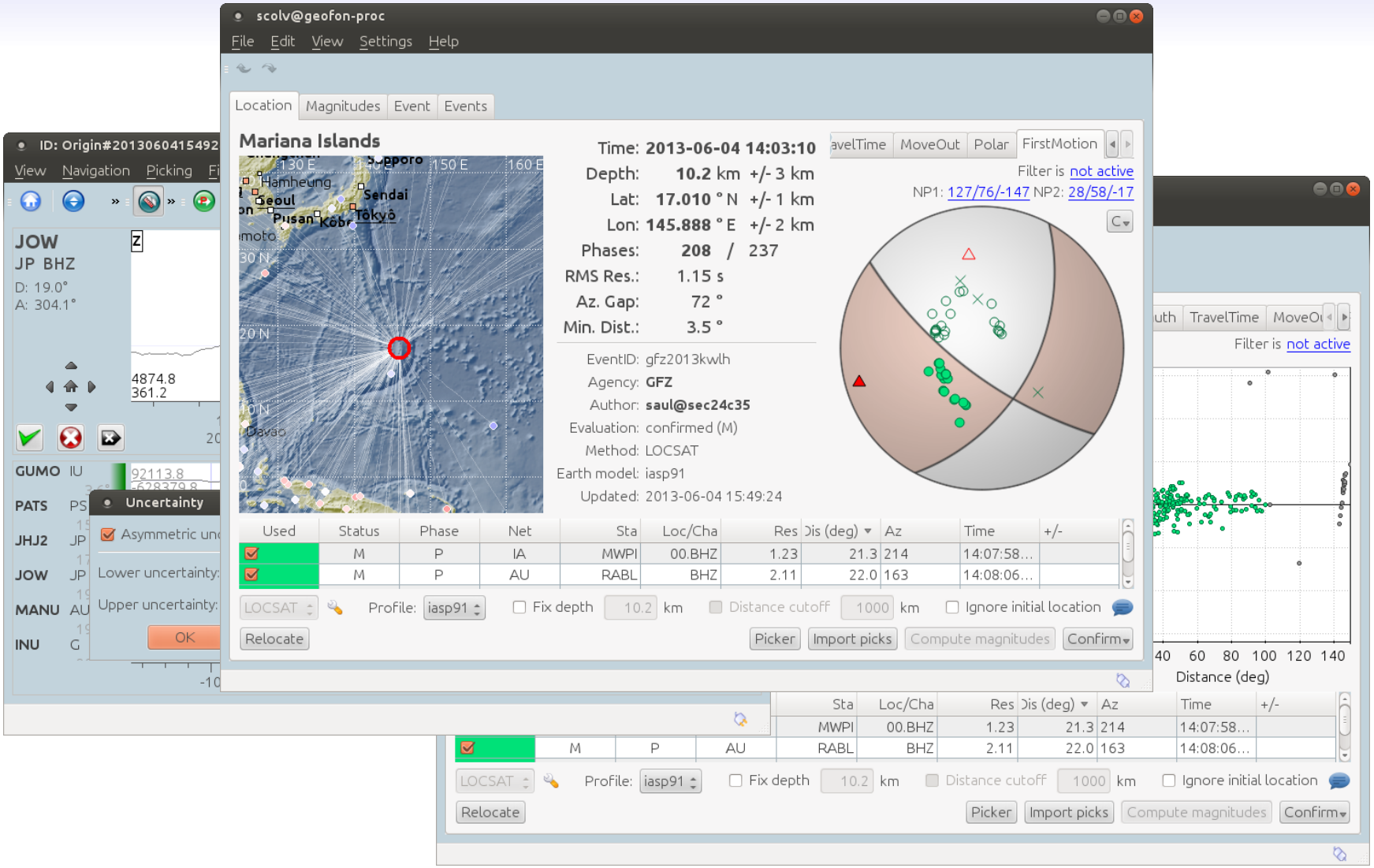
SeisComP3 GUIs - manual picking



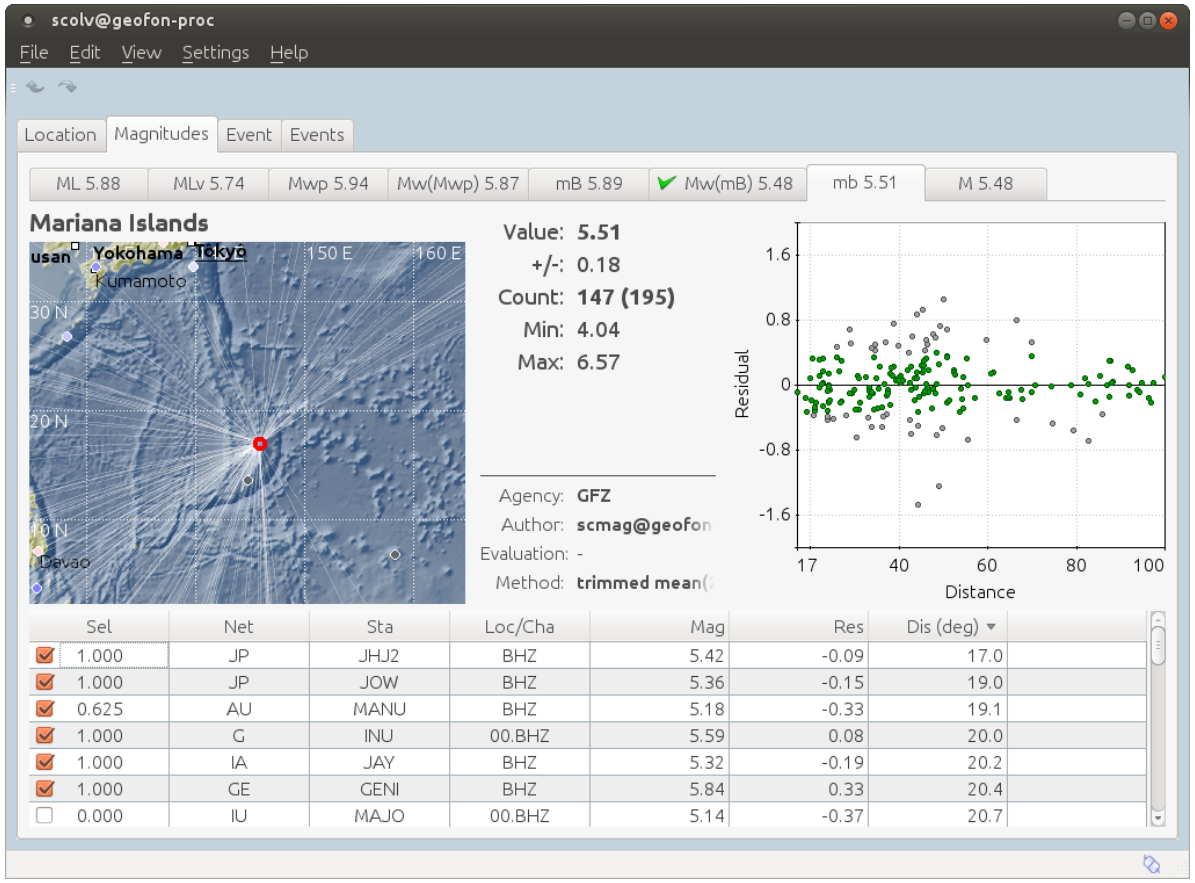
SeisComP3 GUIs - manual picking



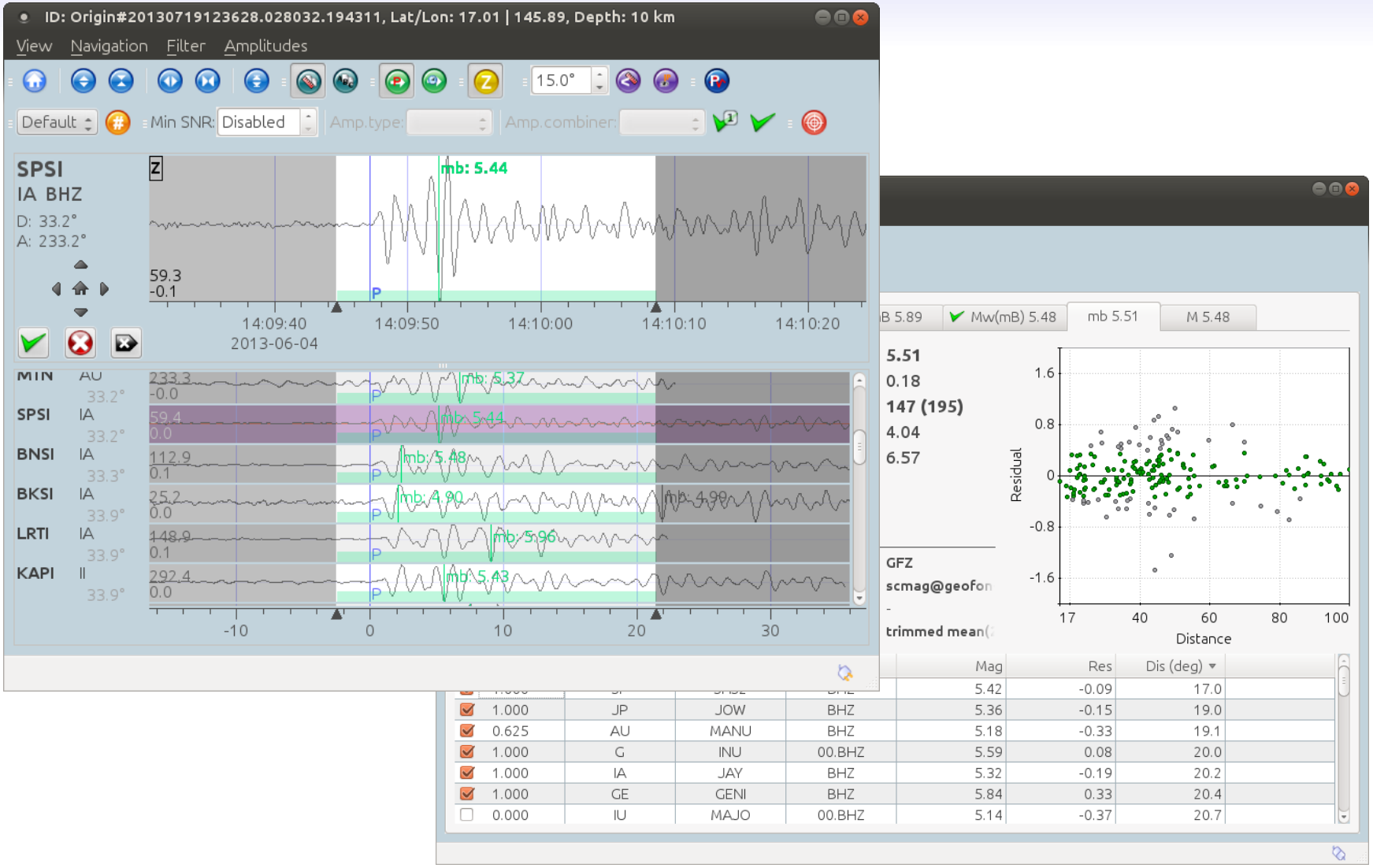
SeisComP3 GUIs - manual picking



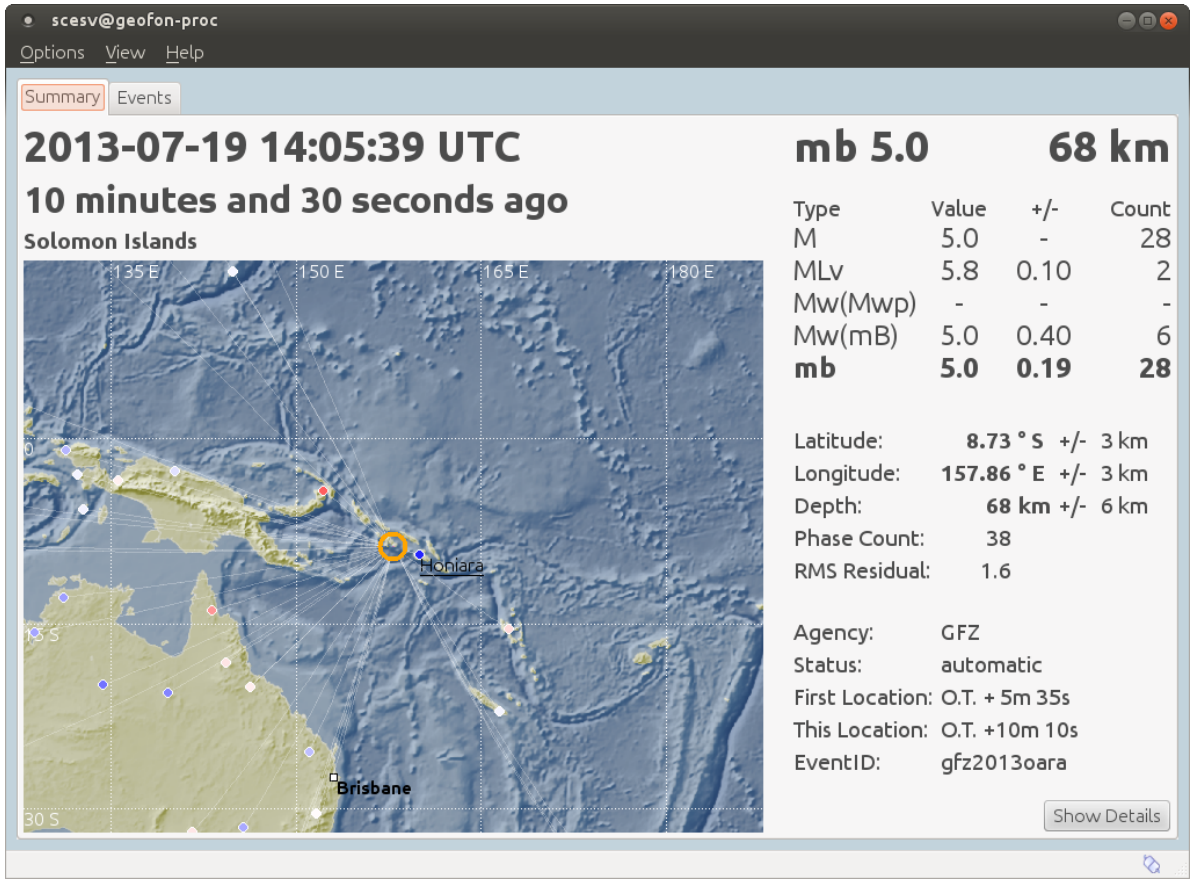
SeisComP3 GUIs - magnitude review



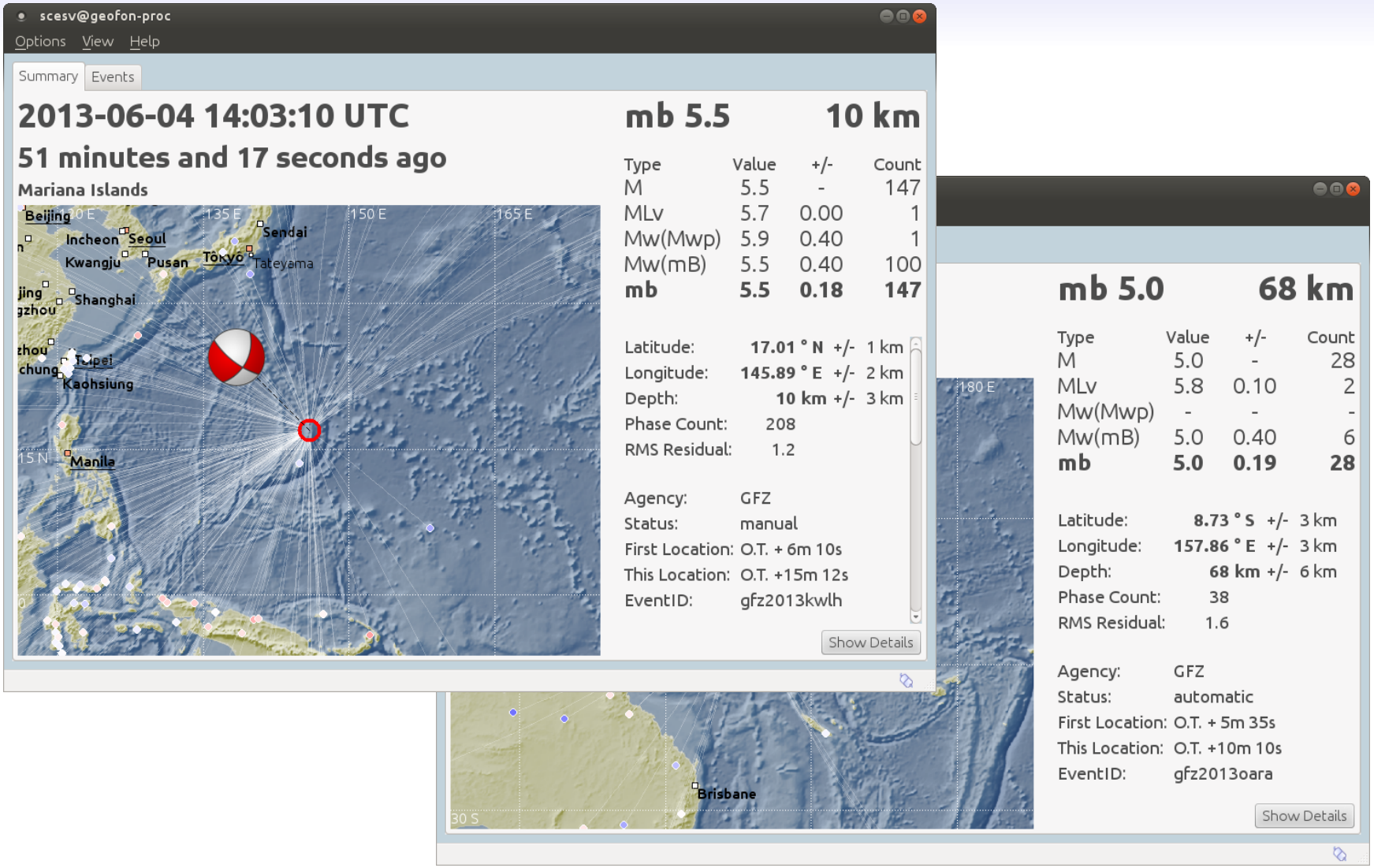
SeisComP3 GUIs - magnitude review



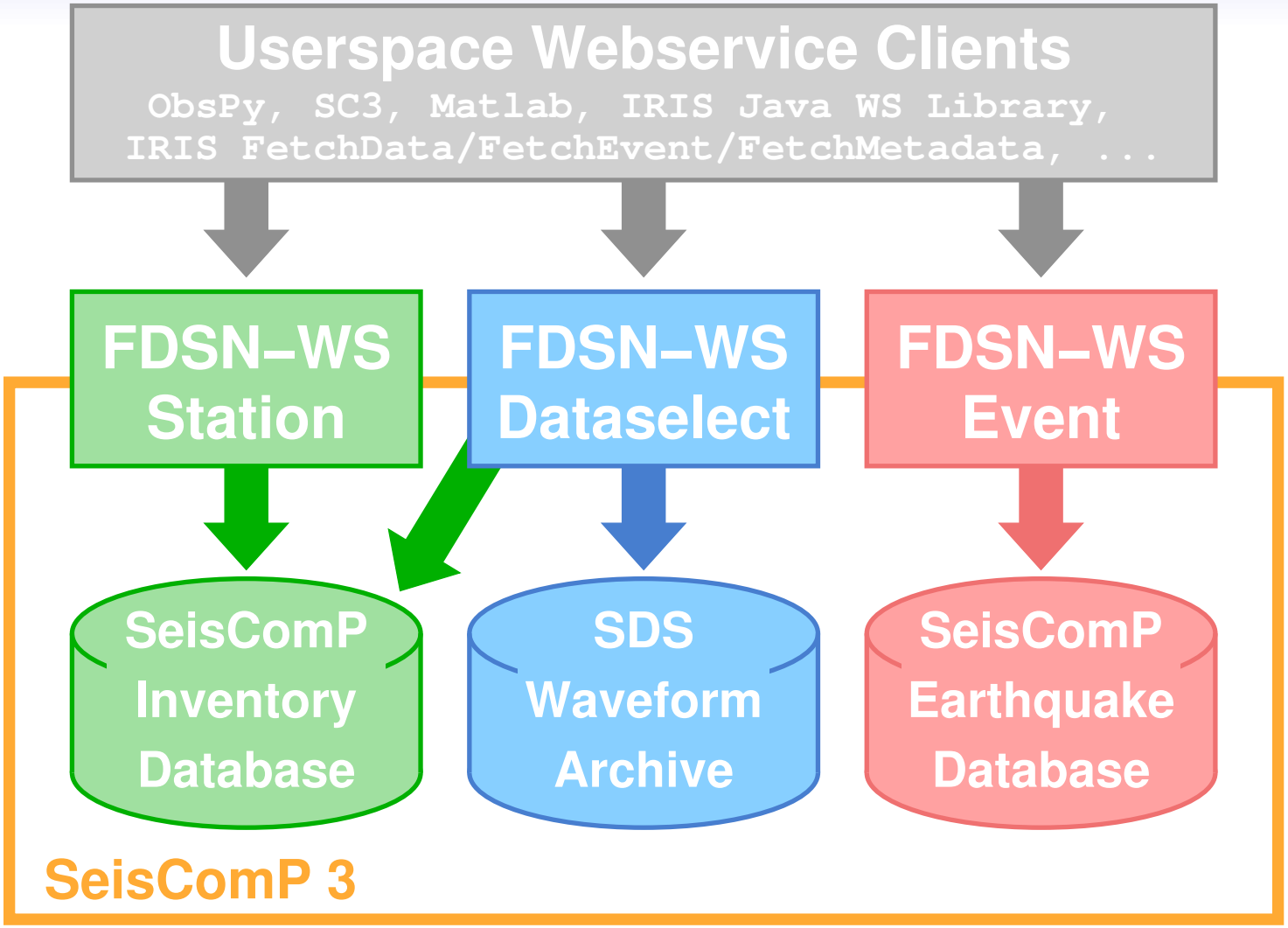
SeisComP3 GUIs - event summary view



SeisComP3 GUIs - event summary view



FDSN Webservices in SeisComP



The SeisComP Data Model

Data Model

A **data model** organizes data elements and standardizes how the data elements relate to one another.

Data Format

A **data format** is a standard way that information is encoded for storage in a computer file. It specifies how bits are used to encode information in a digital storage medium.

Data with the same data model

may be represented in many different **data formats**!

The SeisComP Data Model

Data Model

A **data model** organizes data elements and standardizes how the data elements relate to one another.

Data Format

A **data format** is a standard way that information is encoded for storage in a computer file. It specifies how bits are used to encode information in a digital storage medium.

Data with the same data model

may be represented in many different **data formats**!

The SeisComP Data Model

Data Model

A **data model** organizes data elements and standardizes how the data elements relate to one another.

Data Format

A **data format** is a standard way that information is encoded for storage in a computer file. It specifies how bits are used to encode information in a digital storage medium.

Data with the same data model

may be represented in many different **data formats**!

History of the SeisComP Data Model

Before SeisComP 3

- no formal data model for event information
- only (Mini)SEED as standard **format**
- few modules exchanged information using ad hoc formats
- initial station inventory by A. Heinloo (GFZ) used for ArcLink

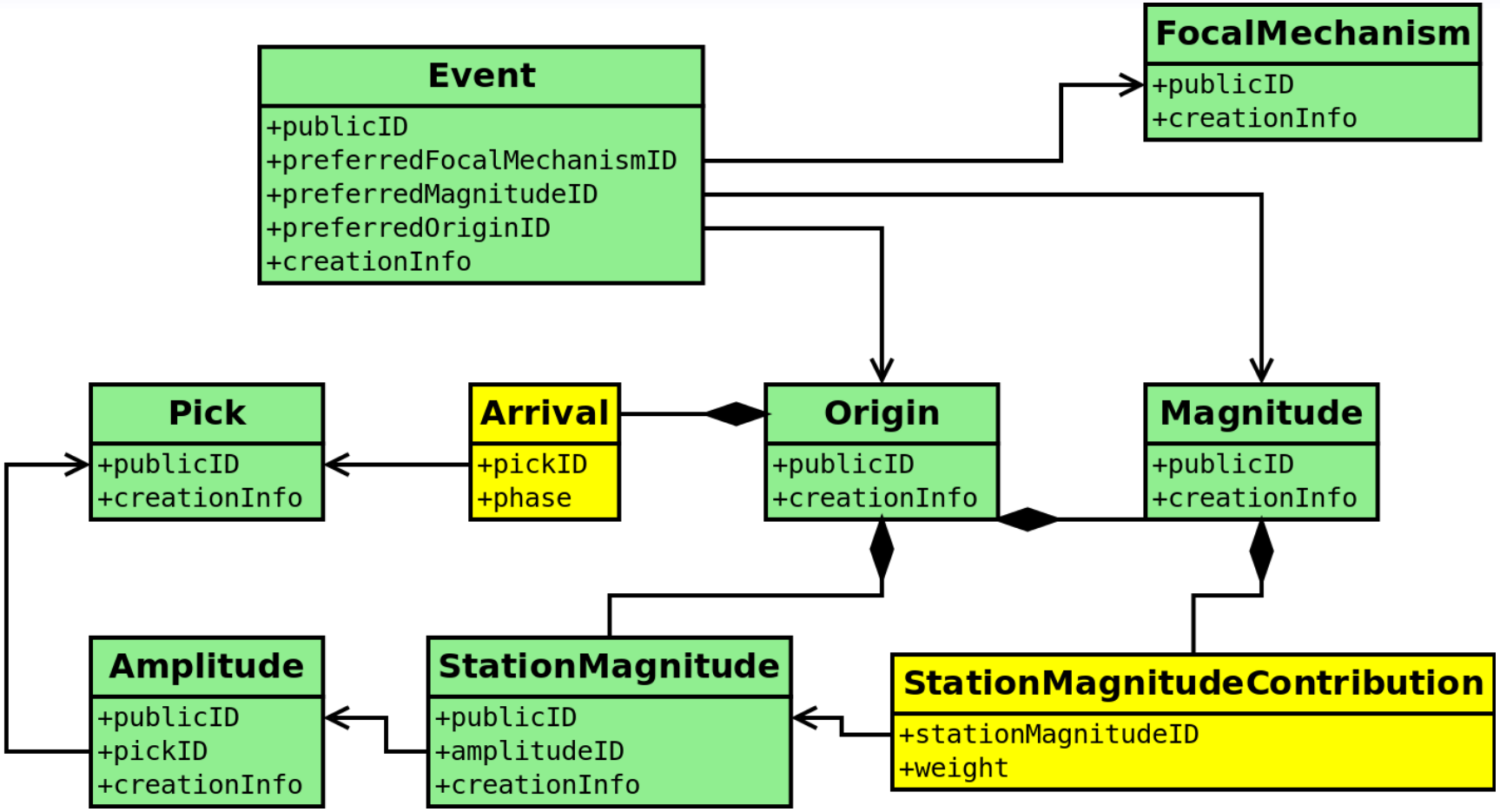
Early SeisComP 3

- QuakeML 1.1 as first formal data model for event related information (colaboration with ETHZ and others)
- improvement of station inventory by A. Heinloo and J. Becker

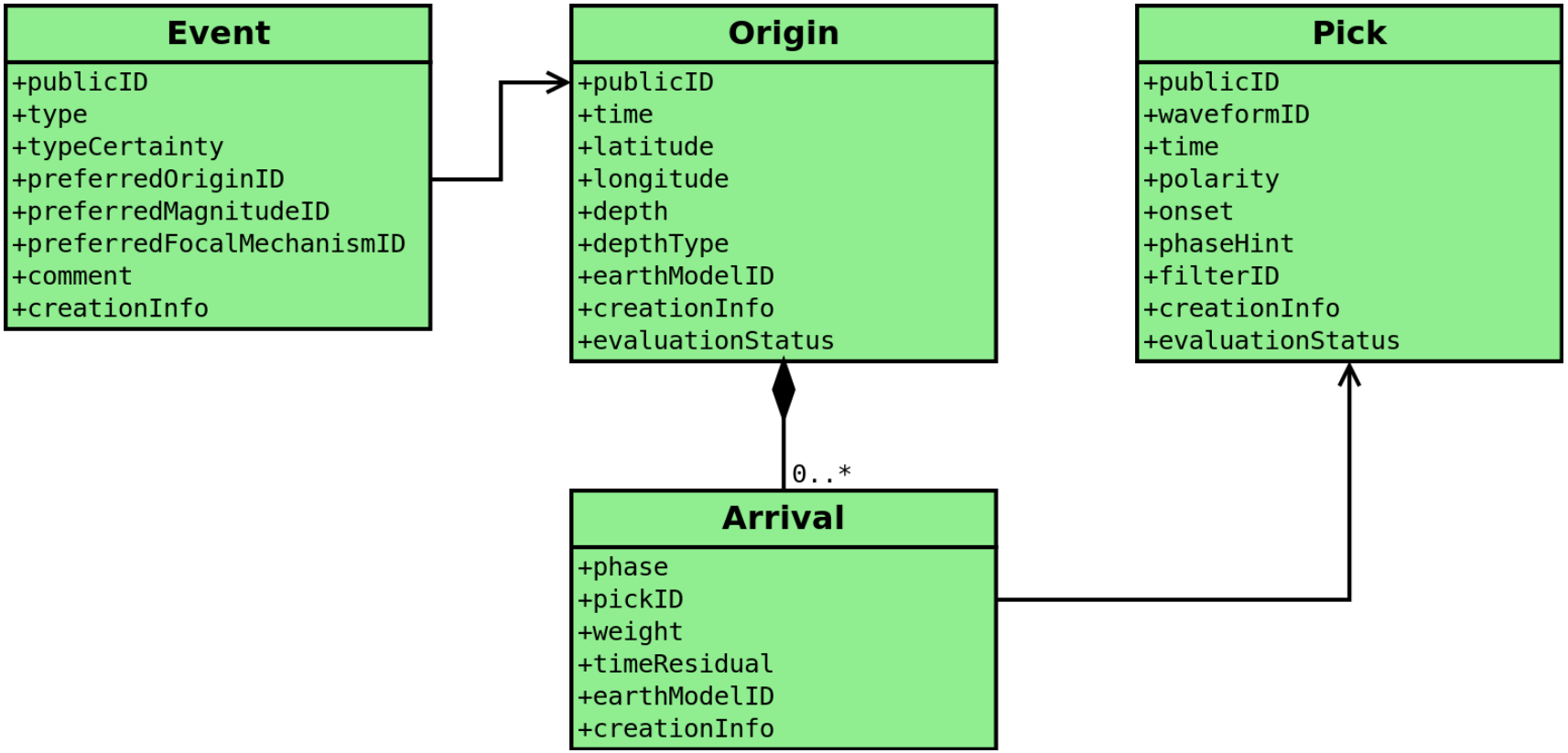
More recently in SeisComP 3

- deviations from QuakeML mostly due to QuakeML evolution
- new, non-QuakeML modules, e.g. focal mechanisms

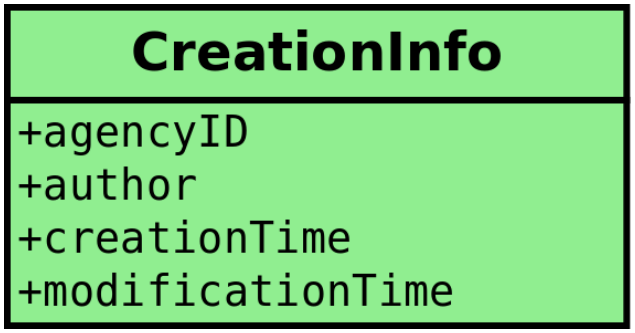
SeisComP EventParameters

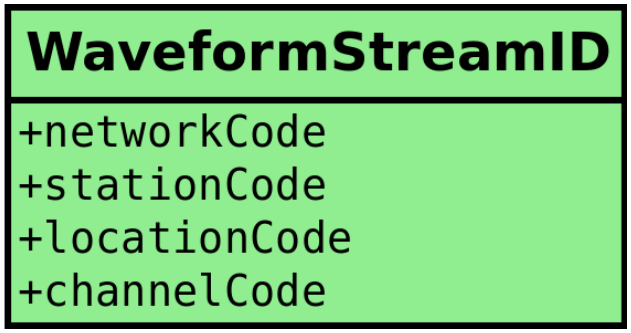


Detailed Example



CreationInfo and WaveformStreamID





SeisComP EventParameters vs. QuakeML

- common root: QuakeML 1.1
- SeisComP schema could not be kept in sync with QuakeML
- main differences
  - relationship between Origin and Magnitude
  - in QuakeML Magnitude is a top-level element whereas in SeisComP is a child of Origin
  - much more comprehensive MomentTensor / Focal Mechanism in SeisComP
- most data classes still identical between SeisComP and QuakeML
- $\Rightarrow$ QuakeML documentation almost entirely usable as reference for SeisComP event parameters
- more info on QuakeML: `http://quakeml.org`

The SeisComP Python interface

Why Python?

- very clean, simple syntax
- easy to learn and "seismologist friendly"
- lots of modules for numerics, geosciences (e.g. NumPy, MatPlotLib, Basemap) with strong communities behind
- specialized seismology modules like ObsPy, SeisComP Python

SeisComP 2

Large parts of the SeisComP processing written in Python in a very ad hoc way:

- Python code quick+dirty but not well organized
- no data model, no database support

since SeisComP 3

- core of SeisComP rewritten from scratch in C++
- thin, automatically generated Python wrapper using Swig

The SeisComP Python API

Access to object attributes via accessor functions

```
t = pick.time().value()
s = pick.waveformID().stationCode()

q = TimeQuantity(t)
pick.setTime(q)
pick.waveformID().setStationCode(s)
```

The SeisComP Python API is not very "pythonic"

because it essentially exposes a C++ API in Python

Exception handling

According to the SeisComP (and QuakeML!) data model, not all attributes of an object are required to be present, so we need to check:

```
try:
    upper = pick.time().upperUncertainty()
except AttributeError:
    upper = None
```

C++ exceptions are nicely mapped to Python exceptions

via the Swig wrappers

Possibility for a more “pythonic” API

```
class Pick(seiscomp3.DataModel.Pick):
    @property
    def slow(self):
        try:
            return self.horizontalSlowness()
        except AttributeError:
            return None

    @slow.setter
    def slow(self, value):
        self.setHorizontalSlowness(value)

# and with this we can write:
pick.slow = p
p = pick.slow
```

The SeisComP Application class

The Application class is a base class which is

responsible for the access and management of

- command line parameters
- messaging connection
- data base connection
- waveform access

overloaded in a user program

to add program specific functionality while also keeping the functionality of the base class

available in two flavors

- Application (light weight)
- StreamApplication (also loads backends for waveform access)

Application class skeleton

```
import sys, seiscomp3.Client, seiscomp3.DataModel

class App(seiscomp3.Client.Application):
    def __init__(self, argc, argv):
        seiscomp3.Client.Application.__init__(self,argc,argv)
        self.setMessagingEnabled(False)
        self.setDatabaseEnabled(True, True)
        self.setLoggingToStdErr(True)
        self.setLoadInventoryEnabled(True)

    def run(self):
        ... # this is where the main action takes place
        return True

app = App(len(sys.argv), sys.argv)
app()
```

Application class run() function

```
# see https://github.com/jsaul/sc3stuff
import sc3stuff.inventory

class App(seiscomp3.Client.Application):
    ...

    def run(self):
        # load current inventory
        now = seiscomp3.Core.Time.GMT()
        inv = seiscomp3.Client.Inventory.Instance().inventory()
        itr = sc3stuff.inventory.InventoryIterator(inv, now)
        for (net, sta, loc, stream) in itr:
            print net.code(), sta.code(), \
              sta.latitude(), sta.longitude(), sta.elevation()
        return True
    ...
```

Application class configuration

How did the application know which database to load the inventory from? This is also handled transparently by the application class!

via config file (without messaging)

```
database.type = mysql
database.parameters = sysop:sysop@geofon-proc/seiscomp3
```

via config file (with messaging)

```
connection.server = geofon-proc
```

via command line interface

```
python myapp.py \
    -d mysql://sysop:sysop@geofon-proc/seiscomp3
```

only inventory, no database connection:

```
python myapp.py --inventory-db inventory.xml
```

Minimal Messaging Example – Python script

```
import sys, seiscomp3.Client, seiscomp3.DataModel

class PickClient(seiscomp3.Client.Application):
    def __init__(self, argc, argv):
        seiscomp3.Client.Application.__init__(self,argc,argv)
        self.setMessagingEnabled(True)
        self.setLoggingToStdErr(True)
        self.addMessagingSubscription("PICK")

    def addObject(self, parentID, obj):
        pick = seiscomp3.DataModel.Pick.Cast(obj)
        if pick:
            print "new pick", pick.publicID()

app = PickClient(len(sys.argv), sys.argv)
app()
```

Minimal Messaging Example – invocation

```
$ seiscomp exec python minimal.py -H geofon-proc -v
[...]
16:31:55 [info] Connected to message server: geofon-proc
16:31:55 [info] Setting message encoding to binary
16:31:55 [info] Joining group: PICK
16:31:55 [info] Connect to database
16:31:55 [info] Received database service parameters
16:31:55 [info] Trying to connect to
mysql://sysop:sysop@geofon-proc1.gfz-potsdam.de/seiscomp3
16:31:56 [info] Connected successfully
16:31:56 [info] Starting message thread
new pick 20150921.143120.76-AIC-CX.PB08..BHZ
new pick 20150921.143140.59-AIC-IA.JMBI.00.BHZ
new pick 20150921.143120.36-AIC-IA.PWJI..BHZ
new pick 20150921.143124.68-AIC-IA.GMJI..BHZ
[...]
```

Minimal Streaming Example – Python script

```
class StreamingApp(seiscomp3.Client.StreamApplication):
    def __init__(self, argc, argv):
        seiscomp3.Client.StreamApplication.__init__(self, argc, argv)
        self.setMessagingEnabled(False)
        self.setDatabaseEnabled(False, False)
        self.setLoggingToStdErr(True)

    def init(self):
        if not seiscomp3.Client.StreamApplication.init(self):
            return False
        now = seiscomp3.Core.Time.GMT()
        t1,t2 = now + seiscomp3.Core.TimeSpan(-1800), now
        stream = self.recordStream()
        stream.addStream("GE", "KARP", "",   "BHZ")
        stream.addStream("GE", "GHAJ", "",   "BHZ")
        stream.addStream("IU", "ANTO", "10", "BHZ")
        return True

    def handleRecord(self, record):
        print record.stationCode(), record.startTime()
        return True

app = StreamingApp(len(sys.argv), sys.argv)
app()
```

Minimal Streaming Example – invocation

```
i="slink://geofon.gfz-potsdam.de:18000"
seiscomp exec python streaming-client.py --debug -I "$i"
```

```
[...]
13:42:34 [debug] trying to open stream slink://geofon.gfz-potsdam.de:18000
13:42:34 [debug] Set timeout=300
13:42:34 [info] Starting record acquisition
13:42:34 [debug] geofon.gfz-potsdam.de:18000 connected
13:42:34 [debug] Handshaking SeedLink server at geofon.gfz-potsdam.de:18000
13:42:34 [debug] Seedlink command: STATION GHAJ GE
13:42:34 [debug] Seedlink command: SELECT BHZ.D
13:42:34 [debug] Seedlink command: DATA
[...]
13:42:34 [debug] handshake done in 0.054422 seconds
GHAJ 2015-09-22 11:32:22.419
KARP 2015-09-22 11:32:25.645
GHAJ 2015-09-22 11:32:35.769
KARP 2015-09-22 11:32:46.595
GHAJ 2015-09-22 11:32:47.919
KARP 2015-09-22 11:33:07.145
[...]
```

Streaming Data Source Specification

```
seiscomp exec python streaming-client.py -I "..."
```

SeedLink

```
slink://geofon.gfz-potsdam.de:18000
```

ArcLink

```
arclink://geofon.gfz-potsdam.de:18001
```

FDSN web services

```
fdsnws://geofon.gfz-potsdam.de/fdsnws/dataselect/1/query
fdsnws://service.iris.edu/fdsnws/dataselect/1/query
```

MiniSEED file

Here: a file containing multiplexed streams for an earthquake

```
file://$HOME/gfz2015iatp.mseed
```

Access to raw MiniSEED records

Purpose

- we often need to keep the raw, binary MiniSEED records
- e.g. if we only want to sort or (de)multiplex streams
- or pass the record to 3rd-party software, e.g. ObsPy
- $\Rightarrow$ we need access to the raw (binary) record

configure the Application class

```
class WaveformApp(seiscomp3.Client.StreamApplication):
    def init(self):
        [...]
        hint = seiscomp3.Core.Record.SAVE_RAW
        self.setRecordInputHint(hint)
        stream = self.recordStream()
        [...]
    def handleRecord(self, record):
        raw = record.raw().str()
```

NumPy Interface to Waveforms

NumPy

- extension to Python
- adds support for large, multi-dimensional arrays and matrices
- come with a large library of high-level mathematical functions
- forms the basis for higher-level SciPy and MatPlotLib

in SeisComP Python

```
from numpy import average, sqrt

[...]
data = record.data().numpy()
rms = sqrt(average( (data-average(data))**2 ))
```

Online Documentation

SeisComP 3 resources and downloads

http://www.seiscomp3.org

SeisComP 3 Python API documentation

http://www.seiscomp3.org/doc/jakarta/current/base/api-python.html

UML schema

http://geofon.gfz-potsdam.de/_uml/

QuakeML

http://quakeml.org

SeisComP 3 related scripts

code snippets etc. to get you started more quickly

https://github.com/jsaul/sc3stuff

⇒ waveforms / stream-client

⇒ messaging / pick-client