



Part 1: Introduction to SeisComP



SeisComP and SeedLink

- ◆ The **Seismological Communication Processor** (SeisComP) is a concept for a networked seismographic system, originally developed for the GEOFON network and further extended within the MEREDIAN project under the lead of GEOFON/GFZ Potsdam and ORFEUS. SeisComP software is free and is distributed in the form of a core package, Python extensions package and several add-on packages.
- ◆ **SeedLink** is a data acquisition system, which lies at the core of SeisComP. SeedLink clients connect to the server using a TCP/IP application level protocol (SeedLink protocol).

Plugins

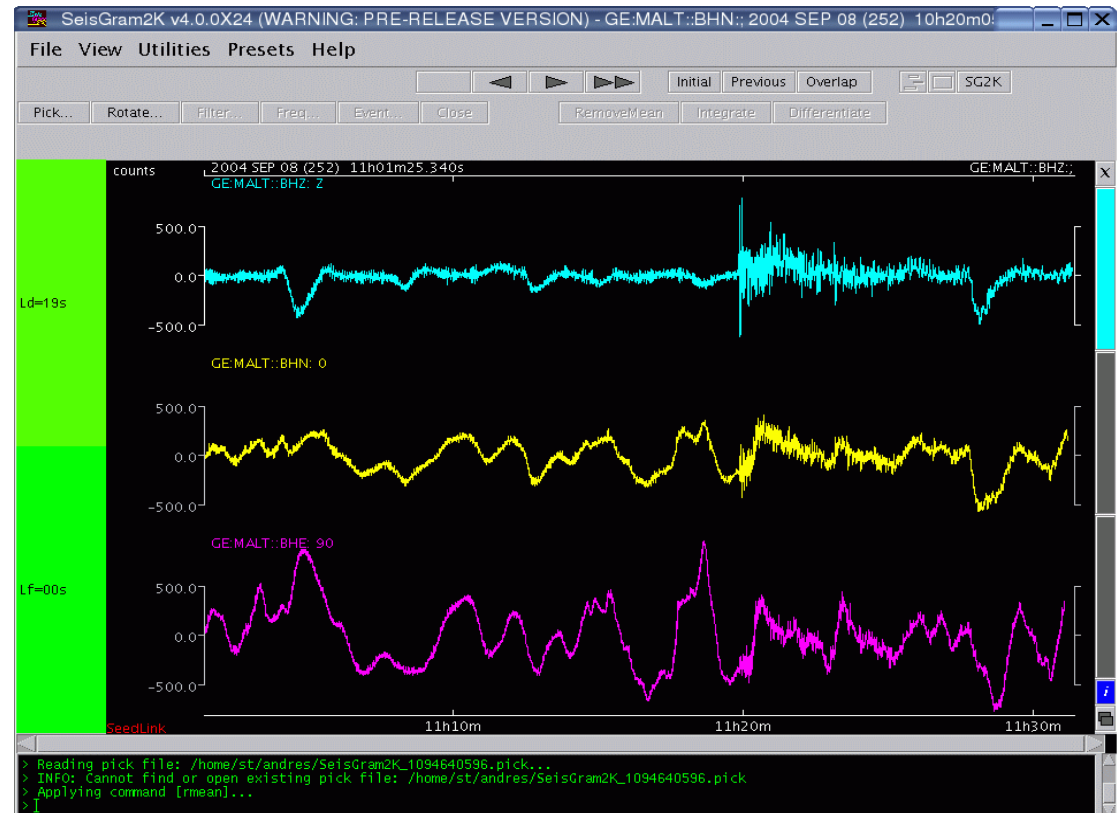
- ◆ The data source of a SeedLink server can be anything which is supported by a SeedLink **plugin**—a small program that sends data to the SeedLink server. The plugin API is defined in C language, but wrappers exist for C++, Java and Python. Data supplied by a plugin can be in a form of Mini-SEED packets or just raw integer samples with accompanying timing information.
- ◆ Plugins for following digitizers have been implemented by GFZ, ORFEUS and others:
 - Quanterra Q380/Q680, Q4120 and Q730 (via Comserv)
 - Quanterra Q330 (UDP/IP)
 - EarthData PS2400 and PS6-24
 - Lennartz M24, PCM 5800 and MARS-88
 - Güralp DM24
 - Kinematics K2
 - Geotech DR-24
 - Nanometrics HRD-24

Clients

- ◆ SeisComP core package includes the following standard clients:
 - **slarchive** saves data to the disk in Mini-SEED format, using SDS (SeisComP Data Structure), BUD (Buffer of Uniform Data) or a user-defined directory structure.
 - **slqplot** is used to plot the traces in real time, either in X-Window or into a file.
 - **slinktool** is used mainly for testing the server and to get info about the available stations, time spans of streams, gaps, etc.
- ◆ Using the **libslink** software library or its Java counterpart **Jlibslink** (created by Anthony Lomax), C and Java programmers can create custom SeedLink clients for their own real-time data processing applications without having to know the details of SeedLink protocol. **libslink** supports Linux/UNIX, Windows and MacOS X platforms.

SeisGram2K

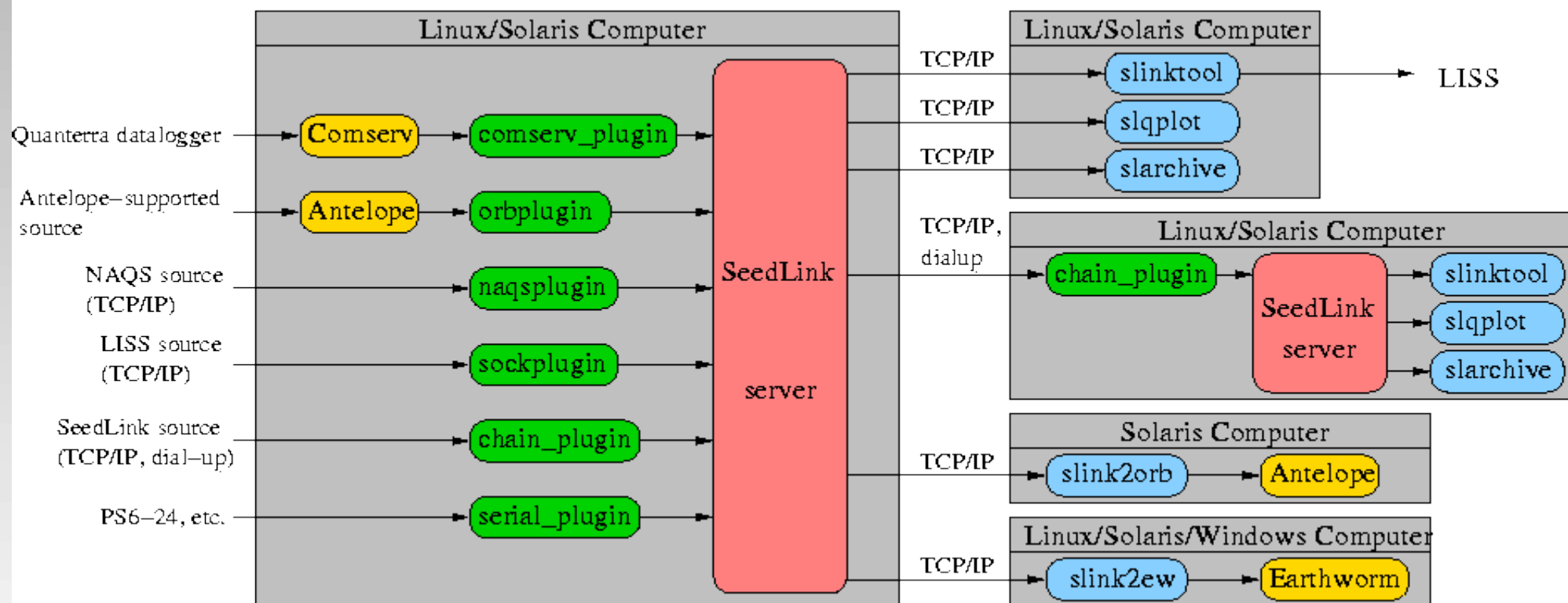
- ◆ **SeisGram2K** (SG2K), a Java-based real-time trace viewer with basic analysis capabilities by Anthony Lomax, can connect to one or more SeedLink servers using **Jlibslink** and display live recording from any number of stations.
- ◆ SG2K runs on any platform supported by Java, including Linux, Windows and Mac OS X.



Data import and export

- ◆ In addition to plugins that talk directly to a digitizer, plugins for exporting data from the following data acquisition systems are available:
 - IRIS/GSN Live Internet Seismic Server (LISS)
 - IRIS/IDA Near Real-Time System (NRTS)
 - Earthworm
 - Kinematics' Antelope
 - Nanometrics' NAQS
 - Gralp's SCREAM
 - RefTek's RTPD
- ◆ Earthworm, Antelope and NAQS can also work as SeedLink clients, importing data from SeedLink.

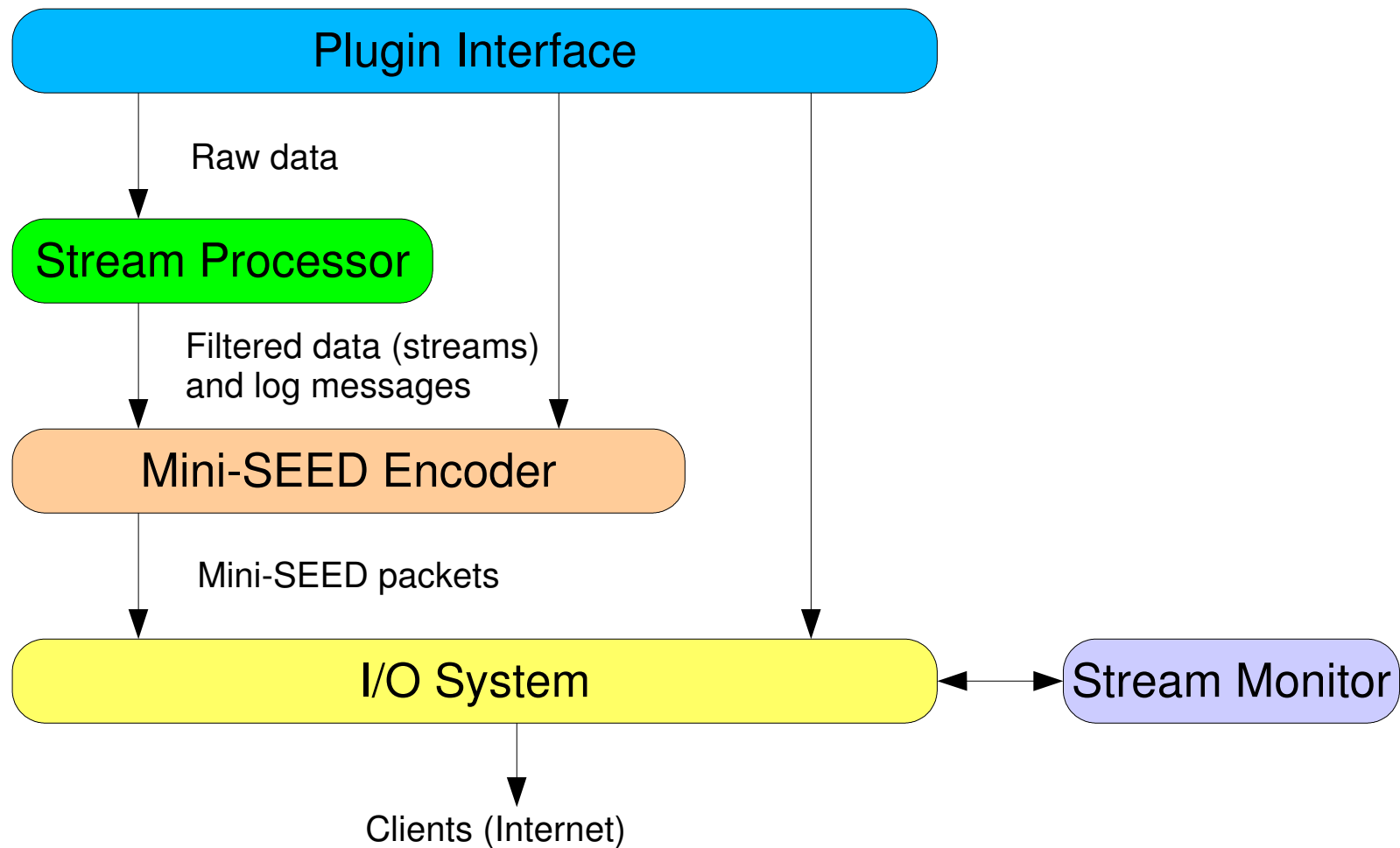
Overview of SeedLink connectivity





Part 2: Advanced SeedLink usage

Data flow in SeedLink server





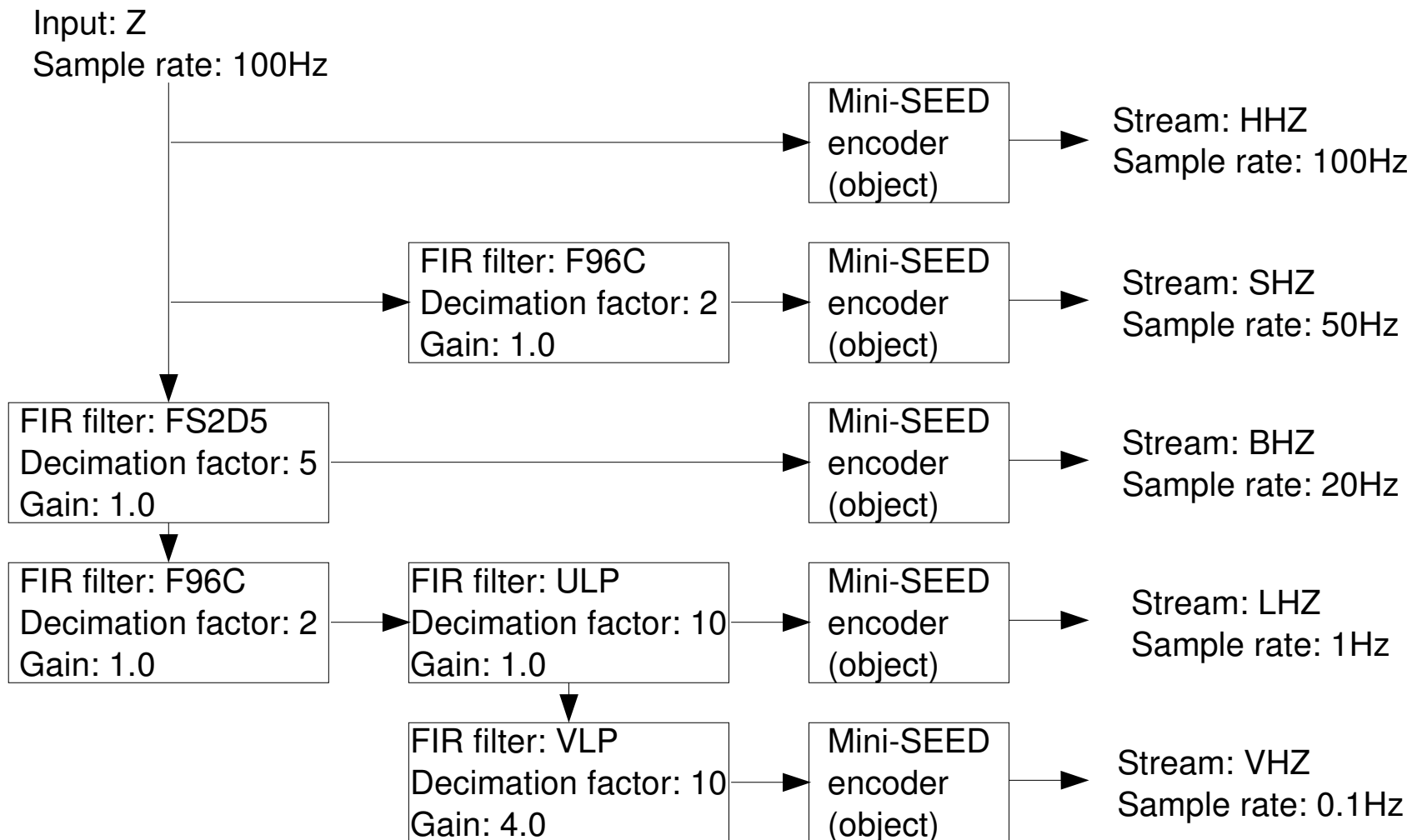
Plugin Interface

- ◆ A plugin is just a normal program that sends data to file descriptor 63 (opened by SeedLink before it executes the plugin). Using shell syntax, it is possible to redirect fd 63 to a file and run a plugin independently of SeedLink for testing purposes.
- ◆ In SeedLink, there is no one-to-one relation between stations and plugin instances, therefore each data packet sent by a plugin to file descriptor 63 contains **station ID**, which tells SeedLink to which station this data belongs. If SeedLink does not know this station, the data is ignored and a warning is written to log file.
- ◆ If a plugin sends Mini-SEED data directly to I/O System, bypassing the StreamProcessor, then the data will not be modified by SeedLink and the plugin is fully responsible that the data is correct big-endian Mini-SEED with 512-byte record size and has correct station and network codes.

Stream Processor

- ◆ While Mini-SEED streams go straight to the I/O System, raw streams pass the StreamProcessor module. StreamProcessor is enabled by pointing the parameters *streams* and *filters* in *seedlink.ini* to respective configuration files (normally *streams.xml* and *filters.fir*). If StreamProcessor is not enabled, then a warning is shown when raw data is received and the data is ignored.
- ◆ Each raw data packet sent by a plugin is labeled with **channel ID** in addition to station ID, so the StreamProcessor knows which packet belongs to which channel. Sometimes the channel IDs are hardcoded within a plugin, but often they can be defined by a user in *plugins.ini* file.
- ◆ When configuring the StreamProcessor, it is very important to make sure that channel IDs and sample rates match the defined inputs of the **stream processing scheme** defined in *streams.xml*, which is selected by the *proc* attribute of a station definition in *seedlink.ini*.

Stream Processing Scheme (data flow)



Stream Processing Scheme (XML definition)

- ◆ The stream processing scheme on previous slide can be defined in file *streams.xml* as follows:

```
<proc name="example">
  <tree>
    <input name="Z" channel="Z" location="" rate="100"/>
    <input name="N" channel="N" location="" rate="100"/>
    <input name="E" channel="E" location="" rate="100"/>
    <node stream="HH"/>
    <node filter="F96C" stream="SH"/>
    <node filter="FS2D5" stream="BH">
      <node filter="F96C">
        <node filter="ULP" stream="LH">
          <node filter="VLP" stream="VH"/>
        </node>
      </node>
    </node>
  </tree>
</proc>
```

I/O System

- ◆ I/O system in the part of SeedLink that manages ringbuffers and TCP connections. The size of ringbuffer is defined by segments (number of files) and segsz (size of one file in 512-byte records) parameters in *seedlink.ini*.
- ◆ Each station has its own ringbuffer, whose size can be individually defined. A ringbuffer consists of Mini-SEED files which can be analyzed for debugging purposes. It is not allowed to modify the ring buffer directly, except removing all files when SeedLink is not running.
- ◆ Each Mini-SEED record in a ringbuffer has a sequence number. If a client requests a record with unexisting sequence number, then transfer will start with the first record or the newest record, depending on the parameter seq_gap_limit in *seedlink.ini*.

Stream Monitor

- ◆ StreamMonitor is activated by setting the parameter stream_check in *seedlink.ini* to “enabled”. In this case the time spans and gaps of streams are monitored (according to parameters gap_check_pattern and gap_treshold) and the information can be requested from the server using **slinktool**. The stream information is also required for **time window extraction**.
- ◆ The value of gap_treshold must not be too low—in case of some digitizers, small gaps between records are normal due to timing errors. Also, gap_check_pattern should be set such that triggered streams are not included. Keeping track of a large number of gaps may require more memory than available, causing a system crash.

buffer.xml files

- ◆ When SeedLink exits, it saves the stream information of each station to a file called *buffer.xml*, which is located in a subdirectory within a ringbuffer. In this case SeedLink can start fast, because it can restore the stream information without scanning the disk buffer. However, if the parameters *stream check*, *gap check pattern* or *gap treshold* were meanwhile changed, the stored stream information may not be valid. In this case the *buffer.xml* files of all stations must be removed before starting SeedLink, so it will re-scan the ringbuffers.
- ◆ Scanning the ringbuffers may take long time (depending on their size and harddisk speed) and during that time it is not possible to stop SeedLink (except with “kill -9”). The progress of scan is indicated by messages in log file.

Testing SeedLink with slinktool

- ◆ Using the command-line utility **slinktool**, it is possible to check which clients are connected, what is the status of connections (eg., if there are many packets in the Queue, the client may be hanging or there may be network errors that prevent it from getting data). If Stream Monitor is enabled, slinktool can also show the timespans of streams in the buffer. Access to this information is subject to *info* and *info_trusted* parameters in *seedlink.ini*.
- ◆ Data retrieval can be also tested with slinktool. For example, the following command will request station APE from the GEOFON server and print verbose information about every packet received:

```
slinktool -vvv -ppp -S GE_APE geofon.gfz-potsdam.de:18000
```

Testing SeedLink with telnet

- ◆ Alternatively, it is possible to connect to SeedLink with telnet and use SeedLink commands directly. For example:

```
$ telnet localhost 18000
Trying 139.17.3.25...
Connected to geofon.gfz-potsdam.de.
Escape character is '^]'.
HELLO
SeedLink v3.0 (2004.129) GEOFON DC
BYE
Connection closed by foreign host.
```

- ◆ Clients can acquire data from SeedLink only via TCP/IP, even if they are running on the local machine. When a client starts, the SeedLink commands it sends are listed in the SeedLink log file.

Restricting SeedLink access

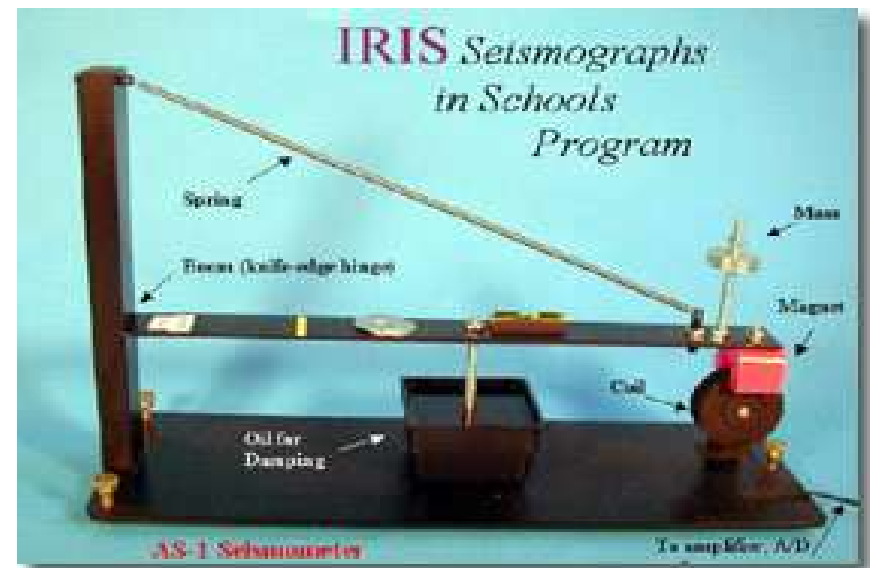
- ◆ Parameters info and info_trusted in *seedlink.ini* can be used to limit access to information obtained via INFO command or **slinktool** options -L, -Q, -G and -C. If the connecting IP matches the value of trusted (by default 127.0.0.0/8, eg., localhost), the parameter with “trusted” suffix is used.
- ◆ Similarly, time window extraction (TIME command or **slinktool** option -tw) can be restricted using parameters window_extraction and window_extraction_trusted in *seedlink.ini*.
- ◆ Access to any individual station can be denied via the access attribute in station definition or the default access parameter in *seedlink.ini*. If the connecting IP does not match, then the station will be completely hidden from the client (this applies to all SeedLink commands, including TIME, INFO, DATA and FETCH).



Part 3: Implementing a SeedLink plugin

AS-1

- ◆ AS-1 is an amateur seismograph system supported by IRIS in frames of their Education and Outreach program. AS-1-compatible digitizer with 16-bit resolution has been designed in GFZ. The digitizer sends data in ASCII format (one sample per line) via RS-232 serial interface. It has no internal timing.
- ◆ As a practical exercise we will show how to implement a SeedLink plugin for the digitizer in Python language.





Problems

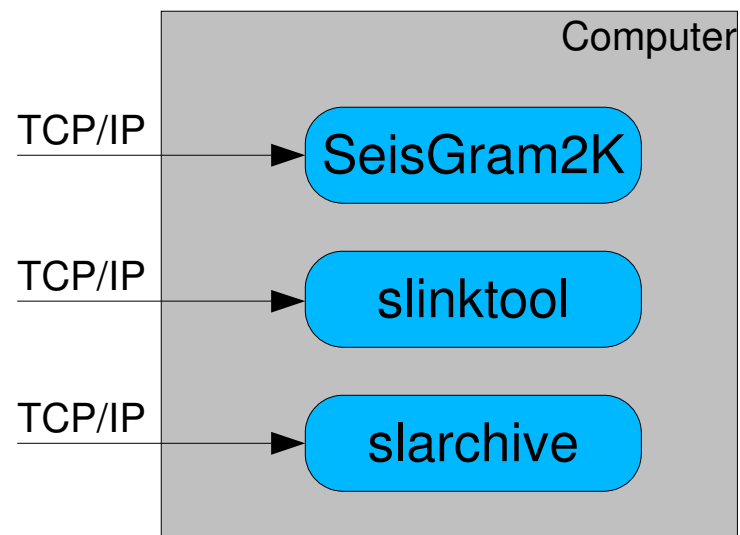
- ◆ Since the digitizer has no internal timing, PC clock must be used as a timing source. Due to the multi-tasking nature of Linux, this is not very accurate.
- ◆ SeedLink requires a constant sample rate, which must be expressed as a rational number. We found that the sample rate of AS-1 fluctuates between 19.93 and 19.94 Hz, which causes some gaps/overlaps between records.



Part 4: chain_plugin and triggered streams

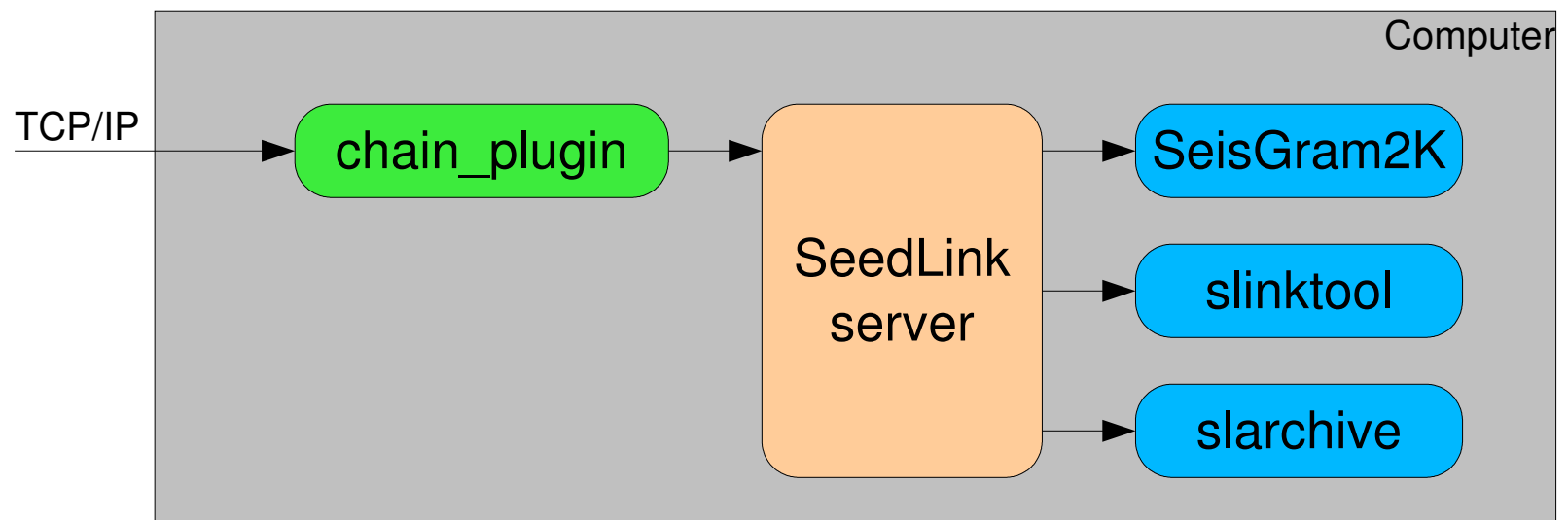
Running SeedLink clients without local server

- ◆ In order to use a SeedLink client, such as **SeisGram2K**, it is not required to install a local SeedLink server. However, it is strongly recommended to install a local SeedLink server if several clients are used simultaneously, because otherwise several copies of the data are transferred over the network. The following scheme depicts the data flow in this type of installation:



Using local SeedLink server and chain_plugin

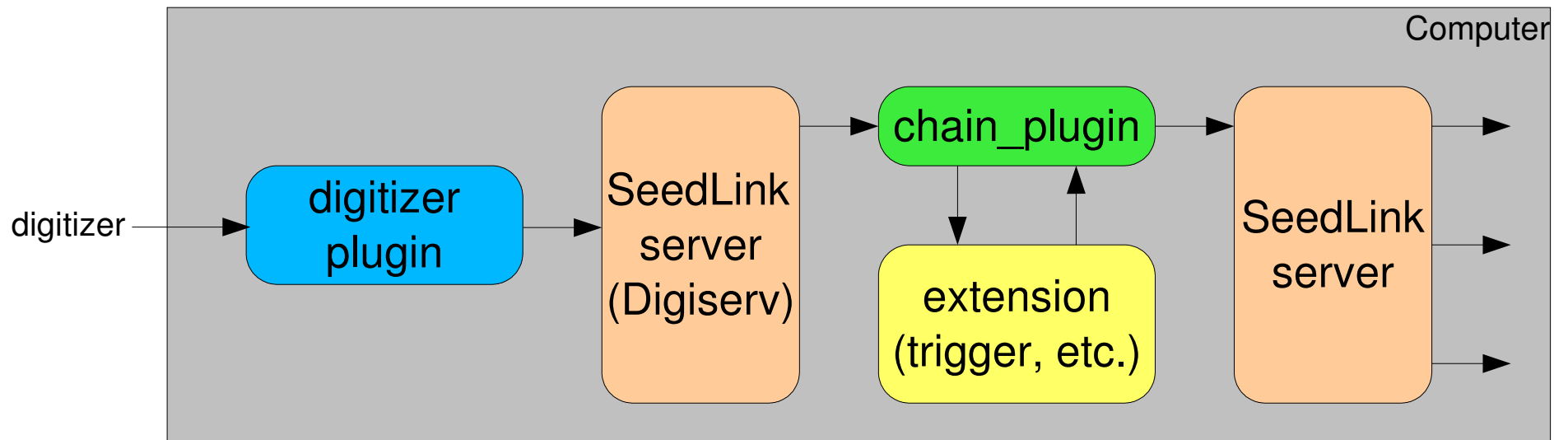
- ◆ **chain_plugin** is a SeedLink plugin that retrieves data from remote SeedLink servers, acting as a SeedLink plugin and client at the same time. Therefore it is possible to install a secondary SeedLink server as a hub for local clients. That saves network bandwidth, because only one TCP/IP connection is made to the remote server independently of the number of local clients. This type of installation is shown on the following figure:



chain_plugin extension interface

- ◆ chain_plugin extension interface was developed as a basis for real-time processing tasks, such as triggering, which cannot be performed by clients. It is based on SeedLink plugin interface—in principle any SeedLink plugin can be used as a chain_plugin extension—with the following additional functions:
 - reading Mini-SEED data from a pipe;
 - sending commands to chain_plugin.
- ◆ If the data comes from a local digitizer, it is necessary to run two instances of the SeedLink server in order to use chain_plugin. The primary server instance is then called **Digiserv** (“digitizer server”).

Data flow with SeedLink, Digiserv and chain_plugin



Defining triggered streams in chain.xml

- ◆ Triggered streams are defined by adding *trigger* elements in *chain.xml*. For example:

```
<station id="OKC" name="OKC" network="CZ" selectors="">
  <trigger src="HHZ" buffer_length="120"
    pre_seconds="80" post_seconds="40"/>
  <trigger src="HHN" buffer_length="120"
    pre_seconds="80" post_seconds="40"/>
  <trigger src="HHE" buffer_length="120"
    pre_seconds="80" post_seconds="40"/>
</station>
```

How does the triggering work?

- ◆ **Each station** defined in *chain.xml* has an internal property called “trigger state”. *chain_plugin* acts as a switch—when trigger state is on, it lets triggered streams through, when trigger state is off, it doesn't. Trigger state can be changed by a *chain_plugin* extension by using “trigger on” and “trigger off” commands with station ID and time as arguments.
- ◆ Each triggered stream has a buffer, whose size in seconds is defined by the *buffer_length* attribute. The buffer makes possible to switch the trigger on in the past. The attribute *pre_seconds* specifies how many seconds before “trigger on” time the triggered streams are switched on. Similarly, *post_seconds* specifies how many seconds after “trigger off” time the triggered streams are switched off.

Detector

- ◆ The “trigger on” and “trigger off” commands are sent to chain_plugin by a detector that is defined as a chain_plugin extension by adding an “extension” element to *chain.xml*. For example:

```
<extension
  name = "trigger"
  filter = ".*_HH._D"
  cmd = "python /home/sysop/python/trigger_ext.py"
  recv_timeout = "0"
  send_timeout = "60"
  start_retry = "60"
  shutdown_wait = "/>
```

- ◆ The attribute *filter* (regular expression to match NET_STA_LOC_CHA_TYPE) defines which streams are sent to the detector. Note that it is **not** required to run detector on the same streams that are triggered. Detector parameters, such as STA and LTA length, etc., are defined in the file *trigger.ini*.

Advanced options of chain_plugin

- ◆ Network and station codes can be overridden with *out_station* and *out_network* attributes of “station” element in *chain.xml*.
- ◆ Streams can be renamed using the “rename” element.
- ◆ Raw streams can be created from Mini-SEED streams using the “unpack” element. Such streams can be downsampled by SeedLink.

```
<station id="TIRR" name="TIRR" network="RO"  
  out_network="GE" selectors="!SN? !SH?">  
  <rename from="SG?" to="SN?" />  
  <rename from="BH?" to="SH?" />  
  <unpack src="SHZ" dest="Z" double_rate="yes" />  
  <unpack src="SHN" dest="N" double_rate="yes" />  
  <unpack src="SHE" dest="E" double_rate="yes" />  
</station>
```



Part 5: Shortcomings of the present SeedLink implementation, ideas for future development

Open files limitation

- ◆ Each SeedLink connection requires maximum $(N+1)$ file descriptors, where N is the number of stations requested. Only 1024 file descriptors (`FD_SETSIZE`) can be used in `select()` call on Linux, therefore $C * (N + 1)$, where C is the maximum number of connections and N is the total number of stations, must be less than approx. 1000 to run the server safely (eg., 30 stations and max. 30 connections).
- ◆ Recently file descriptor reallocation was implemented in SeedLink, so file descriptors not needed in `select()` will be placed above 1023. In this case up to 1000 connections can be supported and the maximum number of stations can be up to $1,000,000/C$, where C is the maximum number of connections (eg., 1000 stations and 1000 connections). Even though it seems plenty, it may not be enough for large hubs with thousands of stations and completely open access (eg., IRIS DMC).

INFO requests

- ◆ When a client issues the INFO command, the complete XML document is generated in the server memory and kept there until it has been fully transferred to the client. If several clients issue an INFO request at the same time, the server may run out of memory. Using temporary files would be one solution.
- ◆ INFO command does not allow to select specific stations/attributes, so even if the client is interested only in the start time of the ringbuffer of a specific station, it has to retrieve and parse the full station list. In case of high volume servers, it would be useful if the INFO command allows to exactly specify which stations and which attributes the user is interested in.
- ◆ XML data could be optionally compressed.



Dynamic configuration

- ◆ In the present implementation, every station must be manually configured and SeedLink must be restarted to re-read the configuration files. Such human interaction is not appropriate for large hubs, where several stations might be added every day. In this case, any new station arriving from plugins should be added to the server configuration automatically.
- ◆ New stations added to the server will not be sent to clients unless explicitly requested. It is necessary to manually check remote servers for any new stations. This problem would be solved by protocol extension, allowing wildcards in station names.
- ◆ The combination of above two options would make possible to install SeedLink hubs that work without any manual intervention.



Connection statistics

- ◆ Currently the server does record connection statistics like which data is delivered to which IP. It would be possible to parse the server log file and generate some statistics but it would not be accurate.
- ◆ One solution is to produce a special log file from where exact statistics can be generated, however, due to the nature of SeedLink (anyone can set up his own server and re-distribute the data), completely reliable statistics is not possible.

Distributed servers

- ◆ In case of large hubs, it would be useful if a client can transparently use a single IP even if the server is internally distributed on several computers. There could be even mirrored sub-servers to distribute load in case of a large number of connections. An additional advantage is that if one of the mirrors fails or is shut down for maintenance, then another one can transparently take over the connections.