Lab::Measurement

Instrumentation control with Perl –
The Next Generation

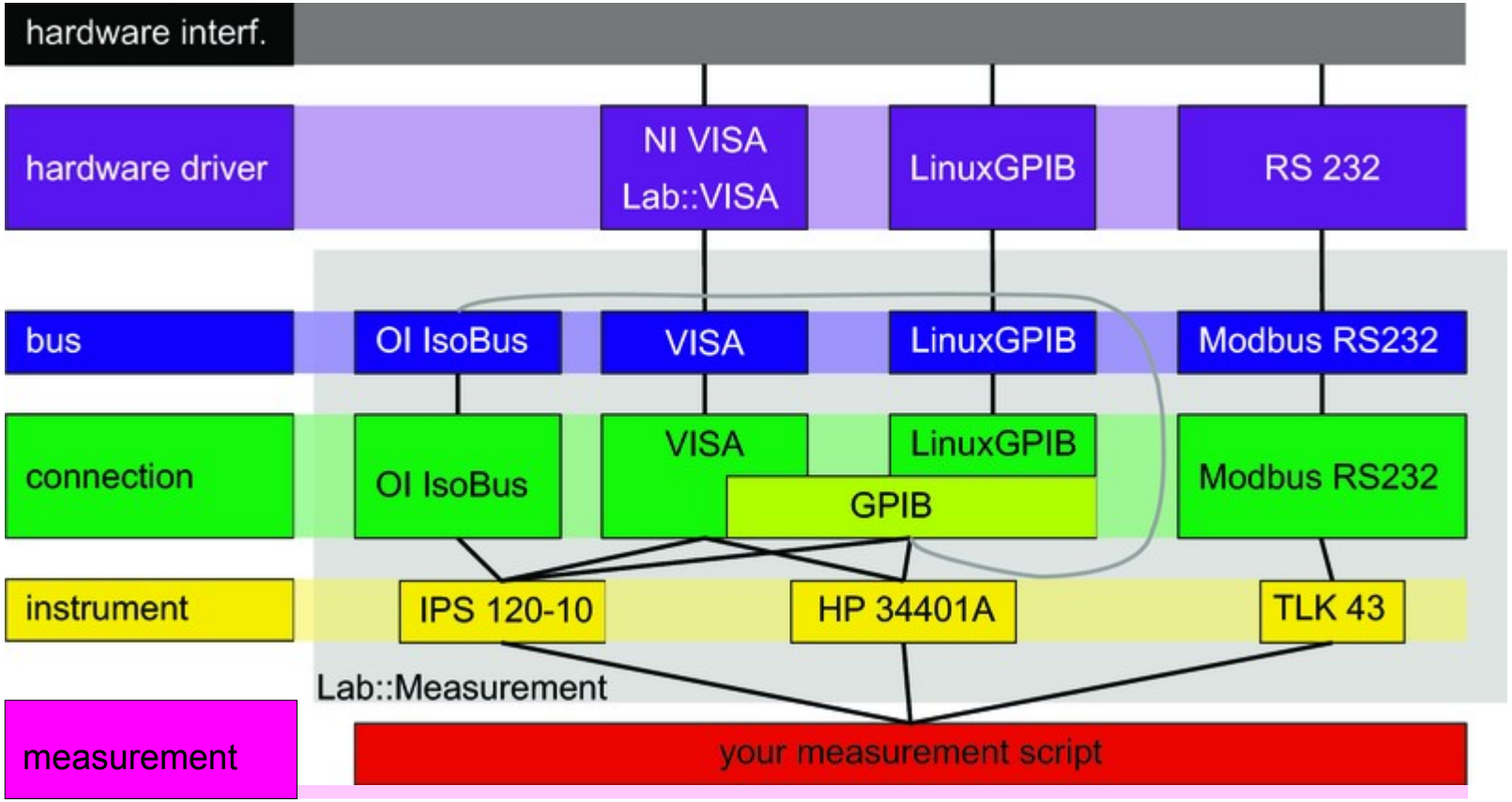
Lab::Measurement is written in Perl

- Written in Perl (and some low-level stuff in C)
- To be used from programs written in Perl

- Perl: interpreted scripting language
- Runs on almost every OS
- Extremely good in reading data files, manipulating data, etc.
- Allows to write quick and dirty scripts that get the job done
- Also allows to write clean fullsize programs

=> ideal for experimental physics

Modular architecture of the core stack



… looks complicated, but:

Using Lab::Measurement

```
#!/usr/bin/perl

use strict;
use Lab::Instrument::HP34401A;

my $hp_gpib=$ARGV[0];

print "Reading HP34401A at GPIB address $hp_gpib\n";

my $hp=new Lab::Instrument::HP34401A(
        connection_type=>'LinuxGPIB',
        gpib_address => $hp_gpib,
        gpib_board=>0,
);

my $volt=$hp->$get_voltage_dc(10,0.00001);

print "Result: $volt V\n";
```

A real measurement!

- How about doing a gate sweep, 100 steps from 0V to 1V?

```
#!/usr/bin/perl

use Lab::Instrument::HP34401A;
use Lab::Instrument::Yokogawa7651;

# Connect to gate voltage source
my $yoko=new Lab::Instrument::Yokogawa7651({
    connection_type => 'LinuxGPIB',
    gpib_address    => 14,
    gate_protect    => 0,
});
# Connect to multimeter
my $hp=new Lab::Instrument::HP34401A(0, 21);

for (my $volt=0; $volt<=1; $volt=$volt+0.01) {
    # set Yokogawa
    $yoko->set_voltage($volt);

    # wait a second
    sleep(1);

    # read multimeter
    my $vmeas=$hp->read_voltage_dc();

    # print values
    print $volt,"\t",$vmeas,"\n";
}
```

Complex measurements possible

- Not just one input, multi-dimensional measurements
- Read out as many instruments as you like, as often as you like
- Combine with monitoring of setup (pressures, temperatures, …)

- Low-level aspects (bus, connection, interface)
  - Want to access as many devices as possible
  - Linux / Windows
  - Different device drivers, different hardware interfaces
  - Thread-safe

- High-level aspects (your script, measurement)
  - Want to make measuring as comfortable as possible
  - Access different hardware with same / similar commands
  - Keep track of metadata
  - Plotting during measurement, easy re-plotting afterwards

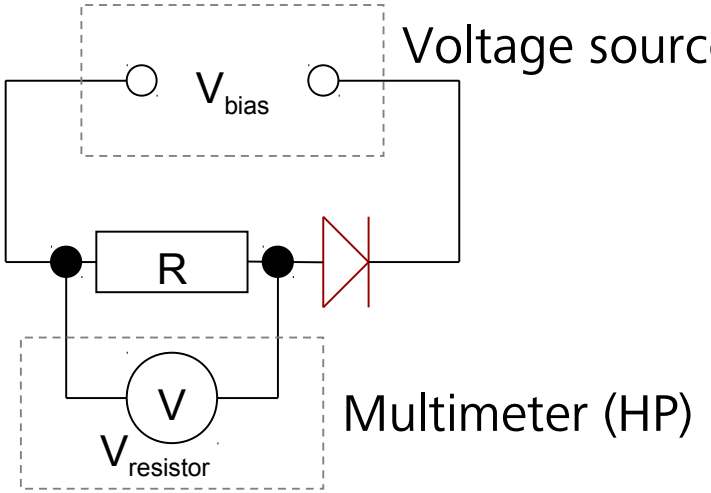
High-level classes: Lab::Measurement & friends

- Provide additional tools to write better measurement scripts

- Store metadata alongside data
  - date and time
  - settings of additional instruments
  - ratio of voltage divider
  - color of the shirt you are wearing
  - everything that might be important for a later interpretation of the data

- Don't repeat yourself
  - Use above collected information automatically
  - Automatically plot data with correct axes, scaling, labels etc.

High-level classes: metadata philosophy



Constants

- R = 1000 Ω

Columns

- C1: V_{bias}
- C2: $V_{resistor}$
- unit
- description…

1.5	+8.19300500E-01
1.4	+7.23413000E-01
1.3	+6.28083900E-01
1.5	+8.19300500E-01

Axes

- Axis "bias voltage" (C1)
- Axis "diode current"(C2 / R)
- Axis "diode resistance" (R * (C1 / C2 – 1))
- unit
- expression
- description…

Plots

- Plot "diode current" axis "diode current" over axis "bias voltage"
- Plot "diode resistance" axis "diode resistance" over axis "bias voltage"
- logscale
- grid
- ranges…

Using Lab::Measurement

```
my $measurement=new Lab::Measurement(
    sample          => $sample,
    title           => $title,
    filename_base   => 'zener_kennlinie',
    description     => $comment,

    live_plot       => 'diode current',


    constants       => [
        {
            'name'          => 'R',
            'value'         => '1000',
        },
    ],
    columns         => [
        {
            'unit'          => 'V',
            'label'         => 'V_{bias}',
            'description'   => 'Bias Voltage',
        },
        {
            'unit'          => 'V',
            'label'         => 'Amplifier output',
            'description'   => 'Voltage drop on serial resistor',
        }
    ],
    axes            => [
        {
            'unit'          => 'V',
            'expression'    => '$C0',
            'label'         => 'V_{bias}',
            'description'   => 'Bias voltage',
            'min'           => ($start_voltage < $end_voltage)
                               ? $start_voltage
                               : $end_voltage,
            'max'           => ($start_voltage < $end_voltage)
```

```
$measurement->start_block();

for (
    my $volt = $start_voltage;
    ($volt-$end_voltage)/$step < 0.5;
    $volt += $step
) {
    $knick->set_voltage($volt);
    sleep(1);
    my $meas = $hp->get_voltage_dc(10,0.0001);
    $measurement->log_line($volt,$meas);
}

my $meta = $measurement->finish_measurement();
```

Run measurement!

Describe measurement!

Result: two files per measurement

„.dat“: your measurement data, in simple text format

```
# comment
# comment
x1      x2      x3      y1      y2      y3
x1      x2      x3      y1      y2      y3
...
```

„.meta“: all the additional metadata in computer-readable xml

- can be read out and evaluated again
- can be used to automatically replot the data for example

Other high-level features

- "**Gate protect**" safety mechanism
  - Makes sure that no voltage is changed to fast
  - Big voltage steps are automatically split into small, slow steps

- **Date/time** handling
  - Date/time column can contain timestamp for every data point
  - Plot data as function of time

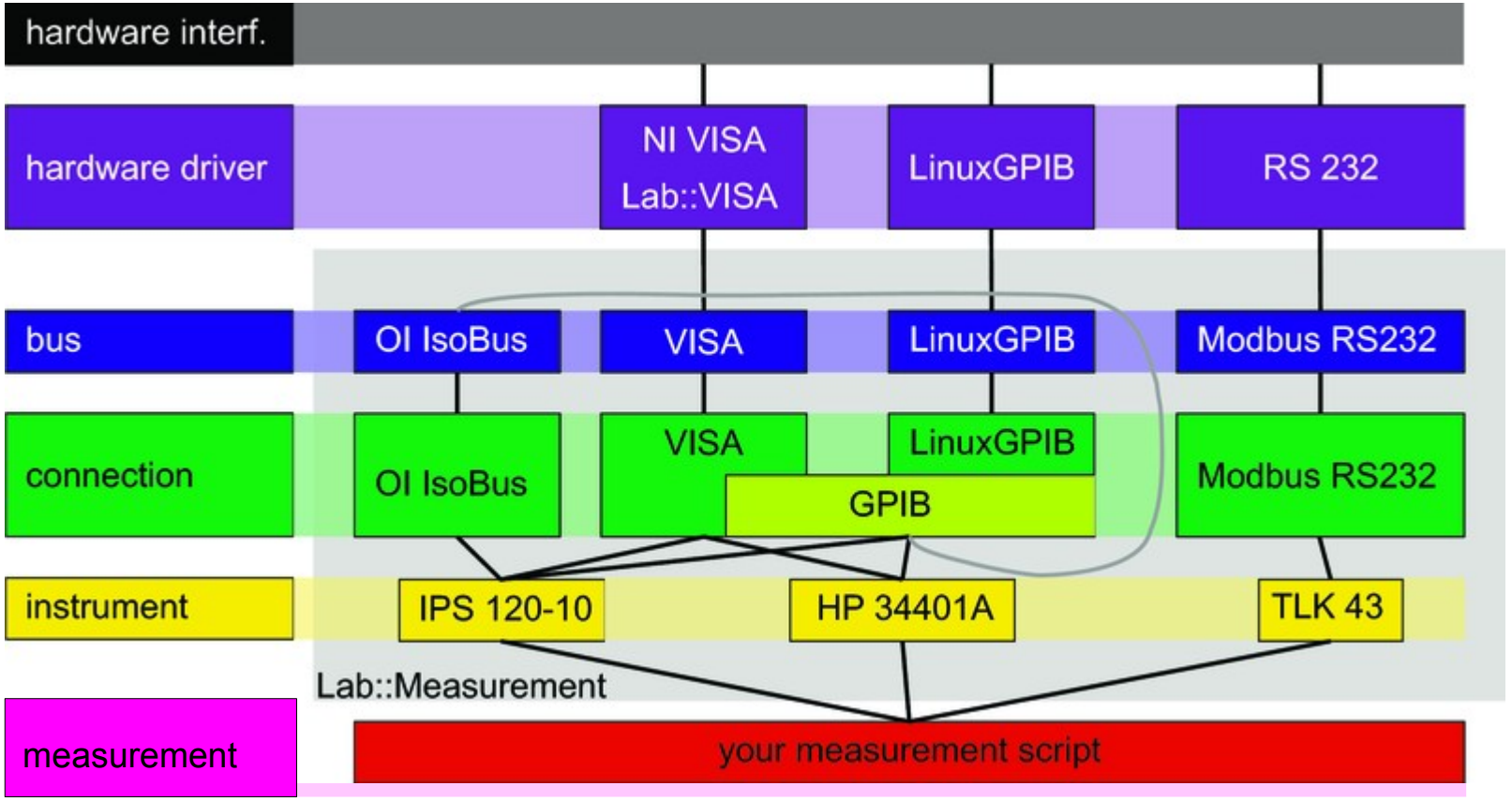
- Measurements with **higher dimensionality**
  - Each trace/sweep/line is a "block"
  - Two-dimensional plots, selections of traces, etc.

Utilities: plotter.pl, make_overview.pl

- **plotter.pl**:
  - Reads measurement file, list avaible plots (axes etc)
  - Creates postscript or pdf output

- **make_overview.pl**:
  - Reads all measurements e.g. in a directory
  - Generates a postscript or pdf file with plots for each measurement, including the metadata (i.e. constants, parameters, color of your shirt) (via LaTeX)
  - Great for completing your lab book

Modular architecture of the core stack



Internal architecture

- Divided into several layers
- **Lab::Measurement** is the highest abstraction layer. Provides support for writing good measurement scripts. Offers means of saving data and meta information to disk, plotting data, etc.
- **Lab::Instrument** package makes communication with instruments easier by silently handling the protocol involved
- **Lab::Connection** and **Lab::Bus** handle communication with the hardware and encapsulate the actual device drivers.
- The lowermost layer is given by the hardware driver and its Perl binding. Several backends are supported, e.g. NI-VISA via **Lab::VISA** or **LinuxGPIB**.

VISA, GPIB, etc.

- Instruments can be connected in various ways:
  Serial port, GPIB, VXI, TCP/IP, USBtm, ...

- **GPIB** (hardware and software)
  - GPIB (IEEE488): Standard by Hewlett-Packard
  - Physical layer IEEE488.1
  - Command layer IEEE488.2
  - SCPI (Standard Commands for Programmable Instruments)

- **VISA** (software)
  - Virtual Instrument Software Architecture
  - VXI, GPIB, serial, or computer-based instruments
  - NI-VISA library is one implementation of the VISA standard

Open source, free software

- Open source, free software
- License: same as Perl (Artistic / GPL-2)

- Homepage: http://www.labmeasurement.de/
- Releases available from CPAN

- Development code and history on Gitorious
  https://www.gitorious.org/lab-measurement/lab

- Contributors welcome!