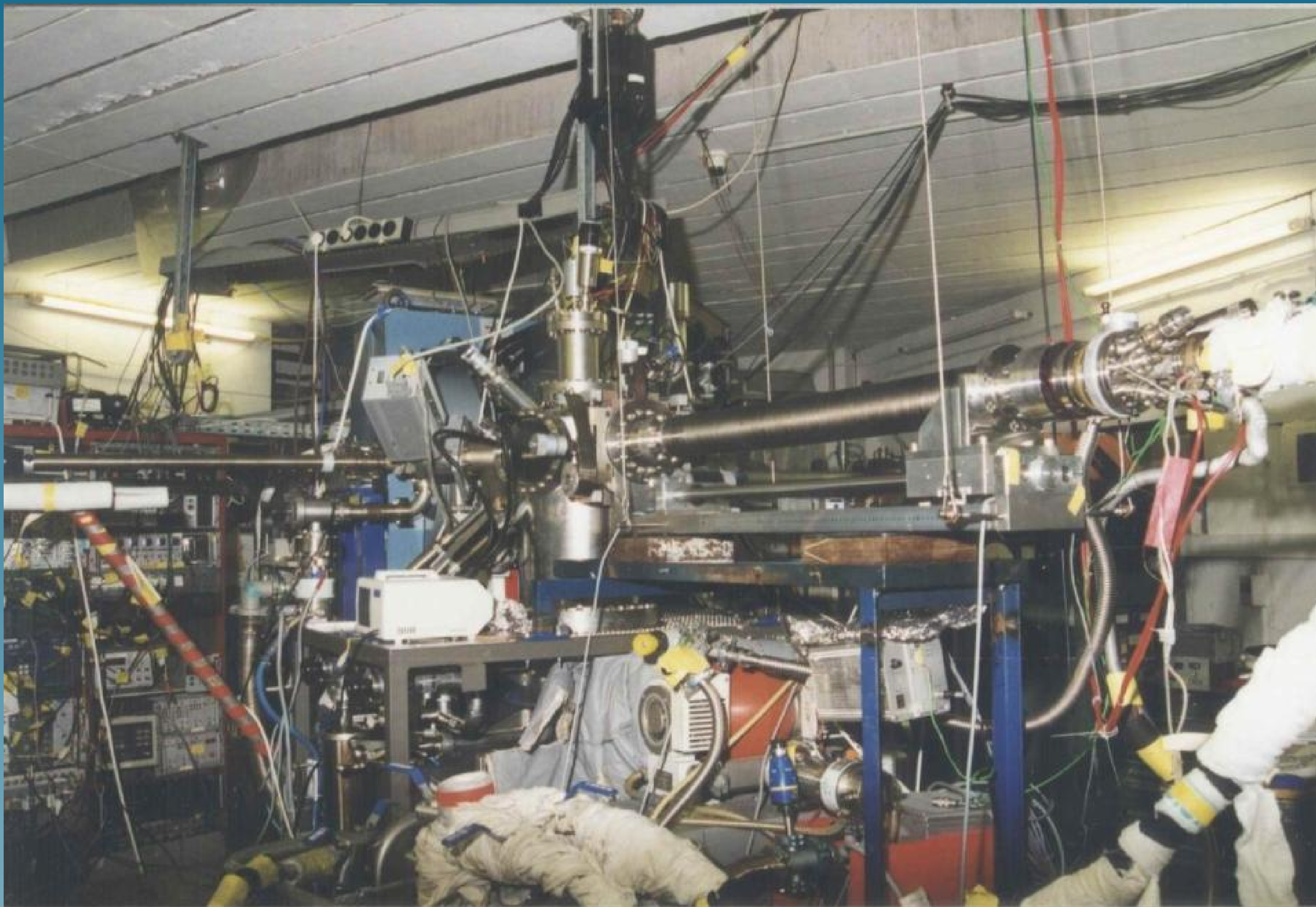
Real Time Measurement System

ELISABET

Extended LIthium Surface Analysis and BETa nmr

Oliver Kühlert, Richard Schillinger, Christian Bromberger and H. J. Jänsch

University of Marburg, Department of Physics / MPI for Nuclear Physics Heidelberg



Outline

- Motivation
- What is Realtime ?
- Kernel- and Userspace
- RTLinux
- Threads in the Kernel-Space
- Interfaces to RTLinux

- Threads in ROOT
- A C++ API for ROOT
- An example: TPD (Thermal Programmed Desorption)

Motivation

- Our old measurement system consists of two parts:
  - ⋆ a control and data acquisition program running on a DEC MicroVax (VMS)
  - ⋆ a lot of PAW-scripts running on an alpha (DIGITAL UNIX)

Motivation

- Our old measurement system consists of two parts:
  - ⋆ a control and data acquisition program running on a DEC MicroVax (VMS)
  - ⋆ a lot of PAW-scripts running on an alpha (DIGITAL UNIX)
- Due to some changes to the experimental setup and an increasing number of hardware failures on the VAX we were encouraged to develop a new system.
- We came to the decision the it should base on RTLinux and ROOT.

What is Realtime ?

An Operating System is capable of **realtime**, when it can react on dedicated requests within a guaranteed space of time.

What is Realtime ?

An Operating System is capable of **realtime**, when it can react on dedicated requests within a guaranteed space of time.

latency: the time elapsing between the request and the reaction of the system.

What is Realtime ?

An Operating System is capable of **realtime**, when it can react on dedicated requests within a guaranteed space of time.

latency: the time elapsing between the request and the reaction of the system.

“soft” realtime: the latency is distributed around a –preferably small– average value. There is no upper limit.

What is Realtime ?

An Operating System is capable of **realtime**, when it can react on dedicated requests within a guaranteed space of time.

latency: the time elapsing between the request and the reaction of the system.

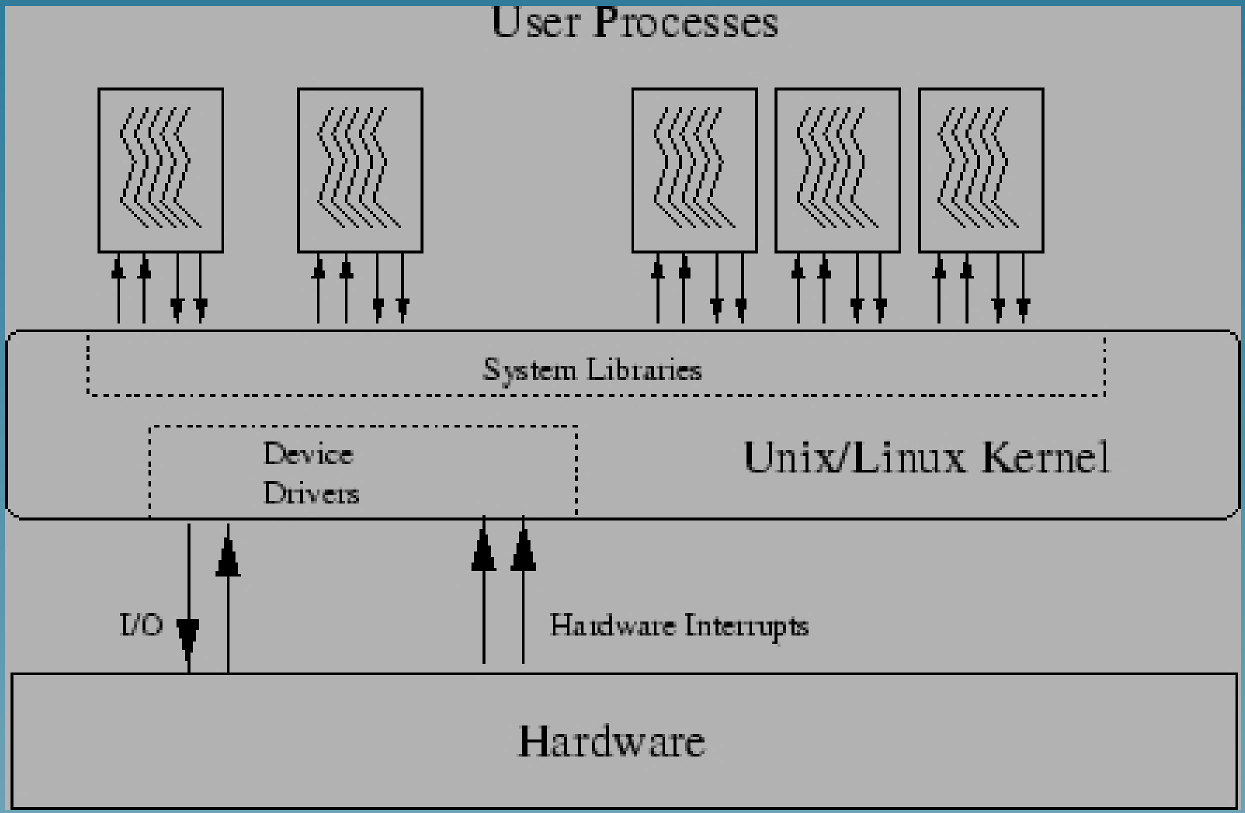
"soft" realtime: the latency is distributed around a –preferably small– average value. There is no upper limit.

"hard" realtime: There is a maximum latency within the system will have reacted certainly.

Linux-Kernel

- Linux differentiates between User- and Kernel-Space
- In contrast to User-Space there's only one process in the Kernel. (Kernel-Thread)
- Code must be linked against the entire Kernel.
- This can be done at runtime. (using `insmod`)

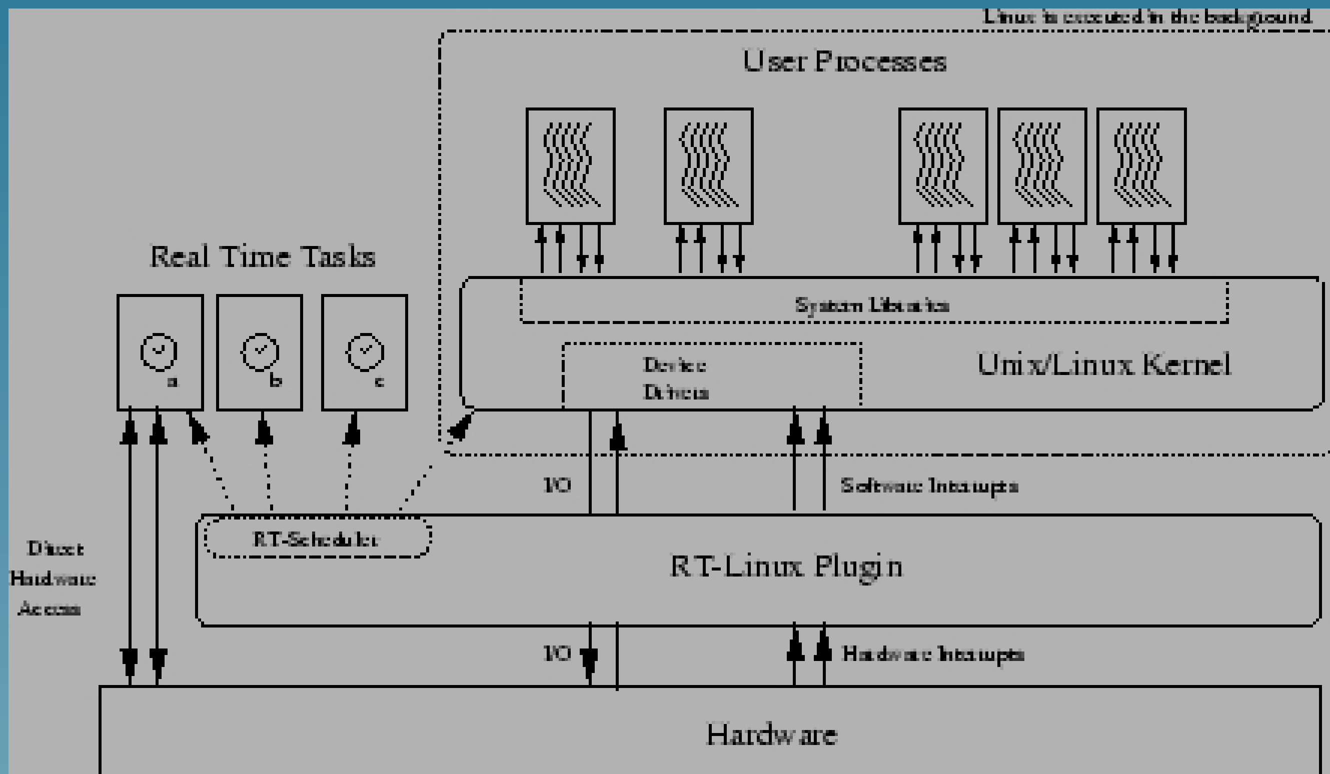
Linux-Kernel



RTLinux Extension

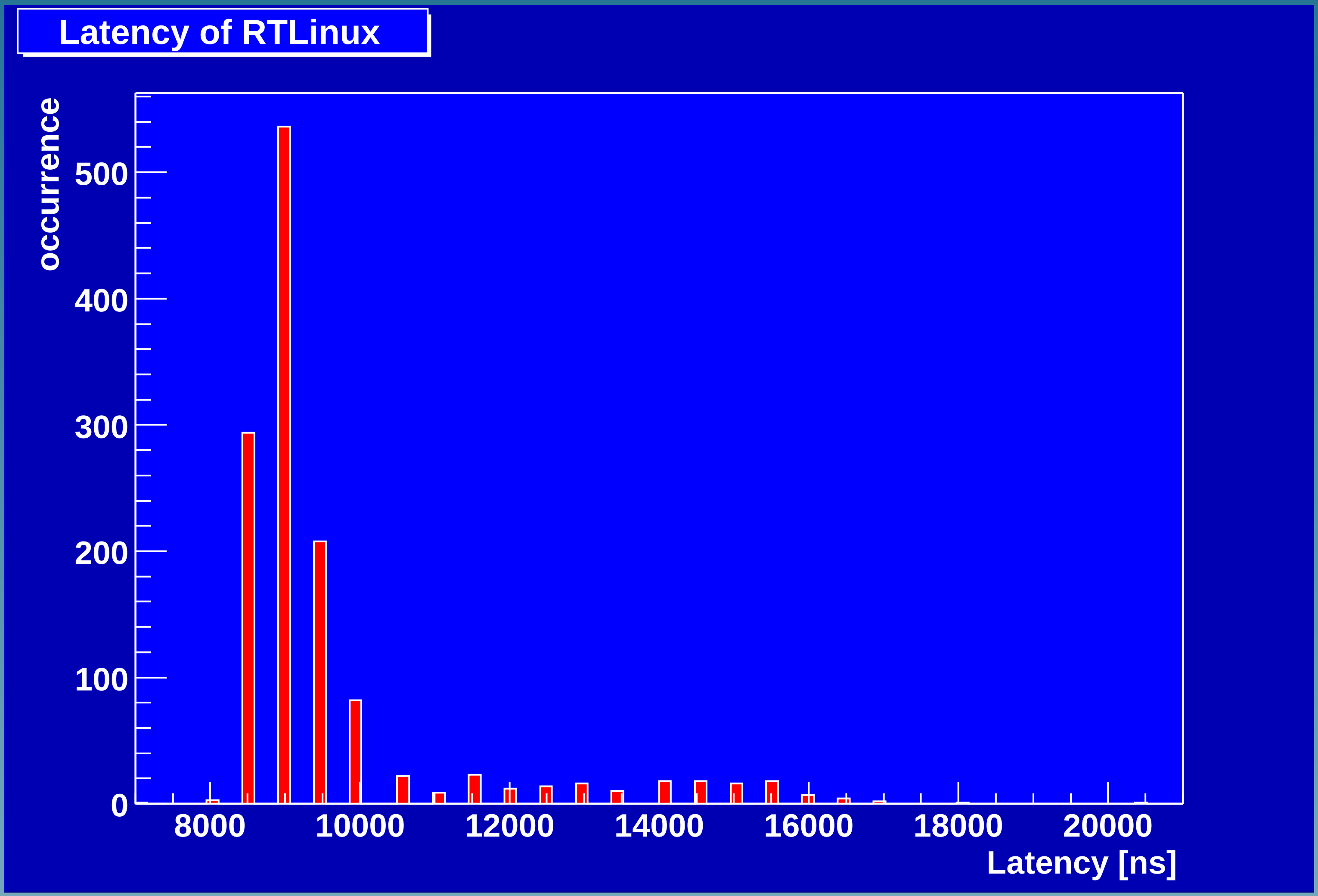
- RTLinux implements Threads to the Linux-Kernel.
- The Kernel-Thread (and so the whole Linux System) is running at lowest priority.
- Linux IRQs are transformed into soft-IRQs and handled by RTLinux at idle-time.
- latency: 20 μs.

RTLinux Extension

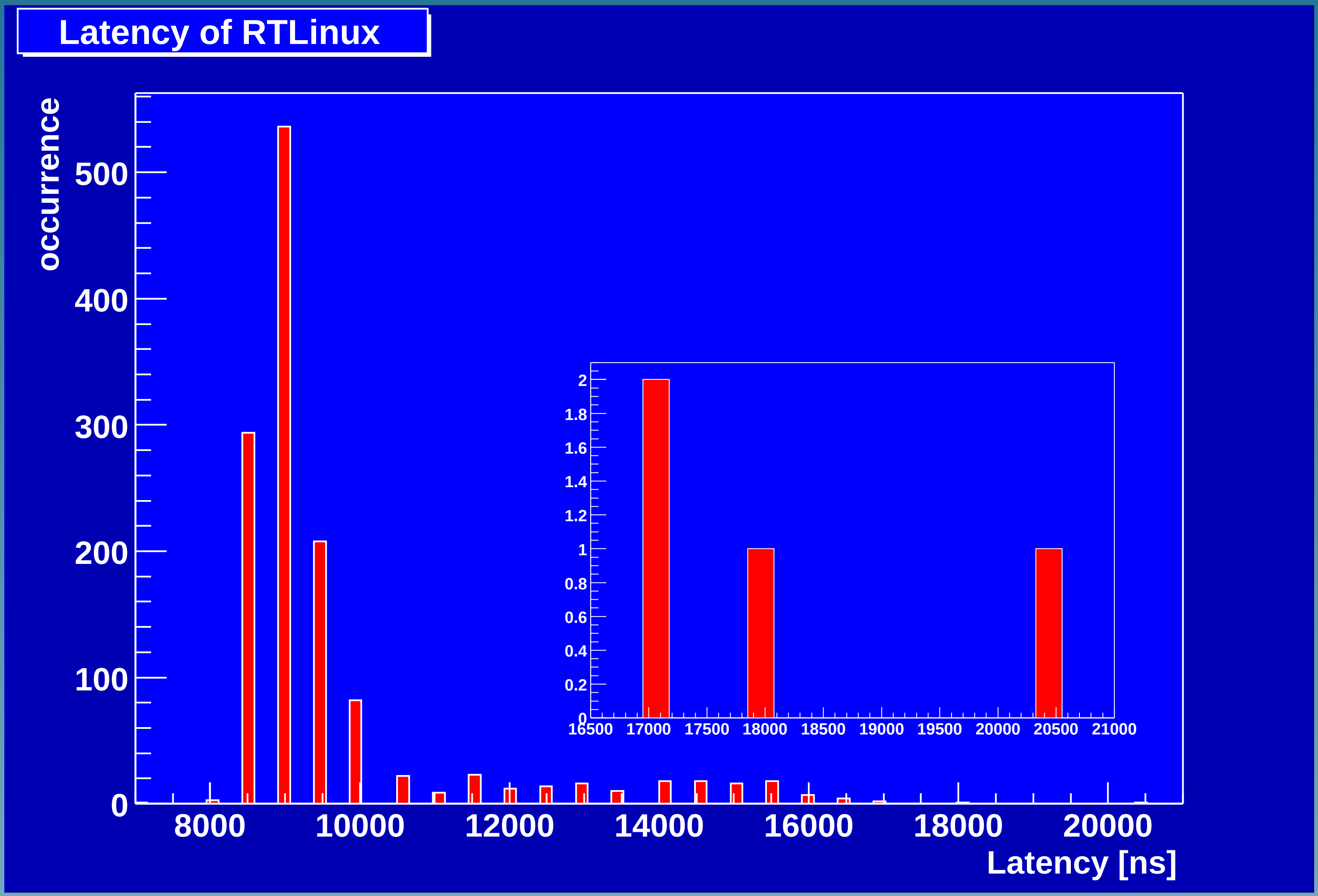


Graphics from: `GettingStarted.html` of the RTLinux documentation.

Latency


Latency of RTLinux
occurrence
0
100
200
300
400
500
8000
10000
12000
14000
16000
18000
20000
Latency [ns]

Latency


Latency of RTLinux
occurrence
500
400
300
200
100
0
8000
10000
12000
14000
16000
18000
20000
Latency [ns]
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
16500
17000
17500
18000
18500
19000
19500
20000
20500
21000

Interfaces to RTLinux

User-Space Applications can communicate with RTLinux modules in two different ways:

1. via FIFOs
   `(/dev/rtf0 - /dev/rtf63)`

2. by using shared memory.
   mbuff-driver

Threads in ROOT

Our Application uses FIFOs to communicate with RTLinux. There's a separate thread to extract the data from the stream.

ROOT must have been compiled with Thread support.

Starting a new Thread in ROOT:

```
thread=new TThread(UserFun, UserArgs);
thread->Run();
```

The starting point of the thread will be the function `UserFun` with the prototype

```
void UserFun(void* UserArgs);
```

Threads in ROOT

ROOT classes dealing with threads:

- TThread
- TMutex
- TCondition
- TSemaphore

C++ API

ELISABET basically consists of three major parts:

C++ API

ELISABET basically consists of three major parts:

- the realtime-component and the CAMAC-module for hardware access.

C++ API

ELISABET basically consists of three major parts:

- the realtime-component and the CAMAC-module for hardware access.
- code to access non-realtime based hardware components. (mostly RS-232 programming)

C++ API

ELISABET basically consists of three major parts:

- the realtime-component and the CAMAC-module for hardware access.
- code to access non-realtime based hardware components. (mostly RS-232 programming)
- An application and some additional tools to interface the measurement process to humans and to display the results respectively analyze the data. Therefore *ROOT* is used.

⇒Need of an API to interface the hardware components to ROOT.

command-module

```
class commandModule
{
public:
  static void SetDAC(byte N, byte A, byte F,
                     D24WORD dac);
  ...
  static void doRamp(byte N, byte A,
                     D24WORD startdac,
                     D24WORD enddac,
                     D24WORD step,
                     unsigned long long delta_t,
                     int hold=0);
  static void stopRamp();
};
```

CBaseExperiment

```
class CBaseExperiment
{
friend class CExperimentThread;
public:
 CBaseExperiment(byte experiment_type,
           unsigned long datapoints,
           unsigned long long delta_t);

 virtual void start();
 virtual void stop();

 int save(const char *path);

 char * getFileName();
```

```
 virtual ~CBaseExperiment();

protected:
  virtual void processData(int fdi)=0;
  virtual void getType(char * buf)=0;
  virtual void printHeader(FILE *f)=0;
  virtual void printData(FILE *f)=0;
  exp_data exp_block;
  TMutex mutex;
  int stopFlag;

private:
  virtual void Run();
  char fileName[256];
 };
```

CTDSExperiment

```
class CTDSExperiment:public CBaseExperiment
{
public:
  CTDSExperiment(byte ramp_N, byte ramp_A,
                 D24WORD ramp_start,
                 D24WORD ramp_end,
                 D24WORD ramp_step,
                 unsigned long long time_diff,
                 unsigned long datapoints,
                 unsigned long long delta_t,
                 unsigned long long wait_t,
                 CTDSMasses massinfo,
                 byte set_mass_N, byte set_mass_A,
                 byte qmssignal_N, byte qmssignal_A,
```

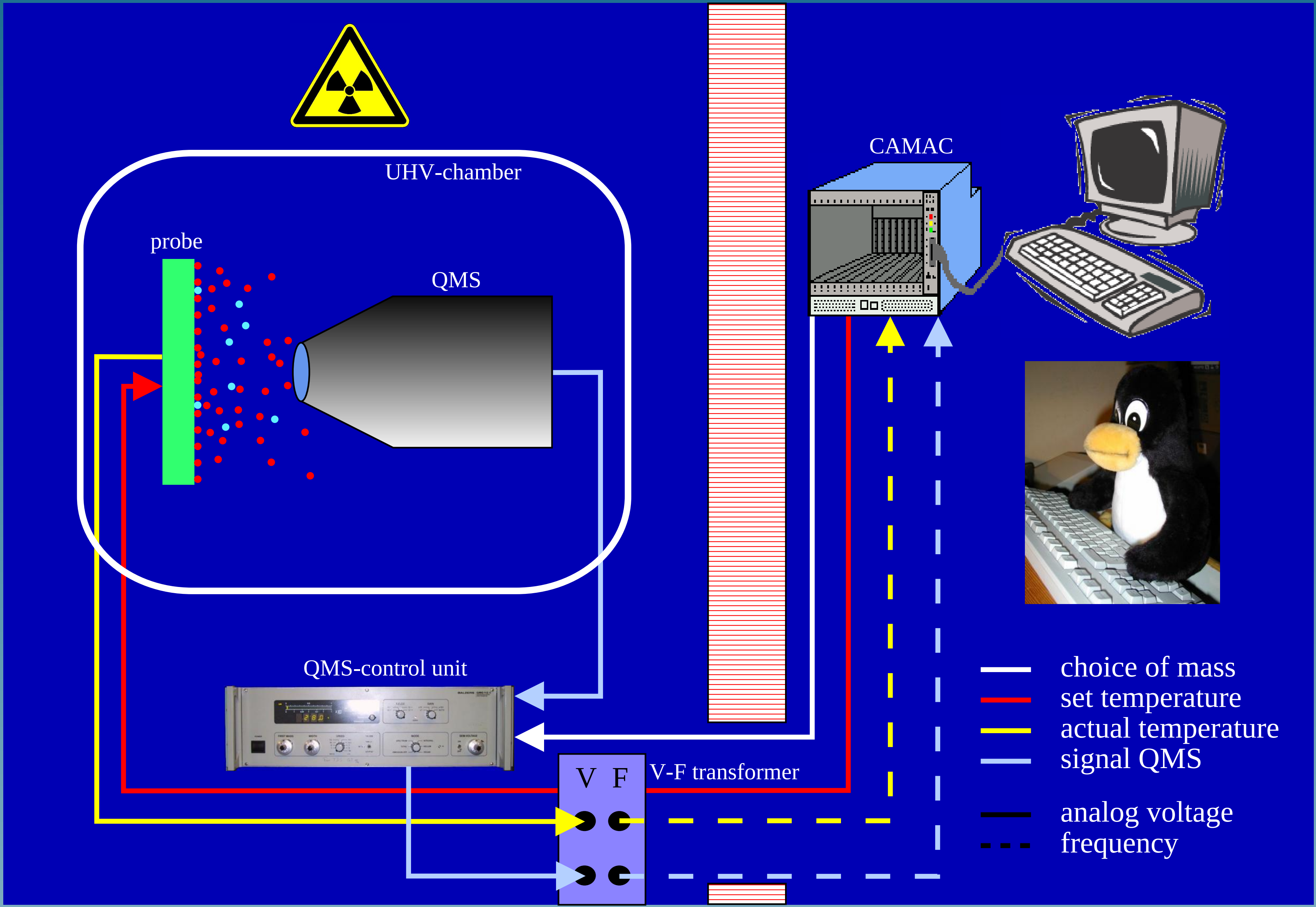
```
               unsigned long long temp_scan_t,
               byte temp_in_N, byte temp_in_A,

               byte range0_N,byte range0_A,
               byte range1_N,byte range1_A,

               byte gain0_N,byte gain0_A,
               byte gain1_N,byte gain1_A);

virtual ~CTDSExperiment();

virtual void start();
virtual void stop();

unsigned long getMaxData();

tdsdata operator[](unsigned long j);
```

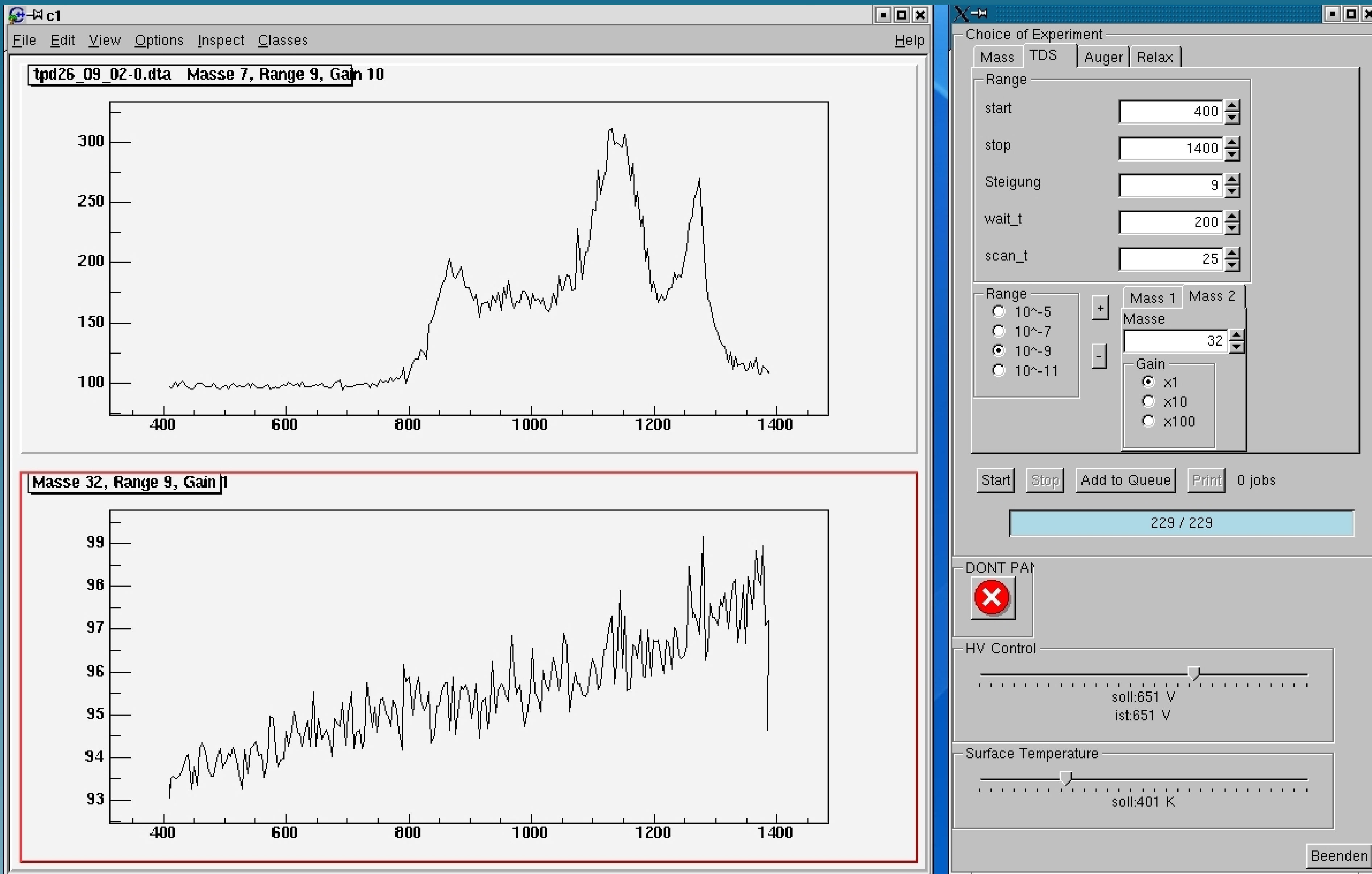
```
protected:
  virtual void processData(int fdi);
  virtual void getType(char *buf);
  virtual void printHeader(FILE *f);
  virtual void printData(FILE *f);

private:
  byte r_N;
  byte r_A;
  D24WORD r_start;
  D24WORD r_end;
  D24WORD r_off;
  unsigned long long r_t_diff;

  unsigned long counter;
  tdsdata *daten;
};
```

An example:TPD

Thermal Programmed Desorption

- Is a very important application in surface science.
- One can determine the number, type and chemical bond strength of particles covering the surface of a solid by this method.
- howto: driving an exact temperature-ramp; parallely: counting the desorbing particles with a quadrupole-mass spectrometer (QMS).


UHV-chamber
probe
QMS
CAMAC
QMS-control unit
V F
V-F transformer
choice of mass
set temperature
actual temperature
signal QMS
analog voltage
frequency


c1
File Edit View Options Inspect Classes Help
tpd26_09_02-0.dta Masse 7, Range 9, Gain 10
300
250
200
150
100
400 600 800 1000 1200 1400
Masse 32, Range 9, Gain 1
99
98
97
96
95
94
93
400 600 800 1000 1200 1400
Choice of Experiment
Mass TDS Auger Relax
Range
start 400
stop 1400
Steigung 9
wait_t 200
scan_t 25
Range
10^-5
10^-7
10^-9
10^-11
Mass 1 Mass 2
Masse
32
Gain
x1
x10
x100
Start Stop Add to Queue Print 0 jobs
229 / 229
DONT PA
HV Control
soll:651 V
ist:651 V
Surface Temperature
soll:401 K
Beenden