std::chrono

THE TIME LIBRARY YOU ALWAYS WANTED BUT WERE AFRAID TO ASK

BUG SQUASHING SEMINAR 2015-06-24

THOMAS KIRCHNER

T.KIRCHNER@DKFZ.DE

COMPUTER-ASSISTIERTE INTERVENTIONEN

DEUTSCHES KREBSFORSCHUNGSZENTRUM HEIDELBERG

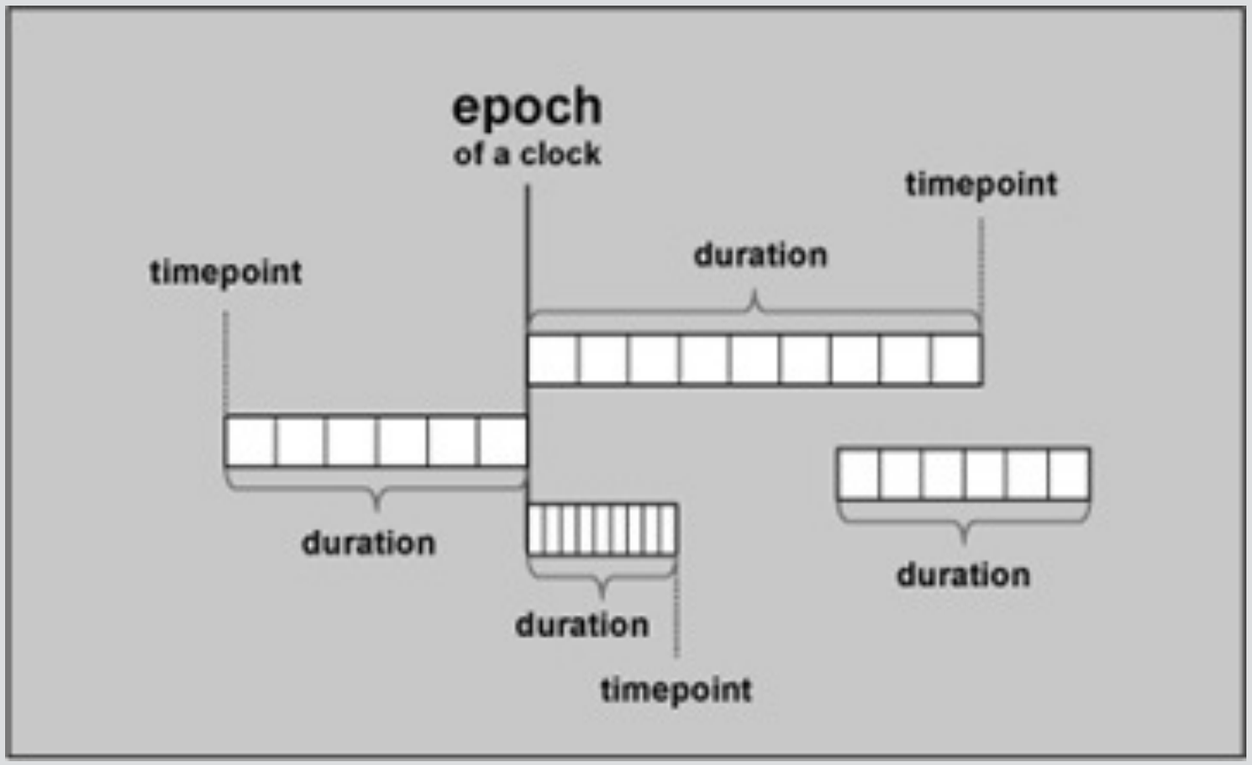
std::chrono

- “new” std library in C++ 11
- “a flexible collection of types that track time” [1]
- platform independent

durations	measure time spans
clocks	consists of a starting point (or epoch) and a tick rate
time points	duration of time passed since the epoch of a specific clock

[1] cppreference.com/w/cpp/chrono

std::chrono



durations — measure time spans

```
template <
    class Rep,
    class Period = std::ratio<1>
> class duration;
```

e.g.

```
using fortnights =
    std::chrono::duration<float, std::ratio<1209600> >;
```

usage

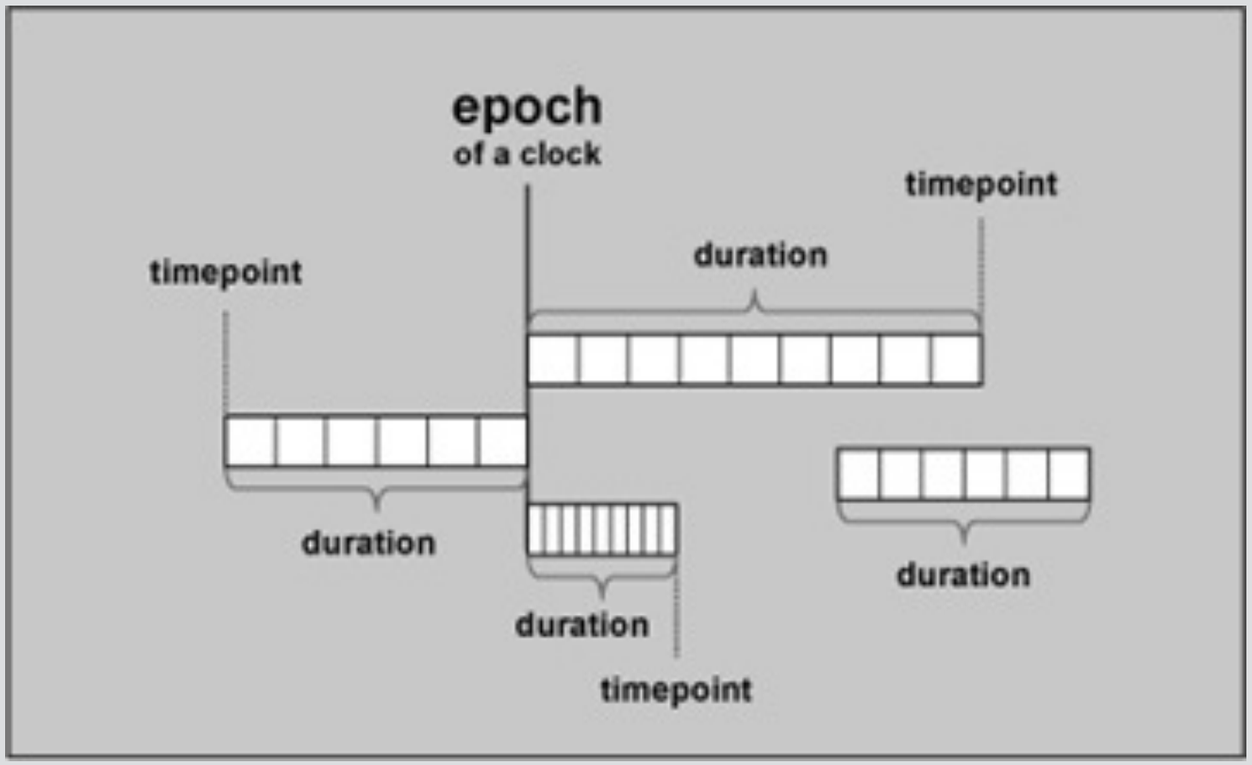
```
std::chrono::milliseconds tenMilliseconds(10);
```

clocks — consists of a starting point (or epoch) and a tick rate

time points — duration of time passed since the epoch of a specific clock

[1] cppreference.com/w/cpp/chrono

std::chrono



durations measure time spans

clocks consists of a starting point (or epoch) and a tick rate

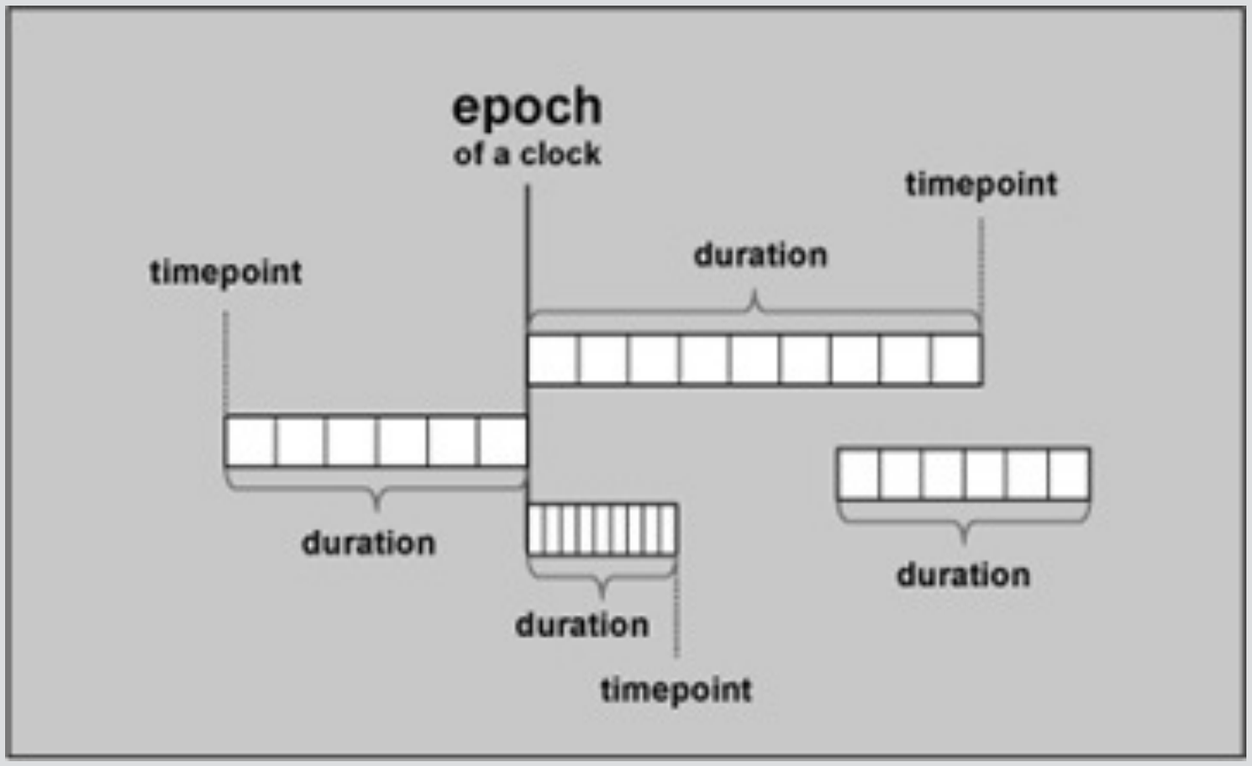
e.g. `class high_resolution_clock; // 100ns ticks`

uses
```
auto timePoint = chrono::high_resolution_clock::now();
auto aTimeStamp = timePoint.time_since_epoch().count();
```

time points duration of time passed since the epoch of a specific clock

[1] cppreference.com/w/cpp/chrono

std::chrono



durations measure time spans

clocks consists of a starting point (or epoch) and a tick rate

time points duration of time passed since the epoch of a specific clock

```
template <
    class Clock,
    class Duration = typename Clock::duration
> class time_point;
```


[1] cppreference.com/w/cpp/chrono

usecase: timestamp e.g. for runtime tests or in IGTBase

```
// check if MITK runs on a Windows-System
#ifdef _WIN32
  #include "mitkWindowsRealTimeClock.h"
#else // should be Linux or Mac OSX
  #include "mitkLinuxRealTimeClock.h"
#endif

mitk::RealTimeClock::Pointer mitk::RealTimeClock::New()
{
  mitk::RealTimeClock::Pointer smartPtr;

#ifdef _WIN32
  smartPtr = mitk::WindowsRealTimeClock::New();
#else
  smartPtr = mitk::LinuxRealTimeClock::New();
#endif

  return smartPtr;

double mitk::LinuxRealTimeClock::GetCurrentStamp()
{
  struct timeval tval;

  if ( ::gettimeofday( &tval, nullptr )!= 0 )
  {
    itkGenericOutputMacro("gettimeofday-method could not successfully acquire the current time");
    return -1;
  }
  double milliseconds;

  milliseconds = static_cast< double >( tval.tv_sec ) +
    static_cast< double >( tval.tv_usec ) / 1e6;

  return milliseconds*1000; // in milliseconds
}

double mitk::WindowsRealTimeClock::GetCurrentStamp()
{
// "if defined" not really necessary in this case, as the class is only available on Windows-systems
  __int64 time, ticks = 0;

  if (m_Frequency.QuadPart < 1)
  {
    return -1.0;
  }

  QueryPerformanceCounter( (LARGE_INTEGER*) &ticks);
  time = (ticks * 100000) / this->m_Frequency.QuadPart;
  double milliseconds = (double) (time & 0xffffffff);
  milliseconds /= (double)100.0;
  return milliseconds;
}
```

usecase: timestamp e.g. for runtime tests or in IGT

nice, portable and readable:

```
double mitk::RealTimeClock::GetCurrentStamp()
{
  return std::chrono::high_resolution_clock::
           now().time_since_epoch().count() / 10000;
}
```

usecase: sleeping 10 milliseconds

there are a few options:

```
Sleep(10);   //only works on Windows and has terrible accuracy
usleep(10);  //only works on Unix
sleep(1);    //only second precision
boost::this_thread::sleep(boost::posix_time::milliseconds(10));
             // needs boost but is portable
```

nice and readable:

```
#include <thread>           // std::this_thread::sleep_for
#include <chrono>           // std::chrono::milliseconds

std::this_thread::sleep_for(std::chrono::milliseconds(10));
```