Wiederholung

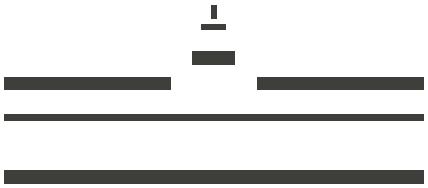
Interaktives C++ Programm

Programmstruktur

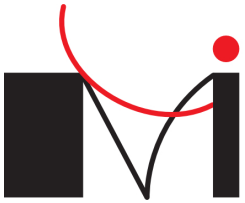
```
#include <fem_library>
int main(){
  // definiere:
  // WENN: User bedient Regler DANN: reagiere darauf
  // WENN: User bewegt Telefon DANN: reagiere darauf
  // WENN: User fasst Pendel DANN: reagiere darauf
  // WENN: 50ms vorbei DANN: mache Zeitschritt

  // warte auf WENNs, mache DANNs,
  // sonst warte
}
```

Gelernt

- Signale definieren
- Slots definieren
- Signale und Slots verbinden



Westfälische Wilhelms-Universität Münster



Applied Mathematics Münster

Interaktive Simulationen

Lektion 2/3: Threading

living.knowledge
WWU Münster

Prof. Dr. Christian Engwer
& Andreas Buhr

23. April 2014

Was ist ein Thread?

- Eine Sequenz von Instruktionen, die ausgeführt wird.
- Ein Ausführungskontext innerhalb eines Prozesses.

- Jeder Thread hat seine eigenen lokalen Variablen (Stack).
- Alle Threads eines Prozesses teilen sich einen Speicherbereich (Heap).

Wofür brauche ich Threads?

- Um mehrere Dinge gleichzeitig zu tun.
- Um Multicore-Rechner auszunutzen.

Beispiel: Interaktive Simulationen

- Ein Thread reagiert auf Usereingaben.
- Ein Thread macht die Arbeit.

- Livedemo

Thread erstellen

Erstellen eines QThread

```
#include <QThread>

int main(){
  QThread thread;
  thread.start();
  return 0;
}
```

Objekt in anderen Thread verschieben

z.B.:

```
simulationController.moveToThread(&thread)
```

Signale von Thread zu Thread

- Ausführung nicht mehr sofort, sondern später.
- Argumente werden immer kopiert.

Registrierung von Klassen zum Kopieren

```
class VisualizationData{
  // ...
};
class VisualizationController : public QObject
{
    Q_OBJECT
public slots:
    void visualize(VisualizationData x);
};
```

Funktioniert so nicht.

Registrierung von Klassen zum Kopieren

Lösung

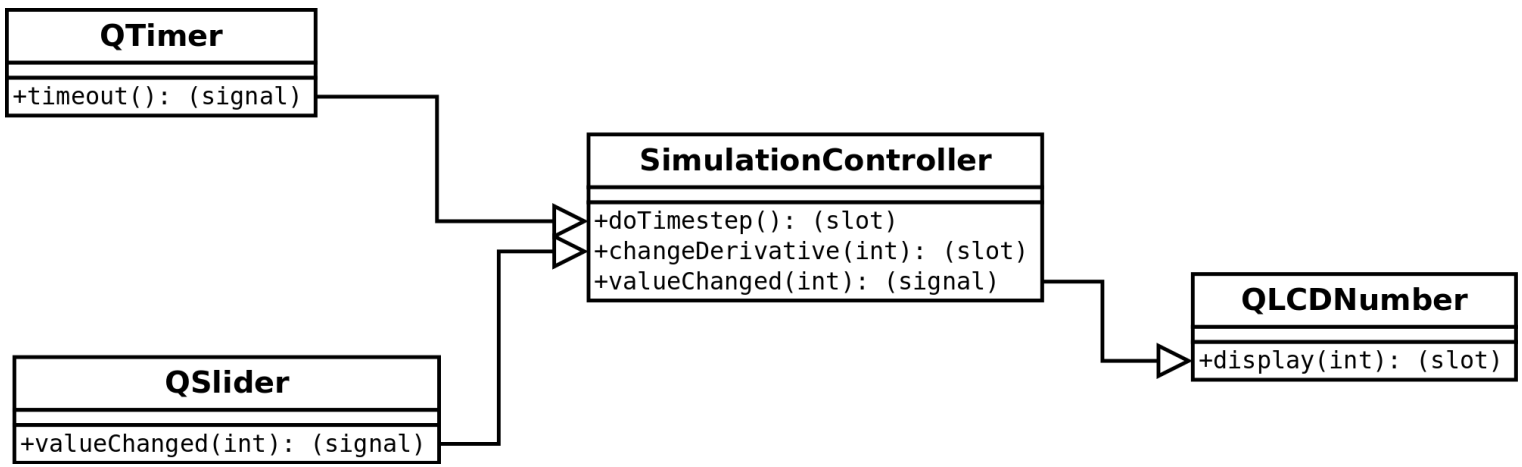
```
qRegisterMetaType<VisualizationData>(
                              "VisualizationData");
```

Gelernt

- Thread erstellen
- Objekt zu Thread verschieben
- Signale von Thread zu Thread schicken
- Klassen registrieren

Problem 1

Livedemo:



Problem 1

- Simulation dauert $\rightarrow$ Programm reagiert kaum
- SimulationController in anderen Thread $\rightarrow$ nichts passiert mehr.
- Problem: QTimer sendet zu viele timeout() Signale

Problem 1

- Simulation dauert $\rightarrow$ Programm reagiert kaum
- SimulationController in anderen Thread $\rightarrow$ nichts passiert mehr.
- Problem: QTimer sendet zu viele timeout() Signale
- Lösung: QTimer im gleichen Thread wie Signalempfänger halten

Problem 1

- Simulation dauert $\rightarrow$ Programm reagiert kaum
- SimulationController in anderen Thread $\rightarrow$ nichts passiert mehr.
- Problem: QTimer sendet zu viele timeout() Signale
- Lösung: QTimer im gleichen Thread wie Signalempfänger halten

Problem 2

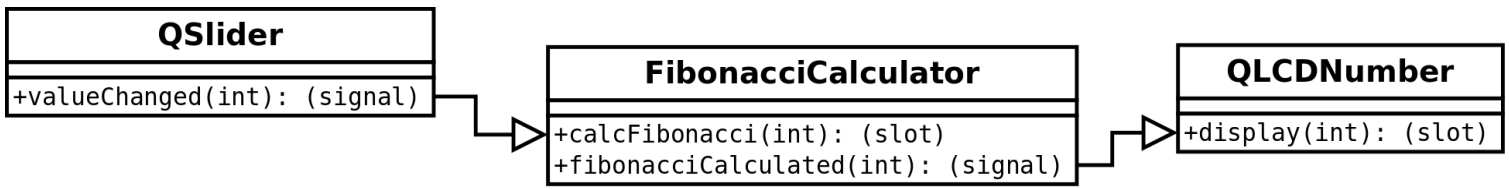
Problem 2

- UserInterface sendet zu viele Signale

Problem 2

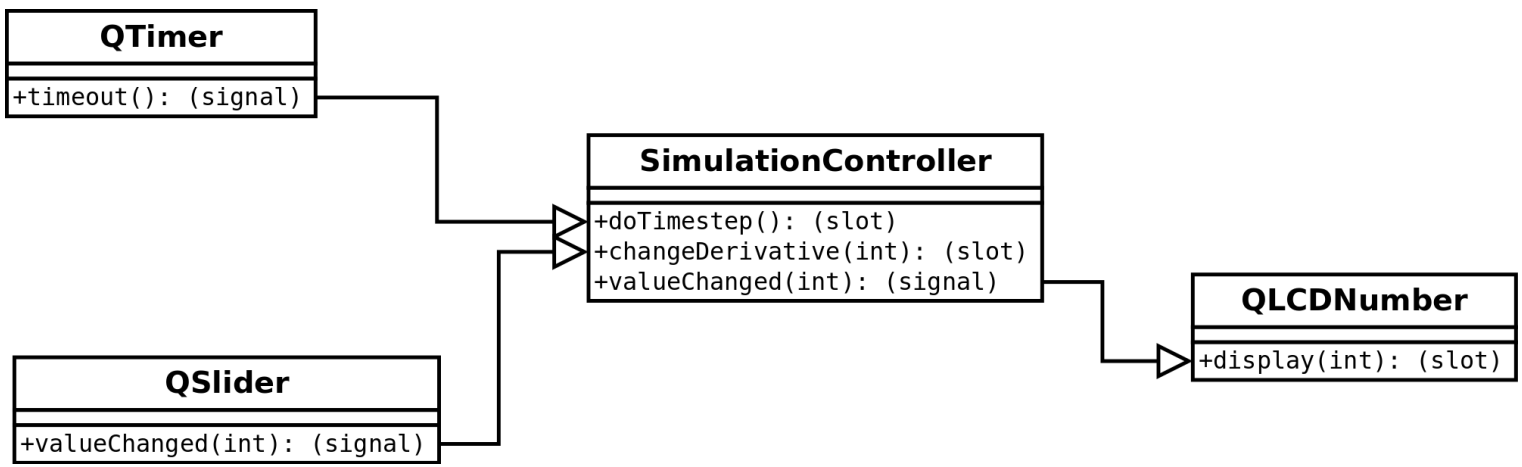
- UserInterface sendet zu viele Signale
- Lösung: Signale filtern

Signale Filtern



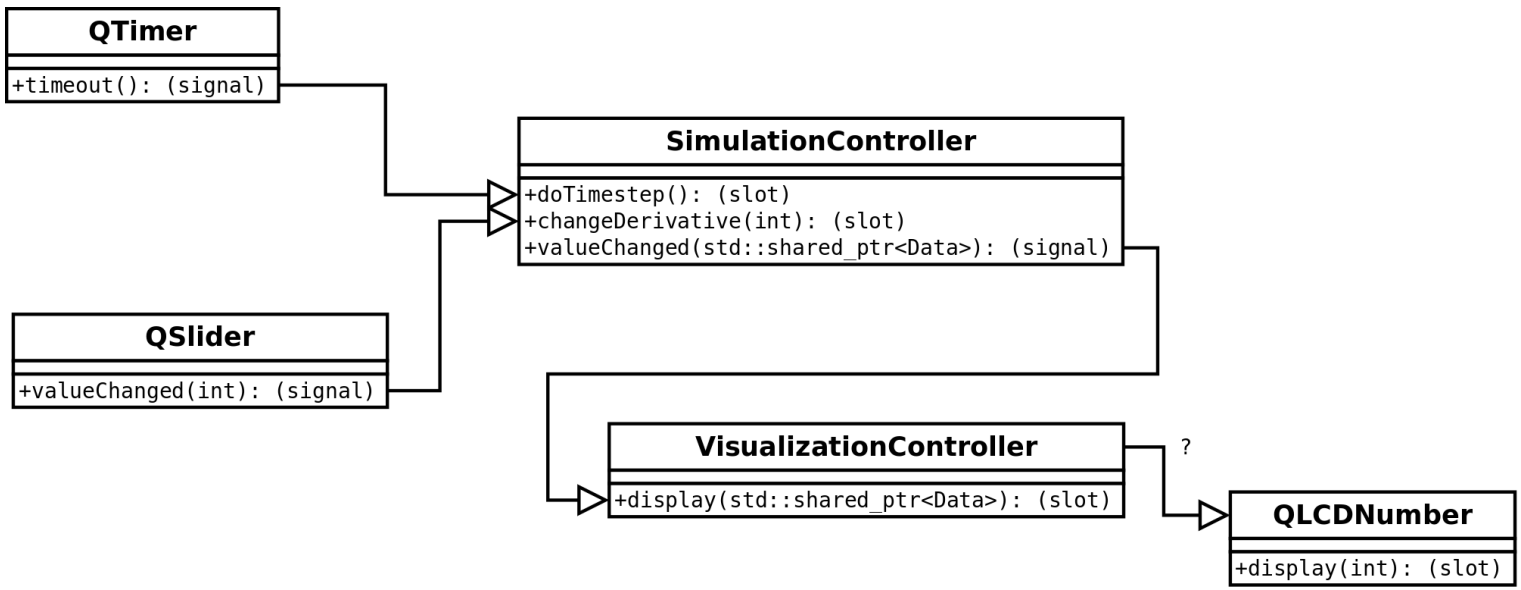

```
int fibonacci(int value){
  if(value <= 1) return value;
  return fibonacci(value - 1) + fibonacci(value - 2);
}
```

Zurück zum Programm von letzter Woche



Was ist, wenn die Daten größer werden?

Zurück zum Programm von letzter Woche



Pointer verschicken:

`std::shared_ptr<Data>`

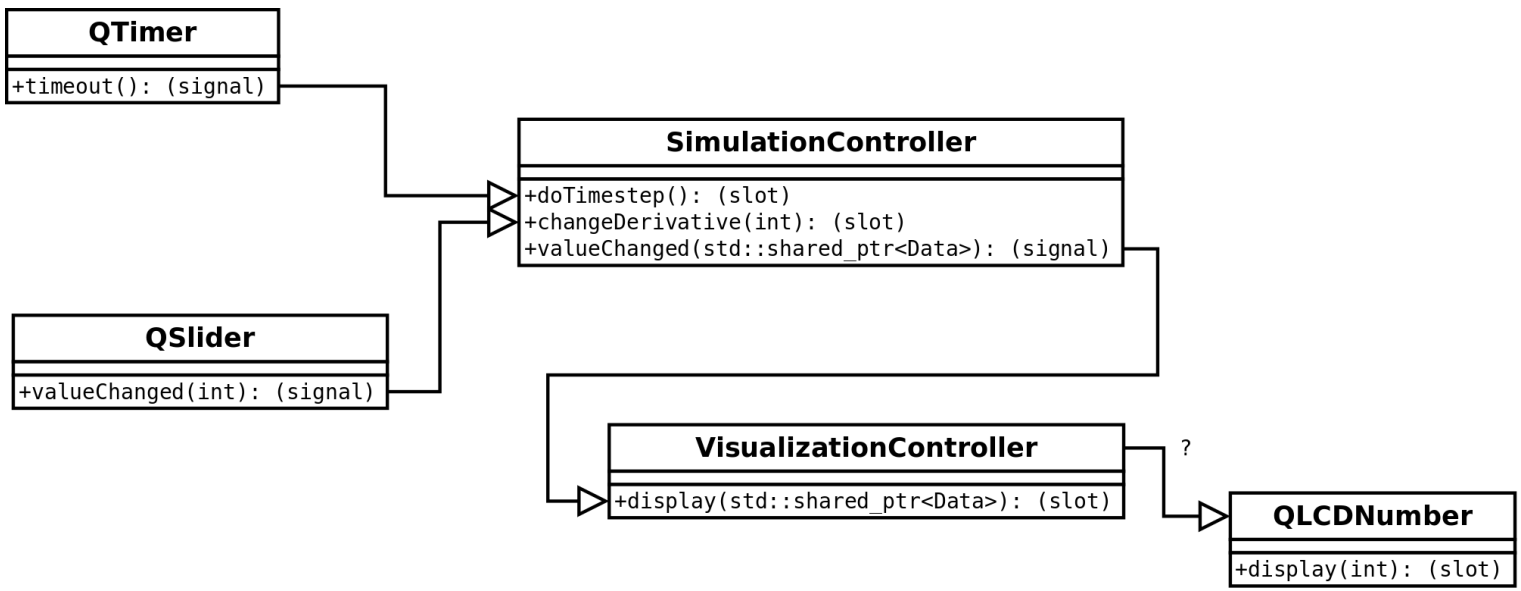
std::shared_ptr<Data> verschicken

- Pointer verschicken statt der Daten.
- Simulation muss in jedem Zeitschritt mit “new” neue Daten holen.
- Simulation darf Daten nach dem Verschicken nicht mehr schreiben.

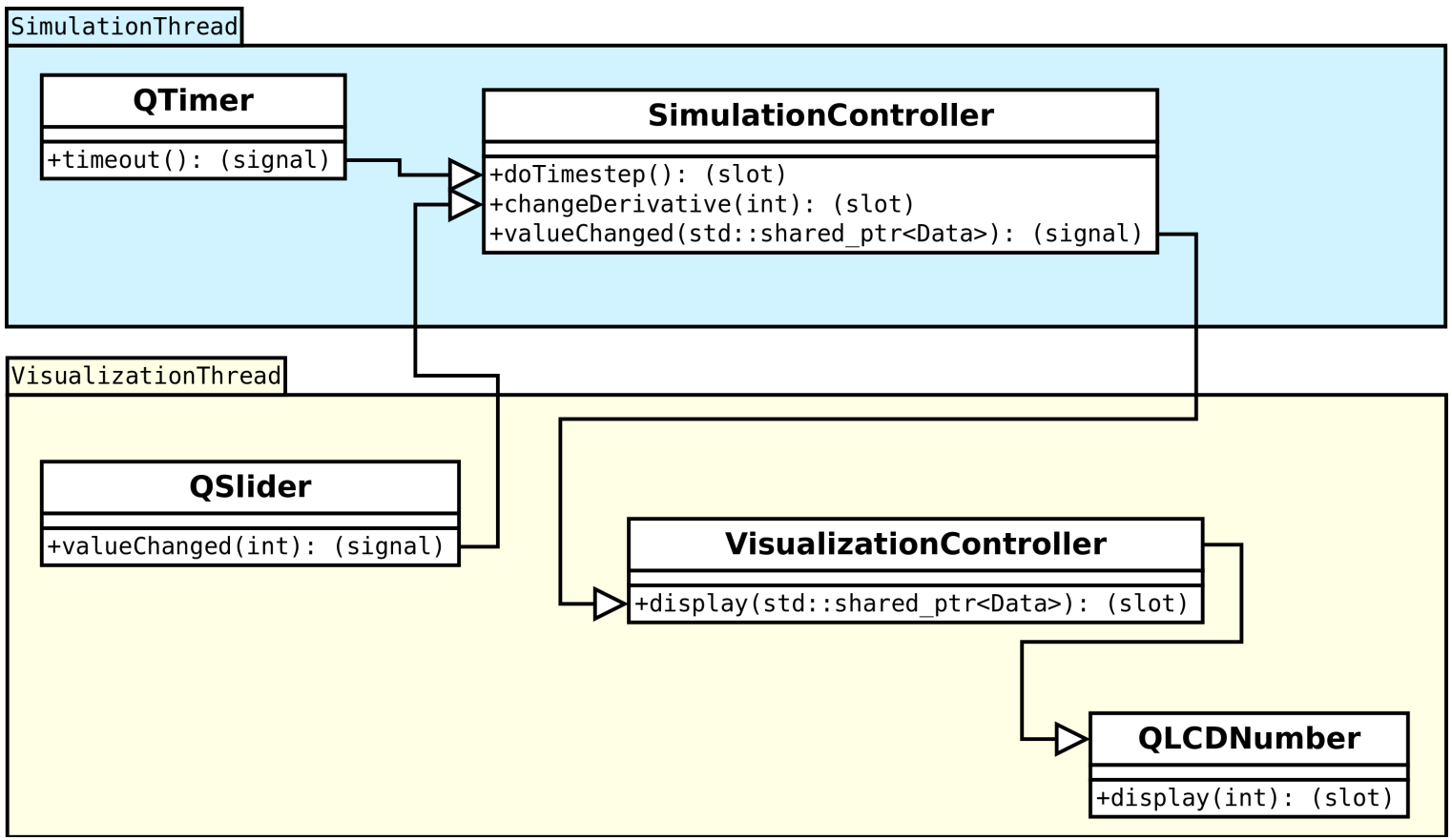
Zeitschleife

```
// olddata is member
void SimulationController::doTimestep(void){
  std::shared_ptr<Data> newdata(new Data());
  //calculate new data out of old data here
  olddata = newdata;
  emit valueChanged(olddata);
}
```

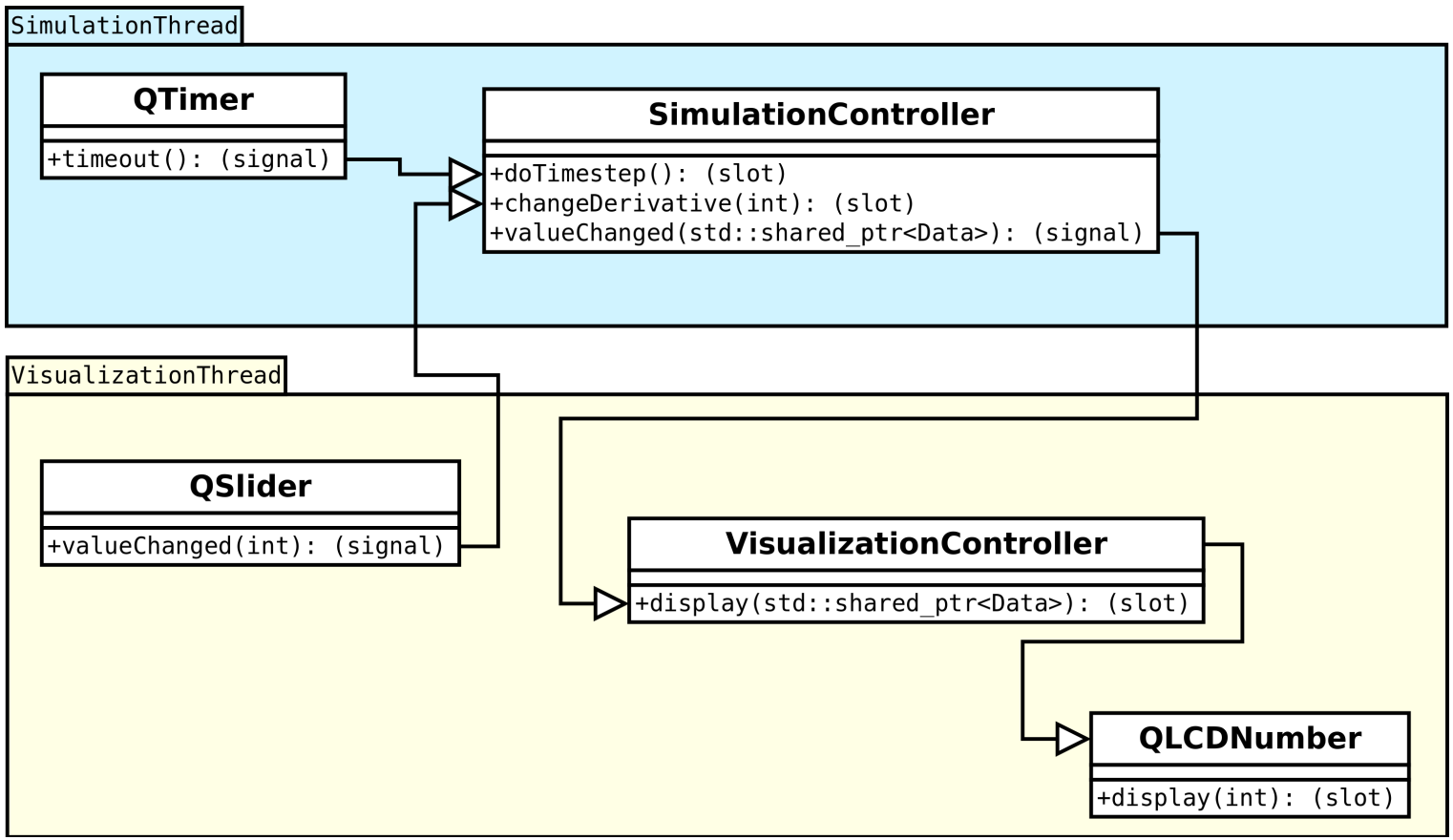
Was passiert, wenn Simulation lange dauert?



Was passiert, wenn Simulation lange dauert?

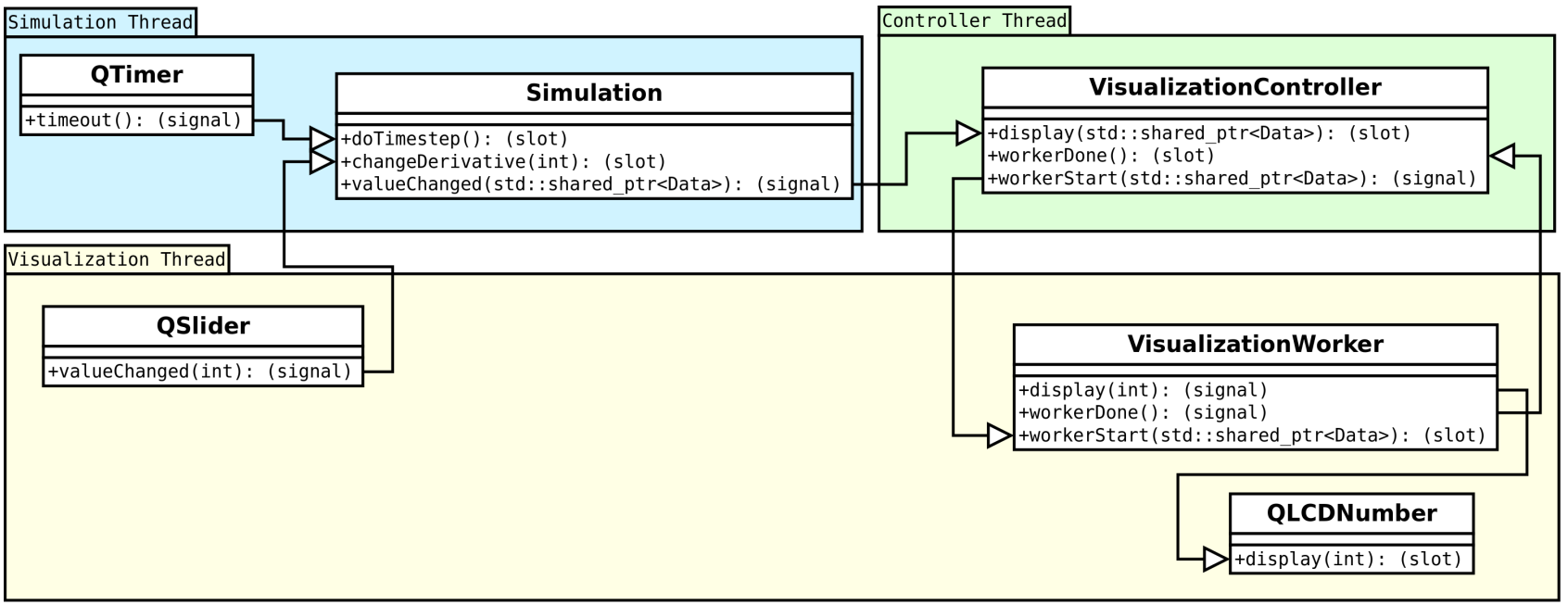


Was, wenn Simulation schneller als Visualisierung?



Dann wird die Visualisierung mit Signalen überschwemmt.

Was, wenn Simulation schneller als Visualisierung?



FAQ

- Wie veranlasse ich einen Thread, zu warten?

```
#include <chrono>
#include <thread>
std::this_thread::sleep_for(std::chrono::seconds(1));
```

In .pro:

```
CONFIG += C++11
```