

# Praxis des Programmierens

WS 2009

Do 8-10, Do 14-16, Do 16-18 im RTK  
(Vorlesung/Übung gemischt)

Dr. habil. Clemens Gröpl  
Prof. Dr. Stefan Funke

clemens.groepl AT uni-greifswald.de

stefan.funke AT uni-greifswald.de

Webseite zur Veranstaltung:

<http://www.alice-dsl.net/clemens.groepl/Praxis-des-Programmierens.html>

*[ Diese Version der Folien ist vom 28.01.2010 ]*

# Inhalte

- Aus der ***Modulbeschreibung***:

Praxis des Programmierens:

- Vertrautheit mit grundlegenden Konzepten des Softwaredesigns,
- Wissen um Grundprinzipien imperativer und objektorientierter Programmiersprachen, Planung und Umsetzung nicht-trivialer Softwareprojekte, Grundlagen des Softwaredesigns

# Inhalte

- Anstoß zum ***Programmieren lernen***
- ... und zwar nicht nur anhand kleinerer Programmfragmente, sondern anhand eines größeren Projektes
- Sie lernen C++, Linux und die GNU g++ Standardwerkzeuge kennen

# Ein Beispiel für ein C++ Programm

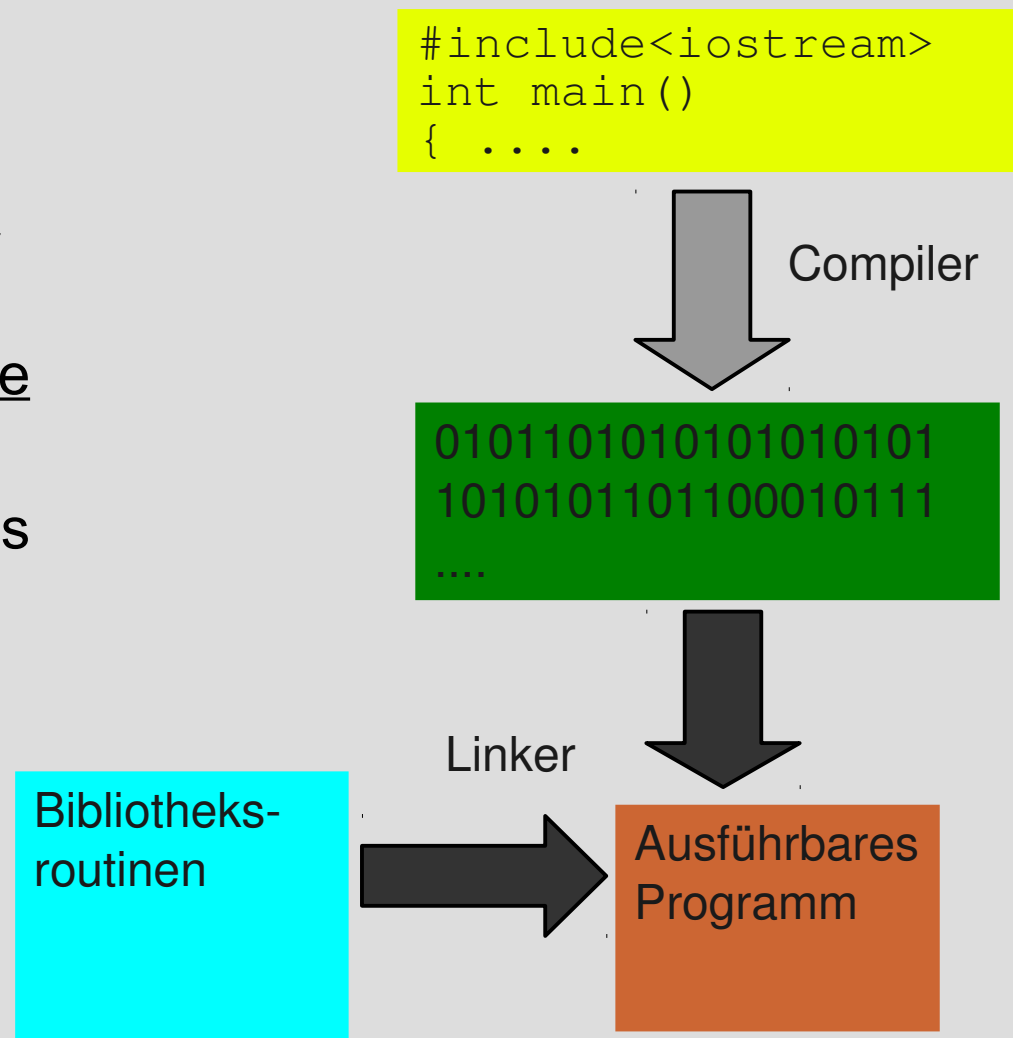
```
// einbinden von nützlichen Funktionen
#include <iostream>

// die Hauptfunktion
int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

**C++ = Sprache + Standardbibliothek**

# Was geschieht mit einem solchen Programm?

- Der Rechner kann dieses Sourcefile nicht direkt ausführen
- Es muß zuerst vom Compiler übersetzt werden, d.h. in ein maschinen-lesbares Objectfile gewandelt
- Der Linker verbindet dann das Objectfile mit den notwendigen Bibliotheksroutinen zum Executable



# Beispiel anhand der GNU g++ Toolchain

Aufruf des Compilers:

```
g++ hello.cc -c
```

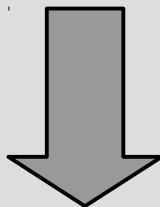
erzeugt Objectfile `hello.o`

Starten des Linkers mit

```
g++ hello.o -o hello
```

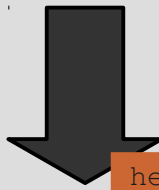
Dies bindet notwendige Bibliotheks-routinen hinzu und erzeugt ausführbares Programm `hello`

```
hello.cc  
#include<iostream>  
int main()  
{ ....
```



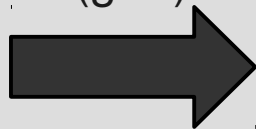
Compiler (g++)

```
hello.o  
0101101010101010101  
1010101101100010111  
....
```



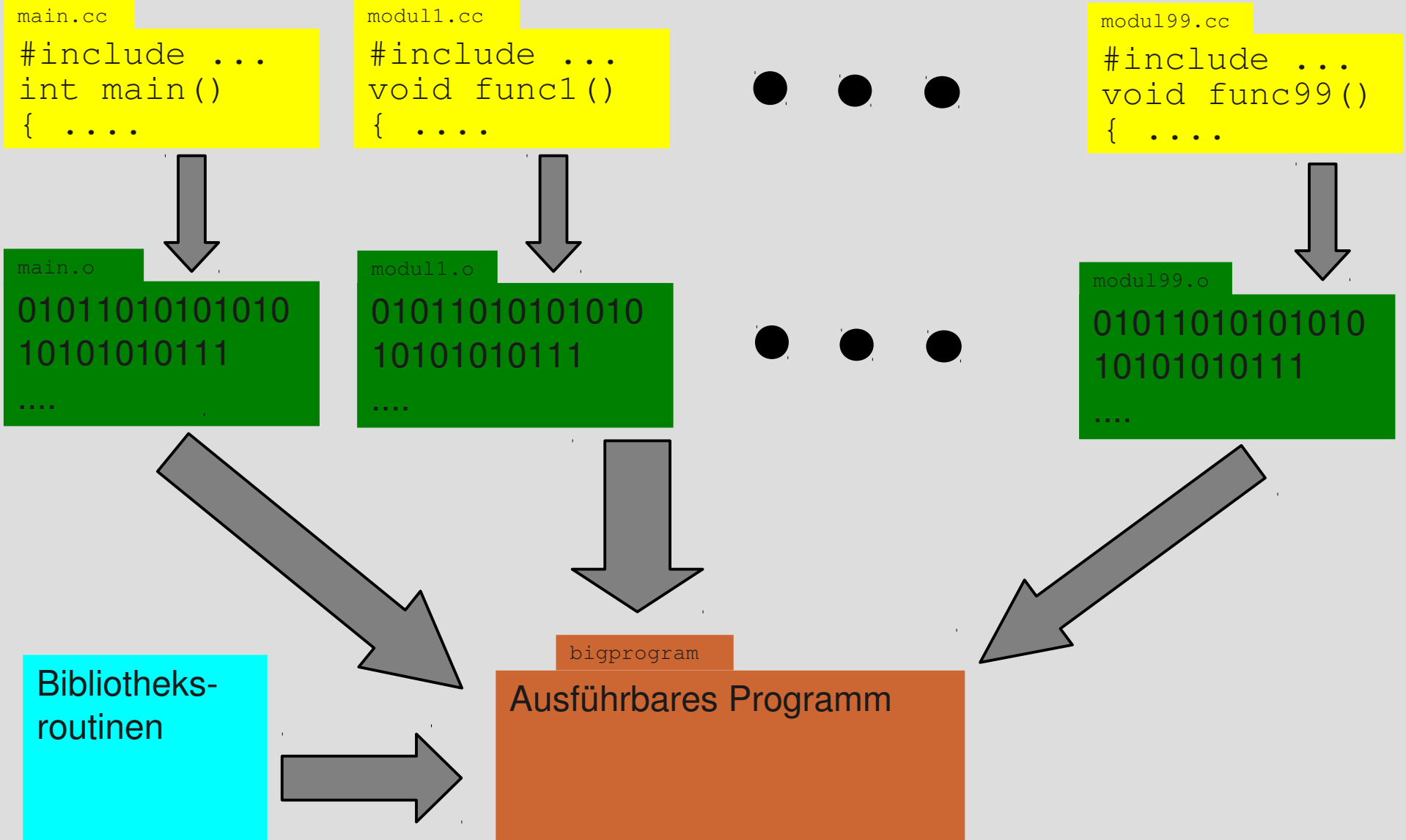
Linker (g++)

Bibliotheks-routinen



hello  
Ausführbares Programm

# Komplexe Projekte bestehen meist aus vielen Sourcefiles



# Komplexe Projekte bestehen meist aus vielen Sourcefiles

main.cc

```
#include ...  
int main()  
{
```

modul1.cc

```
#include ...  
void func1()  
{
```



modul99.cc

```
#include ...  
void func99()  
{
```

Auch dafür gibt es eine Lösung:

## make

- behält den Überblick über alle Abhängigkeiten
- führt nur die notwendigen Schritte erneut durch

Bibliotheks-  
routinen

bigprogram

Ausführbares Programm



# Erfolgreich größere Programmierprojekte durchführen

- ... hat sehr viel mit guter Organisation zu tun
- Wie kann ein großes Projekt in kleine Module zerlegt werden?
- Genau definieren, wie Module interagieren
- Ausführlich kommentieren!
- ... den Überblick bewahren!

# Unsere Arbeitsumgebung: LINUX

- Freies UNIX Betriebssystem
- Enthält vollständige Entwicklungstoolchain (GNU g++), um C/C++ Programme zu entwickeln
- Kommandozeilenorientiert (es gibt auch IDEs = Integrated Development Environment, z.B. Eclipse); benutzen wir aber zunächst nicht
- Multiuser-Betriebssystem: mehrere Benutzer können gleichzeitig auf einem Rechner arbeiten (Remote-Login)

# Start von Linux auf Ihrem lokalen Rechner

- Rechner einschalten :-)
- Im Bootmanager „Ubuntu“ auswählen
- Einloggen, Beispiel:  
Username (z.B.) : esprimo13  
Passwort : *< leer lassen >*
- Shell starten  
(Anwendungen → Zubehör → Terminal)

# Kommandos auf der Shell

[Webseite](#) → [FAQ](#) → [Beliebte Shell Kommandos](#)

ls	Verzeichnisinhalt ausgeben (ls -l ausführlich)
mkdir	Verzeichnis erstellen
cd	in Verzeichnis wechseln (ohne Argument: in Home-Verz. Wechseln)
rmdir	leeres Verzeichnis löschen
mv	Dateien/Verzeichnisse verschieben / umbenennen
man	Hilfe zu Kommando aufrufen
rm	Dateien löschen (VORSICHT!)

# Kommandos auf der Shell

[Webseite](#) → [FAQ](#) → [Beliebte Shell Kommandos](#)

- prog &      Programm im Hintergrund ausführen
- Ctrl-z      Programm anhalten
- bg          angehaltenes Programm in den Hintergrund schicken
- Ctrl-c      Programm abbrechen
- fg          (letztes) Programm wieder in den Vordergrund holen
- jobs        Listet Programme, die im Hintergrund laufen, auf

Eingabe und Ausgabe umleiten:

```
./my_program < input.txt > output.txt
```

# Aufgabe

- Sie haben auf den Poolrechnern keinen “eigenen” Nutzernamen und Account!
- Erstellen Sie folgende Verzeichnisstruktur  
NameVorname/  
    Trockenuebungen/  
    MeinProjekt/
- Erzeugen Sie in NameVorname/ eine Datei `Email.txt`, welche Ihre Email Adresse enthält
- Erzeugen Sie in NameVorname/Trockenuebung/ eine Datei `Changelog.txt`, in welcher Sie dokumentieren, was sie gerade getan haben, z.B. so:  
2009/10/15: Verzeichnisstruktur erstellt

# Ein minimales C++ Programm

```
// einbinden von nützlichen Funktionen
#include <iostream>

// die Hauptfunktion
int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

*Webseite → Directory → Hello World*

# Speichern/Übersetzen des Programms

- Tippen Sie das Programm in einem Editor (z.B. kate) ein und speichern Sie es unter hello.cc
- gehen Sie auf die Shell und kompilieren es:  
`g++ hello.cc -c`
- schauen Sie nach, was der Compiler produziert hat mit:  
`ls`
- Linken Sie das Programm mit  
`g++ hello.o -o hello`
- Programm starten mit  
`./hello`

# Variablen und Konstanten in C/C++

- C/C++ unterscheidet zwischen verschiedenen Arten der Repräsentation von Daten – Datentypen:
  - `int` ganze Zahlen, z.B. 7
  - `float` Gleitkommazahlen, z.B. 4.6
  - `char` Zeichen, z.B. 'a'
  - `void` Spezieller (Nicht-)Typ
  - ...

# Ein-/Ausgabe von Daten

- `cout` gibt Zeichenketten/Variablenwerte aus  
`cout<<"Variable x hat Wert  
<<x<<endl;`
- `endl` erzeugt einen Zeilenvorschub
- `cin` liest Werte ein:  
`int x;  
cout<<"Bitte Wert eingeben: ";  
cin>>x;`

# Aufgabe

- Schreiben Sie ein Programm, welches zwei Zahlen einliest, diese addiert und die Summe ausgibt.

*Lösung: Webseite → Directory → Adder*

# for-Schleife

- **Syntax:**  
for (StartBefehl; Bedingung; SchleifenUpdate)
- **Beispiel:**

```
for (int i=0; i<20; i++)  
{  
    ... code ...  
}
```
- obige Schleife wird 20 Mal durchlaufen, i hat dabei die Werte  $i = 0, 1, 2, \dots, 19$

# while-Loop

- **Syntax:**

```
while (Bedingung)
    { ..<body>... }
```

- führt den Code in <body> aus, solange die Schleifenbedingung erfüllt ist

- kann ersetzt werden durch

```
for( ;Bedingung; )
    { ... <body> ... }
```

- umgekehrt können Sie eine for-Schleife auch durch einen while-Loop ersetzen (mit entsprechenden Variablen)

# while-Loop (nicht ganz überflüssig)

- Alternative Syntax:

```
do  
{  
    ... <body> ...  
}  
while (Bedingung);
```

- führt den Schleifenbody einmal aus (auf jeden Fall) und wiederholt, solange die Bedingung erfüllt ist

# Aufgaben

- Schreiben Sie ein Programm,
  - das die Zahlen von 1 bis 20 aufsteigend ausgibt
  - das die Zahlen von 20 bis 1 absteigend ausgibt
  - das die Summe der ersten n Zahlen berechnet (n ist Eingabe)
  - das n! Berechnet
- Schreiben Sie ein Programm, das die Kommandozeilenargumente ausgibt.

*Lösung: Webseite → Directory → main argc argv*

# Arrays, `std::vector`

- Mit `int A[20];`  
legen Sie ein Array aus 20 Ganzzahlen an,  
die Sie mit `A[0]`, `A[1]`, ... `A[19]` ansprechen  
können. Die Größe kann nicht mehr  
verändert werden! → C-style arrays
- Mit `std::vector<int> A(20);`  
tun Sie das entsprechende in C++.
- Recherchieren Sie die Vorteile von  
`std::vector` gegenüber C-style Arrays.  
→ STL (standard template library)

# Die Standard C++ Library (STL)

- es gibt eine Standardbibliothek für das Programmieren in C++, die „Standard Template Library“ (STL)
- sie stellt eine Vielzahl nützlicher Funktionen bereit, z.B. Routinen zur Sortierung, Dateiein- und ausgabe, etc.

*Webseite → C++ → Webseiten*

# Aufgabe

- Schlagen Sie im STL Manual die Klasse `vector` nach; inwieweit unterscheidet diese sich von einem Array/Vector, wie Sie ihn bislang genutzt haben?
- ... wahrscheinlich werden Sie etwas erschlagen sein von der Fülle der Informationen

# Die `vector` Klasse für Dummies

- Sie können einen `vector` prinzipiell fast genauso wie ein Array behandeln, d.h. `A[0]` greift auf das erste Element, `A[1]` auf ....
- wichtige Unterschiede:
  - Größe eines `vector` ist dynamisch (0 anfangs)!
  - einem `vector v` kann ein Element `e` mit `v.push_back(e)` hinzugefügt werden
  - die Größe kann auch explizit durch `v.resize()` angegeben werden

# Die `vector` Klasse für Dummies

- Sie können einen `vector` von (fast) jeder Klasse/Datentyp erstellen

- Beispiel:

```
#include<vector>
```

```
...
```

```
vector<int> meineZahlen;
```

```
meineZahlen.push_back(7);
```

```
meineZahlen.push_back(8);
```

```
std::cout<<meineZahlen[0]<<"
```

```
"<<meineZahlen[1]<<std::endl;
```

# Komplexe Datentypen

- Oft ist es wünschenswert, mehrere Daten (z.B. Immatrikulationsnummer, Alter, Punktzahl) zu einem komplexeren Objekt zusammenzufassen und dieses wie einen „normalen“ Datentyp behandeln zu können (also wie `int`, `float`, `double`, ...)
- Dazu gibt es in C++ (ansatzweise schon in C) ein Sprachmittel, die sogenannten **Klassen**

# C++ Klassen

```
class ExamGrade
{
    public:
        int matrno;
        int grade;
};
```

- C++ Klassen müssen in einem Header-File deklariert werden und können in einem .C File wie folgt instanziiert werden:

```
ExamGrade pruefung;
```

- Auf die Elemente einer Klasse kann man mit dem operator `.` zugreifen:

```
pruefung.matrno=1234567;
```

- `public:` bedeutet, dass auf die folgenden Elemente von außen zugegriffen werden kann

# C++ Klassen

- Außer „Daten“ können Sie auch Funktionen (oder hier „Methoden“ genannt) in eine C++ Klasse packen, welche jedoch nur auf den Daten der Klasse operieren darf:

```
class ExamGrade{  
    public:  
        int matrno;  
        int grade;  
        void printMe();  
};
```

- **Mögliche Anwendung:**

```
ExamGrade pruefung;  
pruefung.matrno=10;  
pruefung.grade=14;  
pruefung.printMe();
```

# C++ Klassen

- Die Methode `printMe()` müssen Sie jedoch noch entsprechend bereitstellen:

```
void ExamGrade::printMe()
{
    std::cout<<"Matr.nr. " <<matrno<<" hat
        Note " <<grade<<std::endl;
}
```

- Die Klassendefinition (`class ExamGrade ...`) gehört typischerweise in ein `.h` File, die obige Implementierung der Methode in das entsprechende `.C` File, ähnlich der Aufteilung in "Ankündigung" und Implementierung einer Funktion in einem Modul

# Aufgaben

- Modifizieren Sie Ihr letztes Programm, sodass es die Klasse `vector` benutzt anstatt einfacher Arrays
- Benutzen Sie `random_shuffle()`, um die Kommandozeilenargumente in zufälliger Reihenfolge zurückzugeben.

Webseite → Directory → `main argc argv` → `random_shuffle.C`

# Declaration, definition, default ctor, copy ctor, assignment

- Declaration: “Einen Namen erklären.”
- Definition: “Eine Instanz erstellen.”
- Default constructor: nimmt keine Argumente
- Copy constructor: erstellt Kopie
- Zuweisung (assignment): operator =

*Lesen Sie die Kommentare im Programmbeispiel:*

*Webseite → Directory → Declare Define Assign Construct*

# Funktionen

- **Syntax:**

```
returntyp funktionsname (argtyp argname, ...)  
{  
    // „normaler Programmcode“  
}
```

- als `returntyp` bzw. `argtyp` können Sie alle elementaren C/C++ Datentypen verwenden, für `returntyp` auch `void` (dann hat die Funktion keinen Rückgabewert)

- Funktionen müssen **vor** ihrer Verwendung definiert werden

- bei zyklischen Abhängigkeiten können Sie Funktionen „vorab“ ankündigen: **Deklaration**

```
returntyp funktionsname (argtyp, argtyp, ...);  
(Strichpunkt beachten!)
```

# Funktionen: ErgebnISRückgabe

## Call-by-Value:

```
int quadrat1(int x)
{
    return x*x;
}
```

## Call-by-Reference:

```
void quadrat2(int &x)
{
    x = x*x;
}
```

Im ersten Fall würden Sie wie folgt das Quadrat von X berechnen und ausgeben:

```
int y=quadrat(x);
cout<<"Das Quadrat von " <<x<<" ist "
<<y<<endl;
```

Im zweiten Fall würde das Ergebnis direkt in die Variable x geschrieben, d.h. falls der Originalwert von x noch benötigt wird, muss er vorher gesichert werden:

```
int x_alt=x;
quadrat2(x);
cout<<"Das Quadrat von " <<x_alt<<"
ist " <<x<<endl;
```

Im Allgemeinen ist die Rückgabe eines Ergebnisses "by value" weniger fehleranfällig. Andererseits entfällt bei "by reference" der Aufruf des copy constructors für den Rückgabewert.

# Aufgabe

- Schreiben Sie ein Programm, welches zwei (positive ganze) Zahlen  $x$ ,  $y$  einliest und „ $x$  hoch  $y$ “ berechnet
- Die Potenz sollte dabei in einer Funktion berechnet werden und einmal durch einen Rückgabewert, und einmal per Call-by-Reference zurückgegeben werden

Webseite → Directory → pow

# Weiteres zu Arrays

- Übergeben eines Arrays an eine Funktion:

```
void fun(int A[], int n)
{ // Code, der auf A[0], ..., A[n-1]
  // operiert
}
```

- Aufruf:

```
{
  int A[20];
  fun(A, 20);
}
```

- Übergabe hier immer als Call-by-Reference!
- Größe des Vektors (hier: 20) muss immer mit übergeben werden

# Modularisierung (1)

- Oft wollen Sie eine geschriebene Routine auch in anderen Programmen wiederverwenden und nicht immer in das entsprechende Sourcefile einkopieren
- In diesem Fall sollten Sie die Routine in einer separaten Datei speichern (z.B. `myfun.c`), welches Sie separat übersetzen können
- Des Weiteren legen Sie ein Header-file (`myfun.h`) an, sodass andere Module, die Ihre Sortierroutine benutzen möchten, das Interface im entsprechenden Header-File (`myfun.h`) mittels `#include` einbinden können

# Modularisierung (2)

main.cc

```
.....  
#include "myfun.h"  
int main()  
{  
    int A[100];  
    // Zahlen einlesen  
    ..  
    mysort(A, 100);  
    // Ausgabe  
    ..  
}
```

myfun.h

```
void mysort( ... );
```

myfun.cc

```
.....  
#include "myfun.h"  
  
void mysort(...)  
{  
    // Implementierung  
    // der Funktion  
}
```

Übersetzung mit:

```
g++ -c main.cc
```

```
g++ -c myfun.cc
```

```
g++ main.o myfun.o -o main
```

# Modularisierung (3)

- je mehr Module sie haben, desto umständlicher wird das übersetzen
- insbesondere die Übersicht darüber zu behalten, welche Änderungen welche Neuübersetzungen bedürfen, ist schwierig
- wenn Sie z.B. am Interface von `mysort()` ändern, müssen sowohl `myfun.C` als auch `main.C` neu übersetzt werden
- wenn Sie nur die Implementierung/Umsetzung von `mysort()` intern ändern, reicht es, `myfun.cc` neu zu übersetzen
- mit einem `Makefile` können Sie über solche Abhängigkeiten den Überblick bewahren und nur die wirklich notwendigen Module neu übersetzen

# Makefiles

## Syntax:

```
<target>:<list of dependencies>  
<tab><build-command>
```

Ein „Target“ wird nur neu erstellt, wenn es älter (Datum/Zeit) als die „Dependencies“ ist.

Es wird per default immer versucht, das erste Target zu „bauen“: `make`

Alternativ kann das Target angegeben werden:

```
make main.o
```

Abhängigkeiten werden rekursiv verfolgt, d.h. falls sich `myfun.h` geändert hat, wird bei Eingabe von `make` alles neu übersetzt+ gelinkt. Wenn sich nur `myfun.cc` geändert hat, wird nur das übersetzt+gelinkt.

Man speichert ein Makefile typischerweise unter dem Namen `Makefile`.

## Makefile

```
main: main.o myfun.o  
    g++ main.o myfun.o -o main  
  
main.o: myfun.h main.cc  
    g++ -c main.cc  
  
myfun.o: myfun.cc myfun.h  
    g++ -c myfun.cc
```

# Aufgabe

- Zerlegen Sie Ihr “Hello, World!”-Programm in zwei Module:
  - eines welches die eigentliche Routine enthält (mit Header-File):  
`say_hello.C`      `say_hello.h`
  - die „Hauptroutine“ (enthält die Funktion `main()`):  
`main.C`
- Erstellen Sie das entsprechende Makefile dazu
- Übersetzen und testen Sie Ihr Programm

[Webseite](#) → [Directory](#) → [Hello World refactored](#)

# Einschub: Die Klasse string

- Sie können aus der Standardbibliothek für C++ einen speziellen Datentyp `string` benutzen, welcher Zeichenketten repräsentiert und einfacher als Vektoren vom Typ `char` oder C-style `char[]` zu handhaben ist
- Sie müssen dazu das entsprechende Include-File einbinden:  
`#include<string>`
- und können damit Variablen vom Typ `string` instantiieren, einlesen und auslesen:  

```
std::string meinName;  
std::cin>>meinName;  
std::cout<<"Ich heie " <<meinName<<std::endl;
```
- Wichtig: ein mittels `cin` eingelesener String kann keine Leerzeichen enthalten
- Wie schon bei `cin`, `cout` und `endl`, knnen Sie sich das `std::` sparen, wenn Sie zuvor `using namespace std;`

# Dateiein-/ausgabe

- Es ist auch möglich, Daten auf Festplatte zu schreiben bzw. zu lesen
- `#include<fstream>`  
bindet den notwendigen Header ein
- Datei "myFile.txt" zum Lesen öffnen und ersten String lesen:  

```
ifstream meineDatei("myFile.txt");  
meineDatei>>myString;  
meineDatei.close();
```
- Datei "myFile.txt" zum Schreiben öffnen:  

```
ofstream meineDatei("myFile.txt");  
meineDatei<<"Hallo"<<endl;  
meineDatei.close();
```

# Nützliche Tips (1)

- Schreiben Sie sobald wie möglich ein Makefile (!)
- gewöhnen Sie sich ein konsistenten Stil in Sachen Einrückungen an (z.B. 2x <Space> pro Programmblock)

```
if (hallo)
{
    for(int i=0; i<n; i++)
        counter++;
}
```

# Nützliche Tips (2)

- Benutzen Sie wenn möglich aussagekräftige und sinnvolle Variablennamen (Ausnahme evtl. Schleifenvariablen)
- Legen Sie fest, ob Sie Variablen oder Typen groß oder klein schreiben, z.B. alle Typ- bzw. Klassennamen mit Großbuchstaben beginnen, alle Variablen mit Kleinbuchstaben
- Große Projekte haben in der Regel einen “coding style”, “coding guidelines” o. ä.

# Aufgabe

- Unter [Webseite](#) → [Directory](#) → [fasta splitter data](#) finden Sie eine Datei `drei.fasta` im FastA-Format. Jede Proteinsequenz wird von einer Kommentarzeile eingeleitet, z.B.:  
>B. subtilis 168|[BG12556](#)|AapA: amino acid permease
- Schreiben Sie ein Programm, das die Eingabe in einzelne FastA-Files aufteilt.
- Im Beispiel sollte die Ausgabe in die Datei [BG12556.fasta](#) geschrieben werden.

# Aufgabe

- Verwenden Sie `#include <sstream>` und `std::stringstream`, um die Zeichenketten zu bearbeiten.
- *Einfacher:* Sie können auch zunächst einfach fortlaufende Nummern o. ä. als Dateinamen verwenden
- *Schwieriger:* Implementieren Sie eine Suchfunktion: Gegeben ein FastA-File und eine Zeichenkette, gib die Kommentare von Sequenzen aus, die diese Zeichenkette enthalten.

# Beispielsitzung

```
09_fasta_splitter_data $ ls  
drei.fasta  fasta_splitter.C  
09_fasta_splitter_data $ g++ fasta_splitter.C  
09_fasta_splitter_data $ ./a.out  
usage: ./a.out fasta_input_file  
09_fasta_splitter_data $ ./a.out drei.fasta  
BG11037.fasta  
BG12556.fasta  
BG11900.fasta  
09_fasta_splitter_data $ ls  
BG11037.fasta  BG11900.fasta  BG12556.fasta  
a.out*  drei.fasta  fasta_splitter.C
```

# Aufgabe

- Schreiben Sie eine Funktion

```
void parseCmdline (  
    int argc,  
    char **argv,  
    std::vector<std::string> &args ) ;
```

mit der man die Kommandozeilenargumente in eine C++/STL-gemäße Form umkopieren kann.

- In Zukunft wollen wir dann immer diese Form verwenden.  
→ `parseCmdline.h` + `parseCmdline.C`

# Aufgabe

- Schreiben Sie Funktionen

```
void readNumbers (  
    const std::string & filename,  
    std::vector<double> &values );
```

```
void writeNumbers (  
    const std::string & filename,  
    const std::vector<double> &values );
```

welche eine Datei, die nur aus Zahlen besteht,  
in einen vector einliest bzw. wieder herausschreibt.

- Auch hier (am besten) wieder `.h` + `.C`

# Aufgabe

Schreiben Sie ein C++-Programm, das

- eine Datei von Zahlen einliest,
- die Ausreißer entfernt  
(per definitionem sind das hier Zahlen, die mehr als eine Standardabweichung vom Mittelwert entfernt sind),
- die bereinigte Liste speichert.

```
void removeOutliers (  
    const std::vector<double> & data,  
    std::vector<double> & result );
```

$$\sigma = \sqrt{\frac{1}{N-1} \sum_i^N (x_i - \bar{x})^2} \quad \text{mit} \quad N = |\{x_i\}| \quad \text{und} \quad \bar{x} = \frac{1}{N} \sum_i x_i$$

# Aufgabe

Schreiben Sie ein C++-Programm, das

- Zufällig 1000 annähernd standardnormalverteilte Zahlen würfelt *oder mit readNumbers einliest* und
- ein **Histogramm** erstellt.

Anforderungen an das Histogramm:

- Frei wählbare Breite der “Bins”.
- Keine Annahmen über min/max machen!
- Die Datenpunkte werden einzeln eingefügt, das Histogramm wird sofort aktualisiert.
- Tip: **std::map**
- Ausgabe des Histogramms mit *writeNumbers*.

# Lernzielkontrollumfrage

- 1 Ich kenne den Unterschied zwischen Compiler und Linker
- 2 Ich kann ein einfaches Makefile selbst schreiben.
- 3 Ich kann mit der Kommandozeile (shell) umgehen
- 4 Ich kann die an main() übergebenen Kommandozeilenargumente in meinem Programm als Zahlen und als Zeichenketten verwenden.
- 5 Ich kann Zahlen und Zeichenketten von cin lesen und nach cout schreiben, dazu benutze ich die Operatoren << und >>
- 6 Ich kann Zahlen und Zeichenketten in Dateien lesen und schreiben
- 7 Ich kann den Status eines Streams abfragen, z.B. um festzustellen, ob noch weitere Eingaben folgen
- 8 Ich kenne mich mit if, while, for, do while aus
- 9 Ich weiß: c-style arrays sind böse, aber std::vector sind gut, und ich verstehe den Unterschied
- 10 Ich weiß: c-style char[] Zeichenketten sind böse, aber std::string sind gut, und ich verstehe den Unterschied
- 11 Ich weiß so ungefähr, was ein Pointer ist, und erkenne auch die entsprechende Syntax, lasse aber selber lieber die Finger davon
- 12 Ich kann mir auf den Webseiten selbstständig Fragen zur Standardbibliothek (std::..., STL, etc.) beantworten
- 13 Ich weiß, welche #include Anweisungen nötig sind, um die Mittel der Standardbibliothek zu verwenden, bzw. wie ich das herausfinden kann
- 14 Ich kann selber eine class schreiben und verstehe sogar die Syntax dabei
- 15 Ich habe schon einmal random\_shuffle und sort auf einen vector angewendet
- 16 Ich verstehe und könnte selber erklären, was die Begriffe declaration, definition, default constructor, copy constructor, assignment bedeuten
- 17 Ich verstehe allgemein die Syntax einer Funktionsdeklaration und Funktionsdefinition
- 18 Ich verstehe den Unterschied zwischen Argumentübergabe "by value" und "by reference"
- 19 Ich kann meinen Programmcode so auf .h und .C Dateien aufteilen, wie es bei größeren Projekten üblich ist
- 20 Ich weiß, wozu ein #include guard gut ist
- 21 Ich habe die Aufgabe mit der pow() Funktion erfolgreich bearbeitet
- 22 Ich habe die Aufgabe mit dem FastA Splitter erfolgreich bearbeitet, mein Programm tut, was es soll
- 23 Ich habe die Aufgabe, bei der die Kommandozeilenargumente in eine C++/STL-gemäße Form umkopiert werden, erfolgreich bearbeitet
- 24 Ich habe die readNumbers() und writeNumbers() Funktionen implementiert (Datei ↔ vector)
- 25 Ich habe die removeOutliers() Funktion implementiert
- 26 Ich habe die Aufgabe mit dem Histogramm bearbeitet. Dabei habe ich ein std::map benutzt und erfülle die genannten Anforderungen.

# Aufgabe

- Machen Sie aus Ihrem Histogramm eine “richtige” **class Histogram**.
- Member functions:
  - **add(double)** // Wert hinzufügen
  - Ausgabe der Wertepaare nach **std::ostream**
  - **min(), max()**
  - **setBinWidth(double)**
  - **clear()** // ist z.B. nach einem setBinWidth nötig
  - ...

# Aufgabe

- Schauen Sie sich das Beispiel **random\_shuffle\_sort.C** auf der Webseite an
- Begründen Sie, warum sich die Ergebnisse der Anwendung von **sort()** unterscheiden, je nach dem, worauf man es anwendet:
  - **std::vector<char\*>**
  - **std::vector<std::string>**

# DAS PROJEKT

1. LC-MS/MS Daten

2. Map Alignment

- Features gruppieren
- Retentionszeiten korrigieren

3. Visualisierung, differentielle Analyse

# DAS PROJEKT

## 1. LC-MS/MS Daten

## 2. Map Alignment

- Features gruppieren
- Retentionszeiten korrigieren

## 3. Visualisierung, differentielle Analyse

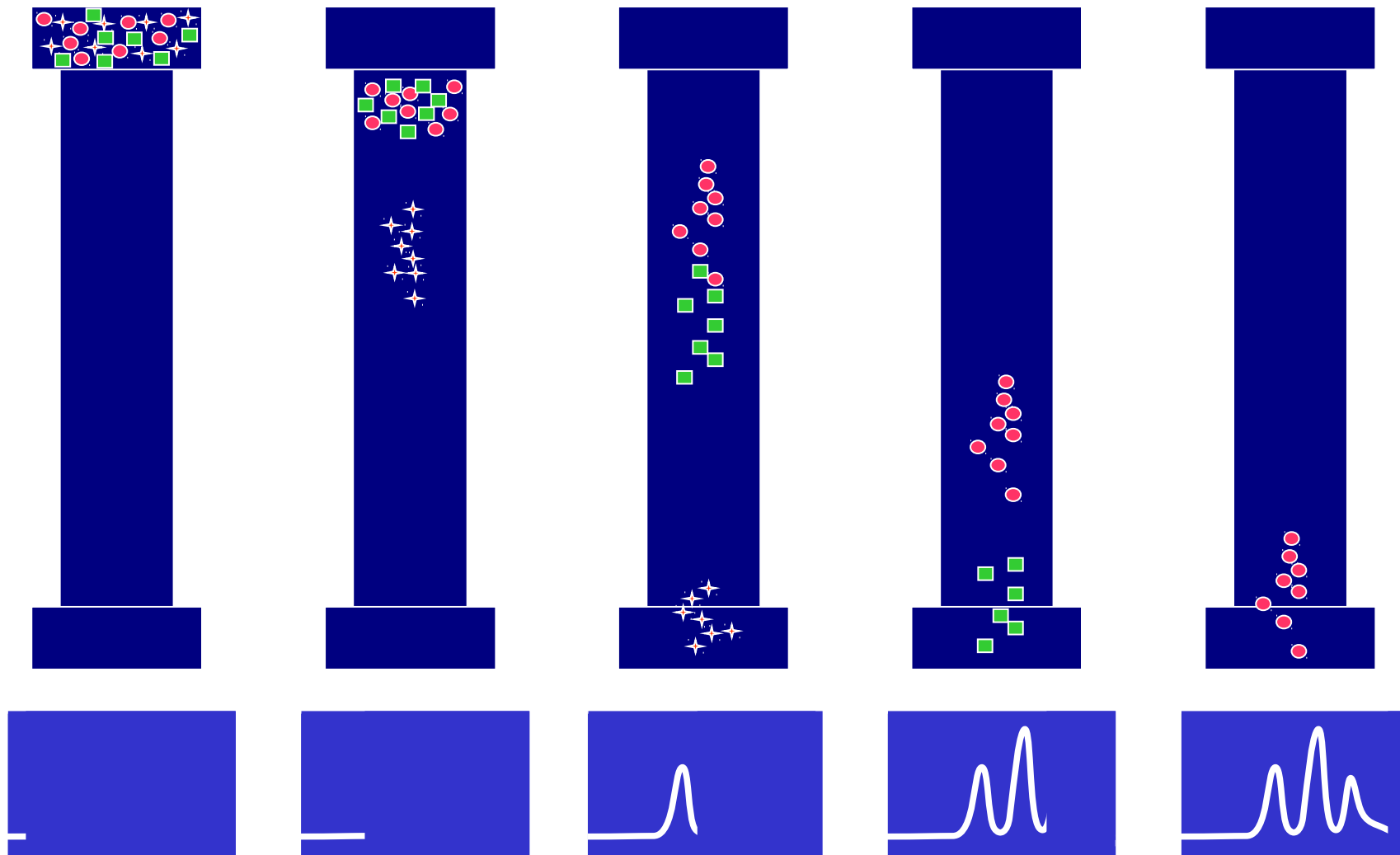
# LC-MS/MS

- Eine wichtige experimentelle Methode in der Bioanalytik und Proteomik
  - LC = liquid chromatography = Flüssigkeitschromatographie
  - MS = mass spectrometry = Massenspektrometrie
  - MS/MS = tandem mass spectrometry = Tandemmassenspektrometrie

# “Bindestrich – Methoden”

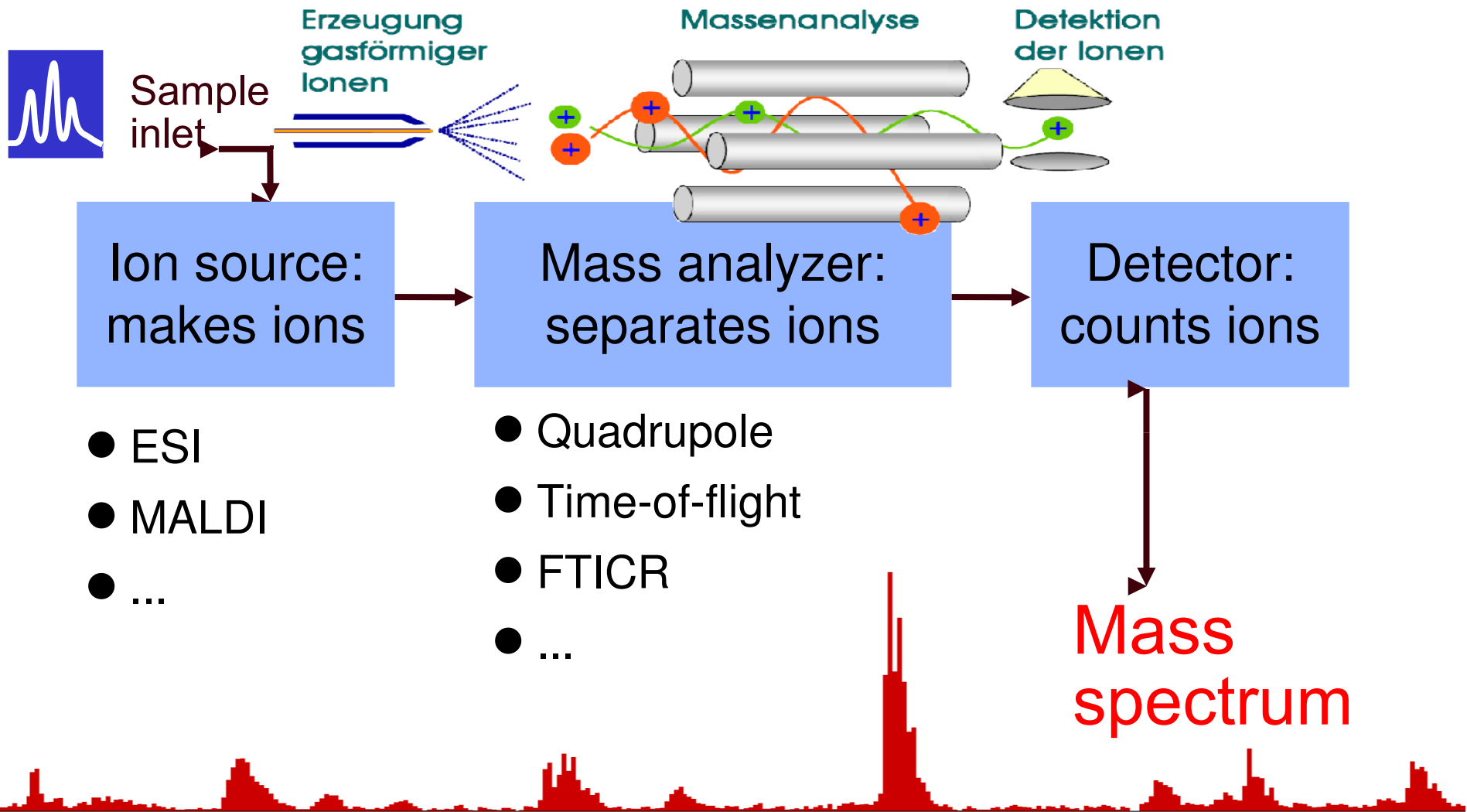
- Ziel: Analyse (qualitativ und quantitativ) von komplexen Proben
- Weg: Kombination *mehrerer* Trennmethoden
- *Zum Beispiel:*
  1. Flüssigkeitschromatographie  
[oder 2D Gele], *dann*
  2. Massenspektrometrie, *dann*
  3. Tandem Massenspektrometrie

# Liquid Chromatography



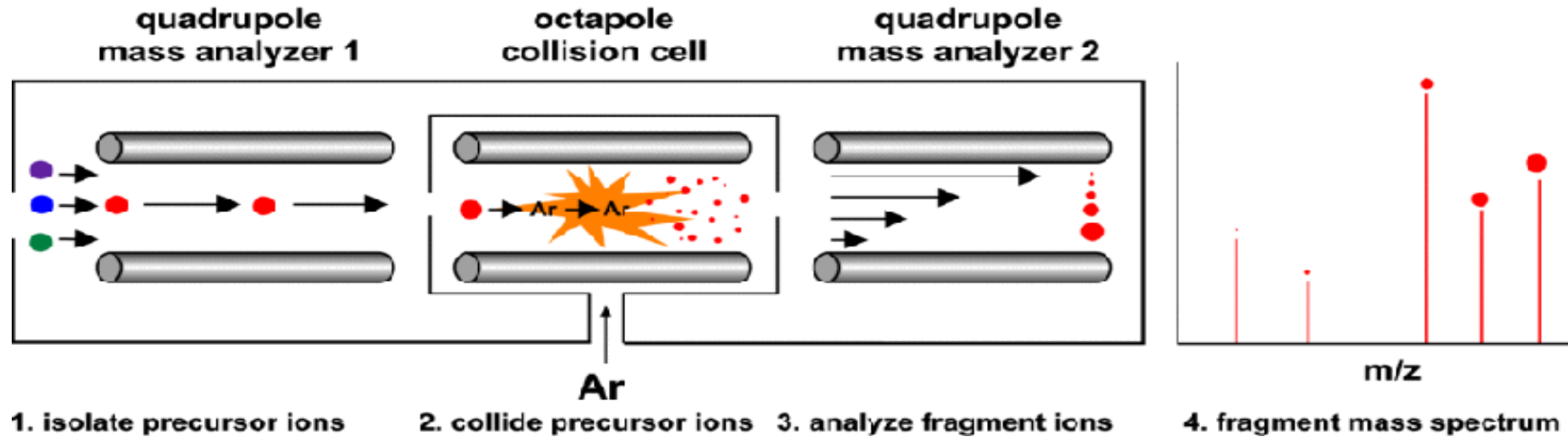
“Stationäre” und “mobile” Phase

# Mass Spectrometry

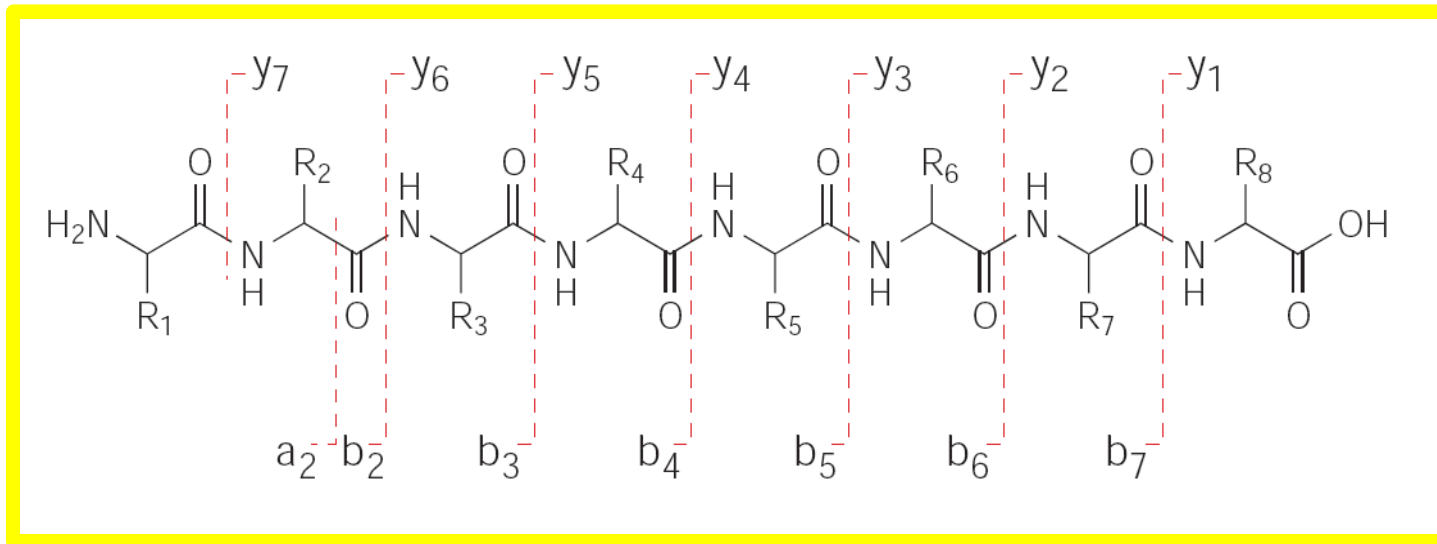


# Tandem Mass Spectrometry

Collision induced dissociation (CID) in a triple quadrupole



Peptide

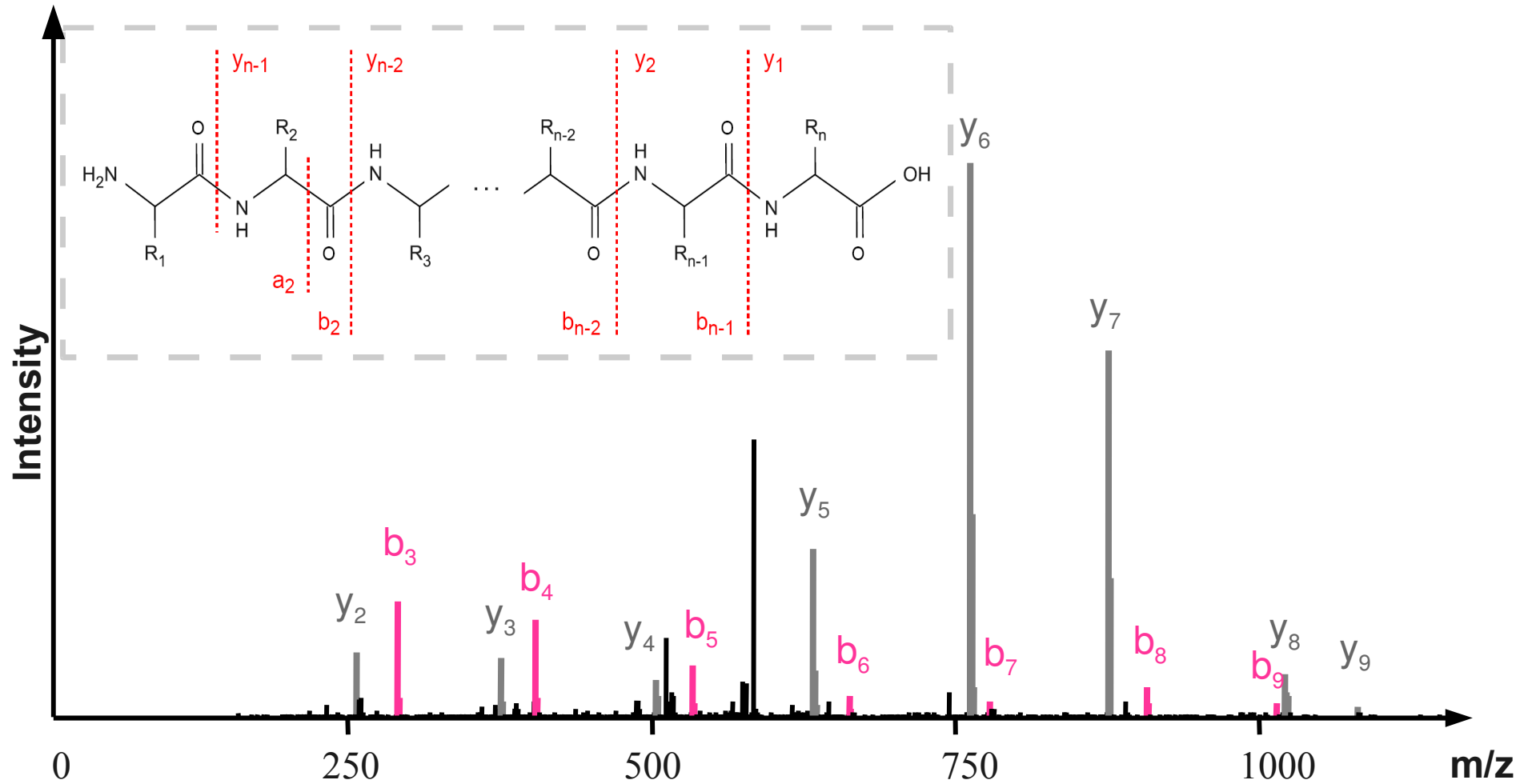


# Tandem Mass Spectrometry

*Peptide: S-G-F-L-E-E-D-E-L-K*

<b>MW</b>	<b>ion</b>			<b>ion</b>	<b>MW</b>
88	b <sub>1</sub>	S	GFLEEDELK	y <sub>9</sub>	1080
145	b <sub>2</sub>	SG	FLEEDELK	y <sub>8</sub>	1022
292	b <sub>3</sub>	SGF	LEEDELK	y <sub>7</sub>	875
405	b <sub>4</sub>	SGFL	EEDELK	y <sub>6</sub>	762
534	b <sub>5</sub>	SGFLE	EDELK	y <sub>5</sub>	633
663	b <sub>6</sub>	SGFLEE	DELK	y <sub>4</sub>	504
778	b <sub>7</sub>	SGFLEED	ELK	y <sub>3</sub>	389
907	b <sub>8</sub>	SGFLEEDE	LK	y <sub>2</sub>	260
1020	b <sub>9</sub>	SGFLEEDEL	K	y <sub>1</sub>	147

# Tandem Mass Spectrometry



<b>88</b>	<b>145</b>	<b>292</b>	<b>405</b>	<b>534</b>	<b>663</b>	<b>778</b>	<b>907</b>	<b>1020</b>	<b>1166</b>	<b>b ions</b>
<b>S</b>	<b>G</b>	<b>F</b>	<b>L</b>	<b>E</b>	<b>E</b>	<b>D</b>	<b>E</b>	<b>L</b>	<b>K</b>	
<b>1166</b>	<b>1080</b>	<b>1022</b>	<b>875</b>	<b>762</b>	<b>633</b>	<b>504</b>	<b>389</b>	<b>260</b>	<b>147</b>	<b>y ions</b>

# LC-MS/MS

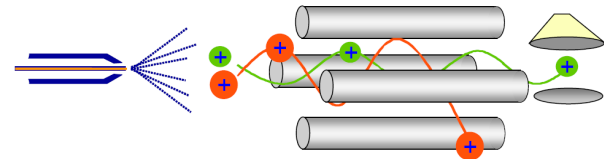
- **HPLC** : High-performance liquid chromatography

Peptides elute at different retention times (**RT**)  
from a chromatographic column



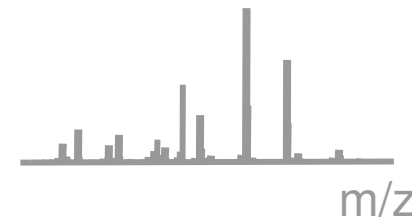
- **MS** : Mass spectrometry

Peptides differ by mass/charge ratios (**m/z**)  
in a mass spectrum

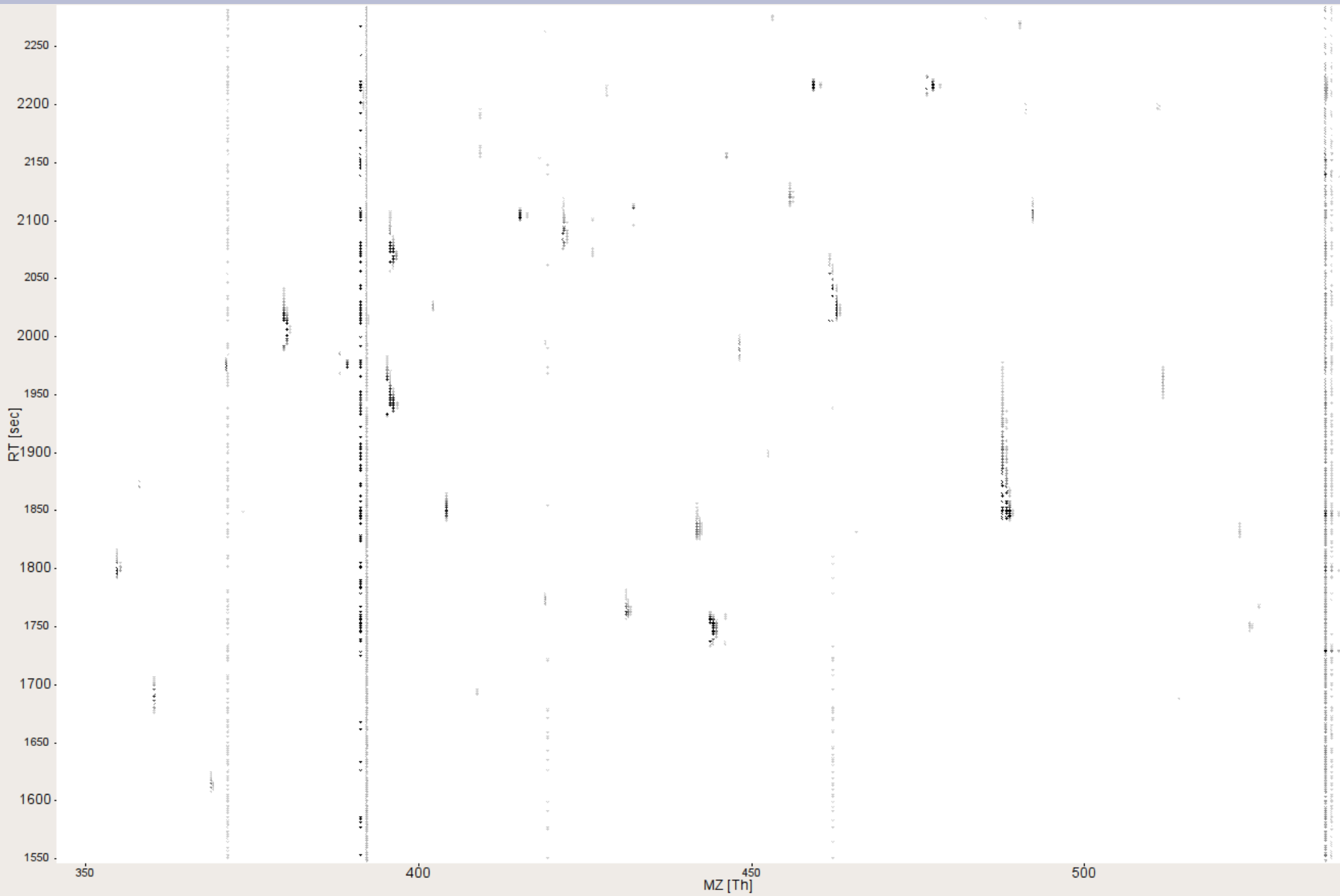


- **MS/MS** : Tandem mass spectrometry

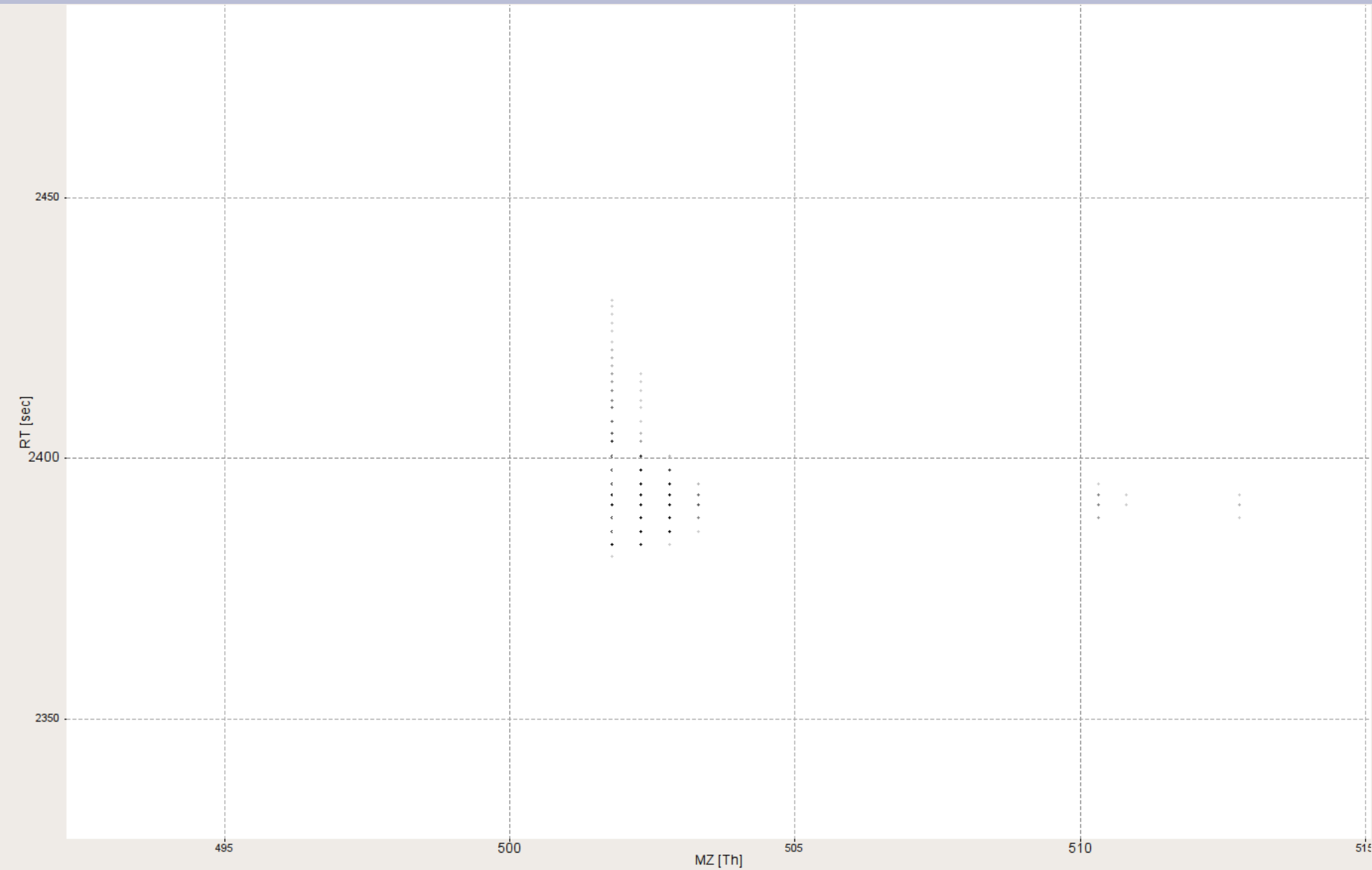
Selected ions in a mass spec can be further  
fragmented => derived mass spectrum



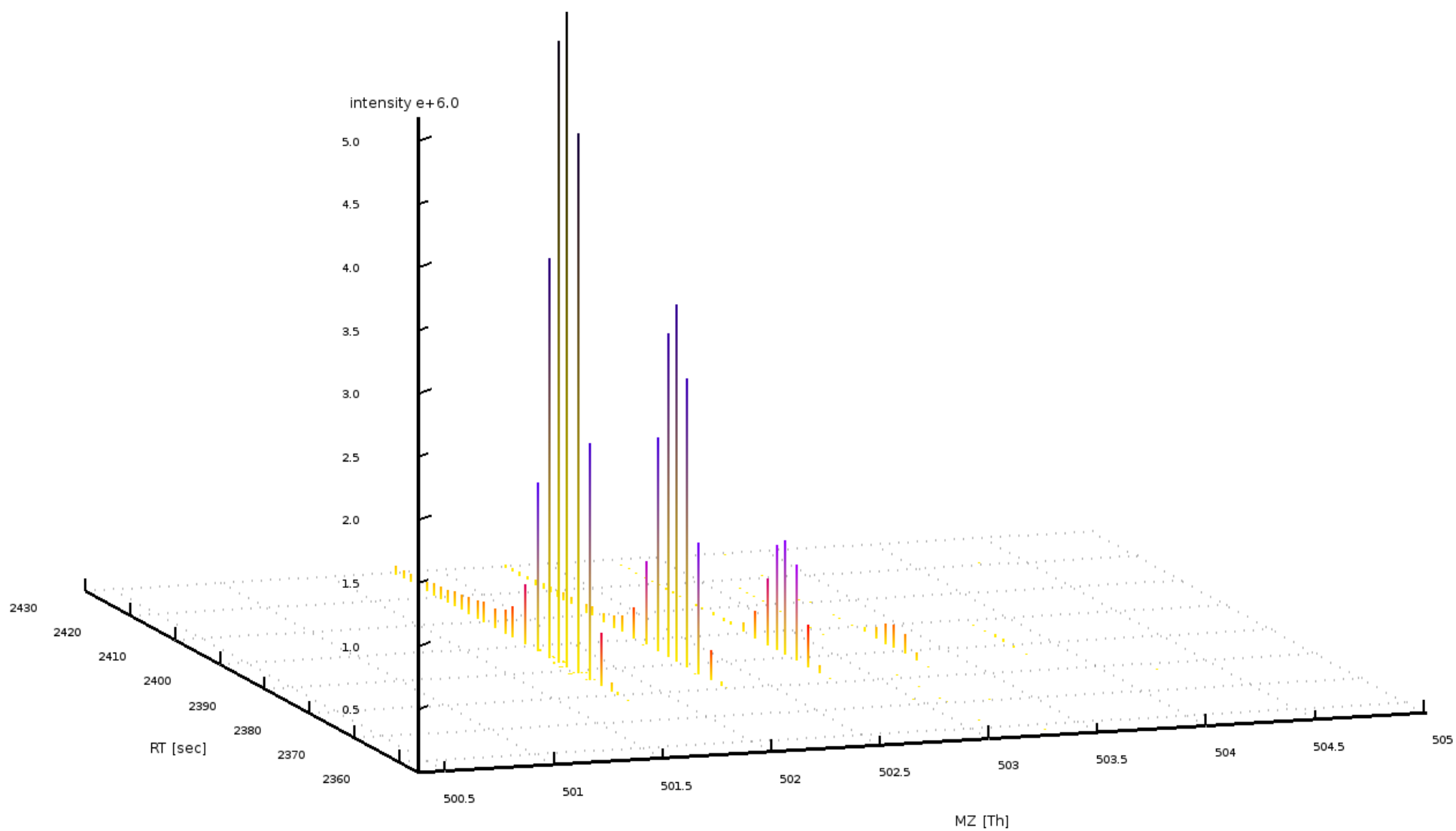
# LC-MS/MS - Daten



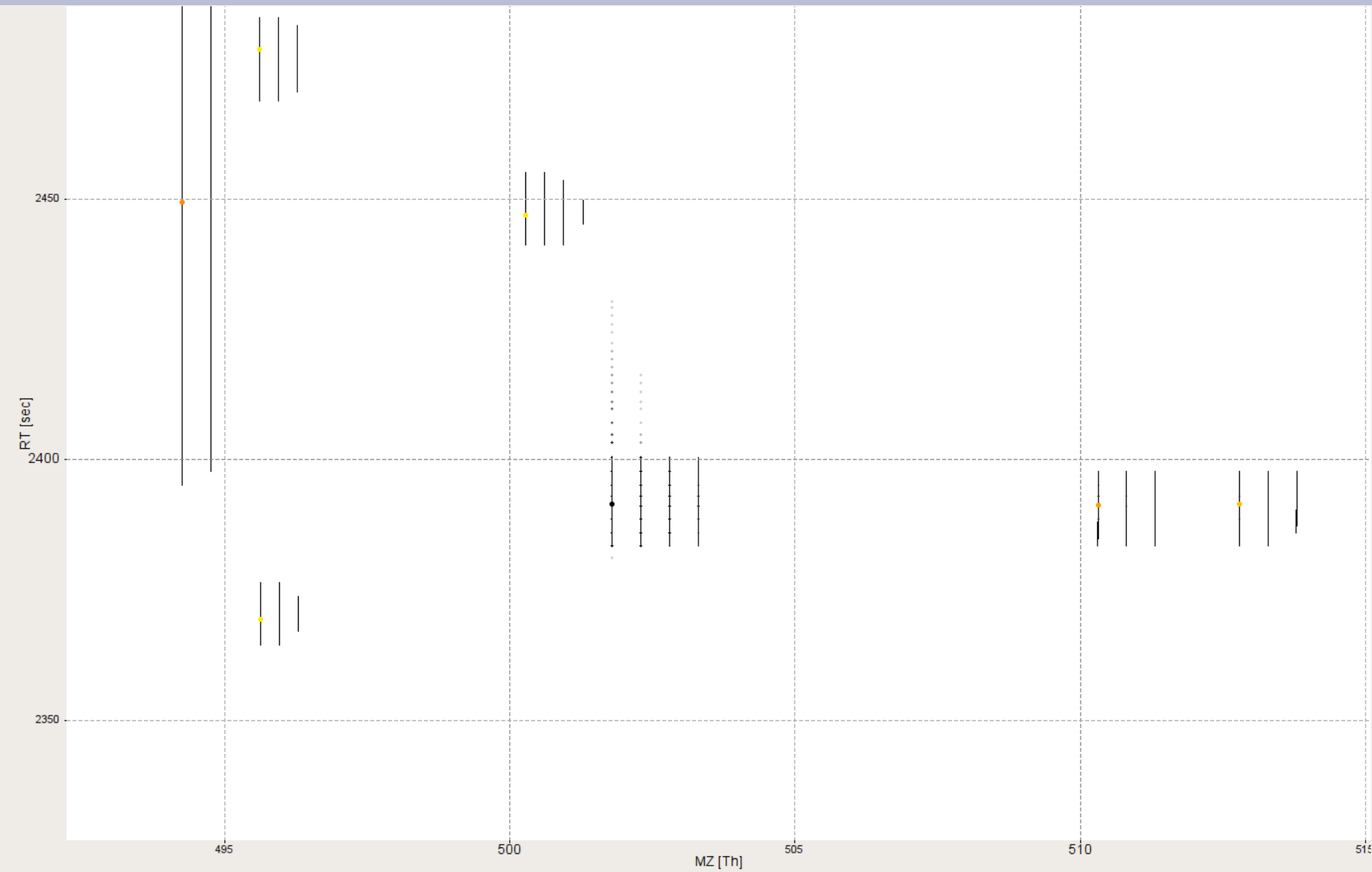
# LC-MS/MS - Daten



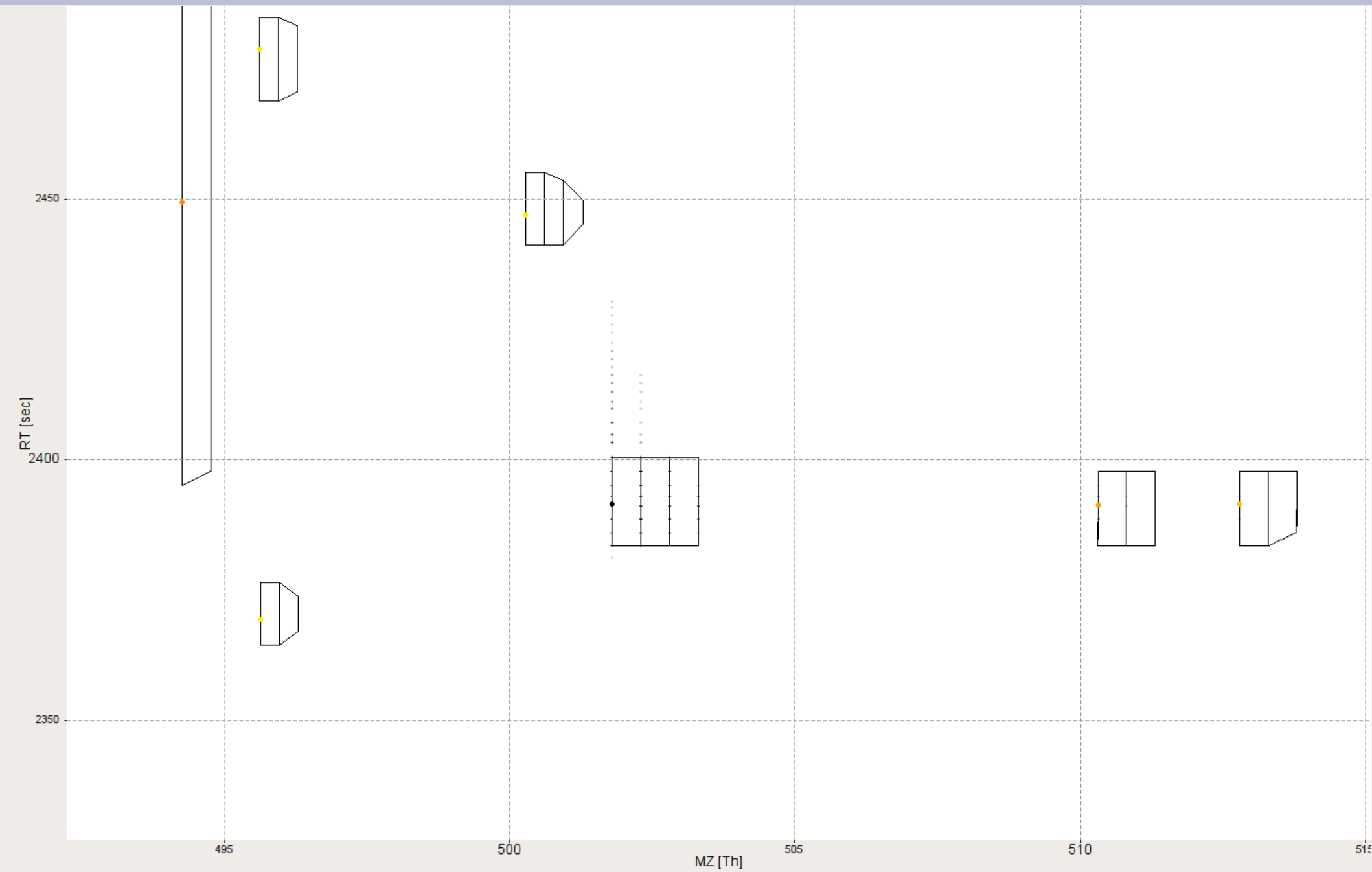
# LC-MS/MS - Daten



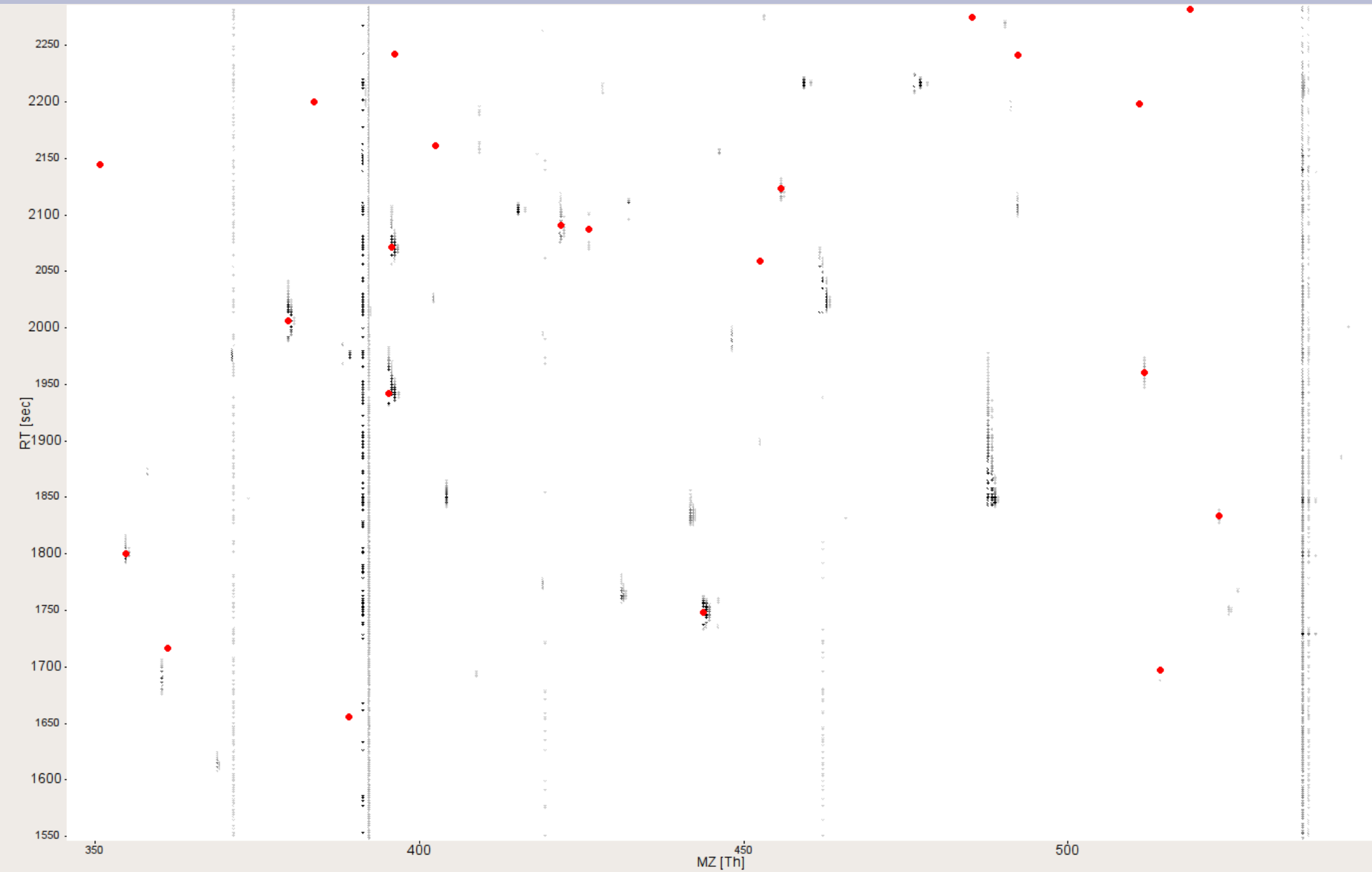
# LC-MS/MS - Daten



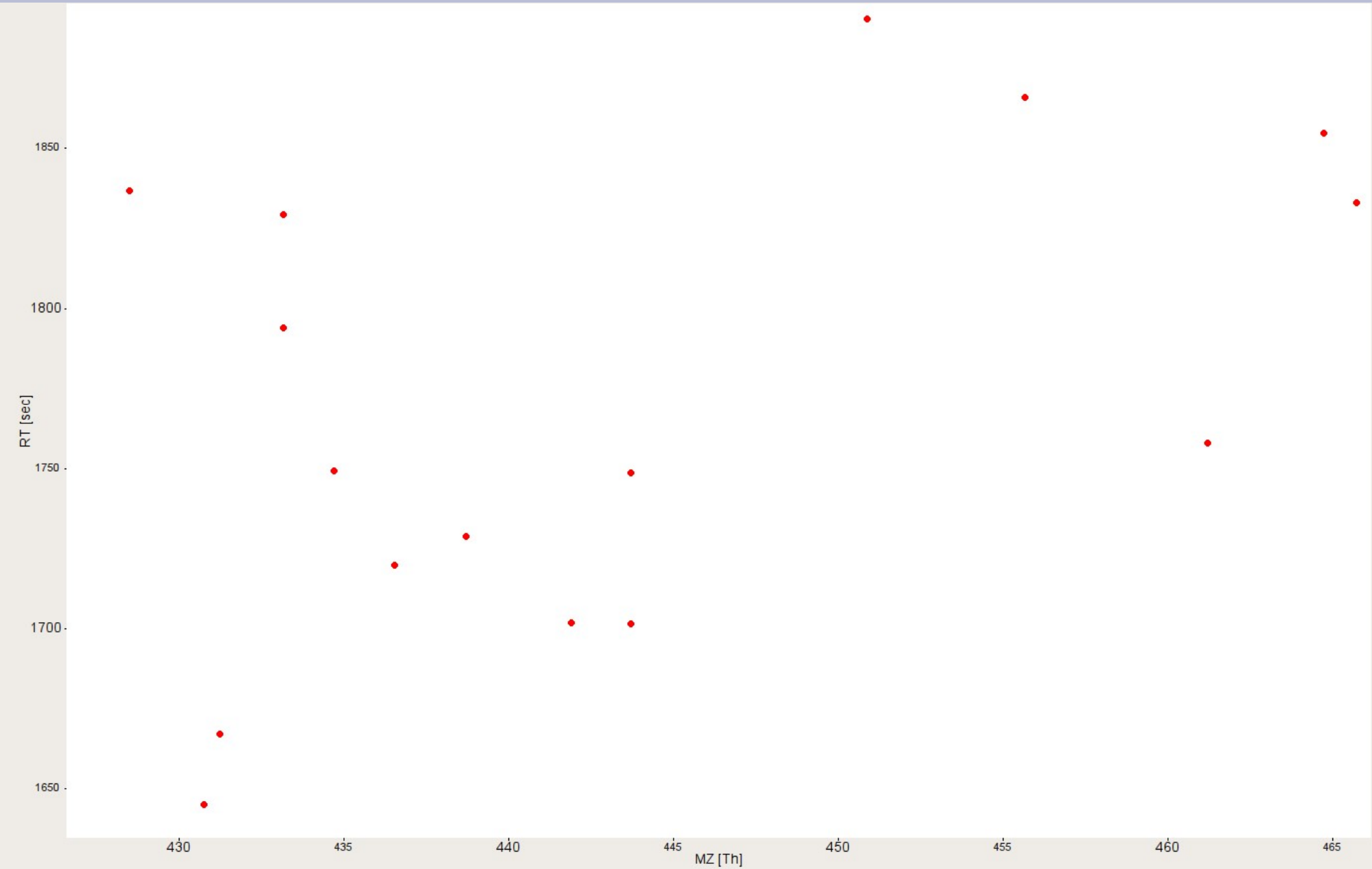
# LC-MS/MS - Daten



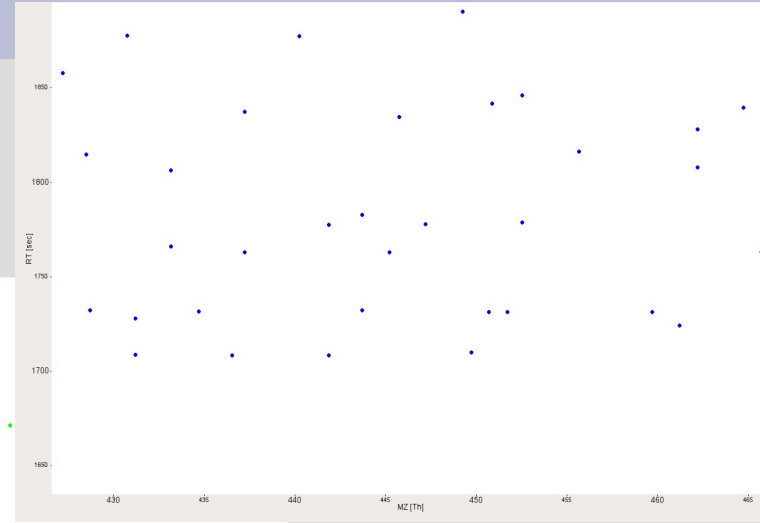
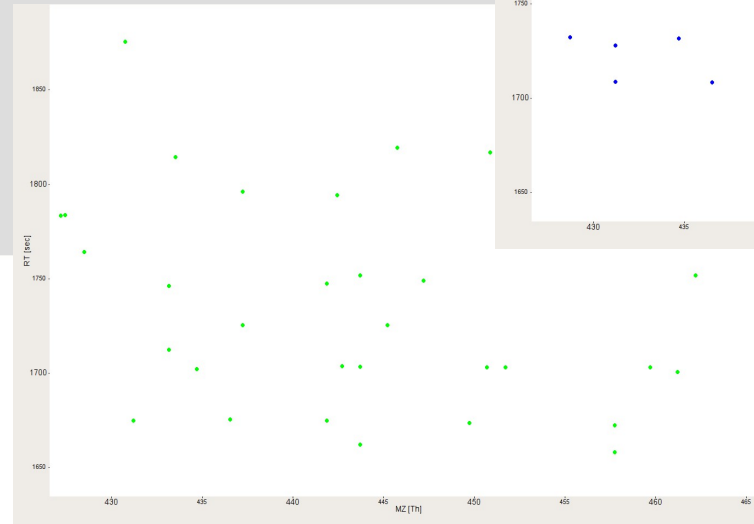
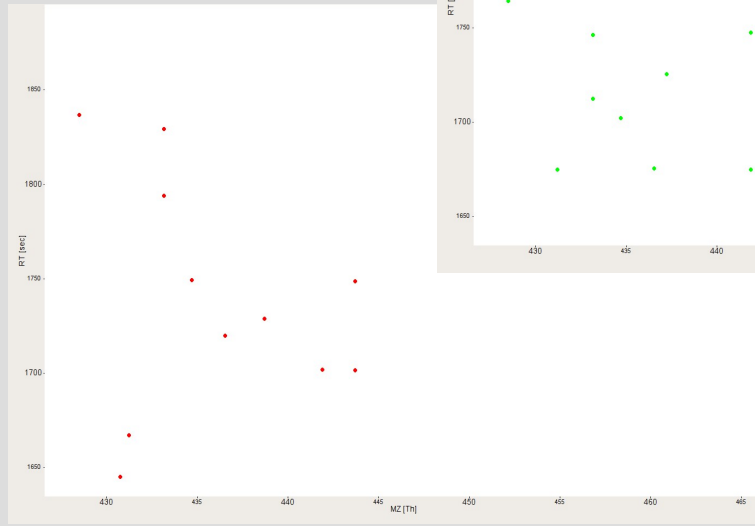
# LC-MS/MS - Daten



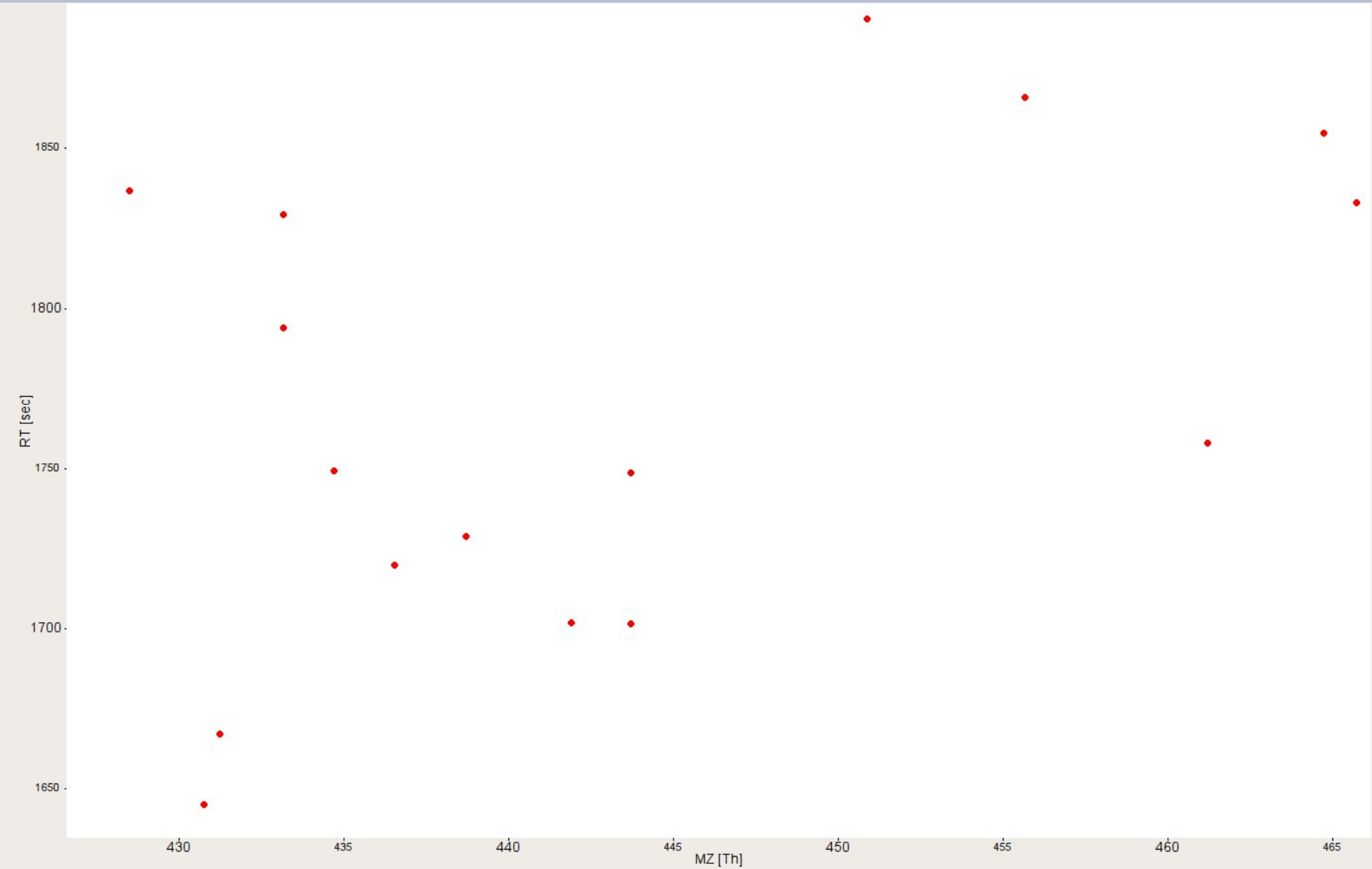
# LC-MS/MS - Daten



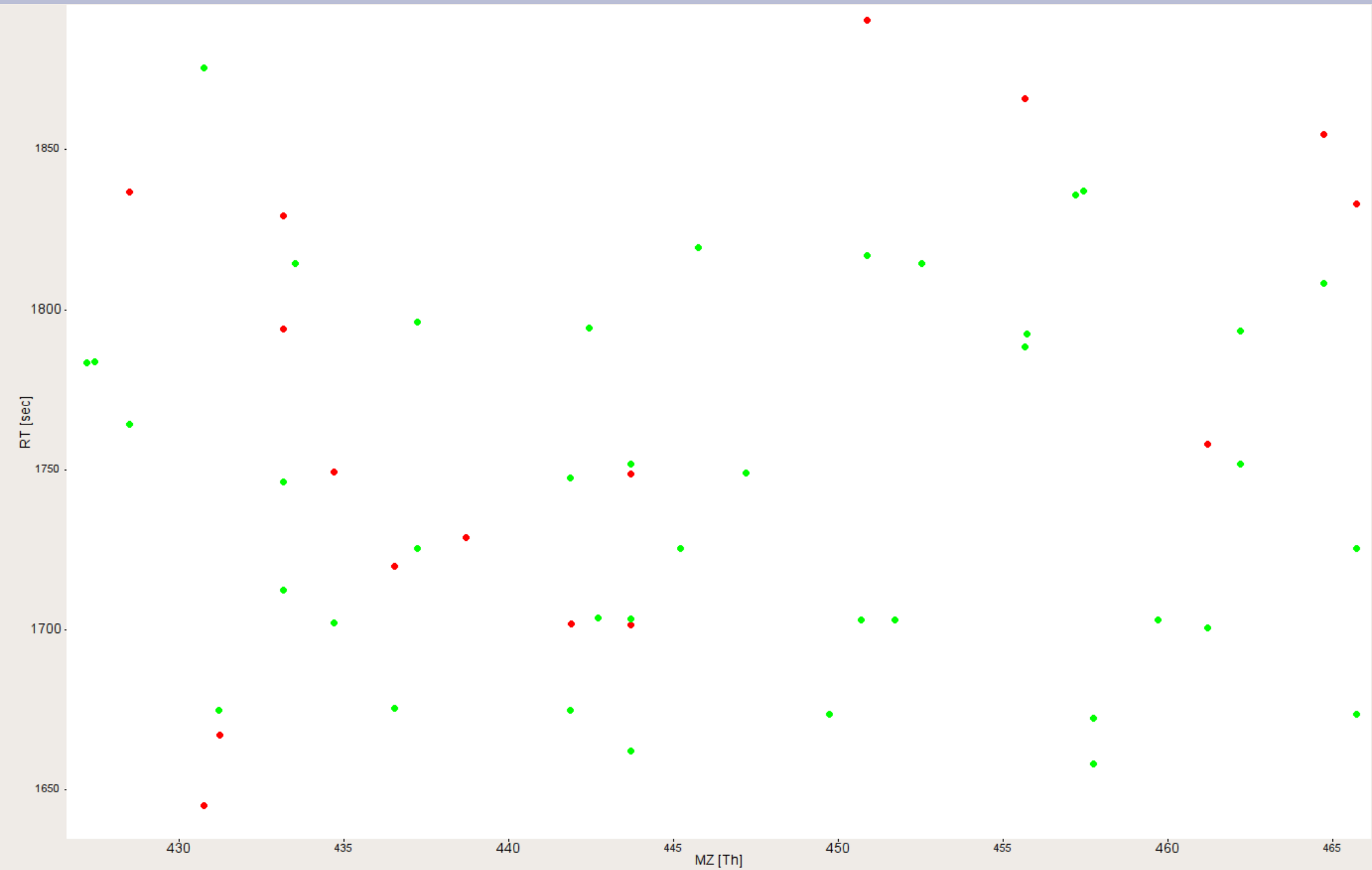
# LC-MS/MS - Daten



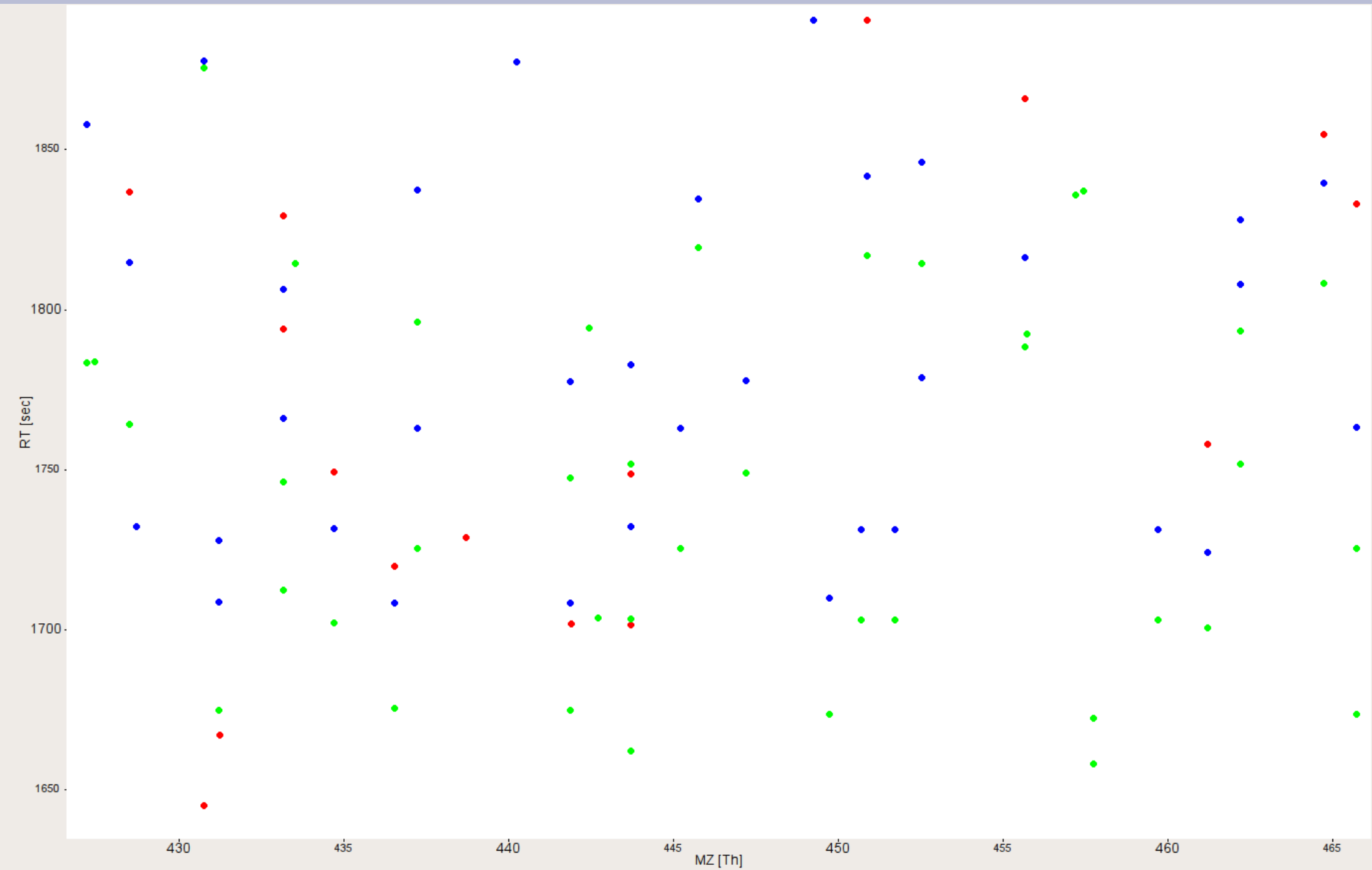
# LC-MS/MS - Daten



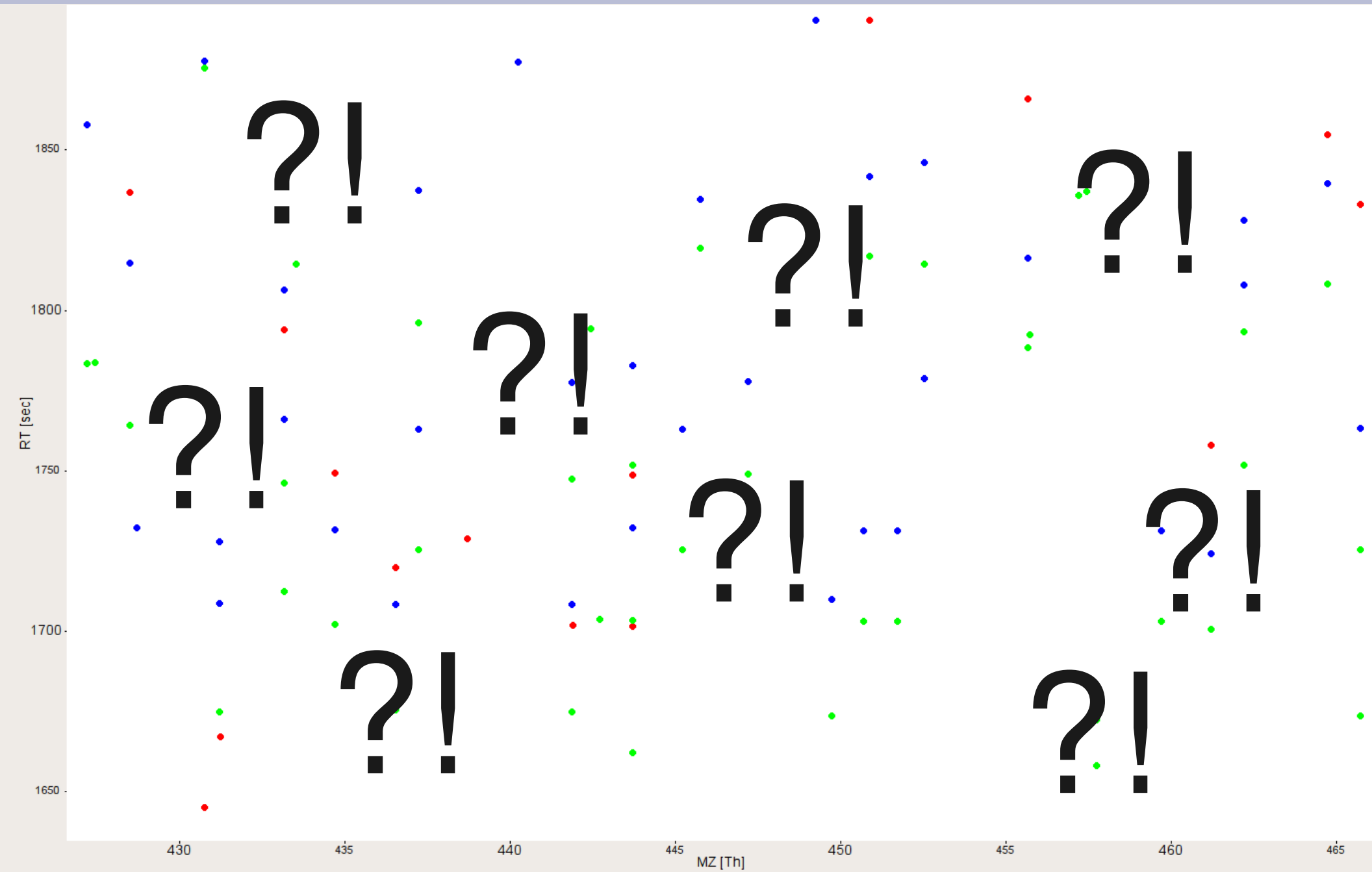
# LC-MS/MS - Daten



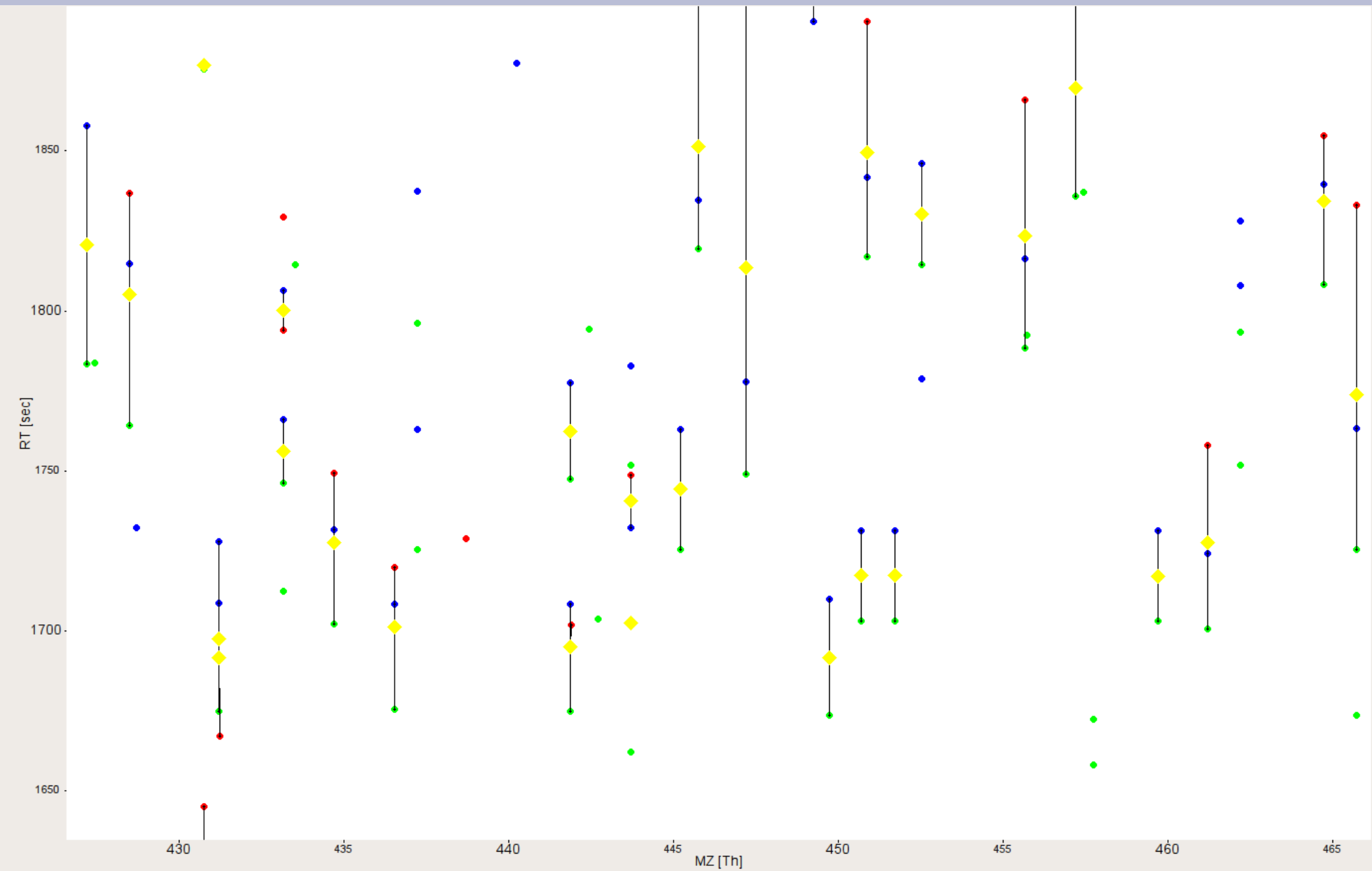
# LC-MS/MS - Daten



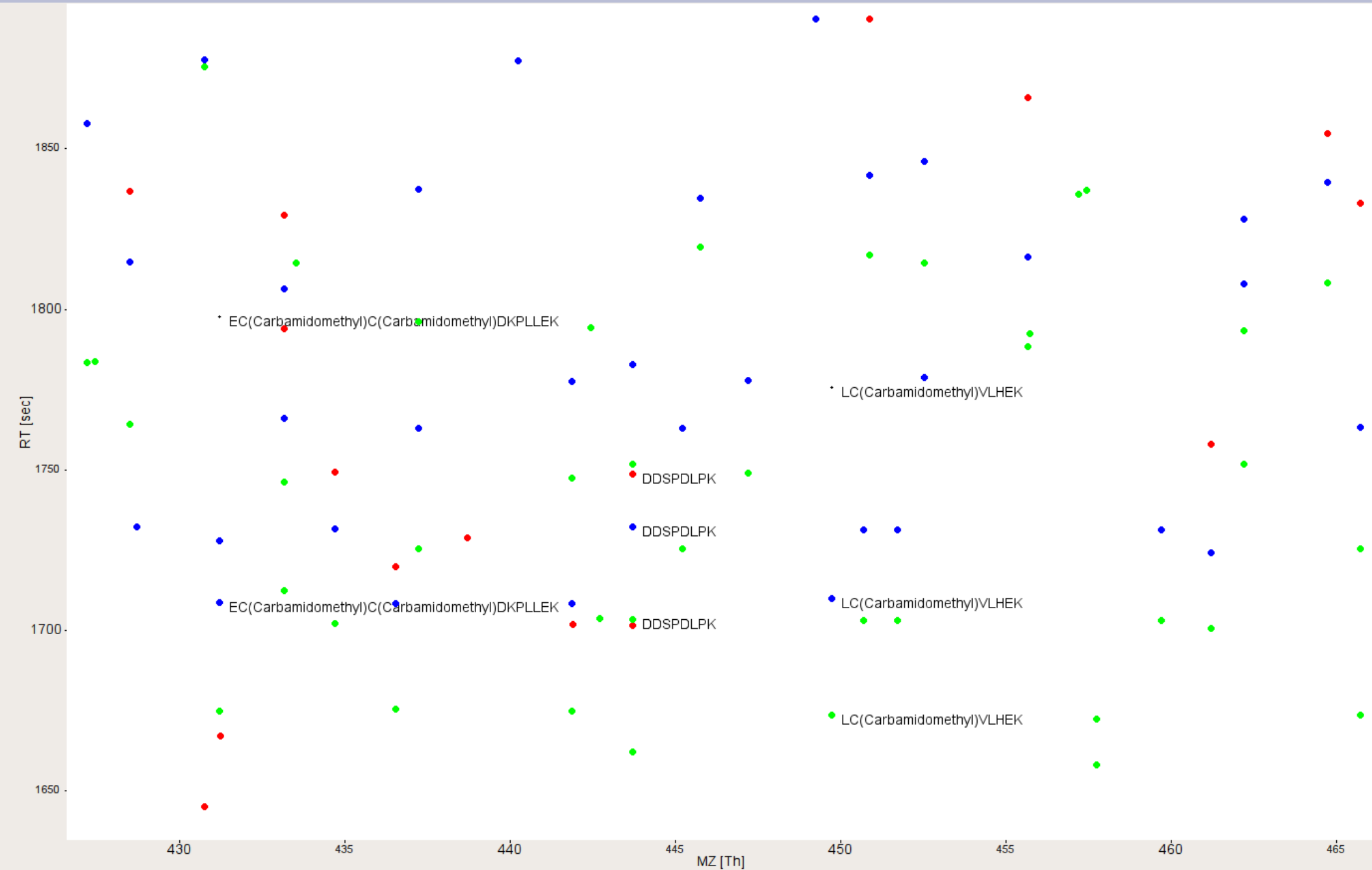
# LC-MS/MS - Daten



# LC-MS/MS - Alignment



# LC-MS/MS - Alignment





# DAS PROJEKT

1. LC-MS/MS Daten

**2. Map Alignment**

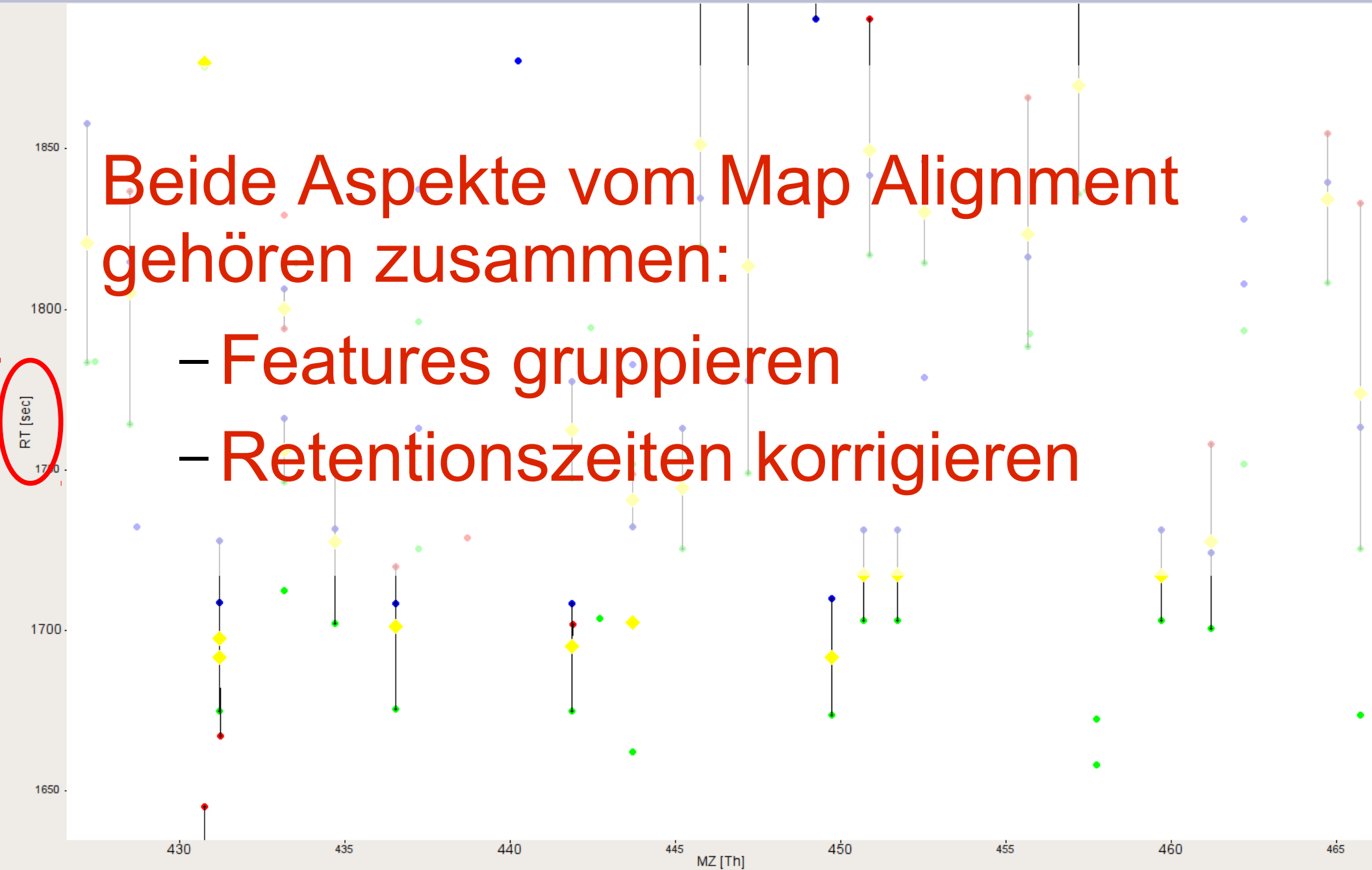
- **Features gruppieren**
- **Retentionszeiten korrigieren**

3. Visualisierung, differentielle Analyse

# LC-MS/MS - Alignment

Beide Aspekte vom Map Alignment gehören zusammen:

- Features gruppieren
- Retentionszeiten korrigieren



# Features gruppieren

- Ähnliches  $m/z$  [Massenspektrometrie]  
(mass-charge ratio, Masse-Ladungs-Verhältnis)
- Ähnliches RT [Chromatographie]  
(retention time, Retentionszeit)
- Gleiche Ladung [Massenspektrometrie]  
(Isotopenmuster)
- Gleiche Peptid-Annotation [Tandem MS]  
(verwendet MS/MS etc.)
- Erlaubte Toleranzen können z.B. anhand von Histogrammdateien geschätzt werden...

# Features gruppieren

- Notation: Feature  $f = ( f.RT, f.MZ )$
- Mögliches Distanzmaß für Features dann:

$$\text{dist} ( f, g ) := a ( f.RT - g.RT ) ^ 2 + b ( f.MZ - g.MZ ) ^ 2$$

wobei  $a, b$  Konstanten sind, die vom Nutzer gegeben werden, oder per Histogramm geschätzt werden.

# Retentionszeiten korrigieren

- Verschiedene Klassen von Funktionen sind anwendbar
  - Shift:  $f(x) = x + a$ , für  $a$  const.
  - Affine:  $f(x) = ax + b$ , für  $a, b$  const.
  - Spline: z.B. cubic B-Spline
- Um robust gegen Ausreißer zu sein, kann man z.B. eine longest increasing subsequence zugrunde legen.

# Ablauf

- Für 2 Maps:
  - Distanzfunktion implementieren
  - [ Schwellenwert für Gruppenbildung schätzen mittels Histogramm ]
  - Gruppen (= Paare) finden zwischen Maps
    - Einfach: Distanz genügend klein
    - [ Besser: Finde für jedes Feature jeweils den nächste Nachbarn im anderen Map. Wenn beide Pfeile aufeinander zeigen, werden sie ein Paar ]
- RT Korrektur paarweise:
  - Gruppen finden, “irgendwie” (Startlösung)
  - [ Ausreißer entfernen, z.B. mit longest increasing subsequence ]
  - Mittels linearer Regression Schätzer für a und b in Korrekturfunktion  $f(x) = ax+b$  finden  
Oder: Verschiebung schätzen für  $f(x) = x+a$
  - Korrekturfunktion f anwenden auf 2. Map
- Zwischen Gruppieren und RT Korrektur hin- und her wechseln, bis Konvergenz
- Für k Maps:
  - z.B. Maps 2, 3, ... jeweils mit Map 1

# Ablauf

- Für 2 Maps:
  - Distanzfunktion implementieren
  - [ Schwellenwert für Gruppenbildung schätzen mittels Histogramm ]
  - Gruppen (= Paare) finden zwischen Maps
- RT Korrektur paarweise:
  - Gruppen finden, “irgendwie” (Startlösung)
  - [ Ausreißer entfernen, z.B. mit longest increasing subsequence ]
  - Mittels linearer Regression Schätzer für a und b in Korrekturfunktion  $f(x) = ax+b$  finden  
Oder: Verschiebung schätzen für  $f(x) = x+a$
  - Korrekturfunktion f anwenden auf 2. Map
- Zwischen Gruppieren und RT Korrektur hin- und her wechseln, bis Konvergenz
- Für k Maps:
  - z.B. Maps 2, 3, ... jeweils mit Map 1

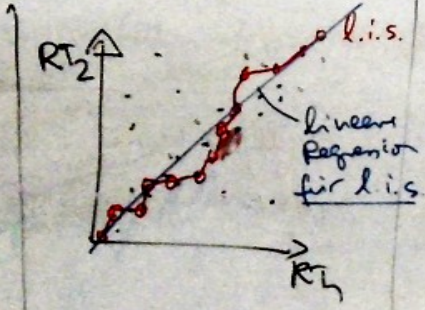
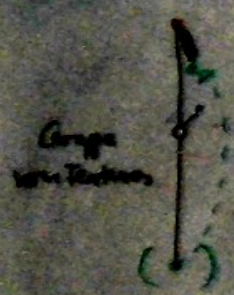
- für 2 Feature Maps,
  - Distanzfunktion implementieren
  - Gruppen finden (zwischen den Maps)

[ Histogramm für Distanzen,  
Schwellenwert für Gruppenbildung schätzen. ]

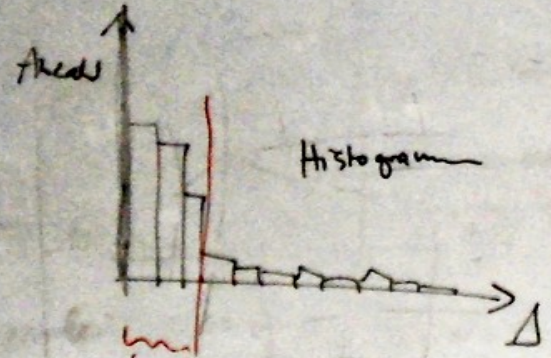
- Für  $k$  viele Feature Maps  
(- Gruppen finden)

- RT Korrekturen paarweise
  - Gruppen finden (irgend wie) Startlösung

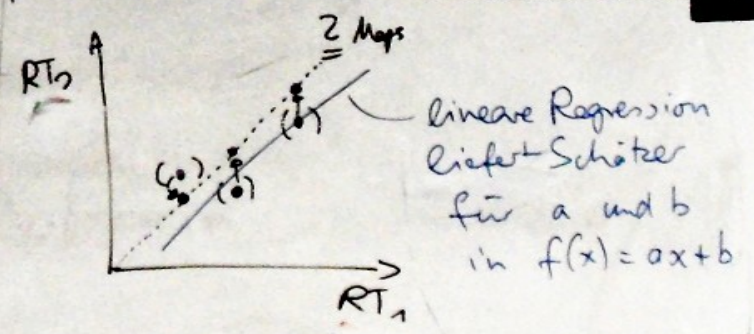
- Anreißer entfernen  
- mit longest increasing subsequence
- Lineare Regression (für fix  $a$  und  $b$ )  
oder Verschiebung schätzen (für  $f(x) = x + c$ )
- $f$  anwenden auf 2. Map.



h.i.s. / l.i.s.  
doxygen  
factory design pattern  
Feature / Feature Map /  
Repräsentation von Gruppen von Features



erlaubte Distanz (geschätzt)



# Multiples Map Alignment

- In der Regel hat man mehr als zwei LC-MS(/MS) Datensätze zu vergleichen.

Mögliche Ansätze hierbei:

- Star-like
- Progressive
- Clustering

# DAS PROJEKT

1. LC-MS/MS Daten

2. Map Alignment

- Features gruppieren
- Retentionszeiten korrigieren

**3. Visualisierung,  
differentielle Analyse**

# Gnuplot anwenden ...

- Beispiel für Visualisierung von longest increasing subsequence:

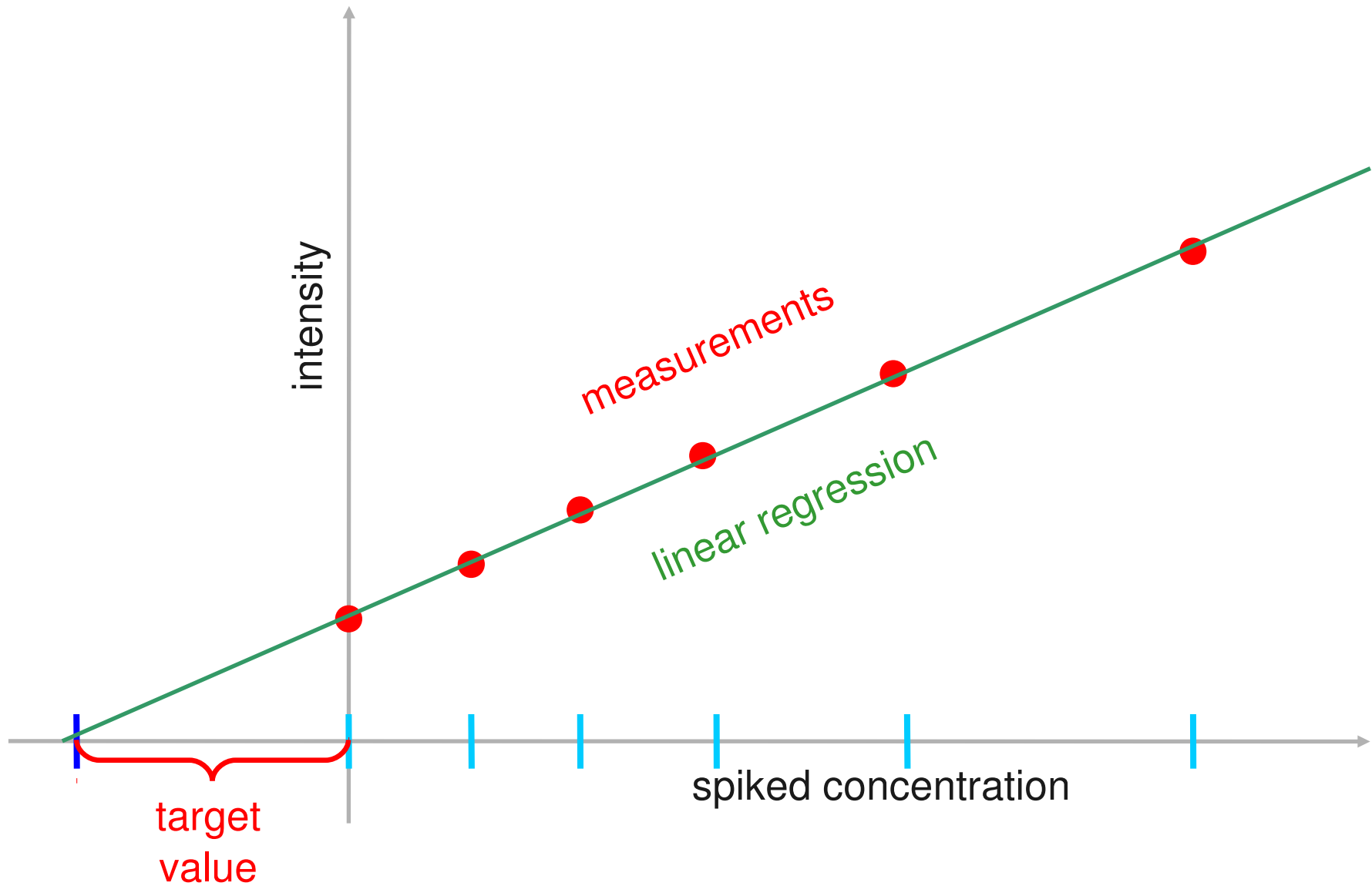
Webseite → Directory → heaviest increasing subsequence  
→ plot\_result.gp, dump\_edges.wsv, dump\_path.wsv

- Aufrufen mit: **gnuplot plot\_result.gp**

# Gnuplot anwenden ...

- Beispiel für Visualisierung von Feature Daten:  
`splot './BSA1.dta2d' with impulses`

# “Additive” Method



# Differentielle Analyse

- Findet *signifikante* Unterschiede.
  - Biomarker, Diagnostik, ...
  - Eventuelle übersehene Features an der erwarteten Stelle detektieren

# Benotung

- Mündliche Prüfung im Februar, 30 min.  
Zwei Bereiche:
  - Vorführung des eigenen Programms
  - Erläuterung der Implementierung in C++
- Noten (ohne Gewähr, nur zur Orientierung)
  - 4: Programm läuft irgendwie, elementare C++ Kenntnisse
  - Note 3: das Programm läuft stabil, C++ wird manchmal sinnvoll eingesetzt
  - Note 2: alles implementiert, C++ ordentlich
  - Note 1: mindestens ein Schmankerl :-)

# Prüfungstermine

- Do. 11. Februar
  - 14:00 Sebastian Pink
  - 14:40 Mathias Bässler
  - Ab 16:00 Stefan Funke weg
- Mi. 24. Februar
  - Bis 11:30 Stefan Funke keine Zeit
  - 11:30 Christina Glock
  - 12:10 Karin Bokelmann
  - 14:00 Sebastian Lüder
  - 14:40 Johannes Dinter
  - [ ~~15:20 Julia Schüler (???)~~ ]

- Ok das reicht erst mal ..

