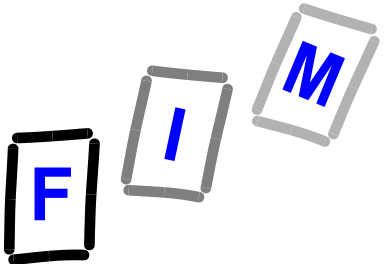


Betriebssysteme

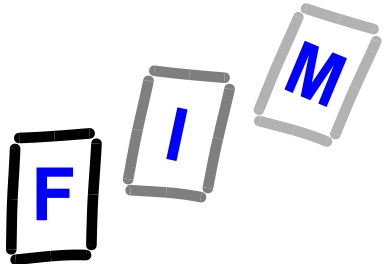
I: Speicherverwaltung
(Teil A: Adress-Bindung)

Ausgangssituation

- **Programm ist auf einem Massenspeicher als „binary executable file“ gespeichert.**
- **Das Programm muss zur Ausführung in den Hauptspeicher geladen werden**
- **Code eines Prozesses kann während der Laufzeit …**
  - **zwischen externem Speicher und Hauptspeicher verschoben werden,**
  - **im physischen Hauptspeicher selbst verschoben werden**

Absolute Adressen

- **Fixe (→ absolute) Adresszuweisung findet man nur mehr in kleinen Mikroprozessor-Systemen, es wird a priori festgelegt**
  - **Wo eine Instruktion gespeichert wird**
  - **Wo (Adresse) die Operanden einer Instruktion gespeichert sind**
- **Diese Art der Adresszuweisung ist typisch für Programmieren in Maschinencode**
  - **Beispiel serielle Schnittstelle: Ist immer (PC!) an Adresse 0x000003F8-3FF (Nr 1; IO, nicht Speicher)**



Absolute Adressen

Adresse Opcodess

- 0110 db 00 a:=p[0]
- 0112 e6 01 a:=a&1; f.z:=(a==0)
- 0114 ca 0110 if(f.z) pc:=110
- 0117 db 01 a:=p[1]
- 0119 32 0168 m[168]:=a

- 0168 ??

Absolute Adressen

- **Ein Compiler oder Assembler kann absolute Adressen erzeugen**
  - **Normalerweise eine Option (switch) zur Übersetzungs-/Assemblierungszeit**
  - **Daher kann man Assemblercode oder sogar Hochsprachen zur Erstellung von ausführbaren Dateien (executables) für Micro-Systeme verwenden**
- **Aber: Übersetzer und Assembler erzeugen normalerweise Code mit relativen Adressen**

Relative Adressen

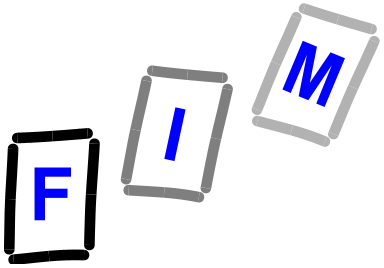
- **Relative Adressen beziehen sich auf einen vorgegebenen Ladepunkt (Ursprung, *starting point, origin*)**
  - **Die „Entfernung“ zu diesem Ursprung heißt „*offset*“**
- **Folgerung:**
  - **Durch relative Adressen kann man den Zeitpunkt der Adressbindung solange verzögern, bis die absolute Adresse des Ladepunktes definitiv bekannt ist, …**
  - **und dann die offsets entsprechend anpassen**

Relative Adressen

Achtung:
Unterschied zu „ca -7“!

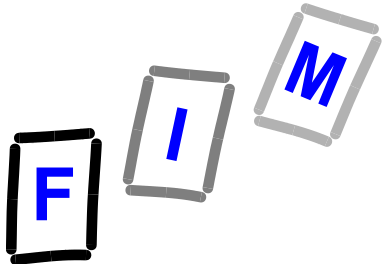
```
                .....
0000r   ..  ...             ...
....r   ..  ...             ...
0010r   db 00     label:    a:=p[0]
0012r   e6 01               a:=a&1; f.z:=(a==0)
0014r   ca 0010r            if(f.z) pc:=label
0017r   db 01               a:=p[1]
0019r   32 0068r            m[adrVal]:=a
....r   ...                 ...
....r
0068r   00          adrVal: ...
```

Legende: r symbolisiert eine relative Adress
Startpunkt 0000 ist fiktiv



Zuweisung von Adressen
(engl.: address binding)

- **Die meisten Systeme erlauben, dass Prozesse in jedem Bereich des physischen Speichers ablaufen können**
  - **Daher muss die erste Adresse eines Prozesses nicht unbedingt 0000h sein**
- **Man benötigt eine Abbildung von den symbolischen Adressen des Quellprogramms auf die tatsächlichen Adressen im Speicher**
  - **Diesen Vorgang nennt man Adresszuweisung *(address binding)***



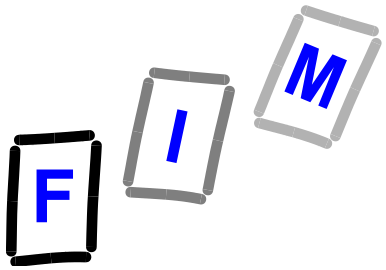
Zuweisung von Adressen

- **Grundsätzlich kann die Zuweisung erfolgen zur**
  - **Übersetzungszeit (engl.: compile time)**
  - **Ladezeit (engl.: load time)**
  - **Laufzeit (engl.: execution time)**
- **Wir werden uns auf die Adresszuweisung zur Ladezeit konzentrieren**

Zuweisung zur Ladezeit (load time binding)

Assembler / Compiler übersetzt den symbolischen Code mit Bezug auf eine symbolische Startadresse, normalerweise 000…0

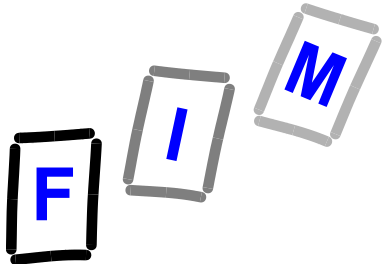
```
                  .....
0000r  .. ...             ...
....r  .. ...             ...
0010r  db 00     label:   a:=p[0]
0012r  e6 01              a:=a&1; f.z:=(a==0)
0014r  ca 0010r           if(f.z) pc:=label
0017r  db 01              a:=p[1]
0019r  32 0068r           m[adrVal]:=a
....r  ...                ...
....r
0068r  00        adrVal:  ...
```



Zuweisung zur Ladezeit

Später (zur Ladezeit) werden die relativen Adressen in absolute Adressen umgewandelt

```
starting_point:= 1200
...
1210  db 00     label:  a:=p[0]
1212  e6 01             a:=a&1; f.z:=(a==0)
1214  ca 1210           if(f.z) pc:=label
1217  db 01             a:=p[1]
1219  32 1268           m[adrVal]:=a
12xx..  ...                ...
....
1268  00        adrVal: ...
```



Zusammenfassung: Adressbindung zur Ladezeit

- **Zur Übersetzungszeit sind die endgültigen Speicheradressen meist noch unbekannt.**
- **Der Compiler erzeugt verschiebbaren (engl.: *relocatable)* Code.**
- **Der Programmlader (siehe später) des BS fügt die „korrekten“ Adressen beim Laden des Programms in den Hauptspeicher ein.**

Lader (Loader)

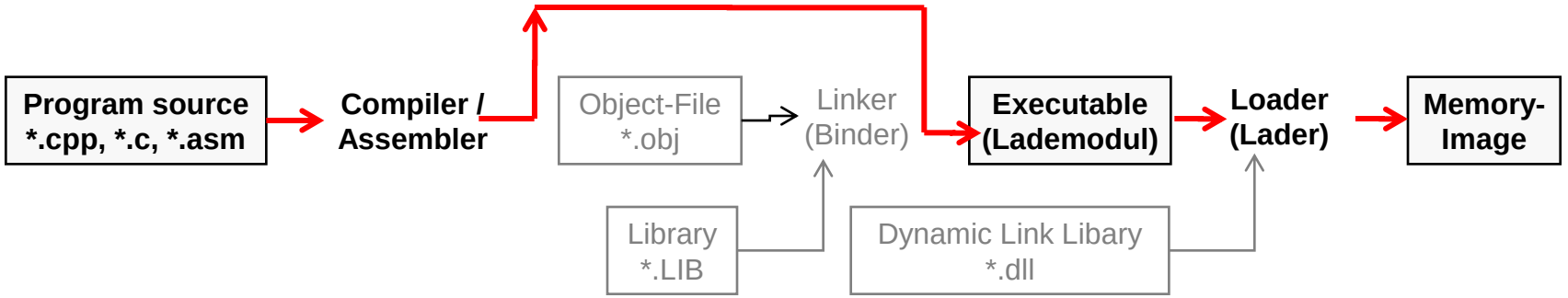
- **Lädt das Programm (z.B. von der Festplatte in den Hauptspeicher)**
- **Macht (falls notwendig) Adressenanpassungen**
  - **Im Falle von absoluten Adressen: Nichts**
- **Veranlasst das Starten des Programms**
  - **Einfach: Weist Adresse der ersten Anweisung dem Programmzähler (Instruktionszähler) zu**
  - **Komplizierter: Führt vorab Initialisierungen aus und dann …**

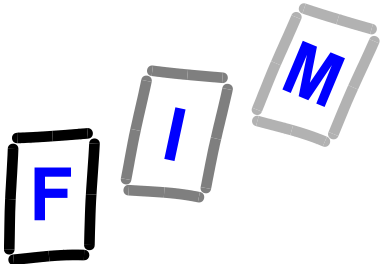
Lader und Adressbindung

- **(i) Binden zur Übersetzungszeit:**
  - **Lader kümmert sich nicht um Adressen, er lädt einfach das Programm-Image 1:1 ab einer fixen Speicheradresse (Ladepunkt)**
- **(ii) Binden zur Ladezeit:**
  - **Der Lader (relocating loader) bildet die relativen Adressen auf absolute ab**

Weg vom Quellcode zur Ausführung

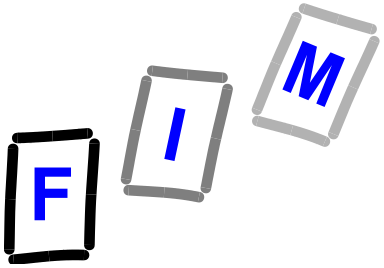
Vereinfachte Annahme:
Nur „1“ Hauptprogramm, d.h.: nur ein Modul gegeben





Nächste Schritte

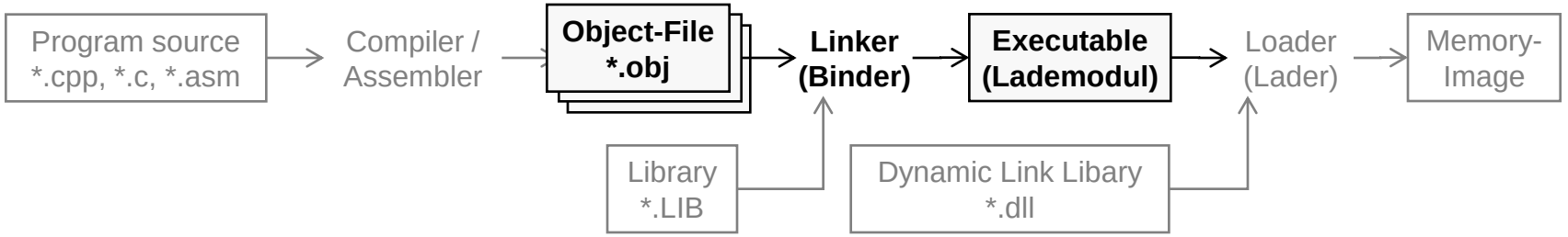
- **Normalerweise besteht ein Programm aus mehr als einem Objektmodul**
- **Jedes *.obj Modul ist das Ergebnis einer getrennten Übersetzung eines Quellcodes**
- **Spezielle Komponente der System-Software:**
  - **Linkage Editor (kurz: Linker)**
  - **Kombiniert all diese Module zu einem einzigen Modul, welches geladen und ausgeführt werden kann**

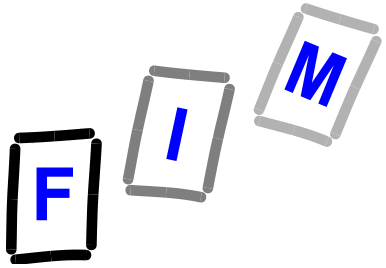


Aufgabe des Linker

- **Linkage Editor = LINKER**
- **Funktion:**
  - **Es gibt genau ein Hauptmodul**
  - **Bindet mehrere verschiebbare Objektmodule in ein Lade-Modul M zusammen**
  - **Adressen werden relativ zur Startadresse des Hauptmoduls gesetzt**
    - **Vor allem kann es Prozeduraufrufe zwischen den einzelnen Modulen geben**
  - **Damit ist Situation auf „nur 1 Modul“ zurückgeführt**

Linker Aufgabe





Output des Linker ist ein Lade-Modul M

Dieses Lade-Modul M enthält dann entweder …

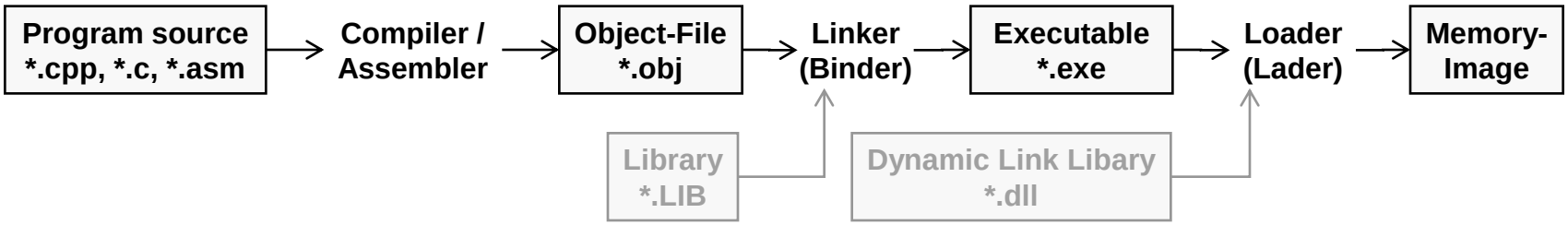
- **nur mehr absolute Adressen**

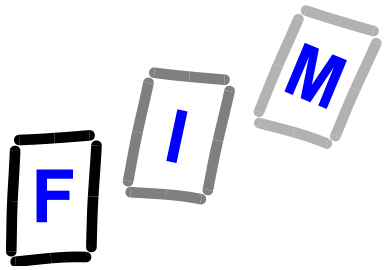
  Modul kann daher vom Lader einfach 1:1 geladen werden

- **oder alle Adressen sind relativ zum Ladepunkt des Lade-Moduls**

  M wird von einem Relokationslader (relocating loader) geladen, der die relativen Adressen auf absolute Adressen umsetzt

Weg vom Quellcode zur Ausführung: Gesamtbild

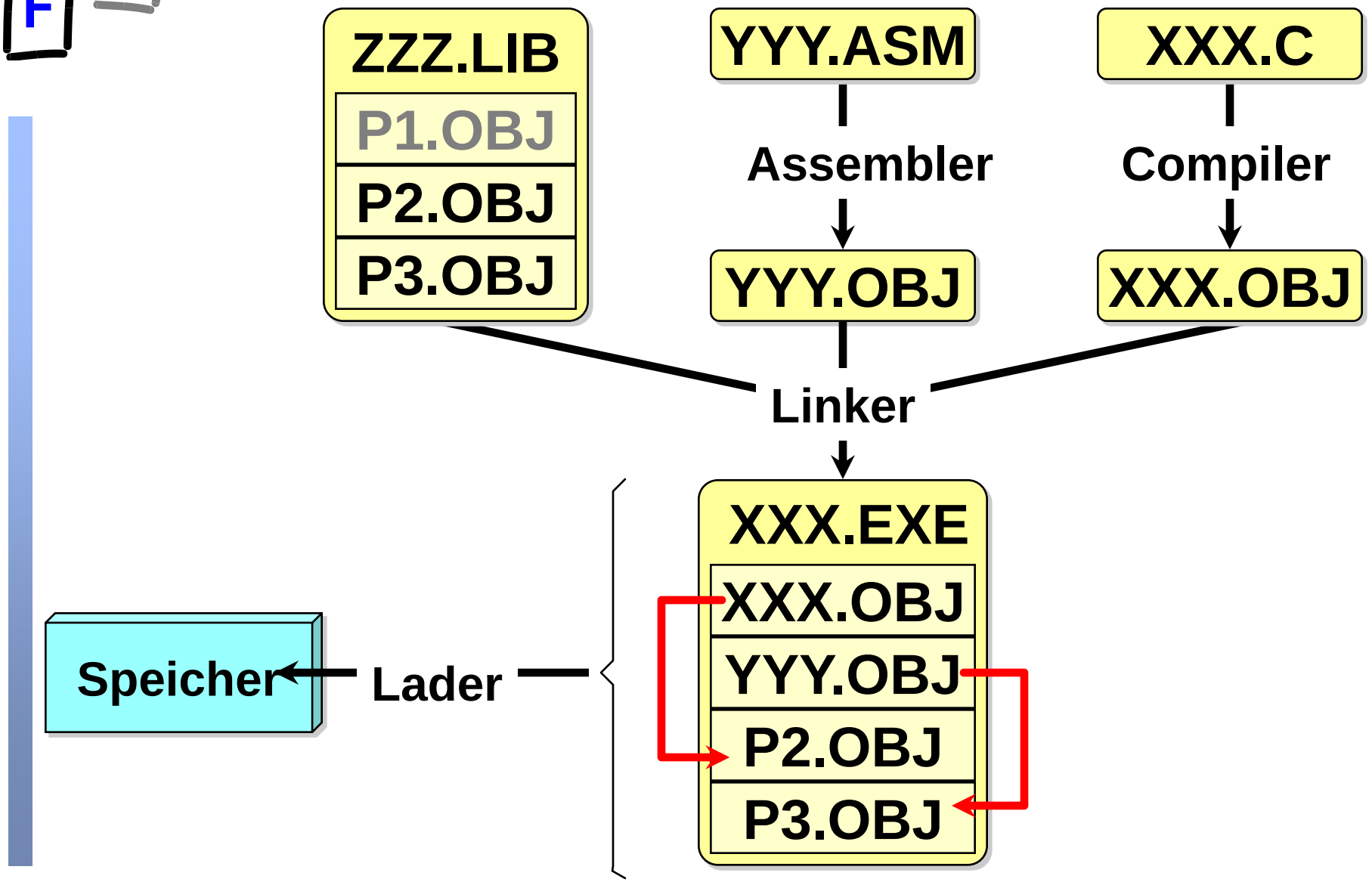




Statisches versus dynamisches Linken

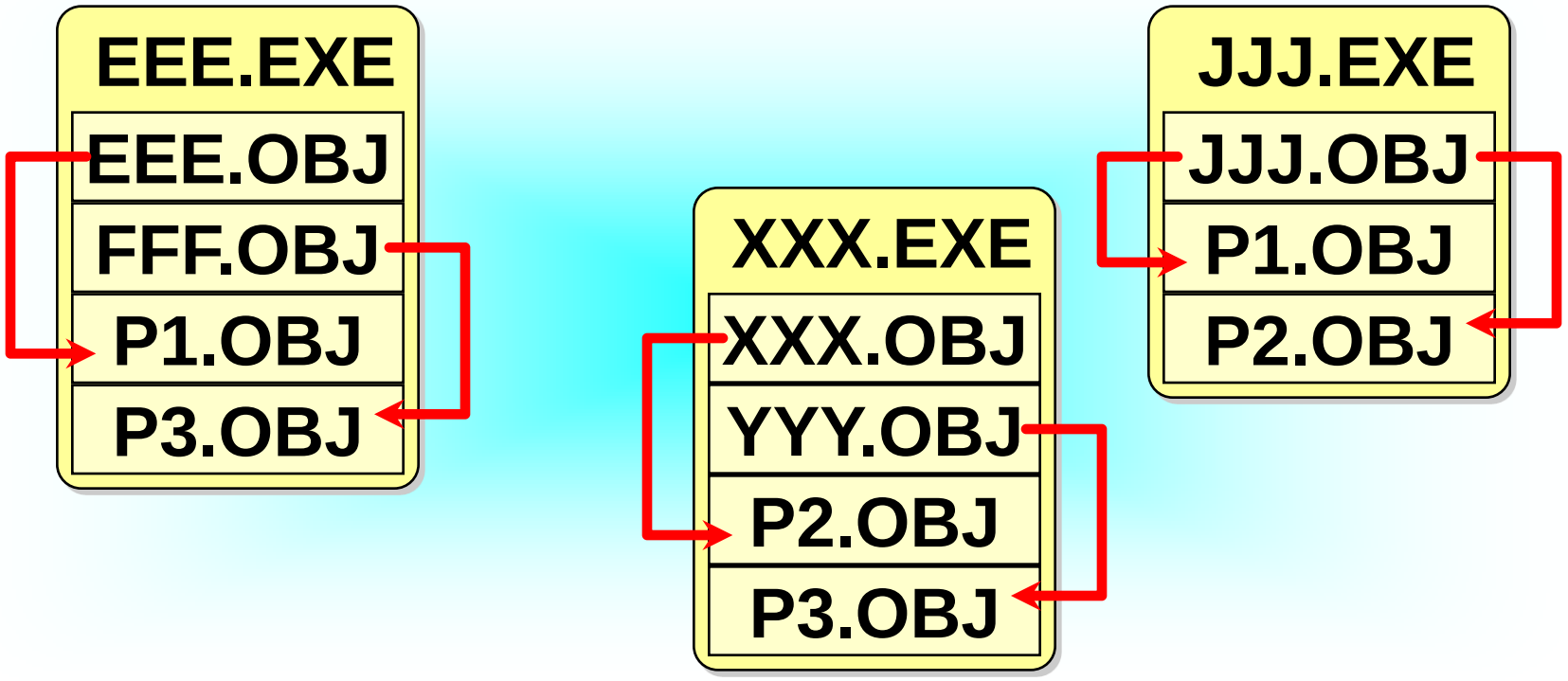
- **Zwei grundlegende Techniken**
  - **Statisches Linken (engl.: static linking)**
    - **Noch in Verwendung für kleine Systeme (zB.) ohne Multiprogramming**
    - **Trotzdem: Veraltet!**
  - **Dynamisches Linken (engl.: dynamic linking)**
    - **Siehe *dynamic link library (DLL)***
    - **Auch wichtig, um Komponenten des Betriebssystems einzubinden**
- **Wir werden das Linken zur Laufzeit (runtime linking) hier nicht besprechen**

Statisches Linken



Statisches Linken: Konsequenz, Nachteile

Mehrfach verwendete OBJ-Module finden sich mehrfach im Hauptspeicher



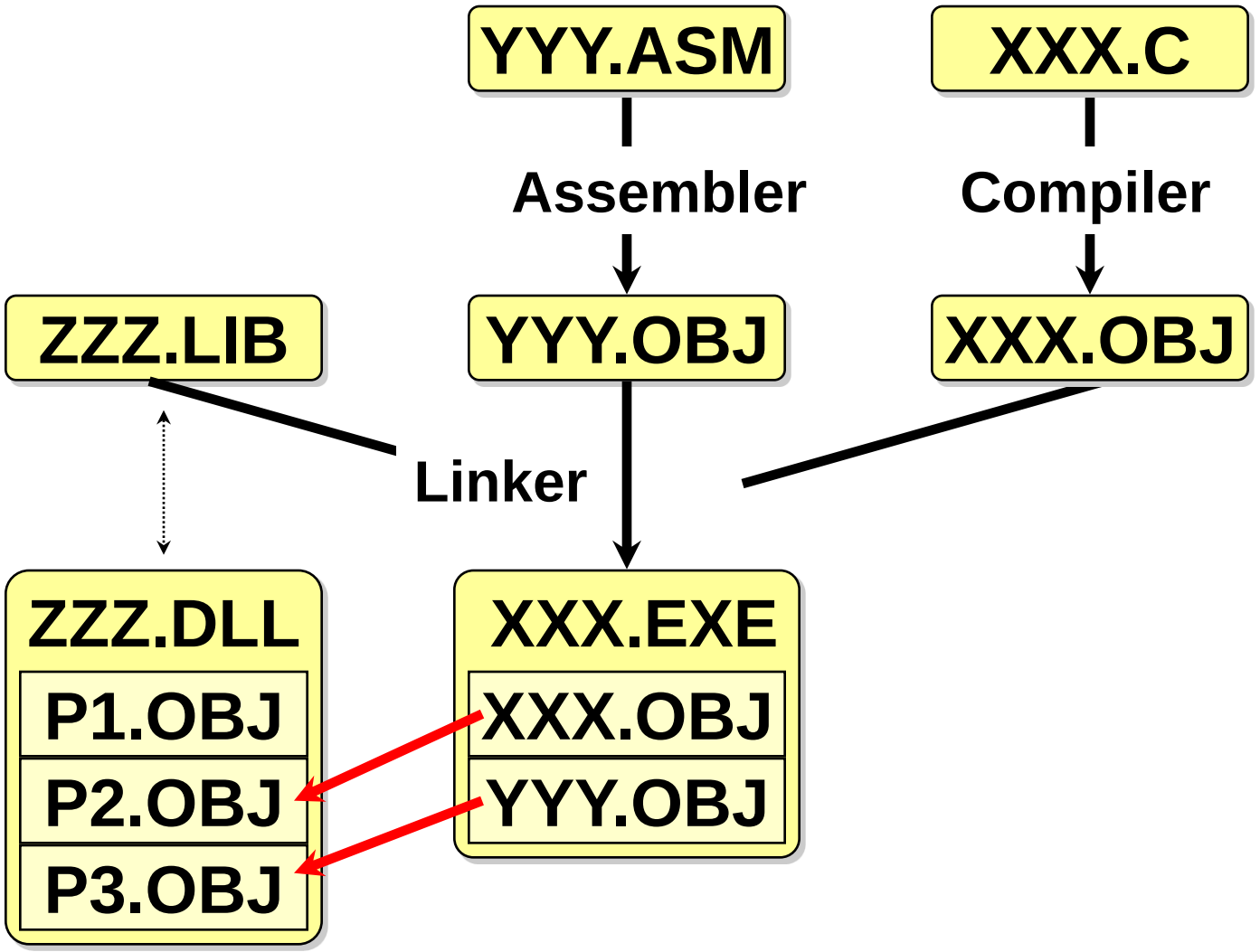
Dynamisches Linken

- Ähnliches Konzept wie dynamisches Laden
- Linken wird zeitlich hinausgeschoben
  - Entweder zur Ladezeit (Normalfall)
  - oder „Linken zur Laufzeit“
  - Falls ein DLL-Modul zur Link-/Laufzeit benötigt wird, prüft man, ob es sich nicht schon im Hauptspeicher befindet
    - DLL könnte schon von einem anderem Prozess geladen worden sein
- Linkage Editor bindet nur kleine „Tabellen/Verweise“ ein (engl.: stubs), mit denen man die DLL finden kann

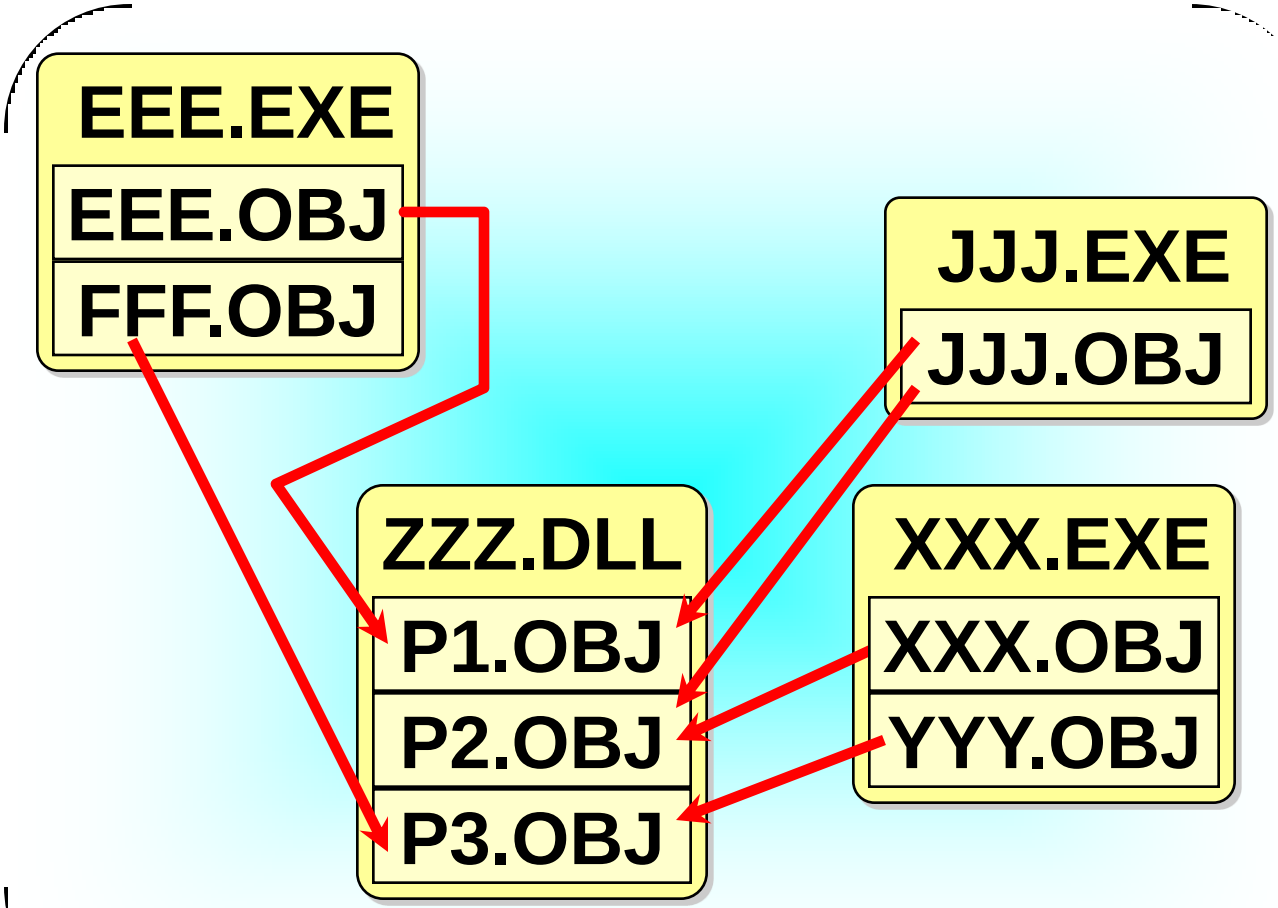
Dynamisches Linken "Stub"

- Ein „Stub“ ist ein kleines Stück Code, welches dem Verweis auf ein Bibliotheks-Programm (Modul) hinzugefügt wird
- Der Stub wird benutzt, um ein referenziertes, speicher-residentes Bibliotheks-Modul zu lokalisieren
- Der Stub wird zum Laden eines Bibliotheks-Moduls verwendet, falls es nicht vorhanden ist
- Vor allem wichtig für das Betriebssystem und seine eigenen Bibliotheken

Dynamisches Linken

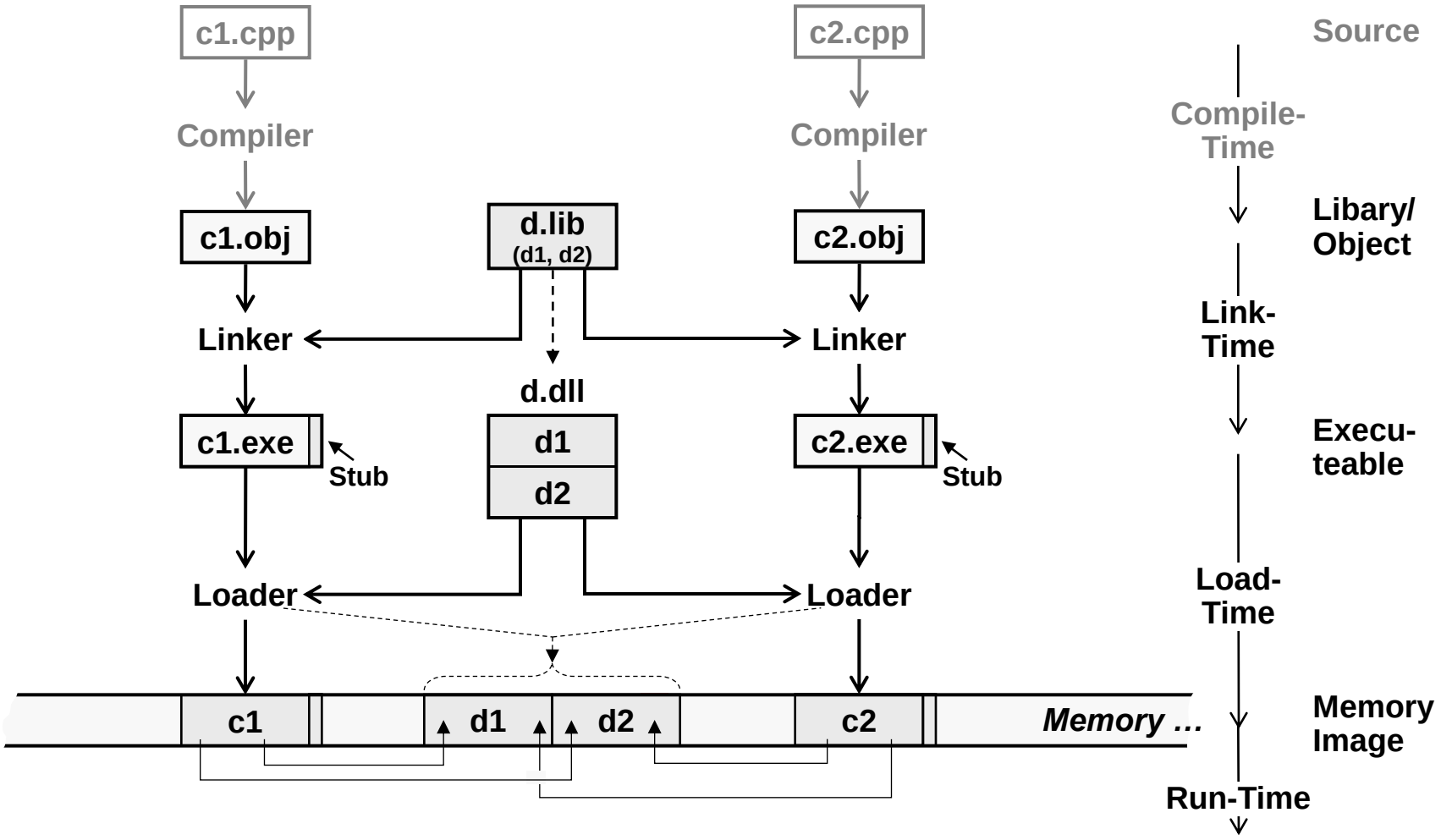


Dynamisches Linken: Vorteile



Mehrfach verwendete Bibliotheks-module stehen nur einmal im Speicher

Gesamtbild
Dynamisches Linken



ASLR: Adress Space Layout Randomization

- **Sicherheitsfeature: Programmcode ist bei jeder Ausführung an einer anderen Speicheradresse zu finden**
  - **Voraussetz.: Adresse wird frühestens zur Ladezeit bestimmt!**
  - **Welche Adresse genau: Zufallsgenerator**
    - **Praxis: Niedrige Bits der Zeit werden verwendet**
    - **Entropie hängt von Implementierung ab (und: 32/64 Bit!)**
    - **Beispiel Windows Vista: Offset wird bei Boot bestimmt (255 mögliche Werte; 8 Bit Entropie)**
- **ASLR: Randomisierung auch von anderen Elementen: Stack und Heap**
  - **Beispiel Vista: Stack wird zufällig platziert, Anfangswert von SP ebenso; Aber: Keine Gleichverteilung!**

http://www.blackhat.com/presentations/bh-dc-07/Whitehouse/Presentation/bh-dc-07-Whitehouse.pdf