OMSI-II

1. Praktikum zu SLX

Aufgabe 1

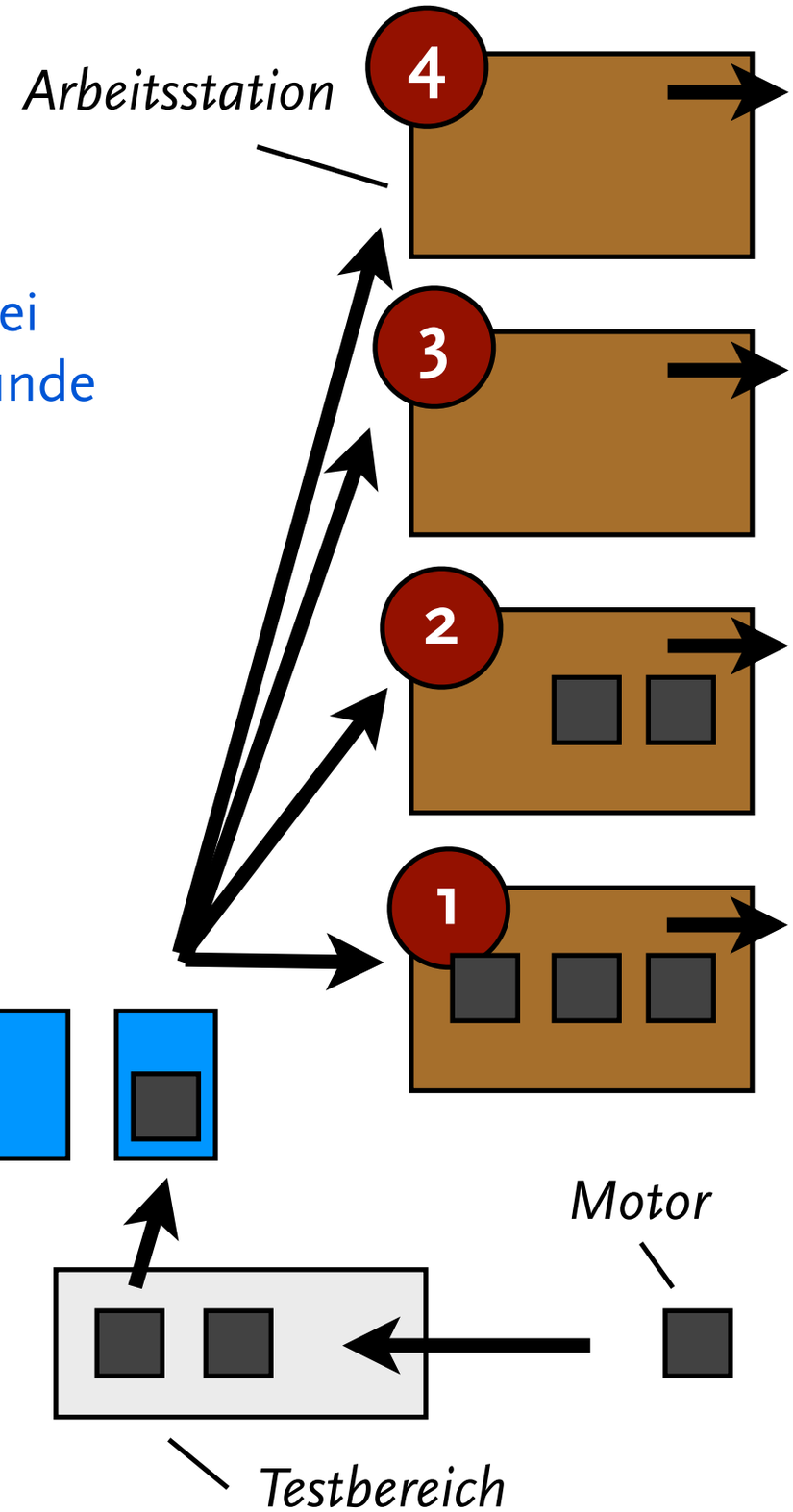
zu SLX

Aufgabe: Verpackungssystem #1

- **Verpackungssystem** bei einem Motorenhersteller
- **Motoren** erreichen das Ende eines **Testbereichs** und warten dort
  - Best. Zeitabstände mtba = 8
  - Warten in Reihenfolge der Ankunft
- Motoren werden dort von einem **Gabelstapler (GS)** aufgenommen
  - Best. Anzahl vorhanden (variierbar) ags = 1
  - Best. Fahrzeit + Aufnahme/Ablage 5 + Zielposition * 5 + 5
- Danach zu einer freien **Arbeitsstation (AS)** mit < 3 Motoren gefahren
  - Best. Verpackzeit mvt = 30
  - Best. Anzahl an Stationen aas = 4
  - Best. Ort von Stationen (Auswirkung auf Fahrzeit)
  - 3 Motoren werden zusammen an einer AS **verpackt**

mversand=0

Anfangswerte bei
Zeiteinheit: 1 Sekunde



Aufgabe: Verpackungssystem #1

- Hinweise & Vorgaben
  - GS stehen am Anfang beim Testbereich
  - GS fahren nach Entladung zurück zum Testbereich
  - Bei mehr als zwei Gabelstaplern könnte folgende Situation eintreten, die verhindert werden soll
    - 1. GS wählt eine AS und fährt los
    - 2. GS wählt die gleiche AS und fährt auch los
    - 1. GS kommt an, entlädt und die AS ist voll
    - 2. GS kommt an und muss warten

Aufgabe: Verpackungssystem #1

- Weitere Vorgaben
  - GS versuchen eine freie AS in der Reihenfolge AS 1 - 4 zu finden
  - Eine AS wird immer nur von genau einem GS angefahren. Erst wenn dieser GS den Motor abgeladen hat, kann die AS von einem weiteren GS angefahren werden.
  - Sind alle AS belegt (oder werden gerade angefahren), so müssen weitere GS solange warten bis eine AS verfügbar wird.
  - Der erste Motor trifft zum Zeitpunkt 8 Sekunden ein.
  - Ein Motor, der von einem GS aufgeladen oder transportiert wird, wartet nicht mehr im Testbereich und ist auch noch nicht versendet.

Aufgabe: Verpackungssystem #1

- Erstellen Sie ein SLX-Modell für das Verpackungssystem
- Führen Sie eine Simulation mit einer Dauer von einem Tag durch
- Bestimmen Sie Gesamtanzahl der verpackten Motoren und die Länge der Warteschlange im Testbereich am Ende der Simulation (als Ausgabe am Ende)
  - Validierung
    - Bei Dauer 1 Tag
      - Versendete Motoren: 3834
      - Motoren im Testbereich: 6960
    - Bei Dauer 1 Stunde
      - Versendete Motoren: 156
      - Motoren im Testbereich: 290
  - Wichtig: Der bestimmende Prozess muss mit geringerer Priorität als alle anderen ausgeführt werden (z.B. main).

Aufgabe: Verpackungssystem #1

- Erhöhen Sie die Anzahl der Gabelstapler und beobachten Sie die Auswirkungen
- Stellen Sie folgende Parameterwerte ein:
  mvt=120, mtba=4
  Führen Sie die Simulation erneut für einen Tag durch und notieren Sie das Ergebnis.
- Abgabe in 1-2er Gruppen
- Bearbeitungszeit 2 Wochen

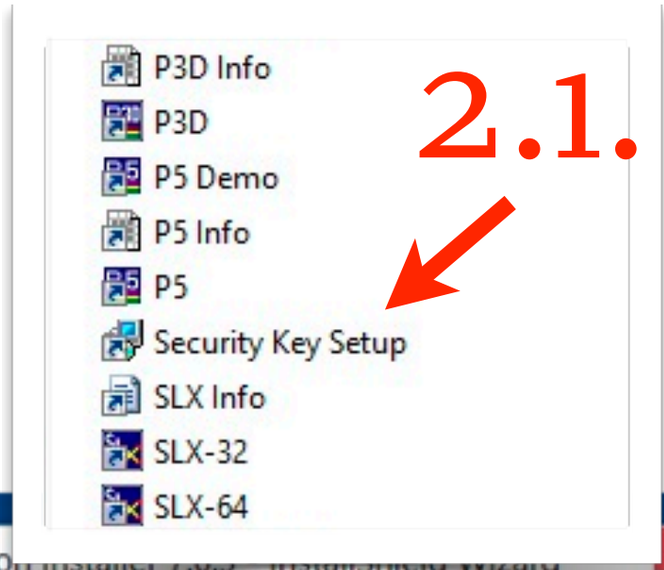
Hinweise zur Installation von SLX

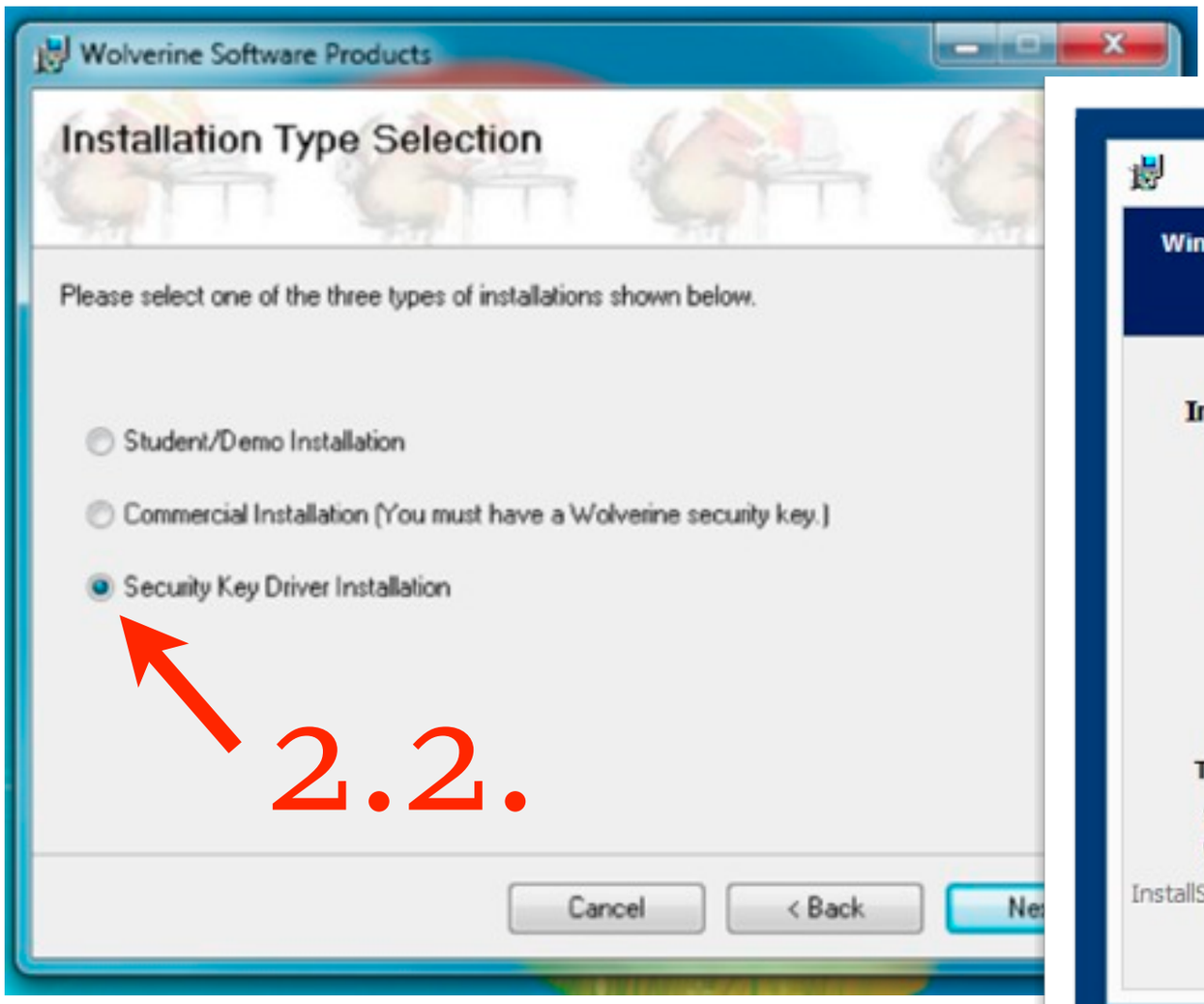
Versionen von SLX

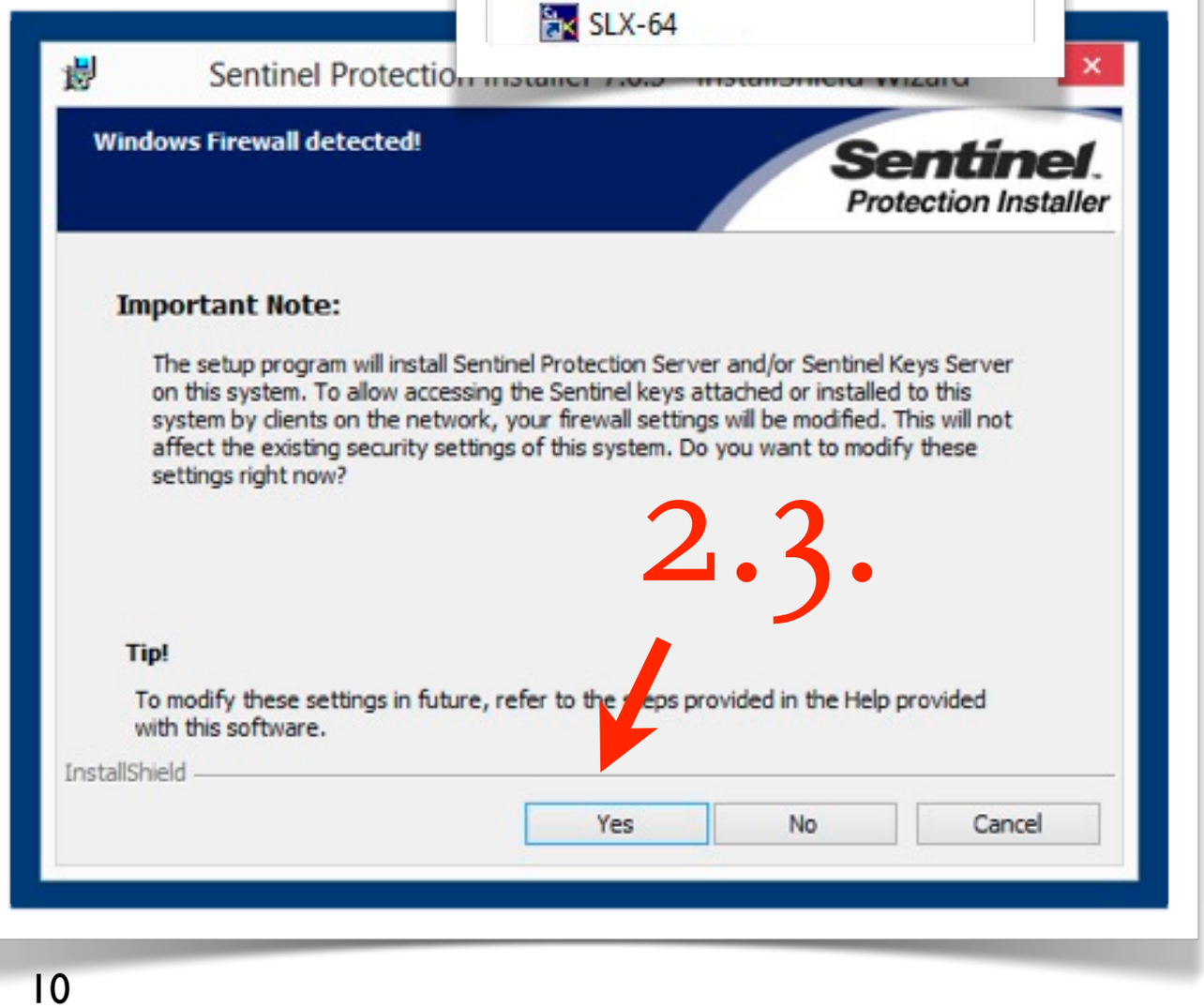
- Alternativ Studentenversion von SLX
  - Beschränkt in der Anzahl von Objekten in einer Simulation
    - max. 450 Objekte (aktive + passive)
    - max. 500 Pucks
- Akademische Version
  - SLX unbeschränkt
  - Netzwerklizenz mit max. 10 gleichzeitig aktiven Rechnern

Installation von SLX-Academic

1. **Baseline** installieren
2. *Security Key Setup* starten







Installation von SLX-Academic

3. Rechner neu starten

4. Verbindung zum internen Netzwerk der HU über eine VPN-Verbindung herstellen

   - siehe https://www2.informatik.hu-berlin.de/rbg/Openvpn_SSL/index.shtml

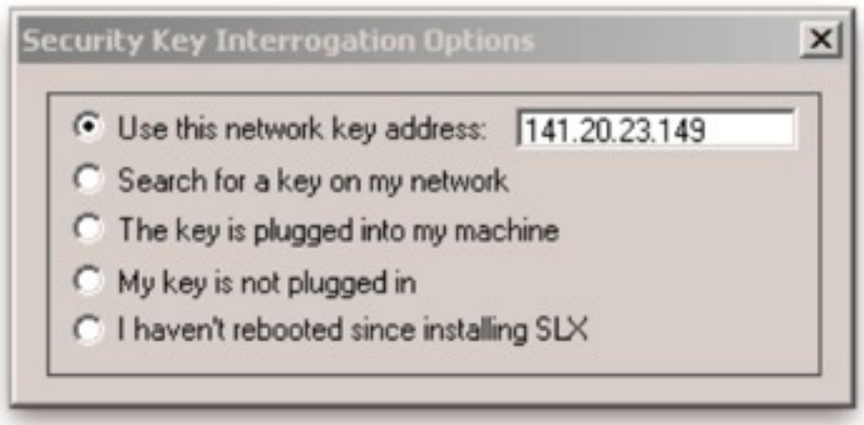
   - Unter Windows 7+: VPN-Client als Admin starten

Installation von SLX-Academic

5. SLX starten

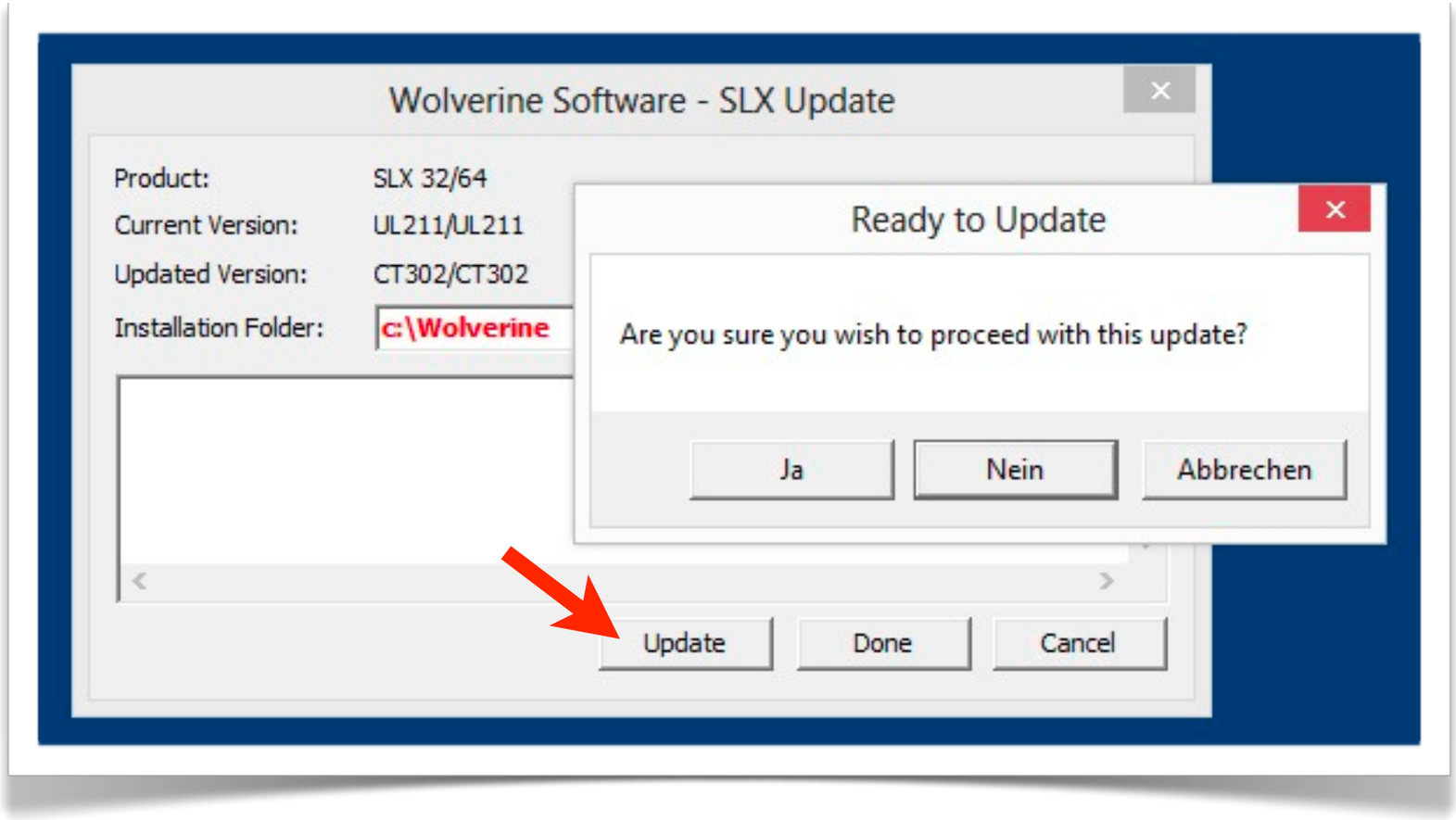
- *Use this network key address* wählen
- Adresse des Lizenz-Servers eingeben:

  141.20.23.149



Installation von SLX-Academic

6. **Update** installieren



Hinweise zu SLX

Lokale Variablen in SLX

Eigenartiger SLX-Bug

```
passive class A {
        int i = 2;
}
```

```
pointer(A) a;
a = new A();
int i = a->i;
```
✘
•• Execution error at time 0: NULL pointer reference

```
pointer(A) a;
a = new A();
int i;
i = a->i;
```
✔

```
pointer(A) a = new A();
int i = a->i;
```
✔

sieht aus wie Fehler im Compiler

Lokale Variablen in C++

- Jeder Block { ... } öffnet einen neuen lokalen Gültigkeitsbereich für Variablen
- Variablen werden erst bei ihrer Deklaration angelegt

```
ctest — bash — 25×15
begin loop
init:0
0
init:1
1
init:2
2
begin loop
init:3
3
init:4
4
init:5
5
picard:ctest blunk$
```

stack.cpp (~/Temp/ctest) - VIM

```
#include <iostream>

using namespace std;

int bcount=0;

class B {
public:
        int id;
        B() {
                id = bcount;
                bcount++;
                cout << "init:" << id << endl;
        }
        void print_id() {
                cout << id << endl;
        }
};

void p() {
        int i=0;
        cout << "begin loop" << endl;
        while (i < 3) {
                B b;
                b.print_id();
                i++;
        }
}

int main() {
        p();
        p();
        return 0;
}
```

Lokale Variablen in SLX

- Lokale Gültigkeitsbereiche für Variablen gibt es nicht (auch nicht in Schleifen)
- Alle (auch später) deklarierten Variablen werden am Anfang einer Prozedur einmalig angelegt, sind aber erst nach ihrer Deklaration namentlich zugreifbar
- Vermutlich Schwäche des Compilers
- Compiler sollte erzwingen, dass Variablen nur am Anfang einer Prozedur deklariert werden dürfen

```
Execution begins
init:  0
begin loop
0
0
0
init:  1
begin loop
1
1
1
Execution complete
```

```
int bcount = 0;

class B {
    int id;
    initial {
        id = bcount;
        bcount++;
        print(id) "init:_\n";
    }
    procedure print_id() {
        print(id) "_\n";
    }
}

procedure p() {
    int i;
    print "begin loop\n";
    while (i < 3) {
        B b;
        b.print_id();
        i++;
    }
}

procedure main() {
    p();
    p();
}
```

Lokale Variablen in SLX

```
class A {
   int id;
   static int count;

   initial {
      count++;
      id = count;
   }

   actions {
      print(id) "_ terminates.\n";
   }
}

procedure main() {
   int i;
   for (i=0; i<3; i++) {
      pointer(A) a = new A;
      activate a;
   }
   yield;
}
```

```
Execution begins
1   terminates.
1   terminates.
1   terminates.
Execution complete
```

```
Execution begins
1   terminates.
2   terminates.
3   terminates.
Execution complete
```

```
procedure main() {
   int i;
   pointer(A) a;
   for (i=0; i<3; i++) {
      a = new A;
      activate a;
   }
   yield;
}
```

Problem der „illegal forward reference“

Lsg. A

```
class A {
    actions {
        place new B() into bs;
        b = new B();
•• Semantic error: illegal forward reference to "b" (defined below in
"verpackungssystem")
        i = 2;
•• Semantic error: illegal forward reference to "i" (defined below in
"verpackungssystem")
        p1(3);
    }
}

set(B) bs;
pointer(B) b;
int i;

procedure p1(int x) {}

class B {
    int j;
}

...
```

pointer(T)

```
class A {
    int i;
}

procedure main() {
    A a1; // Wert der Variablen ist ein A-Objekt
    A a2;

    pointer(A) pa; // Wert der Variablen ist die Adresse eines A-Objektes
    pa = &a1; // &-Operator zum Zugriff auf die Adresse

    a1.i = 1;
    (*pa).i = 1; // *-Operator zum Zugriff auf das Objekt an der Adresse
                 // (und nicht die Adresse selbst)
    pa->i = 1;// ->-Operator zur Vereinfachung

    pa = new A(); // dynamische Objekterzeugung per new, Ergebnis ist Adresse

    // Anwendung z.B. bei sets
    set(A) as;
    place pa into as;
    place &a2 into as;
    // Sets enthalten nur Adressen.
    // Sinnvoll, da Objekte sonst nur in genau einem set enthalten sein könnten
    // und von keinem anderen Objekt referenzierbar wären.
}
```

pointer(*)

```
class A {
    int i;
}

class B {
}

procedure main() {
    A a1;
    A a2;

    pointer(*) p;

    p = &a1;
    p->i = 2;
    p->m = 2;
•• Semantic error: "m" is undefined

    B b;
    p = &b;
    p->i = 2;

    p = &a1;
    pointer(A) pa;
    pa = (pointer(A)) p;
    pa->i = 2;
}
```

in SLX erlaubt, aber unsicher zur Laufzeit

```
p->i = 2;
•• Execution error at time 0: "p" points to
an object of class B, which has no "i"
```

besser den Typ überprüfen

```
p = &a1;
pointer(A) pa;
if (type(p) == type A) {
    pa = (pointer(A)) p;
    pa->i = 2;
}
```