Speicherverwaltung und Datenstrukturen

C - Kurs 2009

Florian Streibelt

freitagsrunde.org

22.09.2009

Inhaltsverzeichniss

Inhaltsverzeichniss

Wiederholung

- Arrays
- Pointer
- Pointerarithmetik
- Strings

Inhaltsverzeichniss

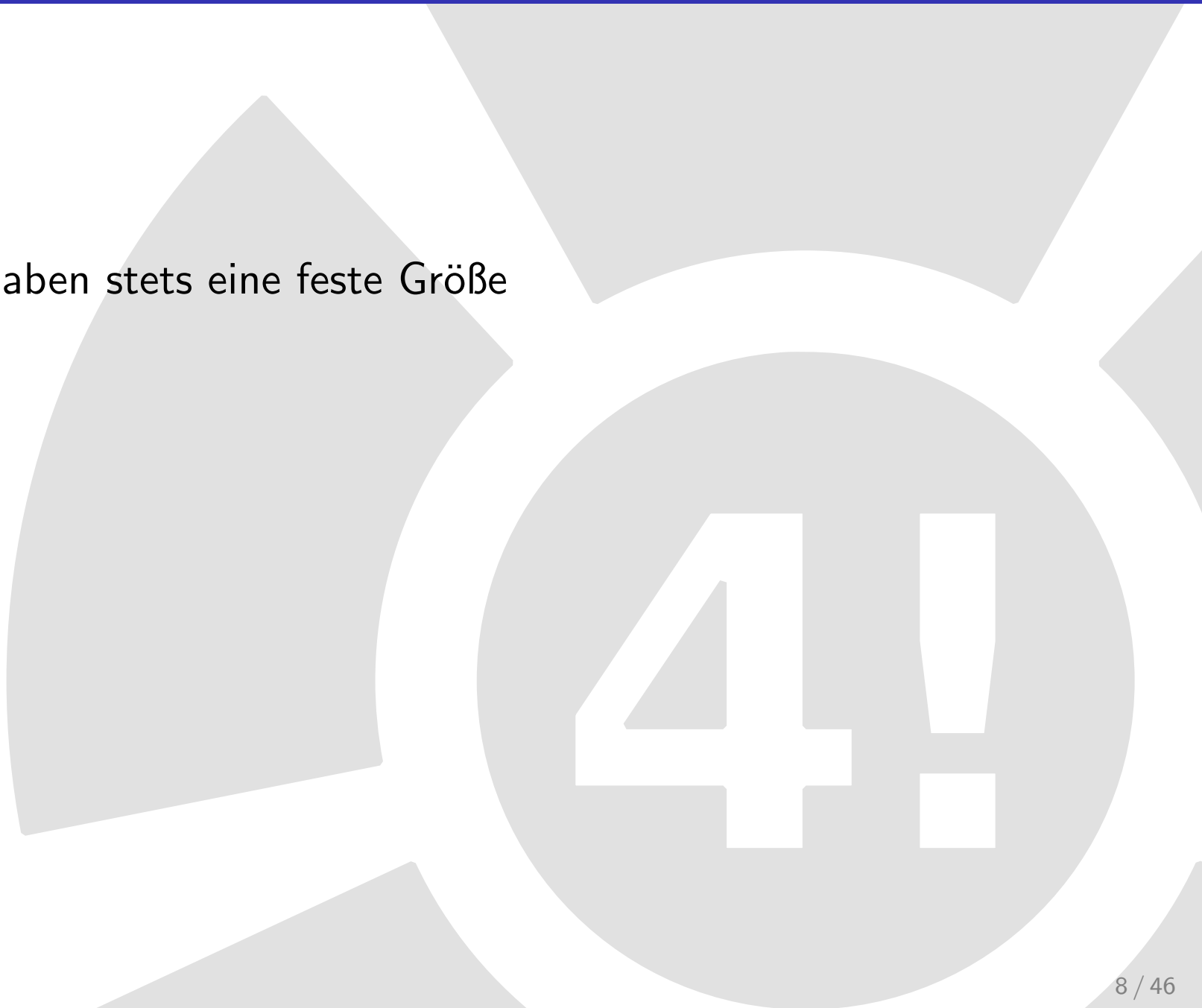
Vorschau

- dynamische Speicherbelegung
- eigene Datentypen
- dynamische Datenstrukturen
- ein Beispielprogramm, dass das alles nutzt
- Ausblick auf morgen

Inhaltsverzeichniss

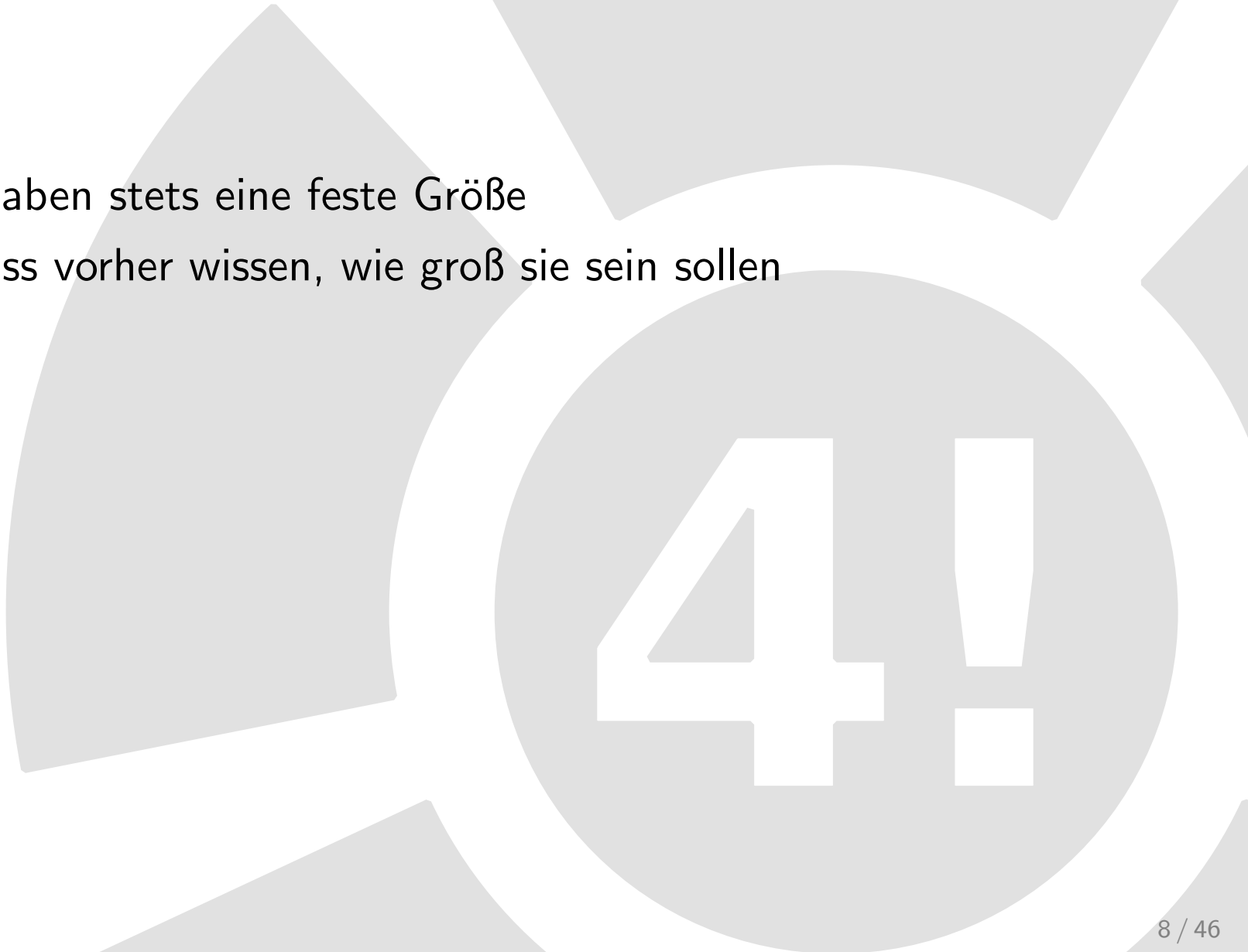
Arrays sind unflexibel

- Arrays haben stets eine feste Größe



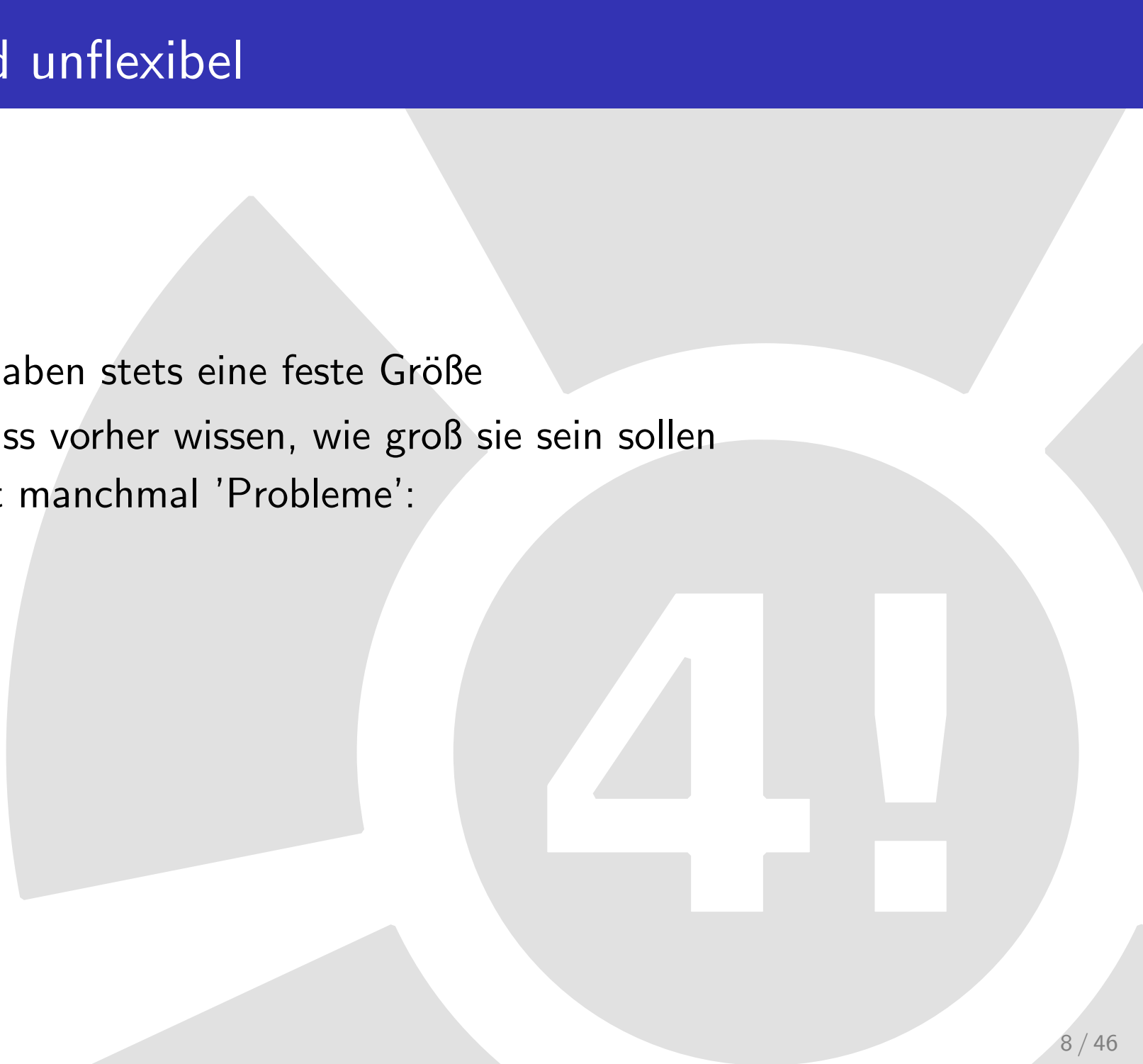
Arrays sind unflexibel

- Arrays haben stets eine feste Größe
- Man muss vorher wissen, wie groß sie sein sollen



Arrays sind unflexibel

- Arrays haben stets eine feste Größe
- Man muss vorher wissen, wie groß sie sein sollen
- Das gibt manchmal 'Probleme':



Arrays sind unflexibel

- Arrays haben stets eine feste Größe
- Man muss vorher wissen, wie groß sie sein sollen
- Das gibt manchmal 'Probleme':
  - Supermarktkasse: Wieviele Produkte?

Arrays sind unflexibel

- Arrays haben stets eine feste Größe
- Man muss vorher wissen, wie groß sie sein sollen
- Das gibt manchmal 'Probleme':
  - Supermarktkasse: Wieviele Produkte?
  - Messwerterfassung: Daten wachsen ständig.

Arrays sind unflexibel

- Arrays haben stets eine feste Größe
- Man muss vorher wissen, wie groß sie sein sollen
- Das gibt manchmal 'Probleme':
  - Supermarktkasse: Wieviele Produkte?
  - Messwerterfassung: Daten wachsen ständig.
  - ...

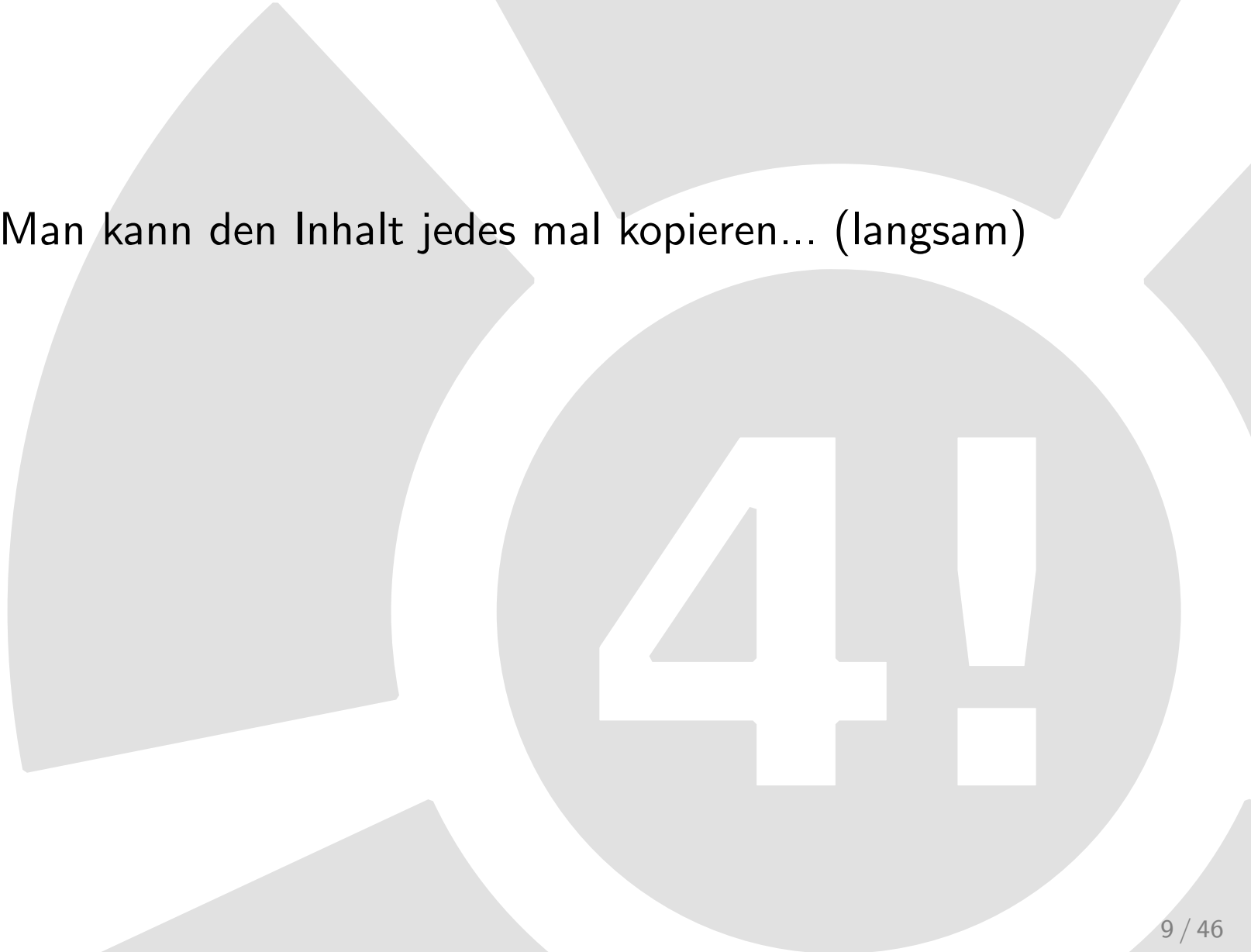
Arrays sind unflexibel

- Arrays haben stets eine feste Größe
- Man muss vorher wissen, wie groß sie sein sollen
- Das gibt manchmal 'Probleme':
  - Supermarktkasse: Wieviele Produkte?
  - Messwerterfassung: Daten wachsen ständig.
  - ...

Wie könnte man diese Probleme umschiffen?

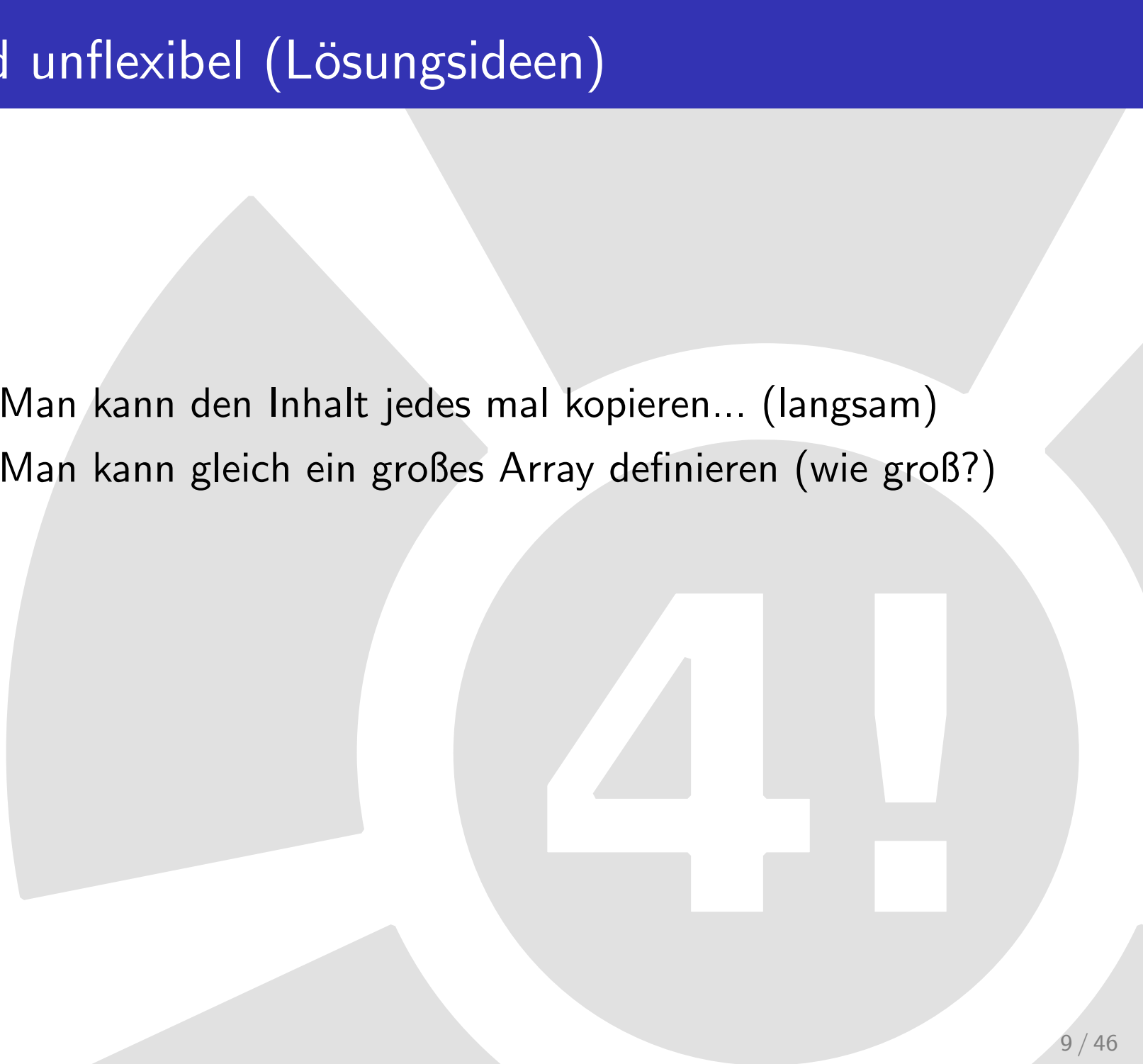
Arrays sind unflexibel (Lösungsideen)

- 1. Idee: Man kann den Inhalt jedes mal kopieren... (langsam)



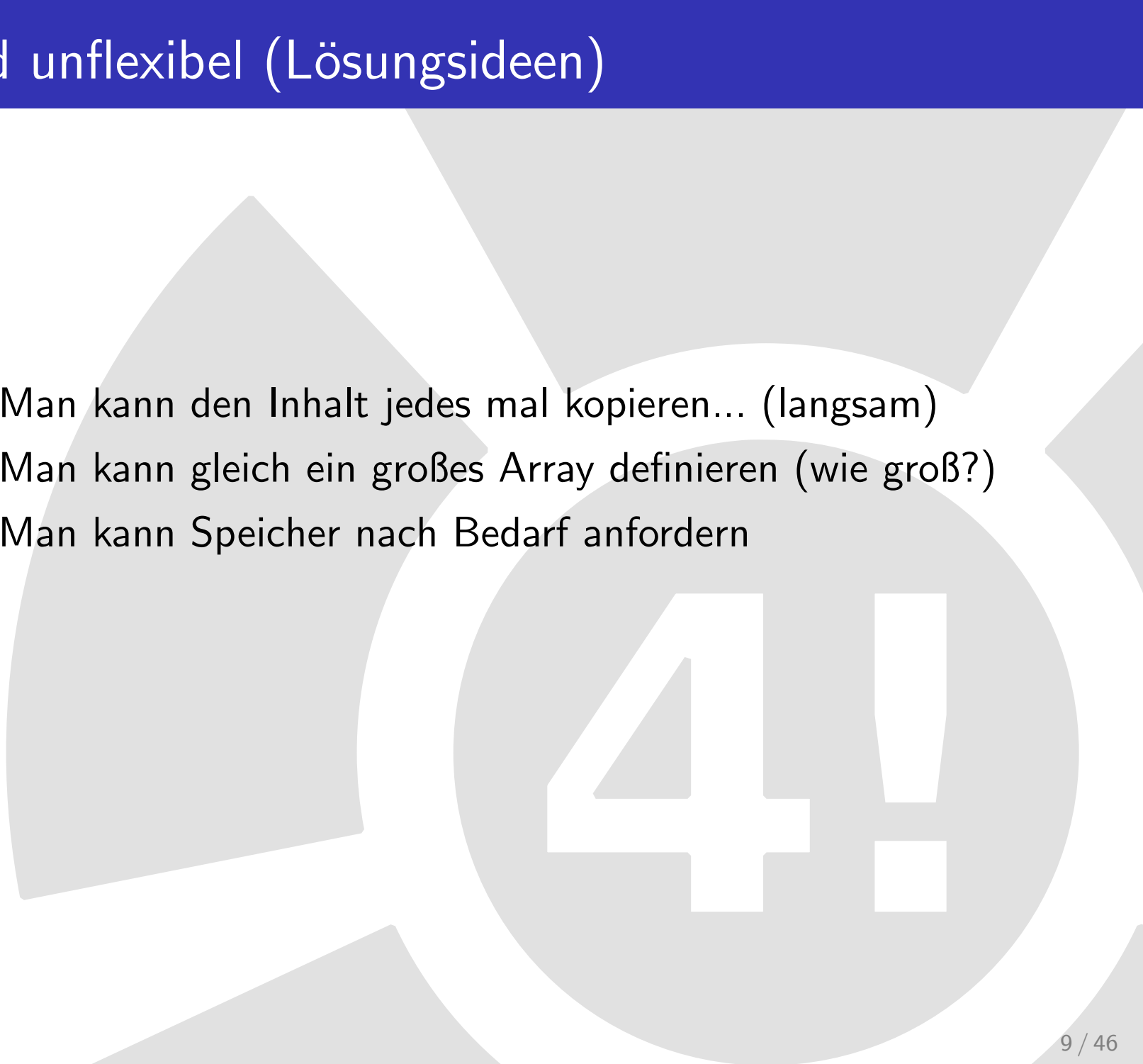
Arrays sind unflexibel (Lösungsideen)

- 1. Idee: Man kann den Inhalt jedes mal kopieren... (langsam)
- 2. Idee: Man kann gleich ein großes Array definieren (wie groß?)



Arrays sind unflexibel (Lösungsideen)

- 1. Idee: Man kann den Inhalt jedes mal kopieren... (langsam)
- 2. Idee: Man kann gleich ein großes Array definieren (wie groß?)
- 3. Idee: Man kann Speicher nach Bedarf anfordern



Arrays sind unflexibel (Lösungsideen)

- 1. Idee: Man kann den Inhalt jedes mal kopieren... (langsam)
- 2. Idee: Man kann gleich ein großes Array definieren (wie groß?)
- 3. Idee: Man kann Speicher nach Bedarf anfordern

Welches von den Mitteln aus dem Kurs könnte hier eine Basis bieten?

Arrays sind unflexibel (Lösungsideen)

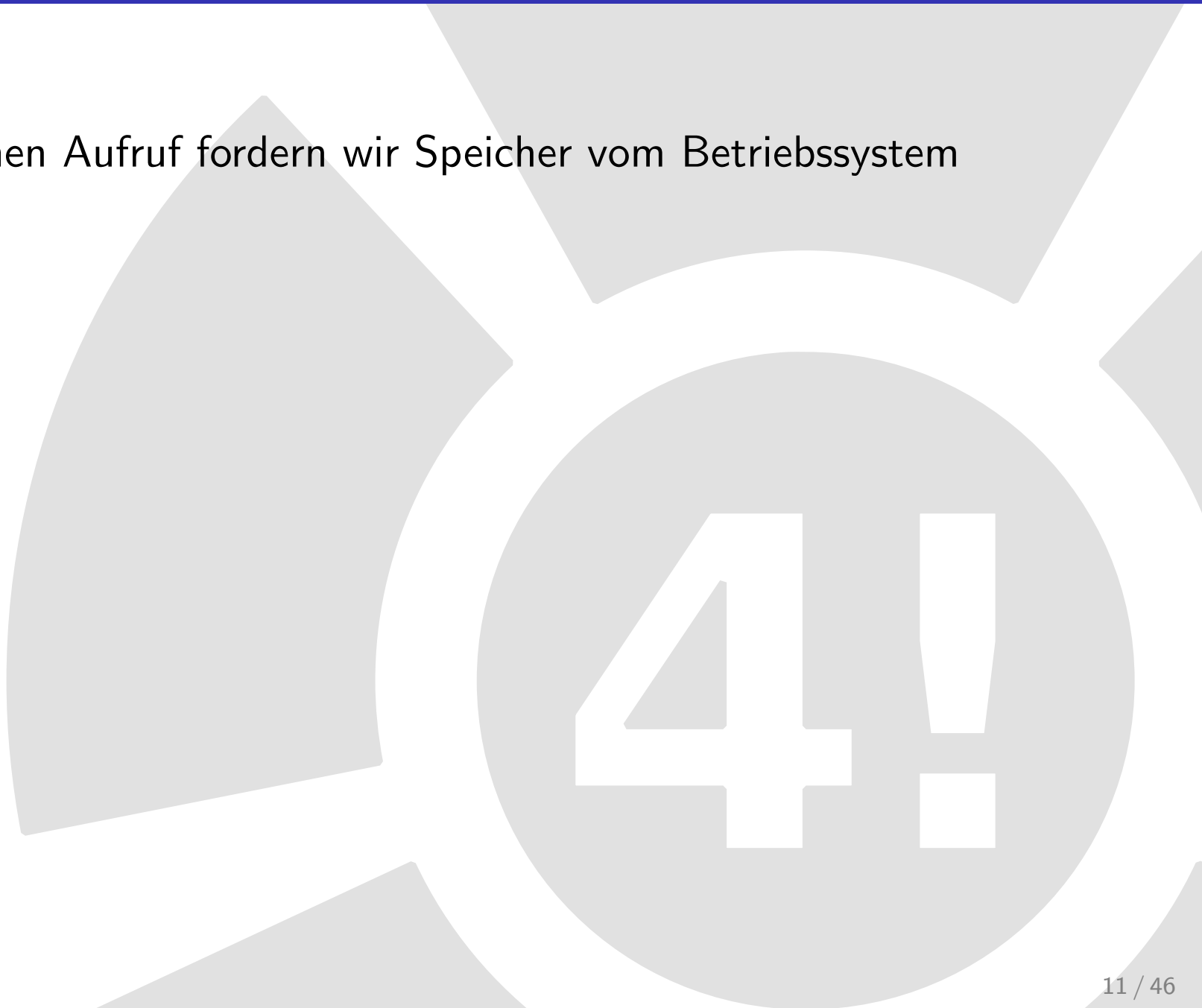
- 1. Idee: Man kann den Inhalt jedes mal kopieren... (langsam)
- 2. Idee: Man kann gleich ein großes Array definieren (wie groß?)
- 3. Idee: Man kann Speicher nach Bedarf anfordern

Welches von den Mitteln aus dem Kurs könnte hier eine Basis bieten?
$\Rightarrow$ Pointer: Sie zeigen auf Speicherbereiche und sind veränderlich!

Inhaltsverzeichniss

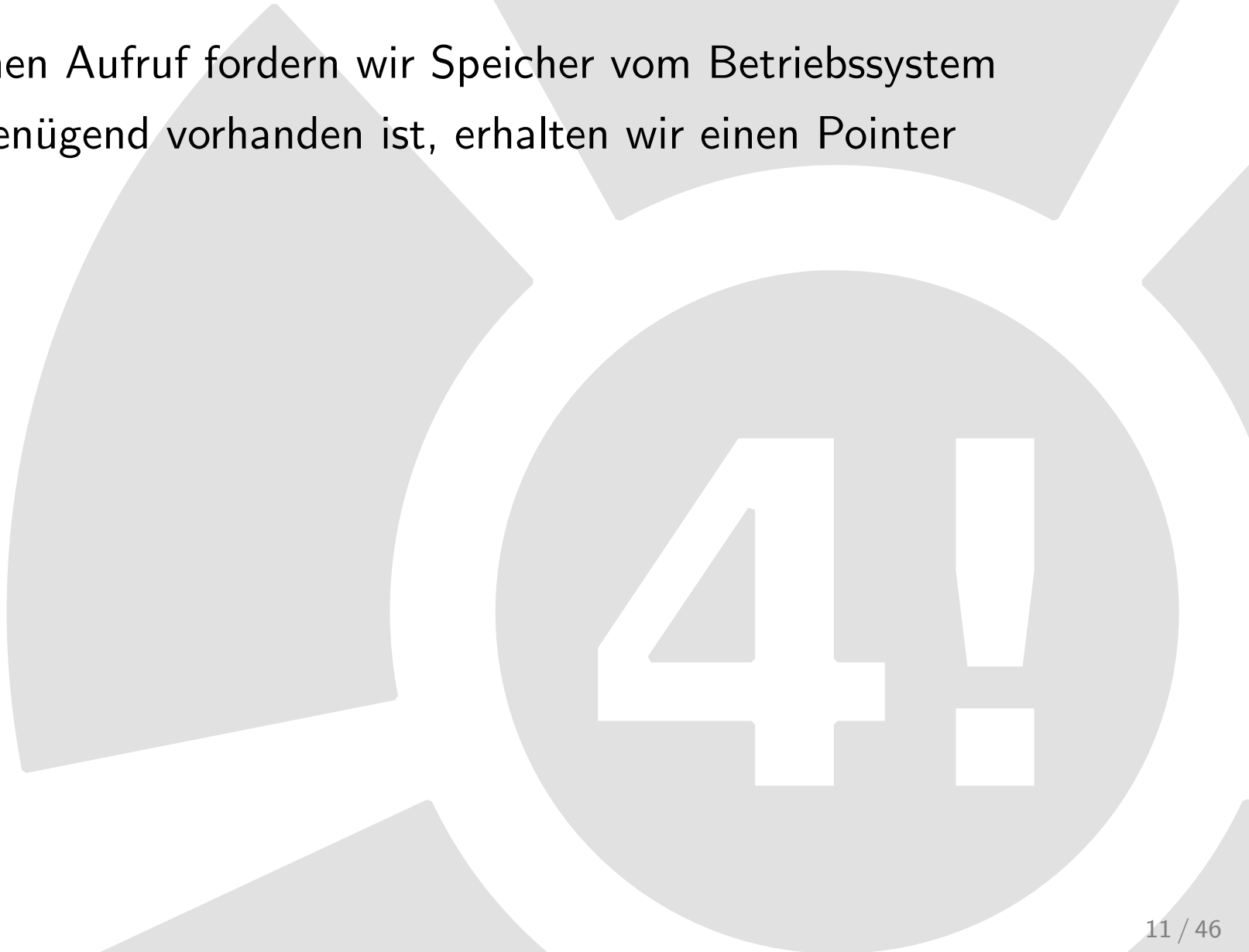
Dynamischer Speicher

- Über einen Aufruf fordern wir Speicher vom Betriebssystem



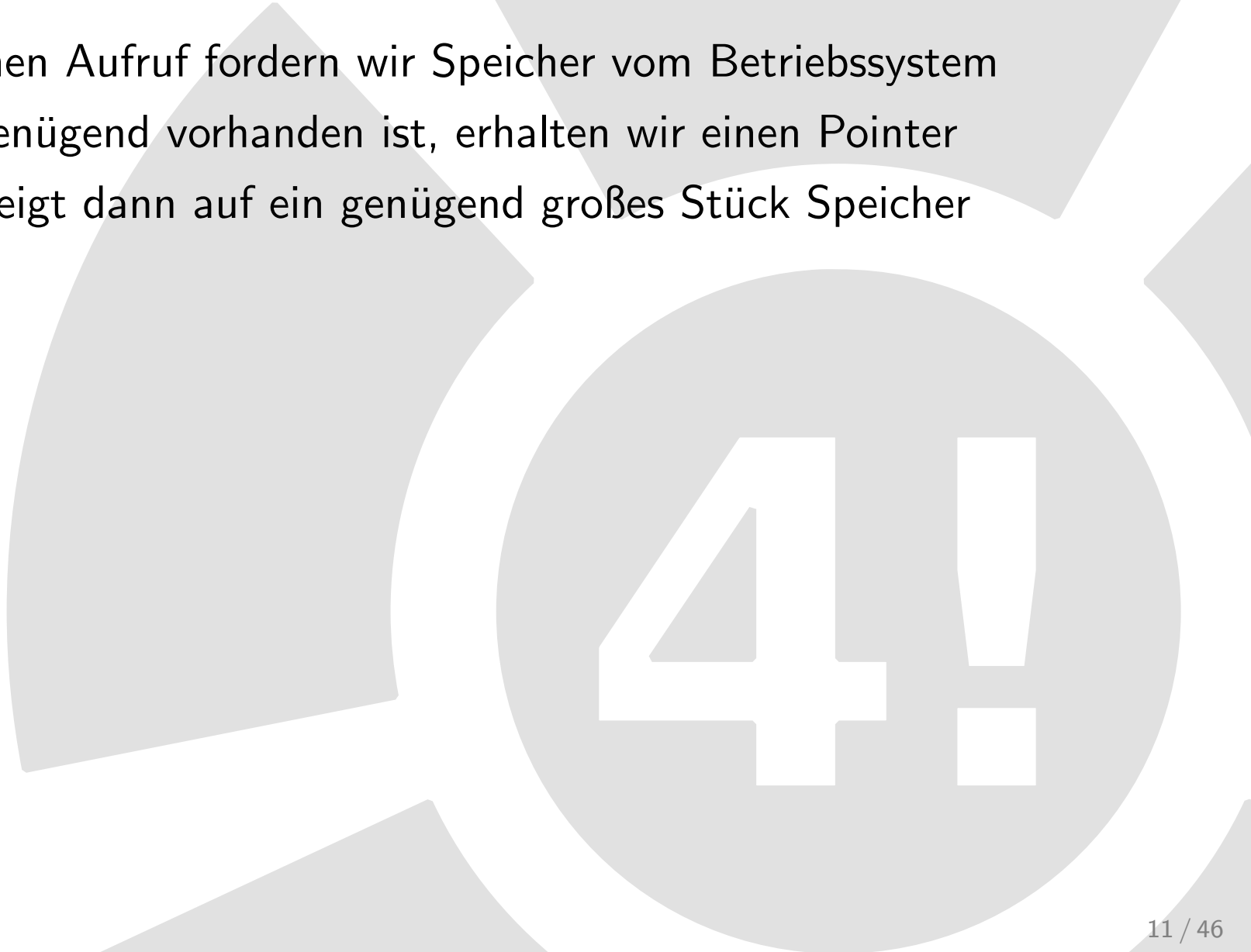
Dynamischer Speicher

- Über einen Aufruf fordern wir Speicher vom Betriebssystem
- Wenn genügend vorhanden ist, erhalten wir einen Pointer



Dynamischer Speicher

- Über einen Aufruf fordern wir Speicher vom Betriebssystem
- Wenn genügend vorhanden ist, erhalten wir einen Pointer
- Dieser zeigt dann auf ein genügend großes Stück Speicher



Dynamischer Speicher

- Über einen Aufruf fordern wir Speicher vom Betriebssystem
- Wenn genügend vorhanden ist, erhalten wir einen Pointer
- Dieser zeigt dann auf ein genügend großes Stück Speicher

Beispiel: Speicher anfordern

```
1 [...]
2 char *foo = (char *) malloc(1024); // cast to char*
3 printf("Contents: %s\n", foo);     // print content
```

Dynamischer Speicher

- Über einen Aufruf fordern wir Speicher vom Betriebssystem
- Wenn genügend vorhanden ist, erhalten wir einen Pointer
- Dieser zeigt dann auf ein genügend großes Stück Speicher

Beispiel: Speicher anfordern

```
1 [...]
2 char *foo = (char *) malloc(1024); // cast to char*
3 printf("Contents: %s\n", foo);     // print content
```

```
1 Contents: (&DS(=*'V+PadsK...')+)
```

... nur kann in dem Speicher noch etwas drinstehen.

malloc explained

Beispiel: Speicher anfordern

```
1 [...]
2 char *foo;
3 foo = (char *) malloc(1024); // cast to char*
```

malloc explained

Beispiel: Speicher anfordern

```
1 [...]
2 char *foo;
3 foo = (char *) malloc(1024); // cast to char*
```

- Einen Pointer 'foo' anlegen

malloc explained

Beispiel: Speicher anfordern

```
1 [...]
2 char *foo;
3 foo = (char *) malloc(1024); // cast to char*
```

- Einen Pointer 'foo' anlegen
- 1024 byte Speicher anfordern

malloc explained

Beispiel: Speicher anfordern

```
1 [...]
2 char *foo;
3 foo = (char *) malloc(1024); // cast to char*
```

- Einen Pointer 'foo' anlegen
- 1024 byte Speicher anfordern
- Speicher auf Zieltyp des Pointers casten

calloc explained

Beispiel: Nullinitialisierten Speicher anfordern

```
1 [...]
2 char *foo;
3 foo = (char *) calloc(1024, sizeof(char));
```

---

calloc explained

Beispiel: Nullinitialisierten Speicher anfordern

```
1 [...]
2 char *foo;
3 foo = (char *) calloc(1024, sizeof(char));
```

- Einen Pointer 'foo' anlegen

calloc explained

Beispiel: Nullinitialisierten Speicher anfordern

```
1 [...]
2 char *foo;
3 foo = (char *) calloc(1024, sizeof(char));
```

- Einen Pointer 'foo' anlegen
- 1024 Elemente der Größe von char anfordern

calloc explained

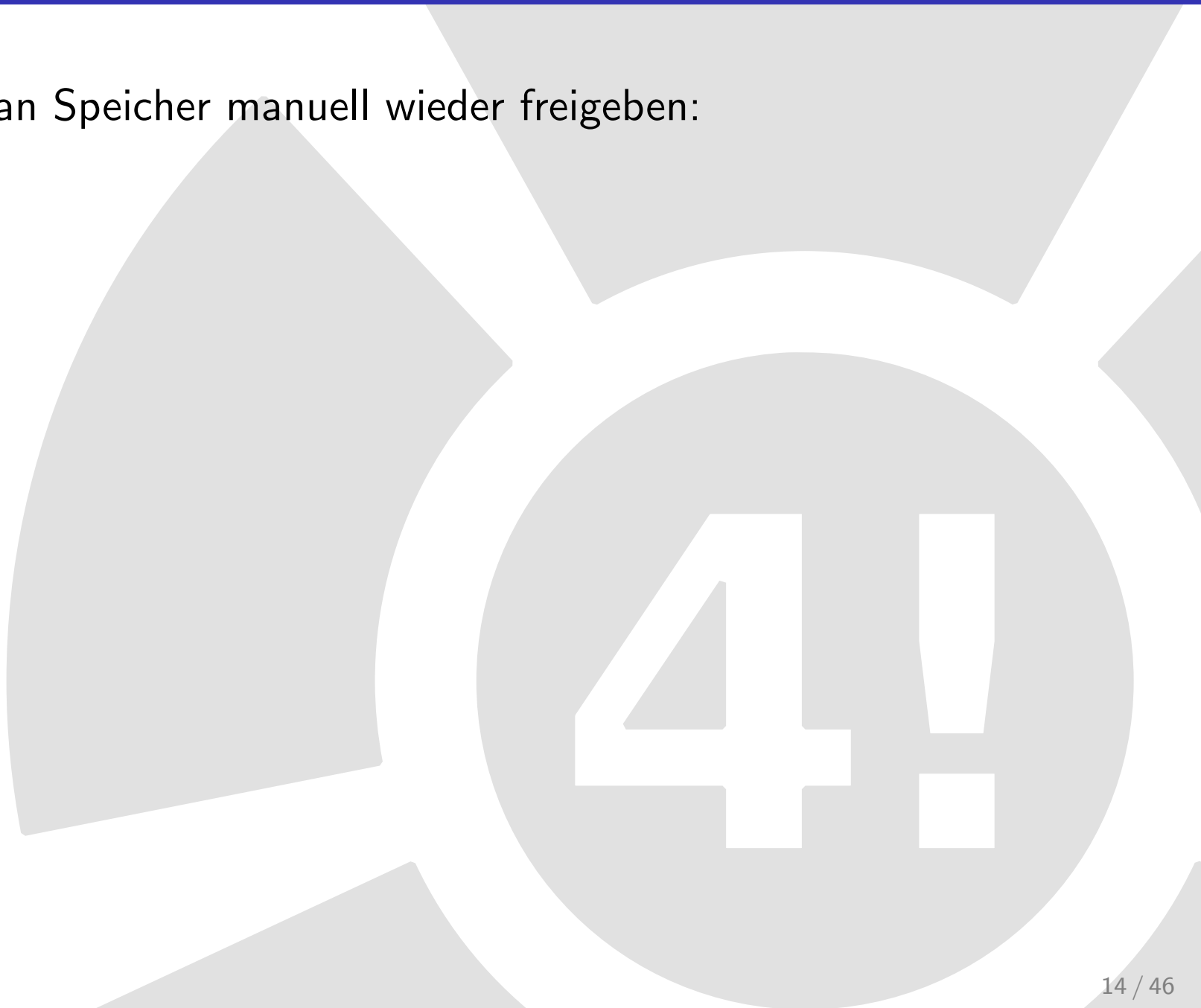
Beispiel: Nullinitialisierten Speicher anfordern

```
1 [...]
2 char *foo;
3 foo = (char *) calloc(1024, sizeof(char));
```

- Einen Pointer 'foo' anlegen
- 1024 Elemente der Größe von char anfordern
- Speicher auf Zieltyp des Pointers casten

free nicht vergessen

In C muss man Speicher manuell wieder freigeben:



free nicht vergessen

In C muss man Speicher manuell wieder freigeben:

Beispiel: Aufruf von free()

```
1 [...]
2 char *foo;
3 foo = (char *) calloc(1024,sizeof(char));
4 free(foo);
```

free nicht vergessen

In C muss man Speicher manuell wieder freigeben:

Beispiel: Aufruf von free()

```
1 [...]
2 char *foo;
3 foo = (char *) calloc(1024,sizeof(char));
4 free(foo);
```

`free()`...

- gibt nur Speicher, der mit malloc oder calloc alloziiert wurde wieder frei,

free nicht vergessen

In C muss man Speicher manuell wieder freigeben:

Beispiel: Aufruf von free()

```
1 [...]
2 char *foo;
3 foo = (char *) calloc(1024,sizeof(char));
4 free(foo);
```

`free()`...

- gibt nur Speicher, der mit malloc oder calloc alloziiert wurde wieder frei,
- darf nicht mit anderen Pointern aufgerufen werden,

free nicht vergessen

In C muss man Speicher manuell wieder freigeben:

Beispiel: Aufruf von free()

```
1 [...]
2 char *foo;
3 foo = (char *) calloc(1024,sizeof(char));
4 free(foo);
```

`free()`...

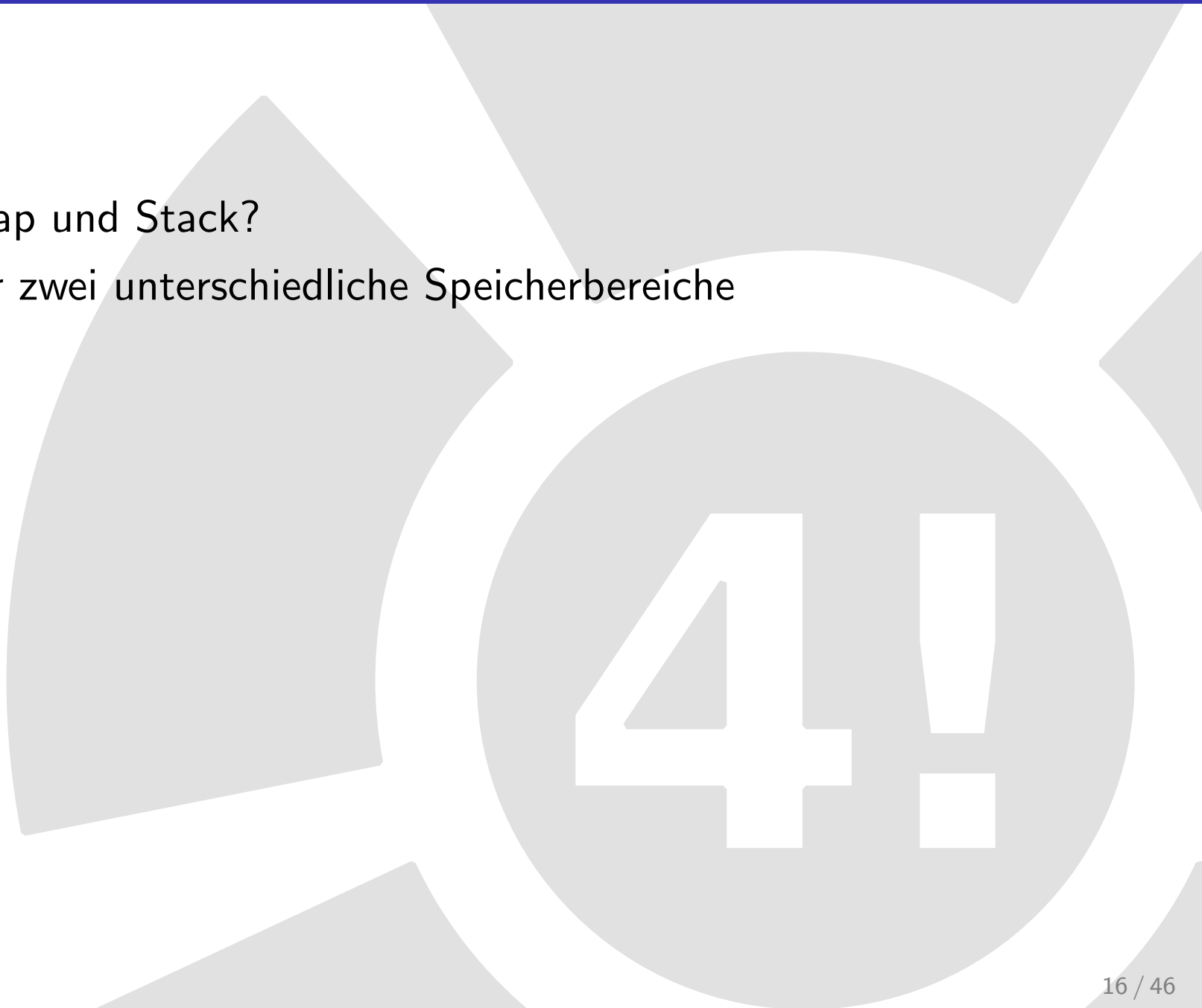
- gibt nur Speicher, der mit malloc oder calloc alloziiert wurde wieder frei,
- darf nicht mit anderen Pointern aufgerufen werden,
- darf nur einmal pro Speicherbereich aufgerufen werden, sonst gibt es ein 'double free'-Fehler.

Inhaltsverzeichniss

Heap und Stack

Was sind Heap und Stack?

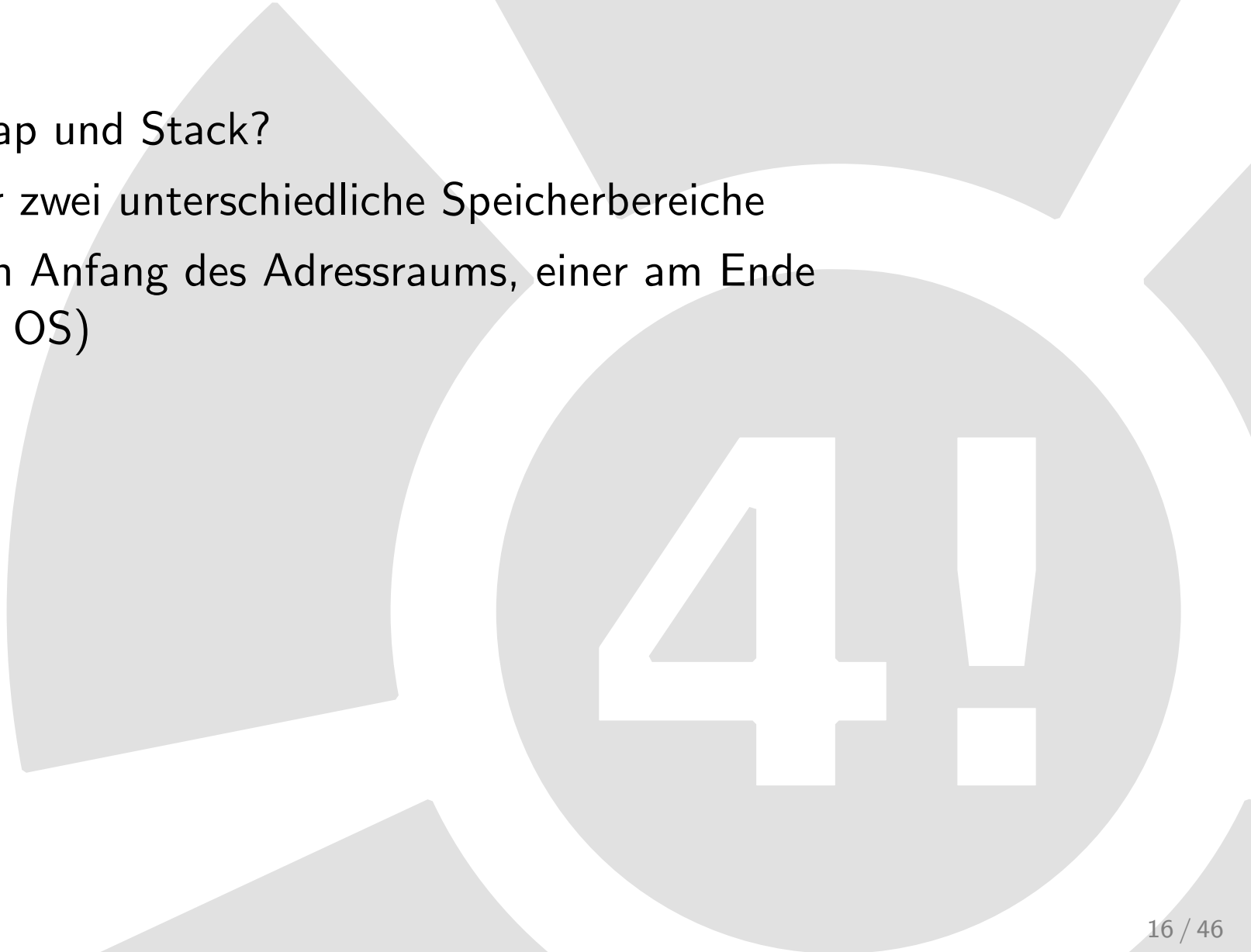
- Sind nur zwei unterschiedliche Speicherbereiche



Heap und Stack

Was sind Heap und Stack?

- Sind nur zwei unterschiedliche Speicherbereiche
- Einer am Anfang des Adressraums, einer am Ende (je nach OS)



Heap und Stack

Was sind Heap und Stack?

- Sind nur zwei unterschiedliche Speicherbereiche
- Einer am Anfang des Adressraums, einer am Ende (je nach OS)
- Bei Belegung wachsen sie aufeinander zu

Heap und Stack

Was sind Heap und Stack?

- Sind nur zwei unterschiedliche Speicherbereiche
- Einer am Anfang des Adressraums, einer am Ende (je nach OS)
- Bei Belegung wachsen sie aufeinander zu
- Heap: malloc, calloc, free

Heap und Stack

Was sind Heap und Stack?

- Sind nur zwei unterschiedliche Speicherbereiche
- Einer am Anfang des Adressraums, einer am Ende (je nach OS)
- Bei Belegung wachsen sie aufeinander zu
- Heap: malloc, calloc, free
- Stack: lokale Variablen

Heap und Stack

Was sind Heap und Stack?

- Sind nur zwei unterschiedliche Speicherbereiche
- Einer am Anfang des Adressraums, einer am Ende (je nach OS)
- Bei Belegung wachsen sie aufeinander zu
- Heap: malloc, calloc, free
- Stack: lokale Variablen
- Es gibt noch mehr 'Sonderbereiche': z.B. Data Segment

Finde den Bug

Beispiel: Fehlerhaftes Programm

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  char* foo(){
5      char moo[12] = {'H','e','l','l','o'};
6      return moo;
7  }
8
9  int main(int argc, char** argv){
10     char* bar = foo();
11     printf("%s\n",bar);
12 }
```

Finde den Bug

Beispiel: Fehlerhaftes Programm

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  char* foo(){
5      char moo[12] = {'H','e','l','l','o'};
6      return moo;
7  }
8
9  int main(int argc, char** argv){
10     char* bar = foo();
11     printf("%s\n",bar);
12 }
```

```
# ./a.out
¿Æ.
```

Korrigierte Version

Beispiel: Korrekte Fassung

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  char* foo(){
5      char *moo = malloc(sizeof(char) * 6);
6      sprintf(moo,"Hello");
7      return moo;
8  }
9
10 int main(int argc, char** argv){
11     char* bar = foo();
12     printf("%s\n",bar);
13 }
```

Korrigierte Version

Beispiel: Korrekte Fassung

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  char* foo(){
5      char *moo = malloc(sizeof(char) * 6);
6      sprintf(moo,"Hello");
7      return moo;
8  }
9
10 int main(int argc, char** argv){
11     char* bar = foo();
12     printf("%s\n",bar);
13 }
```

```
# ./a.out
Hello
```

Stack: Confusion on return

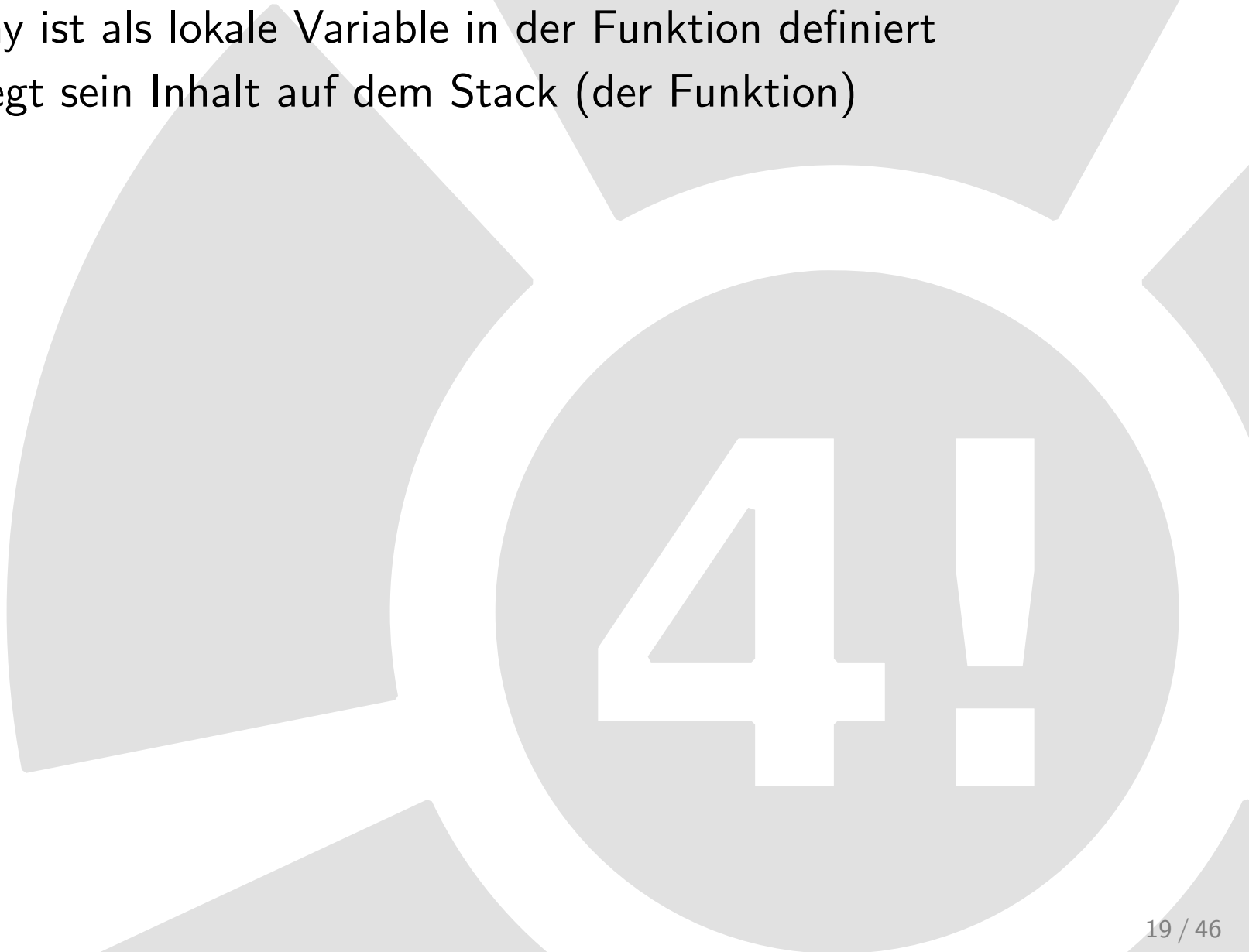
Was ist hier passiert?

- Das array ist als lokale Variable in der Funktion definiert

Stack: Confusion on return

Was ist hier passiert?

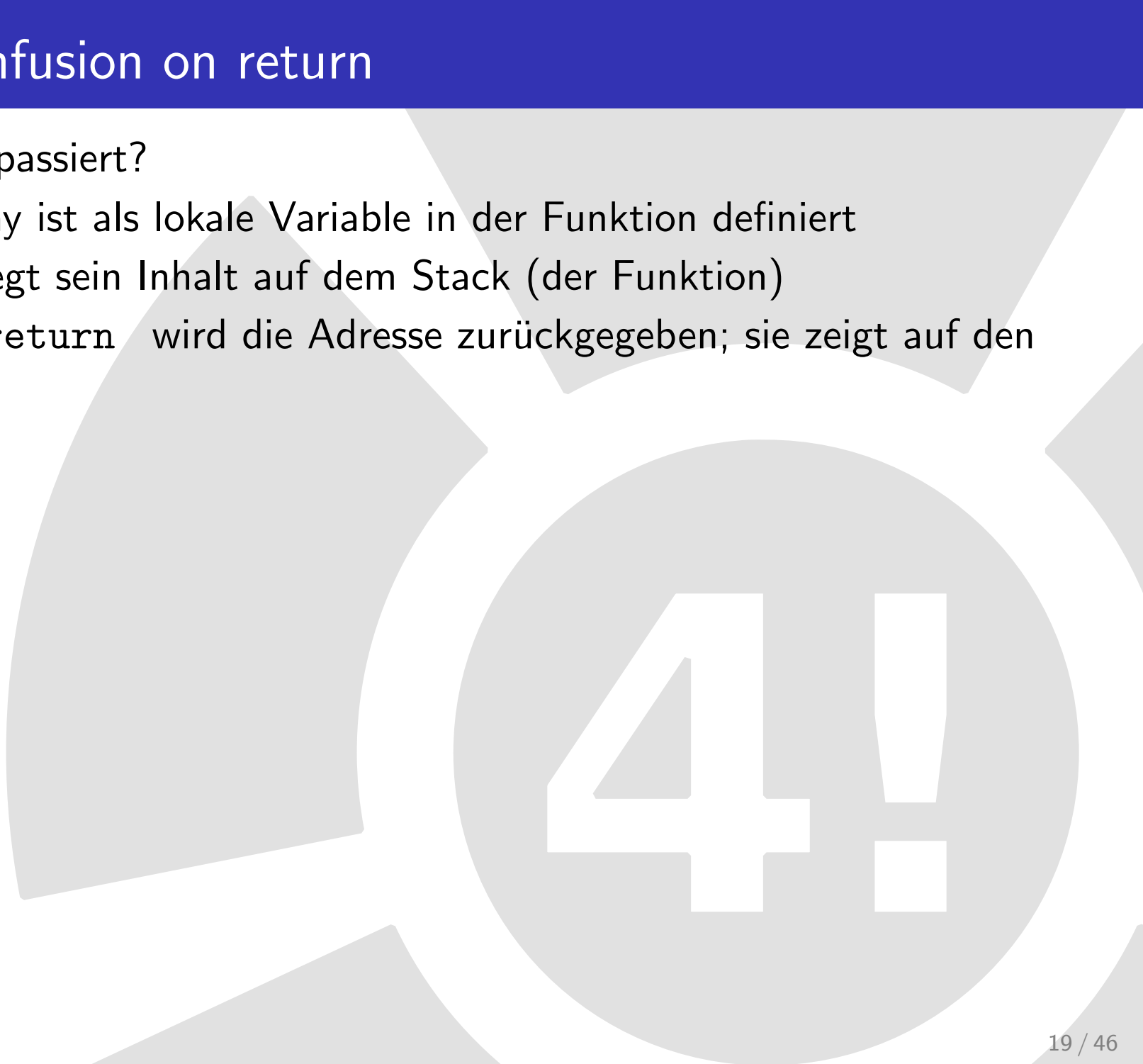
- Das array ist als lokale Variable in der Funktion definiert
- daher liegt sein Inhalt auf dem Stack (der Funktion)



Stack: Confusion on return

Was ist hier passiert?

- Das array ist als lokale Variable in der Funktion definiert
- daher liegt sein Inhalt auf dem Stack (der Funktion)
- Beim `return` wird die Adresse zurückgegeben; sie zeigt auf den Stack



Stack: Confusion on return

Was ist hier passiert?

- Das array ist als lokale Variable in der Funktion definiert
- daher liegt sein Inhalt auf dem Stack (der Funktion)
- Beim `return` wird die Adresse zurückgegeben; sie zeigt auf den Stack
- Gleichzeitig wird der Stack beim `return` 'abgeräumt', da er nicht mehr gebraucht wird

Stack: Confusion on return

Was ist hier passiert?

- Das array ist als lokale Variable in der Funktion definiert
- daher liegt sein Inhalt auf dem Stack (der Funktion)
- Beim `return` wird die Adresse zurückgegeben; sie zeigt auf den Stack
- Gleichzeitig wird der Stack beim `return` 'abgeräumt', da er nicht mehr gebraucht wird
- Der nächste Funktionsaufruf überschreibt so möglicherweise die Daten des Arrays.

Stack: Confusion on return

Was ist hier passiert?

- Das array ist als lokale Variable in der Funktion definiert
- daher liegt sein Inhalt auf dem Stack (der Funktion)
- Beim `return` wird die Adresse zurückgegeben; sie zeigt auf den Stack
- Gleichzeitig wird der Stack beim `return` 'abgeräumt', da er nicht mehr gebraucht wird
- Der nächste Funktionsaufruf überschreibt so möglicherweise die Daten des Arrays.
- Machner Compiler warnt beim Übersetzen:

Stack: Confusion on return

Was ist hier passiert?

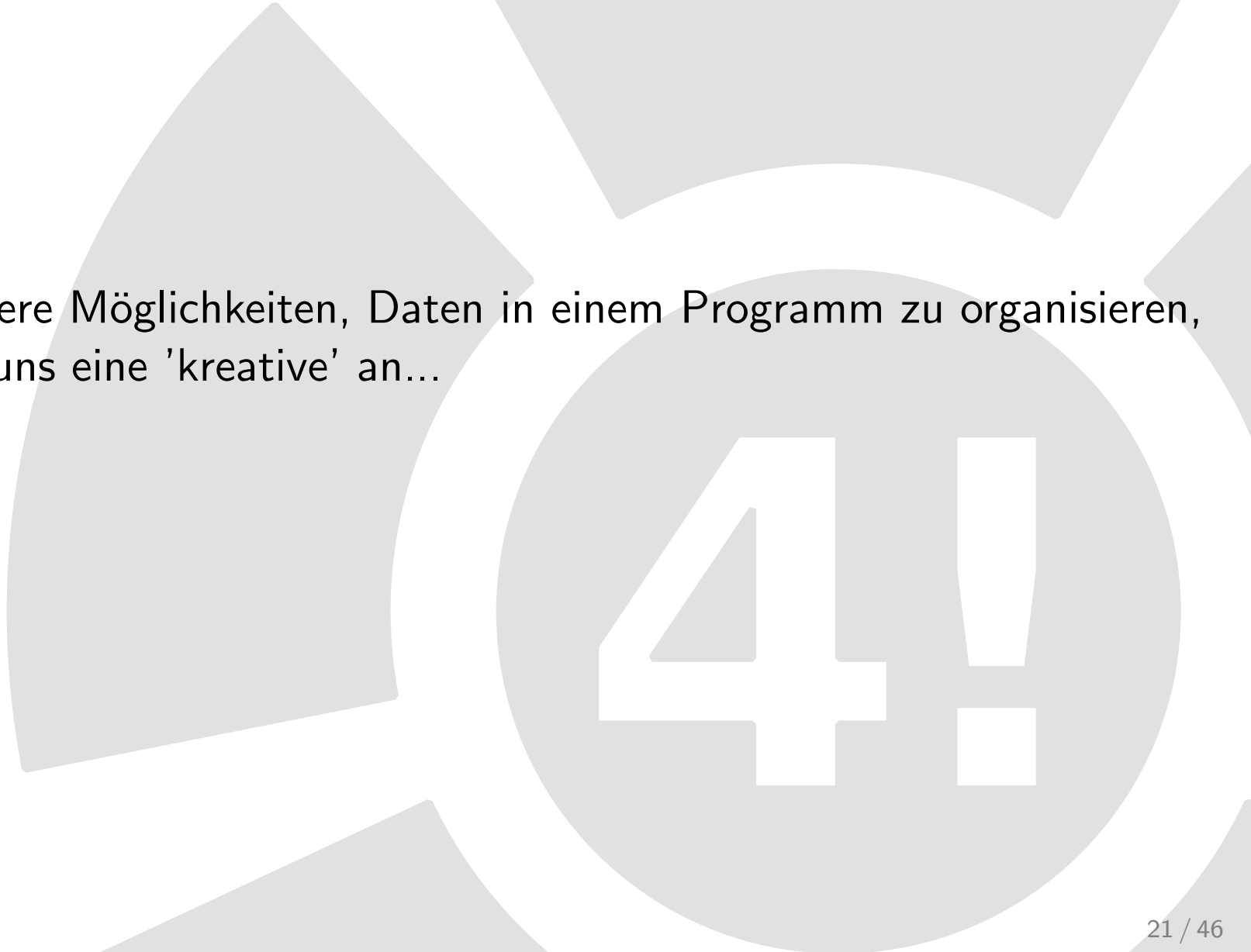
- Das array ist als lokale Variable in der Funktion definiert
- daher liegt sein Inhalt auf dem Stack (der Funktion)
- Beim `return` wird die Adresse zurückgegeben; sie zeigt auf den Stack
- Gleichzeitig wird der Stack beim `return` 'abgeräumt', da er nicht mehr gebraucht wird
- Der nächste Funktionsaufruf überschreibt so möglicherweise die Daten des Arrays.
- Machner Compiler warnt beim Übersetzen:

```
# gcc heapstack.c
heapstack.c: In function 'foo':
heapstack.c:7: warning: function returns address
of local variable
```

Inhaltsverzeichniss

Wie organisiere ich meine Daten?

Es gibt mehrere Möglichkeiten, Daten in einem Programm zu organisieren, schauen wir uns eine 'kreative' an...



Übergabe in einem 'Blob'

Beispiel: Kreative Datenübergabe

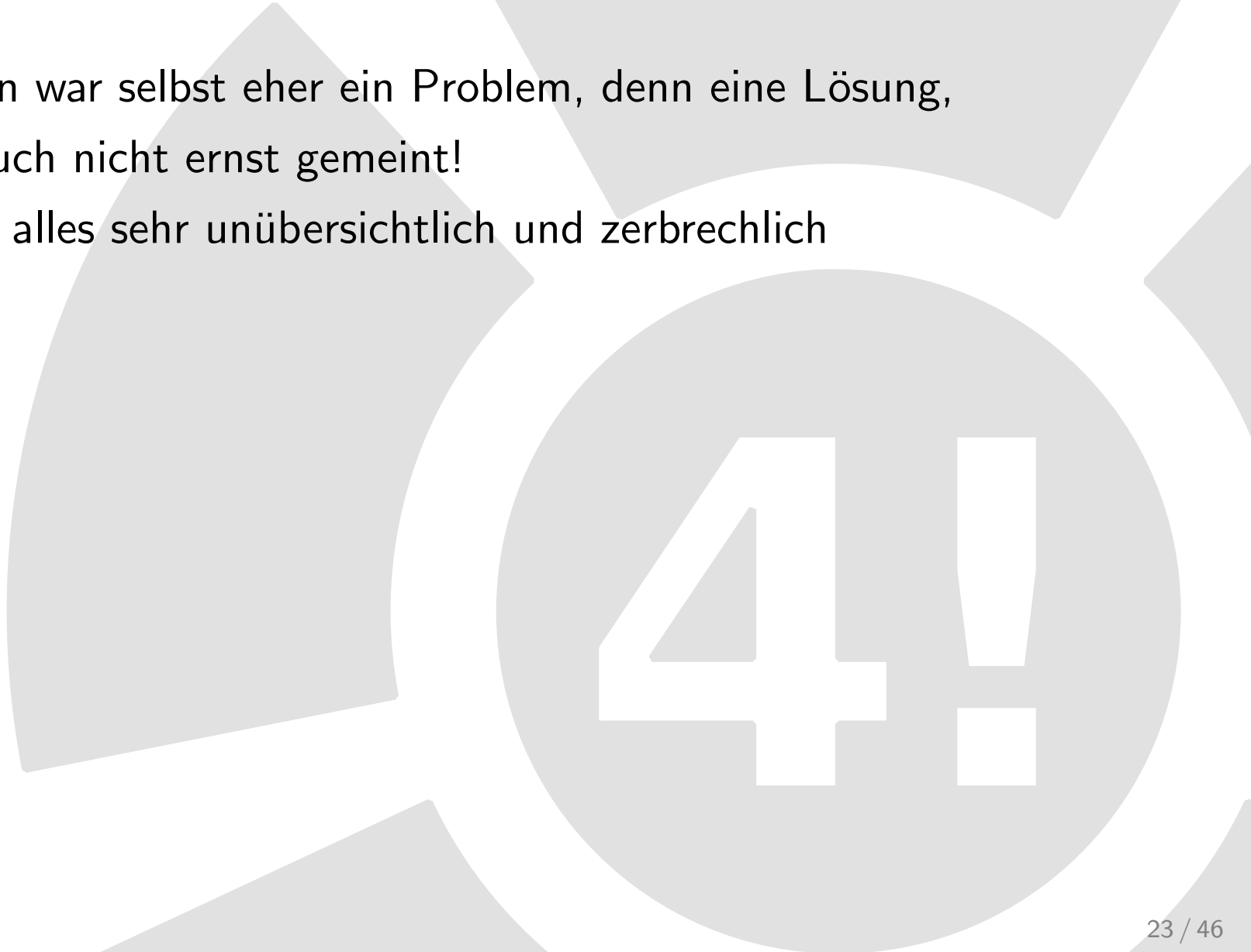
```
1  void print_item(char *blob){
2      char* amount[2];
3      strncpy(amount, blob , 2);
4      char* name[30];
5      strncpy(name, blob+2 , 30);
6      [...]
7      printf("%s mal %s \n",amount,name);
8  }
9      [...]
10     char *bill_item = calloc (1024,sizeof(char));
11     memcpy (bill_item , "05" , 2); // Amount
12     memcpy (bill_item + 2 , "Club Mate, 0,5l Flasche" , 24);
13     memcpy (bill_item + 26 , "1,00" , 4); // Price
14     memcpy (bill_item + 30 , "5,00" , 4); // Sum
15     print_item(bill_item);
```

Nachteile dieser 'Lösung'

- Das eben war selbst eher ein Problem, denn eine Lösung,

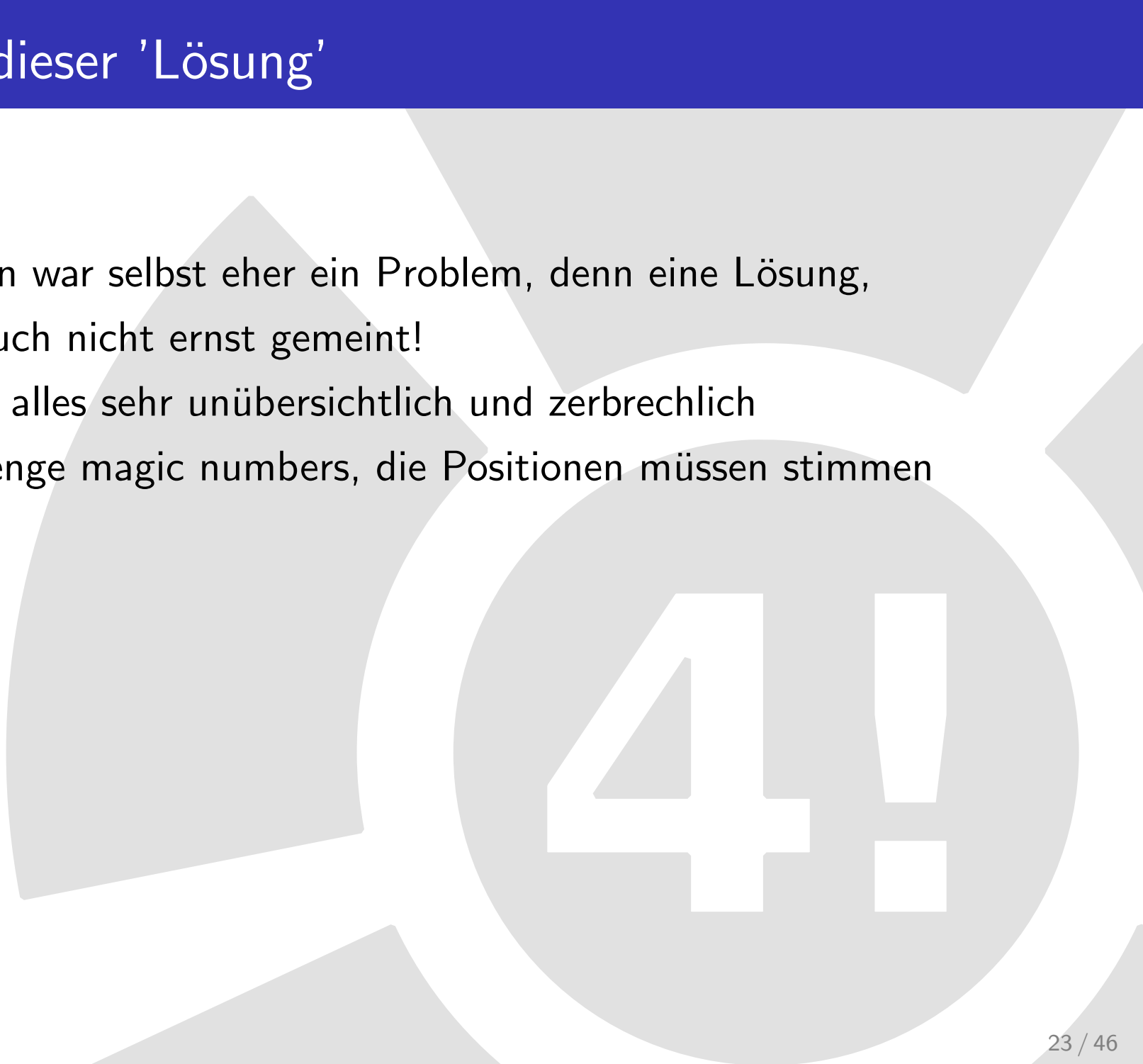
Nachteile dieser 'Lösung'

- Das eben war selbst eher ein Problem, denn eine Lösung,
- ...und auch nicht ernst gemeint!
- Das war alles sehr unübersichtlich und zerbrechlich



Nachteile dieser 'Lösung'

- Das eben war selbst eher ein Problem, denn eine Lösung,
- ...und auch nicht ernst gemeint!
- Das war alles sehr unübersichtlich und zerbrechlich
- Jede Menge magic numbers, die Positionen müssen stimmen



Nachteile dieser 'Lösung'

- Das eben war selbst eher ein Problem, denn eine Lösung,
- ...und auch nicht ernst gemeint!
- Das war alles sehr unübersichtlich und zerbrechlich
- Jede Menge magic numbers, die Positionen müssen stimmen
- die Größe ist dadurch doch wieder festgelegt!

Nachteile dieser 'Lösung'

- Das eben war selbst eher ein Problem, denn eine Lösung,
- ...und auch nicht ernst gemeint!
- Das war alles sehr unübersichtlich und zerbrechlich
- Jede Menge magic numbers, die Positionen müssen stimmen
- die Größe ist dadurch doch wieder festgelegt!
- auch wenn man die Daten mit 'Trennzeichen' übergibt wird es nicht basser

Nachteile dieser 'Lösung'

- Das eben war selbst eher ein Problem, denn eine Lösung,
- ...und auch nicht ernst gemeint!
- Das war alles sehr unübersichtlich und zerbrechlich
- Jede Menge magic numbers, die Positionen müssen stimmen
- die Größe ist dadurch doch wieder festgelegt!
- auch wenn man die Daten mit 'Trennzeichen' übergibt wird es nicht basser
- einzelne Elemente zu vergrößern ist unmöglich

Nachteile dieser 'Lösung'

- Das eben war selbst eher ein Problem, denn eine Lösung,
- ...und auch nicht ernst gemeint!
- Das war alles sehr unübersichtlich und zerbrechlich
- Jede Menge magic numbers, die Positionen müssen stimmen
- die Größe ist dadurch doch wieder festgelegt!
- auch wenn man die Daten mit 'Trennzeichen' übergibt wird es nicht basser
- einzelne Elemente zu vergrößern ist unmöglich
- es gibt keine Datentypen
- ...

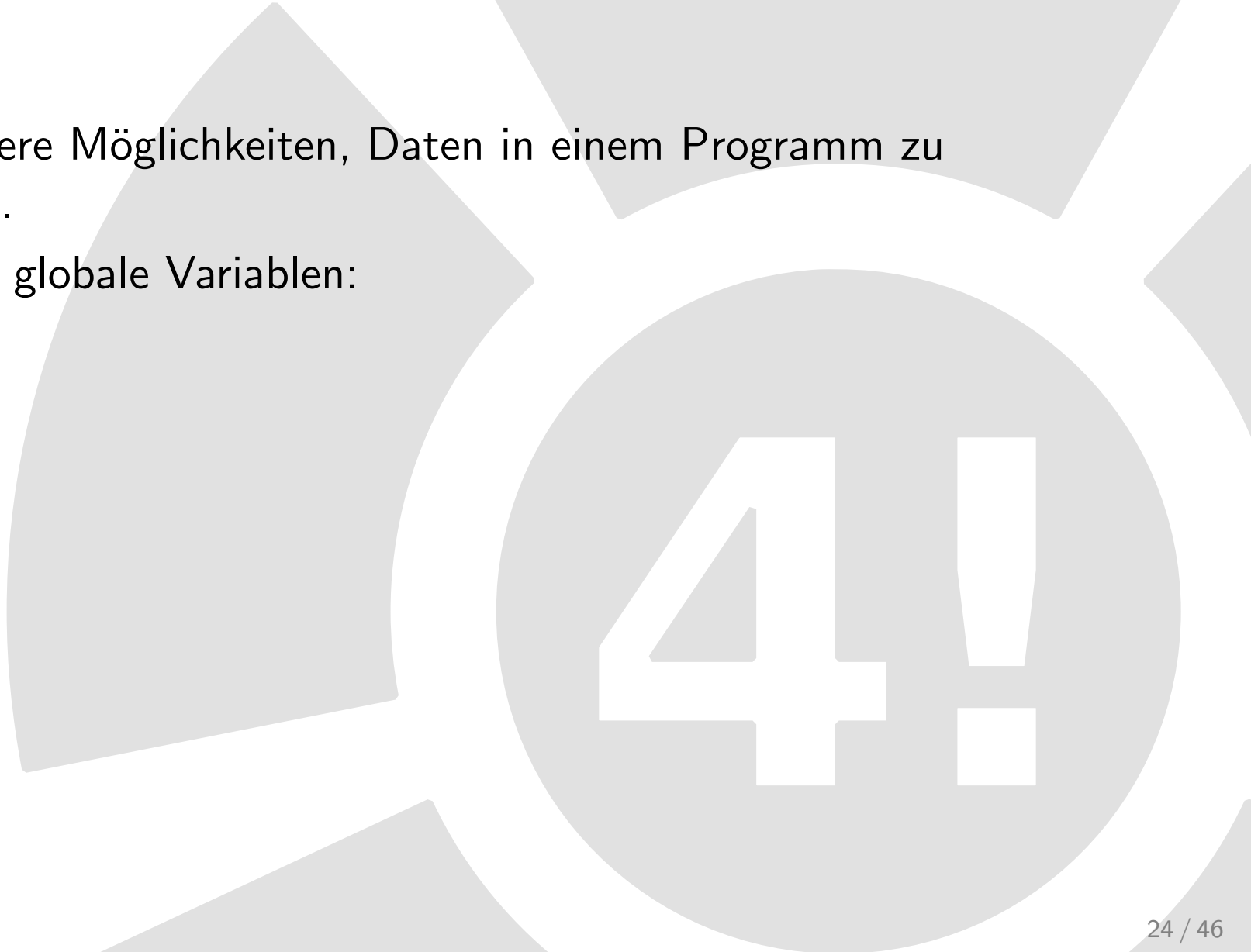
Wie organisiere ich meine Daten?

Es gibt mehrere Möglichkeiten, Daten in einem Programm zu 'organisieren'.

Wie organisiere ich meine Daten?

Es gibt mehrere Möglichkeiten, Daten in einem Programm zu 'organisieren'.

- Alles als globale Variablen:



Wie organisiere ich meine Daten?

Es gibt mehrere Möglichkeiten, Daten in einem Programm zu 'organisieren'.

- Alles als globale Variablen:
  $\Rightarrow$ sehr fehlerträchtig, Probleme bei tests, parallelisierung

Wie organisiere ich meine Daten?

Es gibt mehrere Möglichkeiten, Daten in einem Programm zu 'organisieren'.

- Alles als globale Variablen:
  $\Rightarrow$ sehr fehlerträchtig, Probleme bei tests, parallelisierung
- Alle einzeln als Parameter übergeben:

Wie organisiere ich meine Daten?

Es gibt mehrere Möglichkeiten, Daten in einem Programm zu 'organisieren'.

- Alles als globale Variablen:
  $\Rightarrow$ sehr fehlerträchtig, Probleme bei tests, parallelisierung
- Alle einzeln als Parameter übergeben:
  $\Rightarrow$ unübersichtliche Aufrufe, sehr viel Tipparbeit

Wie organisiere ich meine Daten?

Es gibt mehrere Möglichkeiten, Daten in einem Programm zu 'organisieren'.

- Alles als globale Variablen:
  $\Rightarrow$ sehr fehlerträchtig, Probleme bei tests, parallelisierung
- Alle einzeln als Parameter übergeben:
  $\Rightarrow$ unübersichtliche Aufrufe, sehr viel Tipparbeit
- Mittels eigener Datentypen:

Wie organisiere ich meine Daten?

Es gibt mehrere Möglichkeiten, Daten in einem Programm zu 'organisieren'.

- Alles als globale Variablen:
  $\Rightarrow$ sehr fehlerträchtig, Probleme bei tests, parallelisierung
- Alle einzeln als Parameter übergeben:
  $\Rightarrow$ unübersichtliche Aufrufe, sehr viel Tipparbeit
- Mittels eigener Datentypen:
  $\Rightarrow$ übersichtlich, klar definierte Schnittstellen, guter Stil

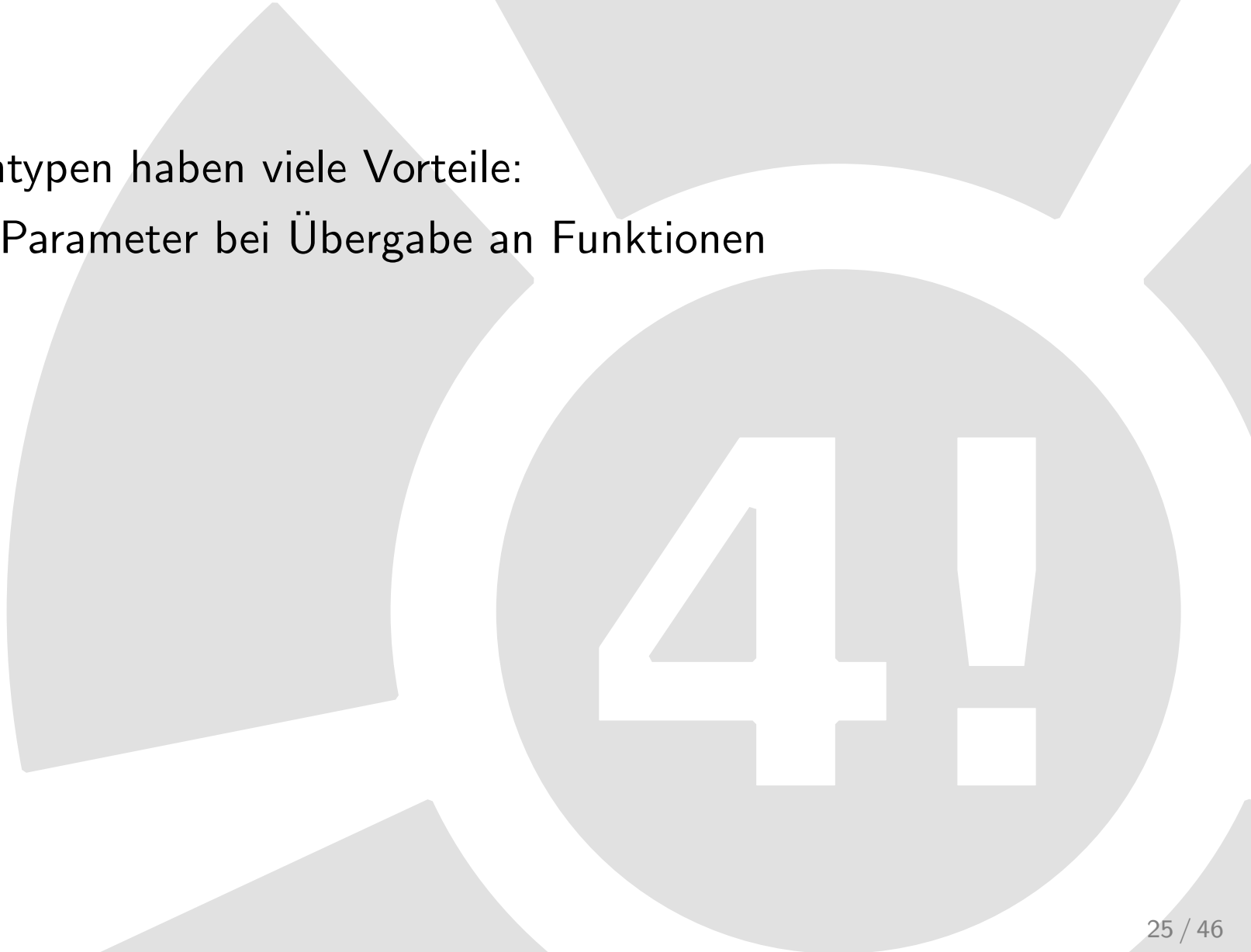
Vorteile eigener Datentypen

Eigene Datentypen haben viele Vorteile:

Vorteile eigener Datentypen

Eigene Datentypen haben viele Vorteile:

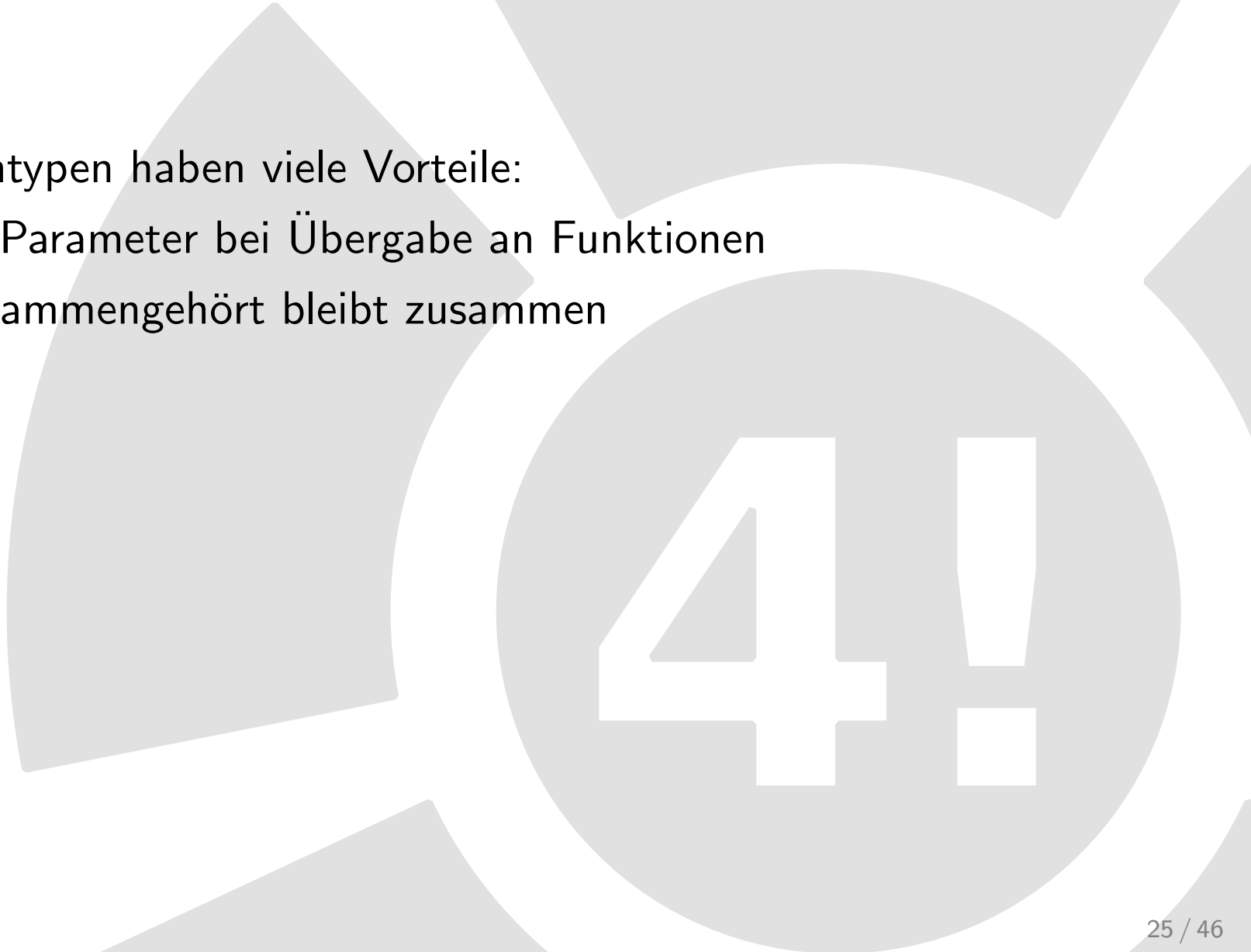
- Nur ein Parameter bei Übergabe an Funktionen



Vorteile eigener Datentypen

Eigene Datentypen haben viele Vorteile:

- Nur ein Parameter bei Übergabe an Funktionen
- Was zusammengehört bleibt zusammen



Vorteile eigener Datentypen

Eigene Datentypen haben viele Vorteile:

- Nur ein Parameter bei Übergabe an Funktionen
- Was zusammengehört bleibt zusammen
- Es erhöht den Überblick:

Vorteile eigener Datentypen

Eigene Datentypen haben viele Vorteile:

- Nur ein Parameter bei Übergabe an Funktionen
- Was zusammengehört bleibt zusammen
- Es erhöht den Überblick:
  `print_user(user);`
  vs.
  `print_user(firstname,lastname,birthday,email,password,...)`

Definition eines eigenen Datentyps

Beispiel: Person als Datentyp

```
1 typedef struct person_t {
2       char* firstname;
3       char* lastname;
4       char  weddingdate[9];
5       int   shoesize;
6  } person;
```

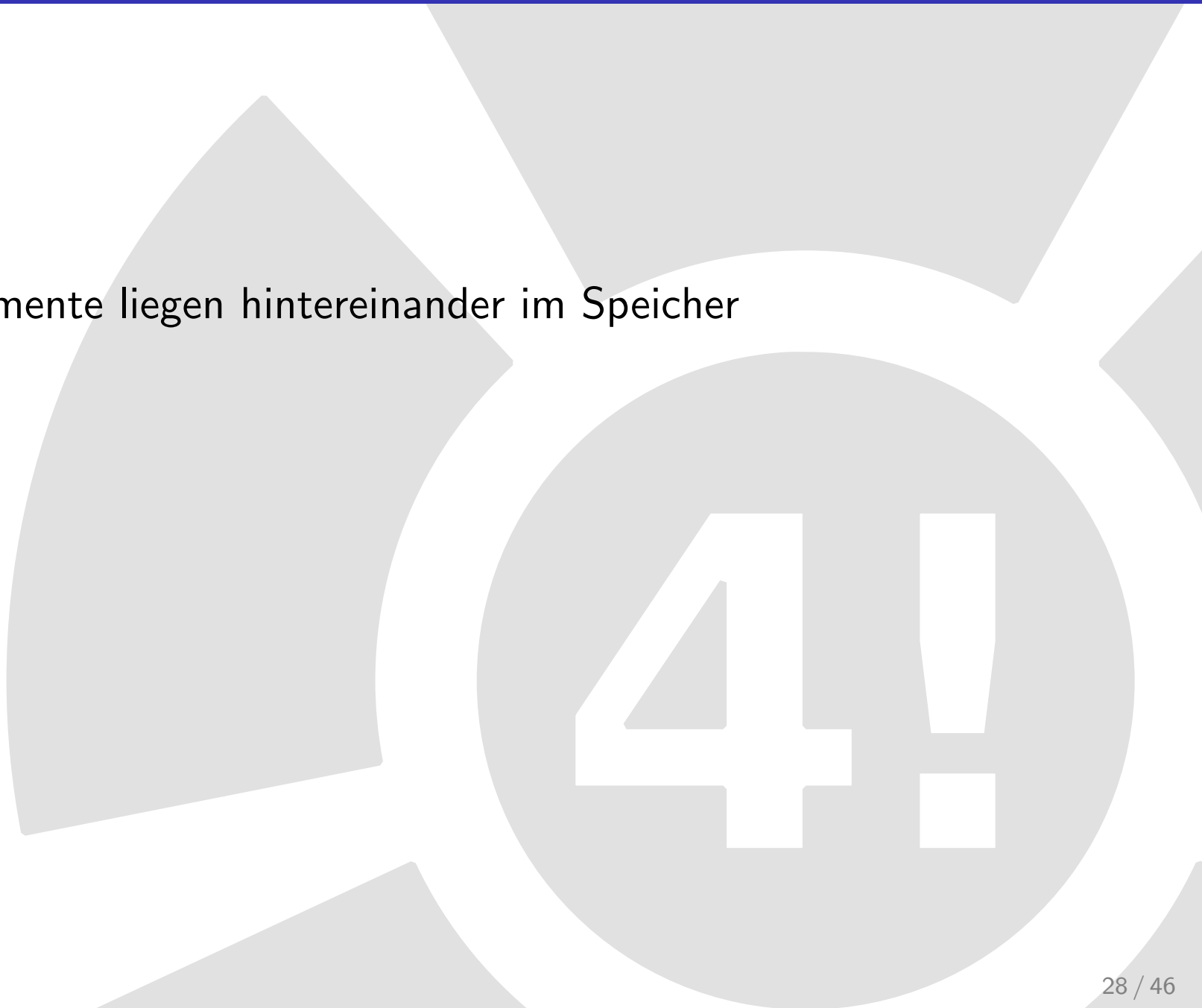
Benutzen eines eigenen Datentyps

Beispiel: struct initialisieren

```
1  typedef struct person_t {
2        char* firstname;
3        char* lastname;
4        char  weddingdate[9];
5        int   shoesize;
6  } person;
7
8  [...]
9  person* p = (person*) calloc(1,sizeof(person));
10
11 p->shoesize=192;
12
13 sprintf(p->weddingdate,"%s","15092007");
14
15 p->firstname=(char*)calloc(8,sizeof(char));
16 sprintf(p->firstname,"%s","Florian");
```

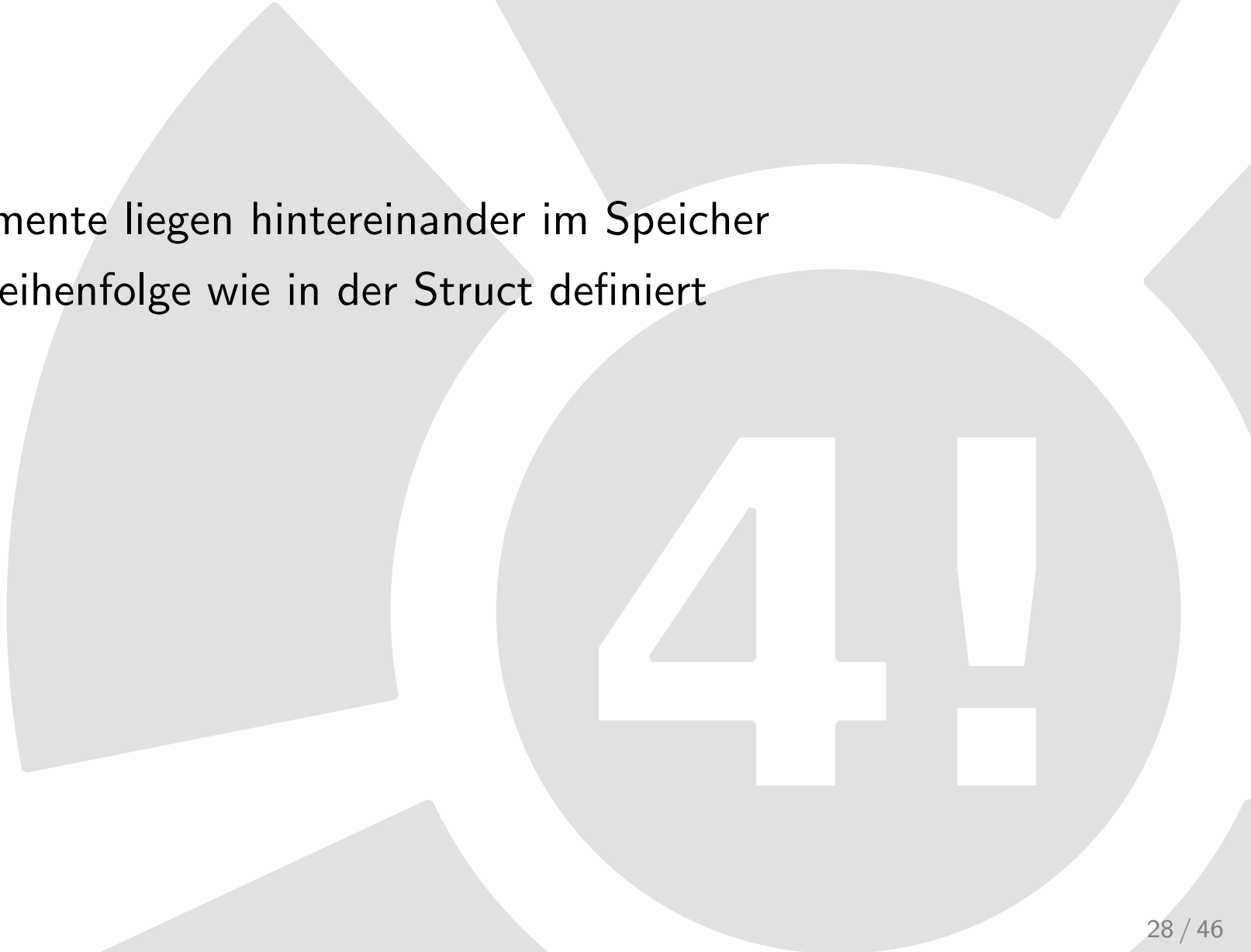
Structs im Speicher

- Alle Elemente liegen hintereinander im Speicher



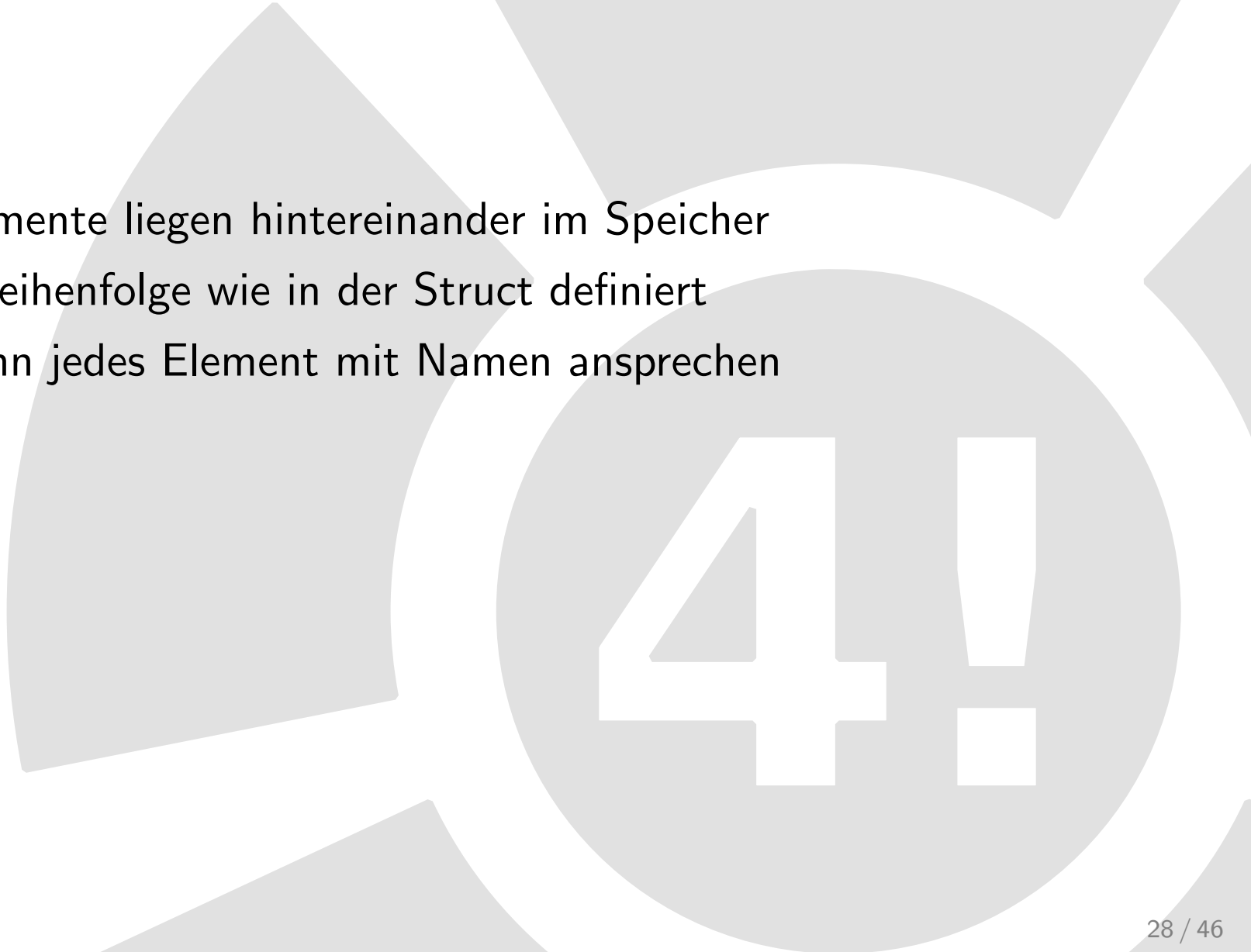
Structs im Speicher

- Alle Elemente liegen hintereinander im Speicher
- In der Reihenfolge wie in der Struct definiert



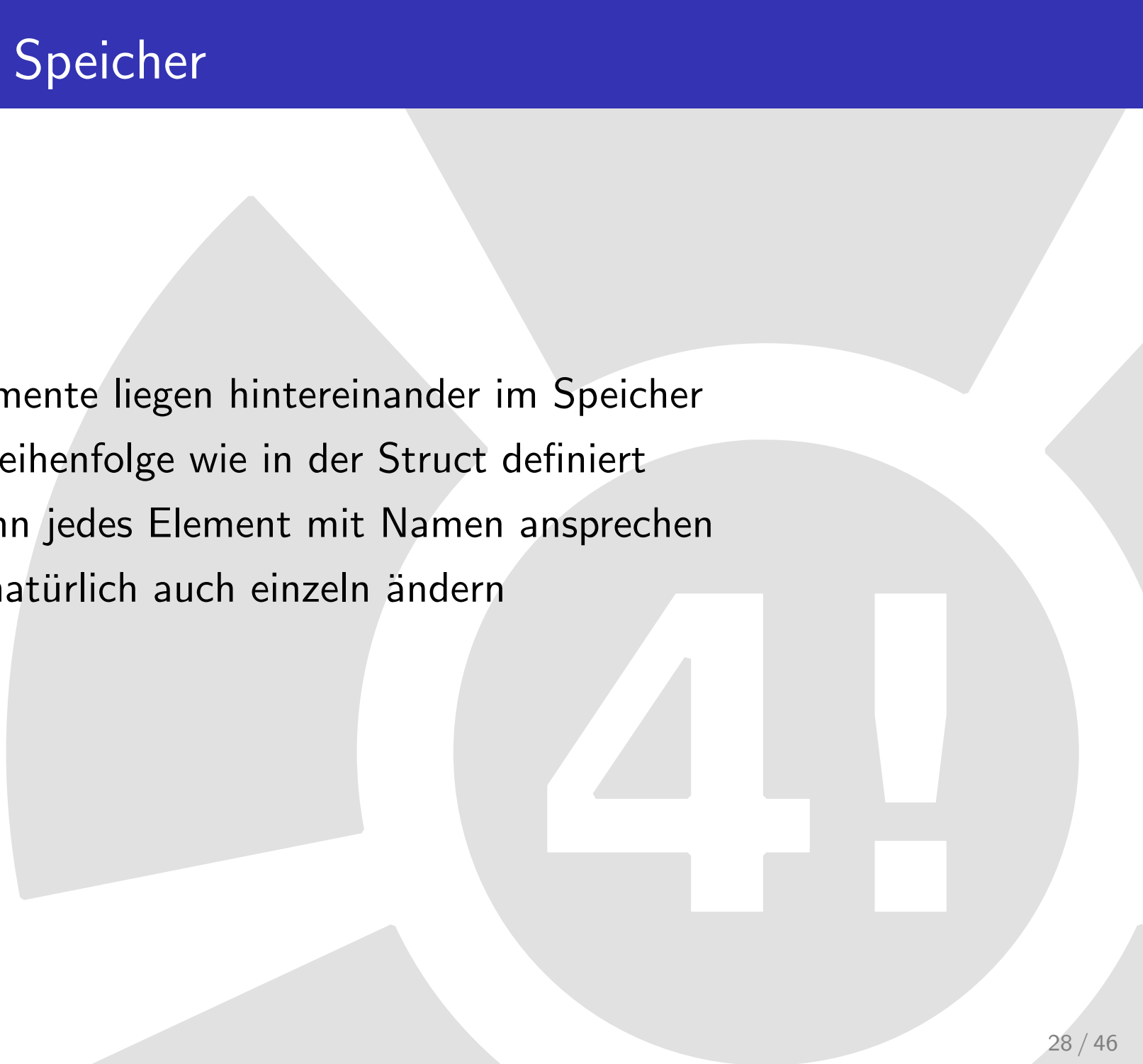
Structs im Speicher

- Alle Elemente liegen hintereinander im Speicher
- In der Reihenfolge wie in der Struct definiert
- Man kann jedes Element mit Namen ansprechen



Structs im Speicher

- Alle Elemente liegen hintereinander im Speicher
- In der Reihenfolge wie in der Struct definiert
- Man kann jedes Element mit Namen ansprechen
- ... und natürlich auch einzeln ändern



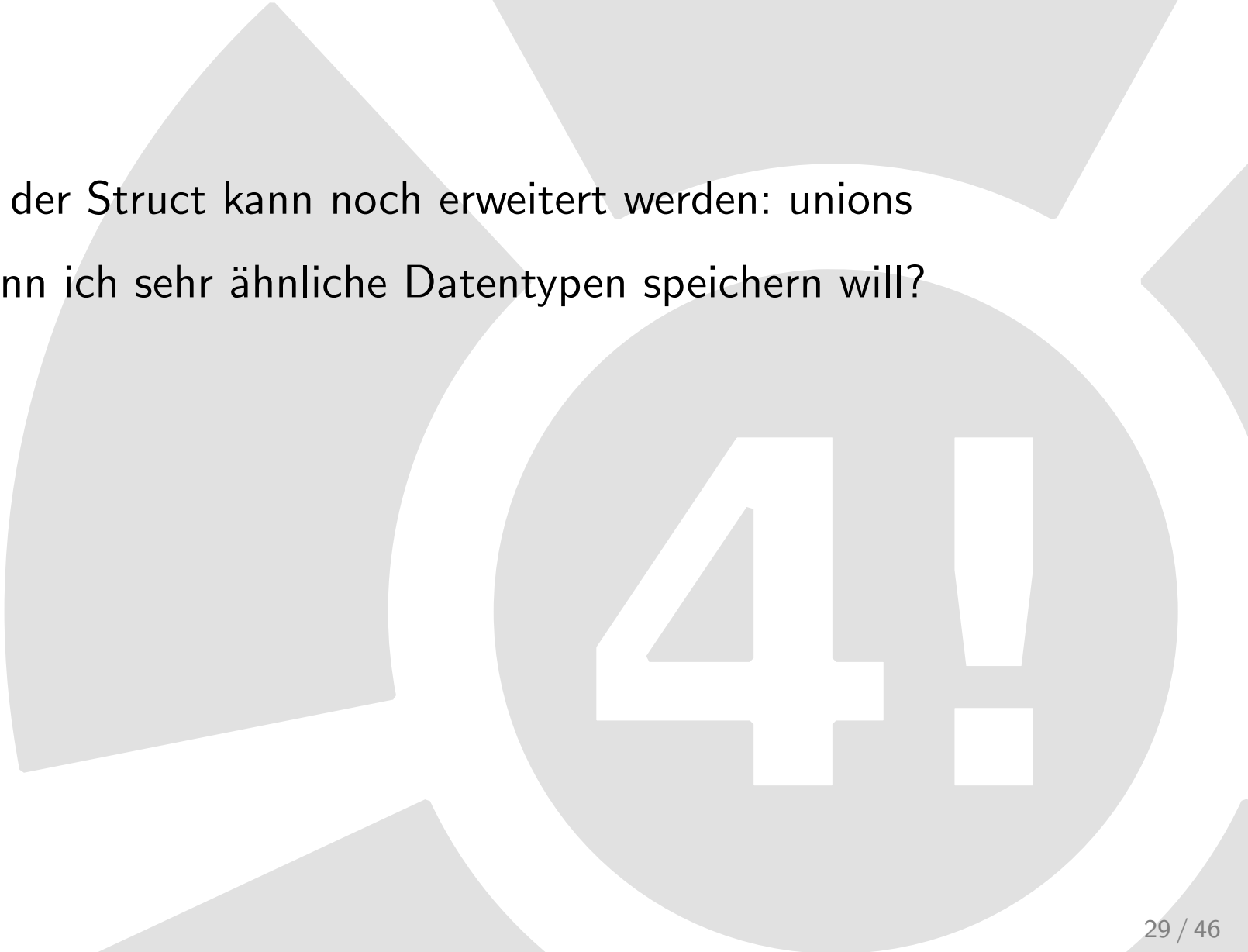
Structs im Speicher

- Alle Elemente liegen hintereinander im Speicher
- In der Reihenfolge wie in der Struct definiert
- Man kann jedes Element mit Namen ansprechen
- ... und natürlich auch einzeln ändern
- Sehr gut geeignet als Rückgabewerte für Funktionen - warum?

Flexibility for the win!

Das Konzept der Struct kann noch erweitert werden: unions

- Was, wenn ich sehr ähnliche Datentypen speichern will?



Flexibility for the win!

Das Konzept der Struct kann noch erweitert werden: unions

- Was, wenn ich sehr ähnliche Datentypen speichern will?
- Beispiel: Punkte in Koordinatensystemen

Flexibility for the win!

Das Konzept der Struct kann noch erweitert werden: unions

- Was, wenn ich sehr ähnliche Datentypen speichern will?
- Beispiel: Punkte in Koordinatensystemen
- Ich möchte einen 'globalen' Datentyp 'Punkt'

Flexibility for the win!

Das Konzept der Struct kann noch erweitert werden: unions

- Was, wenn ich sehr ähnliche Datentypen speichern will?
- Beispiel: Punkte in Koordinatensystemen
- Ich möchte einen 'globalen' Datentyp 'Punkt'
- Ich möchte Polar- und Kartesische Koordinaten übergeben können

Koordinaten

Beispiel: Bunte Punkte als Datentypen

```
1  typedef struct polar_t {
2      float angle;
3      float distance;
4      int color;
5   } polar;
6
7  typedef struct cartesian_t {
8      int x;
9      int y;
10     int color;
11  } cartesian;
```

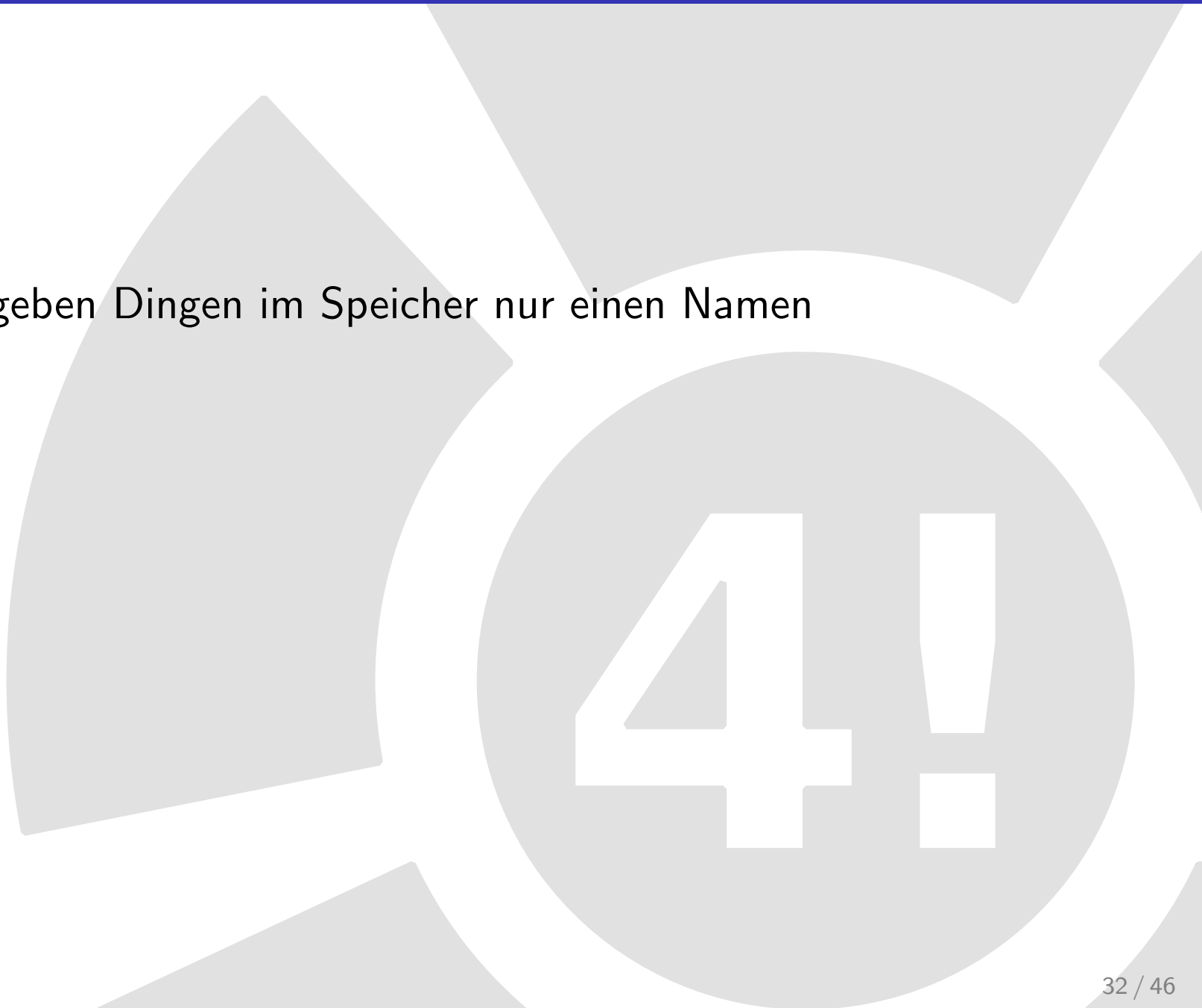
Unite!

Beispiel: Bunte Punkte als komplexer Datentyp

```
1
2  typedef struct point_t {
3    int color;
4    enum ptype_t { UNDEF, CART, POLAR} ptype;
5
6    union {
7      struct {  //polar coordinates
8        float angle;
9        float distance;
10     } polar;
11
12     struct {  //cartesian
13       int x;
14       int y;
15     } cartesian;
16   };
17
18 } point;
```

Unions - Speicherabbild

- Unions geben Dingen im Speicher nur einen Namen



Unions - Speicherabbild

- Unions geben Dingen im Speicher nur einen Namen
- Die Union ist so groß wie das 'größte Element'

Unions - Speicherabbild

- Unions geben Dingen im Speicher nur einen Namen
- Die Union ist so groß wie das 'größte Element'
- Die Elemente der Union belegen denselben Speicherbereich

Unions - Speicherabbild

- Unions geben Dingen im Speicher nur einen Namen
- Die Union ist so groß wie das 'größte Element'
- Die Elemente der Union belegen denselben Speicherbereich
- Dadurch kann ein String auch als Zahl 'interpretiert' werden

Unions - Speicherabbild

- Unions geben Dingen im Speicher nur einen Namen
- Die Union ist so groß wie das 'größte Element'
- Die Elemente der Union belegen denselben Speicherbereich
- Dadurch kann ein String auch als Zahl 'interpretiert' werden
- BigEndian vs. LittleEndian: Anordnung der einzelnen Bytes...

Benutzung von unions

Beispiel: Bunte Punkte als komplexer Datentyp

```
1
2  [...]
3
4  int main(int argc, char** argv){
5
6      point *bar = calloc(1,sizeof(point));
7
8      bar->color=3;
9      bar->type=POLAR;
10     bar->polar.angle=45;
11     bar->polar.distance=123;
12
13     do_sth(bar);
14 }
```

Benutzung von unions

Beispiel: Bunte Punkte als komplexer Datentyp

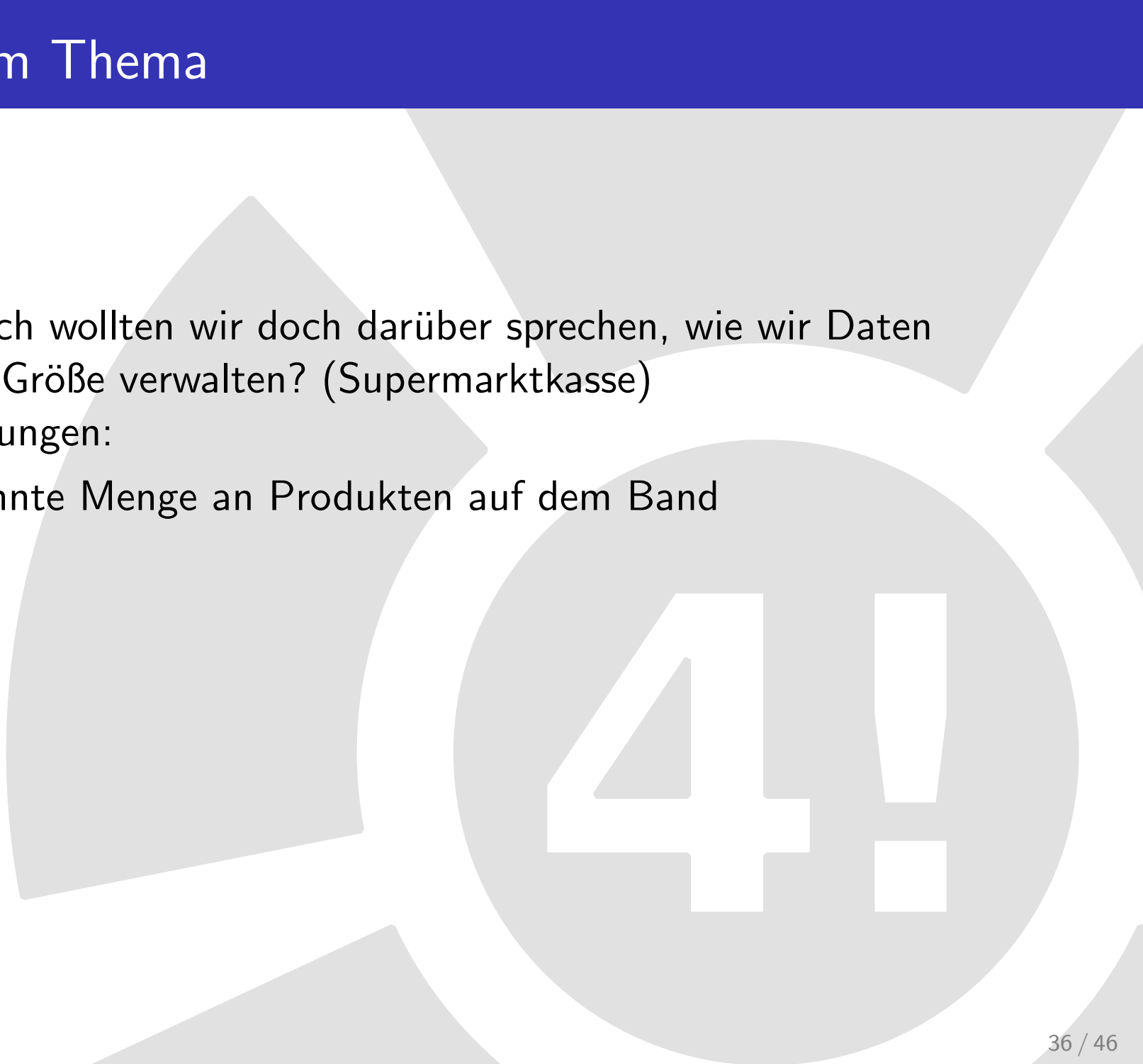
```
1
2  void do_sth(point *somepoint){
3
4      printf("type: %i - ", somepoint->type);
5      switch (somepoint->type){
6          case UNDEF:   printf ("undef\n");
7                        break;
8          case CART:   printf ("Cartesian\n");
9                        break;
10         case POLAR:     printf ("Polar\n");
11                       break;
12     }
13 }
```

Inhaltsverzeichniss

Zurück zum Thema

Aber eigentlich wollten wir doch darüber sprechen, wie wir Daten unbekannter Größe verwalten? (Supermarktkasse)
Die Anforderungen:

- Unbekannte Menge an Produkten auf dem Band



Zurück zum Thema

Aber eigentlich wollten wir doch darüber sprechen, wie wir Daten unbekannter Größe verwalten? (Supermarktkasse)
Die Anforderungen:

- Unbekannte Menge an Produkten auf dem Band
- Produkte nach Gewicht oder Stück auf dem Band

Zurück zum Thema

Aber eigentlich wollten wir doch darüber sprechen, wie wir Daten unbekannter Größe verwalten? (Supermarktkasse)
Die Anforderungen:

- Unbekannte Menge an Produkten auf dem Band
- Produkte nach Gewicht oder Stück auf dem Band
- Funktionen sollen einfach aufzurufen sein

Zurück zum Thema

Aber eigentlich wollten wir doch darüber sprechen, wie wir Daten unbekannter Größe verwalten? (Supermarktkasse)
Die Anforderungen:

- Unbekannte Menge an Produkten auf dem Band
- Produkte nach Gewicht oder Stück auf dem Band
- Funktionen sollen einfach aufzurufen sein

Für die letzten beiden Punkte haben wir das Handwerkszeug, jetzt kommt noch die Lösung für das Fliessbandproblem:

Zurück zum Thema

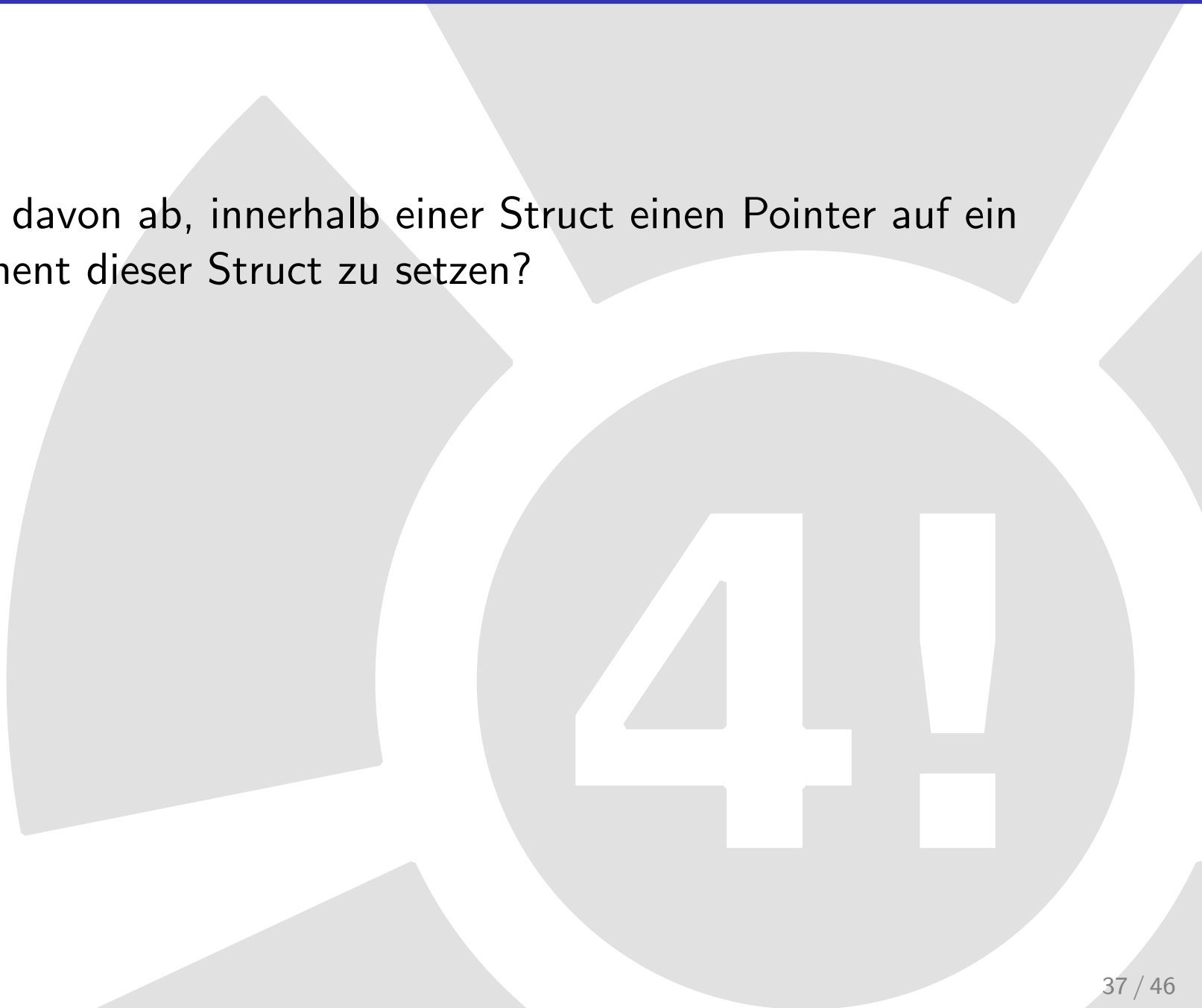
Aber eigentlich wollten wir doch darüber sprechen, wie wir Daten unbekannter Größe verwalten? (Supermarktkasse)
Die Anforderungen:

- Unbekannte Menge an Produkten auf dem Band
- Produkte nach Gewicht oder Stück auf dem Band
- Funktionen sollen einfach aufzurufen sein

Für die letzten beiden Punkte haben wir das Handwerkszeug, jetzt kommt noch die Lösung für das Fliessbandproblem: dynamische Listen!

dynamische Listen

Wer hält uns davon ab, innerhalb einer Struct einen Pointer auf ein weiteres Element dieser Struct zu setzen?



dynamische Listen

Wer hält uns davon ab, innerhalb einer Struct einen Pointer auf ein weiteres Element dieser Struct zu setzen? Niemand:

Beispiel: ein Listenelement

```
1
2  typedef struct bon_t{
3    bonelement* this;
4    struct bon_t* next; // the next bill element
5  }bon;
```

dynamische Listen(cont.)

Beispiel: Listendefinition

```
1
2  typedef struct bonelement_t{
3      int count;
4      char* product;
5      double singleprice;
6      double product_sum;
7  } bonelement;
8
9
10 typedef struct bon_t{
11   bonelement* this;
12   struct bon_t* next;
13 }bon;
```

dynamische Listen (example)

Beispiel: Liste

```
1  bon *k_head, *k_tail;
2
3  //first element
4  k_tail = calloc(1,sizeof(bon));
5  k_head = k_tail;
6  k_tail->next = NULL;
7  k_tail->this = newElem(5,"Club Mate",0.75);
8
9  //next element
10 k_tail->next=calloc(1,sizeof(bon));
11 k_tail=k_tail->next;
12 k_tail->next=NULL;
13 k_tail->this= newElem(2,"Wiener",2.45);
14
15 [...]
16
17 print_bon(k_head);
```

dynamische Listen (example, cont.)

Beispiel: Liste - die Funktionen

```
1  bonelement* newElem(int count, char* product,
2                                          double preis)
3  {
4
5      bonelement* item =
6        (bonelement*) calloc(1,sizeof(bonelement));
7
8      item->count=count;
9      item->product=strdup(product);
10     item->singleprice=preis;
11     item->product_sum=count*preis;
12
13     return item;
14 }
```

dynamische Listen (example, cont.)

Beispiel: Liste - die Funktionen

```
1
2  void print_bon(bon* firstelement){
3       dump_bonhead();
4       bon* kassenbon_current = firstelement;
5
6       while(kassenbon_current!=NULL){
7        dump_bonelement(kassenbon_current->this);
8        kassenbon_current=kassenbon_current->next;
9       }
10
11 }
```

dynamische Listen (example, cont.)

Beispiel: Liste - die Funktionen

```
1  void dump_bonhead(){
2      printf("␣%s␣%-35s␣%s␣␣␣%s␣␣␣␣\n",
3          "Count","Product","SP","SUM");
4  }
5
6  void dump_bonelement(bonelement* elem){
7      printf("␣%03i␣␣␣%-35s␣%04.2f␣%04.2f␣\n",
8          elem->count, elem->product,
9          elem->singleprice, elem->product_sum);
10 }
```

dynamische Listen (example, cont.)

Beispiel: Liste - die Ausgabe

```
# ./a.out
 Count Product                                    SP   SUM
 005   Club Mate, 0,5l Flasche                    0.75 3.75
 002   Wiener Wuerstchen, 200g Packung            2.45 4.90
 001   Senf, extra scharf                         0.56 0.56
```

Inhaltsverzeichniss

Ausblick

- Vormittags
  - Der Compiler und seine Features
  - Was sind Headerfiles, wie nutzt man sie
  - Unser Codeassistent: der Präprozessor
- Nachmittags
  - Fehler finden, debugging

Sie können den Computer jetzt wegwerfen...