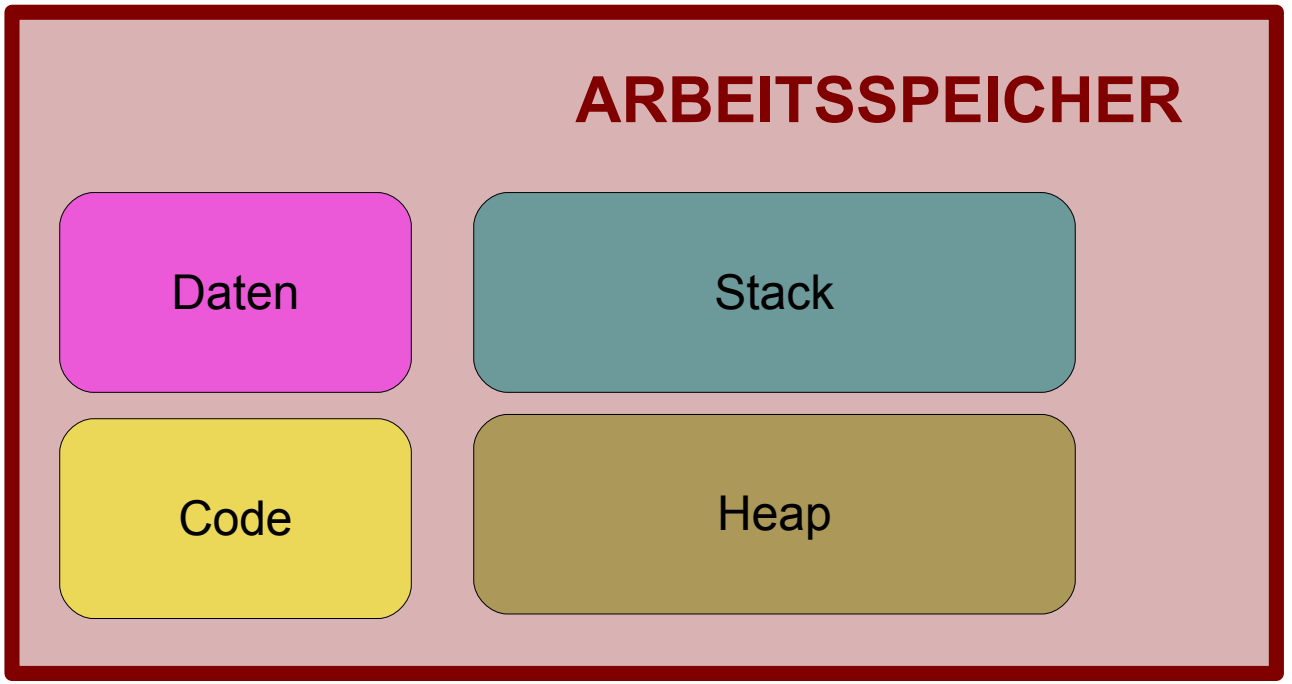
# Vorlesung 10

- Speicherbereiche
  - Aufruf einer Funktion
  - Rekursion und Iteration
- Zeitmessung in C++

# Speicherbereiche

- *Speicherbereiche in **laufenden** Programmen*



# Speicherbereiche

•**CODE:** Hier werden die Befehle des Programms geladen und daraus in den Prozessor übermittelt.

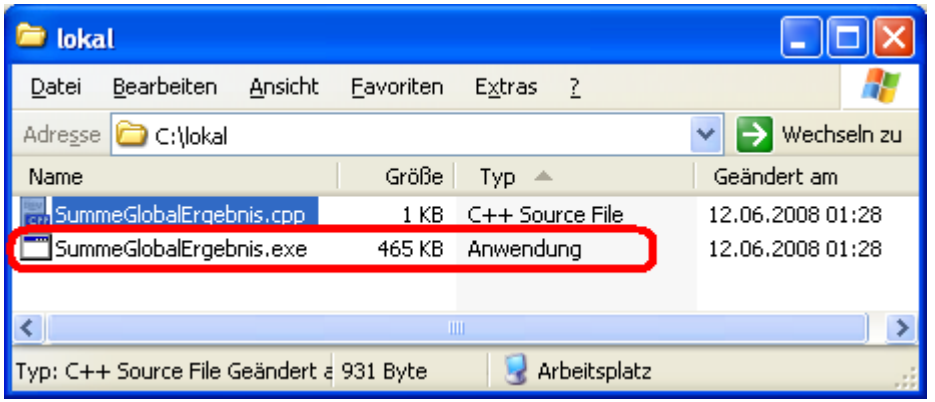
•**DATEN:** In diesem Speicherbereich bleiben die Daten, die verfügbar bis zum Ende des Programms sind (***globale und statische Variablen***)

•**STACK (Stapel):** Behält die Funktionsaufrufe und ***lokalen Variablen***. (Bis zum ca. 8 Megabytes)

•**HEAP:** Hier wird **dynamischen Speicherplatz** reserviert. (theoretisch bis zum ca. 4 Gigabytes)

# Speicherbereiche: Beispiel

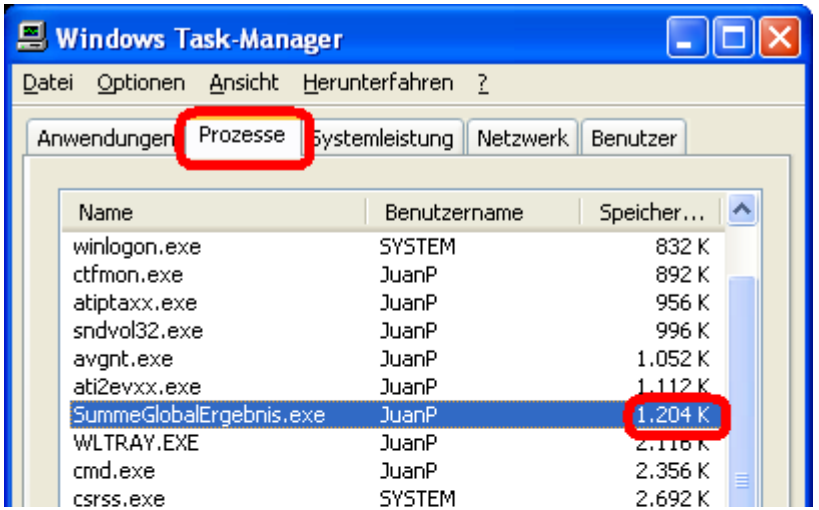
Vor der Ausführung ist das Programm in einem nicht flüchtigen Speichermedium (z.B. Festplatte, USB Flash Disk, Diskette, CD-Rom...) gespeichert




Beipiel: Dieses ausführbare Programm .exe nutzt ca. 500 KB von Speicher im Laufwerk C: (in der Festplatte).

# Speicherbereiche: Beispiel

Beim Ausführen kann man sehen jedes laufende Programm wie ein Prozess im *Windows Task Manager* (Strg + Alt + Entf).
Dafür muss das ausführbare Programm .exe in Arbeitsspeicher aufgeladen (**Bereich Code**) und Speicherplatz für Variablen (**Bereich Stack, Heap, Daten**) reserviert werden.




Beipiel: Dieses ausführbare Programm .exe nutzt ca. 1200 KB von Speicher in Arbeitsspeicher beim Ausführen.

# Weitere Speicherbereiche: Beispiele

```
int c; // Globale Deklaration: Außerhalb des mains

void summe(short, short); /*Kopf der Funktion (Deklaration)*/

int main(void)
{
 short d, e;
 /* c wird nicht im main deklariert */
 d = 3000; e = 5200;  /* Initial.*/

 summe(d,e); /* Aufruf */
 cout << „Die Summe ist  „ << c; /* Ausgabe auf dem Bildschirm
*/
 return 0;
}
/*Körper der Funktion (Definition)*/
void summe (short a, short b)
{ ;
/* c wird nicht in der Funktion deklariert */
  c = a + b;
  return;
}
```

**Die globale Variable wird im Bereich DATEN gespeichert.**

**Die lokalen Variable werden zusammen mit dem Aufruf der Funktion im STAPEL gespeichert.**

SummeGlobalErgebnis.cpp

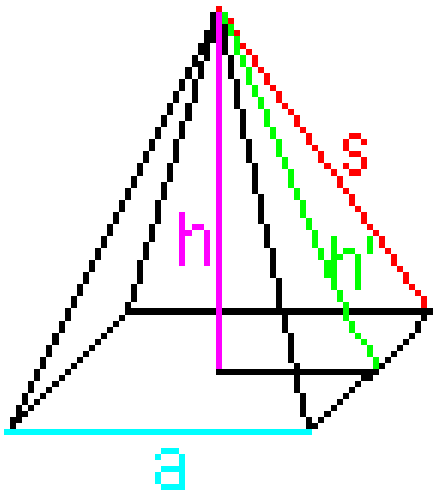
# Der Stapel

- Nacheinander Aufrufe einer Funktion

# Vom Problem zum Programm

***Problem:*** Es soll ein Programm entwickelt werden:

1. die Werte $a_1, h_1$ $a_2, h_2$ werden eingelesen
2. die Werte `volumen1` $\frac{1}{3} a_1^2 h_1$ und `volumen2` $\frac{1}{3} a_2^2 h_2$ werden ausgegeben.



# Funktion: Übergabe - Rückgabe


**Was für Formel?**

# Besonderheit: Grundfläche

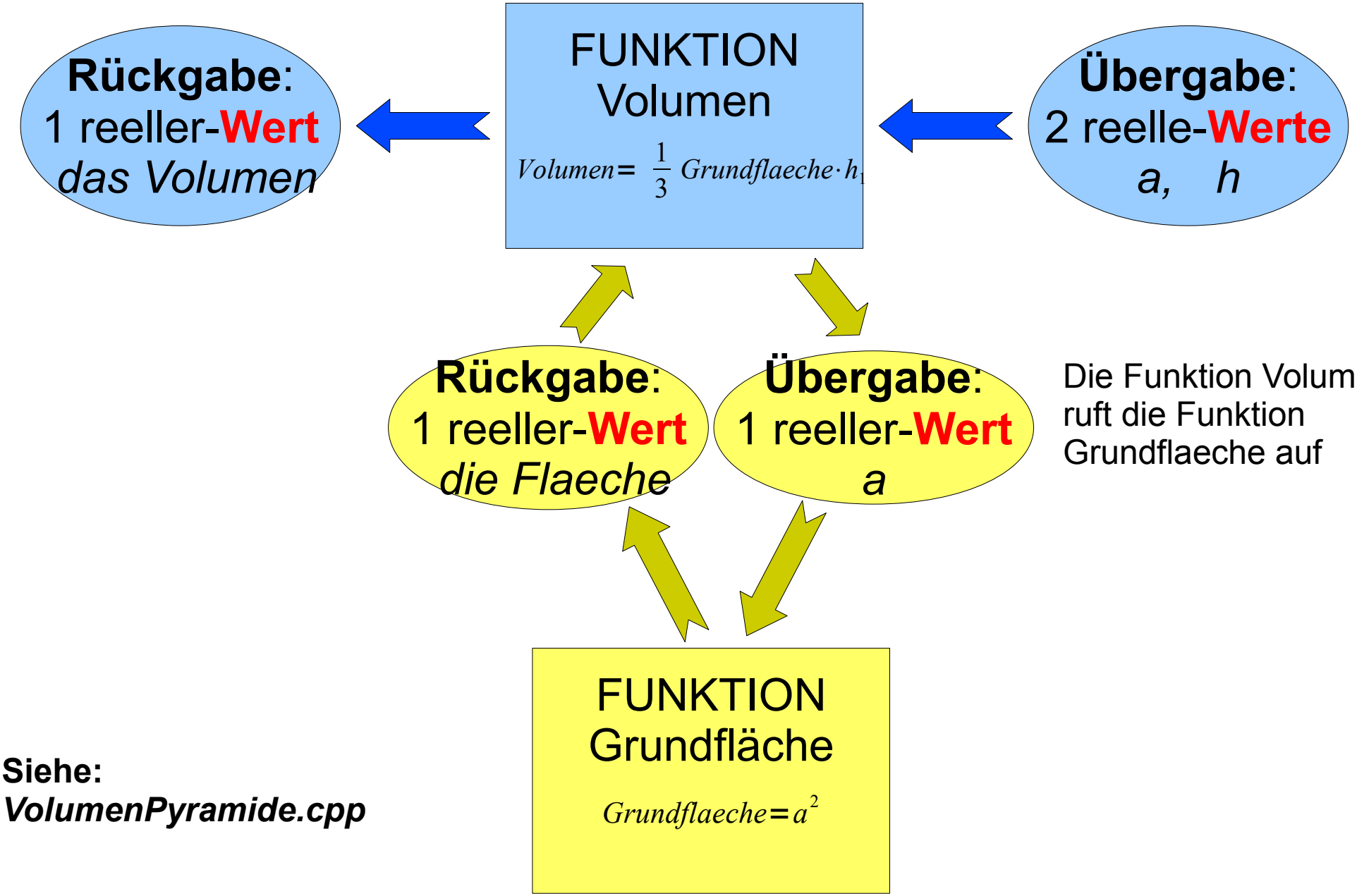
Wir haben eine Funktion zur Berechnung der Grundfläche!

$$Grundflaeche = a^2$$

Also können wir jetzt das Volumen dank der Grundfläche berechnen.

$$Volumen = \frac{1}{3} Grundflaeche \cdot h_1$$

# Funktion: Übergabe - Rückgabe




Die Funktion Volumen ruft die Funktion Grundflaeche auf

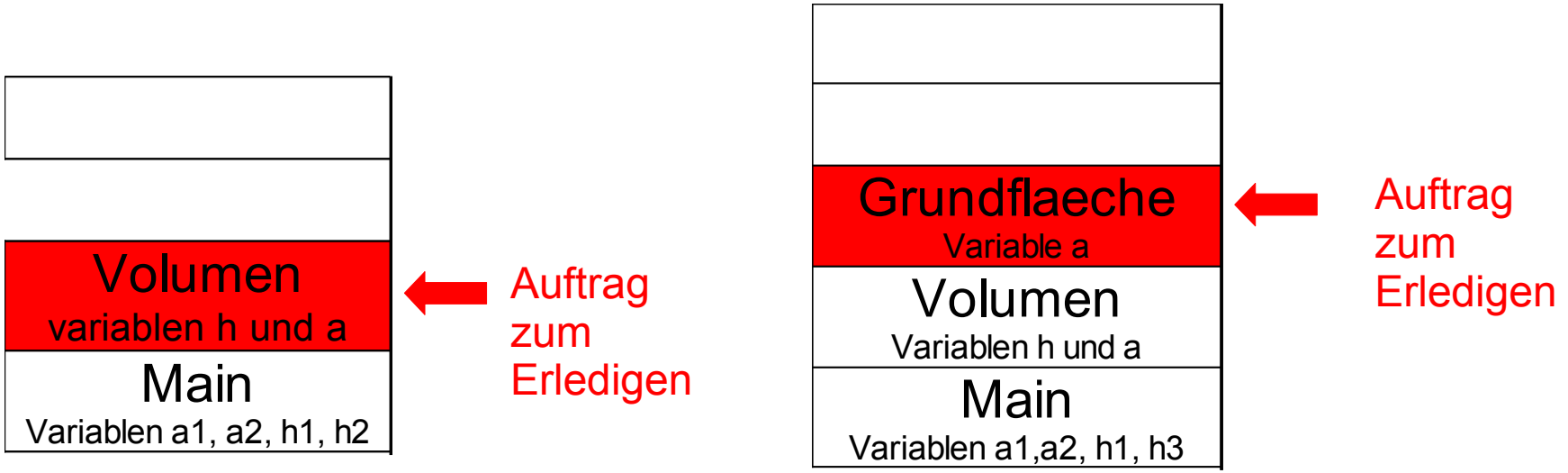
**Siehe:**
***VolumenPyramide.cpp***

# Der Stapel: Eine Funktion ruft eine Funktion auf

- die **Aufrufe** der Funktionen kann man als Aufträge zum Erledigen auf einem Stapel veranschaulichen.

$$Volumen = \frac{1}{3} Grundflaeche \cdot h_1$$

$$Grundflaeche = a^2$$



# Der Stapel: Eine Funktion ruft eine Funktion auf

- Der Stack (Stapel) ist nicht nur eine Veranschaulichung sondern ein Speicherbereich, der wie einer Stapel organisiert ist.
- Nachfolgend wird der Stack dargestellt

# *Der Stack (Der Stapel)*

Wichtige Aufgaben:

- Lokalen Variablen Reservierung.
- Aufruf der Funktionen.

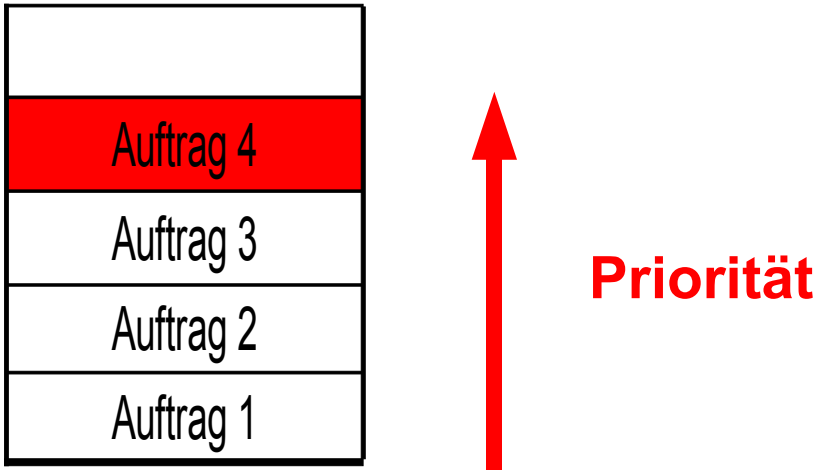
Auch genannt
***Stapelspeicher*** oder ***Kellerspeicher.***

# Der Stack (Der Stapel)

**STACK (Stapel):** Behält die Funktionsaufrufe und ***lokalen Variablen***.

Dessen Funktionsweise kann uns dabei helfen,

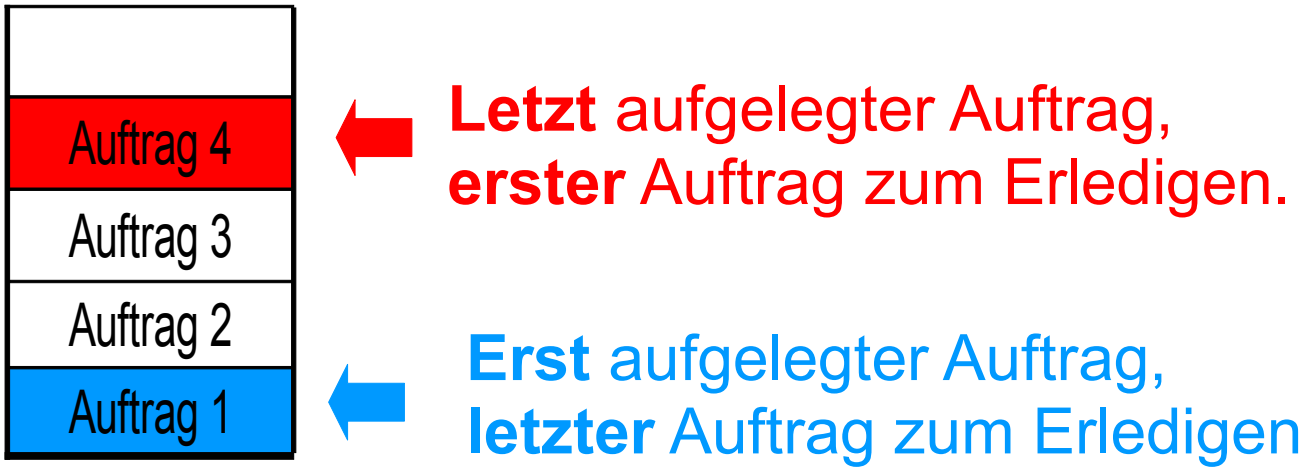
- die **Aufrufe** einer Funktion besser zu verstehen.
- warum die Variablen **Gültigkeitsbereich** haben.



# Der Stack : LIFO

Die letzten Aufträge, die auf den Stapel gelegt wurden, sind die ersten, die erfüllt werden müssen.

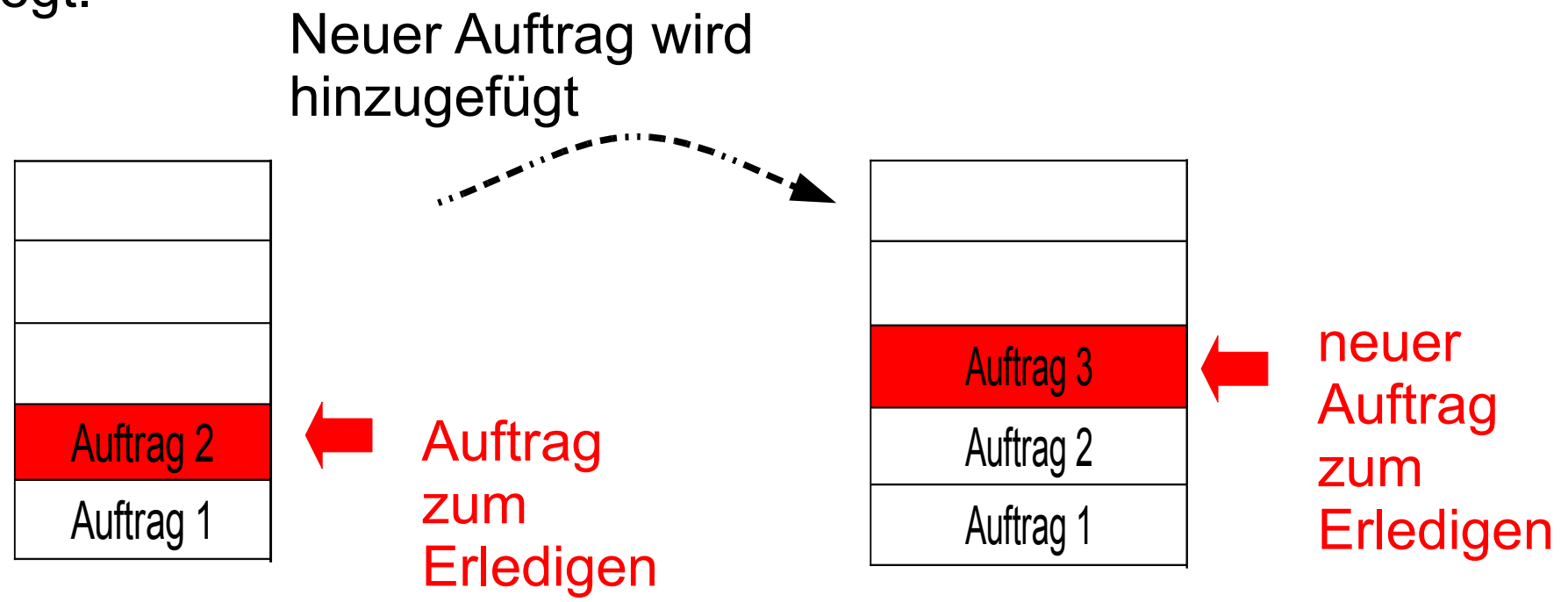
Der Fachname für so eine
**LIFO:** Last In First Out (Erstes Hinein Letztes Heraus)



# Neuer Auftrag: stapeln

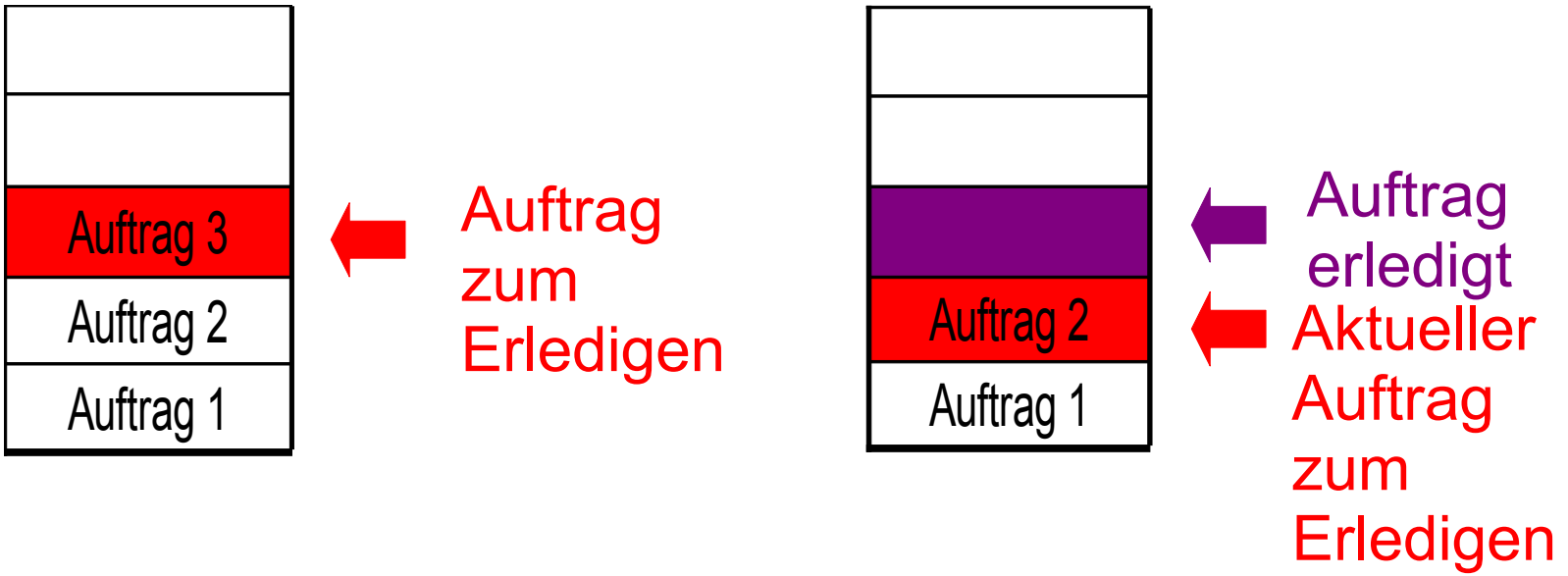
•Der Stack enthält verschiedene gestapelte Aufträge, die müssen in dieser **Reihenfolge** erledigt werden.

•Neue Aufträge zum Erledigen werden immer auf dem Stapel gelegt.



# Auftrag erledigen: entstapeln

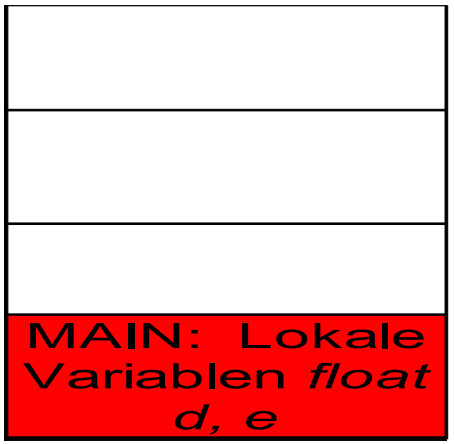
• Wenn der Auftrag **erledigt** ist, wird er aus dem Stapel **entfernt**.



# Der Stack: Funktionsaufruf

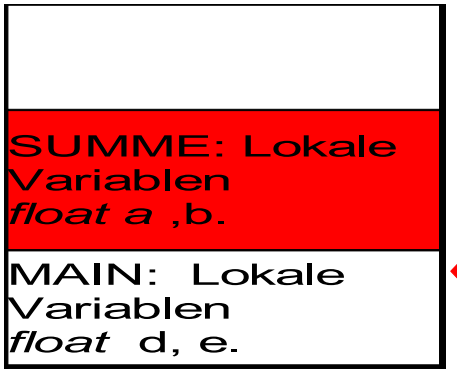
*Praktische Anwendung im laufenden Programm*

`SummeGlobalErgebnis.cpp`




**Am Anfang: Ausführung MAIN**

Auftrag zum erledigen




**Beim Aufruf der Funktion Summe**

Auftrag zum Erledigen

# Iteration

- Die **Iteration** ist der Begriff für ein Verfahren, das wiederholt wird, um ein bestimmtes Ergebnis zu erreichen oder sich anzunähern.

  Eine Iteration in Informatik besteht deshalb wesentlich aus einer **Schleife**.

Beispiel: Fibonacci-Zahlen `f(n) = f(n-1) + f(n-2)`

1, 1, 2, 3, 5, 8, 13, 21, 35,...

werden durch eine Iteration (Schleife) berechnet.

***Siehe Fibo_iterativ.cpp***

# Rekursion

- Als Rekursion versteht man ein Begriff oder Verfahren, das durch sich selbst definiert ist.

  ***n!*** = *n* * ***(n-1)!*** *Dies ist eine rekursive Definition der Fakultät.*

  *aber n! = 1, wenn n=1*

  In Informatik betrifft der Begriff Rekursion u.a. die Funktionen. Eine **rekursive Funktion** ist eine Funktion, die **sich selbst** aufruft. Unten ist die rekursive Definition der Fakultaet als Funktion implementiert.

```
fakultaet (n)
{
  if (n == 1)
  return 1;
  else
   return n*fakultaet(n-1);
}
```

# Iteration und Rekursion

- Generell gilt:
  - Alle rekursive Verfahren können als eine Iteration ausgedrückt werden.
  - Ein iteratives Algorithmus läuft in der Regel schneller als ein rekursives. Rekursiv ist öfters einfacher zu programmieren und eleganter.
- Die Fakultät als iterative Funktion (Schleife)

```
fakultaet (n)
  {int i, f=1;

    for (i=2; i <=n; i++)
    f = f * i;

    return f;

  }
```

***Siehe Fibo_rekursiv.cpp***

# Funktion,die sich selber aufruft:Die Rekursion

***Noch ein Beispiel***

Aufgabe:Schreiben Sie ein Programm,das die Berechnung: $X^y$ durch rekursiven aufruft realisiert.

$$X^y = X * X^{y-1}$$

***und***

$$X^1 = X$$

Voraussetzung: $X^y$ Potenz für ganze Zahlen

# Funktion,die sich selber aufruft:Die Rekursion

## Lösungsweg:

Für die Berechnung von $X^y$ wird eine Funktion verwendet, der X und y mit den Zahlenwerten übergeben wird.

***int Potenz (int x, int y);***

# Funktion,die sich selber aufruft:Die Rekursion

## Lösungsweg:

```
int Potenz (int x,int y)
{
     if(y==1)
    return x;
    else
     return (x* Potenz(x,y-1));
}
```

$$X^y = X * X^{y-1}$$

*und*

$$X^1 = X$$

# Funktion,die sich selber aufruft:Die Rekursion

Lösungsweg:

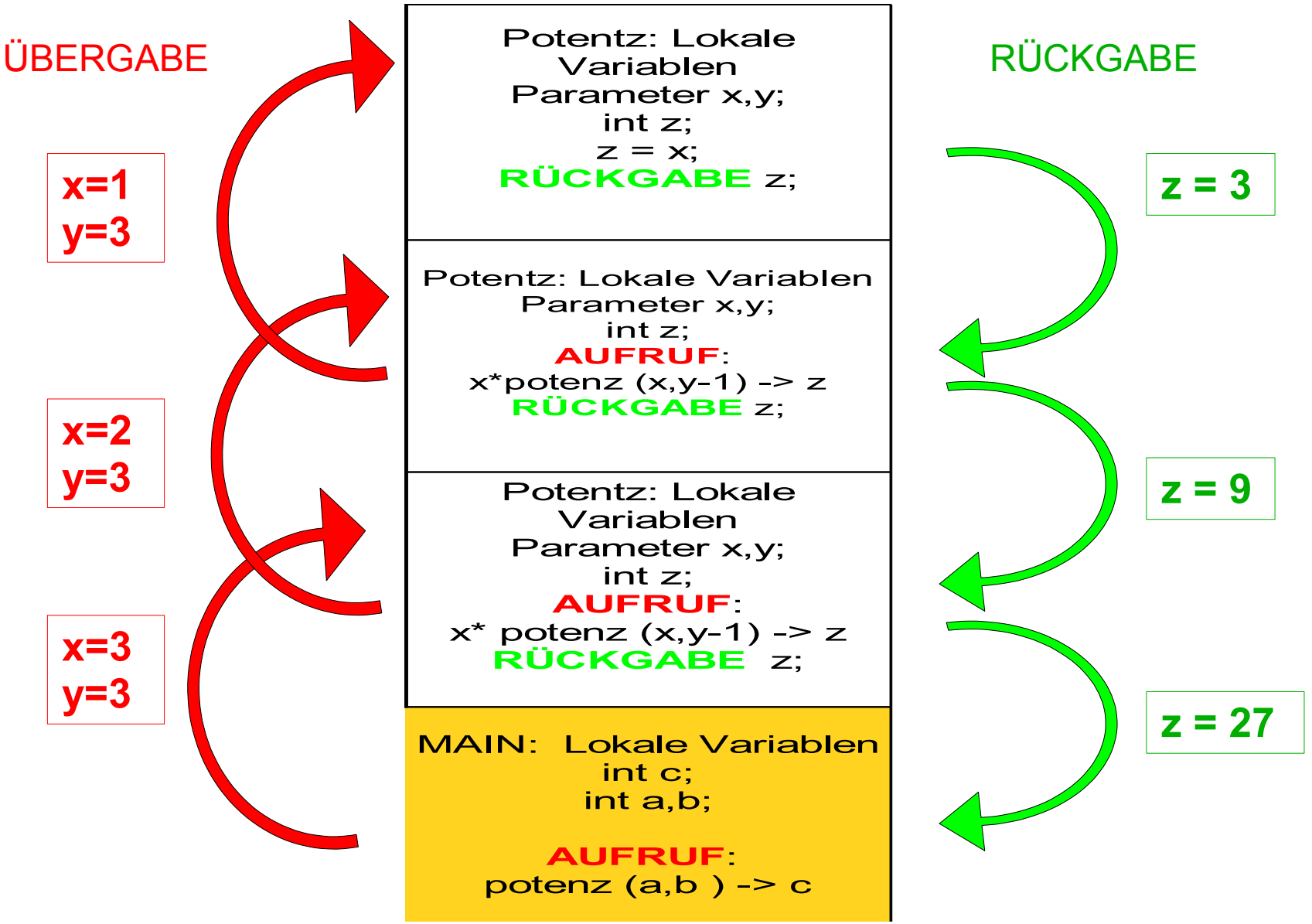
Für y=3 und X=3 ergibt sich folgendes Aufrufschema und als Ergebnis 27:

1.Power(3,3) --->return 27;

2.Power(3,2) --->return 9;

3.Power(3,1) -->return 3;

# Rekursion: Stapel Blick



# Iteration und Rekursion: Beispiel

Zwei Implementierungen der Funktion Potenz: iterativ und rekursiv die werden anhand 2 Projekte in gleichem Hauprogramm ausgeführt.

- Die Funktion Potenz iterativ ist in der Datei *Fibo_iterativ.cpp* implementiert
- Die Funktion Potenz rekursiv ist in der Datei *Fibo_rekursiv.cpp* implementiert
- Das Hauptprogramm für die Berechnung der Potenz ist *MainPotenz.cpp*
- Mit diesem Programm können wir eine beider Funktionen Potenz (rekursiv bzw. iterativ) linken. Jeweils haben wir ein Projekt *ProjektPotenzIterativ* oder *ProjektPotenzRekursiv*