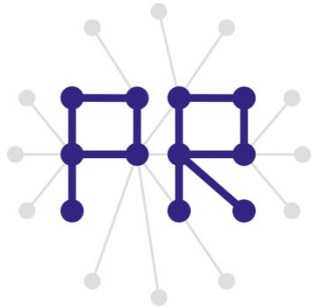
UNIVERSITÄT SIEGEN



Einführung in die Informatik I

Kapitel II.3: Sortieren

Prof. Dr.-Ing. Marcin Grzegorzek
Juniorprofessur für Mustererkennung im Institut für Bildinformatik
Department Elektrotechnik und Informatik
Fakultät IV der Universität Siegen

30.01.2013

Inhaltsverzeichnis

Rekursion

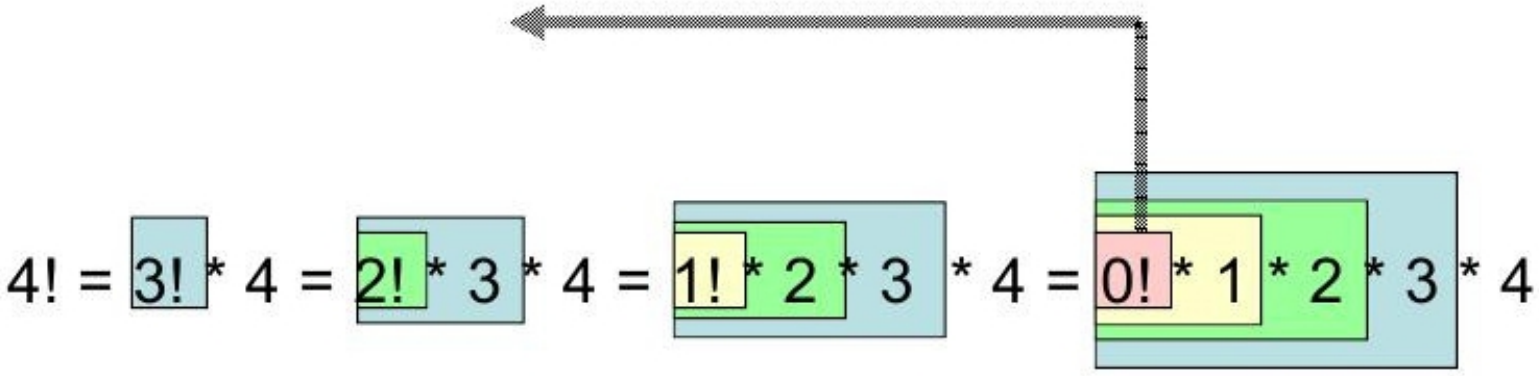
- **Begriff der Rekursion:**

  Funktionen die sich selbst aufrufen:

  - Direkt rekursiv: f() → f()
  - Indirekt rekursiv: f() → g() → f()

- Beispiel: Fakultätsfunktion
  - **`n! = 1 * 2 * … * (n-1) * n`**
  - oder : **`n! = (n-1)! * n`** (mit 0! = 1)

- Beispiel: 4! → für 4! muss 3! bekannt sein, usw....



Beispiel: Fakultät

- Fakultätsfunktion

$n! = (n-1)! * n$ mit $0! = 1$

- Quelltext

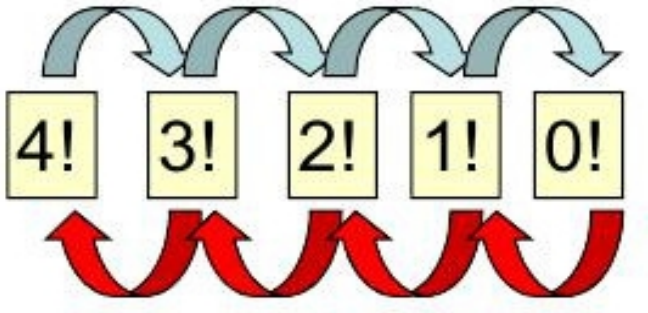
```
function y=fakult(x)
    if x==0
        y=1;
    else
        y=fakult(x-1)*x;
    end
```

Rekursion

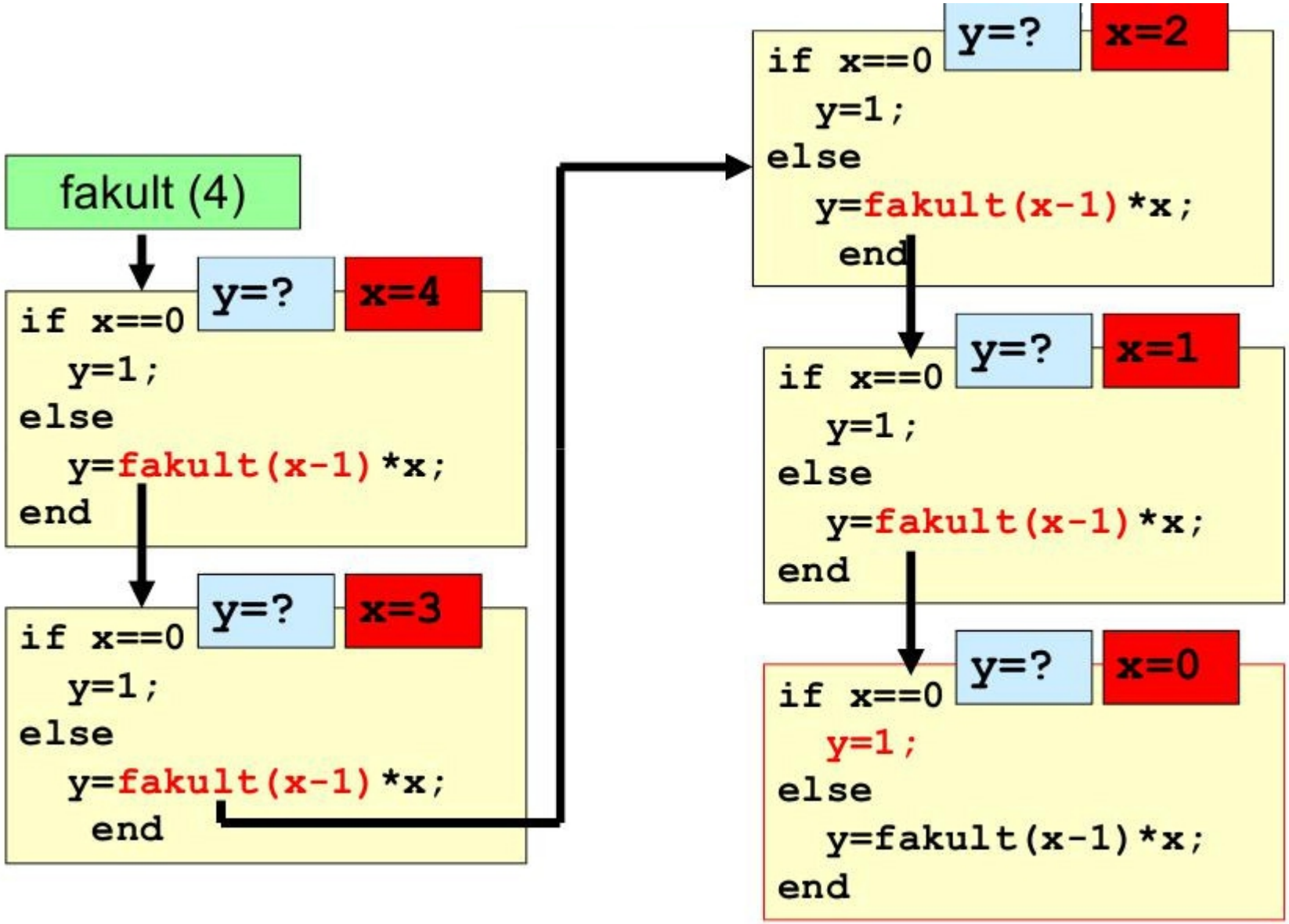
- Frage: Was passiert beim rekursiven Aufruf?

Rekursion: Mehrere Funktionen

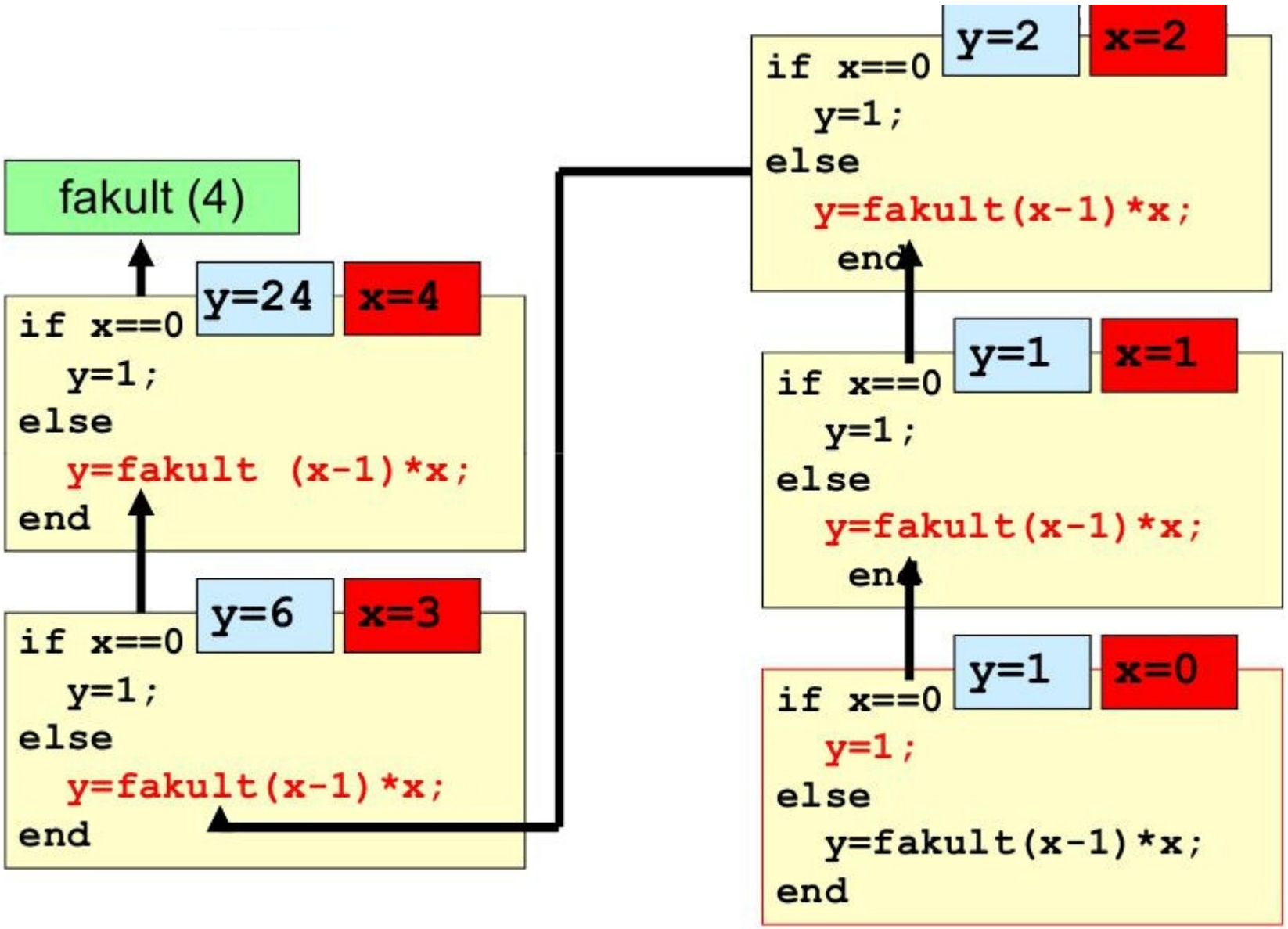
- Beim rekursiven Aufruf ruft sich eine Funktion selbst auf
  - Die **aktive** Kopie der Funktion wird **angehalten**
  - Die **neue** Funktion wird im Speicher angelegt und erhält ihre **eigenen lokalen Variablen**
- Das Verfahren wird so lange wiederholt, bis eine Funktion ohne Rekursion ein Ergebnis berechnen kann
- Danach geben alle rekursiv aufgerufenen Funktionen ihr **Ergebnis an die anrufende Funktion** zurück, bis da Endergebnis vorliegt



Funktionsweise einer Rekursion (1)



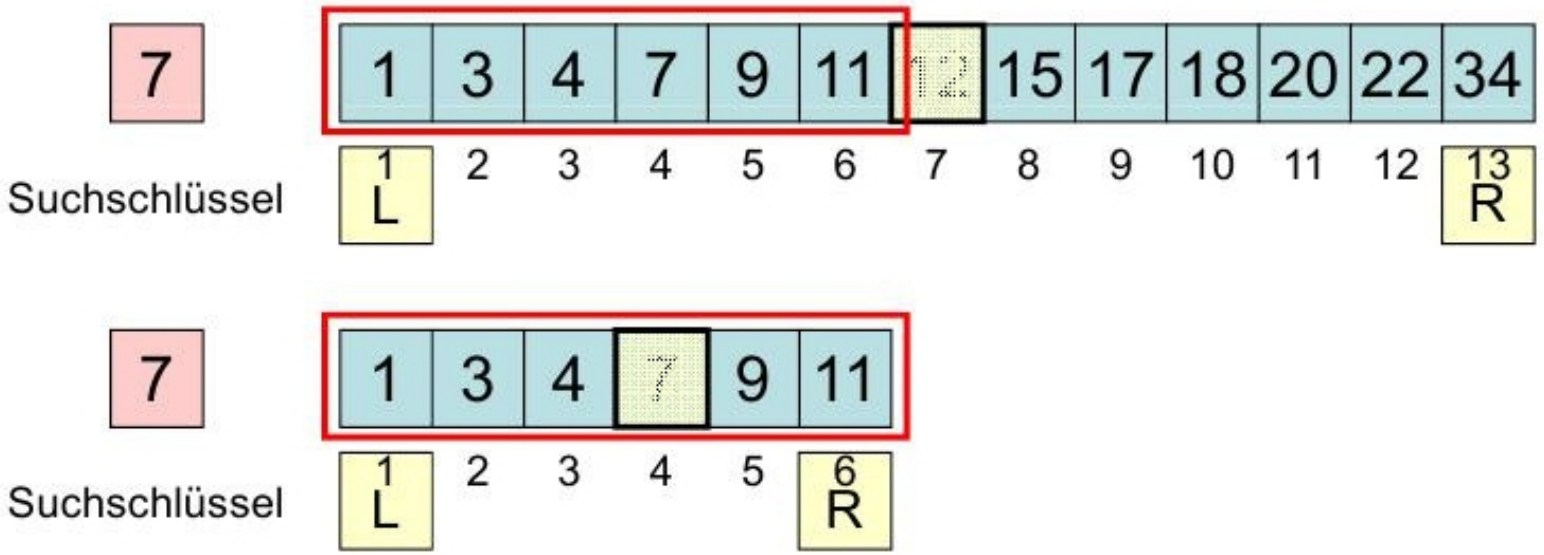
Funktionsweise einer Rekursion (2)



Rekursive Version der binären Suche

- **Divide-and-Conquer** Verfahren lassen sich rekursiv definieren:

  Der Teil, in welchem die Lösung liegt wird als neues Feld aufgefasst und wieder durchsucht



Quelltext

```
function Position=Suche_BiSec(x,Key,Links,Rechts)
Position = 0;                    % Default
if Links<=Rechts
    Mitte = floor((Links+Rechts)/2);
    if     x(Mitte)==Key % gefunden..
        Position = Mitte;
        break;
    elseif x(Mitte)< Key % nicht gefunden..
        Position=BiSec(x,Key,Mitte+1,Rechts);
    else
        Position=BiSec(x,Key,Links,Mitte-1);
    end;
end;
```

Beispielaufruf:

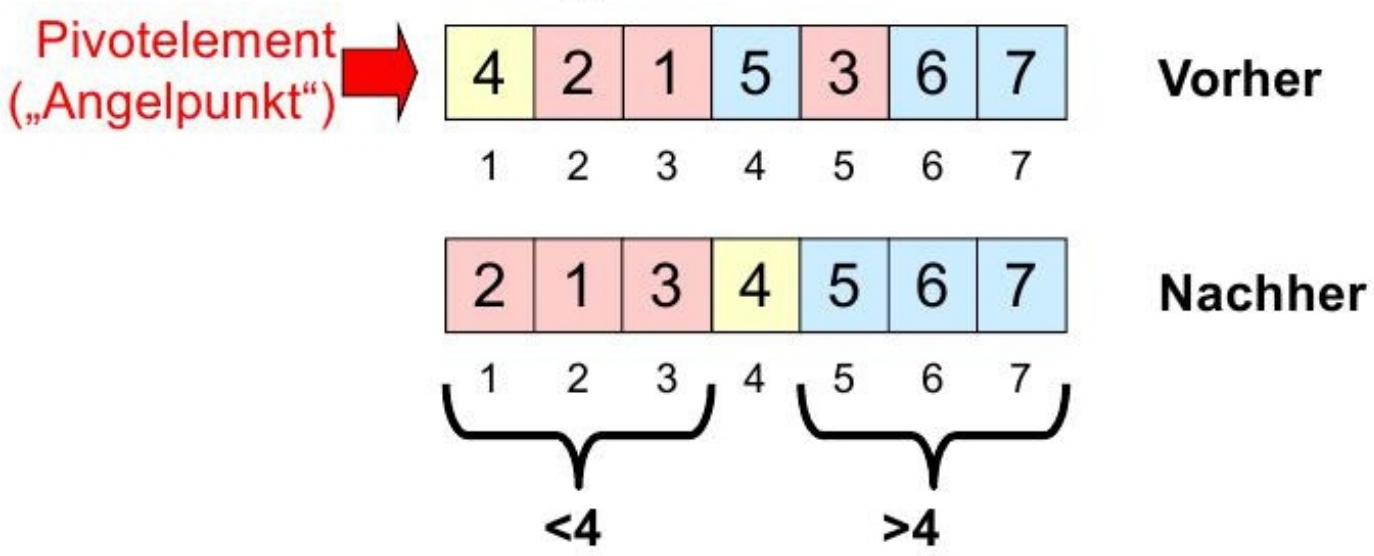
```
x=1:12:100
Suche_BiSec (x,73,1,length(x))
```

Quicksort

- Alle bisher vorgestellten Algorithmus haben eine **quadratische** anwachsende **Rechenzeit**
- Der erste wirklich **schnellere Algorithmus** (Quicksort) wurde 1960 von C.A.R. Hoare entwickelt
- Dieser Algorithmus gehört zu den am **besten erforschten Sortieralgorithmen**
  - Relativ einfach zu implementieren
  - Erfordert wenig Ressourcen
  - Sehr schnell
- **Rekursiver** Algorithmus

Partitionierung im Quicksort

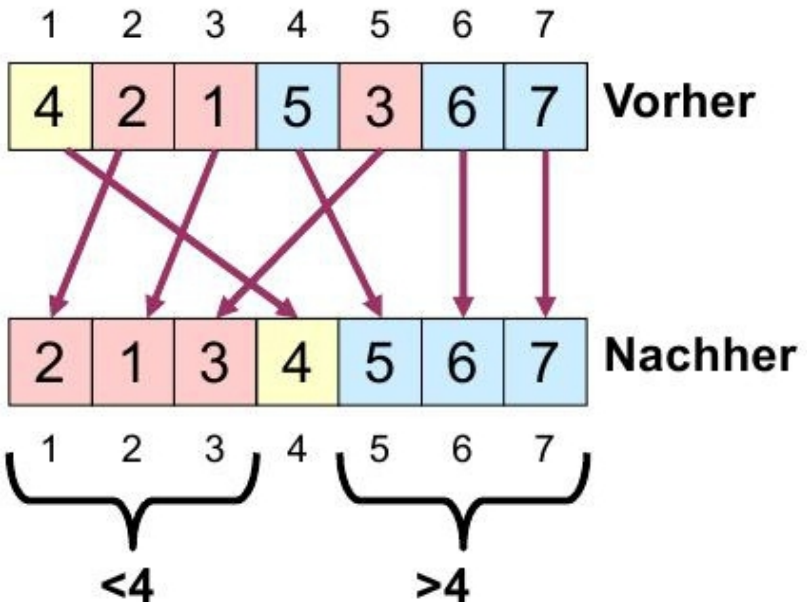
- Quicksort beruht auf einem **Zerlegen** (Partitionierung) des Feldes in **zwei Teile**
- Anschließend werden diese Teile **unabhängig** voneinander **sortiert**
- Der entscheidende Teil ist die **Partitionierung**. Die Partitionierung bewirkt eine **Vorsortierung** der Liste mit Hilfe eines willkürlich gewählten **Pivotelements**:



Partitionierung

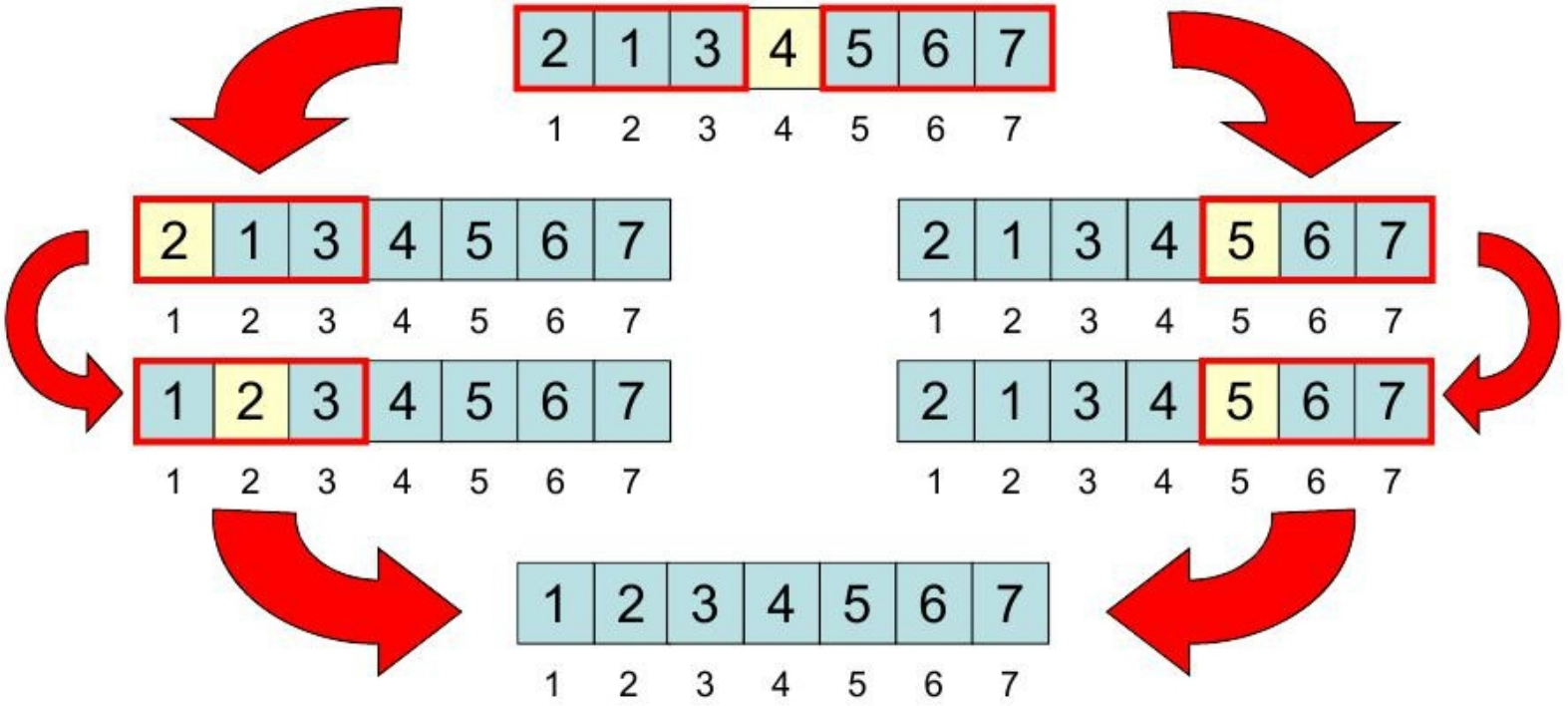
- Nach der Partitionierung soll gelten:
  - Das Pivotelement befindet sich an seinem bzgl. der Sortierung endgültigen Platz i
  - Alle Elemente in den Feldern 1 … i-1 sind kleiner oder gleich dem Pivotelement
  - Alle Elemente in den Feldern i+1 … n sind größer oder gleich dem Pivotelement

- Das Pivot-Element kann beliebig gewählt werden.
  Meistens ist es das erste oder das letzte Element



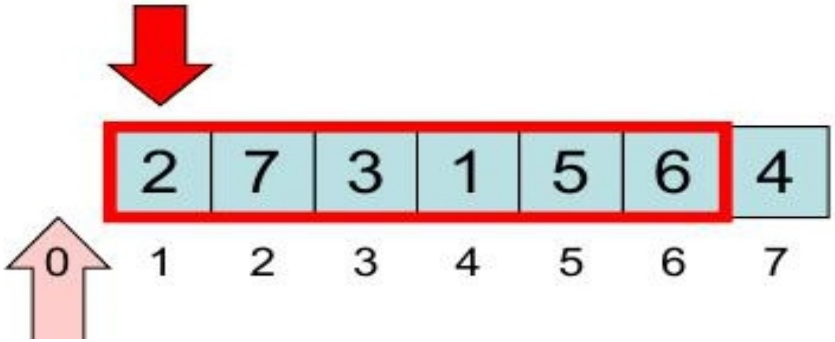
Rekursion

- Nach der Partitionierung wird der linke und rechte Teil nach dem gleichen Verfahren partitioniert
- Am Ende werden die Teile wieder zusammengesetzt



Implementierung der Partitionierung

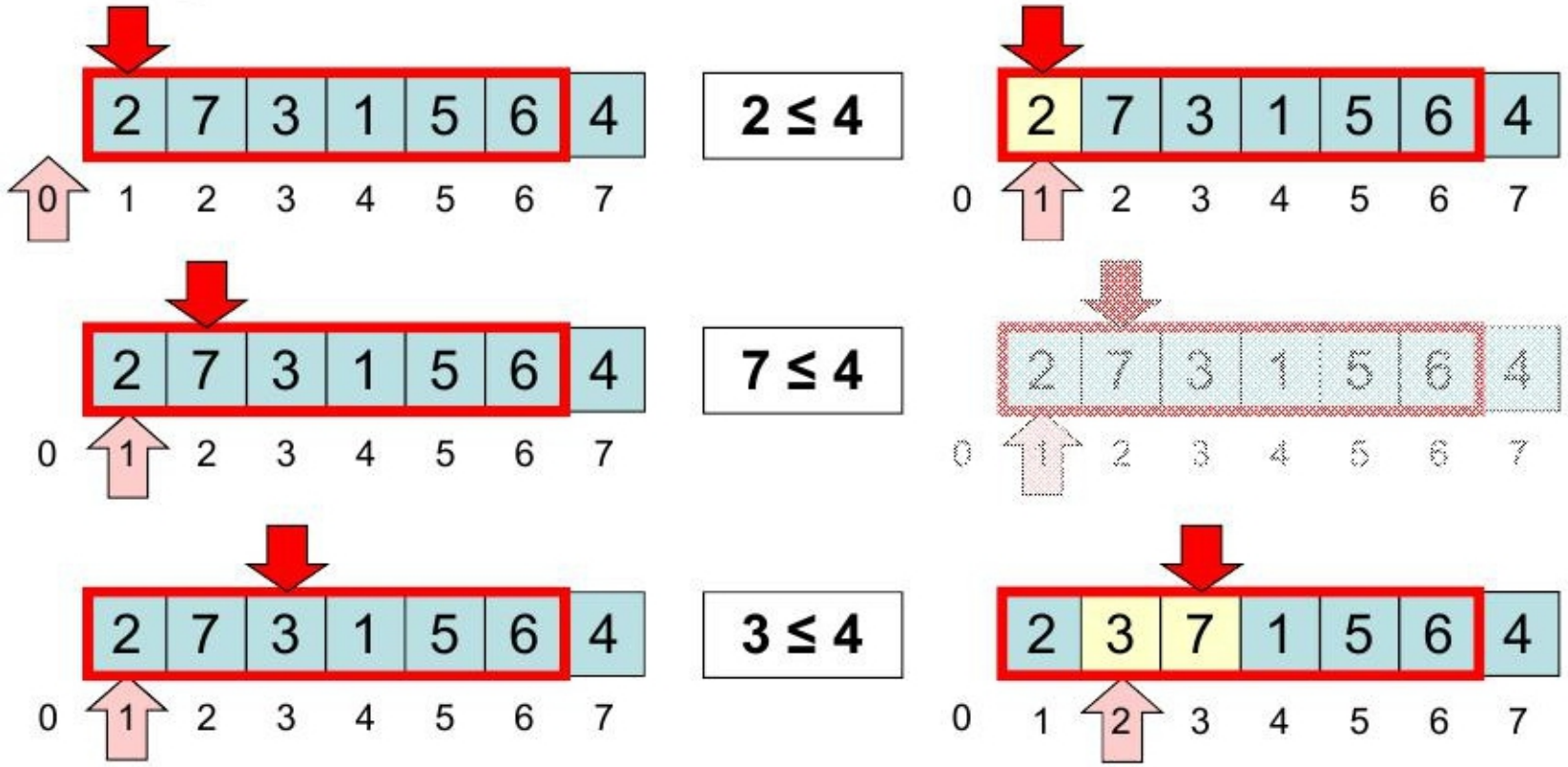
- Partitioniert werden soll zwischen Position 1 und 7
  - **`Links = 1`**
  - **`Rechts = 7`**
- Das Pivot-Element ist ganz rechts (frei gewählt)
- Schleife von **`Links`** bis **`Rechts-1`** (Pivot ist rechts)
- Ein Zeiger markiert das Ende der linken Liste mit Elementen, die sicher kleiner als das Pivotelement sind (Initialisierung mit Null)
- Ein Vergleichszeiger mustert die darauf folgenden Einträge



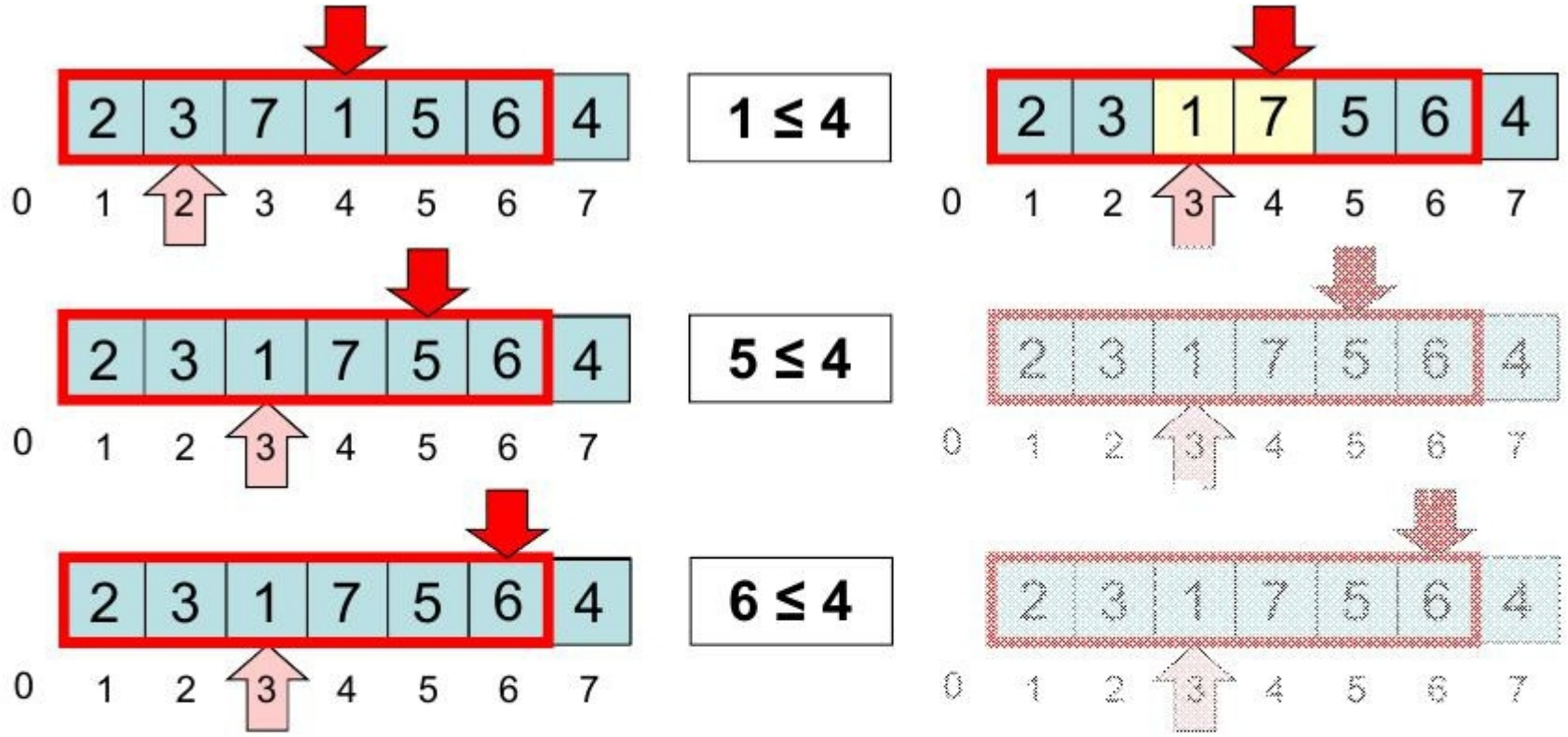
Implementierung der Partitionierung

- In Schleife:

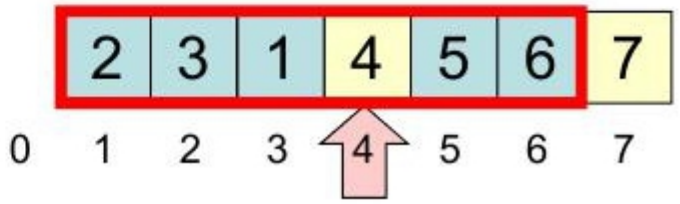
  Vergleich <= Pivot → Tauschposition +1 → Tauschen



Implementierung der Partitionierung



- Nach der Schleife:
  - Tauschposition + 1
  - Tauschen mit Pivot-Element



Quelltext: Partitionierung

```
function [x,swap] = partition(x,left,right)
pivot = x(right);       % Pivot-Element
swap  = left-1;         % Tauschposition
```

```
% Partitionierung L < Pivot < Rechts
for pointer = left:(right-1)
    if x(pointer) <= pivot          % muss auf linke Seite ?
        swap       = swap + 1;      % Nächste Tauschposition
        tmp        = x(swap);       % Tausch...
        x(swap)    = x(pointer);    % ...
        x(pointer) = tmp;           % ...
    end
end
```

```
% Platziert das Pivot-Element in der Mitte.
swap     = swap+1;
tmp      = x(right);                % Tausch...
x(right)= x(swap);                  % ...
x(swap) = tmp;                      % ...
```

Quelltext: Rekursion

```
function x = quicksort(x,left,right)

% Erster Aufruf mit
% left  = 1
% right = length(x)

if left < right              % Nicht Terminierungsfall
    [x,q] = partition(x,left,right);
                             % q ist Pivot-Element
    x = quicksort(x,left,q-1);
                             % sortiere linke  Hälfte
    x = quicksort(x,q+1,right);
                             % sortiere rechte Hälfte
end
```

Laufzeitanalyse: Quicksort

- Quicksort braucht im Mittel $n \cdot ld(n)$ Schritte