

Verdrahtung 1

Andreas Koch
FG Eingebettete Systeme
und ihre Anwendungen
TU Darmstadt

- **Verdrahtungsproblem**
- **Flächenverdrahtung**
 - Lee's Algorithmus
- **Kanalverdrahtung**
 - Klassisches Modell
 - Einschränkungen
 - Modellierung
 - Left-Edge Algorithmus
- **Zusammenfassung**

■ Eingaben

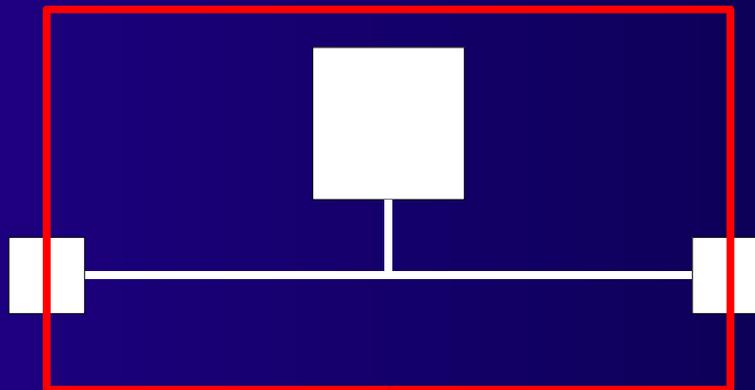
- Lage der Terminals (aus Platzierung)
- Zu verbindende Terminals als Netzliste
- Verdrahtungsfläche pro Layer

■ In der Regel: Zwei Phasen

- Globale Verdrahtung
 - ◆ Bestimmt die Lage ganzer Verdrahtungskanäle
 - ❖ Auf dem ganzen Chip
- Lokale Verdrahtung
 - ◆ Bestimmt den Verlauf einzelner Leitungen
 - ❖ Innerhalb eines Verdrahtungskanales

- **Anzahl der Verdrahtungslagen**
 - Abhängig von Technologie
 - bis zu 8 im kommerziellen Einsatz
- **Erlaubte Ausrichtung in einem Layer**
 - Nur horizontal oder vertikal, beides, 45°
- **Verdrahtung frei oder auf Raster**
- **Behandlung von Hindernissen**
- **Lage der Terminals**
 - Nur an den Grenzen der Verdrahtungsfläche?
 - Mittendrin?

- **Feste oder bewegliche Terminals**
- **Veränderliche Verdrahtungsfläche**
- **Vertauschbare Terminals**
 - z.B. NAND-Eingänge, LUT-Eingänge
- **Elektrisch äquivalente Terminals**
 - z.B. Duplizierte LUT-Ausgänge



Flächenverdrahtung 1

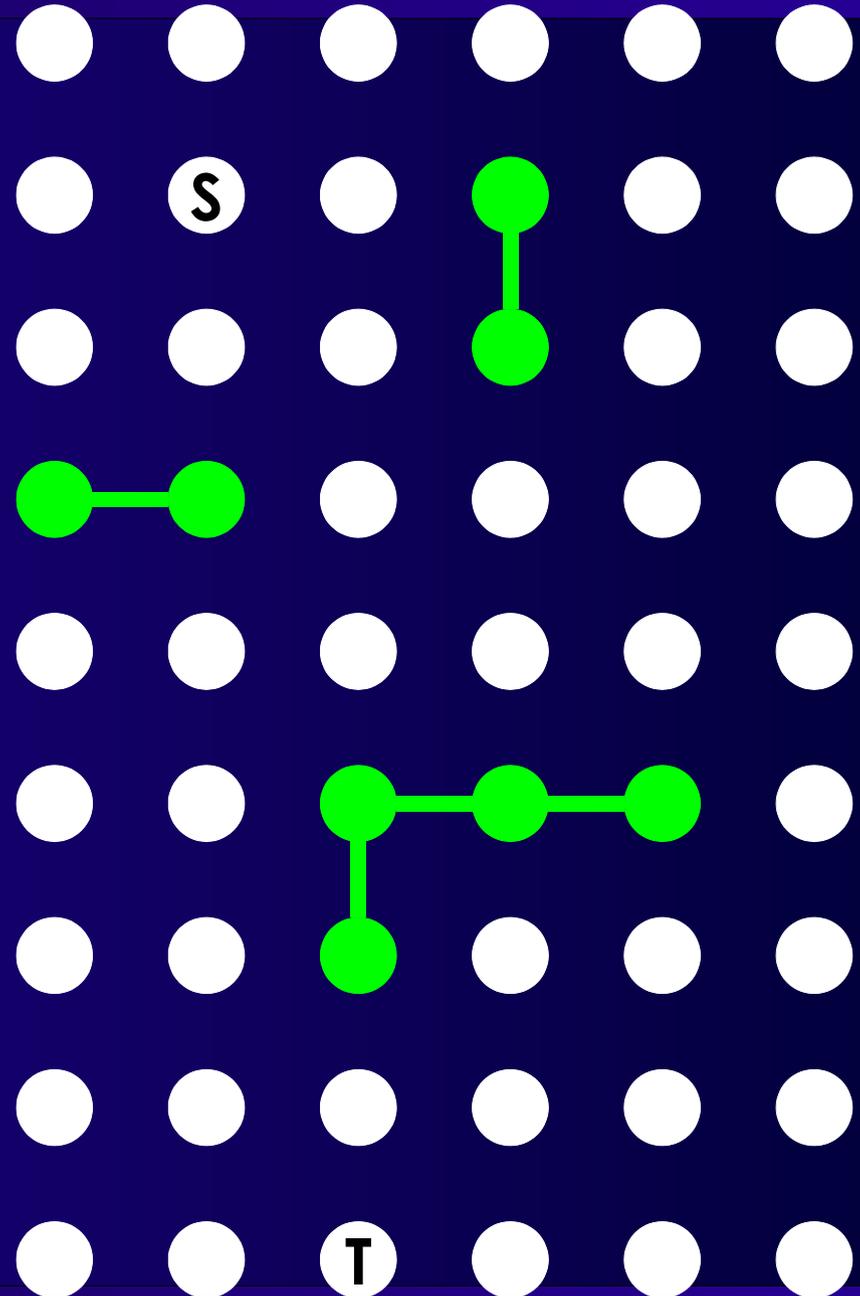
- **Terminals überall in Fläche erlaubt**
- **Algorithmus nach Lee (1961)**
 - Labyrinth-Verdrahtung (Maze Routing)
- **Berechnet**
 - Verbindung zweier Punkte auf Ebene
 - ◆ Quell-Terminal
 - ◆ Senke-Terminal
 - Findet kürzesten Pfad um Hindernisse herum
- **Arbeitet auf Raster**
 - Maß: Kürzester Abstand benachbarter Punkte

Flächenverdrahtung 2

■ Hindernisse

- Rasterpunkte
- Versperren Weg

■ Beispiel



Lee's Algorithmus 1

```
class grid_point : point {
    int value;
};
lee(grid_point S, grid_point T) {
    set<grid_point> wave, new_wave;
    grid_point neighbor, elem, path_elem;
    int label;
    /* 1. Schritt: Wellenausbreitung */
    new_wave := {S};
    label := 0;
    while (T ∉ new_wave) {
        ++label;
        wave := new_wave;
        new_wave := ∅;
        foreach element ∈ wave
            foreach neighbor ∈ N(element)
                if (neighbor.value == 0) {
                    neighbor.value := label;
                    new_wave := new_wave ∪ {neighbor};
                }
    }

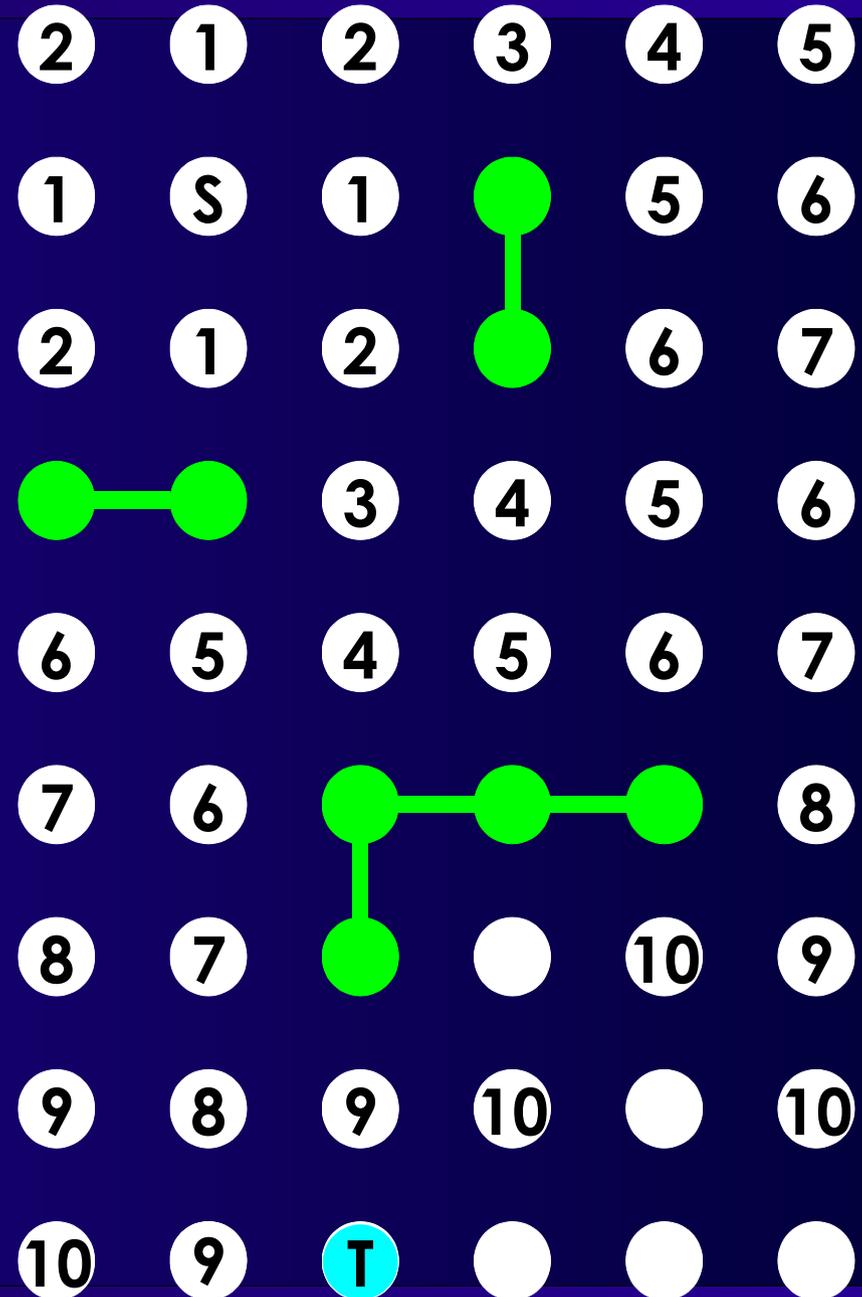
    /* 2. Schritt: Rückverfolgung */
    path_elem := T;
    for (i:=label-1; i ≥ 1; --i) {
        path_elem := "Nachbar mit value=i";
        /* ggf. Auswahlheuristik */

        /* Aktuelle Leitung nun Hindernis */
        path_elem.value := -1;
    }

    /* 3. Schritt: Aufräumen */
    foreach "point on grid"
        if (point.value > 0)
            point.value := 0;
}
```

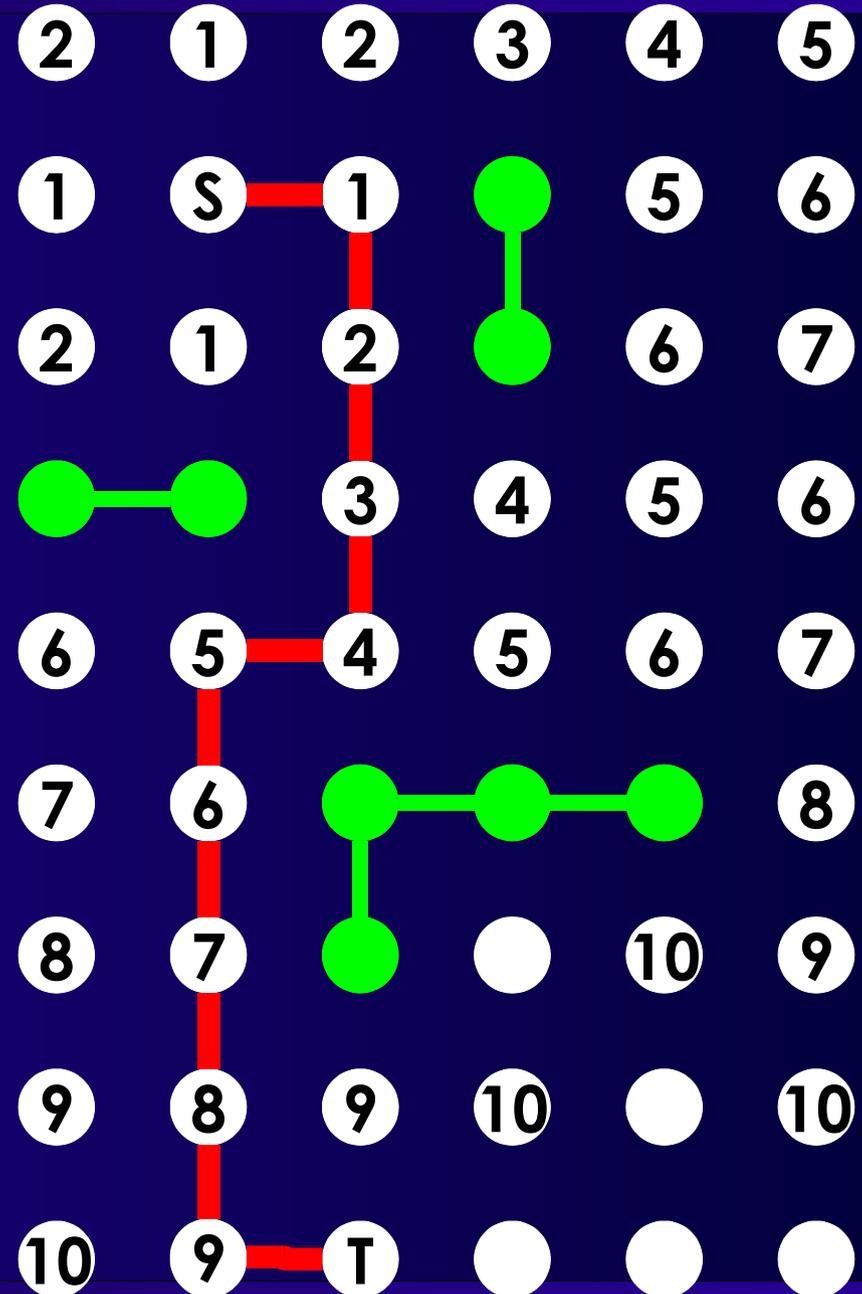
Lee's Algorithmus 2

■ Wellenausbreitung



Lee's Algorithmus 3

■ Rückverfolgung



Lee's Algorithmus 4

- Auf $n \times n$ Raster: $O(n^2)$, auch Speicher
- Erweiterungen
 - Mehrere Ebenen
 - ◆ Dreidimensionaler Ansatz
 - ❖ Höhere Kosten für Vias
 - Multi-Terminal Netze
 - ◆ Verdrahte zunächst zwei Terminals
 - ◆ Benutze dann gesamten Pfad als Quelle/Senke
 - ❖ Weitere Terminals werden an bestehende angeschlossen
 - ◆ Kürzester Pfad *nicht* mehr garantiert!
 - ❖ Wäre Minimaler Rechtwinkliger Steiner-Baum: NP-vollst.

Lee's Algorithmus 5

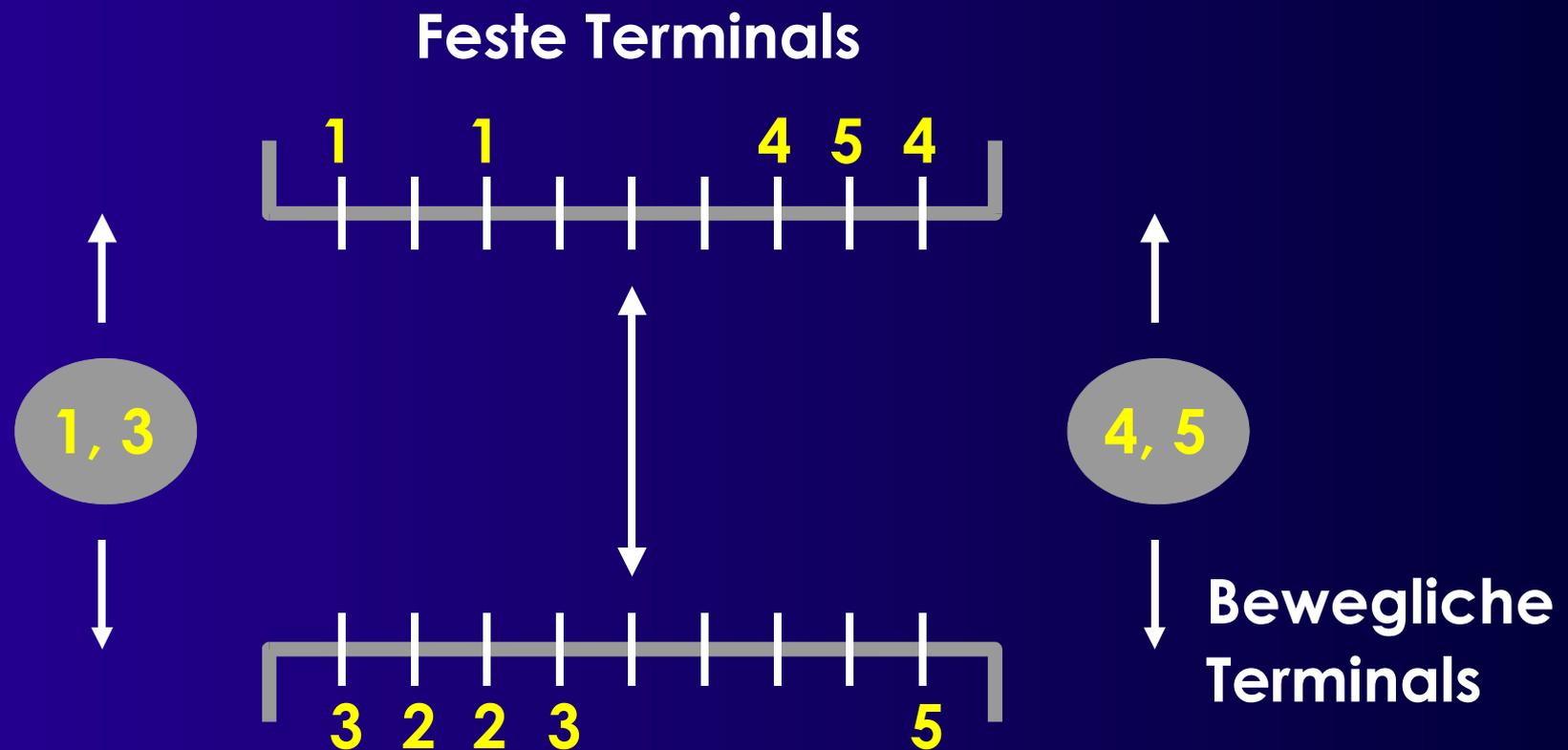
- **Hauptproblem: Sequentielles Vorgehen**
 - **Heuristiken**
 - **Priorisierung von Netzen**
 - ◆ Zeitkritische
 - ◆ Lange
 - ◆ mit hohem Fanout
 - ◆ ...
 - **Aber es existieren unlösbare Probleme**
 - **Unabhängig von Ordnung**
- **Verwendung bei iterativer Verbesserung**

Kanalverdrahtung 1

- **Lee's Algorithmus geeignet für**
 - Umgebung mit vielen Hindernissen
 - ◆ Wenige Pfade mit minimaler Länge
- **Schlecht geeignet**
 - Umgebung mit wenigen Hindernissen
 - Keine Auswahlmechanismen
 - ◆ Bestimmung des "besten" Pfades
- **Kanalverdrahtung**
 - Anfangs keine Hindernisse
 - Anderer Ansatz

Kanalverdrahtung 2

■ Verdrahtung von Netzen in rechteckigem Kanal



■ Ziel: min. Fläche, (min. Länge, min. Vias)

- **Variante: Switchbox-Verdrahtung**
 - Alle Terminals an allen vier Seiten fest
 - Alle Abmessungen fest
 - Entscheidungsproblem
 - ◆ Gibt es überhaupt eine Lösung?
 - ◆ Falls ja, optimiere sekundäre Ziele
 - ❖ min. Vias
 - ❖ min. Länge

■ Klassisches Modell

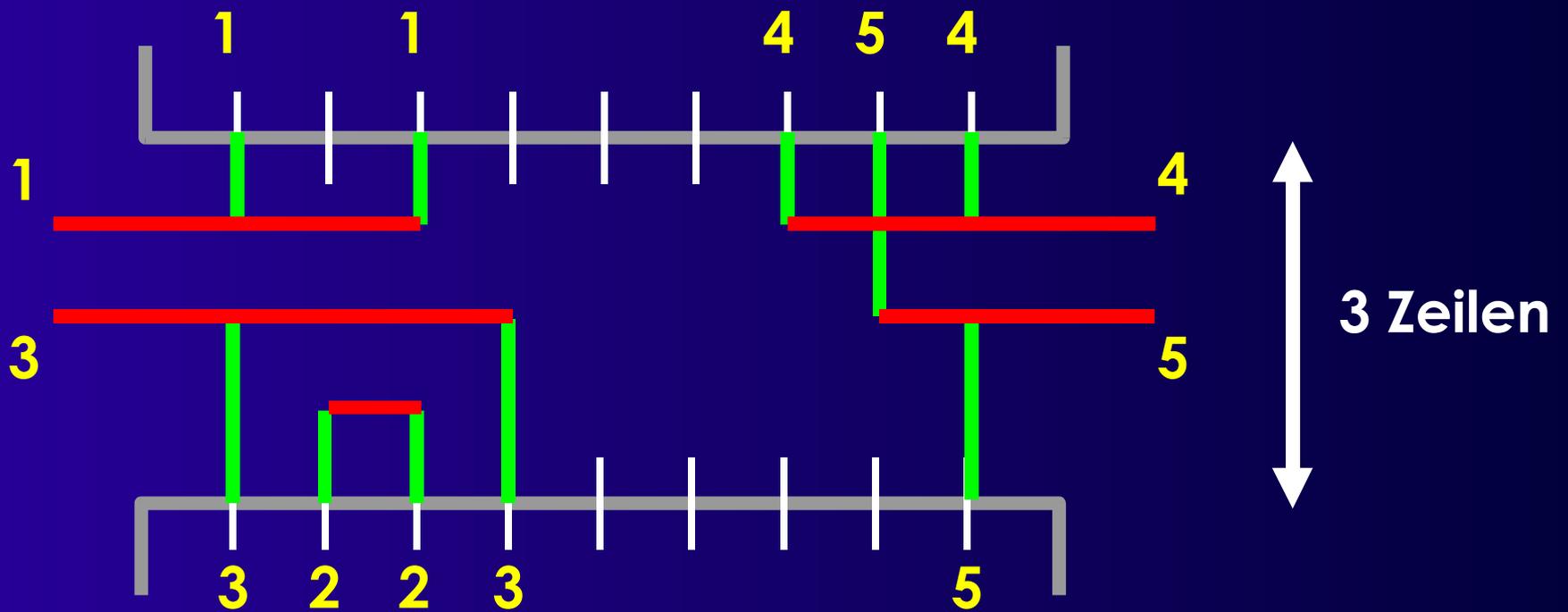
- Verdrahtung läuft auf Einheitsraster
- Zwei Verdrahtungsebenen
 - ◆ Getrennt für horizontale/vertikale Segmente
- Ein (1) horizontales Segment pro Netz
 - ◆ Ausnahme: Bei Konfliktauflösung 2 H-Segmente

■ Erweiterungen

- Verdrahtung ohne Raster
- 45° Verbindungen erlaubt
- Mehr als zwei Verdrahtungsebenen

Kanalverdrahtung 5

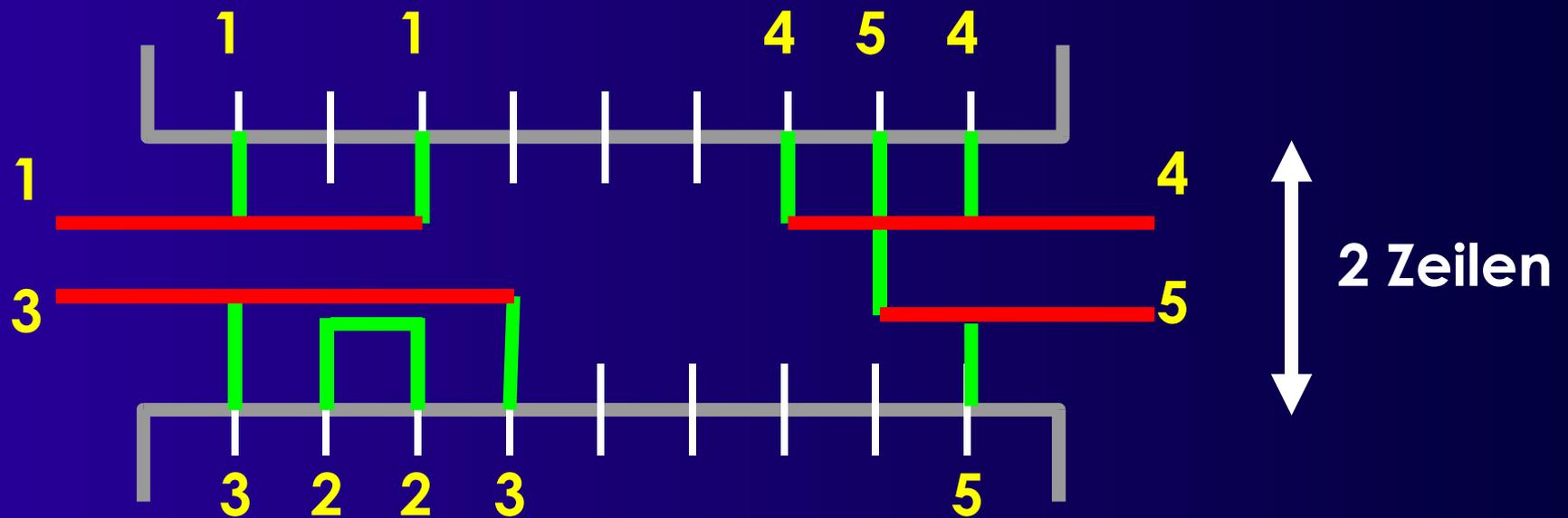
- Beispiel gelöst im klassischen Modell



- **Reservierte Ebenen für H/V-Segmente?**
 - Vermindern des Überschneidens zwischen überlagerten Segmenten
 - Kleinerer Lösungsraum
 - ◆ Schneller zu Lösen
 - ◆ Verlust an Qualität
- **Moderne Router**
 - Ohne reservierte Ebenen
 - Bessere Qualität

Kanalverdrahtung 7

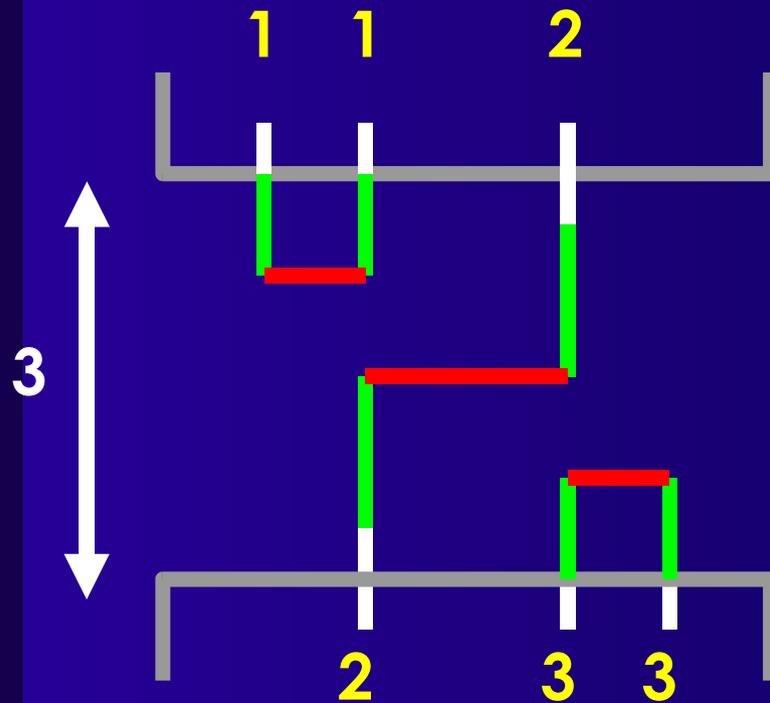
- Beispiel gelöst ohne reservierte Ebenen



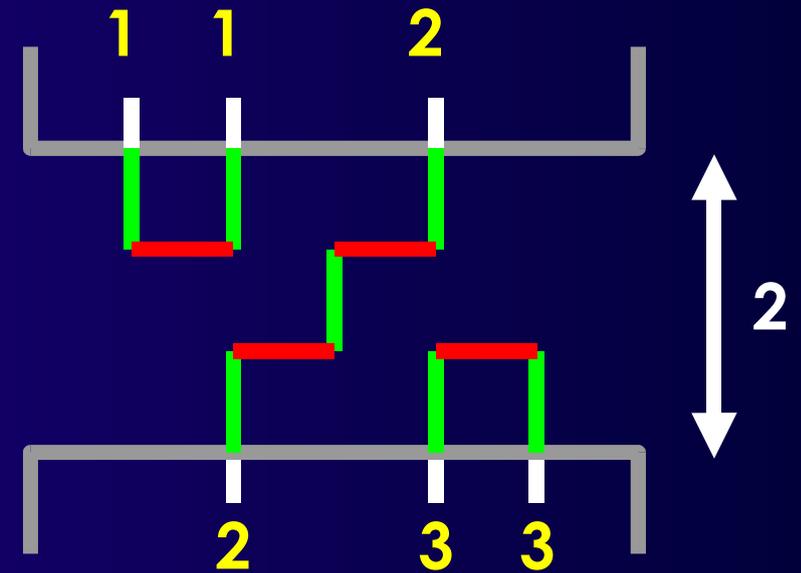
Kanalverdrahtung 8

■ Verwendung von doglegs

- Mehr als ein H-Segment pro Netz



Ohne Doglegs

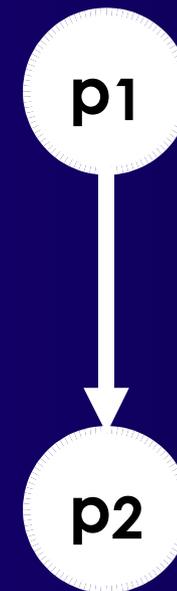
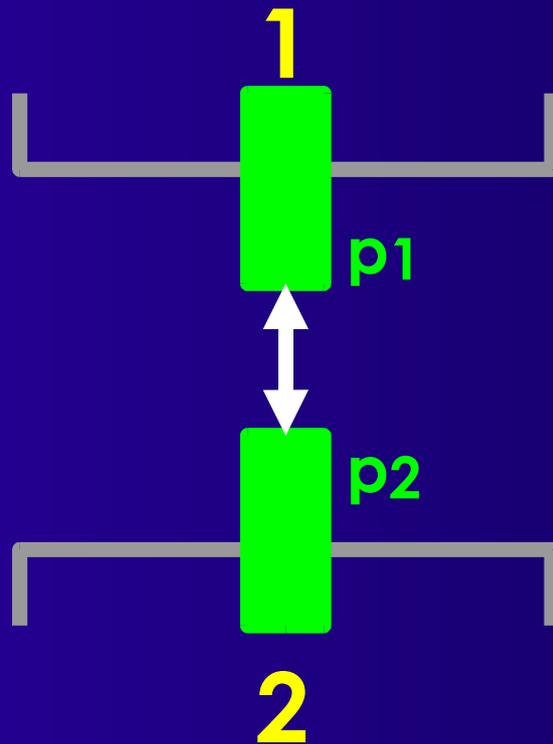


Mit Doglegs

Vertikale Einschränkungen 1

■ Zwei gegenüberliegende Terminals

- Oberes Segment in den Kanal *muß* über unterem Segment in den Kanal liegen
 - ◆ Sonst Kurzschluß

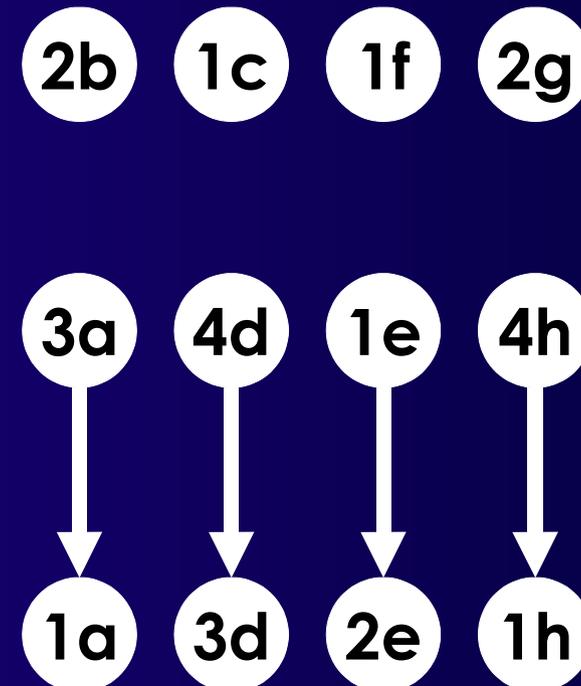


Vertical
Constraint
Graph (VCG)

Vertikale Einschränkungen 2

■ VCG: Einzeln betrachtet

- Wenig aussagekräftig
 - ◆ Ein verbundenes Knotenpaar pro gegenüberliegende unverbundene Terminals



Vertikale Einschränkungen 3

■ Aber: Im klassischen Modell

- Alle Terminals eines Netzes laufen auf *einem* horizontalen Segment

→ Alle Terminalsegmente enden in *einer* Zeile

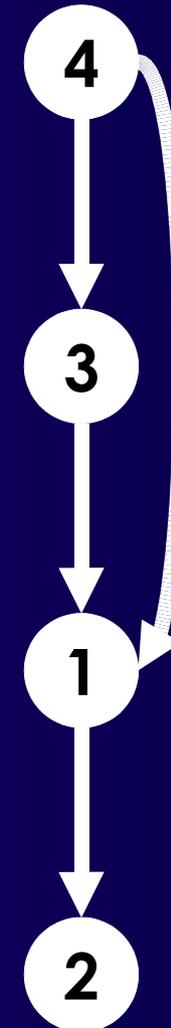
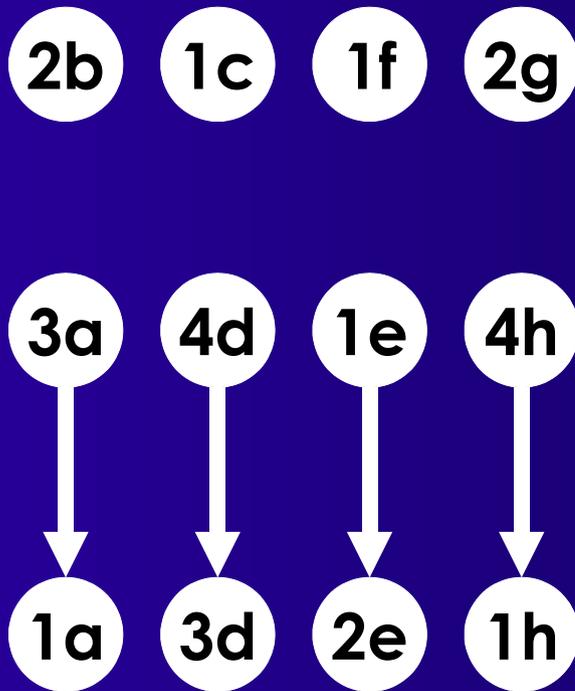
- Zusätzliche Abhängigkeit

■ Darstellung im VCG

- Verschmelzen der Terminal-Knoten
- ... zu einem Knoten *pro* Netz

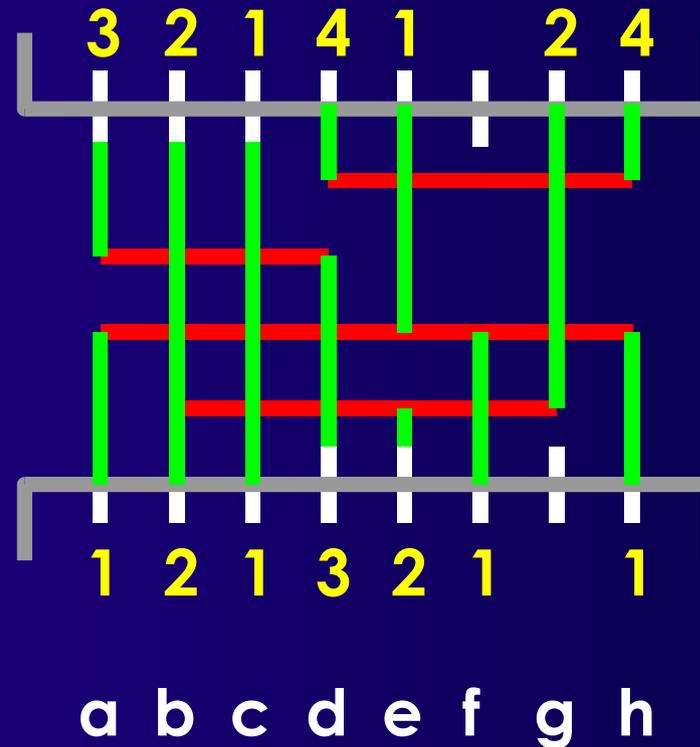
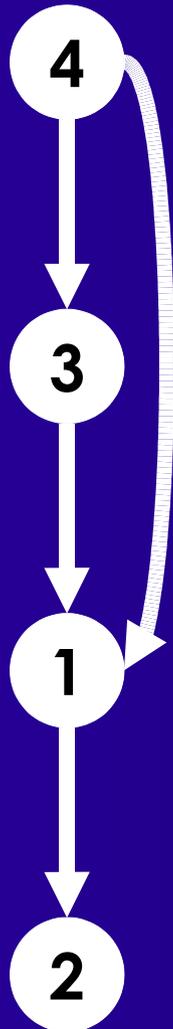
Vertikale Einschränkungen 4

■ Fortführung des letzten Beispiels



Vertikale Einschränkungen 5

■ Eindeutige Lösung des Beispiels



Vertikale Einschränkungen 6

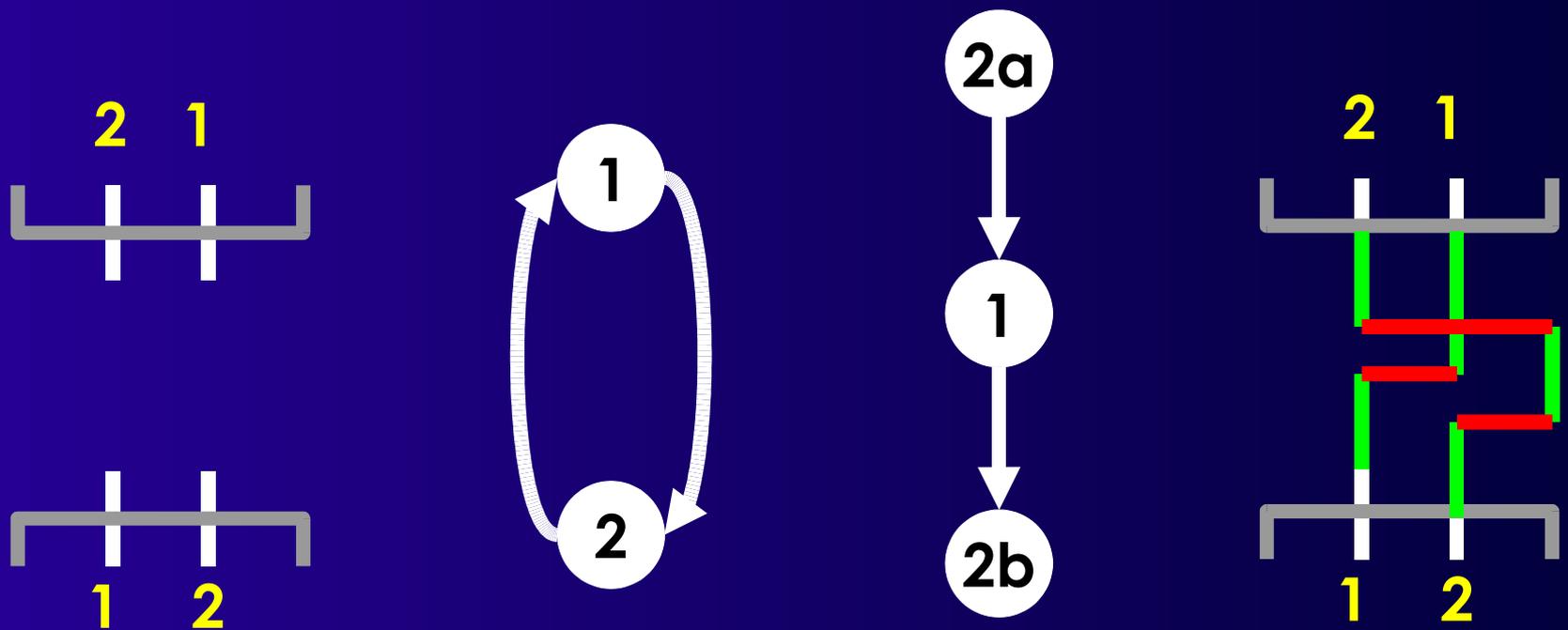
- **Extremformen von VCGs**
 - Vollständig verschmolzen
 - Vollständig getrennt

- **Zwischenstufen möglich**
 - Ein Knoten pro horizontalem Segment
 - ◆ Auch in nicht-klassischen Modellen verwendbar
 - ◆ Mehr als ein H-Segment pro Netz

Vertikale Einschränkungen 7

■ Zyklen im VCG

- Mit einzeltem H-Segment pro Netz nicht mehr lösbar



■ Lösung: Knoten auftrennen

- Auch für Doglegs!

Vertikale Einschränkungen 8

- **Falls nur vertikale Einschränkungen:**
 - Problem leicht lösbar
 - Berechnung des längsten Pfades
 - ◆ Analog zur Kompaktierung
- **Aber**
 - Es gibt auch horizontale Einschränkungen

Horizontale Einschränkungen 1

■ Im klassischen Modell

- Keine Überlappung zwischen H-Segmenten verschiedener Netze in gleicher Zeile
- Sonst Kurzschluß

→ Horizontale Einschränkung

■ Falls keine vertikalen Einschränkungen

- Keine gegenüberliegenden Terminals
- Lösung durch Left-Edge Algorithmus (1971)

Left-Edge Algorithmus 1

■ Modelliere Netz i als Intervall

$$\left[x_{i_{\min}}, x_{i_{\max}} \right]$$

- Begrenzt durch Position der linken/rechten Terminals

■ Ausreichend Informationen, da

- Kein vertikalen Einschränkungen
 - ◆ Zeile des H-Segments kann *überall* erreicht werden

■ Optimale Lösung

- Nicht-überlappende Intervalle in eine Zeile
- Minimale Anzahl von Zeilen

Left-Edge Algorithmus 2

■ Lokale Dichte in Spalte x : $d(x)$

- Anzahl von Intervallen, die Spalte x enthalten

■ Maximale lokale Dichte

$$d_{\max} = \max_x d(x)$$

■ Untere Schranke für Anzahl Zeilen

- Alle überlappenden Intervalle müssen in eigene Zeilen gelegt werden

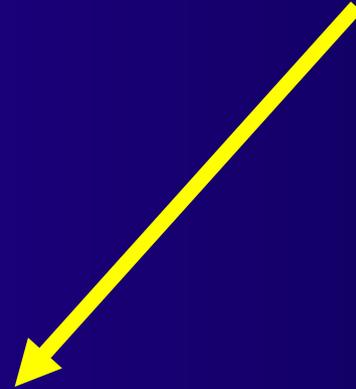
■ Left-Edge Algorithmus findet immer Optimum

Left-Edge Algorithmus 3

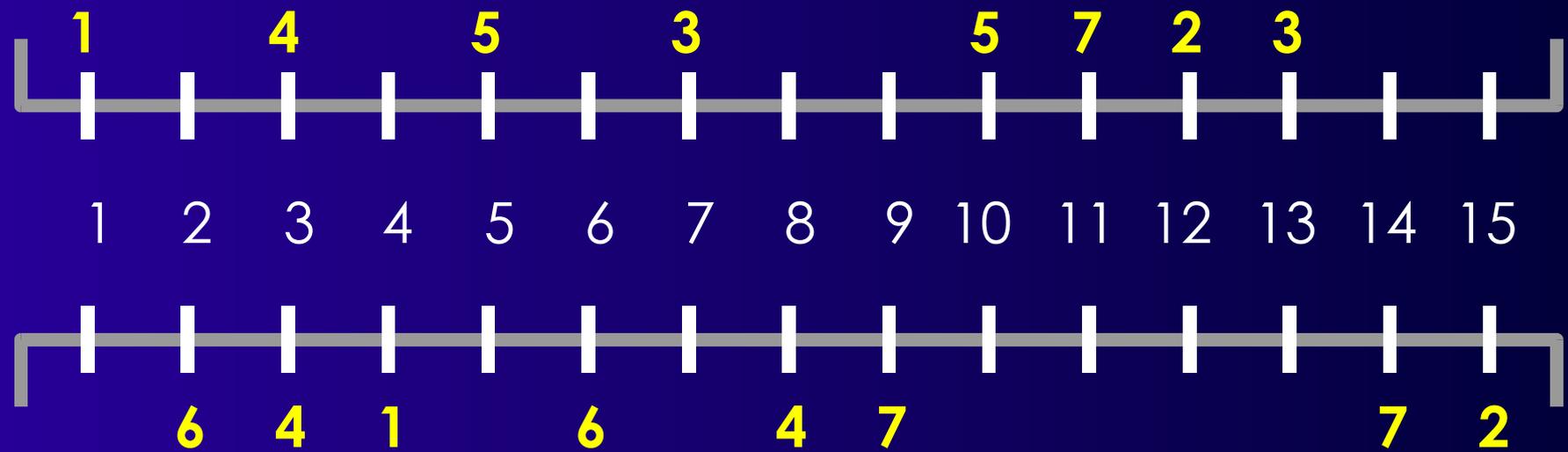
```
left_edge(list<interval> i_list) {
  /* intervall in i_list nach aufsteigender linker Koordinate sortiert */
  set<set<interval>> solution;
  set<interval> row;
  interval f;

  solution := ∅;
  while (!i_list.empty()) {
    f := i_list.head();
    i_list := i_list.tail();
    row := ∅;
    do {
      row := row ∪ {f};
      f := "erstes Element in i_list ohne Überlappung mit f";
      i_list.remove(f);
    } while (f != nil);
    solution := solution ∪ {row};
  }
  return (solution);
}
```

- Greedy Algorithmus
- Findet aber Optimum



Left-Edge Algorithmus 4



$i_1=[1,4]$ $i_2=[12,15]$ $i_3=[7,13]$ $i_4=[3,8]$
 $i_5=[5,10]$ $i_6=[2,6]$ $i_7=[9,14]$

$d_{max} = 3$

$i_list = [1,4], [2,6], [3,8], [5,10], [7,13], [9,14], [12, 15]$

Left-Edge Algorithmus 5

solution = \emptyset

i_list = [1,4], [2,6], [3,8], [5,10], [7,13], [9,14], [12, 15]

f = [1,4]

i_list = [2,6], [3,8], [5,10], [7,13], [9,14], [12, 15]

row = \emptyset

row = $\emptyset \cup \{f\} = \{[1,4]\}$

f = "ohne Überlappung mit f" = [5,10]

i_list = [2,6], [3,8], [7,13], [9,14], [12, 15]

row = $\{[1,4]\} \cup \{f\} = \{[1,4],[5,10]\}$

f = "ohne Überlappung mit f" = [12,15]

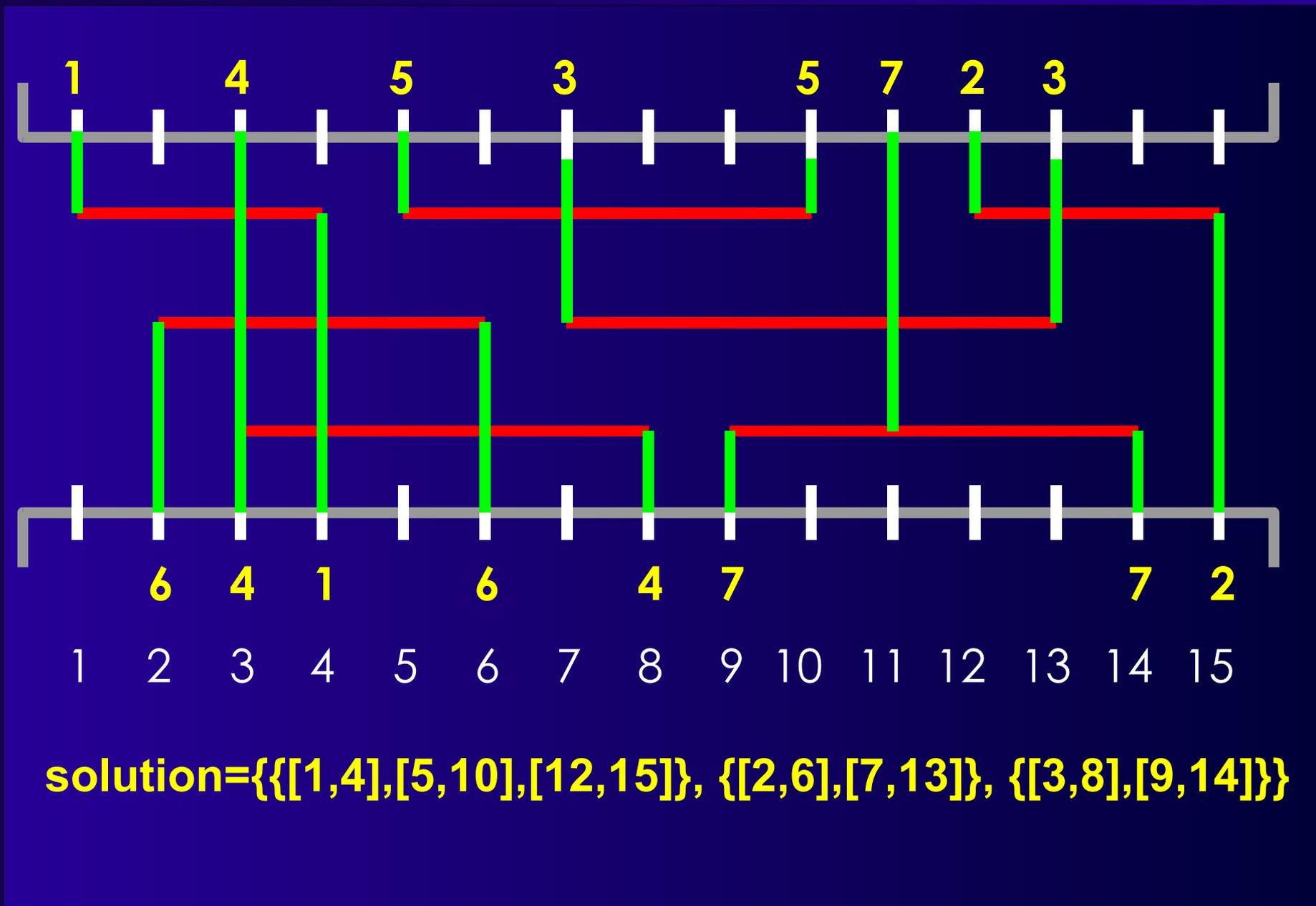
i_list = [2,6], [3,8], [7,13], [9,14]

row = $\{[1,4],[5,10]\} \cup \{f\} = \{[1,4],[5,10],[12,15]\}$

f = "ohne Überlappung mit f" = nil

solution = $\emptyset \cup \{row\} = \{\{[1,4],[5,10],[12,15]\}\}$

Left-Edge Algorithmus 6



Left-Edge Algorithmus 7

■ Komplexität

- n Intervalle
- d Zeilen
- Sortieren nach linker Koordinate: $O(n \log n)$
- Äußere Schleife: d Durchläufe
- Innere Schleife: max. n Intervalle betrachtet
- $O(n \log n + d n)$
 - ◆ Kann noch verbessert werden: $\Theta(n)$

Left-Edge Algorithmus 8

- Als graphentheoretisches Problem
- Intervallgraph $G(V, E)$
 - Knoten pro Intervall
 - Kante zwischen überlappenden Intervallen
- **Untermenge aller Graphen**
- Nicht benachbarte Knoten: Intervalle in einer Zeile möglich

Left-Edge Algorithmus 9

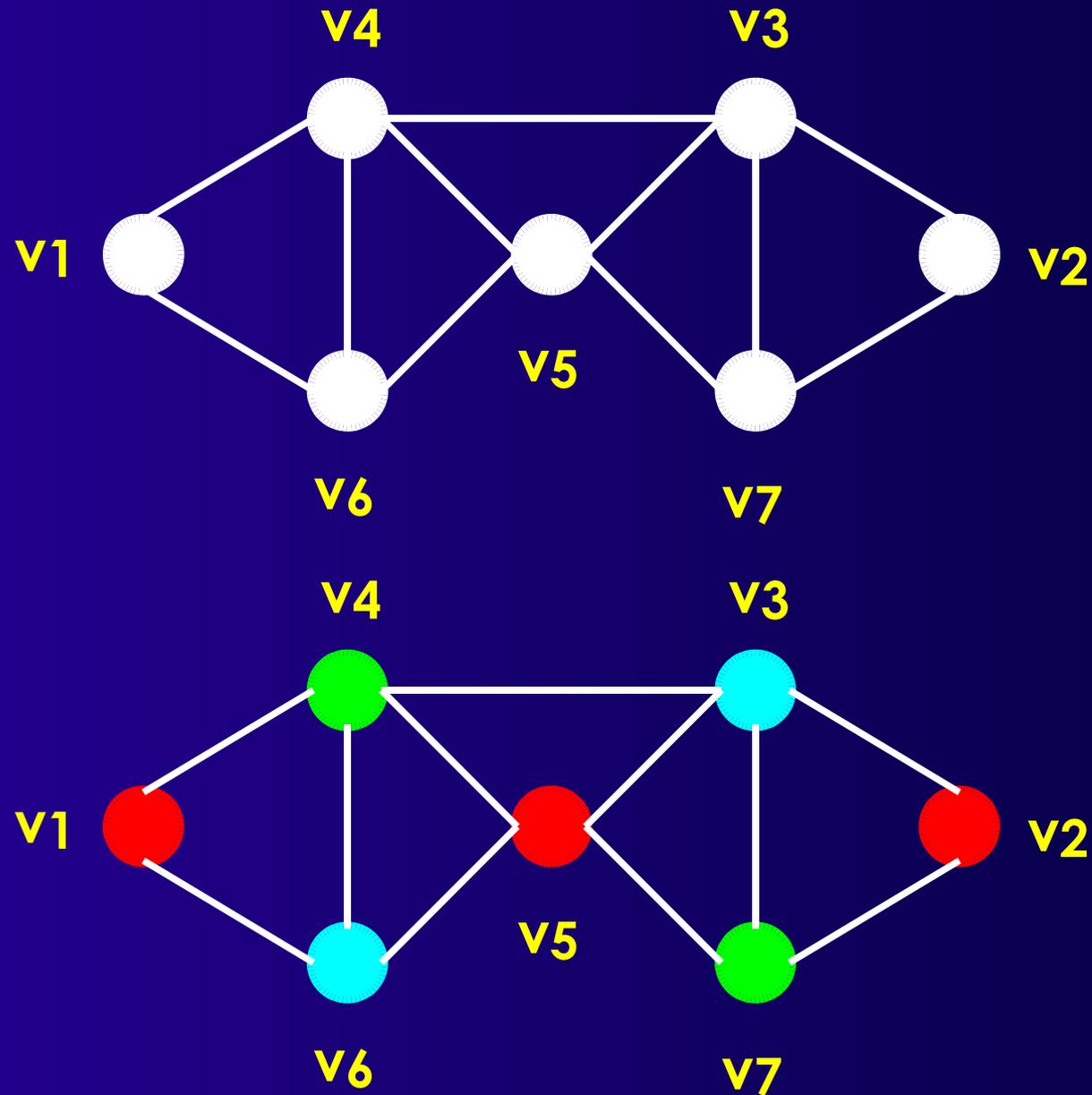
■ Analog zu

- Bestimme minimale Anzahl von Farben, so daß benachbarte Knoten unterschiedliche Farben haben

■ Farben \Leftrightarrow Zeilen

- Klassisches Problem der Graphentheorie
 - ◆ Normalerweise NP-vollständig
 - ◆ Für Intervallgraphen aber in P

Left-Edge Algorithmus 10



Weiterer Ablauf

- **Di: Realer FPGA-Router „Pathfinder“**
 - Wichtig für nächste Praktikumsphase

■ Flächenverdrahtung

- Lee's Algorithmus

■ Kanalverdrahtung

- Klassisches Modell
 - ◆ Ausnahmen
- Einschränkungen
 - ◆ Vertikale
 - ◆ Horizontale
 - ◆ Left-Edge Algorithmus
 - ◆ Graphentheoretische Sicht