

Algorithmen im Chip-Entwurf 2

Kompaktierung, Schaltungsdarstellungen und Timing-Analyse

Andreas Koch
FG Eingebettete Systeme
und ihre Anwendungen
TU Darmstadt

Organisatorisches

■ Vorgehensweise

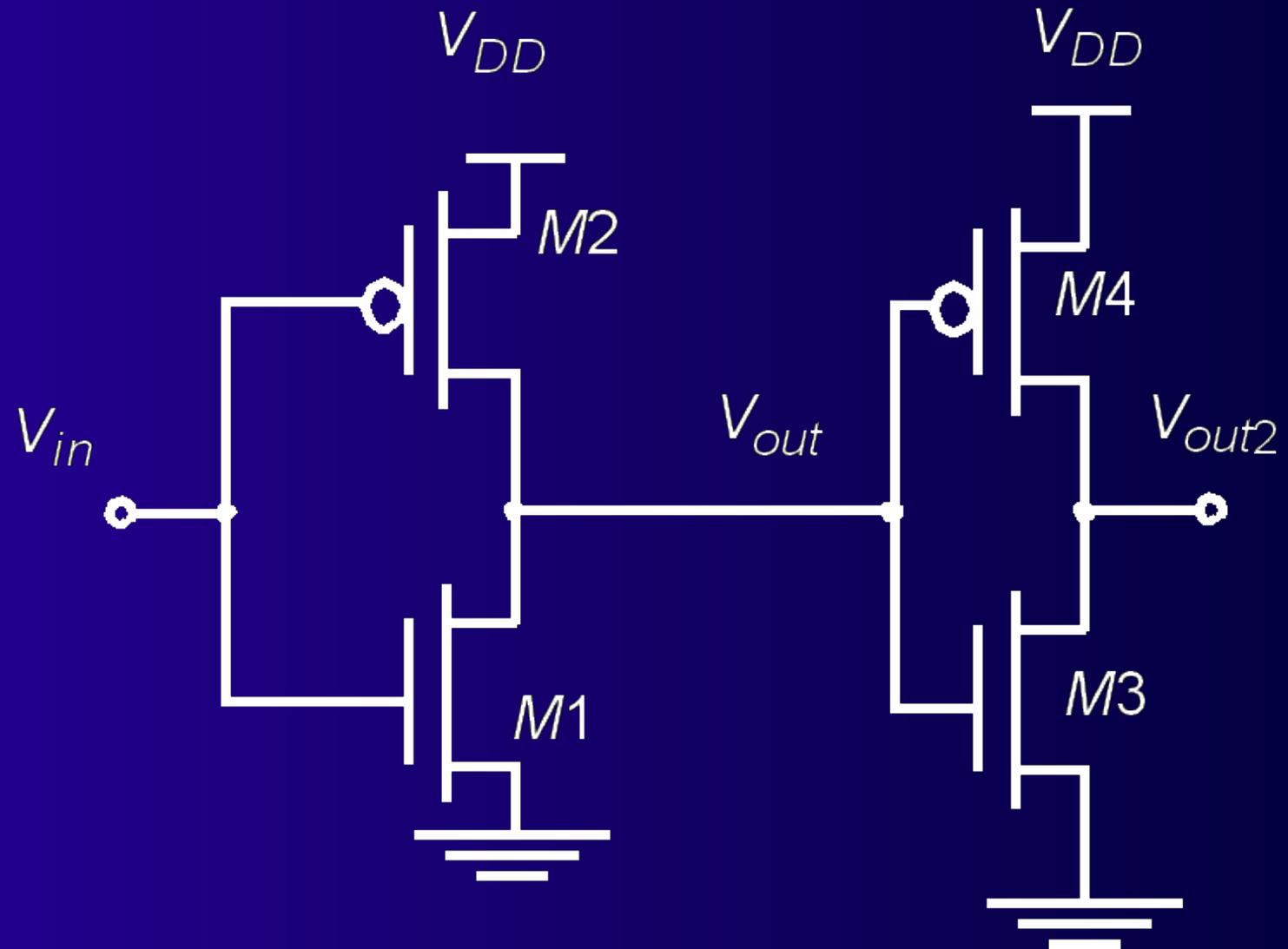
- Anmeldebögen

■ Wichtige Spalten ganz rechts: Ankreuzen

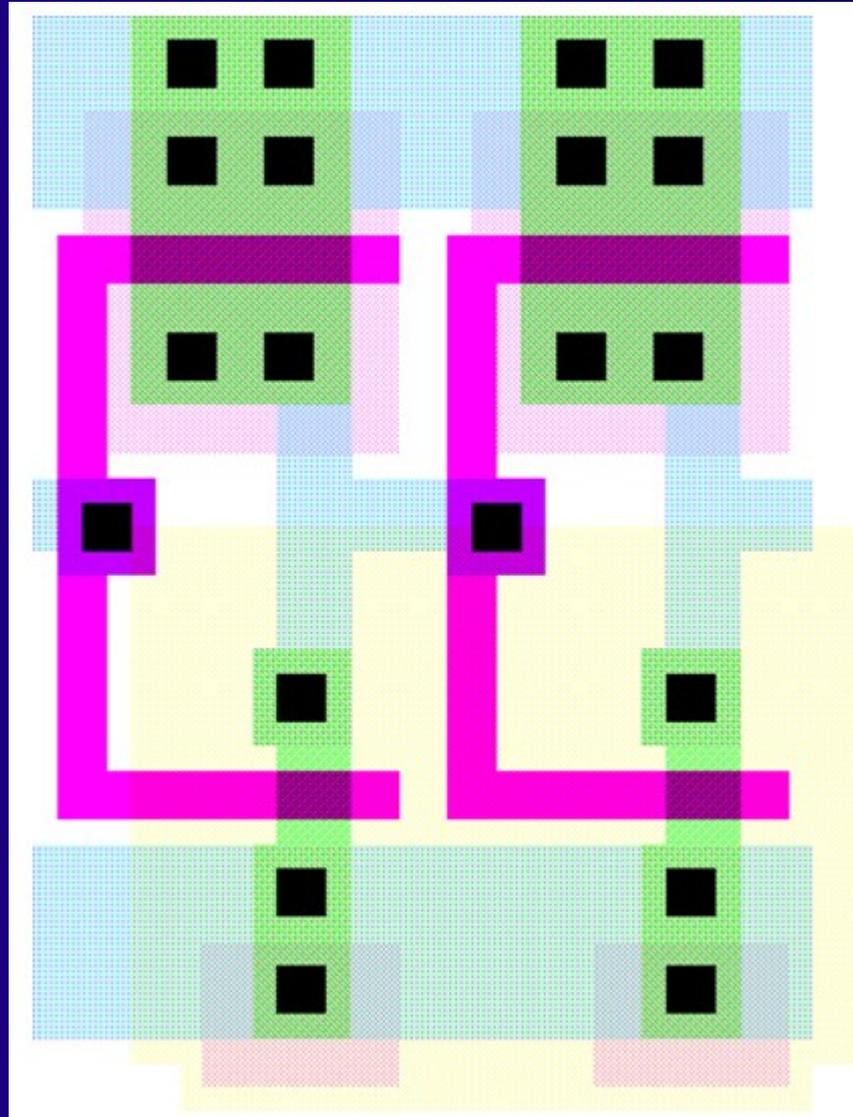
- IV 4 SWS
 - ◆ Sie wollen das ganze Programmierprojekt
- VL 2 SWS
 - ◆ Sie wollen ohnehin nur die VL hören

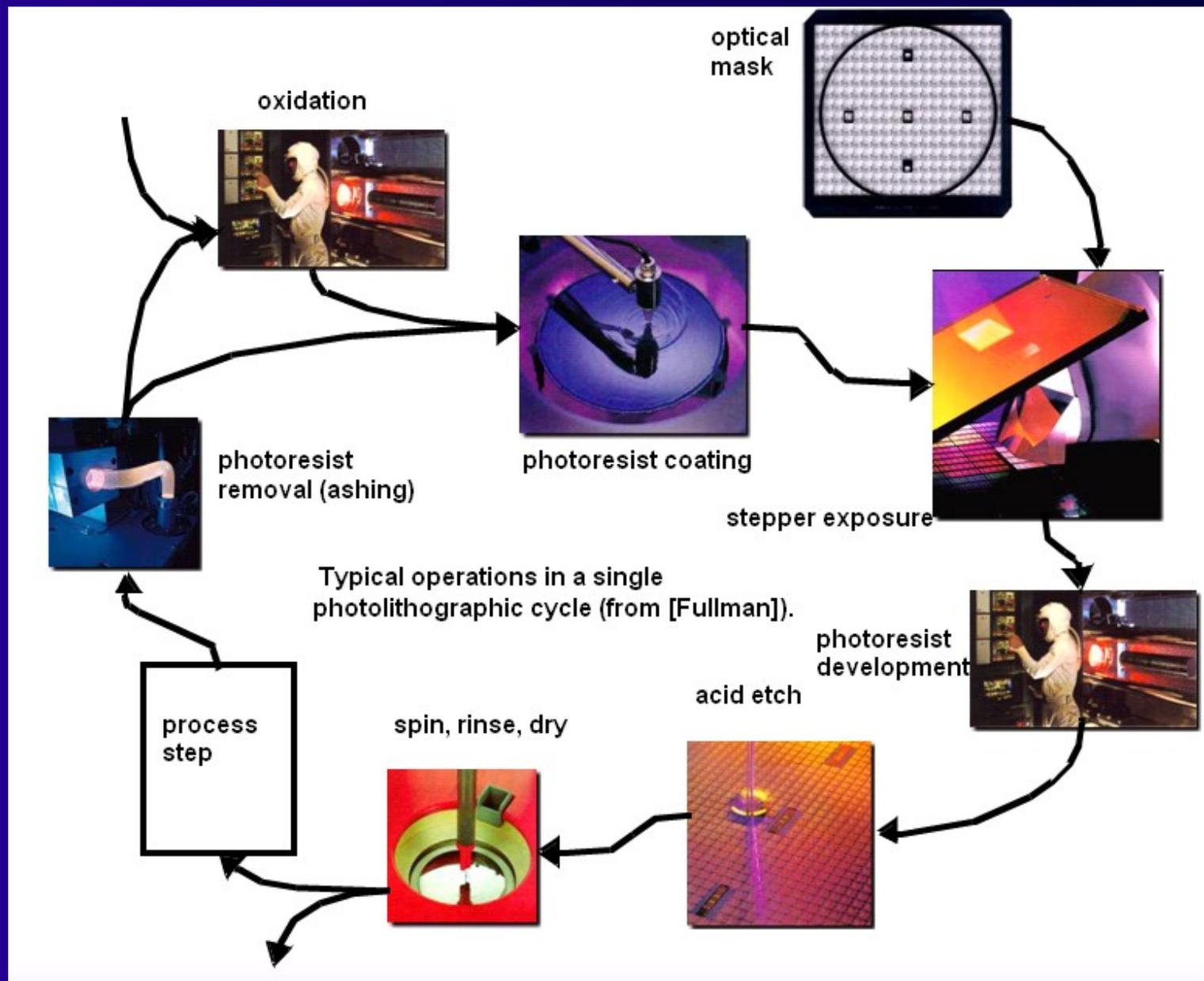
- **Grundlagen von VLSI-Chips**
- **Kompaktierung**
 - Längste Pfade
- **Datenstrukturen für Schaltungen**
- **Timing-Analyse**
- **Zusammenfassung**

Transistorschaltungen



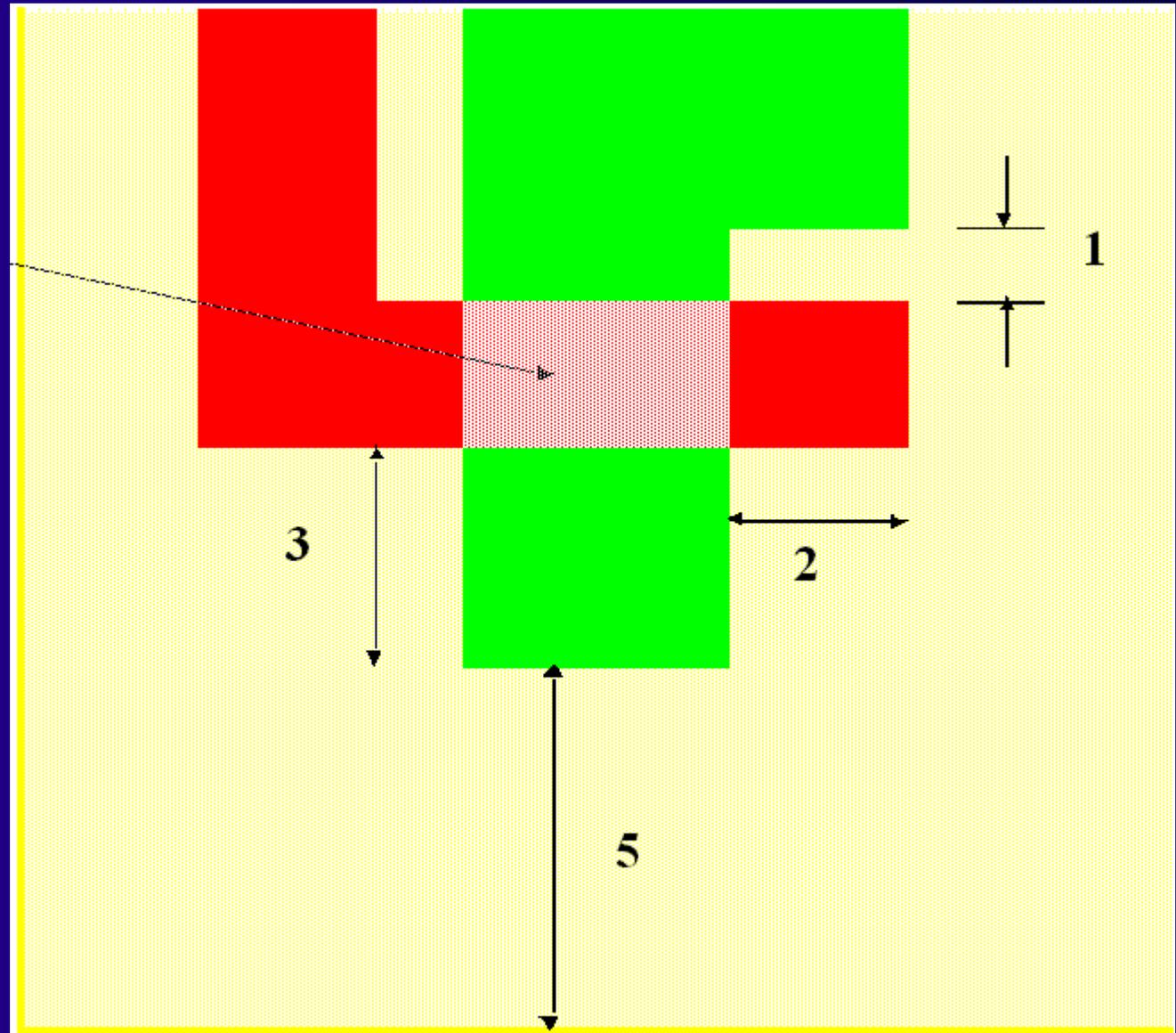
Layout-Sicht



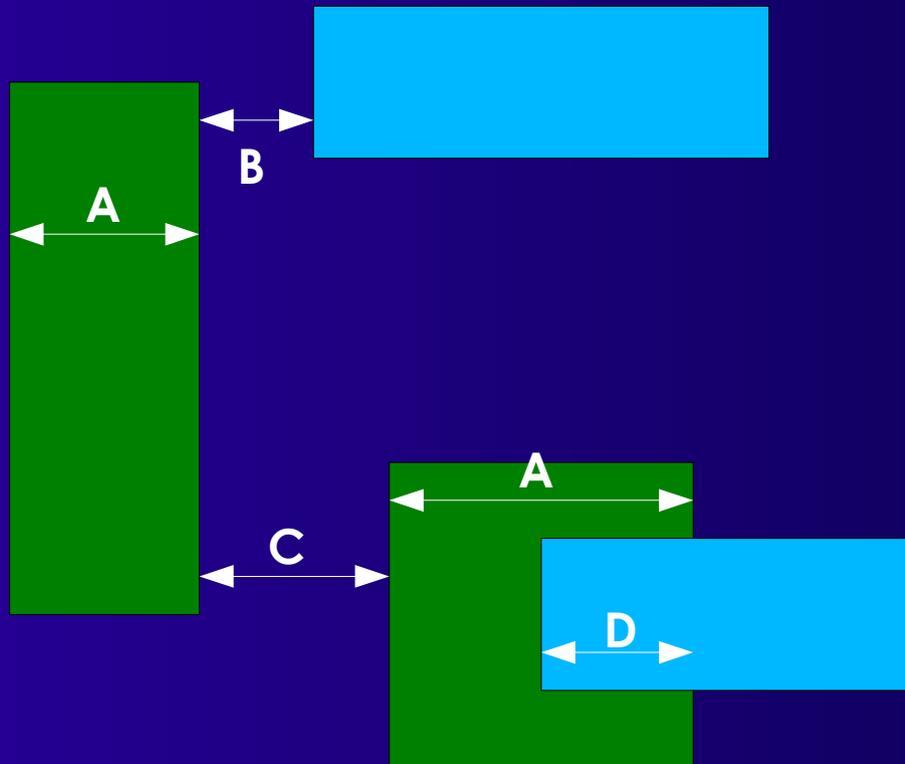


Entwurfsregeln 1

Transistor



Entwurfsregeln 2



A – minimale Breite
B – minimaler Abstand (L1-L2)
C – minimaler Abstand (L1-L1)
D – minimale Überlappung

■ Bei ASIC-Layouts

- Grundlage für erfolgreiche Fertigbarkeit
- Von „Technologen“ erarbeitet

Symbolisches Layout

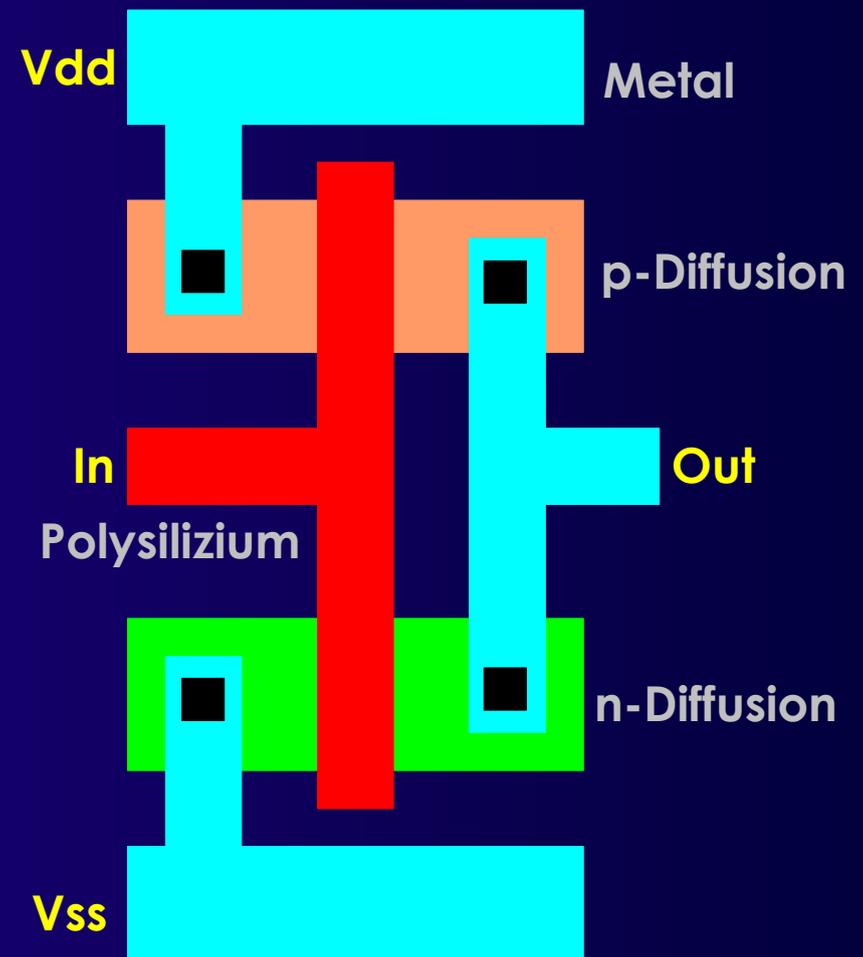
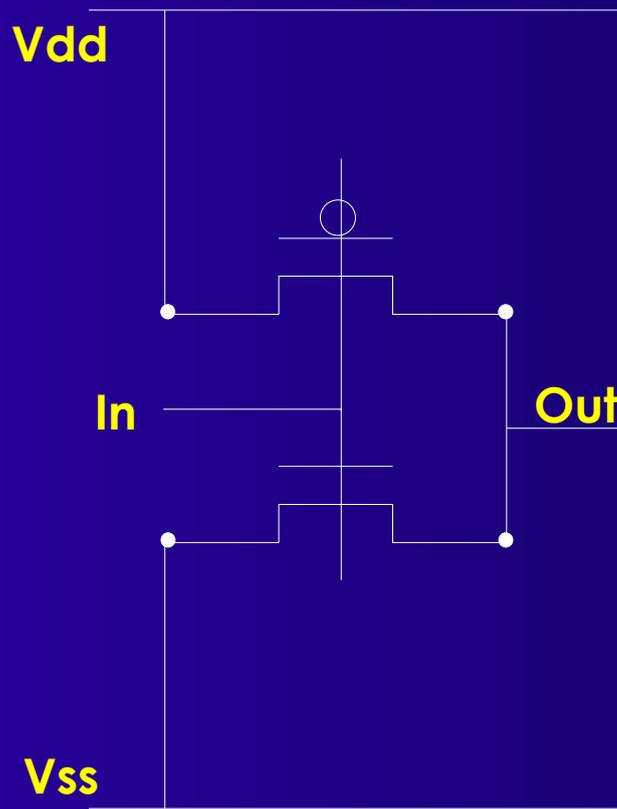
■ Kein vollständiges Layout

- Keine absoluten geometrischen Angaben

■ Stattdessen

- *Symbole* für Elemente
 - ◆ Transistoren, Kontakte
- Für Elemente noch variabel
 - ◆ Länge, Breite, Layer
- Einige Angaben fehlen vollständig
 - ◆ n- und p-Wells (irrelevant für Funktionalität)
 - ◆ Automatisch berechenbar

Symbolisches Layout



■ Komprimieren/Expandieren von Layouts

- Unter Beachtung der Design-Rules

■ Anwendungsgebiete

- Layout-Compilierung
 - ◆ Von symbolischen in geometrische Layouts
- Flächenminimierung
 - ◆ Von bestehenden Layouts
- Korrektur
 - ◆ Entfernung von Entwurfsregelverletzungen
- Skalierung
 - ◆ Portierung eines Layouts auf andere Technologie

Vorgehensweise

■ Eindimensional (1D)

- Nur eine Richtung bearbeitet
 - ◆ Operationen: Bewegen, Stauchen
- Oft abwechselnd in X, Y Richtungen

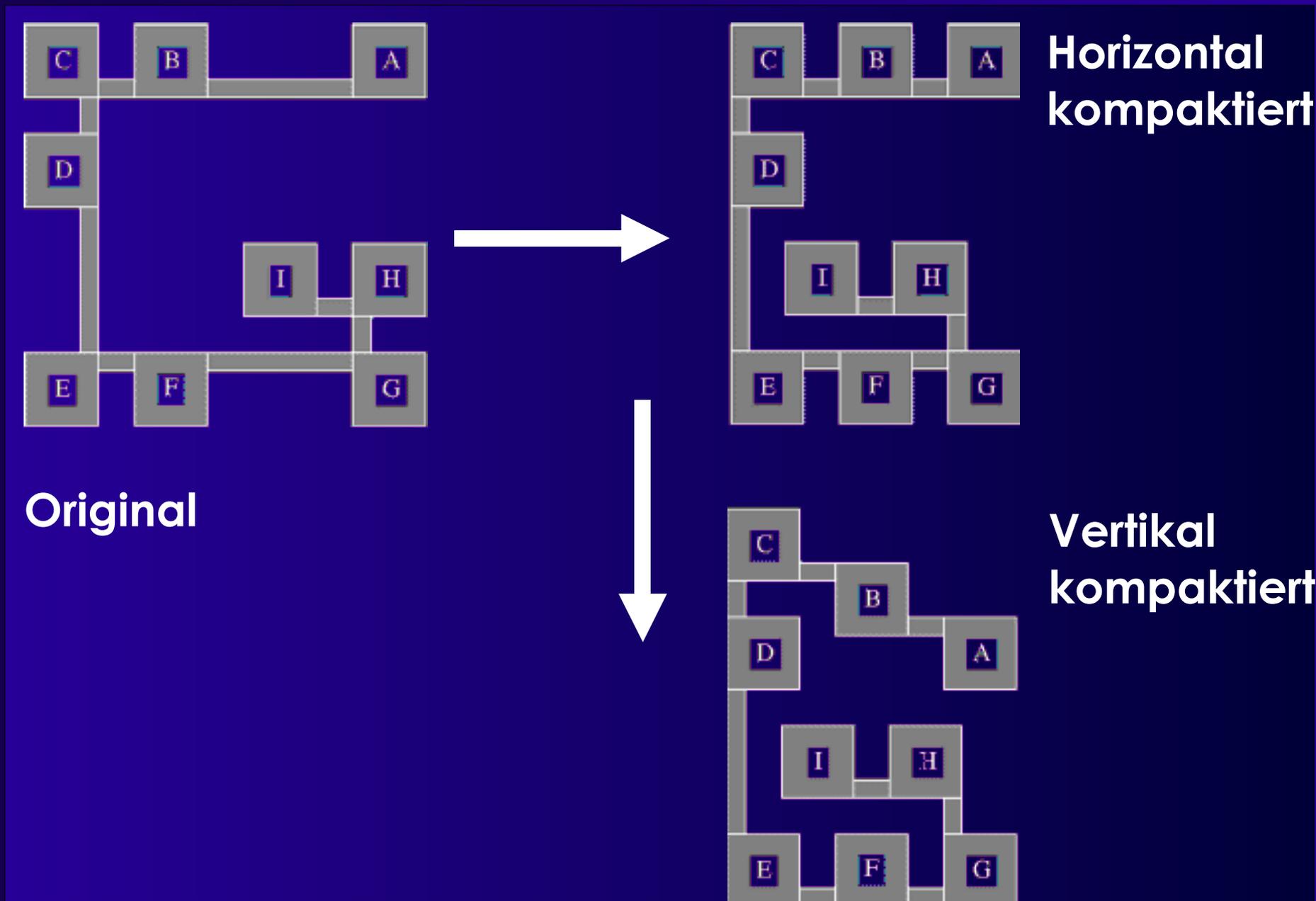
■ Zweidimensional (2D)

- Beide Richtungen simultan bearbeiten

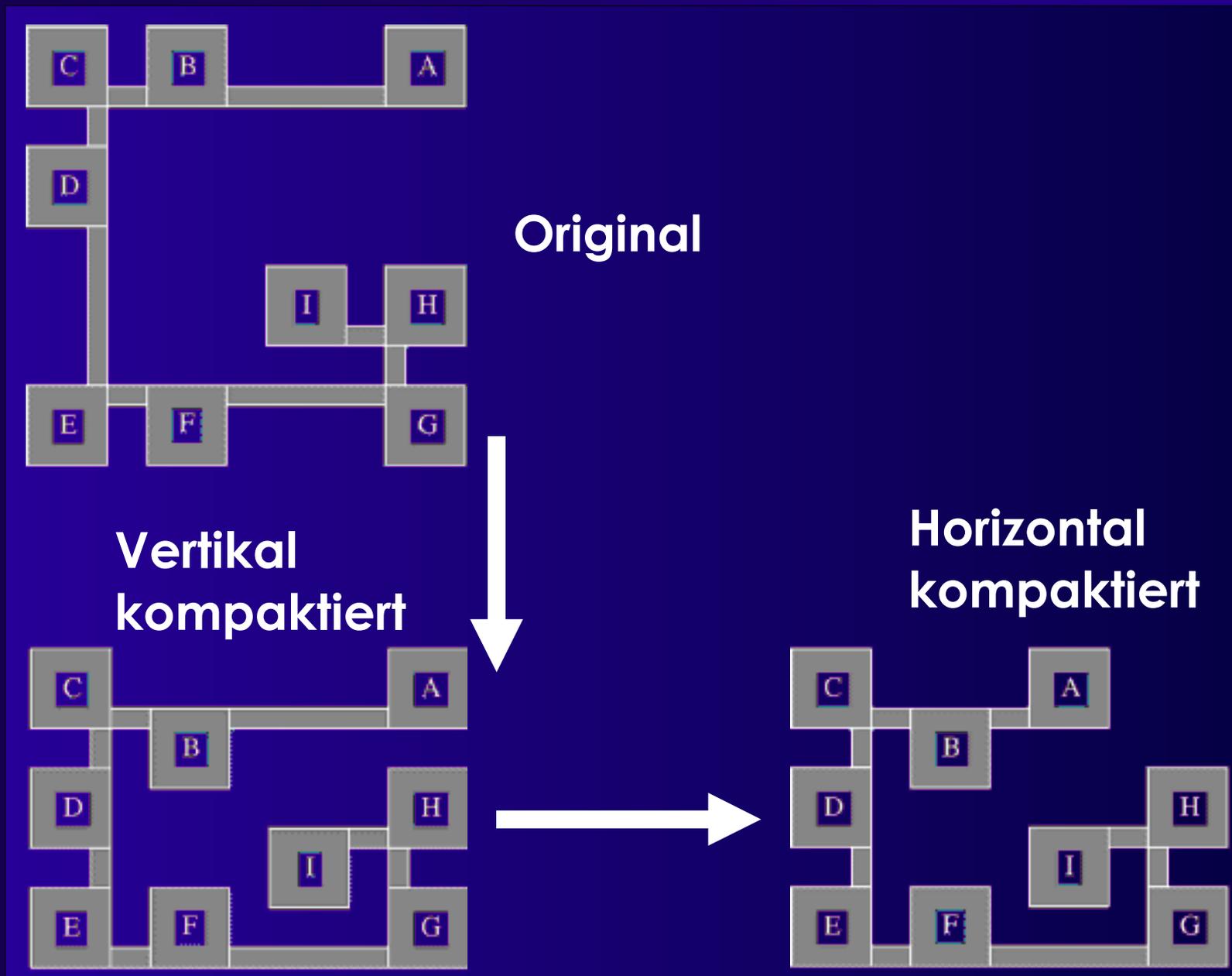
■ Problem

- 1D ist effizient machbar, aber suboptimal
- 2D liefert optimale Lösung, ist aber NP-hart

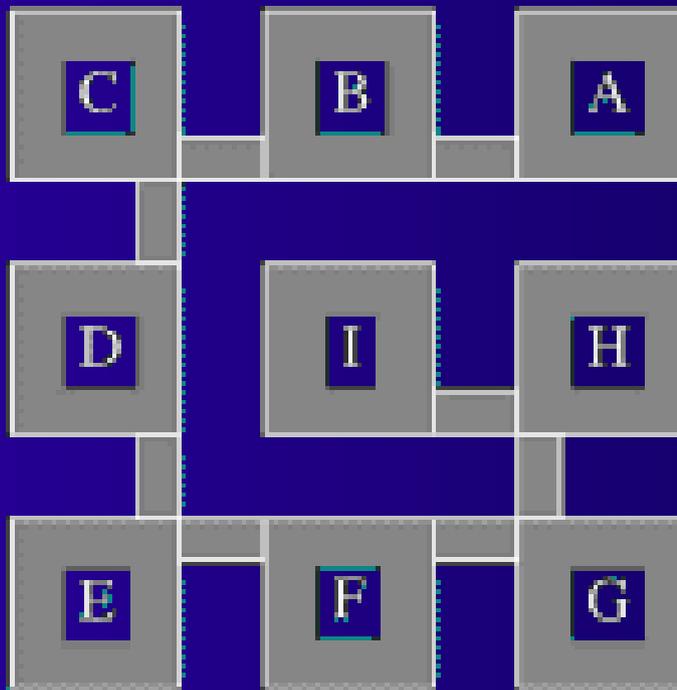
Kompaktierung 1



Kompaktierung 2

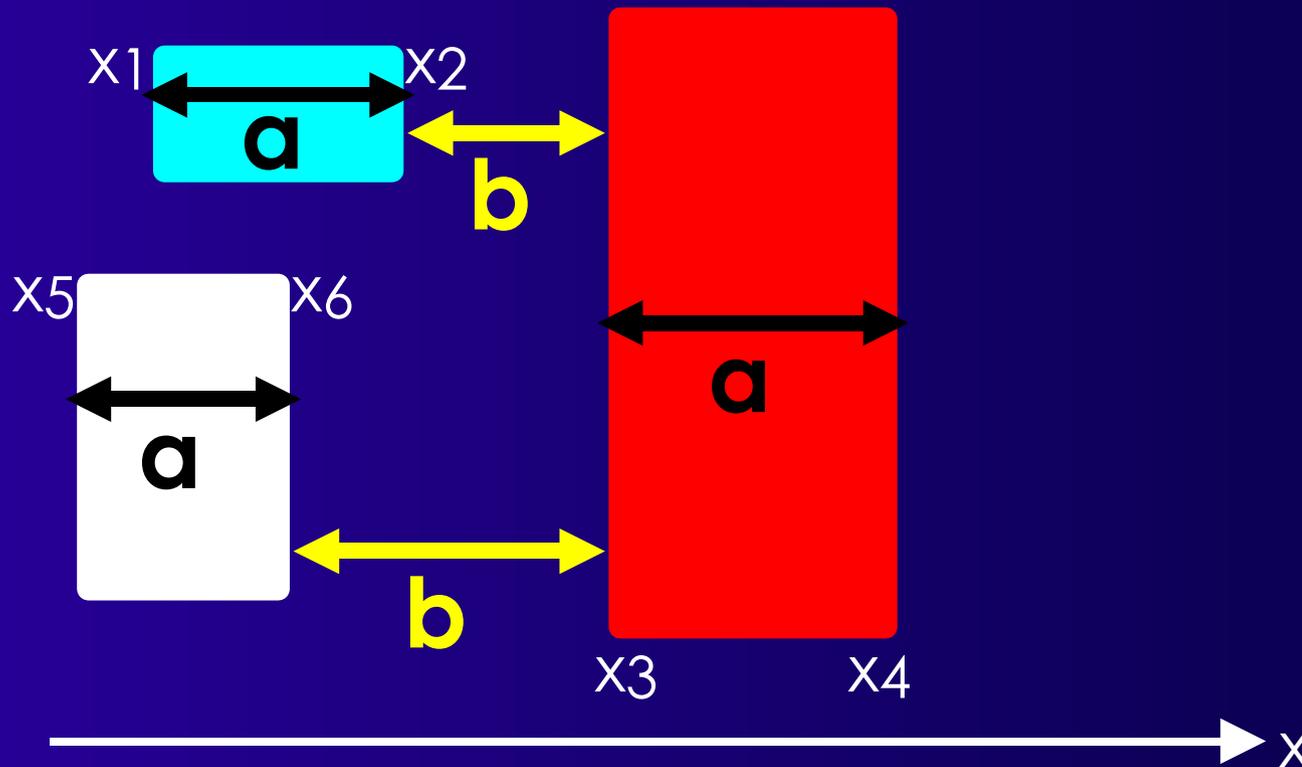


Kompaktierung 3



- **2D Kompaktierung**
 - Findet optimale Lsg.
 - NP-vollständig
- **Vorgehensweise**
 - Mehrfache 1D-K
 - Wechselnd in H, V
 - Aber: nicht optimal

Modellierung 1



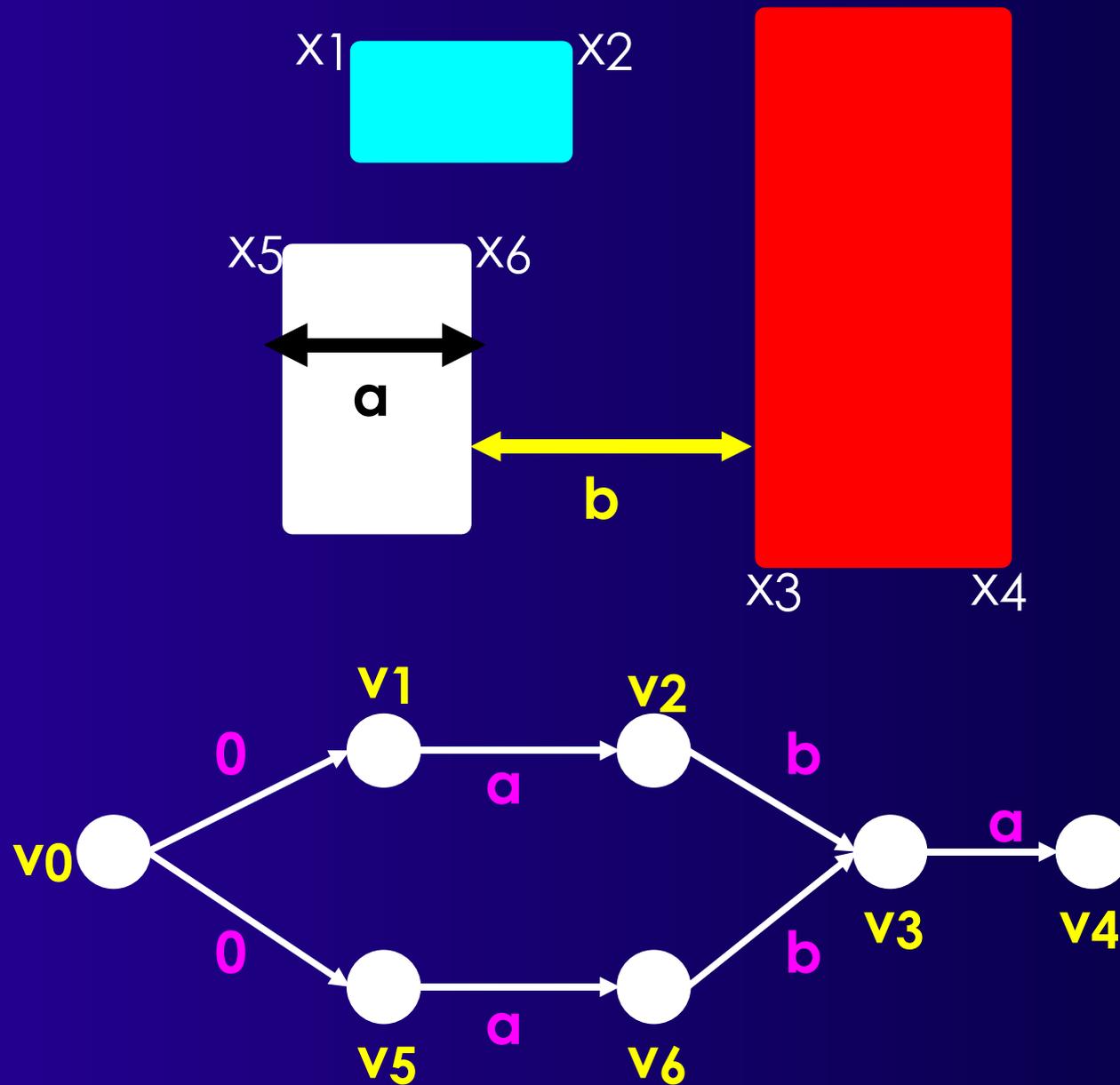
$$x_2 - x_1 \geq a$$

$$x_3 - x_2 \geq b$$

$$x_3 - x_6 \geq b$$

$$x_j - x_i \geq d_{ij}$$

Modellierung 2

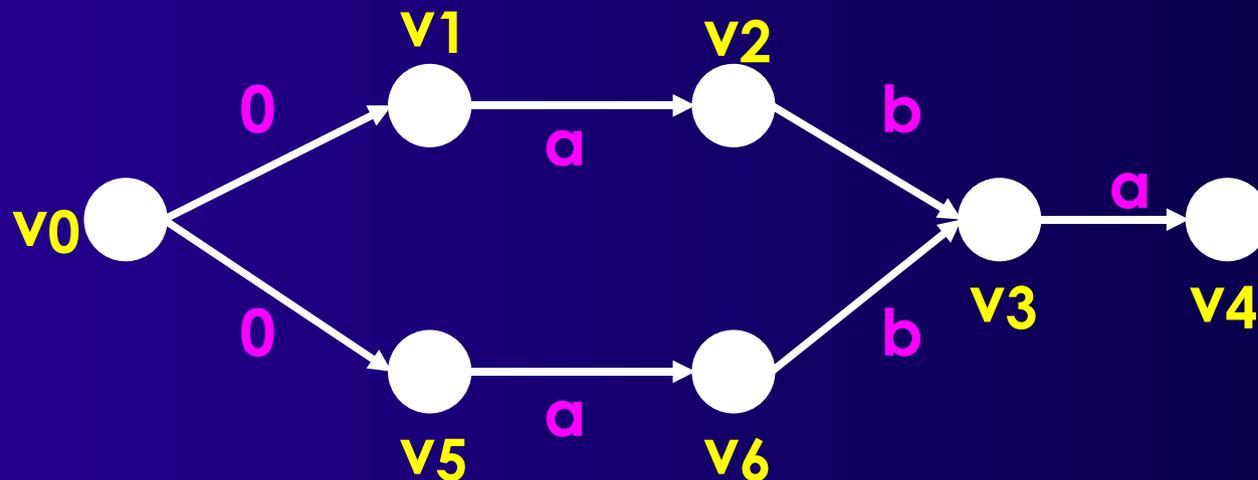


■ Einschränkungsgraph $G(V, E)$

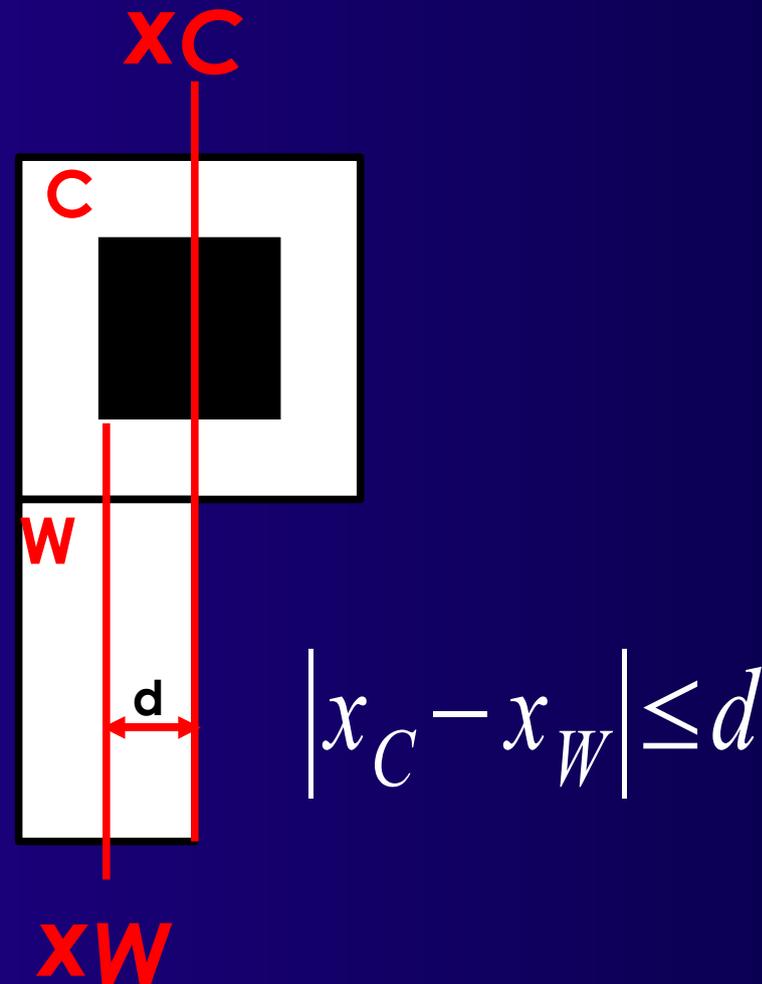
- Gerichtet von (v_i, v_j)
- Zyklenfrei

■ Längster Pfad von v_0 zu v_i

→ Minimale Koordinate von x_i



■ Bedingungen an *maximalen* Abstand



■ $|x_C - x_W| \leq d$

● $x_j - x_i \leq c_{ij}, c_{ij} \geq 0$

● $x_i - x_j \geq -c_{ij}$

■ **Passende Form für Einschränkungsgraph**

- Jetzt aber Richtung (v_j, v_i): Zyklen möglich!

■ **Aufgabe:**

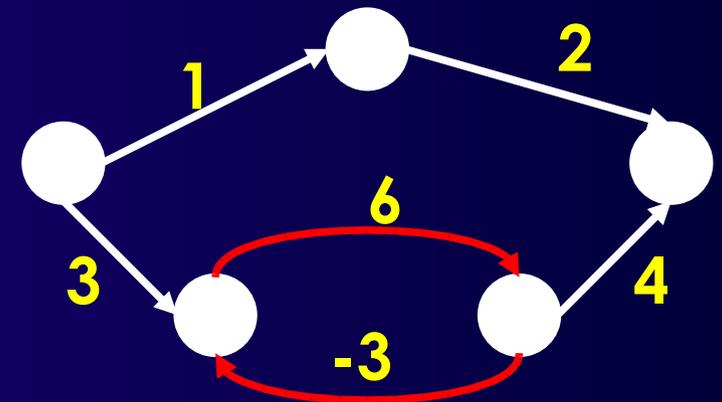
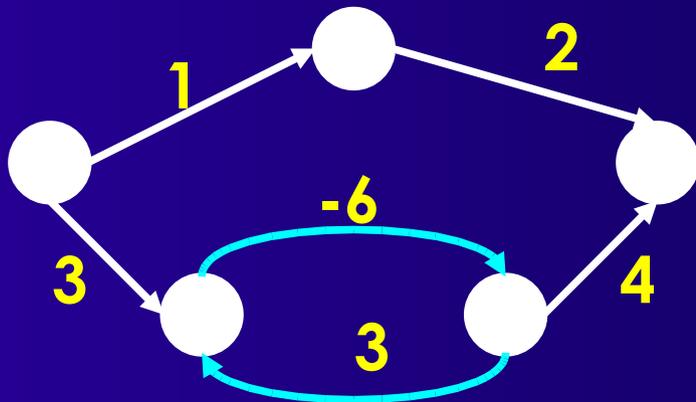
- Berechnung des längsten Pfades in Graphen mit Zyklen

■ Zyklensfreie Graphen

- OK, ähnlich BFS

■ Graphen mit Zyklen

- Ohne positiven Zyklus: **OK**
- Mit positivem Zyklus: **Undefiniert**
 - ◆ Kompaktierung: Überbeschränktes Layout



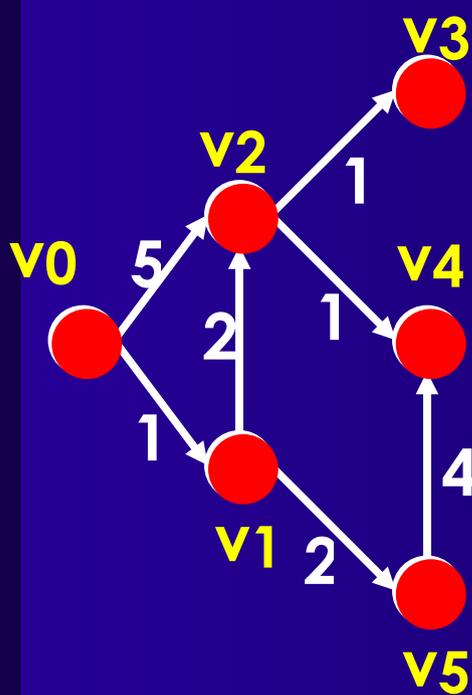
Zyklenfreie Graphen 1

```
main() {  
    for (i:=0; i ≤ n; ++i)  
        xi := 0;  
    longest_path(G);  
}
```

- **Directed Acyclic Graph (DAG)**
- **Längster, gerichteter, einfacher Pfad (*trail*)**

```
longest_path(G) {  
    for (i:=1; i ≤ n; ++i)  
        pi := vi.indegree();  
    set Q := {v0};  
    while (Q ≠ ∅) {  
        vi := Q.pickany();  
        Q := Q \ {vi};  
        foreach (vi, vj) ∈ E {  
            xj := max(xj, xi + dij);  
            --pj;  
            if (pj ≤ 0)  
                Q := Q ∪ {vj};  
        }  
    }  
}
```

Zyklenfreie Graphen 2



Q	p1	p2	p3	p4	p5	x1	x2	x3	x4	x5
nil	1	2	1	2	1	0	0	0	0	0
{v0}	0	1	1	2	1	1	5	0	0	0
{v1}	0	0	1	2	0	1	5	0	0	3
{v2,v5}	0	0	0	1	0	1	5	6	6	3
{v3,v5}	0	0	0	1	0	1	5	6	6	3
{v5}	0	0	0	0	0	1	5	6	7	3
{v4}	0	0	0	0	0	1	5	6	7	3

Graphen mit Zyklen

- Nur mit *negativen* Zyklen
- Erkenne positive Zyklen
 - Überbeschränkte Layouts
- Aber lokalisiere sie nicht

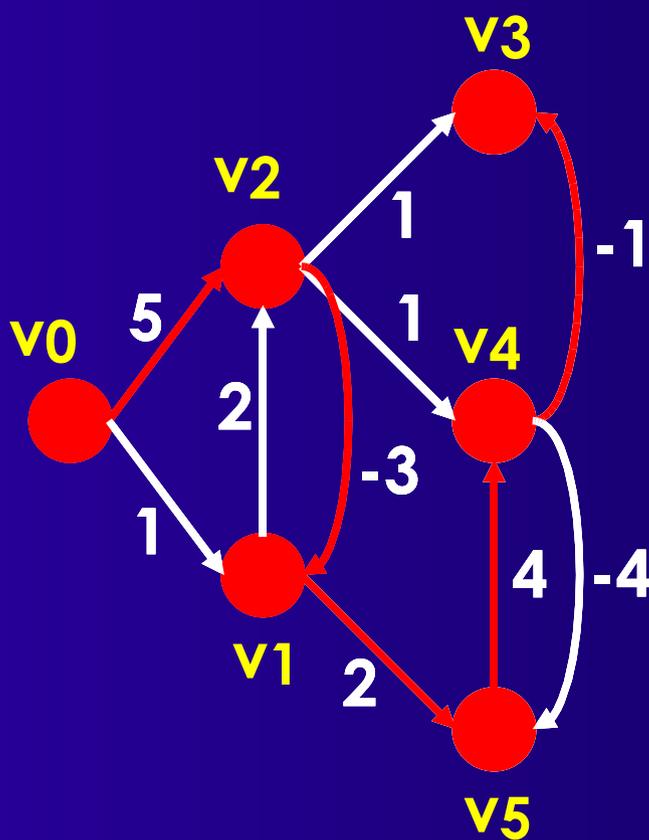
Längster Pfad mit Liao-Wong 1

```
count := 0;
for (i:=1; i ≤ n; ++i)
  xi := -∞;
x0 := 0;
do {
  is_modified := false;
  longest_path(Gf);
  foreach (vi, vj) ∈ Eb
    if (xj < xi + dij) {
      xj := xi + dij;
      is_modified := true;
    }
  ++count;
  if (count > |Eb| && is_modified)
    error("positive cycle!");
} while (is_modified);
```

■ Idee:

- Trennen zwischen
 - ◆ E_f: min. Distanz
 - ◆ E_b: max. Distanz
 - ◆ Erzeugen Zyklen!
- Berechne LP G_f(V, E_f)
- Korrigiere für E_b
 - ◆ Schließen Zyklen
- Stabilisiert sich in |E_b|
 - ◆ Jedes e_b nur 1x in Pfad
- sonst überbeschränkt

Längster Pfad mit Liao-Wong 2



Schritt	x1	x2	x3	x4	x5
Init.	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
Vor 1	1	5	6	7	3
Zurück 1	2	5	6	7	3
Vor 2	2	5	6	8	4
Zurück 2	2	5	7	8	4
Vor 3	2	5	7	8	4
Zurück 3	2	5	7	8	4

- Verbesserung: `longest_path(Gf)` bemerkt Änderung
- $O(|E_f| \times |E_b|)$

LP mit Bellman-Ford 1

- Kein Unterschied zwischen E_f und E_b
- Vergleichbar azyklischem LP
 - Aber mehrere Durchläufe durch Graph

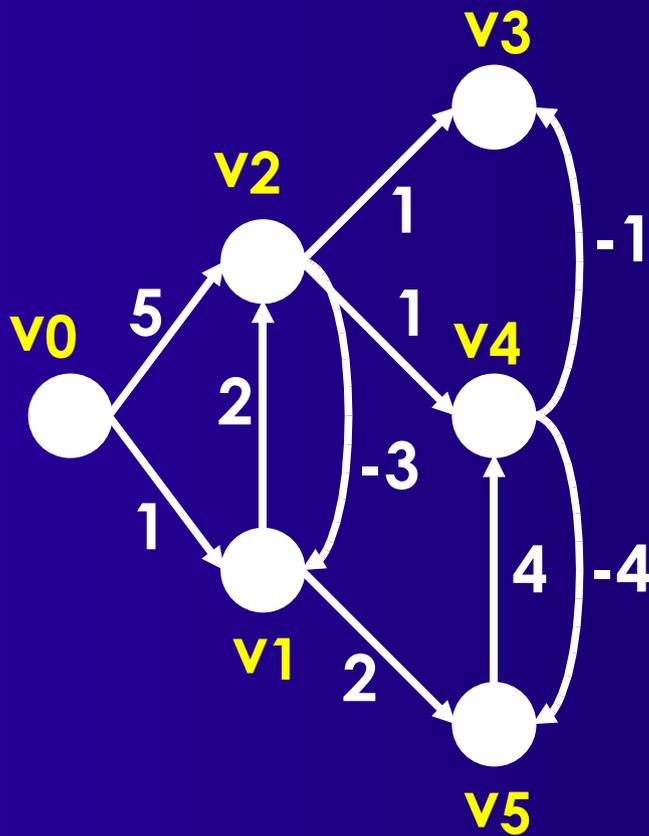
LP mit Bellman-Ford 2

```
for (i:=1; i ≤ n; ++i)
    xi := -∞;
x0 := 0;
count := 0;
S1 := {v0};
S2 := ∅;
while (count ≤ n && S1 ≠ ∅) {
    foreach vi ∈ S1
        foreach (vi, vj) ∈ E
            if (xj < xi + dij) {
                xj := xi + dij;
                S2 := S2 ∪ {vj};
            }
    S1 := S2;
    S2 := ∅;
    ++count;
}
if (count > n) error("positive cycle!");
```

■ Idee:

- Zwei Wellenfronten
 - ◆ S₁: aktuelle
 - ◆ S₂: nächste Iteration
- Zyklendetektion
 - ◆ k-te Iteration
 - LP durch k-1 Knoten
 - ◆ LP hat max. n Knoten
 - ➔ Falls mehr Iterationen
 - ◆ Zyklus!
- $O(n^3)$, avg. $O(n^{1.5})$

LP mit Bellman-Ford 3

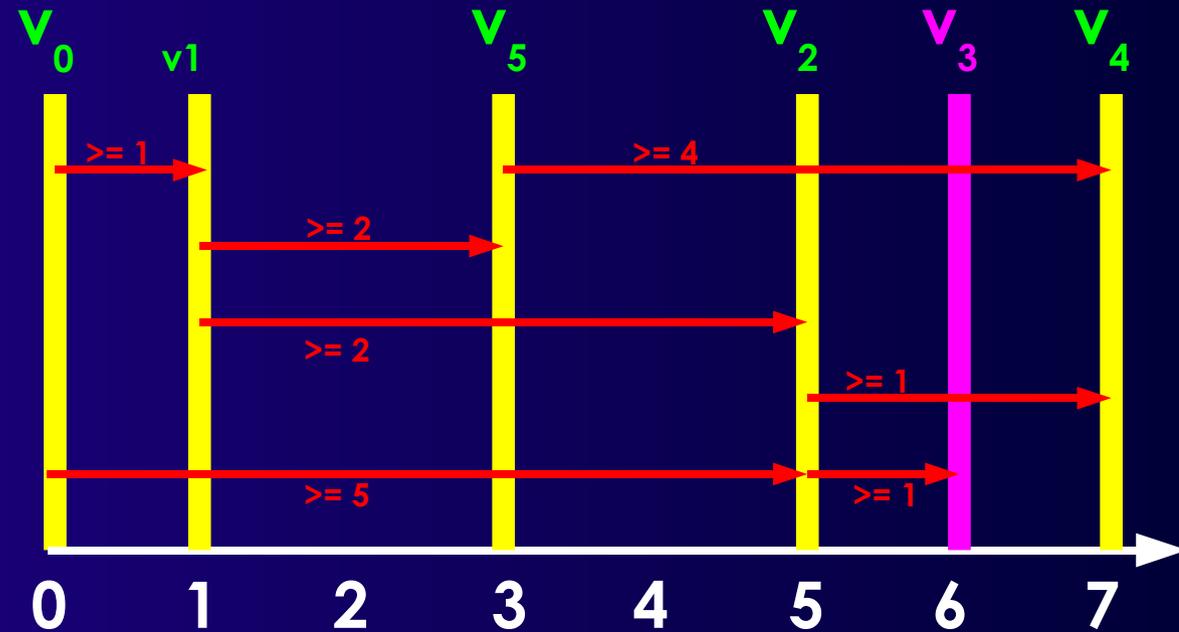
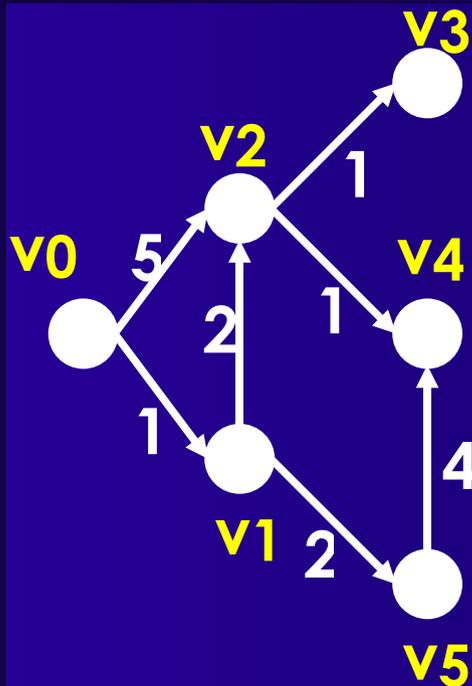


S_1	x_1	x_2	x_3	x_4	x_5
nil	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
{v0}	1	5	$-\infty$	$-\infty$	$-\infty$
{v1, v2}	2	5	6	6	3
{v1, v3, v4, v5}	2	5	6	7	4
{v4, v5}	2	5	6	8	4
{v4}	2	5	7	8	4
{v3}	2	5	7	8	4

Übersicht Pfad-Algorithmen

- LP wird SP bei Multiplikation der c_{ij} mit -1
- Gerichtete zyklensfreie Graphen (DAGs)
 - SP und LP lösbar in linearer Zeit
- Gerichtete Graphen mit Zyklen
 - Alle Gewichte positiv
 - ◆ SP in P, LP ist NP-vollständig
 - Alle Gewichte negativ
 - ◆ LP in P, SP ist NP-vollständig
 - Keine positiven Zyklen: LP in P
 - Keine negativen Zyklen: SP in P
 - Sonst: NP-vollständig

Kritische ./.. Unkritische Elemente



■ Layout-Breite

- Hängt nur von kritischen Elementen ab

■ Unkritische Elemente: Verschiebbar

- Beeinflussen aber weitere Iterationen

Kompaktierungsdetails

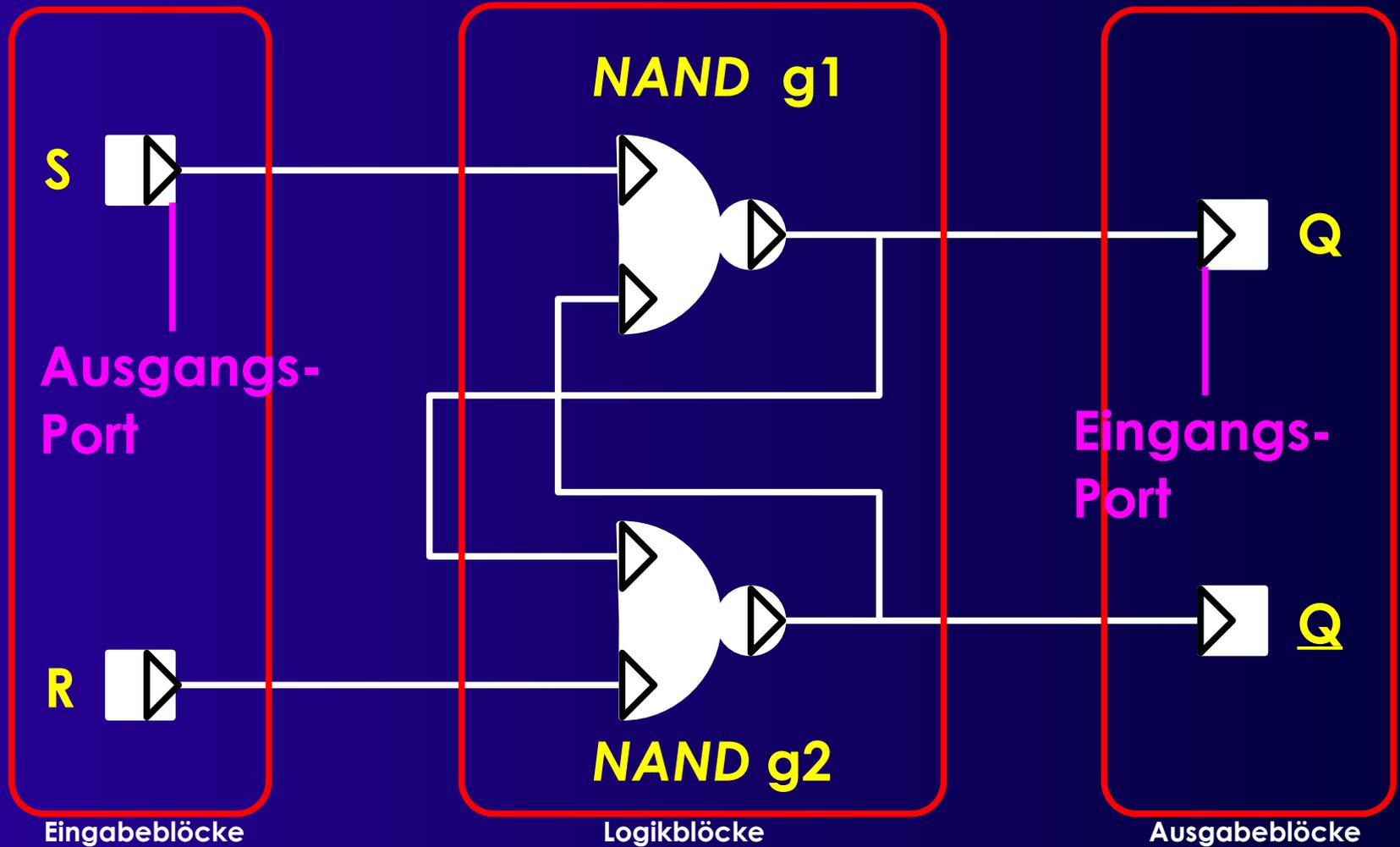
- **Freie Layoutelemente**
 - Optimale Lösung ist 2D-Kompaktierung
- **Einfügen von Jogs (Knicke in Leitungen)**
- **Berechnung der Einschränkungen**
 - Einfacher n^2 -Ansatz: Redundanzen
- **Hierarchisches Vorgehen**

Darstellung von Schaltungen 1

Eingangs-
Terminals

Instanzen

Ausgangs-
Terminals



Darstellung von Schaltungen 2

■ Instanz oder Zelle

- Ein Auftreten einer Master-Zelle
- Speichert Instanz-spezifische Eigenschaften
 - ◆ z.B. Name

■ Master-Zelle

- Speichert Eigenschaften aller Instanzen
 - ◆ z.B. Funktion, Ports, Layout, ...

■ Netz

- Verbindung von mehreren Ports

■ Port

- Anschlusspunkt von Leitung an Zelle
- I.d.R. nicht untereinander austauschbar
- Hierarchie: Terminals werden zu Ports

Darstellung von Schaltungen 3

```
class cell_master {  
    String name;  
    truth_table func;  
    Rect extent;  
    set<port_master> ins, outs;  
    ...  
};
```

```
class cell {  
    cell_master master;  
    String name;  
    set<port> ins, outs;  
    ...  
};
```

```
class net {  
    String name;  
    set<port> joined;  
}
```

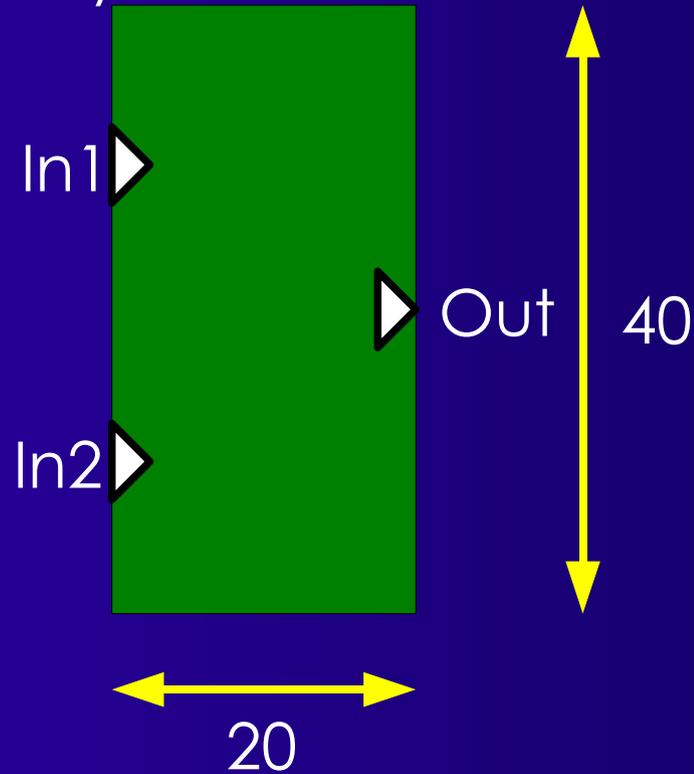
```
class port_master {  
    String name;  
    Point location;  
    ...  
}
```

```
class port {  
    port_master master;  
    String id;  
    cell parent;  
    net connects;  
    ...  
}
```

Darstellung von Schaltungen 4

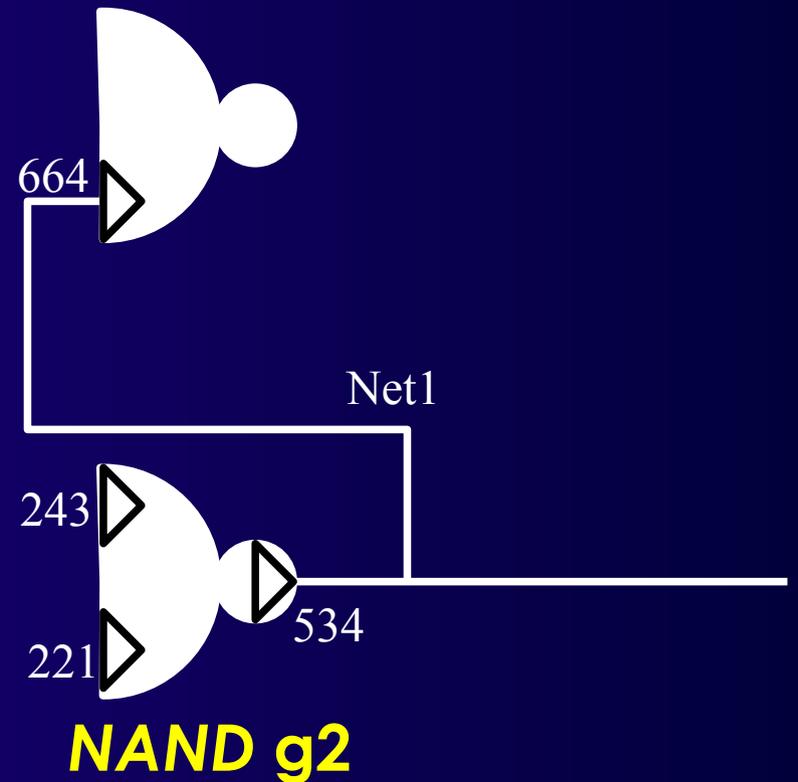
Master

Layout von NAND

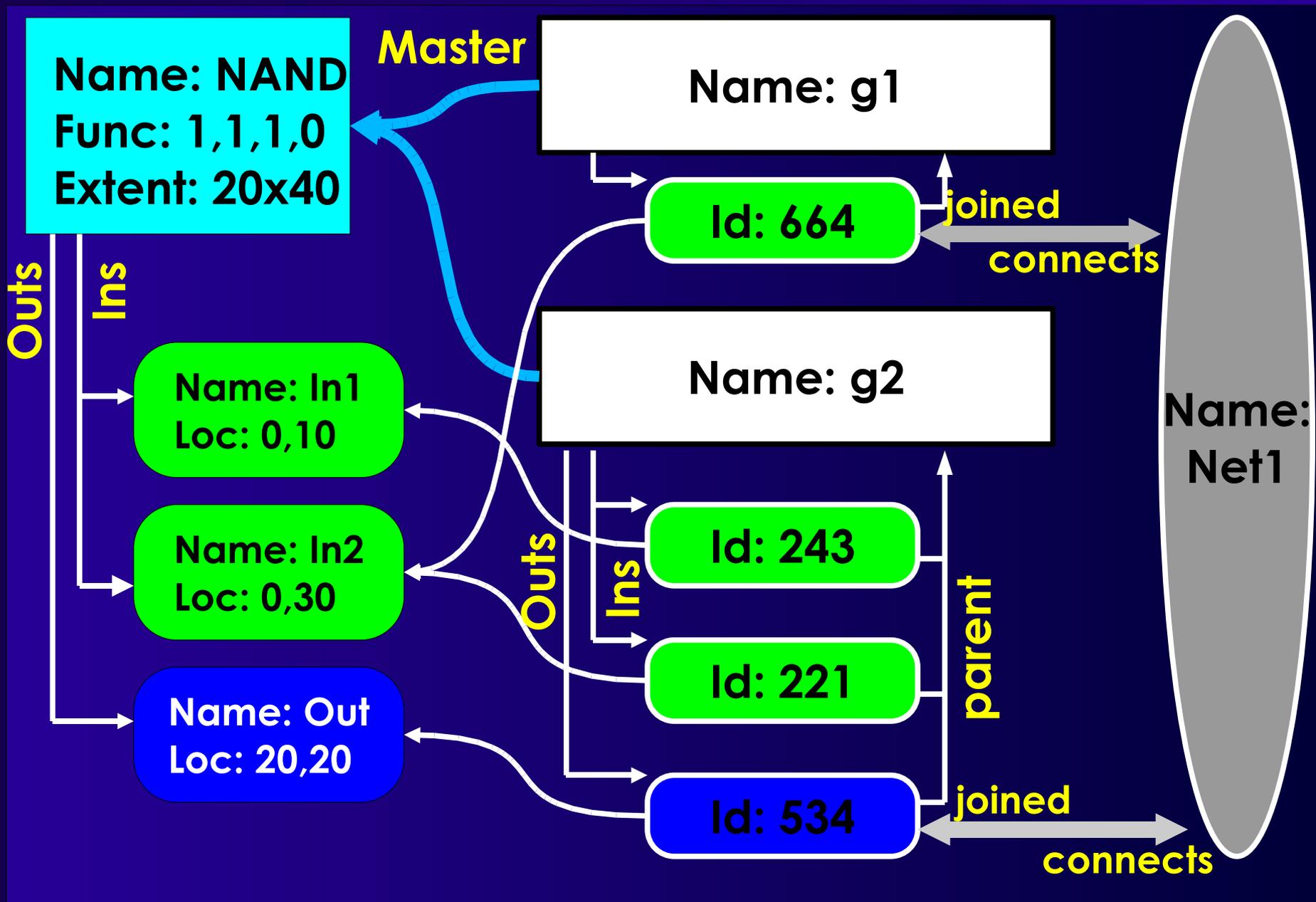


Schaltungsfragment

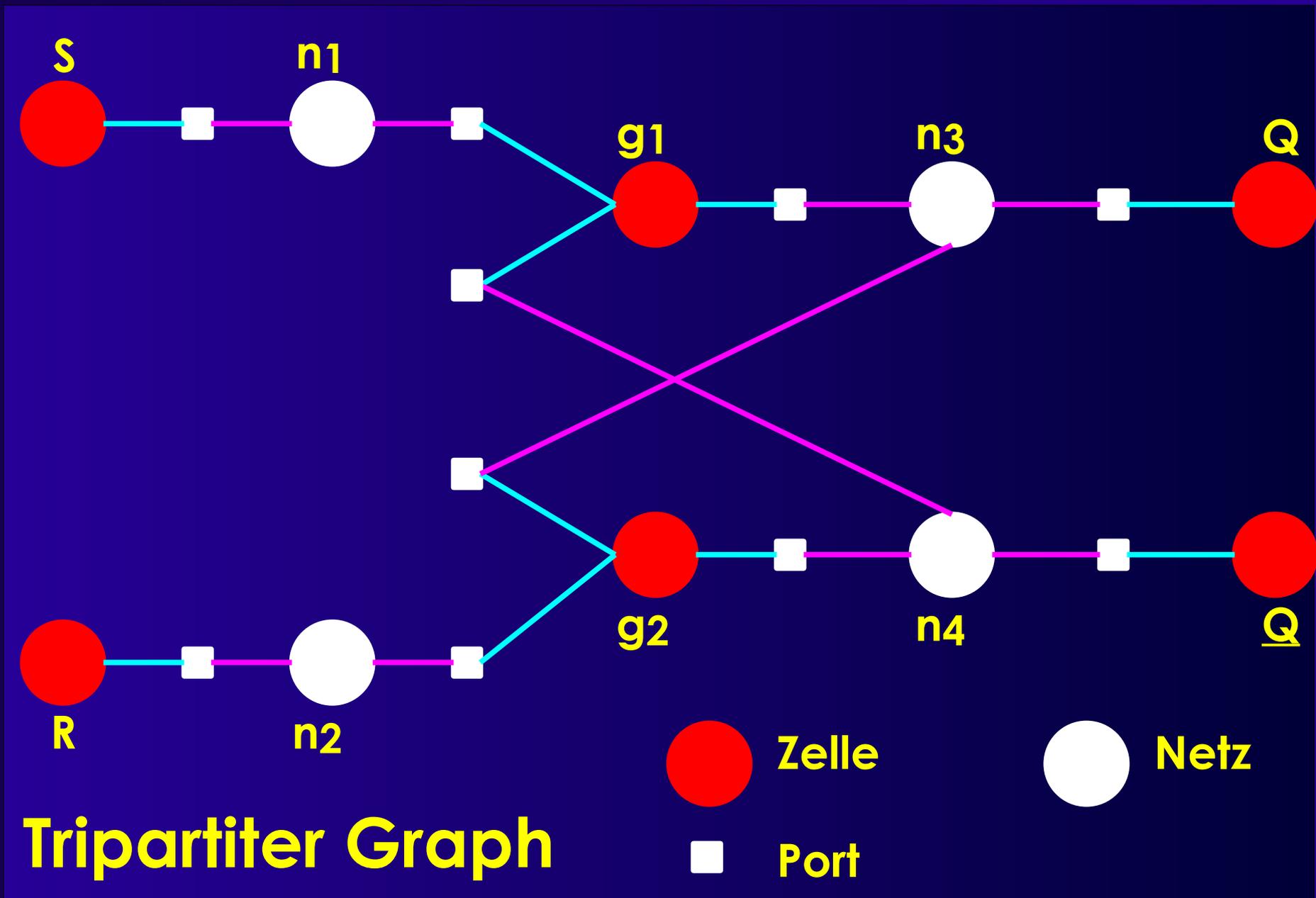
NAND g1



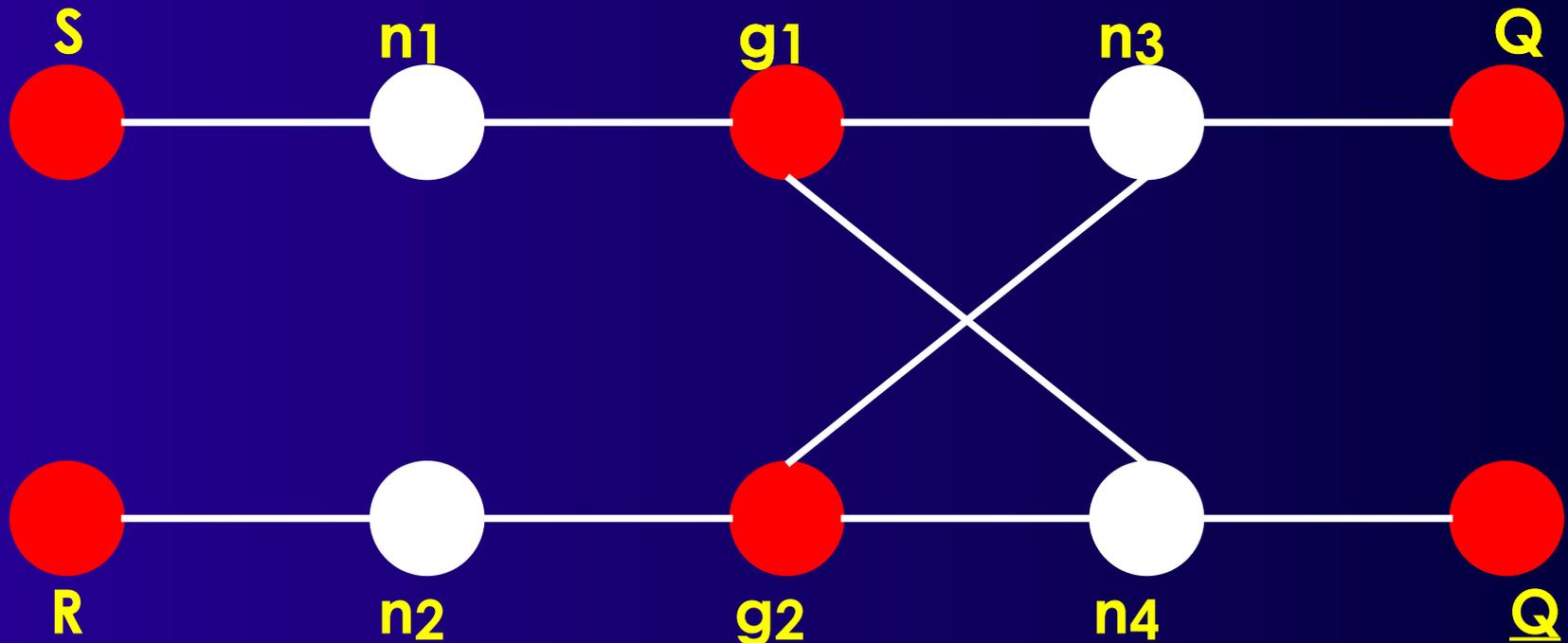
Darstellung von Schaltungen 5



Schaltungen als Graphen 1



Schaltungen als Graphen 2

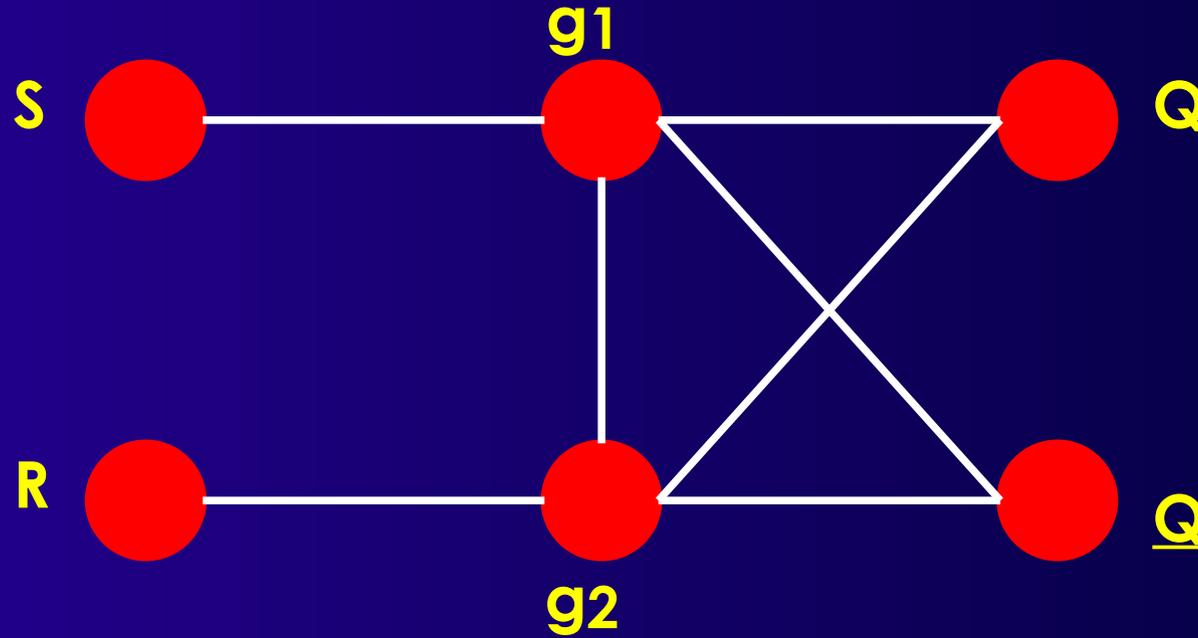


■ Bipartiter Graph

- Weniger Details
- Verschmelze Ports mit Zellen
- Äquivalent zu *Hypergraph*



Schaltungen als Graphen 3



■ Cliquen-Modell

- Netze nicht mehr explizit modelliert
- Zellen an Netzen bilden jetzt Clique

 Zelle

Schaltungsdarstellungen

- Zelle-Port-Netz Modell
- Tripartiter Graph
- Bipartiter Graph
- Clique-Modell

Ungenauer



- Für Problem *passendes* Modell wählen
 - Mehr Daten nicht immer besser
- Konvertierungsroutinen bereitstellen
 - Nur in ungenauere Darstellung möglich
 - Buchführen über Herkunft von Daten

Weiteres Vorgehen

■ Dienstag

- Kick-Off für praktische Arbeiten
- Vorher zu 3er Gruppen zusammenfinden
- Vorher den Leitfaden lesen
 - ◆ ... um gezielt Fragen stellen zu können

■ Nächste Vorlesung: Freitag

■ Allgemeine Vorbereitung

- Buch Kapitel 5.5 - 5.9

Zusammenfassung

- **Kompaktierung**
- **Berechnung der längsten Pfade**
 - Ohne und mit Zyklen
- **Modellierung von Schaltungen**
 - Graphbasiert
 - Hierarchisch