

# Algorithmen im Chip-Entwurf 2

## Kompaktierung, Schaltungsdarstellungen und Timing-Analyse

Andreas Koch  
FG Eingebettete Systeme  
und ihre Anwendungen  
TU Darmstadt

Kompaktierung, Schaltungsdarstellungen und Timing-Analyse

# Organisatorisches

- **Vorgehensweise**
  - **Anmeldebögen**
- **Wichtige Spalten ganz rechts: Ankreuzen**
  - **IV 4 SWS**
    - ◆ Sie wollen nur das ganze Programm
    - ◆ Falls Sie keinen Platz bekommen, vertagen Sie sich
  - **IV 4 SWS / Fallback VL 2 SWS**
    - ◆ Sie versuchen einen Platz zu bekommen
    - ◆ Falls Sie keinen kriegen, hören Sie nur die VL
  - **VL 2 SWS**
    - ◆ Sie wollen ohnehin nur die VL hören
  - **Gar nix**
    - ◆ Sie setzen in diesem Durchgang aus

# Übersicht

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

# Transistorschaltungen

$V_{DD}$

$V_{DD}$

$V_{in}$

$V_{in}$

$V_{in}$

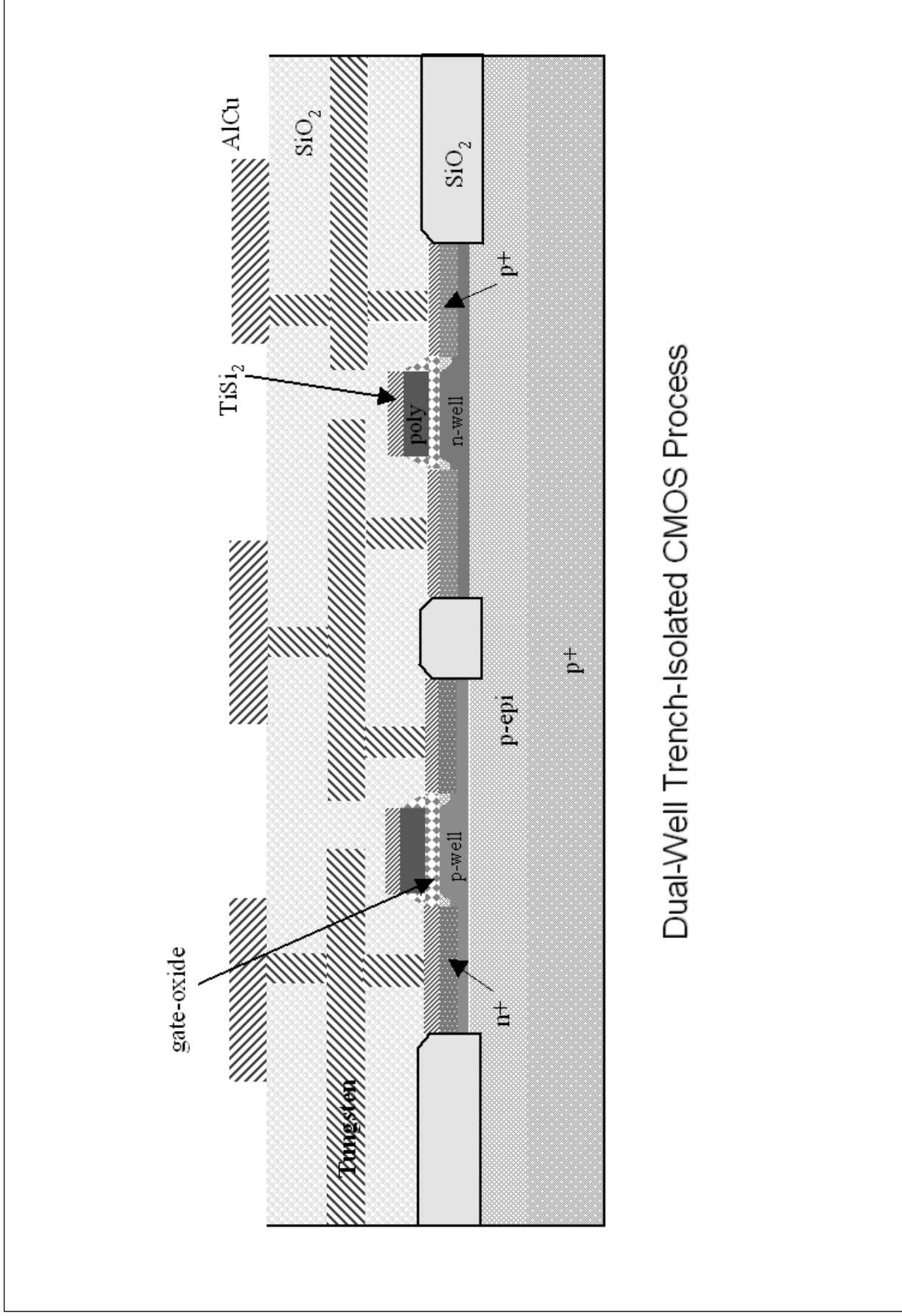
$V_{out}$

$V_{out}$

$V_{in}$

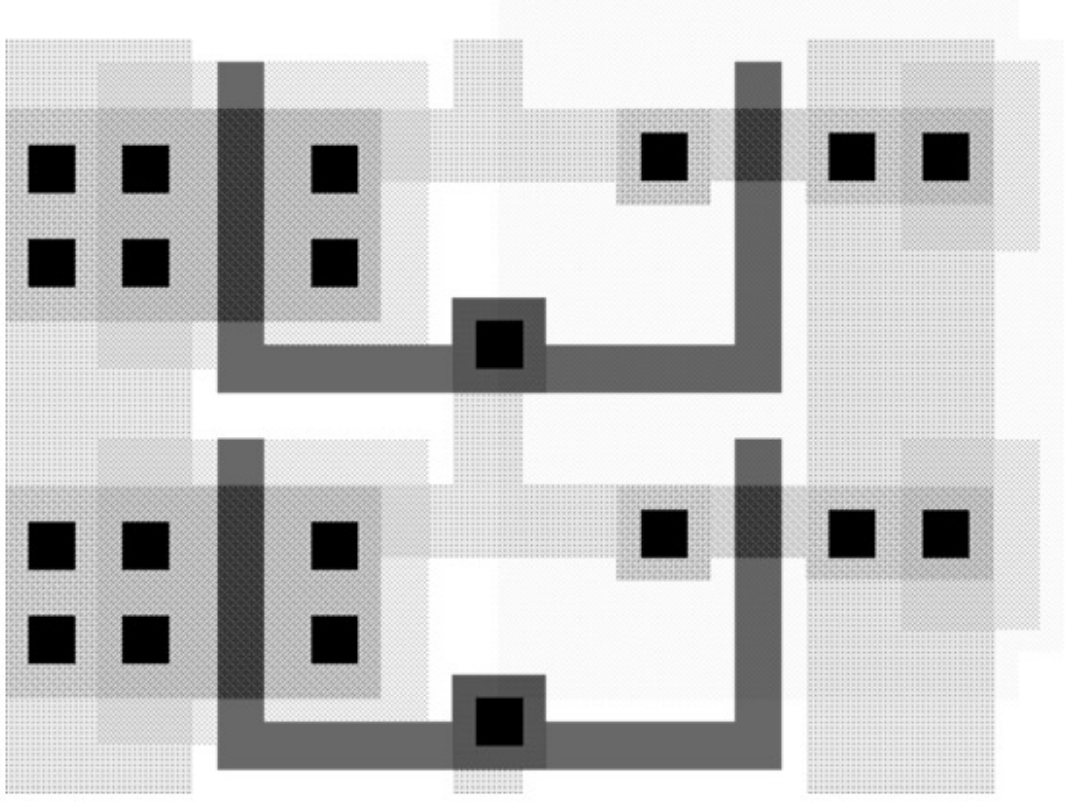
$V_{in}$

# Seitenansicht durch Chip

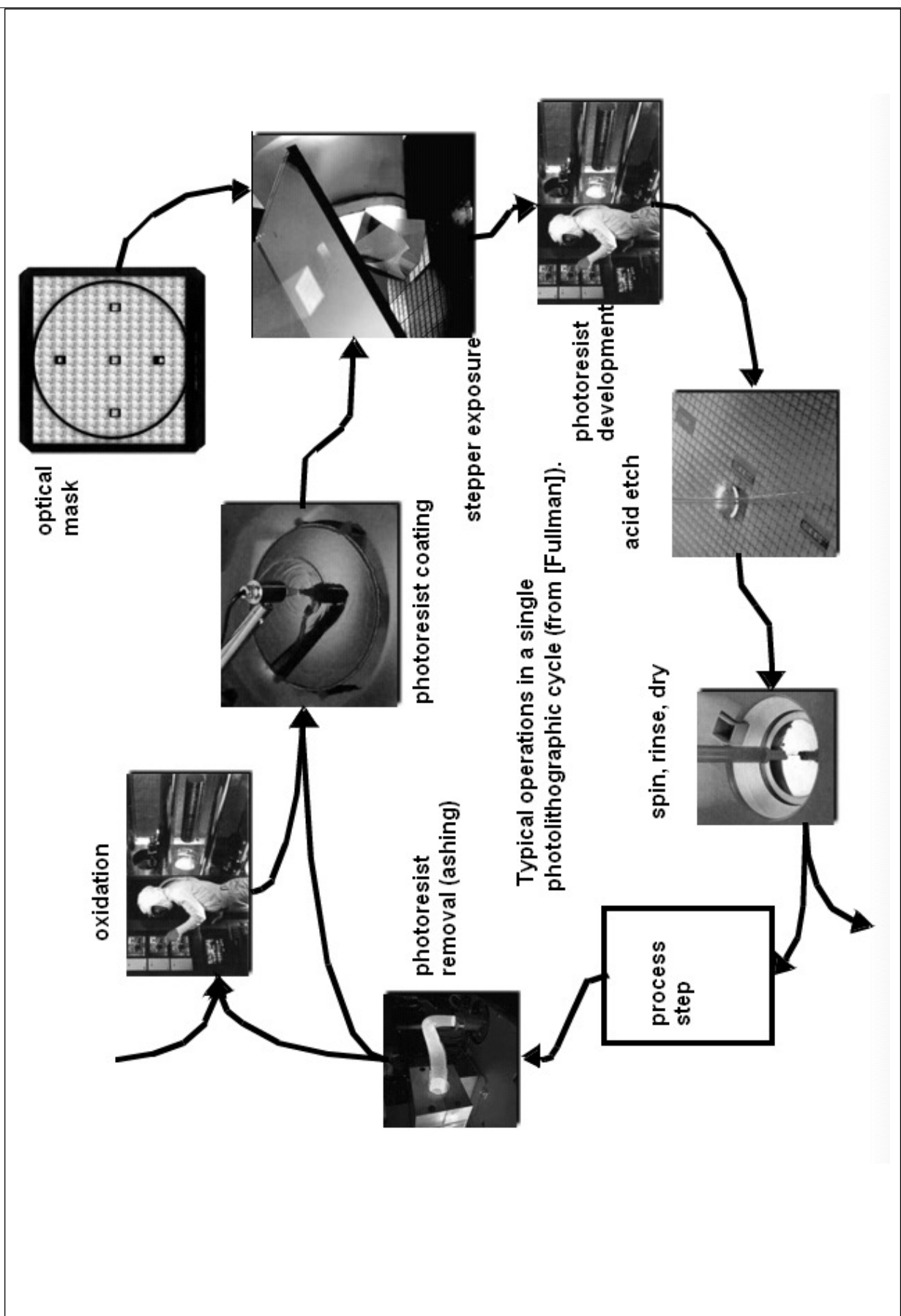


Dual-Well Trench-Isolated CMOS Process

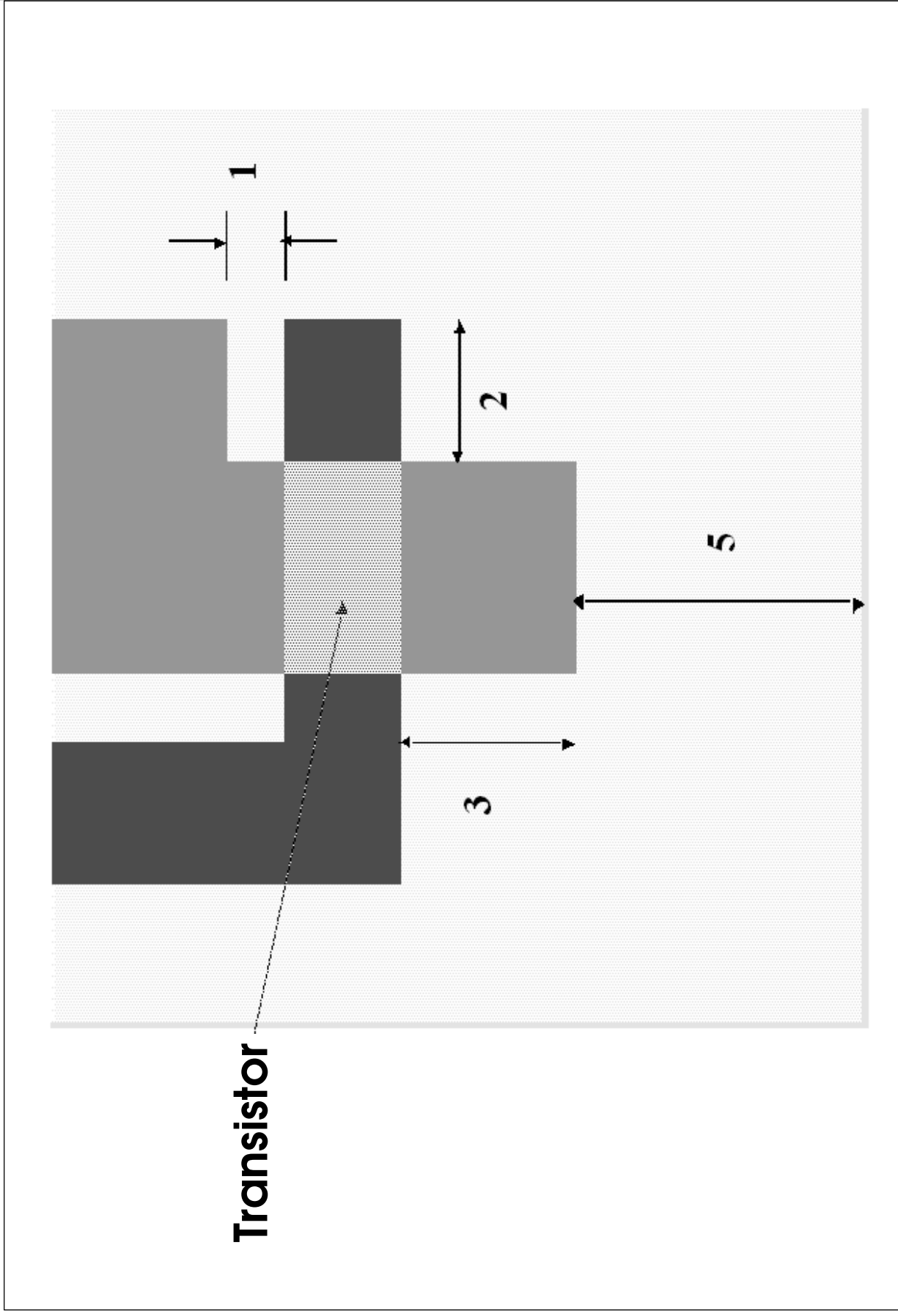
# Layout-Sicht



# Fertigung

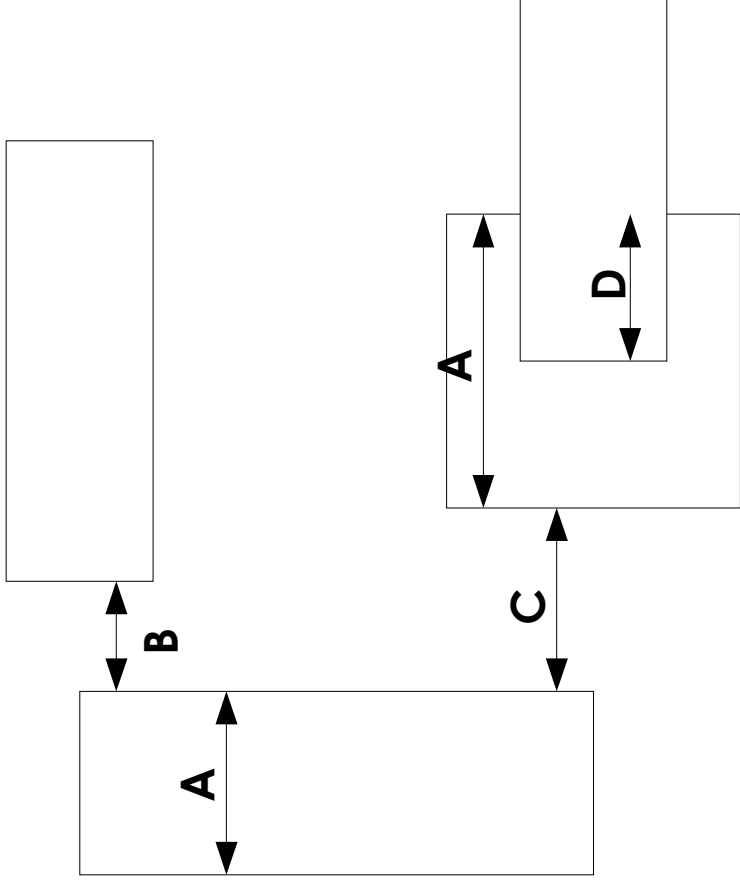


# Entwurfsregeln 1





# Entwurfsregeln 2



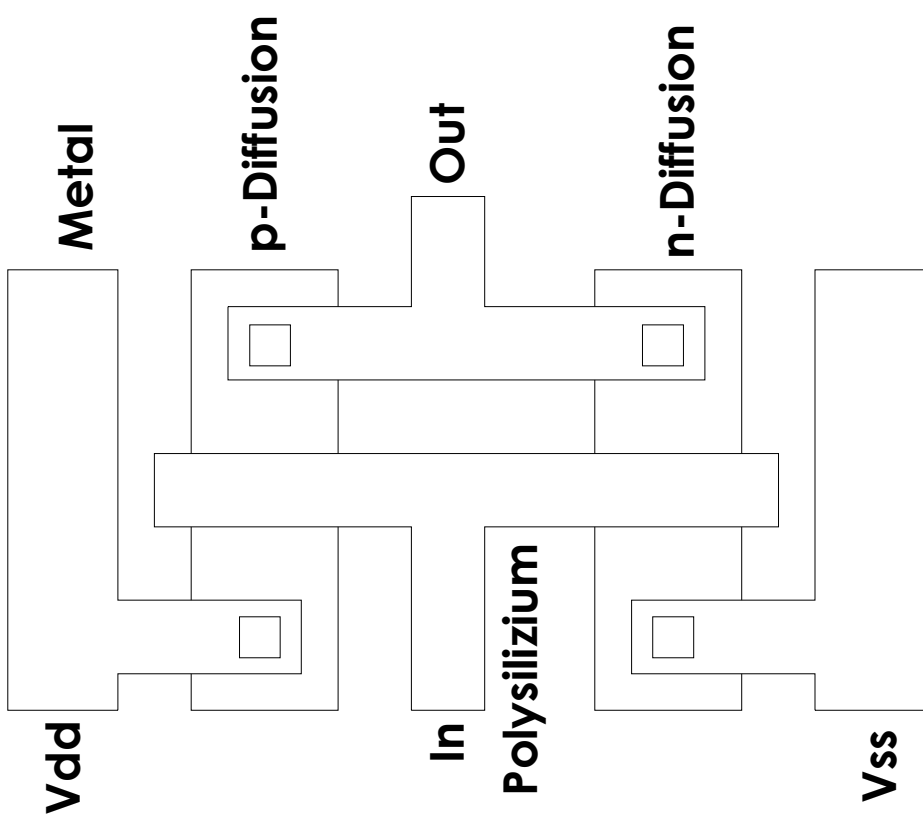
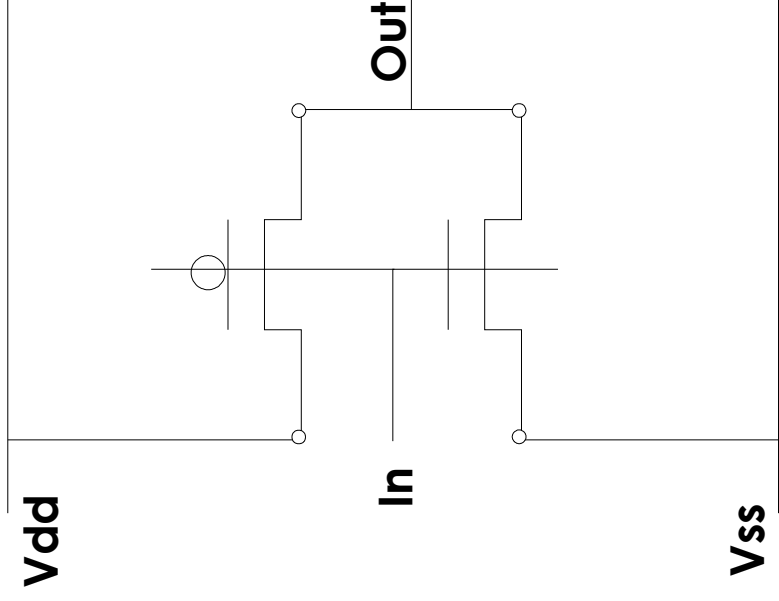
- A – minimale Breite
- B – minimaler Abstand (L1-L2)
- C – minimaler Abstand (L1-L1)
- D – minimale Überlappung

- 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



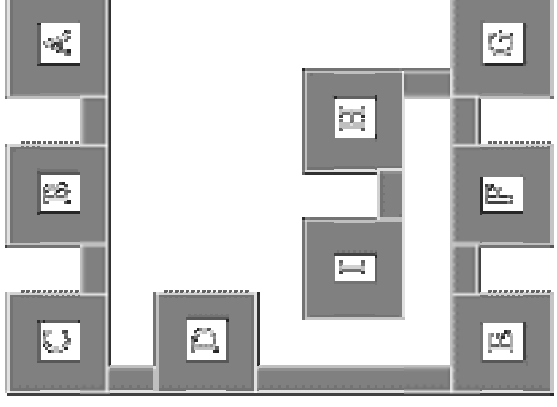
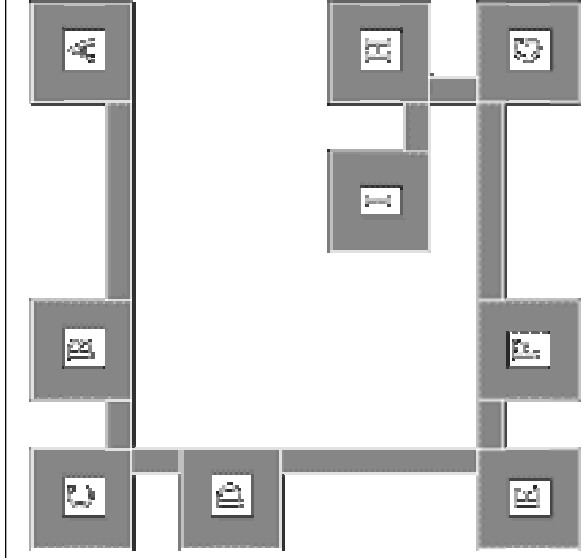
# Kompaktierung

- **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

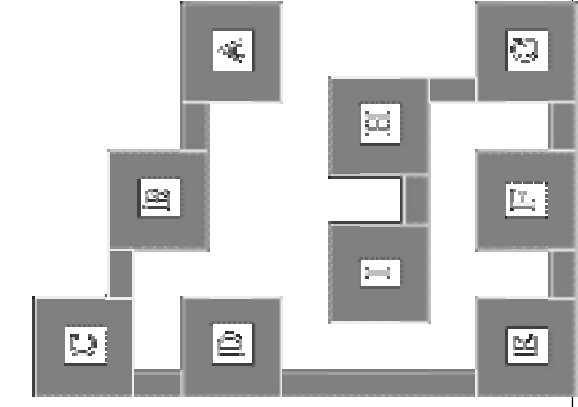
# Vorgehensweise

- **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



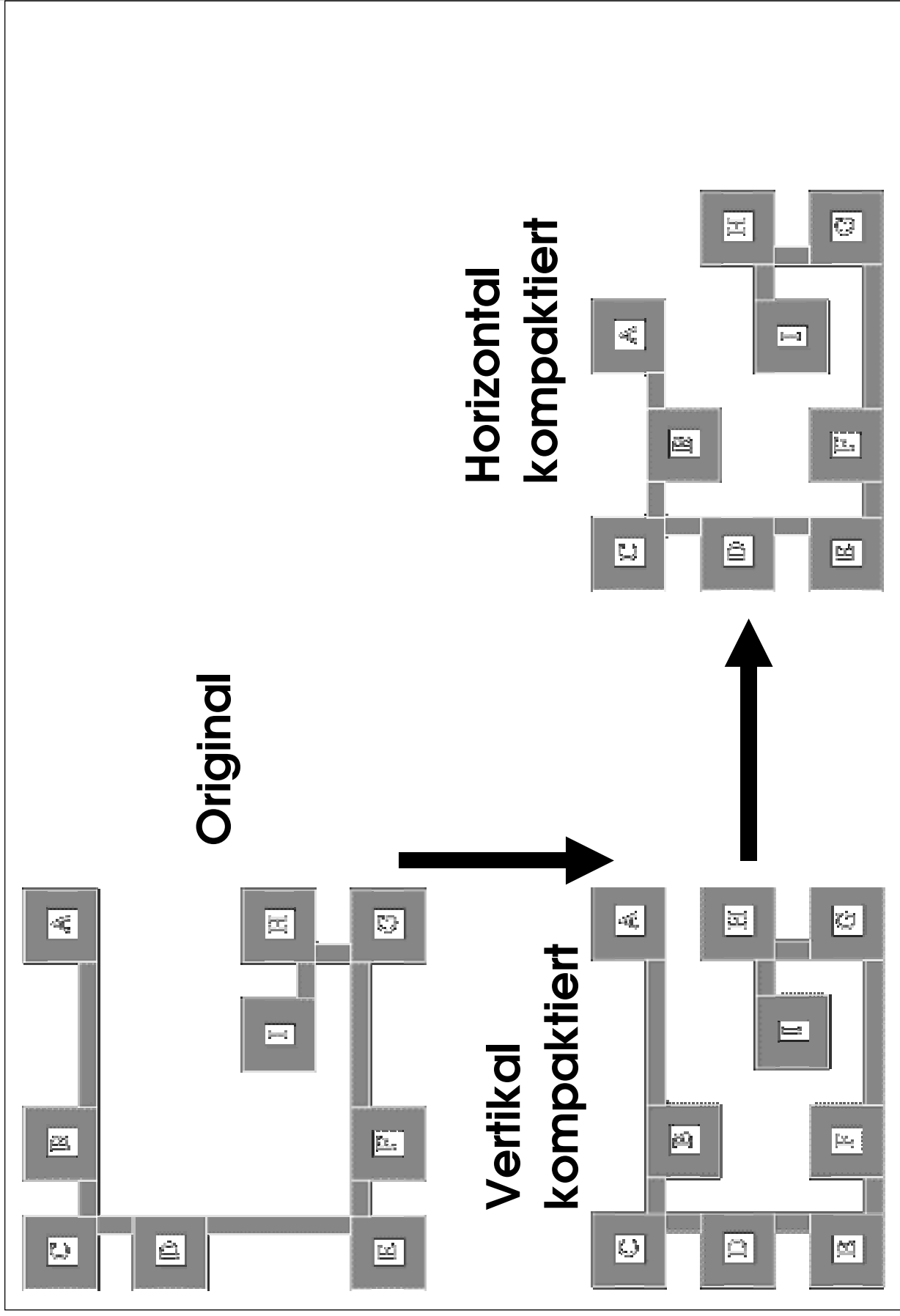
Horizontal  
kompaktiert



Vertikal  
kompaktiert

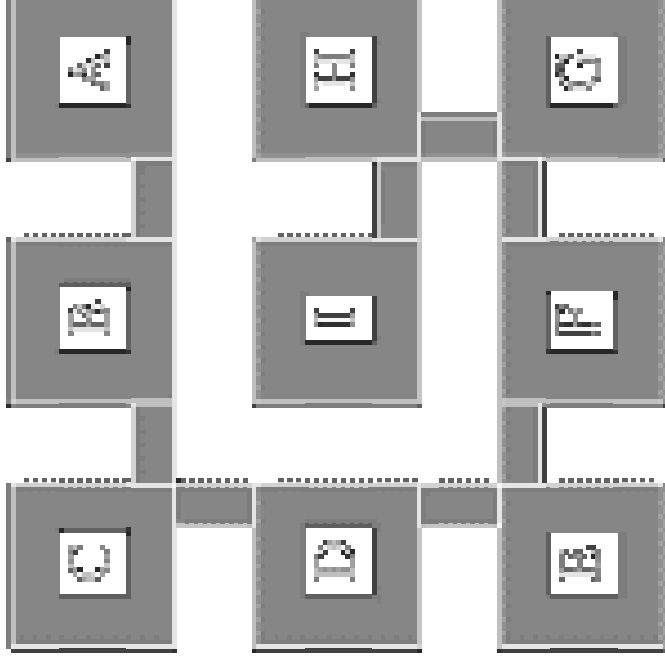
Original

# Kompaktierung 2



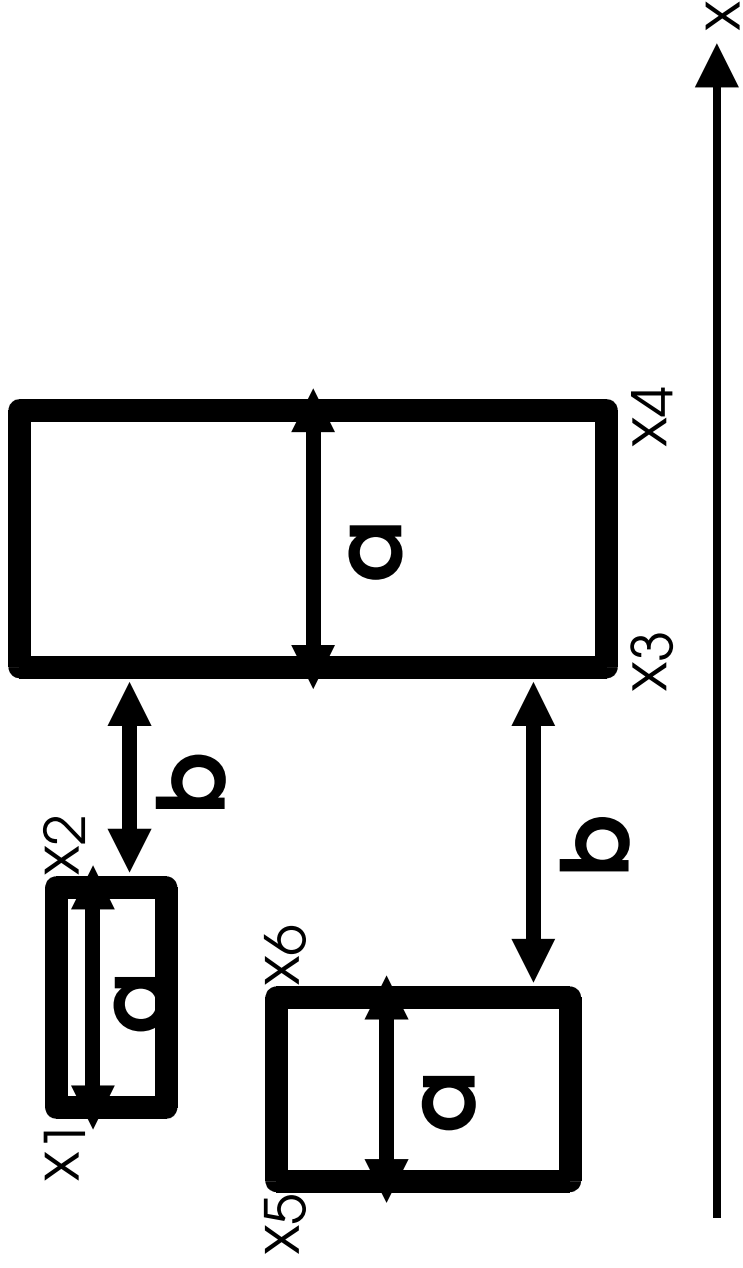
# Kompaktierung 3

- 2D Kompaktierung
  - Optimal
  - NP-vollständig
- Vorgehensweise
  - Mehrfache 1D-K
  - Wechselnd in H, V
  - Aber: nicht optimal





# Modellierung 1



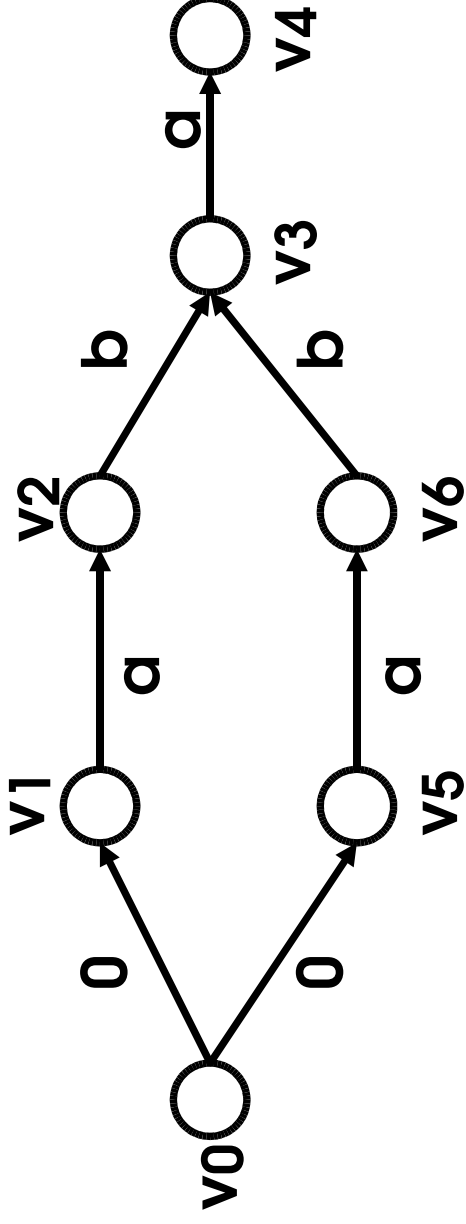
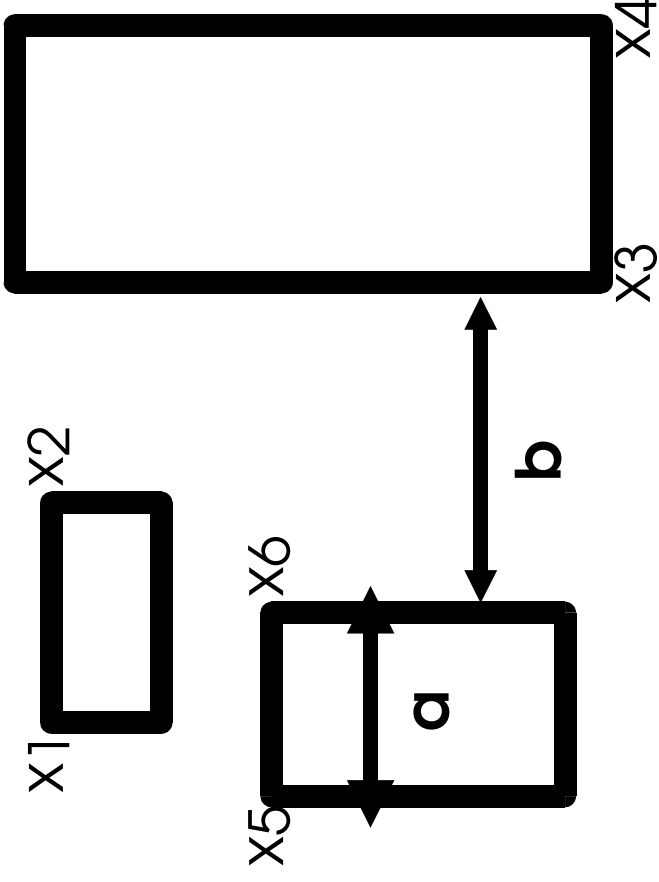
$$x_2 - x_1 \geq a$$

$$x_3 - x_2 \geq b$$

$$x_3 - x_6 \geq b$$

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

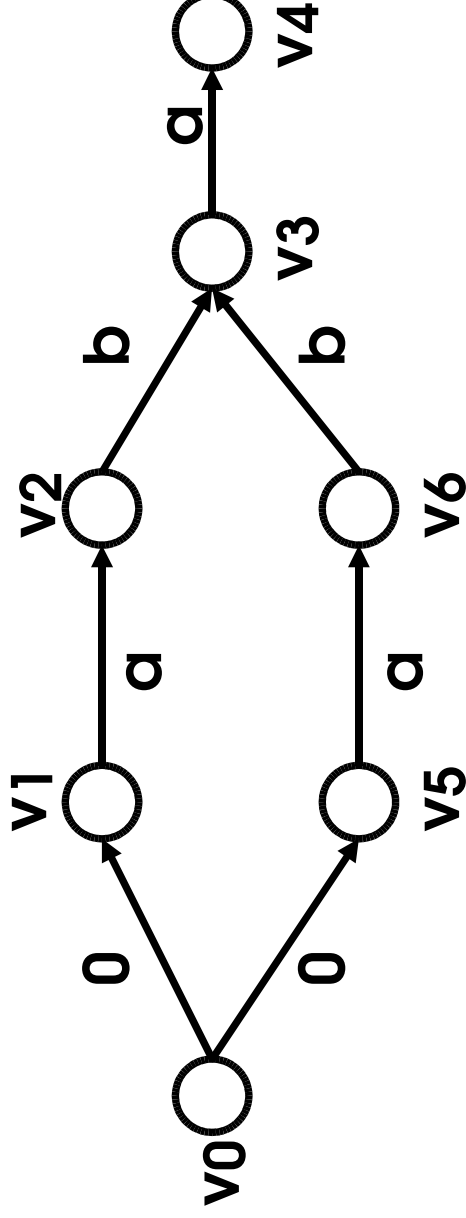
# Modellierung 2



# Modellierung 3

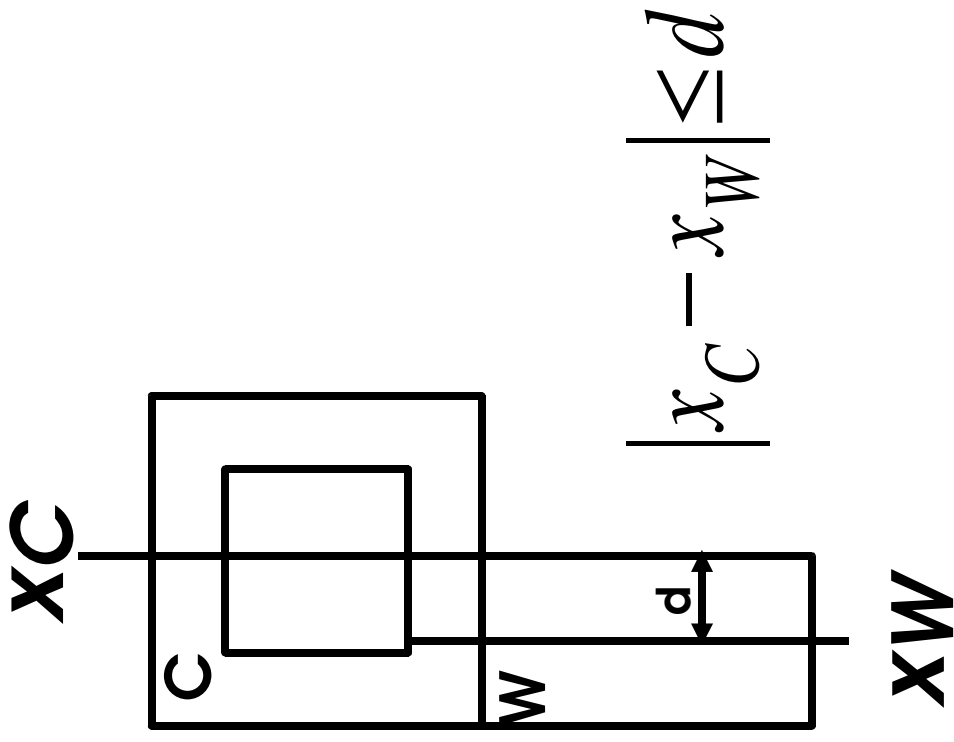
## ■ Einschränkungsgraph $G(V, E)$

- Gerichtet von  $(v_i, v_j)$
  - Zyklentfrei
- ## ■ Längster Pfad von $v_0$ zu $v_i$
- Minimale Koordinate von  $x_i$



# Modellierung 4

- Bedingungen an *maximalen* Abstand

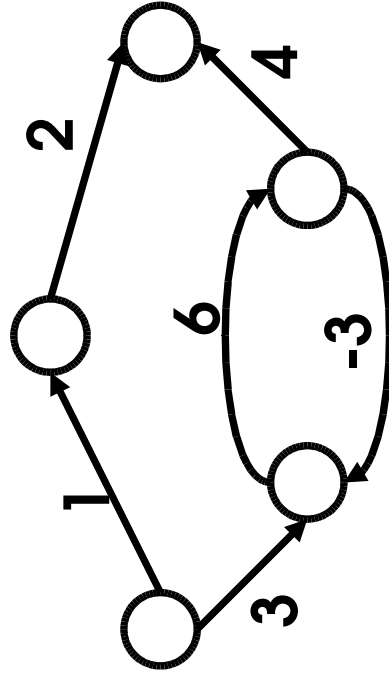
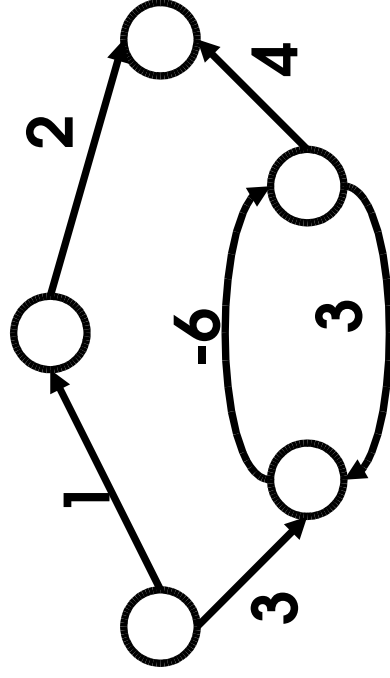


# Modellierung 5

- $|x_C - x_W| \leq d$ 
  - $x_j - x_i \leq c_{ij}$  ,  $c_{ij} \geq 0$
  - $x_i - x_j \geq -c_{ij}$
- Passende Form für Einschränkungsgraph
  - Jetzt aber Richtung ( $v_j$ ,  $v_i$ ): Zyklen möglich!
- Aufgabe:
  - Berechnung des längsten Pfades in Graphen mit Zyklen

# Längster Pfad

- **Zyklenfreie Graphen**
  - OK, ähnlich BFS
- **Graphen mit Zyklen**
  - Ohne positiven Zyklus: OK
  - Mit positivem Zyklus: undefiniert
  - ◆ Kompaktierung: Überbeschränktes Layout



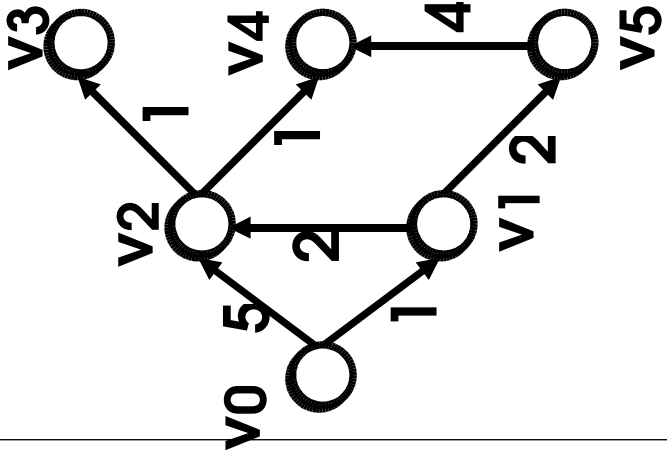
# Zyklusfreie Graphen 1

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

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);
  }
}

- Directed Acyclic Graph (DAG)
- Längster, gerichteter, einfacher Pfad (trail)
  if (pj ≤ 0)
    Q := Q ∪ {vj};
}
```

# Zyklusfreie 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

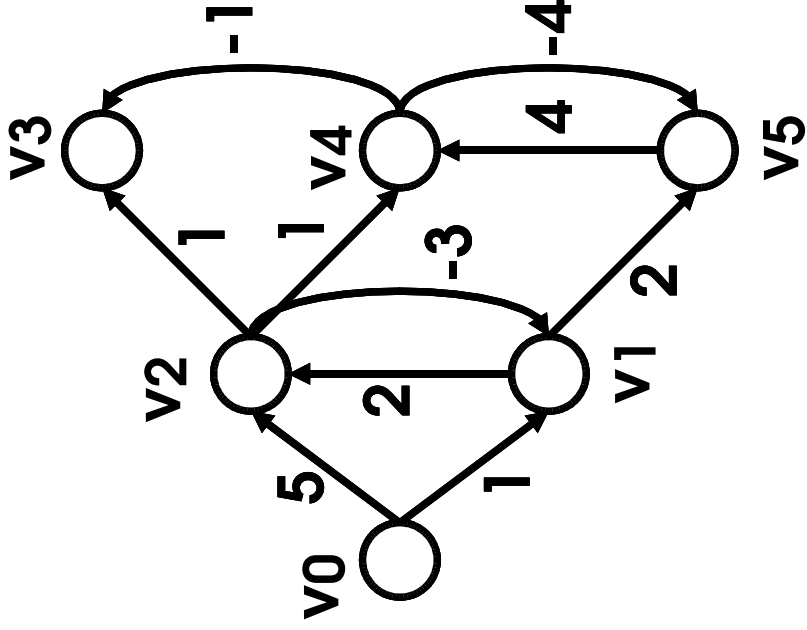
- Nur mit *negativen* Zyklen
- Erkenne *positive* Zyklen
  - Überbeschränkte Layouts
- Aber lokalisiere sie nicht

# 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<sub>f</sub>: min. Distanz
  - ◆ E<sub>b</sub>: max. Distanz
    - ◆ Erzeugen Zyklen!
- Berechne LP G<sub>f</sub>(V, E<sub>f</sub>)
- Korrigiere für E<sub>b</sub>
  - ◆ Schließen Zyklen
- Stabilisiert sich in |E<sub>b</sub>|
  - ◆ Jedes e<sub>b</sub> 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 Ef und Eb
- Vergleichbar azyklischem LP
  - Aber mehrere Durchläufe durch Graph

# LP mit Bellman-Ford 2

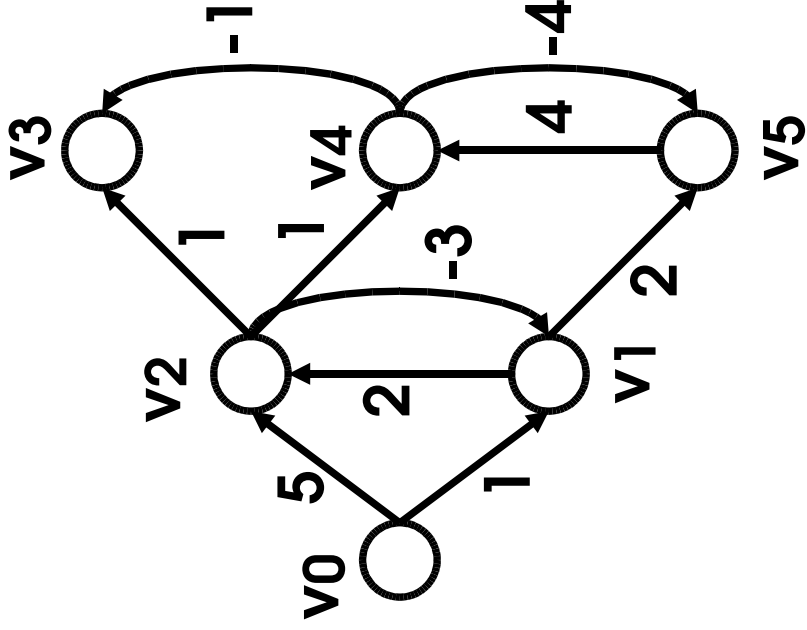
```
for (i:=1; i ≤ n; ++i)
  xi := -∞;
x0 := 0;
count := 0;
S1 := {v0};
S2 := ∅;
while (count ≤ n && S1 ≠ ∅) {
  foreach vi ∈ S1
    foreach (vi, vj) ∈ E
      if (xj < xi + dij) {
        xj := xi + dij;
        S2 := S2 ∪ {vj};
      }
  S1 := S2;
  S2 := ∅;
  ++count;
}
```

## ■ 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.  $O(n^{1.5})$

```
if (count > n) error("positive cycle!");
```

# LP mit Bellman-Ford 3

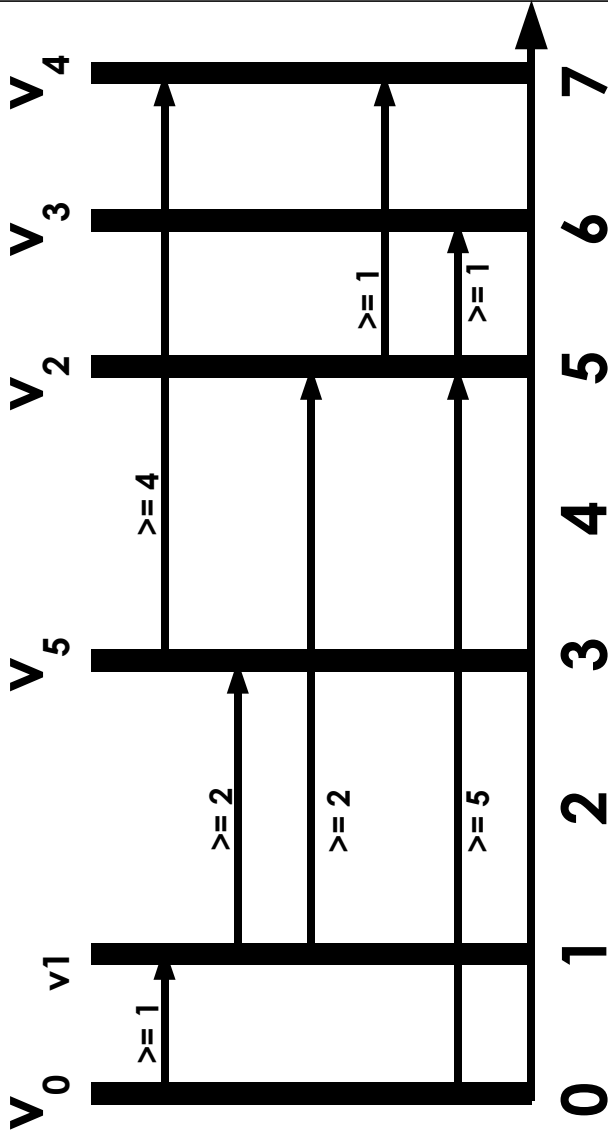
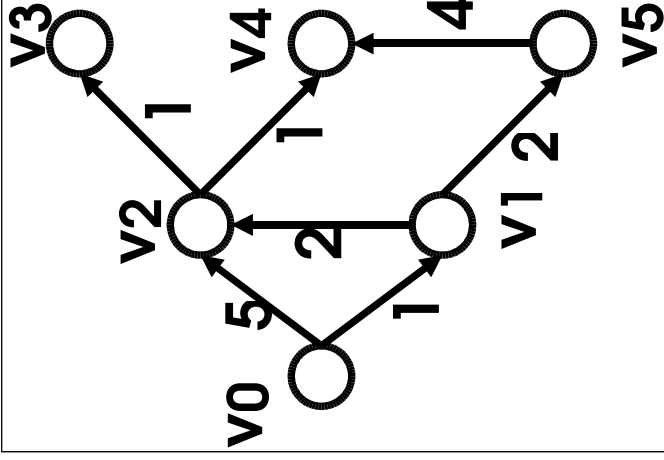


S1	X1	X2	X3	X4	X5
nil	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
{v0}	1	5	$-\infty$	$-\infty$	$-\infty$
{v1, v2}	2	5	6	6	3
{v1, v3, v4, v5}	2	5	6	7	4
{v4, v5}	2	5	6	8	4
{v4}	2	5	7	8	4
{v3}	2	5	7	8	4

# 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 (Dijkstra), 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

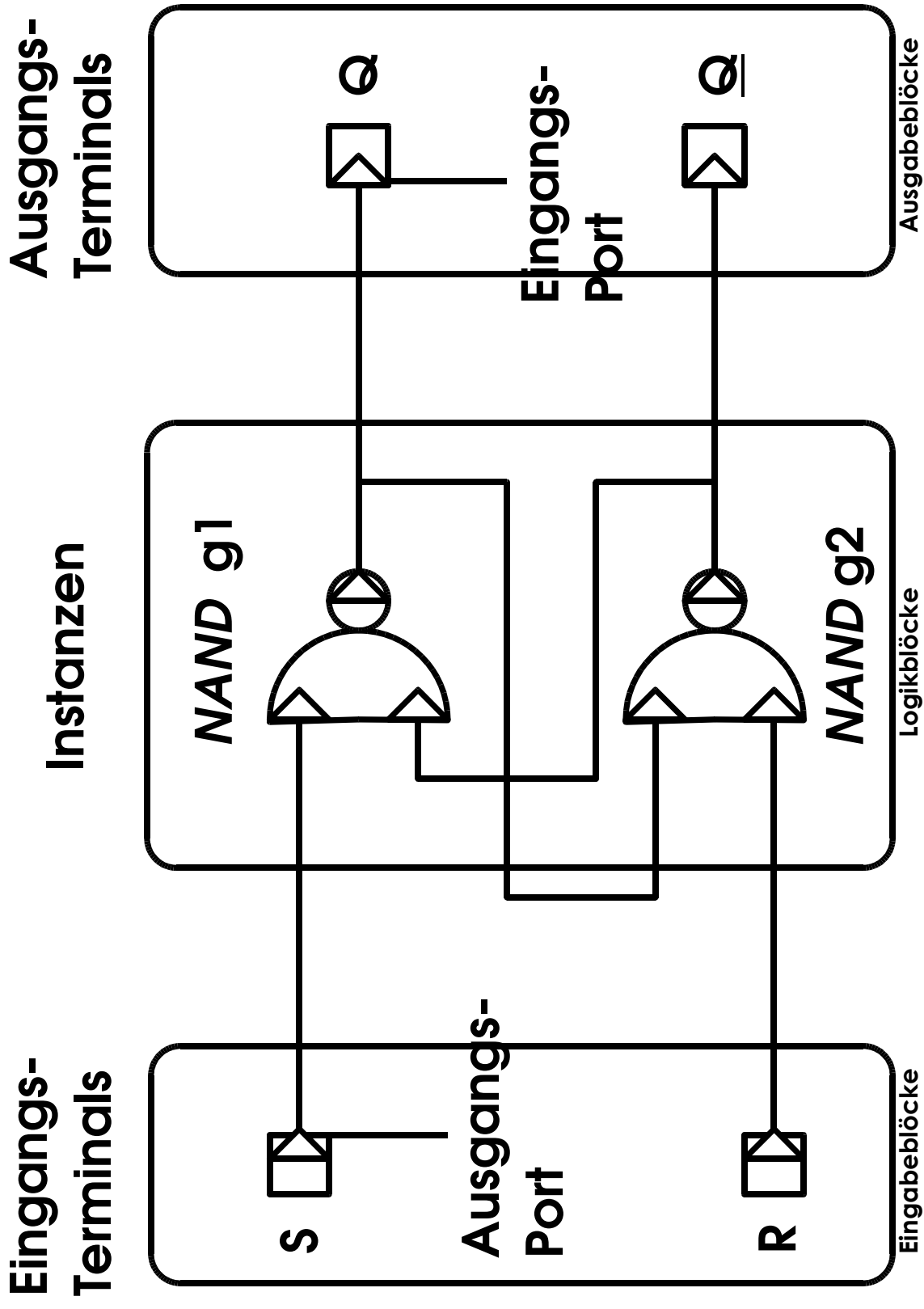
- Hängt nur von kritischen Elementen ab
- Unkritische Elemente: Verschiebbar
- Beeinflussen aber weitere Iterationen



# 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



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

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

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

# Darstellung von Schaltungen 4

## Master

Layout von NAND

In1

In2



Out

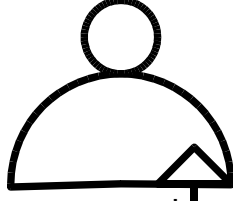
40



20

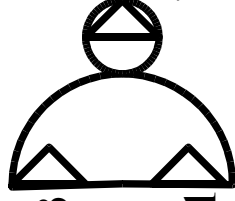
## Schaltungsfragment

NAND g1



664

Net1



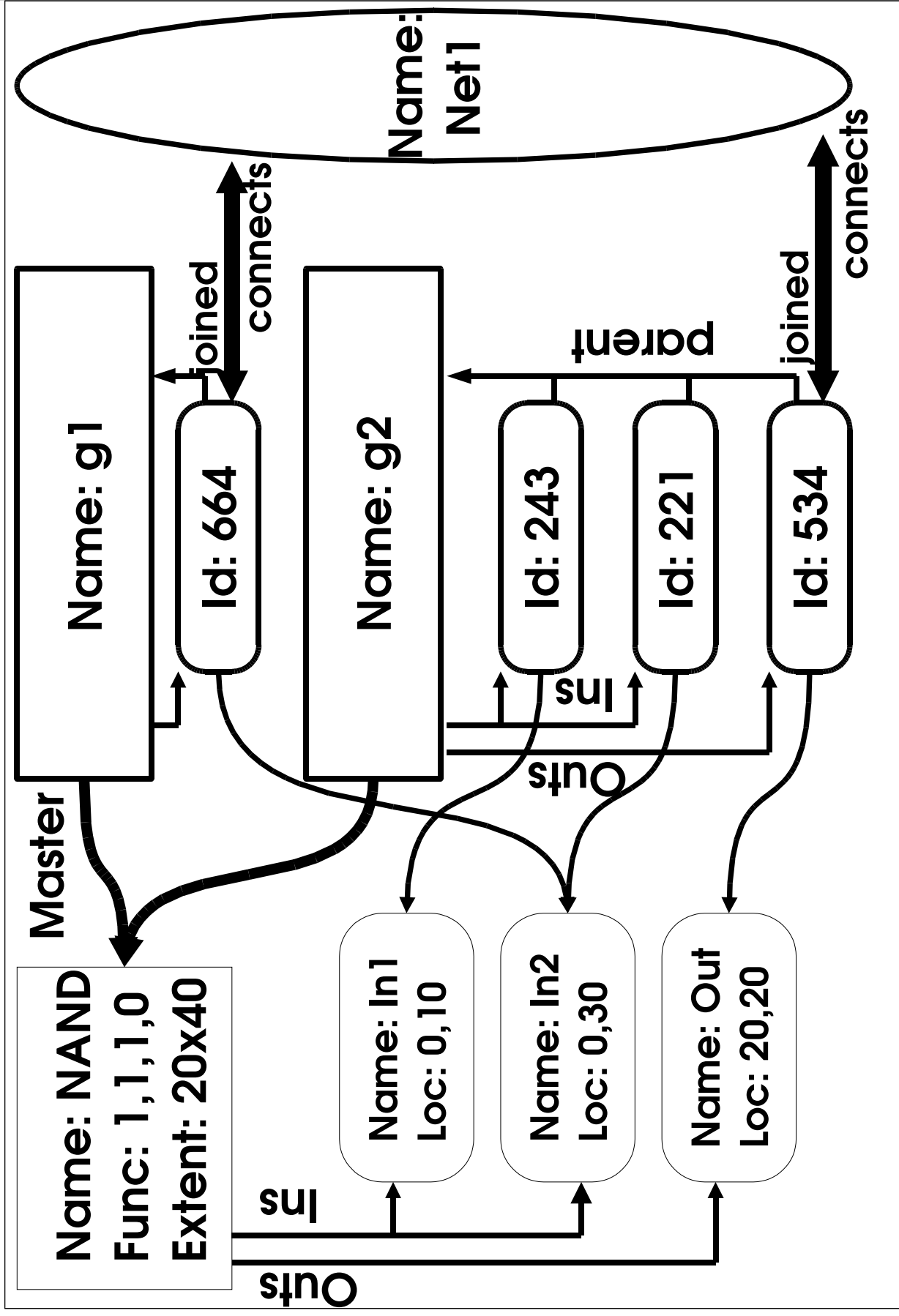
243

221

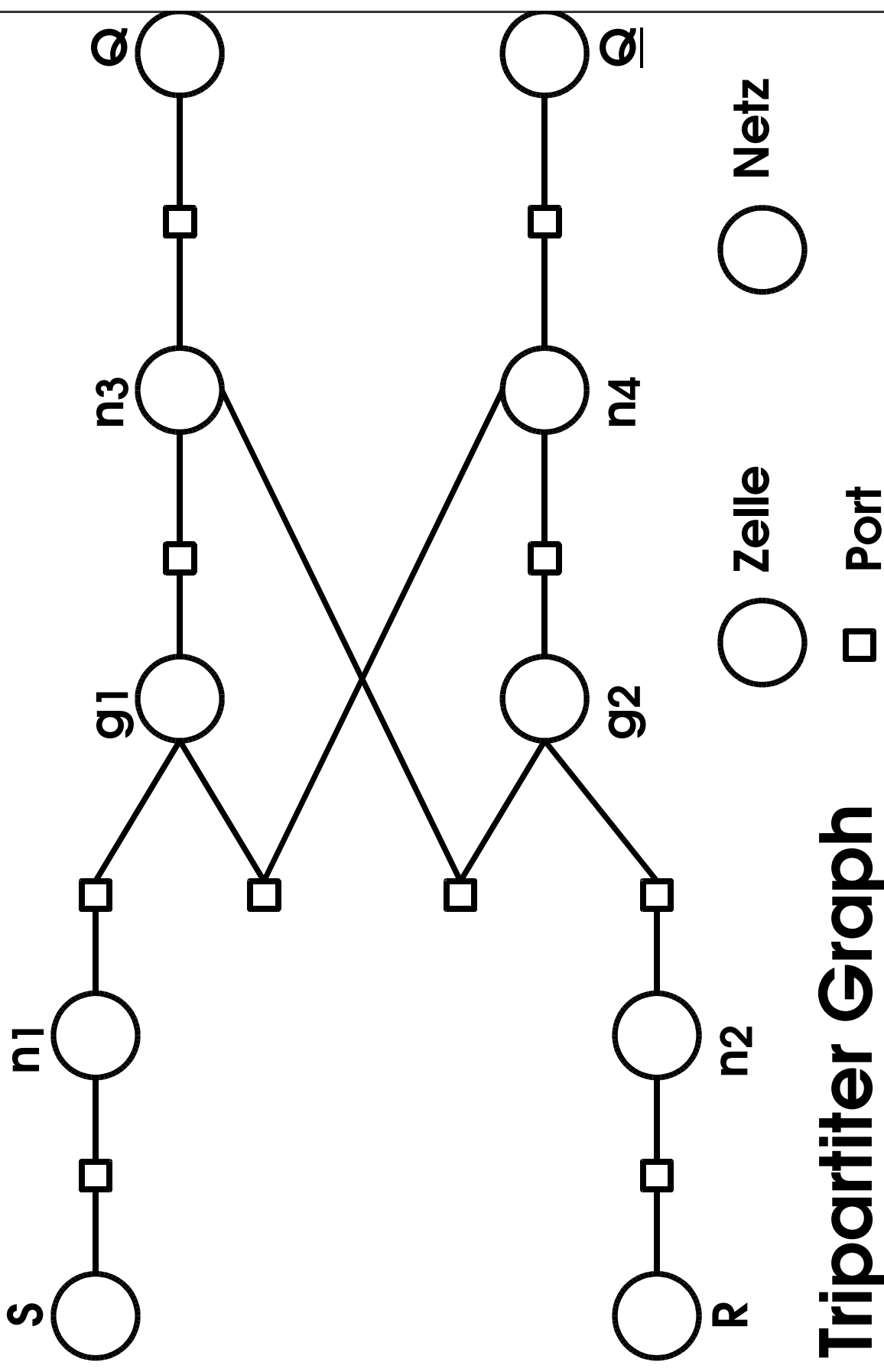
534

NAND g2

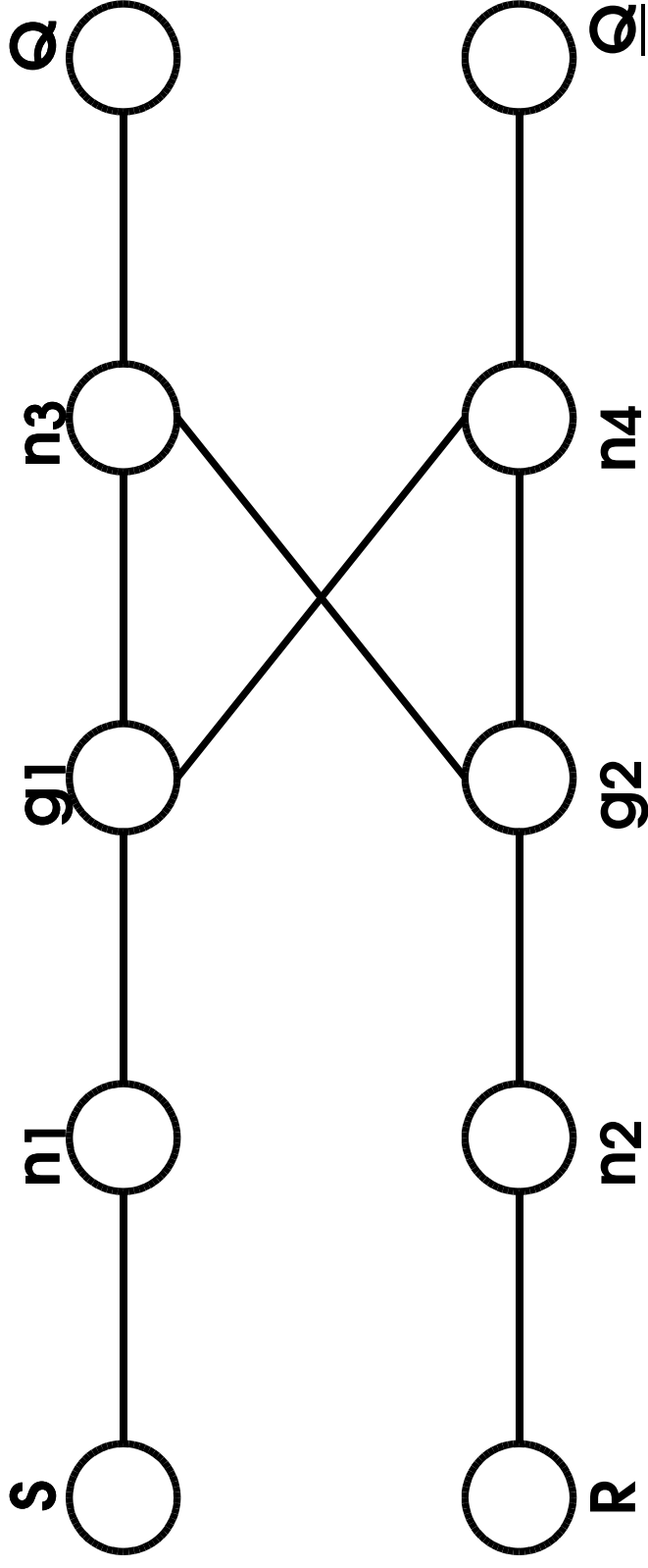
# Darstellung von Schaltungen 5



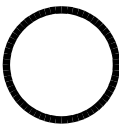
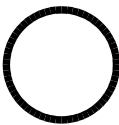
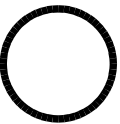
# Schaltungen als Graphen 1



# Schaltungen als Graphen 2

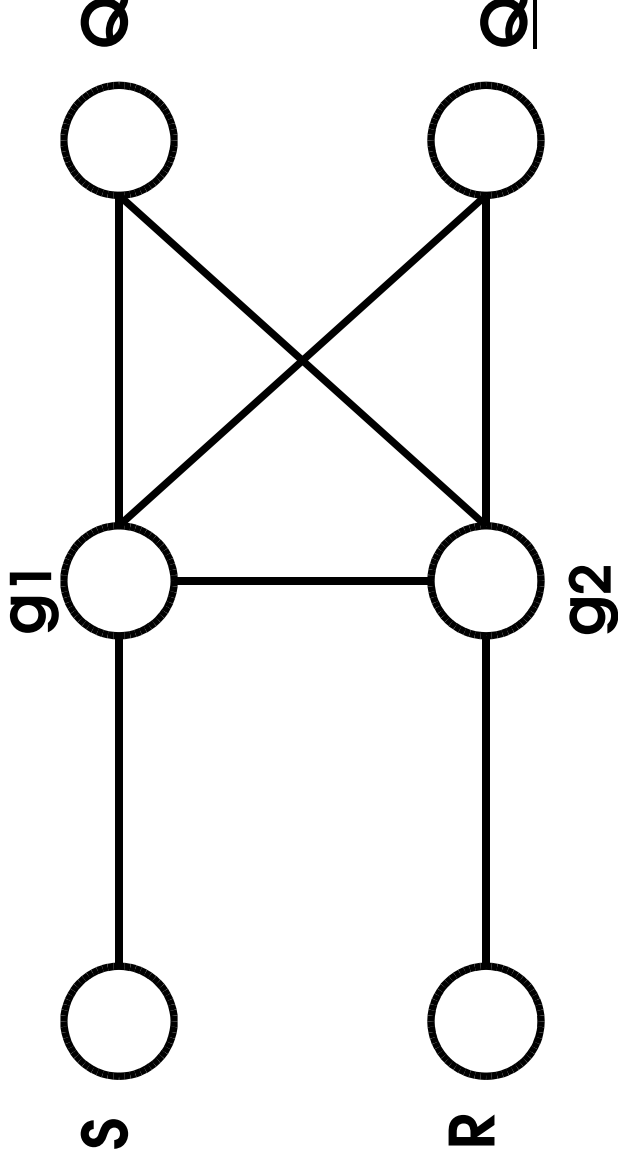


## ■ Bipartiter Graph

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




# Schaltungen als Graphen 3



■ Cliques-Modell  Zelle

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

# 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

# Grundlagen Timing-Analyse

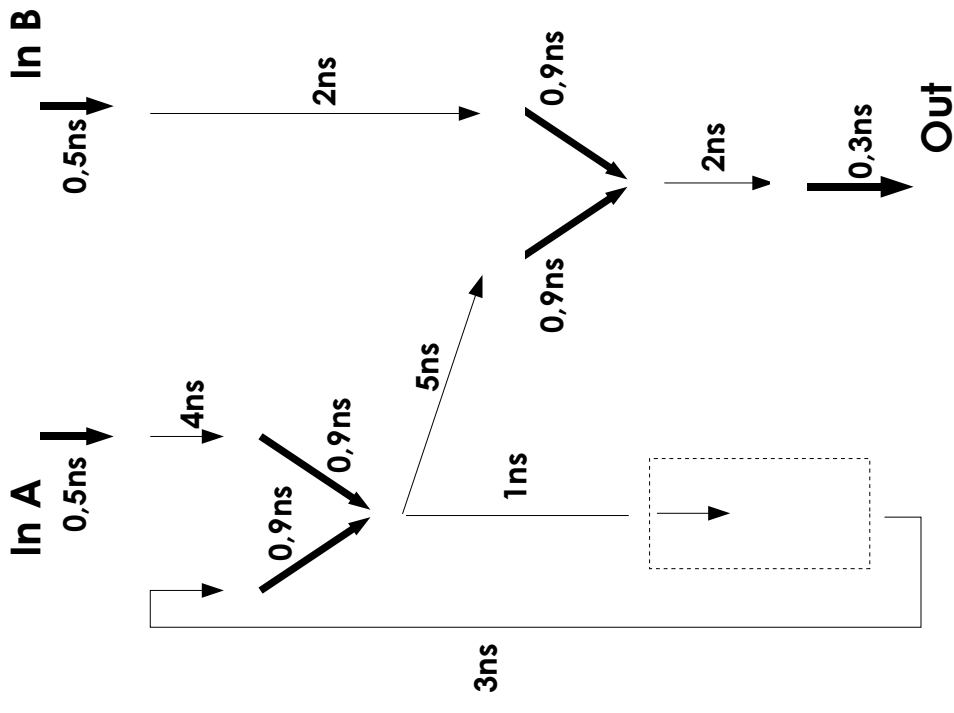
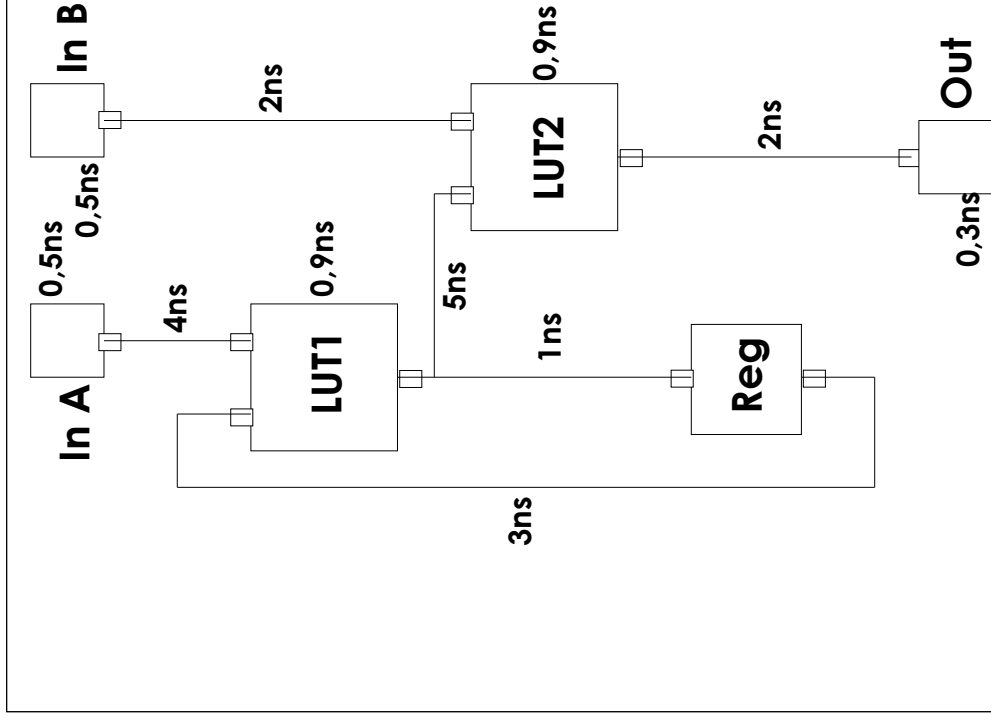
## ■ Wozu?

- Analysiere fertige Layouts
- Analysiere einzelne Verbindungen während Layouterzeugung
  - ◆ Erkenne kritische Verbindungen
  - ◆ Behandle diese mit Vorrang

## ■ Worauf?

- Schaltungselemente
  - ◆ Gatter, Wertetabellen (*LUT*), Register, I/O-Blöcke, ..
  - ◆ Bleiben konstant, exakte Verzögerungen bekannt
- Netze
  - ◆ Nur nach Layouterzeugung bekannt, vorher schätzen

# Modellierung



## ■ Auf „5-partitem“ Graph

- Externe Ein-/Ausgänge, Ein-/Ausgangs-Ports

# Berechnung Ankunftszeit

- Ankunftszeit (Arrival) an Knoten  $v$ :

$$T_a(v) = \underset{(u,v) \in E}{\text{Max}} (T_a(u) + w(u, v))$$

- Idee: BFS oder zyklenfreier LP
  - Beginne mit  $T_a(v) = 0$  für Knoten  $v$  die sind:
    - ◆ Externer Eingang, Registerausgang
  - Bearbeite Knoten mit bearbeiteten Vorgängern
  - Späteste Gesamtankunftszeit  $D_{\max} = \text{Taktper.}$
  - ◆ An externem Ausgang oder Registereingang
    - ◆ Im Beispiel 13,6ns

# Spätestmögliche Ankunftszeit

- **Wie unwichtig sind unkritische Netze?**
  - Idee: Verschiebbare Elemente bei Kompakt.
  - Hier auf Zeitintervalle anwenden (*slack*)
  - „Wievil langsamer kann ein Netz werden, ohne dass die gesamte Schaltung leidet?“
  
- **Berechnung**
  - Mittels spätestmöglicher Ankunftszeit
    - ◆ Required time  $T_r(u)$  an Knoten  $u$
    - ◆ Spätestmöglicher Ankunftszeitpunkt von Signalen
      - ◆ Sonst Verlangsamung der ganzen Schaltung
    - ◆ Analog Kompaktierungsbeispiel
      - ◆ Rechteste Position ohne Breitenvergrößerung

# Berechnung $T_r(u)$ & $\text{slack}(u, v)$

- **Beginne mit  $T_r(u) = D_{\max}$  für Knoten:**
  - **Externer Ausgang, Registereingang**
- **Nun BFS/LP rückwärts**
- **Bearbeite Knoten**
  - **Nur mit komplett bearbeiteten Vorgängern**
    - ◆ **Rückwärts: Vorgänger hier sind sonst Nachfolger!**

$$T_r(u) = \underset{(u, v) \in E}{\text{Min}} (T_r(v) - w(u, v))$$

- **Slack einer Verbindung von  $u$  nach  $v$**

$$\text{slack}(u, v) = T_r(v) - T_a(u) - w(u, v)$$

- **Beachte: Auf kritischem Pfad slack = 0**

# Beispiel s27.critical

```
Node: 4 INPAD_SOURCE Block #2 (s27_in_3_)
T_arr: 0 T_req: -3.88578e-16 Tdel: 5e-10

Node: 5 INPAD_OPIN Block #2 (s27_in_3_)
Pin: 0
T_arr: 5e-10 T_req: 5e-10 Tdel: 5e-09
Net to next node: #2 (s27_in_3_). Pins on net: 5.

Node: 12 CLB_IPIN Block #6 (s27_out)
Pin: 0
T_arr: 5.5e-09 T_req: 5.5e-09 Tdel: 0

Node: 17 SUBBLK_IPIN Block #6 (s27_out)
Pin: 0 Subblock #0
T_arr: 5.5e-09 T_req: 5.5e-09 Tdel: 9e-10

Node: 21 SUBBLK_OPIN Block #6 (s27_out)
Pin: 4 Subblock #0
T_arr: 6.4e-09 T_req: 6.4e-09 Tdel: 0

Node: 16 CLB_OPIN Block #6 (s27_out)
Pin: 4
T_arr: 6.4e-09 T_req: 6.4e-09 Tdel: 1e-09
Net to next node: #5 (s27_out). Pins on net: 2.

Node: 10 OUTPAD_IPIN Block #5 (out:s27_out)
Pin: 0
T_arr: 7.4e-09 T_req: 7.4e-09 Tdel: 3e-10

Node: 11 OUTPAD_SINK Block #5 (out:s27_out)
T_arr: 7.7e-09 T_req: 7.7e-09
```

```
Tnodes on crit. path: 8 Non-global nets on crit. path: 2.
Global nets on crit. path: 0.
Total logic delay: 1.7e-09 (s) Total net delay: 6e-09 (s)
```



# 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**
- **Kleine Übungsaufgabe**
  - Berechne  $T_a$ ,  $T_r$ , Slack
    - ◆ Für das Beispiel auf Folie 46
  
- **Allgemeine Vorbereitung**
  - Buch Kapitel 5.5 - 5.9

# Zusammenfassung

- **Kompaktierung**
- **Berechnung der längsten Pfade**
  - Ohne und mit Zyklen
- **Modellierung von Schaltungen**
  - Graphbasiert
  - Hierarchisch
- **Timing-Analyse**
  - Ankunftszeit
  - Spätestmöglicher Ankunftszeit
  - Slack