

# Algorithmen im Chip-Entwurf 1

## Probleme, Werkzeuge und Graphen

Andreas Koch  
FG Eingebettete Systeme  
und ihre Anwendungen  
TU Darmstadt

# Orga 1 - Material

## ■ Grundlage der Vorlesung

- *Algorithms for VLSI Design Automation*  
Sabih H. Gerez
- In Informatikbibliothek vorhanden

## ■ Wissenschaftliche Arbeiten („Papers“)

## ■ Wissenstiefe

- Kein perfektes Verständnis ...
- ... aber Überblick über das Material
  - ◆ Fragen stellen!

# Orga 2 - Prüfungsmodus

## ■ Idealerweise in Projektarbeit: 4 SWS

- Programmierung, Kolloquien, Vorträge
- Viel Arbeit: 15K-20K LoC in Java
- Hierfür aber max. 18 Plätze (betreuungsintensiv!)

## ■ Für alle anderen: 2 SWS

- Normale vorlesungsbegleitende Prüfung
- Zwei Teilprüfungen
- Je nach Andrang mündlich oder schriftlich

## ■ Aufteilung: Freiwillig 2 SWS, oder ...

- Abgabe von Lösung der ersten Aufgabe
- Beste Abgaben gehen in 4 SWS-Modus ein

# Orga 3 - Teilklausuren

- **Nur im Prüfungsmodus 2 SWS relevant**
- **Geplant in KW 48 und KW 6**
- **Je 45 Minuten, erreichbar je 45 Punkte**
  - **Werden für Endnote addiert**
- **Einbringen der ersten Programmieraufgabe**
  - **War ja für alle**
  - **Mit maximal 10 Punkten, wird aufaddiert**
  - **Kann Note um bis zu einen Wert (1,0) anheben**
  - **Zum Bestehen aber mindestens 27 „echte“ Klausurpunkte erforderlich**

# Orga 4 – Benotung 4 SWS

- Viel Freiheit bei der Realisierung
- Keine starren Bewertungsrichtlinien
  - Analog zu Diplom-Arbeit etc.
- Grundideen
  - Brauchbar kommentierte, brauchbar dokumentierte und **funktionierende** Lösung der Aufgabenstellung: 2,0
    - ◆ Kleinere Schwächen: OK
      - ◆ Einbußen in Lösungsqualität, Rechenzeit, Speicher, ...
  - Aber Luft nach oben (Richtung 1,0), z.B. für
    - ◆ Sehr gute eigene Algorithmen und Datenstrukturen
    - ◆ Umfassende Kommentierung und Dokumentation
    - ◆ Sehr gute Lösungsqualität
    - ◆ Kurze Rechenzeiten
    - ◆ Niedriger Speicherverbrauch

# Orga 5 - Prüfungsleistung

## ■ Benotete Prüfungsleistung

- Beginnend in 4. Semesterwoche (1. Abgabe)
- Gewertet
  - ◆ Programme
    - ◆ Funktion, Code-Qualität, (Dokumentation)
  - ◆ Kolloquien
  - ◆ Vorträge

## ■ Individuelle Prüfung

- Nur in Zweifelsfällen
- Bei nicht nachvollziehbarer Mitarbeit

# Orga 6 - Aufbau

## ■ Integrierte Veranstaltung

- Zu Beginn: Nur Vorlesung (2 pro Woche)
- Dann: praktische Programmierarbeit
  - ◆ In Gruppen
  - ◆ Kolloquien
  - ◆ Vorträge
- Vorlesung nun 1/Woche, am Ende keine mehr

## ■ Kick-Off zu den praktischen Arbeiten

- Anfang KW 43 (= 2. Semesterwoche)
- Vorher Leitfaden lesen!

# Orga 6 – Zeitplan und WWW

## ■ Zeitplan

- Vorlesung
  - ◆ KW 42&43: Di+Fr, KW 44...4: Nur Di 11:40-13:20
  - ◆ Keine mehr in KW 5&6
- Projektarbeit
  - ◆ Abgaben KW 45, 49, 3, 5: Mo 23:59
  - ◆ Vorträge KW 45, 49, 3, 6: Fr 9:50-11:30
  - ◆ Kolloquien KW 45, 49, 3, 5: Do nachmittags

## ■ Web-Seite

- <http://www.esa.informatik.tu-darmstadt.de>  
Unterpunkt „Lehre“
- Material und Ankündigungen



## ■ VLSI Entwurf

- Probleme
- Bereiche
- Tätigkeiten
- ➔ Werkzeuge

## ■ Hierarchie und Abstraktion

## ■ Algorithmische Graphentheorie

- Strukturen
- Verfahren

# VLSI Entwurfsproblem

**„Implementiere eine Spezifikation in Hardware und optimiere dabei ...“**

- Fläche (min.)
- Stromverbrauch (min.)
- Geschwindigkeit (max. oder passend)
- Entwurfszeit (min.)
- Testbarkeit (max.)

**† „Alles auf einmal“ ist zu komplex**

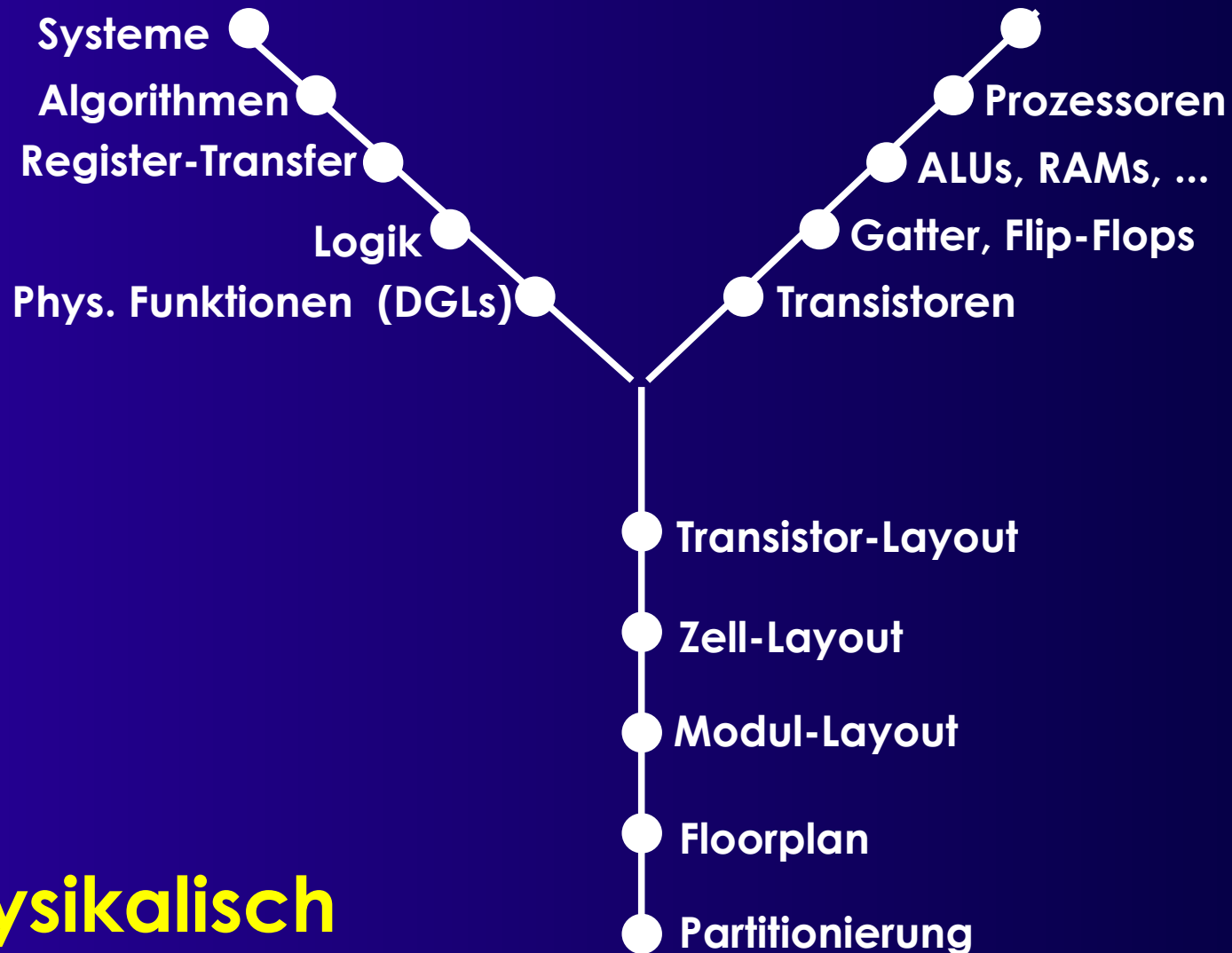
**→ Aufteilen und vereinfachen**

- Qualitätseinbussen

# Entwurfsbereiche - Gajskis "Y"

## Verhalten

## Struktur



## Physikalisch

Probleme, Werkzeuge und Graphen

## ■ Synthese

- Mehr Details durch Anwendung von Regeln

## ■ Verifikation

- Vergleiche Ergebnis mit Spezifikation

## ■ Analyse

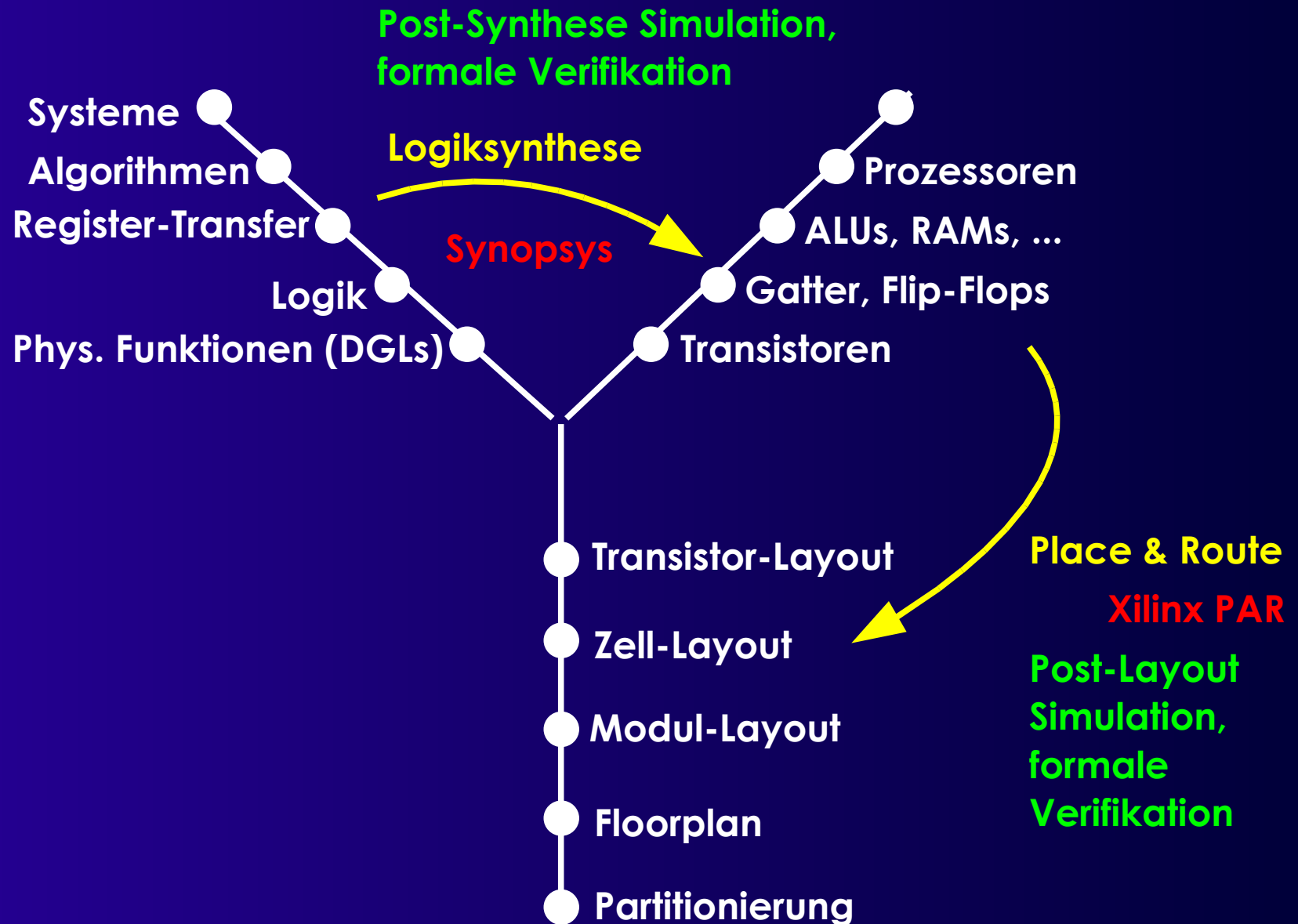
- Untersuche Eigenschaften eines Ergebnisses

## ■ Optimierung

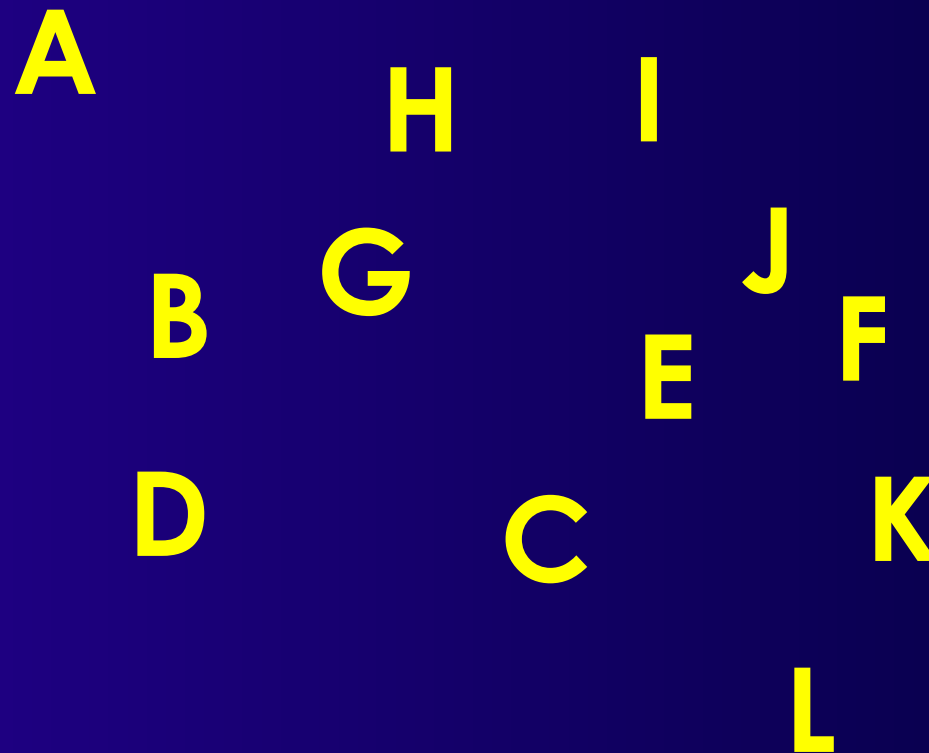
- Verbessere ein Ergebnis

## ■ Datenverwaltung

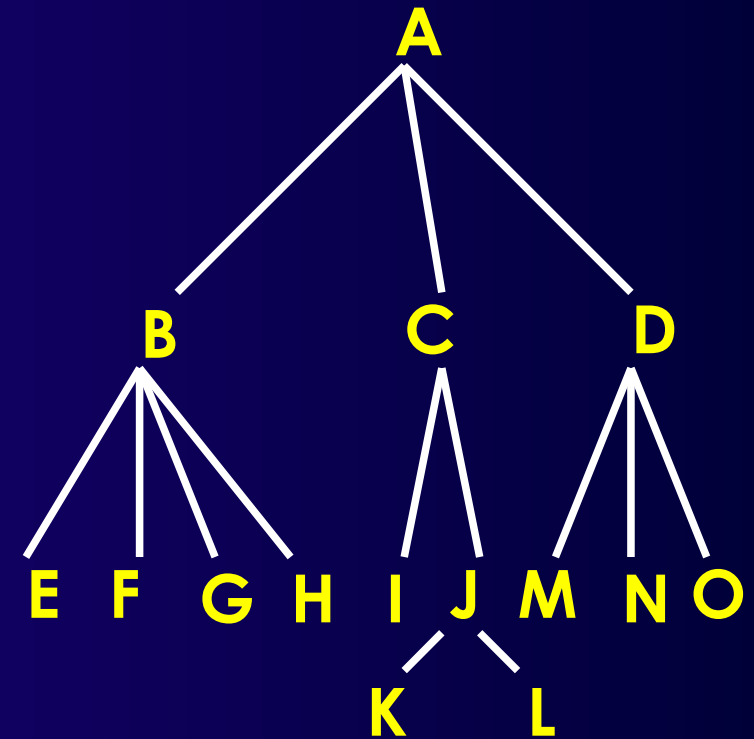
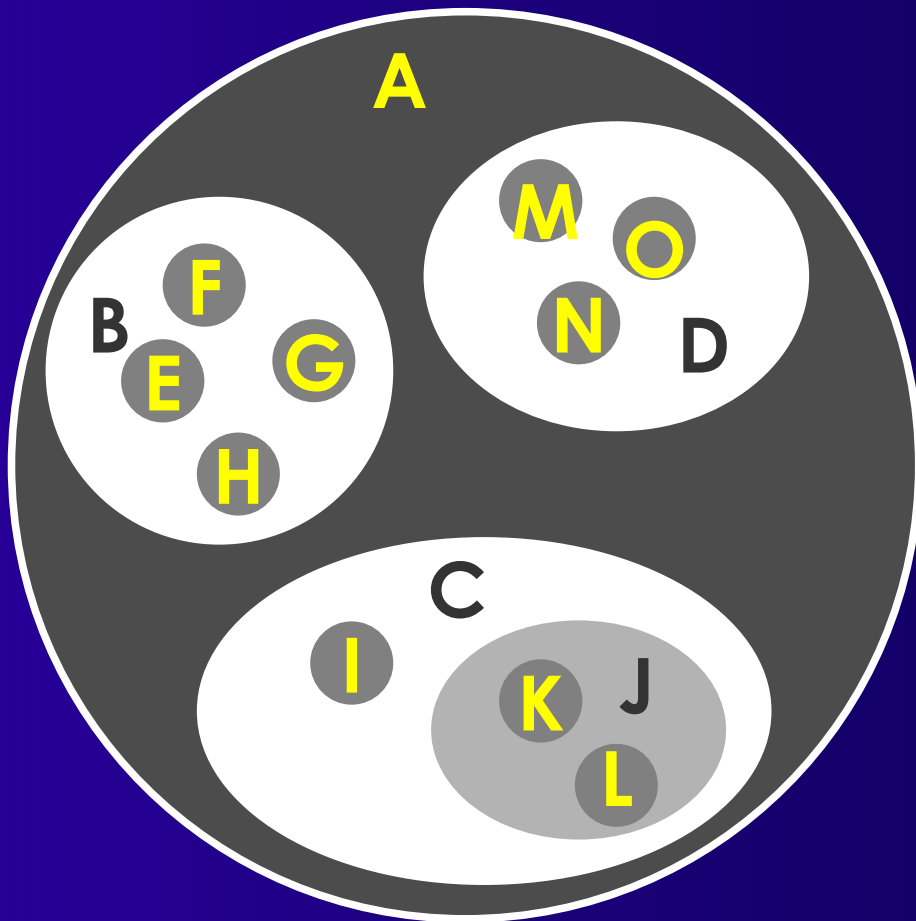
# Werkzeuge



# Strukturierung

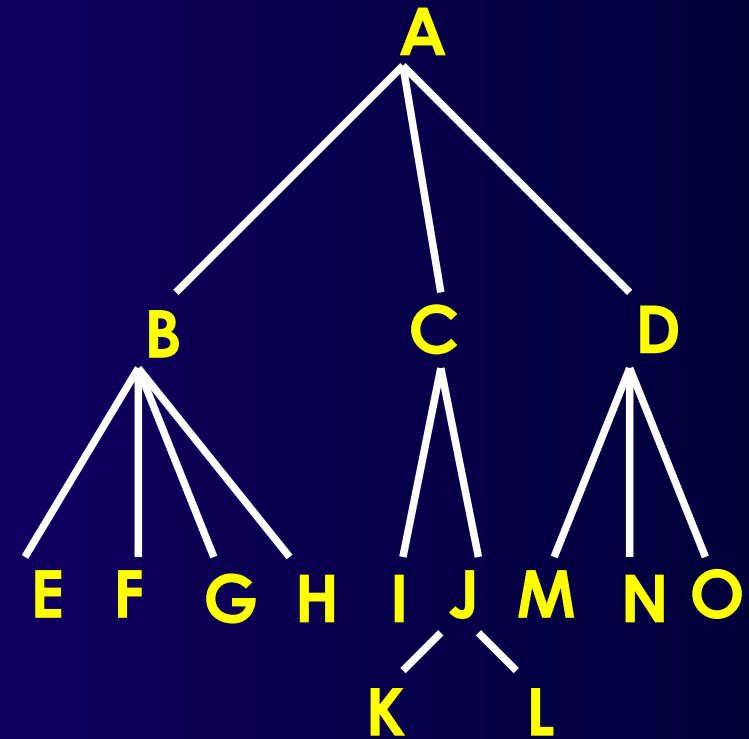
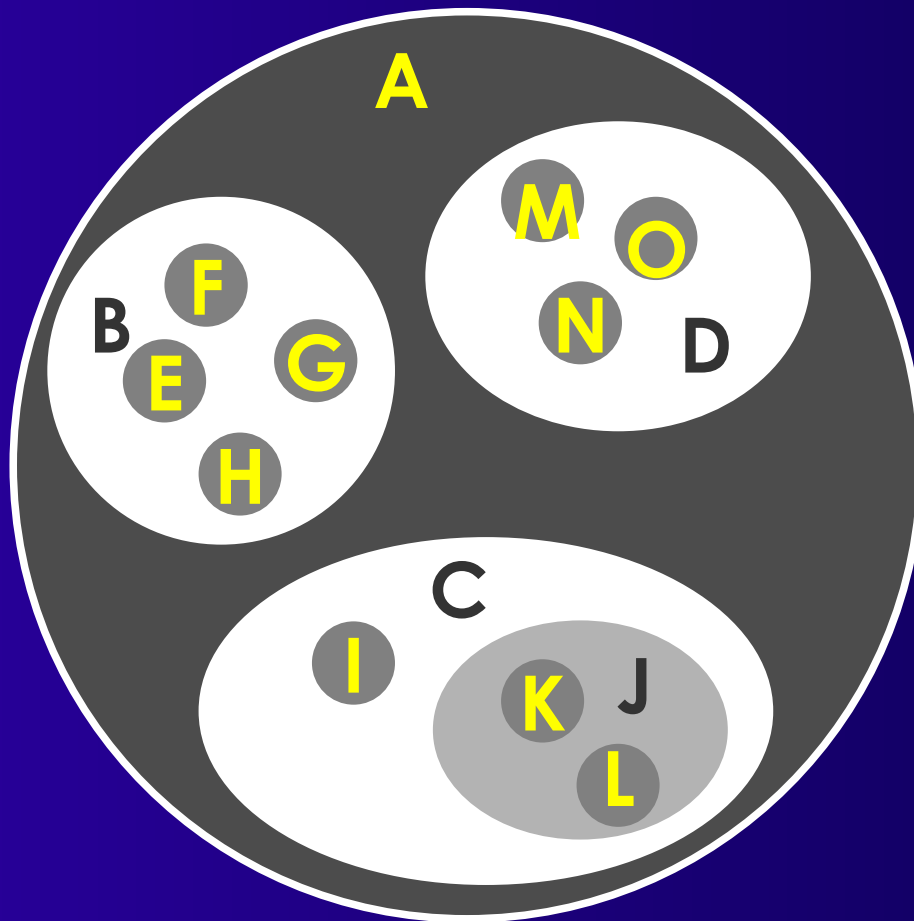


# Hierarchie



# Abstraktion

Abstraktionsebene

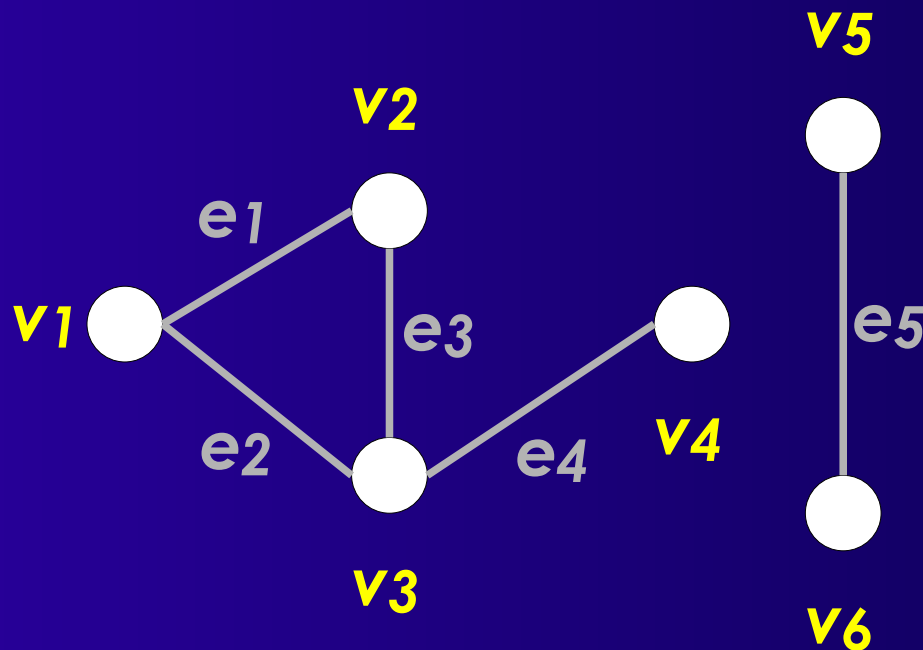




# Graphentheorie

## ■ Graph $G(V, E)$

- Eine Menge  $V$  von Knoten (vertex)
- Eine Menge  $E$  von Kanten (edge)
  - ◆ Kante  $e = \{v_1, v_2\}$  verbindet Knoten  $v_1$  und  $v_2$



$$G = (V, E)$$

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$

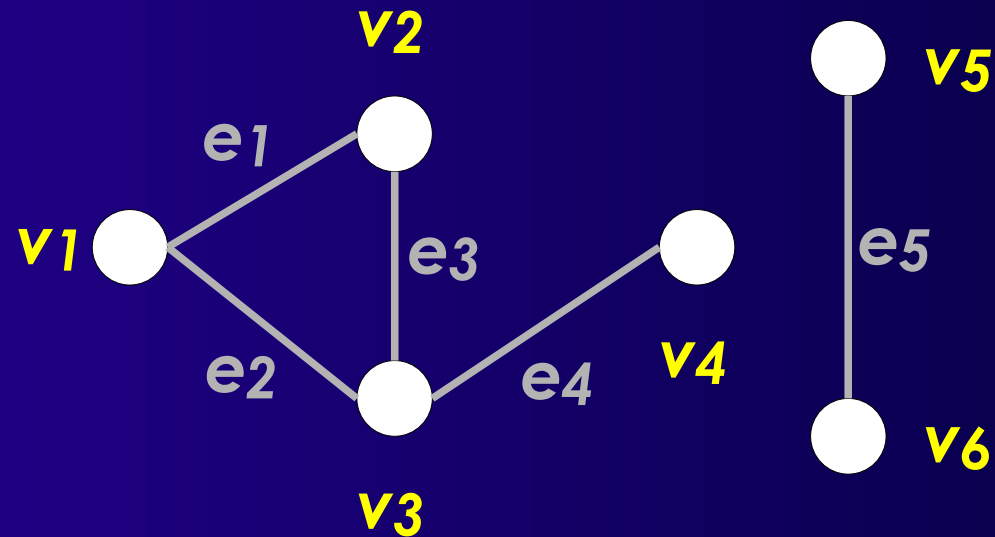
$$E = \{e_1, e_2, e_3, e_4, e_5\}$$

mit

$$e_1 = \{v_1, v_2\}$$

$$e_2 = \{v_1, v_3\}, \dots$$

# Inzidenz, Adjazenz und Grad



■  $e = \{u, v\} \in E$

●  $e$  ist **inzident**  $u$

●  $e$  ist **inzident**  $v$

●  $u$  ist **adjazent**  $v$

■ **Grad**  $g(v) = |\{e \in E \mid v \in e\}|$

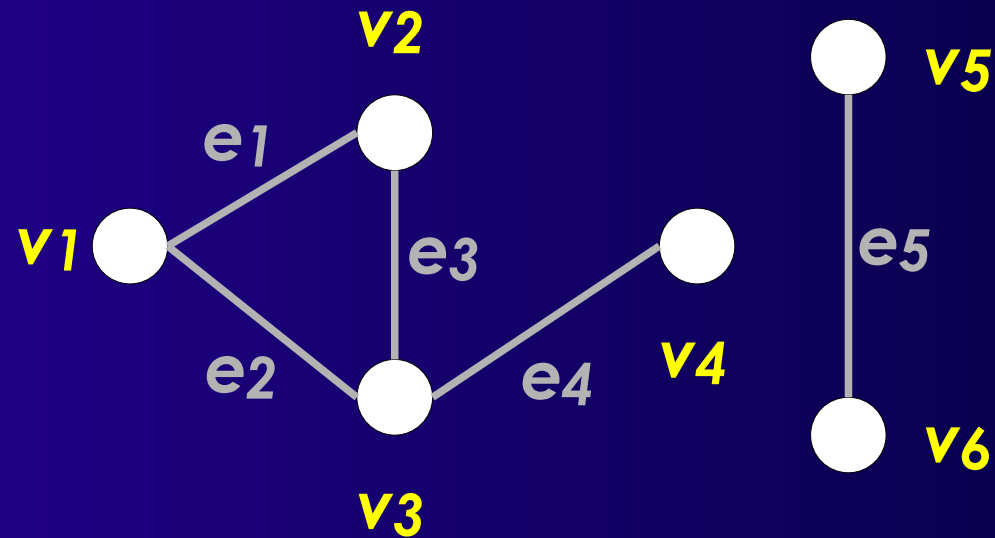
*incident*

*incident*

*adjacent*

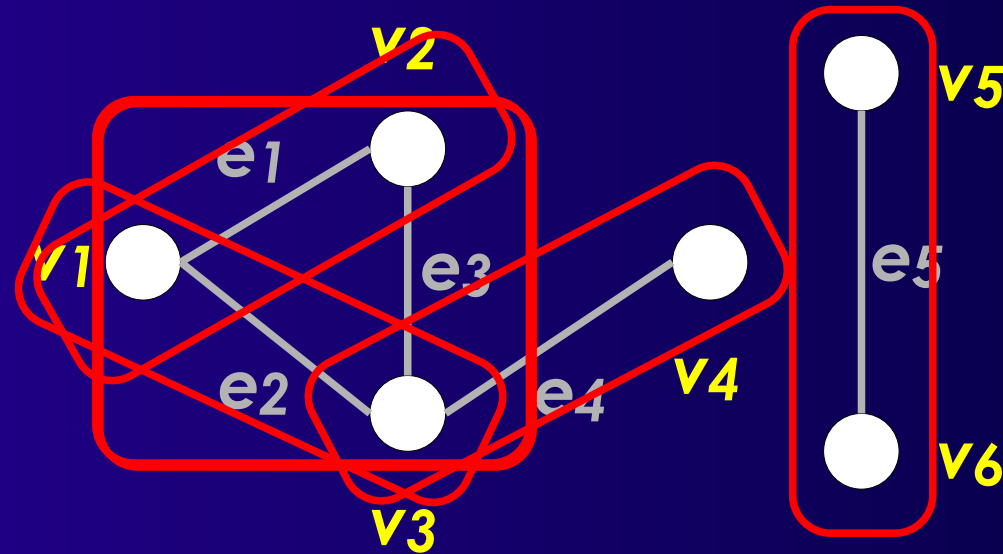
*degree*

# Subgraphen



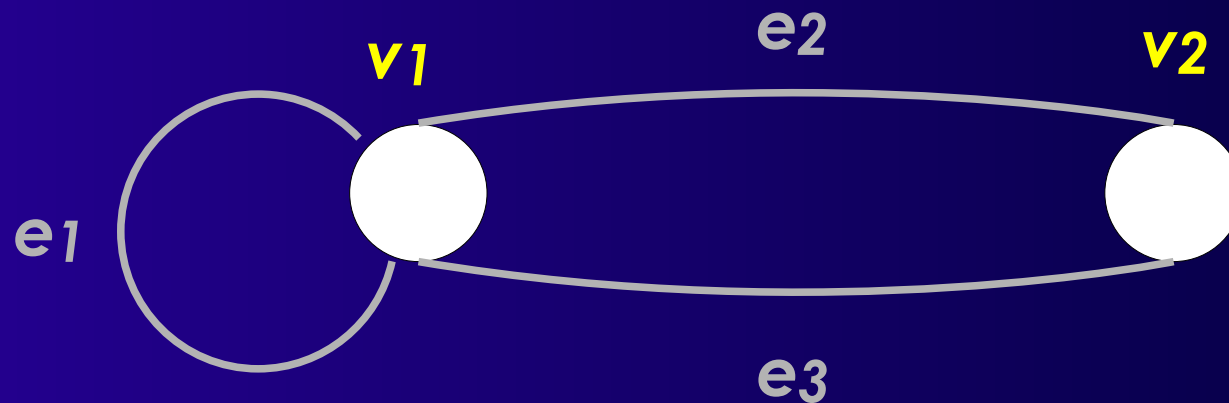
- Subgraph durch Entfernen von Knoten
  - Entferne  $v \in V$
- Entferne Kanten inzident zu  $v$

# Vollständigkeit und Cliques



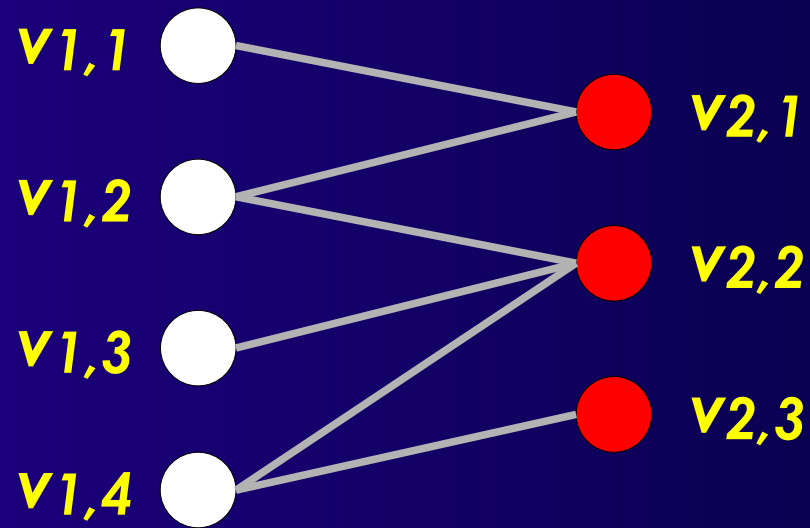
- **Komplett untereinander verbundene Knoten bilden vollständigen Graph (complete graph)**
- **Maximal ausgedehnte vollständige Graphen bilden Cliques**

# Schlingen, parallele Kanten



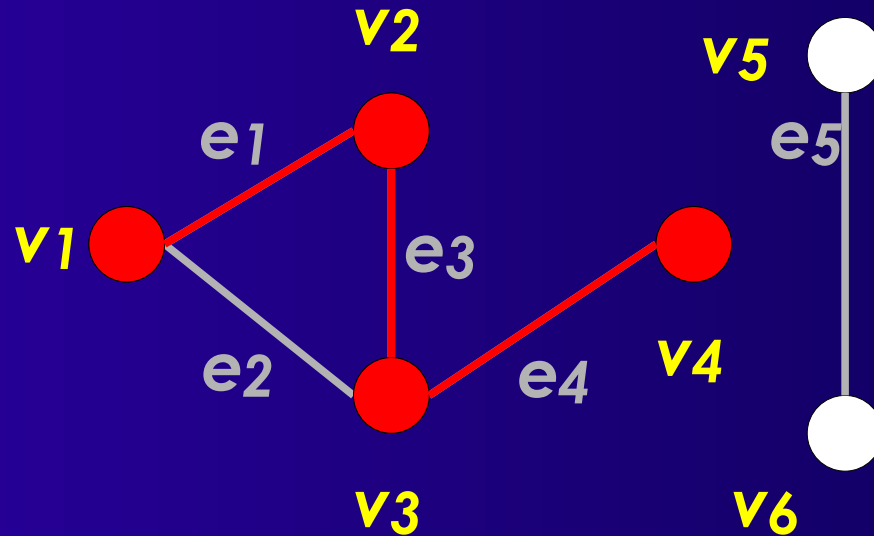
- $e_1$  Schlinge (selfloop)
- $e_2, e_3$  parallele Kanten
- einfache Graphen: weder noch (simple)
- Multigraphen: parallele Kanten OK

# Bipartite Graphen



- Kanten nur zwischen Knoten aus nichtüberlappenden Mengen
- $G = (V_1, V_2, E)$  ist **bipartiter Graph**
  - $V_1 \cap V_2 = \emptyset$
  - $E = \{\{u, w\} \mid u \in V_1 \wedge w \in V_2\}$

# Wege und Zyklen

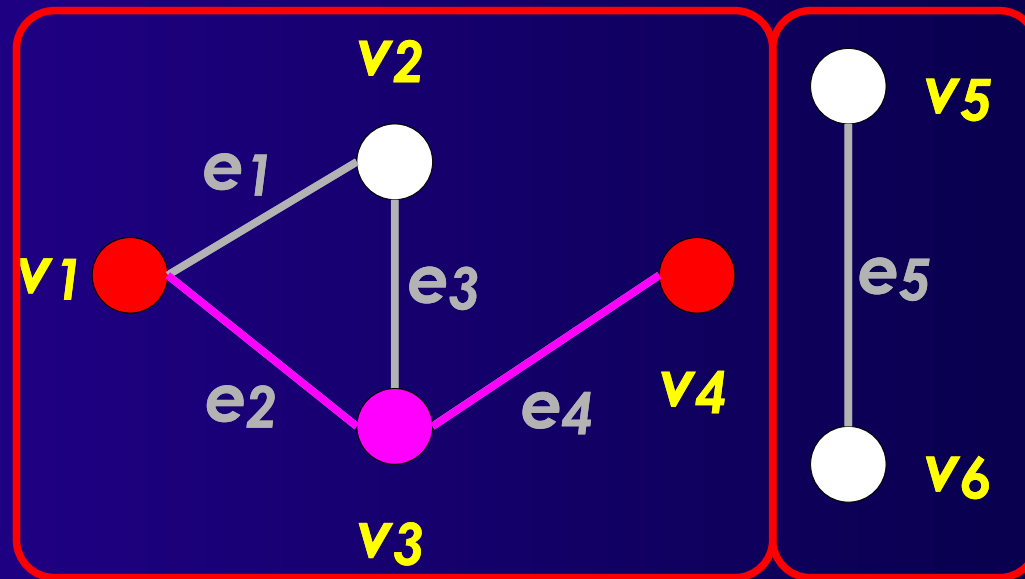


$(v1, e1, v2, e3, v3, e4, v4)$

Länge = 3

- **Weg:** Folge von Knoten und Kanten
  - Beginnend und endend mit Knoten
- **Länge:** Anzahl der Kanten
- **Zyklus:** Anfang = Ende

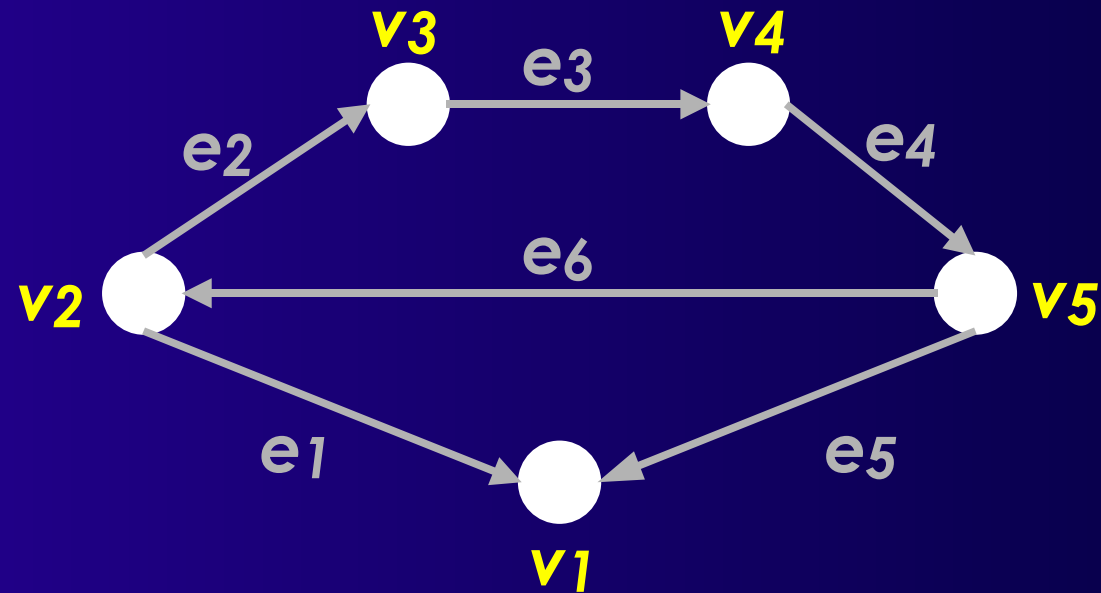
# Zusammenhang



- **u hängt mit v zusammen**
  - Es gibt einen beide verbindenden Weg
- **Zusammenhängender Graph**
  - Alle Knoten hängen zusammen.
- **Zusammenhängende Komponente**
  - Maximale zusammenhängende Subgraphen

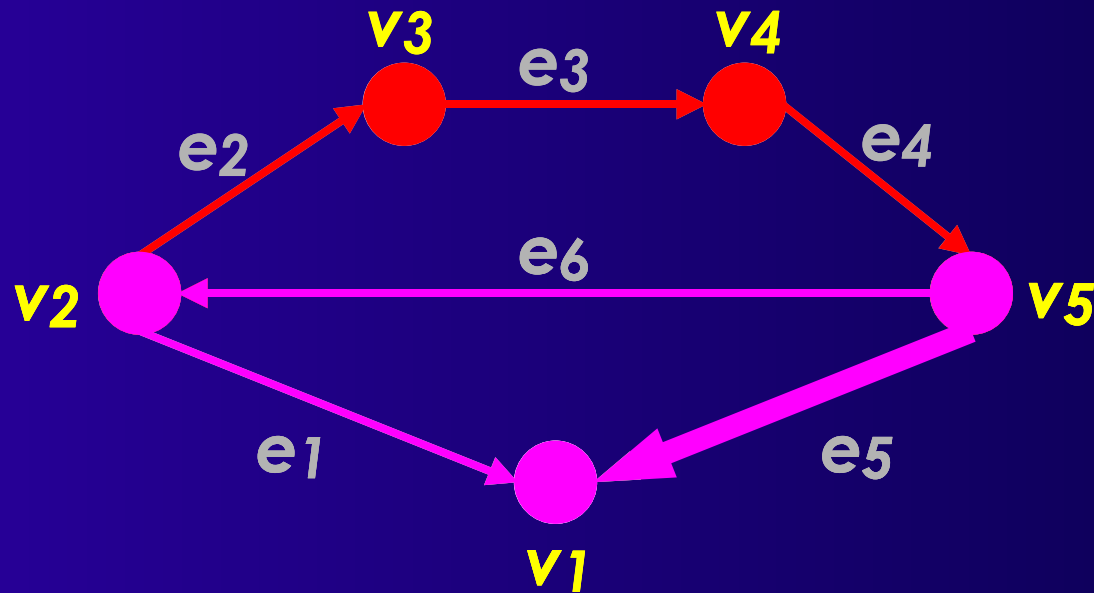


# Gerichtete Graphen



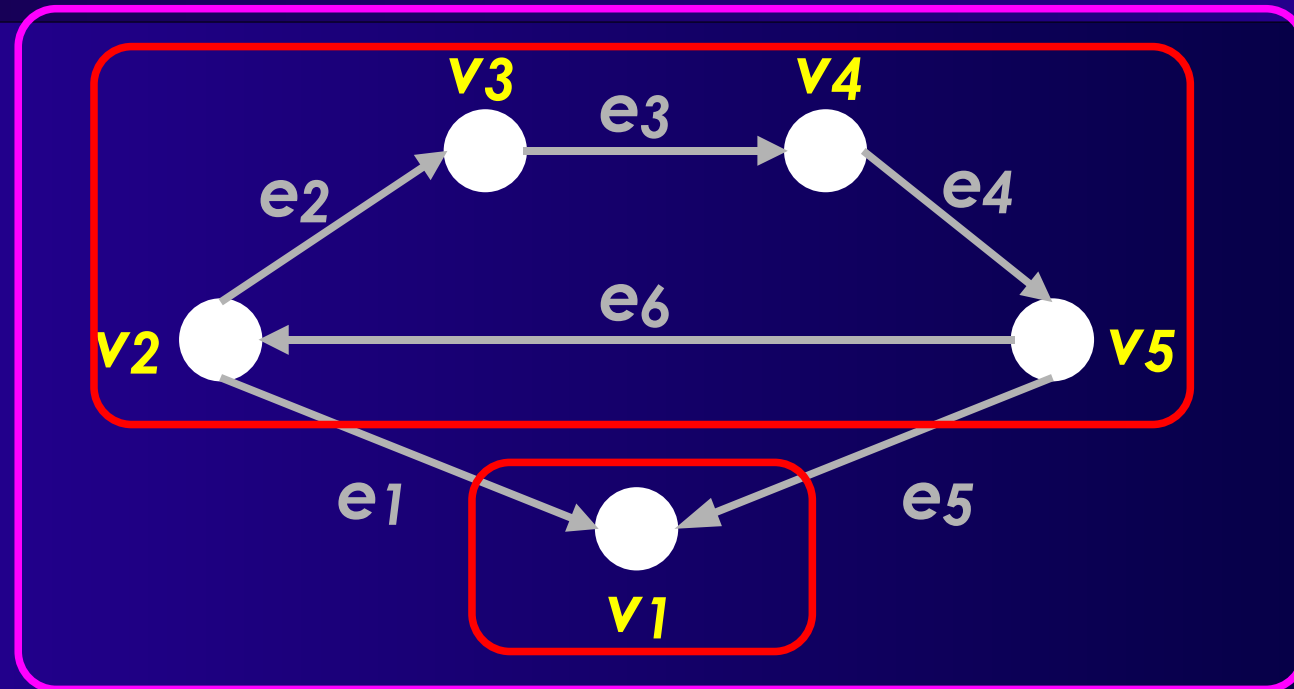
- $G(V, E)$  mit  $e = (u, v) \wedge u, v \in E$ 
  - $e$  inzident von  $u$  (ausgehend)
  - $e$  inzident nach  $v$  (eingehend)
- **Außengrad:** Anzahl ausgehender Kanten
- **Innengrad:** Anzahl eingehender Kanten

# Wege und Zyklen



- **Gerichteter Weg**
- **Gerichteter Zyklus**
- **Weg und Zyklus gelten auch noch!**

# Zusammenhang



## ■ Starker Zusammenhang

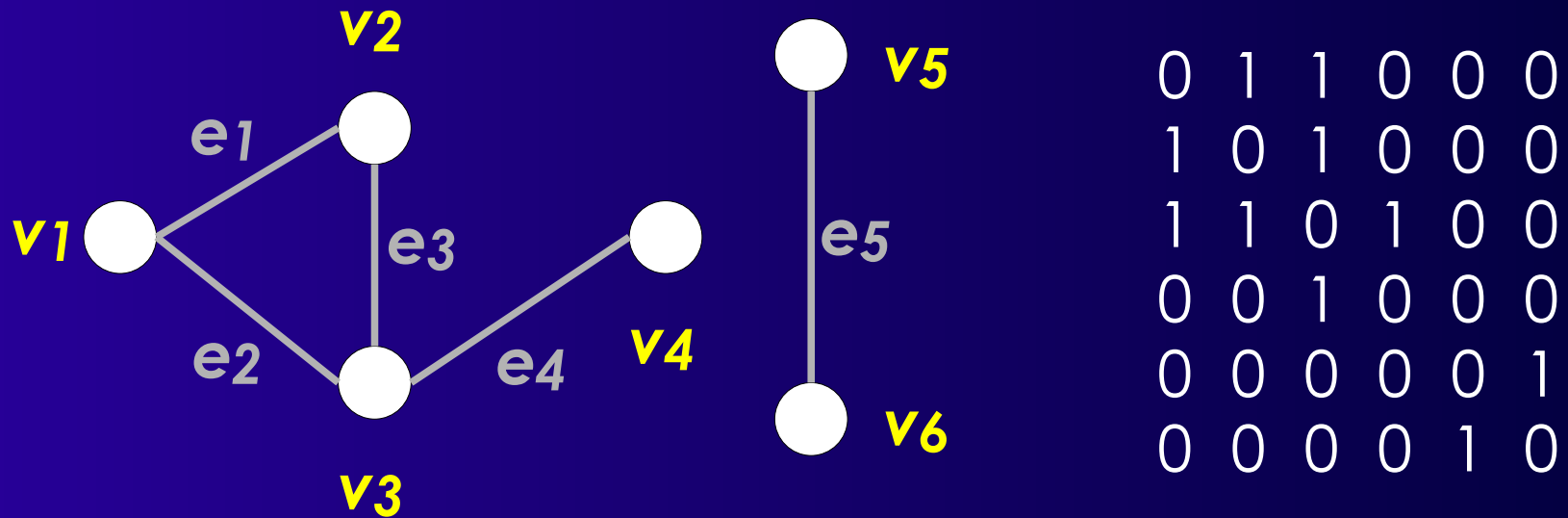
- Gerichteter Weg von  $u$  nach  $v$  & von  $v$  nach  $u$

## ■ Stark zusammenhängende Komponente

- Alle enthaltenen Knoten hängen stark zusam.

## ■ Schwacher Zusammenhang: Weg

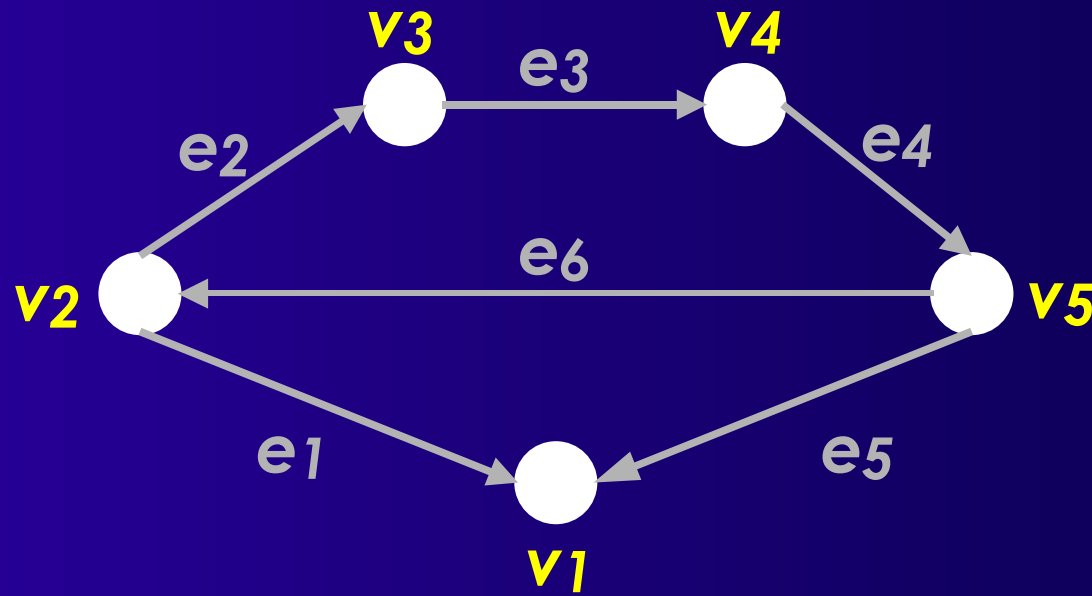
# Datenstrukturen für Graphen



## ■ Adjazenzmatrix $A_G$ von $G(V,E)$

- $n \times n$  Matrix mit  $n = |V|$
- $A_{ij} = 1$  falls  $\{v_i, v_j\} \in E$ , sonst = 0
- Symmetrische Matrix

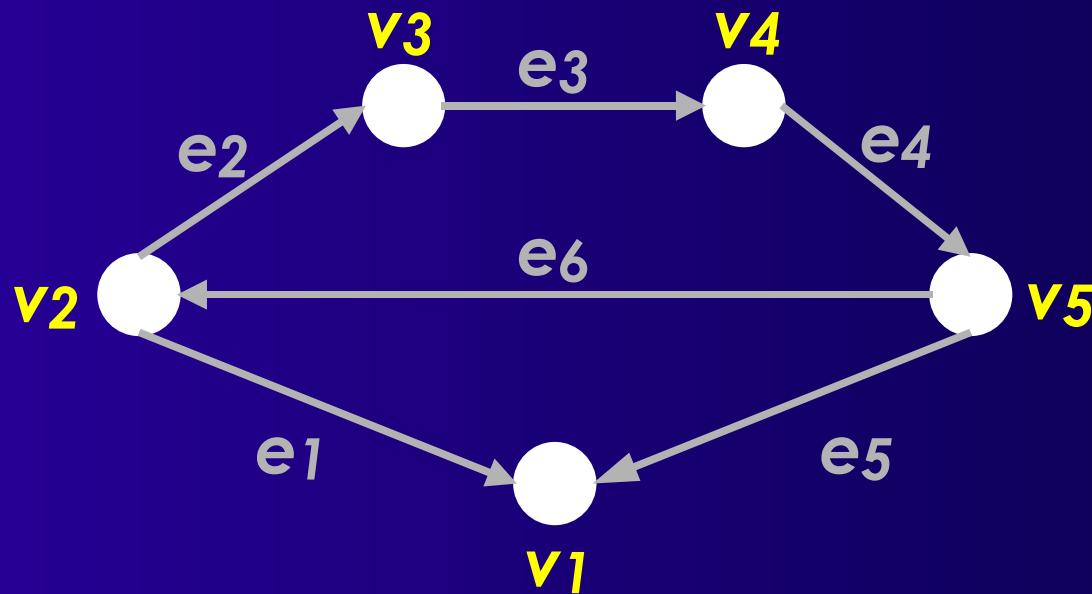
# AG für gerichtete Graphen



0	0	0	0	0
1	0	1	0	0
0	0	0	1	0
0	0	0	0	1
1	1	0	0	0

- Matrix nicht mehr symmetrisch

# Operationen auf AG-Matrizen



0	0	0	0	0
1	0	1	0	0
0	0	0	1	0
0	0	0	0	1
1	1	0	0	0

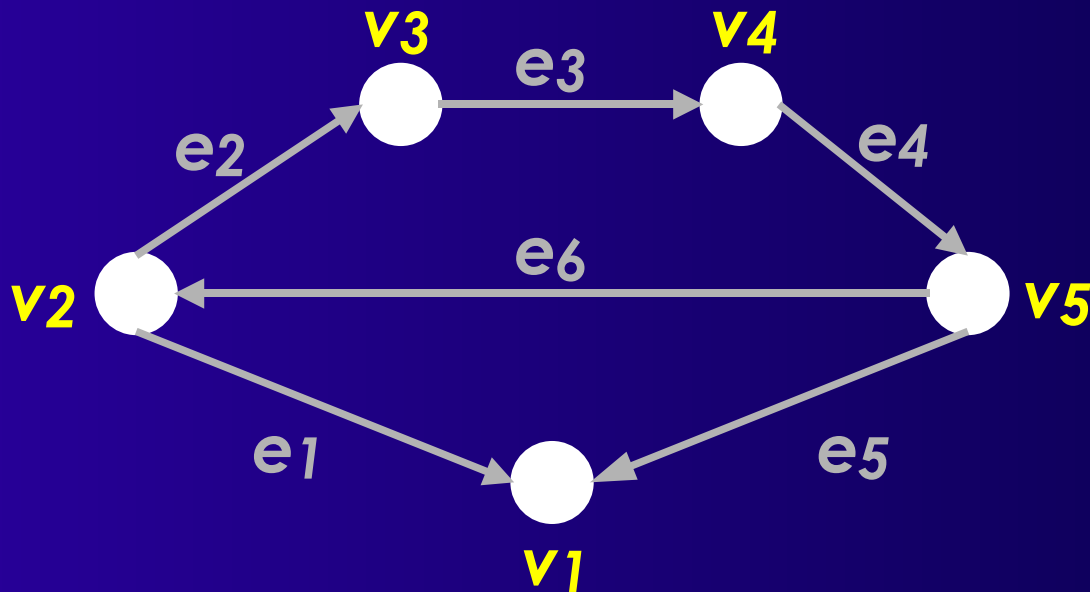
## ■ Test, ob $(v_i, v_j) \in E$

- Nachsehen in  $A_{ij}$ :  $O(1)$

## ■ Welche $v$ sind direkt mit $u_i$ verbunden?

- Zeile  $i$  durchgehen:  $O(n)$
- Ineffizient bei vielen Nullen

# Adjazenzlisten

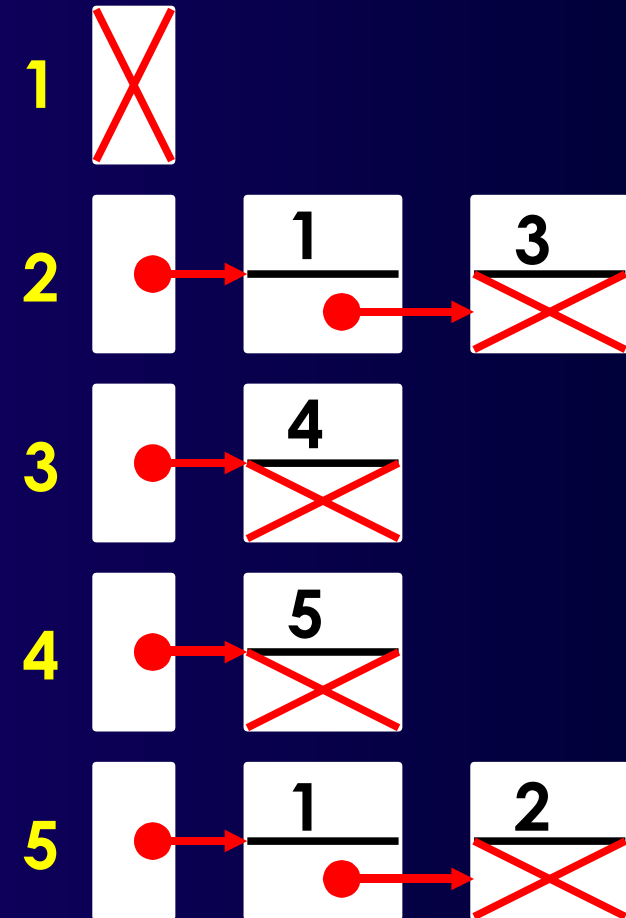


## ■ Array aus Listen

- Knotennummer ist Index

## ■ Listenelemente

- Index des Zielknotens
- Verkettung



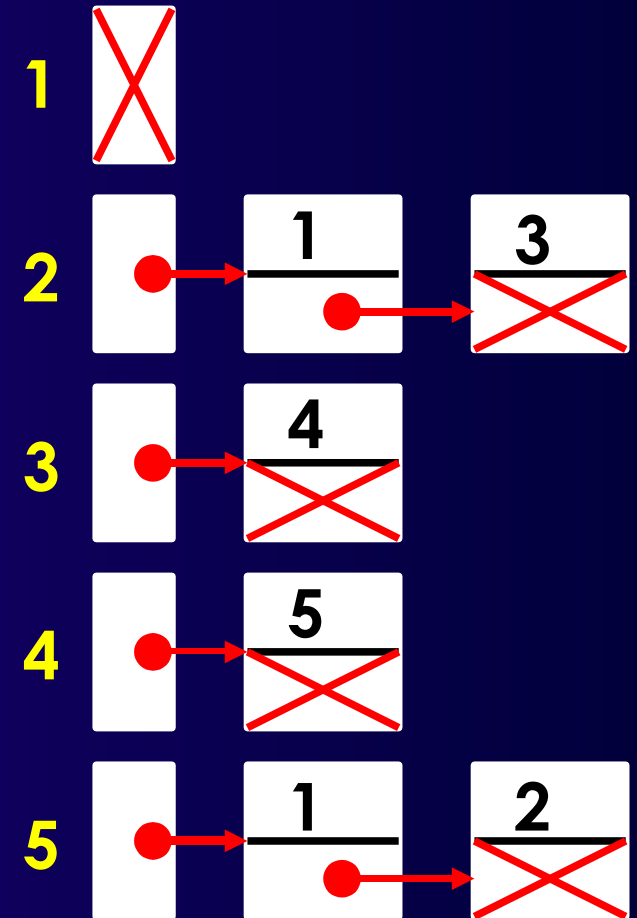
# Operationen auf Adjazenzlisten

## ■ Test, ob $(v_i, v_j) \in E$

- durchschnittlicher Außengrad:  $k(G)$
- $O(k)$
- Unabhängig von  $n$

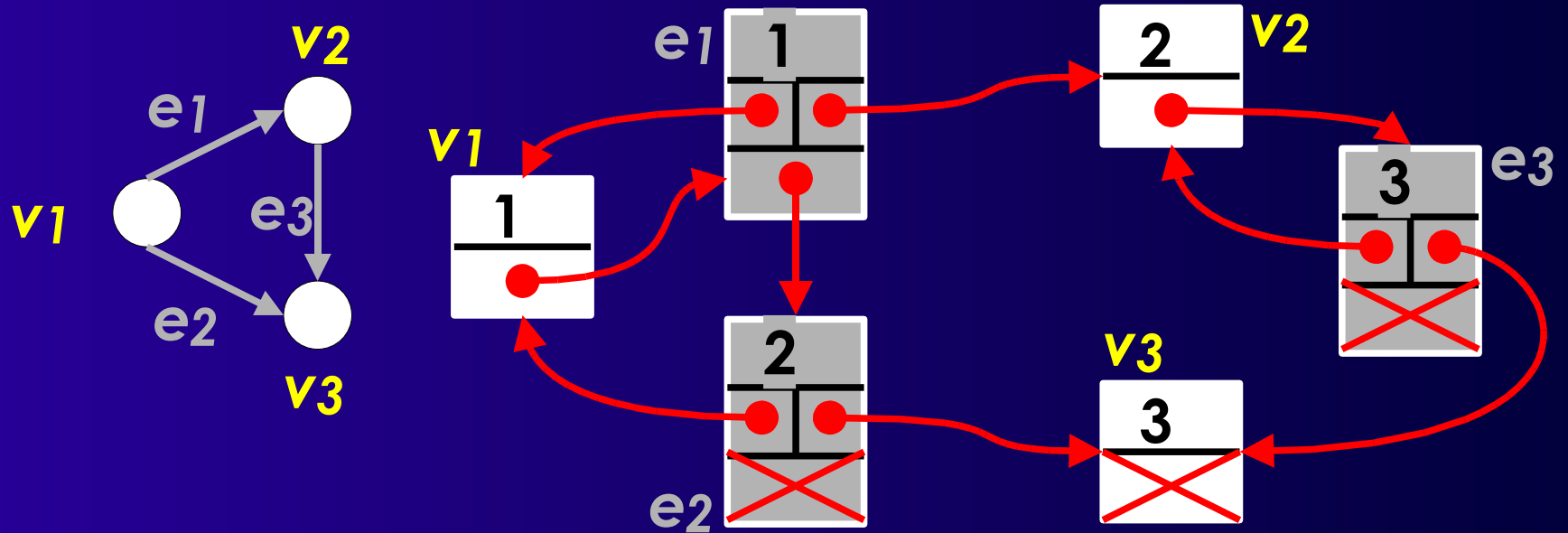
## ■ Welche $v$ sind direkt mit $u_i$ verbunden?

- $O(k)$





# Explizite Knoten und Kanten



## ■ Zugriff auf Knoten und Kanten

## ■ Z.B. Gewichtung von

- Knoten
- Kanten

vertex_index	3
outgoing_edges	

edge_index	3
from, to	
next	

# Komplexitätstheorie

- **O und  $\Theta$  Notation**
- **Siehe Grundstudium!**
- **Wichtige Ordnungen**
  - Exponentiell, z.B.  $2^n$ .
  - Polynomial, z.B.  $n^3$ .
  - Quadratisch, z.B.  $n^2$ .
  - Logarithmisch, z.B.  $n \log n$ .
  - Linear, z.B.  $n$ .
  - Sublinear, z.B.  $1$ .

# Graphen durchlaufen

## ■ Aufgabe

- Besuche alle  $V$  und  $E$  von  $G(V,E)$
- Jedes Element genau einmal!

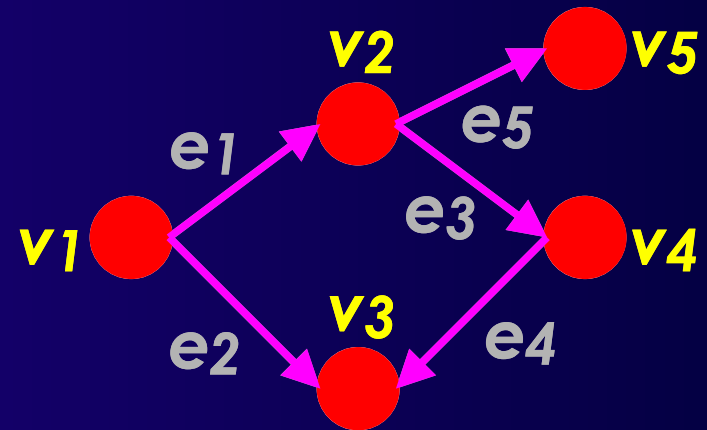
## ■ Unterschiedliche Reihenfolgen möglich

## ■ Weit verbreitet

- Tiefensuche
  - ◆ Suche von Ursprungsknoten entfernen
- Breitensuche
  - ◆ Erstmal angrenzende Knoten bearbeiten

# Tiefensuche (DFS) - 1

```
dfs(vertex v) {  
  v.mark := 0;  
  v.process();  
  foreach (v,u) ∈ E {  
    (v,u).process();  
    if (u.mark) dfs(u);  
  }  
}  
  
main() {  
  foreach v ∈ V  
    v.mark := 1;  
  foreach v ∈ V  
    if (v.mark) dfs(v)  
}
```



```
dfs(v1)  
  (v1, v2)  
    dfs(v2)  
      (v2, v4)  
        dfs(v4)  
          (v4, v3)  
            dfs(v3)  
          (v2, v5)  
            dfs(v5)  
        (v1, v3)
```

# Tiefensuche (DFS) - 2

## ■ Komplexität für DFS auf $G(V,E)$

- Jeder Knoten einmal besucht
- Jede Kante einmal besucht

→  $O(|V|+|E|)$

## ■ Anwendungsbeispiele

- Systematischer Graphdurchlauf
- Finden der von einem Startknoten aus erreichbaren Knoten
  - ◆ Ersetze Schleife in main() durch einfachen Aufruf

# Breitensuche (BFS) - 1

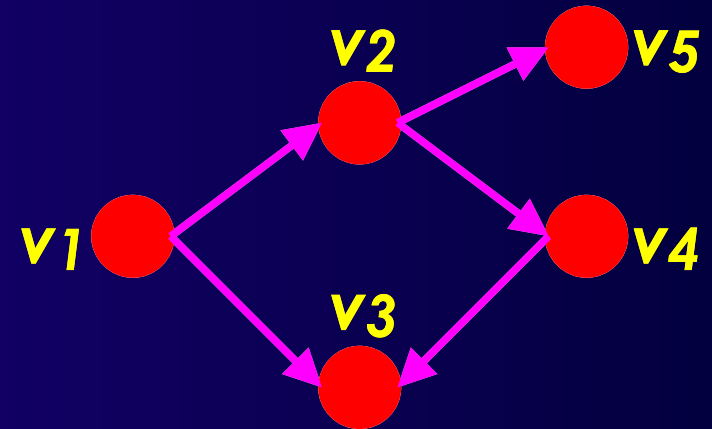
```
bfs(vertex v) {
  FIFO Q = ();
  vertex u, w;

  Q.shift_in(v);
  do {
    w := Q.shift_out();
    w.process();
    foreach (w,u) ∈ E do {
      if (u.mark) {
        u.mark := 0;
        Q.shift_in(u);
      }
    }
  } while (Q ≠ ())
}
```

```
main() {
  foreach v ∈ V do v.mark := 1;
  foreach v ∈ V do
    if (v.mark) {
      v.mark := 0;
      bfs(v);
    }
}
```

# Breitensuche (BFS) - 2

```
bfs(vertex v) {  
  FIFO Q = ();  
  vertex u, w;  
  
  Q.shift_in(v);  
  do {  
    w := Q.shift_out();  
    w.process();  
    foreach (w,u) ∈ E do {  
      if (u.mark) {  
        u.mark := 0;  
        Q.shift_in(u);  
      }  
    }  
  } while (Q ≠ ())  
}
```



(v1)	v1	(v1,v2)
(v2)		(v1,v3)
(v2,v3)	v2	(v2,v4)
(v3,v4)		(v2,v5)
(v3,v4,v5)	v3	
(v4,v5)	v4	(v4,v3)
(v5)	v5	

# Breitensuche (BFS) - 3

## ■ Komplexität für BFS auf $G(V,E)$

- Jeder Knoten einmal besucht
- Jede Kante einmal besucht

→  $O(|V|+|E|)$

## ■ Anwendungsbeispiele

- Systematischer Graphdurchlauf
- Finden der von einem Startknoten aus erreichbaren Knoten
- Besuche Knoten in Reihenfolge der Entfernung (Pfadlänge) vom Startknoten



## ■ Weshalb die die äußeren Schleifen?

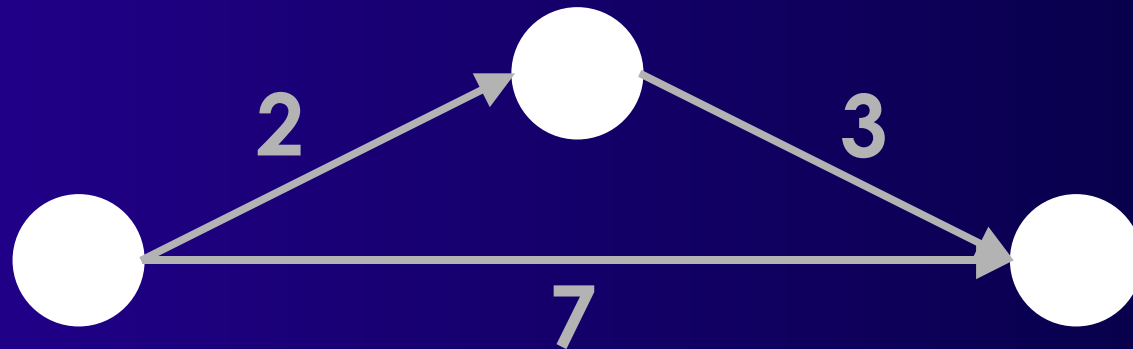
- Jeweils in main()
- Um dfs(v) bzw. bfs(v)

```
main() {  
    foreach v ∈ V  
        v.mark := 1;  
    foreach v ∈ V  
        if (v.mark) dfs(v)  
}
```

```
main() {  
    foreach v ∈ V do v.mark := 1;  
    foreach v ∈ V do  
        if (v.mark) {  
            v.mark := 0;  
            bfs(v);  
        }  
}
```

# Kürzester Pfad

- Bestimme den kürzesten Pfad vom Startknoten zu den anderen Knoten
- Bei ungewichteten Graphen z.B. mit BFS
  - Erweitert um Verwaltung der Pfade
- ✗ Nicht bei gewichteten Graphen!
  - Niedrige Anzahl von Kanten nicht immer kürzester (leichtester) Weg



# Kürzester Pfad nach Dijkstra -1

```
dijkstra(set<vertex> V, vertex vs, vertex vt) {
```

```
  set<vertex> T; vertex u, v;
```

```
  V := V \ {vs}; T := {vs};
```

```
  vs.dist := 0;
```

```
  foreach u ∈ V do
```

```
    if ((vs, u) ∈ E)
```

```
      then u.dist := (vs, u).weight;
```

```
    else u.dist := +∞;
```

```
  while (vt ∉ T) do {
```

```
    u := V.findmin(dist);
```

```
    T := T ∪ {u};
```

```
    V := V \ {u};
```

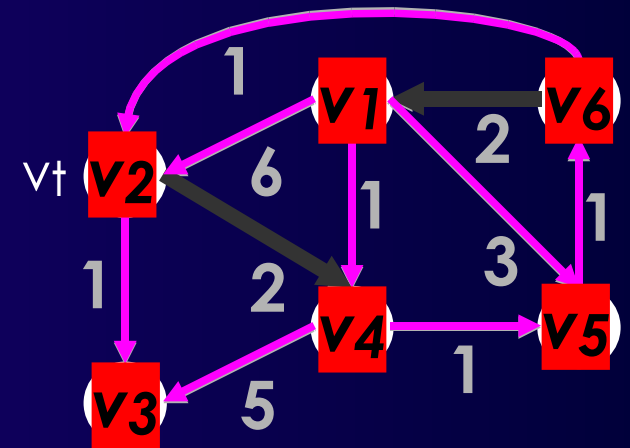
```
    foreach (u, v) ∈ E do
```

```
      if (v.dist > u.dist + (u,v).weight)
```

```
        v.dist := u.dist + (u,v).weight;
```

```
  }
```

```
}
```



T=	v <sub>i</sub> .dist, i=1	2	3	4	5	6
{v <sub>1</sub> }	0	6	∞	1	3	∞
{v <sub>1</sub> , v <sub>4</sub> }		6	6		2	∞
{v <sub>1</sub> , v <sub>4</sub> , v <sub>5</sub> }		6	6			3
{v <sub>1</sub> , v <sub>4</sub> , v <sub>5</sub> , v <sub>6</sub> }		4	6			
{v <sub>1</sub> , v <sub>4</sub> , v <sub>5</sub> , v <sub>6</sub> , v <sub>2</sub> }			5			
{v <sub>1</sub> , v <sub>4</sub> , v <sub>5</sub> , v <sub>6</sub> , v <sub>2</sub> , v <sub>3</sub> }						

# Kürzester Pfad nach Dijkstra -2

## ■ Komplexität

- while ( $v \notin T$ ):  $|V|$ -mal durchlaufen
  - ◆  $V.\text{findmin}(\text{dist})$ :  $O(|V|)$  je Suche
    - ↳  $O(|V|^2)$
- foreach  $(u,v) \in E$ :  $|E|$ -mal *insgesamt*
  - ◆ Einfacher Graph hat max.  $|V|^2$  Kanten
    - ↳  $O(|V|^2)$
- Gesamtaufwand  $O(|V|^2 + |V|^2) = O(|V|^2)$

# Nächste Veranstaltung

- **Vorlesung am Freitag**
- **Vorbereitungstipps**
  - Kapitel 6 und 7.1 lesen
  - Ggf. Kapitel 4 (Komplexität) wiederholen

# Zusammenfassung

## ■ VLSI

- Entwurfsbereiche
- Tätigkeiten
- Werkzeuge

## ■ Hierarchie und Abstraktion

## ■ Graphentheorie

- Konzepte und Begriffe
- Datenstrukturen
- Algorithmen: DFS, BFS, SP